THE UNIVERSITY of MISSISSIPPI

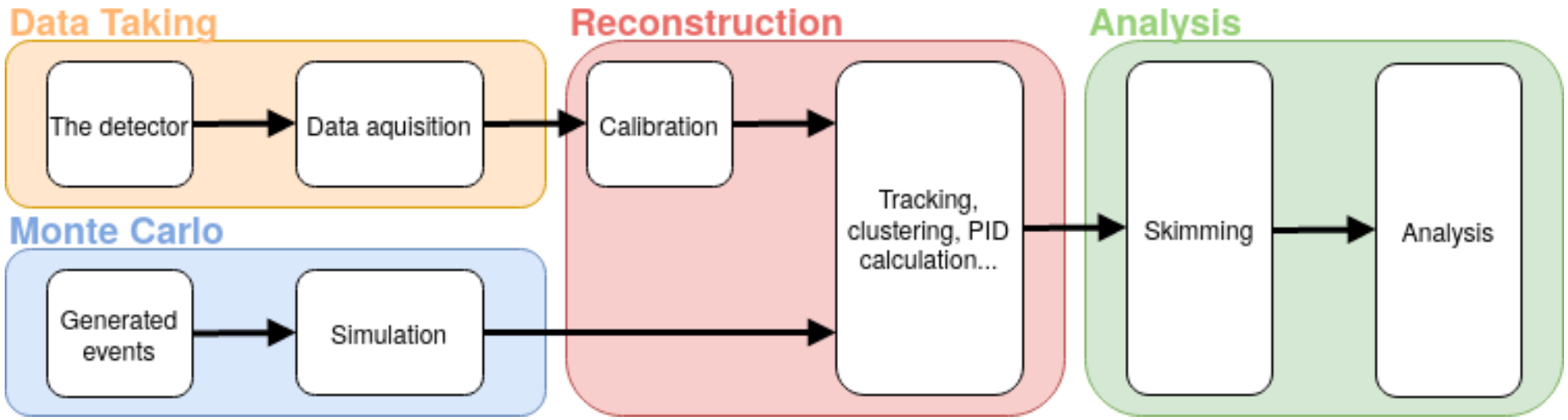
Belle II Analysis Tutorial

Paul Gebeline

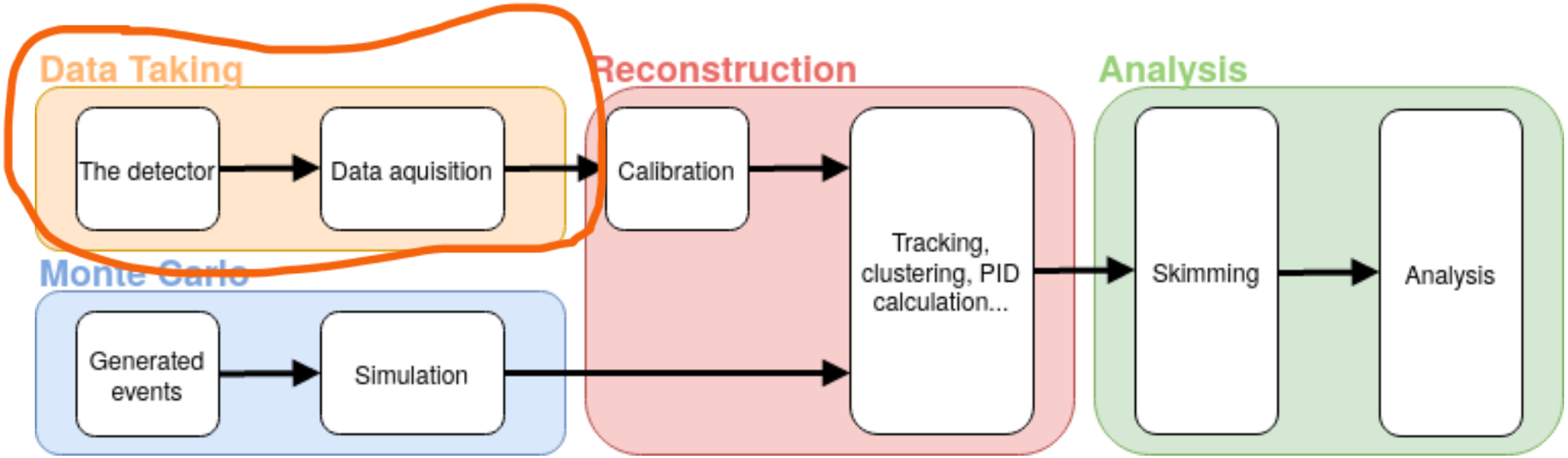
July 14, 2026

Detector → Publication

