

Phenix Focus: The DAQ

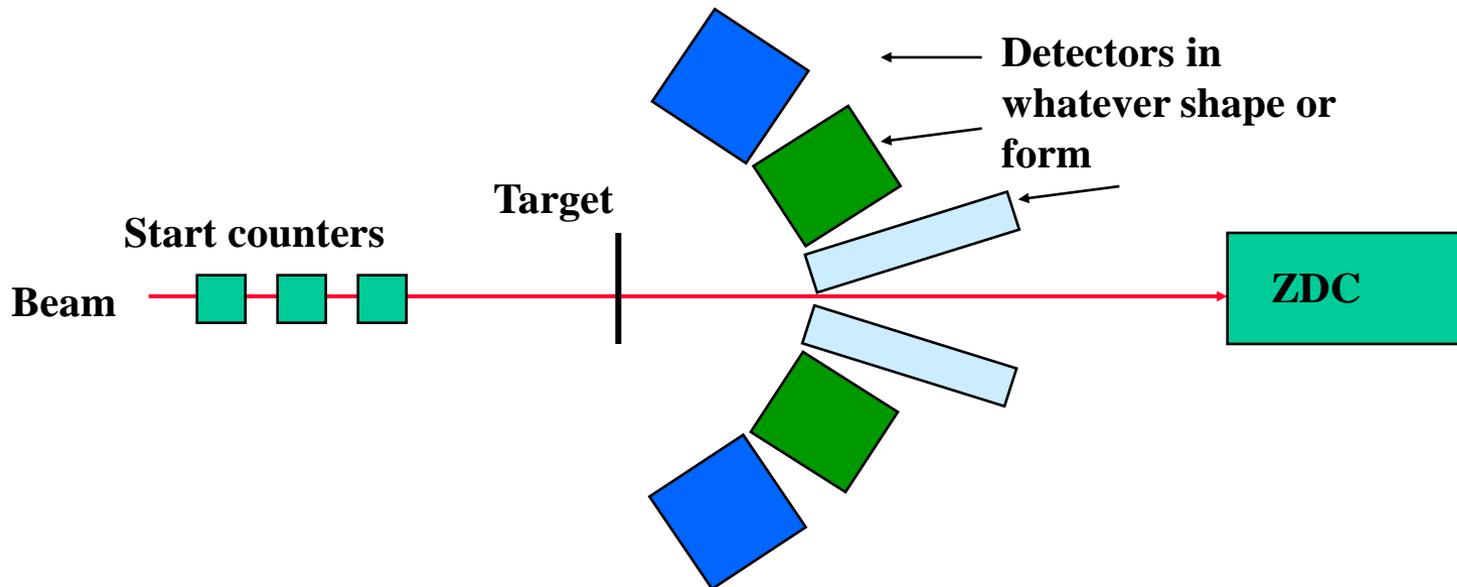
Martin L. Purschke for the DAQ and online crowd

Outline:

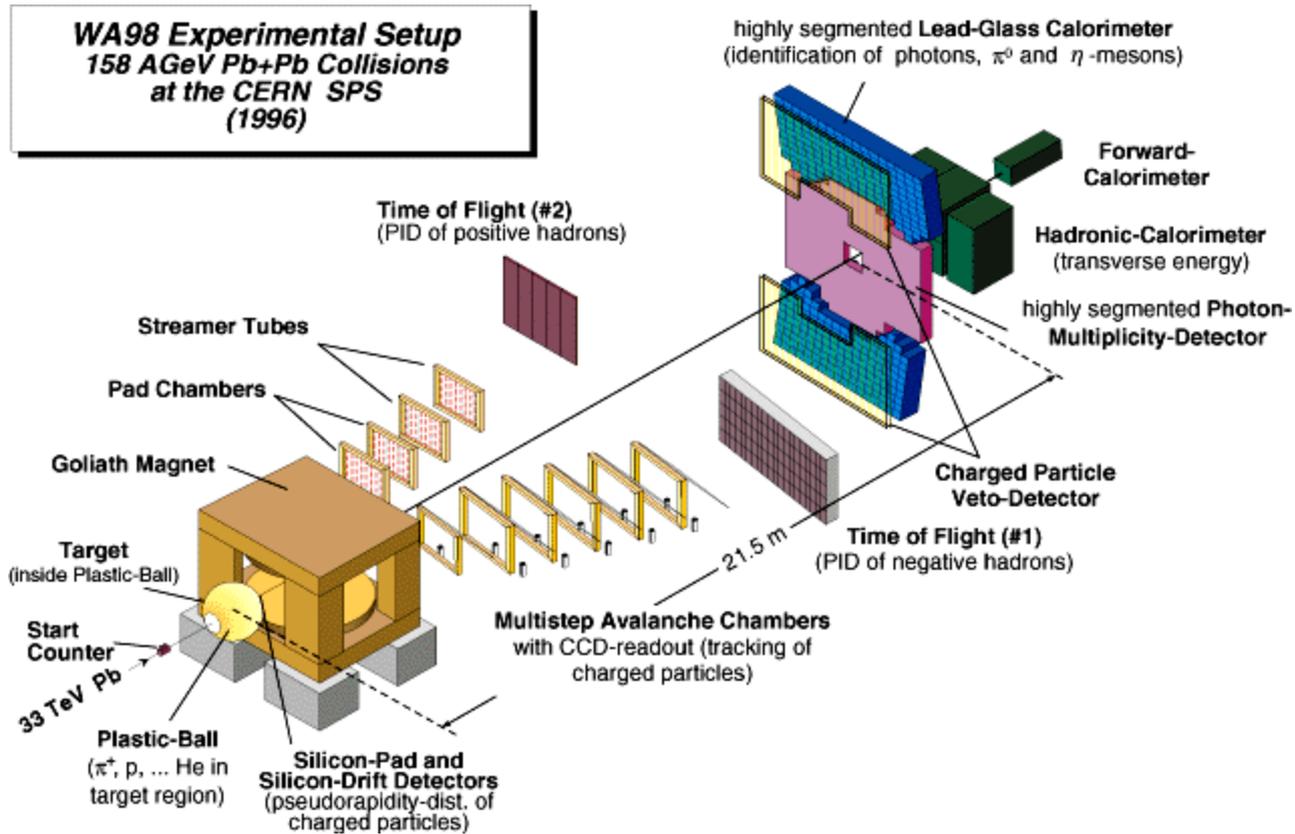
- before we dive in: some facts to better understand the rest
- FEM's and DCM's and other TLA's
- some time on the Timing System etc
- Run control
- EVB, very briefly
- data logging, compression, etc

