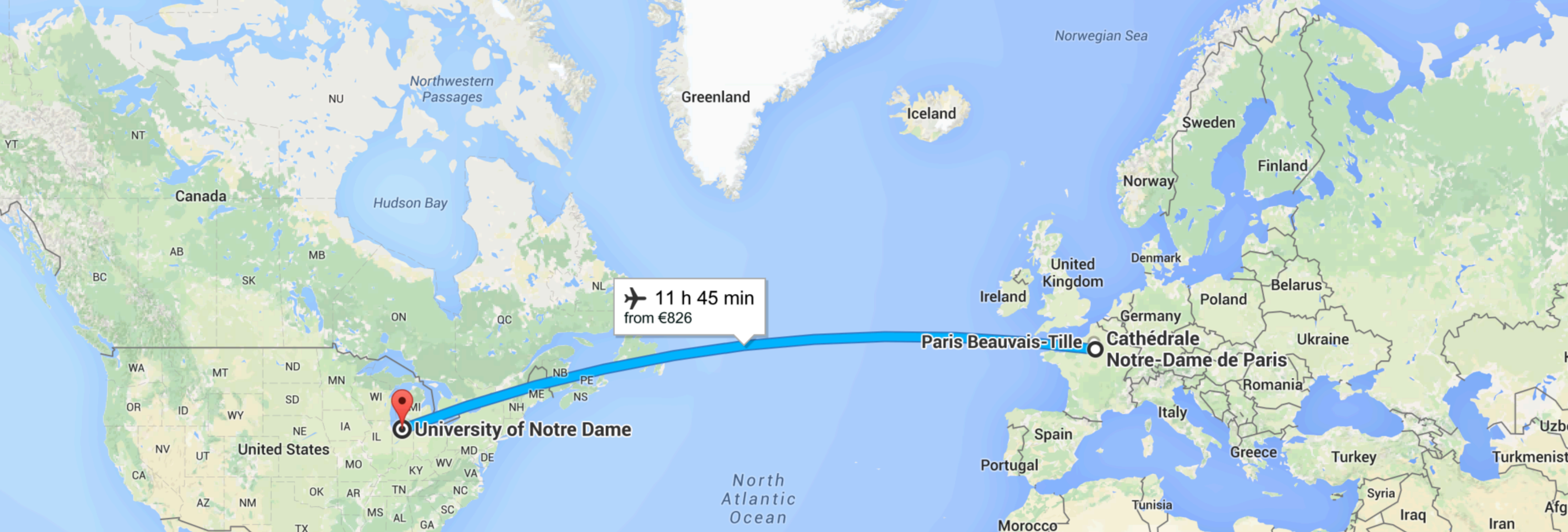
Preservation and Portability in Distributed Scientific Computing

Douglas Thain

Grid 5000 Winter School

Grenoble, February 2016

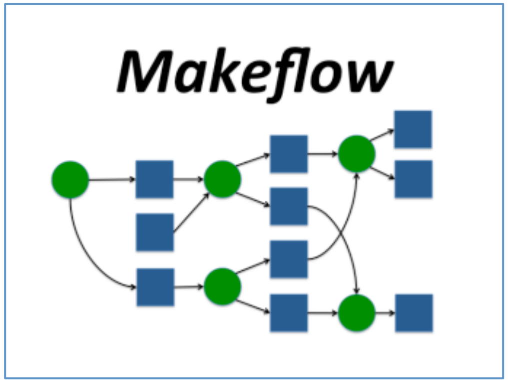
UNIVERSITY OF NOTRE DAME

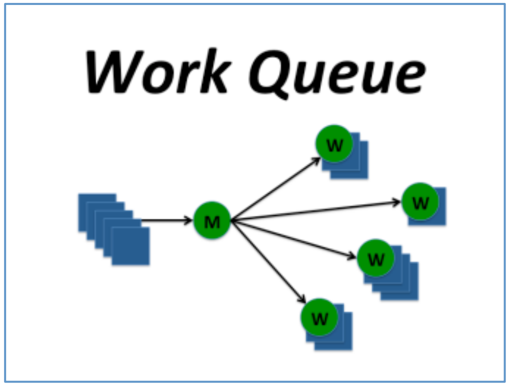


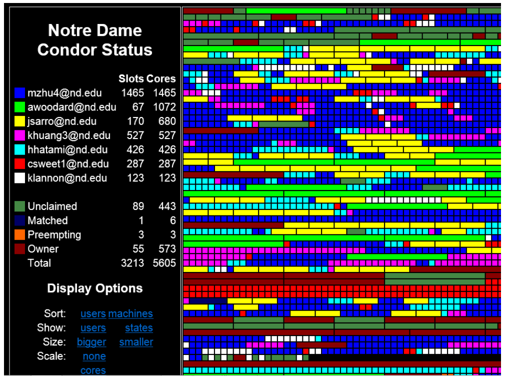
University of Notre Dame du Lac
Est 1842

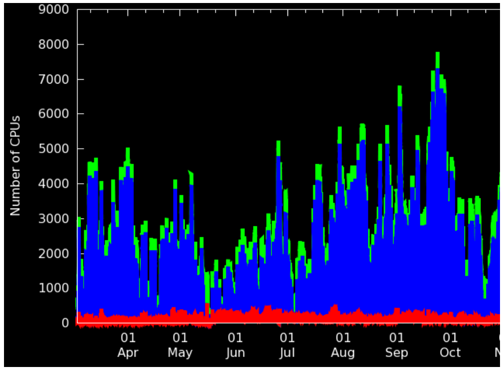
Notre Dame de Paris
Est 1163

The Cooperative Computing Lab









http://ccl.cse.nd.edu

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

Condor Matrix Display

condor.cse.nd.edu/condor_matrix.cgi

Notre Dame Condor Status

	Slots	Cores
csweet1@nd.edu	4153	4153
mzhu4@nd.edu	2878	2878
jsarro@nd.edu	325	1300
ksmith37@nd.edu	18	144
apaul2@nd.edu	12	12
pivie@nd.edu	12	12
Unclaimed	114	504
Matched		
Preempting		
Owner	76	765
Total	7588	9768

Display Options

Sort: users machines

Show: users states

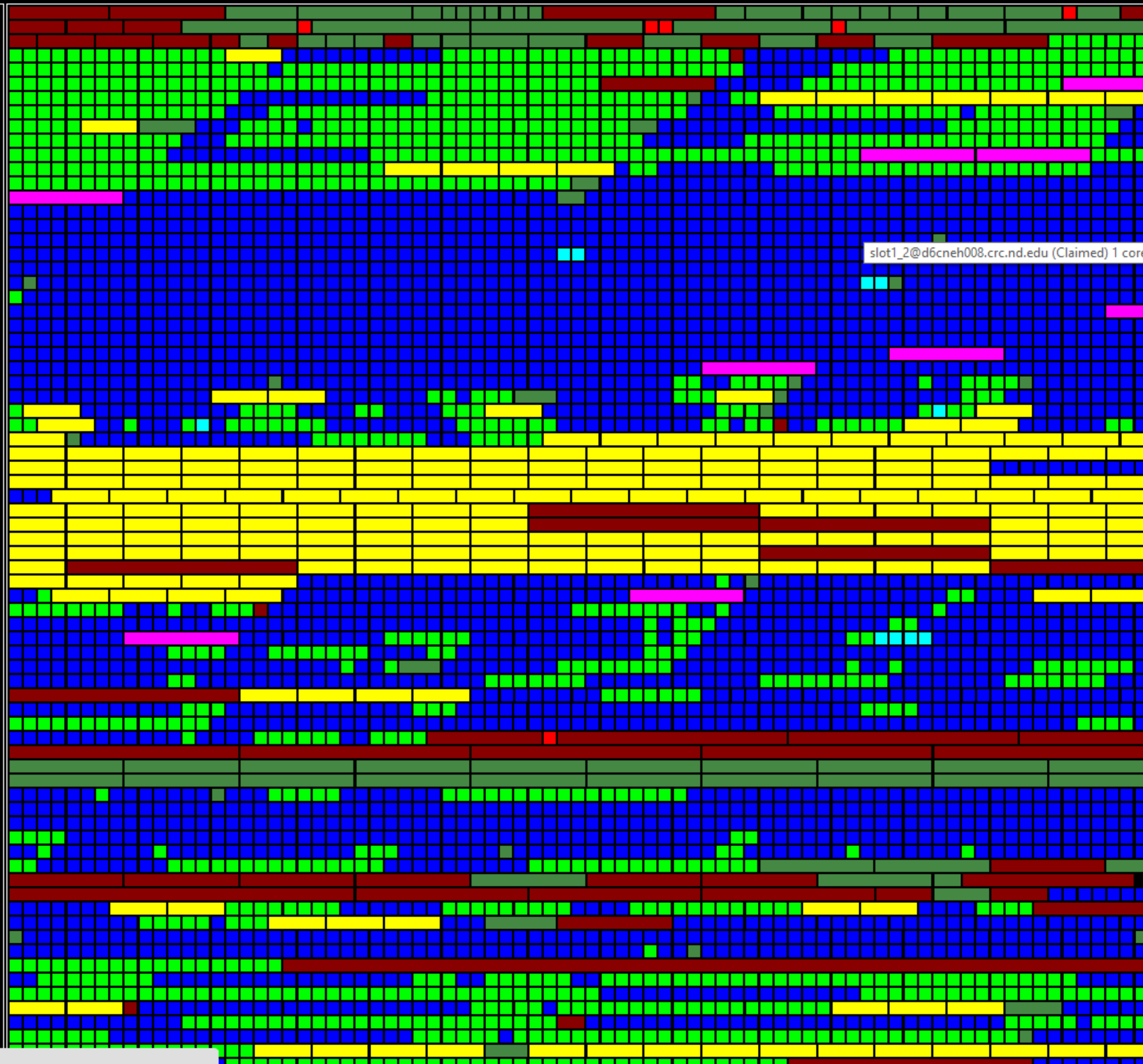
Size: bigger smaller

Scale: none

cores

memory

http://condor.cse.nd.edu



condor.cse.nd.edu/condor_matrix.cgi#

www.daspos.org

DASPOS Data and Software Preservation for Open Science

ABOUT | PEOPLE | WORKSHOPS | RESEARCH | REPORTS

The massive data sets accumulated by High Energy Physics (HEP) experiments represent the most direct result of the often decades-long process of construction, commissioning and data aquisition that characterize this science. Many of these data are unique and represent an irreplaceable resource for potential future studies. Forward-thinking efforts for preservation are necessary now in order to achieve the relevant parameters, analysis paths and software to preserve the usefulness of these rich and varied data sets.

