What will it take to do a HL-LHC analysis in 15'?

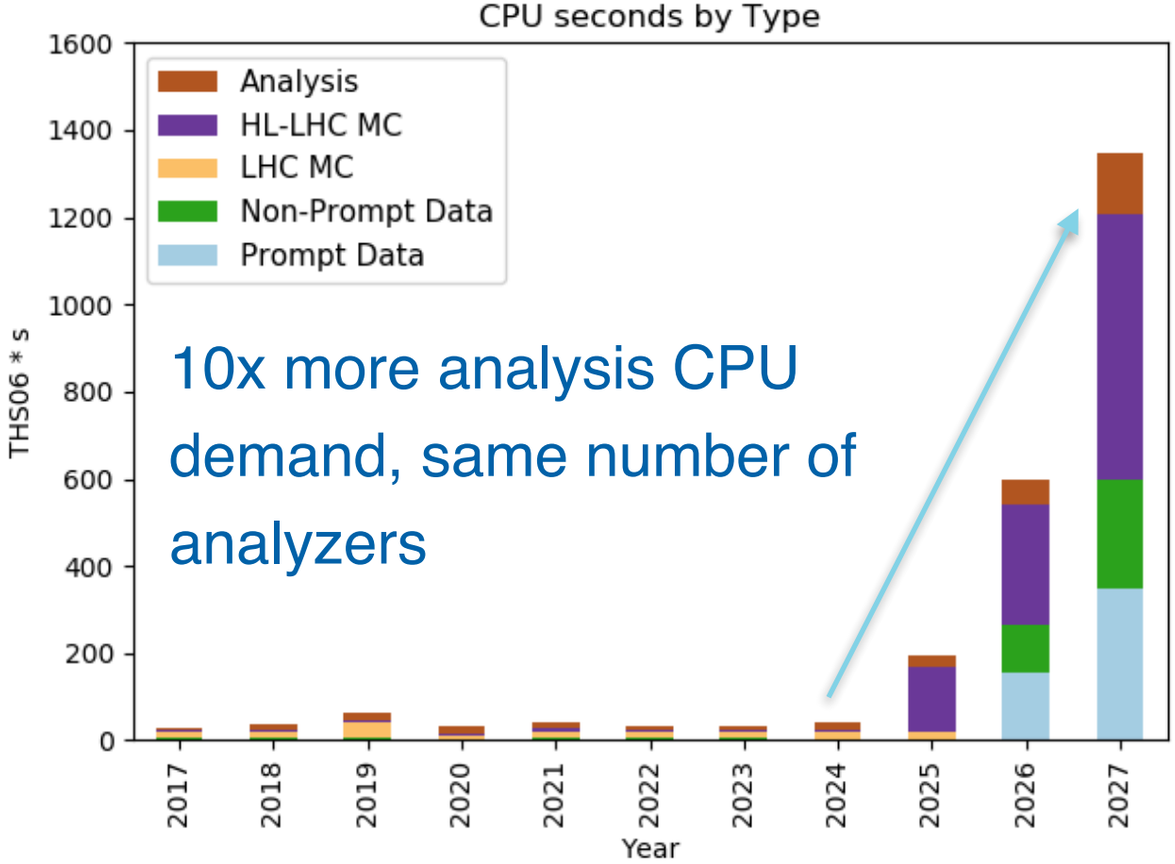
Lindsey Gray

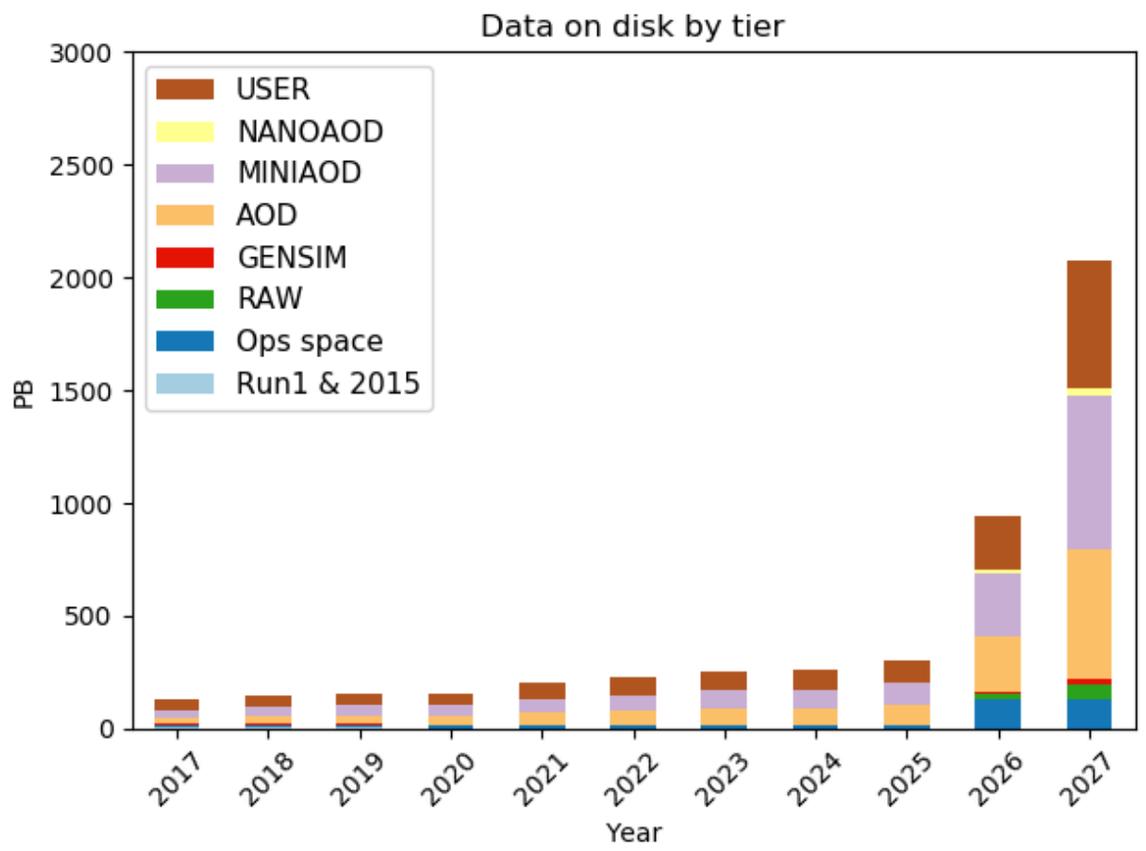
ACAT 2024, Stony Brook University

14 March 2024

The challenge of analysis for HL-LHC

- The increase total data volume of the HL-LHC begets an increase in end-user analysis data
- In order to make scientific progress with such a large amount of intricate data time-to-next-action must be short
  - Input datasets must be easy to efficient to access and interpret
  - Software and hardware infrastructure must deliver prompt, complete results without requiring undesired attention from the user

