Introduction to Parallelism, Grid and ProActive, Distributed Objects and Components (GCM)

Denis Caromel, et al.
http://ProActive.ObjectWeb.org
OASIS Team
INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis, IUF
Version July 2nd 2007

ProActive

a complete solution for Enterprise GRID

Programming

W r a p p i n g

Composing

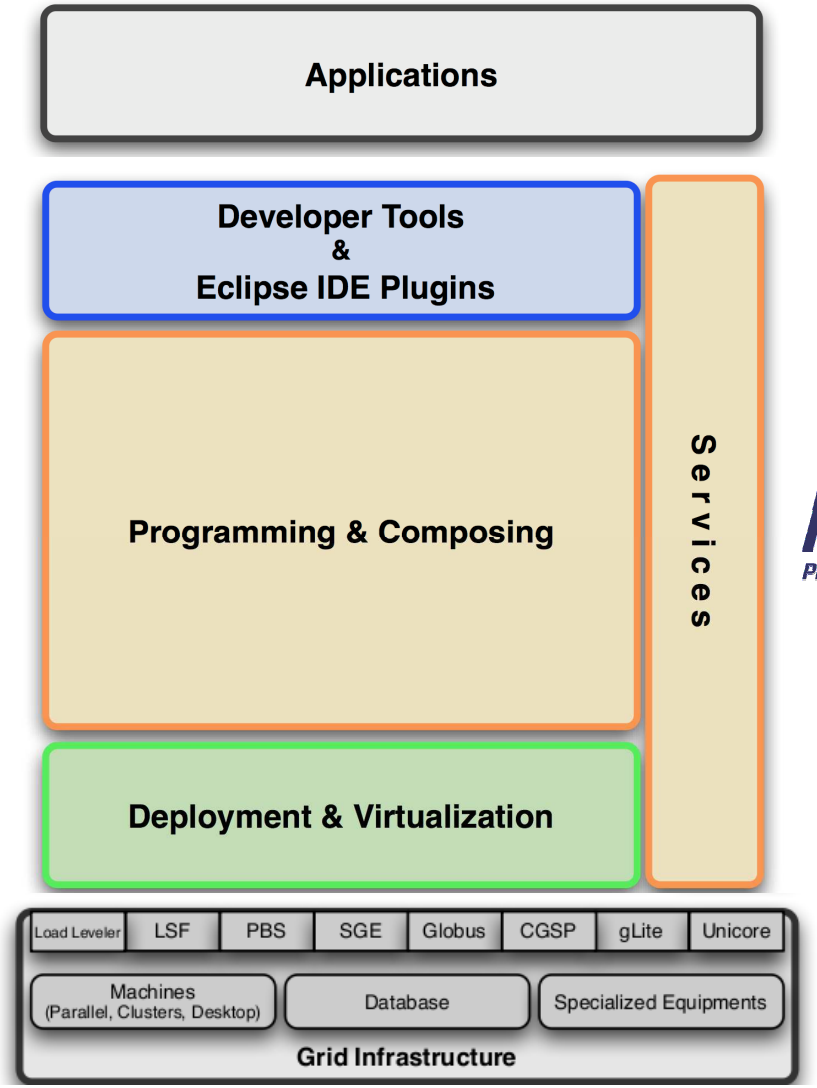
Deploying

Including:

P2P, File Transfer, Branch & Bound,
Load Balancing, Fault Tolerance

Overview

ProActive Parallel Suite (1)



ProActive
Programming, Composing, Deploying on the Grid

Open

Source

+

PROFESSIONAL

SUPPORT

ProActive Parallel Suite (1)

Developer Tools & Eclipse IDE Plugins

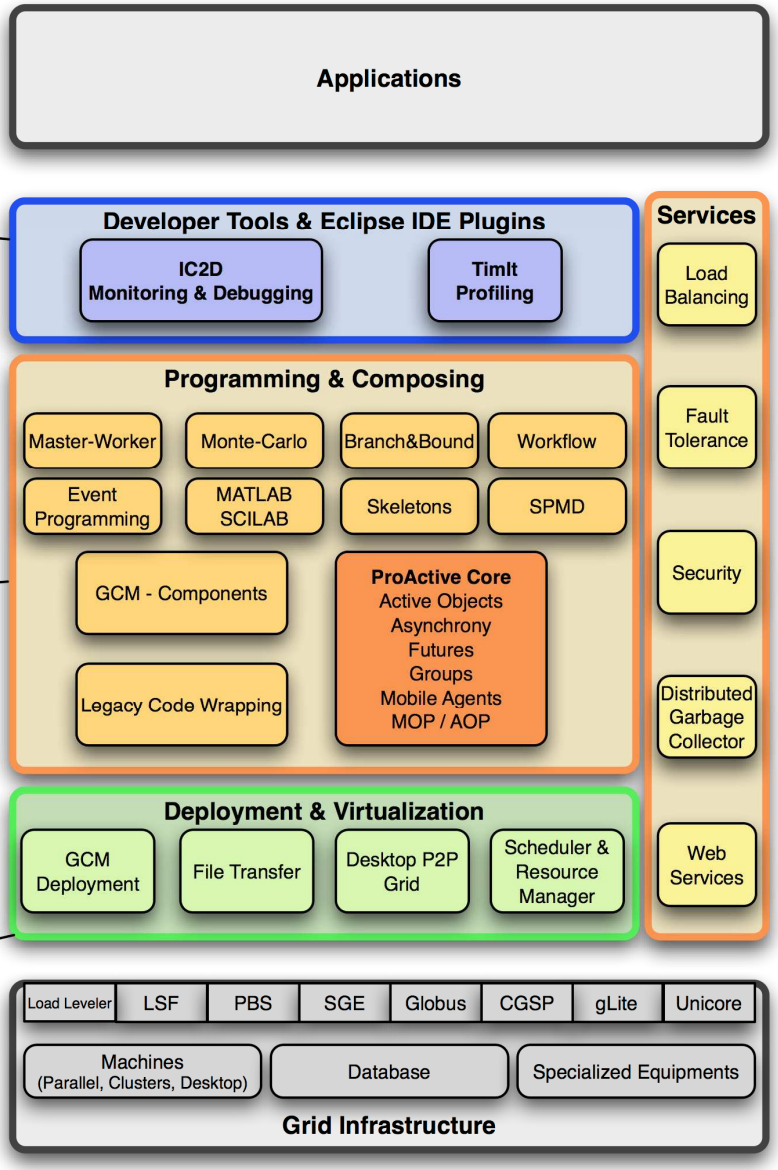
Standalone or Eclipse Plugin for monitoring parallelism and distribution, timing and benchmarking performances to improve applications.

Programming & Composing

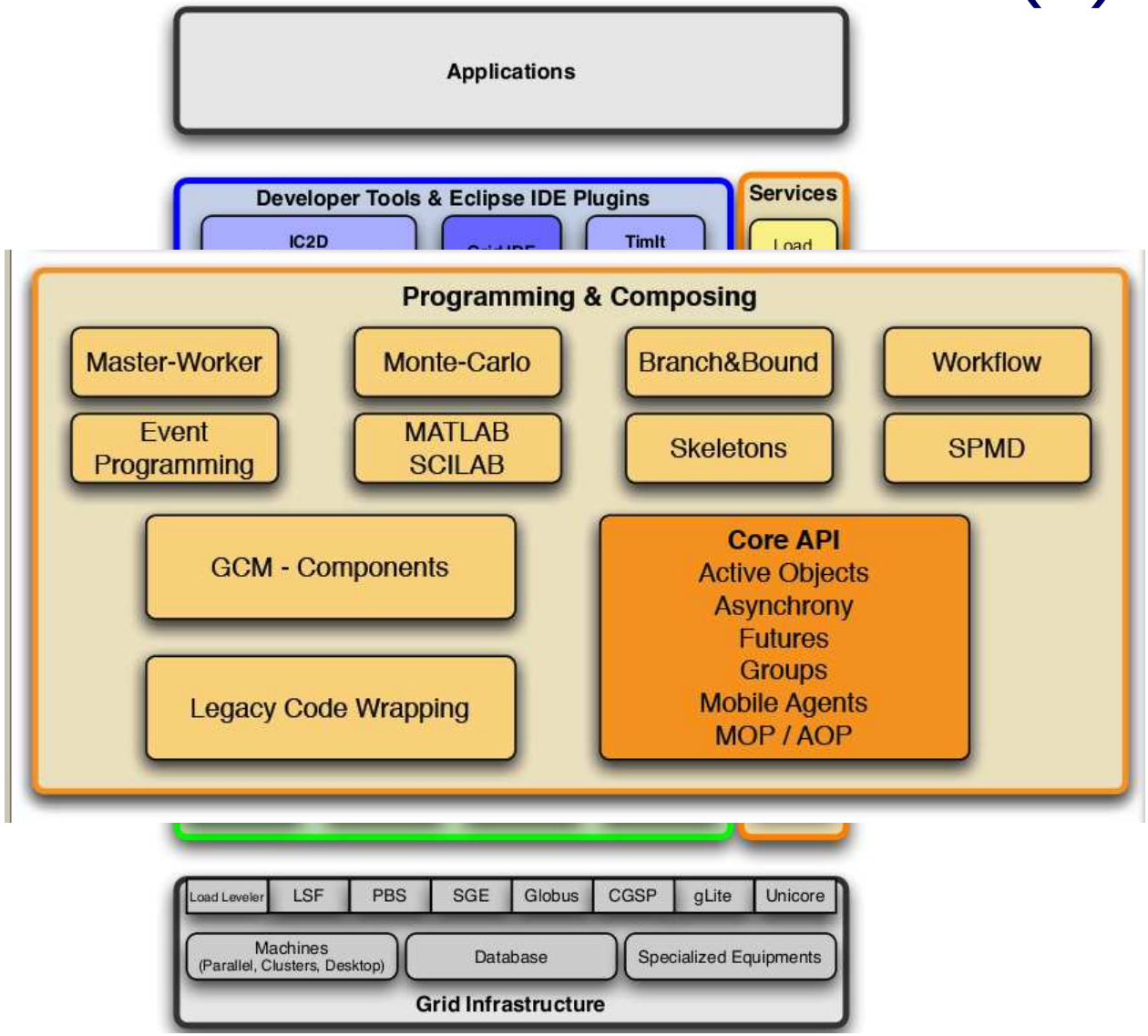
A Java API provides a comprehensive basic interface for the Grid: asynchrony, groups, parallelism, migration, among others. Components enable code reuse and composition, and Legacy code such as C, C++, or MPI can be gridified. High-level frameworks enable easy access to Grid programming for non-specialists.

Deployment & Virtualization

Provides interoperability with the main protocols and Grid infrastructures. Seamlessly integrates SOA with the capacity to configure programs and to specify file transfers.



ProActive Parallel Suite (1)



Programming Models

Programming models

Basic programming models supported by the ProActive middleware

	GUI	XML Format	Procedural API	Objects API
Workflow		**Yes**		
Skeleton				**Yes**
Components	**Yes**	**Yes**		**Yes**
Master worker (slave)				**Yes**
Branch & Bound				**Yes**
SPMD : MPI, object SPMD				**Yes**
Active object (actors)	**Yes**			**Yes**

Data Management

Data management: File transfer

Currently ProActive supports for the following type of transfers:

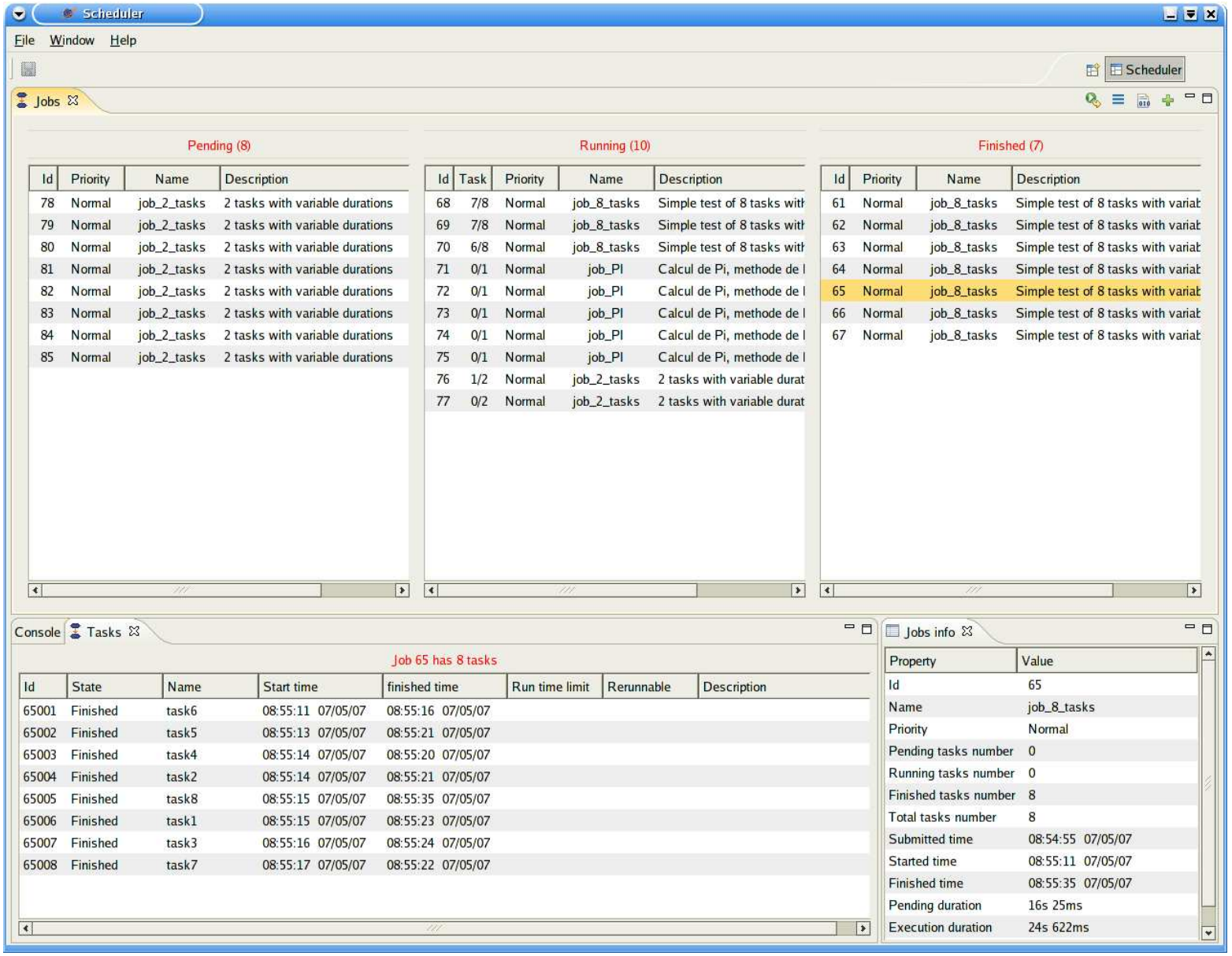
- **To a remote node (Push)**
- **From a remote node (Pull)**

It can be used at:

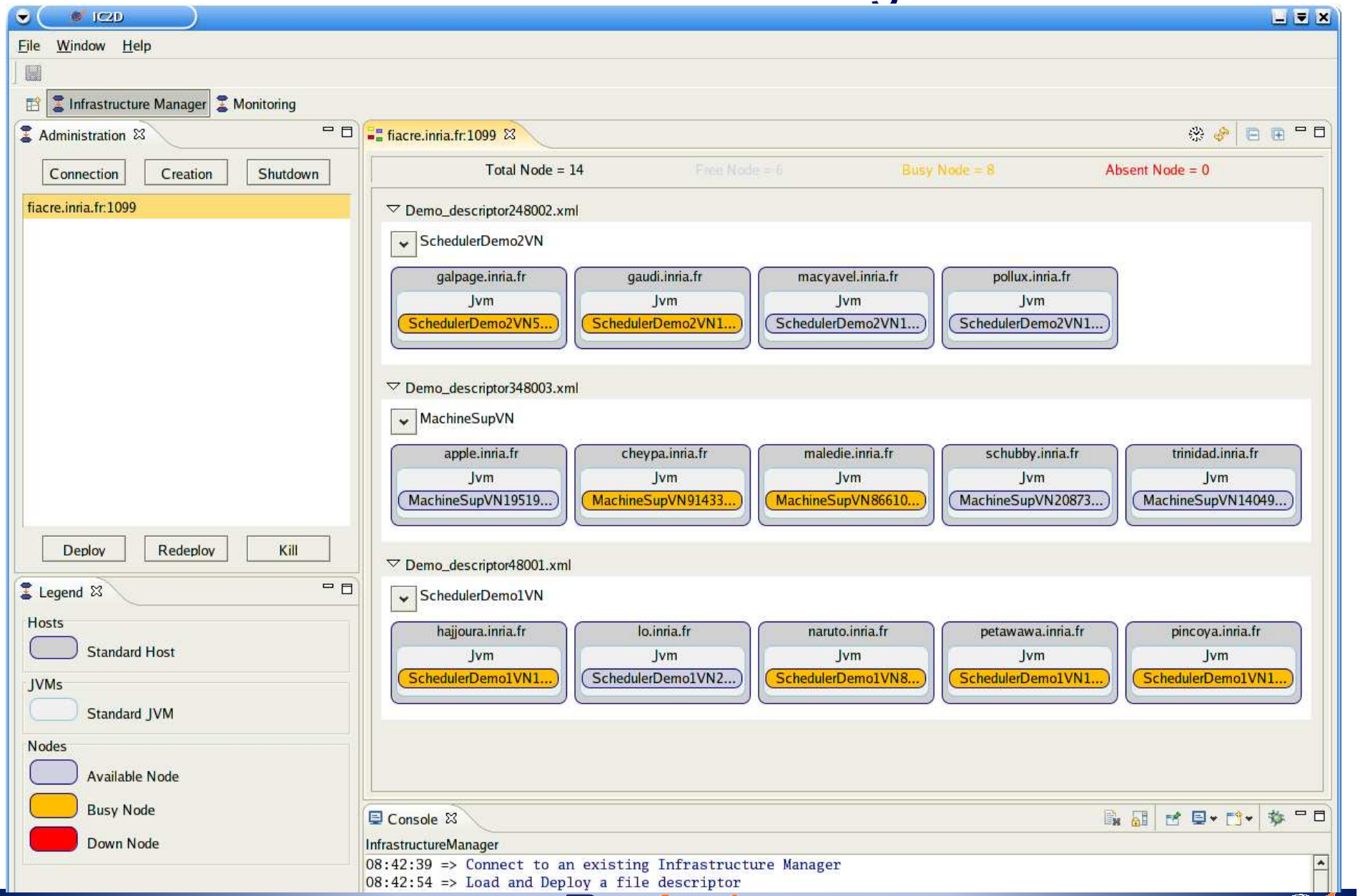
- **Deployment time (XML)**
- **Retrieval time (XML)**
- **Execution time (Objects API)**

Infrastructure:
Resource management & SLA

Scheduler: User Interface



Scheduler: Resource Manager Interface



Infrastructure management

Virtualisation of multi-core, multi-processors, servers, clusters: **Yes**

- ProActive/GCM makes it possible to:
  - Virtualize the infrastructure using host and VM capacity
  - Map the application onto the resulting virtual infrastructure

Meta scheduler, Broker: **Yes**

- ProActive Scheduler and Infrastructure Manager, two graphical application, can talk with primitive schedulers.

Scheduler: **Yes**

- ProActive Scheduler can deploy directly a list of jobs over various machines.

Peer to peer support for auto-organization : **Yes**

- Desktop P2P can be used to build desktop grid. Windows and Unix for heterogeneous grid. No need to change code of ProActive/GCM applications.

Hardware resources handled by the middleware:

- **computers: Yes**
- **Networks, storage devices, scientific instruments: No**

Security:

- AIC: Authentification, Integrity, confidentiality : **Yes**
  - **both middleware and application communications are handled by mechanism**
- Role management: **Partially**
- Full support for construction & administration of VO: **No**

Orthogonal Aspects

Globus based or not

- **No**

Approach to SOA

- **Yes, both component and Active Object can be exported as Web Service. An application can itself registered into a registry and lookup other services for SOA integration.**
- **Dynamic reconfiguration of Components**

Main languages: (C++, Java, C#, Fortran...)

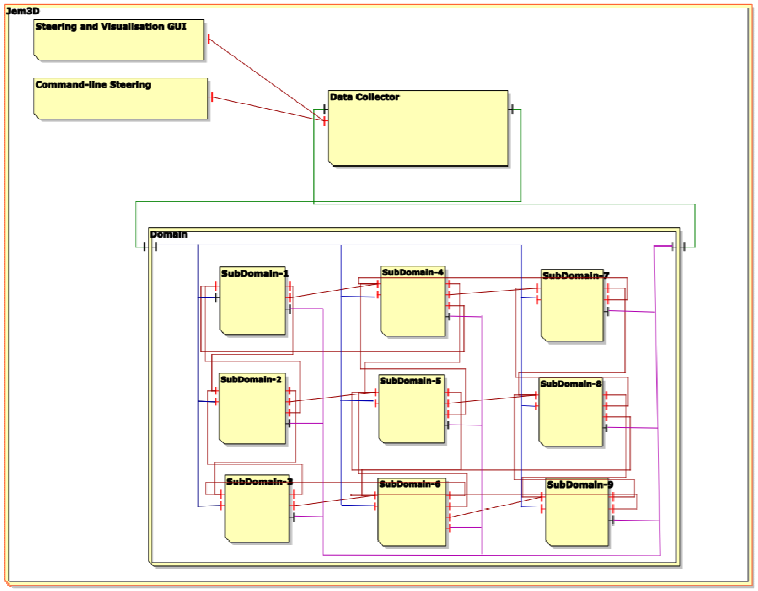
- **Java as principal language, and support wrapping of native legacy applications**

Interoperability with other Grid middleware: **Yes !!**

- **SSH, RSH, OARSH, ARC NorduGrid, CGSP China Grid, EGEE gLite, Fura/InnerGrid, GLITE, GLOBUS, GridBus, IBM Load Leveler, LSF, MPI, OAR, PBS, PRUN, SGE, Sun Grid Engine, UNICORE**

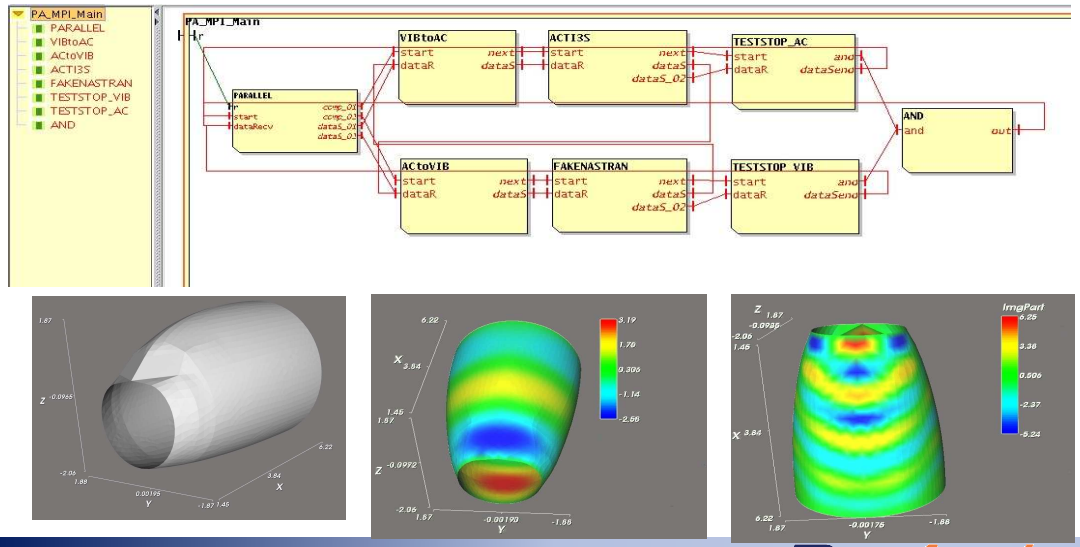
Applications

Current GCM experiments in ProActive



JEM3D: 3D electromagnetic application:

a single Cp on 300+ CPUs on Grid



Vibro-Acoustic application with EADS (legacy MPI coupling)

Applications: successfully executed

GridCOMP EU project use cases:

Biometric Identification System

Days Sales Outstanding

The mean time that clients delay to pay an invoice

EDR processing

Extended Data Record, extract, transform and Load in database large amount of data

WingDesign

Computes the aerodynamic wing performance for a given configuration

Other ProActive/GCM applications

Scilab	**Parallel execution of Scilab and Mathlab Scripts**
Mathlab	
BI	**Distribution of databases for Business Intelligence**
FlowShop	**A distributed approach for solving the FlowShop problem.**
Nbody	**The nbody problem (SPMD)**
NQueen	**An example of the N Queen problem distributed**
JEM3D	**An electromagnetism application**
C3D	**A distributed and collaborative 3D renderer.**
Monte carlo	**Stock pricing (EU and USA options)**
ProPotts	**MonteCarlo simulation for NLP**
OPSSA	**On-line power systems security analysis**
Salishan	**A resource sharing and simulation problem**
SAT solver	**Distributed SAT solvers in Java based on the SAT4J library**
ProFractal	**Parallel fractal image generation**
GeB	**Grid-enabled BLAST**
Cruise Control	**A real-time java controller to maintain a cruise speed**
NAS	**NASA benchmark suite (5 cores/5, 1 application/7)**

e-science, e-business, e-engineering

Distribution model

Open Source

Yes, the source code is under GPL version 2 licence.

Close Source

No

Commercial support

Yes, the ActiveEon company provides commercial support for the ProActive/GCM middleware.

ActiveEon


Activ on
SCALE BEYOND LIMITS
search
Login | Create Account
Home Products Solutions Services Partners Company Purchase Community
Parallel Computing
Multi-Cores to Enterprise Grids
Accelerating your applications
Open source
Virtualization
Simplicity
News
ProActive and Grid Component Model (GCM)
Budapest, Hungary, 3-7 Sept., 2007.
ProActive User Group, Tutorial, and Contest, 28 Oct.-2 Nov. 07, Beijing
Upcoming release Version 4.0, October 2007
More...
What we do
What we offer
Our Technology
ActiveEon is your dedicated technology partner for accelerating and scaling up your applications. We are a Professional Open Source company, co-developing and providing support for the ProActive Parallel Suite.
For Multi-threaded, Distributed and Enterprise Grids, our technology provides a uniform parallel computing interface for Multi-cores, Servers, and Clusters.
Open and non intrusive, ActiveEon is parallel computing made easy, for your applications, within your IT culture, making the most of your own infrastructure.
ProActive Parallel Suite
2 min video overview
Get latest release
Professional support
by Activ on
Site Map | Help | +33 4 1234 12345 Contact us

ActiveEon


Activ on
SCALE BEYOND LIMITS
search
Login | Create Account
Home Products Solutions Services Partners Company Purchase Community
Open Source and Open Standards
Open source
Virtualization
Simplicity
Accelerate
Development, Execution, Deployment, Operations
News
ProActive and Grid Component Model (GCM) Budapest, Hungary, 3-7 Sept., 2007.
ProActive User Group, Tutorial, and Contest, 28 Oct.-2 Nov. 07, Beijing
Upcoming release Version 4.0, October 2007
More...
What we do
What we offer
Our Technology
ActiveEon is your dedicated technology partner for accelerating and scaling up your applications. We are a Professional Open Source company, co-developing and providing support for the ProActive Parallel Suite.
For Multi-threaded, Distributed and Enterprise Grids, our technology provides a uniform parallel computing interface for Multi-cores, Servers, and Clusters.
Open and non intrusive, ActiveEon is parallel computing made easy, for your applications, within your IT culture, making the most of your own infrastructure.
ProActive Parallel Suite
2 min video overview
Get latest release
Professional support
by Activ on
Site Map | Help | +33 4 1234 12345 Contact us

ActiveEon

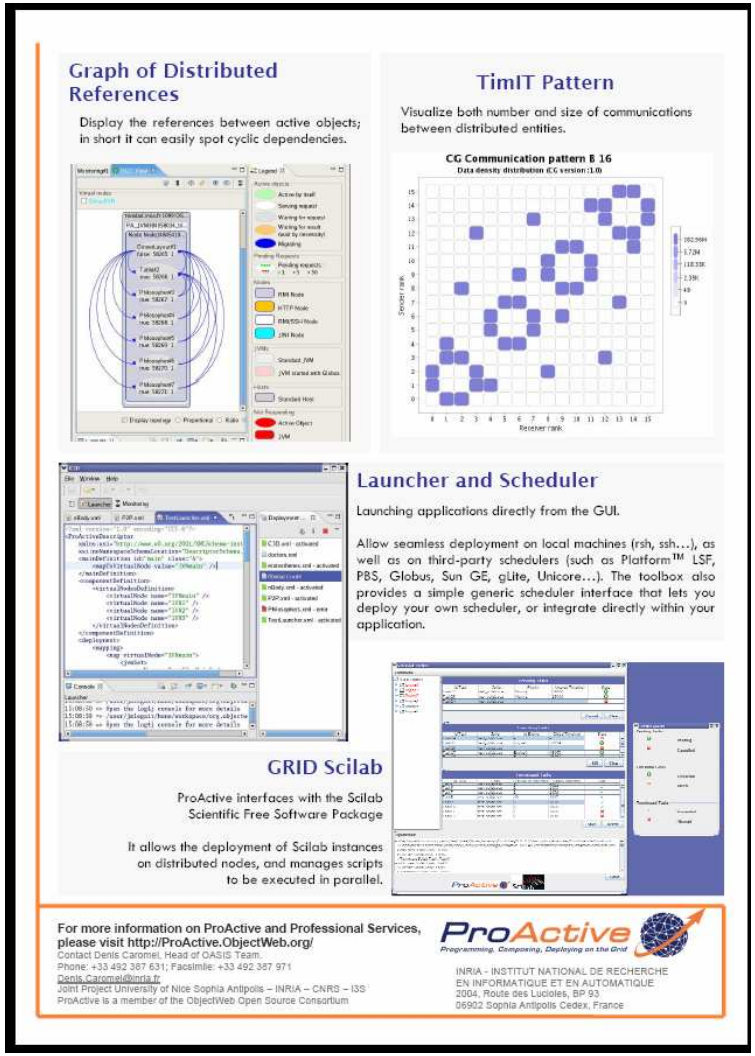

Activ on
SCALE BEYOND LIMITS
search
Login | Create Account
Home Products Solutions Services Partners Company Purchase Community
Tools for Your applications
within Your IT culture
with Your infrastructure
Open source
Virtualization
Simplicity
News
ProActive and Grid Component Model (GCM)
Budapest, Hungary, 3-7 Sept., 2007.
ProActive User Group, Tutorial, and Contest,
28 Oct.-2 Nov. 07, Beijing
Upcoming release Version 4.0, October 2007
More...
What we do
What we offer
Our Technology
ActiveEon is your dedicated technology partner for accelerating and scaling up your applications. We are a Professional Open Source company, co-developing and providing support for the ProActive Parallel Suite.
For Multi-threaded, Distributed and Enterprise Grids, our technology provides a uniform parallel computing interface for Multi-cores, Servers, and Clusters.
Open and non intrusive, ActiveEon is parallel computing made easy, for your applications, within your IT culture, making the most of your own infrastructure.
ProActive
Parallel Suite
2 min video overview
Get latest release
Professional support
by Activ on

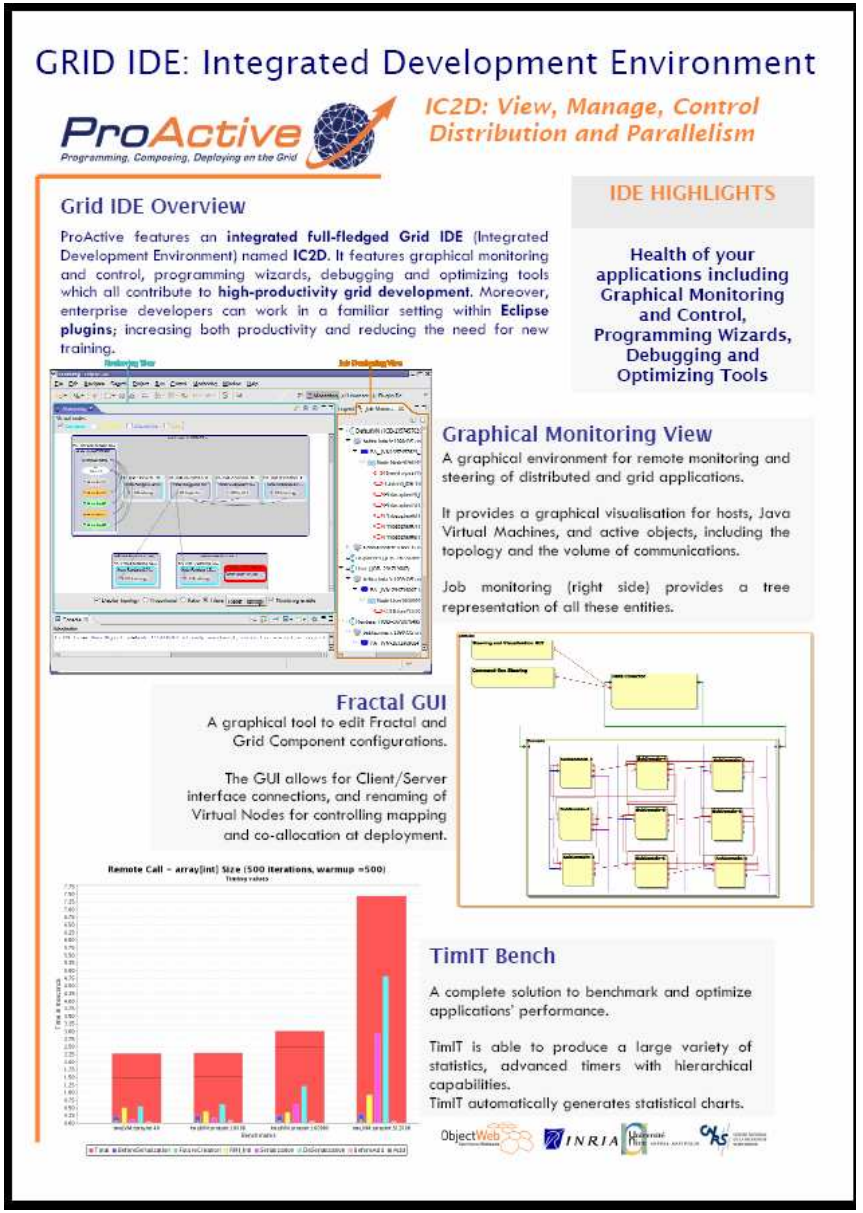
ActiveEon


Activ on
SCALE BEYOND LIMITS
search
Login | Create Account
Home Products Solutions Services Partners Company Purchase Community
Open source
Virtualization
Simplicity
Parallel Computing Made and Easy
News
ProActive and Grid Component Model (GCM)
Budapest, Hungary, 3-7 Sept., 2007.
ProActive User Group, Tutorial, and Contest, 28 Oct.-2 Nov. 07, Beijing
Upcoming release Version 4.0, October 2007
More...
What we do
What we offer
Our Technology
ActiveEon is your dedicated technology partner for accelerating and scaling up your applications. We are a Professional Open Source company, co-developing and providing support for the ProActive Parallel Suite.
For Multi-threaded, Distributed and Enterprise Grids, our technology provides a uniform parallel computing interface for Multi-cores, Servers, and Clusters.
Open and non intrusive, ActiveEon is parallel computing made easy, for your applications, within your IT culture, making the most of your own infrastructure.
ProActive Parallel Suite
2 min video overview
Get latest release
Professional support
by Activ on
Site Map | Help | +33 4 1234 12345 Contact us

ProActive GUIs

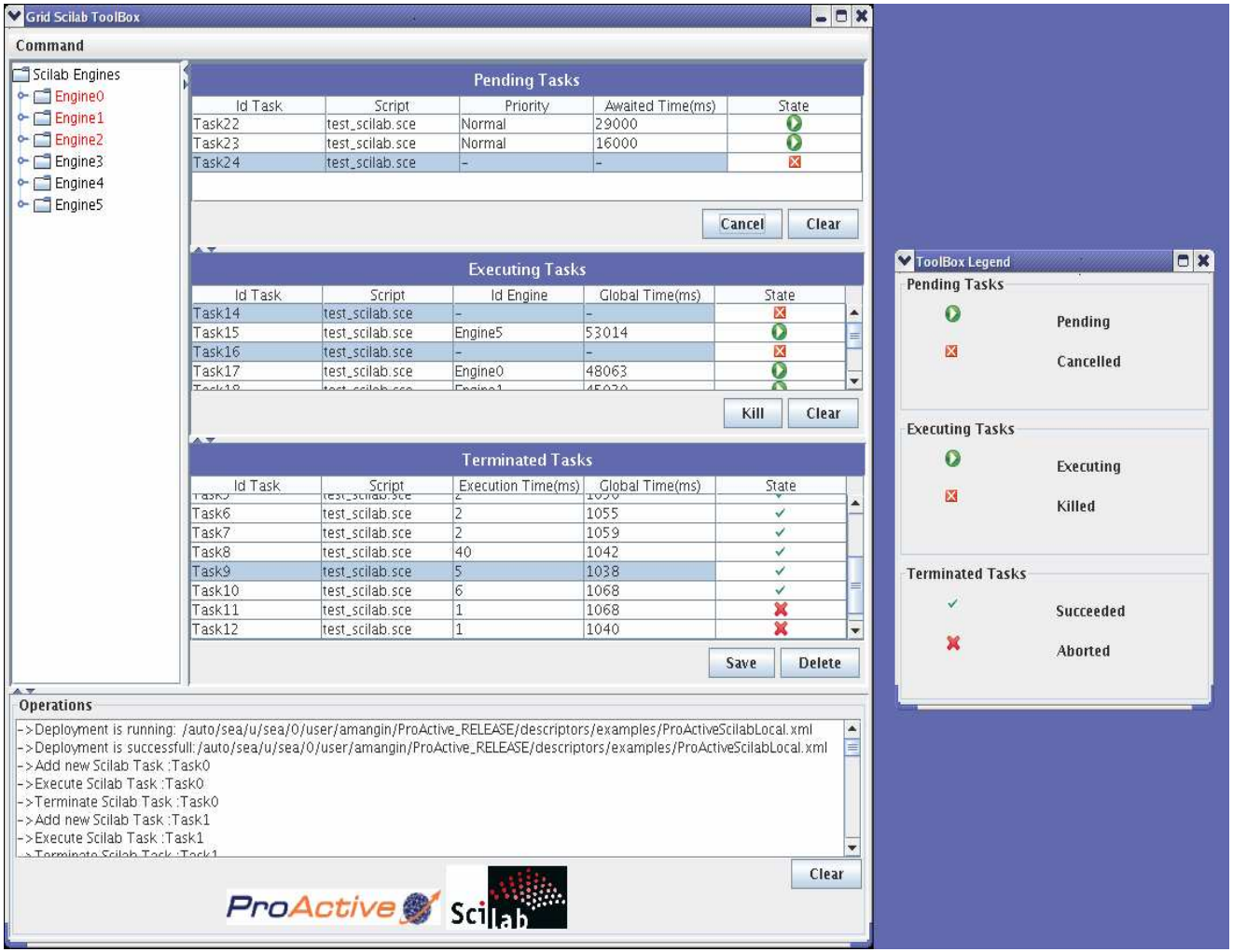
IC2D Eclipse, ***TimIt, ...***





Interface with The Scientific Free Software

SciLab



ProActive

A 100% Java API + Tools for Parallel, Distributed Computing

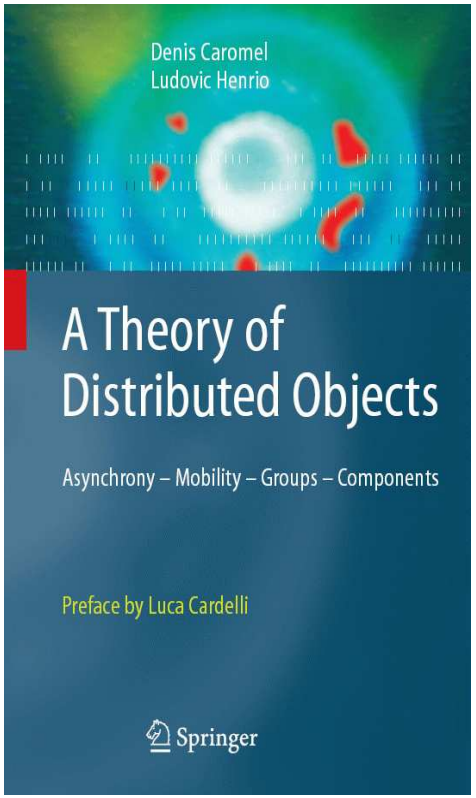
A uniform framework: **An Active Object pattern**
A formal model behind: **Determinism (POPL'04)**

Programming Model:

Remote Objects:
Asynchronous Communications, Wait-By-Necessity, Groups, Mobility, Components, Security, Fault-Tolerance: Checkpoints
Environment:
XML Deployment Descriptors:
Interfaced with: `rsh, ssh, LSF, PBS, Globus, Jini, SUN Grid Engine`
Graphical Visualization and monitoring: IC2D
In the www. ObjectWeb .org Consortium
(Open Source **LGPL**)

ObjectWeb
Open Source Middleware

THEORY



A Theory of Distributed Objects

D. Caromel, L. Henrio,

Springer 2005, Monograph

A Calculus:

ASP: Asynchronous Sequential Processes

- **Based on Sigma-Calculus (Abadi-Cardelli)**
- **Formal Proofs of determinism**
- **Releases a few important implementation constraints**

Contents

Example of ProActive Training

LECTURES

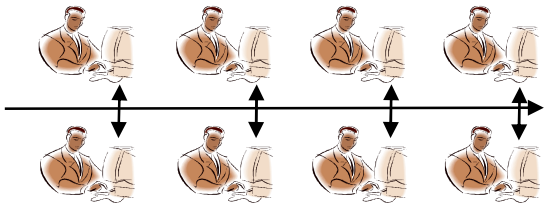
Introduction: The multiple GRIDs
Active Objects, Wait-By-Necessity

Group communications, OO SPMD, Mobility

Deployment:
rsh, ssh, Globus, LSF, gLite, etc.
Fault-Tolerance: Exceptions and Checkpointing
Peer-to-Peer
Web Services

Components:
Introduction to Components
From Objects to Components
Hierarchical Components
A few words about Theory: ASP
A Theory of Distributed Objects

PRACTICALS
Hands-On



1. Installation,
2. C3D: Coll.+Parall.+Dist.
3. Synchro(RW,Phil)
4. Penguin, Jacobi
5. Jacobi Deployment
6. Hands-On programming
7. Finish up previous ex.
8. Components Programming

To be precisely defined according to students' knowledge and main interests

Table of Contents (1)

Table of Contents (2)

Table of Contents (3)

Parallelism

- Multi-Processing: → GRID COMPUTING

  MPP: Massively Parallel Programming or

  Message Passing Parallelism

  Solutions: PVM, MPI, … RMI, sockets

  Distributed programming

- Multi-Threading:
  - SMP: Symmetric Multi-Processing

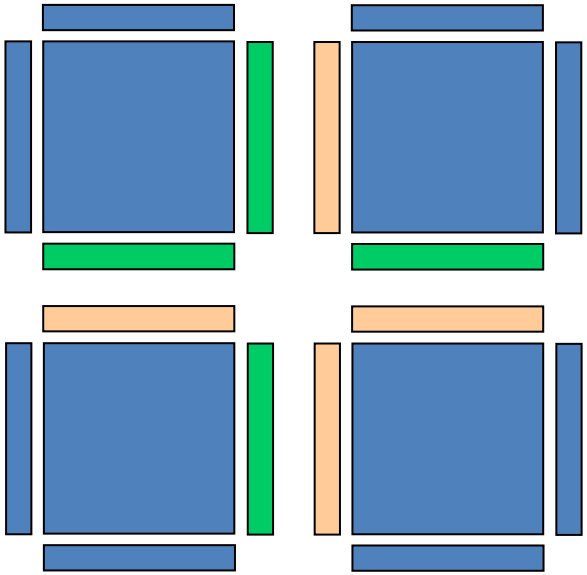
    Shared-Memory Parallelism

    Solutions: OpenMP, pThreads, … Java Threads

    Multi-Core Programming

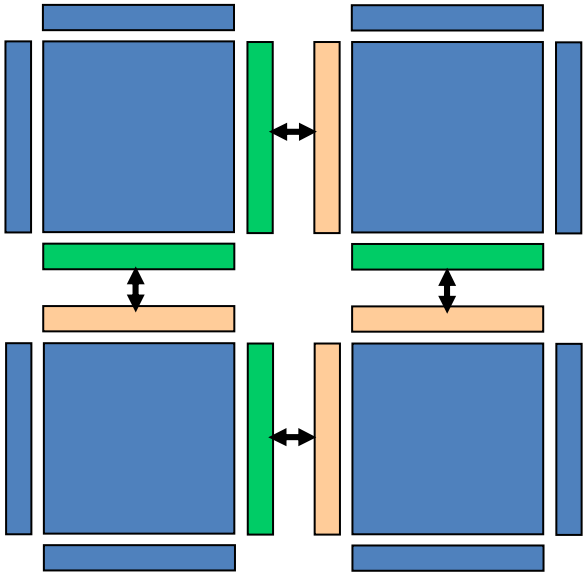
Unification of Multi-Threading and Multi-Processing
NoC for Multi-Core: Network On a Chip

Jacobi: MPI



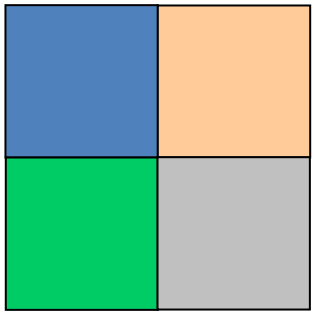
```
! Matrix init
…

Call mpi_barrier(comm2d,ierr)

it = 0

Do while ( diffnorm > tolerance )

      it = it + 1

      Call exchng2( a,sx,ex,sy,ey,comm2d,
     belowX,aboveX,belowY,aboveY,dims)

      Call sweep2d(a,f,nx,sx,ex,sy,ey,b)

      dwork = diff(a,b,sx,ex,sy,ey)

      Call mpi_allreduce(
     dwork,diffnorm,1,mpi_double,mpi_sum,comm2d,ierr)

End Do
```

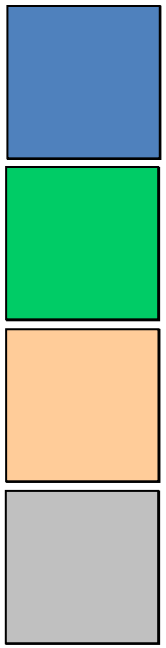
Jacobi: MPI



```
Subroutine exchng2(
    ab,sx,ex,sy,ey,comm2d,belowX,aboveX,belowY,aboveY,dims)

If ( dims(1) > 1 ) then
    Call mpi_sendrecv(
        ab(ex,sy:ey),ey-sy+1,mpi_double,      aboveX,0,   &
        ab(sx-1,sy:ey),ey-sy+1,mpi_double,    belowX,0,   &
        comm2d,status,ierr)

    Call mpi_sendrecv(
        ab(sx,sy:ey),ey-sy+1,mpi_double,      belowX,1,   &
        ab(ex+1,sy:ey),ey-sy+1,mpi_double,    aboveX,1,   &
        comm2d,status,ierr)
End If

If ( dims(2) > 1 ) then
    Call mpi_sendrecv(
        ab(sx:ex,ey),ex-sx+1,mpi_double,      aboveY,2,   &
        ab(sx:ex,sy-1),ex-sx+1,mpi_double,    belowY,2,   &
        comm2d,status,ierr)

    Call mpi_sendrecv(
        ab(sx:ex,sy),ex-sx+1,mpi_double,      belowY,3,   &
        ab(sx:ex,ey+1),ex-sx+1,mpi_double,    aboveY,3,   &
        comm2d,status,ierr)
End If
End Subroutine exchng2
```

Jacobi: OpenMP





error

```
#pragma omp parallel private(resid, i) {
  while (k <= maxit && error > tol) {
    /* copy new solution into old */
    #pragma omp for
      for (j=0; j<m; j++)
          for (i=0; i<n; i++)
              uold[i + m*j] = u[i + m*j];

    #pragma omp for reduction(+:error)
      /** Compute **/
      for (j=1; j<m-1; j++)
           for (i=1; i<n-1; i++){
                u[i + m*j] =( uold[i-1 + m*j] + uold[i+1 + m*j]
                      + uold[i + m*(j-1)] + uold[i + m*(j+1)] )/ 4;

                /* accumulate residual error */
                error = error + u[i +m*j];
           } /* end for */

    /* error check */
    #pragma omp master
      k++;
      error = sqrt(error) /(n*m);

  } /* end while */
} /* end pragma parallel */
```

The GRIDs

PCs : 1 Milliard in 2002 (25 years) Forecast: 2 Milliards in 2008

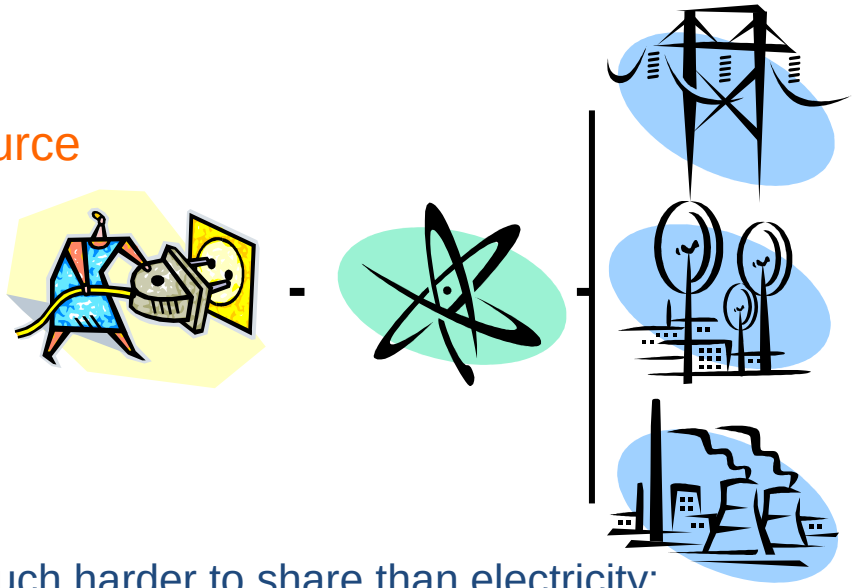
The Grid Concept

GRID = Electric Network

Computer Power (CPU cycles) ⟷ Electricity

Can hardly be stored, if not used → Lost

Global management,
Mutual sharing of the resource



But CPU cycles are much harder to share than electricity:

- Production cannot be adjusted!
- Cannot really be delivered where needed!
- Not yet interoperable: Multiple Administrative Domains

2 important aspects : Computational + Data Grid

Example

Challenge: Scale up

50 Machines, 1.5 year of computation

5000 Machines, with only 50 % Efficiency

==> 10 days

Applications:

- Simulate the stock market evolution
- Research for an urgent vaccine
- Forecast a bush-fire path
- Forecast in real time a flooding consequences ...
- etc.

Ubiquitous: some numbers

PCs in my lab (INRIA Sophia) : ~ 1500 French Riviera : 1.3 Millions

France : 25 Millions Europe : 108 Millions USA : 400 Millions

World : 1 Milliard in 2002 (25 ans) Forecast: 2 Milliards in 2008

France :

- 36 Millions of cellular phones
- 2.2 Millions of laptops
- 630 Thousand PDA

The multiple GRIDs

Scientific Grids :

- Parallel machines, Clusters
- Large equipments: Telescopes, Particle accelerators, etc.

Enterprise Grids :

- Data, Integration: Web Services
- Remote connection, Security

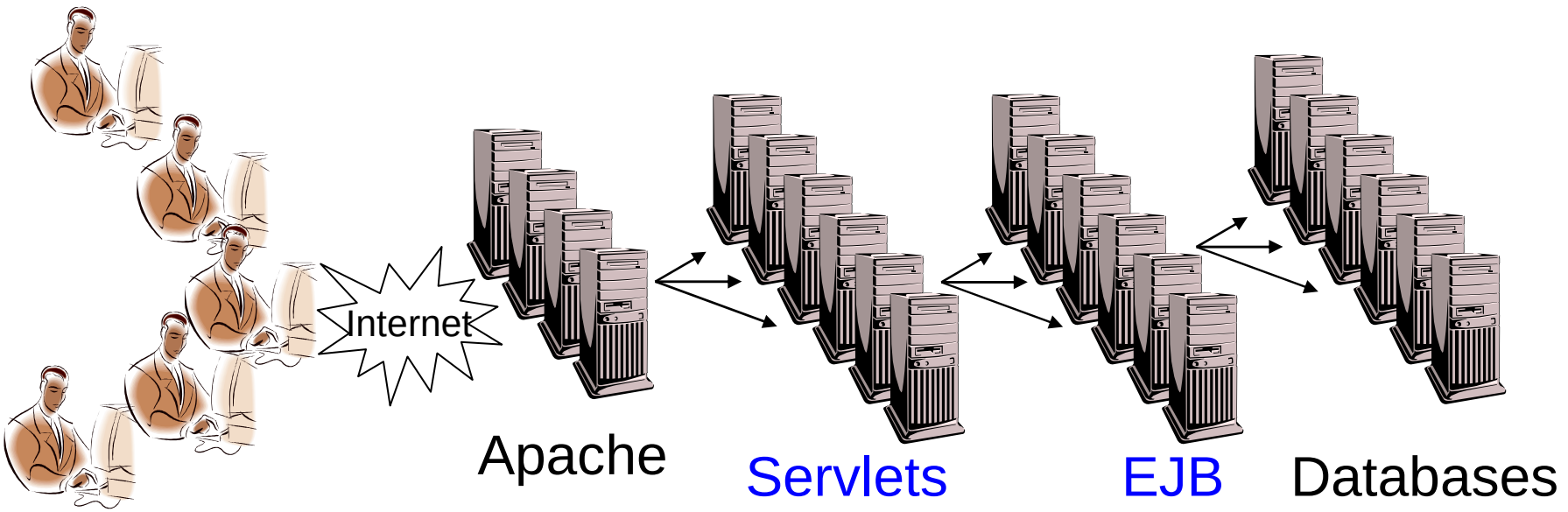
Internet Grids, (miscalled P2P grid):

- Home PC: Internet Grid (e.g. SETI@HOME)

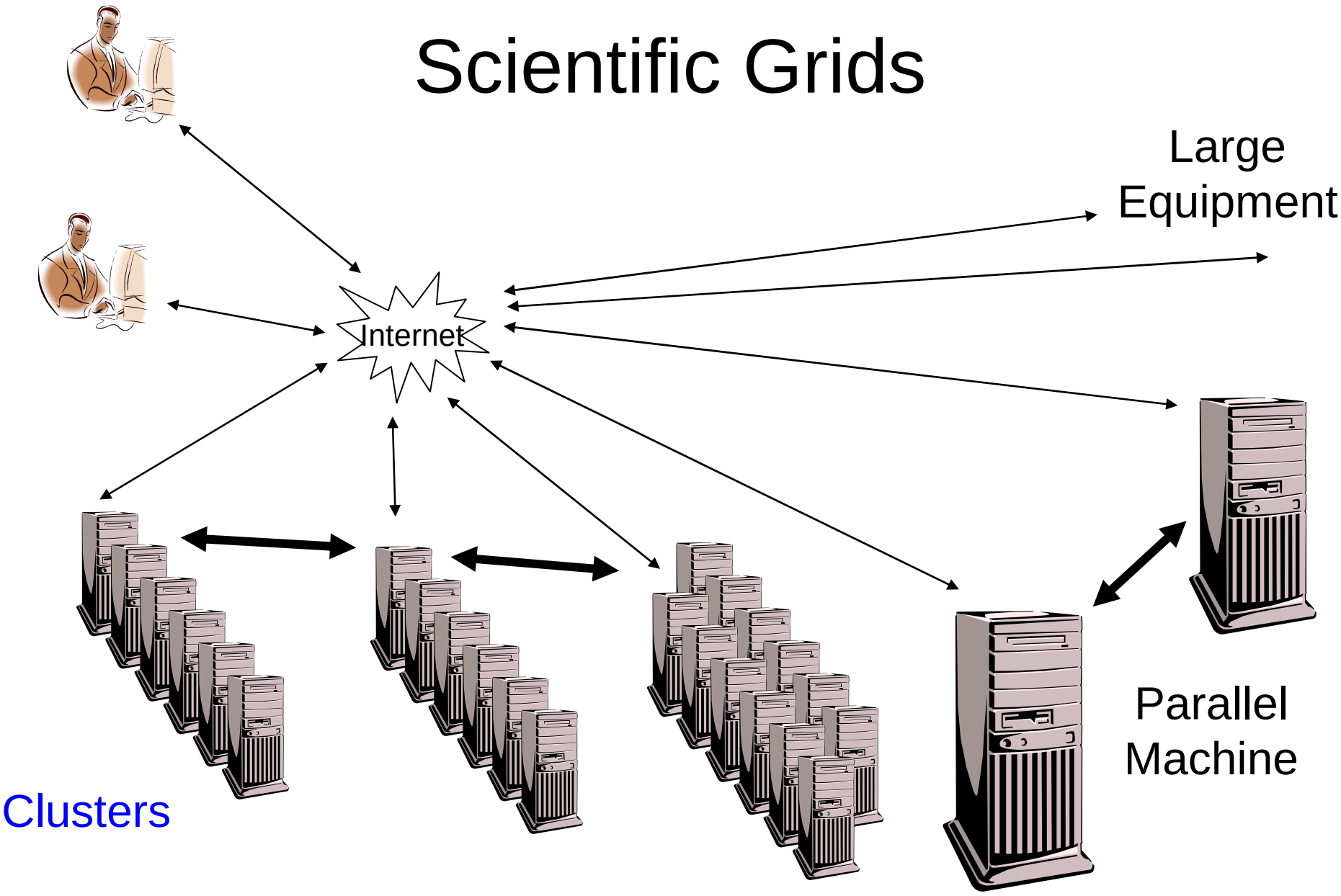
Intranet Desktop Grids

- Desktop office PCs: Desktop Intranet Grid

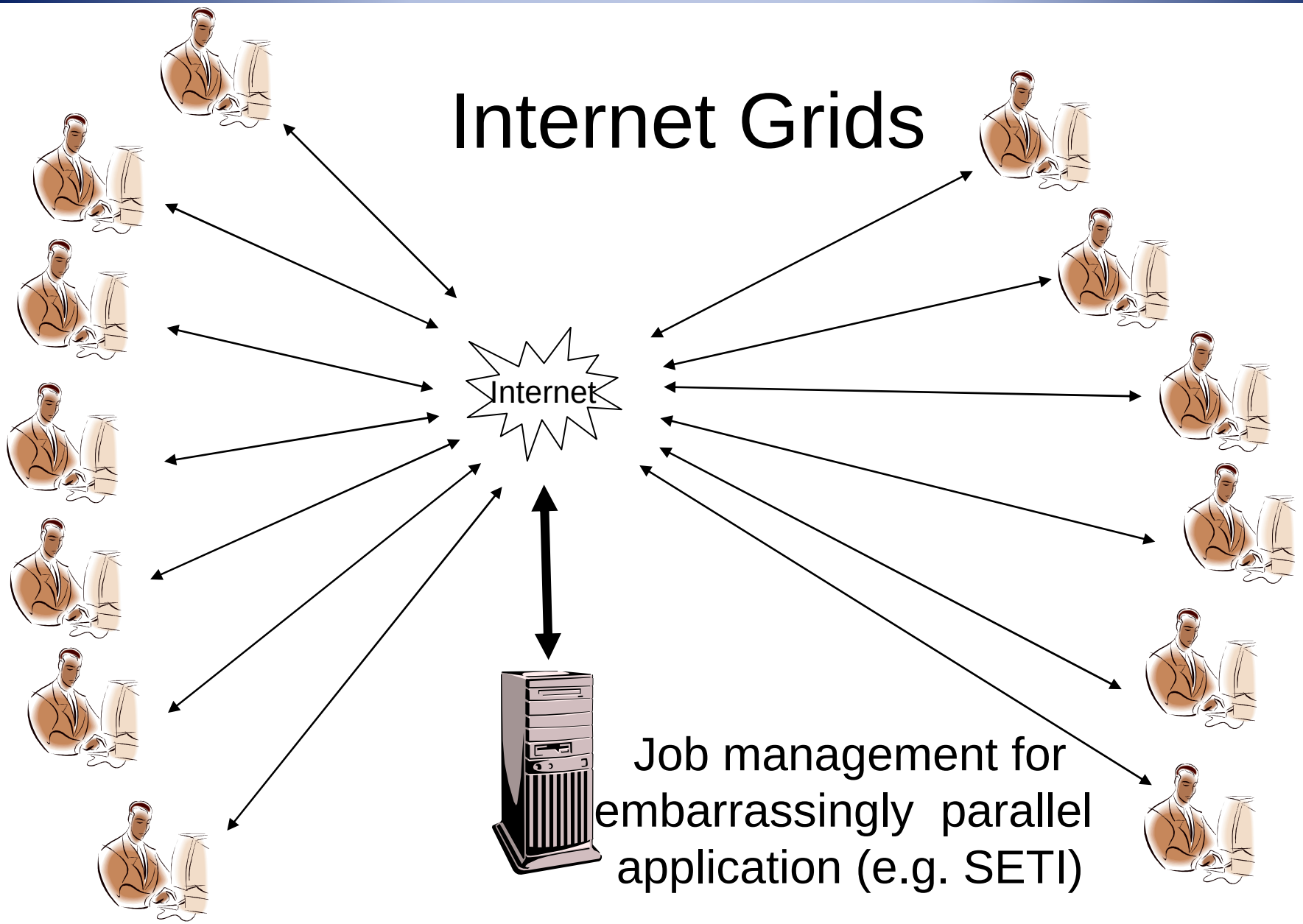
Enterprise Grids



Scientific Grids

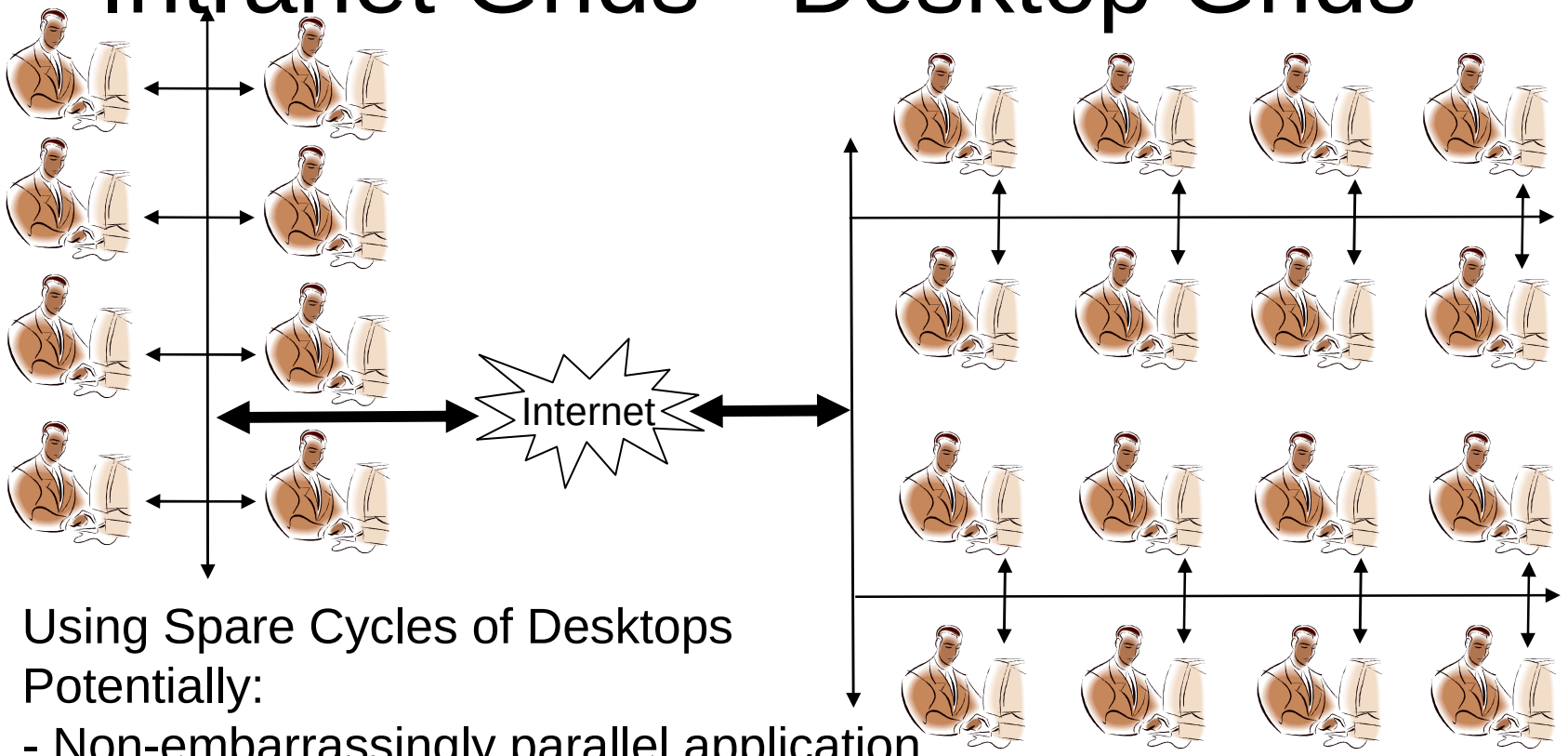


Internet Grids



Job management for embarrassingly parallel application (e.g. SETI)

Intranet Grids - Desktop Grids



Using Spare Cycles of Desktops
Potentially:
- Non-embarrassingly parallel application
- Across several sites

How to Program ?
How to Deploy ?

The multiple GRIDs

Scientific Grids

Enterprise Grids

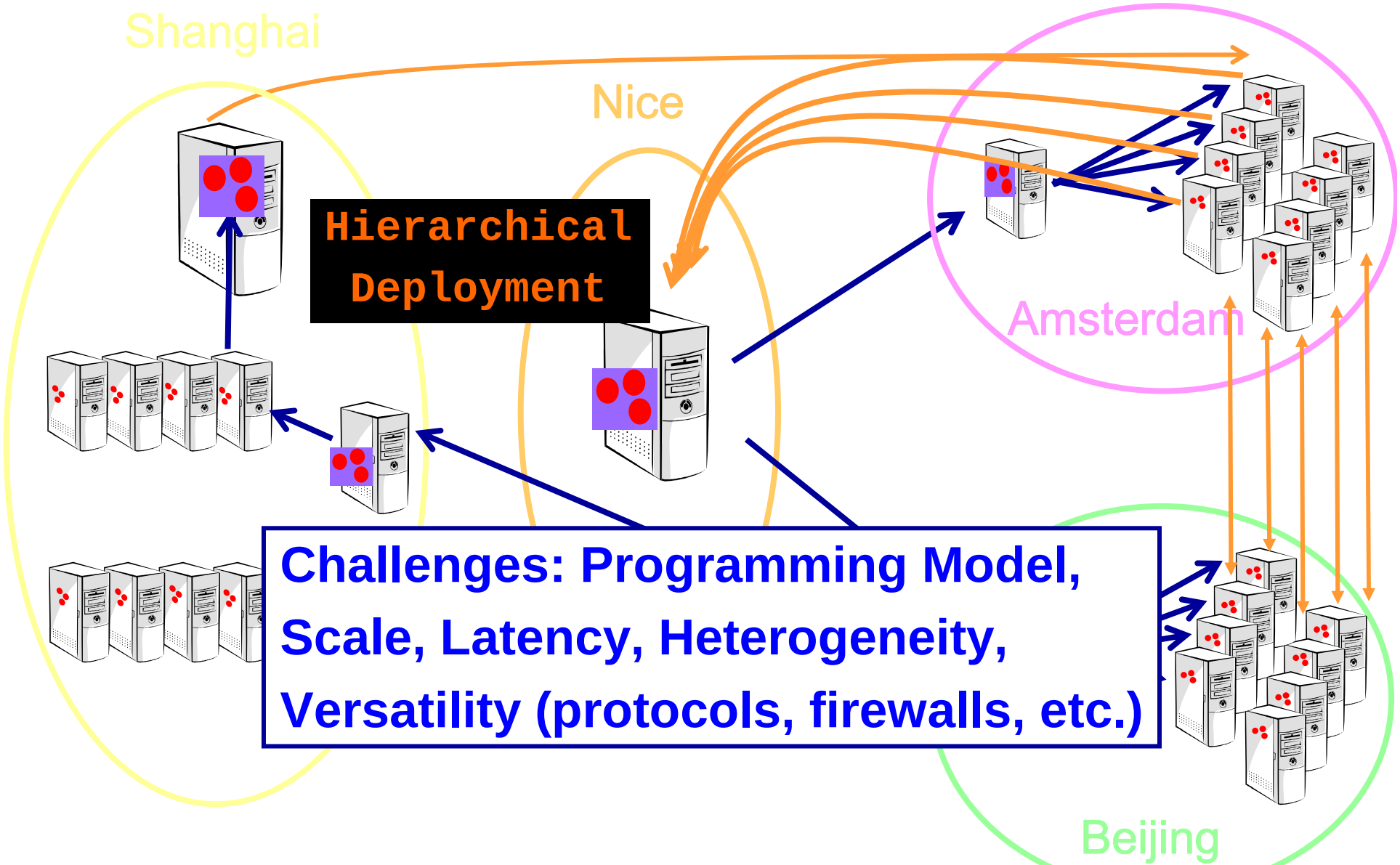
Internet Grids

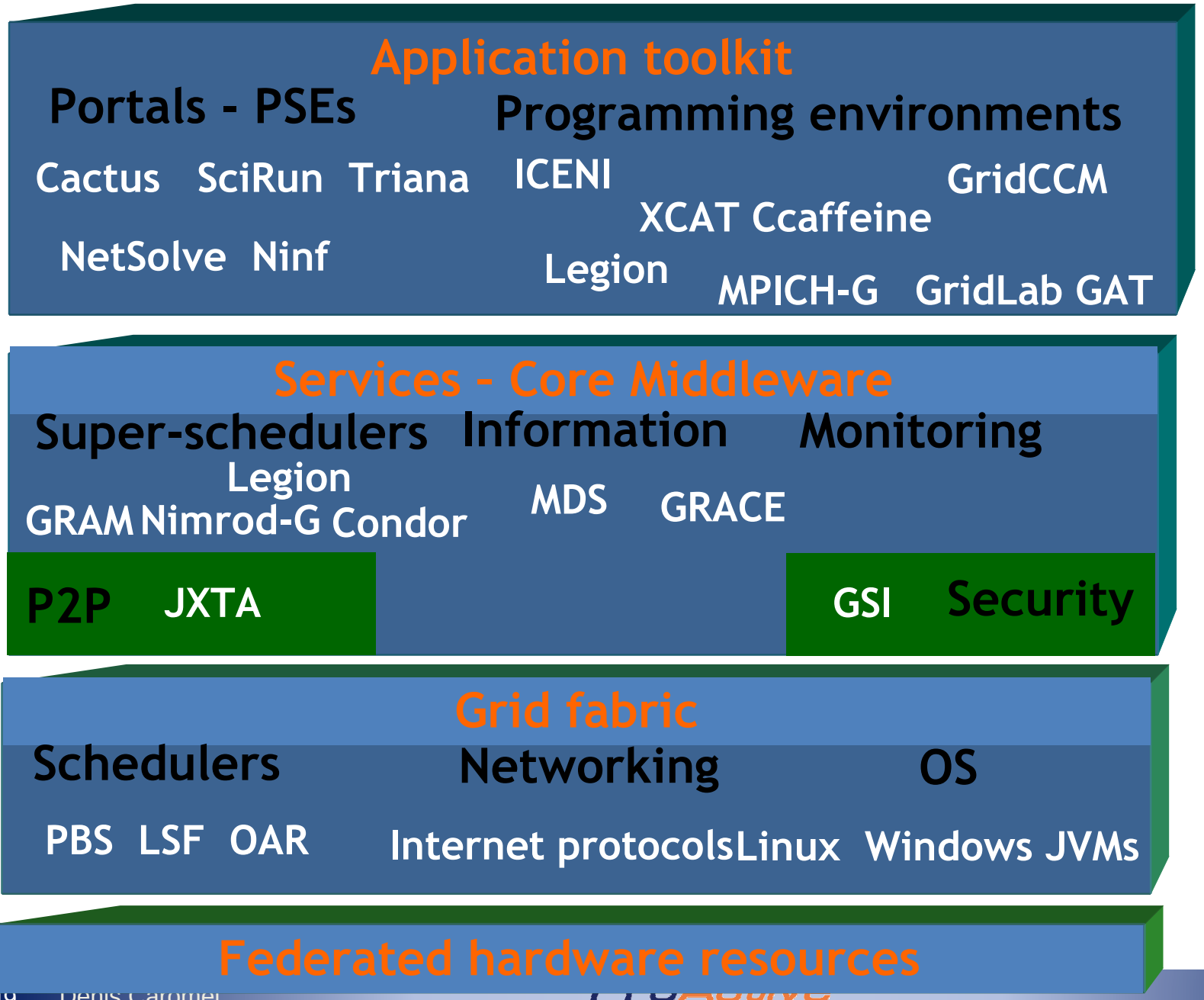
Intranet Desktop Grids

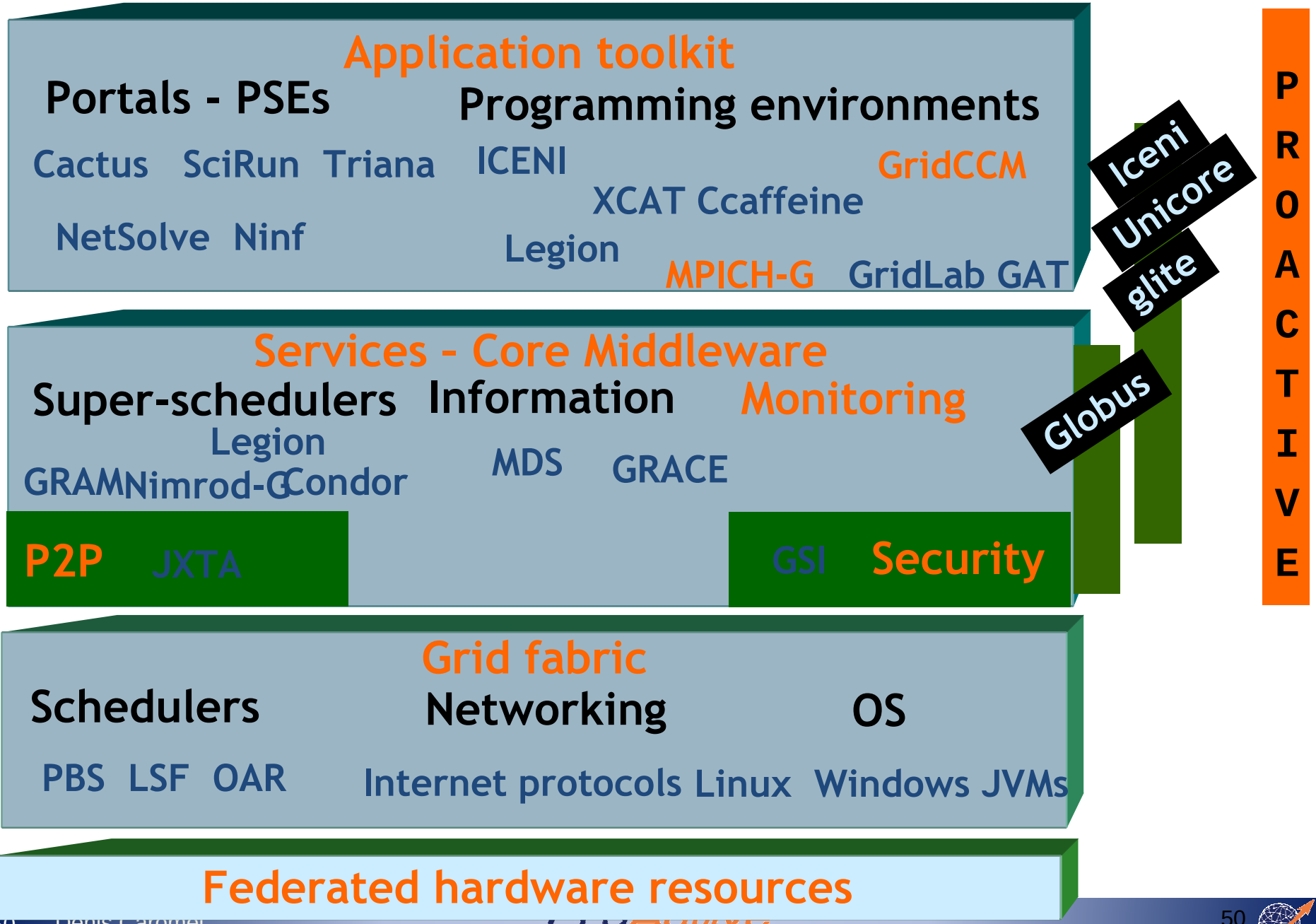
Strong convergence in process!

Infrastructure level, i.e. WS, and Programming

Grid Computing with *ProActive*

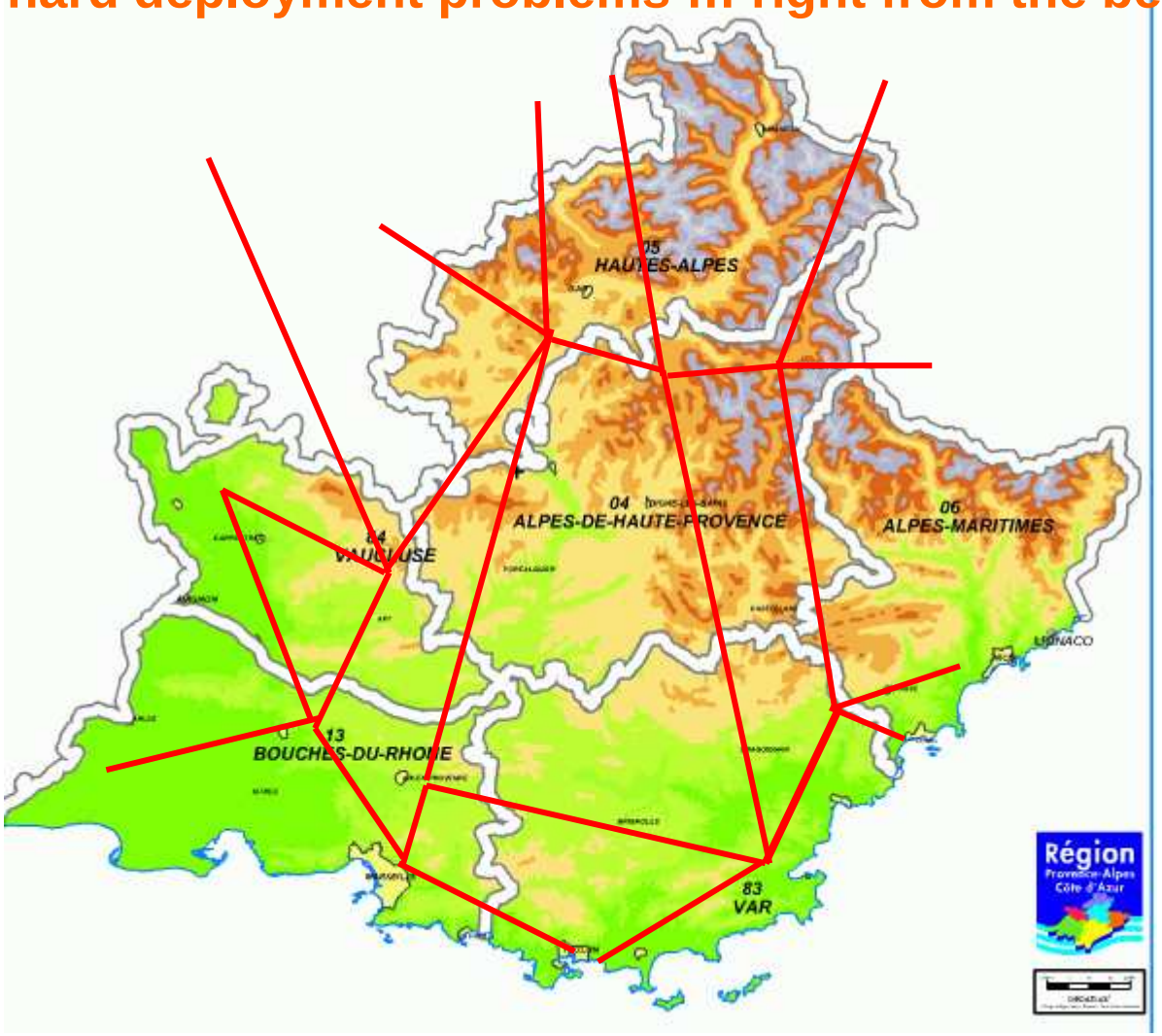



Application toolkit
Portals - PSEs
Programming environments
Cactus
SciRun
Triana
ICENI
GridCCM
XCAT
Ccaffeine
NetSolve
Ninf
Legion
MPICH-G
GridLab GAT
Services - Core Middleware
Super-schedulers
Information
Monitoring
Legion
GRAM
Nimrod-G
Condor
MDS
GRACE
P2P
JXTA
GSI
Security
Grid fabric
Schedulers
Networking
OS
PBS
LSF
OAR
Internet protocols
Linux
Windows
JVMs
Federated hardware resources


Application toolkit
Portals - PSEs
Programming environments
Cactus
SciRun
Triana
ICENI
GridCCM
XCAT
Ccaffeine
NetSolve
Ninf
Legion
MPICH-G
GridLab GAT
Iceni
Unicore
glite
Globus
PROACTIVE
Services - Core Middleware
Super-schedulers
Information
Monitoring
Legion
GRAM
Nimrod-G
Condor
MDS
GRACE
P2P
JXTA
GSI
Security
Grid fabric
Schedulers
Networking
OS
PBS
LSF
OAR
Internet protocols
Linux
Windows
JVMs
Federated hardware resources

Grid: from enterprise ... to regional

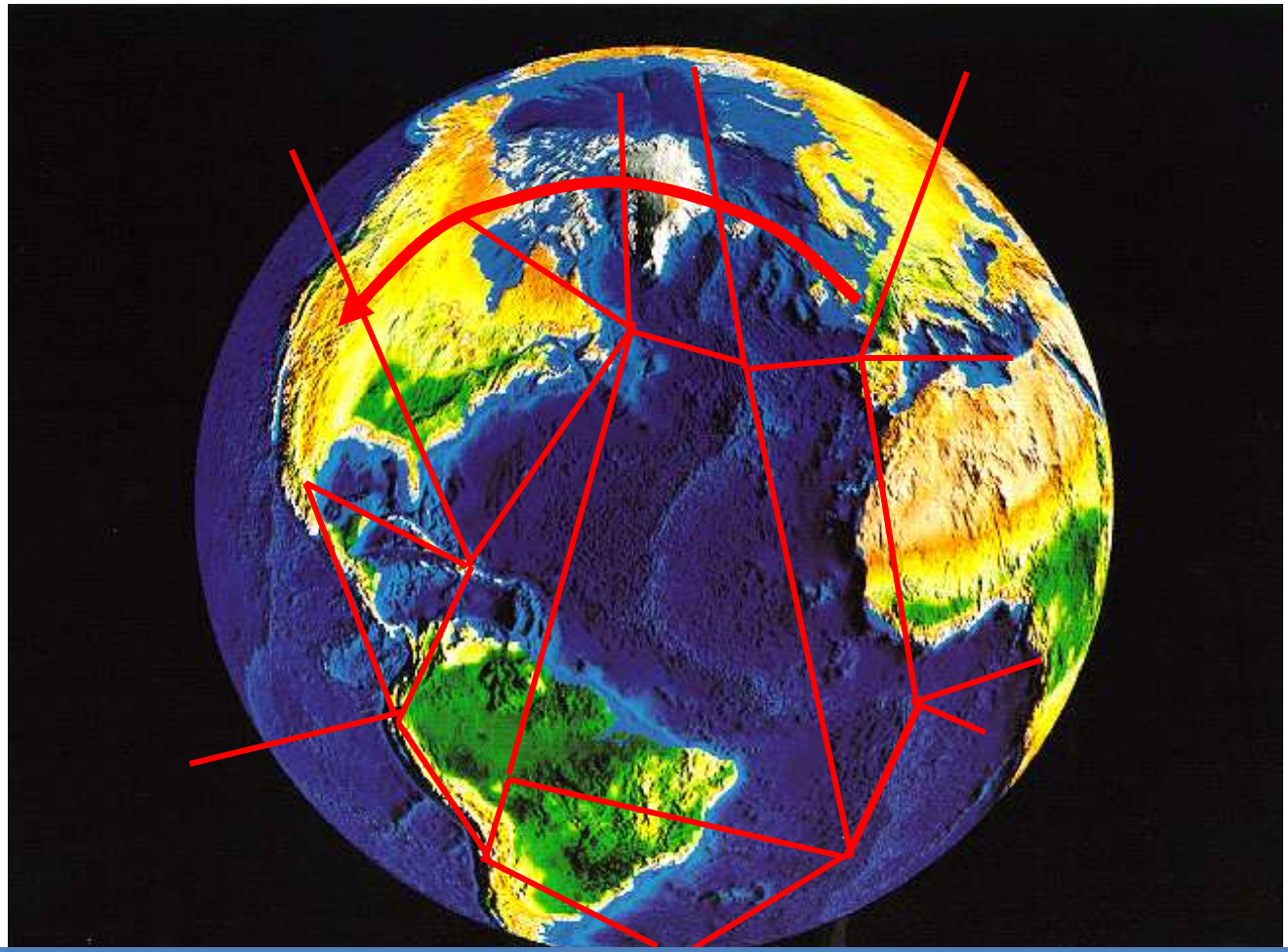
Very hard deployment problems … right from the beginning



Grid: from regional ... to worldwide

Communication Nice-Los Angeles: 70 ms Light Speed

Challenge: Hide the latency !



Define adequate programming model

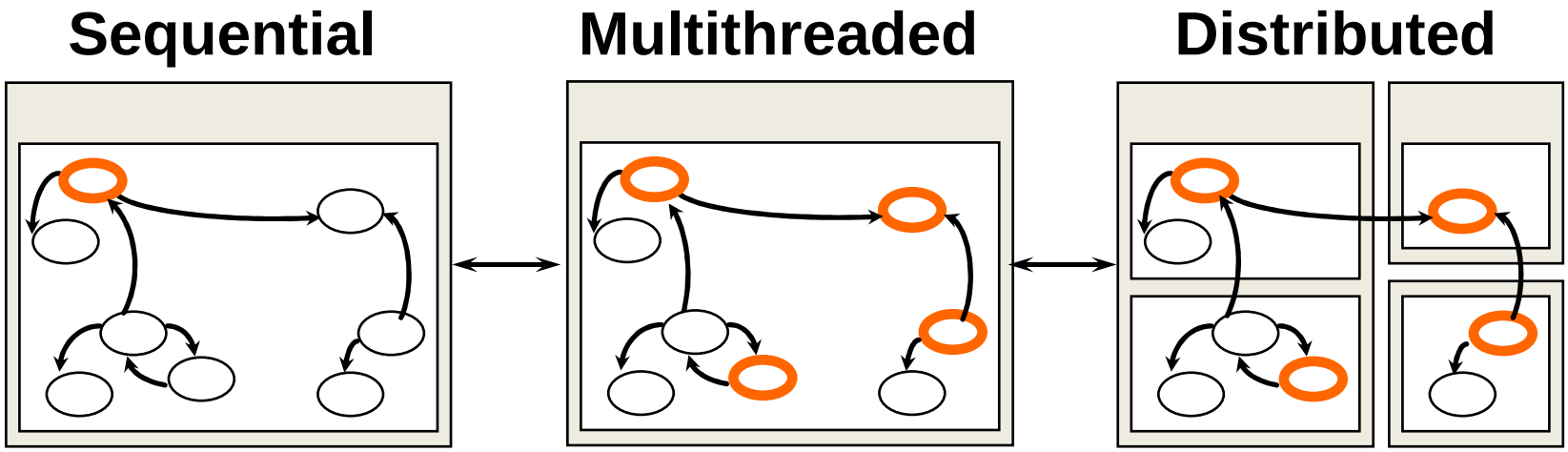
1. Distributed and Parallel Objects

ProActive

Programming

ProActive
Objectives and Rationale

Seamless



Most of the time, activities and distribution are not known at the beginning, and change over time

Seamless implies reuse, smooth and incremental transitions

Library !

ProActive model (1)

Java RMI (Remote Method Invocation = Object RPC = o.foo(p))

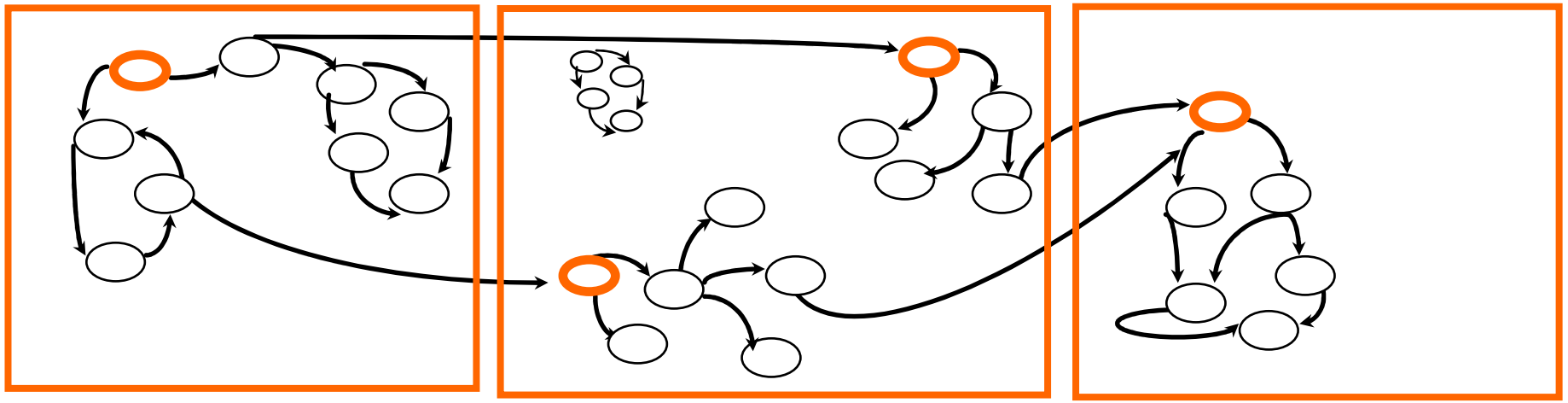
plus a few important features:

Asynchronous Method calls towards Active Objects:

Implicit Futures as RMI results

Wait-By-Necessity:

- » Automatic wait upon the use of an implicit future
- » First-Class Futures:
  - Futures passed to other activities
  - Sending a future is not blocking



ProActive : model (2)

Active objects : coarse-grained structuring entities (subsystems)

Each active object: - possibly owns many passive objects

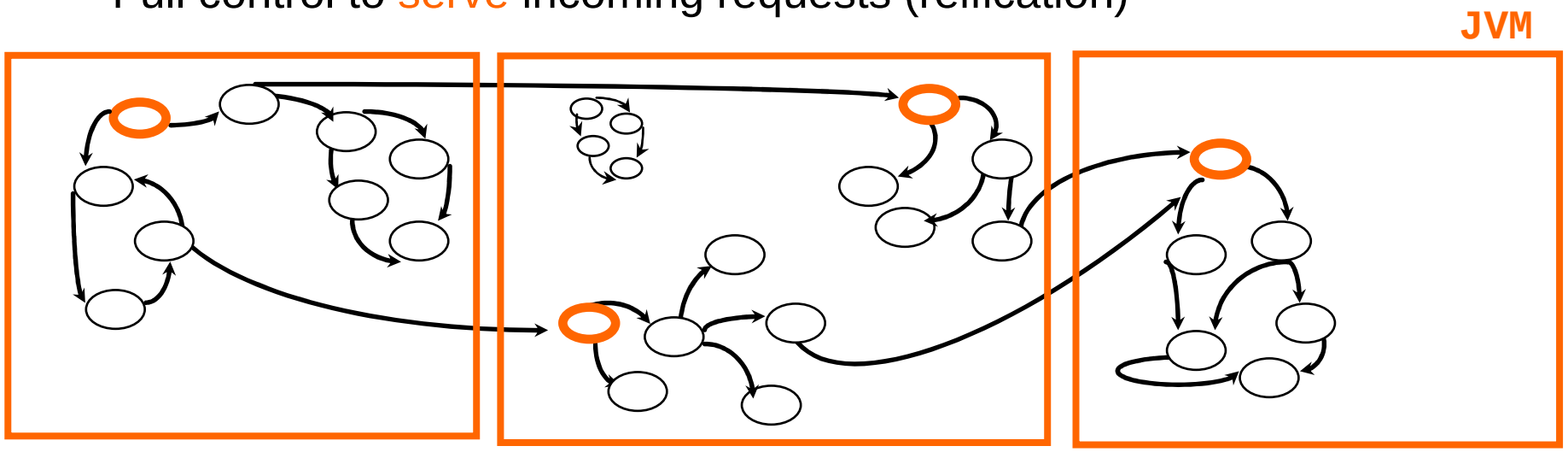
- has exactly one thread.

No shared passive objects -- Parameters are passed by deep-copy

Asynchronous Communication between active objects

Future objects and wait-by-necessity.

Full control to serve incoming requests (reification)



ProActive : Creating active objects

An object created with `A a = new A (obj, 7);`
can be turned into an active and remote object:

Instantiation-based: The most general case.

```
A a = (A)ProActive.newActive(«A», params, node);
```

Class-based: In combination with a static method as a factory

To get a non-FIFO behavior (Class-based):

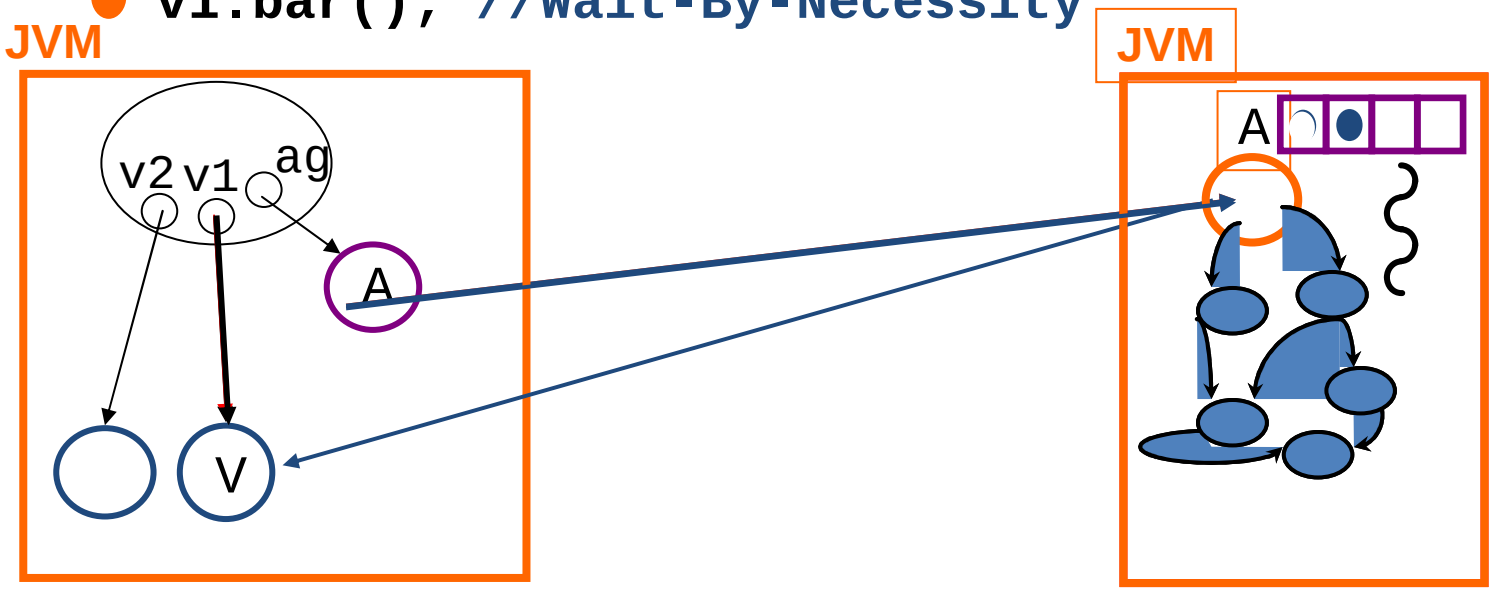
```
class pA extends A implements RunActive { … }
```

Object-based:

```
A a = new A (obj, 7);
...
...
a = (A)ProActive.turnActive (a, node);
```

ProActive : Active objects

```
A ag = newActive ("A", […], VirtualNode)
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```



Java Object

Active Object

Req. Queue

Future Object

Proxy

Request

Thread

Wait-By-Necessity is a Dataflow Synchronization

Wait by necessity

A call on an active object consists in 2 steps

- **A query** : name of the method, parameters…
- **A Reply** : the result of the method call

A query returns a Future object which is a placeholder for the result

The callee will update the Future when the result is available

The caller can continue its execution event if the Future has not been updated

```
foo ()
{
  Result r = a.process();
  //do other things
  ...
  r.toString();
}
```

will block if not available

```
Result process()
{
  //perform long
  //calculation

  return result;
}
```

ProActive : Explicit Synchronizations

```
A ag = newActive ("A", […], VirtualNode)
V v = ag.foo(param);
...
v.bar(); //Wait-by-necessity
```

Explicit Synchronization:

- `ProActive.isAwaited (v);` // Test if available
- `.waitFor (v);` // Wait until availab.

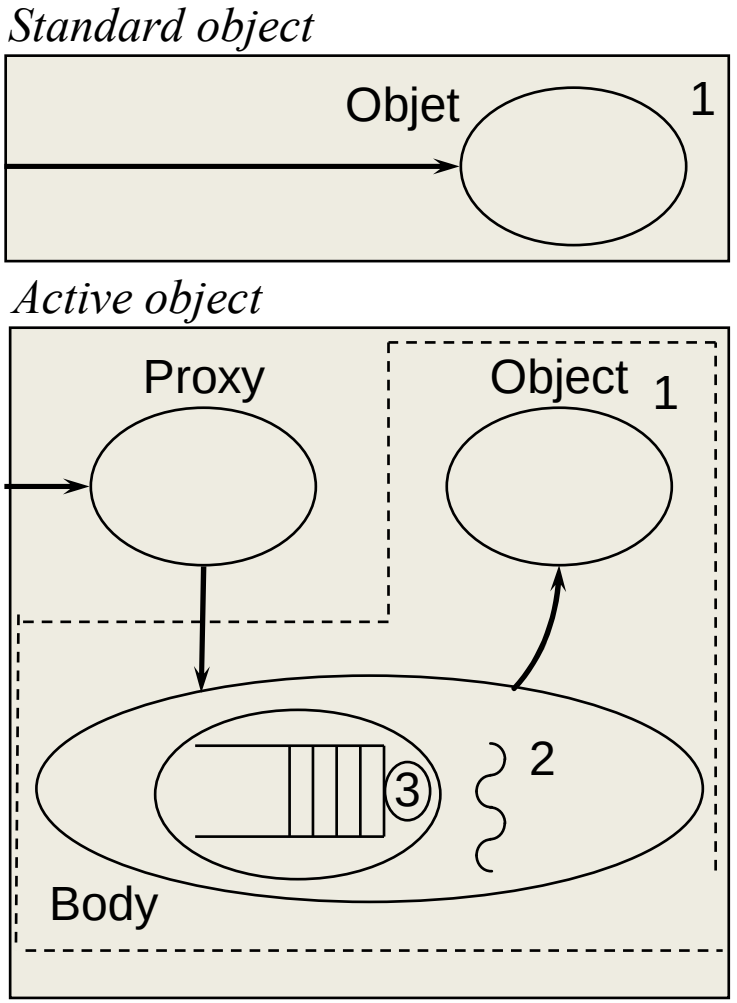
Vectors of Futures:

- `.waitForAll (Vector);` // Wait All
- `.waitForAny (Vector);` // Get First

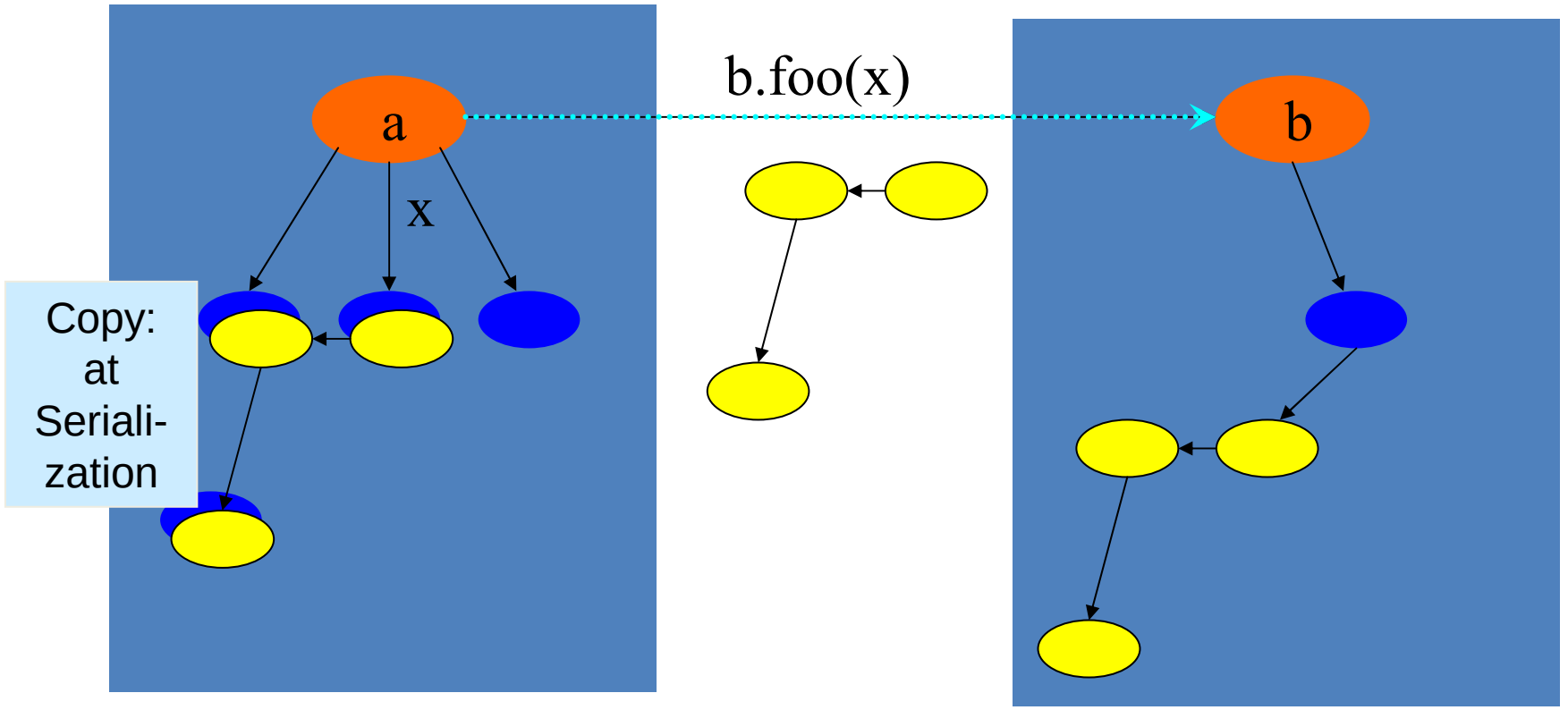
ProActive : Active object

An active object is composed of several objects :

- The object being activated: Active Object (1)
- A set of standard Java objects
- A single thread (2)
- The queue of pending requests (3)



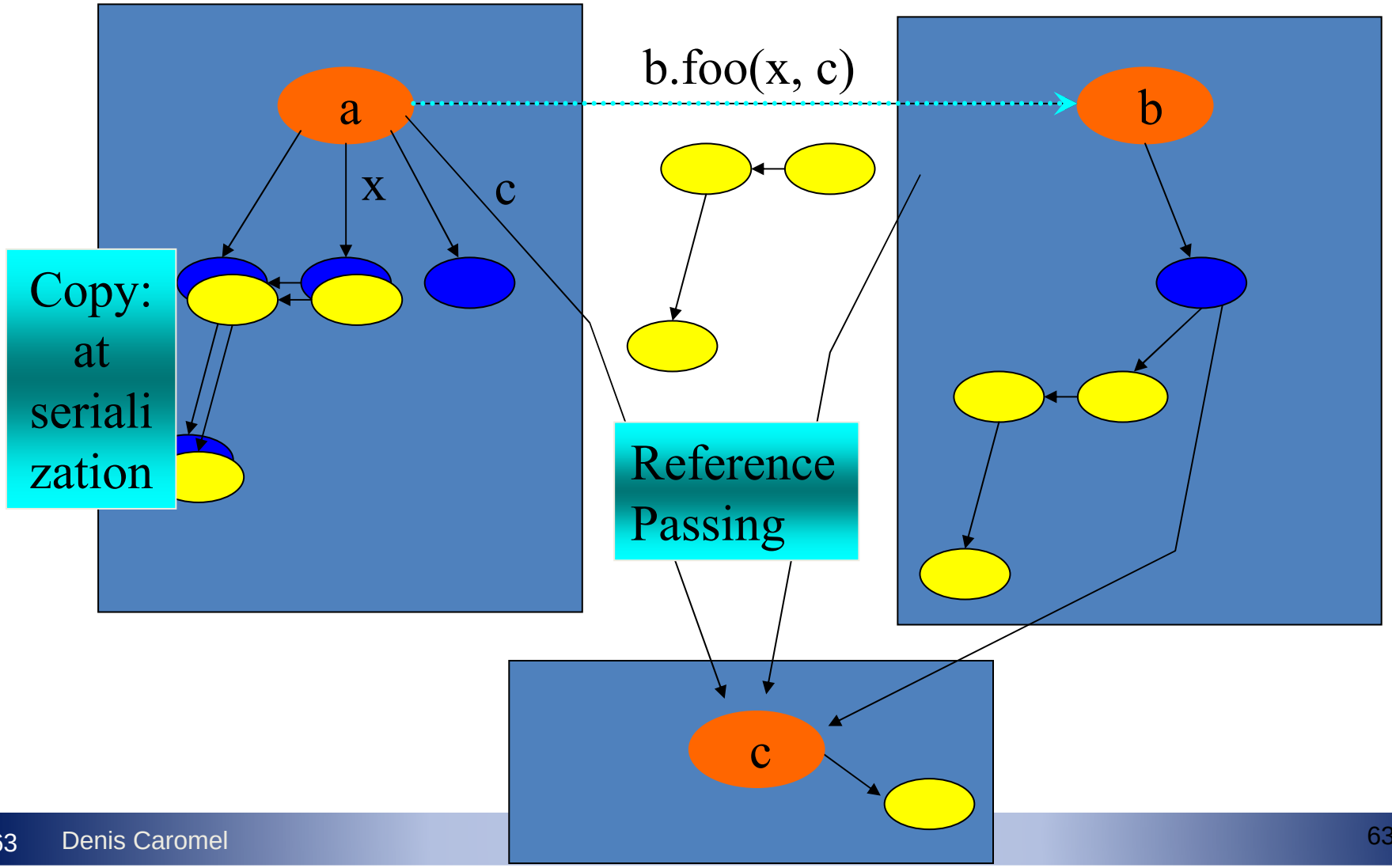
Call between Objects: Parameter passing: Copy of Java Objects



(Deep) Copies evolve independently -- No consistency

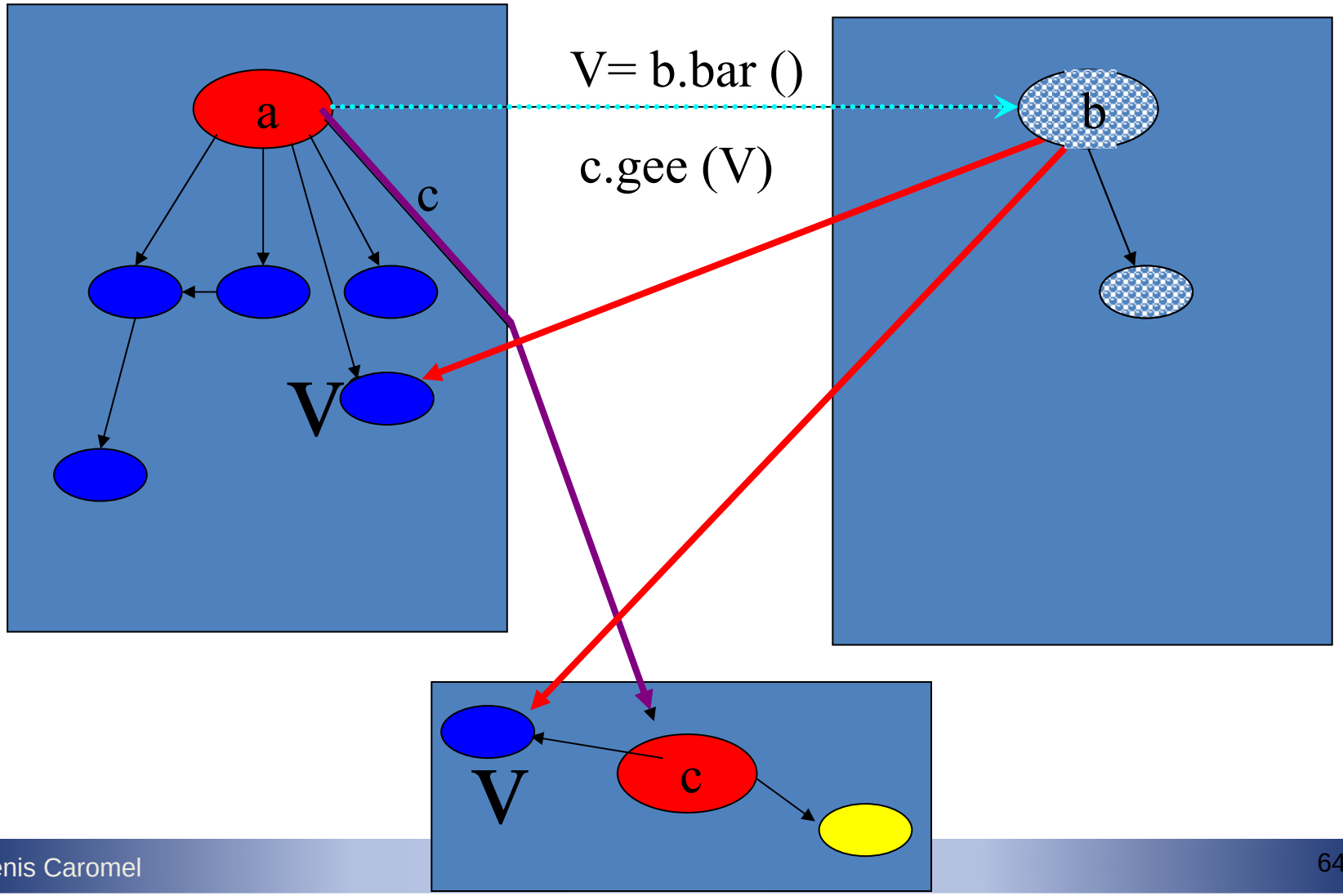
Call between Objects: Parameter Passing: Active Objects

Object passed by Deep Copy - Active Object by Reference



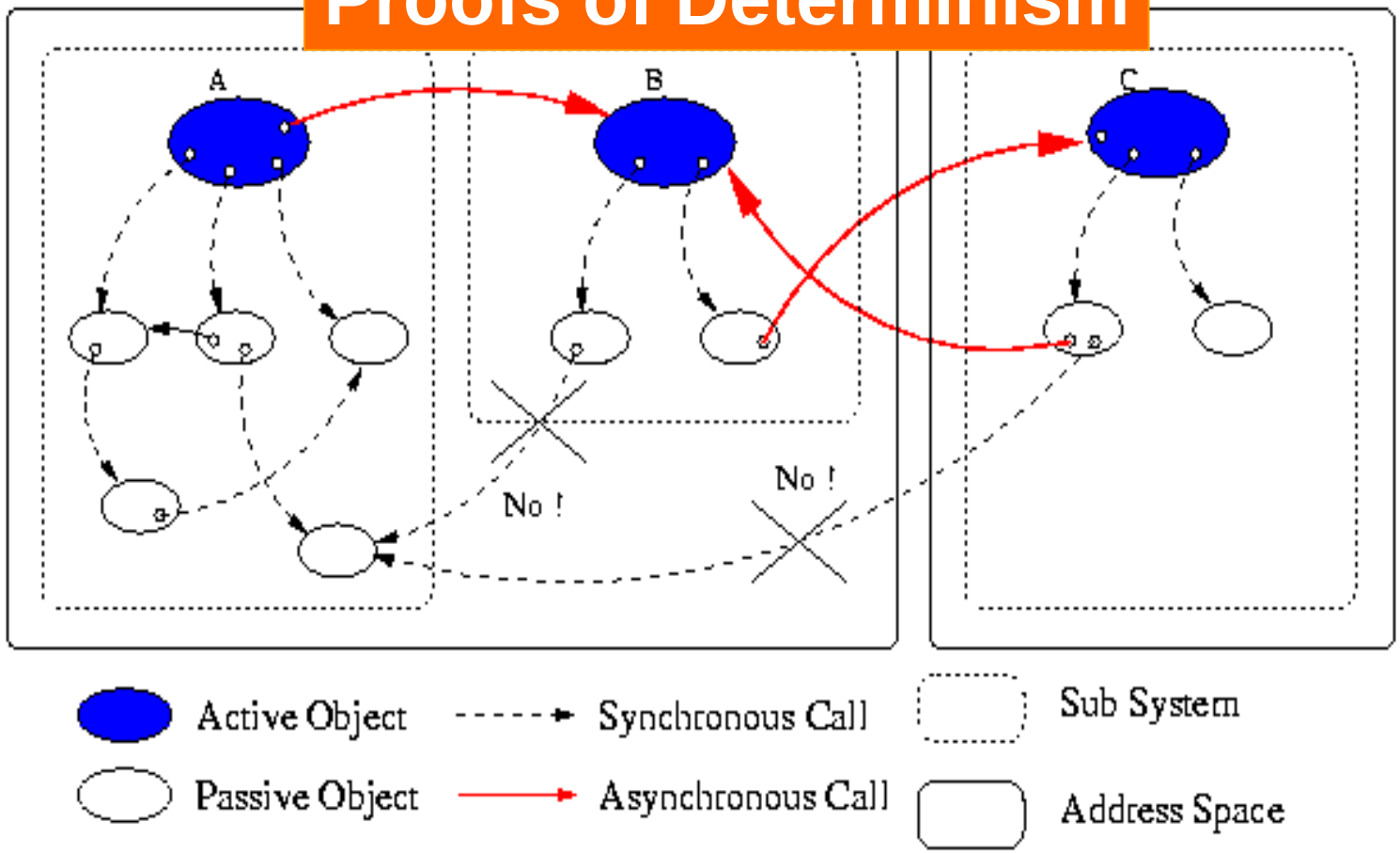
Wait-By-Necessity: First Class Futures

Futures are Global Single-Assignment Variables



Standard system at Runtime: Dynamic Topology, Asynchrony, WbN, ... but no sharing

Proofs of Determinism



Proofs in GREEK

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma')}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P}\ (\text{LOCAL})$$

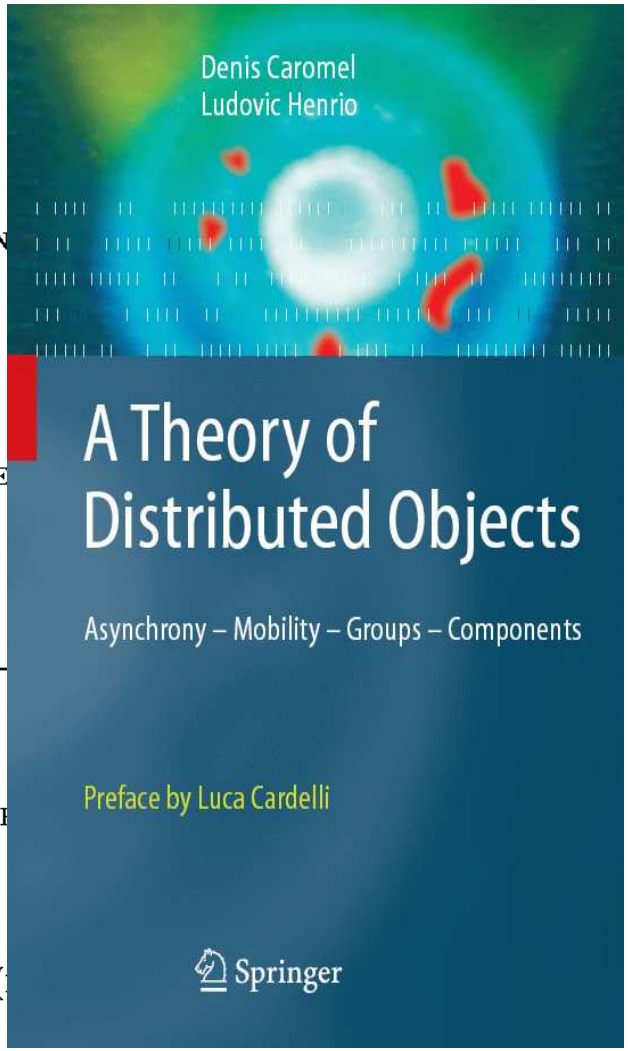
$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = copy(\iota'',\sigma) \qquad Service = (\text{ if } m_j = \emptyset \text{ then } FifoService \text{ else } \iota''.m_j())\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \\ \longrightarrow \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[Service;\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P\end{array}}\ (\text{N}$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \qquad \iota_f \notin dom(\sigma_\alpha) \\ \sigma'_\beta = Copy\&Merge(\sigma_\alpha,\iota' \ ; \ \sigma_\beta,\iota'') \qquad \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}}\ (\text{RE}$$

$$\frac{R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M,\ m \notin R'}{\alpha[\mathcal{R}[Serve(M)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[\iota.m_j(\iota_r) \Uparrow f,\mathcal{R}[[]];\sigma;\iota;F;R' :: R'';f'] \parallel}$$

$$\frac{\iota' \notin dom(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \qquad \sigma' = Copy\&Merge(\sigma,\iota \ ; \ \sigma,\iota')}{\alpha[\iota \Uparrow (f',a);\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P}\ (\text{ENDSE}$$

$$\frac{\sigma_\alpha(\iota) = fut(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \qquad \sigma'_\alpha = Copy\&Merge(\sigma_\beta,\iota_f \ ; \ \sigma_\alpha,\iota)}{\begin{array}{l}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}}\ (\text{}$$

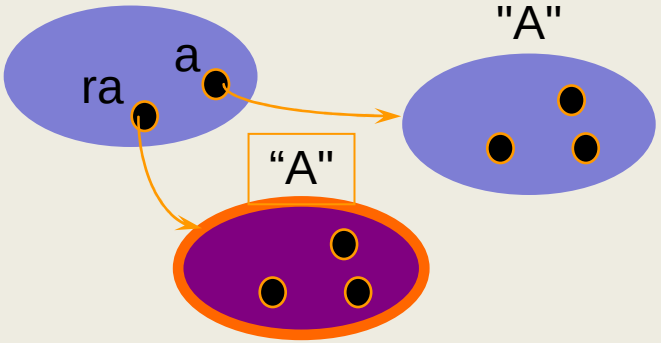


ProActive : Reuse and seamless

Two key features:

Polymorphism between standard and active objects
- Type compatibility for classes (and not only interfaces)
- Needed and done for the future objects also
- Dynamic mechanism (dynamically achieved if needed)

Wait-by-necessity: inter-object synchronization
- Systematic, implicit and transparent futures
- Ease the programming of synchronizations, and the reuse of routines



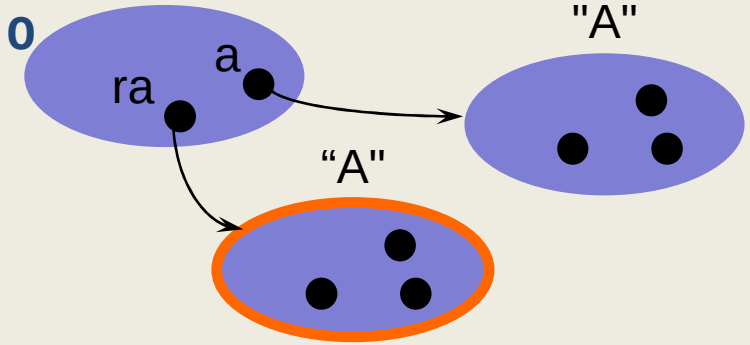

```
foo (A a)
{
  a.g (...);
  v = a.f (...);
  ...
  v.bar (...);
}
```

ProActive : Reuse and seamless

Two key features:

Polymorphism between standard and active objects
- Type compatibility for classes (and not only interfaces)
- Needed and done for the future objects also
- Dynamic mechanism (dynamically achieved if needed)

Wait-by-necessity: inter-object synchronization
- Systematic, implicit and transparent futures
- Ease the programming of synchronizations, and the reuse of routines




```
foo (A a)
{
  a.g (...);
  v = a.f (...);
  ...
  v.bar (...);
}
```

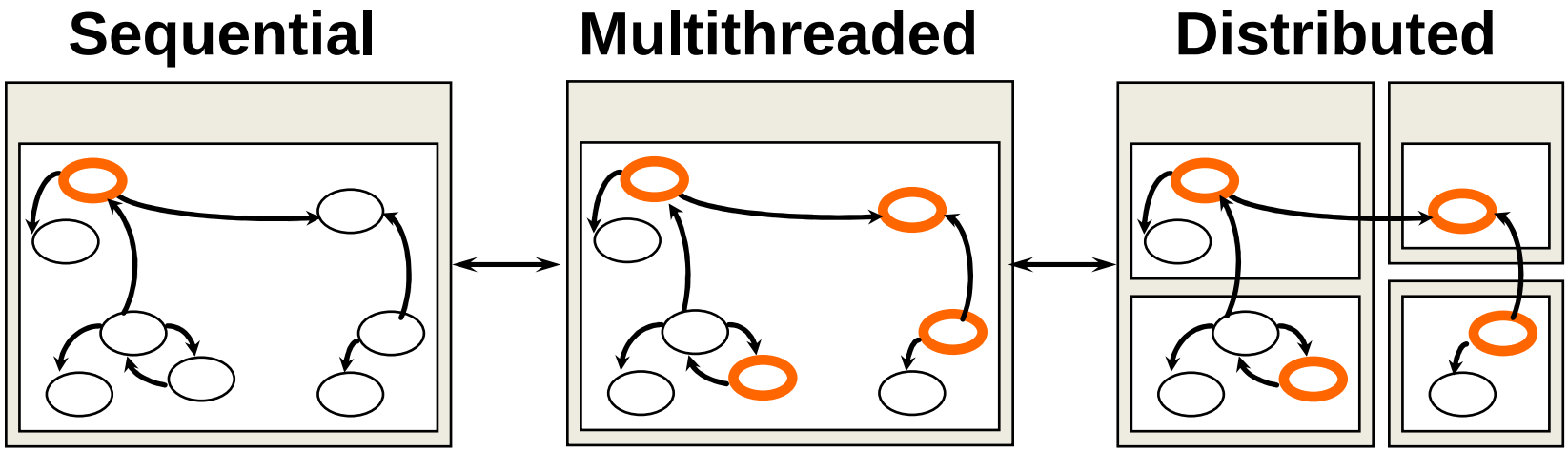
O.foo(a) :
a.g() and a.f() are « local »

O.foo(ra):
a.g() and a.f()are «remote + Async.»

Intra Active Object Synchronizations

ProActive:
Inter- to Intra- Synchronization

Inter-Synchro: mainly Data-Flow



Synchronizations do not dependent upon

the physical location (mapping of activities)

ProActive : Intra-object synchronization

Explicit control:

Library of service routines:

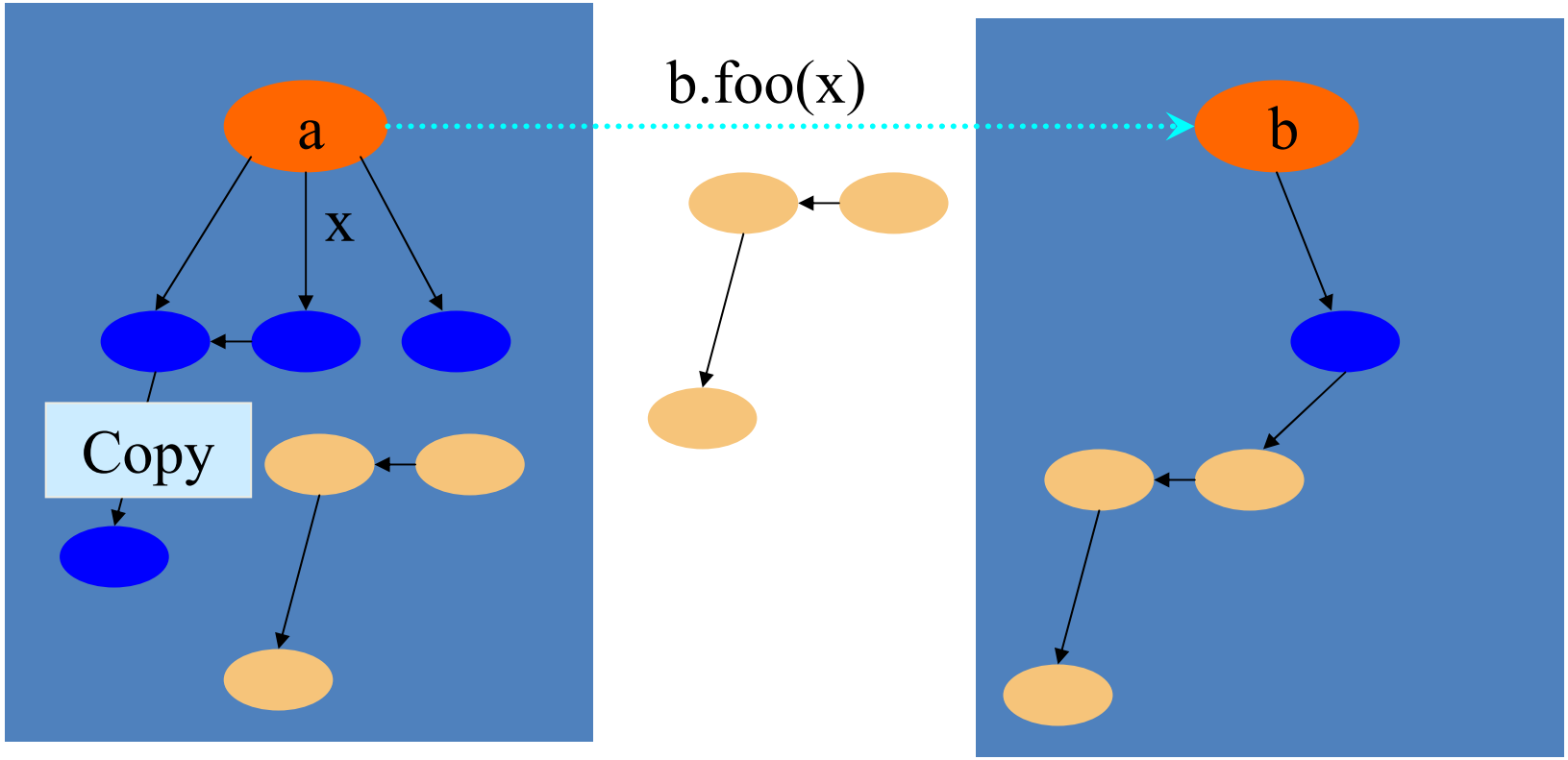
- Non-blocking services,...
  - `serveOldest ();`
  - `serveOldest (f);`
- Blocking services, timed, etc.
  - `serveOldestBl ();`
  - `serveOldestTm (ms);`
- Waiting primitives
  - `waitARequest();`
  - `etc.`

```
class BoundedBuffer extends FixedBuffer
                implements RunActive {

    // Programming Non FIFO behavior
  runActivity (ExplicitBody myBody) {
     while (...) {
       if (this.isFull())
            serveOldest("get");
       else if (this.isEmpty())
            serveOldest ("put");
       else serveOldest ();

       // Non-active wait
       waitArequest ();
   }
}}
```

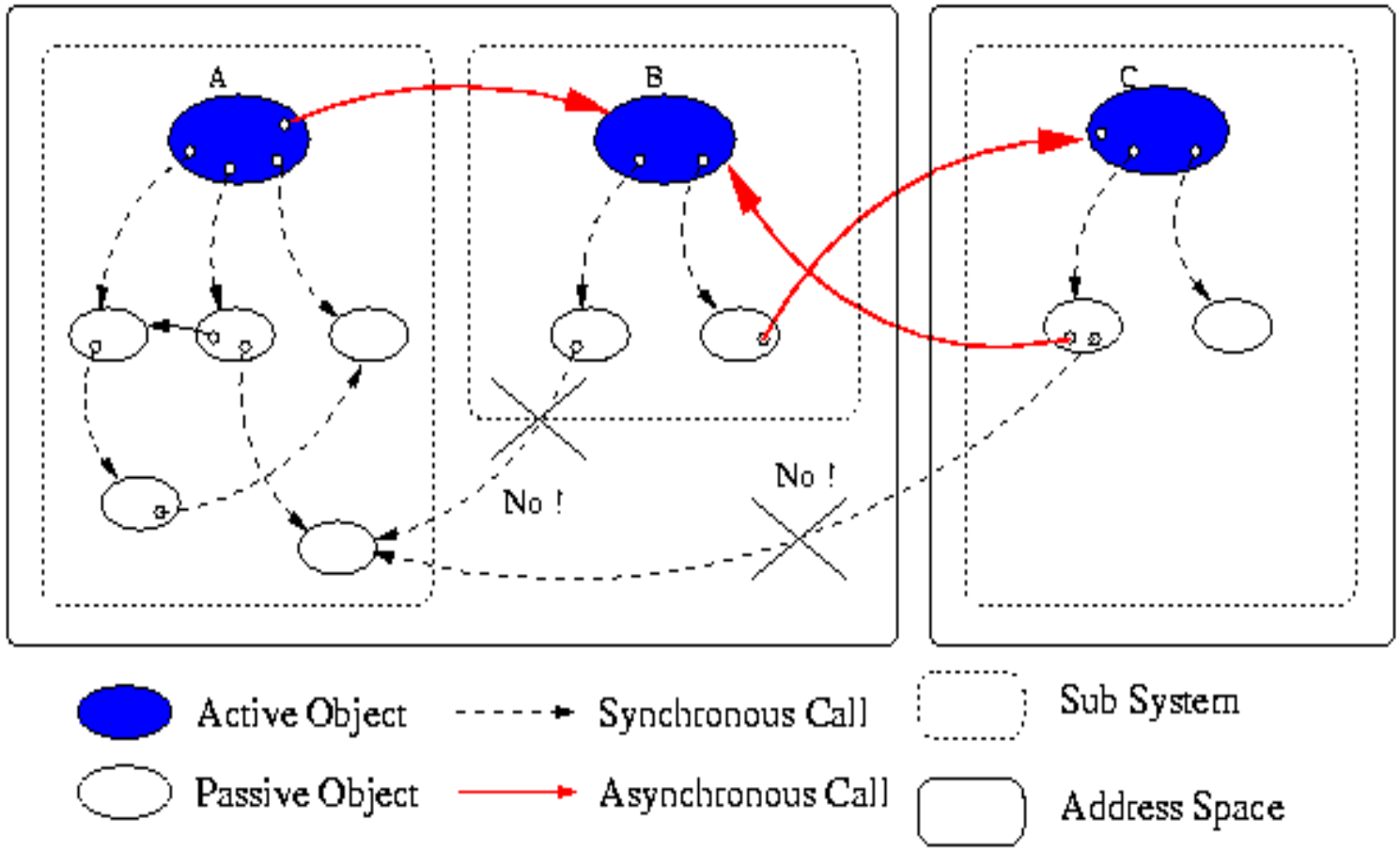
Implicit (declarative) control: library classes

