Additional DAGMan Features

- Provides other handy features for job management…
  - nodes can have PRE & POST scripts
  - failed nodes can be automatically re-tried a configurable number of times
  - job submission can be "throttled"

We've seen how 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 ?

What if each job needed to run for 20 days? What if I wanted to interrupt a job with a higher priority job?

Condor's Standard Universe to the rescue!

- Condor can support various combinations of features/environments in different "Universes"
- Different Universes provide different functionality for your job:
  - Vanilla – Run any Serial Job
  - Scheduler – Plug in a meta-scheduler
  - Standard – Support for transparent process checkpoint and restart

Process Checkpointing

- Condor's Process Checkpointing mechanism saves all the state of a process into a checkpoint file
  - Memory, CPU, I/O, etc.
- The process can then be restarted *from right where it left off*
- Typically no changes to your job's source code needed – however, your job must be *relinked* with Condor's Standard Universe support library

Relinking Your Job for submission to the Standard Universe

To do this, just place "***condor_compile***" in front of the command you normally use to link your job:

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

OR

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

OR

```
condor_compile make –f MyMakefile
```

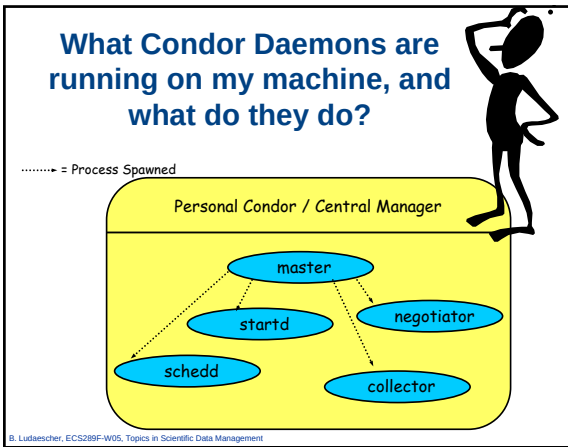

Limitations in the Standard Universe

- Condor's checkpointing is not at the kernel level. Thus in the Standard Universe the job may not
  - Fork()
  - Use kernel threads
  - Use some forms of IPC, such as pipes and shared memory
- Many typical scientific jobs are OK

When will Condor checkpoint your job?

- Periodically, if desired
  - For fault tolerance
- To free the machine to do a higher priority task (higher priority job, or a job from a user with higher priority)
  - Preemptive-resume scheduling
- When you explicitly run *condor_checkpoint*, *condor_vacate*, *condor_off* or *condor_restart* command

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



condor_master

- Starts up all other Condor daemons
- If there are any problems and a daemon exits, it restarts the daemon and sends email to the administrator
- Checks the time stamps on the binaries of the other Condor daemons, and if new binaries appear, the master will gracefully shutdown the currently running version and start the new version
- Acts as the server for many Condor remote administration commands:
  - ***condor_reconfig, condor_restart, condor_off, condor_on, condor_config_val***, etc.

condor_startd

- 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 soon)

condor_schedd

- Represents users to the Condor system
- 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, …***

condor_collector

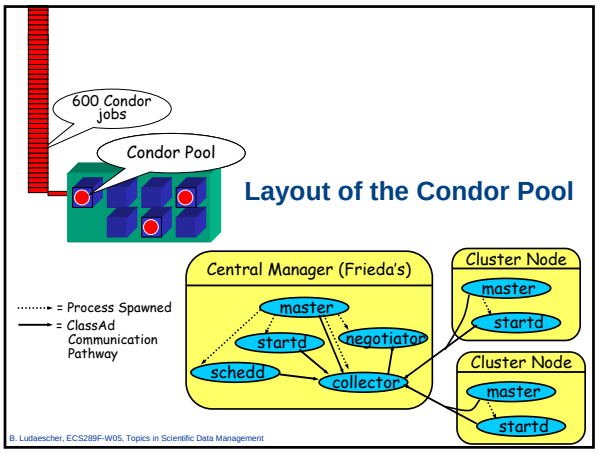
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
- Each daemon sends a periodic update called a "ClassAd" to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (***condor_status***)

condor_negotiator

- Performs "matchmaking" in Condor
- 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

