

ASIA@HOME

Tim

(Yu-Ting Chen)

<yuting.chen@twgrid.org>

AutoDock 3 & 4

- Run on Linux machines
- Scientists may want to run their jobs on both versions since AutoDock 3 and 4 use different docking algorithms and input file formats.
- Therefore, in BONIC server, they are two different applications rather than one application with two different versions

AutoDock 3 & 4

- Use shell script for startup logic
- Use wrapper to run the shell script
- The actual AutoDock 'applications' are zipped
- The startup logic:
 - Read the input config file;
 - Find the zip file and unzip it;
 - Find the executable in the unzipped folder;
 - Feed the input files and execute AutoDock

AutoDock 3 & 4 (Input config)

- The input config file name is hard-coded, so don't change its name
- However, feel free to change the actual input file names in the config file

- Example:
inputConfig-autodock3:

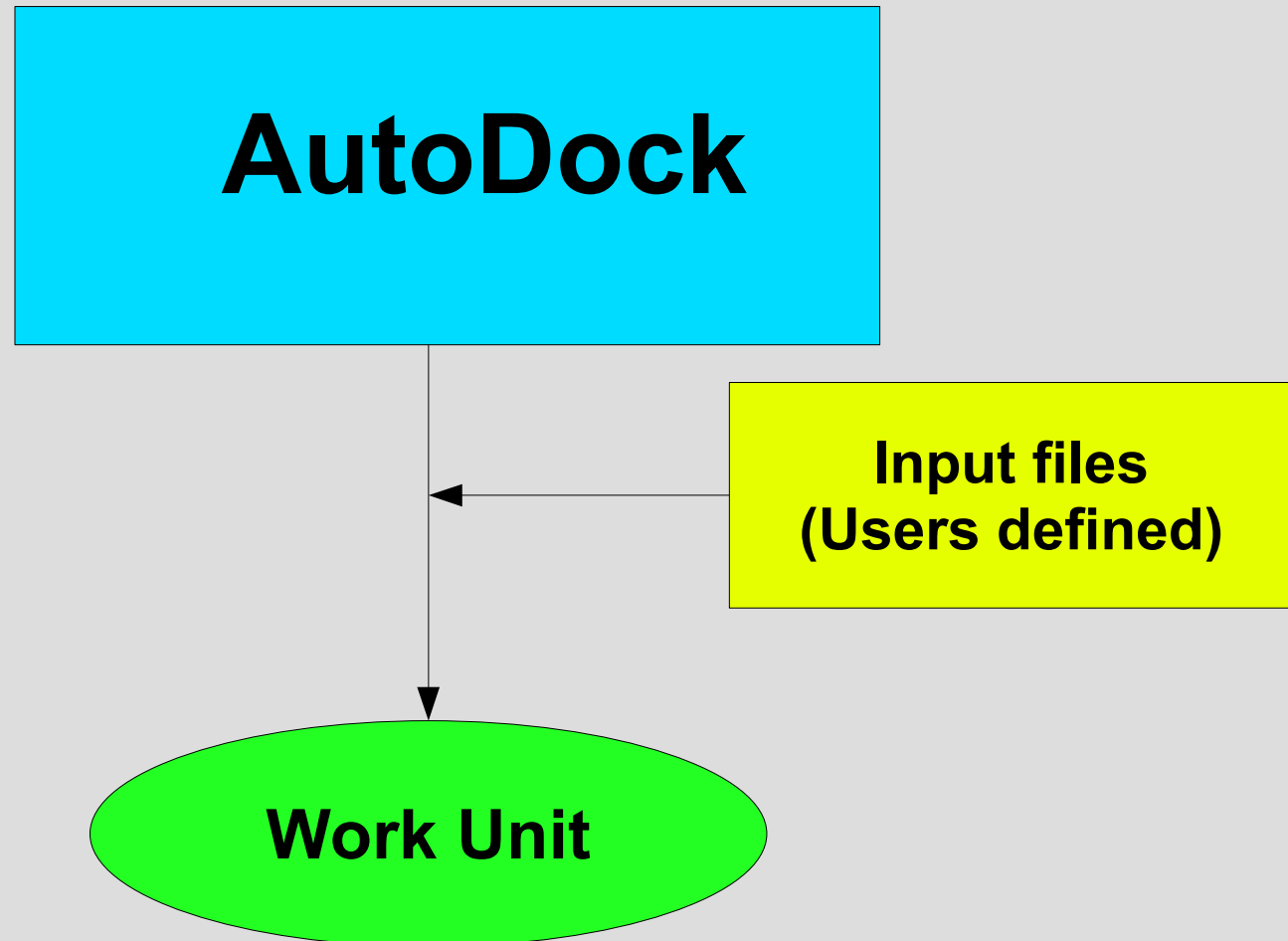
```
dpf=ind-autodock3.dpf  
pdbq=ind-autodock3.pdbq  
data=input-autodock3.tar.gz
```

