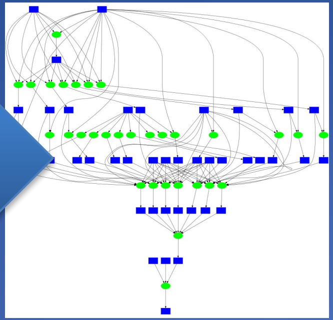




Scalable Application Design: Pitfalls and Possibilities

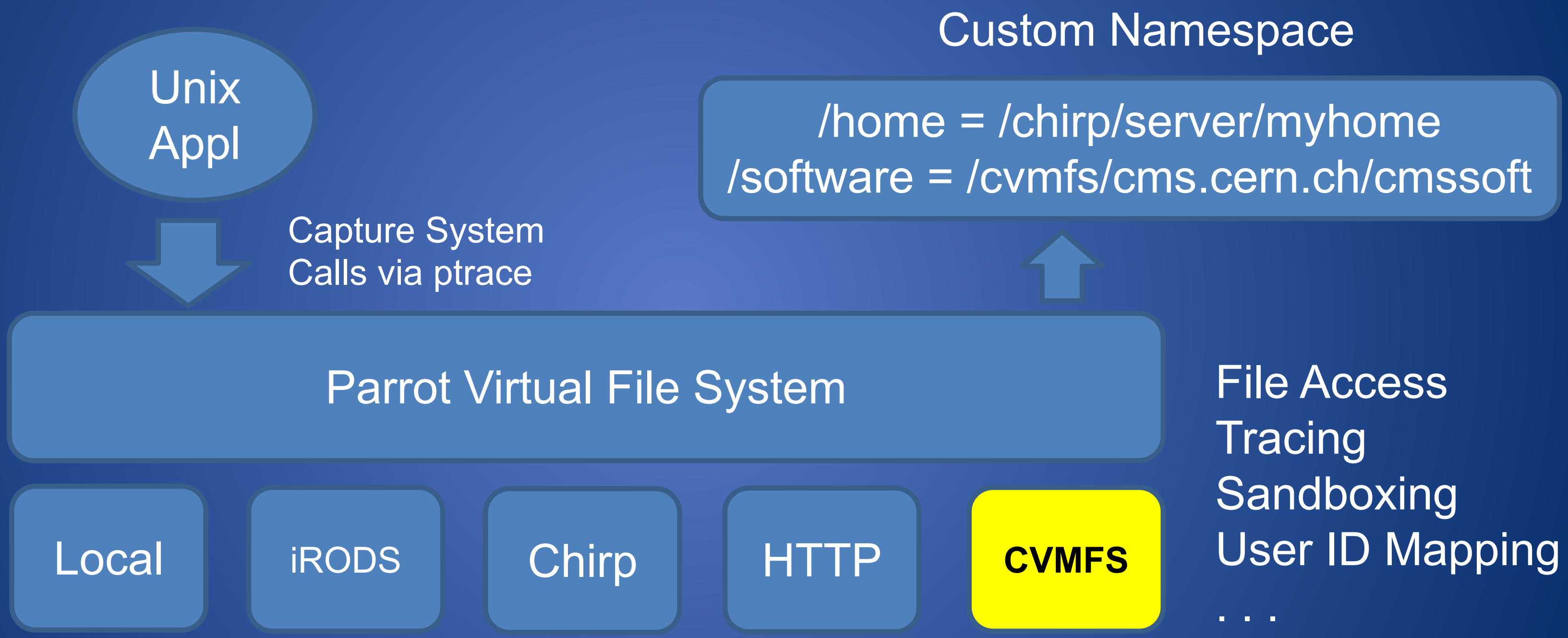
Prof. Douglas Thain, University of Notre Dame

http://www.nd.edu/~dthain
dthain@nd.edu
@ProfThain

The Cooperative Computing Lab

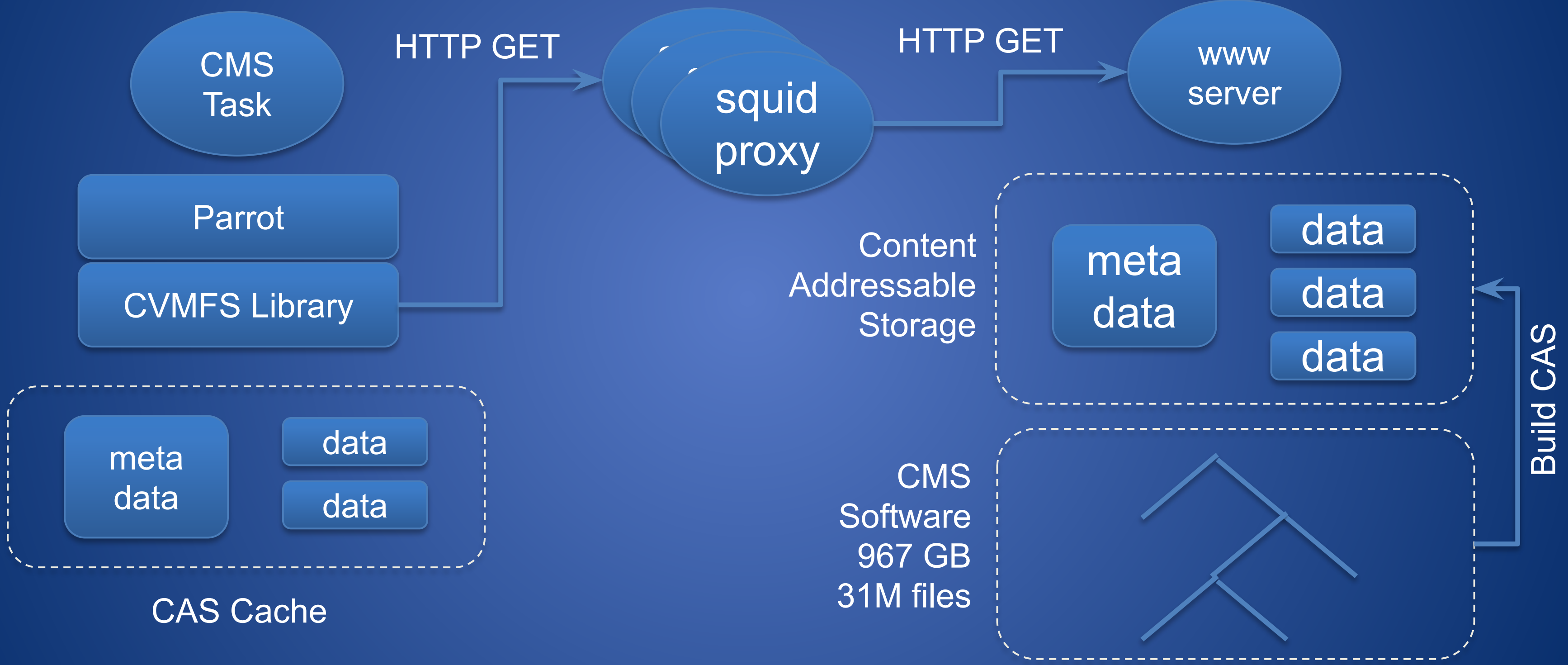
- We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.
- We *operate computer systems* on the O(10,000) cores: clusters, clouds, grids.
- We *conduct computer science* research in the context of real people and problems.
- We *develop open source software* for large scale distributed computing.

Parrot Virtual File System



Douglas Thain, Christopher Moretti, and Igor Sfiligoi, **Transparently Distributing CDF Software with Parrot**, *Computing in High Energy Physics*, pages 1-4, February, 2006.

Parrot + CVMFS



Jakob Blomer, Predrag Buncic, Rene Meusel, Gerardo Ganis, Igor Sfiligoi and Douglas Thain, **The Evolution of Global Scale Filesystems for Scientific Software Distribution**, *IEEE/AIP Computing in Science and Engineering*, **17**(6), pages 61-71, December, 2015. DOI: 10.1109/MCSE.2015.111

From the scientist's perspective…

It took ~~a while~~ (most of a year) but now I have my code written, installed, debugged, calibrated, and verified on my laptop.

Now I want to run at a scale 1000x larger by using a cluster, cloud, grid, or whatever you computer people are calling it today.

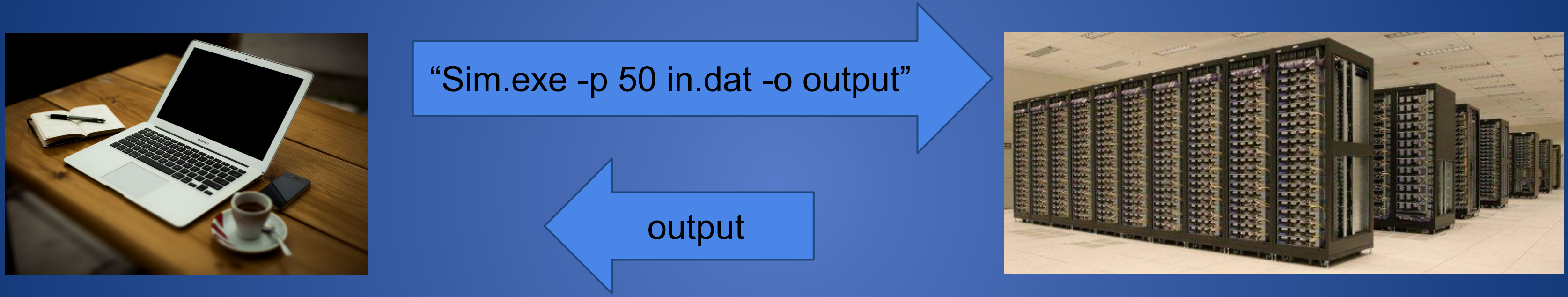
There is no way you are going to convince me to re-write this valuable program in order to run on your crazy cluster / OS / framework!

On my laptop…

"sim.exe –p 50 in.dat -o output.dat"

Output!

What could go wrong?

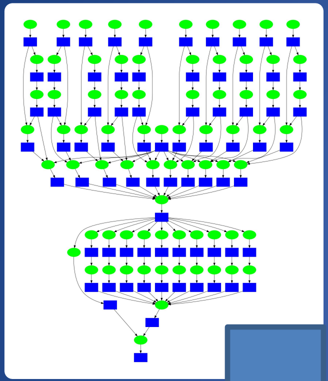


- The Software Dependency Problem
- The Resource Sizing Problem
- The Job Sizing Problem

Outline

- The Laptop Perspective
- Expressing Scalable Applications
- End User Challenges:
  - The Software Dependency Problem
  - The Resource Sizing Problem
  - The Job Sizing Problem
- Lessons Learned

Makeflow = Make + Workflow



- Provides portability across batch systems.
- Enables parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.
- Transactional semantics for job execution.

