Handling of Petabyte-Scale datasets in modern Physics Experiments

Martin L. Purschke

BROOKHAVEN NATIONAL LABORATORY

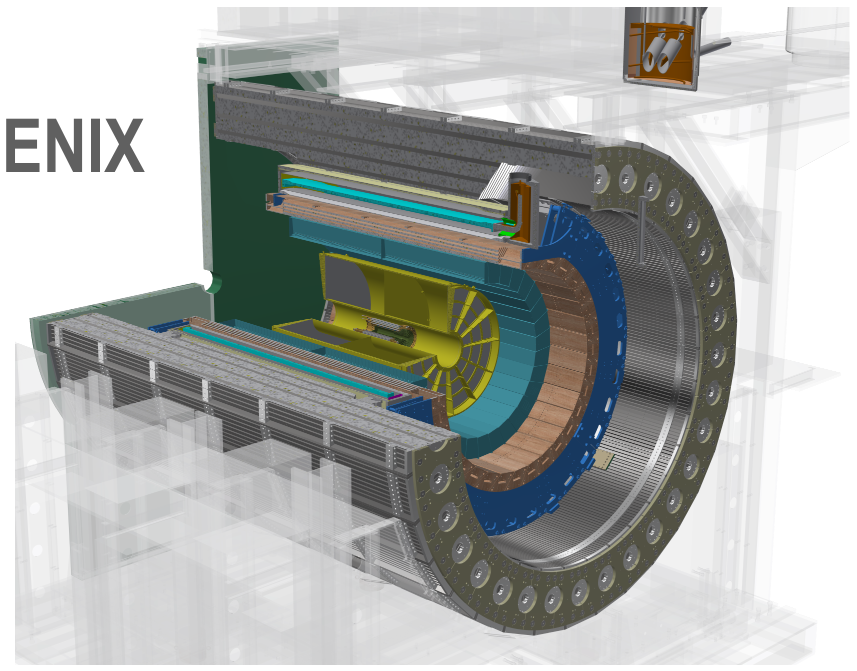
Brookhaven National Laboratory

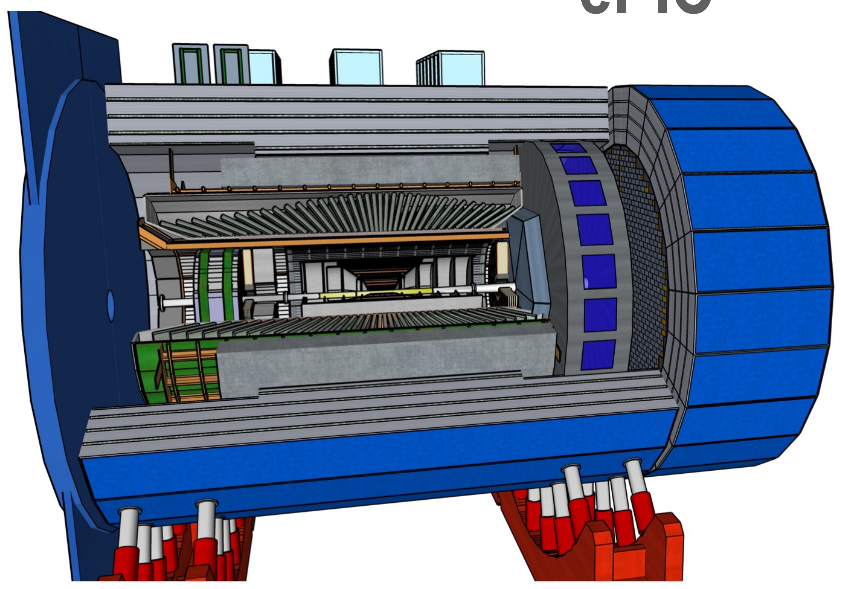
Home of the Relativistic Heavy Ion Collide and the future Electron-Ion Collider…

Long Island, NY

RHIC from space

… and of the new sPHENIX experiment – largest US Nuclear Physics experiment under construction (that’s what I work on)





What I do

I have been working with heavy ion beams since about 1985

First at CERN (WA80,WA93, WA98), spent a total of 11 years at CERN

Moved to Brookhaven Lab in 1996 for the PHENIX experiment that started taking data in 2001

Was the data acquisition coordinator and also part of the offline group for PHENIX

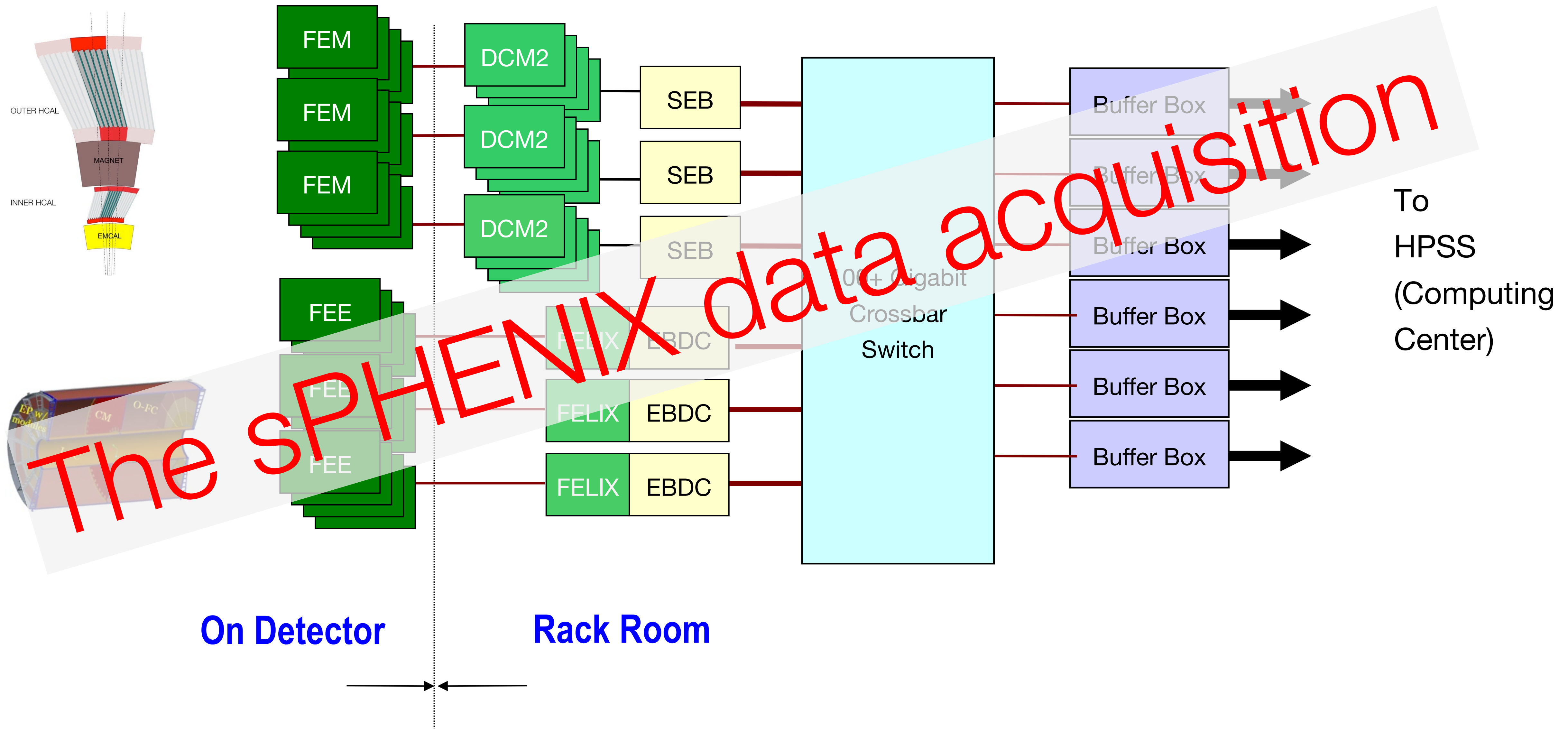
Last PHENIX data taken in 2016

Since then we have been building the successor experiment “sPHENIX”, again DAQ coordinator

Member of the (since disbanded) medical imaging group at BNL, Medical imaging (“RatCAP”)

Also the convener for the DAQ working group for the since accepted “ECCE” experiment proposal for the future Electron-Ion Collider at BNL (around 2031) – since renamed to “ePIC”

the sPHENIX DAQ in one slide ☺



What is “PetaScale”?

On the outside:

1KB = 1024 bytes

… MB… GB… TB

1TB = 1024 GB = 1024^4 Bytes = 1,099,511,627,776 Bytes

1 PB = 1024 TB…

Then comes exa, zetta, yottabyte…

BTW: if someone tries to tell you that a Gigabyte is 1,000,000,000 bytes, you are talking to a disk manufacturer sales guy who wants the number to sound bigger!