Before we dive in: some facts

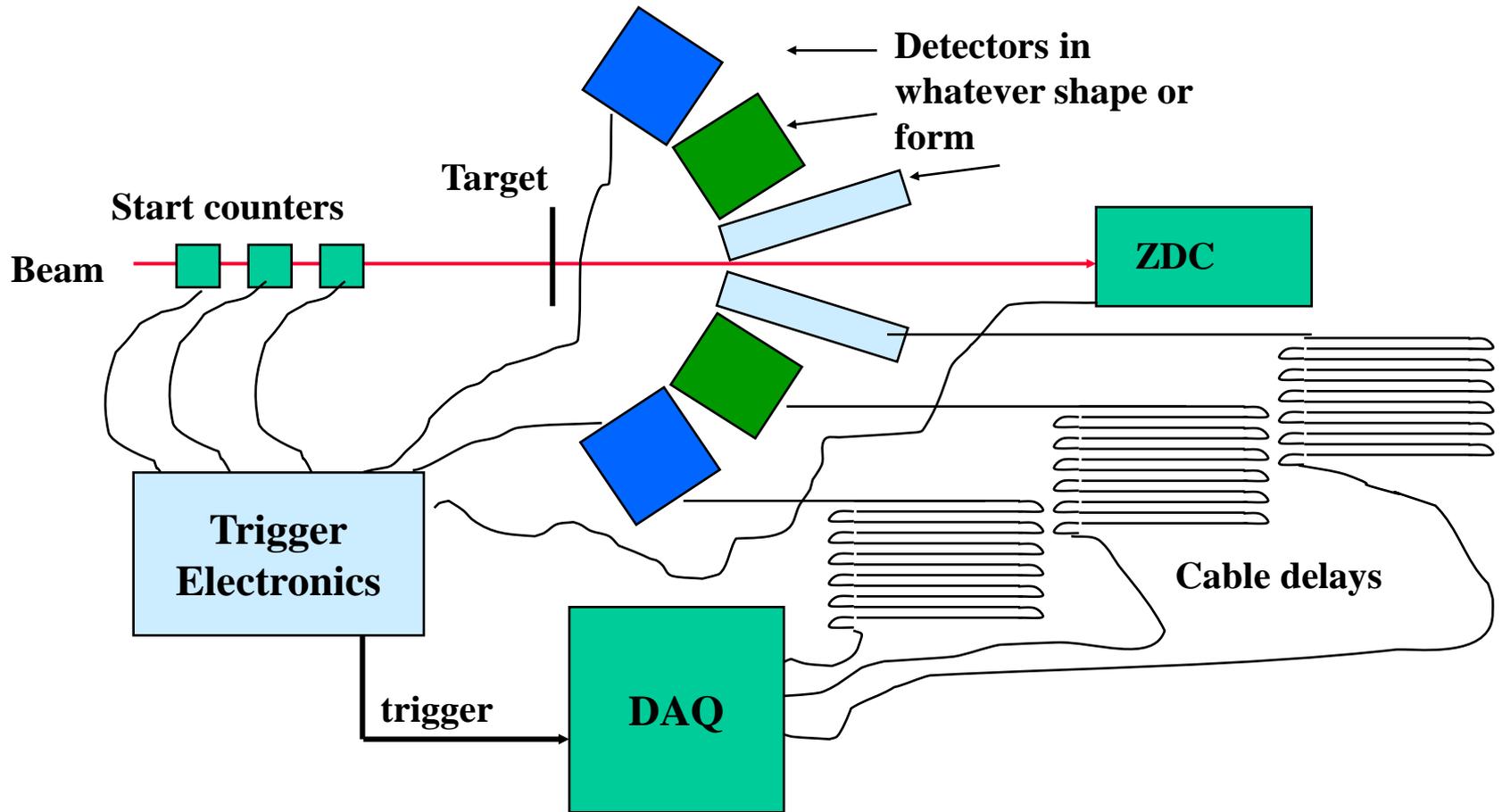
Historically, experiments (AGS, CERN, etc), no matter how different, have been built according to a similar blueprint:



Example: WA98 @ CERN

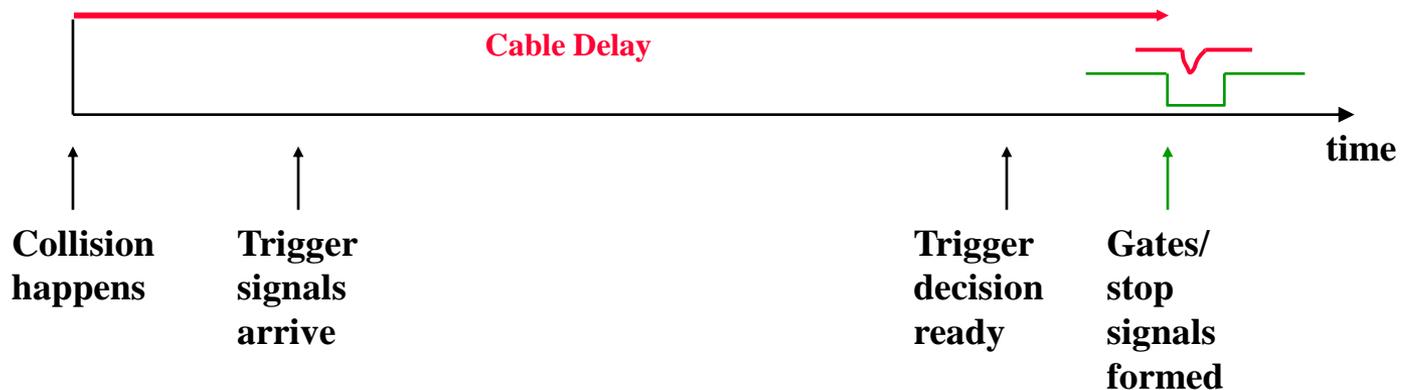


...some facts



What are those delay cables for?

You need some time for your trigger electronics to take a trigger decision.
You bring the signals you need to the electronics as fast as you can
You send the rest of the signals on a detour through a loooooong cable
By the time they arrive at your ADC/TDC/what have you, you have a decision.



Example WA80: 105m delay cables → 525ns delay
~150ns trigger signal transit time
~350ns trigger decision time
and a lot of \$\$\$ for the cables.

That's an awful lot of cables. We even bought special cables for the trigger signals (@4ns/m) to scrape a few ns of the transit time. Still marginal.

That wouldn't scale to RHIC realities...

BNL 52185
UC-414 (High Energy Physics)
DOE/OSTI-4500 — Interim 3

**Proceedings of the Third Workshop
on Experiments and Detectors
for a Relativistic Heavy
Ion Collider (RHIC)**

JULY 18-22, 1988

Edited by B. Shivakumar and P. Vincent



BROOKHAVEN NATIONAL LABORATORY
ASSOCIATED UNIVERSITIES, INC.
UPTON, LONG ISLAND, NEW YORK 11973

UNDER CONTRACT NO. DE-AC02-76CH00016 WITH THE
UNITED STATES DEPARTMENT OF ENERGY

In 1988, there was a RHIC workshop at BNL.

As far as I recall, it was there when everyone finally acknowledged that cable delays are not an option.

Summary of the Working Group on Readout Electronics

W. E. Cleland (convener), J. Hall, R. Ledoux, E. Platner,
S. Rescia, J. Stachel, H. Takai, B. Wadsworth, and G. Young

Other working group participants:

S. Dhawan, O. Dietzsch, S. Eiseman, M. Newcomer, I. Pless, M. Purschke,
V. Radeka, R. Scharenberg, S. Steadman, R. Van Berg, and Bo Yu

Instead of cable delays...

We had to find something that's to be had for a few \$/channel - cables break the bank, right there.

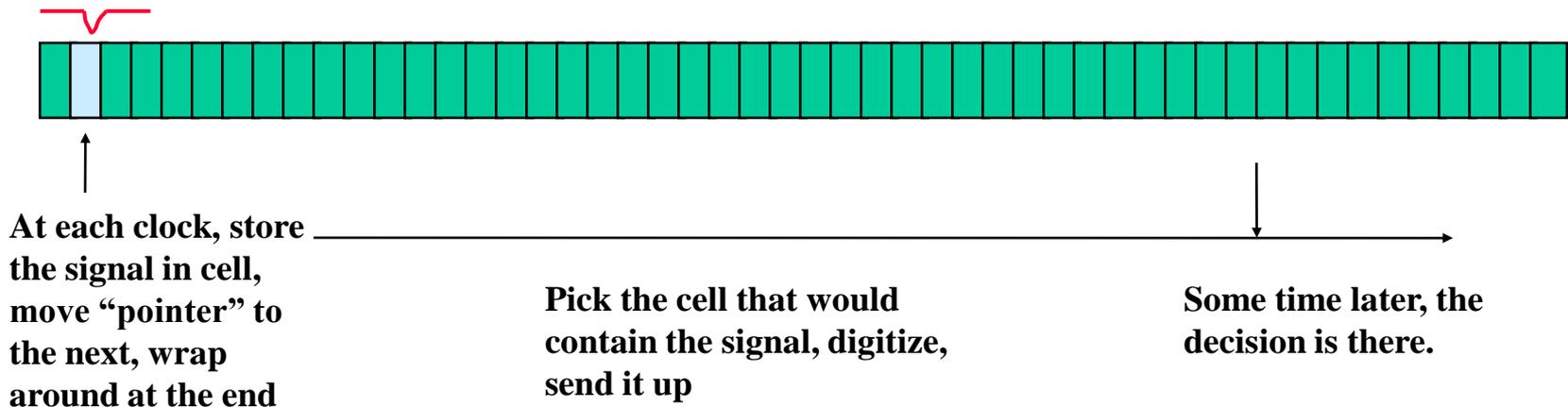
The experts looked at analog memories to store signals for some time (a few us) to have time for the Level-1 trigger decision, then get it or discard it

Mount the storage right on the detector, short cables (Emcal) or even a part of the detector (PC) --> Front-End Module (FEM)

This is possible @RHIC because we know exactly when a collision can happen (only when there is a bunch crossing, determined by the accelerator clock)

So, store the signal in an Analog Memory Unit (AMU), gain time to take a Level-1 decision, then digitize the contents of the memory cell that got filled some time ago

No more cable delays, a few \$ / channel, voila.