Makeflow

Local | HTCondor | Torque | Work Queue | Amazon

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

Workflow Language Evolution

Classic "Make" Representation

```
output.5.txt : input.txt mysim.exe
     mysim.exe –p 10 input.txt > output.5.txt
```

Tim Shaffer
(tshaffe1@nd.edu)

JSON Representation of One Job

```
 {
"command"  :  "mysim.exe –p 10 input.txt > output.5.txt",
"outputs" :  [ "output.5.txt" ],
"inputs"   :  [ "input.dat", "mysim.exe" ]
}
```

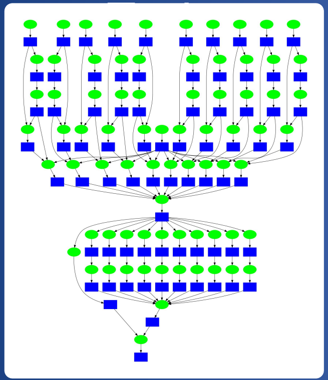
JX (JSON + Expressions) for Multiple Jobs

```
 {
"command"  :  "mysim.exe –p " + x*2 + " input.txt > output." + x + " .txt",
"outputs" :  [ "output" + x + "txt" ],
"inputs"   :  [ "input.dat", "mysim.exe" ]
} for x in [ 1, 2, 3, 4, 5 ]
```

Makeflow Shapes a Workflow

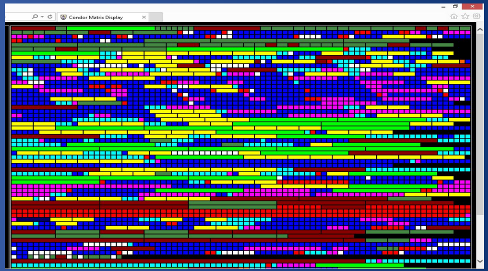
Concurrency
and Policy Control

Millions of



Make
Flow

Cluster or Cloud

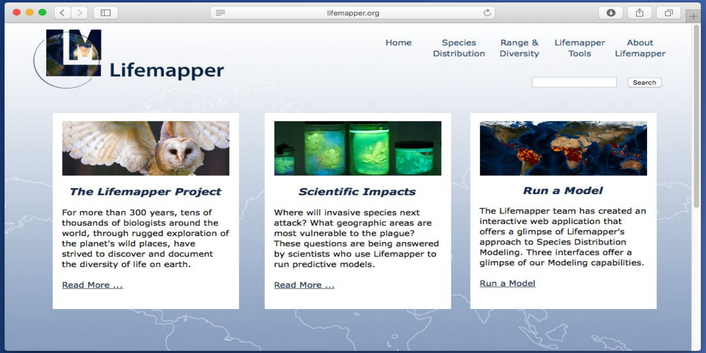


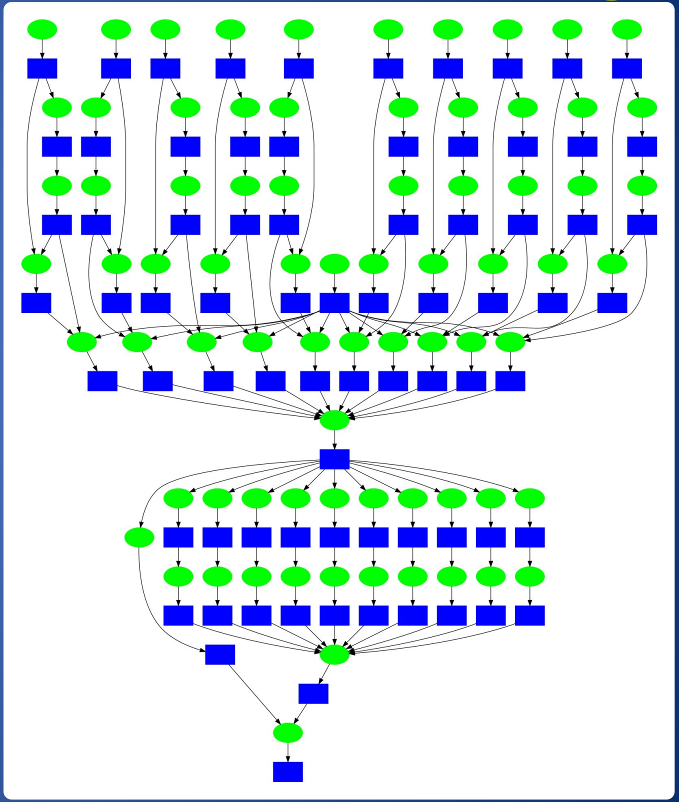
Precise
Cleanup

Transaction Log

Performance
Monitoring

Example: Species Distribution Modeling





Full Workflow:

12,500 species

x 15 climate scenarios

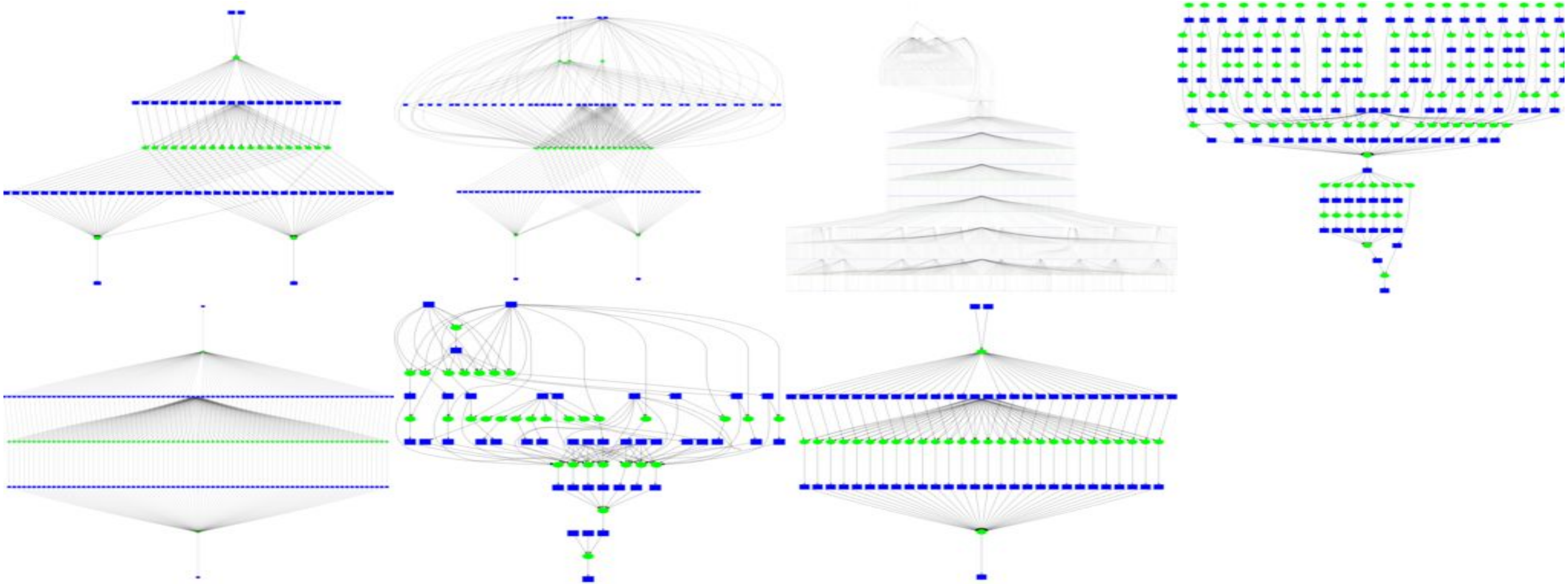
x 6 experiments

x 500 MB per projection

= 1.1M jobs, 72TB of output

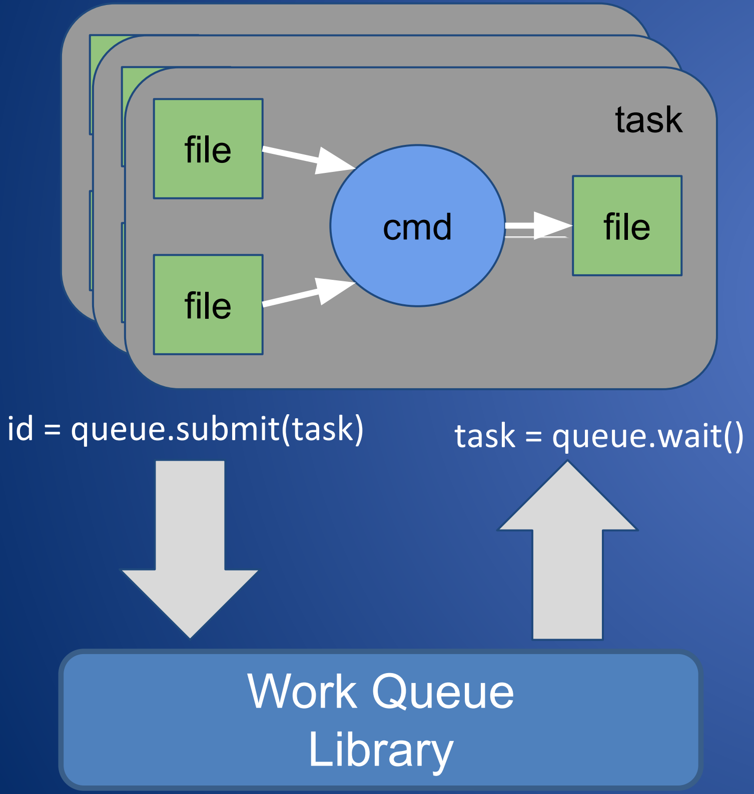
Small Example: 10 species x 10 expts

More Examples



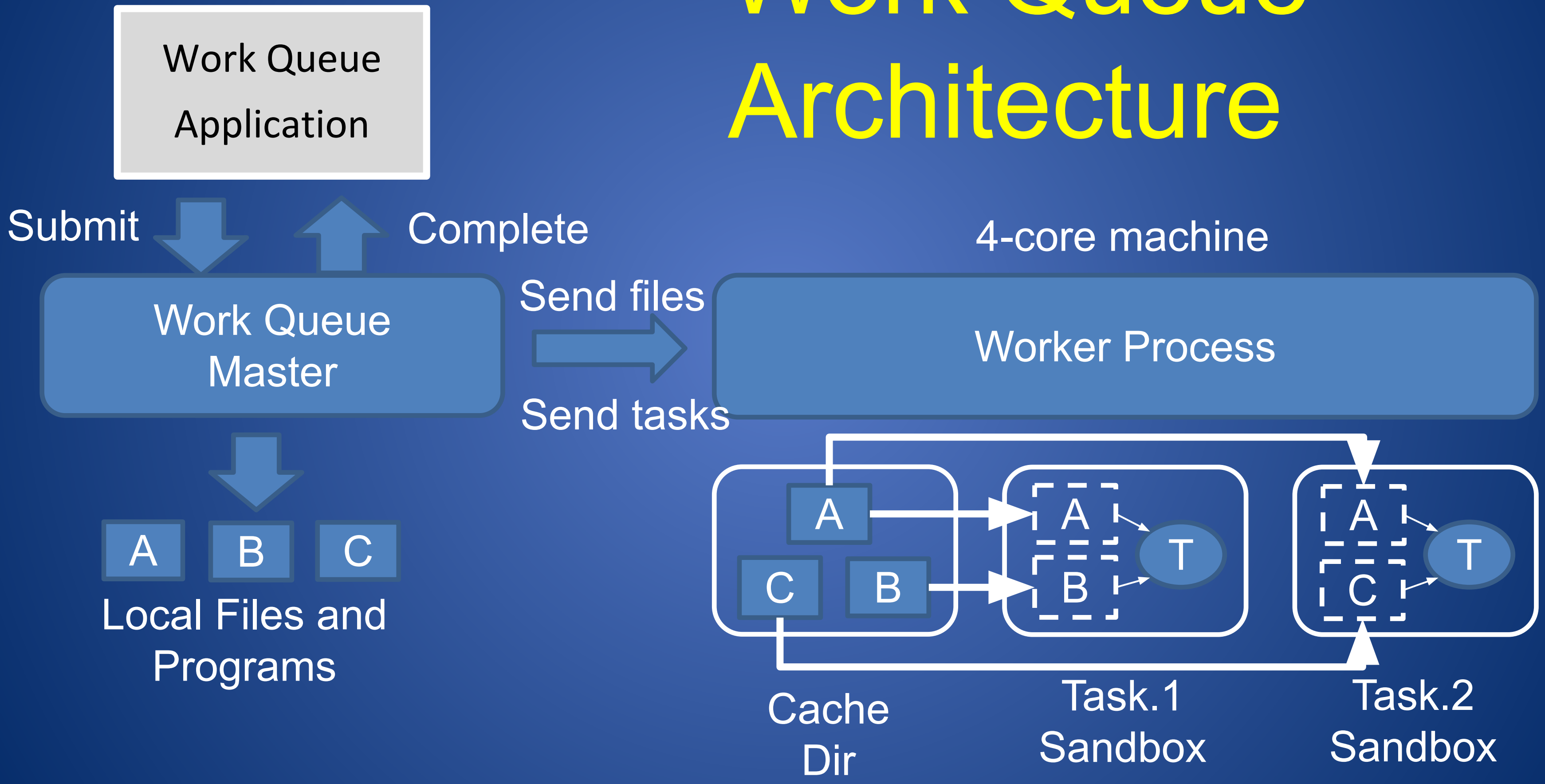
http://github.com/cooperative-computing-lab/makeflow-examples

Work Queue API

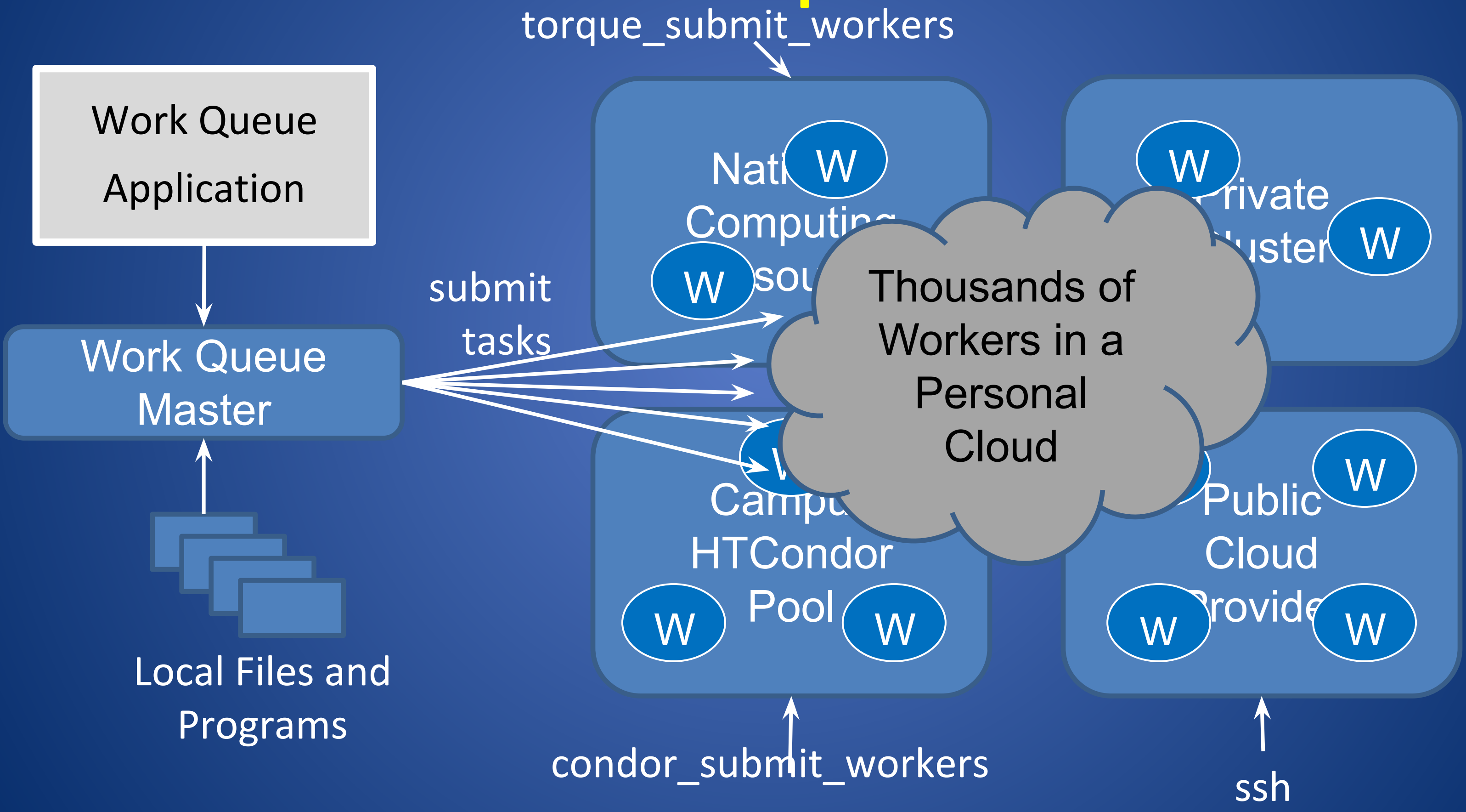



```
queue = WorkQueue(port)

for x in 1..100:
    task = Task(command)
    # add some more details...
    taskid = queue.submit(task)

while not queue.empty():
    task = queue.wait(5)
    if task:
        # deal with output, submit more
```

Work Queue Architecture

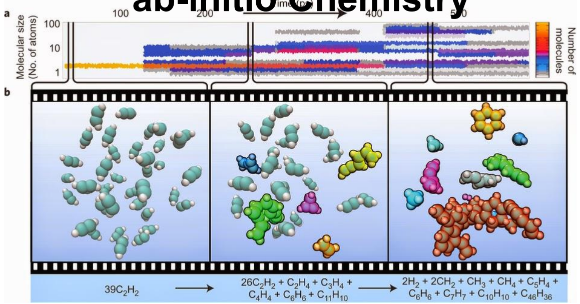


Harness Multiple Resources

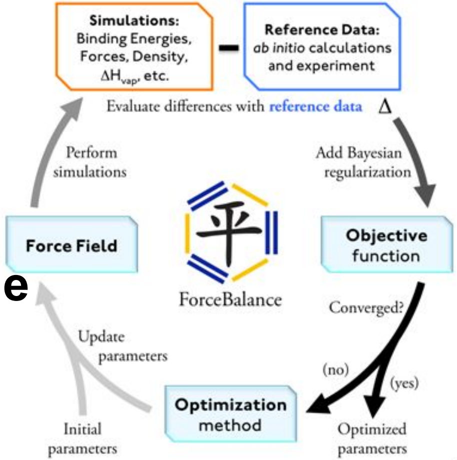


Some Work Queue Applications

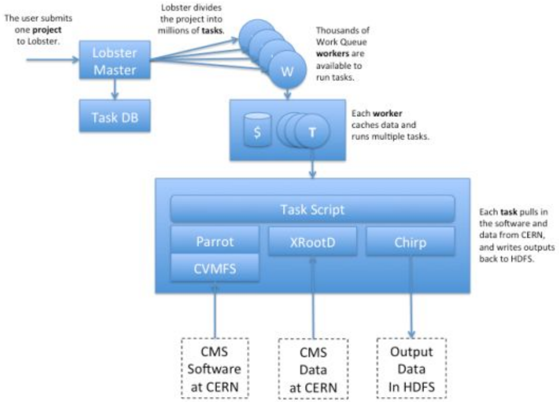
Nanoreactors
ab-initio Chemistry



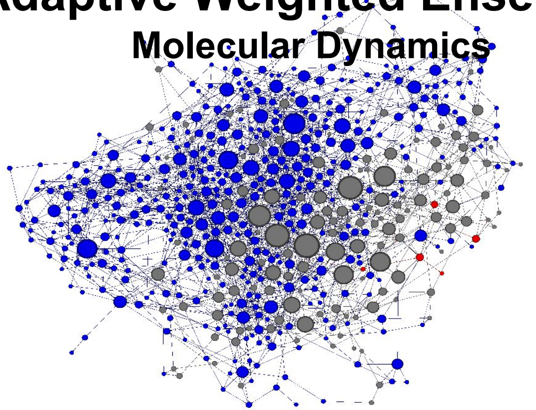
ForceBalance
FF Optimization



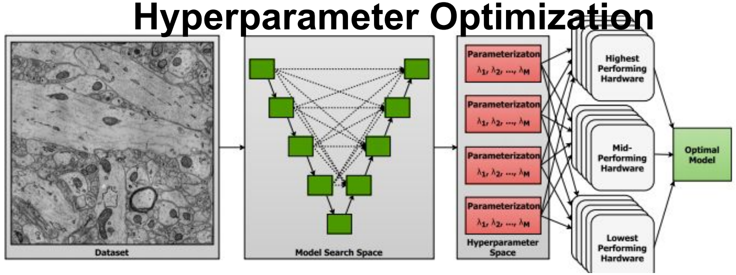
Lobster
CMS Data Analysis



Adaptive Weighted Ensemble
Molecular Dynamics



SHADHO
Hyperparameter Optimization



And now the bad news...

