Condor: High-throughput Computing From Clusters to Grid Computing

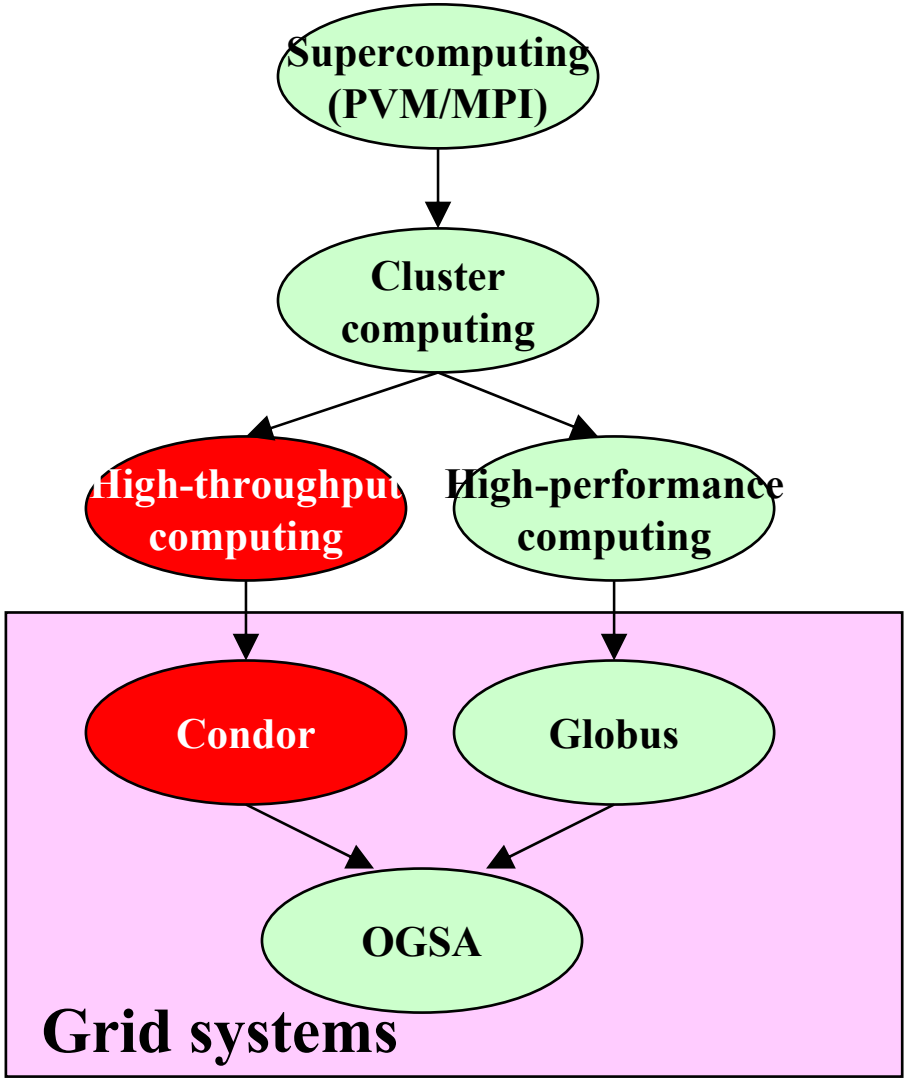
P. Kacsuk – M. Livny

Univ. of Westminster – Univ. of Wisconsin-Madison

kacsuk@sztaki.hu

www.lpds.sztaki.hu

Progress to Grid



Goals of Condor

Goal:
Grid technology should turn every "ordinary" user into a "supercomputing" user.

Computational Grids should give us access to resources ***anywhere****.*

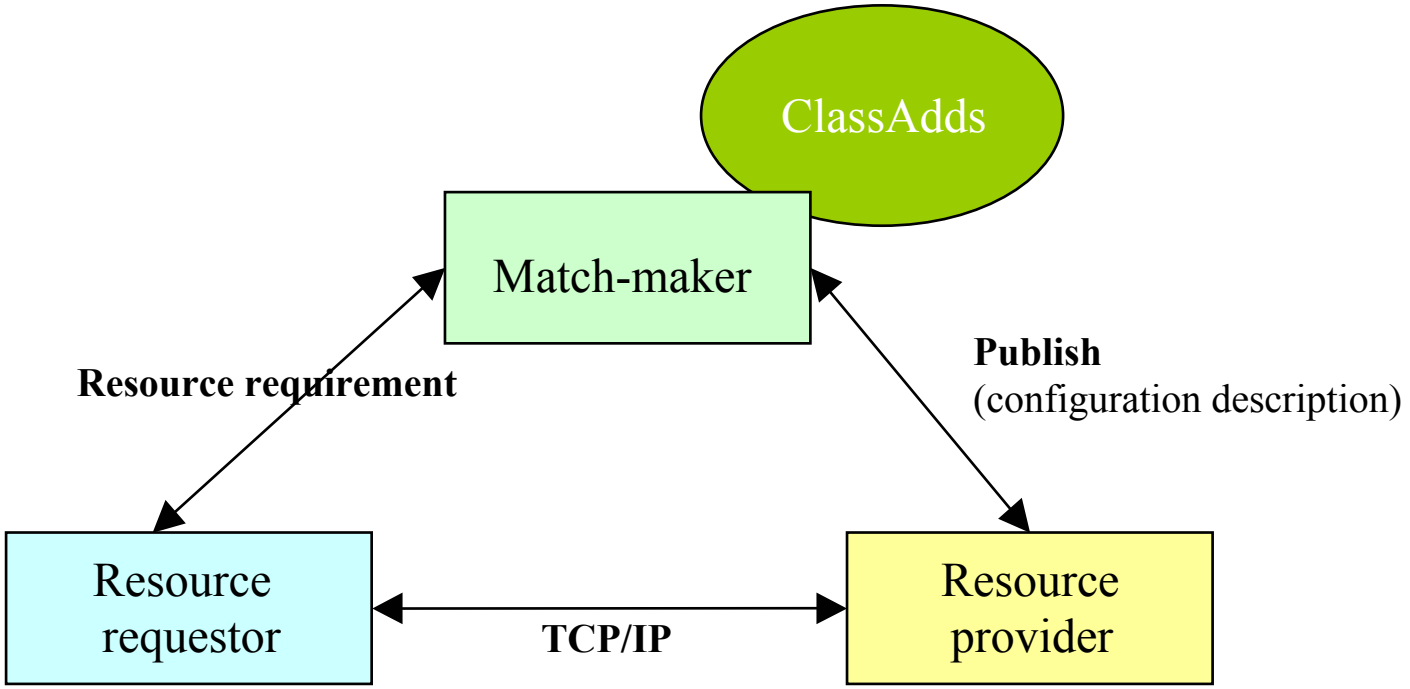
Question:
How can we make them usable by ***anyone****?*

Main features of Condor

Special resource management (batch) system

- Distributed, heterogeneous system.
- Goal: exploitation of spare computing cycles.
- It can migrate sequential jobs from one machine to another.
- The ClassAds mechanism is used to match resource requirements and resources

The Condor model



ClassAds

- Resources of the Grid have different properties (architecture, OS, performance, etc.) and these are described as advertisements (ClassAds)
- Creating a job, we can describe our requirements (and preferencies) for these properties.
- Condor tries to match the requirements and the ClassAds to provide the most optimal resources for our jobs.

Requirements and Ranks

Requirements = Arch=="SUN4u"
&& OpSys == "SOLARIS251"

Exact matching is needed..

Rank = Memory + Mips

If there is a choice, Condor will choose the resource with bigger memory. If all the memories are the same, it will choose the faster machine.

The two sides to match (1)

User side: submit command file

```
Requirements = Arch == "INTEL"
               && OpSys == "LINUX"
Rank = Memory + Disk + Mips
```

The two sides to match (2)

Resource side: configuration file (owners of resources may place constrains and preferences on their machines)

```
Friend  = Owner == "haver"
Trusted = Owner != "judas"
Mygroup = Owner == "zoli" || Owner == "jani"
Requirements  = Trusted && (Mygroup ||
      LoadAvg < 0.5 && KeyboardIdle > 10*60)
Rank = Friend + MyGroup
```

Condor Universes

- Standard
- Vanilla
- PVM
- MPI
- Globus

Standard universe

- checkpointing, automatical migration for sequential jobs
- Existing program should be re-linked with the Condor instrumentation library
- The application cannot use some system calls (fork, socket, alarm, mmap)
- Grabs file operations and passes back to the shadow process

Vanilla universe

- No checkpointing, no migration
- The existing executable can be used without re-compiling or re-linking
- There is no restriction for system calls
- NFS, or AFS is needed.

