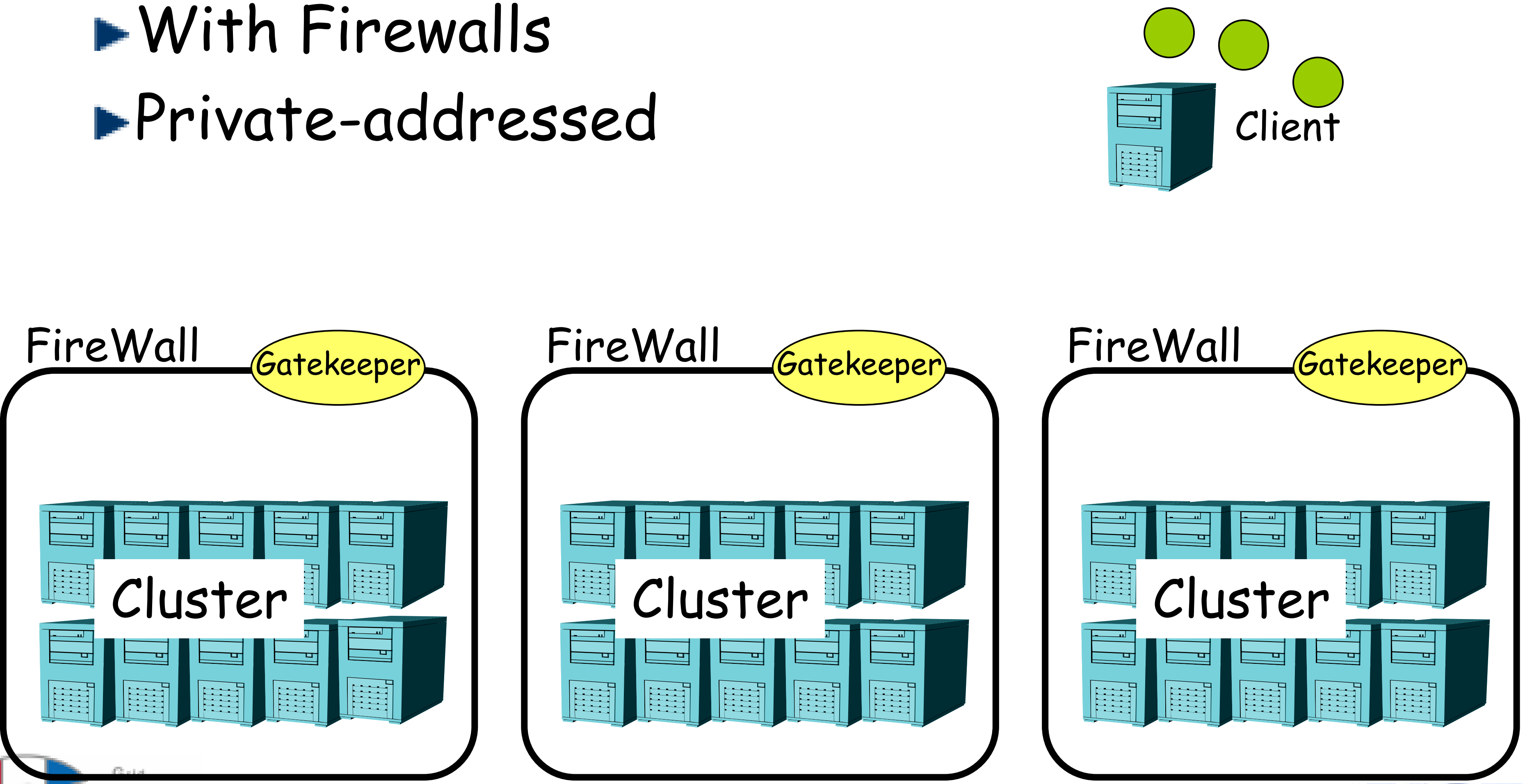
A Java-based Programming Environment for Hierarchical Grid: Jojo

Hidemoto Nakada (AIST / Tokyo-tech)
Satoshi Matsuoka (Tokyo-tech / NII)
Satoshi Sekiguchi (AIST)

