Advance Your Research, Scale Out Your Computing

Amber Lim, PhD
Center for High Throughput Computing
STAT 405/605
March 13, 2025

CHTC Center For High Throughput Computing

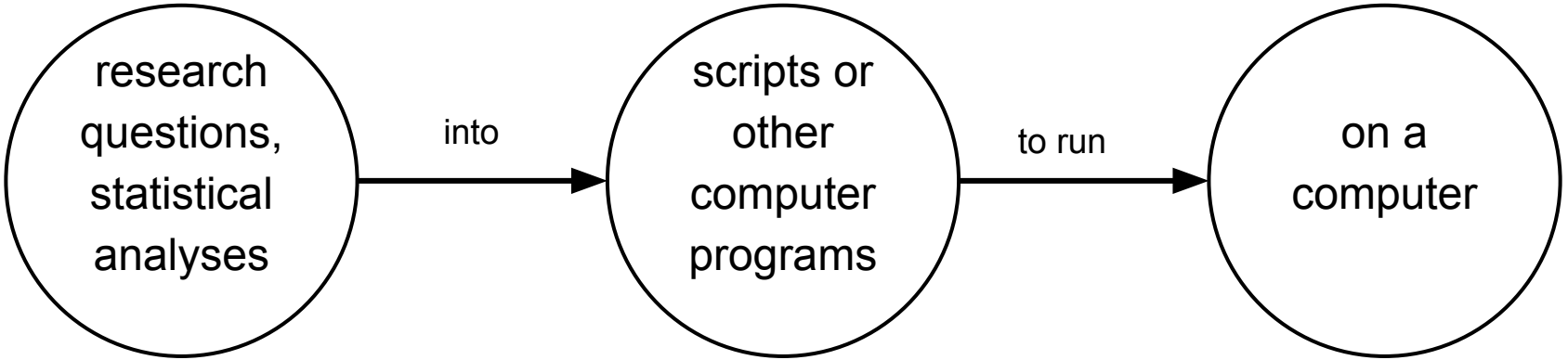
Overview

- Why Large Scale Computing?
- Large Scale Computing Approaches
- The Center for High Throughput Computing
- Running Jobs
- Next Steps For Using CHTC

Why Large Scale Computing?

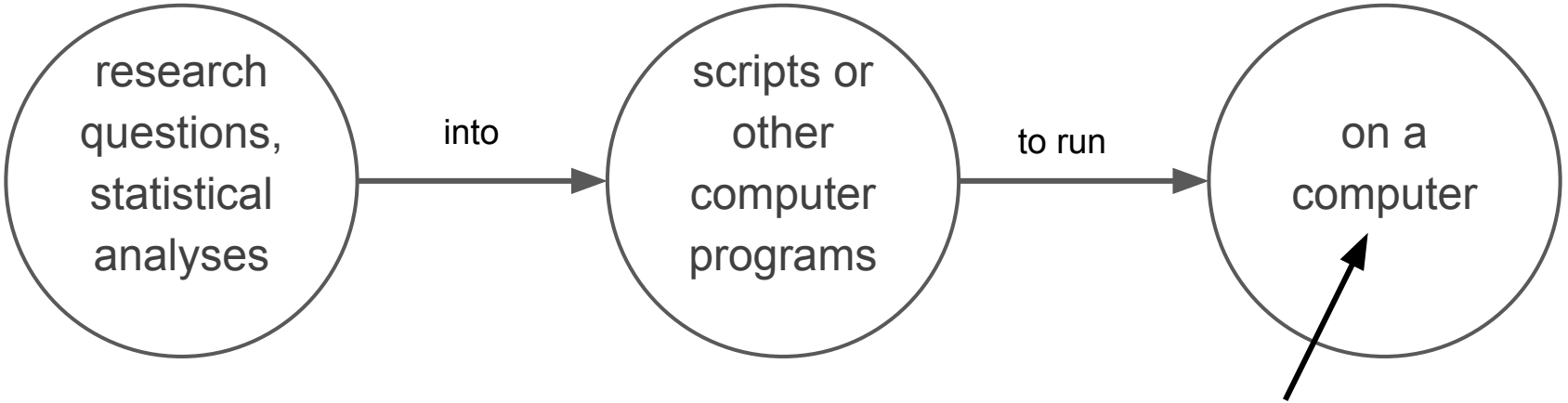
Computing Requires Computers

In this course, you are learning how to translate:



Computing Requires Computers

In this course, you are learning how to translate:



Today, we're going to focus on this part of the process and ask…

What could go wrong?

Richard Ayoade in the IT Crowd, Series 1, Episode 2
https://giphy.com/gifs/richard-ayoade-it-crowd-maurice-moss-dbtDDSvWErdf2

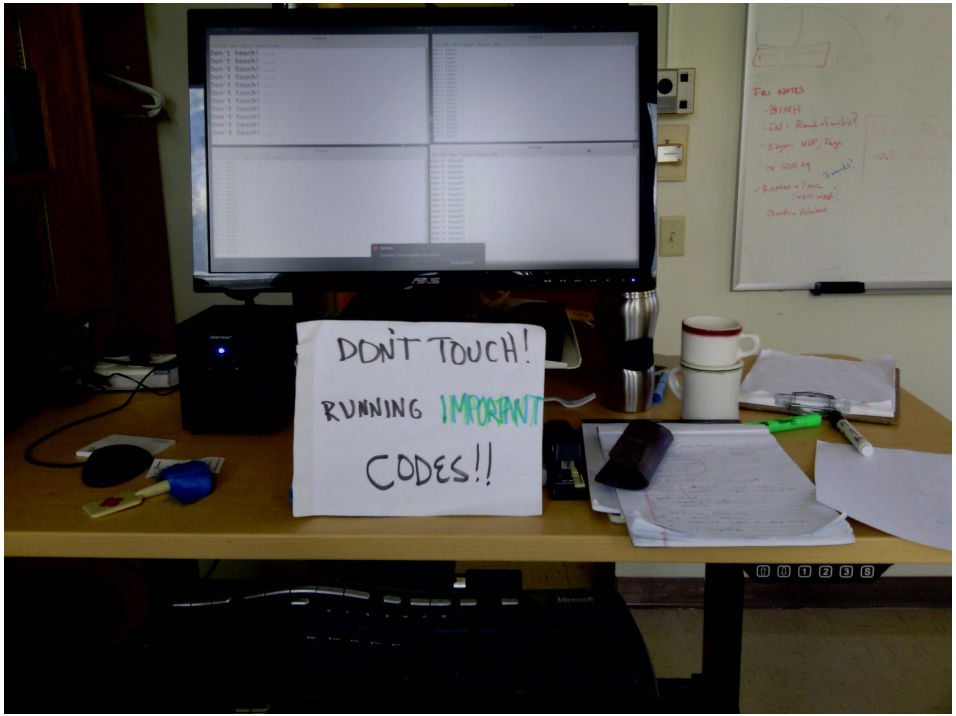


Photo credit Steve Goldstein

Dr. Timothée Poisot
@tpoi

Just use this simple heuristic: if it's slow and the computer goes WOOOOOOOOSH, the bottleneck is the CPU; if it's slow and the mouse stop moving, the bottleneck is the memory.

