Creating Custom Work Queue Applications

Nicholas Hazekamp

Makeflow vs. Work Queue

- Makeflow
  - Directed Acyclic Graph programming model.
  - Static structure known in advance.
  - All communication through files on disk.
- Work Queue
  - Submit-Wait programming model.
  - Dynamic structure decided at run-time.
  - Communicate through buffers or files.
  - More detailed knowledge of how tasks ran.

Work Queue API

```
#include “work_queue.h”

queue = work_queue_create();

while( not done ) {
      while (more work ready) {
            task = work_queue_task_create();
             // add some details to the task
             work_queue_submit(queue, task);
      }

      task = work_queue_wait(queue);
      // process the completed task
}
```

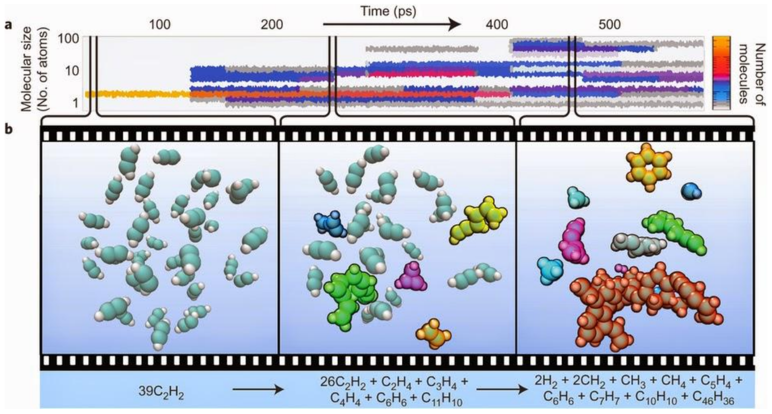
http://ccl.cse.nd.edu/software/workqueue

CCTools

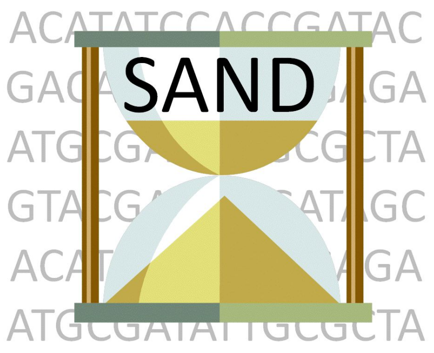
UNIVERSITY OF NOTRE DAME

Work Queue Applications

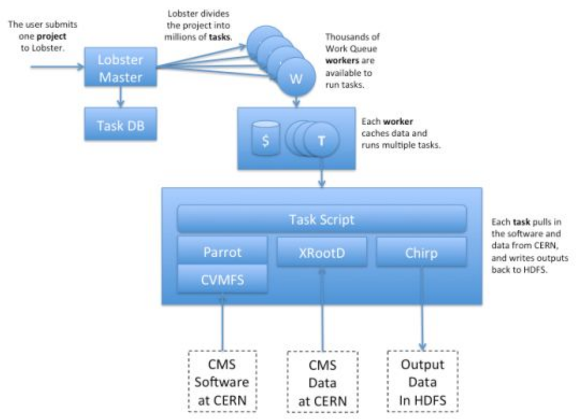
Nanoreactor MD Simulations



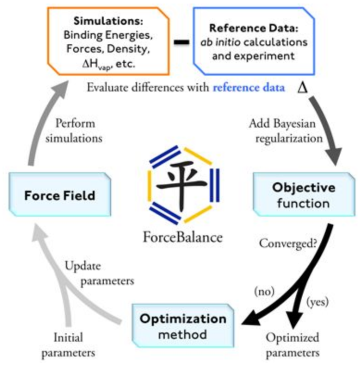
Scalable Assembler at Notre Dame



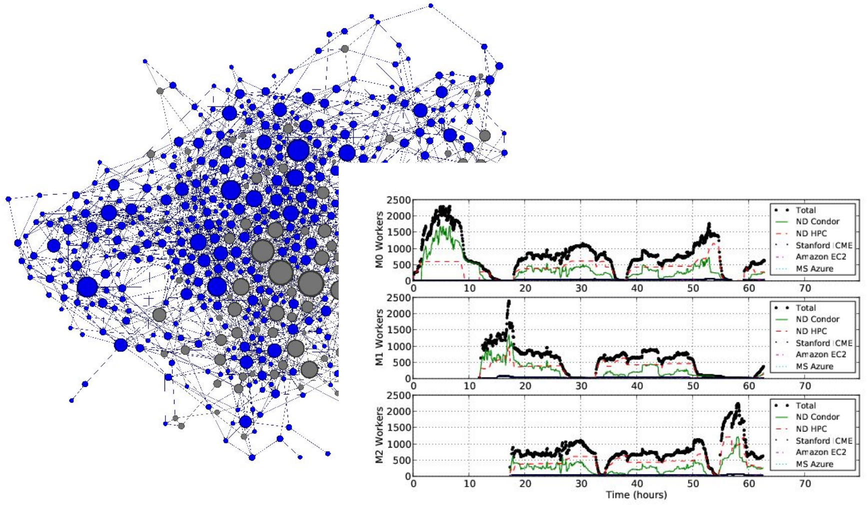
Lobster HEP


The user submits one project to Lobster.
Lobster Master
Task DB
Lobster divides the project into millions of tasks.
W
Thousands of Work Queue workers are available to run tasks.
$
T
Each worker caches data and runs multiple tasks.
Task Script
Parrot
CVMFS
XRootD
Chirp
Each task pulls in the software and data from CERN, and writes outputs back to HDFS.
CMS Software at CERN
CMS Data at CERN
Output Data In HDFS

ForceBalance


Simulations: Binding Energies, Forces, Density, ΔHvap, etc.
Reference Data: ab initio calculations and experiment
Evaluate differences with reference data
Δ
Perform simulations
Add Bayesian regularization
Force Field
ForceBalance
Objective function
Converged?
Update parameters
(no)
(yes)
Initial parameters
Optimization method
Optimized parameters

