Using Condor
An Introduction

Condor Week 2010

Condor Project
Computer Sciences Department
University of Wisconsin-Madison

The Condor Project (Established '85)

- Research and Development in the Distributed High Throughput Computing field
- Team of ~35 faculty, full time staff and students
  - Face software engineering challenges in a distributed UNIX/Linux/NT environment
  - Are involved in national and international grid collaborations
  - Actively interact with academic and commercial entities and users
  - Maintain and support large distributed production environments
  - Educate and train students

The Condor Team

Some free software produced by the Condor Project

- Condor System
- VDT
- Metronome
- ClassAd Library
- DAGMan
- GCB & CCB
- MW
- NeST / LotMan
- And others… all as open source

High-Throughput Computing

- High-Throughput Computing:
  - Allows for many computational tasks to be done over a long period of time
  - Concerned largely with the number of compute resources that are available to people who wish to use the system
  - Very useful system for researchers and other users who are more concerned with the number of computations they can do over long spans of time than they are with short-burst computations

Condor

What is Condor?

- Classic High-Throughput Computing system
- An integral part of many computational grids around the world

Full featured system

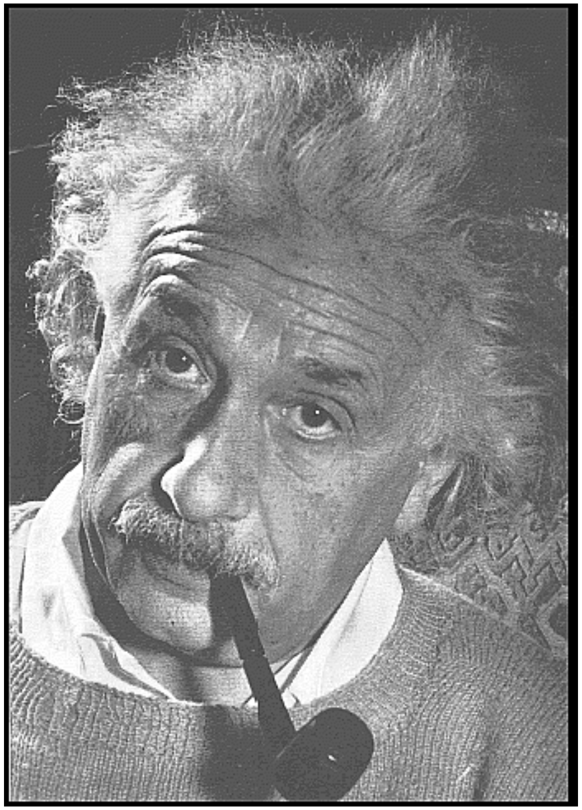
- Flexible scheduling policy engine via ClassAds
  - Preemption, suspension, requirements, preferences, groups, quotas, settable fair-share, system hold…
- Facilities to manage BOTH dedicated CPUs (clusters) and non-dedicated resources (desktops)
- Transparent Checkpoint/Migration for many types of serial jobs
- No shared file-system required
- Federate clusters w/ a wide array of Grid Middleware

Full featured system

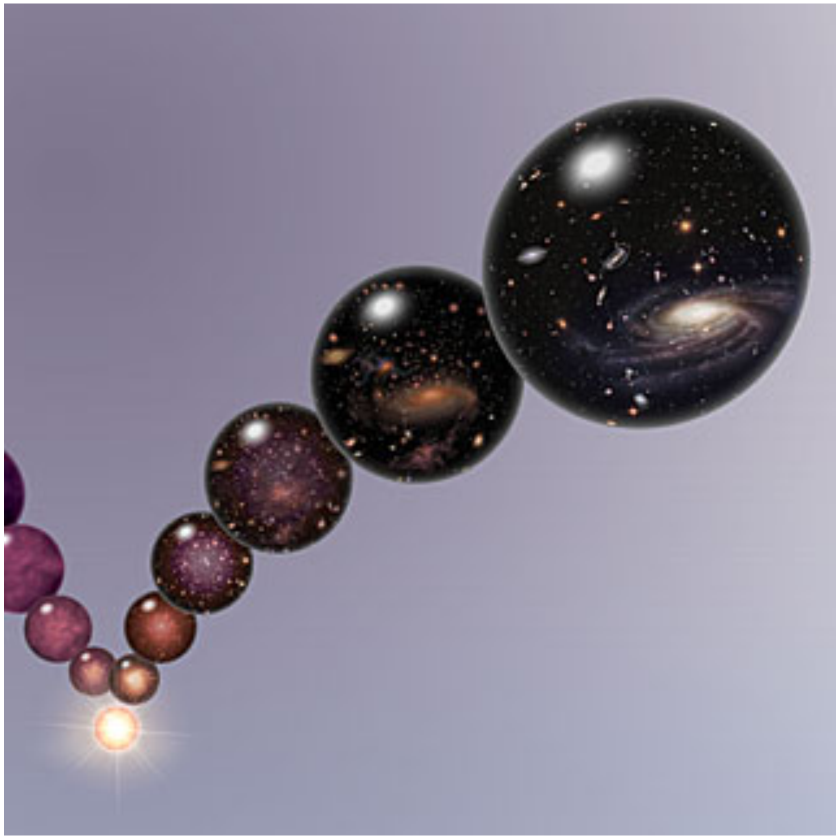
- Workflow management (inter-dependencies)
- Support for many job types – serial, parallel, etc.
- Fault-tolerant: can survive crashes, network outages, no single point of failure.
- Development APIs: via SOAP / web services, DRMAA (C), Perl package, GAHP, flexible command-line tools, MW
- Platforms:
  - Linux i386 / IA64 / X86-64 / PPC
  - Windows XP / Vista / 7
  - MacOS X
  - Solaris
  - HP-UX
  - AIX

The Problem

Our esteemed scientist, while visiting Madison, needs to run a "small" simulation.



The Simulation



Run a simulation of the evolution of the cosmos with various properties

The Simulation Details

Varying values for the value of:

- G (the gravitational constant): 100 values
- $\mathcal{R}_{\mu\nu}$ (the cosmological constant): 100 values
- c (the speed of light): 100 values
- 100 × 100 × 100 = 1,000,000 jobs

Running the Simulation

Each simulation:

- Requires up to 4GB of RAM
- Requires 20MB of input
- Requires 2 – 500 hours of computing time
- Produces up to 10GB of output

Estimated total:

- 15,000,000 hours!
- 1,700 compute YEARS
- 10 Petabytes of output

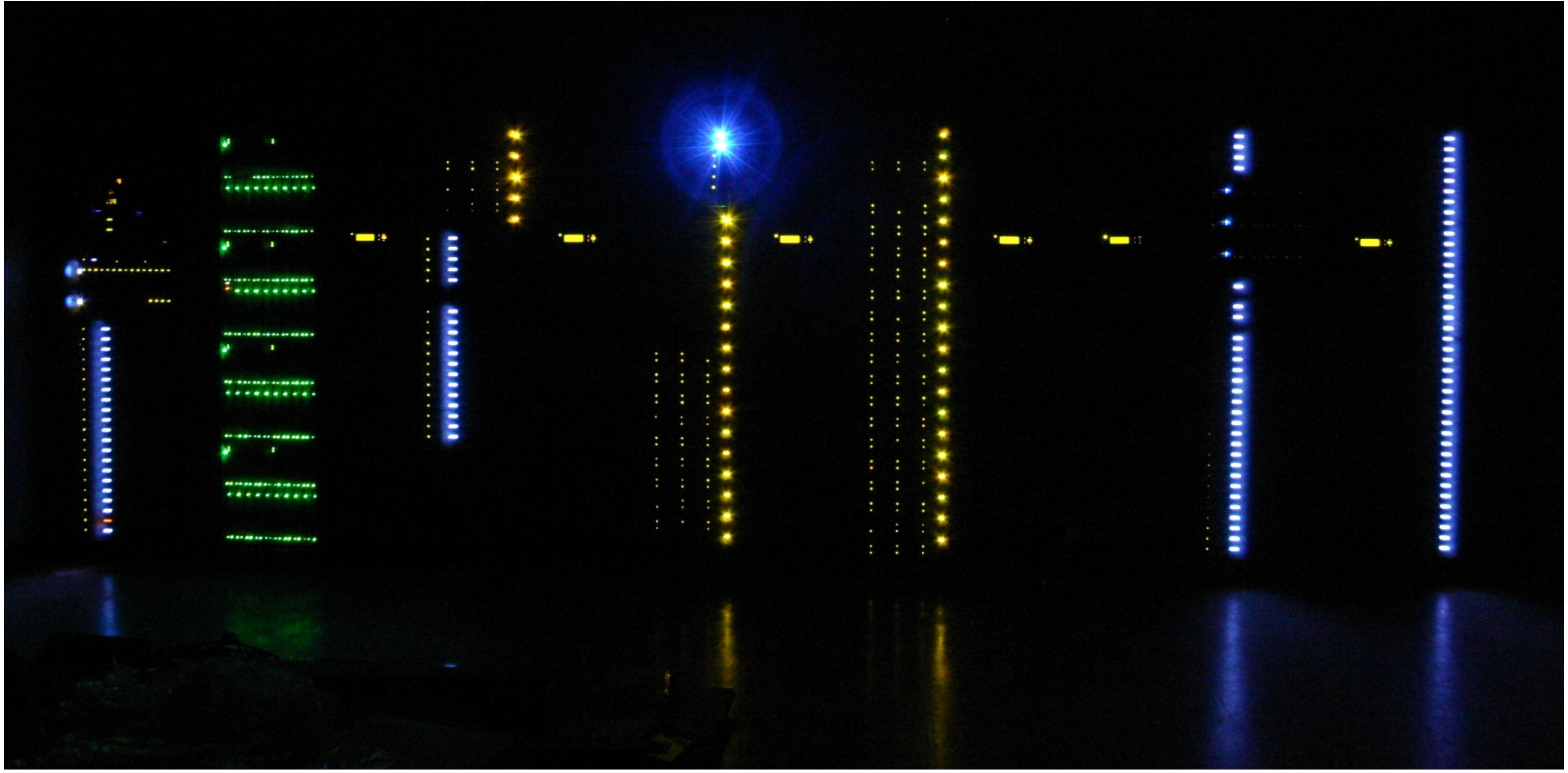
NSF won't fund the Blue Gene that I requested.

While sharing a beverage with some colleagues, Carl asks "Have you tried Condor? It's free, available for you to use, and you can use our CHTC pool. Condor's been used to run billions and billions of jobs."

Condor and CHTC

- Center for High Throughput Computing
  - Approved in August 2006
  - Numerous resources at its disposal to keep up with the computational needs of UW Madison
  - These resources are being funded by:
    - National Institute of Health (NIH)
    - Department of Energy (DOE)
    - National Science Foundation (NSF)
    - Various grants from the University itself

B240
One of the CTHC Clusters



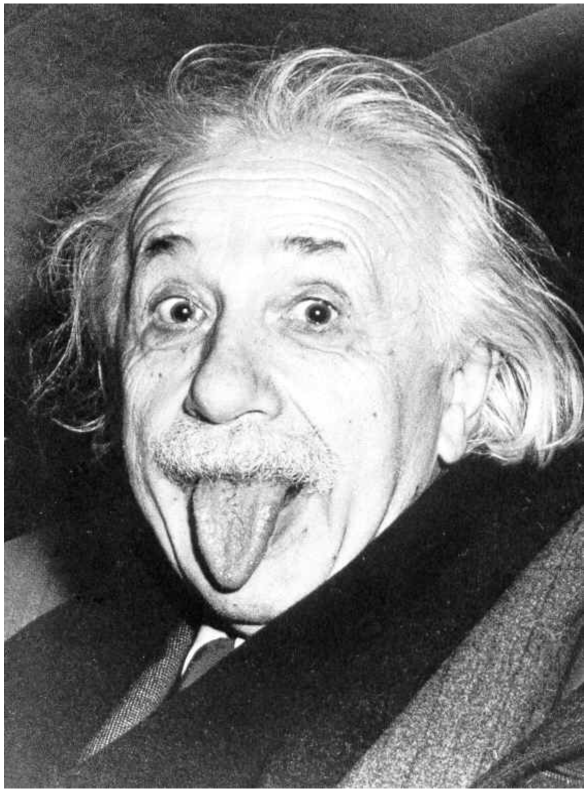
But... Will my jobs be safe?

- No worries!!
  - Condor will take care of your jobs for you
  - Jobs are queued in a safe way
    - More details later
  - Condor will make sure that your jobs run, return output, etc.
    - You can even specify what defines "OK"
  - Like money in the (FDIC insured) bank

Condor will ...

- Keep an eye on your jobs and will keep you posted on their progress
- Implement your policy on the execution order of the jobs
- Keep a log of your job activities
- Add fault tolerance to your jobs
- Implement your policy on when the jobs can run on your workstation

Condor Doesn't Play Dice with My Universes!



Definitions

- Job
  - The Condor representation of your work (next slide)
- Machine
  - The Condor representation of computers and that can perform the work
- ClassAd
  - Condor's internal data representation
- Match Making
  - Matching a job with a machine "Resource"
- Central Manager
  - Central repository for the whole pool
  - Performs job / machine matching, etc.

Job

- Condor's quanta of work
  - Like a UNIX process
  - Can be an element of a workflow

Jobs Have Wants & Needs

- Jobs state their requirements and preferencs:
  - Requirements:
    - I **require** a Linux/x86 platform
  - Preferences ("Rank"):
    - I **prefer** the machine with the most memory
    - I **prefer** a machine in the chemistry department

Machines Do Too!

- Machines specify:
  - Requirements:
    - **Require** that jobs run only when there is no keyboard activity
    - **Never** run jobs belonging to Dr. Heisenberg
  - Preferences ("Rank"):
    - I **prefer** to run Albert's jobs
  - Custom Attributes:
    - I am a machine in the physics department

Condor ClassAds

What is a ClassAd?

- Condor's internal data representation
  - Similar to a classified ad in a paper
    - Their namesake
  - Represent an object & its attributes
    - Usually many attributes
  - Can also describe what an object matches with

ClassAd Types

- Condor has many types of ClassAds
  - A "Job Ad" represents your job to Condor
  - A "Machine Ad" represents the various compute resources in your Condor pool
  - Others represent other pieces of your Condor pool

ClassAds Explained

- ClassAds can contain a lot of details
  - The job's executable is "cosmos"
  - The machine's load average is 5.6
- ClassAds can specify requirements
  - My job requires a machine with Linux
- ClassAds can specify preferences
  - This machine prefers to run jobs from the physics group

ClassAd Structure

- ClassAds are:
  - semi-structured
  - user-extensible
  - schema-free
- ClassAd contents:
  - Attribute = Value
  - Attribute = Expression

The Pet Exchange

Pet Ad

```
Type   = “Dog”
Color = “Brown”
Price = 75
Sex = "Male"
AgeWeeks = 8
Breed = "Saint Bernard"
Size = "Very Large"
Weight = 27
```

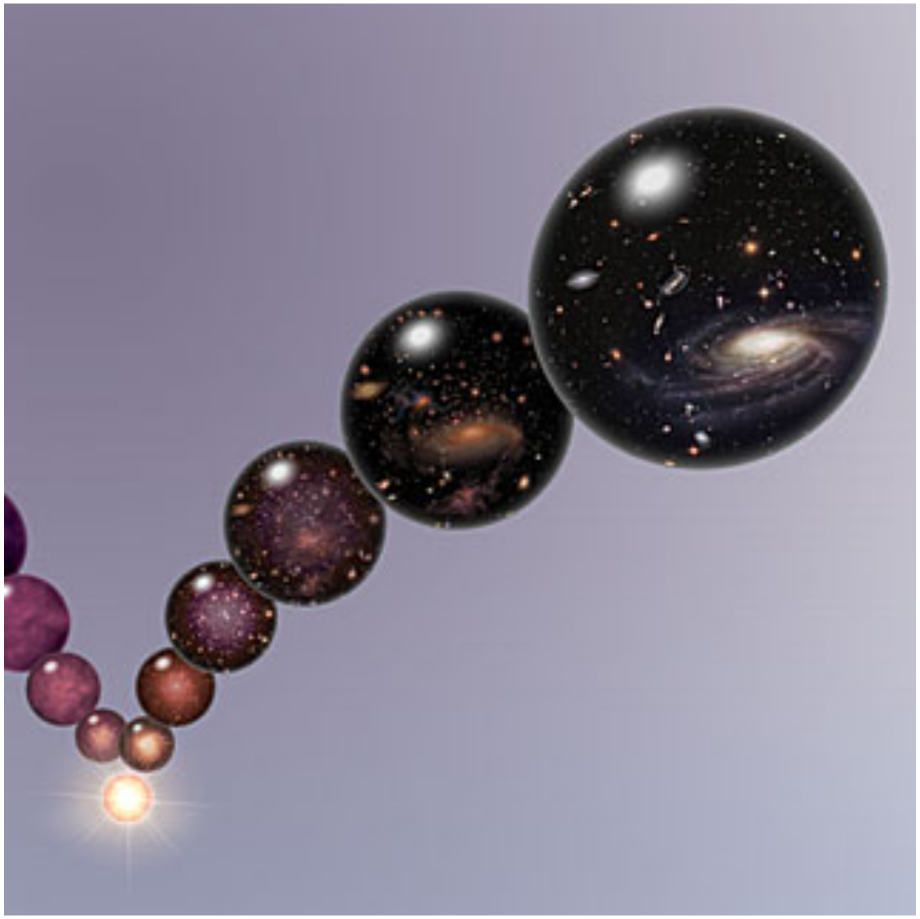
Buyer Ad

```
. . .
Requirements =
  (Type == “Dog”)      &&
  (Price <= 100)       &&
  ( Size == "Large" ||
    Size == "Very Large" )
Rank =
  (Breed == "Saint Bernard")
. . .
```

The Magic of Matchmaking

- The Condor "match maker" matches Job Ads with Machine Ads
  - Requirements:
    - Enforces both machine AND job requirements expressions
  - Preferences:
    - Considers both job AND machine rank expressions
  - Priorities:
    - Takes into account user and group priorities

Back to our Simulation..



Getting Started: Submitting Jobs to Condor

- Get access to submit host
- Choose a "Universe" for your job
- Make your job "batch-ready"
  - Includes making your data available to your job
- Create a *submit description* file
- Run *condor_submit* to put your job(s) in the queue
- Relax while Condor manages and watches over your job(s)

1. Access to CHTC (UW Specific)

- Send email to **`chtc@cs.wisc.edu`**
- An account will be set up for you
- ssh into our submit head node:
  - From UNIX / Linux:
    - **`ssh einstein@submit.chtc.wisc.edu`**
  - From Windows:
    - Install Putty or similar SSH client
    - Use Putty to ssh into **`submit.cht.wisc.edu`**

If You're not at UW…

- Work with your Condor Administrator to get access
- Login to your Condor submit host…

2. Choose the "Universe"

- Controls how Condor handles jobs
- Condors many universes include:
  - Vanilla
  - Standard
  - Grid
  - Java
  - Parallel
  - VM

Using the Vanilla Universe

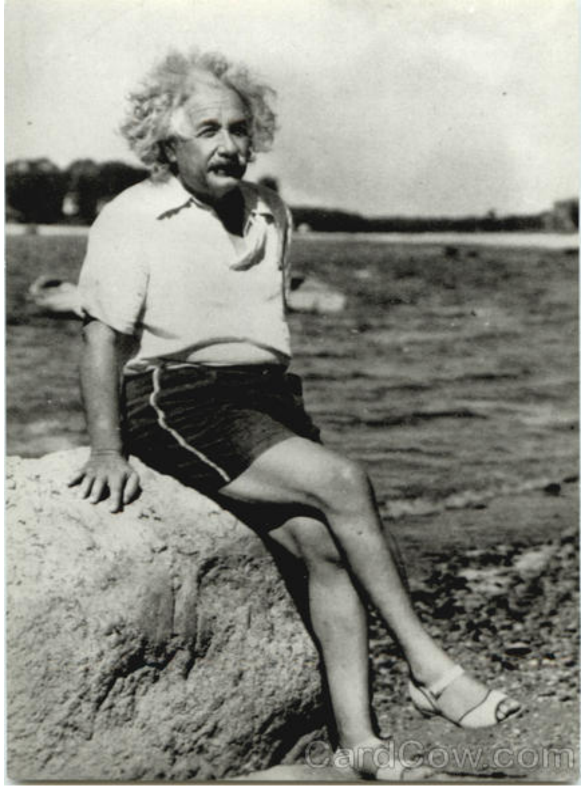
- Allows running almost any "serial" job
- Provides automatic file transfer, etc.
- Like vanilla ice cream
  - Can be used in just about any situation

Introducing:
Todd's Private Universe

3. Make your job batch-ready

Must be able to run in the background

- No interactive input
- No GUI/window clicks
  - We don't need no stinkin' mouse!
- No music ;^)



Batch-Ready: Input & Output

- Job can still use **STDIN**, **STDOUT**, and **STDERR** (the keyboard and the screen), but files are used for these instead of the actual devices
- Similar to UNIX shell:

```
$ ./myprogram <input.txt >output.txt
```

Make your Data Available

- Condor can:
  - Transfer your data files to your job
  - Transfer your results files back from your job
- You need to:
  - Put your data files in a place where Condor can access them

4. Create a Submit Description File

- A plain ASCII text file
- Condor does ***not*** care about file extensions
  - Most people use ".sub" or ".submit", though
- Tells Condor about your job:
  - Which executable, universe, input, output and error files to use, command-line arguments, environment variables, any special requirements or preferences (more on this later)
- Can describe many jobs at once (a “cluster”), each with different input, arguments, output, etc.

Input, output & error files

- Controlled by submit file settings
- You can define the job's standard input, standard output and standard error:
  - Read job's standard input from "input_file":
    - `Input    = input_file`
    - Shell equivalent: `$ program <input_file`
  - Write job's standard ouput to "output_file":
    - `Output   = output_file`
    - Shell equivalent: `$ program >output_file`
  - Write job's standard error to "error_file":
    - `Error    = error_file`
    - Shell equivalent: `$ program 2>error_file`

Simple Submit File

```
# Simple condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
Executable = cosmos
Output     = cosmos.out
Input      = cosmos.in
Log        = cosmos.log
Queue
```

- Job's executable
- Job's STDOUT
- Job's STDIN
- Log the job's activities
- Put the job in the queue!

