Open Science Grid

More HTCondor

Monday AM, Lecture 2

Ian Ross
Center for High Throughput Computing
University of Wisconsin-Madison

Questions so far?

Goals for this Session

- Understand the mechanisms of HTCondor (and HTC in general) a bit more deeply
- Use a few more HTCondor features
- Run more (and more complex) jobs at once

Why is HTC Difficult?

- System must track jobs, machines, policy, …
- System must recover gracefully from failures
- Try to use all available resources, all the time
- Lots of variety in users, machines, networks, ...
- Sharing is hard (e.g. Policy, security)

MAIN PARTS OF HTCONDOR

Main Parts of HTCondor

Function
Track waiting/running jobs
Track available machines
Match jobs and machines
Manage one machine
Manage one job (on submitter)
Manage one job (on machine)

Main Parts of HTCondor

Function	HTCondor Name
Track waiting/running jobs	schedd (“sked-dee”)
Track available machines	collector
Match jobs and machines	negotiator
Manage one machine	startd (“start-dee”)
Manage one job (on submitter)	shadow
Manage one job (on machine)	starter

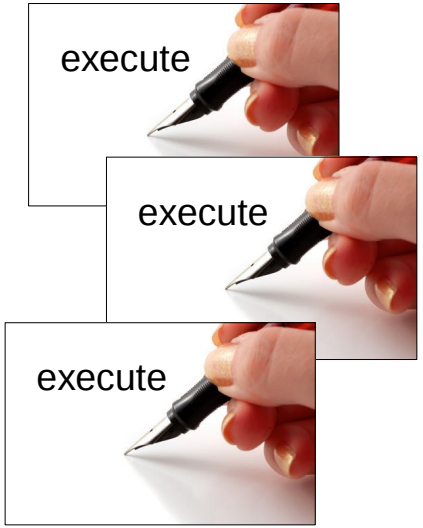
Main Parts of HTCondor

Function	HTCondor Name	#
Track waiting/running jobs	schedd (“sked-dee”)	1+
Track available machines	collector	1
Match jobs and machines	negotiator	1
Manage one machine	startd (“start-dee”)	per machine
Manage one job (on submitter)	shadow	per job running
Manage one job (on machine)	starter	per job running

HTCondor Matchmaking

- HTCondor's central manager matches jobs with computers
- Information about each job and computer in a "ClassAd"
- ClassAds have the format:
  - `AttributeName = value`
  - `Value` can be a boolean, number, or string

central manager
(negotiator + collector)



Job ClassAd

Submit file

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

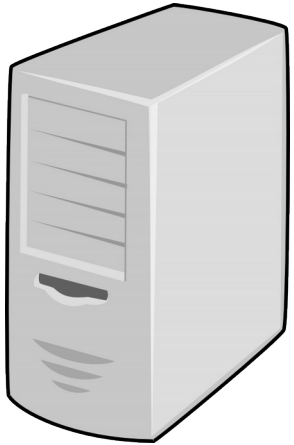
+

Default HTCondor configuration

=

```
RequestCpus = 1
Err = "job.err"
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd =
"/home/alice/tests/htcondor_week/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week”
NumJobStarts = 0
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job"
Out = "job.out"
UserLog = "/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...
```

Machine ClassAd



+

Default HTCondor configuration

=

```
HasFileTransfer = true
DynamicSlot = true
TotalSlotDisk = 4300218.0
TargetType = "Job"
TotalSlotMemory = 2048
Mips = 17902
Memory = 2048
UtsnameSysname = "Linux"
MAX_PREEMPT = ( 3600 * ( 72 - 68 *
( WantGlidein =?= true ) ) )
Requirements = ( START ) &&
( IsValidCheckpointPlatform ) &&
( WithinResourceLimits )
OpSysMajorVer = 6
TotalMemory = 9889
HasGluster = true
OpSysName = "SL"
HasDocker = true

...
```

HTCondor Matchmaking

- On a regular basis, the central manager reviews job and Machine ClassAds and matches jobs to computers

JOB SUBMISSION, REVISITED

Waiting for matchmaking

- Back to our compare_states example:
  - `condor_submit job.submit  # job is submitted to the queue!`
  - `condor_q # let's check the status!`

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat            11/12 09:30   0+00:00:00 I  0   0.0  compare_states

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

- Matchmaking does not happen instantaneously! We need to wait for the periodic matchmaking cycle (on the order of 5 minutes)

Job Idle

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat            11/12 09:30   0+00:00:00 I  0   0.0  compare_states

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

Submit Node

```
(submit_dir)/
     job.submit
     compare_states
     wi.dat
     us.dat
    job.log
    job.out
    job.err
```

Match made! Job Starts

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat            11/12 09:30   0+00:00:00 <  0   0.0  compare_states

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
  job.log
  job.out
  job.err
```

Transfer:

```
compare_states
    wi.dat
    us.dat
```

Execute Node

```
(execute_dir)/
```

Job Running

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat            11/12 09:30   0+00:00:00 R  0   0.0  compare_states

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
  job.log
  job.out
  job.err
```

Execute Node

```
(execute_dir)/
compare_states
    wi.dat
    us.dat
    stderr
    stdout
    wi.dat.out
```

Job Completes

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat            11/12 09:30   0+00:00:00 >  0   0.0  compare_states

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
  job.log
  job.out
  job.err
```

stderr
stdout
wi.dat.out

Execute Node

```
(execute_dir)/
compare_states
    wi.dat
    us.dat
    stderr
    stdout
    wi.dat.out
```

Job Completes

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD


0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
   job.log
   job.out
   job.err
   wi.dat.out
```

HTCondor Priorities

- Job priority
  - Set per job by the user (owner)
  - Relative to that user's other jobs
  - Set in submit file or changed later with `condor_prio`
  - Higher number means run sooner
- User priority
  - Computed based on past usage
  - Determines user's "fair share" percentage of slots
  - Lower number means run sooner (0.5 is minimum)
- Preemption
  - Low priority jobs stopped for high priority ones (stopped jobs go back into the regular queue)
  - Governed by fair-share algorithm and pool policy
  - Not enabled on all pools

SUBMIT FILES

File Access in HTCondor

- Option 1: Shared filesystem
  - Easy to use (jobs just access files)
  - But, must exist and be ready to handle load

```
should_transfer_files = NO
```

- Option 2: HTCondor transfers files for you
  - Must name all input files (except executable)
  - May name output files; defaults to all new/changed

```
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = a.txt, b.tgz
```

Resource requests

```
request_cpus = ClassAdExpression
request_disk = ClassAdExpression
request_memory = ClassAdExpression
```

- Be a good user! Ask for minimum resources of execute machine
- **Check job log for actual usage!!!**
- May be dynamically allocated (very advanced!)

```
request_disk = 2000000          # in KB by default
request_disk = 2GB              # KB, MB, GB, TB
request_memory = 2000           # in MB by default
request_memory = 2GB            # KB, MB, GB, TB
```

Resource requests -- Log File

```
000 (128.000.000) 05/09 11:09:08 Job submitted from host: <128.104.101.92&sock=6423_b881_3>
...
001 (128.000.000) 05/09 11:10:46 Job executing on host:
<128.104.101.128:9618&sock=5053_3126_3>
...
006 (128.000.000) 05/09 11:10:54 Image size of job updated: 220
      1  -  MemoryUsage of job (MB)
      220  -  ResidentSetSize of job (KB)
...
005 (128.000.000) 05/09 11:12:48 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      33  -  Run Bytes Received By Job
      0  -  Total Bytes Sent By Job
      33  -  Total Bytes Received By Job
      Partitionable Resources :    Usage  Request Allocated
         Cpus                 :                1         1
         Disk (KB)            :   612015    20480  17203728
         Memory (MB)          :      312     3000      3000
```

Resource requests -- Log File

- This user needs to update their resource requests!
  - Using more diskspace than they request, which can cause their jobs to go on hold
    - More on this tomorrow!
  - Requesting more memory than they use
    - Their jobs can take longer to match (there may be old 1GB RAM machines sitting around ready for them!)
    - Other users might *actually need* all that memory and are waiting in line.

```
Partitionable Resources :    Usage  Request Allocated
   Cpus                 :                1         1
   Disk (KB)            :   612015    20480  17203728
   Memory (MB)          :      312     3000      3000
```

Email notifications

```
notification = Always|Complete|Error|Never
```

- When to send email:
  - Always: job checkpoints or completes
  - Complete: job completes (default)
  - Error: job completes with error
  - Never: do not send email

```
notify_user = email
```

- Where to send email
  - Defaults to user@submit-machine

Requirements and Rank

```
requirements = ClassAdExpression
```

- Expression must evaluate to true to match slot
- HTCondor adds defaults!
- See HTCondor Manual (esp. 2.5.2 and 4.1) for more

```
rank = ClassAdExpression
```

- Ranks matching slots in order by preference
- Must evaluate to a floatingpoint number, higher is better
  - False becomes 0.0, True becomes 1.0
  - Undefined or error values become 0.0
- Writing rank expressions is an art

Arbitrary Attributes

```
+AttributeName = value
```

- Adds arbitrary attribute(s) to a job's ClassAd
- Useful in (at least) 2 cases:
  - Affect matchmaking with special attributes
  - Report on jobs with specific attribute value
- Experiment with reporting during exercises!