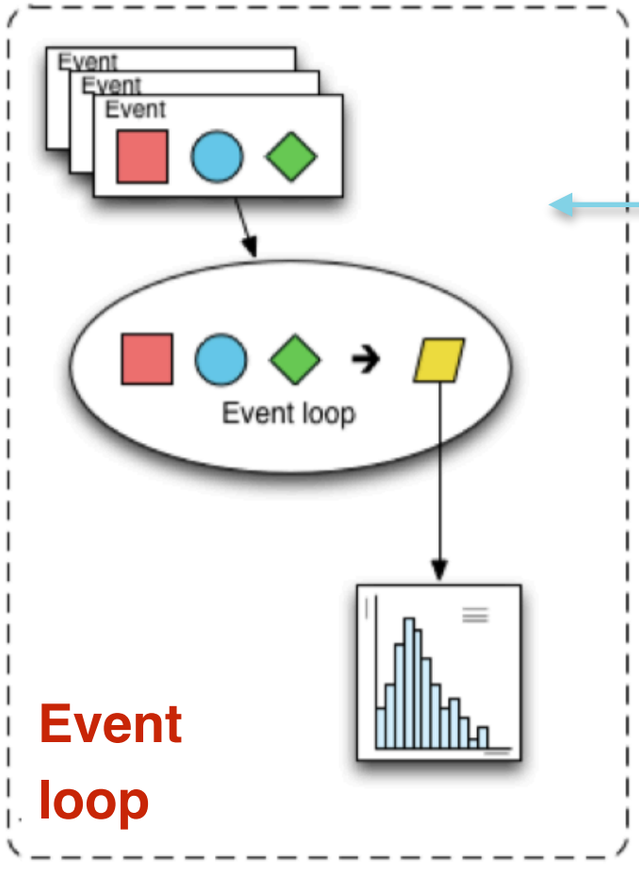


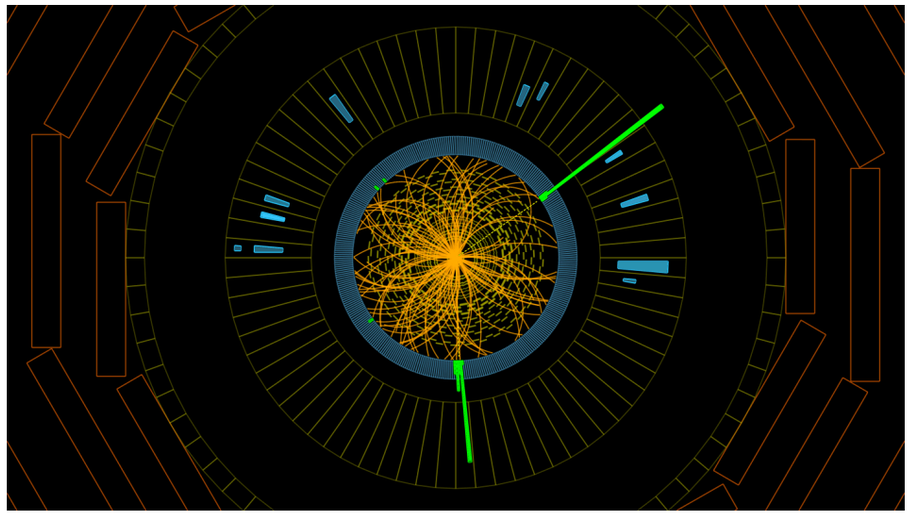


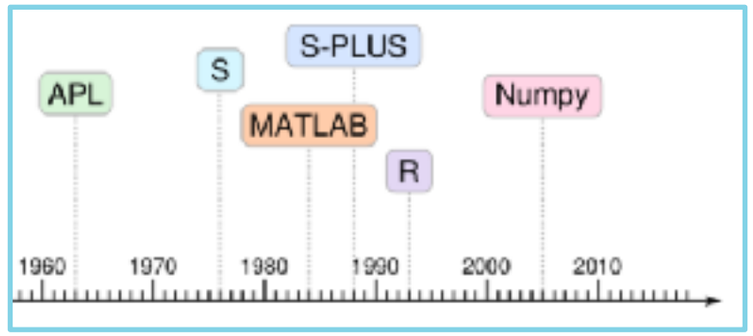
Fermilab

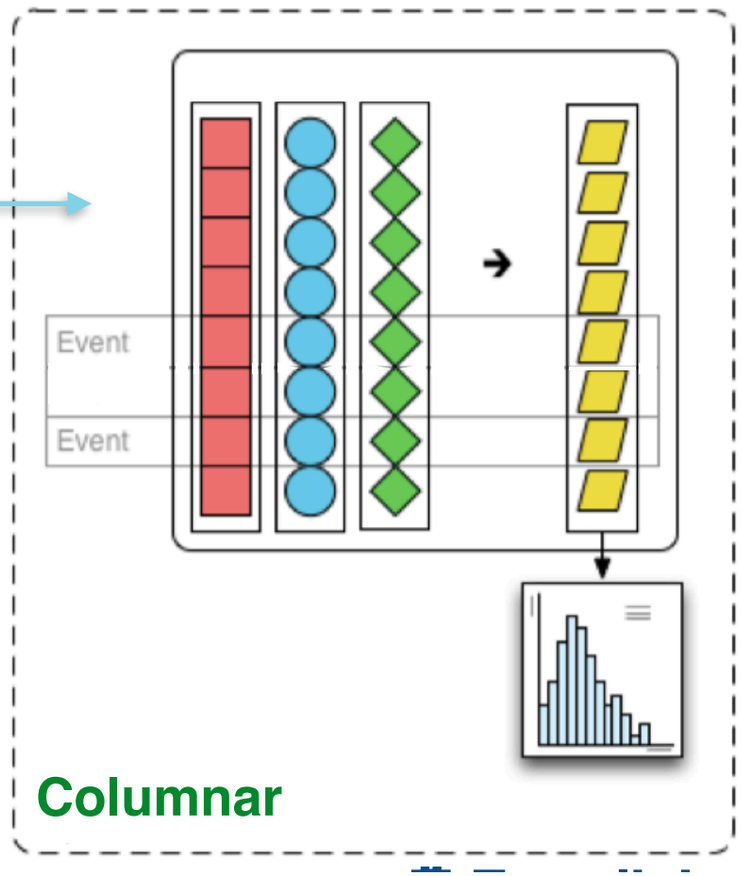
Essential difficulties of describing HEP analysis

- The first step in HEP was organizing around efficient data access as opposed to richness of data representation
  - HEP events are highly structured objects and thinking in that structure aids in analysis design
  - Highly structured data can be inefficient on disk / in memory (better with recent tech.)
- To develop an efficient system need to be able to achieve richness of expression simultaneous to efficient access



