# More Sustainable HTC Computing

## How to better utilize compute resources.

NAF Admin Team
Christoph Beyer, Kruno Sever, Martin Flemming, Thomas Hartmann
DESY IT

# What do you want to use the NAF for?

## What is your User Story

- I am ...
- at group ...
- with background ...
- and have a long term goal ...
- for that I need to reach a short term goal ...
- And for which I need to do ...
- that requires ...

- and also I want to mention ...

# What is the NAF

# Storage & Compute

## Data Centric Science

- Data entails Storage
- Data gets processed on Compute

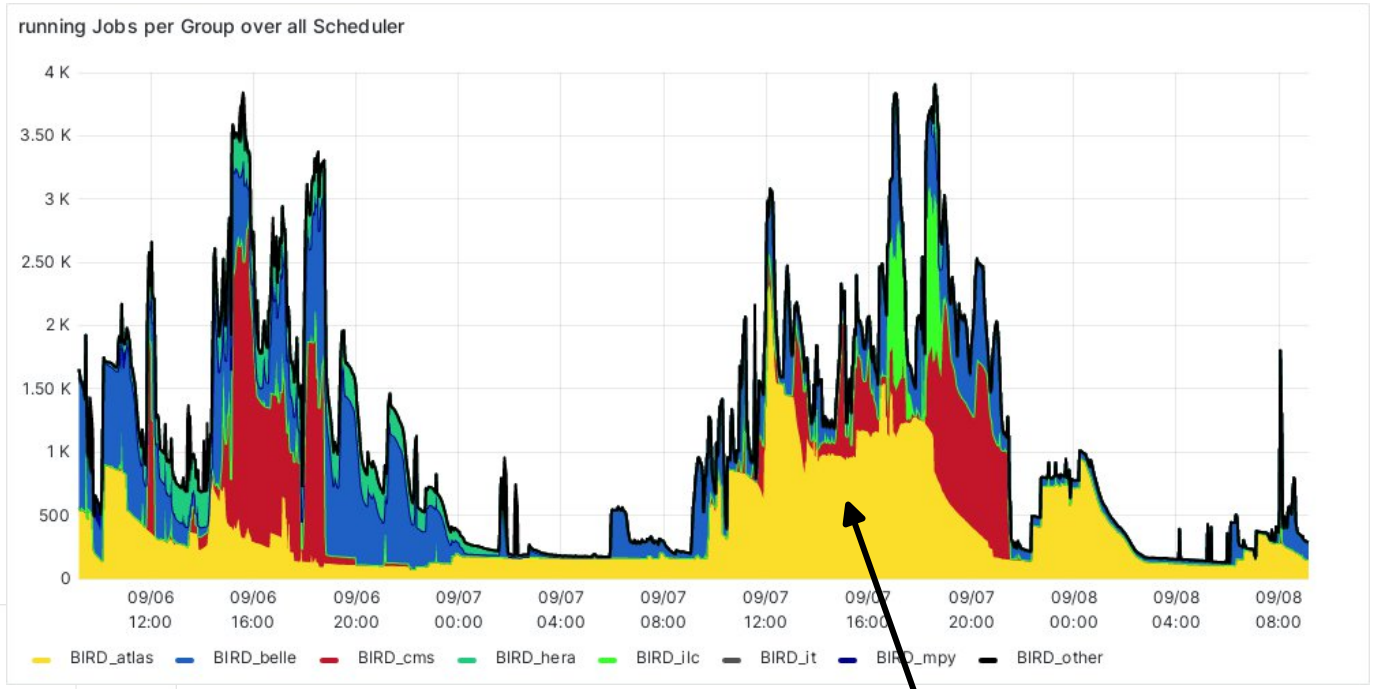
- Storage and Compute integral to each other

- You use in the NAF
  - **HTCondor** to process Data
  - The data reside on Storage like **dCache** or **DUST**
  - with tools as files on local disk and CVMFS

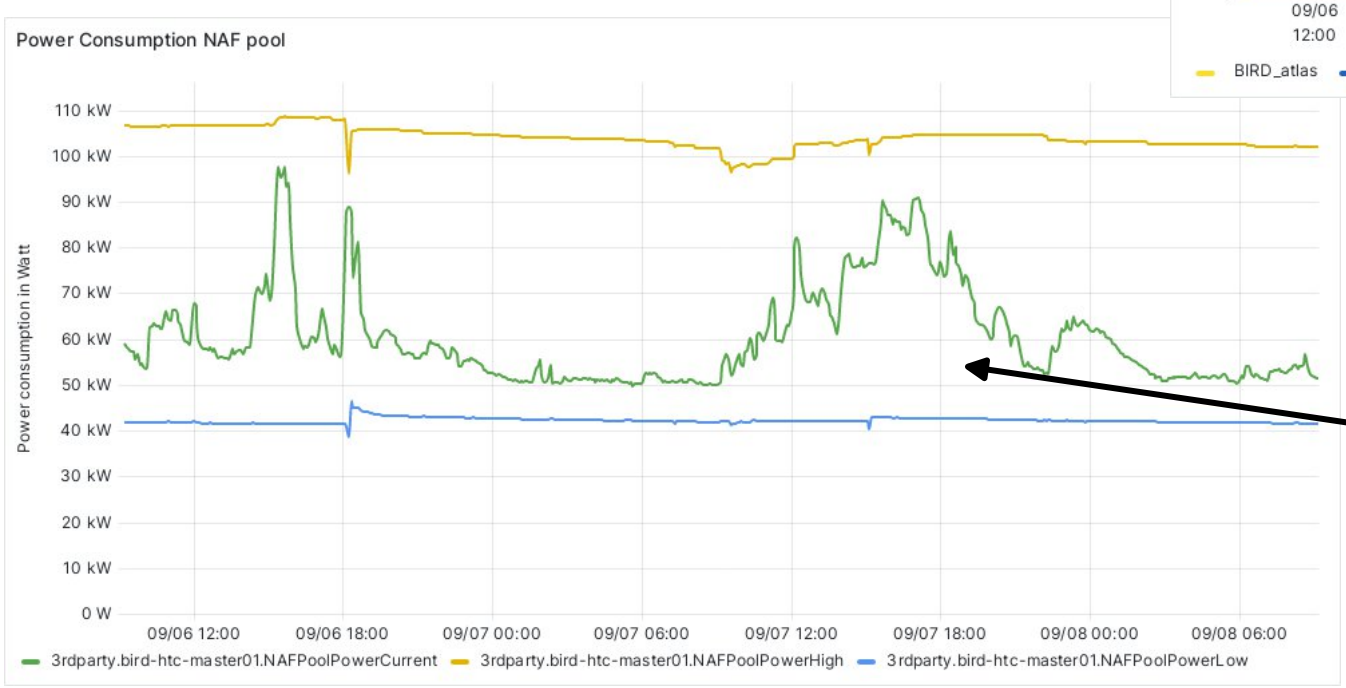
# Your have an Energy Footprint

## Compute and Storage consume significant Wh

- NAF base load ~40 kWh
  - idling around to keep it available for you
  - idling resources are wasted resources
    - HTC vs HPC
- 1 Core under load ~ + 5-7W




