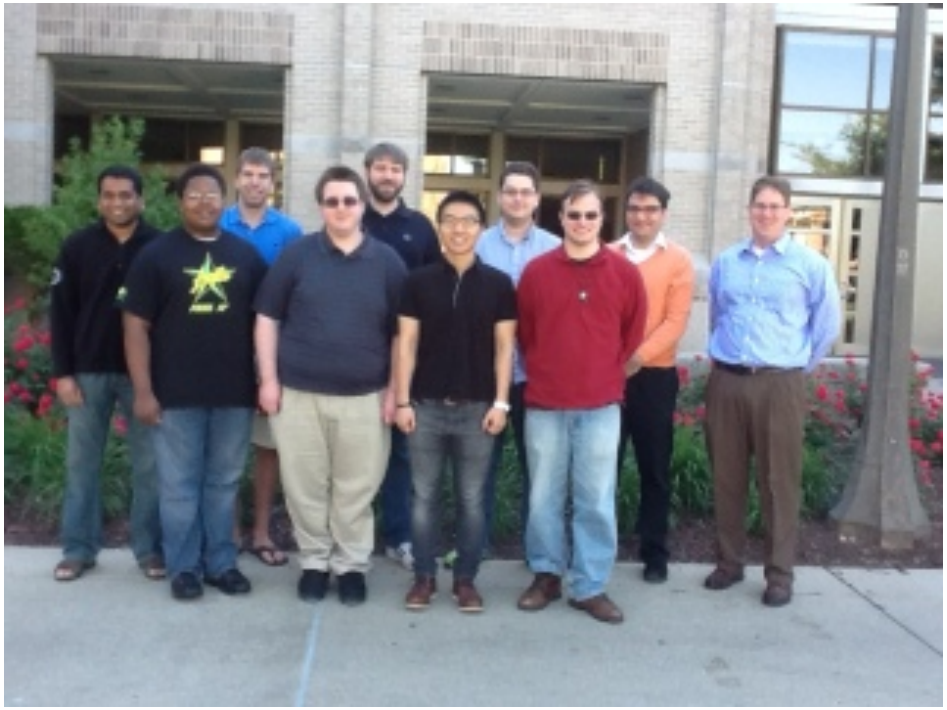
Lobster: Personalized Opportunistic Computing for CMS at Large Scale

Douglas Thain
(on behalf of the Lobster team)
University of Notre Dame

CVMFS Workshop, March 2015

The Cooperative Computing Lab

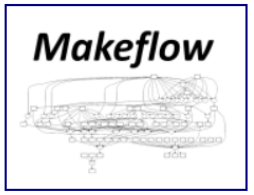
University of Notre Dame



http://www.nd.edu/~ccl

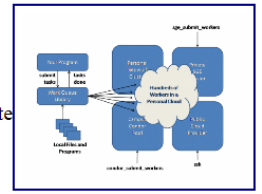
Makeflow

Makeflow is a workflow system for parallel and distributed computing that uses a language very similar to Make. Using Makeflow, you can write simple scripts that easily execute on hundreds or thousands of machines.



Work Queue

Work Queue is a system and library for creating and managing scalable master-worker style programs that scale up to thousands machines on clusters, clouds, and grids. Work Queue programs are easy to write in C, Python or Perl.

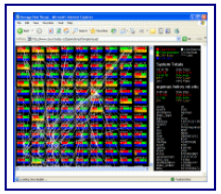


Parrot

Parrot is a transparent user-level virtual filesystem that allows any ordinary program to be attached to many different remote storage systems, including HDFS, iRODS, Chirp, and FTP.

Chirp

Chirp is a personal user-level distributed filesystem that allows unprivileged users to share space securely, efficiently, and conveniently. When combined with Parrot, Chirp allows users to create custom wide-area distributed filesystems.



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 *release open source software* for large scale distributed computing.

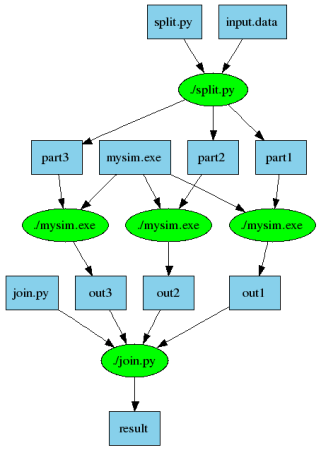
http://www.nd.edu/~ccl

Our Philosophy:

- Harness all the resources that are available: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware…
- Make simple things easy, and complex things possible.
- ***No special privileges required.***

Technology Tour

Makeflow = Make + Workflow



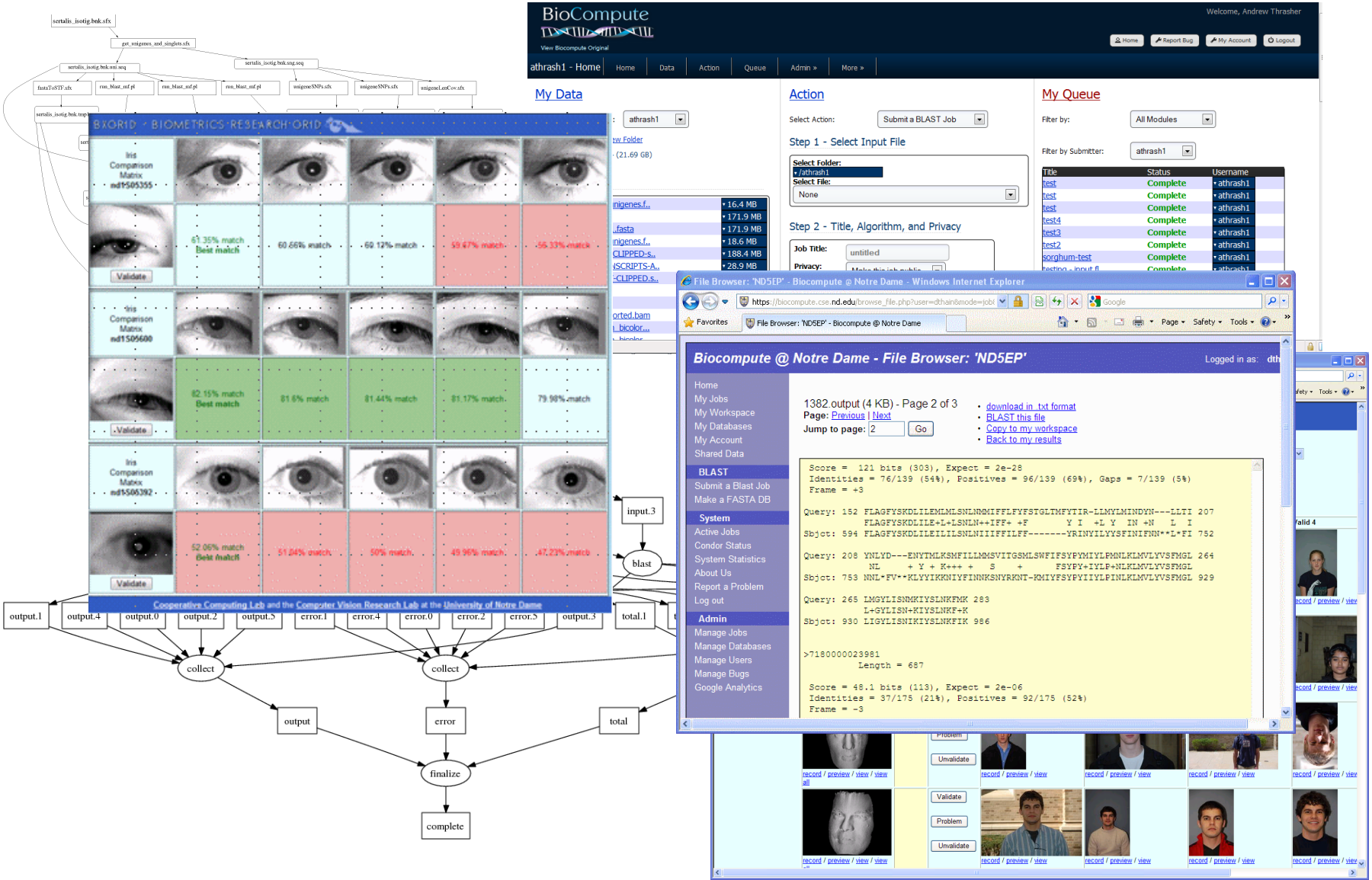
- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.