Logging your Job's Activities

- Create a log of job events
- Add to submit description file:

```
log = cosmos.log
```

- The Life Story of a Job
  - Shows all events in the life of a job
- Always have a log file

Sample Condor User Log

```
000 (0101.000.000) 05/25 19:10:03 Job submitted from host:
<128.105.146.14:1816>

...

001 (0101.000.000) 05/25 19:12:17 Job executing on host:
<128.105.146.14:1026>

...

005 (0101.000.000) 05/25 19:13:06 Job terminated.

    (1) Normal termination (return value 0)

...
```

5. Submit the Job to Condor

- Run *condor_submit*:
  - Provide the name of the submit file you have created on the command line:

```
$ condor_submit cosmos.submit
```

  - *condor_submit*:
    - Parses the submit file, checks for errors
    - Creates one or more job ad(s) that describes your job(s)
    - Hands the job ad(s) off to the schedd

The Job Ad

Example:

```
MyType       = "Job"                     String
TargetType   = "Machine"
ClusterId    = 1                         Number
ProcId       = 0
IsPhysics    = True                      Boolean
Owner        = "einstein"
Cmd          = "cosmos“                  Boolean
Requirements = (Arch == "INTEL")         Expression
…
```

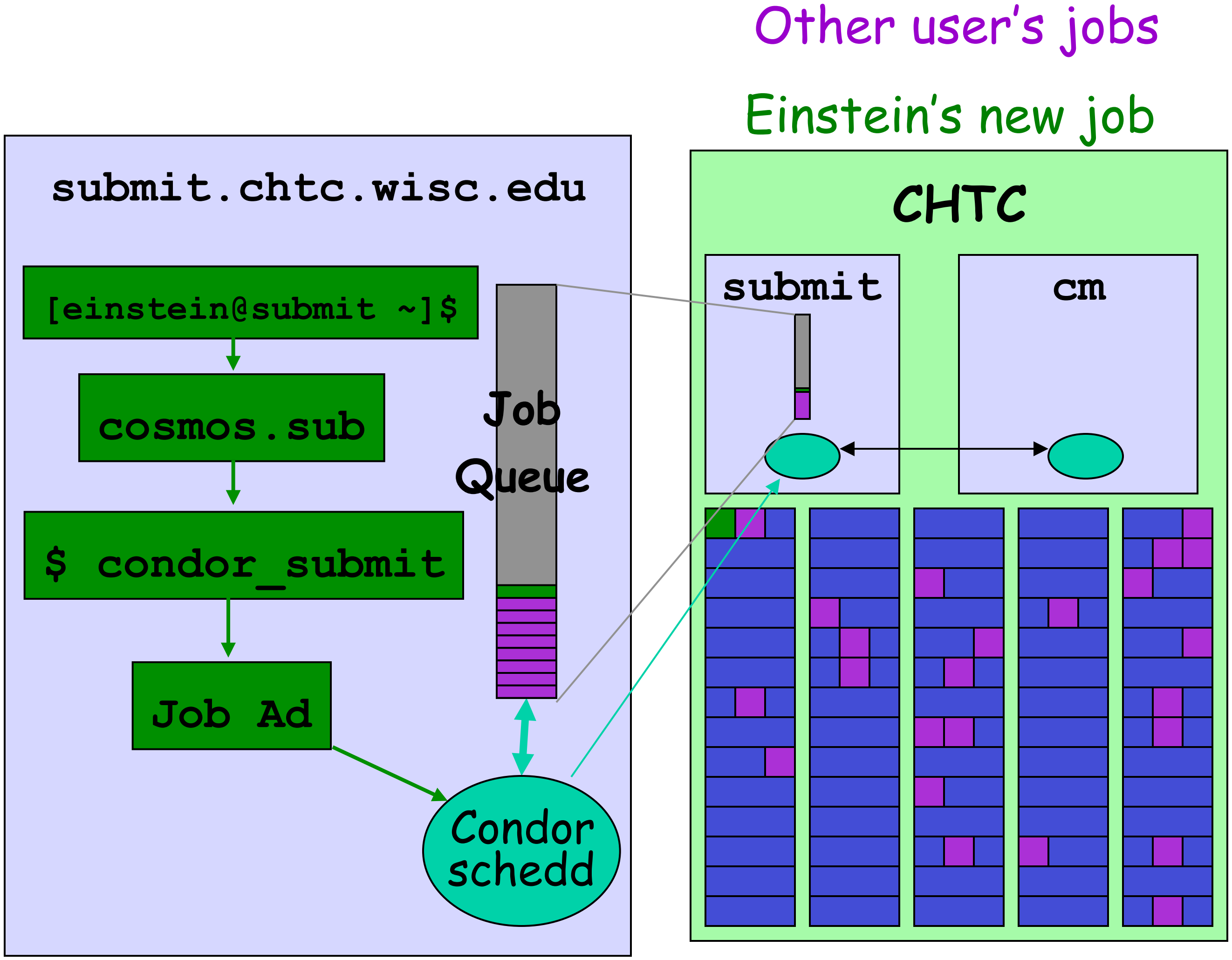
Submitting The Job

```
[einstein@submit ~]$ condor_submit cosmos.submit-file
Submitting job(s).
2 job(s) submitted to cluster 100.
[einstein@submit ~]$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:51883> : submit.chtc.wisc.edu
 ID          OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1.0          sagan           7/22 14:19 172+21:28:36 H  0   22.0 checkprogress.cron
2.0          heisenberg      1/13 13:59   0+00:00:00 I  0   0.0  env
3.0          hawking         1/15 19:18   0+04:29:33 H  0   0.0  script.sh
4.0          hawking         1/15 19:33   0+00:00:00 H  0   0.0  script.sh
5.0          hawking         1/15 19:33   0+00:00:00 H  0   0.0  script.sh
6.0          hawking         1/15 19:34   0+00:00:00 H  0   0.0  script.sh
...
96.0         bohr            4/5  13:46   0+00:00:00 I  0   0.0  c2b_dops.sh
97.0         bohr            4/5  13:46   0+00:00:00 I  0   0.0  c2b_dops.sh
98.0         bohr            4/5  13:52   0+00:00:00 I  0   0.0  c2b_dopc.sh
99.0         bohr            4/5  13:52   0+00:00:00 I  0   0.0  c2b_dopc.sh
100.0        einstein        4/5  13:55   0+00:00:00 I  0   0.0  cosmos
557 jobs; 402 idle, 145 running, 10 held
[einstein@submit ~]$
```

The Job Queue

- condor_submit sends your job's ClassAd(s) to the schedd
- The schedd (more details later):
  - Manages the local job queue
  - Stores the job in the job queue
    - Atomic operation, two-phase commit
    - "Like money in the (FDIC insured) bank"
- View the queue with *condor_q*

CHTC Condor Pool



Condor File Transfer Lists

- `Transfer_Input_Files`
  - List of files that you want Condor to transfer to the execute machine
- `Transfer_Output_Files`
  - List of files that you want Condor to transfer from the execute machine
  - If not specified, Condor will transfer back all "new" files in the execute directory

Condor File Transfer Controls

- **ShouldTransferFiles**
  - **YES:** Always transfer files to execution site
  - **NO:** Always rely on a shared filesystem
  - **IF_NEEDED:** Condor will automatically transfer the files if the submit and execute machine are not in the same **FileSystemDomain** (Use shared file system if available)
- **When_To_Transfer_Output**
  - **ON_EXIT:** Transfer the job's output files back to the submitting machine only when the job completes
  - **ON_EXIT_OR_EVICT:** Like above, but also when the job is evicted

Simple File Transfer Example

```
# Example submit file using file transfer
Universe                 = vanilla
Executable                 = cosmos
Log                      = cosmos.log
ShouldTransferFiles      = IF_NEEDED
Transfer_input_files       = cosmos.dat
Transfer_output_files    = results.dat
When_To_Transfer_Output  = ON_EXIT
Queue
```

Need Command Line Args?

- You can specify command line arguments to pass to your program with the **arguments** directive:

```
arguments = -arg1 -arg2 foo
```

```
# Example submit file with command line arguments
Universe   = vanilla
Executable = cosmos
Arguments  = -c 299792458 –G 6.67300e-112 -f cosmos.dat
log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
Error      = cosmos.err
Queue
```

InitialDir

- With `InitialDir`, you can give jobs a directory with respect to file input and output.
- Also provides a directory (on the **submit** machine) for the user log, when a full path is not specified.
- Executable is **not** relative to `InitialDir`

```
# Example submit input file with InitialDir
Universe    = vanilla
InitialDir = /home/einstein/cosmos/run
Executable = cosmos                ·NOT Relative to InitialDir
Log         = cosmos.log           ·Is Relative to InitialDir
Input       = cosmos.in            ·Is Relative to InitialDir
Output      = cosmos.out           ·Is Relative to InitialDir
Error       = cosmos.err           ·Is Relative to InitialDir
TransferInputFiles=cosmos.dat      ·Is Relative to InitialDir
Arguments   = -f cosmos.dat
Queue
```

Need More Feedback?

- Condor sends email about job events to the submitting user
- Specify "notification" in your submit file to control which events:

```
Notification = complete
Notification = never
Notification = error
Notification = always
```

Jobs, Clusters, and Processes

- If your submit file describes multiple jobs, we call this a "cluster"
- Each cluster has a "cluster number"
  - The cluster number is unique to the schedd
- Each individual job in a cluster is called a "process"
  - Process numbers always start at zero
- A Condor "Job ID" is the cluster number, a period, and the process number (i.e. 2.1)
  - A cluster can have a single process
    - Job ID = 20.0 · Cluster 20, process 0
  - Or, a cluster can have more than one process
    - Job IDs: 21.0, 21.1, 21.2 · Cluster 21, process 0, 1, 2

Submit File for a Cluster

```
# Example submit file for a cluster of 2 jobs
# with separate input, output, error and log files
Universe   = vanilla
Executable = cosmos

Arguments  = -f cosmos_0.dat
log        = cosmos_0.log
Input      = cosmos_0.in
Output     = cosmos_0.out
Error      = cosmos_0.err
Queue                      ·Job 102.0 (cluster 102, process 0)

Arguments  = -f cosmos_1.dat
log        = cosmos_1.log
Input      = cosmos_1.in
Output     = cosmos_1.out
Error      = cosmos_1.err
Queue                      ·Job 102.1 (cluster 102, process 1)
```

Submitting The Job

```
[einstein@submit ~]$ condor_submit cosmos.submit-file
Submitting job(s).
2 job(s) submitted to cluster 101.
[einstein@submit ~]$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:51883> : submit.chtc.wisc.edu
 ID       OWNER         SUBMITTED     RUN_TIME ST PRI SIZE CMD
1.0       sagan        7/22 14:19 172+21:28:36 H  0   22.0 checkprogress.cron
2.0       heisenberg   1/13 13:59   0+00:00:00 I  0   0.0  env
3.0       hawking      1/15 19:18   0+04:29:33 H  0   0.0  script.sh
4.0       hawking      1/15 19:33   0+00:00:00 H  0   0.0  script.sh
5.0       hawking      1/15 19:33   0+00:00:00 H  0   0.0  script.sh
6.0       hawking      1/15 19:34   0+00:00:00 H  0   0.0  script.sh
...
102.0     einstein     4/5  13:55   0+00:00:00 I  0   0.0  cosmos -f cosmos.dat
102.1     einstein     4/5  13:55   0+00:00:00 I  0   0.0  cosmos -f cosmos.dat

557 jobs; 402 idle, 145 running, 10 held

[einstein@submit ~]$
```

1,000,000 jobs...

- We could put all input, output, error & log files in the one directory
  - One of each type for each job
  - 4,000,000+ files (4 files × 1,000,000 jobs)
  - Submit file: 6,000,000+ lines, ~128M
  - Difficult (at best) to sort through
- Better: Create a subdirectory for each run
  - Take advantage of `InitialDir` directive

Organize your files and directories for big runs

- Create subdirectories for each "run"
  - `run_0`, `run_1`, … `run_999999`
- Create input files in each of these
  - `run_0/(cosmos.in,cosmos.dat)`
  - `run_1/(cosmos.in,cosmos.dat)`
  - …
  - `run_999999/(cosmos.in,cosmos.dat)`

- The output, error & log files for each job will be created by Condor from your job's output

More Data Files

- Move the values of G, c & $\mathcal{R}_{\mu\nu}$ for each run to a data file
  - Let's call it `cosmos.in`
  - Each run directory would contain a unique `cosmos.in` file
- The common `cosmos.dat` file could be shared by all jobs
  - Can be symlinks to a common file

cosmos.in files

- These `cosmos.in` files can easily be generated programmatically using Python or Perl

```
run_0/cosmos.in
c = 299792408
G = 6.67300e-112
R = 10.00e-29
```

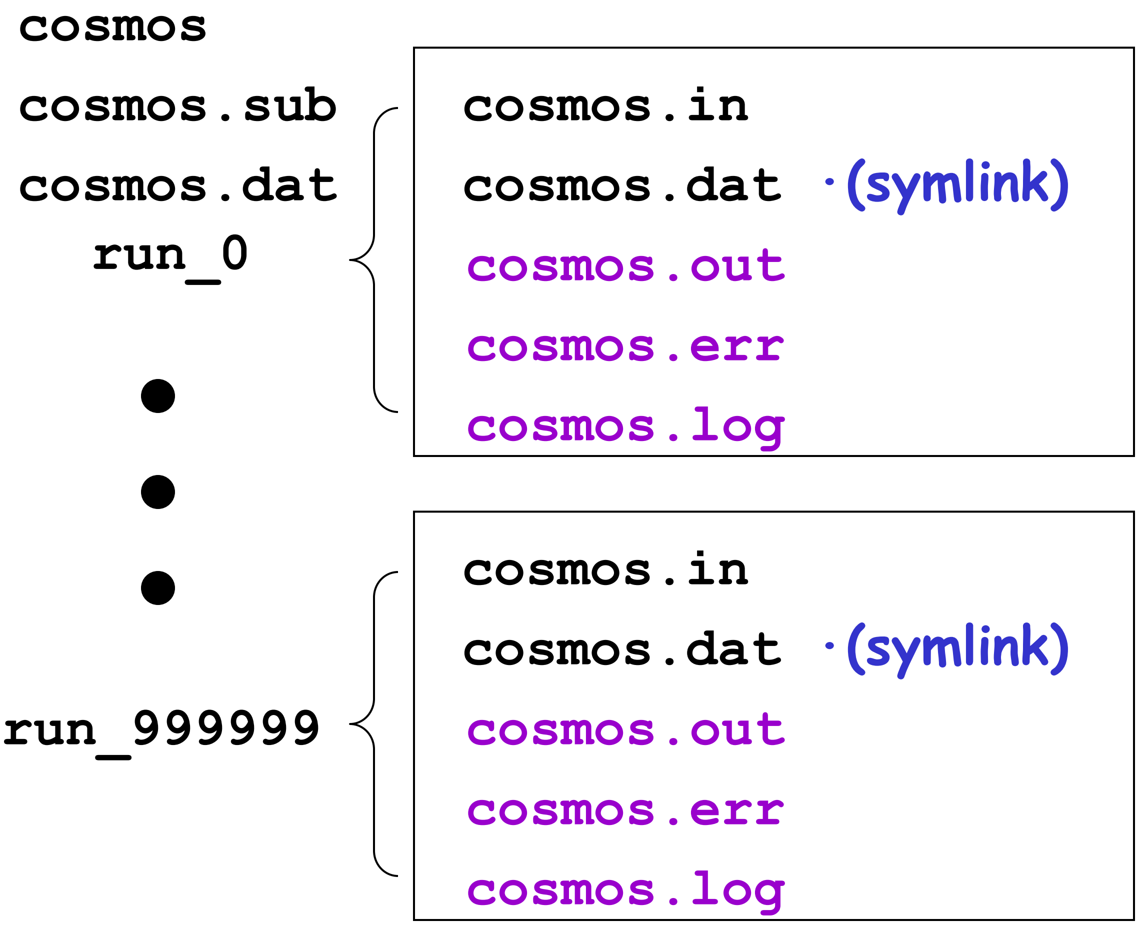
```
run_1/cosmos.in
c = 299792409
G = 6.67300e-112
R = 10.00e-29
```

…

```
run_999998/cosmos.in
c = 299792508
G = 6.67300e-100
R = 10.49e-29
```

```
run_999999/cosmos.in
c = 299792508
G = 6.67300e-100
R = 10.50e-29
```

Einstein's simulation directory



```
cosmos
cosmos.sub
cosmos.dat
   run_0        cosmos.in
                cosmos.dat  · (symlink)
                cosmos.out
                cosmos.err
                cosmos.log
     ●
     ●
     ●
run_999999      cosmos.in
                cosmos.dat  · (symlink)
                cosmos.out
                cosmos.err
                cosmos.log
```

User or script creates black files

Condor creates purple files for you

Submit Description File for 1,000,000 Jobs

```
# Cluster of 1,000,000 jobs with different directories
Universe                  = vanilla
Executable                   = cosmos
Log                       = cosmos.log
Output                    = cosmos.out
Input                     = cosmos.in
Arguments                 = -f cosmos.dat
TransferInputFiles        = cosmos.dat
...
```

```
InitialDir = run_0
Queue
```

- Log, input, output & error files -> run_0
- Job 103.0 (Cluster 103, Process 0)

```
InitialDir = run_1
Queue
```

- Log, input, output & error files -> run_1
- Job 103.1 (Cluster 103, Process 1)

- Do this 999,998 more times…………

Submit File for a Big Cluster of Jobs

- We just submitted 1 cluster with 1,000,000 processes
- All the input/output files will be in different directories
- The submit file still is pretty unwieldy
  - 2,000,000+ lines, ~32M
- Isn't there a better way?

The Better Way

- We can queue all 1,000,000 in 1 "Queue" directive
  - `Queue 1000000`
- Condor provides `$(Process)` and `$(Cluster)`
  - `$(Process)` will be expanded to the process number for each job in the cluster
    - 0, 1, … 999999
  - `$(Cluster)` will be expanded to the cluster number
    - Will be the same for all jobs in the cluster
    - 104 in our example

Using $(Process)

- The initial directory for each job can be specified using `$(Process)`
  - `InitialDir = run_$(Process)`
  - Condor will expand these to:
    - “`run_0`”, “`run_1`”, … “`run_999999`” directories
- Similarly, arguments can be variable
  - `Arguments = -n $(Process)`
  - Condor will expand these to:
    - `“-n 0”, “-n 1”, … “-n 999999”`

Better Submit File for 1,000,000 Jobs

```
# Example condor_submit input file that defines
# a cluster of 100000 jobs with different directories
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
Error      = cosmos.err
Arguments  = -f cosmos.dat
InitialDir = run_$(Process)
Queue 1000000
```

- All share arguments
- run_0 … run_999999
- Jobs 104.0 … 104.999999

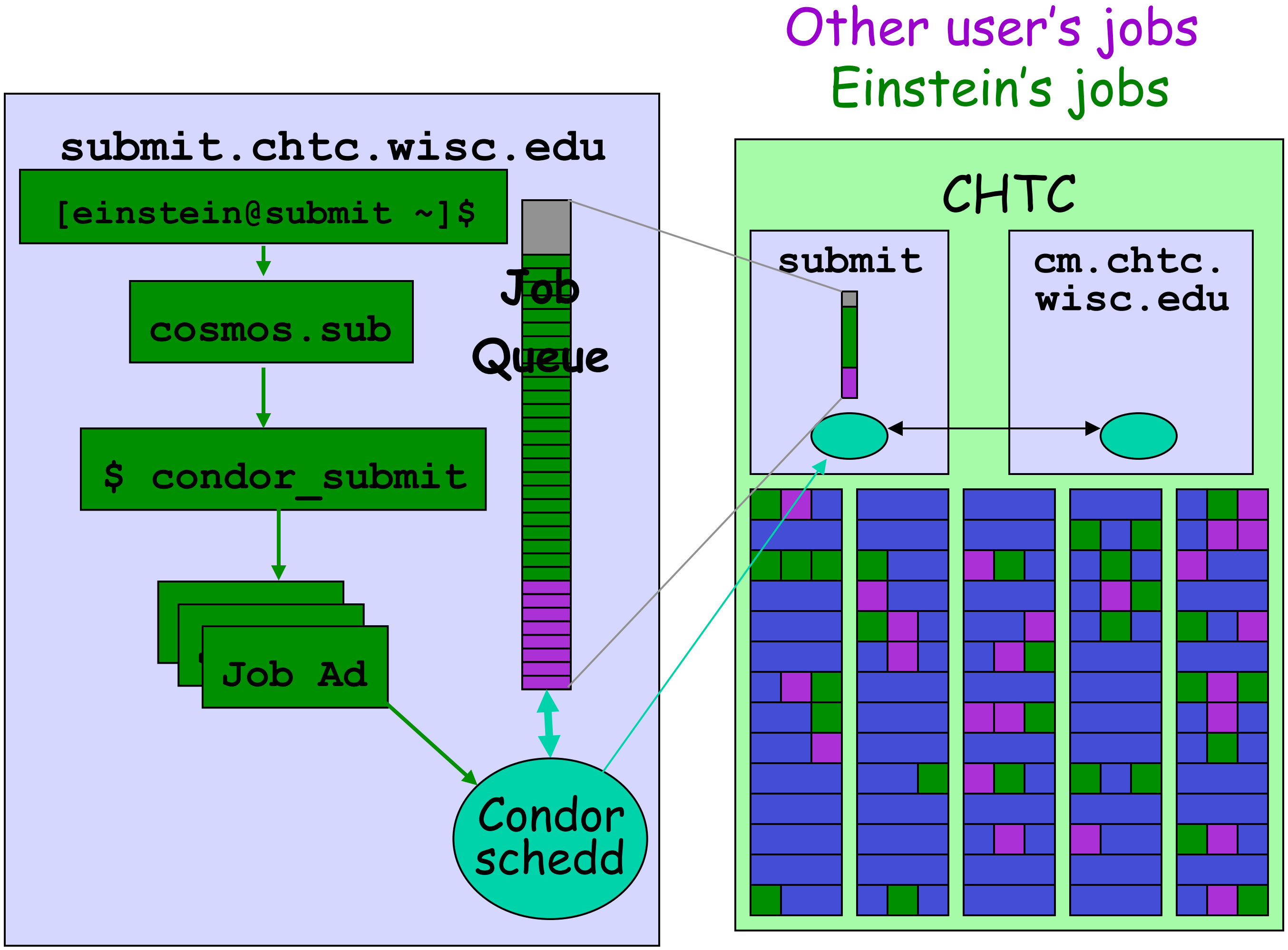
Now, we submit it…

```
[einstein@submit ~]$ condor_submit cosmos.submit
Submitting job
   (s) ............................................
   ................................................
   ................................................
   ................................................
   ................................................
   ...................
Logging submit event
   (s) ............................................
   ................................................
   ................................................
   ................................................
   ................................................
   ...................
1000000 job(s) submitted to cluster 104.
```