you




your compute energy footprint

# Your Footprint is Larger than CPU

## Storage, Network, Cooling

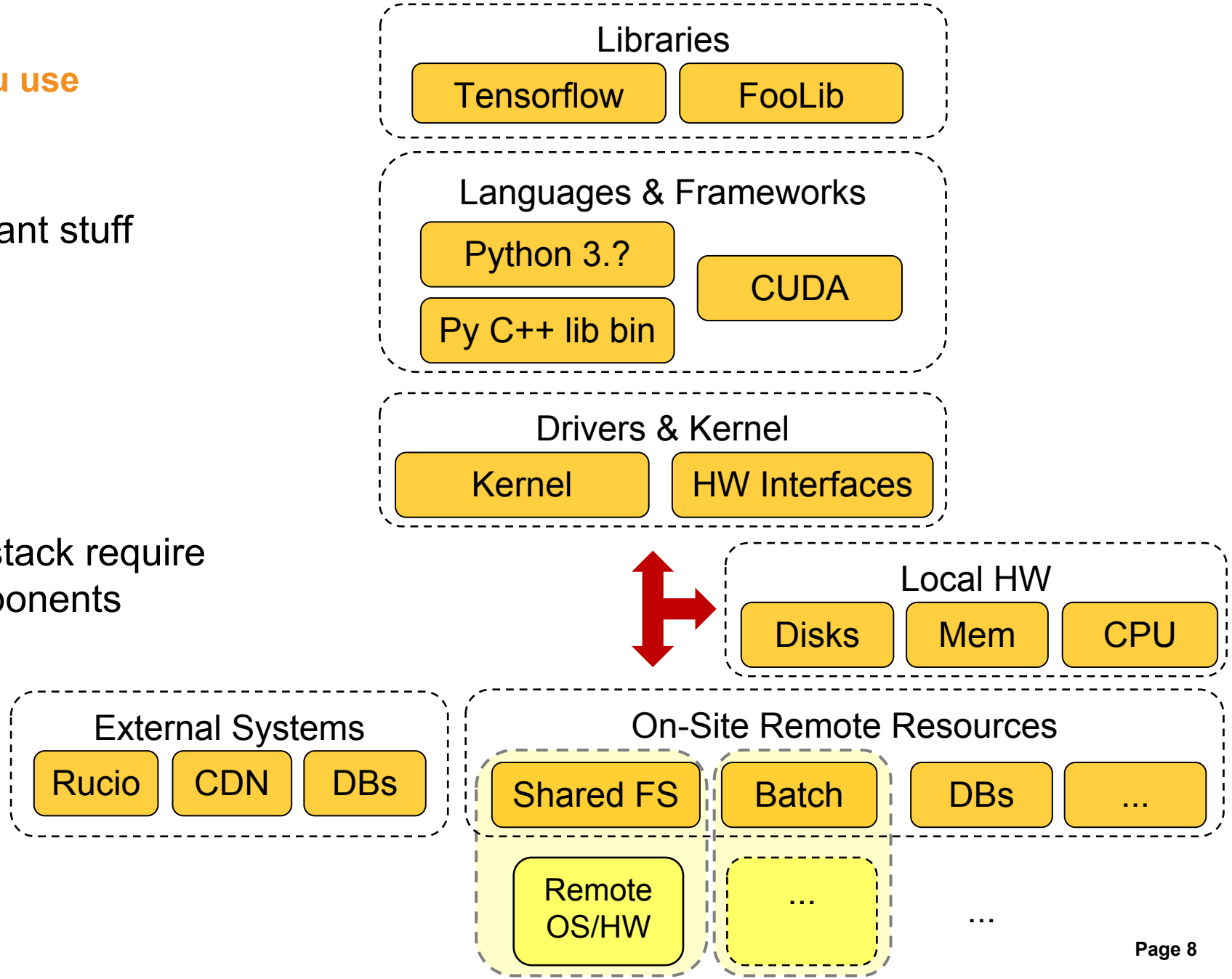
- Disk Storage: ~5Wh / 1TB
  - Tape: significant less operating power consumption
    - not for the end user...
- Network: ???
- Each Watt becomes Heat
  - Cooling
    - PUE ~1.2-1.4 for a non-optimized Computer Centre
    - By Thumb: add 20% Cooling to each Watt

# General Concepts

# Know Your Stack

## Understand the abstractions You use

- Abstractions are good
  - Focus and enable the relevant stuff
  - Hide the complexity
- Abstractions are bad
  - Hide the complexity
  - Efficient utilization of a full stack require an idea of all involved components
- **Full Stack Analyst**
  (ideally)



# Know Your Namespaces

## Understand in what context you are operating

- Learn to orientate yourself in namespaces
  - In what context is the thing?
  - Where is the thing?
  - And how is it called there?

- Namespaces tend to be inherently local (might be a bit strong statement)
  - Mapping between namespaces can be error prone
  - Global consistent namespaces are much, much harder than you thing

**Example:** Linux containers

Ordinary Processes jailed in their own various namespaces

**Example:** user name & resource access

DESY user != CERN user != ...

**Example:** file “U”RLs

A file has a locator in Rucio,
another one in the NAF mount namespace
and another one in the OS

# Know Your Allocations

## Understand where your actual resources are realized

- Resources can be hidden behind abstractions
- Addressing also consumes resources
  - A file has space allocated on a physical medium(s)
    - Realize the properties of the medium (RAM, SSD, HDD, tape,...)
  - Meta data consume resources
    - Large number of small files can exhaust significant resources
- Multi-threading can be sensitive to the location of memory wrt to the involved cores (NUMA/CPU cache levels, MPI,...)

