An Introduction to HTCondor

Guy Tel-Zur

The following slides are based on tutorials by the HTCondor team

http://research.cs.wisc.edu/htcondor/

HTCondor

High Throughput Computing

Last update: 3/1/2022

The IEEE Computer Society Technical Committee on Distributed Processing (TCDP) has named Professor **Miron Livny** from University of Wisconsin as the recipient of the 2020 Outstanding Technical Achievement Award. The citation for Professor Livny is for his Influential Contributions of the Condor system to Distributed and High Throughput Computing.

Miron Livny is the John P. Morgridge Professor of Computer Science at the University of Wisconsin – Madison where he is serving as the Director of the Center for High Throughput Computing and the Lead Investigator of Computing Technologies at the Morgridge Institute for Research. He has been serving as the Technical Director of the Open Science Grid and has been collaborating for more than three decades with researchers from a wide spectrum of science domains. Two of these collaborations – the LHC experiments and the LIGO collaboration – were recently awarded Nobel prizes in physics (2013 and 2017 respectively). Livny's PhD work on Load Balancing Algorithms for Distributed Processing Systems laid the foundation for research in adaptive load sharing algorithms. It also provided the underpinning principals of sharing and local autonomy for the Condor system that was first deployed in 1985. The original Condor paper was published in ICDCS 1988 and was awarded as one of the two ICDCS High Impact Papers in the 40-year anniversary of the conference in 2020. Experience with the Condor system led him to pioneer the area of High Throughput Computing (HTC) in the mid 90's. Livny has been promoting the adoption of HTC methodologies and technologies to advance scientific discovery ever since. The widely adopted HTCondor Software Suite embodies four decades of research, software development and experimental evaluation of distributed computing frameworks and technologies. The suite includes novel concepts like the ClassAd language, resource acquisition through Matchmaking, and on-the-fly deployment and configuration of software modules. The

Teacher's notes

- Starting HTCondor on my laptop:

```
sudo systemctl start condor
[sudo] password for telzur:
condor_status
$ condor_status
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.260  980  0+00:00:04
slot2@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:24
slot3@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:25
slot4@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:26
slot5@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:27
slot6@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:28
slot7@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:29
slot8@LIFEBOOK     LINUX      X86_64 Unclaimed Idle      0.000  980  0+00:00:22
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        X86_64/LINUX     8     0       0         8       0          0        0

               Total     8     0       0         8       0          0        0
```

CPU over usage on hobbits

Inbox x BGU/PP2016B x

Michal Gutin <michalg@exchange.bgu.ac.il> 2:12 PM (2 hours ago)
to Guy, אייל

Images are not displayed. Display images below - Always display images from michalg@exchange.bgu.ac.il

גיא, שלום

אתה מריץ job מאד כבד על הhobbits ועל ה matserv
השרתי ESX מתקרבים לגבול היכולת שלהם - בשתיים השימוש ב CPU מתקרב ל 100%

כמה זמן זה אמור לרוץ?

...

Guy Tel-Zur <gtelzur@post.bgu.ac.il> 3:29 PM (1 hour ago)
to Michal, אייל

שלום מיכל

אכן, אני מריץ ג'ובים של קונדור בתור הדגמה לשיעור שיהיה אחרי הצהרים בנושא.
החישוב שאת מתכוונת אליו כבר הסתיים

גיא

...

--

Guy Tel-Zur, Ph.D.
External Lecturer | Computer Engineering

gtelzur@bgu.ac.il

Tel: +972-54-444-3477 | +972-50-624-4876

Meet Frieda.

Frieda is a scientist. She has a big problem.

Frieda's Problem

a Parameter Sweep:

Find **F**(x,y,z) for

20 values of x

10 values of y

3 values of z

20 x 10 x 3 = 600 combinations!

- **F** takes about 6 hours to compute on a typical workstation
  - 600 runs X 6 = 3600 hours
- **F** requires a moderate amount of memory
  - 256 Mbytes
- **F** performs a moderate amount of I/O:
  - (x,y,z) is 5 MBytes
  - **F**(x,y,z) is 50 MBytes

I have 600 simulations to run.

Where can I get help?

Frieda needs a

batch processing system

- a sequential execution of a series of programs
- run without human interaction

Examples of batch processing systems

- PBS (Portable Batch System) and Open PBS
- LSF (Load Sharing Facility)
- Sun Grid Engine
- Condor

Condor's strengths

- Cycle scavenging (חיפוש שאריות) works!
- High throughput computing
- Very configurable, adaptable
- Supports strong security methods
- Interoperates with many types of computing Grids

A very simple installation of Condor will ...

- Watch jobs and notify you of their progress
- Implement your desired ordering of jobs
- Log your job's activities
- Add fault tolerance to your jobs
- Implement your policy on when the jobs can run on your workstation

With Condor running,

Frieda's 600 simulations are

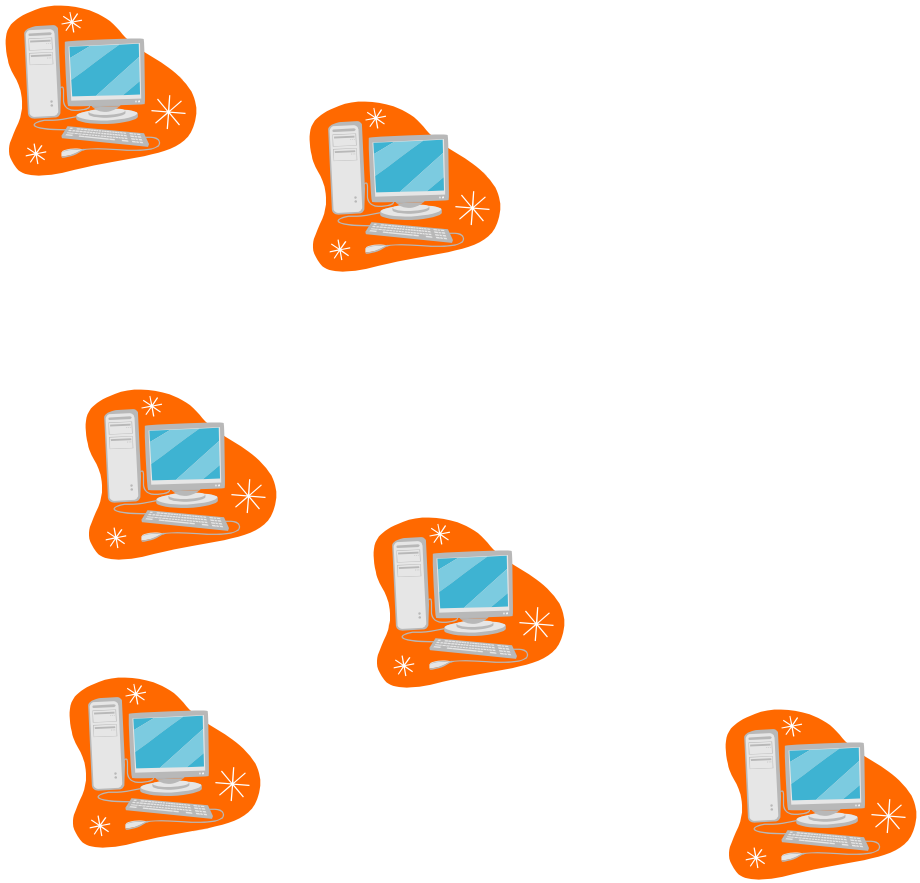
600 Condor jobs,

described by 1 file,

submitted with

1 command

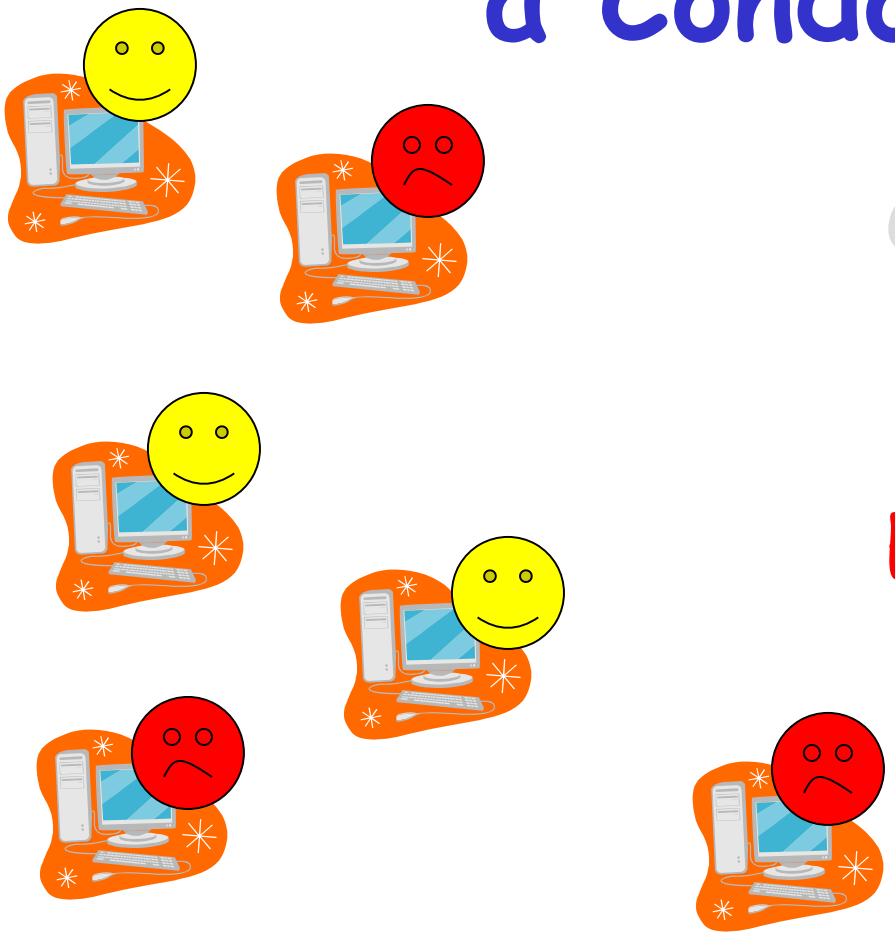
a Condor pool



One or more machines running Condor

Each machine sets its own policy for when to run jobs

a Condor pool



One or more machines running Condor

Each machine sets its own policy for when to run jobs

the Magic of Matchmaking

Jobs and machines state their requirements and preferences

Condor matches jobs with machines based on requirements and preferences

Job

Jobs state their requirements and preferences:

- I need a Linux/x86 platform
- I want the machine with the most memory
- I prefer a machine in the chemistry department

Machine

Machines state their requirements and preferences:

- Run jobs only when there is no keyboard activity
- I prefer to run Frieda's jobs
- I am a machine in the physics department
- Never run jobs belonging to Dr. Smith

Getting Started: Submitting Jobs to Condor

1. Choose a universe for the job
2. Make the job batch-ready
3. Create a submit description file
4. Run condor_submit to put the job in the queue

1. Choose a Universe

- Controls how Condor handles jobs
- some universes:
  - * vanilla
  - * standard
  - * grid
  - * java
  - * mpi
  - * scheduler

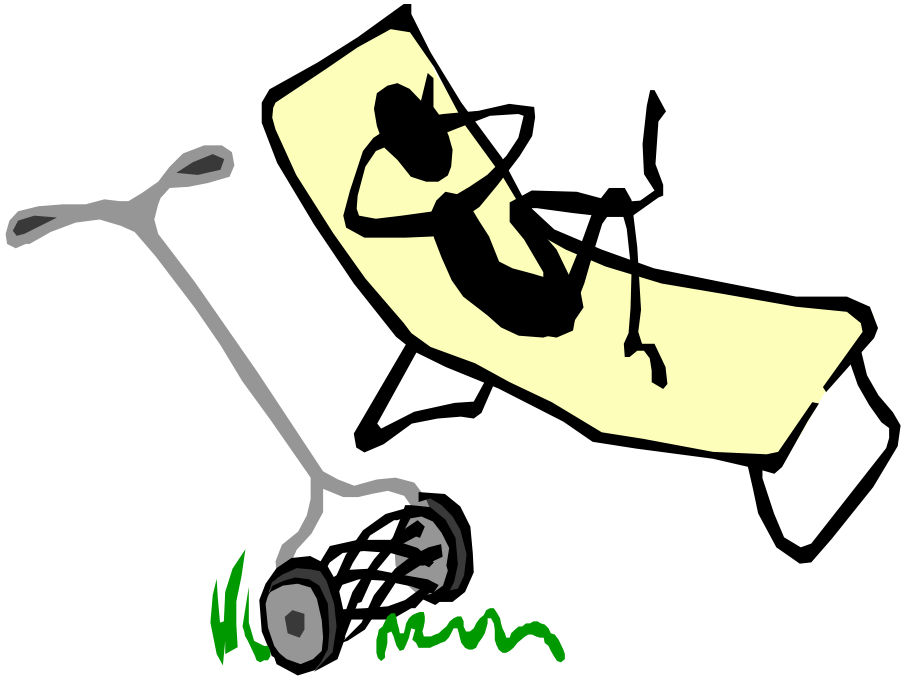
Using the Vanilla Universe

- The vanilla universe:
  - For serial jobs
  - Like vanilla ice cream, can be used for (almost) all jobs

2. Make the job batch-ready

Must run in the background:

- no interactive input
- no windows
- no GUI



2. Make the job batch-ready

- May use `STDIN`, `STDOUT`, and `STDERR` (the keyboard and the screen), but these are files (not devices)
- Similar to Unix

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

3. Create a Submit Description File

- A plain ASCII text file
- File name extensions are irrelevant
- Tells Condor about the job
- Can describe many jobs at once, each with different input, arguments, output, etc.

