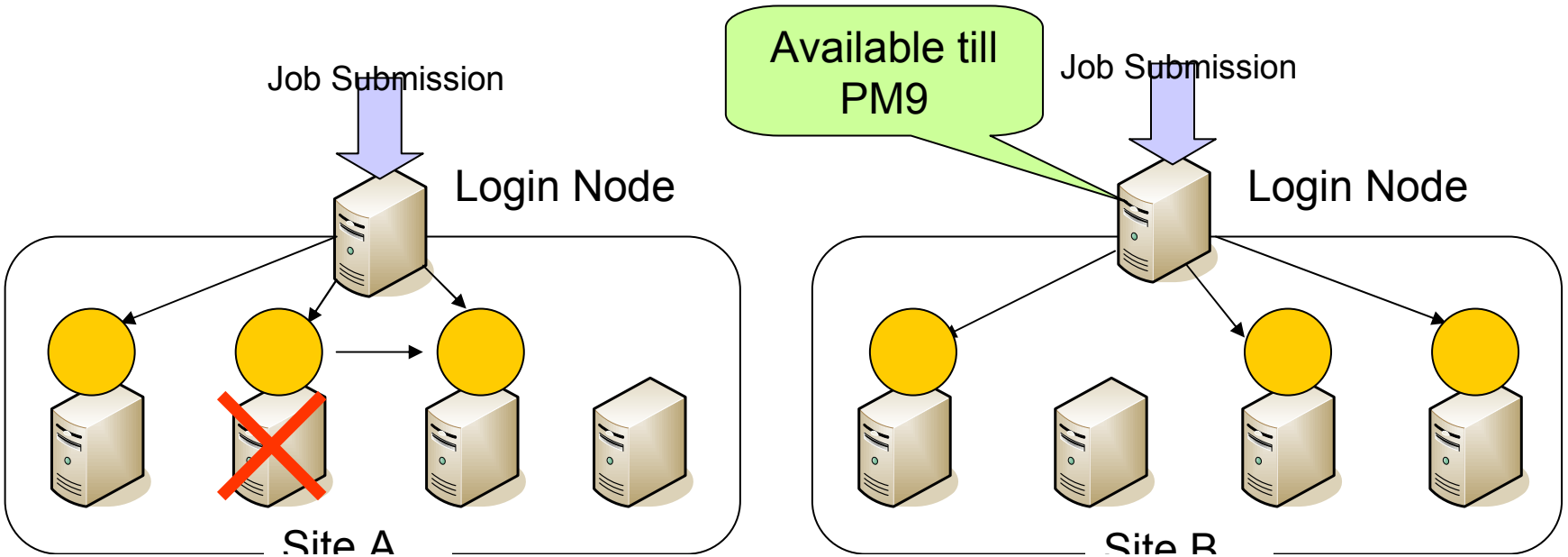
Autonomically-Adapting Master-Worker Programming Framework for Multi-Layered Grid-of-Clusters

Hitoshi Aoki[1], Hidemoto Nakada[2],
Kouji Tanaka[3], Satoshi Matsuoka[1,4]

1.Tokyo Institute of Technology
2. National Institute of Advanced Industrial Science and Technology (AIST)
3.Waseda University
4.National Institute of Infomatics

Background

- Grid : cluster of clusters
- Each cluster is managed by some queuing system
  - PBS, Grid Engine, Condor
  - Availability of nodes depends on the administration policy
  - Available node set will change dynamically



Background

- Master-Worker style programming
  - Latency tolerant
  - Fault tolerant
  - Lot of problems can be mapped on to this style
    - Genetic algorithms
    - Branch and bound method
    - Parameter sweep

Background

- Several globally distribute grid middleware are proposed
  - BOINC / Condor type
    - Robust
    - Very Course Grained
      - Not suitable for fine-grained master-worker
  - MPI type
    - High speed
      - Can map fine-grained master-worker
    - Fragile
      - Even though the programming style itself is robust, with MPI you cannot leverage the robustness
      - Cannot add / remove participating nodes dynamically

Goal

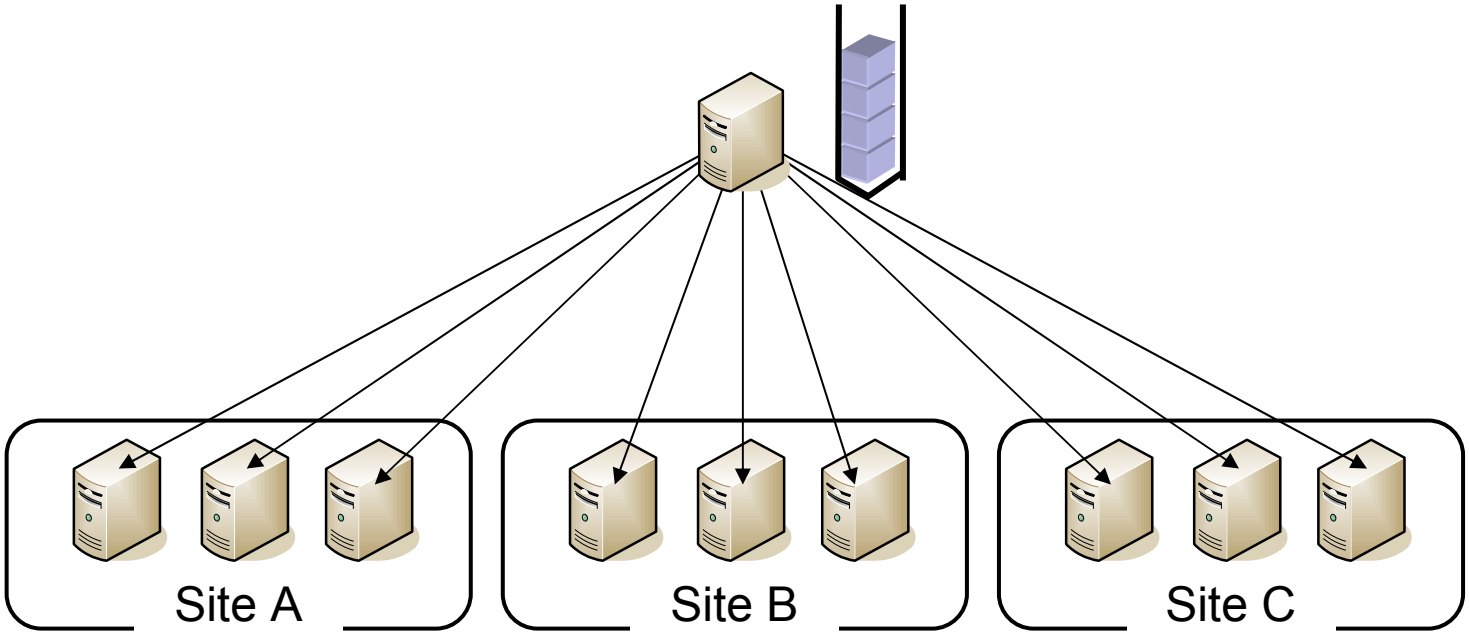
- Propose a grid middleware and programming framework suitable for master-worker style, that can leverage existing general grid configurations
  - Multi-layered
  - Robust
    - Autonomic node tree configuration
  - Affinity with batch queuing systems
    - Node can be added or removed

