The Case for Columnar Analysis

Nick Smith, on behalf of the Coffea team

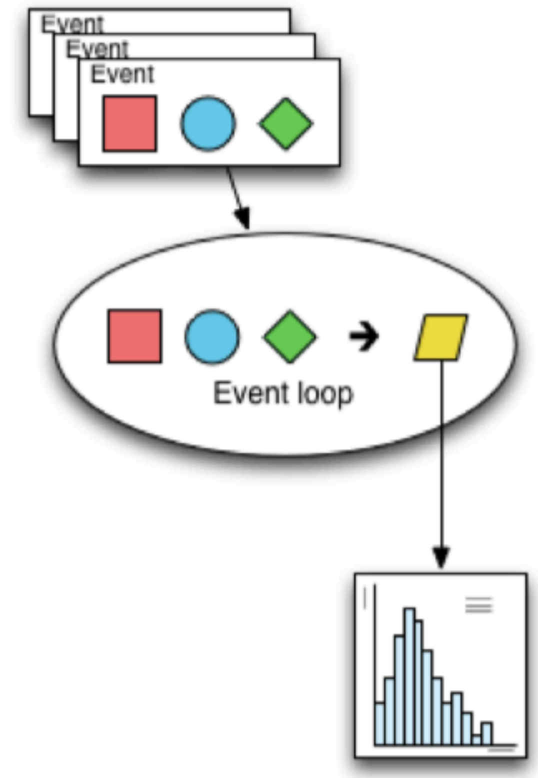
Lindsey Gray, Matteo Cremonisi, Bo Jayatilaka, Oliver Gutsche, Nick Smith, Allison Hall, Kevin Pedro (FNAL); Jim Pivarski (Princeton); and others

DAWG Technology and Innovation Survey

13 Feb. 2019

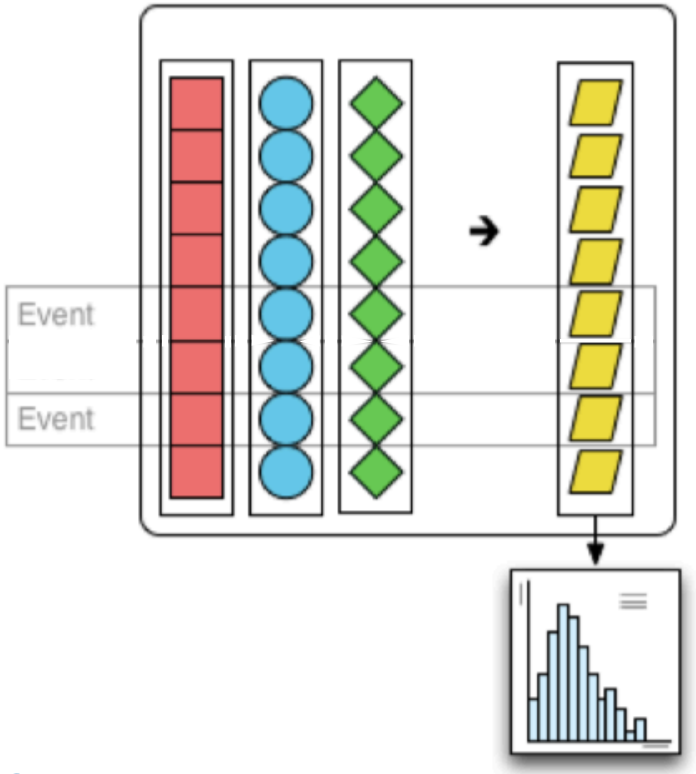
Terminology

- Event loop analysis:
  - Load relevant values for a specific event into local variables
  - Evaluate several expressions
  - Store derived values
  - Repeat (explicit outer loop)



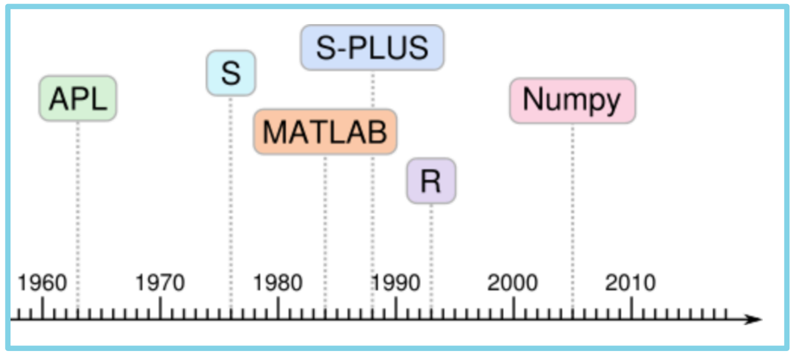
Event loop

- Columnar analysis:
  - Load relevant values for many events into contiguous arrays
    - Nested structure (array of arrays) → flat content + offsets
  - Evaluate several **array programming** expressions
    - Implicit *inner* loop**s**
  - Store derived values



Columnar

- Array programming:
  - Simple, composable operations
  - Extensions to manipulate offsets



