

Using Condor

An Introduction

EVERLAB WORKSHOP

Condor Project
Computer Sciences Department
University of Wisconsin-Madison
condor-admin@cs.wisc.edu
<http://www.cs.wisc.edu/condor>



First Steps

- > Jobs are submitted from specific machines/slices
- > el_condor@planet9.huji.ac.il
- > huji_condor@planet8.huji.ac.il
- > ucl_condor@calc1.info.ucl.ac.be
- > tau_condor@b10.evergrow.iucc.ac.il
- > sics_condor@evgsics1.sics.se

Condor Submissions

- > Each PI should add local users to their XXX_condor slice
- > Users without a home cluster will be added to the el_condor slice

Getting Started: Submitting Jobs to Condor

- > Choosing a "Universe" for your job
 - Just use VANILLA for now
- > Make your job "batch-ready"
- > Creating a submit description file
- > Run `condor_submit` on your submit description file

Making your job batch-ready

- > Must be able to run in the background: no interactive input, windows, GUI, etc.
- > Can still use `STDIN`, `STDOUT`, and `STDERR` (the keyboard and the screen), but files are used for these instead of the actual devices
- > Organize data files

Creating a Submit

Description File

- > A plain ASCII text file
- > Condor does **not** care about file extensions
- > 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.

Simple Submit Description 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    = my_job
Queue
```

Running condor_submit

- > You give `condor_submit` the name of the submit file you have created:
 - `condor_submit my_job.submit`
- > `condor_submit` parses the submit file, checks for it errors, and creates a "ClassAd" that describes your job(s)
 - ClassAds: Condor's internal data representation
 - Similar to classified ads (as the name implies)
 - Represent an object & it's attributes
 - Can also describe what an object matches with

The Job Queue

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

Running condor_submit

```
% condor_submit my_job.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	my_job

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

```
%
```

More information about jobs

- > Controlled by submit file settings
- > Condor sends you email about events
 - Turn it off: `Notification = Never`
 - Only on errors: `Notification = Error`
- > Condor creates a log file (user log)
 - "The Life Story of a Job"
 - Shows all events in the life of a job
 - Always have a log file
 - To turn it on: `Log = filename`

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)

Usr 0 00:00:37, Sys 0 00:00:00 - Run Remote Usage

Usr 0 00:00:00, Sys 0 00:00:05 - Run Local Usage

Usr 0 00:00:37, Sys 0 00:00:00 - Total Remote Usage

Usr 0 00:00:00, Sys 0 00:00:05 - Total Local Usage

9624 - Run Bytes Sent By Job

7146159 - Run Bytes Received By Job

9624 - Total Bytes Sent By Job

7146159 - Total Bytes Received By Job

...

Another Submit Description File

```
# Example 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   = /home/frieda/condor/my_job.condor
Log           = my_job.log
Input        = my_job.stdin
Output       = my_job.stdout
Error        = my_job.stderr
Arguments    = -arg1 -arg2
InitialDir   = /home/frieda/condor/run_1
Queue
```

"Clusters" and "Processes"

- If your submit file describes multiple jobs, we call this a "cluster"
- Each cluster has a unique "cluster number"
- Each 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 ("20.1")
 - A cluster can have only one process ("21.0")

Example Submit Description File for a Cluster

```
# Example submit description file that defines a  
# cluster of 2 jobs with separate working directories  
Universe      = vanilla  
Executable    = my_job  
log           = my_job.log  
Arguments     = -arg1 -arg2  
Input         = my_job.stdin  
Output        = my_job.stdout  
Error         = my_job.stderr  
InitialDir    = run_0  
Queue         · Becomes job 2.0  
InitialDir    = run_1  
Queue         · Becomes job 2.1
```

Submitting The Job

```
% condor_submit my_job.submit-file
```

```
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	my_job
2.0	frieda	4/15 06:56	0+00:00:00	I	0	0.0	my_job
2.1	frieda	4/15 06:56	0+00:00:00	I	0	0.0	my_job

```
3 jobs; 2 idle, 1 running, 0 held
```

```
%
```

Submit Description File for a BIG Cluster of Jobs

- > The initial directory for each job can be specified as `run_$(Process)`, and instead of submitting a single job, we use "Queue 600" to submit 600 jobs at once
- > The `$(Process)` macro will be expanded to the process number for each job in the cluster (0 - 599), so we'll have "run_0", "run_1", ... "run_599" directories
- > All the input/output files will be in different directories!

Submit Description File for a BIG Cluster of Jobs

```
# Example condor_submit input file that defines  
# a cluster of 600 jobs with different directories
```

```
Universe      = vanilla  
Executable   = my_job  
Log           = my_job.log  
Arguments    = -arg1 -arg2  
Input        = my_job.stdin  
Output       = my_job.stdout  
Error        = my_job.stderr
```