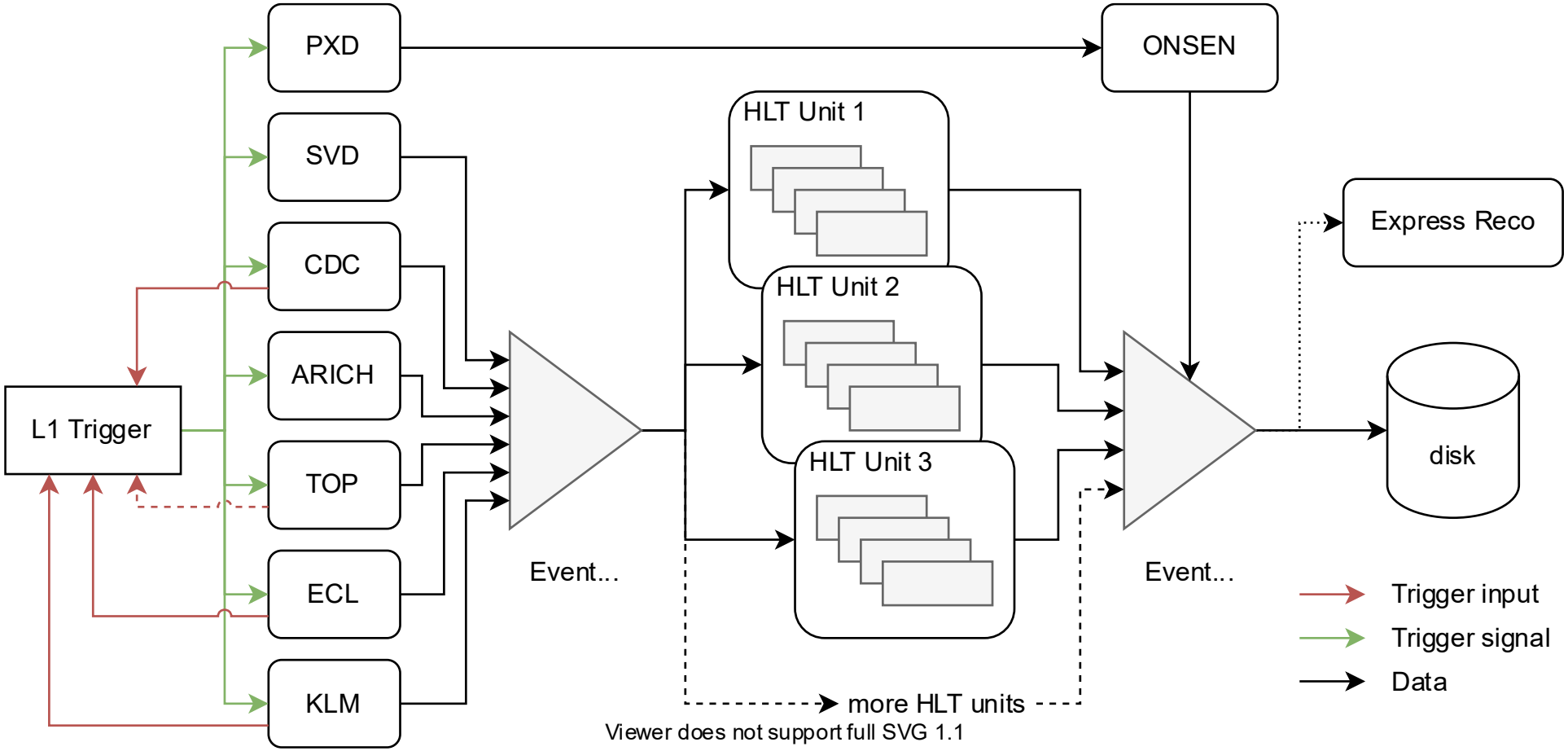
- At Belle II, the workflow leading from data taking to an actual published analysis is quite complex, and several steps can take **years.** Let's see what we can do in a week!

