Advanced Sun™ Grid Creating Applications for Horizontal Scale

Amir Halfon
Keith Thompson

Sun Grid Engineering
Sun Microsystems, Inc.

http://developer.network.com

TS-3117

Goals

Develop a clearer understanding of the design considerations associated with developing massively scalable applications for Sun™ Grid

Agenda

- Background
  - Grid and Utility Computing
  - The Sun Grid Compute Utility
- Distributed Parallel Computing Paradigm
  - Problem Space
  - Solution Space
  - Implementation Mechanisms
- Compute Server
  - Overview
  - Demo

Problem Statement

"Only 5% of today's applications are suitable for grid computing..."

How do we accelerate the development of applications that can benefit from the grid?

First, some background...

Grid Computing

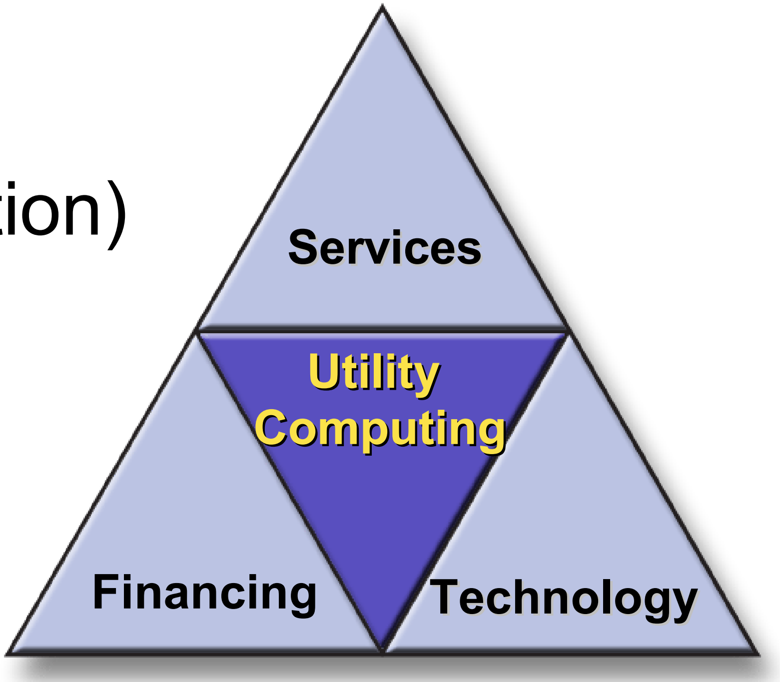
- A node-centric, rather than a CPU-centric paradigm—
the commodity is the computer...
  - Enterprises are following HPTC
- Economies of scale: “a cluster of cheap boxes” instead of a large, expensive SMP box
  - Thousands of CPUs instead of dozens
  - Distributed instead of shared memory
  - Distributed parallelism—has to be designed

Utility Computing (UC)

The Ability to Intelligently Match IT Resources to Business Demand on a Pay-for-Use Basis

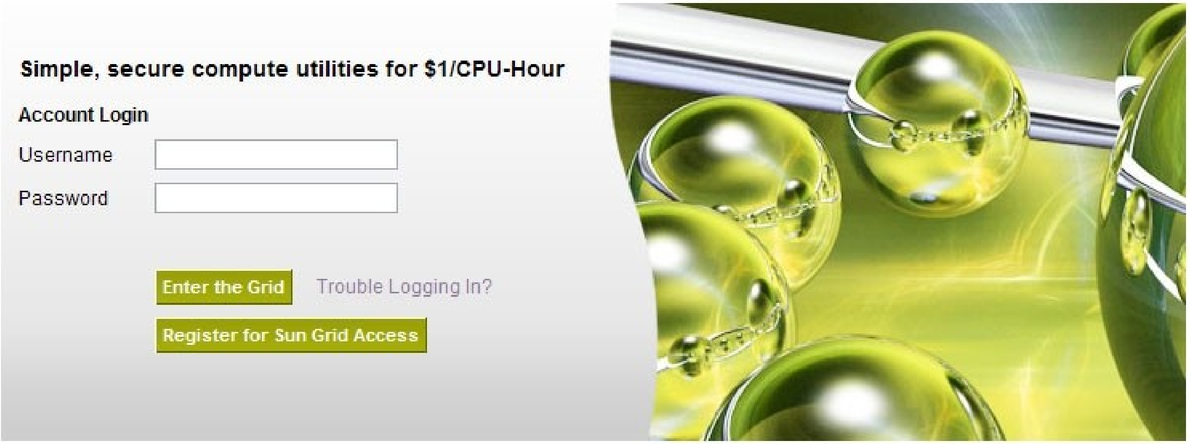
Attributes

- Standardization (aggregation)
- High utilization
- Multi-tenancy
- Immediate provisioning
- Granular costing



Sun Grid Compute Utility

Available Today! (www.network.com)



- Intersection of grid computing and the utility computing model
- Pay-per-use compute power at a standard pricing —$1/CPU-hour
- No contract or minimum commitment

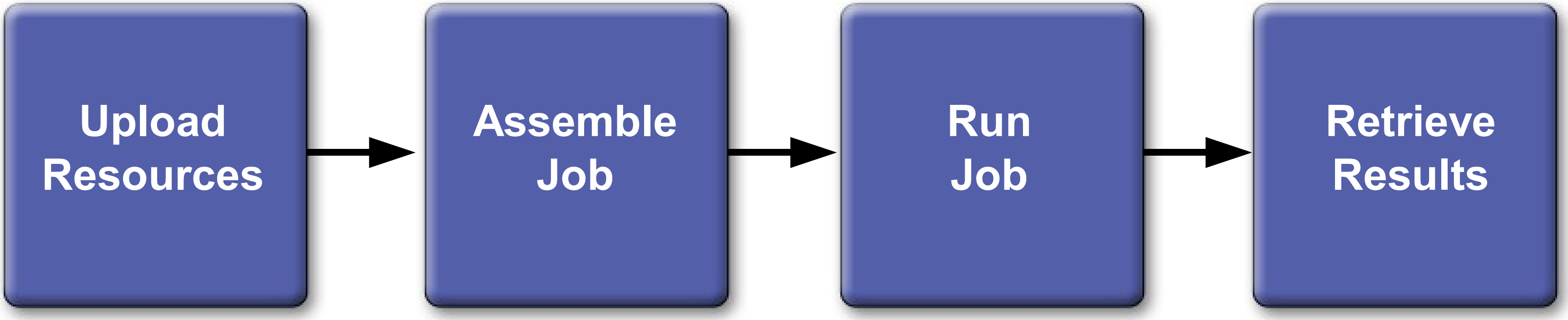
The Sun Grid Commodity Unit

- Sun Fire™ V20z servers, each containing:
  - Dual 2.4 GHz AMD Opteron™ processors with HyperTransport technology for memory and I/O interface
  - 8 GByte RAM
  - Solaris™ 10 Operating System
- Grid network infrastructure built on a Gbit/s switched-Ethernet data network