Overview of this talk

- What is Master-Worker programming
- Proposal of Jojo2
  - Requirement
  - Architecture
  - Programming API
- Evaluation
- Conclusion

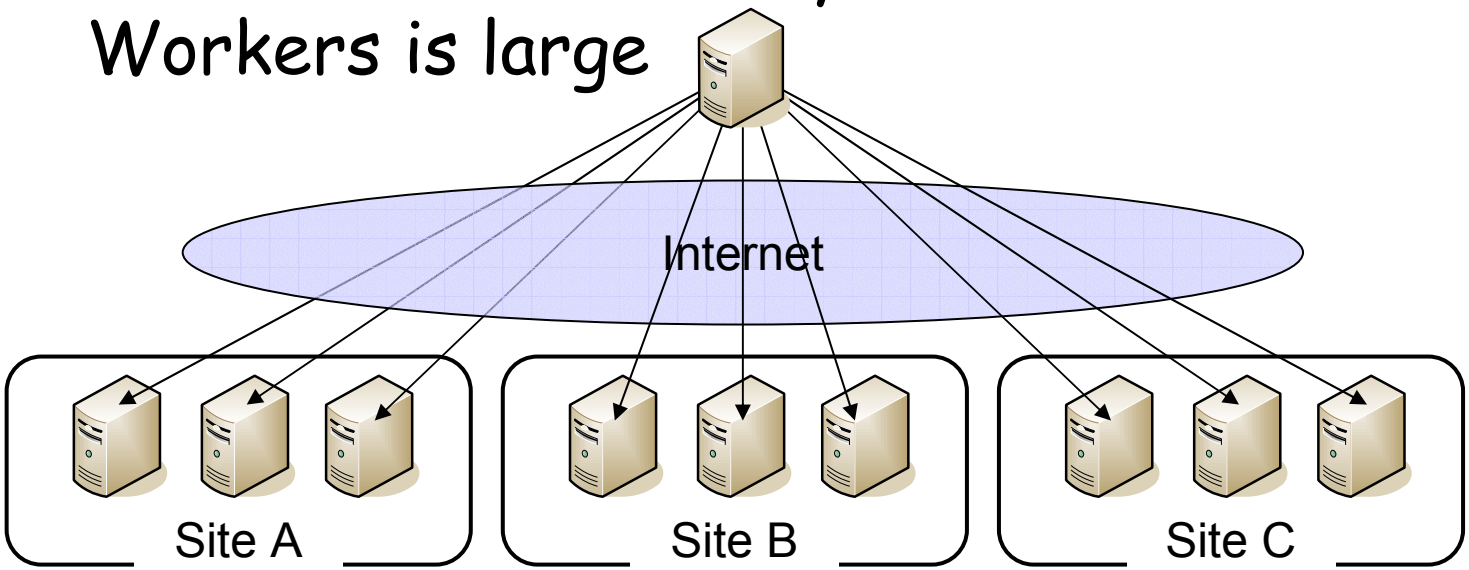
Master-Worker

- Divide problems into small sub problems
- Master manages a queue to keep sub problems
- Distribute them to the workers



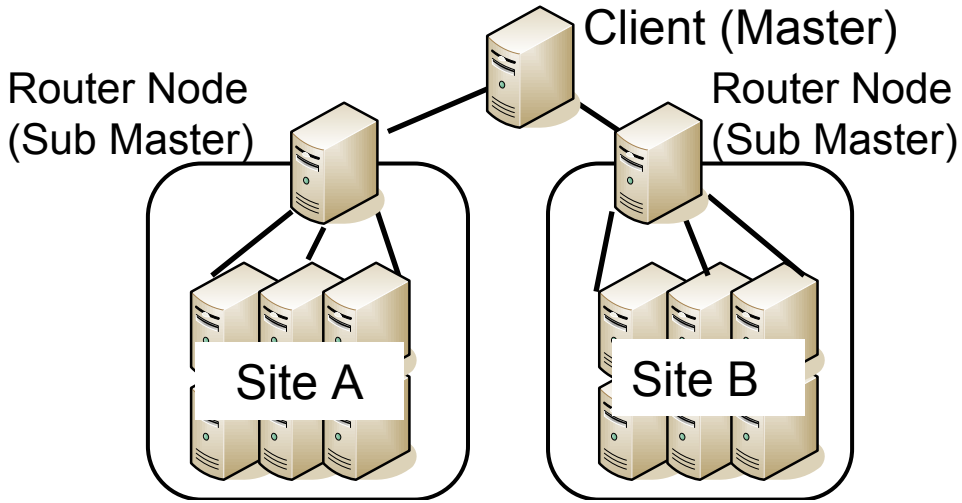
Simple Master-Worker

- Scalability
  - With several hundreds of nodes, Master will be overloaded and become bottleneck
- Efficiency
  - Communication latency between Master and Workers is large



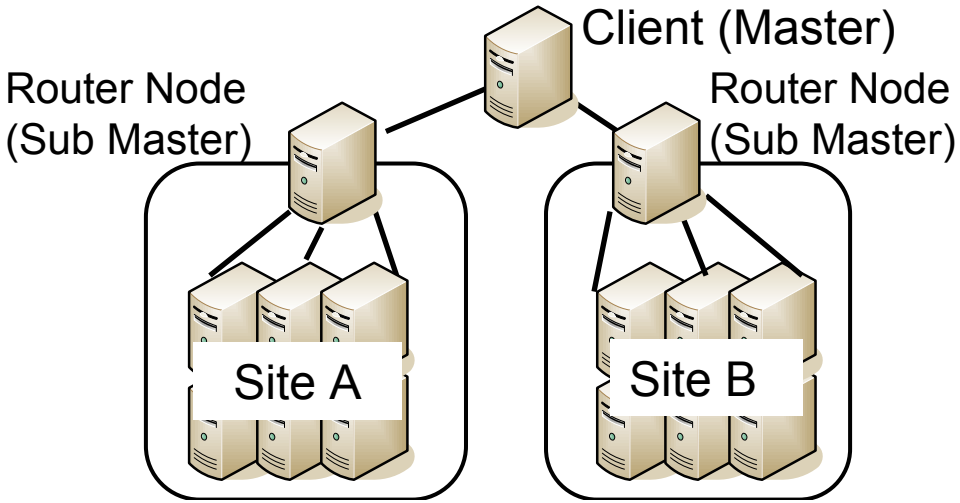
Hierarchical Master-Worker

- Introduce 'Sub-master' between Master and Worker
- Scalability
  - Distribute loads on Sub-masters
- Efficiency
  - Sub-masters are near from workers