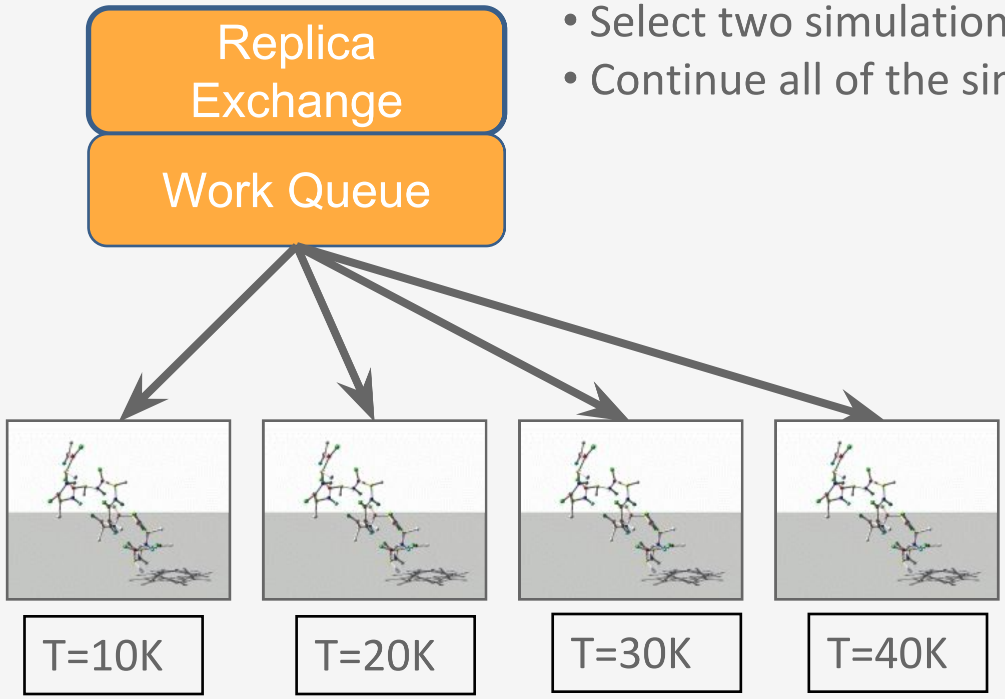
Adaptive Weighted Ensemble

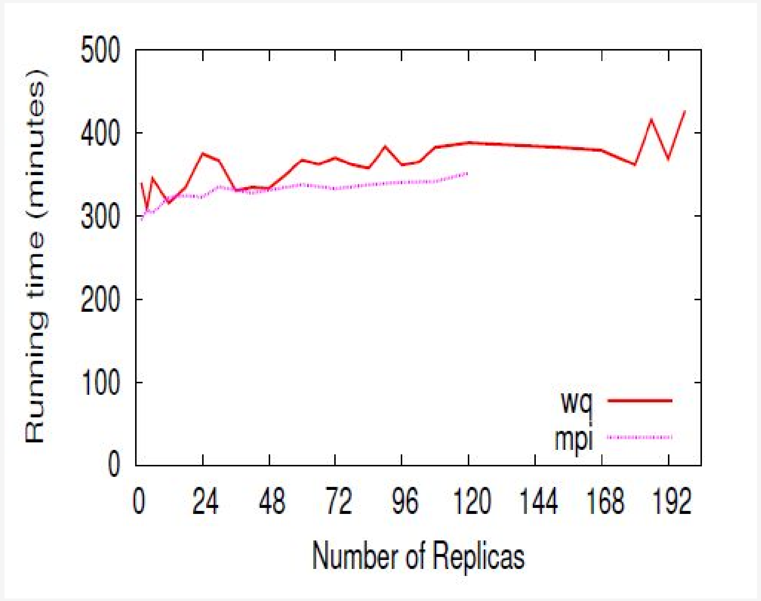


Replica Exchange

Simplified Algorithm:

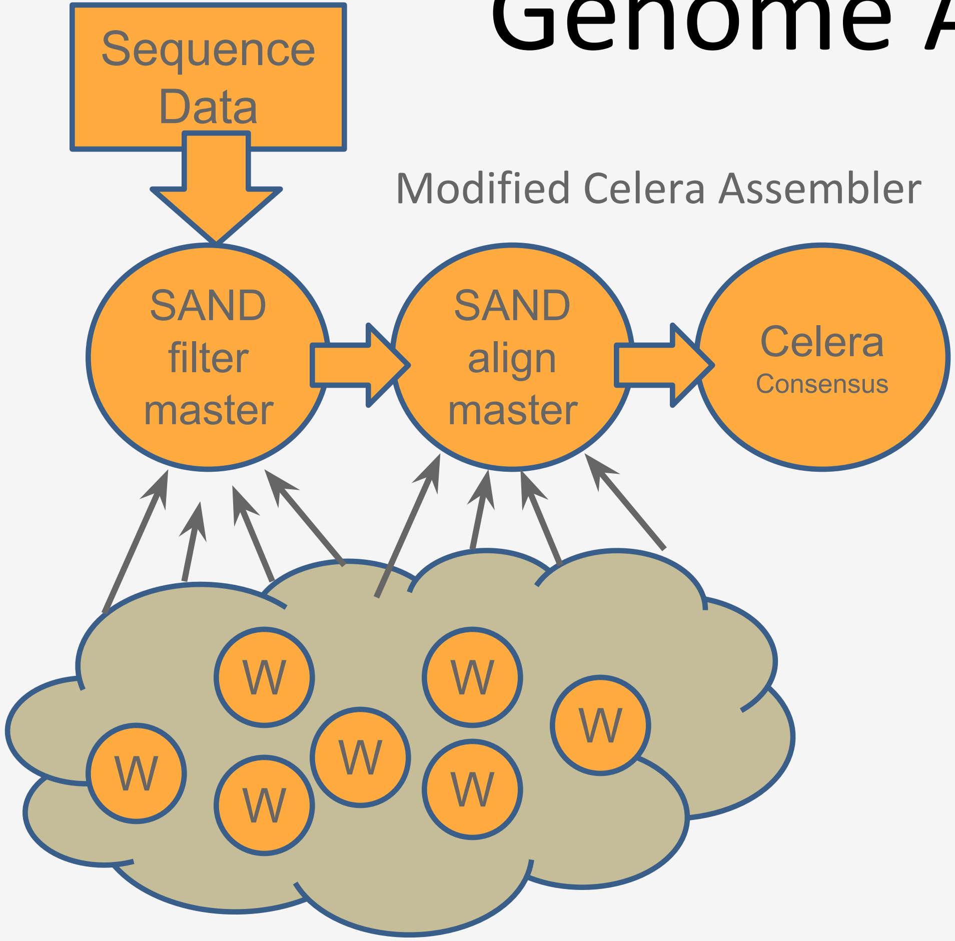
- Submit N short simulations at different temps.
- Wait for all to complete.
- Select two simulations to swap.
- Continue all of the simulations.

