Computation in Experimental Nuclear Physics
(Particularly for CLAS at Jefferson Lab)

David P. Heddle

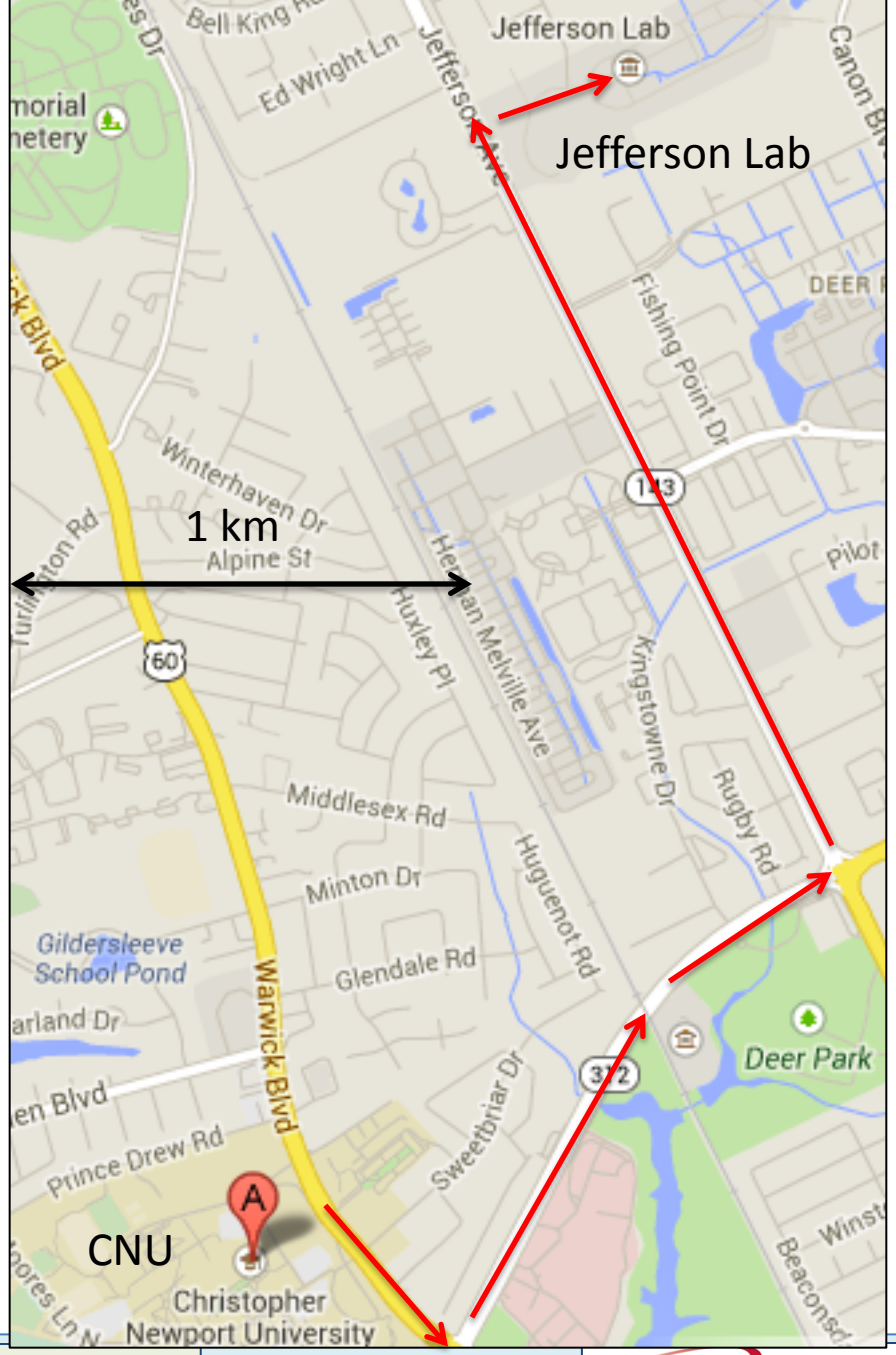
Christopher Newport University and Jefferson Lab

david.heddle@cnu.edu

I have the best job. Ever.

I teach at a new university with great students.

My university (Christopher Newport www.cnu.edu) is five minutes from Jefferson Lab!



Goal of this Class

- Appreciate the role of computation and software in experimental nuclear and particle physics
- More of a “big picture” overview rather than details
- Demonstrate that experimental physics has great opportunities for young researchers who are skilled at software

Outline

1. Introduction to Jefferson Lab and CLAS (also, see lectures from Dr. Elouadrhiri
2. Categories of software in a nuclear physics experiment
   A. Data Acquisition and Triggering
   B. Visualization and Monitoring
   C. Simulation
   D. Reconstruction
   E. Physics Analysis (briefly mentioned)
3. A segment finding algorithm

(crude) Physicist Zoology

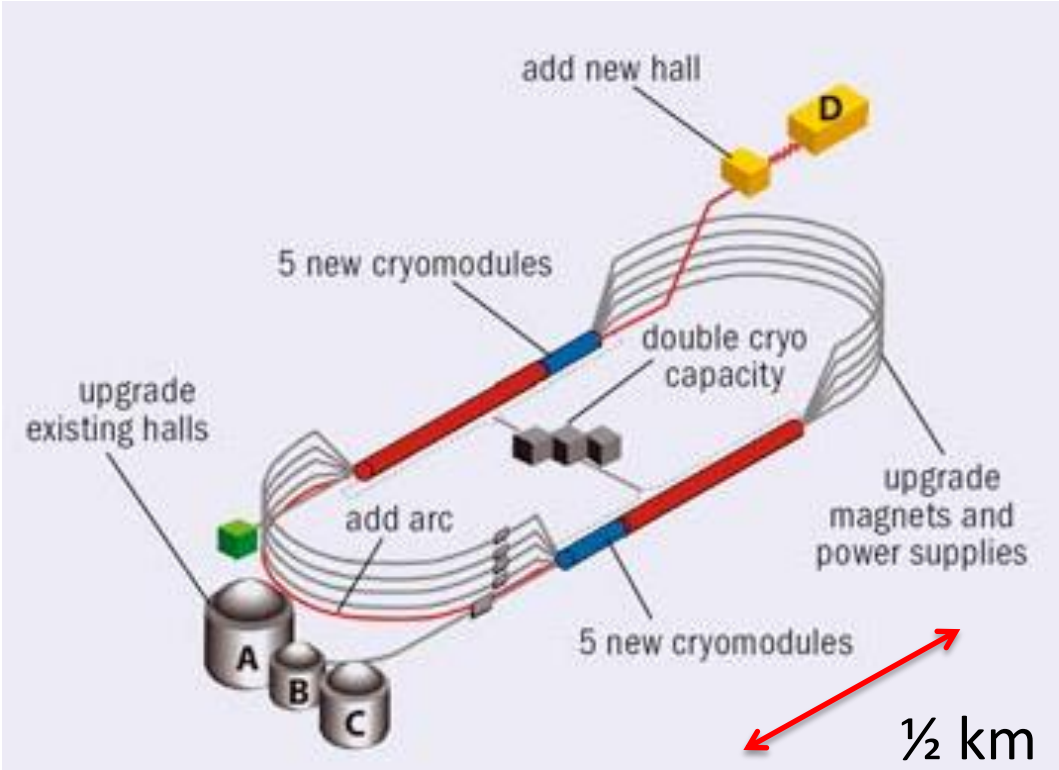
Solid State
High Energy
Nuclear
Astrophysics
Cosmology
(etc)

Theorist
Experimentalist

Hardware
Software

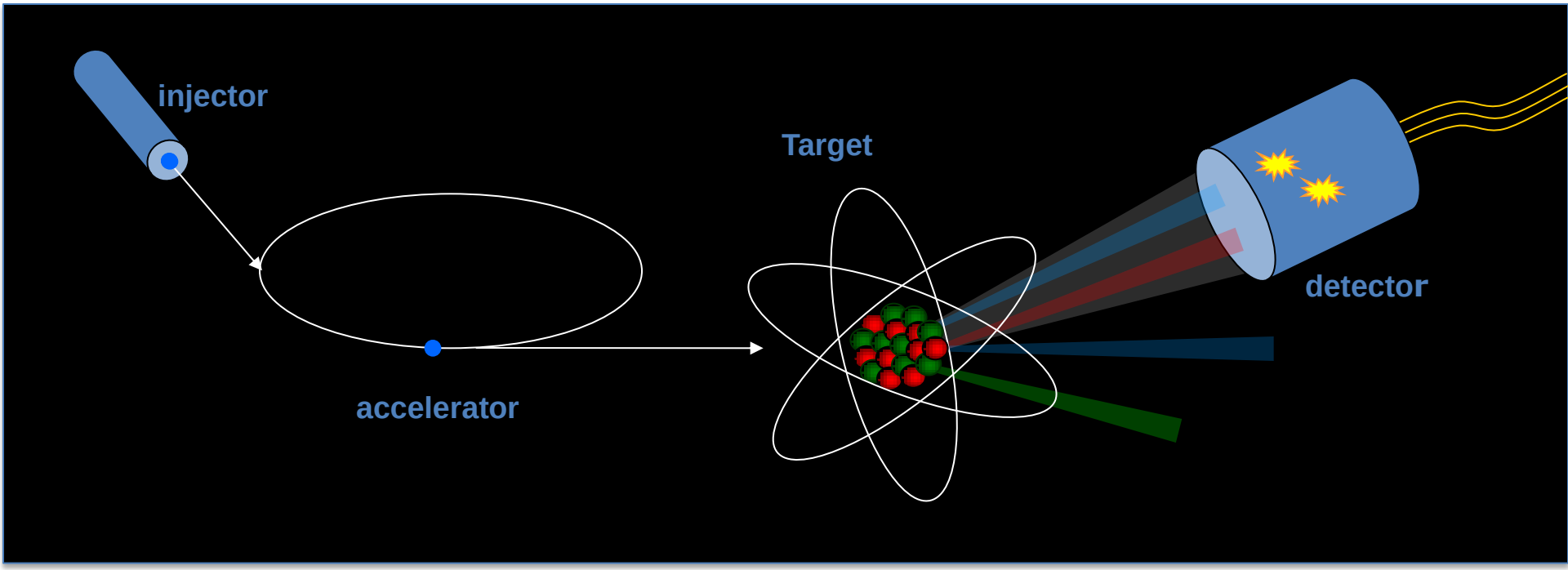
This is my world.
This is what I'll
be talking about.