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

- Frieda Installs Condor on all the dedicated Cluster nodes, and configures them with her machine as the central manager…
- Now her Condor Pool can run multiple jobs at once



condor_status

```
% condor_status

Name          OpSys      Arch   State      Activity   LoadAv Mem   ActvtyTime

haha.cs.wisc. IRIX65     SGI    Unclaimed  Idle       0.198   192  0+00:00:04
antipholus.cs 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  Claimed    Busy       0.810   511  0+00:01:45
ferdinand.cs. LINUX      INTEL  Claimed    Suspended  1.130   511  0+00:00:55
vm1@pinguino. LINUX      INTEL  Unclaimed  Idle       0.000   255  0+01:03:28
vm2@pinguino. LINUX      INTEL  Unclaimed  Idle       0.190   255  0+01:03:29
```


Frieda tries out parallel jobs…

- MPI Universe
  - MPI (Message Passing Interface): de facto standard for communication among nodes running a parallel program on a **distributed memory system**.
  - as Fortran, C, C++ libraries
- PVM Universe
  - Parallel Virtual Machine (PVM) is designed to allow a network of heterogeneous machines to be used as a single distributed parallel processor
- Schedule and start an MPICH job on dedicated resources

```
Executable = my-mpi-job
Universe = MPI
Machine_count = 8
queue
```


MPI

Message Passing Interface Quick Reference in C

```
#include <mpi.h>
```

Blocking Point-to-Point

Send a message to one process. (§3.2.1)
`int MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`

Receive a message from one process. (§3.2.4)
`int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`

Count received data elements. (§3.2.5)
`int MPI_Get_count (MPI_Status *status, MPI_Datatype datatype, int *count)`

Wait for message arrival. (§3.8)
`int MPI_Probe (int source, int tag, MPI_Comm comm, MPI_Status *status)`

Related Functions: MPI_Bsend, MPI_Ssend, MPI_Rsend, MPI_Buffer_attach, MPI_Buffer_detach, MPI_Sendrecv, MPI_Sendrecv_replace, MPI_Get_elements

Non-blocking Point-to-Point

Begin to receive a message. (§3.7.2)
`int MPI_Irecv (void *buf, int count, MPI_Datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)`

Complete a non-blocking operation. (§3.7.3)
`int MPI_Wait (MPI_Request *request, MPI_Status *status)`

Check or complete a non-blocking operation. (§3.7.3)
`int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status)`

Check message arrival. (§3.8)
`int MPI_Iprobe (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)`

Related Functions: MPI_Isend, MPI_Ibsend, MPI_Issend, MPI_Irsend, MPI_Request_free, MPI_Waitany, MPI_Testany, MPI_Waitall, MPI_Testall, MPI_Waitsome, MPI_Testsome, MPI_Cancel, MPI_Test_cancelled

Persistent Requests

Related Functions: MPI_Send_init, MPI_Bsend_init, MPI_Ssend_init, MPI_Rsend_init, MPI_Recv_init, MPI_Start, MPI_Startall

Derived Datatypes

Create a strided homogeneous vector. (§3.12.1)
`int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)`

Save a derived datatype (§3.12.4)
`int MPI_Type_commit (MPI_Datatype *datatype)`

Pack data into a message buffer. (§3.13)
`int MPI_Pack (void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outsize, int *position, MPI_Comm comm)`

Unpack data from a message buffer. (§3.13)
`int MPI_Unpack (void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)`

Determine buffer size for packed data. (§3.13)
`int MPI_Pack_size (int incount, MPI_Datatype datatype, MPI_Comm comm, int *size)`

Related Functions: MPI_Type_contiguous, MPI_Type_hvector, MPI_Type_indexed, MPI_Type_hindexed, MPI_Type_struct, MPI_Address, MPI_Type_extent, MPI_Type_size, MPI_Type_lb, MPI_Type_ub, MPI_Type_free