**Example: file path**

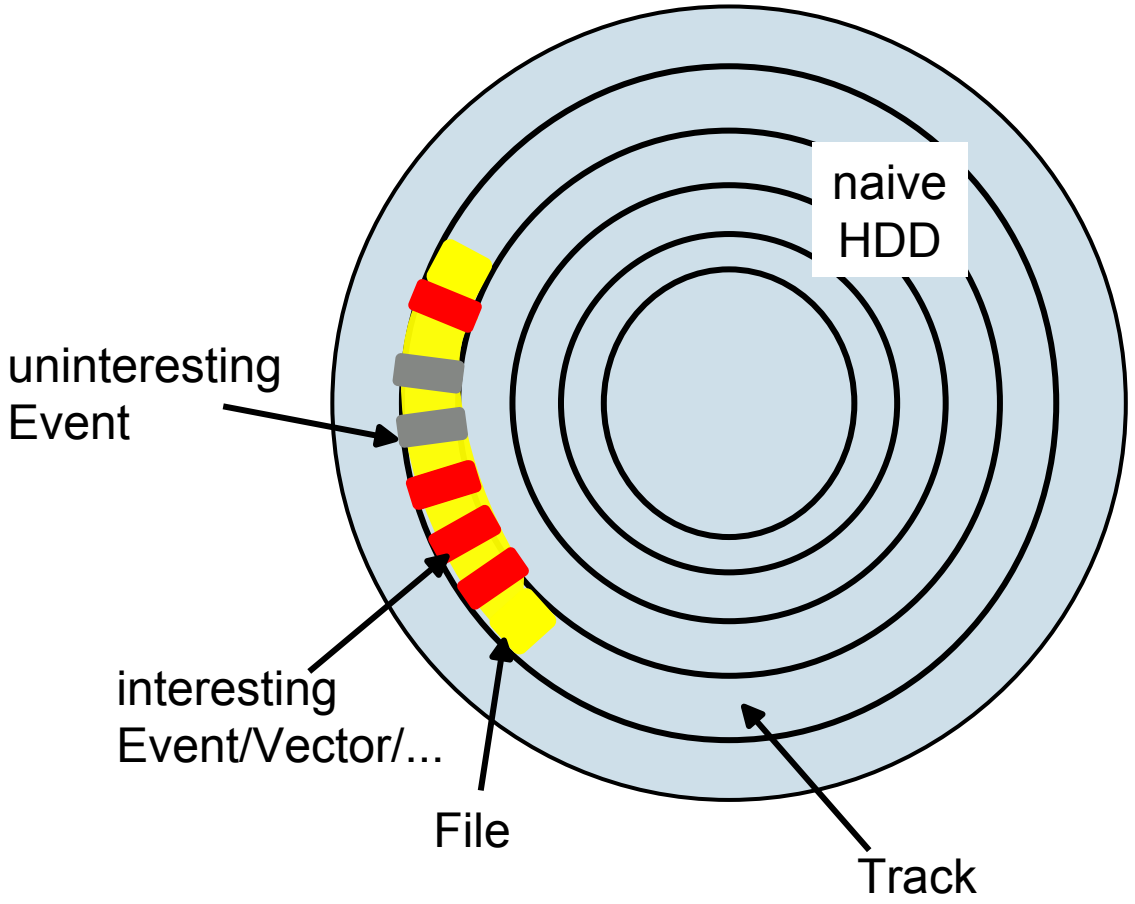
A file under a path
/pnfs/desy.de/foo/file.txt
has (currently) its bits on a disk on one of many pool nodes in the foo instance

A file under path
/nfs/dust/foo/file.txt
has its bits erasure encoded spread over a number of GPFS nodes

# Example: Event File Access

## ROOT Ntuple, HDF5 and similar file containing “events”

- Files as (ideally sequentially) blocks on a track
  - Physics *might* differ: RAID, Erasure Enc,...
- Files consisting of sub-elements ala Events
  - Event ~= Vector (at least in HEP)
    - (De)Serialization?
    - $x_1\ y_1\ z_1\ t_1\ x_2\ y_2\ z_2\ x_3\ t_1$... ?
    - $x_1\ x_2\ x_3\ y_1\ y_2\ y_2\ z_1$... ?
- Who reads what in what order?
  - (read ahead) prefetch caches involved?
    - ROOT: TTreeCache
    - File System




- **Take Away: (Quasi-) Random Access is deadly for HDD performance**

# File System/Storage Usage

## Dos & Dont's

(all coming with small print)

- Avoid unnecessary file I/O traffic 
  - Staging a file from a shared FS into each job 
- Replicate files, that are accessed in parallel 
  - Reading from dCache a single file, that lies on a single disk, from 1000 jobs 
  - E.g., a small condition file staged from the submit machine to each job's transient local space 
- In file seek and writes for files on (quasi-) WORM media 
  - Changes within a files on dCache or object stores cause a new file 
  - Using local storage or DUST for such operations and keeping files for long term to dCache 
- Waste space on expensive storage like DUST leaving unused files for month 
- Organize your data for easy retrieval 

# Implicit and Explicit Knowledge

## How to know, what is reasonable, efficient and sustainable

- User Question?
  - How do I know, if the things I do are *reasonable*?
- Unfortunately, no direct feedback mechanism
  - You learn only about inefficient usage, when an admin (y/t)ells
    - Unfortunate for a responsive development
    - But intended to shield from unnecessary constrains
- Counter World: Cloud Providers
  - Immediate and merciless response on your credit card

# That all said

You do not want to become IT specialists

**You want to do Physics**

You do not need to know all the nitty-gritty details
(but knowing them does no harm...)

# HTCondor Batch System

# Condor Job Details

## How does Condor work?

- No concept of a "***Queue***" as such
- Users ask for resources
- Worker Nodes offer resources
- Condor brokers between requests and offers
- ~~> classified advertisement

**User Ad:**
Looking for someone running my job
for 2 hours
with 16GB memory
and 8 cores

**Worker Ad:**
I have currently
56GB Memory
and 23 cores free.

Who wants to use them?

# Condor Submission Ads

## Submitting a job ~= preparing an ad requests

your CPU & memory needs ads

**User Ad:**
Looking for someone running my job
for 2 hours
with 16GB memory
and 8 cores

**Submission File Example**

```
Request_Cpus          = 8
Request_Memory        = 16384
+RequestRuntime       = 7200

Requirements    = ( OpSysAndVer == "CentOS7" )

Executable = /nfs/dust/foo/myjobs/myapp
Log          = /nfs/dust/foo/myjobs/log.d/$(Cluster).log
Output       = /nfs/dust/foo/myjobs/output.d/$(Cluster).out
Error        = /nfs/dust/foo/myjobs/error.d/$(Cluster).err
should_transfer_files   = false
Transfer_Executable     = false

Queue 1
```

more specific requirement ads go here

ad for the program/script to be executed

ads where logs, stdout, stderr should go
**these directories have to exists!**

ads to tell Condor not to copy input and result files (as DUST is used here)

queue one job

**Quiz:**
What does this mean for storage I/O?

# Hello World Job on the NAF

## Create a file helloworld.submit on your workgroup server

### window side

```
Executable    = /usr/bin/sleep
Arguments     = 240
Request_Cpus  = 1
Request_Memory = 1024
+RequestRuntime      = 300

output       = myjob_$(Cluster)_$(Process).out
error        = myjob_$(Cluster)_$(Process).error
log          = myjob_$(Cluster)_$(Process).log

queue 1
```

### door side

```
Executable    = /usr/bin/stress
Arguments     = --cpu 3  --vm 3 --timeout 240
Request_Cpus  = 1
Request_Memory = 1024
+RequestRuntime      = 300

output       = myjob_$(Cluster)_$(Process).out
error        = myjob_$(Cluster)_$(Process).error
log          = myjob_$(Cluster)_$(Process).log

queue 1
```

**Submit to Condor with**

```
condor_submit helloworld.submit
```

**Quiz:** Where is the executable?

# Interactive Hello World

## Debugging your jobs on Worker Nodes

**Create a small file on your workgroup server**

touch /tmp/$(whoami).moin

**Submit to Condor as interactive job**

condor_submit –i helloworld.submit

(wait a moment)

Where are you “now”?