Jefferson Lab (Newport News, VA, USA)



- 12 GeV polarized electron beam
- Superconducting
- Fixed target
- Continuous
- Four experimental Halls
- The accelerator itself is named CEBAF (Continuous Electron Beam Accelerator Facility)

Electron Scattering



The **C**EBAF **L**arge **A**cceptance **S**pectrometer (CLAS) in Hall B

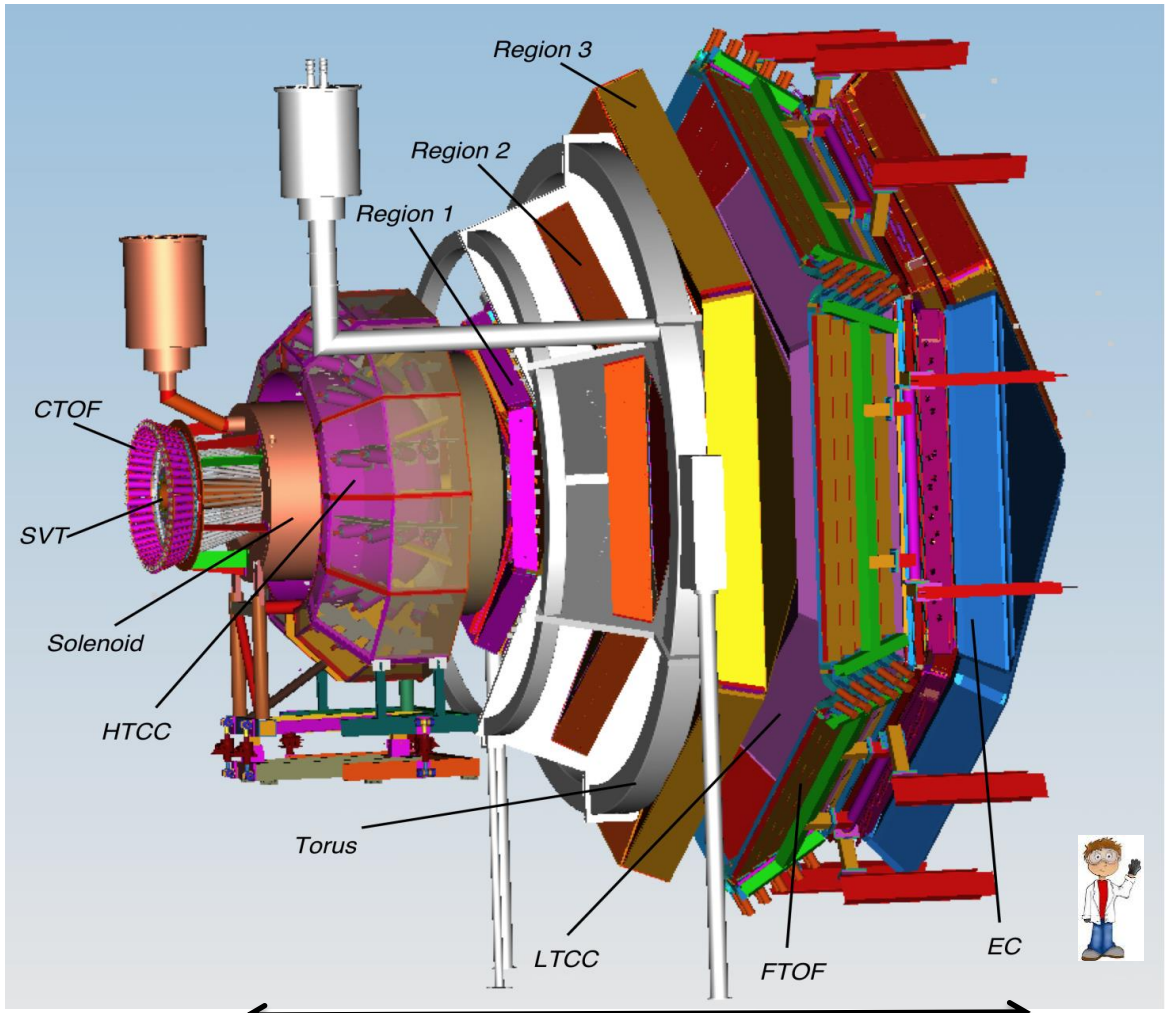
Baseline equipments

Forward Detector (FD)

- TORUS magnet (6 coils)
- HT Cherenkov Counter
- Drift chamber system
- LT Cherenkov Counter
- Forward ToF System
- Pre-shower calorimeter
- E.M. calorimeter

Central Detector (CD)

- SOLENOID magnet
- Barrel Silicon Tracker
- Central Time-of-Flight



8m

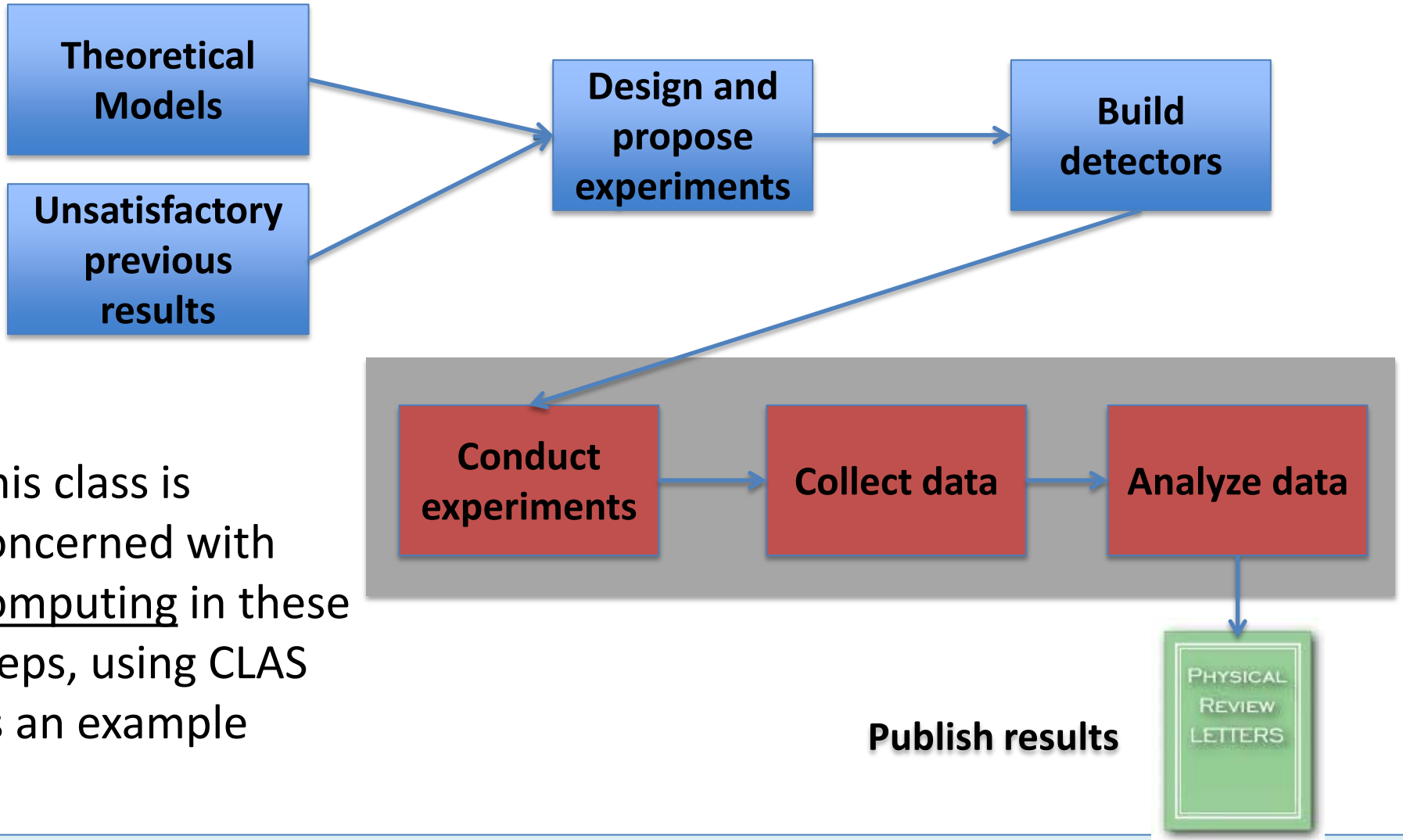
Jefferson Lab Computing

- High Performance Computing (HPC) for LQCD, ~11K cores, ~1K GPUs, and ~100 Xeon Phi cards
- Batch Computing for Experimental Physics (the "farm"), ~2K cores
- Multiple Disk Systems (online storage), ~1.3 Petabytes (1 Petabyte = 10^{15} bytes)
- The Tape Library for offline storage, >10 Petabytes

Computer Languages Used for CLAS

- High-level OOP languages, C++ and JAVA
- High-level scripting languages, Python and Jython (etc.)
- FORTRAN for legacy code (not for new development)
- Lower-level C and C-like languages for embedded systems
- Language enhancements for massively parallel applications (GPUs, Xeon-Phi, FPGA, etc.)

The (simplified) Physics Process



This class is concerned with <u>computing</u> in these steps, using CLAS as an example

Computing is *everywhere* in the process

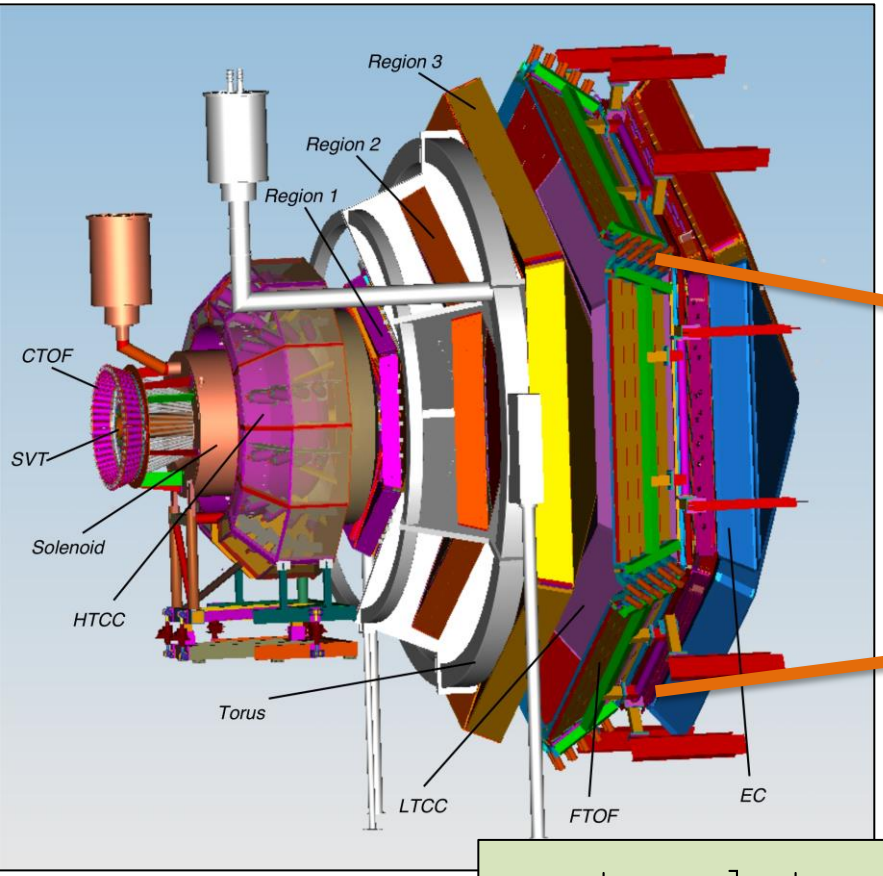
Online Computing

- Conducting the experiment
  - Trigger, Data Acquisition (DAQ), Data Archival
- Visualizing the data
- Simulating the experiment
  - Rates, acceptance
- Calibration (setting operating parameters)
- Reconstructing the data
  - Pattern recognition, nonlinear fitting
- Analyzing the data
  - Connecting with theory & models