Makeflow

Local | Condor | SGE | Work Queue

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

Makeflow Applications

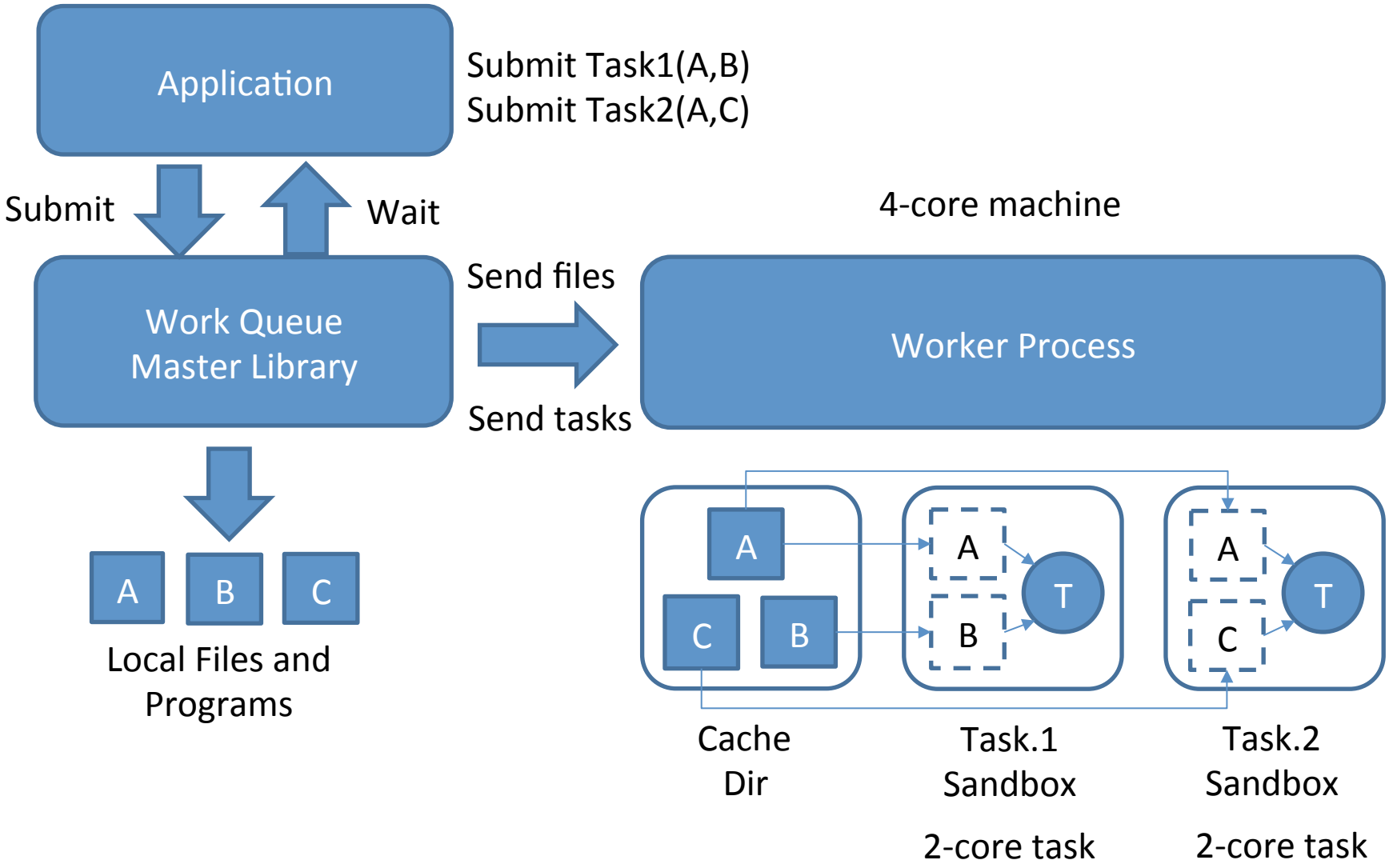


Work Queue Library

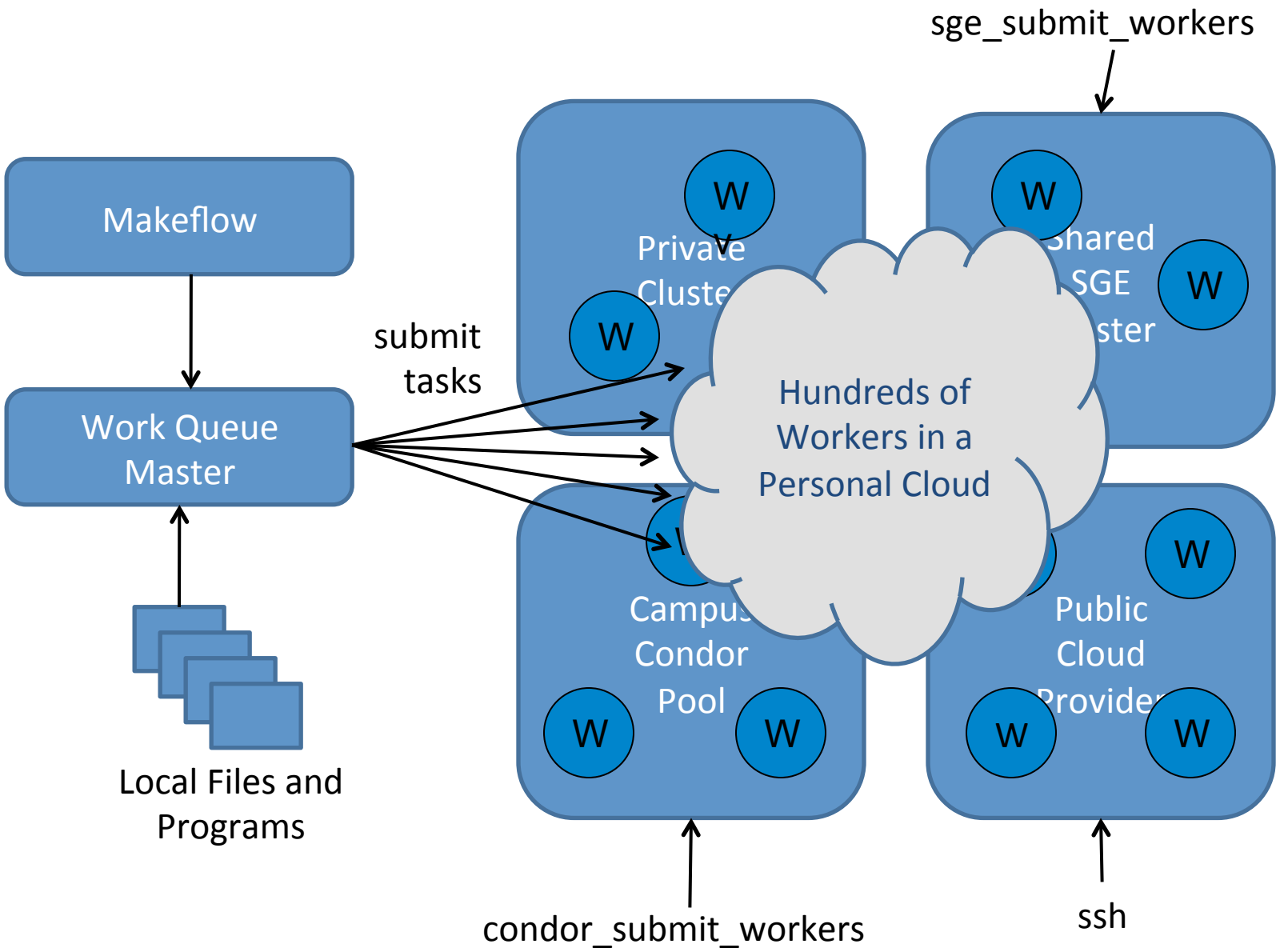
```
#include “work_queue.h”

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

Work Queue Architecture

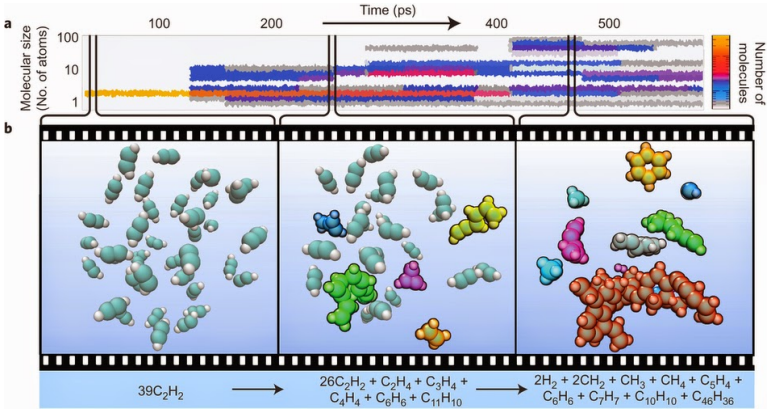


Run Workers Everywhere

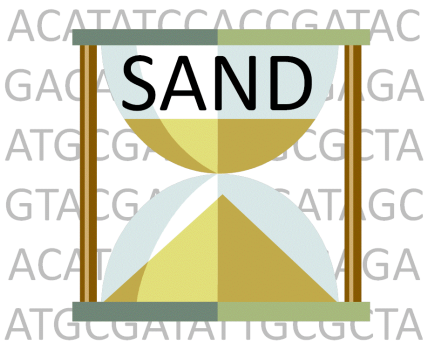


Work Queue Applications

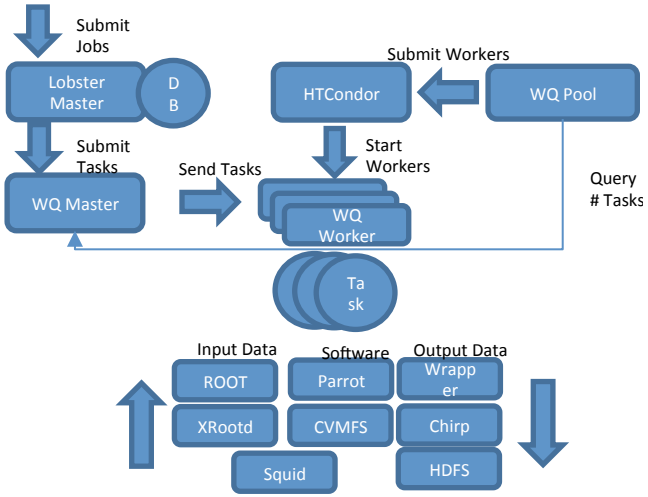
Nanoreactor MD Simulations



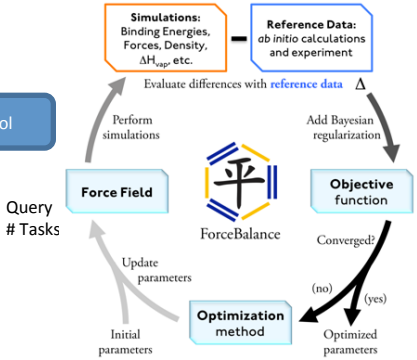
Scalable Assembler at Notre Dame



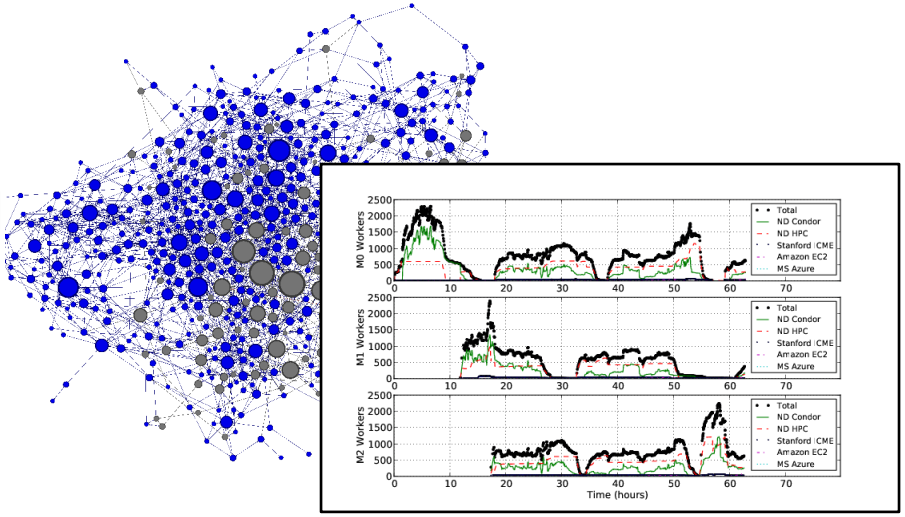
Lobster HEP



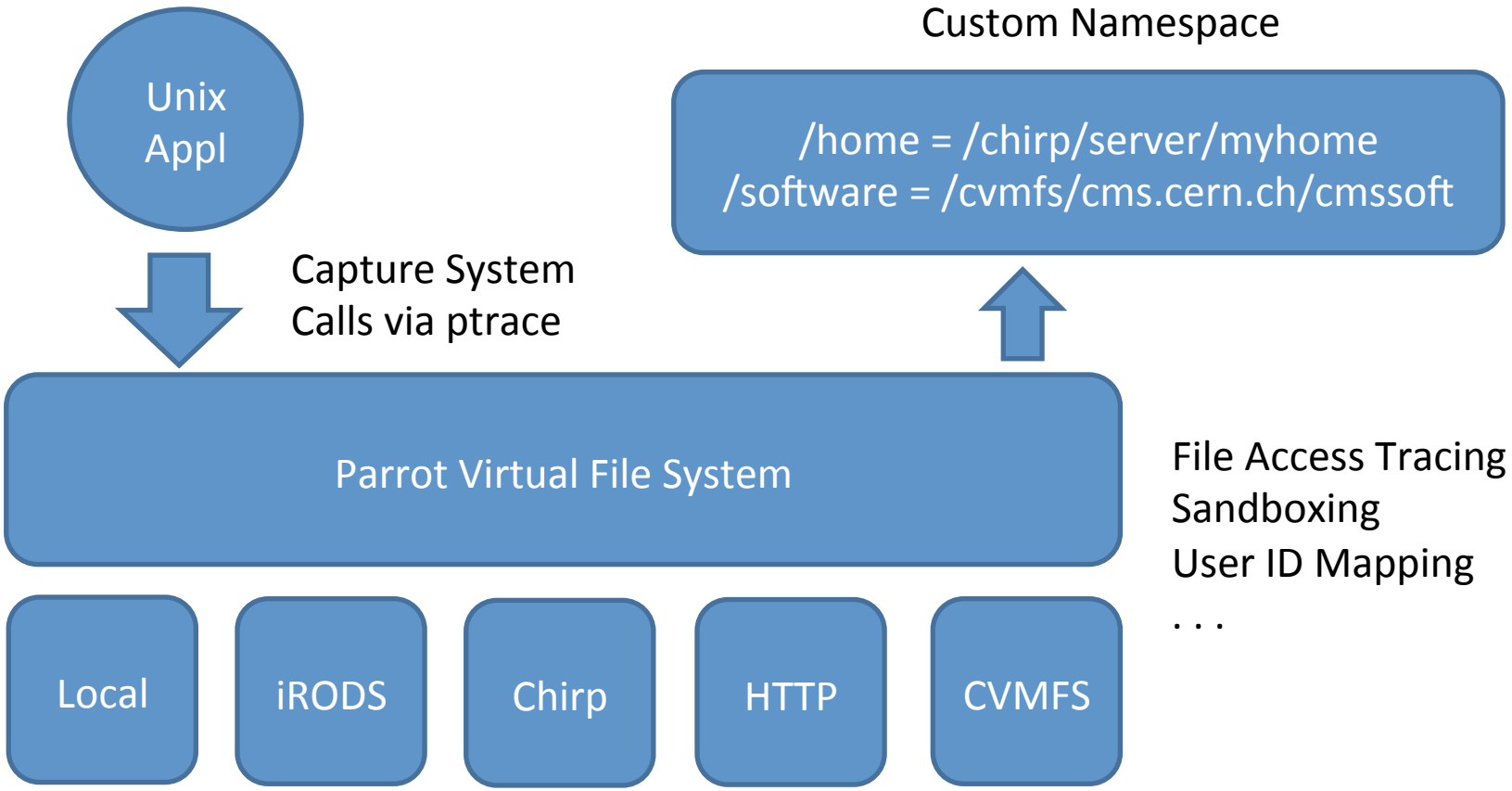
ForceBalance



Adaptive Weighted Ensemble



Parrot Virtual File System



Parrot runs as an ordinary user, so no special privileges required to install and use.
Makes it useful for harnessing opportunistic machines via a batch system.