Simple questions that are hard to answer at scale:

- What software must be installed to run my application at multiple sites?
- How much memory do I need to run this task?
- How finely should I divide up my work?

Outline

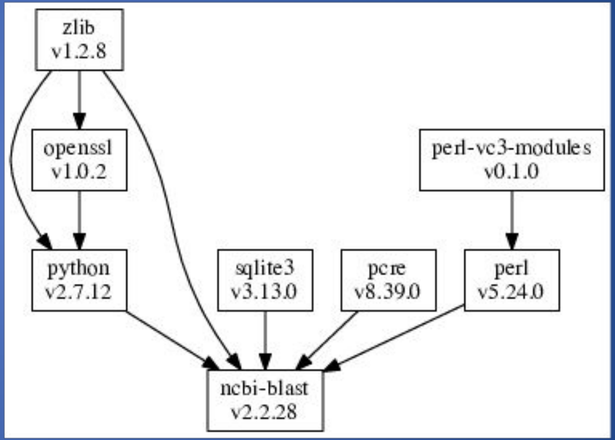
- The Laptop Perspective
- Expressing Scalable Applications
- End User Challenges:
  - The Software Dependency Problem
  - The Job Sizing Problem
  - The Data Splitting Problem
- Lessons Learned

Problem: Software Deployment

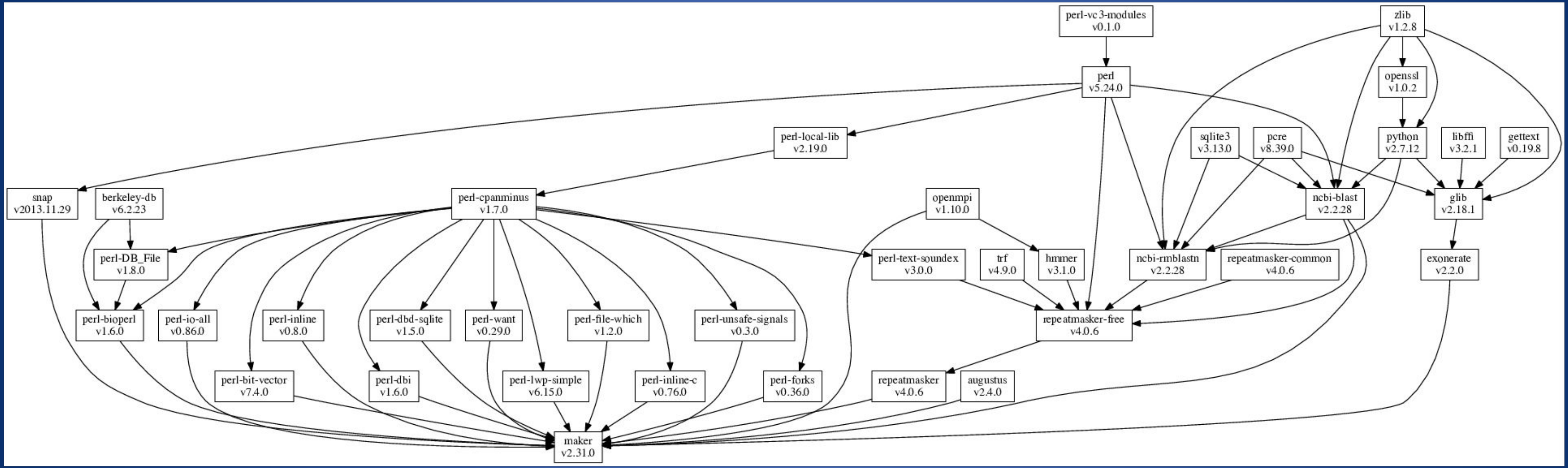
- Getting software installed on a new site is a big pain! The user (probably) knows the top level package, but doesn't know:
  - How they set up the package (sometime last year)
  - Dependencies of the top-level package.
  - Which packages are system default vs optional
  - How to import the package into their environment via PATH, LD_LIBRARY_PATH, etc.

Typical User Dialog Installing BLAST

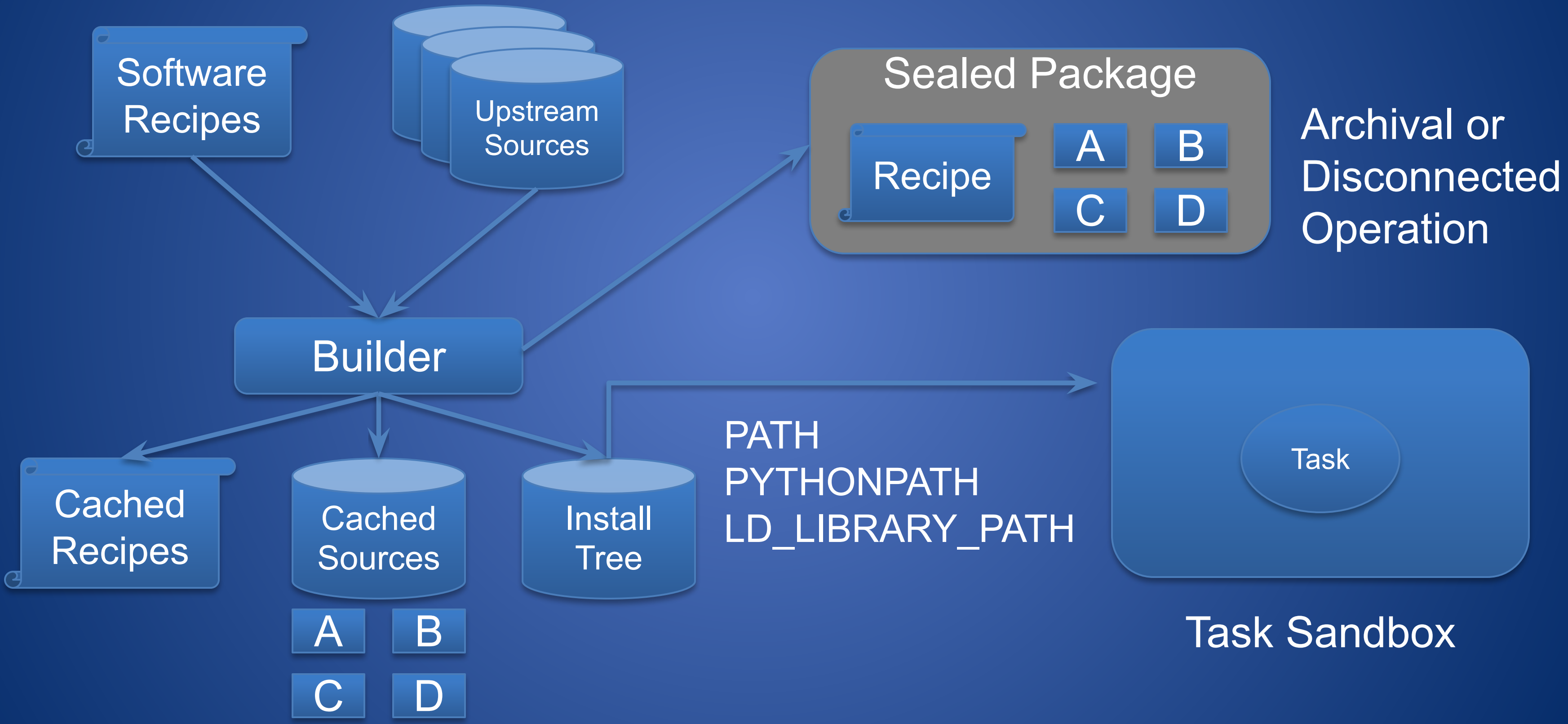
"I just need BLAST."
"Oh wait, I need Python!"
"Sorry, Python 2.7.12"
"Python requires SSL?"
"What on earth is pcre?"
"I give up!"



MAKER Bioinformatics Pipeline



VC3-Builder Architecture



"vc3-builder –require ncbi-blast"

..Plan: ncbi-blast => [,]
..Try: ncbi
....Plan: pe
....Try: pe
....could not
....Try: pe
....could not
....Try: pe
......Plan: p
......Try: p
......Success
....Success:
....Plan: py
....Try: pyt

(New Shell with Desired Environment)

```
bash$  which blastx
/tmp/test/vc3-root/x86_64/redhat6/ncbi-blast/v2.2.28/bin/blastx

bash$ blastx –help
USAGE
  blastx [-h] [-help] [-import_search_strategy filename]
   . . .

bash$ exit
```

....could not add any source for: python v2.006 => [v2.6.0,]
....Try: python => v2.7.12
......Plan: openssl => [v1.000,]
…………………
Downloading 'Python-2.7.12.tgz' from http://download.virtualclusters.org/builder-files
details: /tmp/test/vc3-root/x86_64/redhat6/python/v2.7.12/python-build-log

Problem: Long Build on Head Node

- Many computing sites limit the amount of work that can be done on the head node, so as to maintain quality of service for everyone.
- Solution: Move the build jobs out to the cluster nodes.   (Which may not have network connections.)
- Idea: Reduce the problem to something we already know how to do: Workflow!