Offline Computing

These are not independent. For example, *analyzing* the data is dependent on the ability to *simulate* the experiment

Data Acquisition (on-line)



Particles produce signals

electronics

signals

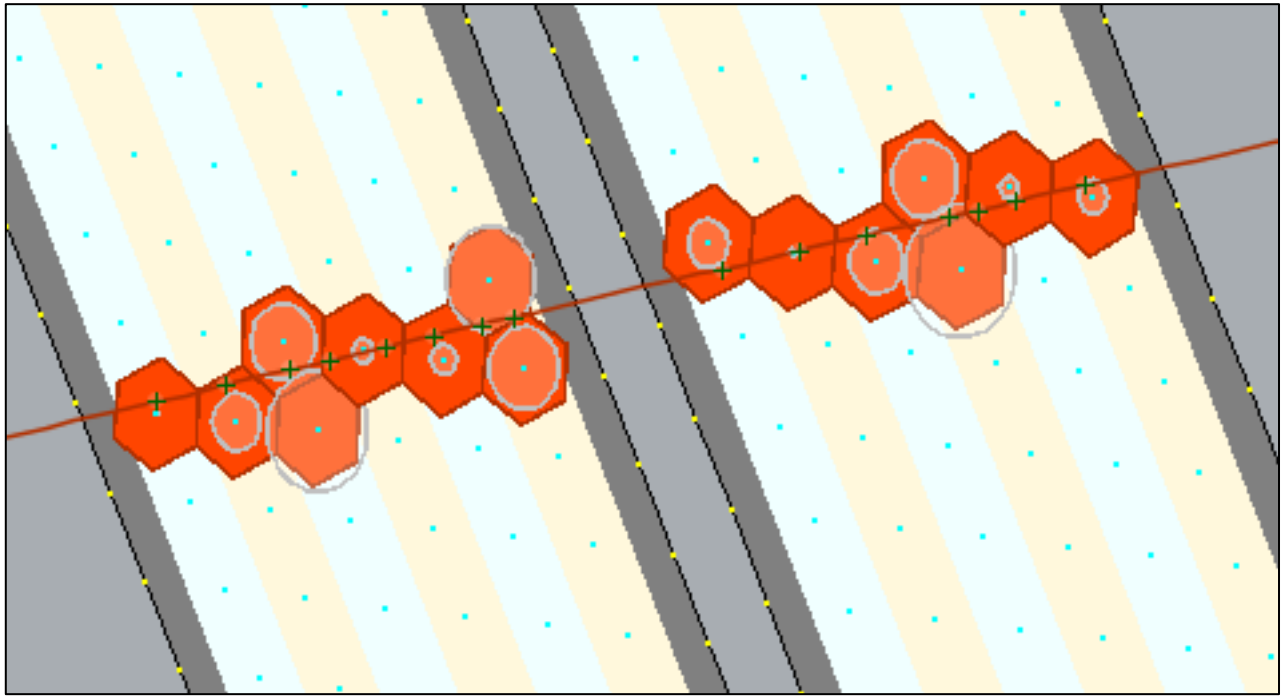
Readout

Raw Event

```
crate slot channel signal
crate slot channel signal
crate slot channel signal
    …
```

Particle deposits energy → *Signal*

- Example: **Drift Chambers**—detect charged particles by the ionization the particle creates when passing through a gas.
- The ionization drifts to *sense wires*, and the time is recorded (TDC) and (later) translated to a distance of closest approach (DOCA)

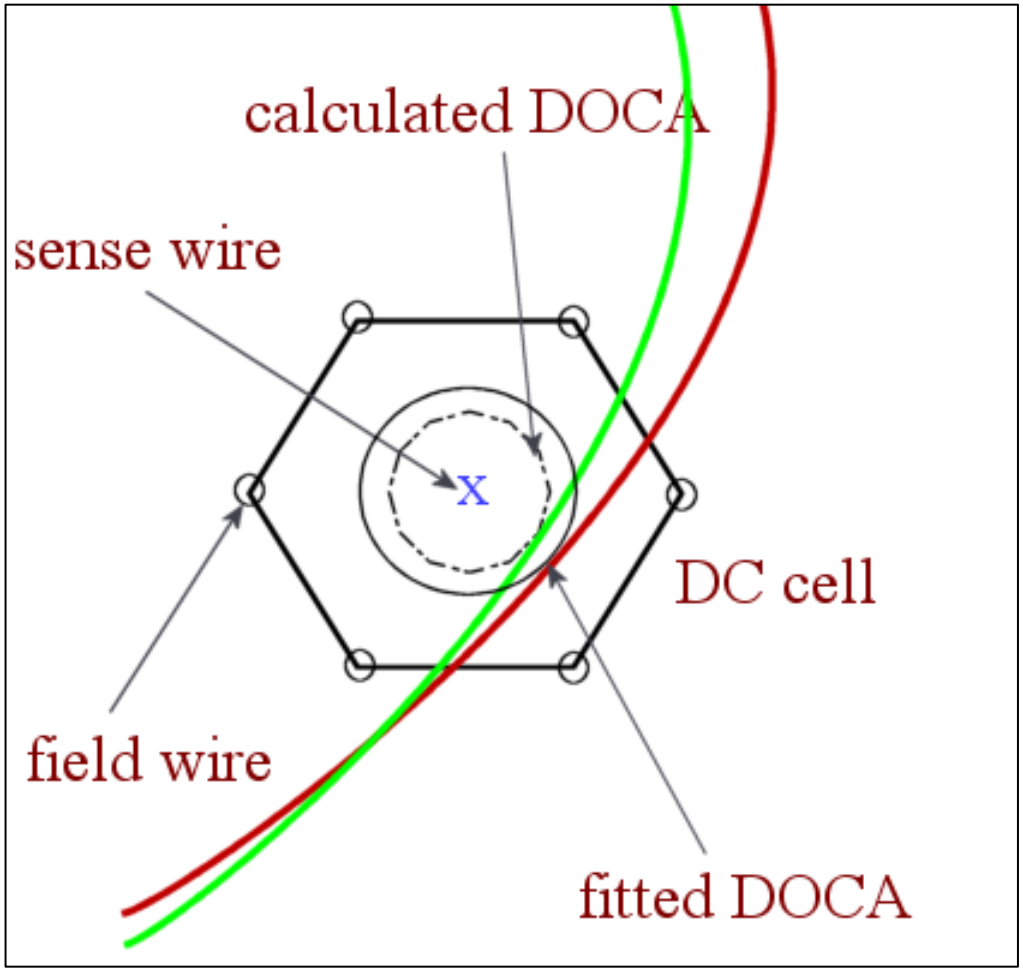


The **little dots** are wires perpendicular to the page

The **line** is the path of a charged particle

The circles represent the DOCAs

Drift Chamber (DC) cells



Field wires are arranged about each sense wire forming hexagonal **cells**.

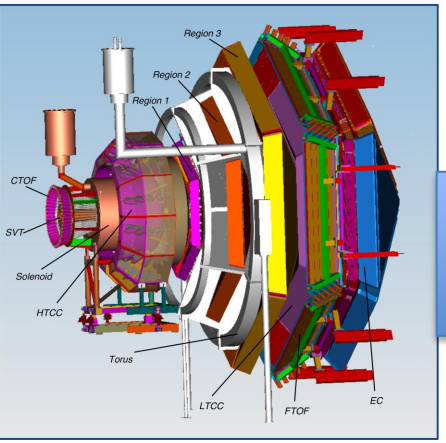
The electric field produced by the field wires causes the ionization to drift to the sense wire

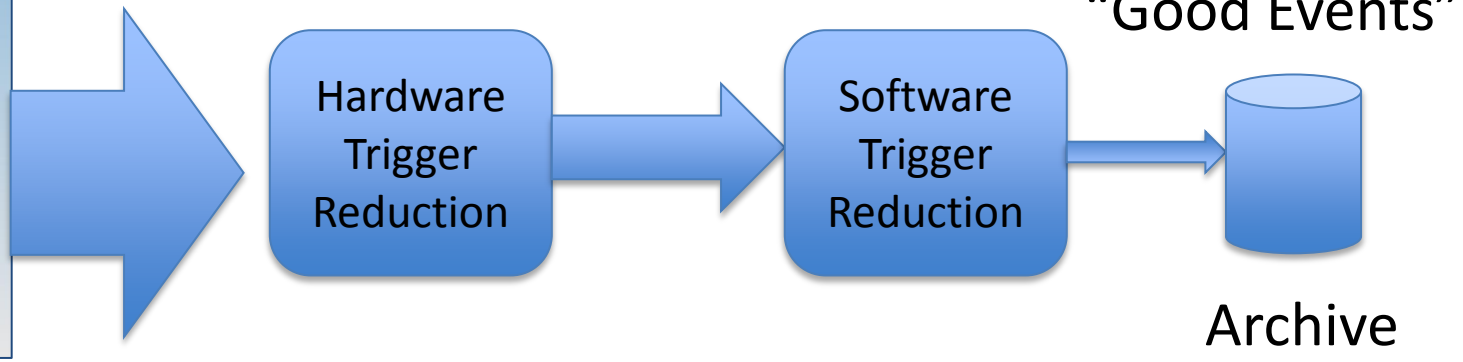
Event Building and Translation (on-line)

- The "raw event" (*hundreds* or *thousands* of [crate, slot, channel, signal] electronic addresses) is assembled from multiple crates of data (not as easy as it sounds!)
- The event is *translated* into more useful detector-based addresses (e.g., [Drift Chamber 3, wire 27, signal].)
- Events are archived in this translated format.
- This happens in real-time (i.e., as the experiment is running.)
- The software must be blazing fast (to keep up), and is written in lower level computer languages like *C*.