"Ten or 20 years ago we might have been able to repeat an experiment. They were simpler, cheaper and on a smaller scale. Today that is not the case. So if we need to re-evaluate the data we collect to test a new theory, or adjust it to a new development, we are going to have to be able to resuse it. That means we are going to need to save it as open data..."

Rolf-Dieter Heur 2008
Director General, CERN

Data and Software Preservation for Open Science, DASPOS, represents an initial exploration of the key technical problems that must be solved to provide appropriate data, software and algorithmic preservation for HEP, including the contexts necessary to understand, trust and reuse the data. While the archiving of HEP data may require some HEP-specific technical solutions, DASPOS will create a template for preservation that will be useful across many different disciplines, leading to a broad, coordinated effort.

First Workshop Scheduled

The first DASPOS Workshop has been scheduled for Thursday - Friday, March 21-22, 2013, at CERN. More information

Workshop 1

2012-12-17 19:11:04

WORKSHOP 1 Establishment of Use Cases for Archived Data and Software in HEP Date: Thursday-Friday...

Workshop 2

2012-12-17 19:11:04

WORKSHOP 2 Survey of Commonality with other Disciplines Attendees: Broad participation from many...

Discovery and Coordination

Series of highly-structured public workshops to define, discuss and document the details of data and software preservation

Prototyping and Experimentation

Key areas of reserach: data and query models and software sustainability models

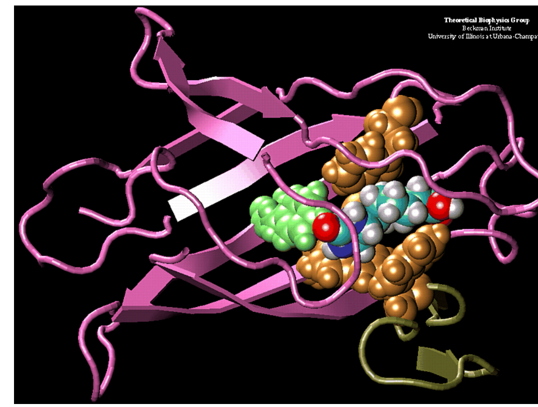
The DASPOS Team

Computer science experts, experienced digital librarians, and experts in data-intensive fields, such as physics, astrophysics and bioinformatics

Some of Our Collaborators

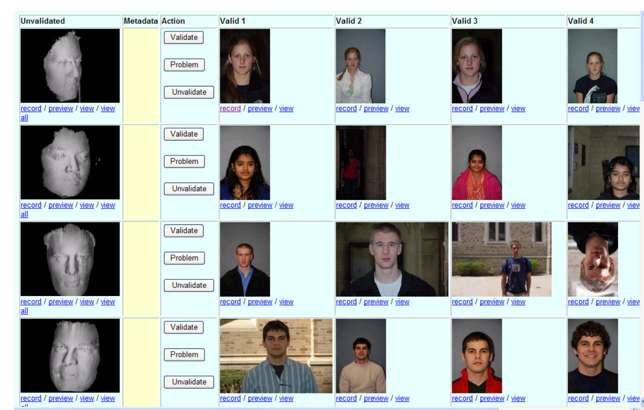
K. Lannon: Analyze 2PB of data produced by the LHC experiment at CERN

J. Izaguirre: Simulate 10M different configurations of a complex protein.



S. Emrich: Analyze DNA in thousands of genomes for similar sub-sequences.

P. Flynn: Computational experiments on millions of acquired face videos.



Reproducibility in Scientific Computing is Very Poor Today

- Can I re-run a result from a colleague from five years ago, and obtain the same result?
- How about a student in my lab from last week?
- Today, are we preparing for our current results to be re-used by others five years from now?
- Multiple reasons why not:
  - Rapid technological change.
  - No archival of artifacts.
  - Many implicit dependencies.
  - Lack of backwards compatibility.
  - Lack of social incentives.

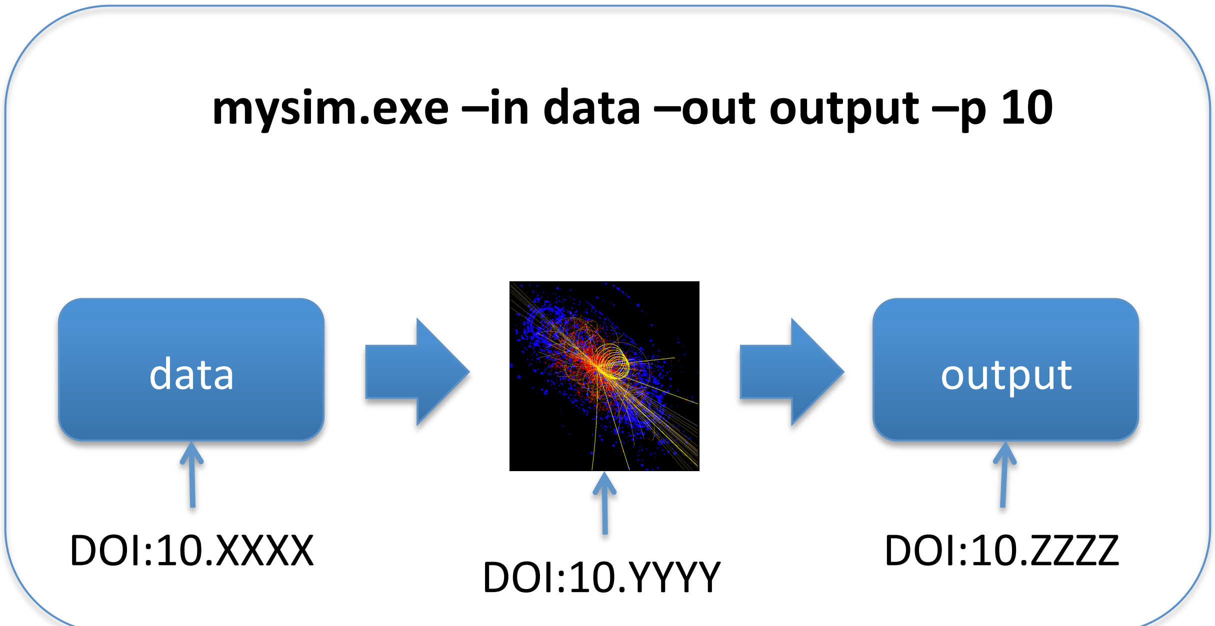
Our scientific collaborators see the value in reproducibility…

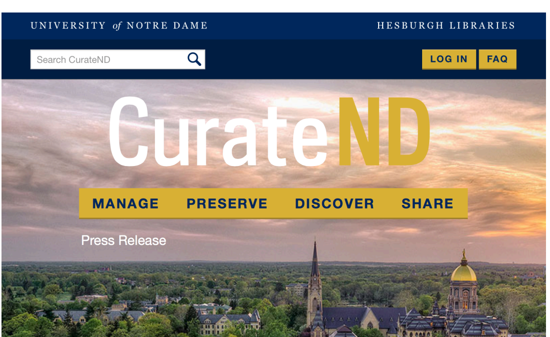
But only if it can be done

- easily
- at large scale
- with high performance

In principle, preserving a software execution is easy.

I want to preserve my simulation method **and** results so other people can try it out.