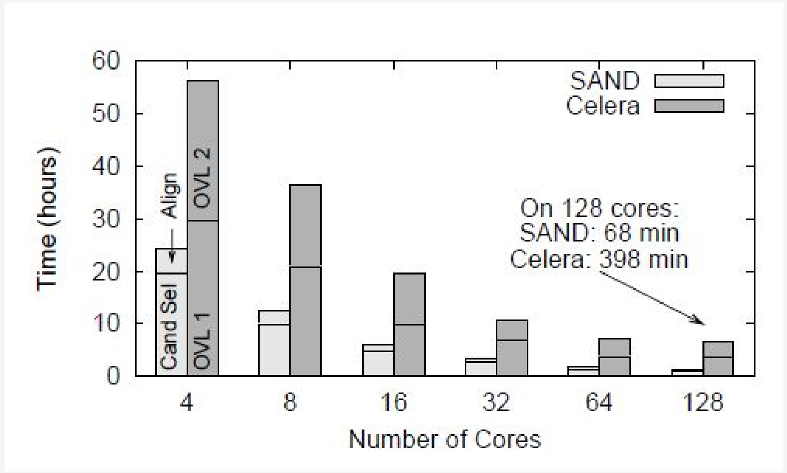




Dinesh Rajan, Anthony Canino, Jesus A Izaguirre, and Douglas Thain, **Converting A High Performance Application to an Elastic Cloud Application**, Cloud Com 2011.

Genome Assembly



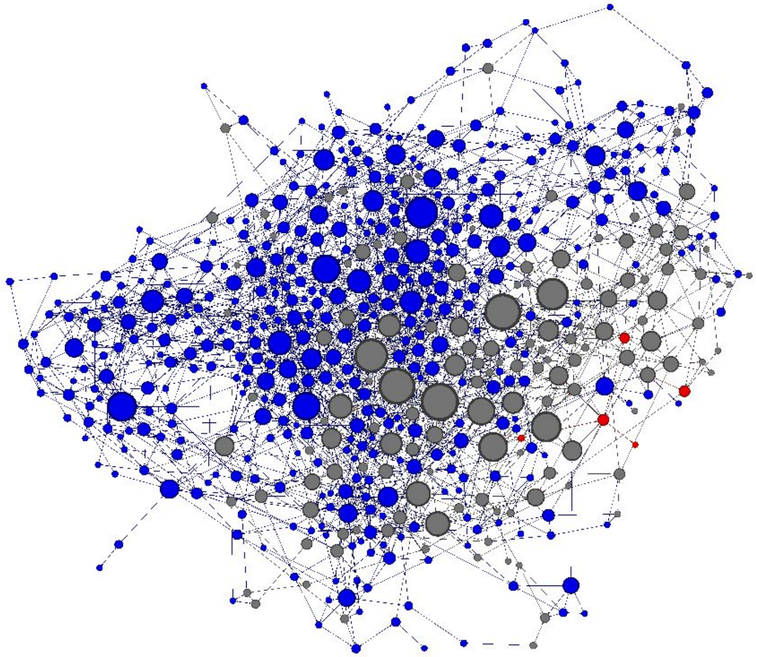


Using WQ, we could assemble a human genome in 2.5 hours on a collection of clusters, clouds, and grids with a speedup of 952X.

Christopher Moretti, Andrew Thrasher, Li Yu, Michael Olson, Scott Emrich, and Douglas Thain,
A Framework for Scalable Genome Assembly on Clusters, Clouds, and Grids,
IEEE Transactions on Parallel and Distributed Systems, **2012**