The Distributed Resource Manager

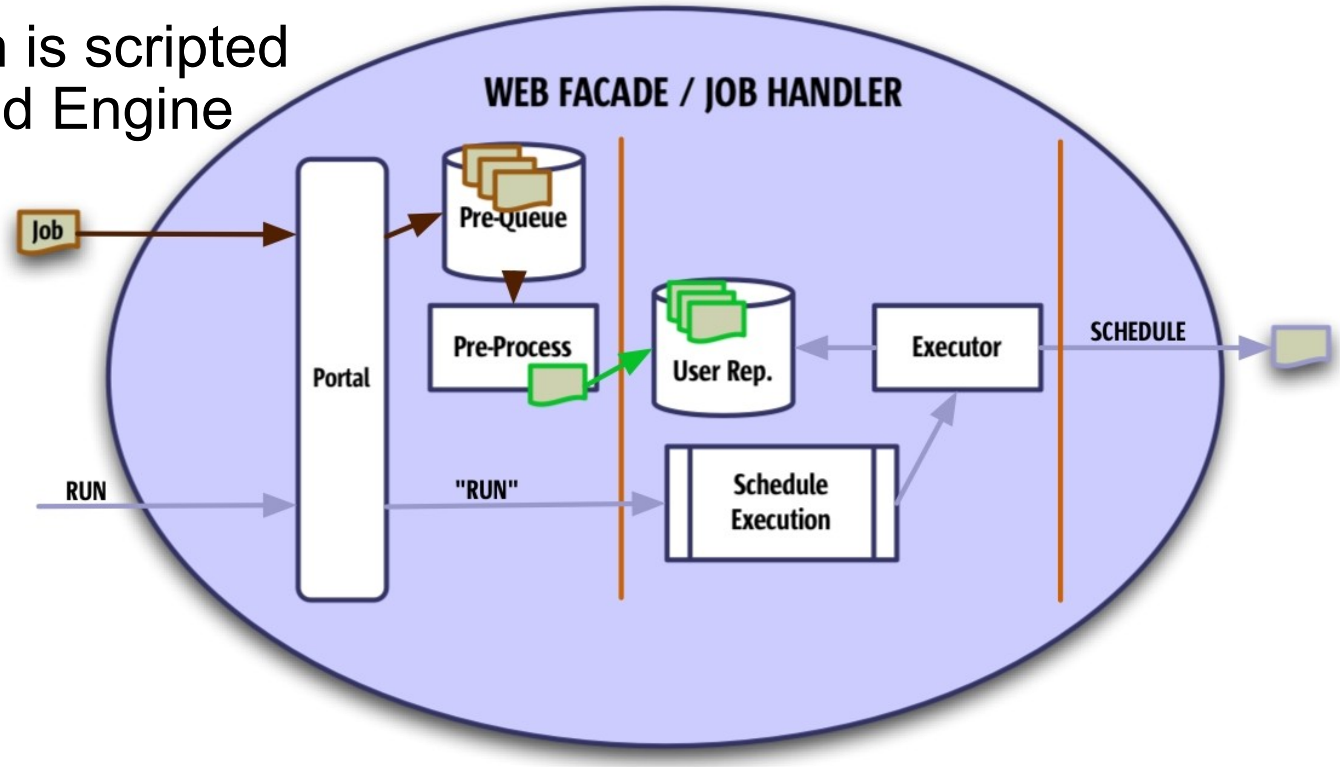
- A utility model requires sharing a highly utilized pool of systems
- Demand is managed by a scheduler
- Sun Grid uses Sun N1™ Grid Engine 6 software
- As a developer, this means working in a constrained environment...

Typical Sun Grid Workflow



Behind the Scenes…

- Resources are uploaded through the portal
- Jobs are assembled from resources
- Job execution is scripted using Sun Grid Engine commands
- Users can schedule jobs to run “at will”
- Results are picked up through the portal



Agenda

- Background
  - Grid and Utility Computing
  - The Sun Grid Compute Utility
- Distributed Parallel Computing Paradigm
  - Problem Space
  - Solution Space
  - Implementation Mechanisms
- Compute Server
  - Overview
  - Demo

Back to Our Problem Statement…

- Clusters of cheap boxes abound,utility computing models beginning to appear…
- **Yet**, Applications that fully exploit this infrastructure are still relatively scarce (except in HPC)
- The challenge: Can we make parallel programming easier?

Designing for Concurrency*

- Decomposition
  - Tasks
  - Data
- Task dependency analysis
  - Temporal constraints
  - Grouping
  - Data sharing

* Based on Patterns for Parallel Programming by Timothy G. Mattson, et al

Mapping to a Solution

- Determine major organizing principle to choose the right algorithm
- Map tasks to Units of Execution (UE)
- Target platform considerations
  - Number of UEs: efficiency vs. overhead
  - Communications between UEs
  - Synchronization needs
- Deutsch’s fallacies still apply…