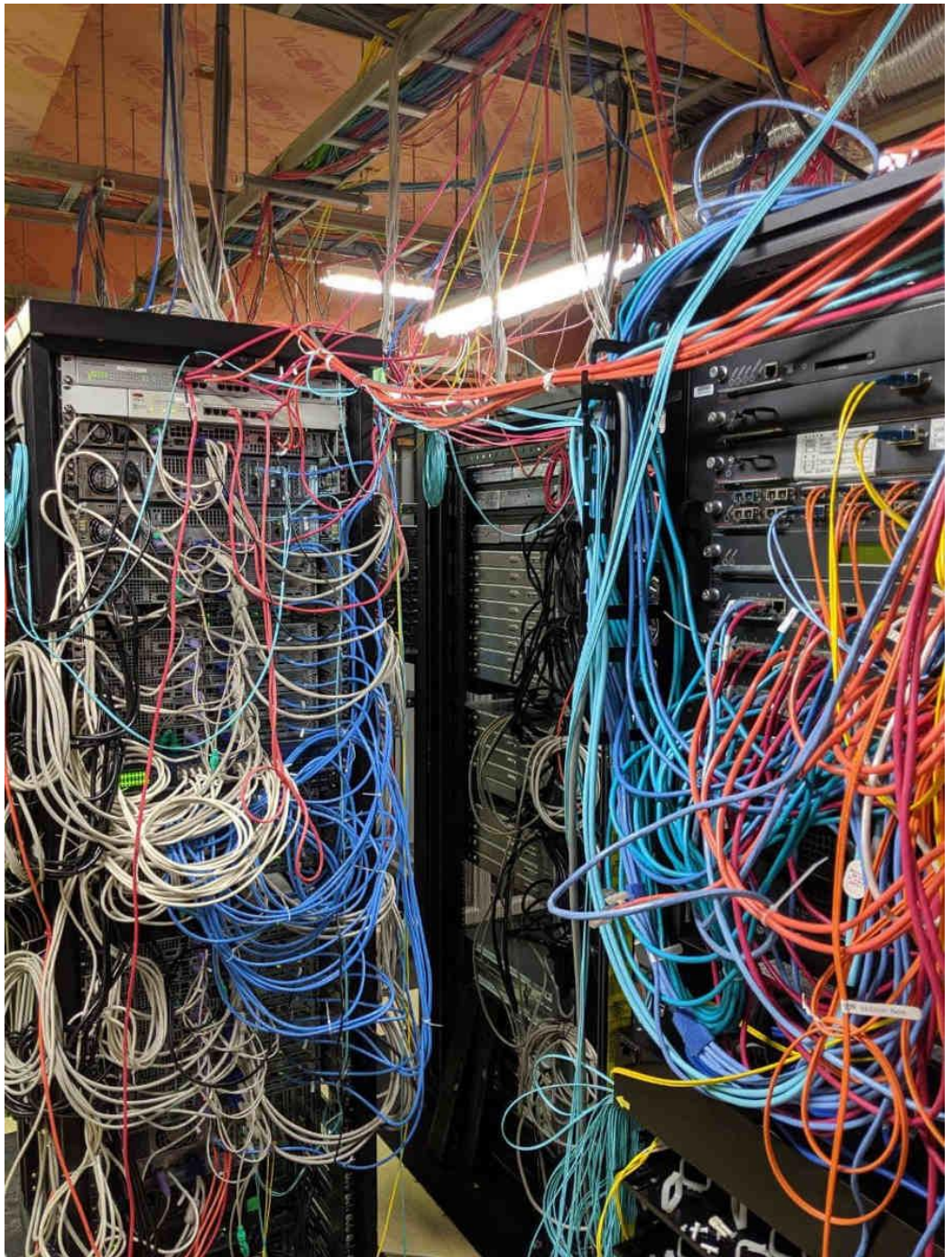


Data Acquisition



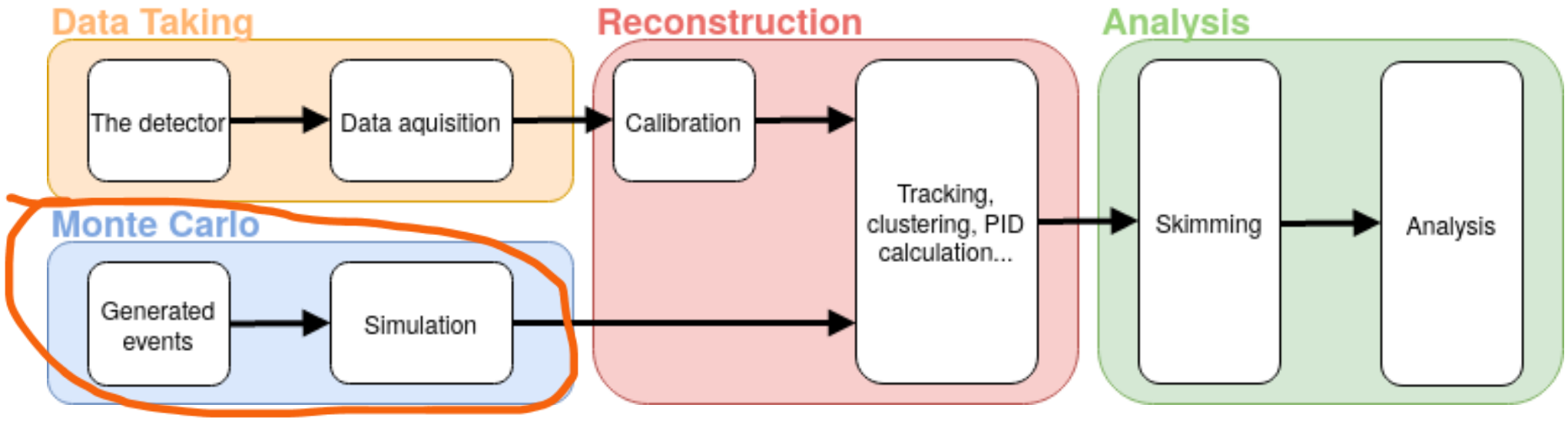
Data Acquisition





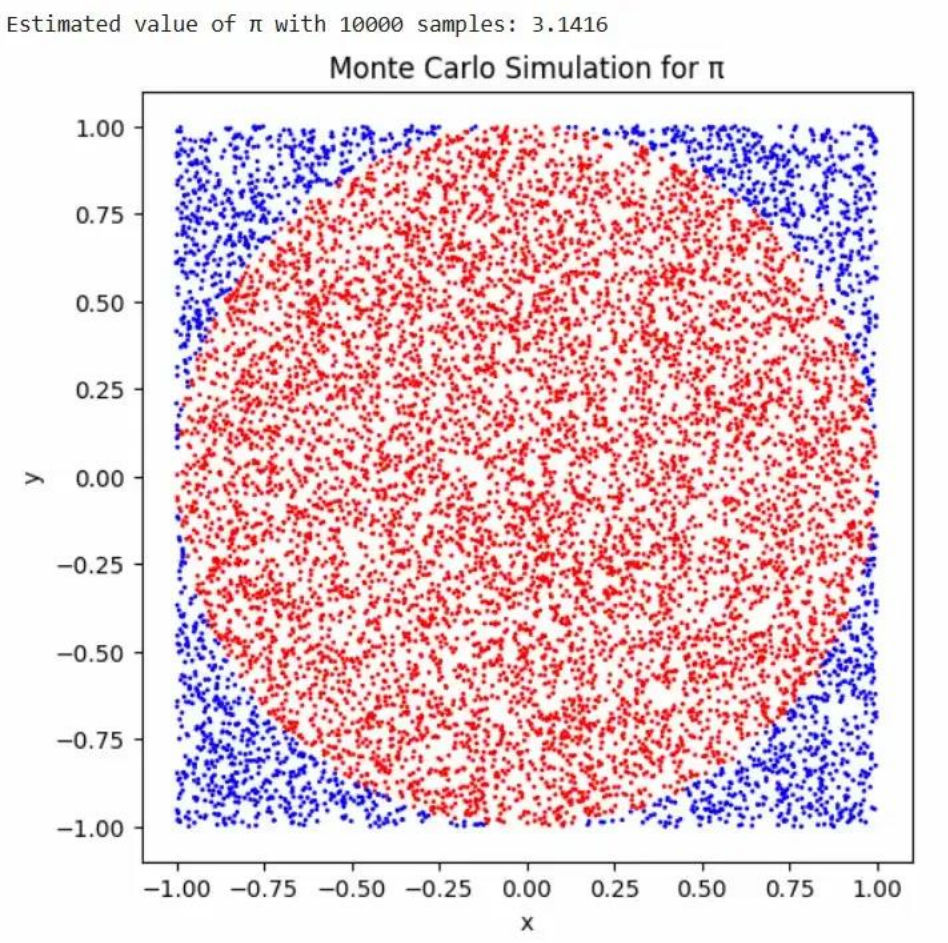
THE UNIVERSITY of MISSISSIPPI

Monte Carlo



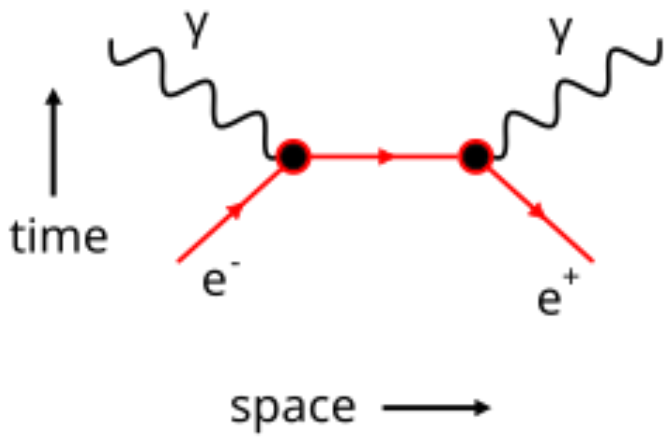
Monte Carlo

- We need something to compare data to: what is an artefact and what is a discovery?
- Simulate events by sampling random numbers repeatedly; in other words, we gamble (hence the name)
- The Monte Carlo (MC) processing consists of two key parts
  - Event Generation
  - Simulation



MC: Event Generation

- Given initial conditions of e^+e^- collision, generate particles according to the model we want to study
- Usually, we generate specific samples for our decay, called “Signal MC”, and compare it with simulation of standard model processes, or “Generic MC”
- Generation produces positions and momenta of particles, according to our model, using a *generator*
  - Different use cases: EvtGen, KKMC, Tauola, Madgraph…



MC: Simulation

- Need to make the event (now with position and momenta) look like real output from the detector
- A particle at some position and momentum will interact with the physical detector, causing a variety of processes that must be simulated
  - Ionisation, Cherenkov radiation, Scintillation, …
- **Ge**ometry **an**d **T**racking (Geant4) is open-source software from CERN that simulates the passage of particles through these matter subdetector systems
  - Turns our generated event into what we would see from a real event: deposited energy, decay products we'd spot in different subdetectors, and so on.

Lets make some MC!

- Part of this will require submitting jobs to our local computing cluster, so we can't have too many jobs running.
  - Split into X groups of Y (not sure at time of writing this…)
- Each group: Pick a decay you want to generate from Jake's talk yesterday, and login to our batch cluster

```
[psgebeli@Belle2HEPs-MBP ~ % ssh -L <xxxx>localhost:<xxxx> \