```
e.g. : Blocking Condition Abstraction for concurrency control:
                doNotServe ("put", "isFull");
```

Optimization: SharedOnRead
Call between Objects



Standard system at Runtime:
No Sharing



Optimization: SharedOnRead

Share a passive object if same address space

Automatic copy if required

Implementation: Use of the Mop

Help from the user:

- Point out functions that make read access
- Write access by default

SharedOnRead (2)

Algorithm

If a SharedOnRead Object is send during a method call:

If within the same VM:

send a reference instead of the real object

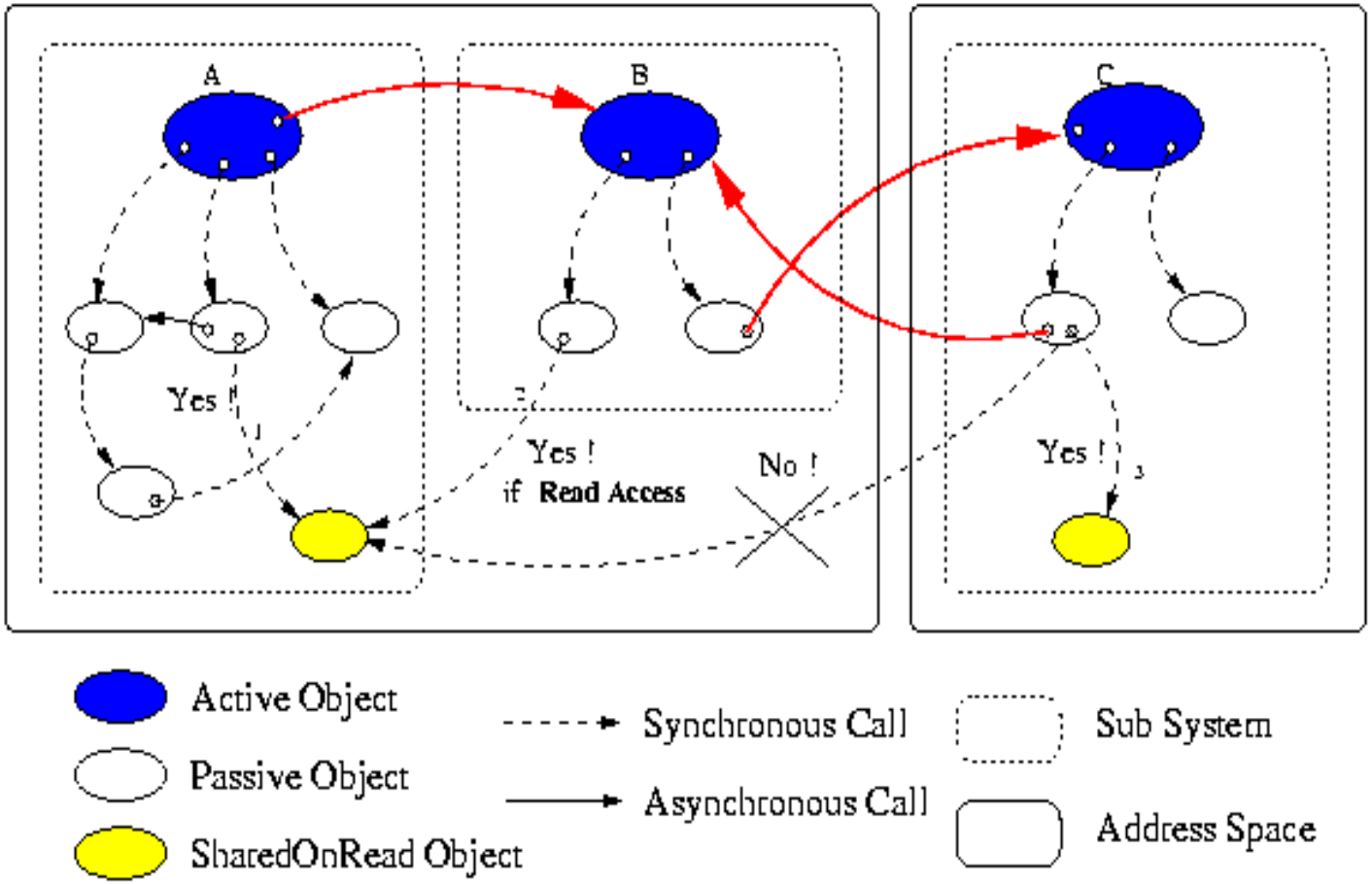
(reference counter)+1

After that:

Read access can be freely achieved

Write access triggers an object copy

SharedOnRead (3)



Adaptive Feature:
Multi-transports layer
RMI, RMI-ssh, …, Ibis, HTTP XML, ...

Adaptive choice of transport layer between:

- » RMI
- » ssh/RMI

Also available with static configuration:

- » Ibis (TCP, Myrinet, etc.)
- » HTTP
- » … ssh/HTTP

Short Term Perspective:

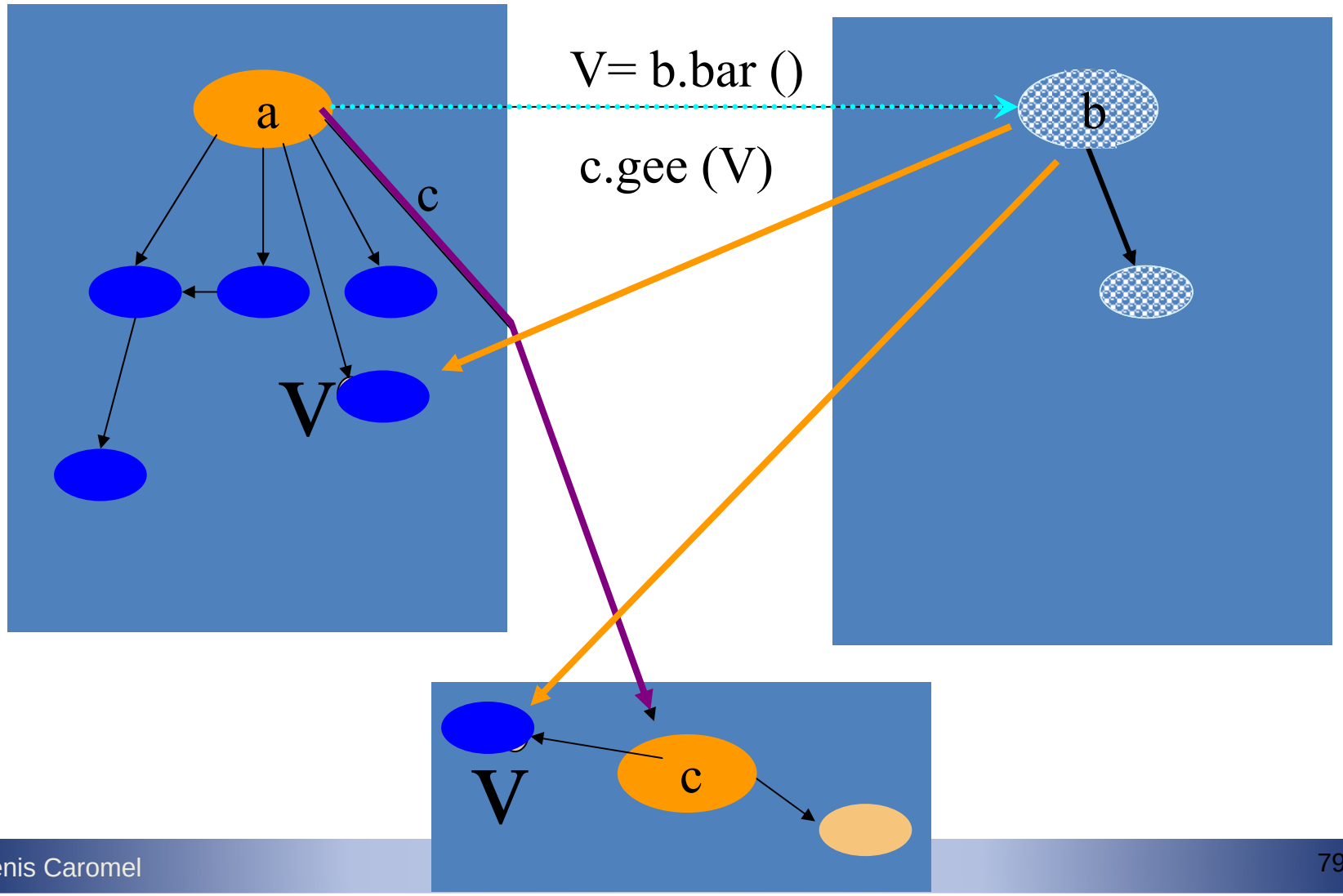
Fully Adaptive Choice between all transports

First-Class Futures

Update

Wait-By-Necessity: First Class Futures

Futures are Global Single-Assignment Variables



Future update strategies

No partial replies and requests:

No passing of futures between activities, more deadlocks

Eager strategies: as soon as a future is computed

Forward-based:

- Each activity is responsible for updating the values of futures it has forwarded

Message-based:

- Each forwarding of future generates a message sent to the computing activity
- The computing activity is responsible for sending the value to all

Mixed strategy:

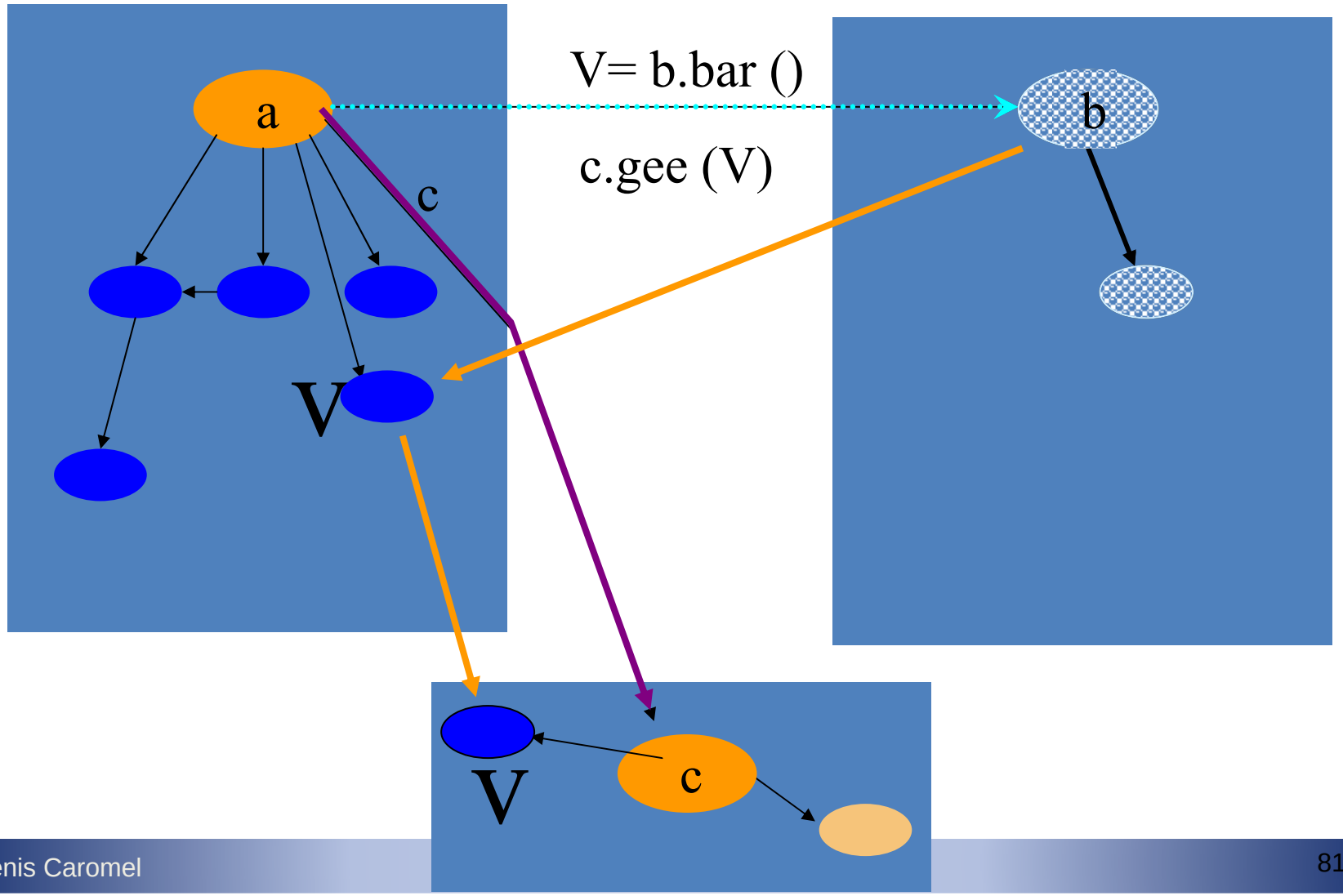
Futures update any time between future computation and WbN

Lazy strategy:

On demand, only when the value of the future is needed (WbN on it)

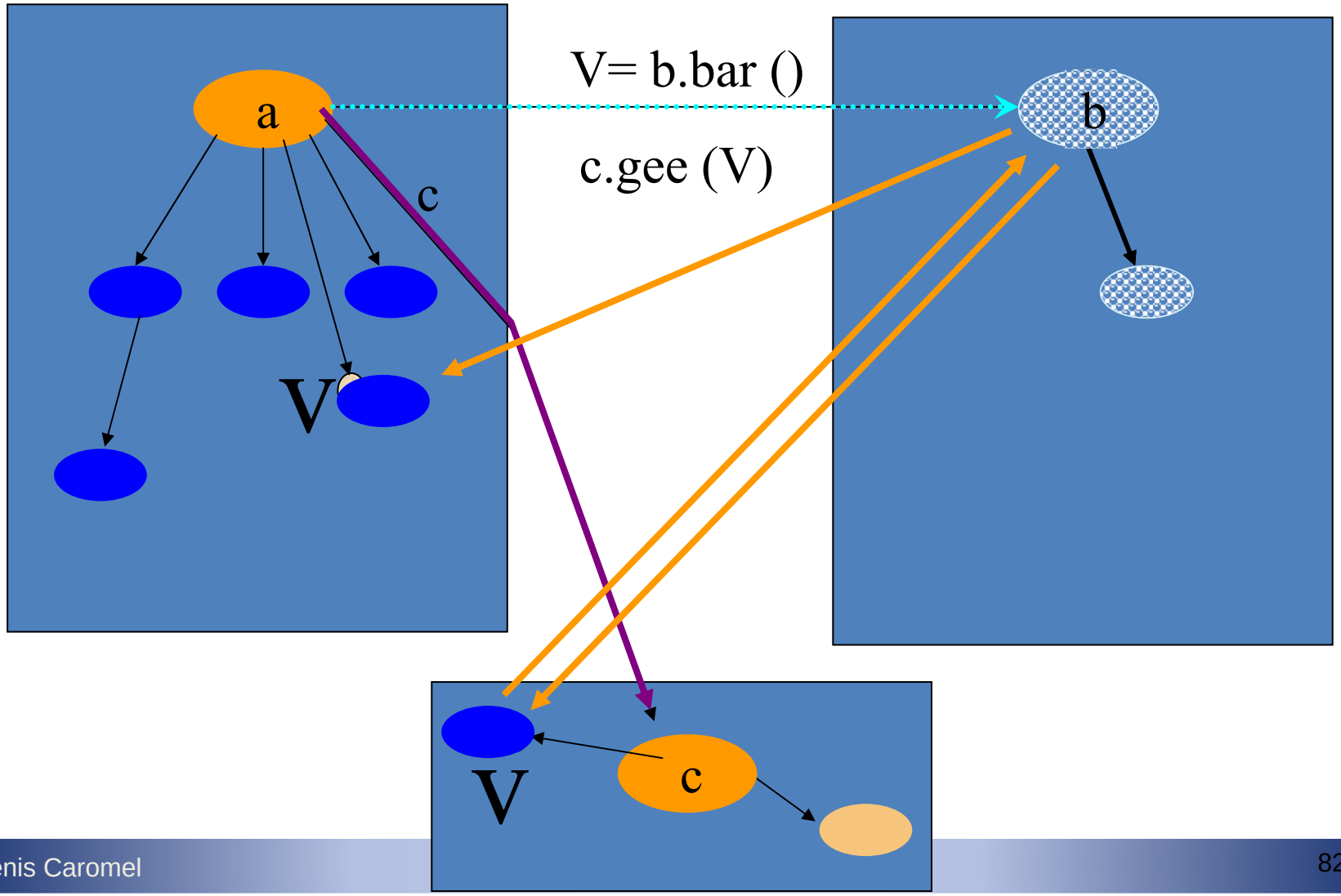
Wait-By-Necessity: Eager Forward Based

AO forwarding a future: will have to forward its value



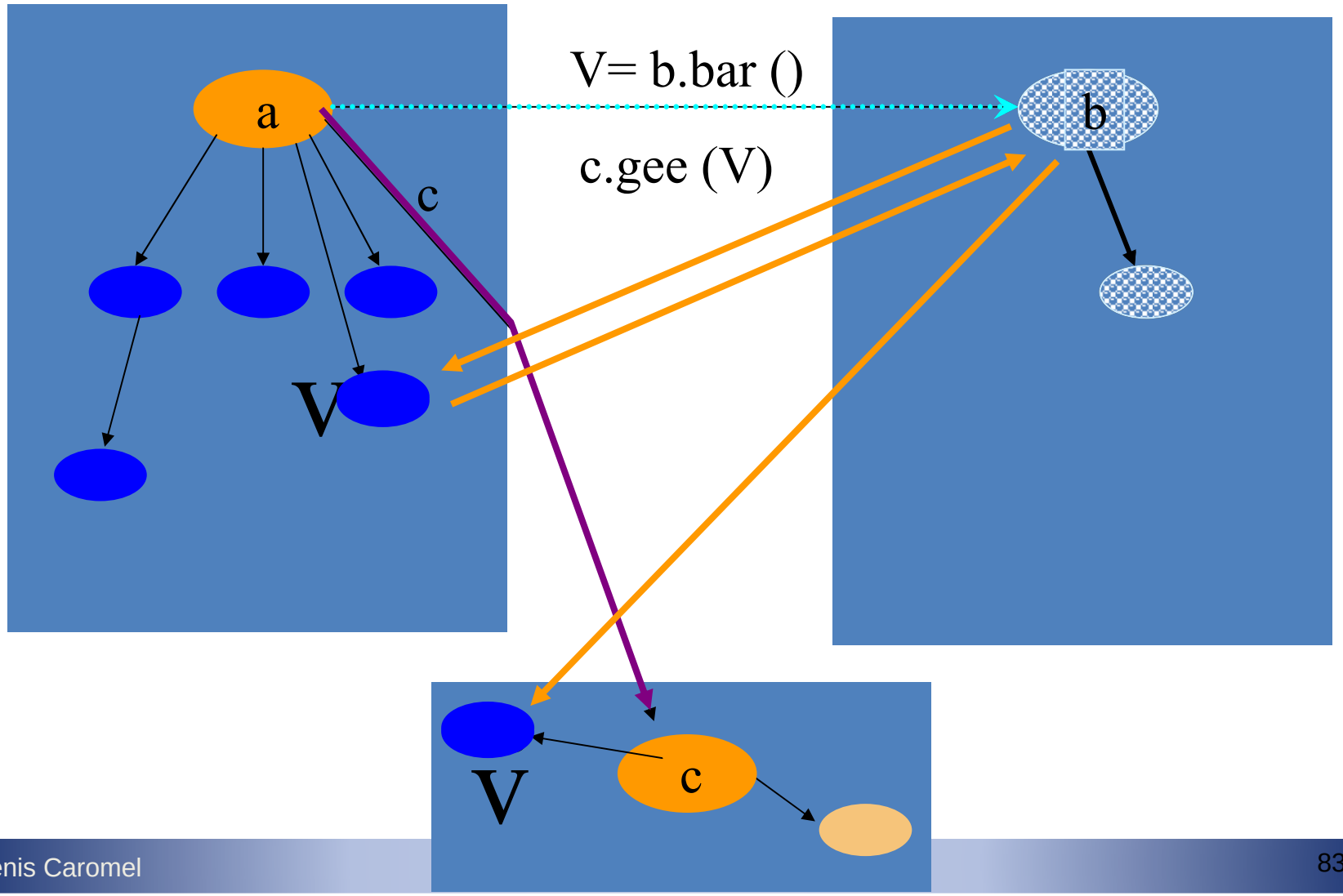
Wait-By-Necessity: Eager Message Based

AO receiving a future: send a message



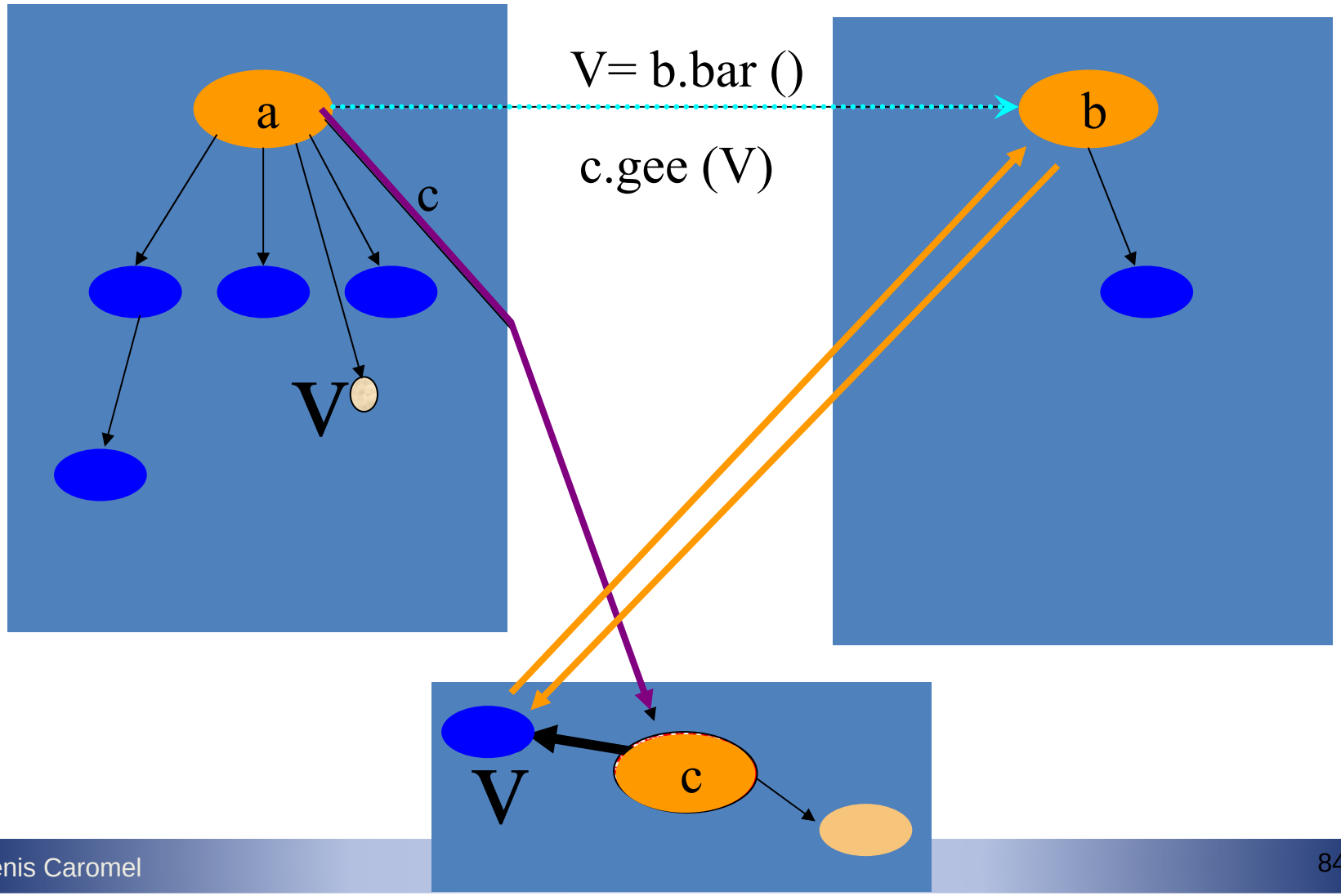
Wait-By-Necessity: Eager Message Based(2)

AO forwarding a future: send a message



Wait-By-Necessity: Lazy Strategy

An Active Object requests a Future Value when needed



TYPED

ASYNCHRONOUS

GROUPS

1.2. Collective Communications: Groups

- Manipulate groups of Active Objects, in a simple and typed manner:
  - ➡ Typed and polymorphic Groups of active and remote objects
  - ➡ Dynamic generation of group of results
  - ➡ Language centric, Dot notation
- Be able to express high-level collective communications (like in MPI):
  - broadcast,
  - scatter, gather,
  - all to all

```
A ag=(A)ProActiveGroup.newActiveGroup(«A»,{{p1},...},{Nodes,..});
V v = ag.foo(param);
v.bar();
```

Group Communications

Method Call on a Typed Group

Based on the ProActive communication mechanism

- Replication of N ‘ single ’ communications
- Parallel calls within a group (latency hiding)
- Preserve the « rendez-vous »

Groups: Analysis of Related Works

3 projects close to ProActive Group Communications:

JGroups

- Centered on low layers of communication: messages rather than RMI
- Socket programming style

Object Group Service

- CORBA focused
- Group communication return only one result (by default)

Group Method Invocation

- Group members have to implement and extend a class and an interface

Groups: Analysis of Related Works

Lack of transparency:

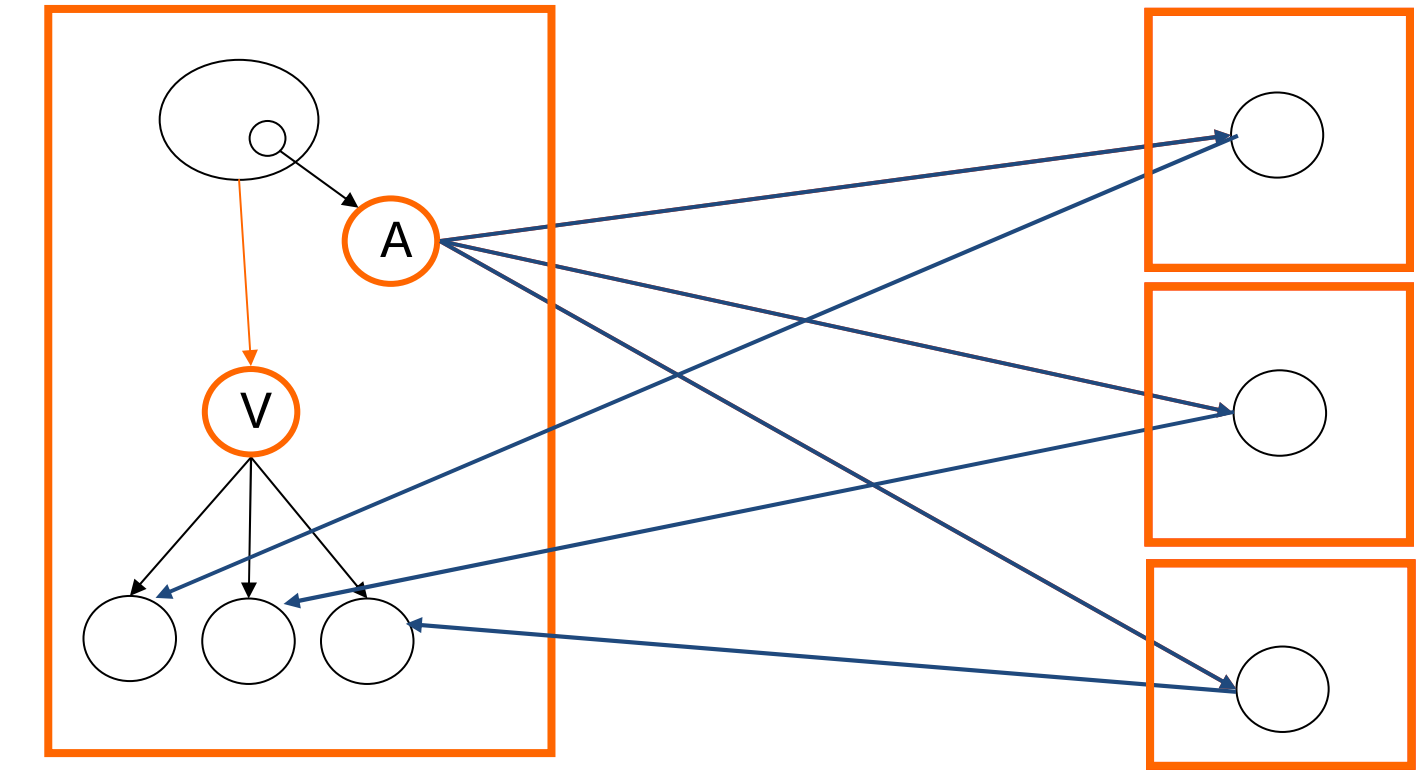
- Specific interface for functional calls
- Inheritance from specific classes and interfaces

Sometimes too application-specific or domain-specific

ProActive Groups address those problems !

Creating AO and Groups

- `A ag = newActiveGroup (“A”, […], VirtualNode)`
- `V v = ag.foo(param);`
- `…`
- `v.bar(); //Wait-by-necessity`



Typed Group Java or Active Object

Group, Type, and Asynchrony are crucial for Cpt. and GRID

Collective Operations : Example

```
class A {…
 V foo(P p){...}
}
class B extends A
   { ...}
A a1=PA.newAct(A,);
A a2=PA.newAct(A,);
B b1=PA.newAct(B,);
```

```
A a3=PA.newAct(A,);


Group ga =
   ProActiveGroup.g
   etGroup(ag);
ga.add(a3
```

```
// Build a group of STANDARD OBJECTS « A »
A agS = (A)ProActiveGroup.newGroup("A",{…})
V v = ag.foo(param); // foo(param) invoked
                                // on each member
// A group v of result of type V is created
```

```
// Build a group of ACTIVE OBJECTS « A »
A ag = (A) newActiveGroup("A",{…}, Nodes)
V v=ag.foo(param);//invoked // on each member
```

```
v.bar();
```

```
ProActiveGroup.waitForAll
   (v); //bloking -> all
v.bar();
```

```
V vi =
ProActiveGroup.getOne(v);
      //bloking -> on
vi.bar(); //a single call
```

Group as Result of Group Communication

Ranking Property:

Dynamically built and updated

```
B groupB = groupA.foo();
```

Ranking property: order of group member = order of reply origin

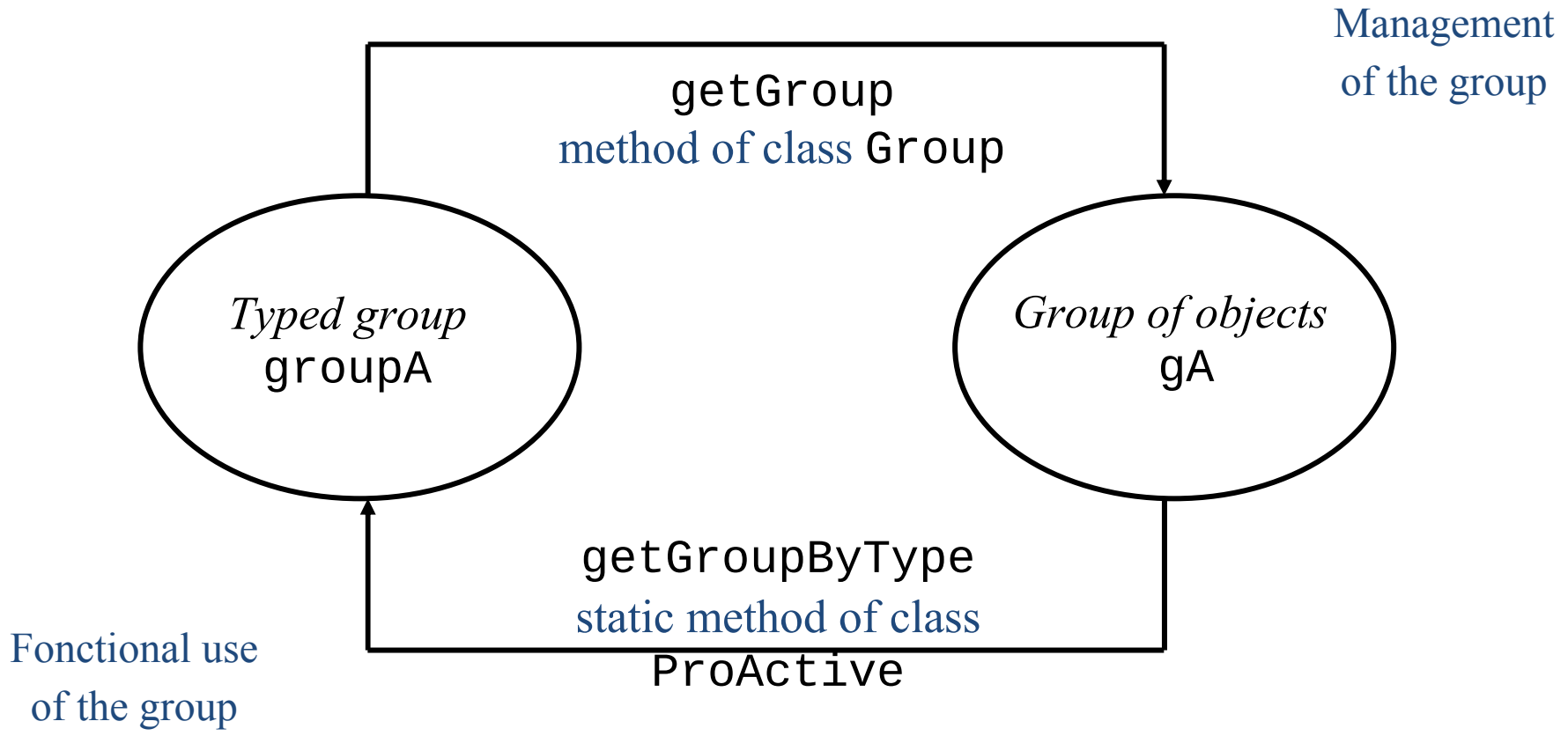
Explicit Group Synchronization Primitive:

```
ProActiveGroup.waitOne(groupB);
ProActiveGroup.waitAll(groupB);
... waitForAll, ...
```

Predicates:

```
noneArrived, kArrived, allArrived, ...
```

Two Representation Scheme



Two Representations (2)

Management operations `add`, `remove`, `size`, ...

2 possibility : static methods, second representation

2 representations of a same group : Typed Group / Group of objects

ability to switch between those 2 representations

```
Group gA = ProActiveGroup.getGroup(groupA);
gA.add(new A());
gA.add(new B()); //B herits from A
A groupA = (A) gA.getGroupeByType();
```

Broadcast or scatter

Broadcast is the default behavior

Scatter is also possible

- Scatter uses a group as parameter
- Distribution relies on rank

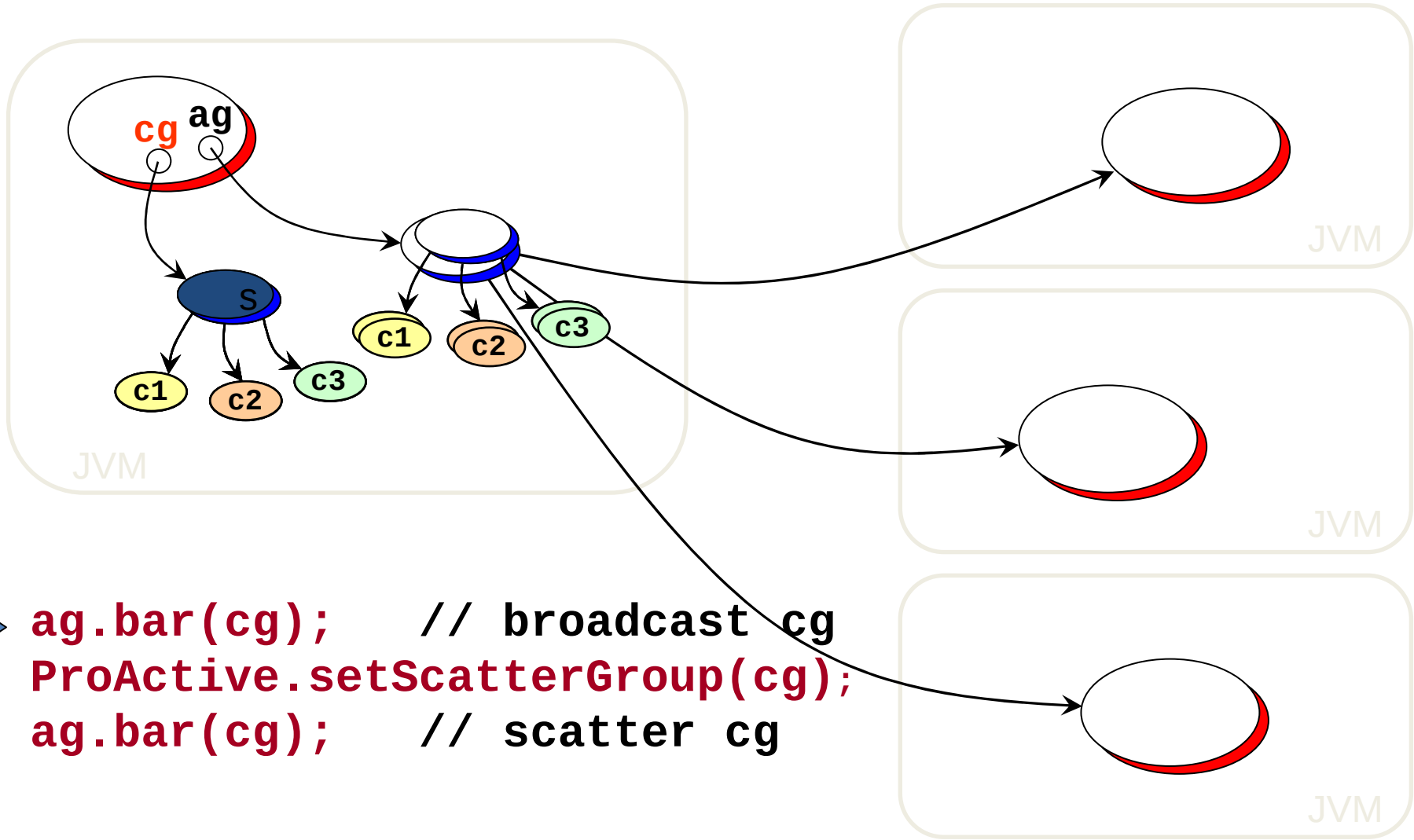
```
ag.bar(cg, dg); // broadcast cg and dg
ProActive.setScatterGroup(cg);
ag.bar(cg, dg); // scatter cg, still broadcast dg
```

One call can both **scatter** and **broadcast**

in the same group call

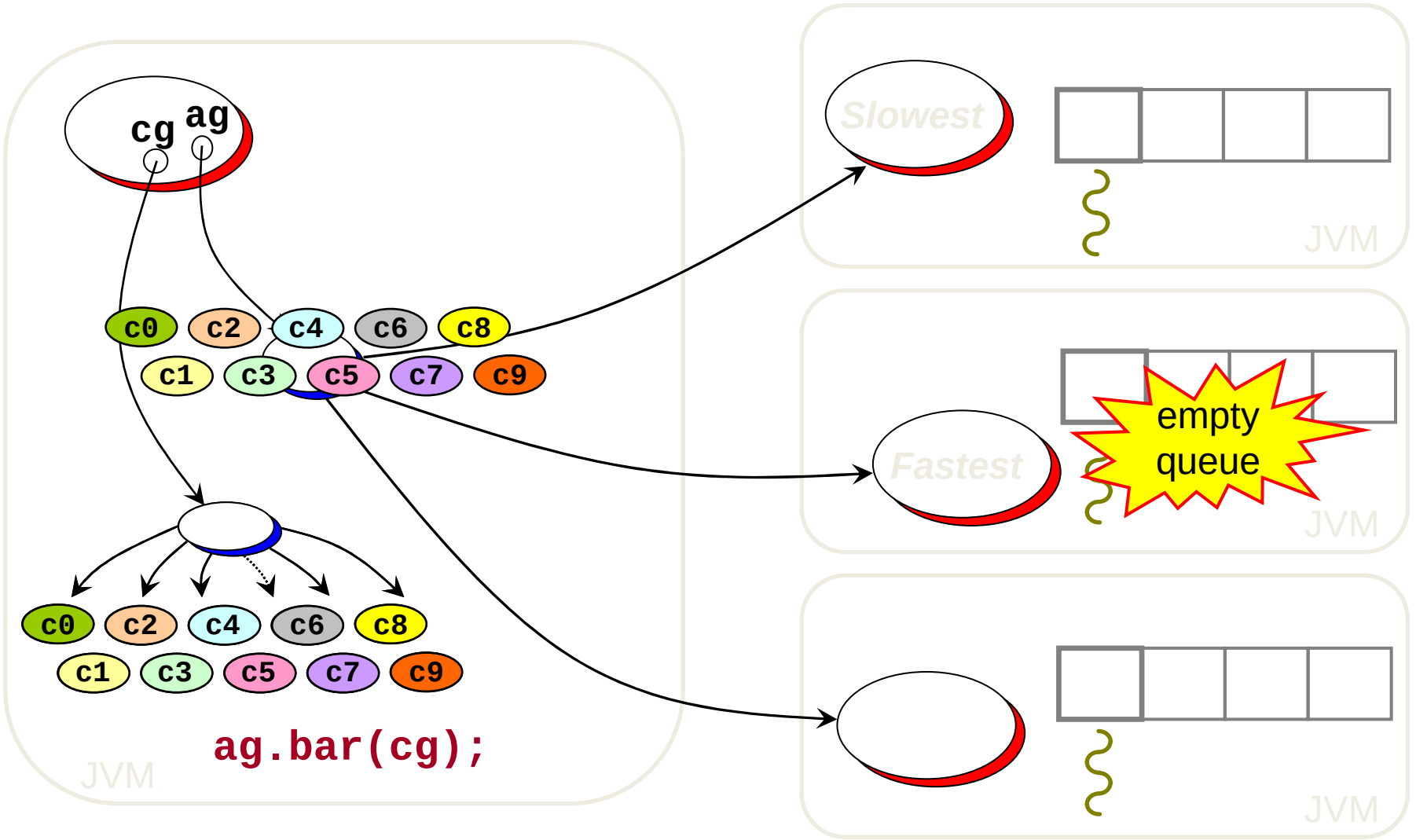
Broadcast and Scatter

Broadcast is the default behavior
Use a group as parameter, Scattered depends on rankings

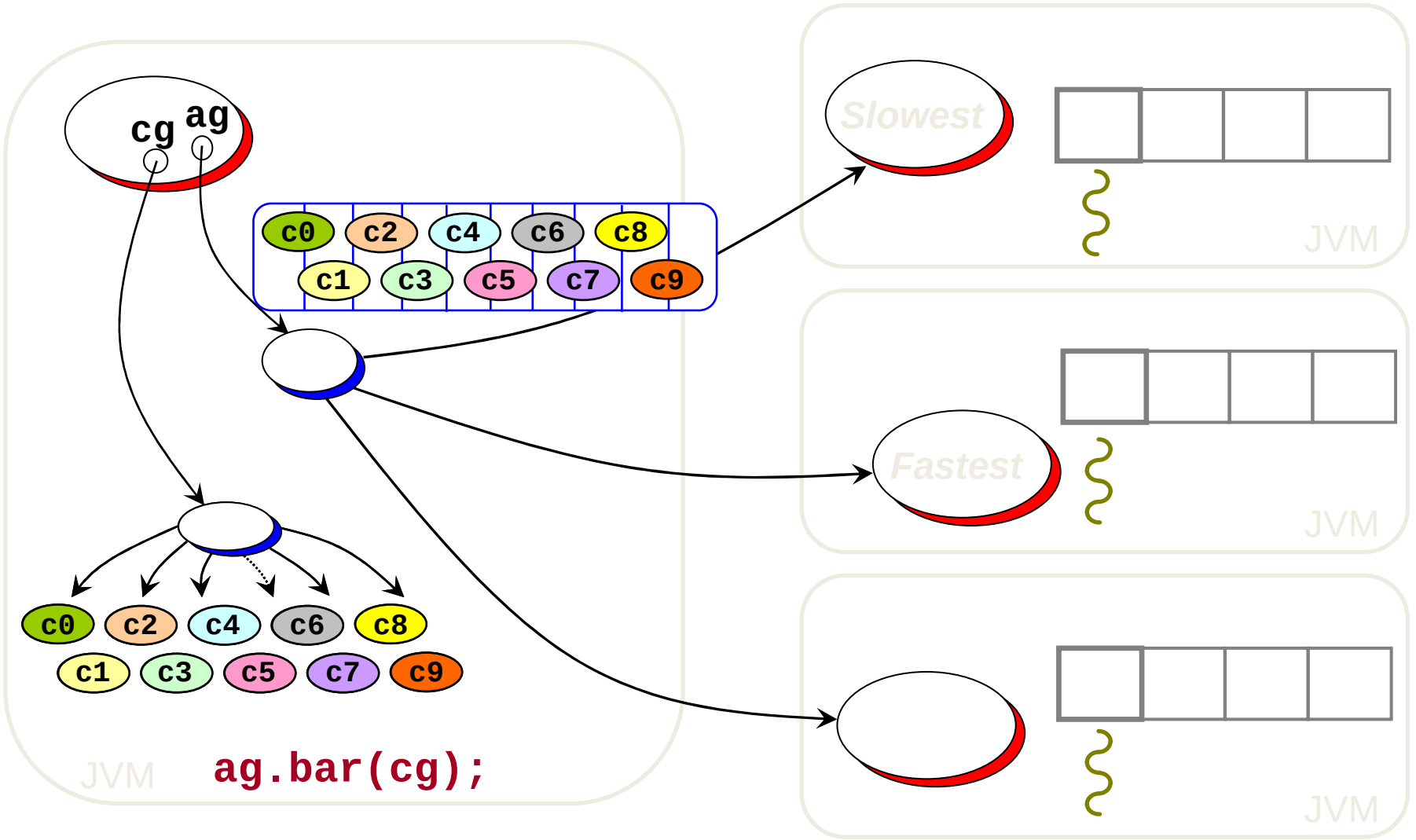



```
ag.bar(cg);    // broadcast cg
ProActive.setScatterGroup(cg);
ag.bar(cg);    // scatter cg
```

Static Dispatch Group



Dynamic Dispatch Group

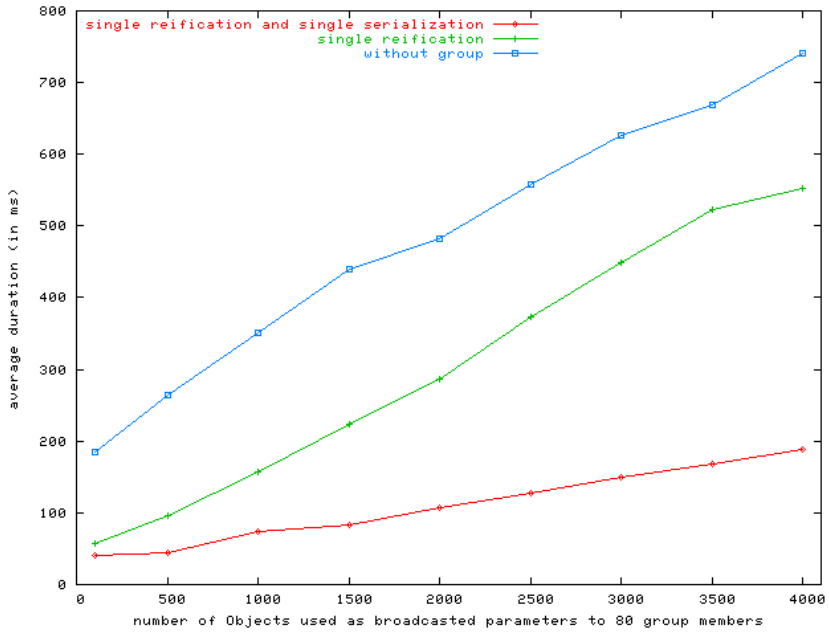


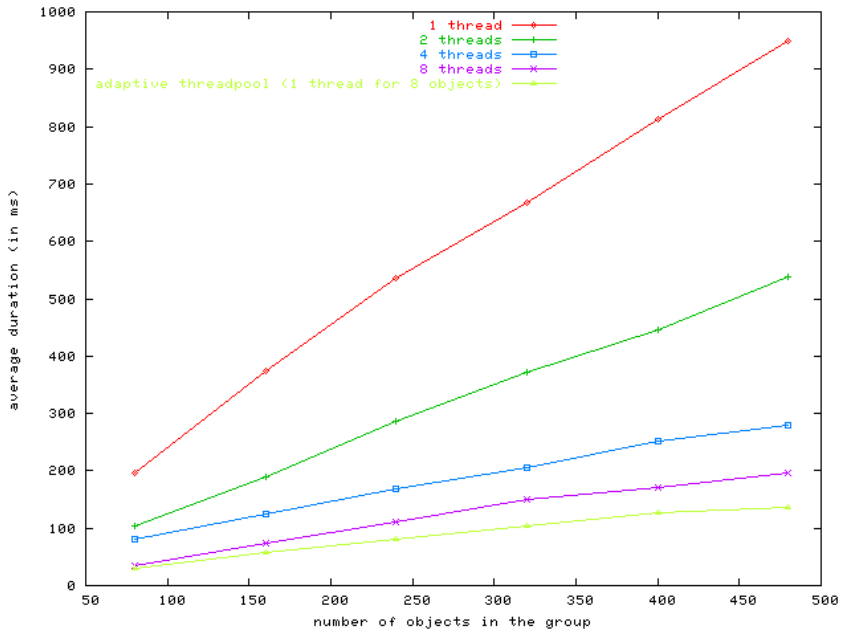
Performances and Optimizations

Common operations

- Single reification
- Single serialization

Parallel calls (adaptive thread pool)





Handling Group Failures (1)

Using Java exceptions

Common way to express failure in Java

A Group may produce N exceptions during one Group Com.

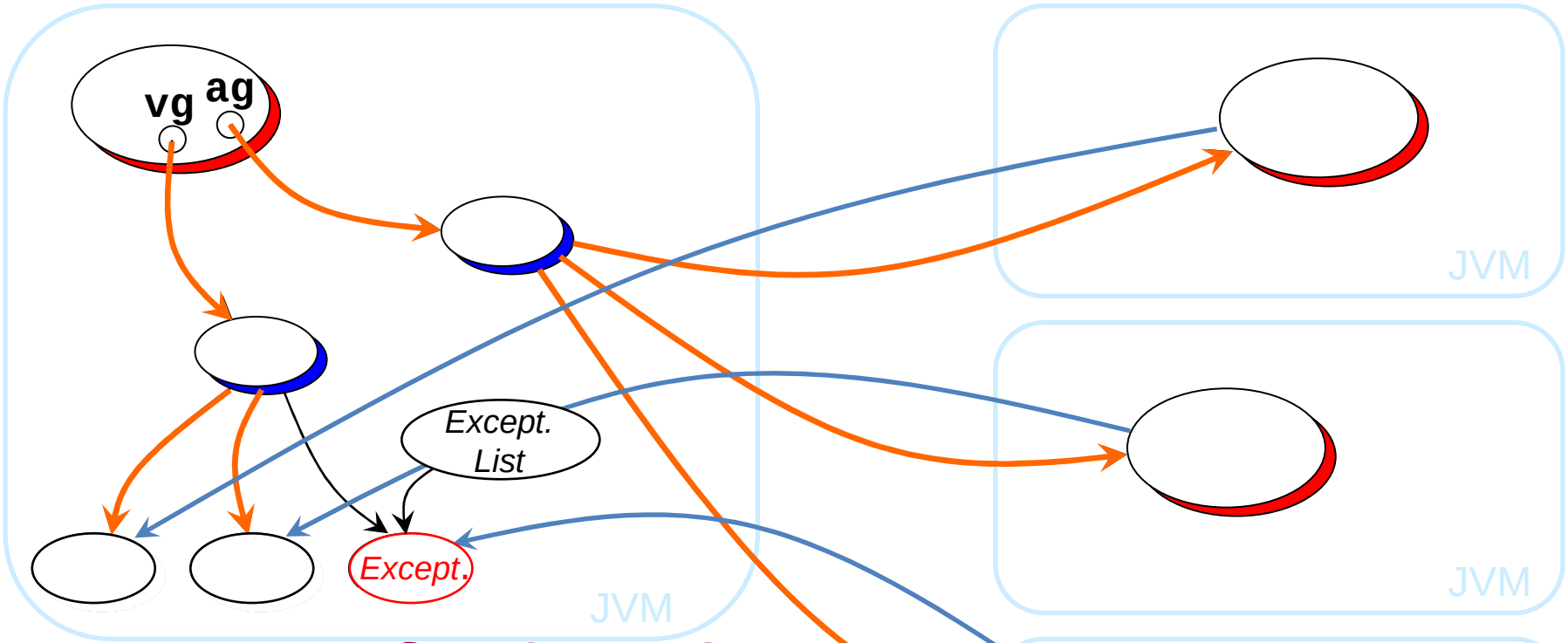
Result Group is used to store exceptions

To get them all at once:

```
E.L. el = group.GetExceptionList();
```

`ExceptionList`: an exception that contains all raised exceptions + a reference to the Group that produced them

Handling Group Failures (2)




```
V vg = ag.foo (param);
Group groupV = PAG.getGroup(vg);
el = groupV.getExceptionList();
...
vg.gee();
```

Active and Hierarchical Groups

Active Group:

a group being a (Remote) Active Object

A group becomes remotely accessible
Multiple coherent view of a single Group from several JVMs
Modifiable service policy

Hierarchical group

A group of groups
Dynamically built
Achieves scalability

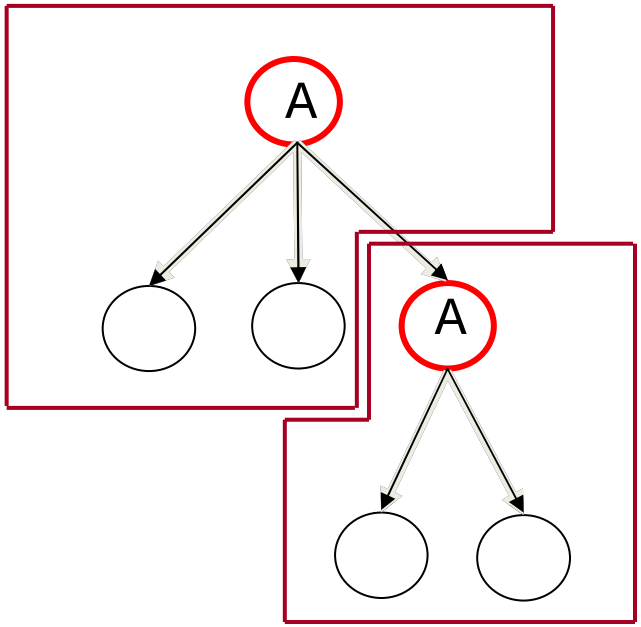
Hierarchical Groups

Add a typed group into a typed group

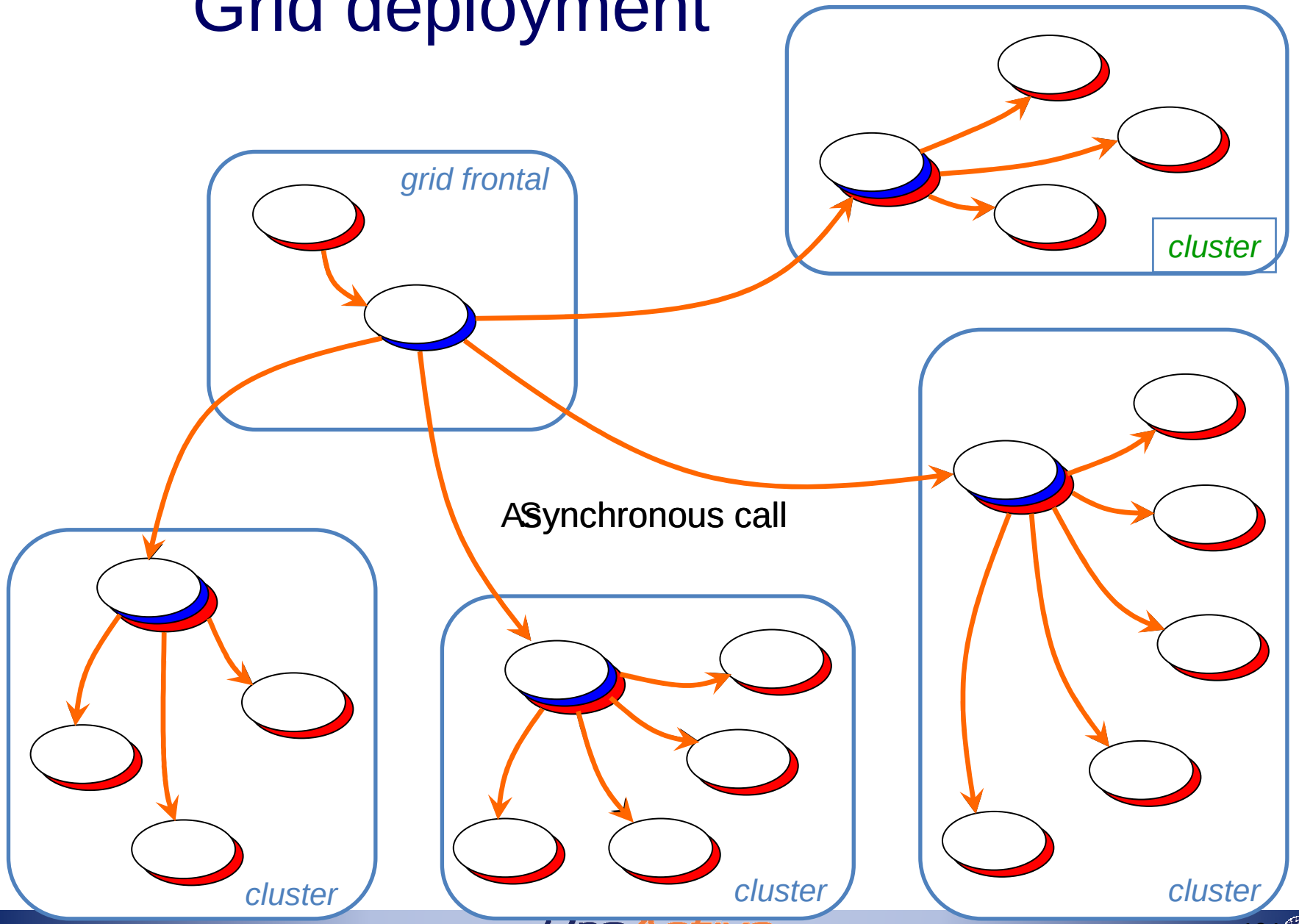
vs. Adding an element in a group

Scalability of groups

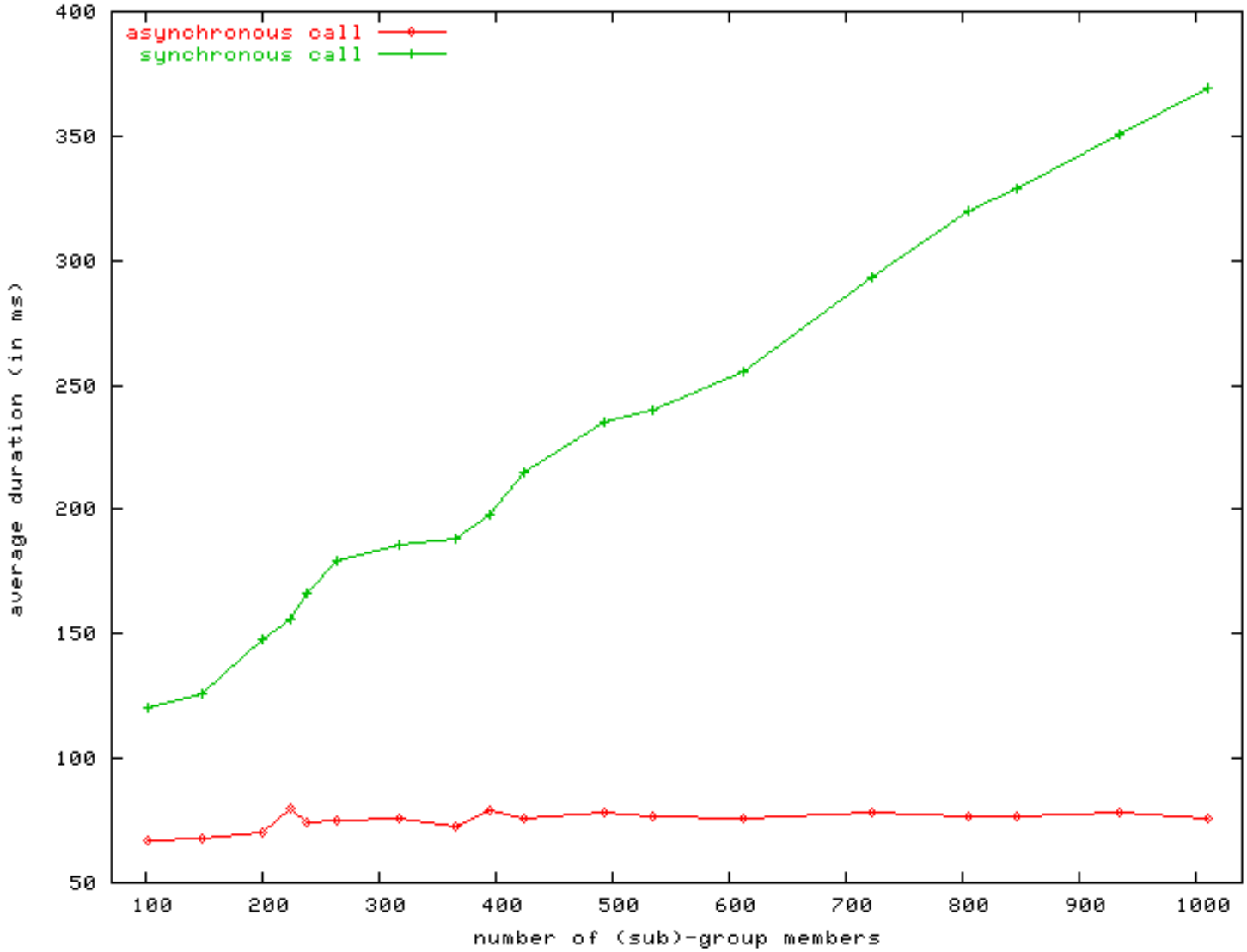
Can match the network topology



Grid deployment



Grid Group Calls on a 1000 CPU at once



Grid'5000

Implementation

MOP generates Stub

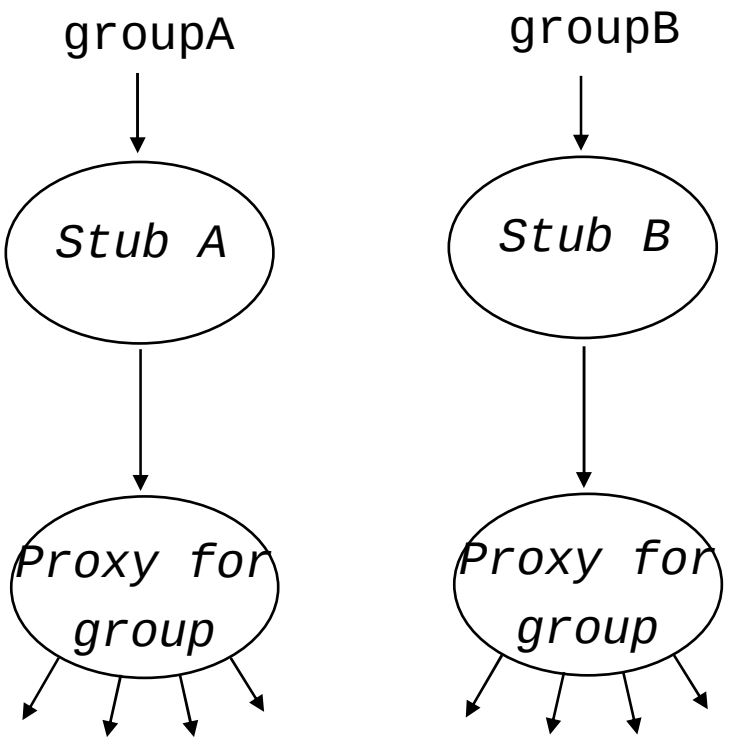
Stub inherits from the class of object

Stub connects a proxy

special proxy for group

result is stub+proxy

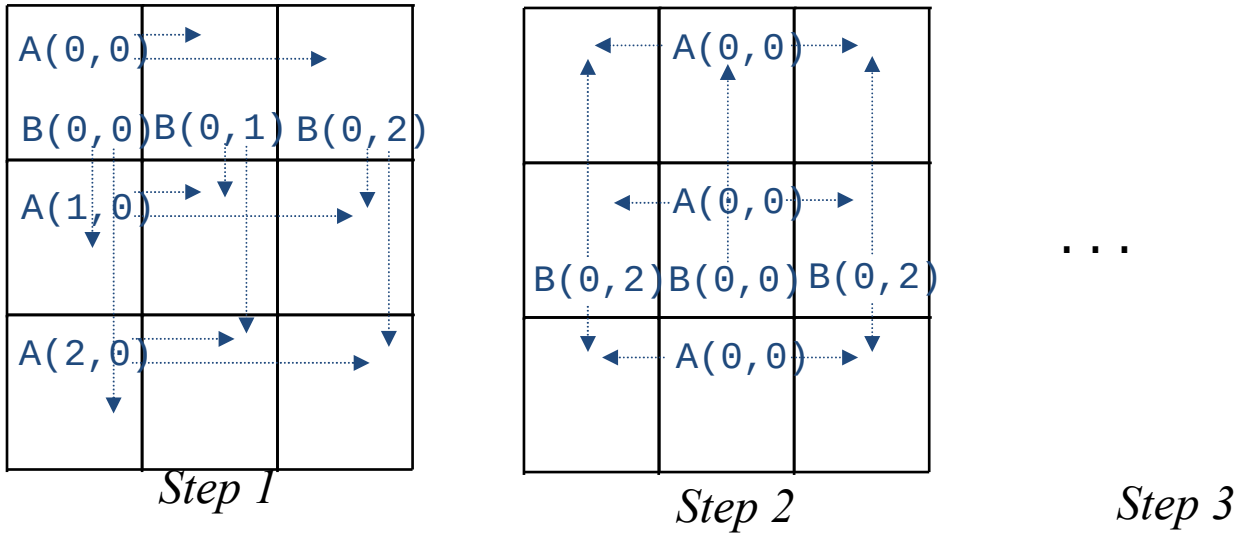
```
B groupB =groupA.foo();
```



Example : Matrix multiplication

Matrix code : Broadcast to Broadcast approach

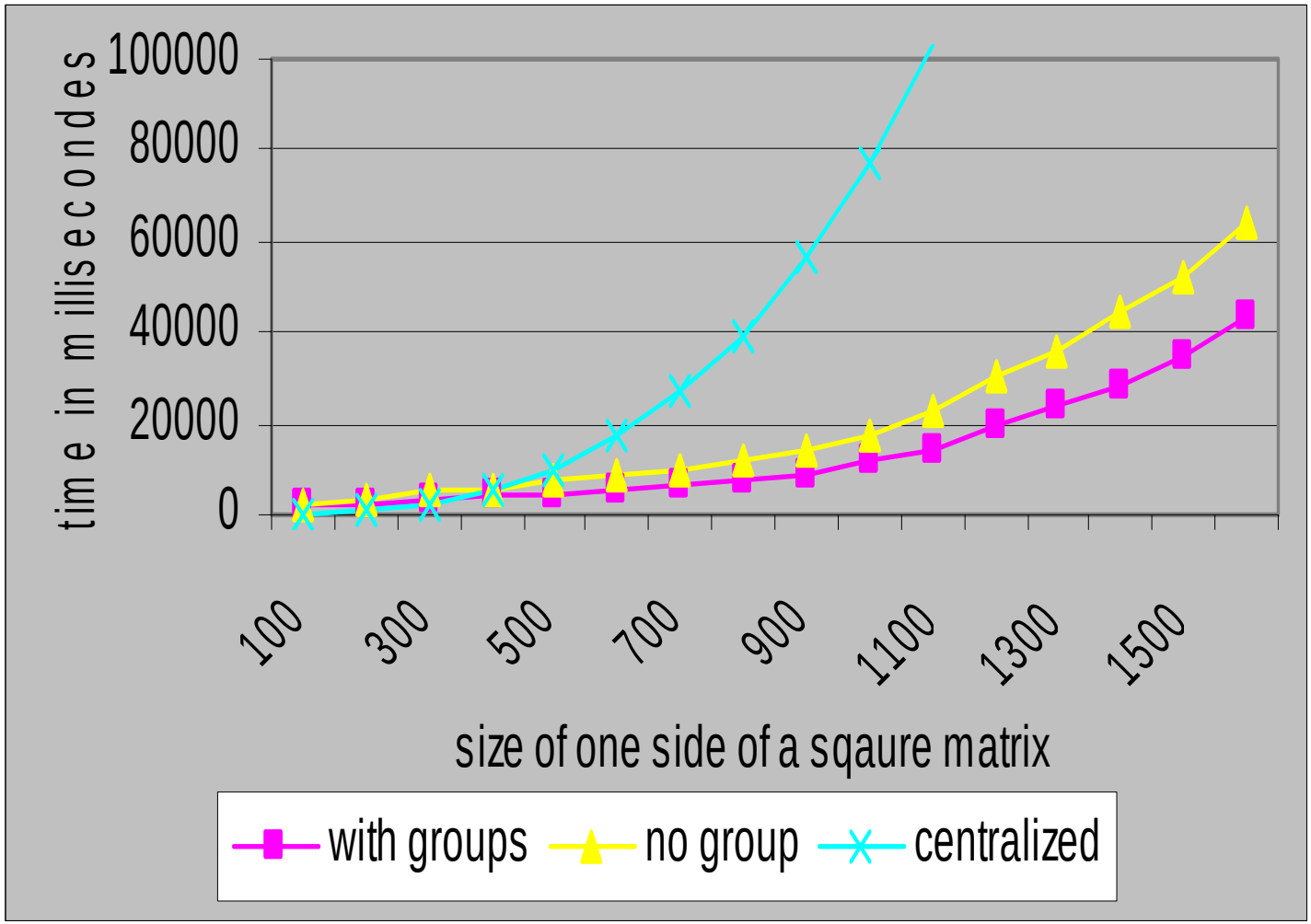
more than 20 lines of Java code



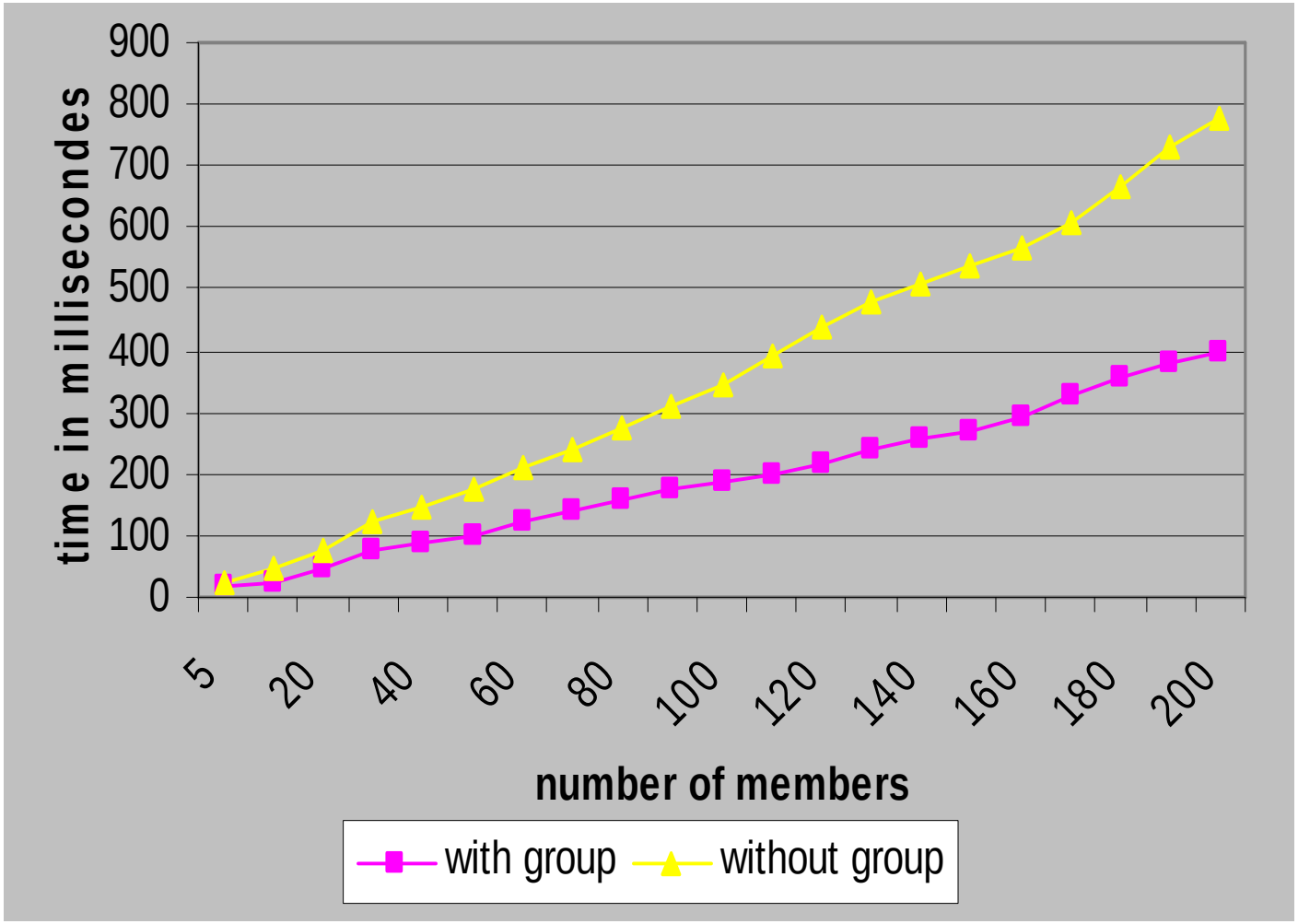
- 2 lines with ProActive groups

```
for (int i = 0 ; i<P ; i++)
  A.grouprow(i).multiply(B.groupcolumn(i));
```

Measurement : Matrix Multiplication



Measurement : Method Call



Features of Groups

Optimization:

- Parallel calls within a group (latency hiding)
- Treatment in the result order (if needed)
- Scatter (a group as a parameter to be dispatched in Group. Com.)
- A single serialization of parameters

Flexibility: Active Group

- A group becomes remotely accessible so: **updateable and consistent**

---> Dynamic changes

---> Migration of a group is possible

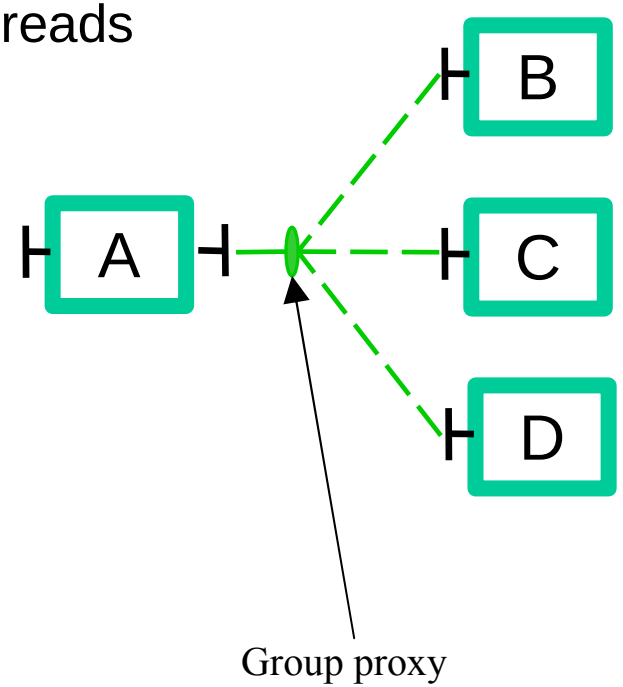
Perspective:

use network multicast, and new IPv6 features

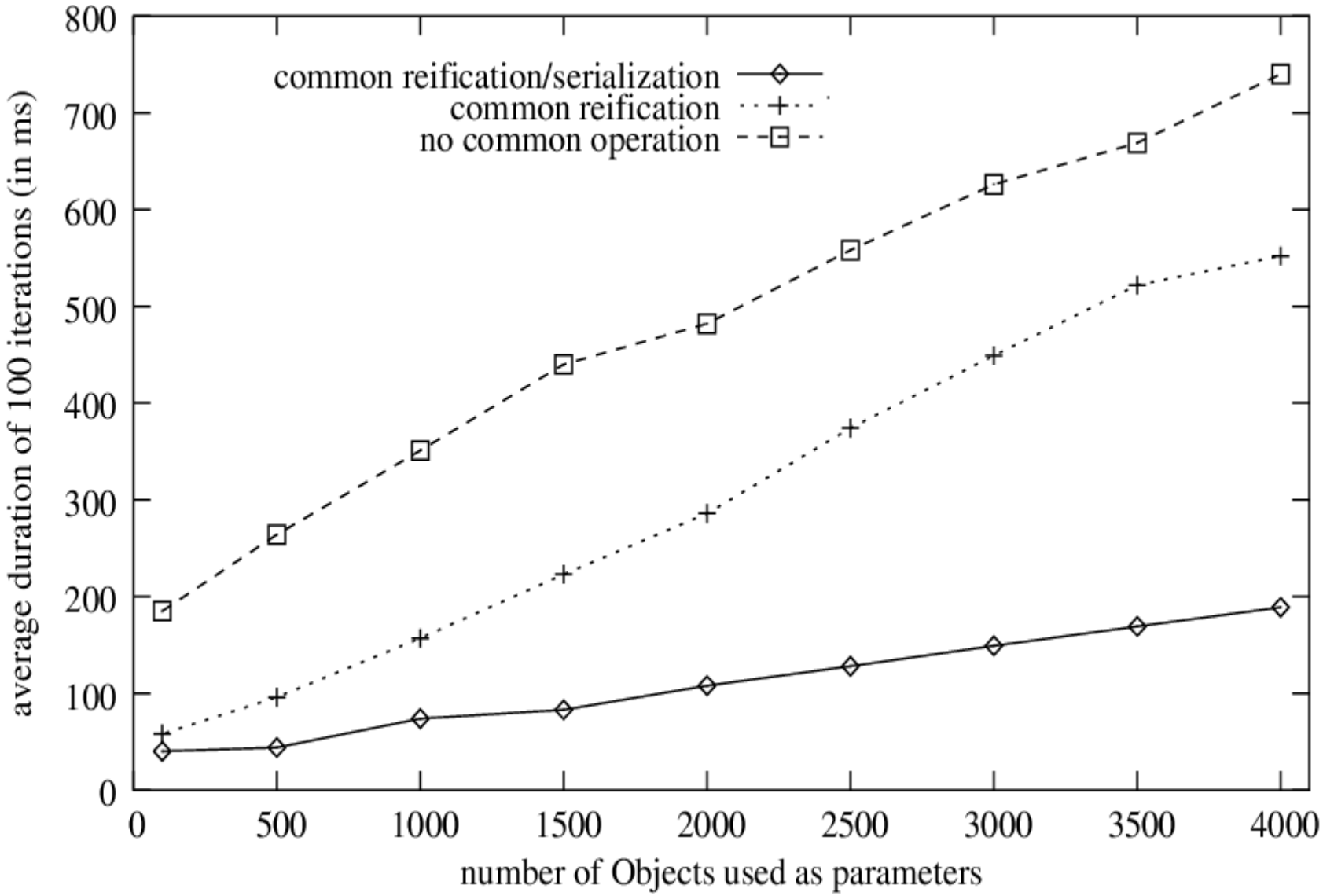
Adaptive Feature: Parallel Group Communications

Adaptive strategy to manage the number of threads

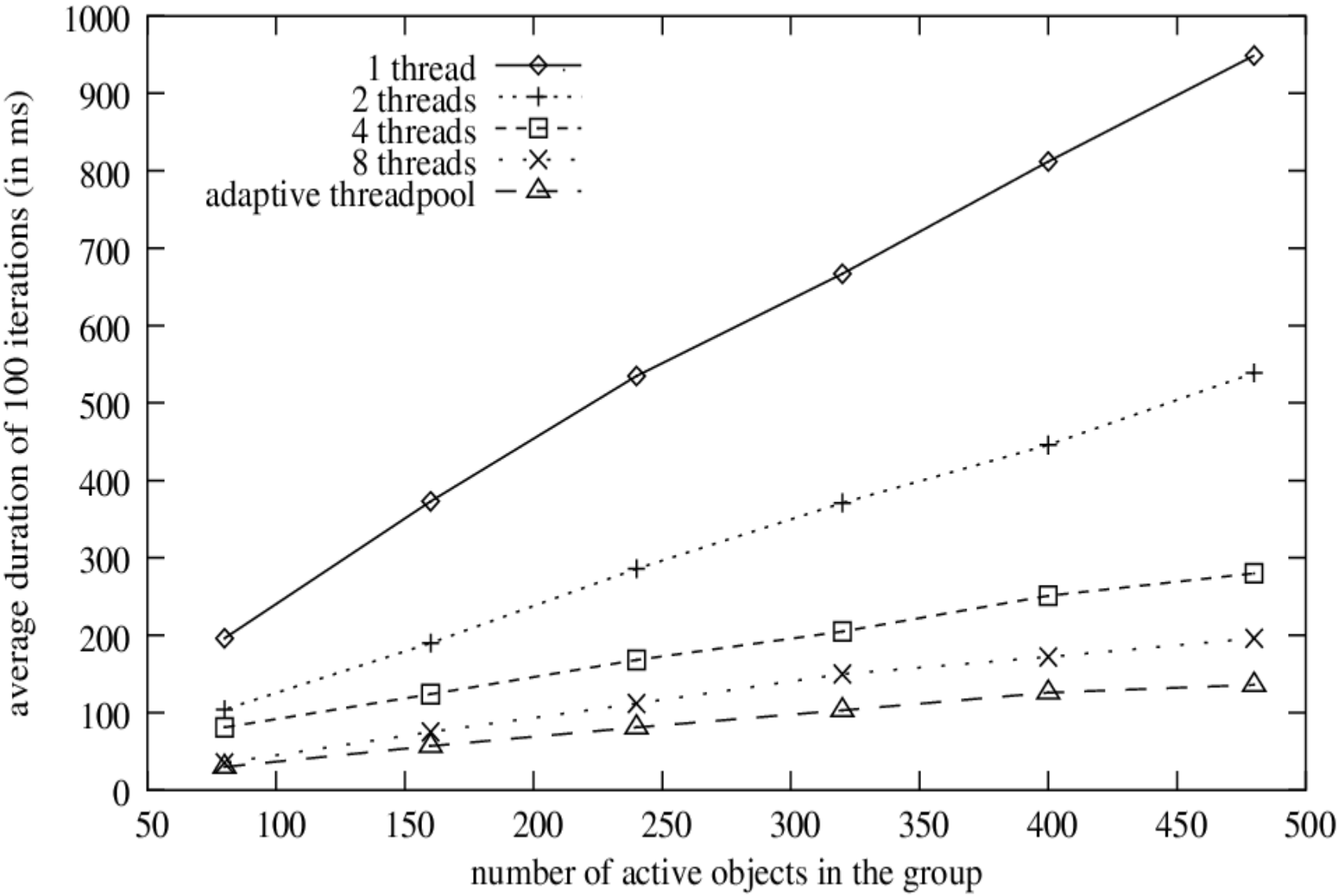
Execution of N calls in //



Different Optimizations in Groups



Thread Pool for sending Requests in //



Object-Oriented SPMD
Single Program Multiple Data

Motivation

- Cluster / GRID computing
- SPMD programming for many numerical simulations
- **Use enterprise technology (Java, Eclipse, etc.) for Parallel Computing**

Able to express most of MPI's Collective Communications:

broadcast	**reduce**
scatter	**allscatter**
gather	**allgather**

and Barriers, Topologies.

Main MPI problems for the Grid

Too static in design

Too complex interface (API)

More than 200 primitives and 80 constants

Too many specific primitives to be adaptive

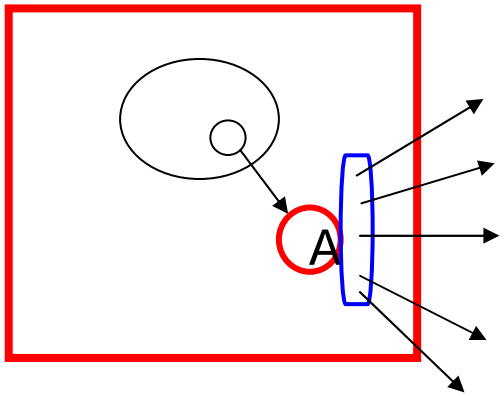
`Send`, `Bsend`, `Rsend`, `Ssend`, `Ibsend`, etc.

Typeless (Message Passing rather than RMI)

Manual management of complex data structures

OO SPMD

```
A ag = newSPMDGroup ("A", […], VirtualNode)
        // In each member
        myGroup.barrier ("2D"); // Global Barrier
        myGroup.barrier ("vertical"); // Any Barrier
        myGroup.barrier ("north","south","east","west");
```

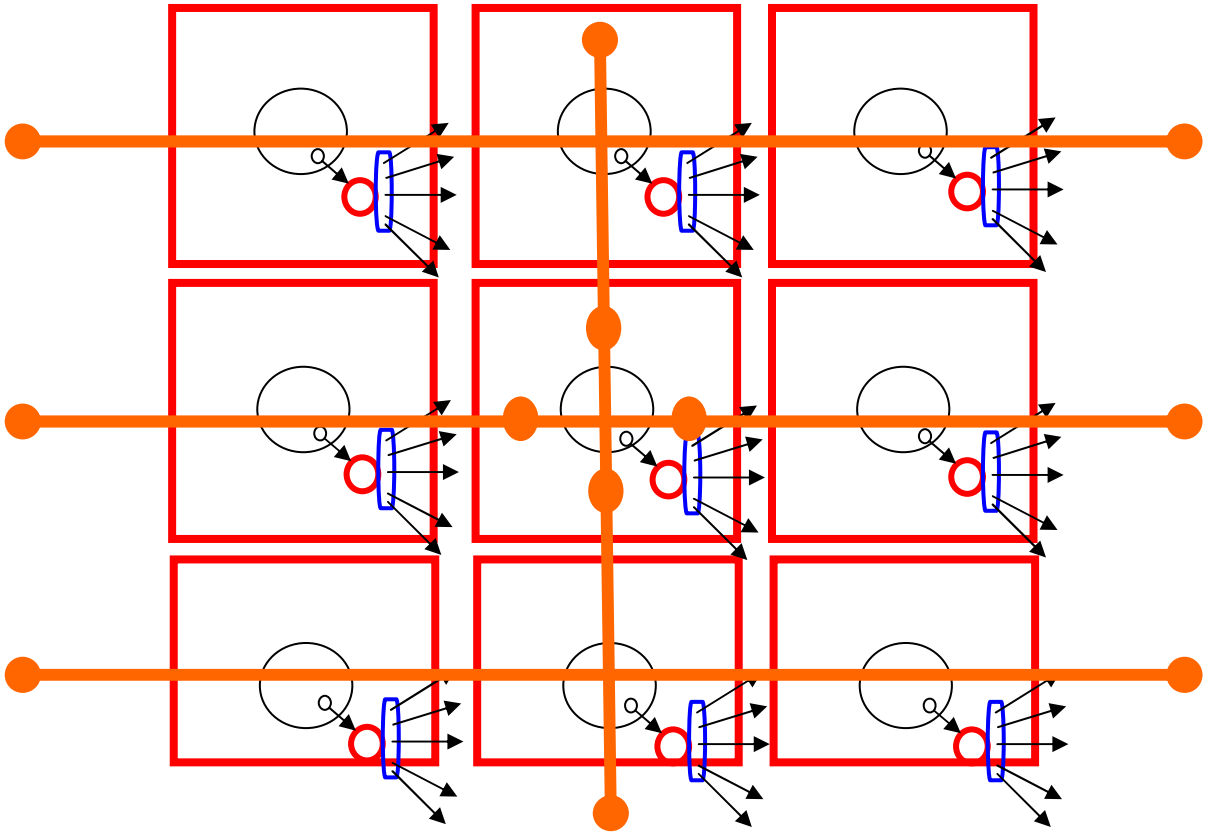


Still,

not based on raw messages, but

Typed Method Calls

==> Components



API

Topologies

- Table, Ring, Plan, Torus, Cube, …
- Open API
- Neighborhood

Barrier

- Global
- Neighbor-based
- Method-based

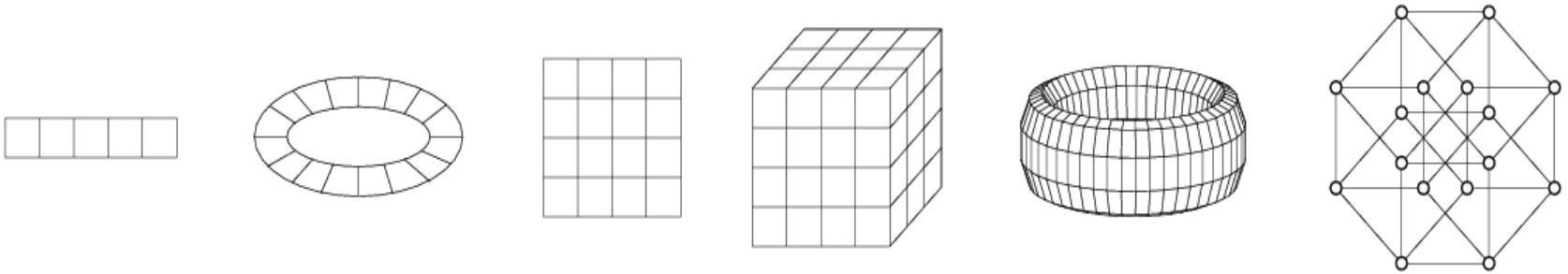
Topologies

Topologies are typed groups

Open API

Neighborhood

Creation by extraction




```
Plan plan = new Plan(groupA, Dimensions);
Line line = plan.getLine(0);
```

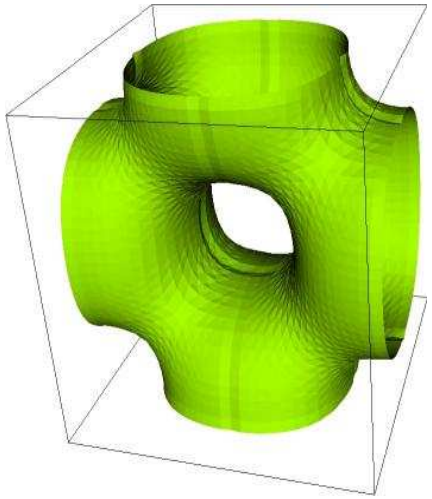
ProActive OO SPMD

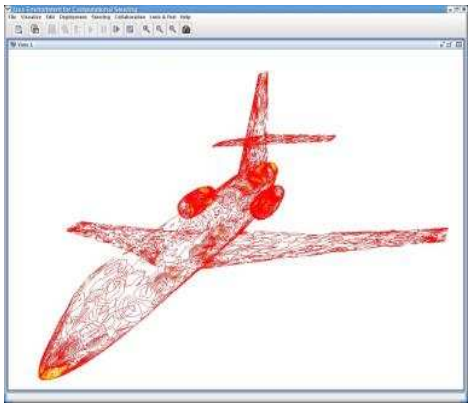
A simple communication model

- Small API
- No “Receive” but data flow synchronization
- No message passing but RPC (RMI)
- User defined data structure (Objects)
- SPMD groups are dynamics
- Efficient and dedicated barriers

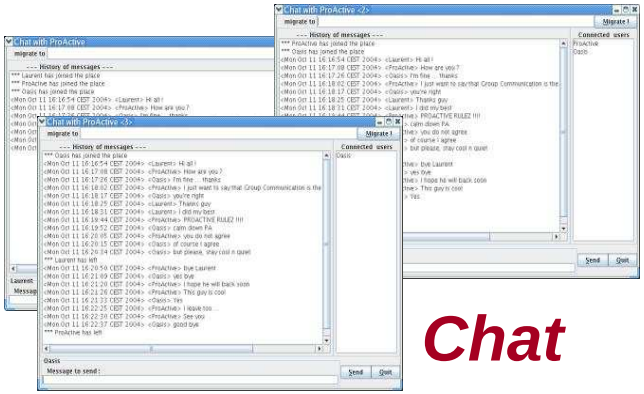
Group communication is a key feature of ProActive

Used in many applications and other features

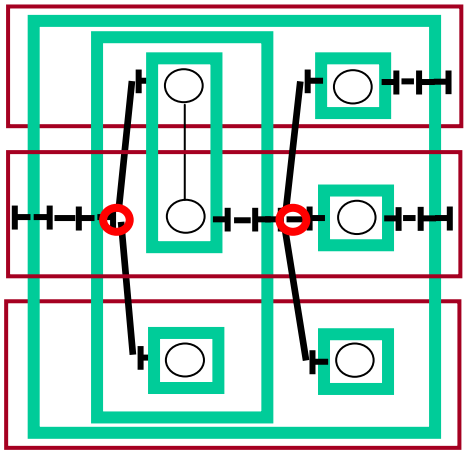




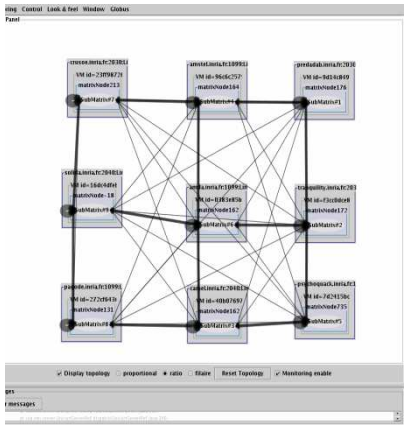
Jem3D



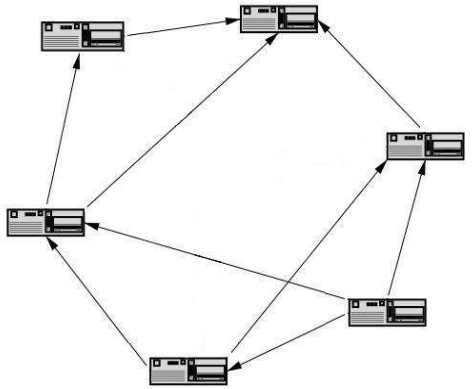
Chat



Multiport component



Jacobi



Peer to peer

MPI Communication primitives

For some (historical) reasons, MPI has many com. Primitives:

MPI_Send	Std	MPI_Recv	Receive
MPI_Ssend	Synchronous	MPI_Irecv	Immediate
MPI_Bsend	Buffer	… (any) source, (any) tag,	
MPI_Rsend	Ready		
MPI_Isend	Immediate, async/future		
MPI_Ibsend, …			

I'd rather put the burden on the implementation, not the Programmers !

How to do adaptive implementation in that context ?

Not talking about:

- the combinatory that occurs between send and receive
- the semantic problems that occur in distributed implementations

Is Recv at all needed ? First adaptive feature: Dynamic Control Flow of Mess.

MPI and Threads

MPI was designed at a different time

When OS, languages (e.g. Fortran) were single-threaded

No longer the case.

Programmers can write more simple, ”sequential” code,

the implementation, the middleware, can execute things in parallel.

Main MPI problems for the GRID

Too static in design

Too complex in Interface (API)

Too many specific primitives to be adaptive

Type Less

… and you do not ’’lamboot’’ / ’’lamhalt’’ the GRID !

Adaptive GRID

The need for adaptive middleware is now acknowledged,
with dynamic strategies at various points in containers, proxies, etc.

Can we afford adaptive GRID ?

with dynamic strategies at various points
(communications, groups, checkpointing, reconfiguration, …)
for various conditions (LAN, WAN, network, P2P, ...)

HPC vs. HPC

High Performance Components vs. High Productivity Components

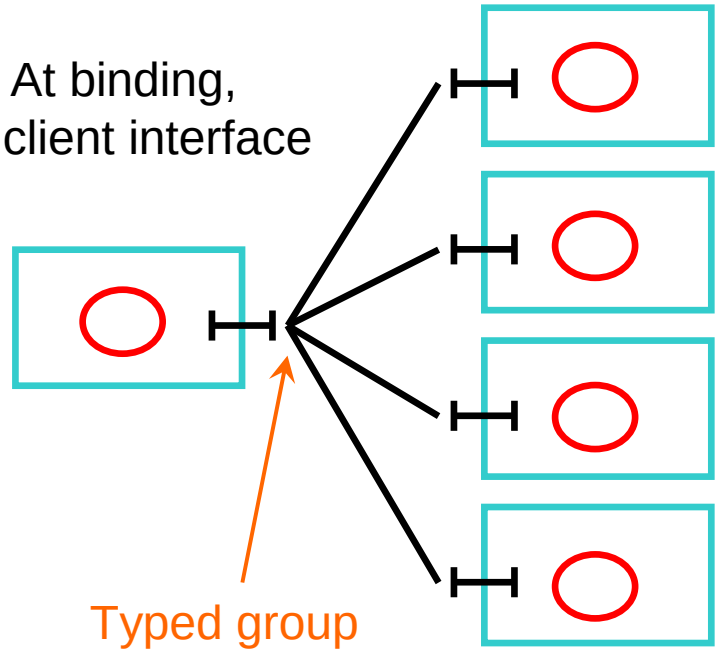
Groups in Components

Implementation of the Fractal component model

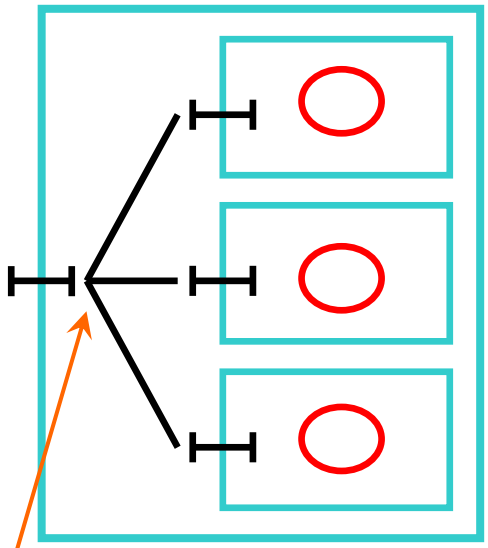
Collective ports

Composite components

At binding,
on client interface



At composition,
on composite inner server interface



Sum up: MPI vs. ProActive OO SPMD

A simple communication model, with simple communication primitive(s):

No RECEIVE but data flow synchronization

Adaptive implementations are possible for:

- // machines, Cluster, Desktop, etc.,
- Physical network, LAN, WAN, and network conditions
- Application behavior

Typed Method Calls:

==> Towards Components

Reuse and composition:

No main loop, but asynchronous calls to myself,

SPMD Programming : MPI to ProActive translation

`mpirun`	*Deployment on virtual nodes*
`MPI_Init`	`newActive()`
`MP_Finalize`	`terminate()`
`MPI_Comm_Size`	`getMyGroupSize()`
`MPI_Comm_rank`	`getMyRank()`
`MPI_*Send / MPI_*Recv`	*Method call :* `result=obj.foo()`
`Recv + Send + Wait` *sequence*	`exchange()`
`MPI_Barrier`	`barrier()`
`MPI_BCast`	*Group Method call :* `objGroup.foo()`
`MPI_Scatter`	*Method call with a scatter group as parameter*
`MPI_Gather`	*Result of a group communication*
`MPI_Reduce`	*Programmer's method*

SPMD Programming : MPI to ProActive translation

MPI	ProActive	Example
`mpirun`	Deployment on virtual nodes	
`mpi_init`	Activity creation	`a = newActive(..)`
`mpi_finalize`	Activity termination	`a.terminate()`
`mpi_comm_size`	Get the size of my group	`getMyGroupSize()`
`mpi_comm_rank`	Retrieve my rank number	`getMyRank()`
`mpi_send/recv`	Method call on active object	`result = a.foo(..)`
`recv+send+wait`	Exchange operation	`exchange(..)`
`mpi_barrier`	Synchronization request	`barrier()`
`mpi_bcast`	Group method call	`groupOfA.foo(..)`
`mpi_scatter`	Method call with a scatter group as parameter	`groupOfA.foo(scatterGroup)`
`mpi_gather`	Result of a group call	`result = groupOfA.foo(..)`
`mpi_reduce`	Programmer's method	`groupOfA.myReduce()`

1.3. ProActive architecture:

a simple MOP

Design of the library

ProActive architecture: a simple MOP

- MOP (Meta-Object Protocol)
  - Runtime reflection (for dynamicity)
  - New semantics for method and constructor calls
  - Same technique for Future implementation
  - Uses the Java Reflection API
- ProActive
  - Implemented on top of the MOP
  - Other models can be built on top of ProActive or on top of the MOP

MOP principles

Static or dynamic generation of stubs:

Take place of a reified object

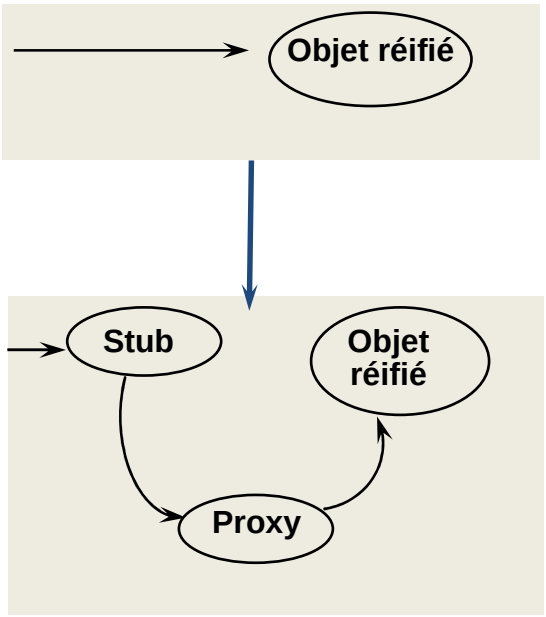
Reification of all method calls

Sub-class of the reified object: type compatible

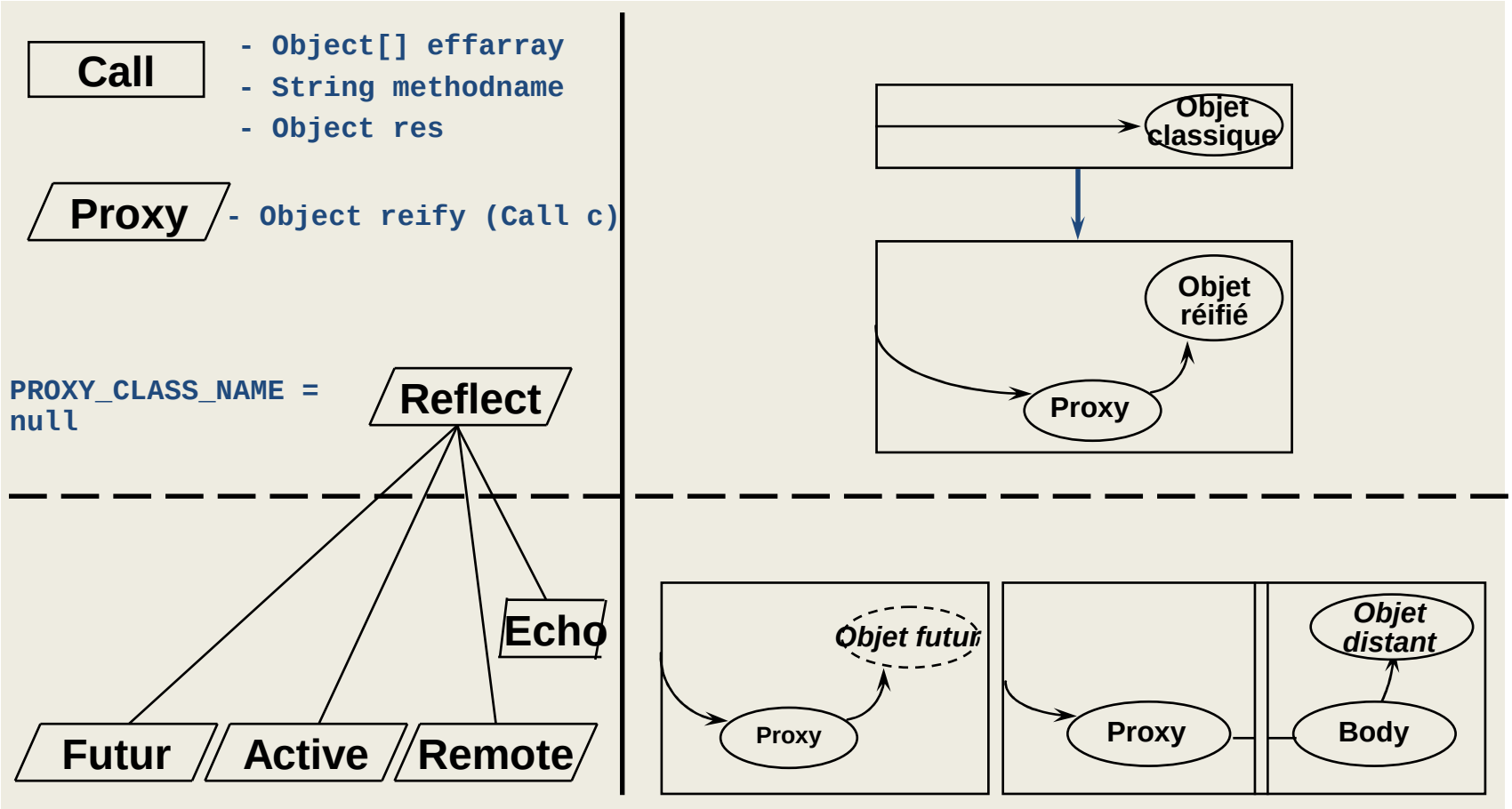
Stub only depends of the reified object type,
not of the proxy

Any call will trigger the creation of an object **`Call`** that represents the invocation.

It will be passed to the proxy: has the semantics to achieve



The MOP - principle



All interfaces that inherit Reflect

are marker interfaces for reflexion

User Interface of the MOP

Instantiation of reified objects with static method newInstance of the MOP class

Programming *class par class* with interfaces deriving from **`Reflect`**

```
Vector v = (Vector) MOP.newInstance ( <name of reified class (impl. Reflect)>,
                                      <parameters passed to proxy>,
                                      < parameters of reified object constructor > );
```

Or *instance per instance :*

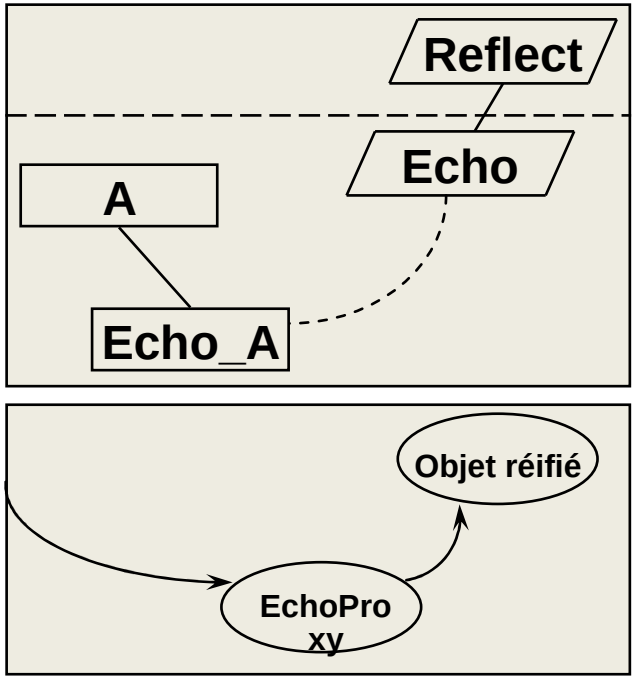
```
Vector v = (Vector) MOP.newInstance ( <name of class standard>,
                                      <class proxy name to use>,
                                      <parameters given to proxy>,
                                      <parameters of reified object constructor > );
```

Or *object per object :*

```
Vector v = (Vector) MOP.turnReified  ( <objet standard>,
                                       <class proxy name to use>,
                                       <parameters given to proxy> );
```

Example : EchoProxy

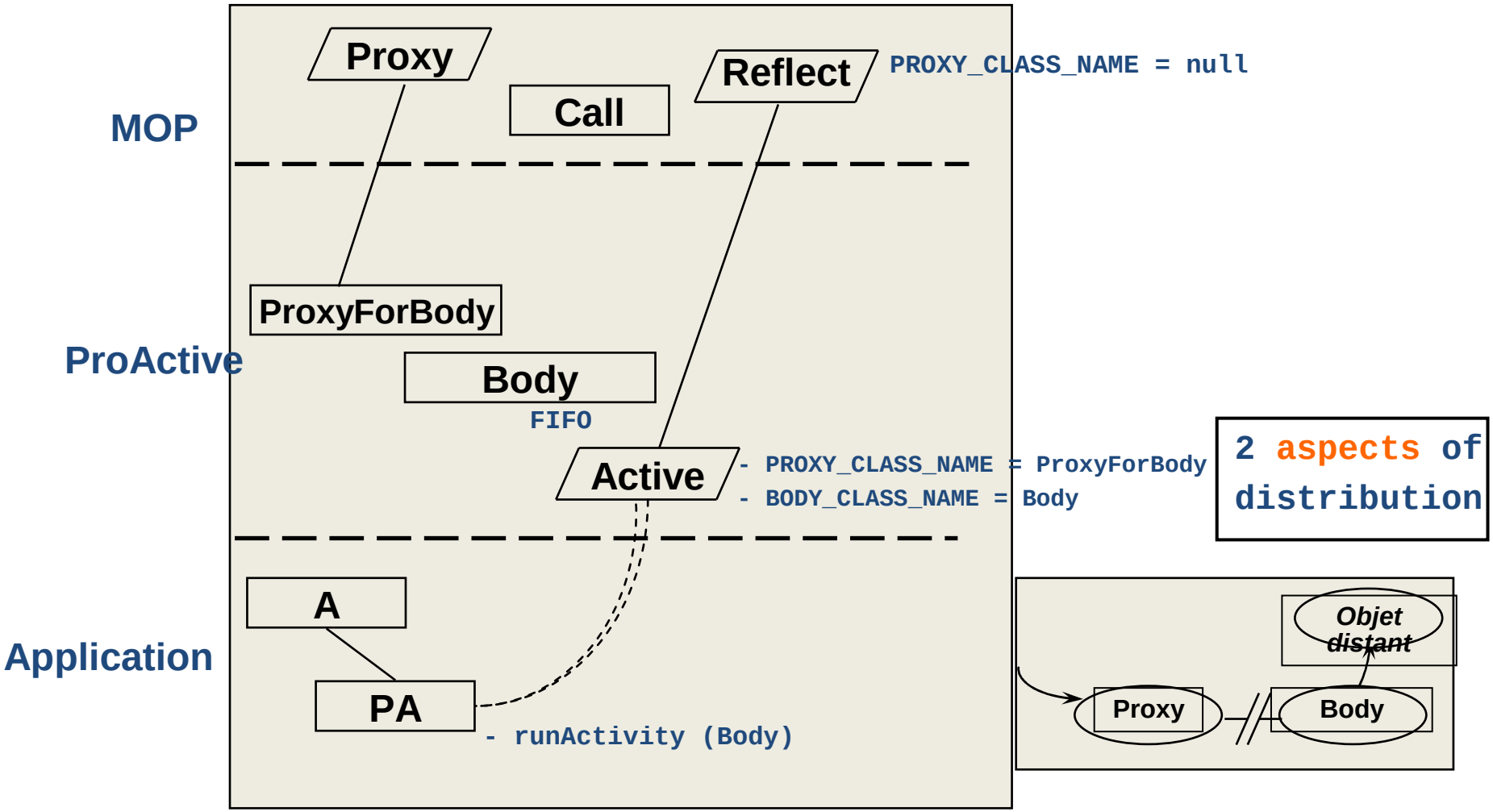
```
public class EchoProxy implements Proxy {
// Attributes
  Object myobject;
// Constructor
  public EchoProxy (Call c, Object[] p) {
     this.myobject = c.execute();
  }
// Method of the Proxy interface
  public Object reify (Call c) {
     System.out.println ("Echo->"+c.methodname+");
     return result = c.execute (myobject);
  }
}
```




```
public interface Echo_A extends Reflect {
  PROXY_CLASS_NAME = "EchoProxy"; }
```

