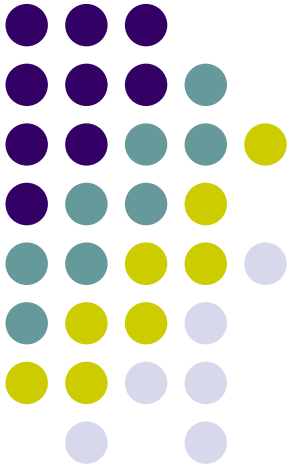
Grid Enabled JaSkel Skeletons with GMarte

J.M. Alonso, V. Hernández, G. Moltó,
Universidad Politécnica de Valencia, Valencia, Spain

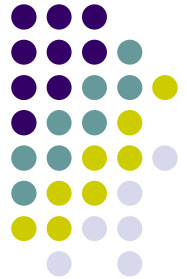
A. Proença, J.L. Sobral
Universidade do Minho, Braga, Portugal



1st IBERIAN GRID INFRASTRUCTURE CONFERENCE

Santiago de Compostela, 16 May 2007

Grid Enabled JaSkel Skeletons with GMarte



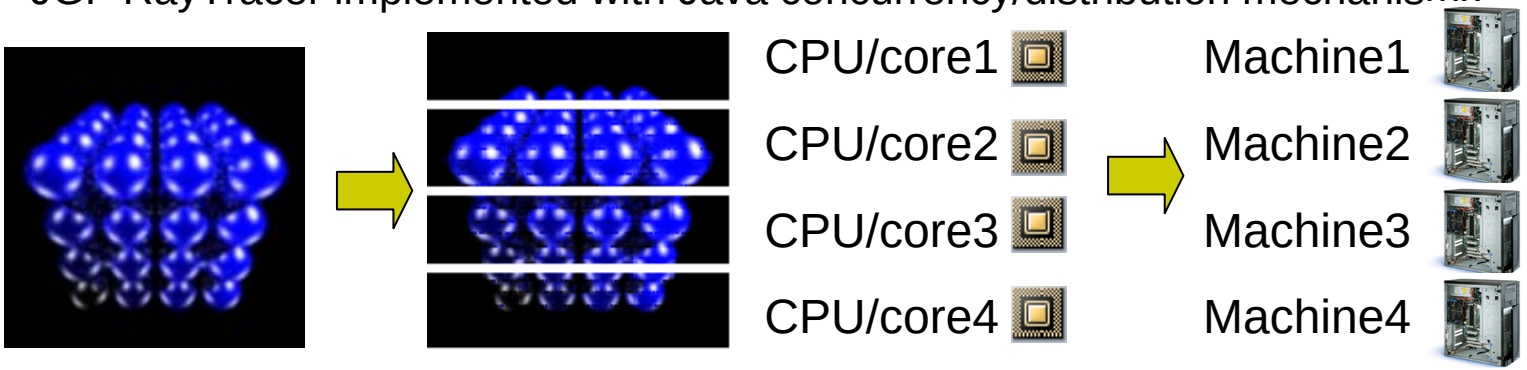
Presentation Outline

- Motivation for a Skeleton Framework
- JaSkel Overview
- GMarte Overview
- Performance Results
- Conclusions and Future work

Motivation for a Skeleton Framework

Traditional application development

- JGF RayTracer implemented with Java concurrency/distribution mechanisms



Sequential

```
RayTracer rt = new RayTracer();
Image result = rt.render(0,500);
```

Multicore/SMP

```
RayTracer rt[]=new RayTracer[4];

for(i=0; i<4; i++)
    rt[i] = new RayTracer();

for(int i=0; i<4; i++)
    new Thread()  {
        void run() {
            res[i] = rt[i].render(/*sub-interval*/);
        }
    }.start();

for(int i=0; i<4; i++)    result =  …
```

