Programmatic Access to a Compute Utility

Murali Kaundinya

Senior Staff Engineer
Sun Microsystems, Inc.
http://www.sun.com

TS-5622

Copyright © 2006, Sun Microsystems Inc., All rights reserved.

Designing Middleware for Grid Services!

Enabling client applications to consume grid services

Learn to architect and develop flexible and extensible middleware across insecure networks

Agenda

Motivations for Utility Computing
Compute Utility
Programming Model
Communication Model
Jini™ ERI
Interesting Technologies
Summary

Agenda

Motivations for Utility Computing

Compute Utility

Programming Model

Communication Model

Jini™ ERI

Interesting Technologies

Summary

Motivations for Utility Computing

HPC, grid and enterprise computing

- HPC/grid communities pursue grand challenges
  - Clusters, internetworking, algorithms, etc.
- Enterprises leverage technology for business
  - Usability, commerce
- IT vendors like Sun, etc.
  - Jini, N1™ software, Solaris™ OS, www.network.com
- Increasing convergence
  - Continuum of computing
  - “Localized everything”→“Everything networked”

Motivations for Utility Computing

Are we at a tipping point with grids in enterprises?

- Service oriented architecture/infrastructure
  - Composable, virtualized, policy-based
- Industry standards
- Software as a service
  - Rearden Commerce, Salesforce.com
- Can computing be a service?
  - CMT, Solaris OS Zones, Java™ technology Isolates
  - Java technology/CLR

Enterprise Grid Alliance

GGF

DMTF distributed management task force, inc.

salesforce.com Success. Not Software.

Hewitt

REARDEN commerce

Sun microsystems

Data Synapse

Tsunamic Technologies Inc.

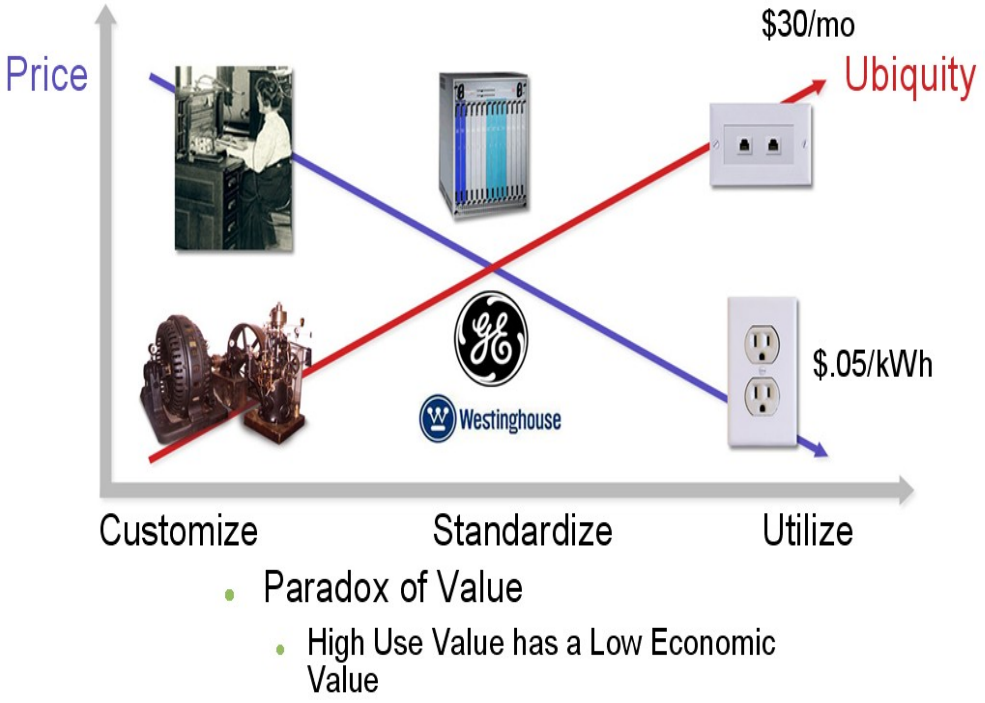
PENGUIN COMPUTING®

SOFTRICITY

Motivations for Utility Computing

All industries move from custom to utility models

- Data center challenges
- Space
- Cooling
- Capacity on demand
- Complexity
  - Servers
  - Desktops
- Staying current
- Licensing
- Costs



- Paradox of Value
  - High Use Value has a Low Economic Value

Agenda

Motivations for Utility Computing

Compute Utility

Programming Model

Communication Model

Jini ERI

Interesting Technologies

Summary

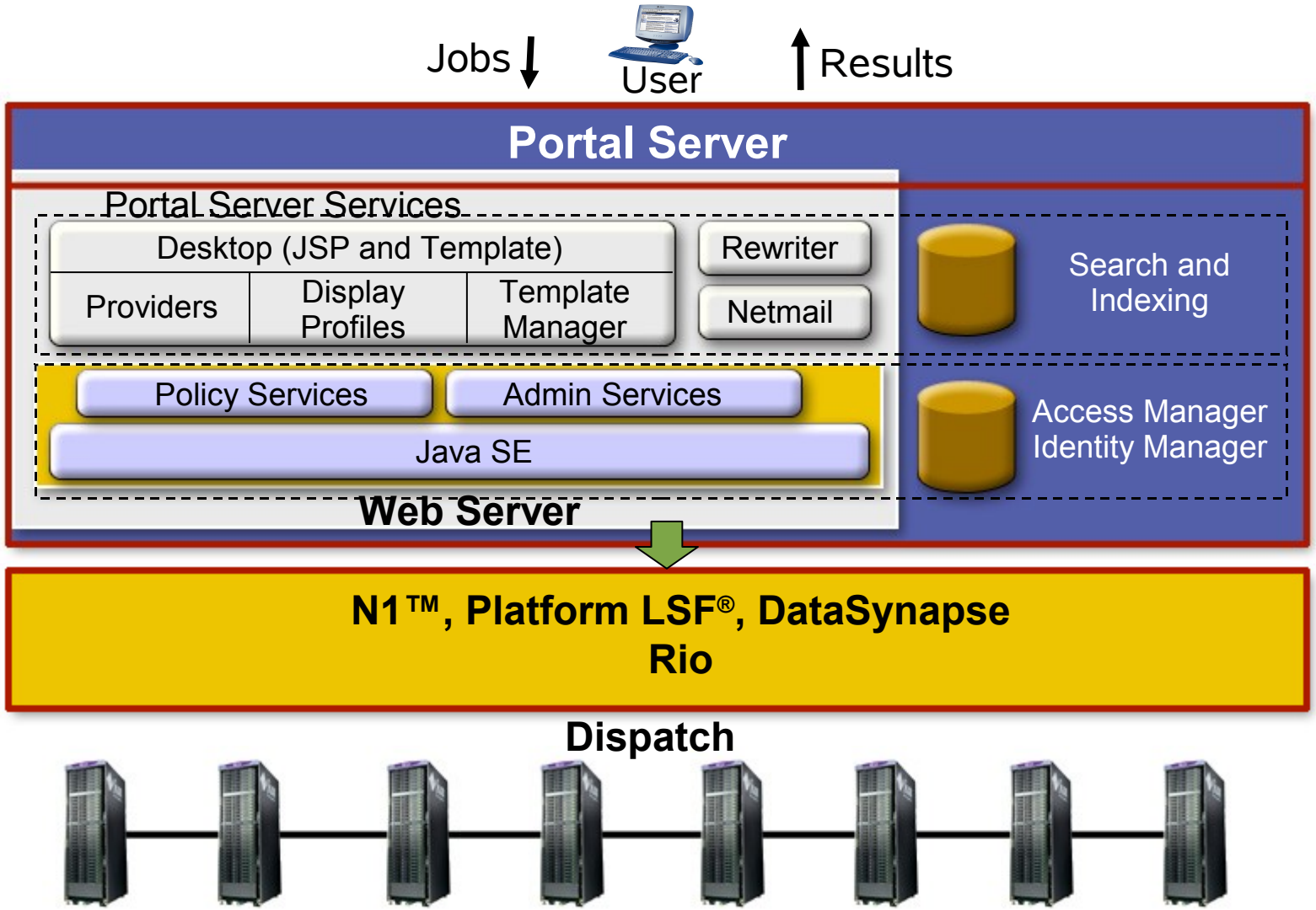
Compute Utility: What's Out There?

