Software Challenges in Streaming Readout

Jan C. Bernauer

Future Trends in Nuclear Physics Computing, September 2020

RBRC
RIKEN BNL Research Center

Stony Brook University

What is streaming readout

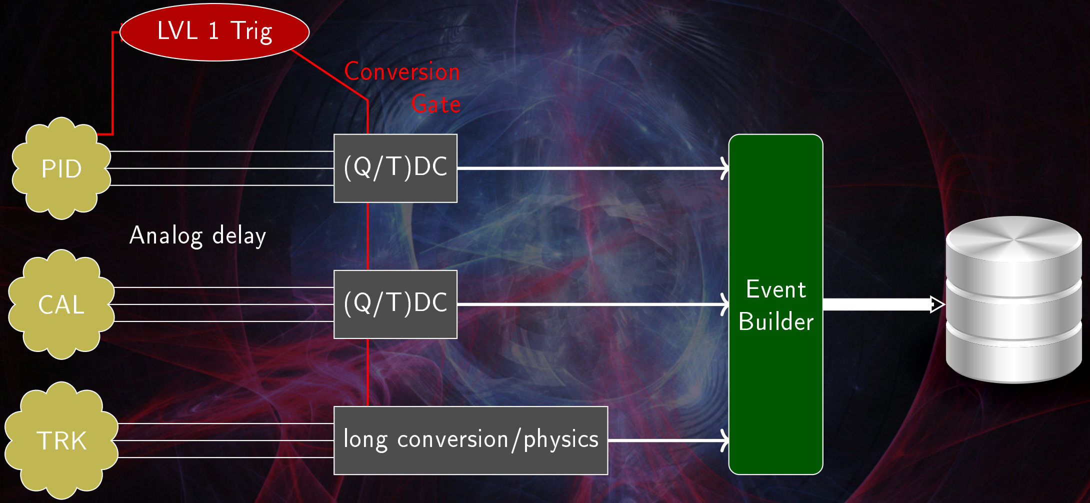
Paradigm shift from traditional, triggered readout.

Two ways to define it:

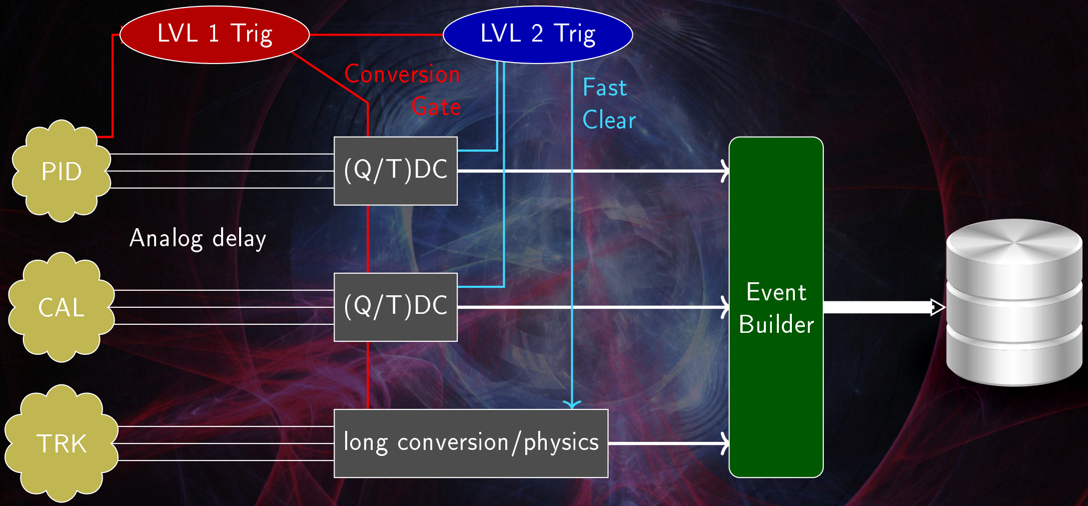
1. In an SRO system, there is no non-local, fixed-latency trigger signal that controls the DAQ
2. In an SRO system, detector information are tagged/combined by time stamps

I believe that 2 follows from 1, and that it's more exact. Everything else follows, or would be possible with a triggered readout.