Check the Job Queue

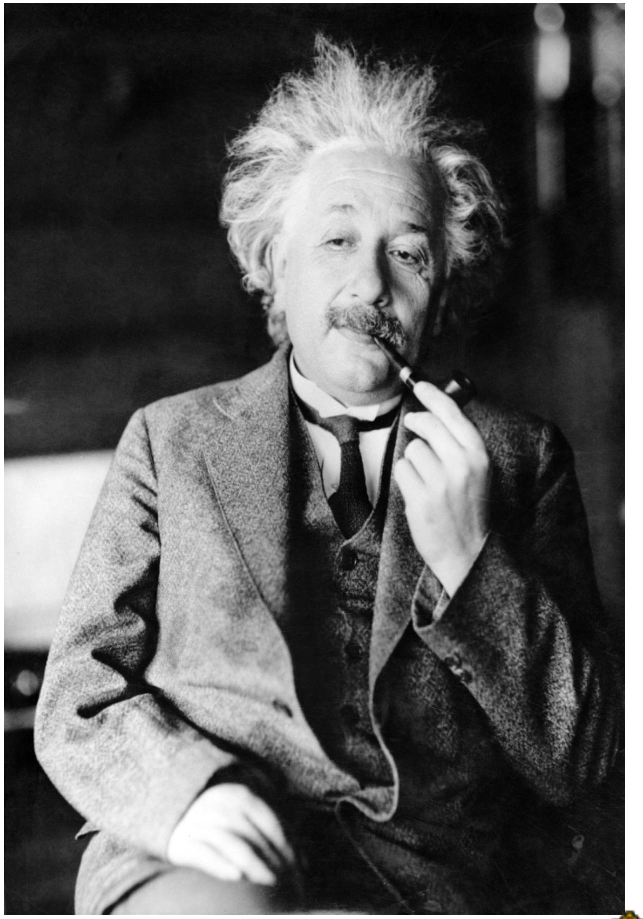
```
[einstein@submit ~]$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:51883> :
   submit.chtc.wisc.edu
ID        OWNER     SUBMITTED     RUN_TIME   ST PRI SIZE CMD
104.0      einstein 4/20 12:08 0+00:00:05 R 0 9.8 cosmos -f cosmos.dat
104.1      einstein 4/20 12:08 0+00:00:03 I 0 9.8 cosmos -f cosmos.dat
104.2      einstein 4/20 12:08 0+00:00:01 I 0 9.8 cosmos -f cosmos.dat
104.3      einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos -f cosmos.dat
...
104.999998 einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos -f cosmos.dat
104.999999 einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos -f cosmos.dat

999999 jobs; 999998 idle, 1 running, 0 held
```

CHTC Condor Pool



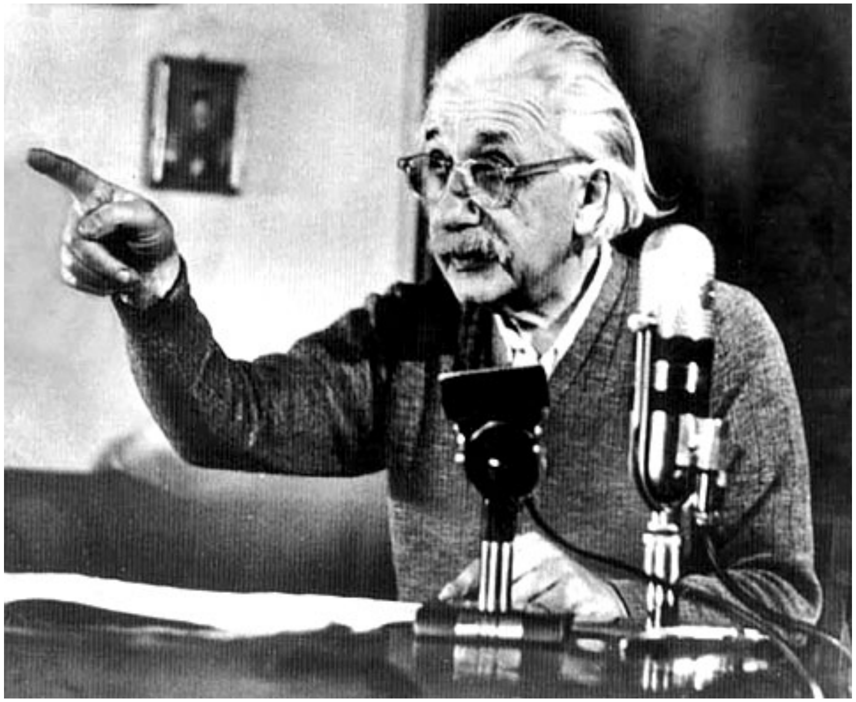
6. Relax

- Condor is watching over your jobs
  - Will restart them if required, etc.
- While I'm waiting…
  - Is there more that I can do with Condor?



Oh <censored>!!! My Biggest Blunder Ever

- Albert removes $\mathcal{R}_{\mu\nu}$ (Cosmological Constant) from his equations, and needs to remove his running jobs
- We'll just ignore that modern cosmologists may have re-introduced $\mathcal{R}_{\mu\nu}$ (so called "dark energy")



Removing Jobs

- If you want to remove a job from the Condor queue, you use *condor_rm*
- You can only remove jobs that you own
- Privileged user can remove any jobs
  - "root" on UNIX
  - "administrator" on Windows

Removing jobs (continued)

- Remove an entire cluster:
  - `condor_rm 4` · Removes the whole cluster
- Remove a specific job from a cluster:
  - `condor_rm 4.0` · Removes a single job
- Or, remove **all** of your jobs with "*-a*"
  - `condor_rm -a` · Removes all jobs / clusters

How can I tell Condor that my jobs are Physics related?

- In the submit description file, introduce an attribute for the job

```
+Department = "physics"
```

Causes the Job Ad to contain

```
Department = "physics"
```

- Machines can be configured to:
  - Give higher rank to physics jobs
  - Pre-empt non-physics jobs when a physics job comes along
  - See Alan's "Administering Condor" tutorial for more about machine "policy expressions"

How Can I Control Where my Jobs Run?

- Some of the machines in the pool can't successfully run my jobs
  - Not enough RAM
  - Not enough scratch disk space
  - Required software not installed
  - Etc.

Specify Job Requirements

- A boolean expression (syntax similar to C or Java)
- Must evaluate to True for a match to be made

```
Universe       = vanilla
Executable     = cosmos
Log            = cosmos.log
InitialDir     = run_$(Process)
Input          = cosmos.in
Output         = cosmos.out
Error          = cosmos.err
Requirements   = ( (Memory >= 4096) && \
                   (Disk > 10000)    )

Queue 1000000
```

Advanced Requirements

- Requirements can match custom attributes in your Machine Ad
  - Can be added by hand to each machine
  - Or, automatically using the “Hawkeye” mechanism

```
Universe       = vanilla
Executable     = cosmos
Log            = cosmos.log
InitialDir     = run_$(Process)
Input          = cosmos.in
Output         = cosmos.out
Error          = cosmos.err
Requirements   = ((Memory >= 4096) && \
                  (Disk > 10000)   && \
                  (CosmosData =!= UNDEFINED))

Queue 1000000
```

CosmosData =!= UNDEFINED ???

- What's this "=!=" and "UNDEFINED" business?
  - Is this like the state of Schrödinger's Cat?
- Introducing ClassAd Meta Operators:
  - Allow you to test if a attribute is in a ClassAd
  - Is identical to operator: "=?="
  - Is not identical to operator: "=!="
  - Behave similar to == and !=, but are not strict
  - Somewhat akin to Python's "is NONE" and "is not NONE"
  - Without these, ANY expression with an UNDEFINED in it will always evaluate to UNDEFINED

Meta Operator Examples

Expression	Evaluates to
`10 == UNDEFINED`	`UNDEFINED`
`UNDEFINED == UNDEFINED`	`UNDEFINED`
`10 =?= UNDEFINED`	`False`
`UNDEFINED =?= UNDEFINED`	`True`
`UNDEFINED =!= UNDEFINED`	`False`

More Meta Operator Examples

Expression	x	Evaluates to
`x == 10`	10	`True`
	5	`False`
	"ABC"	`ERROR`
	*	`UNDEFINED`
`x =!= UNDEFINED`	5	`True`
	10	`True`
	"ABC"	`True`
	*	`False`

*: `x` is not present in the ClassAd

One Last Meta Example

<table>
<tr><th>Expression</th><th>X</th><th>Evaluates to</th></tr>
<tr><td rowspan="4">((X =!= UNDEFINED) &&
(X == 10))
Is logically equivalent to:
(X =?= 10)</td><td>10</td><td>True</td></tr>
<tr><td>5</td><td>False</td></tr>
<tr><td>11</td><td>False</td></tr>
<tr><td>*</td><td>False</td></tr>
<tr><td rowspan="4">((X =?= UNDEFINED) ||
(X != 10))
Is logically equivalent to:
(X =!= 10)</td><td>10</td><td>False</td></tr>
<tr><td>5</td><td>True</td></tr>
<tr><td>11</td><td>True</td></tr>
<tr><td>*</td><td>True</td></tr>
</table>

*: X is not present in the ClassAd

Using Attributes from the Machine Ad

- You can use attributes from the matched Machine Ad in your job submit file
  - **`$$(`*`<attribute>`*`)` will be replaced by the value of the specified attribute in the Machine Ad**
- **Example:**
  - Matching Machine Ad has:

```
CosmosData      = "/local/cosmos/data"
```

  - Submit file has:

```
Executable      = cosmos
Requirements    = (CosmosData =!= UNDEFINED)
Arguments       = -d $$(CosmosData)
```

  - Resulting command line:

```
cosmos -d /local/cosmos/data
```

Specify Job Rank

- Rank:
  - Numerical expression
  - All matches which meet the requirements can be sorted by preference with a Rank expression..
  - Higher values match first

```
Universe         = vanilla
Executable       = cosmos
Log              = cosmos.log
Arguments        = -arg1 -arg2
InitialDir       = run_$(Process)
Requirements     = (Memory >= 4096) && (Disk > 10000)
Rank             = (KFLOPS*10000) + Memory
Queue 1000000
```

Need More Control of Your Job?

- Exit status isn't always a good indicator of job success
- What if my job gets a signal?
  - **`SIGSEGV`**
  - **`SIGBUS`**
- …

Job Policy Expressions

- User can supply job policy expressions in the submit file.
- Can be used to describe a successful run.

```
on_exit_remove  = <expression>
on_exit_hold    = <expression>
periodic_remove = <expression>
periodic_hold   = <expression>
```

Job Policy Examples

- Do not remove if exits with a signal:

```
on_exit_remove = ExitBySignal == False
```

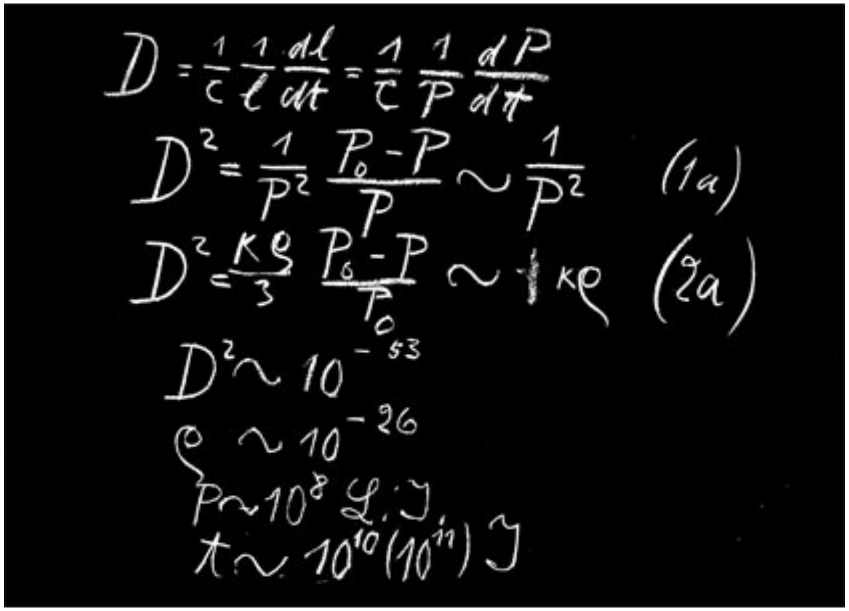
- Place on hold if exits with nonzero status or ran for less than an hour:

```
on_exit_hold =
  ( (ExitBySignal==False) && (ExitSignal != 0) ) ||
  ( (ServerStartTime - JobStartDate) < 3600)
```

- Place on hold if job has spent more than 50% of its time suspended:

```
periodic_hold =
  ( CumulativeSuspensionTime >
    (RemoteWallClockTime / 2.0) )
```

How can my jobs access their data files?


$D = \frac{1}{c}\frac{1}{l}\frac{dl}{dt} = \frac{1}{c}\frac{1}{P}\frac{dP}{dt}$
$D^2 = \frac{1}{P^2}\frac{P_0 - P}{P} \sim \frac{1}{P^2}$ (1a)
$D^2 = \frac{K\rho}{3}\frac{P_0 - P}{P_0} \sim \frac{1}{3}K\rho$ (2a)
$D^2 \sim 10^{-53}$
$\rho \sim 10^{-26}$
$P \sim 10^8$ L.J.
$t \sim 10^{10} (10^{11})$ J

Access to Data in Condor

- Condor can transfer files
  - We've already seen examples of this
  - Can automatically send back changed files
  - Atomic transfer of multiple files
  - Can be encrypted over the wire
- NFS / AFS
- HDFS
- Remote I/O Socket (parrot)
- Standard Universe can use remote system calls (more on this later)

NFS / AFS

- Condor can be configured to allow access to NFS and AFS shared resources
- AFS is available on most of CHTC
- Your program can access /afs/...
- Note: Condor runs without AFS credentials
  - At UW Computer Sciences, you must grant net:cs access to all Condor job input, output, and log files stored in AFS directories.
  - Elsewhere, you'll have to do something similar

I Need to run LOTS of Short Jobs

- Condor is a High Throughput system, designed for long running jobs
- Starting a job in Condor is somewhat expensive
- There are some configuration parameters that may be able to help
  - Contact a Condor staff person for more

What else can I do to run my Short Jobs with Condor?

- Batch your short jobs together
  - Write a wrapper script that will run a number of them in series
  - Submit your wrapper script as your job
- Explore Condor's parallel universe

Need to Learn Scripting?

- CS 368 / Summer 2010
- Introduction to Scripting Languages
- Two Sections
  - Both taught by Condor Staff Members
  - Section 1
    - Perl
    - Instructor: Tim Cartwright
  - Section 2
    - Python
    - Instructor: Nick LeRoy

Parallel Universes

MW: A Master-Worker Grid Toolkit

- Provides a mechanism for controlling parallel algorithms
  - Fault tolerant
  - Allows for resources to come and go
  - Ideal for Computational Grid settings
- To use, write your software using the MW API
- http://www.cs.wisc.edu/condor/mw/

MPI jobs

```
# Example condor_submit input file that for MPI
# jobs
universe = parallel
executable = mp1script
arguments = my_mpich_linked_executable arg1 arg2
machine_count = 4
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = my_mpich_linked_executable
queue
```

Map Reduce

- Condor provides a powerful execution environment for running parallel applications like MPI.
  - The Parallel Universe (PU) of Condor is built specifically for this purpose
  - The Map-Reduce (MR) is a relatively recent programming model particularly suitable for applications that require processing a large set of data on cluster of computers.
  - A popular open-source implementation of MR framework is provided by Hadoop project by apache software foundation.

Map Reduce On Condor

- Uses Condor's Parallel Universe resource manager to select a subset of machines within a cluster
  - Sets up a Hadoop MR cluster on these machines
  - Submits a MR job and clean-up once the job is finished
  - These machines will be available as dedicated resources for the duration of the job
  - User can choose which machine should act as a master and communication channels between masters and slave nodes are also established

- `http://condor-wiki.cs.wisc.edu/index.cgi/wiki?p=MapReduce`

Accessing Large Data Sets via HDFS

- HDFS
  - Allows disk space to be pooled into one resource
  - For the CS CHTC cluster, that is on the order of a couple hundred terabytes
- Can enable jobs with large I/O to run without filling up the spool on submit machine
- However, HDFS has no security so cannot yet be used for sensitive data

HDFS @ CHTC

- HDFS on CHTC
  - Managed by the **condor_hdfs** daemon which runs under the **condor_master**
- Command line tools are available to move files in and out of the HDFS
- Soon, Condor will support having the **SPOOL** directory in HDFS
- Plugins for Condor's file transfer will also support HDFS
- We're looking into the possibility of an overlay network to solve the security issues.
  - Any of Condor's security mechanisms could then be used

We've seen how Condor can:

- Keep an eye on your jobs
  - η Keep you posted on their progress
- Implement your policy on the execution order of the jobs
- Keep a log of your job activities

Using Condor

An Introduction

Part II

Condor Week 2010

More Issues...

- We need more disk space for our jobs
- We have users that come and go

Your own Submit Host

- Benefits:
  - As much disk space as you need
  - Manage your own users
- Getting Started:
  - Download & install appropriate Condor binaries
  - "Flock" into CHTC and other campus pools

Getting Condor

- Available as a free download from http://www.cs.wisc.edu/condor
- Download Condor for your operating system
  - Available for most modern UNIX platforms (including Linux and Apple's OS/X)
  - Also for Windows XP / 2003 / Vista
- Repositories
  - YUM: RHEL 4 & 5
    - `$ yum install condor`
  - APT: Debian 4 & 5
    - `$ apt-get install condor`

Condor Releases

- Stable / Developer Releases
  - Version numbering scheme similar to that of the (pre 2.6) Linux kernels …
- Major.minor.release
  - If minor is even (a.b.c): Stable series
    - Very stable, mostly bug fixes
    - Current: 7.4
    - Examples: 7.2.5, 7.4.2
      - 7.4.2 just released
  - If minor is odd (a.b.c): Developer series
    - New features, may have some bugs
    - Current: 7.5
    - Examples: 7.3.2, 7.5.1
      - 7.5.1 just released

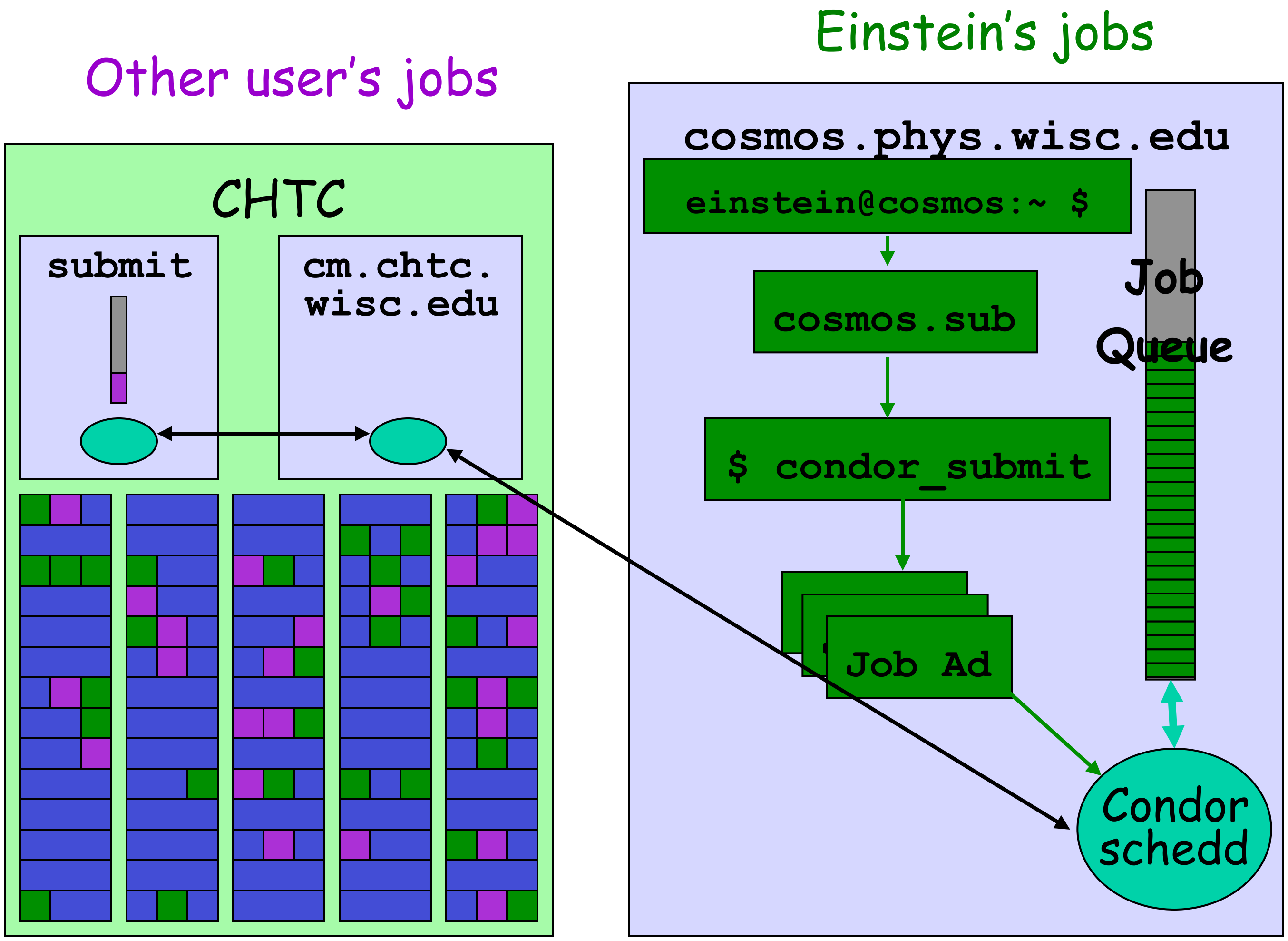
Condor Installation

- Albert's sysadmin installs Condor
  - This new submit / manager machine
  - On department desktop machines
    - Submission points
    - Non-dedicated excution machines
      - Configured to only run jobs when the machine is idle
  - Enables flocking to CHTC and other campus pools

Flocking?

- Flocking is a Condor-specific technology
- Allows Condor jobs to run in other friendly Condor pools
- Needs to be setup on both ends
- Can be bi-directional

Flocking to CHTC

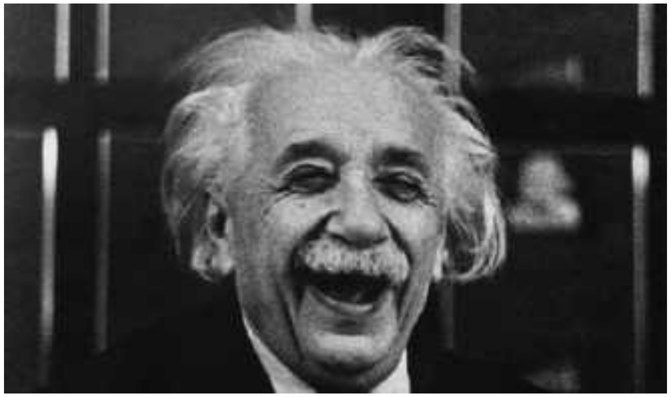


We *STILL* Need More

Condor is managing and running our jobs, but:

- Our CPU requirements are greater than our resources
- Jobs are preempted more often than we like

Happy Day! The Physics Department is adding a cluster!