```
% export HTTP_PROXY=http://ndcms.crc.nd.edu:3128
% parrot _run bash
```

```
% cd /cvmfs/cms.cern.ch
% ./cmsset_default.sh
% . . .
```

```
% parrot _run –M /cvmfs/cms.cern.ch=/tmp/cmspackage bash
```

```
% cd /cvmfs/cms.cern.ch
% ./cmsset_default.sh
% . . .
```

Parrot Requires Porting

- New, obscure, system calls get added and get used by the standard libraries, if not by the apps.
- Custom kernels in HPC centers often have odd bugs and features. (mmap on GPFS)
- Some applications depend upon idiosyncratic behaviors not formally required. (inode!=0)
- Our experience: a complex HEP stack doesn't work out of the box…. but after some focused collaboration with the CCL team, it works.
- Time to rethink if a less comprehensive implementation would meet the needs of CVMFS.

We Like to Repurpose Existing Interfaces

- POSIX Filesystem Interface (Parrot and Chirp)
  - open/read/write/seek/close
- Unix Process Interface (Work Queue)
  - fork / wait /kill
- Makefile DAG structure (Makeflow)
  - inputs : outputs \n command

Opportunistic Opportunities

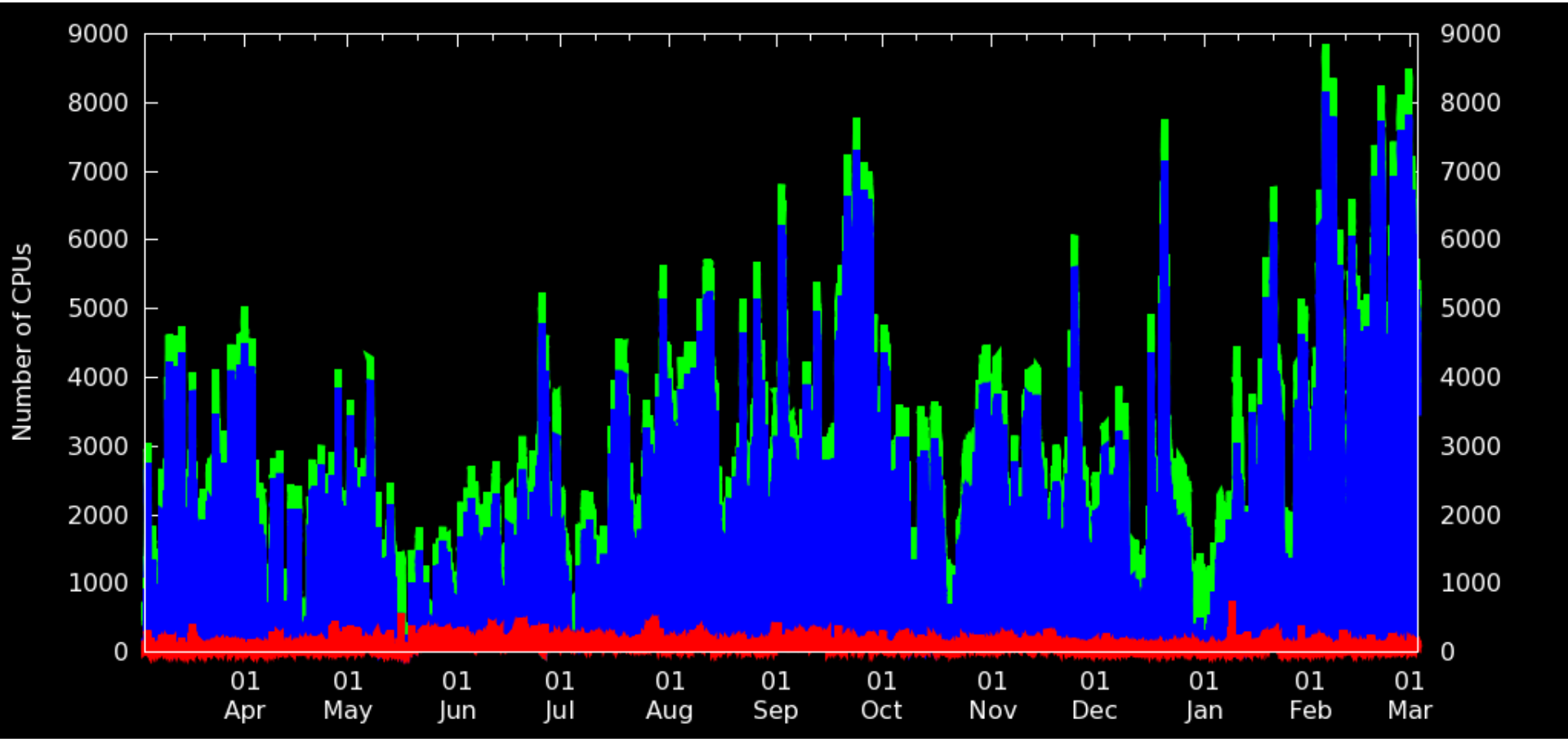
Opportunistic Computing

- Much of LHC computing is done in conventional computing centers with a fixed operating environment with professional sysadmins.
- But, there exists a large amount of computing power available to end users that is not prepared or tailored to your specific application:
  - National HPC facility that is not associated with HEP.
  - Campus-level cluster and batch system.