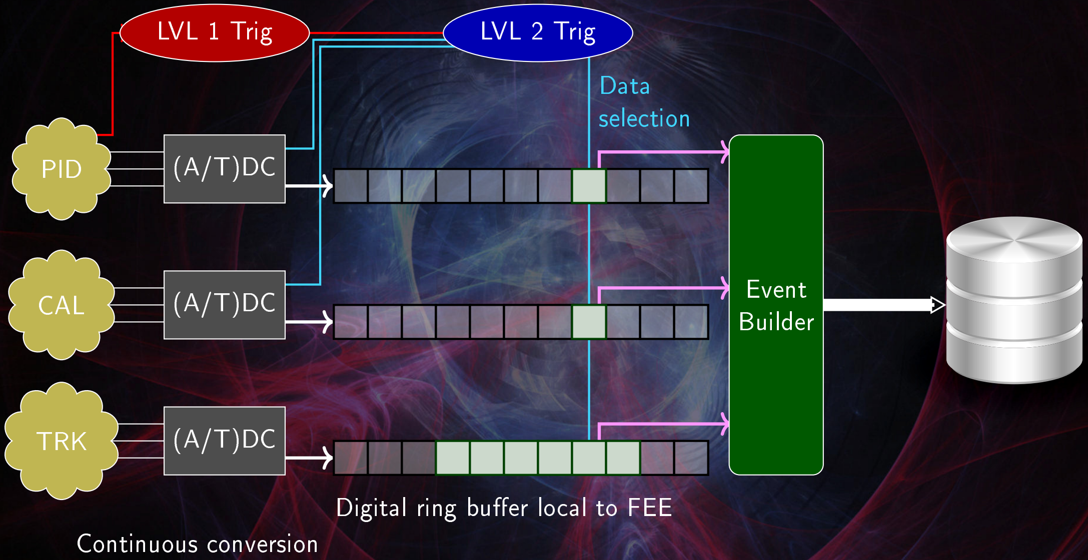
A classic readout system



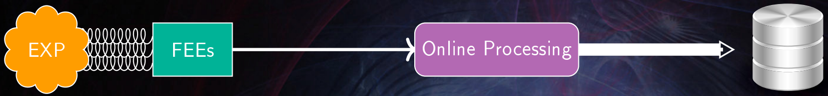
A classic readout system



The semi-old ways

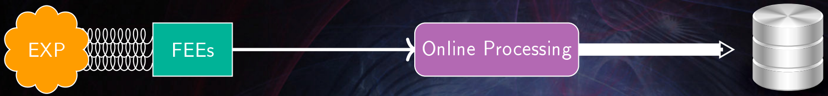


Streaming readout anatomy



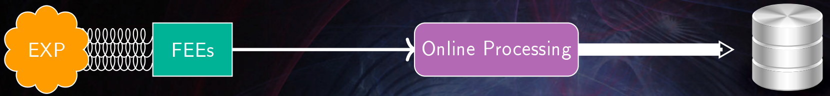
- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression

Streaming readout anatomy



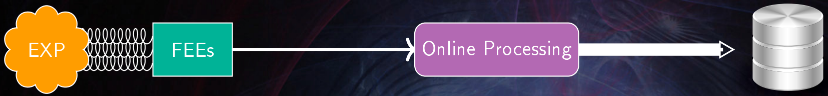
- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression

Streaming readout anatomy



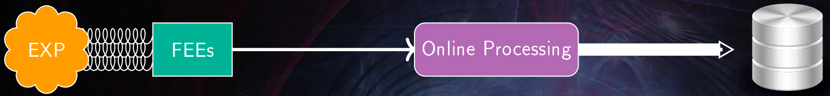
- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression
- Maybe: Per channel / detector feature extraction (total energy, tracks, etc.)

Streaming readout anatomy



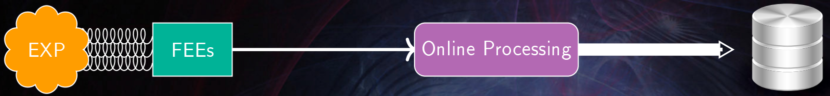
- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression
- Maybe: Per channel / detector feature extraction (total energy, tracks, etc.)
- Maybe: High level feature extraction/physics extraction

Streaming readout anatomy



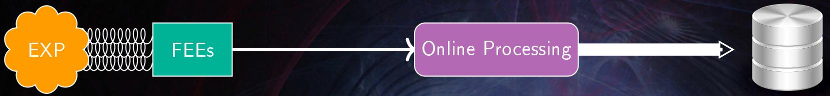
- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression
- Maybe: Per channel / detector feature extraction (total energy, tracks, etc.)
- Maybe: High level feature extraction/physics extraction
- Maybe: Data selection based on extracted features ← This is equivalent to a trigger!