vc3-builder

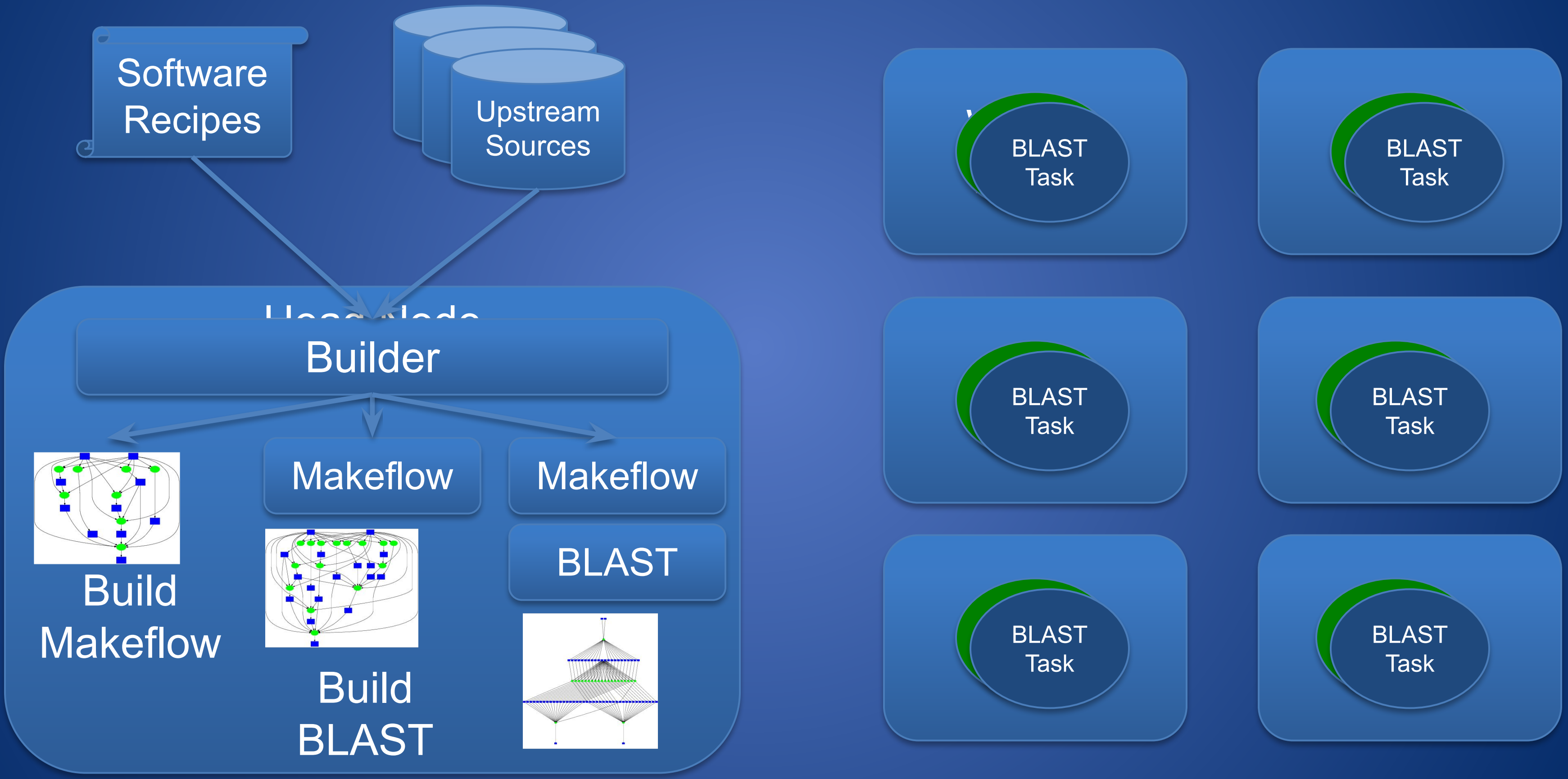
--require makeflow

--require ncbi-blast

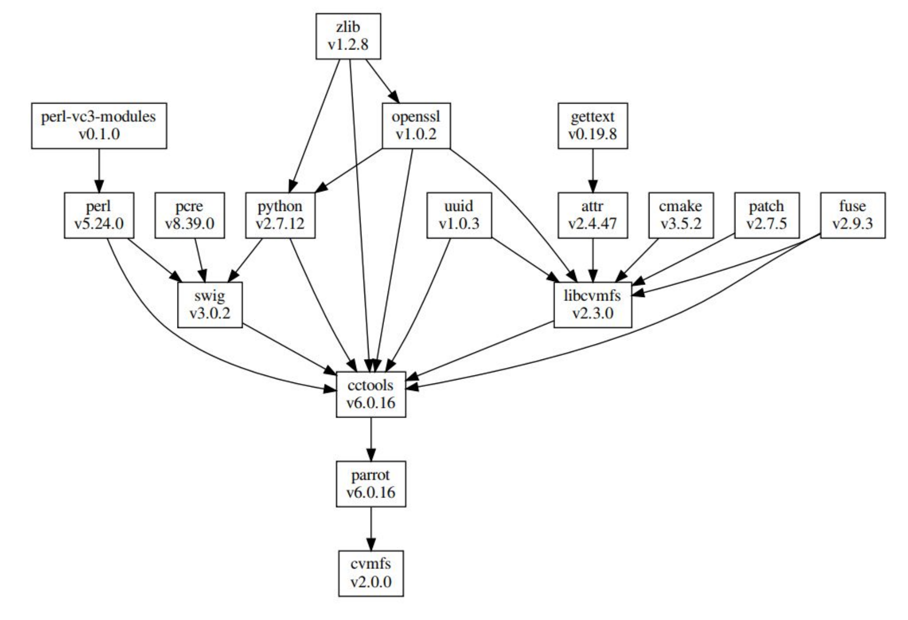
--

makeflow –T condor blast.mf

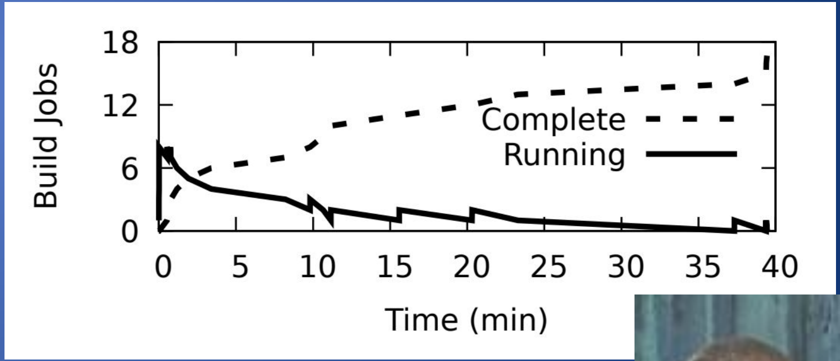
Bootstrapping a Workflow



What About CVMFS?



Use VC3-Builder to bootstrap from perl to Parrot + CVMFS in order to access your global filesystem!



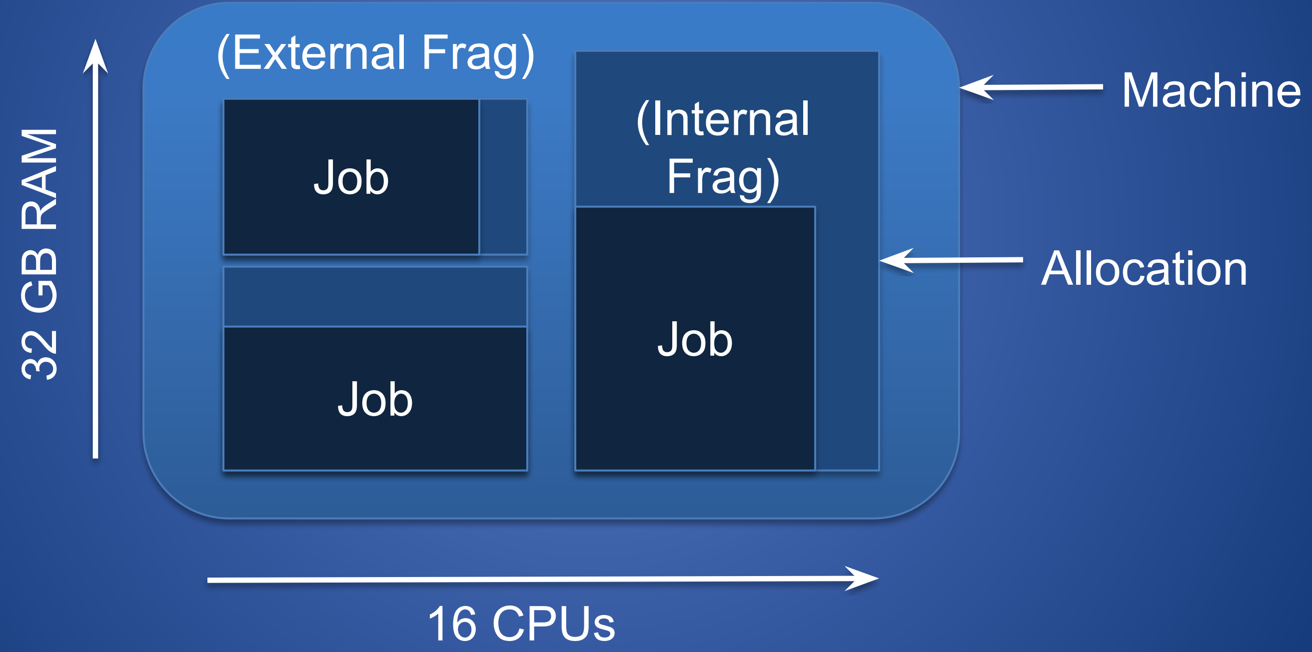
Benjamin Tovar, Nicholas Hazekamp, Nathaniel Kremer-Herman, and Douglas Thain,
Automatic Dependency Management for Scientific Applications on Clusters,
IEEE International Conference on Cloud Engineering (IC2E) , April, 2018.
DOI: 10.1109/IC2E.2018.00026

Ben Tovar
btovar@nd.edu

Outline

- The Laptop Perspective
- Expressing Scalable Applications
- End User Challenges:
  - The Software Dependency Problem
  - The Resource Sizing Problem
  - The Job Sizing Problem
- Lessons Learned

The Resource Sizing Problem



Client vs. Resource Provider

Client selects allocation:
- Too big? Wasted resources.
- Too small? Job fails, retry.

Provider places the allocation:
- Too big? Get paid.
- Too small? Still get paid!
- Scheduling is not the client's problem.

Client

Provider

"Slicing the Infinite Cake"

Allocations Too Big

Allocations Just Right: 2x Throughput

How do we know how big?

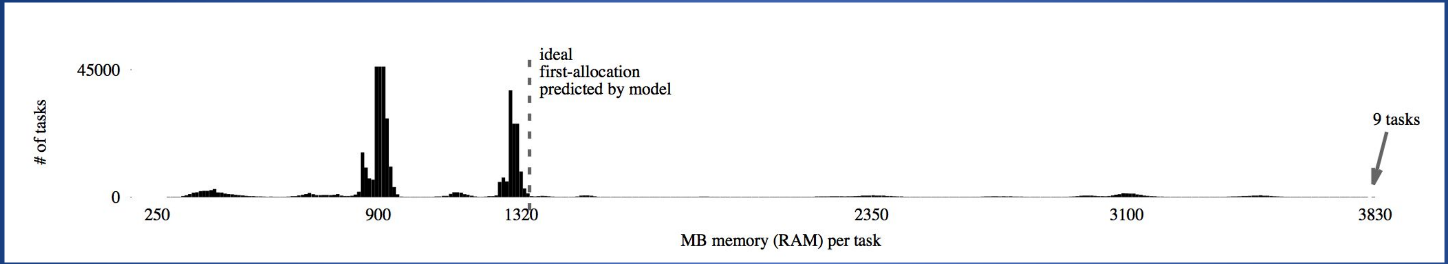

```
"command": "python task.py parameters.json",
"taskid":             "195",
"user":               "mfrohike",
"category":           "ttZ_mAODv2",
"executable_type":    "dynamic",
"monitor_version":    "6.0.0.0bdd9ca9",
"exit_status":        143,
"exit_type":          "limits",
"limits_exceeded":    { "memory": [2000,"MB"] },
"start":              [1454163721,"s"],
"end":                [1454165828,"s"],
"wall_time":          [2107,"s"],
"cpu_time":           [2066,"s"],
"cores":              2,
"cores_avg":          0.98,
"concurrent_procs":   8,
"total_procs":        273,
"virtual_memory":     [2255,"MB"],
"memory":             [2142,"MB"],
"swap_memory":        [0,"MB"],
"bytes_read":         [2274265095,"MB"],
"bytes_written":      [104022016,"MB"],
"bytes_sent":         [0,"MB"],
"bytes_received":     [0,"MB"],
"bandwidth":          [0,"Mbps"],
"total_files":        1175,
"disk":               [104,"MB"]
```

Suppose that you run 1M
analysis jobs that are all the same.