About the Job

Items that may appear in the submit description file describe:

- input and output files
- command-line arguments
- environment variables
- requirements
- preferences (called rank)

Simple Submit Description File

```
# file name is sim.submit
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but file names are!

universe   = vanilla
executable = sim.exe
output     = output.txt
queue
```

4. Run condor_submit

- Give condor_submit the name of the submit description file:

```
condor_submit sim.submit
```

- condor_submit then
  - checks the submit description file for errors
  - creates a ClassAd that describes the job or jobs
  - places the job or jobs in the queue

ClassAds

- Condor's internal data representation

  Similar to classified advertisements
- Each ClassAd may have many attributes
- Represents an object and its attributes

ClassAds

- ClassAds state facts
  - The job's executable is `analysis.exe`
  - The machine's load average is 5.6
- ClassAds state requirements
  - I require a machine with Linux
- ClassAds state preferences
  - This machine prefers to run jobs from the physics group

ClassAds

ClassAds are:

- semi-structured
- user-extensible
- schema-free
- format is:

  Attribute = Expression

Example:

```
MyType        = "Job"          <-- String
TargetType    = "Machine"
ClusterId     = 1377           <-- Number
Owner         = "roy"
Cmd           = "sim.exe"
Requirements =                 <-- Boolean
    (Arch == "INTEL")
&& (OpSys == "LINUX")
&& (Disk >= DiskUsage)
&& ((Memory * 1024)>=ImageSize)
```

…

The Dog

ClassAd

```
Type = "Dog"
Color = "Brown"
Price = 12
```

ClassAd for the "Job"

```
...
 Requirements =
   (type == "Dog") &&
   (color == "Brown") &&
   (price <= 15)
...
```

The Job Queue

- condor_submit sends the job's ClassAd(s) to the queue
- Jobs enter the queue
  - with an atomic operation: a two-phase commit
  - well defined; no partial failure!
- View the queue with condor_q

Example
condor_submit and condor_q

```
% condor_submit sim.submit
Submitting job(s).
1 job(s) submitted to cluster 1.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER          SUBMITTED     RUN_TIME ST PRI SIZE CMD

  1.0    frieda         6/16 06:52    0+00:00:00 I  0    0.0 sim.exe

1 jobs; 1 idle, 0 running, 0 held

%
```

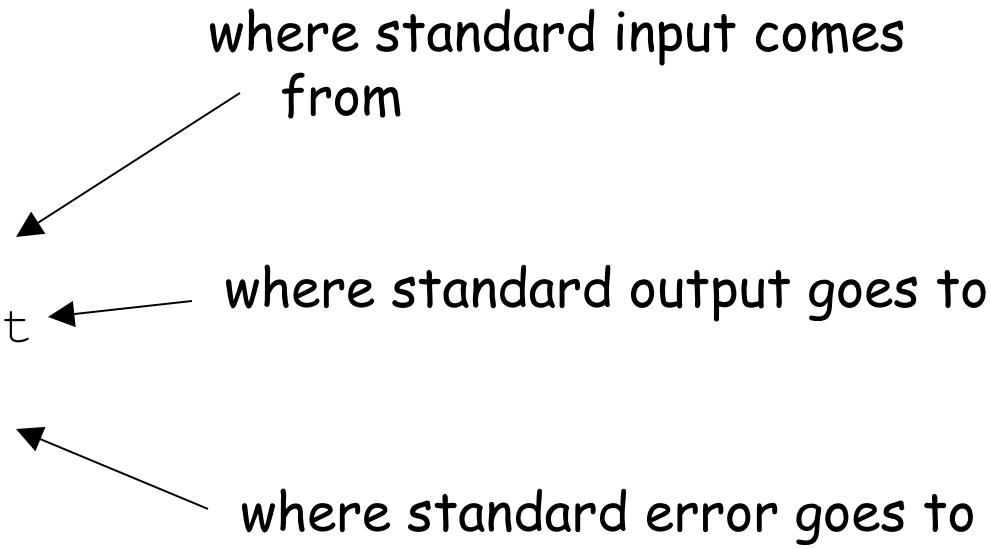
Inspect the full ClassAd

```
% condor_q -l

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
MyType = "Job"
TargetType = "Machine"
ClusterId = 1
QDate = 1150921369
CompletionDate = 0
Owner = "frieda"
RemoteWallClockTime = 0.000000
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
RemoteUserCpu = 0.000000
RemoteSysCpu = 0.000000
ExitStatus = 0
...
```

Input, Output, and Error Files

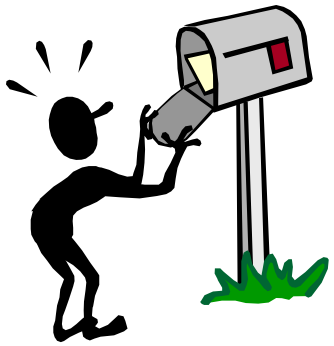
```
universe   = vanilla
executable = sim.exe
input      = input.txt
output     = output.txt
error      = error.txt
log        = sim.log
queue
```



Feedback about jobs

- Condor sends e-mail about events to the person that submits the job
- An entry in the submit description file specifies when

```
Notification = Never
             = Error
             = Always
             = Complete
```



Feedback about jobs

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

```
log = sim.log
```

- Becomes the Life Story of a Job

Sample Condor User Log

```
000 (0001.000.000) 05/25 19:10:03 Job submitted from host:
<128.105.146.14:1816>

...

001 (0001.000.000) 05/25 19:12:17 Job executing on host:
<128.105.146.14:1026>

...

005 (0001.000.000) 05/25 19:13:06 Job terminated.

    (1) Normal termination (return value 0)

...
```

Job Numbering

- Cluster (simplified)

  monotonically increasing integer for each new submission to the queue

- Process

  monotonically increasing integer, for each individual job within a cluster; starts at 0

Job Numbering

3.0

cluster.process

2.6

cluster.process

Each cluster.process is called a job ID

Another Submit Description File for Frieda

```
# Example for one cluster with 2 processes
Universe   = vanilla
Executable = analyze
Input      = a1.in
Output     = a1.out
Error      = a1.err
Log        = a1.log
Queue
Input      = a2.in
Output     = a2.out
Error      = a2.err
Log        = a2.log
Queue
```

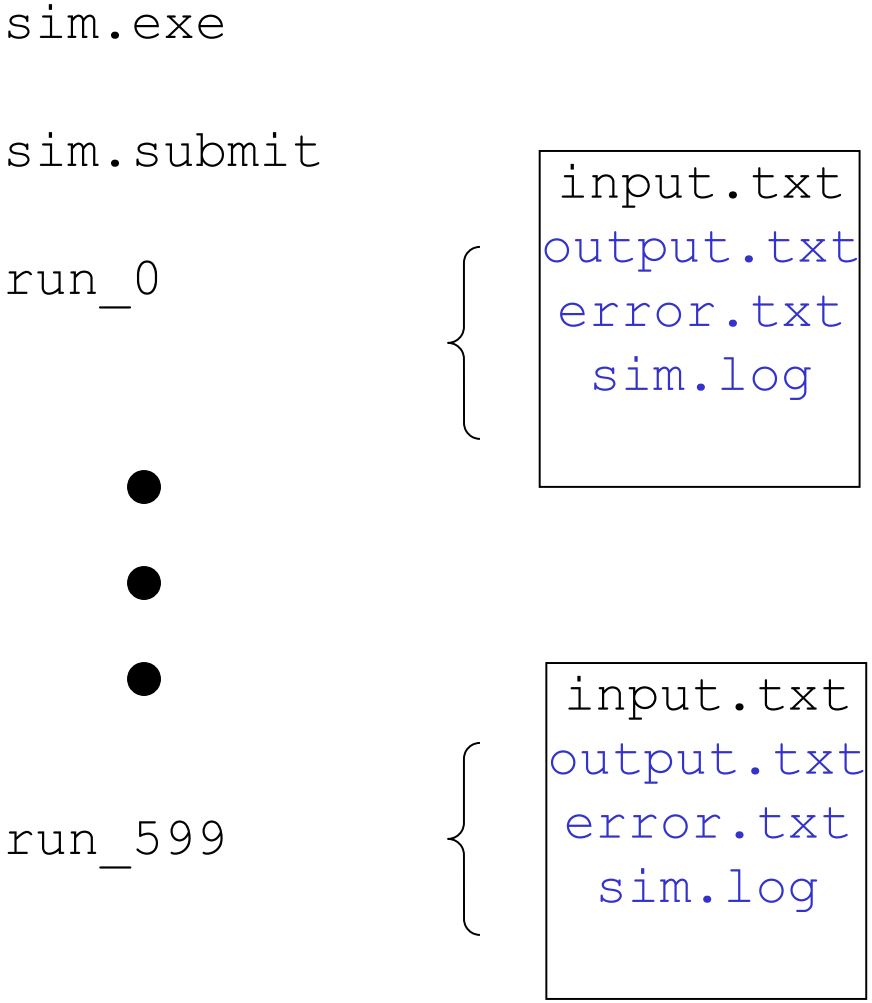
Frieda's Jobs, So Far

```
% condor_submit a.submit
Submitting job(s).
2 job(s) submitted to cluster 2.
% condor_q
-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER    SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda   4/15 06:52  0+00:02:11 R  0   0.0  sim.exe
  2.0    frieda   4/15 06:56  0+00:00:00 I  0   0.0  analyze
  2.1    frieda   4/15 06:56  0+00:00:00 I  0   0.0  analyze
3 jobs; 2 idle, 1 running, 0 held
%
```

Back to Frieda's 600 jobs…

- Place all `input`, `output`, `error` and `log` files in one directory
  - One file of each type for each job
  - 4 files x 600 jobs a 2400 files !!
- Better organization: Create a subdirectory for each job

Frieda's simulation directory



Frieda's 600 Jobs

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_0
Queue
InitialDir = run_1
Queue

    .  .  .

InitialDir = run_599
Queue
```

600 repetitions

Submit Description file is Too Big!

> 1200 lines

Use a Substitution Macro

Syntax:

$(AttributeName)

ClassAd attribute created with value substituted

Frieda needs to use $(Process)

Frieda's 600 Jobs

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_0
Queue
InitialDir = run_1
Queue

    .  .  .
InitialDir = run_599
Queue
```

Frieda's 600 Jobs

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_$(Process)
Queue 600
```

Frieda submits the 600 …

```
% condor_submit sim.submit
Submitting
   job(s) ..................................................
   ..........................................................
   ..........................................................
   ..........................................................
   ..........................................
Logging submit
   event(s) ................................................
   ..........................................................
   ..........................................................
   ..........................................................
   ..........................................
600 job(s) submitted to cluster 3.
```

And, check the queue

```
% 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
3.0    frieda   4/20 12:08 0+00:00:05 R  0   9.8  sim.exe
3.1    frieda   4/20 12:08 0+00:00:03 I  0   9.8  sim.exe
3.2    frieda   4/20 12:08 0+00:00:01 I  0   9.8  sim.exe
3.3    frieda   4/20 12:08 0+00:00:00 I  0   9.8  sim.exe
...
3.598  frieda   4/20 12:08 0+00:00:00 I  0   9.8  sim.exe
3.599  frieda   4/20 12:08 0+00:00:00 I  0   9.8  sim.exe

600 jobs; 599 idle, 1 running, 0 held
```

Command-line Arguments

- For arguments that would be

```
% sim.exe 26 100
```

place in the submit description file:

```
Executable = sim.exe
Arguments = ”26 100”
```

- Or, make use of the substitution macro

```
Arguments = ”$(Process) 100”
```

Removing jobs

- condor_rm removes a job or set of jobs from the queue
- You may only remove your own jobs
  - root on Unix or administrator on Windows may remove any jobs

Removing jobs

- Specify job IDs:

```
condor_rm 4.1
```

(removes job ID cluster 4, process 1)

```
condor_rm 4
```

(removes all cluster 4 jobs)

- Remove all of your own jobs with the -a option

```
condor_rm -a
```

Installation

- Start small

Frieda starts with Personal Condor

- Condor runs on Frieda's workstation
- No root / administrator access required
- No system administrator needed
- After installation, Frieda submits her jobs to her Personal Condor…

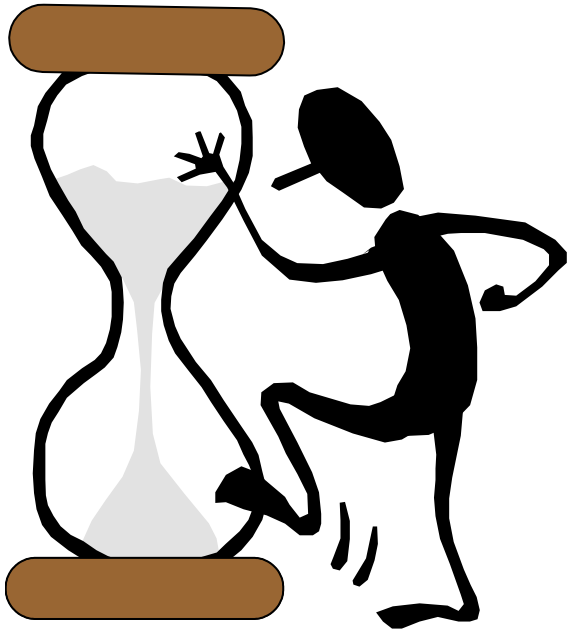
Getting Condor

- Available as a free download from

http://research.cs.wisc.edu/htcondor/

- Download Condor for your platform (operating system and architecture)
  - Available for most Unix (including Linux and Apple's OS/X) platforms
  - Available for Windows NT / XP/7…

Personal Condor. Nice, but not enough!



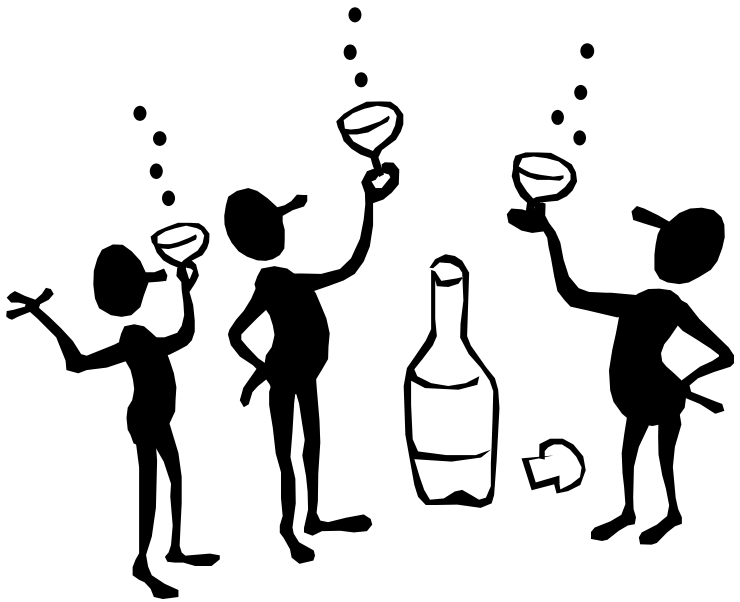
Frieda's 600,

6-hour jobs take

150 days !

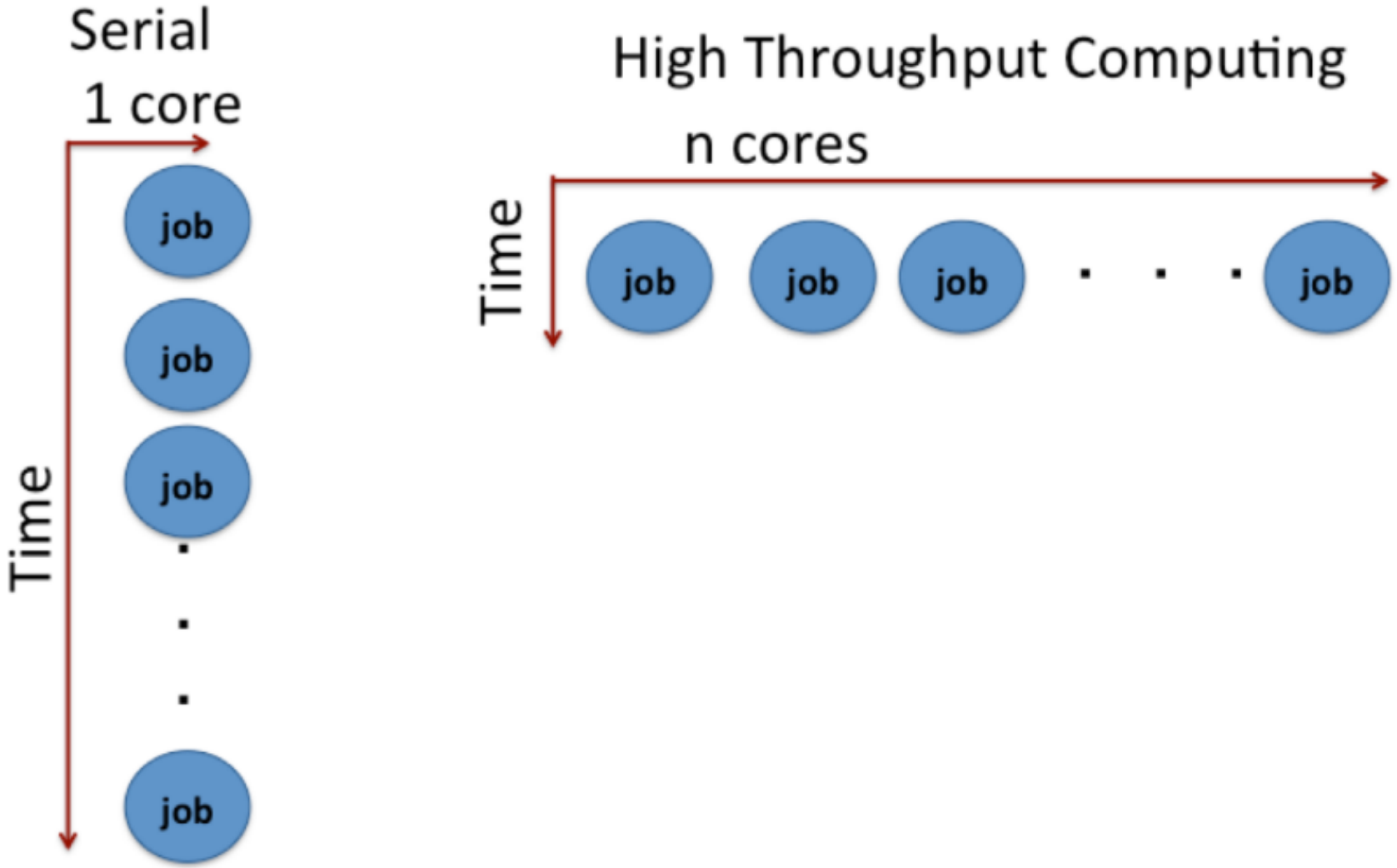
(If jobs run 24 hours per day.)

Good News



Frieda has friends (with machines):
Fred,
Ford,
Fiona, and
Francine.

Efficient approach to handle independent jobs



Frieda's friends

Fred: only run jobs at night

Fiona: never run Ford's jobs

Francine

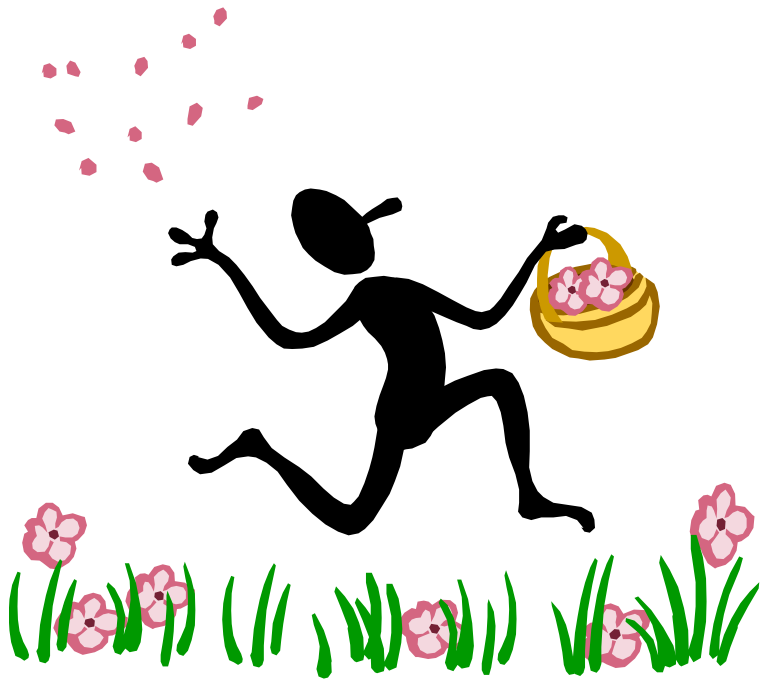
Ford: never run Fiona's jobs

Frieda's Condor pool

- Install Condor on each friend's machine to form a pool
- Configure each machine to implement individual policies for circumstances under which the machine may run jobs

Frieda's jobs finish faster!

- Frieda's friends also submit their jobs to Frieda's pool
- Everyone is happier



condor_status

gives information about the pool:

```
% condor_status
Name           OpSys   Arch    State      Activ LoadAv Mem   ActvtyTime
perdita.cs.wi LINUX   INTEL   Unclaimed  Idle  0.020   511   0+02:28:42
coral.cs.wisc LINUX   INTEL   Claimed    Busy  0.990   511   0+01:27:21
doc.cs.wisc.e LINUX   INTEL   Unclaimed  Idle  0.260   511   0+00:20:04
dsonokwa.cs.w LINUX   INTEL   Owner      Idle  0.810   511   0+00:01:45
ferdinand.cs. LINUX   INTEL   Claimed    Suspe 1.130   511   0+00:00:55
```

Frieda's new question: How can my jobs access their data files?

Condor is flexible

- Shared file system: standard input, output, and error files are assumed accessible (Unix default)
- No shared file system: tell Condor what and when to transfer (Windows default)

Access to Data in Condor

- These commmands do not apply to standard universe jobs
- Use shared file system as available
- No shared file system?

  Condor can transfer files

  - Automatically sends back all changed files
  - Atomic transfer of multiple files
  - Can be encrypted

Condor File Transfer

In the submit description file:

```
should_transfer_files = YES
                        NO
                        IF_NEEDED


when_to_transfer_output = ON_EXIT
                          ON_EXIT_OR_EVICT

transfer_input_files = filename1, filename2 . . .
```

Evict = לפנות

Frieda is happy; Fred is not happy

- Fred's jobs run for a long time
- Before a job finishes, the machine becomes unavailable
- The job goes back into the queue, and starts over again from the beginning

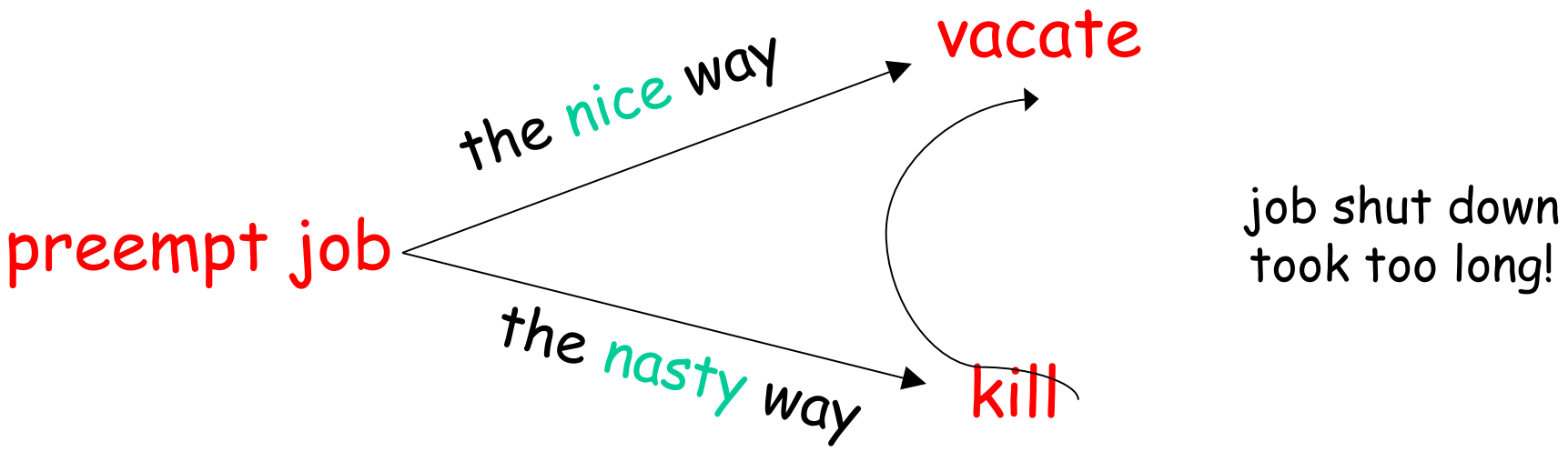
Why Fred's jobs do not finish

Preemption: Condor's decision to stop a currently running job

Why?

1. The machine's policy together with the machine's state lead to this decision
2. Another job or another user's job is prioritized higher, and should be run instead of Fred's job

What / How ?


vacate
the nice way
preempt job
the nasty way
kill
job shut down
took too long!

Condor's standard universe rescues Fred!

- Support for transparent process checkpoint and restart

Remote System Calls in the Standard Universe

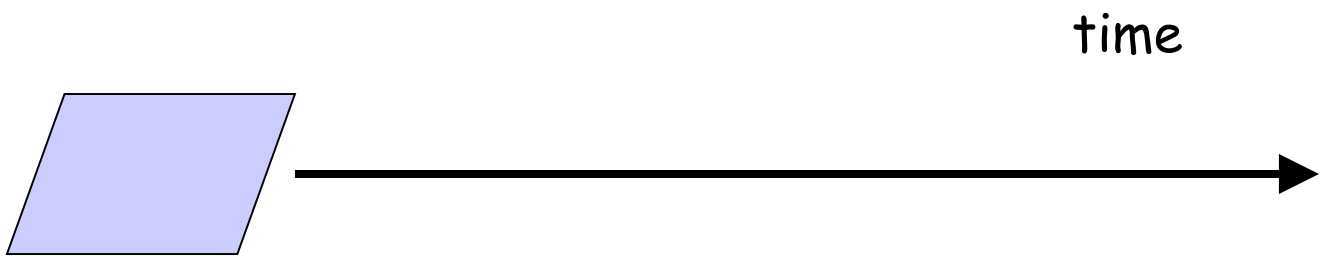
- I/O system calls are trapped and sent back to the submit machine

  Examples: open a file, write to a file
- No source code changes typically required
- Programming language independent

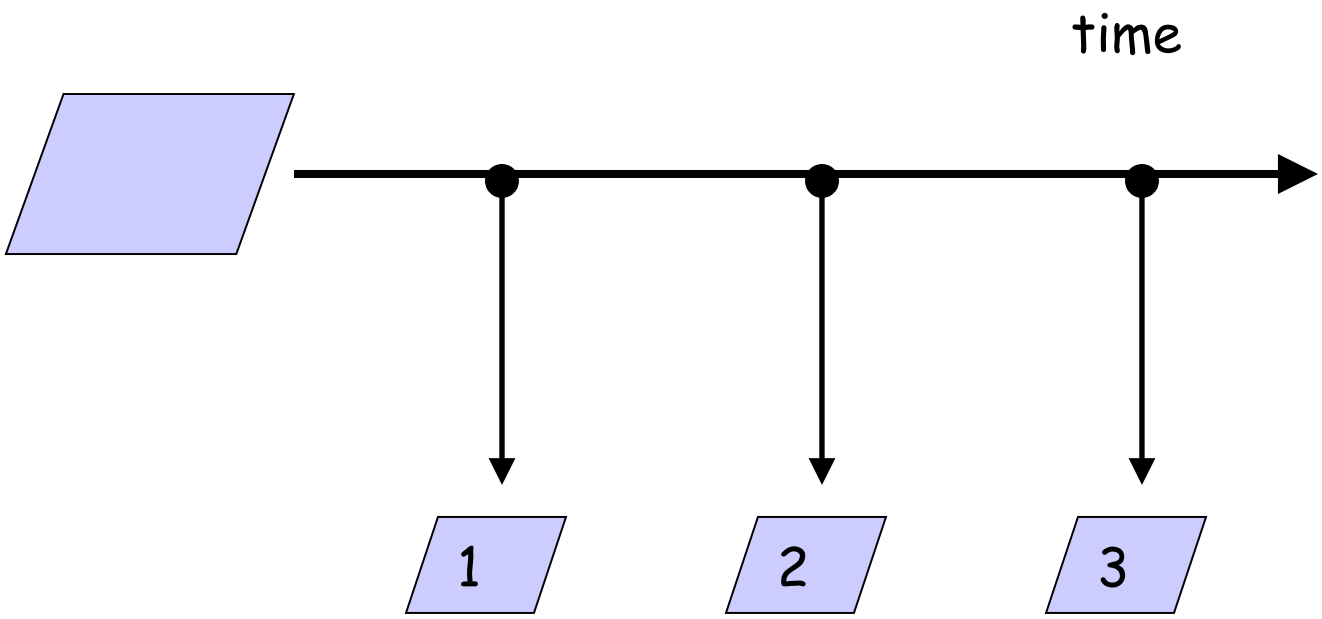
Checkpoints

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

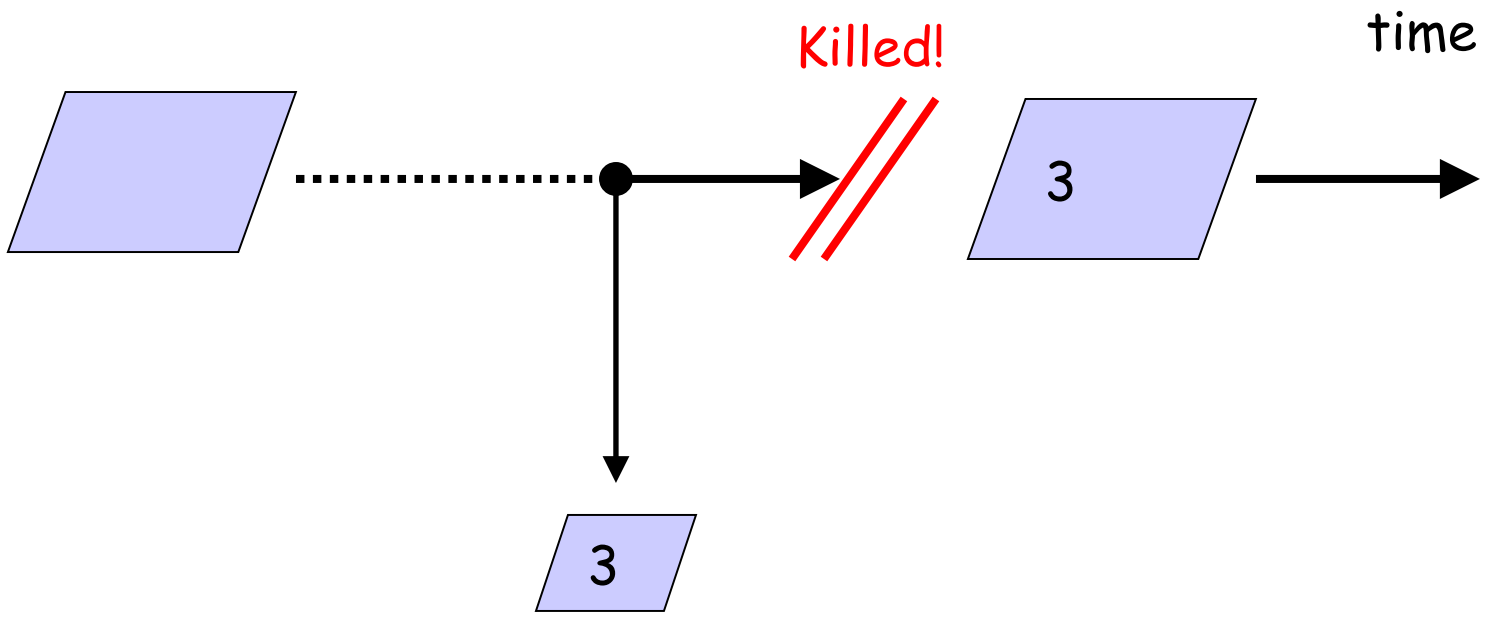
- CPU registers, memory image, I/O



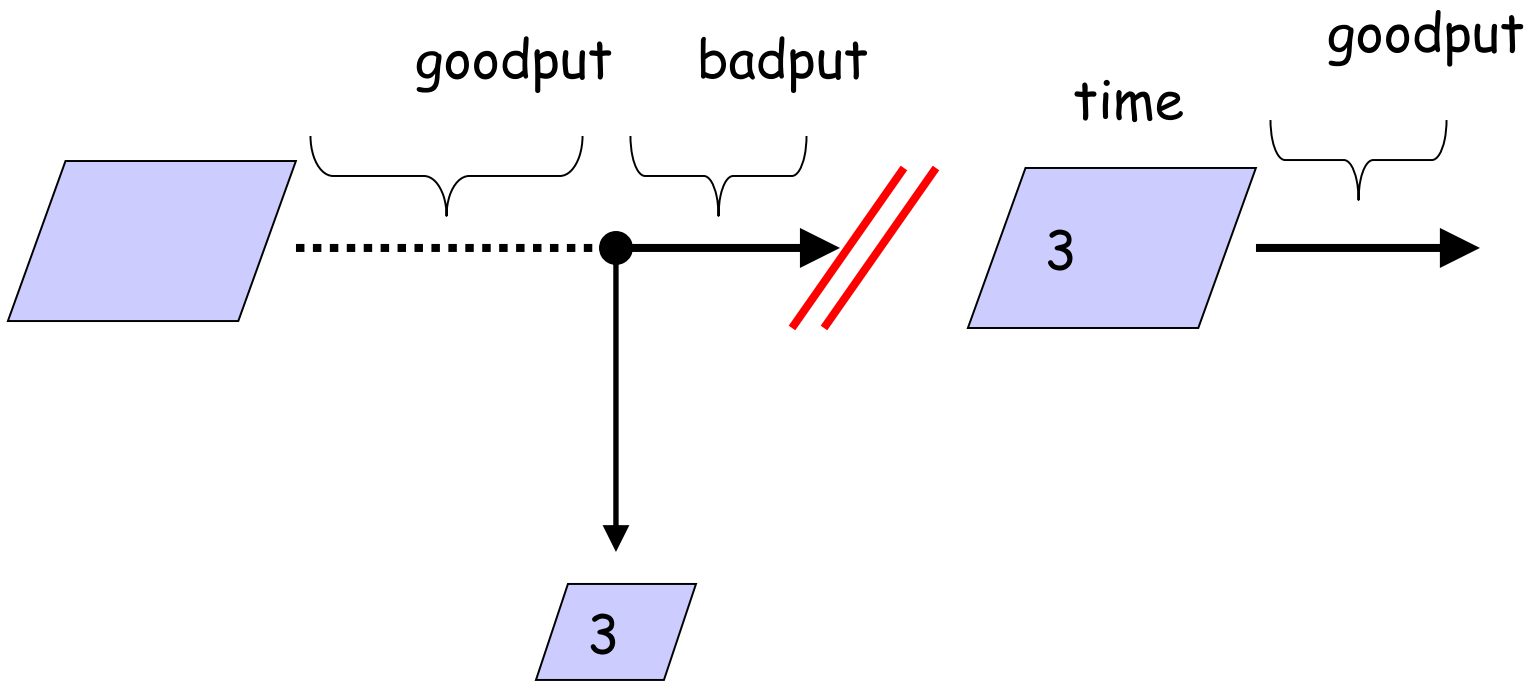
Checkpoints



Checkpoints



Checkpoints



When will Condor produce a checkpoint?

- Periodically, if desired, for fault tolerance
- When the job is preempted, either by a higher priority job, or because the execution machine becomes busy
- When the user explicitly runs a condor_checkpoint, condor_vacate, condor_off or condor_restart command

Making It 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 checkpoints are not at the kernel level, so standard universe jobs may not:
  - Fork()
  - Use kernel threads
  - Use some forms of IPC, such as pipes and shared memory
  - Use unsupported compilers (for example, icc)
  - Execute on Windows machines
- Must have access to source code to relink
- Many typical scientific jobs are OK

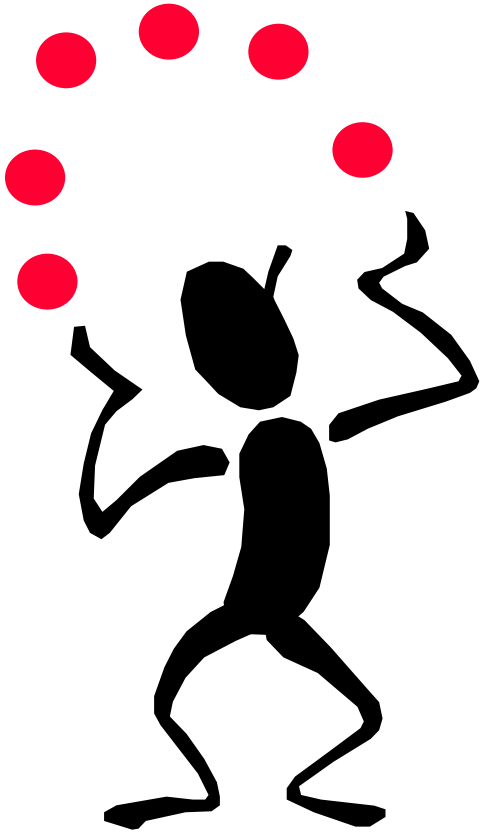
We Always Want More

Condor is managing and running our jobs, but

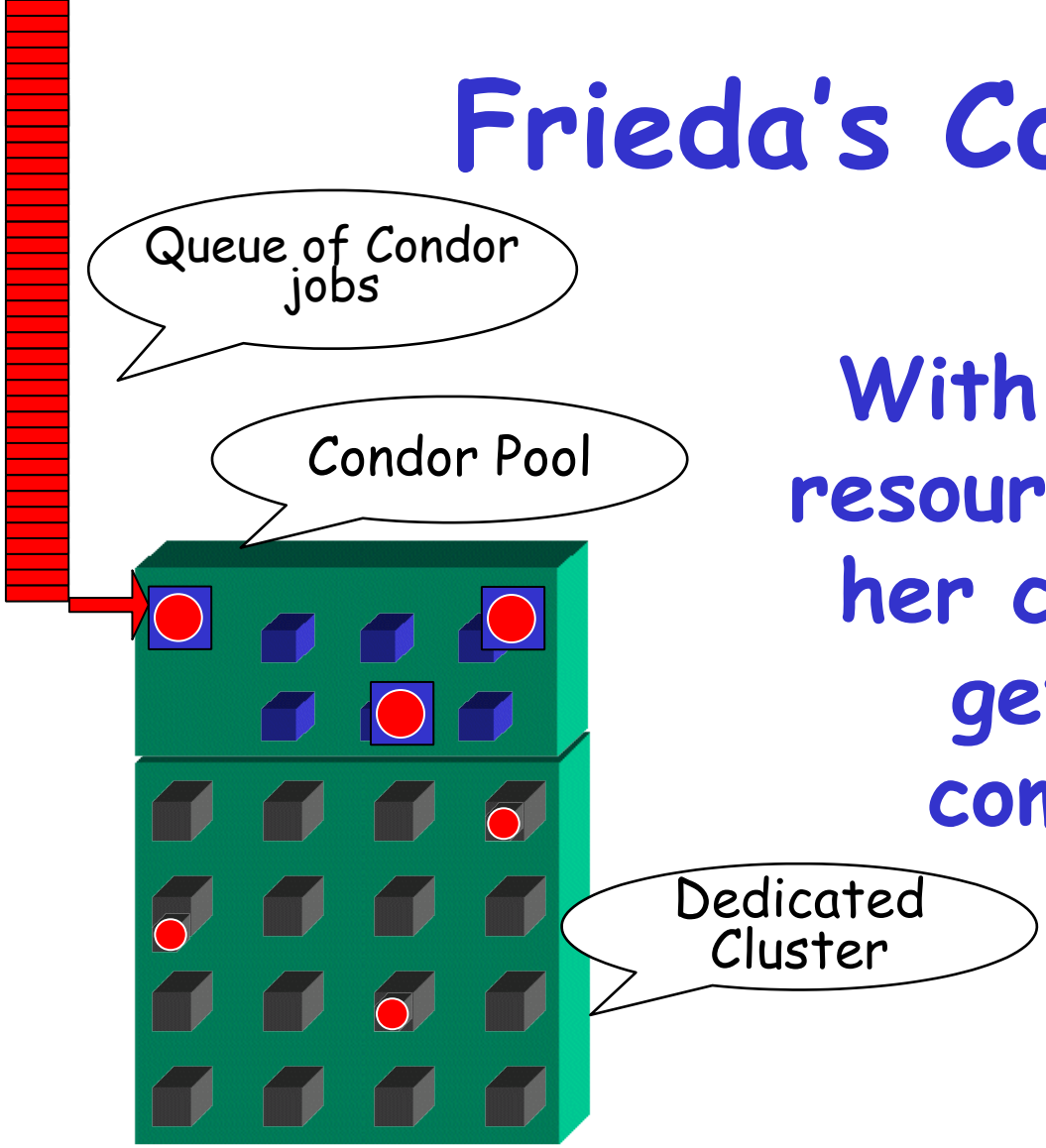
- Our CPU requirements are greater than our resources
- Jobs are preempted more often than we like

Happy Day! Frieda's organization purchased a Dedicated Cluster!

- Frieda installs Condor on all the dedicated Cluster nodes
- Frieda also adds a dedicated central manager
- She configures the entire pool with this new host as the central manager…



Frieda's Condor Pool



With the additional resources, Frieda and her co-workers can get their jobs completed even faster.