So: PetaScale -> we are dealing with at least a PetaByte of data.

Let me throw out some numbers…

CERN Data Centre passes the 200-petabyte milestone

The CERN Data Centre passed a major milestone on 29 June 2017 with more than 200 petabytes of data now archived on tape

what	Total	Per year	
My WA80 CERN Thesis Experiment	0.0007PB		
All of CERN's data before the LHC	~90PB		
PHENIX Experiment	25PB		
The LHC experiments collectively	200PB	30PB/year	
sPHENIX (next year)	300PB	120PB/year	1PB/day
Hi-Lumi LHC	?	250PB/year?	
NetFlix Storage	~10-20 PB		

What is “a lot of data”? Changes over time

In 1986/87 it took us about 8 months to just reconstruct (not analyze) the data from one year’s run

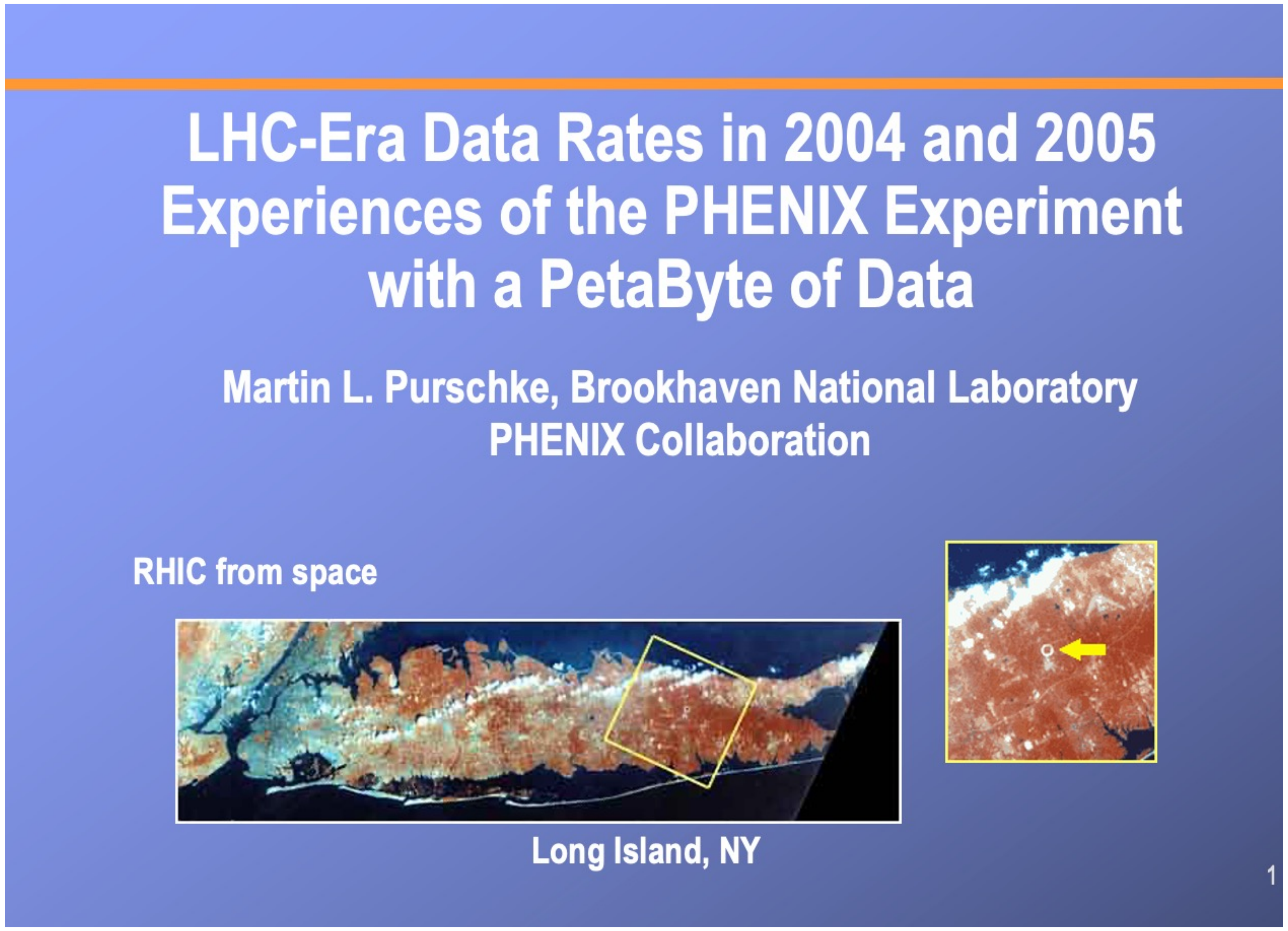
Today I could store the entire WA80 dataset on my home PC and analyze it in maybe a week or less

Never listen to people telling you that you don’t get the data analyzed in time. Take all the data you can get. Time is on your side.

An example

My opening slide of my presentation at the “Computing for High-Energy Physics” (CHEP) conference in Mumbai, India (2006)

That was before the LHC started taking data. A PetaByte of data from one experiment was unheard of then.



LHC-Era Data Rates in 2004 and 2005
Experiences of the PHENIX Experiment with a PetaByte of Data

Martin L. Purschke, Brookhaven National Laboratory
PHENIX Collaboration

RHIC from space
Long Island, NY
1

I got some good feedback from the LHC crowd –

“thank you, we think we can do it but didn’t have a proof-of-principle”

What I’ll go through with you today

The “local” stuff at your experiment, university cluster, small computing center

- How to set up large file systems
- RAID, ZFS, Lustre, and all that
- HPSS and other tape libraries