What do you see in /tmp ?

How long does this command take?
time md5sum /cvmfs/atlas.cern.ch/repo/containers/images/singularity/x86_64-slc6.img

Try to run directly the example the slide before

# Test Your Jobs

## Wasted jobs are the most ineffective

- Before submitting jobs for 1000h of CPU time
- Validate your job and executable
  - Does the job itself run?
  - Do all paths and directories exist?
  - Are the jobs within the run time limits (plus a safety margin) ?
  - Do your results look sane?
  - Do you have included all output variables?

- And before everything else, dry rub the submission itself in

  ```
  > condor_submit condor.submit -dry -
  ```

# Submitting many Jobs

## Clusters of Jobs

HTCondor Documentation

**Submitting 100x the same job**

```
...

Executable = /nfs/dust/foo/myjobs/myapp
Log    = /nfs/dust/foo/myjobs/log.d/my_$(Cluster).$(Process).log
Output = /nfs/dust/foo/myjobs/output.d/my_$(Cluster).$(Process).out
Error  = /nfs/dust/foo/myjobs/error.d/my_$(Cluster).$(Process).err
...
Queue 100
```

**Jobs in the NAF**

Cluster and Proc IDs
12345678.[0..99]

Each with output files ala

/nfs/dust/foo/myjobs/log.d/my_12345678.73.log

# Submitting many Jobs

## Different Job Inputs

**Submitting 100x the same job with different inputs**

```
...

Initialdir = MYTASKDIR/job.$(PROCESS)
Executable = myapp

Log    = /nfs/dust/foo/myjobs/log.d/my_$(Cluster).$(Process).log
Output = /nfs/dust/foo/myjobs/output.d/my_$(Cluster).$(Process).out
Error  = /nfs/dust/foo/myjobs/error.d/my_$(Cluster).$(Process).err
...
Queue 100
```

**Job Submission Side**

- MYTAKSDIR/job.0/myapp
- MYTAKSDIR/job.1/myapp

...

- MYTAKSDIR/job.99/myapp

**Quiz:**
Where can scaling out affect resources?

# Efficient Job Submission

## Use Job Clusters whenever possible

- Individual job loops 👎
  - foreach x in $(ls *.dat)
    do condor_submit -….. $x
    👎
- let Condor loop over your input files:

**Go over a dir in 'foreach-style'**

```
<begin submit file>
< usual stuff like executable, logfiles etc.>
Args = $(Item)
queue 1 Item matching files (*.dat)
<end submit file>

In the submit dir:

ls *.dat
1.dat  2.dat  3.dat  4.dat


In the executable:

MY_INPUT_FILE = $1

<do something with input file>

In the queue:

[chbeyer@htc-it01]~/htcondor/testjobs% condor_submit sleep_foreach.submit Submitting job(s)....
4 job(s) submitted to cluster 2203711.

[chbeyer@htc-it01]~/htcondor/testjobs% condor_q -nobatch
-- Schedd: bird-htc-sched01.desy.de : <131.169.56.32:9618?... @ 06/29/18 12:05:07
 ID         OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
2203711.0   chbeyer         6/29 12:04   0+00:00:00 I  0    0.0 sleep_args.sh 1.dat
2203711.1   chbeyer         6/29 12:04   0+00:00:00 I  0    0.0 sleep_args.sh 2.dat
2203711.2   chbeyer         6/29 12:04   0+00:00:00 I  0    0.0 sleep_args.sh 3.dat
2203711.3   chbeyer         6/29 12:04   0+00:00:00 I  0    0.0 sleep_args.sh 4.dat
```

# Efficient Job Submission

## Supply Job Arguments from a file

**Go through a list and reads the arguments from every line**

```
[chbeyer@htc-it02]~/htcondor/testjobs% cat list.txt
60, eine
120, zwei
180, drei
240, vier


The submit file:
Args = $(var1) $(var2)
queue var1,var2 from list.txt


In the queue:
-- Schedd: bird-htc-sched02.desy.de : <131.169.56.95:9618?... @ 07/17/18 11:45:24
 ID          OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
6253267.0   chbeyer         7/17 11:45   0+00:00:00 I  0    0.0 sleep_args.sh 60 eine
6253267.1   chbeyer         7/17 11:45   0+00:00:00 I  0    0.0 sleep_args.sh 120 zwei
6253267.2   chbeyer         7/17 11:45   0+00:00:00 I  0    0.0 sleep_args.sh 180 drei
6253267.3   chbeyer         7/17 11:45   0+00:00:00 I  0    0.0 sleep_args.sh 240 vier
```

# Efficient Job Submission

## Supply Job Arguments from a script

**Using a script for the input of arguments, each line of the scriptoutput is treated as an item**

```
[chbeyer@htc-it01]~/htcondor/testjobs% cat items.sh
#!/bin/bash
ls -1 *.dat
[chbeyer@htc-it01]~/htcondor/testjobs% ./items.sh
1.dat
2.dat
3.dat
4.dat


The submit file:
Args = $(Item)queue from ./items.sh |
```

# Efficient Job Submission

## Being nice to Condor

- MANY Jobs ~ 10000+
  - use late materialization
  - generator-like job clusters
  - Example: https://confluence.desy.de/display/IS/large+number+of+jobs

- Use Condor internals when possible
  - https://chtc.cs.wisc.edu/uw-research-computing/multiple-jobs

# Job not running?

# Job/Worker Matchmaking Procedure

## (Never) Fair Shares and Quotas

- **Reminder:** Condor matching Job Ads with Worker Resource Ads
- Each Group has “their” share on the NAF
  - ATLAS XX%, Belle YY%, CMS ZZ%,...
  - Condor balances only between the Groups!
  - Group internal utilizations are group internals!