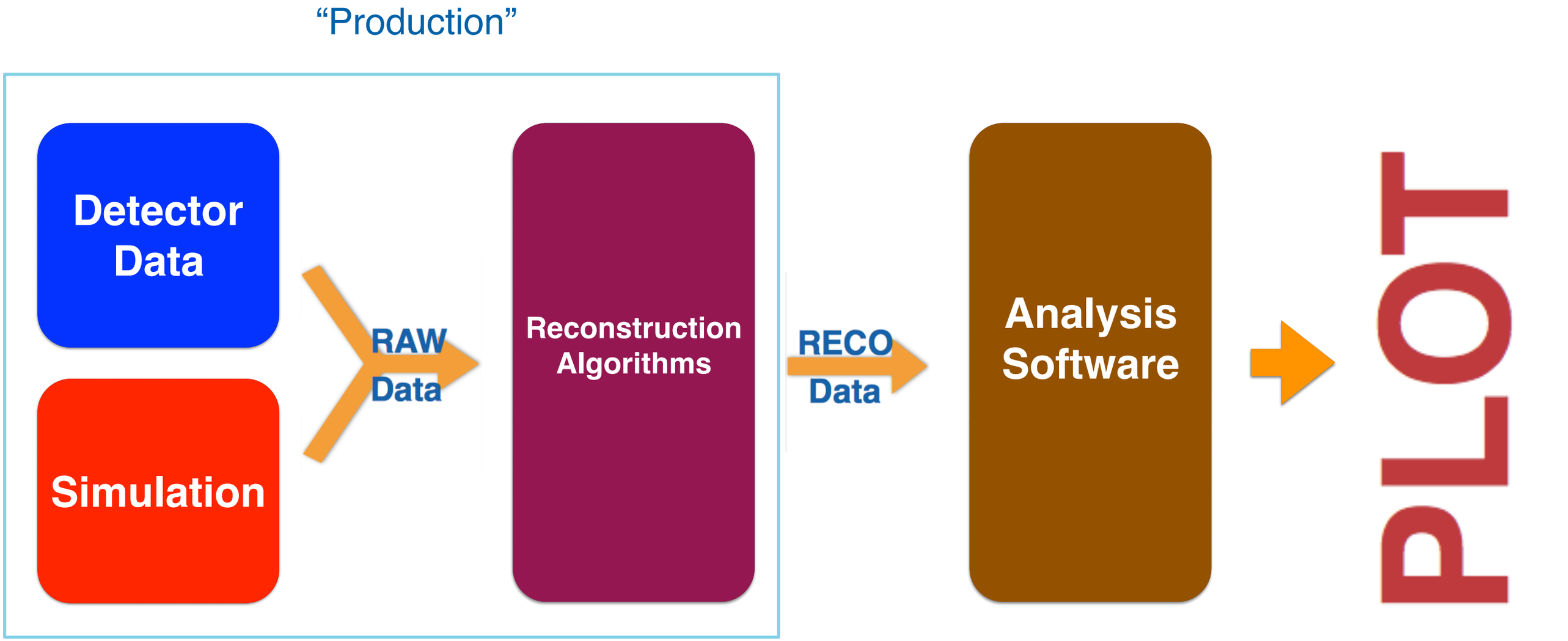






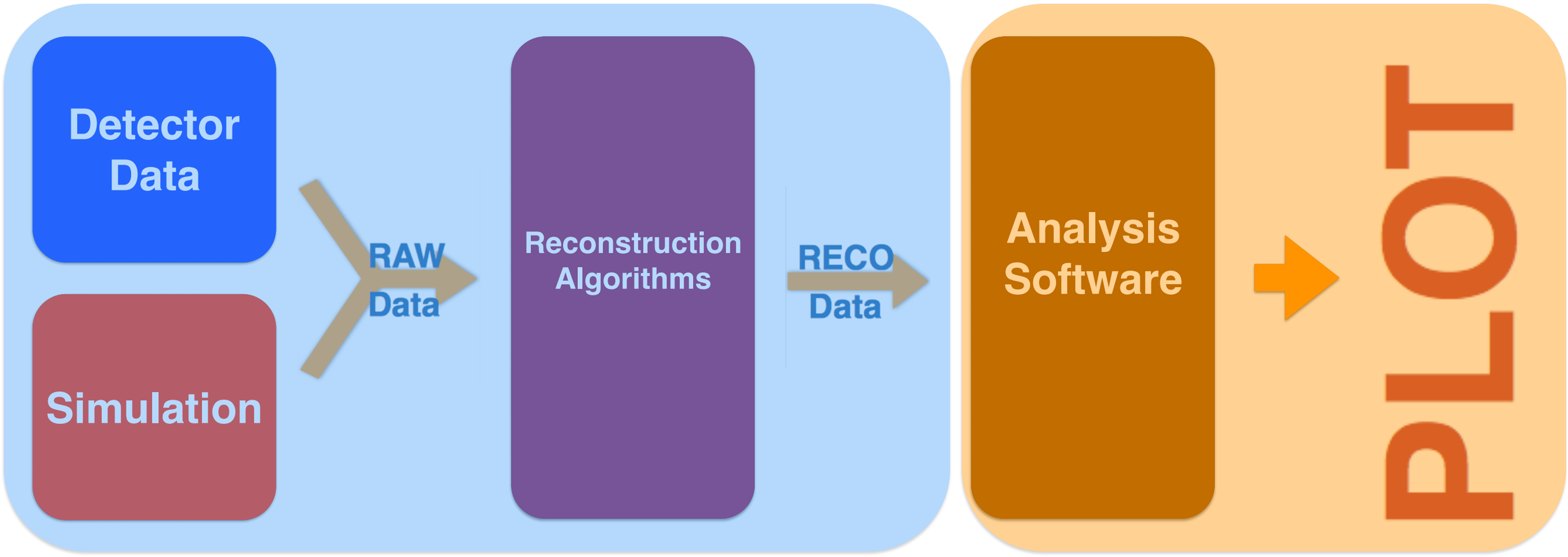
Fermilab

Outline of HEP data derivation



Fermilab

Typical Difference between Production and Analysis

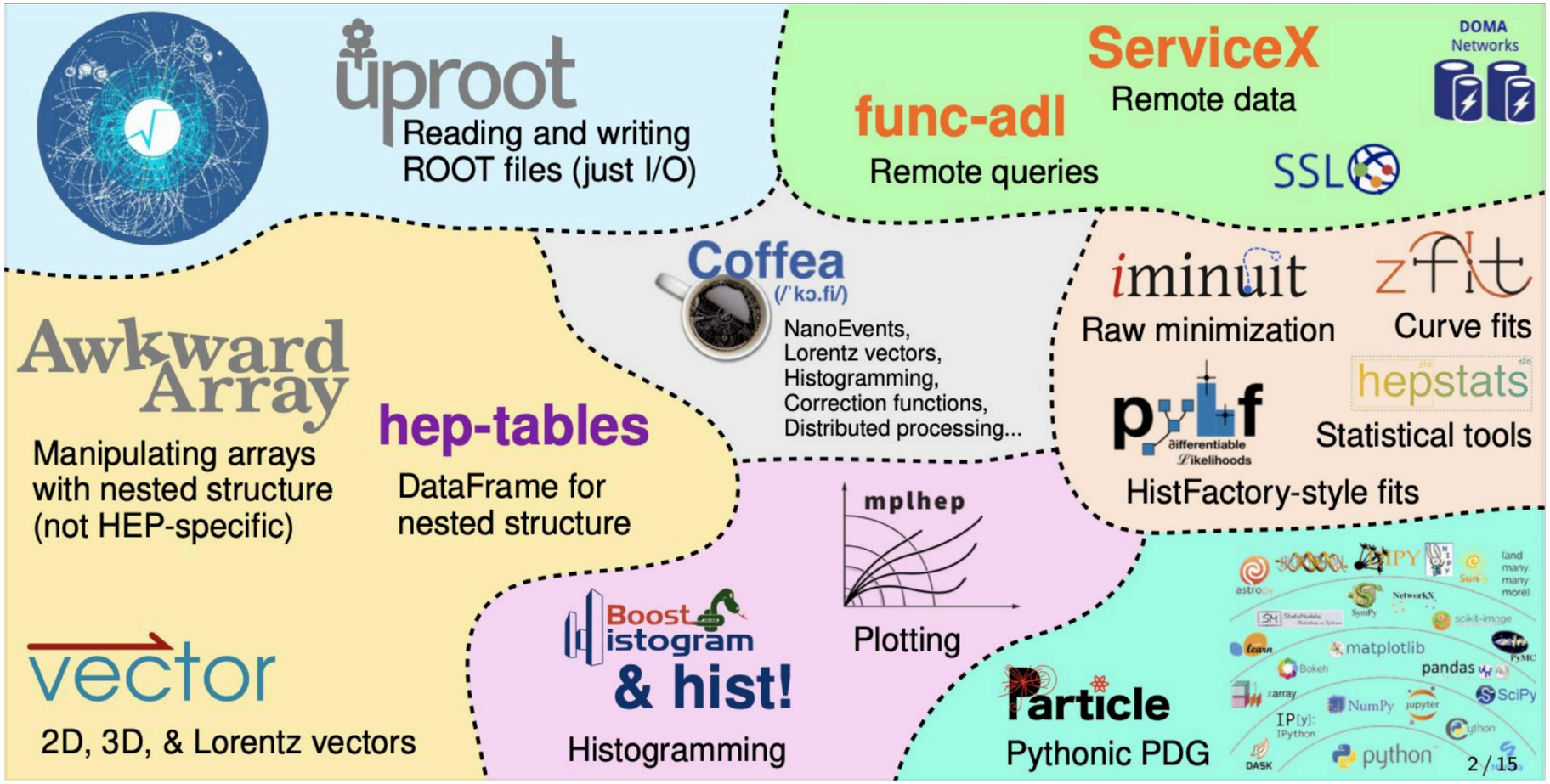


Regimented / Prescribed

Subset of similar goals, innovation breeds variation, specificity, opinion.