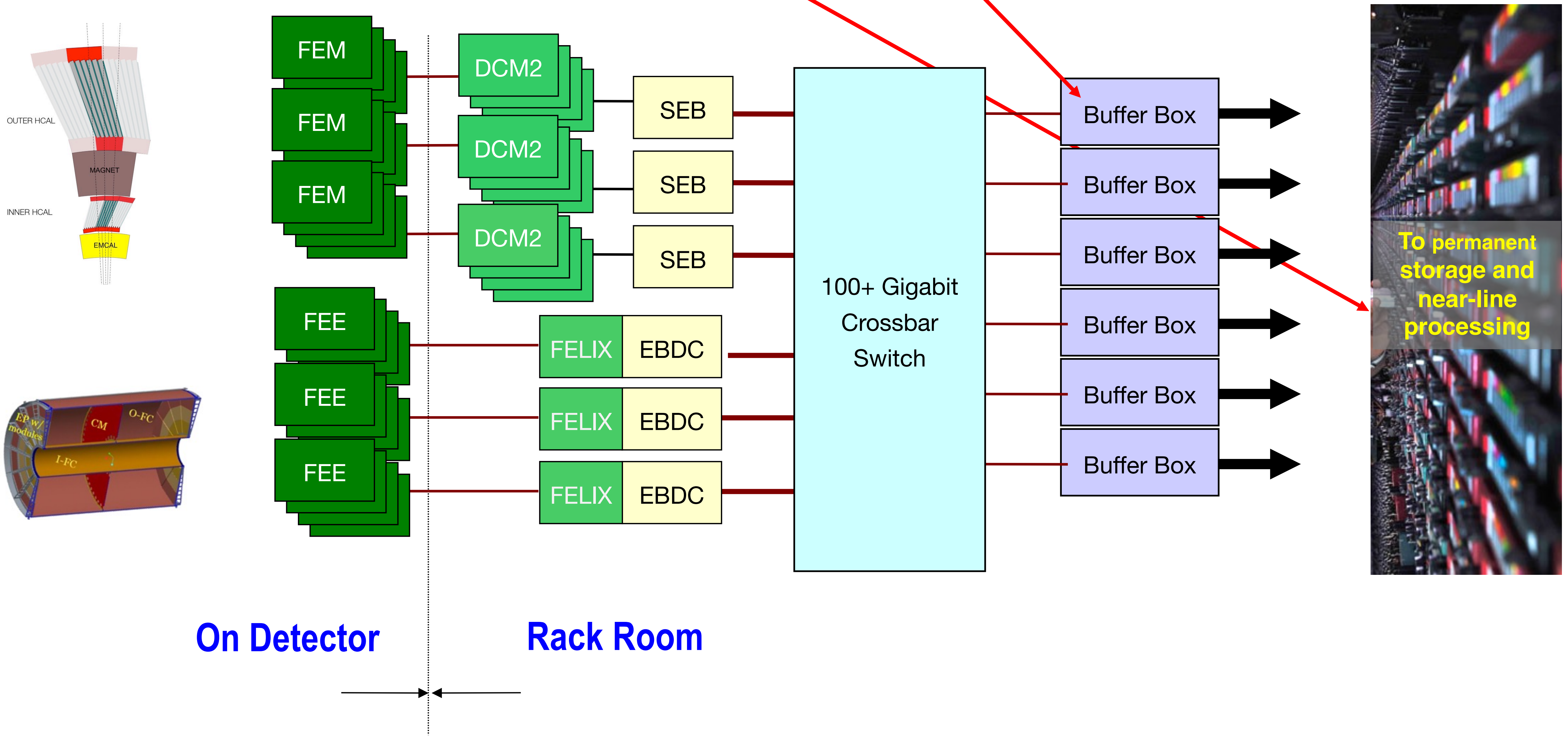
Using clusters/farms efficiently

- Condor
- Using the grid
- Really large-scale computing efforts

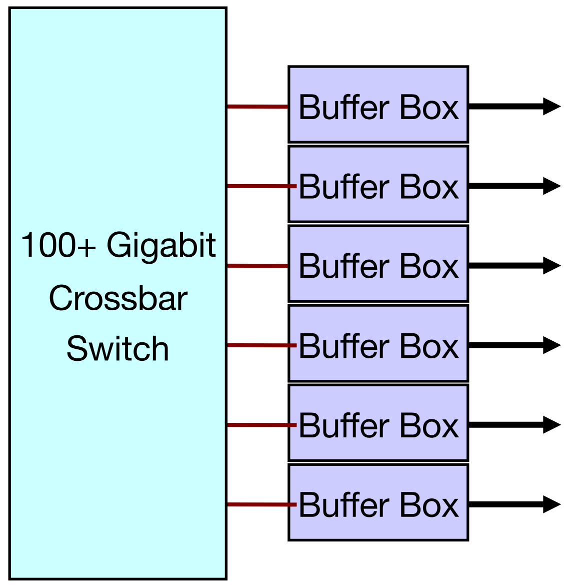
Gotta start somewhere. Here's my setup.

I don't really know any High-Energy / Nuclear Physics/ other experiment that doesn't have its own local storage system (short term, hours / days)

permanent storage "somewhere else"



Why do we call those “BufferBoxes”?

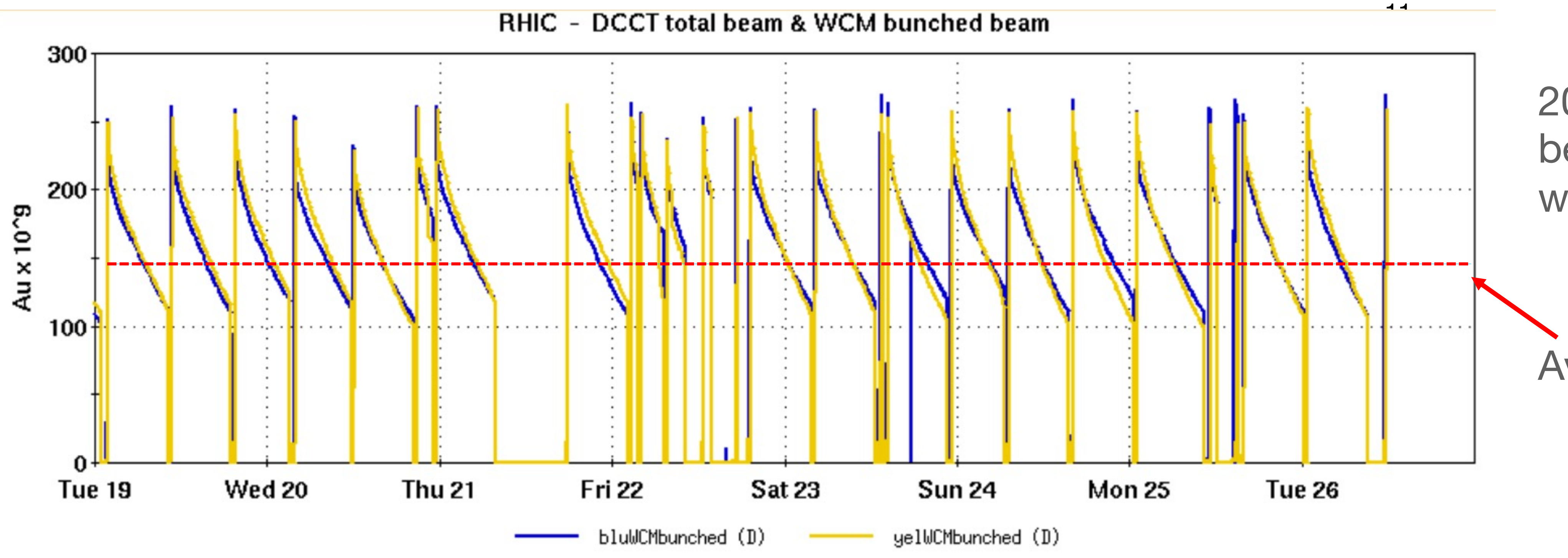


The data rate at a collider is “bursty” – high luminosity at the begin of a store, then ”burning off” – change of a factor of 2

Even if not, you create your own data rate variation by starting/stopping/ fixing/ changing parameters

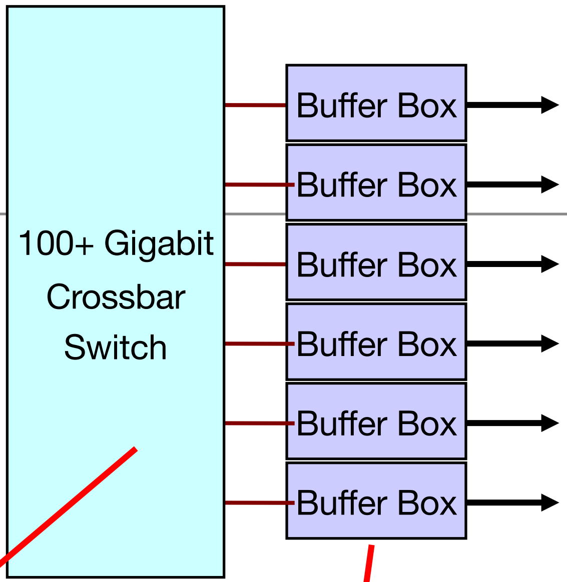
This Buffer boxes allow us to send the average, rather than the peak rate through the WAN

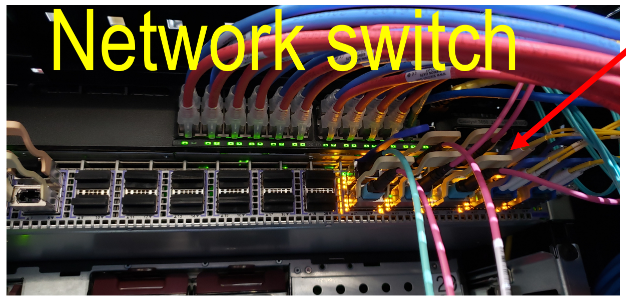
That is the largest piece of local storage that I have.

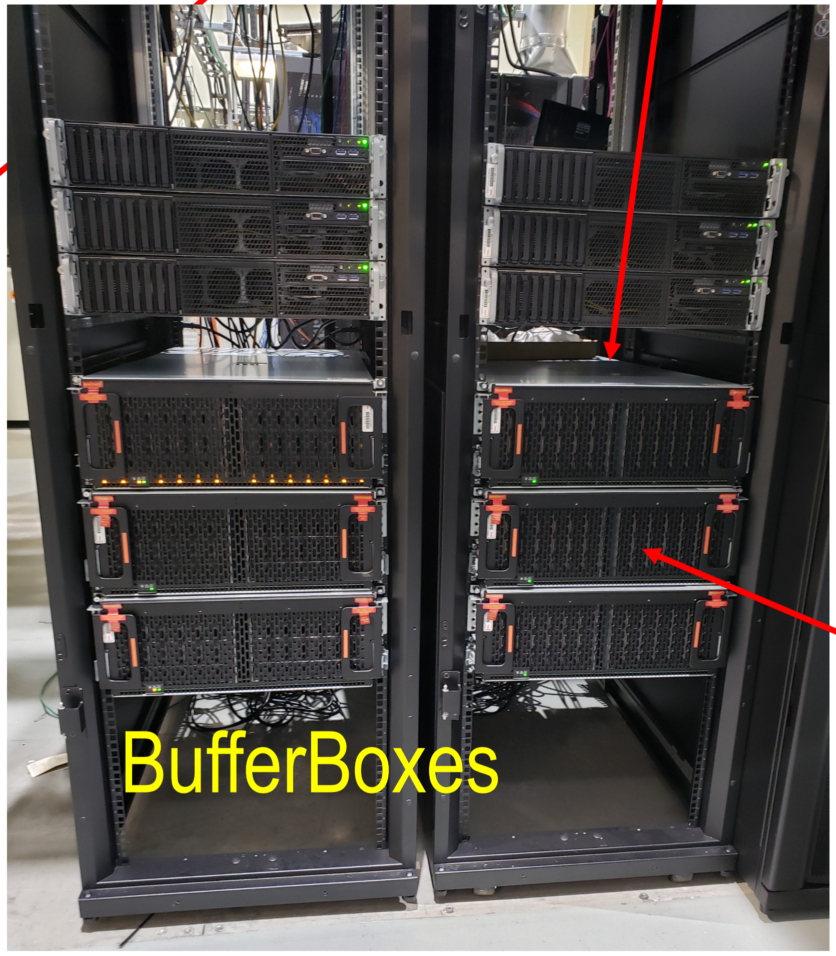


2016 (last PHENIX run) beam intensity over a week

Some Pictures







Let’s start small (let’s make like 50-100 TB storage)