Theoretical Motivation

- Ease of use:
  - Event loop is very imperative
    - User writes all nested loops, aggregations, filters by hand
    - Notable exceptions: std::max(), TTreeFormula, RDataFrame, …
  - Columnar analysis is a higher-level description of manipulations

- Performance benefits:
  - Aligned with strengths of modern CPUs
    - Simple instruction kernels aid pipelining, branch prediction, and pre-fetching
    - Event loop = input data controlling instruction pointer = less likely to exploit all three!
    - *Unnecessary work is cheaper than unusable work*
  - Inherently SIMD*-friendly
    - Event loop cannot leverage SIMD unless inter-event data sufficiently large
  - In-memory data structure *exactly* matches on-disk serialized format
    - Event loop must transform data structure - significant overhead
    - Memory consumption managed by chunking (event groups, or baskets)
  - Array programming kernels form computation graph
    - Could allow query planning, automated caching, non-trivial parallelization schemes

*Single-Instruction Multiple-Data

Scope

- Domain of applicability depends on:
  - Complexity of algorithms
  - Size of per-event input state
- Examples:
  - JEC (binned parametric function): use binary search, masked evaluation: **columnar ok**
  - Object gen-matching, cross-cleaning: min(metric(pairs of offsets)): **columnar ok**
  - Deterministic annealing PV reconstruction: large input state, iterative: **probably not**
- How far back can columnar go?
  - *Missing array programming primitives not a barrier, can always implement our own*

Event loop → **Columnar**

Event Reconstruction 1 MB/evt Complex algorithms operating on large per-event input state Inter-event SIMD	**Analysis Objects** 40-400 kB/evt Fewer complex algorithms, smaller per-event input state	**Filtering & Projection** (skimming & slimming) 1 kB/evt Few complex algorithms, O(1 column) input state	**Empirical PDFs** (histograms) No event scaling Trivial operations

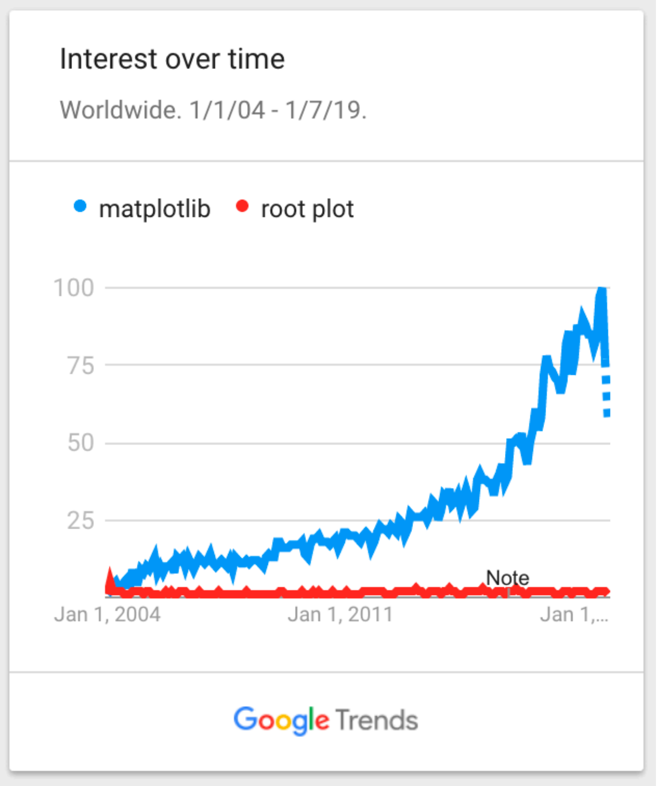
Fermilab

The Coffea framework

- COmpact Framework For Effective Analysis:
  - Prototype analysis framework utilizing columnar approach
  - Provides object-class-style view of underlying arrays
  - Implements typical recipes needed to operate on NANOAOD-like nTuples
  - Currently in fnal-column-analysis-tools
    - Functionality will be factorized into targeted packages as it matures
- Realized using:
  - Scientific python ecosystem:
    - numpy, numba, scipy, matplotlib
  - Awkward-array:
    - array programming primitives to handle “Jagged Arrays” (e.g. Muon_pt)
- Factorized data delivery:
  - Uproot: *direct* conversion from TTree to numpy arrays
  - Striped: NoSQL database of column chunks, caching layer, job scheduler
  - In discussion with other interested parties, any column chunk delivery mechanism is viable

User experience

- Two analyses being ported to columnar style
  - End-to-end: nTuple to templates + control plots
  - We export TH1s and use combine…for now
  - Dark Higgs search
    - Starting from private NanoAOD (w/addl. DeepAK8 info)
  - Boosted SM Hbb
    - Starting from BaconProd (similar to NanoAOD)
    - Already providing useful input into analysis strategy
- Alpha testers!
  - One student was given setup script, and three days later had a 2D S/sqrtB optimization plot
- Fast learning curve for scientific python stack
  - Excellent 'google-ability'
  - The quality and quantity of off-the-shelf components is impressive—many analysis tool implementations contain very little original code