Organization Principles

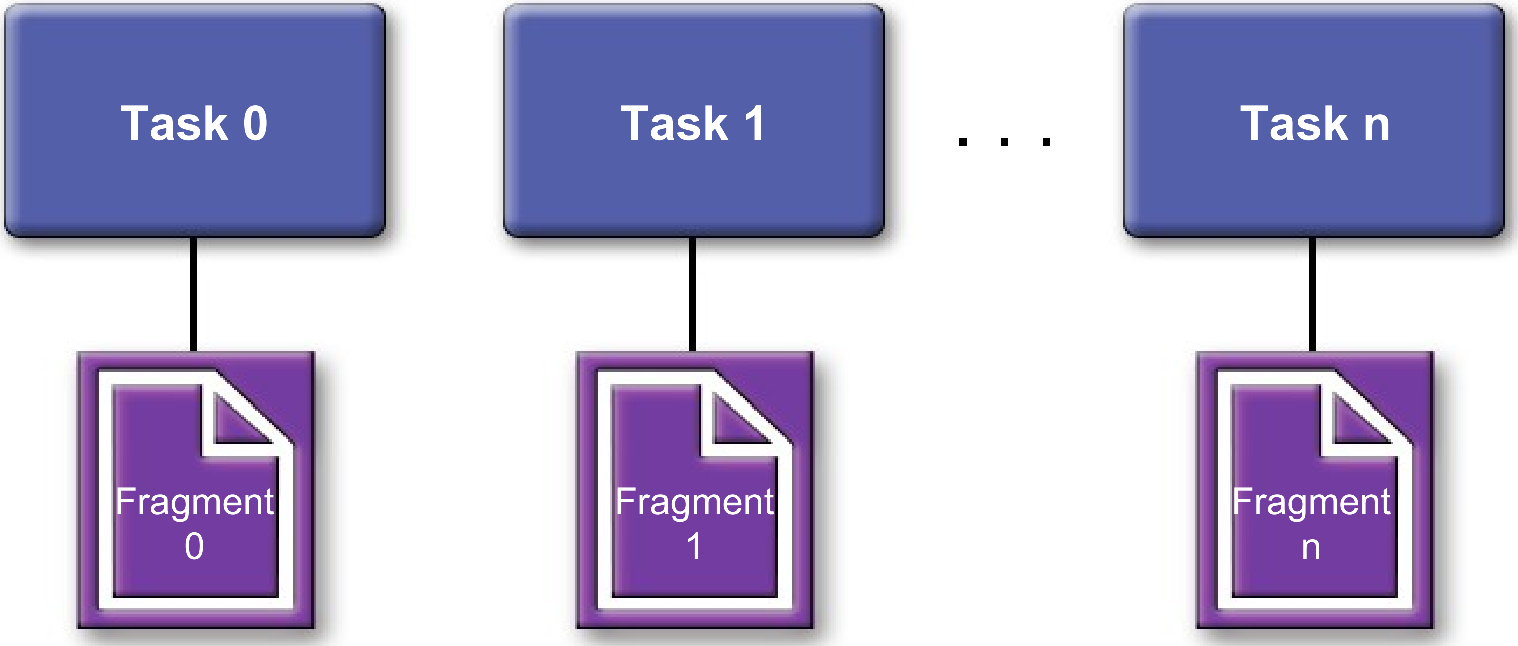
- Tasks
  - Problem can readily be described as a group of relatively independent tasks
- Data decomposition
  - Focus on data decomposition and data sharing between tasks
- Data flow
  - Focus on the interaction between the tasks

Task Parallelism

- Based directly on the tasks
- Design involves task definition, dependencies (e.g. data sharing), and scheduling
- In most cases, tasks are associated with the iterations of a loop
- Example: imaging (ray tracing), frame rendering, financial risk management/interest calculation/ index calculation

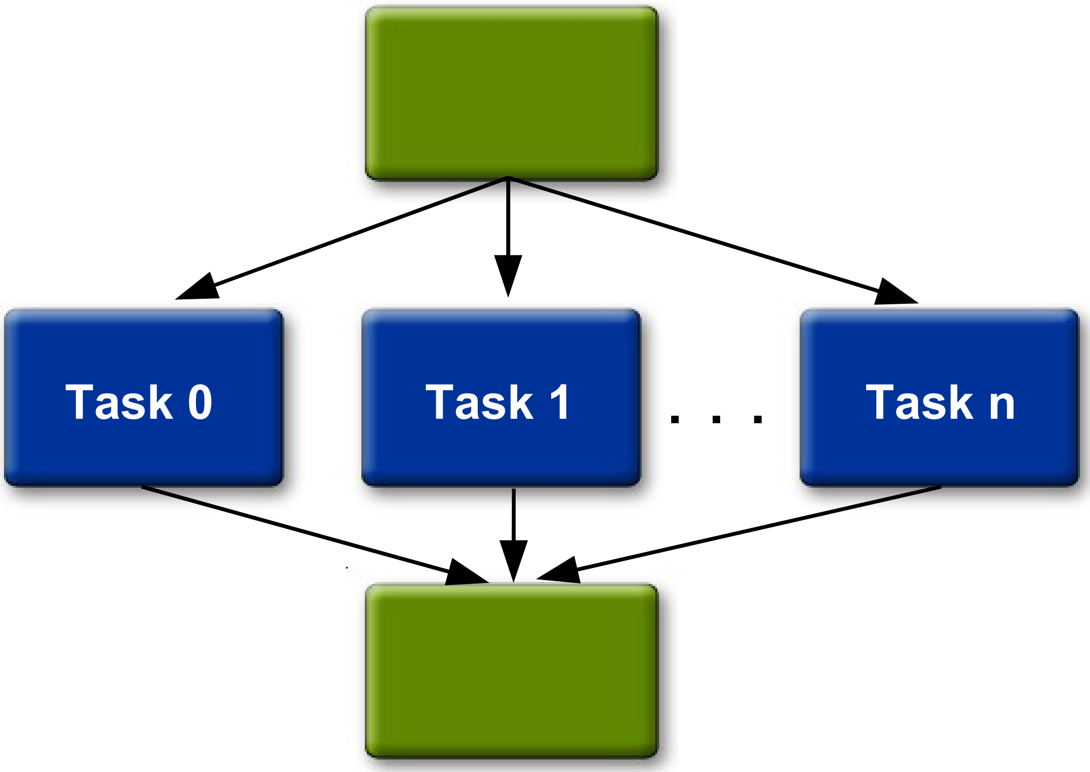
Program Structuring Patterns: SPMD

- Single program, multiple data
- Units execute the same program, each on a different set of data