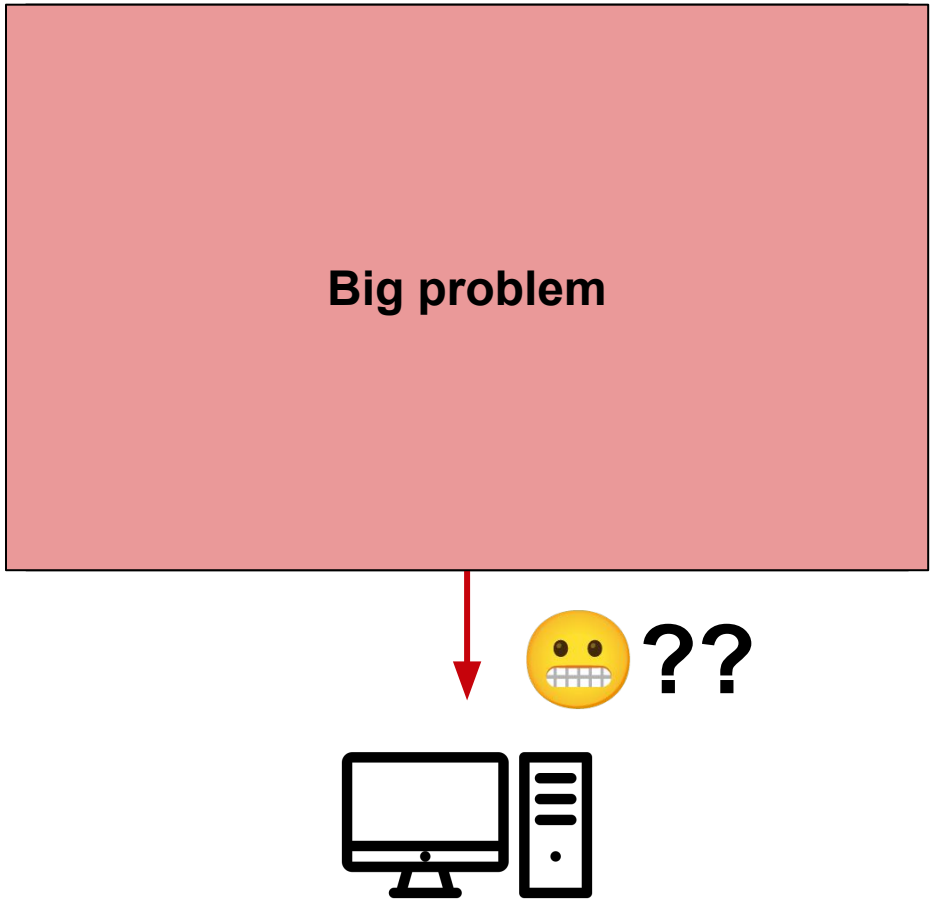
https://twitter.com/tpoi/status/1169692855201402885

Big Problems

How do you solve a big computational problem?

For example:

- Lots of data
- Thousands of parameters
- Many grid points

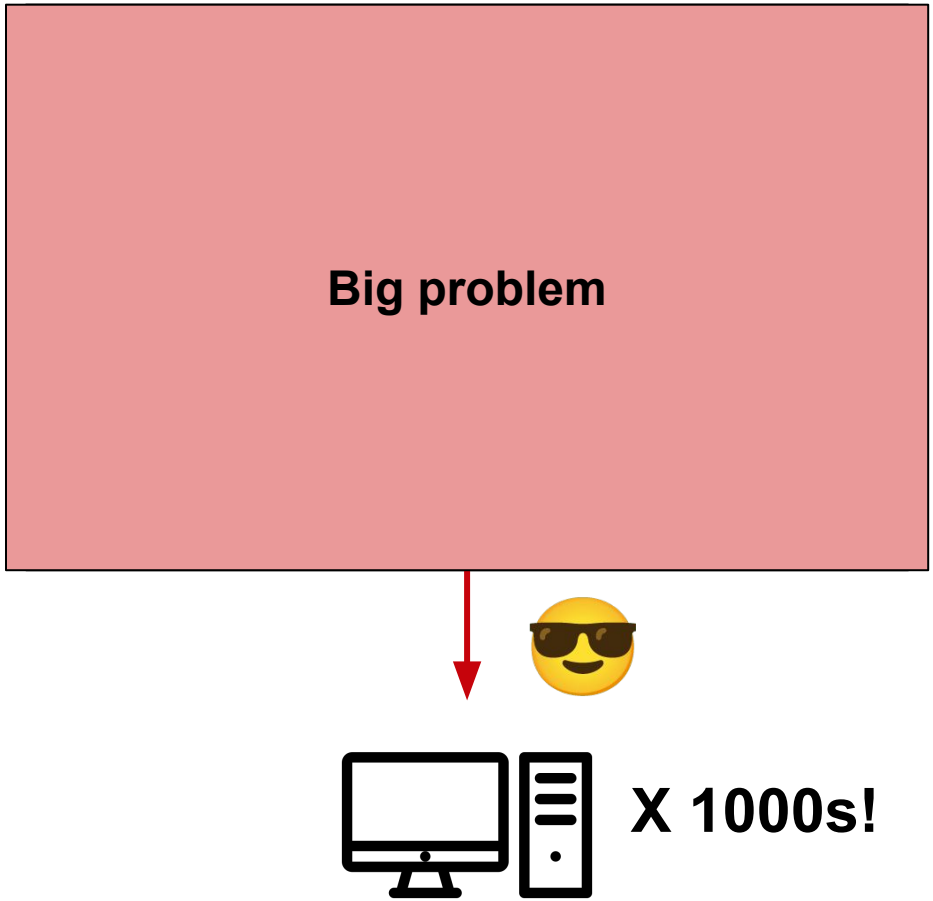


Our Goal: Scaling Out

We want to be able to tackle big problems.

We want to go from using a small number of resources to a LOT of resources.

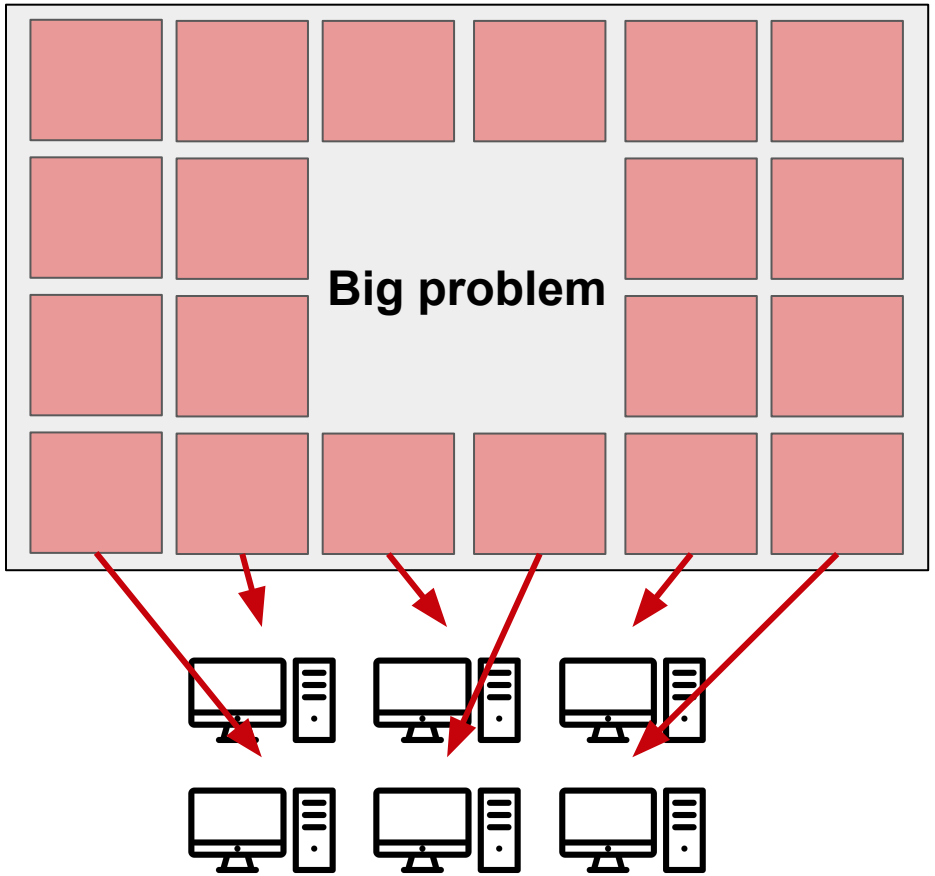
There is one strategy to do this with computers: divide and conquer



Big Solutions

To solve a big problem, break it into many sub-problems and use more/specialized computers.