PVM universe

- To run MW (Master/Worker) PVM programs
- PVM 3.4.2 + extensions for task management
- Dinamic Virtual Machine creation.
- Support for heterogeneous environment
- User can define check-points in the master process
- Worker processes are net check-pointed

MPI universe

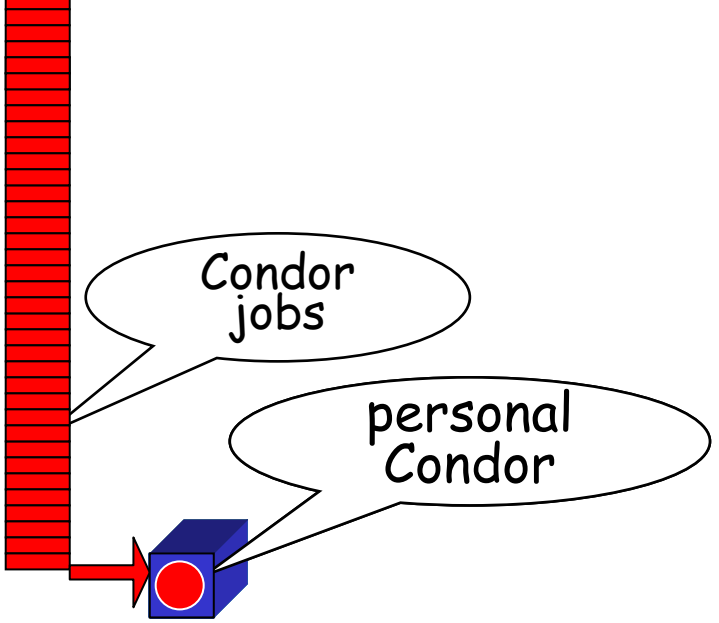
- MPICH usage without any necessary changes
- Dinamic changes are not supported
- No check-pointing
- The application cannot be suspended
- NFS or AFS is needed.

Turn your workstation into a Personal Condor

- Install the Condor software on your workstation:
  - submit
  - execute
  - scheduling services.
- Submit your application to your Condor system, for example, as a "Standard" universe job

Your Personal Condor ...

- ... provides reliable and recoverable job and resource management services
- ... keeps an eye on your jobs and keeps you posted on their progress
- ... implements your policy on
  - when the jobs can run on your workstation
  - the execution order of the jobs
- .. adds fault tolerance to your jobs
- ... keeps a log of your job activities


Condor
jobs
personal
Condor

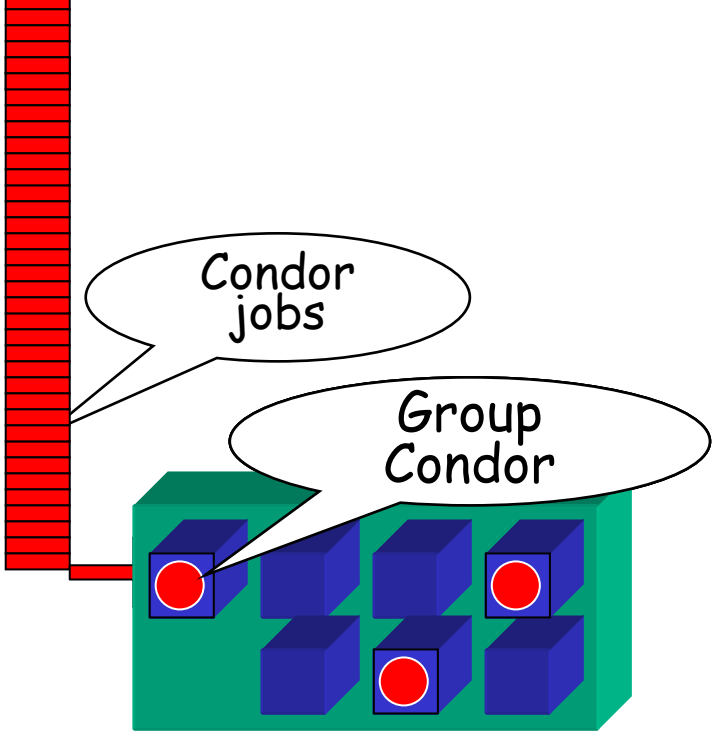
Components of Your Personal Condor

- Central manager
  - Scheduler
  - Matchmaker
- Checkpoint server
- Submit daemon
- Startd daemons (each representing a worker)

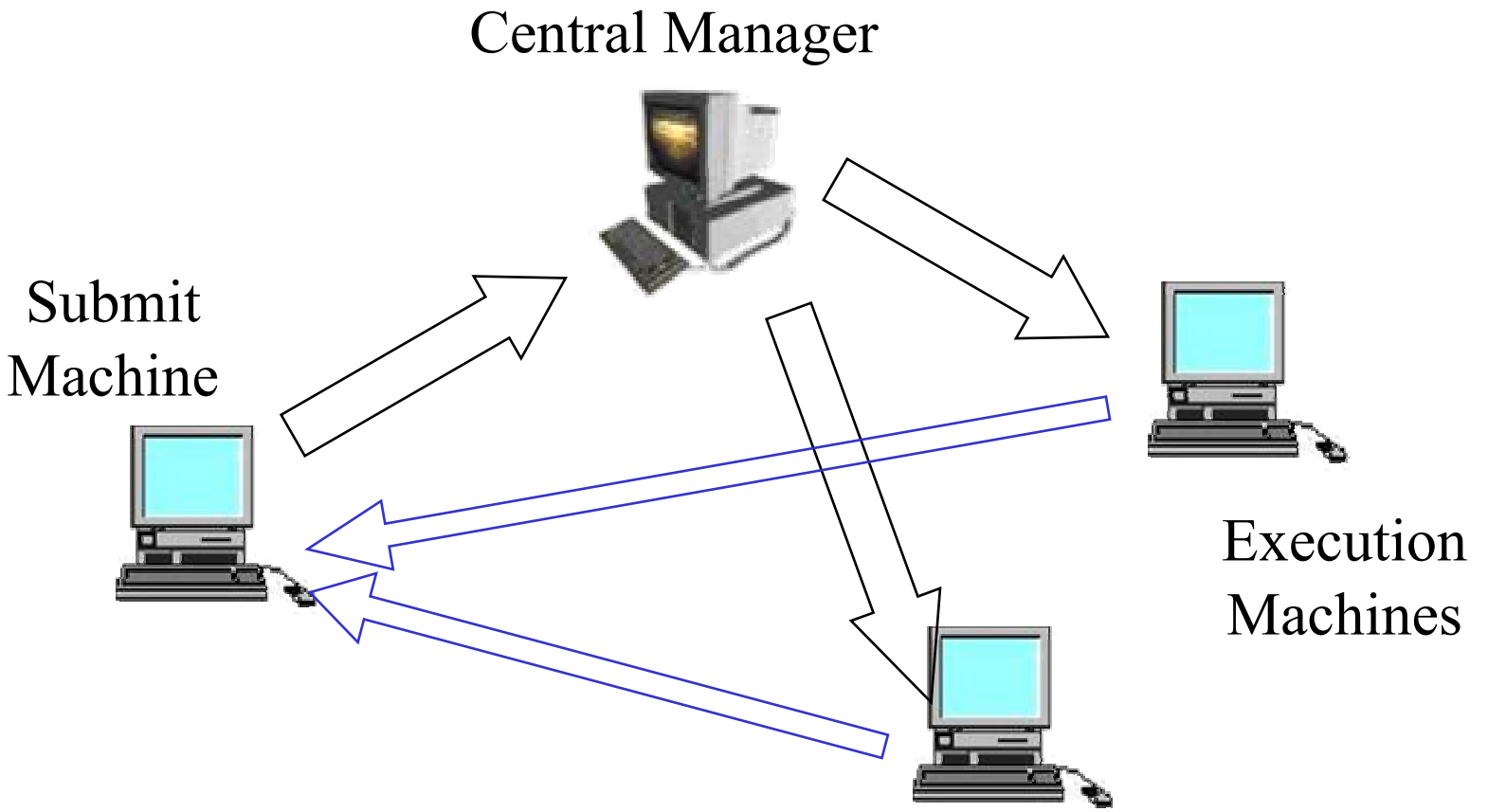
- All of them on your personal computer

Build a personal Condor Pool

- Install Condor on the desk-top machine next door
- Install Condor on the machines in the class room
- Install Condor on the O2K in the basement.
- Configure these machines to be part of your Condor pool