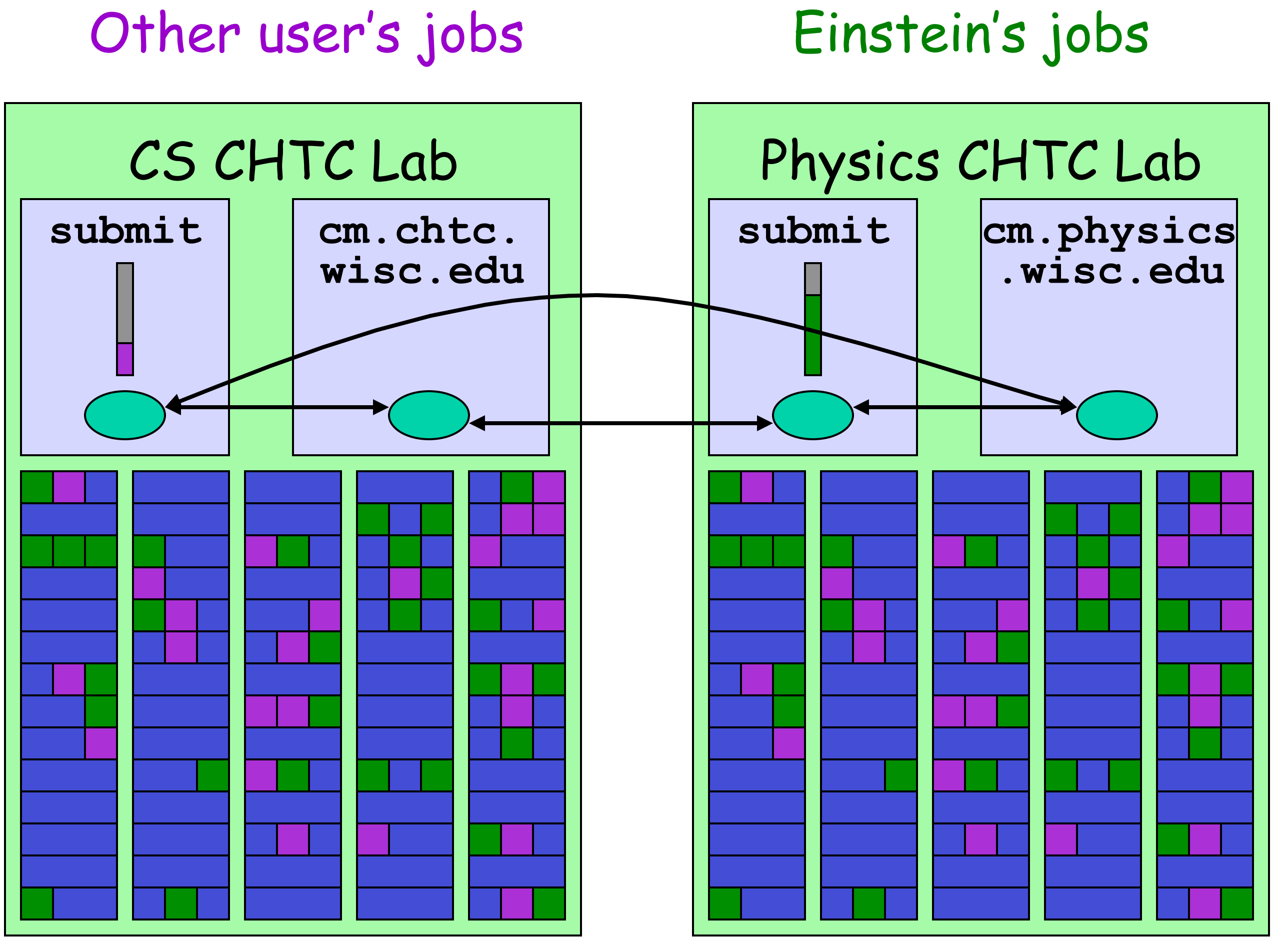


- The administrator installs Condor on all these new dedicated cluster nodes

Adding dedicated nodes

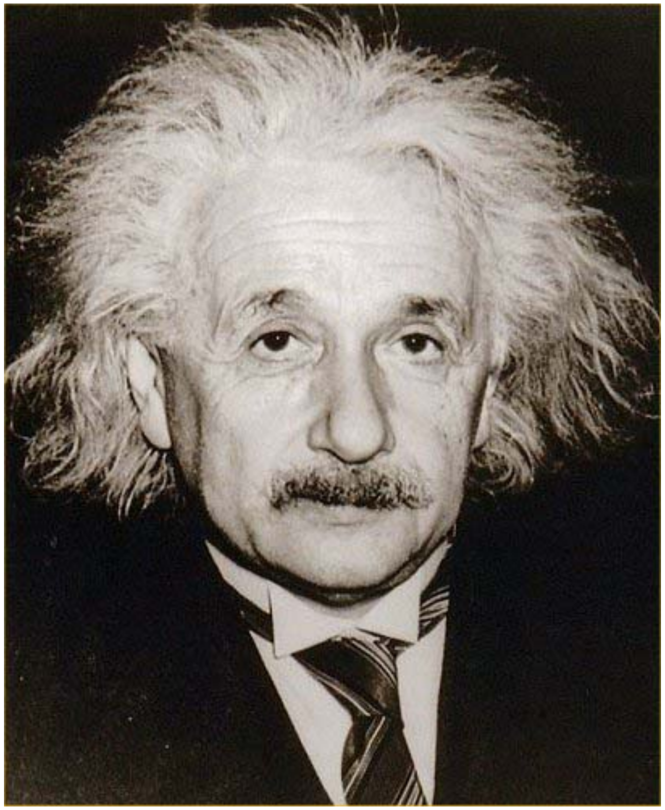
- The administrator installs Condor on these new machines, and configures them with his machine as the central manager
  - The central manager:
    - Central repository for the whole pool
    - Performs job / machine matching, etc.
- These are dedicated nodes, meaning that they're always able run Condor jobs

Flocking to CHTC

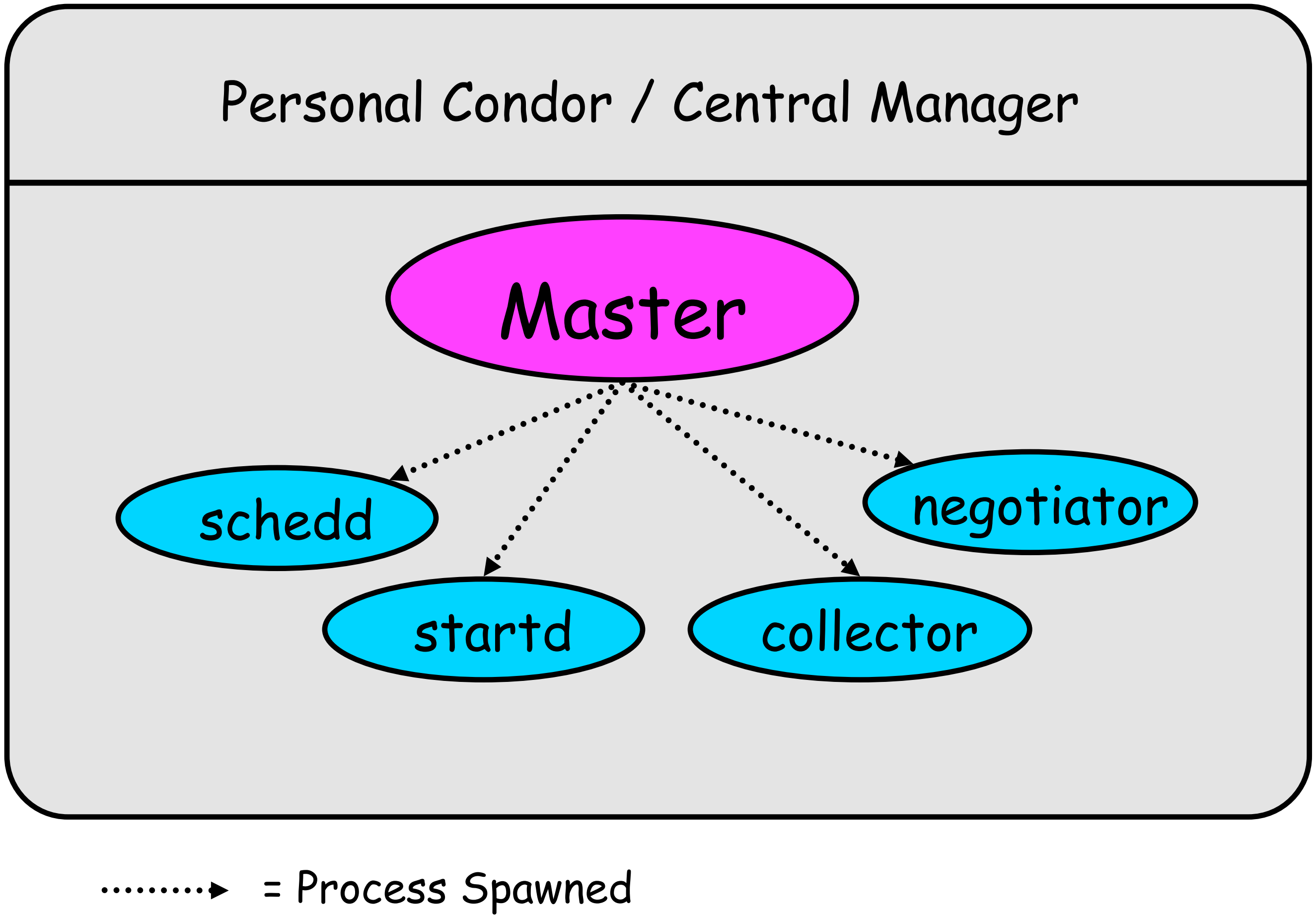


Some Good Questions…

- What are all of these Condor Daemons running on my machine?
- What do they do?



Condor Daemon Layout



condor_master

- Starts up all other Condor daemons
- Runs on **all** Condor hosts
- If there are any problems and a daemon exits, it restarts the daemon and sends email to the administrator
- Acts as the server for many Condor remote administration commands:
  - ***condor_reconfig, condor_restart***
  - ***condor_off, condor_on***
  - ***condor_config_val***
  - etc.

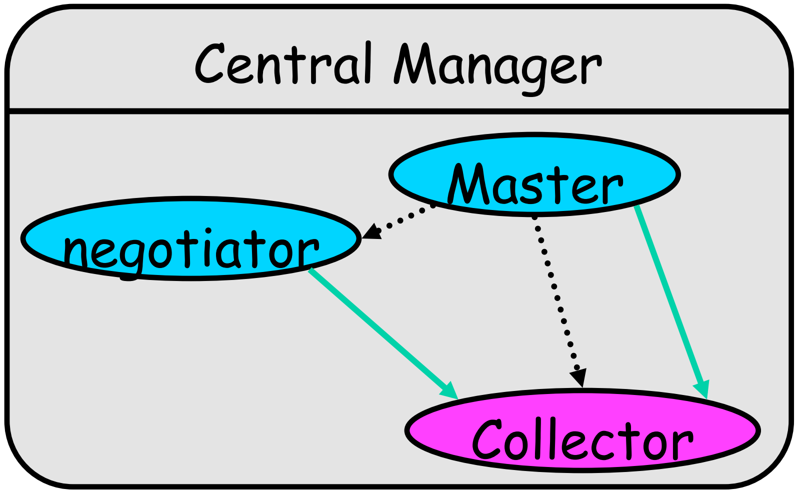
Central Manager: condor_collector

- Central manager: central repository and match maker for whole pool
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
  - Each daemon sends a periodic update ClassAd to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (***condor_status***)
- Only on the Central Manager(s)
- At least one collector per pool

Condor Pool Layout: Collector

⋯⋯▸ = Process Spawned

⟶ = ClassAd Communication Pathway



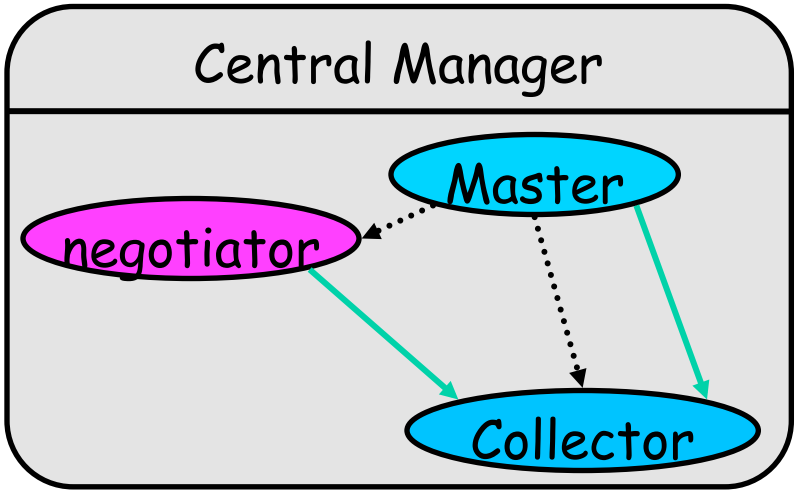
Central Manager: condor_negotiator

- Performs "matchmaking" in Condor
- Each "Negotiation Cycle" (typically 5 minutes):
  - Gets information from the collector about all available machines and all idle jobs
  - Tries to match jobs with machines that will serve them
  - Both the job and the machine must satisfy each other's requirements
- Only one Negotiator per pool
  - Ignoring HAD
- Only on the Central Manager(s)

Condor Pool Layout: Negotiator

········▸ = Process Spawned

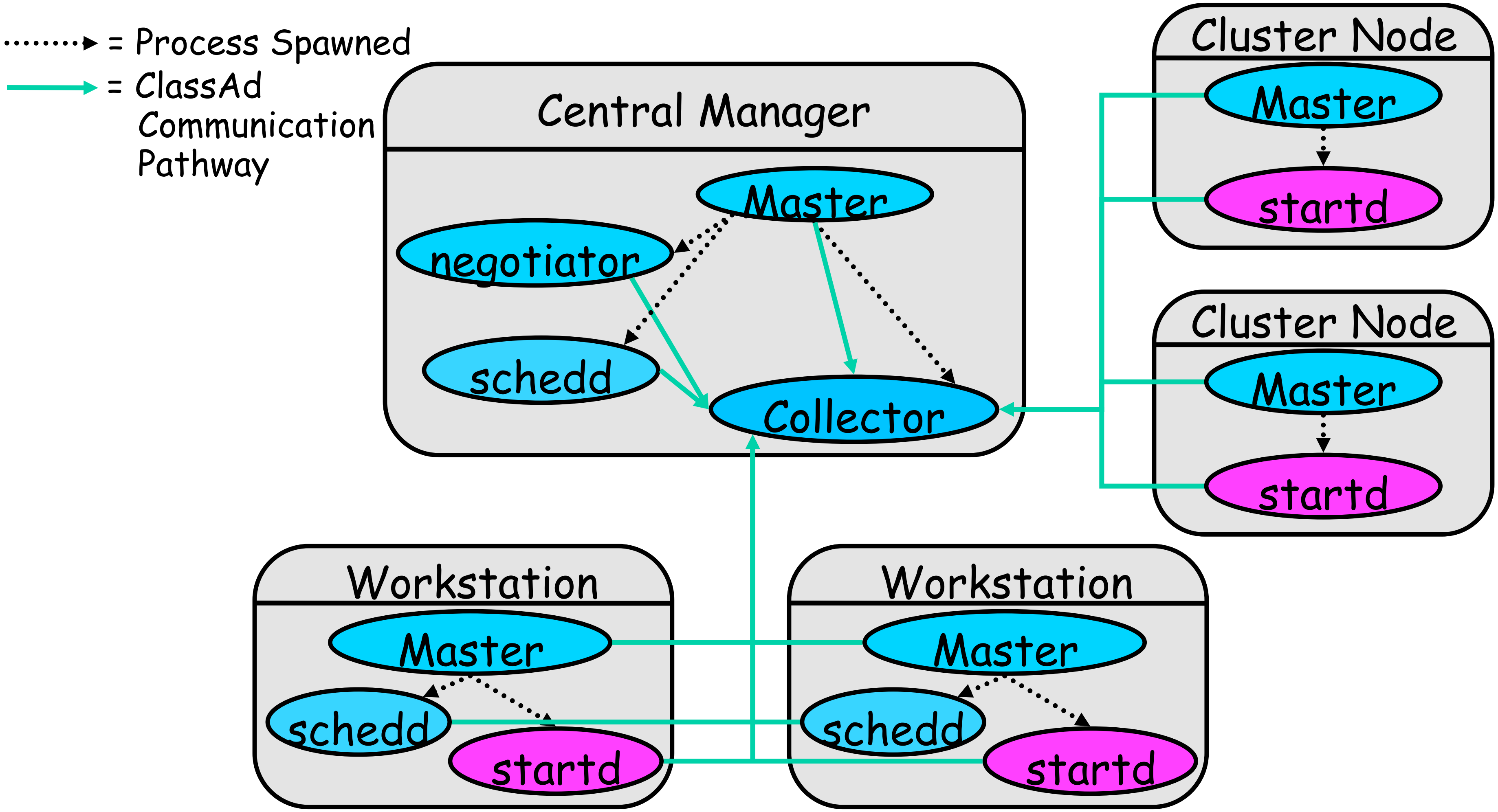
⟶ = ClassAd Communication Pathway



Execute Hosts: condor_startd

- Execute host: machines that run user jobs
- Represents a machine to the Condor system
- Responsible for starting, suspending, and stopping jobs
- Enforces the wishes of the machine owner (the owner's "policy"… more on this in the administrator's tutorial)
- Creates a "starter" for each running job
- One startd runs on each execute node

Condor Pool Layout: startd



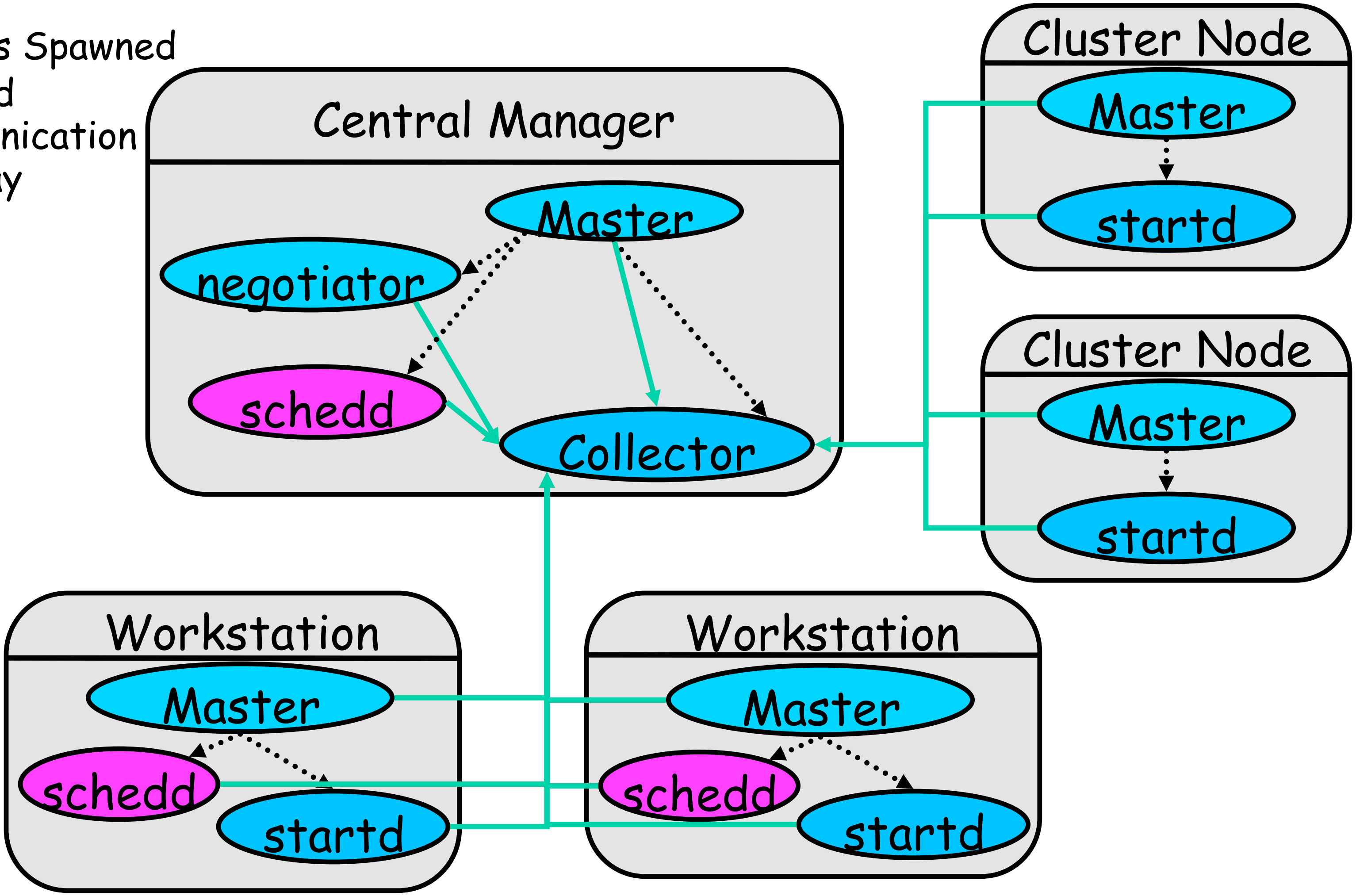
Submit Hosts: condor_schedd

- Submit hosts: machines that users can submit jobs on
- Maintains the persistent queue of jobs
- Responsible for contacting available machines and sending them jobs
- Services user commands which manipulate the job queue:
  - *condor_submit, condor_rm, condor_q, condor_hold, condor_release, condor_prio, …*
- Creates a "shadow" for each running job
- One schedd runs on each submit host