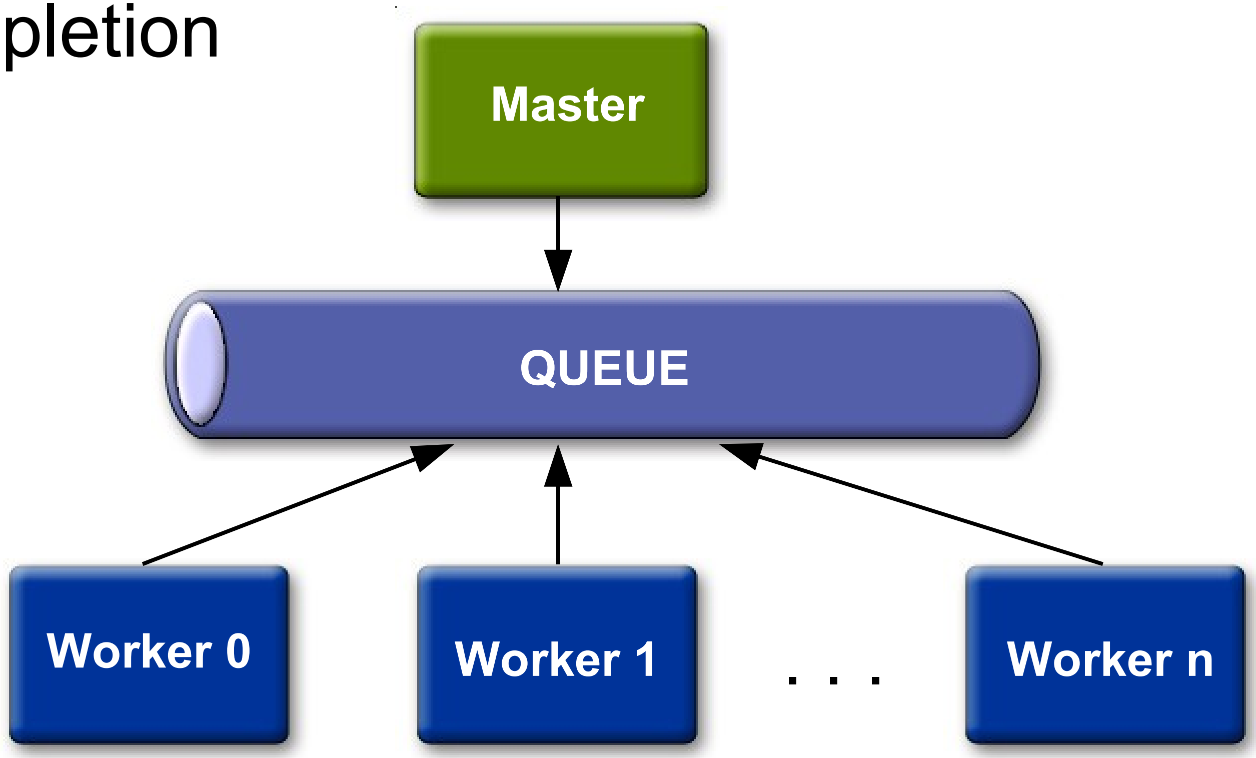
Program Structuring Patterns: ForkJoin

- Main Process forks units, which continue in parallel before re-joining



Program Structuring Patterns: MasterWorker

- Master sets up a pool of workers and a task queue; workers pull from the queue until completion



Implementation Mechanisms

- Process management
  - UE creation and destruction
- Communication
  - Inter-task
  - Collective
- Synchronization
  - Temporal constraints
  - Serial constraints

UE Management

- Java technology typically uses threads
  - Executors in Java SE 5 platform
  - Higher level Containers (app servers, etc.)
- On Sun Grid: Using N1 Grid Engine...
  - Grid Engine directives (qsub commands)
  - DRMAA (Distributed Resource Management Application API) instructions to the GE
- …Or programmatically
  - MPI (Java based encapsulation available at hpjava.org)
  - Custom Mechanisms

Grid Engine Commands

`qsub` submit "task" to queue

`qstat` get snapshot of queue status to determine task status

`qdel` hard stop of a queued task by name or identity

```
#submit a task to up to 4 nodes on the grid, with a
   minimum of 1 node
qsub -t 1-4:1 foo
```

- Remember: scheduler is non-deterministic
  - Request what you think that you'll need, realize the potential wait time for requesting more

Continuum of Job Control

- "Command line" scripting
  - (+) Fine grained control of Grid Engine Tasks = minimize spend
  - (–) May "spin" while acquiring "incremental" resources and no guarantee
  - (–) Complex "scripting"
- Programmatic control
  - (+) Application level language flexibility
  - (–) Growing resource pool requires calls to "native" queue management

Inter-Task Communications

- Do you need it?
  - Some problems are embarrassingly parallel
- Cost
  - Overhead
  - Communication instead of computation
  - Network saturation
  - Usually implies synchronization
- Challenge
  - Lookup and discovery

Collective Communications

- Broadcast
  - Single message to all UEs
- Reduction
  - Reducing a collection to a single item (sum, max, etc.)
- Barrier
  - Synchronization—could be implemented as a collective communication

Communications (Cont.)

- In Java technology: Sockets, RMI, JavaSpaces™ technology, Java Message Service (JMS)
- On Sun Grid:
  - RMI is an option, more on JavaSpaces technology later…
  - MPI libraries are available (tightly coupled with Grid Engine)
  - File system is easiest to use
    - Applications read/write files
    - NFS for sharing data (home directory as a shared file system)
    - Don't forget to clean up

Establishing Lookup and Discovery Context

1) Write hostname to a file and have other tasks use this file to locate each other
2) Start a lookup service first, then pass the location of the lookup (from qstat) as a parameter to all other tasks

Caveat: Multicast discovery is not currently allowed on Sun Grid

Synchronization

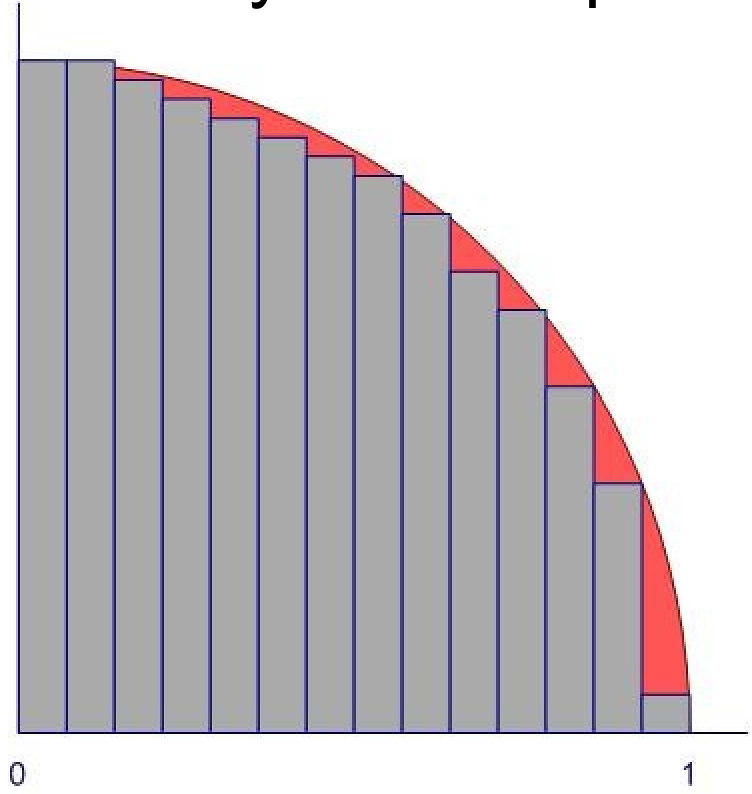
- Barriers
  - All UEs must arrive at a certain point before proceeding
- Mutual exclusion
  - Modifying a shared resource: data, file, etc.
- Serialization
  - Certain sections cannot proceed in parallel (dependencies)

Synchronization (Cont.)