Take a modern PC

Pretty much any modern PC these days has slots for NVME disks, and a bunch of SATA disks

NVME is what I use for the system disk, home disk (where needed) etc

Less than 10, 20 TB “on a budget” – use SSDs (2022 ballpark: 10cts/GB)

Larger than that – use spinning disks as large as you can afford them (~ 2cts/GB)

Say, 6 x 16TB = 106 TB raw disk capacity (~$300/disk, $1800)

Call this $3500 for a not-too-shabby but bare-bones machine like that, make 10, $35K gives you about a PetaByte

This is often referred to as “JBOD” – “just a bunch of disks”.

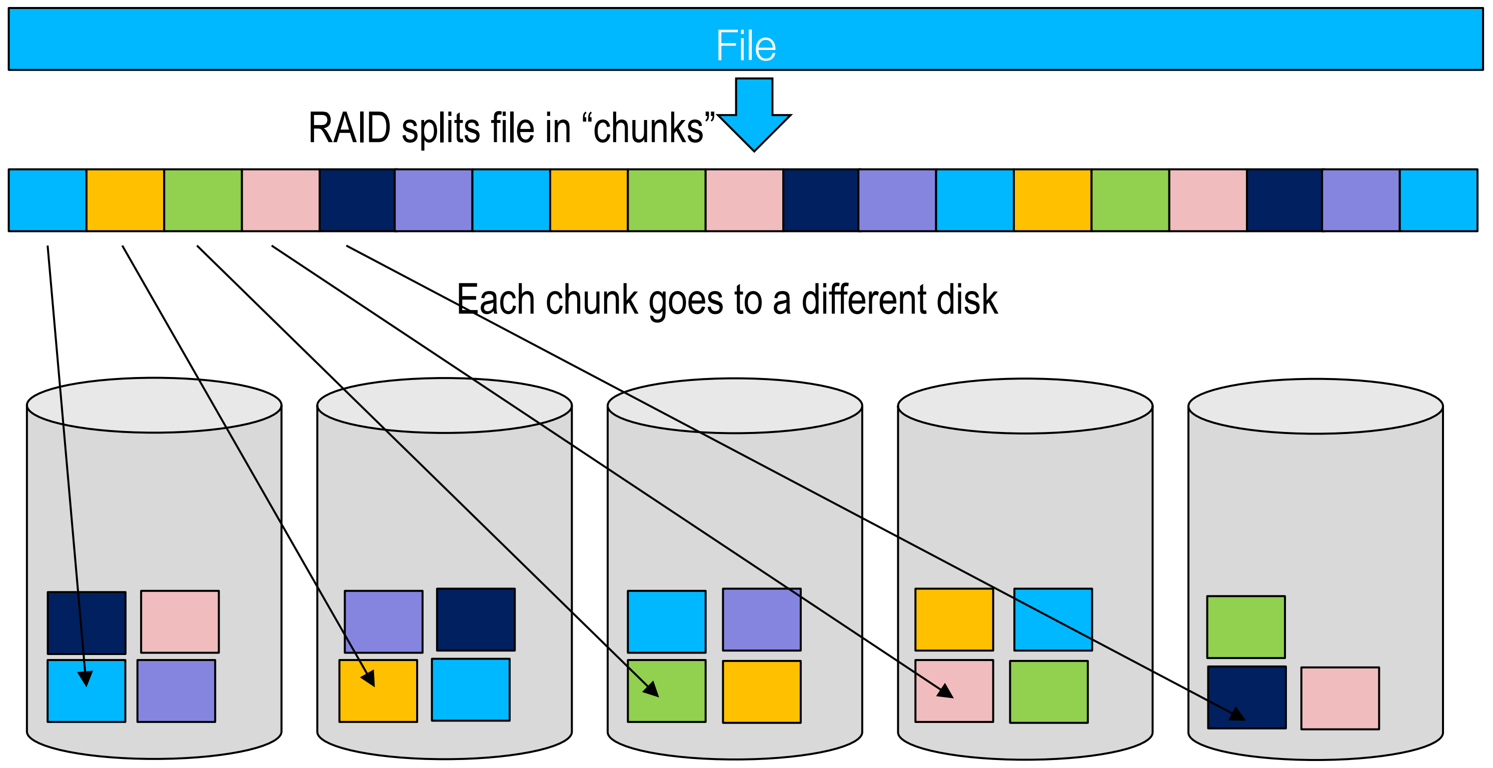
Now what?

RAID

“**R**edundant **A**rray of **I**ndependent **D**isks”

Cheapest way to get large storage sizes AND speed

Break up files written into chunks and send them to different disks, distribute I/O across many disks.


File
RAID splits file in “chunks”
Each chunk goes to a different disk

One disk ~ 150MB/s give or take (spinning) or 400MB/s (SSD) or 2000MB/s (NVME)

$ SSD / $ Spinning = ~5 so take 5x the disks, 150MB/s x 5 > 400MB/s (SSD)

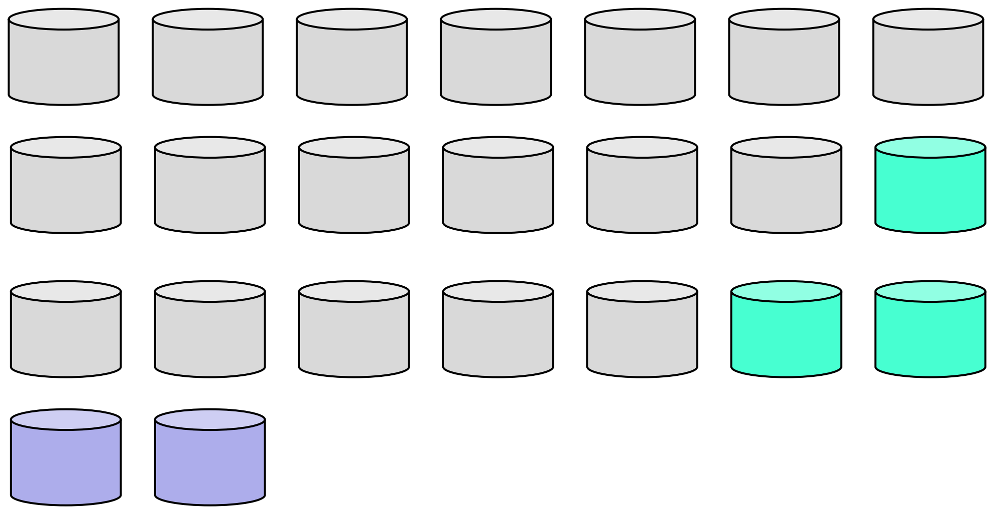
You win! But that’s not all.

Raid “levels”

This is not a raid tutorial, so I’ll be brief here

This is the “**R**edundant” part -

- Use additional disks for parity or other recovery mechanisms
- Survive the loss of one, two, or more disks without losing data
- Replace the failed disks, RAID rebuilds itself with a small performance hit, all new



RAID 0 (stripe)	No redundancy
RAID 5	Can lose 1 disk
RAID 6	Can lose 2 disks
Mirror	Both disks hold the exact same data

With RAID 5 you lose the capacity of one disk for parity, lose 2 disks with Raid 6

Raid Technologies

```
$ lspci
 . . .
 09:00.0 RAID bus controller: Adaptec AAC-RAID (rev 09)
```

Hardware raid – use actual hardware (Silicon) to manage the raid array

Many Vendors (3Ware, Adaptec, IBM, …)

Disadvantage: You can't bug-fix Silicon, and it's still proprietary

Today there is not much need for hardware RAID since by now the software has become faster

Software RAID gives more "peace of mind" – data not behind a black box

"md" RAID (mdadm)

Oldest technology, mature, the workhorse technology for software RAID

Set up in minutes

Lots of expertise on the internets

"RAID Write Hole" unsolvable problem

"zfs" RAID (zpool)

Ground-up redesign

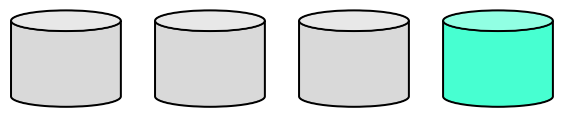
No "RAID Write Hole"

More flexible setups

Going mainstream quickly

If you start new, it's the way to go

Making RAID



The other day I set up a new modest-sized database server

4 1TB SSDs with a “raid 5” setup

Gave me an opportunity to show this here – I made a throwaway “md” array:

```
# mdadm -C -n 4  -l raid5 /dev/md0 /dev/sd{a,b,c,d}
```

(-C = create, -n 4 = 4 disks, raid5, name, list constituent disks). Easy!