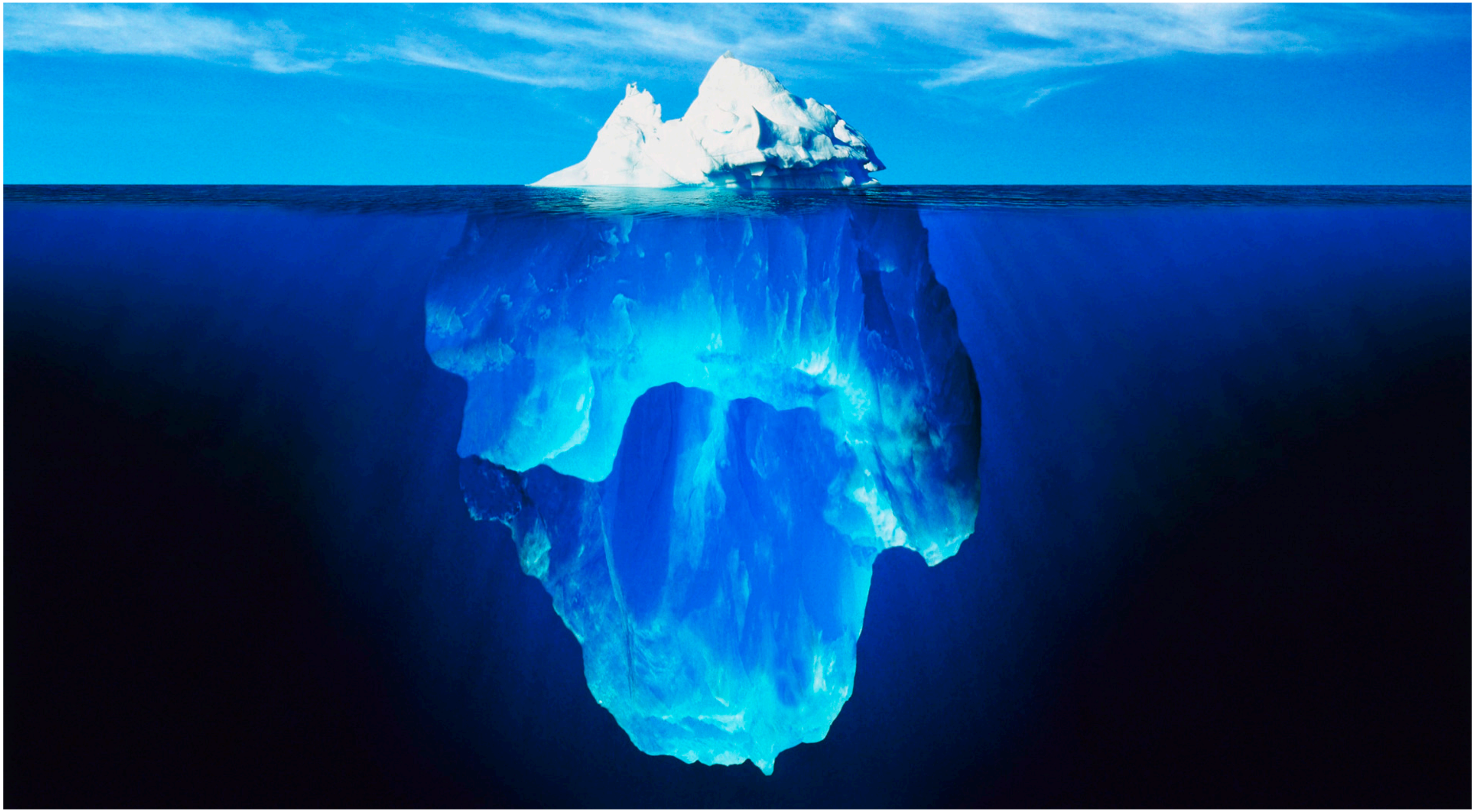




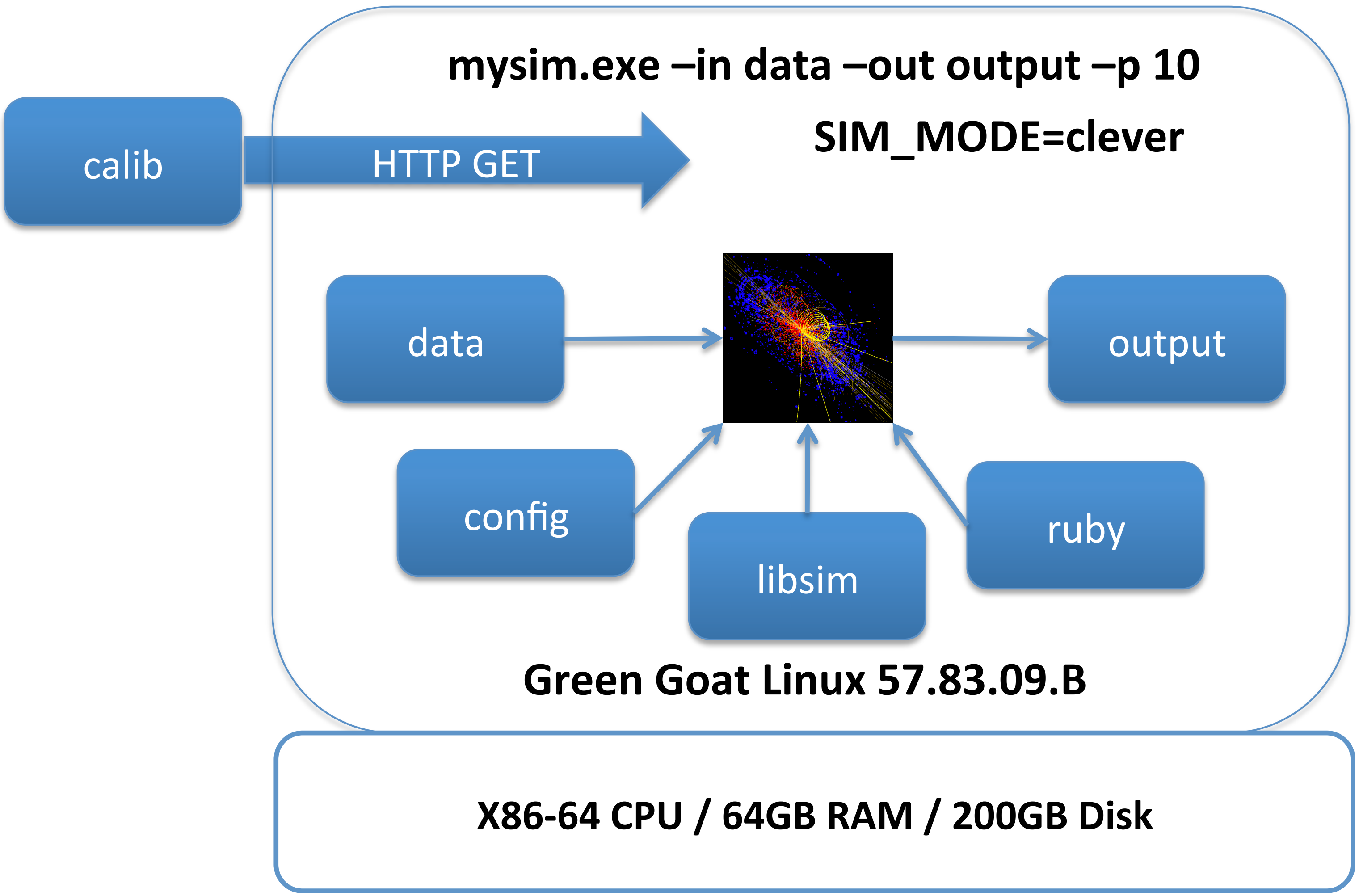
DOI:10.CCCC

… and repeat this 1M times with different –p values.

But it’s not that simple!



I want to preserve my simulation method **and** results so other people can try it out.



The problem is
implicit dependencies:

(things you need but cannot see)

How do we find them?
How do we deliver them?

Two approaches:

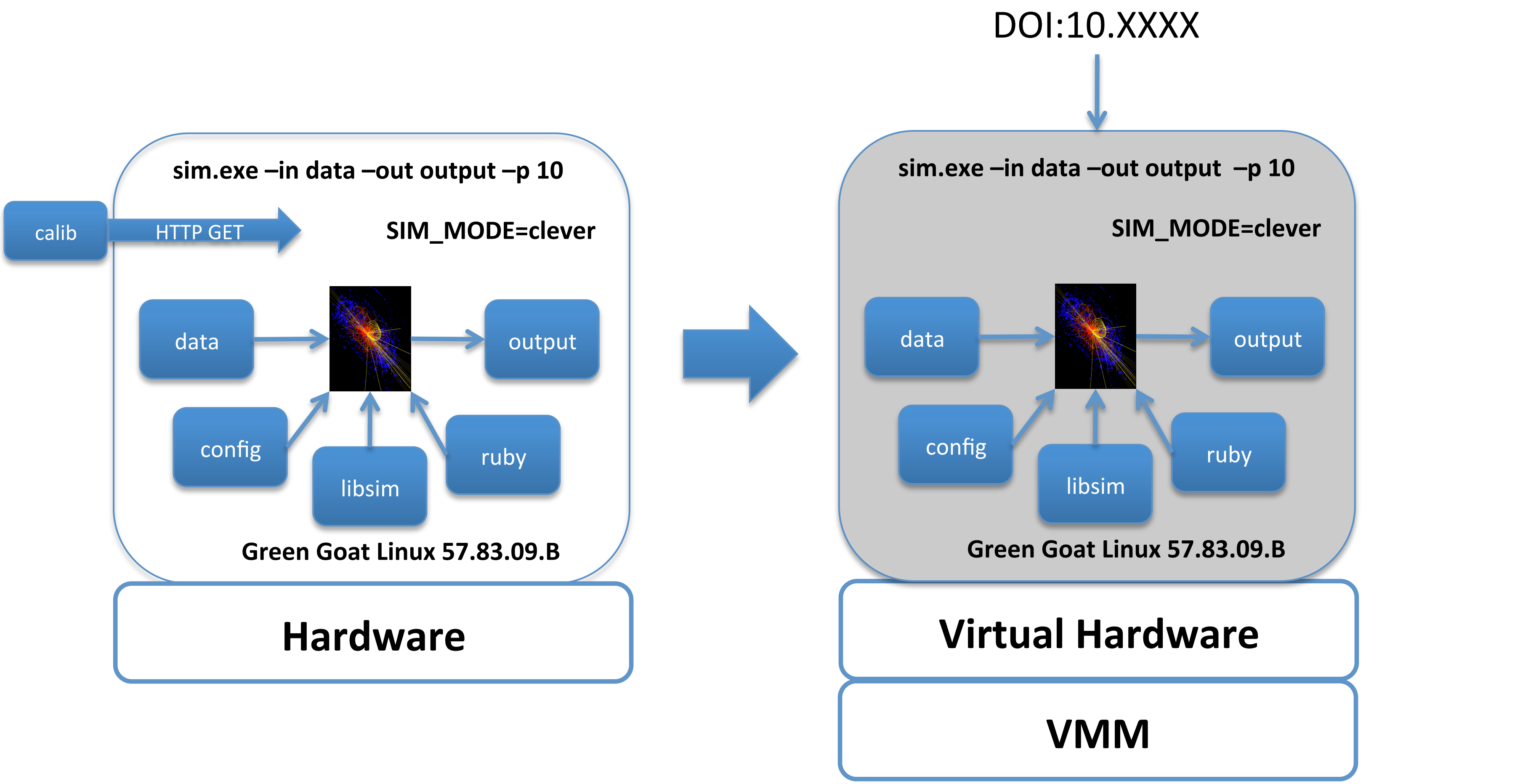
Preserve the Mess

(VMs, Packaging)

Encourage Cleanliness

(CVMFS, Umbrella, and Prune)

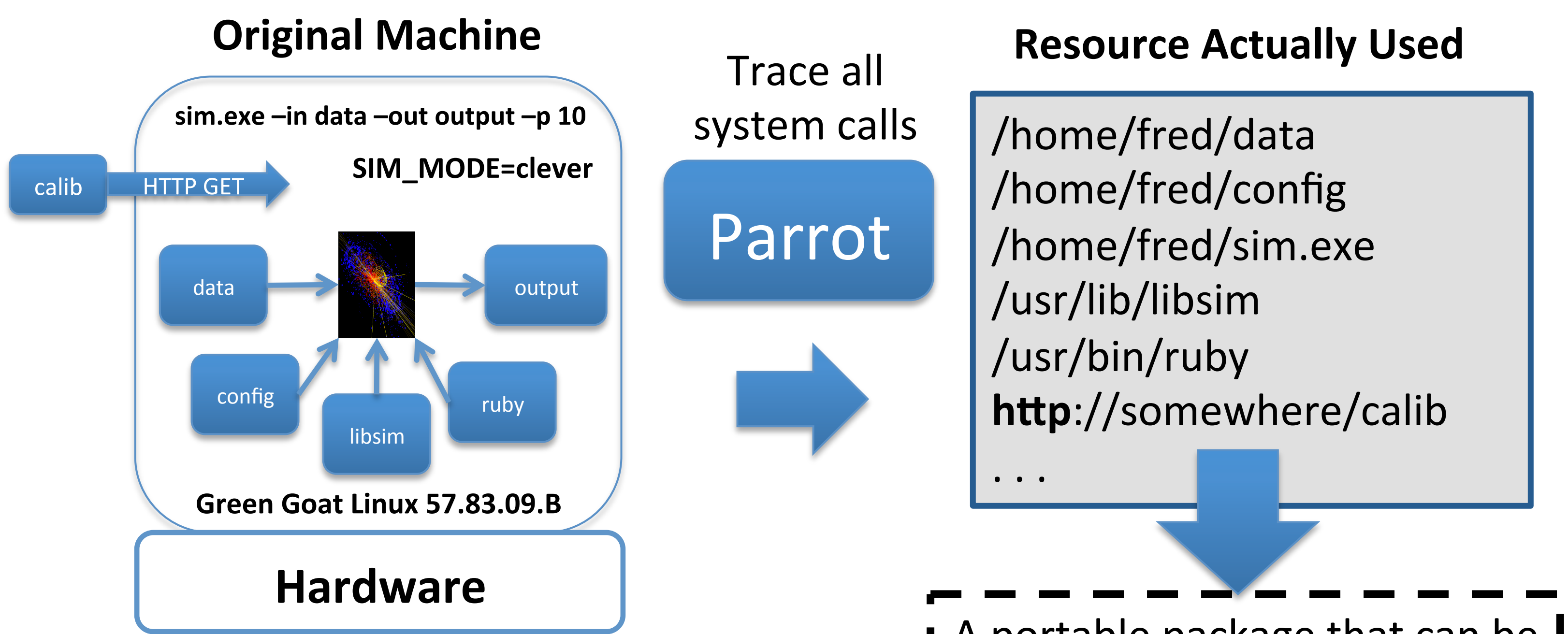
Preserve the Mess: Save a VM



Preserve the Mess: Save a VM

- Not a bad place to start, but:
  - Captures more things than necessary.
  - Duplication across similar VM images.
  - Hard to disentangle things logically – what if you want to run the same thing with a new OS?
  - Doesn't capture network interactions.
  - May be coupled to a specific VM technology.
  - VM services are not data archives.

Preserve the Mess: Trace Individual Dependencies



A portable package that can be re-executed using Docker, Parrot, or Amazon

Case Study: 166TB reduced to a 21GB package.

Haiyan Meng, Matthias Wolf, Peter Ivie, Anna Woodard, Michael Hildreth, Douglas Thain, [A Case Study in Preserving a High Energy Physics Application with Parrot,](#) *Journal of Physics: Conference Series*, December, 2015.

Preserve the Mess: Trace Individual Dependencies

- Solves some problems:
  - Captures only what is necessary.
  - Observes network interactions.
  - Portable across VM technologies.
- Still has some of the same problems:
  - Hard to disentangle things logically – what if you want to run the same thing with a new OS?
  - Duplication across similar VM images / packages.
  - VM services are not data archives.

Encourage Cleanliess:

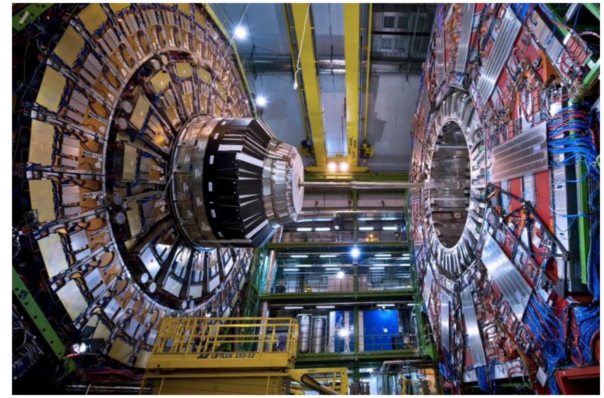
First, preserve the necessary software.
Then, design apps to access it.

Case Study:

CMS Data Analysis at Global Scale
with Parrot and CMVFS

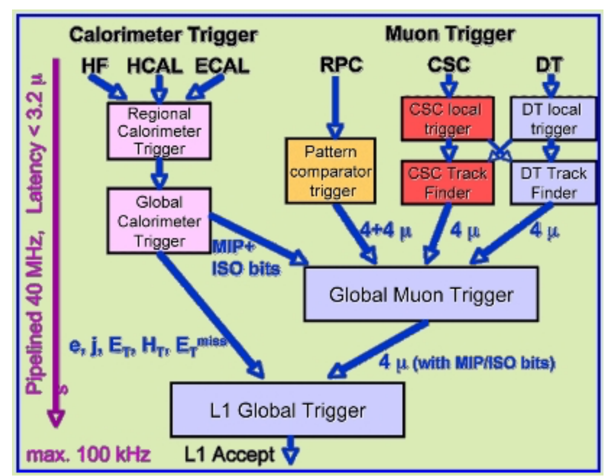
Large Hadron Collider

Compact Muon Solenoid



100 GB/s

Online Trigger



Many PB
Per year

Worldwide LHC Computing Grid

CMS Group at Notre Dame

Matthias Wolf

Anna Woodard

Prof. Hildreth

Prof. Lannon

Sample Problem:

Search for events like this:

t t H -> τ τ -> (many)

τ decays too quickly to be observed directly, so observe the many decay products and work backwards.

Was the Higgs Boson generated?

(One run requires successive reduction of many TB of data using hundreds of CPU years.)