That concept has been implemented in PHENIX

Everyone has heard the word "AMU" cell, especially in the EMCAL, TOF, etc.

It's mainly done to do away with the cable delays. But once you have it, the concept offers another intriguing possibility: Multi-event buffering.

Why just store one event in the AMU? Store a few, beat down the dead time a bit. Easy?

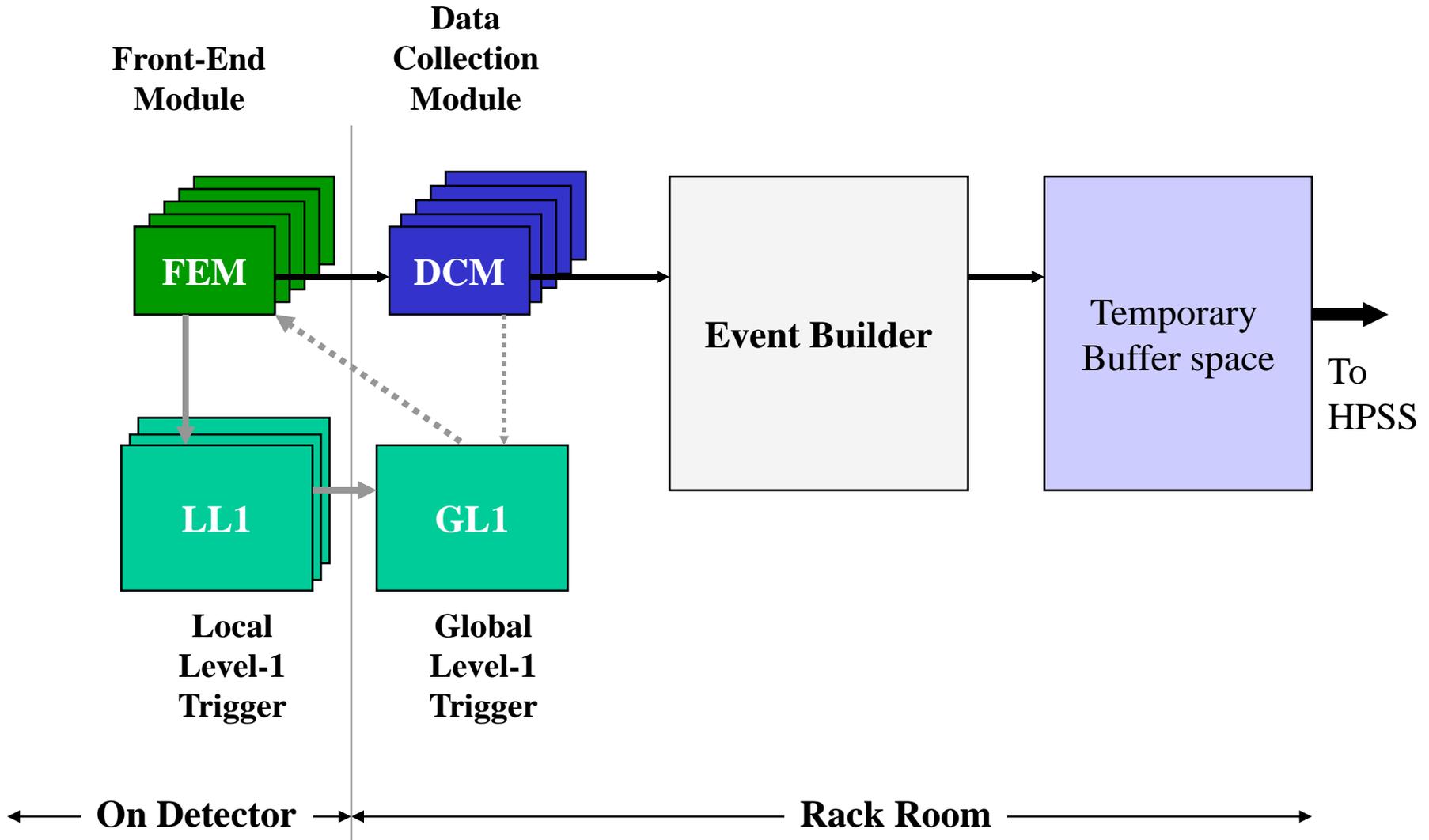
Well, it's no secret that we aren't quite there yet. Now you overlap digital readout on the previous sample with digitizing the next (might introduce noise)

The firmware that controls this hasn't been debugged in this mode

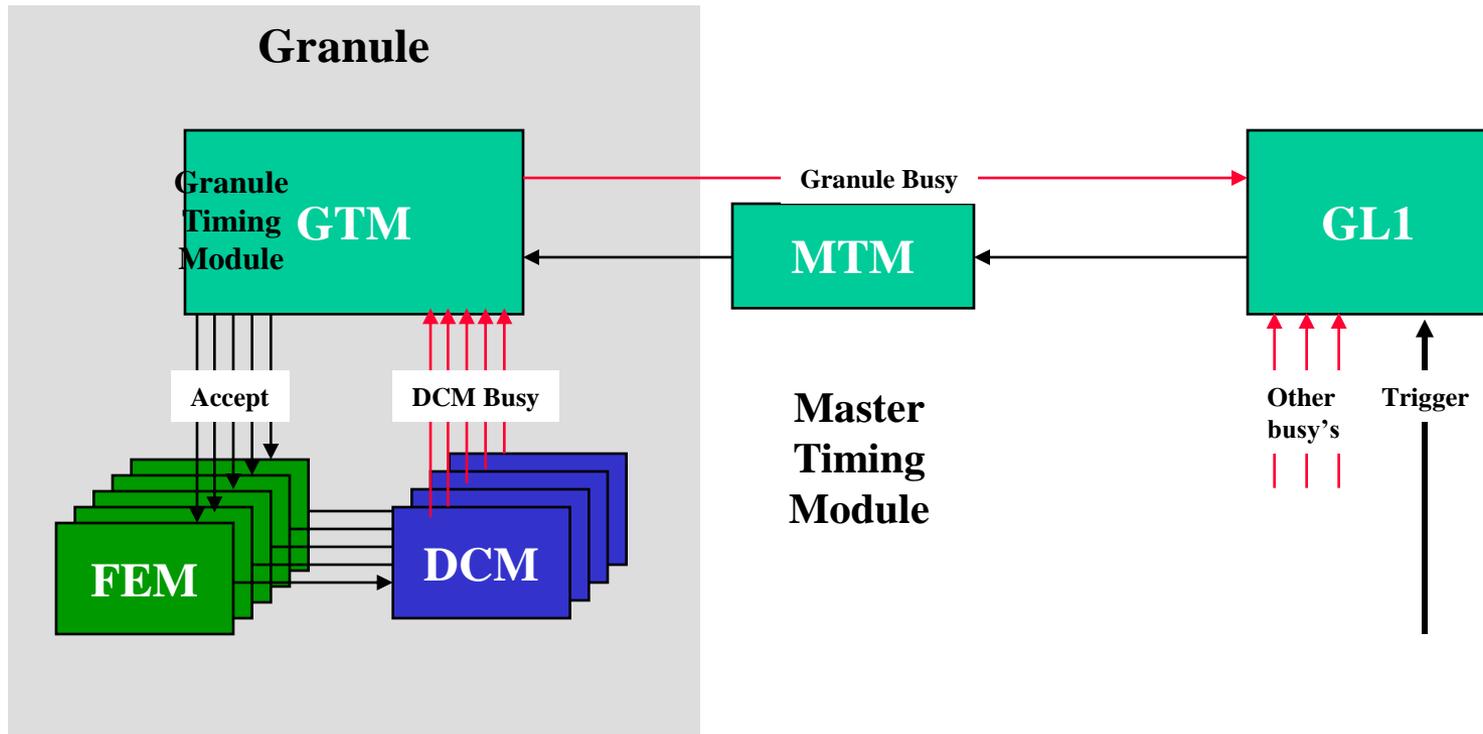
We still have data corruption in the Mutr, Emcal with MEB

Experts are working. Not quite yet.