AutoDock 3 & 4 (Input config)

- Example:
inputConfig-autodock4:

dpf=x1hpv.dpf
data=input.tar.gz

AutoDock 3 & 4



Problems

- The progress bar on BOINC client is inaccurate
- Can't suspend, resume and stop the job from client's GUI (Although the client GUI shows that the job has been suspended, resumed, or stopped, in fact the process still runs on volunteer's PC).

Solution(Partial)

- Use BOINC API
- BOINC API functions are embedded in AutoDock4 source code
- Need to defined BOINC flag and recompile
- However, it lacks of job fractions report and check points
- At least volunteers can suspend/resume/stop theirs jobs

Solution(Partial)

- How if
 - No source code can be found?
 - Impossible to build native BOINC
 - BOINC API is not embedded?
 - Great pain to insert BOINC API functions into the code
- Job manipulation can be solved by using job.xml in wrapper mode

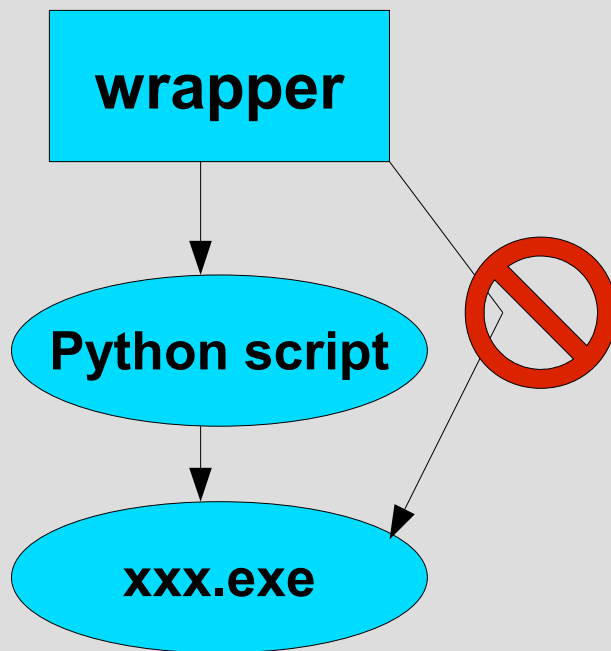
R

- Run on Windows machines
- Uses python script for the startup logic
- Use py2exe to convert python script to exe file
- Use wrapper to run the exe file

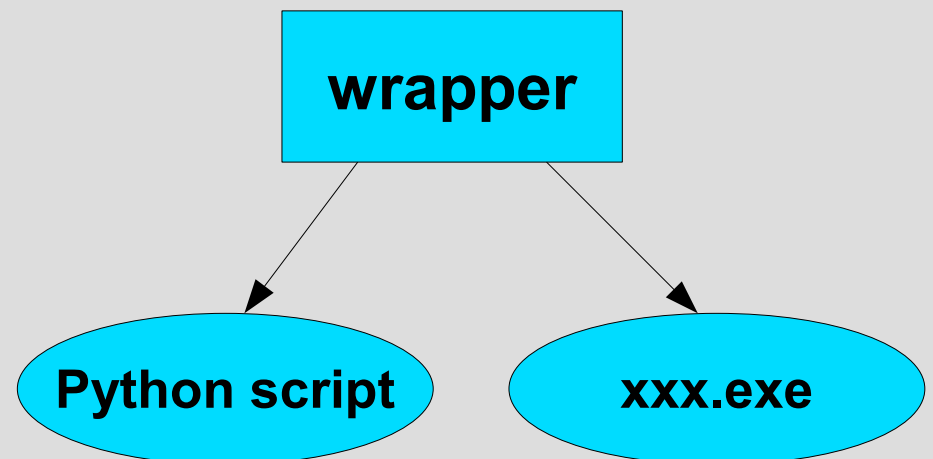
R(Solution of job manipulation)

