Some thoughts on analysis tools

Jan Bernauer

Software & Computing round table, June 1, 2021

RBRC
RIKEN BNL Research Center

Stony Brook
University

Where I come from

- Socialized in small to mid-sized collaborations
- Started in A1 collab in Mainz: Custom analysis package not based on ROOT!

Where I come from

- Socialized in small to mid-sized collaborations
- Started in A1 collab in Mainz: Custom analysis package not based on ROOT!
- Wrote analysis framework for OLYMPUS
  - ROOT based
  - Now also in use in DarkLight, TPEX, MUSE, TREK
  - Compiles on all linux distros I tried, Mac OS X
- Students at small experiments will be postdocs at the big experiments.
- PI of eRD23: Streaming readout for EIC

What is an event?

Physics event: Causaly linked chain of "things happening"

- This is what a theorists thinks about
- Or a Monte Carlo generator/simulation (typically)
- And what we need in the final physics analysis (we count these)

What is an event?

Physics event: Causaly linked chain of "things happening"

- This is what a theorists thinks about
- Or a Monte Carlo generator/simulation (typically)
- And what we need in the final physics analysis (we count these)

DAQ events / events on tape: coincidentally linked "things happening"

- Typically: Trigger
- In SRO: "same time bucket", "same bunch" or similar

They say she's the same, but she isn't the same

These things are not the same!

- A DAQ event can have multiple physics events + noise
- In SRO, a physics event can span multiple DAQ events

Analysis software typically works on DAQ events. Why?
Because data are stored this way!

My humble opinion

- What belongs to a physical event is an analysis decision.
  - Might actually be really high level (especially in SRO)
  - i.e. delayed decay of DM
- Analysis tools need to support this difference!
- Analysis framework should not force "eventification"

The data must flow

- Decompose data into a series of streams
  - Elements in a stream are of same/similar type
  - Time ordered
- Analysis is then a DAG (almost)
- An operation can add/replace streams

The data must flow

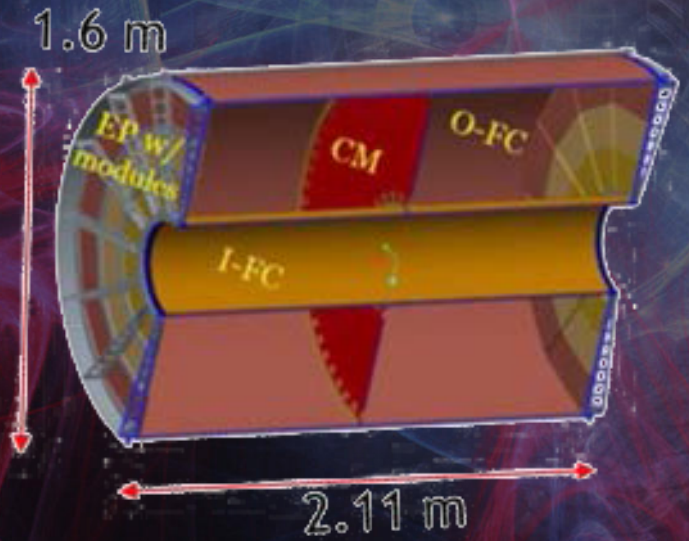
- Decompose data into a series of streams
  - Elements in a stream are of same/similar type
  - Time ordered
- Analysis is then a DAG (almost)
- An operation can add/replace streams
- "A stream of tracks"
- "A stream of physics events"

How much of your code looks like this?

```
for all events:
        for all hits:
                do_something(hit)
```

An example: TPC reconstruction

- First step: Input: TPC hits. Output: track stubs (projected axis unresolved)
- Parallel first step: Input: timing detector waveforms, Output: timing detector timestamps
- Second step: Input: track stubs, hits in timing detector. Output: full tracks



Effects on storage

- No massive event, but semi-independent streams.
  Possibly in multiple files.
- Columnar storage instead of row storage.
  (But: data is not a table)
- Comes naturally for SRO
- Only load what you need
- Only write what is new
- no DSTs.

Effects on storage

- No massive event, but semi-independent streams.
  Possibly in multiple files.
- Columnar storage instead of row storage.
  (But: data is not a table)
- Comes naturally for SRO
- Only load what you need
- Only write what is new
- no DSTs.