Fermilab

Disclaimer



- I am going to discuss from the perspective of a scientific python HEP user
  - This will bias architectural choices to some degree, but not so much
- I am doing this because it is the system, and ecosystem, that I know better
- A majority of the concepts and usage patterns are also possible with the ROOT ecosystem and other ways of ingesting/manipulating data
- I will use a number of nVidia infographics and product references
  - I am not trying to sell nVidia products, just sketching system architectures
  - Again, ecosystem bias… and they make really nice infographics

Evolutions in Abstraction

```
void MyClass::Loop() {
  size_t nEvents;
  // load...

  for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
    double MET_pt;
    int nElectron;
    double * Electron_pt;
    double * Electron_eta;
    // load...

    if ( MET_pt > 100. ) continue;

    for(size_t iEl=0; iEl<nElectron; ++iEl) {
      if ( Electron_pt[iEl] > 30. ) {
        hist->Fill(Electron_eta[iEl]);
      }
    }
  }
}
```

Event loop

```
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=events.Electron.eta[cut].flatten())
```

Columnar

Awkward Array + Coffea

Above

+

dask-awkward

```
# “array” operations only describe what is to be done
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=events.Electron.eta[cut].flatten())
# in order to render a result, we ask for it
hist.compute()
```

Delayed Columnar

Kinds of Data in the CMS Experiment

RECO: O(MB) per event

AOD: O(500 kB) per event

MiniAOD: O(40 kB) per event

NanoAOD and similar: O(2 kB) per event

- LHC experiments, and here CMS, use a variety of tiered data
- Very few analyses these days use the larger data tiers
- All analyses do histogram making, statistical analysis, ML training on flexible, concise analysis ntuples
  - This paradigm is likely not going away, and data like this will always materialize at some point in an analysis workflow
  - Software products like serviceX are helping to make these more transient and composable

Setting the Stage for 15’

- The scope of the question is enormous, so I am choosing to make some reasonable simplifications to come close to answering it!

- Analysis we’re doing in HL-LHC is going to have a large overlap with cutting edge analysis right now
  - Cutting edge analysis then will look very different, the input data volumes will be **larger** than what is outlined here due to low level input for ML
  - If we are careful, it will not be that much larger
- Assuming NanoAOD + delta we expect ~2 kB per event
  - O(1 PB) NanoAOD for HL-LHC, a typical analysis will use ~30-40% of data
  - -> Average **450 Gbit/s** to ingest data without decompression in 15 minutes
  - 2 kB / event -> 500 billion events to process
    - average **560 MHz** event processing throughput
- For facilities needs Analysis Facilities Whitepaper a good start
  - Outlines capabilities that analysis users desire, reconciles with facilities
  - This talk will largely be about capabilities and where suitable software abstractions provide the flexibility to speed things up and plan execution efficiently

Can you do this right now with CPUs and current software?

Some may ask do you need to do analysis HL-LHC in 15'? Really?

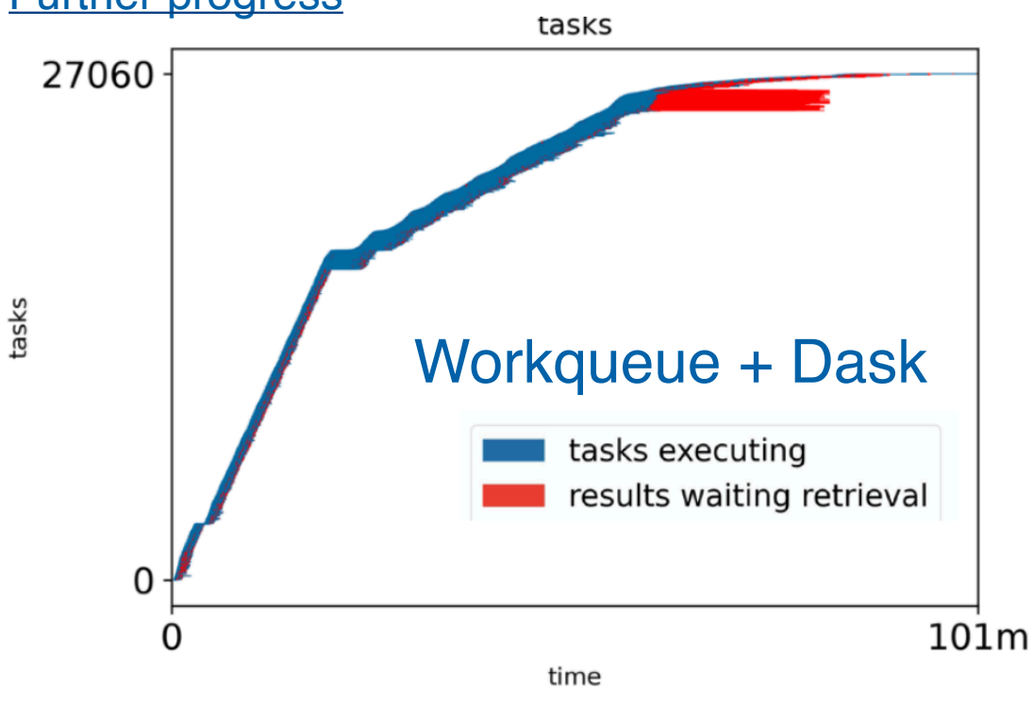
I think that's beyond the scope of this talk!