Source: http://www.gridcomputing.com

- Consortiums (9)
- Middleware (15)
- Data Grid Access Initiatives (11)
- Grid Schedulers (15)
- Grid Portals (11)
- Grid Programming Environments (15)
- Grid Testbeds and Developments (35)

Compute Utility

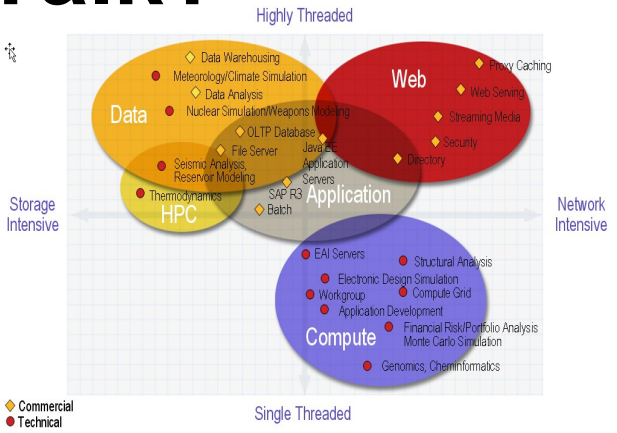
A prototypical architecture

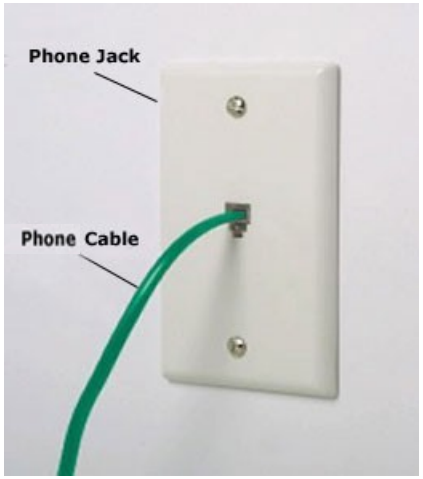


Compute Utility: Why this Talk?

Why do we need programmatic access?

- Users shouldn't feel different
  - Seamless
- Composability
  - Clients have a job to run
  - HTML/Batch is not easy to compose
- Upload/download of large files
- Overflow into grid
  - Dispatch threads to grid
  - Transactions
  - Java technology and cluster computing





Compute Utility

Architectural requirements for programmatic access

- Programming model
- Communication model
  - Asynchronous
- Security model
- Failure model
  - Events and notification
- QoS
  - Reliable, available, scalable