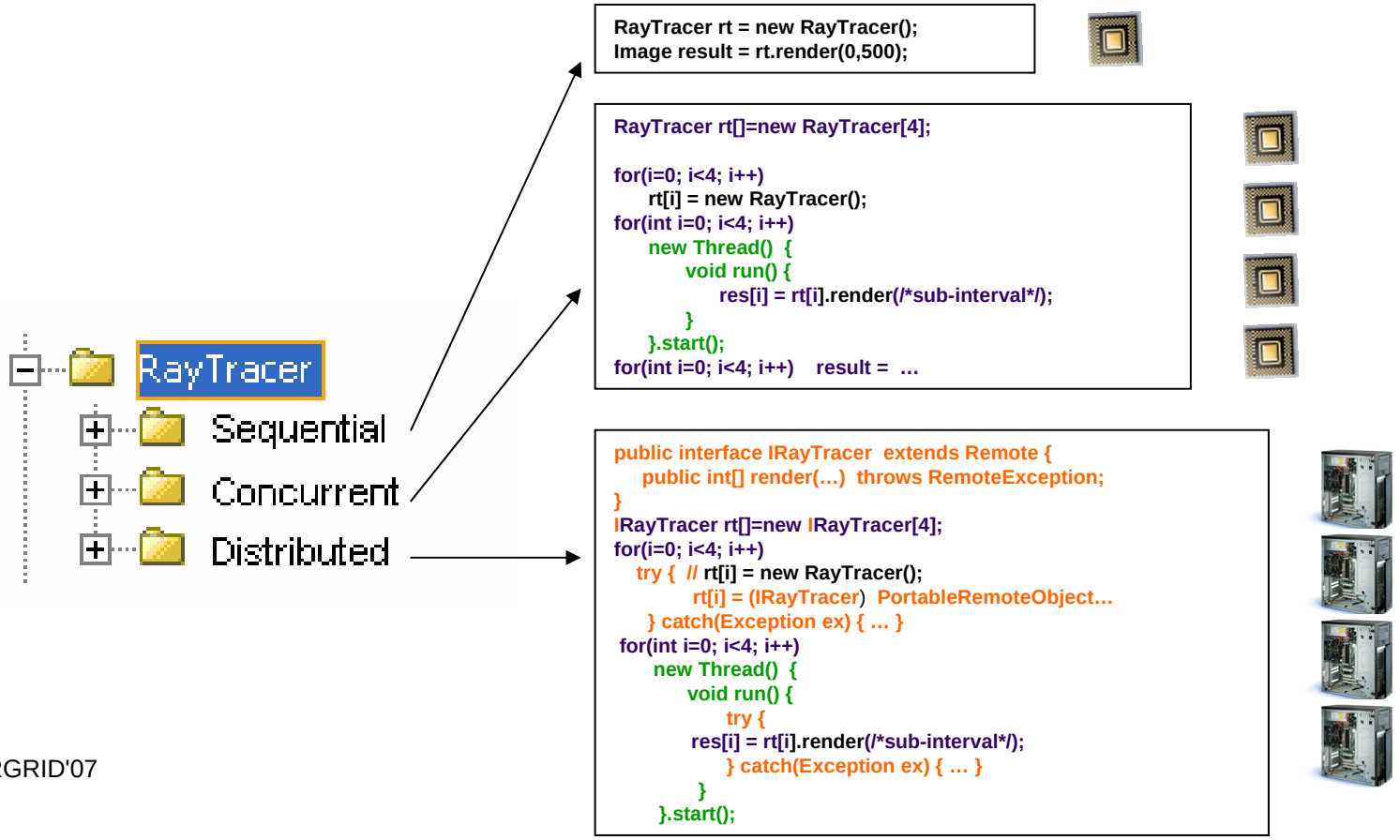
MPP/Cluster/Grid

```
public interface IRayTracer  extends Remote {
    public int[] render(…)  throws RemoteException;
}
IRayTracer rt[]=new IRayTracer[4];

for(i=0; i<4; i++)
   try {  // rt[i] = new RayTracer();
          rt[i] = (IRayTracer)  PortableRemoteObject…
    } catch(Exception ex) { … }

for(int i=0; i<4; i++)
     new Thread()  {
         void run() {
             try {
                 res[i] = rt[i].render(/*sub-interval*/);
             } catch(Exception ex) { … }
```