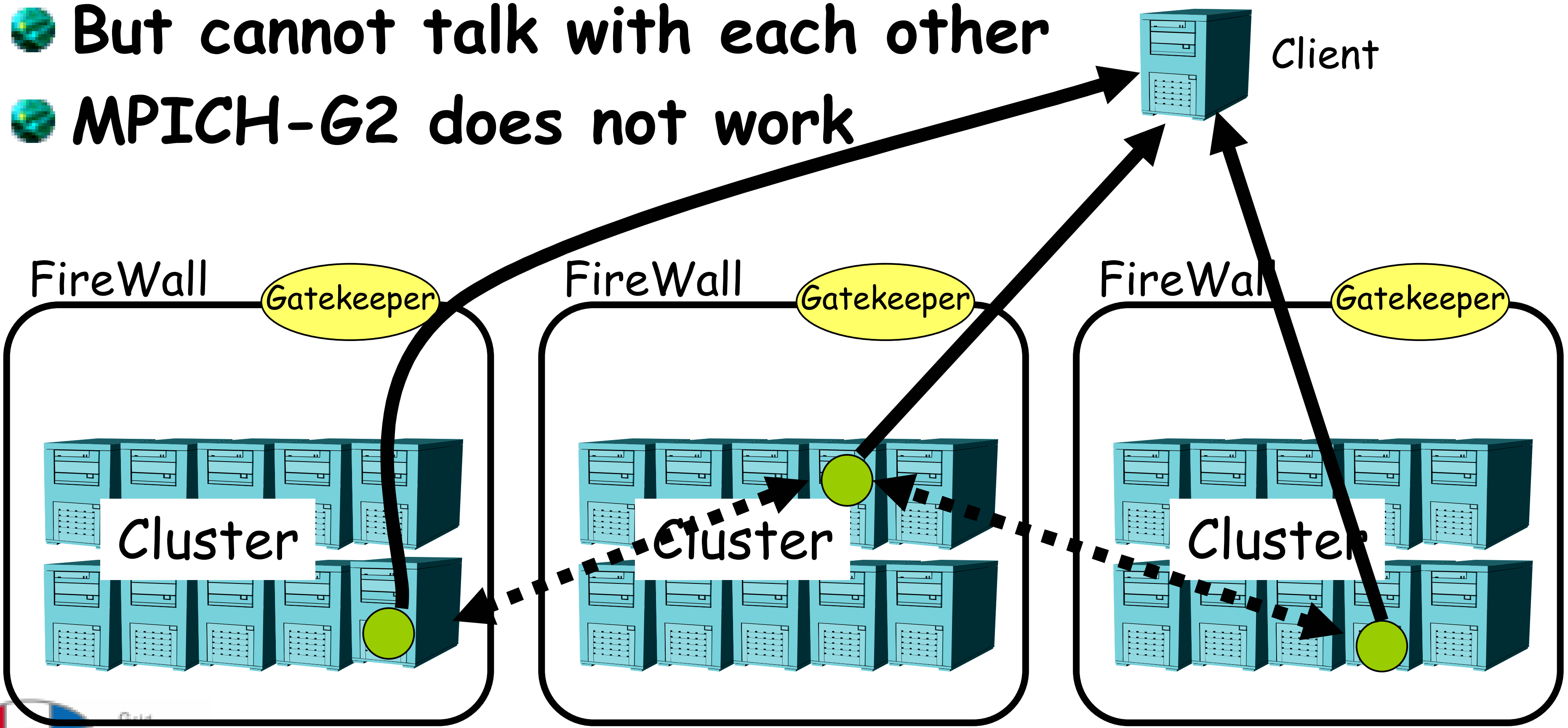
The Grid, Today

- **Cluster of Clusters**
  - With Firewalls
  - Private-addressed



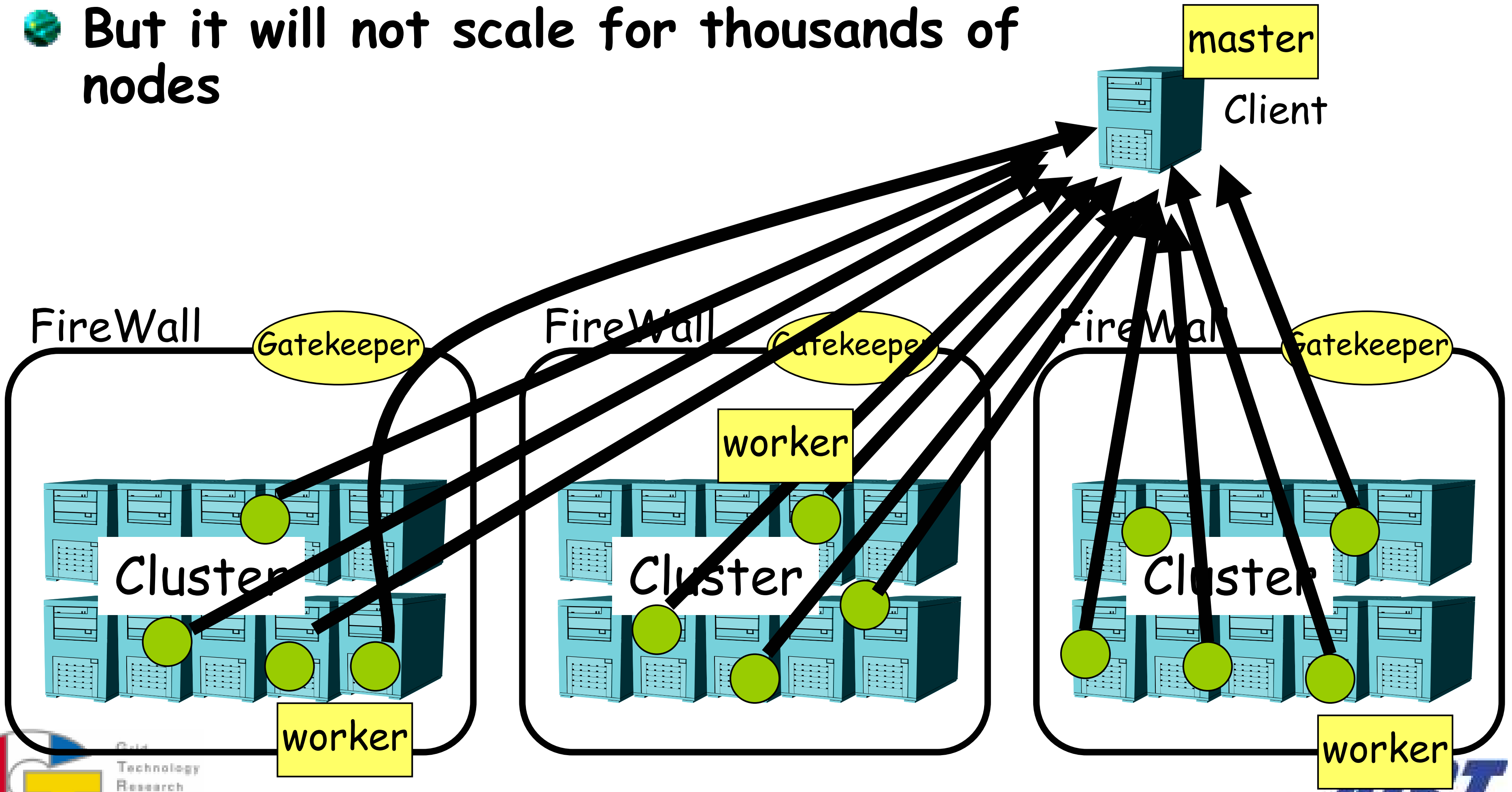
The Grid, Today

- **The running job can talk with the client, thanks to NAT**
- **But cannot talk with each other**
- **MPICH-G2 does not work**



The Grid, Today

- **Master-worker style program can utilize the Cluster**
- **But it will not scale for thousands of nodes**



Problems, in summary

- **MPICH-G2 will not work well for private addressed clusters**
- **Master-worker style will not scale for thousands of nodes.**

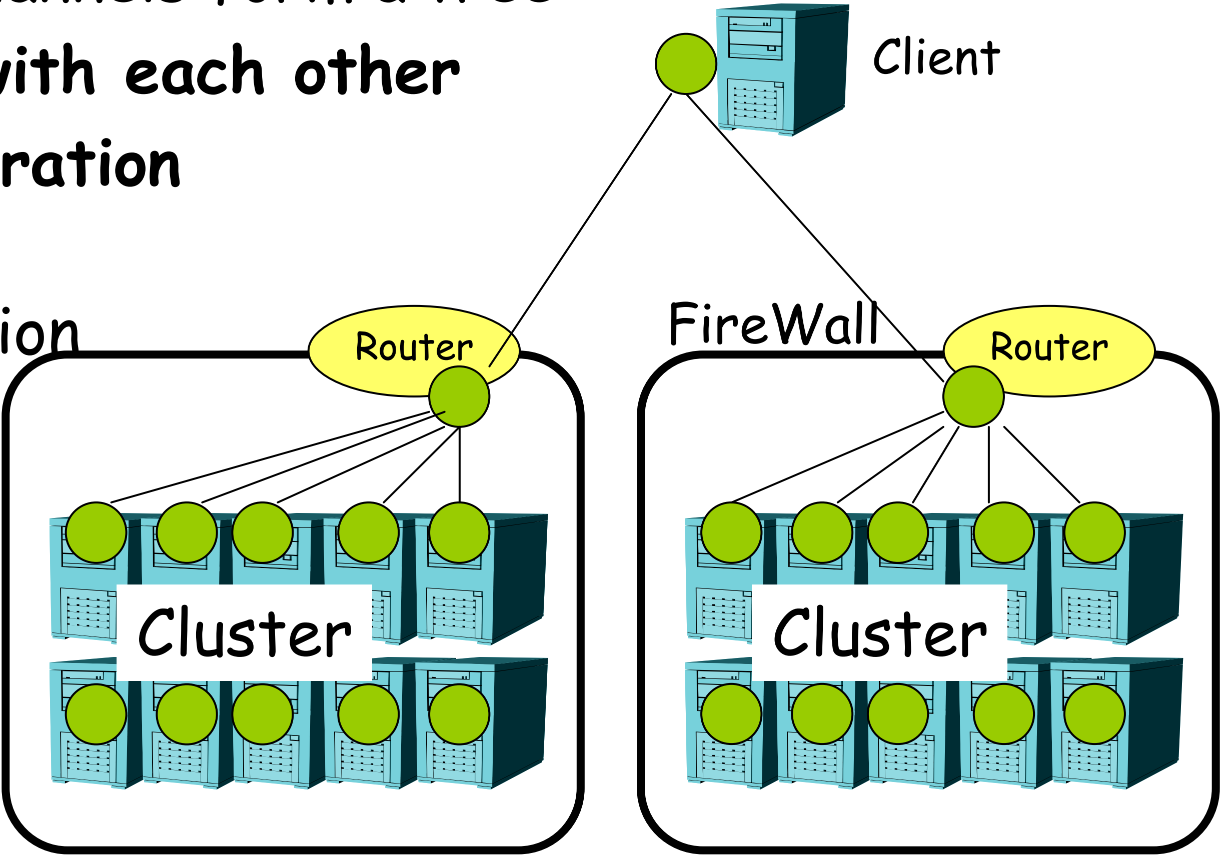
- **Another problem: Installation cost**
  - Installing user application on several clusters can be a huge burden for the users
  - They also have to install the middleware

The Goal

- **Provide a programming environment that**
  - Works with private addressed clusters
  - Scales to thousands of nodes
  - Ease the burden of installation

Key Idea: Hierarchical Grids

- **Have jobs also on the Firewall**
  - Works as application level routers
  - Communication channels form a tree
- **Each job can talk with each other**
- **No massive concentration**
  - Take advantage of the configuration



Jojo: a middleware for Hierarchical Grids

- **All the system is started up from the Client, recursively**
  - Forms communication channel tree
  - Protocol :Globus GRAM、ssh/rsh
- **Java, as the target program**
- **All the programs are dynamically loaded from the client**

Jojo is Java based, because

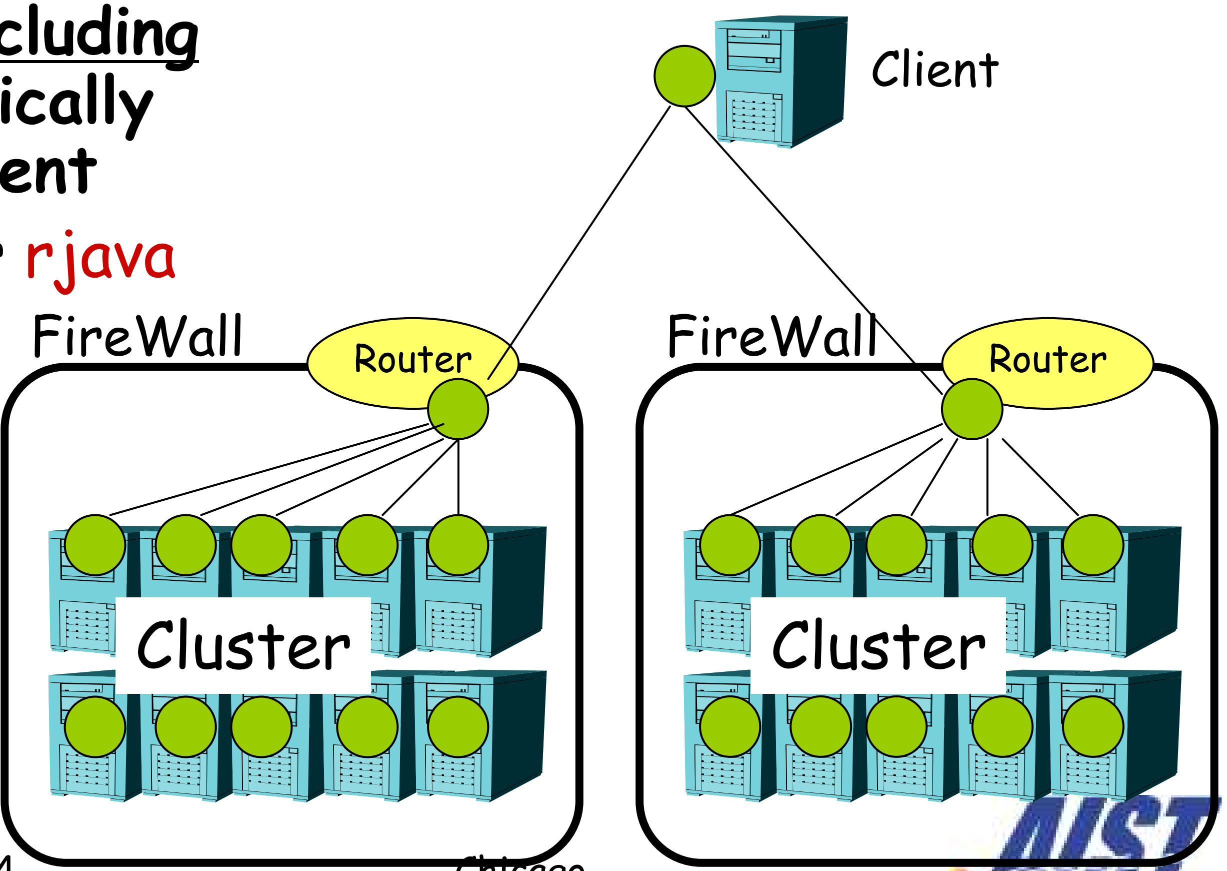
- **Code portability**
  - Good for heterogeneous environment
- **Integrated Thread support**
  - Good for latency hiding
- **Lot of libraries are available**
  - XML, Web related, network communication