Difficulties of hierarchical Master-worker

- Configuration is rather complex
  - Difficult to configure, especially in the dynamically changing environment



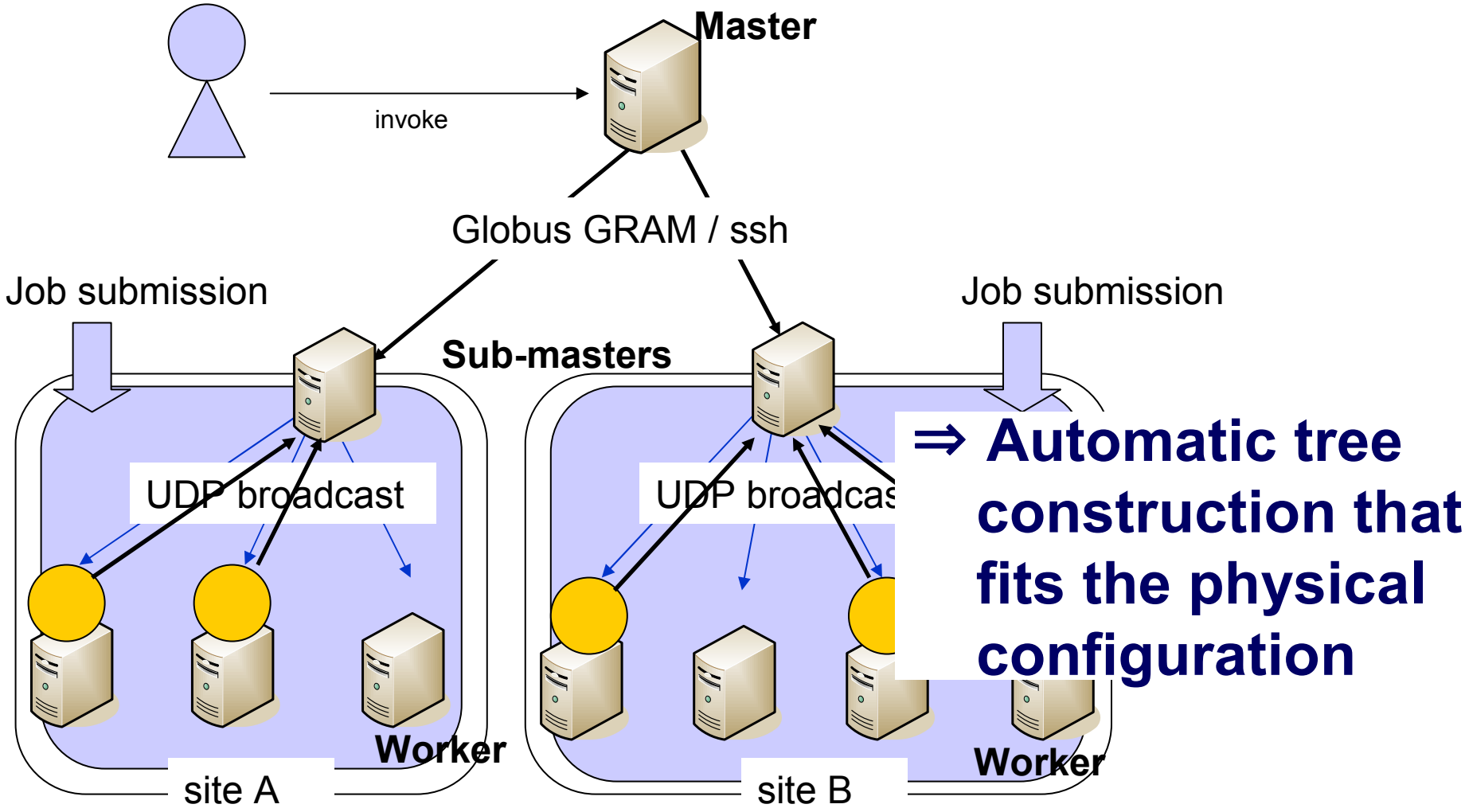
Requirements for grid middleware framework for Master-worker

- Robustness
  - Have to survive changing environment
    - Node might be added and removed during execution
- Easiness
  - Have to be configured semi-automatically
  - Have to be easy to program on it