- Jobs without own requirements == “lite” default jobs: 1core/2GB/3h

**What are the current priorities per group & user**
condor_userprio
(smaller prio is better)

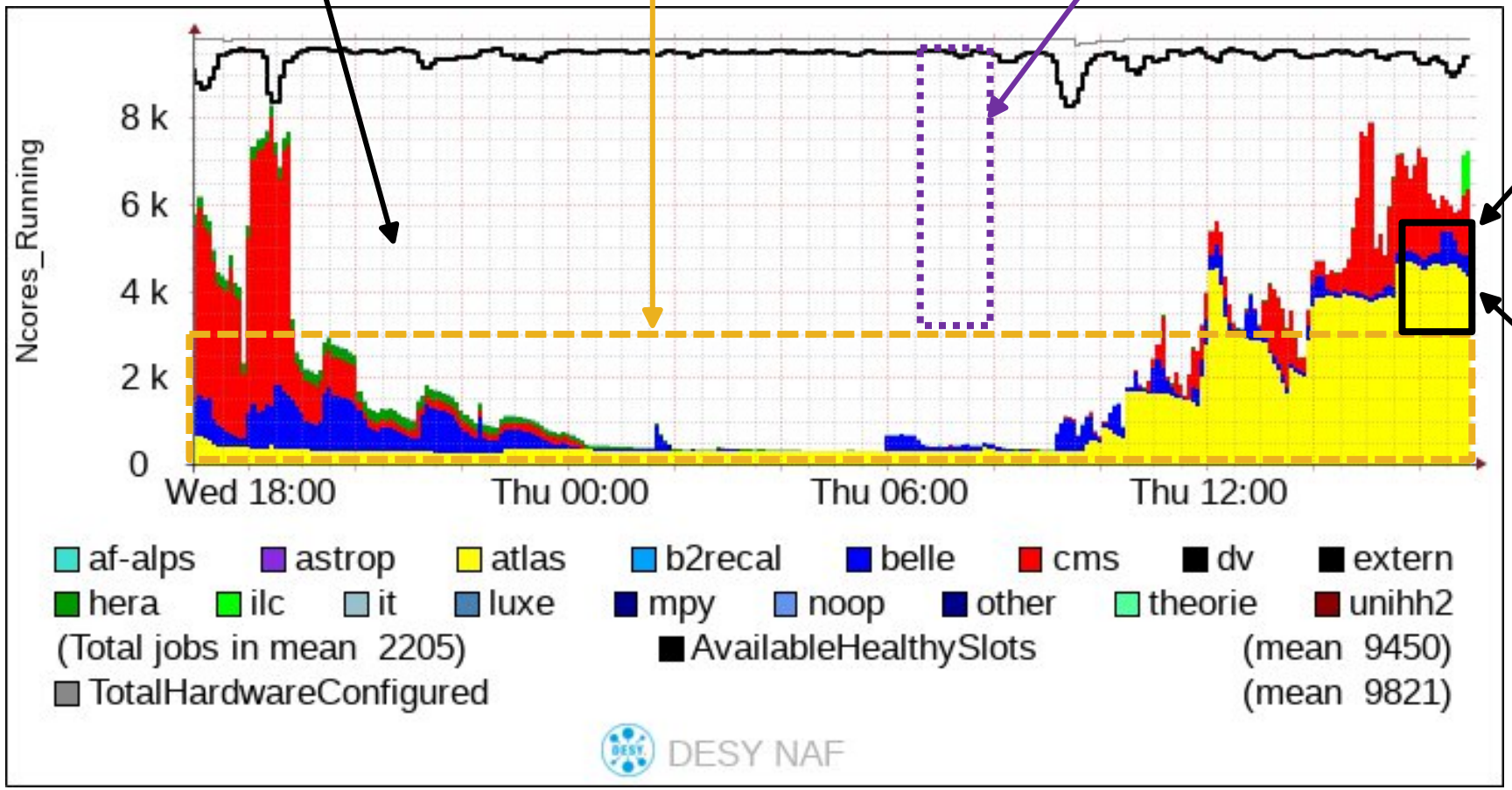
# Opportunistic Usage

## Short Jobs on other Group Shares

unused resources
= wasted resources

Group A's nominal share on the NAF

How to utilize free resources?
(w/o affecting other groups)




allow only short jobs to jump on other groups' nominal shares!

"*lite*" jobs 1core/1GB/3h

~ small enough

"*rightful owners*" get "*their*" shares back in time

# Why is my job still not starting?

## Debugging jobs in idle states

- Is the NAF full?
  - If full, how is my priority compared to all others
  - Has my group used up their share?
  - Do I request non-lite resources?
    - no jumping on other groups' shares
- Do I request *unusual* resources
  - has Condor to free resources for my slot?
- How many worker nodes can serve me?
- How do I fare against my group colleagues?

**What are the current priorities per group & user**
condor_userprio

**Which nodes match my Job?**
condor_q **-analyze** Cluster.Id

```
-- Schedd: bird-htc-sched12.desy.de : <131.169.223.40:9618?...

19528067.2497:  Run analysis summary ignoring user priority.  O
      0 are rejected by your job's requirements
    439 reject your job because of their own requirements
      0 match and are already running your jobs
      0 match but are serving other users
    258 are able to run your job
```

**Quiz:**
Why is this job not immediately getting a slot?

**Check for more details**
condor_q **-better-analyze** Cluster.Id

```
Job 19528067.2497 defines the following attributes:

    Base0Requirements = (TARGET.Arch == "X86_64") && (
sFileTransfer))
    Base1Requirements = Base0Requirements && (OpSysAnd
    Base2Requirements = Base1Requirements && (BIRD_Res
    FileSystemDomain = "desy.de"
    RequestDisk = 20480000
    RequestMemory = 4096
    RequestRuntime = 14400
    SysDefaultOS = "CentOS7"
```

# Understanding Jobs and Ads

# Job Debugging

## Understanding failing Jobs

- Job States in Condor
  - **idle**/pending : waiting for free resource slots
  - **run**/active : running on the assigned slot
  - **hold**/suspended : something went wrong, user needs to check

...

jobs do something 🤞

...