Motivation for a Skeleton Framework

Traditional application development

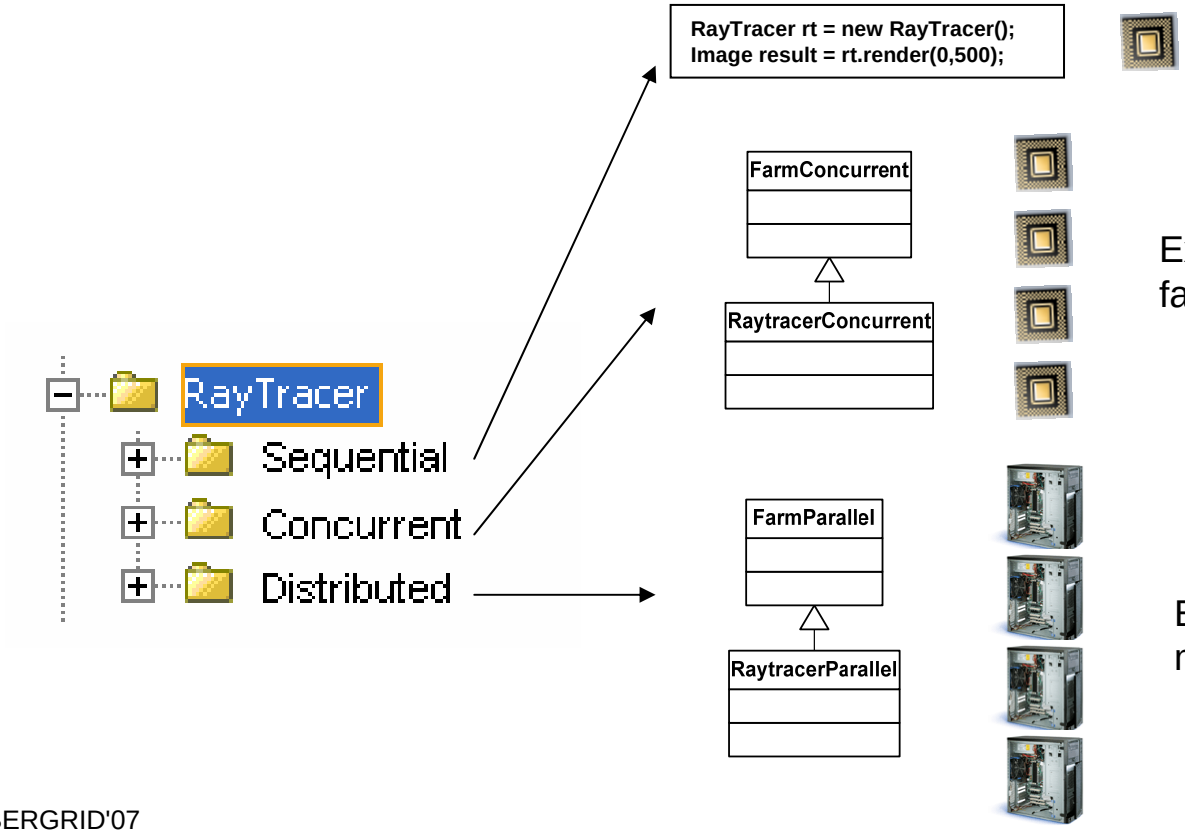
- (Non structured) support to application development



Motivation for a Skeleton Framework

Skeleton-based application development

- Modular (and structured) support to application development



Extend built-in shared-memory farm skeleton

Extend built-in distributed-memory farm skeleton

JaSkel Overview

Java-based skeleton framework

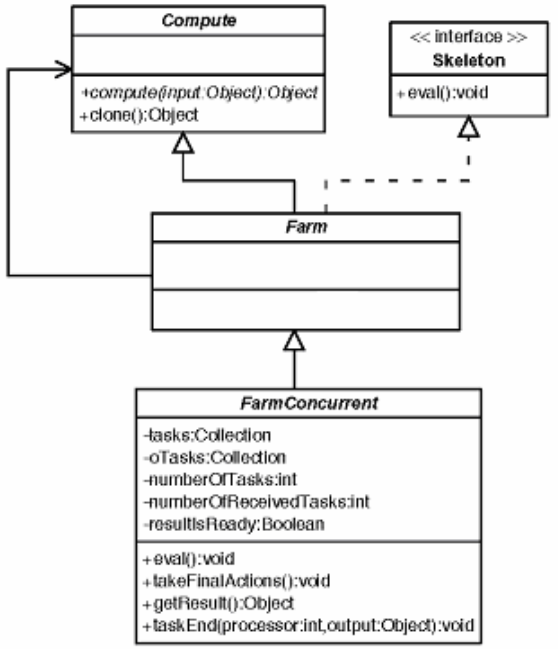
- Provides farm, pipeline and heart-beat skeletons
- Skeletons can be composed (e.g., farm+pipeline)
- Includes sequential skeletons for debugging purposes