[>                              <user>@umiss001.hep.olemiss.edu
```

Generation with basf2

- Three steps to MC generation
  - Generate MC particles using an event generator
  - Simulate the detector/trigger response
  - Reconstruct tracks, electromagnetic clusters, etc
- Belle II's Analysis Software Framework 2 (basf2) makes this very easy – lets look at a script!

Generation with basf2

/belle2work/psgebeli/condor/epic26/y4s_gen.py

```python
#!/usr/bin/env python3

import basf2 as b2
import generators as ge
import simulation as si
import reconstruction as re
import mdst

# Create the steering path
main = b2.Path()

# Define number of events and experiment number
main.add_module('EventInfoSetter', evtNumList=[10], expList=[0])

# Generate B0B0bar events
ge.add_evtgen_generator(
    path=main,
    finalstate='signal',
    signaldecfile="/belle2work/psgebeli/condor/epic26/y4s.dec"
)

# Simulate the detector response and the L1 trigger
si.add_simulation(path=main)

# Reconstruct the objects
re.add_reconstruction(path=main)

# Create the mDST output file
mdst.add_mdst_output(path=main, filename='y4s.mdst.root')

# Process the steering path
b2.process(path=main)
```

THE UNIVERSITY of MISSISSIPPI

= model

Dec files

- Dec files are passed to the generator, and tell it the specific decays you wish to generate
- Your first task: make a .dec file for your decay of interest
- Resources
  - The script on this slide is /belle2work/psgebeli/condor/epic26/y4s.dec
  - /belle2work/psgebeli/condor/epic26/DECAY_BELLE2.dec is the “central” dec file used by Belle 2. Use it to find the syntax for your particles, and which models to use

```
Alias MyB+ B+
Alias MyB- B-

Decay Upsilon(4S)
1.0 MyB+ MyB- VSS;
Enddecay

Decay MyB+
1.0 mu+ nu_mu PHOTOS SLN;
Enddecay

Decay MyB-
1.0 pi- D0 PHOTOS PHSP;
Enddecay

Decay D0
0.2   K-  pi+           PHOTOS PHSP;
0.2   K-  pi+ pi0       PHOTOS PHSP;
0.2   K-  pi+ pi+ pi-   PHOTOS PHSP;
0.2   K-  K+            PHOTOS PHSP;
0.2   pi- pi+           PHOTOS PHSP;
Enddecay
CDecay anti-D0

End
```