Agenda

Motivations for Utility Computing

Compute Utility

Programming Model

Communication Model

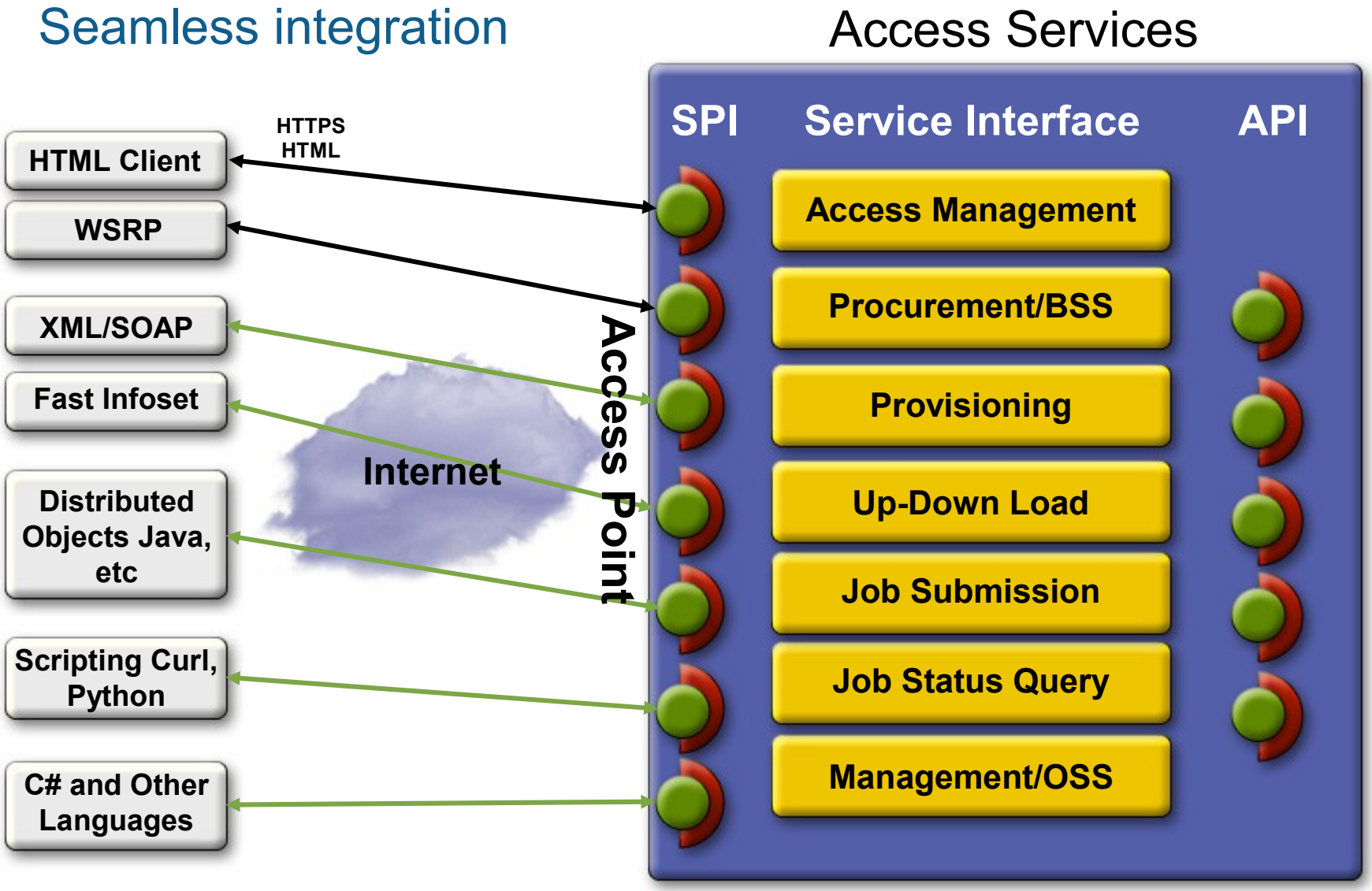
Jini ERI

Interesting Technologies

Summary

Programming Model

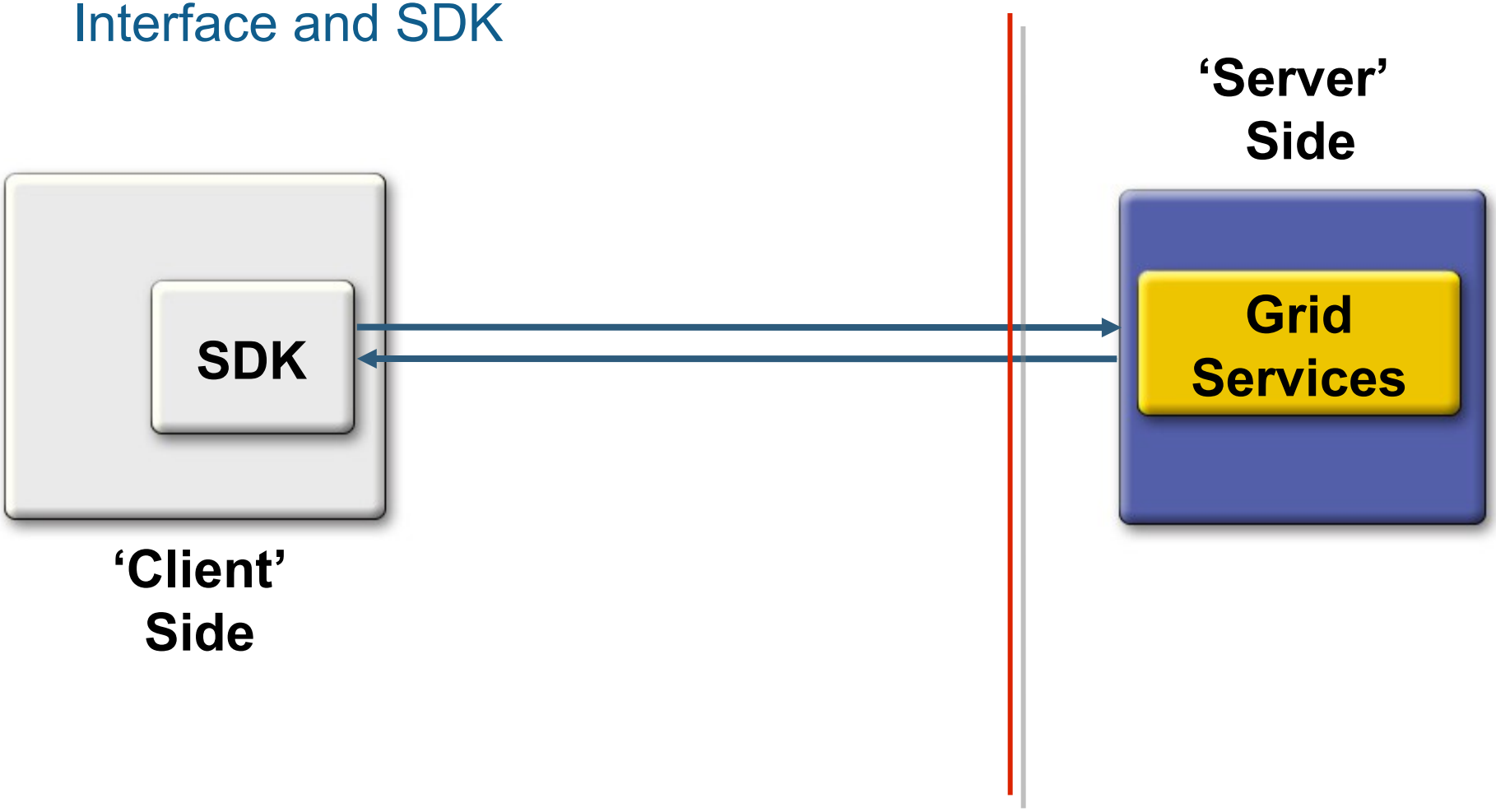
Seamless integration



Programming Model

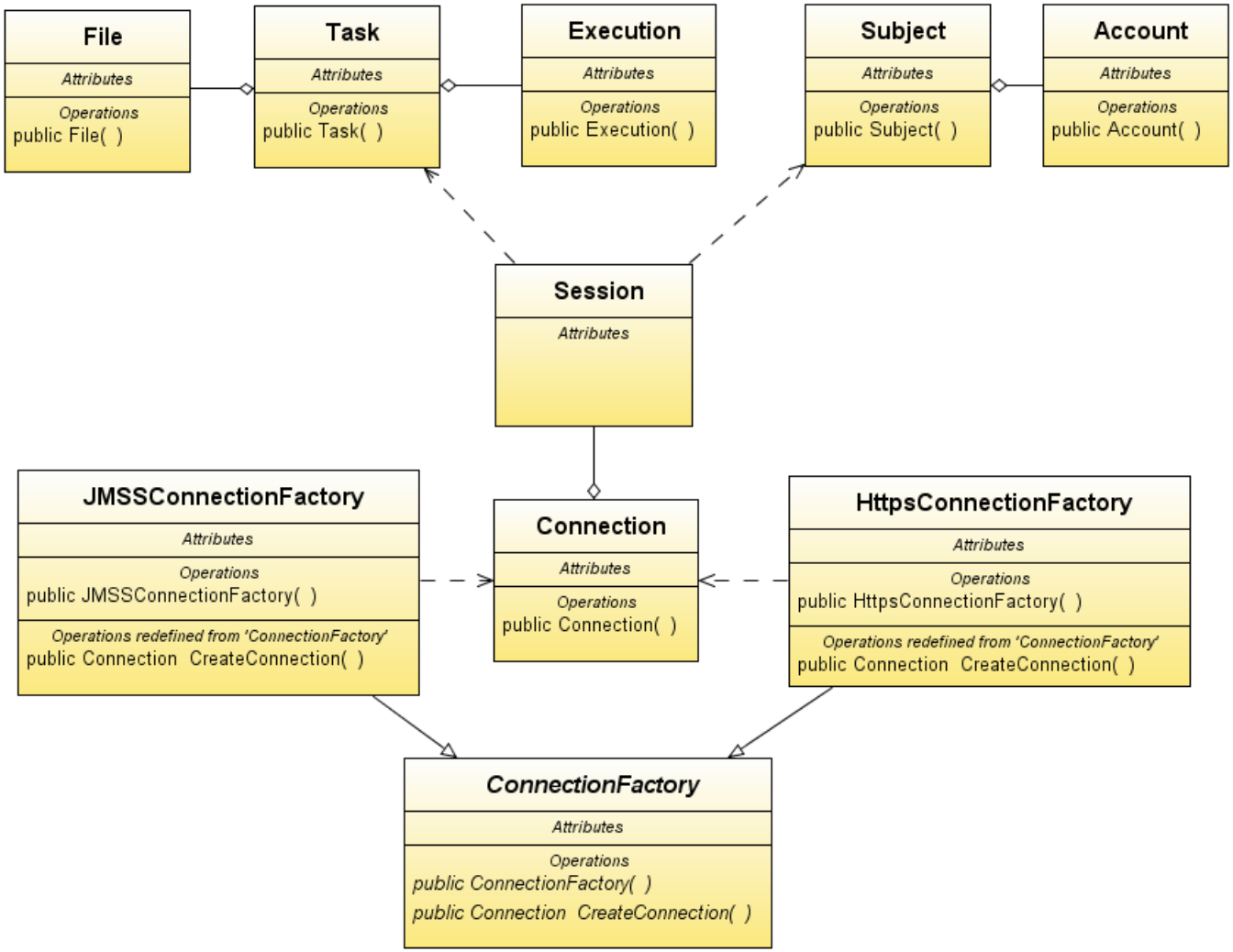
Seamless integration

Interface and SDK



Programming Model

Domain Model



Programming Model

```
// Connection Management Services
ConnectionFactory cf = new HttpsConnectionFactory() ;
Connection conn = cf.createConn() throws Exception;

// Session Management Services
Session s = conn.createSession() throws Exception;

// Account Management Services
Account s.buy(int cpuHours) throws Exception;
Account s.queryAccount() throws Exception;

// File Management Services
File s.downloadFile(String fileId) throws Exception;
void s.uploadFile(File file) throws Exception;

// Task Management Services
Task s.createTask(Task task) throws Exception;
Task s.runTask(String taskId) throws Exception;
void s.deleteTask(String taskId) throws Exception;
```

Communication Model

RMI/JRMP (Java Technology Remote Method Protocol)

```
// Service Interface
import java.rmi.*;

public interface GridSvc extends Remote(
       public int add(int i, int j);
}


// Service Implementation
import java.rmi.*;
public class GridSvcImpl extends UnicastRemoteObject
implements GridSvc {
       public int add(int i, int j) {
              int k = i+j;
              return k;
       }
}
```

Communication Model

RMI/JRMP

```
// Server
public class GridServer {
       public static void main() {
       // Server starting an RMI Registry
       Process svcReg =
Runtime.getRuntime().exec("rmiRegistry");
       // Create Grid Service & Bind it with a Registry
       GridSvcImpl gsi = new GridSvcImpl(svcReg);
       Naming.rebind("GridSvc", gsi);
       …
}

// Client Side
public class GridClient {
       Object obj = Naming.lookup("rmi://yourURLhere/GridSvc");

       int i = 1; j = 2;
       System.out.println(obj.add(i, j));
}
```