Admired it for a bit, then took it down, set up zfs (what I had really wanted)

Making ZFS RAID

```
# zpool create db raidz  /dev/sd{a,b,c,d}
# df -h /db
Filesystem      Size  Used Avail Use% Mounted on
db              2.7T  1.7G  2.7T   1% /db


# mount | grep  /db
db on /db type zfs (rw,xattr,noacl)
```

That's my new database server!

Lustre (and Zeph)

Remember, RAID allows you to “stripe” your I/O across multiple disks

You gain throughput (and data protection).

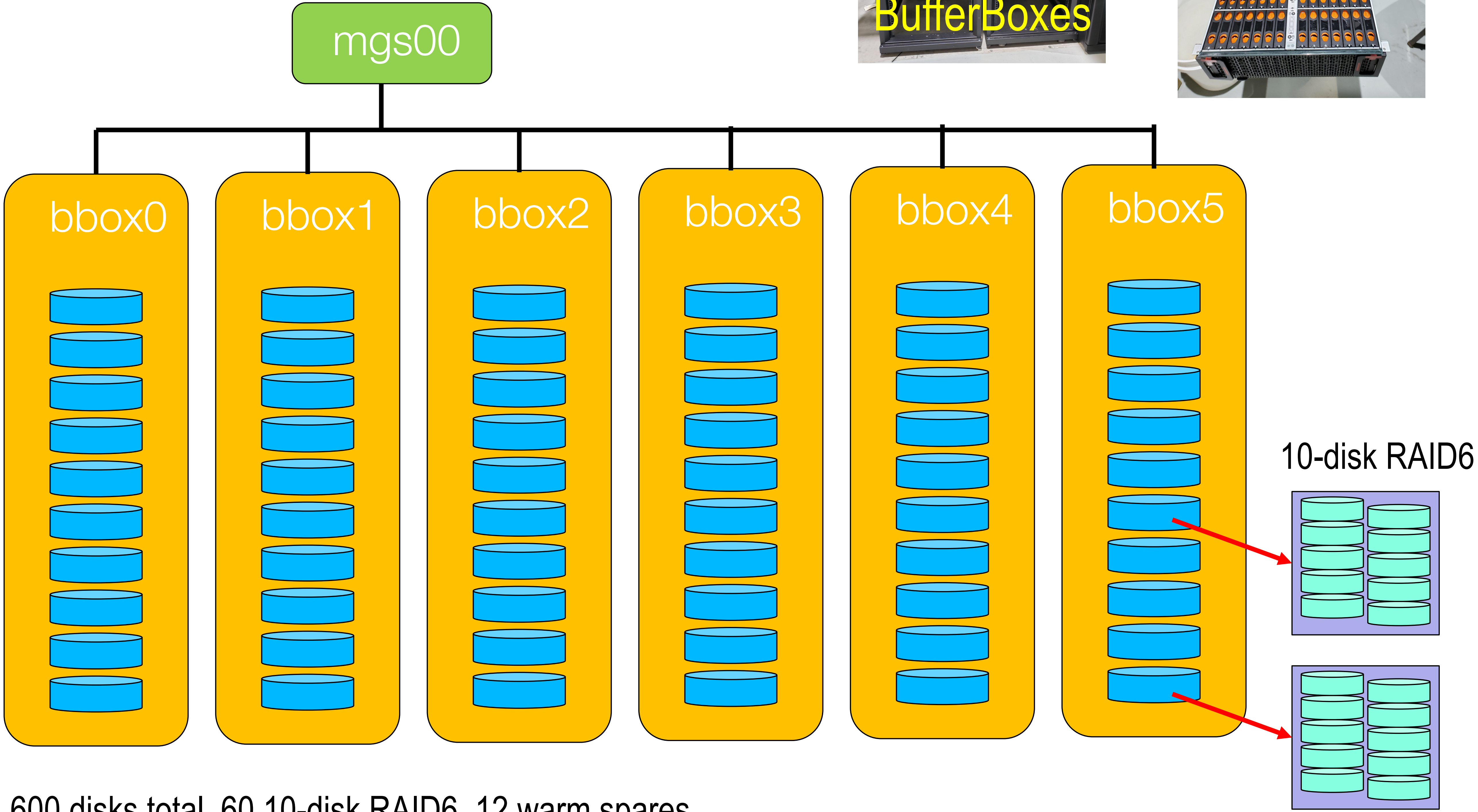
Lustre takes this same concept to multiple file servers

Same idea, distribute I/O across multiple servers, gain I/O capacity and (additionally) network capacity

Else this works pretty much like RAID, break up files into chunks, distribute across file servers.

Lustre is open-source, Zeph is fee-based from RedHat (pretty much the same concept)

Lustre



600 disks total, 60 10-disk RAID6, 12 warm spares

Permanent Storage – we stay with tape!

You might think that in 2022, a lot of an experiment’s dataset would be “held online”, that is, on disk somehow

The argument goes the other way – if you only take what you can keep on disk, you are not maxing out your experiment’s capacity

- No shortage of proposals and attempts to make super-large disk arrays but keep most of the disks powered down (“tape on disk”) – these proposals get routinely shot down
- Or use a “tape” library and have it move SSDs or traditional disks instead
- The motivation seems to, one way or another, eliminate tapes – tapes sound so 1990!

But that’s not true!

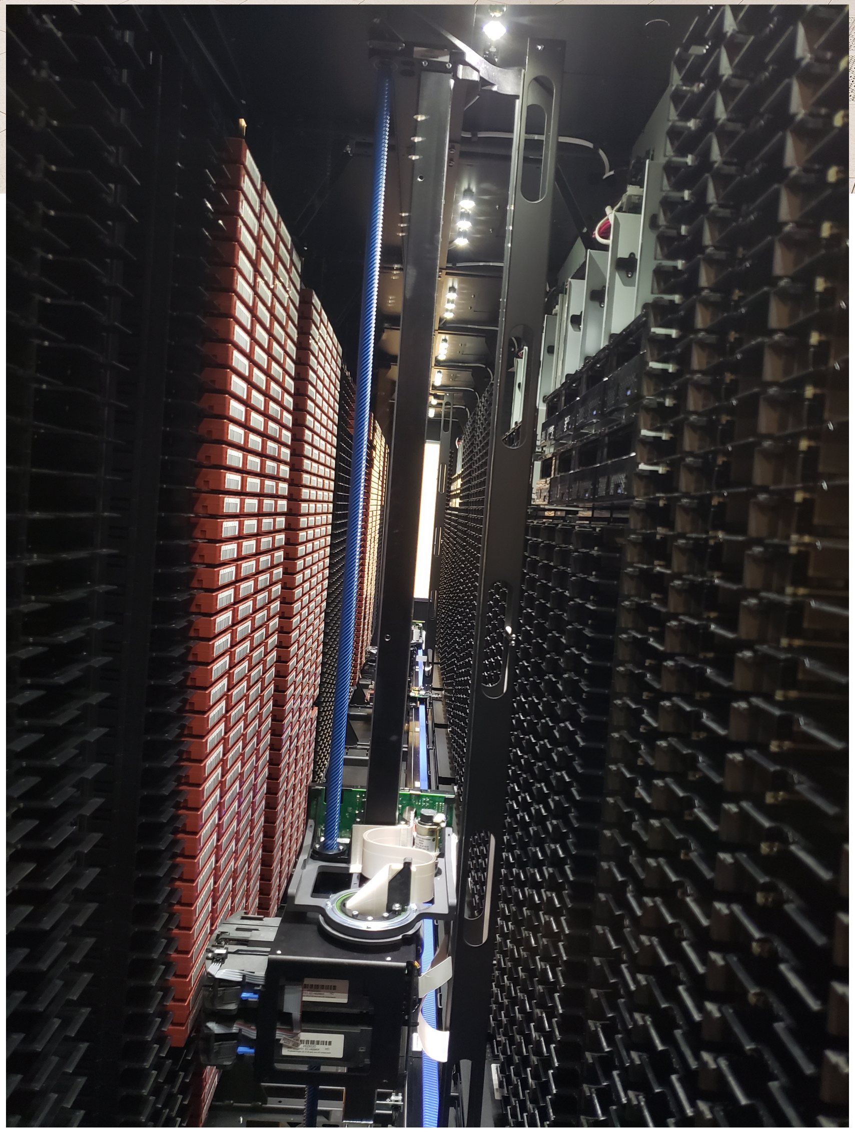
Nothing beats tape for large-scaler datasets. Not by a long shot