Condor
jobs
Group
Condor

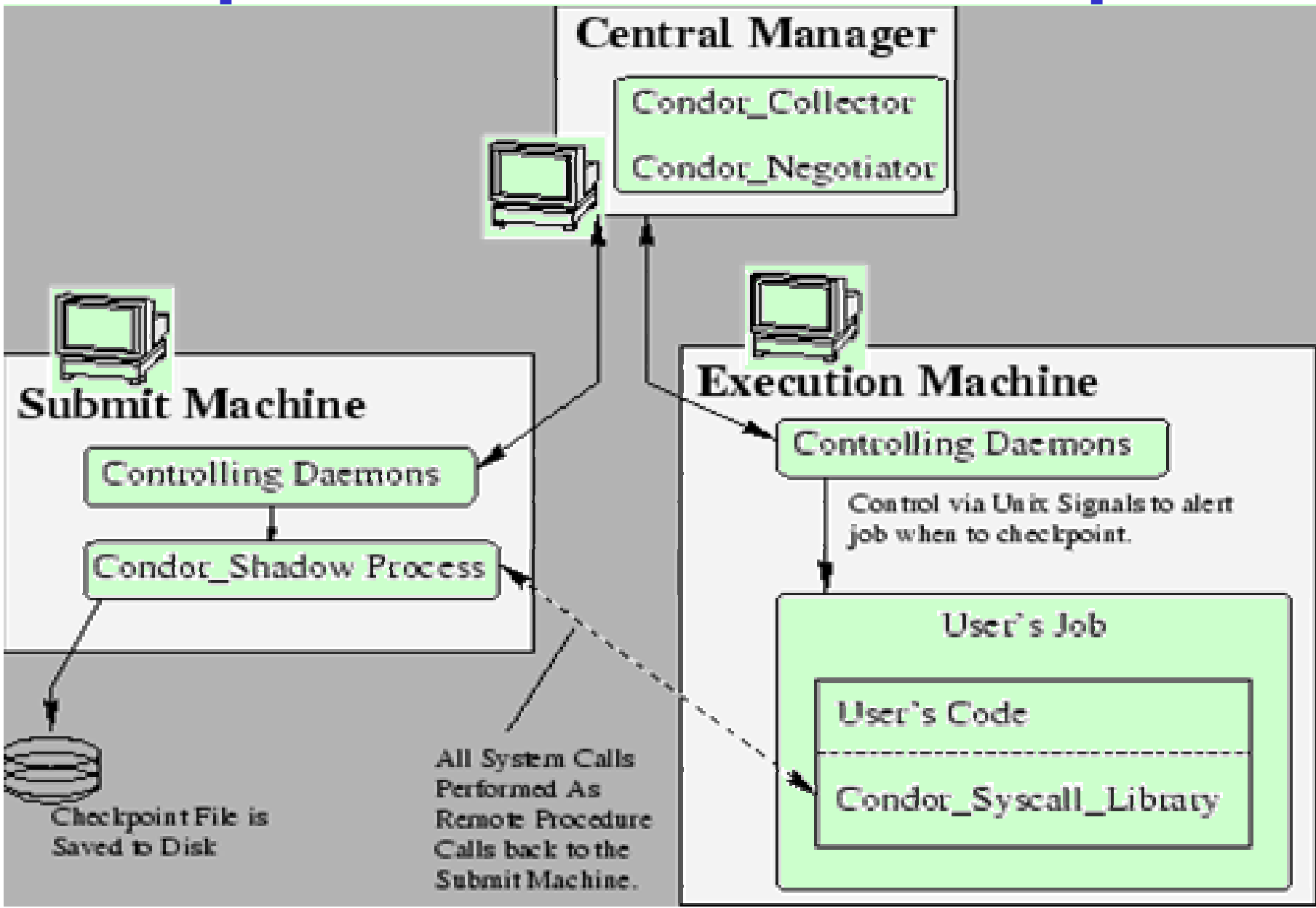
Architecture of a Condor pool



Components of Your Personal*Condor Pool

- A particular host of the pool:
  - Central manager
    - Collector (scheduler)
    - Negotiator (matchmaker)
- On each host of the pool:
  - Controlling daemons
    - Submit machine: Submit daemon
    - Execution machine: Startd daemon
  - Checkpoint server (on Submit machine)

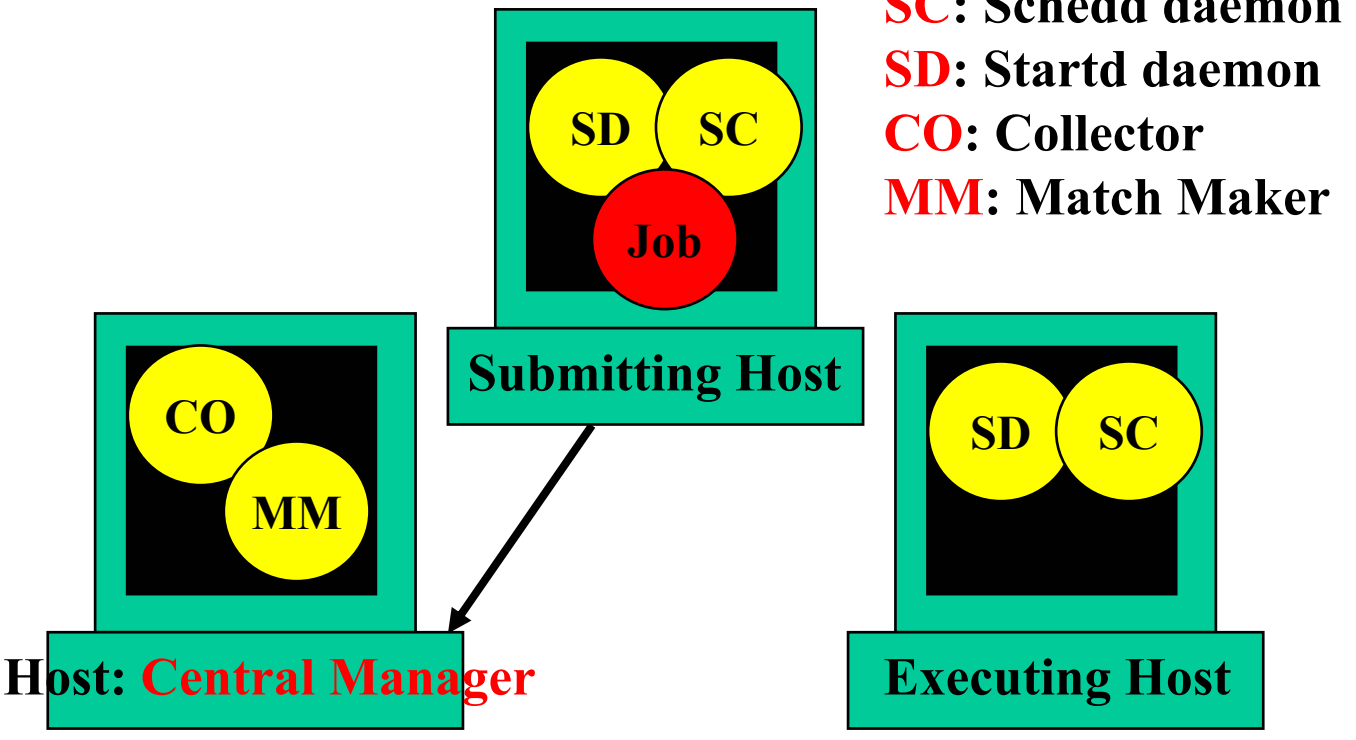
Components of a Condor pool



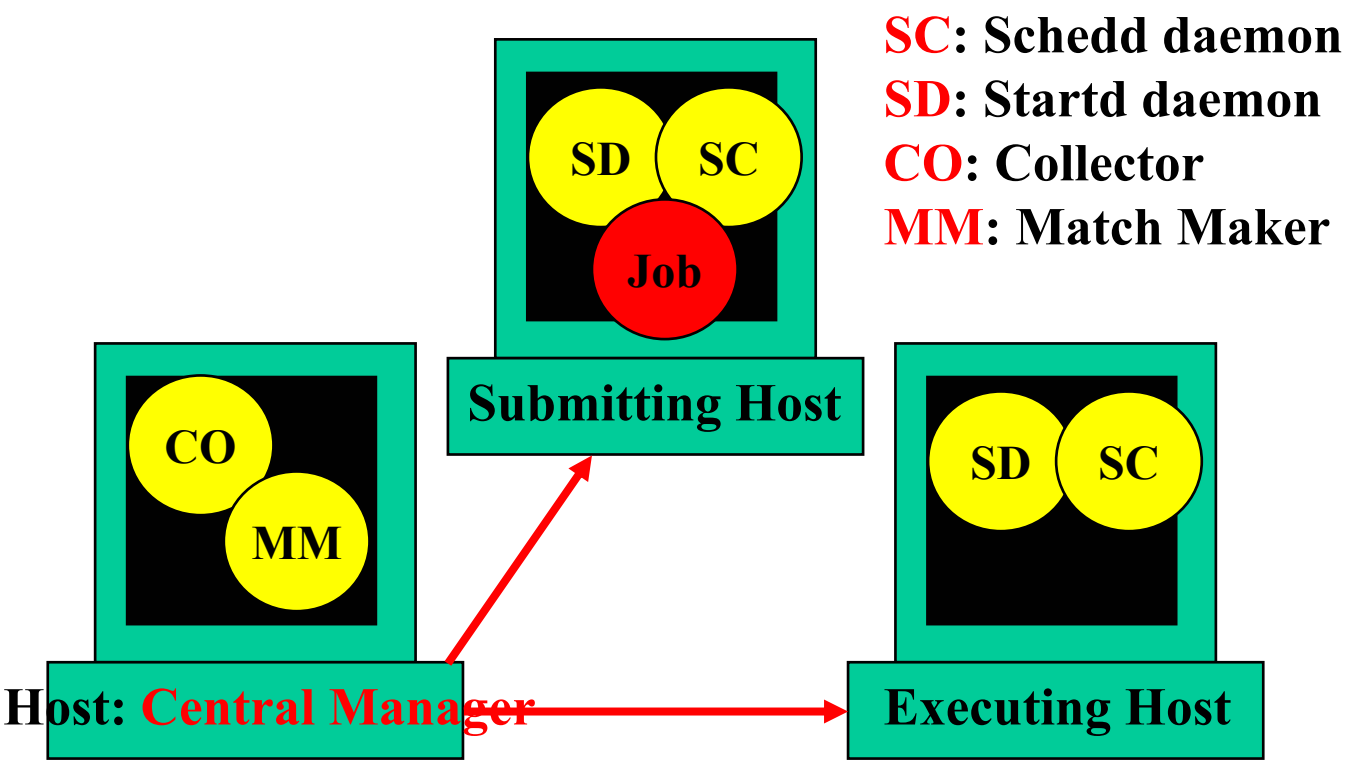
How does Your Personal Condor Pool work?