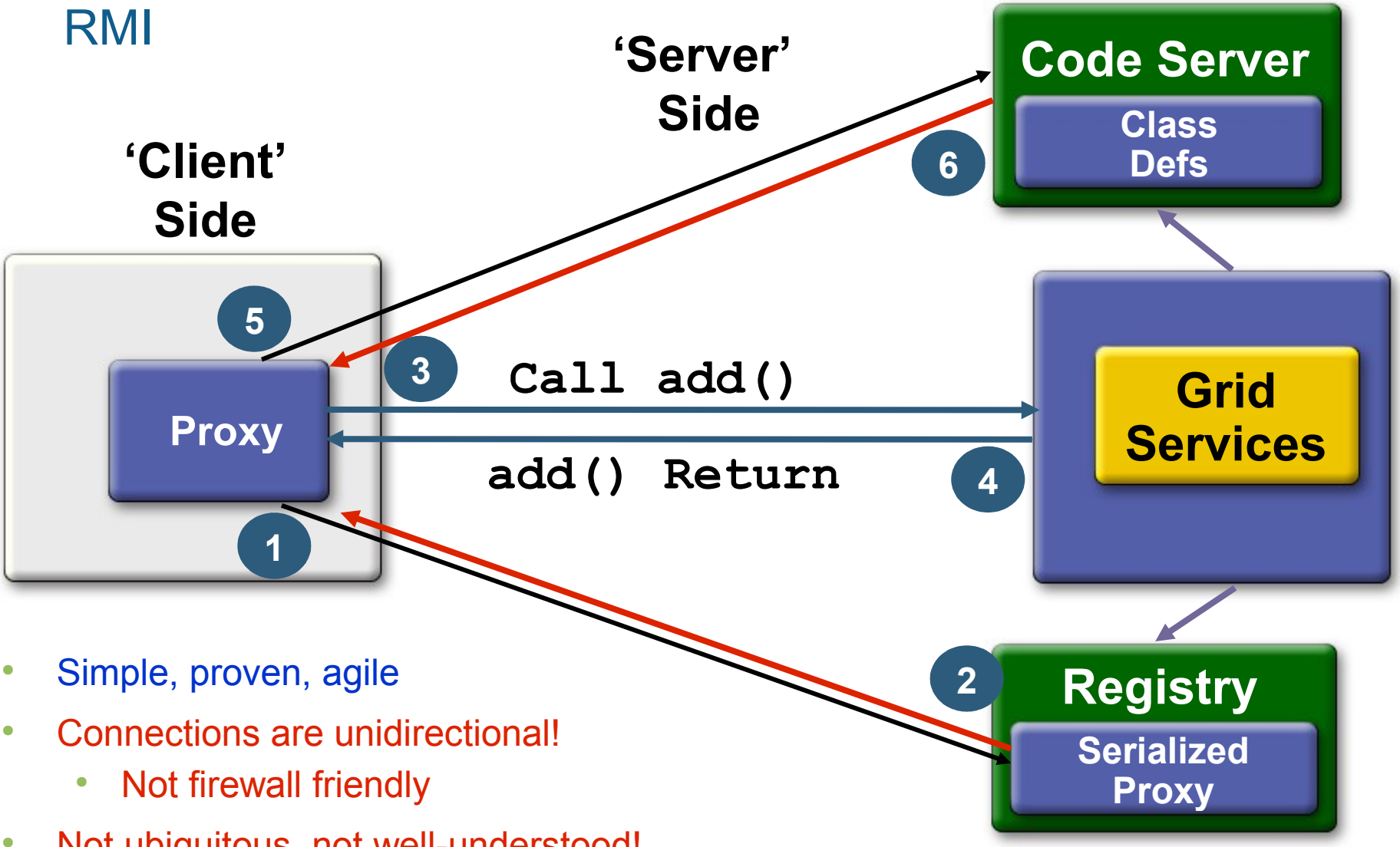
Communication Model

RMI over SSL

```
// Server
public class GridServer {
    public static void main() {
    // Create Grid Service
    GridSvcImpl gsi = new GridSvcImpl();
    RMIClientSocketFactory csf = new
GridSvcClientSocketFactory(…);
    RMIServerSocketFactory ssf = new
GridSvdServerSocketFactory(…);
    GridSvc stub = (GridSvc)
UnicastRemoteObject.exportObject(gsi, 0, csf, ssf);
     // Server starting an RMI Registry
LocateRegistry.createRegistry(2002);
    Registry registry = LocateRegistry.getRegistry(2002);
    // Bind Stub to the registry
    registry.rebind("GridSvc", stub);
}
```

Communication Model

RMI



- Simple, proven, agile
- Connections are unidirectional!
  - Not firewall friendly
- Not ubiquitous, not well-understood!

Communication Model

Web Services: XML over SSL

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="add">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="arg1" /> <xs:element ref="arg2" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="arg1"> <xs:complexType mixed="true" />
 </xs:element>
 <xs:element name="arg2"> <xs:complexType mixed="true" />
 </xs:element>
 <xs:element name="compute">
   <xs:complexType>
     <xs:sequence> <xs:element ref="add" /></xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Communication Model

Web Services: XML over SSL

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="“http://server/GridAdd.xsd">
        <add>
                <arg1>1</arg1>
                <arg2>2</arg2>
        </add>
</compute>
```

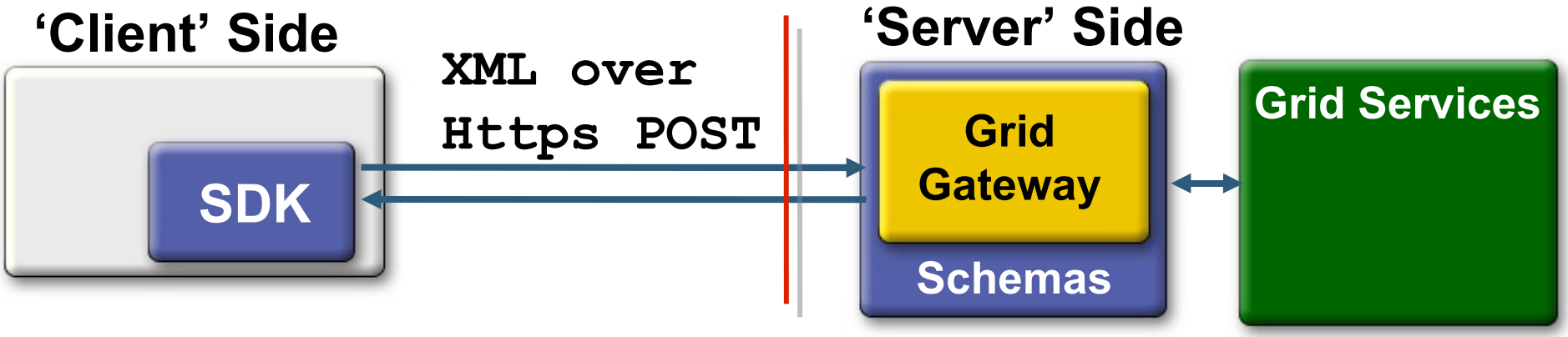
Communication Model

Web Services: XML over SSL

```
// Apache Commons HttpClient App.
PostMethod post = new PostMethod(myURL);
post.setRequestEntity(new
InputStreamRequestEntity(
     new FileInputStream(in), in.length()));
post.setRequestHeader("Content-type",
"text/xml; charset=ISO-8859-1");
httpclient = new HttpClient();
try {
 int result = httpclient.executeMethod(post);
}
finally {post.releaseConnection(); }
```

Communication Model

Web Services: XML over SSL



- Simple, proven, ubiquitous
- Schemas for every object/message!
- Client and server share (de)serialization!
  - XML tools and libraries
- Not naturally asynchronous!

Communication Model

Synchronous SOAP over Https

```
// Service based on JAX-WS
import javax.jws.WebService;
@WebService()
public class Compute {
        public Compute() {}

        @WebMethod()
        public int add() {int i, int j) {
                return i+j;
        }
}
```

Communication Model

Synchronous SOAP over Https

```
// jax-ws imports
import javax.xml.ws.WebServiceRef;
import WSGrid.endpoint.ComputeService;
import WSGrid.endpoint.Compute;

public class GridClient {
@WebServiceRef(wsdlLocation="http://localhost:8080/WSGr
id/Compute?wsdl");
static ComputeService cs;
…
public void testWSGrid() {
      // Retrieve a proxy to the service
      Compute port = cs.getComputePort();
      int arg1=1; int arg2=2;
      Sytem.out.println(port.add(arg1, arg2));
}
```

Communication Model

Asynchronous SOAP over Https

```
// Service based on JAX-WS
import javax.jws.WebService;
@WebService()
public class Compute {
        @WebMethod()
        public Response<AddResponse> addPoll() {int i, int j) {
                …
        }
        @WebMethod()
        public Future<?> addCB(int i, int j,
                        AsyncHandler<Integer> handler) {
                …
        }
}
```

Communication Model

Asynchronous SOAP over Https with Polling

```
public class GridClient {
@WebServiceRef(wsdlLocation="http://localhost:8080/WSGr
id/Compute?wsdl");
static ComputeService cs;
…
public void testWSGrid() {
      // Retrieve a proxy to the service
      Compute port = cs.getComputePort();
      Response<AddResponse> resp =  port.addPoll(1,2);
      while(!resp.isDone()){ //do something }
      System.out.println("The sum is: " +
      resp.get().getReturn());
      ...
}
```

Communication Model

Asynchronous SOAP over Https with Callback