  - Volunteer computing systems: Condor, BOINC, etc.
  - Cloud services.
- Can we effectively use these systems for HEP computing?

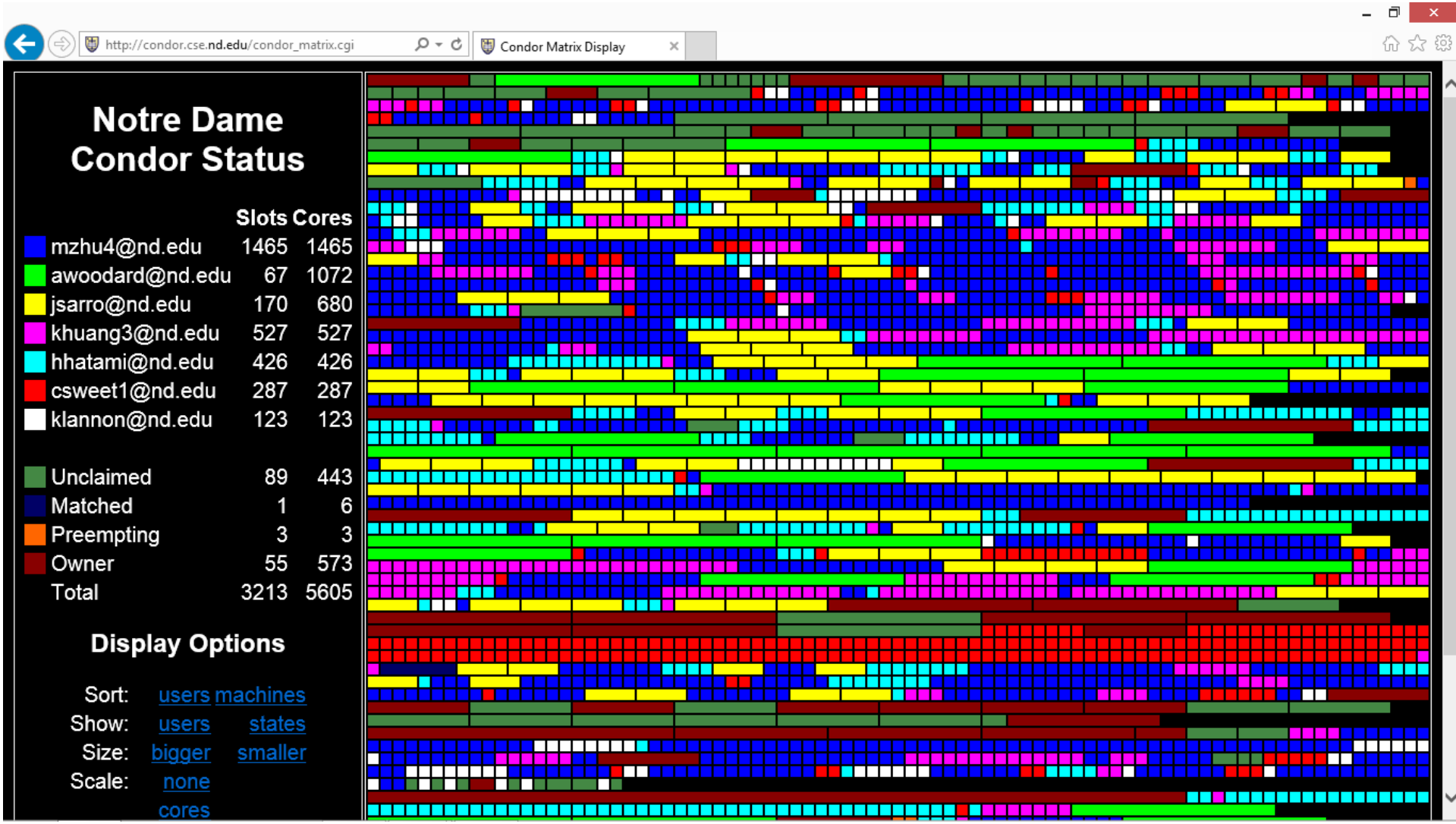
Opportunistic Challenges

- When borrowing someone else's machines, you cannot change the OS distribution, update RPMs, patch kernels, run as root…
- This can often put important technology **just barely** out of reach of the end user, e.g.:
  - FUSE might be installed, but without setuid binary.
  - Docker might be available, but you aren't a member of the required Unix group.
- The resource management policies of the hosting system may work against you:
  - Preemption due to submission by higher priority users.
  - Limitations on execution time and disk space.
  - Firewalls only allow certain kinds of network connections.

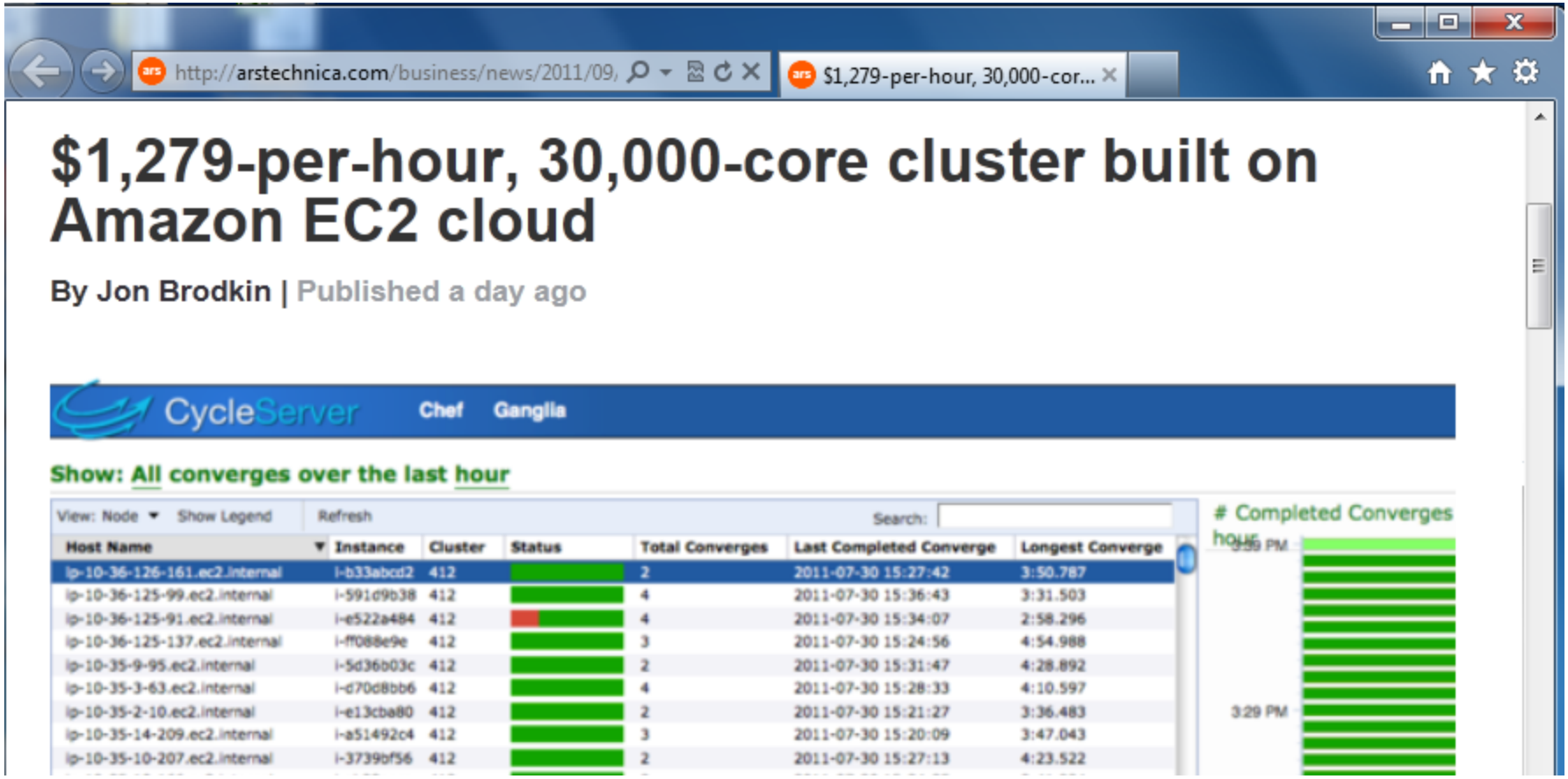
Backfilling HPC with Condor at Notre Dame



Users of Opportunistic Cycles



Superclusters by the Hour



http://arstechnica.com/business/news/2011/09/30000-core-cluster-built-on-amazon-ec2-cloud.ars

Lobster:
An Opportunistic
Job Management System

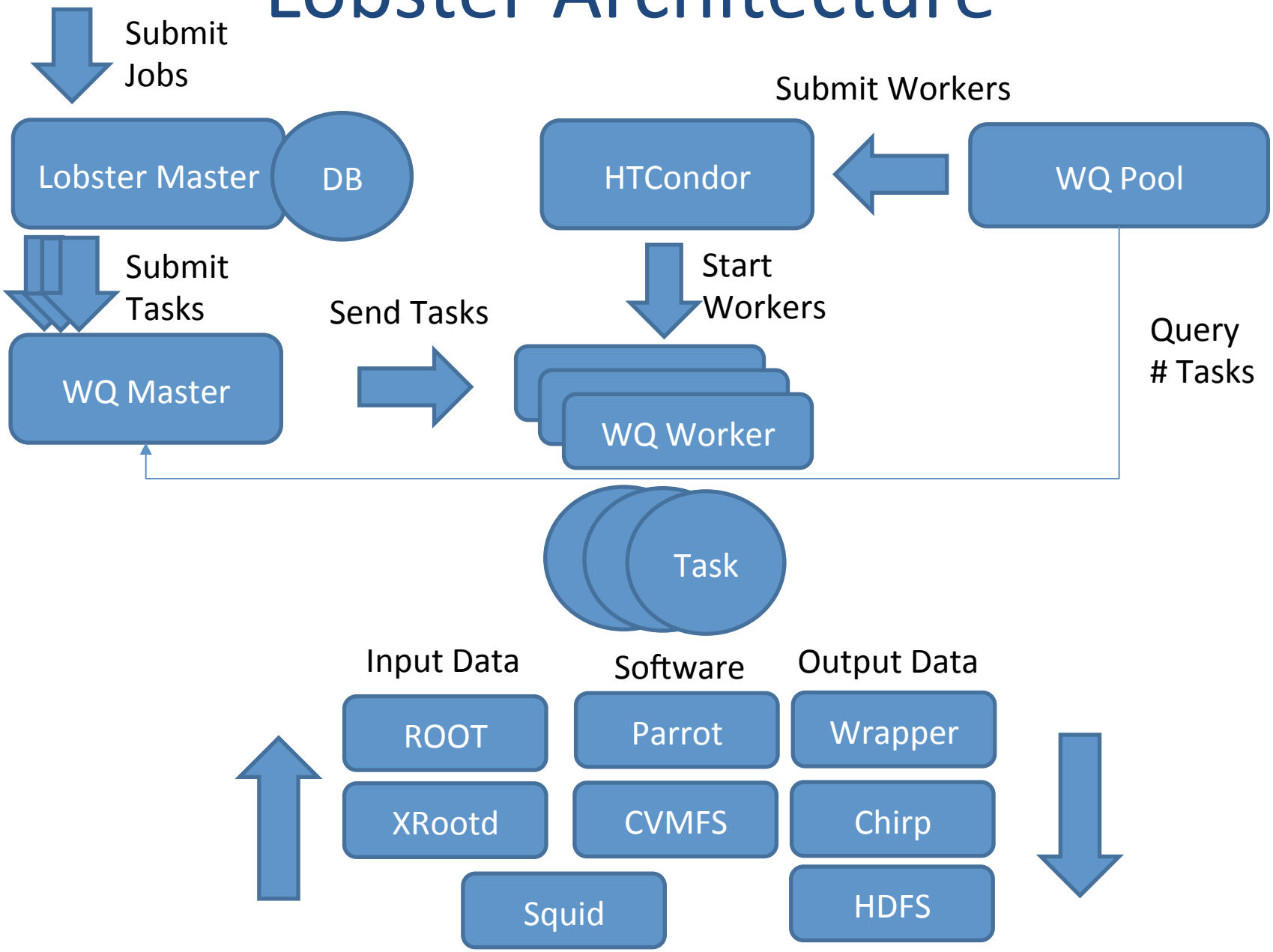
Lobster Objectives

- ND-CMS group has a modest Tier-3 facility of O(300) cores, but wants to harness the ND campus facility of O(10K) cores for their own analysis needs.
- But, CMS infrastructure is highly centralized
  - One global submission point.
  - Assumes standard operating environment.
  - Assumes unit of submission = unit of execution.