- You can submit a Condor job on any host of your pool (submit machine)
- The submitted job registers at the central manager with its classads (requirements)
- The CM performs matchmaking and selects an execution host
- The matchmaker notifies the two hosts of their compatibility with respect to the particular job
- The job is transferred to the execution host
- A shadow process is created at the submit machine

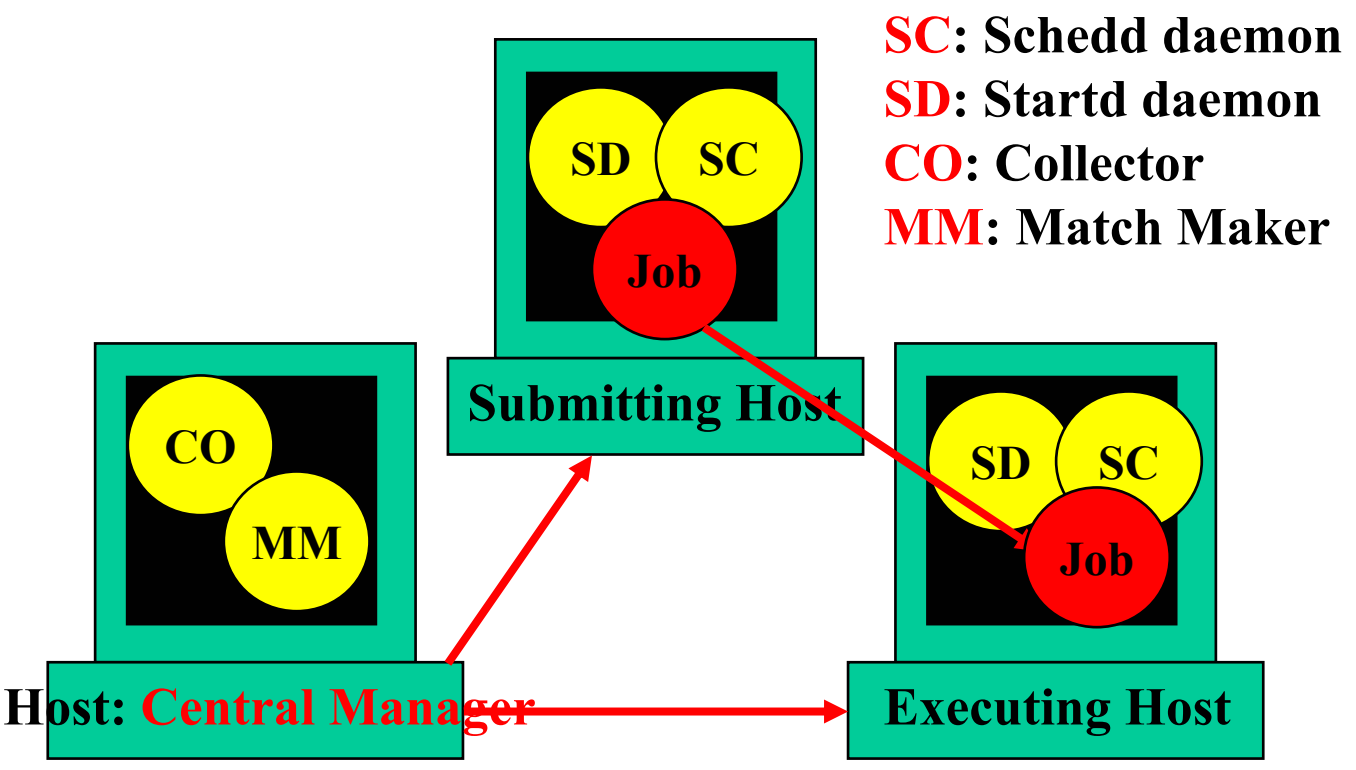
How to execute a Condor job on a cluster pool?



How to execute a Condor job on a cluster pool?



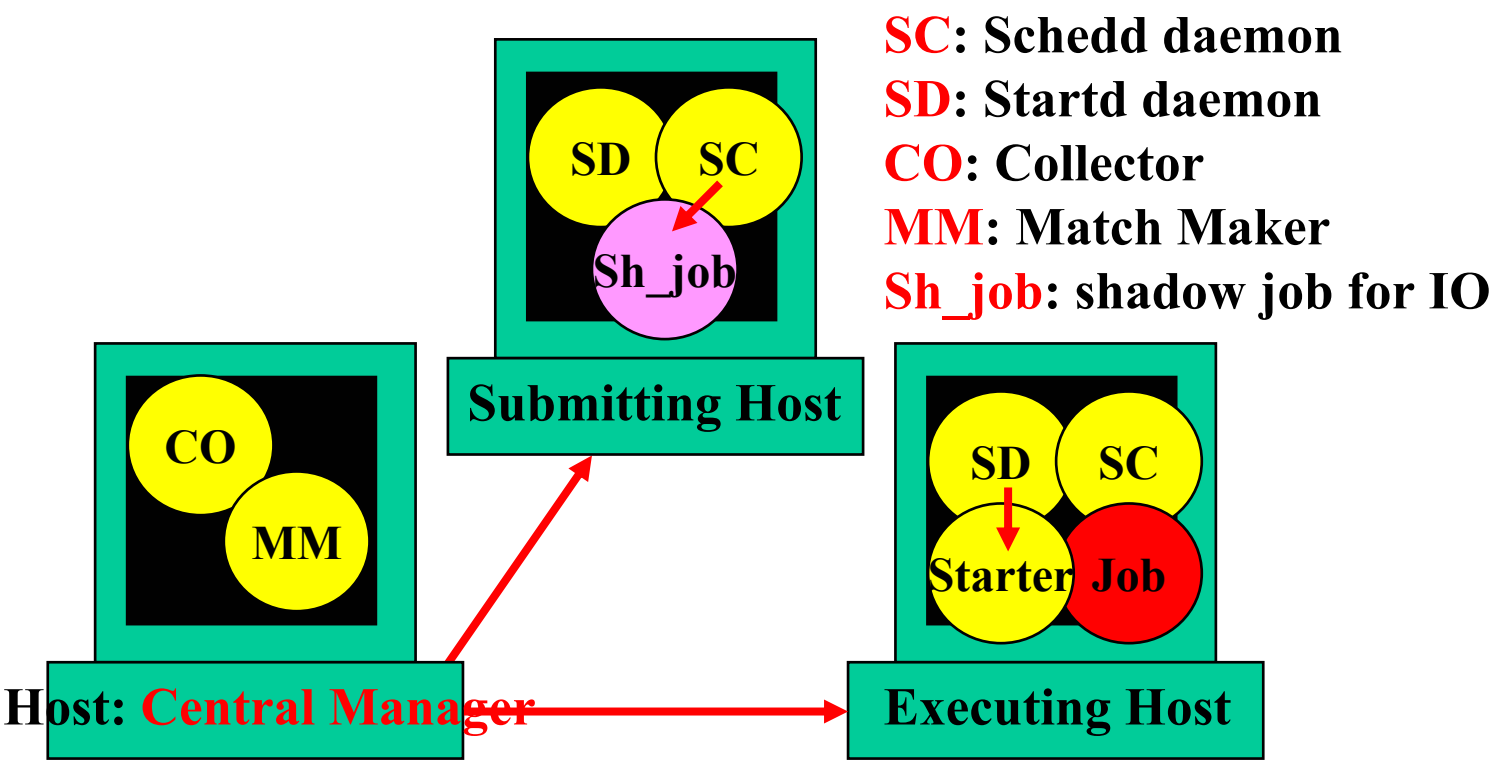
How to execute a Condor job on a cluster pool?