Inheritance based framework

- Applications are developed by extending JaSkel skeletons
- Built-in skeletons can be refined by inheritance
- Built-in skeletons are organised into class hierarchies

Distributed execution is an orthogonal feature

- External tools adapt skeletons for distributed execution
- Improves control over skeleton distributed execution (useful for multi-level skeletons)
- Supports multiple distribution middleware

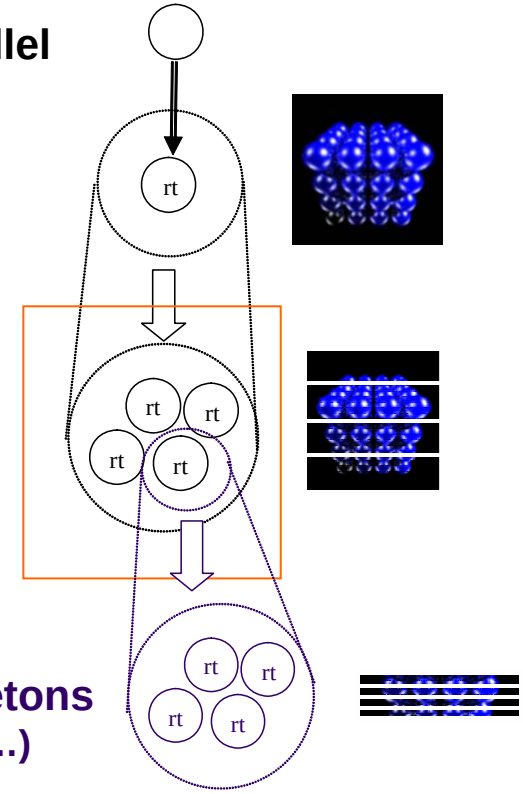


JaSkel Overview

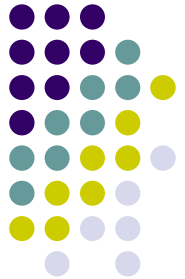
Structured approach to build complex parallel applications

Applying distribution to specific skeletons
(MPI, RMI, GMarte, …)

Composition of farm skeletons
(Farm+Farm, Pipeline+Farm, …)

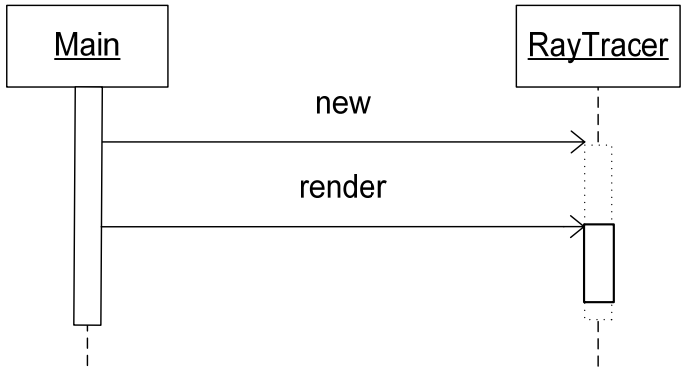


JaSkel Code Example



RayTracer: renders an image of 64 spheres

- Core Functionality

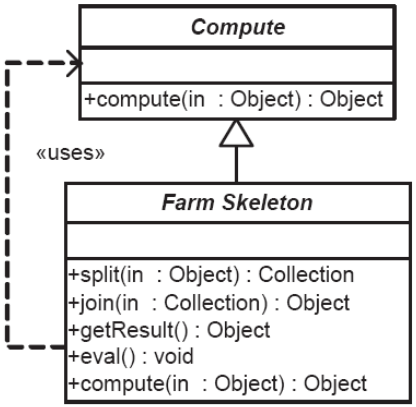



```
public class RayTracer {

    // renders an image
    public int[] render(Interval in) { … }
}


void public static void main(String[] arg) {

    RayTracer rt = new RayTracer();
    Interval in = new Interval(0,500);
    Image result = rt.render(in);
}
```

JaSkel Code Example (cont.)