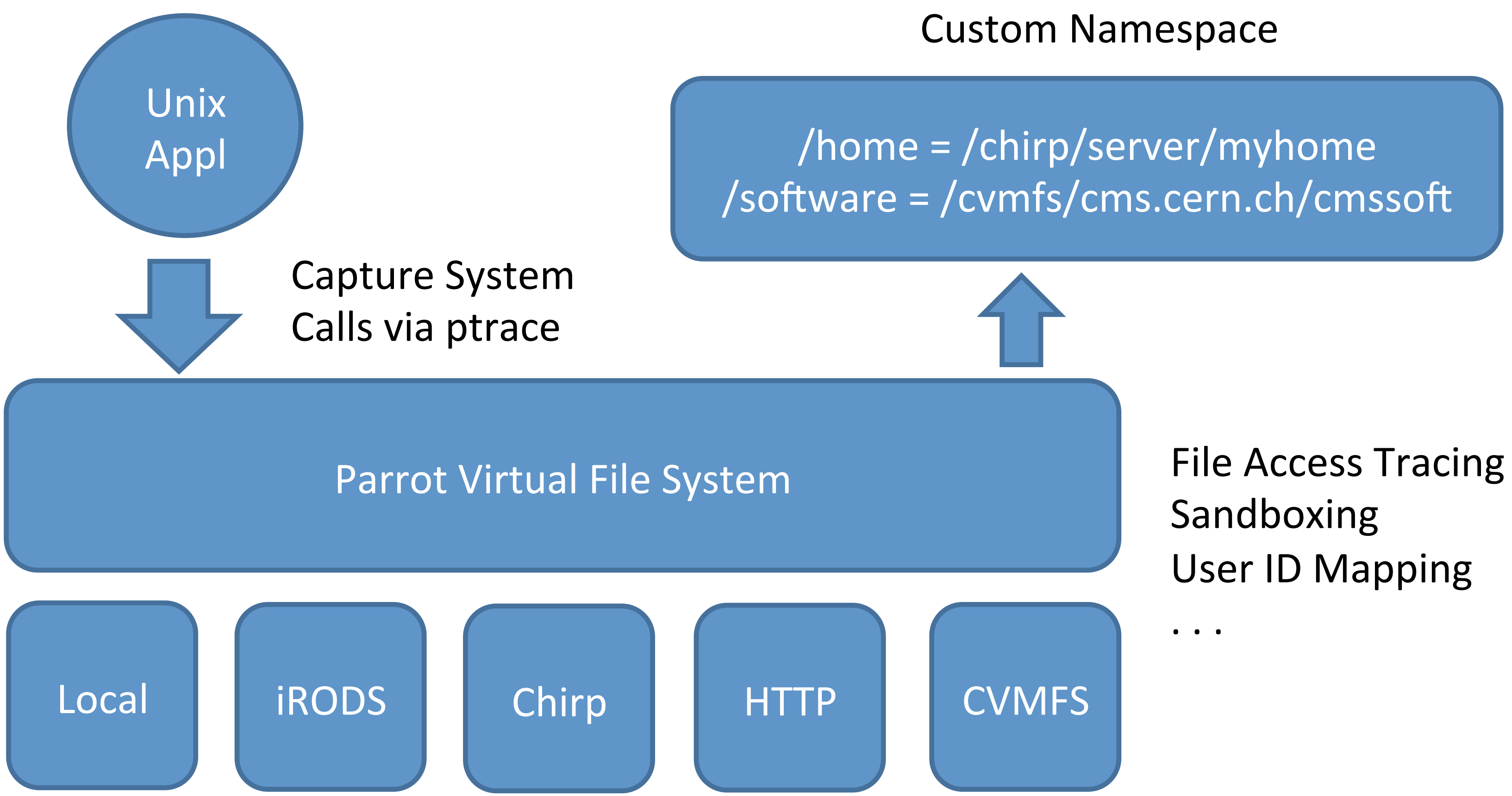
CMS Application Software

- Carefully curated and versioned collection of analysis software, data access libraries, and visualization tools. (Good news!)
- Several hundred GB of executables, compilers, scripts, libraries, configuration files…
- User expects:

```
export CMSSW /path/to/cmssw
$CMSSW/cmsset_default.sh
```

- How can we deliver the software everywhere?

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.

How to Use Parrot

% parrot_run bash

(starts new shell with parrot enabled)

% cat /http/www.google.com

(see html source of web page)

% cd /anonftp/ftp.gnu.org

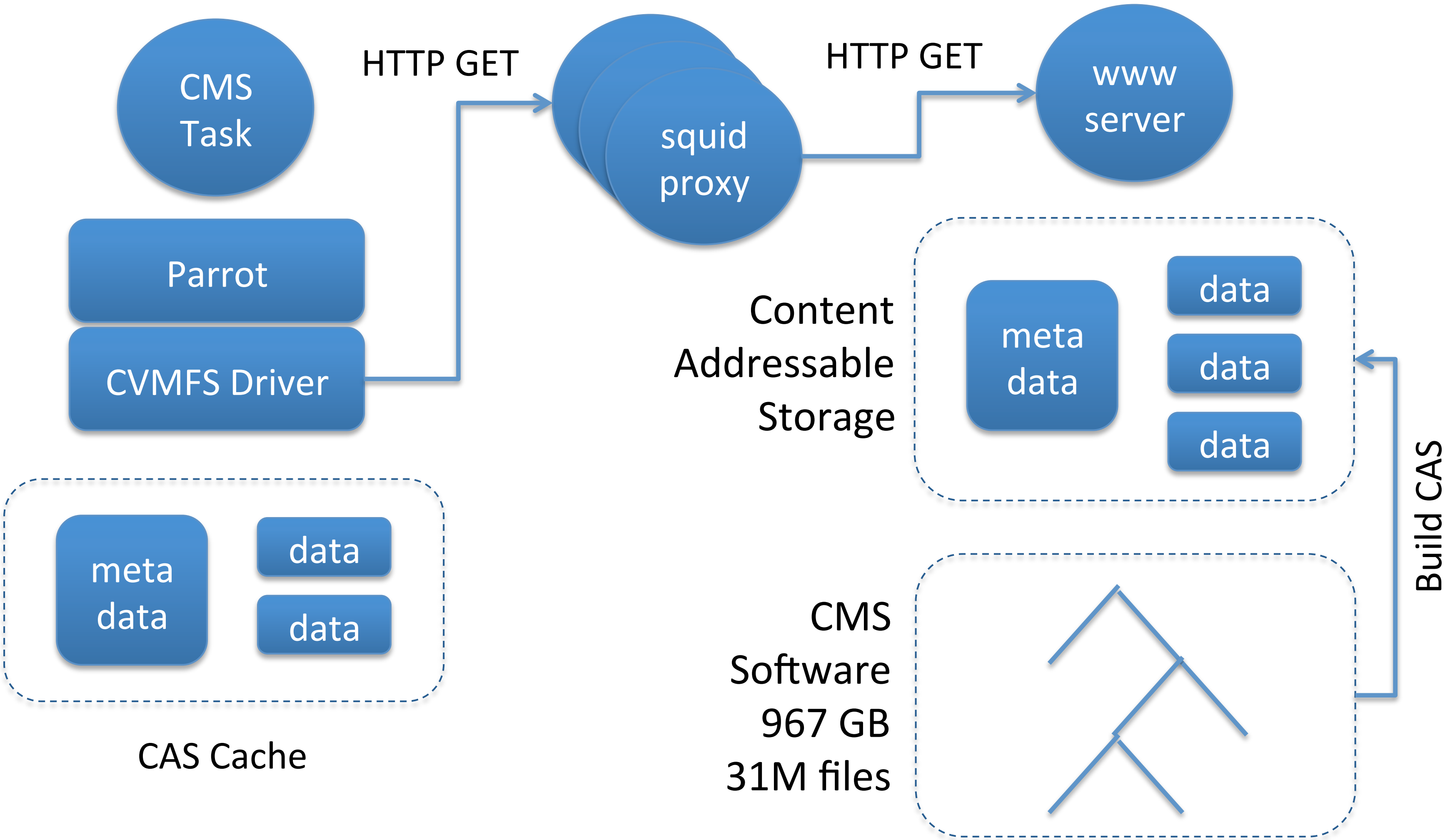
(browse GNU software archive)

% cd /cvmfs/cms.cern.ch

(see global view of CMS software via CVMFS)