There are different ways to do this.



Large Scale Computing Approaches

A bigger computer

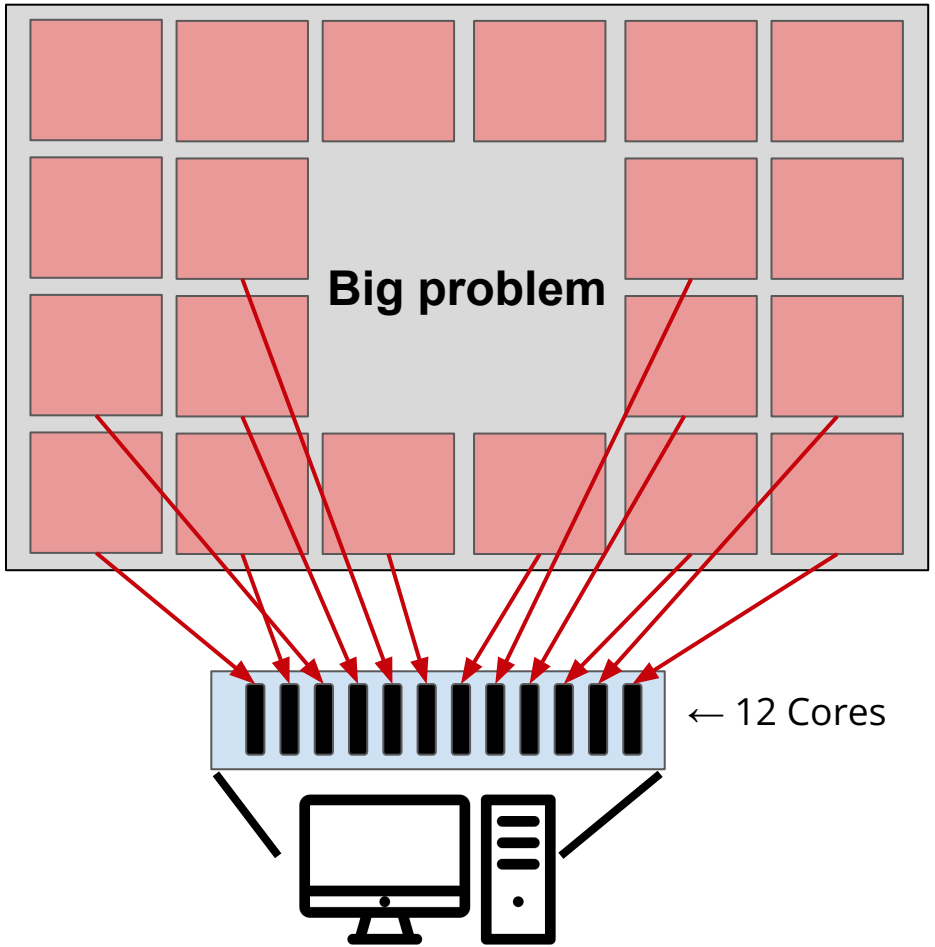
Sometimes, getting a larger computer can solve your problem - more cores/CPUs, more memory, more disk space.

- Workstation
- Server that requires remote log in

Option 1: More CPU Cores

Run sub-problems on individual CPU cores.

Requires programming this capability into the code.



Specialized computer components (like GPUs)

GPUs are a common way to accelerate certain kinds of computing.

Like before, can be used from a:

- Workstation
- Server

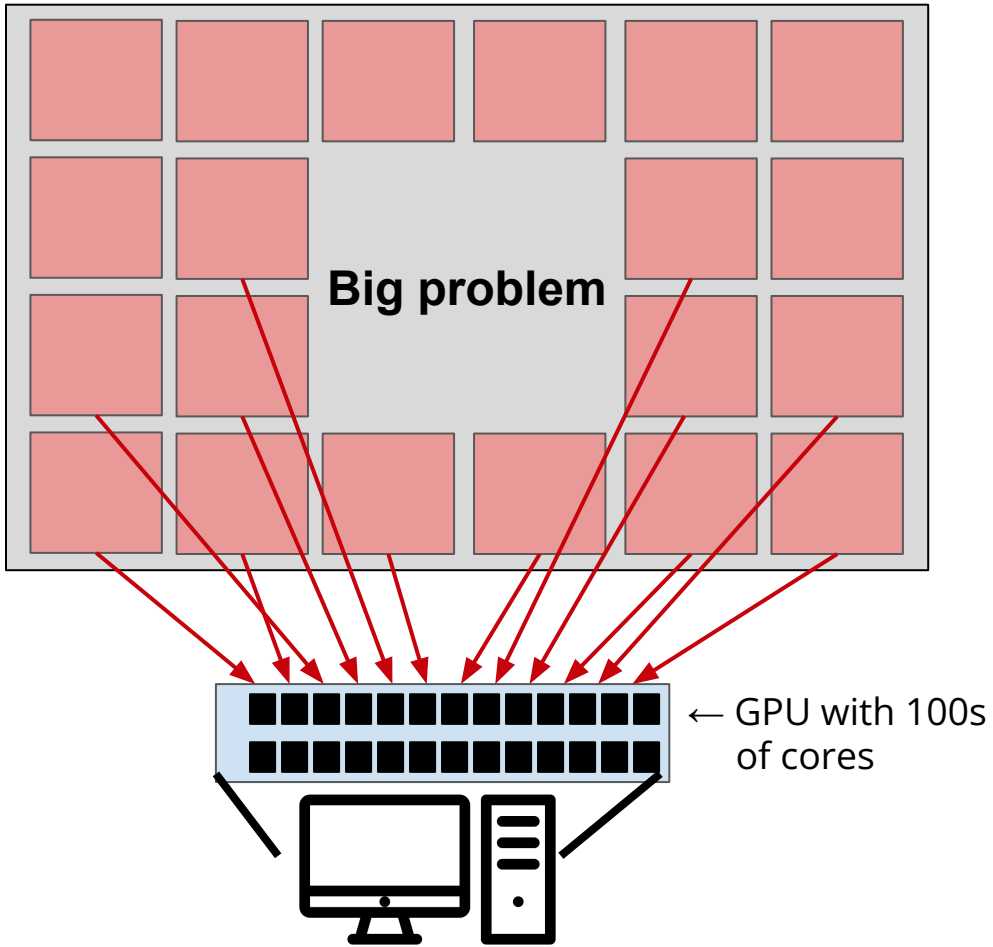
Option 2: Use a GPU

Run sub-problems on little cores inside a GPU card.