Collective

Send one message to all group members. (§4.4)
`int MPI_Bcast (void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)`

Receive from all group members. (§4.5)
`int MPI_Gather (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`

Send separate messages to all group members. (§4.6)
`int MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`

Combine messages from all group members. (§4.9.1)
`int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)`

Related Functions: MPI_Barrier, MPI_Gather, MPI_Scatterv, MPI_Allgather, MPI_Allgatherv, MPI_Allreduce, MPI_Alltoall, MPI_Alltoallv, MPI_Op_create, MPI_Op_free, MPI_Allreduce, MPI_Reduce_scatter, MPI_Scan

Groups

Related Functions: MPI_Group_size, MPI_Group_rank, MPI_Group_translate_ranks, MPI_Group_compare, MPI_Comm_group, MPI_Group_union, MPI_Group_intersection, MPI_Group_difference, MPI_Group_incl, MPI_Group_excl, MPI_Group_range_incl, MPI_Group_range_excl, MPI_Group_free

Basic Communicators

Count group members in communicator. (§5.4.1)
`int MPI_Comm_size (MPI_Comm comm, int *size)`

Determine group rank of self. (§5.4.1)
`int MPI_Comm_rank (MPI_Comm comm, int *rank)`

Duplicate with new context. (§5.4.2)
`int MPI_Comm_dup (MPI_Comm comm, MPI_Comm *newcomm)`

Split into categorized sub-groups. (§5.4.2)
`int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm)`

Related Functions: MPI_Comm_compare, MPI_Comm_create, MPI_Comm_free.

MPI_Comm_test_inter, MPI_Comm_remote_size, MPI_Comm_remote_group, MPI_Intercomm_create, MPI_Intercomm_merge

Communicators with Topology

Create with cartesian topology. (§6.5.1)
`int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)`

Suggest balanced dimension ranges. (§6.5.2)
`int MPI_Dims_create (int nnodes, int ndims, int *dims)`

Determine rank from cartesian coordinates. (§6.5.4)
`int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)`

Determine cartesian coordinates from rank. (§6.5.4)
`int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords)`

Determine ranks for cartesian shift. (§6.5.5)
`int MPI_Cart_shift (MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)`

Split into lower dimensional sub-grids. (§6.5.6)
`int MPI_Cart_sub (MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)`

Related Functions: MPI_Graph_create, MPI_Topo_test, MPI_Graphdims_get, MPI_Graph_get, MPI_Cartdim_get, MPI_Cart_get, MPI_Graph_neighbors_count, MPI_Graph_neighbors, MPI_Cart_map, MPI_Graph_map

Communicator Caches

Related Functions: MPI_Keyval_create, MPI_Keyval_free, MPI_Attr_put, MPI_Attr_get, MPI_Attr_delete

LAM & MPI Information

1224 Kinnear Rd.
Columbus, Ohio 43212
614-292-8492

lam@tbag.osc.edu

http://www.osc.edu/lam.html
ftp://tbag.osc.edu/pub/lam

Error Handling

Related Functions: MPI_Errhandler_create, MPI_Errhandler_set, MPI_Errhandler_get, MPI_Errhandler_free, MPI_Error_string, MPI_Error_class

Environmental

Determine wall clock time. (§7.4)
`double MPI_Wtime (void)`

Initialize MPI. (§7.5)
`int MPI_Init (int *argc, char ***argv)`

Cleanup MPI. (§7.5)
`int MPI_Finalize (void)`

Related Functions: MPI_Get_processor_name, MPI_Wtick, MPI_Initialized, MPI_Abort, MPI_Pcontrol

Constants

Wildcards (§3.2.4)
`MPI_ANY_TAG, MPI_ANY_SOURCE`

Elementary Datatypes (§3.2.2)
`MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED`

Reserved Communicators (§5.2.4)
`MPI_COMM_WORLD, MPI_COMM_SELF`

Reduction Operations (§4.9.2)
`MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR`

LAM Quick Reference

LAM / MPI Extensions

Spawn processes.
`int MPIL_Spawn (MPI_Comm comm, char *app, int root, MPI_Comm *child_comm);`