Trigger

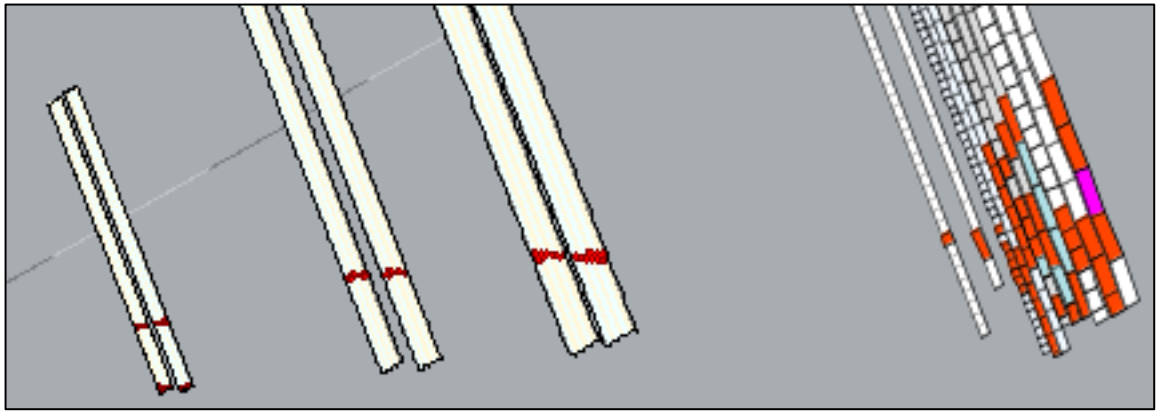
- A *trigger* is a *filter* that reduces the raw rate to something manageable by only passing "good events" that meet experiment-specific criteria.
- The first-pass triggers are embedded in fast on-line electronics and processors ("hardware trigger")
- Further reduction through a slower *software* trigger.
- Careful: be <u>absolutely sure</u> nothing interesting gets tossed out!





Triggers

- Programmable and experiment dependent, but to get a rough idea
  - (FAST) Hardware trigger is something like*: located in the same general vicinity should be hits in the DC, TOF, and PCAL/EC*
  - (SLOW) Software trigger is something like: *the hits in the DC should form segments that roughly point to the TOF hits*



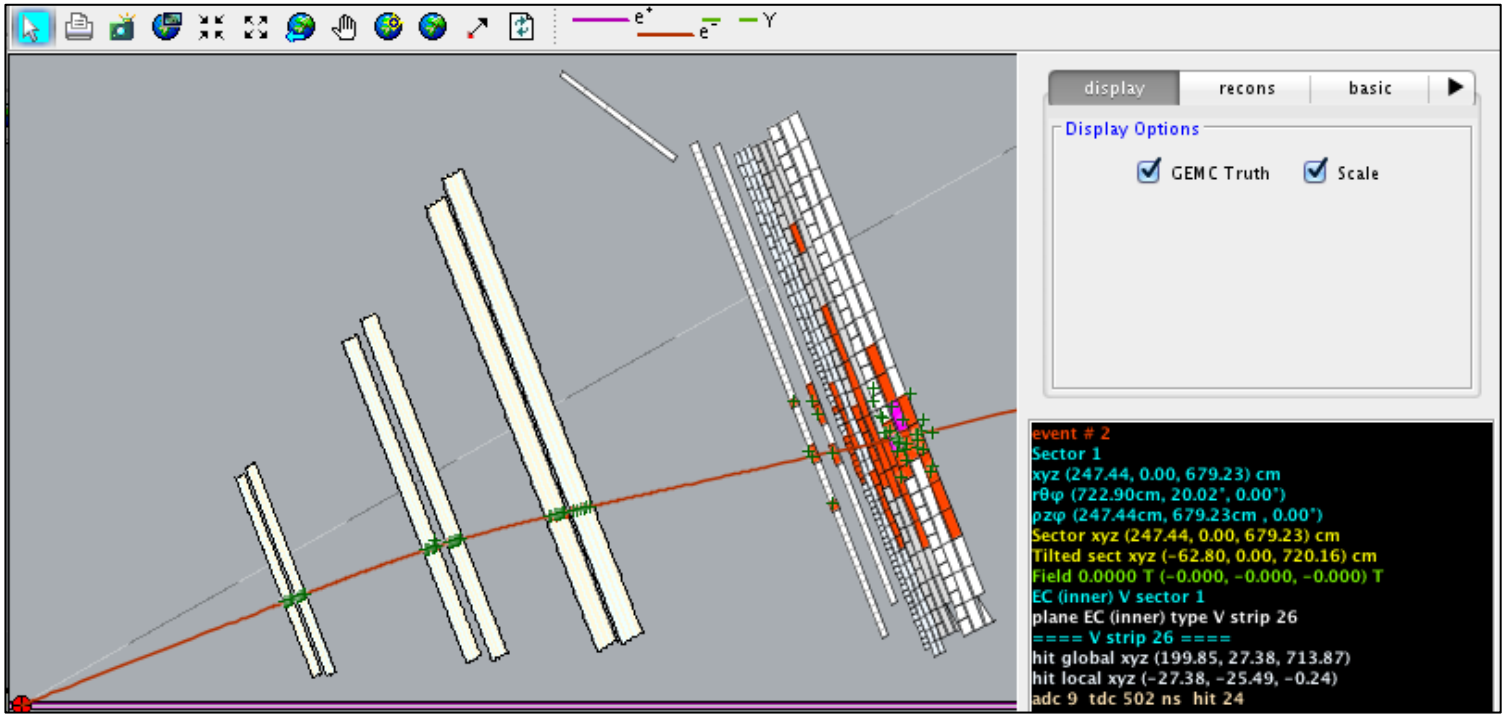
CLAS (12 GeV) Data Rates

- 20 kHz “good” event rate
- 5-10 kB event size
- ~100 MB/s data rate (comparable to LHC experiments)

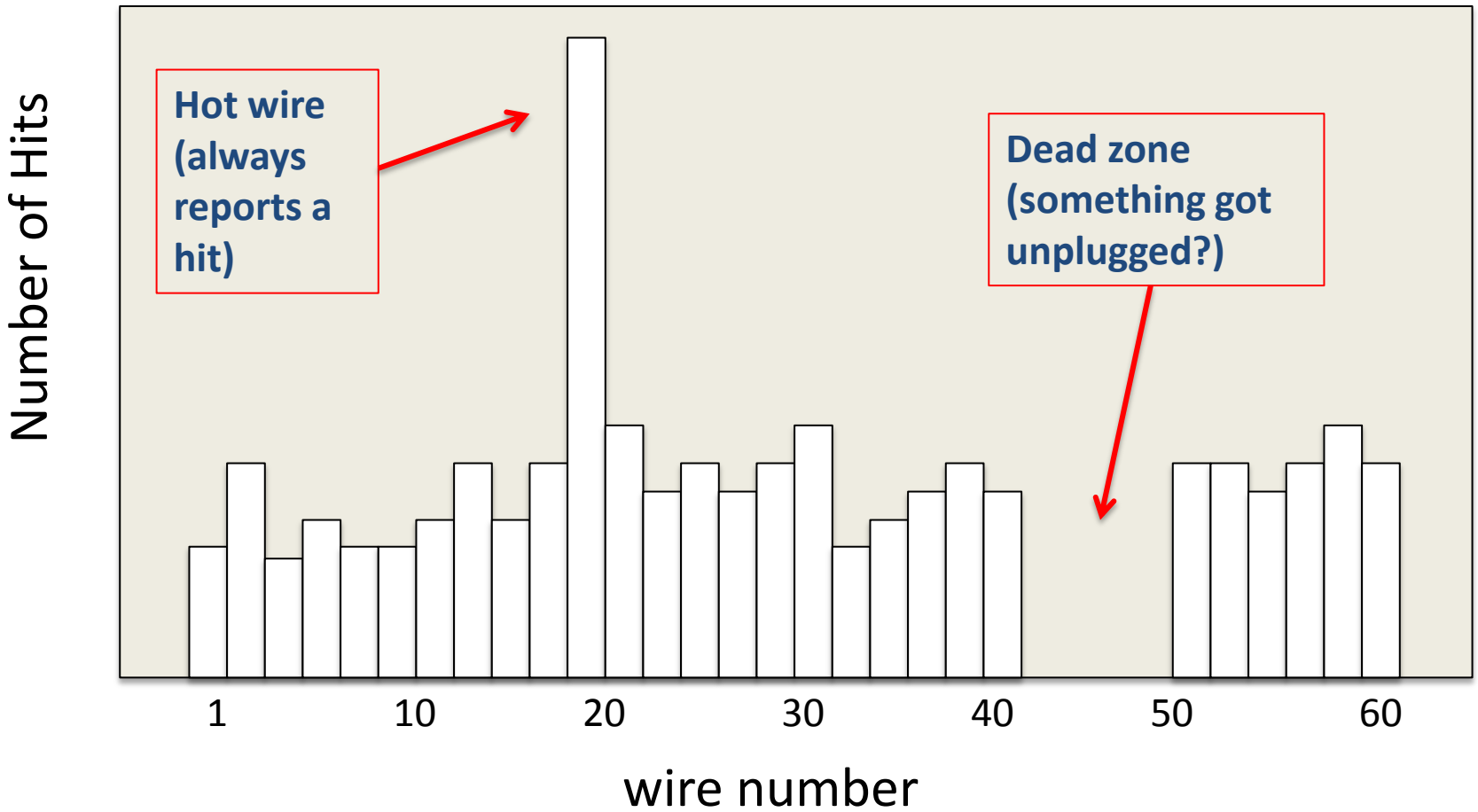
- To keep up, you have to be able to *analyze* data as fast as you *take* it! This is not easy!