SUBMITTING MULTIPLE JOBS

Many Jobs, One Submit File

- HTCondor offers many ways to submit multiple jobs from one submit file, allowing you to:
  - Analyze multiple data files
  - Test many parameter or input combinations
  - Modify arguments
- …without having to
  - Start each job individually
  - Create submit files for each job

Multiple numbered input files

job.submit

```
executable = analyze.exe
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.out
error = job.err

queue
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- Goal: create 3 jobs that each analyze a different input file.

Multiple numbered input files

job.submit

```
executable = analyze.exe
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.out
error = job.err

queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- Generates 3 jobs, but doesn't change inputs and overwrites outputs
- So how can we specify different values to each job?

Manual Approach (Not recommnded!)

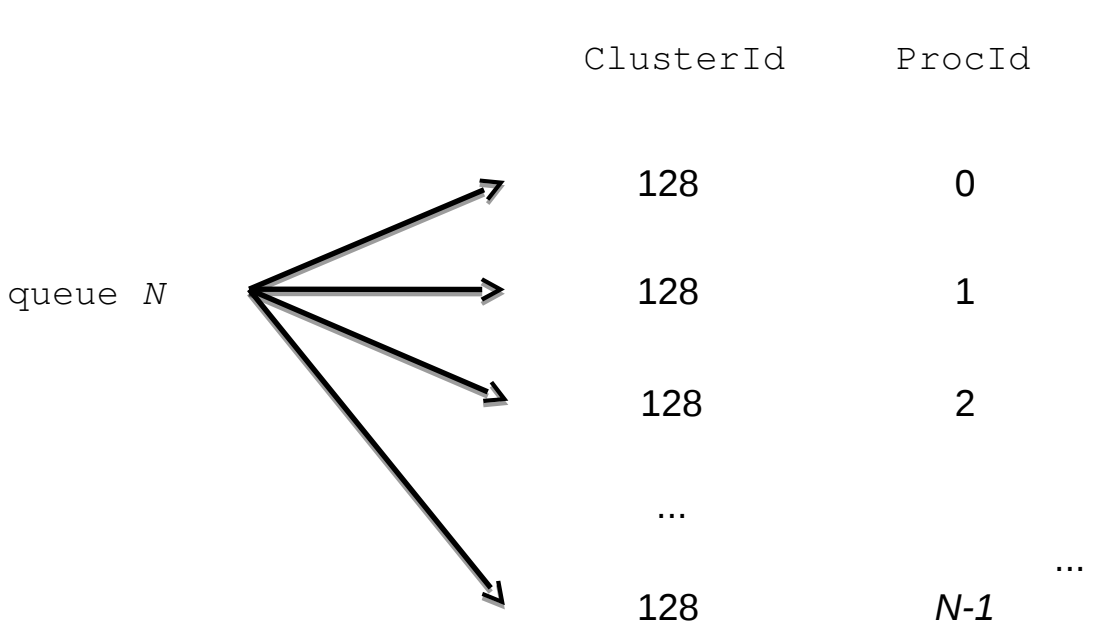
job.submit

```
executable = analyze.exe
log = job.log

arguments = file0.in file0.out
transfer_input_files = file0.in
output = job0.out
error = job0.err
queue 1

arguments = file1.in file1.out
transfer_input_files = file1.in
output = job1.out
error = job1.err
queue 1

...
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

Automatic Variables



- Each job's `ClusterId` and `ProcId` numbers are saved as job attributes
- They can be accessed inside the submit file using:
  - $(`Cluster`)*
  - $(`Proc`)*

* : $(ClusterId) and $(ProcId) are also used

Multiple numbered input files

job.submit

```
executable = analyze.exe
arguments = file$(Process).in file$(Process).out
transfer_input_files = file$(Process).in

log = job_$(Cluster).log
output = job_$(Process).out
error = job_$(Process).err

queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- $(Cluster) and $(Process) allow us to provide unique values to jobs!

Separating Jobs with initialdir

- `Initialdir` changes the submission directory for each job, allowing each job to “live” in separate directories
- Uses the same name for all input/output files
- Useful for jobs with lots of output files

Separating jobs with initialdir

```
(submit_dir)/
job.submit
analyze.exe
job0/
  file.in
  job.log
  job.err
  file.out
job1/
  file.in
  job.log
  job.err
  file.out
job2/
  file.in
  job.log
  job.err
  file.out
```

job.submit

```
executable = analyze.exe
initialdir = job$(ProcId)
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
error = job.err

queue 3
```

Executable should be in the directory with the submit file, *not* in the individual job directories

Many jobs per submit file

- Back to our compare_states example…
- What if we had data for each state? We could do 50 submit files (or 50 `queue 1` statements) ...

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out
…
```

```
executable = compare_states
arguments = ca.dat us.dat ca.dat.out
…
```

```
executable = compare_states
arguments = mo.dat us.dat mo.dat.out
…
```

```
executable = compare_states
arguments = md.dat us.dat md.dat.out
…
```

```
executable = compare_states
arguments = wv.dat us.dat wv.dat.out
…
```

```
executable = compare_states
arguments = fl.dat us.dat fl.dat.out
…
```

Many jobs per submit file

- Back to our compare_states example…
- What if we had data for each state? We could do 50 submit files (or 50 `queue 1` statements) ...

```
executable = compare_states
arguments = wa.dat us.dat wa.dat.out
…
```

```
executable = compare_states
arguments = co.dat us.dat co.dat.out
…
```

```
executable = compare_states
arguments = mi.dat us.dat mi.dat.out
…
```

```
executable = compare_states
arguments = nv.dat us.dat nv.dat.out
…
```

```
executable = compare_states
arguments = sd.dat us.dat sd.dat.out
…
```

```
executable = compare_states
arguments = mn.dat us.dat mn.dat.out
…
```

Many jobs per submit file

- Back to our compare_states example…
- What if we had data for each state? We could do 50 submit files (or 50 `queue 1` statements) ...

```
executable = compare_states
arguments = vt.dat us.dat vt.dat.out
```

```
executable = compare_states
arguments = al.dat us.dat al.dat.out
```

```
executable = compare_states
arguments = tx.dat us.dat tx.dat.out
```

```
executable = compare_states
arguments = ut.dat us.dat ut.dat.out
```

```
executable = compare_states
arguments = ak.dat us.dat ak.dat.out
```

```
executable = compare_states
arguments = tn.dat us.dat tn.dat.out
```

Many jobs per submit file

- Back to our compare_states example…
- What if we had data for each state? We could do 50 submit files (or 50 `queue 1` statements) ...
- Or we could organize our data to fit the $(Process) or initialdir approaches…
- Or we could use HTCondor's `queue` language to submit jobs smartly!

Submitting Multiple Jobs

- Replace job-level files…

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out
…
transfer_input_files = us.dat, wi.dat
```

- …with a variable:

```
executable = compare_states
arguments = $(state).dat us.dat $(state).dat.out
…
transfer_input_files = us.dat, $(state).dat

queue ???
```

- ...But how do we define these variables in our queue statement?

Submitting Multiple Jobs – Queue Statements

multiple "queue" statements

```
state = wi.dat
queue 1
state = ca.dat
queue 1
state = mo.dat
queue 1
```

matching ... pattern

```
queue state matching *.dat
```

in ... list

```
queue state in (wi.dat ca.dat co.dat)
```

from ... file

```
queue state from state_list.txt
```

```
wi.dat
ca.dat
mo.dat
```

state_list.txt

Submitting Multiple Jobs – Queue Statements

multiple queue statements	Not recommended. Can be useful when submitting job batches where a single (non-file/argument) characteristic is changing
matching .. pattern	Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories Requires good naming conventions
in .. list	Supports multiple variables, all information contained in a single file, reproducible Harder to automate submit file creation
from .. file	Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible Additional file needed

Using Multiple Variables

- Both the "`from`" and "`in`" syntax support multiple variables from a list.

job.submit

```
executable = compare_states
arguments = -y $(option) -i $(file)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = $(file)

queue file,option from job_list.txt
```

job_list.txt

```
wi.dat, 2010
wi.dat, 2015
ca.dat, 2010
ca.dat, 2015
mo.dat, 2010
mo.dat, 2015
```

Submitting Multiple Jobs -- Advanced

- Variables
  - $(Step), $(Item), $(Row), $(ItemIndex) provide additional handles when queuing multiple jobs
- Function Macros
  - E.g. $Fn(state) becomes “wi” when state is “wi.dat”
- Python-style slicing
  - `queue state matching [:1] *.dat`
    - Only submits one job – great for testing!
- Lots of additional (and powerful!) features
  - Experiment if you finish exercises early!
  - See documentation in Section 2.5

YOUR TURN!