A tape cartridge is, after all, just a dense block of plastic

No SSD, no traditional disk can stand the g-forces that a tape library exerts on a tape

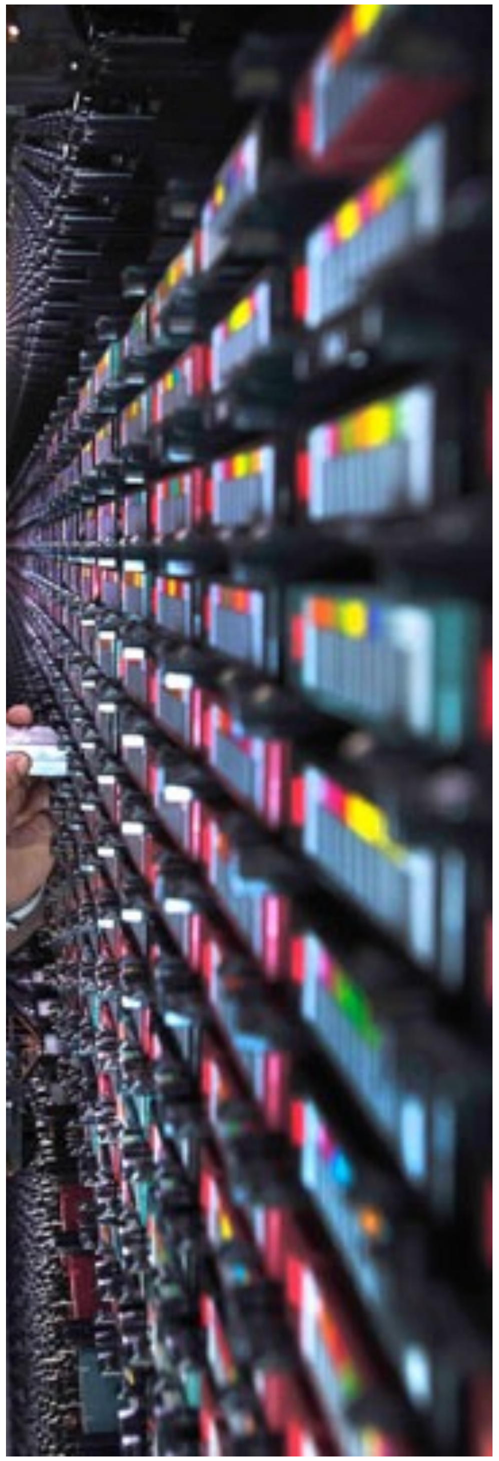
The cheapest and fastest way to store large data sets.

Permanent Tape Storage (HPSS)



Our tape libraries (being set up)

Our existing tape library



HPSS = High Performance Storage System

Changing gears…

Analyzing Peta-Scale data

Analysis of peta-sized data

The operational words here are “parallel” and “farm”

Get many – as much as possible identical – farm nodes that can all do the same

Break up your work into “small” chunks – a few wall-clock hours to 2 days or so

Very often: one raw data file worth

Then hand out such a quantum of work to each node

That’s how virtually any large-scale processing is done these days

Some assumptions here:

- Each quantum can be processed independent of the others
- Not always the case – sometimes the analysis of file *n* depends on the results of file *n-1*
- Then it’s getting a lot more complicated… too much for today

Condor

The goal is to maximize the load for each machine, each CPU core @100% all the time

100% not realistic, but 90% is achievable. **Idle computers are expensive!**

“Use it or lose it" policies of many computing centers

We need a job management system that keeps all farm nodes occupied

Usually you want at least x10 more jobs than slots, 90% waiting their turn

The most commonly used system (best known, free) is Condor

https://htcondor.com/

Other system are around, I’ll concentrate on Condor

Condor – the principle

Each participant worker PC says how much it can “take”, register with a central scheduler

For example, 64 CPU cores –> 64 “job slots” – often set to less to boost the memory per core

Each slot advertises what it “can do” – like how much memory, scratch disk, etc it can guarantee (“ClassAds”), and so on

Then you submit jobs that specify the requirements (for example, >1G memory, > 10G disk space, and on and on)

Then your job gets matched with a slot that satisfies the requirements, and then runs

You can analyze how much resources your job actually used, and tweak this – sometimes you find that there is not a single slot that can satisfy your initial requirements!

Then your job is executed according to its priority and place in queue (pretty much like boarding a plane)

ClassAdd and simple job example

```
HasTDP = true
TotalLoadAvg = 17.55
HasMPI = true
JavaMFlops = 2100.275391
MachineResources = "Cpus Memory Disk
Swap"
has_ssse3 = true
Disk = 7043048
OpSysMajorVer = 7
OpSysShortName = "RedHat"
HibernationSupportedStates = "S4"
SlotID = 1
OpSysLegacy = "LINUX"
SlotTypeID = 0
Rank = 0.0
MyType = "Machine"
JobUserPrioPreemptions = 0
HasVM = false
TotalSlotCpus = 1
LoadAvg = 1.0
Cpus = 1
TotalVirtualMemory = 464940056
TotalMemory = 191919
. . .
```

There can be hundreds of fields.

You usually only ask for very few, if any

transfer.job

```
Executable =  /home/sphnxpro/mlp_transfer/transfer.sh
Universe = vanilla
Initialdir =  /home/sphnxpro/mlp_transfer
Queue
```

```
$ condor_submit transfer.job
```

Again, keep it simple first

Here is a PC (one of my BufferBoxes) that stands up 6 slots just confined to itself

```
$ condor_status
Name                   OpSys      Arch    State     Activity LoadAv Mem    ActvtyTime

slot1@bbox0            LINUX      X86_64 Unclaimed Idle       1.000 31986 11+06:25:10
slot2@bbox0            LINUX      X86_64 Unclaimed Idle       1.000 31986 11+06:25:42
slot3@bbox0            LINUX      X86_64 Unclaimed Idle       1.000 31986 11+06:23:32
slot4@bbox0            LINUX      X86_64 Unclaimed Idle      10.210 31986 11+06:18:00
slot5@bbox0            LINUX      X86_64 Unclaimed Idle       1.000 31986 11+06:30:44
slot6@bbox0            LINUX      X86_64 Unclaimed Idle       1.000 31986 11+06:33:10
                         Machines Owner Claimed Unclaimed Matched Preempting

         X86_64/LINUX         6     0        0          6       0          0

                Total         6     0        0          6       0          0
```

Why? Well, it's a super-convenient way to regulate the number of processes that are executing concurrently - submit 100,000 but only 6 run, all others wait their turn

Here: I want to transfer hundreds of files to storage, but only 6 at a time

I use this a lot even on my laptop – 8 cores, make 6 slots, manage ”for all” jobs easily!

Giving parameters

You virtually always need to give parameters to your executable, and also specify stdin and stderr files

```
Executable     =  /home/sphnxpro/mlp_transfer/transfer.sh
Universe = vanilla
Initialdir     =  /home/sphnxpro/mlp_transfer
Arguments = /lustre_bbox/bbox0/junk/seb04_junk-00000102-0000.evt
output = out.log
error  = err.err
Queue
```

This does absolutely not scale! Don't do it this way

-a adds the input as if it had been given in the job file

Giving dynamic parameters with -a

The “-a” switch acts as if what follows had been added to the job file

```
Executable      =   /home/sphnxpro/mlp_transfer/transfer.sh
Universe = vanilla
Initialdir      =   /home/sphnxpro/mlp_transfer
Queue
```

```
$ condor_submit transfer.job \
  -a "Arguments = /lustre_bbox/bbox0/junk/seb04_junk-00000102-0000.evt” \
  -a "output = out.log” \
  -a "error  = err.err"
```

is completely equivalent to this

```
Executable      =   /home/sphnxpro/mlp_transfer/transfer.sh
Universe = vanilla
Initialdir      =   /home/sphnxpro/mlp_transfer
Arguments = /lustre_bbox/bbox0/junk/seb04_junk-00000102-0000.evt
output = out.log
error  = err.err
Queue
```

This allows you to….