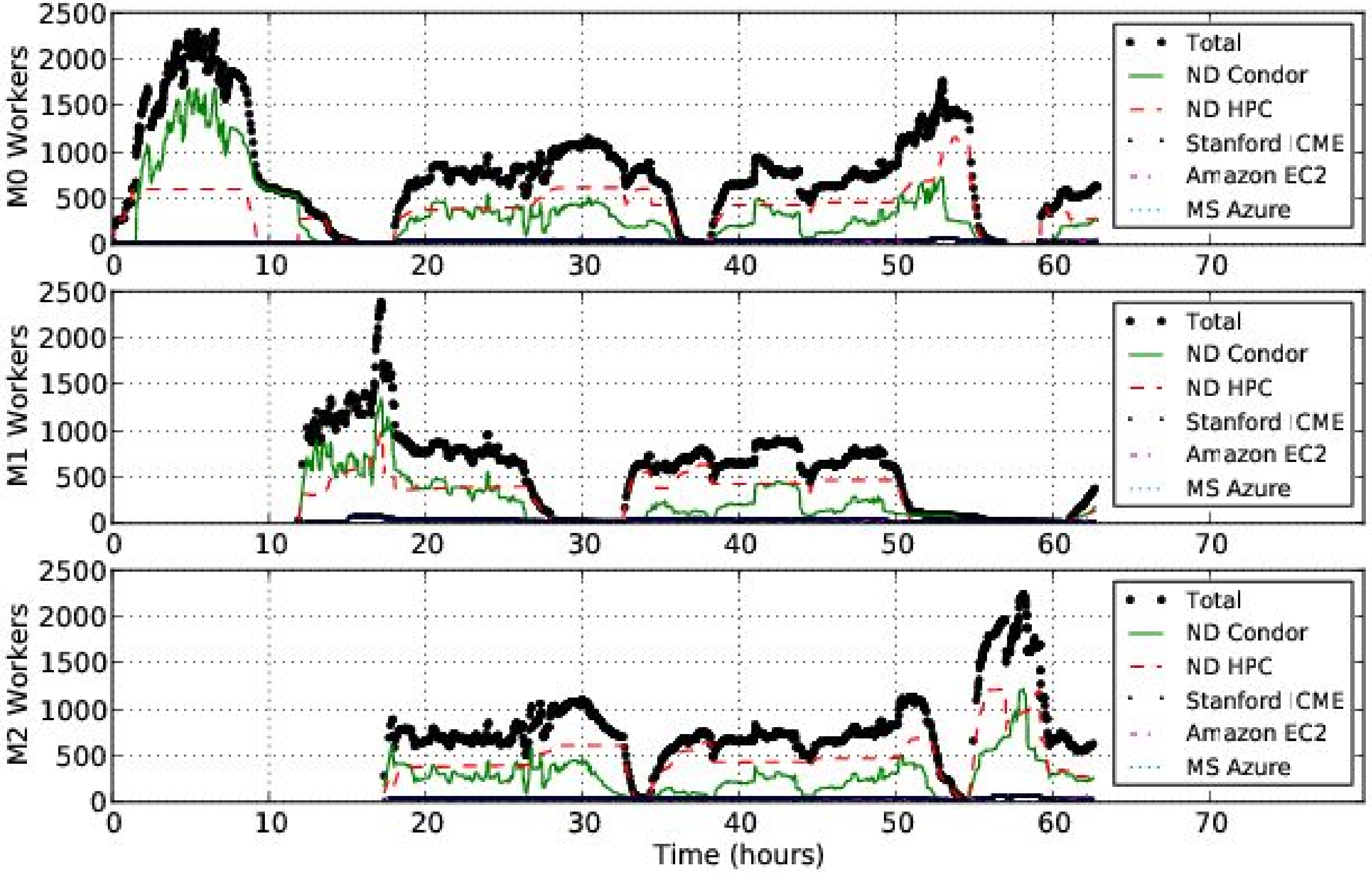
Adaptive Weighted Ensemble

Proteins fold into a number of distinctive states, each of which affects its function in the organism.

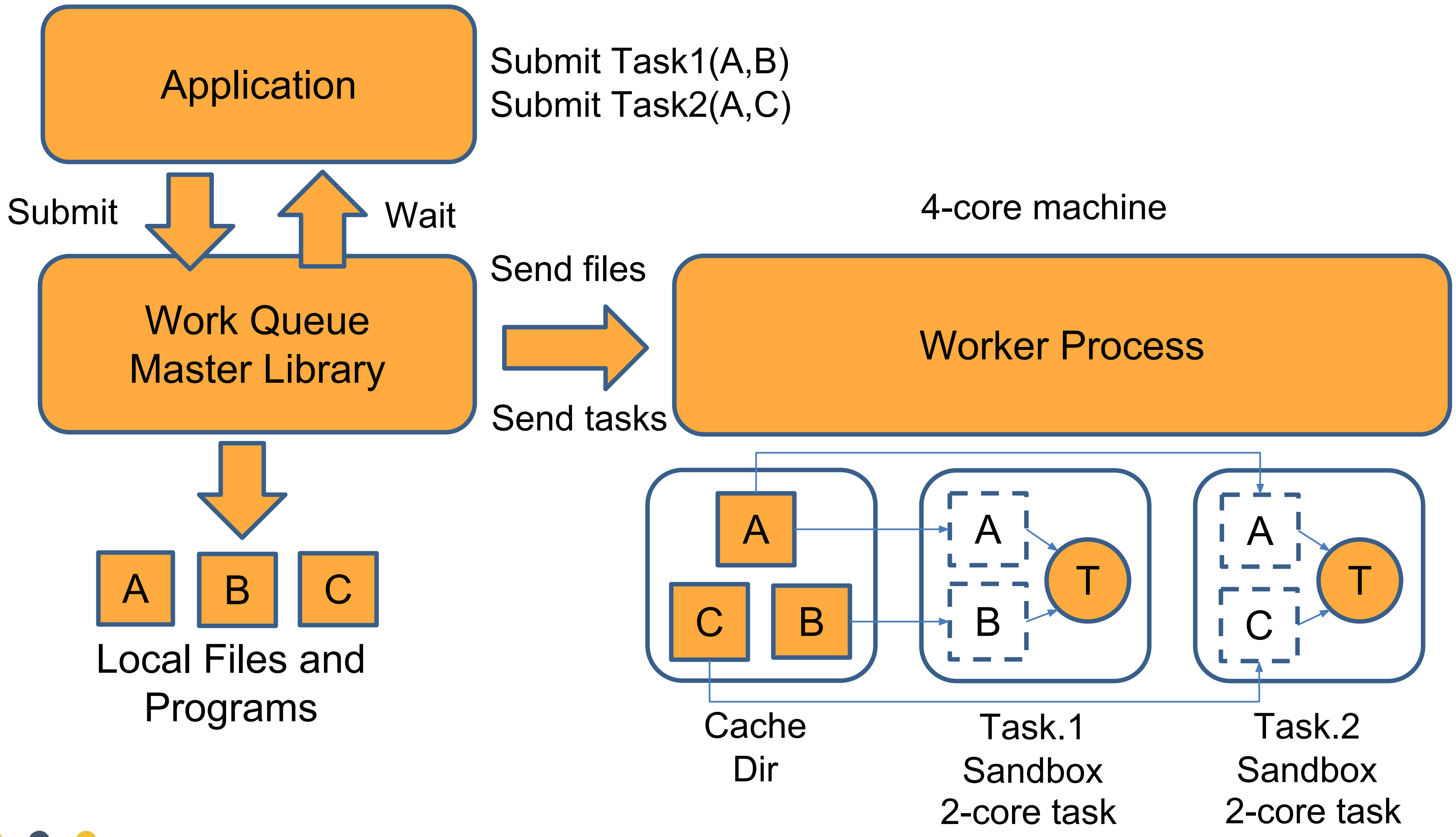


How common is each state?
How does the protein transition between states?
How common are those transitions?