- In Java technology: synchronized blocks and methods
  - Locks in Java 5 (blocking and non blocking)
- On Sun Grid: Coordination using N1 Grid Engine
  - Jobs may have dependencies
    - One job can wait for another to complete
    - Use ***qsub*** command with various options
    - DRMAA is also available
  - MPI offers a set of synchronization constructs

Example: Integral Pi Computation

- Computing time increases by an order of magnitude for every decimal place of precision



Code Sample: Integral Pi Computation

```
BigDecimal x0 = BigDecimal.ZERO;

while (x0 < BigDecimal.ONE) {
    BigDecimal x1 = x0.add(sliceSize);

    //calculate the rectangle's height
    height = sqrt(BigDecimal.ONE.subtract(x1.multiply(x1));

    //add the rectangle's area to the sum
    sum = sum.add(sliceSize.multiply(height));

    x0 = x1;
}
BigDecimal pi = sum.multiply(new BigDecimal(4));
```

Parallelizing the Computation

- Slice the problem… (parallelize the loop)
- Organizing principle: Task Parallelization
  - No dependency between tasks
  - No data sharing, except for reducing the result
  - Embarrassingly parallel
- Structuring patterns: SPMD, Master Worker
- Communication mechanism: Reduction

Code Sample: Computing Pi in Parallel

```
BigDecimal x0 = BigDecimal.ZERO;

while (x0 < BigDecimal.ONE) {
    BigDecimal x1 = x0.add(sliceSize);

    // create the task for this slice
    Task task = new Task(x0, x1);

    // send the task to be executed
    // and add the results to the sum when its done
    theHardPart(task);

    x0 = x1;
}
BigDecimal pi = sum.multiply(new BigDecimal(4));
```

The Hard Part...

```
#fire off server
GSC=`qsub -sync n -N gsee-gsc -v GSEE_HOME=$GSEE_HOME -v \
GRID_HOME=$GRID_HOME -t 1-100:1
#get id from return
MATCH=″\(.*\) \(.*\) \([0-9]*\)\.\([0-9]*\)-\([0-
9]*\):\([0-9]*\)‶  GSCparsed=( `echo $GSC | sed -n -e
″s/${MATCH}/\3/p‶` )
#wait for service to start before proceeding
GSCstatus=0
until [[(″$GSCstatus‶ > 0)]] do
GSCstatus=$(qstat -s r | nawk '/'${GSCparsed}'/{var1+=1}
END {print var1}')
sleep 10
done
#now submit task(s) using service
~/integral-pi.sh $1
#clean up
$(qdel $GSCparsed)
```

Agenda

- Background
  - Grid and Utility Computing
  - The Sun Grid Compute Utility
- Distributed Parallel Computing Paradigm
  - Problem Space
  - Solution Space
  - Implementation Mechanisms
- Compute Server
  - Overview
  - Demo

Compute Server Project Overview

- Sun Grid Developers Network project that eases use of Sun Grid
- Supports Master/Worker pattern
  - Sub-dividable into independent pieces of work—tasks
  - Single master generates tasks
  - Multiple workers process the tasks
- IDE integration to support development
  - NetBeans™ software plugin provides templates and tools
  - Local debug environment
  - Packing/unpacking grid resources

Compute Server Project Overview (Cont.)

- Content experts not Distributed Computing Experts
  - Compute Server takes care of the details
    - Provisioning of master and workers
    - Distribution of tasks to workers
    - Facilitates feedback and output
- Simple Java programming model
  - Single-threaded POJOs
    - Single-threaded tasks executed by workers
    - Single-threaded task generator executed by master
    - Output processed off-grid