Like before, requires programming this capability into the code.

Examples:

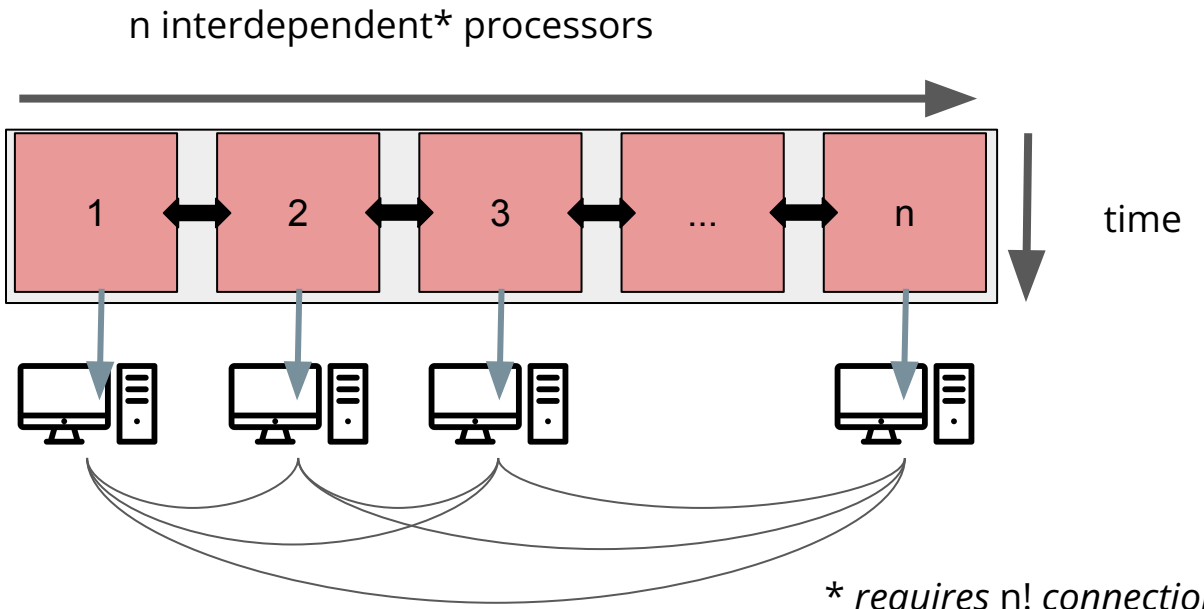
- Neural networks and other deep learning algorithms
- Certain molecular dynamic codes



Use more computers at once.

This is the idea behind the large-scale computing systems at CHTC.

High **Performance** Computing (HPC)



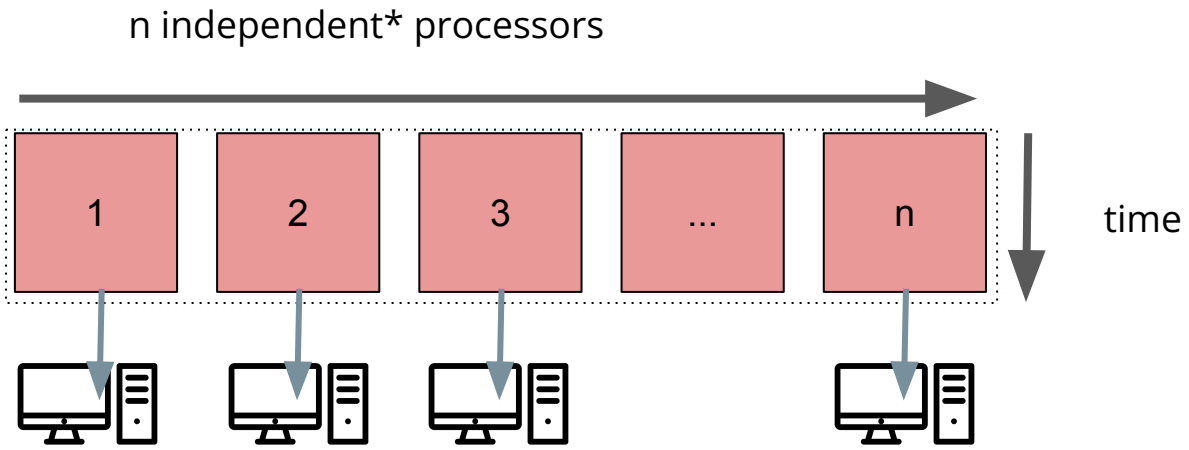
Solve sub-problems by distributing across multiple CPUs on multiple computers.

Also requires special coding (MPI) to coordinate calculations across multiple computers.

HPC Examples:

- Complex physical simulations with many inter-connected components.
- Optimization problems that use many local calculations to drive the global optimization.

High **Throughput** Computing (HTC)



Sub-problems are independent and self-contained, can be run on many computers.

Most scalable.

No special programming needed.

HTC Examples:

- Test many parameter combinations
- Analyze multiple images or datasets
- Do a replicate/randomized analysis
- Align genome/RNA sequence data

One of our favorite HTC examples: baking the world's largest/longest cake

In computational terms: solving a big problem (the world's longest cake) by executing many small, self-contained tasks (individual cakes) and joining them.

Photos: Arun Sankar via https://www.theguardian.com/world/2020/jan/16/indian-bakers-rise-to-task-of-making-worlds-longest-cake

Large Scale Computing at CHTC

Center for High Throughput Computing (CHTC)

The Center for High Throughput Computing (CHTC) operates a **High Throughput Computing** system and a **High Performance Computing** system.

High Throughput Computing System

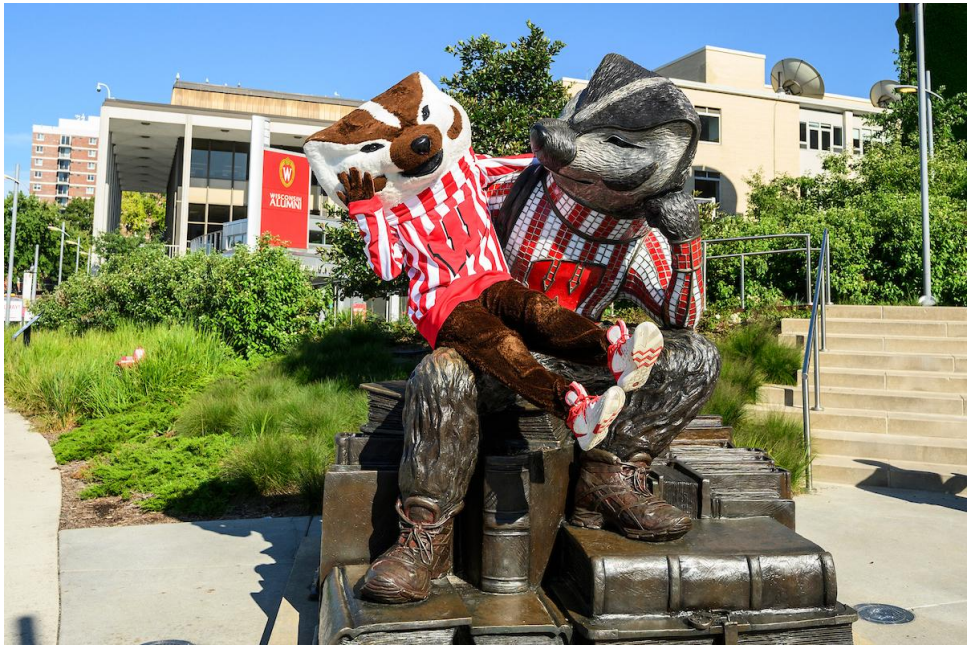
- 100s of servers
- Access to over 40,000 cores
- 80+ GPUs
- High memory machines (TBs RAM)
- Operating system: CentOS Stream 9

