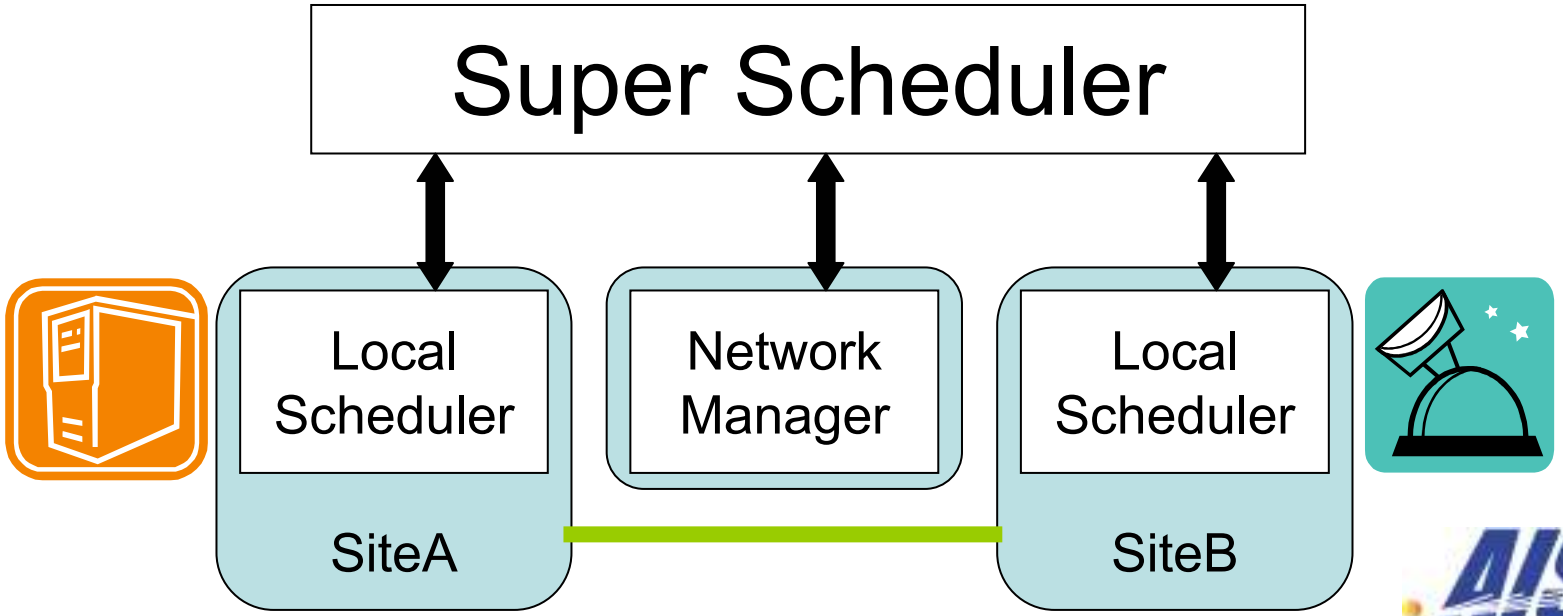
PluS: An Advance Reservation Plug In for Sun Grid Engine

Hidemoto Nakada, Astuko Takefusa
Tomohiro Kudoh, Yoshio Tanaka,
Satoshi Sekiguchi

Background

- Large scale computation with Grid technology
  - Resources are spanning on several sites
  - Co-allocation of multiple resources is essential
- Resources might include
  - Network
  - Appliances, such as Telescopes, Microscopes, Storages



Background, cont.

- Advance Reservation is one of the easiest way to accomplish resource co-allocation
  - Specify a time slot and make reservations on all the resource in advance
- Note:
  - Computational resources might not be important as other resources
    - Appliances are more rare resources
    - Computers are cheap

    -> Preemption of existing jobs is required to fully utilize the more rare (important) resources

AIST

What is PluS?

- A plug-in module to enable advance reservation
  - Preemption based
    - Reserved jobs will kick out running non-reserved jobs
  - Two implementations
    - Queue-based
    - Scheduling module replacement
- It also provides a toolkit to write your own scheduler

AIST

Why PluS ?