DAQ components



More details



A Granule is the smallest unit of hardware that can be read out.

1 GTM (Granule Timing Module)

1 or more FEM's

1 or more DCM's

Some pictures...



GTM - Granule timing module

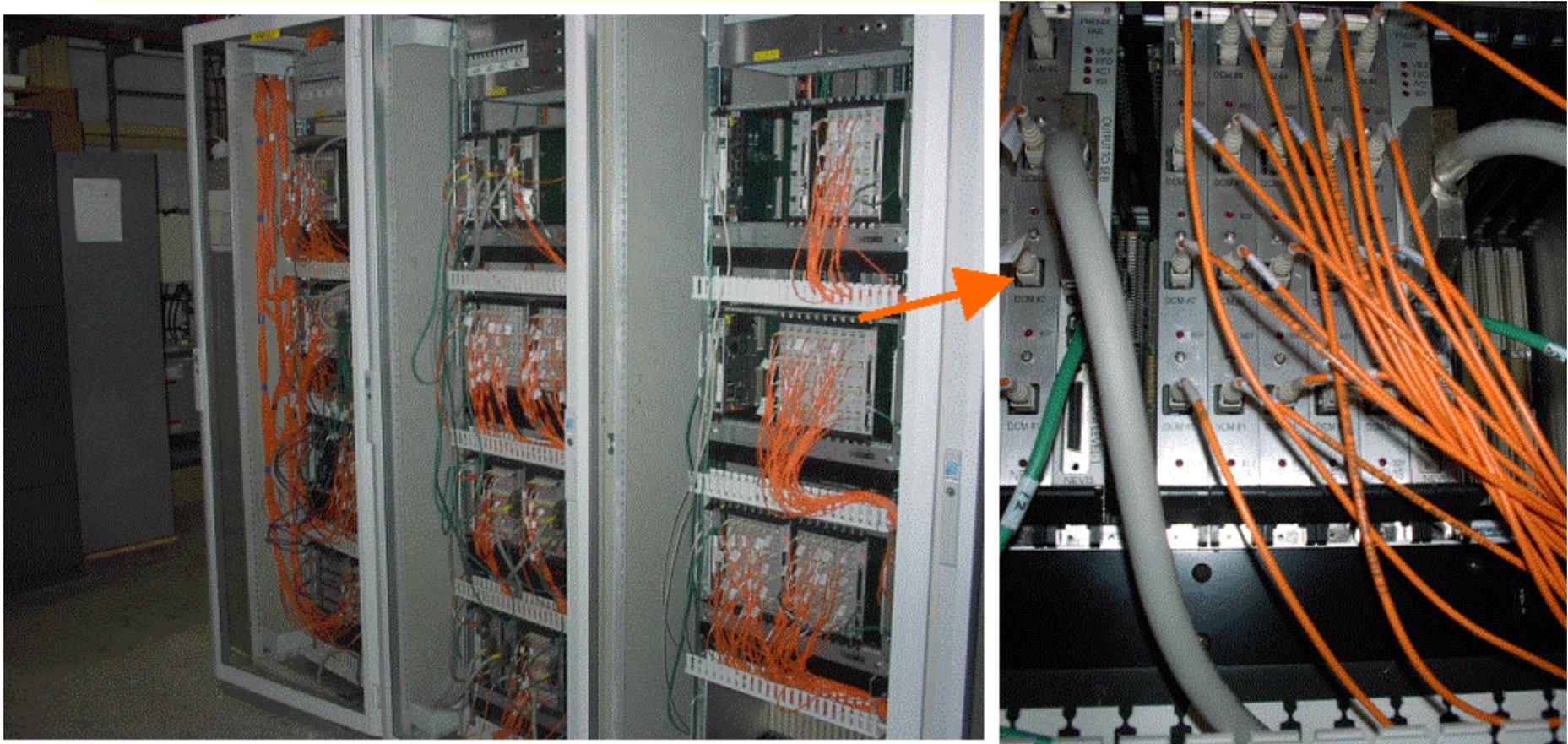
is the “conductor” of all FEM operations
sends a copy of the RHIC clock to the FEM
forwards the trigger signal to the FEM to
start digitizing/readout
Sends “Mode Bits” to the FEM.

Mode bits...

a concept that was (I believe) carried over
from CERN - a way to schedule certain
things to happen over and over again.
Over there it was called the “Yellow Box”

schedule resets of the AMU’s
resets of baseline for ADC
PPG events
much more

DCM's and DCM Boards



A whole bunch of DCM boards (4 DCM's) left, a group and two **partition** boards right. The gray cables go to the event builder.

Careful! "Partition" is used in two different meanings in PHENIX - here it means "dividing the Crate's backplane".

DCM's

The DCM is, at its core, a DSP (think specialized CPU)

it has some memory, and a slot for a daughtercard with a FPGA

the DCM receives the data from the FEM, zero-suppressed the data (these days, mostly done by that FPGA), and packages the data. This is where the packets in the PRDF are made.

Ships the packaged data to the partition module and on to the event builder.

4 DCM's + 1 more DSP are on one DCM Board (that's what you see).

GL1

Individual detectors can generate Local Level-1 signals - their way of saying "hey, I have something here that looks interesting. BBLL1, ZDC, ERT, several MUID LL1's, and so on.

They send their signal to the GL1 which takes the final decision.

GL1 receives LL1's plus other (some 130) signals to form the global trigger decision

Forwards this decision to MTM and so distributes it to all granules of the partition in question.

Run Control

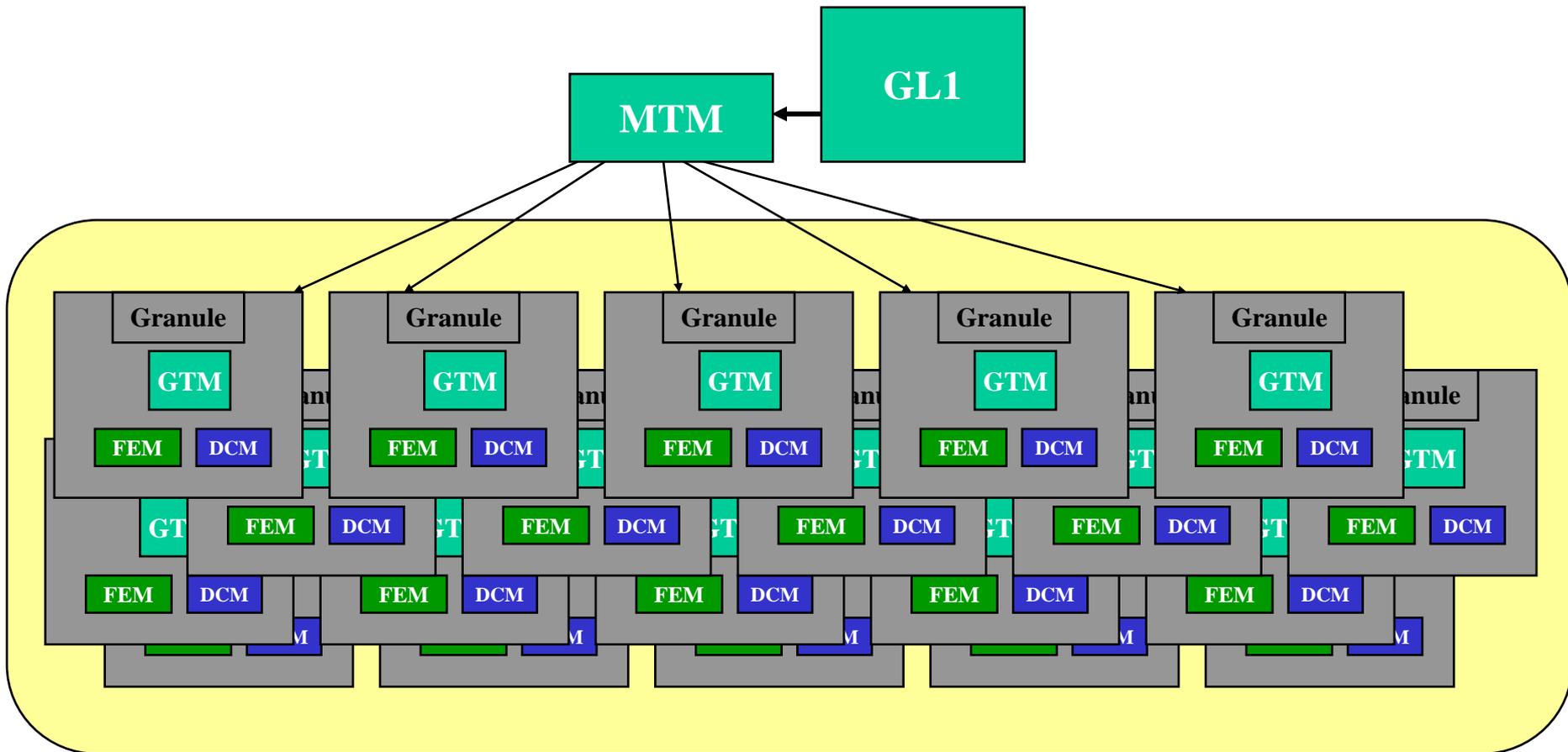
Someone or something has to control it all so it all happens in the proper order
That is Run Control, originally written by Steve Adler and now Ed Desmond's baby
controls every aspect of the DAQ, trigger, etc
Main user interface to the DAQ.