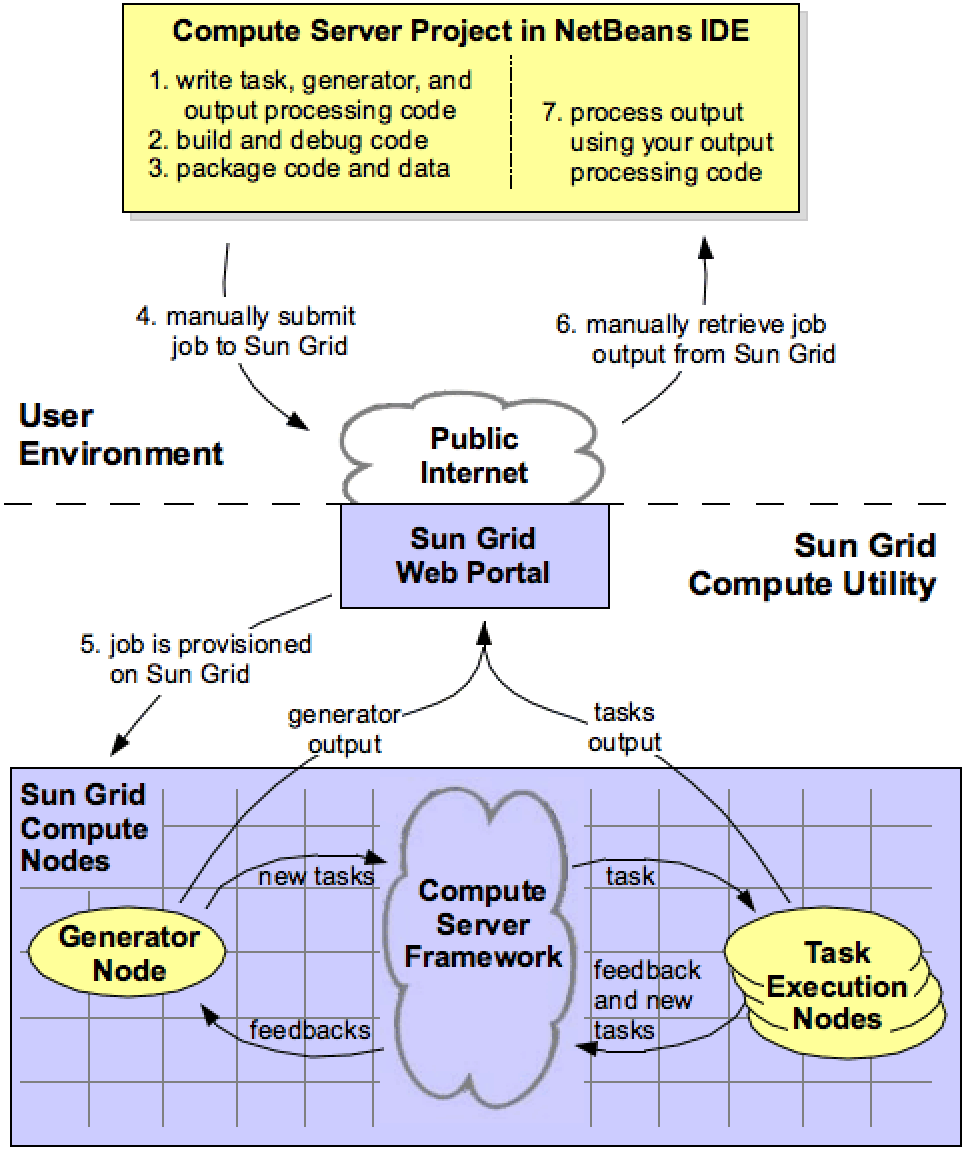
What We Will See

- Using the NetBeans IDE
  - Create a Compute Server project
  - “Write” the application-specific code—Pi in parallel
  - Test locally to ensure correctness
  - Specify key execution parameters
  - Generate package for submission to Sun Grid
- Submit to Sun Grid
- Use IDE to process output and displays results

DEMO

Sun Compute Server

Compute Server



Summary

- The Sun Grid Compute Utility is a unique offering that brings the benefits of Grid and Utility Computing to the masses
- The real challenge in realizing these benefits is designing massively scaling applications
- Patterns have been established to help solve this problem
- Frameworks such as Compute Server make things easier

For More Information

Sessions

- 1109: The Sun Grid Compute Utility

BOFs

- 7995: What's Next for Sun Grid

Labs

- 7135: Building Grid-Enabled Applications

URLs

- http://developer.network.com
- http://www.llnl.gov/computing/tutorials/parallel_comp

Related books

- *Patterns for Parallel Programming* by Timothy G. Mattson, et al

Q&A

amir.halfon@sun.com
keith.thompson@sun.com

http://developer.network.com

Advanced Sun™ Grid Creating Applications for Horizontal Scale

Amir Halfon
Keith Thompson

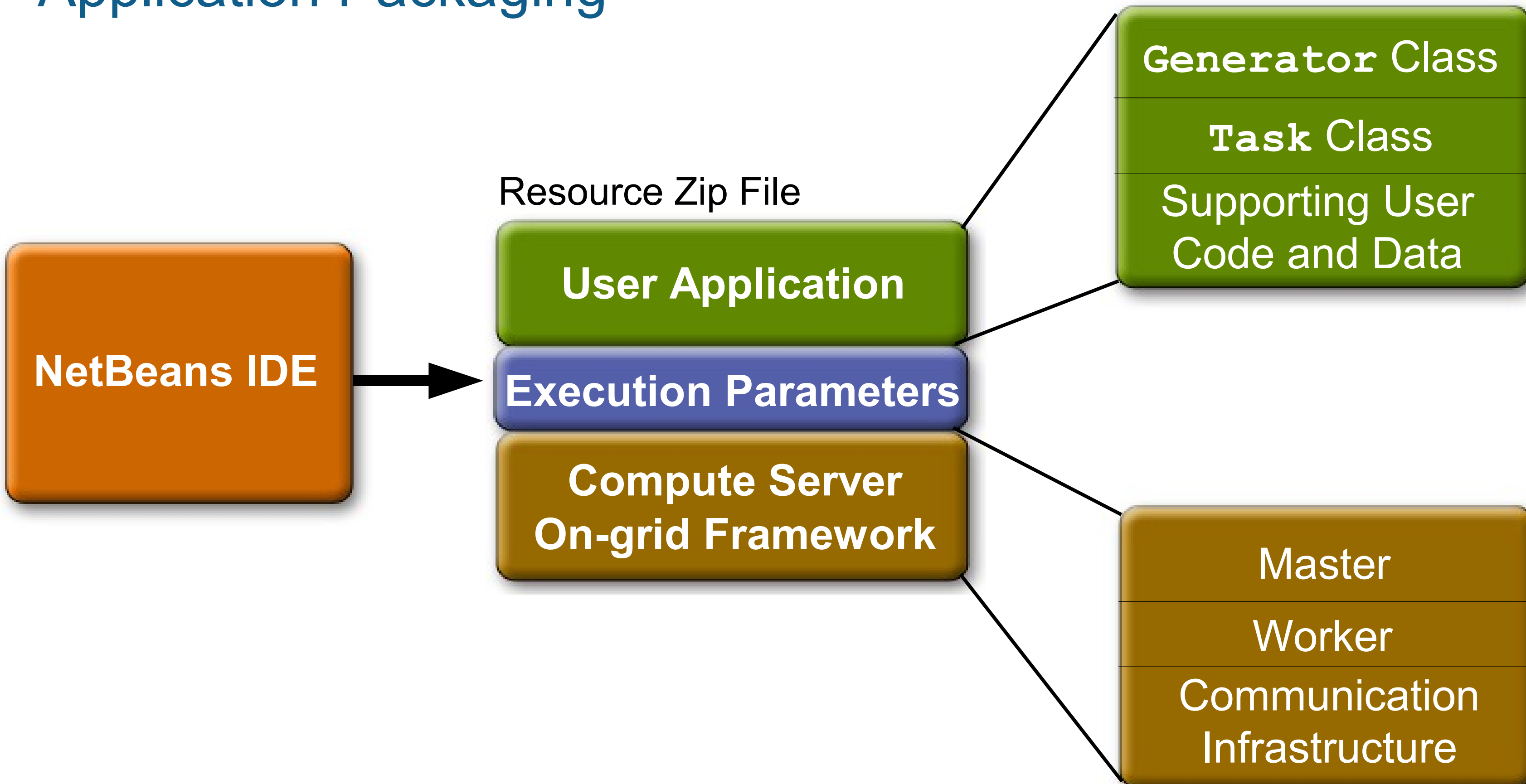
Sun Grid Engineering
Sun Microsystems, Inc.