- We need a different infrastructure to harness opportunistic resources for local purposes.

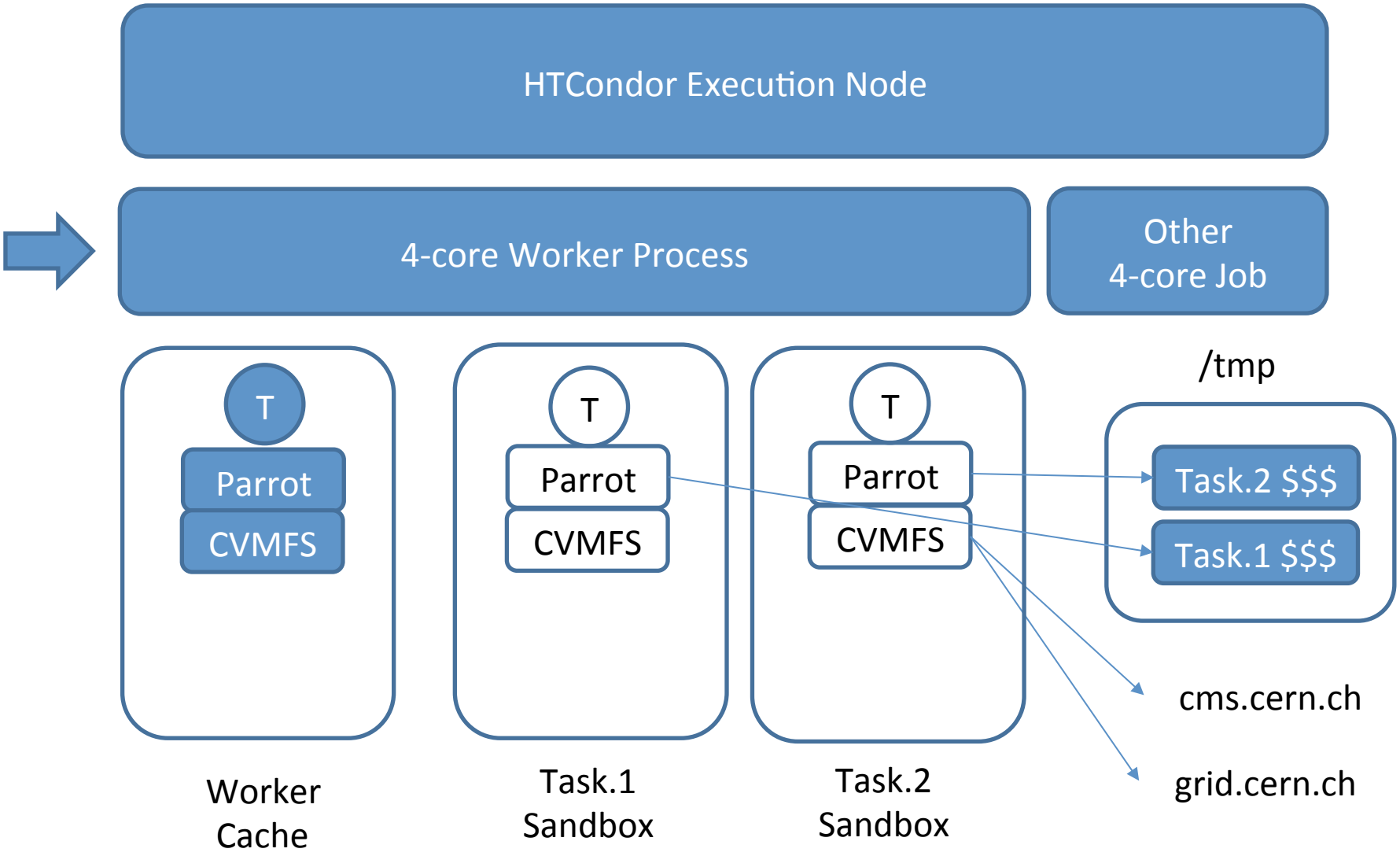
Key Ideas in Lobster

- Give each user their own scheduler/allocation.
- Separate task execution unit from output unit.
- Rely on CVMFS to deliver software.
- Share caches at all levels.
- Most importantly:
  Nothing left behind after eviction!

Lobster Architecture



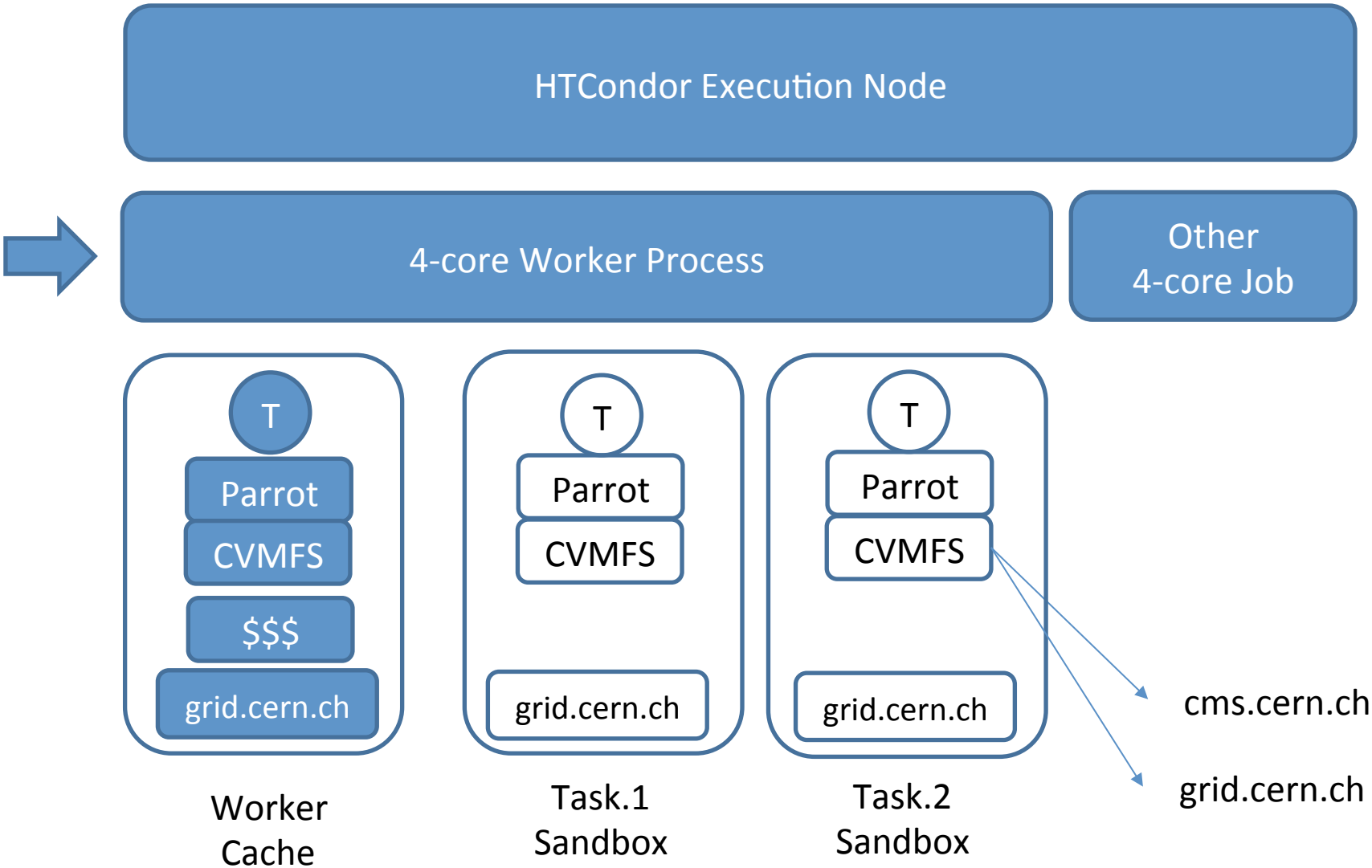
First Try on Worker



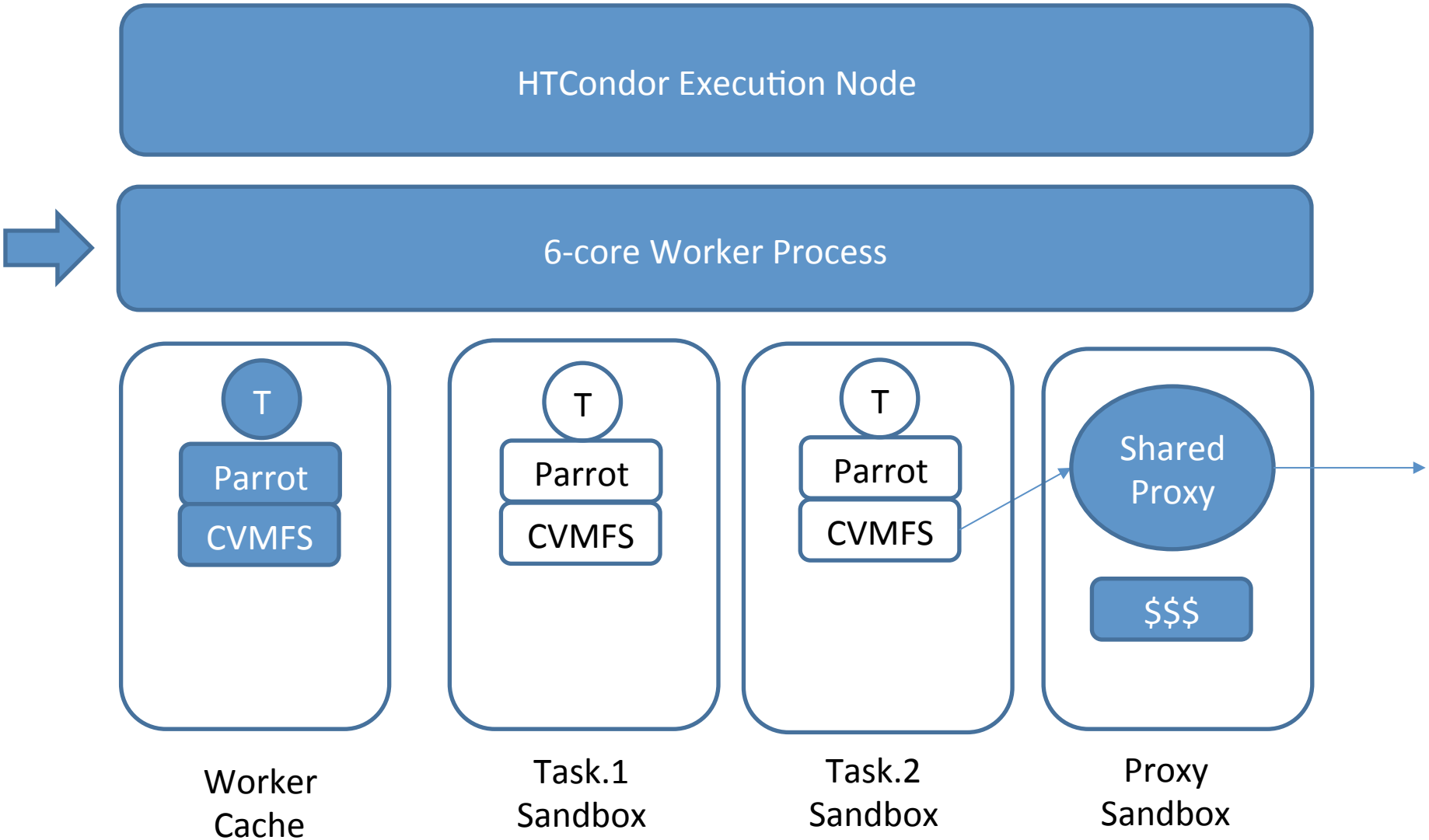
Problems and Solutions

- Parrot-CVMFS could not access multiple repositories simultaneously; switching between was possible but slow.
  - Workaround: Package up entire contents of (small) grid.cern.ch and use parrot to mount in place.
  - Fix: CVMFS team working on multi-instance support in libcvmfs.
- Each instance of parrot was downloading and storing its own copy of the software, then leaving a mess in /tmp.
  - Fix: Worker directs parrot to proper caching location.
  - Fix: CVMFS team implemented shared cache space, known as “alien” caching.

Improved Worker in Production

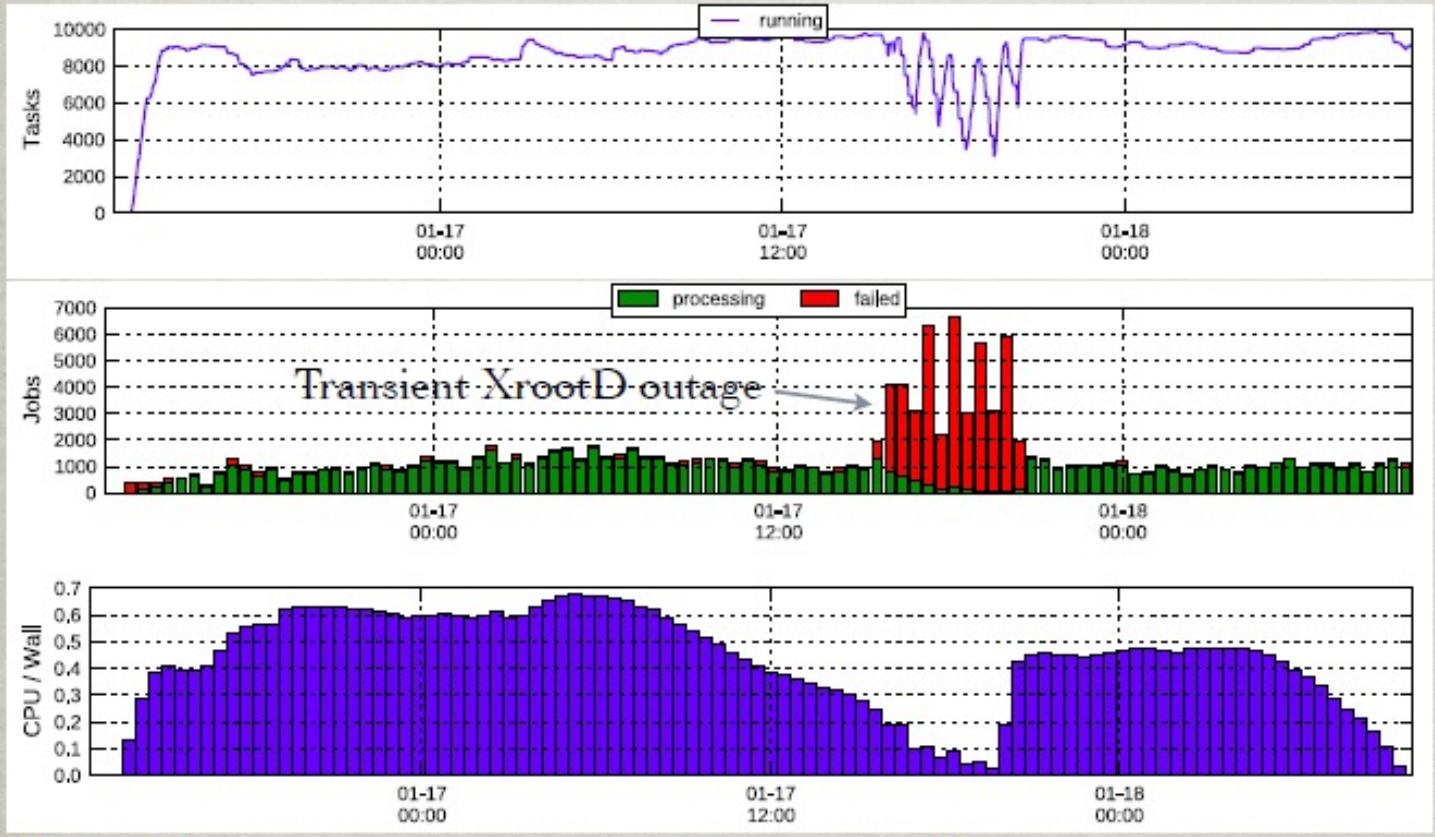


Idea for Even Better Worker



Reaching ~10k running jobs

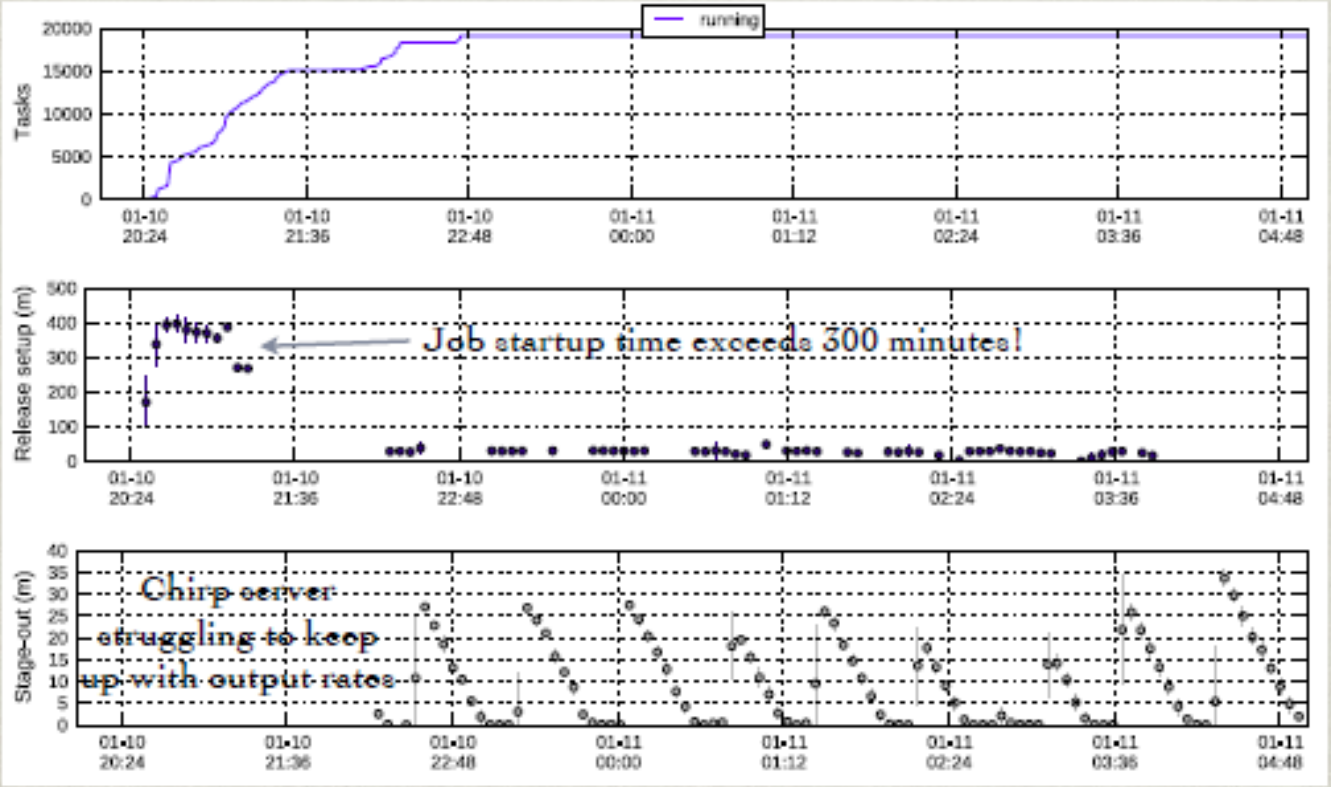
- This is ~10% of size of CMS global pool
- Comparable to scale of one US CMS T2 site
- More than total of all US CMS dedicated T3 resources



UNIVERSITY OF NOTRE DAME

Challenges

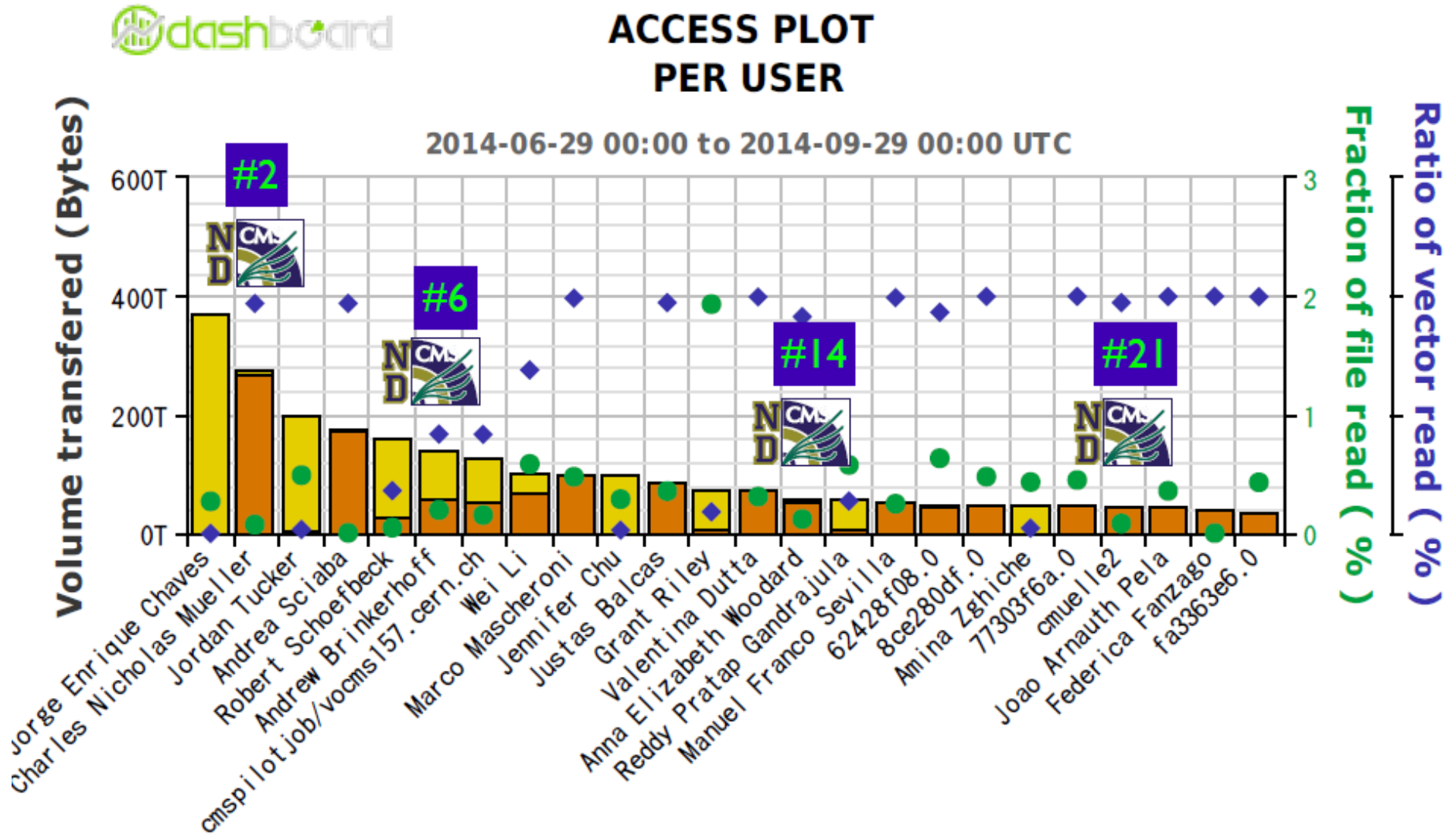
- Bottlenecks when running at large scales
  - Job overhead time increases non-linearly: bottleneck in squid cache?