Generic Design of Jojo2

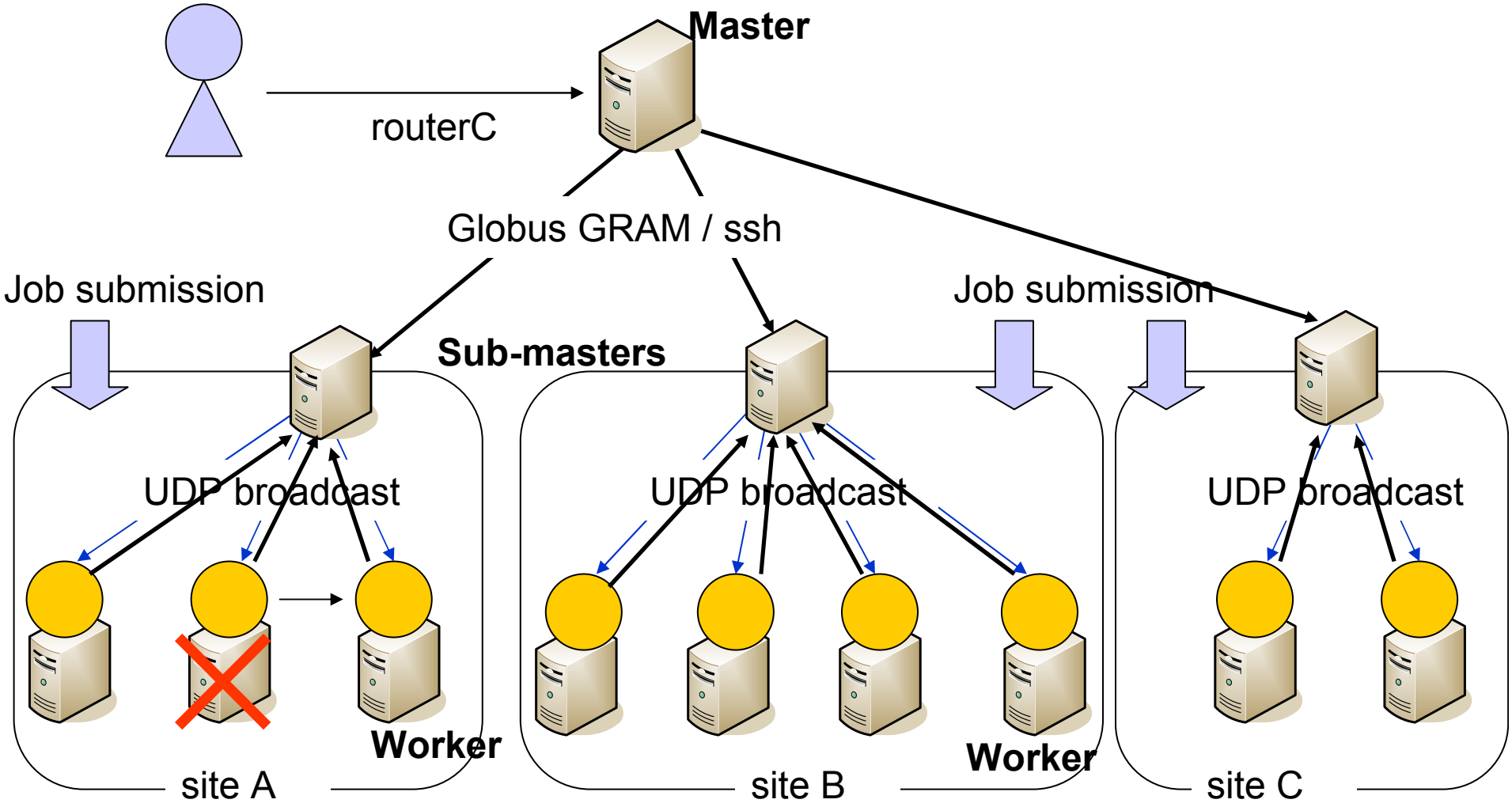
- UDP based automatic / dynamic tree configuration
  - Robust
  - Easy to configure
- Pure Java
  - To cope with CPU / OS heterogeneity
  - Automatic user program shipping
    - Ease the burden of setting up before execution
    - Avoid version mismatch error
- Simple yet powerful API

Implementation: Autonomous tree construction

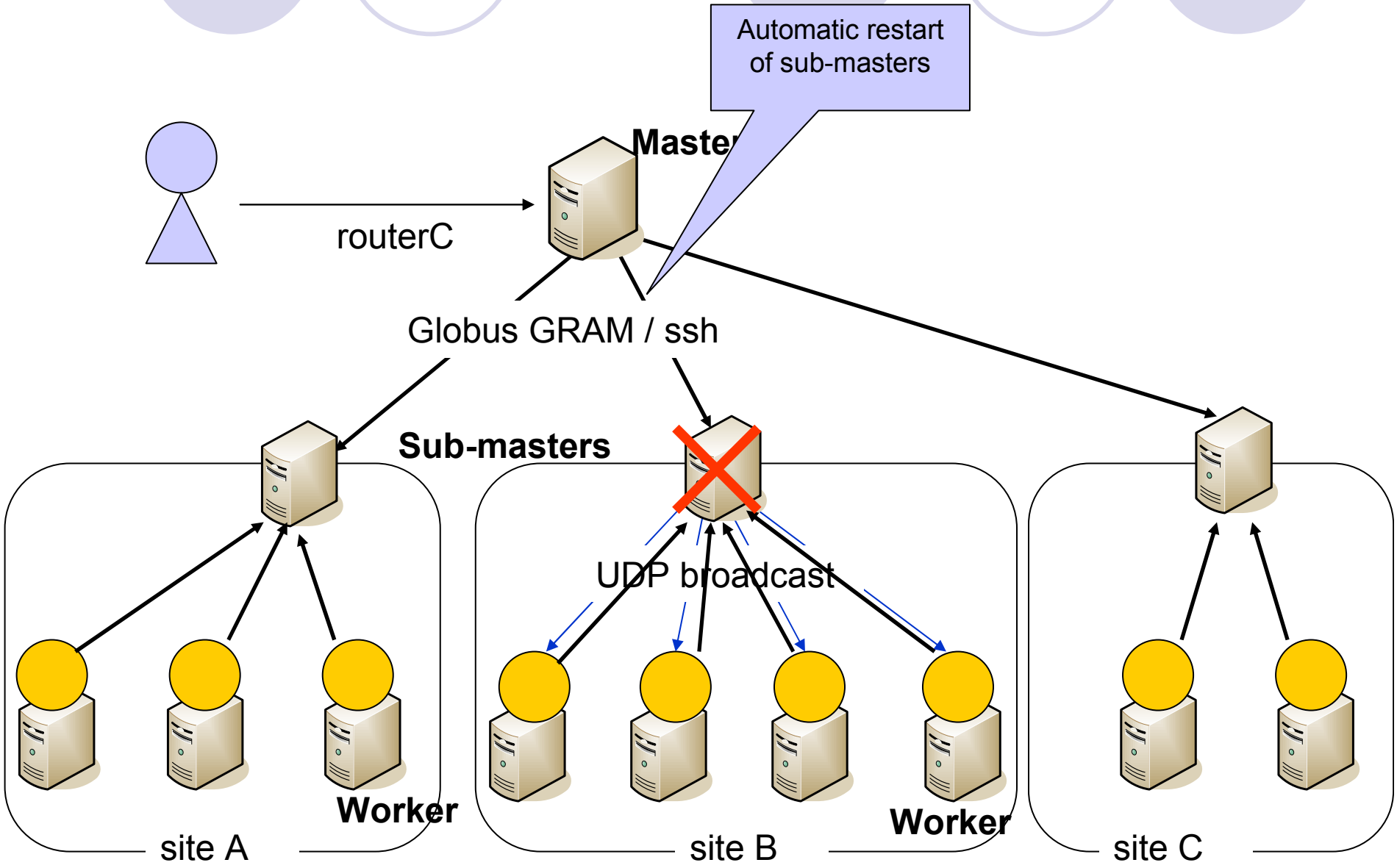


⇒ **Automatic tree construction that fits the physical configuration**

Implementation: Dynamic Node join/leave



Implementation: Fault Tolerant



Requirements on API

- Flexibility
  - Have to be flexible enough to implement several styles of programming.
- Robustness
  - The program have to aware of join / leaving of nodes