Jojo ease the installation burden by

- **Automatically downloading the user programs, and Jojo system program itself.**
  - Avoids system version miss-match
  - Requires Java VM only on the cluster nodes

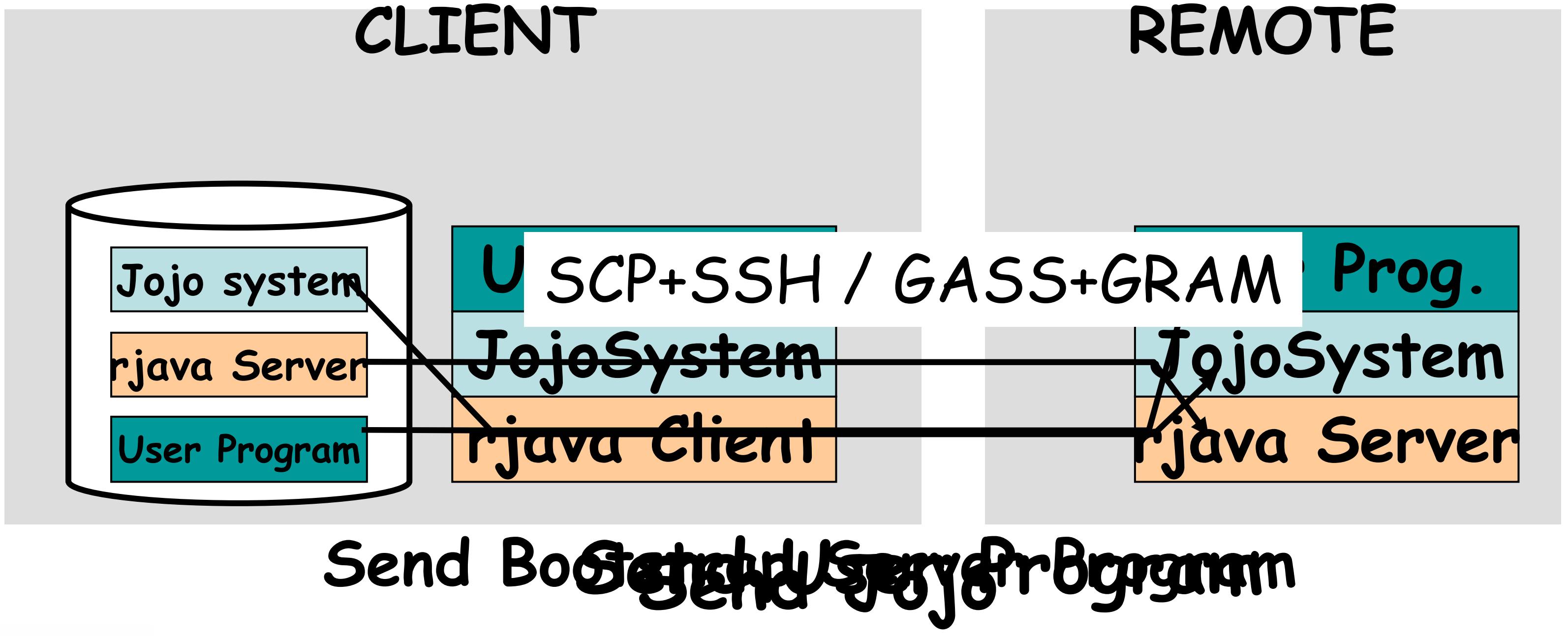
Starting up a Jojo Program

- **The client (the 1st level node) invokes 2nd level nodes, and the 2nd level nodes invokes 3rd level**
- **All the programs <u>including Jojo itself</u> automatically staged from the client**
  - Boot strap server rjava



Bootstrapping with *rjava*

- **First of all small rjava server core will be staged and executed**
  - It provides a customized code loader
- **All the class binaries are loaded from the Client, as needed, with the class loader**



Programming model of Jojo

- **On Each node a representative Java Class will run**
  - Subclass of the “Code” class
  - c.f. Applet
- **Object based messaging**
  - The Classes on the node will talk each other with passing Message
  - Incoming messages will be handled by separate handler method
    - To overlap communication and computation
  - RPC style call is supported
    - Several message transfer modes are supported

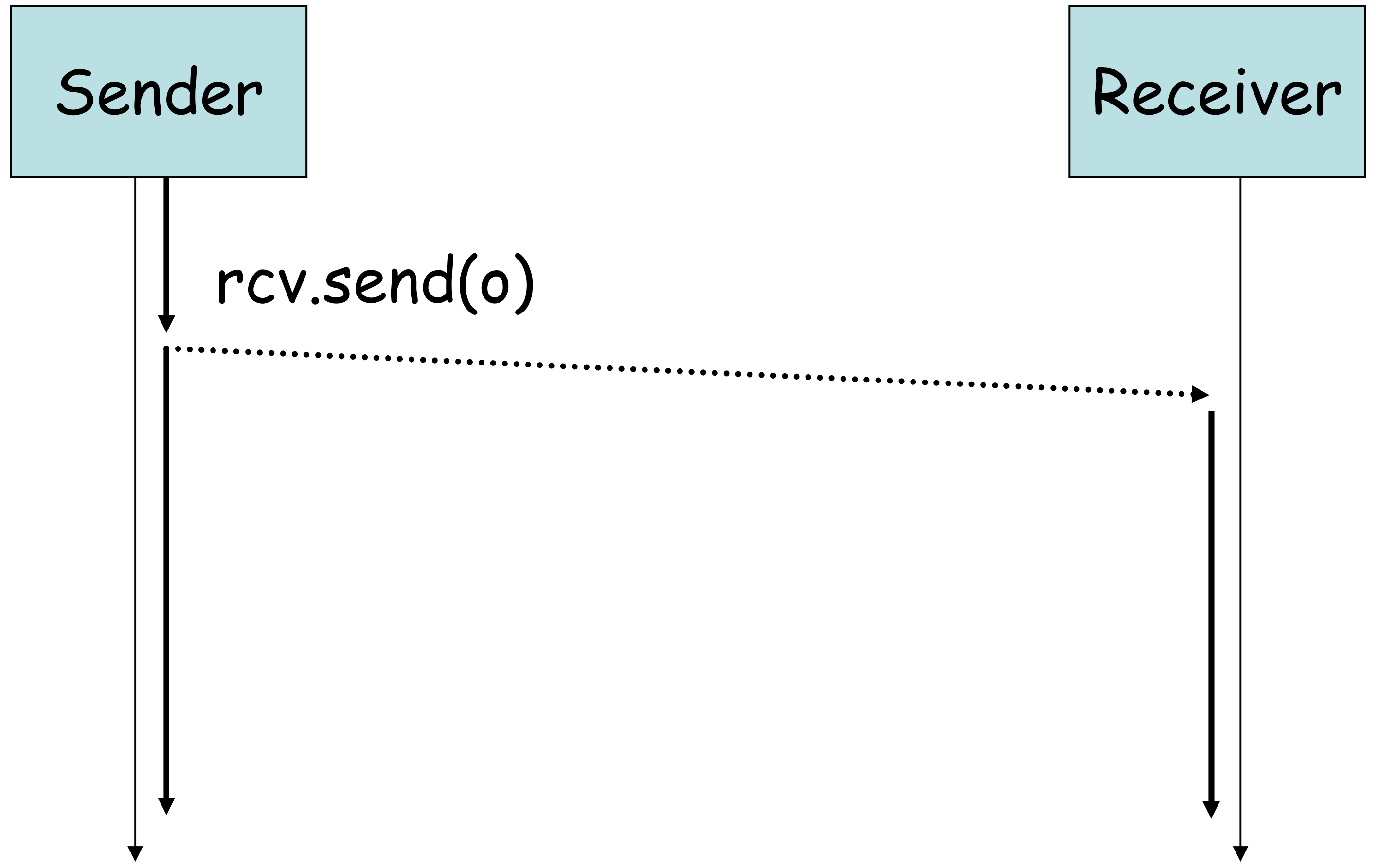
The “Code” class

```
abstract class Code{
  Node [] siblings;     /** Brothers  */
  Node [] descendants; /** children */
  Node     parent;       /** parent */
  int      rank;  /** order in the brothers */

  /** initialize */
  public void init(Map arg);

  /** actual task */
  public void start();

  /** handler to handle incoming messages */
  public Object handle(Message mes);
}
```