- **completed/done** : job has finished

**Quick Check on "Alive" Jobs:**

> condor_q { -idle / -run / -hold } USERNAME

**All Job Ads:**

> condor_q –long JOB.ID

**Selecting Specific Job Ads:**

> condor_q JOB.ID –af adname anotherad

```
> condor_q 12345678.0 -af JobStatus JobRunCount RequestMemory ResidentSetSize_RAW  HoldReason
5 1 1536 7463516 Memory usage higher than 3 x requested memory
```

**Finished Jobs:**

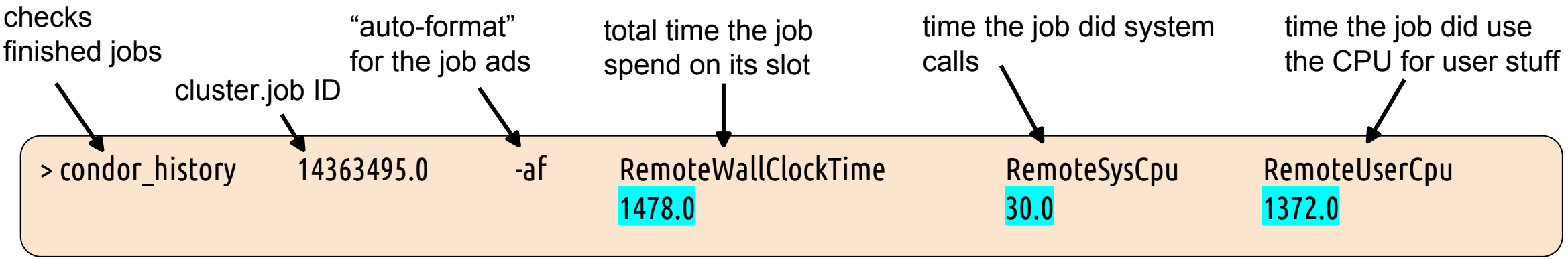
> condor_history

pretty similar to condor_q

# What is my Job Efficiency

## Analysing Jobs a posteriori

- How CPU efficient was my job? ~~> “How much time were my processes not idling around“
  - Need to know the overall time (in seconds) of the job on a node and how long it was actual active




**Unix Epoch to Human**

```
> date -d @1693482422
Thu Aug 31 13:47:02 CEST 2023
```

**Job Efficiency**

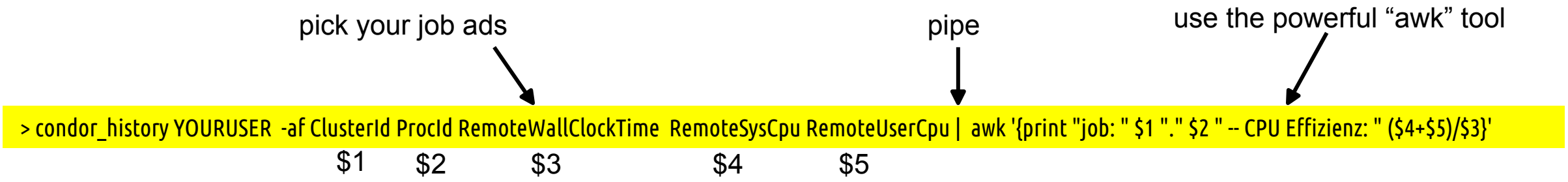
( 30.0 + 1372 ) / 1478 ~= 0.95

95% is pretty good
mostly CPU and hardly any waiting

# What is my Job Efficiency II

## Analysing Jobs

- More CLI – calculating on the shell




```
[hartmath@naf-cms30 ~]$ condor_history USRFOO  -af ClusterId ProcId RemoteWallClockTime  Re
moteSysCpu RemoteUserCpu |  awk '{print "job: " $1 "." $2 " -- CPU Effizienz: " ($4+$5)/$3}'
job: 14368495.0 -- CPU Effizienz: 0.895279
job: 14370804.0 -- CPU Effizienz: 0.961637
job: 14371954.0 -- CPU Effizienz: 0.88
job: 14372038.0 -- CPU Effizienz: 0.767442
job: 14368220.0 -- CPU Effizienz: 0.914023
job: 14369885.0 -- CPU Effizienz: 0.930719
job: 14370768.0 -- CPU Effizienz: 0.921569
job: 14368102.0 -- CPU Effizienz: 0.942817
```

# Check Your Hello World

## How has your test Job fared?

**Is the job still running?**

> condor_q

**or has it already finished?**

> condor_history $(whoami)

**Which one was the interactive job?**

condor_history $(whoami) -af clusterid procid JobDescription

**Get for a Job all its Ads**

condor_history Clusterid.Procid -l

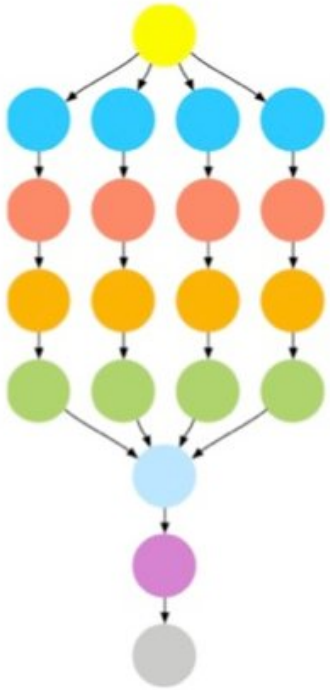
**How do you compare to other Users NAF Usage?**

condor_userprio

# Complex Workflows

## condor_dagman: directed acyclic graph

- DAGMan (Directed Acyclic Graph Manager) is a workflow management system based on graphs
- Complex dependencies of tasks can be realized in DAGs
- Define dependencies and let DAGMan manage them for you
- Linear and parallel computation models
- Resubmitting of incomplete jobs with one command



- For some examples see
  - https://bird.desy.de
  - https://swc-osg-workshop.github.io/OSG-UserTraining-AHM18/novice/DHTC/04-dagman.html

# Less known features in HTCondor

## Something to test on your own

- `condor_ssh_to_job <job-id>`
  - jumps right into the running job sandbox on the worker
- `condor_tail <job_id>`
  - see what a job writes to <stdout>
- `condor_chirp ulog " …"`
  - creates custom entry in the job log file
  - Example: https://confluence.desy.de/display/IS/Individual+job+status+and+return+codes
- Condor Python Bindings
  - use Condor natively from your Python code
  - powerful - but with great power comes a bit responsibility...
  - https://htcondor.readthedocs.io/en/latest/apis/python-bindings/index.html