```
InitialDir   = run_$(Process)
```

```
Queue 600
```

·run_0 ... run_599

·Becomes job 3.0 ... 3.599

Using condor_rm

- > If you want to remove a job from the Condor queue, you use `condor_rm`
- > You can only remove jobs that you own (you can't run `condor_rm` on someone else's jobs unless you are root)
- > You can give specific job ID's (cluster or cluster.proc), or you can remove all of your jobs with the "-a" option.

□ `condor_rm 21.1` · Removes a single job

□ `condor_rm 21` · Removes a whole cluster

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

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
 - Can be encrypted over the wire
 - Remote I/O Socket
 - Standard Universe can use remote system calls (more on this later)

Condor File Transfer

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

`Universe = vanilla`

`Executable = my_job`

`Log = my_job.log`

`ShouldTransferFiles = IF_NEEDED`

`Transfer_input_files = dataset$(Process), common.data`

`Transfer_output_files = TheAnswer.dat`

`Queue 600`

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

Log = my_job.log

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.
- > Higher the Rank, the better the match

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

We've seen how Condor can:

- ... keeps an eye on your jobs and will keep you posted on their progress
- ... implements your policy on the execution order of the jobs
- ... keeps a log of your job activities

My jobs run for 20 days...

- > What happens when they get pre-empted?
- > How can I add fault tolerance to my jobs?



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 the entire 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 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 of the Standard Universe

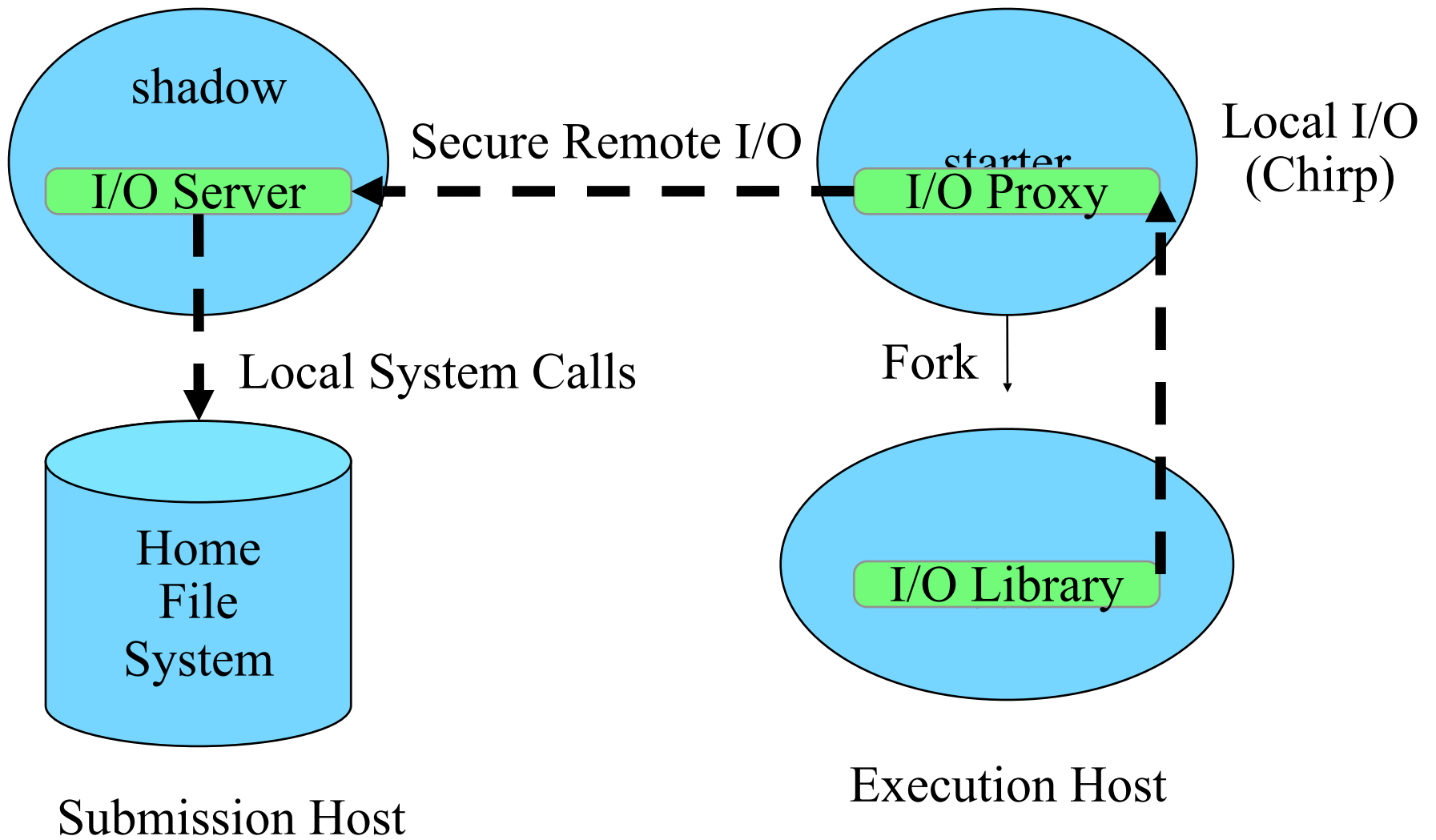
- > Condor's checkpointing is not at the kernel level. Thus in the Standard Universe the job may not:
 - Fork()
 - Use kernel threads
 - Use some forms of IPC, such as pipes and shared memory
- > Many typical scientific jobs are OK