New problem for Frieda:

Some of the machines in the pool do not have enough memory or scratch disk space for my job!

Specify Requirements

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

```
Universe    = vanilla
Executable = b
Input       = b.input
Log         = b.log
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Queue 8
```

Or, Specify Rank

- All matches which meet the requirements are sorted by preference with a Rank expression.
- The higher the rank, the better the match

```
Universe    = vanilla
Executable = b
Log         = b.log
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 8
```

Now my jobs are not running…

What's wrong?

Check the queue

```
% 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 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.1 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.2 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.3 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.4 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.5 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.6 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.7 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
6.0 frieda  4/20 13:22  0+00:00:00 H 0   9.8  my_job
8 jobs; 8 idle, 0 running, 1 held
```

H=Held
I=Idle

Look at jobs on hold

```
% condor_q -hold
-- Submiter: x.cs.wisc.edu :
   <128.105.121.53:510> :x.cs.wisc.edu
 ID      OWNER            HELD_SINCE HOLD_REASON
 6.0   frieda            4/20 13:23 Error from starter
   on vm1@skywalker.cs.wisc

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

Or, See full details for a job

```
% condor_q -l 6.0
```

Check machine status

Verify that there are idle machines with condor_status:

```
% condor_status
Name          OpSys   Arch   State    Activity LoadAv Mem   ActvtyTime
vm1@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:20
vm2@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:19
vm3@tonic.c LINUX   INTEL Claimed Busy      0.040   501  0+00:00:17
vm4@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:05

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

Look in the Job Log

Look in the job log for clues:

```
% cat b.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/frieda/workspace/v67/condor-test/test3/run_0
   /b.input' as standard input: No such file or directory
   (errno 2)
        0   -   Run Bytes Sent By Job
        0   -   Run Bytes Received By Job
...
```

How long does it take? Have patience.

- On a busy pool, it can take a while to match and start your jobs
- Wait at least one negotiation cycle (typically 5 minutes)

Look to condor_q for help:

```
% condor_q -analyze 29
---
029.000:  Run analysis summary.  Of 1243 machines,
   1243 are rejected by your job's requirements
0 are available to run your job

WARNING:  Be advised:
   No resources matched request's constraints
   Check the Requirements expression below:
Requirements = ((Memory > 8192)) && (Arch == "INTEL") &&
   (OpSys == "LINUX") && (Disk >= DiskUsage) &&
   (TARGET.FileSystemDomain == MY.FileSystemDomain)
```

Better, but slower: condor_q -better-analyze

```
% condor_q -better-analyze 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 resources:

```
% condor_status -constraint 'Memory > 8192'
```

(no output means no matches)

```
% condor_status -constraint '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
```

Insert ClassAd attributes

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

```
+Department = biochemistry
```

causes the ClassAd to contain

```
Department = "biochemistry"
```

Frieda's pool may prefer biochemistry jobs

- In the configuration for the machines that are owned by the biochemistry department:

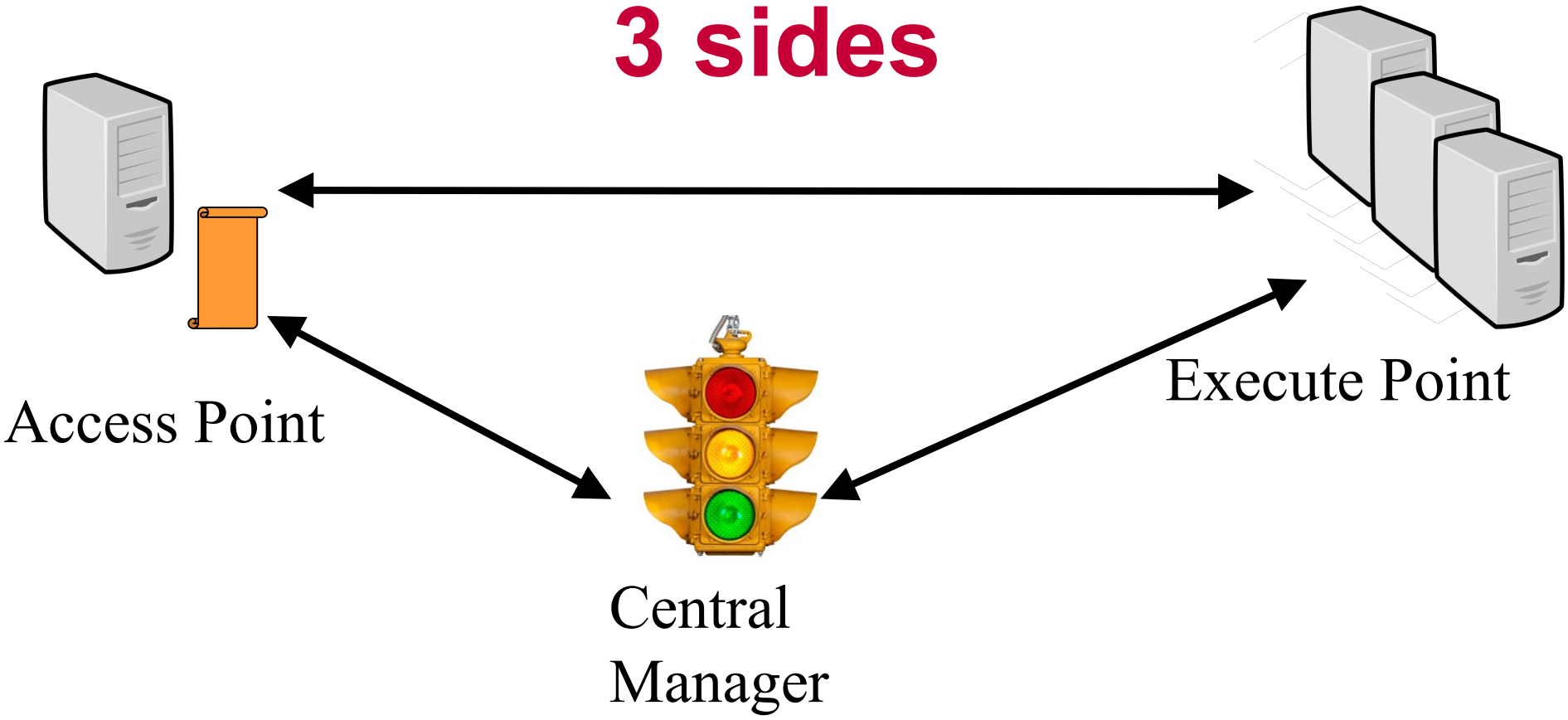
```
RANK = (Department == ”biochemisty”)
```

What Condor Daemons are running on my machine, and what do they do?