# And much more

## Learn all the Condor knobs (if you have the time...)

**Quiz:**

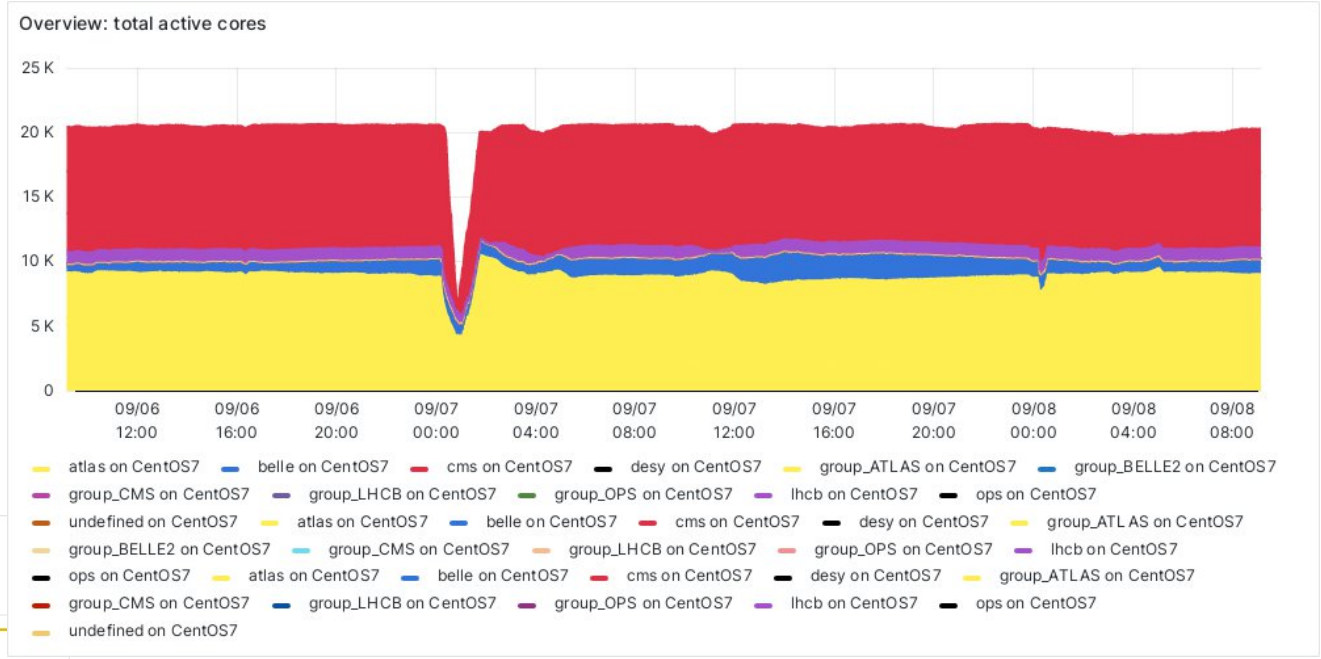
What Questions do You have?

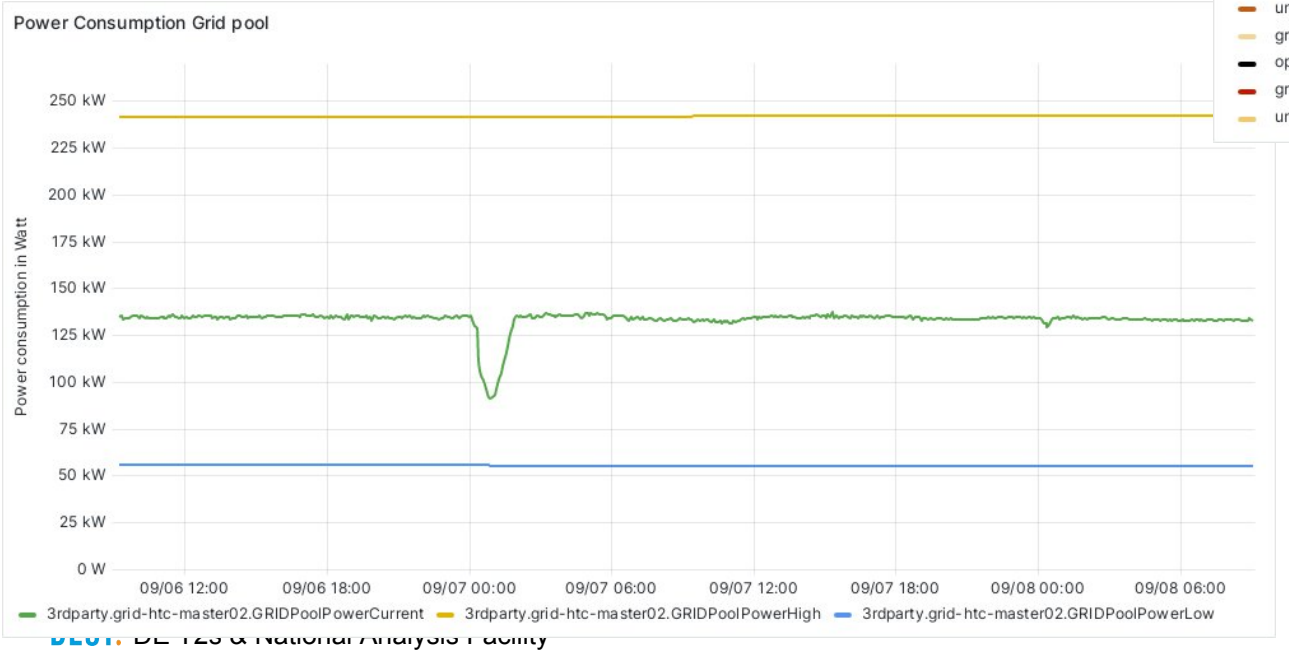
# Appendix

# Your Jobs have an Energy Footprint

## Grid

- HTC: maximize utilization
  - minimizes idle overhead waste
  - **HTC**ondor
- HPC: watch your (non-)utilization!
  - full node scheduling - all for yourself
  - including waste



# Know Your Physics and Sociology Bromides

## Understand what non-IT Constraints you have

- How do your actions realize in the physical reality?
  - What happens at the other end of an abstraction?

- Most things scales with money
  - Unfortunately, tends to be limited
  - You have to counterbalance funding constraints...

- Science and Tool Use is Sociology
  - Understand what drives and limit you as well as the system you are embedded

**Example:**
Reading from 1000 jobs in parallel the same file, that is stored on a single rotating platter on a HDD

**Example:**
A dedicated server with 384 cores would be ideal for W-studies
but costs €€€
~~> Have to improvise with shared compute clusters