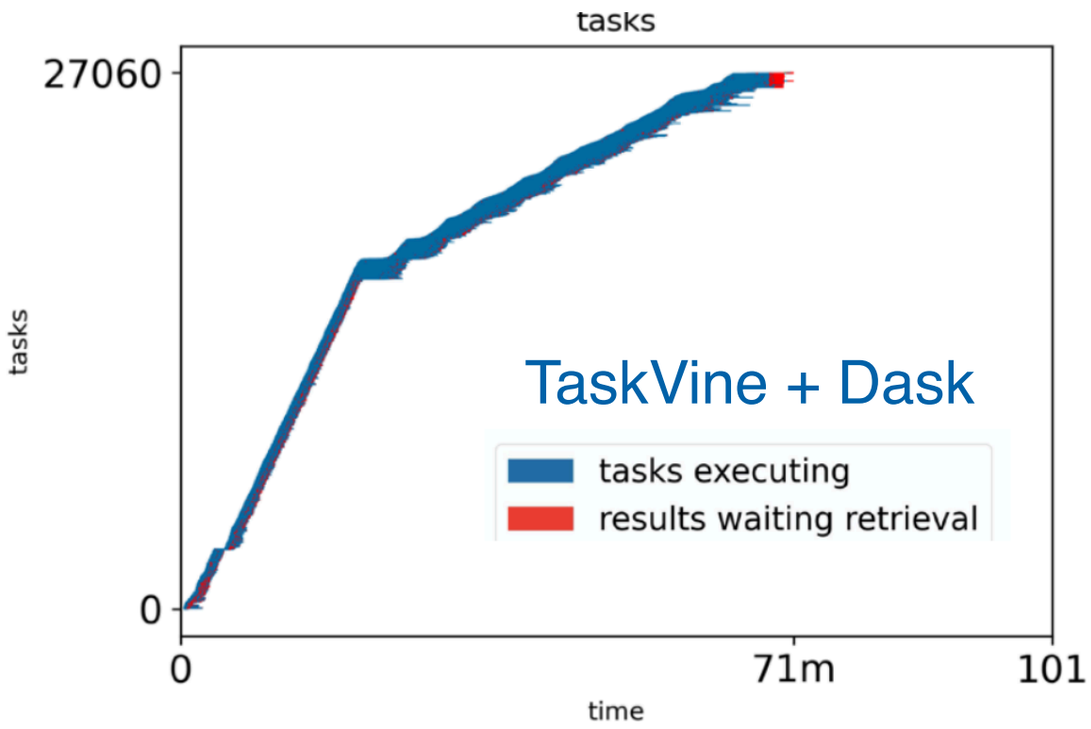
- Technically yes - but with a 400 kHz, throughput rate you'd need to
  - Would need to scale immediately to 500k cores that all have high quality access to data
- So probably really no, given the law of large numbers and many users…
- What's a way forward?

Scheduling matters more as you have more tasks!

K. Morhman and UND

Further progress

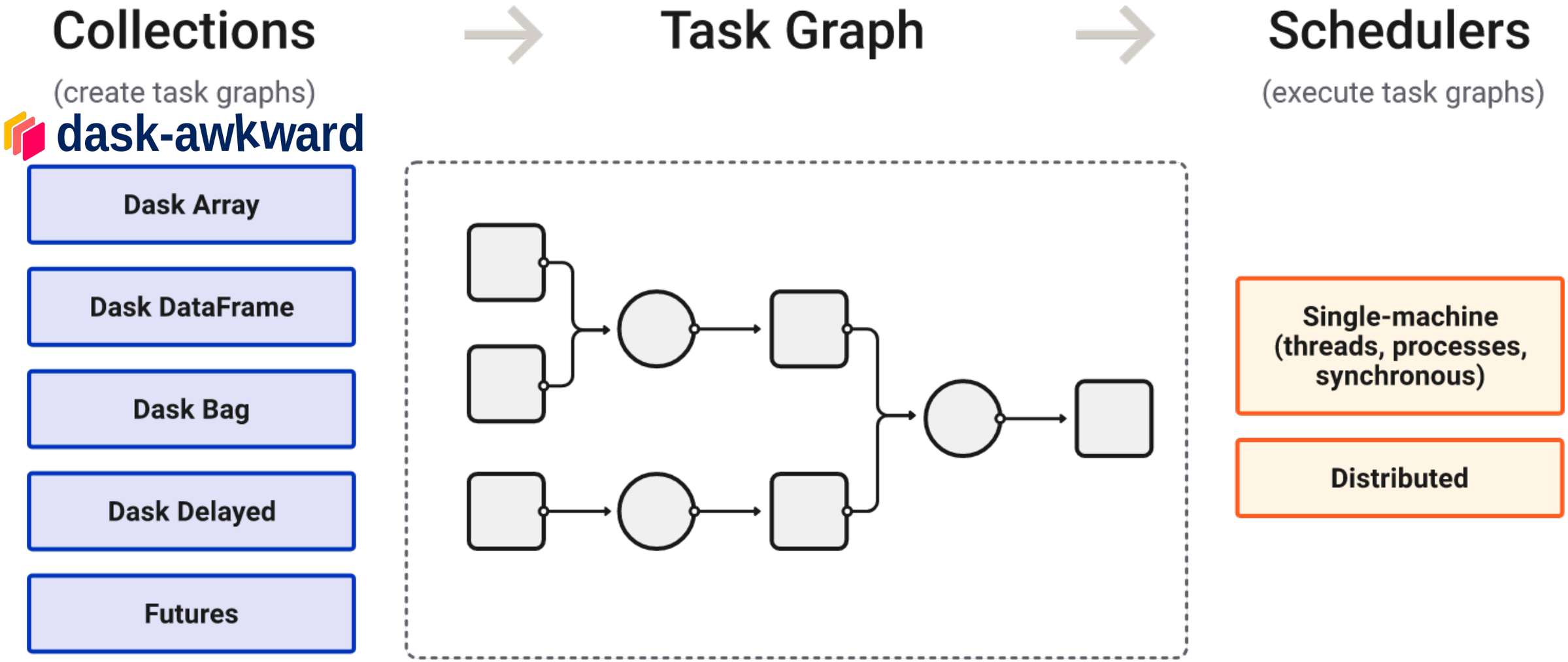




- 1h30m-2h for run 2 top analysis with full systematics achieved last year
  - Since then repeated by UCSD, UF, and time improved to 45 minutes
  - This corresponds to O(400 kHz) throughput on 1B events: 1000x away from 500 MHz!
- For ATLAS: analysis with 40s turnaround time using serviceX + coffea
  - see M. Tost poster today (note here fewer events in final processing step)
- However if we want to go faster we must expose more parallelism and not treat analyses as black boxes that fill histograms
  - Prototypes show 20m turn around for analysis with full jet re-clustering, ~1200 cores

Fermilab

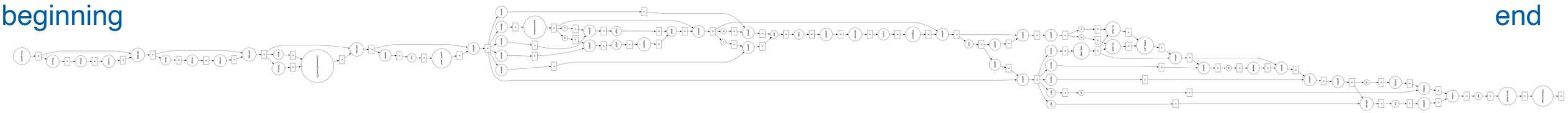
Preserving analysis flexibility with task graphs



- Dask provides an interface for specifying/locating input data and then describing manipulations on that data are organized into a task graph
  - This task graph can then be executed on local compute or on a cluster
  - This defines a DAG but not how it is traversed, **intent separated from compute**!
- A recent contribution, dask-awkward, provides this for HEP data
  - See Jim's poster from Wednesday for more info on that package family

Task graphs provide flexibility with abstraction

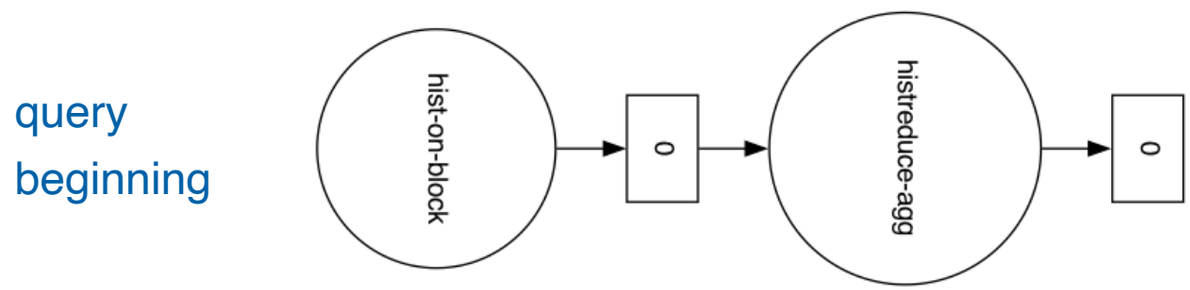
query beginning



query end

dask.optimize(q8_hist)

Bonus: free data transport and query plan optimization (here fusing linear chains)



Having *plan* of analysis lets us predict data needs!