- Q: SGE natively supports AR, finally, why PluS?
- A:
  - Still missing functionalities
    - Job preemption
      - Computation resources are cheaper than other resources
    - Modification with 2 phase-commit
    - Policy enforcement on reservation request
  - Two years ago, it did not. ☹

Overview of the talk

- How PluS looks Like
  - CLI
- PluS implementation
  - Scheduling module replacement
  - Queue control
- Policy enforcement in PluS
- PluS as a scheduler toolkit

CLI: Reservation Related Commands

- **plus_reserve**
  - Requests for a reservation
  - In: start/end time, # of Nodes
  - Out: Reservation ID
- **plus_cancel**
  - Cancel a reservation
  - In: Reservation ID
- **plus_status**
  - Query status of the reservation
  - In: Reservation ID
  - Out: Status of the reservation
- **plus_modify**
  - Modify the reservation
  - In: Reservation ID, start/end time, # of Nodes

AIST

Reservation Usage Scenario

- Make a reservation

```
> plus_reserve -s 12:00 -e 14:00 -n 1
Reserve succeeded: reservation id is 14
```

- Confirm the reservation with the reservation ID

```
> plus_status
 id owner          start             end     duration state
 R14 nakada     Feb 20 12:00  Feb 20 14:00     2h00m Confirmed
```