… submit hundreds of files in one fell swoop

```
$ cat  submit_file.sh
#! /bin/bash

[ -z "$1" ] && exit

[ -f "$1" ] || exit

FILE=$(basename $1 .evt)
echo $FILE

condor_submit  transfer.job \
    -a "Arguments = $1" \
    -a "output = log/${FILE}.log" \
    -a "error  = log/${FILE}.err"
```

And go through the list of files and submit an individual job for each

```
$ for f in /data/files/*.evt ; do ./submit_file.sh $f ; done
```

Local farm clusters

This is an example from a few years ago when we were running medical imaging applications

In 2005 our medical-type folks would initially run the simulations to generate a “system matrix” for the image reconstruction on desktop PCs, super-inefficient.

”*It will take years to simulate the entire thing!*”

I offered to use my experiment’s online monitoring cluster (about 600 cores) at a time when the experiment wasn’t taking data for about two weeks or so

About 15,000 jobs

In a few days we got about 9 years worth of CPU time out of this

(easy math – with 600 cores each wall-clock hour gets you 600 CPU-hours)

And, after a day of setting up, it was “submit-and-forget”

BTW: A great example of synergy between people with different backgrounds!

Beware of "black holes"

Sometimes you have a mis-configured node where a job immediately fails

A missing library, missing installed support package, not enough memory, whatever

With 600 cores and 15000 jobs, 599 jobs start running

The 600th immediately fails on a black hole, but now that slot is free again

Next job starts there, fails again immediately, then the next, then the next

In this way you can get the bulk of your jobs to fail quickly

If you control the node, get it fixed. If not possible to fix, take it out of the cluster

Last resort, write a requirement in "not on that node"

Going global - the Grid

Idle machines are expensive

It has long been recognized that everyone wins when you pool resources more globally

Your cluster may be only 30% used this week, but you could need 150% the week after

So you open up your cluster to others, and others open up to you

Each side typically gives priority to their own jobs and take others when not fully used

Lots of flexibility to configure this, max jobs running time, mem usage, disk usage,…

Also, running in a “Sandbox” – a job is not allowed to see or access local storage or other resources

That’s the idea of the computing grid – pool resources and everyone wins

Grid access/authentication

Each group (say, a university cluster) sets rules who is allowed to run jobs there

Completely impossible to do this on a user-by-user basis

Use a hierarchical system of ”Virtual Organizations” (VO), such as the ATLAS VO

Then your university (if it’s part of ATLAS) adds you to their local VO, which in turn is part of the ATLAS VO (and perhaps others, or perhaps with intermediate VOs)

In this way, a huge VO such as ATLAS delegates the user authentication and each site trusts the others

You get a Grid Certificate (a PK12-level key) that is your ”VO membership” card

(No time to go into that, that’s an hour or two by itself)

Many Grids for many purposes

Sometime real organizations (such as CERN), sometimes regions (like northern Europe) make grids for specific purposes

Just a few logos I found in 60 seconds:

WLCG
Worldwide LHC Computing Grid

Open Science Grid

NORDUGRID
Grid Solution for Wide Area Computing and Data Handling

egi

Going from local (before) to the Grid (Medical Imaging)

Over time my initial 600 cores in my experiment's online cluster grew to about 2000

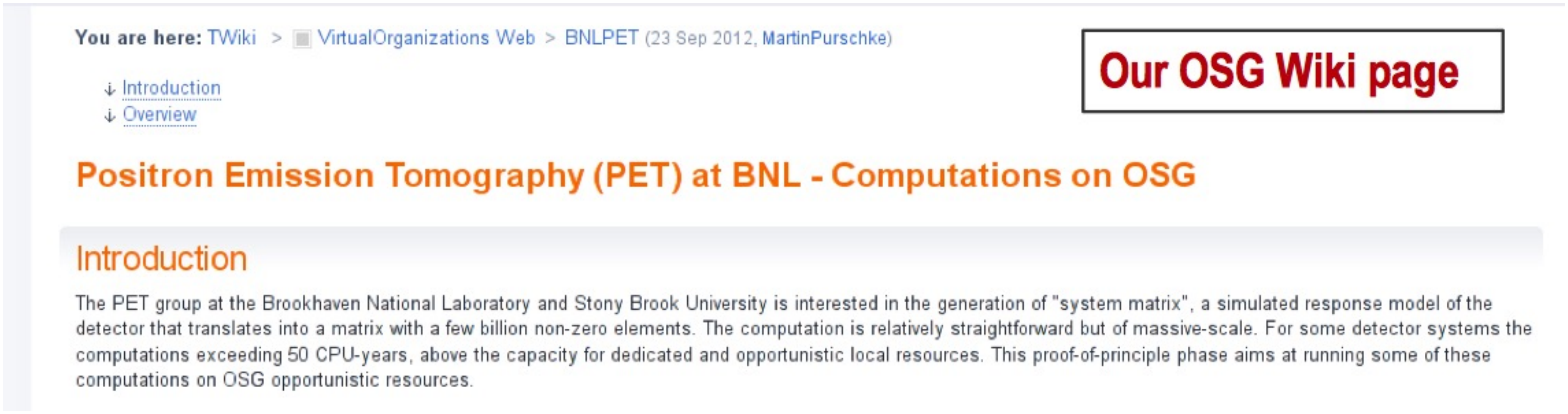
The real problem was the "seasonal availability" of the cluster to our friends/collaborators

The system is not available half of the year when we were running

So we decided to apply to join a VO on the Open Science Grid (OSG)

We could get about 1000 CPUs / day there when we needed it

Open Science Grid



You are here: TWiki > VirtualOrganizations Web > BNLPET (23 Sep 2012, MartinPurschke)

↓ Introduction
↓ Overview

Our OSG Wiki page

Positron Emission Tomography (PET) at BNL - Computations on OSG

Introduction

The PET group at the Brookhaven National Laboratory and Stony Brook University is interested in the generation of "system matrix", a simulated response model of the detector that translates into a matrix with a few billion non-zero elements. The computation is relatively straightforward but of massive-scale. For some detector systems the computations exceeding 50 CPU-years, above the capacity for dedicated and opportunistic local resources. This proof-of-principle phase aims at running some of these computations on OSG opportunistic resources.

And then a REALLY BIG project came along (for sPHENIX)

"Can we simulate 5 trillion events?"

- This is the odd-one-out large-scale simulations project
- Determining the various contributions to the muon production (Drell-Yan, Heavy Flavor, combinatorial) in the forward region
- Looking for (simulated) events which have one or more muons in the extreme forward region ($1 < \eta < 5$) with the Pythia event generator
- These are somewhat rare processes (~1.4*10-3)
- No way to influence the event generator to only produce desired event topologies – leads to severe biases
- brute-force crank through events and discard the unwanted topologies
- This gives this project an unusually low IO/CPU ratio – a few 10s of MB per job

And then a REALLY BIG project came along (for sPHENIX)

Work Breakdown

Typical “sweet spot” is to run jobs for ~10-12 hours

Optimum was found to be 10 million events/job (try and error)

Half a million jobs!

No way to do this manually!

Challenges

The earlier PET/Medical Imaging-related simulations only had a simple monolithic executable, a few scripts and data files, and 20,000 jobs or so

Here we are dragging an entire framework along

How to get the framework to the execution node?

How to get the data back "home"

How to automate everything that it remains a manageable endeavor that leaves time for my day job?

Target was a commitment of 10 days setup + 3 hours/week for 10 weeks of routine running

Also, "just me" – with no offense to anyone, automation beats more warm bodies

How to bring the project to and from the remote node

"to" is, in principle, the easier part

Looked at cvmfs, but… not set up for us fully at the time, not agile enough by far

No shortage of anonymous – not authenticated – ways to pull data from somewhere. wget, ftp, git,…

Getting data back to RCF is harder – no non-authenticated way to transfer, no safe method to furnish a job with the proper credentials (usually ssh keys)

Settled on condor file transfers for both directions

Data flowing back to the submitter host's /local-scratch

I/O levels well below "the radar"

In numbers…

A job needs about 1550MB in executables and shared libraries to run

All are 64bit binaries

Put a “support bundle” together with all required files, bzip2-compressed to 17% -> 253MB

Output is just a few 10’s of MB

Output flowing back to submitter host is getting rsync’ed back to home continuously, every 2hrs or so (ssh-agent is your friend)

Got into a routine to delete files from SH Wednesdays and Sundays – “safe delete procedure”

Got close to quota only once when I was alone on the OSG for a while [a good problem to have…]

How much CPU?

This is a snapshot close to the end of the project.