Exercises!

- Ask questions!
- Lots of instructors around

- Coming up:
  - Now-12:15 Hands-on Exercises
  - 12:15 – 1:15 Lunch
  - 1:15 – 5:30 Afternoon sessions

BACKUP SLIDES

HTCONDOR COMMANDS

List jobs: `condor_q`

- Select jobs: by user (defaults to you), cluster, job ID
- Format output as you like
- View full ClassAd(s), typically 80-90 attributes
  - Most useful when limited to a single job ID)
- Ask HTCondor why a job is not running
  - May not explain everything, but can help
  - Remember: Negotiation happens periodically
- Explore `condor_q` options in coming exercises

List slots: `condor_status`

- Select slots: available, host, specific slot
- Select slots by ClassAd expression
  - E.g. slots with SL6 and ≥ 10 GB memory
- Format output as you like
- View full ClassAd(s), typically 120-250 attributes
  - Most useful when limited to a single slot
- Explore `condor_status` options in exercises

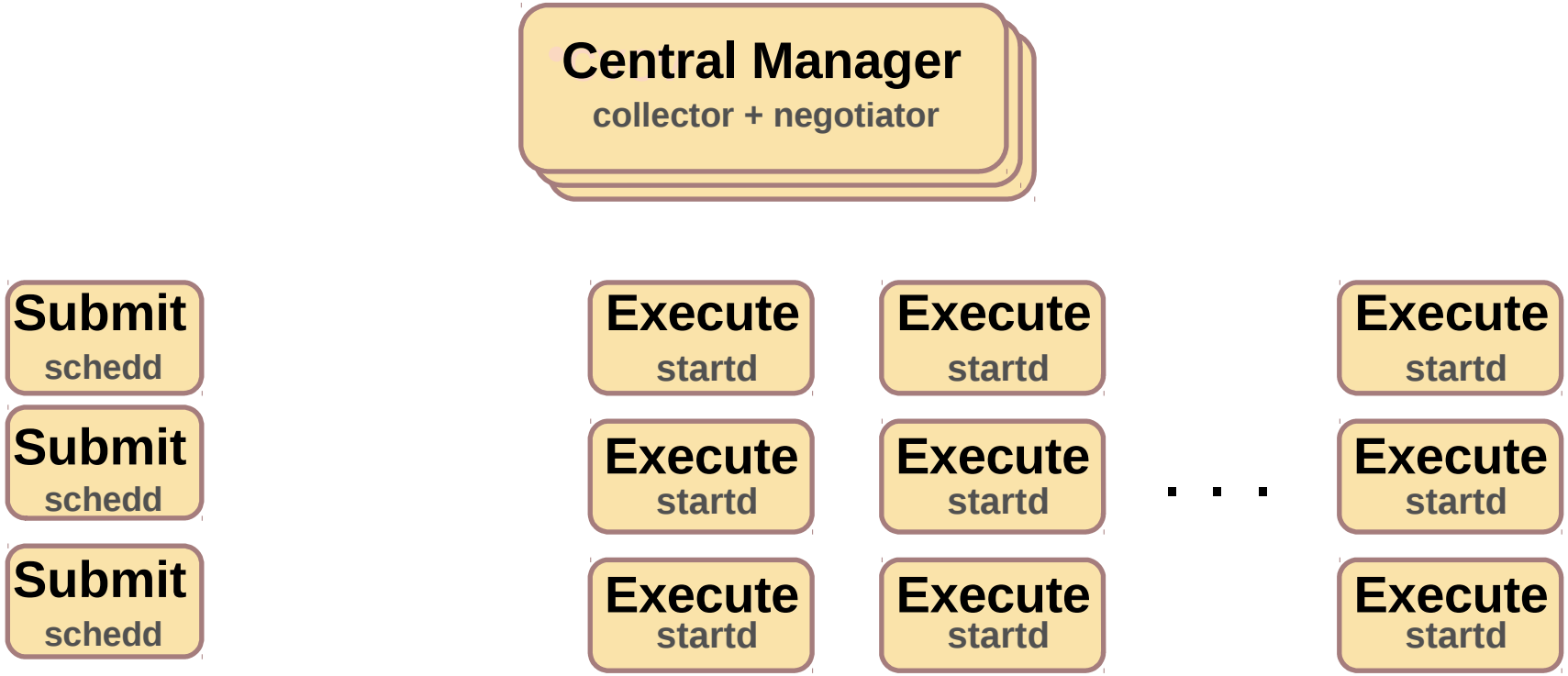
HTCondor Universes

- Different combinations of configurations and features are bundled as ***universes***

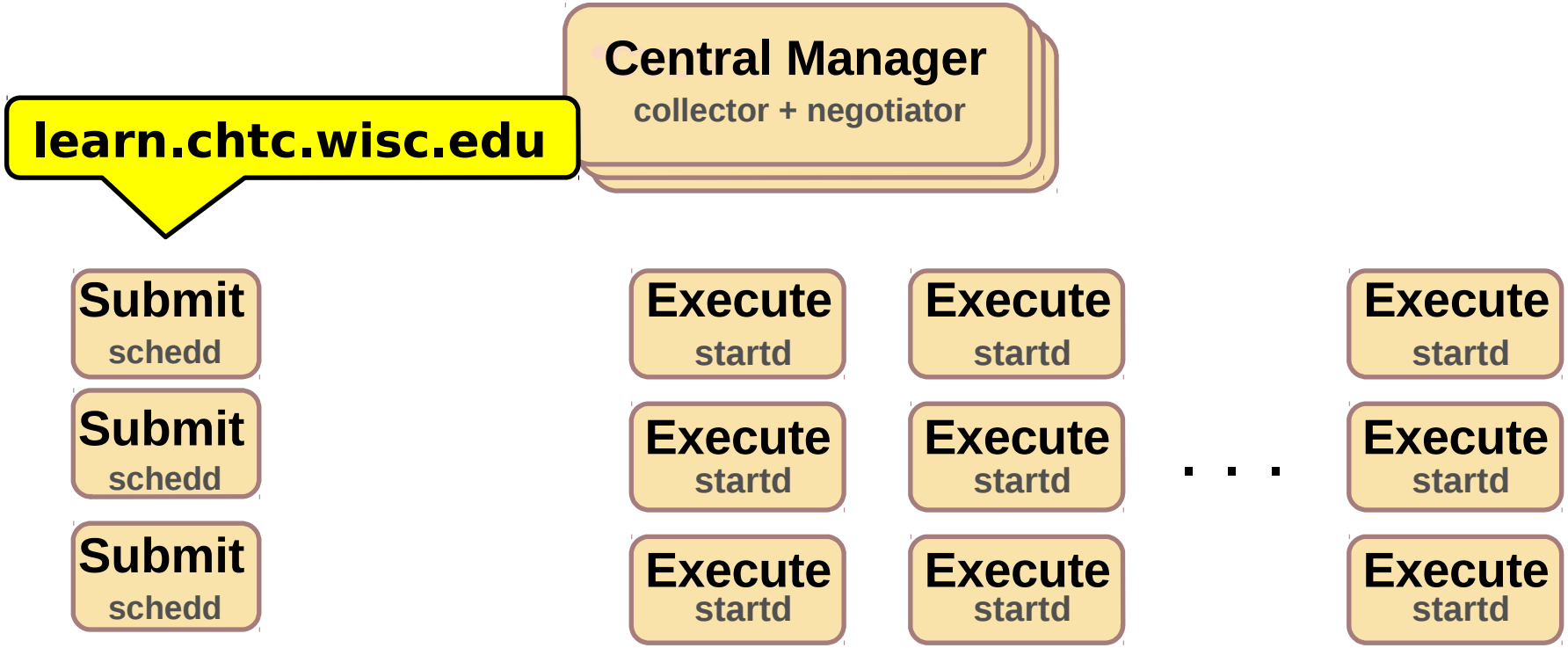
`vanilla`	A “normal” job; default, fine for today
`standard`	Supports checkpointing and remote I/O
`java`	Special support for Java programs
`parallel`	Supports parallel jobs (such as MPI)
`grid`	Submits to remote system (more tomorrow)