Behind the Scenes

- There is a fair amount of software running to make Condor work
- The various pieces of the software are called daemons.
  - Condor daemons communicate with each other
  - Condor daemons are responsible for specific tasks

Overview of condor: 3 sides



Condor daemons

master: Takes care of other processes

collector: Stores ClassAds

negotiator: Performs matchmaking

schedd: Manages job queue

shadow: Manages job (submit side)

startd: Manages computer

starter: Manages job (execution side)

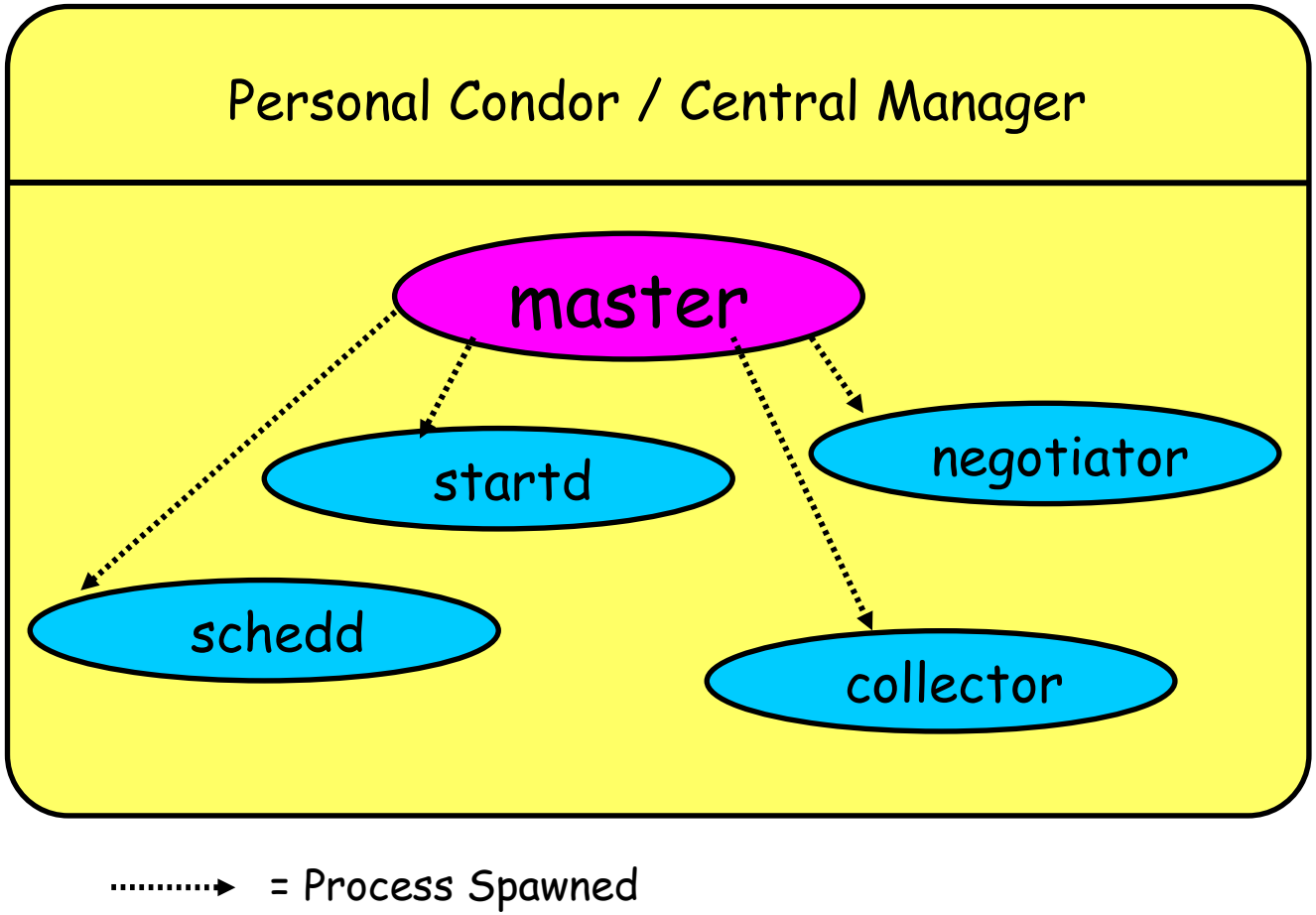
Roles of machines within a Condor pool

- Central manager: central repository and match maker for whole pool
- Execute machine: a machine that may run user jobs
- Submit machine: a machine upon which users may submit jobs

condor_master

- Starts up all other Condor daemons
- If there are any problems and a daemon exits, the condor_master restarts the daemon, and it sends e-mail 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.

Condor Daemon Layout



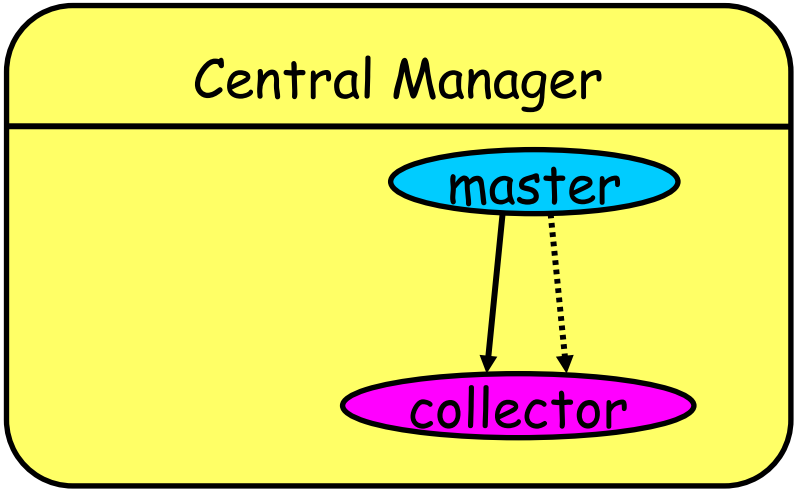
condor_collector

- Located on the **central manager**
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
- Each daemon sends periodic ClassAd to the condor_collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (**condor_status**)
- At least one collector per pool

Condor Pool Layout: condor_collector

= Process Spawned

= ClassAd Communication Pathway



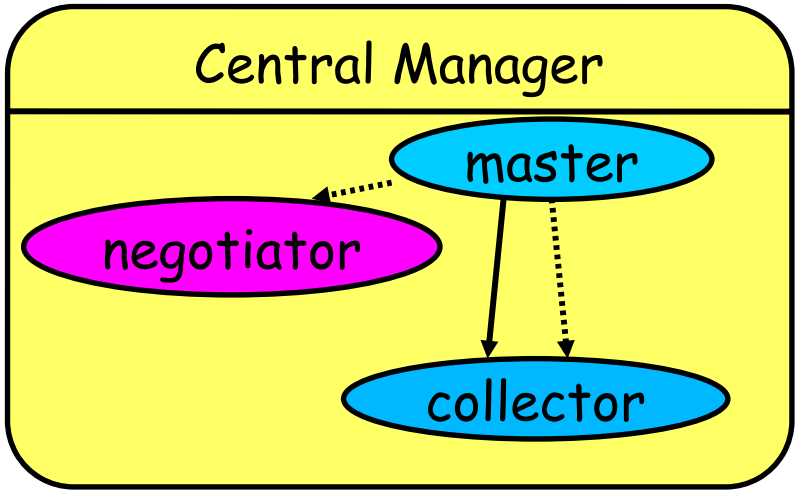
condor_negotiator

- Located on the central manager
- **Performs matchmaking**
- 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
  - Both the job and the machine must satisfy each other's requirements
- Only one condor_negotiator per pool

Condor Pool Layout: condor_negotiator

= Process Spawned

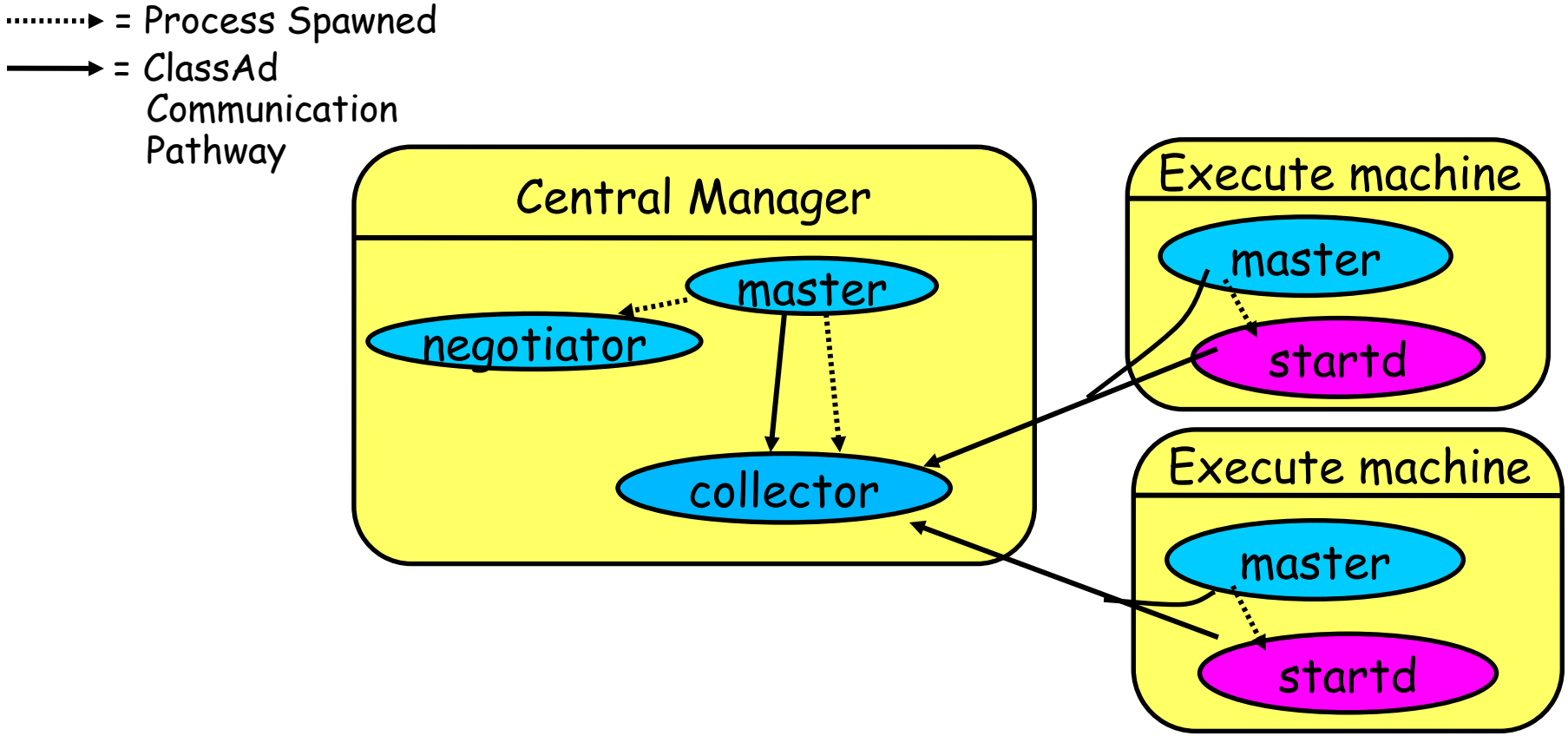
= ClassAd Communication Pathway



condor_startd

- One condor_startd runs on each execute machine
- Represents a machine to Condor
- Responsible for starting, suspending, and stopping user jobs
- Enforces the wishes of the machine's owner (the owner's policy)
- Creates a condor_starter for each running job