Example: RayTracer – Farm Parallelisation

```
public class RayTracer extends Compute  {

    public int[] render(Interval in) { … }

    public Object compute(Object input) {
        return this.render((Interval) input);
    }
}

public class Farmer extends FarmConcurrent {
    public Farmer(Computer worker, Object inputTask) {
        super(worker, inputTask);
    }
    public Collection split(Object initialTask) {
        return(/* split interval */);
    }
    public Object join(Collection partialResults) {
        return(/* join partial results */);
    }
}
```

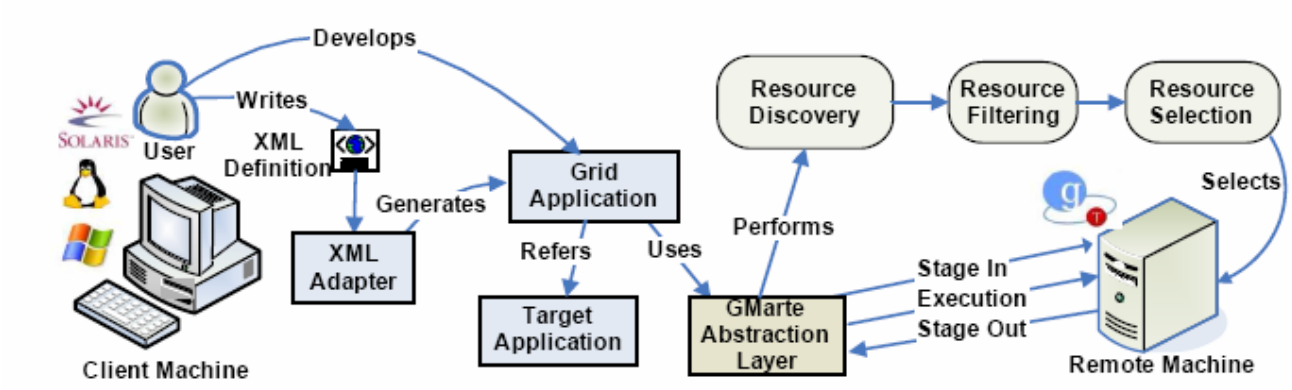



```
void public static void main(String[] arg) {

   RayTracer rt = new RayTracer();
   Interval in = new Interval(0,500);

   Farmer g = new Farmer(rt,in);              // farmer
   g.eval();                          // starts the farming process
   Image result = (Image) g.getResult();   // get results
}
```

GMarte Overview

Java library to interact with computational Grids

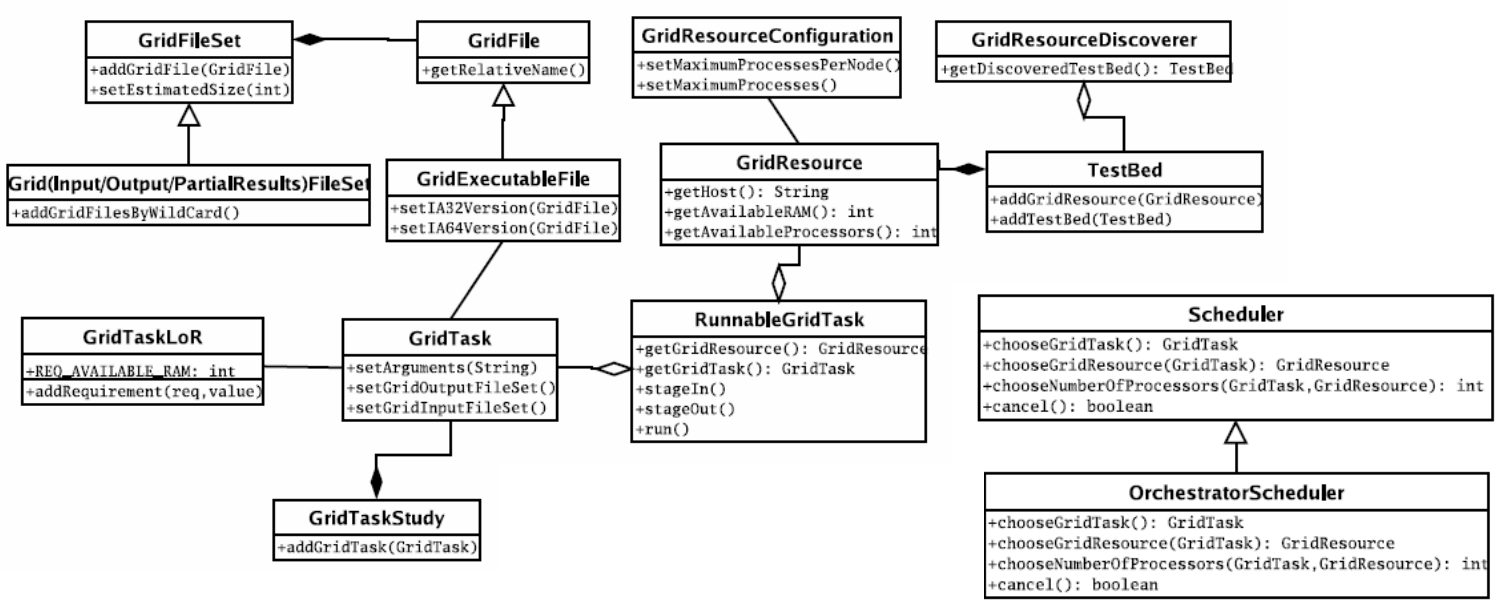
- Exposes an high level object-oriented API
- Simplifies the process of remote task execution by hiding grid-specific issues
  - **Resource discovery and selection**
  - **Grid meta-schedule**
  - **File stage-in and stage-out**