Generation with basf2

/belle2work/psgebeli/condor/epic26/y4s_gen.py

Step 2: make your own generation script!

```
#!/usr/bin/env python3

import basf2 as b2
import generators as ge
import simulation as si
import reconstruction as re
import mdst

# Create the steering path
main = b2.Path()

# Define number of events and experiment number
main.add_module('EventInfoSetter', evtNumList=[10], expList=[0])

# Generate B0B0bar events
ge.add_evtgen_generator(
    path=main,
    finalstate='signal',
    signaldecfile="/belle2work/psgebeli/condor/epic26/y4s.dec"
)

# Simulate the detector response and the L1 trigger
si.add_simulation(path=main)

# Reconstruct the objects
re.add_reconstruction(path=main)

# Create the mDST output file
mdst.add_mdst_output(path=main, filename='y4s.mdst.root')

# Process the steering path
b2.process(path=main)
```

Running the Generation

- This is a python script, but it is executed with basf2 software. We need to source it
  source /cvmfs/belle.cern.ch/tools/b2setup release-08-03-01
- Then, we could run this script directly:
  basf2 my_gen_script.py
- This will generate 10 events. But particle physics is a game of statistics. We will aim for 100,000 events, but if you try to run that on your laptop (eg *basf2 my_gen_script.py –n 100000*), you will notice that it will take a very long time.
  - How can we generate 100,000 events in a reasonable amount of time?

THE UNIVERSITY of MISSISSIPPI

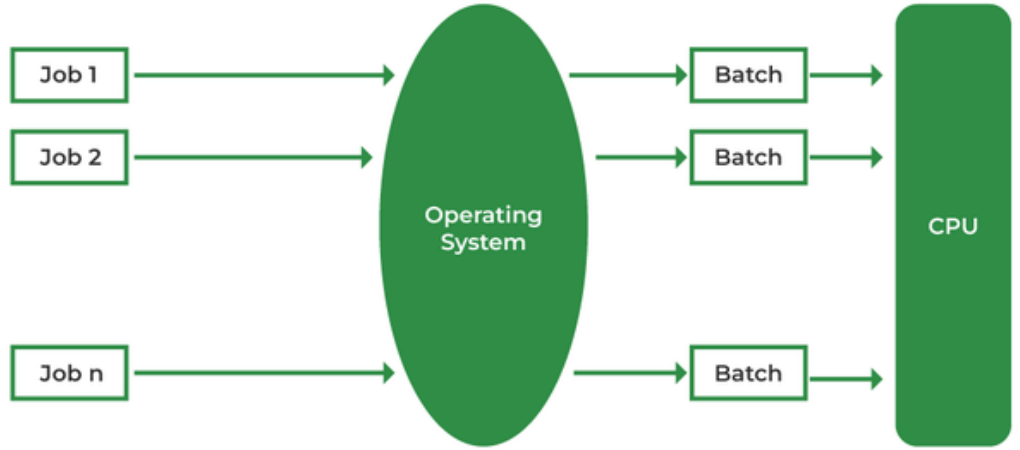
Submitting Jobs to Batch Systems

Paul Gebeline

MESH 2026

What are batch systems?

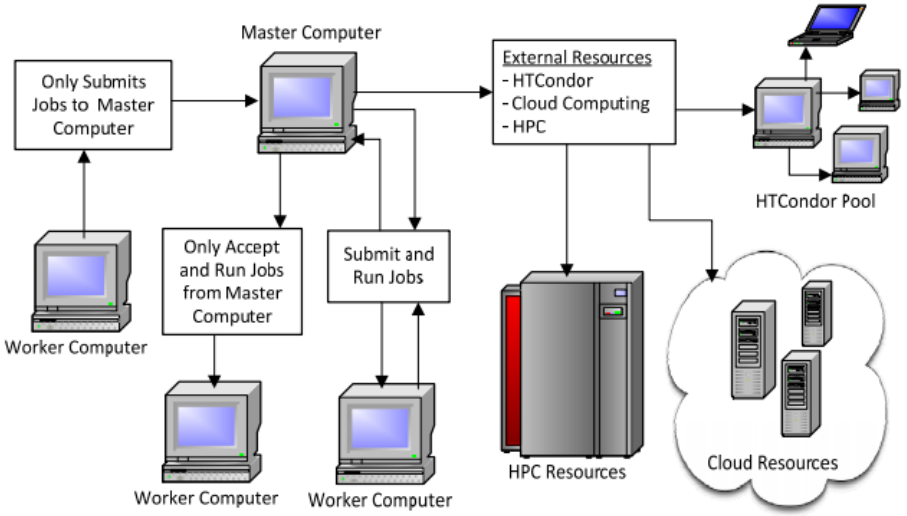
- A computer executes jobs in “batches” without user intervention
  - Jobs prepared offline then submitted to some architecture
  - Little manual effort required: just write a script then go make some coffee
  - Can handle large amounts of data
- Examples include **HTCondor,** gbasf2, LSF, Slurm, …
  - They mainly differ by syntax



HTCondor Software Suite

HTCondor

- Job descriptions submitted to the queue on an **Access Point or AP** (a machine that may submit HTCondor jobs)
  - User creates a submit file describing the job, typically pointing to a bash executable script and defining output file locations
  - HTCondor then locates a machine that can run each job **(Execution Point, or EP**), packages it up, and sends it to that machine
- A Job may use multiple cores on one EP, but may not (in general) run across multiple Eps
- Jobs run asynchronously to each other




```
Executable = /belle2work/psgebeli/condor/sigmakpi_akstar.sh
Universe = vanilla
Log = /belle2work/psgebeli/condor/logs/sigmakpi_akstar.log
Output = /belle2work/psgebeli/condor/logs/sigmakpi_akstar.out
Error = /belle2work/psgebeli/condor/logs/sigmakpi_akstar.err

Queue
```