```
A a =(A)newInstance ("Echo_A",null,null);
A a =(A)newInstance ("A","EchoP.",null,null);
A a =(A)turnReified ( a, "EchoP.", null);
```

ProActive : implementation principle



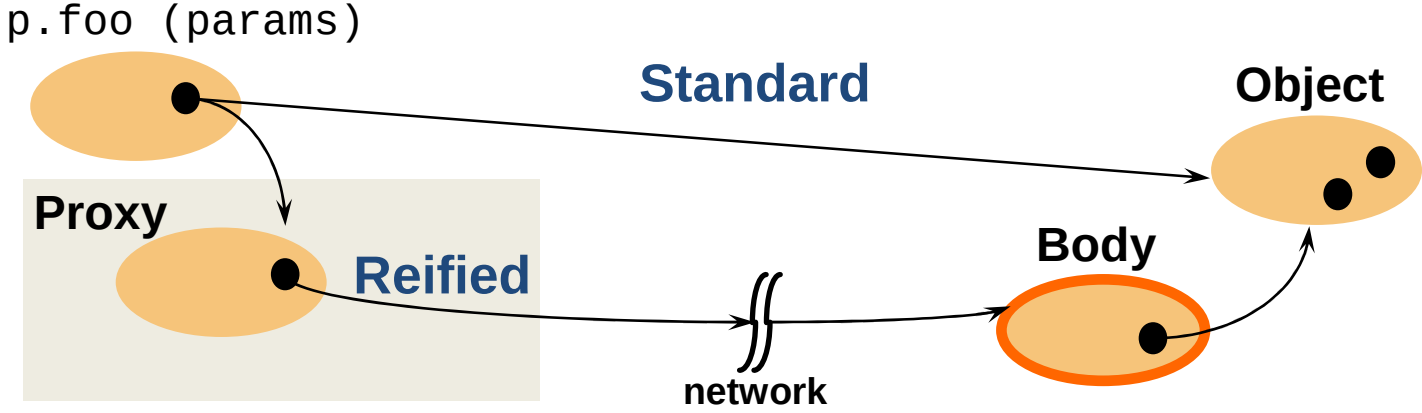
Proxy and Body

Based on interface **`Active`** and class **`ProActive`**

MOP **Reflect**

ProActive **Active** **Future** **...**

A proxy / body model



Originalities:

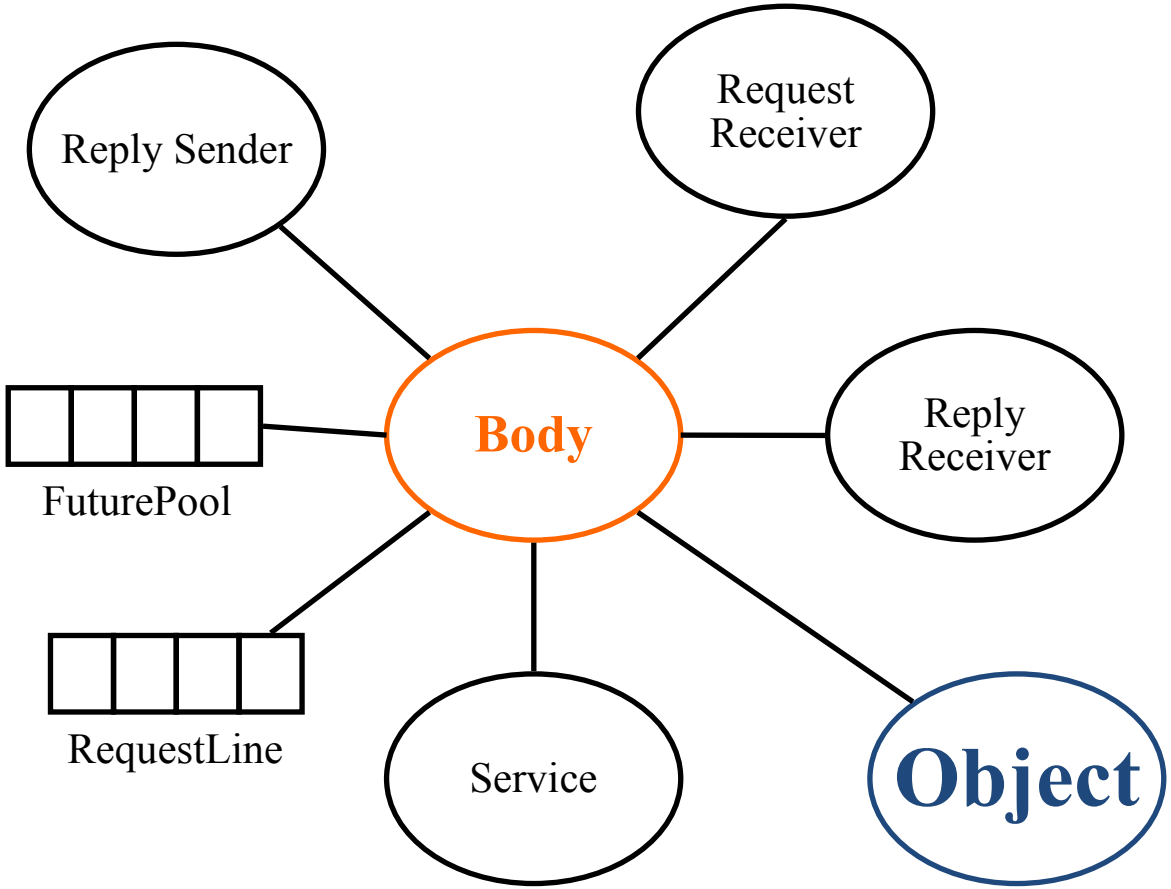
Extensions through inheritance of the **`Reflect`** interface

3 ways to turn a standard object into a reified one

Reuse of existing classes, polymorphism between standard and reified objects

1.4 Meta-Objects for Distribution

An Active Object



Composition of an Active Object

Multiples Objets

RequestSender: *Send requests (proxy + body)*

RequestReceiver**:** Receve the requests

ReplySender**:** Send back the result to the caller

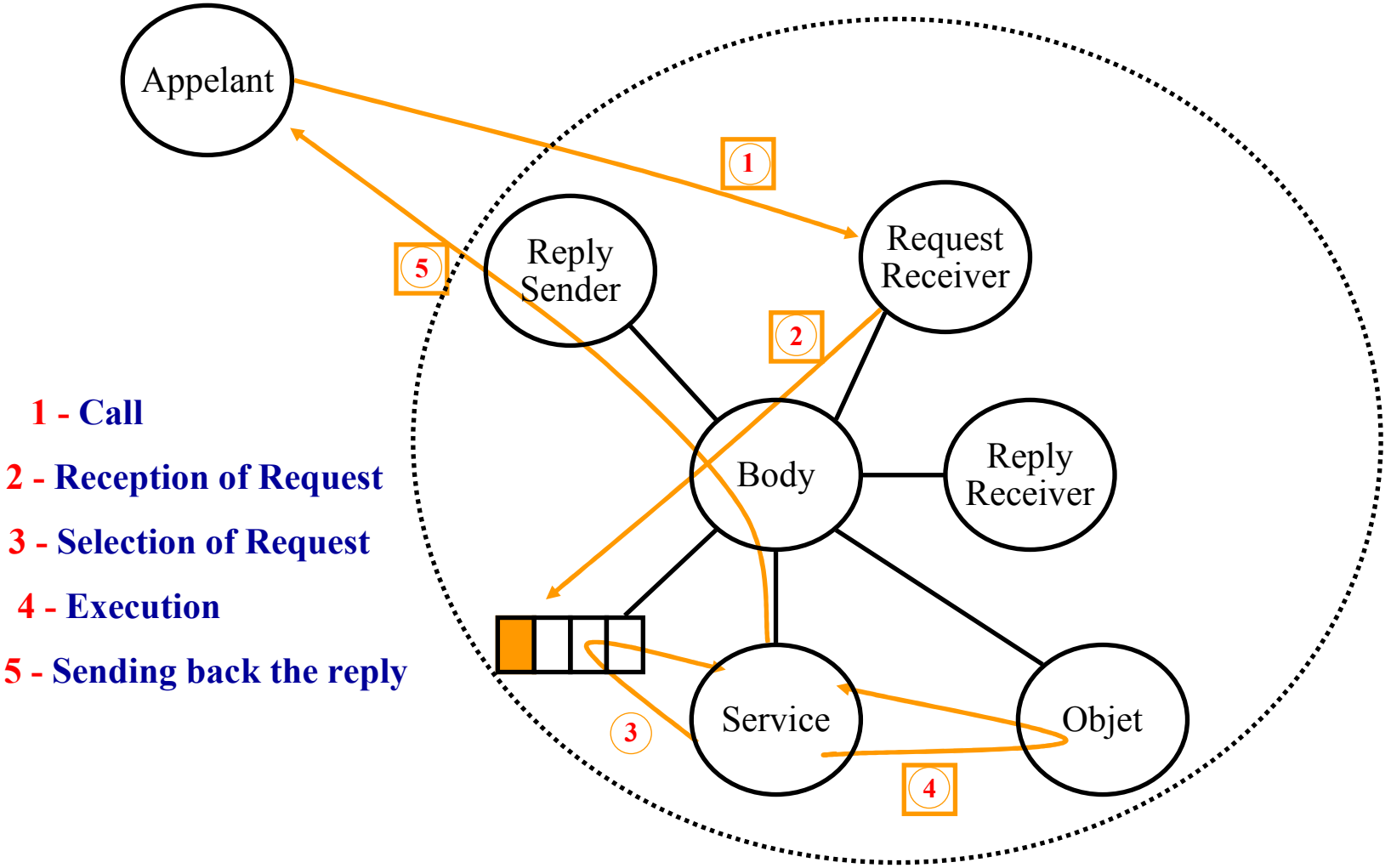
ReplyReceiver**:** Receive the future updates

Service**:** Chose (select) and executes the requests

RequestLine**:** Pending Requests

FuturePool**:** Pending Futures

Request to an Active Object



1 - Call

2 - Reception of Request

3 - Selection of Request

4 - Execution

5 - Sending back the reply

Listener

Pattern Event Listener

Events are (if needed) generated for each important step

If asked for, sent to the listeners

These Listeners can be added/suppressed dynamically

Event Types (1)

3 Main Categories:

1. Active Object:
   - Creation
   - Migration (activation, Inactivation : Life Cycle)

2. Communications:
   - **Requests:**
   - RequestSent
   - RequestReceived
   - **Reply:**
   - ReplySent
   - ReplyReceived

3. Service (activity of an AO):
   - WaitForRequest
   - WaitByNecessity

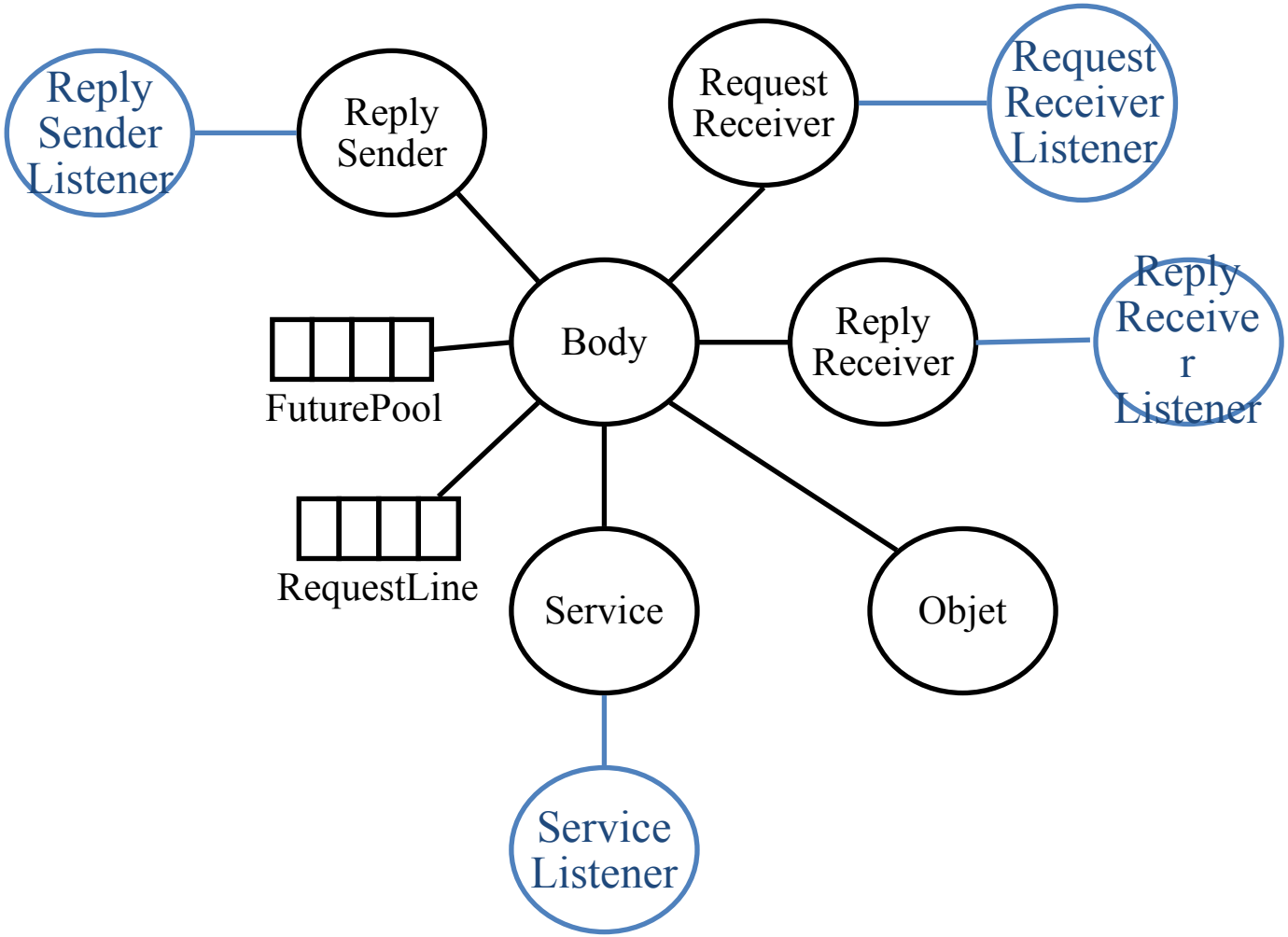
Listener - Modifier

Idem Listener + modification of the AO execution:

At creation: change the VM of creation
At migration: changer the destination VM
Step-by-step on communications
etc.

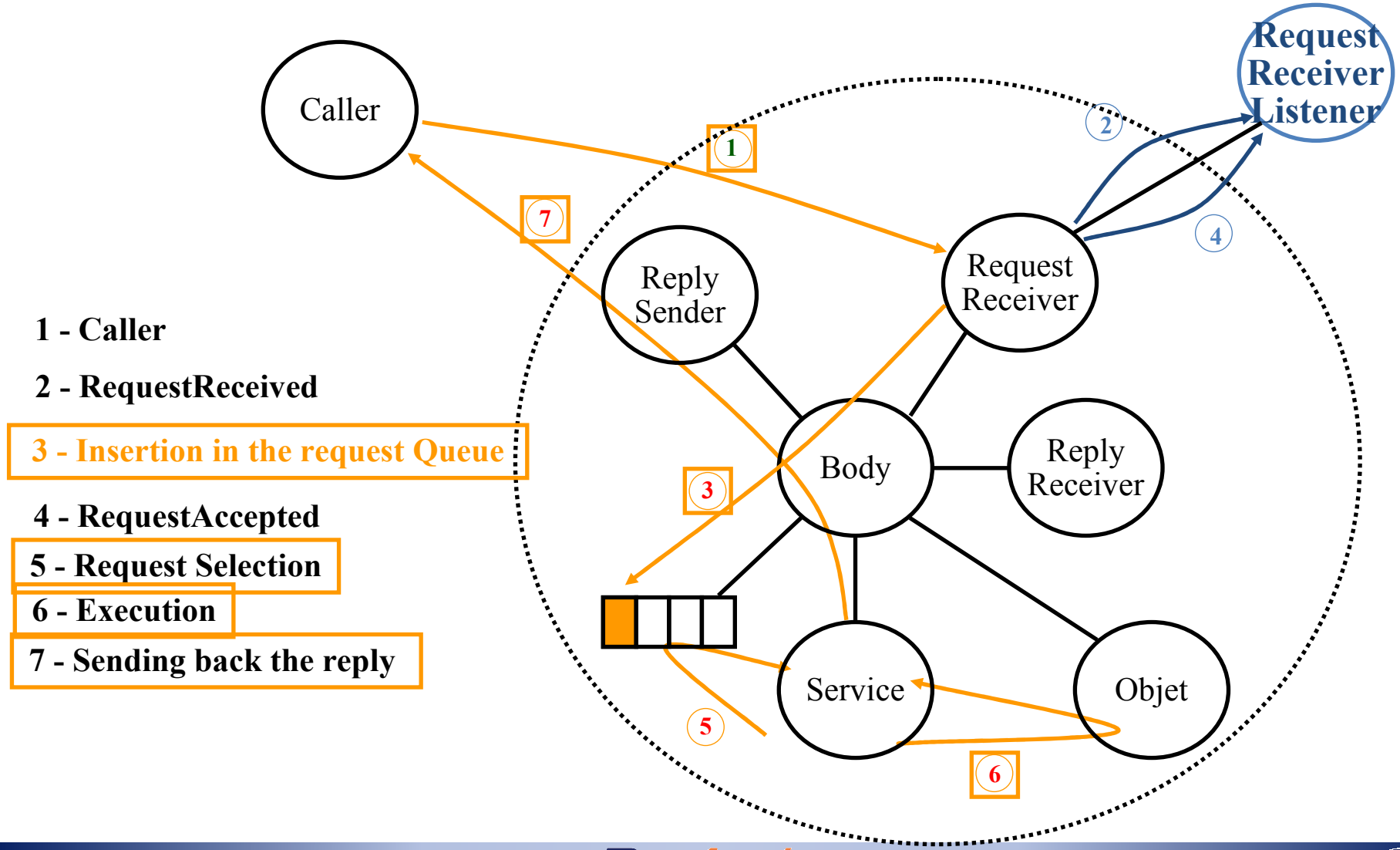
Application: debugging, monitoring, interactive Control of execution

Localization of listeners



Request Reception with a Listener

Used for IC2D implementation (see below)



2. *ProActive* : Migration of active objects

Migration is initiated by the active object itself through a primitive: migrateTo

Can be initiated from outside through any public method

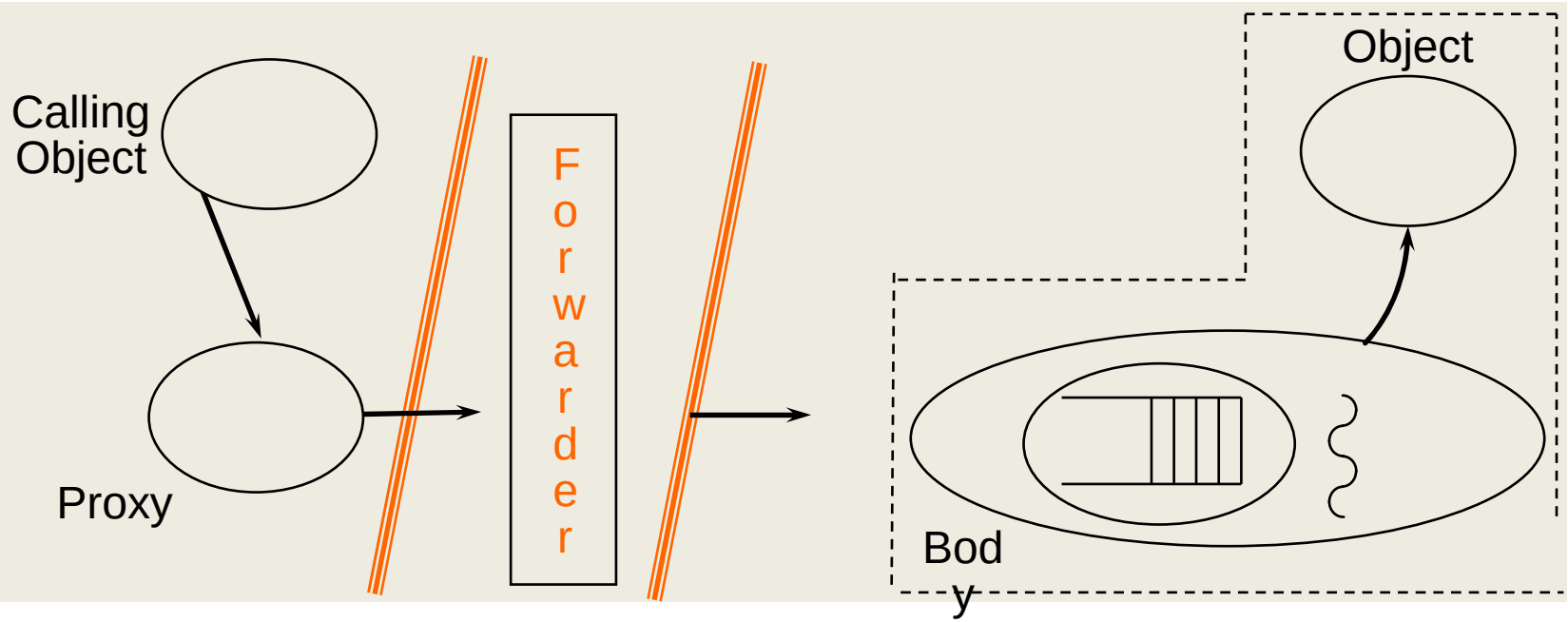
The active object migrates with:

- all pending requests
- all its passive objects
- all its future objects

Automatic and transparent forwarding of:

- requests (remote references remain valid)
- replies (its previous queries will be fullfilled)

ProActive : Migration of active objects



Migration is initiated by the active object through a request

The active object migrates with

- its passive objects - the queue of pending requests - its future objects

2 Techniques : Forwarders or Centralized server

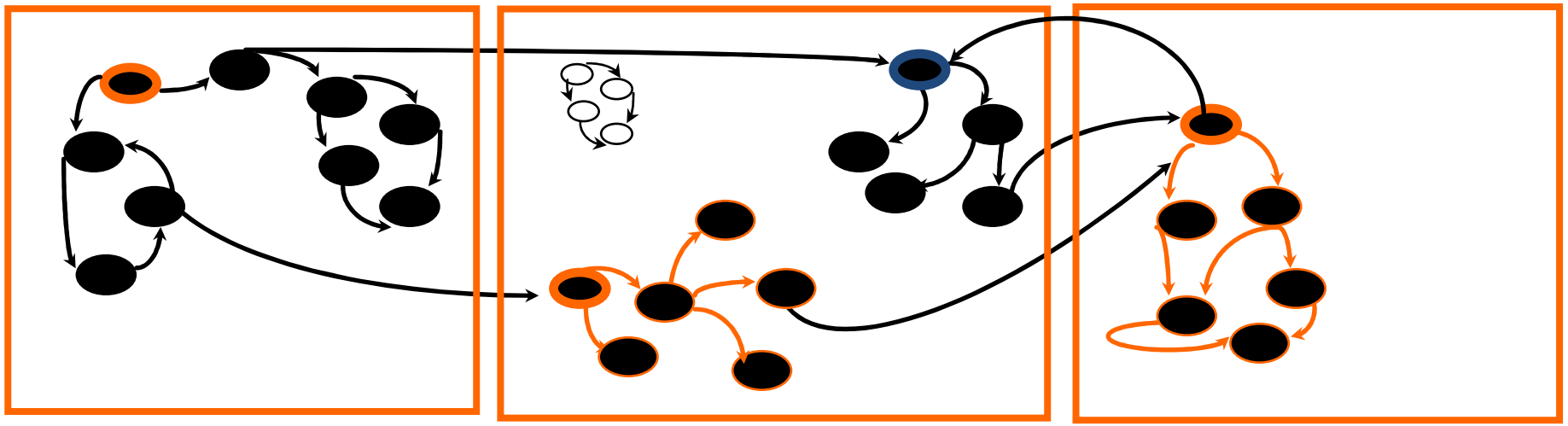
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



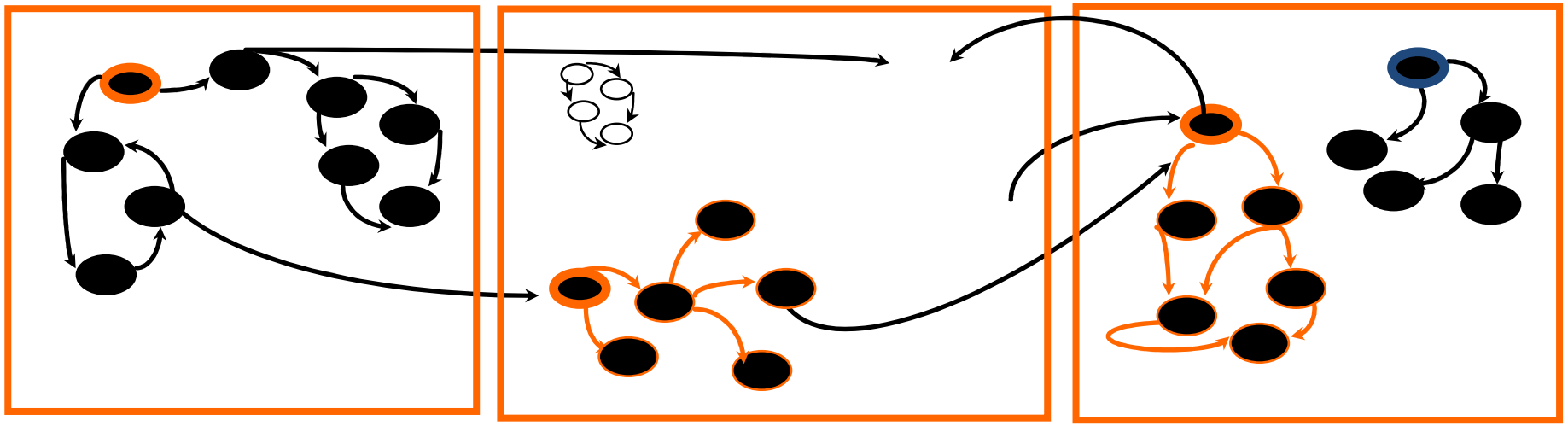
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



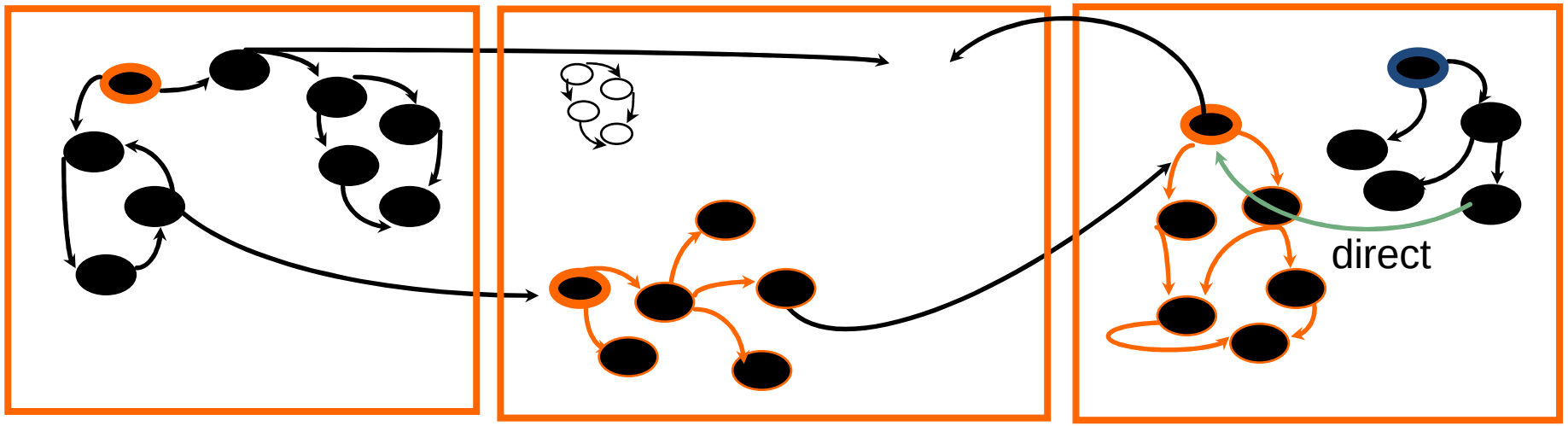
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



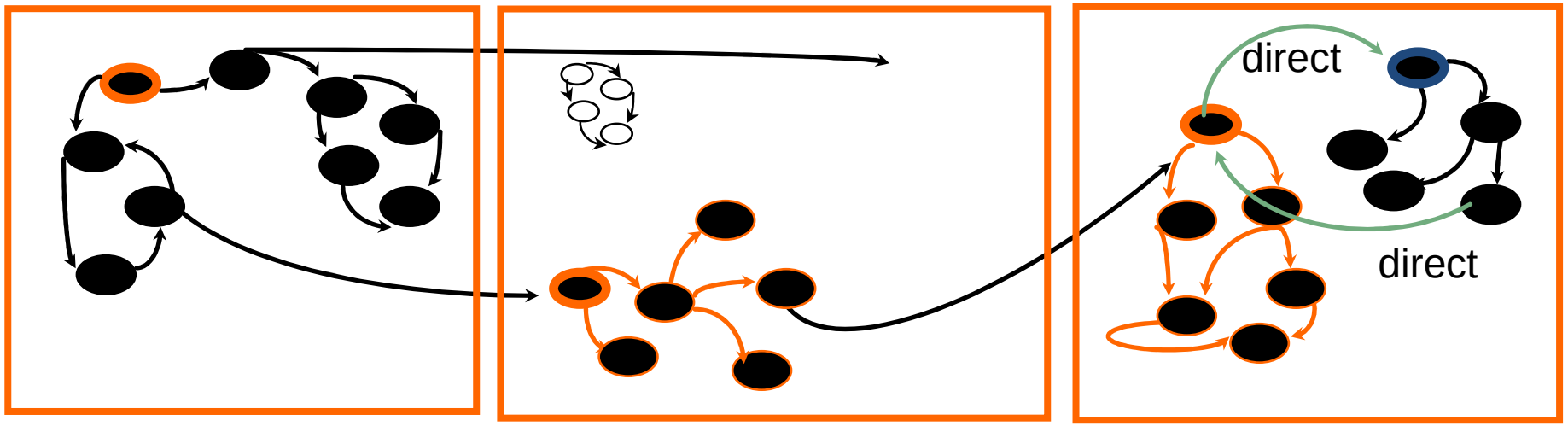
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



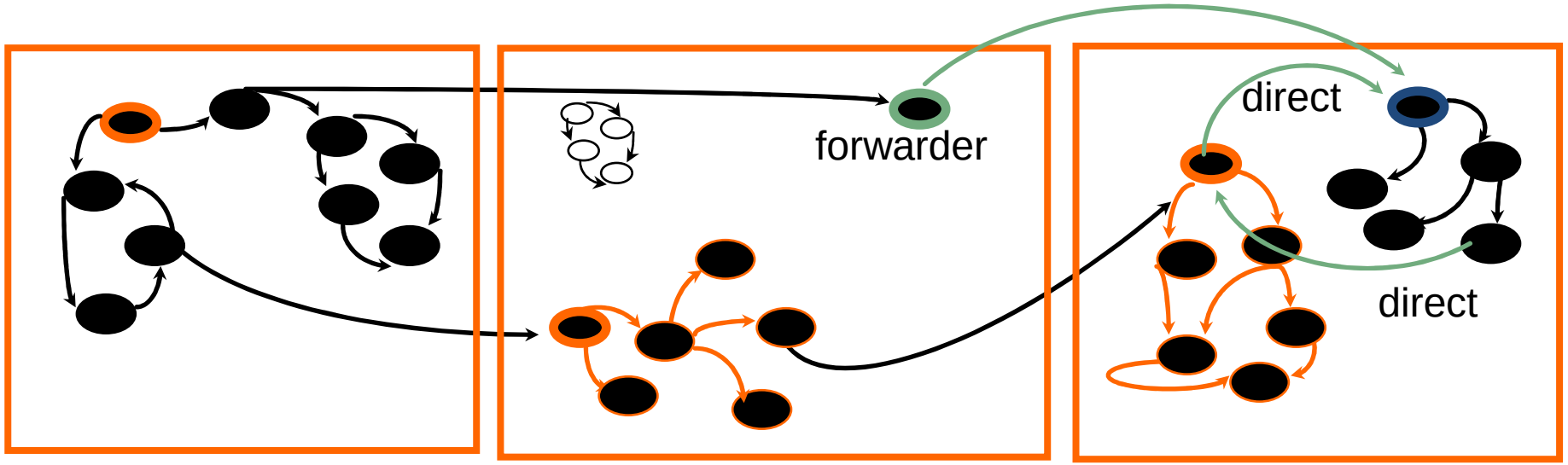
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



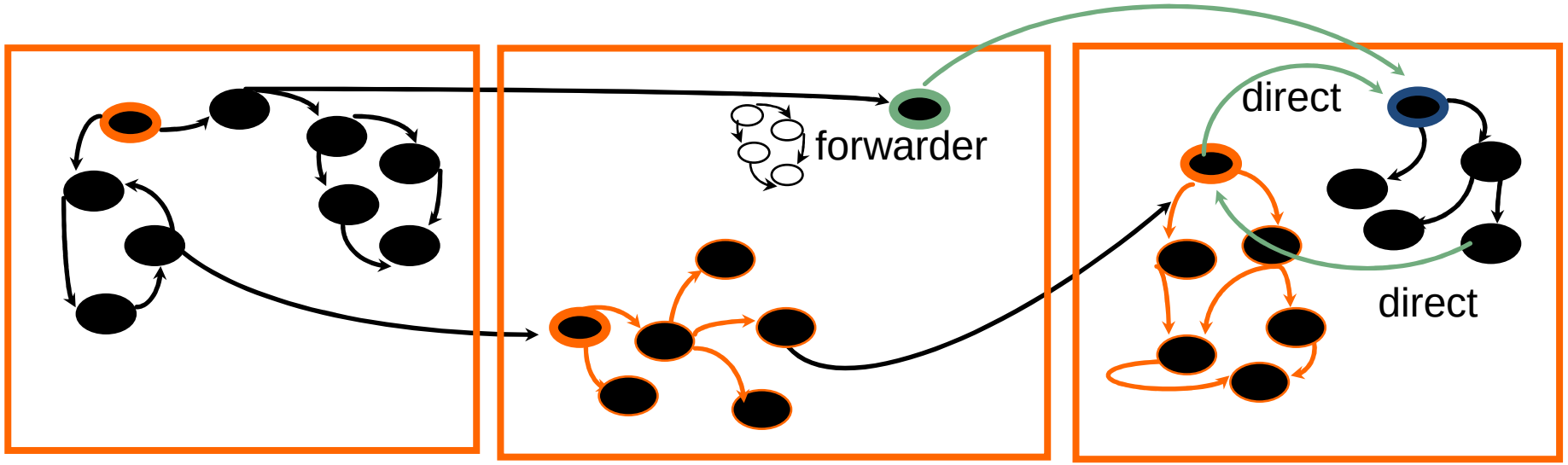
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



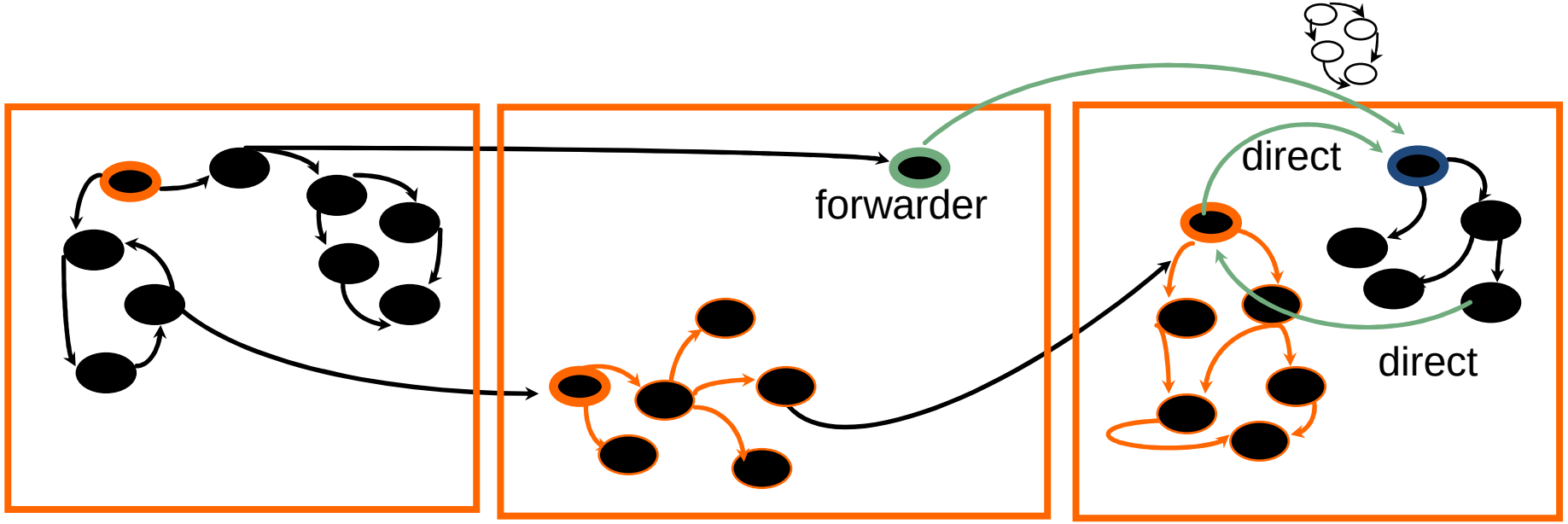
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



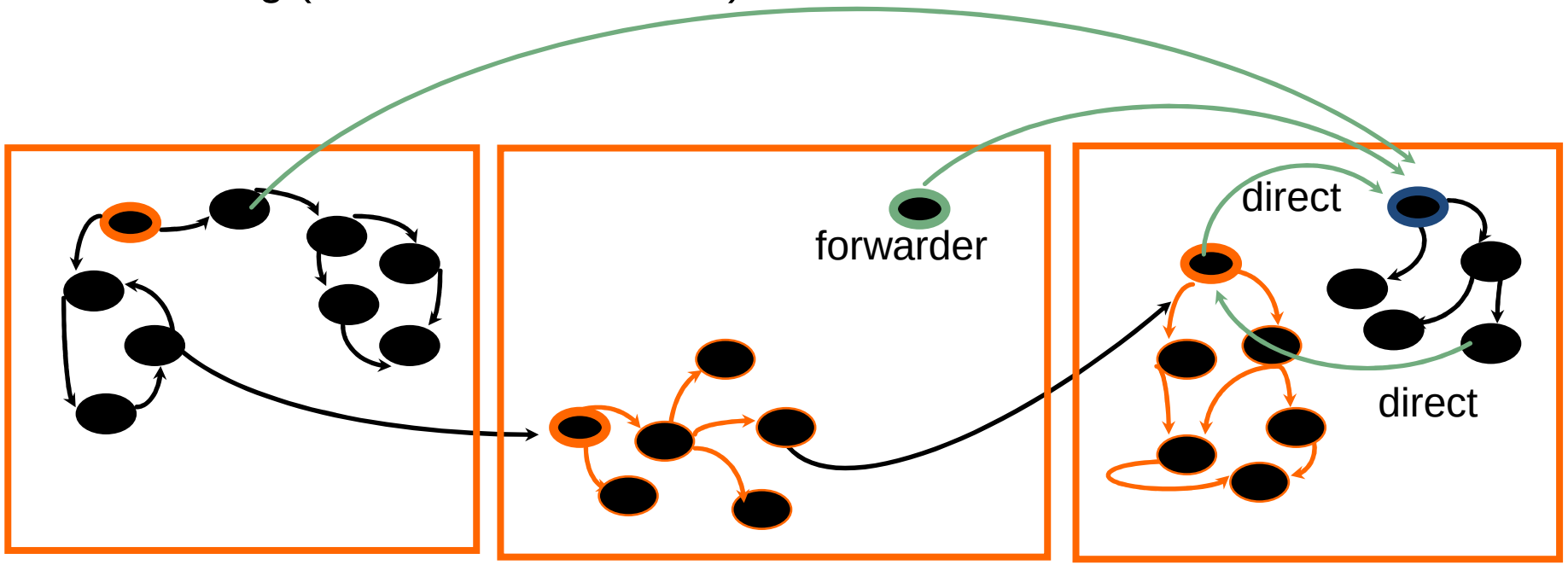
Principles and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)

Safe migration (no agent in the air!)

Local references if possible when arriving within a VM

Tensionning (removal of forwarder)



ProActive : API for Mobile Agents

Mobile agents (active objects) that communicate

Basic primitive: migrateTo

```
public static void migrateTo (String u)
        // string to specify the node (VM)

public static void migrateTo (Object o)
        // joinning another active object

public static void migrateTo (Node n)
        // ProActive node (VM)

public static void migrateTo (JiniNode n)
        // ProActive node (VM)
```

ProActive : API for Mobile Agents

Mobile agents (active objects) that communicate

```
// A simple agent
class SimpleAgent   implements runActive, Serializable {
  public SimpleAgent () {}
  public void moveTo (String t){ // Move upon request
        ProActive.migrateTo (t);
    }
    public String whereAreYou (){ // Repplies to queries
          return ("I am at " + InetAddress.getLocalHost ());
    }
    public runActivity (Body myBody){
          while (… not end of itinerary …){
                res = myFriend.whatDidYouFind () // Query other agents

          }
          myBody.fifoPolicy(); // Serves request, potentially moveTo
    }
 }
```

ProActive : API for Mobile Agents

Mobile agents that communicate

Primitive to automatically execute action upon migration

public static void **onArrival** (String r)

// Automatically executes the routine r upon arrival

// in a new VM after migration

public static void **onDeparture** (String r)

// Automatically executes the routine r upon migration

// to a new VM, guaranted safe arrival

public static void **beforeDeparture** (String r)

// Automatically executes the routine r before trying a migration

// to a new VM

ProActive : API for Mobile Agents
Itinerary abstraction

Itinerary : VMs to visit

specification of an itinerary as a list of (site, method)

automatic migration from one to another

dynamic itinerary management (start, pause, resume, stop, modification, …)

API:

myItinerary.add (“machine1’’, “routineX”); ...

itinerarySetCurrent, itineraryTravel, itineraryStop, itineraryResume, …

Still communicating, serving requests:

itineraryMigrationFirst ();

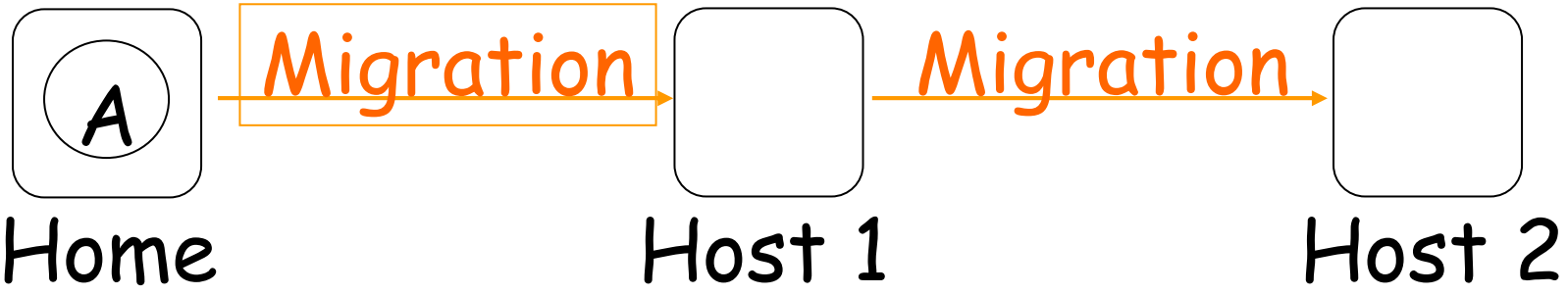
// Do all migration first, then services, Default behavior

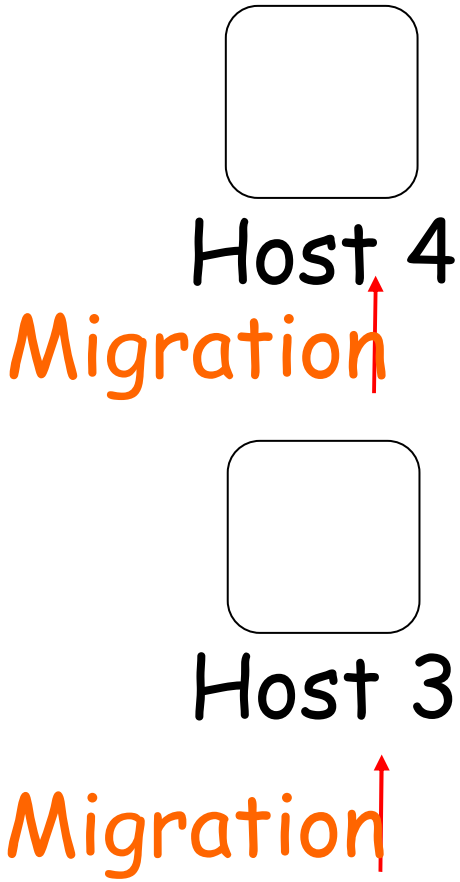
itineraryRequestFirst ();

// Serving the pending requests upon arrival before migrating again

Dynamic itineraries

Destination	Methods
Host 1	echo
Host 2	callhome
Host 3	processData
Host 4	foo





Communicating with mobile objects

Ensuring communication in presence of migration

Should be transparent (i.e. nothing in the application code)

Impact on performance should be limited or well known

ProActive provides 2 solutions to choose from at object creation

- Location Server
- Forwarders

also, it is easy to add new ones!

Forwarders

Migrating object leaves forwarder on current site

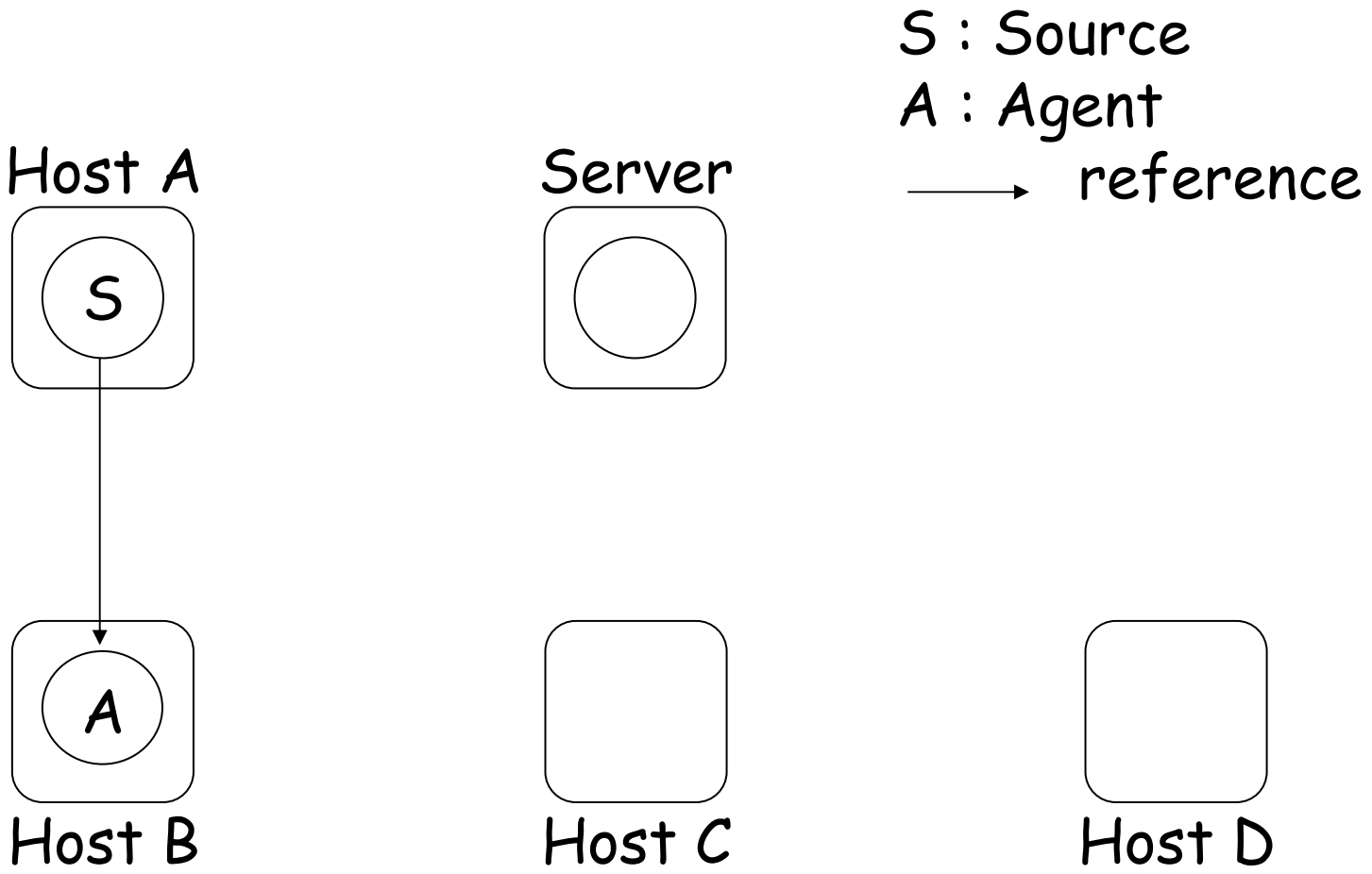
Forwarder is linked to object on remote site

- Possibly the mobile object
- Possibly another forwarder => a *forwarding chain* is built

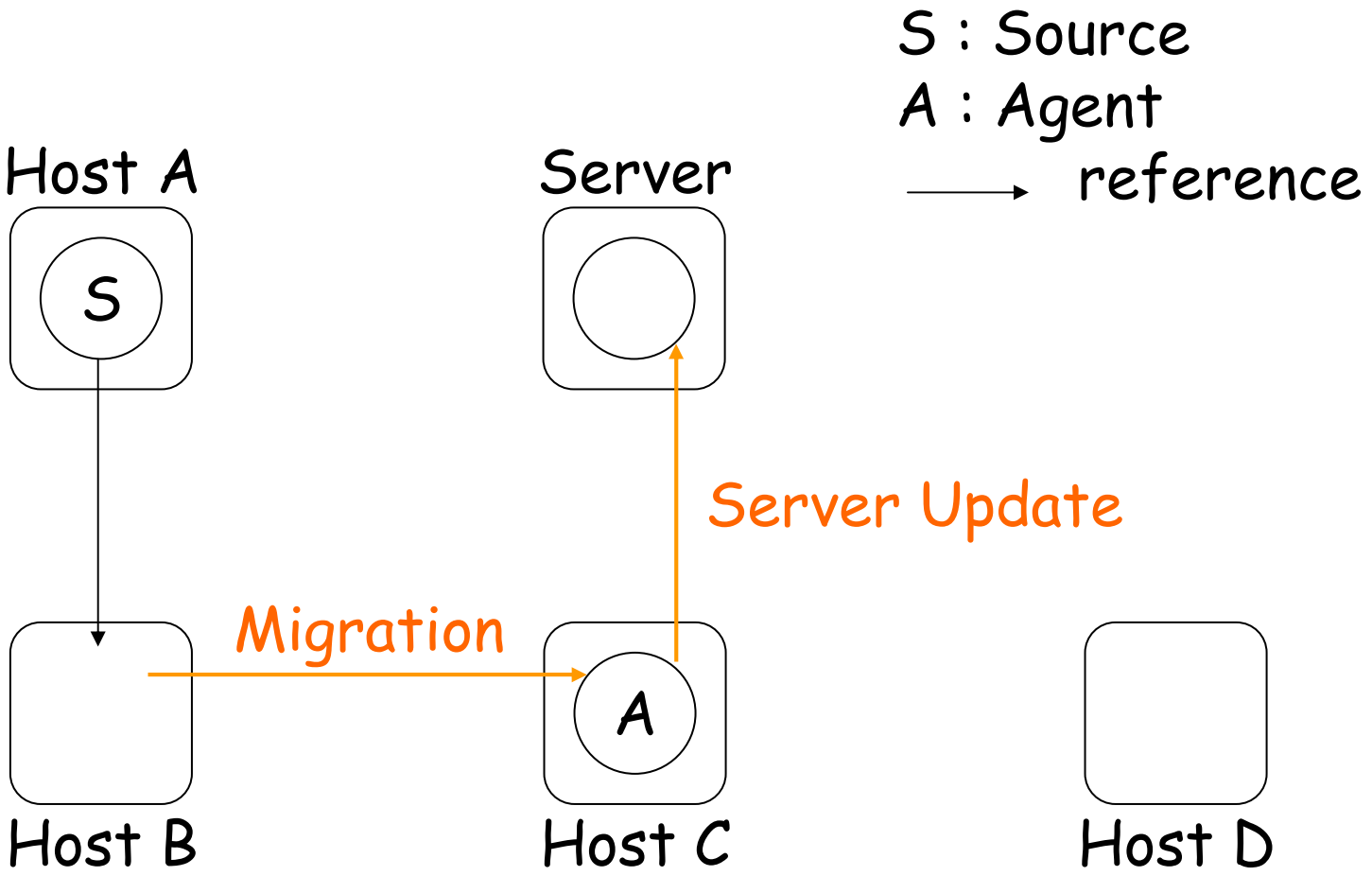
When receiving message, forwarder sends it to next hop

Upon successful communication, a tensioning takes place

Other Strategy: Centralized (location Server)

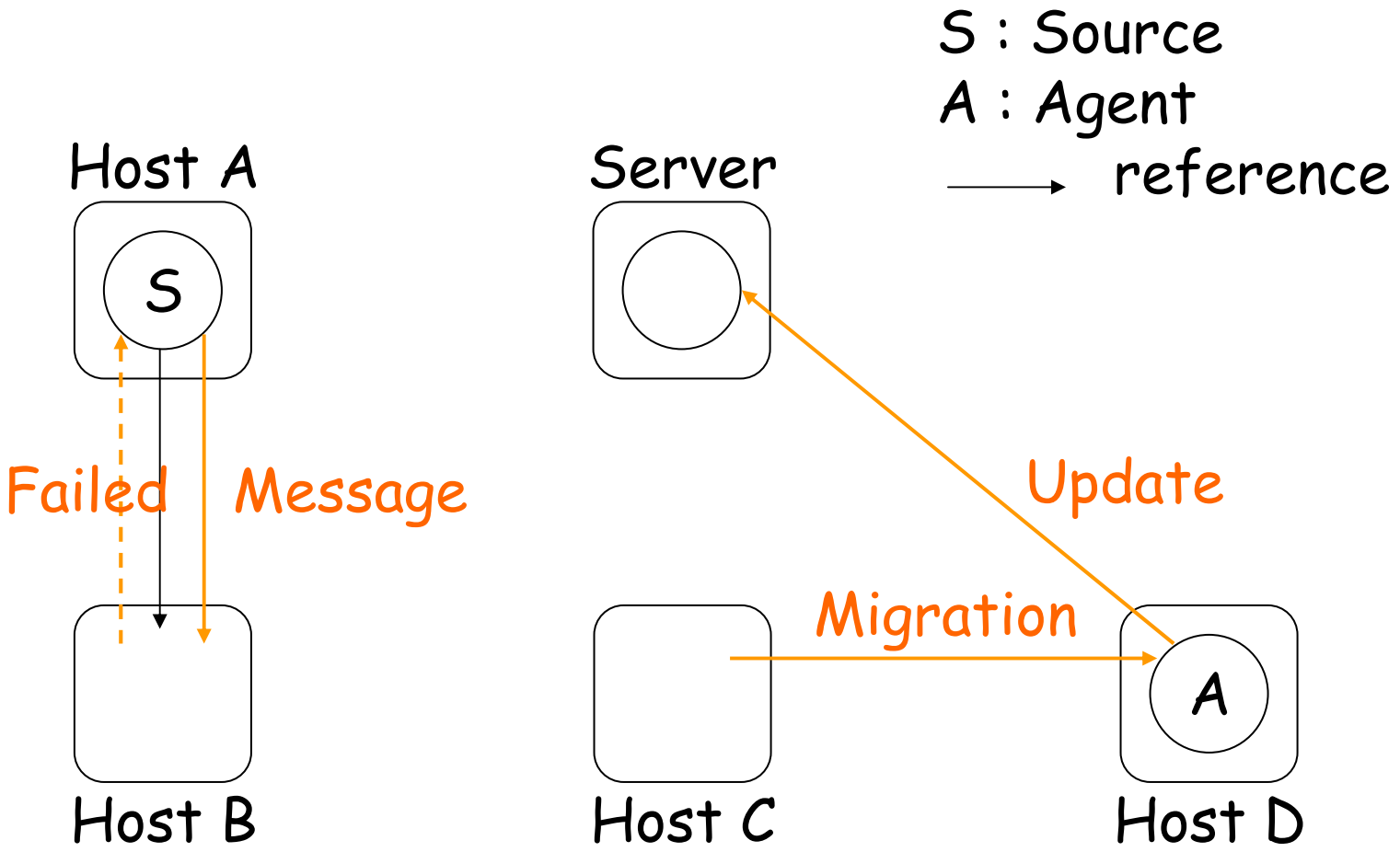


Centralized Strategy (2)



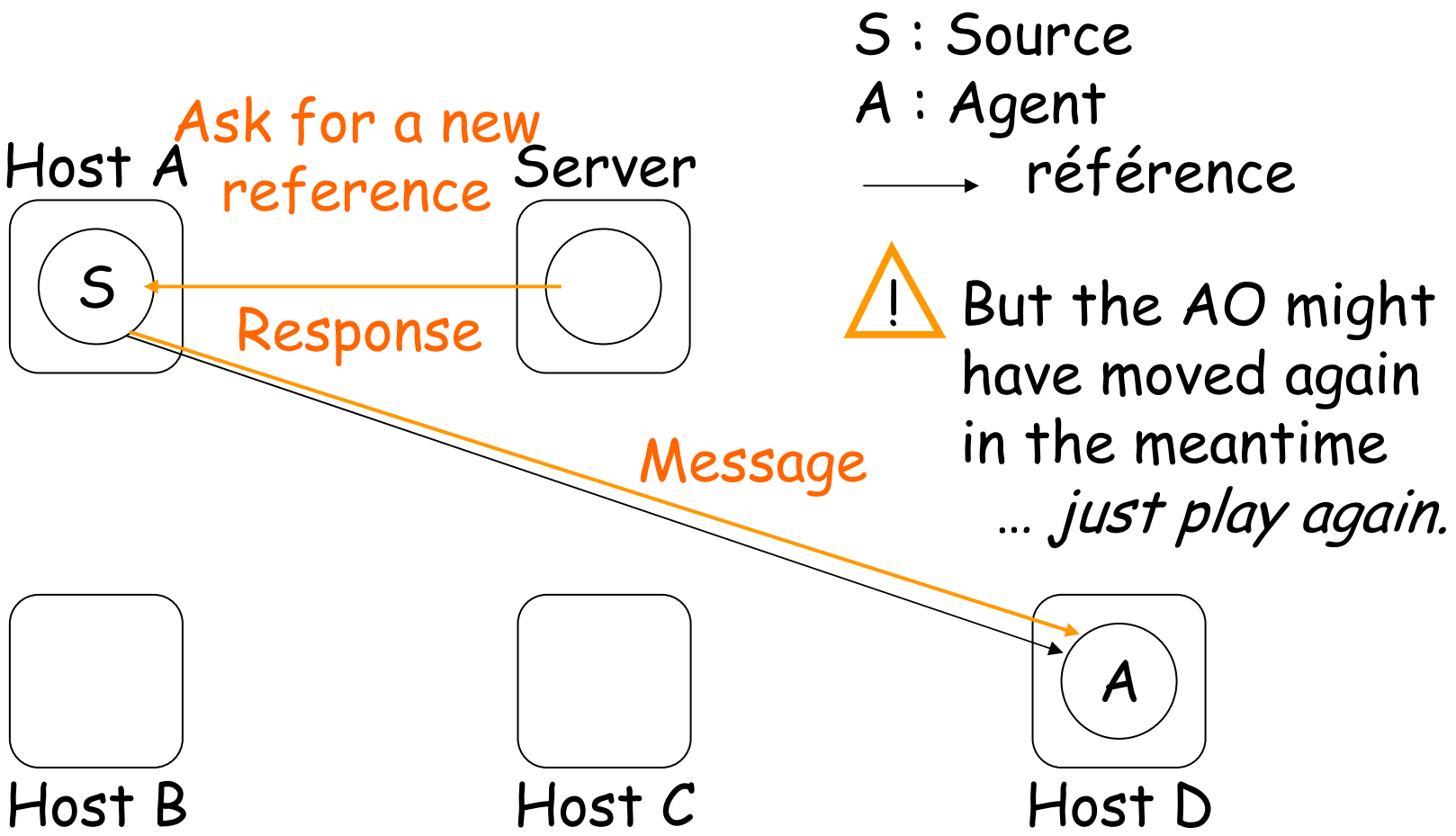
A migrating object updates the server

Centralized Strategy (3)



A migrating object updates the server

Centralized Strategy (4)



The source get a new reference from the server

Location Server vs Forwarder

Server

- No fault tolerance if single server
- Scaling is not straightforward
- Added work for the mobile object
- The agent can run away from messages

Forwarders

- Use resources even if not needed
- The forwarding chain is not fault tolerant
- An agent can be lost

What about performance?

Impact on performance

A Simple test

Measure the time for a simple call on a mobile object
no parameters, no return value

Application made of 2 objects: The source and the Agent

Source

Does not migrate

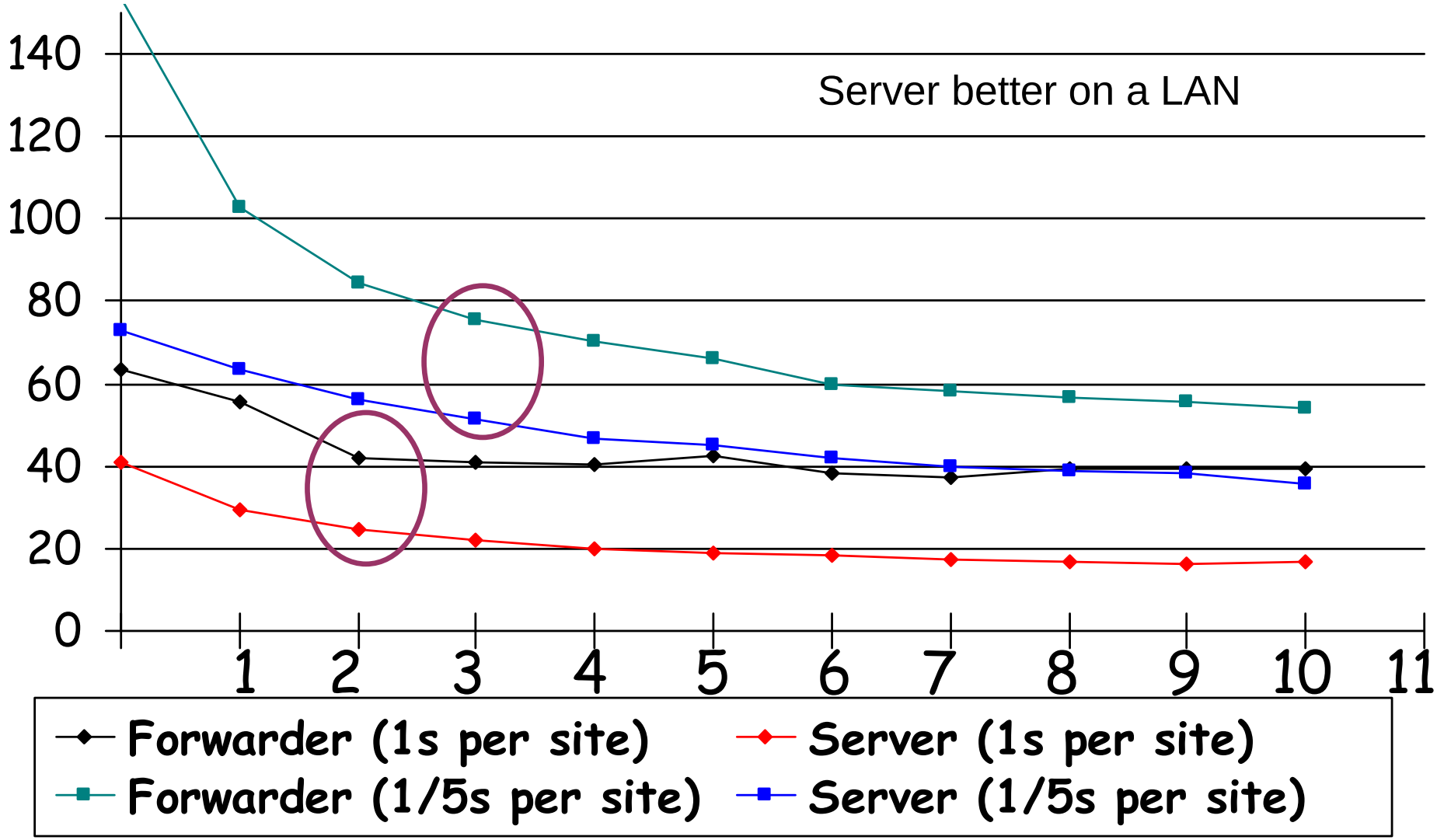
Wait an average time before calling the Agent (1/communication rate)

Agent

Wait an average time on a site before migrating (1/migration rate)

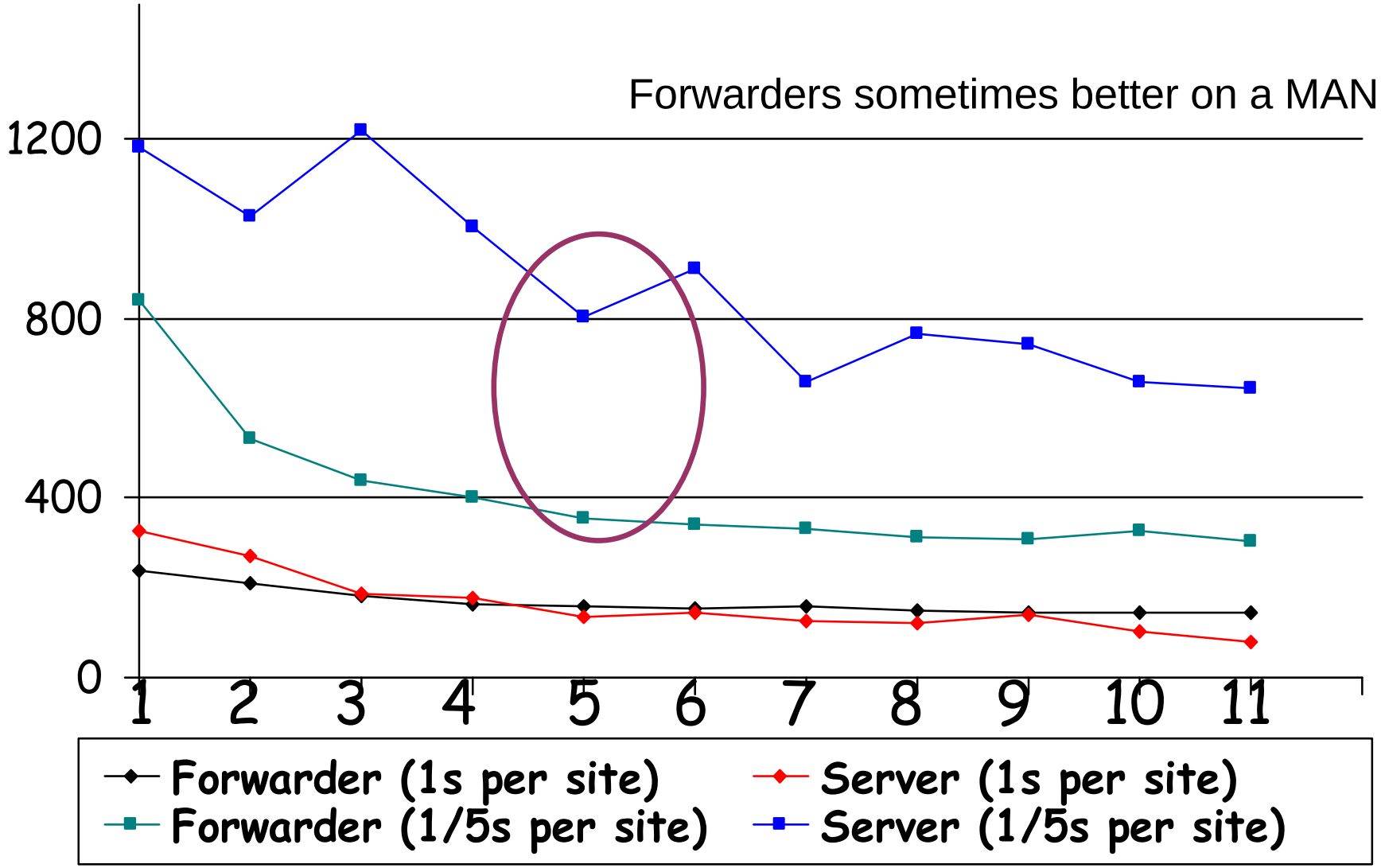
Forwarder vs. Server LAN (100 Mb/s)

Response time (ms) vs. Communication rate



Forwarder vs Server MAN (7 Mb/s)

Response time (ms) vs. Communication rate



On the cost of the communication

Server:

- The agent must call the server => the migration is longer
- Cost for the source:
  - Call to site where the agent was
  - Call to the server and wait for the reply
  - Call to the (maybe) correct location of the agent

Forwarder:

- The agent must create a forwarder (< to calling server)
- Cost for the source:
  - Follow the forwarding chain
  - Cost of the tensioning (1 communication)

Conclusion

Weak Migration of any active object

Communications using two schemes: server and forwarders

Current applications:

- Network Administration
- Desktop to Laptop

Perspective: Taking the best of the forwarders and the server

- Forwarder with limited lifetime
- Server as a backup solution

TTL-TTU mixed parameterized protocol

TTL: Time To Live + Updating Forwarder:

5 s.

After TTL, a forwarder is subject to self destruction

Before terminating, it updates server(s) with last agent known location

TTU: Time To Update mobile AO:

After TTU, AO will inform a localization server(s) of its current location

Dual TTU: first of two events:

maxMigrationNb: the number of migrations without server update

maxTimeOnSite: the time already spent on the current site

10

5 s.

Conclusion on Mobile Active Objects

AO = a good unit of Computational Mobility

Weak Migration OK (even for Load Balancing)

Both Actors and Servers

Ensuring communications: several implementation to choose from:

- Location Server
- Forwarders
- Mixed: based on TTL-TTU

Primitive + Higher-Level abstractions:

- migrateTo (location)
- onArrival, onDeparture
- Itinerary, etc.

Performance Evaluation of Mobile Agent

Together with Fabrice Huet and Mistral Team

Objectives:

Formally study the performance of Mobile Agent localization mechanism

Investigate various strategies (forwarder, server, etc.)

Define adaptative strategies

1-

Analyse Markovienne des répéteurs

Paramètre	Description
$1/\lambda$	Temps moyen d'inactivité de la source
$1/\nu$	Temps moyen d'inactivité de l'agent
$1/\delta$	Durée moyenne de migration
$1/\gamma$	Délai moyen inter-sites

TAB. 4.1 – *Paramètres de la modélisation du mécanisme des répéteurs*

2-

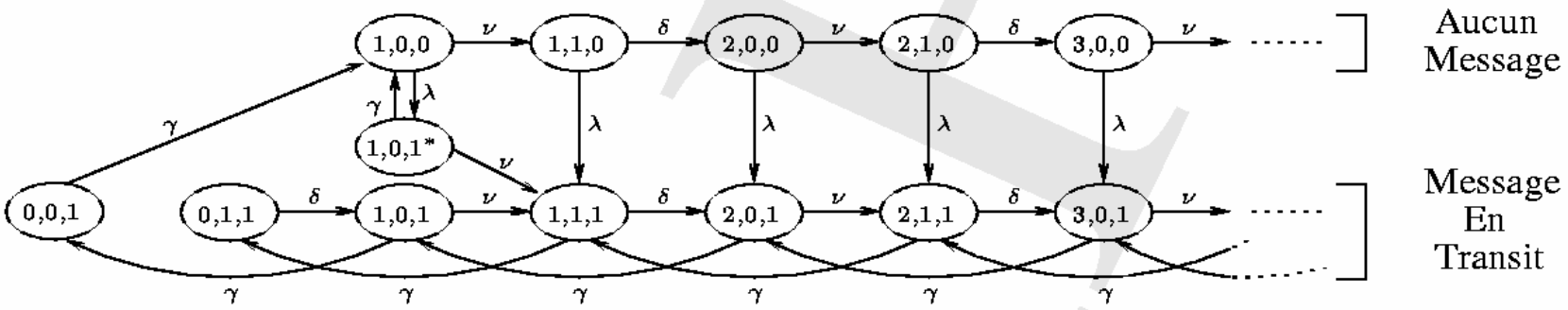


FIG. 4.2 – *États et transitions du mécanisme des répéteurs*

3 -

Départ	Transition	Arrivée	Description
(1,0,0)	$\xrightarrow{\nu}$	(1,1,0)	Début de migration
	$\xrightarrow{\lambda}$	(1,0,1*)	Nouveau message généré
$(i,0,0)$ avec $i \geq 2$	$\xrightarrow{\nu}$	$(i,1,0)$	Début de migration
	$\xrightarrow{\lambda}$	$(i,0,1)$	Nouveau message généré
$(i,1,0)$ avec $i \geq 1$	$\xrightarrow{\delta}$	$(i+1,0,0)$	Fin de migration
	$\xrightarrow{\lambda}$	$(i,1,1)$	Nouveau message généré
(1,0,1*)	$\xrightarrow{\nu}$	(1,1,1)	Début de migration
	$\xrightarrow{\gamma}$	(1,0,0)	Message a atteint l'agent
$(i,0,1)$ avec $i \geq 1$	$\xrightarrow{\nu}$	$(i,1,1)$	Début de migration
	$\xrightarrow{\gamma}$	$(i-1,0,1)$	Message a effectué un saut
$(i,1,1)$ avec $i \geq 1$	$\xrightarrow{\delta}$	$(i+1,0,1)$	Fin de migration
	$\xrightarrow{\gamma}$	$(i-1,1,1)$	Message a effectué un saut
(0,1,1)	$\xrightarrow{\delta}$	(1,0,1)	Fin de migration
(0,0,1)	$\xrightarrow{\gamma}$	(1,0,0)	Message a atteint la source

TAB. 4.2 – *Détails des transitions dans le modèle des répéteurs*

4 -

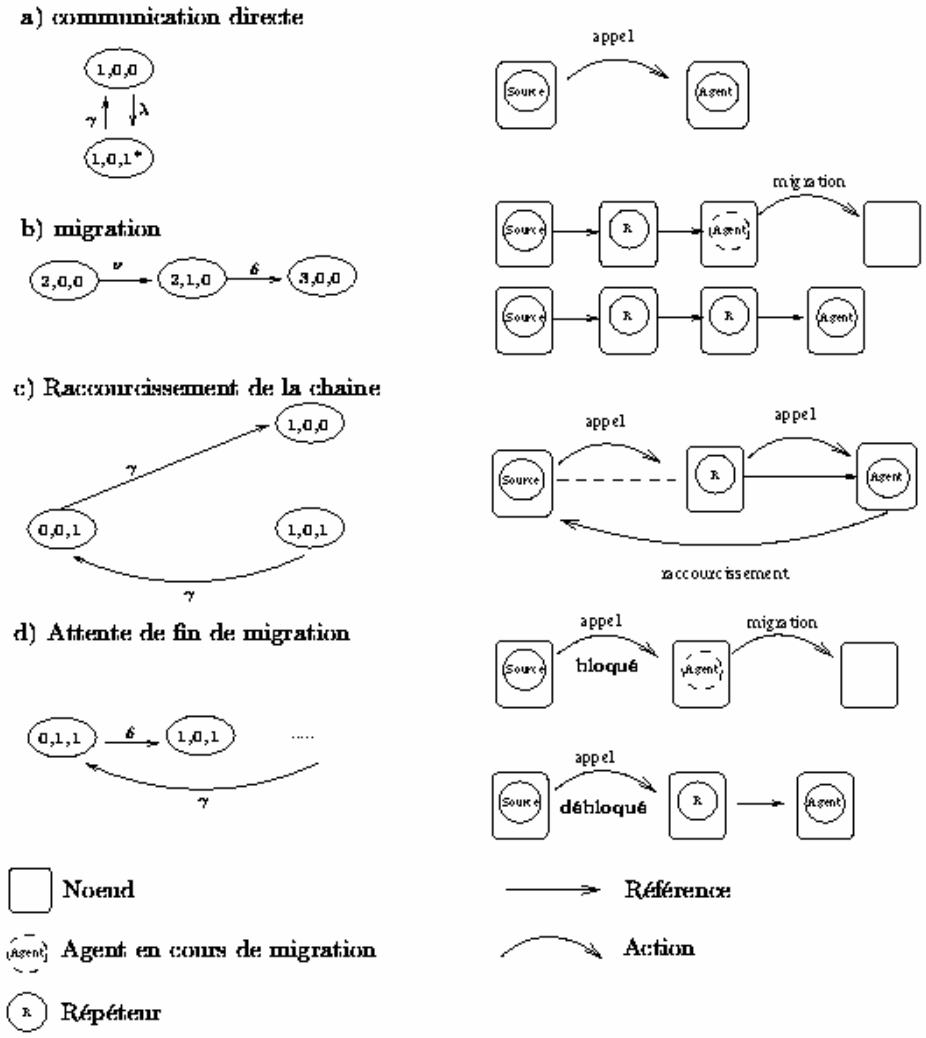


FIG. 4.3 – *Détails des transitions du diagramme des répéteurs*

5 -

Analyse Markovienne du serveur centralisé

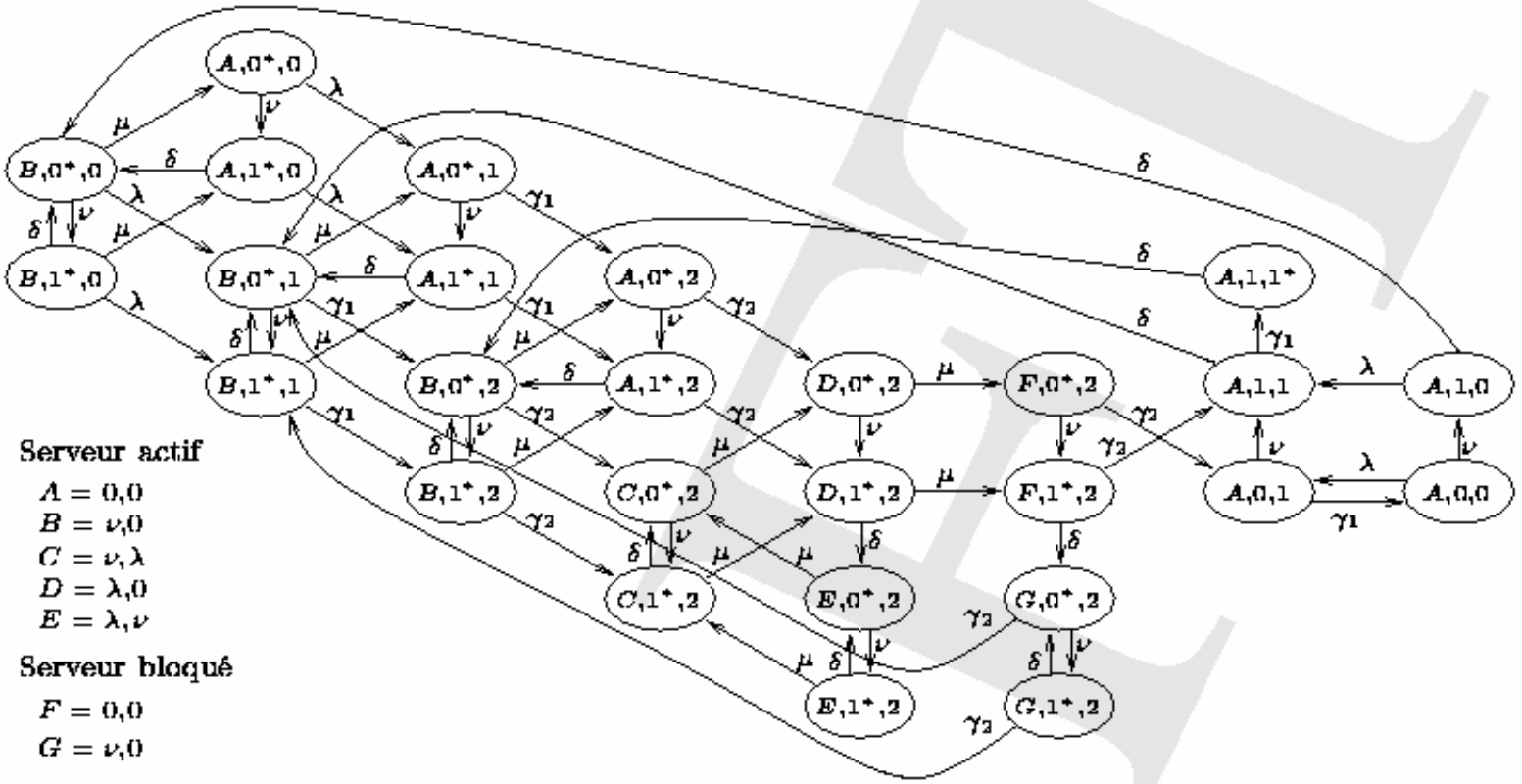


FIG. 4.6 – *État du système et taux de transitions dans le mécanisme du serveur.*

6 -

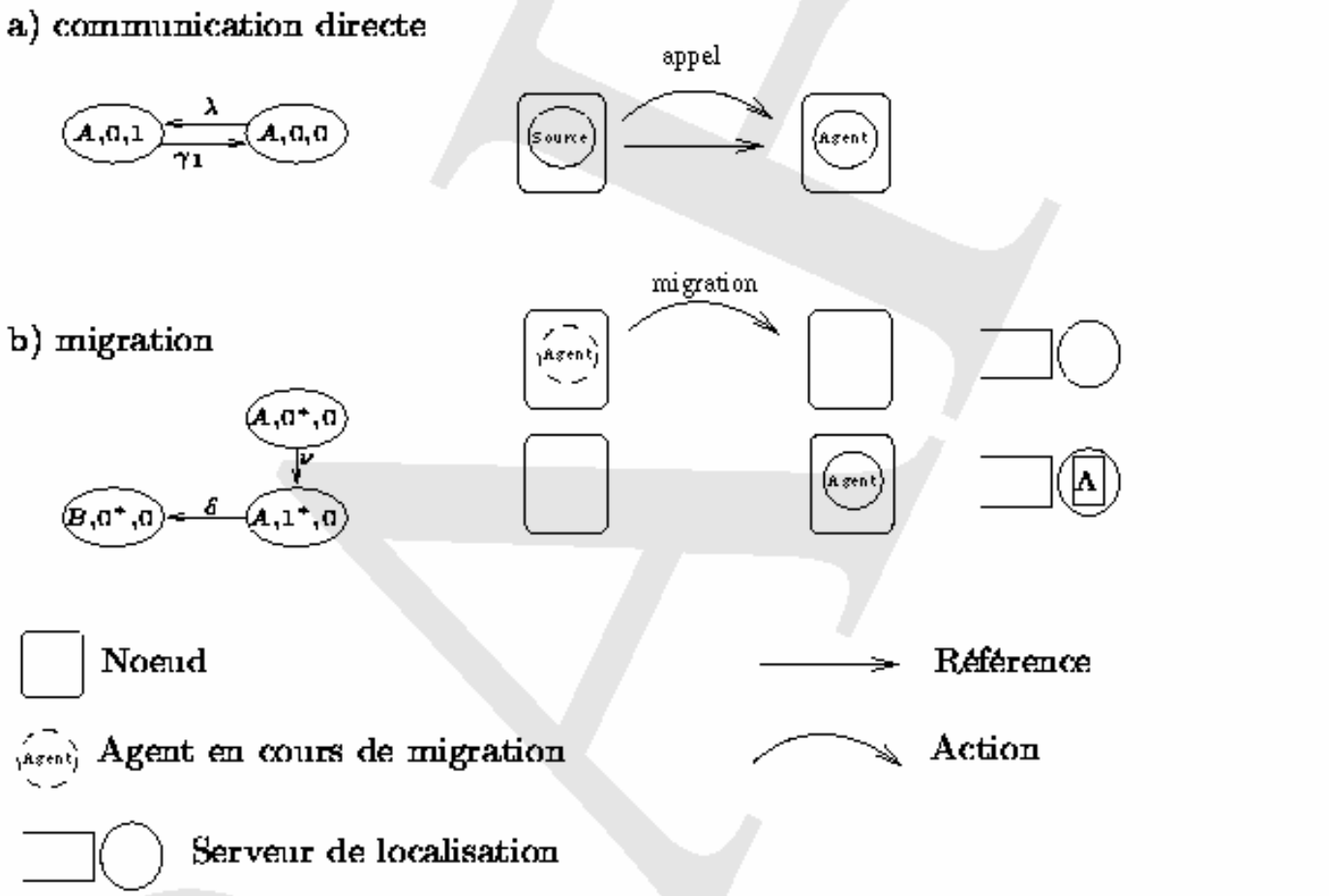


FIG. 4.7 – *Détails des transitions du modèle du serveur - source et agent*

7 -

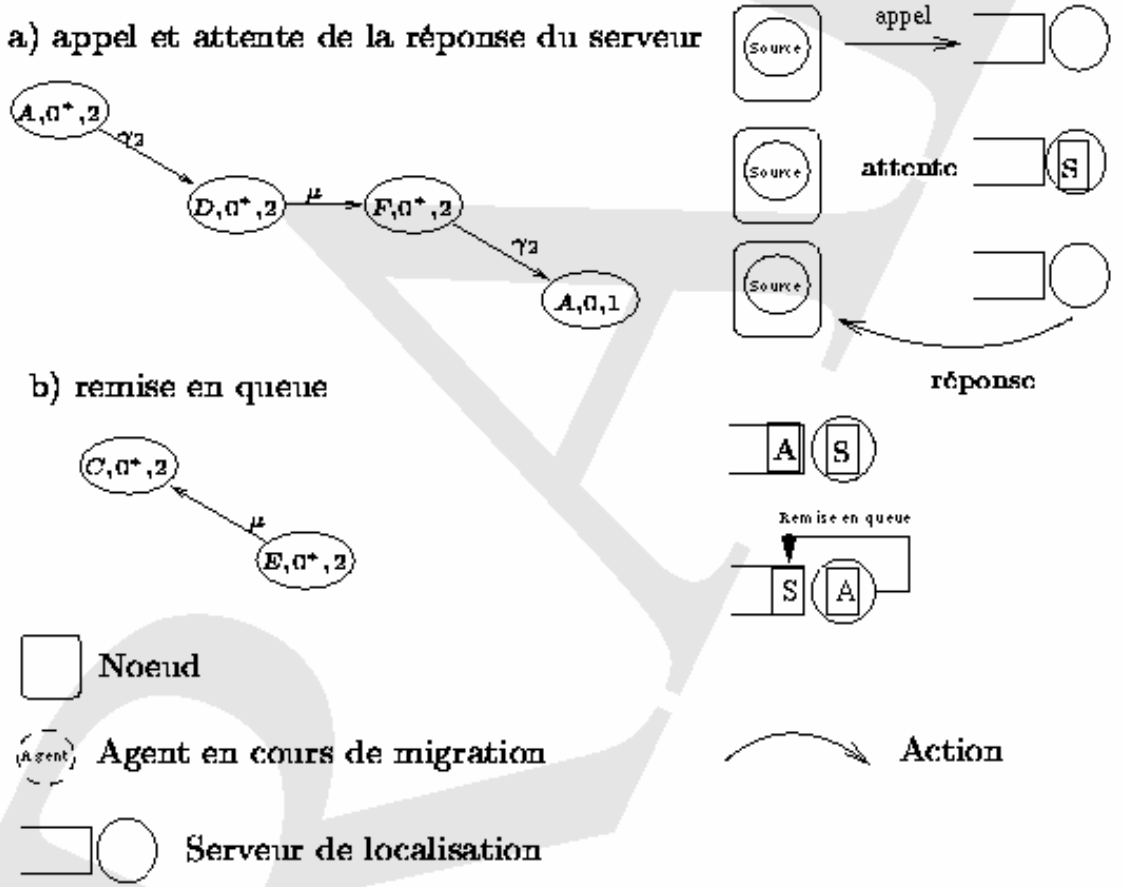


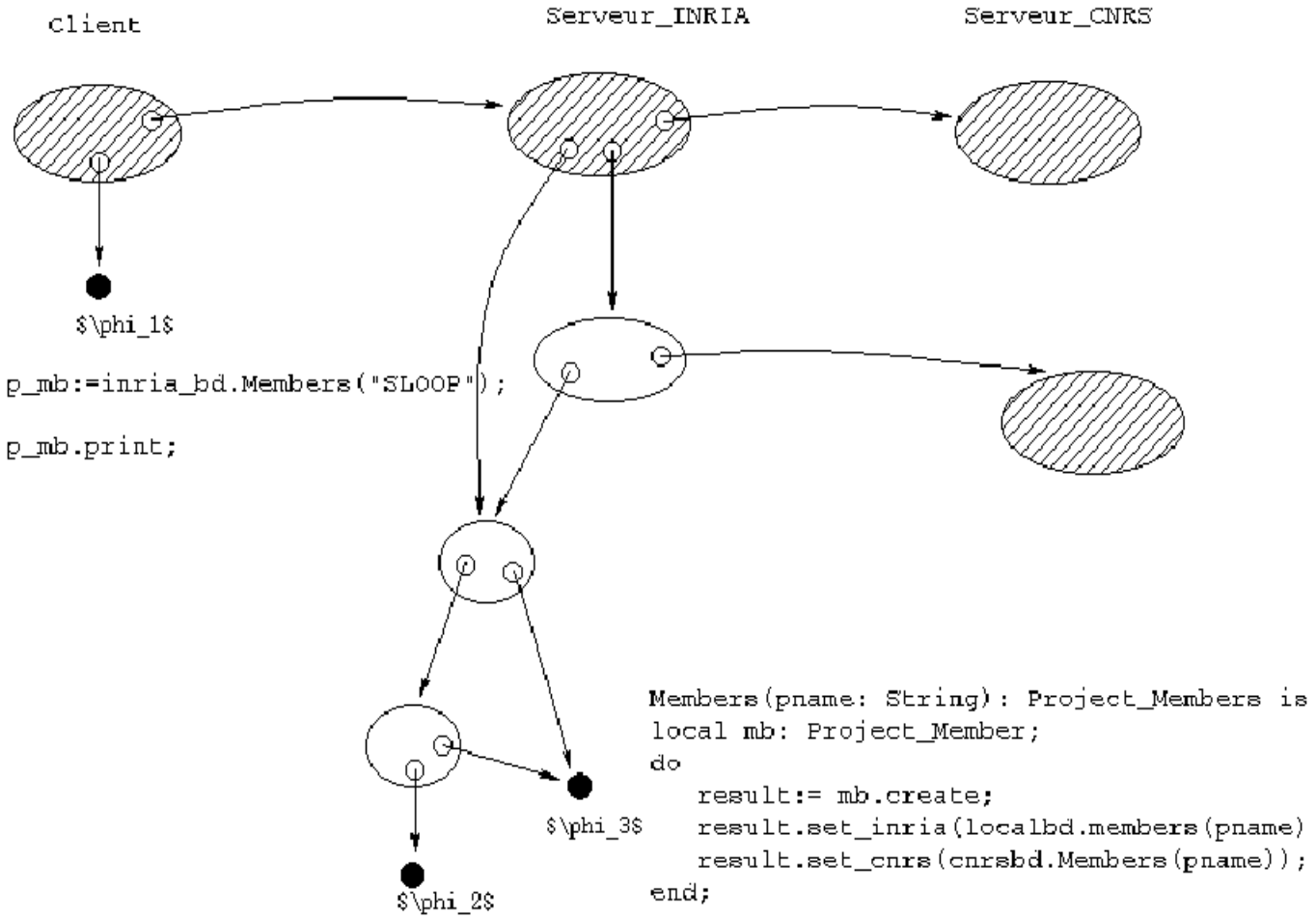
FIG. 4.8 – *Détails des transitions du modèle du serveur - le serveur*

8 -

Paramètre	Description
λ	Attente de la source
ν	Attente de l'agent
δ	Inverse de la durée de migration
γ_1	Inverse de la latence vers l'agent
γ_2	Inverse de la latence vers le serveur
μ	Taux de service

TAB. 4.5 – *Description des paramètres de la modélisation du serveur de localisation*

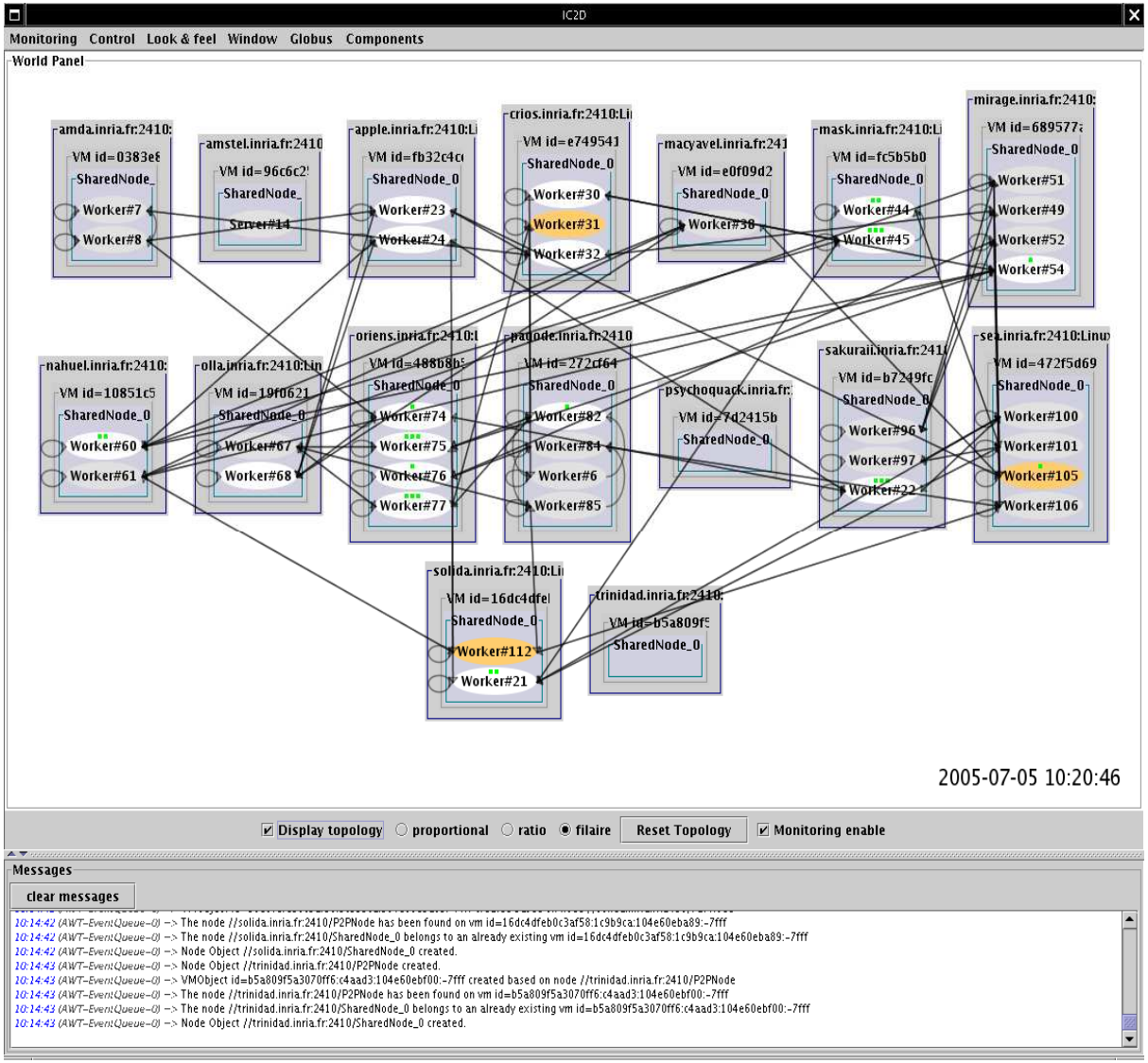
Automatic Continuations



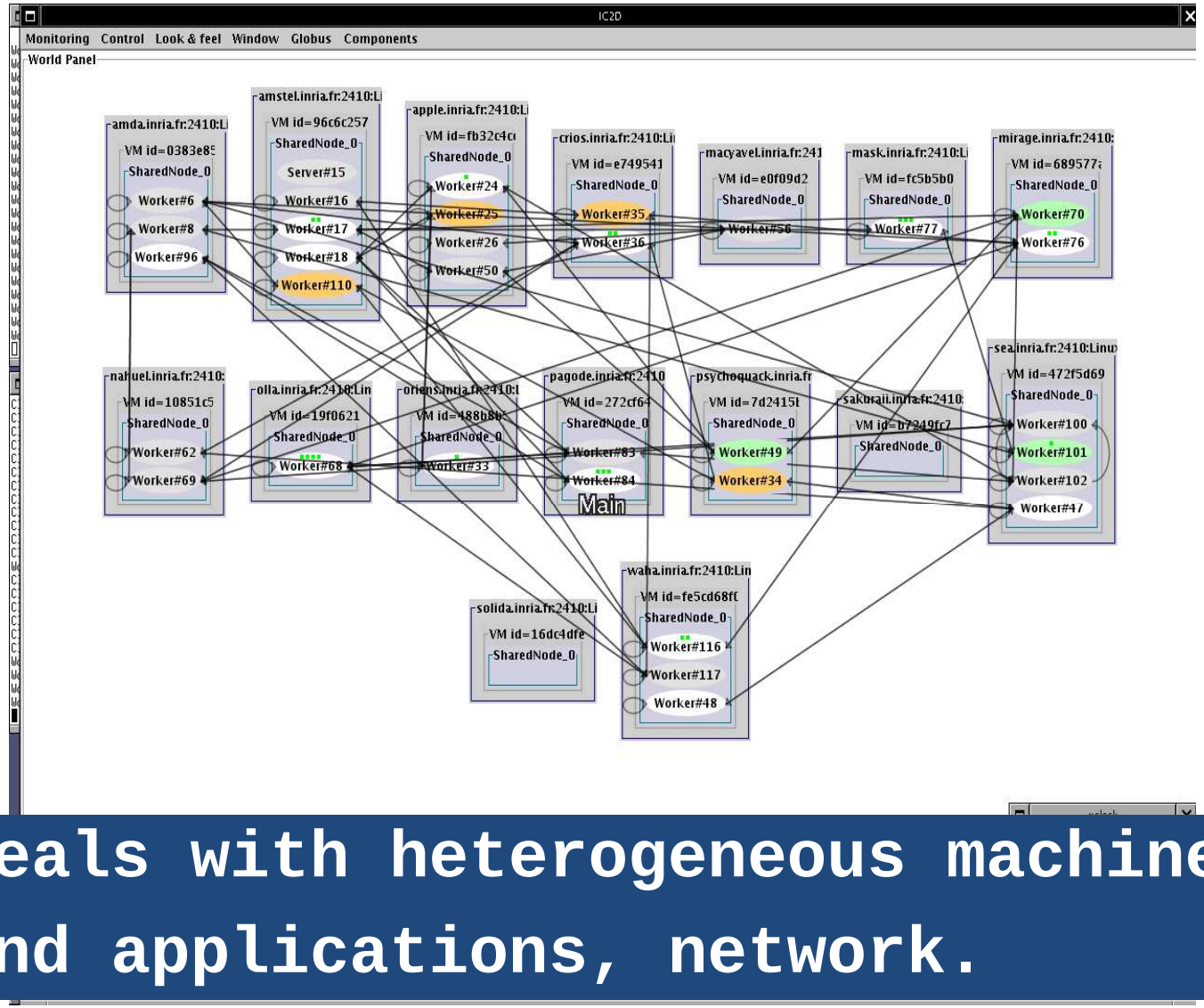
Transparent Future transmissions (Request, Reply)

Load Balancing

Load Balancing using Mobility (1)



Load Balancing using Mobility (2)



Deals with heterogeneous machines, and applications, network.

Master / Workers

Motivations

Embarrassingly parallel problems : simplest and frequent model

Write embarrassingly parallel applications with ProActive :

- May require a sensible amount of code (fault-tolerance, load-balancing, …). Example : NQueenManager.java, 1074 lines of code !
- Requires understanding of ProActive concepts (Futures, Stubs, Group Communication) => can take a while

Goals of the M/S API

Provide a easy-to use framework for solving embarrassingly parallel problems:

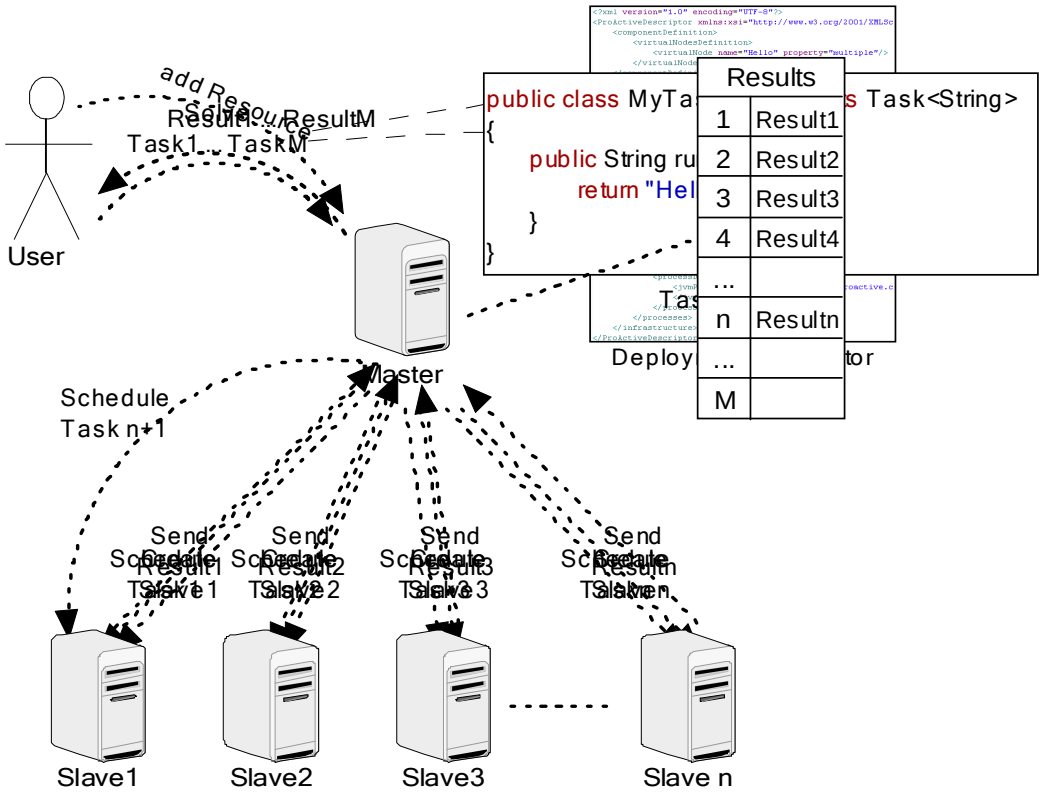
- Simple Task definition
- Simple API interface (few methods)
- Simple & efficient solution gathering mechanism

Provide automatic fault-tolerance and load-balancing mechanism

Hide ProActive concepts from the user.

Principles

How it works ?


User
Master
Slave1
Slave2
Slave3
Slave n
Schedule
Task n+1
Results
1 Result1
2 Result2
3 Result3
4 Result4
n Resultn
M
Task<String>

API

Creating/Terminating the Master

Local creation :

```
public ProActiveMaster();
```

Remote creation :

```
public ProActiveMaster(Node remoteNodeToUse);

public ProActiveMaster(URL descriptorURL, String
   masterVNName);
```

Termination :

```
public void terminate(boolean freeResources);
```


Adding Resources

```
public void addResources(URL descriptorURL, String
   virtualNodeName);

public void addResources(VirtualNode virtualNode);

public void addResources(Collection<Node> nodes);

public void addResources(URL descriptorURL);
```

Task Definition / Submission

Task definition :

```
public interface Task<R extends Serializable>
   extends Serializable {
   public R run(WorkerMemory memory) throws
   Exception;
}
```

Task submission :

```
 public void solve(List<T> tasks)
        throws TaskAlreadySubmittedException;
```

Result reception order :

```
 public void setResultReceptionOrder(OrderingMode
```


Retrieving the Results

Blocking calls :

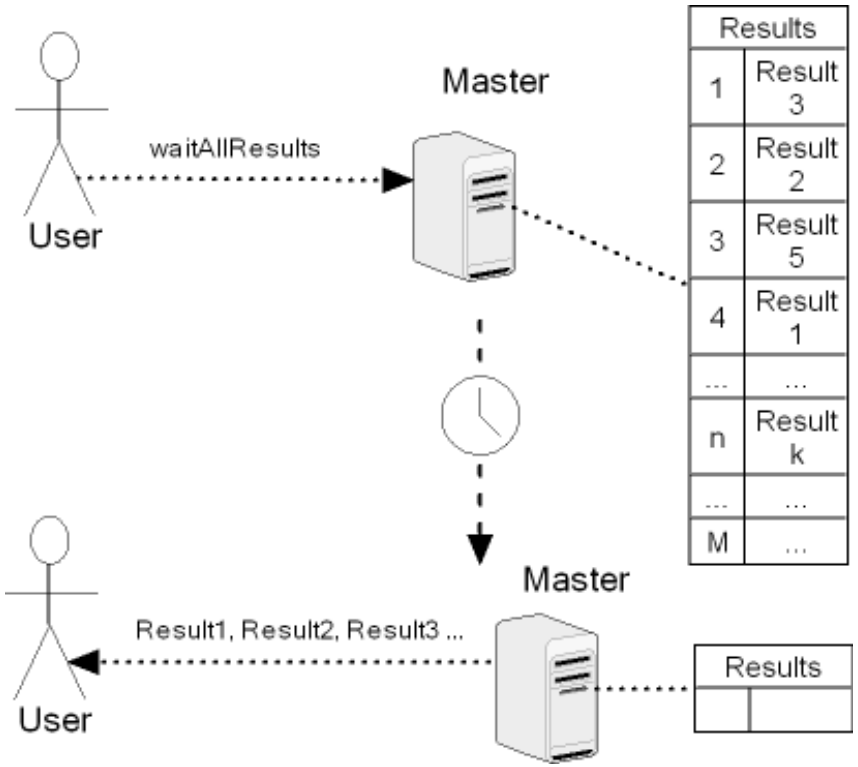
```
public List<R> waitAllResults() throws
   TaskException;

public List<R> waitKResults(int k) throws
   TaskException;

public R waitOneResult() throws TaskException;
```

Non-Blocking calls :

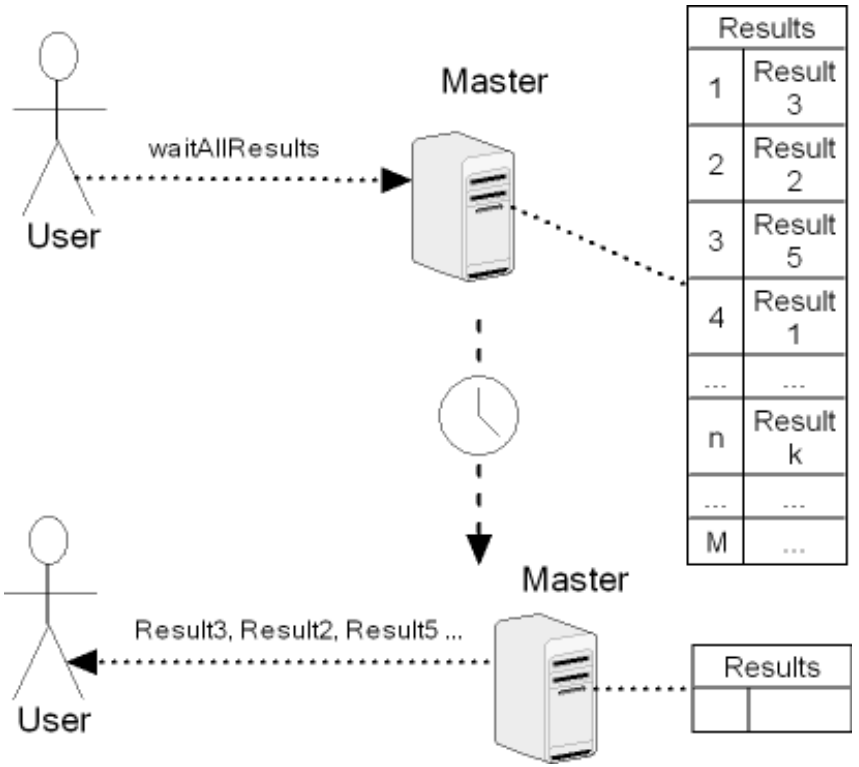
```
public boolean isEmpty();

public int countAvailableResults();
```

Results reception order

SubmissionOrder Mode

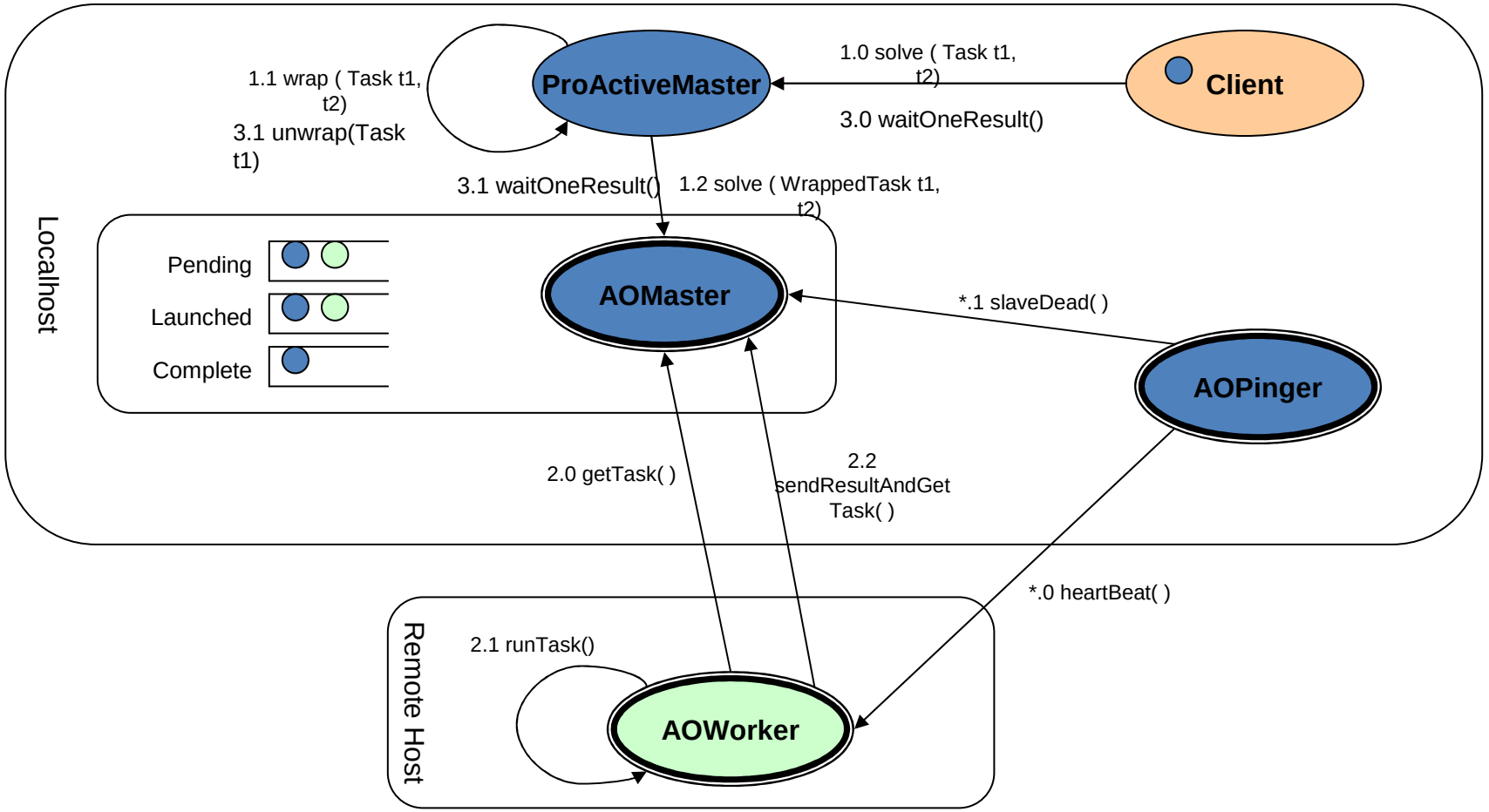


CompletionOrder Mode



Design

Internal Behavior



Example : Calculating PI(Monte-Carlo)

Task definition

```
public static class ComputePIMonteCarlo implements Task<Long> {
        public ComputePIMonteCarlo() {
        }

        public Long run(WorkerMemory memory) throws Exception {
            long remaining = NUMBER_OF_EXPERIENCES;
            long successes = 0;
            while (remaining > 0) {
                remaining--;
                if (experience()) {
                    successes++;
                }
            }
            return successes;
        }

        public boolean experience() {
            double x = Math.random();
            double y = Math.random();
            return Math.hypot(x, y) < 1;
```


Creating the Master

```
// creation of the master
ProActiveMaster<ComputePIMonteCarlo, Long> master =
  new ProActiveMaster<ComputePIMonteCarlo, Long>();

// adding resources
master.addResources(PIExample.class.getResource(
  "/org/objectweb/proactive/examples/masterslave/Wor
  kersLocal.xml"));
```

Creating the tasks and submitting

```
// defining tasks
Vector<ComputePIMonteCarlo> tasks = new
  Vector<ComputePIMonteCarlo>();
for (int i = 0; i < NUMBER_OF_TASKS; i++) {
    tasks.add(new ComputePIMonteCarlo());
}

// adding tasks to the queue
master.solve(tasks);
```

Collecting the results /Finishing

```
// waiting for results
List<Long> successesList = master.waitAllResults();

// computing PI using the results
long sumSuccesses = 0;

for (long successes : successesList) {
     sumSuccesses += successes;
}

double pi = (4 * sumSuccesses) / ((double)
 NUMBER_OF_EXPERIENCES * NUMBER_OF_TASKS);

System.out.println("Computed PI by Monte-Carlo
 method : " + pi);
```

Benchmarks

Experiment:

On Grid5000 Helios (x4 proc)

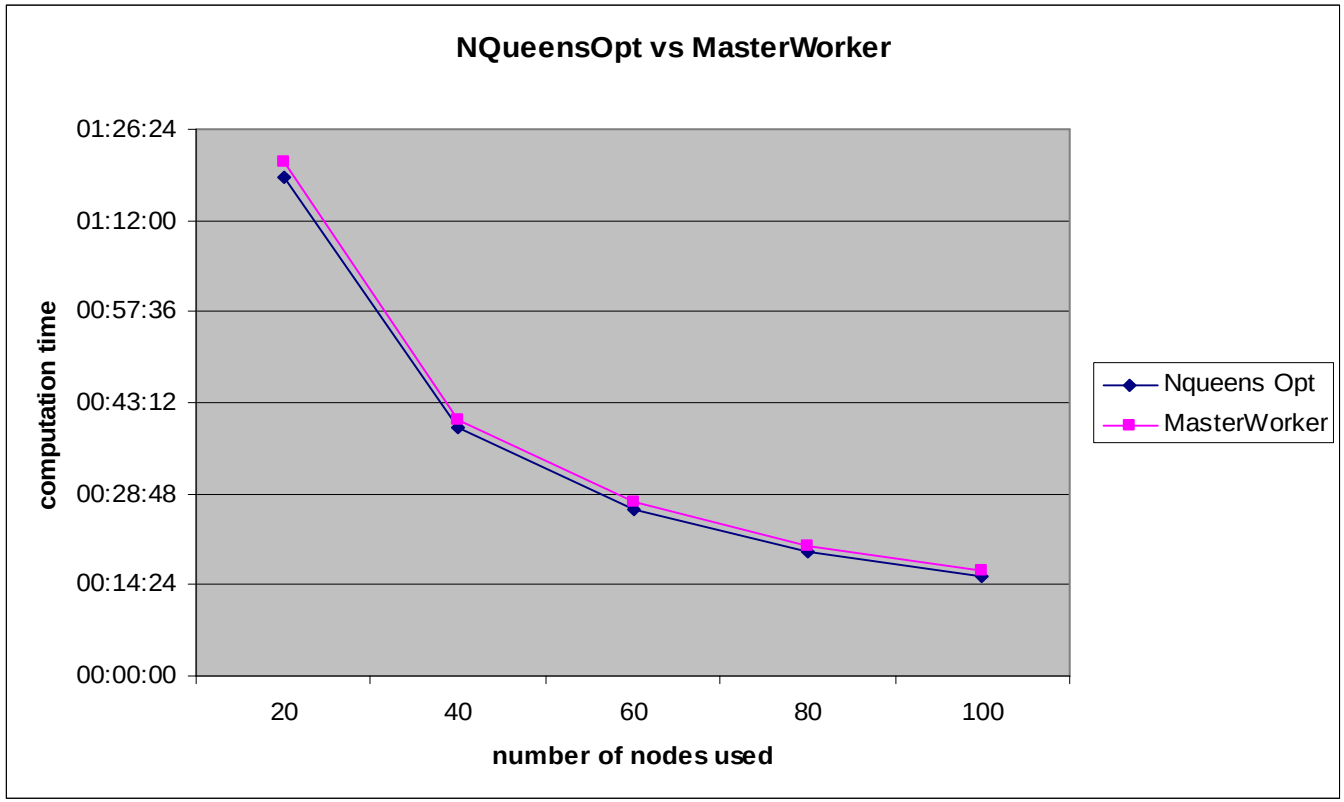
NQueens (Optimized Version) vs MasterWorker API (on the same algorithm)

Very small tasks, but a lot of tasks (large nb of comms)

Runs

On 5,10, 15, 20, 25 nodes (20, 40, 60, 120, 150 slaves)

Results


NQueensOpt vs MasterWorker
01:26:24
01:12:00
00:57:36
00:43:12
00:28:48
00:14:24
00:00:00
computation time
20
40
60
80
100
number of nodes used
Nqueens Opt
MasterWorker

Results(2)


Efficiency loss
6,00%
5,00%
4,00%
3,00%
2,00%
1,00%
0,00%
(MWtime - NQtime) / NQtime
20
40
60
80
100
number of nodes

Conclusion

Conclusion

Current use:

Sylvain Cussat-Blanc IRIT / Université Toulouse I

Future Work:

Hierarchical Master/Worker

Parametric Sweep version

Componentized version

Parallel, Distributed, Hierarchical

3. Components

for the Grid

Composing

3.1 Component Introduction

Componentizing software

Douglas McIlroy :

Mass Produced Software Components, 1968

Component - What is it ?

A Component = a unit of and

From Objects (Classes) to Components:

Objects:

Programming in the small

Composition by programming (Inheritance, Instantiation/Aggregation)

Components:

Building software in the large

Tools for assembling and configuring the execution

Component = a module (80s!) but subject to:

- Needs, Configuration (variation on Non Functional Properties)
- Instantiation, life Cycle management

To be deployed on various platforms (some portability)

Characteristics -- How ?

How it works --- Common characteristics

A standardized way to describe a component:

a specification of what a component does:

- Provide (Interfaces, Properties to be configured)
- Require (services, etc.)
- Accept as parameterization

Usually dynamic discovery and use of components:

Auto-description (Explicit information: text or XML, reflection, etc.)

Usually components come to life through several classes, objects

Legacy code: OO code wrapper to build components from C, Fortran, etc.

My Definition of Software Components

A component in a given infrastructure is:

a software module,

with a standardized description of what it needs and provides,

to be manipulated by tools for Composition and Deployment

A primitive example:
JavaBeans
Graphical components in Java

Quite simple :

a Java class (or several)

a naming convention to identify **properties:**

method: `public T getX ()`

method: `public void setX ()`

an attribute: `private T X = <default value>;`

a communication pattern: **Events, Source, Listeners**

and … a class is turned into a graphical component !

The Java introspection allows to discover dynamically the properties, and to configure them, assemble JB interactively

JavaBeans (2)

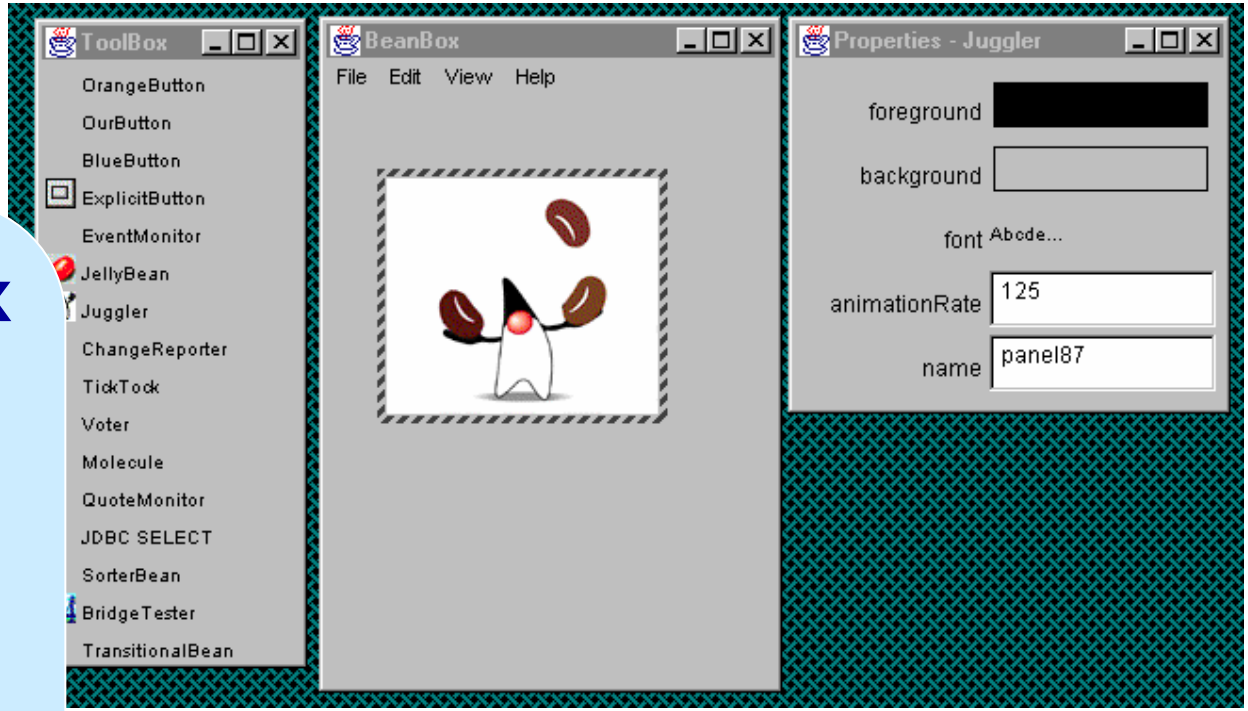
The BeanBox

So for JavaBeans:

Software module =
- Java Class

Standardized description =
- getX, setX, X,
- listeners

Tools:
- Composition = BeanBox
- Deployment = JVM



Nothing very new (cf. NeXTStep Interface Builder),
but life made a bit easier with byte code and introspection

Deploying and Executing Components

Components have to be configured on their Non-Functional Properties:

Functional Properties, Calls (Def.):

Application level services a component provides (e.g. Balance, Saxpy)

Non-Functional Properties, Calls (Def.):

The rest, mainly infrastructure services:

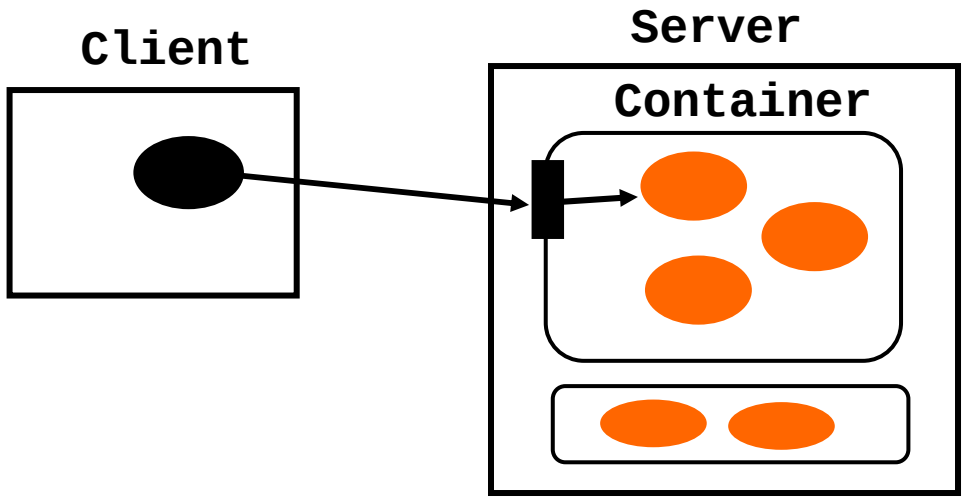
- Transaction, Security, Persistence, Remote/Asynchronous Com., Migration, …
- Start, Stop, Reconfiguration (location, bindings), etc.

so, Typical Infrastructure : Container for Isolation

Allows to manage and implement:

the non-functional properties

Life cycle of components



Enterprise Java Beans

A 3 tiers architecture (Interface, Treatment, DB), in Java

Objectives: ease development + deployment + execution

Java portability

A few concepts and definitions:

EJB Home object:

management of life cycle: creation, lookup, destruction, etc.

EJB Remote object:

Client view and remote reference

EJB object (or Bean):

the component itself (Session Stateless or Statefull, Entity Bean)

Functional Properties = Business Methods

Summary: Enterprise Java Beans

So for EJB:

Software module =

Java Classes and Interfaces (Home, Remote, Beans, …)

Only Provides (server),

no Uses

Standardized description =

a file with a standard format

Tools:

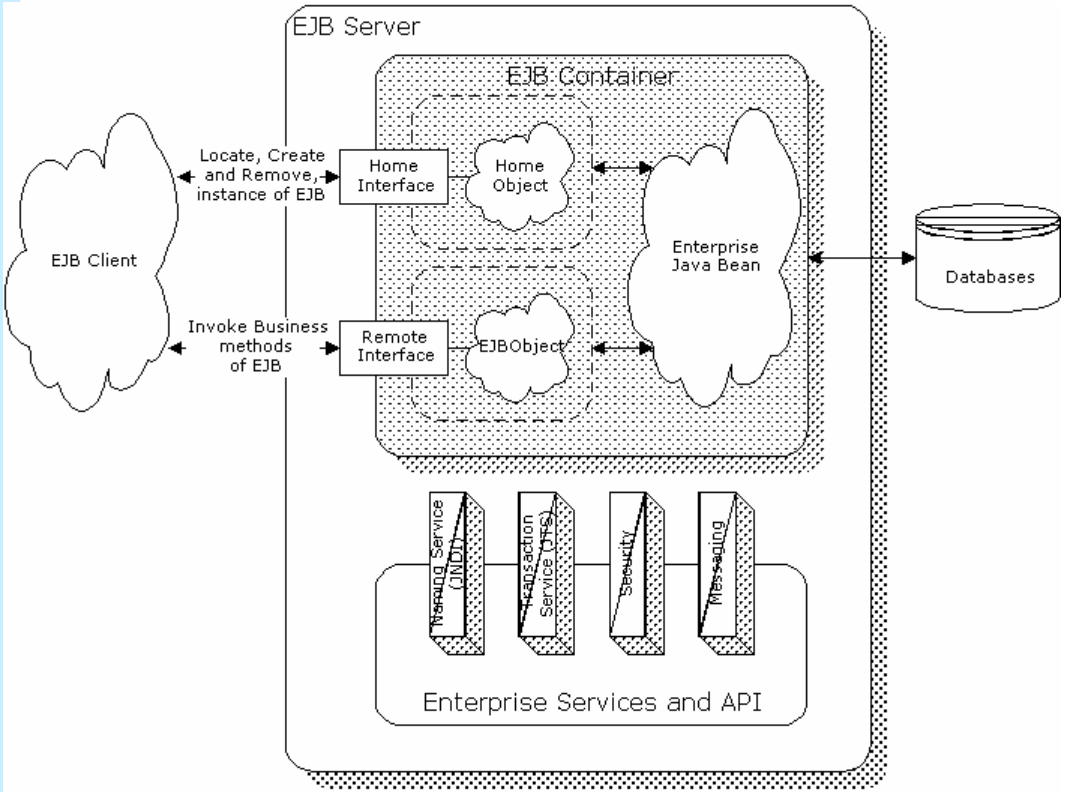
Composition = ? EJBrew ?

Deployment = JVM + RMI,

JTS, +

Generators +

Servers EJB



From www.tripod.com , G. S. Raj article

Components in Windows .Net

.Net basics:

A VM designed for several languages (C, C++, VB, + others)

CLR (Common Language Runtime)

CIL (Common Intermediate Language, MSIL) wider than ByteCode

– Boxing/Unboxing (value type <--> object), etc.

A new language: C#

An interactive tool (Visual Studio) to manipulate the “components”

A key choice: Extraction of description from program code

C# introduces language constructions for component information:

– Properties

– Attributes

– XML tags in source code (in Attributes)

Components in Windows .Net (2)

Example of Attributes, and Properties in C#:

```
[HelpUrl ("http://someUrl/Docs/SomeClass")]
class SomeClass {
private string caption;
public string Caption {
    get { return caption; }
    set { caption = value;
          Repaint (); }
}
```

An attribute: HelpUrl

It is actually a user define class (derive from Syst.Att.)

Attribute exists at RT.

A Property: Caption

JavaBeans in a language

Also: Indexed properties

Components for Web program. : WSDL (Web Services Description Lang.)

WSDL (Def. of Web callable methods) + Directories +

SOAP as wire format + Classes with Attributes and properties

Components in Windows .Net

Components characteristics:

Software module =

- Classes and Interfaces in various languages, but focus on C#

Standardized description =

- Still the COM, DCOM interfaces
- Extraction of Attributes, Properties **from source code!**
- WSDL

Tools:

- Composition = Visual Studio, etc.
- Deployment = Windows, .NET CLR

A Web Service: the instance of a component, … running...

Assembly of Components
Corba 3 and CCM

CCM: Corba Components Model =

EJB + a few things :

More types of Beans defined:

– **Service**, Session, **Process**, Entity, **...**

Not bound to Java (Corba)

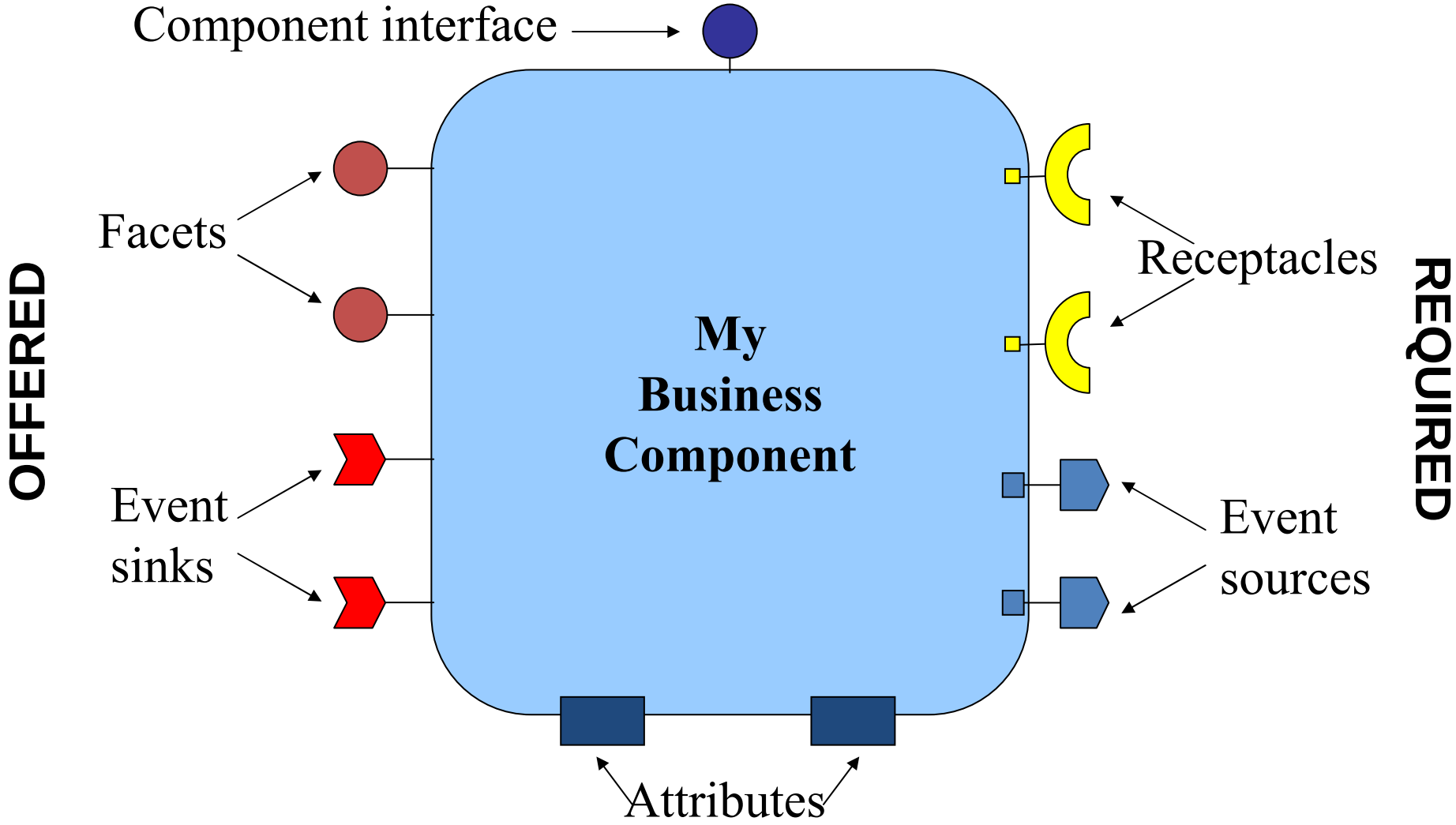
Uses :

– Specification of the component needs, dependencies

– “Client Interfaces’’

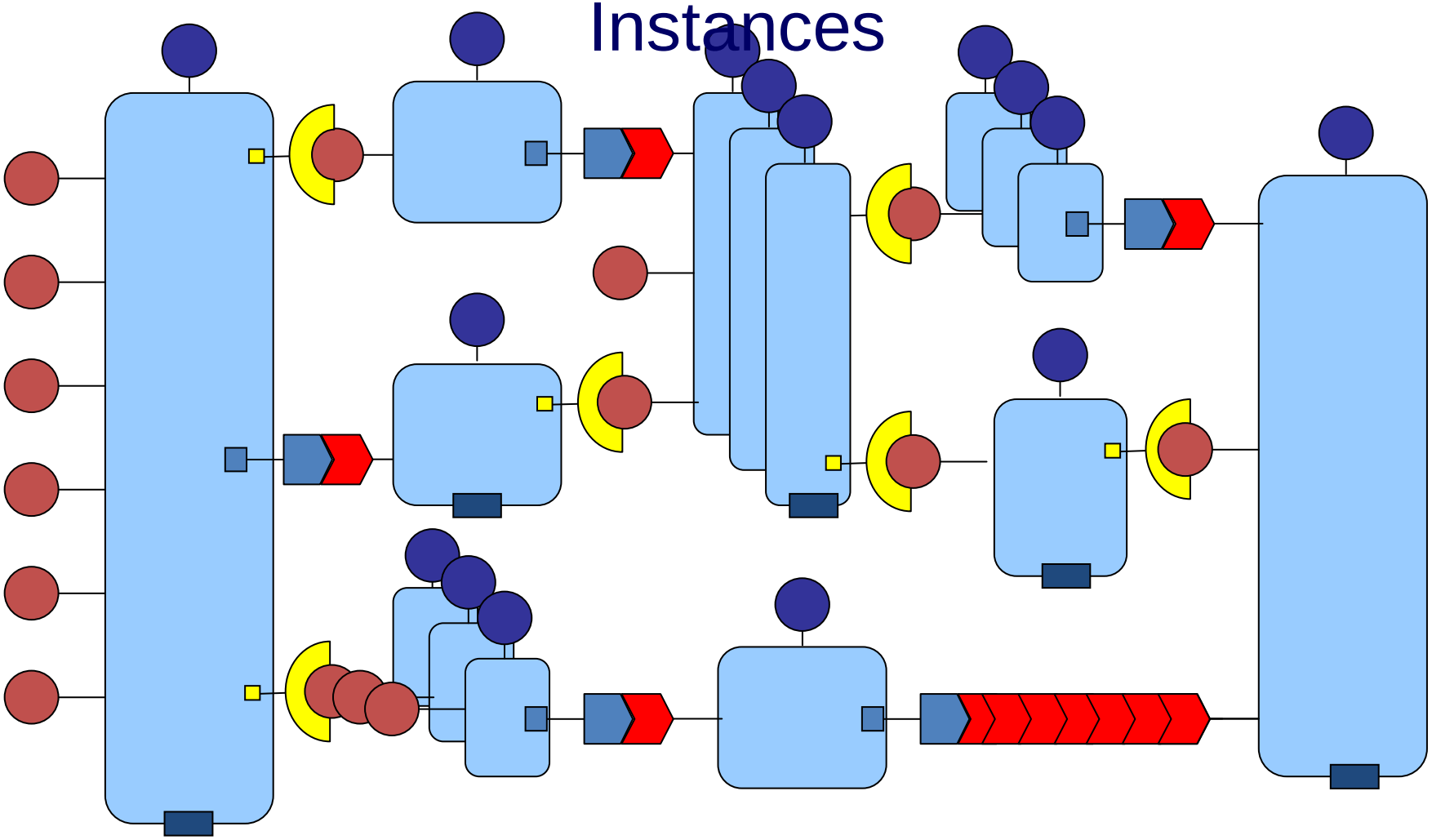
A deployment model (ongoing at OMG)

A CORBA Component



Courtesy of Philippe Merle, Lille, OpenCCM platform

Building CCM Applications = Assembling CORBA Component Instances



Provide + Use, but flat assembly

Towards GRID Components

Parallel and Distributed:

High-Performance a specificity ?
Not sure: an EJB component handling 1 000 000 of requests already needs High-Performance!

--> Group Communications

Business Cp plus specificity

High performance Communication :

Networks grow faster than Procs

Important Bandwidth

Techniques for hiding it

Very High Latency

Complexity: **--> Abstractions**

- Various remote execution tools (rsh, ssh, Globus, Web Services)
- Various registry and lookup protocols (LDAP, RMI, WS, etc.)
- Large variations in nodes being used (1 to 5000, … 200 000)

,

Across the world ??

3.2 ProActive Components

Component Orientedness

Level 1: Instantiate - Deploy - Configure

Simple Pattern

Meta-information (file, XML, etc.) — JavaBeans, EJB

Level 2: Assembly (flat)

Server and client interfaces — CCM

Level 3: Hierarchic

Composite — Fractal, ProActive, ...

Level 4: Distributed + Reconfiguration

Binding, Inclusion, Location — ProActive + On going work

ProActive

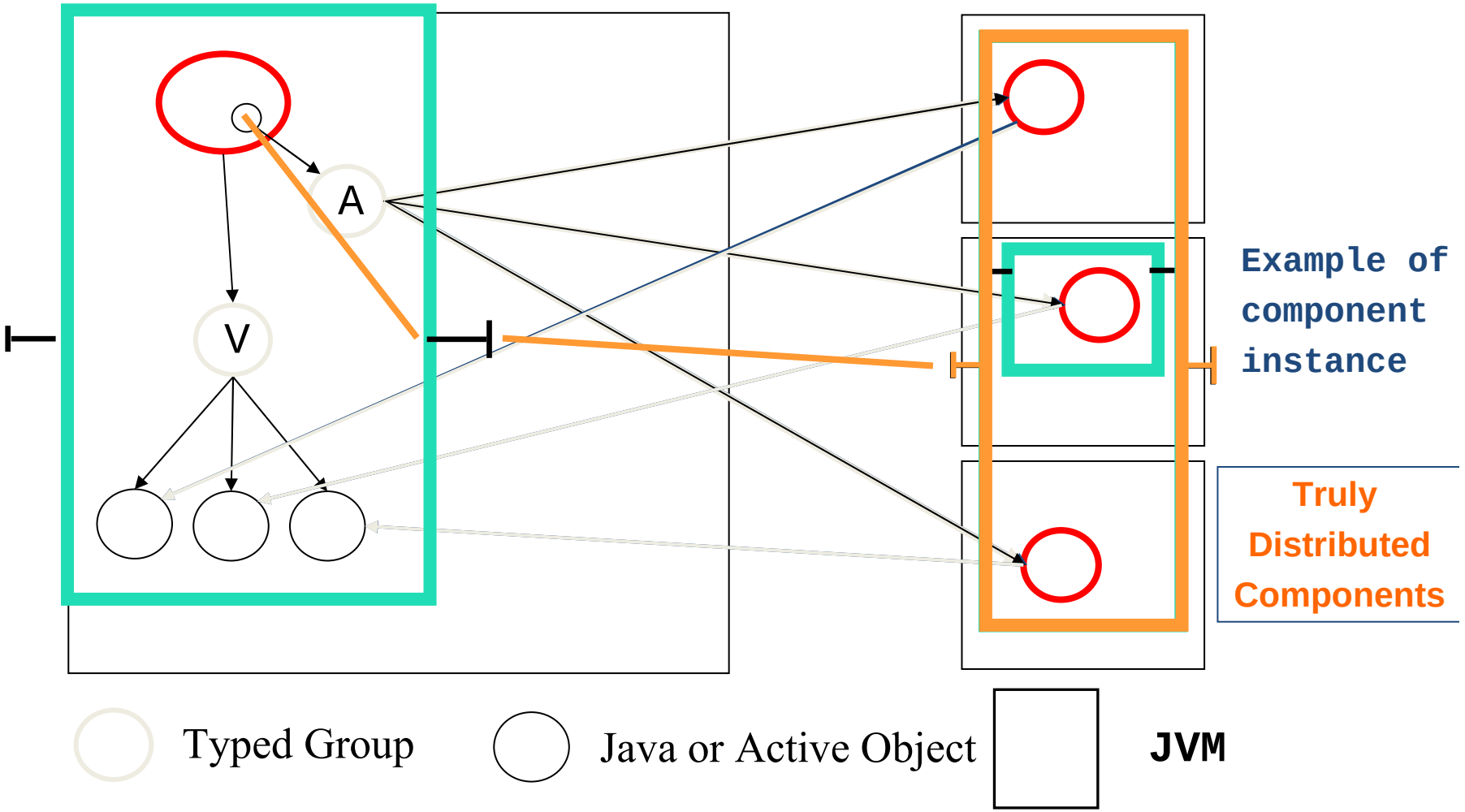
Interactions / Communications:

Functional Calls: service, event, stream

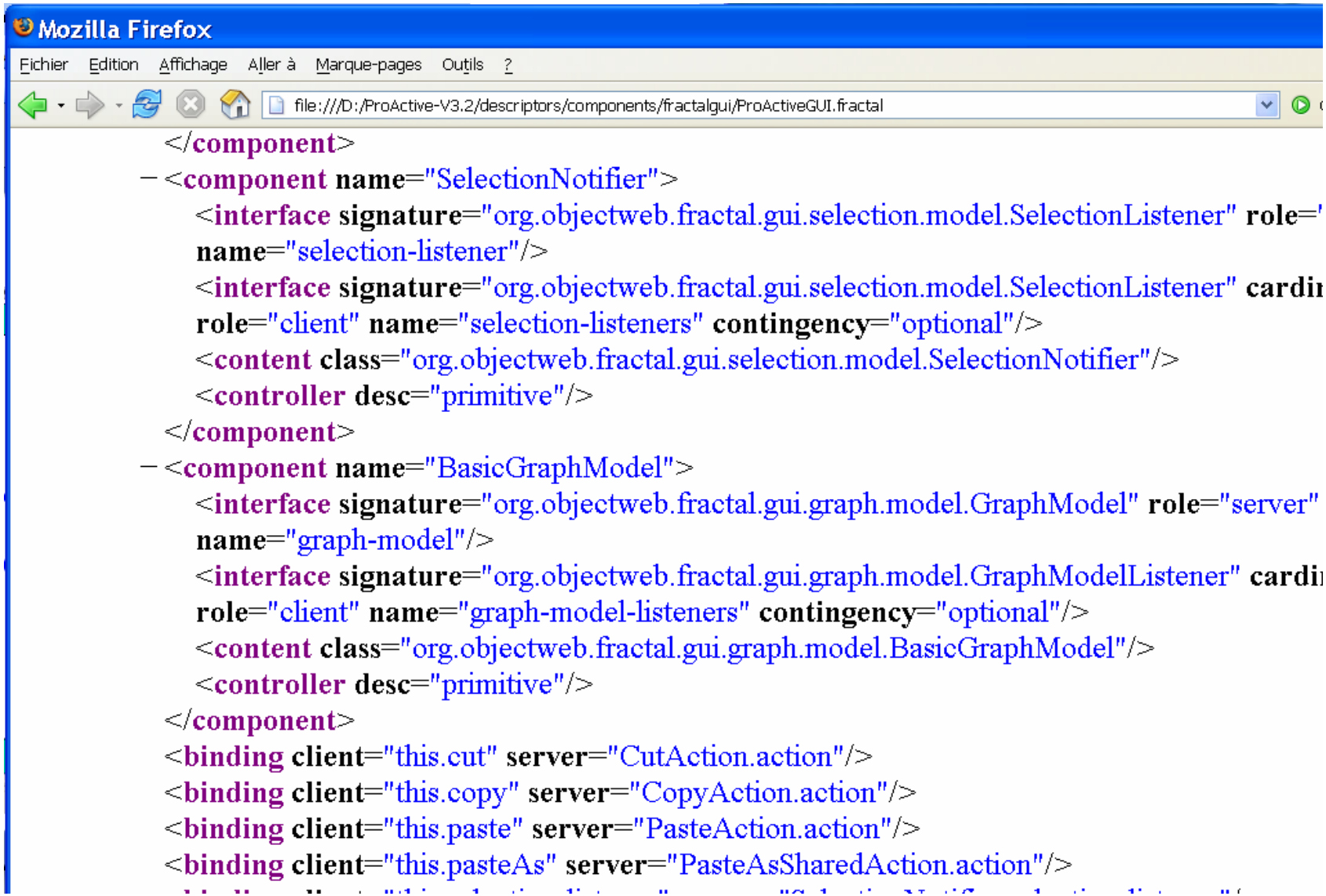
Non-Functional: instantiate, deploy, start/stop, inner/outer, re-bind

Objects to Distributed Components (1)

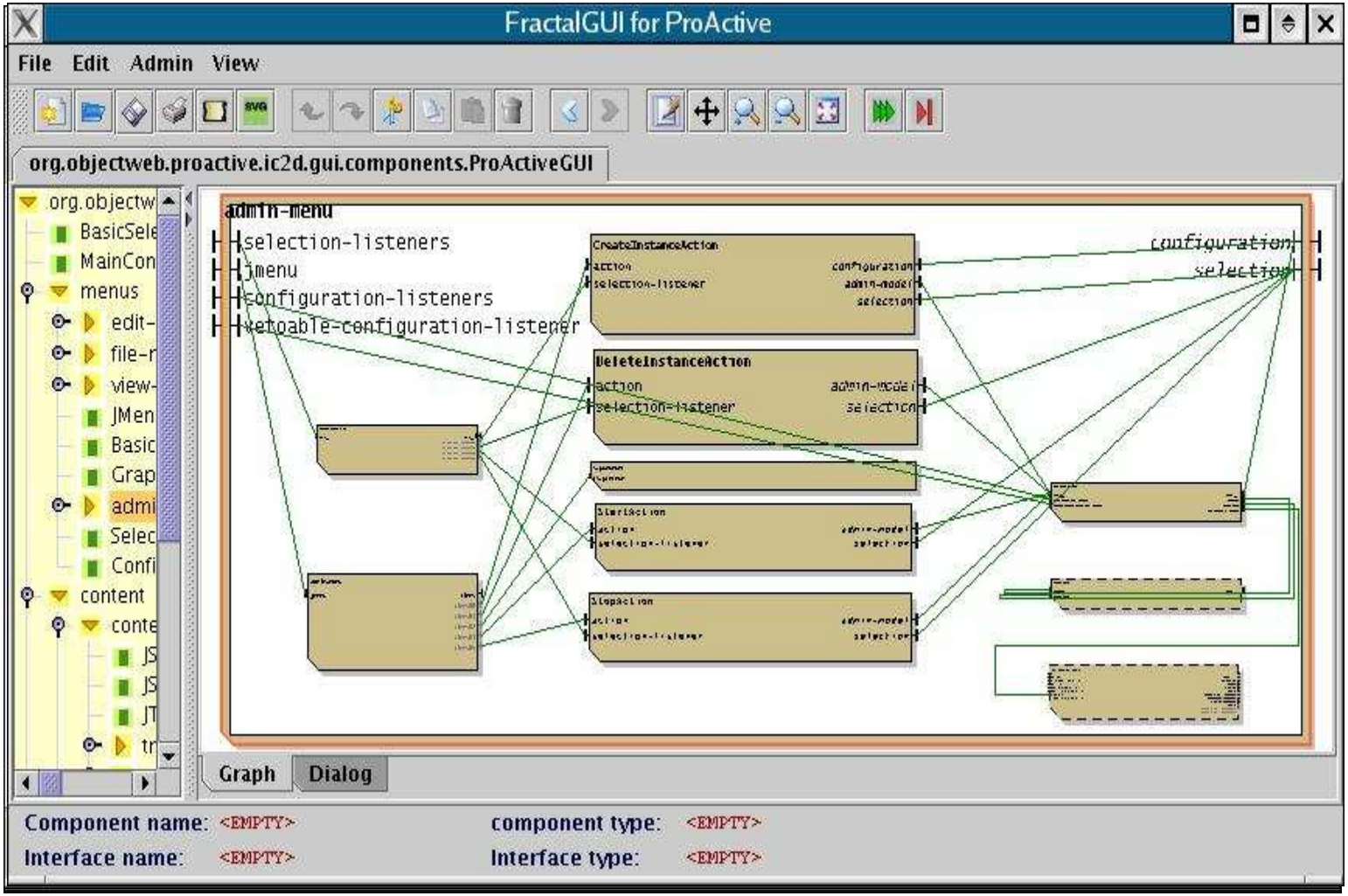
```
ComponentIdentity Cpt = newActiveComponent (params);
A a = Cpt … .getFcInterface ("interfaceName");
V v = a.foo(param);
```



Binding with XML ADL

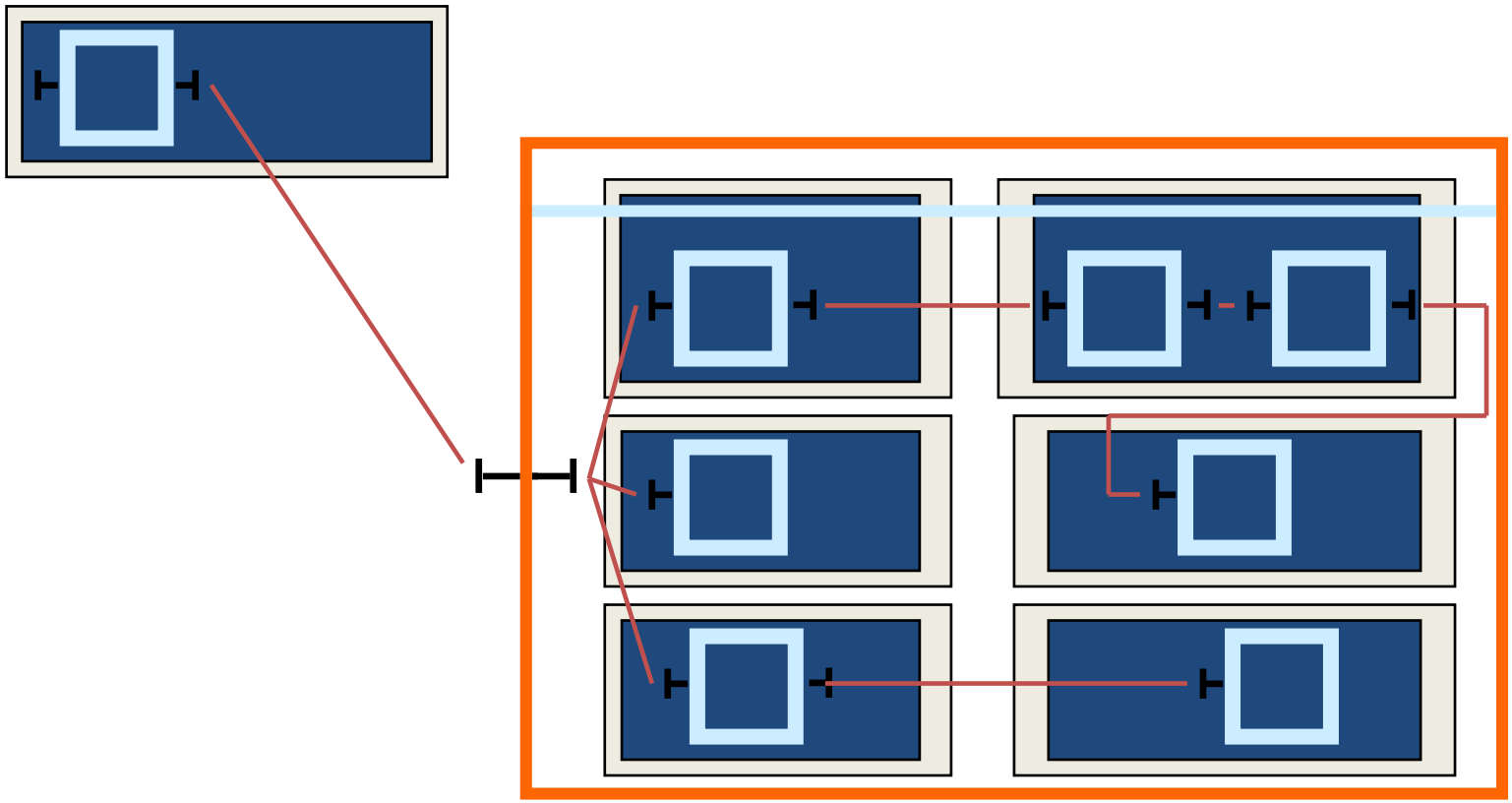


On-going work : GUI for Components

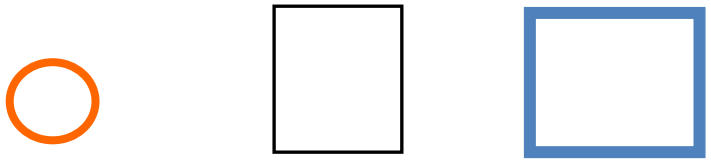


Distributed Components (2)

1 component can be distributed over several hosts



Components vs. Activity and JVMs

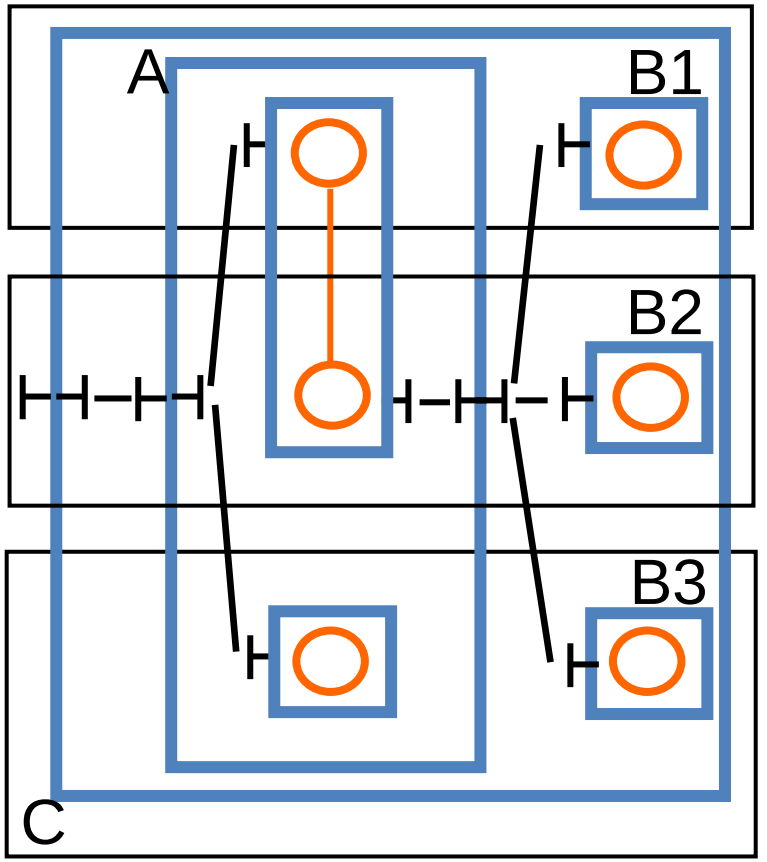


Cp. are rather orthogonal to activities and JVMs:

contain activities, span across several JVMs

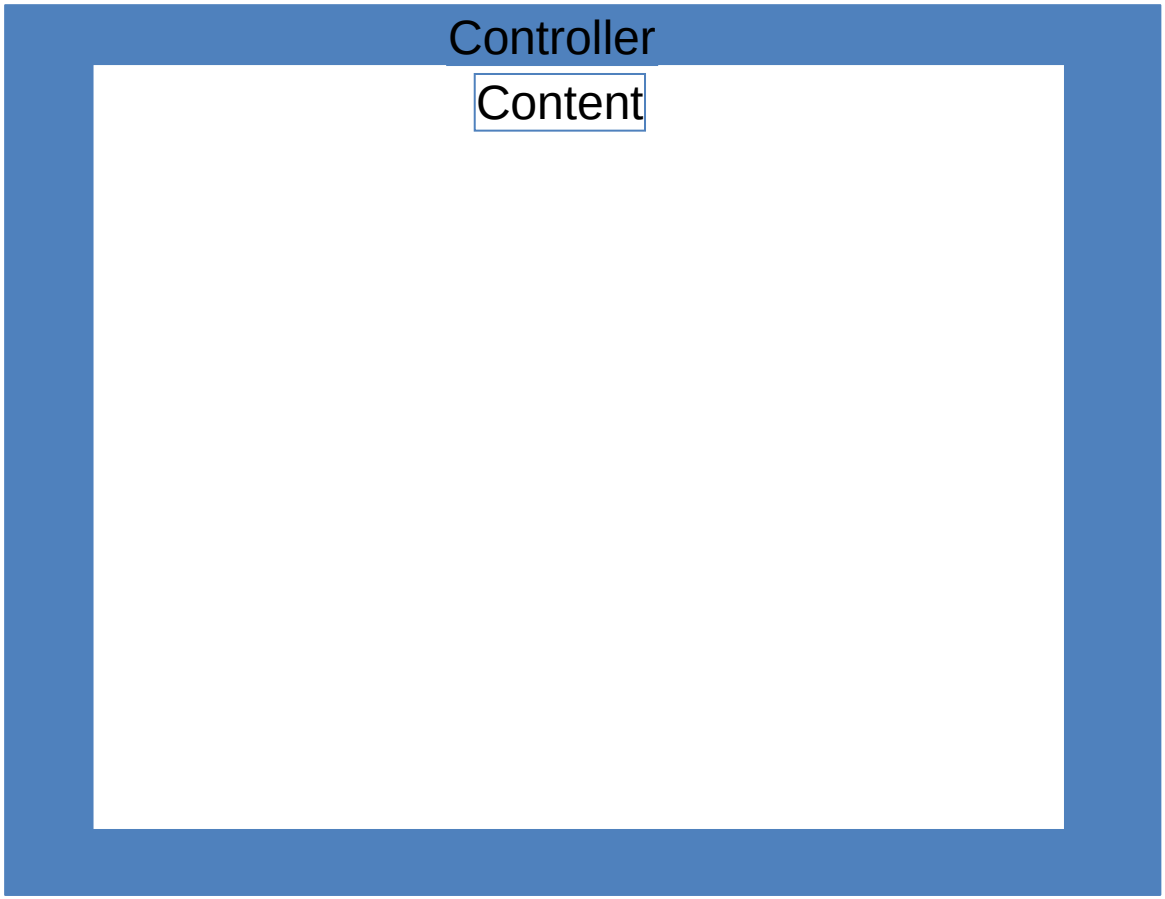
Components are a way to globally manipulate

distributed, and running activities



Using The Fractal model:
Hierarchical Component

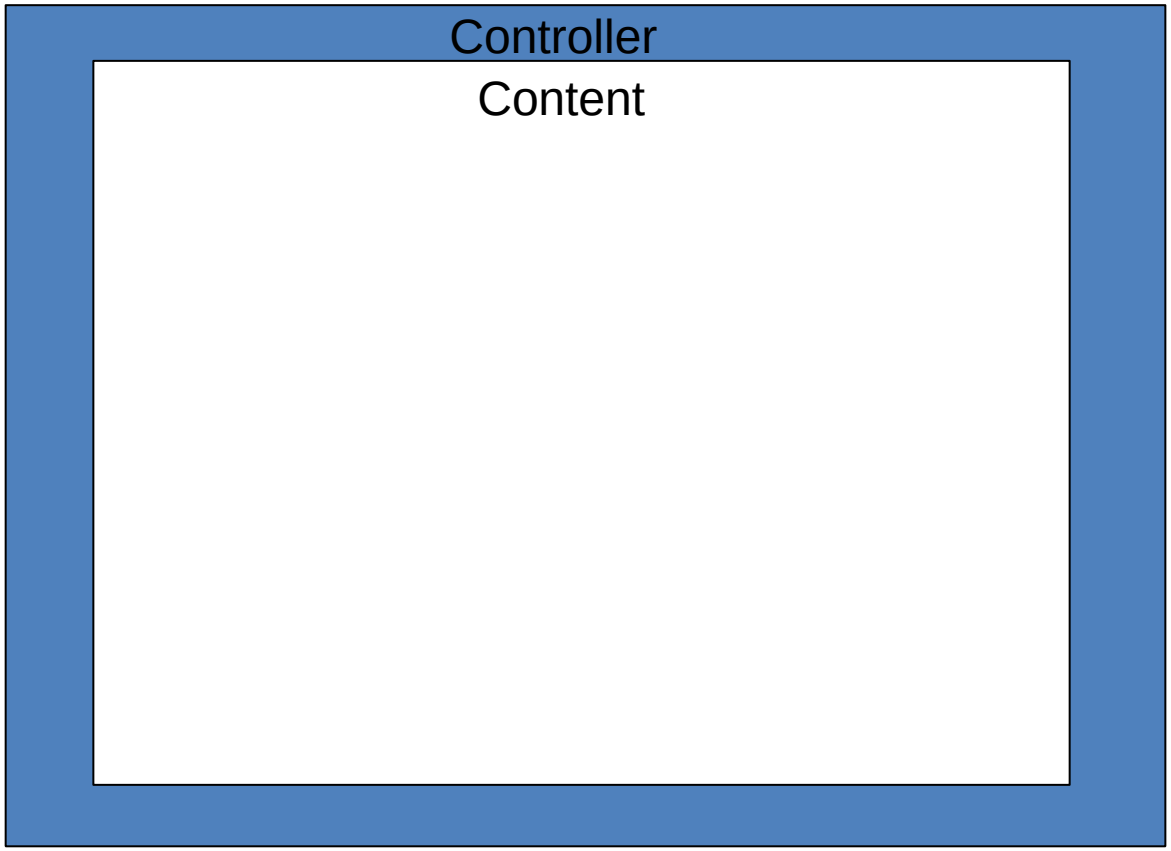
Orange & INRIA (E. Bruneton, T. Coupaye, J.B. Stefani)



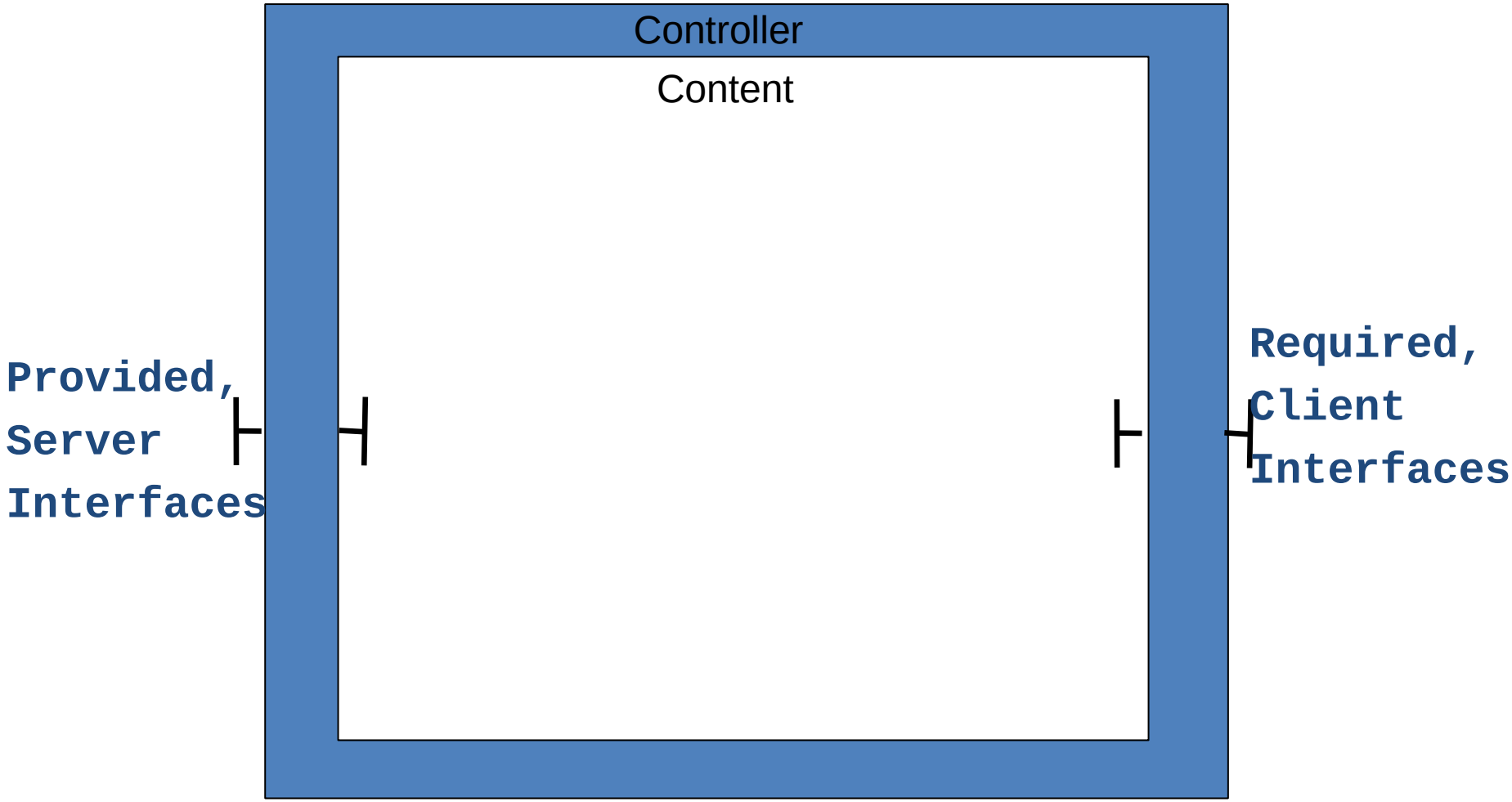
The Fractal model:

Hierarchical Components

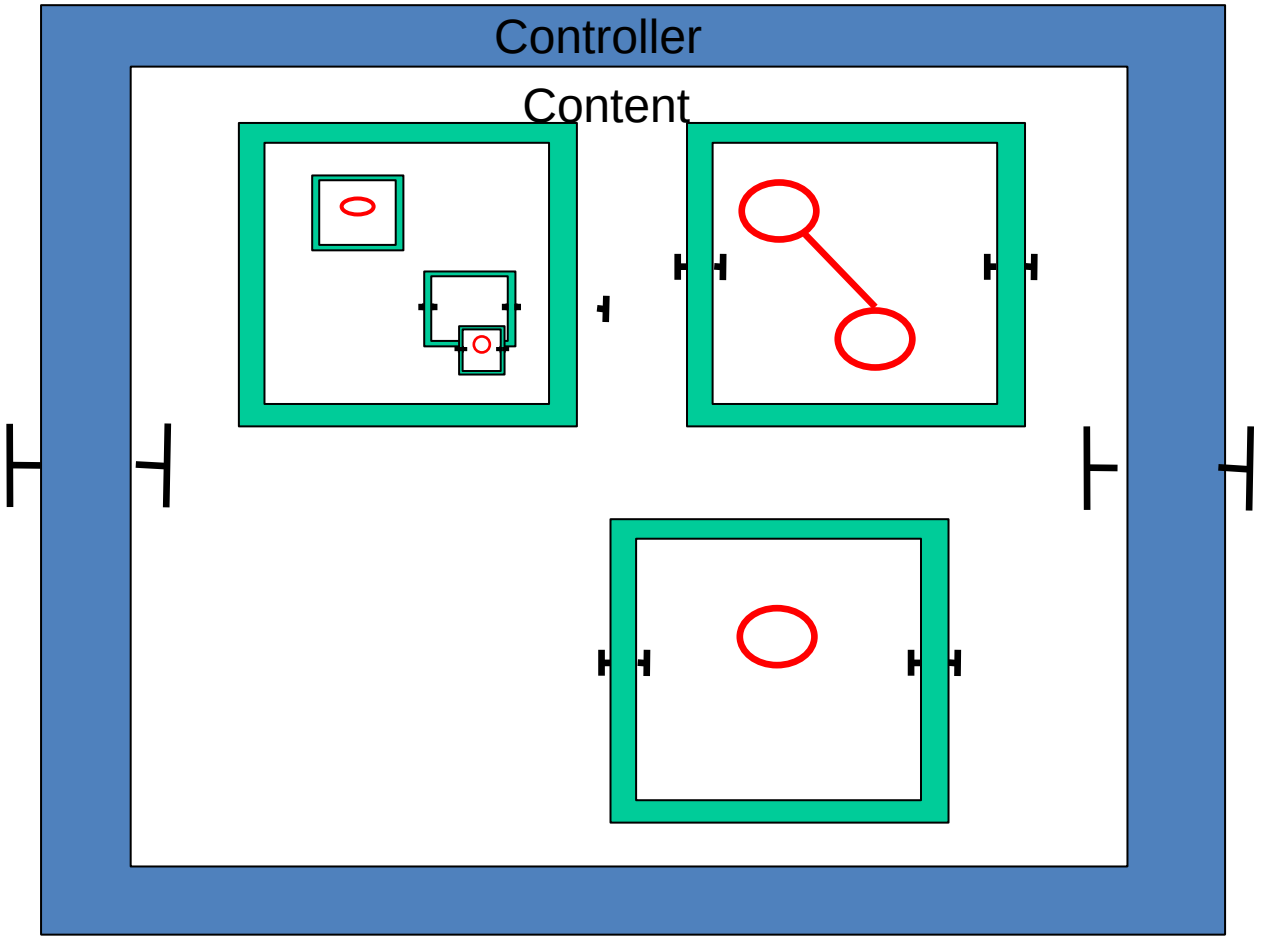
Common component model of the ObjectWeb consortium



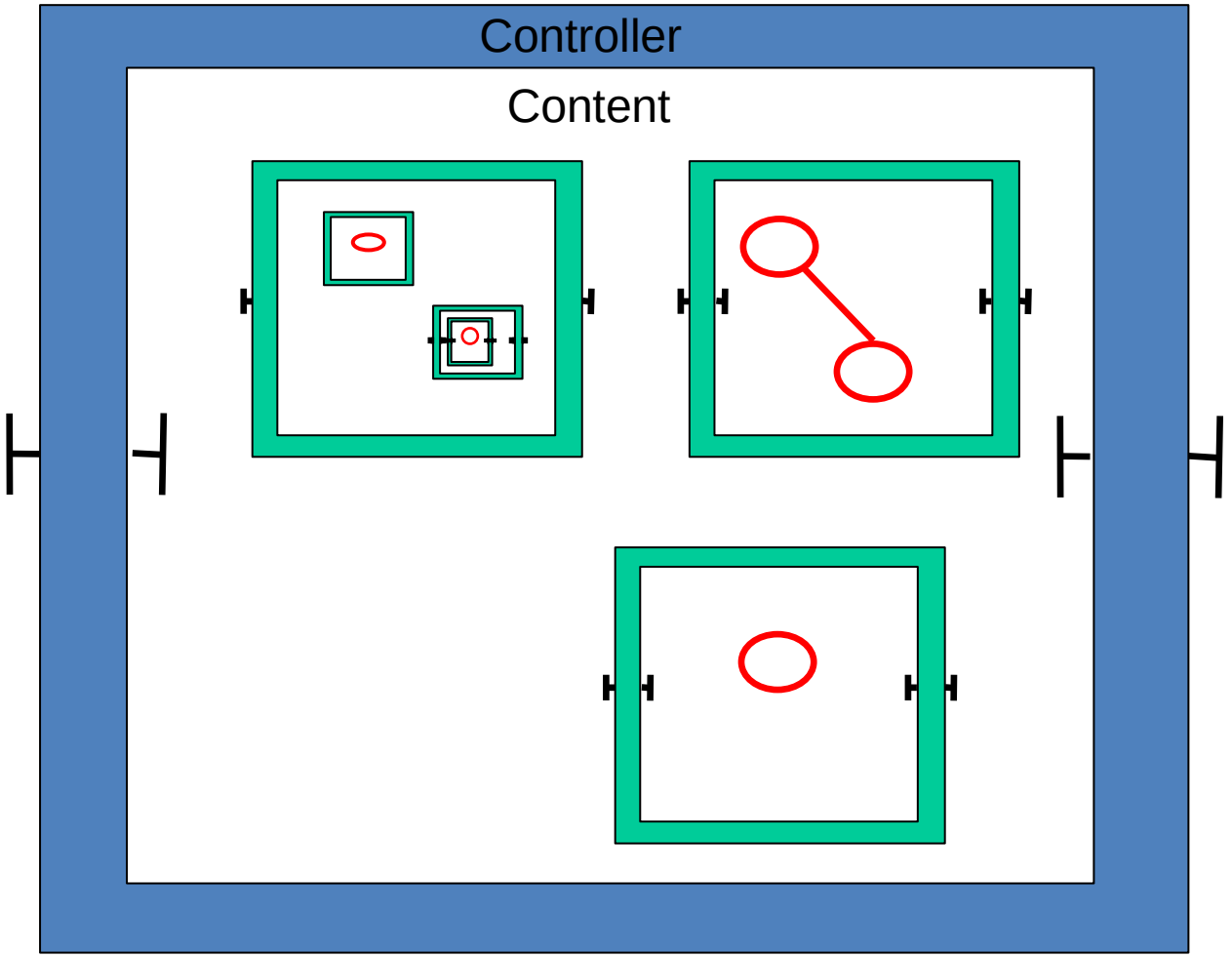
Interfaces = Provided and Required



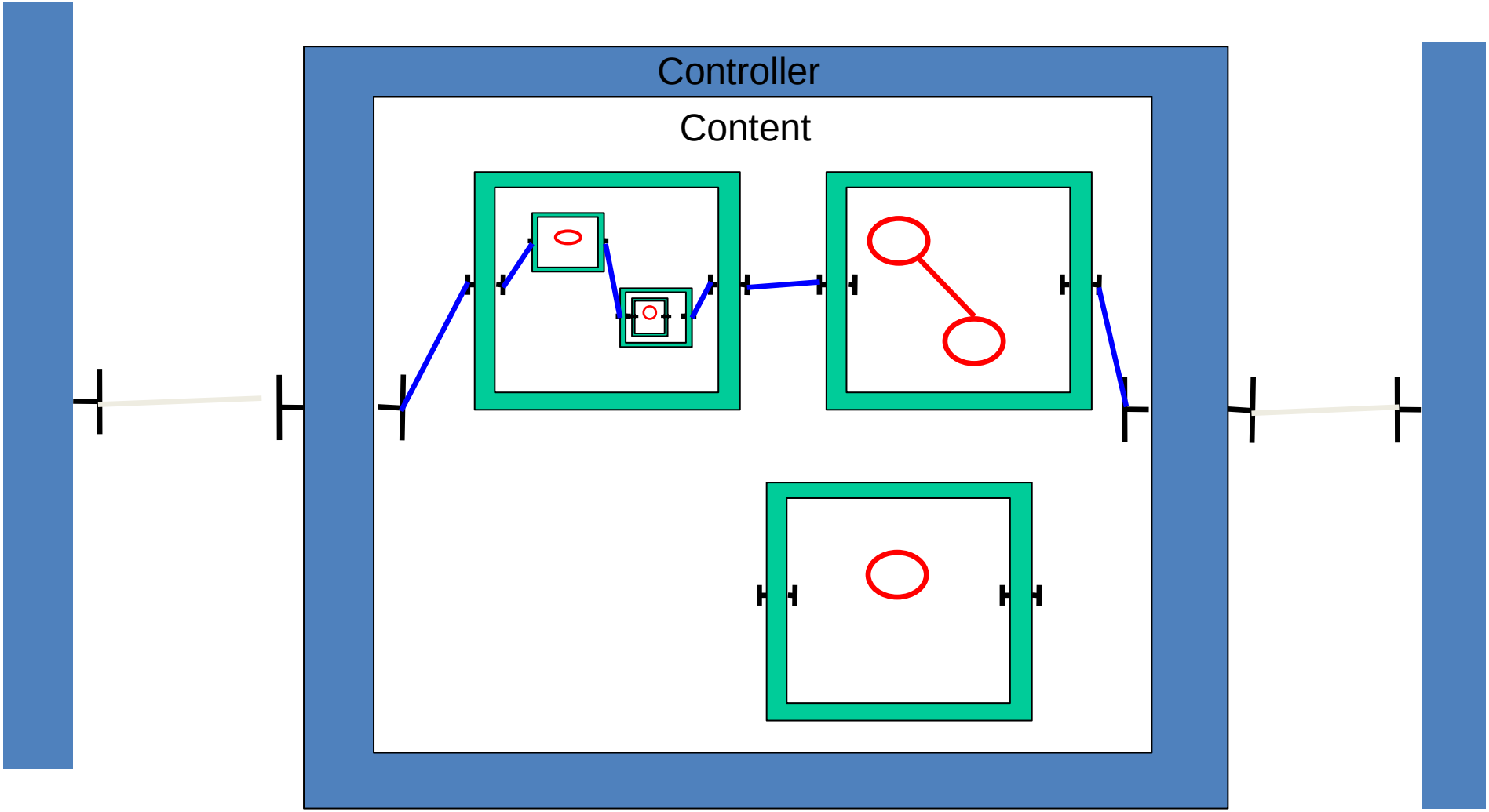
Hierarchical model : Composites encapsulate Primitives, Primitives encapsulate Code



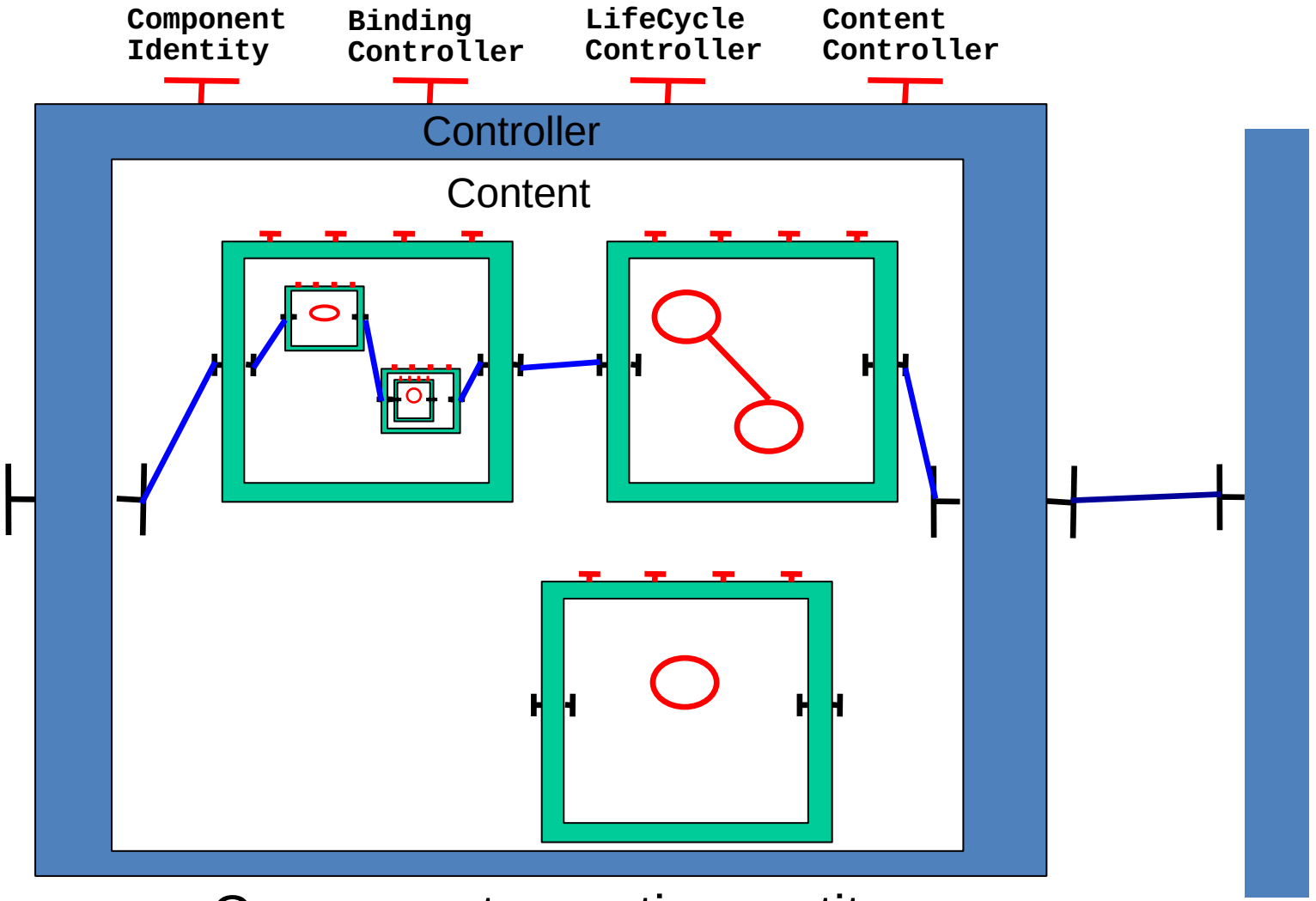
Binding = in an external file (XML ADL), Not in programs



Binding = in an external file (XML ADL), Not in programs



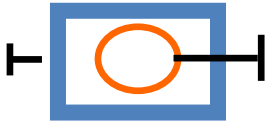
Controllers : non-functional properties



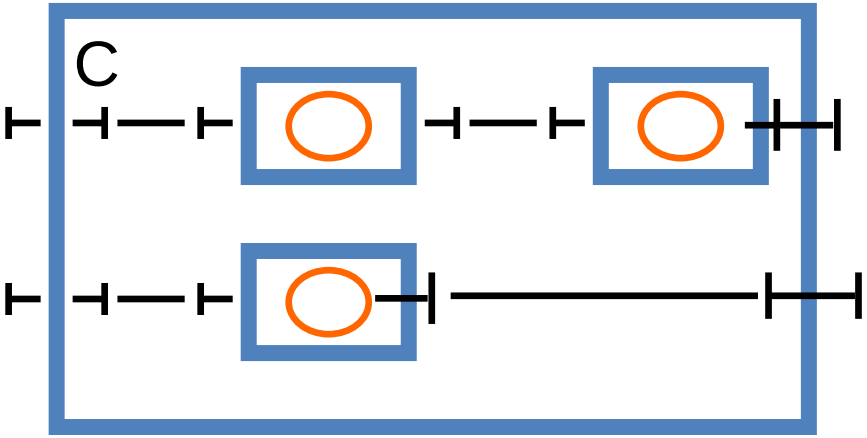
Component = runtime entity

ProActive Components for the GRID

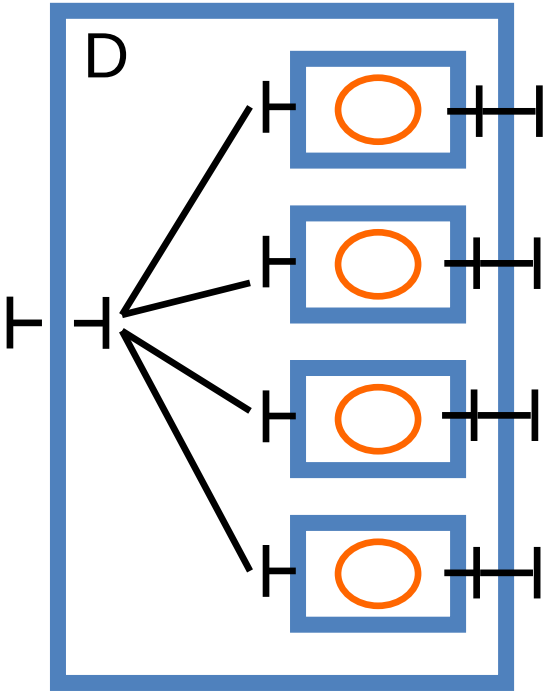
An activity, a process, …
potentially in its own JVM



1. Primitive component



2. Composite component



3. Parallel and composite component

Composite: Hierarchical, and Distributed over machines

Parallel: Composite + Broadcast (group)

ProActive Component Definition

A component is:

- Formed from one (or several) Active Object
- Executing on one (or several) JVM
- Provides a set of server ports: **Java Interfaces**
- Uses a set of client ports: **Java Attributes**
- Point-to-point or Group communication between components

Hierarchical:

- **Primitive component**: define with Java code and a descriptor
- **Composite component:** composition of primitive + composite
- **Parallel component:** multicast of calls in composites

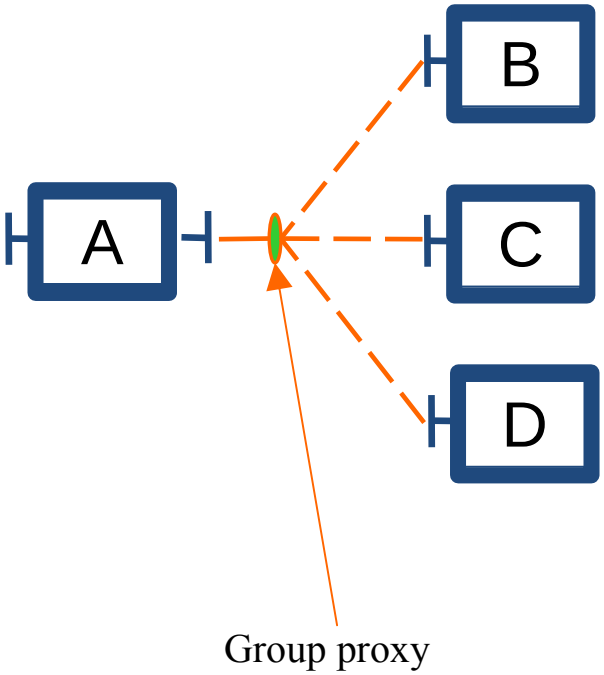
Descriptor:

- XML definition of primitive and composite (ADL)
- Virtual nodes capture the deployment capacities and needs

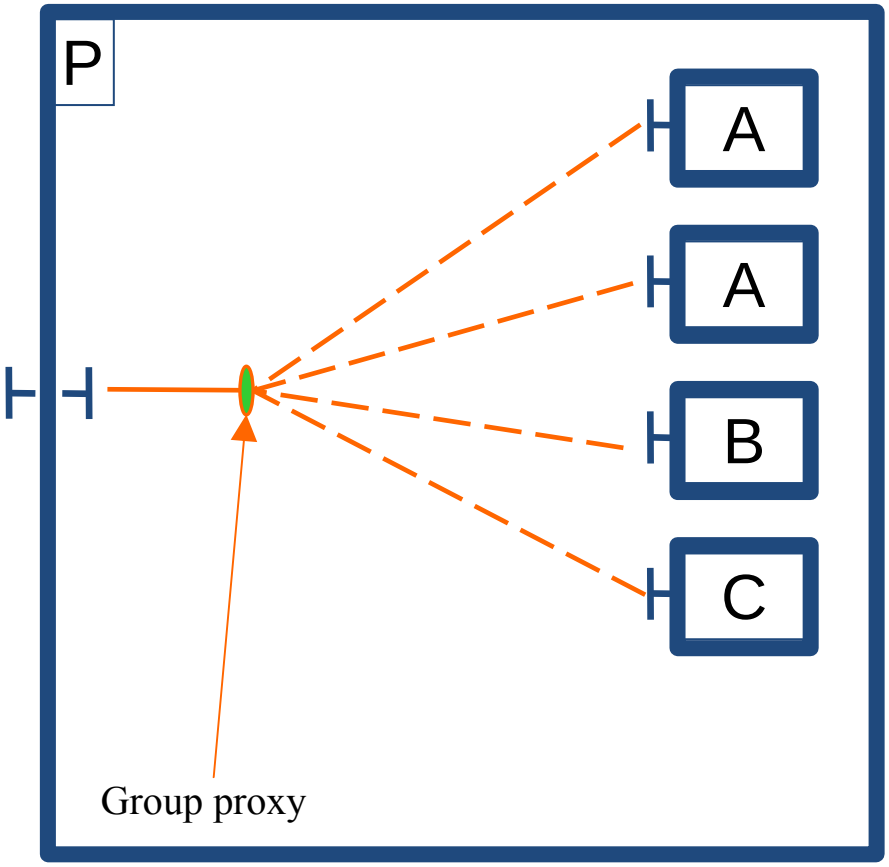
Virtual Node: a very important abstraction for GRID components

Groups in Components (1)

A parallel component!



Broadcast at binding,
on client interface

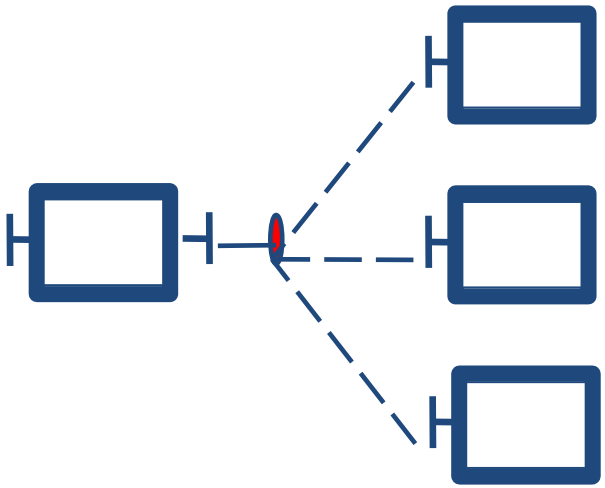


At composition,
on composite inner server interface

Groups in Components (2)

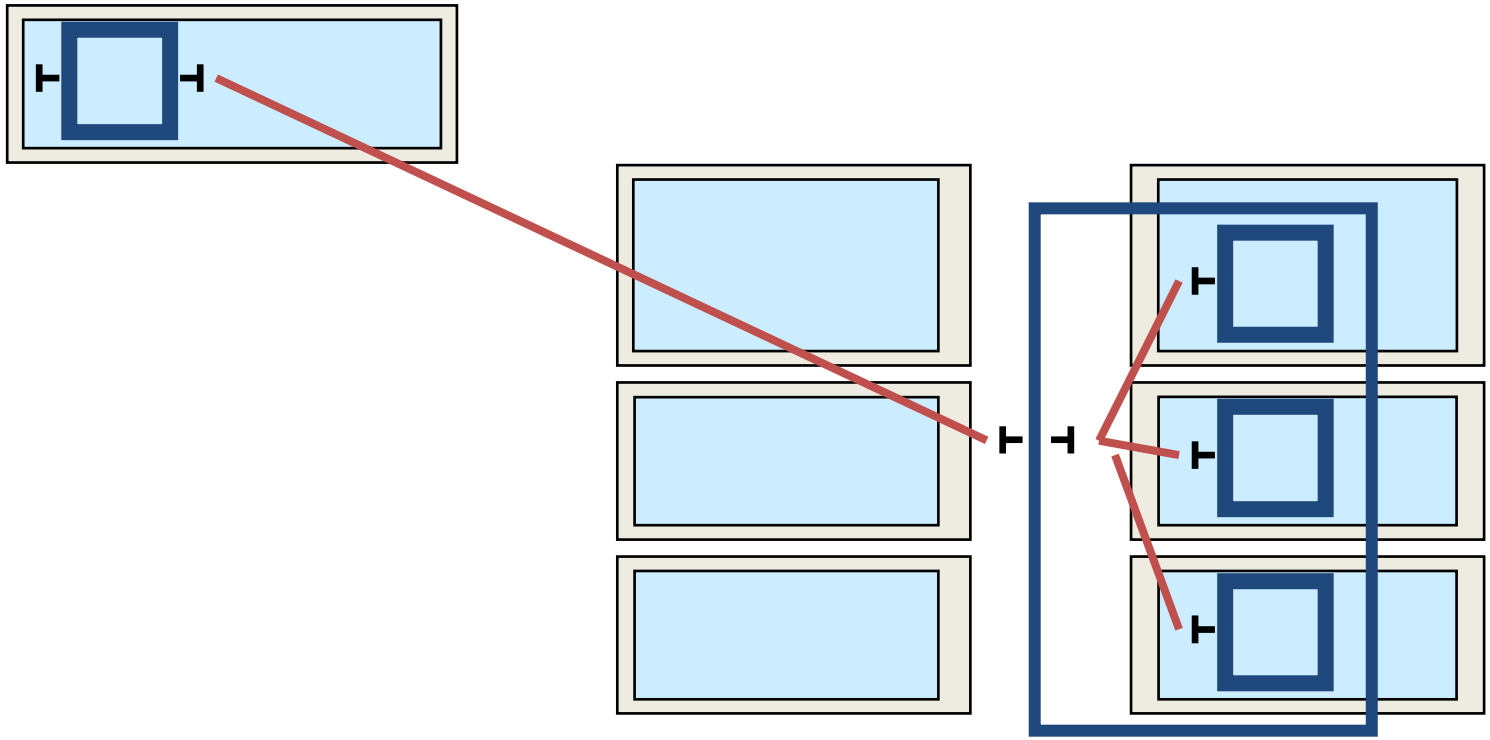
Typed group communications as the implementation of collective interfaces

2 modes : broadcast or scatter



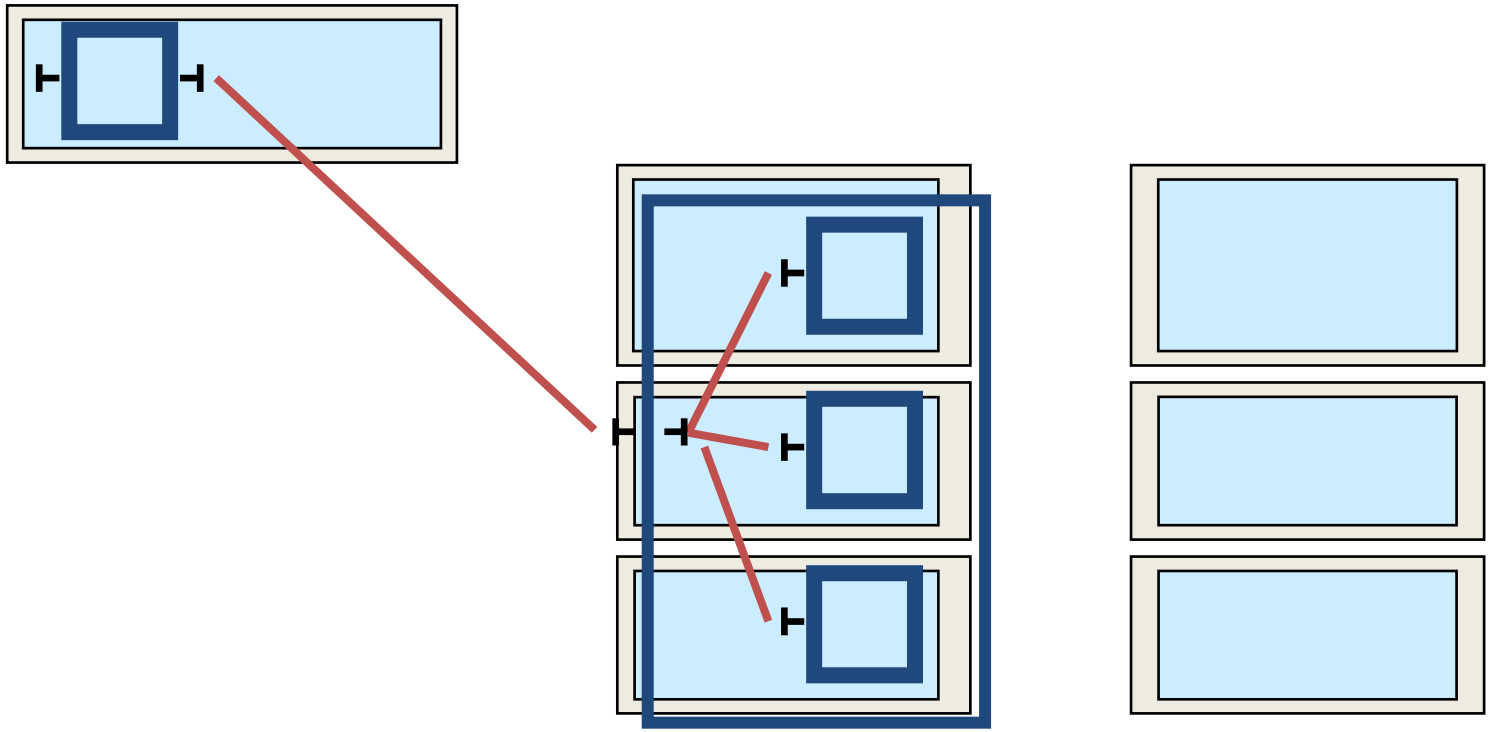
Migration Capability of composites

Migrate sets of components, including composites



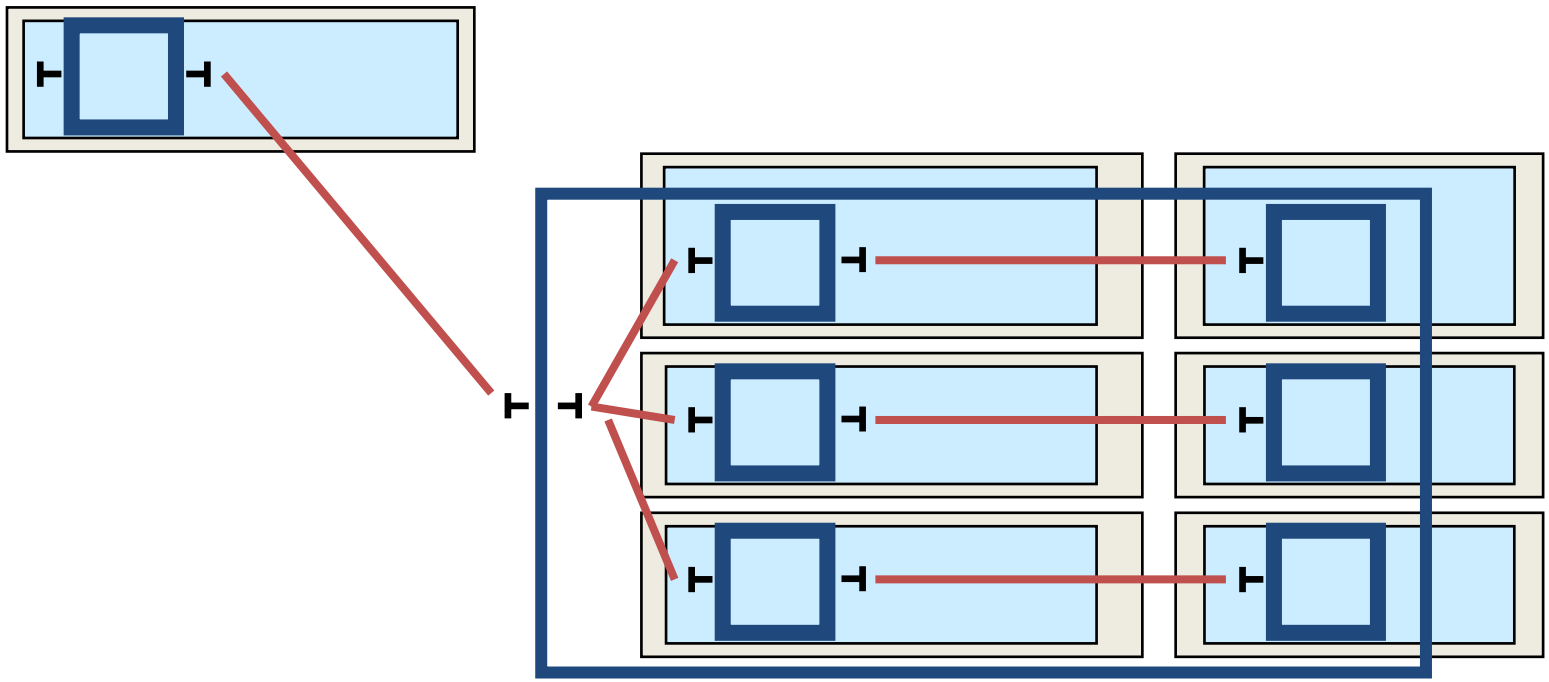
Migration Capability of composites

Migrate sets of components, including composites



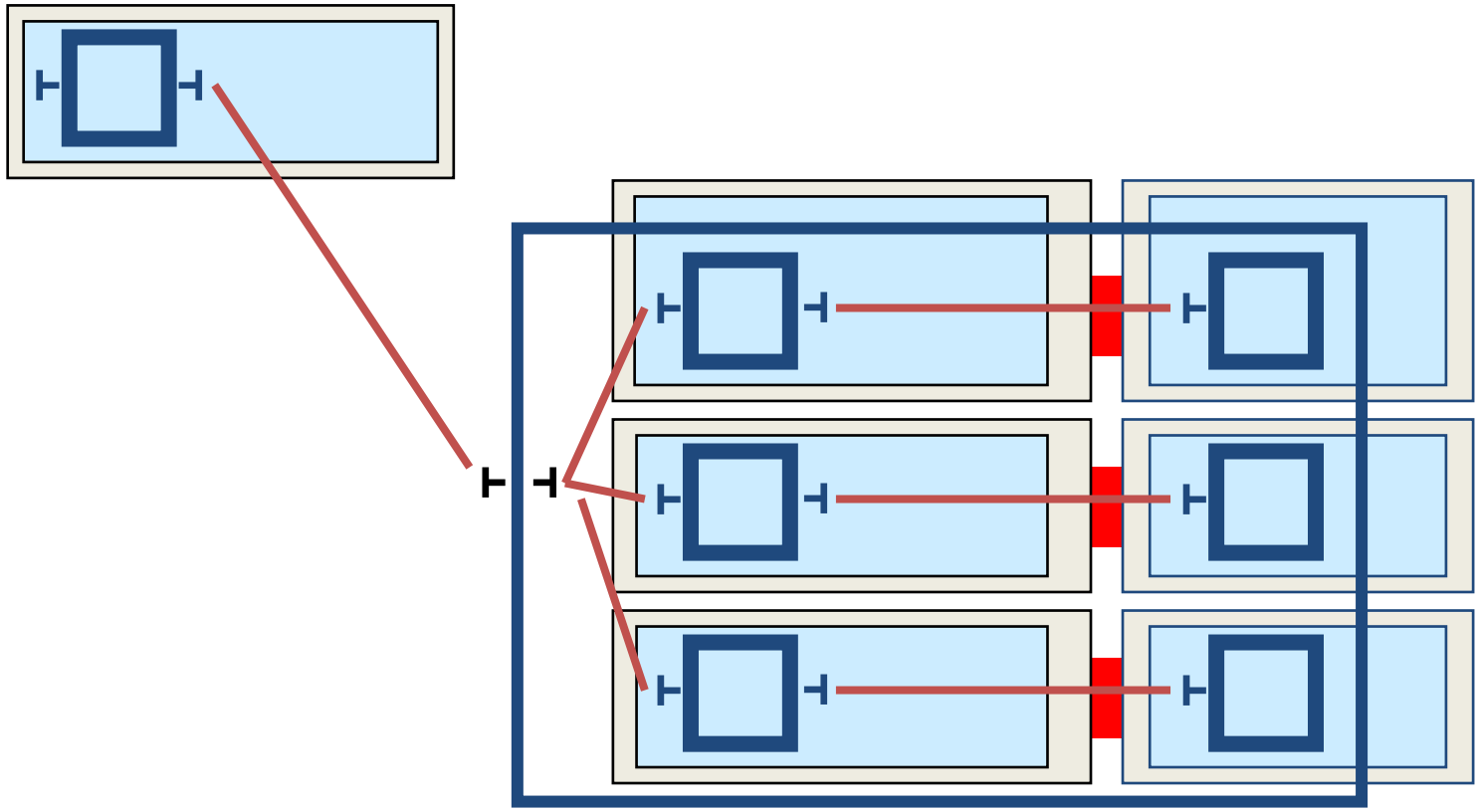
Co-allocation, Re-distribution

e.g. upon communication intensive phase



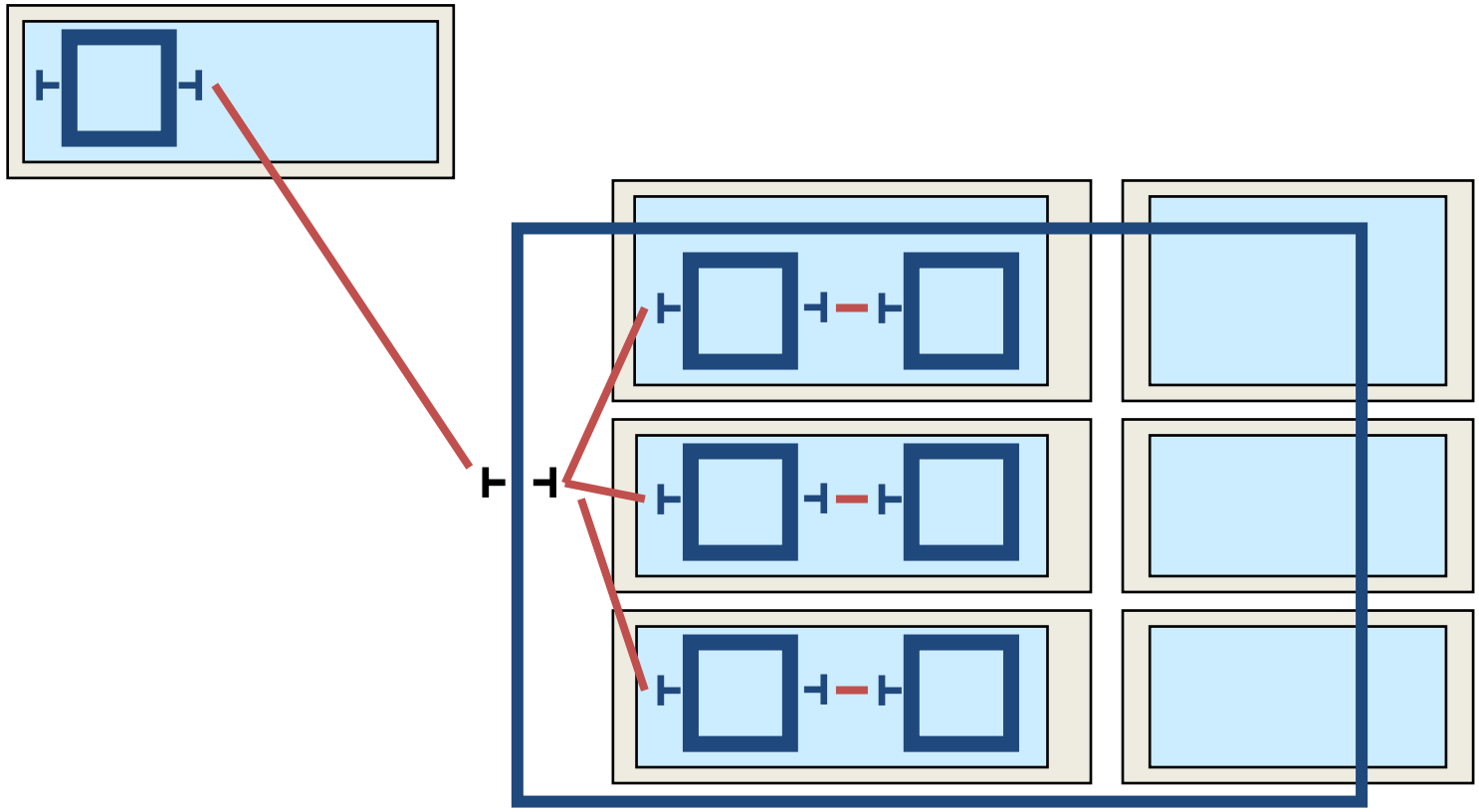
Co-allocation, Re-distribution

e.g. upon communication intensive phase



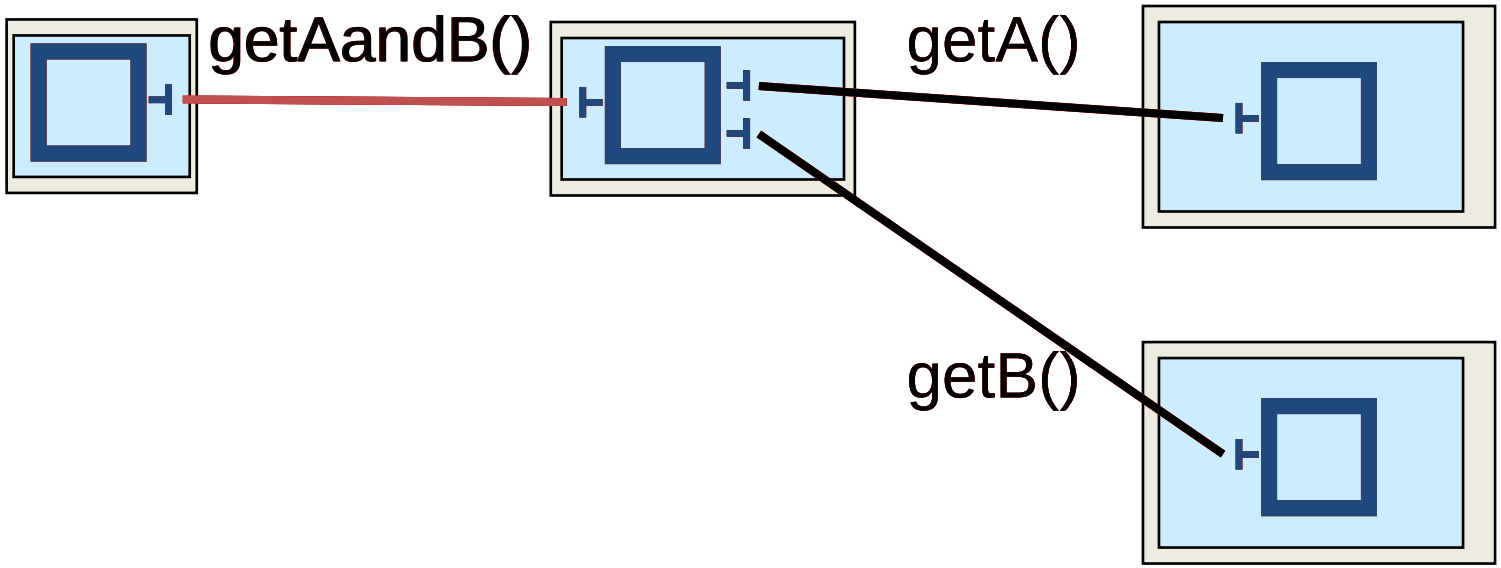
Co-allocation, Re-distribution

e.g. upon communication intensive phase



Functionalities : Without First Class Futures

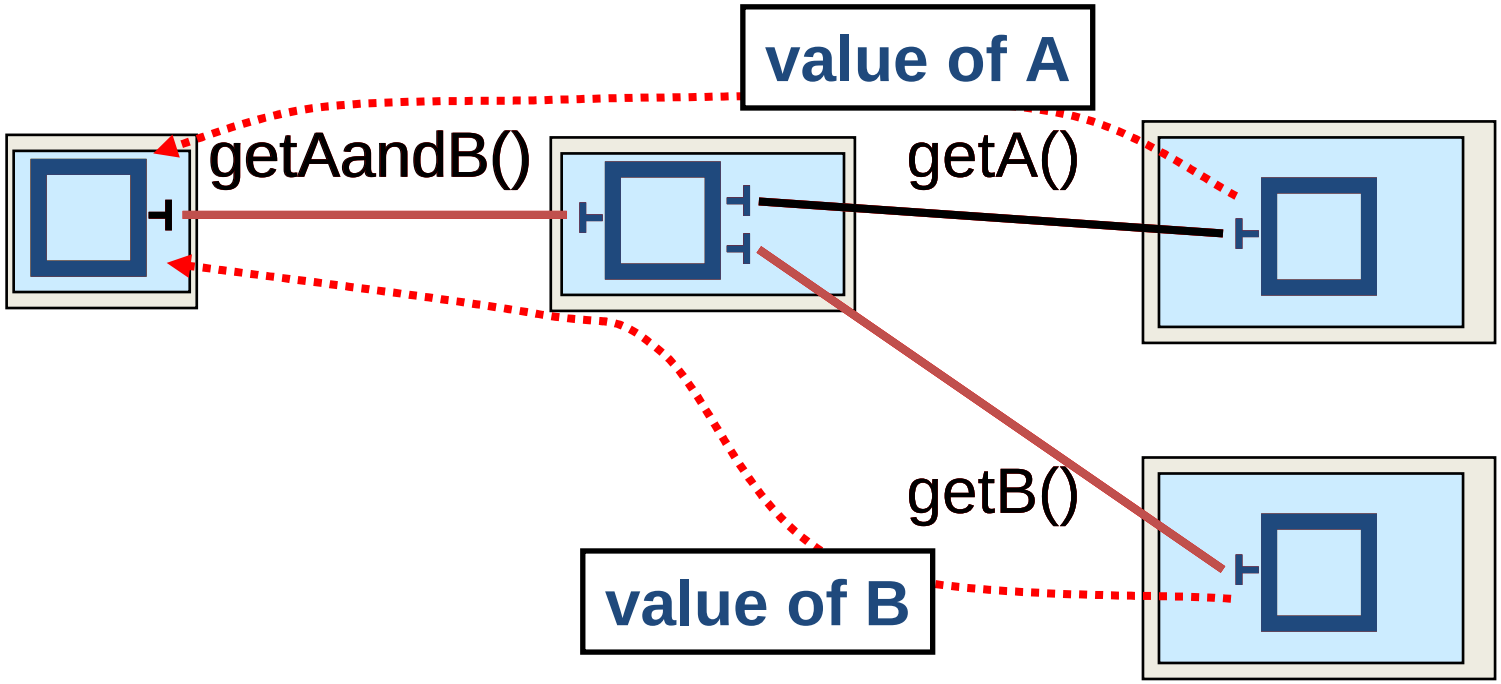
Or in the case of **Synchronous** method calls



Functionalities : With First Class Futures

Non-blocking method calls

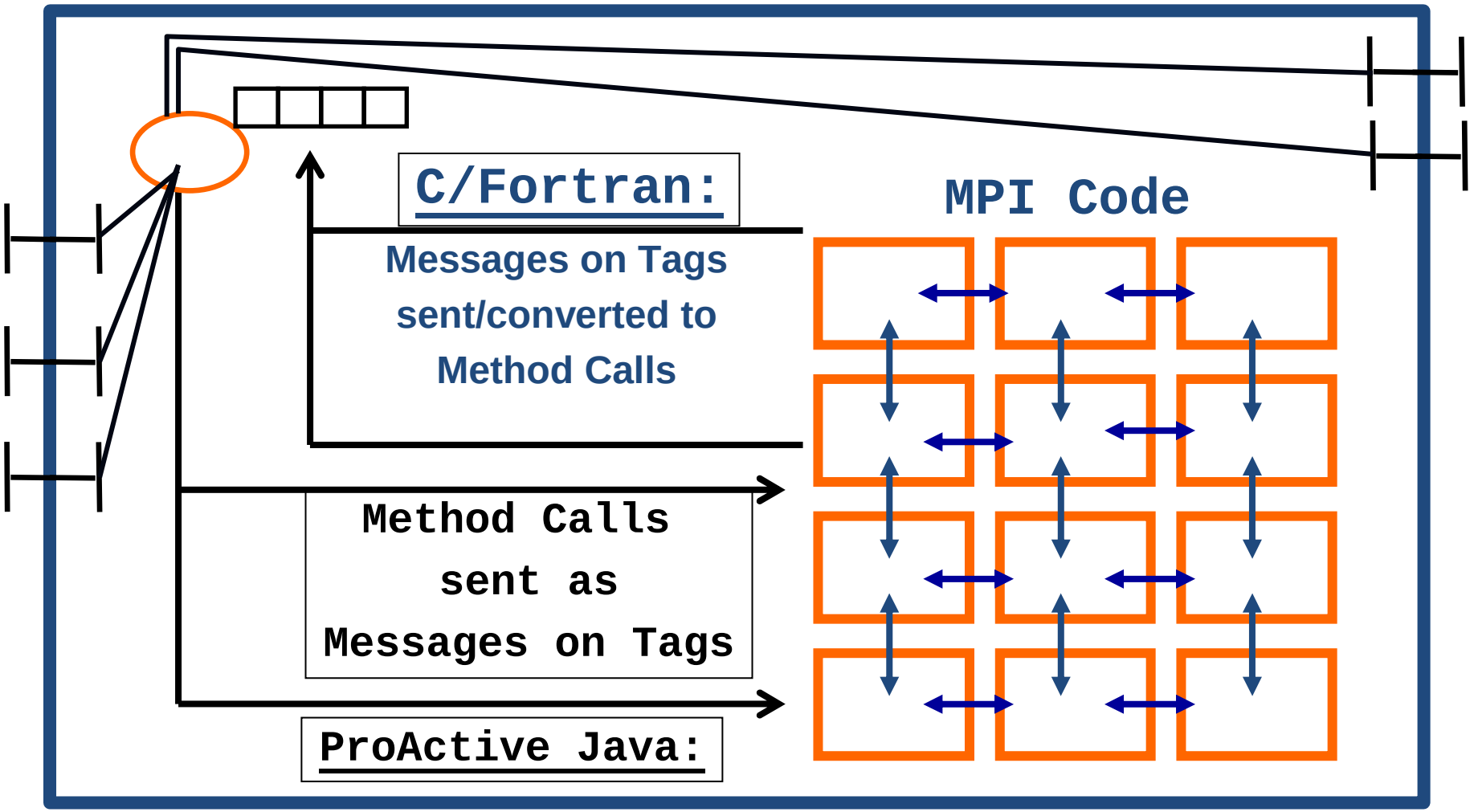
Example 2 : **Asynchronous** method calls with full-fledge **Wait-By-Necessity**



Assemblage are not blocked with Asynchrony + WbN

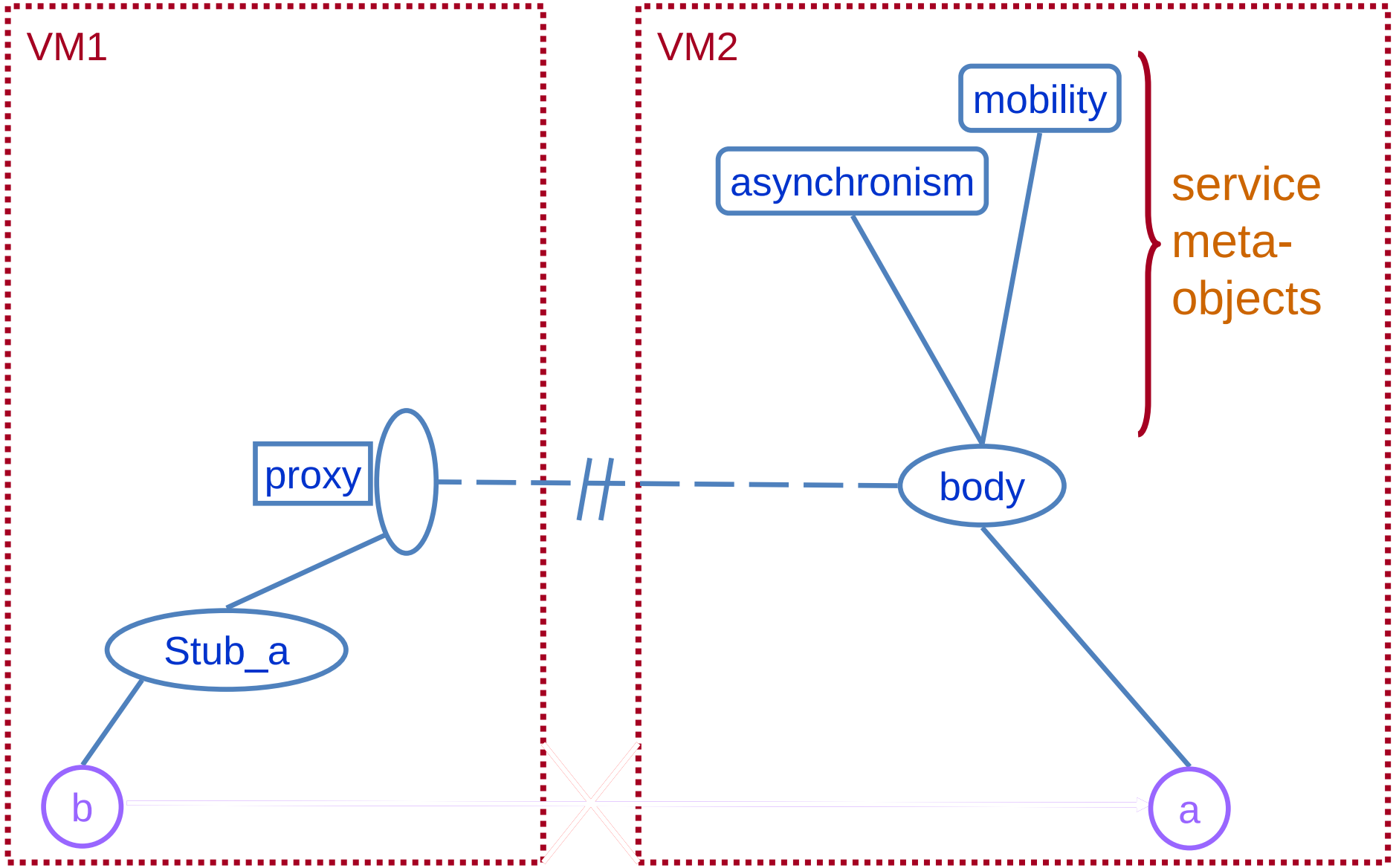
Wrapping Legacy MPI Components

Virtual Nodes for Deployments

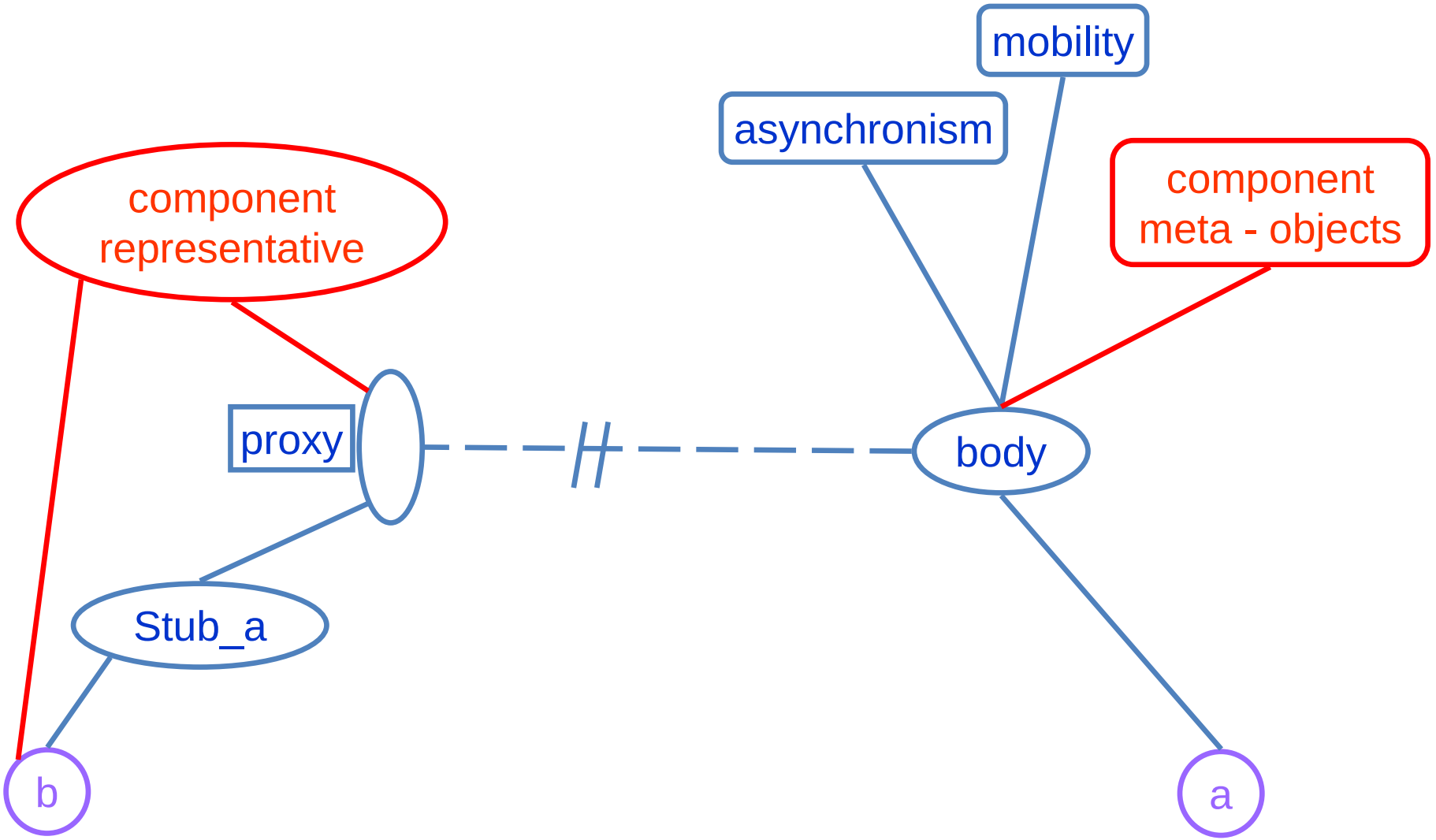


3.3 Component Implementation

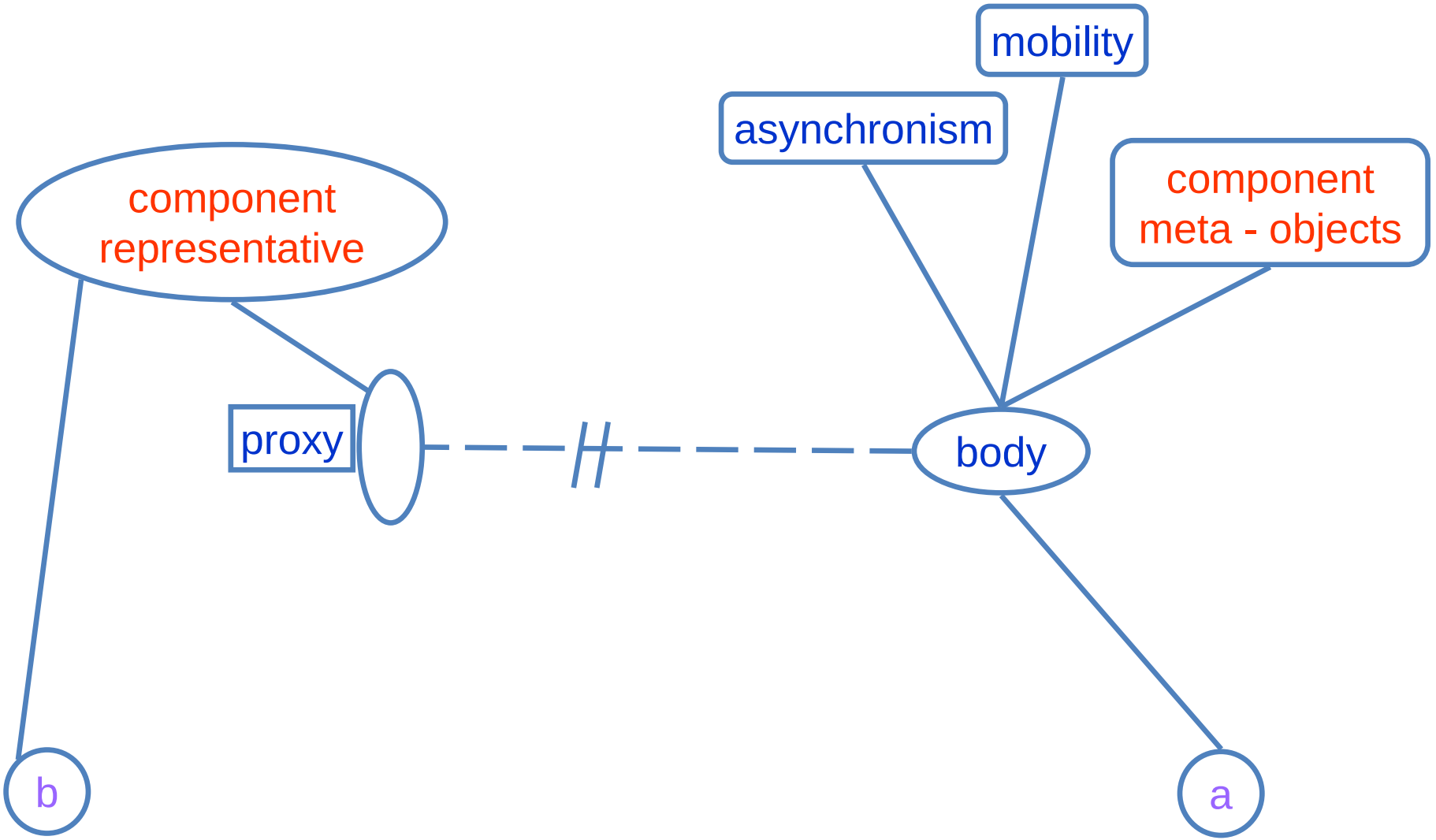
Architecture : based on the MOP



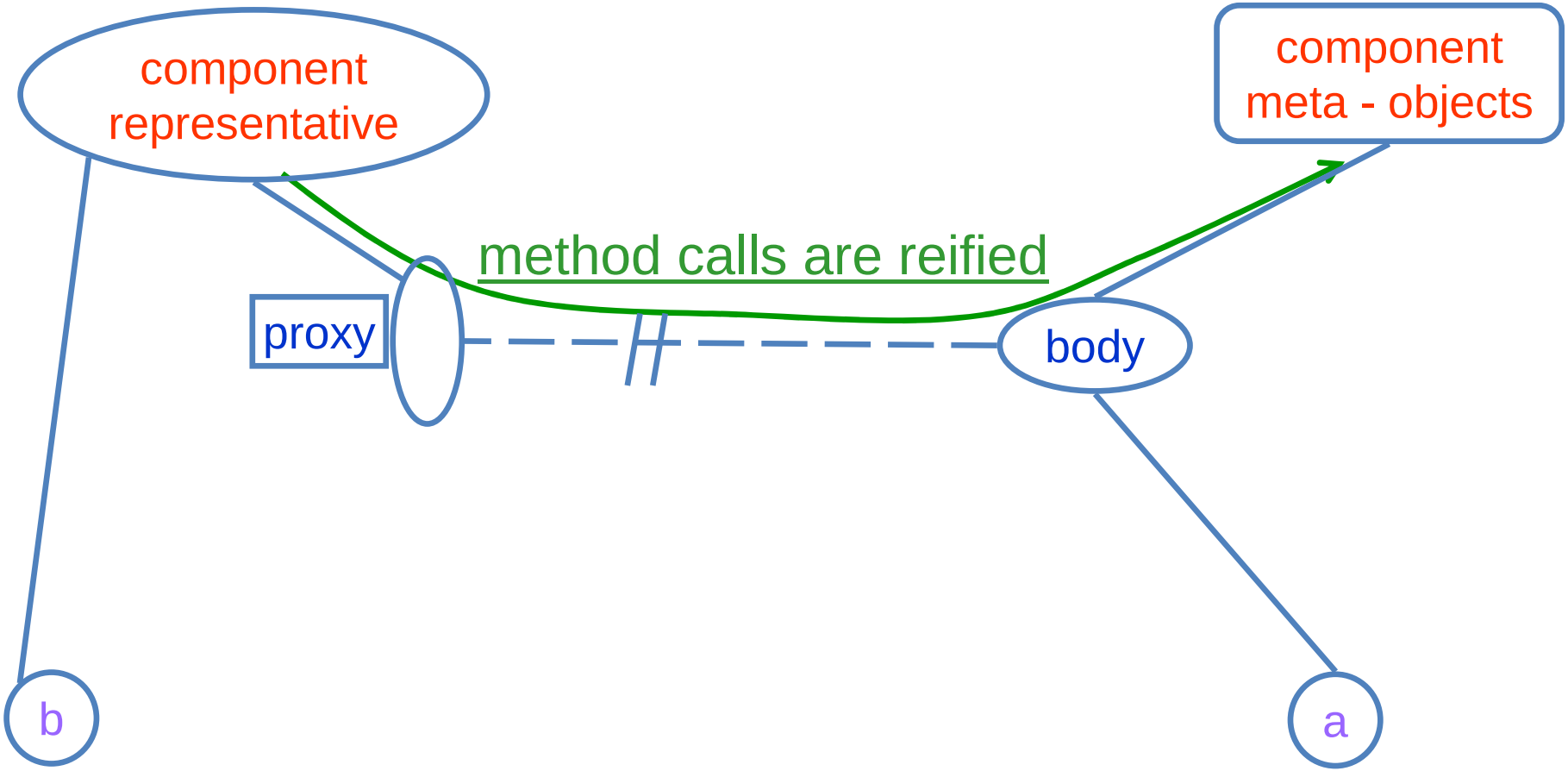
Architecture : component stubs



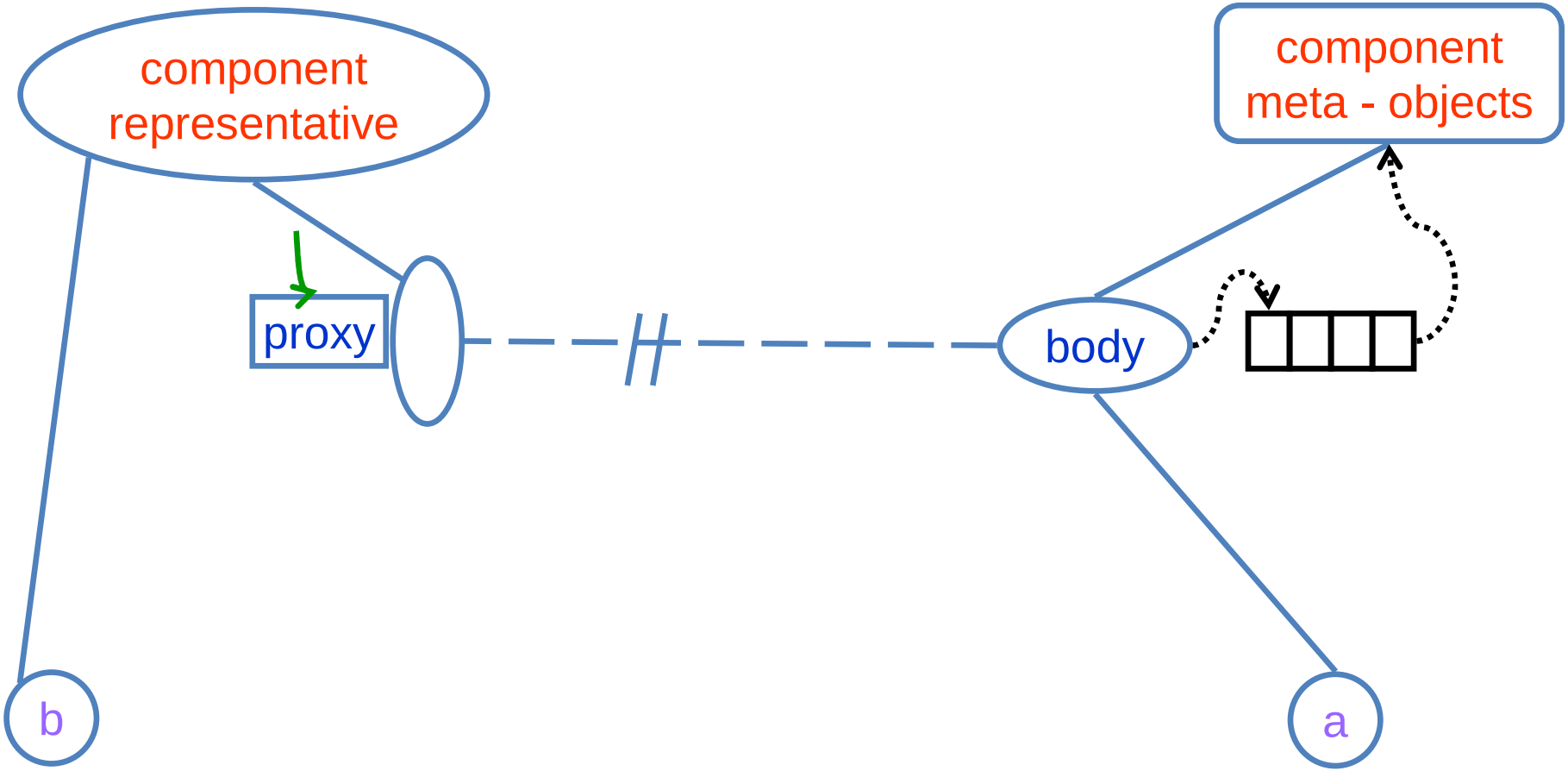
Architecture : component stubs



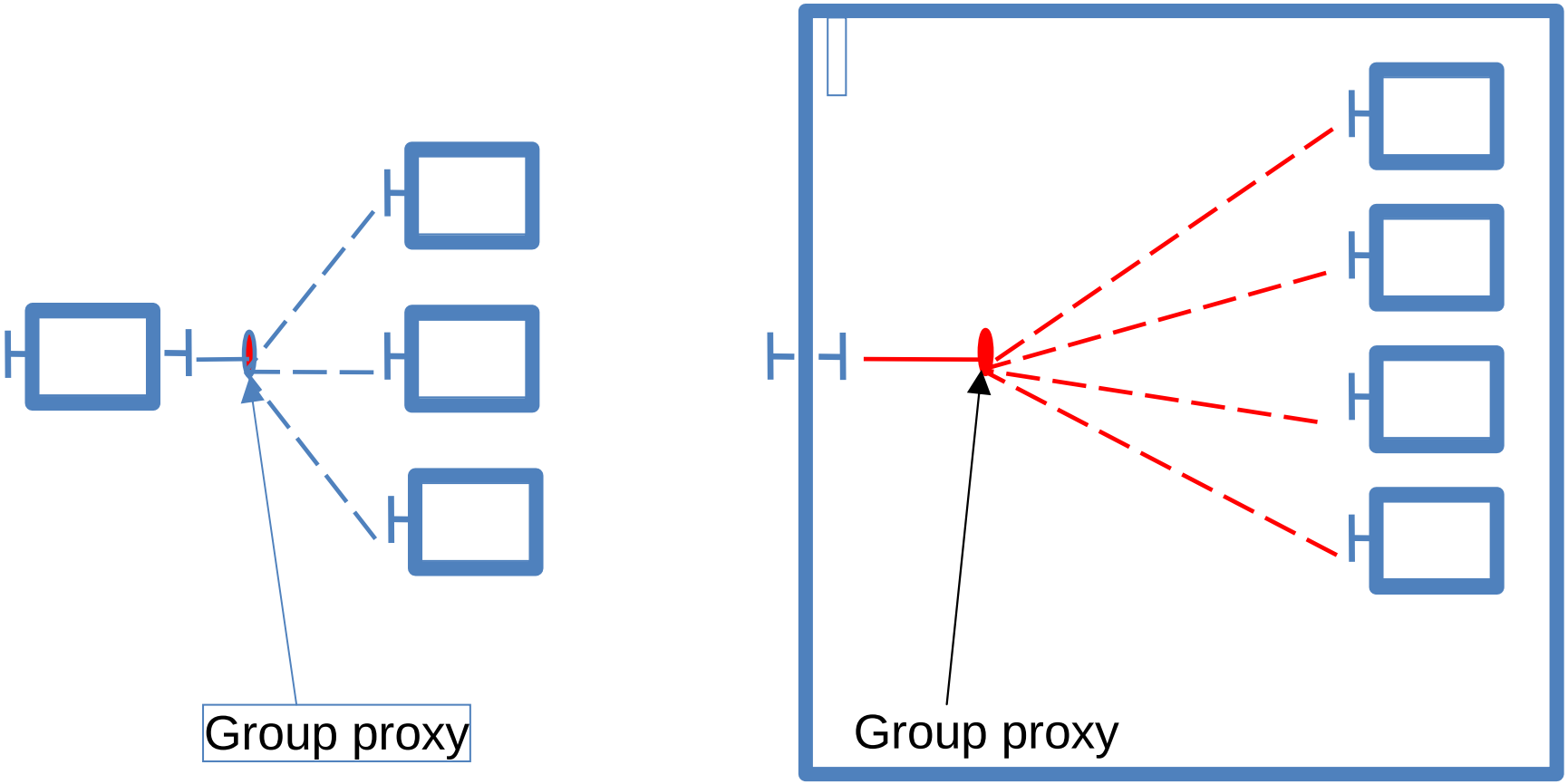
Architecture : communications



Architecture : request queue



Architecture : collective interfaces



➔ typed groups API
➔ Management of Collective Client Interface
➔ broadcast or scattering

Architecture : sum-up

Design

- component meta-object
- component stub (representative)
- Fractal interfaces / functional interface references
- 1 component = 1+ active object(s)

Communications

- reification
- request queue
- collective interfaces :
  - typed groups API

Using the ProActive implementation

Code

Standard Fractal code

```
fractal.provider =
  org.objectweb.proactive.core.component.Fractive
```

Implementation-specific parameters for instantiation

```
Component c = componentFactory.newFcInstance(     Type type,
  ControllerDescription controllerDesc, ContentDescription
  contentDesc);
```

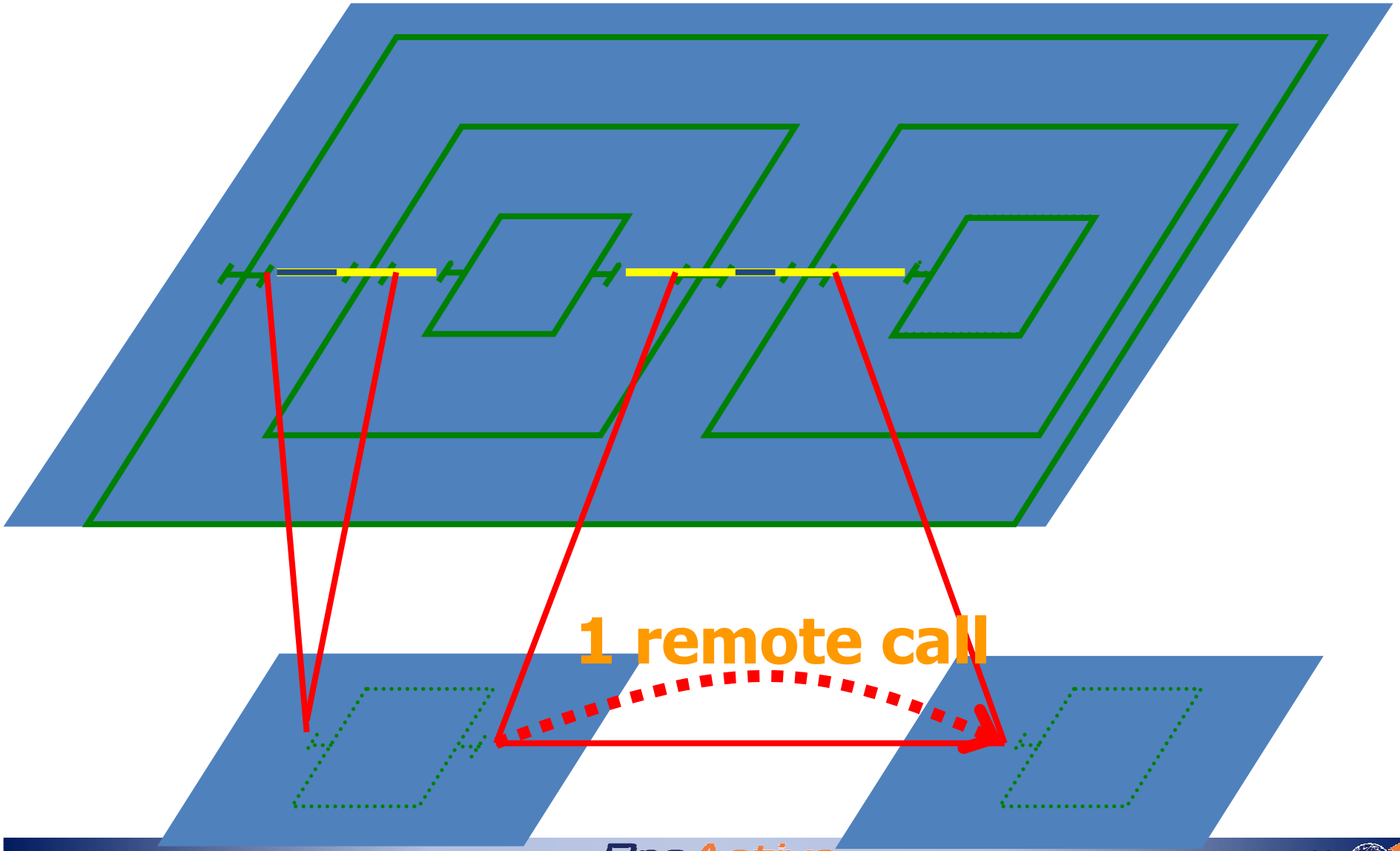
Collective interfaces

```
I i1 = Fractive.createCollectiveClientInterface(
  String itfName, String itfSignature);
ProActiveGroup.getGroup(i1).add(serverItf);
i1.foo(toto); // scattered or broadcasted
```

no templates

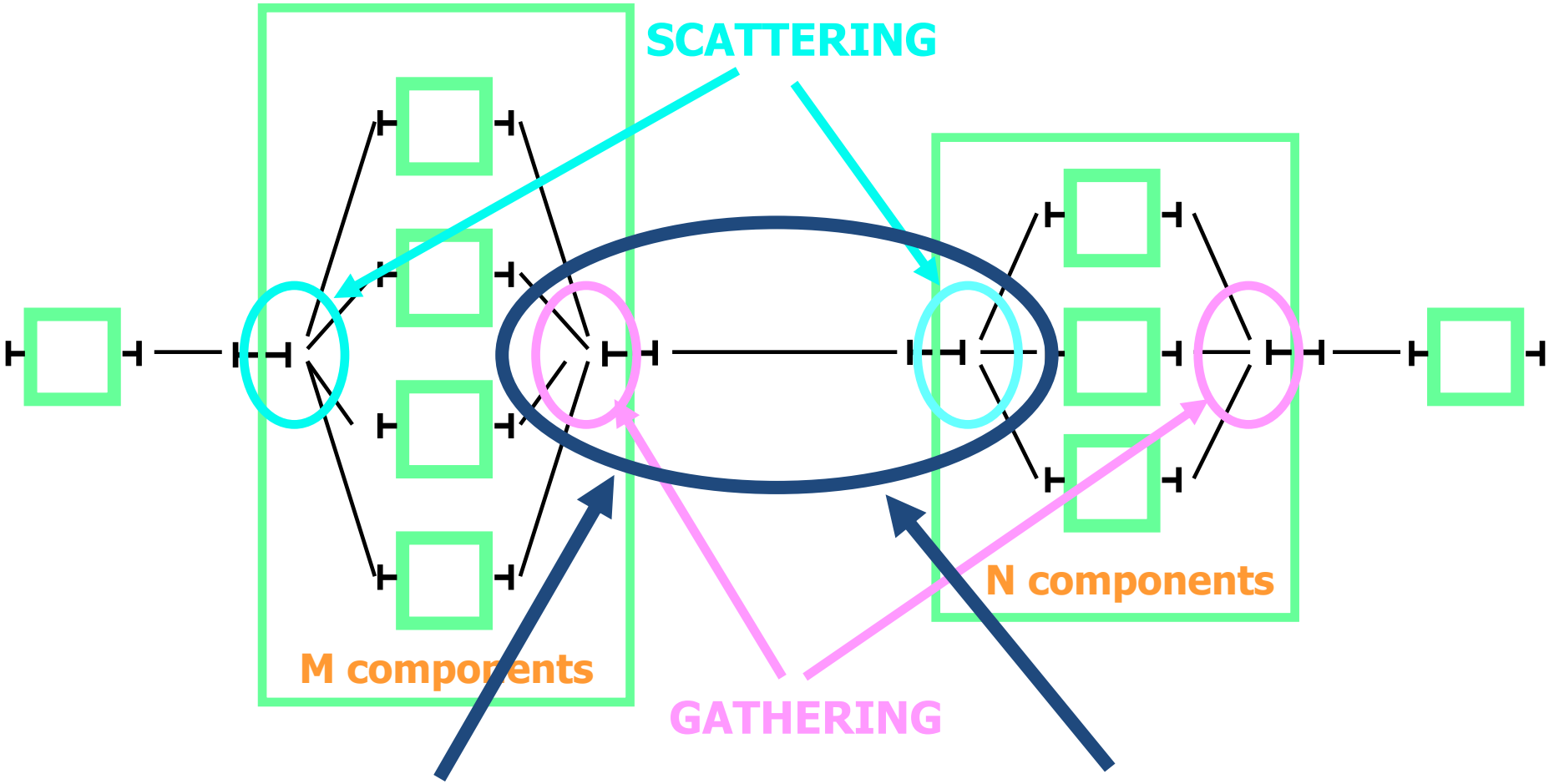
On-going work : optimizations

Dynamic shortcuts for distributed bindings (through tensioning)



On-going : MxN communications

Control at binding points



REDISTRIBUTION from M to N also, Functional Code

GCM GridCOMP

Grid Standard – EU Project

GCM Components

GCM: Grid Component Model

- GCM Being defined in the NoE CoreGRID
  (42 institutions)
- Open Source ObjectWeb *ProActive*
  implements a preliminary version of GCM

GridCOMP takes:

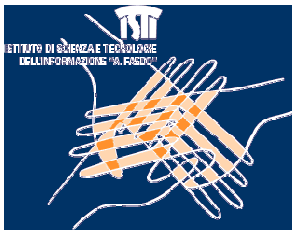
- GCM as a first specification,
- *ProActive* as a starting point, and
- Open Source reference implementation.

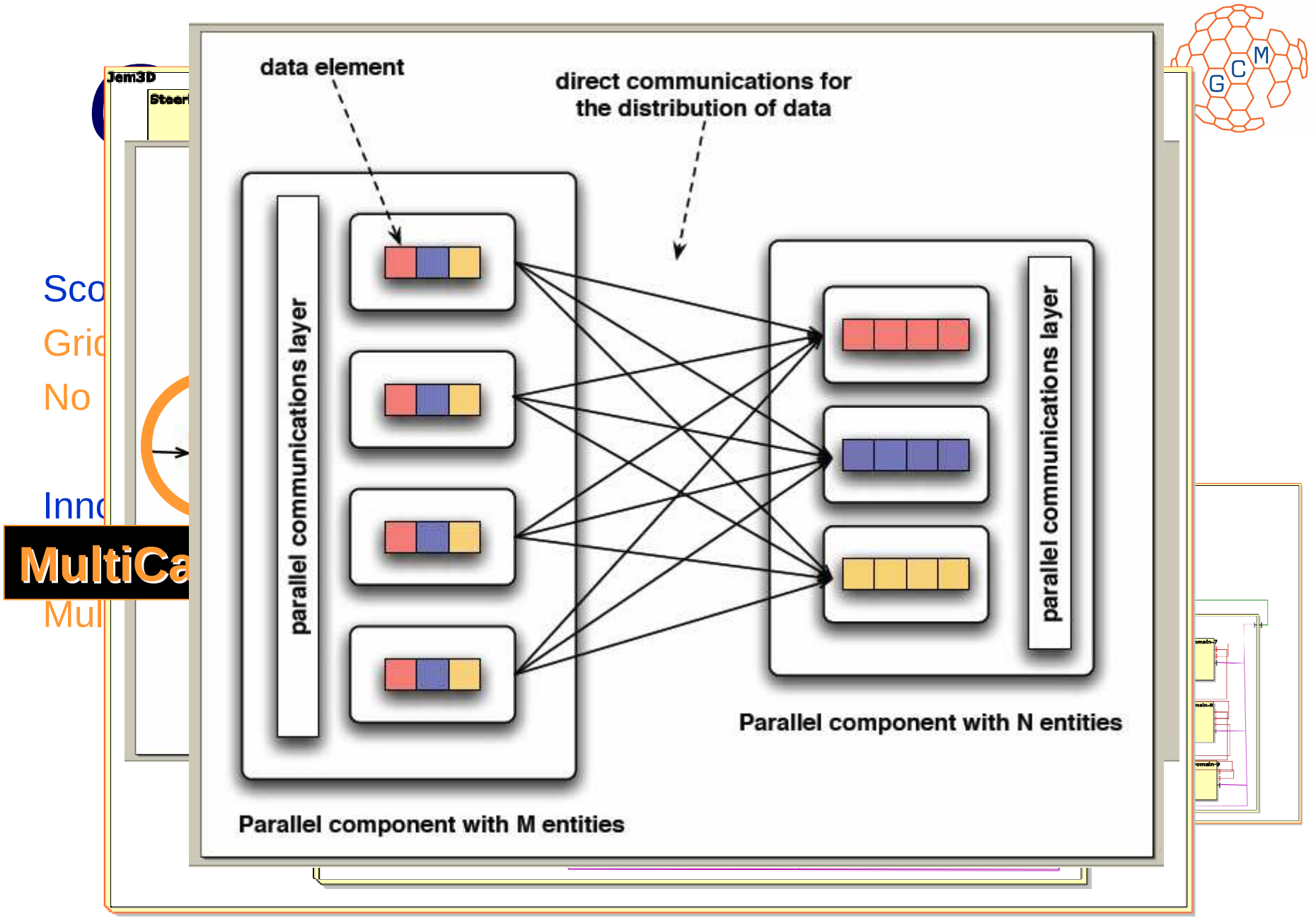
The vision: GCM to be the GRID GSM

Scopes and Objectives:

- Grid Codes that Compose and Deploy
- No programming, No Scripting, … No Pain

GCM Partners




data element
direct communications for the distribution of data
parallel communications layer
parallel communications layer
Parallel component with N entities
Parallel component with M entities
Jem3D
Steer
Sco
Gric
No
Inno
MultiCa
Mul

Collective Interfaces

Collective Interfaces

Simplify the design and configuration of component systems

Expose the collective nature of interfaces

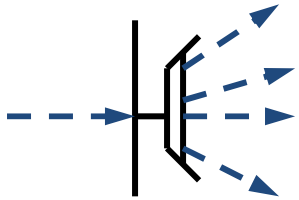
- Cardinality attribute
- Multicast, Gathercast, gather-multicast

The framework handles collective behaviour
at the level of the interface

Based on ObjectWeb Fractal API :

- Dedicated controller
- Interface typing ➔ Verifications

Multicast interfaces

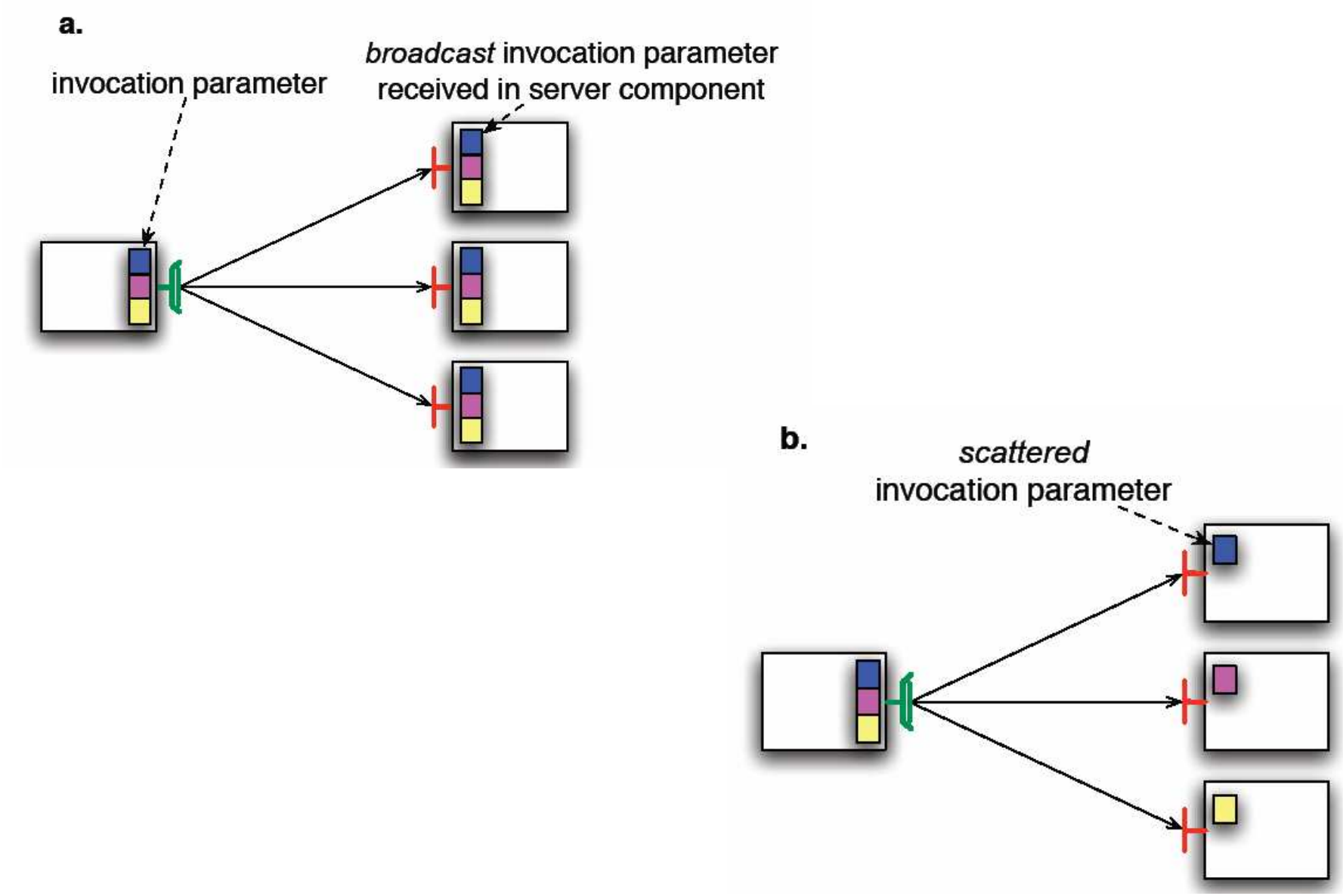


Transform a single invocation into a list of invocations

Multiple invocations
- Parallelism
- Asynchronism
- Dispatch

Data redistribution (invocation parameters)
- Parameterizable distribution function
- Broadcast, scattering
- Dynamic redistribution (dynamic dispatch)

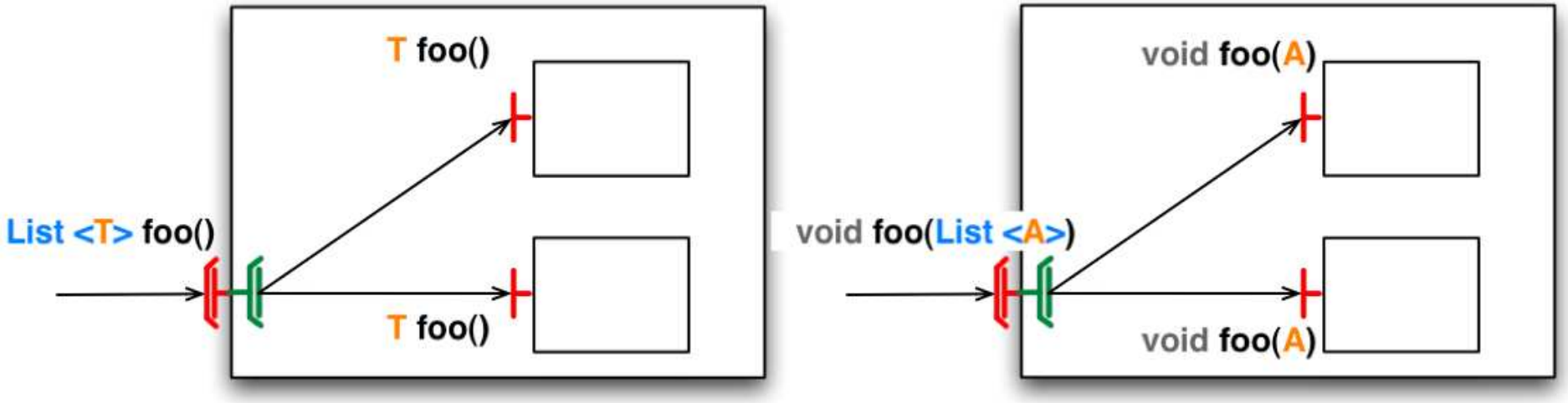
Result = list of results


a.
invocation parameter
broadcast invocation parameter
received in server component
b.
scattered
invocation parameter

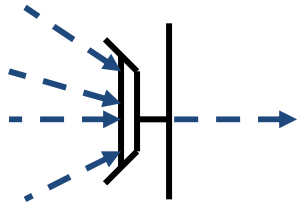
Multicast interfaces

Results as lists of results

Invocation parameters may also be distributed from lists



Gathercast interfaces



Transform a list of invocations into a single invocation

Synchronization of incoming invocations

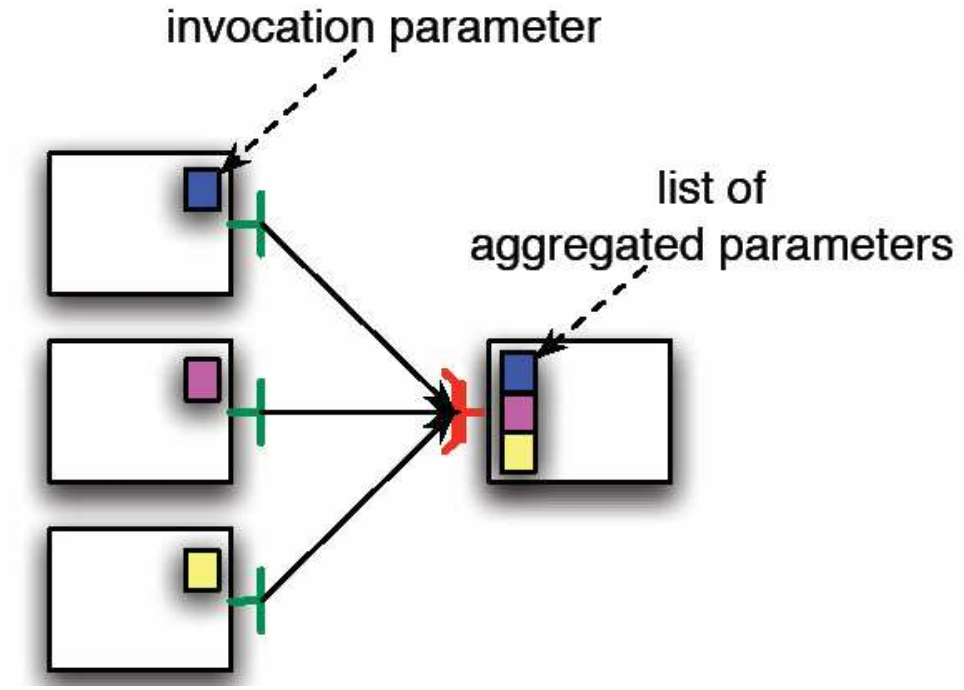
- ~ “join” invocations
- Timeout / Drop policy
- Bidirectional Bindings (callers ⇔ callee)

Data gathering

- Aggregation of parameters into lists

Result: redistribution of results

Redistribution function



Virtual Nodes

```
<virtualNodesDefinition>
  <virtualNode name="Dispatcher" property="unique_singleAO"
      />
  <virtualNode name="Renderer" property="Multiple"
               constraintFile="RendererConstraints.xml" />
</virtualNodesDefinition>
```

Permits a program to generate automatically a deployment plan:

find the appropriate nodes on which processes should be launched.

In the future, we envisage the adjunction of more sophisticated descriptions of the application needs with respect to the execution platform: topology, QoS, …

Virtual Nodes in the ADL

```
<exportedVirtualNodes>
  <exportedVirtualNode name="VN1">
    <composedFrom>
      <composingVirtualNode component="this" name="myNode"/>
    </composedFrom>
  </exportedVirtualNode>
</exportedVirtualNodes>
...
<virtual-node name="myNode" cardinality="single"/>
```

Renames a VN

Exports a VN name

The final version of the GCM specification will precisely define the syntax for the virtual node definition, and their composition.

High-Level Parallel Programming

Skeletons

High Level Patterns (Skeletons)

Skeletons exploit nestable parallelism patterns.

Primitive: **seq** : Sequential wrapper.

Task Parallelism (Several initial data processed in parallel)

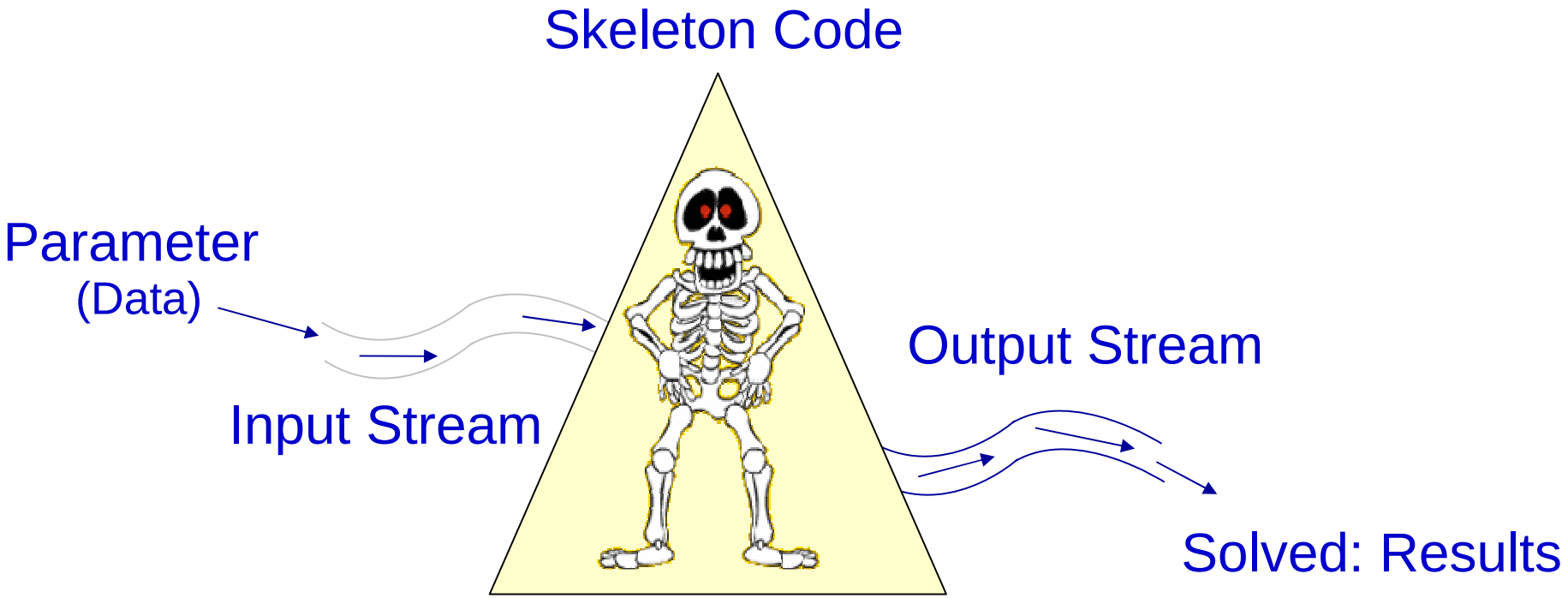
- **farm**: Task replication.
- **pipe**: Staged computation.
- **if**: Conditional Computation.
- **while**: Cycles.
- **for**: Iteration.

Data Parallelism (Initial Data is cut to be processed in parallel)

- **divide and conquer**: divide a problem into solvable sizes.
- **map**: apply *the same* function to many data elements.
- **fork**: apply *different* functions to many data elements.

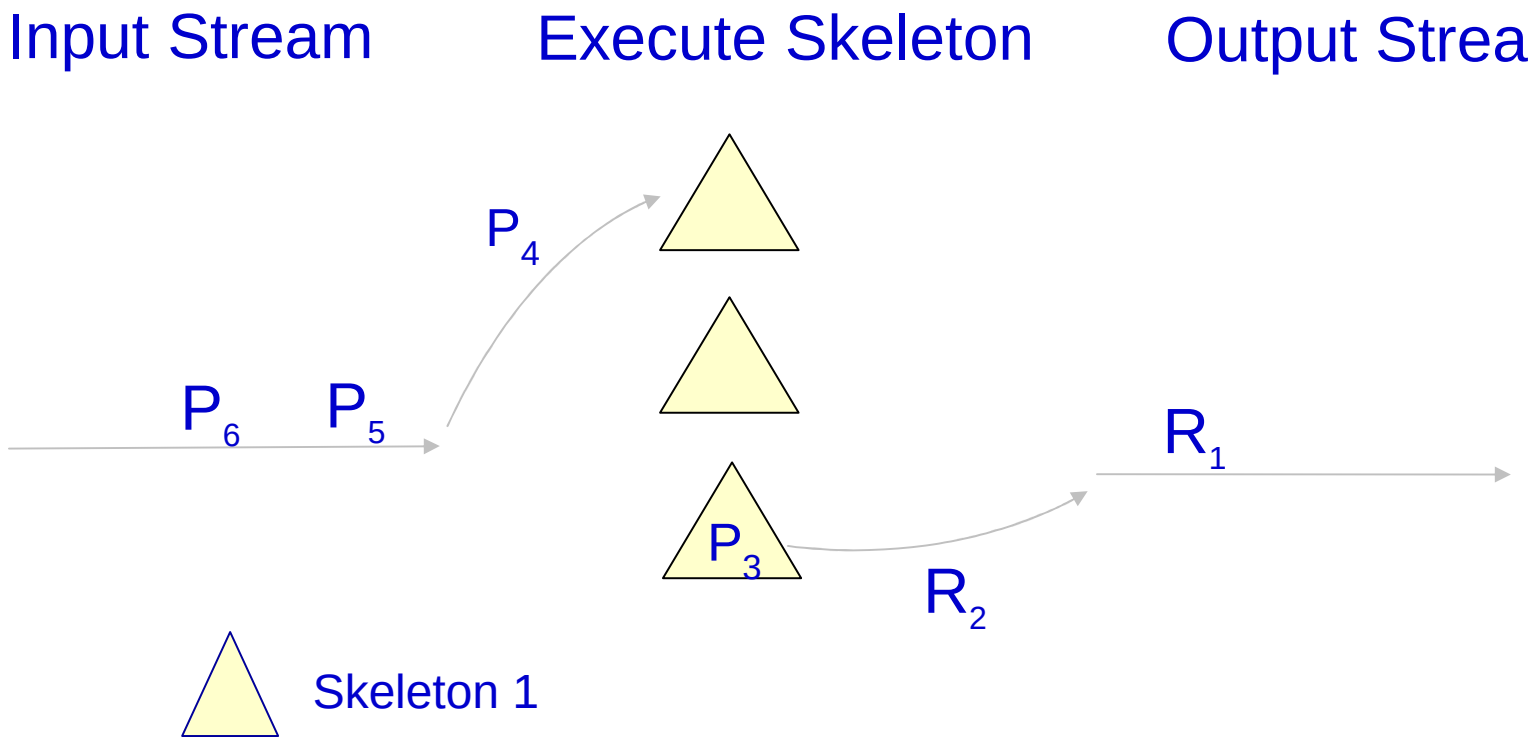
Skeletons Big Picture

Parameters/Results are passed through streams
Streams are used to connect skeletons (CODE)



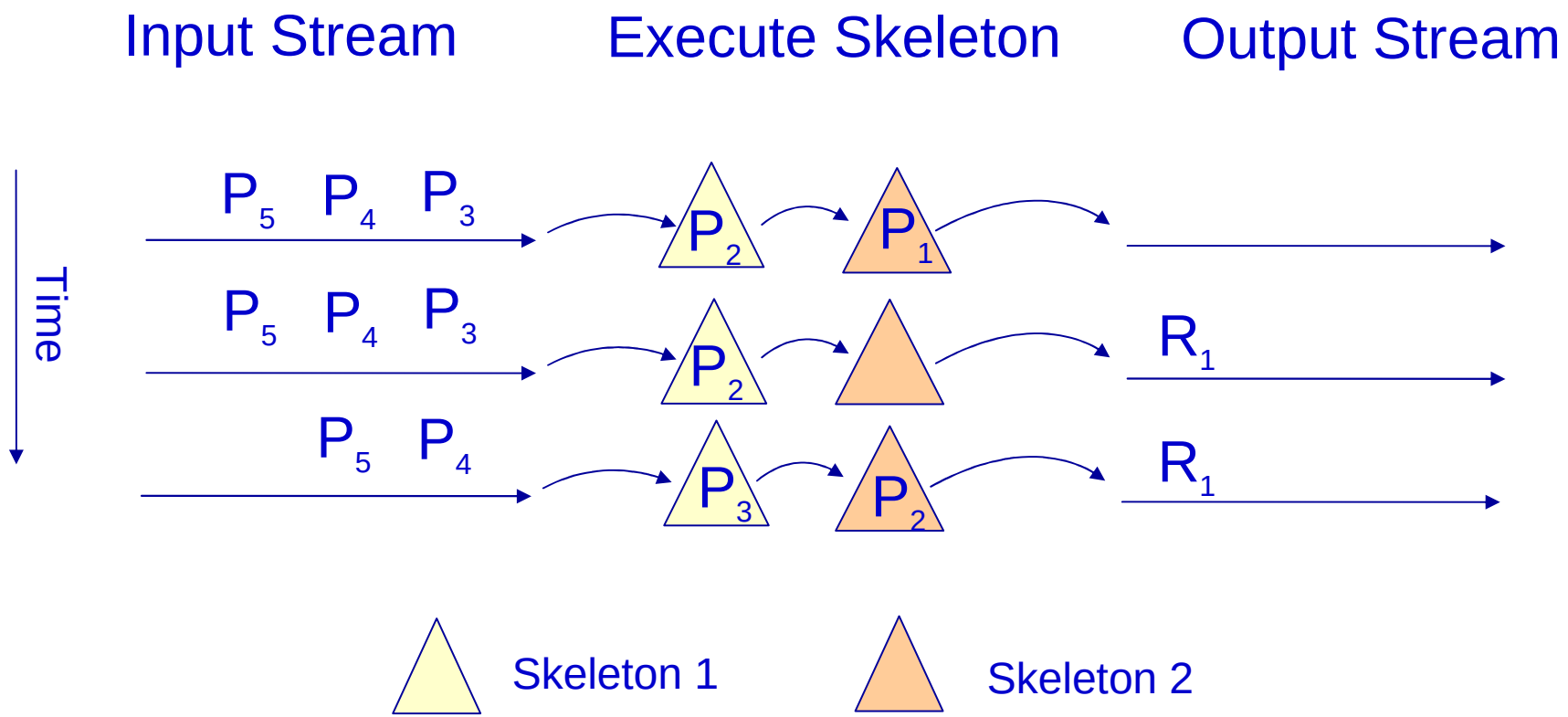
Farm Skeleton

- Also known as **master-slave**, represents replication.
- Different parameters are computed in parallel for the same skeleton.



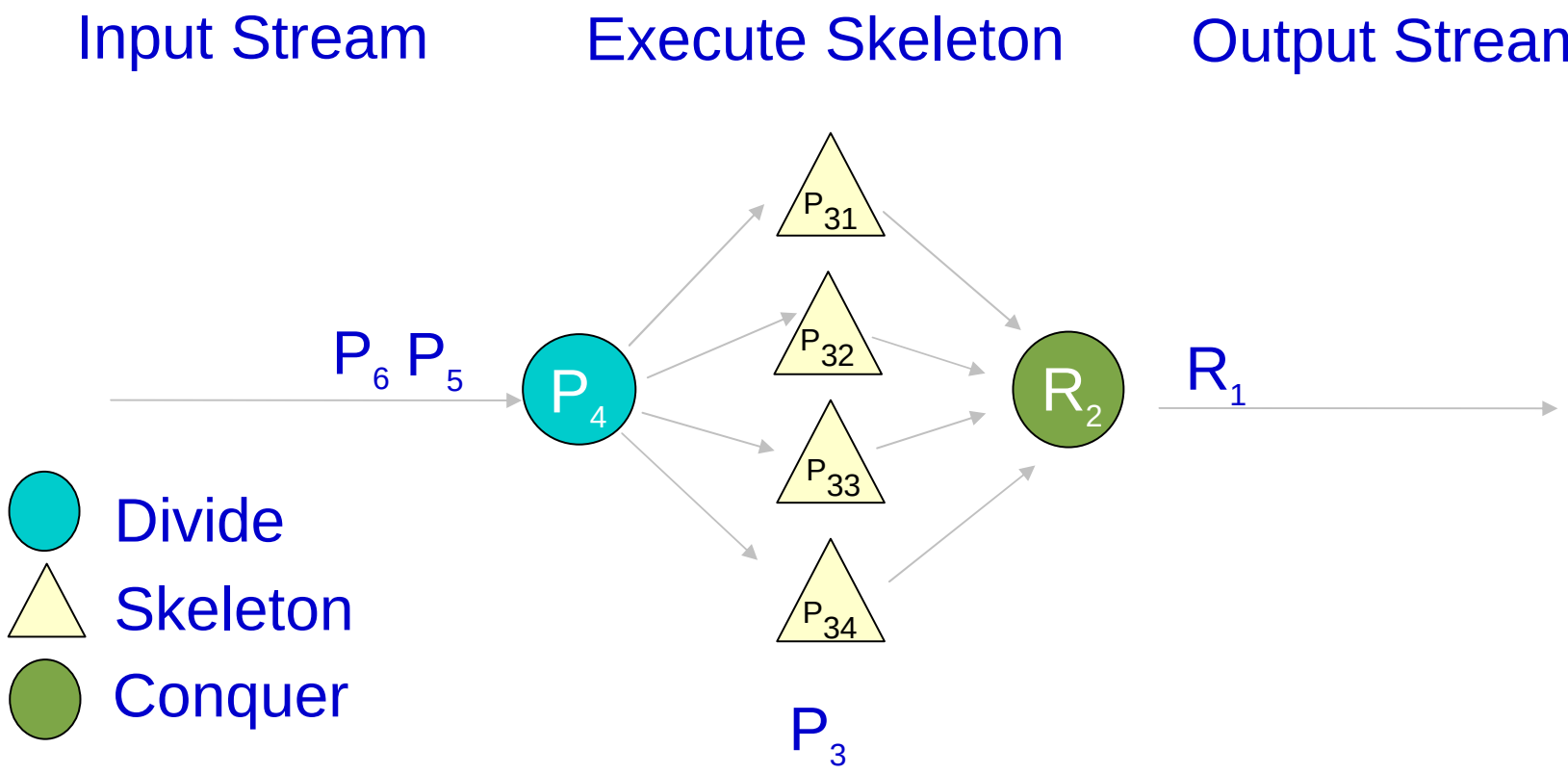
Pipe Skeleton

- Represents computation by stages.
- Stages are computed in parallel for different parameters.



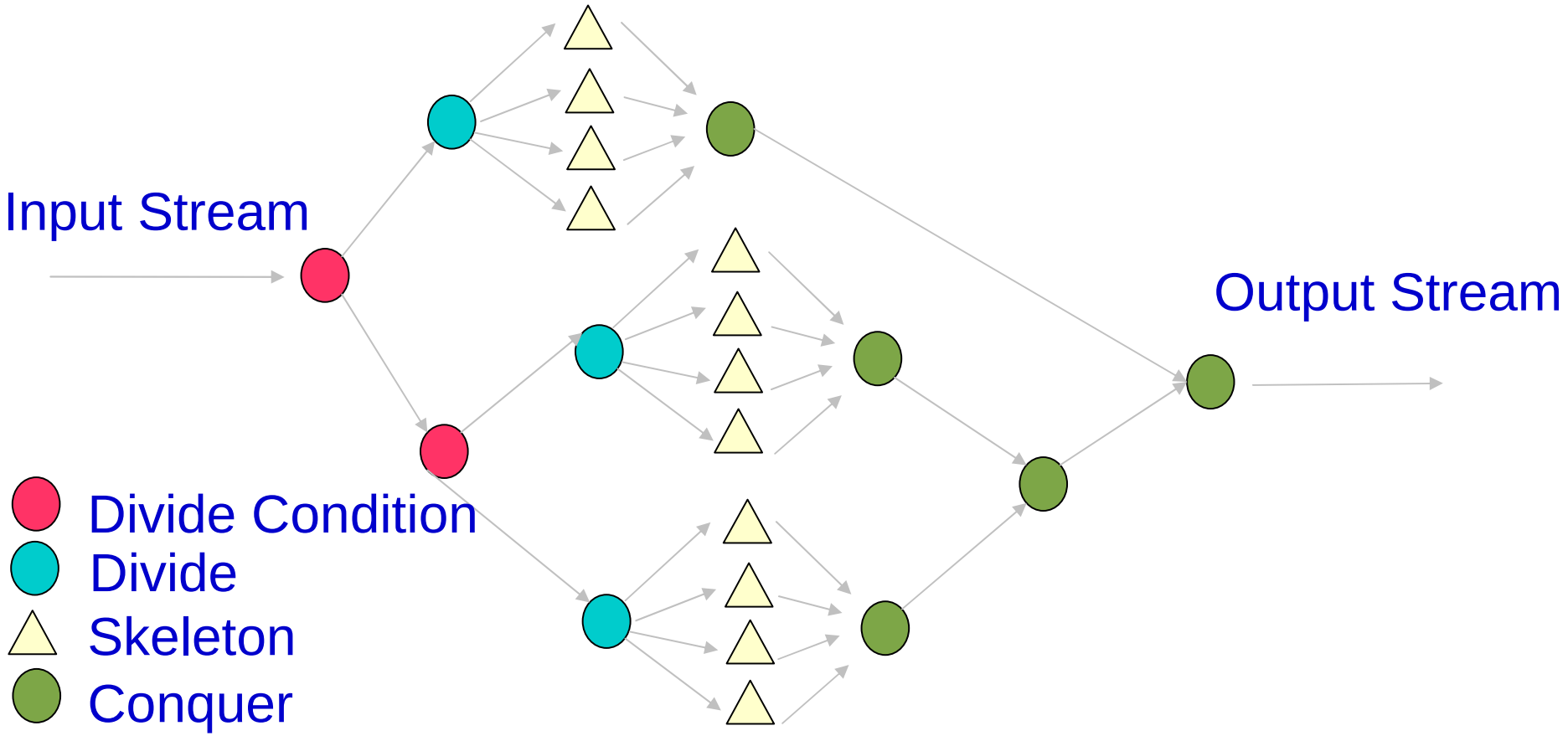
Map Skeleton

- Represents Data Parallelism
- A function is applied to each of the data parts in parallel.

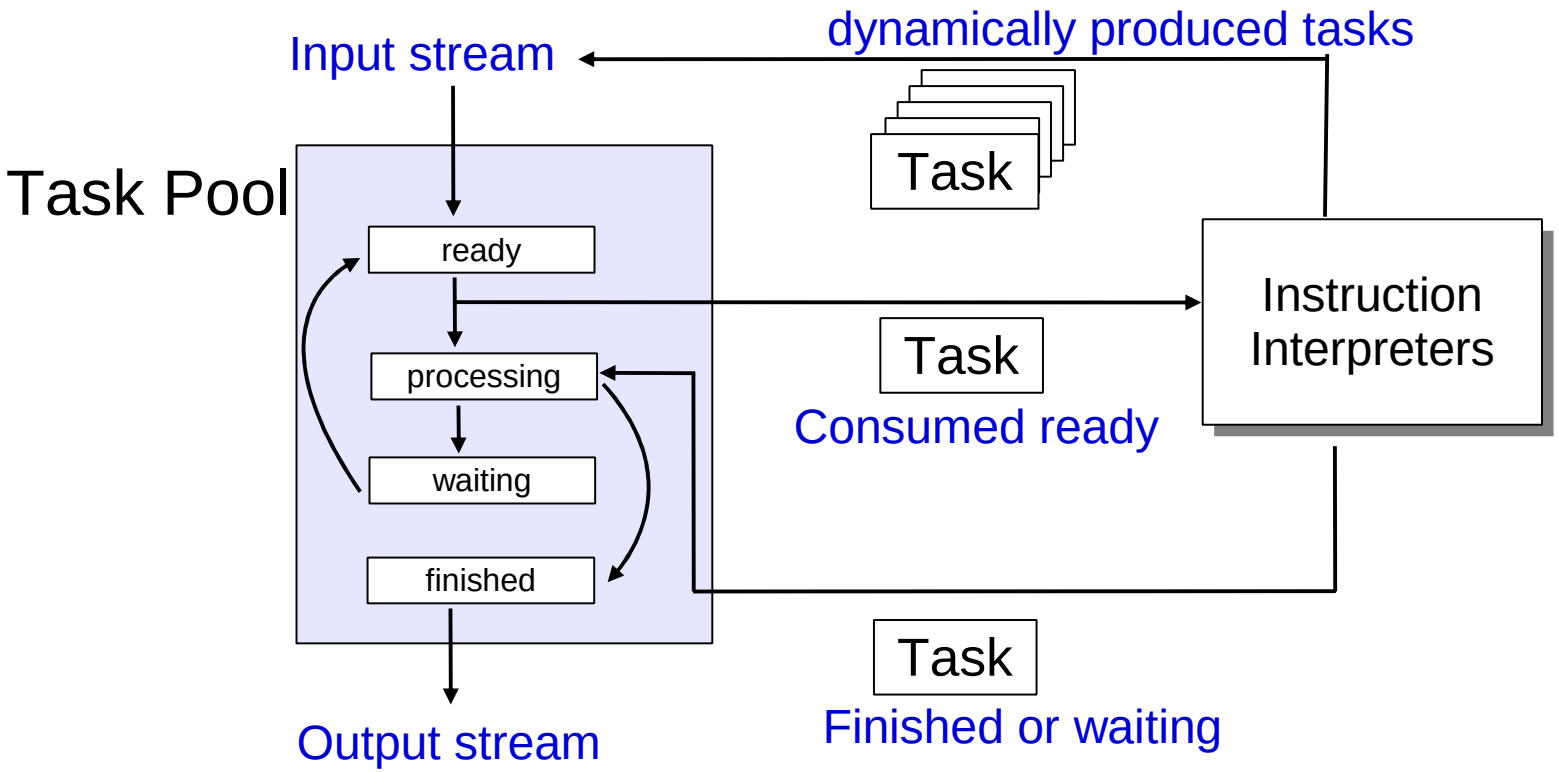


Divide and Conquer Skeleton

- Represents Data Parallalism

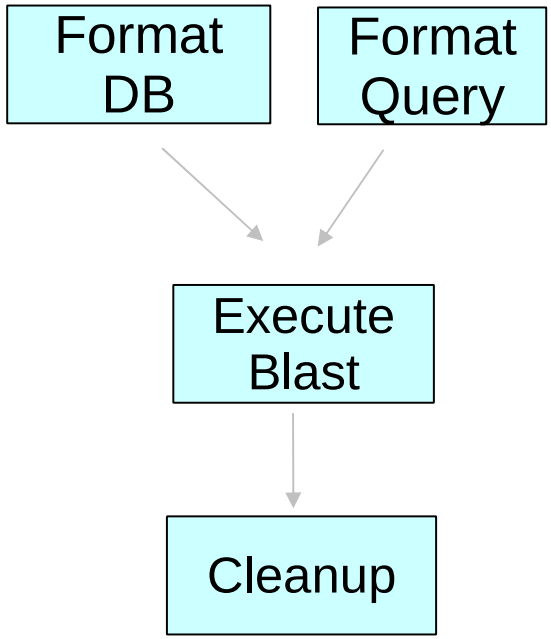


ProActive's Skeleton Framework

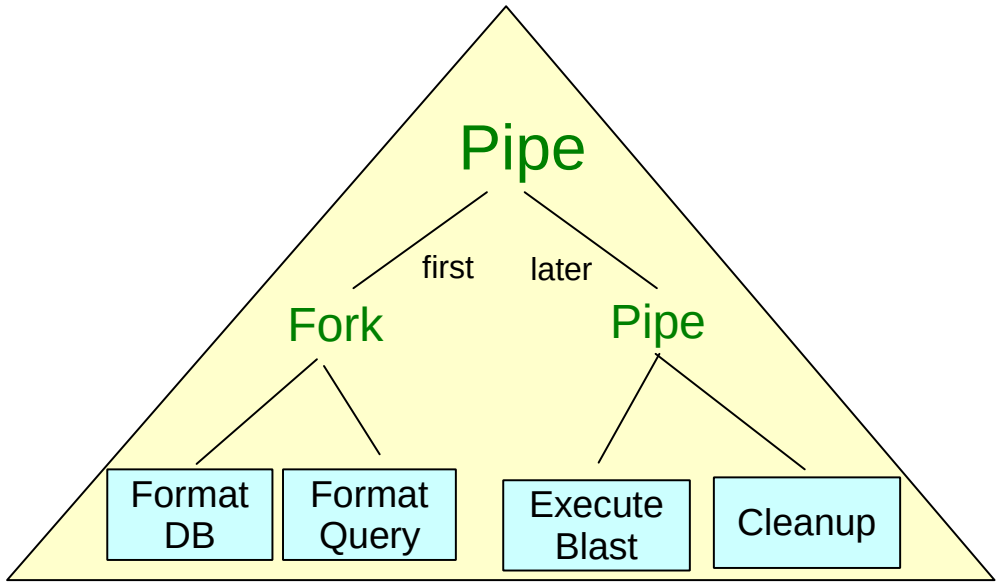


BLAST Skeleton Example

- BLAST is the sequence alignment tool (bio-informatics).



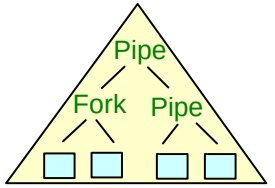
Blast Pipe Skeleton



BLAST Skeleton Example

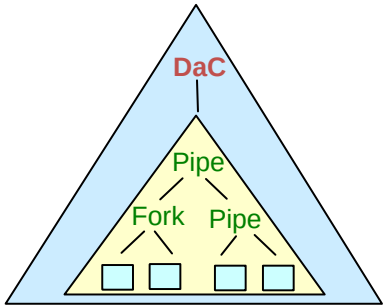
```
Pipe blastPipe = new Pipe(
                    new Fork(
                        new Seq(new ExecuteFormatDB()),
                        new Seq(new ExecuteFormatQuery())),
                    new Pipe(
                        new Seq(new ExecuteBlast()),
                        new Seq(new CleanBlast())));
```

BLAST Skeleton