Condor Pool Layout: schedd

········► = Process Spawned

⟶ = ClassAd Communication Pathway



Cluster Node

Master

startd

Cluster Node

Master

startd

Central Manager

Master

negotiator

schedd

Collector

Workstation

Master

schedd

startd

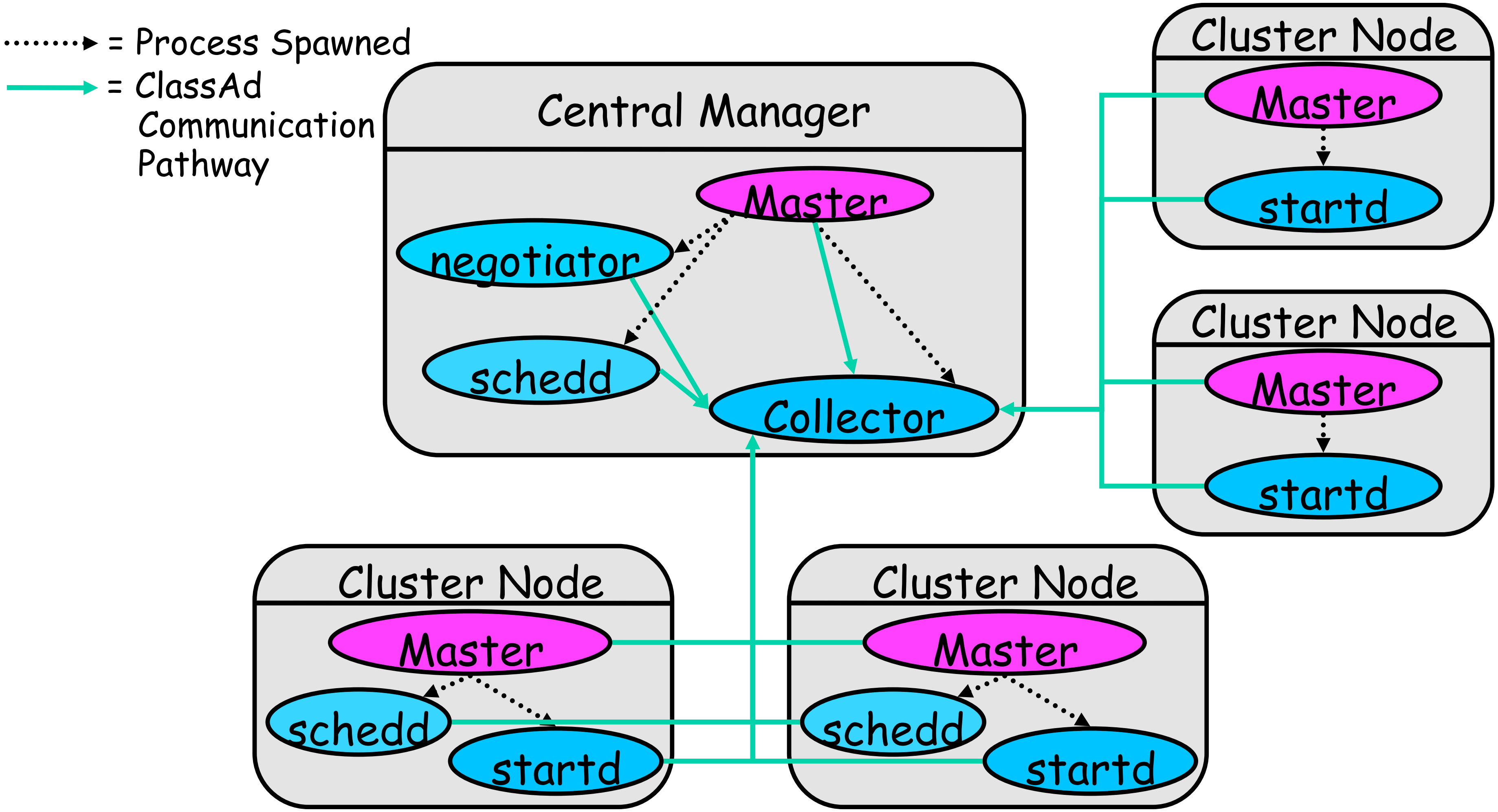
Workstation

Master

schedd

startd

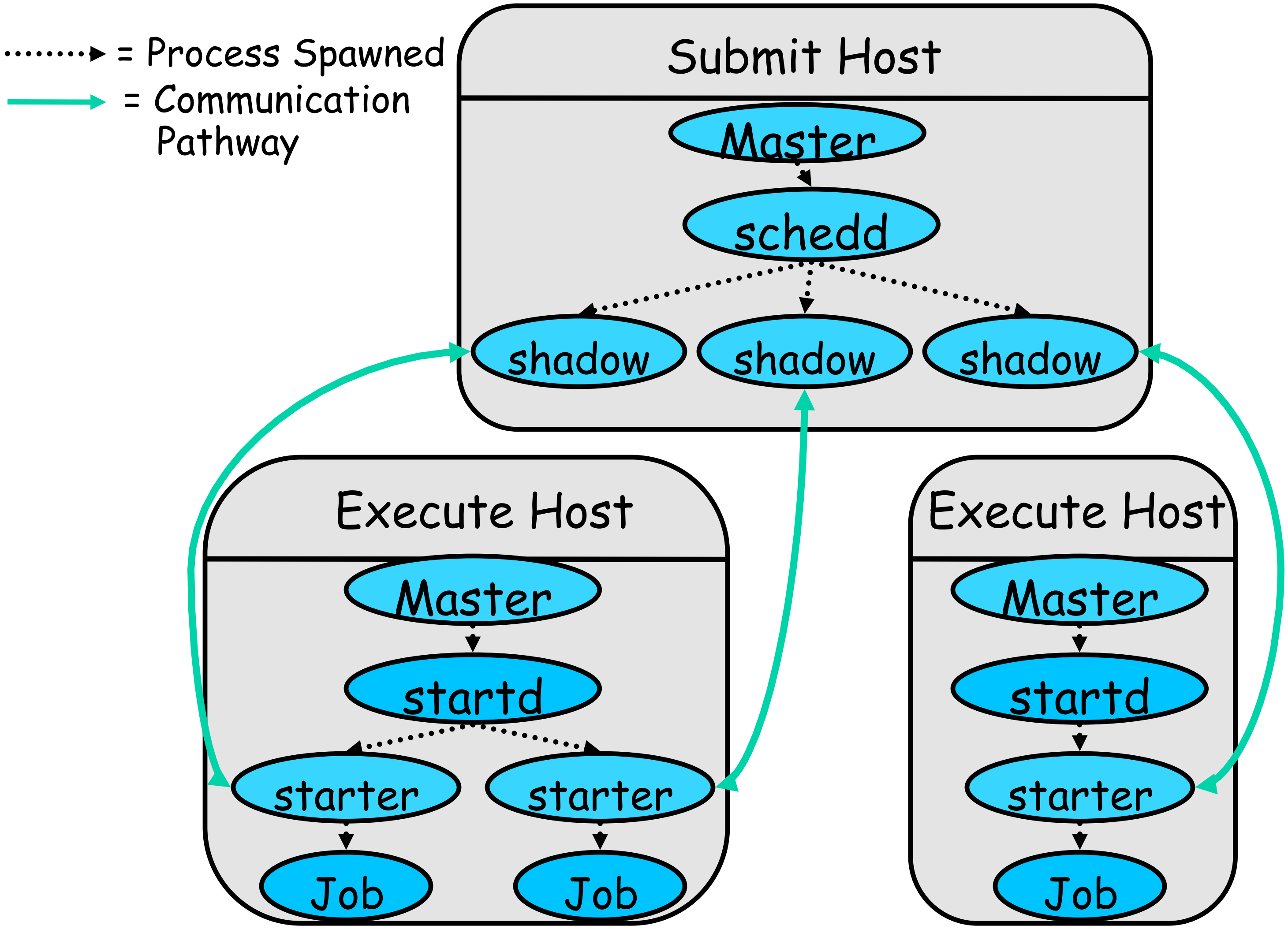
Condor Pool Layout: master



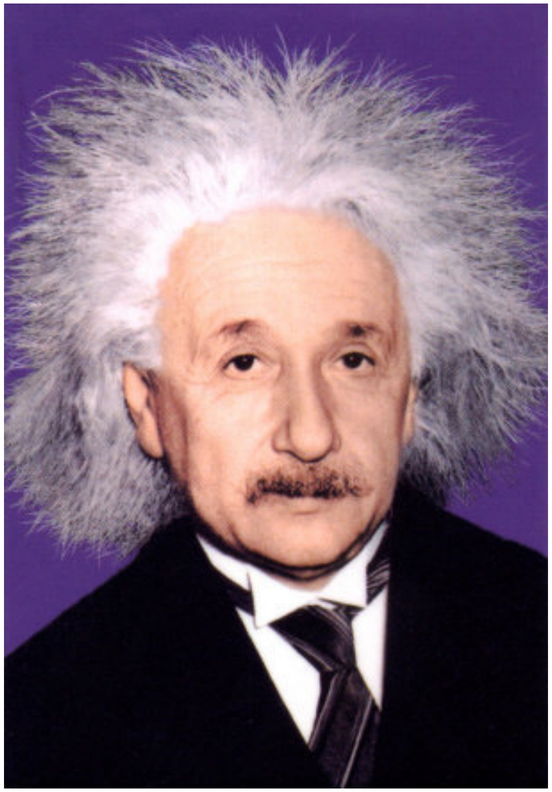
What about these "condor_shadow" processes?

- The Shadow processes are Condor's local representation of your running job
  - One is started for each job
- Similarly, on the "execute" machine, a condor_starter is run for each job

Condor Pool Layout: running a job



My jobs aren't running!!



Check the queue

- Check the queue with condor_q:

```
[einstein@submit ~]$ condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510> :x.cs.wisc.edu
ID  OWNER   SUBMITTED    RUN_TIME ST PRI SIZE CMD
5.0 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 0
5.1 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 1
5.2 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 2
5.3 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 3
5.4 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 4
5.5 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 5
5.6 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 6
5.7 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 -n 7
6.0 einstein  4/20 13:22  0+00:00:00 H 0   9.8  cosmos -arg1 -arg2
8 jobs; 8 idle, 0 running, 1 held
```

Look at jobs on hold

```
[einstein@submit ~]$ condor_q -hold
-- Submiter: submit.chtc.wisc.edu :
  <128.105.121.53:510> :submit.chtc.wisc.edu
 ID       OWNER                HELD_SINCE HOLD_REASON
 6.0    einstein               4/20 13:23 Error from starter
  on skywalker.cs.wisc.edu

9 jobs; 8 idle, 0 running, 1 held
```

Or, See full details for a job

```
[einstein@submit ~]$ condor_q -l 6.0
```

Check Machine's Status

```
[einstein@submit ~]$ condor_status
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime
slot1@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  4599  0+00:10:13
slot2@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  1024  1+19:10:36
slot3@c002.chtc.wi LINUX      X86_64 Claimed   Busy     0.990  1024  1+22:42:20
slot4@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:22:10
slot5@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:17:00
slot6@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:09:14
slot7@c002.chtc.wi LINUX      X86_64 Claimed   Busy     1.000  1024  0+19:13:49
...
vm1@INFOLABS-SML65 WINNT51    INTEL  Owner     Idle     0.000   511   [Unknown]
vm2@INFOLABS-SML65 WINNT51    INTEL  Owner     Idle     0.030   511   [Unknown]
vm1@INFOLABS-SML66 WINNT51    INTEL  Unclaimed Idle     0.000   511   [Unknown]
vm2@INFOLABS-SML66 WINNT51    INTEL  Unclaimed Idle     0.010   511   [Unknown]
vm1@infolabs-smlde WINNT51    INTEL  Claimed   Busy     1.130   511   [Unknown]
vm2@infolabs-smlde WINNT51    INTEL  Claimed   Busy     1.090   511   [Unknown]
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

       INTEL/WINNT51   104    78      16        10       0          0        0
        X86_64/LINUX   759   170     587         0       0          1        0

               Total   863   248     603        10       0          1        0
```

Look in the Job Log

- Look in your job log for clues:

```
[einstein@submit ~]$ cat cosmos.log
000 (031.000.000) 04/20 14:47:31 Job submitted from
   host: <128.105.121.53:48740>
...
007 (031.000.000) 04/20 15:02:00 Shadow exception!
        Error from starter on gig06.stat.wisc.edu:
   Failed to open '/scratch.1/einstein/workspace/v67/
   condor-test/test3/run_0/cosmos.in' as standard
   input: No such file or directory (errno 2)
        0  -  Run Bytes Sent By Job
        0  -  Run Bytes Received By Job
...
```

Still not running?
Exercise a little patience

- On a busy pool, it can take a while to match and start your jobs
- Wait at least a negotiation cycle or two (typically a few minutes)

Let Condor help: condor_q -analyze

```
[einstein@submit ~]$ condor_q -ana 29
The Requirements expression for your job is:

( (target.Memory > 8192) ) && (target.Arch == "INTEL") &&
(target.OpSys == "LINUX") && (target.Disk >= DiskUsage) &&
(TARGET.FileSystemDomain == MY.FileSystemDomain)
Condition                     Machines    Matched  Suggestion
---------                     ----------- -------- -----------
1 ( ( target.Memory > 8192 ) )  0              MODIFY TO 4000
2 ( TARGET.FileSystemDomain == "cs.wisc.edu" )584
3   ( target.Arch == "INTEL" )  1078
4   ( target.OpSys == "LINUX" ) 1100
5   ( target.Disk >= 13 )       1243
```

Learn about available resources:

```
[einstein@submit ~]$ condor_status -const 'Memory > 8192'
```

(no output means no matches)

```
[einstein@submit ~]$ condor_status -const 'Memory > 4096'
Name              OpSys   Arch   State     Activ LoadAv Mem   ActvtyTime
vm1@s0-03.cs.     LINUX   X86_64 Unclaimed Idle  0.000  5980  1+05:35:05
vm2@s0-03.cs.     LINUX   X86_64 Unclaimed Idle  0.000  5980 13+05:37:03
vm1@s0-04.cs.     LINUX   X86_64 Unclaimed Idle  0.000  7988  1+06:00:05
vm2@s0-04.cs.     LINUX   X86_64 Unclaimed Idle  0.000  7988 13+06:03:47

                Total Owner Claimed Unclaimed Matched Preempting
   X86_64/LINUX     4     0       0         4       0          0
          Total     4     0       0         4       0          0
```

It's Still not Working

- Verify that the submitted job runs stand alone
  - We've had many cases in which users blame Condor, but haven't tried running it outside of Condor

Interact With Your Job

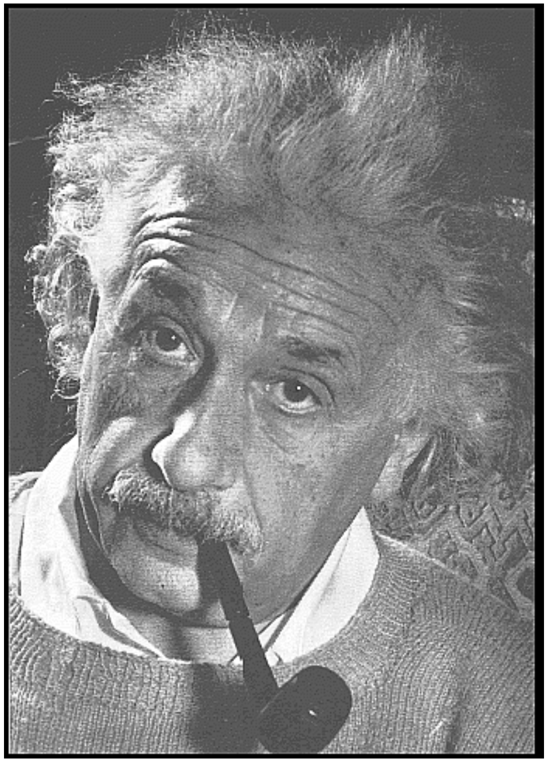
- Why is my job still running?
  Is it stuck accessing a file?
  Is it in an infinite loop?
- Try this: `$ condor_ssh_to_job`
  - Interactive debugging in UNIX
  - Use ps, top, gdb, strace, lsof, …
  - Forward ports, X, transfer files, etc.

Interactive Debug Example

```
einstein@phy:~$ condor_q

-- Submitter: cosmos.phy.wisc.edu : <128.105.165.34:1027> :
ID       OWNER       SUBMITTED    RUN_TIME   ST PRI SIZE CMD
 1.0     einstein    4/15 06:52   1+12:10:05 R  0   10.0 cosmos

1 jobs; 0 idle, 1 running, 0 held

[einstein@submit ~]$ condor_ssh_to_job 1.0

Welcome to slot4@c025.chtc.wisc.edu!
Your condor job is running with pid(s) 15603.

$ gdb -p 15603
 …
```

My new jobs can run for 20 days…

- What happens when a job is forced off it's CPU?
  - Preempted by higher priority user or job
  - Vacated because of user activity
- How can I add fault tolerance to my jobs?



Condor's Standard Universe to the rescue!

- Support for transparent process checkpoint and restart
- Remote system calls (remote I/O)
  - Your job can read / write files as if they were local

Remote System Calls in the Standard Universe

- I/O system calls are trapped and sent back to the submit machine
  Examples: open a file, write to a file
- No source code changes typically required
- Programming language independent

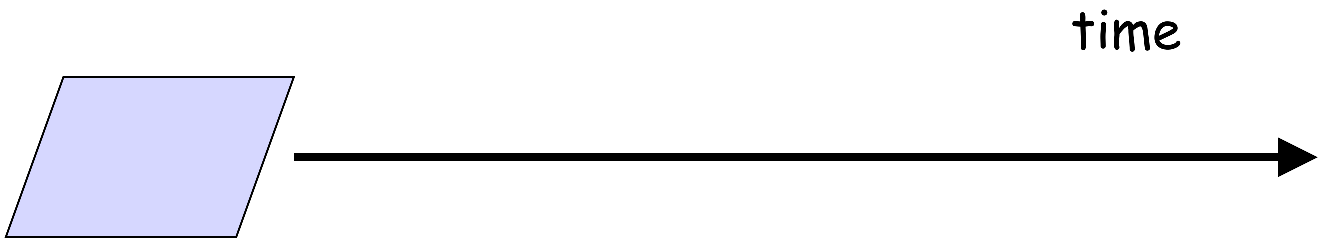
Process Checkpointing in the Standard Universe

- Condor's process checkpointing provides a mechanism to automatically save the state of a job
- The process can then be restarted *from right where it was checkpointed*
  - After preemption, crash, etc.