Streaming readout anatomy



- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression
- Maybe: Per channel / detector feature extraction (total energy, tracks, etc.)
- Maybe: High level feature extraction/physics extraction
- Maybe: Data selection based on extracted features←This is equivalent to a trigger!
- Maybe: Remove raw data

Streaming readout anatomy



- Convert all signals continuously (or self triggered)
- Almost always: Per channel zero suppression
- Maybe: Per detector noise suppression
- Maybe: Per channel / detector feature extraction (total energy, tracks, etc.)
- Maybe: High level feature extraction/physics extraction
- Maybe: Data selection based on extracted features←This is equivalent to a trigger!
- Maybe: Remove raw data
- Save data to long term storage

Most functionality is now software!

General advantages of a SRO

- Reduce complexity in FEE
  - Remove "trigger" module from already streaming electronics

General advantages of a SRO

- Reduce complexity in FEE
  - Remove "trigger" module from already streaming electronics
- Relax hard timing constraints. Instead of latency, focus only on throughput.
  - Buffer memory in PCs is cheap. Could even use disk (LHCb)!

General advantages of a SRO

- Reduce complexity in FEE
  - Remove "trigger" module from already streaming electronics
- Relax hard timing constraints. Instead of latency, focus only on throughput.
  - Buffer memory in PCs is cheap. Could even use disk (LHCb)!
- Move complexity from hardware to software (and from hard-online to also-offline)
  - More flexible, and more people can contribute. (Also gain do-overs)

General advantages of a SRO

- Reduce complexity in FEE
  - Remove "trigger" module from already streaming electronics
- Relax hard timing constraints. Instead of latency, focus only on throughput.
  - Buffer memory in PCs is cheap. Could even use disk (LHCb)!
- Move complexity from hardware to software (and from hard-online to also-offline)
  - More flexible, and more people can contribute. (Also gain do-overs)
- Remove bottleneck of event building.
  - Can easily scale to large channel counts.
  - Event building always "brittle". What happens if FEE dies?

Streaming readout continuum

min ⟵⟶ max

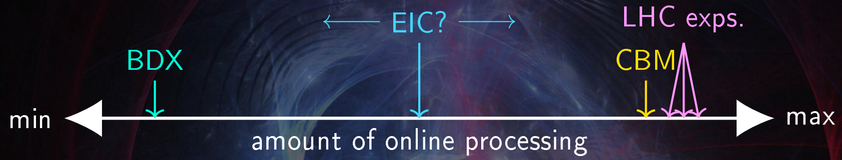
amount of online processing

- Save all data
- Lowest risk
- Maximum physics
- Highest rate

- Only keep high level data
- Highest risk
- Maximum physics/byte

- Generally: Can reach more physics than with a triggered system

Streaming readout continuum



- Save all data
- Lowest risk
- Maximum physics
- Highest rate

- Only keep high level data
- Highest risk
- Maximum physics/byte

- Generally: Can reach more physics than with a triggered system

The software goal

We want to replace

```
run_exp ; process_raw; analyze ; write_paper
```

with

```
run_exp | process_raw | analyze | write_paper
```

The software goal

We want to replace

```
run_exp ; process_raw; analyze ; write_paper
```

with

```
run_exp | process_raw | analyze | write_paper
```

...and everything right of run_exp be better non-blocking.

This task is somewhat orthogonal to SR (see HLT), but SR organically pulls in analysis into DAQ.

Software components

- A framework which can orchestrate the DAQ
  - Device bring up, node bring-up, streaming connections
  - resilient, highly performant, scalable
  - data flow management
  - composable system

Software components

- A framework which can orchestrate the DAQ
  - Device bring up, node bring-up, streaming connections
  - resilient, highly performant, scalable
  - data flow management
  - composable system
- (As universal as possible) protocols for data streams between components
- Data structures which are streaming, and HPC/accelerator compatible.
- The required filters, data reducers, analyzers, writers, readers.
  - This can potentially include everything up to the analysis
- Infrastructure for run monitoring.
- Also: MC which can produce streaming data.