- Before we usually used python script to do some job preparations (e.g. unzip) and fork a process to run executable
- Now use job.xml to define two tasks

(A) Wrapper can't control executable



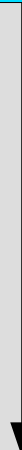
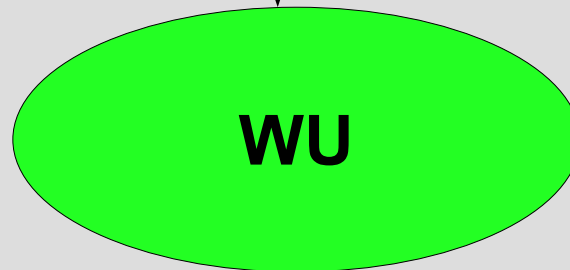
(B) Wrapper can control executable



R(Solution of job manipulation)

- Can suspend/resume/stop job
- Coarse granularity check points and job fractions report

R



Social Simulation

- Agent-based simulation becomes more and more important in many domains, such as social science, economics, and biology.
- To run larger scale of simulations or more complex models, the requirement of computing power grows fast.
- More and more simulations are programmed and deployed on distributed systems such as clusters and grids.
- Why not run simulations on BOINC!

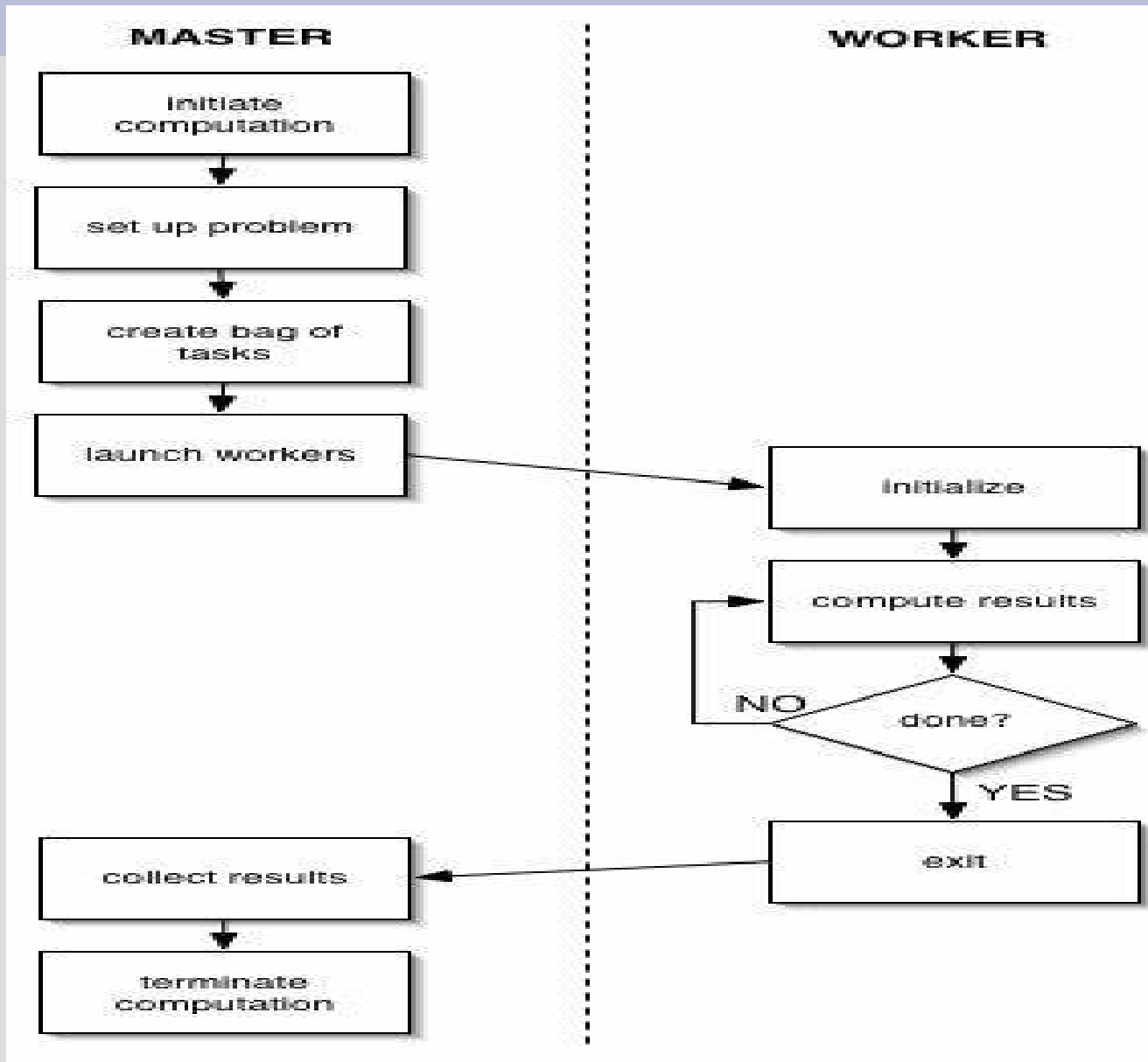
Social Simulation

- A toy simulation program is written with RepastJ, a Java based agent-based simulation toolkits.
- Run distributively by using Terracotta and master/worker pattern