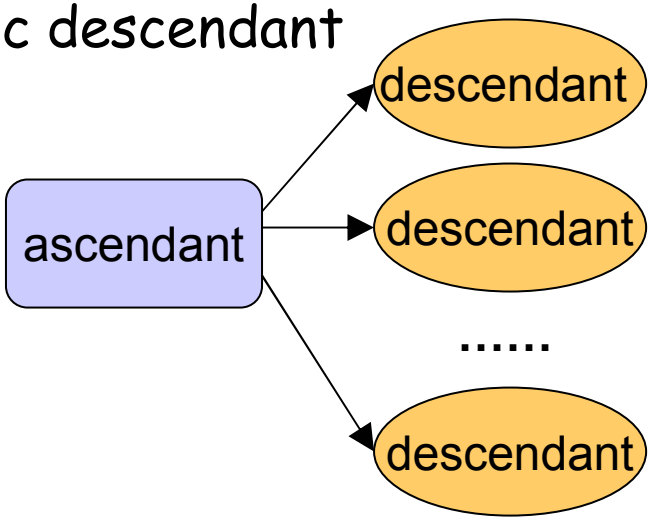
Message Passing Design (1)

- Adaptation to dynamic node addition/ removal
  - × message passing with target node ID

  Difficult to manage added / removed nodes' ID

⇒ Broadcast based

- to descendants, broadcast only
  - No method to talk with one specific descendant
- With ascendant, uni-cast



Message Passing Design (2)

- The program have to aware of joining / leaving nodes
  - Re-distribution of jobs

⇒Provides methods for handling with joining / leaving nodes
  - Invoked on join/leave of nodes
  - User have to write handling methods on the events
    - Application dependent

API Implementation

- 'Code' abstract class
  - Stands for nodes in the system
  - Programmers have to provide each layer
    - Master, Sub-master, Worker
  - Supporting classes are also provided
    - ParentNode
    - DescendantNodes

API (1): Code

```
public abstract class Code {
    ParentNode parent; /* the Ascendant */
    DescendantNodes descendants; /* the Descendant */

    /* initialization method: will be called on start */
    public void start();

    /* message handling methods */
    public void handleReceiveParent(Message msg);
    public Object handleReceiveDescendants(Message msg);

    /* handling methods on descendant node join/leave */
    public void handleAddDescendant(int id);
    public void handleDeleteDescendant(int id);
}
```

API (2): ParentNode, DescendantNodes

```
public class ParentNode { // Parent
    /* Send Only */
    public void send(Message msg);
    /* Blocking call */
    public Object call(Message msg);
    /* Non-blocking call with future*/
    public Future callFuture(Message msg);
    /* Non-blocking call with Context */
    public void callWithContext(Message msg, Context context);
}

public class DescendantNodes { // Child
    /* Broadcast to children */
    public void broadcast(Message msg);
    /* Returns number of descendants */
    public int size();
}
```

Several types of Message passing Is supported

Broadcast only

A Sample Program (Worker)

```
public class PiWorker extends Code {
  static final int MSG_TRIAL_REQUEST = 1;
  Random random = new Random();
  public void start() {
    long doneTimes = 0, trialTimes;
    while(true) {
      Message msg =
        new Message(MSG_TRIAL_REQUEST, doneTimes);
      trialTimes = (Long)parent.call(msg);
      if (trialTimes == 0) break;
      doneTimes = trial(trialTimes);
    }
   }
  /** give a trial */
  private long trial(long trialTimes) { ... }
}
```

A Sample Program (Master)

```
public class PiMaster extends Code {
  public synchronized Object handleReceiveDescendant(Message msg) {
    if (jobMap.containsKey(msg.nodeID)) {
      doneTrial += jobMap.remove(msg.nodeID);
      doneResult += (Long)(msg.contents);
    }
    while (jobQueue.isEmpty())
          try {wait();}catch(InterruptedException e) {}
    long perNode = jobQueue.remove();
    jobMap.put(msg.nodeID, perNode);
    return perNode;
  }

  public synchronized void handleDeleteDescendantNode(int nodeID) {
    long perNode = jobMap.remove(nodeID);
    jobQueue.add(perNode);
    notifyAll();
  }
}
```