High Performance Computing System

- 5,120 cores of capacity
- 40 general use execute nodes (128 AMD Epyc 7763 processor cores, 512GB memory)
- 1.5TB of local (not shared) fast NVMe disk
- Operating system: CentOS Stream 9

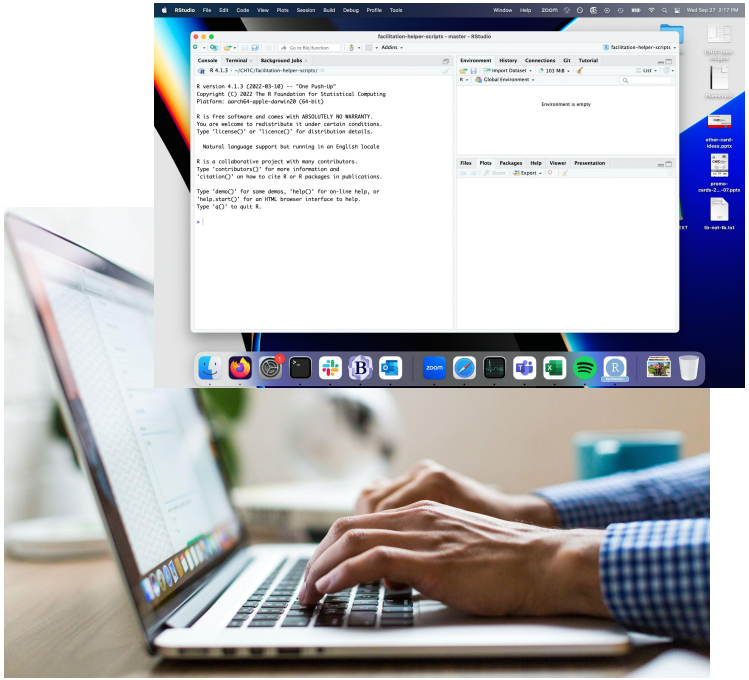
Who can use CHTC?

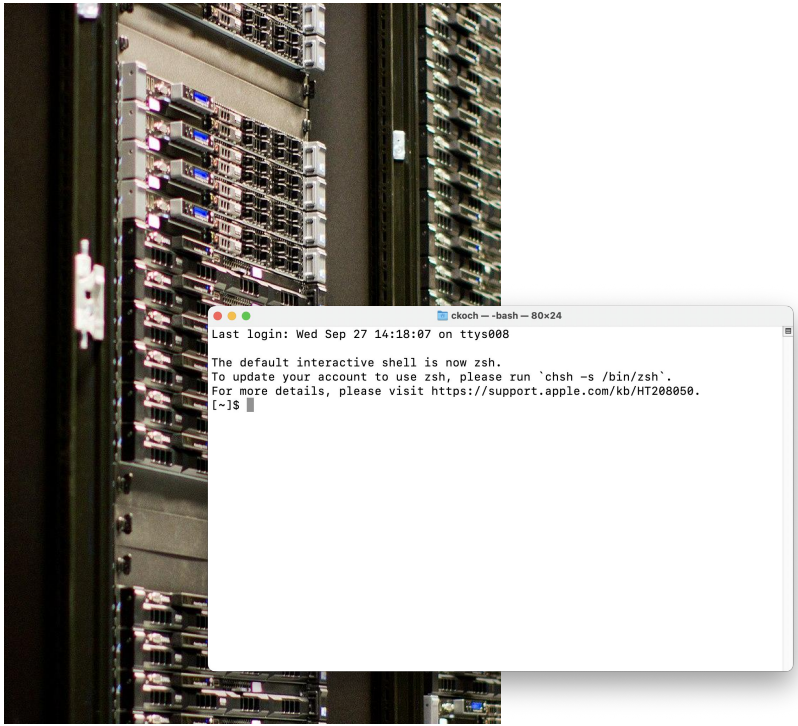
Any UW Madison affiliate (faculty, staff, student, post-doc) in a class or research group can access CHTC services **at no cost**.



Running Jobs in CHTC's HTC System

Transitioning From Your Computer to CHTC



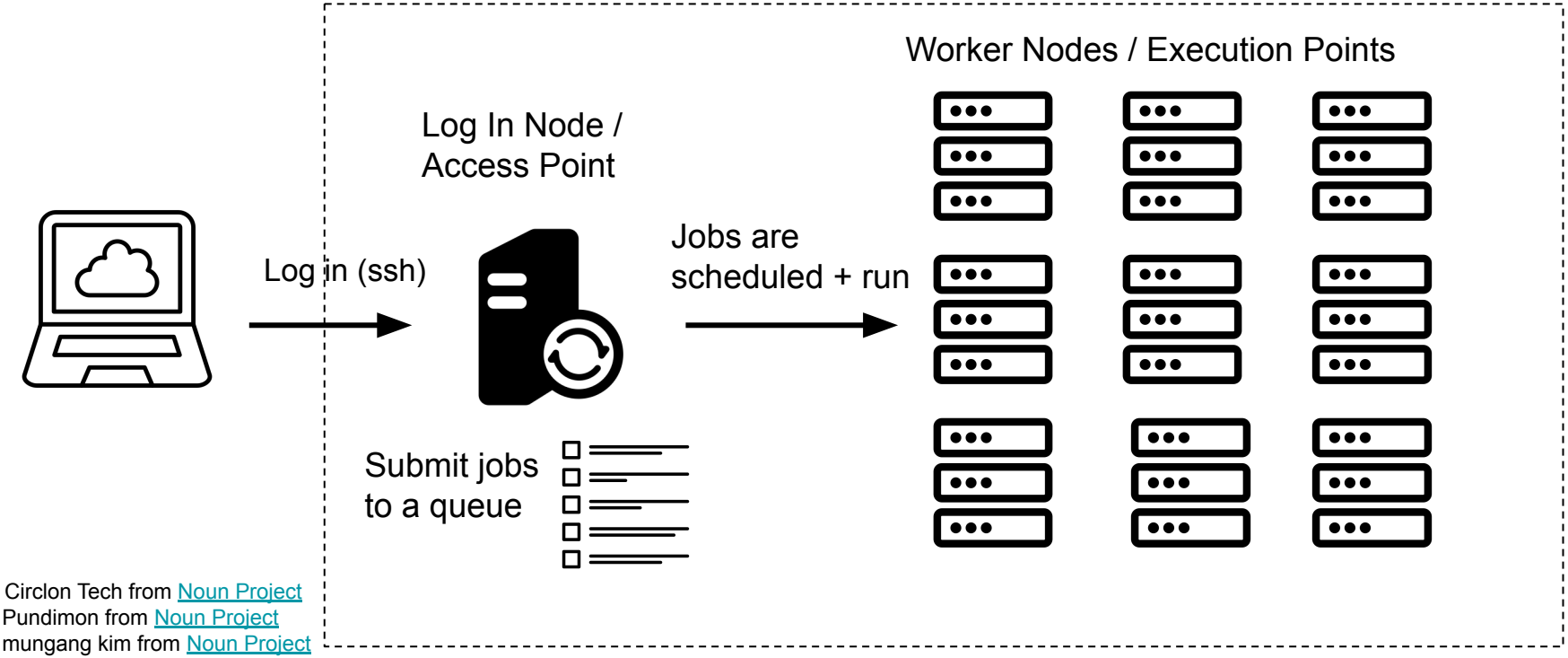


Transitioning From Your Computer to CHTC

Key Differences:

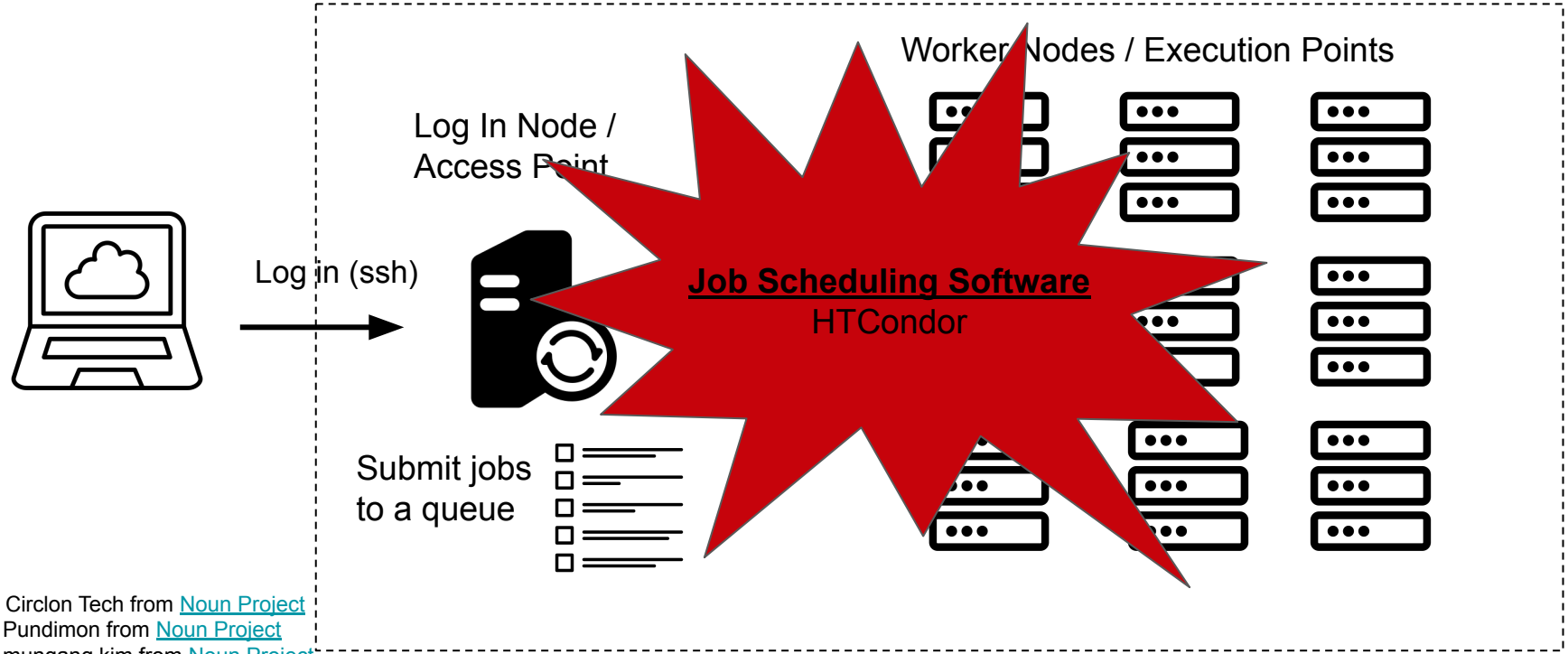
- Noninteractive
  - Tasks need to be scripted
- Command line interface
  - Need to learn some basic bash commands
- Shared system
  - Following policies and recommendations matters for you and others
- Need portable software, data
  - “Backpacking” mentality - can only use what you take with you
- No admin permissions
  - Can’t use **sudo** for setting up/installing software

Using a Large-Scale Computing System



Laptop by Circlon Tech from Noun Project
Server by Pundimon from Noun Project
Server by mungang kim from Noun Project
Checklist by Sofía Trujillo Quesada from Noun Project (CC BY 3.0)

Using a Large-Scale Computing System



Laptop by Circlon Tech from Noun Project
Server by Pundimon from Noun Project
Server by mungang kim from Noun Project
Checklist by Sofía Trujillo Quesada from Noun Project (CC BY 3.0)

Job Components

- Command(s) to run
- Resource requests
  - Cores, memory
  - Disk (HTC)
- How to record job information
  - Options for standard out / standard error (what prints to the terminal)
  - Log file
- Software environment
  - Containers, tar.gz files
- Input files

HTC Example

Job Example

We define what we want to run as a “job.”
A job is basically whatever you want to run on the system.

For example, suppose we have a script called “`wordcount.py`” that has one input file (`Dracula.txt`) and produces an output file (`count.Dracula.tsv`)

```
$ wordcount.py Dracula.txt
```

To run this as a job, we need to communicate:

- What command to run
- Which files to use (script and input file)
- How much of the computer we need

Let’s submit a job!

1. Log in:
   ```
   ssh netid@learn.chtc.wisc.edu
   ```

2. Get files:
   ```
   cp -r /home/groups/STAT_DSCP/wordcount-jobs ./
   ```

3. Move into directory
   ```
   cd wordcount-jobs
   ```

4. Look at some of the files.
   ```
   ls
   ```

Job Submit File

List your executable and any arguments it takes.

Arguments are any options passed to the executable from the command line

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Job Submit File

Indicate your input files.

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Job Submit File

HTCondor will transfer back all new and changed files (usually output) from the job.

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Job Submit File

log: file created by HTCondor to track job progress

output/error: captures stdout and stderr

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Job Submit File

Request the appropriate resources for your job to run.

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Job Submit File

queue: keyword indicating “create a job.”

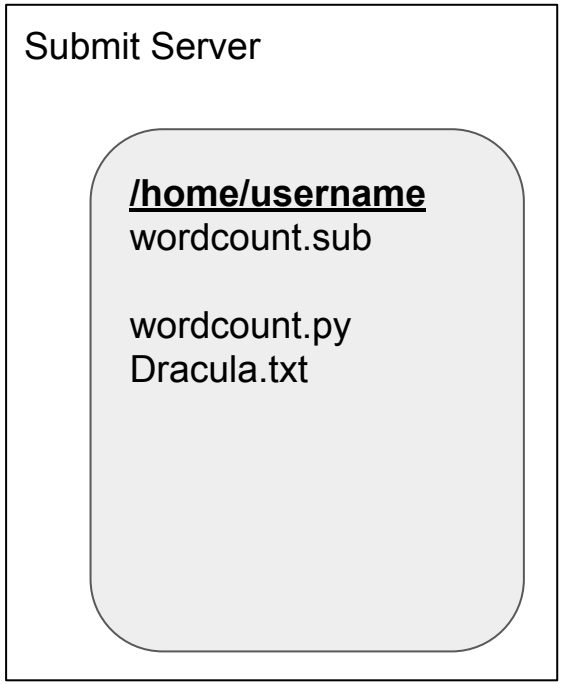
```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 1GB
request_memory = 1GB

queue 1
```