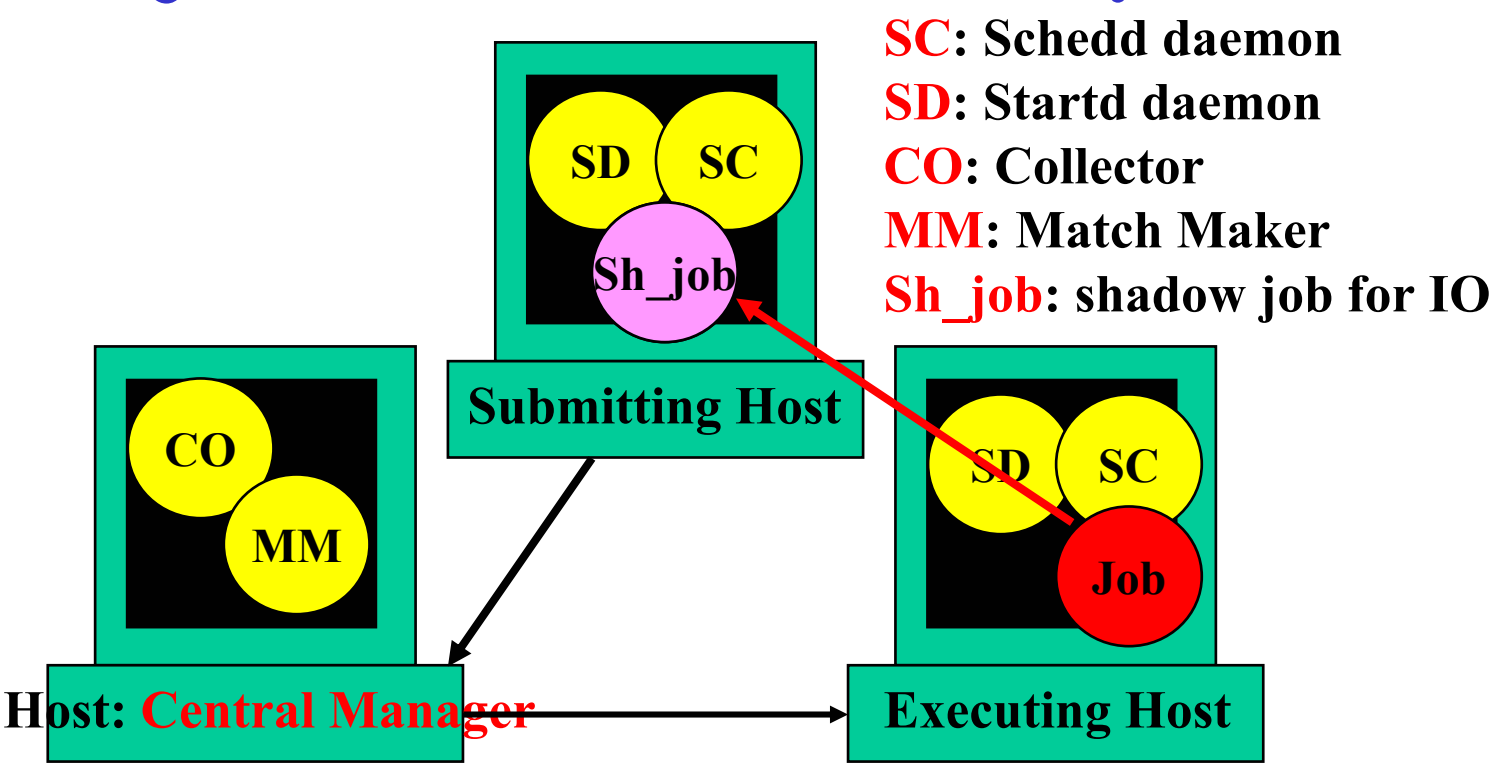
How to execute a Condor job?

- The job is passed to the host that matches to the classads of the job.
- The schedd on the submitting machine will initiate the shadow condor job to execute IO operations locally on the submitting host
- The startd on the executing machine creates a starter process to start the job and monitor it
- The job is running on the selected host, while its shadow job runs on the submitting host. (I/O operation are executed by the shadow job.)

How to execute a Condor job on a cluster pool?



How to execute a Condor job on a cluster pool?



How to create a Condor job?

- Write the program of the job
- Compile it with the Condor library that will instrument your IO activities with the call of shadow condor job's IO activities.
- Create the classad of your job
- Submit the job on any host of your pool to the Condor system, i.e. to the central manager

A simple job description

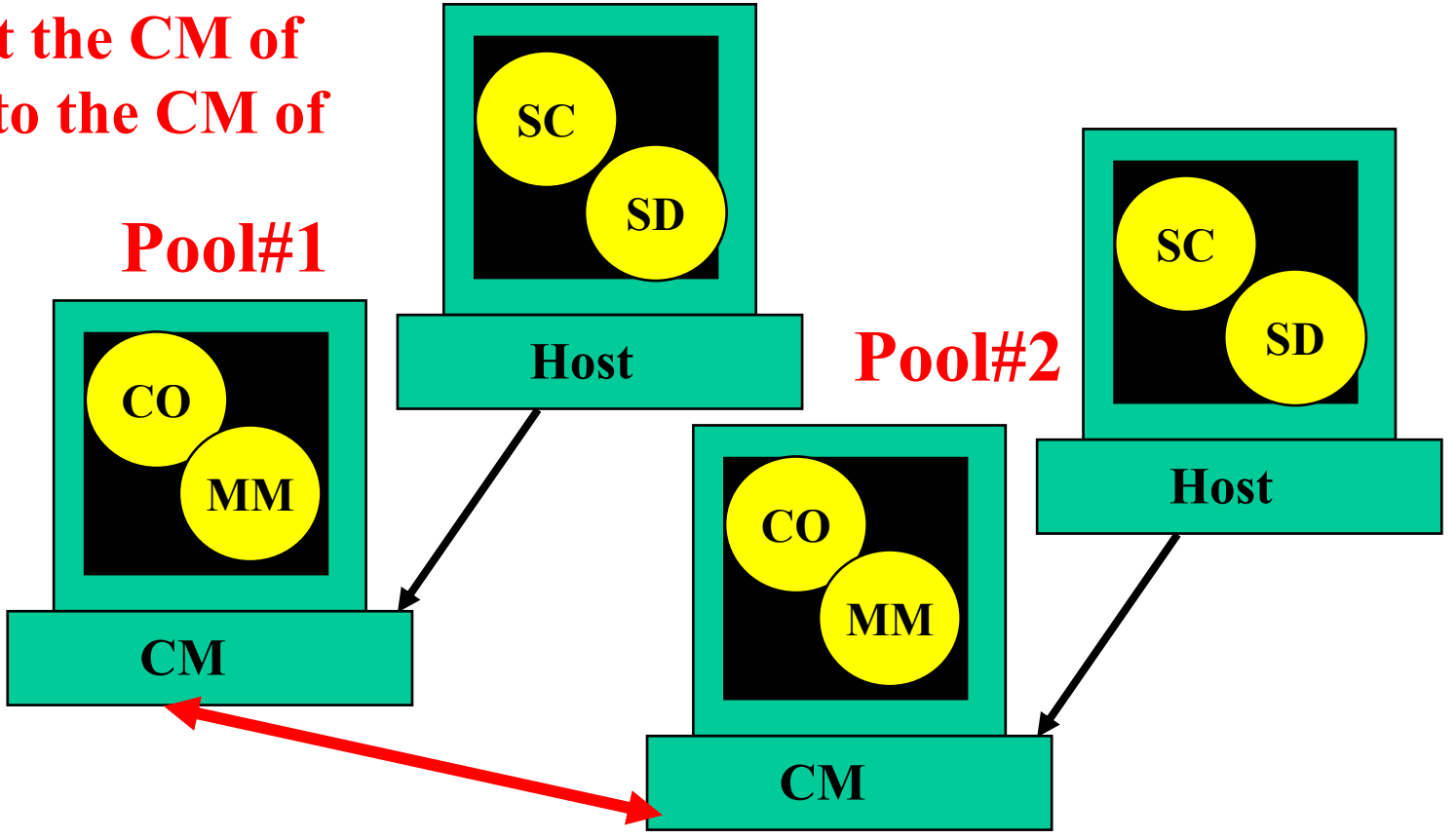
```
universe = vanilla
executable = mathematica
input = in$(Process).dat
output = out$(Process).dat
queue 5
```