**Examples:**
Documentation
Funding
Documentation
Environment
Documentation

# Storage Recap

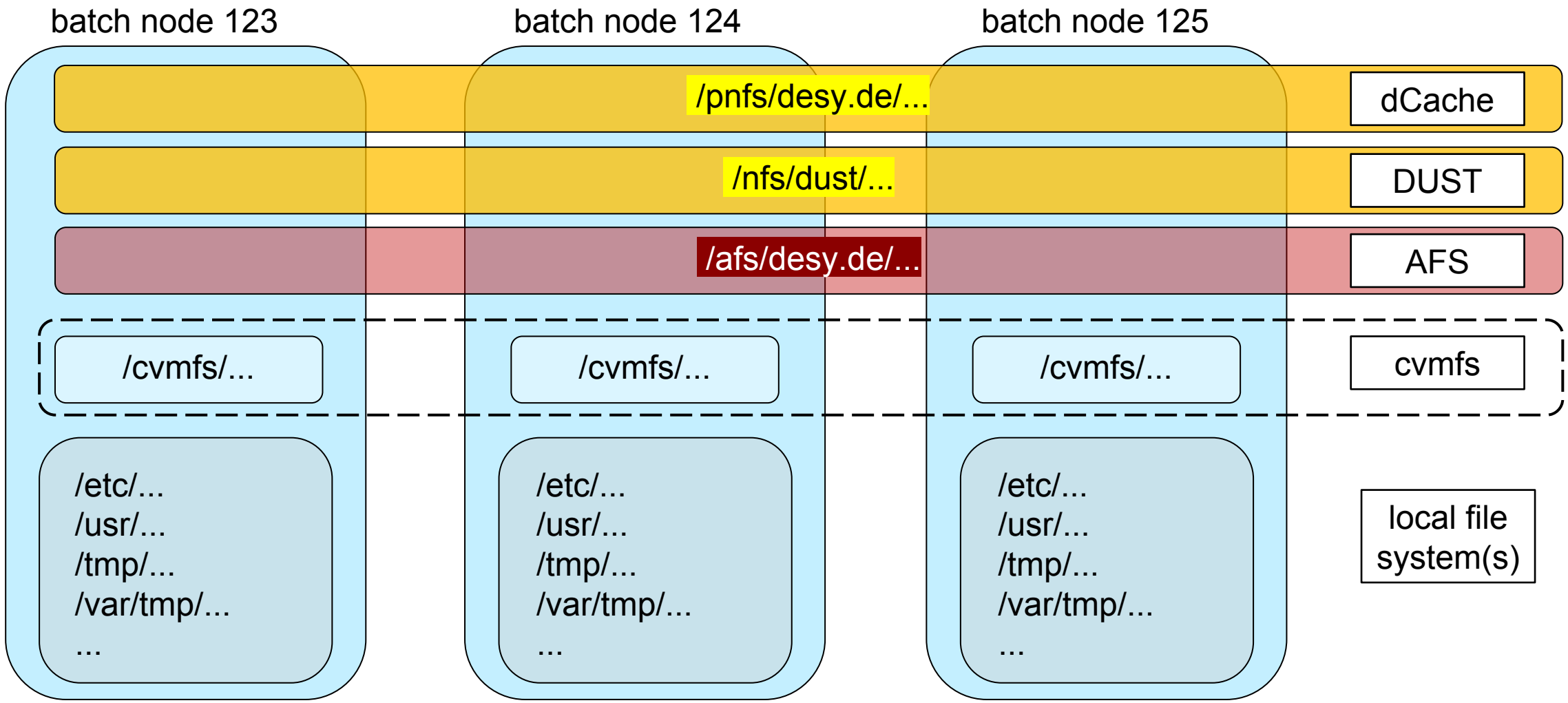
# File Systems in the NAF

## File Systems with Different Properties for different Use Cases

- HOME directory on AFS `/afs/desy.de/...`
  - Shared file system available on workgroup servers and workers nodes
  - In back up, (nominally) available globally
  - Not performant for heavy work or small files
- Scratch space on DUST (GPFS) `/nfs/dust/...`
  - Shared file system
  - Day to day work area
  - Performant for parallel use from multiple jobs
  - Expensive
  - No backup
  - Don’t let files rot - clean up!
- Long Term Storage on dCache `/pnfs/desy.de/...`
  - Shared file system - available also via Grid
  - Readable on the NAF
  - Writable through Experiment data management frameworks
- Local node space `/var/{tmp,spool}/..., /dev/shm,...`
  - On worker nodes – only in a job’s context
  - Most direct I/O
  - Should be the place to go for transient files
  - Stage results to the target file system

# File Systems

## Local and Shared Mount Namespaces



# Job Types in the HEP Community

## HTC and HPC clusters

- CPU oriented scheduling

- **H**igh-**T**hroughput **C**lusters prevalent
  - HEP events can be processed embarrassingly parallel
  - Optimal utilization of resources

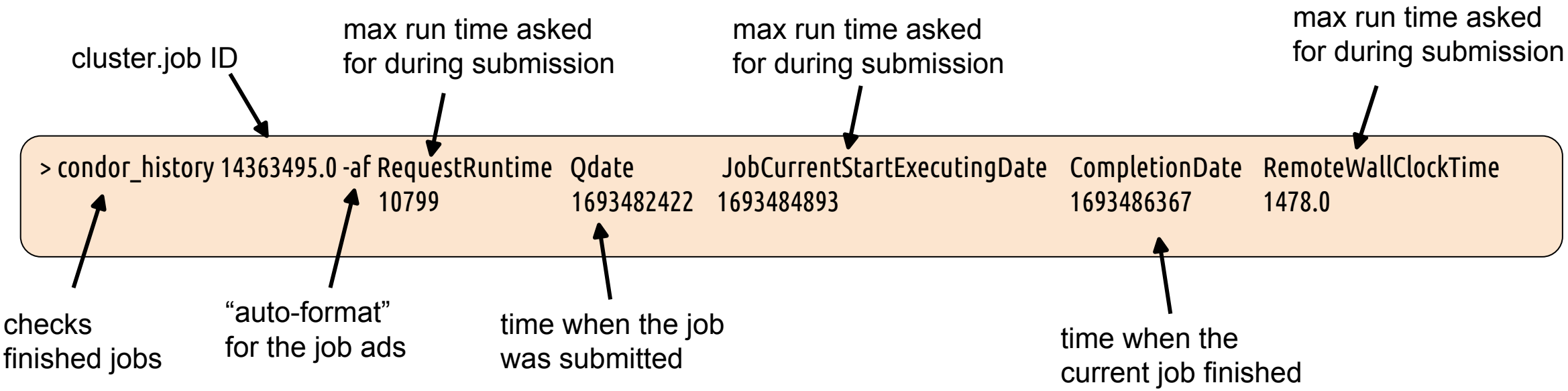
- **H**igh-**P**erformance **C**lusters opportunistically used by some groups

- “Memory Scheduling” like Apache Spark or DASK might become more complementary
  - Avoiding idling CPUs with Memory focused application might be an issue

# Job Details a posteriori

## Analysing Jobs

- How CPU efficient was my job? ~~> “How much time were my processes not idling around“
  - Need to know the overall time of the job on a node and how long it was actual active




**Unix Epoch to Human**

```
> date -d @1693482422
Thu Aug 31 13:47:02 CEST 2023
```