AWE on Clusters, Clouds, and Grids



Work Queue Architecture



Basic Queue Operations

```
#include “work_queue.h”
struct work_queue *queue;
struct work_queue_task *task;

// Creates a new queue listening on a port, use zero to pick any port.
queue = work_queue_create( port );
// Submits a task into a queue.  (non-blocking)
work_queue_submit( queue, task );
// Waits for a task to complete, returns the complete task.
task = work_queue_wait( queue, timeout );
// Returns true if there are no tasks left in the queue.
work_queue_empty( queue );
// Returns true if the queue is hungry for more tasks.
work_queue_hungry( queue );
```

Basic Task Operations

```
#include "work_queue.h"
struct work_queue_task *task;

// Create a task that will run a given Unix command.
task = work_queue_task_create( command );

// Indicate an input or output file needed by the task.
work_queue_task_specify_file( task, name, remote_name, type, flags );

// Indicate an input buffer needed by the task.
work_queue_task_specify_buffer( task, data, length, remote_name, flags);

// Destroy the task object.
work_queue_task_delete( task );
```

Run One Task in C

```
#include “work_queue.h”

struct work_queue *queue;
struct work_queue_task *task;

queue = work_queue_create( 0 );
work_queue_specify_name( “myproject” );

task = work_queue_task_create(“sim.exe –p 50 in.dat >out.txt”);
/// Missing: Specify files needed by the task.
work_queue_submit( queue, task );

while(!work_queue_empty(queue)) {
    task = work_queue_wait( queue, 60 );
    if(task) work_queue_task_delete( task );
}
```

CCTools

UNIVERSITY OF NOTRE DAME

Run One Task in Perl

```
use work_queue;

$queue = work_queue_create( 0 );

work_queue_specify_name( “myproject” );

$task = work_queue_task_create(“sim.exe –p 50 in.dat >out.txt”);
### Missing: Specify files needed by the task.

work_queue_submit( $queue, $task );

while(!work_queue_empty($queue)) {
    $task = work_queue_wait( $queue, 60 );
    if($task) work_queue_task_delete( $task );
}
```

CCTools

UNIVERSITY OF NOTRE DAME

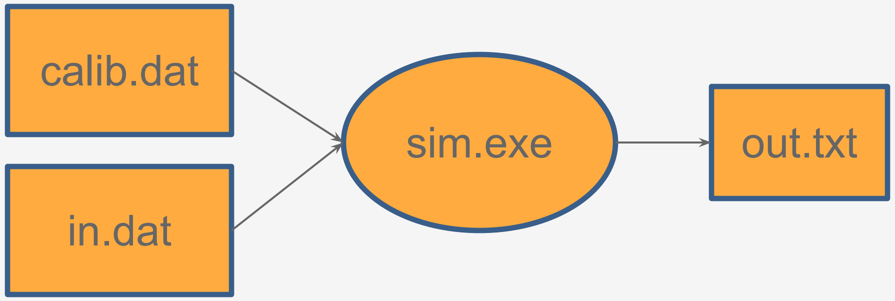
Run One Task in Python

```
from work_queue import *

queue = WorkQueue( port = 0 )

queue.specify_name( “myproject” );

task = Task(“sim.exe –p 50 in.dat >out.txt”)

### Missing: Specify files needed by the task.
queue.submit( task )

While not queue.empty():
    task = queue.wait(60)
```

CCTools

UNIVERSITY OF NOTRE DAME

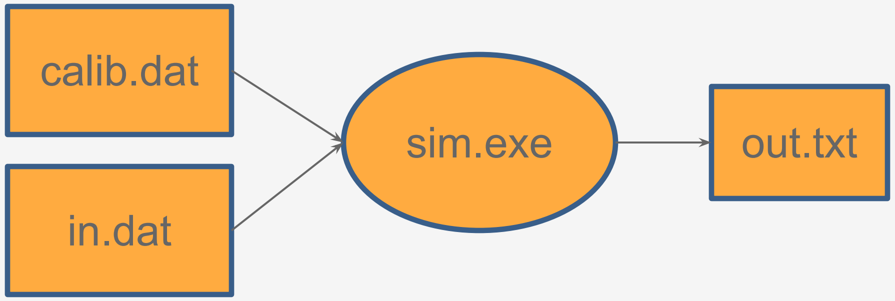
C: Specify Files for a Task



```
sim.exe in.dat –p 50 > out.txt
```

```
work_queue_task_specify_file( $task,“in.dat”,”in.dat”,
            $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file($task,“calib.dat”,”calib.dat”,
           $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,“out.txt”,”out.txt”,
           $WORK_QUEUE_OUTPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,“sim.exe”,”sim.exe”,
            $WORK_QUEUE_INPUT, $WORK_QUEUE_CACHE );
```

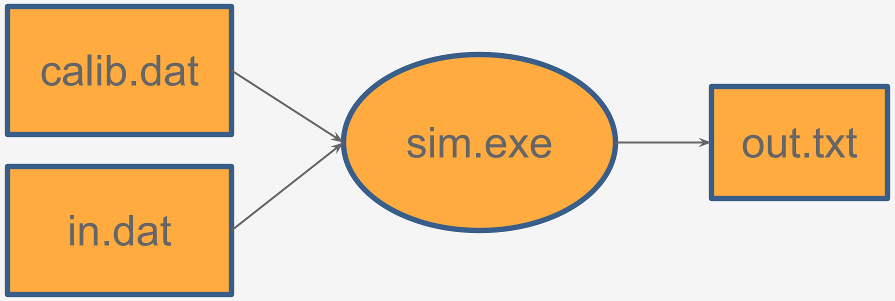
Perl: Specify Files for a Task



sim.exe in.dat –p 50 > out.txt

```
work_queue_task_specify_file( $task,“in.dat”,”in.dat”,
            $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file($task,“calib.dat”,”calib.dat”,
           $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,“out.txt”,”out.txt”,
           $WORK_QUEUE_OUTPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,“sim.exe”,”sim.exe”,
            $WORK_QUEUE_INPUT, $WORK_QUEUE_CACHE );
```

Python: Specify Files for a Task



```
sim.exe in.dat –p 50 > out.txt
```

```
task.specify_file( “in.dat”, ”in.dat”,
           WORK_QUEUE_INPUT, cache = False )

task.specify_file( “calib.dat”, ”calib.dat”,
           WORK_QUEUE_INPUT, cache = False )

task.specify_file( “out.txt”, ”out.txt”,
           WORK_QUEUE_OUTPUT, cache = False )

task.specify_file( “sim.exe”, ”sim.exe”,
            WORK_QUEUE_INPUT, cache = True )
```

You must state
all the files
needed by the command.