Didn’t keep all log files so I cannot run the final tally

```
$ find log/ -name ``*.log'' -exec grep 'Total Remote Usage' {} \; | \
   sed -e 's/,//g' | awk '{print $3}' | \
   awk -F: '{X += ($1 *3600 + $2*60 + $3)/3600} END {print X}'

1.34523e06

$ bc -lq

1345230 / 24
56051.25000000000000000000
1345230 / 24/365
153.56506849315068493150
```

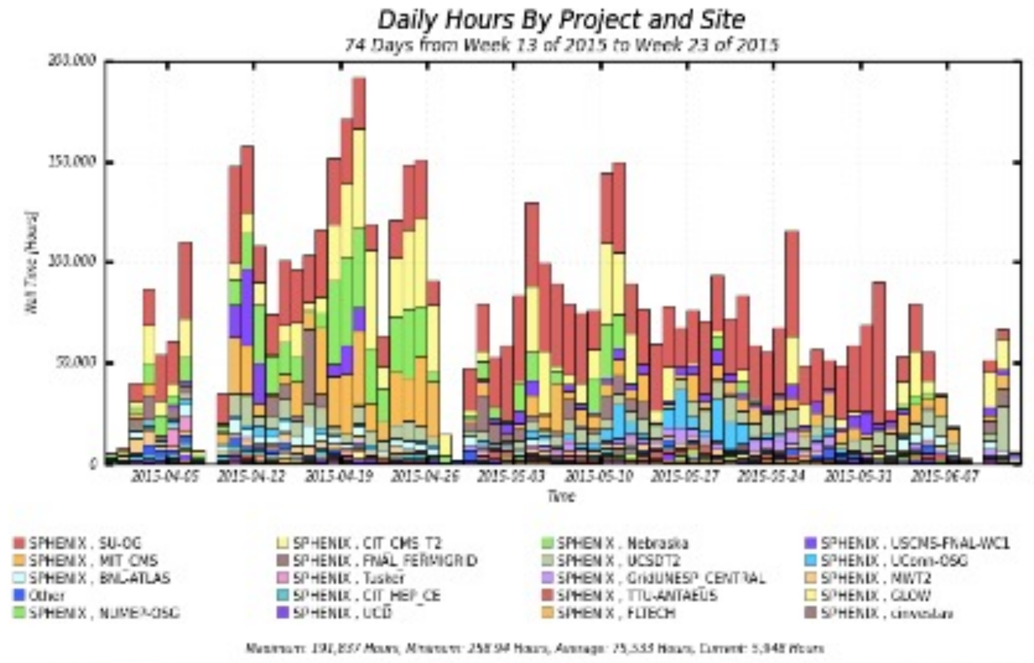
1345230 hours
56051 days
153 years

The OSG was very proud of this accomplishment

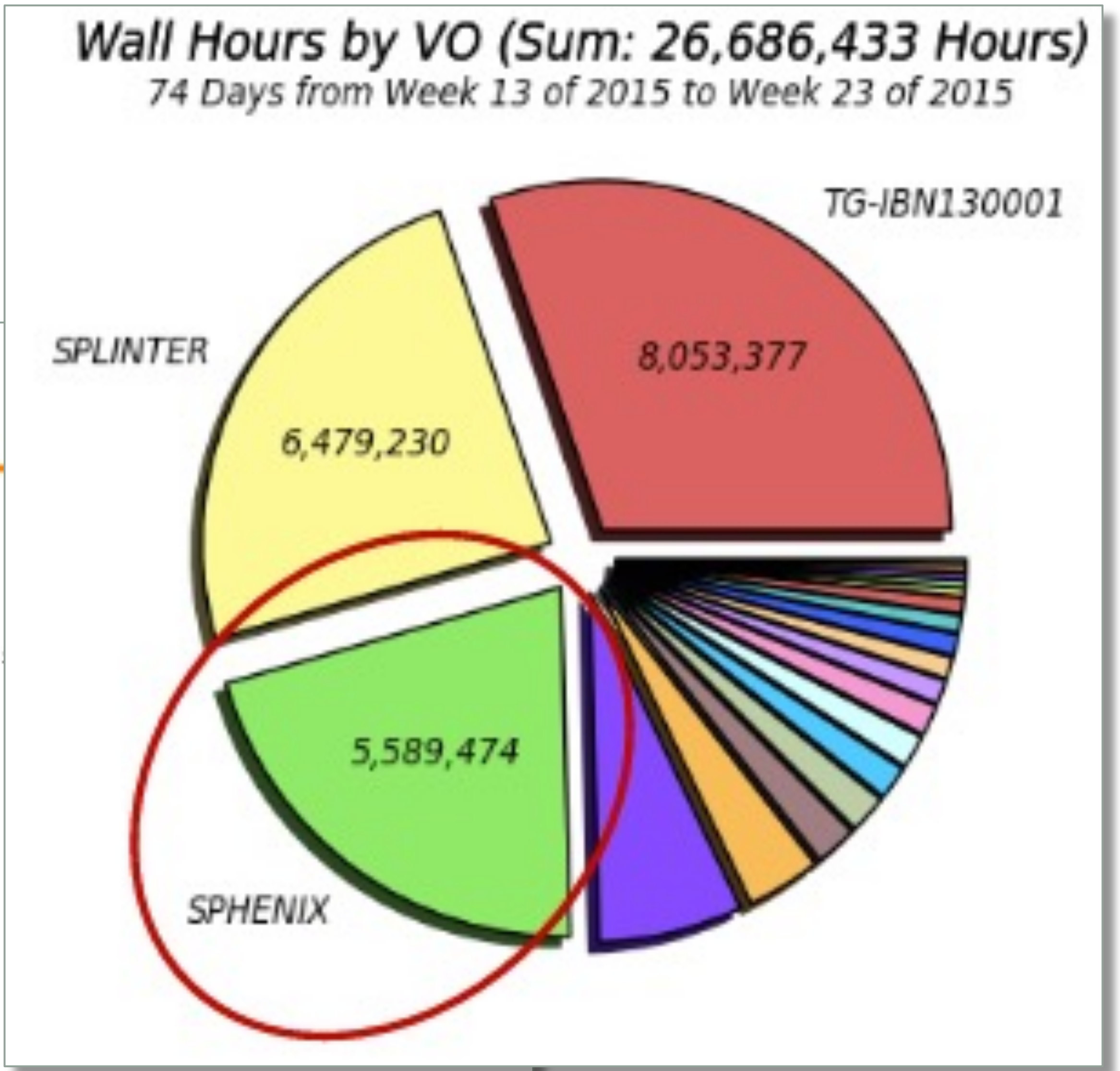
Of course our OSG marathon got noticed…

Open Science Grid

sPHENIX



- Studies for upgraded PHENIX detector at BNL (~5 trillion collisions)
- ~10 week computing campaign (M. Purschke)
  - 5.6M hours
  - Able to ramp up rapidly - #3 project on OSG in that time



Summary

Peta-Scale setups are not so much of an issue, can be done for a large university group

File systems – if it’s all the same, look into Lustre

Local setups - use condor to set up cluster

Large setups – talk to your experiment/organization to join a VO to access the Grid

All the mechanics can easily be mastered in a week, then you have enough experience

I haven’t talked about data center tiers, closest-copy access strategies, etc etc

There would be a lot more ground to cover in a dedicated workshop

Thank you!

Backup

Condor job (specific example as illustration)

```
InitialDir = /local-scratch/purschke/pythia/output

Executable     =  run.sh
Universe = vanilla

when_to_transfer_output = ON_EXIT
notification  = Never

should_transfer_files = YES

transfer_input_files = ../files.tar.bz2

output = x.out
error = x.err
Log = x.log

requirements = (( OpSysAndVer == "rhel6" ) ||  ( OpSysAndVer == "SL6" ) ||  (
OpSysAndVer == "CentOS6" )  )

+ProjectName = "sPHENIX"
queue
```

run.sh

```
#! /bin/sh
export JOBNR=$1
[ -z "$JOBNR" ] && JOBNR=0
NAMEEXT=$(printf "%08d\n" $JOBNR)

mkdir run_area
cd run_area
time tar xfj ../files.tar.bz2
rm -f ../files.tar.bz2

export LHAPATH=$_CONDOR_SCRATCH_DIR/run_area/PDFsets
export
LD_LIBRARY_PATH=$_CONDOR_SCRATCH_DIR/run_area/install/lib:$_CONDOR_SCRATCH_DIR/run_area/sys
libs:$_CONDOR_SCRATCH_DIR/run_area/root/lib
export ROOTSYS=$_CONDOR_SCRATCH_DIR/run_area/root
PATH=$ROOTSYS/bin:$PATH

ldd $_CONDOR_SCRATCH_DIR/run_area/syslibs/libfun4all.so > ldd.txt 2>&1
if  grep -q "not found" ldd.txt; then
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$_CONDOR_SCRATCH_DIR/run_area/oslibs
fi

root -b -q phpythia.C\(10000000,\"pythia_MB.cfg\",1\)
if [ -e phpythia.root ] ; then
    mv phpythia.root ../phpythia_single_${NAMEEXT}.root
    mv phpy_xsec.root ../phpy_xsec_single_${NAMEEXT}.root
fi
cd ..
rm -rf run_area
```