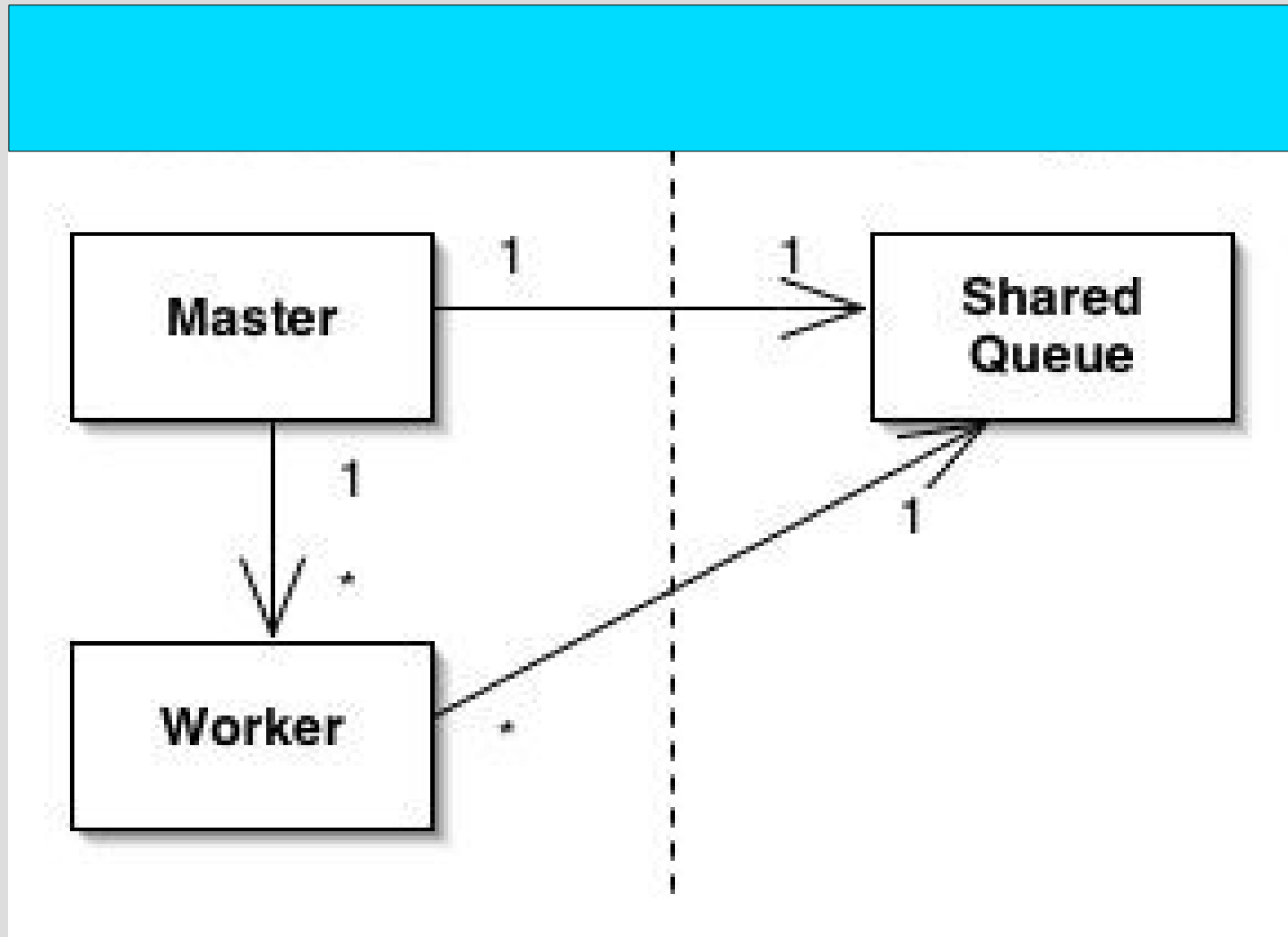
Terracotta

- “Terracotta is Java infrastructure software that allow you to scale your application for use on as many computers as needed, without expensive custom code or databases.” --The Definitive Guild to Terracotta
- Terracotta Distributed Shared Objects (DSO) is at the core of operations in Terracotta clusters

Master/Worker Pattern



Master/Worker Pattern



Master

- The Master's logic:

Set up simulation model;

For each tick

 Prepare jobs;

 Submit them into shared queue;

 wait until all jobs are done;

 Collect the results and update view

Worker

- The worker's logic:

while there are jobs in shared queue
take a job
run the job

Start master

The screenshot displays a Java IDE window titled "Repast" with a toolbar and a "Run: 1" indicator. The main workspace is divided into several panels:

- Buttons Display:** A window showing a scatter plot of colored dots on a black background, representing the spatial distribution of buttons.
- Buttons:** A graph titled "Buttons" showing the cluster size (y-axis, labeled "cluster" and scaled by $\times 10^2$) over time (x-axis, labeled "time" and scaled by $\times 10^2$). The y-axis ranges from 0.0 to 2.0, and the x-axis ranges from 0.0 to 4.0. A red dot labeled "MaxCluster" is visible at approximately (4.0, 2.0).
- A Repast Model Set:** A configuration panel with two tabs: "Parameters" and "Custom Actions". Under "Parameters", there are fields for "Buttons" (200), "SpaceSize" (50), "CellDepth" (5), "CellHeight" (5), "CellWidth" (5), "PauseAt" (-1), and "RandomSeed" (1). An "Inspect Model" button is also present.