```
class AddCallbackHandler implements AsyncHandler<AddResponse>{
        private AddResponse output;
        public void handleResponse(Response<AddResponse> resp) {
                try {
                        output = resp.get();
                } catch (ExecutionException e) {
                        e.printStackTrace();
                }
                catch (InterruptedException e) {
        e.printStackTrace();
                }
        }
        AddResponse getResponse() {
                return output;
        }
}
```

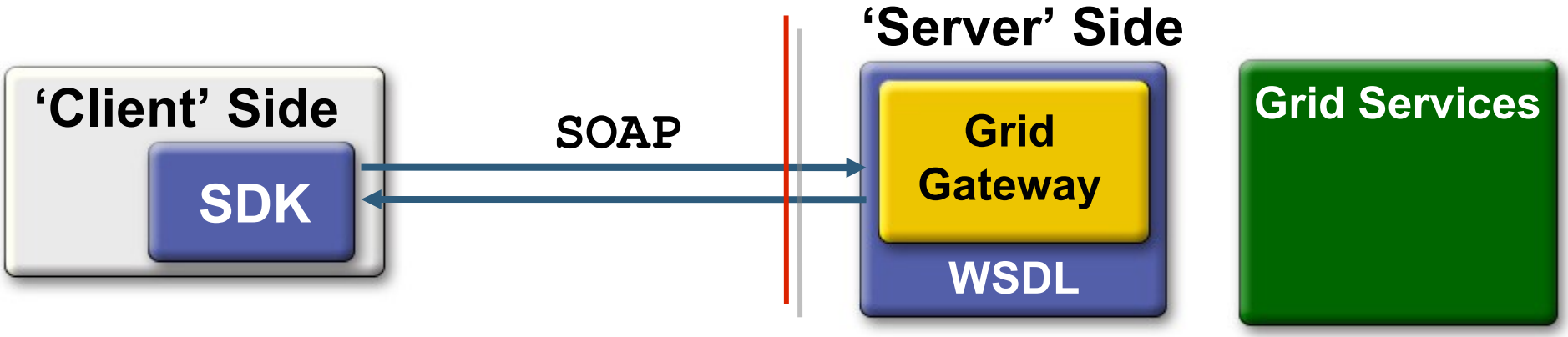
Communication Model

Asynchronous SOAP over Https with Callback

```
//instantiates the callback handler AddCallbackHandler
callbackHandler = new AddCallbackHandler();