CCTools

UNIVERSITY OF NOTRE DAME

Running a Work Queue Program

```
gcc  work_queue_example.c   -o work_queue_example
     -I $HOME/cctools/include/cctools
     -L $HOME/cctools/lib
     -lwork_queue -ldttools  -lm
```

```
./work_queue_example
Listening on port 8374 …
```

In another window:

```
./work_queue_worker   master.host.name.org  8374
```

… for Perl

```
setenv PERL5LIB ${PERL5LIB}:   (no line break)
    ${HOME}/cctools/lib/perl5/site_perl
```

```
./work_queue_example.pl
Listening on port 8374 …
```

In another window:

```
./work_queue_worker   master.host.name.org   8374
```

… for Python

```
setenv PYTHONPATH ${PYTHONPATH}:    (no line break)
    ${HOME}/cctools/lib/python2.6/site-package

./work_queue_example.py
Listening on port 8374 …
```

In another window:

```
./work_queue_worker   master.host.name.org   8374
```

Start Workers Everywhere

Submit workers to Condor:

```
condor_submit_workers master.hostname.org 8374 25
```

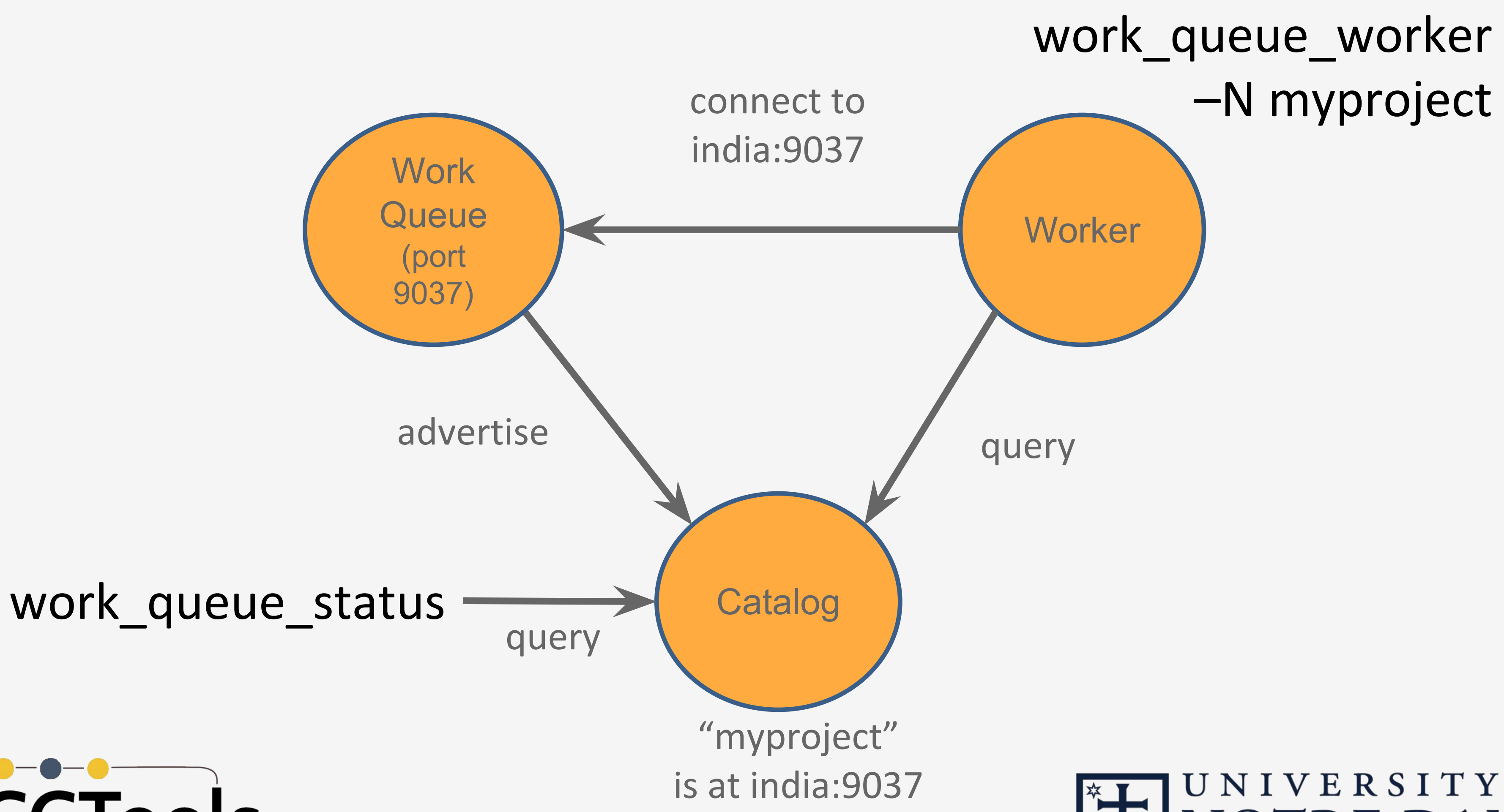
Submit workers to SGE:

```
sge_submit_workers  master.hostname.org 8374 25
```

Submit workers to Torque:

```
torque_submit_workers  master.hostname.org 8374 25
```

Use Project Names



Specify Project Names in Work Queue

Specify Project Name for Work Queue master:

C:

```
work_queue_specify_name (q, “myproject”);
```

Perl:

```
work_queue_specify_name ($q, “myproject”);
```

Python:

```
q.specify_name (“myproject”)
```

Start Workers with Project Names

Start one worker:

```
$ work_queue_worker  -N myproject
```

Start many workers:

```
$ sge_submit_workers -N myproject  5
$ condor_submit_workers -N myproject  5
$ torque_submit_workers -N myproject  5
```