```
Skeleton root = new DaC(
            new DivideDB(), new DivideCondition(),
                    blastPipe, new ConquerResults());
```

BLAST Divide & Conquer Skeleton



Divide and Conquer Blast

- Divide Condition: While database > X[MB].
- Divide: Split database into Y parts.
- Execute: Blast Skeleton.
- Conquer: Merge the results.

Example of Skeleton Code

```
public class DivideCondition implements Condition {

   public boolean evalCondition(Blast param) {
        File file = param.getDatabaseFile();
        return file.length() > param.getMaxDBSize();
    }
}
```

Divide if this condition is true

```
public class DivideDB implements Divide{

   public Vector divide(Blast param) {
      Vector<BlastParameters> children = new Vector<BlastParameters>();
      Vector<File> files  files = divideDBFile(param);
       for (File newDBFile : files) {
           Blast newParam = new Blast(param);
           param.setDBName(newDBFile);
           children.add(newParam);
       }
       return children;
   }
}
```

Divide DB File into smaller parts

Skeleton Framework

```
Skeleton root = ...;

ResourceManager manager =
            new ProActiveManager(“descriptor.xml”, "vn-name");

Calcium calcium = new Calcium(manager);
Stream stream = calcium.getStream(root);
```

Deployment & Instantiation

```
stream.input(new Blast(“db-file”,”query-file1”));
stream.input(new Blast(“db-file”,”query-file2”));
stream.input(new Blast(“db-file”,”query-file3”));
```

Input Parameters

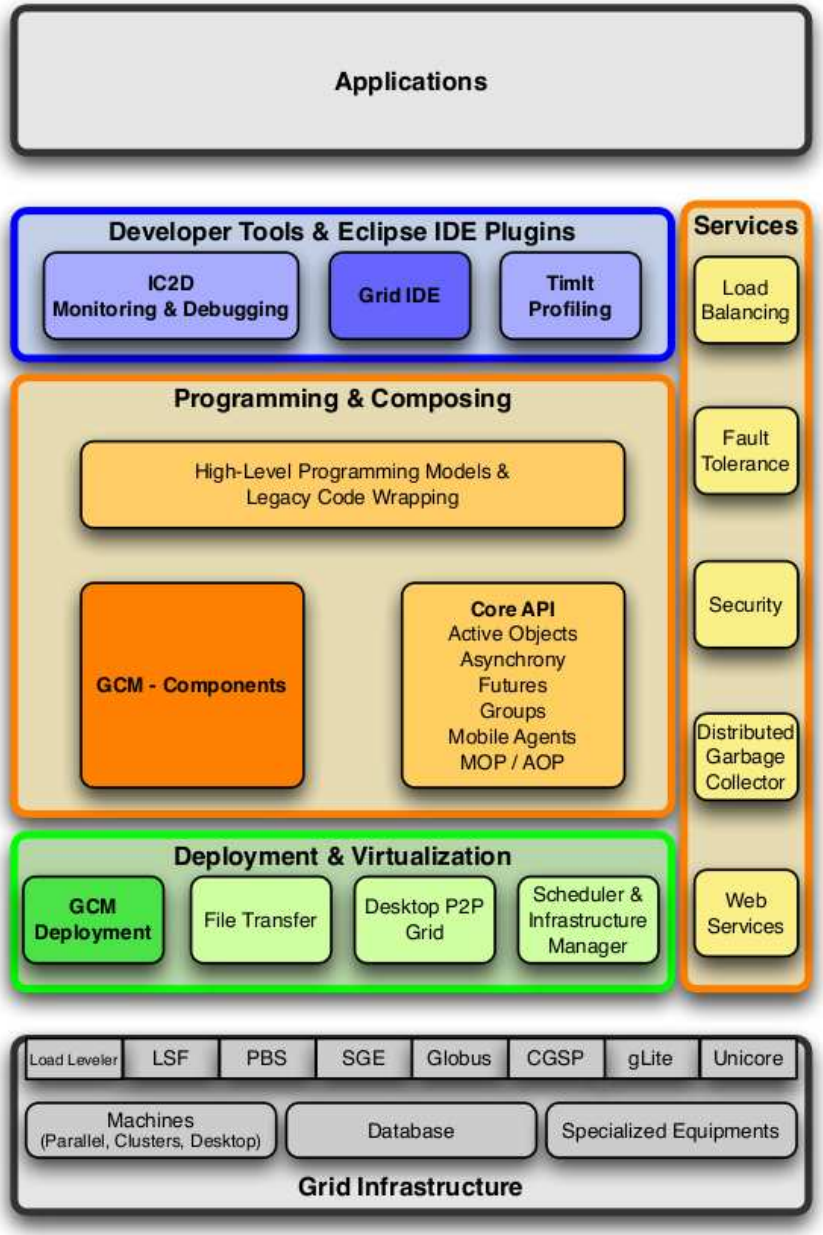
```
Blast res1 = stream.getResult();
Blast res2 = stream.getResult();
Blast res3 = stream.getResult();
```

Get results

4.1 Environment

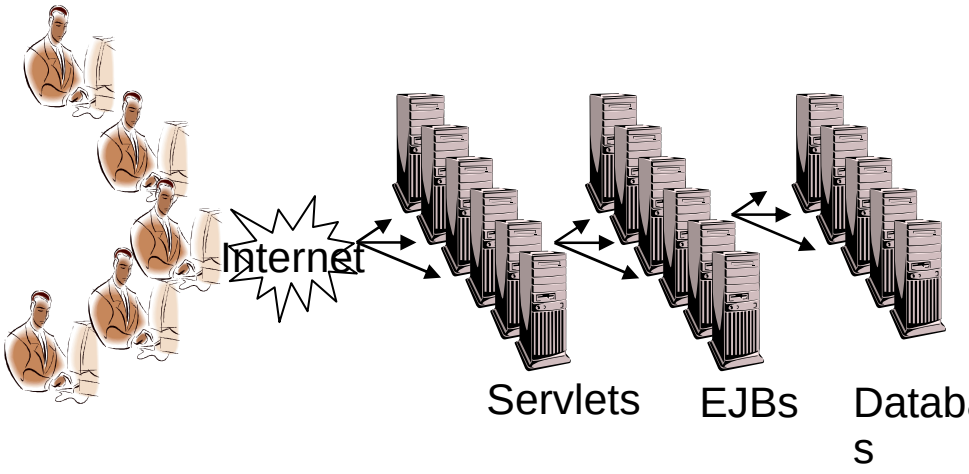
Deploying

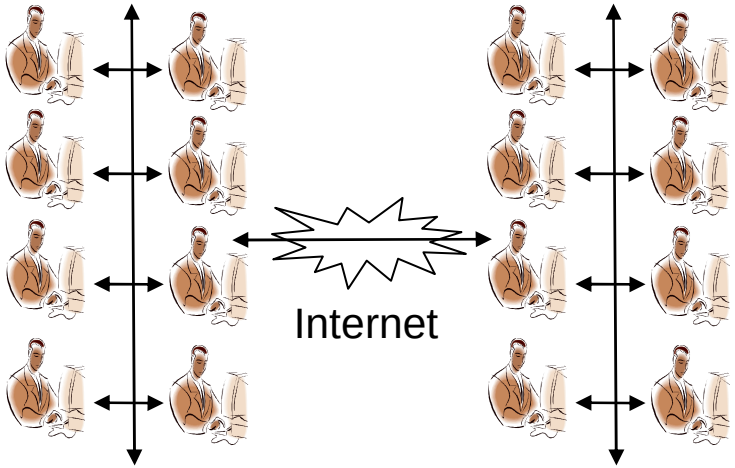
ProActive Parallel Suite (1)

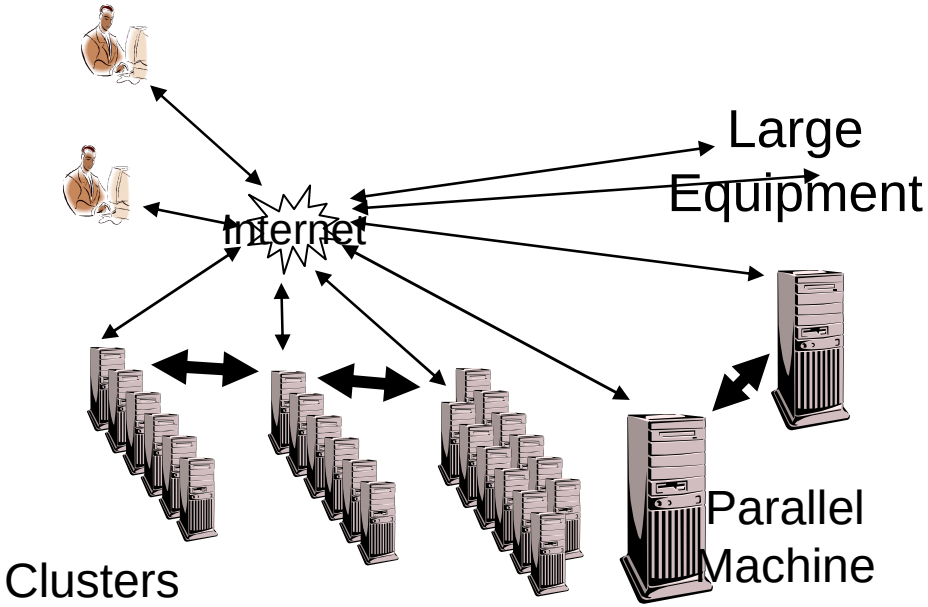


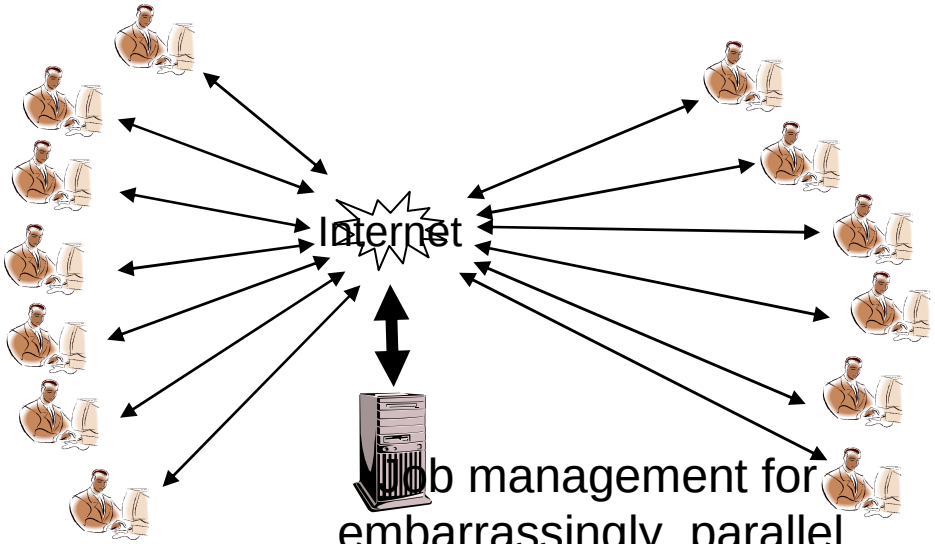
ProActive Parallel Suite (1)

How to deploy on the Various Kinds of Grid









4.1 Abstract Deployment Model

Problem:

Difficulties and lack of flexibility in deployment

Avoid scripting for: configuration, getting nodes, connecting, etc.

A key principle: Virtual Node (VN) + XML deployment file

Abstract Away from source code:

- Machines
- Creation Protocols
- Lookup and Registry Protocols

Protocols and infrastructures:

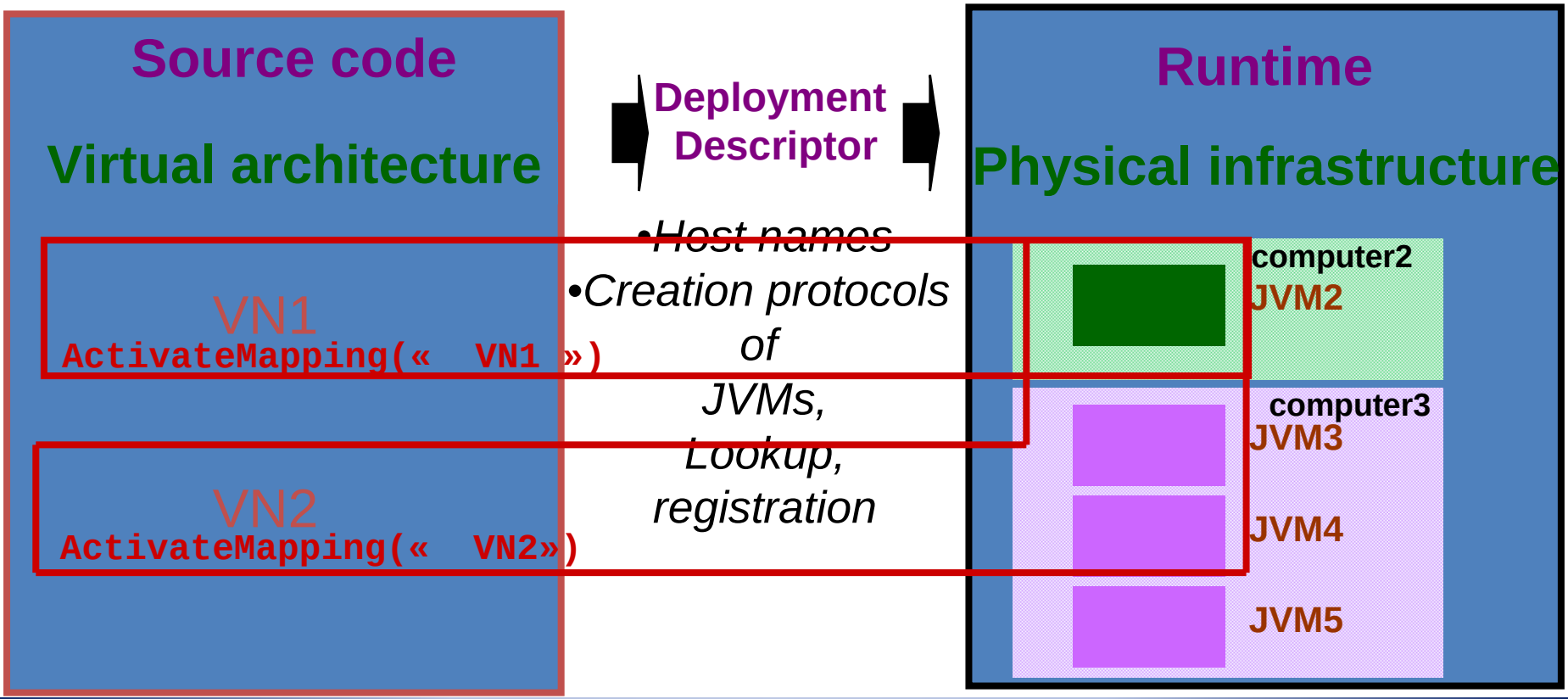
Globus, ssh, rsh, LSF, PBS, … Web Services, WSRF, ...

Abstract deployment model

Separates design from deployment infrastructure

Virtual nodes

Dynamic enactement of the deployment, from the application



Context

Grid

Scalable

Heterogeneous resources

- OS, CPU, Memory
- Security policies: firewall, NAT, private IP addresses,…
- Model,...

Painless deployment

Based on well-known technologies

- Java, XML

Abstract Deployment model

Abstract Deployment Model

Problem:

Difficulties and lack of flexibility in deployment

Avoid scripting for: configuration, getting nodes, connecting, etc.

A key principle: Virtual Node (VN) in XML deployment file

- **Abstract Away from source code:**
  - Machines names
  - Creation/Connection Protocols
  - Lookup and Registry Protocols
- **Interface with various protocols and infrastructures:**
  - Cluster: LSF, PBS, SGE , OAR and PRUN(custom protocols)
  - Intranet P2P, LAN: intranet protocols: rsh, rlogin, ssh
  - Grid: Globus, Web services, ssh, gsissh

XML Deployment files

Virtual Node (VN):

- **Identified as a string name**
- **Used in program source**
- **Configured (mapped) in the XML descriptor file --> Nodes**

Operations specified in descriptors:

- **Mapping of VN to JVMs (leads to Node in a JVM on Host)**
- **Register or Lookup VNs**
- **Create or Acquire JVMs**
- **Security**

Program Source	Descriptor (RunTime)
Activities (AO)--> VN--> Nodes	VN --> JVMs --> Hosts

Runtime structured entities: 1 VN --> n Nodes in m JVMs on k Hosts

Descriptors: Virtual Nodes in Programs

```
Descriptor pad = ProActive.getDescriptor("file:DeploymentDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");
// Triggers the JVMs and Nodes creation
Node node = vn.getNode();
C3D c3d = ProActive.newActive("C3D", param, node);
log (  ... "created at:  " + node.name() + node.JVM() + node.host() );
```

Descriptors: Virtual Nodes in Programs

```
Descriptor pad = ProActive.getDescriptor("file:DeploymentDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");
    // Triggers the JVMs and Nodes creation
Node node = vn.getNode();
C3D c3d = ProActive.newActive("C3D", param, node);
log (  ... "created at:  " + node.name() + node.JVM() + node.host() );
```

// Cyclic mapping: set of nodes

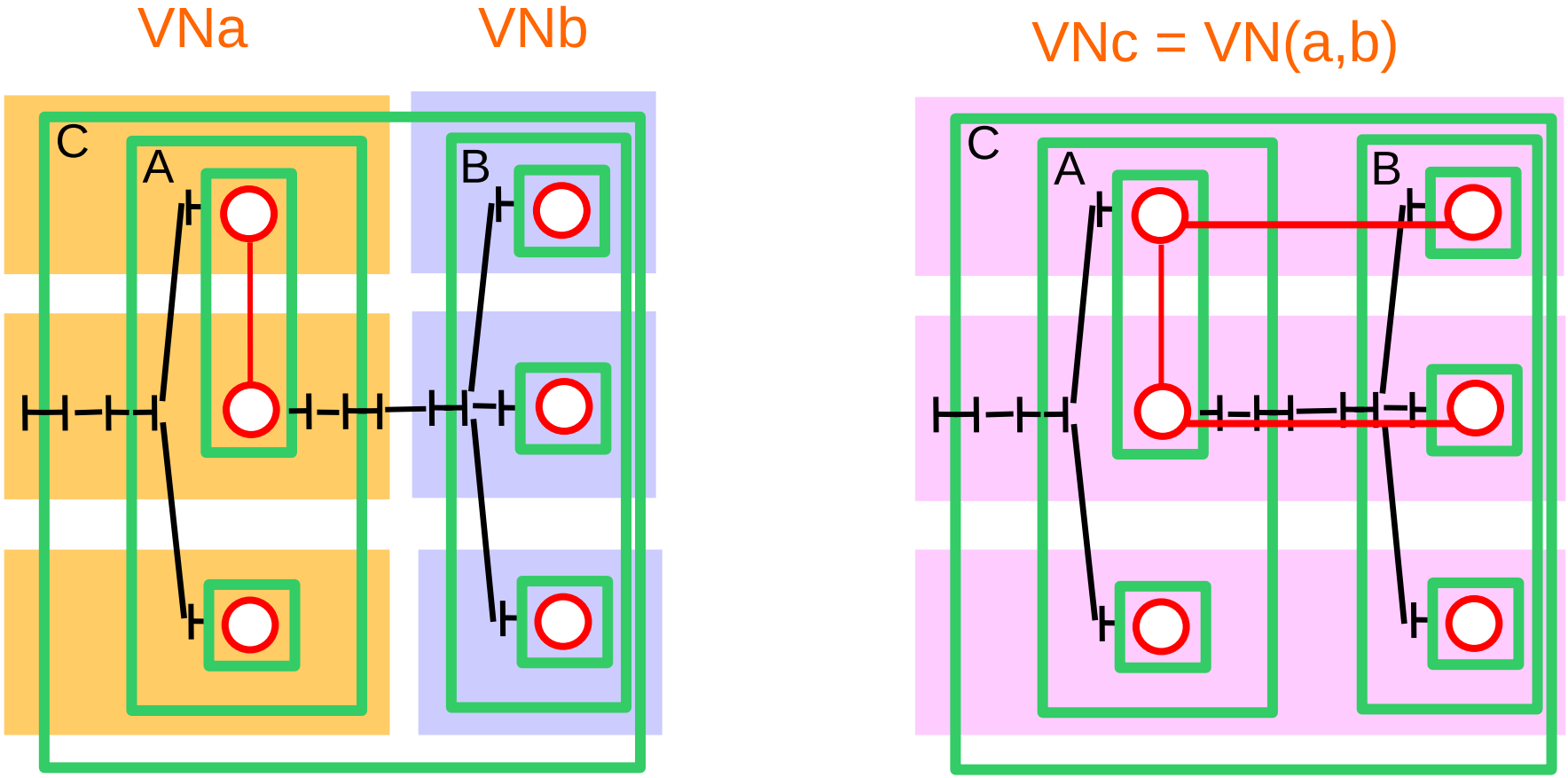
```
VirtualNode vn = pad.activateMapping ("RendererSet");
while  (  … vn.getNbNodes … ) {
    Node node = vn.getNode();
    Renderer  re = ProActive.newActive(”Renderer", param, node);
```


Descriptors: Mapping Virtual Nodes

Example of an XML file descriptor:

```
Component Dependencies:
  Provides: …  Uses: ...
VirtualNodes:
  Dispatcher  <RegisterIn RMIregistry, Globus, Grid Service, … >
  RendererSet
Mapping:
  Dispatcher --> DispatcherJVM
  RendererSet --> JVMset
JVMs:
  DispatcherJVM = Current  // (the current JVM)
  JVMset=//ClusterSophia.inria.fr/ <Protocol GlobusGram … 10 >
...
```


XML Deployment (Not in source)



Separate or Co-allocation

Model and Tools for Grid Deployment

Abstract away machines, creation, registry, lookup protocols

Use only **Virtual Nodes** in the source code

Describe mapping of virtual nodes in XML Deployment Descriptors

Interfaced with various protocols for creation/lookup:
rsh,ssh,Jini, LSF,PBS,Globus,OGSA...

Unification of various deployment systems through ProActive runtimes

VirtualNodes:

Dispatcher <RegisterIn RMIregistry, Globus, Grid Service, …>

RendererSet

Mapping:

Dispatcher --> Dispatcher

RendererSet --> JVMset

JVMs:

DispatcherJVM = <javapath …/> <classpath…/> sshProcess

JVMset = <javapath …/> <classpath…/> GlobusProcess

Processes:

sshProcess = <hostname di.unice.fr/> <login../>

GlobusProcess = <hostname cluster.inria.fr/> <gram port 2119/> <node 10/>

Other option: acquire ProActive runtimes from voluntary PCs (P2P computing) running a ProActive P2P infrastructure

Mapping Virtual Nodes: example (1)

Definitions and mapping

```
<virtualNodesDefinition>
    <virtualNode name="Dispatcher"/>
</virtualNodesDefinition>

<map virtualNode="Dispatcher">
    <jvmSet>
        <vmName value="Jvm1"/>
    </jvmSet>
</map>

<jvm name="Jvm1">
    <acquisition method="rmi"/>
    <creation>
        <processReference refid="linuxJVM"/>
    </creation>
</jvm>
```

Definition of Virtual Nodes

Mapping of Virtual Nodes

Mapping Virtual Nodes: example (2)

Definitions and mapping

```
<virtualNodesDefinition>
    <virtualNode name="Jem3DNode"/>
</virtualNodesDefinition>

<map virtualNode=" Jem3DNode">
    <jvmSet>
        <vmName value="clusterJvm"/>
    </jvmSet>
</map>

<jvm name="clusterJvm">
    <acquisition method="rmi"/>
    <creation>
        <processReference refid="clusterProcess"/>
    </creation>
</jvm>
```

Definition of Virtual Nodes

Mapping of Virtual Nodes

Mapping Virtual Nodes: example (3)

Infrastructure informations

```
<processDefinition id="linuxJVM">
    <jvmProcess
    class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>
```

JVM on the current Host

```
<processDefinition id="rshProcess">
    <rshProcess
    class="org.objectweb.proactive.core.process.rsh.RSHJVMProcess"
                hostname="sea.inria.fr">
        <processReference refid="linuxJVM"/>
    </rshProcess>
</processDefinition>
```

JVM started using RSH

Mapping Virtual Nodes: example (4)

Infrastructure information

```
<processDefinition id=”singleJVM">
    <jvmProcess
     class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>


<processDefinition id=" clusterProcess ">
  <bsubProcess
    class="org.objectweb.proactive.core.process.lsf.LSFBSubProcess"
                  hostname=”cluster.inria.fr">
    <processReference refid=”singleJVM"/>
    <bsubOption>
          <processor>12</processor>
     </bsubOption>
  </bsubProcess>
</processDefinition>
```

Definition of bsub process

Mapping Virtual Nodes: example (5)

Infrastructure information

```
<processDefinition id=" clusterProcess ">
  <bsubProcess
    class="org.objectweb.proactive.core.process.lsf.LSFBSubProcess"
                hostname=”cluster.inria.fr">
    <processReference refid=”singleJVM"/>
    <bsubOption>
          <processor>12</processor>
     </bsubOption>
  </bsubProcess>
</processDefinition>
```

Definition of LSF deployment, … Globus

Mapping Virtual Nodes: example (6)

Infrastructure information

```
<processDefinition id="linuxJVM">
    <jvmProcess
    class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>
```

JVM on the current Host

```
<processDefinition id="sshProcess">
    <sshProcess
    class="org.objectweb.proactive.core.process.ssh.SSHJVMProcess"
                hostname="sea.inria.fr">
        <processReference refid="linuxJVM"/>
    </sshProcess>
</processDefinition>
```

JVM started using SSH

Mapping Virtual Nodes: example (7)

Infrastructure information

```
<processDefinition id=" plugtestProcess ">
  <sgeProcess class="org.objectweb.proactive.core.process.sge.SGEProcess"
        hostname=" frontal">
    <processReference refid="localJVMprocess"/>
   <sgeOptions>
        <processor>12</processor>
    </sgeOptions>
  </sgeProcess>
</processDefinition>
```

Definition of SGE process

```
<processDefinition id="localJVMprocess">
        <jvmProcess class="org.objectweb.proactive.core.process.
JVMNodeProcess"/>
</processDefinition>
```

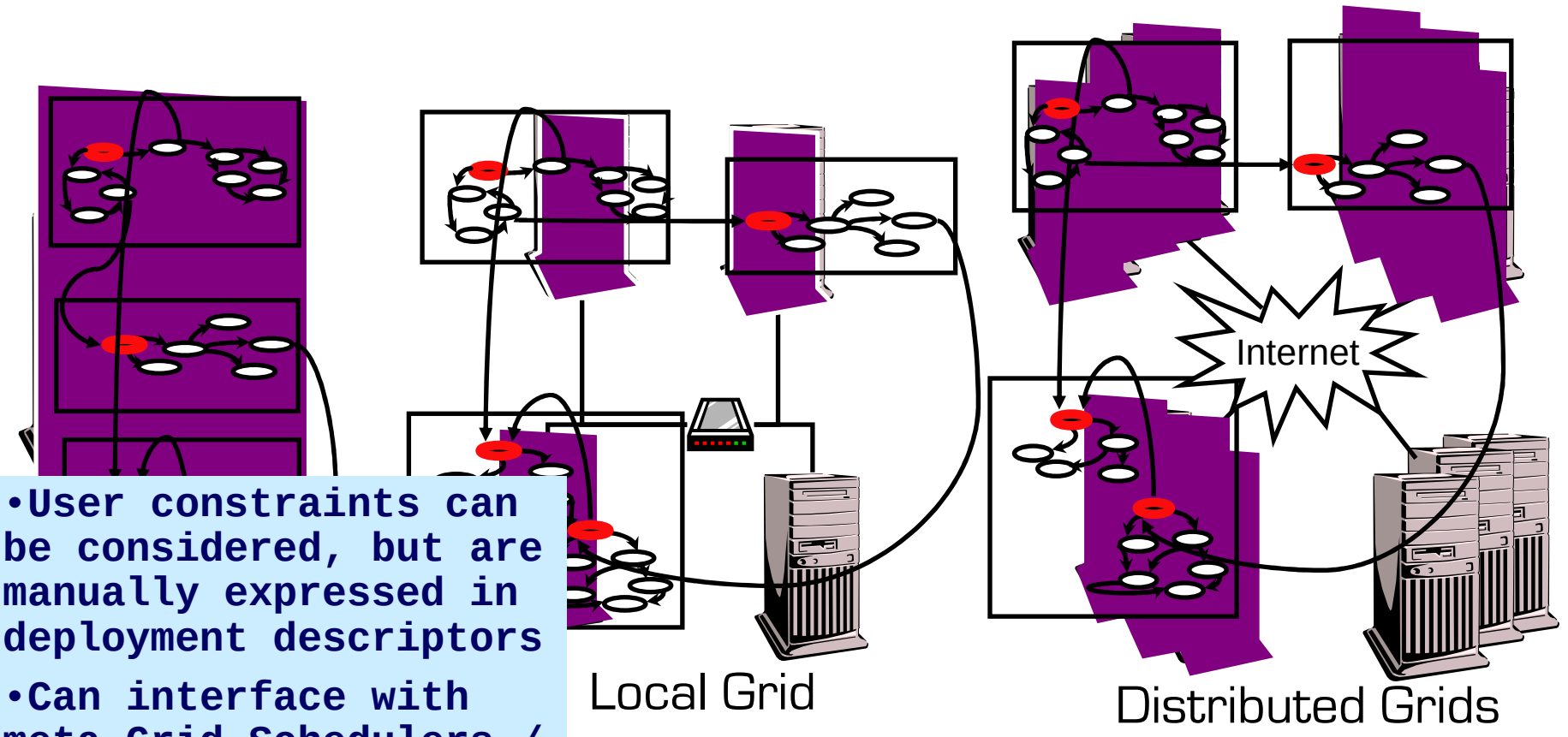
JVM created on the remote Hosts

Mapping Virtual Nodes (8): Mixed Protocol

Infrastructure information

```
<processDefinition id="plugtestProcess">
    <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
                hostname="sea.inria.fr">
        <processReference refid="clusterProcess"/>
    </sshProcess>
</processDefinition>

<processDefinition id=" clusterProcess ">
  <pbsProcess class="org.objectweb.proactive.core.process.pbs.PBSProcess"

    <processReference refid="localJVMprocess"/>
        ...
  </pbsProcess>
</processDefinition>
<processDefinition id="localJVMprocess">
        <jvmProcess class="org.objectweb.proactive.core.
process.JVMNodeProcess"/>
</processDefinition>
```


Same application, many deployments



- User constraints can be considered, but are manually expressed in deployment descriptors
- Can interface with meta-Grid Schedulers / mappers

4.2 IC2D

Interactive Control & Debug for Distribution

Eclipse

GUI for the GRID

4.2 IC2D

Interactive Control & Debug for Distribution

Features:

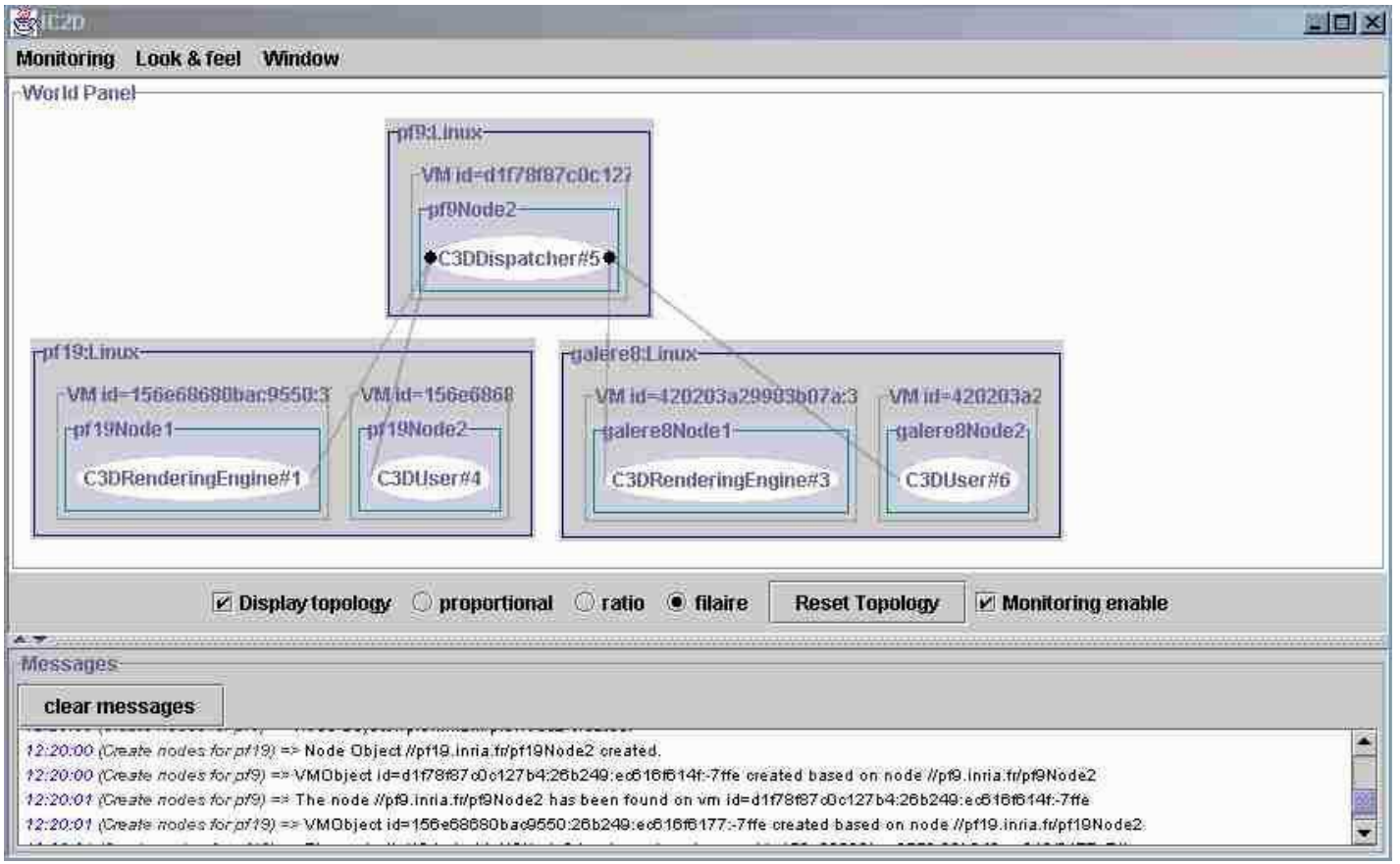
- Graphical visualization
- Textual visualization
- Monitoring and Control

IC2D: Interactive Control and Debugging of Distribution

Key Features:

- Hosts, JVM,
- Nodes
- Active Objects
- Topology
- Migration

- Logical Clock



IC2D: An Overview of its Basic features

Graphical Visualisation:

Hosts, Java Virtual Machines, Nodes, Active Objects
Topology: reference and communications
Status of active objects (executing, waiting, etc.)
Migration of activities

Textual Visualisation:

Ordered list of messages
Status: waiting for a request or for a data
Causal dependencies between messages
Related events (corresponding send and receive, etc.)

Control and Monitoring:

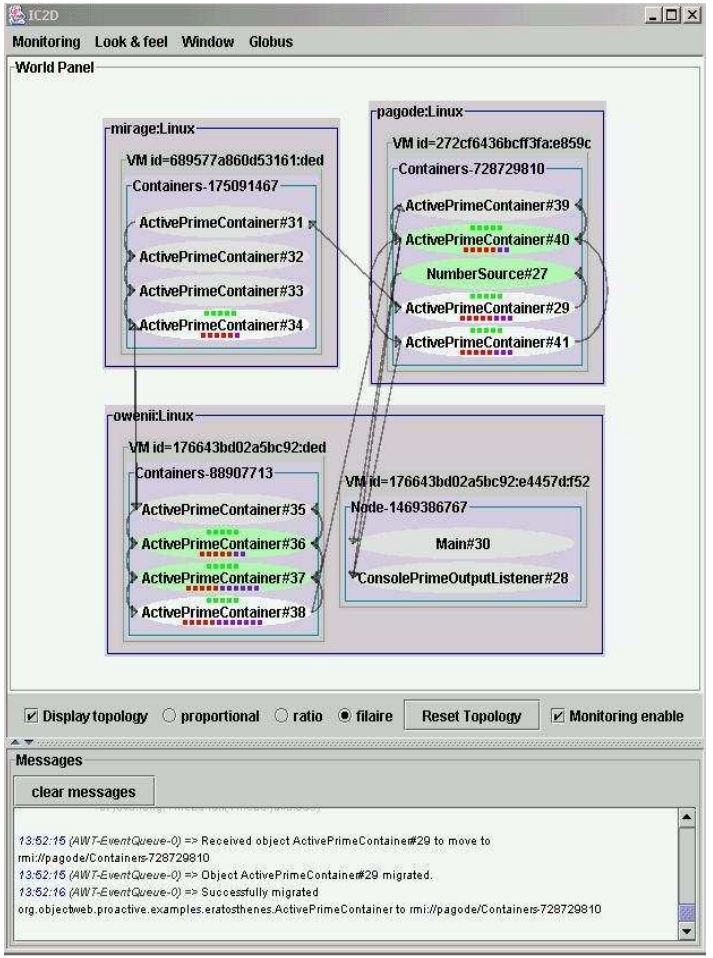
Drag and Drop migration of executing tasks
Creation of additional JVMs and nodes

Job Management:

JVM, AO per Job ID
Textual visualisation, control (kill all, etc.)

Graphical Visualization:

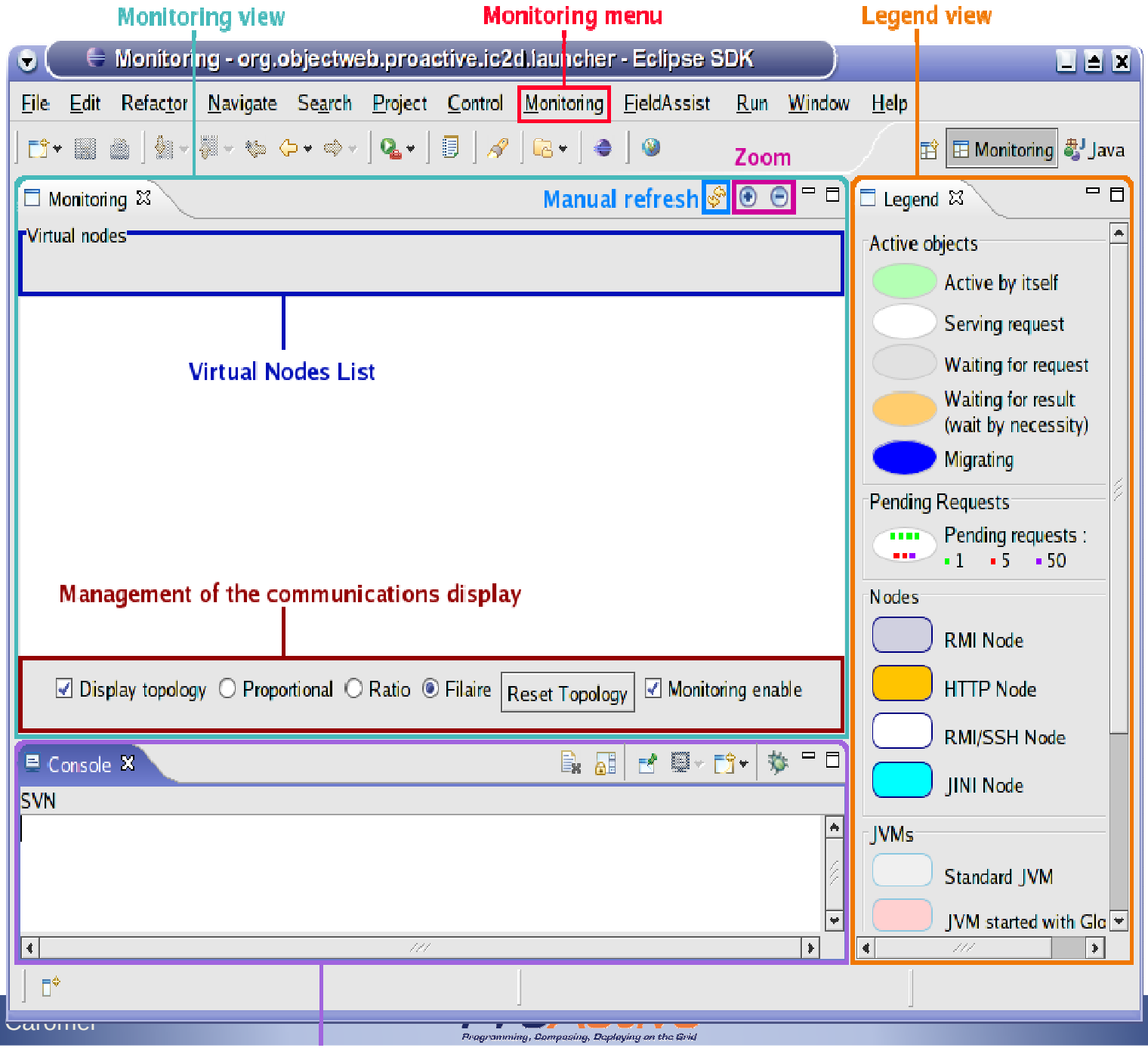
Hosts, Java Virtual Machines, Nodes, Active Objects
Topology: reference and communications
Status of active objects (executing, waiting, etc.)
Migration of activities

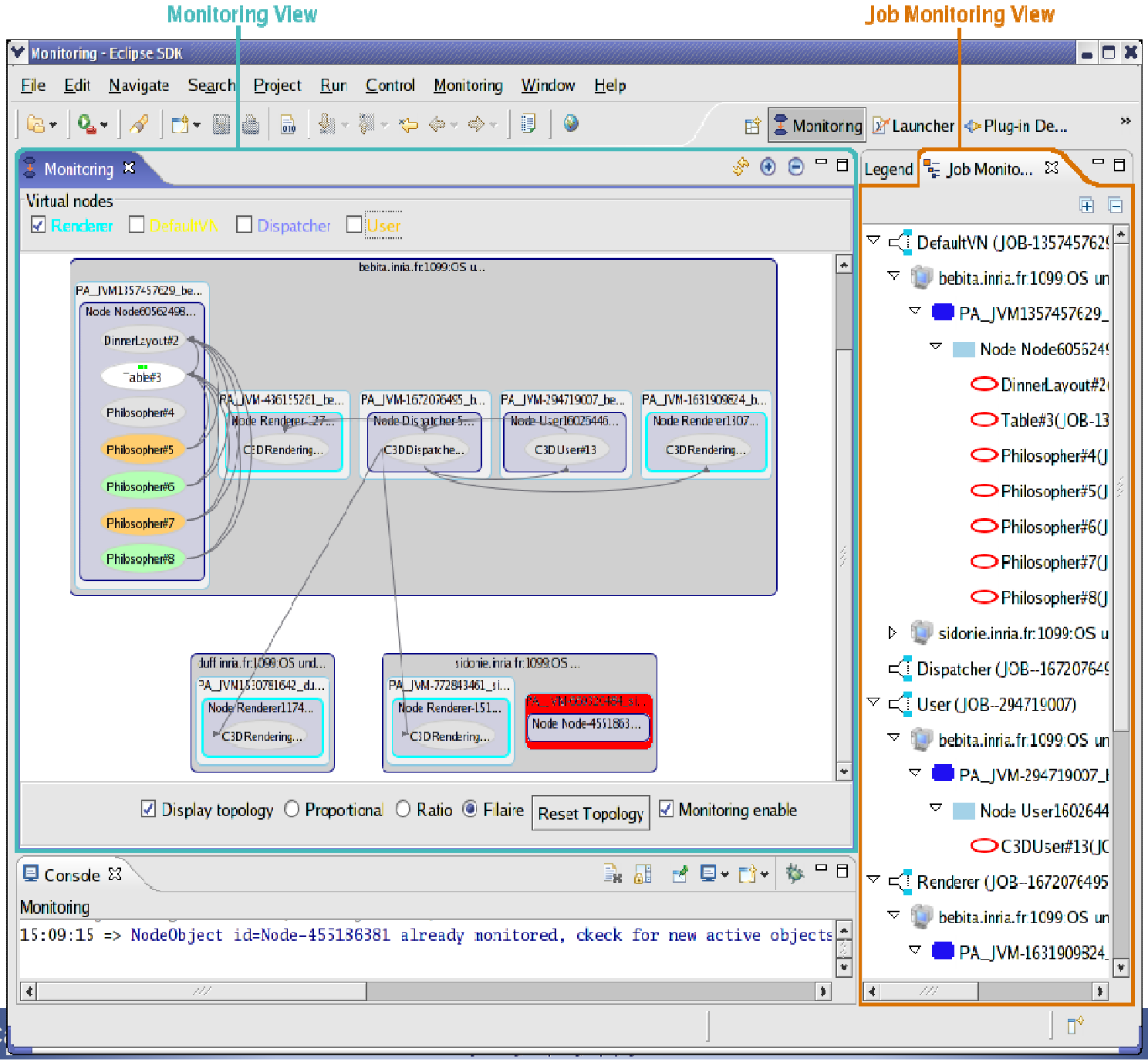


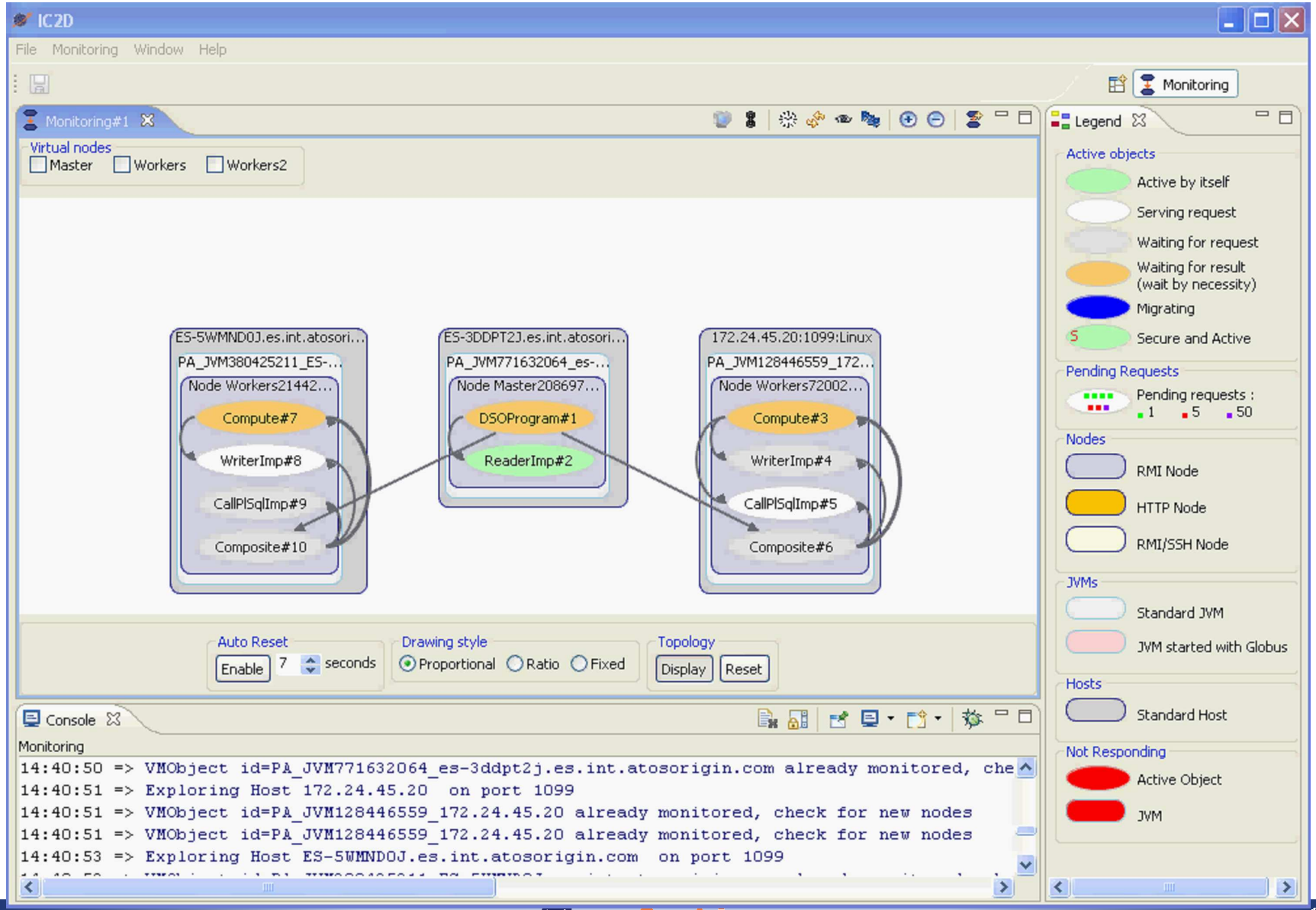
ProActive

Eclipse UI

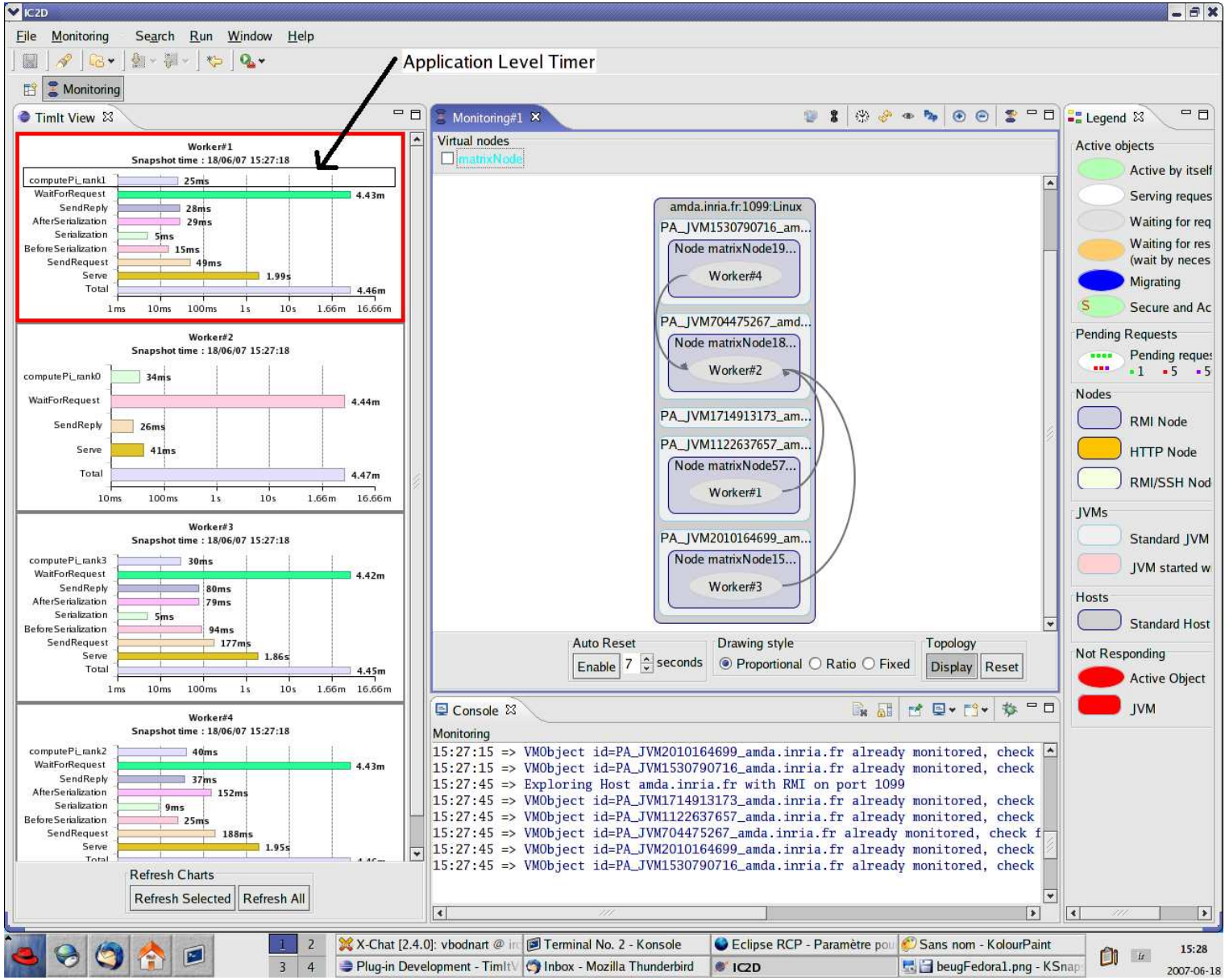
Integrated Development Environments (IDE)

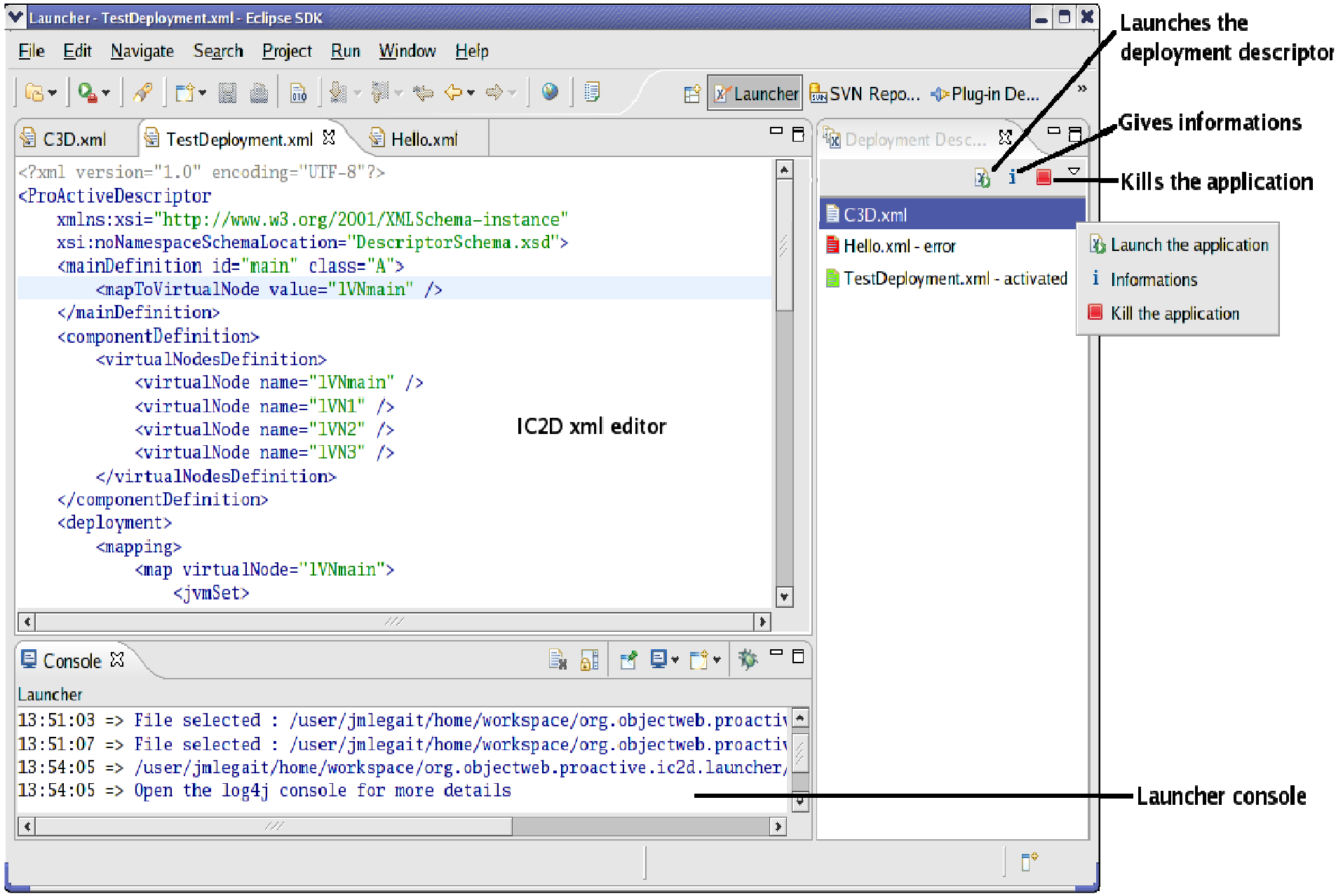

Monitoring view
Monitoring menu
Legend view
Monitoring - org.objectweb.proactive.ic2d.launcher - Eclipse SDK
File Edit Refactor Navigate Search Project Control Monitoring FieldAssist Run Window Help
Zoom
Monitoring
Java
Monitoring
Manual refresh
Virtual nodes
Virtual Nodes List
Management of the communications display
Display topology
Proportional
Ratio
Filaire
Reset Topology
Monitoring enable
Console
SVN
Legend
Active objects
Active by itself
Serving request
Waiting for request
Waiting for result
(wait by necessity)
Migrating
Pending Requests
Pending requests :
1
5
50
Nodes
RMI Node
HTTP Node
RMI/SSH Node
JINI Node
JVMs
Standard JVM
JVM started with Glo


Monitoring View
Job Monitoring View
Monitoring - Eclipse SDK
File Edit Navigate Search Project Run Control Monitoring Window Help
Monitoring
Launcher
Plug-in De...
Monitoring
Virtual nodes
Renderer
DefaultVN
Dispatcher
User
bebita.inria.fr:1099:OS u...
PA_JVM1357457629_be...
Node Node60562498...
DinnerLayout#2
Table#3
Philosopher#4
Philosopher#5
Philosopher#6
Philosopher#7
Philosopher#8
PA_JVM-436155261_be...
Node Renderer...27...
C3DRendering...
PA_JVM-1672076495_b...
Node Dispatcher 5...
C3DDispatche...
PA_JVM-294719007_be...
Node User160256446...
C3DUser#13
PA_JVM-1631909824_b...
Node Renderer1307...
C3DRendering...
duff.inria.fr:1099:OS und...
PA_JVM1530781642_du...
Node Renderer1174...
C3DRendering...
sidonie.inria.fr:1099:OS ...
PA_JVM-772843461_si...
Node Renderer-151...
C3DRendering...
Node Node-4551863...
Display topology
Propotional
Ratio
Flaire
Reset Topology
Monitoring enable
Console
Monitoring
15:09:15 => NodeObject id=Node-455136381 already monitored, ckeck for new active objects
Legend
Job Monito...
DefaultVN (JOB-135745762
bebita.inria.fr:1099:OS un
PA_JVM1357457629_
Node Node6055249
DinnerLayout#2(
Table#3(JOB-13
Philosopher#4(J
Philosopher#5(J
Philosopher#6(J
Philosopher#7(J
Philosopher#8(J
sidorie.inria.fr:1099:OS u
Dispatcher (JOB--16720764
User (JOB--294719007)
bebita.inria.fr:1099:OS un
PA_JVM-294719007_I
Node User1602644
C3DUser#13(JO
Renderer (JOB--1672076495
bebita.inria.fr:1099:OS un
PA_JVM-1631909824_

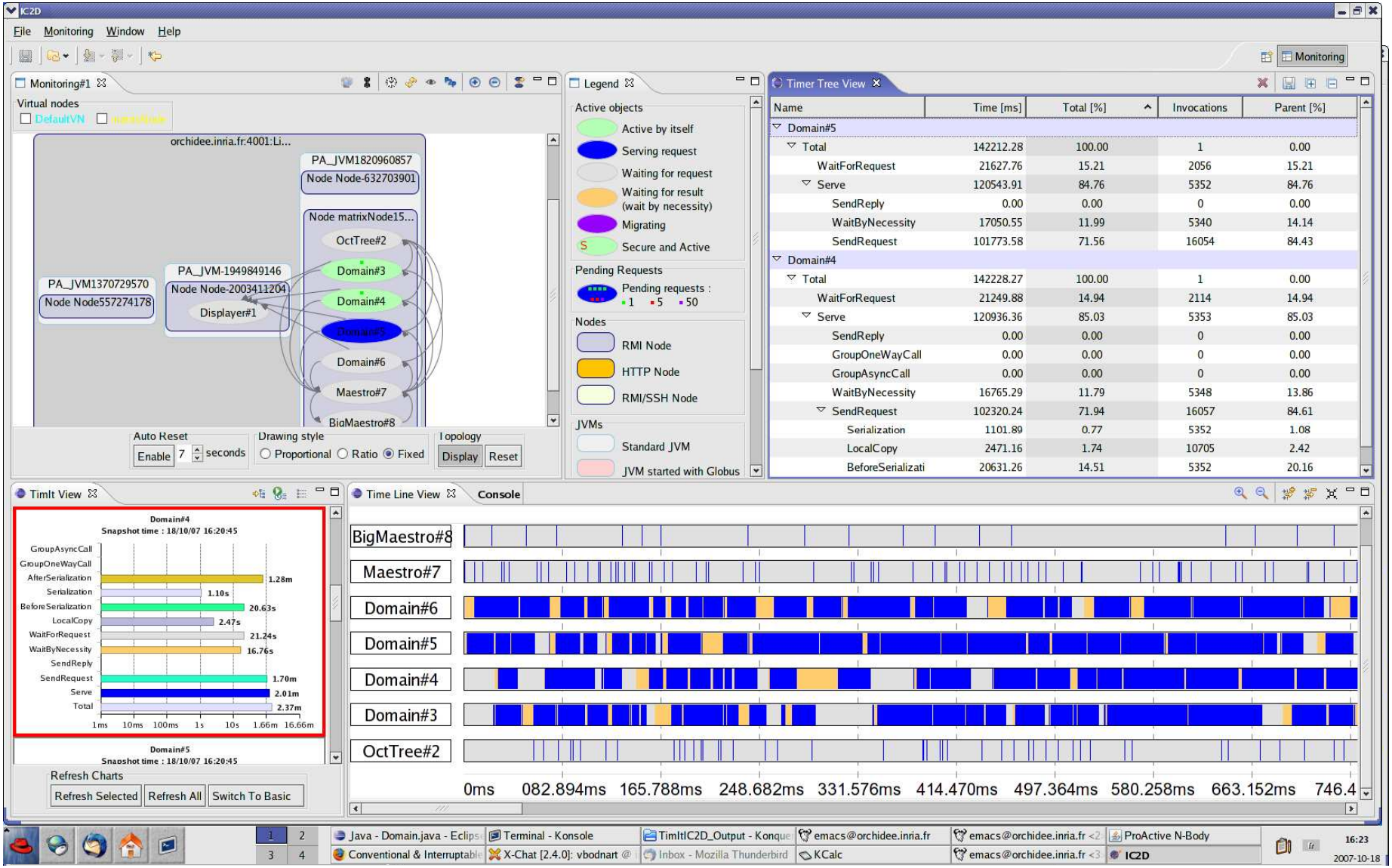

IC2D
File Monitoring Window Help
Monitoring
Monitoring#1
Virtual nodes
Master
Workers
Workers2
ES-5WMNDOJ.es.int.atosori...
PA_JVM380425211_ES-...
Node Workers21442...
Compute#7
WriterImp#8
CallPlSqlImp#9
Composite#10
ES-3DDPT2J.es.int.atosori...
PA_JVM771632064_es-...
Node Master208697...
DSOProgram#1
ReaderImp#2
172.24.45.20:1099:Linux
PA_JVM128446559_172...
Node Workers72002...
Compute#3
WriterImp#4
CallPlSqlImp#5
Composite#6
Auto Reset
Enable
7
seconds
Drawing style
Proportional
Ratio
Fixed
Topology
Display
Reset
Legend
Active objects
Active by itself
Serving request
Waiting for request
Waiting for result (wait by necessity)
Migrating
Secure and Active
Pending Requests
Pending requests : 1 5 50
Nodes
RMI Node
HTTP Node
RMI/SSH Node
JVMs
Standard JVM
JVM started with Globus
Hosts
Standard Host
Not Responding
Active Object
JVM
Console
Monitoring
14:40:50 => VMObject id=PA_JVM771632064_es-3ddpt2j.es.int.atosorigin.com already monitored, che
14:40:51 => Exploring Host 172.24.45.20 on port 1099
14:40:51 => VMObject id=PA_JVM128446559_172.24.45.20 already monitored, check for new nodes
14:40:51 => VMObject id=PA_JVM128446559_172.24.45.20 already monitored, check for new nodes
14:40:53 => Exploring Host ES-5WMNDOJ.es.int.atosorigin.com on port 1099

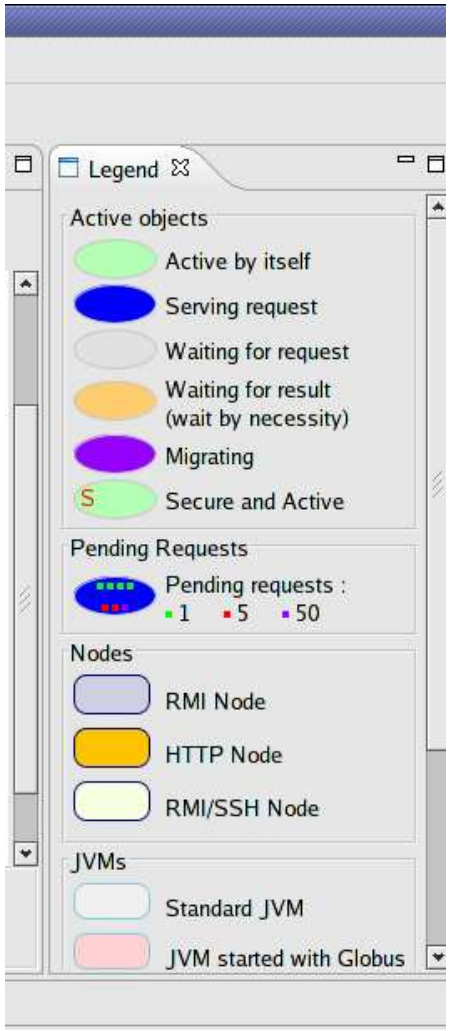
TimIt Automatic Timers in IC2D

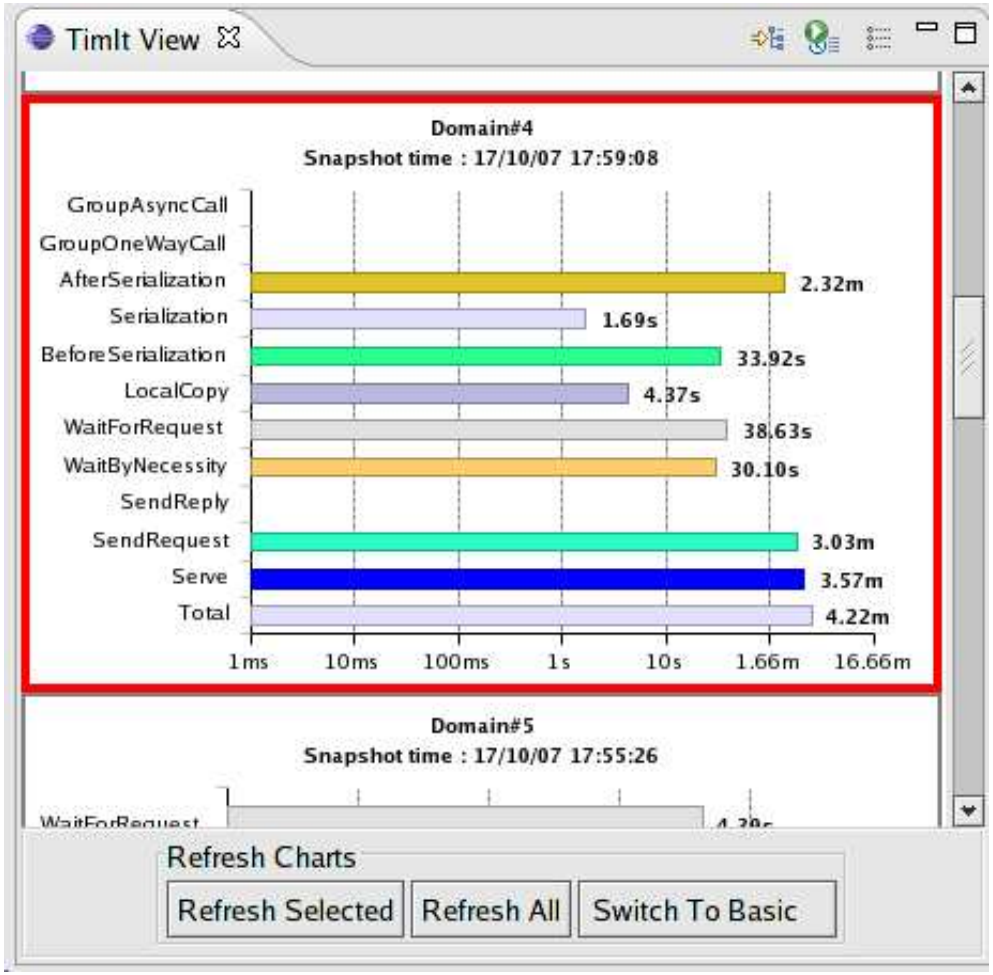


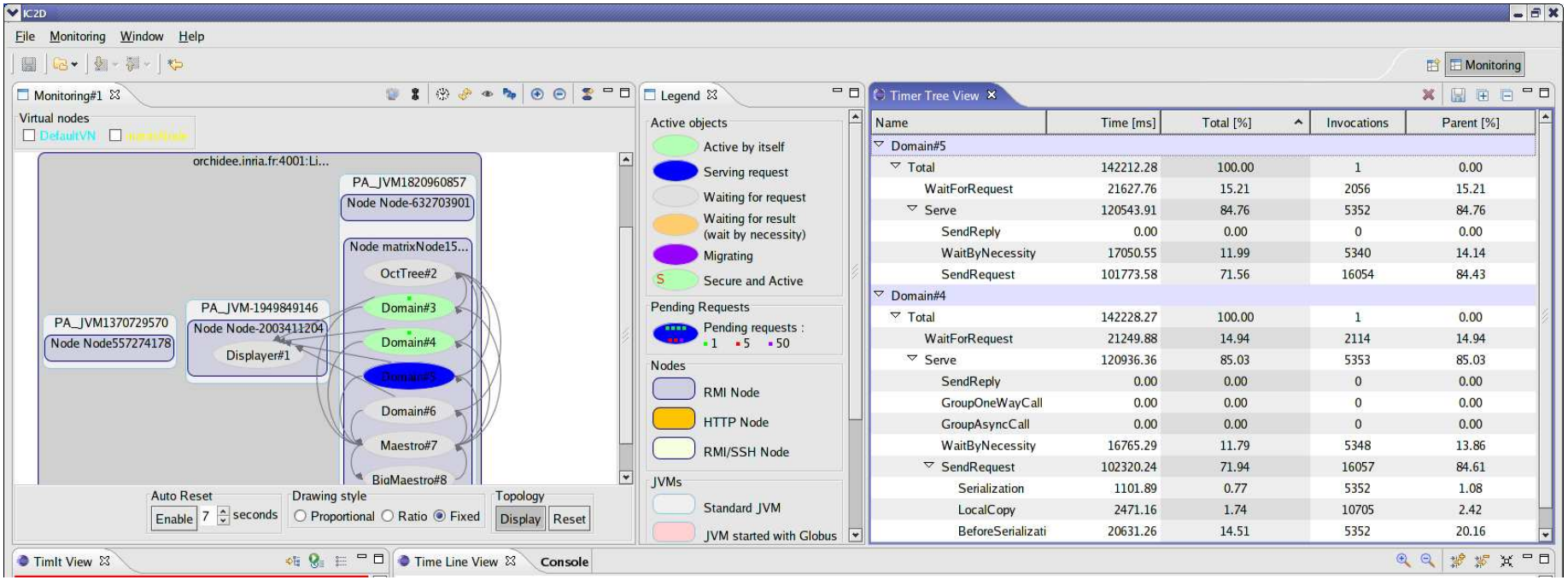

Launcher - TestDeployment.xml - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Launcher
SVN Repo...
Plug-in De...
C3D.xml
TestDeployment.xml
Hello.xml
<?xml version="1.0" encoding="UTF-8"?>
<ProActiveDescriptor
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DescriptorSchema.xsd">
<mainDefinition id="main" class="A">
<mapToVirtualNode value="lVNmain" />
</mainDefinition>
<componentDefinition>
<virtualNodesDefinition>
<virtualNode name="lVNmain" />
<virtualNode name="lVN1" />
<virtualNode name="lVN2" />
<virtualNode name="lVN3" />
</virtualNodesDefinition>
</componentDefinition>
<deployment>
<mapping>
<map virtualNode="lVNmain">
<jvmSet>
IC2D xml editor
Deployment Desc...
C3D.xml
Hello.xml - error
TestDeployment.xml - activated
Launch the application
Informations
Kill the application
Launches the deployment descriptor
Gives informations
Kills the application
Console
Launcher
13:51:03 => File selected : /user/jmlegait/home/workspace/org.objectweb.proactiv
13:51:07 => File selected : /user/jmlegait/home/workspace/org.objectweb.proactiv
13:54:05 => /user/jmlegait/home/workspace/org.objectweb.proactive.ic2d.launcher/
13:54:05 => Open the log4j console for more details
Launcher console

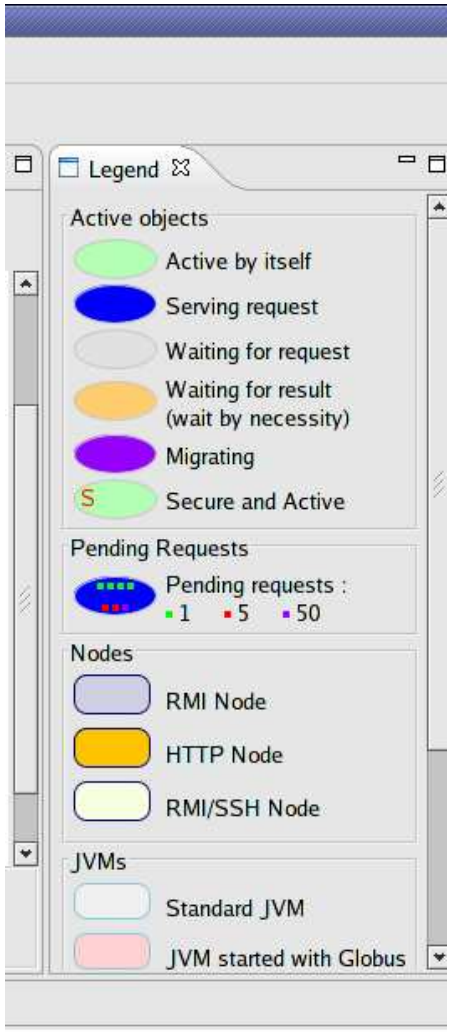
TimIt and TimeLine

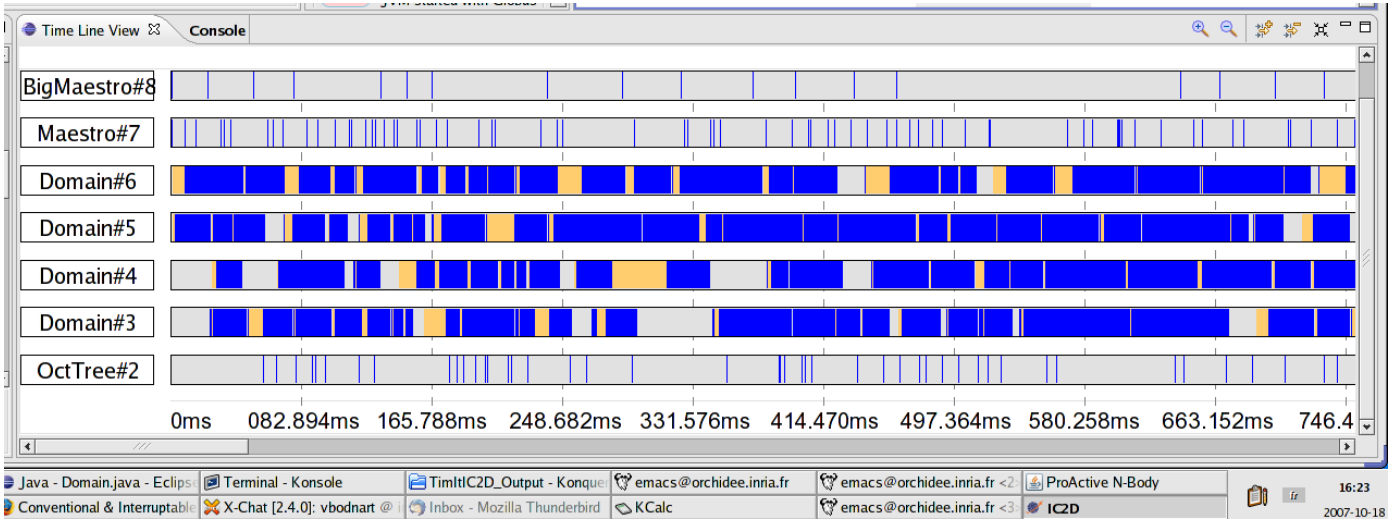



Legend
Active objects
Active by itself
Serving request
Waiting for request
Waiting for result
(wait by necessity)
Migrating
Secure and Active
Pending Requests
Pending requests :
1 5 50
Nodes
RMI Node
HTTP Node
RMI/SSH Node
JVMs
Standard JVM
JVM started with Globus


TimIt View
Domain#4
Snapshot time : 17/10/07 17:59:08
GroupAsyncCall
GroupOneWayCall
AfterSerialization
Serialization
BeforeSerialization
LocalCopy
WaitForRequest
WaitByNecessity
SendReply
SendRequest
Serve
Total
2.32m
1.69s
33.92s
4.37s
38.63s
30.10s
3.03m
3.57m
4.22m
1ms
10ms
100ms
1s
10s
1.66m
16.66m
Domain#5
Snapshot time : 17/10/07 17:55:26
Refresh Charts
Refresh Selected
Refresh All
Switch To Basic

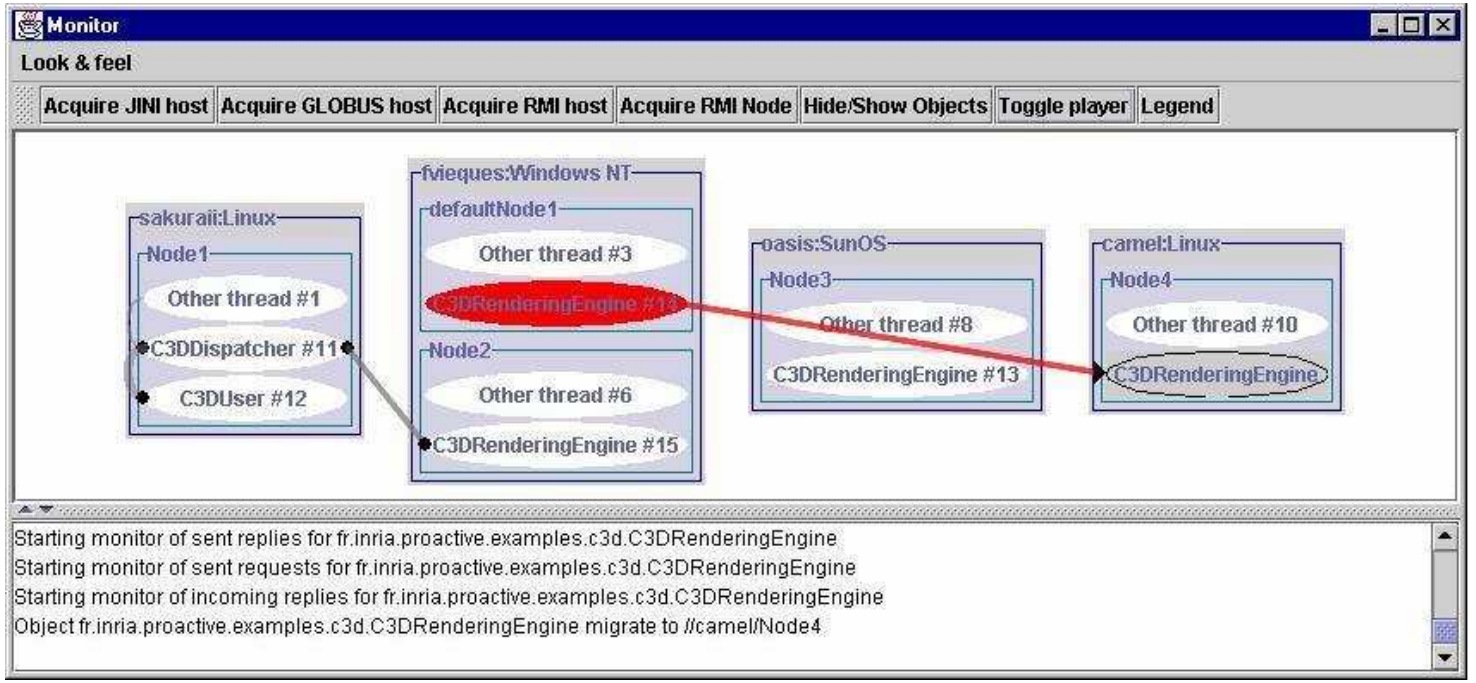

IC2D
File Monitoring Window Help
Monitoring
Monitoring#1
Virtual nodes
DefaultVN
orchidee.inria.fr:4001:Li...
PA_JVM1820960857
Node Node-632703901
Node matrixNode15...
OctTree#2
Domain#3
Domain#4
Domain#5
Domain#6
Maestro#7
BioMaestro#8
PA_JVM-1949849146
Node Node-2003411204
Displayer#1
PA_JVM1370729570
Node Node557274178
Auto Reset
Enable 7 seconds
Drawing style
Proportional Ratio Fixed
Topology
Display Reset
Legend
Active objects
Active by itself
Serving request
Waiting for request
Waiting for result (wait by necessity)
Migrating
Secure and Active
Pending Requests
Pending requests : 1 5 50
Nodes
RMI Node
HTTP Node
RMI/SSH Node
JVMs
Standard JVM
JVM started with Globus
Timer Tree View
Name | Time [ms] | Total [%] | Invocations | Parent [%]
Domain#5
Total 142212.28 100.00 1 0.00
WaitForRequest 21627.76 15.21 2056 15.21
Serve 120543.91 84.76 5352 84.76
SendReply 0.00 0.00 0 0.00
WaitByNecessity 17050.55 11.99 5340 14.14
SendRequest 101773.58 71.56 16054 84.43
Domain#4
Total 142228.27 100.00 1 0.00
WaitForRequest 21249.88 14.94 2114 14.94
Serve 120936.36 85.03 5353 85.03
SendReply 0.00 0.00 0 0.00
GroupOneWayCall 0.00 0.00 0 0.00
GroupAsyncCall 0.00 0.00 0 0.00
WaitByNecessity 16765.29 11.79 5348 13.86
SendRequest 102320.24 71.94 16057 84.61
Serialization 1101.89 0.77 5352 1.08
LocalCopy 2471.16 1.74 10705 2.42
BeforeSerializati 20631.26 14.51 5352 20.16
TimIt View
Time Line View
Console


Legend
Active objects
Active by itself
Serving request
Waiting for request
Waiting for result
(wait by necessity)
Migrating
Secure and Active
Pending Requests
Pending requests :
1
5
50
Nodes
RMI Node
HTTP Node
RMI/SSH Node
JVMs
Standard JVM
JVM started with Globus


Time Line View
Console
BigMaestro#8
Maestro#7
Domain#6
Domain#5
Domain#4
Domain#3
OctTree#2
0ms
082.894ms
165.788ms
248.682ms
331.576ms
414.470ms
497.364ms
580.258ms
663.152ms
746.4
Java - Domain.java - Eclips
Terminal - Konsole
TimItIC2D_Output - Konque
emacs@orchidee.inria.fr
emacs@orchidee.inria.fr <2
ProActive N-Body
Conventional & Interruptable
X-Chat [2.4.0]: vbodnart @
Inbox - Mozilla Thunderbird
KCalc
emacs@orchidee.inria.fr <3
IC2D
16:23
2007-10-18

IC2D: Dynamic change of Deployment Drag-n-Drop Migration

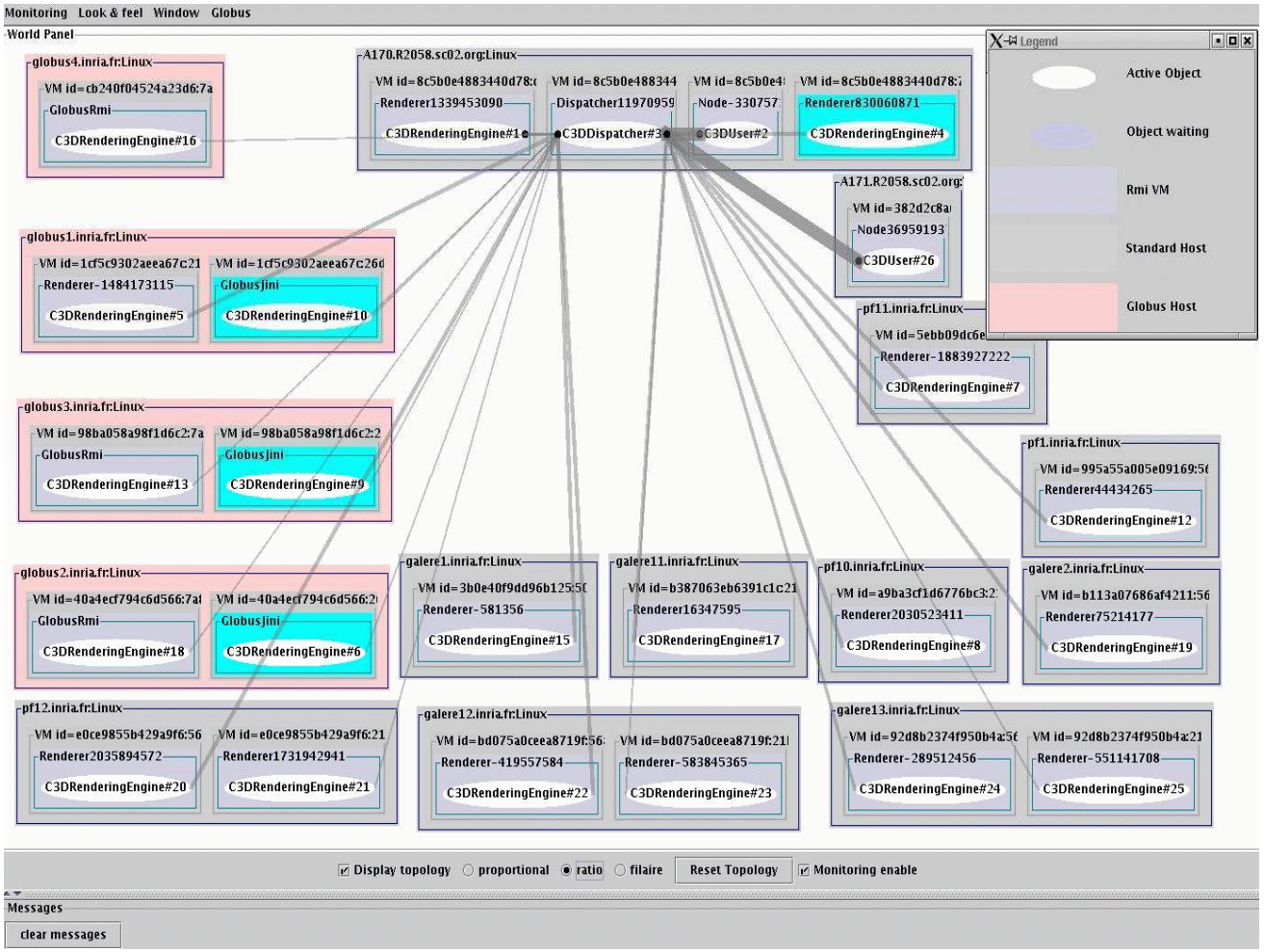
Drag-n-Drop tasks around the world

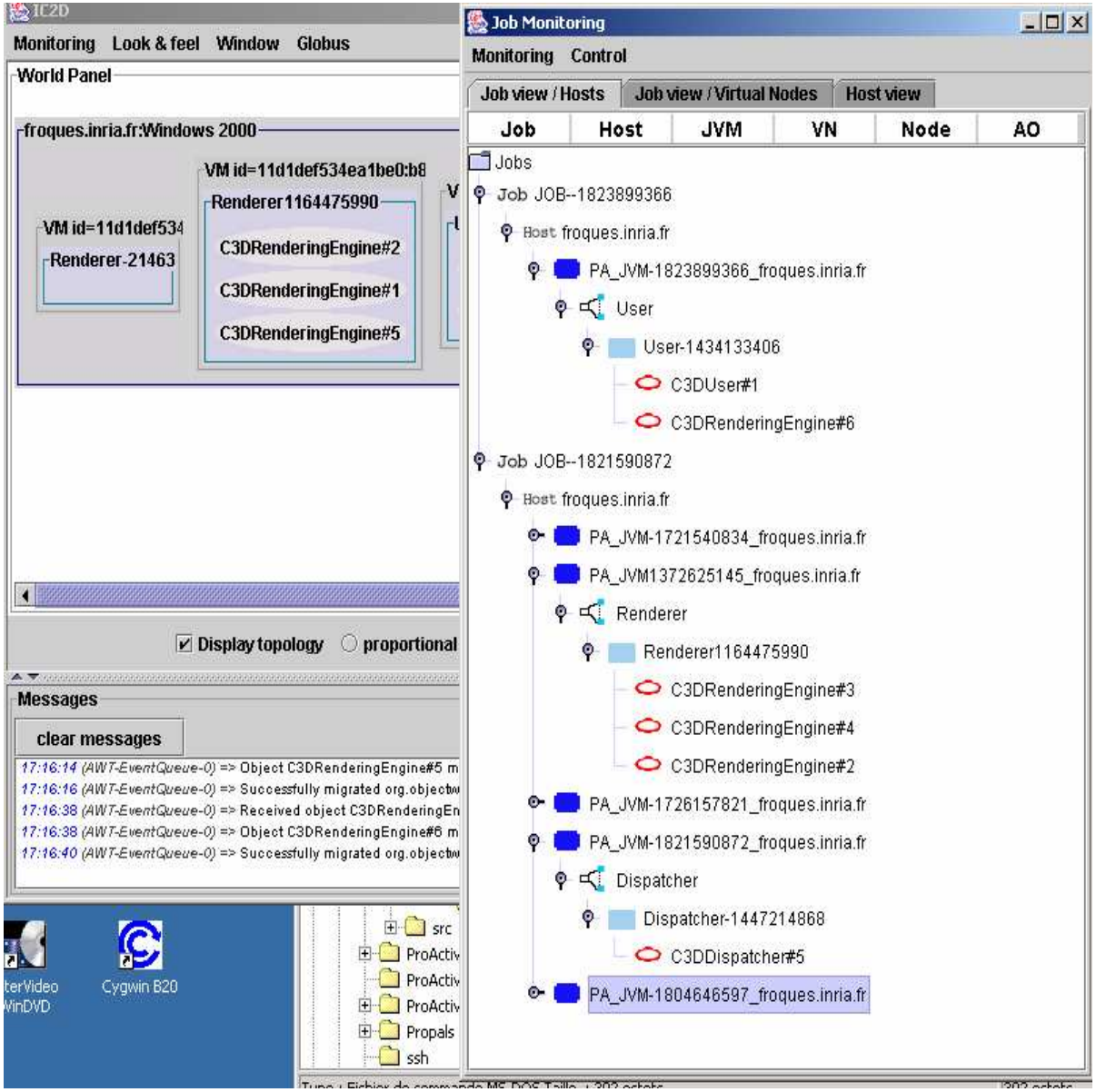


Monitoring of RMI, Globus, Jini, LSF cluster Nice -- Baltimore

ProActive IC2D:

Width of links proportional to the number of communications

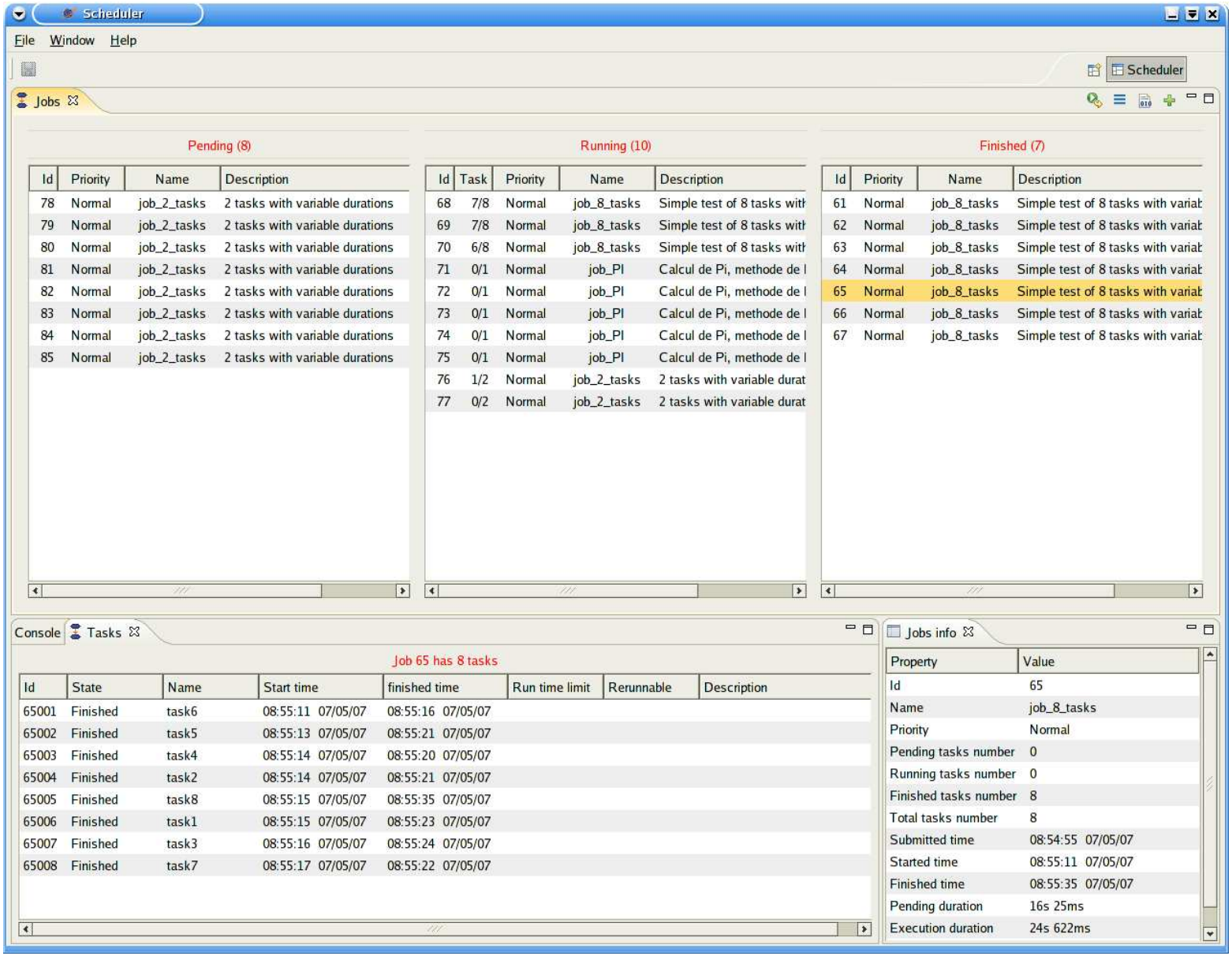




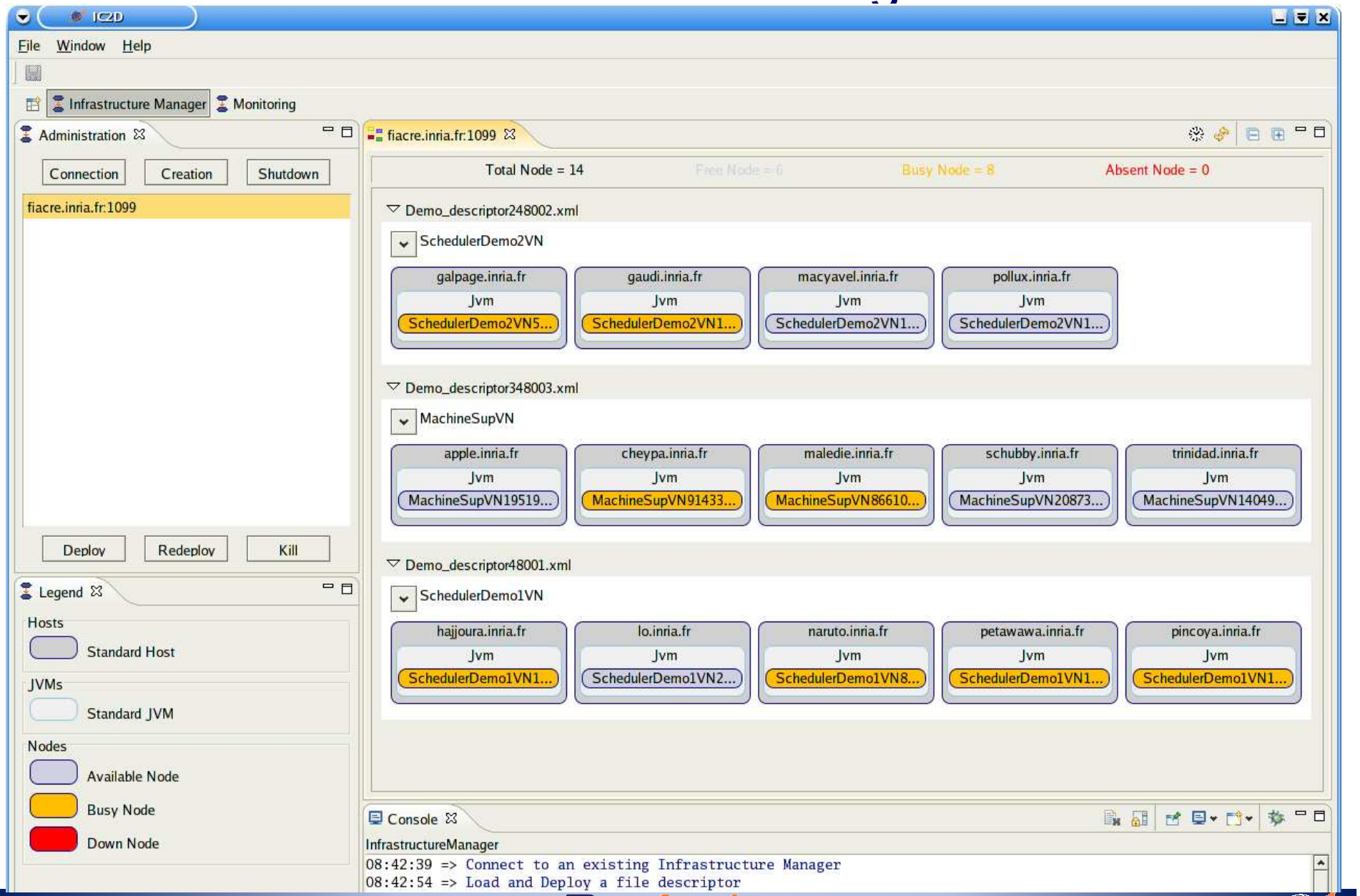
Textual View

Scheduler and Resource Manager: User Interface

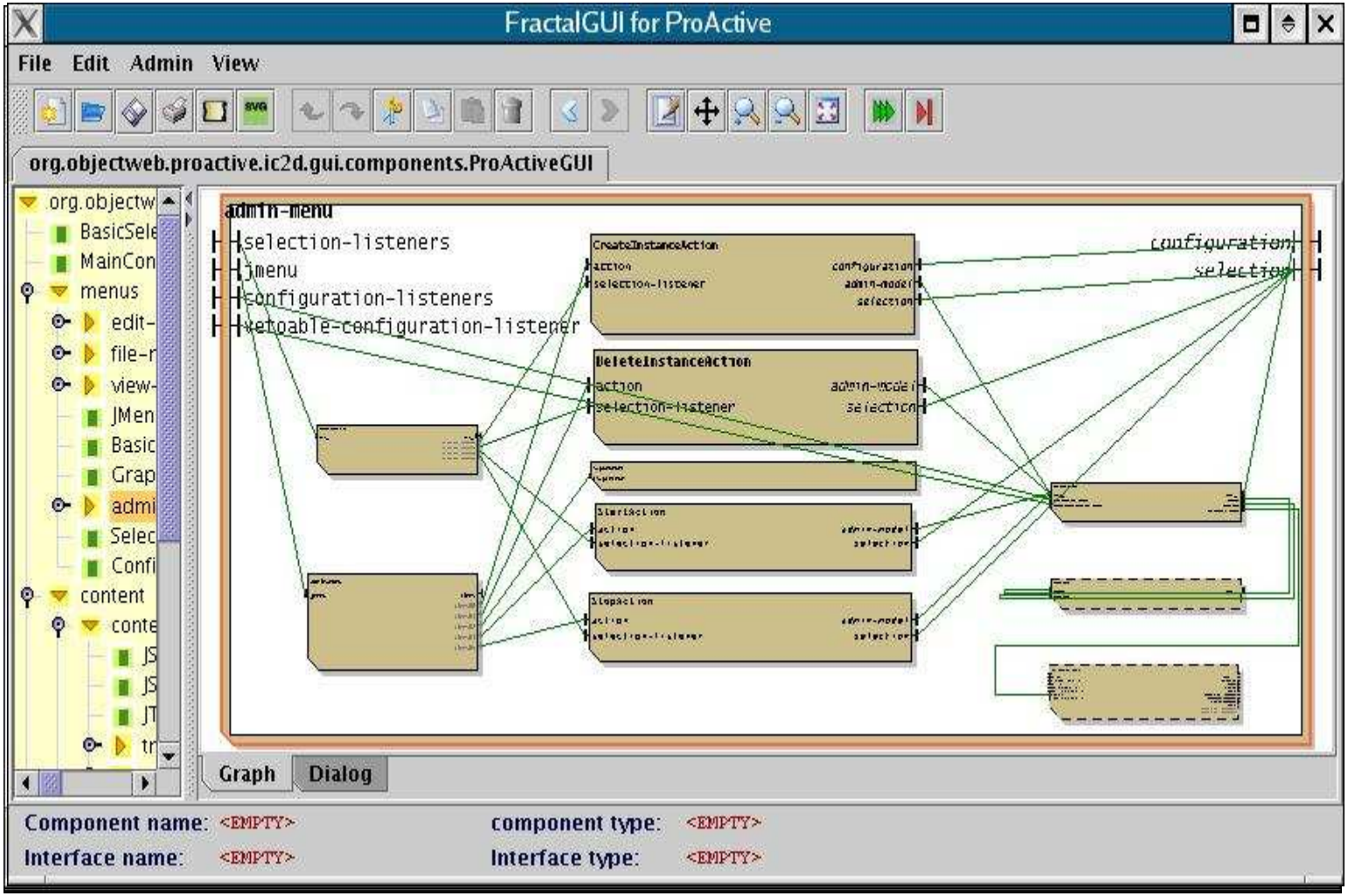
Scheduler: User Interface



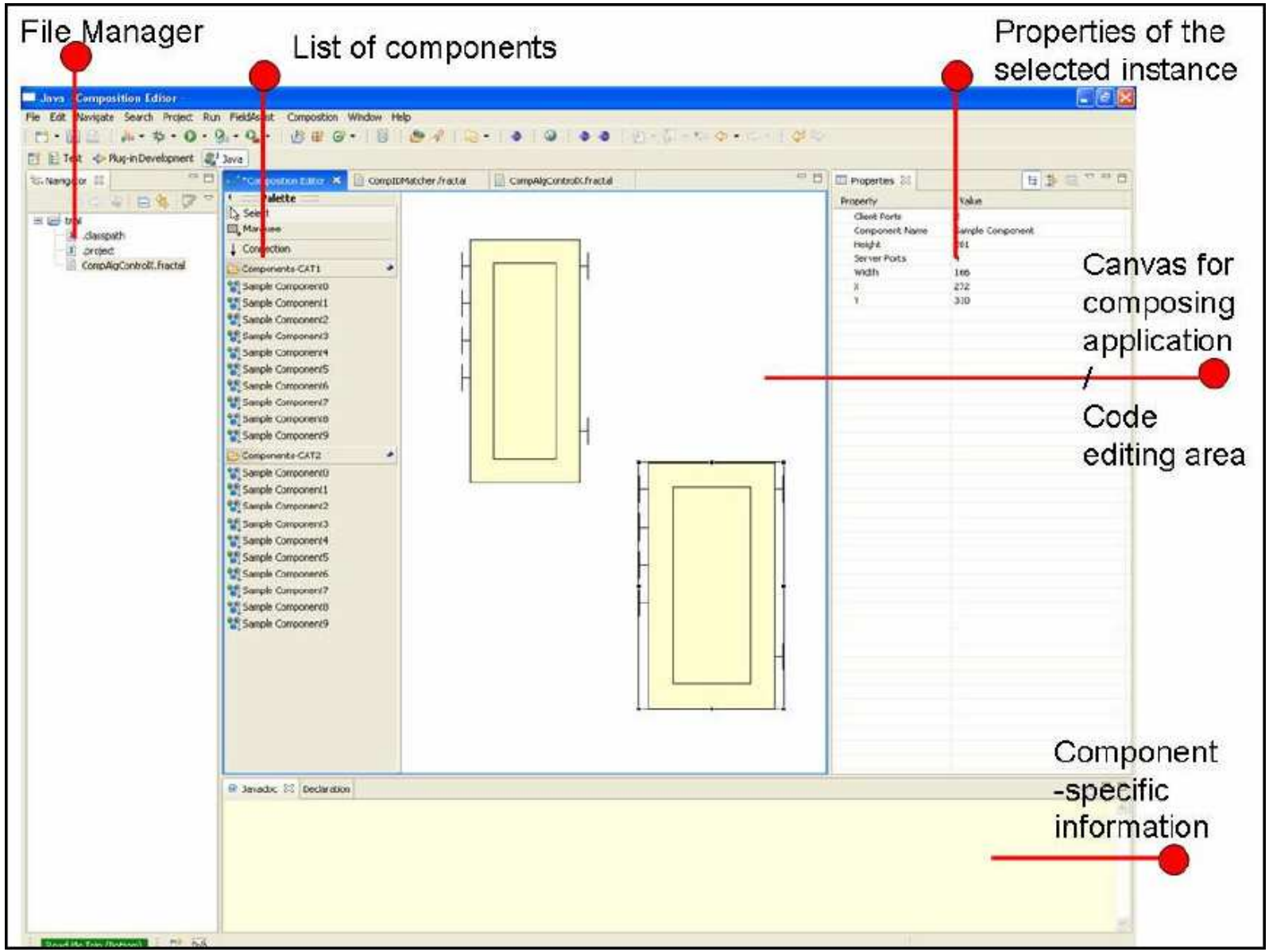
Scheduler: Resource Manager Interface



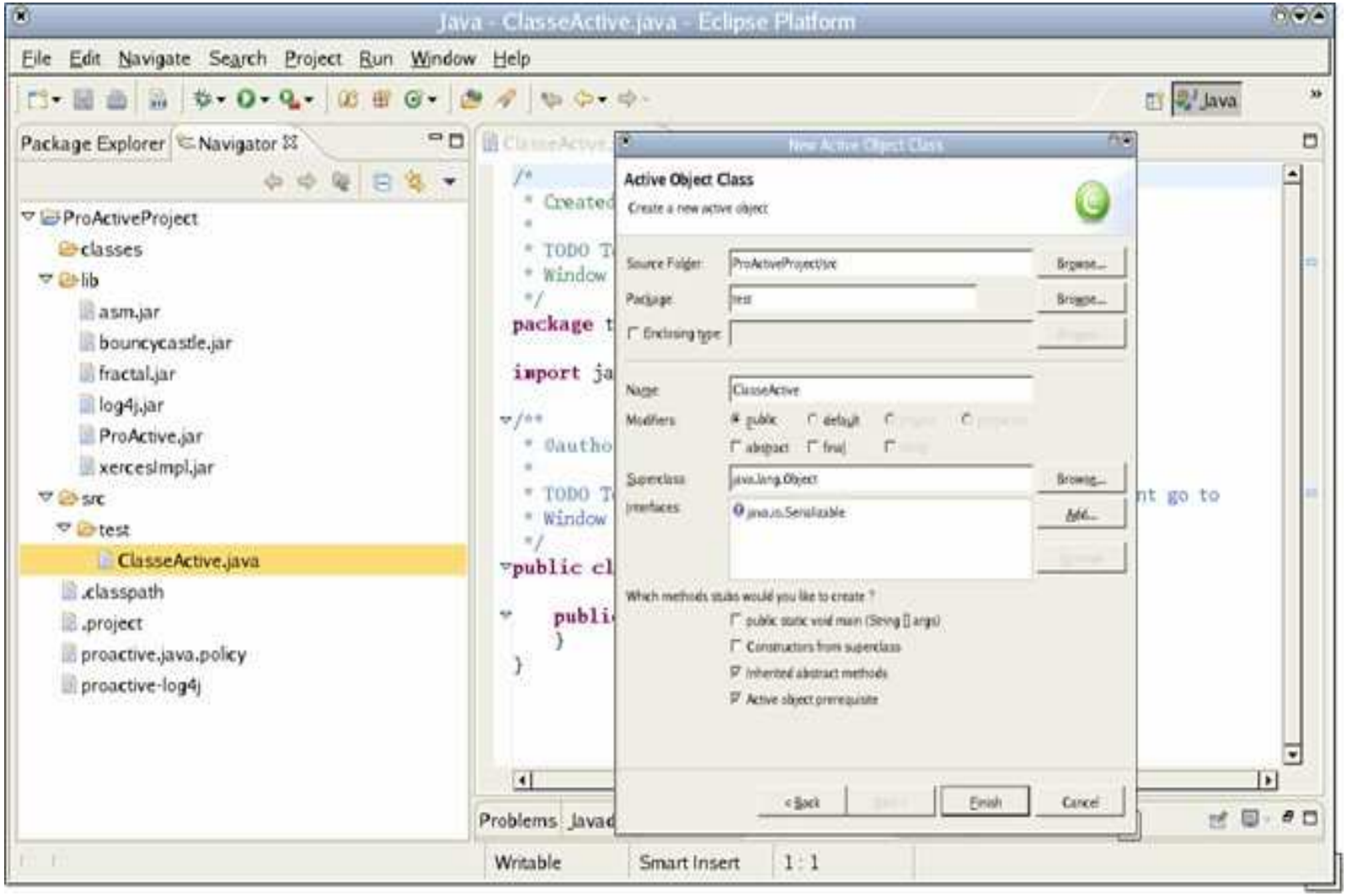
On-going work : GUI for Components

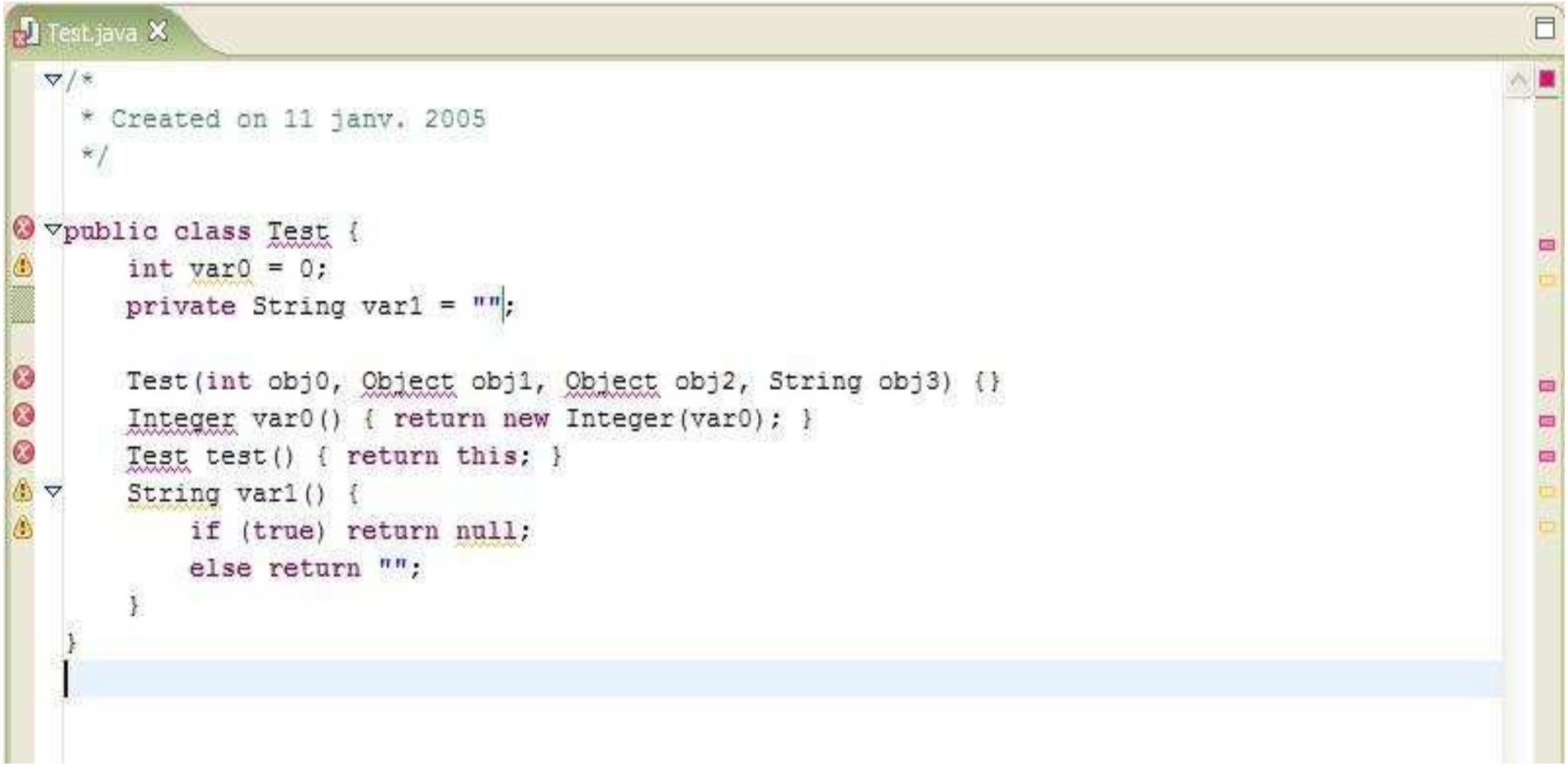


Component GUI under Dev. at Westminster Univ.

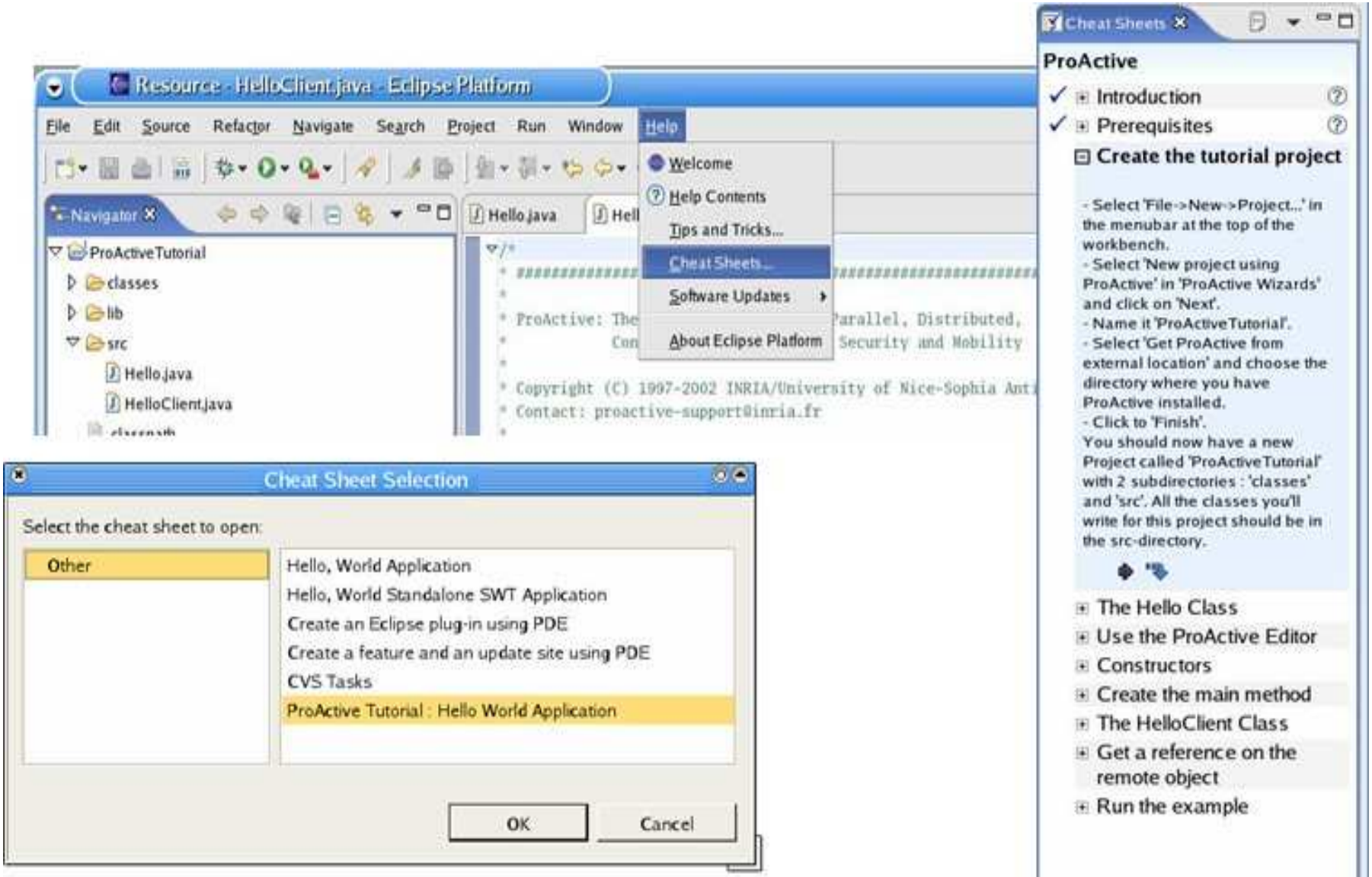


Wizard

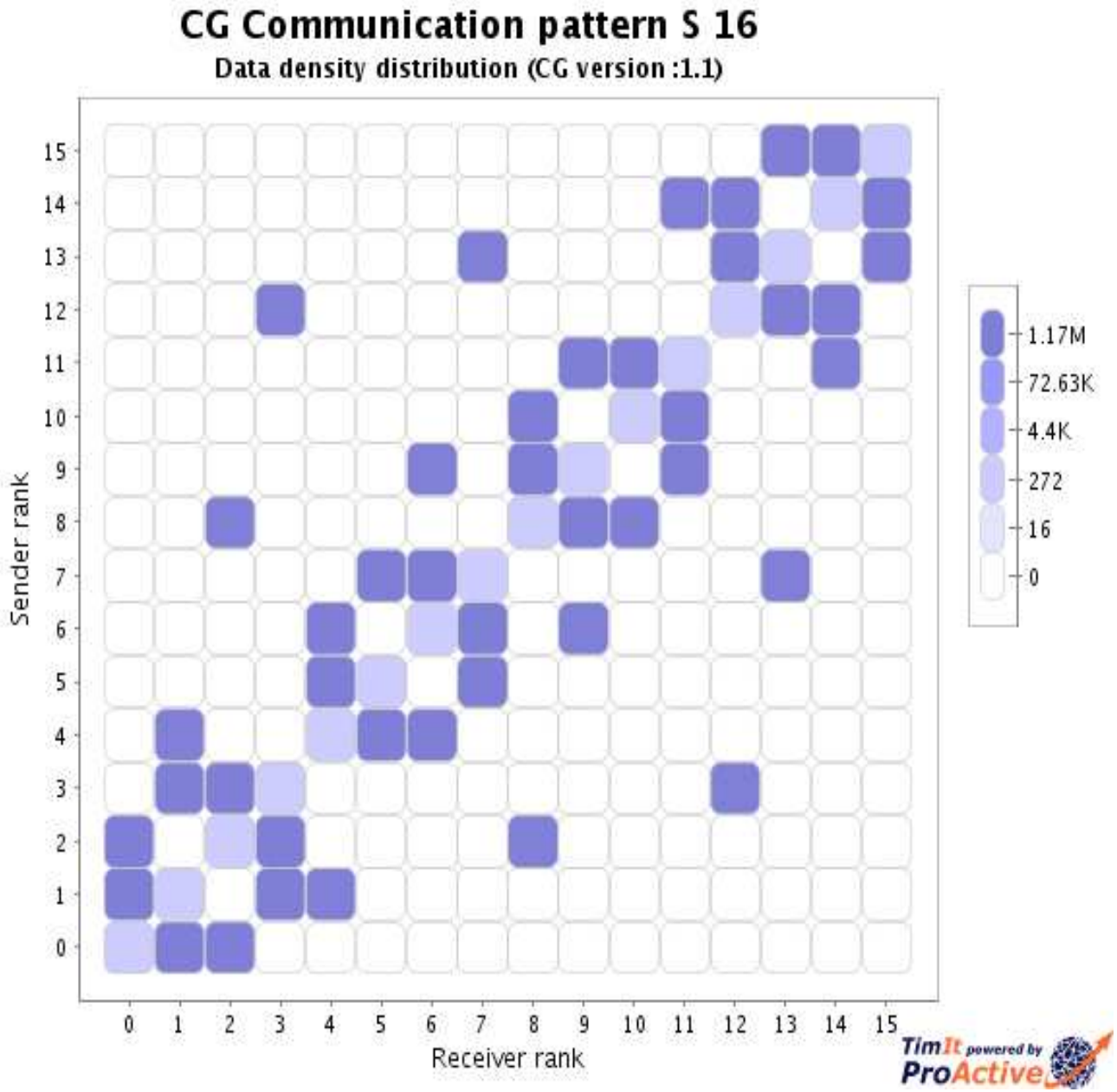

Java - ClasseActive.java - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Package Explorer
Navigator
ProActiveProject
classes
lib
asm.jar
bouncycastle.jar
fractal.jar
log4j.jar
ProActive.jar
xercesImpl.jar
src
test
ClasseActive.java
.classpath
.project
proactive.java.policy
proactive-log4j
Active Object Class
Create a new active object
Source Folder:
Package:
Enclosing type:
Name:
Modifiers:
Superclass:
Interfaces:
java.io.Serializable
Which methods stubs would you like to create ?
Constructors from superclass
Inherited abstract methods
Active object prerequisite
< Back
Finish
Cancel
Problems
Writable
Smart Insert
1 : 1

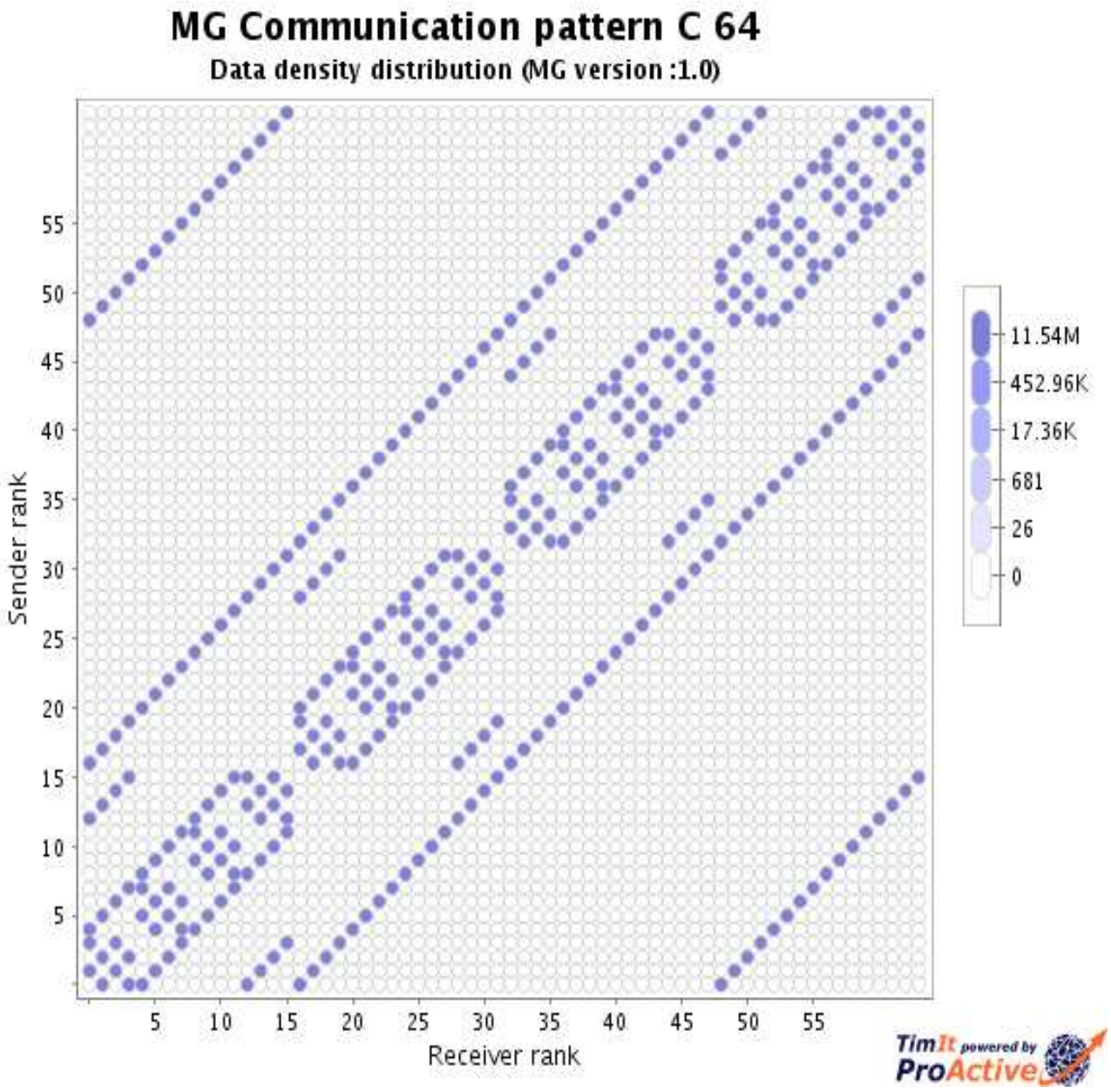


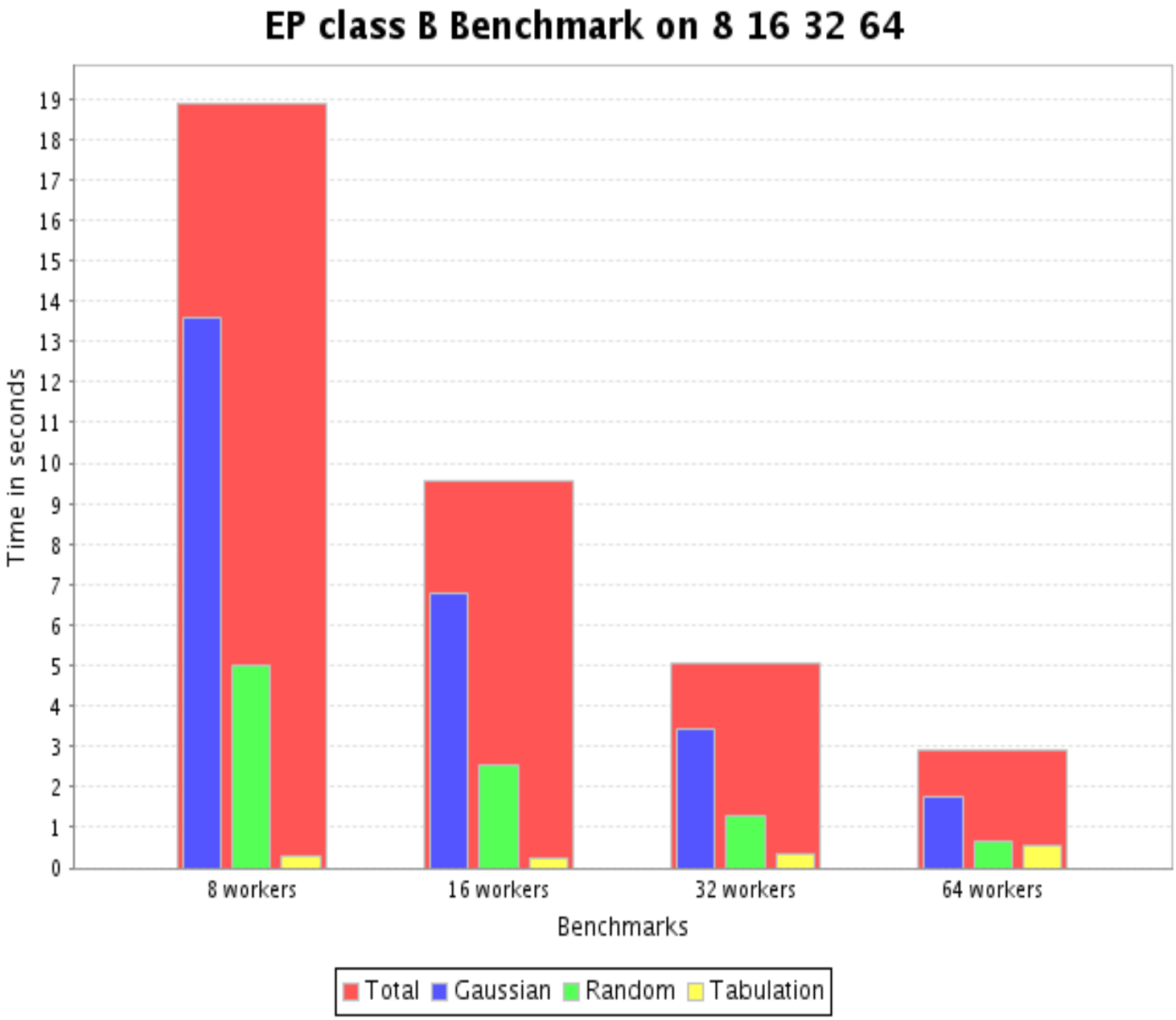
Guided Tour

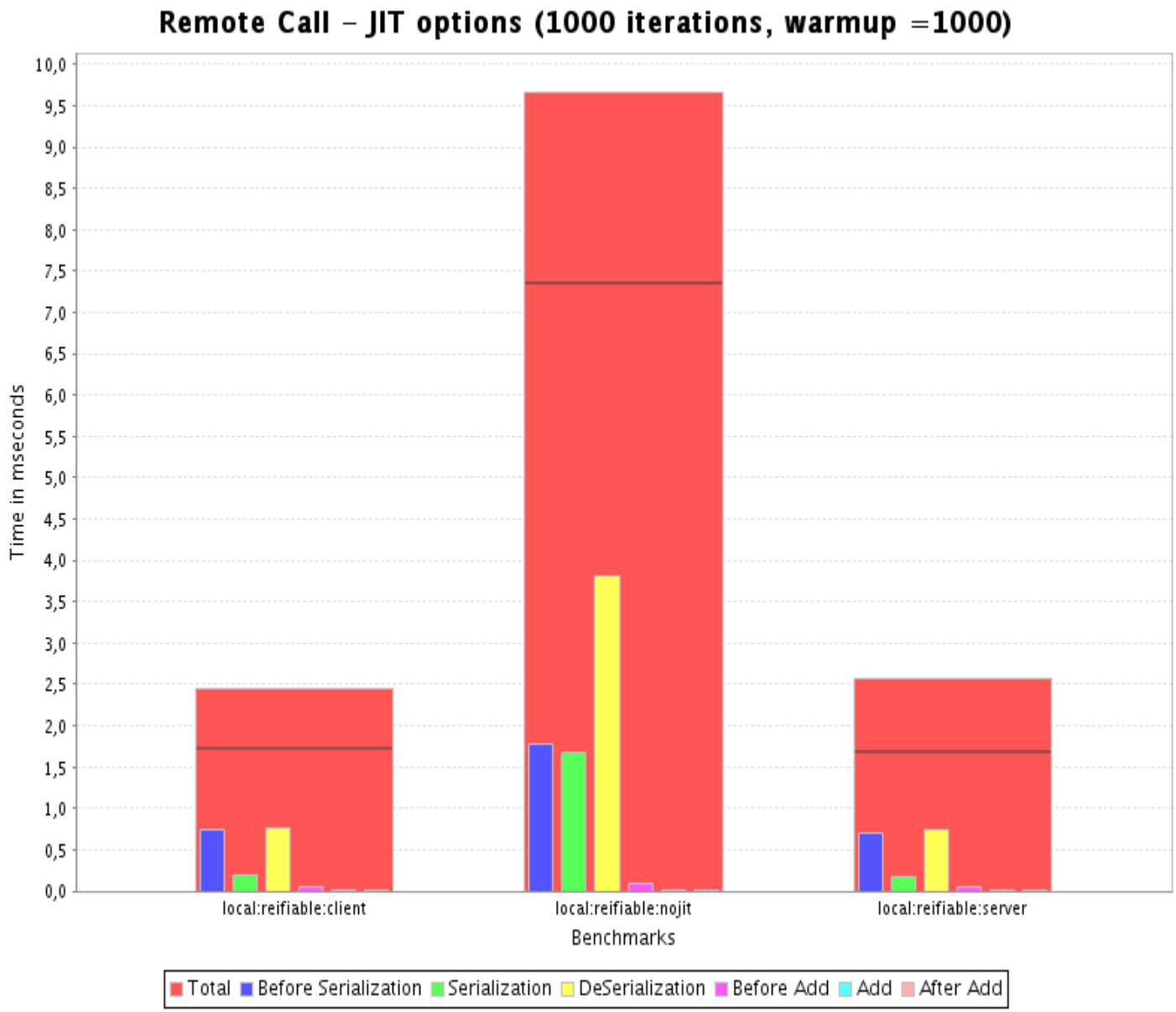

Resource - HelloClient.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Help Contents
Tips and Tricks...
Cheat Sheets...
Software Updates
About Eclipse Platform
Navigator
ProActiveTutorial
classes
lib
src
Hello.java
HelloClient.java
Hello.java
Cheat Sheets
ProActive
Introduction
Prerequisites
Create the tutorial project
- Select 'File->New->Project...' in the menubar at the top of the workbench.
- Select 'New project using ProActive' in 'ProActive Wizards' and click on 'Next'.
- Name it 'ProActiveTutorial'.
- Select 'Get ProActive from external location' and choose the directory where you have ProActive installed.
- Click to 'Finish'.
You should now have a new Project called 'ProActiveTutorial' with 2 subdirectories : 'classes' and 'src'. All the classes you'll write for this project should be in the src-directory.
The Hello Class
Use the ProActive Editor
Constructors
Create the main method
The HelloClient Class
Get a reference on the remote object
Run the example
Cheat Sheet Selection
Select the cheat sheet to open:
Other
Hello, World Application
Hello, World Standalone SWT Application
Create an Eclipse plug-in using PDE
Create a feature and an update site using PDE
CVS Tasks
ProActive Tutorial : Hello World Application
OK
Cancel

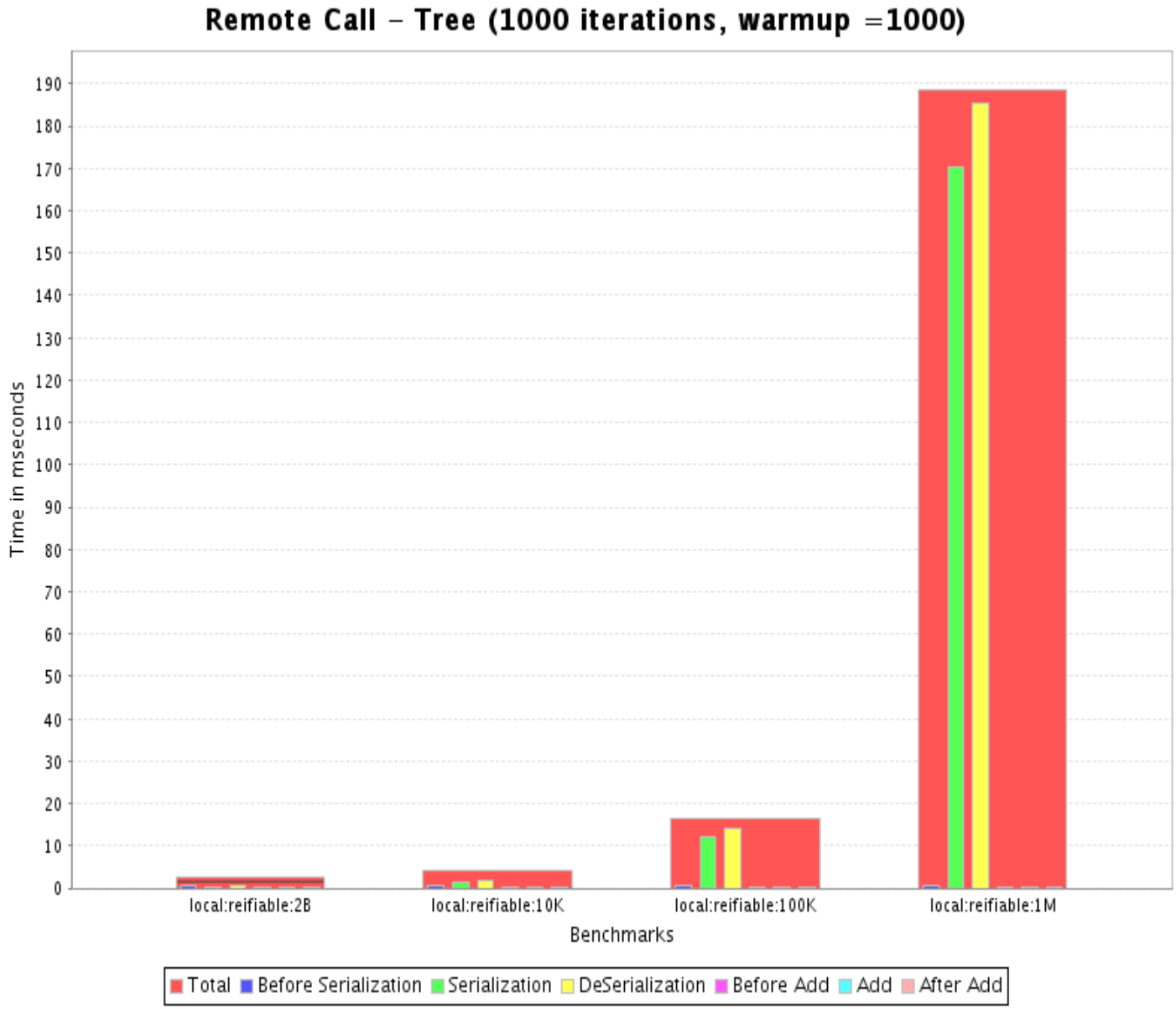
TimIt

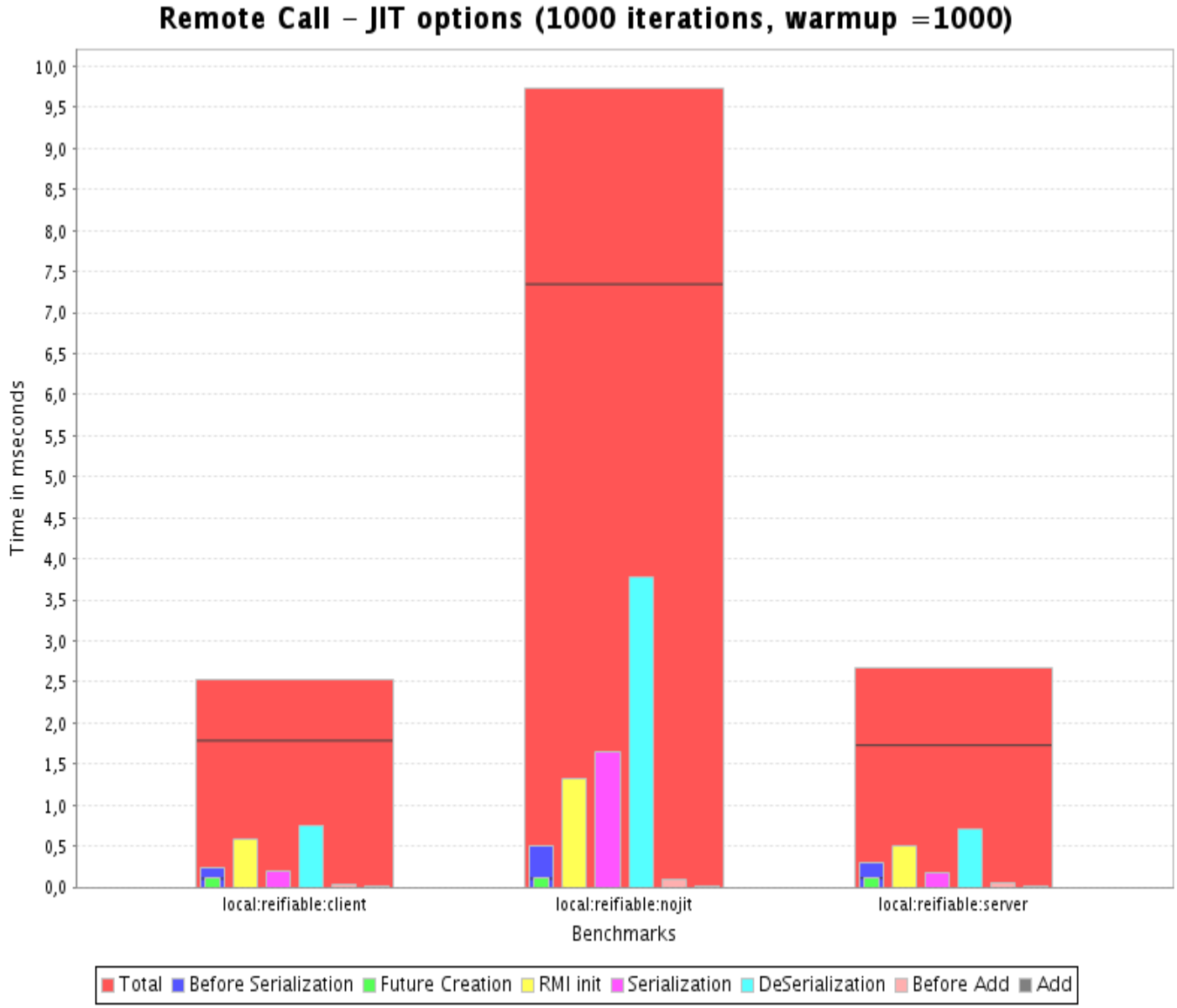

CG Communication pattern S 16
Data density distribution (CG version :1.1)
Sender rank
Receiver rank
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1.17M
72.63K
4.4K
272
16
0
TimIt powered by ProActive


MG Communication pattern C 64
Data density distribution (MG version :1.0)
Sender rank
Receiver rank
5
10
15
20
25
30
35
40
45
50
55
11.54M
452.96K
17.36K
681
26
0
TimIt powered by
ProActive

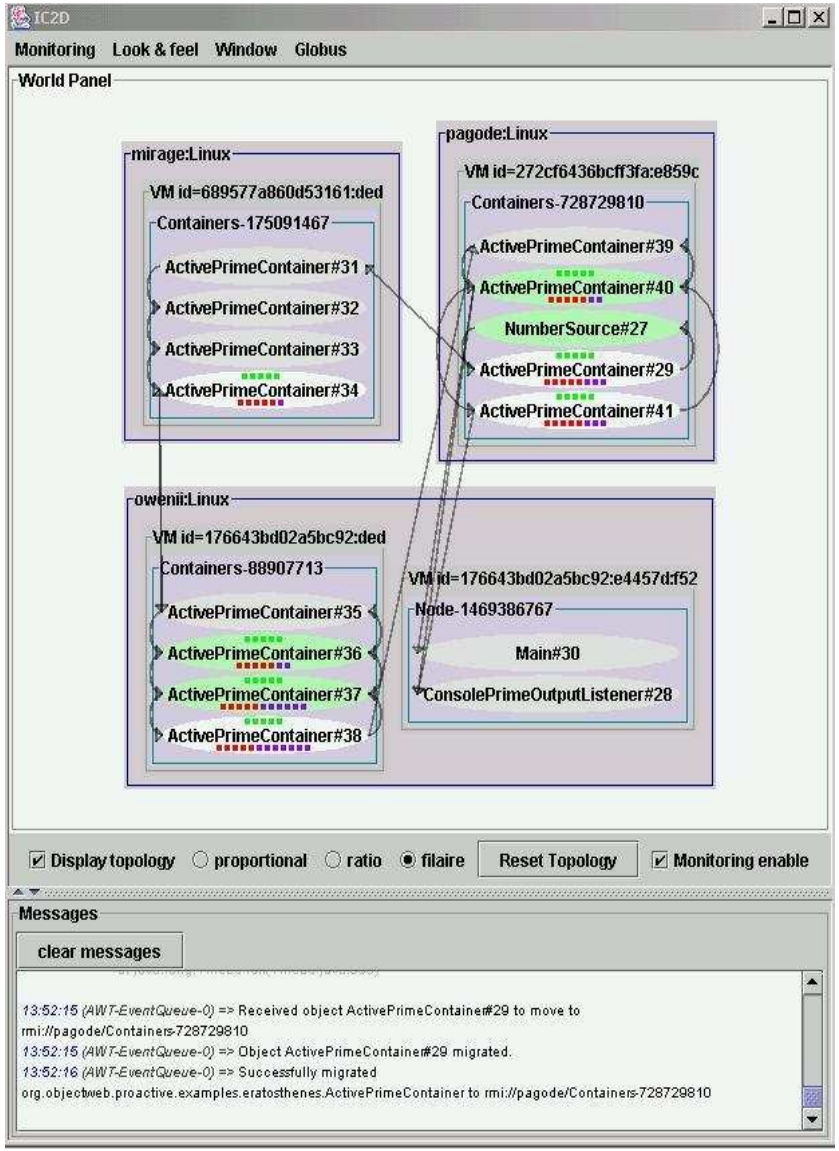

EP class B Benchmark on 8 16 32 64
Time in seconds
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
8 workers
16 workers
32 workers
64 workers
Benchmarks
Total
Gaussian
Random
Tabulation

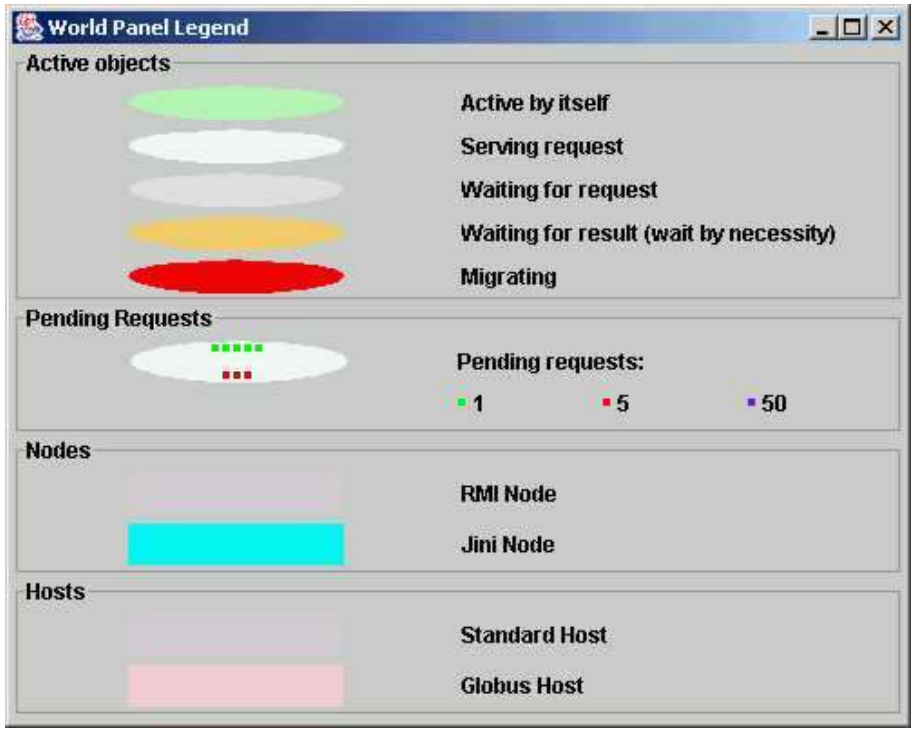

Remote Call – JIT options (1000 iterations, warmup =1000)
Time in mseconds
10,0
9,5
9,0
8,5
8,0
7,5
7,0
6,5
6,0
5,5
5,0
4,5
4,0
3,5
3,0
2,5
2,0
1,5
1,0
0,5
0,0
local:reifiable:client
local:reifiable:nojit
local:reifiable:server
Benchmarks
Total
Before Serialization
Serialization
DeSerialization
Before Add
Add
After Add


Remote Call – Tree (1000 iterations, warmup =1000)
Time in mseconds
190
180
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
local:reifiable:2B
local:reifiable:10K
local:reifiable:100K
local:reifiable:1M
Benchmarks
Total
Before Serialization
Serialization
DeSerialization
Before Add
Add
After Add


Remote Call – JIT options (1000 iterations, warmup =1000)
Time in mseconds
10,0
9,5
9,0
8,5
8,0
7,5
7,0
6,5
6,0
5,5
5,0
4,5
4,0
3,5
3,0
2,5
2,0
1,5
1,0
0,5
0,0
local:reifiable:client
local:reifiable:nojit
local:reifiable:server
Benchmarks
Total
Before Serialization
Future Creation
RMI init
Serialization
DeSerialization
Before Add
Add

IC2D: Interactive Control and Debugging of Distribution





With any ProActive application features:
Graphical and Textual visualization
Monitoring and Control

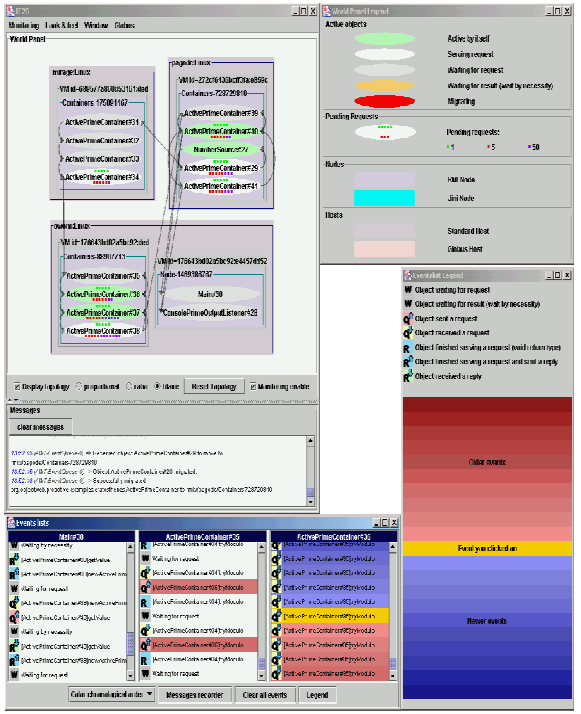
IC2D: Basic features

Graphical Visualization:

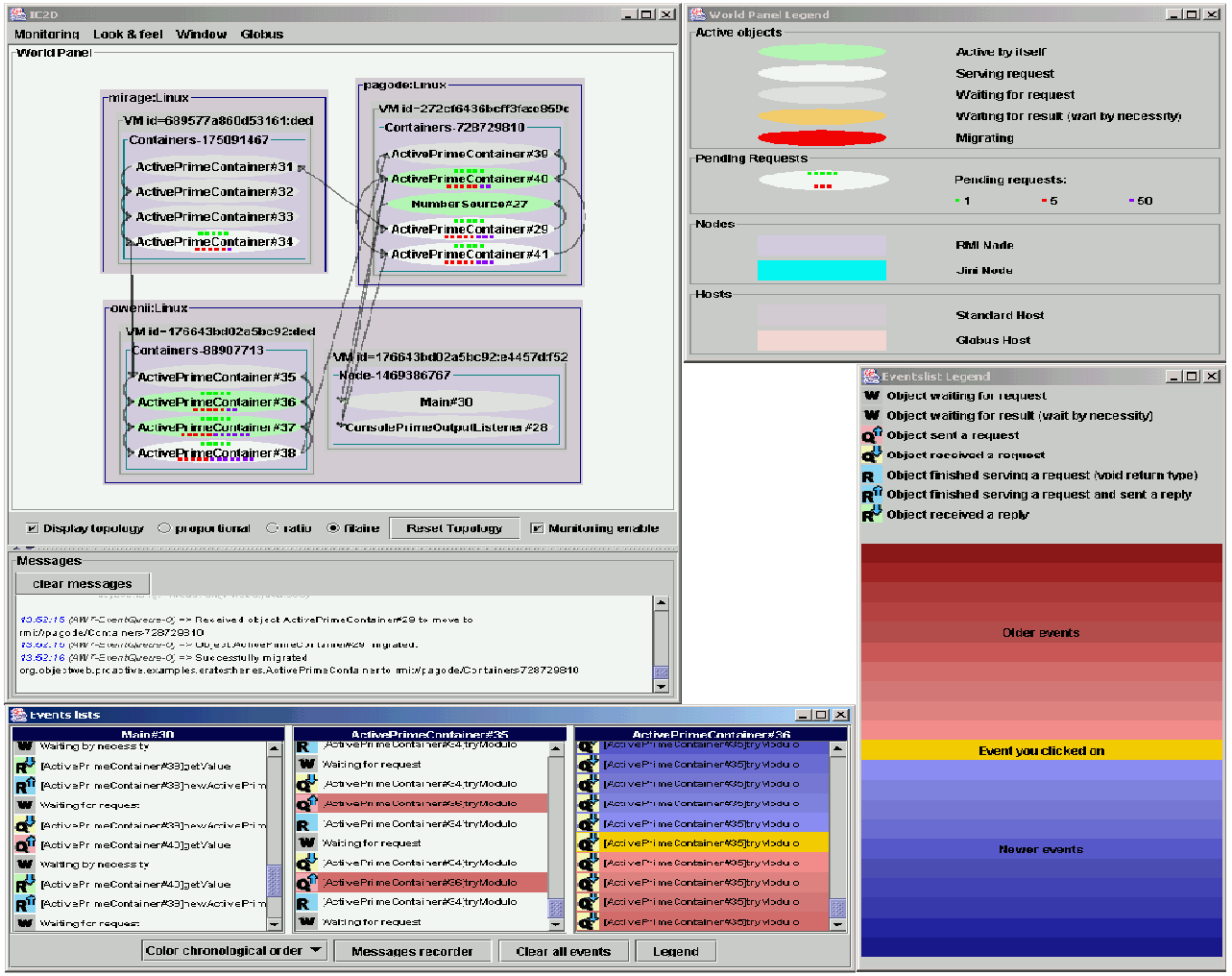
- Hosts, Java Virtual Machines, Nodes, Active Objects
- Topology: reference and communications
- Status of active objects (executing, waiting, etc.)
- Migration of activities

Textual Visualization:

- Ordered list of messages, Logical Clock
- Status: waiting for a request or for a data
- Causal dependencies between messages
- Related events (corresponding send and receive, etc.)



IC2D: Logical Clocks



IC2D: Related Events

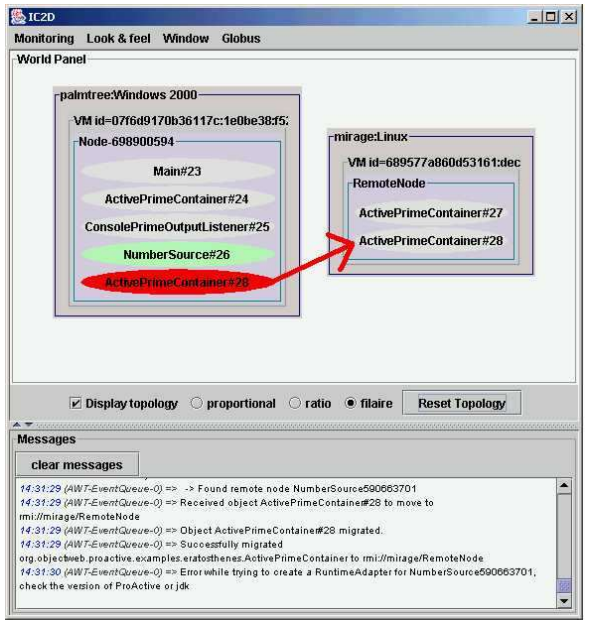
Events:

- Textual and ordered list of events for each Active Object
- Logical clock: related events, ==> Gives a Partial Order

IC2D: Basic features

Graphical Visualization:

Hosts, Java Virtual Machines, Nodes, Active Objects
Topology: reference and communications
Status of active objects (executing, waiting, etc.)
Migration of activities



Textual Visualization:

Ordered list of messages, Logical Clock
Status: waiting for a request or for a data
Causal dependencies between messages
Related events (corresponding send and receive, etc.)

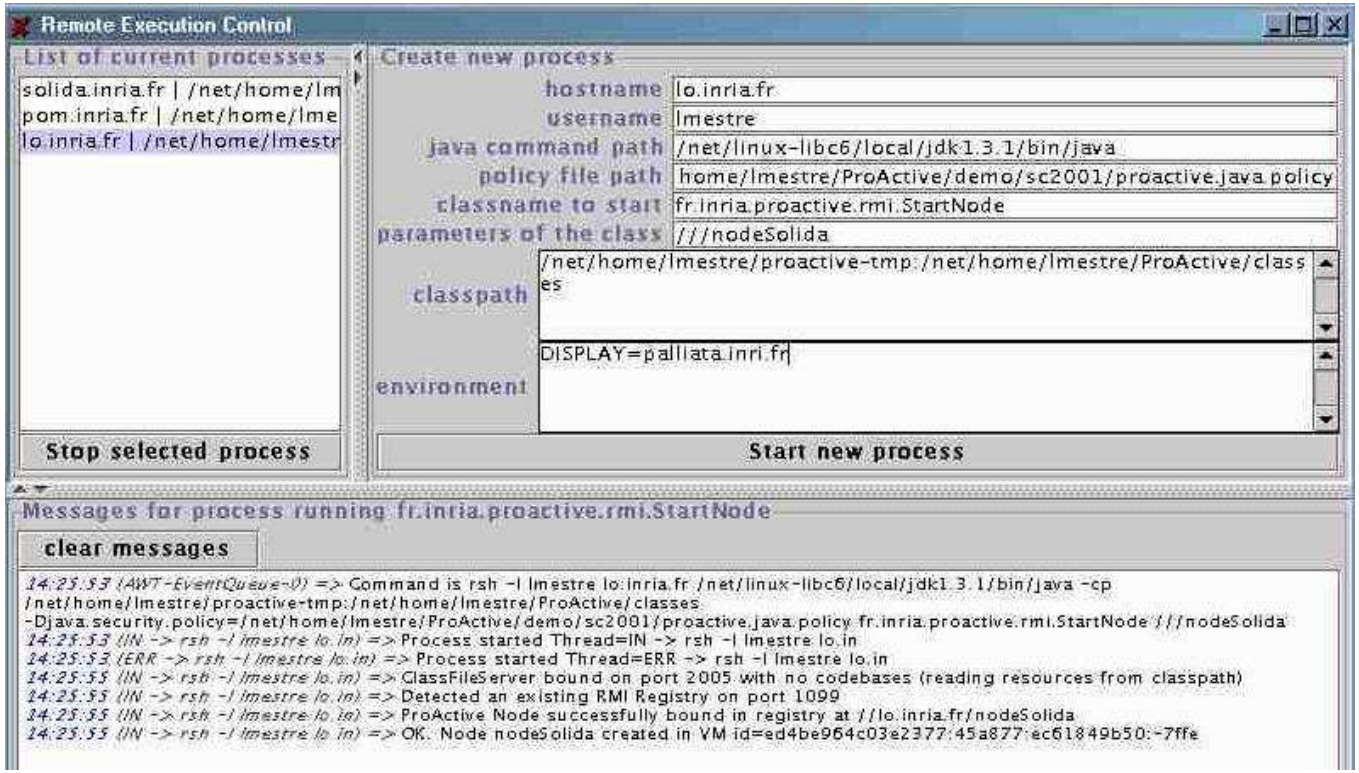
Control and Monitoring:

Drag and Drop migration of executing tasks
Creation of additional JVMs and nodes

IC2D: Dynamic change of Deployment New JVMs

Creation,
Acquisition
of
new JVMs,
and Nodes

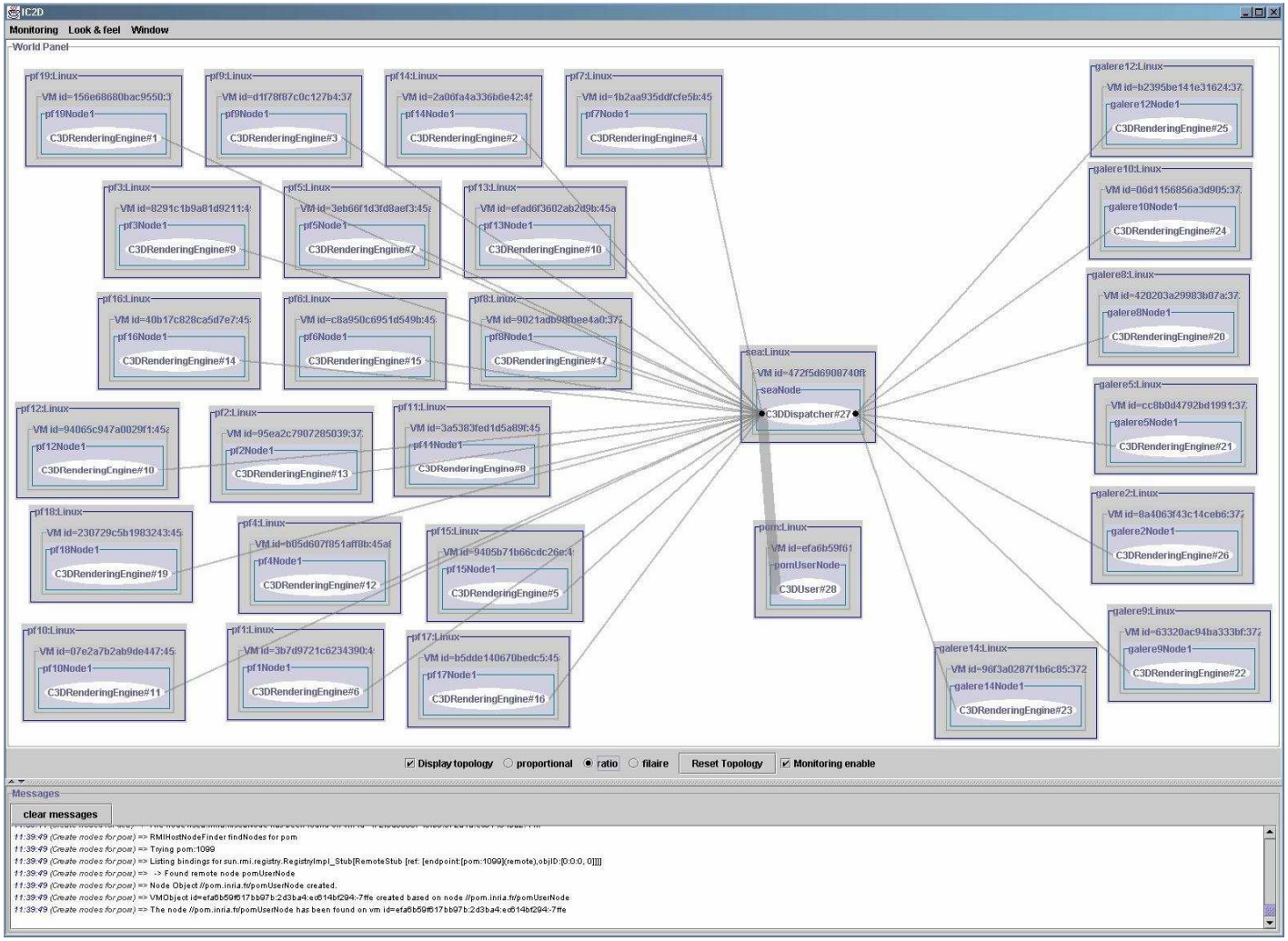
Protocols:
rsh, ssh
Globus,
LSF



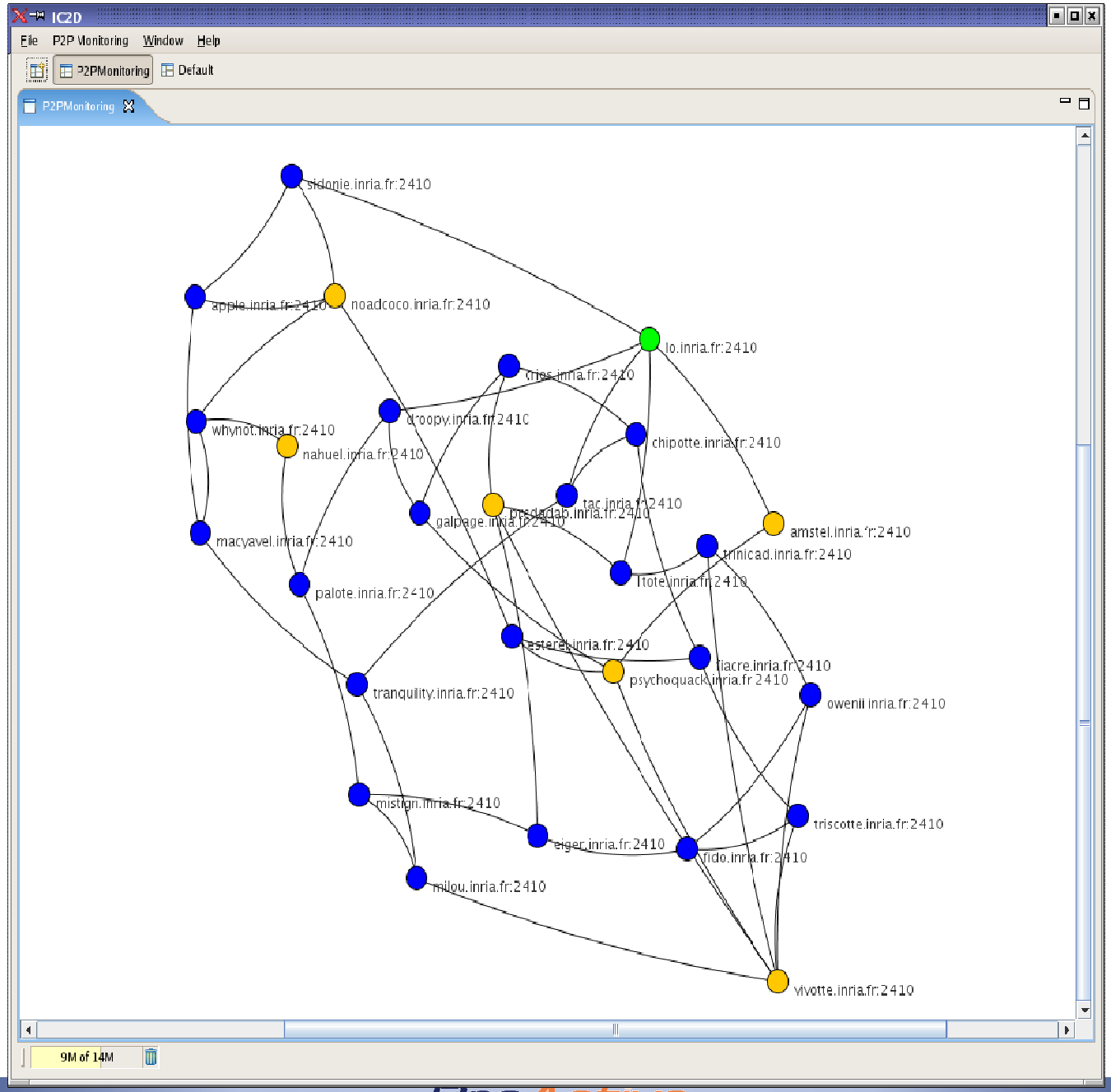
IC2D: Cluster Visualization

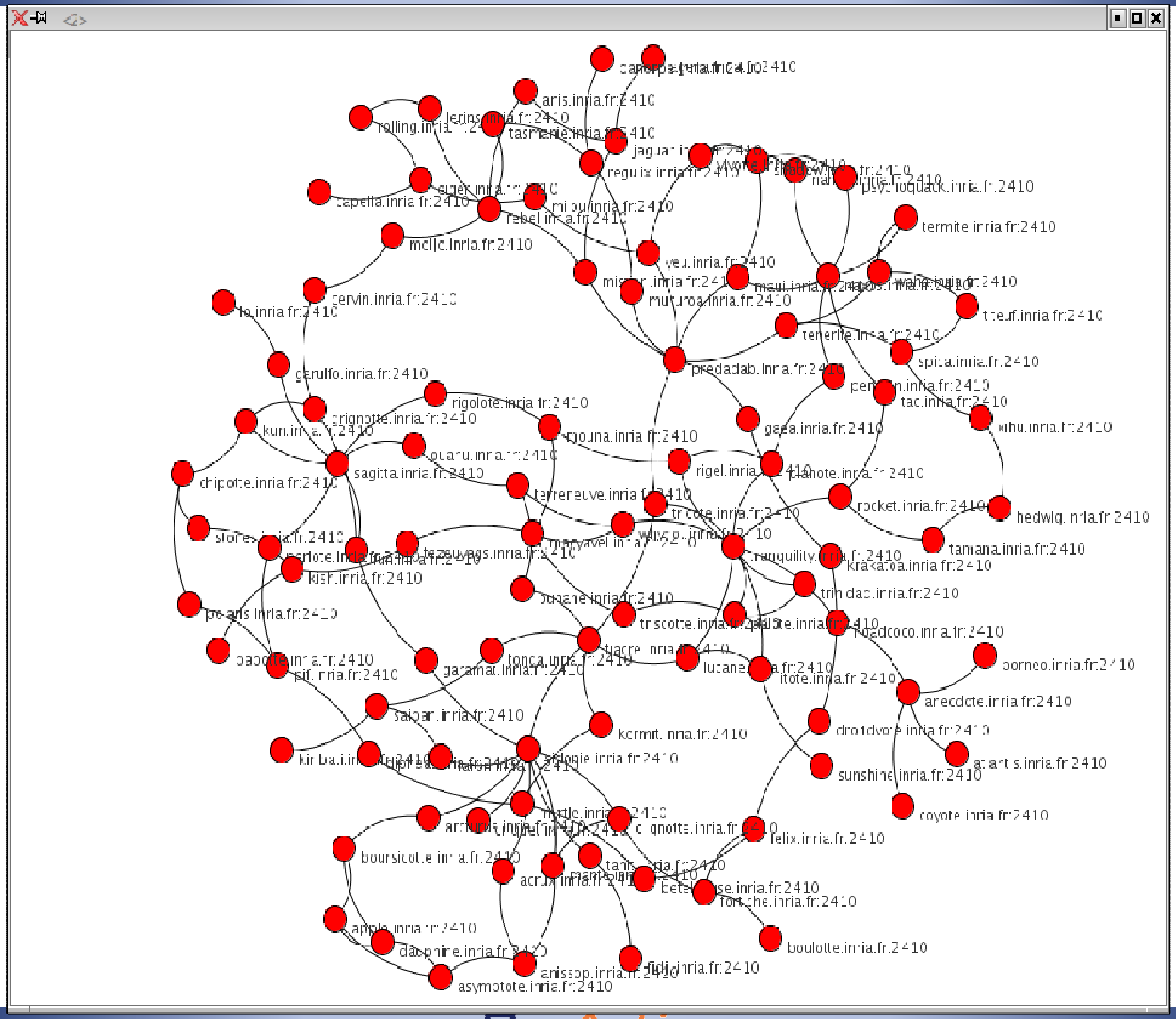
Visualization
of 2 clusters
(1Gbits links)

Featuring
the current
communications
(proportional)



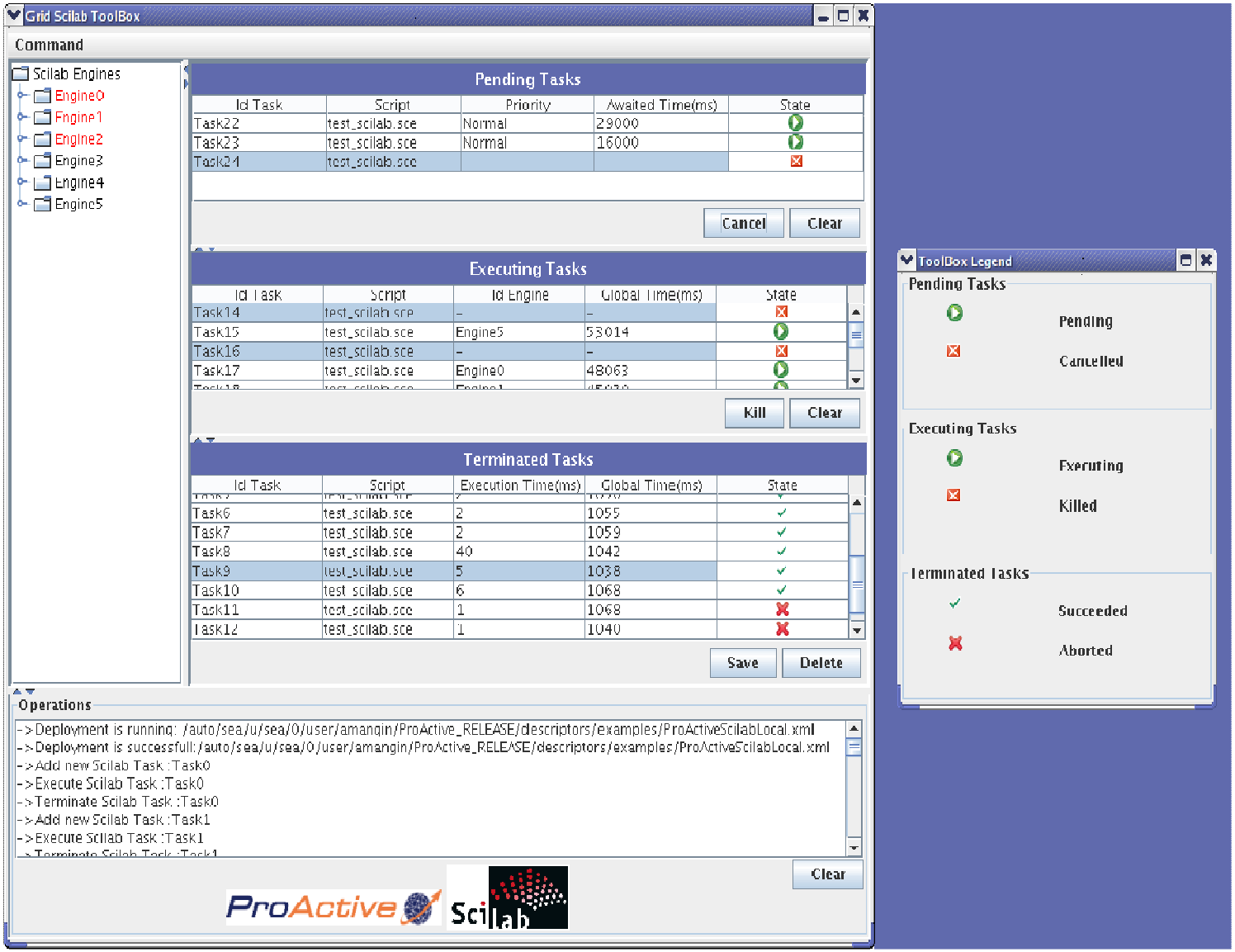
P2P Topology


IC2D
File
P2P Monitoring
Window
Help
P2PMonitoring
Default
P2PMonitoring
sidonie.inria.fr:2410
apple.inria.fr:2410
noadcoco.inria.fr:2410
lo.inria.fr:2410
whynot.inria.fr:2410
nahuel.inria.fr:2410
chipotte.inria.fr:2410
macyavel.inria.fr:2410
palote.inria.fr:2410
amstel.inria.fr:2410
trinicad.inria.fr:2410
tranquility.inria.fr:2410
psychoquack.inria.fr:2410
owenii.inria.fr:2410
triscotte.inria.fr:2410
fido.inria.fr:2410
milou.inria.fr:2410
vivotte.inria.fr:2410
9M of 14M



Grid Scilab:

Free Scientific Software Package


Grid Scilab ToolBox
Command
Scilab Engines
Engine0
Engine1
Engine2
Engine3
Engine4
Engine5
Pending Tasks
Id Task | Script | Priority | Awaited Time(ms) | State
Task22 | test_scilab.sce | Normal | 29000
Task23 | test_scilab.sce | Normal | 16000
Task24 | test_scilab.sce
Cancel
Clear
Executing Tasks
Id Task | Script | Id Engine | Global Time(ms) | State
Task14 | test_scilab.sce | - | -
Task15 | test_scilab.sce | Engine5 | 53014
Task16 | test_scilab.sce | - | -
Task17 | test_scilab.sce | Engine0 | 48063
Kill
Clear
Terminated Tasks
Id Task | Script | Execution Time(ms) | Global Time(ms) | State
Task6 | test_scilab.sce | 2 | 1055
Task7 | test_scilab.sce | 2 | 1059
Task8 | test_scilab.sce | 40 | 1042
Task9 | test_scilab.sce | 5 | 1038
Task10 | test_scilab.sce | 6 | 1068
Task11 | test_scilab.sce | 1 | 1068
Task12 | test_scilab.sce | 1 | 1040
Save
Delete
Operations
->Deployment is running: /auto/sea/u/sea/0/user/amangin/ProActive_RELEASE/descriptors/examples/ProActiveScilabLocal.xml
->Deployment is successfull: /auto/sea/u/sea/0/user/amangin/ProActive_RELEASE/descriptors/examples/ProActiveScilabLocal.xml
->Add new Scilab Task :Task0
->Execute Scilab Task :Task0
->Terminate Scilab Task :Task0
->Add new Scilab Task :Task1
->Execute Scilab Task :Task1
Clear
ProActive
Scilab
ToolBox Legend
Pending Tasks
Pending
Cancelled
Executing Tasks
Executing
Killed
Terminated Tasks
Succeeded
Aborted

ProActive Security

A key principle:

Specify security policies in XML deployment, NOT IN SOURCE!

Security Models

Confidentiality models

- control on data access: Discr (HRU + Take-Grant)
  Mandat (BLP)
- control on data flow

Integrity models

- commercial domain : Biba

Availability models

- resource allocation problem

Access Control Models

Focus on access data control

Users are properly connected:

- Identification + Authentication OK
- no user can steal rights from another user.

Formal Rules to state if a given subject can access to a given object.

Access Control Models

Discretionary Access Control

- control is deduced from subject/group identity
- a subject can give his own rights to some body else: confidence in users !?

Mandatory Access Control

- control is deduced from security level of objects and subjects
- security levels can not be changed nor transfered by a subject, they are managed by a third party

HRU Model (Discr.)

Harrison, Ruzzo & Ullman

Access Control Matrix

S in rows, O + S in columns

Rights:

own , read, write, add, execute, control

	O1	O2	O3	O4	O5	S1	S2	S3
S1	rw		r		w		x	
S2	rw	w			r			
S3	rw		w		rw	x		

Security : Objectives

Access control, communication privacy and integrity

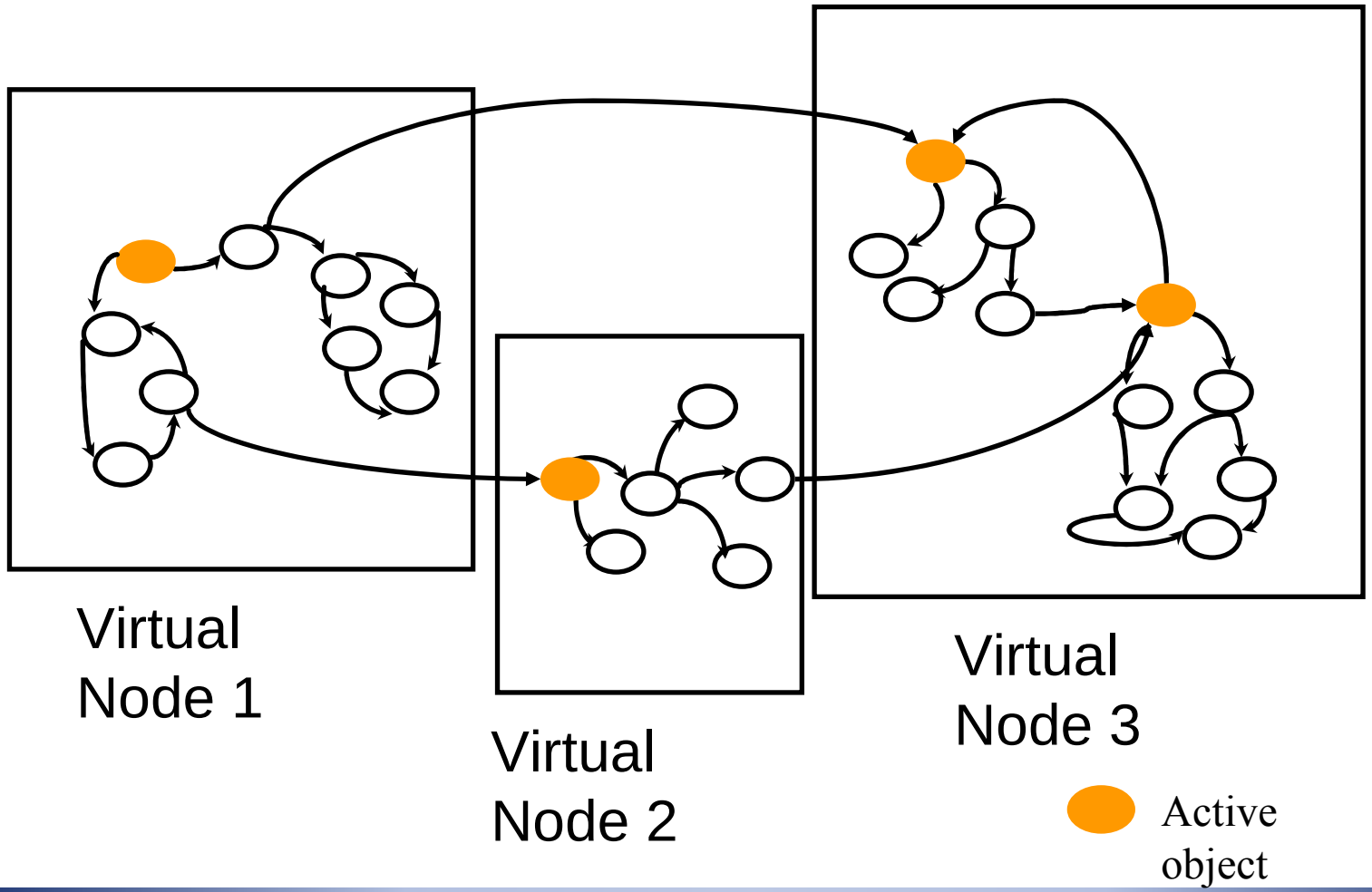
Security at user- and administrator-level

Non functional security

Declarative security language

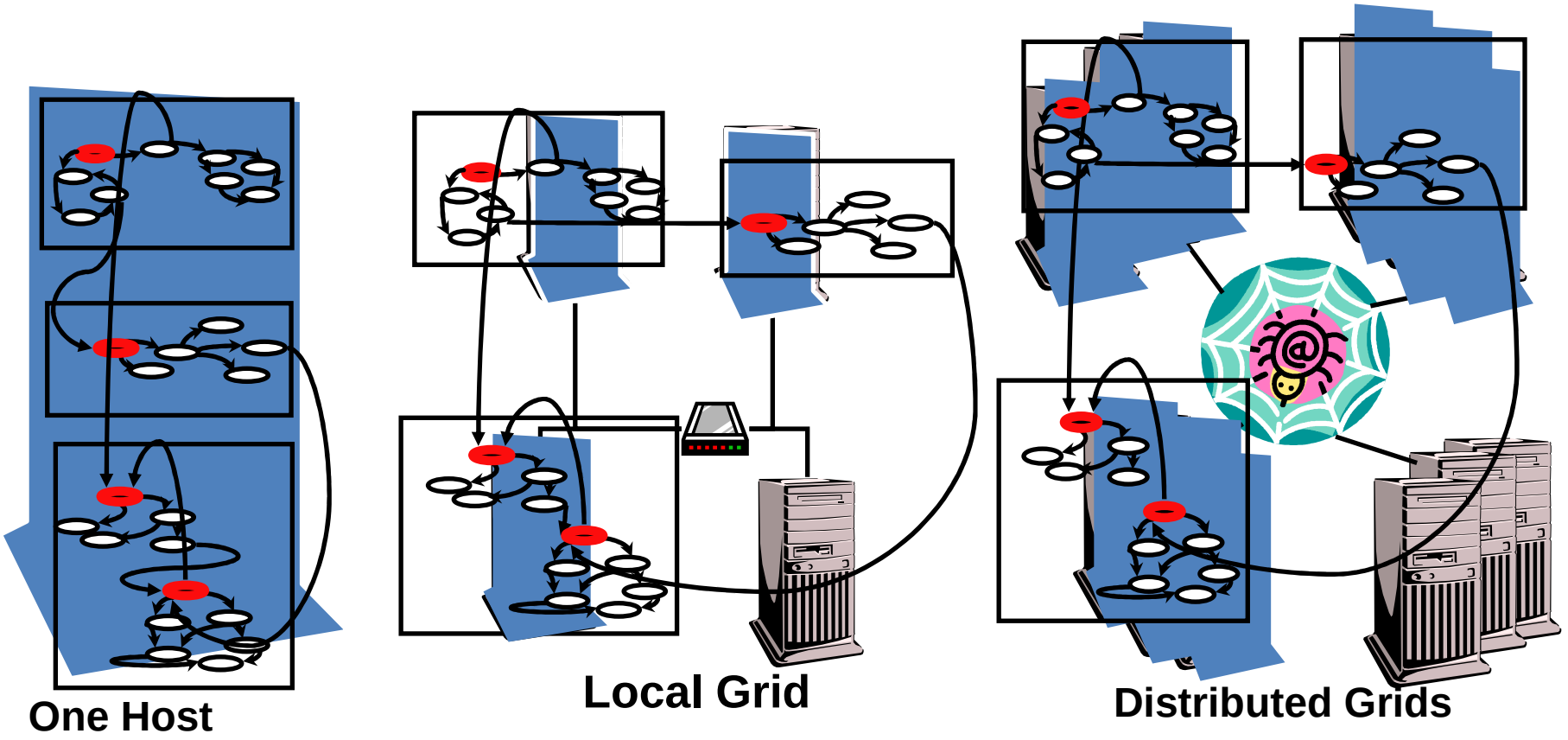
Dynamic when needed

A ProActive Application



Multiple Deployments Issues

Different Deployments ➔ Different Security Policies



Security Issues, in Grid

Authentication of **Computers, Users, and Applications**

Creation, connection to, and monitoring of activities

Authentication, Integrity, Confidentiality (AIC) of communications

Various security policies (physical and logical organizations)

SSH tunneling

Static authentication and encryption

SSH Tunneling

A fact : overprotected clusters

- Firewalls prevent incoming connections
- Use of private addresses
- NAT, IP Address filtering, …

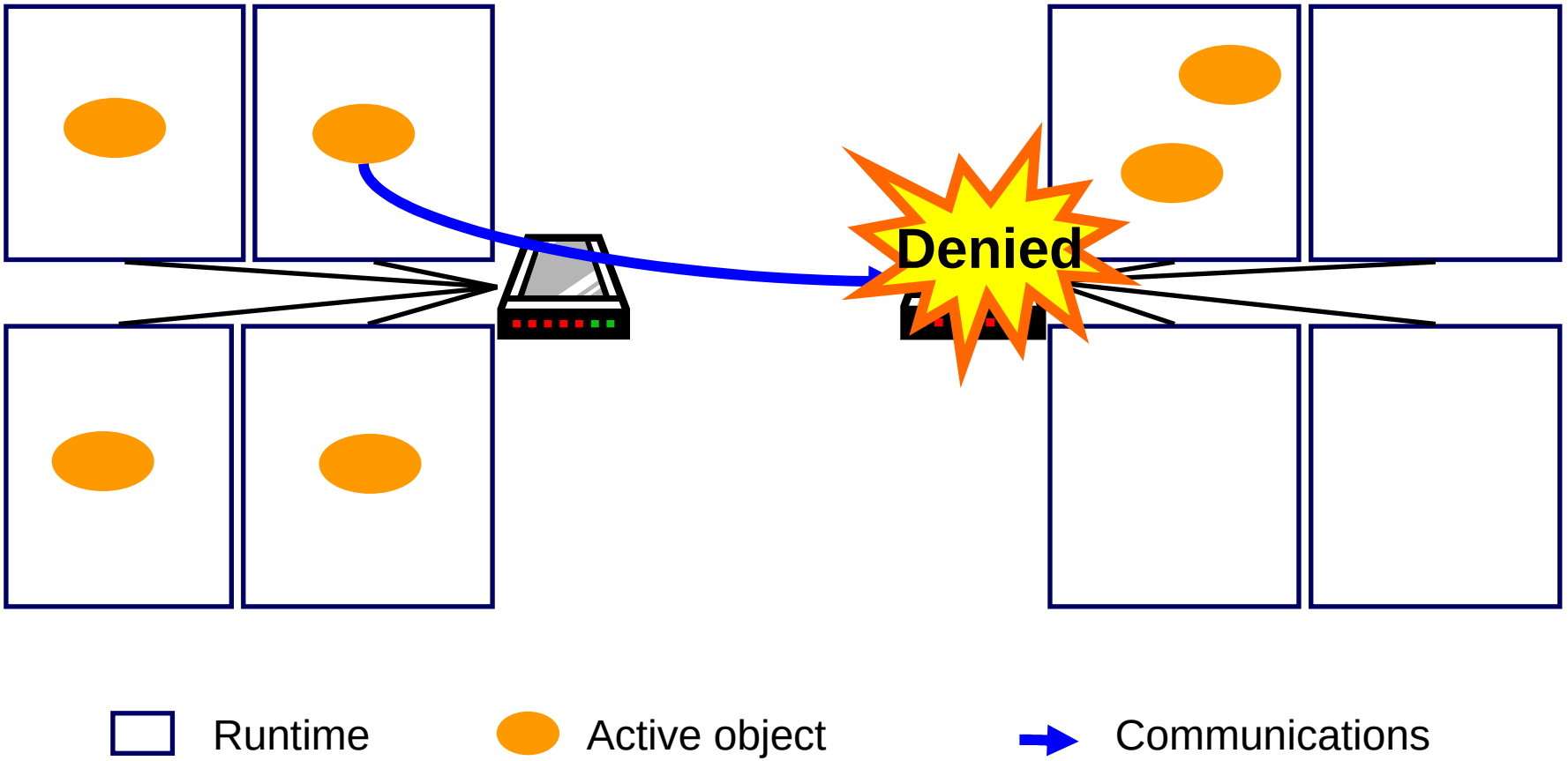
A consequence :

- Multi clustering is a NIGHTMARE

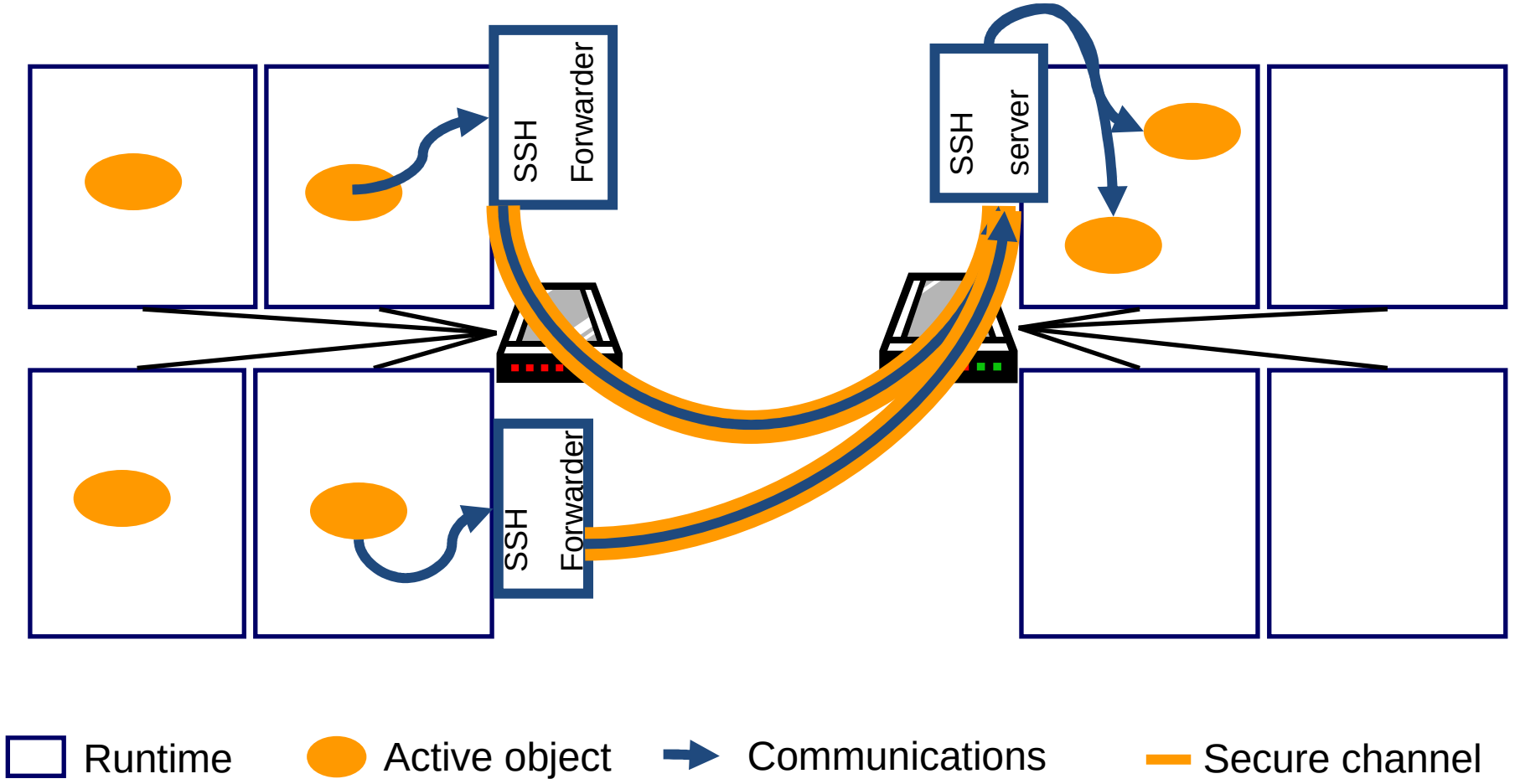
Context :

- SSH protocol : encrypt network traffic
- Administrators accept to open SSH port
- SSH provides encryption

Without SSH Tunneling



With SSH Tunneling



Abstract

Pros :

- VPN like
- Static authentication
- Static encryption of communication

Cons :

- No notion of security at application level
- Static feature

Dynamic Grid Security

Principles :

- Non-functional security
- Hierarchical security domains
- Dynamic policy negotiation
- Certification chain to identify users, JVMs, objects
- Application security policies set by deployment descriptors

Issues for (Dynamic) Grid Security

Authentication of Computers, Users, and Applications

Creation, connection to, and monitoring of activities

Authentication, Integrity and Confidentiality (AIC)

Hierarchical domains

Security Policies: Domain, User, Application

Variation in Grid connectivity: LAN, Wireless, VPN, Internet

Variation in deployment

Objectives

Goals :

- Authentication of Computers, Users, and Applications
- Communication authentication, privacy and integrity
- Security defined at user and administrator level
- Easy and adaptable configuration
- Support for current middlewares features :
  - deployment, migration, group communication, components

Ways :

- Ubiquitous Security (Meta Object Protocol)
- Logical Security Architecture / Abstract Deployment
- Declarative Security Language

Public Key Infrastructure

Elements of PKI

Certificate Authorities (CA)

- Public/Private Key Pairs - Key management
- X.509 Identity Certificates - Certificate management

Certificate Authority

A trusted third party - must be a secure server
Signs and publishes **X.509 Identity certificates**
Revokes certificates and publishes a Certification Revocation List (CRL)
Many vendors :

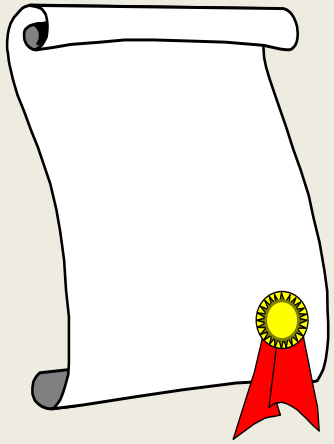
- *OpenSSL* - open source, very simple
- *Netscape* - free for limited number of certificates
- *Entrust* - Can be run by enterprise or by Entrust
- *Verisign* - Run by Verisign under contract to enterprise
- *RSA Security* - Keon servers

Authentication : X509 Certificate

Requestor Generates Key Pair

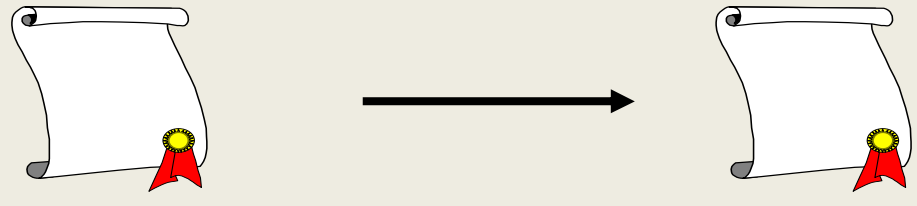
CA Verifies ID, Key Pair, and User Eligibility

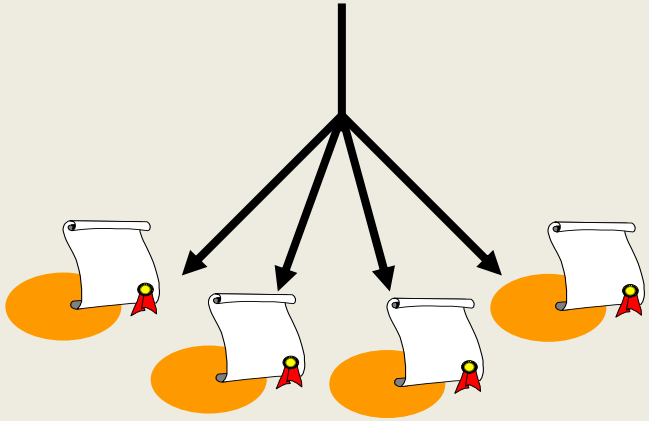
CA Binds Public Key to ID by Signing the Certificate



CA Presents Signed X509 v3 Certificate to Requestor

Application authentication





X.509 Identity Certificates

Distinguished Name of user

C=FR, O=INRIA , OU=Sophia, CN=Arnaud Contes

DN of Issuer

C=FR, O=INRIA, CN=SOPHIA-CA

Validity dates:

Not before <date>, Not after <date>

User's public key

V3- extensions

Signed by CA

Authorization with ACLs

Access control lists (ACLs)

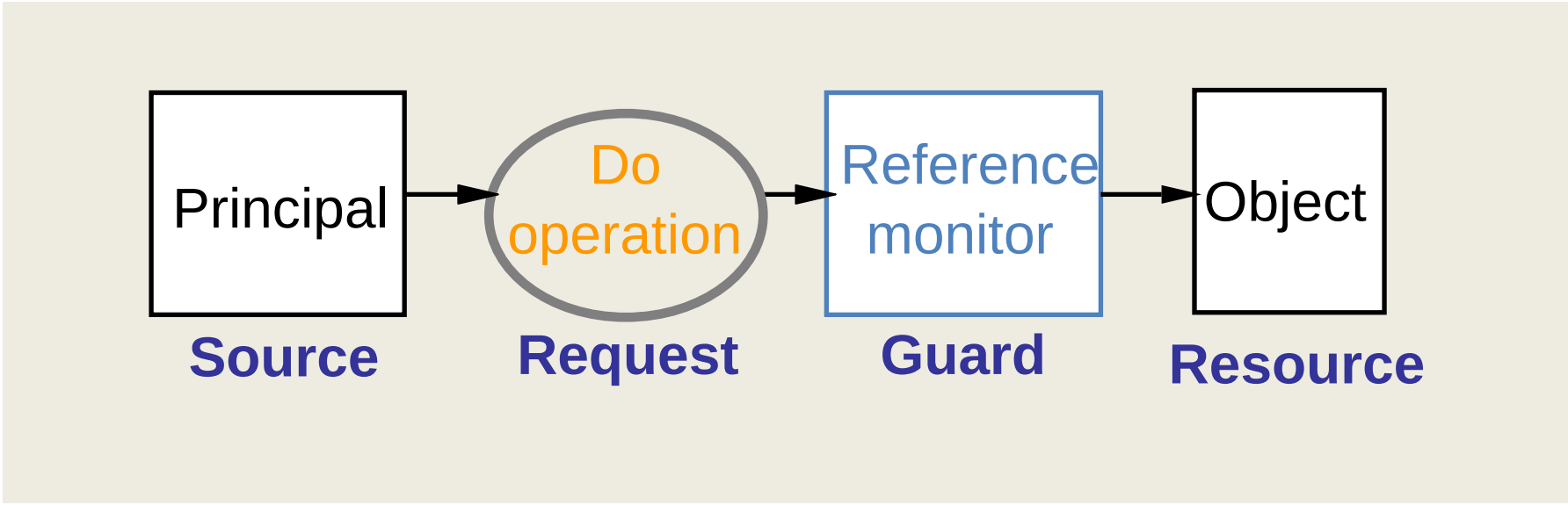
- An object O has an ACL that says: principal P may access O.
  - Lampson may read and write O
  - MSR may append to O

ACLs must use names for principals

- so that people can read them

The Access Control Model

Guards control access to valued resources.



Application

Composed of active objects

each active object has :

- an owner
- depends on an application

Each application has a different identity even if launched by the same person, on the same machine, only certificate are important

Active objects belongs to an application which belongs to an user which belongs to a domain.

User certificate ==> Application certificate

- user private key used only for generating application certificate

Application certificate ==> active object certificate

ProActive Security Principles

Non-functional security

Hierarchical security domains

Dynamic policy negotiation

Certification chain to identify users, JVMs, objects

Application security policies set by deployment descriptors

ProActive Security: Key Features

Authentication of users and applications (PKI X 509 certificates)

Authentication, Integrity and Confidentiality of communications

[A,I,C]

In XML deployment files, Not In Source

Mobility Aware

Dynamically negotiated policies

What's a secure ProActive application

Composed of 'classic' active objects, no change in sources.

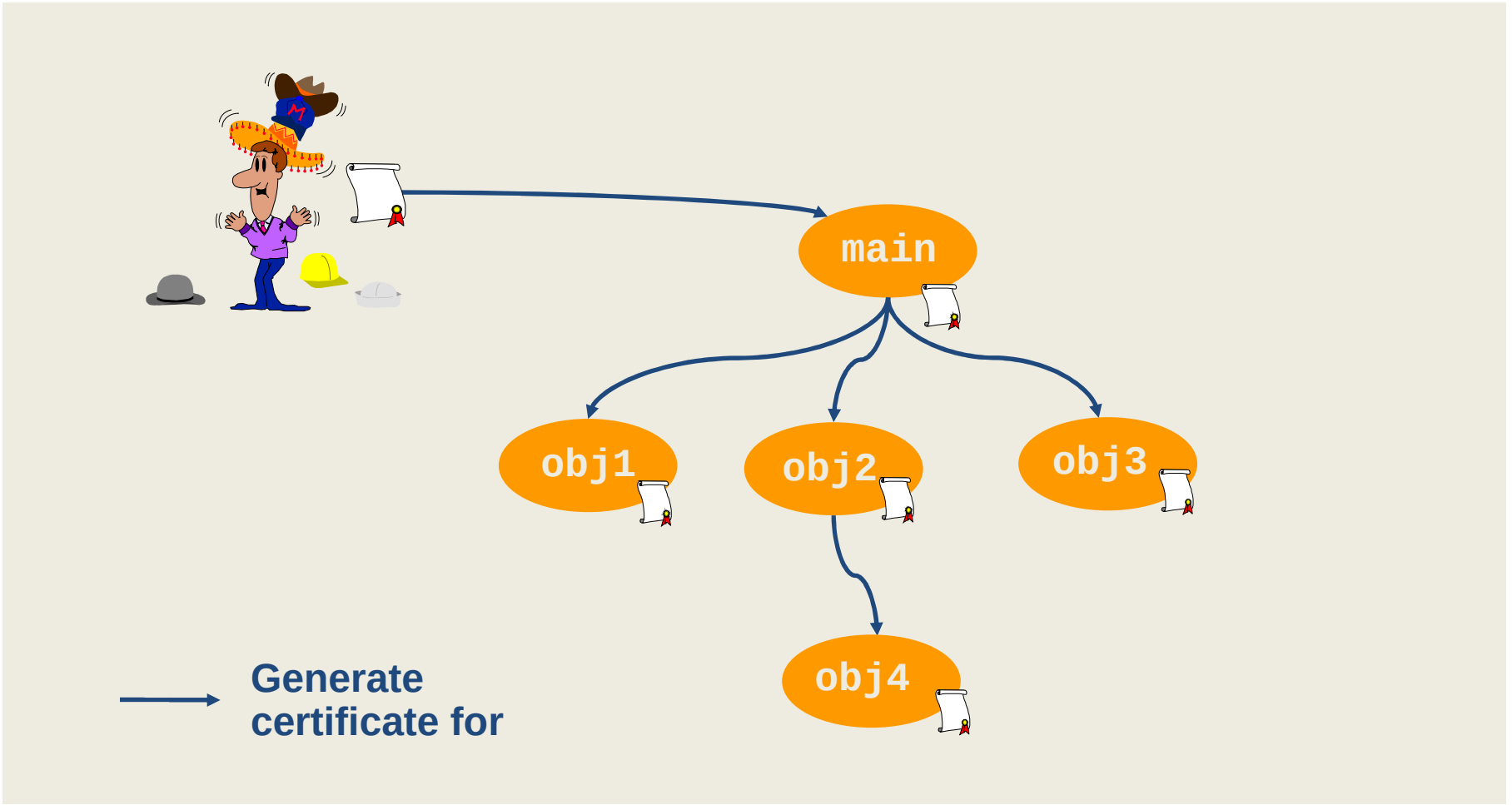
Each application is different, even if launched by the same person, on the same machine.

PKI Certification chain to identify users,applications,objects

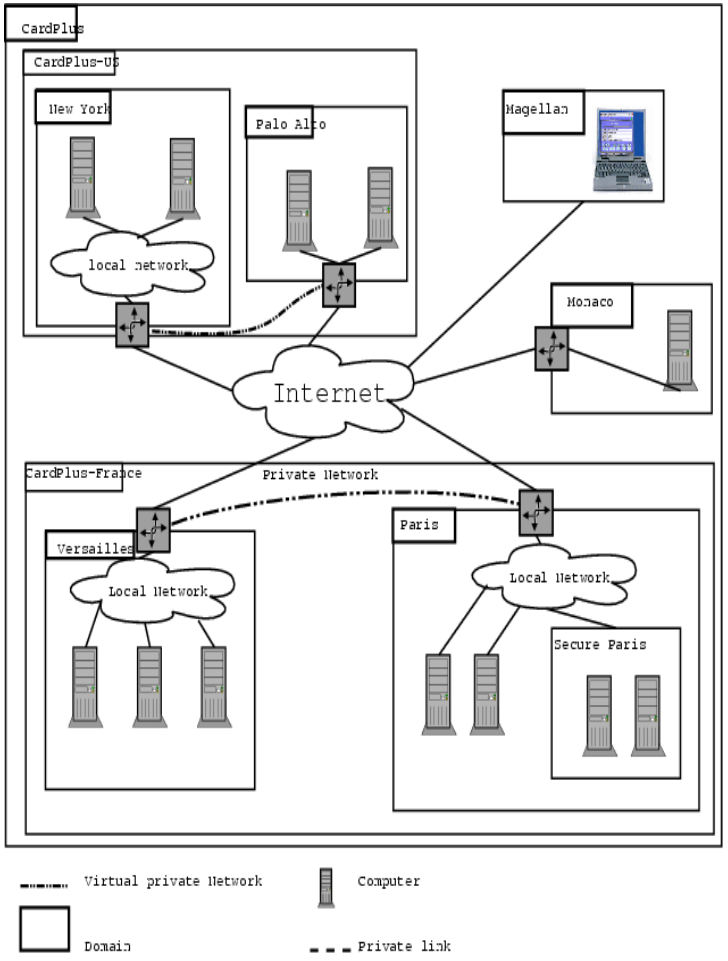
- User certificate => Application certificate =>active object certificate
- user private key used only once for generating application certificate

Security policy gets from deployment descriptors.

Certification Chain



Domains



Logical way to **group many entities** that have the same security needs.

Domains are hierarchical.

Sub-domains **inherits** parent's security policies.

Default: Sub-domains cannot weaken parent's security policies.

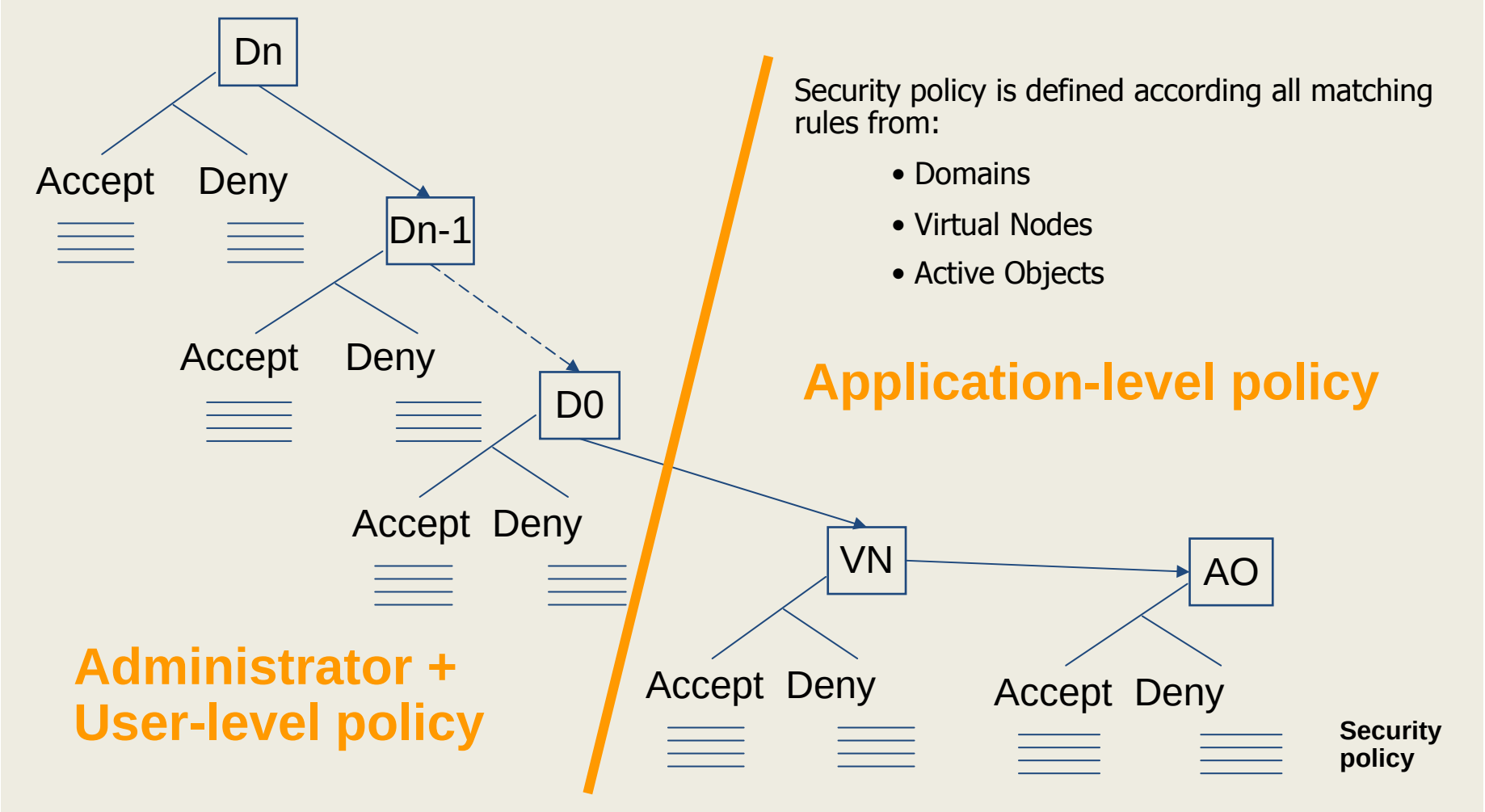
'Can override': a domain authorizes an entity to **override its policies**

Find the first common domain if exists

A domain has a policy server + a certification authority

Dynamically configurable via SSL connections

Multi-level Policies



Security Rule

Entities -> Entities : Interactions # Security Attributes

Entities :

- Domain
- User
- Virtual Node
- Object

Interactions :

- JVMCreation
- NodeCreation
- CodeLoading
- ObjectCreation
- ObjectMigration
- Request
- Reply
- Listing

Attributes :

- Authentication
- Integrity
- Confidentiality

Each attribute can be :

- Allowed
- Optional
- Disallowed

Combining Policies

Search for the most specific rule in each domain.

Retrieve all matching rules in the Domain hierarchy, the Virtual Node and the Active Object.

Compute policies according to security attributes.

Sender \ Receiver	Required (+)	Optional (?)	Disallowed (-)
Required (+)	+	+	invalid
Optional (?)	+	?	-
Disallowed (-)	invalid	-	-

Descriptor Security Model

A key principle:

- Specify security policies in the XML deployment, NOT IN SOURCE!

In program source:

- Virtual Node (VN, a string name):

In XML descriptors:

- List of policy rules
- Trusted Certification Authorities