- Submit a job with the reservation ID

```
> qsub -q R14 script
```

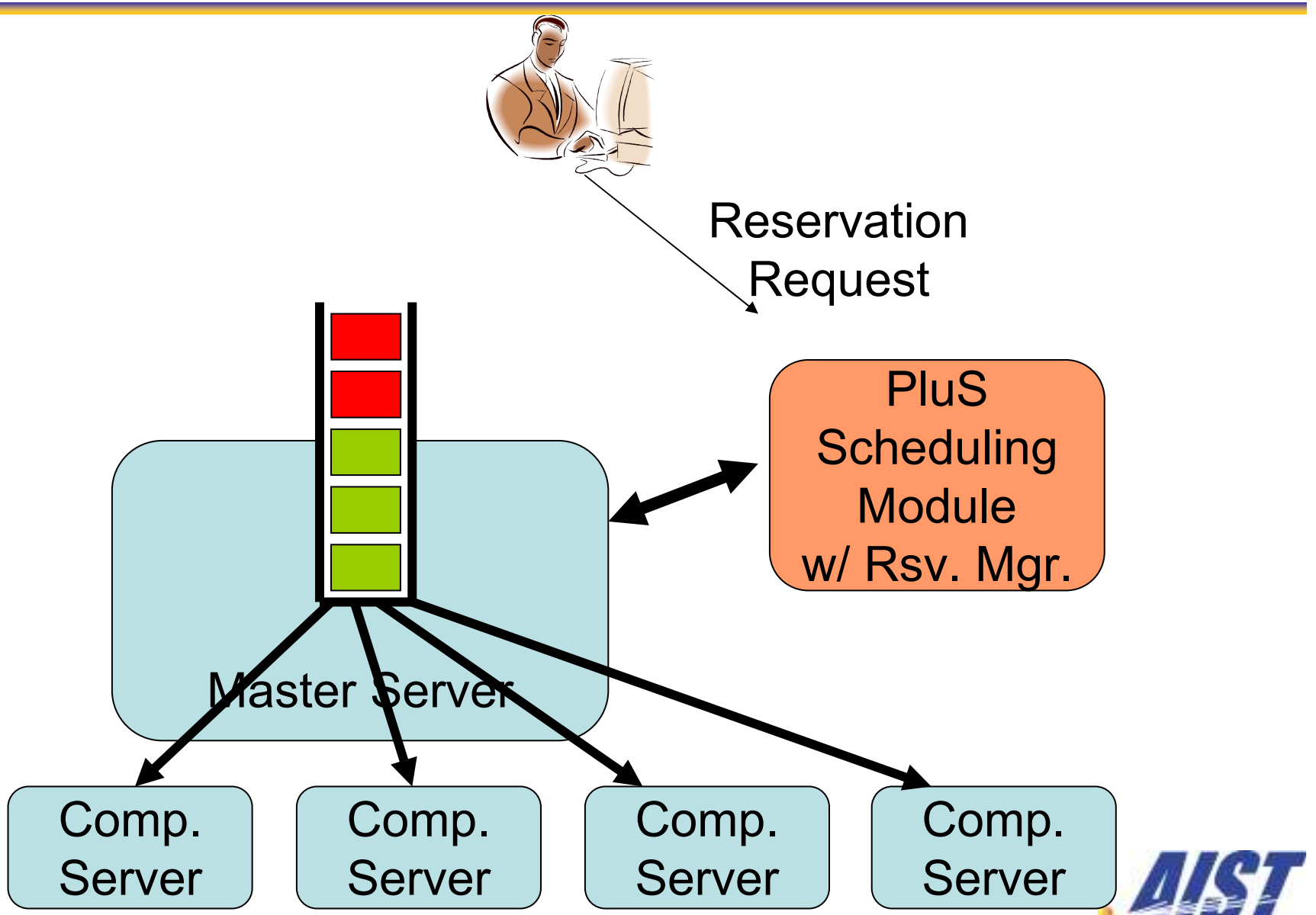
Overview of the talk

- How PluS looks Like
  - CLI
- PluS implementation
  - Scheduling module replacement
  - Queue control
- Policy enforcement in PluS
- PluS as a scheduler toolkit

Implementation

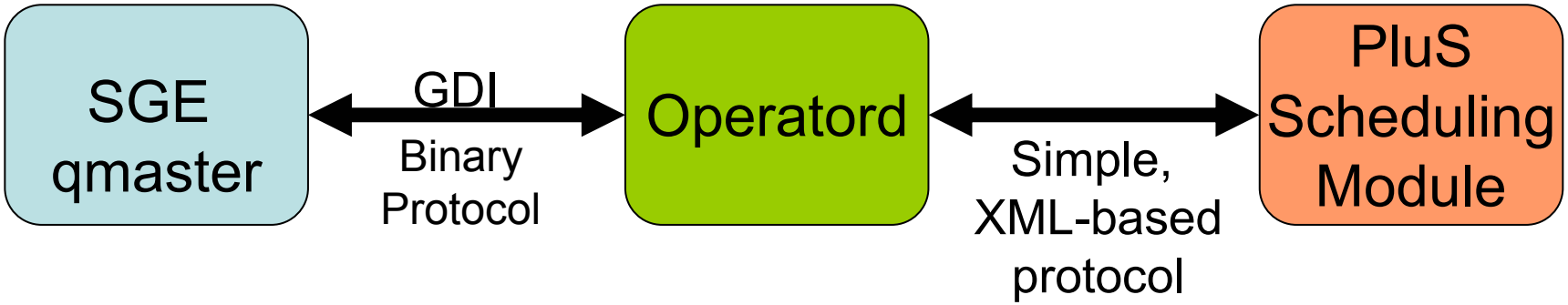
- PluS has two styles of implementation
  - Scheduling module replacement
    - Completely replaces the scheduling module of SGE
  - Queue control
    - Enables advance reservation using job queues.

Scheduling Module Replacement



PluS operatord

- **Operatord: a proxy module that translate the binary protocol into plain text protocol**
  - Sits between the scheduling module and the master module
  - Operatord is implemented in C, to leverage GDI (GridEngine Database Interface) implementation
  - Scheduring module can be implemented in any language, since the operatord can talk plain text protocol.