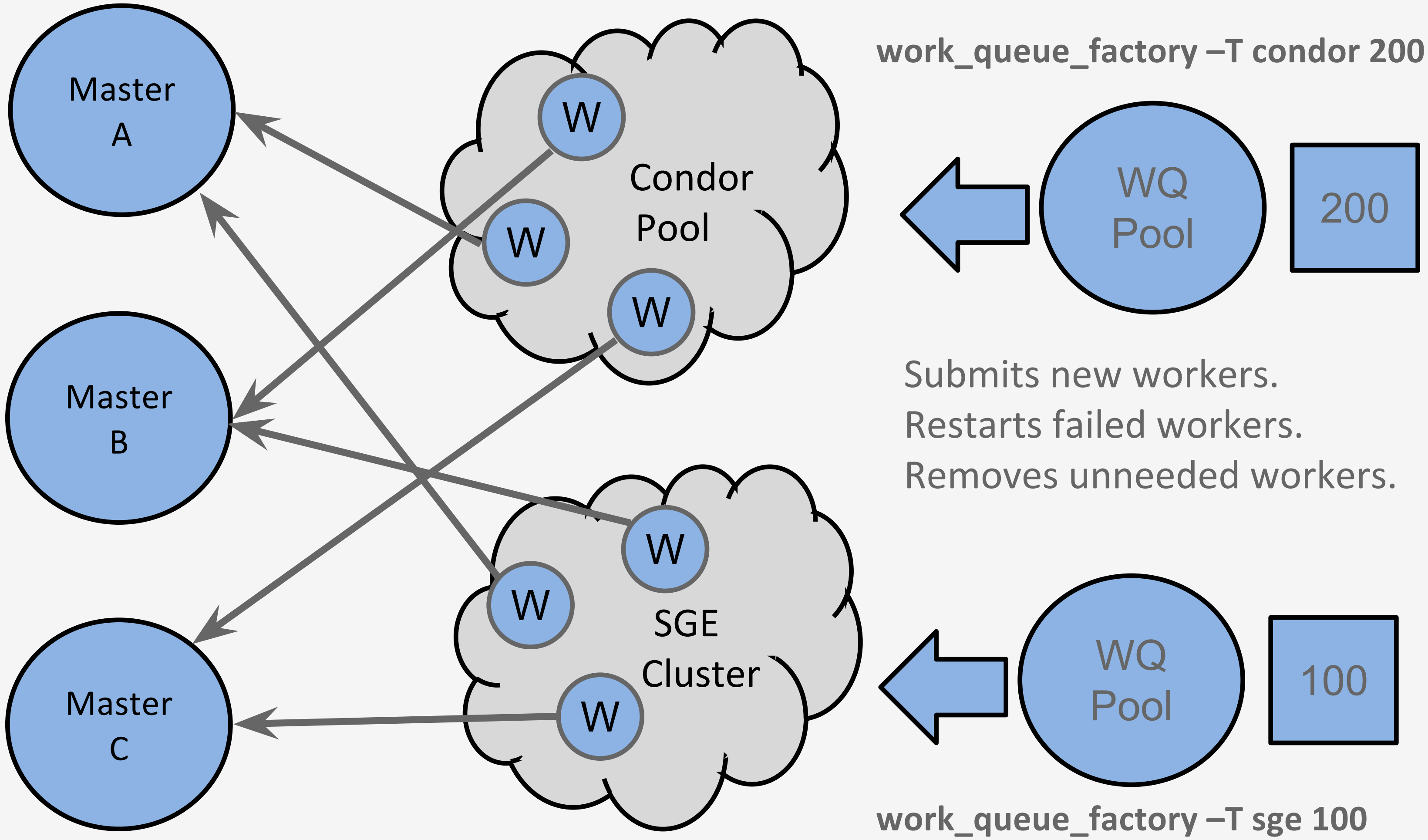
CCTools

UNIVERSITY OF NOTRE DAME

Advanced Features (in the docs)

- Submit / remove tasks by tag / name.
- Auto reschedule tasks that take too long.
- Send in-memory data as a file.
- Log and graph system performance
- Much more!