The screenshot displays the 'Run Control for Big Owned by 02' interface. It features several panels: 'Configured', 'BRL11 Status North Glirk', and 'South Glirk'. A 'Run Control Log' window is open, showing a list of commands such as 'Issuing command: set evb on', 'Issuing command: wall', 'Issuing command: down oac', 'Issuing command: set runmode physics', 'Issuing command: scaler read activate', 'Issuing command: scaler attach', 'Issuing command: start', and 'Issuing command: scaler y/lp read eur'. Below these panels are two large tables. The top table lists granules with columns for Name, L1, Run, Busy, OK, L1, DCM Status, S1KID, Name, #Events, Event Size, Data Rate, Buff Usage, Read Error, Bus y, OK, Name, #Events, #123kLept, #Read, Assen Rate, Ave Data Rate, STP OK, ET OK, L1, BK, and EBCO. The bottom table is a 'Scaler Monitor' showing the status of various triggers like TRIG, DCM, ZDCNS, ZDC1L1, ZDC1L1ntr0p, and various sub-detector triggers (ERT, FRT, MUE, MUE1, MUE2, MUE3, MUE4, MUE5, MUE6, MUE7, MUE8, MUE9, MUE10, MUE11, MUE12, MUE13, MUE14, MUE15, MUE16, MUE17, MUE18, MUE19, MUE20, MUE21, MUE22, MUE23, MUE24, MUE25, MUE26, MUE27, MUE28, MUE29, MUE30, MUE31, MUE32, MUE33, MUE34, MUE35, MUE36, MUE37, MUE38, MUE39, MUE40, MUE41, MUE42, MUE43, MUE44, MUE45, MUE46, MUE47, MUE48, MUE49, MUE50, MUE51, MUE52, MUE53, MUE54, MUE55, MUE56, MUE57, MUE58, MUE59, MUE60, MUE61, MUE62, MUE63, MUE64, MUE65, MUE66, MUE67, MUE68, MUE69, MUE70, MUE71, MUE72, MUE73, MUE74, MUE75, MUE76, MUE77, MUE78, MUE79, MUE80, MUE81, MUE82, MUE83, MUE84, MUE85, MUE86, MUE87, MUE88, MUE89, MUE90, MUE91, MUE92, MUE93, MUE94, MUE95, MUE96, MUE97, MUE98, MUE99, MUE100).

Let's now move on to the other meaning of "partition" in PHENIX...

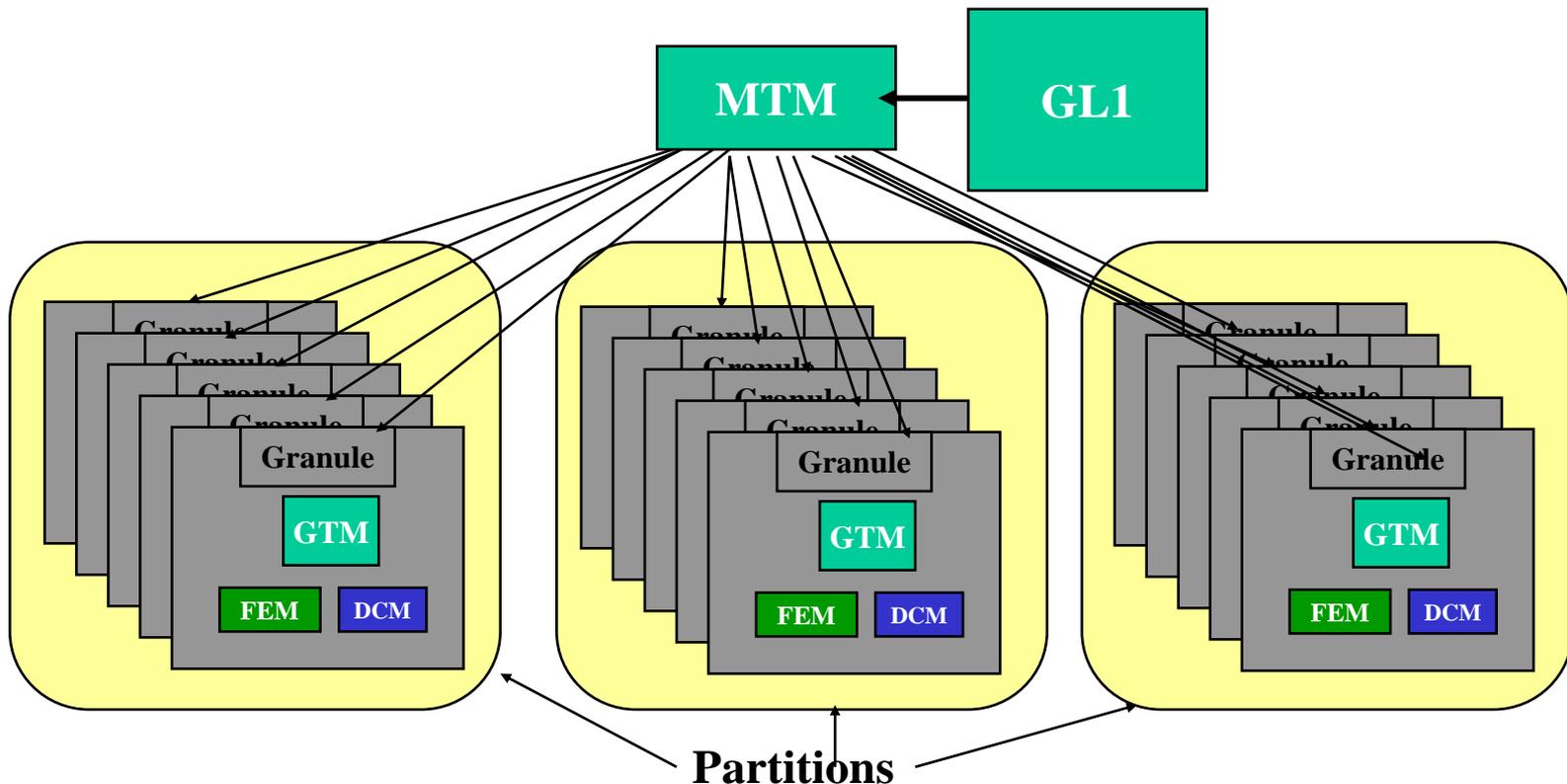
This is what you'd expect

Well, we just read out “the detector”. We know this a the “BigPartition”.



But that's a special case of this:

Partitioned running - the GL1 is designed to allow splitting the components into "Partitions", largely independent mini-DAQ's without re-arranging any cables. That's very handy during commissioning, calibration, etc. You can, for example, read out the EmCal by itself, concurrent with, say, Drift Chamber commissioning.



But...

That independence only goes so far - you share triggers, the *GL1*, etc.