Visualization (online and offline)

- Event Display (And other monitors)
  - Provide live electronic diagnostics
  - Help debug reconstruction and analysis
  - Written in a high level language (e.g., JAVA)

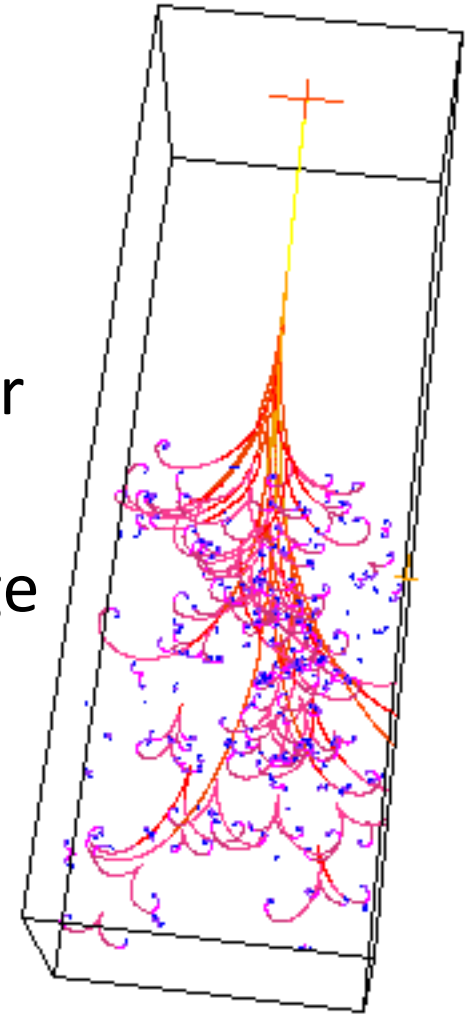


Monitoring by real-time Histograms



Simulation

- Experiments must be simulated to an astonishing degree of accuracy
- The standard tool is *GEANT4* from CERN
- Developers (using C++) describe the detector down to the nuts and bolts (well, almost)
- GEANT then accurately simulates the passage of particles through the detector (including all the things particles do, like have nuclear reactions or radiate)
- This includes the energy deposited by the particles on the various components which become electronic signals



Simulation is Crucial

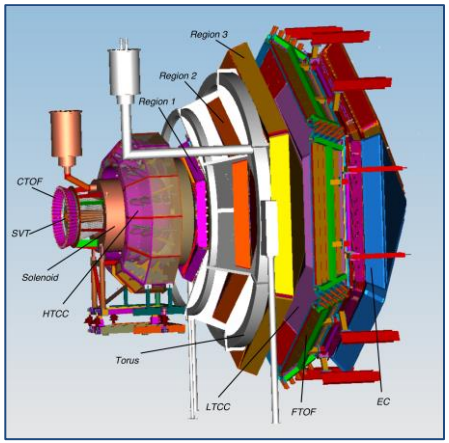
- Tells us whether an experimental proposal is viable—i.e., is the effect we want to measure in fact, measurable?

- It *teaches* us about our detector, especially its *acceptance*. Acceptance is the volume of phase space (momentum, positions) that the detector can "see".
  - The detector has "blind spots" for reasons as simple as its structural framework.

- It is critical information needed to correct analysis. *Simulation is as important <u>after</u> the experiment is done collecting data as it was before.*

Acceptance

What physics (nature) produces



CLAS doesn't "accept" everything

Archive

Archive

Reconstruction (of what we "saw")

Account for "blind spots"

Acceptance Correction

Final estimation of nature

GEANT 4

- A library that developers link with their C++ application code
- Physics “generators” are used to simulate the interactions of a proposed experiment
- GEANT “swims” the final state particles through the detector (which creates more particles)
- The application code must define the detector geometry and materials and how energy deposited is “digitized” into signals
- In CLAS, simulation of a single event can range from ~ms to ~minutes depending on the complexity of the physics that is “turned on.”

Different Simulations for Different Purposes

- For testing *basic* track finding (more about that later) we do not need all the physics. In GEANT, you can "turn off" physics → <u>fast simulation</u>
- For designing the detector, mapping acceptance, modeling the experiment, or testing *full* reconstruction we want realistic simulations, so "turn on" physics (nuclear interactions, multiple scattering, etc.) → <u>slow simulation</u>

Online software summary

- The end-result of the *online* software are data files of events in translated (detector, component, signal) format.
- The detectors are big and complicated and so the events are big.
- The event rates are high, so we are talking many files and much data (Petabytes).
- This is where the *offline* software takes over.

Offline software tasks: 1) Reconstruction

- **Reconstruction**: convert the data files created by the online software (detector, component, signal) to "four vectors" which are essentially the identity, starting position, and momentum of the final state particles
- This also reduces the size of the data

detector, component, signal
detector, component, signal
detector, component, signal
…

reconstruction →

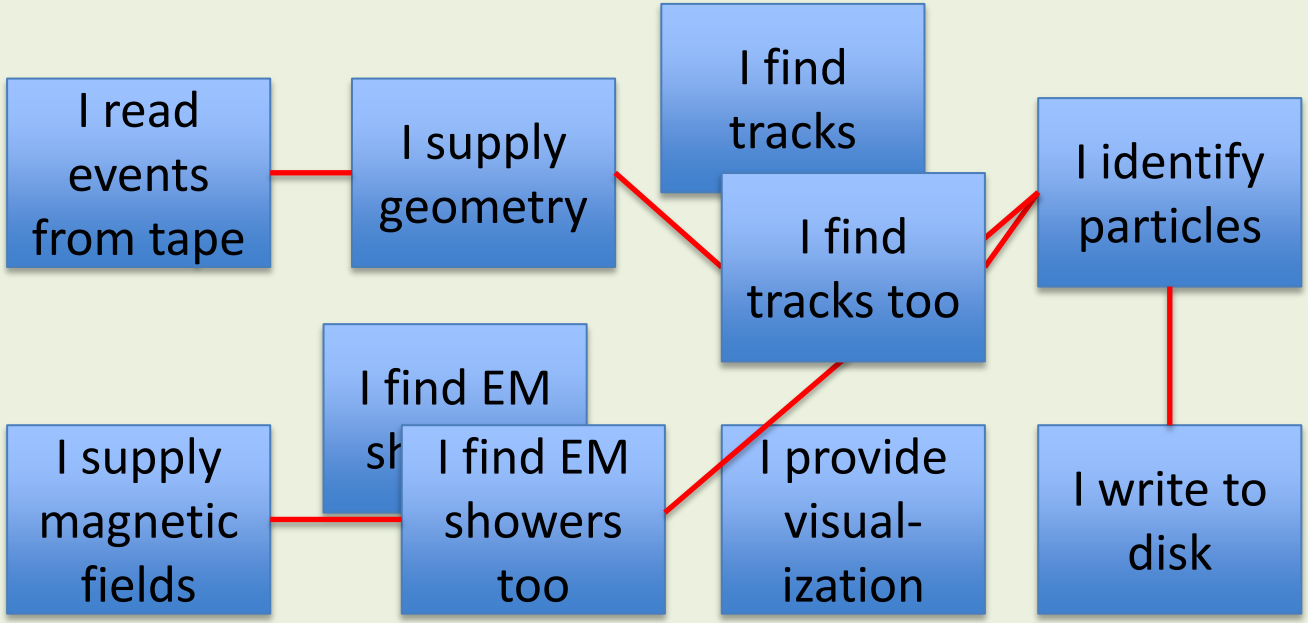
1. e-, vertex = (x_o, y_o, z_o)
   momentum = (p_x, p_y, p_z)
2. p, vertex = (x_o, y_o, z_o)
   momentum = (p_x, p_y, p_z)
3. π-, vertex = (x_o, y_o, z_o)
   momentum = (p_x, p_y, p_z)

First Rule of Reconstruction

- If you take 100 Petabytes of data per year, then you must be able to *reconstruct* 100 Petabytes per year
- *The reconstruction cannot be slower than the data acquisition, or data will begin to "pile up."*

CLAS offline reconstruction software *framework*

- Service oriented architecture (SOA)
  - Application is built from linking simple services

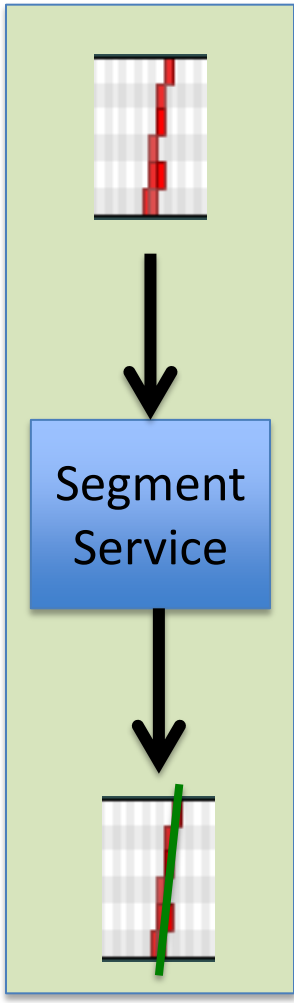


"Small" specialized independent software *services* written in C++, Java or Python

Reconstruction: an application built by linking services

SOA: *Loose Coupling*

- A service is identified by a simple contract, for example:
  - If you give me the drift chamber hits
  - I promise to give you the segments
- A service knows nothing about the application that uses it! (Loose coupling)
- Competing services are *interchangeable*
- The test of a good service oriented architecture is that it gets used in ways the designers did not imagine (e.g., Amazon)



Reconstruction Roughly Speaking

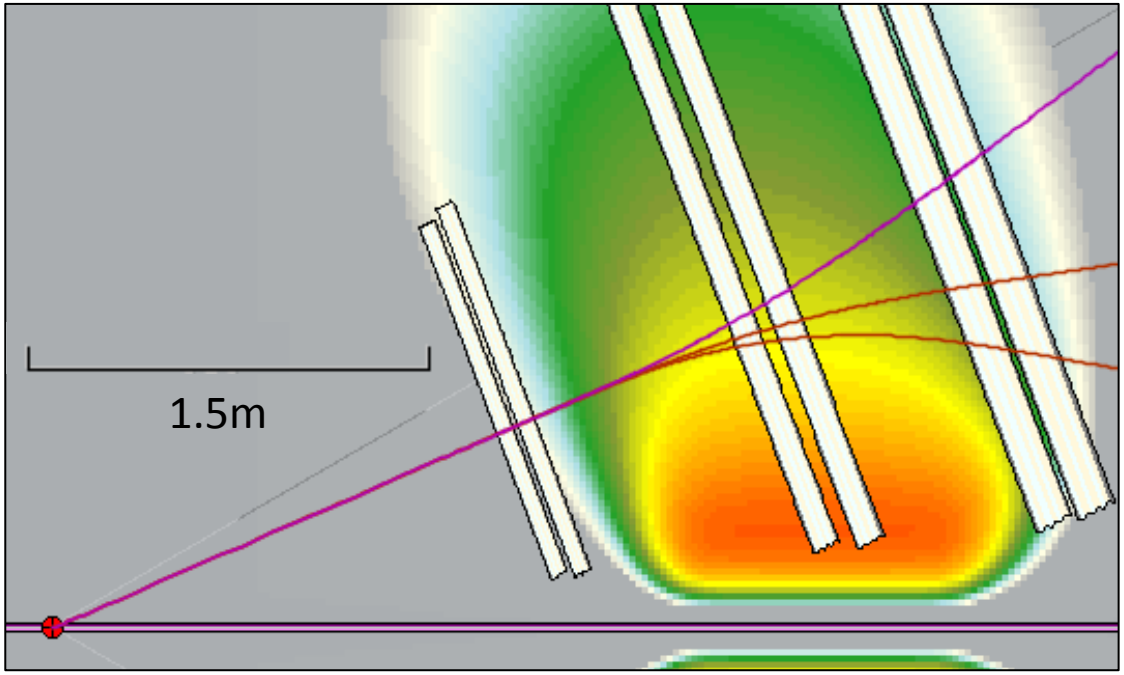
- One set of detectors (primarily drift chambers) determines the *trajectory* or *track* of the charged particles—which, because this happens in a magnetic field, then gives their *momentum* ($\rho \sim p/qB$)
- Another set of detectors determines the *energy* of the particles
- The particle is identified by its mass, found via

$$E^2 = p^2c^2 + m^2c^4$$

- More information comes from conservation of momentum and energy ("missing mass")

Finding Trajectories

Made possible by magnetic fields! (In this picture, the field is perpendicular to the page)



QUIZ!