What would be the distribution of memory consumption across all jobs?

Impulse? Gaussian? Poisson?

Surprise: Complex Distributions!



How to pick the first allocation?

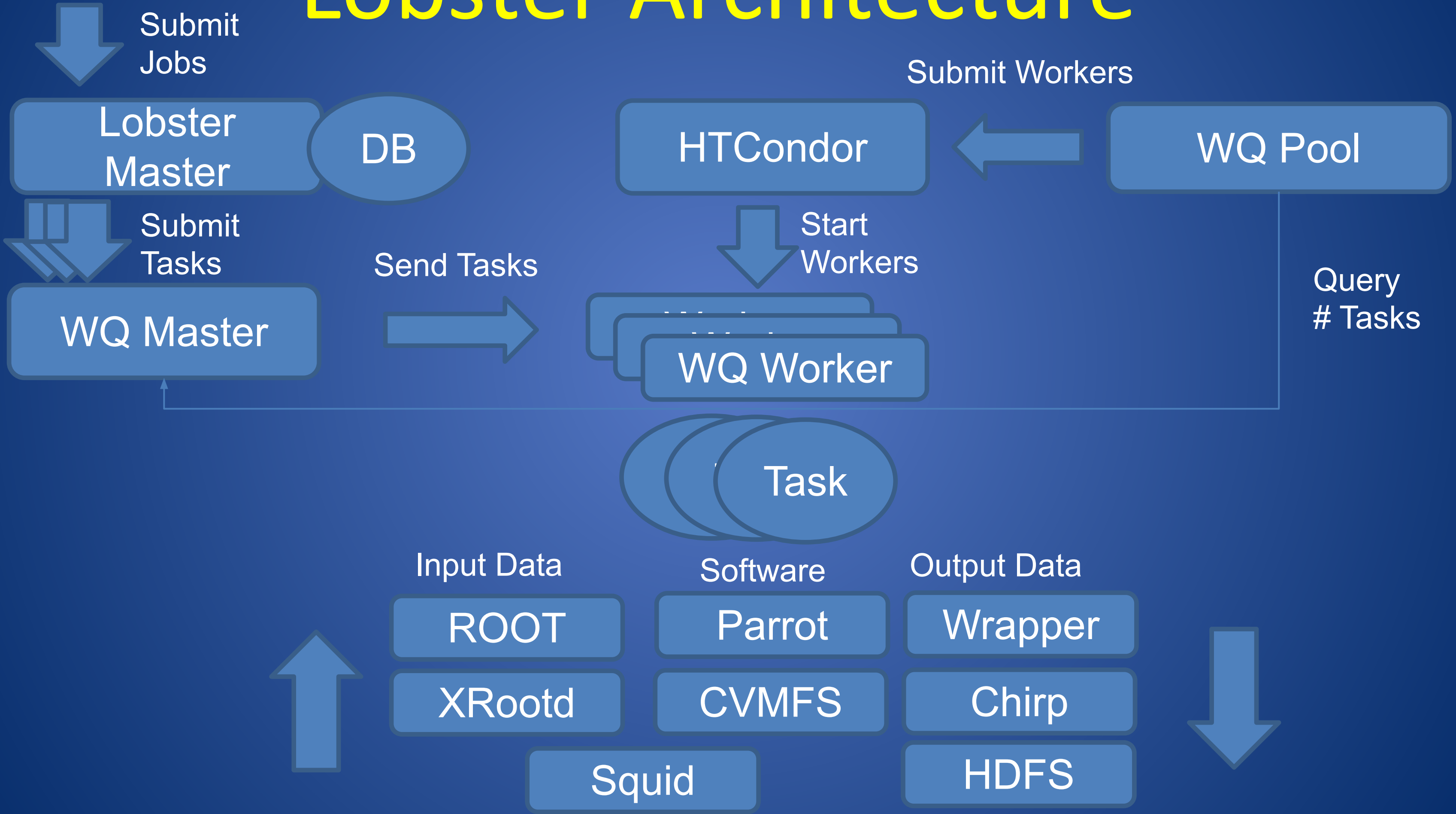
Ben Tovar says:
Minimize probability of first attempt succeeding + fallback succeeding, weighted by resources.

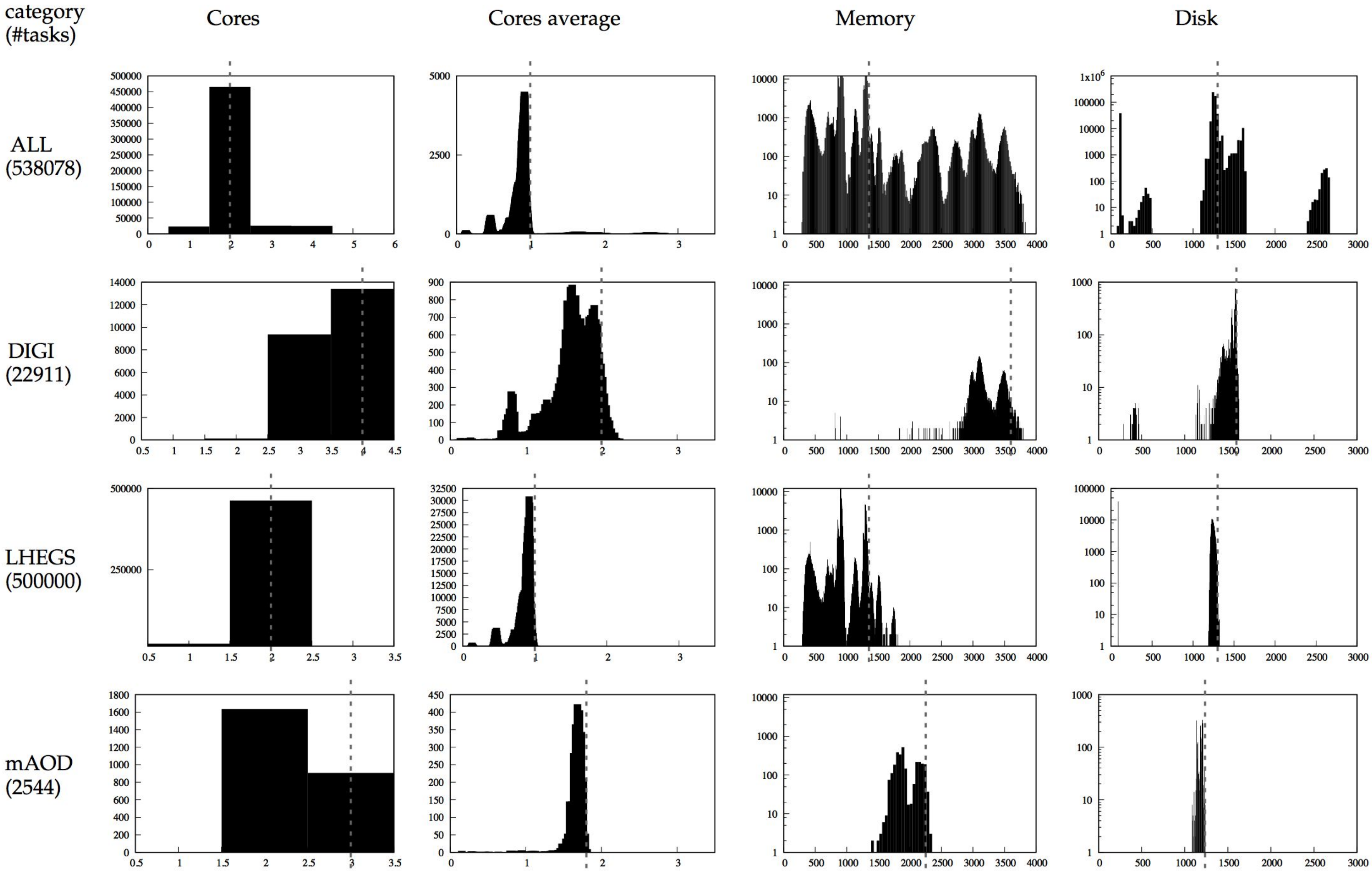
$$
\begin{aligned}
\mathrm{E}[\text{waste}(\mathrm{r},\tau,a_1)] &= \int_0^\infty \Bigg(\underbrace{\int_0^{a_1} (a_1 - \mathrm{r})\tau p(\mathrm{r},\tau)\mathrm{dr}}_{\text{first-allocation succeds}} \\
&\quad + \underbrace{\int_{a_1}^{a_m} ((a_m + a_1 - \mathrm{r})\tau p(\mathrm{r},\tau)\mathrm{dr}}_{\text{final allocation succeds}} \Bigg) \mathrm{d}\tau \\
&= a_1 \underbrace{\int_{a_1}^{a_m} \int_0^\infty \tau p(\mathrm{r},\tau)\mathrm{d}\tau \mathrm{dr}}_{\text{mean wall-time for all tasks}} \\
&\quad + a_m \int_{a_1}^{a_m} \underbrace{\int_0^\infty \tau p(\tau|\mathrm{r})\mathrm{d}\tau}_{\text{mean wall-time taks w. peak r}} p(\mathrm{r})\mathrm{dr} \\
&\quad - \underbrace{\int_0^\infty \int_0^\infty \mathrm{r}\tau p(\mathrm{r},\tau)\mathrm{d}\tau \mathrm{dr},}_{\text{used resources}}
\end{aligned}
$$

Production Application: Lobster

- Lobster: High energy physics analysis workload harnesses heterogeneous non-dedicated resources at Notre Dame.
- 535,078 tasks run on 25,000 core cluster over several months with the resource monitor.
- Five categories of tasks identified by user: DIGI (22911), LHEGS(500K), mAOD (2544), RECO (11582)

Lobster Architecture





category (#tasks)
Cores
Cores average
Memory
Disk
ALL (538078)
DIGI (22911)
LHEGS (500000)
mAOD (2544)