Some of these things we have done before, some are newish.

Real-time, near-line, offline

Reality strikes.

There are several reason, realtime analysis (that is, of data in flight) might not be possible.

Real-time, near-line, offline

Reality strikes.

There are several reason, realtime analysis (that is, of data in flight) might not be possible.

- Analysis is not ready.
- Too CPU intensive. This can be attacked with better (or approximating) algorithms, better use of accelerators.
- Latency sensitive: Cannot make use of compute not available right now. Federated resources (traffic), work queues, etc.

Real-time, near-line, offline

Reality strikes.

There are several reason, realtime analysis (that is, of data in flight) might not be possible.

- Analysis is not ready.
- Too CPU intensive. This can be attacked with better (or approximating) algorithms, better use of accelerators.
- Latency sensitive: Cannot make use of compute not available right now. Federated resources (traffic), work queues, etc.
- Intrinsically: Some detector might need several loops over an extended set of data.

So we need to support "near online" (delays of minutes to days) and "offline" (delays of weeks, months++)

Framework must support different data sources.

Challenges I: Social issues

- Compared to HEP, NP exps are often underfunded for computing aspects.
- And not enough people.
- This problem is inverse to the project size. NP has more small-scale exps than HEP. There is a world beside EIC.

Challenges II: Data, Events

Data amounts continue to grow.

- Despite SRO (See LHCb – SRO is used to maximize physics/byte)
- Because of SRO (see sPHENIX – we plan to record data we cannot trigger on!)

Main storage will likely be tape. Have to handle working set sensibly.

Challenges II: Data, Events

Data amounts continue to grow.

- Despite SRO (See LHCb – SRO is used to maximize physics/byte)
- Because of SRO (see sPHENIX – we plan to record data we cannot trigger on!)

Main storage will likely be tape. Have to handle working set sensibly.

Streaming readout changes the type of data.

- Not event-id tagged, but time-tagged
- Event definition up to the analysis
- Different way to think about things:
  - Is imitating an "event builder" at an early stage beneficial, or does it limit us?
  - How to pump data through the analysis?

Challenges III: Compute

Compute will be centralized / federalized

- Centralized to on-site farms, away from per experiment, and counting room
- Federalized to off-site farms

Challenges:

- Orchestration
- Data storage. Traffic can be expensive.
- Less control over compute infrastructure.
- Must make good use of available hardware. Heterogenous!

Users or programmers?

Some people do analysis. Some do core programming (tracking, etc). Some do both.

Users or programmers?

Some people do analysis. Some do core programming (tracking, etc). Some do both. Fraction of both larger for small exps. For larger exps, physicists become users.

Users or programmers?

Some people do analysis. Some do core programming (tracking, etc). Some do both. Fraction of both larger for small exps. For larger exps, physicists become users.

- There is a big "black box" problem here.

Possibly conflicting goals: Make it easy for analysis people? Or make it easy to contribute?

Users or programmers?

Some people do analysis. Some do core programming (tracking, etc). Some do both. Fraction of both larger for small exps. For larger exps, physicists become users.

- There is a big "black box" problem here.

Possibly conflicting goals: Make it easy for analysis people? Or make it easy to contribute?

- DSL / high level "slow" language for analysis plumbing. Heavy lifting by libraries.
  - Easier to start. Flexibel. More productive (You don't have to sell me on python)
  - But additional barrier to start digging into the libraries.

Users or programmers?

Some people do analysis. Some do core programming (tracking, etc). Some do both. Fraction of both larger for small exps. For larger exps, physicists become users.

- There is a big "black box" problem here.

Possibly conflicting goals: Make it easy for analysis people? Or make it easy to contribute?

- DSL / high level "slow" language for analysis plumbing. Heavy lifting by libraries.
  - Easier to start. Flexibel. More productive (You don't have to sell me on python)
  - But additional barrier to start digging into the libraries.
- Jupyter / AaaS / docker images
  - Again, easier to start. Until you hit a wall.

Hic sunt dracones

The following are some devil's advocate / discussion stimulating thoughts.