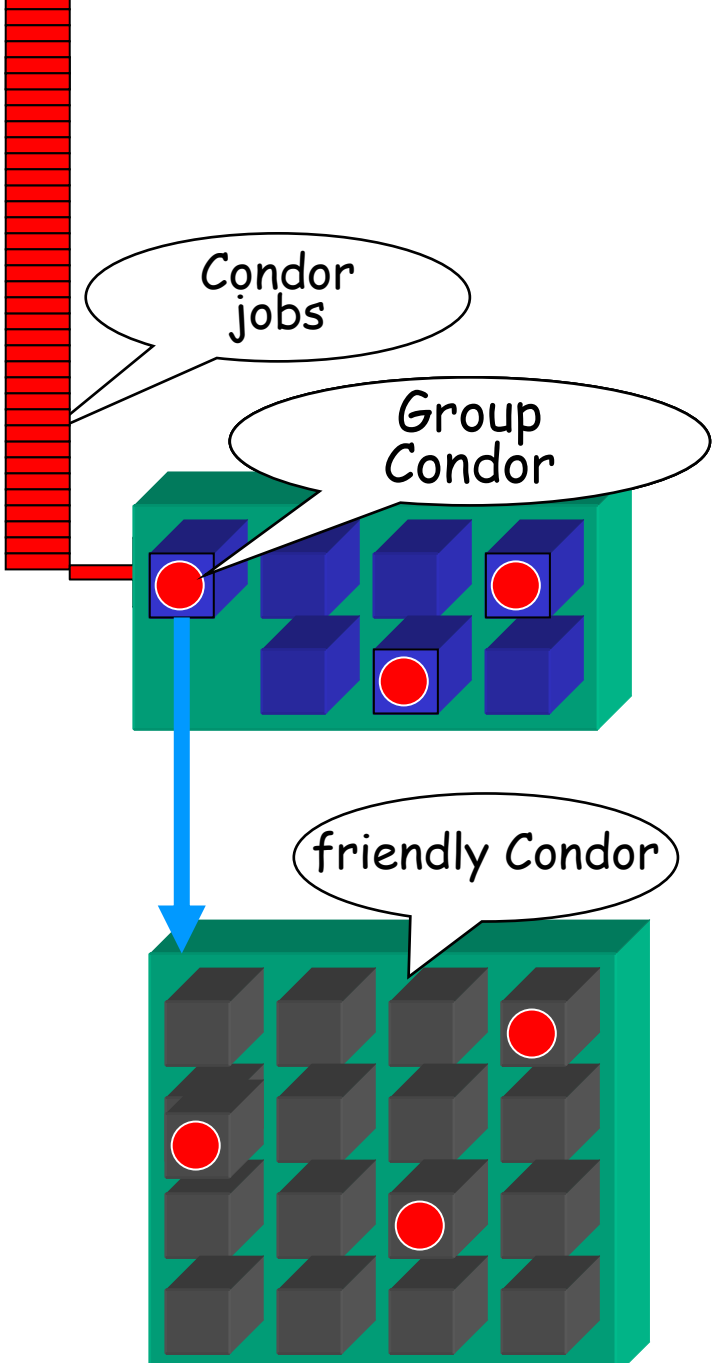
Take advantage of your friends

- Get permission from "friendly" Condor pools to access their resources
- Configure your personal Condor to "flock" to these pools:
  - additional central managers of remote Condor pools can be specified as configuration parameter of schedd.
  - When the local pool doesn't satisfy all its job requests, the schedd will try these remote pools in turn

How to create a friendly Condor pool?

Connect the CM of Pool#1 to the CM of Pool#2





Your schedd daemons see the CM of the other pool as if it was part of your pool

Think **big**.

Go to the **Grid!**

Condor-glide-in

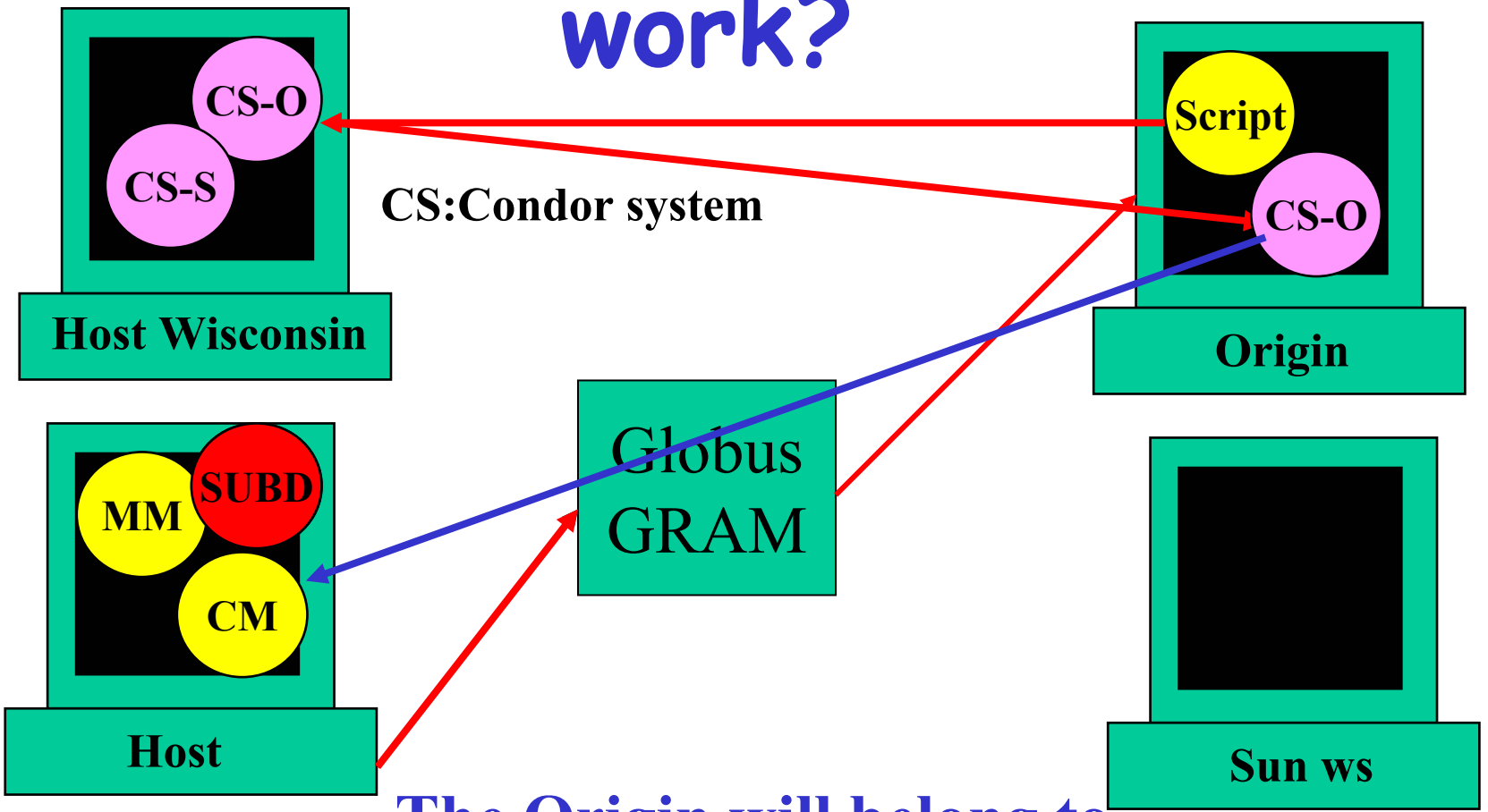
Enable an application to dynamically turn allocated Grid resources into members of your Condor pool even if there is no Condor system on those resources.

- Easy to use on different platforms
- Robust
- Supports SMPs
- Provides uniformity