When will Condor checkpoint your job?

- Periodically, if desired
 - For fault tolerance
- 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

Remote I/O Socket

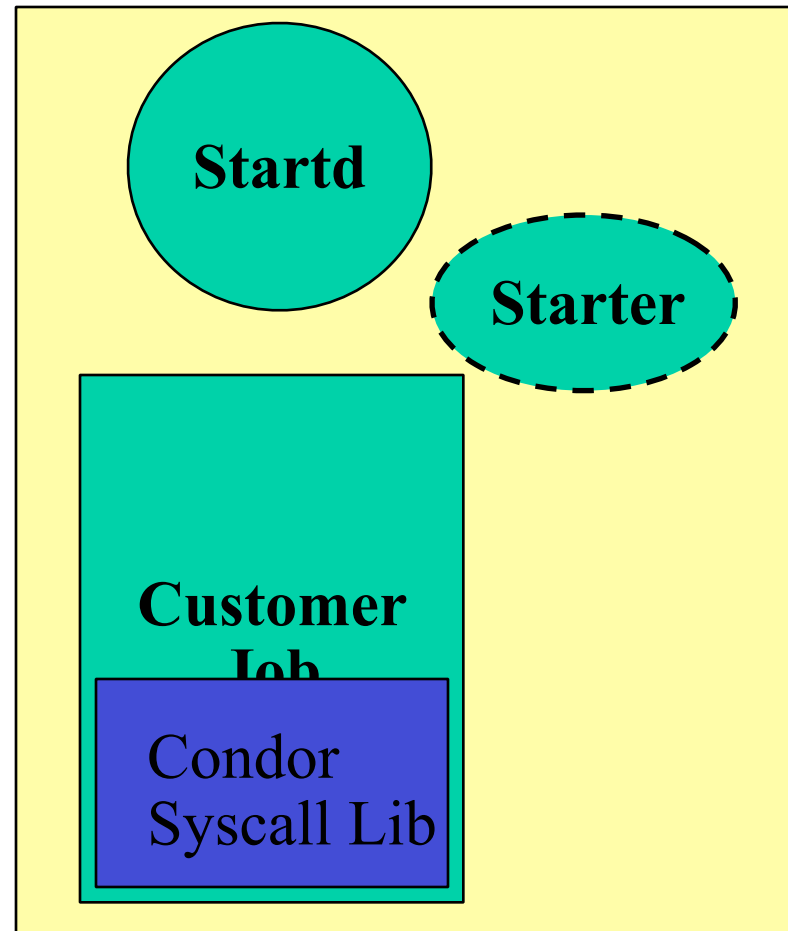
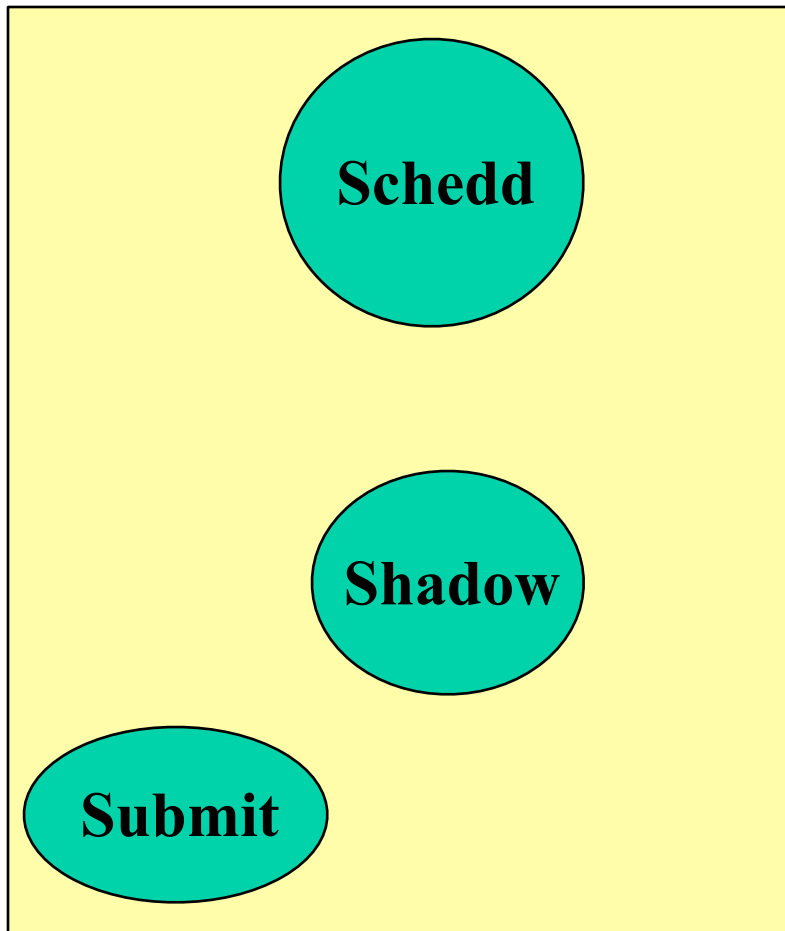
- > Job can request that the condor_starter process on the execute machine create a **Remote I/O Socket**
- > Used for online access of file on submit machine - without Standard Universe.
 - Use in Vanilla, Java, ...
- > Libraries provided for Java and for C, e.g. :
Java: FileInputStream -> ChirpInputStream
C : open() -> chirp_open()



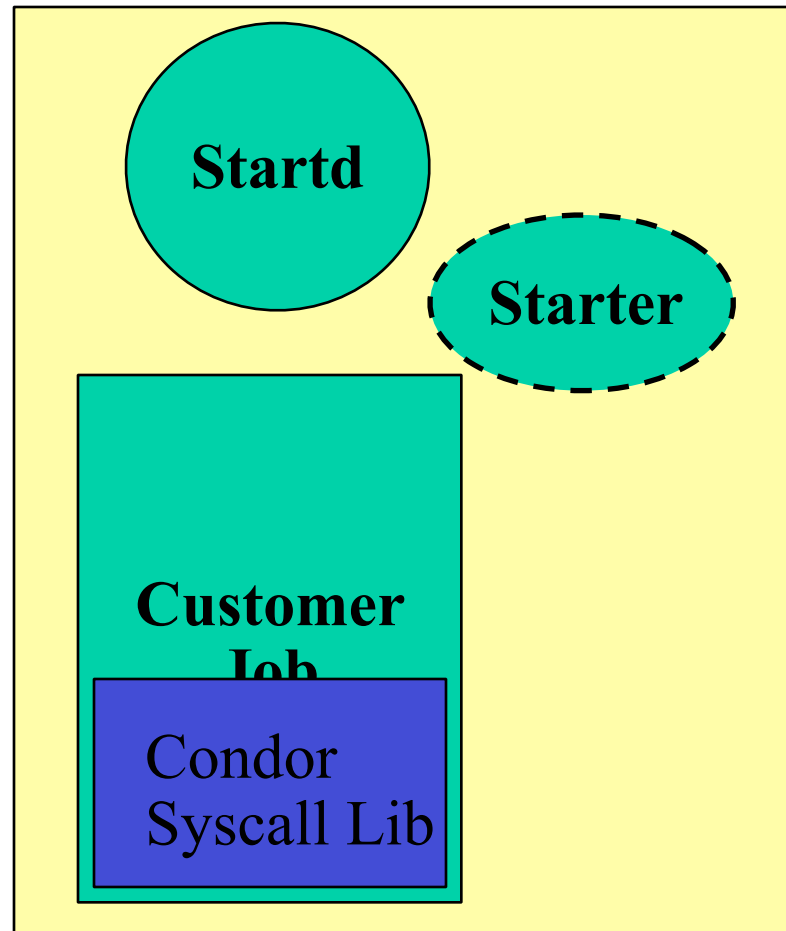
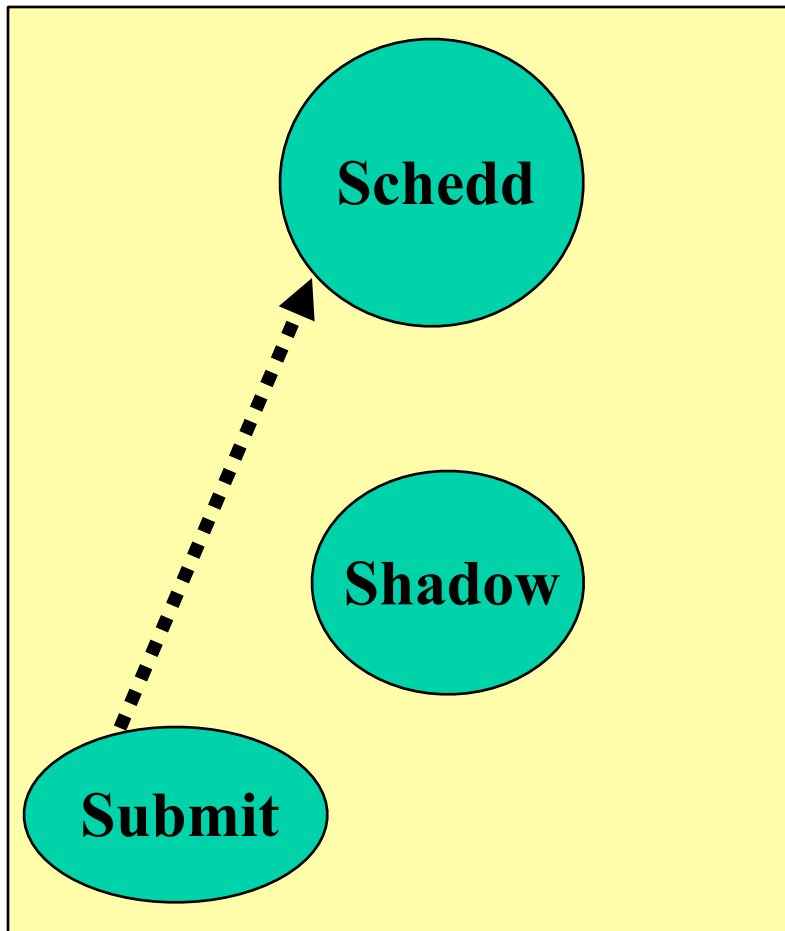
Remote System Calls