Condor Pool Layout: condor_startd



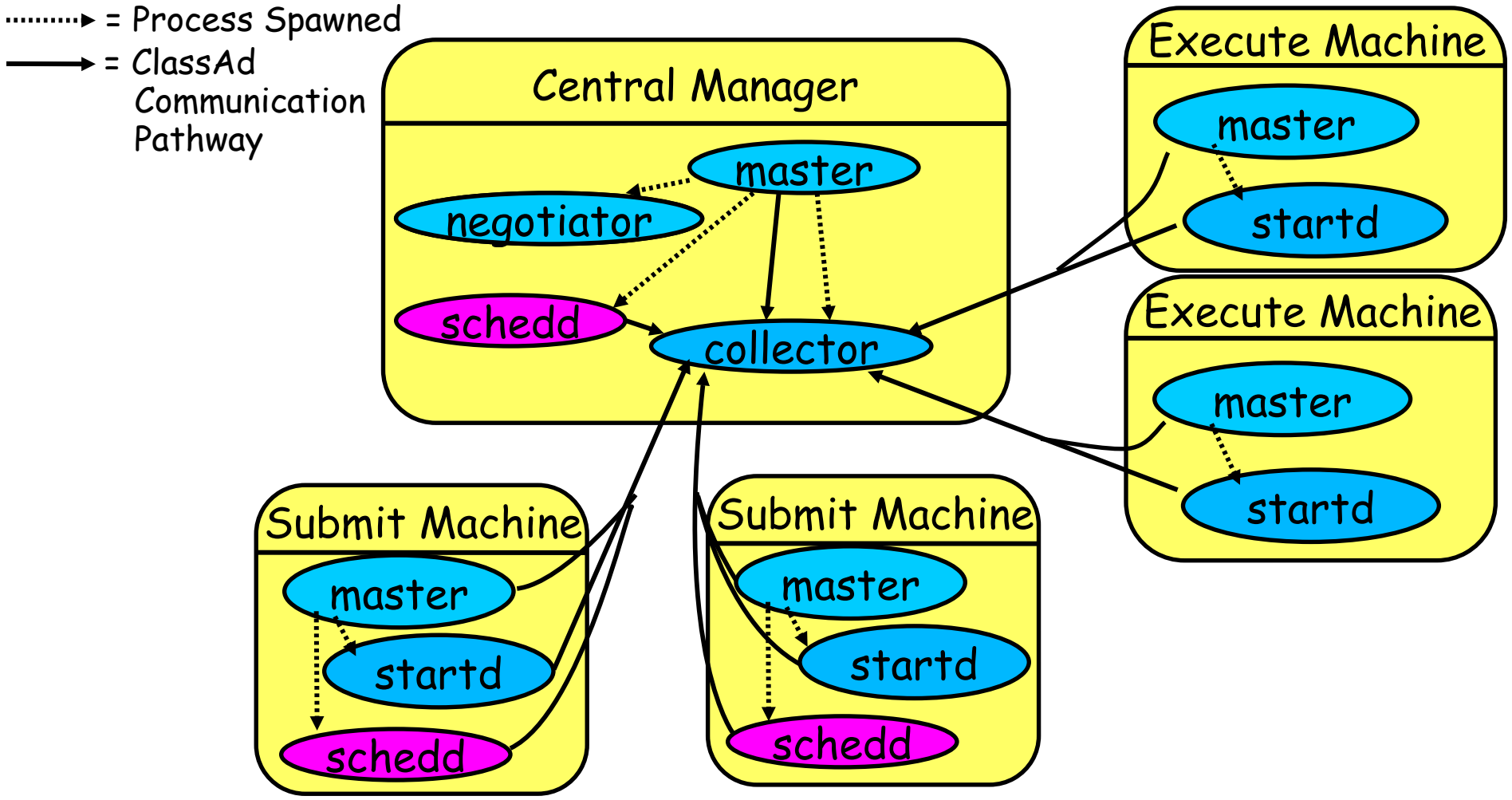
condor_schedd

- One condor_schedd runs on each submit machine
- 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 condor_shadow for each running job

Condor Pool Layout: schedd



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 Priorities
condor_history	Completed Job Info
condor_checkpoint	Force a checkpoint
condor_compile	Link Condor library

Administrator Commands

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

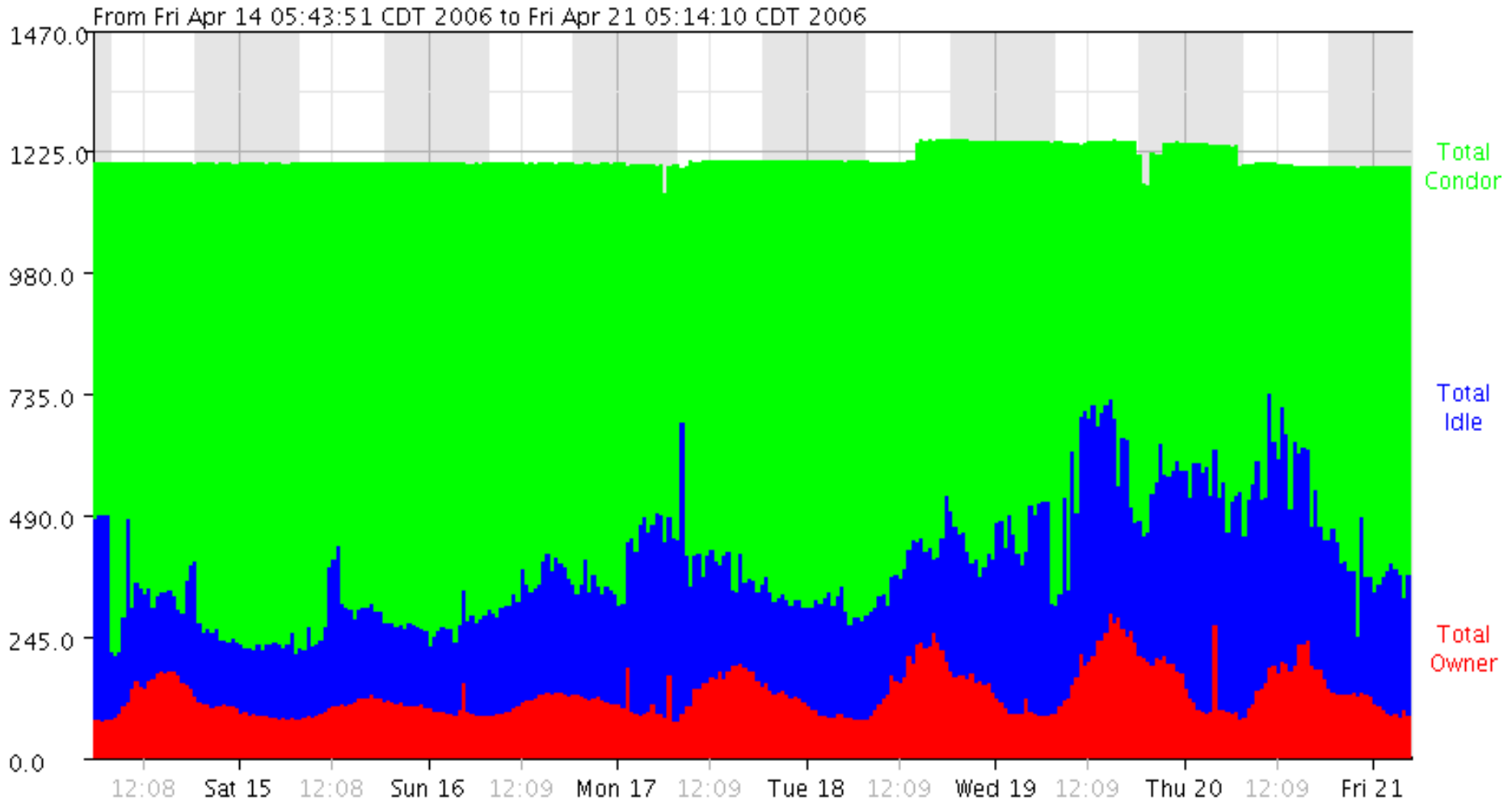
How to watch and record what Condor is doing

Use CondorView!

- 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 cycles
  - Utilization by machine platform, or by user

http://hobbit9.ee.bgu.ac.il/condorview/

CondorView Usage Graph



Condor Version Numbering

- Version numbering scheme similar to that of the (pre 2.6) Linux kernels
- Stable releases, mostly bug fixes
  - Numbering: major.minor.release, minor value is even
  - Current stable release: 6.6.10
- Developer releases, with new features, but may have some bugs
  - Numbering: major.minor.release, minor value is odd
  - Current developer release: 6.7.20

Some Extra Slides from:

Condor and Workflows: An Introduction

Condor Week 2011

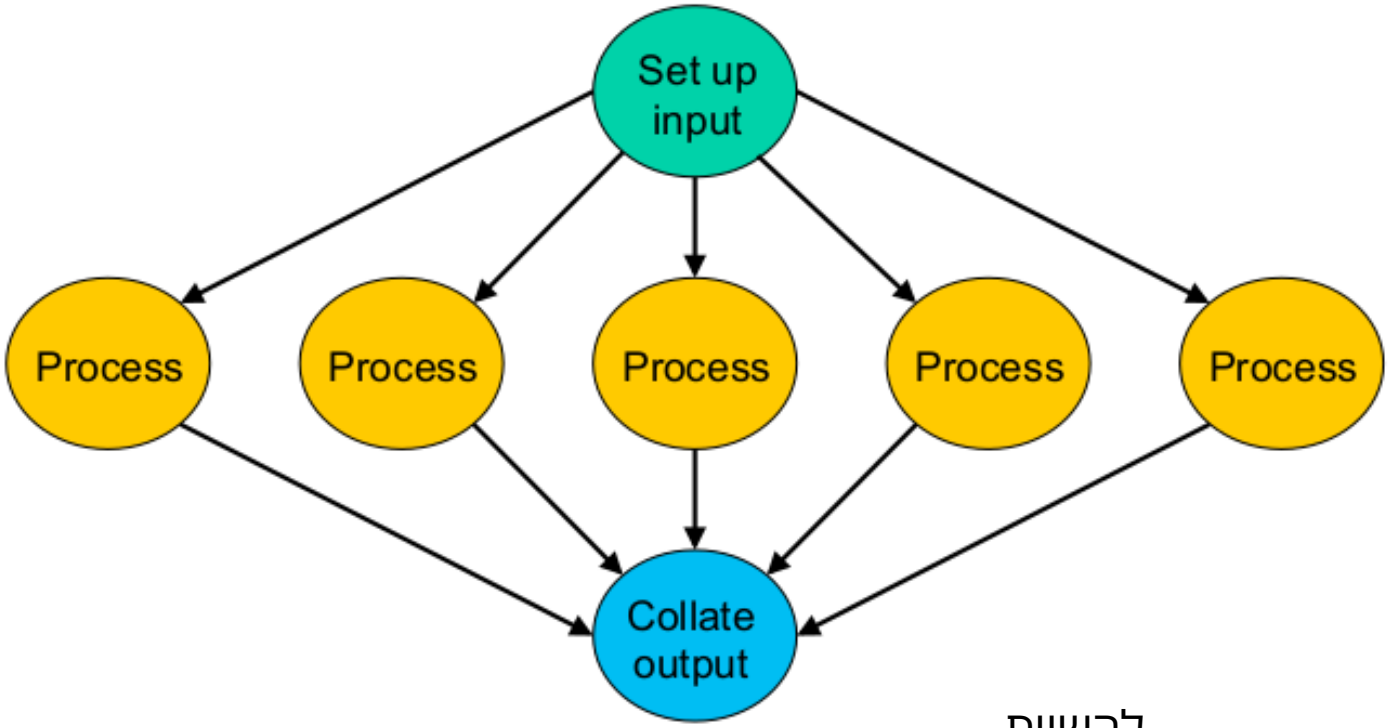
Kent Wenger

Condor Project
Computer Sciences Department
University of Wisconsin-Madison

What are workflows?

- General: a sequence of connected steps
- Our case
  - Steps are Condor jobs
  - Sequence defined at higher level
  - Controlled by a Workflow Management System (WMS), *not just a script*

Workflow example



להשוות

Workflows – launch and forget

- A workflow can take days, weeks or even months
- Automates tasks user *could* perform manually…
  - But WMS takes care of automatically
- Enforces inter-job dependencies
- 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

Workflow tools

- DAGMan: Condor's workflow tool
- Pegasus: a layer on top of DAGMan that is grid-aware and data-aware

- Others…
- This talk will focus mainly on DAGMan

LIGO inspiral search application

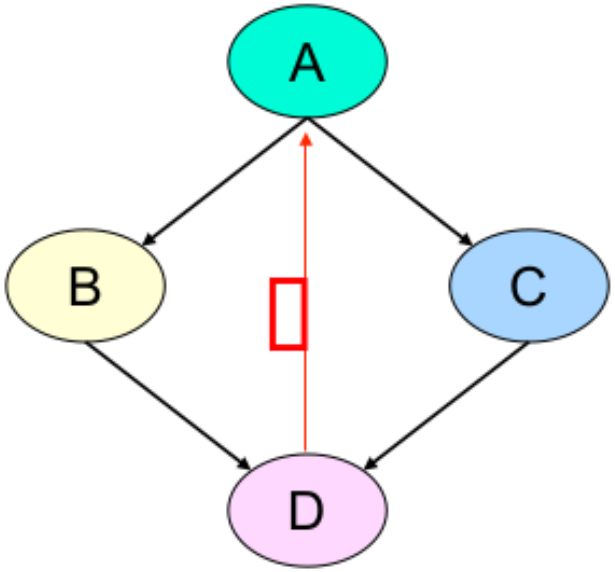
Inspiral workflow application is the work of Duncan Brown, Caltech, Scott Koranda, UW Milwaukee, and the LSC Inspiral group

Albert learns DAGMan

- **Directed Acyclic Graph Manager**
- DAGMan allows Albert to specify the dependencies between his Condor jobs, so DAGMan manages the jobs automatically
- Dependency example: do not run job B until job A has completed successfully

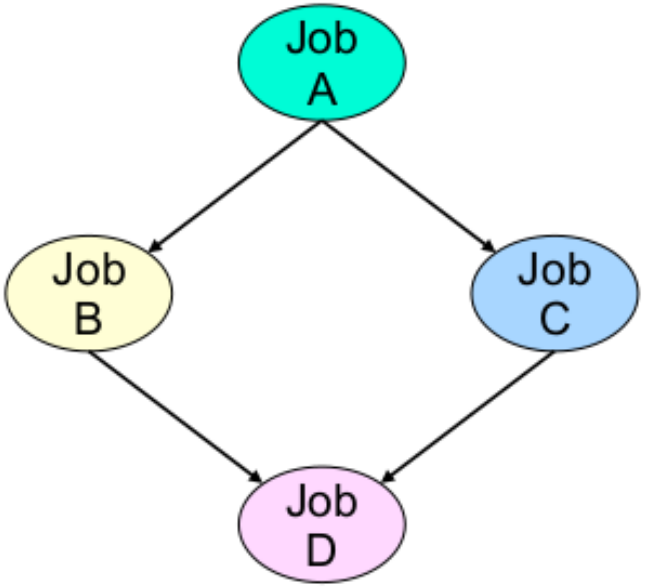
DAG definitions

- DAGs have one or more nodes (or vertices)
- Dependencies are represented by arcs (or edges). These are arrows that go from parent to child)
- No cycles!