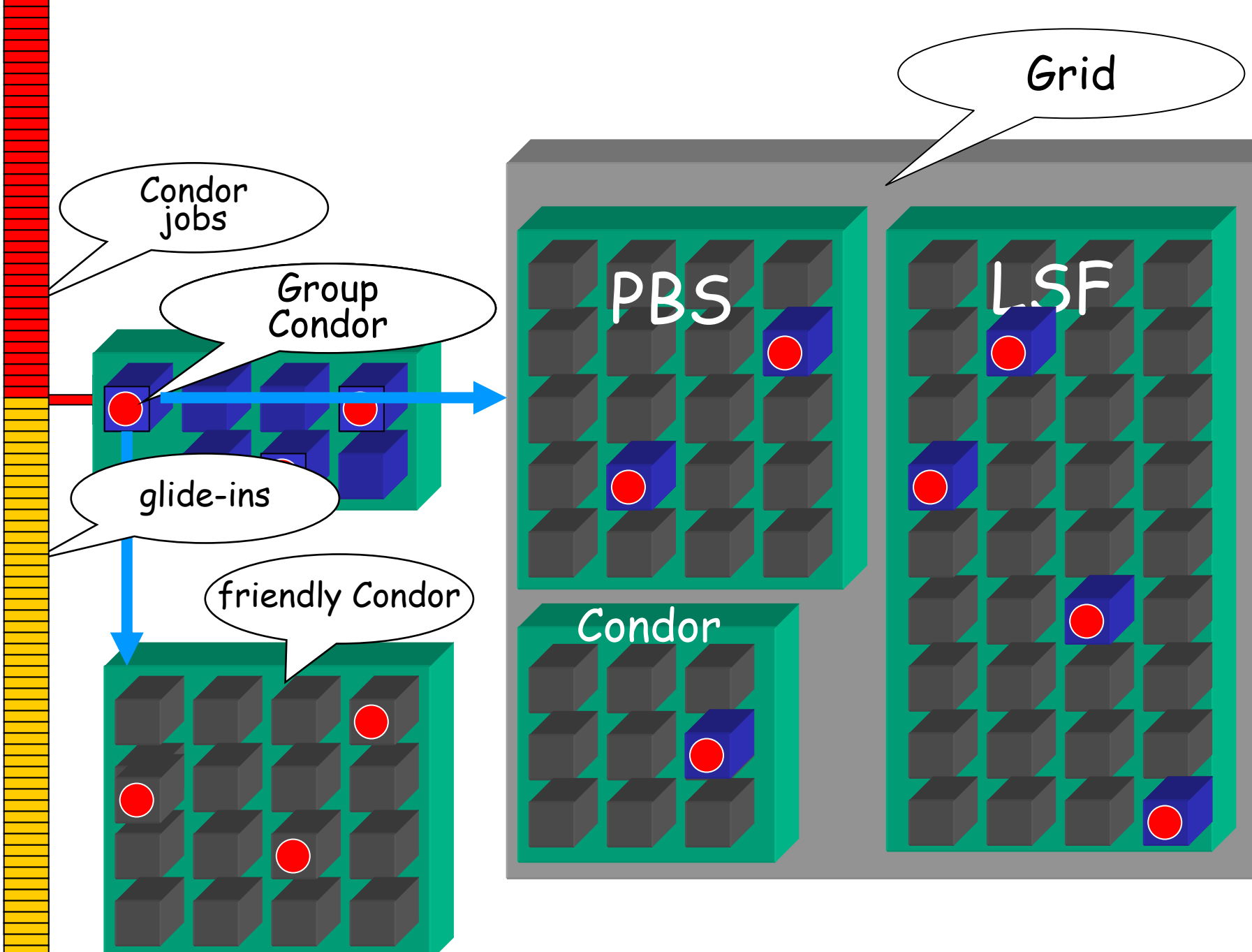
Exploit Grid resources

- Get access (account(s) + certificate(s)) to a "Computational" Grid
- Submit "Grid Universe" Condor-glide-in jobs to your personal Condor
- Manage your glide-ins

How does Condor glide-in work?



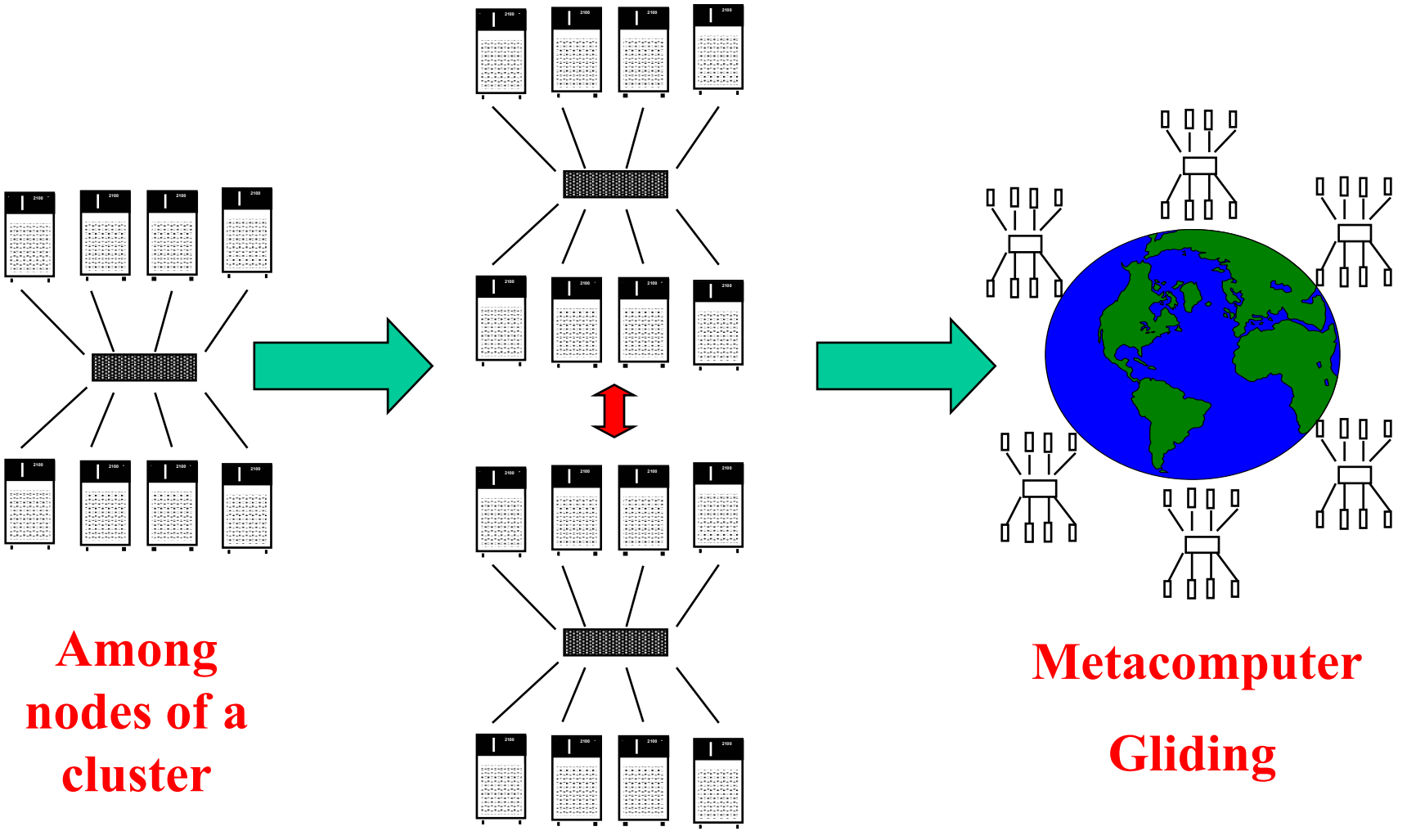
The Origin will belong to your Condor pool


Grid
Condor jobs
Group Condor
glide-ins
friendly Condor
PBS
LSF
Condor

General rule for Condor pools

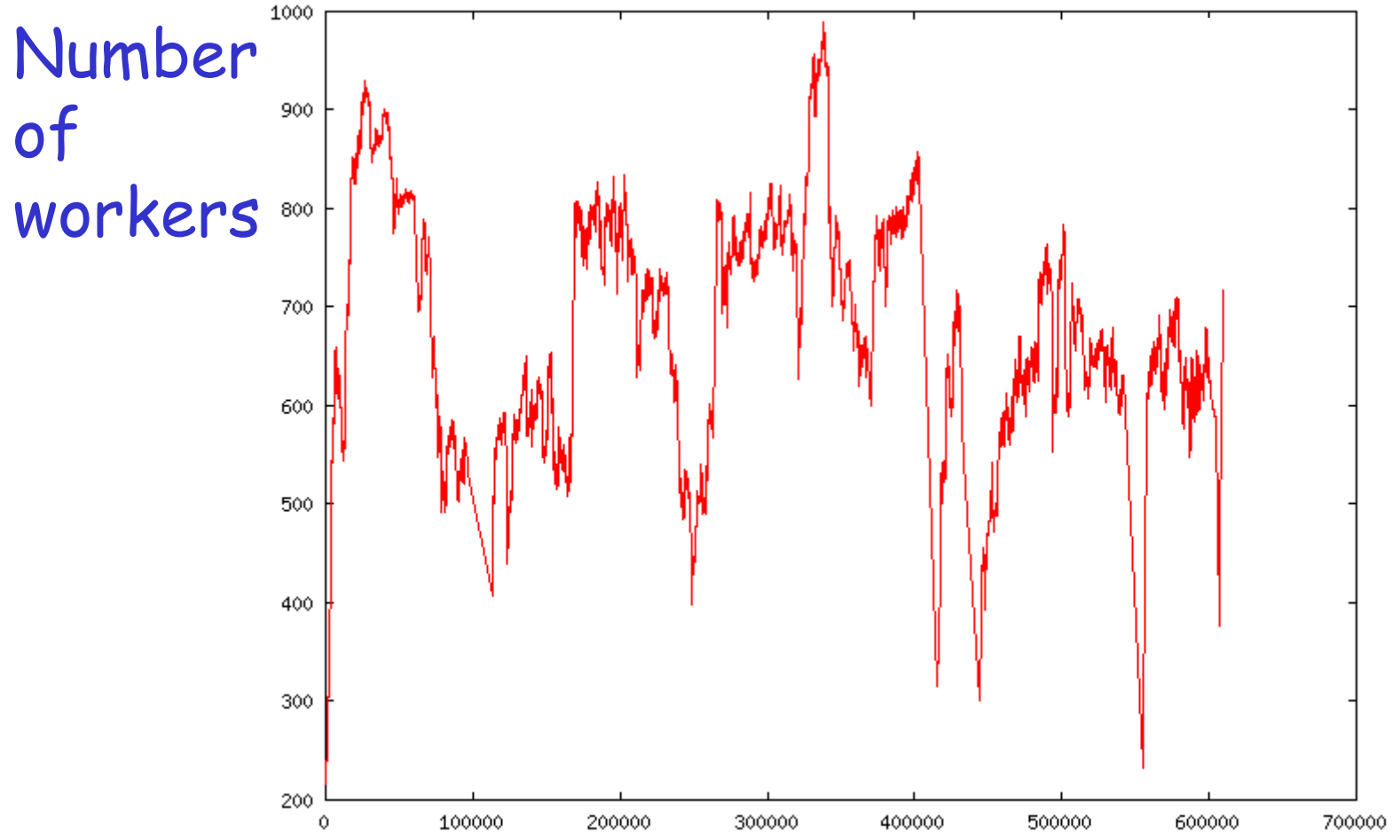
- A Condor pool has a Central Manager
- If you are flocking the Central Manager of the friendly pool becomes visible to your pool and hence hosts in the other pool can be reached
- If you are gliding all the new Condor resources will directly connect to your CM