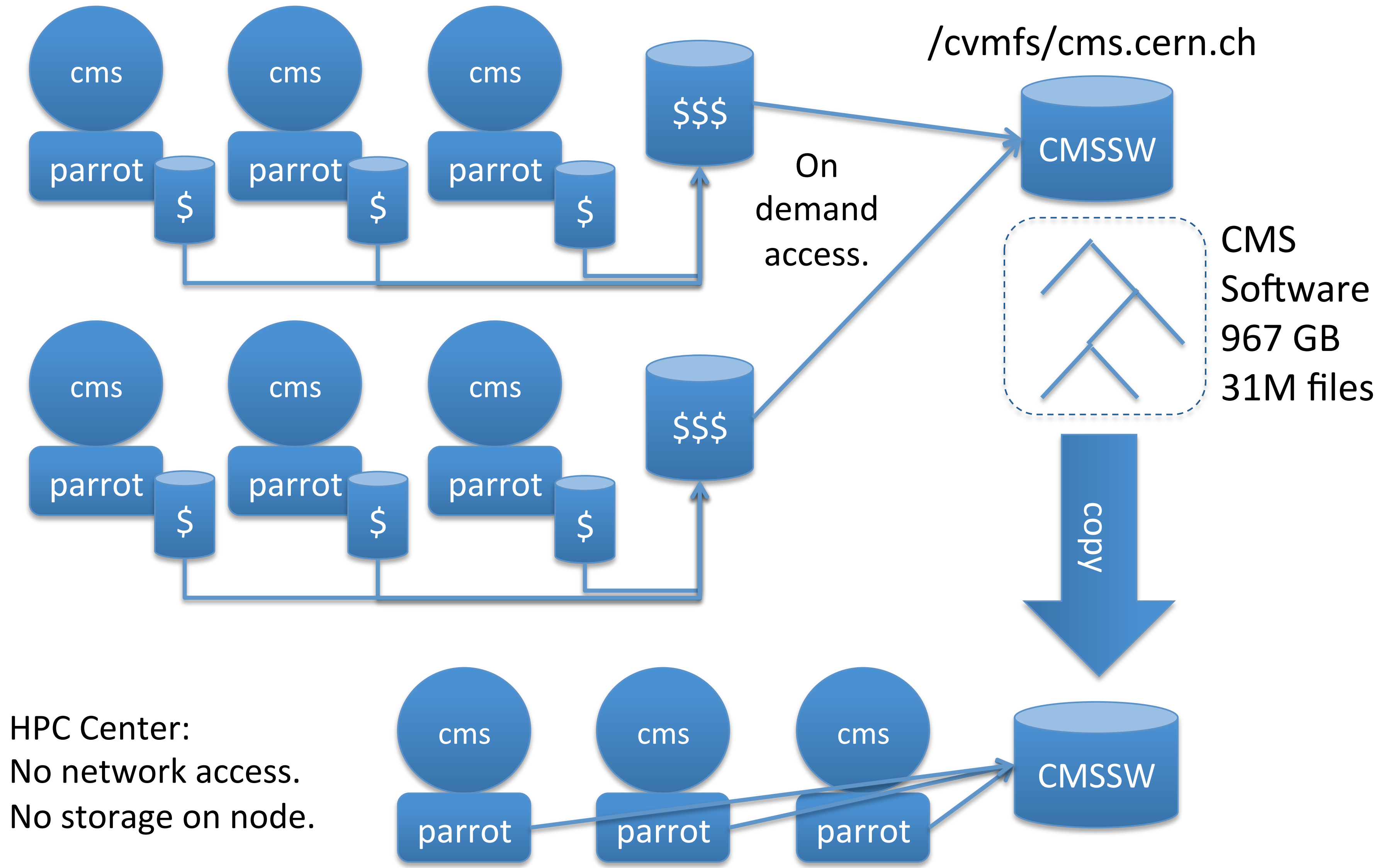
http://ccl.cse.nd.edu/software/parrot

CVMFS Filesystem



http://cernvm.cern.ch/portal/filesystem

Parrot + CVMFS at Scale



Parrot + CVMFS

- Global distribution of a widely used software stack, with updates automatically deployed.
- Metadata is downloaded in bulk, so directory operations are all fast and local.
- Only the subset of files actually used by an applications are downloaded. (Typically MB)
- Data sharing at machine, cluster, and site.

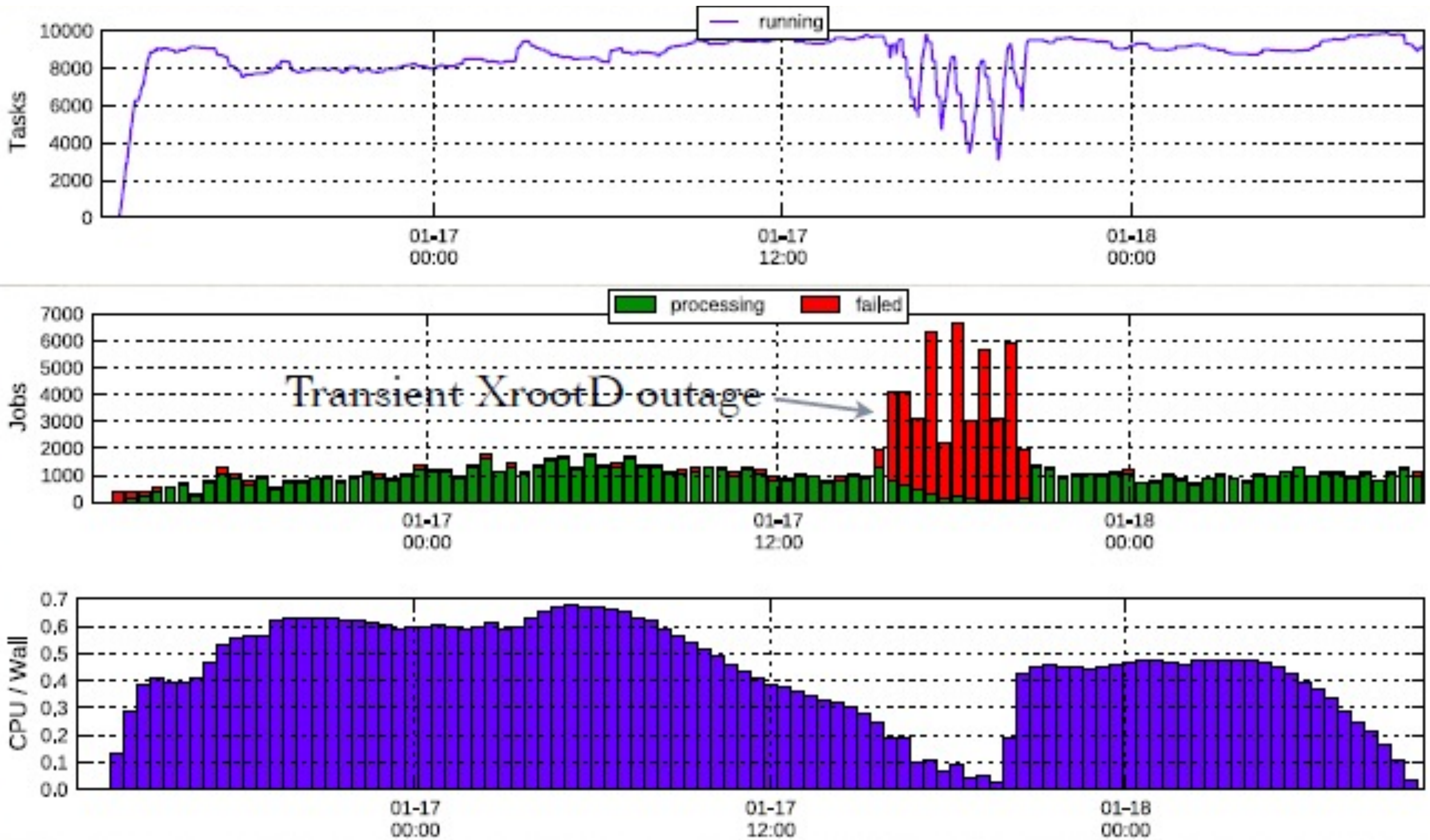
Jakob Blomer, Predrag Buncic, Rene Meusel, Gerardo Ganis, Igor Sfiligoi and Douglas Thain,
The Evolution of Global Scale Filesystems for Scientific Software Distribution,
IEEE/AIP Computing in Science and Engineering, **17**(6), pages 61-71, December, 2015.

The Good News

- ND daily production runs on 1K cores.
- Largest runs: 10K cores on data analysis jobs, and 20K cores on simulation jobs.
- One instance of Lobster at ND is larger than all CMS Tier-3s, and 10% of the CMS WLCG.
- CVMFS distributes software to O(100K) cores around the world via FUSE or Parrot.

Anna Woodard, Matthias Wolf, Charles Mueller, Nil Valls, Ben Tovar, Patrick Donnelly, Peter Ivie, Kenyi Hurtado Anampa, Paul Brenner, Douglas Thain, Kevin Lannon and Michael Hildreth, **Scaling Data Intensive Physics Applications to 10k Cores on Non-Dedicated Clusters with Lobster,** *IEEE Conference on Cluster Computing*, September, 2015.

Running on 10K Cores



Portability and Reproducibility are Closely Related!

- To get **portability** around the world, we:
  - Store a single, consistent environment image.
  - Import that image at execution sites.
  - Verify that the environment is correct.
  - Allow the end-user to control the namespace.

- To get **reproducibility**, we need more:
  - Disallow access to anything **not** in the image.
  - Give user control over **storage** of the image.
  - Bring together **multiple** kinds of dependencies.

Encourage Cleanliness:

We want a structured way
to compose an application with
multiple dependencies.

Enable preservation and sharing of
data and images for efficiency.