Condor and DAGs

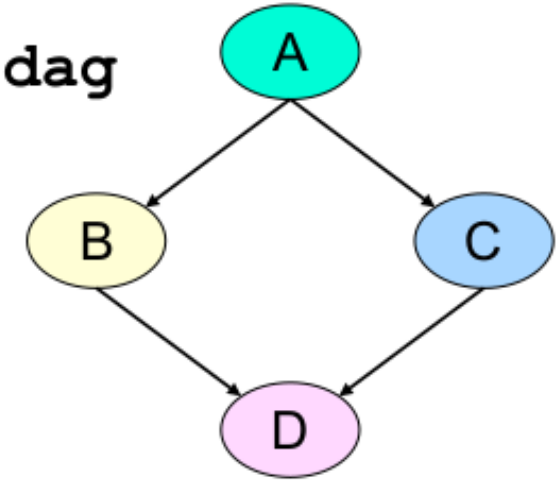
- Each node represents a Condor job (or cluster)
- Dependencies define the possible order of job execution



Defining a DAG to Condor

A DAG input file defines a DAG:

```
# file name: diamond.dag
Job A a.submit
Job B b.submit
Job C c.submit
Job D d.submit
Parent A Child B C
Parent B C Child D
```



Submit description files

For node B:

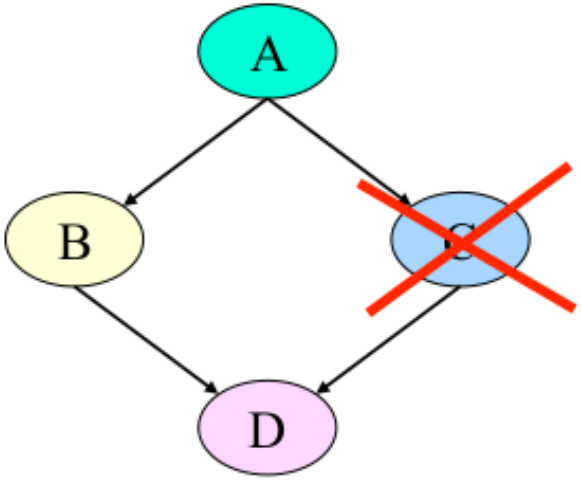
```
# file name:
#    b.submit
universe   = vanilla
executable = B
input      = B.in
output     = B.out
error      = B.err
log        = B.log
queue
```

For node C:

```
# file name:
#    c.submit
universe   = standard
executable = C
input      = C.in
output     = C.out
error      = C.err
log        = C.log
queue
```

Node success or failure

- A node either succeeds or fails
- Based on the return value of the job(s)
  - 0 ⇨ success
  - not 0 ⇨ failure
- This example: C fails
- Failed nodes block execution; DAG fails



Submitting the DAG to Condor

- To submit the entire DAG, run

  `condor_submit_dag DagFile`

- condor_submit_dag creates a submit description file for DAGMan, and DAGMan itself is submitted as a Condor job (in the scheduler universe)
- `-f(orce)` option forces overwriting of existing files

Controlling running DAGs

- **`condor_rm`**
  - Removes all queued node jobs, kills PRE/POST scripts (removes *entire* workflow)
  - Creates rescue DAG
- **`condor_hold`** and **`condor_release`**
  - Node jobs continue when DAG is held
  - No new node jobs submitted
  - DAGMan "catches up" when released

Monitoring a DAG run

- `condor_q -dag`
- `dagman.out` file
- Node `status` file
- `jobstate.log` file
- Dot file

condor_q -dag example

```
% condor_q -dag
-- Submitter: wenger@tonic.cs.wisc.edu : <128.105.121.53:59972> :
   tonic.cs.wisc.edu
 ID      OWNER/NODENAME   SUBMITTED     RUN_TIME ST PRI SIZE CMD
  82.0   wenger          4/15 11:48   0+00:01:02 R  0   19.5 condor_dagman -f
  84.0    |-B1            4/15 11:49   0+00:00:02 R  0   0.0  job_dagman_node
  85.0    |-B2            4/15 11:49   0+00:00:00 I  0   0.0  job_dagman_node
  86.0    |-B3            4/15 11:49   0+00:00:00 I  0   0.0  job_dagman_node
  87.0    |-B4            4/15 11:49   0+00:00:00 I  0   0.0  job_dagman_node
  88.0    |-B5            4/15 11:49   0+00:00:00 I  0   0.0  job_dagman_node
```

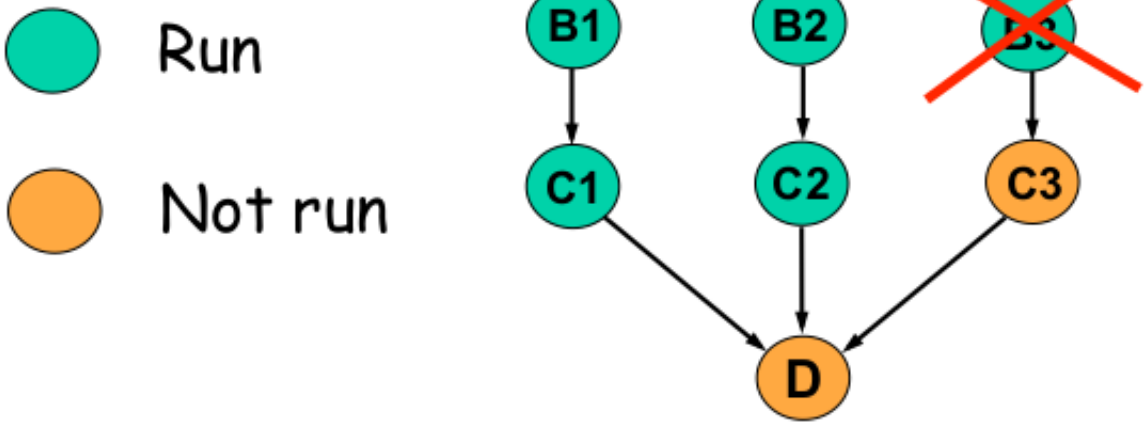
dagman.out contents

```
...
04/17/11 13:11:26 Submitting Condor Node A job(s)...
04/17/11 13:11:26 submitting: condor_submit -a dag_node_name' '=' 'A -a +DAGManJobId' '='
    '180223 -a DAGManJobId' '=' '180223 -a submit_event_notes' '=' 'DAG' 'Node:' 'A -a
    +DAGParentNodeNames' '=' '"" dag_files/A2.submit
04/17/11 13:11:27 From submit: Submitting job(s).
04/17/11 13:11:27 From submit: 1 job(s) submitted to cluster 180224.
04/17/11 13:11:27       assigned Condor ID (180224.0.0)
04/17/11 13:11:27 Just submitted 1 job this cycle...
04/17/11 13:11:27 Currently monitoring 1 Condor log file(s)
04/17/11 13:11:27 Event: ULOG_SUBMIT for Condor Node A (180224.0.0)
04/17/11 13:11:27 Number of idle job procs: 1
04/17/11 13:11:27 Of 4 nodes total:
04/17/11 13:11:27  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
04/17/11 13:11:27   ===     ===      ===     ===     ===        ===      ===
04/17/11 13:11:27     0       0        1       0       0          3        0
04/17/11 13:11:27 0 job proc(s) currently held
...
```

Rescue DAGs

- Save the state of a partially-completed DAG
- Created when a node fails or the **condor_dagman** job is removed with **condor_rm**
  - DAGMan makes as much progress as possible in the face of failed nodes
- Automatically run when you re-run the original DAG (unless -f) (since 7.1.0)

Rescue DAGs, cont.



PRE and POST scripts

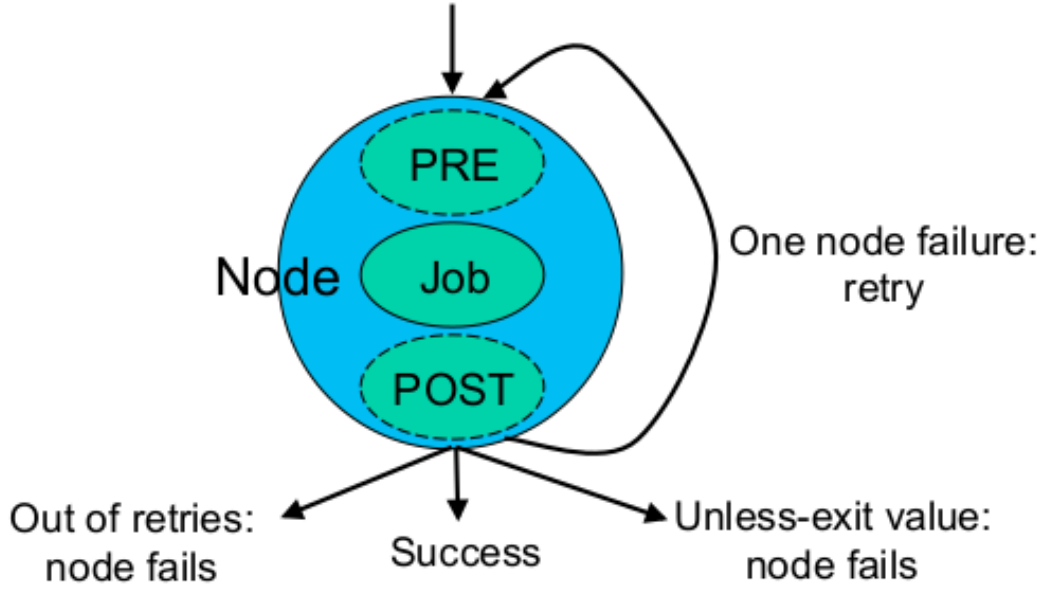
- DAGMan allows PRE and/or POST scripts
  - Not necessarily a script: any executable
  - Run before (PRE) or after (POST) job
  - Run on the submit machine
- In the DAG input file:

```
Job A a.submit
Script PRE A before-script arguments
Script POST A after-script arguments
```

- No spaces in script name or arguments

Node retries, continued

- Node is retried as a whole



Relevant Links

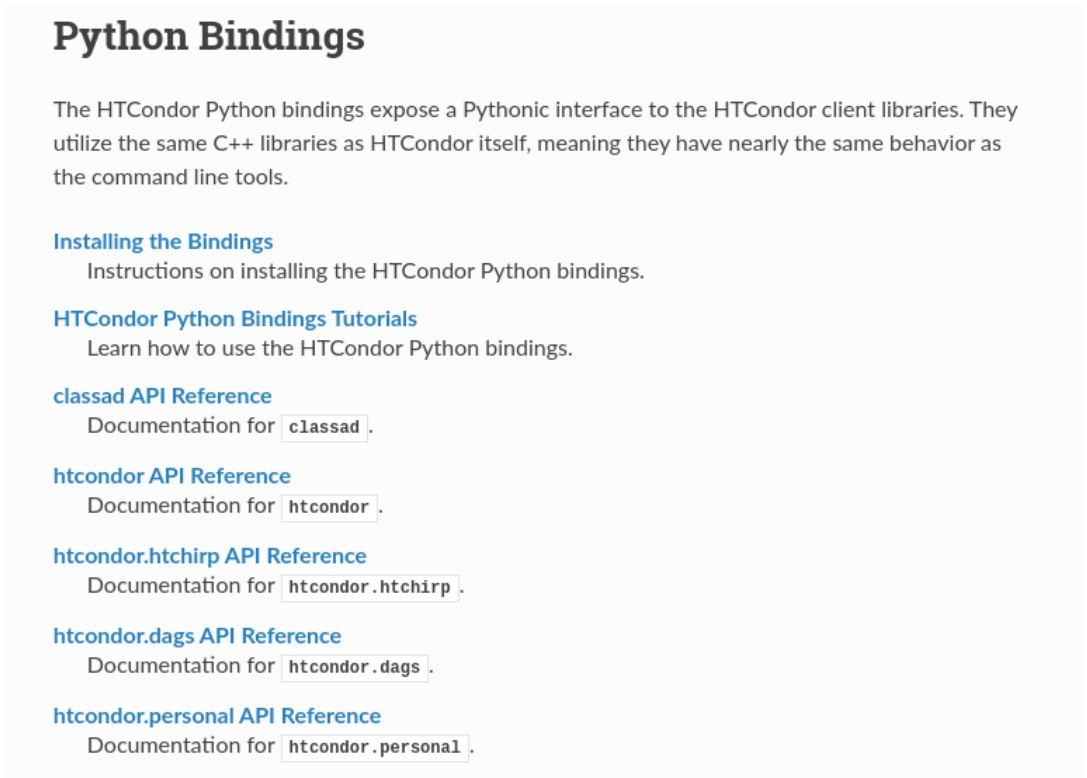
- DAGMan: www.cs.wisc.edu/condor/dagman
- Pegasus: http://pegasus.isi.edu/
- Makeflow: http://nd.edu/~ccl/software/makeflow/
- For more questions: condor-admin@cs.wisc.edu

HTCondor and Python

For further reading:

https://research.cs.wisc.edu/htcondor/HTCondorWeek2014/presentations/TheisenT-Python.pdf

https://htcondor.readthedocs.io/en/v9_0/apis/python-bindings/index.html



Python Bindings

The HTCondor Python bindings expose a Pythonic interface to the HTCondor client libraries. They utilize the same C++ libraries as HTCondor itself, meaning they have nearly the same behavior as the command line tools.

Installing the Bindings
Instructions on installing the HTCondor Python bindings.

HTCondor Python Bindings Tutorials
Learn how to use the HTCondor Python bindings.

classad API Reference
Documentation for `classad`.

htcondor API Reference
Documentation for `htcondor`.

htcondor.htchirp API Reference
Documentation for `htcondor.htchirp`.

htcondor.dags API Reference
Documentation for `htcondor.dags`.

htcondor.personal API Reference
Documentation for `htcondor.personal`.

4 Examples

Example 1:

The application is python

And the program is the argument:

```
universe = vanilla
executable = /usr/bin/python
arguments = mytest.py $(Process)
output = myoutput$(Process).out
error = myerror$(Process).err
log = mylog$(Process).log
queue 10
```

במערכת מחשוב מבוזרת צריך, במקרה כזה, לשנע את התכנית עם file transfer אל צמתי החישוב.

Example 2:

A stand alone python app

insert `#!/bin/python` in the program and then add execution permission:

`chmod u+x ./mytest.py`

If it runs from the command line it runs with HTCondor

```
universe = vanilla
executable =
/home/telzur/science/Teaching/PP/lectures/12/code/HTCondor/Python/
method2/mytest.py
arguments = mytest.py $(Process)
output = myoutput$(Process).out
error = myerror$(Process).err
log = mylog$(Process).log
queue 10
```

Example 3: A Jupyter notebook and Python binding

Please note: This method currently does not work on the Hobbits.