http://developer.network.com

TS-3117

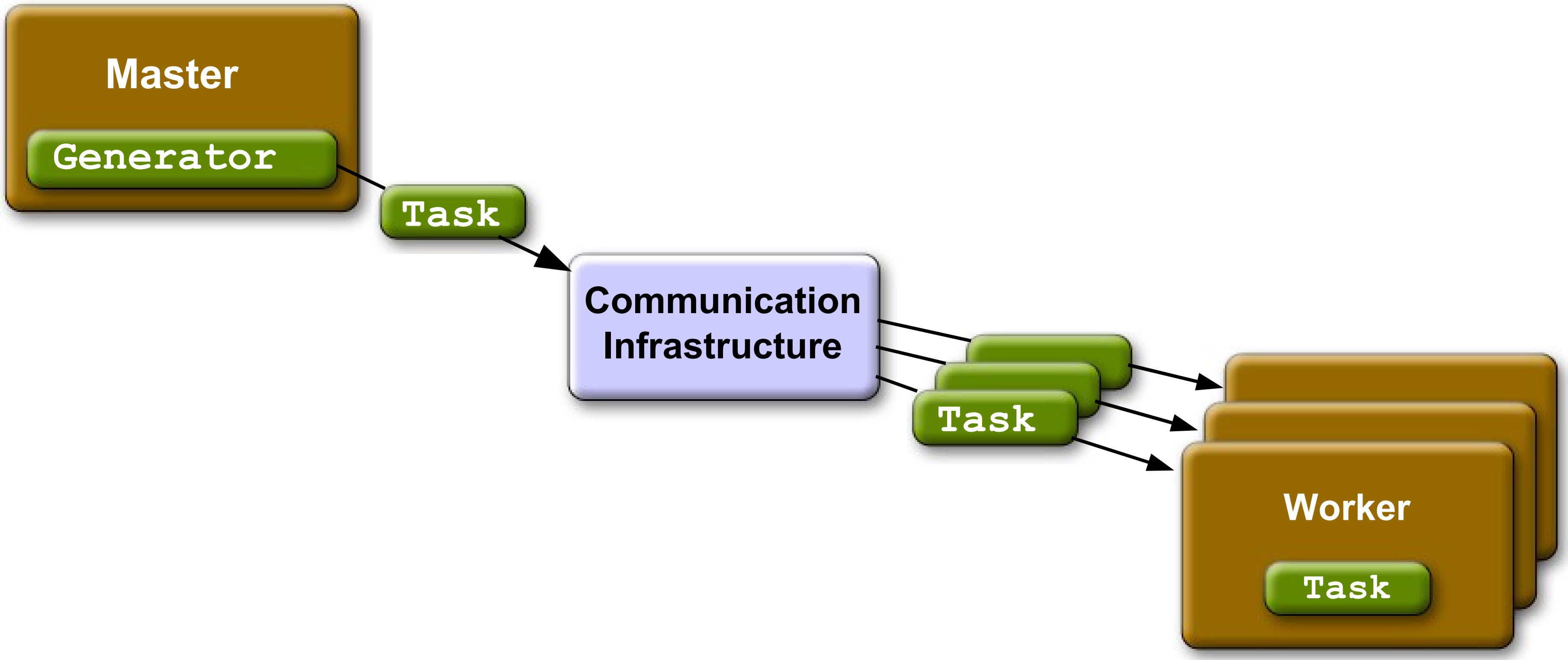
Supplemental Material

Compute Server
Application Packaging



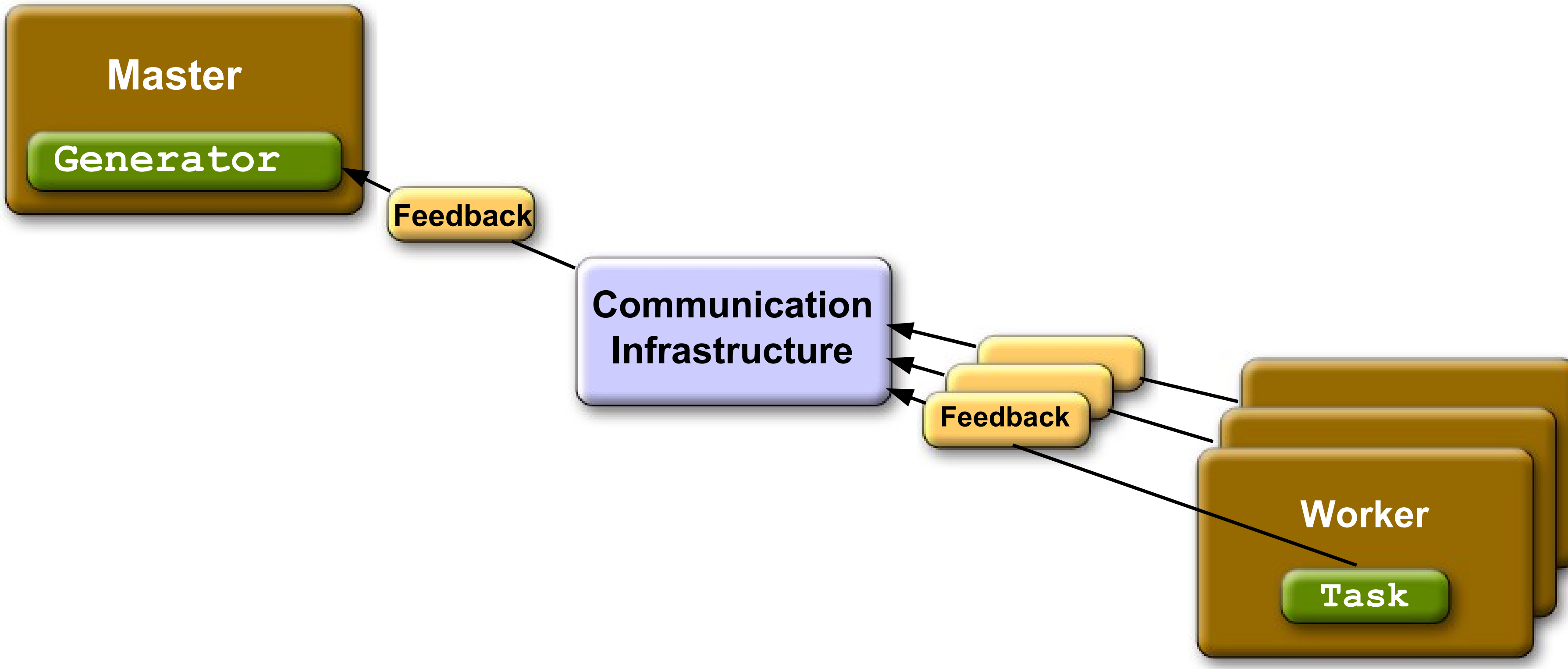
Compute Server

Task Generation and Distribution



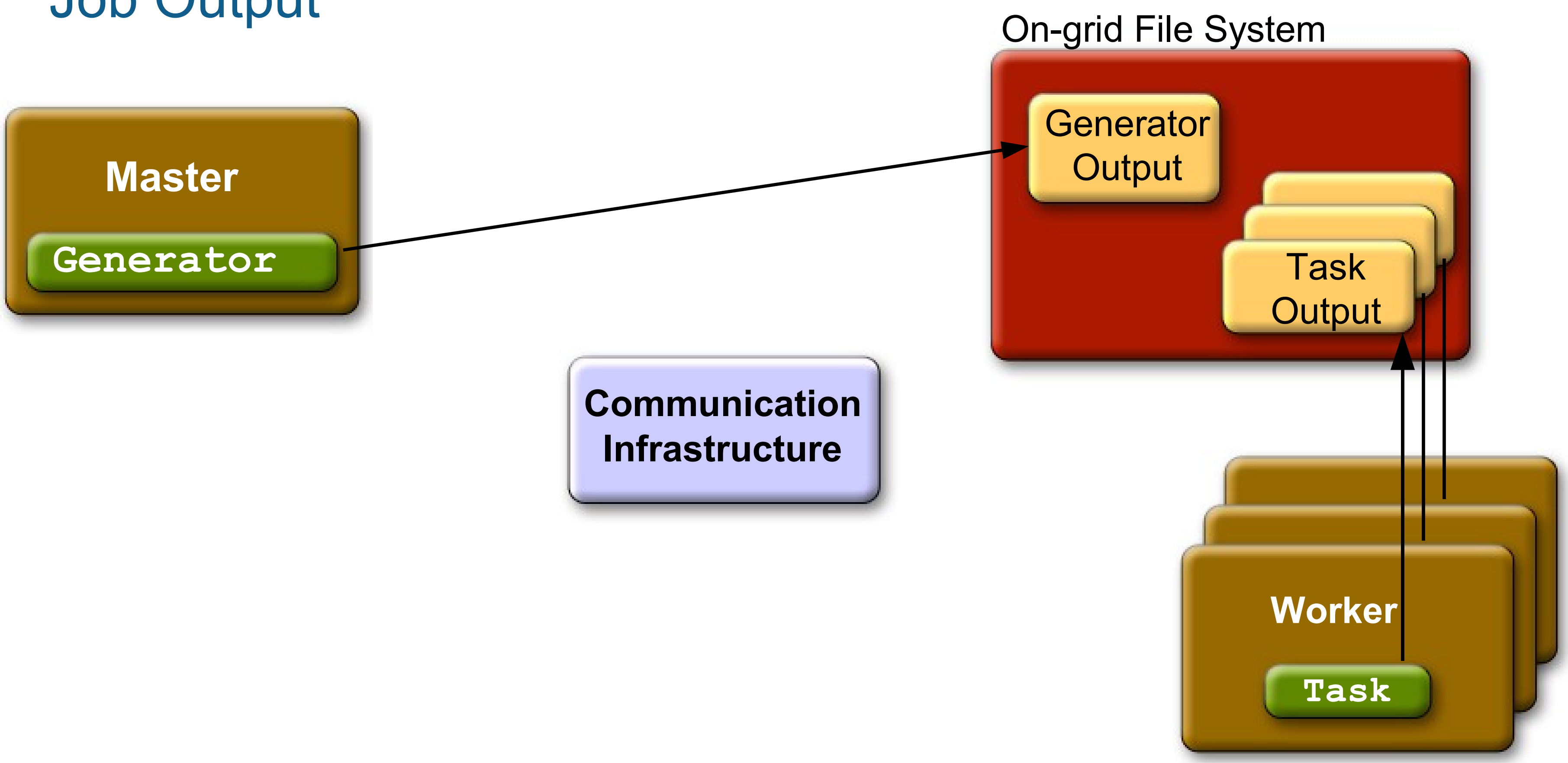
Compute Server

Task Execution and Feedback

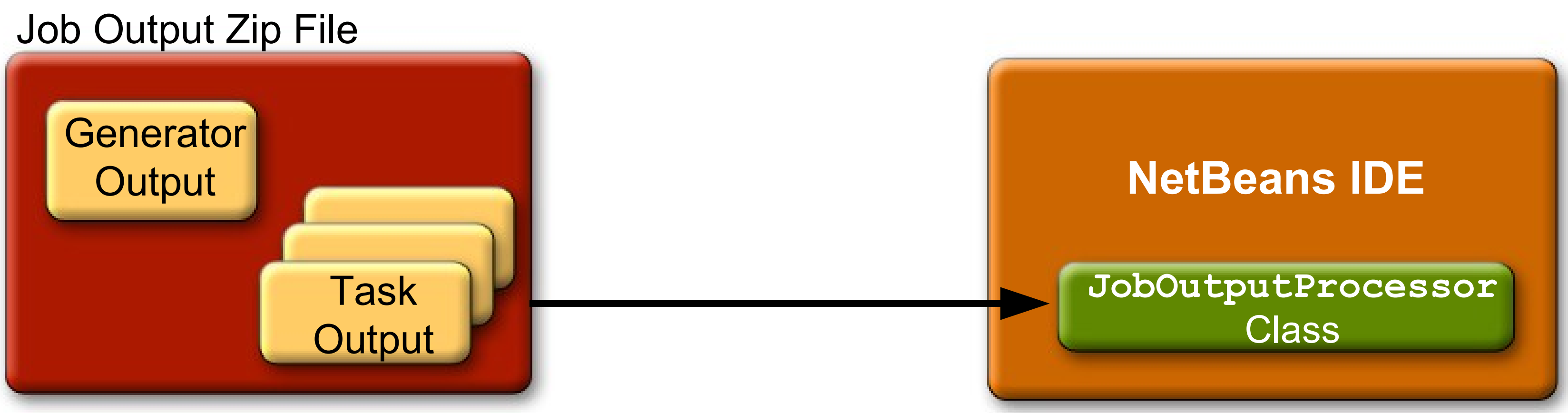


Compute Server

Job Output



Compute Server
Output Processing



Generator Interface

```
public interface Generator<F, TO, GO> {

    public interface Context<GO> {

        void addOutput(GO output);
    }

    public enum State {

        GENERATE,
        WAIT,
        DONE
    }

...
```

Generator Interface (Cont.)

```
public void init(Context<GO> genCtx, String... args)
    throws Exception;

public State getState() throws Exception;

public Task<F, TO> generate() throws Exception;

public void consume(F feedback) throws Exception;

public void done() throws Exception;

}
```

Task Interface

```
public interface Task<F, O> {

    public interface Context<GO> {

        public void setFeedback(F feedback);
        public void setOutput(O output);
        public void addTask(Task<F, O>);
    }

    public void run(Context<F, O> taskCtx) throws Exception;
}
```