Submit It!
Monitor Progress.

```
executable = wordcount.py
arguments = Dracula.txt

should_transfer_files = YES
transfer_input_files = Dracula.txt
when_to_transfer_output = ON_EXIT

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

request_memory = 1GB

queue 1
```

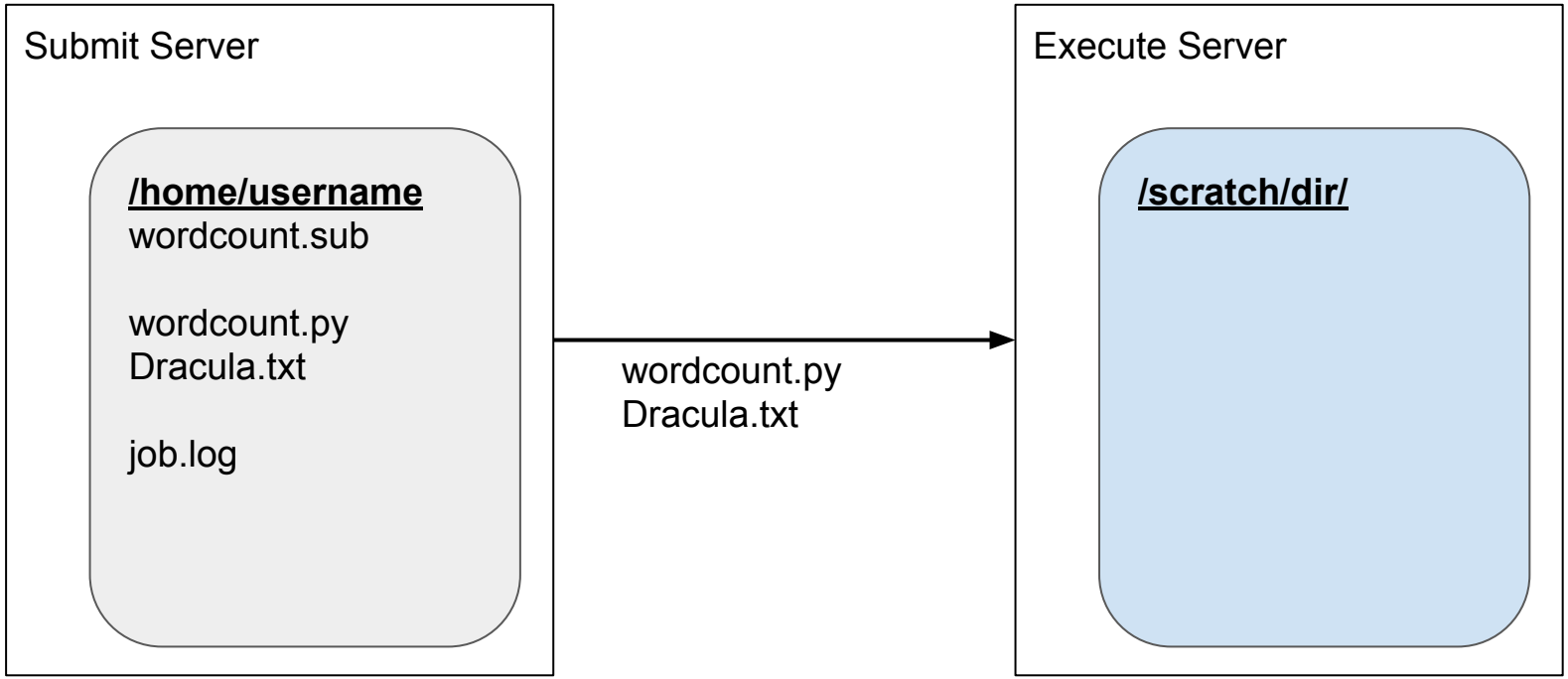
```
[user@submit]$ condor_submit wordcount.sub
[user@submit]$ condor_q
```