Remember that a granule is defined by its *GTM* - you can have as many partitions as *GTM*, or 32, whatever is less. (And each *Granule* can belong to at most one *Partition*, obviously)

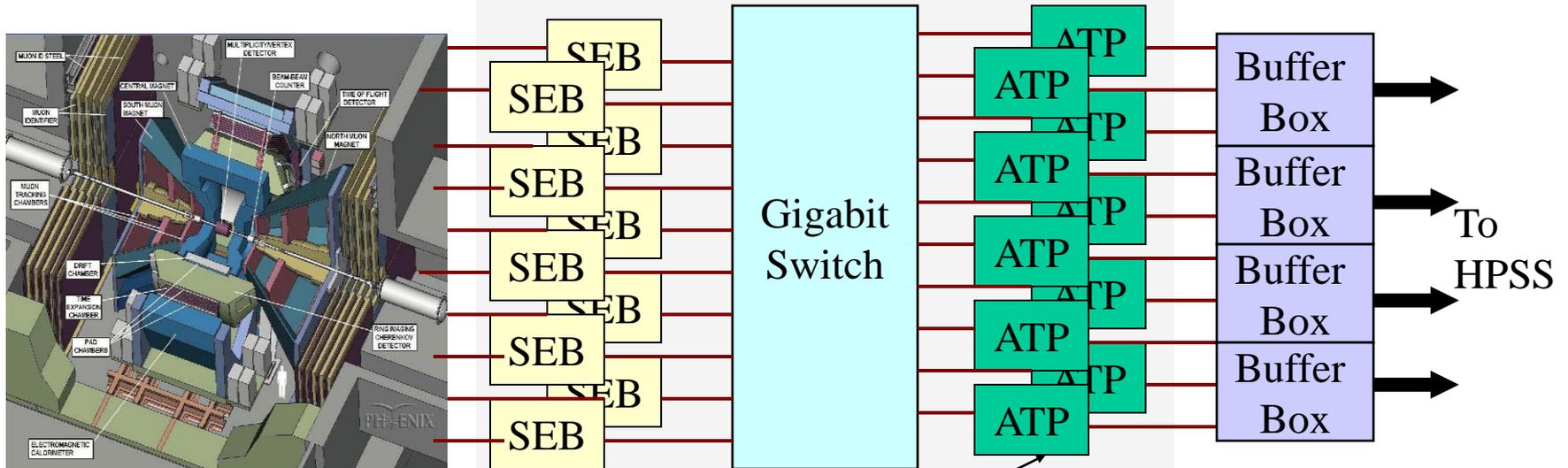
People tend to forget this once in a while -- the busy's are independent; that means that by and large, the individual partitions **DO NOT** read data from the same bunch crossings

It's really only for commissioning and debugging.

Let's move on to the back end now...

Data flow

Event Builder

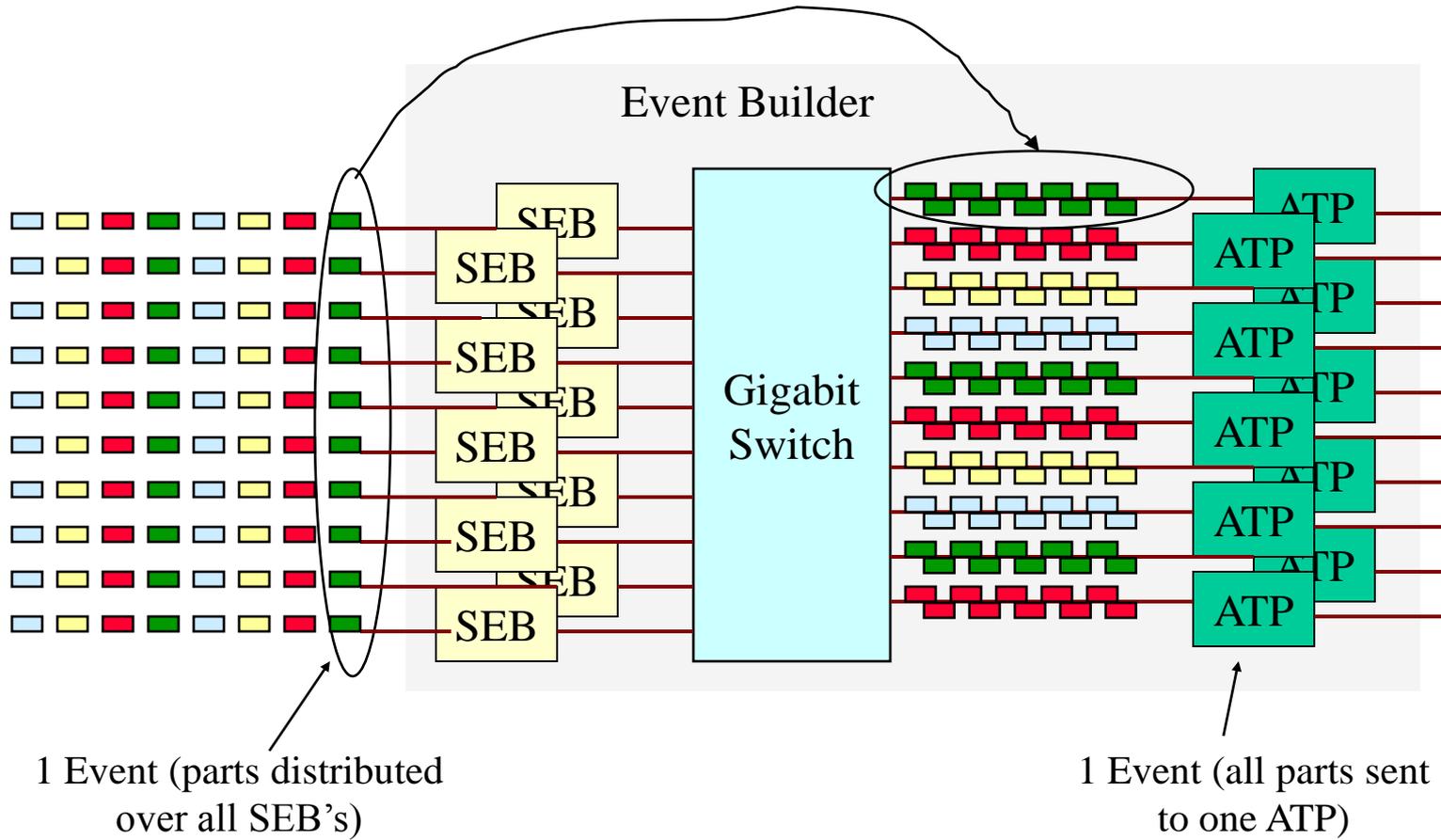


SubEvent Buffers (26)
(see parts of the event)

Assembly & Trigger
Processors (see whole
events)

Buffer boxes 2TB disk each
buffers approx. 12 hrs worth
of data before they are
shipped off to HPSS

Event Builder



The Buffer Boxes

Back in 2000, we started out with a SUN Enterprise System -- and got 8MB/s despite our best efforts.

We had what the experts had previously recommended, only to learn that another \$45K in hardware and licenses would finally get us what we wanted - perhaps.

Also, with SUN/SGI/IBM/... there is a byte-swapping issue between the buffer box and the Intel-based Event Builder

In 2001, we went with \$5K Linux PC's, and never looked back

In Fall 2002 we upgraded to the then-latest dual 2.4GHz E7500 power machines (SuperMicro), later added two newer ones @3.06GHz

Each one has 2 1TB file systems, ext3, Software Raid 0 striped across 12 or 14 disks, 2 SCSI controllers

iozone gives disk write speeds of 150-170MB/s for file sizes in our ballpark.

Network throughput is about 85MB/s in from the ATPs (regular network) and 100 MB/s out to HPSS (different network, Jumbo frames)

Transfer speeds...

Way back when, the PHENIX founding fathers based their data rate estimates on what was available then - Exabyte tapes at 1MB/s or so. Take 20 tape drives and 10 post-docs to change tapes, and you get 20MB/s. For a long time, that was the figure of merit.

We soon managed to get more - in Run 2, we saw about 40MB/s, and could improve that to about 70-80MB/s in Run 3. The LVL2 trigger group was extrapolating to 130MB/s. Then:

Dear Yasuyuki and Tony,

there are two more factors of 2 in Yasuyuki's point 7, [130MB/s](#) logging rate (with the existing setup we should be able to go to 2x80, by the way, see my slides which will appear in a few seconds).

We can go to 4 buffer boxes, which, assuming we get no more than the current 60MB/s*bufferbox, gives you [240](#).

Then my pet project, the compressed buffers from the DAQ, will shrink the late-stage output volume to about 45% of the original. Both combined would give you the equivalent of [500MB/s](#) in today's units (you would actually write < 250 to disk).