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no "leaps" after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)
  - Python 3.1.2 (anybody here still using 2.7?)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no "leaps" after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)
  - Python 3.1.2 (anybody here still using 2.7?)
  - ROOT 5.26, Geant 4.9.4

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no "leaps" after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)
  - Python 3.1.2 (anybody here still using 2.7?)
  - ROOT 5.26, Geant 4.9.4
  - 14 qubits (in 2011. We are now at 53)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no “leaps” after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)
  - Python 3.1.2 (anybody here still using 2.7?)
  - ROOT 5.26, Geant 4.9.4
  - 14 qubits (in 2011. We are now at 53)
  - Aggregated top 10 of TOP500: 12.6 PFlop/s, 3 with GPU, (Today: just below 1 EFlop/s, 6+1)

Other thoughts: EIC will come in 10 years

- Infrastructure-level software must be available earlier (no "leaps" after that)
- 10 years ago:
  - I finished my PhD!
  - No TensorFlow (first release 2015)
  - No Jupyter (also 2015. IPython 2001!)
  - Windows 7, Linux kernel 2.6.??
  - GO released 2009, Rust in 2015
  - CUDA 3.0 (now 11.1)
  - Python 3.1.2 (anybody here still using 2.7?)
  - ROOT 5.26, Geant 4.9.4
  - 14 qubits (in 2011. We are now at 53)
  - Aggregated top 10 of TOP500: 12.6 PFlop/s, 3 with GPU, (Today: just below 1 EFlop/s, 6+1)
- Be ready for new things. Do not rely on them.

Other thoughts: Languages I

Close to hardware, C, C++ and things like System-C will probably be widely dominant.

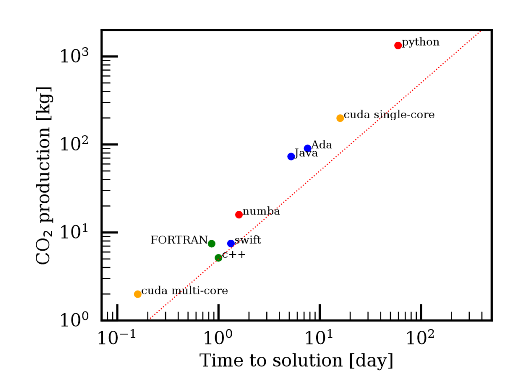
- This is where much activitiy will be done early.
- Performance, closeness to hardware. Kernel.
- Rust might be a player, but not enough people know it.

Other thoughts: Languages I

Close to hardware, C, C++ and things like System-C will probably be widely dominant.

- This is where much activitiy will be done early.
- Performance, closeness to hardware. Kernel.
- Rust might be a player, but not enough people know it.

Performance-critical infrastructure libs likely in C/C++.
True even for new accelerator/compute hardware. OpenCL, CUDA all C-like.
Some stuff will be in Fortran.



(from arxiv:2009.11295)

Other thoughts: Languages II

I despise CINT. I hate ROOT macros. I hate the bad behaviors ROOT macros have taught to generations of programmers.

Other thoughts: Languages II

I despise CINT. I hate ROOT macros. I hate the bad behaviors ROOT macros have taught to generations of programmers.
But.

Other thoughts: Languages II

I despise CINT. I hate ROOT macros. I hate the bad behaviors ROOT macros have taught to generations of programmers.

But.

It is a rather straight forward transition from using ROOT GUI, ROOT command line interface, to ROOT macros, to C++ programs using ROOT.

All documentation is bad

It really is.

All documentation is bad

It really is.

Multiple reasons:

- Nobody likes to write documentation. Doxygen is enough, right?
- Most code is written by physicists. Documentation is time/resource intensive. And hard to include on a grant application.
- The best documentation can only express what the author thinks the code does. Not what the code does.

All documentation is bad

It really is.
Multiple reasons:

- Nobody likes to write documentation. Doxygen is enough, right?
- Most code is written by physicists. Documentation is time/resource intensive. And hard to include on a grant application.
- The best documentation can only express what the author thinks the code does. Not what the code does.

Only code speaks the truth.
I regularly read ROOT/Geant4 source code to identify what I'm doing wrong (and sometimes what they are doing wrong).
Consequence: Make it easy to explore the code.