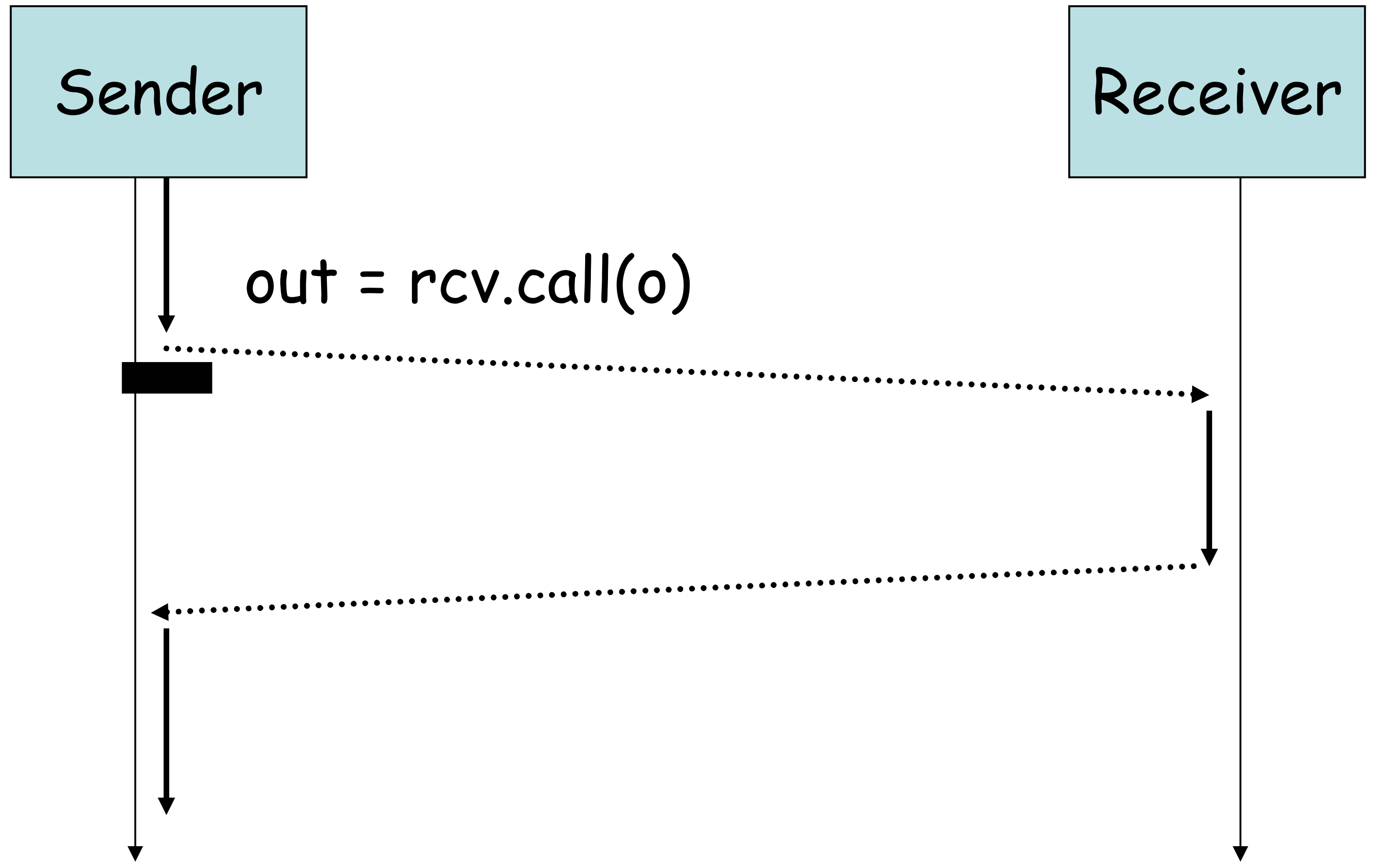
Programming model of Jojo

- On Each node a representative Java Class will run
  - Subclass of the "Code" class
  - c.f. Applet
- **Object based messaging**
  - The Classes on the node will talk each other with passing Message
  - Incoming messages will be handled by separate handler method
    - To overlap communication and computation
  - RPC style call is supported
    - Several message transfer modes are supported