  - Output Chirp server can get overloaded and fail



Monitoring

- Opportunistic resources change dynamically (chaotically)
  - Resources come and go depending on owner activity
  - Heterogeneous quality
  - Can fail randomly
- Monitoring critical to Lobster success
  - Lobster tracks time stamps of every phase of task setup and execution
  - Collects information in plots and tables on webpage
  - Gives comprehensive picture of system components so that bottlenecks and failures can be diagnosed and overcome

Lobster@ND Competitive with CSA14 Activity



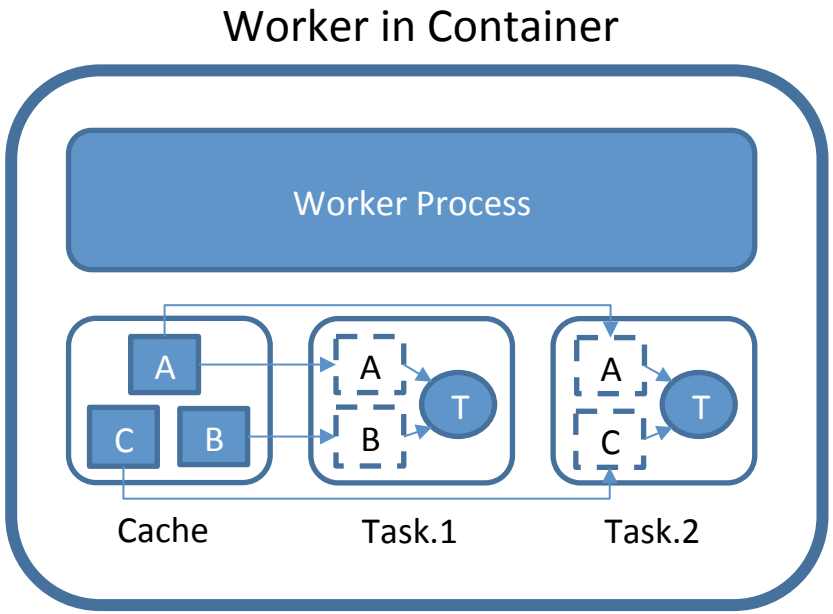
Looking Ahead

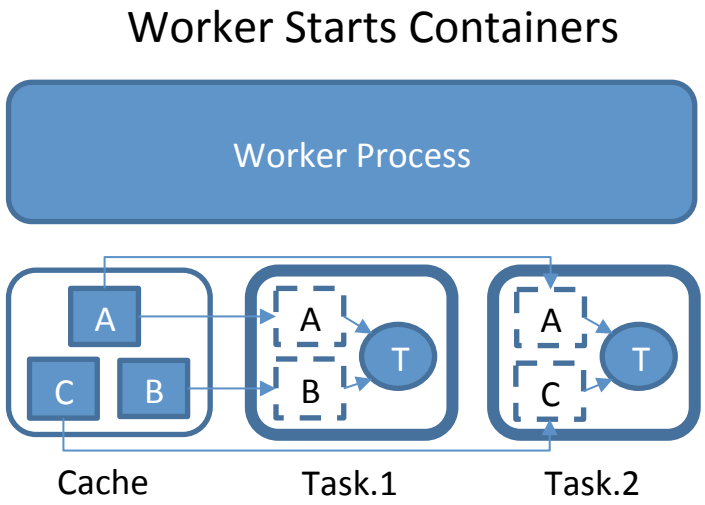
- Dynamic Proxy Cache Allocations
- Containers: Hype and Reality
- Pin Down the Container Execution Model
- Portability and Reproducibility
- Taking Advantage of HPC

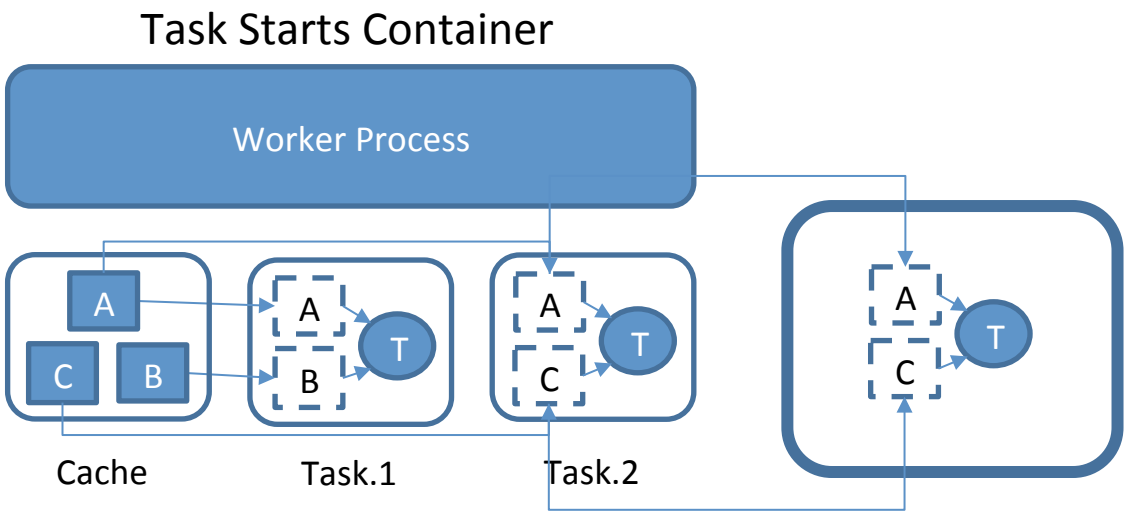
Containers

- Pro: Are making a big difference in that users now expect to be able to construct and control their own namespace.
- Con: Now everyone will do it, thus exponentially increasing efforts for porting and debugging.
- Limitation: Do not pass power down the hierarchy. E.g. Containers can't contain containers, create sub users, mount FUSE, etc…
- Limitation: No agreement yet on how to integrate containers into a common execution model.

Where to Apply Containers?

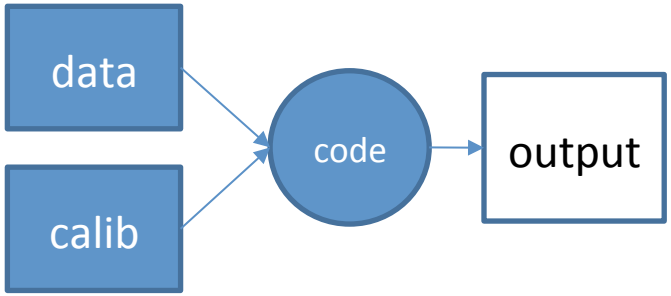


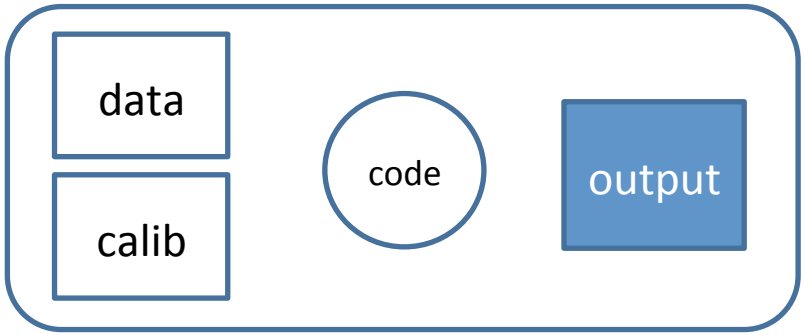




Abstract Model of Makeflow and Work Queue