Preserves the ability to innovate while yielding an abstraction for execution.

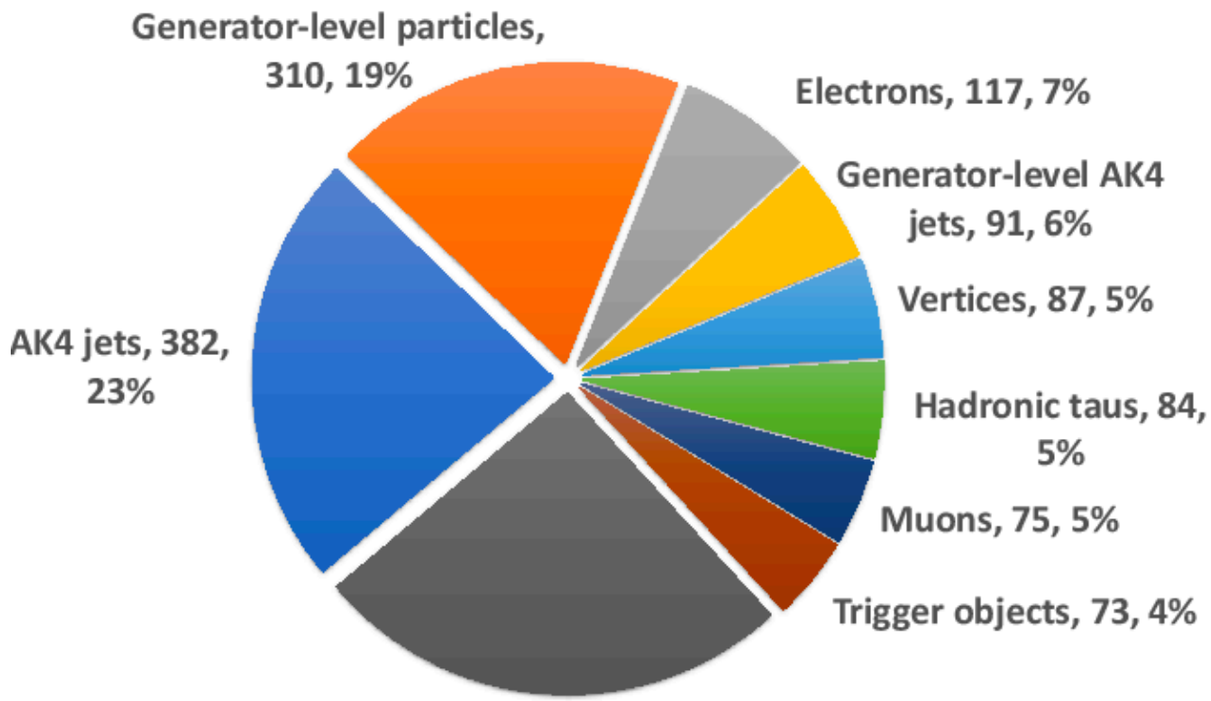
- Above an example of *part* of an analysis
  - A full HEP analysis O(1000s) of such nodes, infeasible to put on a slide
  - These graphs (with care) can be manipulated after being built and split across tools
  - Nodes in graph can be ML inference, output to disk, histogram fills, …
- By meticulously recording intent, rather than performing execution, the full depth of an any HEP analysis can be abstracted further away from compute
  - See G. Watts parallel today for further variations on how to do this!
  - With additional research it is possible to start abstracting away hardware too

Fermilab

Some first roadblocks to 15’ in data management

2kb per event size, and relative data-product size are **after** LZMA compression.

Typical NanoAOD



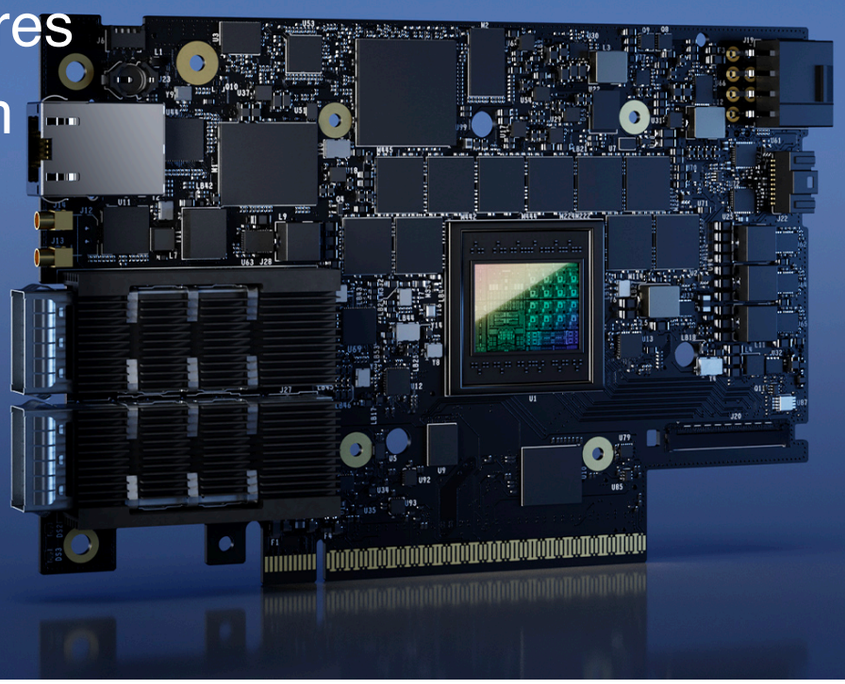
- LZMA compression is in widespread use because it is the most space efficient
  - 2x-10x slower decompression than zstandard depending on data, code dictionary size
  - Typically 1.2x-2x better in space efficiency
- Either we accelerate this algorithm or divest ourselves of it quickly
  - Both are possible with various trade offs
- Can be solved with parallelism but at the cost of node acquisition and task overhead which may easily lose throughput if not uniformly careful