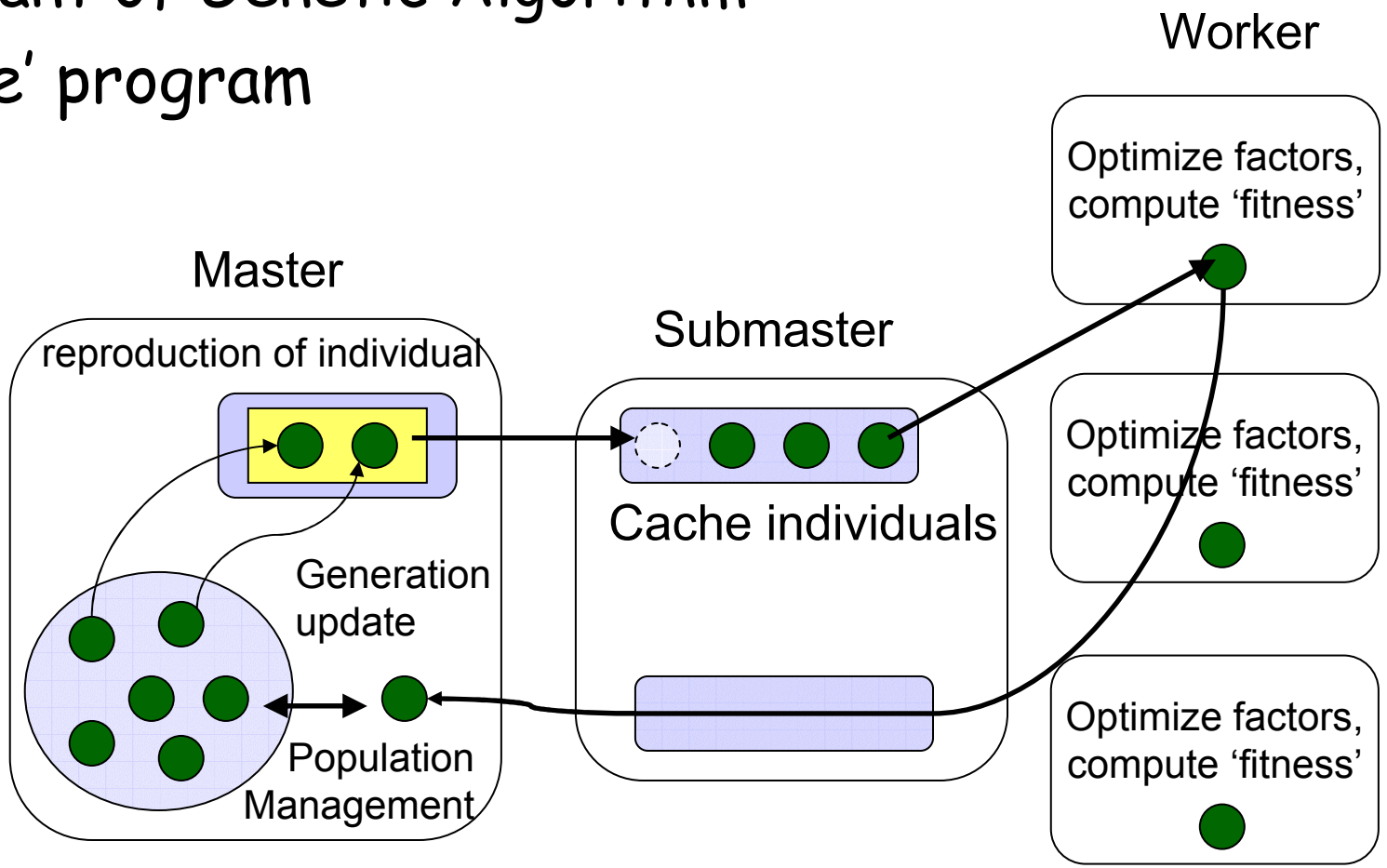
Evaluation

- Evaluate
  - Scalability
  - Robustness
- Target application
  - Genetic programming for genetic network inference
  - Environment
    - TSUBAME Grid Cluster

CPU	Opteron 2.4GHz
RAM	32 GB
Network	inifiniBand Voltaire ISR9288
OS	Linux 2.6.5
Java	JDK 1.5.0_06

Genetic network inference with Genetic Programming

- Genetic Programming
  - A variant of Genetic Algorithm
  - 'evolve' program

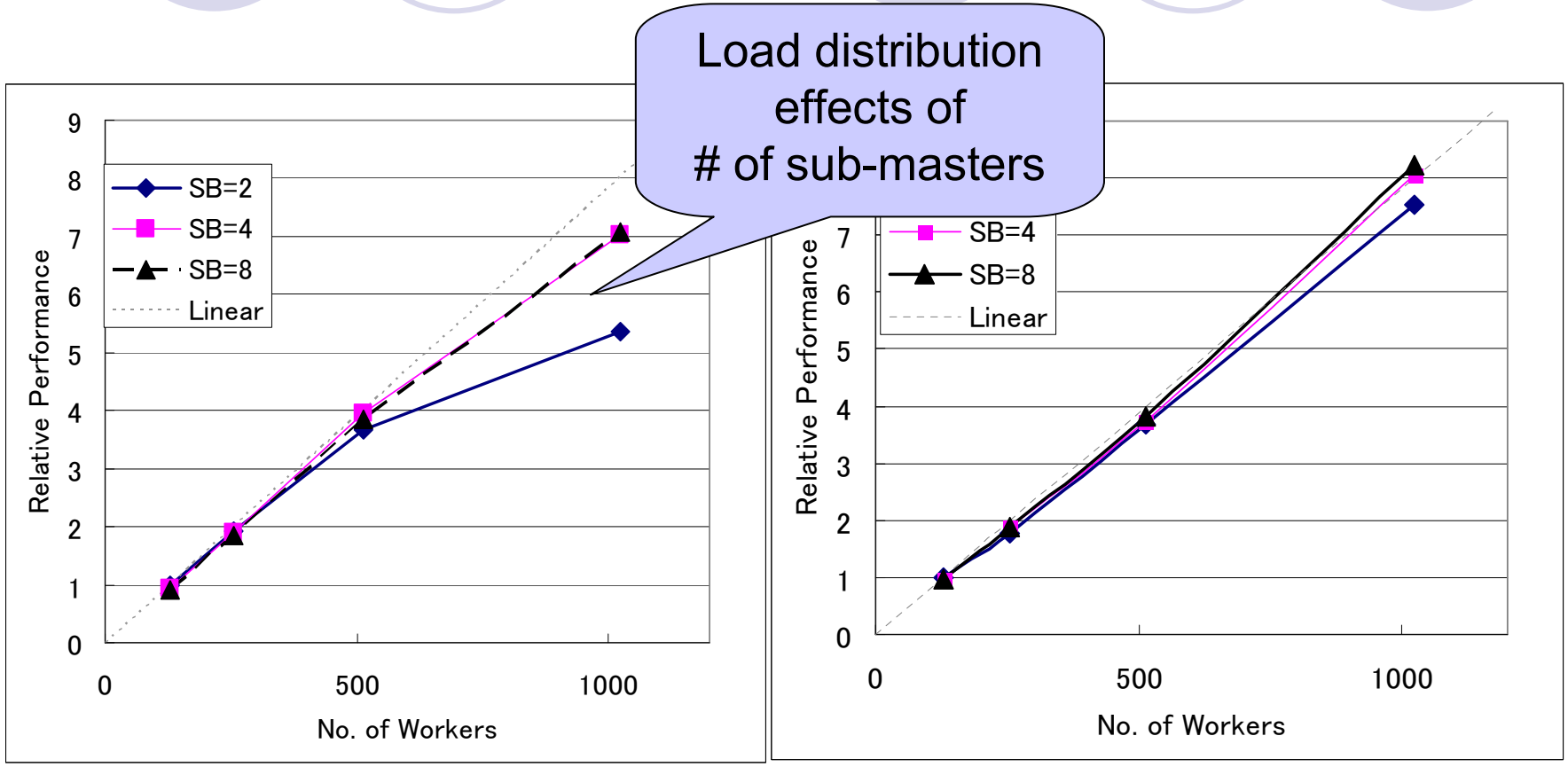


Scalability Evaluation

- Changed No. of sub-masters and workers
- Parameter RK: Runge-Kutta step size
  - Parameter used by fitness calculation on workers
  - Affects on processing time for each task on workers

RK	Proc. time [ms]
2E-2	1086
1E-2	2157

Results of scalability evaluation



RK=2E-2
Proc. time: 1086 [ms]

RK=1E-2
Proc. time: 2157 [ms]

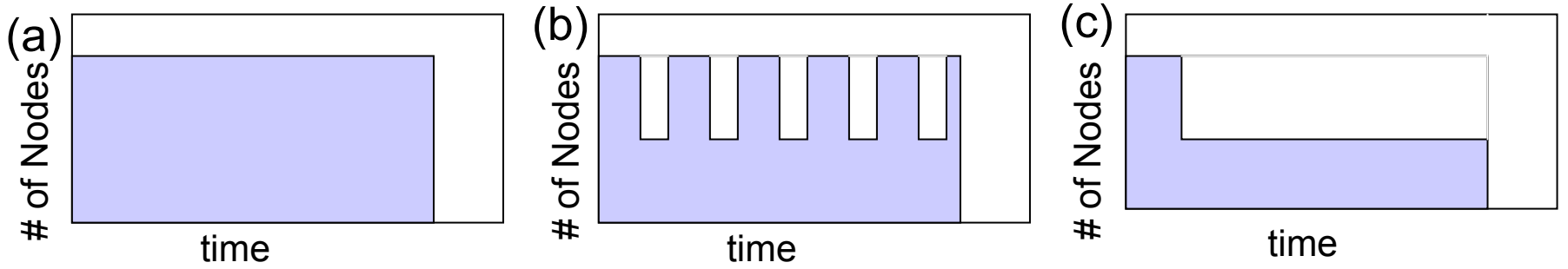
Robustness

- 3 patterns of disturbance
  - On start up: sub-masters: 4, workers: 256

(a) No disturbance

(b) Half of nodes down and comes back
  - Every 15 min. half of nodes dies (5 times in total)
  - 5min. Later, they comes back