- RePast Output:** A window at the bottom showing the output of the simulation, with the text "Start sending works" and "All works are sent".

Below the "Buttons Display" window, a snippet of Java code is visible:

```
}  
public void setSpaceSize(int spaceSize) {  
    this.spaceSize = spaceSize;  
}
```

Start workers

The screenshot displays the Repast simulation environment with several windows:

- Buttons Display:** A window showing a spatial distribution of buttons (represented as small colored squares) on a black background.
- Buttons:** A graph showing the cluster size (y-axis, labeled "cluster" and scaled by $\times 10^2$) over time (x-axis, labeled "time" and scaled by $\times 10^2$). The curve shows a sigmoidal growth, starting near zero, rising sharply between $t = 1.0 \times 10^2$ and $t = 2.0 \times 10^2$, and leveling off at approximately 2.0×10^2 . A red dot indicates the "MaxCluster" value.
- A Repast Model Set:** A configuration window with the following parameters:
 - Buttons: 200
 - SpaceSize: 50
 - CellDepth: 5
 - CellHeight: 5
 - CellWidth: 5
 - PauseAt: -1
 - RandomSeed: 1
- RePast Output:** A text window showing the following log output:

```
Start sending works
All works are sent
thisClusterSize = 97 ClusterSize = 197
Start sending works
All works are sent
thisClusterSize = 197 ClusterSize = 197
Start sending works
All works are sent
```

How to kill workers

- When the master decides to stop the simulation, we need to make workers commit suicides
- Master uses Terracotta Dso Cluster event listener to find out the number of connecting workers
- According to the number of workers, master submits as many suicide jobs into shared queue.
- Worker commits suicide when it gets the cold-blooded job.