What Happens to the Job - Before Submission

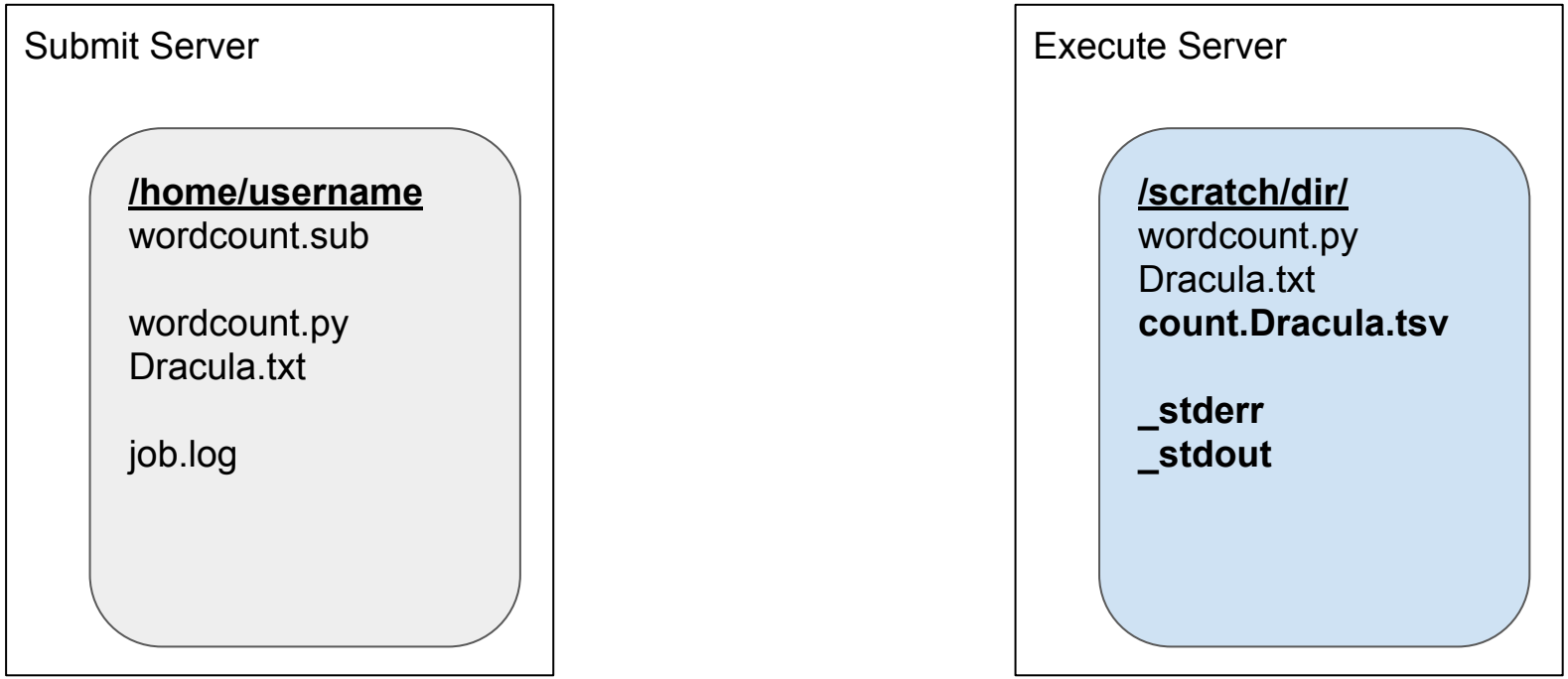


What Happens to the Job - Idle

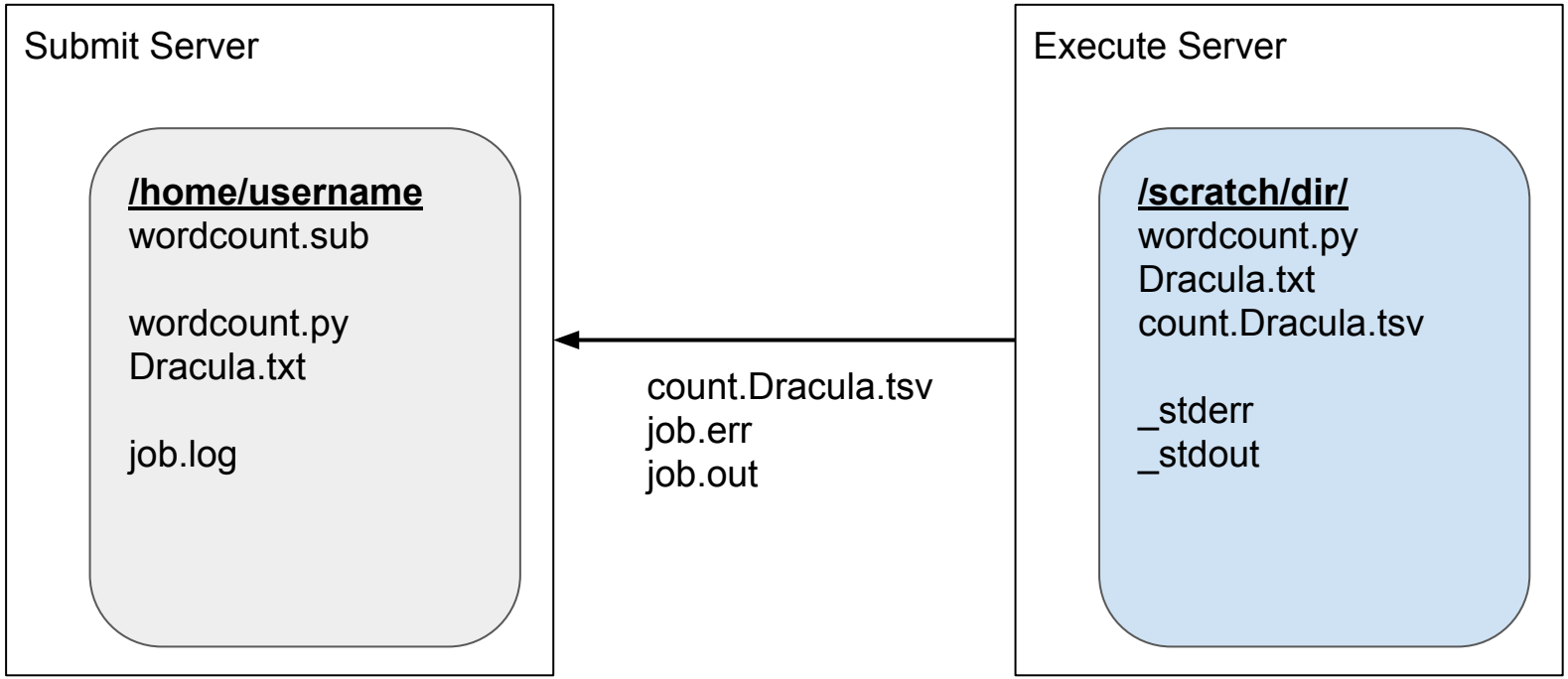
What Happens to the Job - Starting



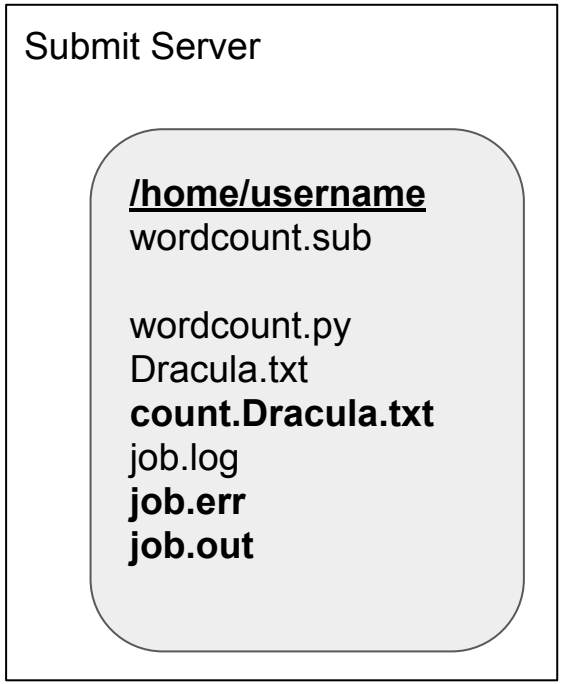
What Happens to the Job - Running



What Happens to the Job - Completing



What Happens to the Job - Done



Output From Jobs

- **Output from script**
  - counts.Dracula.tsv
- **“Standard” output and error**
  - logs/1234.0.out, logs/1234.0.err
  - Look here for errors and messages!
- **HTCondor Log**
  - logs/1234.0.log
  - Gives information about how long the job ran, and number of CPUs, memory, disk used
  - Check this if a job goes on hold!

```
Part. Resources: Usage Request Allocated
Cpus           :       1       1
Disk (KB)      : 853   524288  2074303
Memory (MB)    : 20     512     512
```

Multiple Jobs

Submitting Multiple Jobs

Replace unchanging values...

```
executable = wordcount.py
arguments = Dracula.txt

transfer_input_files = Dracula.txt

...

queue 1
```

...with variables

```
executable = wordcount.py
arguments = $(book)

transfer_input_files = $(book)

...

queue book from ...
```

Submitting Multiple Jobs

And provide a list of values to provide for each job.

```
executable = wordcount.py
arguments = $(book)

transfer_input_files = $(book)

...

queue book from book.list
```

```
Alice in Wonderland.txt
Dracula.txt
Huckleberry_Finn.txt
...
```

Let’s submit multiple jobs!

1. List all the book text files in a file called “`book.list`”
   ```
   ls *.txt > book.list
   ```

2. Change lines in “`multi-wordcount.sub`”
   a. `arguments = $(book)`
   b. `transfer_input_files = $(book)`
   c. `queue name from book.list`

3. Submit the jobs to see what happens. Try using `` `condor_watch_q` `` to view progress.

Next Steps for Using CHTC

Building Up a Larger Job Submission

Try to get ONE job running

- Troubleshoot errors and problems
- Check memory/disk requirements

Do a small scale test of 5-10 jobs

- Check memory + disk requirements *again*

Run full-scale set of jobs

TESTING IS KEY!!

ALWAYS run test jobs before submitting many jobs at once.

Be Responsible

- These resources are shared -- be a good citizen! Actions you take may affect your classmates ability to get their work done.
- Don’t run programs directly on the submit server.
- **Your personal quota (in /home and /home/groups) is 20GB. For your independent projects, if you want to use more than 10-15GB of data, please contact us or let Professor Gillett know.**
- Ask questions if you aren’t sure about something.

Get Help

We work for CHTC as Research Computing Facilitators. It's our job to answer questions and help people get started with computing at CHTC.

Email us! chtc@cs.wisc.edu

Or come to office hours online - go.wisc.edu/chtc-officehours

- Tuesdays: 10:30am - 12pm
- Thursdays: 3 - 4:30pm

http://chtc.cs.wisc.edu/uw-research-computing/get-help

Contact Us

chtc@cs.wisc.edu

http://chtc.cs.wisc.edu