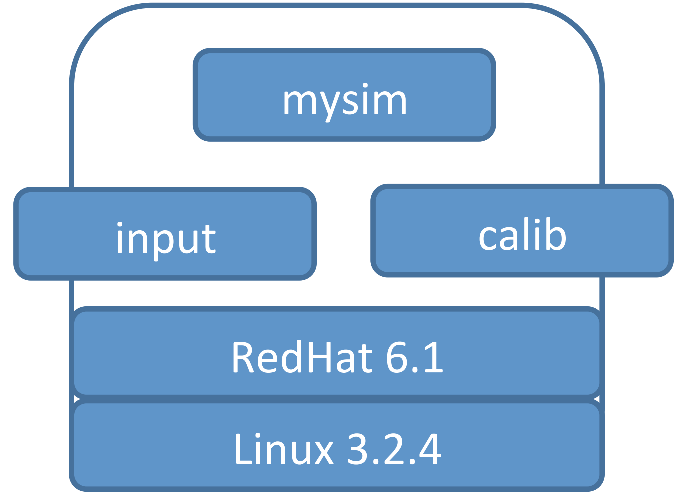
Encourage Cleanliness: Umbrella

mysim.json

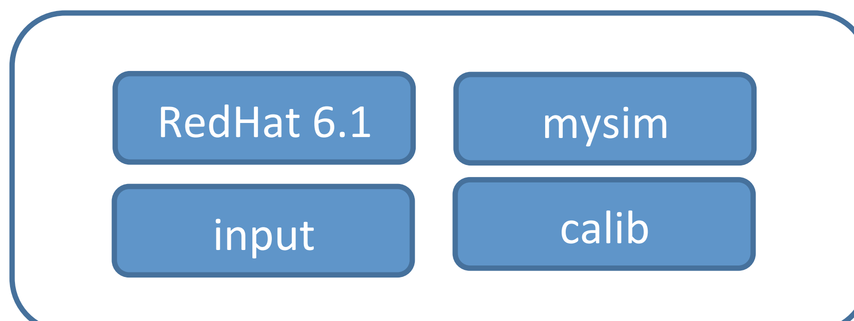
```
kernel: { name: “Linux”,  version: “3.2”; }
opsys: { name: “Red Hat”,  version: “6.1” }

software: {
        mysim: {
                url:          “doi://10.WW/ZZZZ”
                mount:    “/soft/sim”,
        }
}
data:  {
        input: {
                url:          “http://some.url”
                mount :   “/data/input”,
        }
        calib:  {
                url:          “doi://10.XX/YYYY”
                mount:    “/data/calib”,
        }
}
```

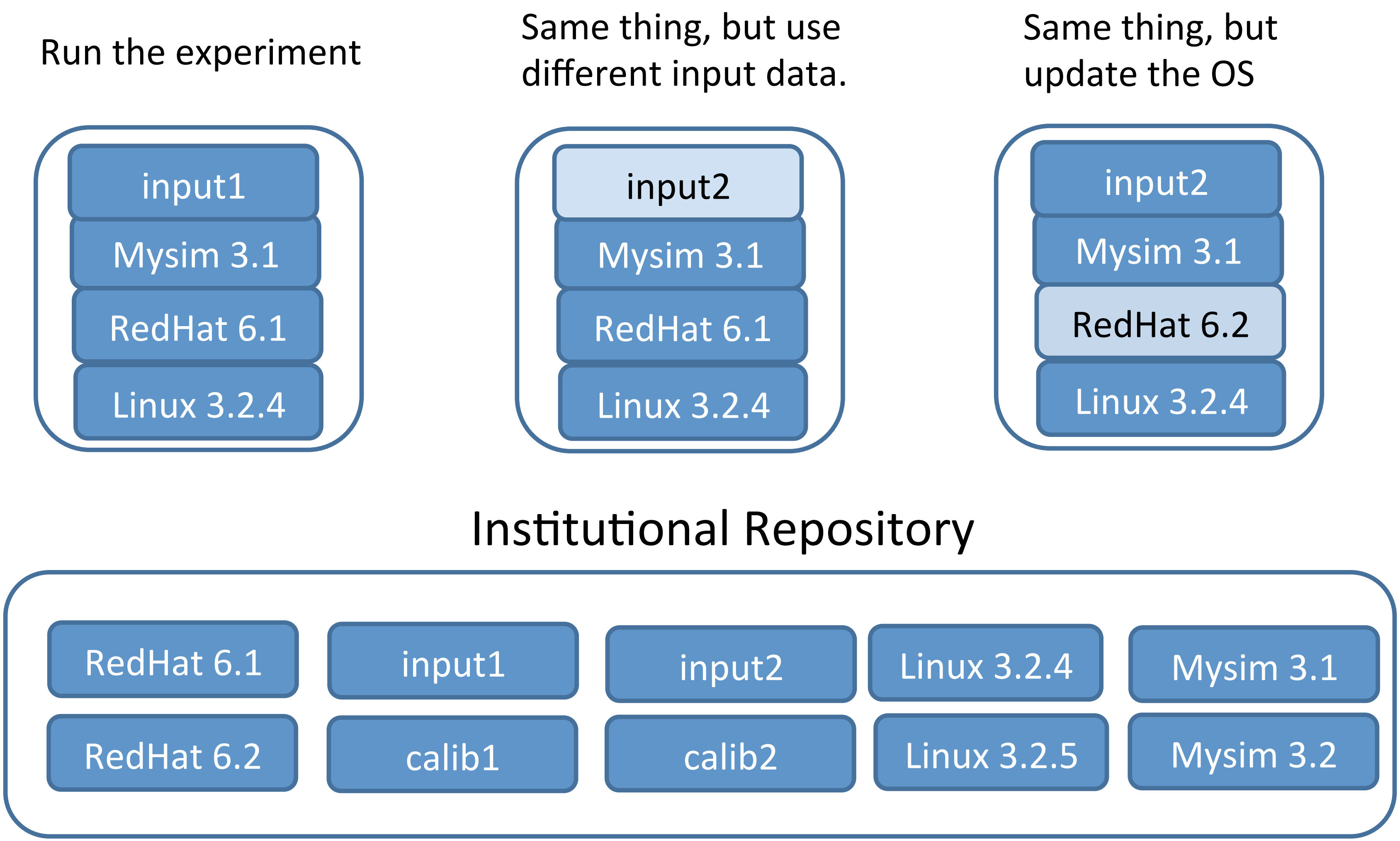
“umbrella run mysim.json”



Online Data Archives



Umbrella specifies a reproducible environment while avoiding duplication and enabling precise adjustments.



Haiyan Meng and Douglas Thain, **Umbrella: A Portable Environment Creator for Reproducible Computing on Clusters, Clouds, and Grids,** *Virt. Tech. for Distributed Computing, June 2015.*

Specification is More Important Than Mechanism

- Umbrella can work in a variety of ways:
  - Native Hardware: Just check for compatibility.
  - Amazon: allocate VM, copy and unpack tarballs.
  - Docker: create container, mount volumes.
  - Parrot: Download tarballs, mount at runtime.
  - Condor: Request compatible machine.
- More ways will be possible in the future as technologies come and go.
- Key requirement: Efficient runtime composition, rather than copying, to allow shared deps.

Encourage Cleanliness:

Construct workflows from carefully specified building blocks.

Encouraging Cleanliness: PRUNE

- Observation: The Unix execution model is part of the problem, because it allows implicit deps.
- Can we improve upon the standard command-line shell interface to make it reproducible?
- Instead of interpreting an opaque string:

  mysim.exe –in data –out calib
- Ask the user to invoke a function instead:

  output = mysim(input, calib) IN ENV mysim.json

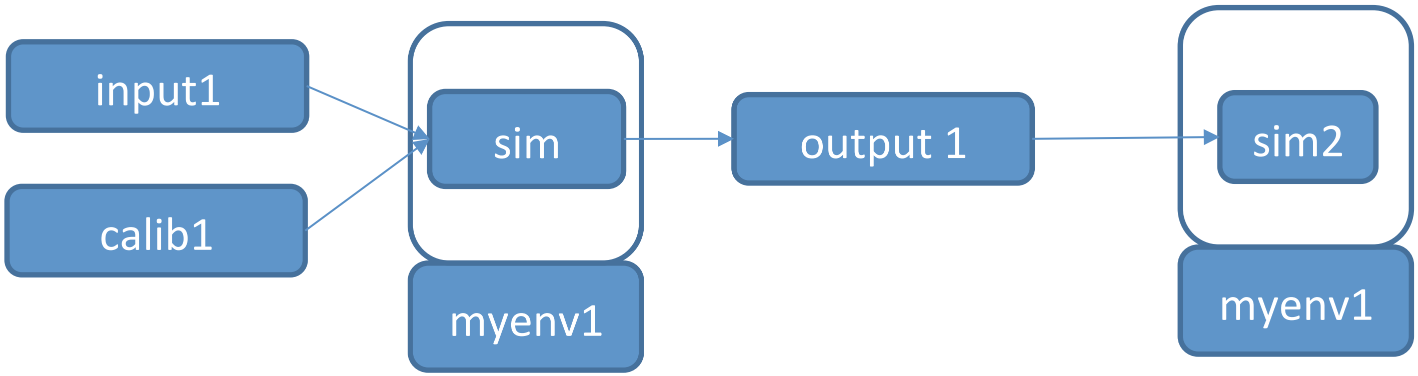
PRUNE – Preserving Run Environment

PUT “/tmp/input1.dat” AS “input1” [id 3ba8c2]
PUT “/tmp/input2.dat” AS “input2” [id dab209]
PUT “/tmp/calib.dat” AS “calib” [id 64c2fa]
PUT “sim.function” AS “sim” [id fffda7]

out1 = sim(input1, calib) IN ENV mysim.json
[out1 is bab598]
out2 = sim(**input2**, calib) IN ENV mysim.json
[out2 is 392caf]
out3 = sim(input2, **calib2**) IN ENV **bettersim.json**
[out3 is 232768]