- No, actually: custom DSTs at any level!

Effects on compute

- Only load what you need
- Only write what is new
- Simpler data structure: better for HPC
- More homogenous data. Better compression?
- Can aggressively cache intermediate results. Faster dev cycles.

Effects on coding

- Naturally decomposes analysis in individual, (semi-) independent tasks
- Data structures define interface.
- Can exchange parts.
- Maps trivially to responsibilities of groups.
- Contains effects of code breakage.

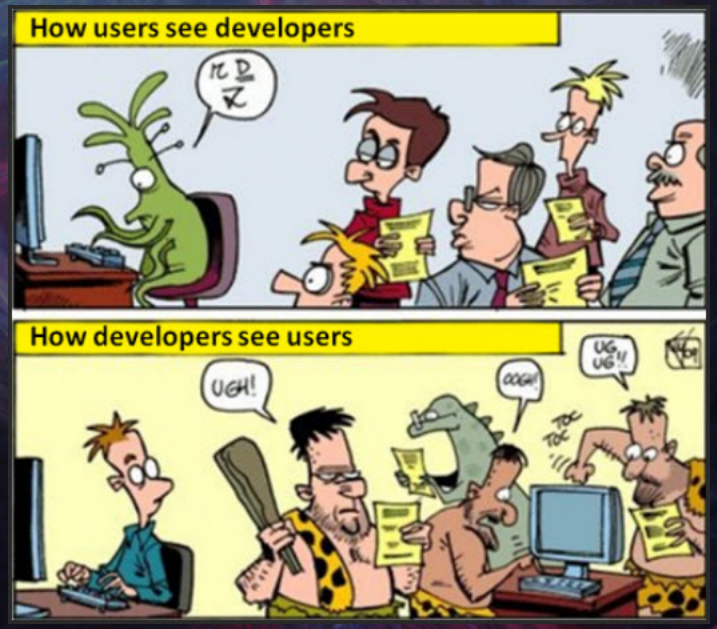
```
bernauer@wolfpack:~$ make sens
make: *** No rule to make target 'sens'.  Stop.
bernauer@wolfpack:~$
```

Data coordination

- We need tools to organize this data flow
- Have to support disk caching
- data locality awareness
- support disconnected mode

Optimize for Users or Developers?

- In a small collaboration, all users are developers.
- Everyone needs more developers!
- Make sure it is easy to become a developer!
- Do not adopt REPL.
- A web browser is a bad replacement for an IDE.



Avoid hard transitions

- python/jupyter:
  - Great for users.
  - But cannot write high perfomance algos in pure python.
  - How well does scipy fit to your data?
- singularity/docker:
  - Great to get started.
  - And then? What is my editor?
  - How do I compile a new module?
  - How do I get data in and out?

At every transition, you use possible developers. Have the pain once.

How to avoid transitions

- Set them up with a complete setup, including sources. Compile stuff.
- Avoid macros. Give them templates to start with.
- Stay in one language
  - otherwise people have to learn two
  - AND how they interface
- If C/C++, it should be compiled
  - Generations of people were ruined by the lenient CINT.

What language to use?

Modern C++

- too late for Rust (for EIC, and in general)
- Most people speak C, C++
- translates to SystemC, OpenCL etc.
- Python is great. But too slow.

Black box problem

- All documentation lies.
- And rots, fast.
- Best case: Documents what author thought, not what code does.
- People need to be able to look at lower-level code. Train them in that. It makes them developers!

How to get people to use your stuff

- If people are using something else already, and get results, you have already lost

How to get people to use your stuff

- If people are using something else already, and get results, you have already lost
- Uphill battle. You can only win if you:
  - Have (essentially) feature parity
  - Are easier/similar to use
  - Have a good selling point. (Nicer design is not a selling point, unfortunately)

Assorted comments (mostly root rant)

- Do one thing, and do it well.
  - Why does ROOT wrap every library under the sun? Is this systemd?
- Please get rid of global state.
- Think about names
  - What does TH1::clear() do?
- Don't be brain dead
  - TTimeStamp::GetTime()
- Avoid magic
  - Every magic button means less people will look under the hood.