…and many others

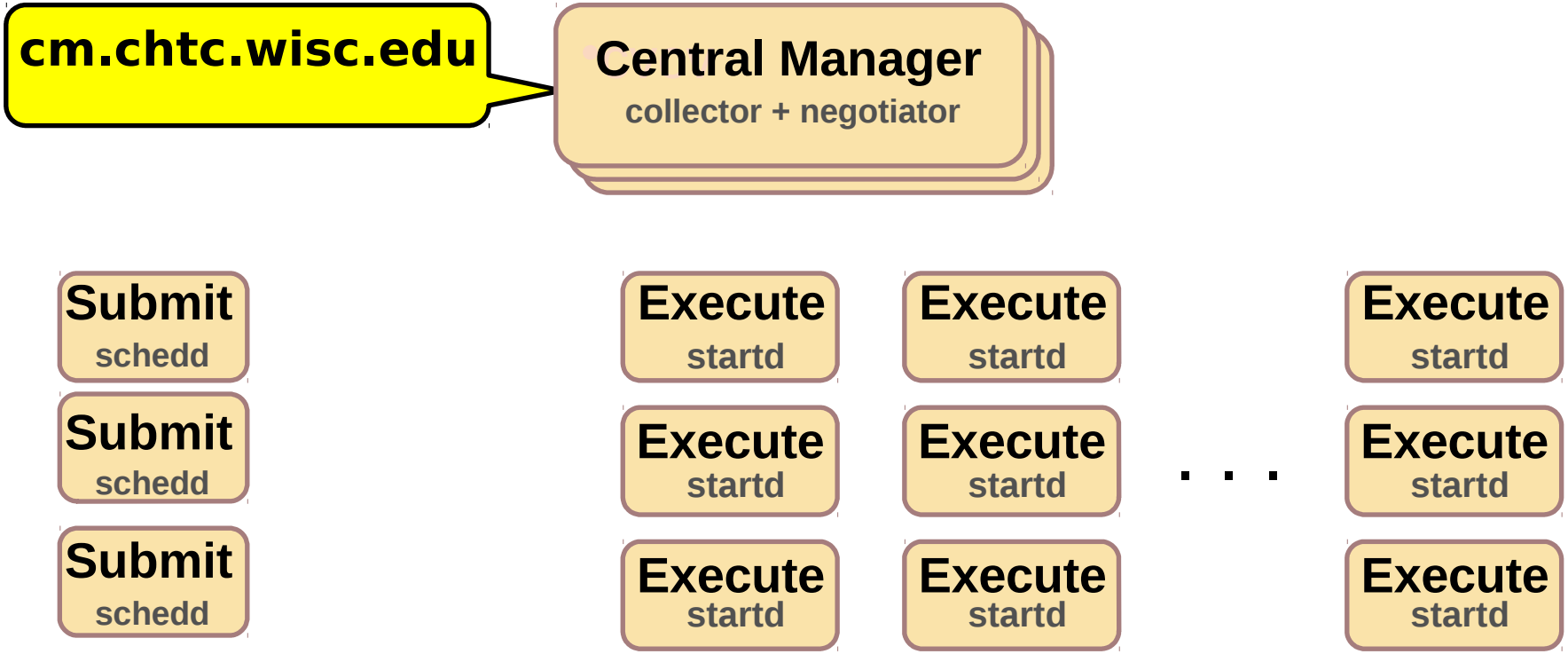
Typical Architecture



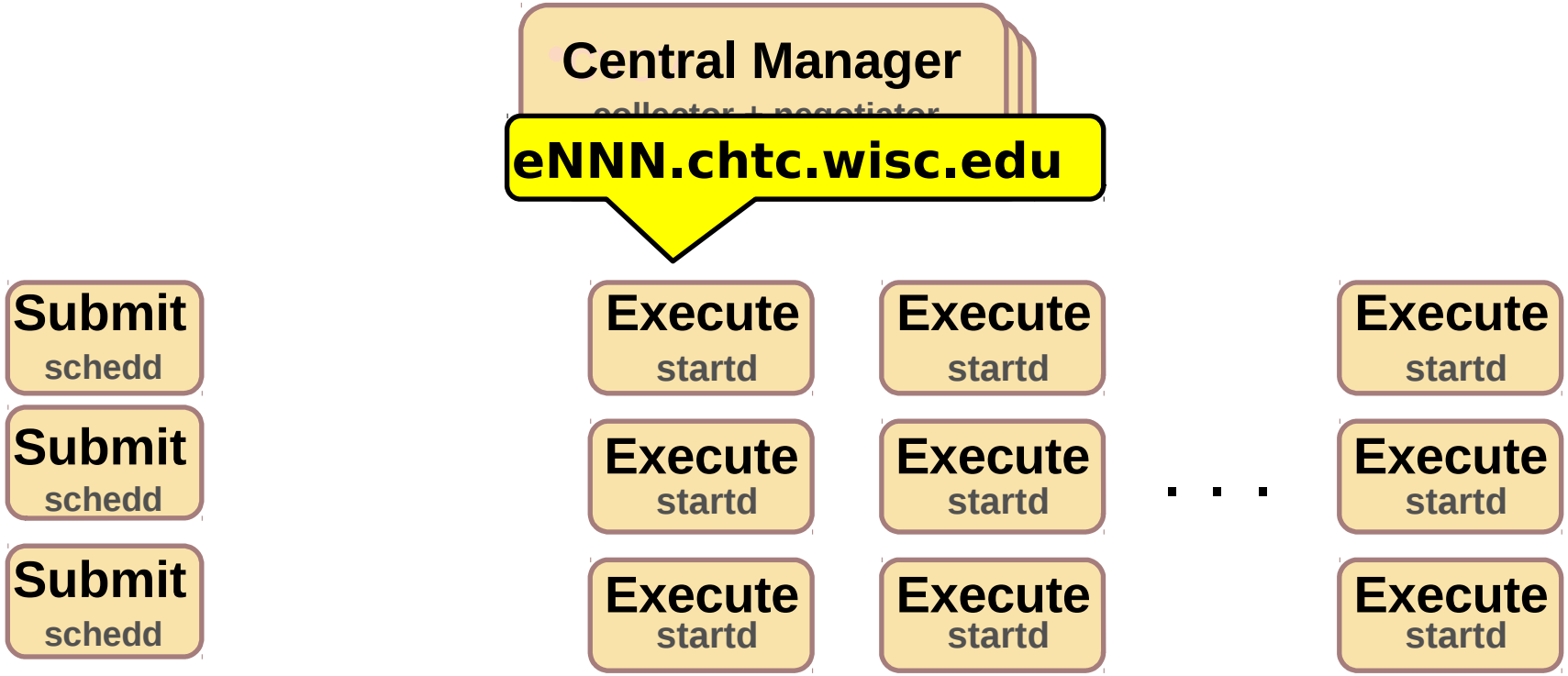
Typical Architecture



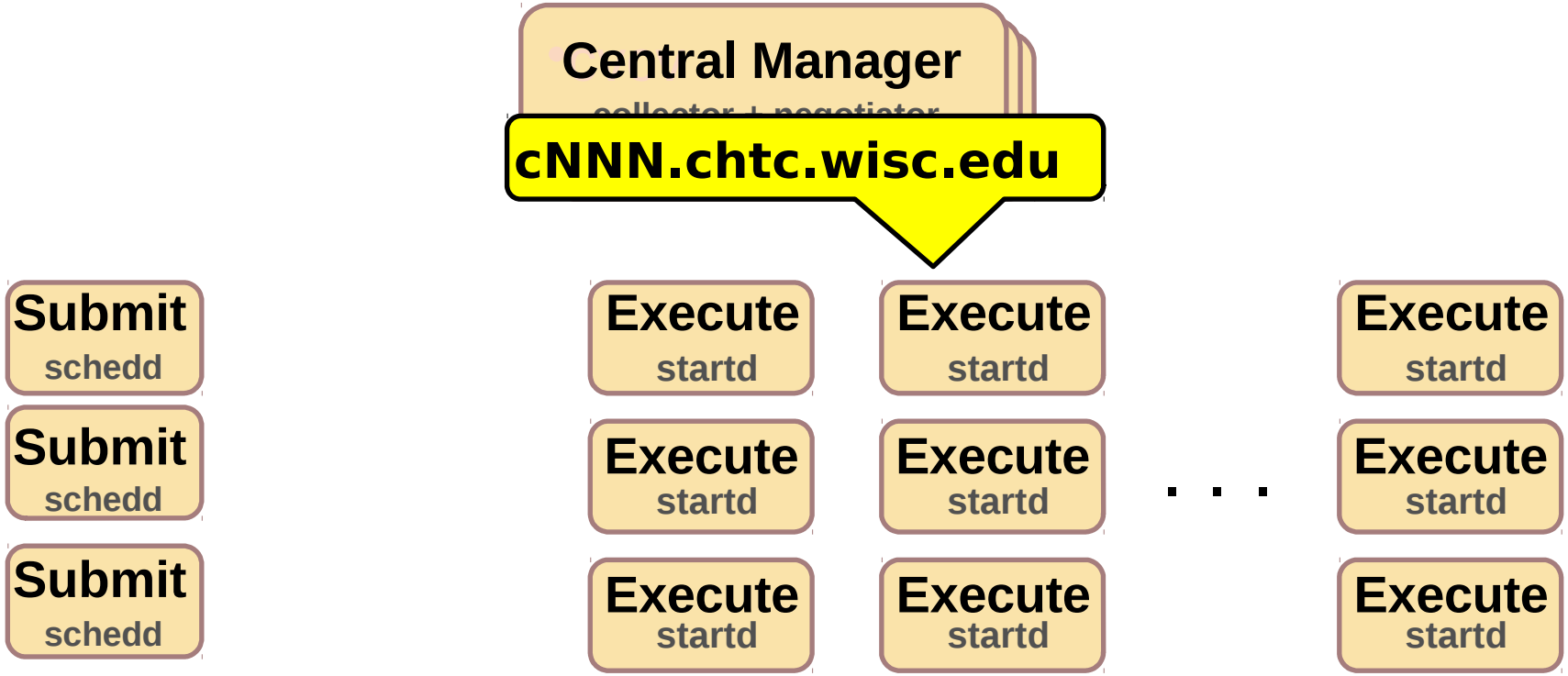
Typical Architecture



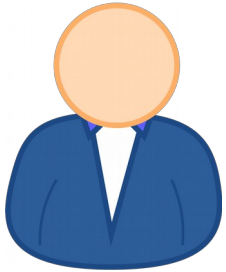
Typical Architecture

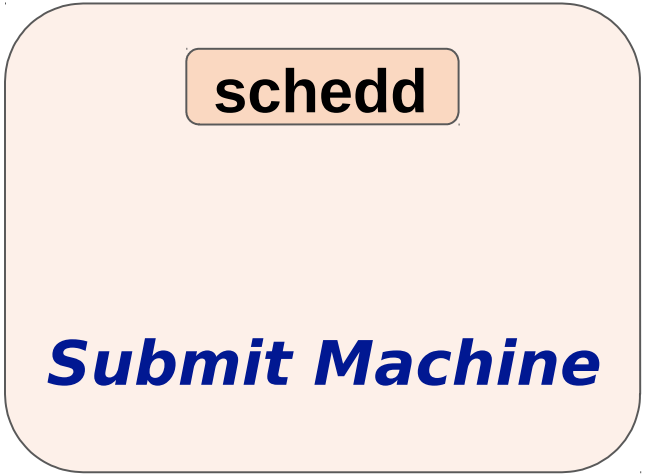


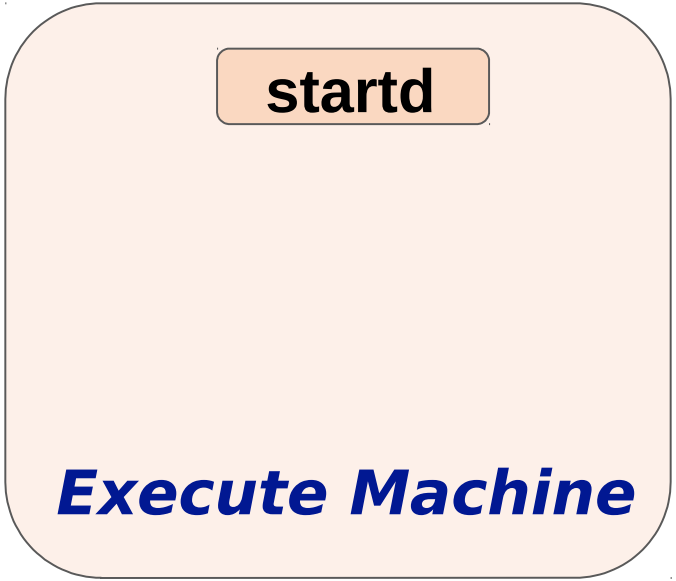
Typical Architecture



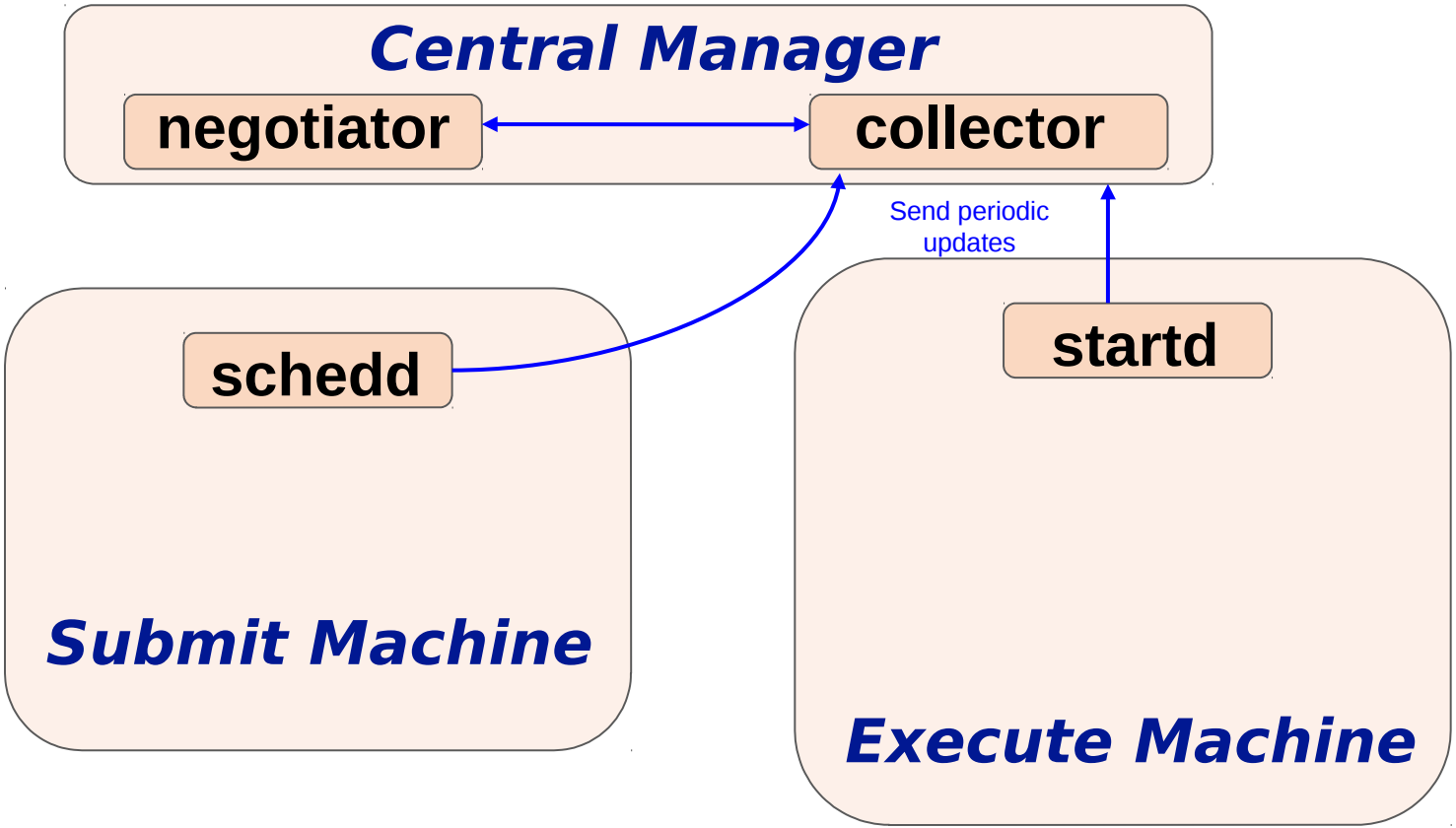
The Life of an HTCondor Job

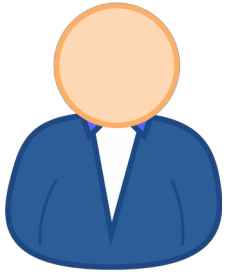




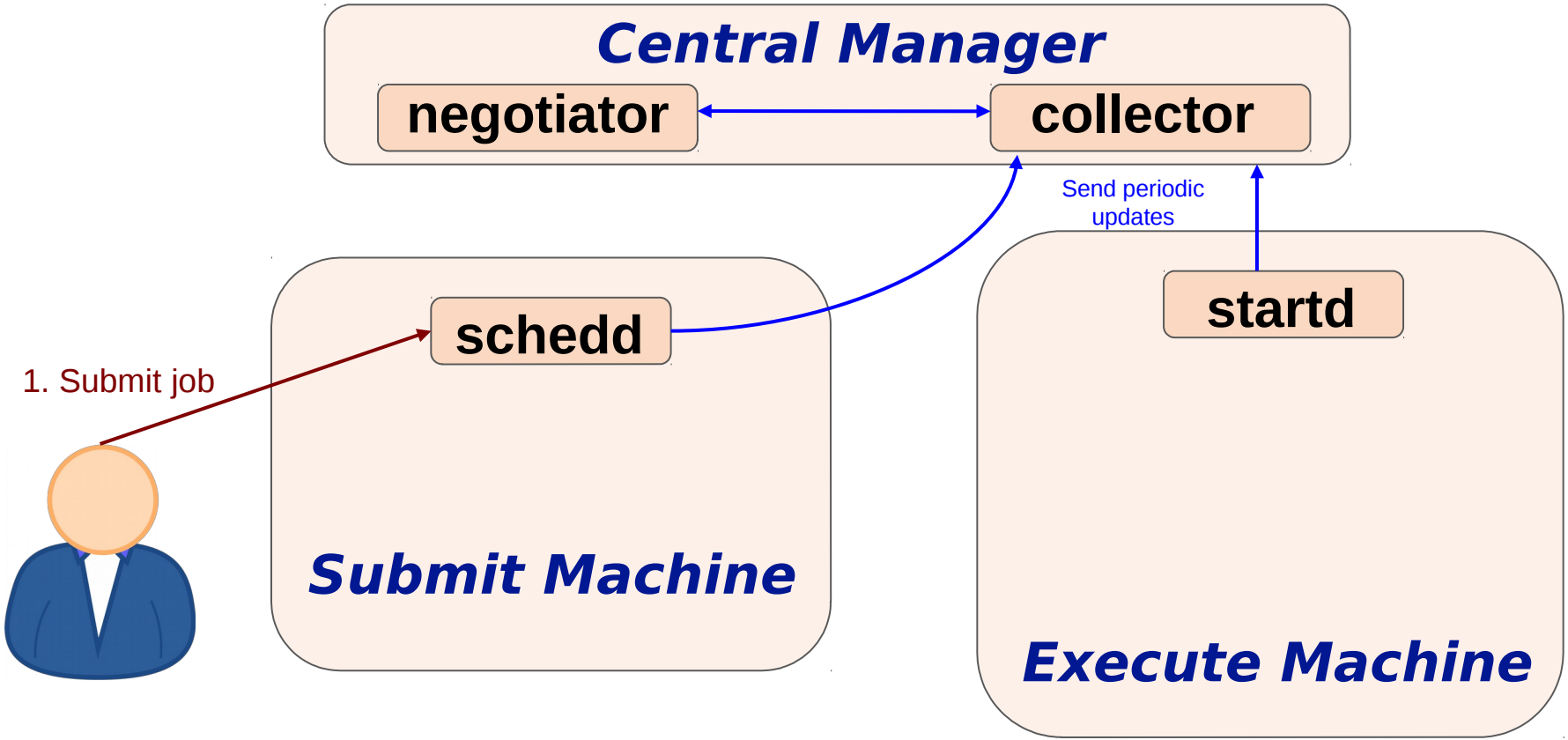


The Life of an HTCondor Job