```
VirtualNodes
                vn1, vn2
SECURITY:
  VN [vn1] -> VN [vn2] : Q,P  #  [+A,?I,+C]
  VN [vn1] -> VN [vn2] : M  #  Forbidden
  VN [vn2] -> VN [vn1] : Q,P  #  [?A,?I,?C]
  VN [vn2] -> VN [vn1] : M  # Forbidden
Mapping:
   vn1 --> GridAComputers,
GridBComputers
   vn2 --> GridAComputers
JVMs:
   /…/
```


ProActive Security Manager

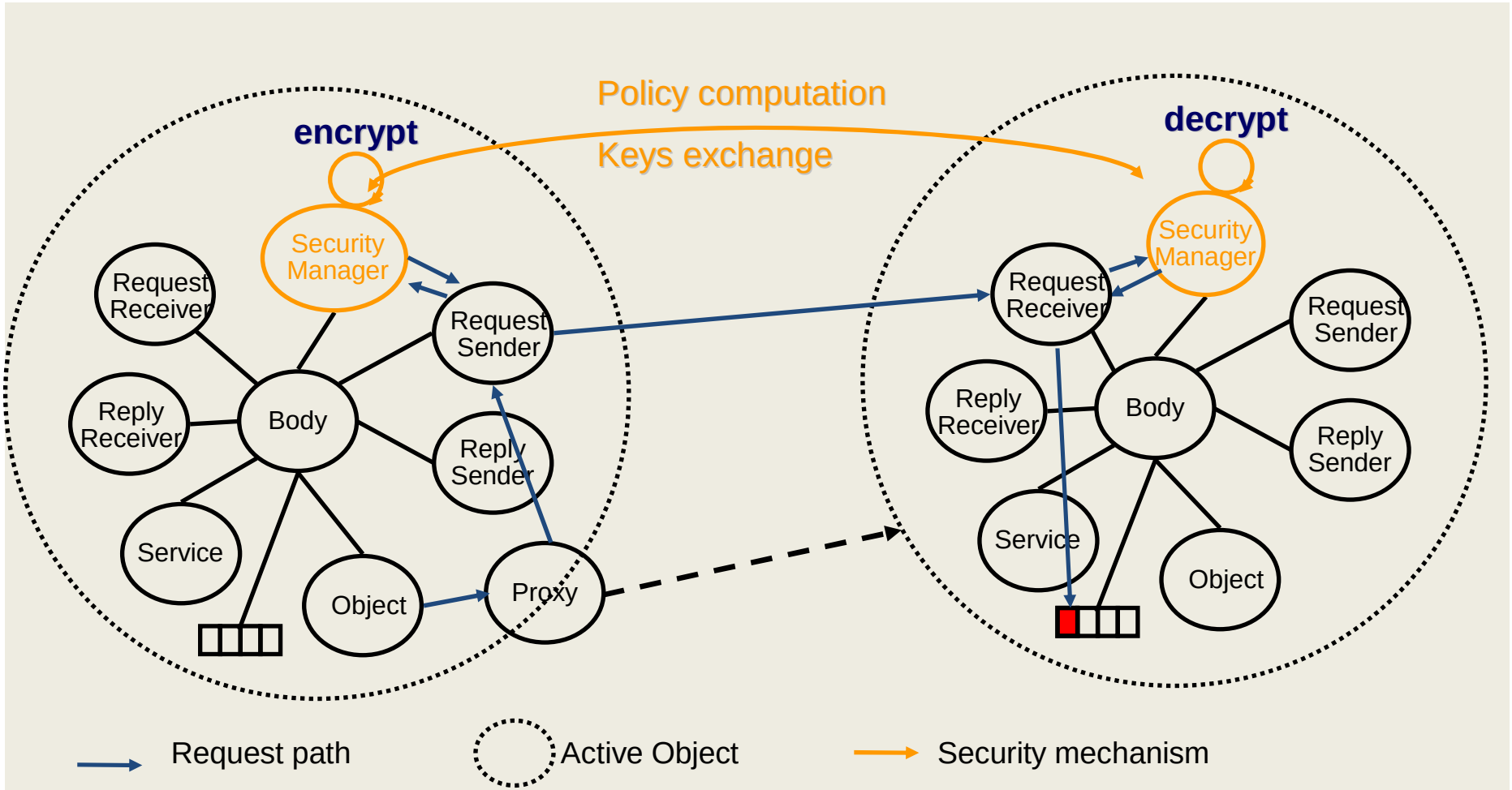
In charge of security for an active object

Retrieve, combine and negotiate policies

Negotiate session key,

Encrypt/decrypt messages

Secure Request to an Active Object



Security Example

2 domains GridA & GridB with security policies

Domain [GridA] -> Domain [GridB] : Q,P,M # [+A,+I,+C]

Domain [GridB] -> Domain [GridA] : Q,P,M # [+A,+I,+C]

Application :

- 2 Virtual Nodes (vn1,vn2)
- 2 Active objects

VirtualNodes

vn1, vn2

SECURITY:

VN [vn1] -> VN [vn2] : Q,P # [+A,?I,+C]

VN [vn1] -> VN [vn2] : M # Forbidden

VN [vn2] -> VN [vn1] : Q,P # [?A,?I,?C]

VN [vn2] -> VN [vn1] : M # Forbidden

Mapping:

vn1 --> GridAComputers, GridBComputers

vn2 --> GridAComputers

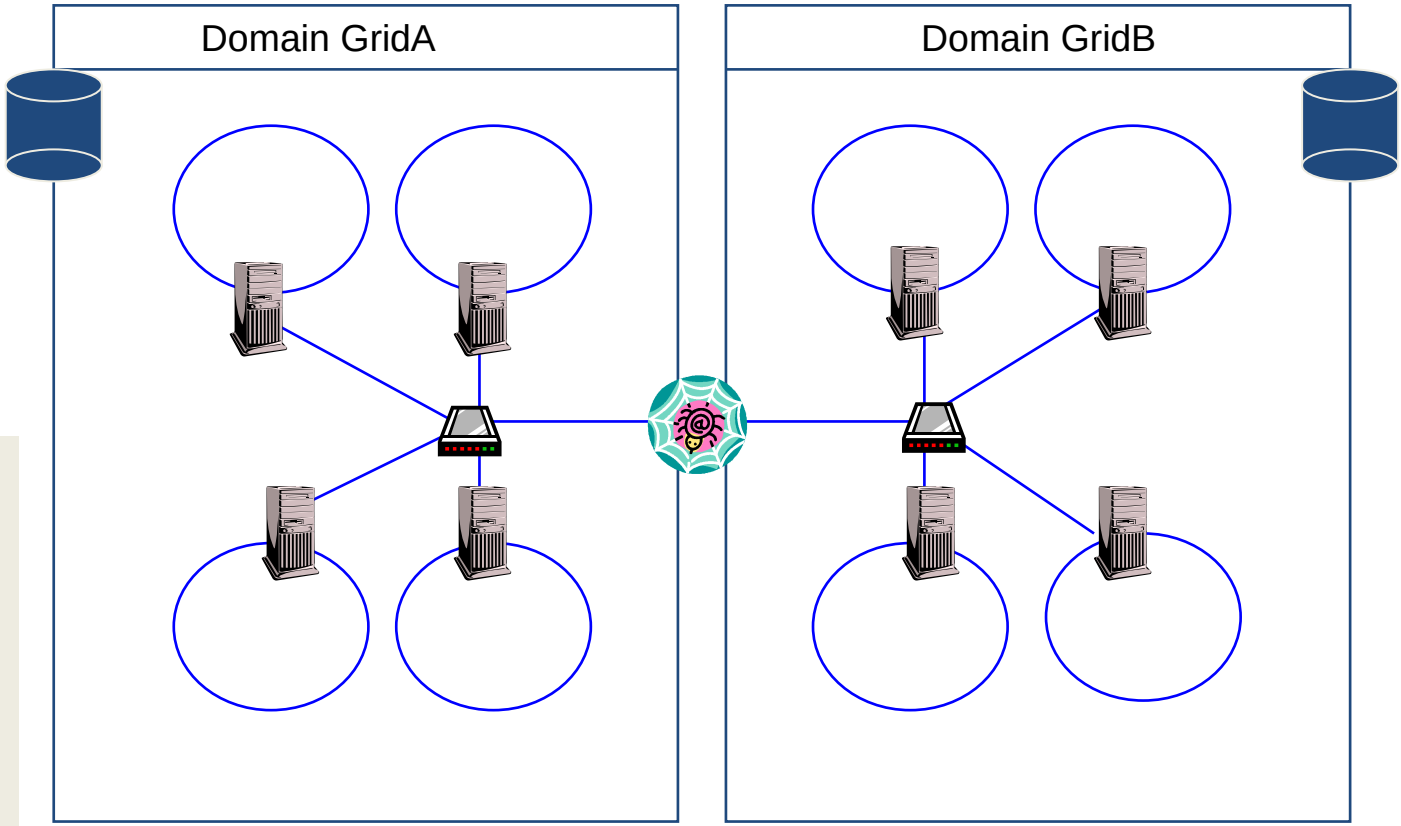
JVMs:

/.../

Example: std. code, no security