Shown are the trajectories of a 1 GeV electron, a 2 GeV electron, and a 2 GeV positron.

Match the trajectory to the particle.

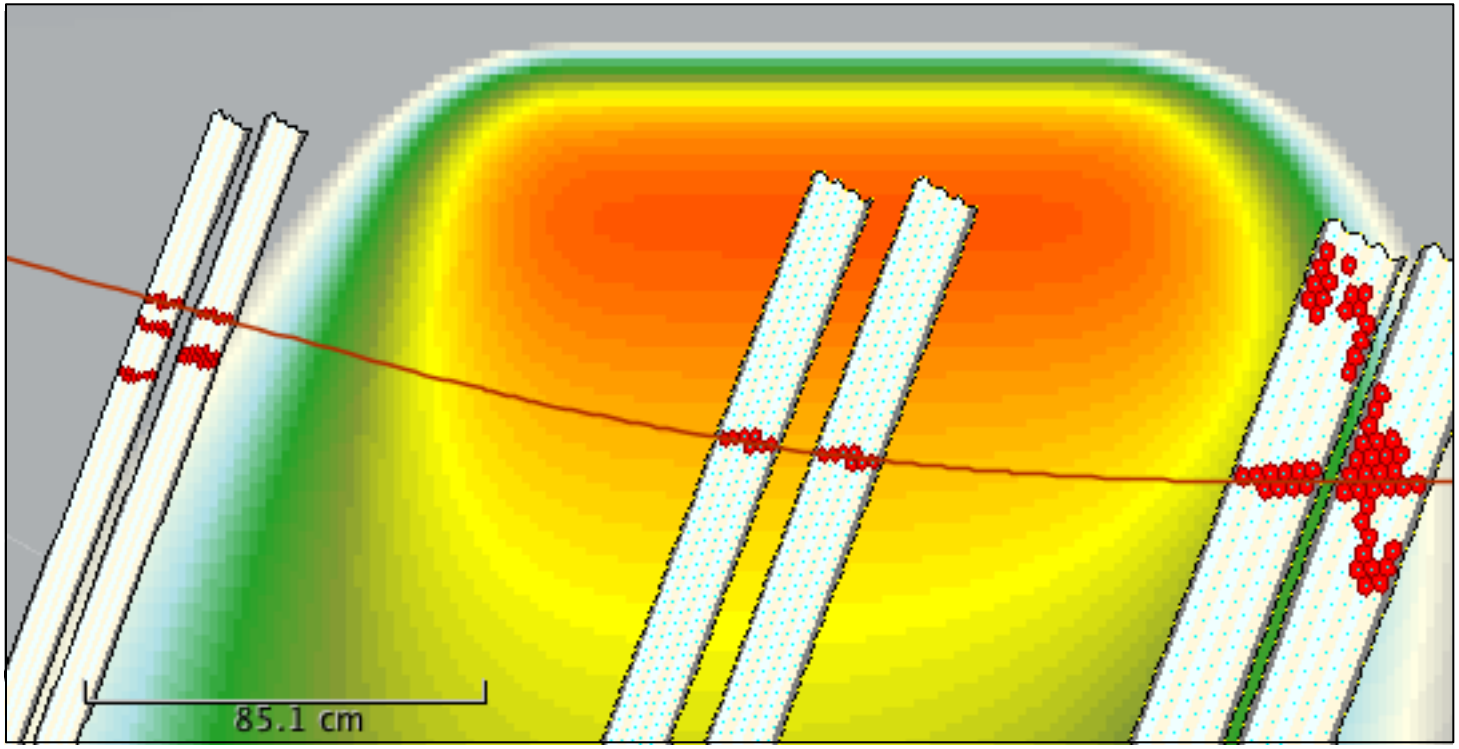
Bonus question: is the field in or out of the page?

Offline Software Tasks: 2) Physics Analysis

- Recall that the result of the *Reconstruction* step is the reduction of the data to collections of four-vectors (particles and their locations and momenta) for each event.
- *Physics Analysis* then converts those four-vectors into physics quantities, such as cross sections, structure functions, distributions, sum-rules, branching ratios, etc.
- This is the most time-consuming part of the process and the most "physics intense" (Time ~years)**. Ph.D.'s come here!**
- We will not discuss physics analysis. (No generic approach.)

Back to Reconstruction.

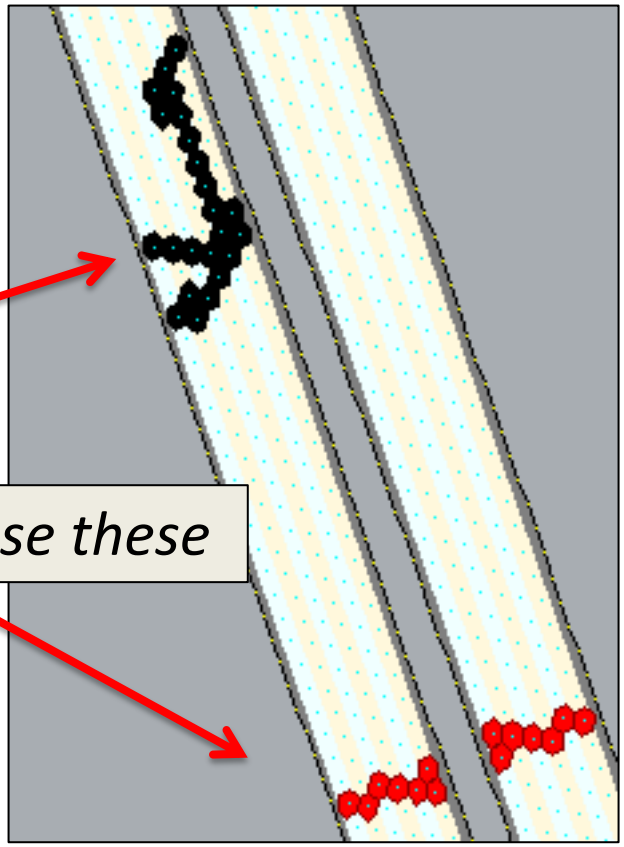
- The first step: pattern recognition.
- Physics is never “clean.” Can you find the trajectory?



Extra hits (“noise”) can come from 1) random electronic noise and/or 2) secondary particles

Finding Tracks

Ignore these (how do we know?)

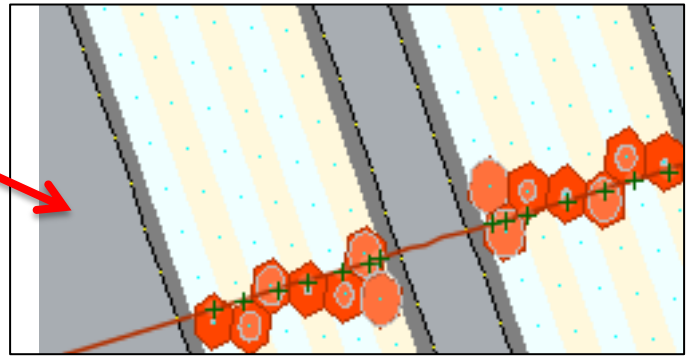


Use these

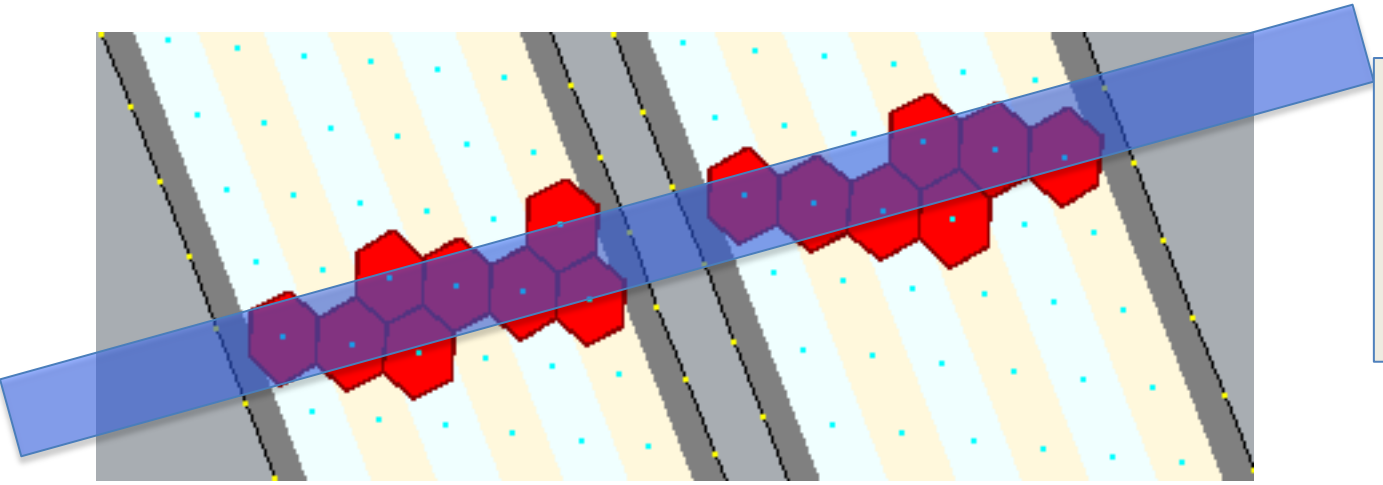
- Step 1: Use only the pattern of drift chamber hits (*hit-based tracking*) for an initial (fast) estimate

- Step 2: Use *time* information and non-linear fitting (time-based tracking) to get a precise estimate

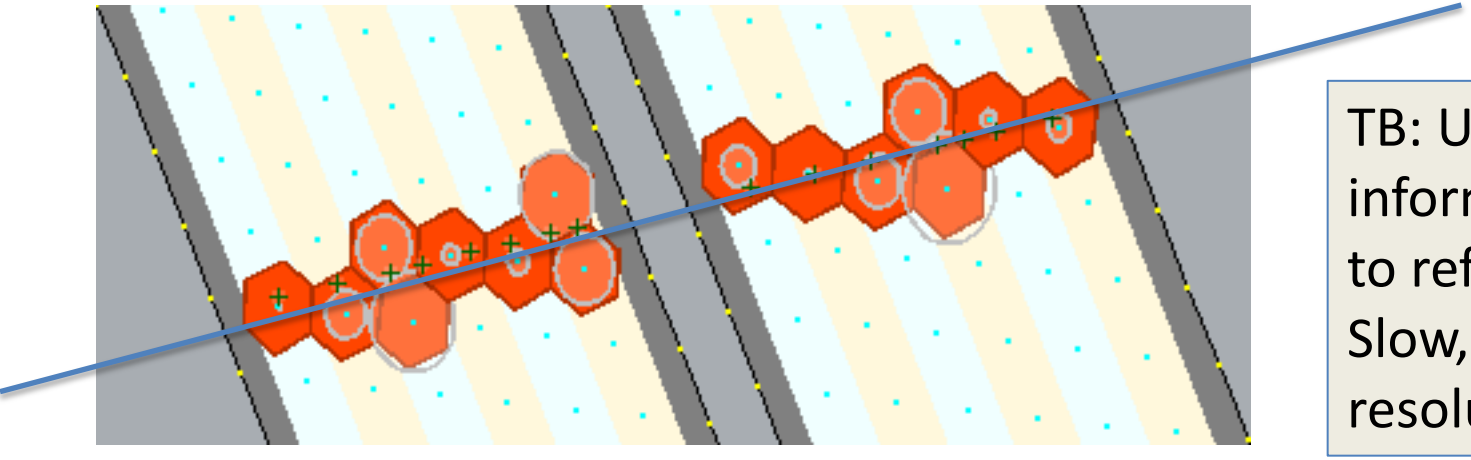
Fit using time



Hit Based (HB) vs. Time Based (TB)



HB: Just use the hit-cell pattern. Fast, but low resolution. First step.