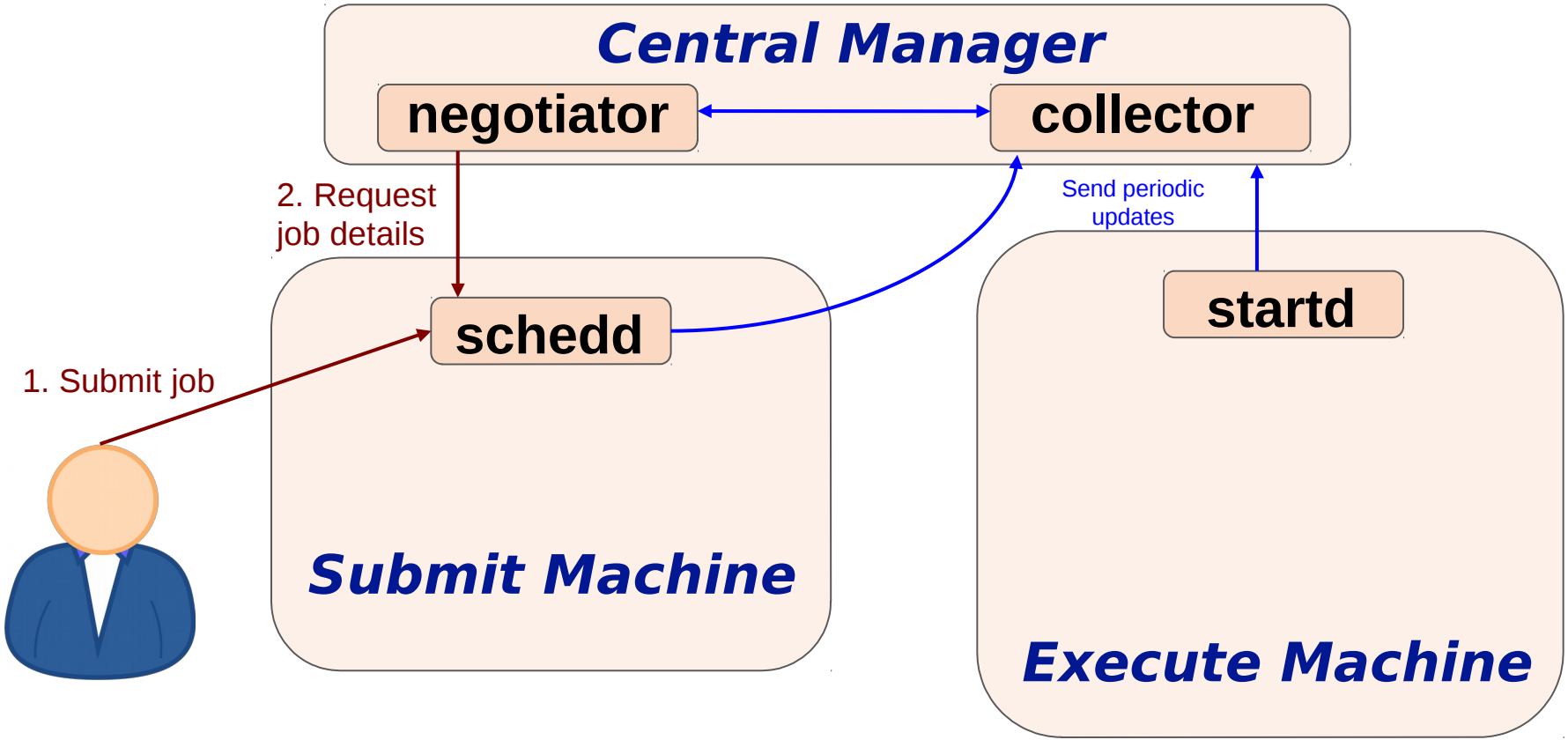




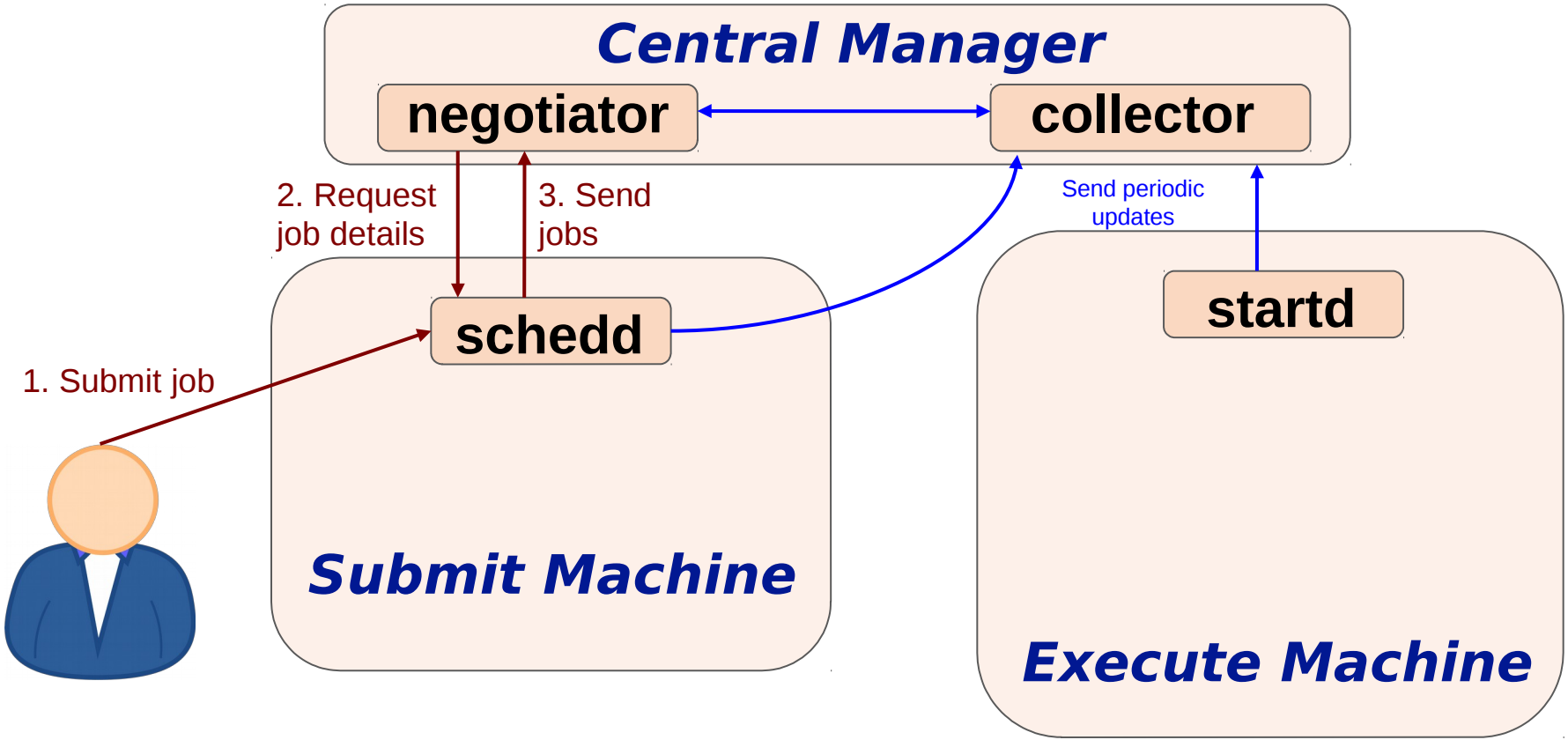
The Life of an HTCondor Job



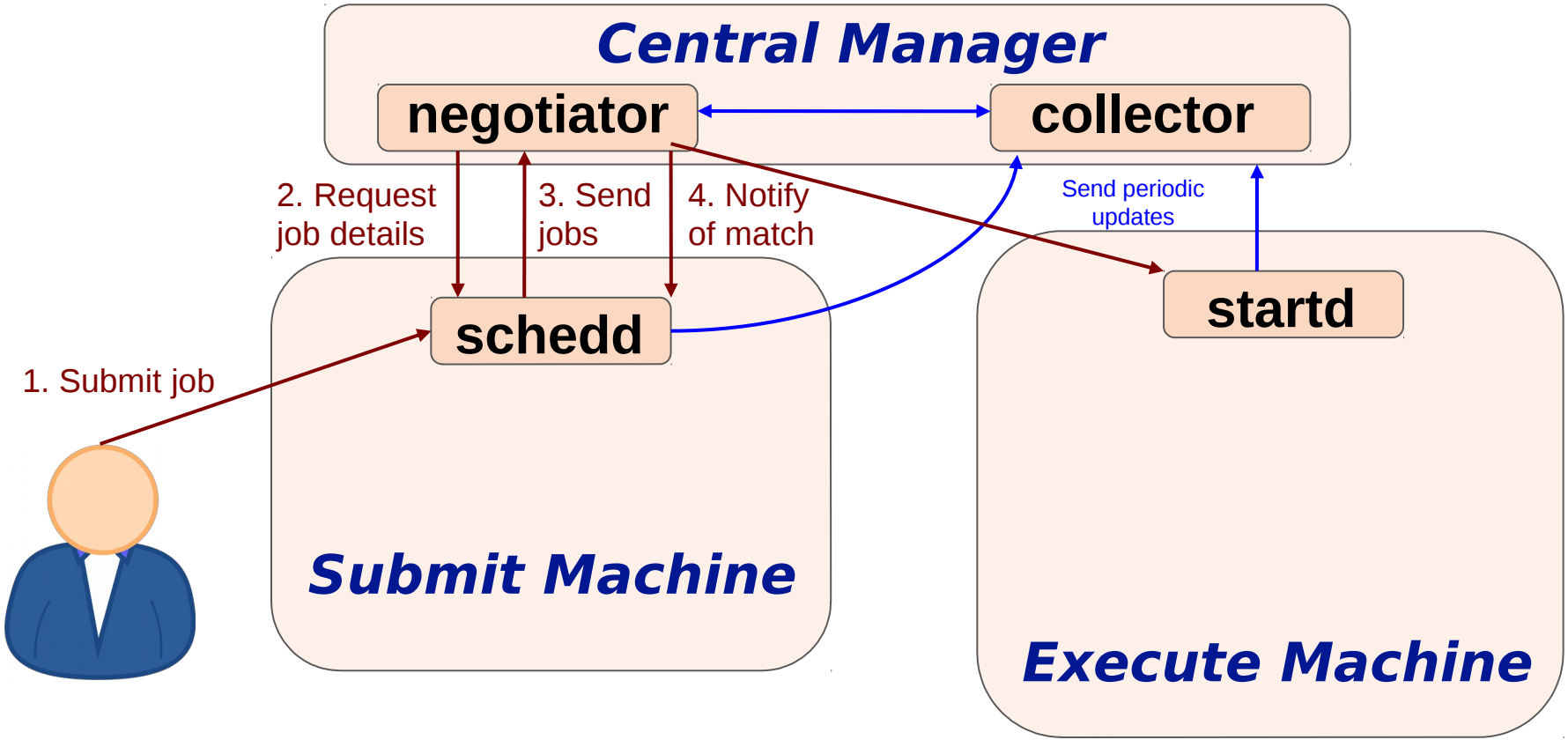
The Life of an HTCondor Job



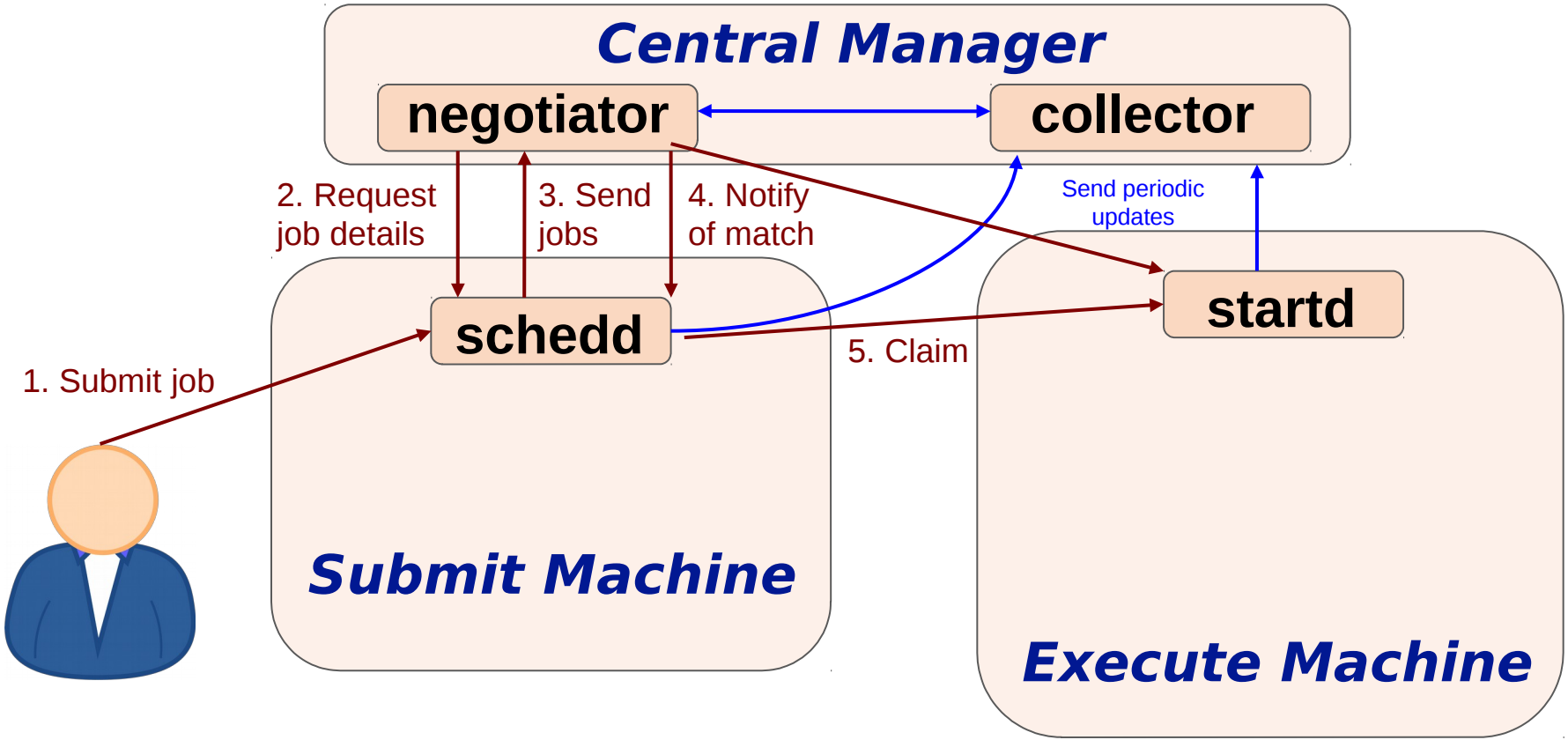
The Life of an HTCondor Job



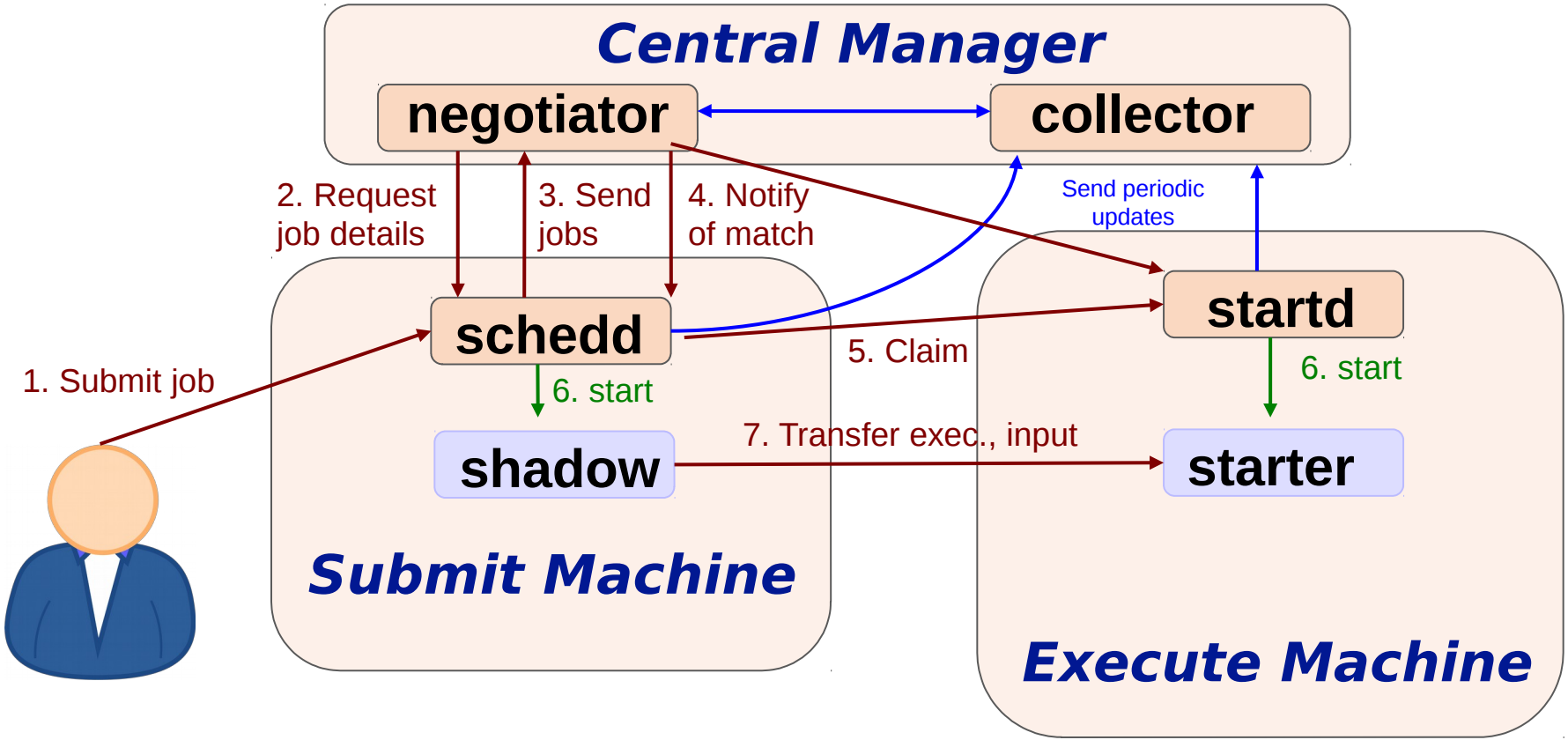
The Life of an HTCondor Job



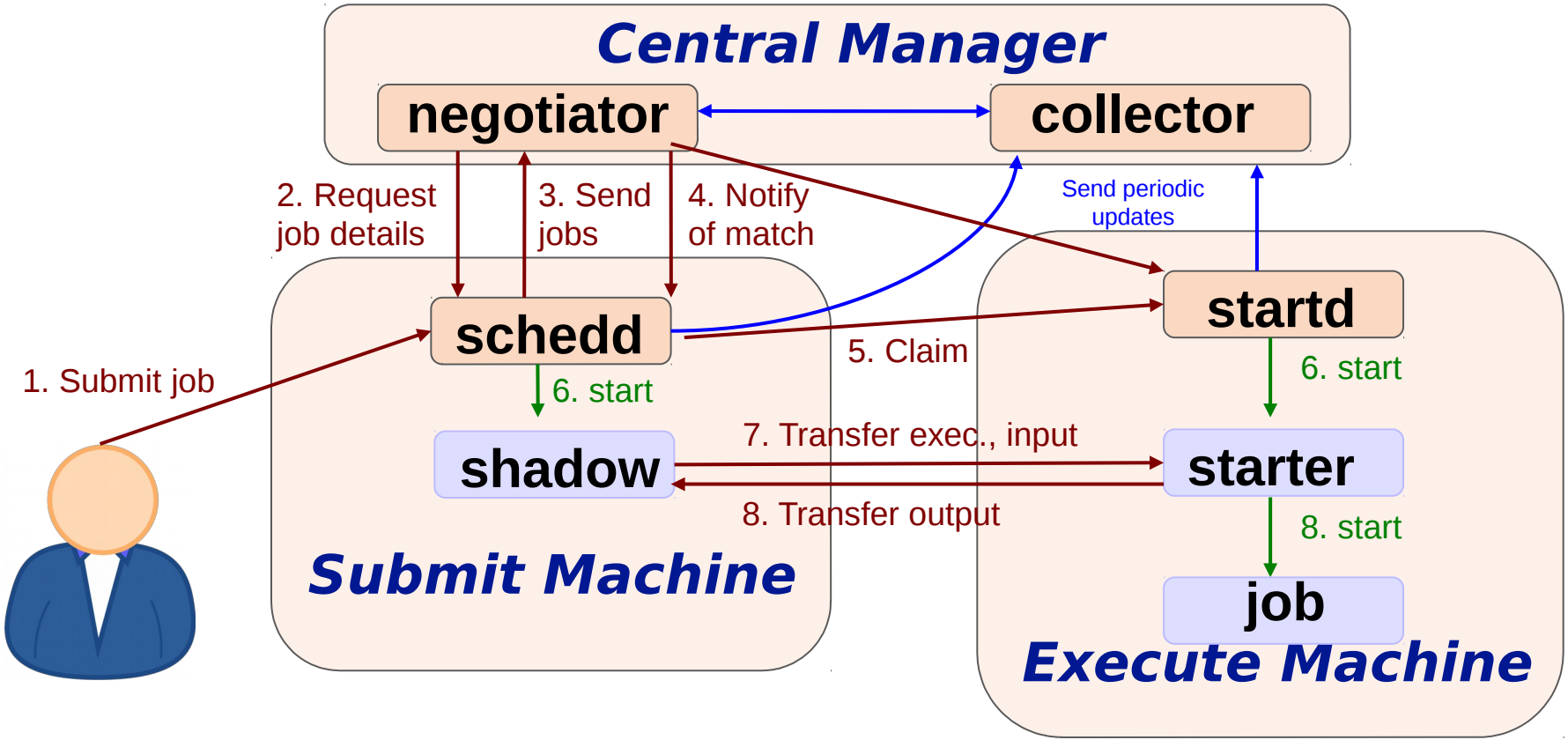
The Life of an HTCondor Job



The Life of an HTCondor Job



The Life of an HTCondor Job



Matchmaking Algorithm (sort of)

- A. Gather lists of machines and waiting jobs