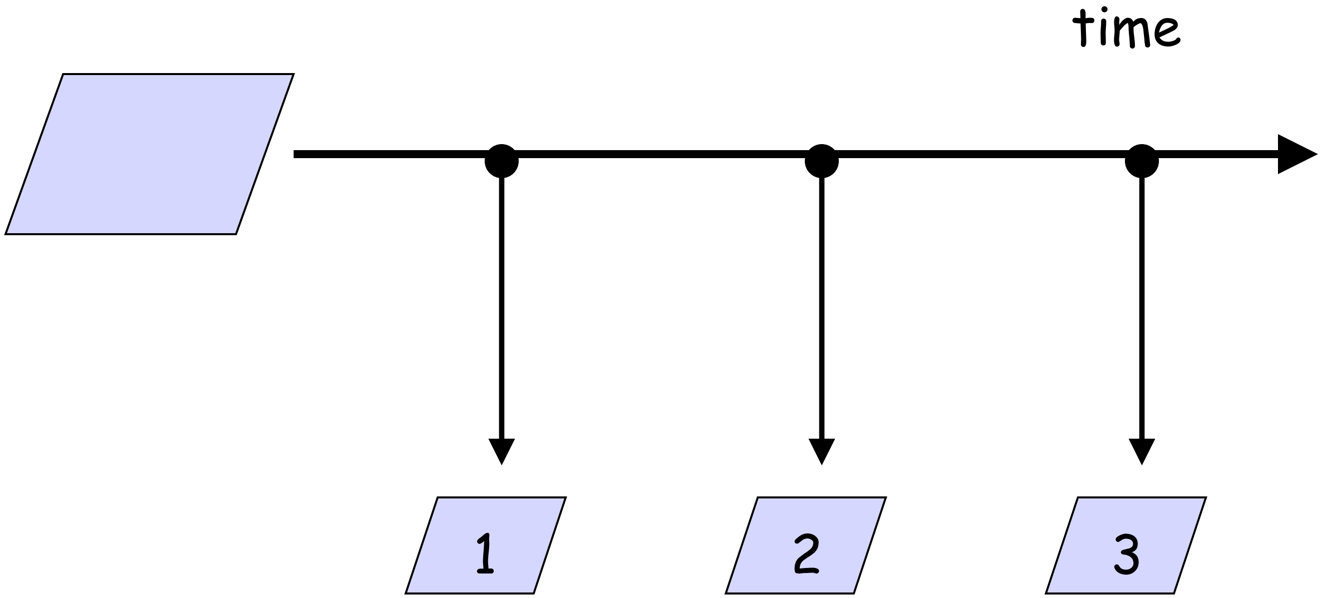
Checkpointing: Process Starts

checkpoint: the entire state of a program, saved in a file

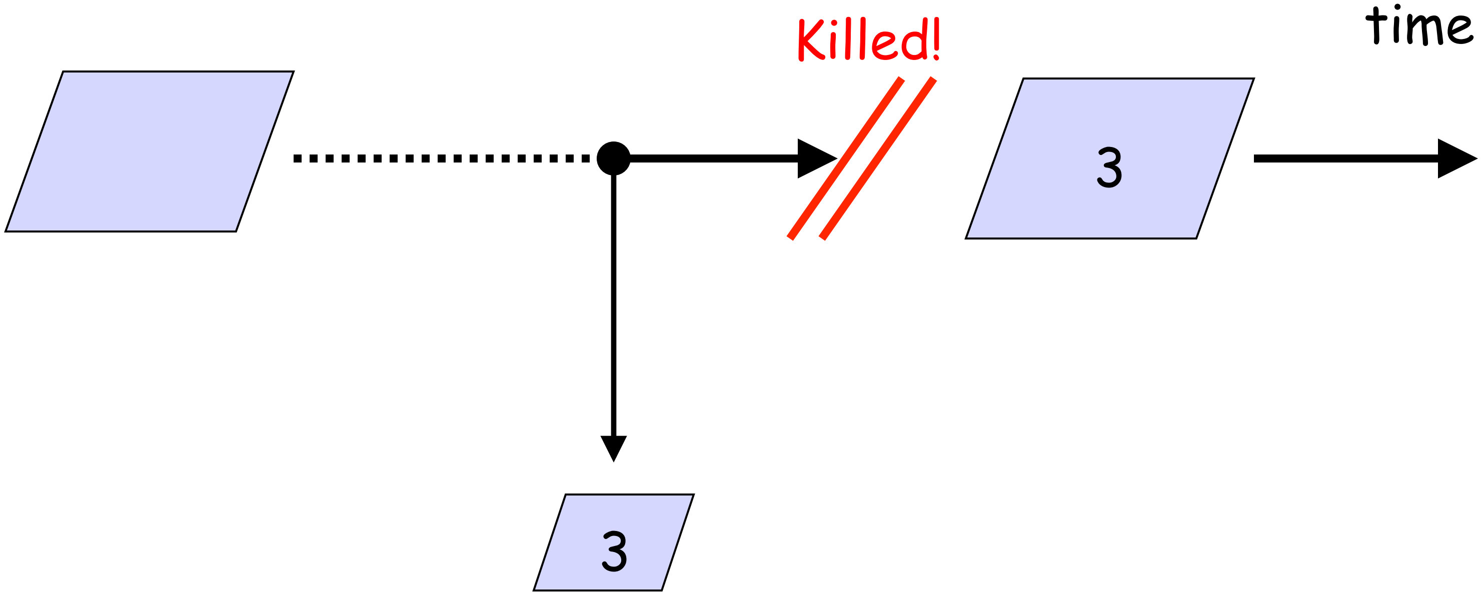
- CPU registers, memory image, I/O



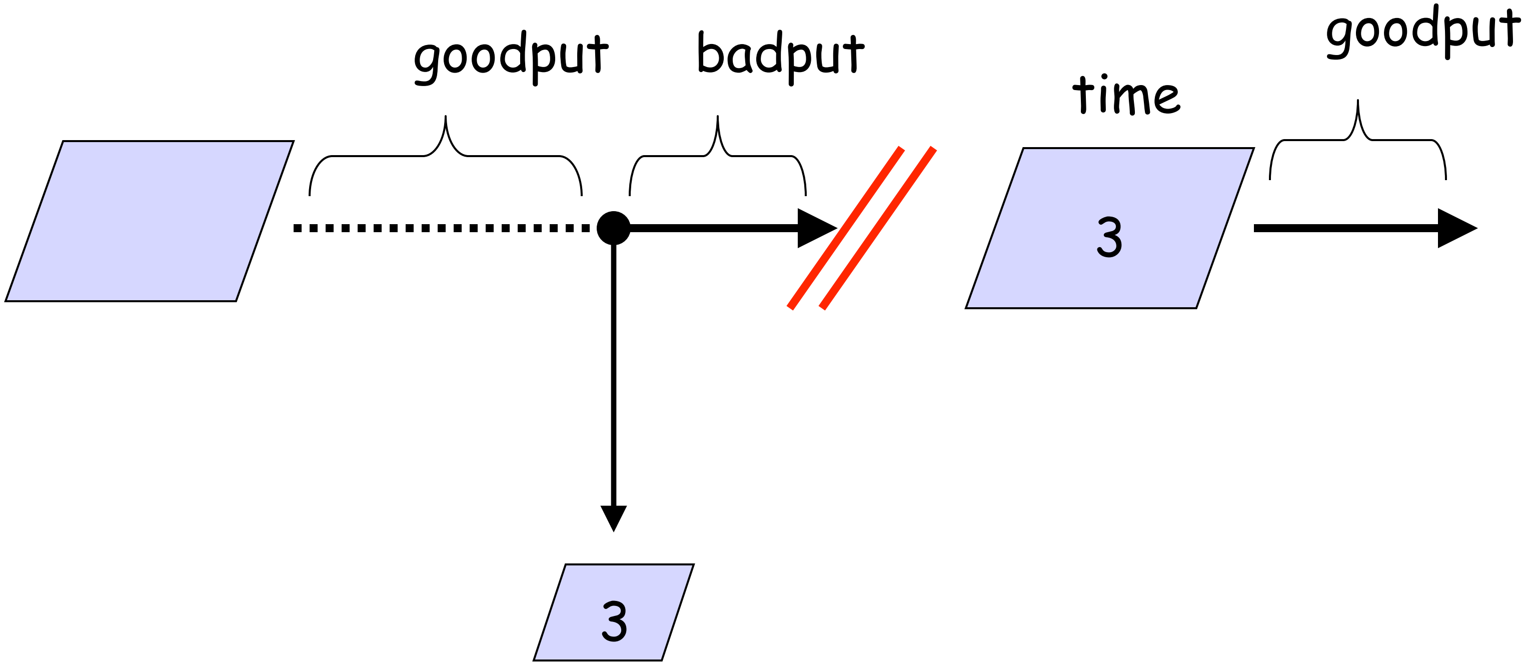
Checkpointing:
Process Checkpointed



Checkpointing: Process Killed



Checkpointing: Process Resumed



When will Condor checkpoint your job?

- Periodically, if desired
  - For fault tolerance
- When your job is preempted by a higher priority job
- When your job is vacated because the execution machine becomes busy
- When you explicitly run *condor_checkpoint*, *condor_vacate*, *condor_off* or *condor_restart* command

Making the Standard Universe Work

- The job **must be relinked** with Condor's standard universe support library
- To relink, place **condor_compile** in front of the command used to link the job:

```
% condor_compile gcc -o myjob myjob.c
```

- OR -

```
% condor_compile f77 -o myjob filea.f fileb.f
```

- OR -

```
% condor_compile make -f MyMakefile
```

Limitations of the Standard Universe

- Condor's checkpointing is not at the kernel level.
  - Standard Universe the job may not:
    - Fork()
    - Use kernel threads
    - Use some forms of IPC, such as pipes and shared memory
- Must have access to source code to relink
- Many typical scientific jobs are OK
- Only available on Linux platforms

Death of the Standard Universe

DMTCP & Parrot

- DMTCP (Checkpointing)
  - "Distributed MultiThreaded Checkpointing"
  - Developed at Northeastern University
  - http://dmtcp.sourceforge.net/
  - See Gene Cooperman's (Northeastern University) talk tomorrow @ 2:20
- Parrot (Remote I/O)
  - "Parrot is a tool for attaching existing programs to remote I/O system"
  - Developed by Doug Thain (now at Notre Dame)
  - http://www.cse.nd.edu/~ccl/software/parrot/
  - dthain@nd.edu

VM Universe

- Runs a virtual machine instance as a job
- VM Universe:
  - Job sandboxing
  - Checkpoint and migration
  - Safe elevation of privileges
  - Cross-platform

More on VM Universe

- Supports VMware, Xen, KVM
- Input files can be imported as CD-ROM image
- When VM shuts down, modified disk image is returned as job output

Example VMware Job

- This example uses the `vmware_dir` command to identify the location of the disk image to be executed as a Condor job.
- The contents of this directory are transferred to the machine assigned to execute the Condor job.

```
universe                     = vm
executable                   = vmware_sample_job
log                          = simple.vm.log.txt
vm_type                      = vmware
vm_memory                    = 64
vmware_dir                   = C:\condor-test
vmware_should_transfer_files = True
queue
```

- See Jaime's talk at 4:30 today for lots more!

Albert meets The Grid

- Albert also has access to grid resources he wants to use
  - He has certificates and access to Globus or other resources at remote institutions
- But Albert wants Condor's queue management features for his jobs!
- He installs Condor so he can submit "Grid Universe" jobs to Condor

“Grid” Universe

- All handled in your submit file
- Supports a number of “back end” types:
  - Globus: GT2, GT4
  - NorduGrid
  - UNICORE
  - Condor
  - PBS
  - LSF
  - EC2
  - NQS

the globus alliance
Condor High Throughput Computing
amazon webservices™
NORDUGRID
Grid Solution for Wide Area Computing and Data Handling
PBS
UNICORE

Grid Universe & Globus 2

- Used for a Globus GT2 back-end
  - “Condor-G”
- Format:

```
Grid_Resource = gt2 Head-Node
Globus_rsl = <RSL-String>
```

- Example:

```
Universe = grid
Grid_Resource = gt2 beak.cs.wisc.edu/jobmanager
Globus_rsl = (queue=long)(project=atom-smasher)
```

Grid Universe & Globus 4

- Used for a Globus GT4 back-end
- Format:

```
Grid_Resource = gt4 <Head-Node> <Scheduler-Type>
Globus_XML = <XML-String>
```

- Example:

```
Universe = grid
Grid_Resource = gt4 beak.cs.wisc.edu Condor
Globus_xml = <queue>long</queue><project>atom-
   smasher</project>
```

Grid Universe & Condor

- Used for a Condor back-end
  - "Condor-C"
- Format:

```
Grid_Resource = condor <Schedd-Name> <Collector-Name>
Remote_<param> = <value>
```

  - "Remote_" part is stripped off
- Example:

```
Universe = grid
Grid_Resource = condor beak condor.cs.wisc.edu
Remote_Universe = standard
```

Grid Universe & NorduGrid

- Used for a NorduGrid back-end

```
Grid_Resource = nordugrid <Host-Name>
```

- Example:

```
Universe = grid
Grid_Resource = nordugrid ngrid.cs.wisc.edu
```

Grid Universe & UNICORE

- Used for a UNICORE back-end
- Format:

```
Grid_Resource = unicore <USite> <VSite>
```

- Example:

```
Universe = grid
Grid_Resource = unicore uhost.cs.wisc.edu vhost
```

Grid Universe & PBS

- Used for a PBS back-end
- Format:

```
Grid_Resource = pbs
```

- Example:

```
Universe = grid
Grid_Resource = pbs
```

Grid Universe & LSF

- Used for a LSF back-end
- Format:

```
Grid_Resource = lsf
```

- Example:

```
Universe = grid
Grid_Resource = lsf
```

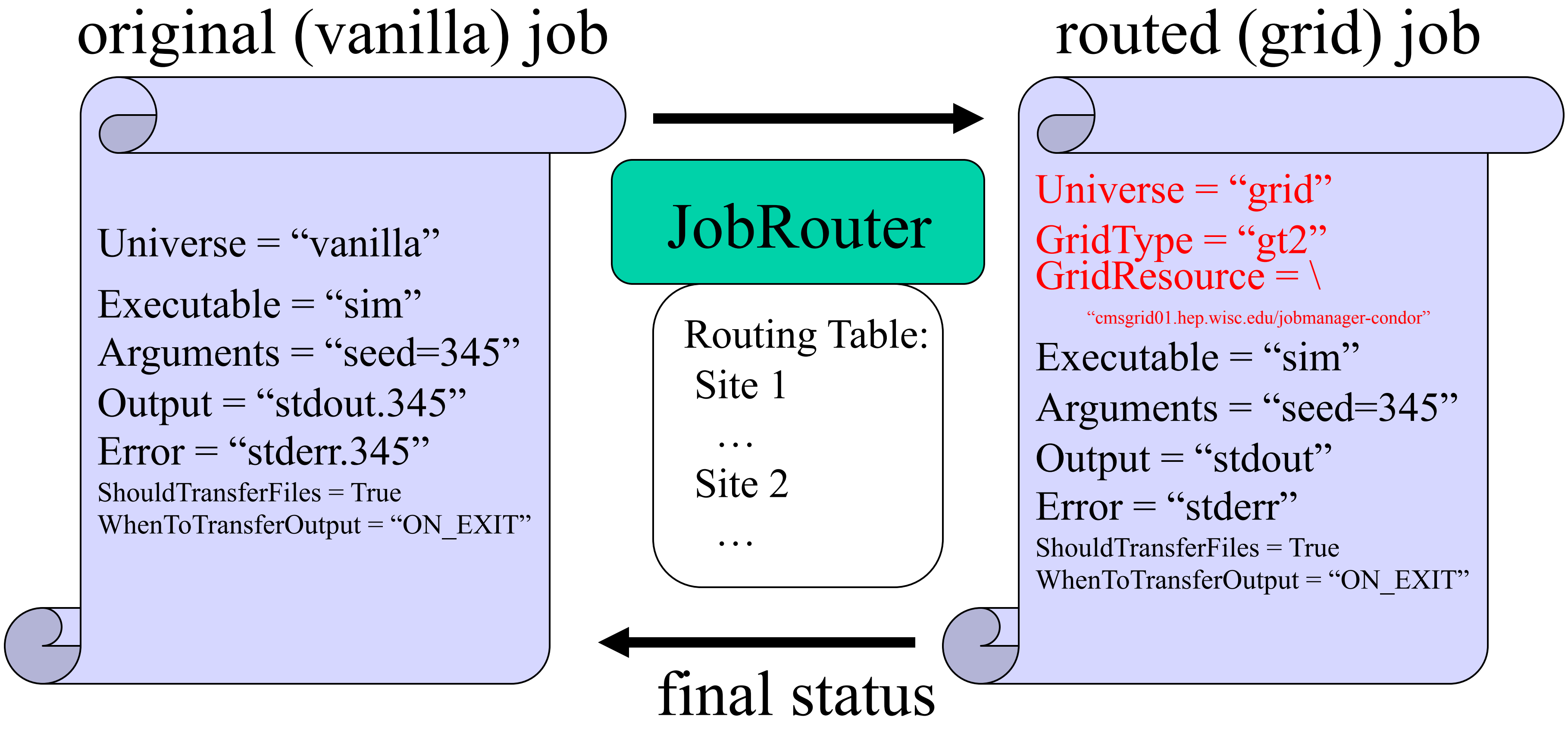
Credential Management

- Condor will do The Right Thing™ with your X509 certificate and proxy
- Override default proxy:
  - `X509UserProxy = /home/einstein/other/proxy`
- Proxy may expire before jobs finish executing
  - Condor can use MyProxy to renew your proxy
  - When a new proxy is available, Condor will forward the renewed proxy to the job
  - This works for non-grid jobs, too

JobRouter
A Flexible Job Transformer

- Acts upon jobs in queue
- Policy controls when
  - (jobs currently routed to site X) < max
  - (*idle* jobs routed to site X) < max
  - (rate of recent failure at site X) < max
- And how
  - change attribute values (e.g. Universe)
  - insert new attributes (e.g. GridResource)
  - other arbitrary actions in hooks

Example: sending excess vanilla jobs to a grid site



original (vanilla) job

```
Universe = “vanilla”
Executable = “sim”
Arguments = “seed=345”
Output = “stdout.345”
Error = “stderr.345”
ShouldTransferFiles = True
WhenToTransferOutput = “ON_EXIT”
```

JobRouter

Routing Table:
Site 1
…
Site 2
…

routed (grid) job

```
Universe = “grid”
GridType = “gt2”
GridResource = \
    “cmsgrid01.hep.wisc.edu/jobmanager-condor”
Executable = “sim”
Arguments = “seed=345”
Output = “stdout”
Error = “stderr”
ShouldTransferFiles = True
WhenToTransferOutput = “ON_EXIT”
```

final status

JobRouter vs. Glidein

- Glidein - Condor overlays the grid
  - Job never waits in remote queue
  - Full job management (e.g. **`condor_ssh_to_job`**)
  - Private networks doable, but add to complexity
  - Need something to submit glideins on demand
- JobRouter
  - Some jobs wait in remote queue (**`MaxIdleJobs`**)
  - Job must be compatible with target grid semantics
  - Job managed by remote batch system
  - Simple to set up, fully automatic to run

Condor Universes: Scheduler and Local

- Scheduler Universe
  - Plug in a meta-scheduler
  - Similar to Globus's fork job manager
  - Developed for DAGMan (next slides)
- Local
  - Very similar to vanilla, but jobs run on the local host
  - Has more control over jobs than scheduler universe

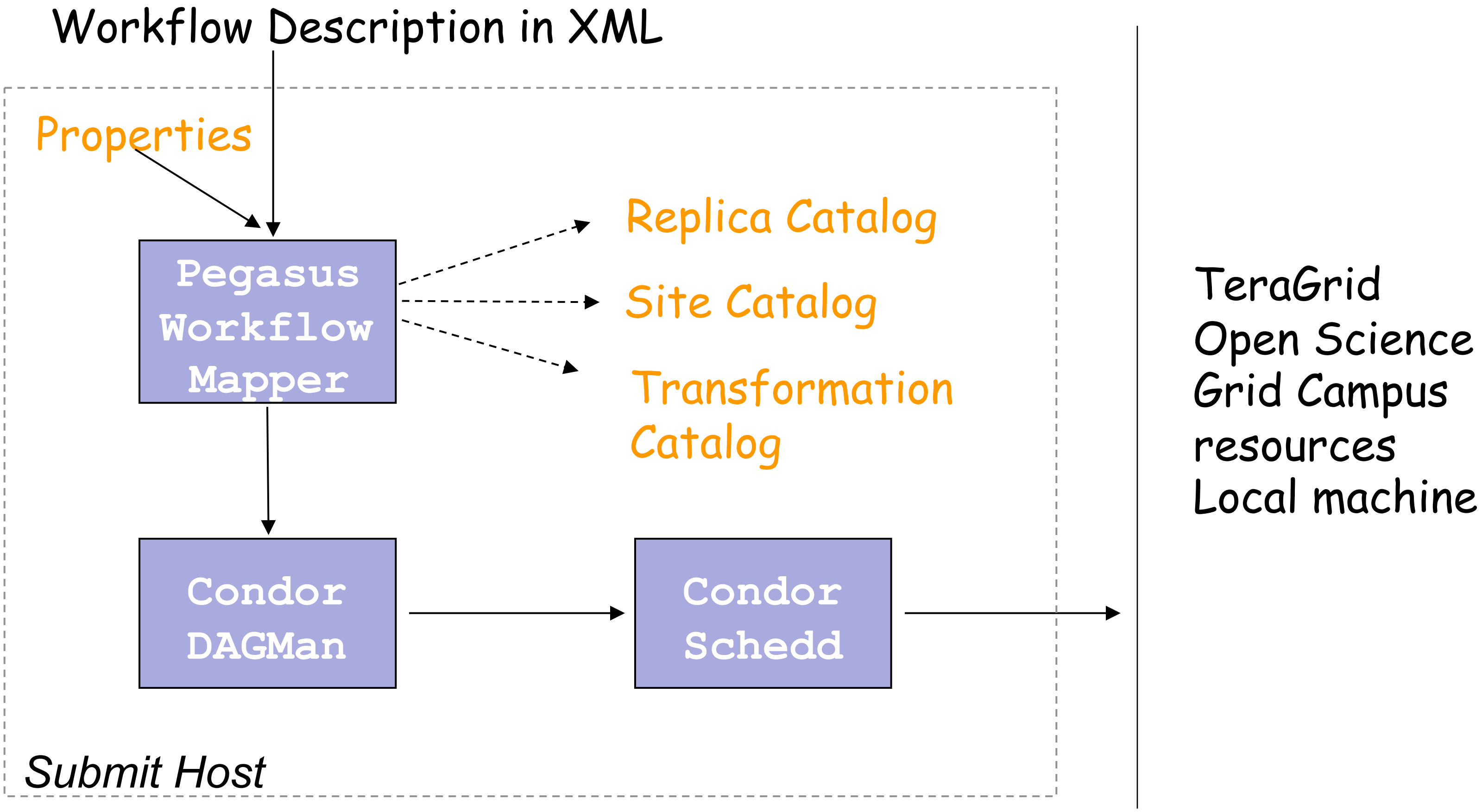
My jobs have have dependencies…

Can Condor help solve my dependency problems?

Workflows – Launch and Forget

- A single workflow can take days, weeks or even months
- Automates tasks user *could* perform manually…
  - But WMS takes care of automatically
- Includes features such as retries in the case of failures – avoids the need for user intervention
- The workflow itself can include error checking
- The result: one user action can utilize many resources while maintaining complex job inter-dependencies and data flows
- Maximizes compute resources / human time

Pegasus WMS



Pegasus WMS restructures and optimizes the workflow, provides reliability

DAGMan Configuration

- Condor configuration files
- Environment variables:
  - `_CONDOR_<macroname>`
- DAGMan configuration file
- `condor_submit_dag` command line

Submit the DAG

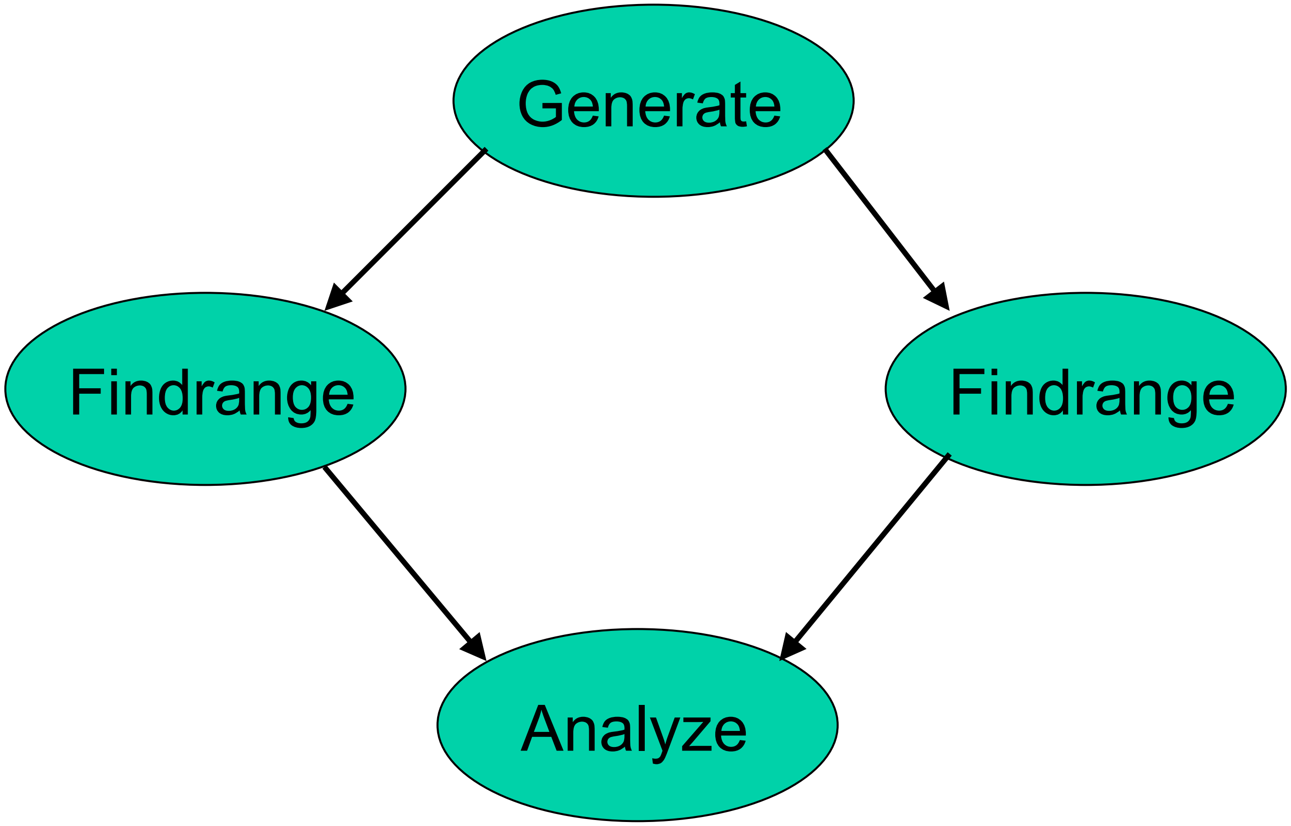
- Run:
  - `condor_submit_dag <file>`
- Creates a Condor submit file for DAGMan
- Also submits it to Condor
  - Unless **`–no_submit`** option is given
- **`-f`** option forces overwriting of existing files