Get communicator ID.
`int MPIL_Comm_id (MPI_Comm comm, int *id);`

Deliver an asynchronous signal.
`int MPIL_Signal (MPI_Comm comm, int rank, int signo);`

Enable trace collection.
`int MPIL_Trace_on (void);`

Related Functions: MPIL_Comm_gorent, MPIL_Universe_size, MPIL_Type_id, MPIL_Comm_grs, MPIL_Trace_off

Session Management

Confirm a group of hosts.
`recon -v <hostfile>`

Start LAM on a group of hosts.
`lamboot -v <hostfile>`

Terminate LAM.
`wipe -v <hostfile>`

Hostfile Syntax
```
# comment
<hostname> <userid>
<hostname> <userid>
...etc...
```

Compilation

Compile a program for LAM / MPI.
`hcc -o <binary> <source> -I<incdir> -L<libdir> -lcilib> -lmpi`

Processes and Messages

Start an SPMD application.
`mpirun -v -s <src_node> -o <copies> <nodes> <program> -- <args>`

Start a MIMD application.
`mpirun -v <appfile>`

Appfile Syntax
```
# comment
<program> -s <src_node> <nodes> -- <args>
<program> -s <src_node> <nodes> -- <args>
...etc...
```

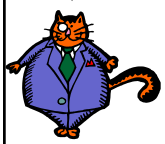
Examine the state of processes.
`mpitask`

Examine the state of messages.
`mpimsg`

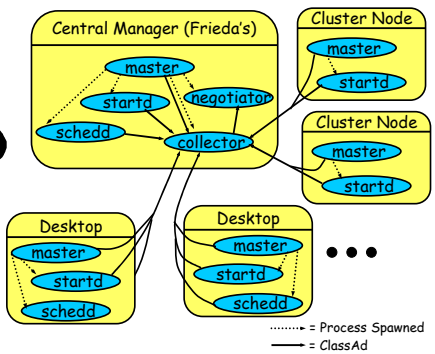
Cleanup all processes and messages.
`lamclean -v`

Layout of the Condor Pool

(Boss Fat Cat)



The Boss says Frieda can add her co-workers' desktop machines into her Condor pool as well… but only if they can also submit jobs.



Some of the machines in the Pool do not have enough memory or scratch disk space to run 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 = my_job
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Queue 600
```


Specify Rank!

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