That would allow us to relax some scaledowns, and switch on LVL2 later in the luminosity ramp-up.

...and compression

That had indeed been a long-standing dream of mine, apply data compression (not just zero-suppression) in the DAQ. Used to be way too slow, doesn't run properly on some FE-CPU's, just couldn't be done before.

Let's take a little detour to see what changed this.

In 1999 or so, the ET software was re-engineered and improved at Jefferson Lab (Carl Timmer et al) from the "DD" software previously developed by us, nice technology transfer project.

It's still the workhorse system for accessing data online in real-time.

Allows a virtually unlimited number of clients to retrieve data according to an event profile

Is network-transparent

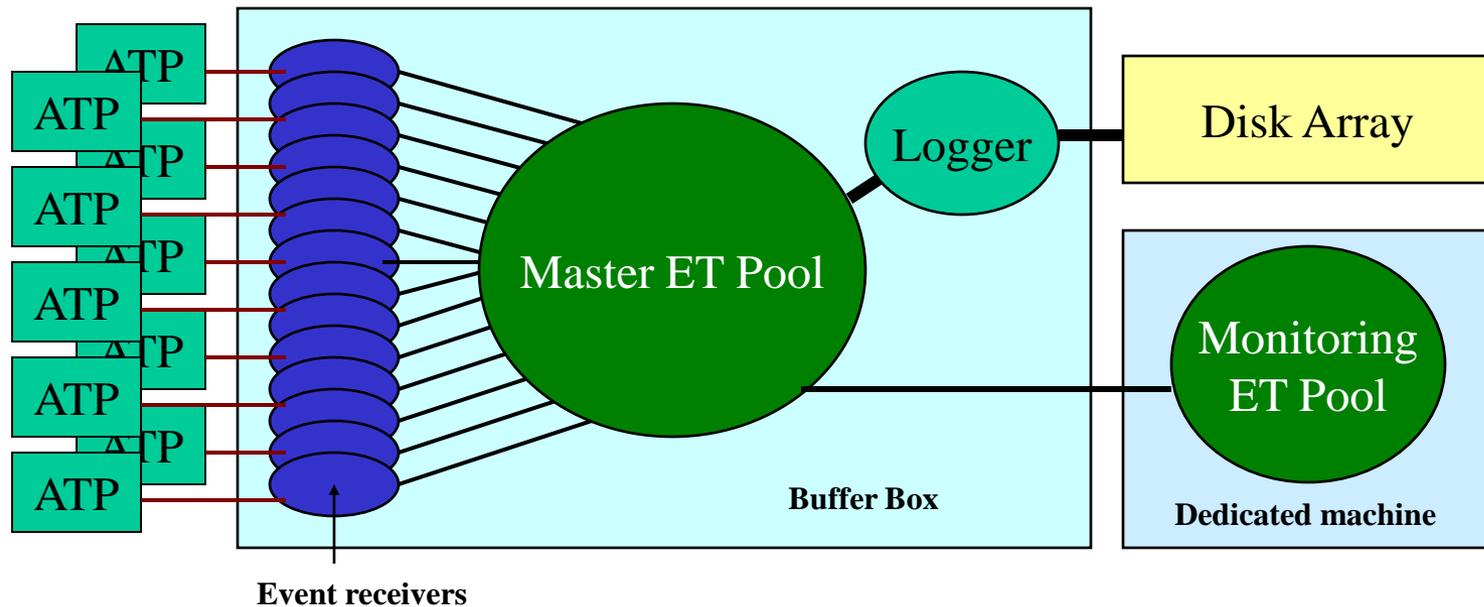
Supports all data access modes you can think of and then some - profile-select, load balancing shared, non-exclusive shared, ratio, guaranteed delivery, etc, etc. Very versatile.

Data Flow (old, Run 2)

Until Run 2, we transferred the data through an "ET" pool on the buffer boxes, which was our way to merge the data streams from the ATP's.

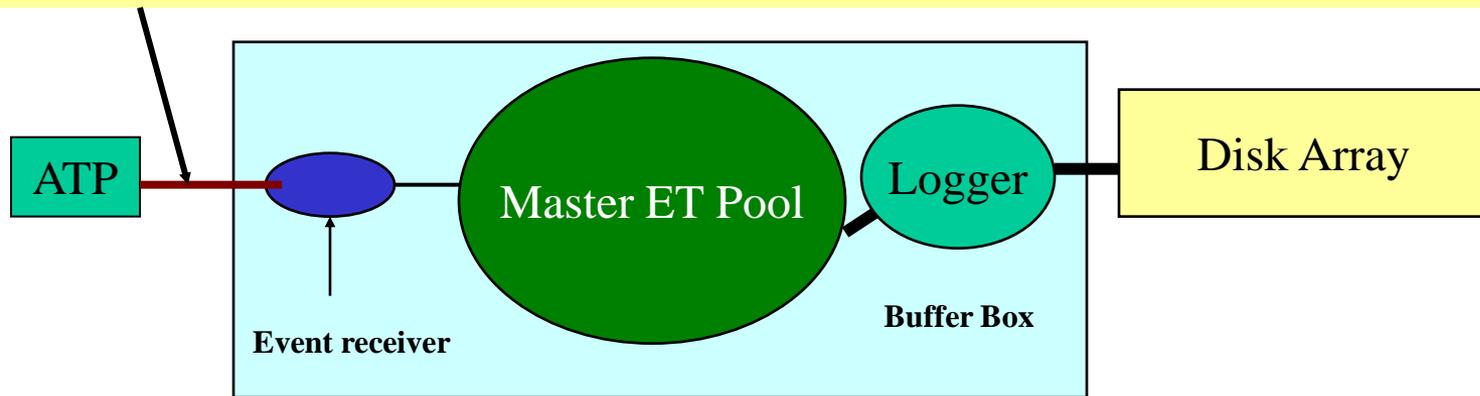
The logger would read the data from the pool, and write them to disk

A separate secondary monitoring pool is fed from the primary with about 20% of the data for online monitoring purposes



Buffers, not Events

We don't transfer single-event data chunks but buffers of events, strictly for network performance reasons -- larger transfers, more speed.



At the back end, we again write buffers...

And that's a lot of overhead -- buffers get disassembled into events, fed to the pool, re-assembled into buffers...

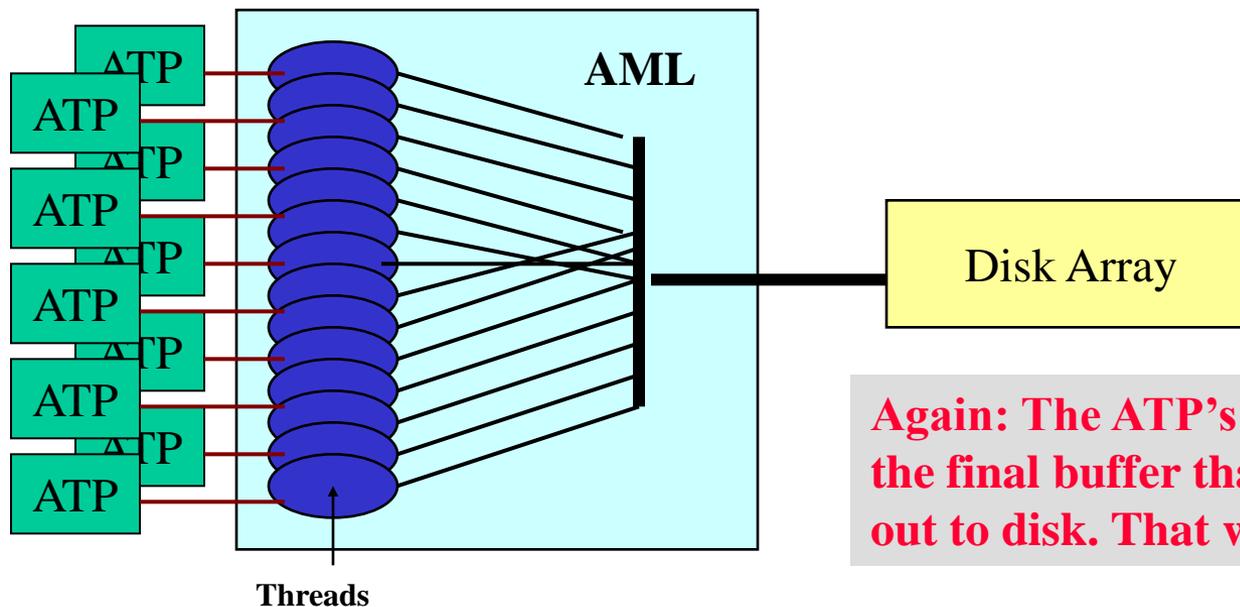
The new(er) stuff...