```
output: data calib code
    code –i data –p 5 > output
```



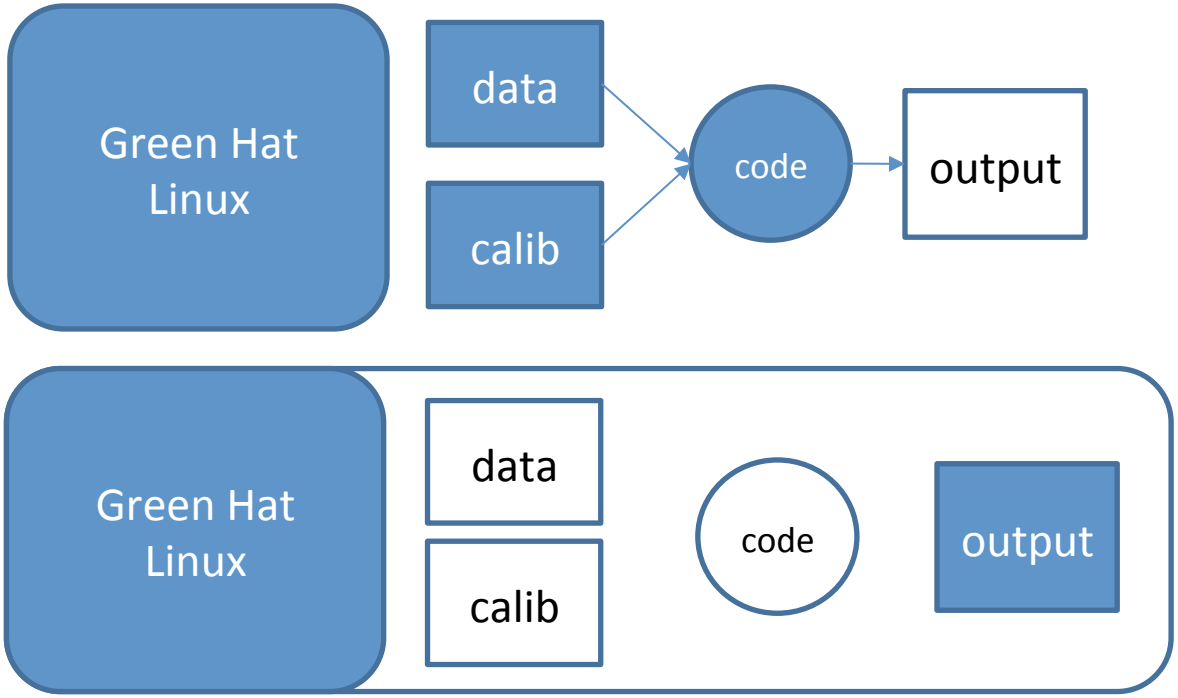


If inputs exist, then:
Make a sandbox.
Connect the inputs.
Run the task.
Save the outputs.
Destroy the sandbox.

Ergo, if it's not an input dependency, it won't be present.

Abstract Model of Execution in a Container