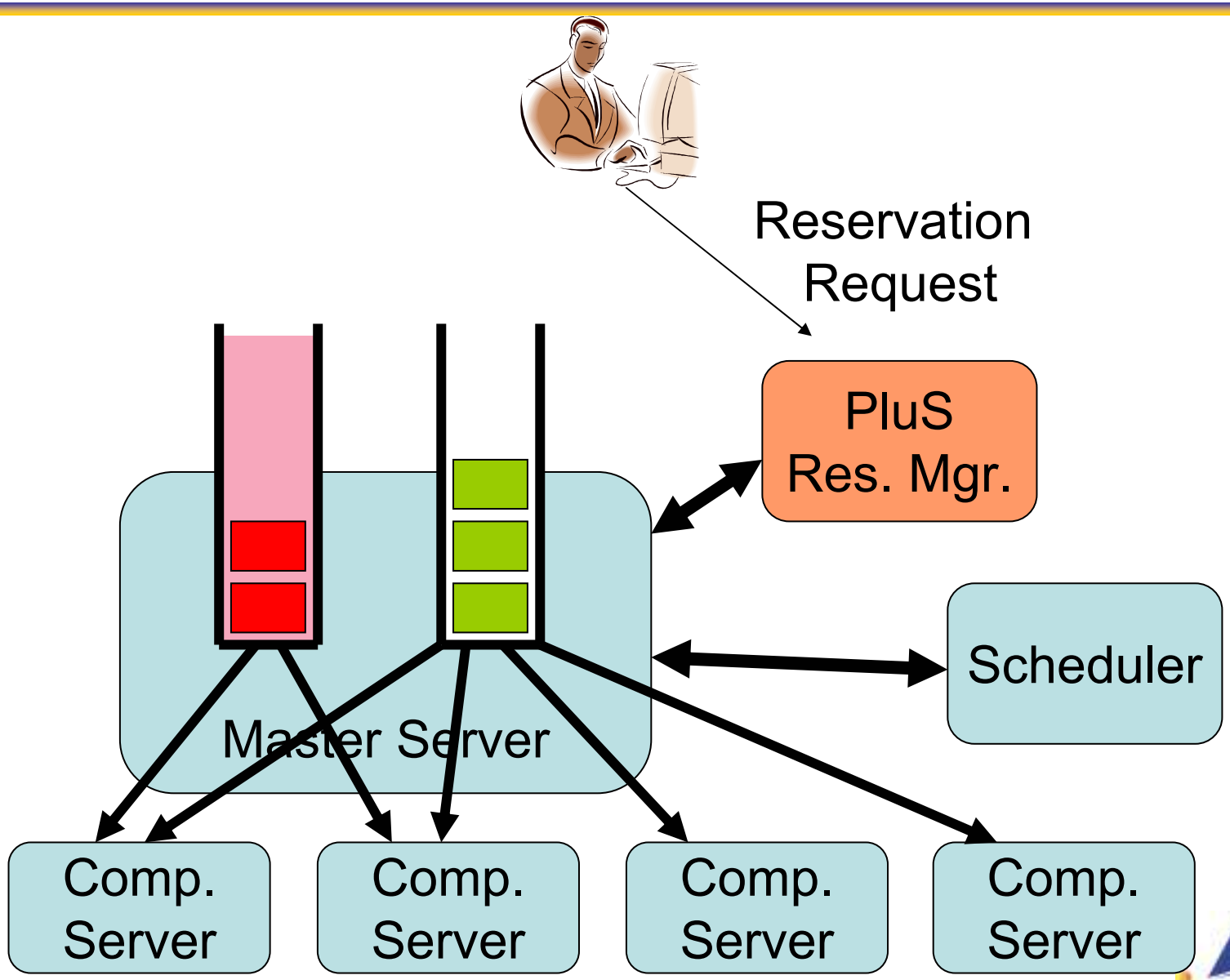
not working for 6.2, now ☹

Advance Reservation with Queue Control

- **What are queues?**
  - Abstract ‘submit point’ for jobs
  - Can be allocated for specific group of users
  - Can be allocated for specific set of nodes
- **Advance Reservation by Queue Control**
  - Create Advance Reservation as a queue
  - Activate the queue for specific time of period
- **Key Characteristics of the Method**
  - ○ (Relatively) Easy to implement
  - ○ No need to understand internal protocol of the target system - means easy to catch up updates.
  - × Requires multiple invocations of command to control queues - overhead

AIST

Advance Reservation by Queue Control



Overview of the talk

- How PluS looks Like
  - CLI
- PluS implementation
  - Scheduling module replacement
  - Queue control
- Policy enforcement in PluS
- PluS as a scheduler toolkit

Reservation Acceptance Policy

- Reservation once accepted should not be canceled or delayed by the system
  - Cancel or delay will screw up co-allocated resources
  - Acceptance is the only chance to enforce site-specific policy
- What are needed
  - User priority
  - Resource affinity

Policy Implementation

- Employ Condor ClassAd for policy description
  - Administrators define site-specific policies in ClassAds.
  - PluS provides node status and job status information as ClassAds
  - Client requests are translated into ClassAds
  - Evaluate all the ClassAds and determine whether to accept request or not.