Social Simulation

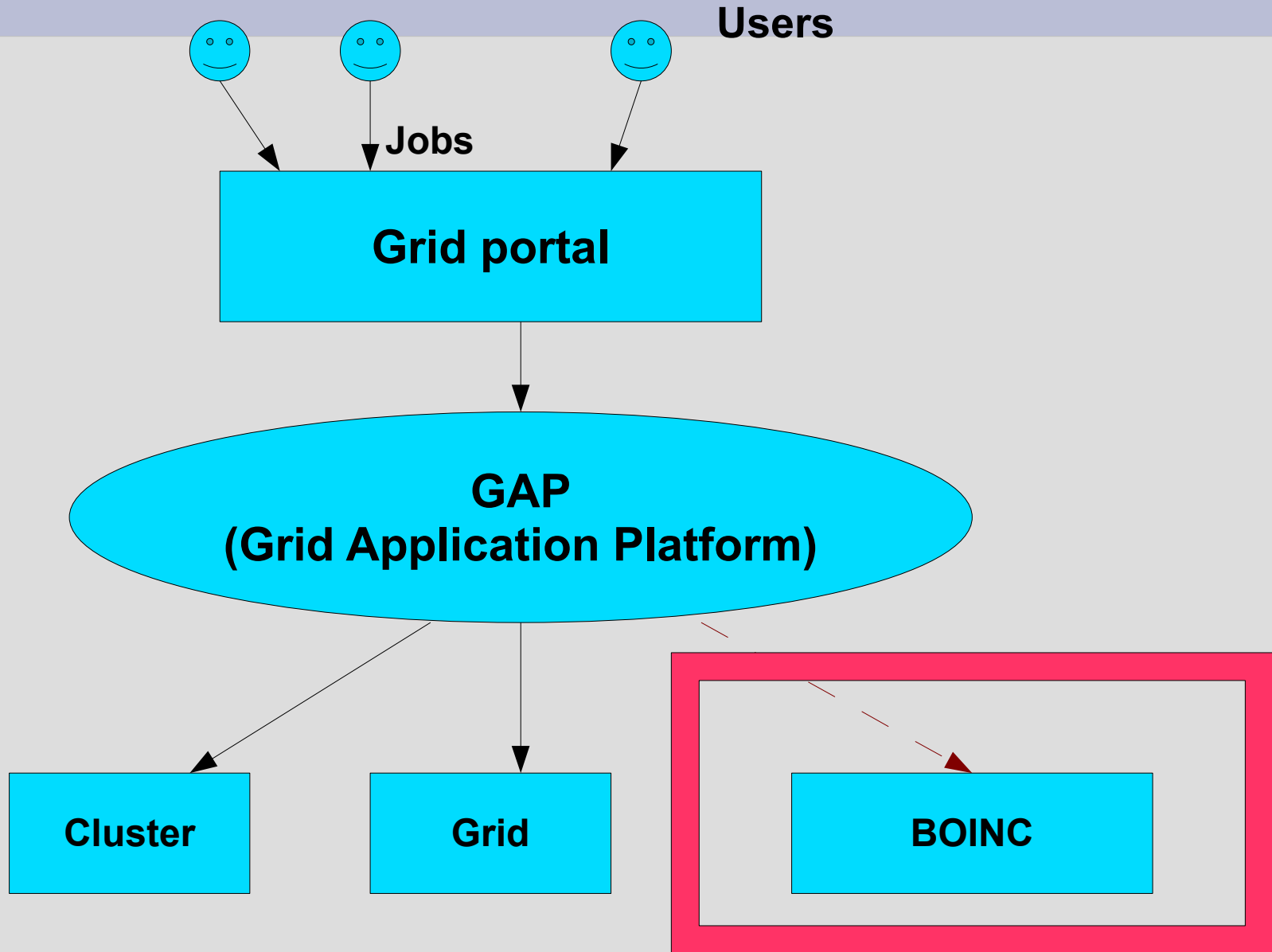
The challenges:

- Performance issue:
 - Performance of terracotta on different networks
 - Enhance the master/worker framework
 - Split proper size per job
- Porting on BOINC:
 - Terracotta uses dso-java.bat(.sh) to build up its running environment and then executes java. Wrapper can't control processes that .bat forked

Integrate BOINC and Grid

- EDGeS develop their Bridge server to connect BOINC and Grid
- We develop GAP(Grid Application Platform) to integrate different grids and clusters environments
- We want to connect BOINC with our GAP