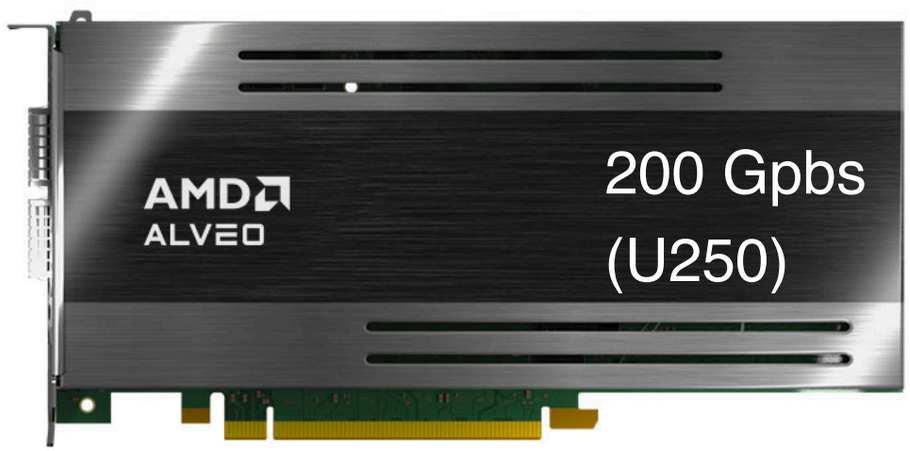
Advanced Networking Possibilities

400 Gpbs

16 arm cores

32 GB ram





Passive Option

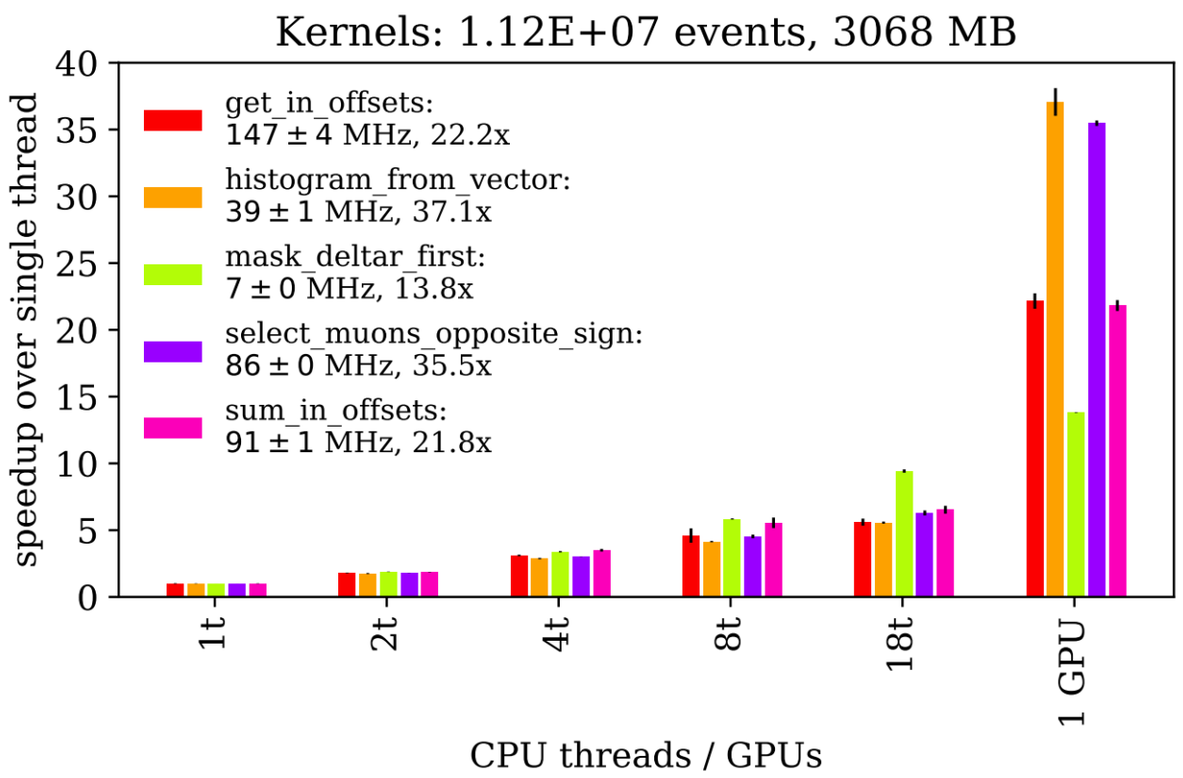
- Recall need average 450 Gbit/s for a single user to achieve necessary data ingestion
  - Using a different compression algorithm for all data this ingestion outstrips
- Left nVidia bluefield DPU, right Xilinx Alveo ultrascale FPGA
  - Both of these options can provide decompression into local memory or DMA to system
  - Knowing analysis plan yields the ability to prefetch data
- Since we are able to separate analysis intent from execution we can use accelerators like this to run basic cuts before data hits processing elements
  - Work at UCSD/UF ongoing investigating FPGA decompression / data reduction

Fermilab

Revisiting analysis on GPUs…

J. Pata: https://arxiv.org/abs/1906.06242

Reminder: need **560 MHz** event throughput



- Last benchmarked on a GeForce Titan X (many architectures ago…)
- Slowest kernel on that hardware O(100 MHz) throughput in ideal conditions
  - Local ssd cache, 10 Gbps link to node with a hefty ceph cluster backing data.
- This is incredibly ripe for benchmarking again *with the modern ecosystem*
  - An interesting target for a completed suite of awkward array GPU kernels
  - There are important details with the software ecosystem that must be explored
  - Most GPU kernels above ~100 MHz already, **5x likely from recent hardware alone**.

GPUs and the current pythonic ecosystem

Implementing Awkward CUDA Kernels #2946

ManasviGoyal started this conversation in **Ideas**

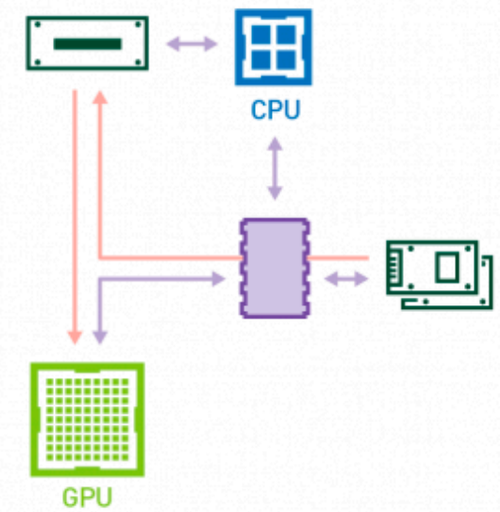
ManasviGoyal on Jan 15 Maintainer — edited

These are the 88(+13)/147 CUDA kernels that need to be implemented.

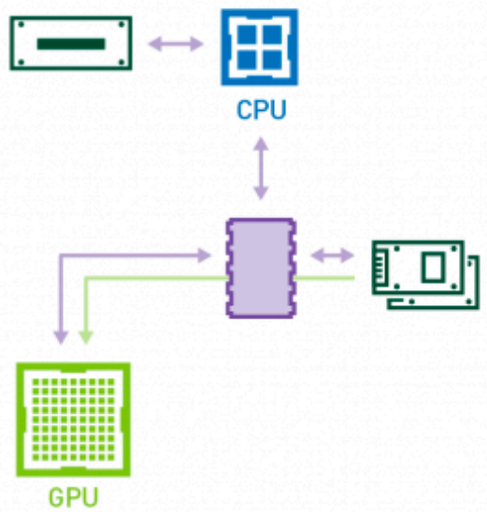
- [x] awkward_ByteMaskedArray_numnull
- [x] awkward_ByteMaskedArray_reduce_next_nonlocal_nextshifts_fromshifts_64
- [x] awkward_Index_nones_as_index
- [x] awkward_IndexedArray_fill
- [x] awkward_IndexedArray_flatten_none2empty
- [x] awkward_IndexedArray_index_of_nulls
- [ ] awkward_IndexedArray_local_preparenext_64 (A)
- [x] awkward_IndexedArray_numnull
- [x] awkward_IndexedArray_numnull_parents
- [x] awkward_IndexedArray_numnull_unique
- [x] awkward_IndexedArray_ranges_carry_next_64
- [x] awkward_IndexedArray_ranges_next_64
- [ ] awkward_IndexedArray_unique_next_index_and_offsets_64 (A)
- [x] awkward_ListArray_broadcast_tooffsets

- Awkward array GPU kernel development is underway and progressing
  - Concurrent effort to integrate awkward with cuDF (Martin Durant @ Anaconda)
- With awkward CUDA kernels current awkward python code movable to use GPUs with little change (post-hoc manipulation of task graph!)
  - Issues arise through this transition, e.g. kernel call multiplicity, *scheduling on the GPU*
- Can envision in-transit decompression on smart-NIC (DPU/FPGA) with RDMA to GPU memory across PCIe bus
  - Efficient data placement requires knowledge of analysis execution plan

What might a facility architecture look like when we can plan?