.submit script example

Hands-on: HTCondor at UMiss

The executable

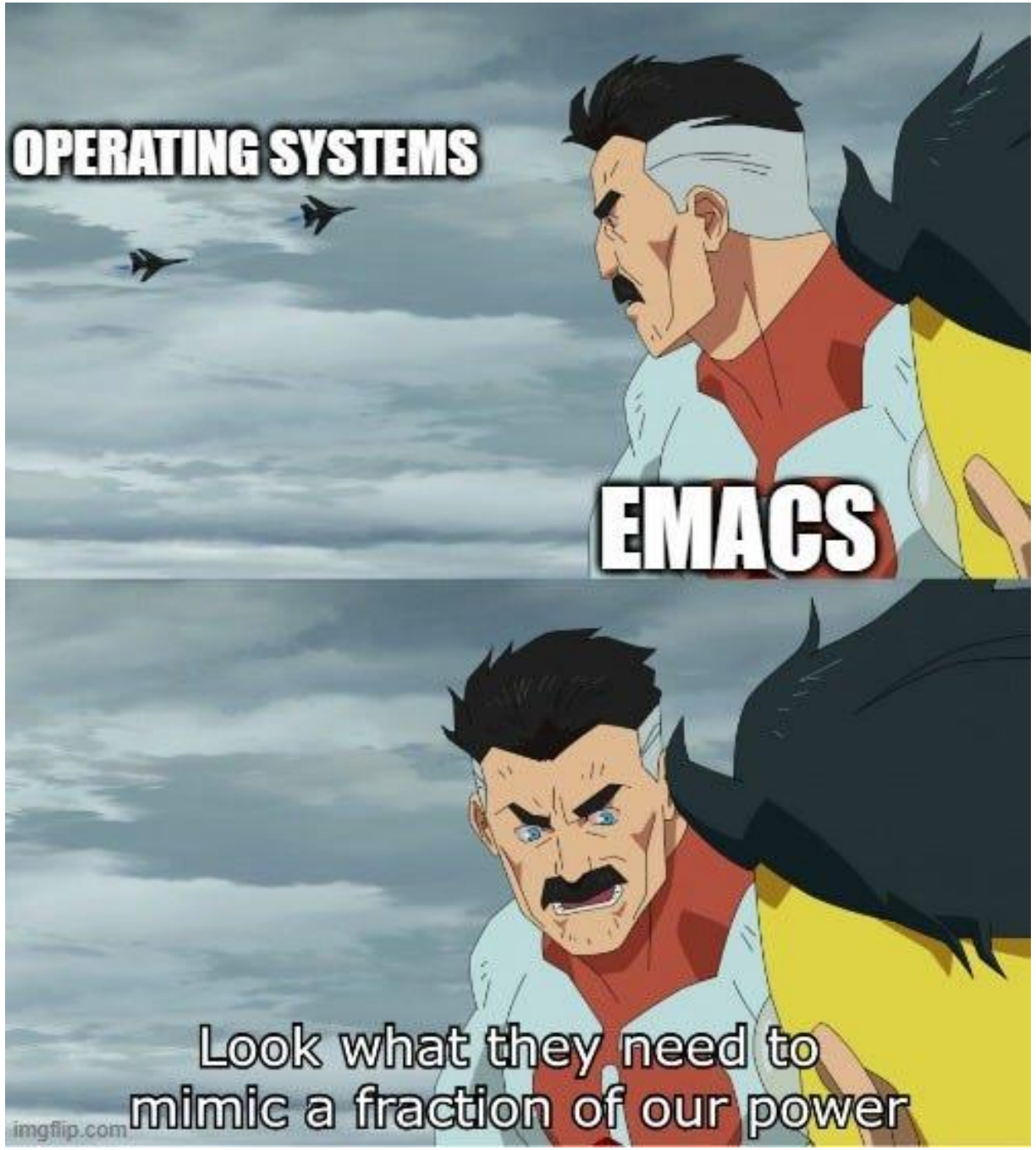
- umiss001 is a linux system, so let's create a shell script to run our generation *[emacs –nw] gen.sh* (use your favorite text editor, but emacs is best)

  /belle2work/psgebeli/condor/epic26/y4s.sh

```
#!/bin/bash
source /cvmfs/belle.cern.ch/sl6/tools/b2setup release-08-03-01 # source the basf2 software
basf2 /belle2work/psgebeli/condor/mesh26-handson/y4s/y4s_gen.py -n 1000 -o y4s.root # generate 1000 MC events
#    |-----------------------------------------------------|
#         FULL path - shared file system!!
```

Condor needs to be able to execute the script – it must be an *executable*

chmod u+x sleep.sh


OPERATING SYSTEMS
EMACS
Look what they need to
mimic a fraction of our power
imgflip.com

Submitting the job

- We will need a .sub file to submit the condor process
- Modify the .sub file at /belle2work/psgebeli/condor/epic26/y4s.sub to submit 100,000 generated events of your decay
- The $(Process) macro substitutes the process # (1, 2, 3…) into any field of your submit file, so you can give each job a unique directory and avoid output files overwriting each other. But the directories must already exist.

  mkdir –p run{0..99}

  Can you fill out the submission script to submit **100 jobs?**

```
executable              = y4s.sh

should_transfer_files   = no

# Output directory for job
initialdir              = run$(Process)

output                  = run$(Process).out
error                   = run$(Process).err
# jobs share ONE log!
log                     = ???

request_cpus            = 1
request_memory          = 2G
request_disk            = 2G

max_retries             = 2

queue ???
```