What is ClassAd

- Developed for Condor Project by Univ. Wisconsin
  - Classified Advertisement
  - Condor uses ClassAd for ‘matchmaking’, i.e. allocation of resources for jobs
    - Jobs advertise resource requirements as ClassAds
    - Resources advertise their status as ClassAds
    - Negotiator make match with jobs and resources
- Employed by other projects
  - EGEE gLite
  - CoG kit

AIST

ClassAd as a policy language

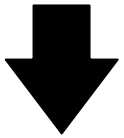
- Declarative, not imperative
  - Side-effect free
- Evaluation cost is linearly proportional to the length of the expression

- Primitives
  - Integer, floating, string, boolean, time, time period
  - Error, Undefined
- Structured
  - Record : Dictionary with keys and values
  - List: enumeration
  - Can be nested

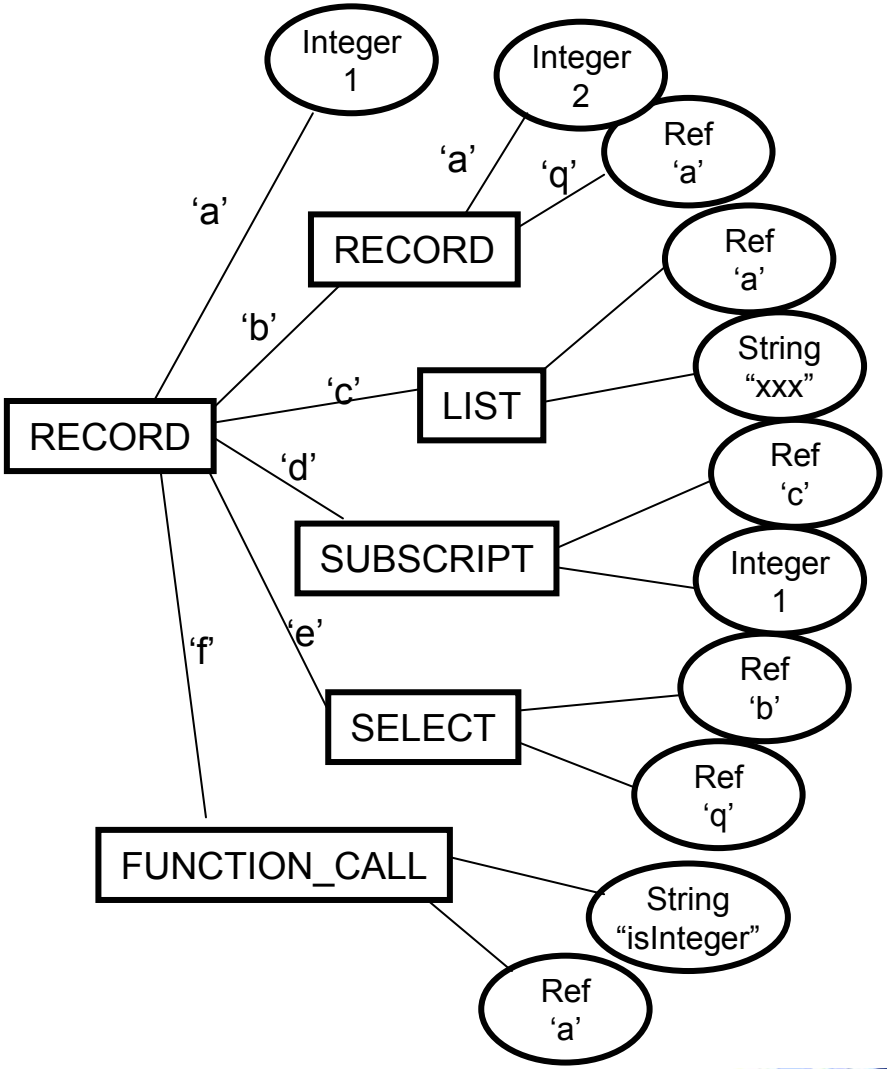
AIST

ClassAd Evaluation

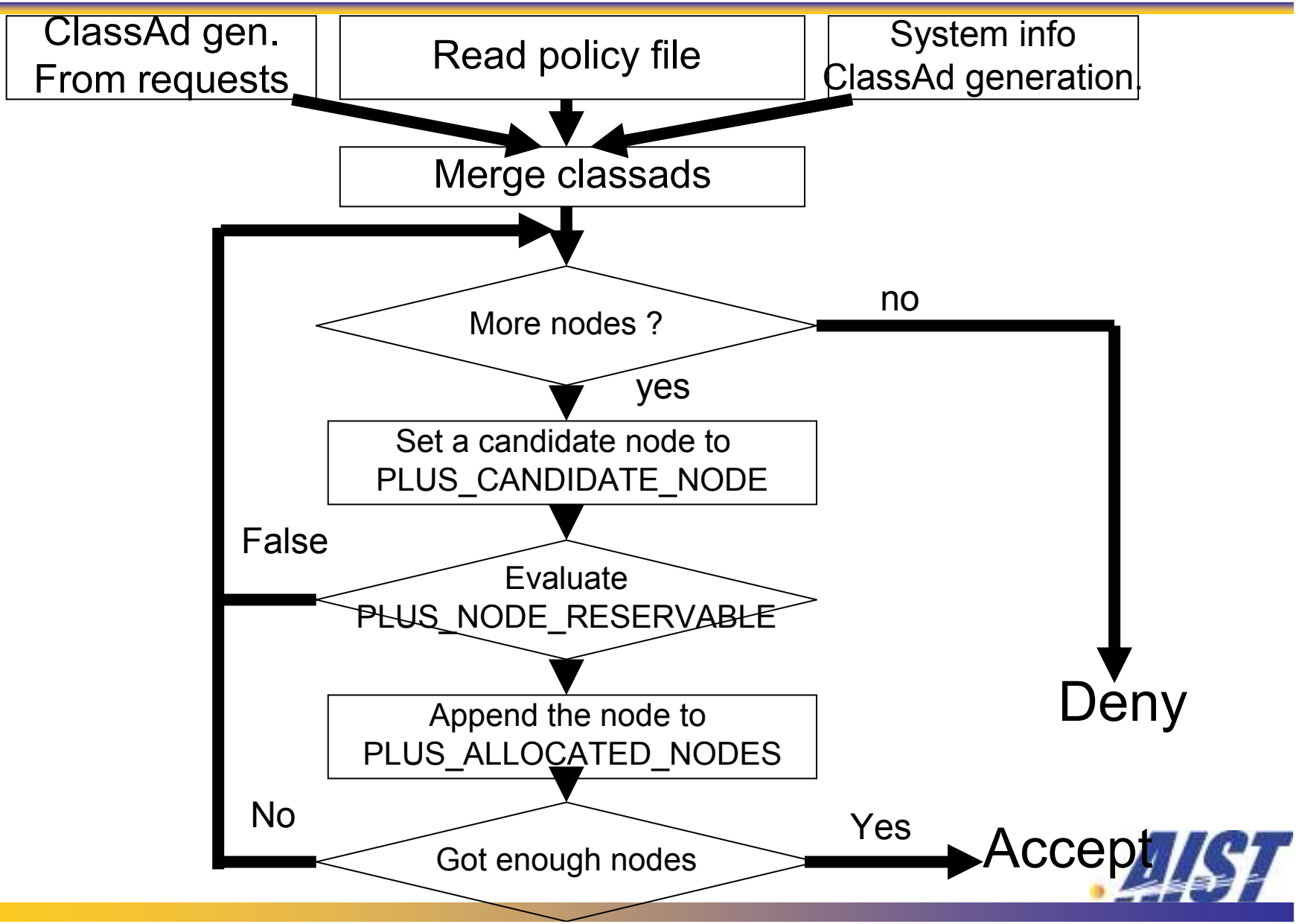
```
[
  a = 1;
  b = [a = 2; q = a;]
  c = {a, "xxx"};
  d = c[0];
  e = b.q;
  f = isInteger(a);
]
```



```
[
  a = 1;
  b = [a = 2; q = 2;]
  c = {1, "xxx"};
  d = 1;
  e = 2;
  f = true;
]
```



Policy decision flow



System provided information for policy decision

- **Request information**
  - PLUS_RSV_OWNER: string： requesting user
  - PLUS_RSV_START: time：reservation period start time
  - PLUS_RSV_END: time： reservation period end time
- **System internal status**
  - PLUS_ALL_NODES: List of node records： all the nodes
  - PLUS_CANDIDATE_NODE: node record ：policy target node
- **Node selection information**
  - PLUS_ALLOCATED_NODES: List of node records： nodes already allocated during the current reservation cycle

Node record and Job record

- **Node Record**
  - name — string : name of the node
  - isAlive — boolean : execution daemon status
  - loadavg — float : load average
  - nRunJobs — integer : number of running jobs
  - jobs — list of job records : running jobs status
- **Job Record**
  - id — string : job ID
  - owner — string : job owner
  - state — string : job status
    - Queued, Running, Exiting, Held, Suspend
  - priority — integer : priority
  - starttime — time : job start time
  - walltime — time period : walltime