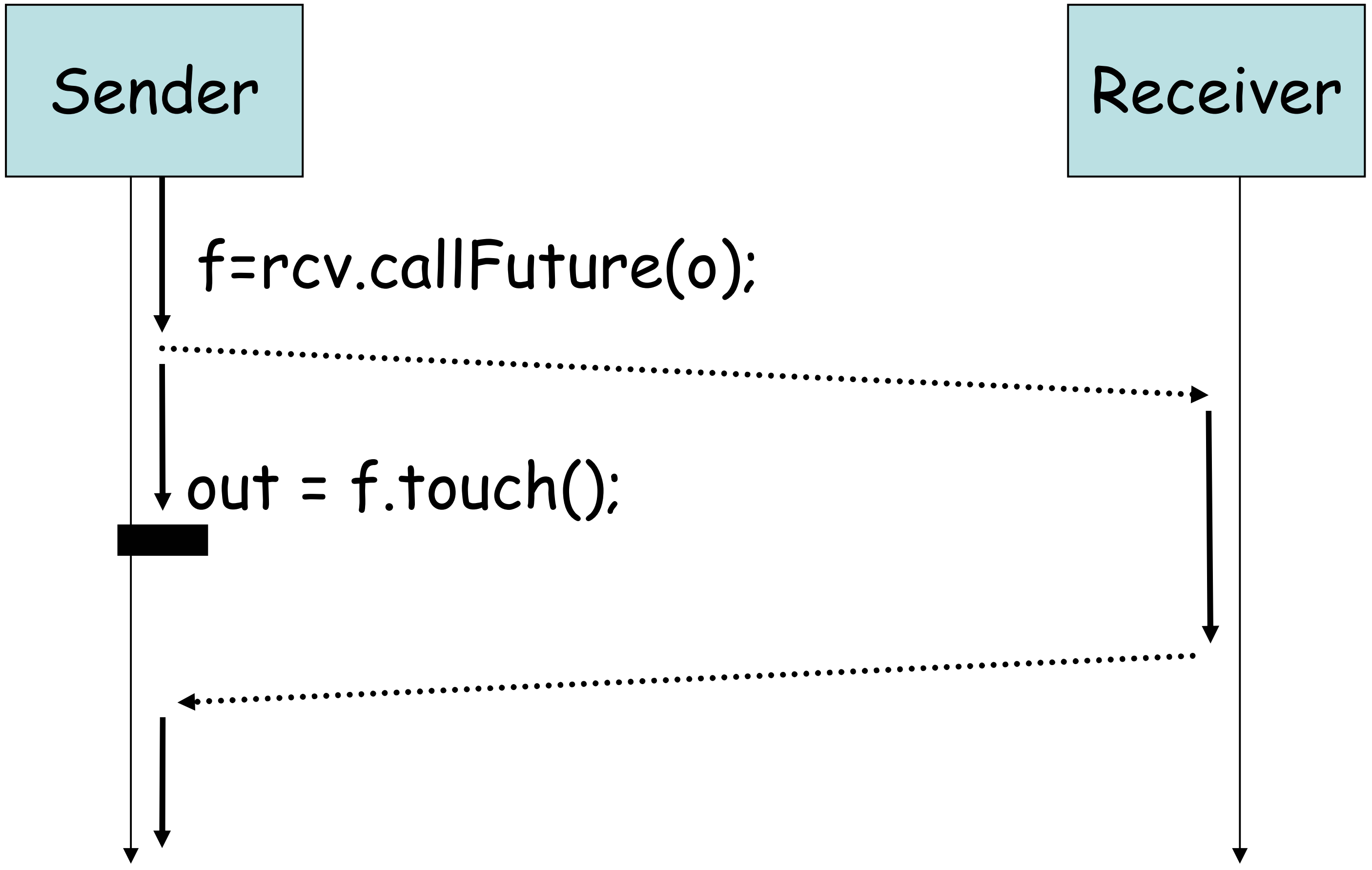
Transmission mode (1) send



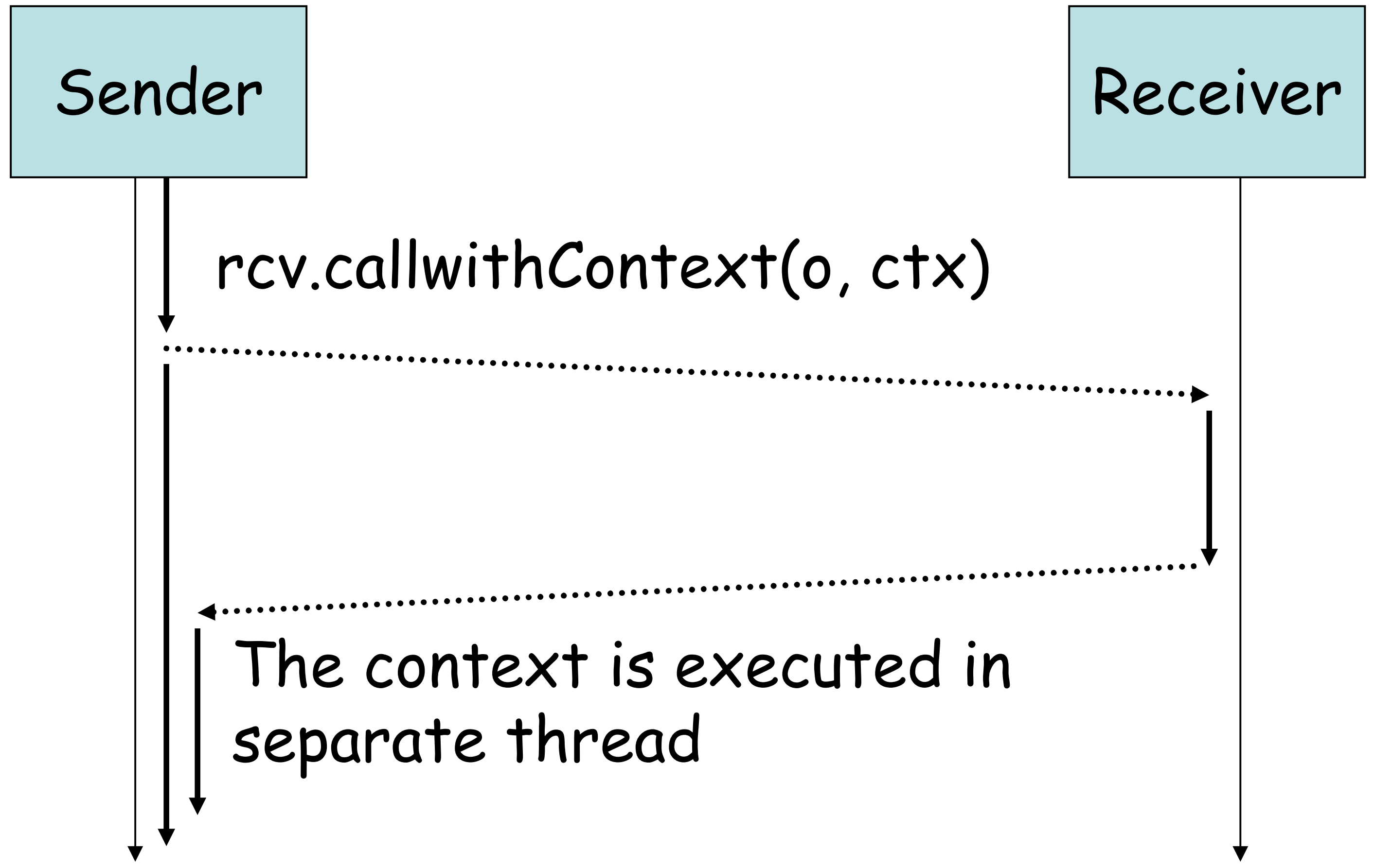
Transmission mode (2) blocking call



Transmission mode(3) Future



Transmission mode(4) with Context


Sender
Receiver
rcv.callwithContext(o, ctx)
The context is executed in separate thread

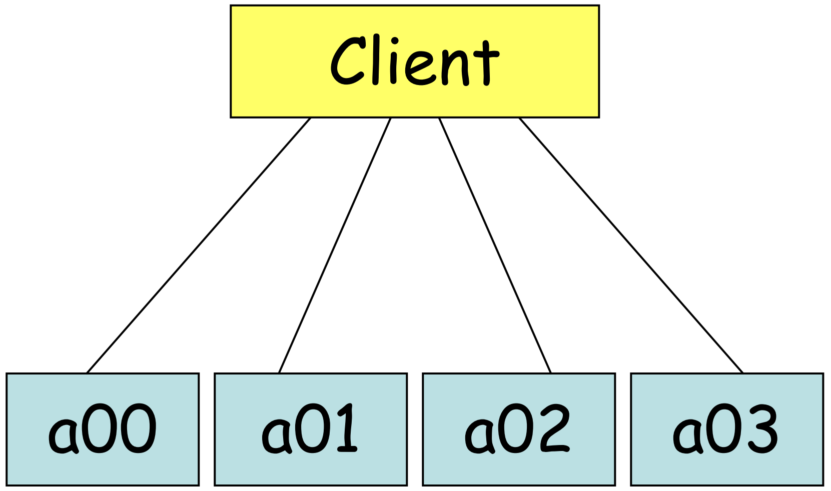
Configuration Files

- **To specify**
  - Nodes participates
  - Code to execute on each nodes
  - Invocation method to be used
- **Described in XML**
  - Represent hierarchical structure

```
<!ELEMENT node (code?,invocation?,node*)>
<!ATTLIST node host CDATA #REQUIRED>
<!ELEMENT code (#PCDATA)>
<!ELEMENT invocation EMPTY>
<!ATTLIST invocation
        javaPath CDATA #IMPLIED
        rjavaProtocol CDATA #IMPLIED
        rjavaRsh CDATA #IMPLIED
        rjavaRcp CDATA #IMPLIED
        xtermDisplay CDATA #IMPLIED
        xtermPath CDATA #IMPLIED
>
```

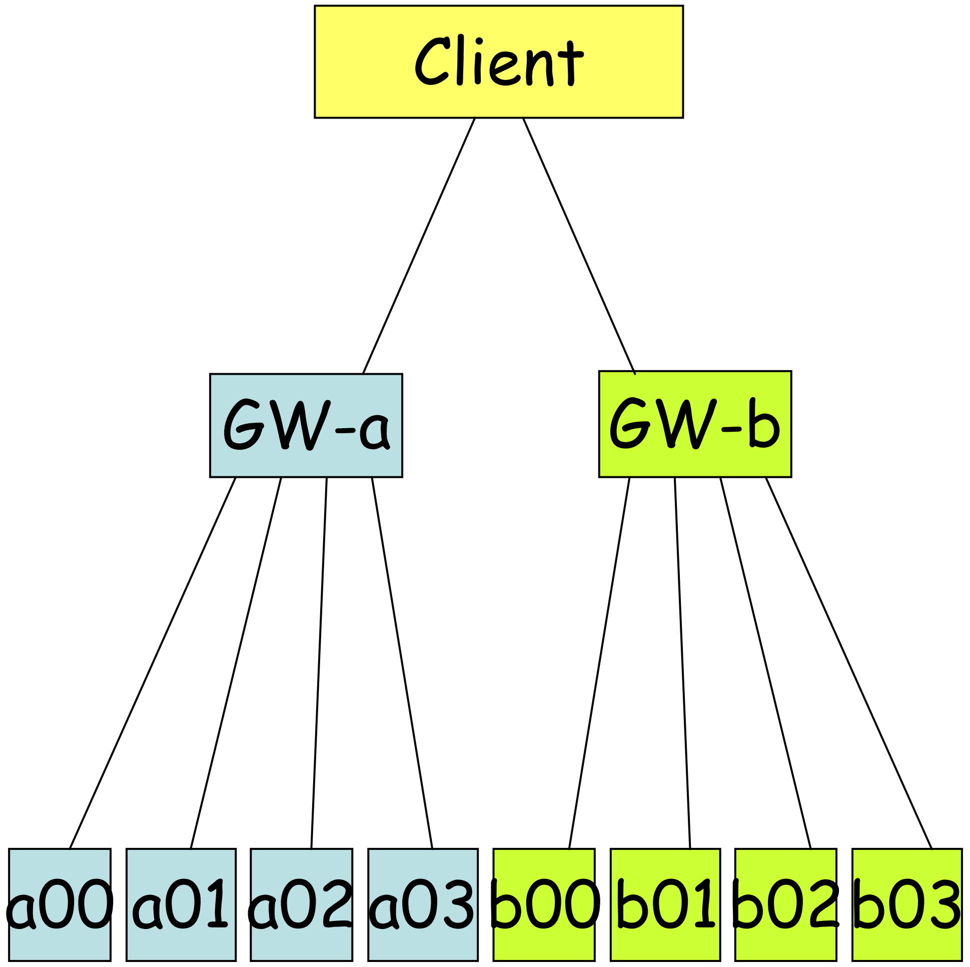
Sample configuration file

```
<node host="root">
 <code> PiMaster </code>
 <node host="default">
   <code> PiWorker </code>
   <invocation
    javaPath="java"
    rjavaJarPath="/tmp/rjava.jar"
    rjavaProtocol="ssh"
    rjavaRsh="ssh"
    rjavaRcp="scp"/>
  </node>
  <node host="a00"/>
  <node host="a01"/>
  <node host="a02"/>
  <node host="a03"/>
</node>
```



Sample configuration file, cont'd

```
<node host="root">
  <code> PiMaster </code>
  <node host="default">
    <code> PiInter </code>
    <invocation  OMITTED/>
  </node>
  <node host="GW-a">
    <node host="default">
      <code> PiWorker </code>
       <invocation OMITTED/>
    </node>
    <node host="a00"/>
    <node host="a01"/>
    <node host="a02"/>
    <node host="a03"/>
  </node>
  <node host="GW-b">
    <node host="default">
      <code> PiWorker </code>
      <invocation OMITTED/>
     </node>
     <node host="b00"/>
     <node host="b01"/>
     <node host="b02"/>
     <node host="b03"/>
  </node>
</node>
```