```
Universe   = vanilla
Executable = my_job
Arguments  = -arg1 –arg2
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 600
```


How can my jobs access their data files?

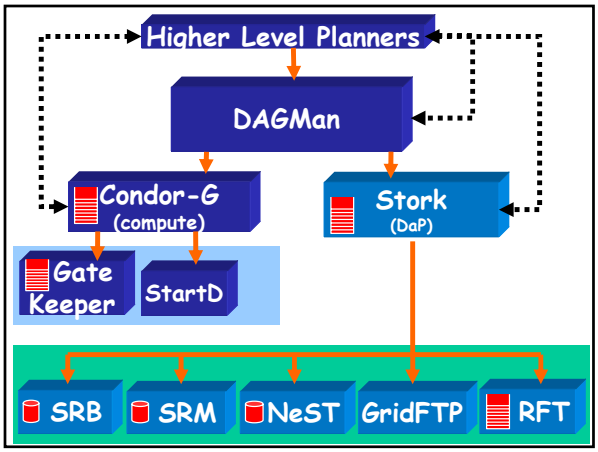
Access to Data in Condor

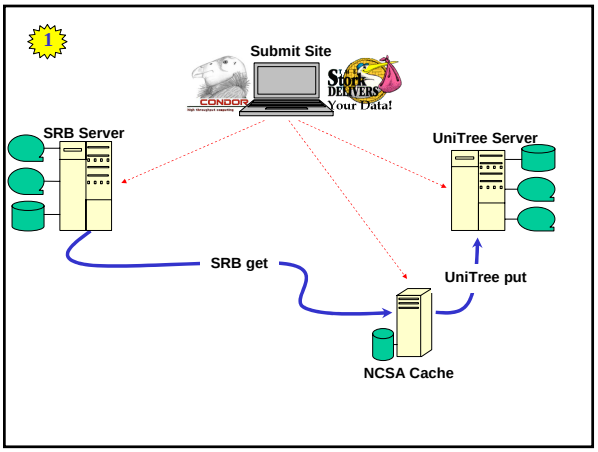
- Use Shared Filesystem if available
- No shared filesystem?
  - Condor can transfer files
    - Can automatically send back changed files
    - Atomic transfer of multiple files
  - Remote I/O Socket
  - Standard Universe can use *Remote System Calls*

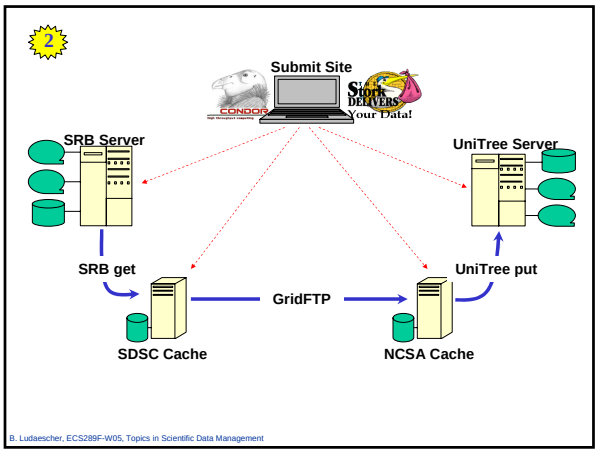
Condor File Transfer

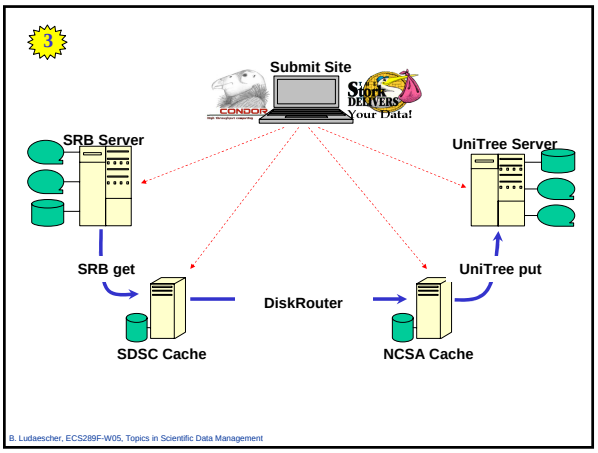
- Set ShouldTransferFiles
  - YES : Always transfer files to execution site
  - NO : Rely on a shared filesystem
  - IF_NEEDED : will automatically transfer the files if the submit and execute machine are not in the same FileSystemDomain

```
Universe   = vanilla
Executable = my_job
Requirements = Memory >= 256 && Disk > 10000
ShouldTransferFiles = IF_NEEDED
Transfer_input_files = dataset$(Process), common.data
Transfer_output_files = TheAnswer.dat
Queue 600
```







Outcomes of the Study

1. Stork interacted easily and successfully with different underlying systems: SRB, UniTree, GridFTP and Diskrouter.

2. We had the chance to compare different pipeline topologies and configurations:

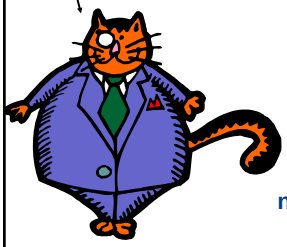
Configuration	End-to-end rate (MB/sec)
1	5.0
2	3.2
3	5.95

3. Almost all possible network, server, and software failures were recovered automatically.

B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management

Policy Configuration

(Boss Fat Cat)



I am adding nodes to the Cluster… *but the Engineering Department has priority on these nodes.*

The Cluster is fine. But not the desktop machines. *Condor can only use the desktops when they would otherwise be idle.*

B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management

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 — Specify Dependencies
- condor_checkpoint — Force a checkpoint
- condor_compile — Link Condor library

B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management

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

B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management

Condor Job Universes

- Serial Jobs
  - Vanilla Universe
  - Standard Universe
- Scheduler Universe
- Parallel Jobs
  - MPI Universe
  - PVM Universe
- Java Universe

Java Universe Job

condor_submit ⟶