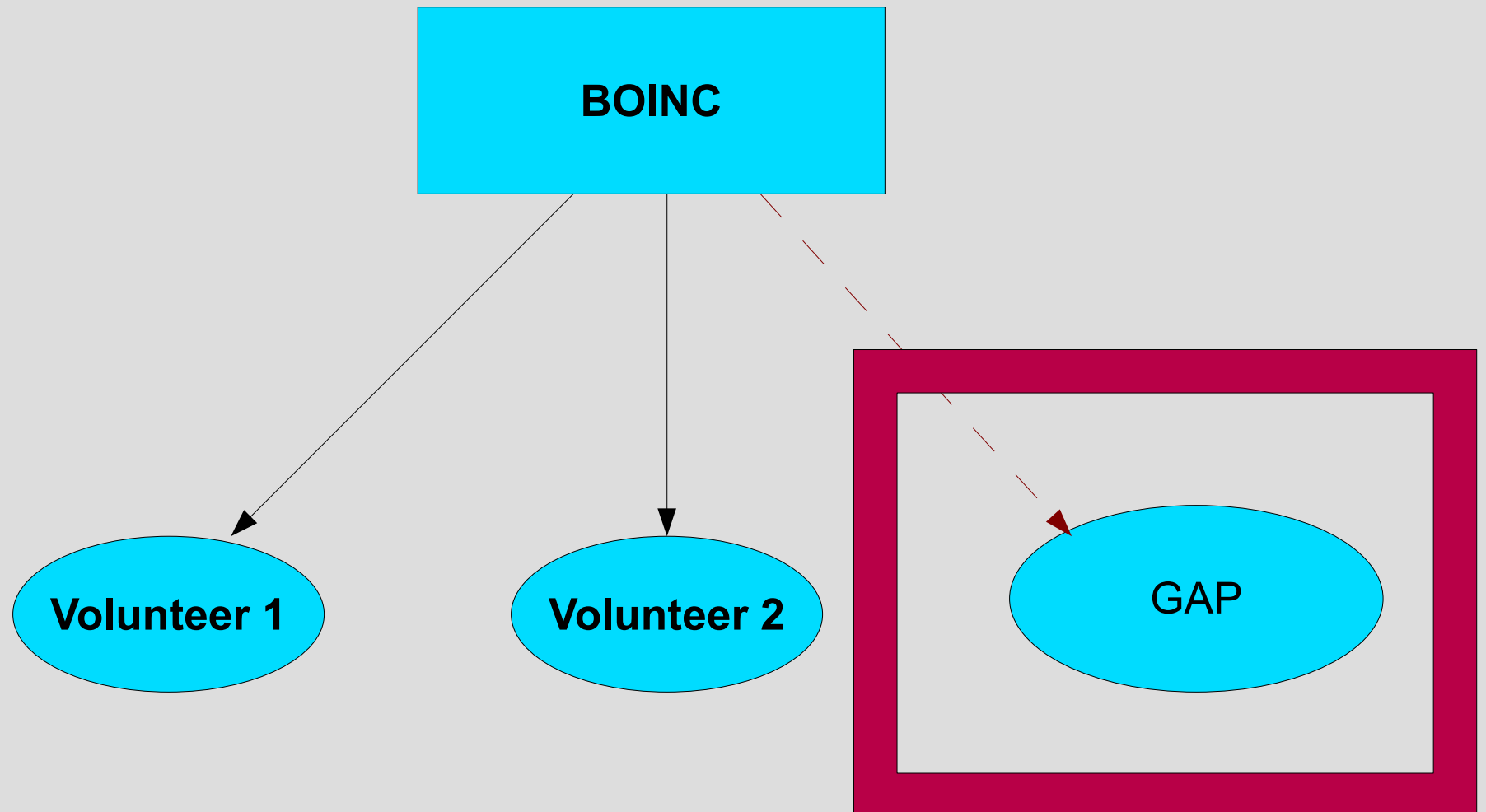
Integrate BOINC and Grid



Integrate BOINC and Grid

- BOINC may accept users' submitted jobs in two levels
 1. Accept only input data. (Apps on BOINC server are static)
 2. Accept both application and input data. (Need to port new App onto BOINC server dynamically)
- Security issue
Both levels can be pretty dangerous!

Integrate BOINC and Grid



Integrate BOINC and Grid

- GAP as a super volunteer
- More secure!