//invoke the async callback method
Future<?> resp = port.addAsync(1, 2, callbackHandler);
while(!resp.isDone()){
        //do something
        …
}
System.out.println("The sum is: " + callbackHandler
.getResponse().getReturn());
```

Communication Model

SOAP over Https



- Simple, proven, ubiquitous (all getting better)
- WSDL
  - Java APIs for XML Web Services (JAX-WS): better tooling and interoperability (Project Tango)!
- Client and server share (de)serialization!
  - Verbosity, impedance mismatch
  - Perceived to require higher skill set

Programming and Communication Model

How to rationalize the choices?

Composability Platforms, Languages

Toolable Protocols

Interop.

Quality of Service

SDK

Rules

Fast and Mutually Secure

RMI Over SSL

JMS Over SSL

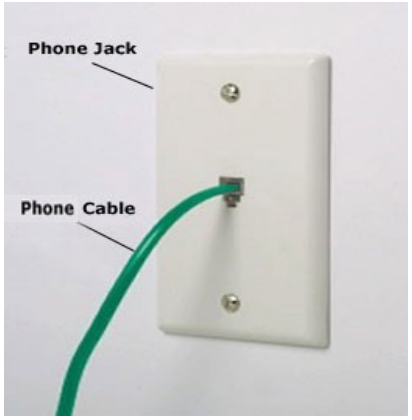
XML Over Http/SSL

SOAP with WS-Security

Designing Middleware for Grid Services!

Develop enterprise applications with Java Technology-based grid services

How much Java technology will we see in Utility Computing and how to “future proof” middleware choices?



Agenda

Motivations for Utility Computing

Compute Utility

Programming Model

Communication Model

Jini ERI

Interesting Technologies

Summary

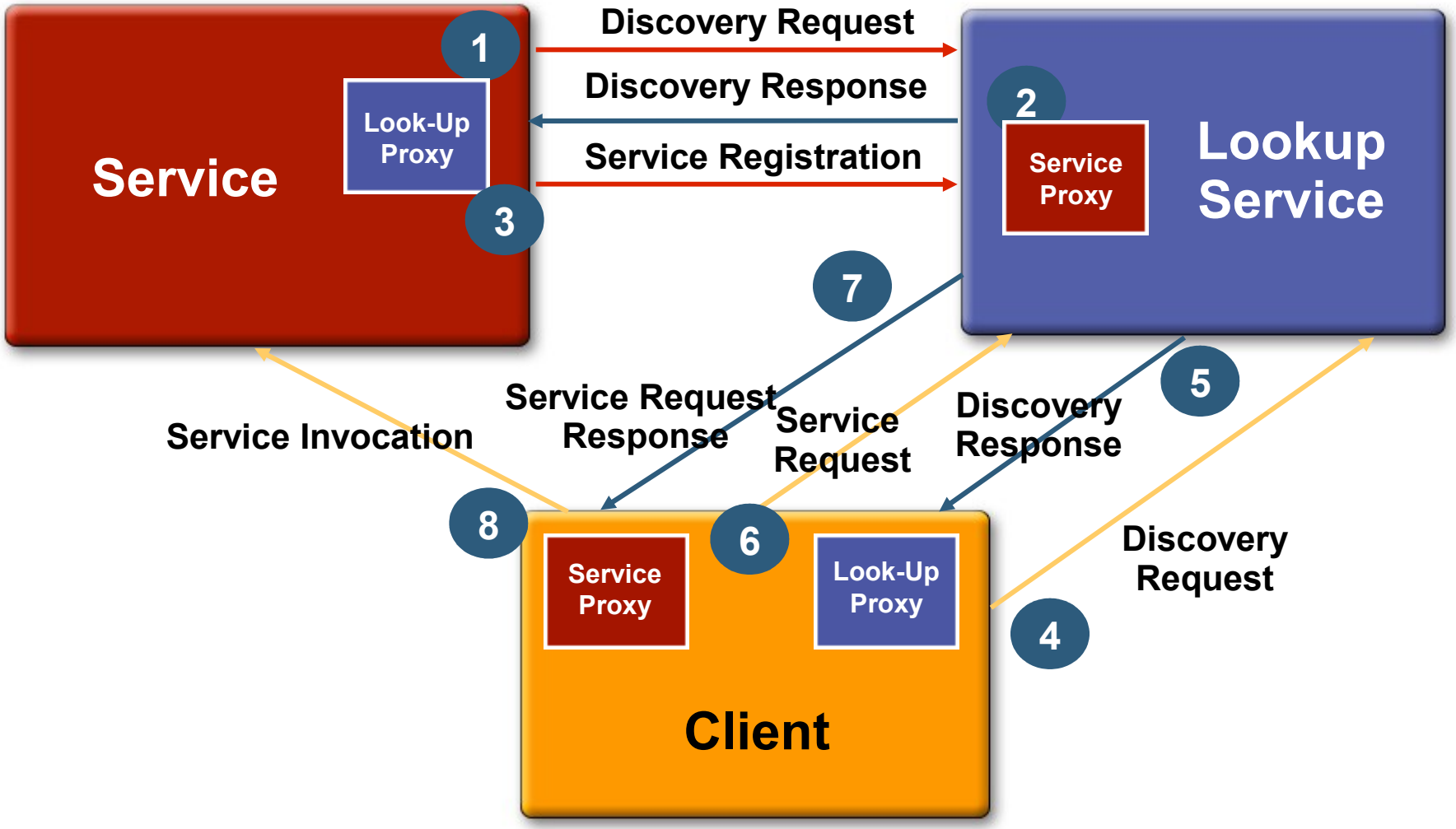
Jini Network Technology: a Crash Course

Architectural characteristics

- Interface centric
- Protocol agnostic
- Failure model: Leasing/RemoteException
- Insulates client from server-impl. changes with mobile code

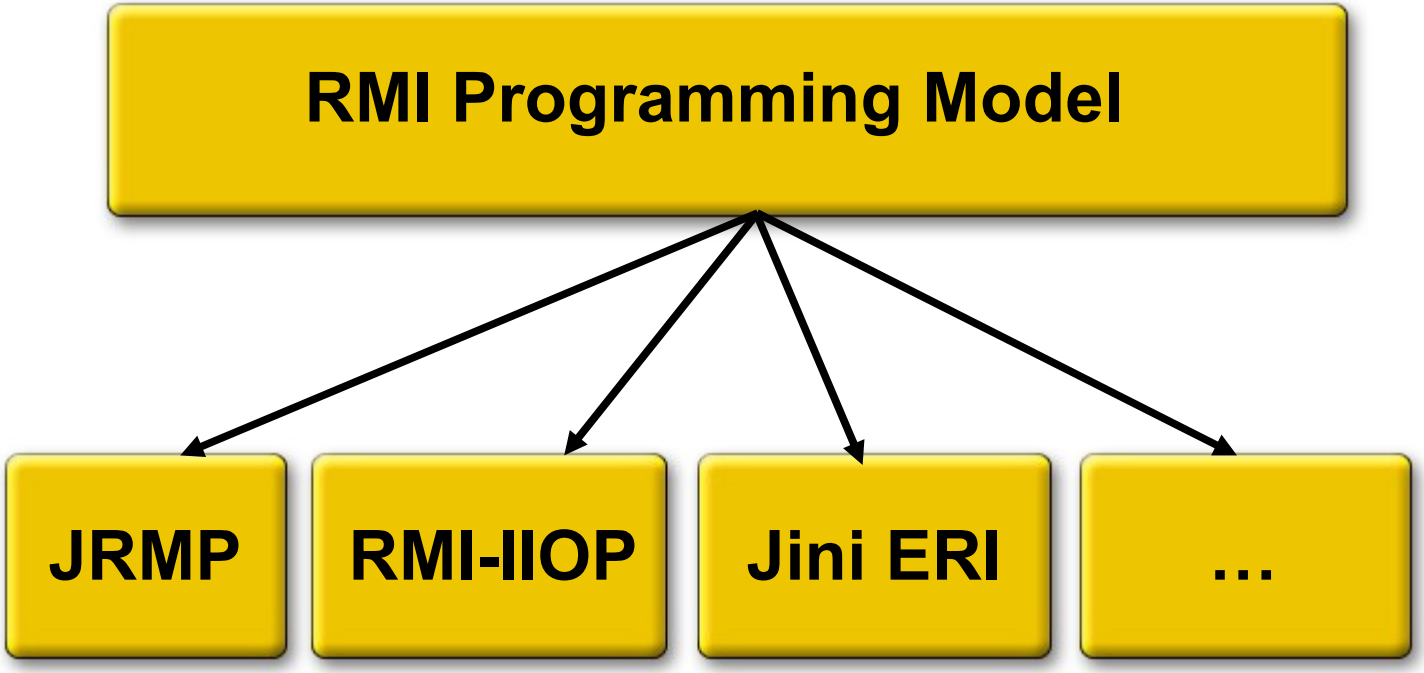
- Open Source Apache License 2.0
- http://www.jini.org
- http://www.jini.org/resources/

Jini Network Technology: a Crash Course



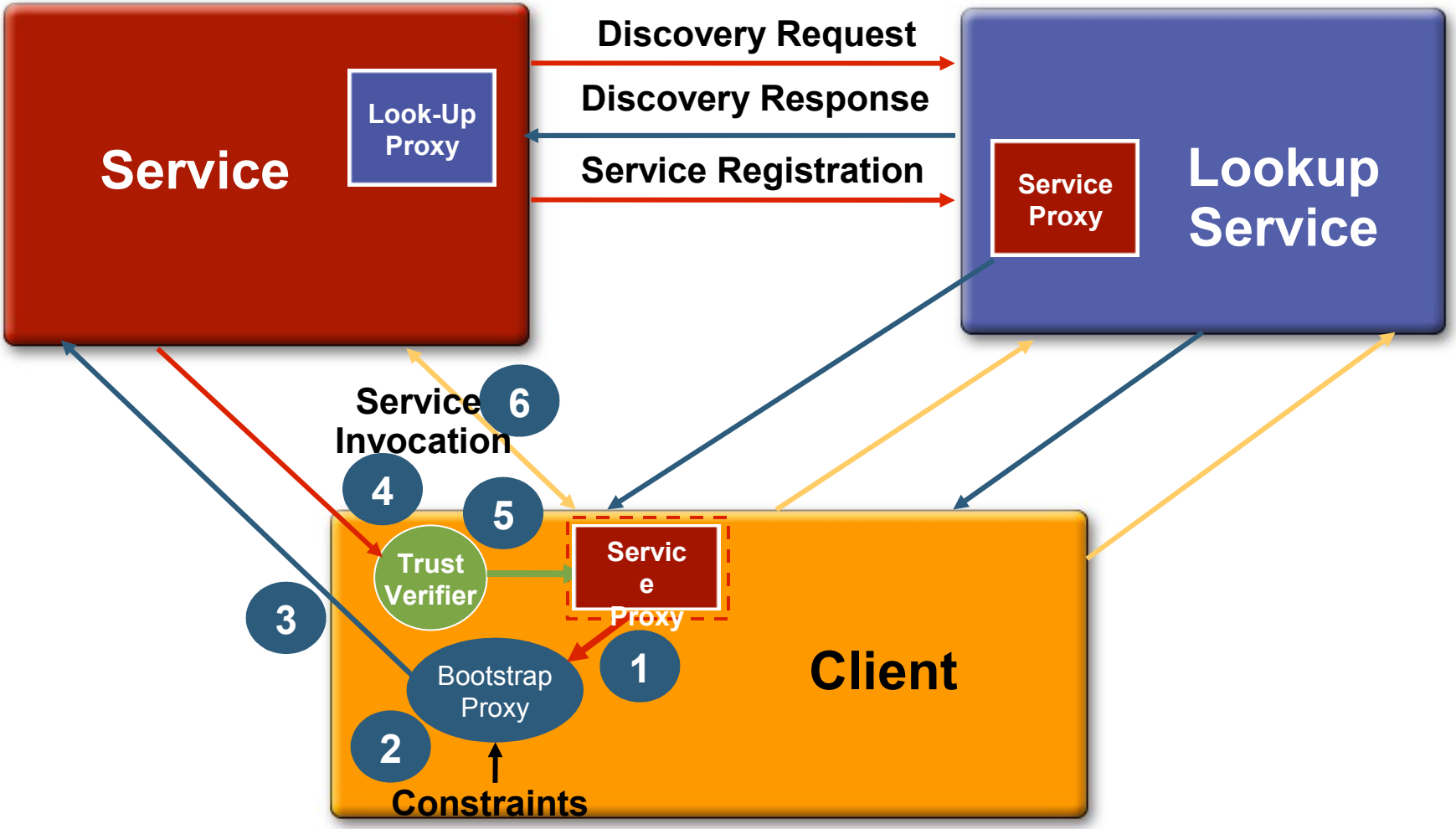
Jini Extensible Remote Invocation (ERI)

What is its legacy?



Jini ERI

Can mobile code be secure?



Jini ERI

Why is it more secure?

- Client receives and executes ‘foreign’ proxy code
  - Client must verify that the proxy code can be trusted
  - Before granting any permissions to the proxy
  - Before making any remote calls through the proxy
- Solution: proxy trust verifiers
  - Client obtains a verifier from the trusted server
  - Through verifier, asks server if proxy can be trusted
- Trust verifiers minimize client’s prior knowledge
  - Client has to know only who the server authenticates as
  - Not codebase or signers or protocols

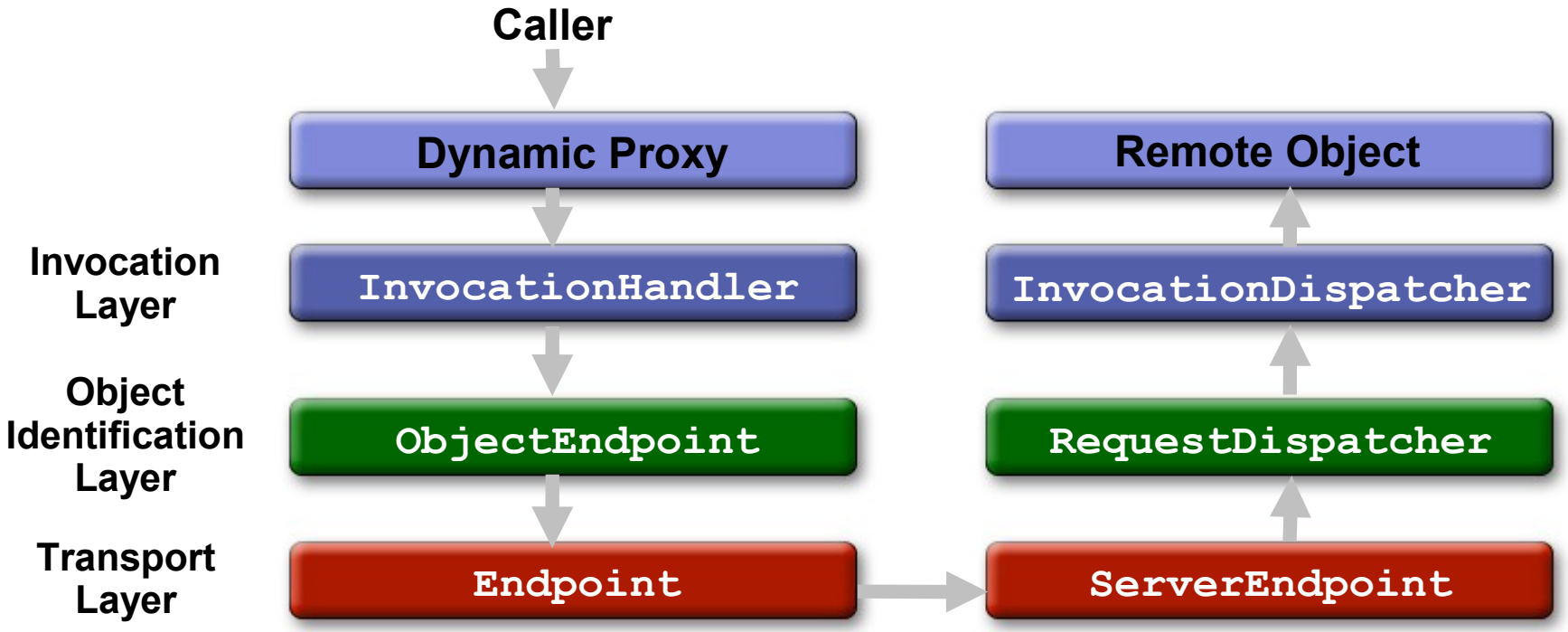
Jini ERI

Another RMI with authentication, integrity, confidentiality

- Specify what a Subject must do, must not do
  - Server/ClientAuthentication; YES/NO (Java Authentication and Authorization Service [JAAS] Subjects)
  - Integrity/Confidentiality; YES/NO
  - QoS (max. threads, connection timeout, etc.)
- Enforced on a per-method basis
  - E.g., authenticate on write, but anonymous for read
- Proxy implements RemoteMethodControl interface
  - Indicates proxy supports network security
  - Allows client to attach constraints to proxy

Jini Extensible Remote Invocation

Architectural Protocol Stack



- TCP, HTTP, SSL, HTTPS, Kerberos, UDP, …
  - Some transports may be connectionless

Jini ERI Transport Layer