- I/O System calls are trapped and sent back to submit machine
- Allows Transparent Migration Across Administrative Domains
 - Checkpoint on machine A, restart on B
- No Source Code changes required
- Language Independent
- Opportunities for Application Steering
 - Example: Condor tells customer process "how" to open files

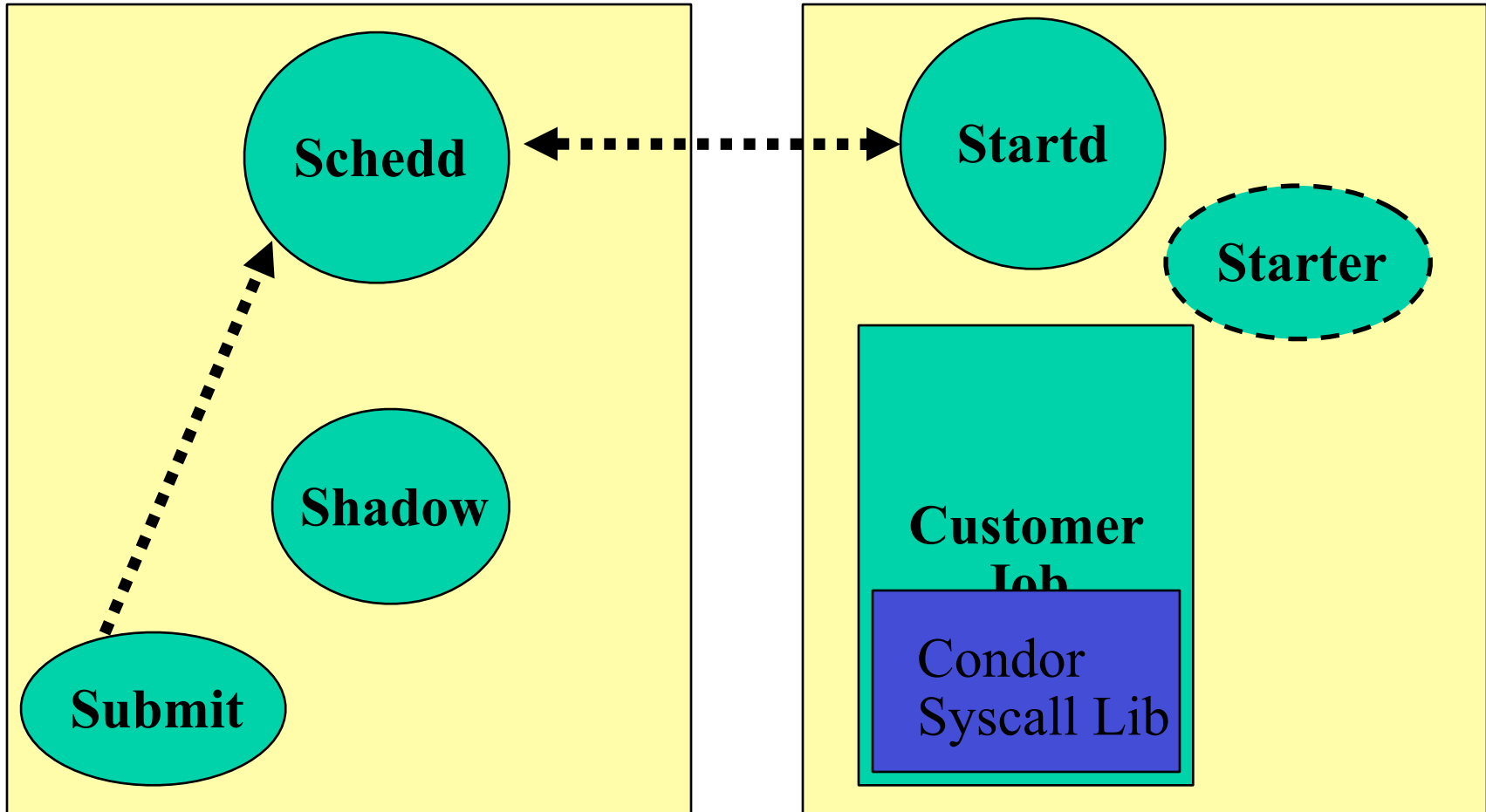
Job Startup



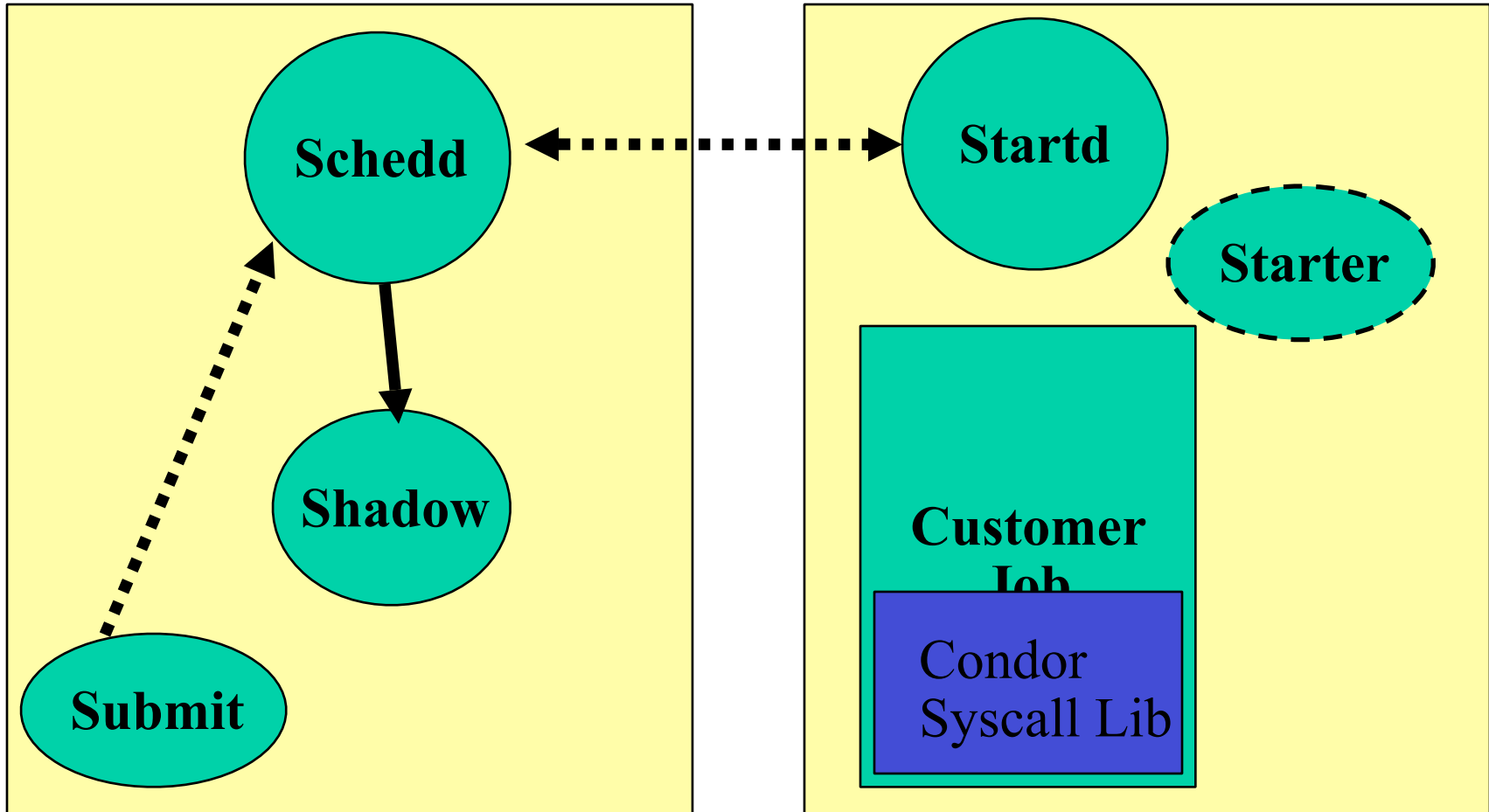
Job Startup



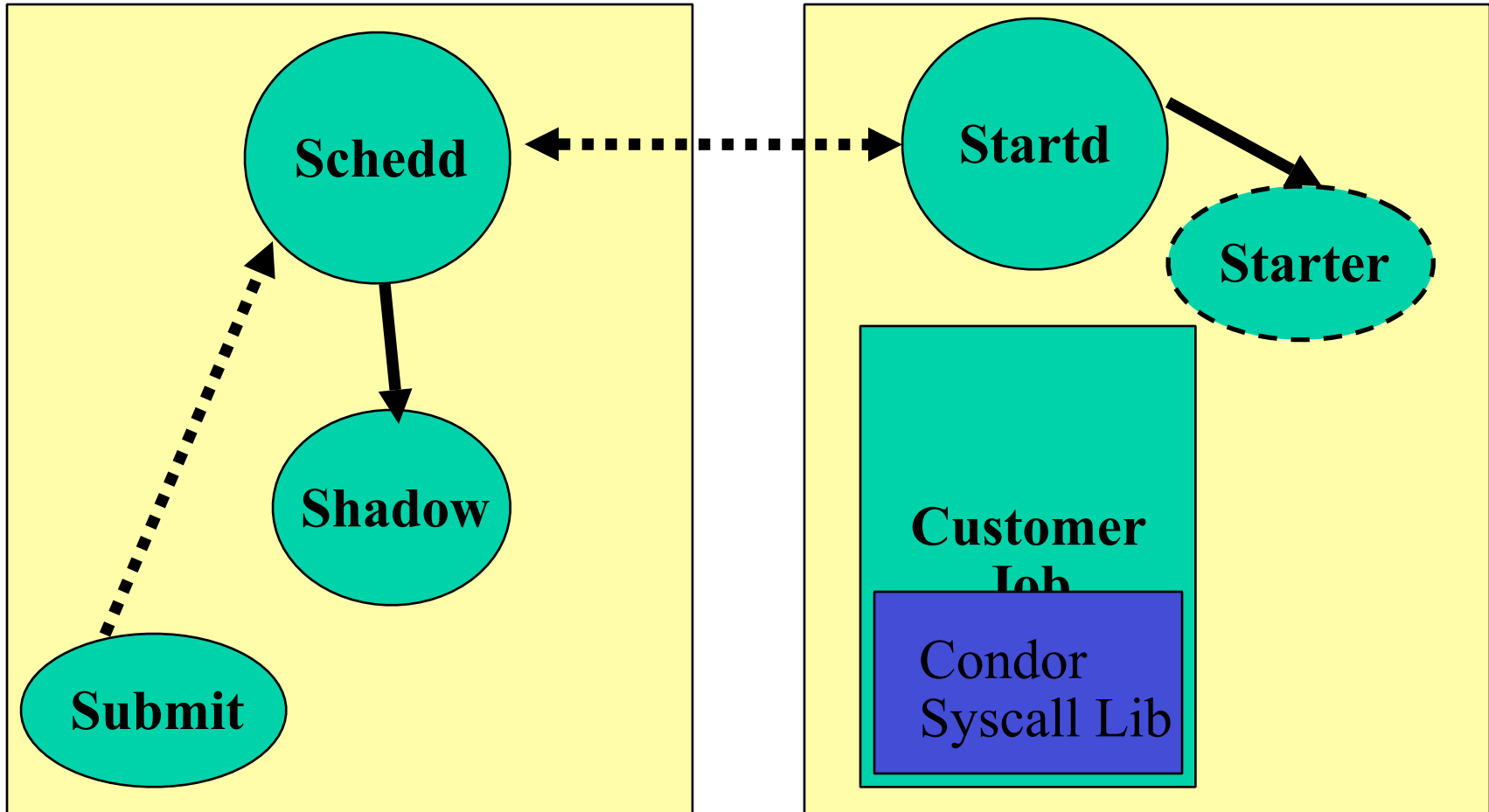
Job Startup



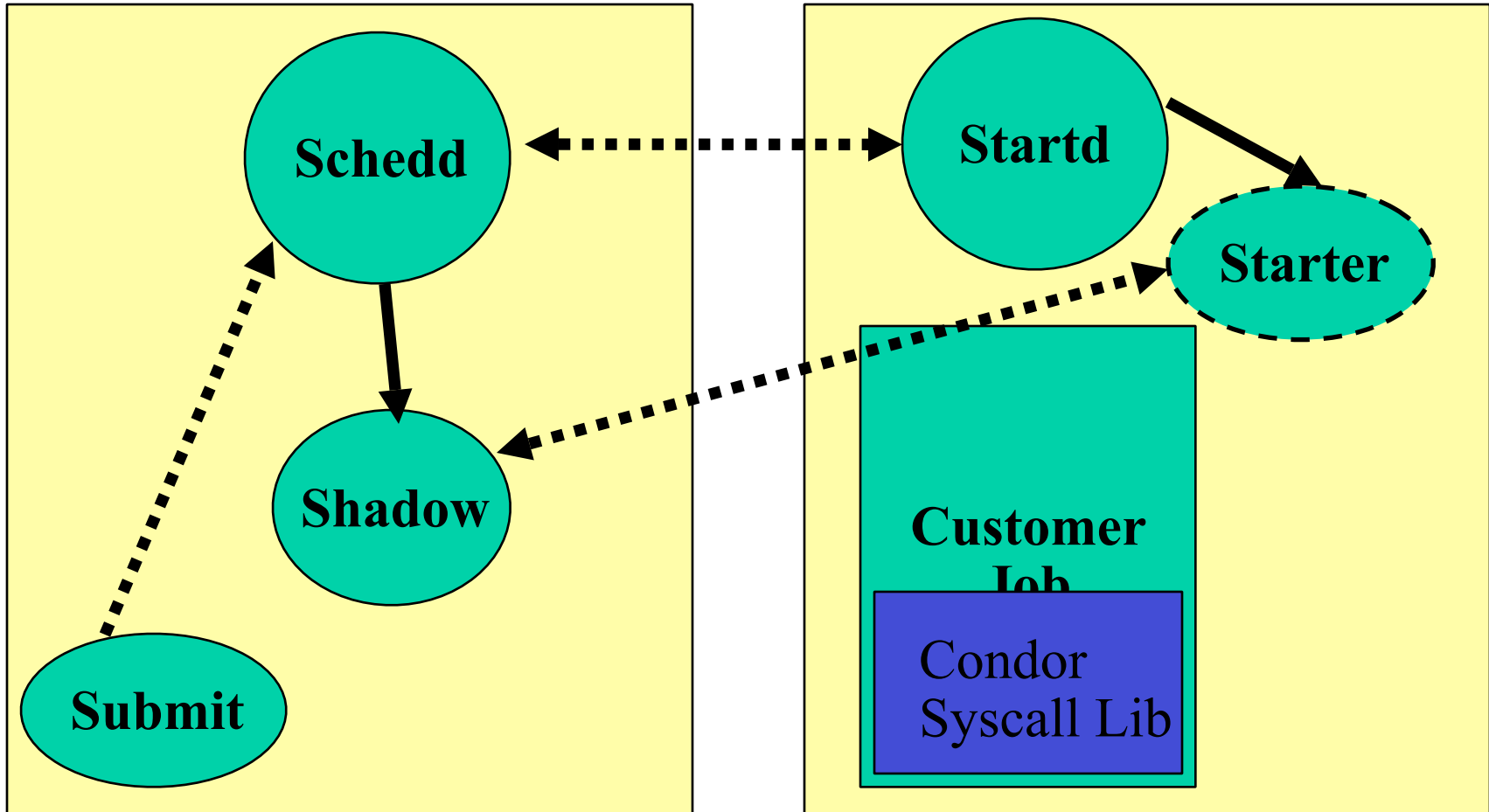
Job Startup



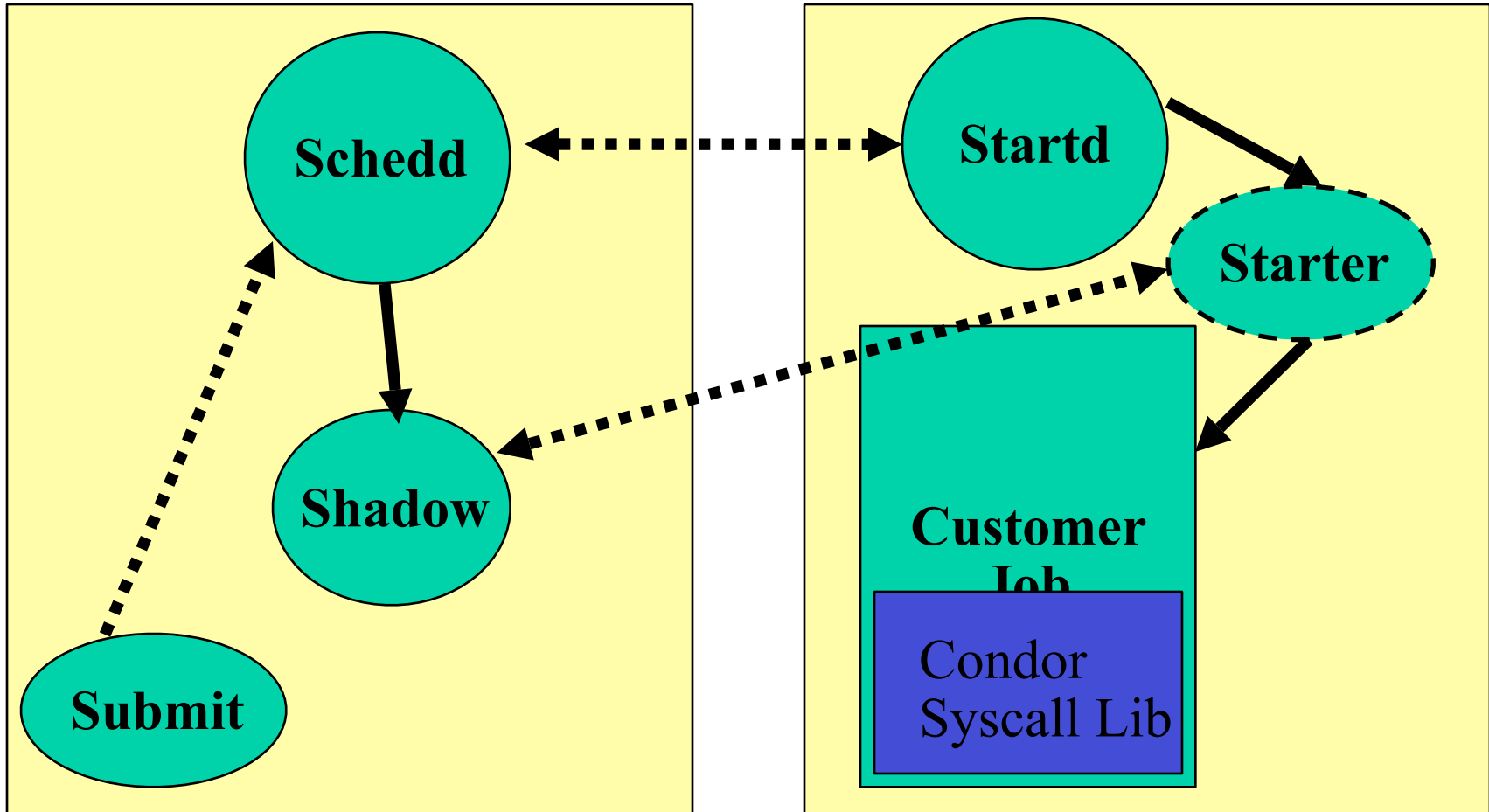
Job Startup



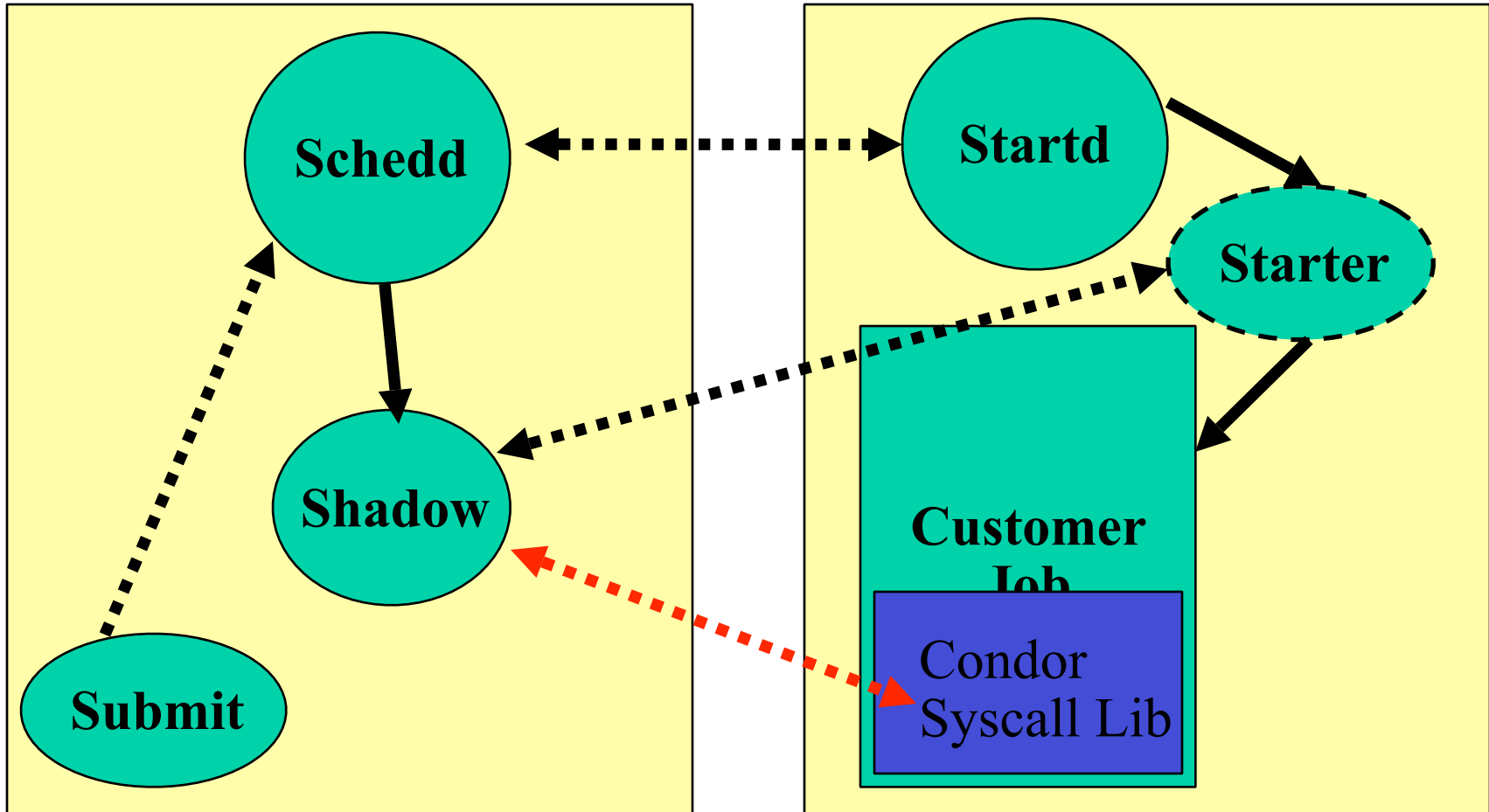
Job Startup



Job Startup



Job Startup



condor_q -io