```
/…/
proActiveDescriptor.activateMappings();
vn1 = proActiveDescriptor.getVirtualNode("vm1");
vn2 = proActiveDescriptor.getVirtualNode("vm2");
/…/
Flower rose = (Flower) ProActive.newActive(Flower.class,new
                            Object[]{« Rose »}, vn1.getNode()};
Flower daliah = (Flower) ProActive.newActive(Flower.class,new
                            Object[]{« Daliah »}, vn2.getNode()};
/* next VN1 node inside the same domain */
rose.migrateTo(vn1);
/* communication inside the same domain */
rose.sayHelloTo(daliah);
/* other virtual node, forbidden */
rose.migrateTo(vn2);
/* next VN1 Node, other domain */
rose.migrateTo(vn1);
/* communication with another domain */
rose.sayHelloTo(daliah);
```

Example



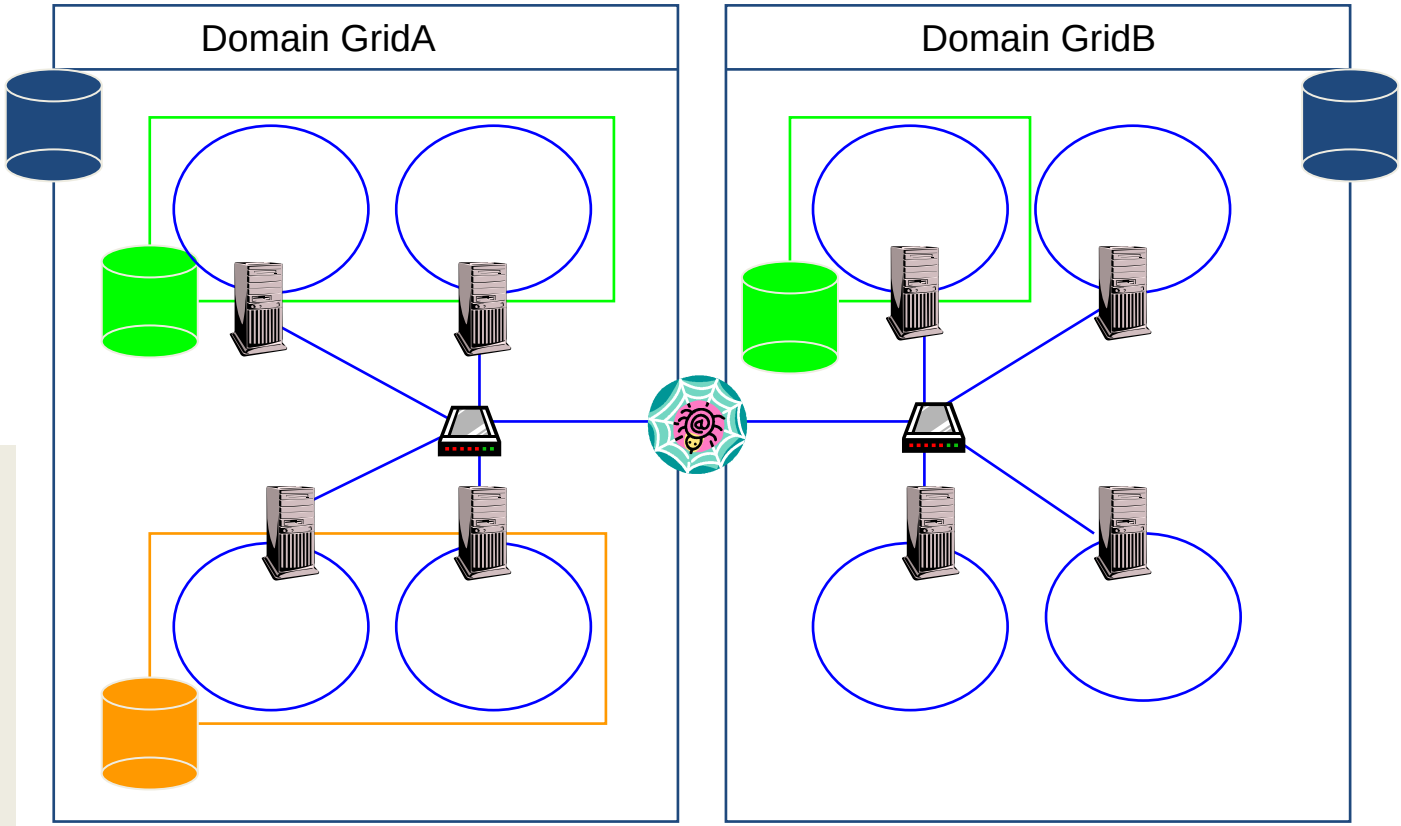
Policy rules databases

VN1

VN2

JVM

Example



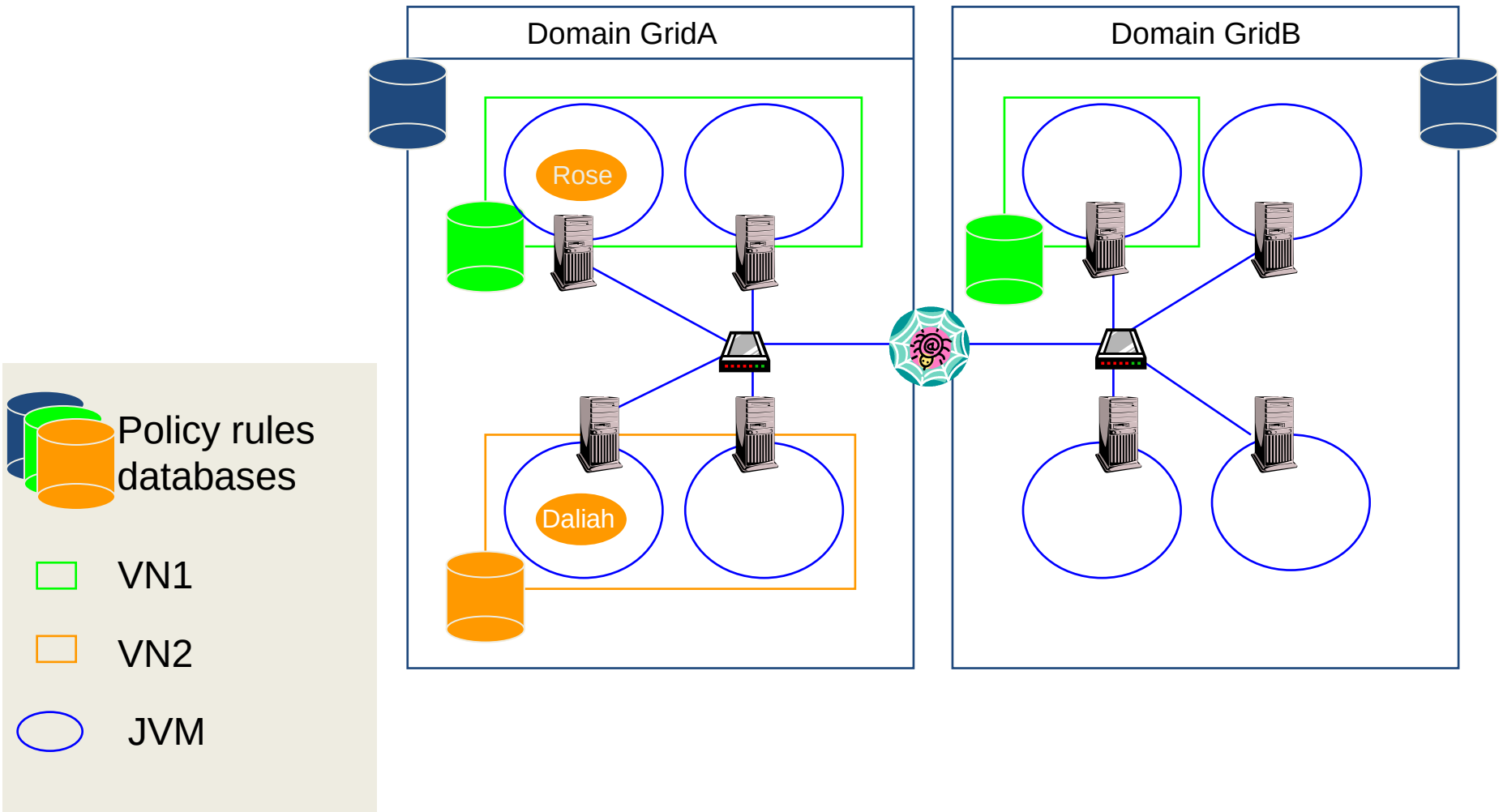
Policy rules databases

VN1

VN2

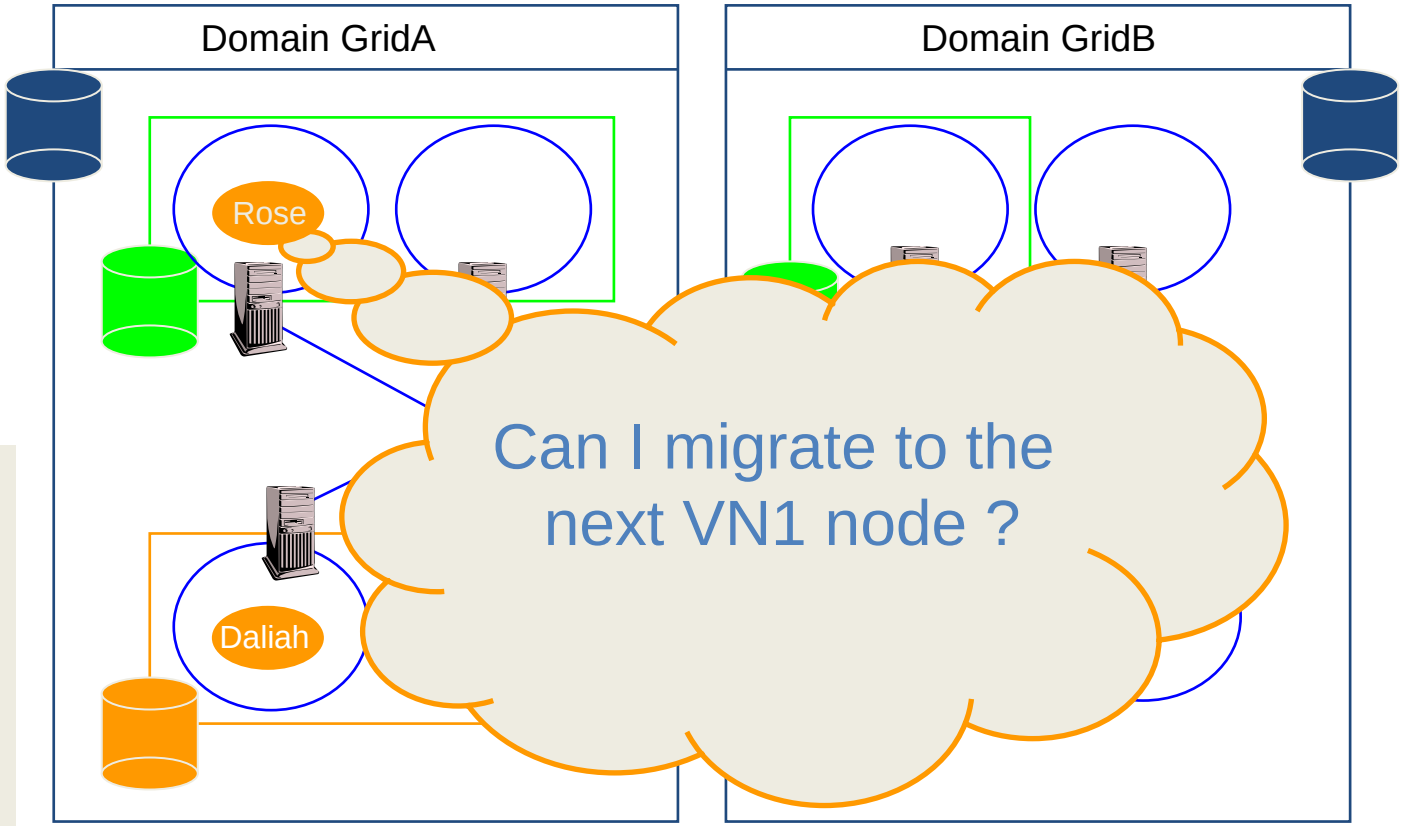
JVM

Example


Domain GridA
Domain GridB
Rose
Daliah
Policy rules databases
VN1
VN2
JVM

Example

Migration :
- same VN
- same domain



Policy rules databases

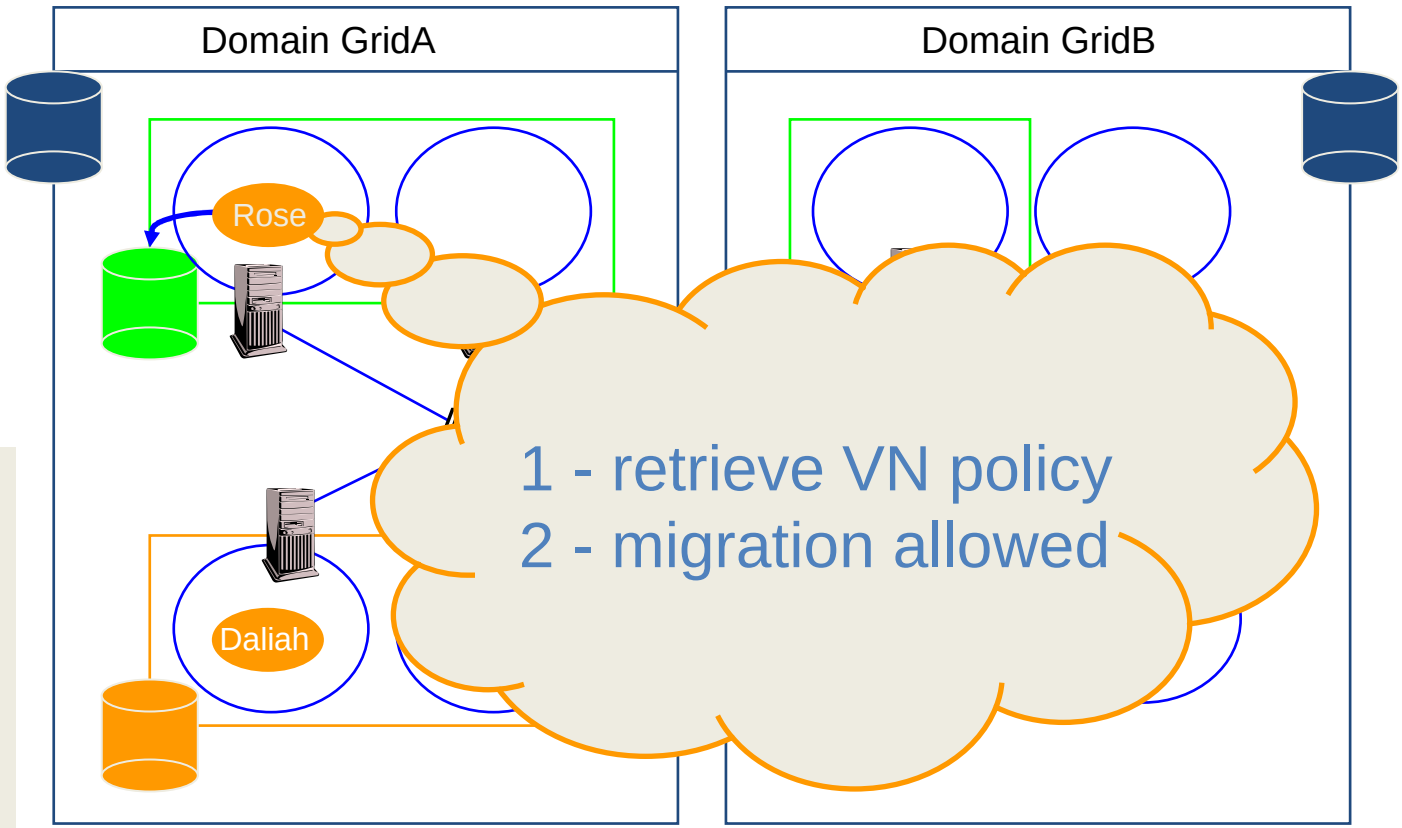
VN1

VN2

JVM

Example

Migration :
- same VN
- same domain



Policy rules databases

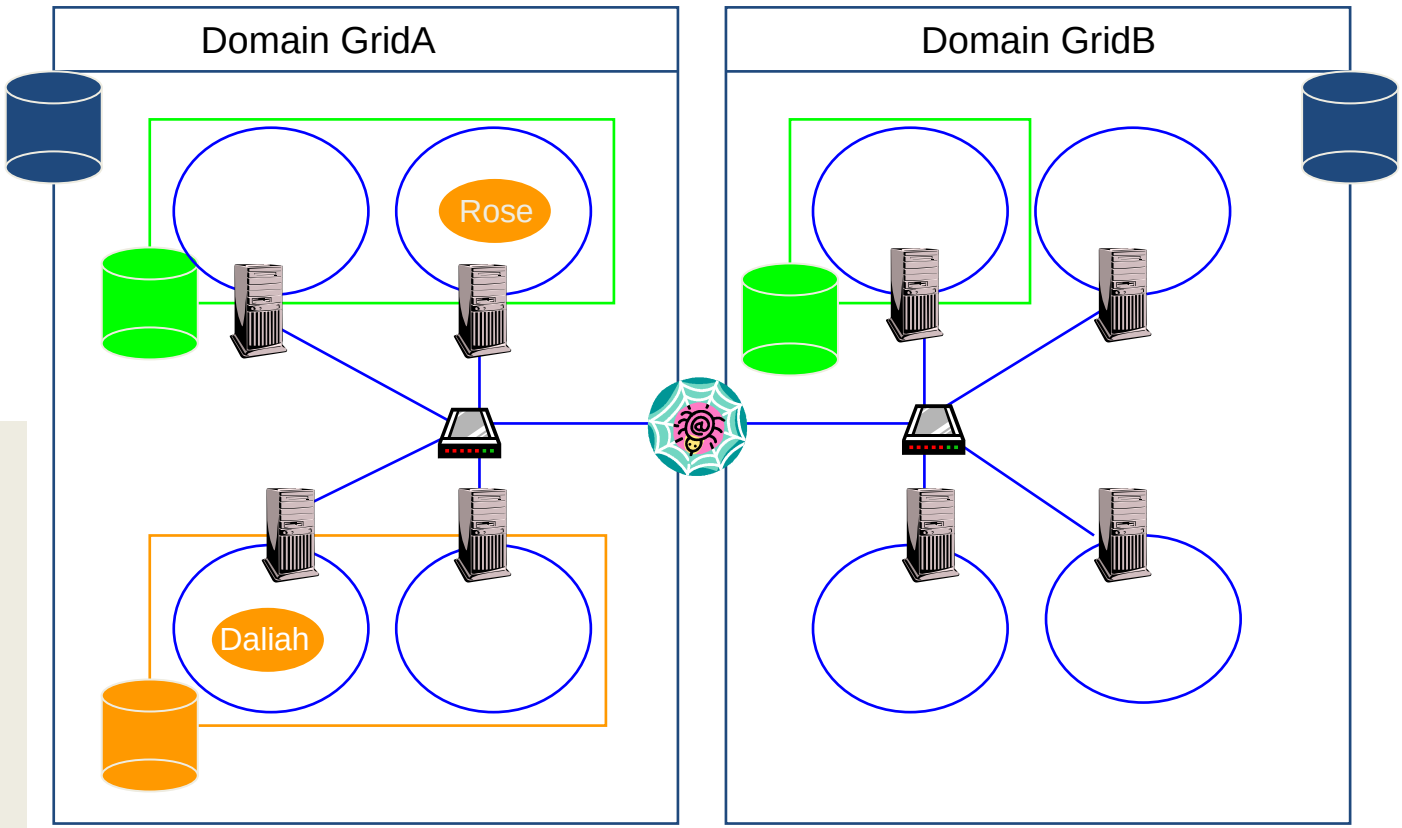
VN1

VN2

JVM

Example

Migration :
- same VN
- same domain




Policy rules databases

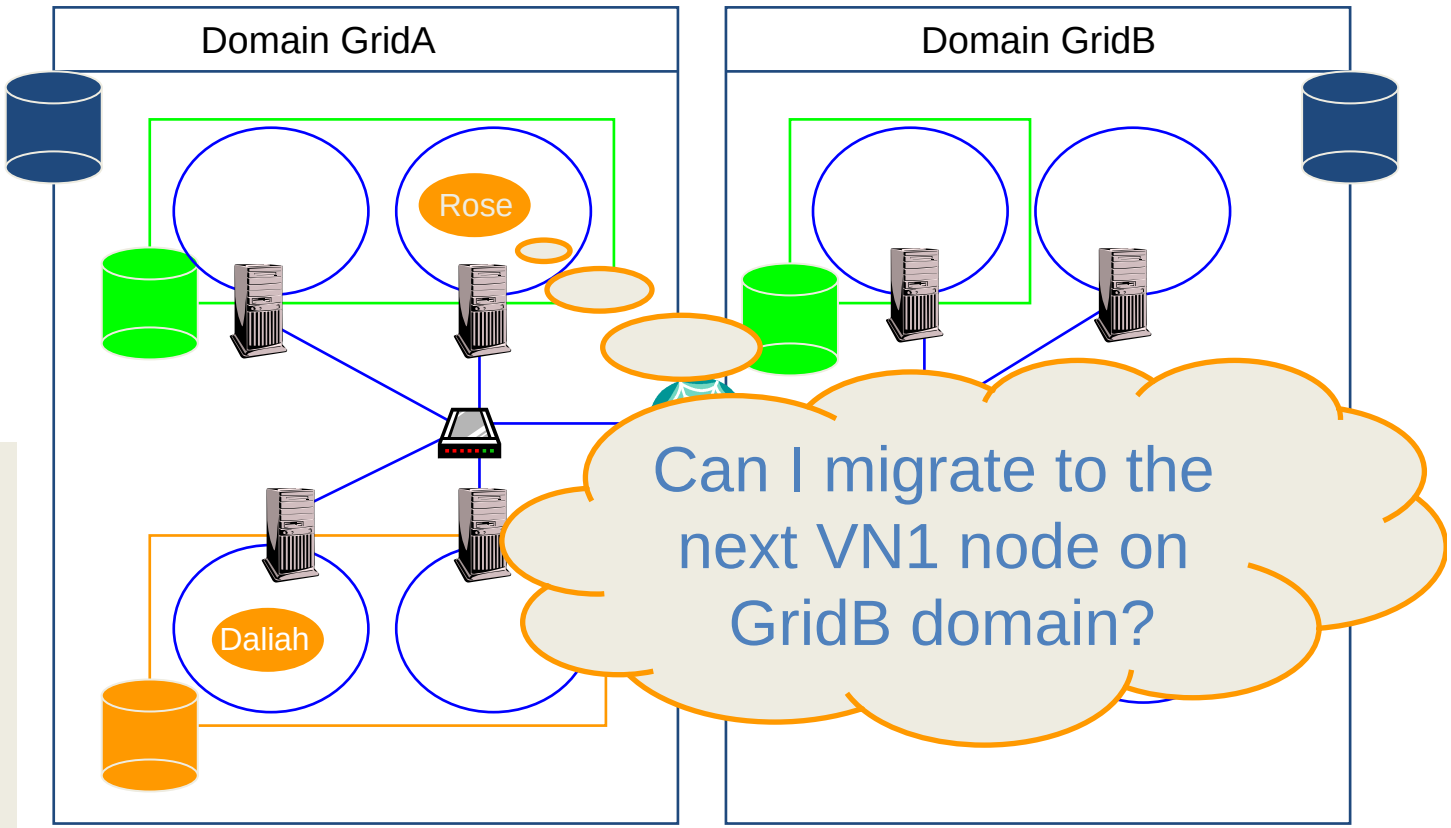
VN1

VN2

JVM

Example

Migration :
- same VN
- same domain



Policy rules databases

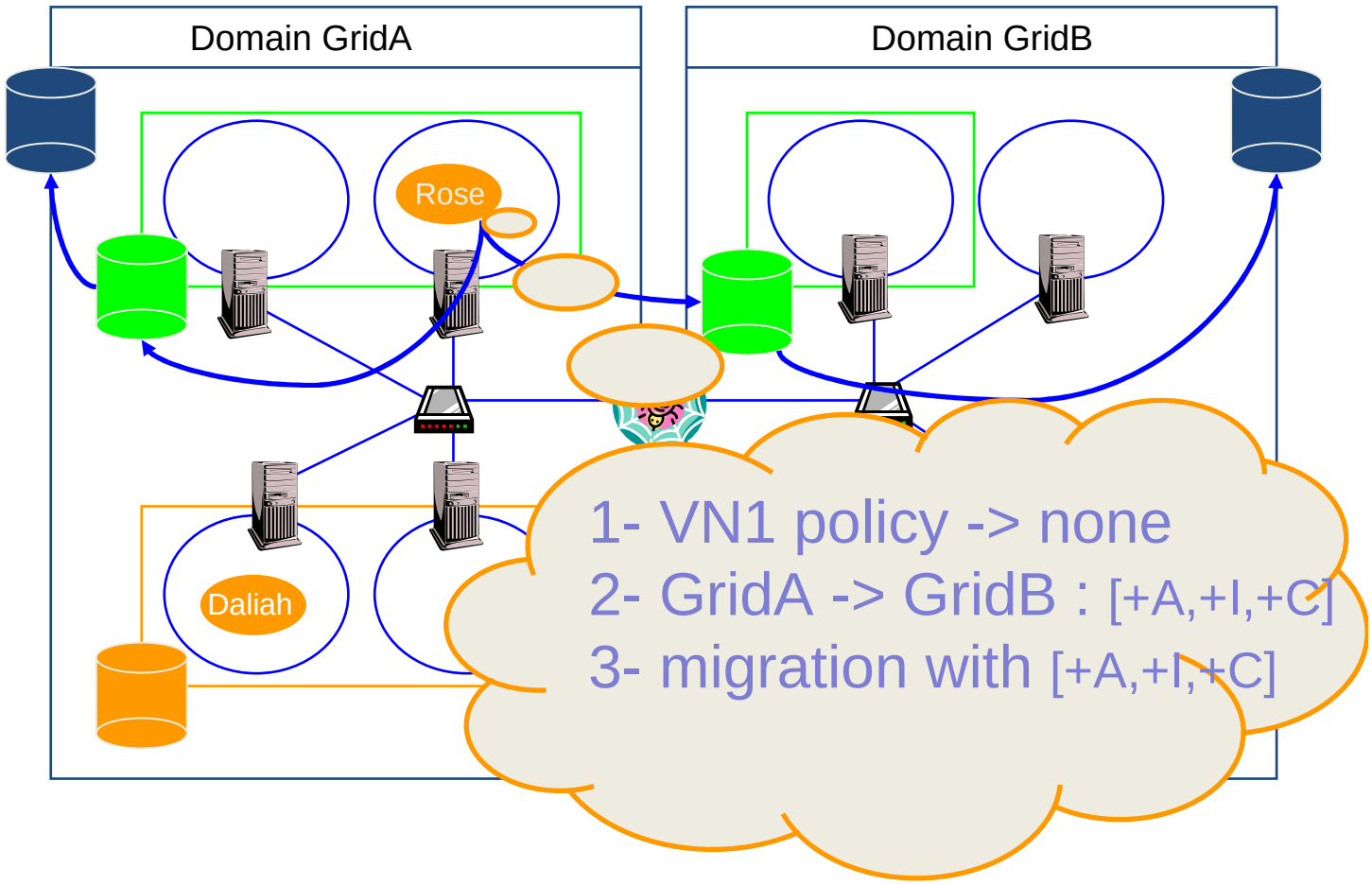
VN1

VN2

JVM

Example

Migration :
- same VN
- same domain



Policy rules databases

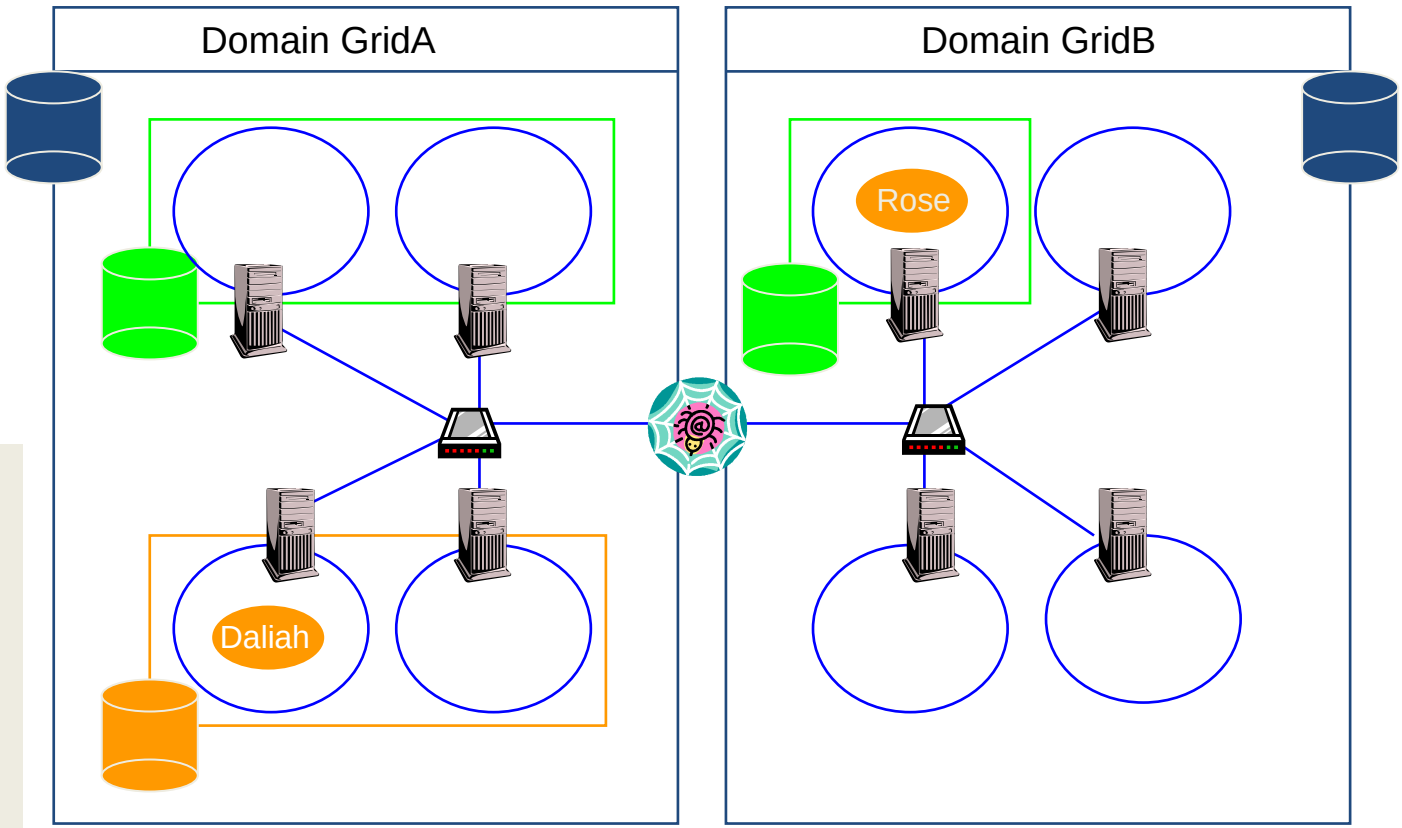
VN1

VN2

JVM

Example

Migration :
- same VN
- same domain




Policy rules databases

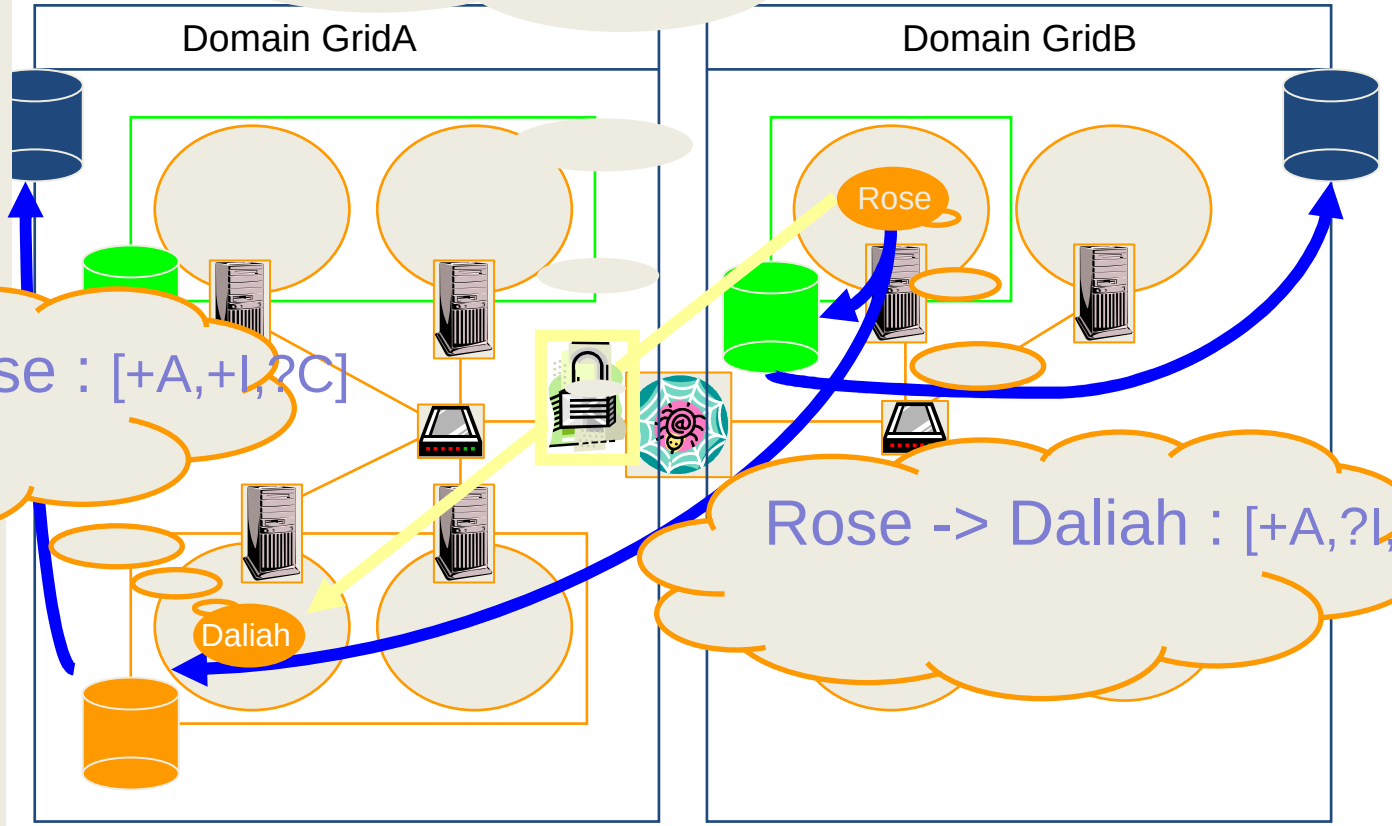
VN1

VN2

JVM

Negotiated Policy:
Rose -> Daliah : [+A,+I,+C]

Method call :
- other VN
- other domain
From
Rose --> Daliah



Policy rules databases

VN1

VN2

JVM

Adaptive Security

Dynamic setting of the security attributes

Dynamic negotiation between different:

- Domain and Sub-domain
- Virtual Nodes
- Active Objects

on JVMs on different Machines

Security summary

ProActive Security Features

- Authentication of users and applications (PKI X 509 certificate)
- Authentication, Integrity and Confidentiality of communications
- In XML deployment files, Not In Source
- Security model for mobile applications
- Dynamically negotiated policies, non-functional security
- Logical security representation : security is easily adaptable to the deployment

Perspectives:

- Group communication

Example of

ProActive

Applications

DEMO: Appli. with the IC2D monitor

1. C3D : Collaborative 3D renderer in //

a standard ProActive application

2. Penguin

a mobile agent application

IC2D: Interactive Control & Debug for Distribution

works with any ProActive application

Features:

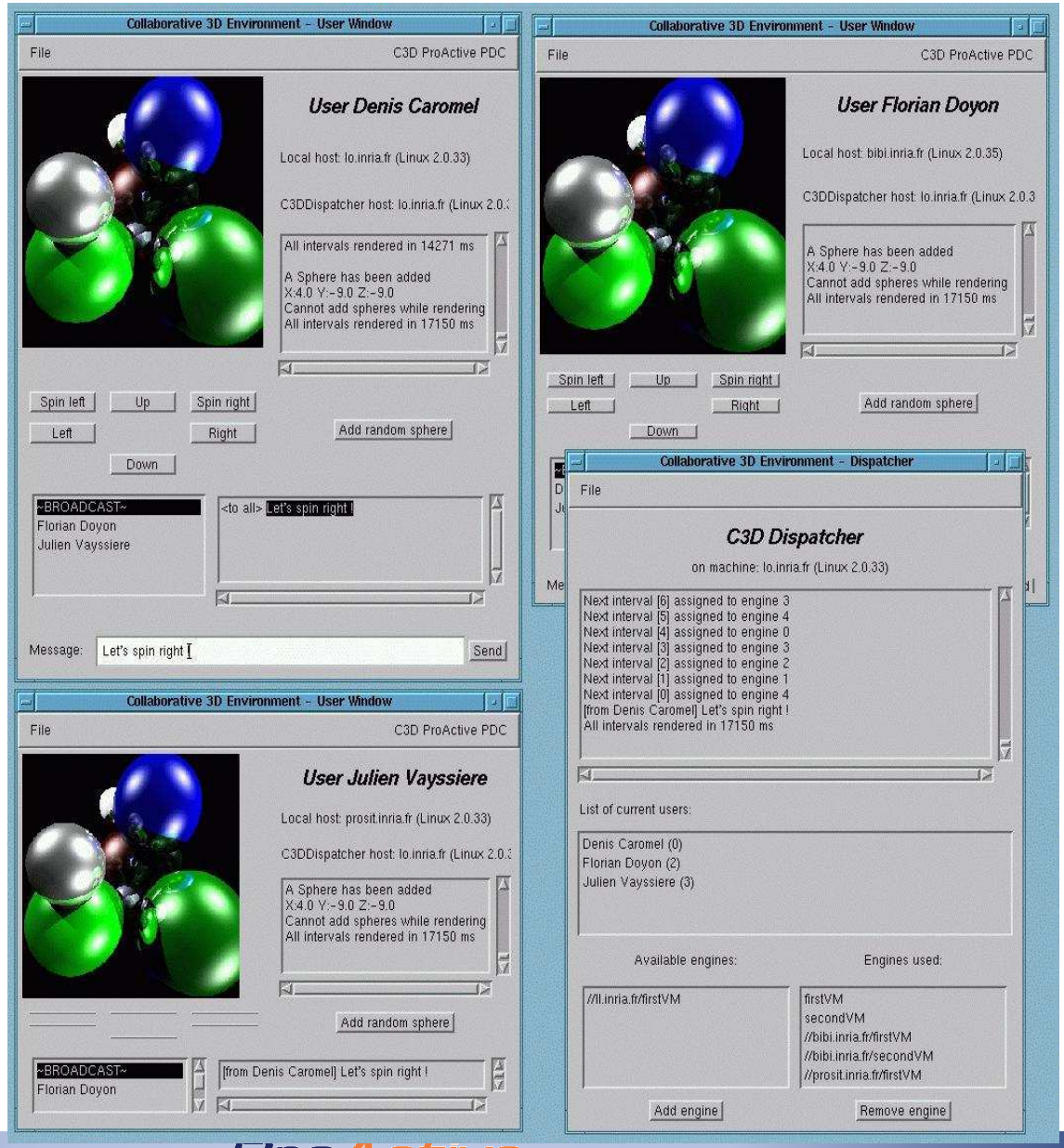
Graphical and Textual visualization

Monitoring and Control

Benchmarks and Optimizations

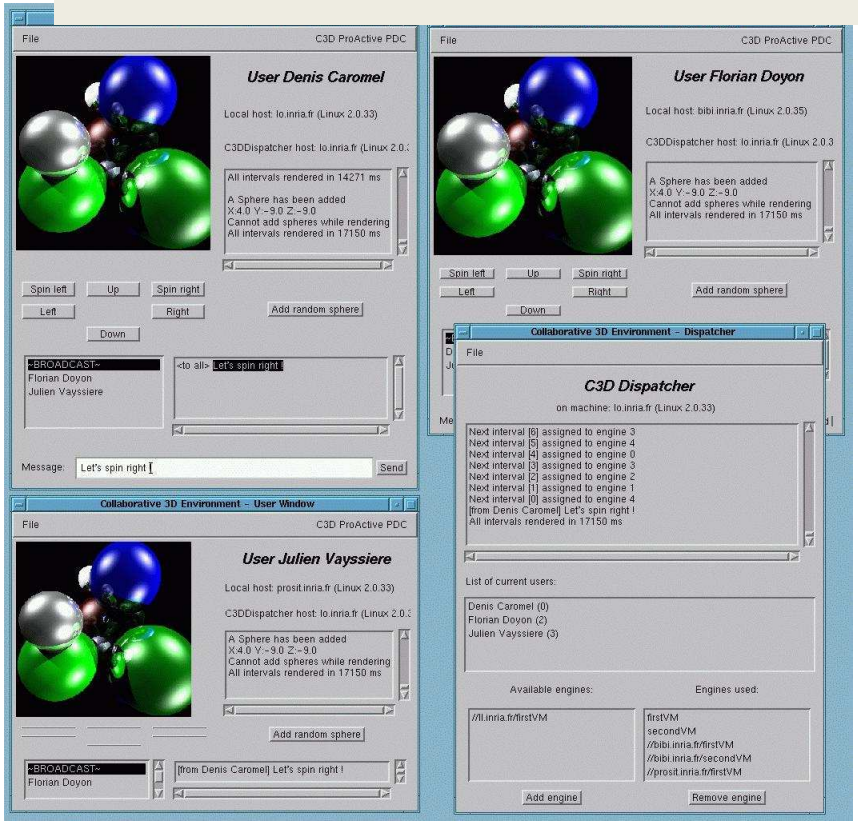
C3D: distributed-//-collaborative

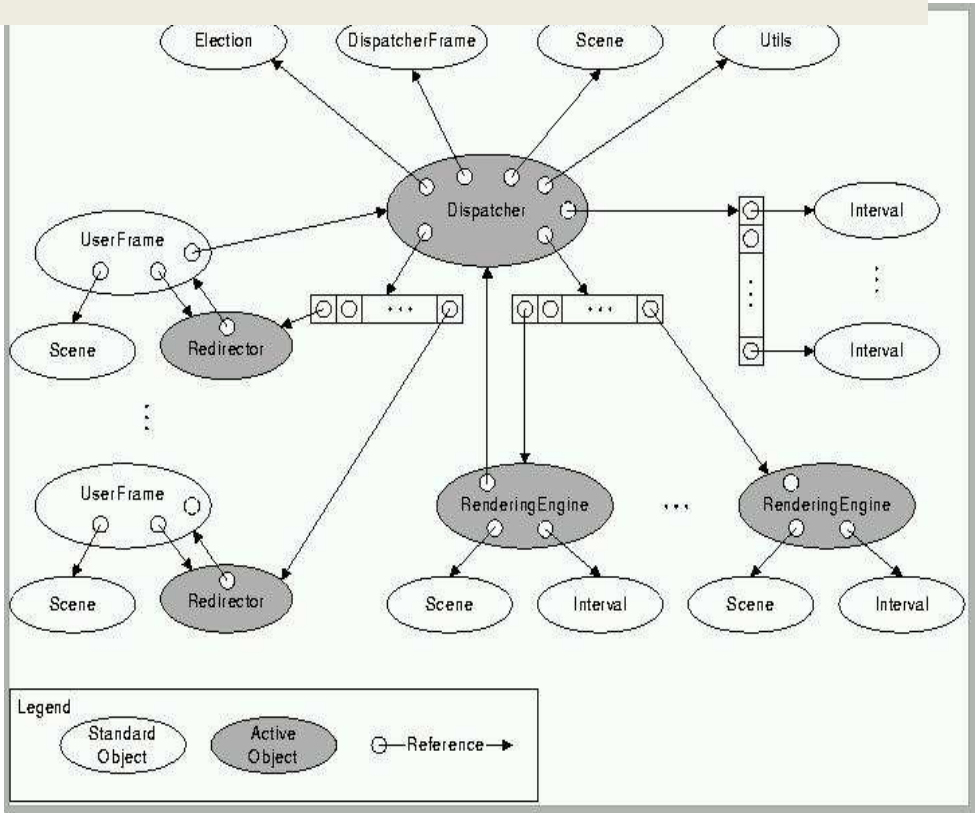
Collaborative 3D, Rendering in //, Application mobility



C3D: distributed-//-collaborative

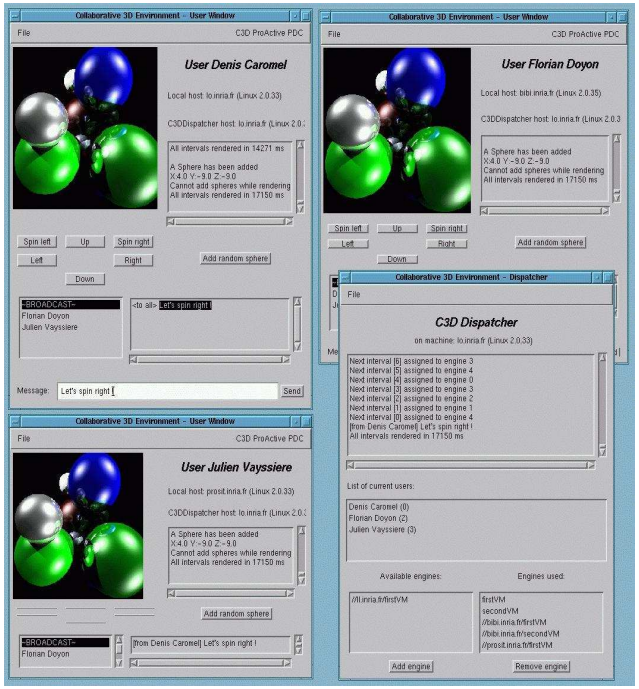
Collaborative 3D, Rendering **in //,** Application **mobility**

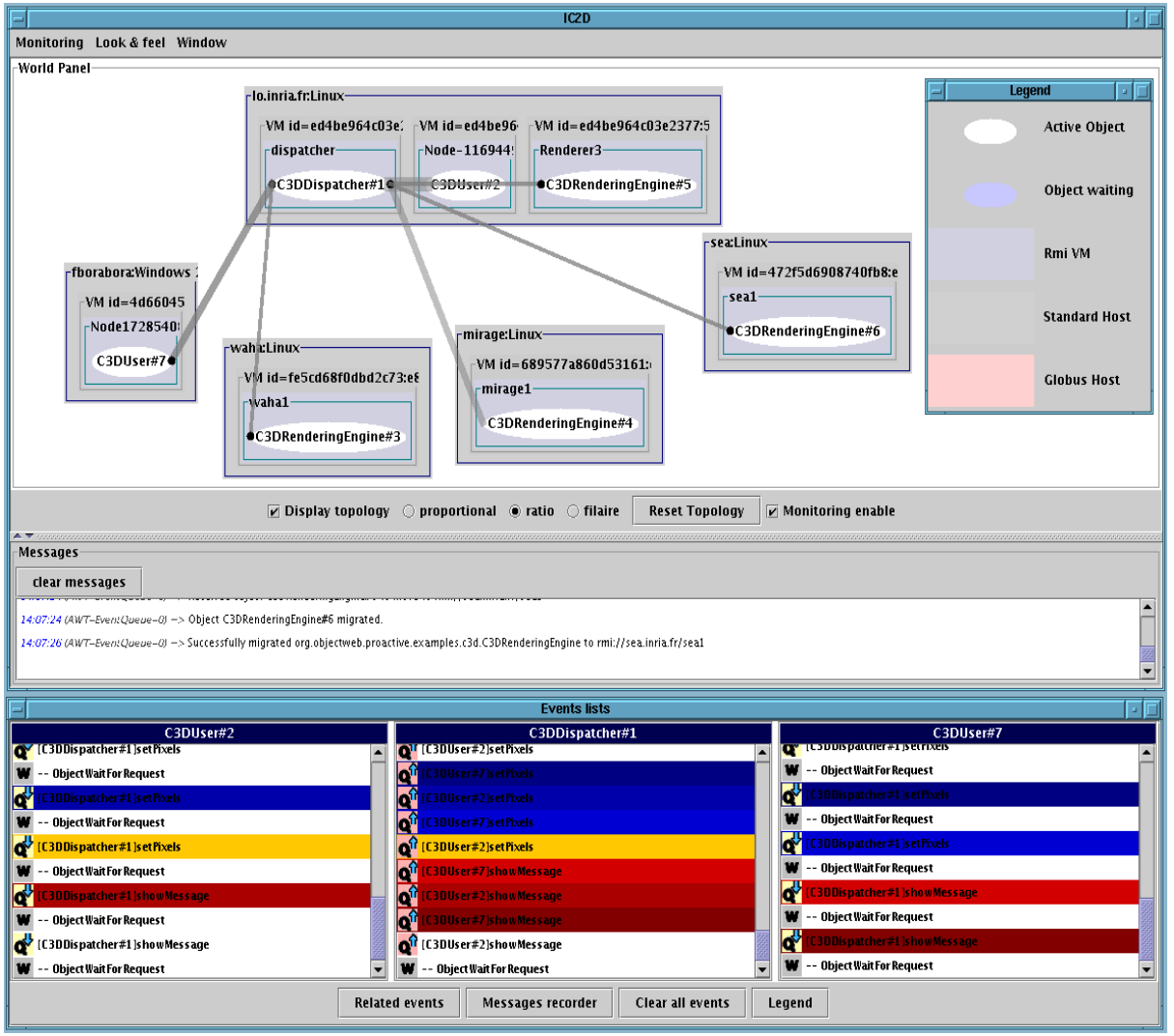




ProActive
Programming, Composing, Deploying on the Grid

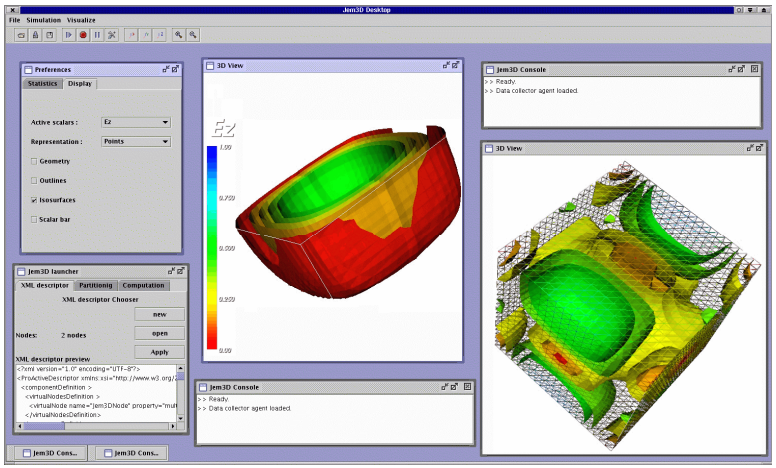
C3D Monitoring: graphical and textual com.



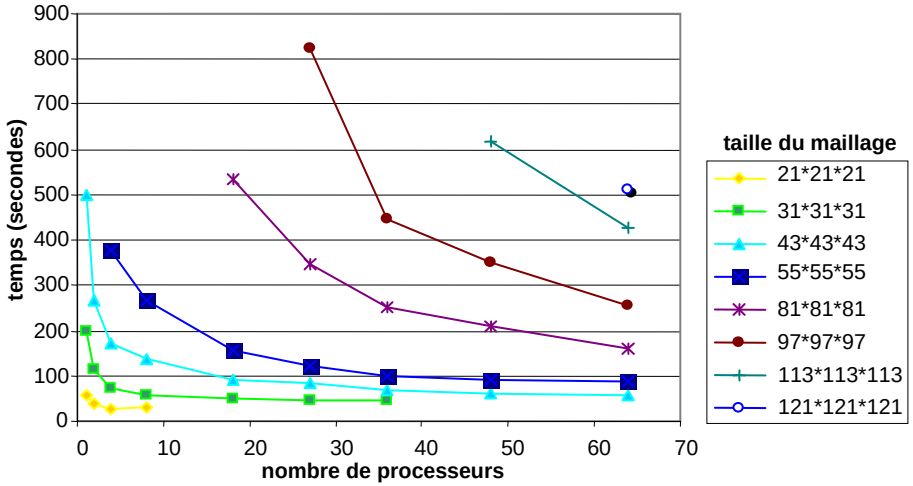


JEM 3D : Java 3D Electromagnetism

together with Said El Kasmi, Stéphane Lanteri (caiman)



temps d'exécution de la boucle principale (sur cluster)



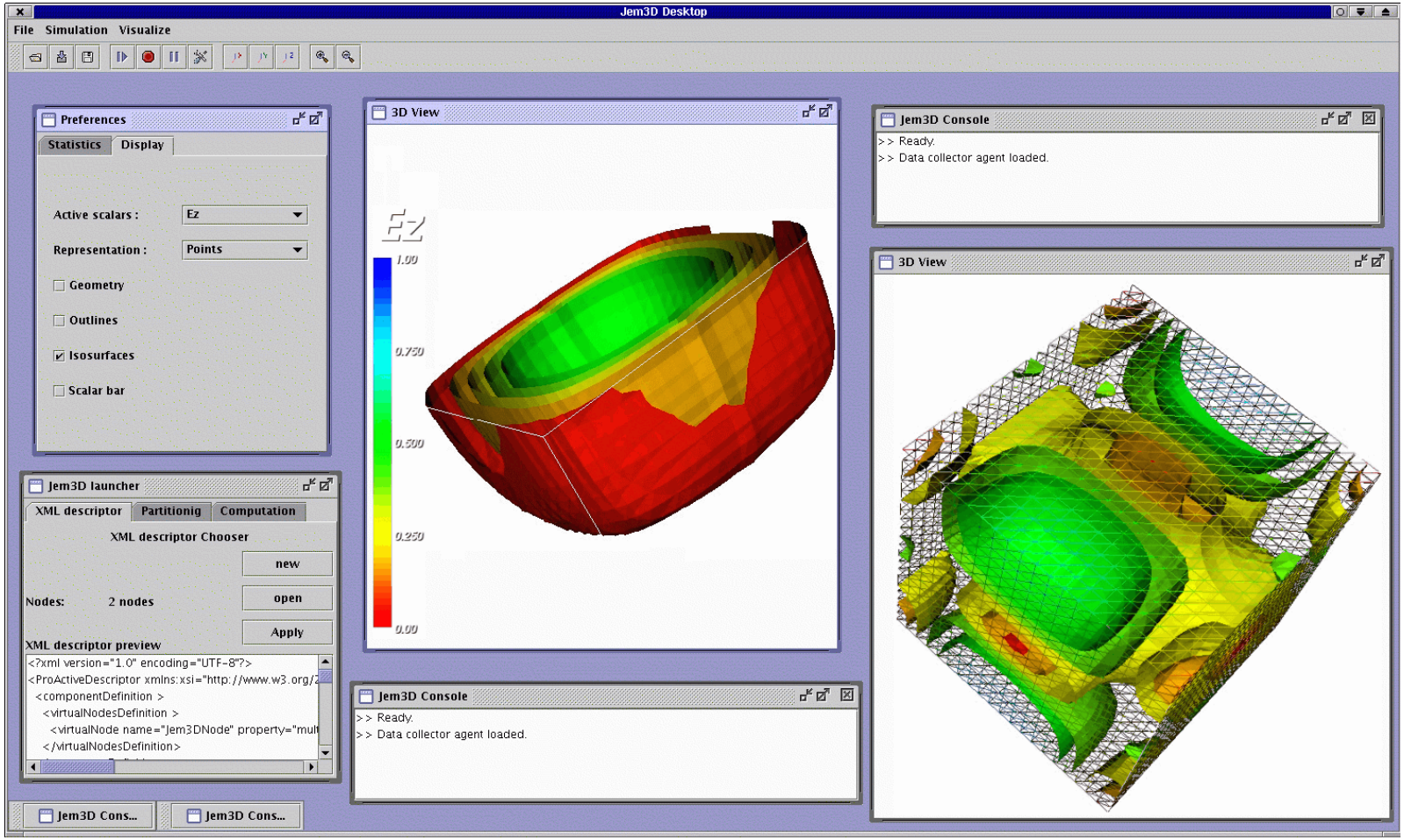
Maxwell 3D equation solver, Finite Volume Method (FVM)

Pre-existing **Fortran MPI** version: **EM3D** (CAIMAN team @ INRIA)

Up to 294 machines at the same time (Intranet and cluster)

Large data sets: 150x150x150 (100 million facets)

Jem3d


Jem3D Desktop
File Simulation Visualize
Preferences
Statistics Display
Active scalars : Ez
Representation : Points
Geometry
Outlines
Isosurfaces
Scalar bar
3D View
1.00
0.750
0.500
0.250
0.00
Jem3D Console
>> Ready.
>> Data collector agent loaded.
3D View
Jem3D launcher
XML descriptor Partitionig Computation
XML descriptor Chooser
new
Nodes: 2 nodes
open
Apply
XML descriptor preview
<?xml version="1.0" encoding="UTF-8"?>
<componentDefinition >
<virtualNodesDefinition >
<virtualNode name="Jem3DNode" property="mul
</virtualNodesDefinition>
Jem3D Console
>> Ready.
>> Data collector agent loaded.
Jem3D Cons...
Jem3D Cons...

Jem3D: Overall Objectives

3D Electromagnetic Application:

Visualize the radar reflection of a plane, medicine on head, etc.

Pre-existing **Fortran MPI** version: **EM3D**

Jem3D:

Sequential object-oriented design, modular and extensible **(in Java)**

Sequential version can be smoothly distributed:

--> keeping structuring and object abstractions

Efficient distributed version, large domains, Grid env.

Jem3D: Geometry definition

A Generic Numerical Method:

Finite Volume Method (FVM)

(vs. Finite Element Methods)

Calculation of unknowns as average of Control Volume

(vs. Vertices of the mesh for FEM)

Valid on structured, unstructured, or hybrid meshes

Computation:

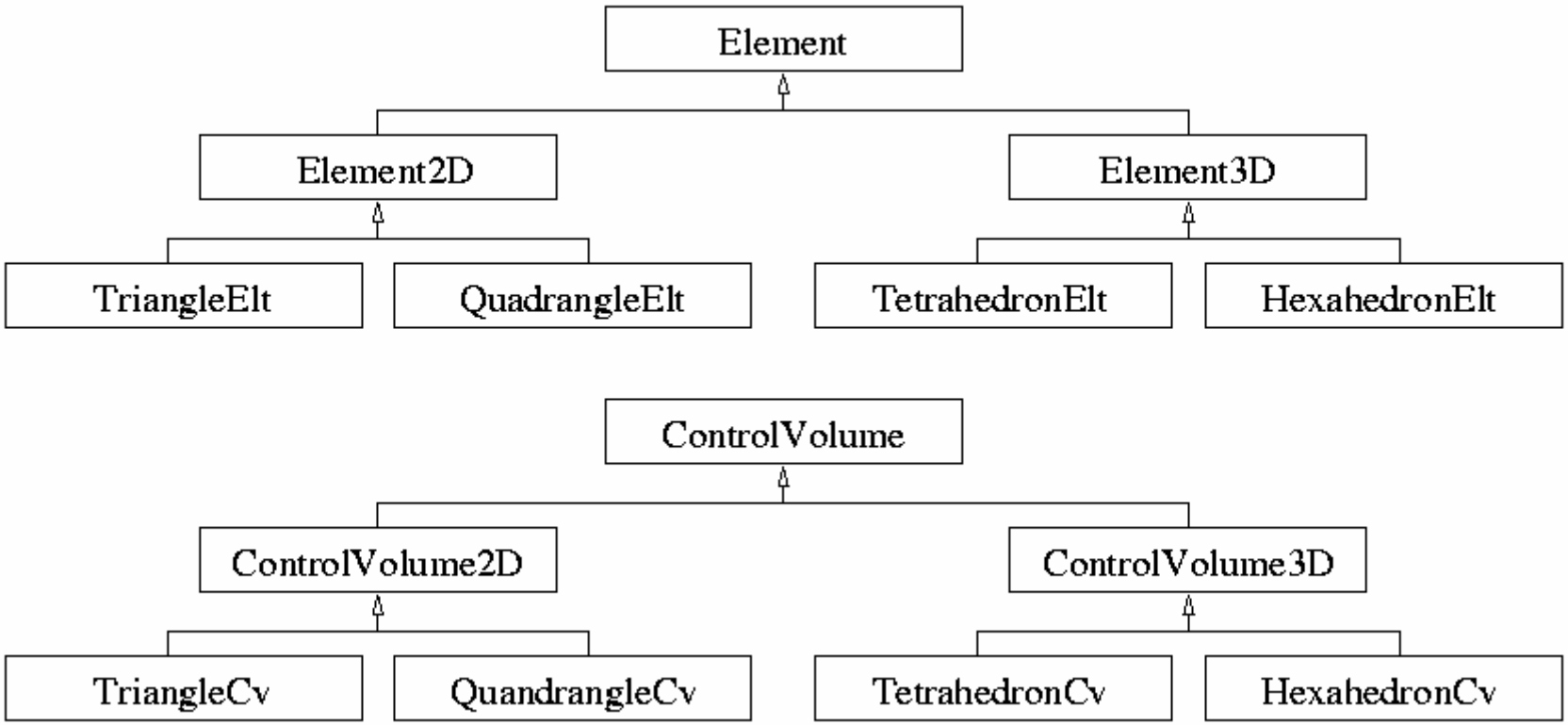
a flux balance through the boundary of Control Volume (M, E, EM, loop)

Example, and benchmarks here:

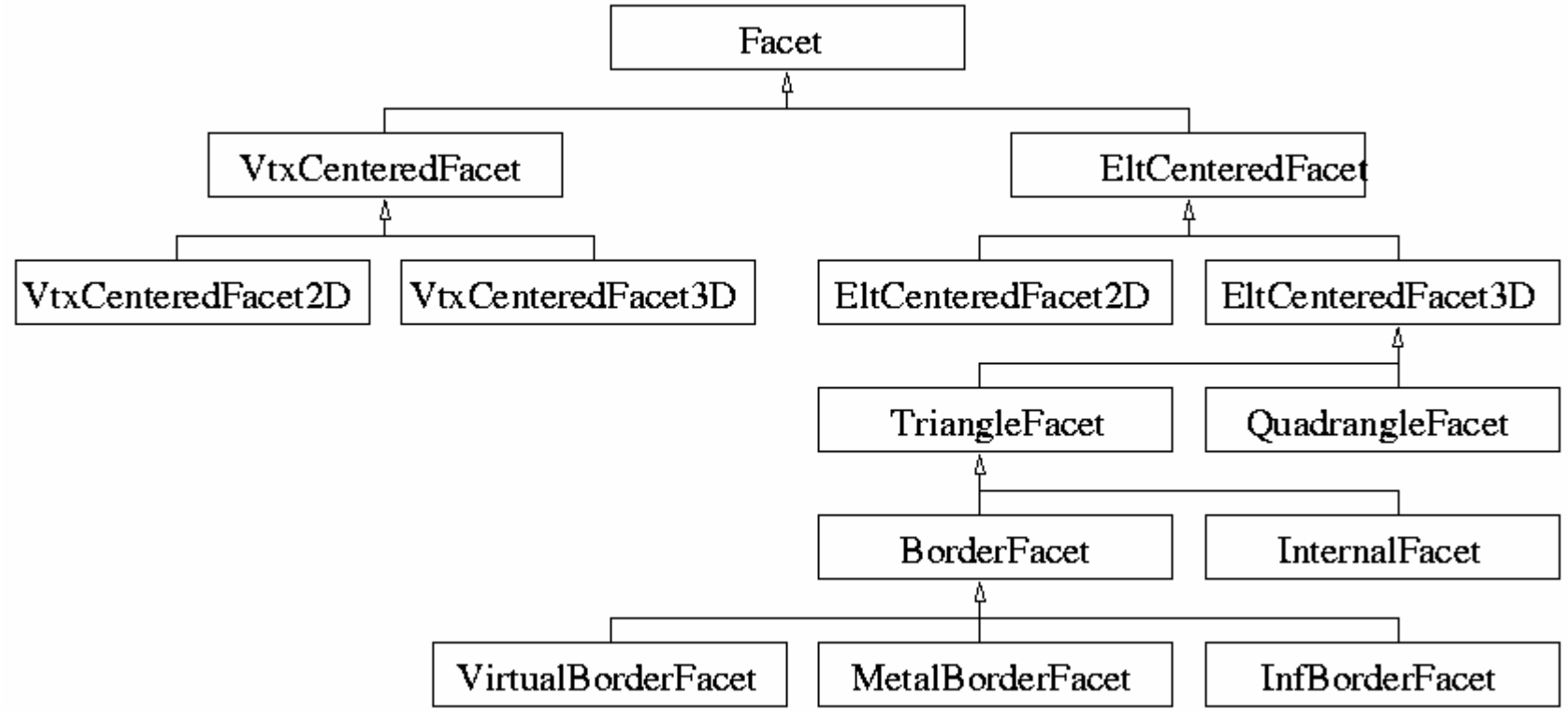
Control Volume = Tetrahedron

Facet = Triangle

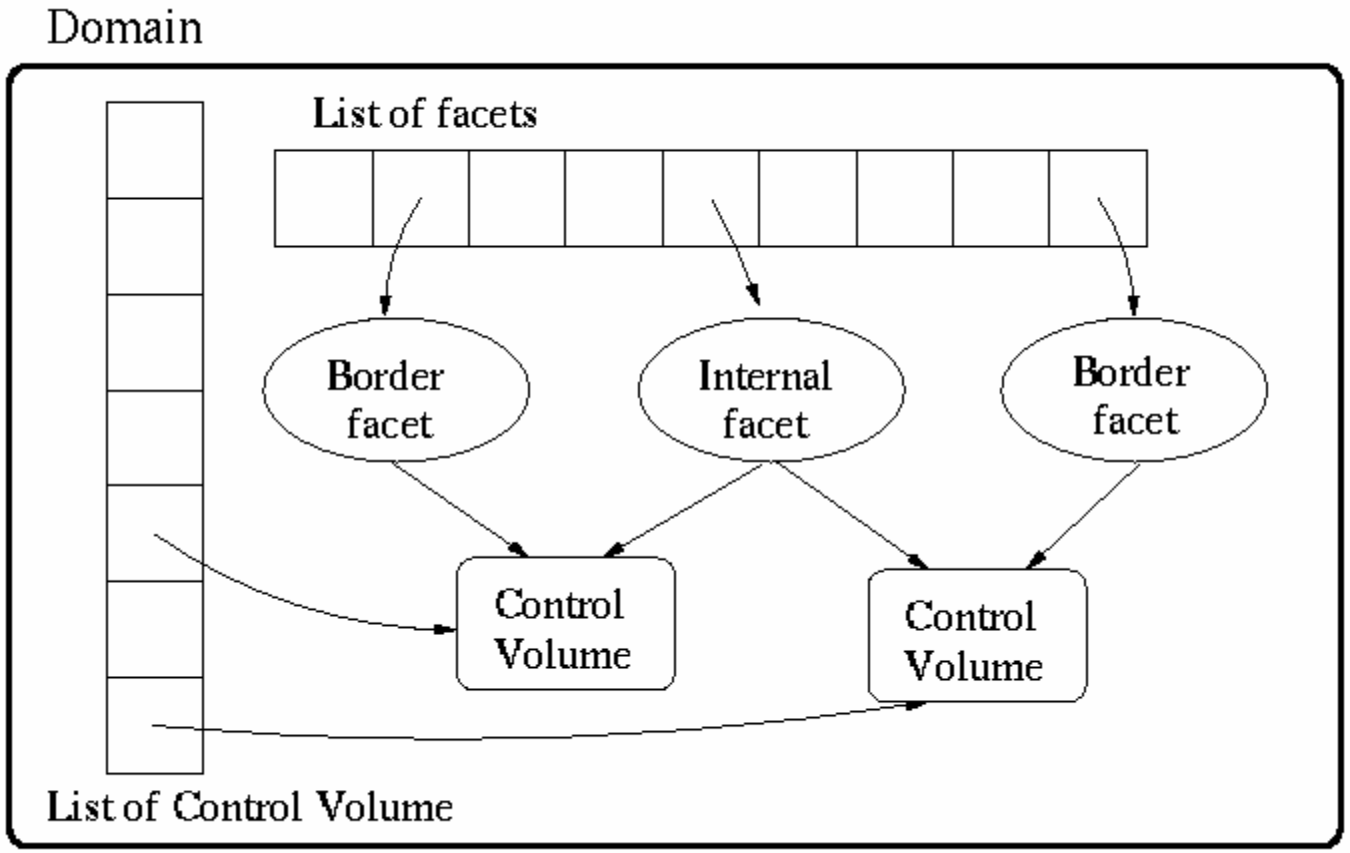
Jem3D: Control Volume in 2D and 3D



Jem3D: Facets in 2D and 3D

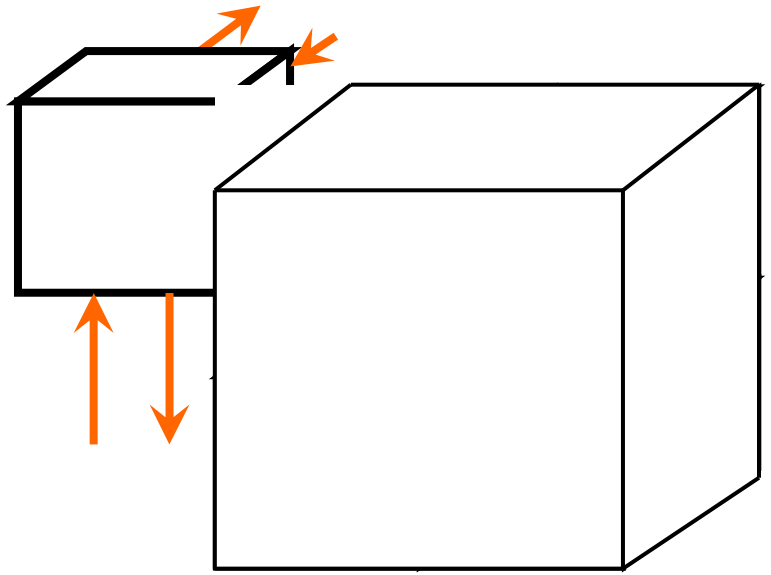


JEM 3D : Architecture of the sequential version

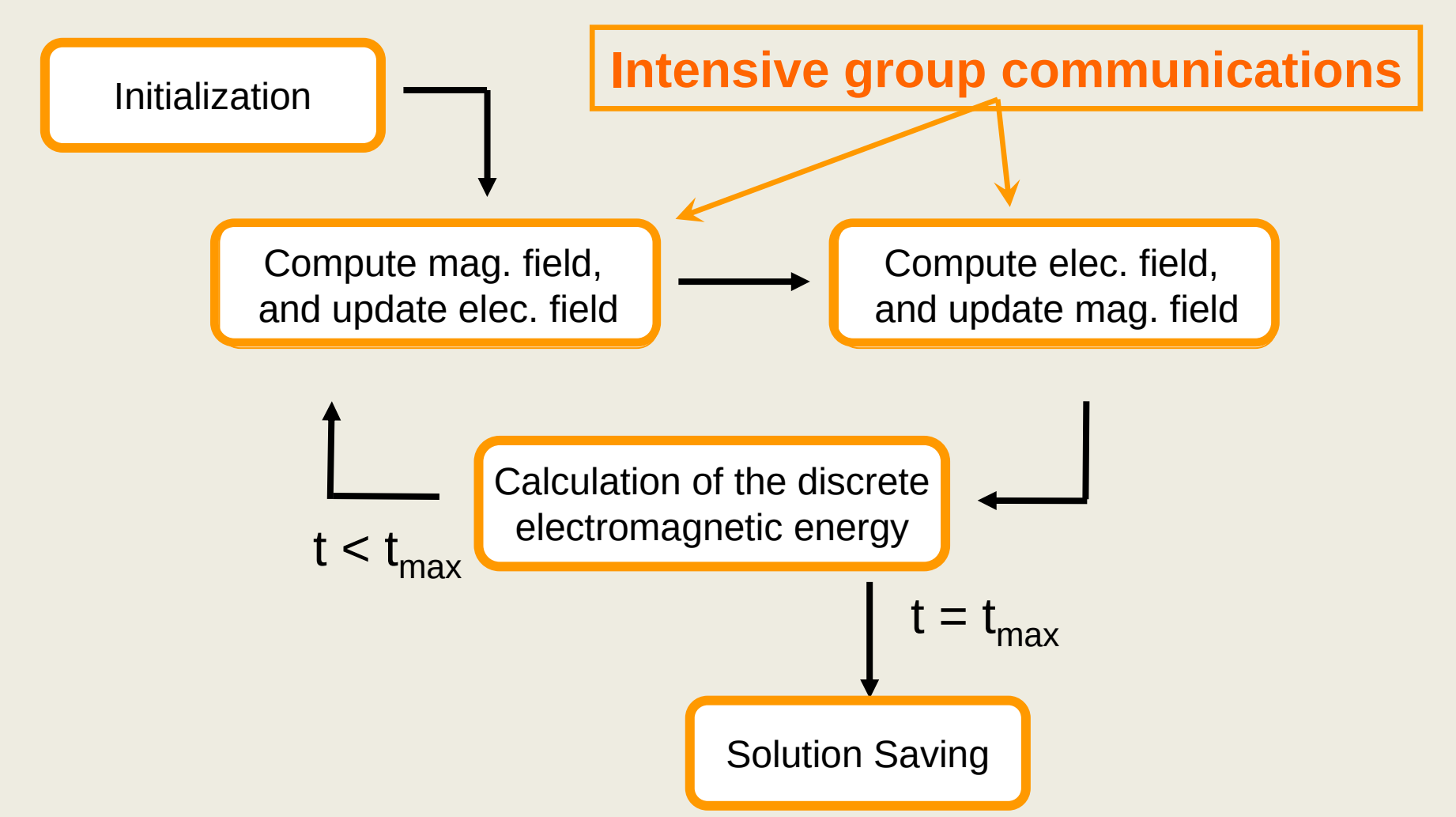


Communication based on groups

Groups allow sub-domains communication



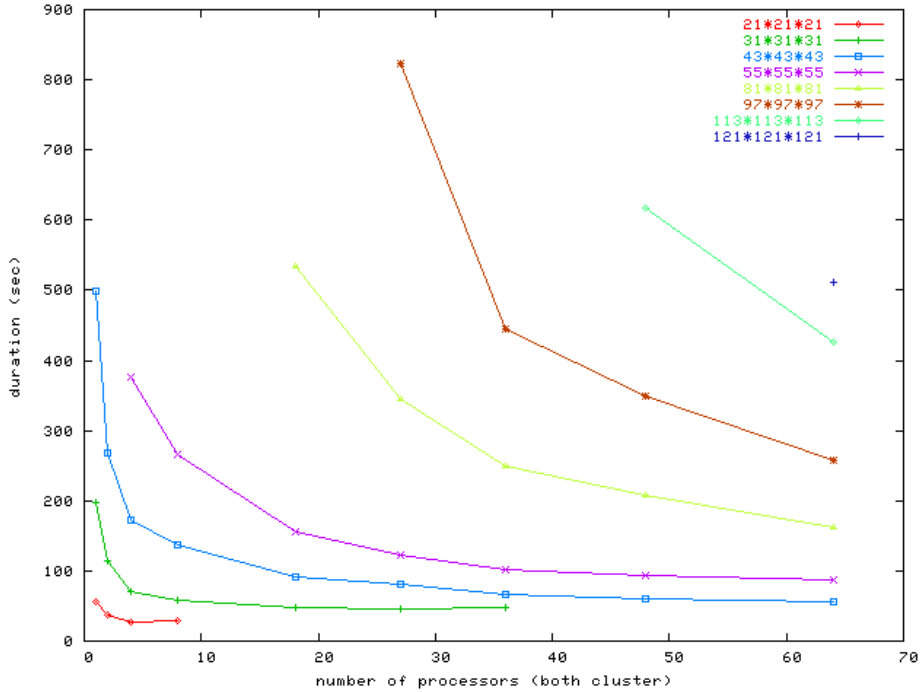
Algorithm of Jem3D

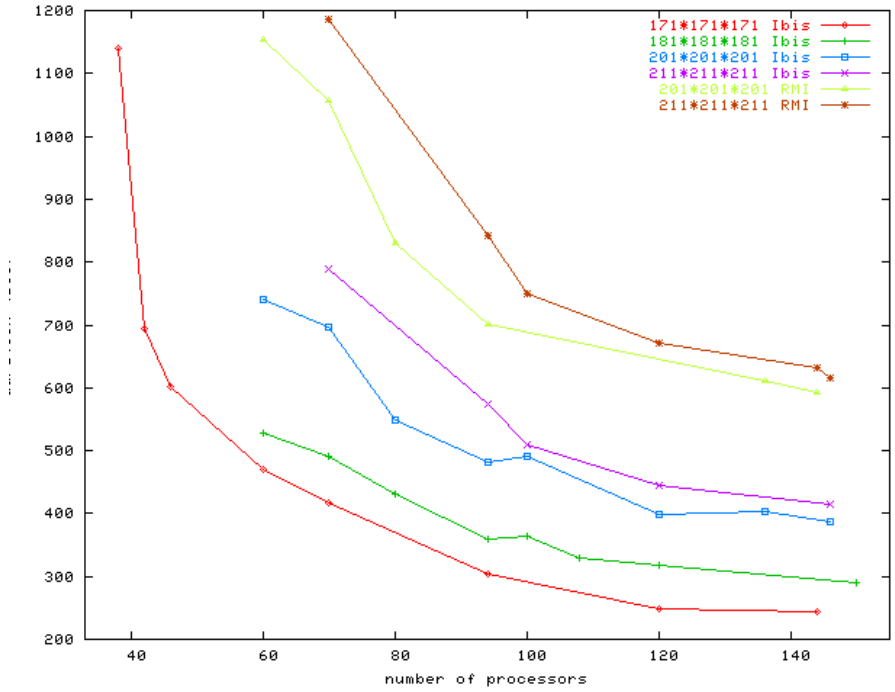


Jem3D

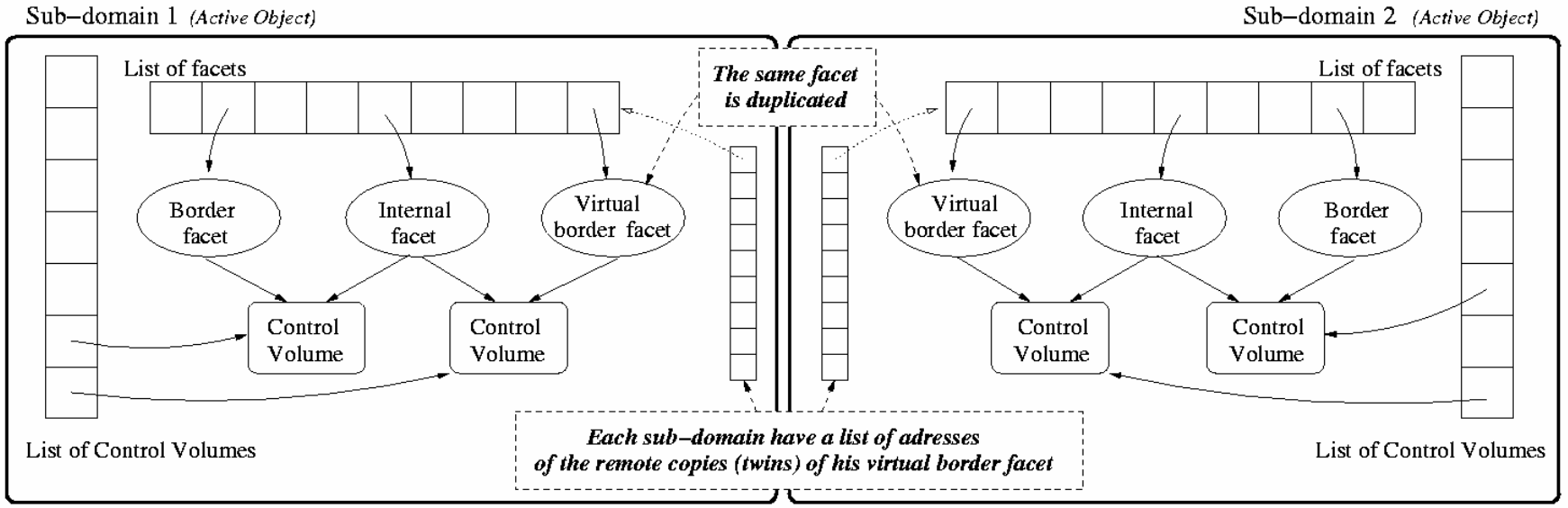
INRIA Sophia's cluster
DAS-2

Computation of the propagation of an electromagnetic wave in a cubic metallic cavity (standard test case)





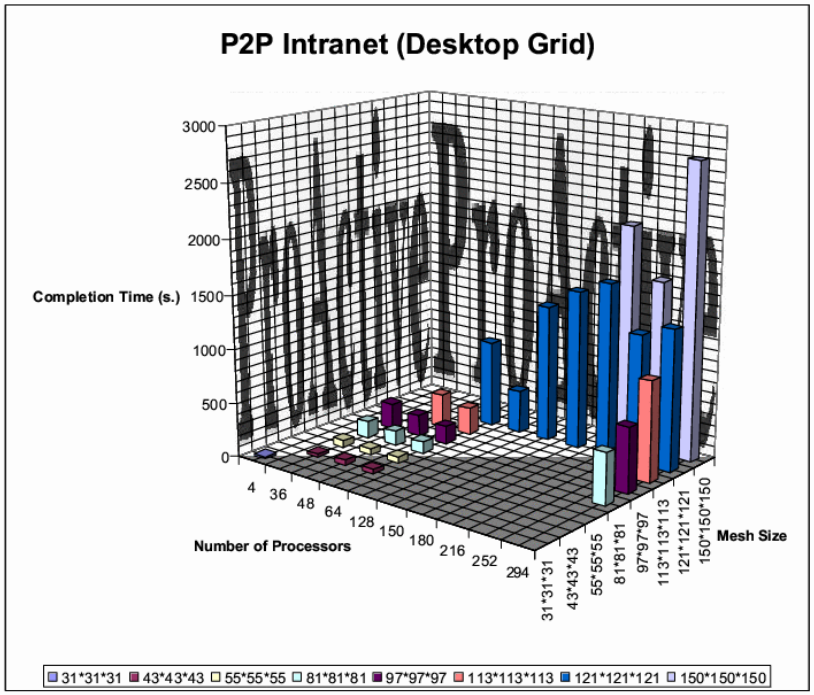
JEM 3D : Architecture of the ProActive distributed version



An Object-Oriented SPMD program

JEM 3D : Benchmarks on desktop machines

ProActive: Jem3D Benchmarks



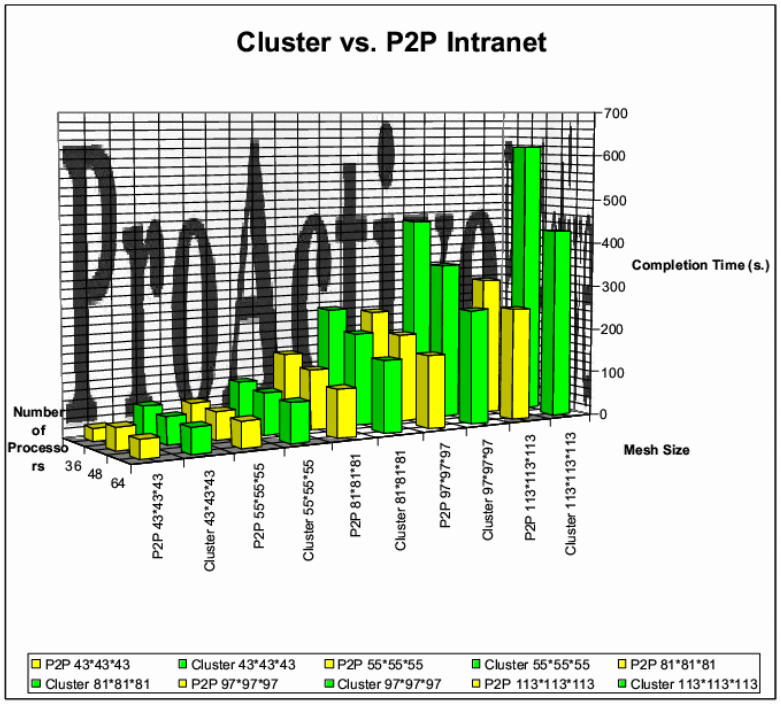
Jem3D is an object-oriented time domain finite volume solver for the 3D Maxwell equations

ProActive
http://www.inria.fr/oasis/ProActive

Java for Parallel, Distributed, Concurrent, Secured, and Mobile Computing

JEM 3D : Cluster vs. Intranet P2P

ProActive: Jem3D Benchmarks



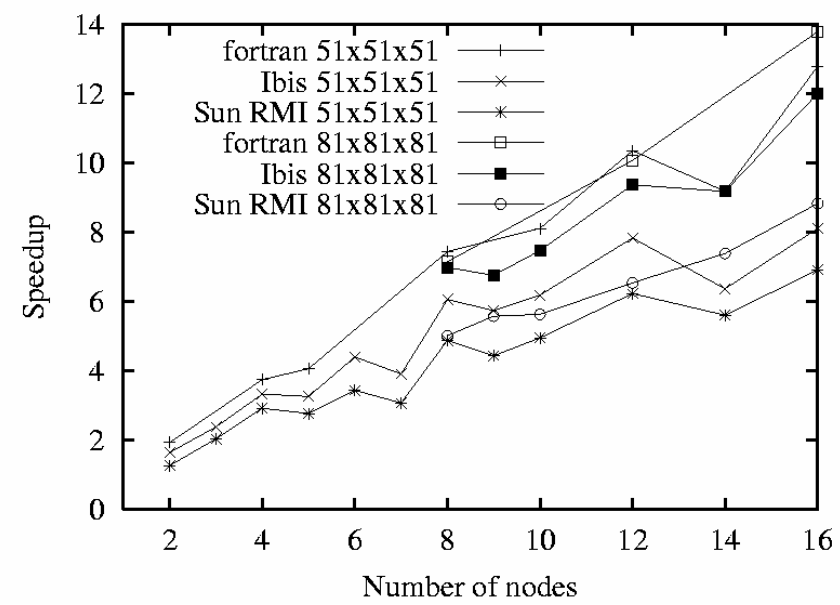
Cluster: 16 Pentium IV bi-xeon at 2 GHz / GigaEthernet & 16 bi-Pentium III at 933 MHz / FastEthernet

P2P: in average, Intranet desktop machines are changed every 3 years

ProActive
http://www.inria.fr/oasis/ProActive

Java for Parallel, Distributed, Concurrent, Secured, and Mobile Computing

JEM 3D : Recent Benchmarks



Grid experiment on 5 clusters (DAS 2):

Speed up of 100 on 150 machines

Seq. Java/Fortran: 2

Comparison:

Jem3D over

- ProActive/RMI Sun

- ProActive/RMI Ibis

Em3D in

- Fortran/MPI

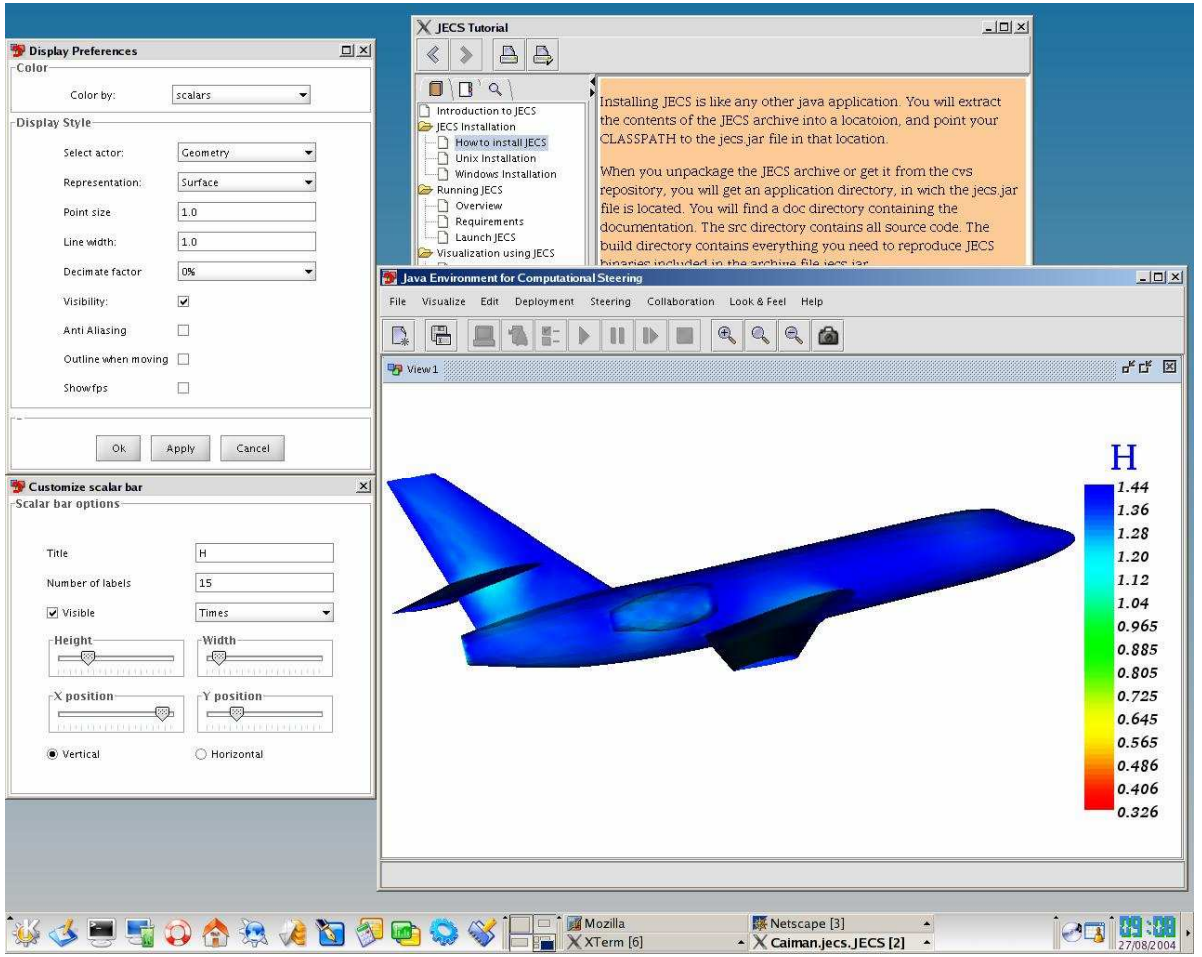
On 16 machines:

Fortran: 13.8

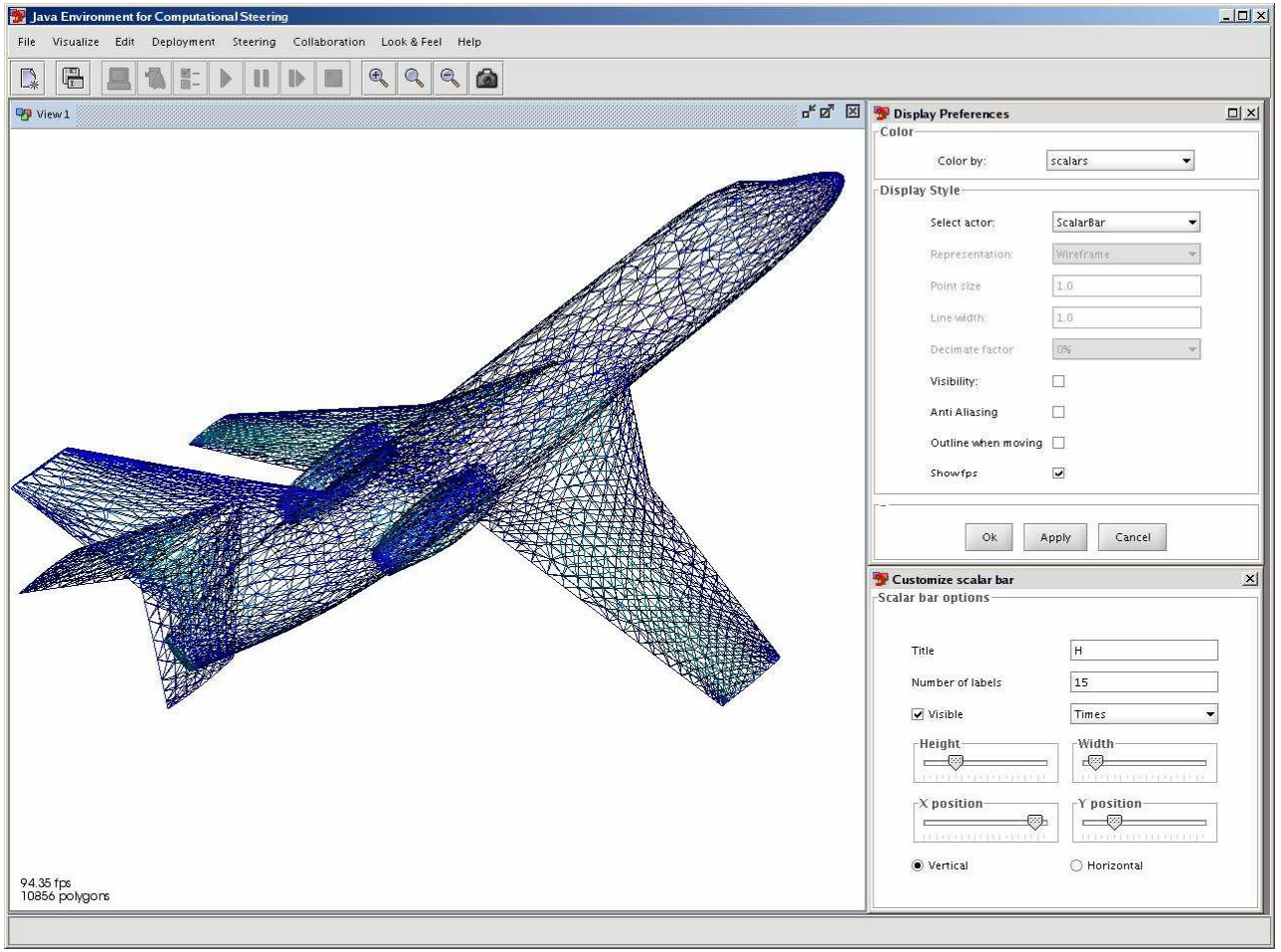
ProActive/Ibis: 12

ProActive/RMI: 8.8

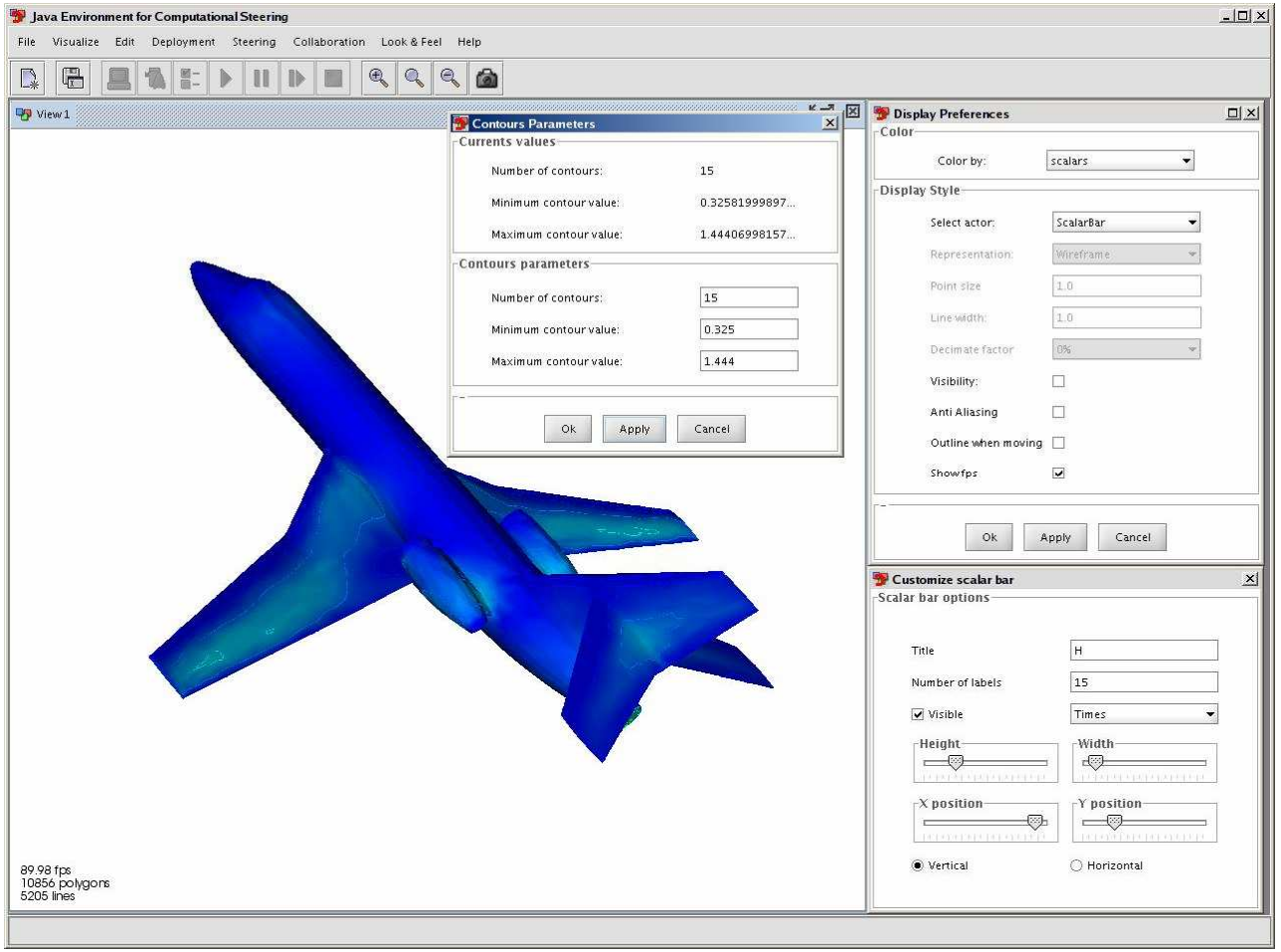
JECS : 3D Electromagnetism
A Generic Version of Jem3D



JECS : A Generic Version of Jem3D



JECS : A Generic Version of Jem3D



Beating Fortran ?

Current status:

Sequential Java vs. Fortran code: **2 times slower**

Large data sets in Java ProActive: **150x150x150 (100 million facets)**

Large number of machines: **up to 294 machines in Desktop P2P**

Speed up on **16 machines:**

- **Fortran: 13.8**
- **ProActive/Ibis: 12**
- **ProActive/RMI: 8.8**

Grid on 5 clusters (DAS 2): **Speed up of 100 on 150 machines**

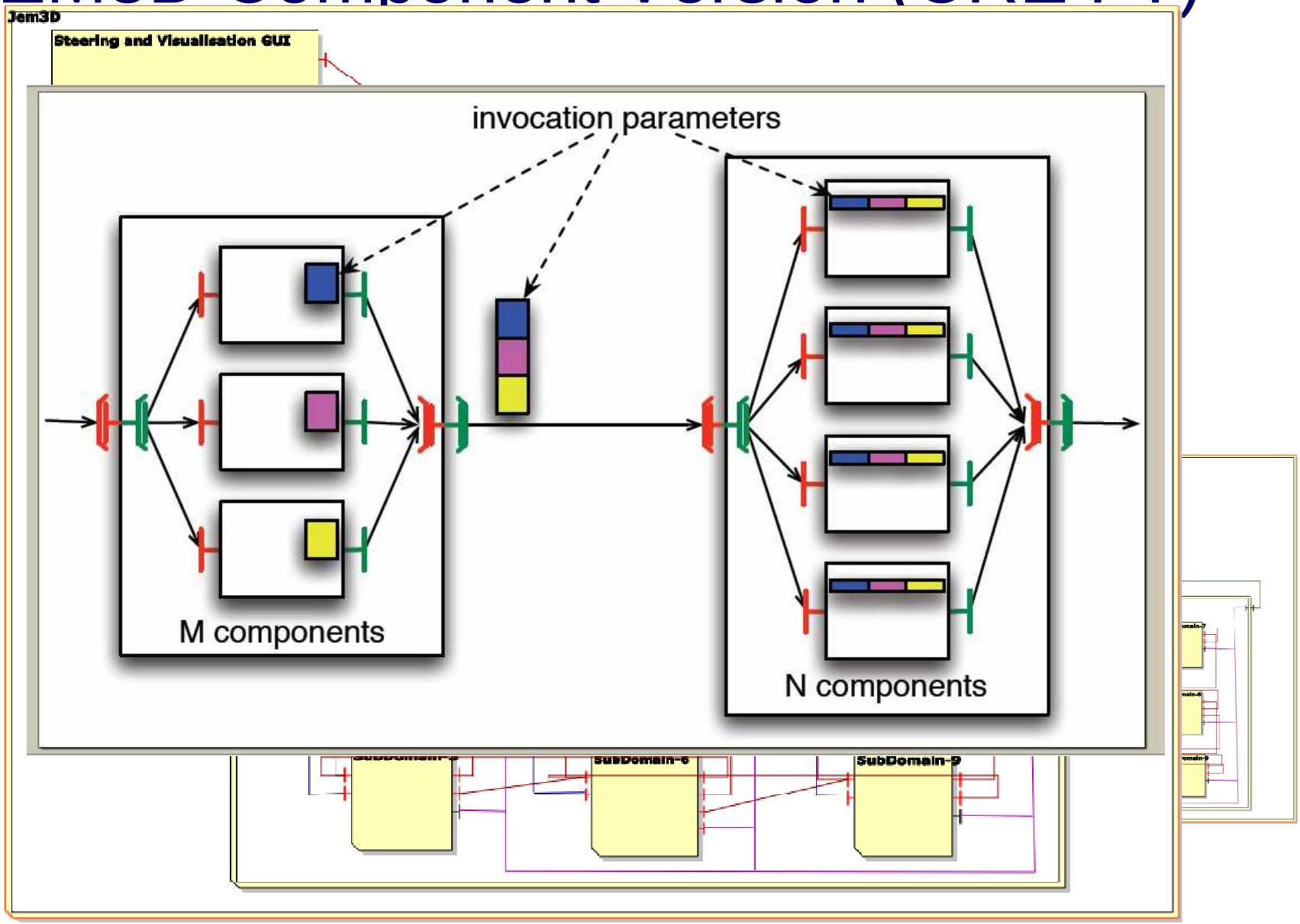
Fortran: **no more than 40 proc. …**

Beating Fortran MPI with Java ProActive? X/40 (14/16) = 2X/ n (100/150)

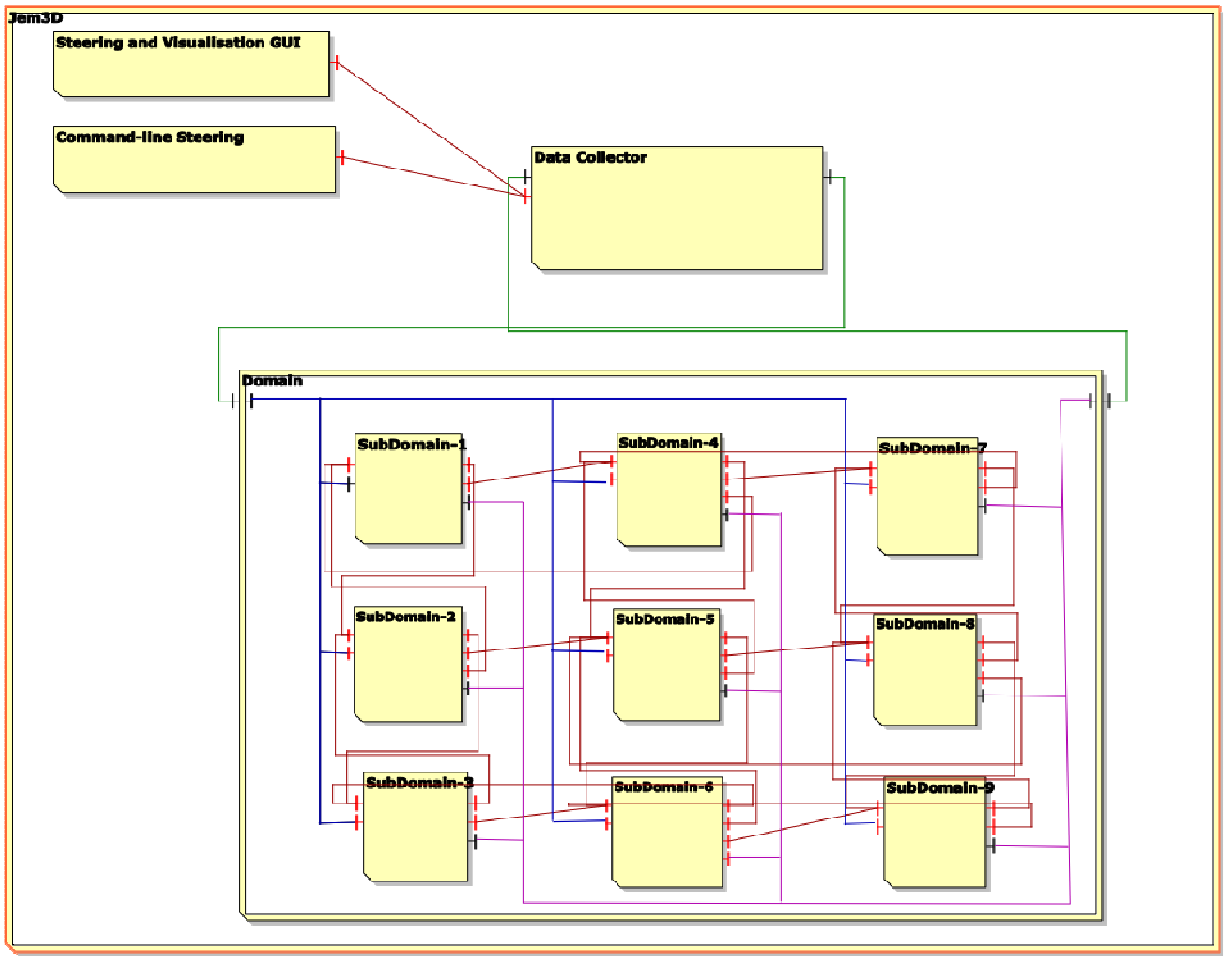
Yes, starting at 105 machines !

Component Applications

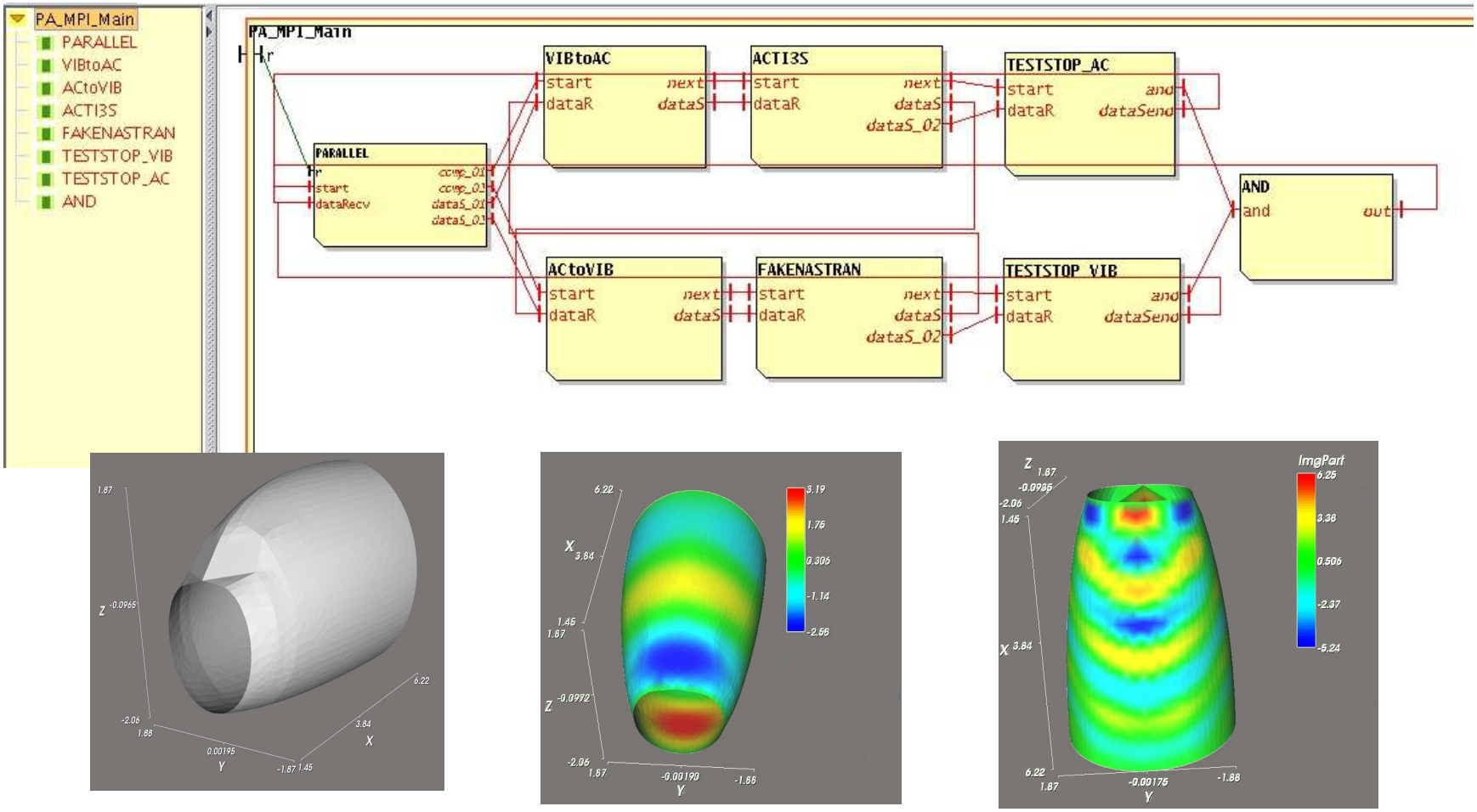
JEM3D Component Version (CRE FT)



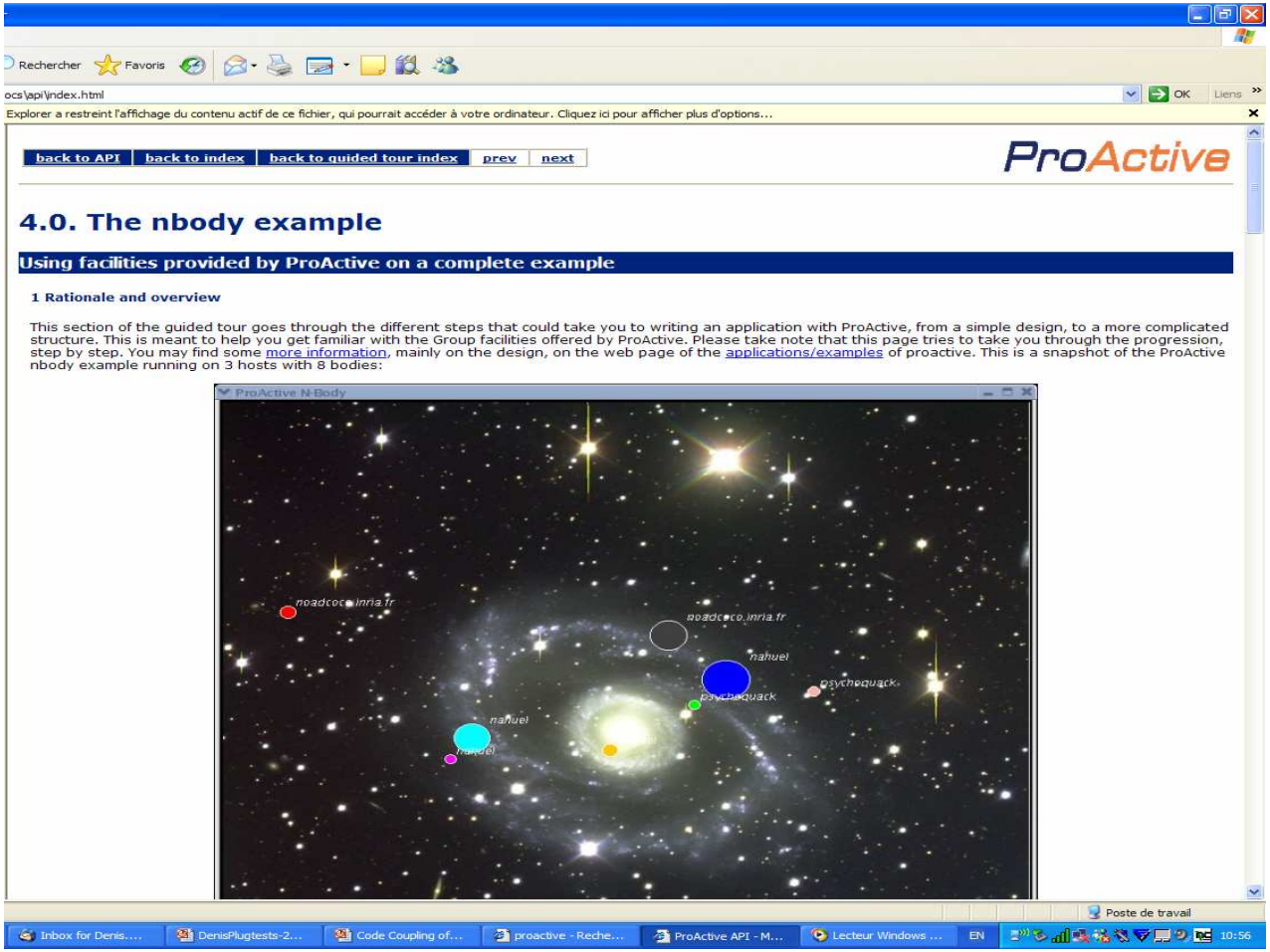
Jem3D Component Version



Code Coupling : Vibro Acoustic (courtesy of EADS)



N-Body Particules

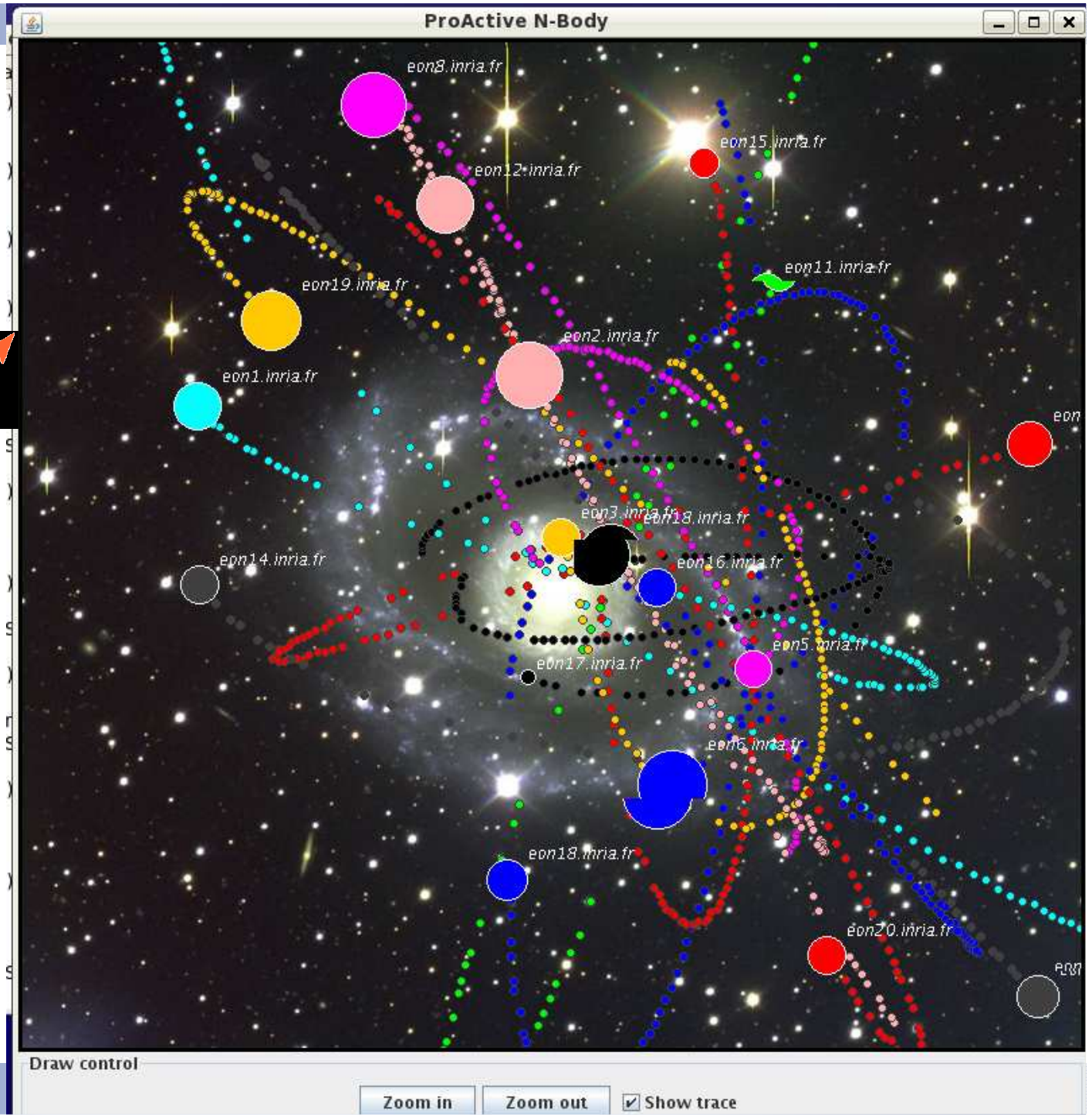


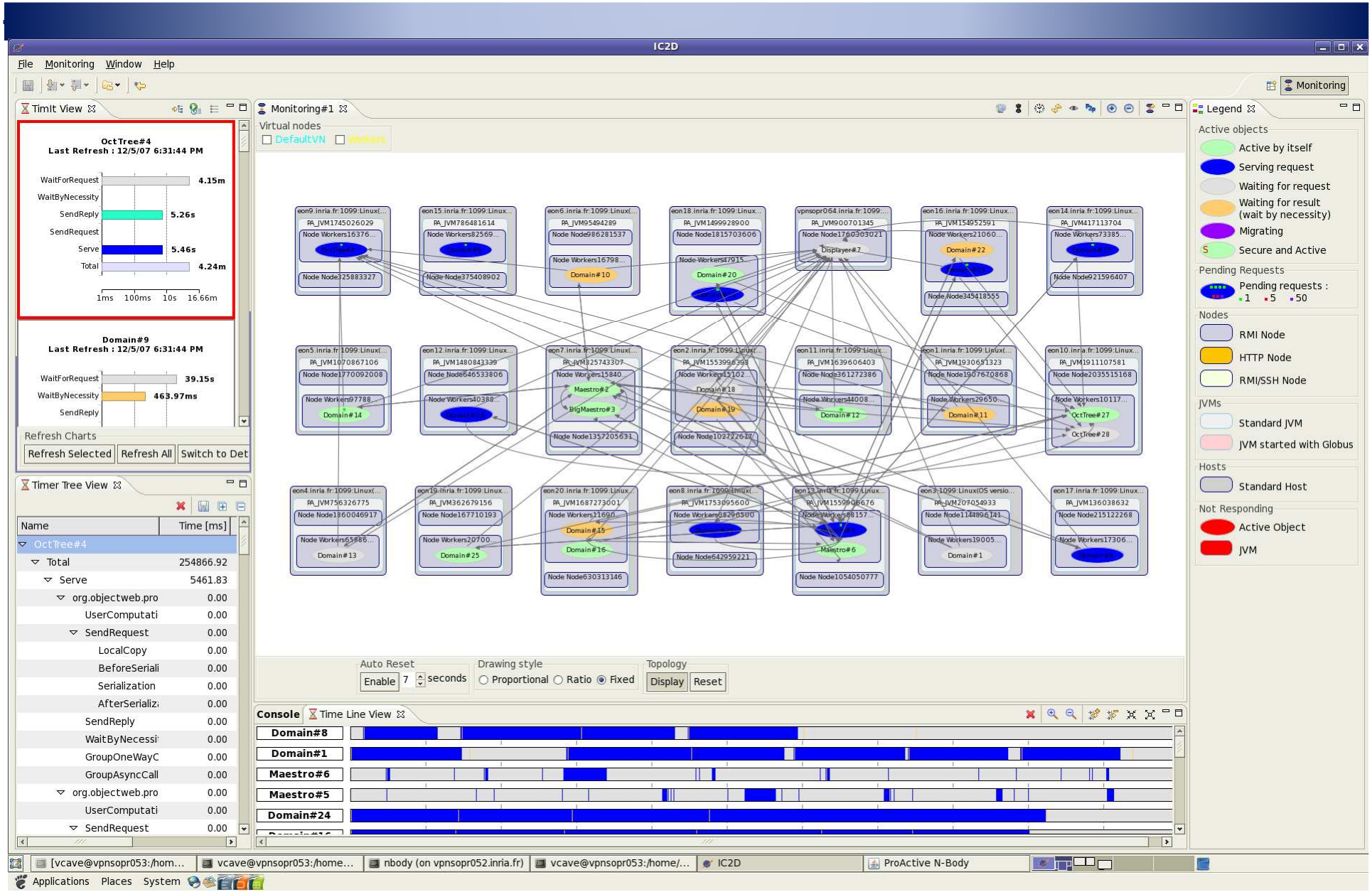
Demo

Upstairs

V. Cavé

B. Amédro





ObjectWeb
Open Source Middleware

ProActive

Programming, Composing, Deploying on the Grid

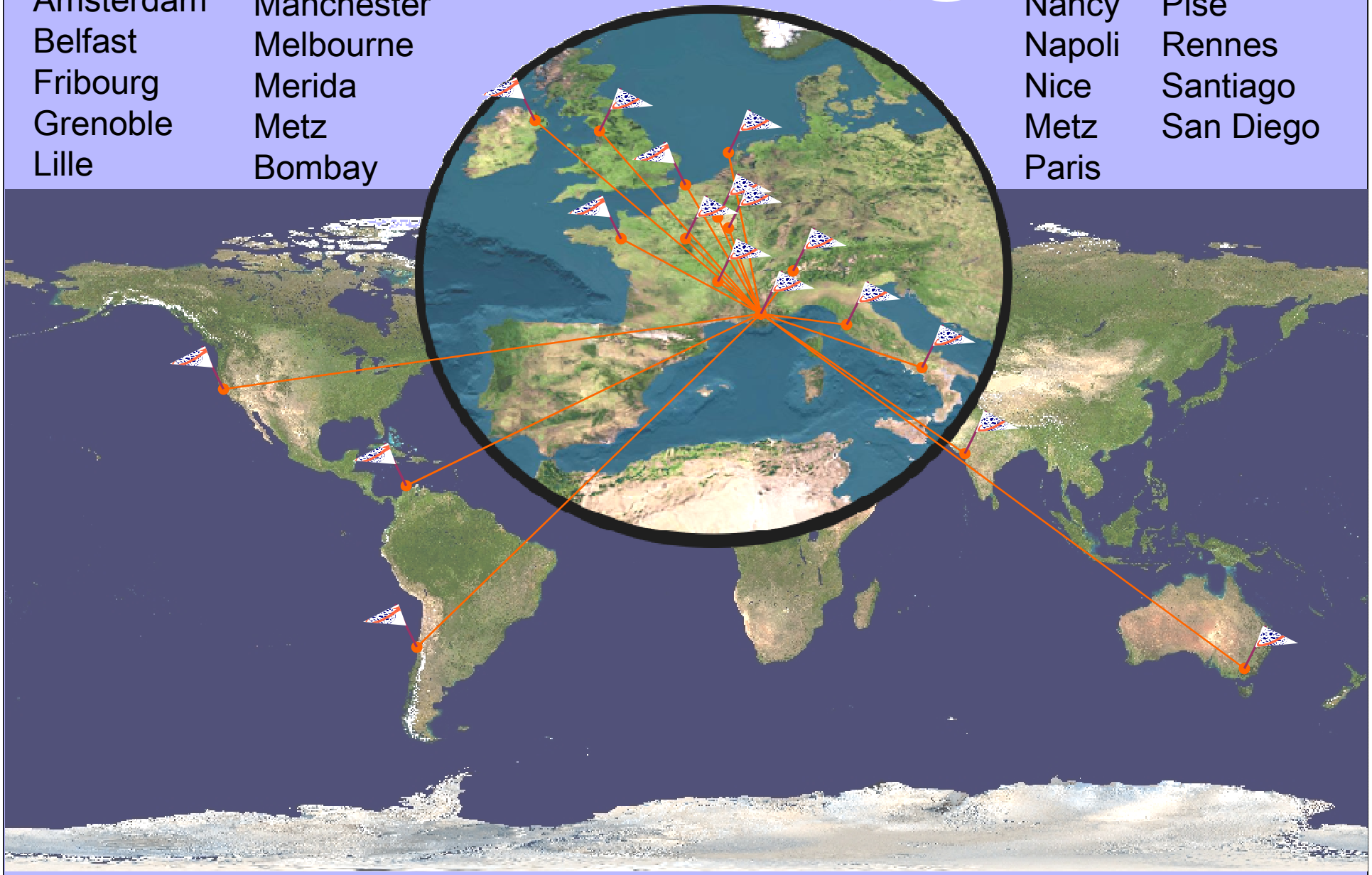
PLUGTESTS
THE INTEROPERABILITY SERVICE

Amsterdam
Belfast
Fribourg
Grenoble
Lille

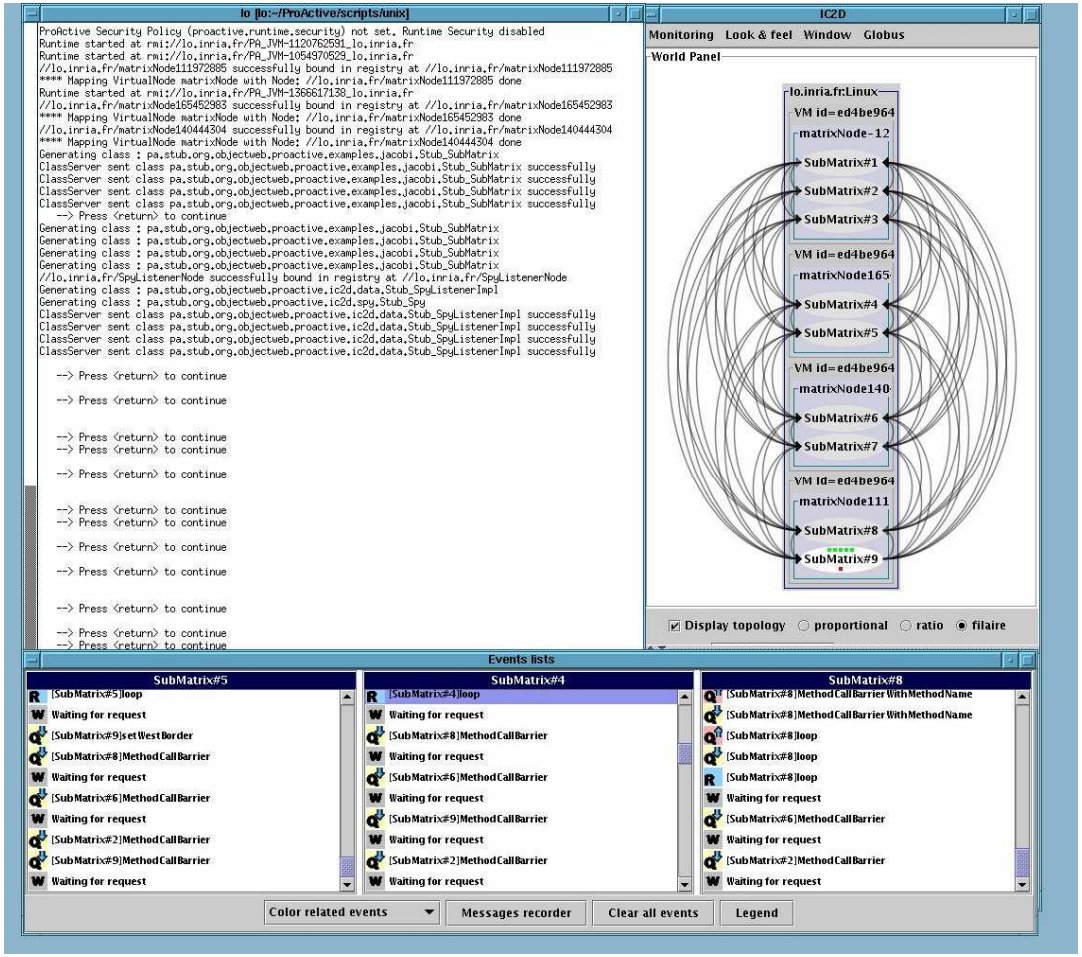
Manchester
Melbourne
Merida
Metz
Bombay

Nancy
Napoli
Nice
Metz
Paris

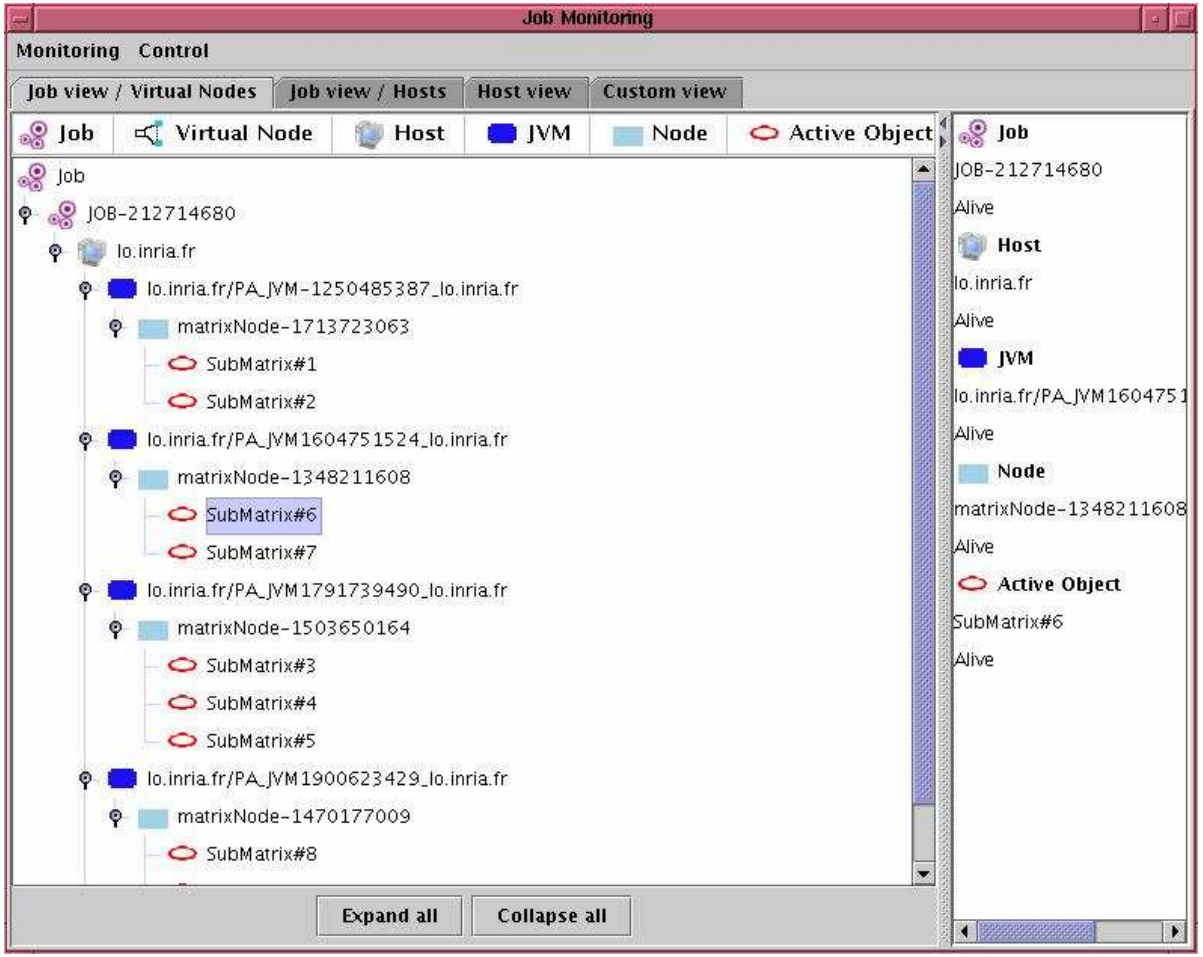
Pise
Rennes
Santiago
San Diego



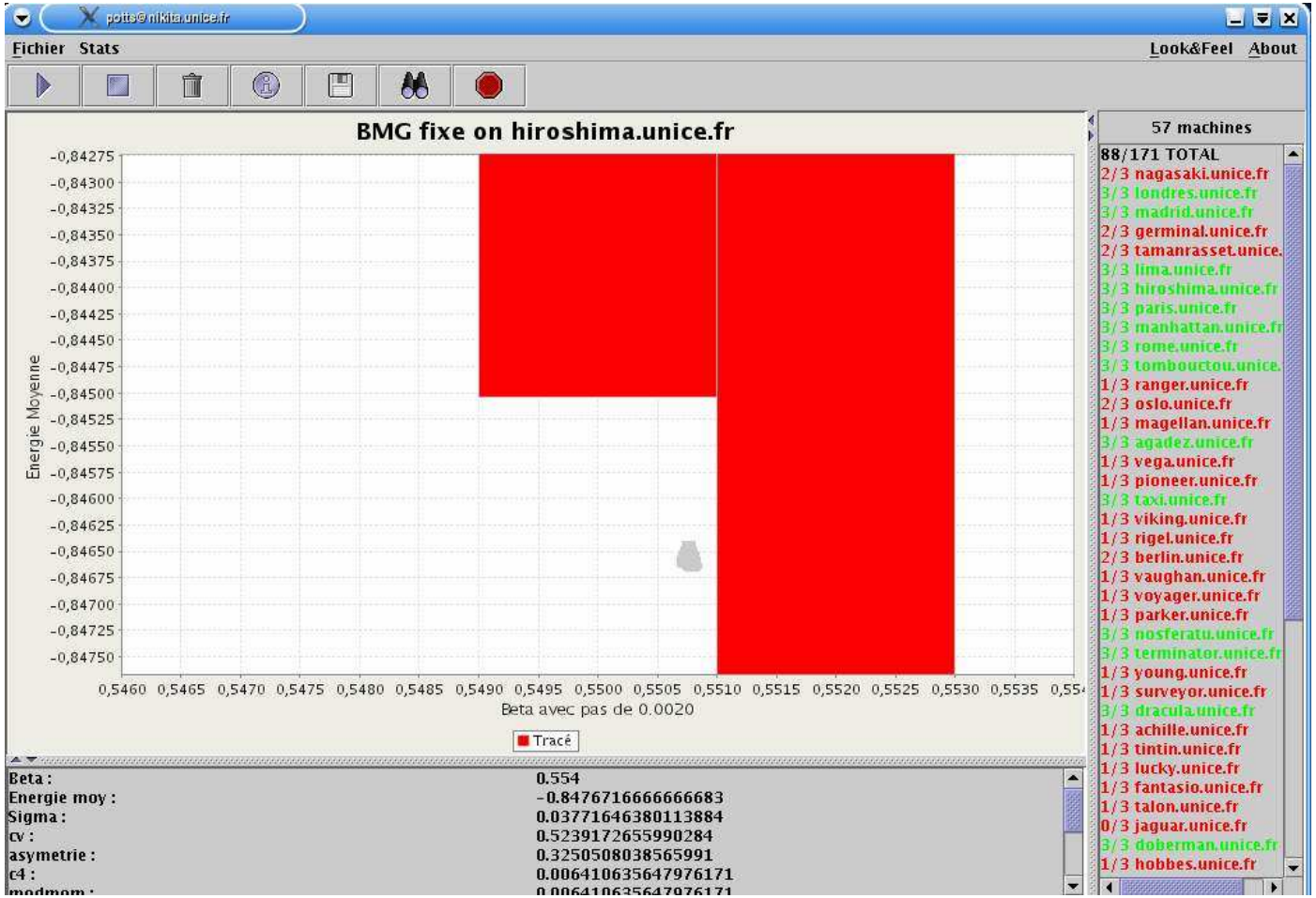
OO SPMD with a Jacobi



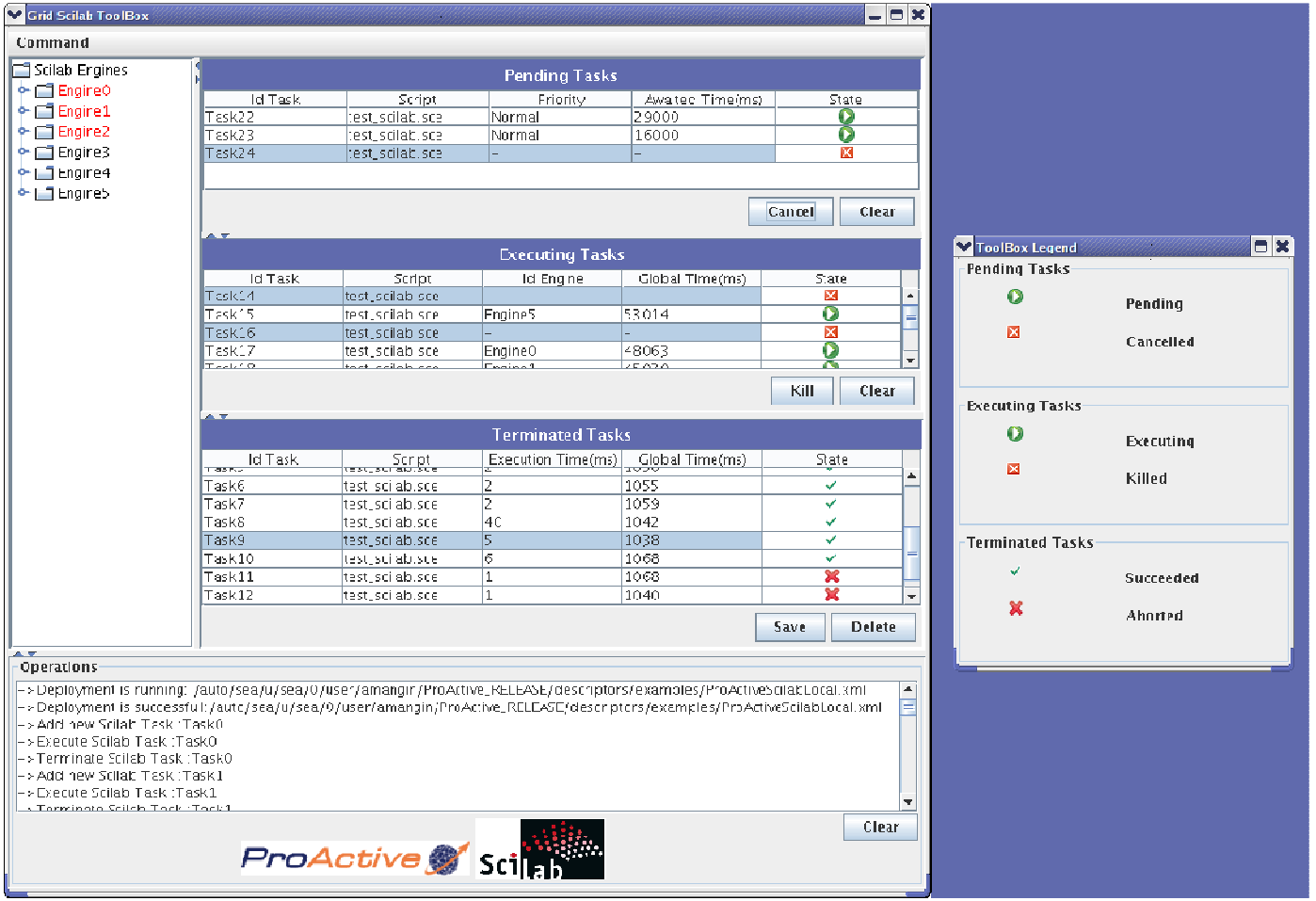
OO SPMD with a Jacobi



Monte Carlo Simulations, Non-Linear Physics, INLN

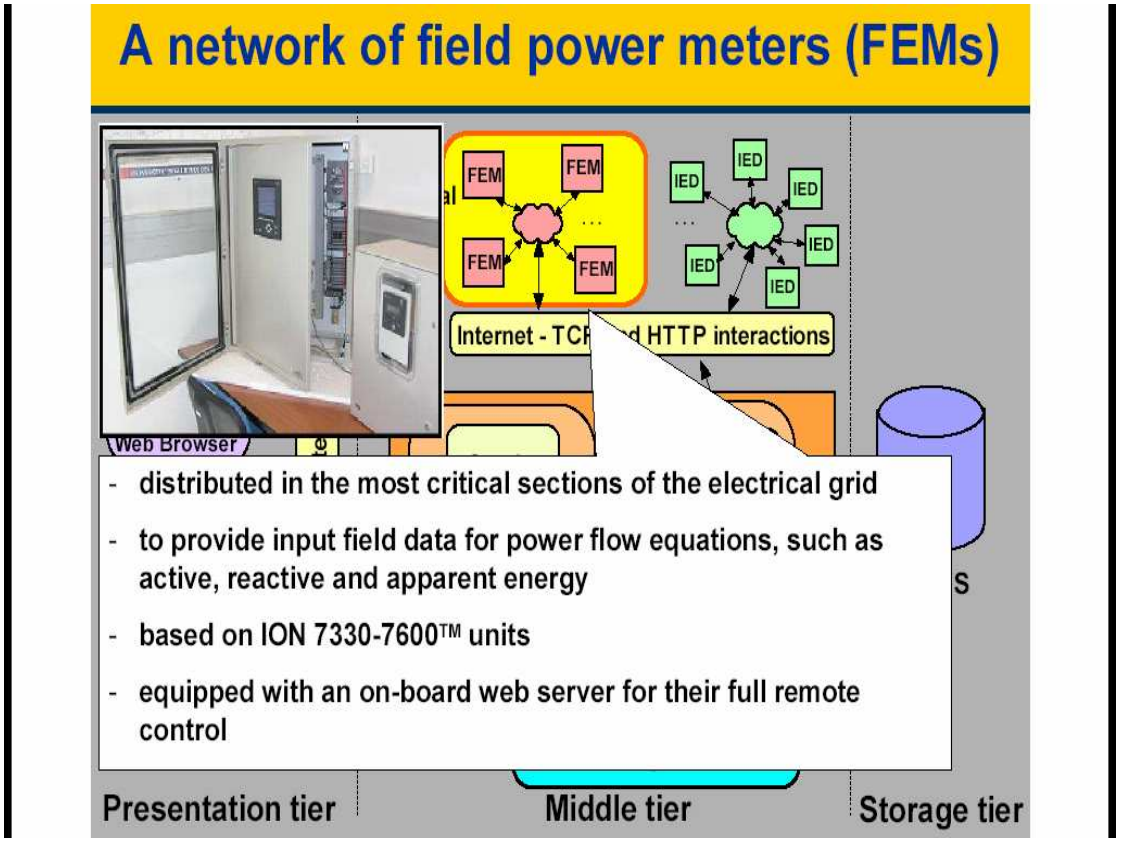


Scilab Grid Toolkit

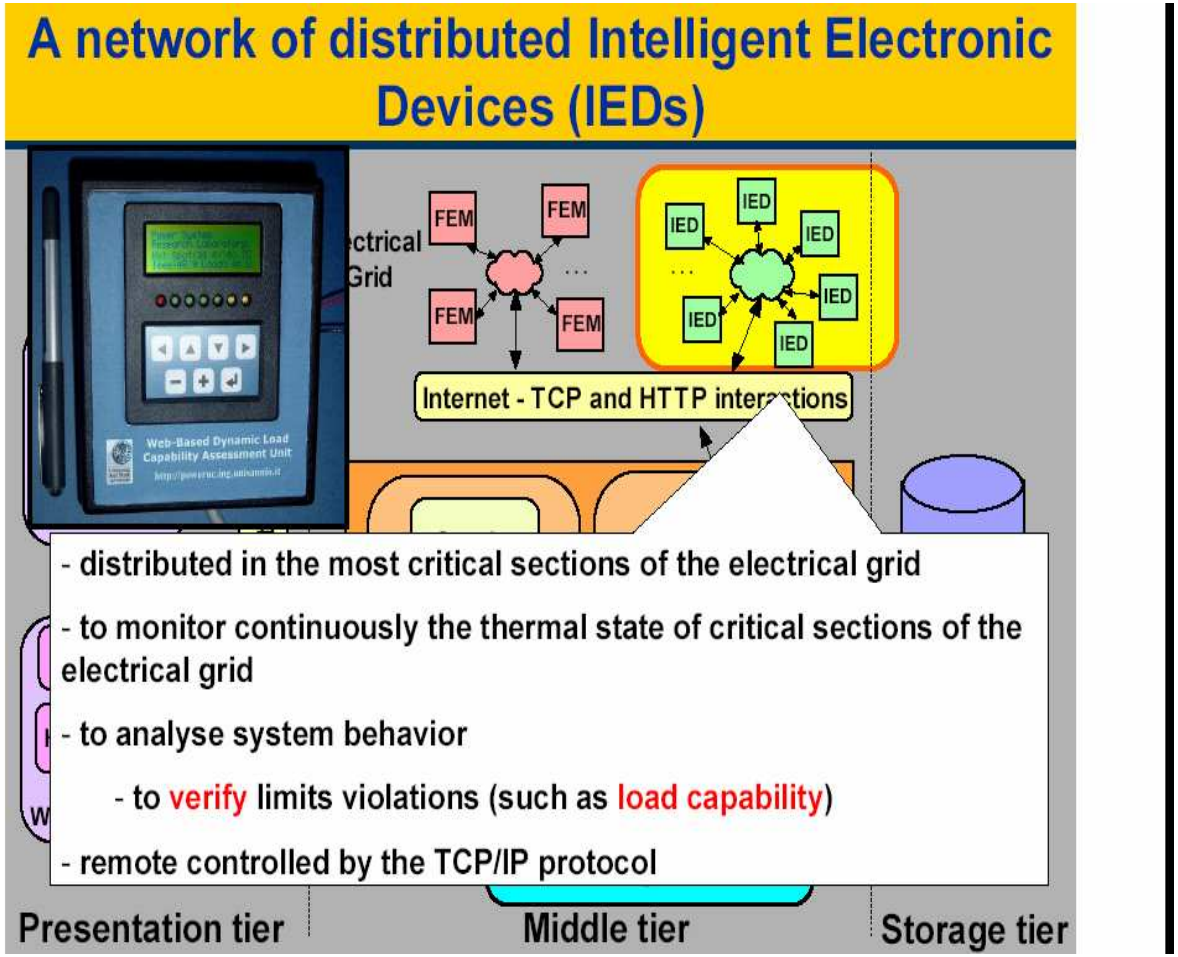


Electric Network Planning,
E. Zimeo et al., Benevento (Naples), Italy

On-line Power Systems Security Analysis (OPSSA)



Electric Network Planning,
E. Zimeo et al., Benevento (Naples), Italy

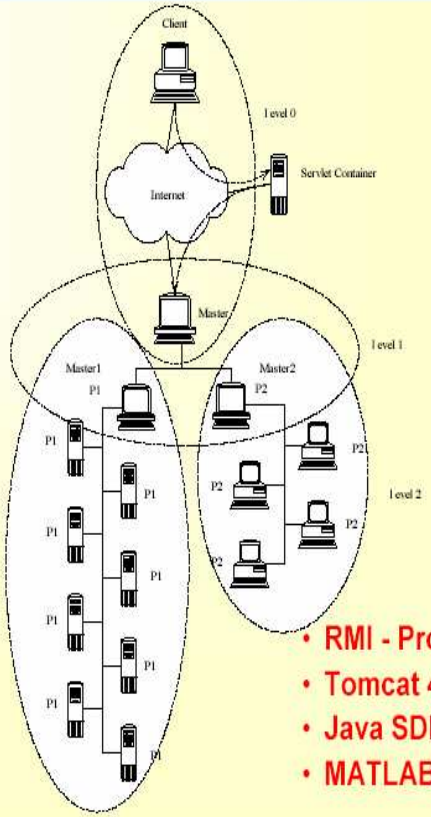


Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

Software platform evaluation on a testbed 1/2

- Standard IEEE 118-nodes test network
  - electrical network description is local to each computational resource
- The experiments refer to 186 contingencies
- A COW with P1 proc.
  - P1 = Pentium II 350 MHz
- A NOW with P2 proc.
  - P2 = Pentium IV 2 GHz

$$\begin{cases} n_1 = n_2 = \ldots = n_8 = 6 & P_1 \\ n_1 = n_2 = \ldots = n_4 = 35 & P_2 \end{cases}$$

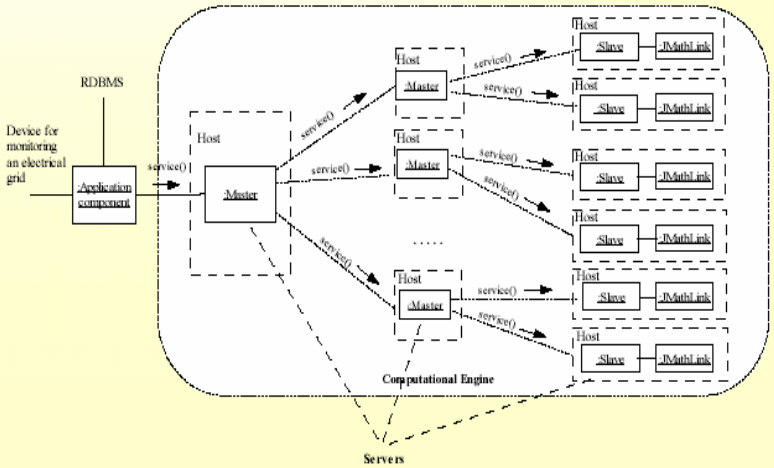


- RMI - ProActive
- Tomcat 4.1
- Java SDK 1.4.1
- MATLAB 6.5

Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

ProActive implementation of HM/S p.

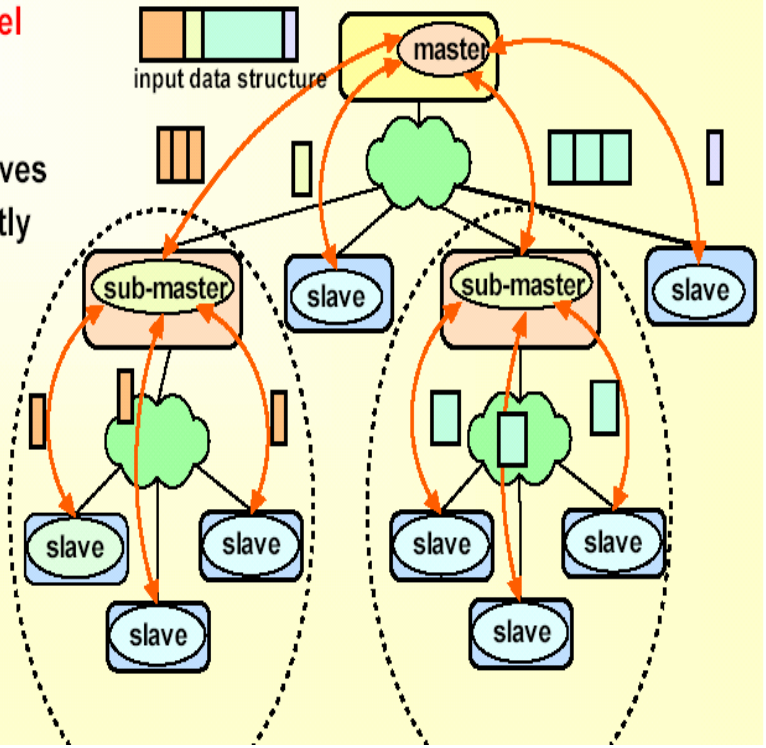
- **Each Server**
  - is defined as an active object (that can be dynamically created)
  - is allocated on a different computational resource (dynamically)
  - its method `service()` has to be asynchronously and remotely invoked by the client.
  - the asynchronous invocation is directly supported by ProActive



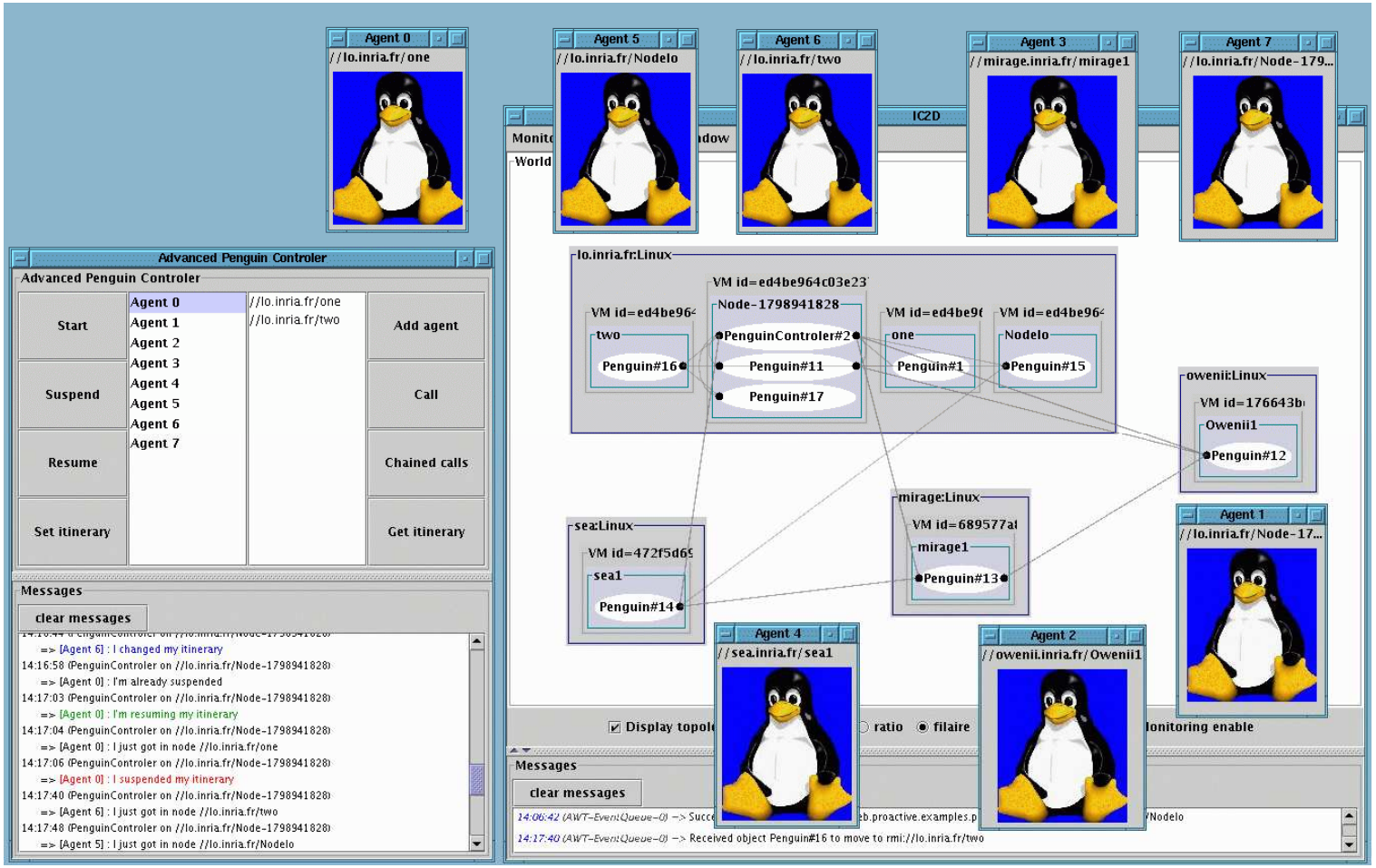
Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

The hierachical m/s model 1/3

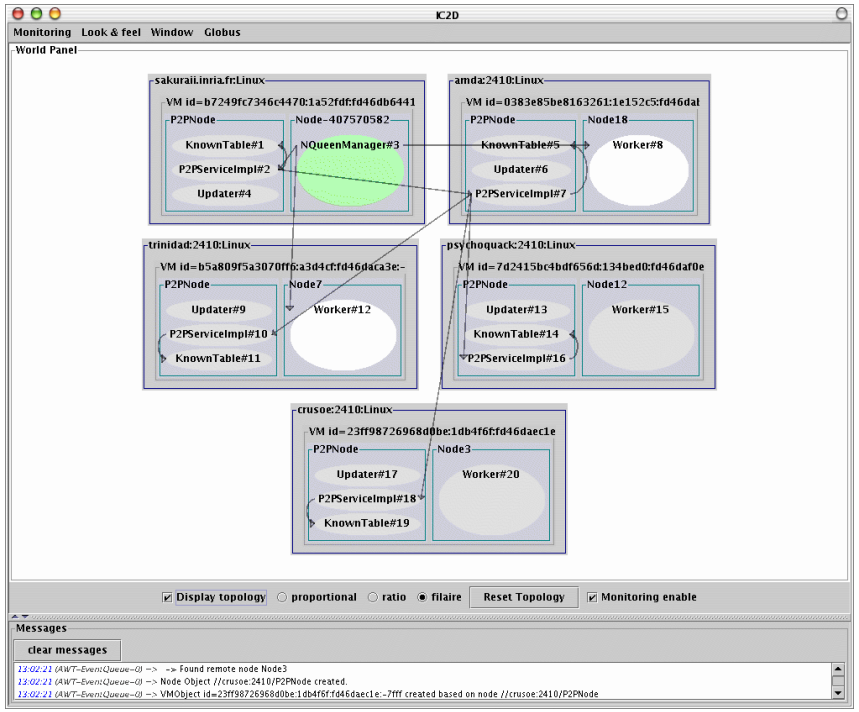
- hierarchical master/slave model
  - object-level parallelism
  - master and slaves are transparently created

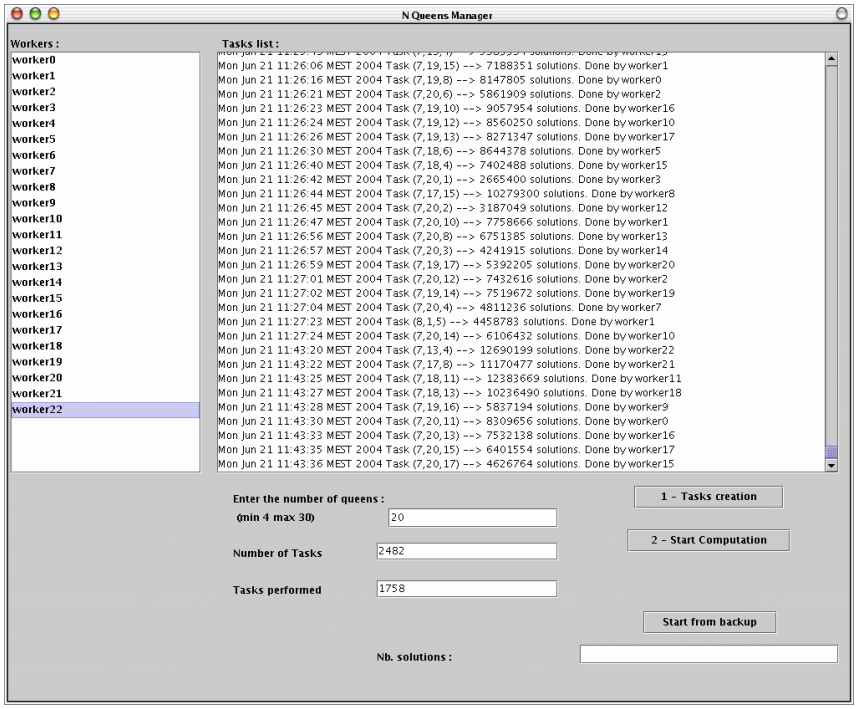


Mobile Application executing on 7 JVMs

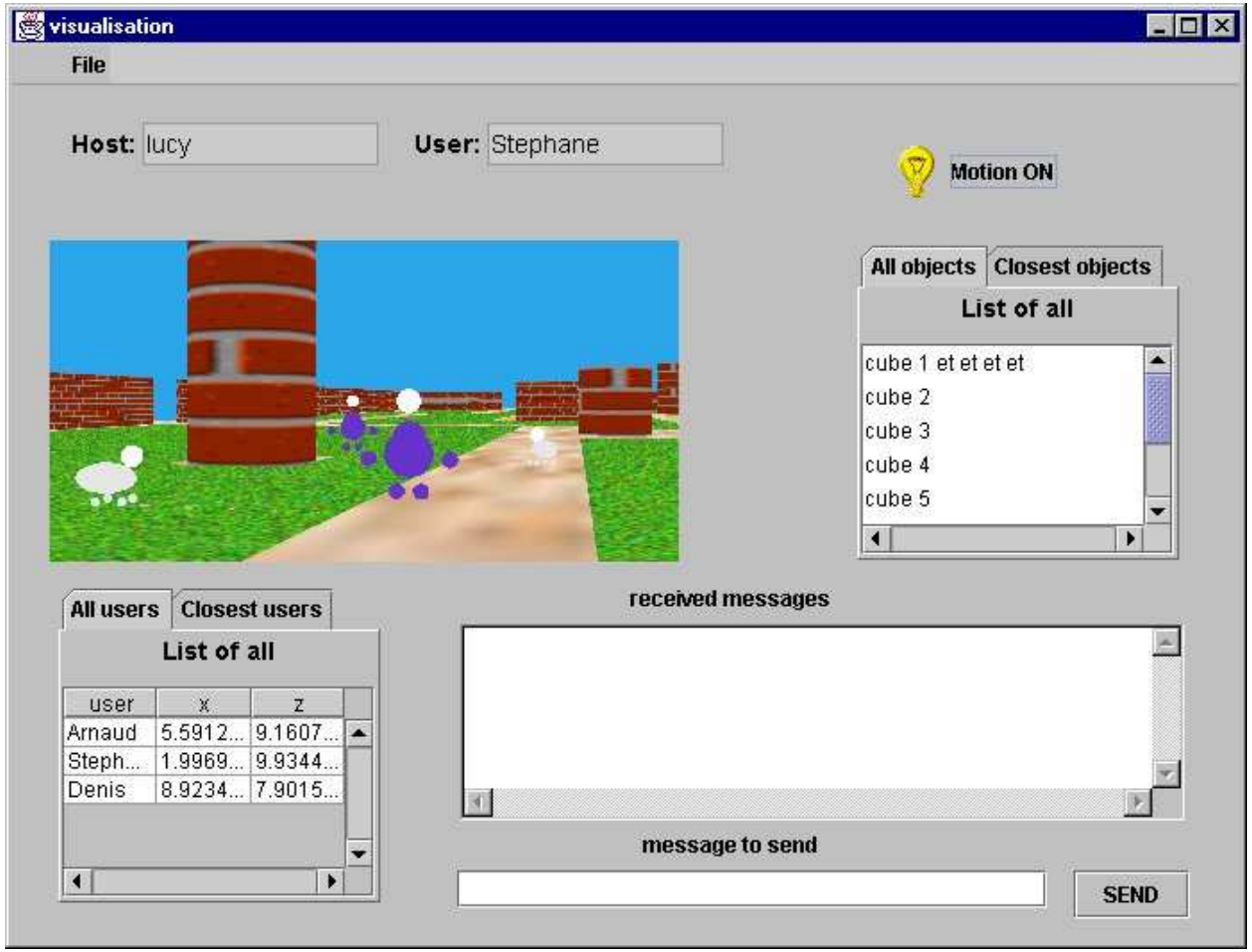


P2P N-Queens





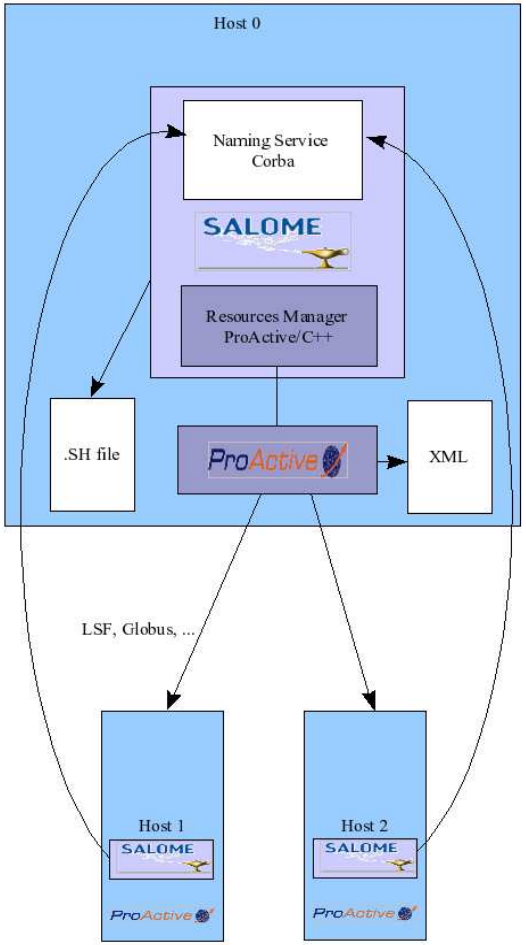
DIVA: Distributed Interactive Virtual World



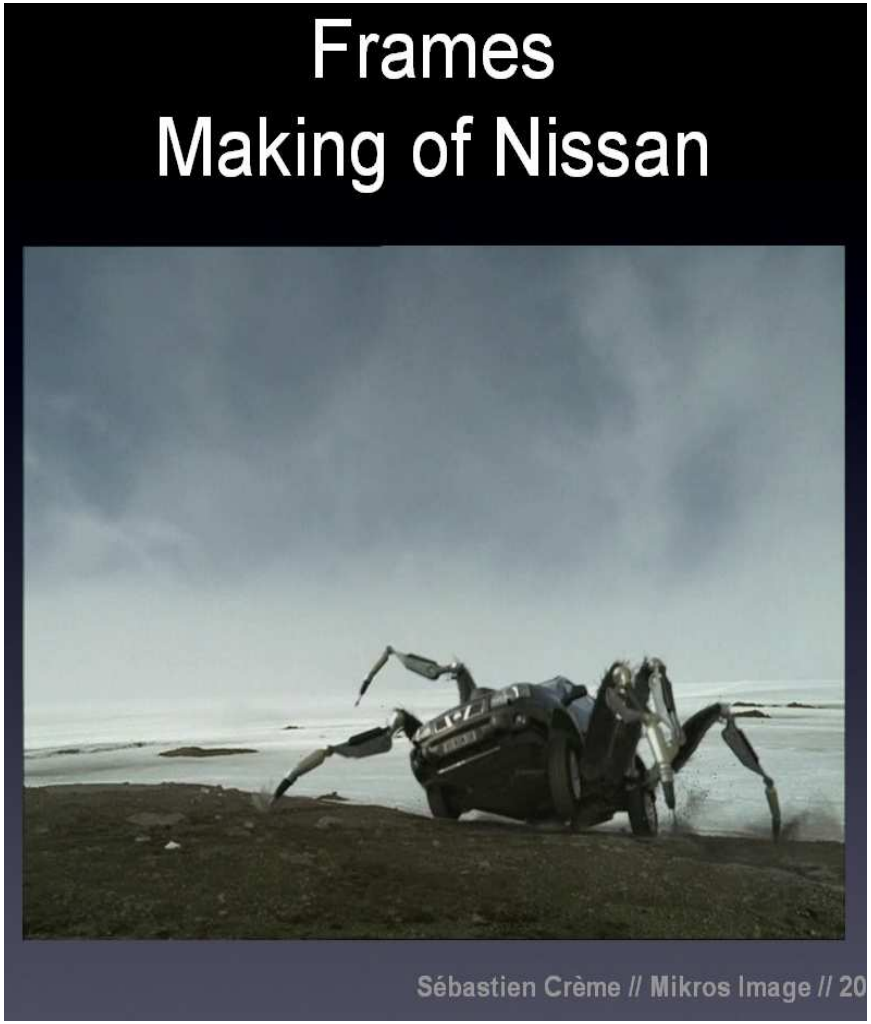
Salome

Co-developped with

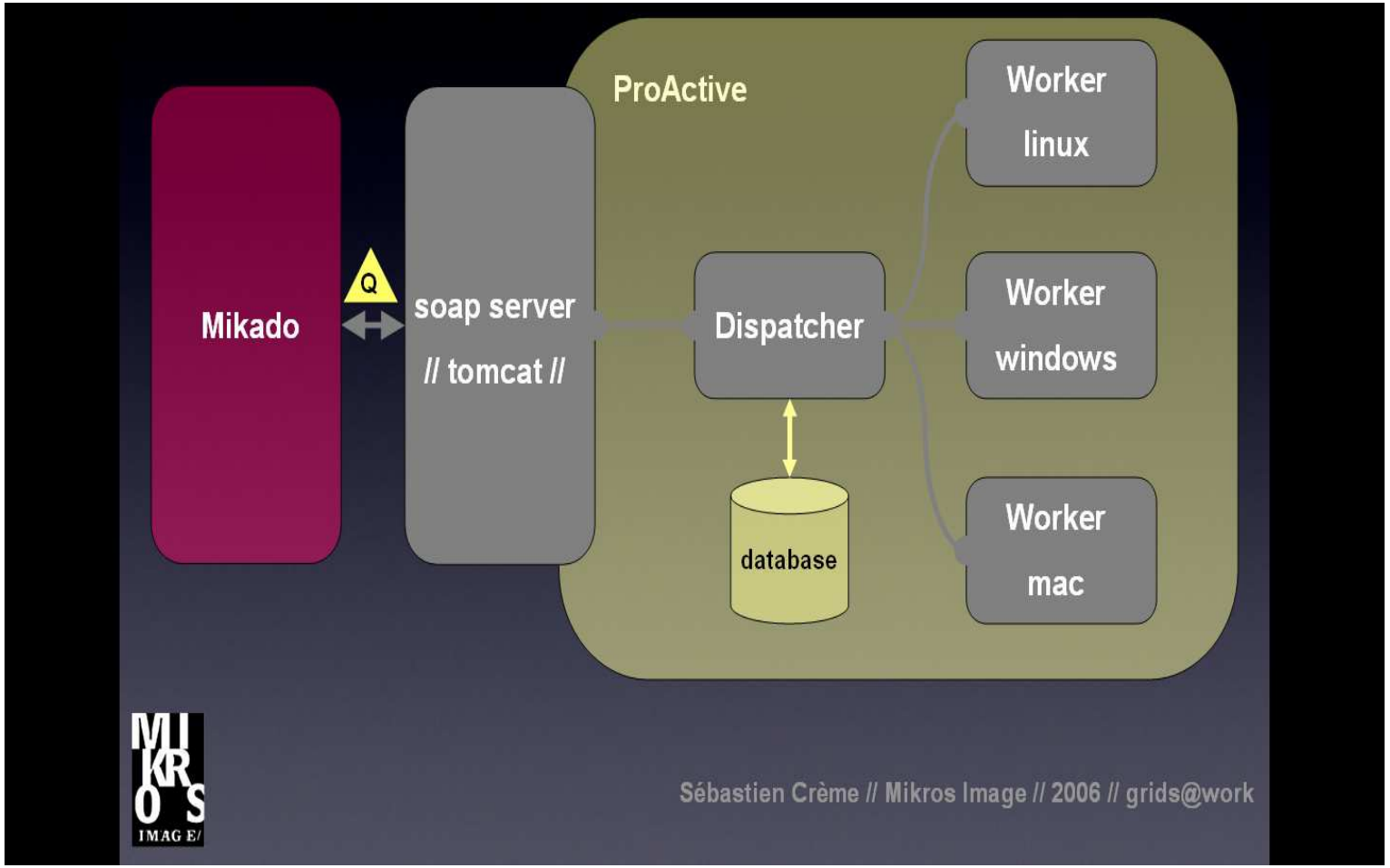
EDF and CEA



Mikros Image: Post Production

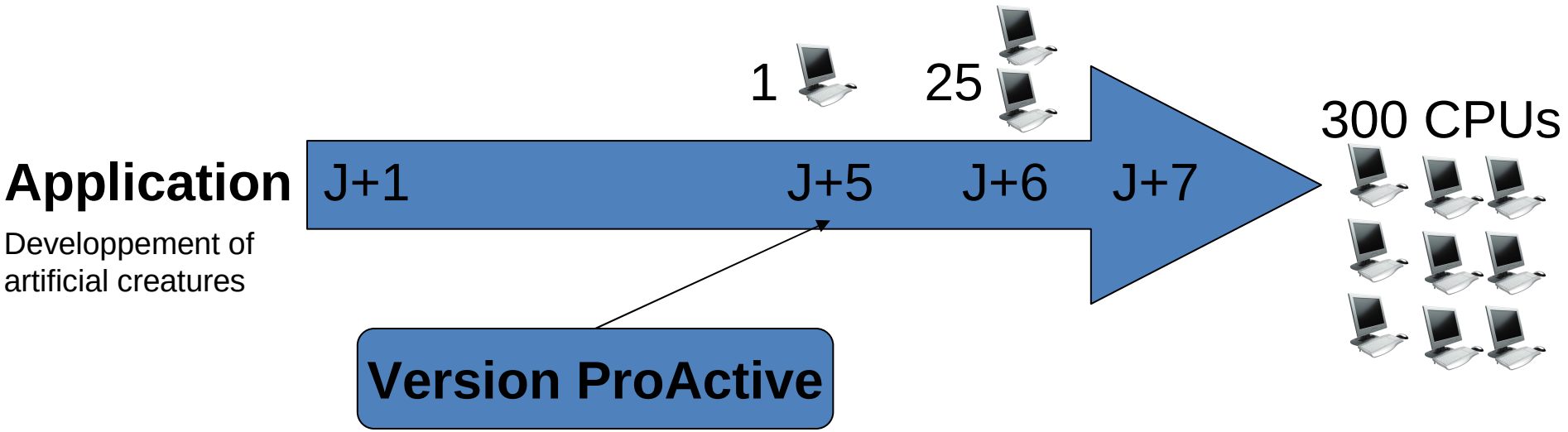


Mikros Image: Post Production



Artificial Life Generation

Sylvain Cussat-Blanc, Yves Duthen – IRIT TOULOUSE

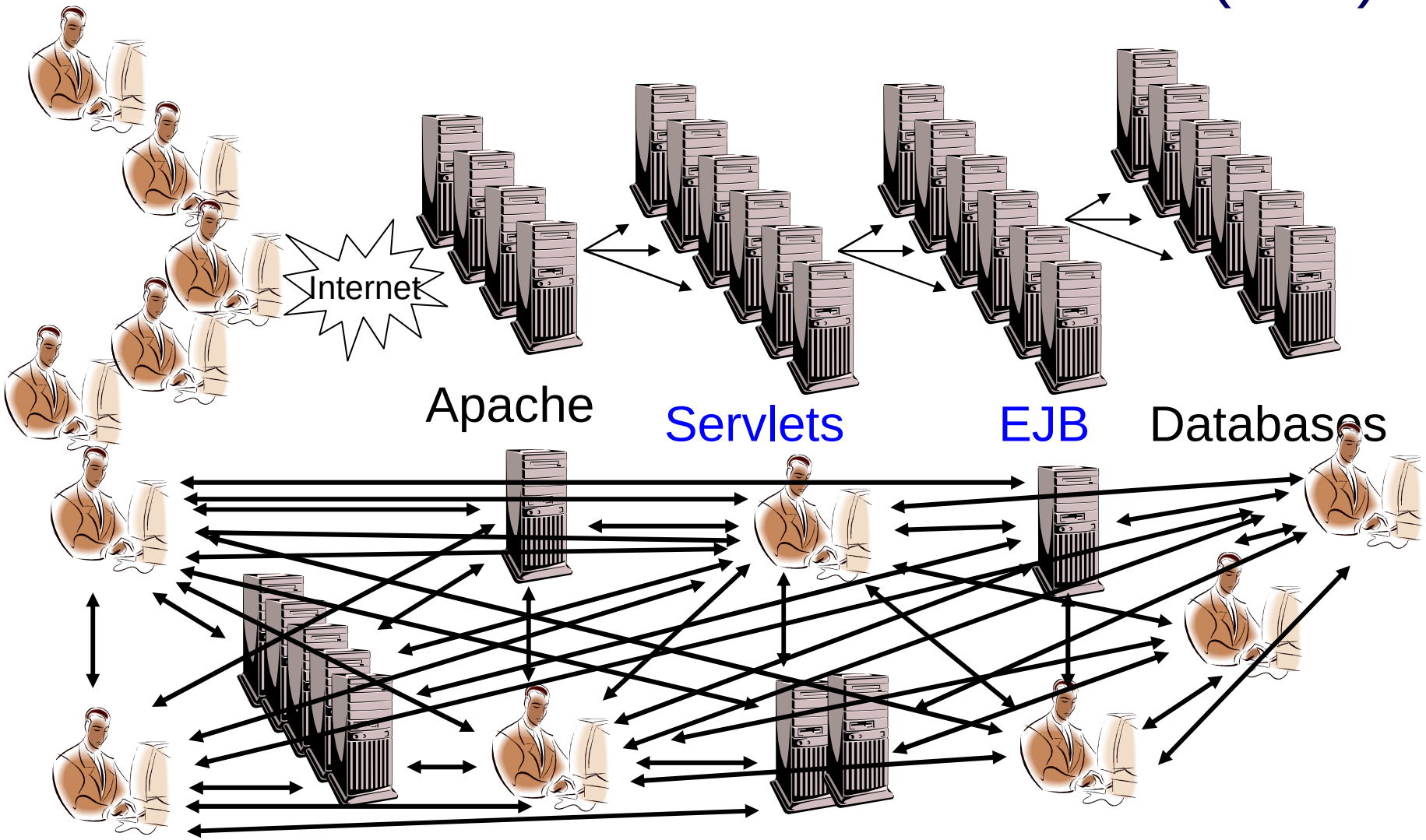


Initial Application	**1 PC**	**56h52 => Plantage**
ProActive Version	**300 CPUs**	**19 minutes**

Perspectives:

Real P2P

Architectures: Server to Peer-To-Peer (P2P)



SOA: Service Oriented Architectures

Pure P2P: Definition

Only PEERs, no above everything, top level, server(s)

Every peer is, somehow, also a server

No master … No slave !

System get organized dynamically, without static configuration

Coherent, desired behavior, dynamically emerges

P2P Examples (1)

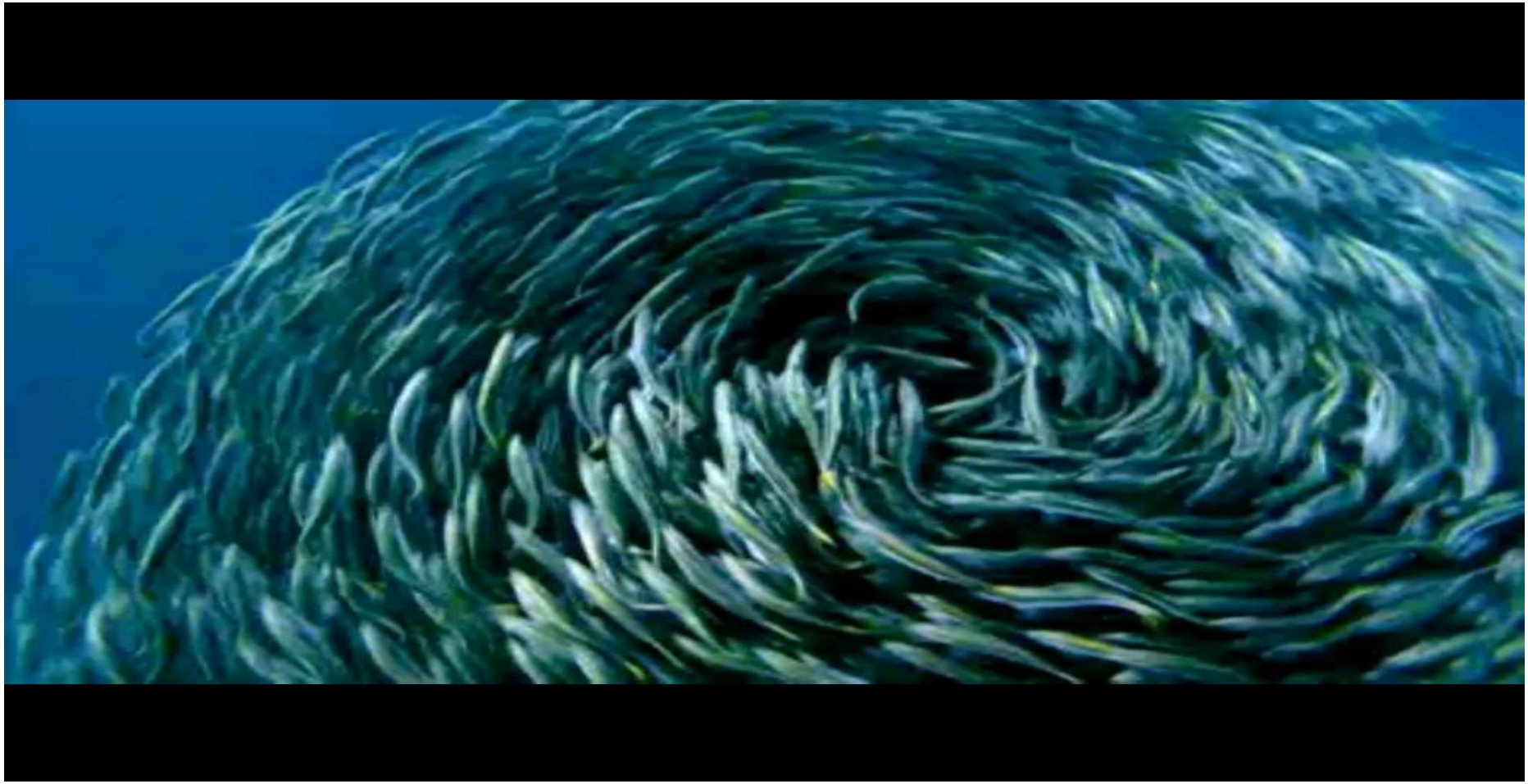
P2P can be difficult: need to be fault-tolerant, self healing

Not a P2P system

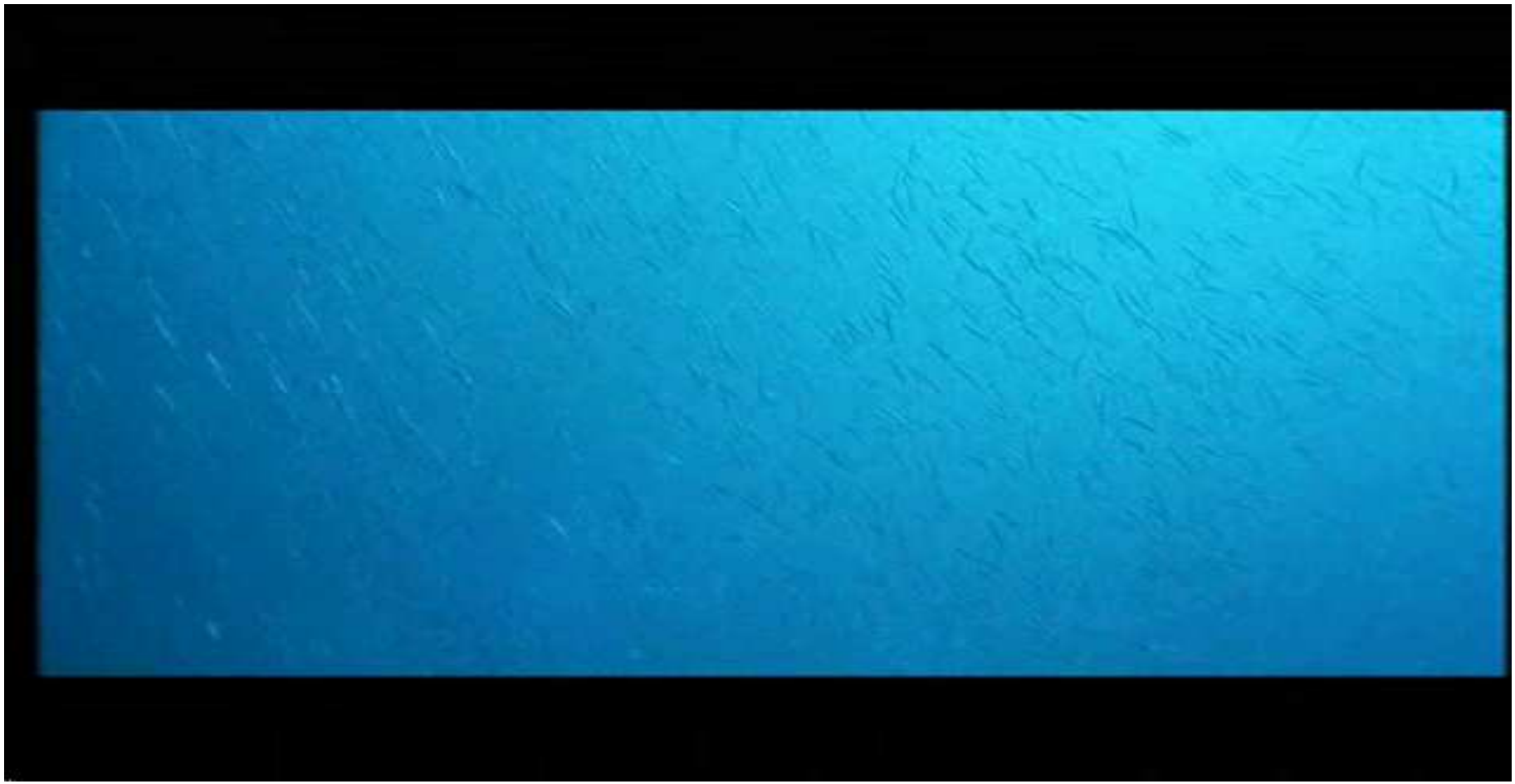
Neither a P2P system

P2P Examples (2)

P2P Examples (2bis)



A P2P system at work



P2P
in ProActive

Peer-to-Peer Computing
Model & Infrastructure

Use Sparse CPU Cycles from Desktop Workstations

Dynamic Computational Peer-to-Peer (P2P):

- Intranet & Internet
- Propose a High Level Model
  - Dynamic JVMs Network (computation nodes)
  - P2P Programming Model for Branch and Bound (B&B) problems

ProActive Context: no modification of Java language, of JVMs, …

P2P Infrastructure

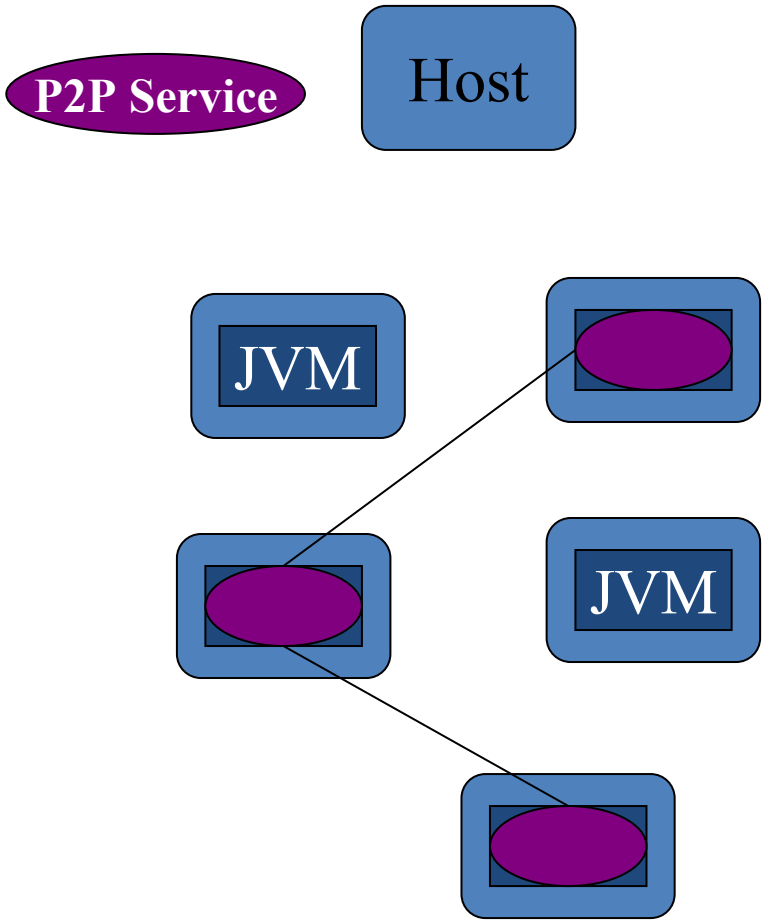
Hosts Network ➔ JVMs Network

Dynamic environment :

- Discovery : recording and un-recording
- Resources (JVMs) acquisition

Self-Organizing and Tunable Infrastructure :

- Time To Update (TTU): peer availability
- Number Of Acquaintances (NOA): keep up infrastructure
- Time To Live (TTL): in hop for JVMS depth search, use for NOA



P2P Programming Model Branch & Bound

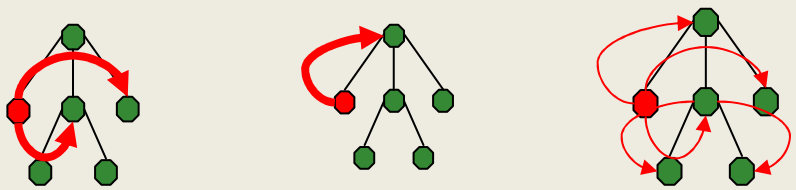
Dynamic P2P Programming for B&B, etc.:

- Tasks, sub-tasks managing?
- Tasks communications: discovery tasks, volatility?

Entities:

- **Worker**: connects model with infrastructure.
- **Solver**: Worker associate.
- **Problem**: Worker associate (Dividing, Merging, Finding)
- **Result**: solution abstraction

Communications between Workers

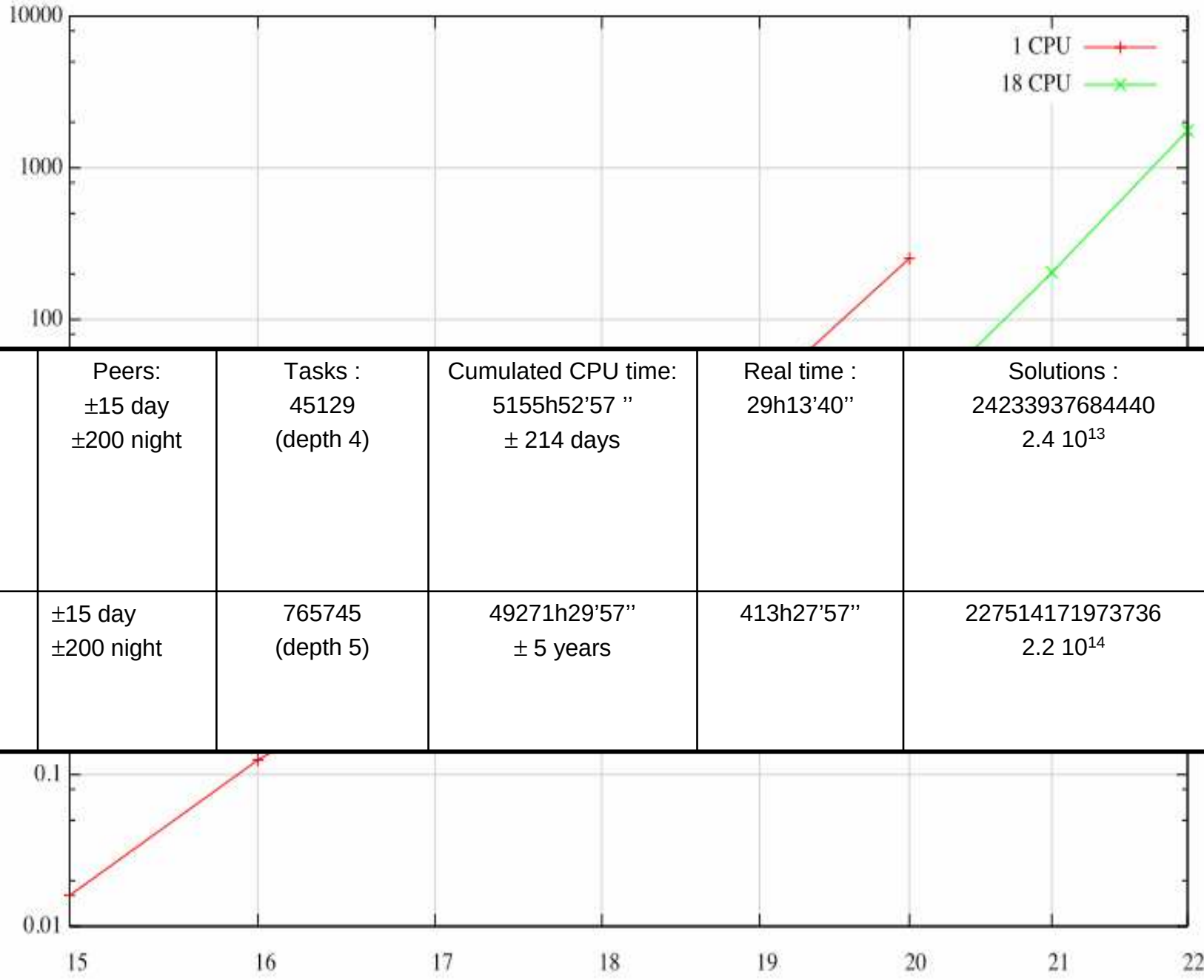


P2P NQueens with Vincent Cavé

Master-slave application with Dynamic Workers Acquisition

Generic Workers (reusable)

Fault tolerant (Workers availability)

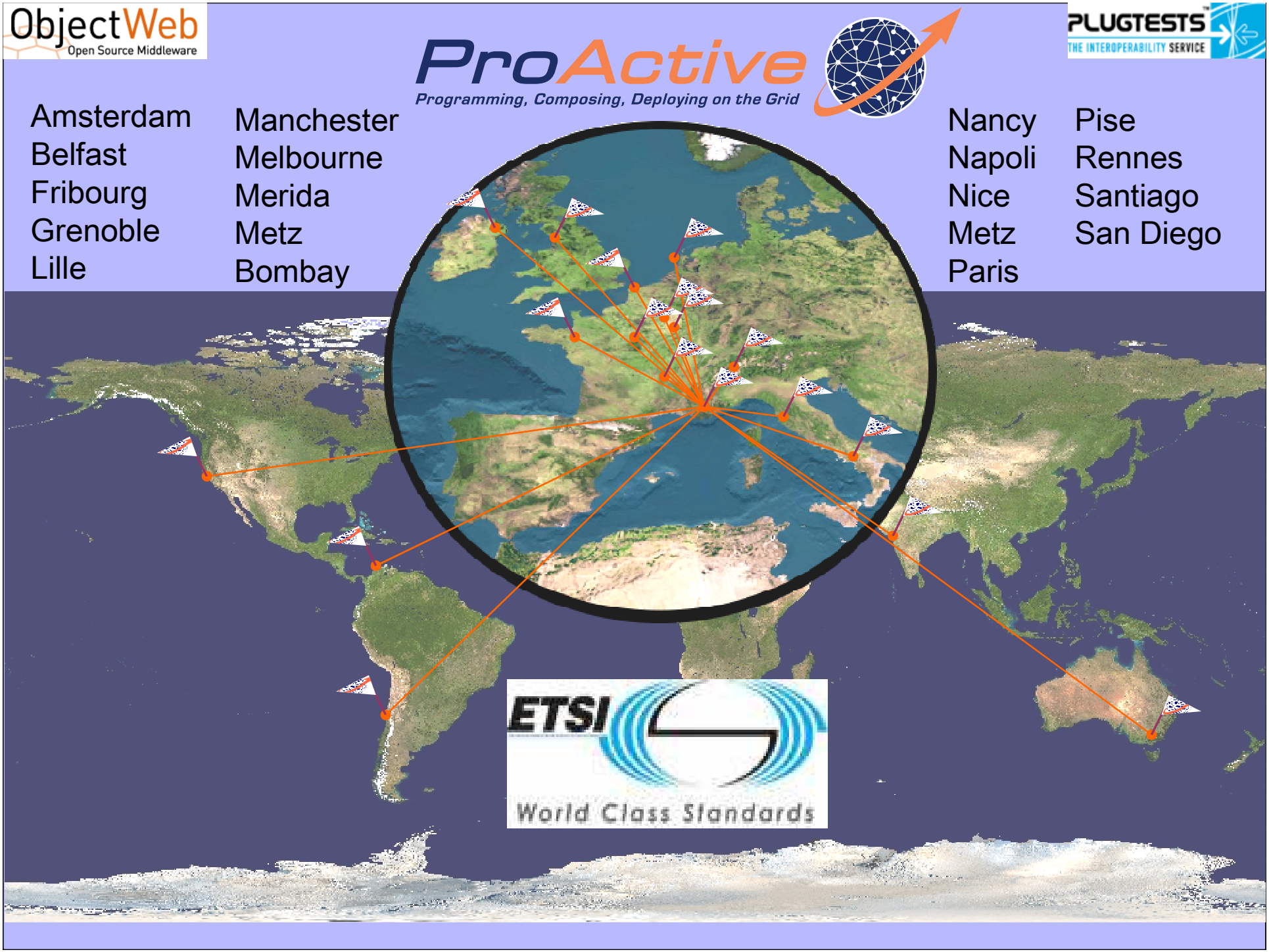


23-Q	Peers: ±15 day ±200 night	Tasks : 45129 (depth 4)	Cumulated CPU time: 5155h52'57 '' ± 214 days	Real time : 29h13'40''	Solutions : 24233937684440 $2.4\ 10^{13}$
24-Q	±15 day ±200 night	765745 (depth 5)	49271h29'57'' ± 5 years	413h27'57''	227514171973736 $2.2\ 10^{14}$

ProActive User Group Grid Plugtests

2004, 2005, 2006

A successful event!


ObjectWeb
Open Source Middleware
ProActive
Programming, Composing, Deploying on the Grid
PLUGTESTS
THE INTEROPERABILITY SERVICE
Amsterdam
Belfast
Fribourg
Grenoble
Lille
Manchester
Melbourne
Merida
Metz
Bombay
Nancy
Napoli
Nice
Metz
Paris
Pise
Rennes
Santiago
San Diego
ETSI
World Class Standards

ProActive Grid Plugtests, 04, 05, 06

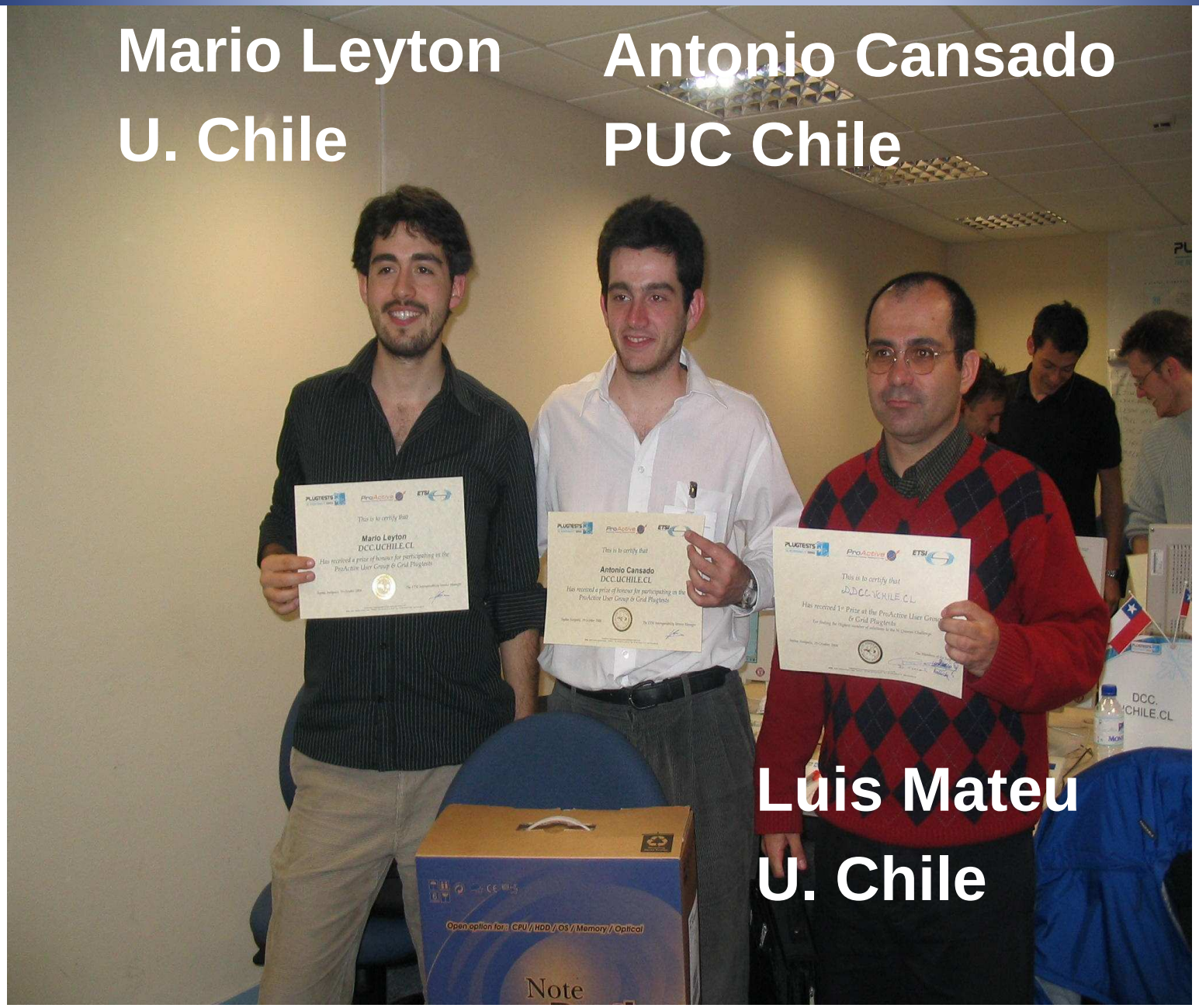
Between 20 to 40 sites around the world:

2006: 4130 cores

Total power: ~ 1700 GFlops (100 Giga Flops in 04)

Highly heterogeneous :

Machines:	IBM, SGI, Sun, Bull, Mac
OS:	Linux, Windows, Solaris, MacOS, SGI Irix
JVMs:	Sun, SGI, BEA
Protocols:	ssh, rsh, sshGSI, Globus Gram
Job Schedulers:	PBS, LSF, Grid Engine, Oar, Prun, Globus


Mario Leyton
U. Chile
Antonio Cansado
PUC Chile
Luis Mateu
U. Chile

2006 Winners :

Eight Samurai - University of Tokyo – JAPAN

The Grid Guru

New Features

Fault-Tolerance for ProActive

Provide a fault-tolerance protocol for ProActive

- Maintain portability of ProActive: use only standard Java serialization for checkpointing
- Scalable: target Grid applications

First Step: applications running on a **single cluster** (homogeneity and low failure rate)

Communication Induced Checkpointing: CIC

Next Step: **interconnect clusters** using message logging

Implementation

This protocol is implemented within ProActive

- Single checkpoint and location server
- Fault detection (fail-stop) by heartbeat messages

Fully transparent use:

~> java org.objectweb.MyAppli

~> java -Dproactive.ft.server=host.inria.fr org.objectweb.MyAppli

No need to alter nor recompile code !

Benchmarks

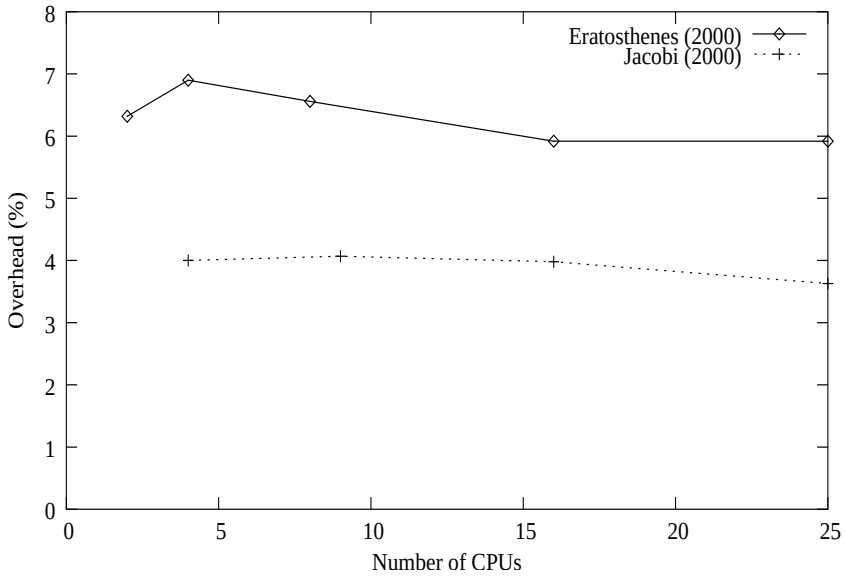
Master-slaves application: Sieve of Eratosthenes

Overhead less than 7.5 %

Peer-to-Peer application: Jacobi iteration

Overhead less than 4.5 %

Scalability



Available from January 2005 Release

HTTP communication layer

Why ?

Facilitate firewalls crossing between ProActive runtimes

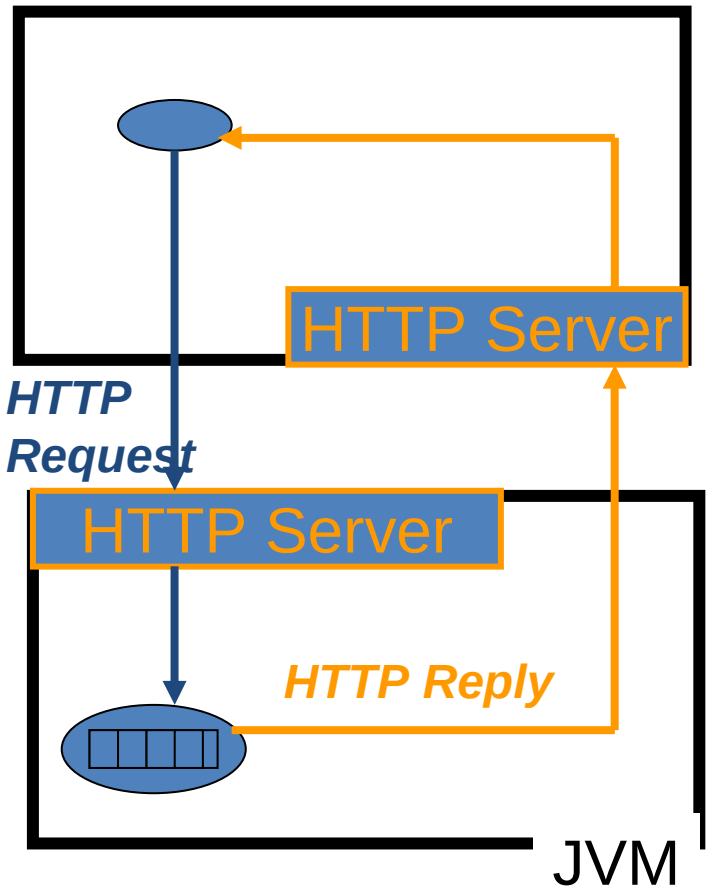
How ?

Encapsulate ProActive Requests into HTTP

Usage :

Transport configuration in a XML file

➔ Fully transparent !



9.2 Active objects as Web Services

Why ?

Make active objects accessible from any foreign language

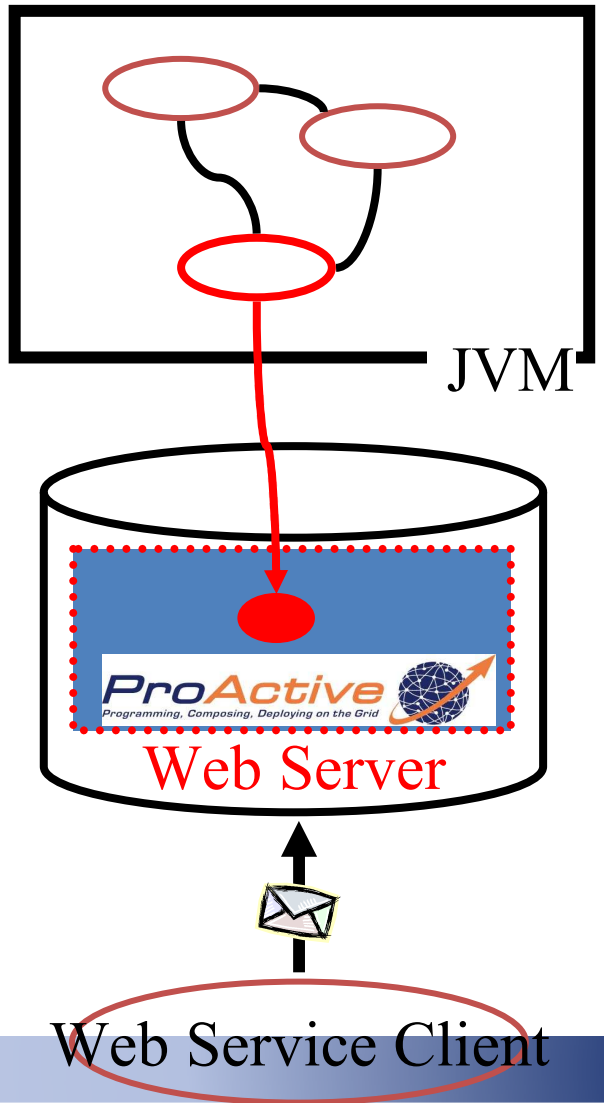
How ?

HTTP Web Server

Extended SOAP Engine (*Apache SOAP or Axis*)

Usage :