```
universe = java
executable = Main.class
jar_files = MyLibrary.jar
input = infile
output = outfile
arguments = Main 1 2 3
queue
```


Why not use Vanilla Universe for Java jobs?

- Java Universe provides more than just inserting "java" at the start of the execute line
  - Knows which machines have a JVM installed
  - Knows the location, version, and performance of JVM on each machine
  - 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.

Java support, cont.

```
condor_status -java

Name          JavaVendor  Ver    State      Activity LoadAv Mem

aish.cs.wisc. Sun Microsy 1.2.2  Owner      Idle    0.000   249
anfrom.cs.wis Sun Microsy 1.2.2  Owner      Idle    0.030   249
babe.cs.wisc. Sun Microsy 1.2.2  Claimed    Busy    1.120   123
...
```


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>

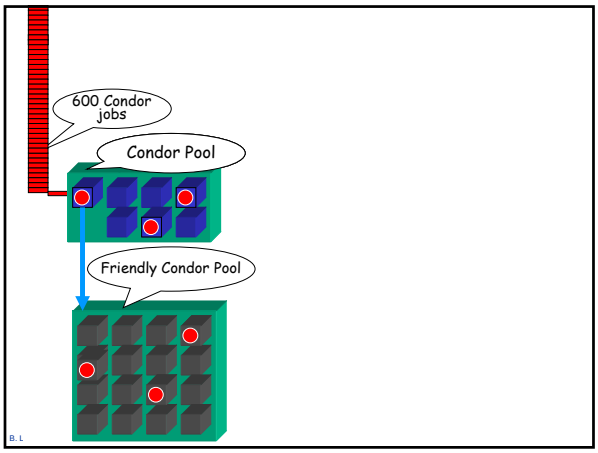
Job Policy Examples

- 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:

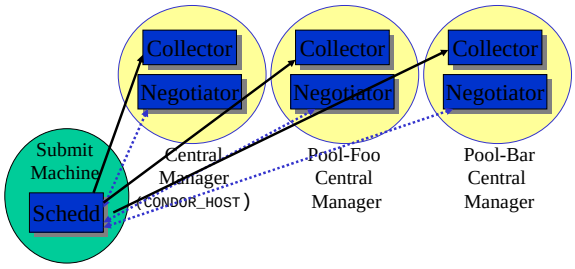
  on_exit_hold = ((ExitBySignal==False) && (ExitSignal != 0)) || ((ServerStartTime – JobStartDate) < 3600)
- Place on hold if job has spent more than 50% of its time suspended:

  periodic_hold = CumulativeSuspensionTime > (RemoteWallClockTime / 2.0)



How Flocking Works

- Add a line to your condor_config :
  `FLOCK_HOSTS = Pool-Foo, Pool-Bar`



Condor Flocking

- Remote pools are contacted *in the order specified* until jobs are satisfied
- The list of remote pools is a property of the Schedd, not the Central Manager
  - So different users can Flock to different pools
  - And remote pools can allow specific users
- User-priority system is "flocking-aware"
  - A pool's local users can have priority over remote users "flocking" in.

Condor Flocking, cont.

- Flocking is "Condor" specific technology…
- Frieda also has access to Globus resources she wants to use
  - She has certificates and access to Globus gatekeepers at remote institutions
- But Frieda wants Condor's queue management features for her Globus jobs!
- She installs Condor-G so she can submit "Globus Universe" jobs to Condor

Condor-G: Globus + Condor

the globus project www.globus.org

Globus

- middleware deployed across entire Grid
- remote access to computational resources
- dependable, robust data transfer

Condor High Throughput Computing

Condor

- job scheduling across multiple resources
- strong fault tolerance with checkpointing and migration
- layered over Globus as "personal batch system" for the Grid