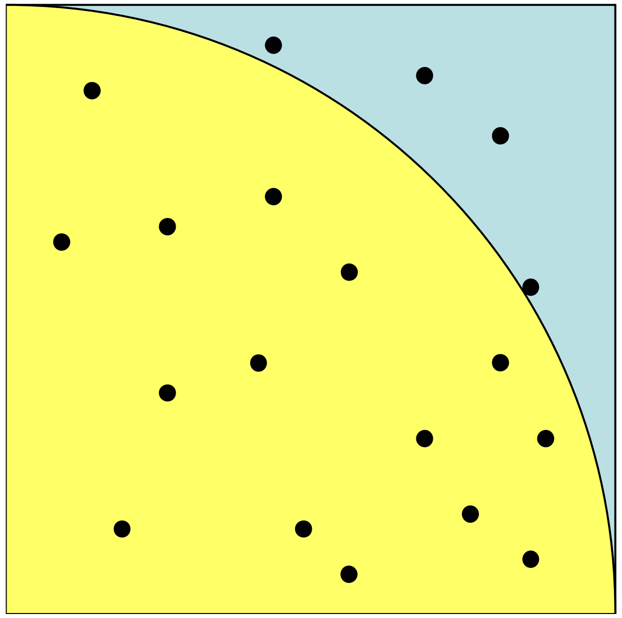
A Sample Program

- **Calculate PI with random points**
  - Randomly generates large number of points in a square
  - Count the number in the arc
  - Calculate PI from the probability

$$\text{PI} \approx 4 * \frac{\text{no. points in the quadrant}}{\text{no. of whole points}}$$

$$= 4 * \frac{15}{19}$$

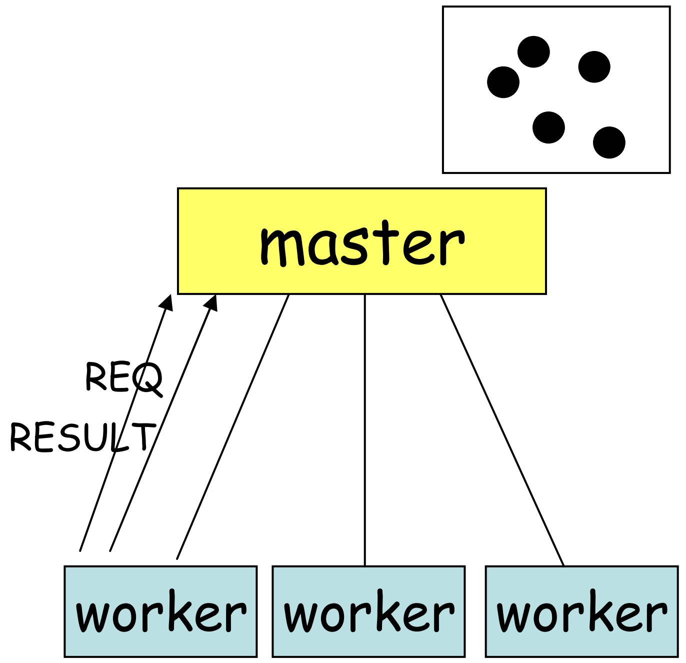
$$= 3.1579...$$



- **Master - Worker model**
  - Dynamic load balancing

A Sample Program (cont'd)

- **Self-scheduling for load balancing**
- **Worker**
  - Request the number of points to generate to the Master
  - Return the number with in the quadrant
- **Master**
  - On request, provide the number to the worker
  - Accumulate the number of points in the quadrant and whole.



Sample Program (Worker)

```
public class PiWorker2 extends Code{

  public void start() throws JojoException{
    long trialTimes = 0, doneTimes  = 0;
    while (true){
      Message msg =
        new Message(MSG_TRIAL_REQUEST,
                new long[]{trialTimes, doneTimes});
      trialTimes =
            ((Long)(parent.call(msg))).longValue();
      if (trialTimes == 0) break;
      doneTimes = trial(trialTimes);
    }
  }
```

Compose a message

Send the message and get the result

```
private long trial(long trialTimes){
  long counter = 0;
  for (long i = 0; i < trialTimes;i++){
    double x =
            random.nextDouble();
    double y =
            random.nextDouble();
    if (x * x + y * y < 1.0)
    counter++;
  }
  return counter;
 }
}
```

Sample Program (Master)

```
public class PiMaster2 extends Code{
  ……
  synchronized public Object handle(Message msg)
   throws JojoException{
    if (msg.tag == PiWorker.MSG_TRIAL_REQUEST){
      long [] pair = (long[])(msg.contents);
      doneTrial  += pair[0];
      doneResult += pair[1];
      if (doneTrial >= times){
          synchronized (this) {done = true; notifyAll();}
          return new Long(0);
      } else
          return new Long(perNode);
    } else
      throw new JojoException(
      "cannot handle the message: " + msg);
  }
}
```

Check the Message

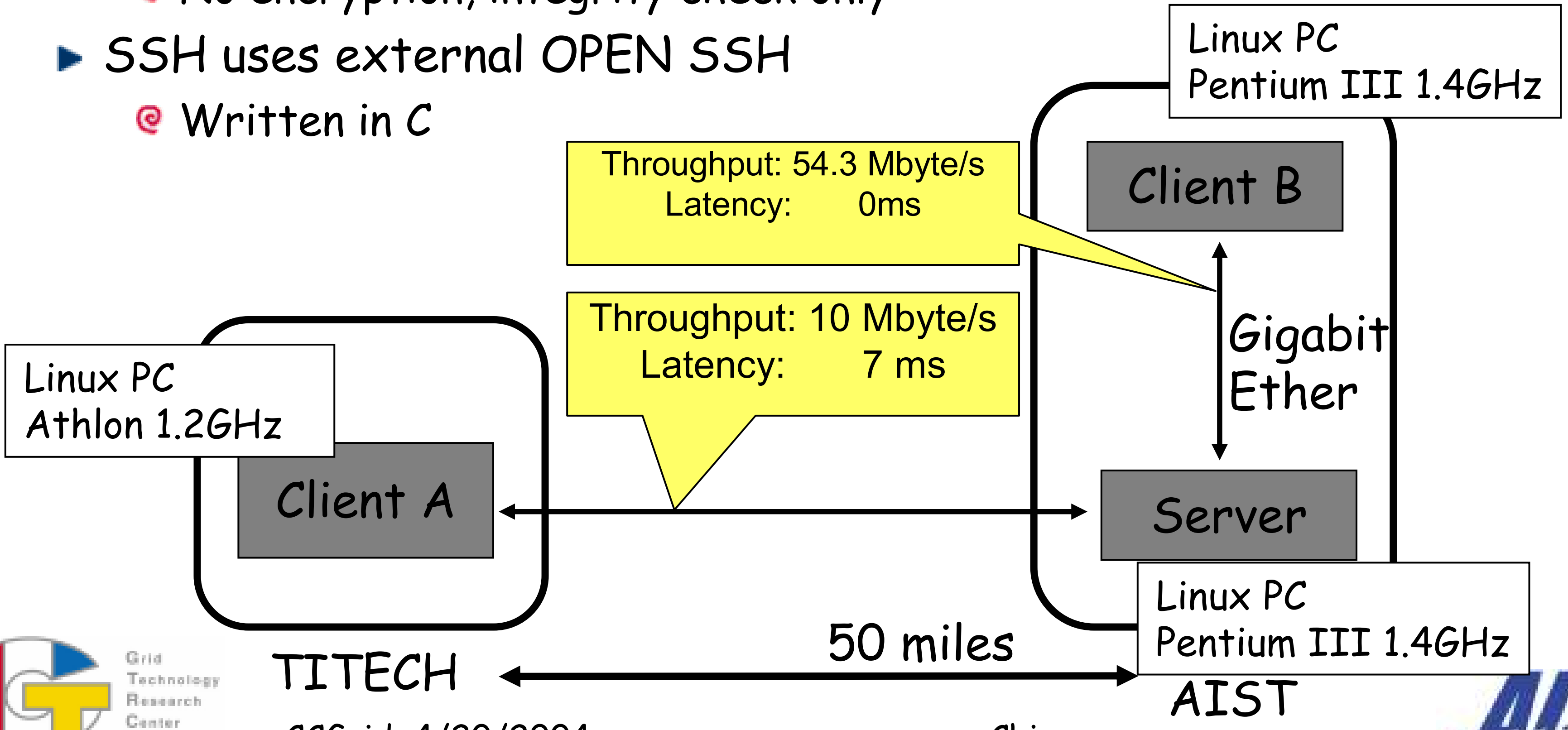
Return the number to try

Preliminary Evaluations

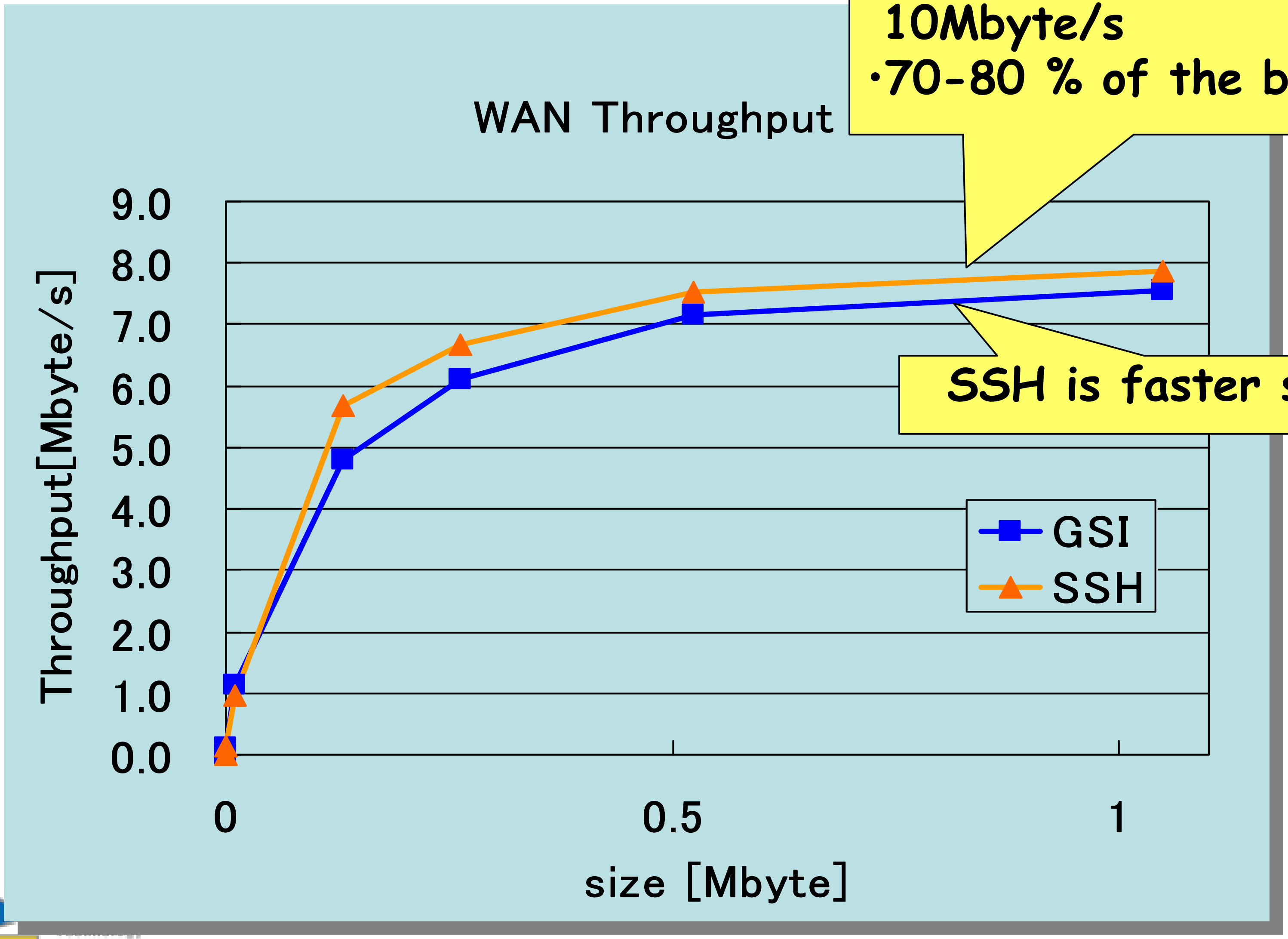
- **Throughput measurement**
  - WAN / LAN
  - GSI / SSH
- **Master-Worker program**
  - 2 Layered / 3 Layered