```
public interface OutboundRequest {
 OutputStream getRequestOutputStream();
 InputStream getResponseInputStream(); ..
}
public interface InboundRequest {
 InputStream getRequestInputStream();
 OutputStream getResponseOutputStream(); ..
}
public interface Endpoint {
 OutboundRequestIterator newRequest(InvocationConstraints
i);
}
public interface ServerEndpoint {
 Endpoint enumerateListenEndpoints(
             ListenContext lc) throws IOException;
interface ListenEndpoint {
 ListenHandle listen(RequestDispatcher rd) throws
IOException;
}
```

Jini ERI Object Identification Layer

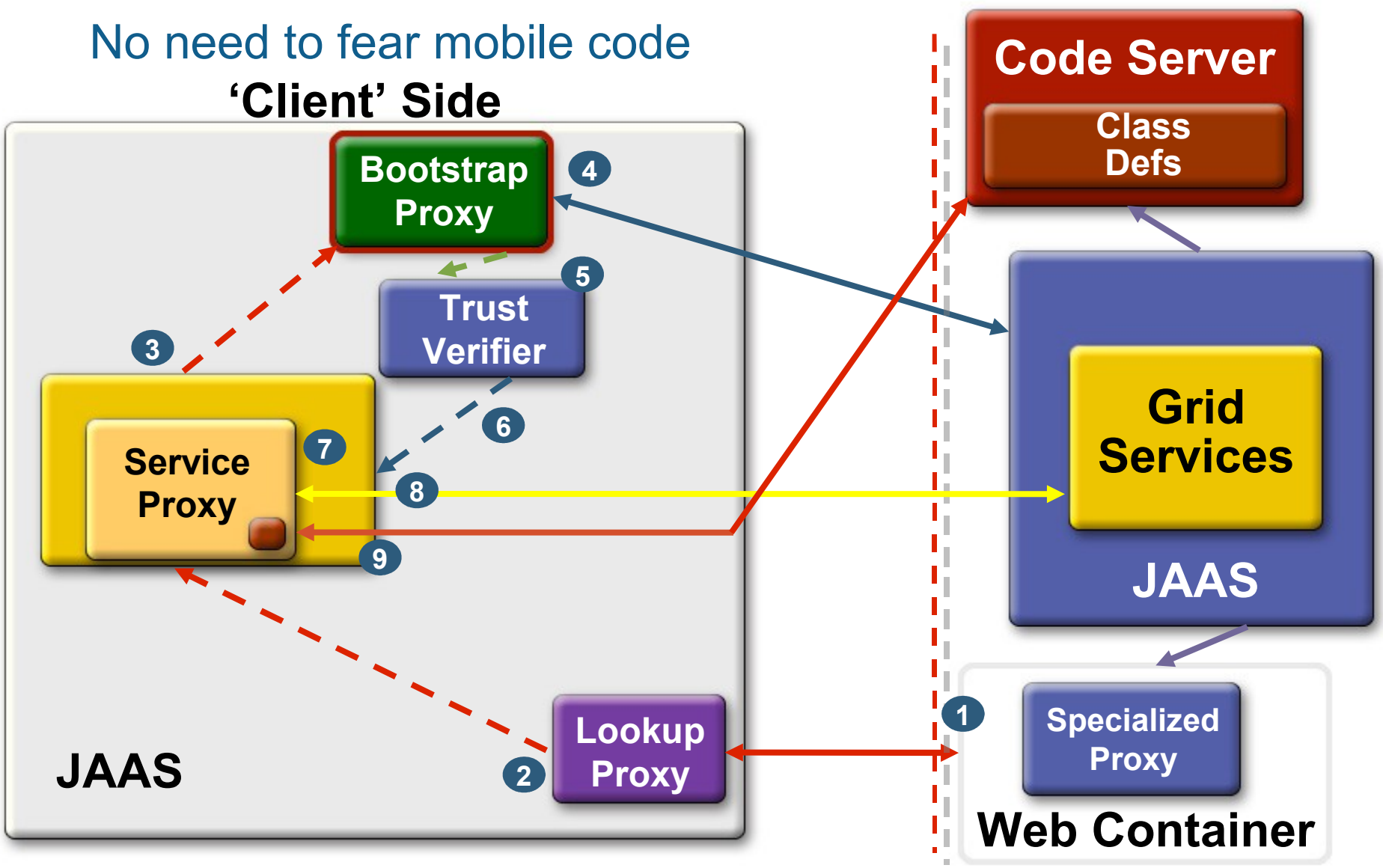
```
public interface ObjectEndpoint {
      OutboundRequestIterator newCall(
            InvocationConstraints ic);
      RemoteException executeCall(OutboundRequest
or)
            throws IOException;
}

public interface RequestDispatcher {
      void dispatch(InboundRequest ir);
}
```

Jini ERI Invocation Layer