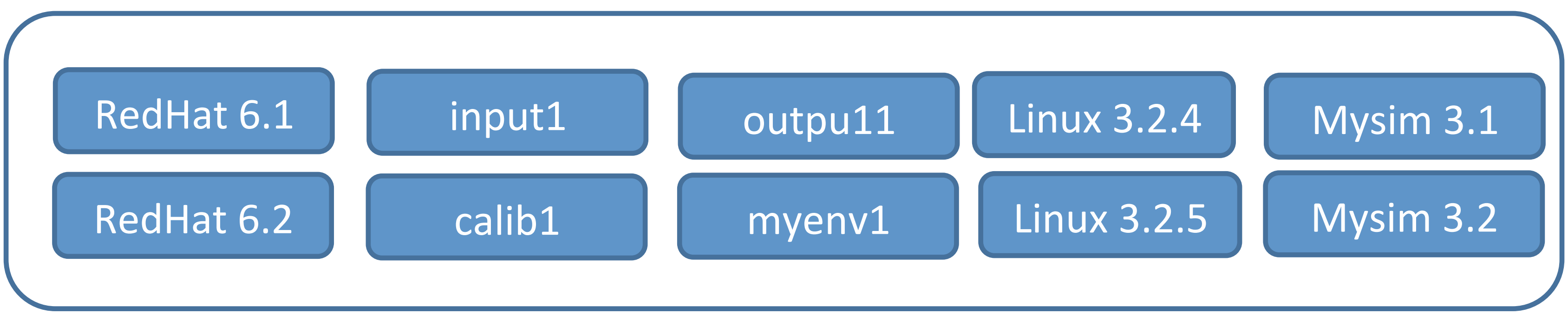
PRUNE connects together precisely reproducible executions and gives each item a unique identifier

output1 = sim(input1, calib1) IN ENV mysim.json

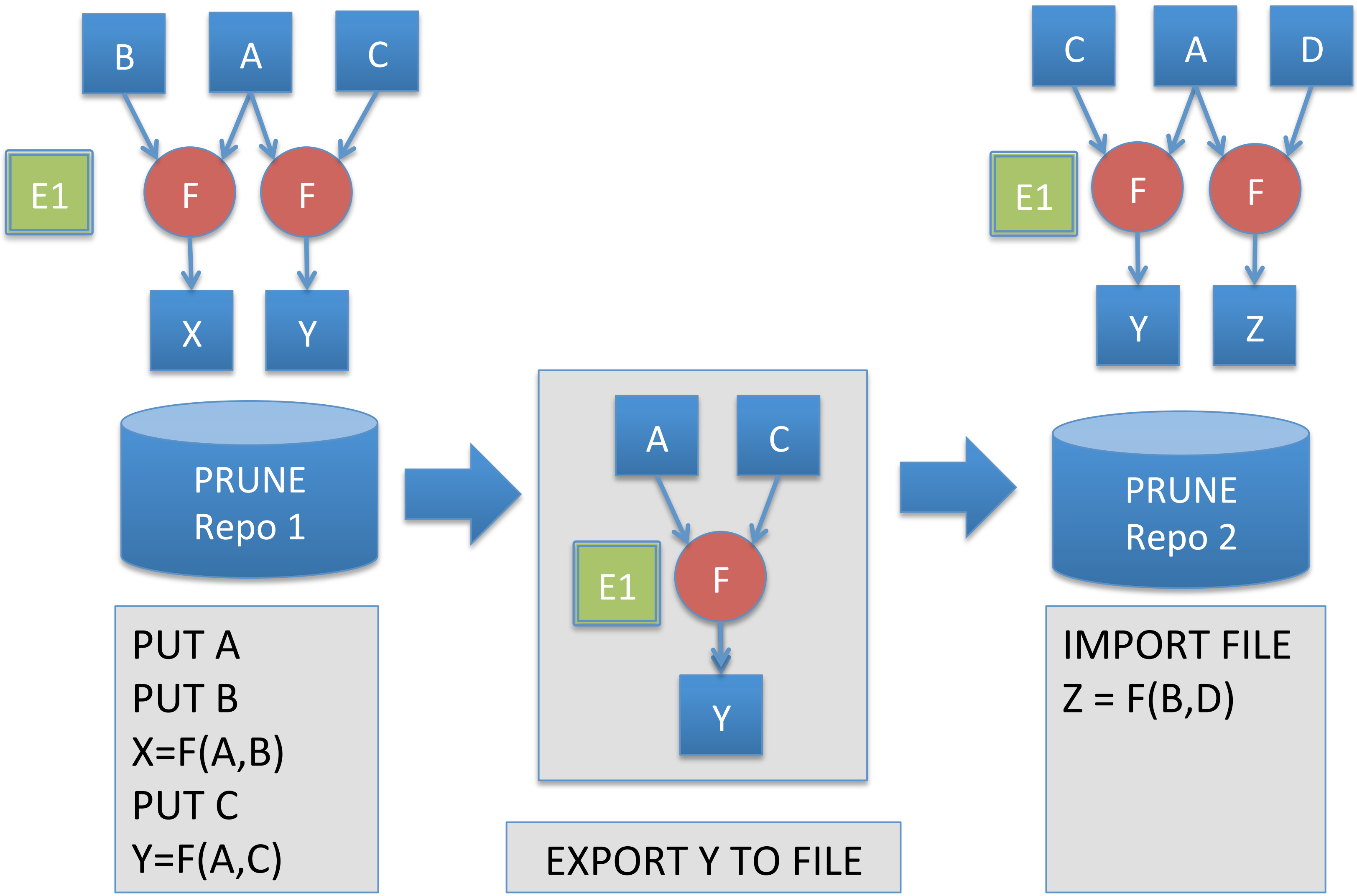


bab598 = fffda7 (3ba8c2, 64c2fa) IN ENV c8c832

Online Data Archive



PRUNE is like Version Control for Workflows



Scientific Reproducibility is also a Social Problem

- Do we reward researchers that provide detailed descriptions of their work.
- Do we insist that publications reveal their configurations in a rigorous way?
- Do we provide resources for archiving and using shared configurations?
- Technology can help, but there must be appropriate incentives.

Recapitulation

- Key problem: User cannot see the **implicit dependencies** that are critical to their code.
- Preserve the Mess:
  – VMs: Just capture everything present.
  – Parrot+Packaging: Capture only what is actually used.
- Encourage Cleanliness:
  – Parrot+CVMFS: Access deps over the network.
  – Umbrella: Describe all deps of a single execution.
  – PRUNE: Like version control for workflows.

Advice on Designing for Reproducibility

- Start with a clean slate.
  (Clean filesystem, empty environment, etc.)
- Use explicit reference to dependencies.
  (Prefer command line args over environment vars.)
- Do not permit unused dependencies.
  (Otherwise dep lists grow without bound.)
- Separate the logical and physical namespaces.
  (Otherwise you cannot move things around.)
- Preserve dependencies ***before*** using them.
  (Otherwise you will forget to preserve them.)

Many Open Problems!

- Naming: Tension between usability and durability: DOIs, UUIDs, HMACs, . . .
- Overhead: Tools must be close to native performance, or they won't get used.
- Usability: Do users have to change behavior?
- Layers: Preserve program binaries, or sources + compilers, or something else?
- Repositories: Will they take provisional data?
- Compatibility: Can we plug into existing tools?
- Composition: Connect systems together?

Acknowledgements

Notre Dame CMS Team

- Anna Woodard
- Matthias Wolf
- Chales Mueller
- Nil Valls
- Kenyi Hurtado
- Kevin Lannon
- Michael Hildreth

NSF Grant ACI 1148330: “Connecting Cyberinfrastructure with the Cooperative Computing Tools”

CCL Team

- Ben Tovar
- Peter Ivie
- Patrick Donnelly

Center for Research Computing

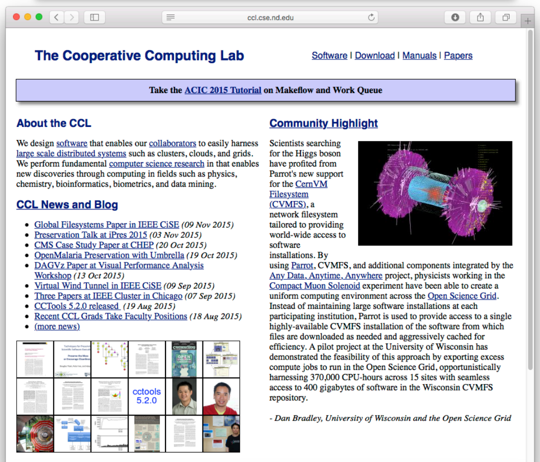
- Paul Brenner
- Sergeui Fedorov

HEP Community

- Jakob Blomer – CVMFS
- David Dykstra - Frontier

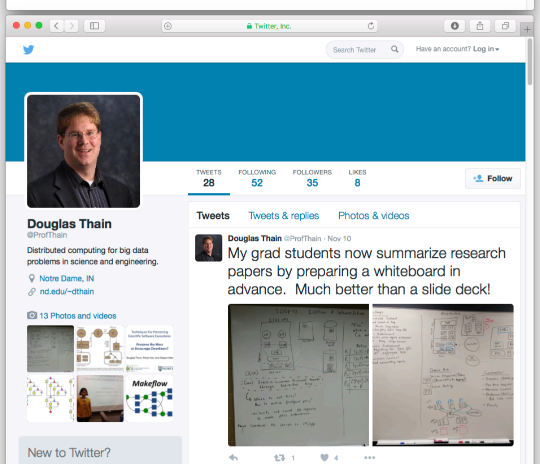
Data and Software Preservation for Open Science

http://www.daspos.org



The Cooperative Computing Lab

http://ccl.cse.nd.edu



Prof. Douglas Thain

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

@ProfThain