Condor Monitoring

- Monitoring your DAG
  - condor_q [–dag] [*name*]
  - dagman.out file

```
[einstein@submit ~]$ condor_q –dag train15

-- Submitter: einstein@cosmos.phys.wisc.edu : <128.9.72.178:43684>
 ID     OWNER/NODENAME SUBMITTED   RUN_TIME ST PRI SIZE CMD
1860.0 train15         5/31 10:59 0+00:00:26 R 0   9.8  condor_dagman -f -
1861.0 |-Setup         5/31 10:59 0+00:00:12 R 0   9.8  nodejob Setup node

2 jobs; 0 idle, 2 running, 0 held
```

Exercise 2.6 - A simple DAG

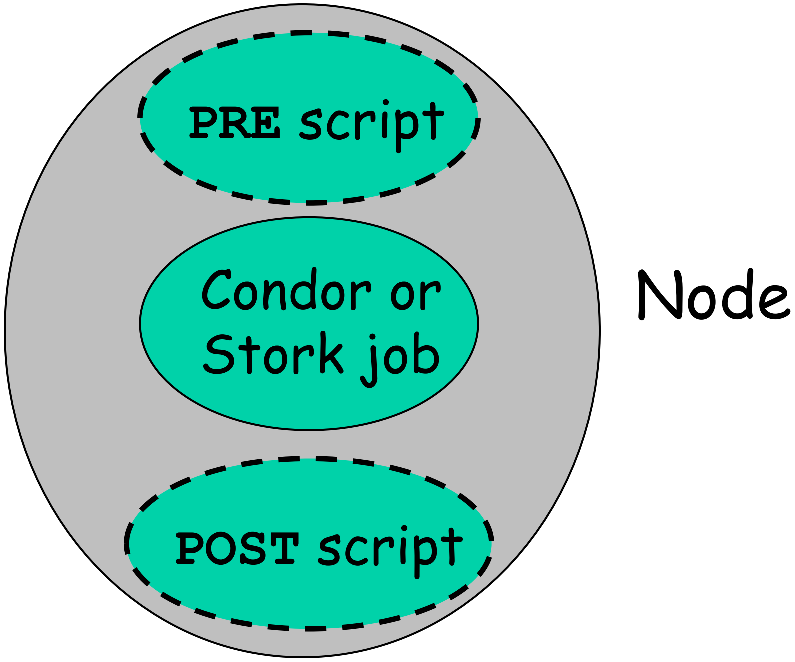


DAG file

- Defines the DAG shown previously
- Node names *are* case-sensitive
- Keywords are not case-sensitive

```
# Simple DAG file.

JOB Generate generate.submit
JOB Findrange1 findrange1.submit
JOB Findrange2 findrange2.submit
JOB Analyze analyze.submit
PARENT Generate CHILD Findrange1 Findrange2
PARENT Findrange1 Findrange2 CHILD Analyze
```

DAG node

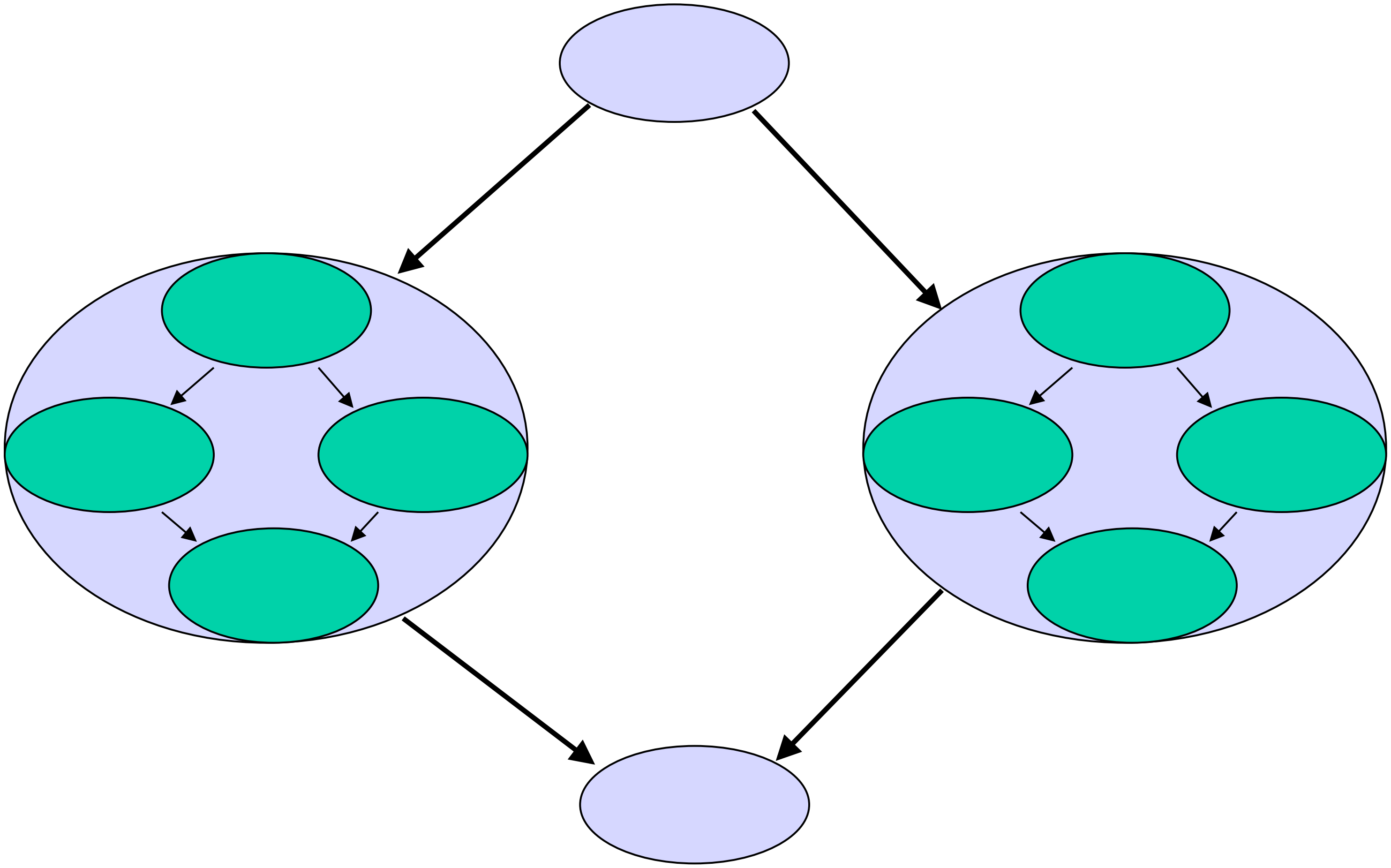


- Treated as a unit
- Job or POST script determines node success or failure

PRE/POST in DAGMan scripts

- `SCRIPT PRE|POST` *`node script`* `[`*`arguments`*`]`
- All scripts run on submit machine
- If `PRE` script fails, node fails w/o running job or `POST` script (for now…)
- If job succeeds or fails, `POST` script is run
- If `POST` script fails, node fails
- Special macros:
  - `$JOB`
  - `$RETRY`
  - `$JOBID` (`POST` only)
  - `$RETURN` (`POST` only)

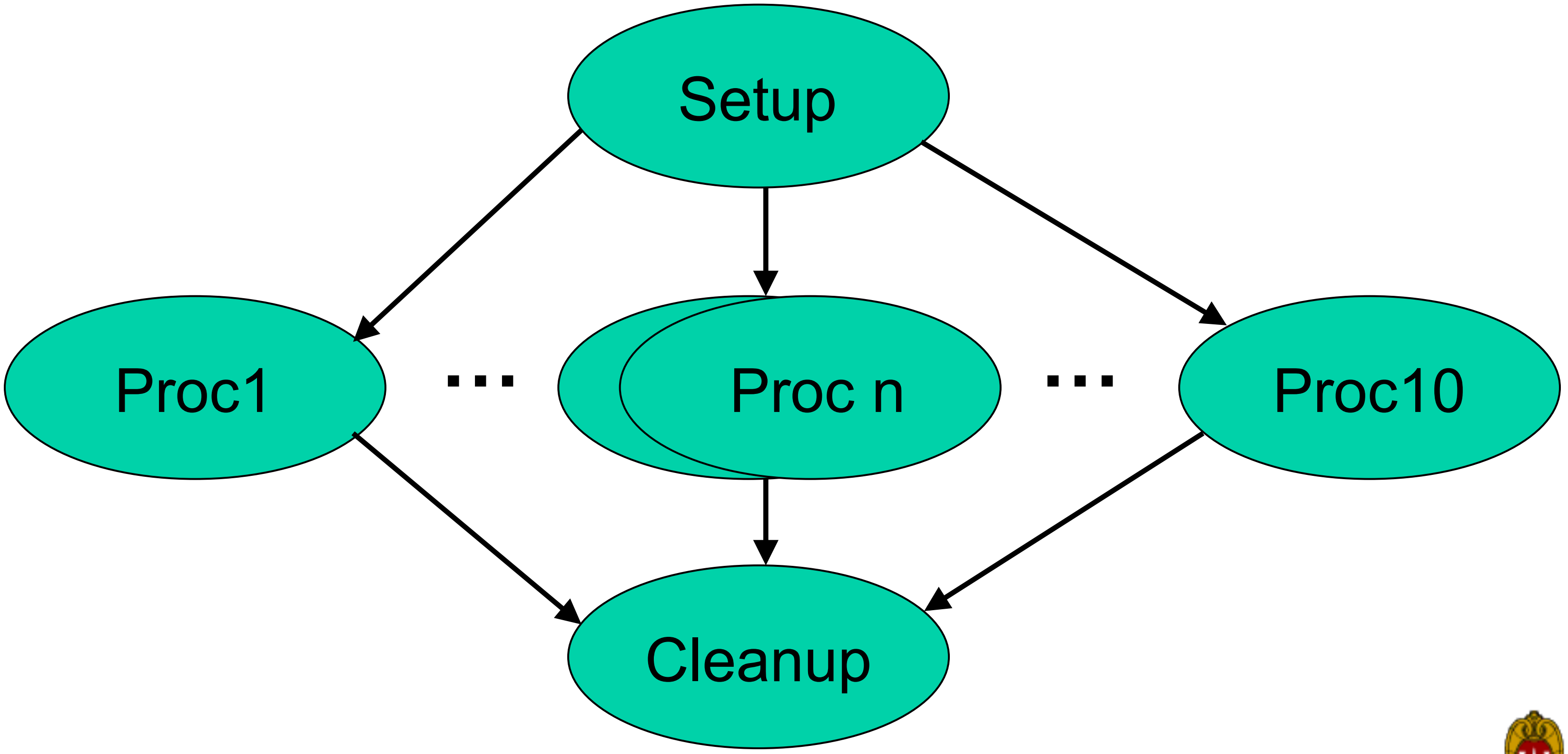
Nested DAGs



VARS (per-node variables)

- **VARS *JobName* *macroname*="*string*" [*macroname*="*string*"...]**
- Macroname can only contain alphanumeric characters and underscore
- Value can't contain single quotes; double quotes must be escaped
- Macronames cannot begin with "**queue**"
- Macronames are not case-sensitive

Exercise 3.4 – VARS and CONFIG



Exercise 3.4, continued

```
[einstein@submit ~]$ cat dagman/vars/vars.dag
# DAG to illustrate VARS and CONFIG.

CONFIG vars.config

JOB Setup setup.submit
SCRIPT PRE Setup setup.pre

JOB Proc1 pijob.submit
VARS Proc1 ARGS = "-sleep 60 -trials 10000 -seed 1234567"
PARENT Setup CHILD Proc1

JOB Proc2 pijob.submit
VARS Proc2 ARGS = "-sleep 80 -trials 20000 -seed 7654321"
PARENT Setup CHILD Proc2

JOB Proc3 pijob.submit
PARENT Setup CHILD Proc3
VARS Proc3 ARGS = "YOUR ARGS HERE"
[....]
```

Exercise 3.4, continued

```
[einstein@submit ~]$  cat dagman/vars/vars.config
# DAGMan configuration file for vars.dag.

DAGMAN_MAX_JOBS_SUBMITTED = 3
DAGMAN_STARTUP_CYCLE_DETECT = true
DAGMAN_SUBMIT_DEPTH_FIRST = true

[einstein@submit ~]$  cat dagman/vars/pijob.submit
# Simple Condor submit file.

Executable      = ../pi/pi
Universe        = scheduler
#Error          = pi.err.$(cluster)
Output          = output/pi.out.$(cluster)
Getenv          = true
Log             = pi.log

Arguments       = $(ARGS)
Notification    = never
Queue
```

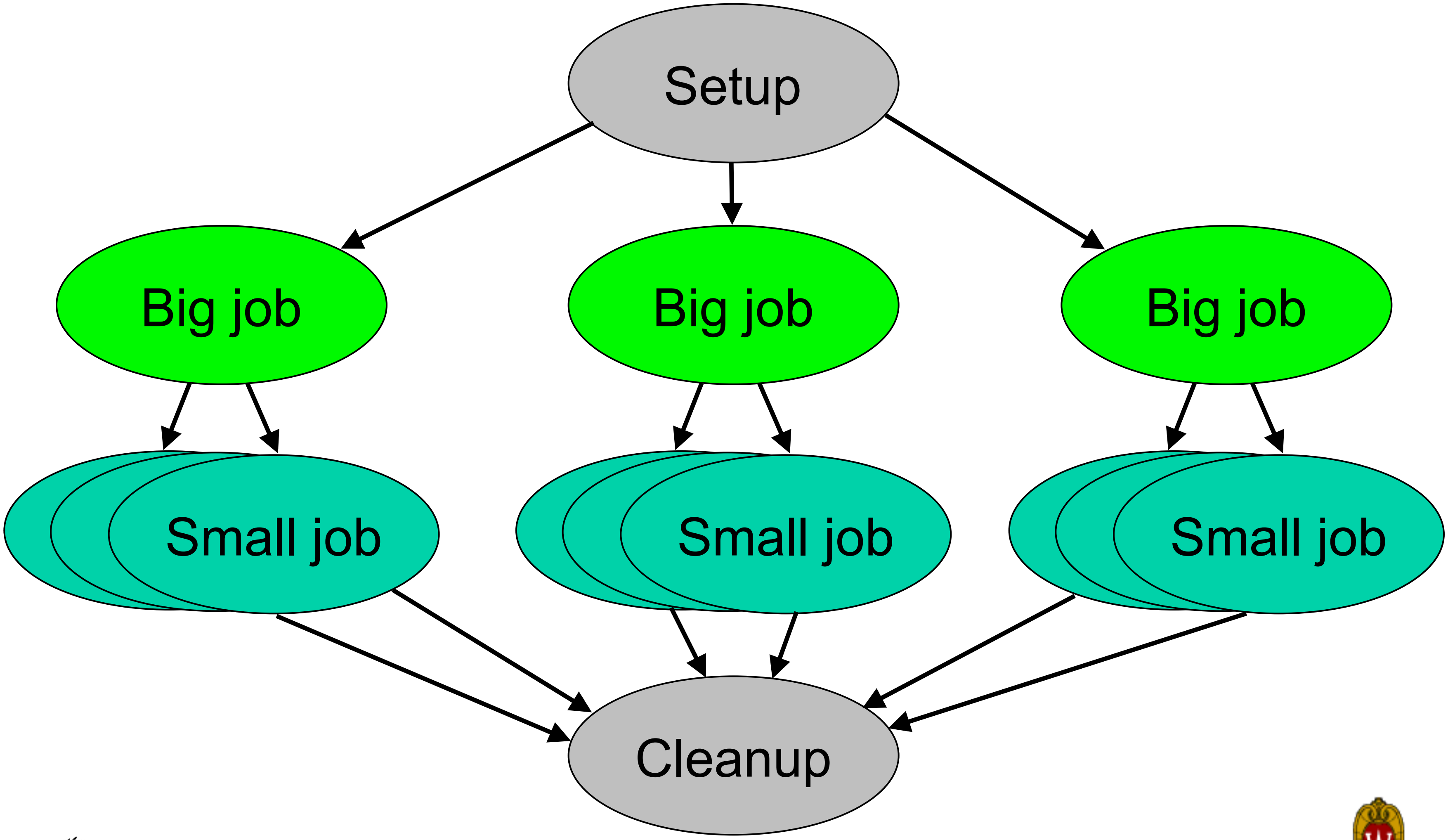
Throttling in DAGMan

- **`Maxjobs`** (limits jobs in queue/running)
- **`Maxidle`** (limits idle jobs)
- **`Maxpre`** (limits PRE scripts)
- **`Maxpost`** (limits POST scripts)
- *All* limits are *per **DAGMan***, not global for the pool
- Limits can be specified as arguments to **`condor_submit_dag`** or in configuration

Throttling by category

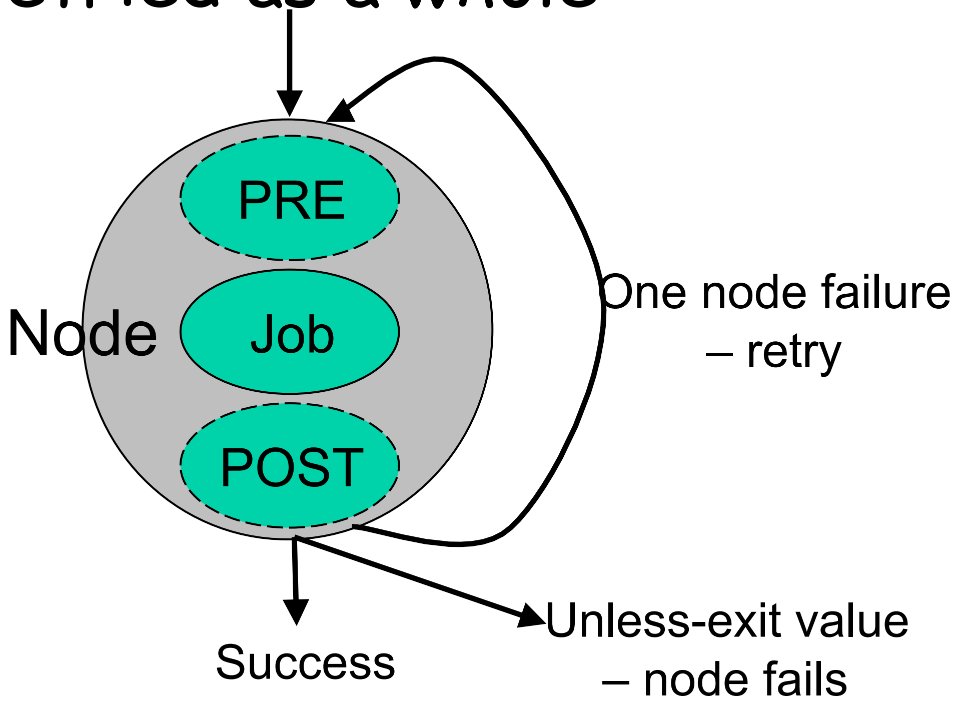
- **`CATEGORY`** ***`JobName CategoryName`***
- **`MAXJOBS`** ***`CategoryName MaxJobsValue`***
- Applies the **`maxjobs`** setting to only jobs assigned to the given category
- Global throttles still apply
- Useful with different types of jobs that cause different loads

Node categories



Node retries

- **RETRY *JobName NumberOfRetries* [UNLESS-EXIT *value*]**
- Node is retried as a whole



Node Categories and Retries

```
[einstein@submit ~]$ more montage.dag
# DAG to illustrate node categories/category throttles.

MAXJOBS projection 2

CATEGORY mProjectPP_ID000002 projection
JOB mProjectPP_ID000002 mProjectPP_ID000002.sub
SCRIPT POST mProjectPP_ID000002 /nfs/software/pegasus/
   default/bin/exitpost  …..
RETRY mProjectPP_ID000002 2
...
```

Rescue DAG

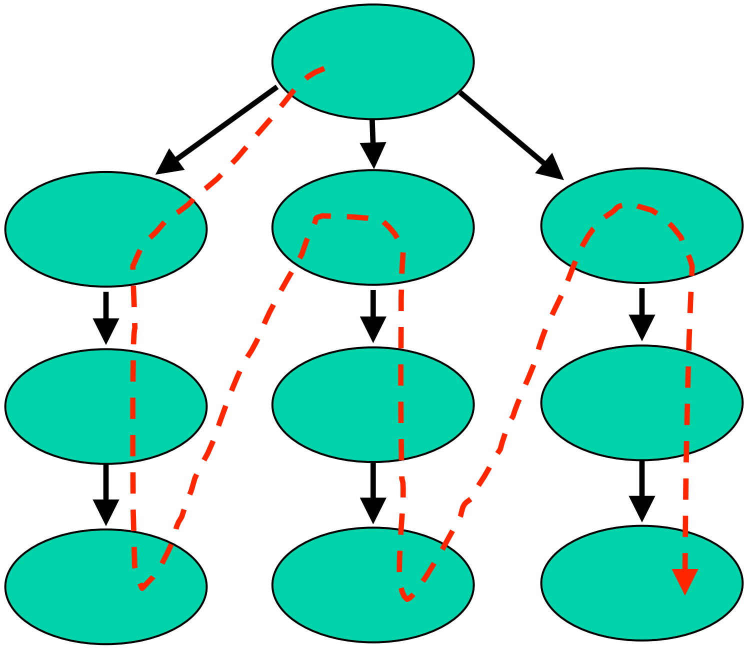
- Generated when a node fails or DAGMan is condor_rm'ed
- Saves state of DAG
- Run the rescue DAG to restart from where you left off

Recovery/bootstrap mode

- Most commonly, after condor_hold/ condor_release of DAGMan
- Also after DAGMan crash/restart
- Restores DAG state by reading node job logs

Depth-first DAG traversal

- Get results more quickly
- Possibly clean up intermediate files more quickly
- **`DAGMAN_SUBMIT_DEPTH_FIRST=True`**