...called the **Advanced Multithreaded Logger (AML)**.

Advances in multi-threading in Linux allowed us to make a logger which receives data from a number of network sockets (ATP's), one thread per ATP handles the connection.

Writing to the output file can be easily regulated by semaphores (hence threads, not processes).

But the real gain is that the buffers sent from the ATP's are written to the file as they are. No disassembly/assembly step. Less overhead. Much easier.

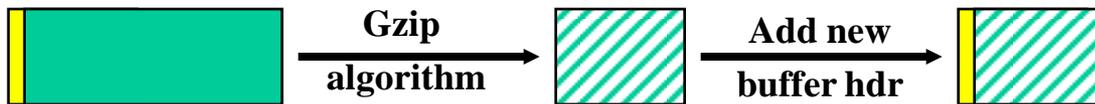


Again: The ATP's now assemble the final buffer that gets written out to disk. That was new then.

A PHENIX standard of compressed data, ca 1999

In the I/O layer:

This is what a file normally looks like



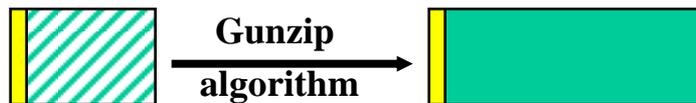
New buffer with the compressed one as payload



This is what

We used this for simulated data and other “slow” data (PPG, etc) - the compression is *really* slow...

On readback:



Original uncompressed

All this is handled completely in the I/O layer, the higher-level routines just receive a buffer as before.

The price of compression

Compressed PRDF's are expensive to get. We have used them so far for "slow" data such as calibrations. It's too computing-intensive to do on the buffer boxes.

```
/data> time dpipe -s f -d f -z  
      /common/b0/eventdata/EVENTDATAxxx_P01-0000077374-0000.PRDFFF xx.prdfz  
112.130u 12.290s 2:21.86 87.7%  0+0k 0+0io 301pf+0w
```

```
ls -l  /common/b0/eventdata/EVENTDATAxxx_P01-0000077374-0000.PRDFFF xx.prdfz  
1574395904 Mar 10 01:29  
      /common/b0/eventdata/EVENTDATAxxx_P01-0000077374-0000.PRDFFF  
699604992 Mar 10 23:37 xx.prdfz
```

699604992/1574395904 = 44%

We could shrink the output size to **below 50%** - but look at the CPU consumption. Impossible to do fast enough even on the fastest CPU.

But wait

Didn't we say that now with the AML, the ATP's send ready-made, final buffers? Couldn't the ATP's do the compression and send compressed buffers instead?

Indeed, that was the ticket. We could now distribute the compression workload over many CPU's.

We found that the standard compression engine of gzip ("compress2") was too slow, still. We needed another factor of 4 in speed or so.

A summer student at BNL, Luke St. Clair, helped me search for a better compression algorithm, 4x faster, comparable compression, open source, available on all platforms, etc.

We found LZO - Lempel-Ziv-Oberhumer. Demonstrated that it works. Some trade-off between speed and compression.

JiangYong then implemented LZO in the ATP code, and we use it routinely now.

LZO gives slightly worse compression than compress2, shrinks to ~60% (100MB become 60).

So the compression turns our 4x100MB/s raw data logging capacity into about 700MB/s in the EVB.

Most well-taken concerns (how do we analyze THAT much data?) addressed by our apparent ability to push 100 million events through RCF in a week

Some brainstorming going on to get another factor of 2, and forget about classic Level2 for good.

Data rates



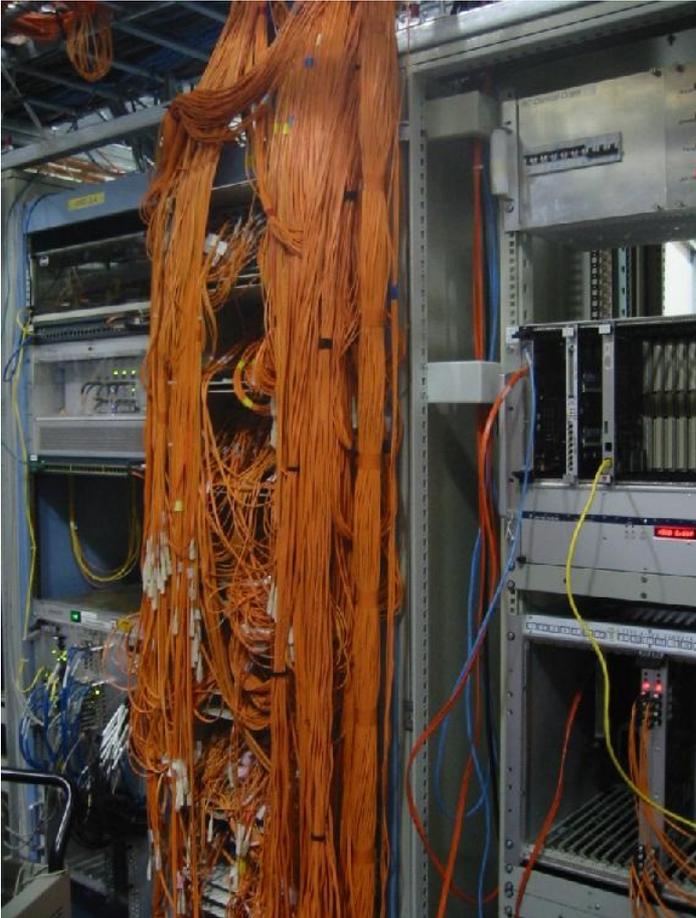
~240MB/s from DAQ
~180MB/s out to HPSS

— from DAQ
— to HPSS

Max capacity: 400MB/s from DAQ

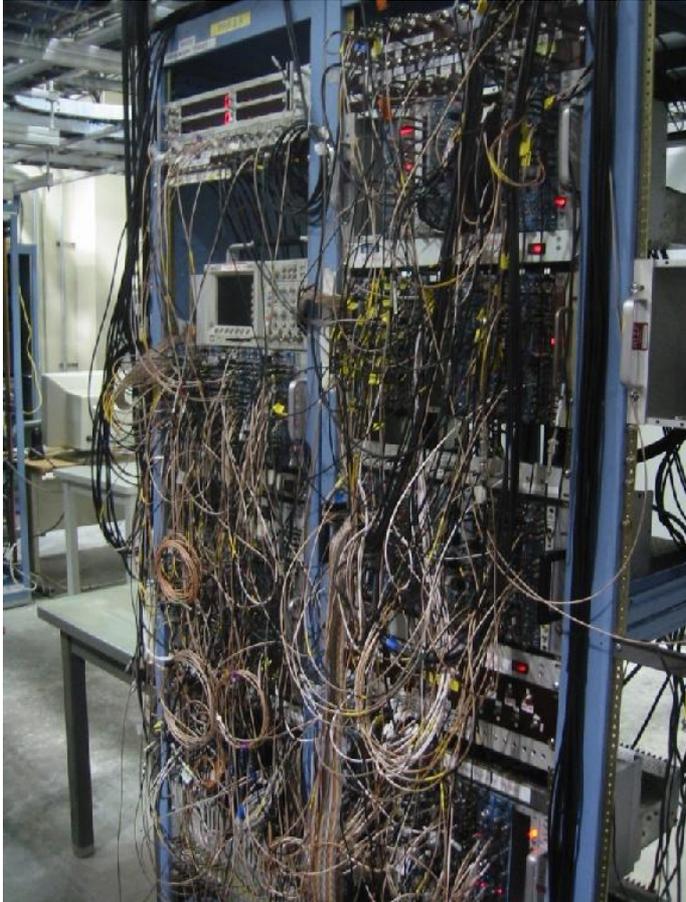
~220MB/s to HPSS (not the whole day, though)

Some more pictures



This is where the fibers arrive from the IR - all data go through here.

Blue Logic Trigger



The Event builder and VA farm



Seen from the back where there are fewer cables

The End. Thank you.