Three levels of scalability in Condor



NUG30 - Solved!!!

- Solved in 7 days instead of 10.9 years
- The first 600K seconds …



NUG30 Personal Grid …

Managed by **one** Linux box at Wisconsin

Flocking:
-- Condor pool at Wisconsin (500 processors)
-- Condor pool at Georgia Tech (284 Linux boxes)
-- Condor pool at UNM (40 processors)
-- Condor pool at Columbia (16 processors)
-- Condor pool at Northwestern (12 processors)
-- Condor pool at NCSA (65 processors)
-- Condor pool at INFN Italy (54 processors)

Glide-in:
-- Origin 2000 (through LSF) at NCSA. (512 processors)
-- Origin 2000 (through LSF) at Argonne (96 processors)

Hobble-in:
-- Chiba City Linux cluster (through PBS) at Argonne (414 processors).

Solution Characteristics.

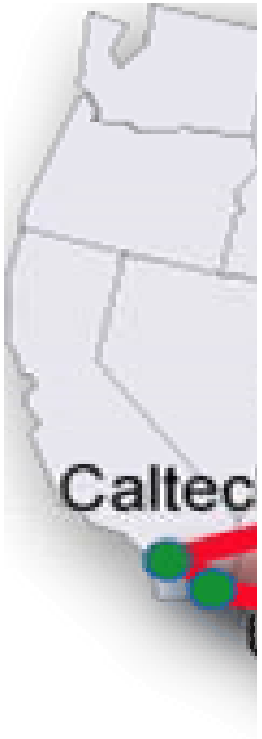
Scientists	4
Wall Clock Time	6:22:04:31
Avg. # CPUs	653
Max. # CPUs	1007
Total CPU Time	Approx. 11 years
Nodes	11,892,208,412
Parallel Efficiency	92%

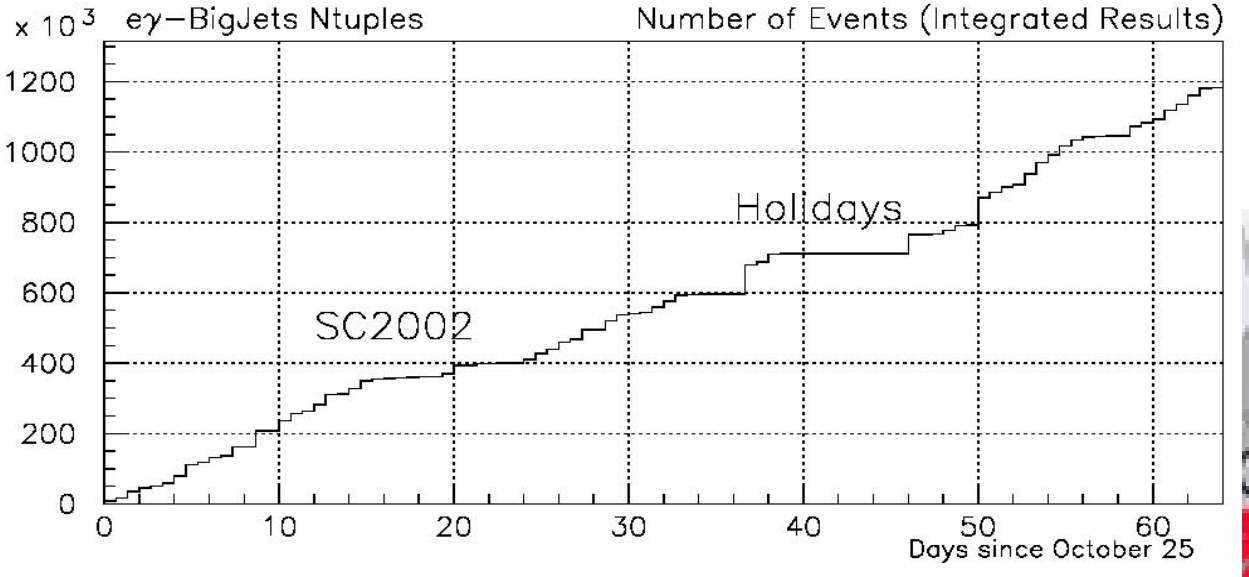
Accomplish an official production request for the CMS collaboration of 100,000 Monte Carlo simulation events with Grid resources.

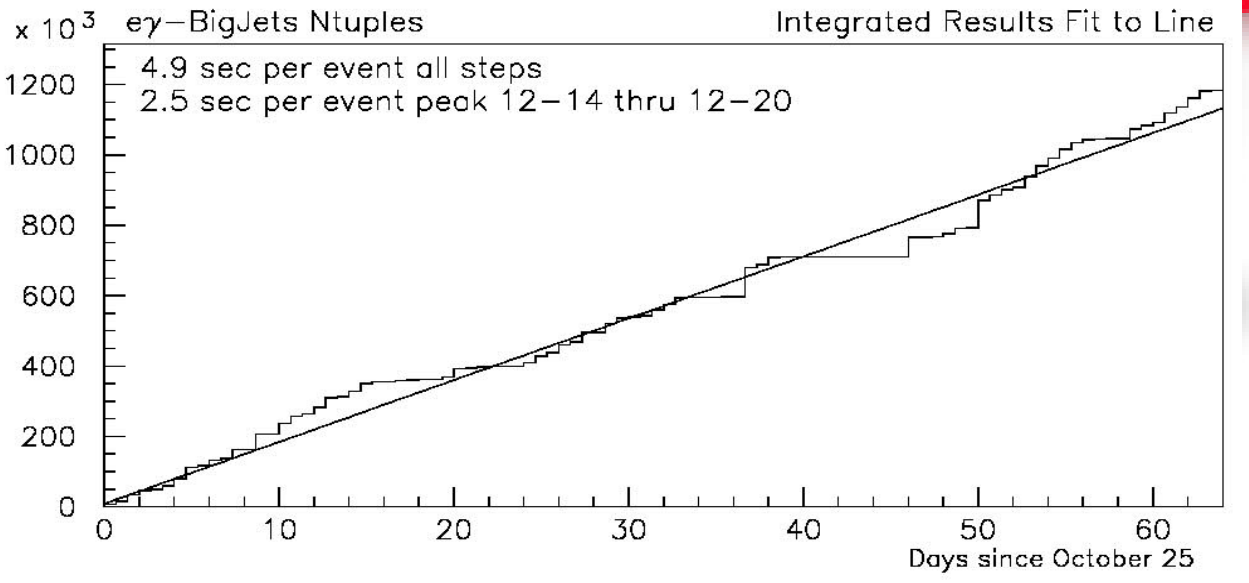
Accomplished!

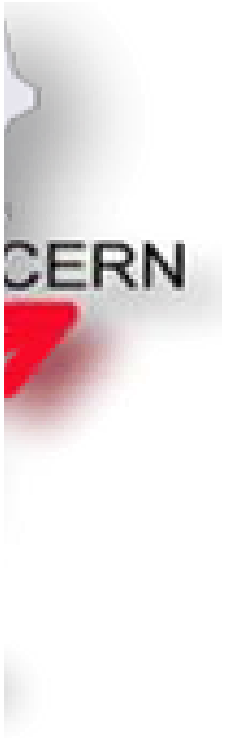
CMS Integration Grid Testbed

ime to process
1 event:
00 sec @ 750 MHz









Thank you