Submitting the job

- Question: are we ready to go? Should we submit this .sub file?

```
executable              = y4s.sh

should_transfer_files   = no

# Output directory for job
initialdir              = run$(Process)

output                  = run$(Process).out
error                   = run$(Process).err
# jobs share ONE log!
log                     = ../gen.log

request_cpus            = 1
request_memory          = 2G
request_disk            = 2G

max_retries             = 2

queue 100
```

Submitting the job

- Question: are we ready to go? Should we submit this .sub file?
- Answer: **NO!** Test it first, or David will get very angry

```
executable              = y4s.sh

should_transfer_files   = no

# Output directory for job
initialdir              = run$(Process)

output                  = run$(Process).out
error                   = run$(Process).err
# jobs share ONE log!
log                     = ../gen.log

request_cpus            = 1
request_memory          = 2G
request_disk            = 2G

max_retries             = 2

queue 100
```

Submitting the job

- Question: are we ready to go? Should we submit this .sub file?
- Answer: **NO!** Test it first, or David will get very angry
  - Modify your .sh script to generate, say, 10 events
  - Modify the .sub script to submit one job
  - Only if this succeeds should we submit 100 jobs of 1000 events.

```
executable              = y4s.sh

should_transfer_files   = no

# Output directory for job
initialdir              = run$(Process)

output                  = run$(Process).out
error                   = run$(Process).err
# jobs share ONE log!
log                     = ../gen.log

request_cpus            = 1
request_memory          = 2G
request_disk            = 2G

max_retries             = 2

queue 100
```

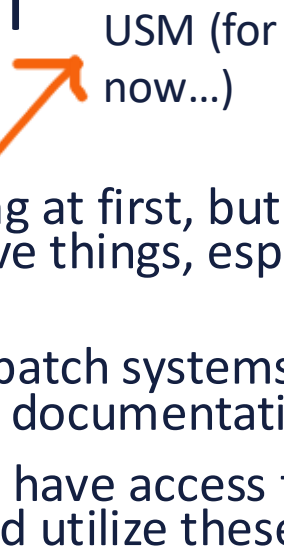
Submission and monitoring

```
[[psgebeli@umiss001 test]$ condor_submit y4s_test.sub                    ]
Submitting job(s).
1 job(s) submitted to cluster 1955791.
[[psgebeli@umiss001 test]$ condor_q                                      ]


-- Schedd: umiss001.hep.olemiss.edu : <130.74.15.11:9618?... @ 07/14/26 10:08:24
OWNER    BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB IDS
psgebeli ID: 1955791   7/14 10:08      _      1       _      1 1955791.0

Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 su
spended
Total for psgebeli: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0
 suspended
Total for all users: 4 jobs; 0 completed, 0 removed, 0 idle, 4 running, 0 held,
0 suspended
```

When you no longer have a process running, investigate the different logs .err, .out, .log.

If there are no issues, go ahead and submit 100 jobs of 1000 events (AFTER calling me or other grad student/faculty over to verify you aren't going to cause the apocalypse)

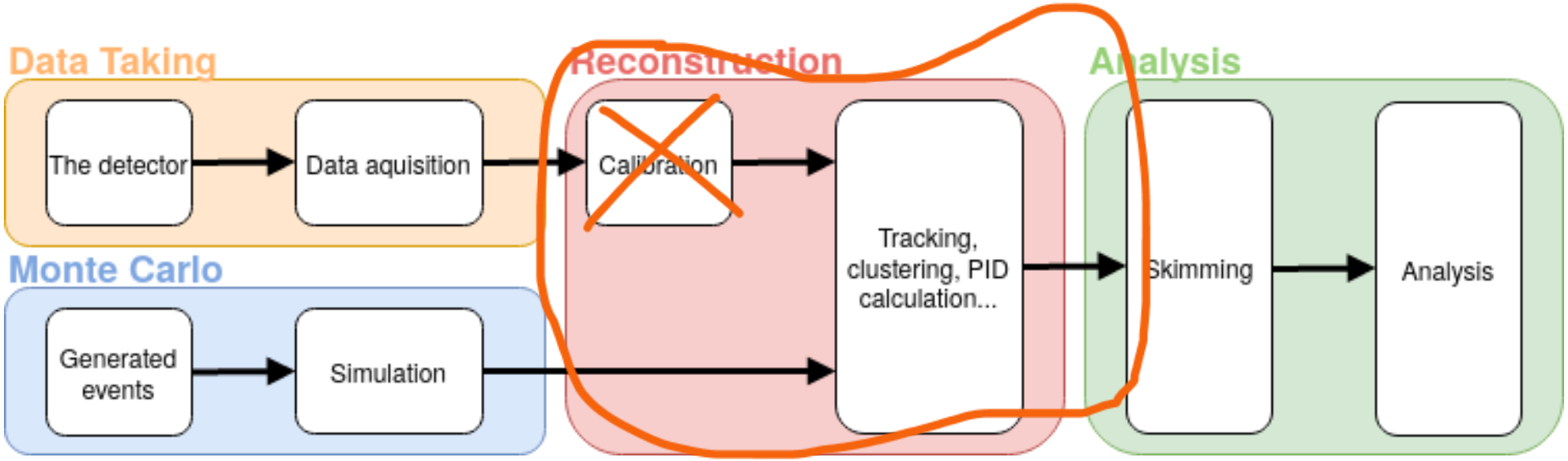
Summary of batch



- Batch systems may seem intimidating at first, but once learned, they are an incredibly useful way to run repetitive things, especially some physics script over many different input files
- Once you learn HTCondor, all other batch systems are equally intuitive: you just have to learn their syntax from their documentation.
- As a student/faculty at ole miss, you have access to computing resources. Stop running scripts on your 2007 Dell and utilize these – it is much, much faster!
  - To be concrete, generating 500,000 events took less than five minutes on HTCondor, while doing it with basf2 –n 500000 on my laptop has been running since before lunch (over 2 hrs)
- Learn bash and batch systems, it will make your physics life a lot easier.
- When your jobs are done, run