AIST

Policy Example 1

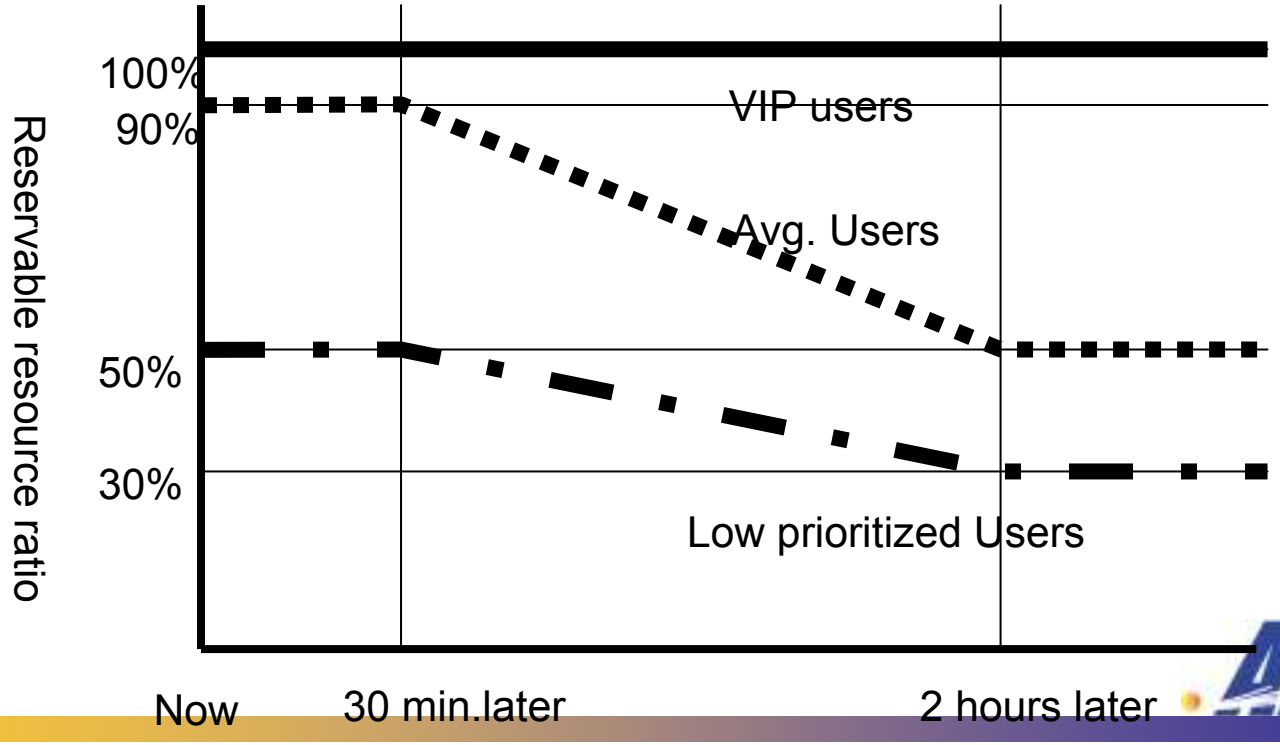
- Simple Policy
  - Provides nodes, which does not have running job, is alive, with low load average, and is not in the UnusableNode list.

```
[
  UnusableNodes = {"unusableA","unusableB" };

  PLUS_NODE_RESERVABLE =
     (PLUS_CANDIDATE_NODE.nRunJobs == 0   &&
      PLUS_CANDIDATE_NODE.isAlive         &&
      PLUS_CANDIDATE_NODE.loadavg <= 0.3 &&
      !member(PLUS_CANDIDATE_NODE.name,
              UnusableNodes));
]
```

Policy example2

- Specifies maximum ‘allocatable resource ratio’, depends on user ‘rank’
- The maxim ratio changes along with time
  - The restriction is loosened just before the time.