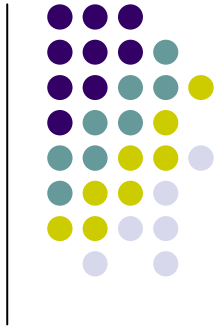
GMarte Overview

API overview

- GridTask, GridTaskStudy – tasks to execute on computational Grid
- GridResource, GridTestBed – Grid resources to use
- Scheduler – Meta-scheduler functionality



GMarte Code Example



1. **Specify task to execute**

```
GridTask gt = new GridTask("My Java Task");
gt.setGridExecutableFile(new JavaGridExecutableFile());
gt.setArguments("-jar myapp.jar schema.xsd");

GridInputFileSet gifs = new GridInputFileSet();
gifs.addGridFile("/home/user/schema.xsd");
gt.setGridInputFileSet(gifs);

GridOutputFileSet gofs = new GridOutputFileSet();
gofs.addWildCard("*.txt");
gt.setGridOutputFileSet(gofs);

GridTaskStudy gts = new GridTaskStudy();
gts.addGridTask(gt);
```

2. **Specify resource to use (optional)**

```
GridResource gr = new GridResource("amachine.lab.com");
TestBed tb = new TestBed();
tb.addGridResource(gr);
```

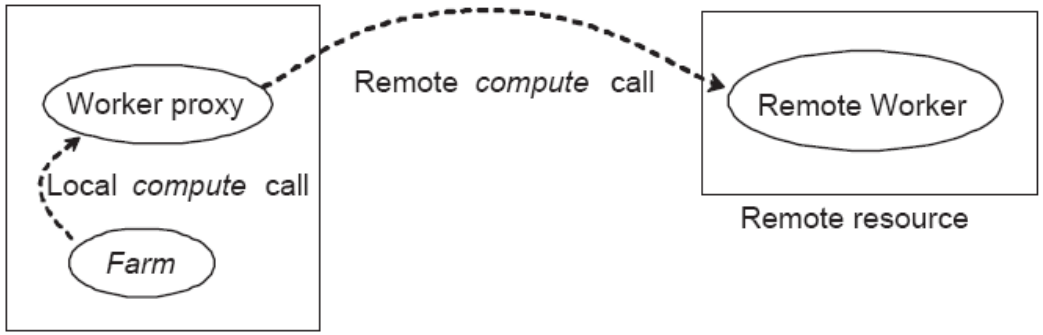
3. **Call GMarte scheduler**

```
Scheduler scheduler = new OrchestratorScheduler(tb, gts);
scheduler.start();
scheduler.waitUntilFinished();
```

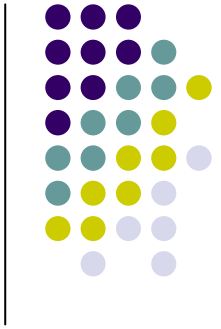
JaSkel-GMarte binding

Enables JaSkel applications to run on computational Grids

- Based on a code generator that rewrites skeleton implementations
  - **Client Skeleton (*Compute* instances) are replaced by proxies**
  - ***Compute* instances become stand-alone applications**
  - **Communication between client and server performed by files**