Resource Selection Approaches

	naive		brute-force		min. waste	max. through
resource	max. peak	$P(0.95 > r)$	min. waste	max. throug.	Equation 2	Equation 3
first allocation						
cores (cores)	5	3	2	2	2	2
cores_avg (cores)	2.9	1.5	1	1	1	1
memory (MB)	3830	2416	1350	1350	1350	1350
disk (MB)	2657	1338	1300	1300	1300	1300
proportion of wasted resources per task						
cores	58%	34%	13%	13%	13%	13%
cores_avg	70%	48%	23%	23%	23%	23%
memory	72%	57%	32%	32%	32%	32%
disk	55%	16%	15%	15%	15%	15%
throughput normalized						
cores	1.00	1.58	2.18	2.18	2.18	2.18
cores_avg	1.00	1.74	2.69	2.69	2.69	2.69
memory	1.00	1.51	2.54	2.54	2.54	2.54
disk	1.00	1.88	1.91	1.91	1.91	1.91
percentage of tasks retried						
cores	0%	5%	9%	9%	9%	9%
cores_avg	0%	5%	7%	7%	7%	7%
memory	0%	5%	8%	8%	8%	8%
disk	0%	5%	6%	6%	6%	6%
overhead						
overhead (s)	—	—	0.78	0.83	0.07	0.06
538078 tasks read in 27.60 seconds						

Benjamin Tovar, Rafael Ferreira da Silva, Gideon Juve, Ewa Deelman, William Allcock, Douglas Thain, and Miron Livny, A Job Sizing Strategy for High-Throughput Scientific Workflows,
IEEE Trans Parallel Dist Sys, 29(2), pages 240-253, February, 2018. DOI: 10.1109/TPDS.2017.2762310

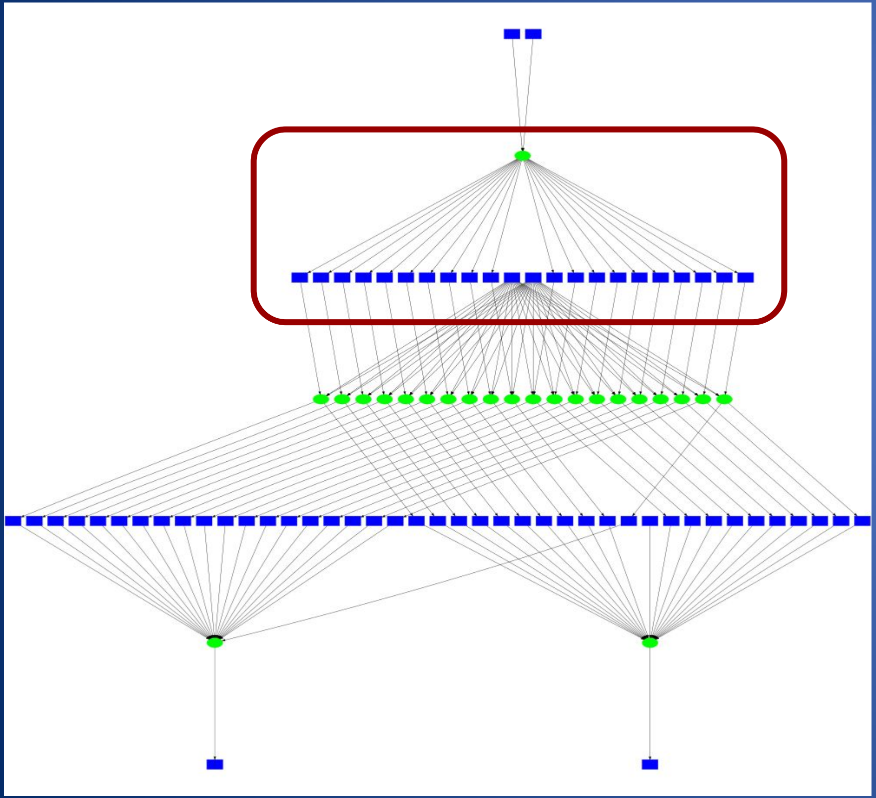
What's the upshot?

- By selecting first allocations appropriately, we double the throughput of the system while accepting a 9 percent task failure rate.
- This approach is applied entirely from the client side, without provider assistance.
- Same approach can be applied to any cluster/cloud/grid with simple techniques.

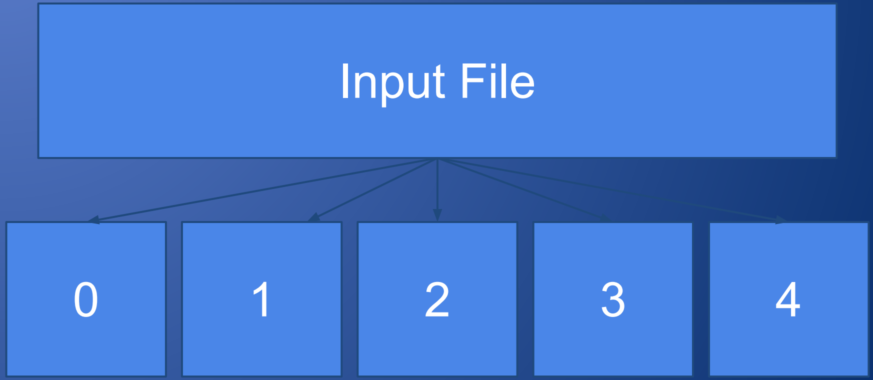
Outline

- The Laptop Perspective
- Expressing Scalable Applications
- End User Challenges:
  - The Software Dependency Problem
  - The Resource Sizing Problem
  - The Job Sizing Problem
- Lessons Learned

What’s Going on Here?



The user is starting off by splitting an input file into pieces in order to prepare for parallel tasks:



Static Job Splitting

User often makes a choice based on some rule of thumb without really understanding the tradeoffs:

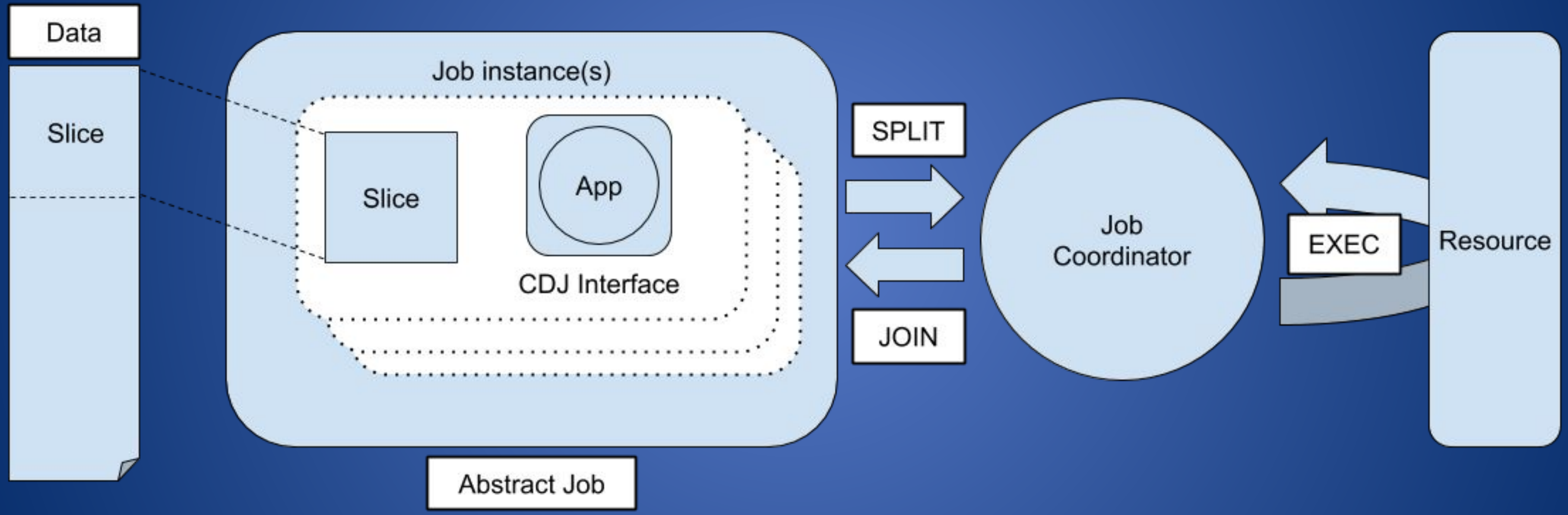
- Jobs too small: Overhead of splitting dominates cost of actually doing work!
- Jobs too large: Insufficient parallelism to get the job done in a timely way.

Difference between an acceptable choice and a bad one can be a factor of 100X in performance!

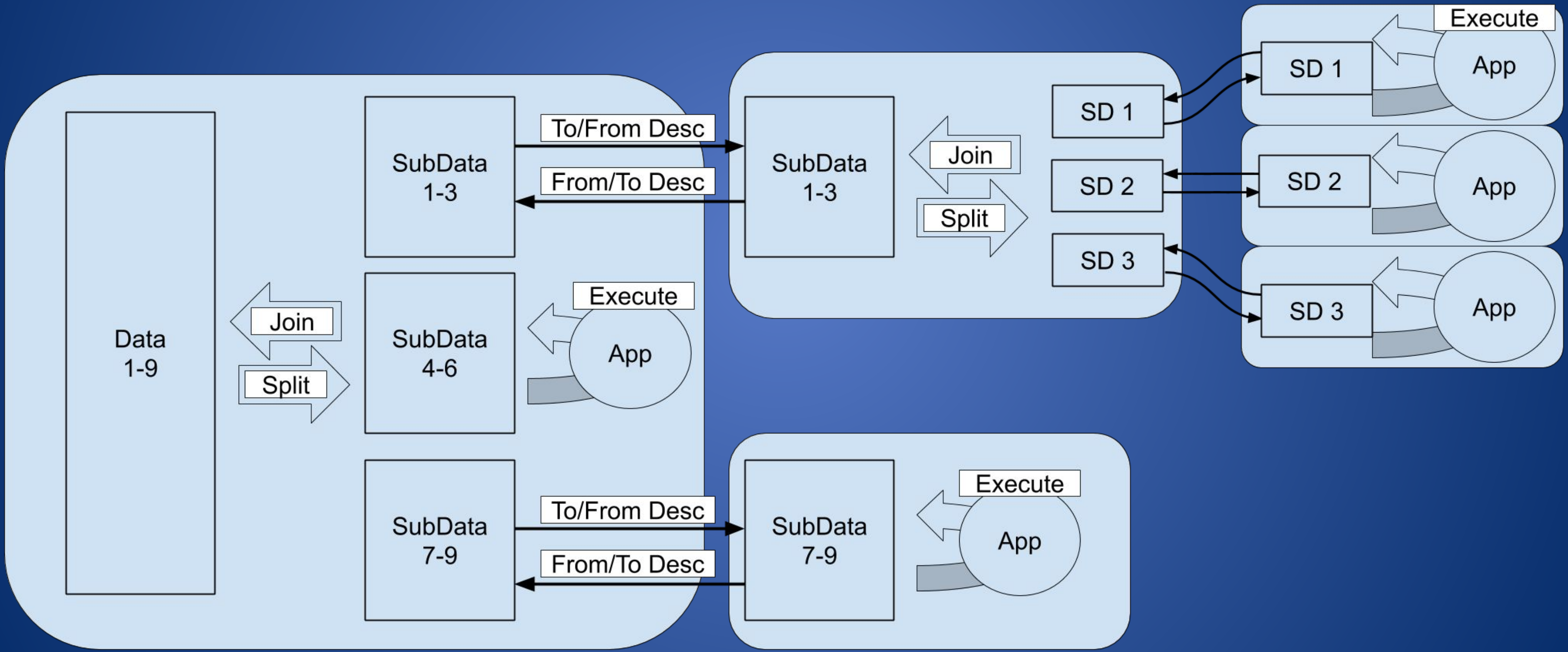
Continuously Divisible Jobs

- Defer work splitting until computational demand requires a new job.
- Instead of materializing physical files, keep track of data indices as “virtual files” to be materialized on demand. (Job still sees files.)
- Compute ideal job sizes based on observed performance properties of the system.