```
ProActive.exposeAsWebService();
ProActive.unExposeAsWebService();
```



Active objects as Web Services

Why ?

Make active objects accessible from any foreign language

How ?

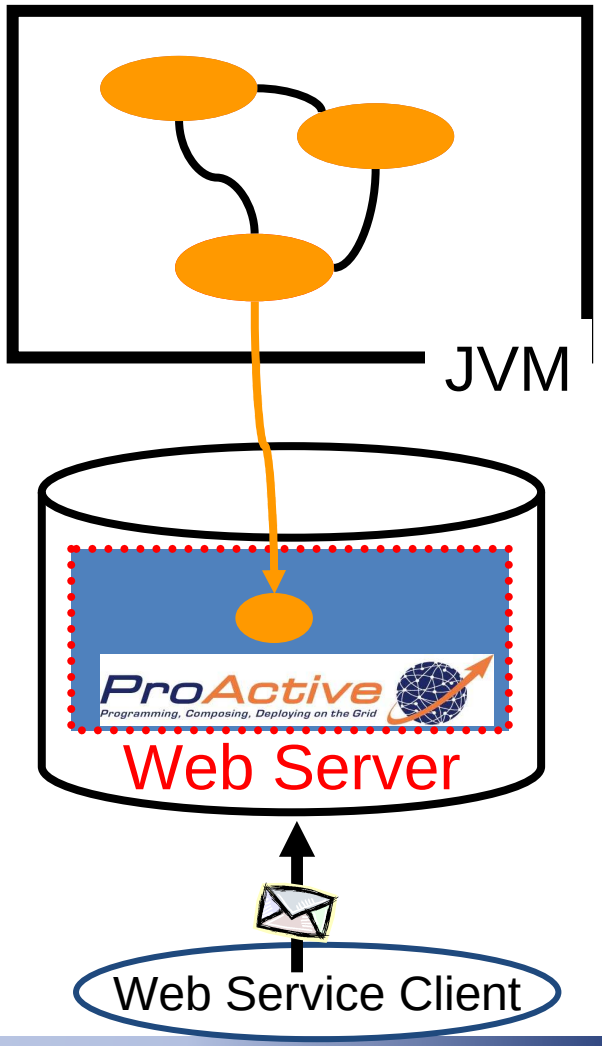
HTTP Web Server

Extended SOAP Engine (*Apache SOAP or Axis*)

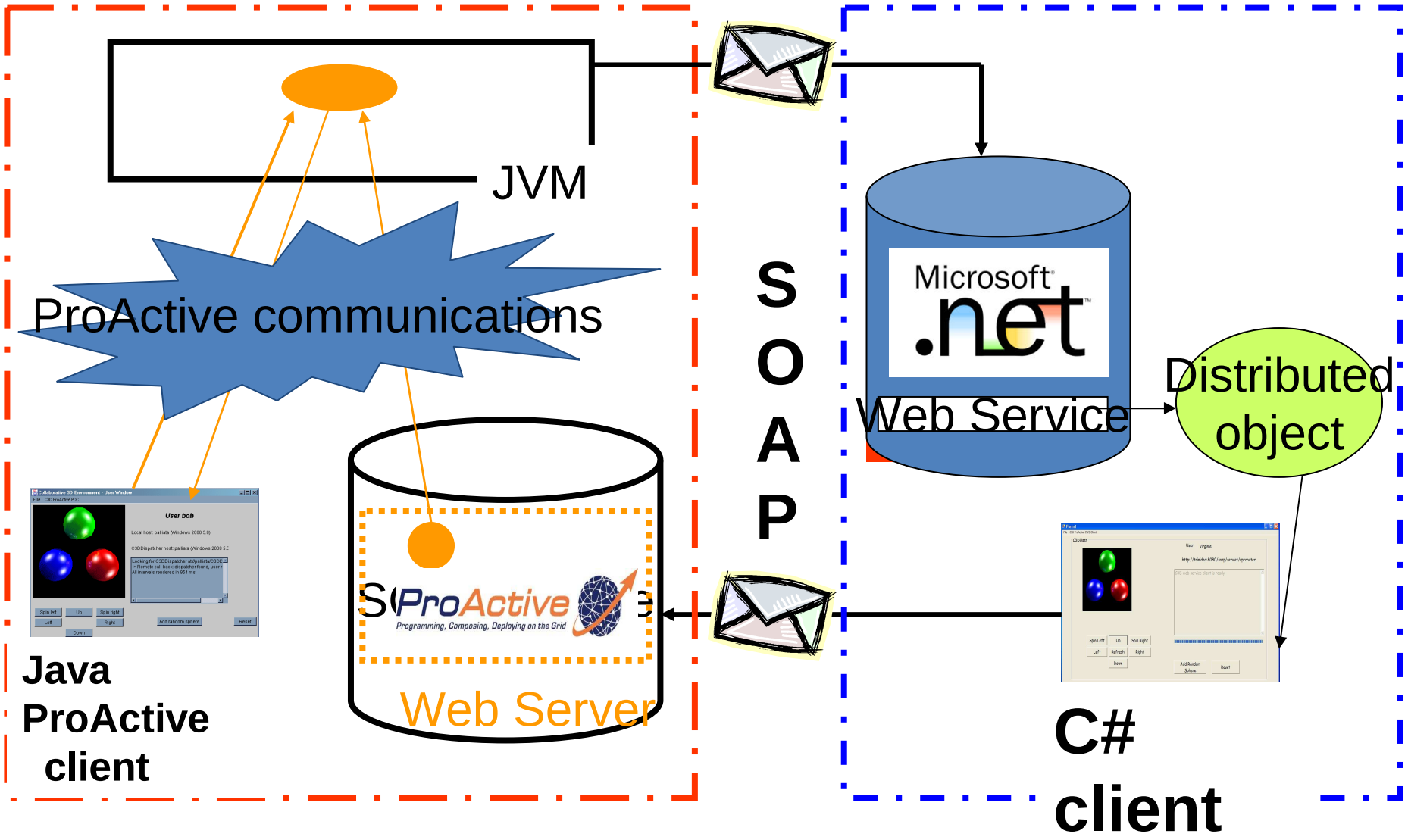
Usage

ProActive.exposeAsWebService();

ProActive.unExposeAsWebService();



.net / C# interoperability



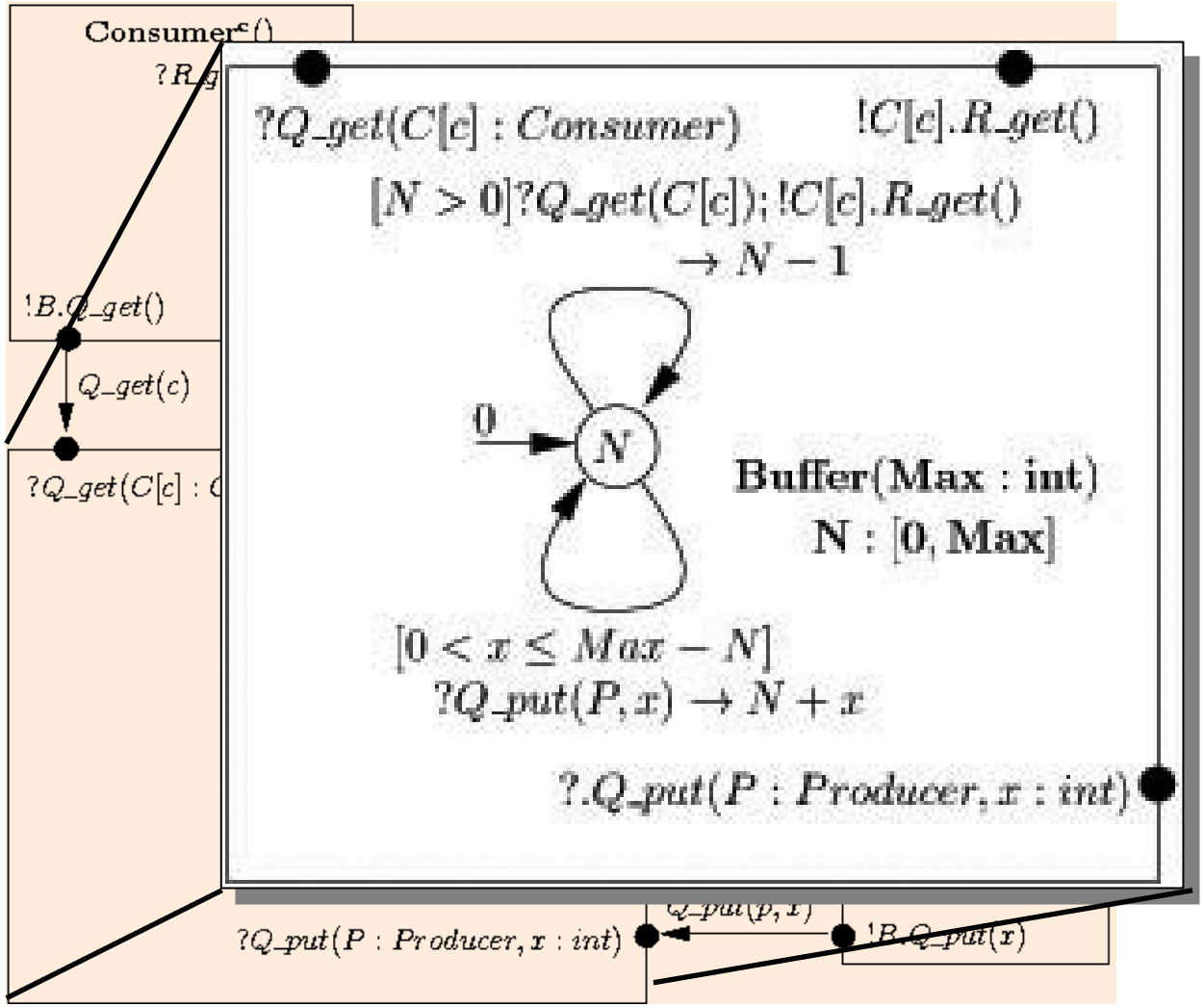
10.
On-Going
+ Perpectives

Formal Verification of Behavioral Properties

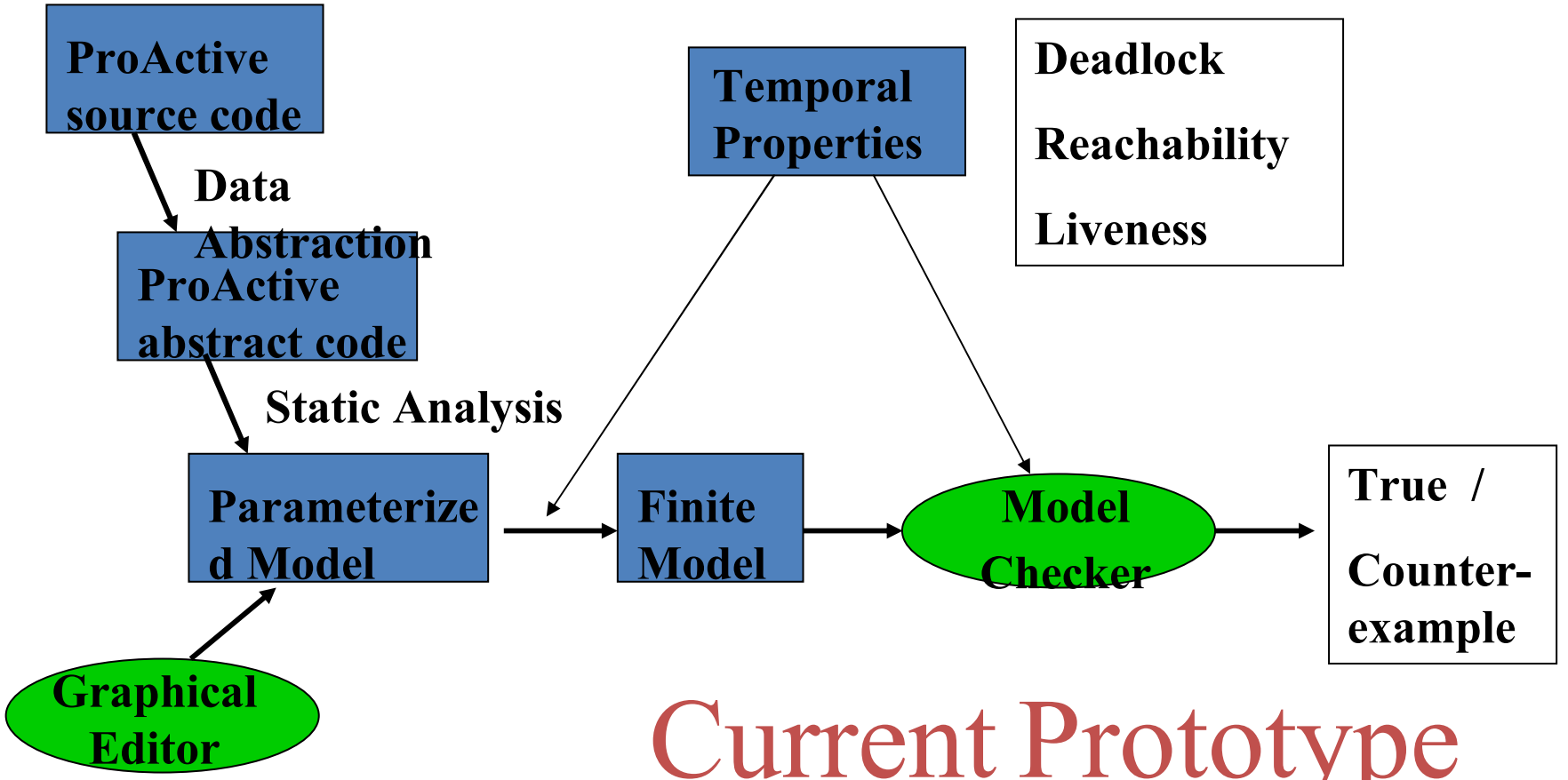
Eric Madelaine
work with
Rabea Boulifa, Tomas Barros, Christophe Massol

- For Asynchronous Distributed Objects
- Based on Strong Semantic Guaranties
- Application Level Properties

Models for Distributed Objects

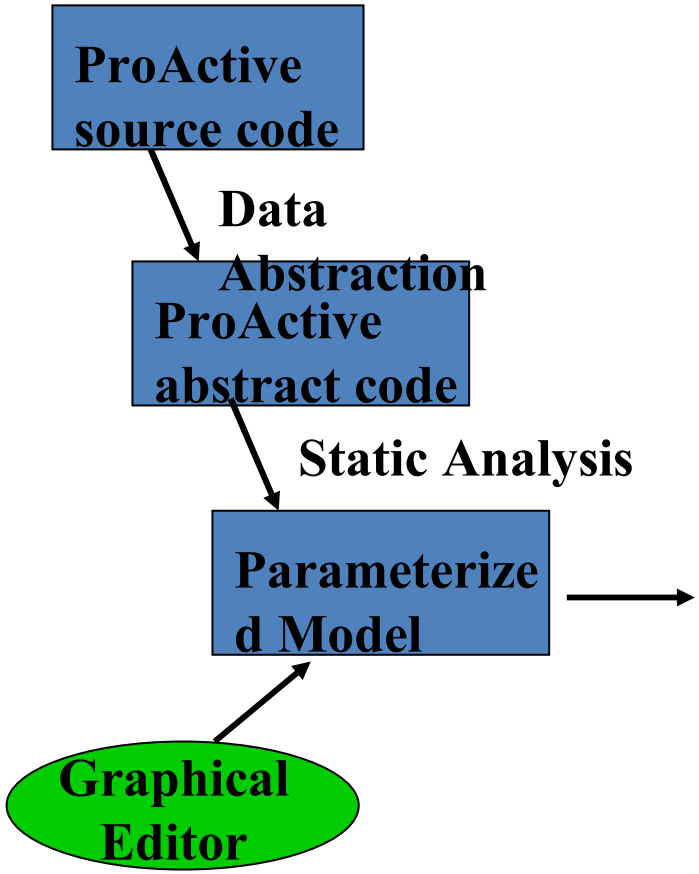


Verification of Properties



Current Prototype

Perspectives



Specification / Implementation

Correct composition of components

Safe Component-based Development

- ***Aim :*** **guarantee correct behaviour of composition of components**
  (deadlock freeness, progress properties, safety and liveness properties)
  - **By formal verification and model checking**
  - **Including control features of the component model (deployment, dynamic binding)**
  - **During execution, before replacing/updating a subcomponent.**

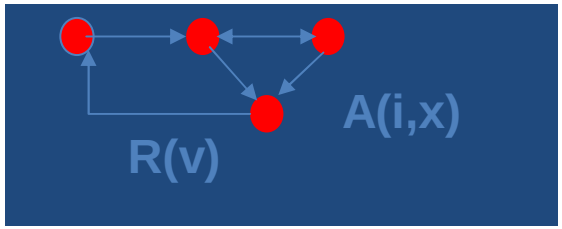
- ***Tools :***
  - **Analysis of ADL (composition) and Java code (primitives)**
  - **Automatic Generation of a finite behavioural model**
  - **Standard – State of the Art – Verification and Model-Checking tools.**

Safe Component-based Development

```
<definition name="components.System">
    <component name="BufferSystem"
     definition="BufferSystem(3)">
  <interface name="alarm" role="client"/>
```

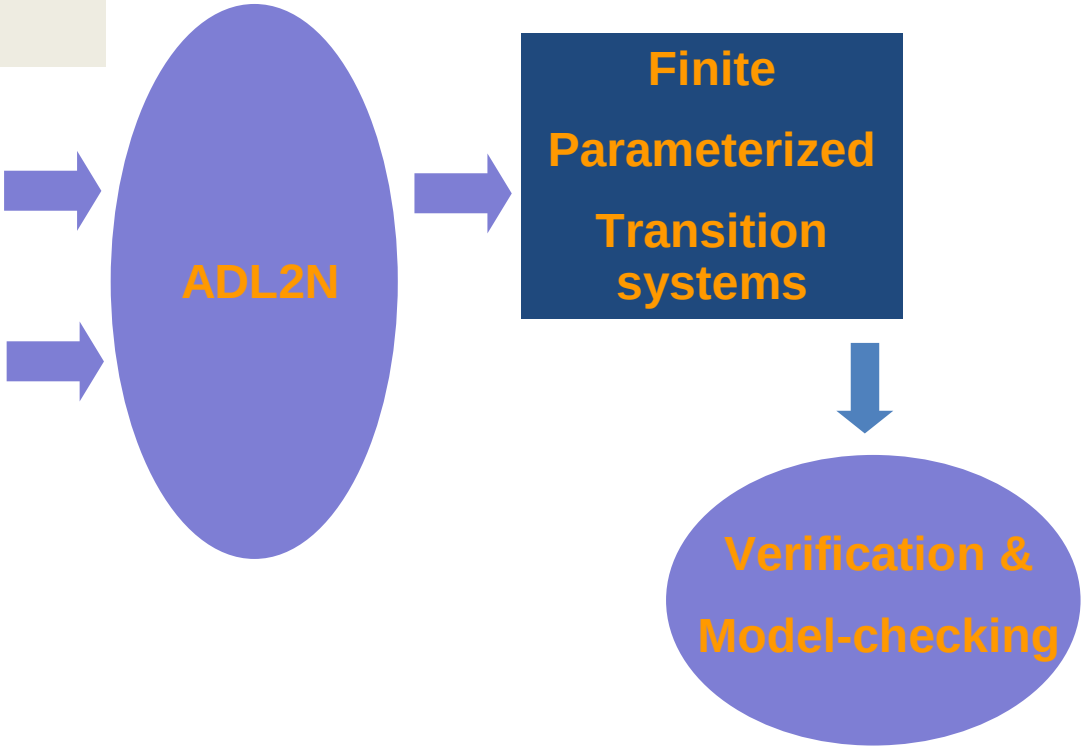
Architecture Description (ADL)

(Composites)



Behaviour Specification

(Primitive Components)



ProActive over OSGi

Why targetting OSGi ?

Usage of the OSGi platform is widespread

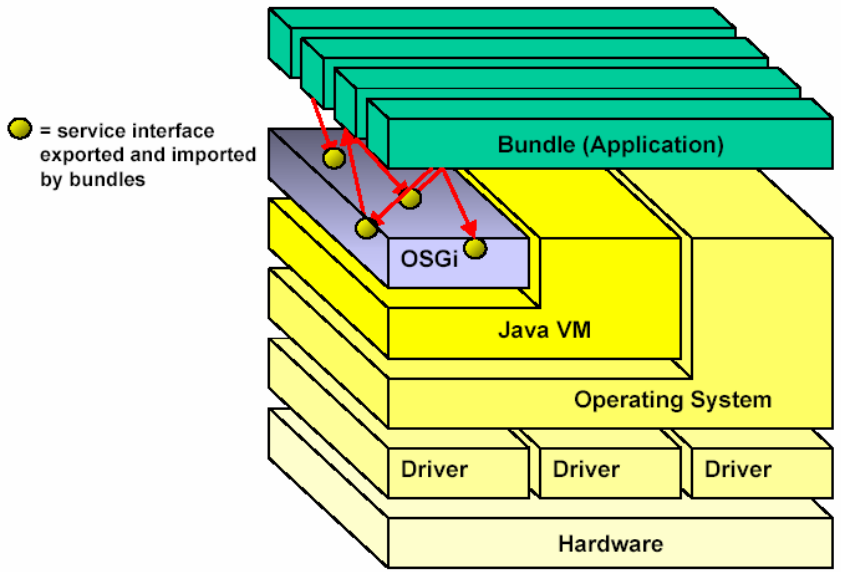
- Residential or Industrial Gateway
  - OSGi enables incremental upgrades and extensions in mission critical « always on » situations
- **PDA, smart phones**, etc
  - **OSGi is a unified mobile platform, with a small footprint and can run disconnectable applications from multiple, independant sources**
- Embedded devices (e.g. automotive devices)
  - OSGi provides a viable management solution: enables both end user « pull » and management/operator « push » applications and services

ProActive-enabled OSGi Services : what for and how ?

A ProActive-based remote management solution of **multiple OSGi platforms**

- mgt bundle on each platform
- a remote monitoring acting in parallel, using ProActive

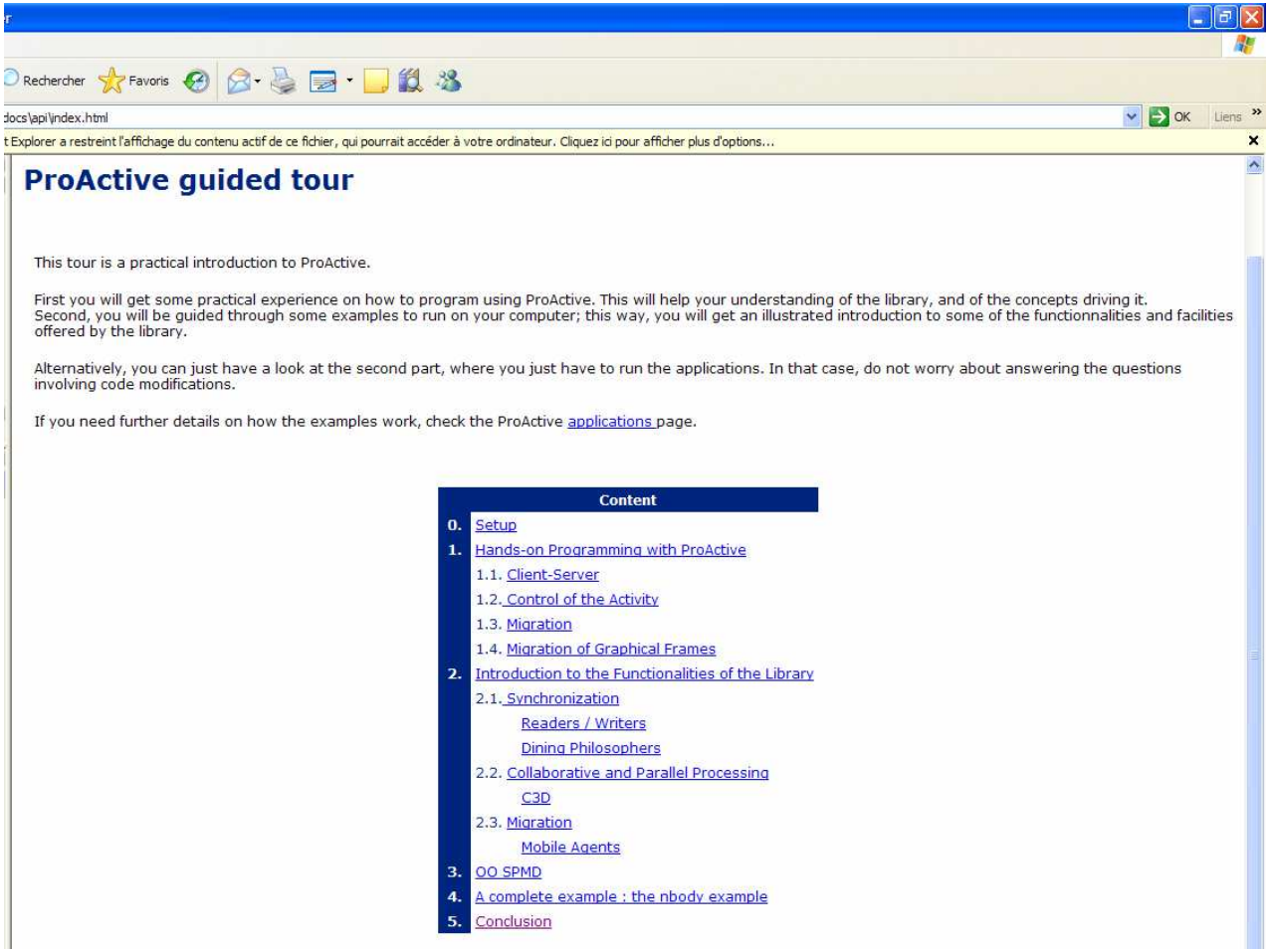
Future generation apps built as coordinated OSGi distributed and/or mobile services (e.g. load distribution in a car)



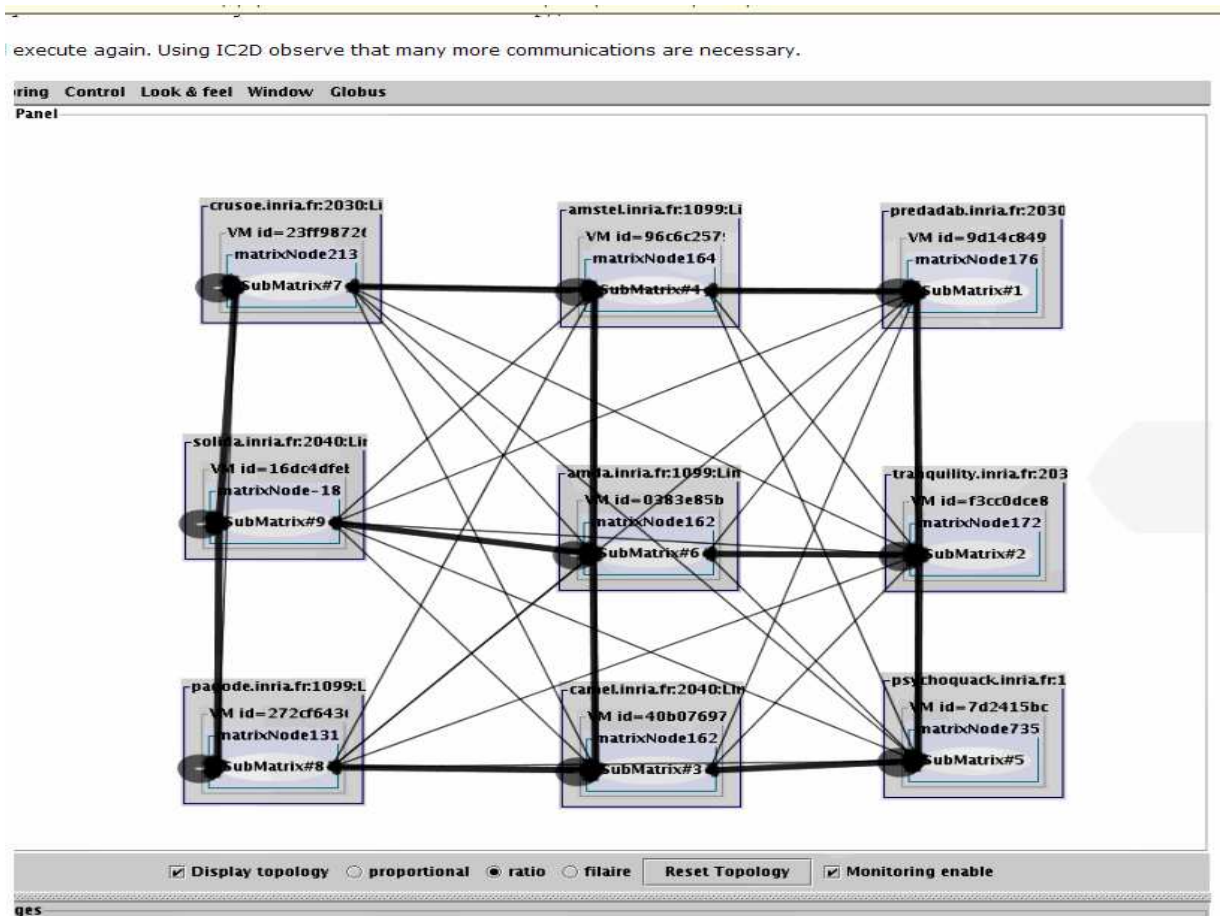
Teaching GRID, Parallelism, Distribution, Concurrency, Multi-Threading

with ProActive

Teaching the GRID with ProActive



Teaching the GRID with ProActive

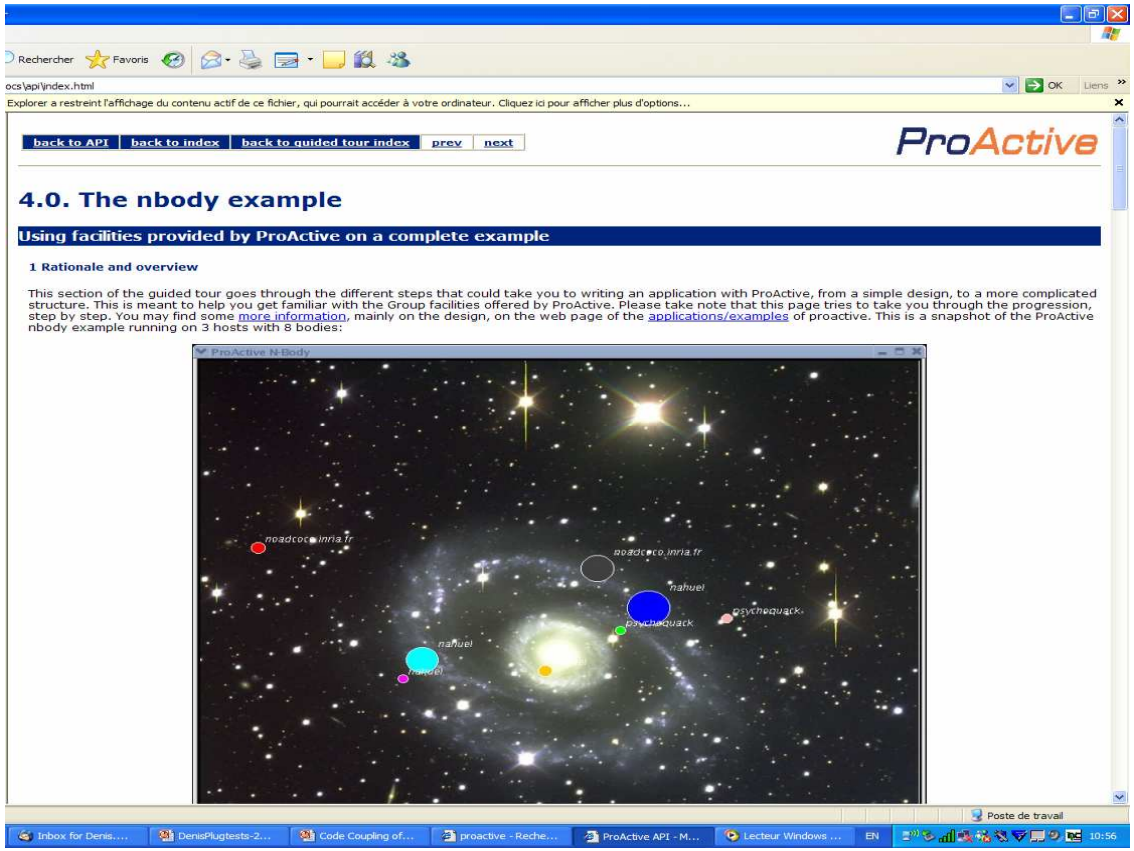


OO SPMD

Step by step

On a Jacobi

Teaching the GRID with ProActive

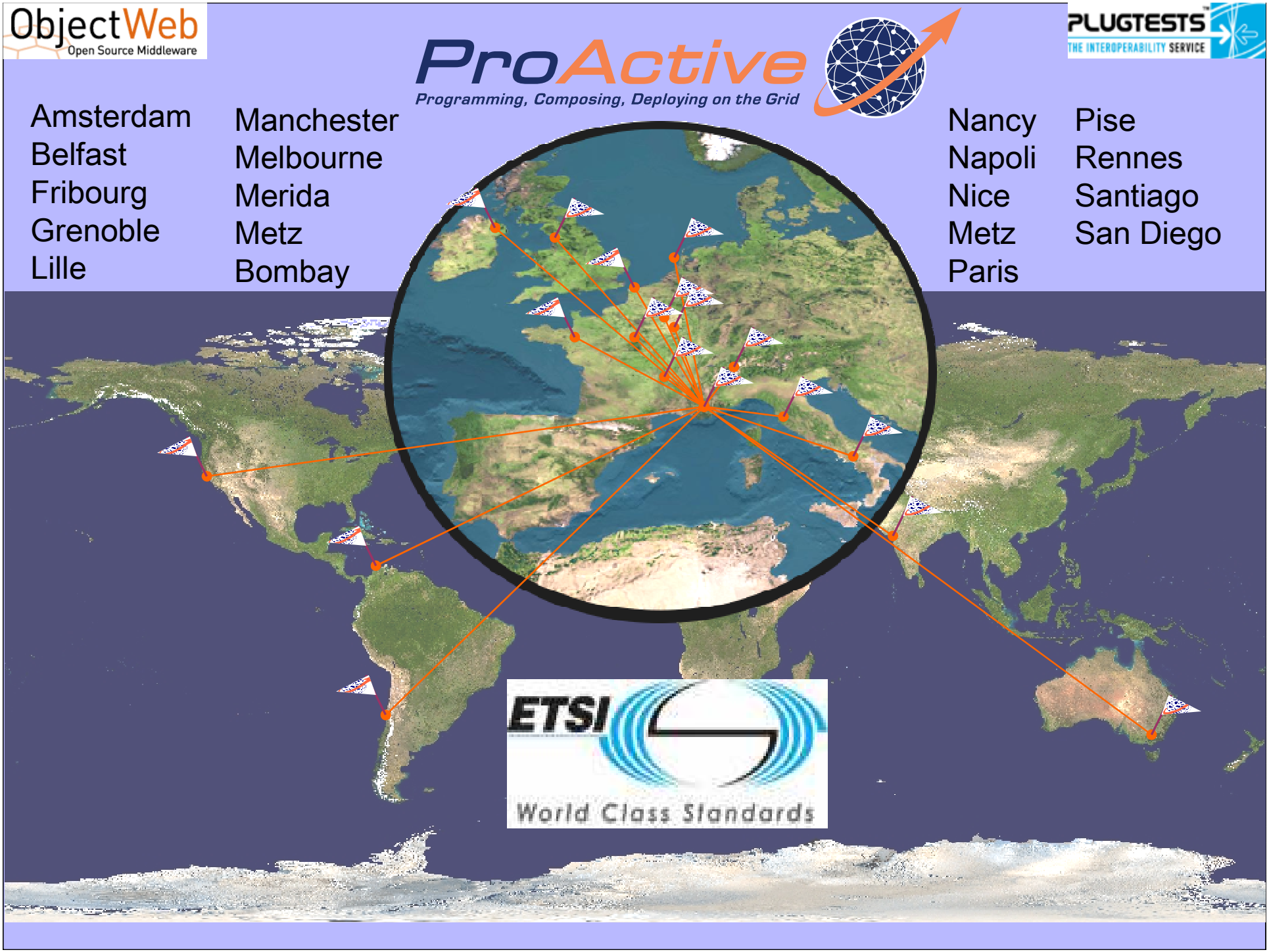


OO SPMD

Step by step
On a N-Body

On-Going:
From Object
to Components

Overall:
Share Slides,
Exercise Sheets,
Experience, …


ObjectWeb
Open Source Middleware
ProActive
Programming, Composing, Deploying on the Grid
PLUGTESTS
THE INTEROPERABILITY SERVICE
Amsterdam
Belfast
Fribourg
Grenoble
Lille
Manchester
Melbourne
Merida
Metz
Bombay
Nancy
Napoli
Nice
Metz
Paris
Pise
Rennes
Santiago
San Diego
ETSI
World Class Standards

2006 Winners :

Eight Samurai - University of Tokyo – JAPAN

The Grid Guru

Conclusions and A Few Directions

ProActive Programming, Composing, Deploying on the Grid

A Strong Programming Model + Components + Env

NOT discussed so far: Web Service, FT, Exceptions

FACTS AND FIGURES

53 years of computation in 6 months in Desktop P2P

Deployed at once on 2111 CPUs (PLUGTESTS on ssh, Globus, LSF, ...)

(Close to) Beating Fortran on an Electromagnetic Application

PERSPECTIVES:

Behavior Specification, Fully Integrated Debugger

A great alchemy for the Grid + SOA:

Asynchrony + Wait By Necessity + Groups + Components

CoreGRID

ObjectWeb Open Source Middleware

GridCOMP Effective Components for the Grids

Theory:

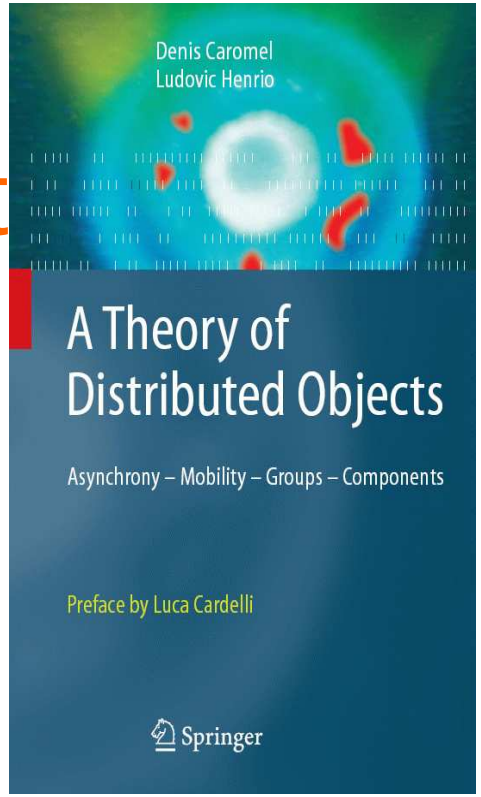
Why is ProActive Adaptive

THEORY

A Theory of Distributed Object

D. Caromel, L. Henrio,

Springer 2005, Monograph



A Calculus:

ASP: Asynchronous Sequential Processes

- **Based on Sigma-Calculus (Abadi-Cardelli)**
- **Formal Proofs of determinism**
- **Releases a few important implementation constraints**

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma')}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P}\ (\text{LOCAL})$$

Local

$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = copy(\iota'',\sigma) \qquad Service = (\text{ if } m_j = \emptyset \text{ then } FifoService \text{ else } \iota''.m_j())\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \\ \longrightarrow \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[Service;\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P\end{array}}\ (\text{NEWACT})$$

Creating an Activity

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \qquad \iota_f \notin dom(\sigma_\alpha) \\ \sigma'_\beta = Copy\&Merge(\sigma_\alpha,\iota' \ ; \ \sigma_\beta,\iota'') \qquad \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}}\ (\text{REQUEST})$$

Sending a Request

$$\frac{R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M,\ m \notin R'}{\alpha[\mathcal{R}[Serve(M)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[\iota.m_j(\iota_r) \Uparrow f,\mathcal{R}[[]];\sigma;\iota;F;R' :: R'';f'] \parallel P}\ (\text{SERVE})$$

$$\frac{\iota' \notin dom(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \qquad \sigma' = Copy\&Merge(\sigma,\iota \ ; \ \sigma,\iota')}{\alpha[\iota \Uparrow (f',a);\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P}\ (\text{ENDSERVICE})$$

Service

$$\frac{\sigma_\alpha(\iota) = fut(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \qquad \sigma'_\alpha = Copy\&Merge(\sigma_\beta,\iota_f \ ; \ \sigma_\alpha,\iota)}{\begin{array}{l}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}}\ (\text{REPLY})$$

Sending a Reply

ASP theory: Summary and Results

ASP $\Rightarrow$ Confluence and Determinacy

Proved Properties:

Future updates can occur at any time, in any order

Asynchronous FIFO point-to-point is enough for Requests

Execution characterized by the order of request senders

Applications:

Determinacy of programs based on a dynamic property (DON)

Determinacy of programs communicating over trees,

Deterministic Components, …

ProActive also in GREEK

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma')}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P} \text{(LOCAL)}$$

$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = copy(\iota'',\sigma) \qquad Service = (\text{ if } m_j = \emptyset \text{ then } FifoService \text{ else } \iota''.m_j())\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \\ \longrightarrow \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[Service;\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P\end{array}} \text{(N}$$

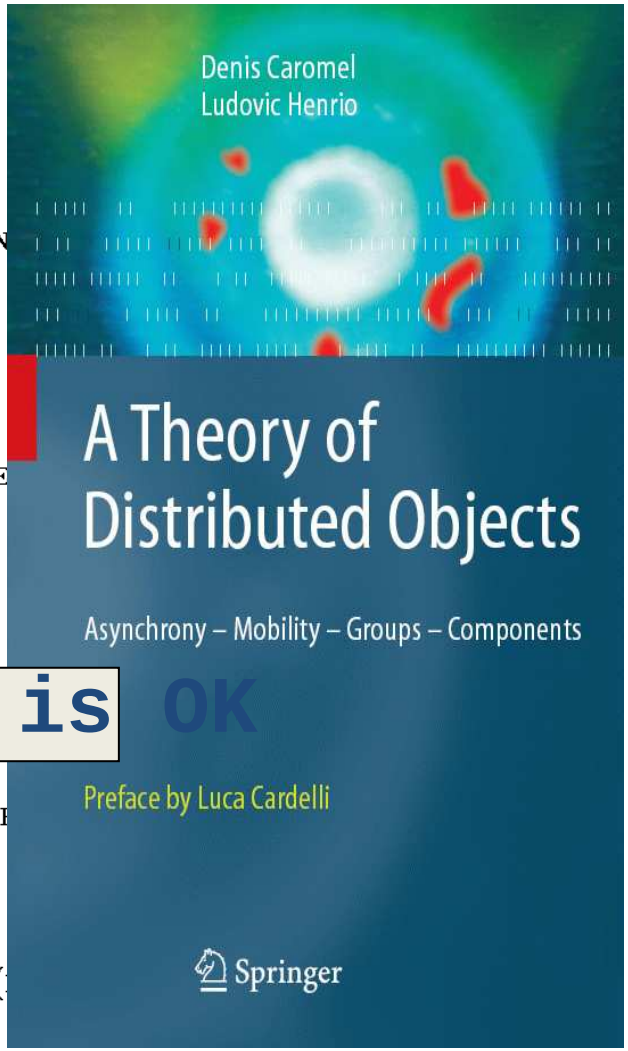
$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \qquad \iota_f \notin dom(\sigma_\alpha) \\ \sigma'_\beta = Copy\&Merge(\sigma_\alpha,\iota' \ ; \ \sigma_\beta,\iota'') \qquad \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}} \text{(RE}$$

$$R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M, m \notin R'$$

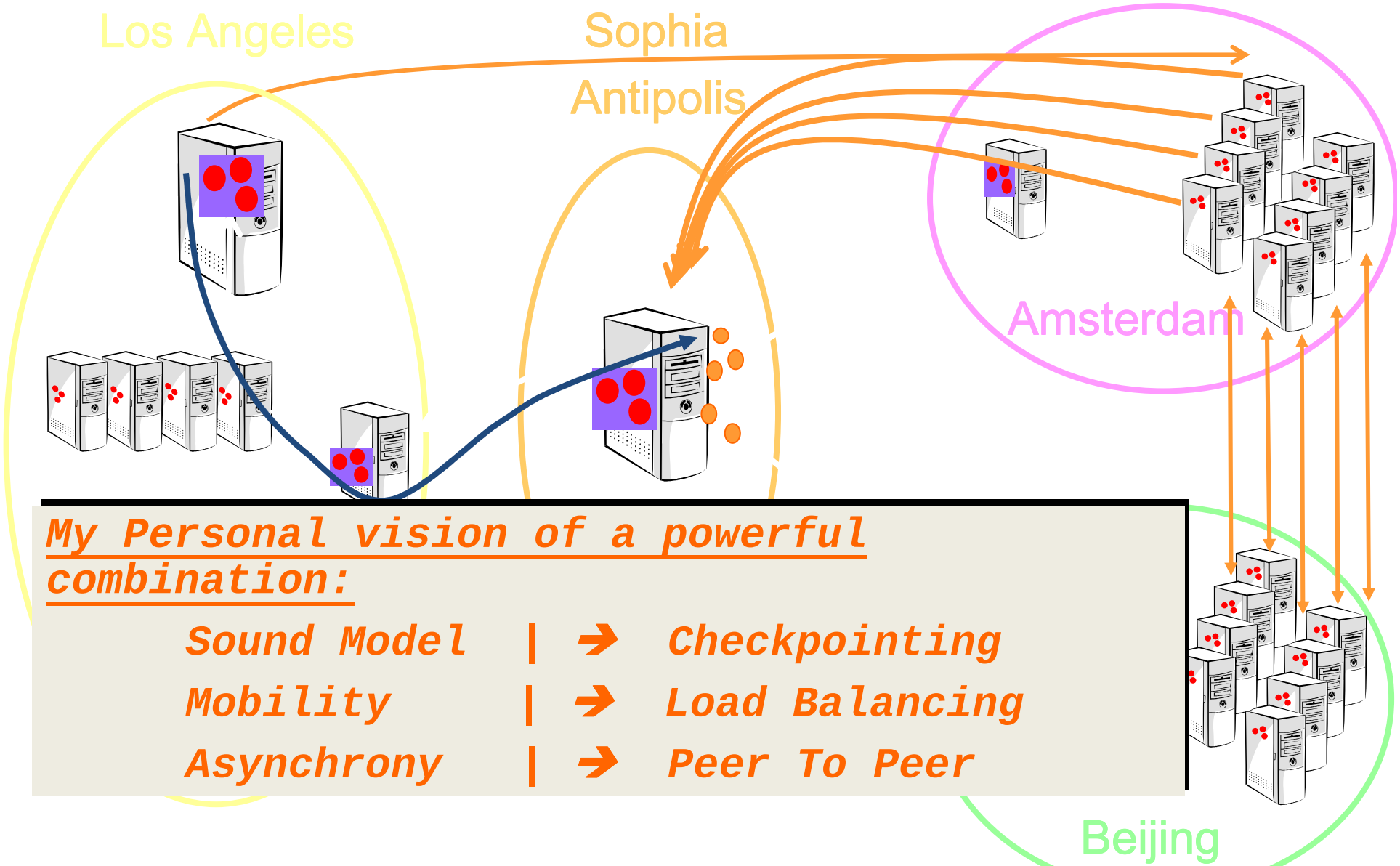
ADAPTIVE implementation is OK

$$\frac{\iota' \notin dom(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \qquad \sigma' = Copy\&Merge(\sigma,\iota \ ; \ \sigma,\iota')}{\alpha[\iota \Uparrow (f',a);\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P} \text{(ENDSE}$$

$$\frac{\sigma_\alpha(\iota) = fut(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \qquad \sigma'_\alpha = Copy\&Merge(\sigma_\beta,\iota_f \ ; \ \sigma_\alpha,\iota)}{\begin{array}{l}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}} \text{(}$$



ProActive Adaptive Potential



My Personal vision of a powerful combination:

Sound Model | ➔ Checkpointing

Mobility | ➔ Load Balancing

Asynchrony | ➔ Peer To Peer

Perspective Adaptive Feature:
Future update strategies

No partial replies and requests:

No passing of futures between activities, more deadlocks

Eager strategies: as soon as a future is computed

Forward-based:

– Each activity is responsible for updating the values of futures it has forwarded

Message-based:

– Each forwarding of future generates a message sent to the computing activity

– The computing activity is responsible for sending the value to all

Mixed strategy:

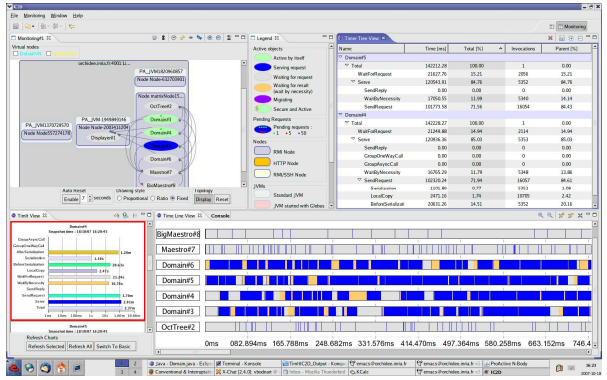
Futures update any time between future computation and WbN

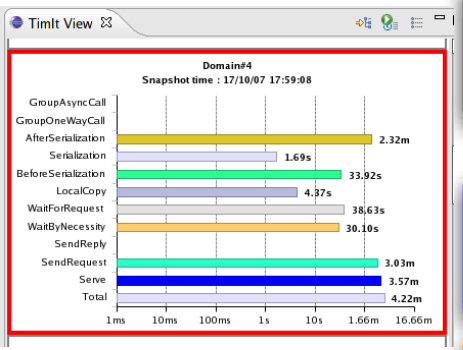
Lazy strategy:

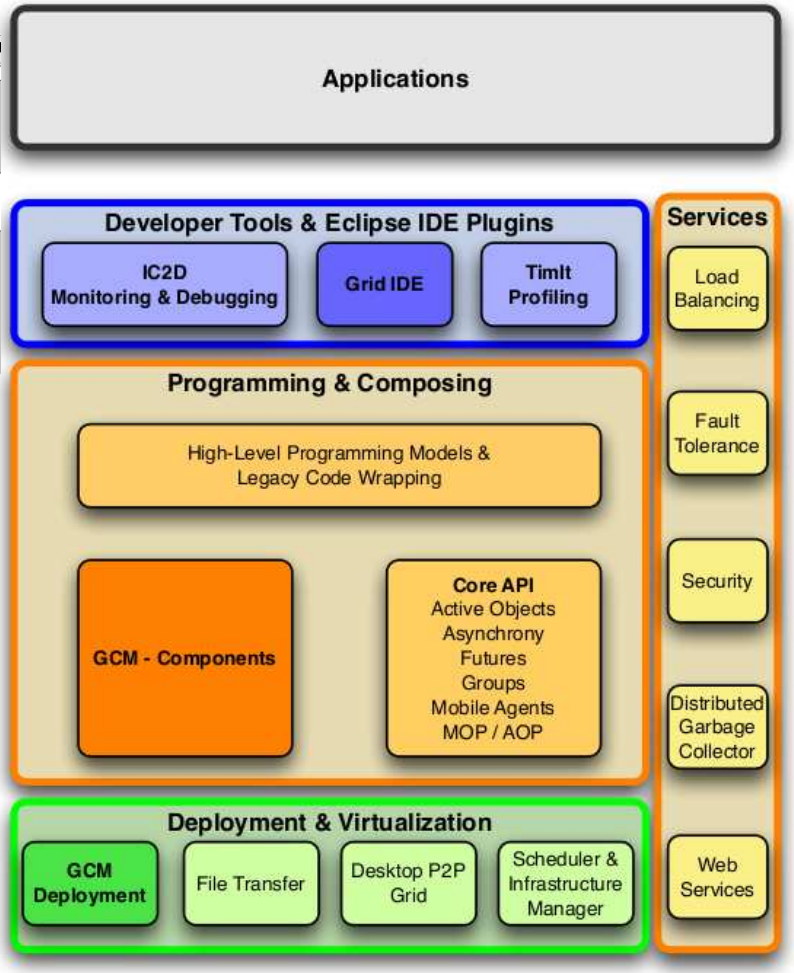
On demand, only when the value of the future is needed (WbN on it)

Conclusions

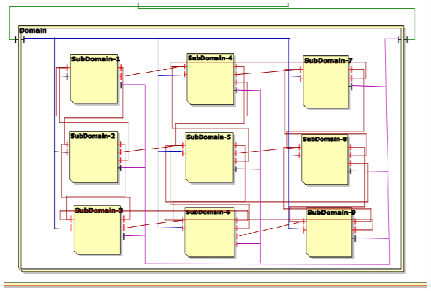
Summary-Perspective: **Full-fletched Upper Ware**

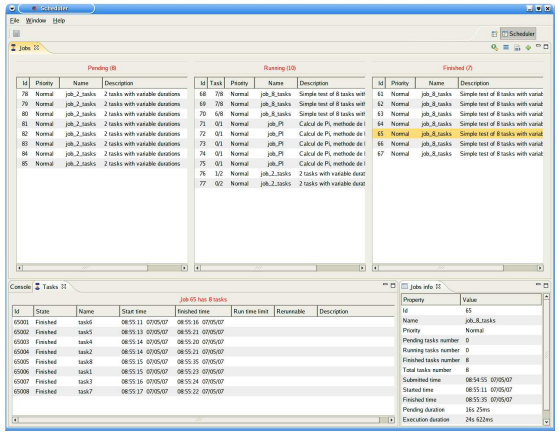


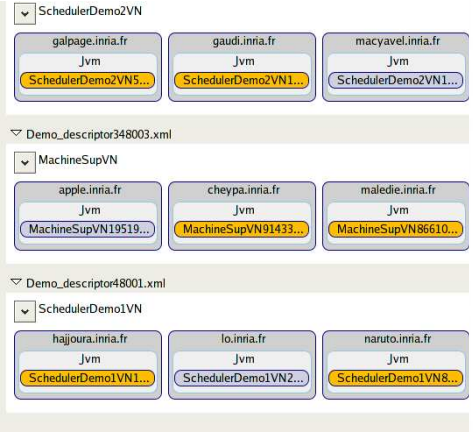




Active Objects
SPMD







ProActive
Programming, Composing, Deploying on the Grid

Upcoming Releases:
November 2007 : ProActive 3.9

Scheduling

- Job Scheduler and Resource Manager:
  - native or Java tasks scheduling and execution
  - unified resource management (Cluster, P2P Desktop, Lan)

Documentation :

- Improved Guided Tour and Full documentation

Programming: New restructured API

- High Level SPMD (beta release)
  - simplified and enhanced SPMD with NAS Benchmarks implementation
- Master / Slave programming framework
- Improved DGC, with automatic Deadlock Detection
- Monitoring the futures for failure detection

IDE

- Extended IC2D Eclipse plugin with
  - JMX integration: JMX control and monitoring through IC2D
  - Applications Profiling

ProActive and *De Facto* Standards

RMI, RMI-Ibis, Jini, HTTP

rsh, ssh, scp

Globus GTx, sshGSI, Unicore, EGEE gLite

LSF, PBS, OAR, Sun Grid Engine

Fractal Components

Web Services

OSGi

Openness

Open Source, under LGPL licence

Easy to add new Deployment Protocols
new File Transfer Protocols

Not so hard to add new Transport
(one could imagine hardware specific implementations)
Possible to add and test Checkpointing Protocols
Possible to add localization strategies for mobility
Adaptive mechanisms … more to come

Conclusions and A Few Directions

ProActive: A Strong Programming Model + A Strong Deployment

Features not mentioned: Fault Tolerance (CIC), Load-Balancing, P2P

PERSPECTIVES:

- Better Error Management (Exceptions, …)
- Behavior Specification (Model Checking)
- Tools for Distributed Programming:
  - Code Analysis, Transformation, Integrated IDE

A Grid Alchemy to further develop:

- Asynchrony / Wait By Necessity + Groups + Components

Conclusion (1)

GRIDs: - Scientific - Enterprise - Internet

Strong convergence in process (infrastructure): WS, WSRF

Challenge: adequate and precise programming+composing model

Asynchronous distributed objects, Mobility, Components

High-level of abstraction, still Open and flexible

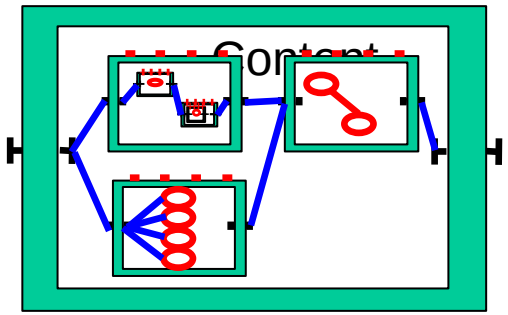
Modern languages: Java, or others

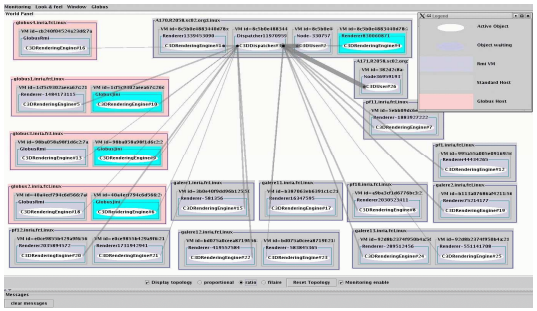
not Fortran, MPI, C++, … not the answer

Perspectives:

Real P2P

Interactive, Graphical, Deployment and Control:

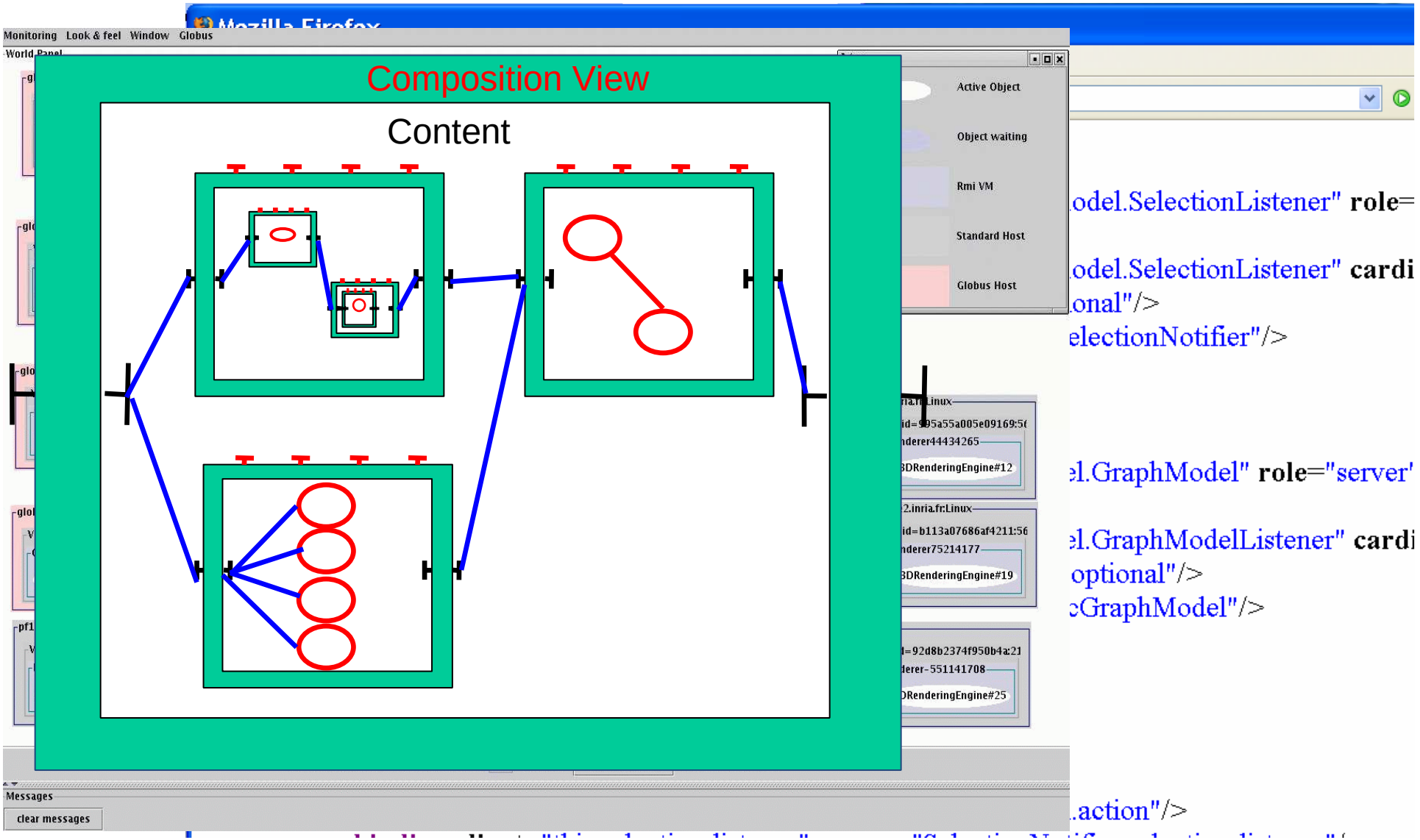




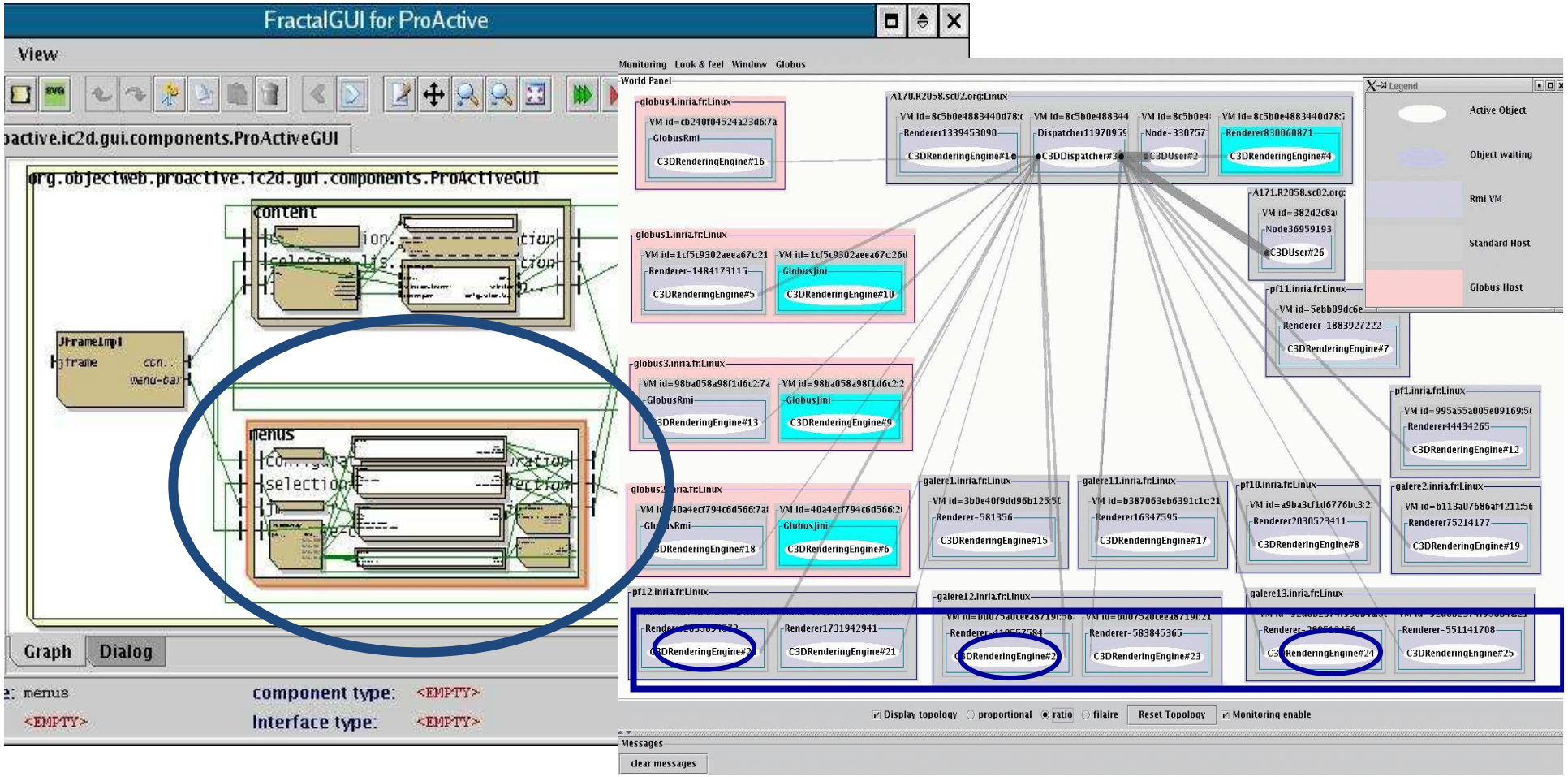
http://ProActive.ObjectWeb.org

Interactive Composition in IC2D

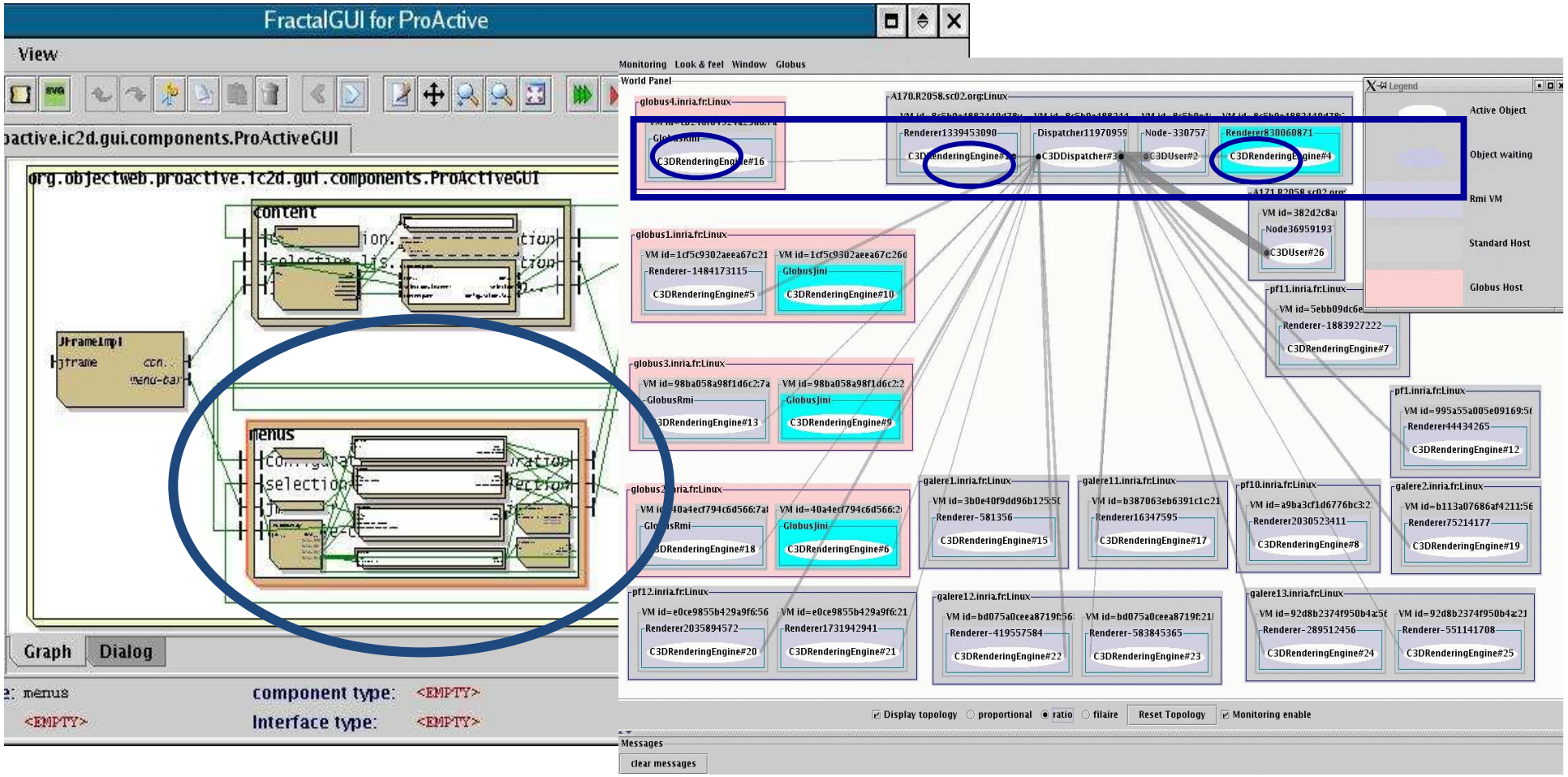
Instead of XML ADL



Perspective for Components - PSE
Graphical Composition, Monitoring, Migration



Perspective for Components - PSE
Graphical Composition, Monitoring, Migration



Conclusion (2)

Async. Distributed Objects

Groups, Components, Mobility

Grid Deployment, Secu, FT

Current Applications and Benchmarks:

- 5 years of computation in 17 days in Desktop P2P
- 53 years of in 6 months, 200 M. spare cycle
- A single application on a 2111 processors GRID

Goal: towards 5 K !

Looking for collaborations:

- Building reusable Cpts from Numerical Codes
- Generic Techniques for Coupling Codes

CoreGRID

Available in LGPL with **ObjectWeb**

http://ProActive.ObjectWeb.org

ObjectWeb
Open Source Middleware

Conclusion (3)

Infrastructure and Deployment

Virtual Nodes and XML

Protocols: ssh, Globus, LSF, PBS, …

Transport: RMI, Ibis (Amsterdam)

Web Services: XML, WSDL, SOAP

OSGi

Conclusion (4): Formal Background

Performance Evaluation for mobility: Markov Chains

- Forwarder strategy
- Server strategy
- Mixed: TTL-TTU

A calculus: ASP: Asynchronous Sequential Processes

- Capture the semantics, and demonstrates the independence of activities
- Results on Confluence and Determinism [POPL 2004]

Conclusion: Why does it scale?

Thanks to a few key features:

- Connection-less, RMI+JMS unified
- Messages rather than long-living interactions

Conclusion: Why does it Compose?

Thanks to a few key features:

Because it Scales: asynchrony !

Because it is Typed: RMI with interfaces !

No unstructured Call Backs and Ports

Very Last Conclusion: Key Aspects

Distributed Objects: Active Objects

Asynchronous Method Calls First-Class Futures

Calculus: ASP

Confluence (very General and Dynamic)

Determinism only based on Request-Sender Reception Order

Dist. Component Specification: GCM

Hierarchical and Runtime (Fractal)

Distributed (VN, …), Multicast, Gathercast

Middleware: ProActive

Programming, Composing, Deploying + GUI

Model Checking: Vercors

Hierarchical, Parameterized,

Practical (Multi. Source for Information, Checking vs. Telling)

Conclusion's Conclusion

Summary:

Programming: **Distributed Objects, OO SPMD**

Composing: **Hierarchical Components**

Deploying: **ssh, Globus, LSF, PBS, …, Web Services**

Application: 3D Electromagnetism on 300 machines at once,
Groups of over 1000!

Goal: **Towards a few 1000s !**

Available in Open Source **ProActive**
http://ProActive.ObjectWeb.org **in**

Conclusions and A Few Directions

***ProActive*:** A Strong Programming Model **+** Components

FACTS AND FIGURES

5 years of computation in 17 days in Desktop P2P

Deployed at once on 2111 CPUs (Plugtests on ssh, Globus, LSF, ...)

(Close to) Beating Fortran on an Electromagnetic Application

PERSPECTIVES FOR COMPONENTS

Safe Reconfiguration

Behavior Specification (SCCC'04, Attali-Barros-Madelaine)

A great alchemy for the Grid:

Asynchrony + Wait By Necessity + Groups + Components

Some More Perspectives

Non-functional Exceptions:

- Exception handler (object) attached to Future, Proxy, AO, JVM, middleware
- Dynamic transmission of handler
- Use to managed Disconnected Mode (e.g. wireless PDA)

ProActive Non Functional Properties

Currently in ProActive:

- Remotely accessible Objects
  - (Classes, not only Interfaces, Dynamic)
- Asynchronous Communications
- First Class Futures: Wait-By-Necessity
- Group Communications, OO SPMD
- Mobility
- Visualization and monitoring (IC2D)
- Fault-Tolerance (checkpointing),
- Load Balancing
- Security
- Components
- Non-Functional Exceptions: Handler reification (prototype)

ProActive and Adaptive

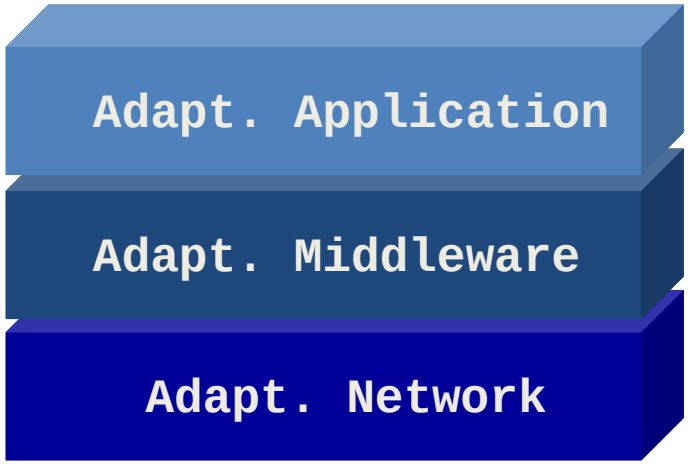
5 Adaptive/Parameterized features in ***ProActive***:

- RMI <--> ssh/RMI … HTTP - Ibis/TCP - Ibis/Myrinet - ...
- Groups, Localization in Mobility, Security, Future Update

Perspectives -- On-going work:

- Adaptive Components:
  - Tensioning
  - Re-configuration
- Adaptive Checkpointing

Better off with simple Functional RMI / Two-sided MPI Message PassingS

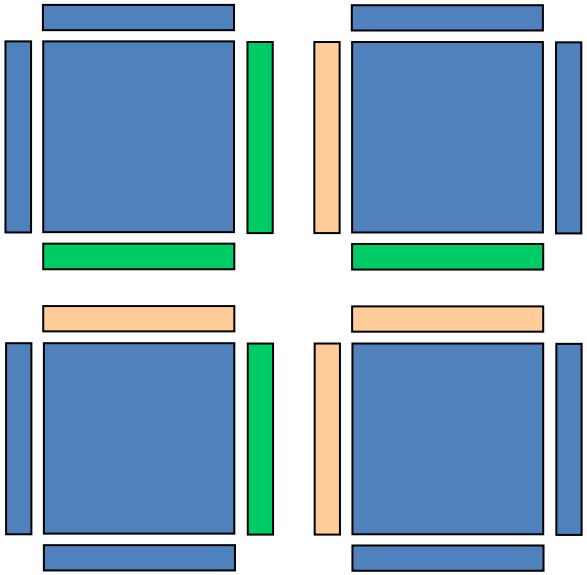


Lets just be careful:
otherwise, we'll just build …

Maladaptive Adaptive Grids

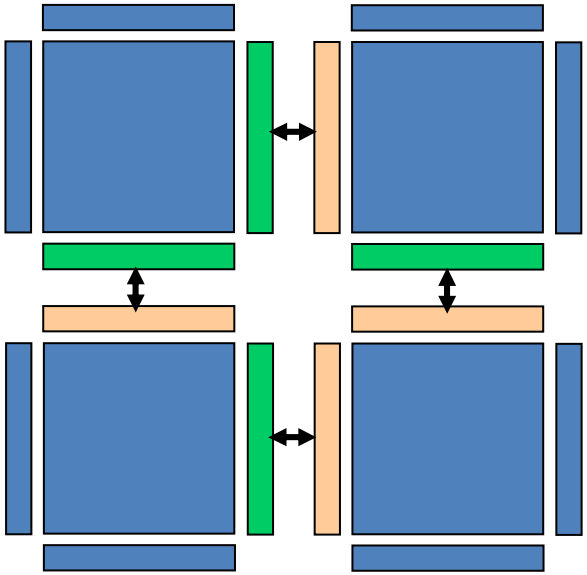
TCP is an Adaptive Middleware

Jacobi: MPI



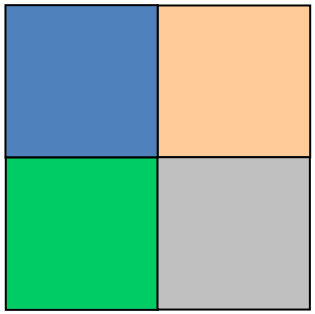
```
! Matrix init
…

Call mpi_barrier(comm2d,ierr)

it = 0

Do while ( diffnorm > tolerance )

      it = it + 1

      Call exchng2( a,sx,ex,sy,ey,comm2d,
     belowX,aboveX,belowY,aboveY,dims)

      Call sweep2d(a,f,nx,sx,ex,sy,ey,b)

      dwork = diff(a,b,sx,ex,sy,ey)

      Call mpi_allreduce(
     dwork,diffnorm,1,mpi_double,mpi_sum,comm2d,ierr)

End Do
```

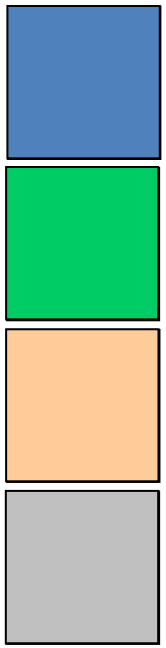
Jacobi: MPI



```
Subroutine exchng2(
    ab,sx,ex,sy,ey,comm2d,belowX,aboveX,belowY,aboveY,dims)

If ( dims(1) > 1 ) then
    Call mpi_sendrecv(
       ab(ex,sy:ey),ey-sy+1,mpi_double,      aboveX,0,    &
       ab(sx-1,sy:ey),ey-sy+1,mpi_double,    belowX,0,    &
       comm2d,status,ierr)

    Call mpi_sendrecv(
       ab(sx,sy:ey),ey-sy+1,mpi_double,      belowX,1,    &
       ab(ex+1,sy:ey),ey-sy+1,mpi_double,    aboveX,1,    &
       comm2d,status,ierr)
End If

If ( dims(2) > 1 ) then
    Call mpi_sendrecv(
       ab(sx:ex,ey),ex-sx+1,mpi_double,      aboveY,2,    &
       ab(sx:ex,sy-1),ex-sx+1,mpi_double,    belowY,2,    &
       comm2d,status,ierr)

    Call mpi_sendrecv(
       ab(sx:ex,sy),ex-sx+1,mpi_double,      belowY,3,    &
       ab(sx:ex,ey+1),ex-sx+1,mpi_double,    aboveY,3,    &
       comm2d,status,ierr)
End If
End Subroutine exchng2
```

Jacobi: OpenMP

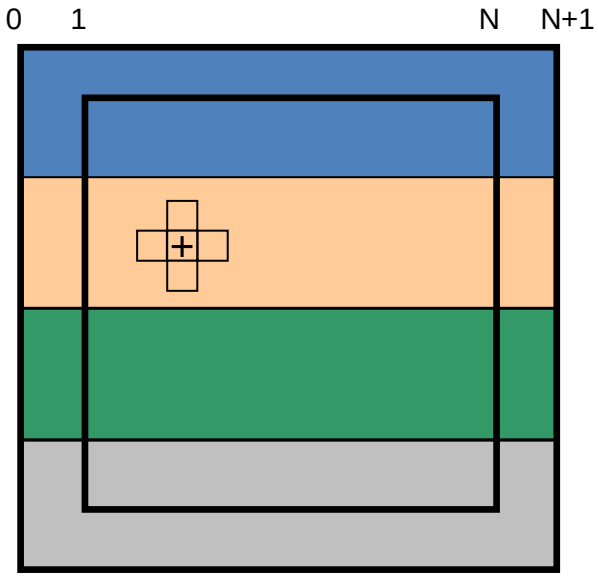




error

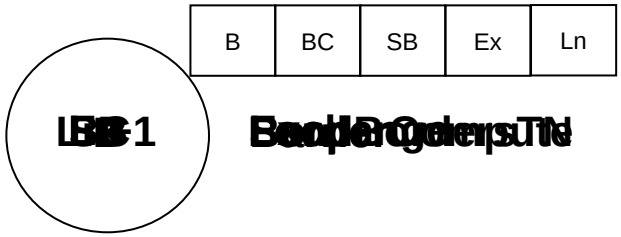
```
#pragma omp parallel private(resid, i) {
  while (k <= maxit && error > tol) {
    /* copy new solution into old */
    #pragma omp for
      for (j=0; j<m; j++)
          for (i=0; i<n; i++)
              uold[i + m*j] = u[i + m*j];

    #pragma omp for reduction(+:error)
      /** Compute **/
      for (j=1; j<m-1; j++)
           for (i=1; i<n-1; i++){
                u[i + m*j] =( uold[i-1 + m*j] + uold[i+1 + m*j]
                      + uold[i + m*(j-1)] + uold[i + m*(j+1)] )/ 4;

                /* accumulate residual error */
                error = error + u[i +m*j];
           } /* end for */

    /* error check */
    #pragma omp master
      k++;
      error = sqrt(error) /(n*m);

  } /* end while */
} /* end pragma parallel */
```

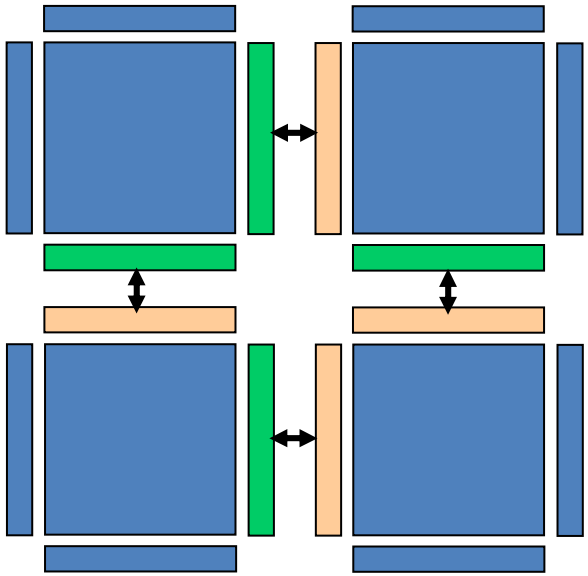
Jacobi: X10




```
const region R = [0:N+1, 0:N+1,    RInner= [1:N, 1:N];
const distribution D = distribution.factory.block(R);
const distribution DInner = D | RInner,    DBoundary = D - RInner;

double[D] B = new double[D] (point p[i,j])
   { return DBoundary.contains(p) ? (N-1)/2 : N*(i-1)+(j-1); };

public boolean run() {
   while(true) {
        double[.] Temp = new double[DInner] (point [i,j])
     { return (get(i+1,j)+get(i-1,j)+get(i,j+1)+get(i,j-1))/4.0; };

        if((err=((B | DInner) - Temp).abs().sum()) < epsilon)
              break;

        B.update(Temp);
    }
    return abs(err)<epsilon2 && iters==ITERMAX;
}

public double get(final int i, final int j) {
    return future(D[i,j]) B[i,j].force();
}
```

Jacobi: ProActive





Loop();

```
this.internalCompute();

// synchronization
ProSPMD.barrier("SynchNeighbors" + iter, neighbors);

// compute the border values
this.asyncRefToMe.borderCompute();

// send the borders to neighbors
this.asyncRefToMe.sendBordersToNeighbors();

// continue or stop ?
if ((iter > 0) && (this.minDiff > Jacobi.MINDIFF))
        this.asyncRefToMe.exchange();
        this.asyncRefToMe.loop();

if (this.minDiff < Jacobi.MINDIFF)
        this.matrix.stop(); // end
```

borderCompute();

sendBordersToNeighbors();

exchange();

Parallelism: Problem / Solution

Embarrassingly Parallel Applications	• Independent Tasks --> Master Slave package *Monte Carlo Simulation (in Financial Math, Non Linear Physic, ...)* • Dynamic Generation of Tasks --> High-Level Patterns *Post-Production*
Slightly Communicating Applications	• Branch & Bound package Flow-Shop
Highly Communicating Applications	• Dynamic,Numerical --> OO SPMD *Electro-Magnetism, Vibro-Acoustic, Fluid Dynamics* • Unstructured --> Basic API with Active Objects and Groups *EDA (Electronic Design Automation for ICs), N-Body*
Non-Java code	• Numerical MPI --> Legacy Code Wrapping • Script code --> Script Legacy Code Wrapping

Q & A

Typical Questions
about Parallelism, Multi-Threading

Criteria to select multi-thread (Shared-Memory) versus multi-process (Message Passing)

Before all :

Message Passing : High overhead (network dependant) and abstract system design
Shared Memory : Faster and can be more intuitive (at the beginning)

Because threads share the same memory space, a runaway thread can corrupt the entire system by writing to the wrong part of memory.
With separate processes, the only data that is shared is that which is meant to be sent, which reduces the potential for such damage.

Various cases:

- If the application is well parallelized and can be highly scalable : MP; it's easier to add an extra machine than a extra SM processor, same for the memory.
- If the application contains a lot of inter-dependant tasks and is performance sensitive: a SMP can be a better choice.
- If the application is fault sensitive : in case of a general machine crash on a SM, all cores are stopped and we need to restart all the application. If the same happens in a Message Passing system the fault is local to a single machine therefore there is only one task to restart.
- If the application is an embedded one : The SM is a natural choice (but more and more complex embedded systems with communications.

In parallelization, what are the most common mistakes done by developers ?

Blocking calls is an important source of deadlocks :

"One of the most common mistakes made by MPI users is to assume that an MPI implementation provides some amount of message buffering. To buffer a message is to make a temporary copy of a message between its source and destination buffers. This allows MPI_Send to return control to the caller before a corresponding MPI_Recv has been called. In this way, it is possible for two processes to exchange data by first calling MPI_Send and then calling MPI_Recv.''

MPI does not require any message buffering, and portable applications must not rely on it. To avoid deadlock conditions, applications should use the non-blocking routines MPI_Isend and/or MPI_Irecv. It is common for programs ported from PVM or applications that use message ``portability'' packages to make assumptions about buffering."

In multithread, how to minimize the cost of locks?

The most obvious answer is to keep them locked for the shortest possible time.

Some lock profiling analysis (exists in Java, difficult to get in C, C++ because locks are not part of the language, exists for the linux kernel: ***lockmeter***) should reveal the most contended locks, which should be split into finer grained ones.

This should take into account the added complexity (deadlocks) and overhead of atomic operations (at the micro optimization level: processor atomic style "**lock cmpxchg**", with which locks are implemented, implies a synchronization of all processors on an SMP or multi-core, so it is a slow operation that slows down everybody.

Mostly read data structures may benefit from read-write locks, this time taking into account the possible starvation of readers or writers (according to priority). Seqlocks may also be considered if the readers / to writers ratio is big enough (many readers).

For an advanced optimization level:

- recursive locks (reentrancy) may be forbidden to avoid useless lock acquisitions, provided enough discipline is applied to avoid deadlocks.

- Also, locks acquired in sequence should be released in the reverse sequence to avoid useless wake-ups when the locks become sequentially available.

When to use Components?

ProActive and *SOA Integration*

SOA Main Principles

- Loosely coupled software services t
  processes (Flexibility, Portability)
- A ure is r
- I cations
  i
- V
- S
- S on a f
  I
- I techno
- S d by the
- C ring of

SOA:

An architecture to Loosely Couple applications as services

ProActive:

A middleware to Program Tightly Coupled //, Dist, M.-Threaded applications and seamlessly integrate in SOA

ProActive and *SOA Integration*