TB: Use time information (DOCA) to refine the track. Slow, but good resolution.

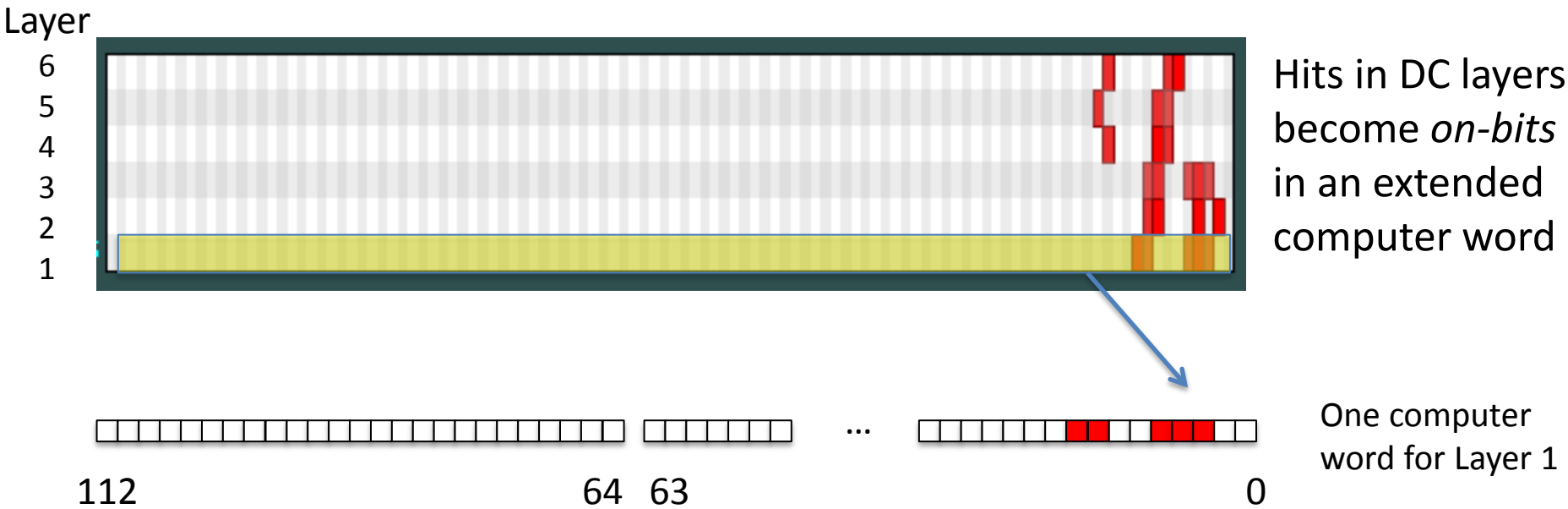
One algorithm

To give you a flavor, lets look at an easy-to-understand and very fast algorithm for finding track candidates

Algorithm

- Uses bitwise algorithm on extended words. (Each DC layer represented by composite 64 bit integers)
- Effectively turns a single CPU core into 64 parallel processors—*this algorithm is very fast.*
- Programmable
  - Handles any number of missing layers (described later)
  - No special layer
  - Programmable for different curvatures (momenta)
  - Within parameter coverage
    - Some false negatives (noise left behind)
    - No false positives (good data identified as noise)

Bitwise? What? Why?



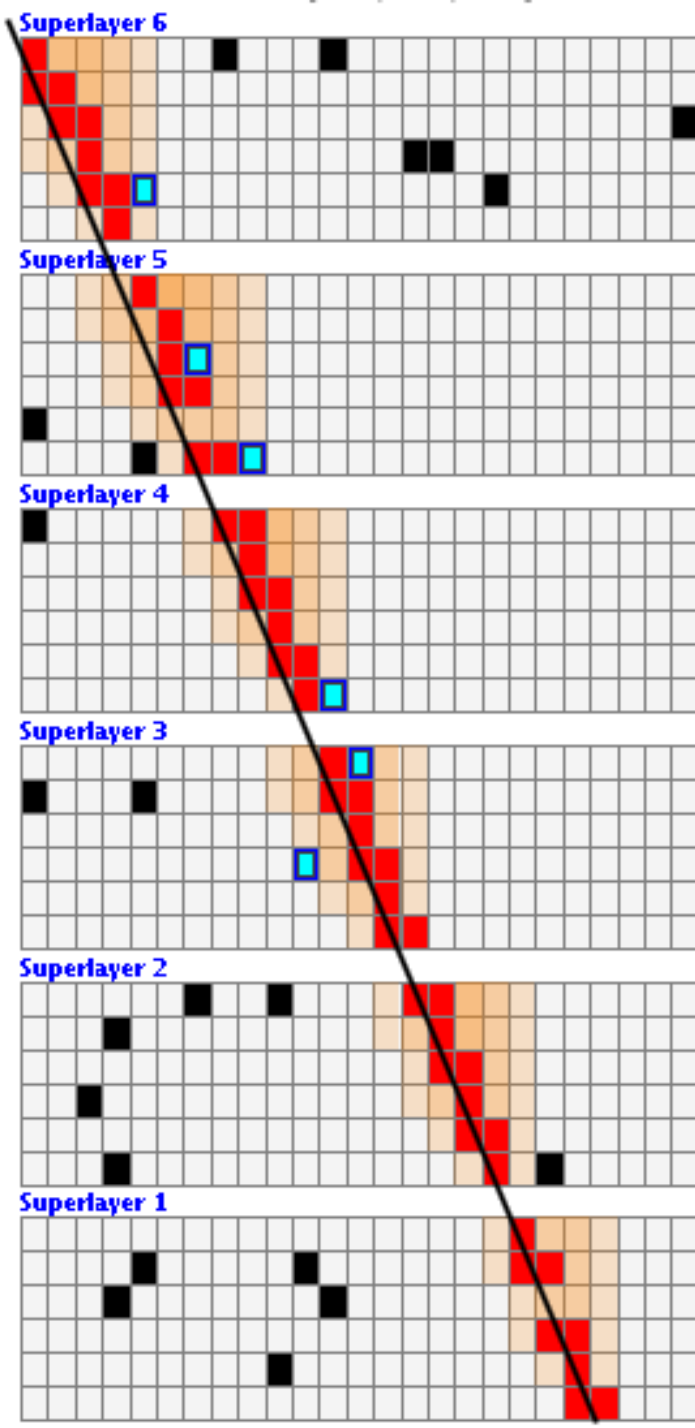
IDEA: If we can find tracks (or pieces of tracks, called segments) using standard logical operations (like AND, OR, etc.) on these computer words, I avoid loops over wires and effectively have a massively parallel processor! This should be very fast. (And it is!)

This is a toy detector (but similar to CLAS) with 36 layers (1 layer = 112 wires) of drift chambers arranged into six “superlayers”.

The red squares are determined to be “good”. The black are marked as *noise*. The blue are noise that the algorithm will not discard (they sneak through.)

MISSING LAYERS. Note Some layers have no hits where you’d expect—this may be due to any number of reasons (real physics is always messy.) The algorithm must allow for a certain amount of missing data.

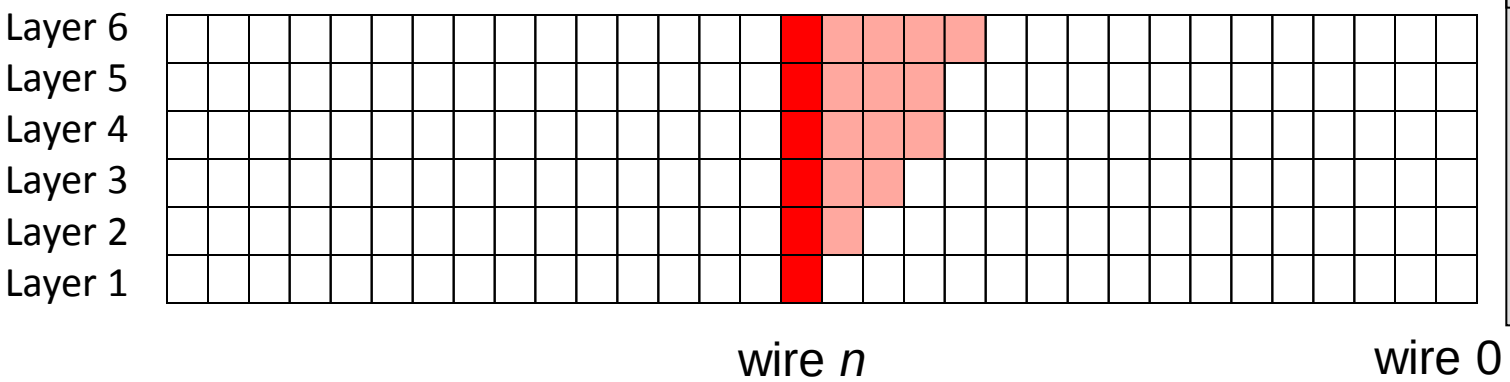
The LESSON: what is easy for us is not so easy for a computer!