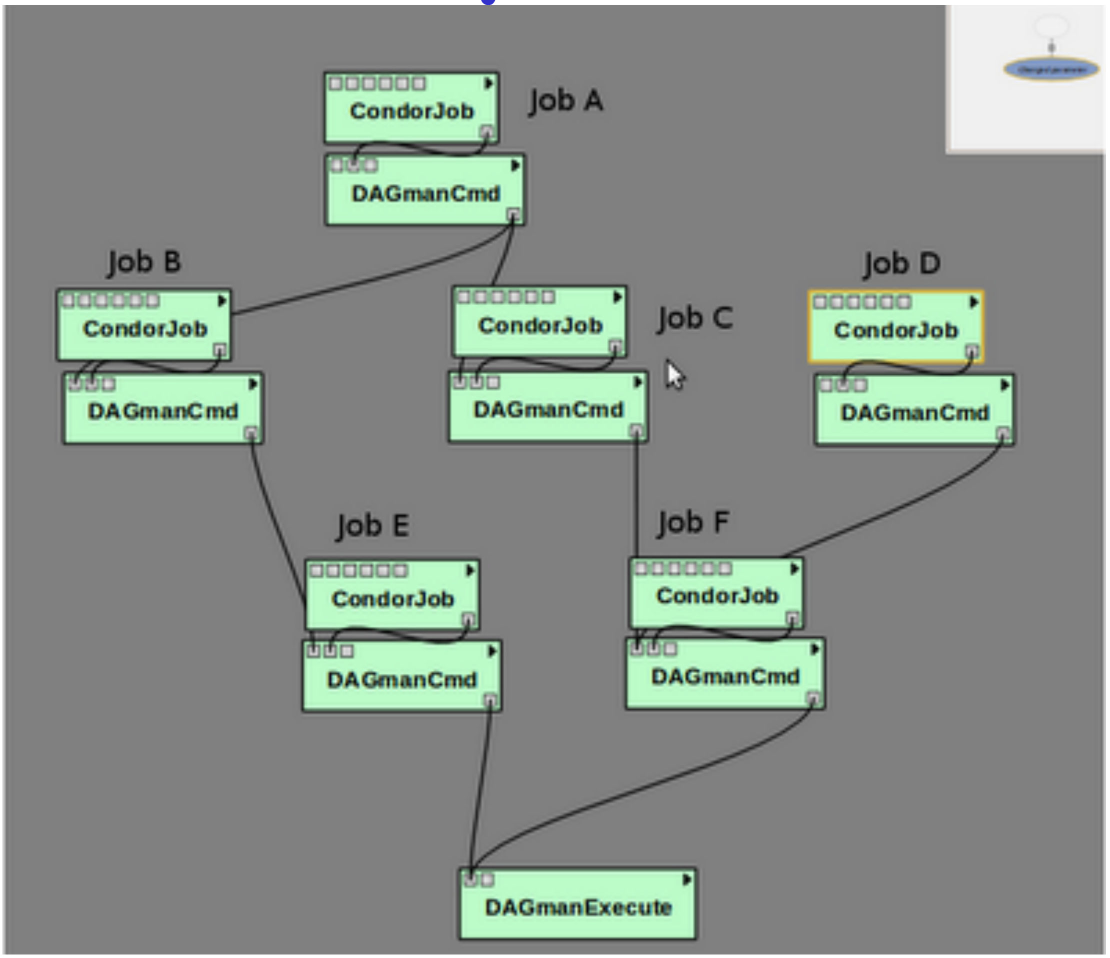
DAG node priorities

- `PRIORITY` ***`JobName PriorityValue`***
- Determines order of submission of ready nodes
- Does *not* violate/change DAG semantics
- Mostly useful when DAG is throttled
- Higher Numerical value equals higher priority

VisTrails

- An open-source scientific workflow and provenance management system developed at the University of Utah
- We are working with the VisTrails group to add DAGMan support to VisTrails
- See www.vistrails.org

VisTrails, continued



Relevant Links

- DAGMan: www.cs.wisc.edu/condor/dagman

- For more questions: condor_admin@cs.wisc.edu

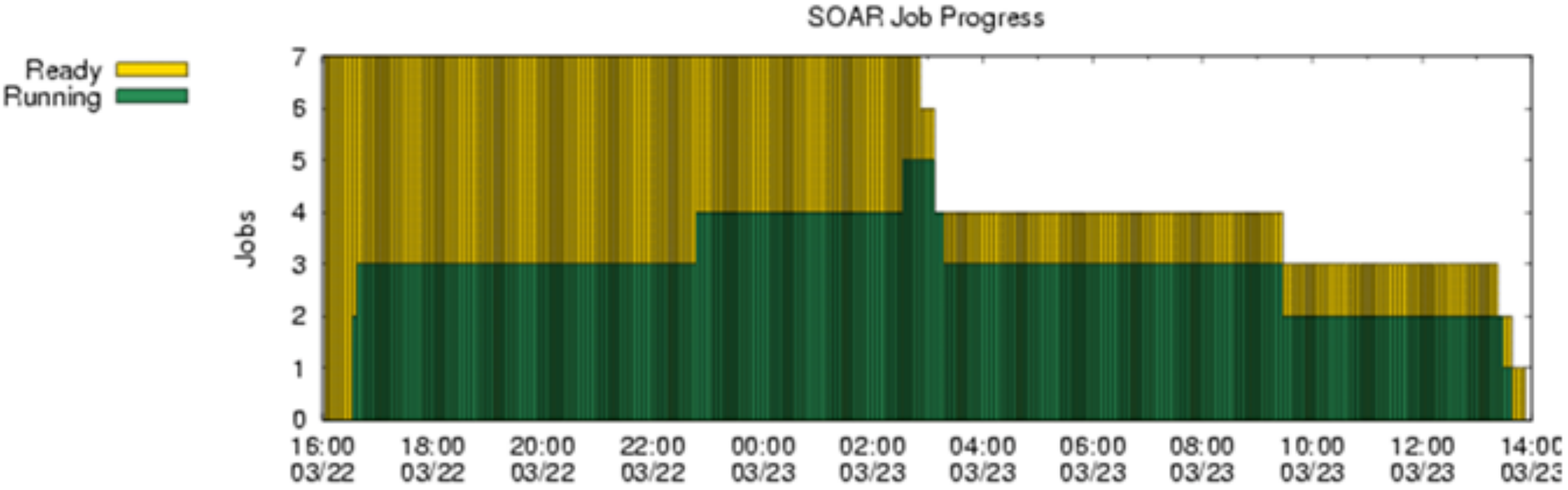
SOAR

- What is SOAR?
  - A System Of Automatic Runs
  - A framework for collecting N jobs into a DAG, submitting them to Condor and tracking the run
  - A tool that lets one make these jobs complex workflows
  - An environment to control production of large sets of data
  - A simple web interface for tracking runs and downloading results.

How does SOAR work?

- SOAR:
  - Sweeps drop box for new job data
  - Creates the run
  - Periodically creates plot and reports showing progress of run
  - After the DAG completes, SOAR makes your results available through the web interface

View SOAR Job Progress



SOAR

- When is it best used?
  - When a production environment is desired.
  - When a researcher is Linux challenged
  - When each job is a complex DAG in itself.
- Web peak: www.submit.chtc.wisc.edu/SOAR/
- Info: Bill Taylor bt@cs.wisc.edu CHTC Staff

General User Commands

condor_status	View Pool Status
condor_q	View Job Queue
condor_submit	Submit new Jobs
condor_rm	Remove Jobs
condor_prio	Intra-User Prios
condor_history	Completed Job Info
condor_submit_dag	Submit new DAG
condor_checkpoint	Force a checkpoint
condor_compile	Link Condor library

Condor Job Universes

- Vanilla Universe
- Standard Universe
- Grid Universe
- Scheduler Universe
- Local Universe
- Virtual Machine Universe
- Java Universe

- Parallel Universe
  - MPICH-1
  - MPICH-2
  - LAM
  - …

Why have a special Universe for Java jobs?

- Java Universe provides more than just inserting "java" at the start of the execute line of a vanilla job:
  - Knows which machines have a JVM installed
  - Knows the location, version, and performance of JVM on each machine
  - Knows about jar files, etc.
  - Provides more information about Java job completion than just JVM exit code
    - Program runs in a Java wrapper, allowing Condor to report Java exceptions, etc.

Universe Java Job

```
# Example Java Universe Submit file
Universe   = java
Executable = Main.class
jar_files  = MyLibrary.jar
Input      = infile
Output     = outfile
Arguments  = Main 1 2 3
Queue
```

Java support, cont.

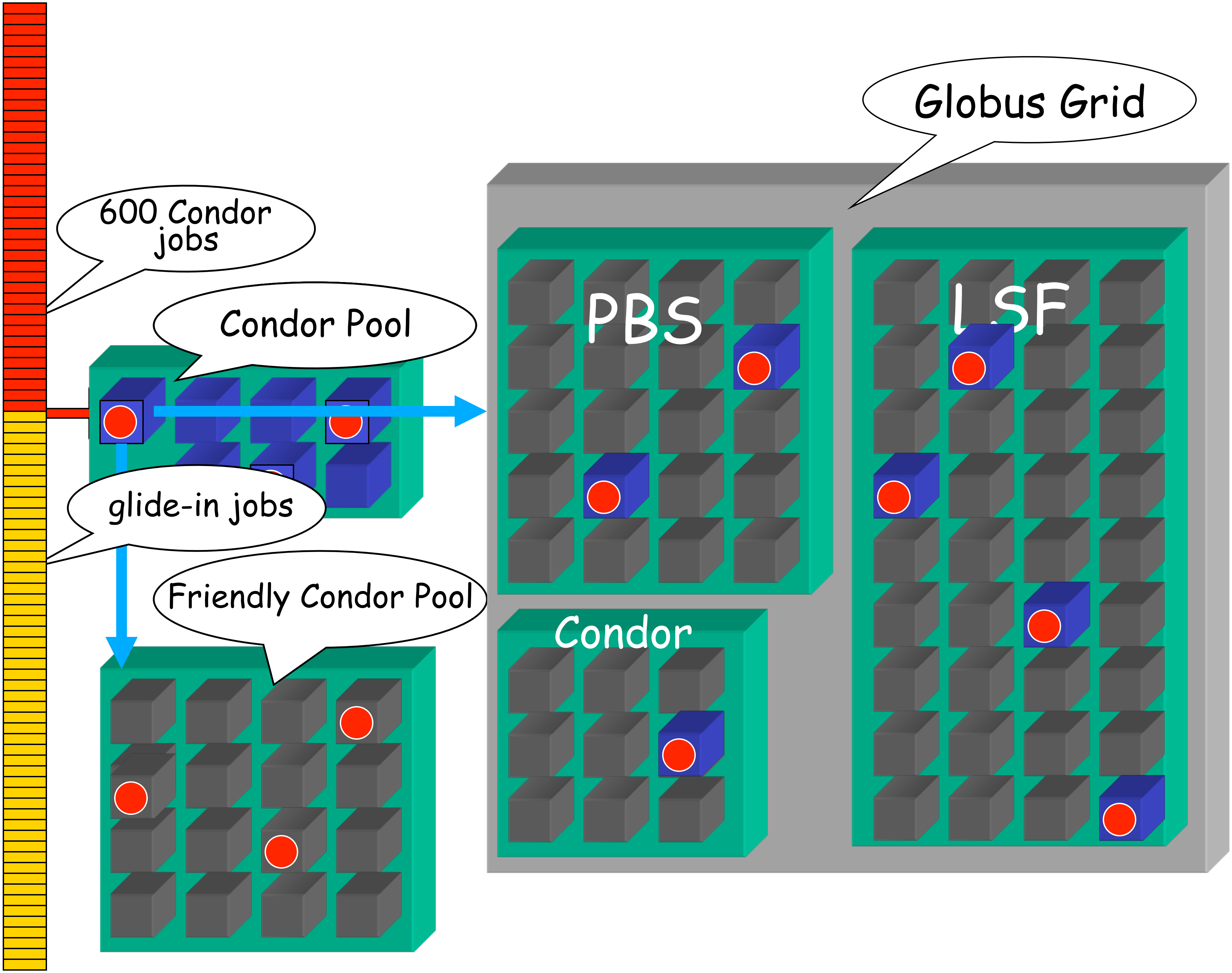
```
bash-2.05a$ condor_status -java
 Name        JavaVendor  Ver    State    Actv LoadAv Mem
abulafia.cs Sun Microsy 1.5.0_ Claimed   Busy 0.180   503
acme.cs.wis Sun Microsy 1.5.0_ Unclaimed Idle 0.000   503
adelie01.cs Sun Microsy 1.5.0_ Claimed   Busy 0.000  1002
adelie02.cs Sun Microsy 1.5.0_ Claimed   Busy 0.000  1002
…
                Total Owner Claimed Unclaimed Matched Preempting
    INTEL/LINUX  965  179    516       250      20          0
  INTEL/WINNT50  102    6     65        31       0          0
SUN4u/SOLARIS28    1    0      0         1       0          0
   X86_64/LINUX  128    2    106        20       0          0

          Total  1196 187     687       302      20          0
```

Albert wants Condor features on remote resources

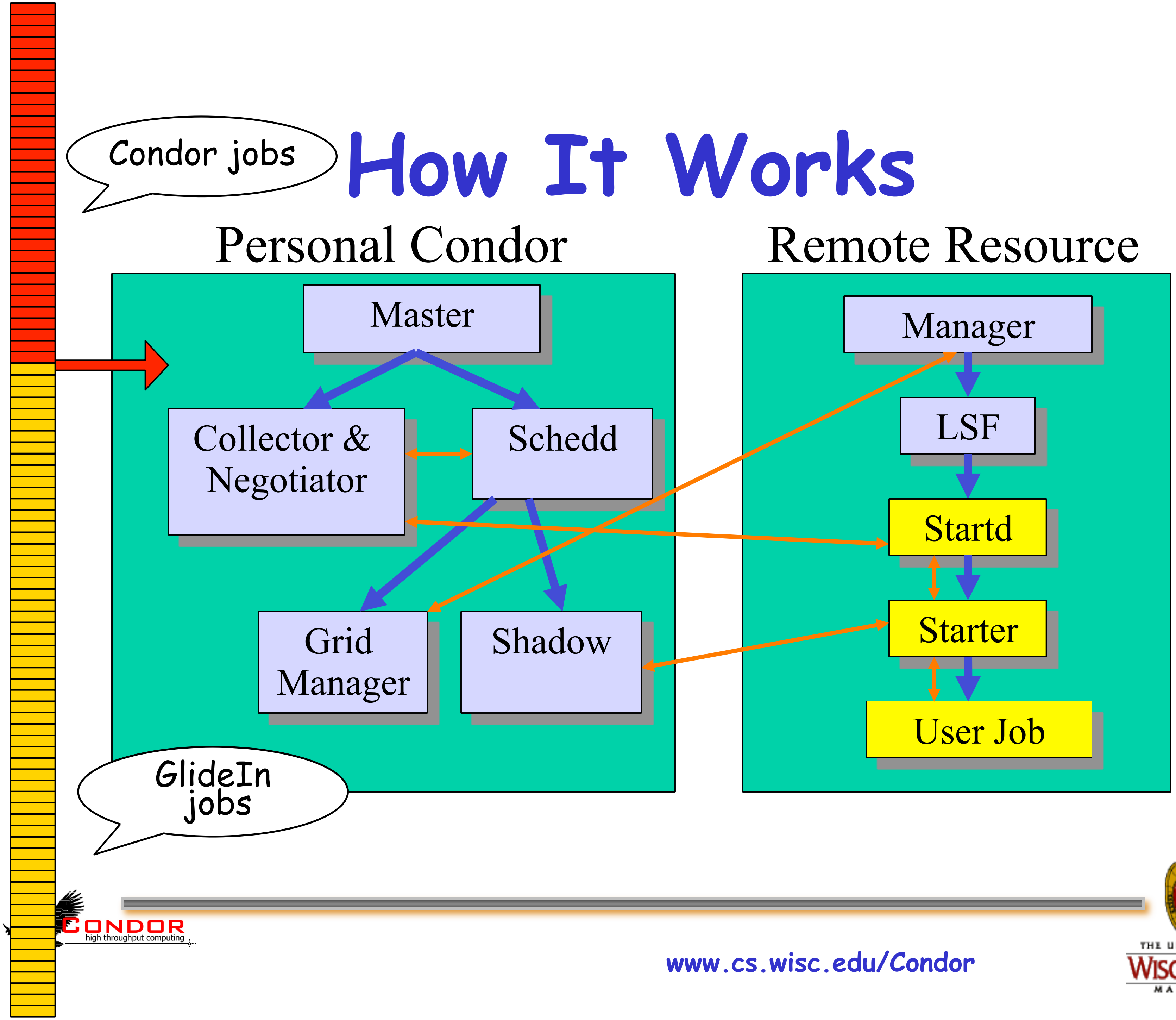
- He wants to run standard universe jobs on Grid-managed resources
  - For matchmaking and dynamic scheduling of jobs
  - For job checkpointing and migration
  - For remote system calls

Condor Glide

- Albert can use the Grid Universe to run Condor daemons on Grid resources
- When the resources run these GlideIn jobs, they will temporarily join his Condor Pool
- He can then submit Standard, Vanilla, PVM, or MPI Universe jobs and they will be matched and run on the remote resources
- Currently only supports Globus GT2
  - We hope to fix this limitation


Globus Grid
600 Condor jobs
Condor Pool
PBS
LSF
glide-in jobs
Friendly Condor Pool
Condor

How It Works


Condor jobs
Personal Condor
Remote Resource
Master
Collector &
Negotiator
Schedd
Grid
Manager
Shadow
Manager
LSF
Startd
Starter
User Job
GlideIn
jobs

GlideIn Concerns

- What if the remote resource kills my GlideIn job?
  - That resource will disappear from your pool and your jobs will be rescheduled on other machines
  - Standard universe jobs will resume from their last checkpoint like usual
- What if all my jobs are completed before a GlideIn job runs?
  - If a GlideIn Condor daemon is not matched with a job in 10 minutes, it terminates, freeing the resource

In Review

With Condor’s help, Albert can:

- Manage his compute job workload
- Access local machines
- Access remote Condor Pools via flocking
- Access remote compute resources on the Grid via “Grid Universe” jobs
- Carve out his own personal Condor Pool from the Grid with GlideIn technology

Administrator Commands

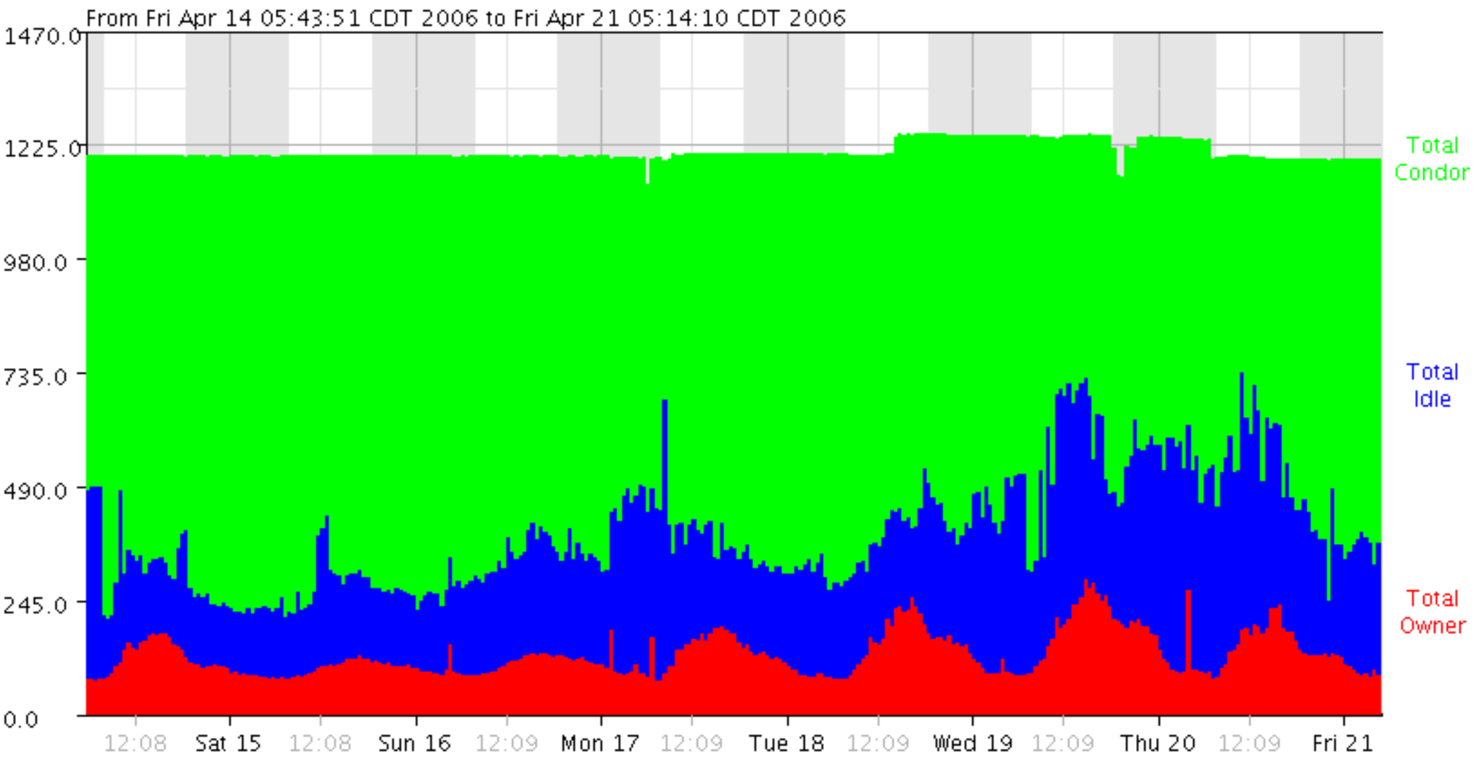
condor_vacate	Leave a machine now
condor_on	Start Condor
condor_off	Stop Condor
condor_reconfig	Reconfig on-the-fly
condor_config_val	View/set config
condor_userprio	User Priorities
condor_stats	View detailed usage accounting stats

My boss wants to watch what Condor is doing

Use CondorView!

- Provides visual graphs of current and past utilization
- Data is derived from Condor's own accounting statistics
- Interactive Java applet
- Quickly and easily view:
  - How much Condor is being used
  - How many cycles are being delivered
  - Who is using them
  - Utilization by machine platform or by user

CondorView Usage Graph



I could also talk lots about…

- GCB: Living with firewalls & private networks
- Federated Grids/Clusters
- APIs and Portals
- MW
- Database Support (Quill)
- High Availability Fail-over
- Compute On-Demand (COD)
- Dynamic Pool Creation (“Glide-in”)
- Role-based prioritization and accounting
- Strong security, incl privilege separation
- Data movement scheduling in workflows
- …

Thank you!

Check us out on the Web:
http://www.condorproject.org

Email:
condor-admin@cs.wisc.edu