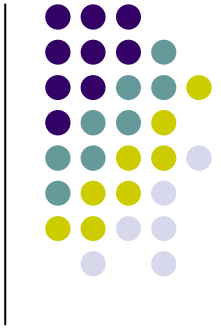
JaSkel-GMarte binding



- Client side code (worker proxy)
  - **Replaces *compute* method implementation**
    1. Writes method arguments to a file
    2. Creates a new GMarte *GridTask*
    3. Calls GMarte *scheduler* to execute task on remote resource
    4. Reads results from staged-out file
  - **GMarte tasks**
    1. select a remote resource
    2. stage-in the input files
    3. request the remote execution of the skeleton code
    4. stage-out the results.

```
public class RayTracer extends Compute { // worker proxy
    public Object compute(Object obj) {
            ... // generate RPC request (plain text file)
          GridTask gt = new GridTask("JaSkel worker);
          GridInputFileSet gifs = new GridInputFileSet();
          gifs.addGridFile("jaskel.jar"); // remote skeleton jar file
          gifs.addGridFile(/* RPC request file name */);
          gt.setGridInputFileSet(gifs);
          GridOutputFileSet gofs = new GridOutputFileSet();
          gofs.addGridFile(/* result file name */);
          gt.setGridOutputFileSet(gofs);
          ... // request task execution to GMarte
          ... // wait & read result from staged-out file
       return(/*result*/);
    }
}
```

JaSkel-GMarte binding



- Server side code (remote worker)
  - **Generates a new *main***
    1. creates a *Compute* instance
    2. reads *compute* arguments to a file
    3. calls *compute* method
    4. Write results to a file

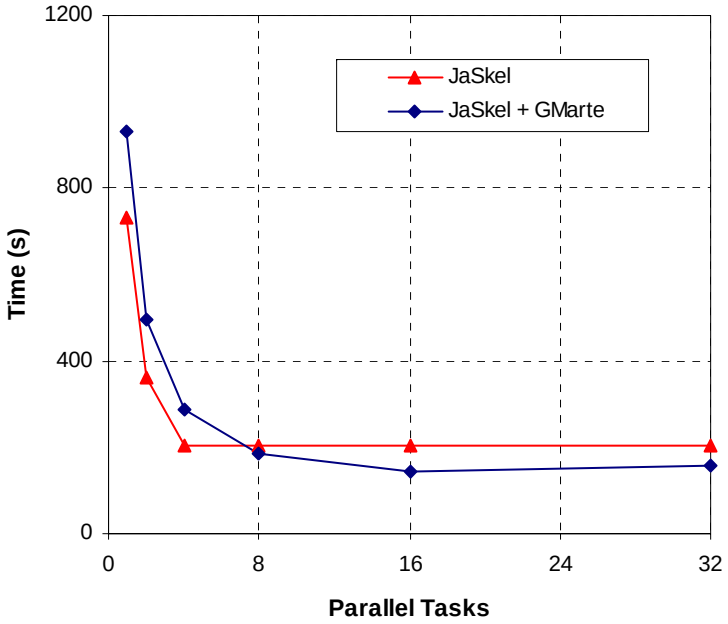
```
public static void main(String args[]) {
   RemoteRayTracer myWorker = new RemoteRayTracer();
   Object myData = ... // read data from file staged in by GMarte
   Object result = myWorker.render(myData);
   ... // save result into output file to be staged out by GMarte
}
```

  - **Generated code does not depend on GMarte**

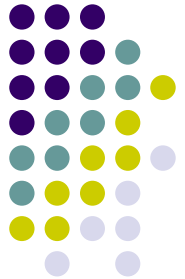
Performance Evaluation

JGF RayTracer (4000x4000 Image)

- 4-CPU Xeon 5130 vs computational Grid (two clusters)



Grid Enabled JaSkel Skeletons with GMarte



Conclusions

- Integrated framework address execution on multi-core, clusters and Grids
- Goals achieved:
  - **JaSkel framework supports a new distribution middleware (GMarte)**
  - **GMarte binding strongly reduces the amount of generated code and is loosely dependent of Grid related issues**

Future Work

- Tighter integration of JaSkel and GMarte meta-scheduling
  - **Support for multi-level skeletons**
- Evaluation on Grid environments with large scale applications