```
package java.lang.reflect;
public interface InvocationHandler {
       Object invoke(Object proxy,
                     Method method,
                     Object[] args)
              throws Throwable;
}
public interface InvocationDispatcher {
       void dispatch(Remote impl,
                     InboundRequest ir,
                     Collection ctx);
}
```

Jini ERI across Firewalls

No need to fear mobile code



Jini ERI

In a nutshell

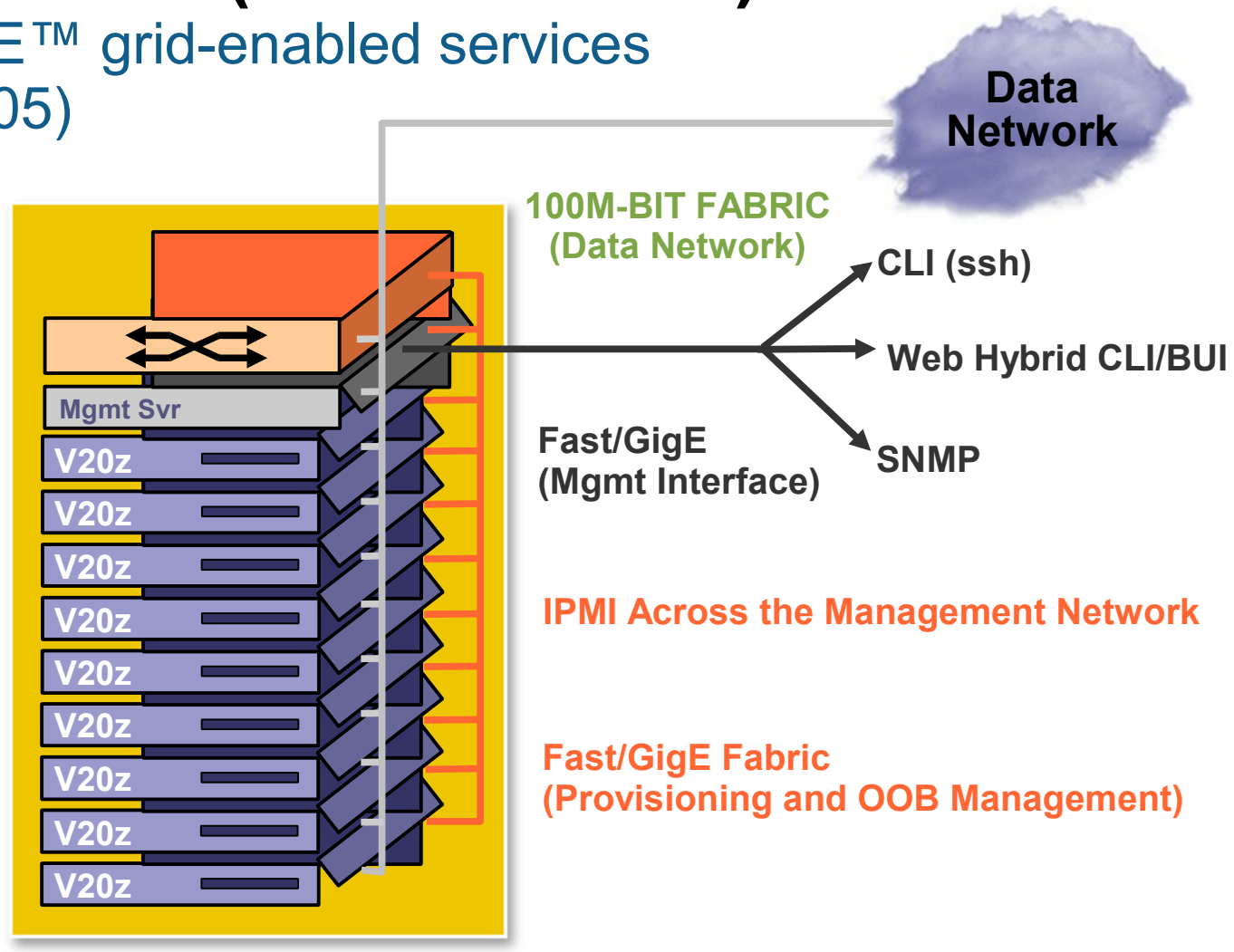
- Pluggable transport
  - TCP, SSL, HTTP, JRMP, IIOP etc
- Pluggable security
  - JAAS: with PKI or Kerberos
- Bi-directional security
  - Object integrity: data and downloaded code
  - Proxy trust
  - Mutual authorization
    - Server authorizes clients
    - Client authorizes the downloaded proxy code
- Can easily work across firewalls

Agenda

Motivations for Utility Computing
Compute Utility
Programming Model
Communication Model
Jini ERI
Interesting Technologies
Summary

Adaptive Grid (Skunkworks)

Building J2EE™ grid-enabled services
TS-3230 (2005)



Cluster-MVM

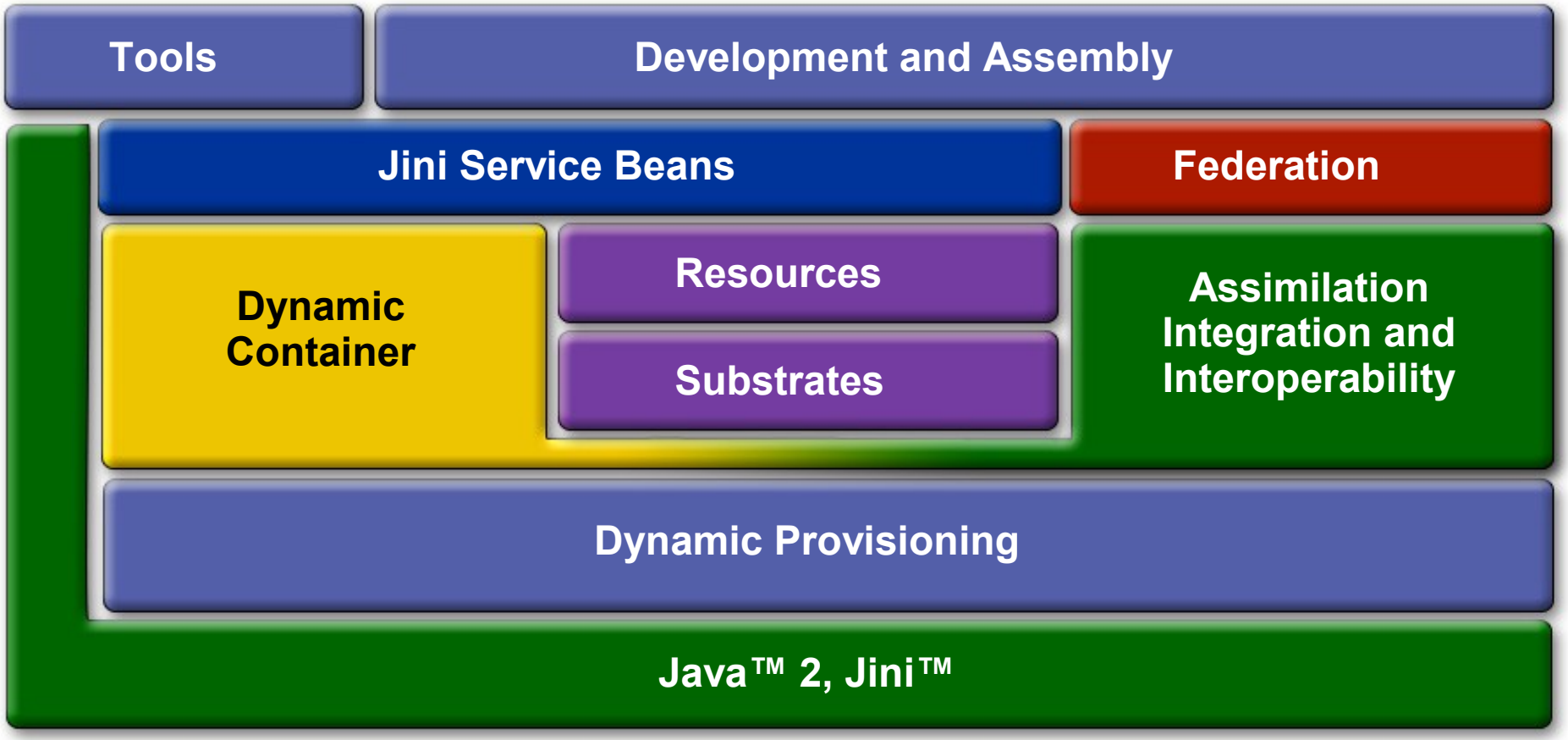
Java platform clustering: present and future: TS-7159 (2005)

- An architecture of federated Java virtual machines
- Programming: extended “Isolate API” (JSR 121)
  - Isolate == an isolated Java technology-based application
  - No sharing among isolates
  - Copy-only communication
  - Clean termination
  - Unambiguous resource management
- Implementation based on the Multitasking Virtual Machine (MVM)
- Integrated, comprehensive resource management

Project RIO

http://rio.jini.org

- Dynamically instantiate, monitor and manage services based on meta-data and policies



Agenda

Motivation for Utility Computing

Compute Utility

Programming Model

Communication Model

Jini ERI

Summary

Summary

- Increasing convergence
  - SOA and grid
- You too can build a compute utility
  - Technologies like Jini have solved most of the problems
- Composability and Interoperability
  - Foremost when it comes to middleware design
- Jini™ Extensible Remote Invocation
  - Flexible, pluggable-transport/security
  - Can easily work with firewalls and more so without
  - Can be useful with future Java™ grid technologies

For More Information

- http://www.network.com
- http://blogs.sun.com/murali
- BOF-0668 : The Grid Appliance
  - Thursday, 05/18/2006, 08:30 PM–09:20 PM

Q&A

Murali Kaundinya

Programmatic Access to a Compute Utility

Murali Kaundinya

Senior Staff Engineer
Sun Microsystems, Inc.
http://www.network.com

TS-5622