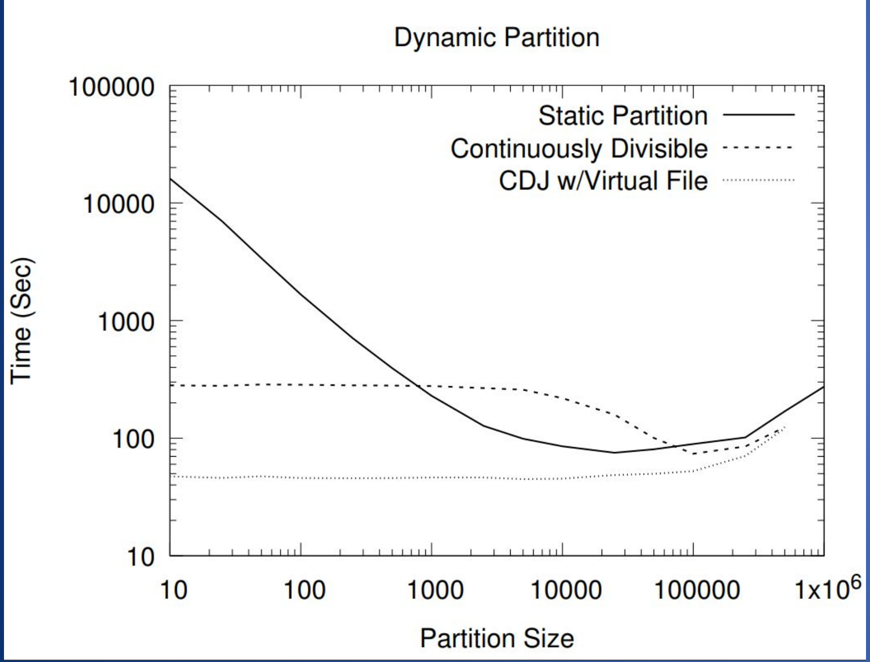
Continuously Divisible Jobs

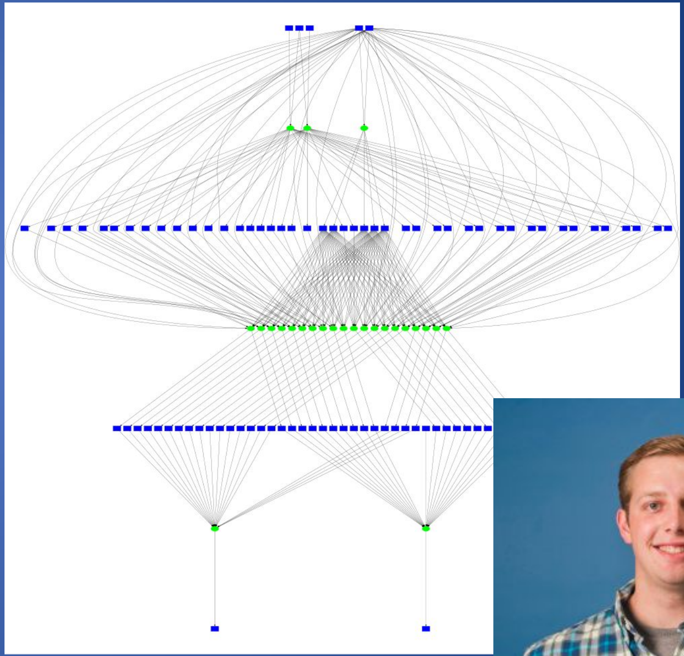


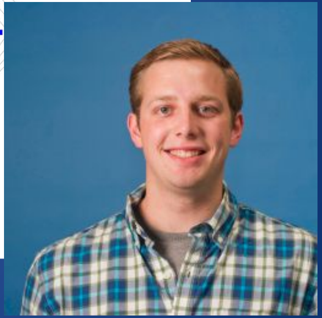
Hierarchical Job Division



Initial Results on BWA Workflow







Nick Hazekamp
nhazekam@nd.edu

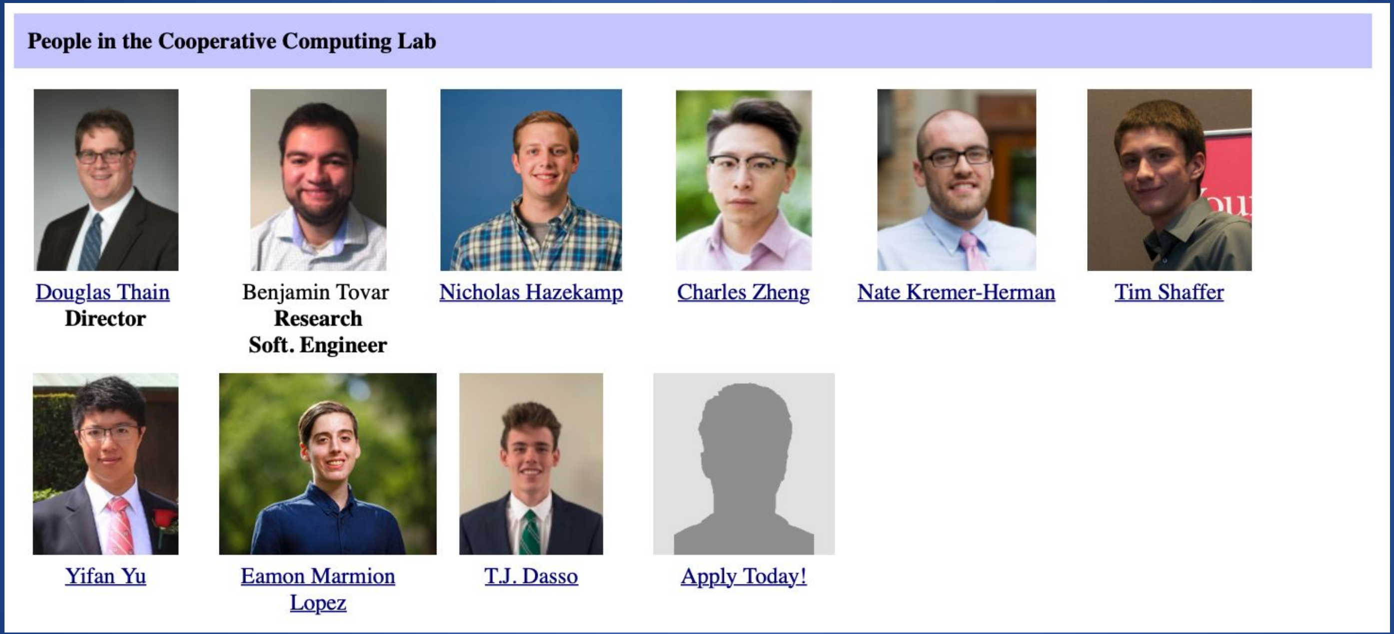
Outline

- The Laptop Perspective
- Expressing Scalable Applications
- End User Challenges:
  - The Software Dependency Problem
  - The Resource Sizing Problem
  - The Job Sizing Problem
- Lessons Learned

Thoughts and Lessons Learned

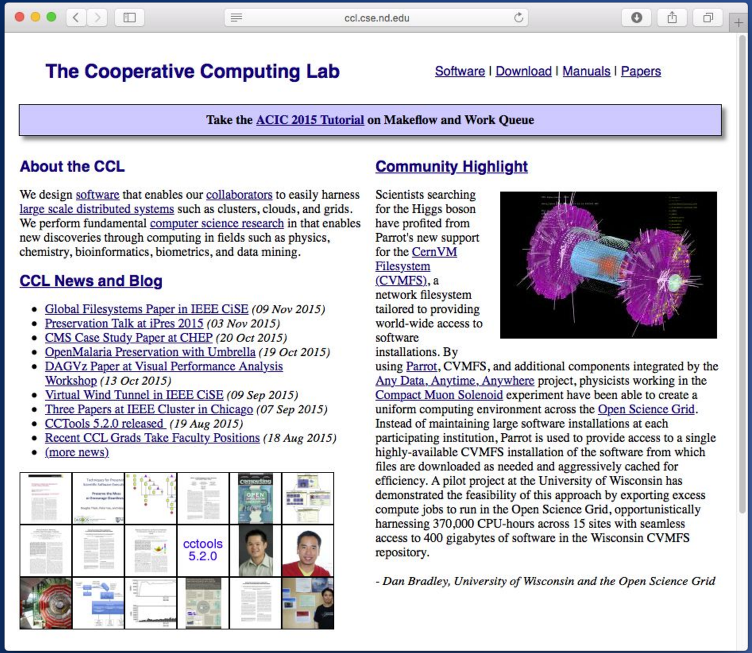
- Make software dependencies more explicit!
  - Proposed: Nothing should be available by default, all software should require an "import" step.
- Make resource consumption more visible!
  - The laconic nature of the shell hides too much about resource consumption.
- Users are poorly equipped to do performance tuning: don’t commit to decisions too early.

Acknowledgements



DE-SC0015711
VC3: Virtual Clusters for Community Computation

ACI-1642409
SI2-SSE: Scaling up Science on Cyberinfrastructure with the Cooperative Computing Tools



http://ccl.cse.nd.edu

@ProfThain