(c) Half of nodes down
  - 15 min. after the start time



Robustness Result

	(a)	(b)	(c)
Time spent [sec.]	5937	6768	11165
Total CPU x sec.	1519872	1540608	1544320
Relative Efficiency	1.0	0.98	0.98

(b) Average time for a newly joind node to get task assignment: 18.4 sec

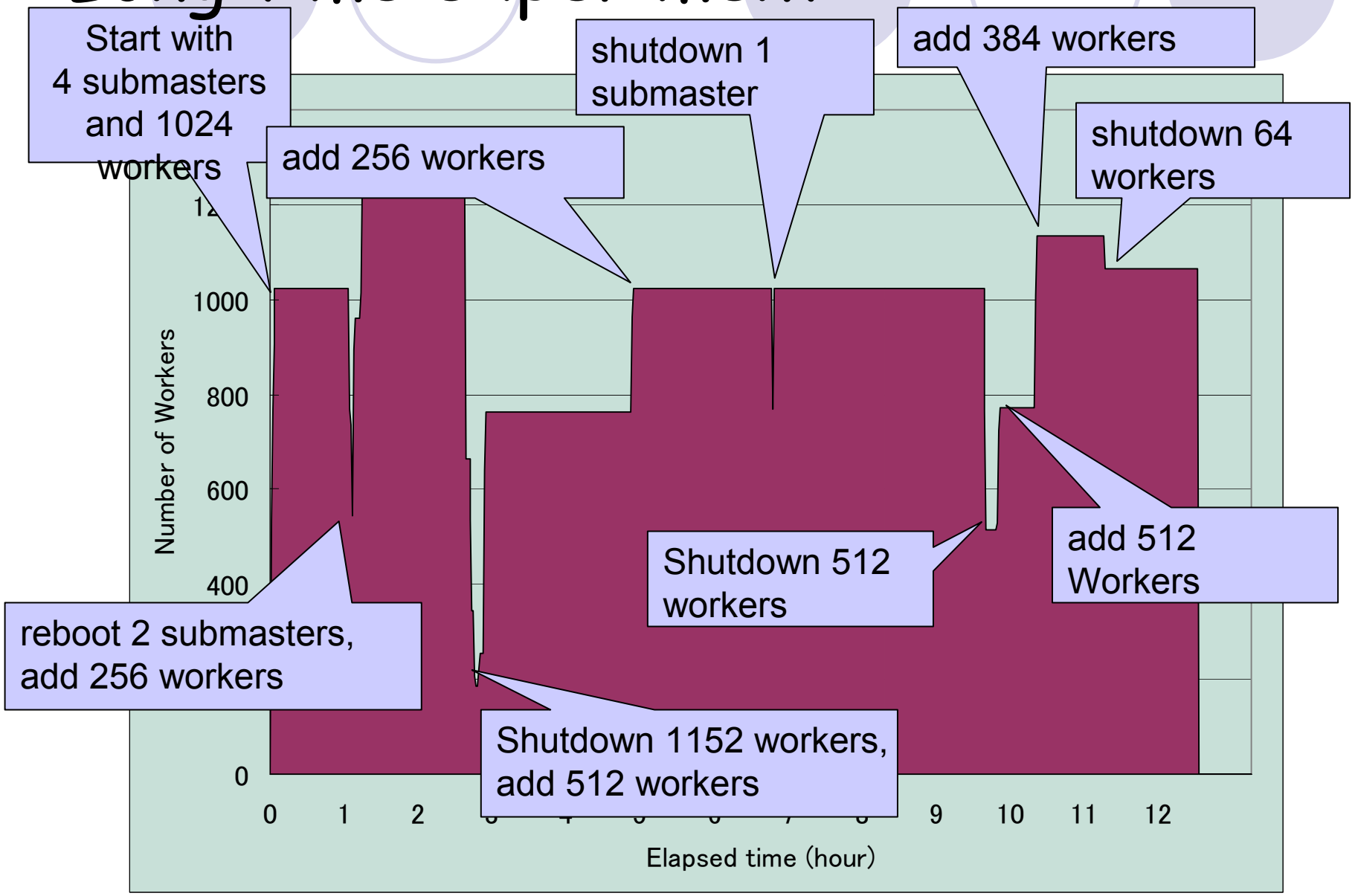
- UDP packet waiting: 17.6 sec.
- Job assignment : 0.8 sec.