```
c01(69)% condor_q -io
```

```
-- Submitter: c01.cs.wisc.edu : <128.105.146.101:2996> : c01.cs.wisc.edu
```

ID	OWNER	READ	WRITE	SEEK	XPUT	BUFSIZE	BLKSIZE
72.3	edayton	[no i/o data collected yet]					
72.5	edayton	6.8 MB	0.0 B	0	104.0 KB/s	512.0 KB	32.0 KB
73.0	edayton	6.4 MB	0.0 B	0	140.3 KB/s	512.0 KB	32.0 KB
73.2	edayton	6.8 MB	0.0 B	0	112.4 KB/s	512.0 KB	32.0 KB
73.4	edayton	6.8 MB	0.0 B	0	139.3 KB/s	512.0 KB	32.0 KB
73.5	edayton	6.8 MB	0.0 B	0	139.3 KB/s	512.0 KB	32.0 KB
73.7	edayton	[no i/o data collected yet]					

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

My jobs have have dependencies...

Can Condor help solve my dependency problems?

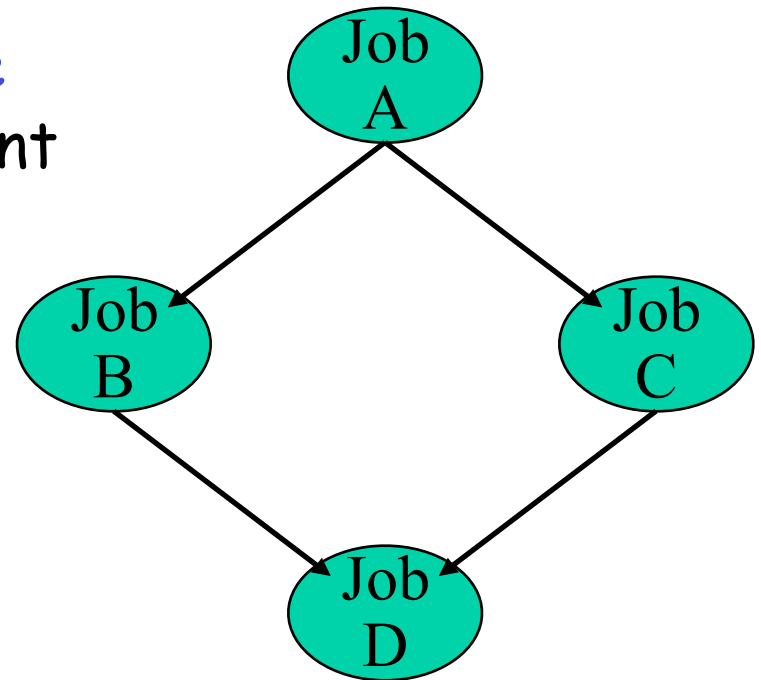


Frieda learns DAGMan

- > Directed Acyclic Graph Manager
- > DAGMan allows you to specify the **dependencies** between your Condor jobs, so it can **manage** them automatically for you.
- > (e.g., "Don't run job "B" until job "A" has completed successfully.")

What is a DAG?

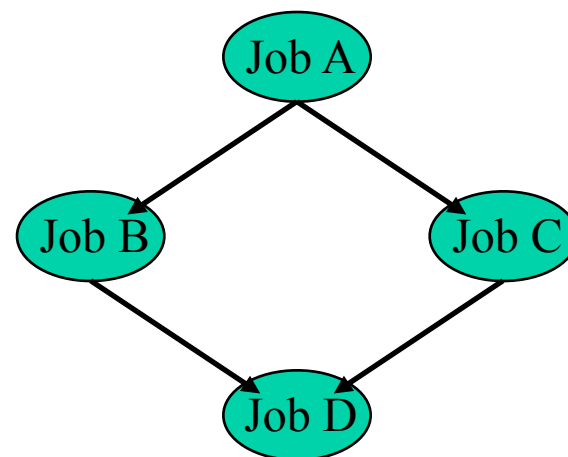
- > A DAG is the **data structure** used by DAGMan to represent these dependencies.
- > Each job is a **"node"** in the DAG.
- > Each node can have any number of **"parent"** or **"children"** nodes - as long as there are **no loops!**



Defining a DAG

- > A DAG is defined by a `.dag` file, listing each of its nodes and their dependencies:

```
# diamond.dag
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub
Parent A Child B C
Parent B C Child D
```



- > each node will run the Condor job specified by its accompanying `Condor submit file`

Submitting a DAG

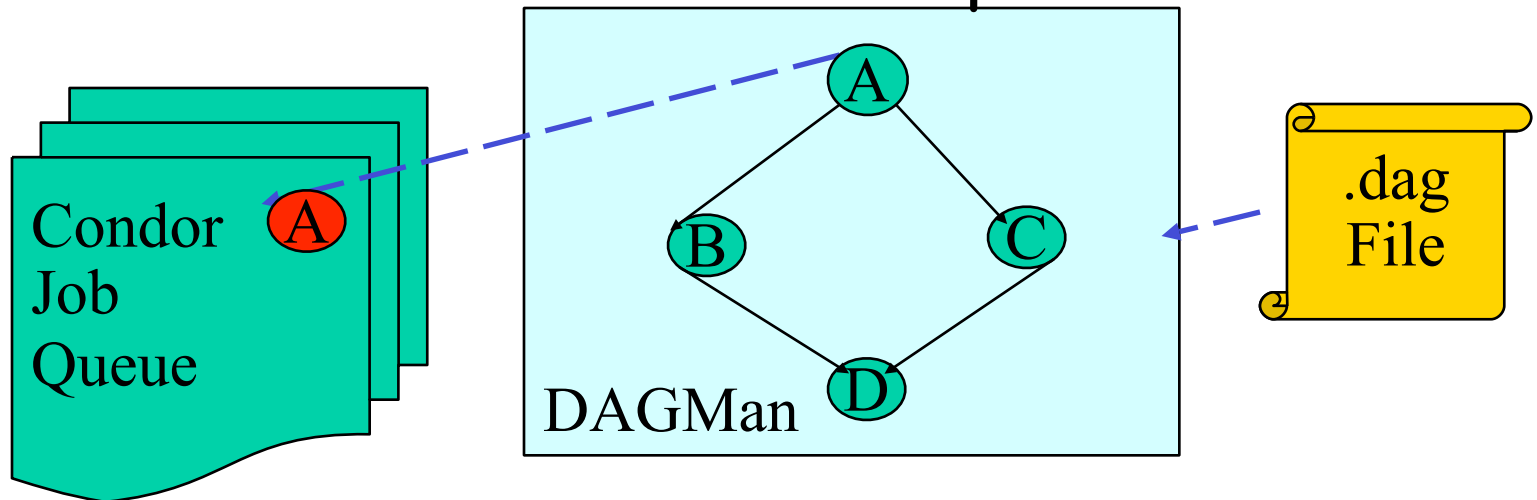
- > To start your DAG, just run `condor_submit_dag` with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% condor_submit_dag diamond.dag
```

- > `condor_submit_dag` submits a Scheduler Universe Job with DAGMan as the executable.
- > Thus the DAGMan daemon itself runs as a Condor job, so you don't have to baby-sit it.

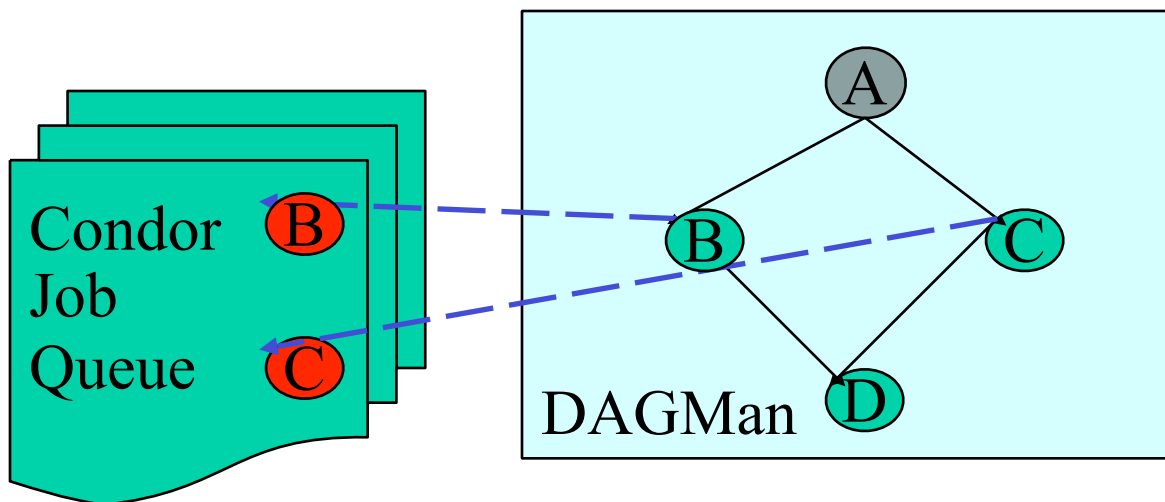
Running a DAG

- > DAGMan acts as a "meta-scheduler", managing the submission of your jobs to Condor based on the DAG dependencies.