את התמיכה בקונדור ניתן להתקין בהתאם לסוג מנהל ההתקנות conda או pip.

```
telzur@GL553VD ~/science/Teaching/PP/lectures/12/code/HTCondor/Python/method3 $ conda install htcondor

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    boost-1.74.0               |   py38h2b96118_4         354 KB  conda-forge
    boost-cpp-1.74.0           |       hc6e9bd1_3        16.3 MB  conda-forge
    conda-build-3.21.7         |   py38h578d9bd_0         562 KB  conda-forge
    gettext-0.19.8.1           |    h0b5b191_1005         3.6 MB  conda-forge
    glib-2.68.4                |       h9c3ff4c_0         447 KB  conda-forge
    glib-tools-2.68.4          |       h9c3ff4c_0          86 KB  conda-forge
    gsoap-2.8.117              |       h90a1d37_0         1.8 MB  conda-forge
    gst-plugins-base-1.18.5    |       hf529b03_0         2.6 MB  conda-forge
    gstreamer-1.18.5           |       h76c114f_0         2.0 MB  conda-forge
    harfbuzz-2.9.1             |       h83ec7ef_1         2.0 MB  conda-forge
    htcondor-9.4.0             |   py38h578d9bd_0          17 KB  conda-forge
    htcondor-classads-9.4.0    |       hd3c618e_0         9.8 MB  conda-forge
htcondor-9.4.0        | 17 KB     | ################################################################################ | 100%
libglib-2.68.4        | 3.0 MB    | ################################################################################ | 100%
glib-2.68.4           | 447 KB    | ################################################################################ | 100%
voms-2.1.0rc0         | 597 KB    | ################################################################################ | 100%
htcondor-utils-9.4.0  | 21.6 MB   | ################################################################################ | 100%
python-htcondor-9.4.  | 9.4 MB    | ################################################################################ | 100%
pyqt-impl-5.12.3      | 5.9 MB    | ################################################################################ | 100%
pyqtwebengine-5.12.1  | 175 KB    | ################################################################################ | 100%
harfbuzz-2.9.1        | 2.0 MB    | ################################################################################ | 100%
gsoap-2.8.117         | 1.8 MB    | ################################################################################ | 100%
gstreamer-1.18.5      | 2.0 MB    | ################################################################################ | 100%
scitokens-cpp-0.6.3   | 107 KB    | ################################################################################ | 100%
libpq-13.5            | 2.8 MB    | ################################################################################ | 100%
nss-3.73              | 2.1 MB    | ################################################################################ | 100%
pyqtchart-5.12        | 257 KB    | ################################################################################ | 100%
libcondor_utils-9.4.  | 24.9 MB   | ################################################################################ | 100%
conda-build-3.21.7    | 562 KB    | ################################################################################ | 100%
pyqt5-sip-4.19.18     | 311 KB    | ################################################################################ | 100%
```

pip

```
telzur@GL553VD ~/science/tests/condor $ which python
/usr/bin/python
telzur@GL553VD ~/science/tests/condor $ pip install htcondor
Defaulting to user installation because normal site-packages is not writeable
Collecting htcondor
  Downloading htcondor-9.4.0-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_6
4.whl (41.9 MB)
     |████████████████████████████████| 41.9 MB 172 kB/s
Installing collected packages: htcondor
Successfully installed htcondor-9.4.0
telzur@GL553VD ~/science/tests/condor $
```

Show demo under method3:
`http://localhost:8889/notebooks/htcondor_binding_example.ipynb`

jupyter htcondor_binding_example Last Checkpoint: 12 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
# HTCondor Python binding
# Reference: https://htcondor.readthedocs.io/en/latest/apis/python-bindings/tutorials/Submitting-and-Managing-Jobs.htm
# Before running this notebook execute "clean.sh" in the job folder - see below.
# Guy, 1/1/2022
import htcondor
import classad
```

In [2]:
```
hostname_job = htcondor.Submit({
    "executable": "/home/telzur/science/Teaching/PP/lectures/12/code/HTCondor/Python/method3/mytest.py",  # the progra
    "arguments": "$(Process)",
    "output": "myoutput$(Process).out",       # anything the job prints to standard output will end up in this file
    "error": "myerror$(Process).err",        # anything the job prints to standard error will end up in this file
    "log": "mylog$(Process).log",          # this file will contain a record of what happened to the job
    "request_cpus": "1",             # how many CPU cores we want
    "request_memory": "128MB",       # how much memory we want
    "request_disk": "128MB",         # how much disk space we want
})

print(hostname_job)
```

```
executable = /home/telzur/science/Teaching/PP/lectures/12/code/HTCondor/Python/method3/mytest.py
arguments = $(Process)
output = myoutput$(Process).out
error = myerror$(Process).err
log = mylog$(Process).log
request_cpus = 1
request_memory = 128MB
request_disk = 128MB
```

In [3]:
```
schedd = htcondor.Schedd()          # get the Python representation of the scheduler
with schedd.transaction() as txn:   # open a transaction, represented by `txn`
    cluster_id = hostname_job.queue(txn)     # queues one job in the current transaction; returns job's ClusterId

print(cluster_id)
```

```
263
```

In [4]:
```
import os
import time

output_path = "/home/telzur/science/Teaching/PP/lectures/12/code/HTCondor/Python/method3/myoutput0.out"

# this is a crude way to wait for the job to finish
# see the Advanced tutorial "Scalable Job Tracking" for better methods!
while not os.path.exists(output_path):
    print("Output file doesn't exist yet; sleeping for 1 second")
    time.sleep(1)
with open(output_path, mode = "r") as f:
    print(f.read())
```

```
Output file doesn't exist yet; sleeping for 1 second
Hi, I am job number 0
```

Example 4

Reference:
https://swc-osg-workshop.github.io/OSG-UserTraining-PEARC17/novice/DHTC/04a-ScalingUp-python.html

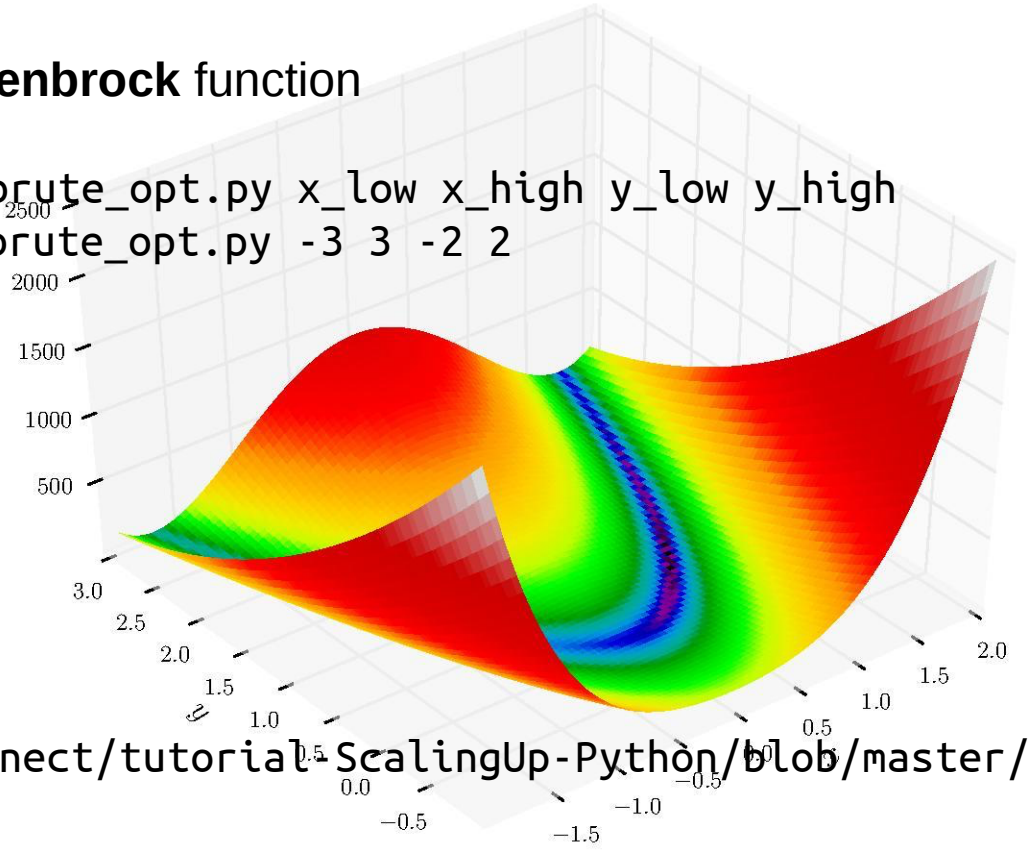
https://support.opensciencegrid.org/support/solutions/articles/12000062019-scaling-up-with-htcondor-s-queue-command

f = (1 - x)**2 + (y - x**2)**2

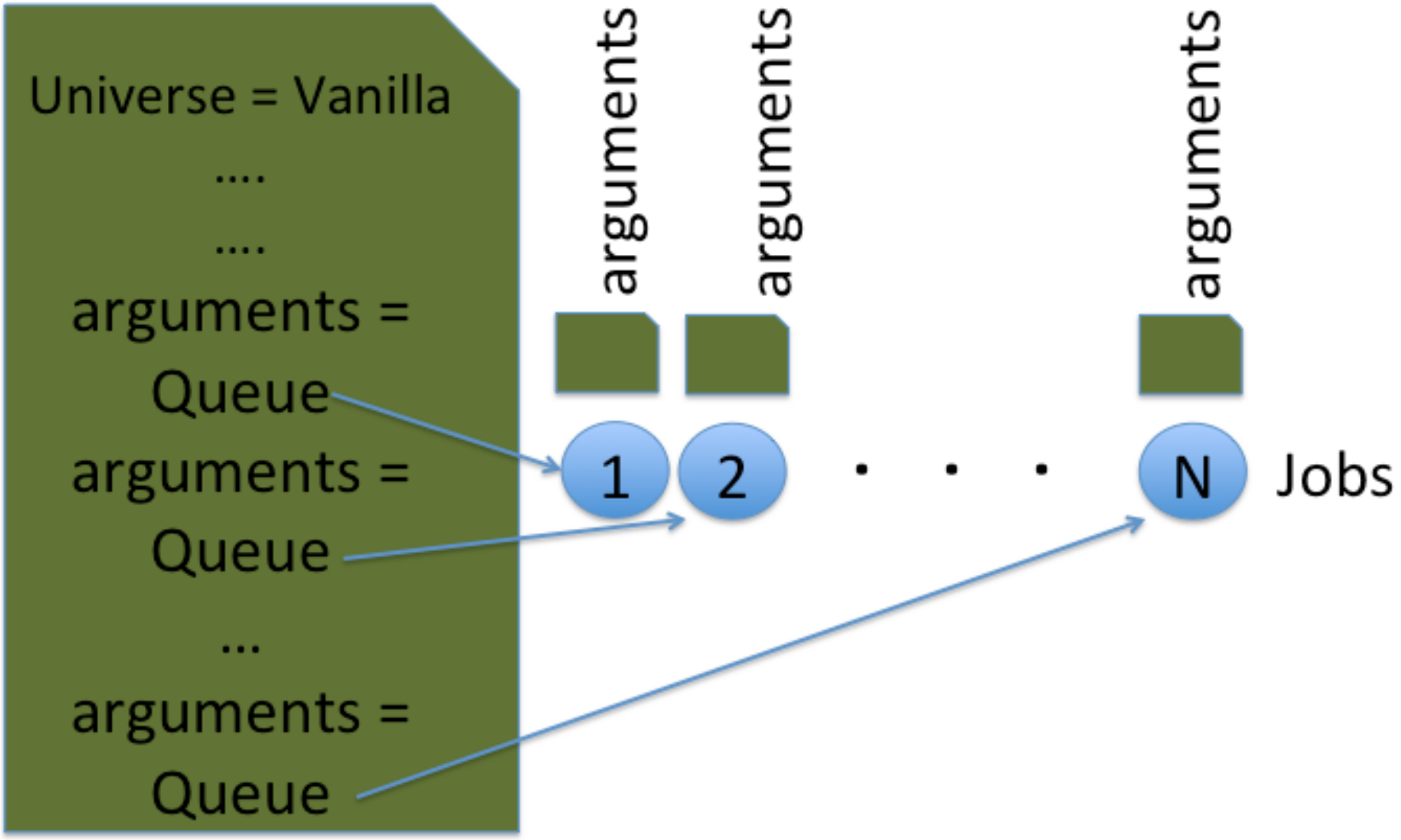
בדומה לדוגמה של המעדנית פרידה...

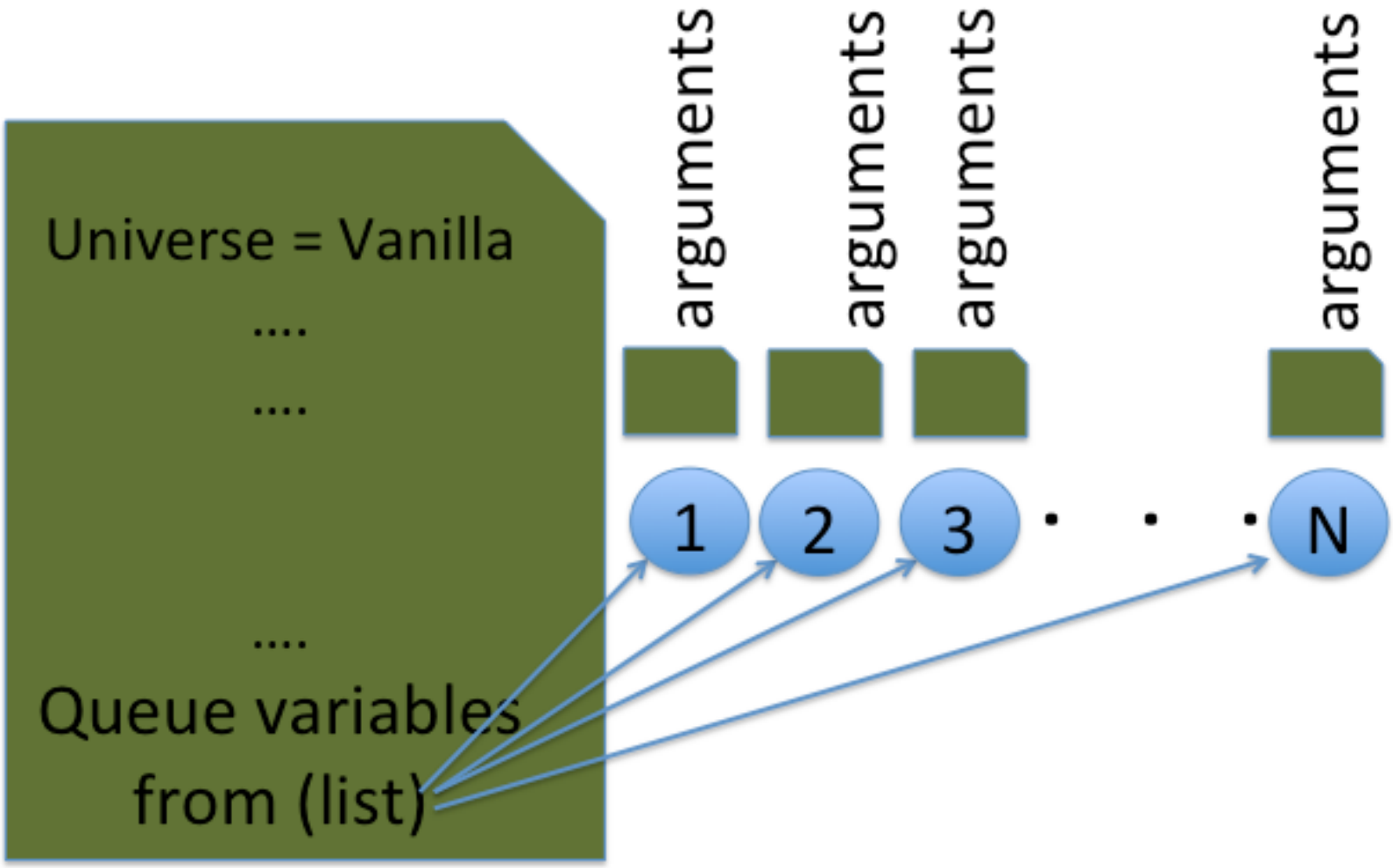
This is the two dimensional **Rosenbrock** function

Syntax: `python rosen_brock_brute_opt.py x_low x_high y_low y_high`
Example: `python rosen_brock_brute_opt.py -3 3 -2 2`



Code: `https://github.com/OSGConnect/tutorial-ScalingUp-Python/blob/master/rosen_brock_brute_opt.py`


Universe = Vanilla
....
....
arguments =
Queue
arguments =
Queue
...
arguments =
Queue
arguments
arguments
arguments
1
2
N
Jobs



Demo folder:
`~/science/Teaching/PP/lectures/12/code/HTCondor/OSG/HCCWorkshops/OSG/ScalingUp-Python/Example1`

Show the codes in an editor and execute the jobs


HEPiX
HTCondor
High Throughput Computing
HTCondor Workshop Autumn 2021

HTCondor channel on YouTube

https://go.wisc.edu/chtc-youtube