Without GPUDirect Storage



With GPUDirect Storage

Notably, this also looks like a high performance inference facility, which 15 minute analysis will also need.

- RDMA + SmartNICs + GPUs + good caching policies + excessive networking
  - nVidia has already positioned themselves well in this space
  - It is almost surprising that with good abstractions the software and hardware realign
- Map a task graph onto hardware and make appropriate library substitutions
  - User could develop code on laptop with CPU and deploy to high performance GPUs
- Reminder: this is specced for one user in 15 minutes to answer the prompt!
  - Actually scaling this out will require modification to topology, and sharing of hardware
  - Managing user expectations when system load is high is also a necessary discussion

Typical Challenges of Applying Amazing Tech to HEP Analysis

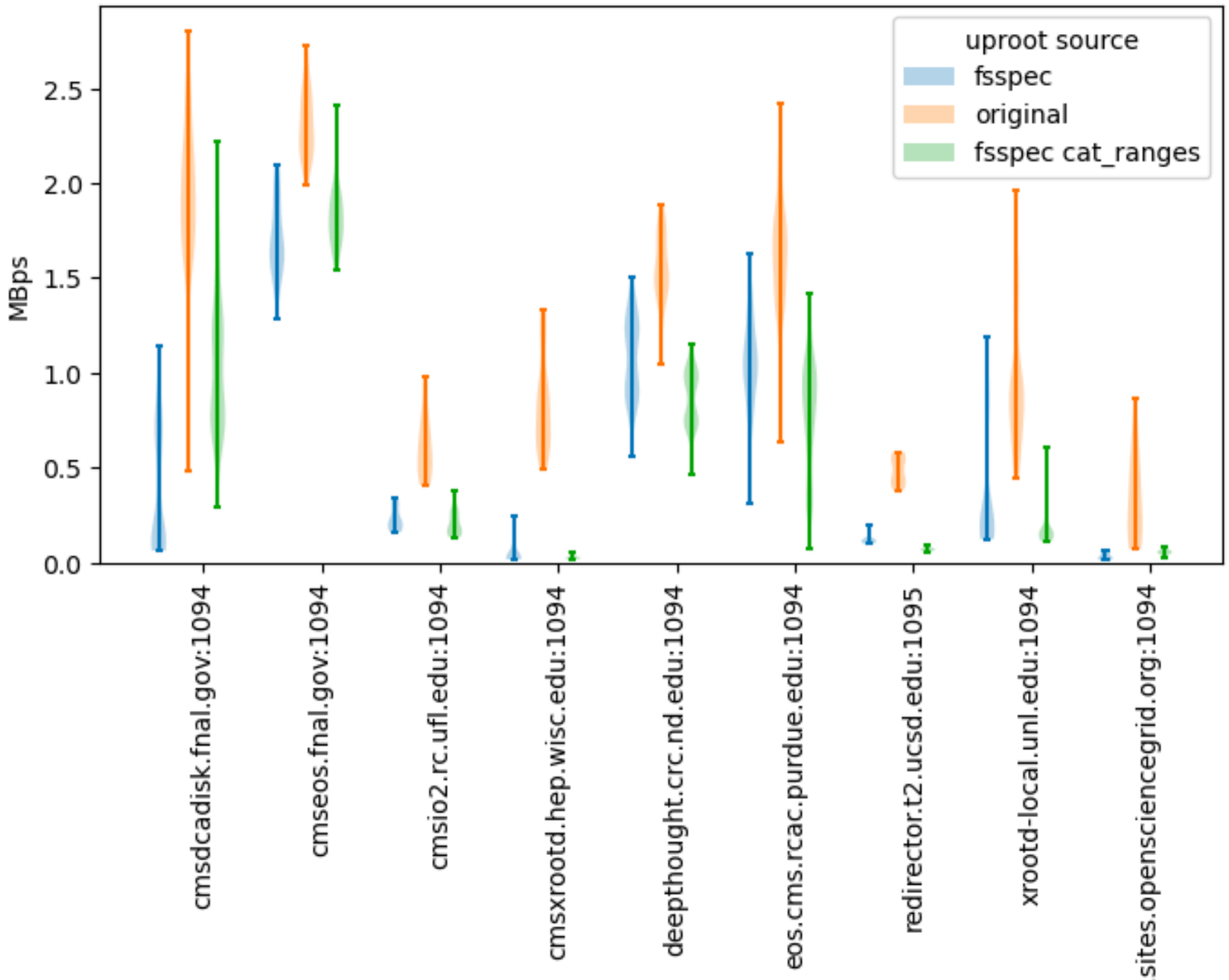
- User + high-tech gadget = CPU batch jobs with condor
  - Without really involved co-design and years of planning

- The history of HEP analysis computing has left many fantastic data processing ideas defeated and ignored
  - Often a technique works well in a specific task and is not generalizable
  - Efficient techniques that make it difficult to reason about our complex, structured, and voluminous data
  - New languages that satisfy a subset of analysis needs and are expressive but are difficult to match with tools or have unintuitive leaks

- Why do tools like Coffea (awkward array, dask, etc.), RDataFrame beat this trend, both are in routine use for years?
  - Co-design with many HEP physicists, dogfooding the tools, and years of planning
  - Extreme care is paid to making good abstractions and robust interfaces (thanks Jim!) that pay off

What else is standing in our way today?

- Our access to data (at least on CMS shows some throughput problems)
  - This is an issue *between* software and hardware for wide-area network access
  - We need to address this to really scale the previous slide's architecture



- HEP has very few systems available for testing that look like the previous slide
  - And so far all we've covered is one user for 15 minutes, assuming we can engineer away all the inefficiencies.

Please don't take this as reflective of the skill of site administrators, it is not, this is a multifaceted plot.

Fermilab

How do we keep track of progress: Benchmarks!

- Yesterday we saw the success of the MLPerf benchmarks in driving efficient ML research forward
  - Open benchmark where people can share their insights and achievements encourages friendly competition and drives the spread of knowledge
  - We should do the same if we want to achieve this goal

- Analysis Grand Challenge within IRIS-HEP already an enormous step in the right direction
  - Encapsulates typical HEP analysis needs in a variety of implementations to promote good use-case coverage of analysis platforms
    - This helps avoid the problem of creating tools that don't fill all the corners of HEP analysis
  - Can compare analysis platforms to each other at one site or compare suite of results across sites to understand inefficiencies arising from configuration or infrastructure
    - Excellent multi-modal debugging tool

- We need to support and expand this effort if we want to deploy advanced systems that are robust and also **fit user needs**.
  - More analysis varieties, more cutting edge ML - it needs input from the ACAT crowd!

Closing Remarks / Conclusions

- So - can we do HL-LHC analysis in 15'?
  - Yes but it's a long road to truly achieve that and it is presently very expensive
  - Time is on our side

- There are software projects and computing hardware tools that cover every aspect we will need to accelerate analysis to this degree
  - Modern columnar analysis tools manage the complexity and richness of HEP data and let HEP physicists express any analysis efficiently
  - Task graphs are an old proven idea with enormous utility to abstract analysis description from its execution, especially if you let them be mutable!
  - The hardware tools to accelerate HEP analysis to the necessary data throughput and event processing rate exist today.
    - The HEP community should (re-)invest in understanding those tools to a larger degree than we are if we want to achieve this lofty goal

- Benchmarking will be critical to our success and it will pay forward 10x over to develop well-covering, thorough benchmarks now
  - This has been started with the Analysis Grand Challenge and we should expand it!