Throughput measurement

- **In LAN and WAN**
  - AIST and Titech
- **GSI(Globus I/O) and SSH**
  - GSI uses pure-Java SSL
    - No encryption, integrity check only
  - SSH uses external OPEN SSH
    - Written in C



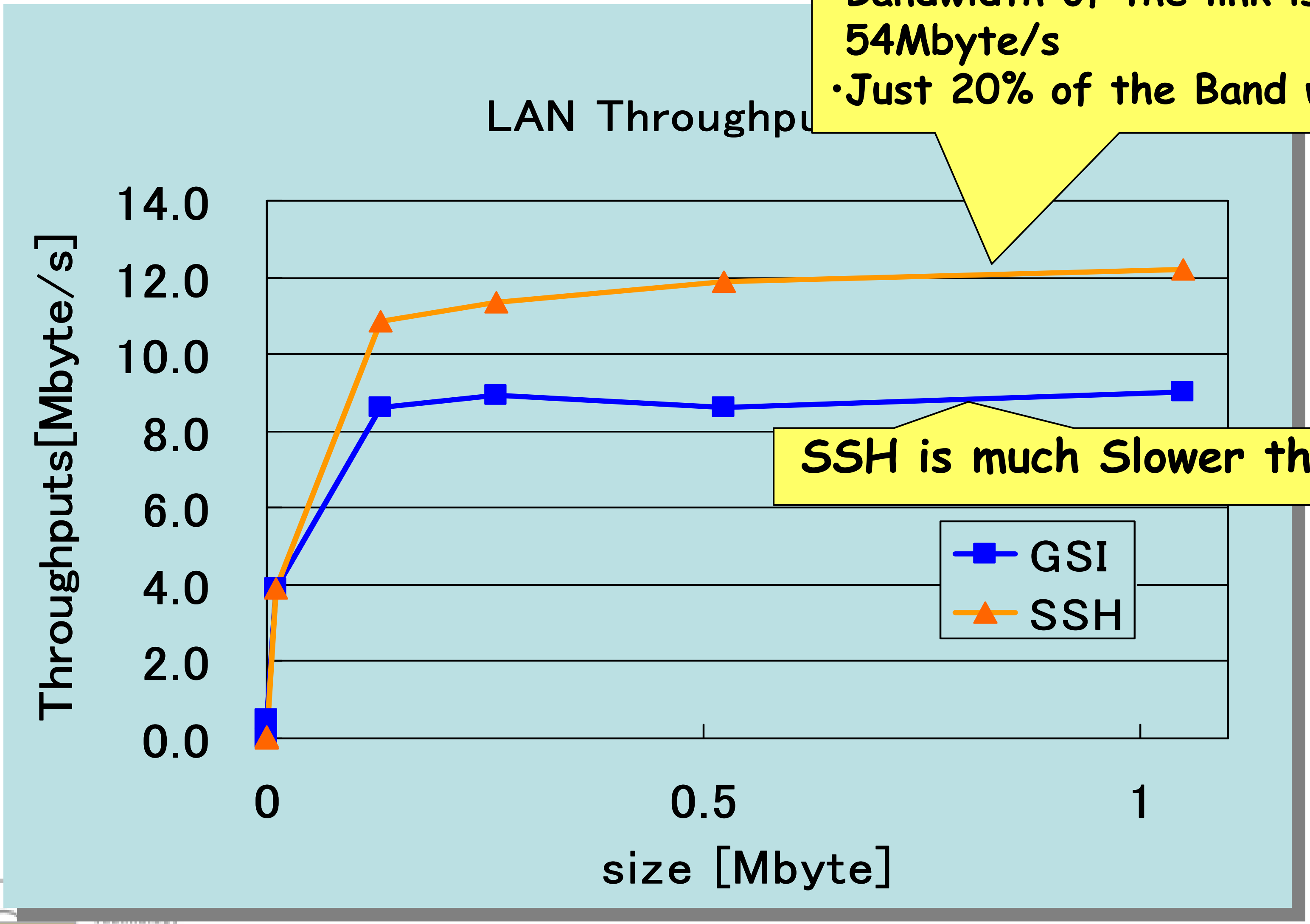
Result(WAN)



- Bandwidth of the link is 10Mbyte/s
- 70-80 % of the bandwidth

SSH is faster slightly

Result(LAN)