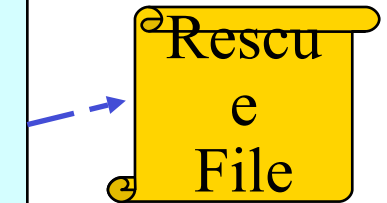
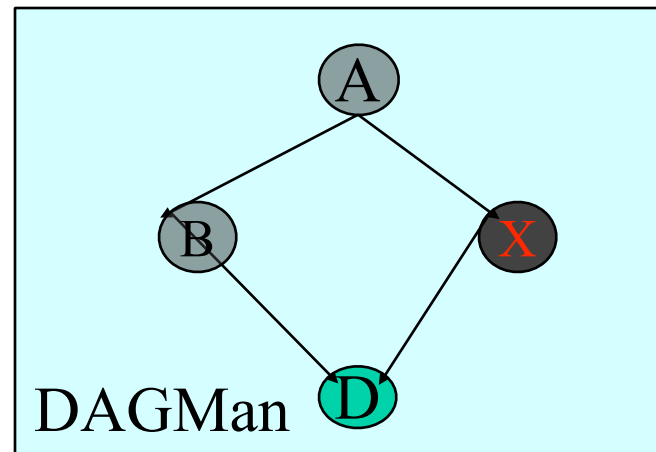
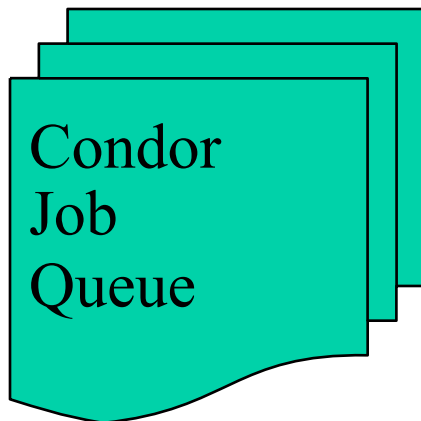
Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.



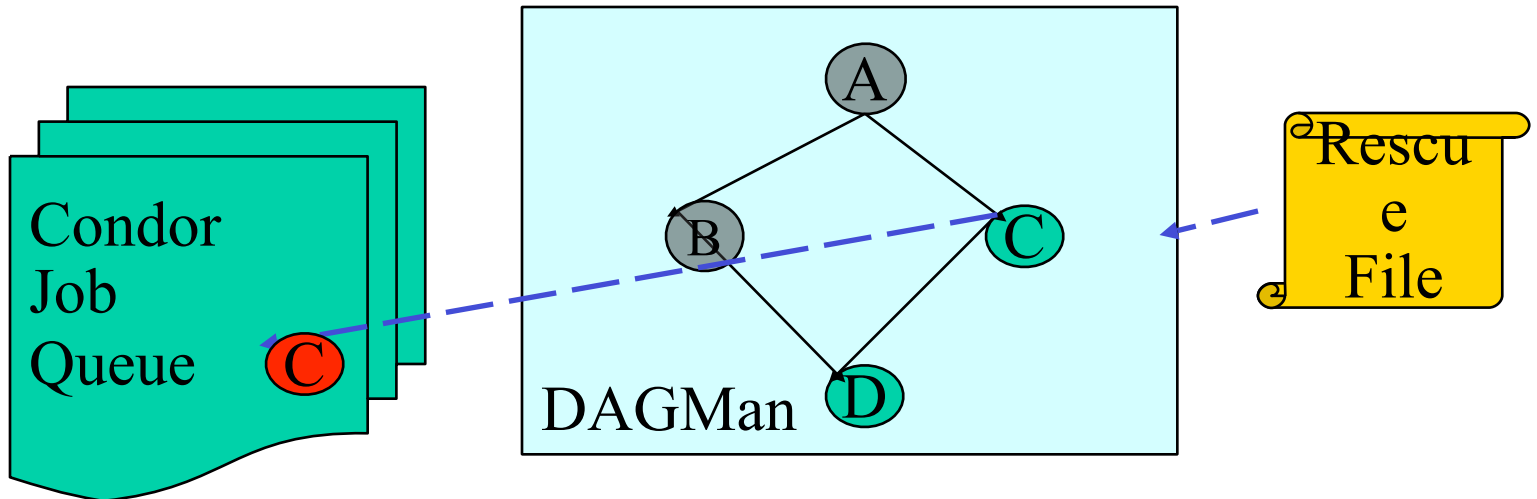
Running a DAG (cont'd)

- > In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a "rescue" file with the current state of the DAG.



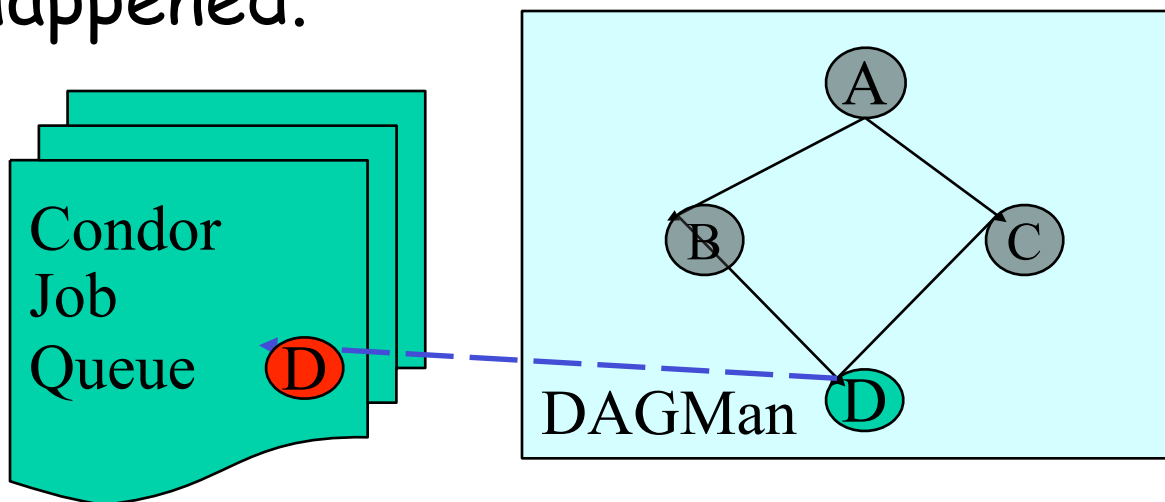
Recovering a DAG

- > Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.



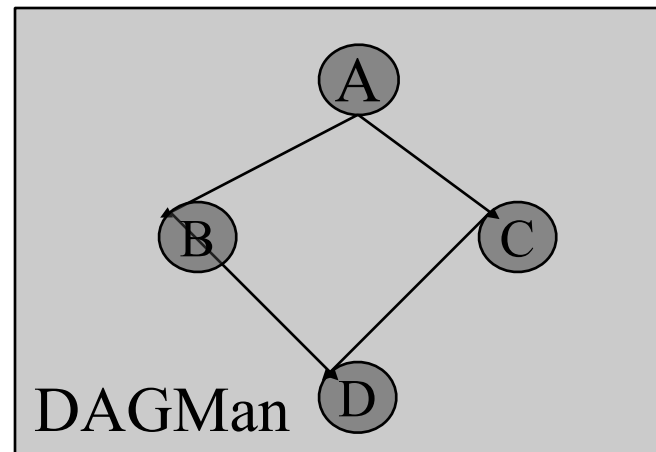
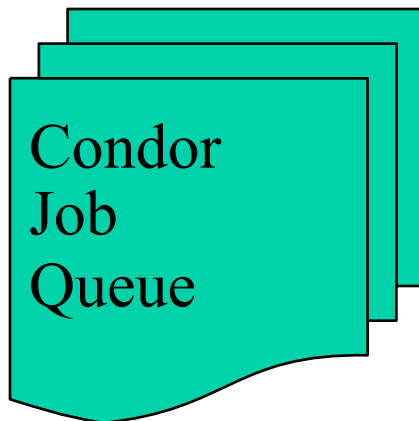
Recovering a DAG (cont'd)

- Once that job completes, DAGMan will continue the DAG as if the failure never happened.



Finishing a DAG

- Once the DAG is complete, the DAGMan job itself is finished, and exits.



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”

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

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

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)`

Thank you!

Check us out on the Web:

<http://www.condorproject.org>

Email:

condor-admin@cs.wisc.edu