Managing Your Workforce



```
work_queue_factory –T condor 200
```

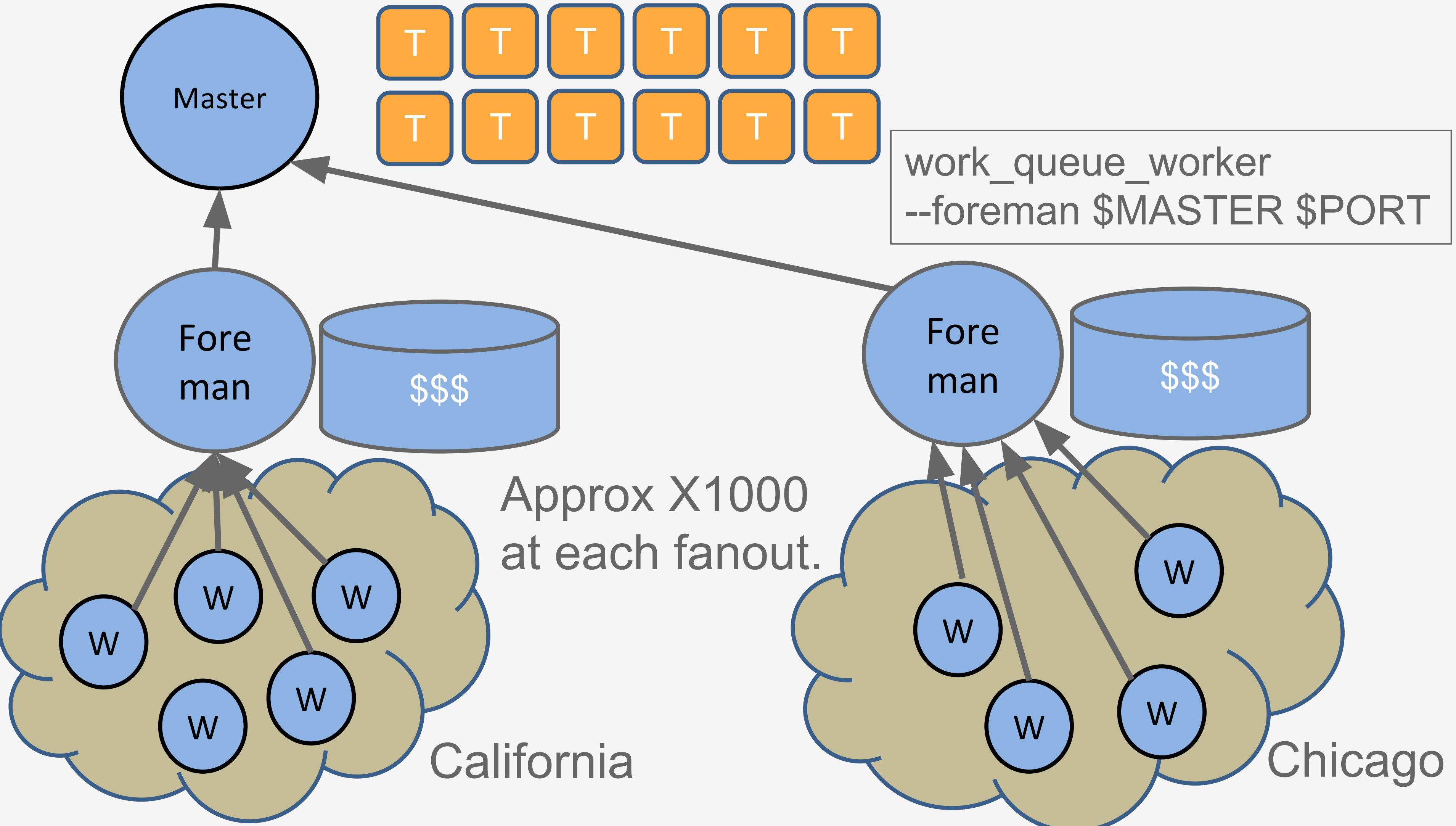
Submits new workers.
Restarts failed workers.
Removes unneeded workers.

```
work_queue_factory –T sge 100
```

CCTools

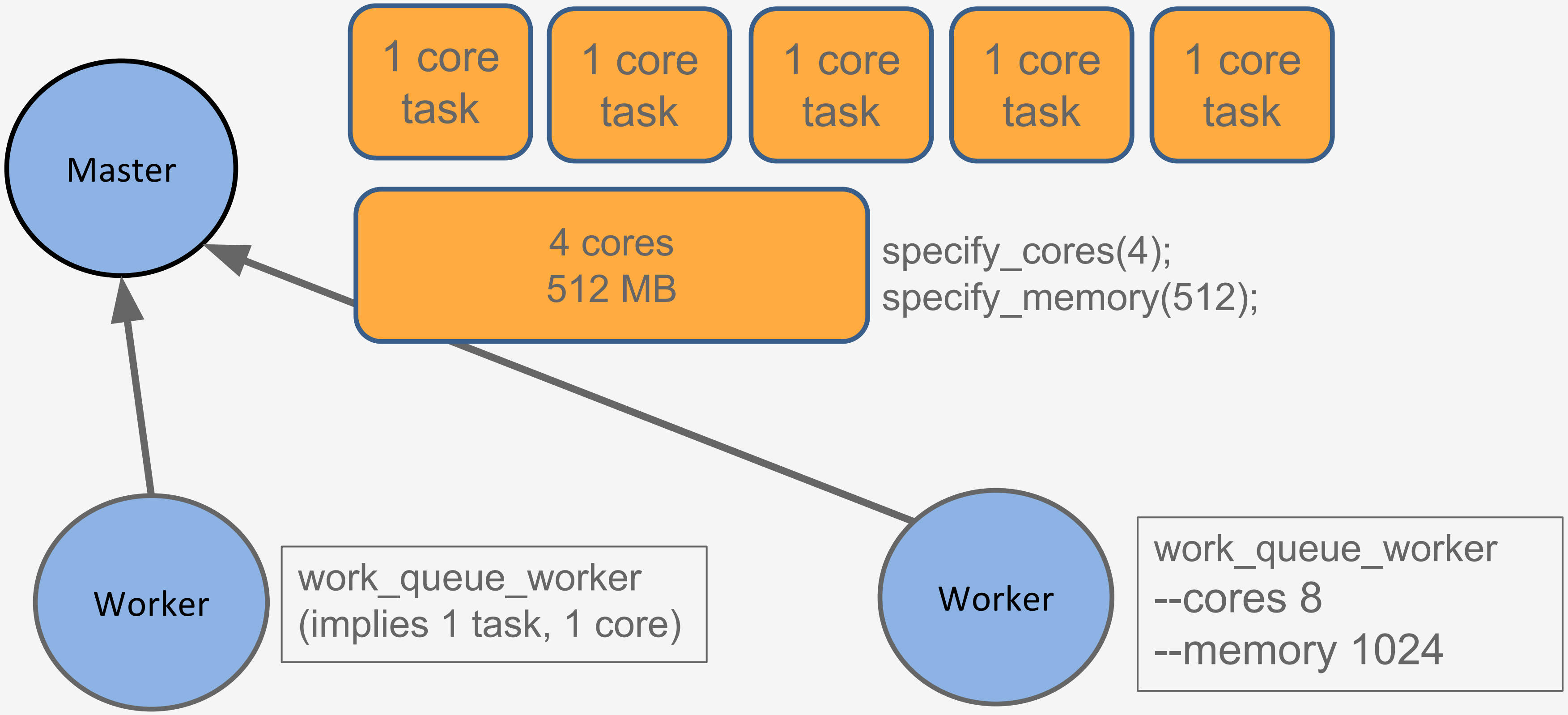
UNIVERSITY OF NOTRE DAME

Using Foremen



```
work_queue_worker
--foreman $MASTER $PORT
```

Multi-Slot Workers



```
specify_cores(4);
specify_memory(512);
```

```
work_queue_worker
(implies 1 task, 1 core)
```

```
work_queue_worker
--cores 8
--memory 1024
```