  ⇒ Overhead for node join is small

Long time experiment

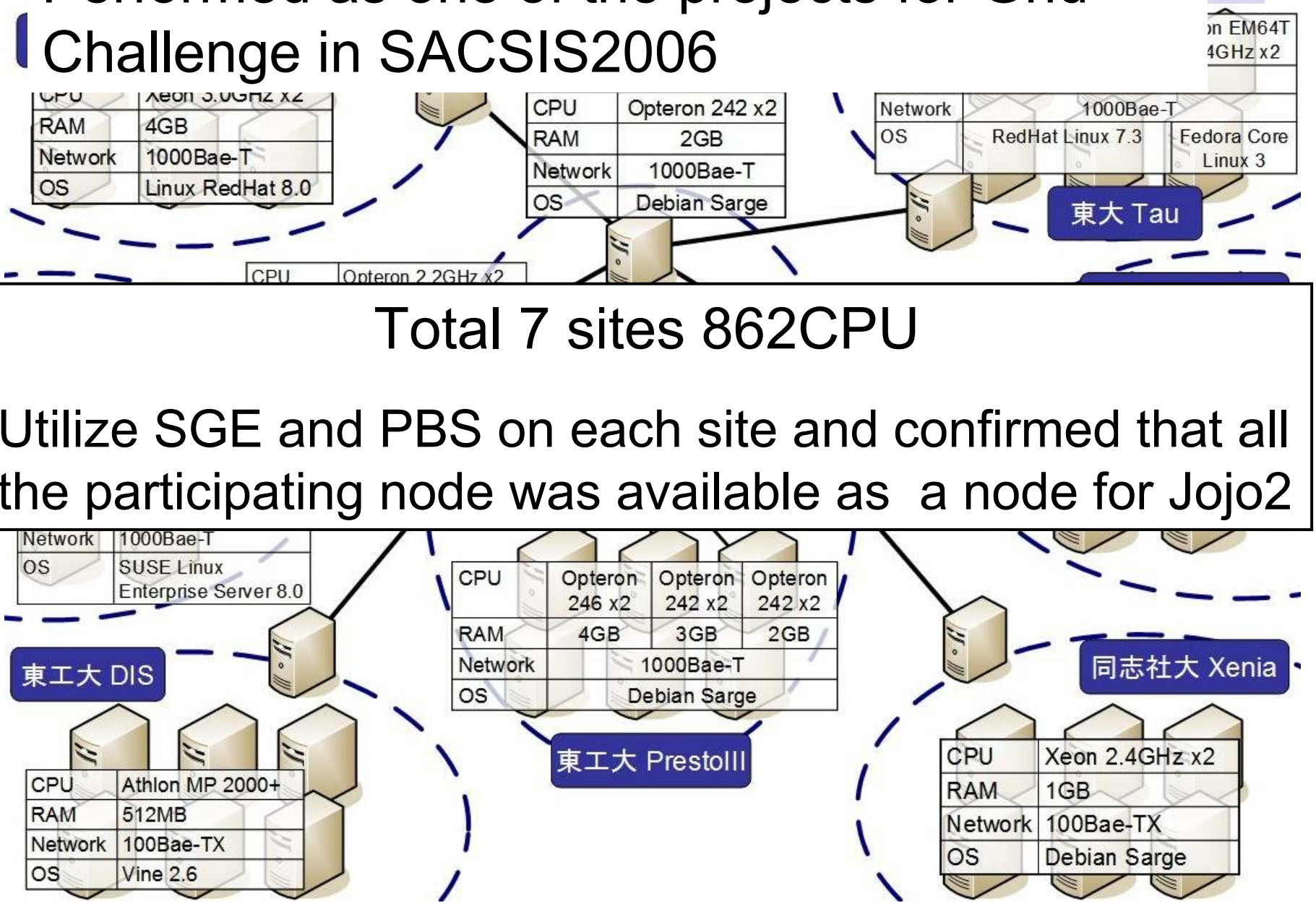
- Confirmed that Jojo2 is stable and robust enough to survive long-time experiment.
  - In the environment where nodes come and leave
- During the experiment, randomly start and stop sub-masters and workers

Long time experiment



Experiment in a Wide Area Grid

Performed as one of the projects for Grid Challenge in SACSIS2006



Total 7 sites 862CPU

Utilize SGE and PBS on each site and confirmed that all the participating node was available as a node for Jojo2

Related work

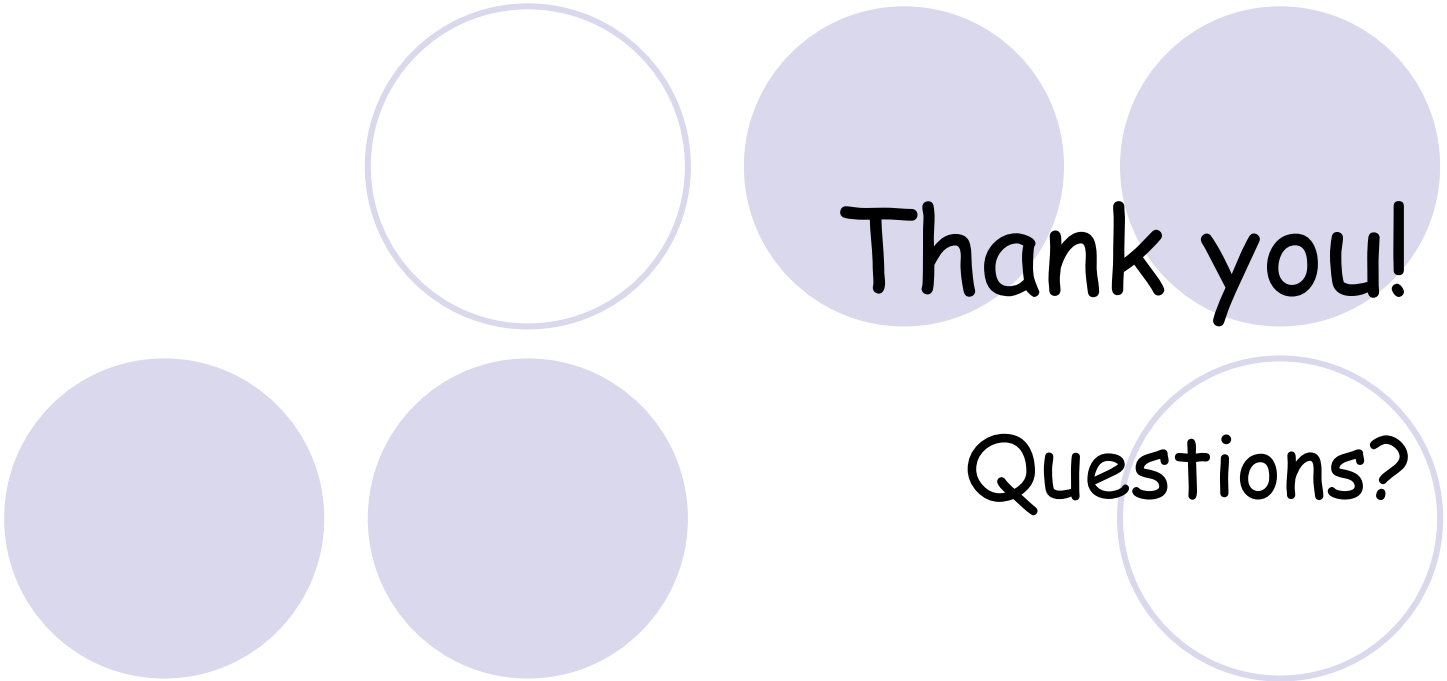
- Jojo
  - Our previous work
  - Not robust
- Cascaded GridRPC [Aida '05]
  - Not robust enough
- Phoenix [Taura '03]
  - Message passing interface, not MPI
  - Allows node increase / decrease

Conclusion

- Proposed Jojo2
  - Efficient Programming of fine-grained hierarchical master-worker applications
  - Allows node fault, intentional addition/removal
- Evaluation
  - Confirmed Robustness and Efficiency

Future Work

- Higher level API
  - For specific application areas
  - For specific algorithms
- Linking with other Languages
  - 'Serious' applications are mostly written in C++ or Fortran
  - To call them from Jojo2 will make sense



Thank you!

Questions?