  b2file-merge my_gen.root run/my_mdst_output.root*

Reconstruction



Reconstruction

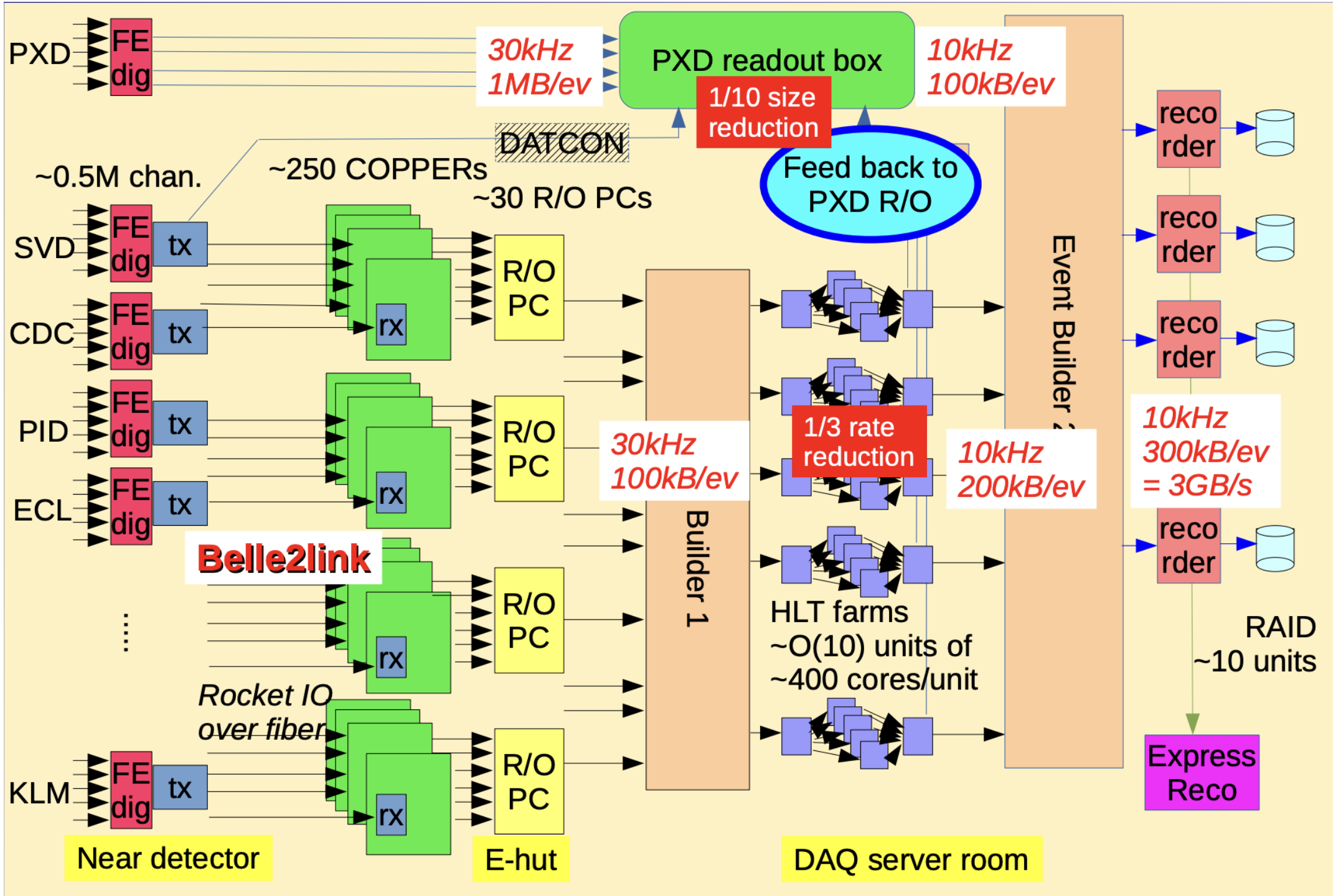
- Reconstruction converts raw detector signals into particle candidates by estimating the particles' trajectories and energies.
- Because of detector noise and short-lived particles, reconstruction cannot perfectly identify every particle—it finds the most likely interpretation of the event, and will likely have plenty of “background”
- The same reconstruction algorithms are applied to both real data and simulated (MC) data. For MC, we can compare the reconstructed particles to the known "MC Truth" to evaluate performance.
- This is also easy (but more involved than generation) with basf2!

Reconstruction

- I have prepared a walkthrough at */belle2work/psgebeli/condor/epic26/xicrec.ipynb*
  - This reconstructs a particular mode other than the ones you are studying.
  - This is intentional. You need to modify this script to work for your decay mode! Walk through the exercise, and change what is needed.
  - We are here to help.
    - *jupyter notebook --port=XXXX*

Thank you!

Backup


PXD
FE dig
30kHz
1MB/ev
PXD readout box
1/10 size reduction
10kHz
100kB/ev
DATCON
Feed back to PXD R/O
~0.5M chan.
~250 COPPERs
~30 R/O PCs
SVD
CDC
PID
ECL
KLM
tx
rx
R/O PC
Belle2link
Rocket IO over fiber
30kHz
100kB/ev
Builder 1
1/3 rate reduction
10kHz
200kB/ev
Event Builder 2
HLT farms
~O(10) units of
~400 cores/unit
reco rder
10kHz
300kB/ev
= 3GB/s
RAID
~10 units
Express Reco
Near detector
E-hut
DAQ server room