Method

- Created arbitrary size words (e.g., 112 bits) (There are many libraries for doing this; we wrote our own.)
- Use one bit per wire
- Developed common (and one uncommon) bitwise operators on these extended words

Layer Shift Parameters



Layer	*Shift*
6	4
5	3
4	3
3	2
2	1
1	0

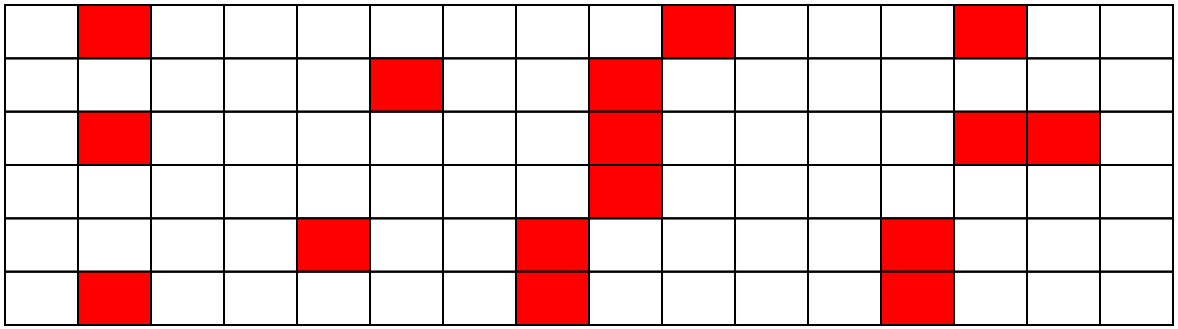
What makes a track (segment) candidate?

- Ignoring (for now) missing layers, a right bending candidate has a hit at wire *n* and at least one hit in a red cell in all other layers.
- There are left shifts as well, and they need not be symmetric.

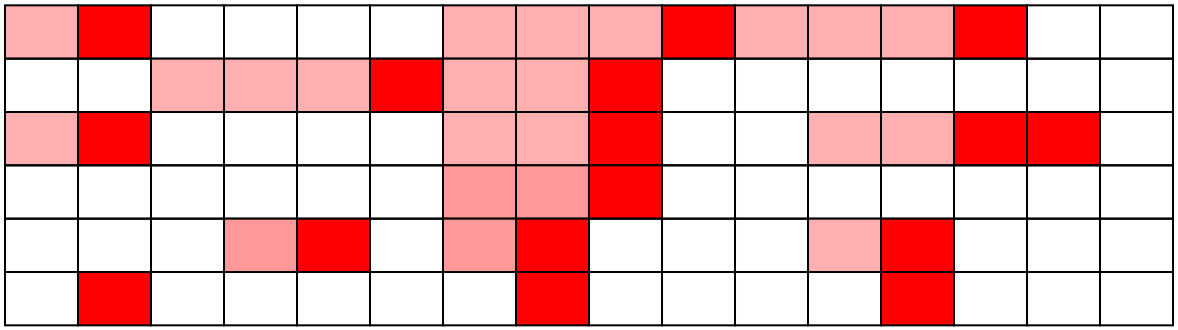
e.g., Right Benders

- **Allow two missing layers**
- **Layer Shifts: 0,1,2,2,3,3**

Data



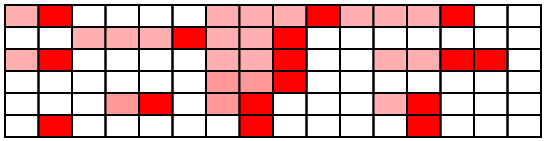
Step 0: "Bleed" the data *left* based on the layer shifts



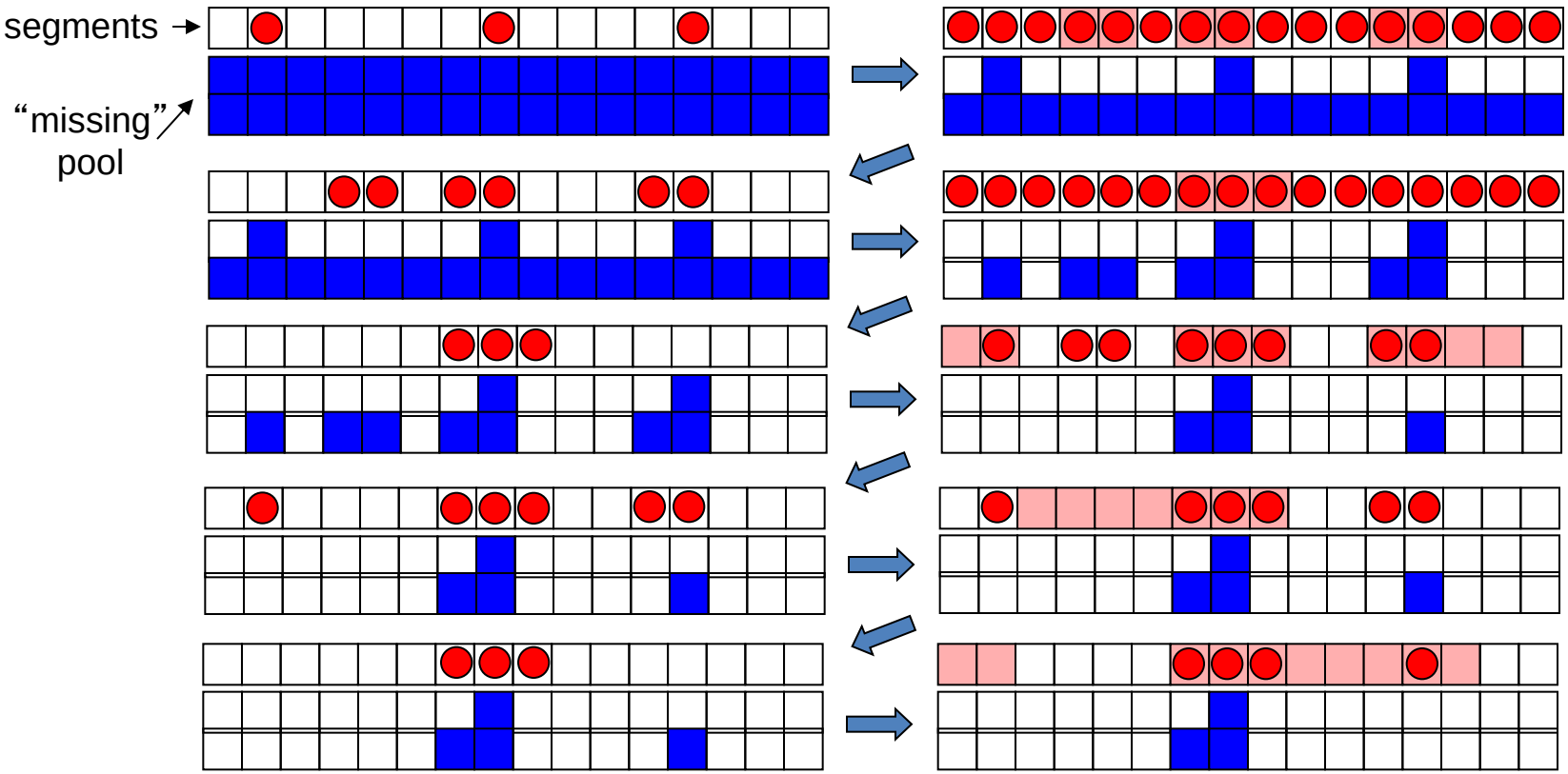
Look for complete columns!

Step 1: segments = segments .AND. Data from current layer (init as layer 1)

data

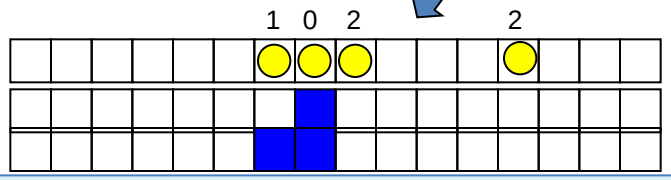


Step 2: fill in missing segments using "missing" pool—until depleted.



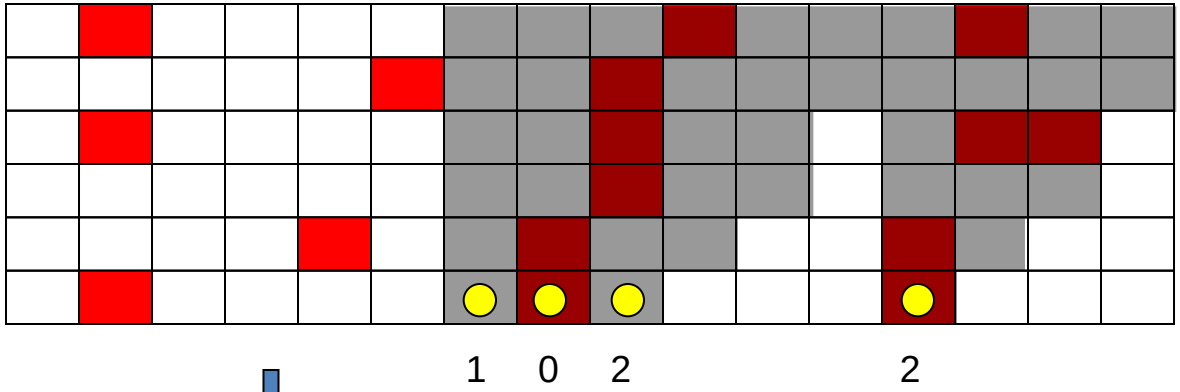
shaded boxes from layer 2 data

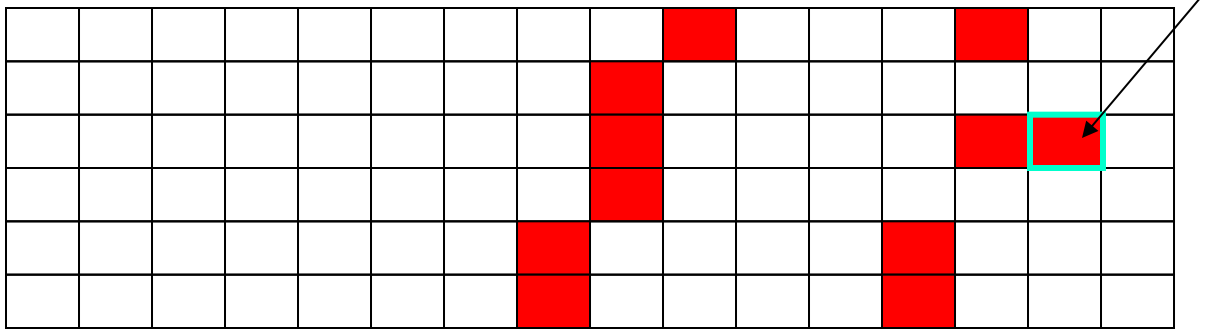
Four candidates. (Two with two missing layers, one with one, one with none)



Final Step: Noise Removal

1. Create mask based on segments found and level shifts.
2. .AND. Mask with original data





Summary

- Software is used throughout the experiment
- It roughly breaks down into
  - *online* or real-time which is concerned with getting good events to storage, as fast as possible
  - offline, which is concerned with reconstructing and analyzing the data
- If you like software—physics software is some of the most challenging and leading edge development around—it is extremely rewarding!

Questions?