Policy example2

```
[ MaxPeriod = relTime("00:30:00");
  MinPeriod = relTime("02:00:00");
  LimitPeriod = relTime("7d");
  MaxReserveDuration = relTime("2d");
  VIPs = { "userA", "userB", "userC" };
  Users = { "userX" };
  VIPRatio = 100.0;
  UsersMaxRatio = 90.0;
  UsersMinRatio = 50.0;
  OthersMaxRatio = 50.0;
  OthersMinRatio = 30.0;

  MaxRatio = member(PLUS_RSV_OWNER, Users) ?  UsersMaxRatio : OthersMaxRatio;
  MinRatio = member(PLUS_RSV_OWNER, Users) ?  UsersMinRatio : OthersMinRatio;
  now = absTime(time());
  prev = PLUS_RSV_START - now;
  duration = PLUS_RSV_END - PLUS_RSV_START;
  ratioFunc = linear(prev, MaxPeriod, MaxRatio, MinPeriod, MinRatio);
  rsvRatio = (prev<=relTime("0") || LimitPeriod<=prev || duration>=MaxReserveDuration)?0:
        member(PLUS_RSV_OWNER, VIPs) ? VIPRatio :
           (prev <= MaxPeriod) ? MaxRatio :
              (prev >= MinPeriod) ? MinRatio : ratioFunc;

  nAllocate =    size(PLUS_ALLOCATED_NODES) + 1;
  nAllocatable = size(PLUS_ALL_NODES) * 0.01 * rsvRatio;
  PLUS_NODE_RESERVABLE = (nAllocate <= nAllocatable);
]
```

Policy example3

- History based policy
  - Refer previous reservations from the requesting user
  - Special built-in function ‘plus_rsv_util’
    - Returns history information

```
[ UtilCheckPeriod = relTime("7d");
  MaxReserveCount = 100;
  MaxReserveDuration = relTime("2d");
  MaxReserveHourNode = 1000.0;
  now  = absTime(time());
  util = plus_rsvutil(PLUS_RSV_OWNER,
          now - UtilCheckPeriod, now + UtilCheckPeriod);
  PLUS_NODE_RESERVABLE =((util[0] <= MaxReserveCount)     &&
                         (util[1] <= MaxReserveDuration)  &&
                         (util[2] <= MaxReserveHourNode));
]
```

Overview of the talk

- How PluS looks Like
  - CLI
- PluS implementation
  - Scheduling module replacement
  - Queue control
- Policy enforcement in PluS
- PluS as a scheduler toolkit

PluS as a scheduling module toolkit

- PluS provides Java API to write your own scheduling modules for Sun Grid Engine
  - Every organization has its own policies and requirements for scheduling. PluS makes it easy to implement your own scheduler
    - Provide Java API to talk with qmaster
  - C.f. TORQUE (OpenPBS)
    - There are several 3rd party schedulers include Catalina from SDSC, Maui from Cluster Resources, inc.
    - TORQUE uses simple text base protocol between the master module and scheduling module, encouraging third party implementation of the scheduling module

AIST

PluS Scheduling Module API

```
public interface MainServer {
    void runJob(JobID jobID, NodeStatus node);
    void runJob(JobStatus job, Collection<NodeInfo> nodes);
    void deleteJob(JobID jobID);
    void suspendJob(JobID jobID);
    void resumeJob(JobID jobID);
    void holdJob(JobID jobID);
    void releaseJob(JobID jobID);
    void rerunJob(JobStatus job);
    void signalJob(JobID jobID, SignalType signal);
// status collector
    Collection<NodeStatus> getNodeStatus();
    Collection<QueueStatus> getQueueStatus();
    Collection<JobStatus> getJobStatus();
}
```

Ex. Simple FIFO Scheduler

```
Collection<NodeStatus> nodes = srv.getNodeStatus();
for (JobStatus job : srv.getJobStatus()) {
        if (job.getState() == JobStateType.Queued) {
                for (NodeStatus node : nodes) {
                        // run this job on a first found node
                        srv.runJob(job.getJobID(), node);
                        return;
                }
        }
}
```

Not working for 6.2, now ☹

- In 6.2, scheduler is embedded in the qmaster as a thread
- We'll find some workaround

Current Status

- PluS is available from

  http://www.g-lambda.net/plus

  - Works with Sun Grid Engine, TORQUE, and Condor (experimental)

Summary

- PluS provides Sun GridEngine with advance reservation capability
  - Pre-emptive
  - Policy specification with ClassAds
- PluS also works as a toolkit to implement your scheduler
  - Provides Java API to implement scheduler that works with GridEngine

AIST

Acknowledgement

This work is partly funded by the Science and Technology Promotion Program's "Optical Paths Network Provisioning based on Grid Technologies" of MEXT, Japan.

http://www.g-lambda.net/plus

AIST