- B. For each user:
  - 1. Compute maximum # of slots to allocate to user
    - This is the user's "fair share", a % of whole pool
  - 2. For each job (until maximum matches reached):
    - A. Find all machines that are acceptable (i.e., machine ***and*** job requrements are met)
    - B. If there are no acceptable machines, skip to next job
    - C. Sort acceptable machines by job preferences
    - D. Pick the best one
    - E. Record match of job and slot

ClassAds

- In HTCondor, information about machines and jobs (and more) are represented by ClassAds
- You do not write ClassAds (much), but reading them may help understanding and debugging
- ClassAds can represent persistent fact, current state, preferences, requirements, …
- HTCondor uses a core of predefined attributes, but users can add other, new attributes, which can be used for matchmaking, reporting, etc.

Sample ClassAd Attributes

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Sample ClassAd Attributes

String

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Sample ClassAd Attributes

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Number

Sample ClassAd Attributes

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Operations/ expressions

Sample ClassAd Attributes

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Boolean

Sample ClassAd Attributes

```
MyType = "Job"
TargetType = "Machine"
ClusterId = 14
Owner = "cat"
  Cmd = "/.../test-job.py"
Requirements = (Arch == "X86_64") && (OpSys ==
"LINUX")

Rank = 0.0
  In = "/dev/null"
   UserLog = "/.../test-job.log"
   Out = "test-job.out"
  Err = "test-job.err"
  NiceUser = false
  ShoeSize = 10
```

Arbitrary