```
OPSYS=GreenHatLinux-87.12
output: data calib code $OPSYS
    code –i data –p 5 > output
```



Ergo, if it's not an input dependency, it won't be present.

Clearly Separate Three Things:

Inputs/Outputs:

The items of enduring interest to the end user.

Code:

The technical expression of the algorithms to be run.

Environment:

The furniture required to run the code.

Portability and Reproducibility

- Observation by Kevin Lannon @ ND: *Portability and reproducibility are two sides of the same coin.*
- Both require that you know all of the input dependencies and the execution environment in order to save it, or move to the right place.
- Workflow systems: Force us to treat tasks as proper functions with explicit inputs and outputs.
- CVMFS allows us to be explicit about *most* of the environment critical to the application.
- VMs/Containers handle the rest of the OS.
- The trick to efficiency is to identify and name common dependencies, rather than saving everything independently.

Acknowledgements

CCL Team

- Ben Tovar
- Patrick Donnelly
- Peter Ivie
- Haiyan Meng
- Nick Hazekamp
- Peter Sempolinski
- Haipeng Cai
- Chao Zheng

Collaborators

- Jakob Blomer - CVMFS
- Dave Dykstra – Frontier
- Dan Bradley – OSG/CMS
- Rob Gardner – ATLAS

Lobster Team

- Anna Woodard
- Matthias Wolf
- Nil Valls
- Kevin Lannon
- Michael Hildreth
- Paul Brenner

Cooperative Computing Lab

http://ccl.cse.nd.edu

Douglas Thain

dthain@nd.edu

The Cooperative Computing Lab

Software | Download | Manuals | Papers

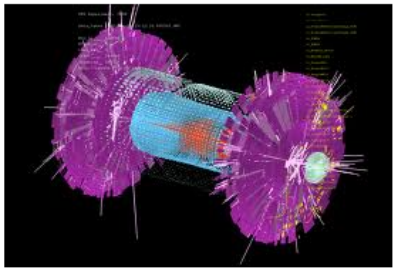
About the CCL

We design software that enables our collaborators to easily harness large scale distributed systems such as clusters, clouds, and grids. We perform fundamental computer science research in that enables new discoveries through computing in fields such as physics, chemistry, bioinformatics, biometrics, and data mining.

CCL News and Blog

- Creating Better Force Fields on Distributed GPUs with Work Queue
- CCTools 4.3 released
- Work Queue Powers Nanoreactor Simulations
- Open Sourcing Civil Engineering with a Virtual Wind Tunnel
- DeltaDB - A Scalable Database Design for Time-Varying Schema-Free Data
- Packaging Applications with Parrot 4.2.0
- CCTools 4.2.0 released
- DeltaDB at IEEE BigData 2014

Community Highlight



Scientists searching for the Higgs boson have profited from Parrot's new support for the CernVM Filesystem (CVMFS), a network filesystem tailored to providing world-wide access to software installations. By using Parrot, CVMFS, and additional components integrated by the Any Data, Anytime, Anywhere project, physicists working in the Compact Muon Solenoid experiment have been able to create a uniform computing environment across the Open Science Grid. Instead of maintaining large software installations at each participating institution, Parrot is used to provide access to a single highly-available CVMFS installation of the software from which files are downloaded as needed and aggressively cached for efficiency. A pilot project at the University of Wisconsin has demonstrated the feasibility of this approach by exporting excess compute jobs to run in the Open Science Grid, opportunistically harnessing 370,000 CPU-hours across 15 sites with seamless access to 400 gigabytes of software in the Wisconsin CVMFS repository.

- Dan Bradley, University of Wisconsin and the Open Science Grid

Research

- Papers
- Projects
- People
- Jobs
- REU

Software

- Download
- Manuals
- Makeflow
- Work Queue
- Parrot
- Chirp
- SAND
- AWE

Community

- Highlights
- Annual Meeting
- Workshops
- Getting Help
- Mailing List
- For Developers

Operations

- Condor Display
- Condor Pool
- Hadoop Cluster
- Biocompute
- BXGrid
- Condor Log Analyzer
- Internal