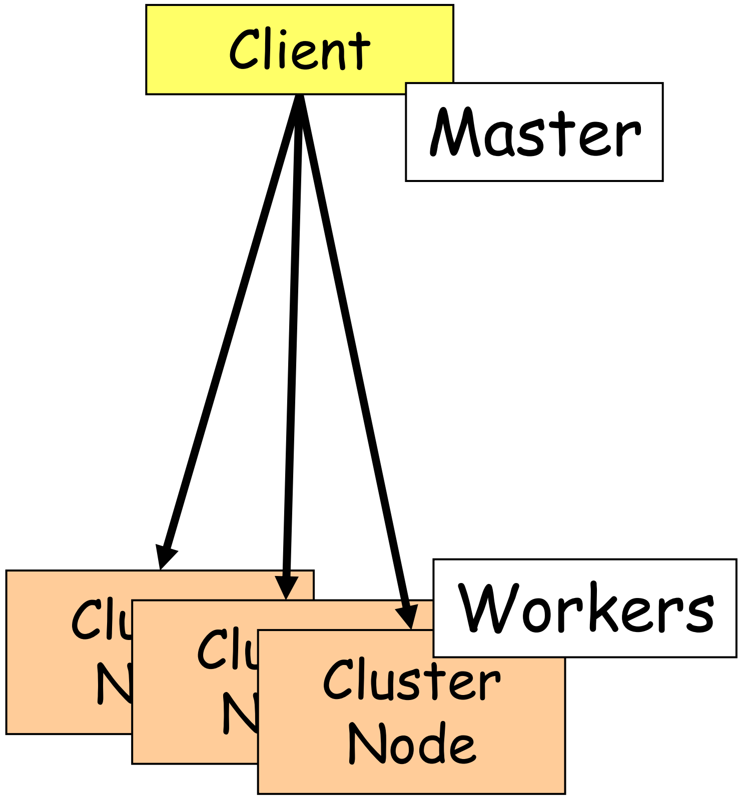


- Bandwidth of the link is 54Mbyte/s
- Just 20% of the Band width

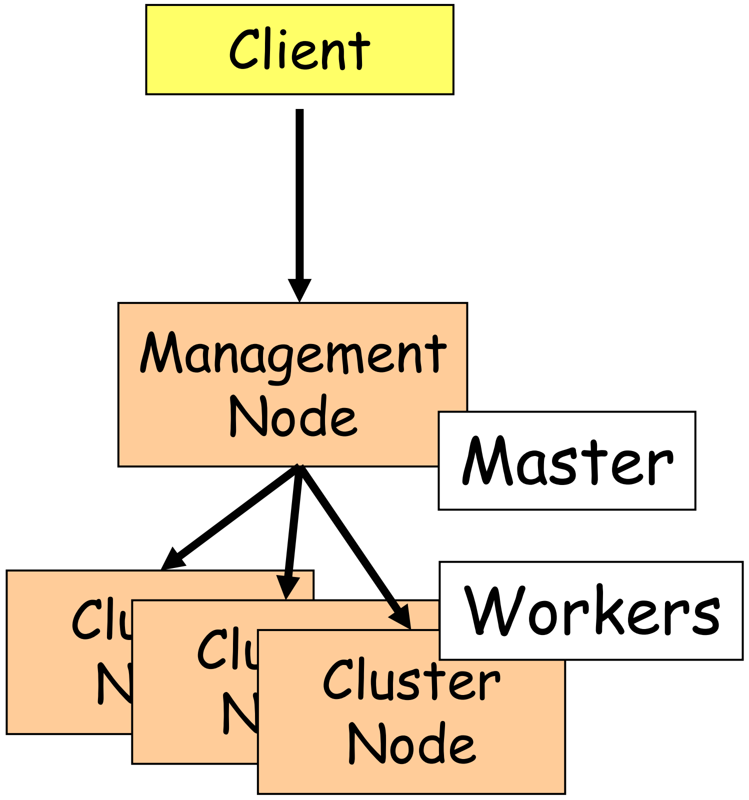
SSH is much Slower than GSI

Master-Worker evaluation

- **Compare 2-layered and 3-layered setting**



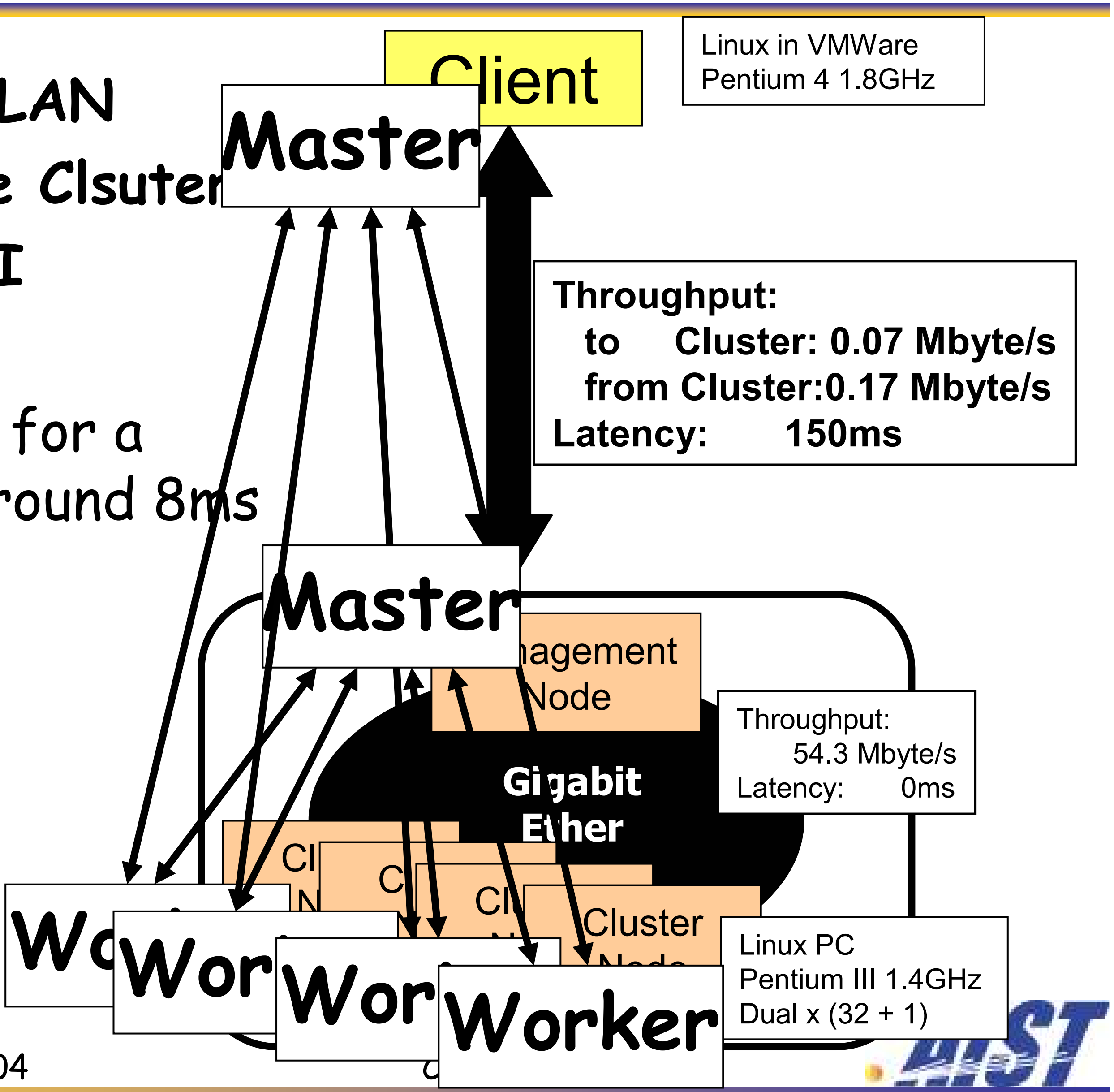
2layered setting



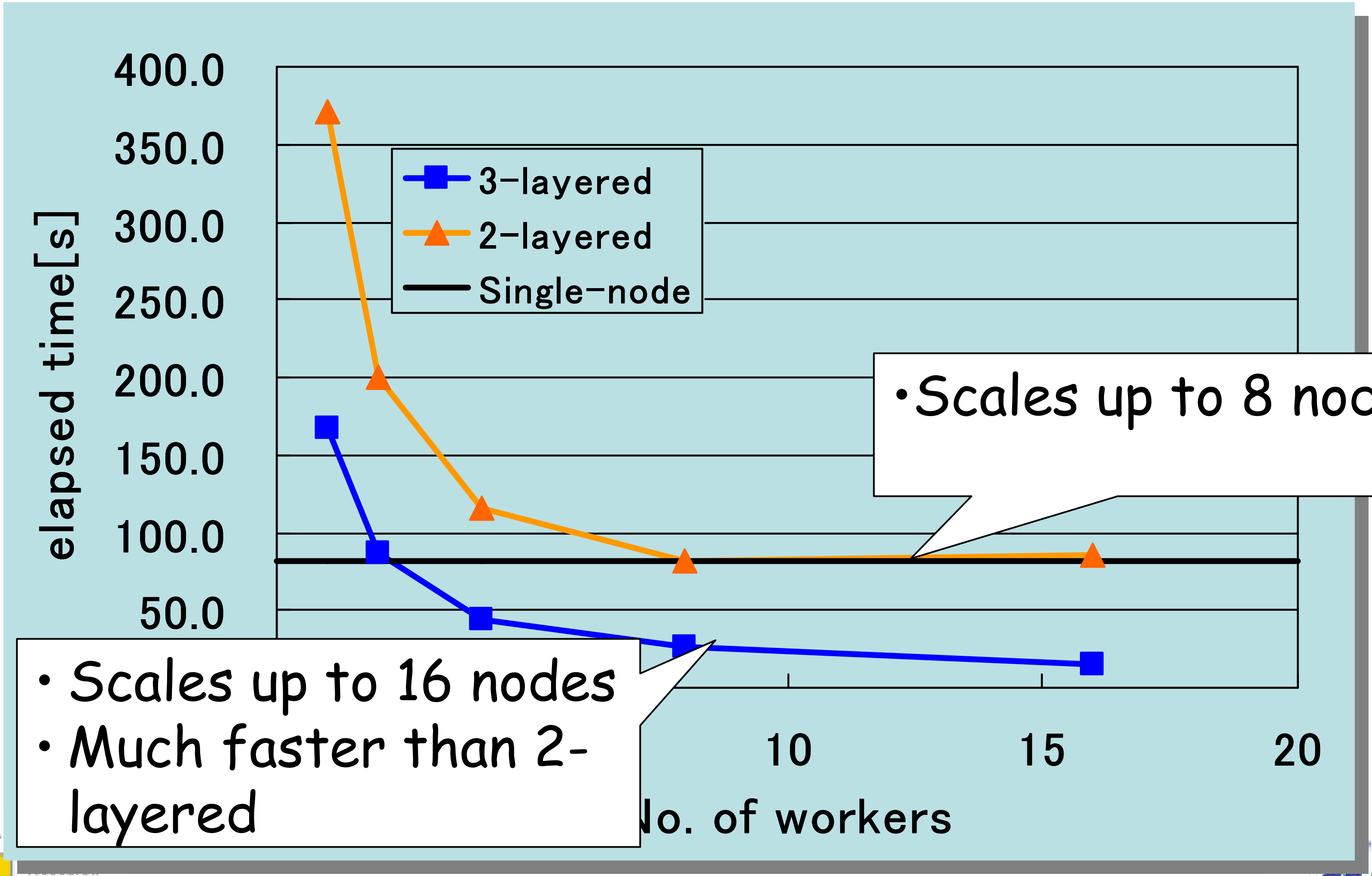
3layered setting

Experiment Environment

- CATV+Wireless LAN
- Giga-Ether in the Clsuter
- Master-worker PI
  - 10^4 tasks
  - Execution time for a single task is around 8ms



Master-Worker result



•Scales up to 8 nodes

- Scales up to 16 nodes
- Much faster than 2-layered

Discussion

- **Data size for each task is just few bytes**
  - Data transfer time is negligible
  - Latency does slow the execution
- **Execution time for each task is just 8ms**
  - This application may be not suitable for master-worker execution
    - As shown in the 2-layered model score
  - Still can be effectively executed in 3-layered model

Summary

Jojo works well with hierarchical Grids

- Firewall-aware
- No-pre installation required

Jojo Provides simple, easy-to-use API

- To hide latency

Preliminary evaluation shows

- It is fast enough for WAN
- With hierarchical setting we can take advantage of high speed LAN for master-worker programs

Future work

- **Scalability evaluation**
  - Planning to perform experiments with thousands of PEs using Genetic Algorithm and Branch and Bound method programs
- **Fault Tolerance**
  - Single trouble may stop the whole computation
  - Jojo API designed to be generic, but we found that the API design is preventing the system being FT
  - Redesign the API to enable Jojo to be FT

Thank you!