Code samples

- Idea of what it might look like (heavily biased by our experiences and tastes)
- Python allows very flexible interface, under-the-hood data structure is columnar

```
ele = electrons[(electrons.p4.pt > 20) &
                                   (np.abs(electrons.p4.eta) < 2.5) &
                                   (electrons.cutBased >= 4)]

mu = muons[(muons.p4.pt > 20) &
                           (np.abs(muons.p4.eta) < 2.4) &
                           (muons.tightId > 0)]
```

- Select good candidates (per-entry selection)

```
ee = ele.distincts()
mm = mu.distincts()
em = ele.cross(mu)
```

- Pair combinatorics (creates new pairs array, also jagged)

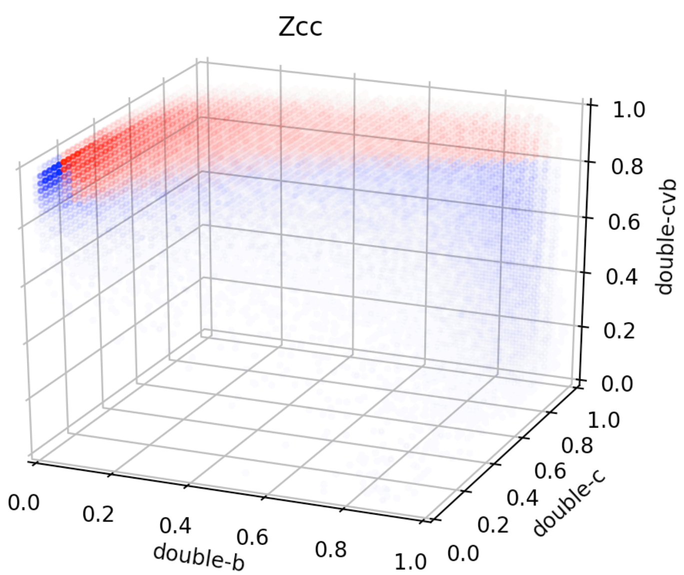
```
channels['ee'] = good_trigger & (ee.counts == 1) & (mu.counts == 0)
channels['mm'] = good_trigger & (mm.counts == 1) & (ele.counts == 0)
channels['em'] = good_trigger & (em.counts == 1) & (ele.counts == 1) & (mu.counts == 1)
```

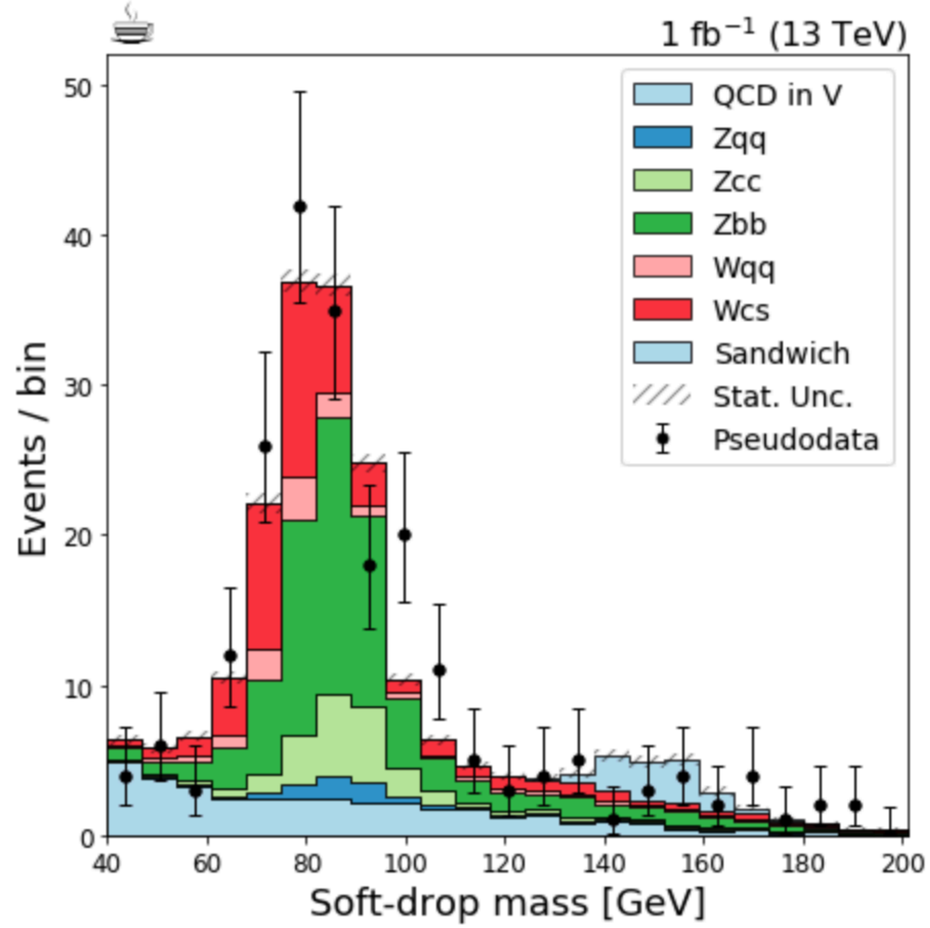
- Select good events, partitioning by type (per-event selection)

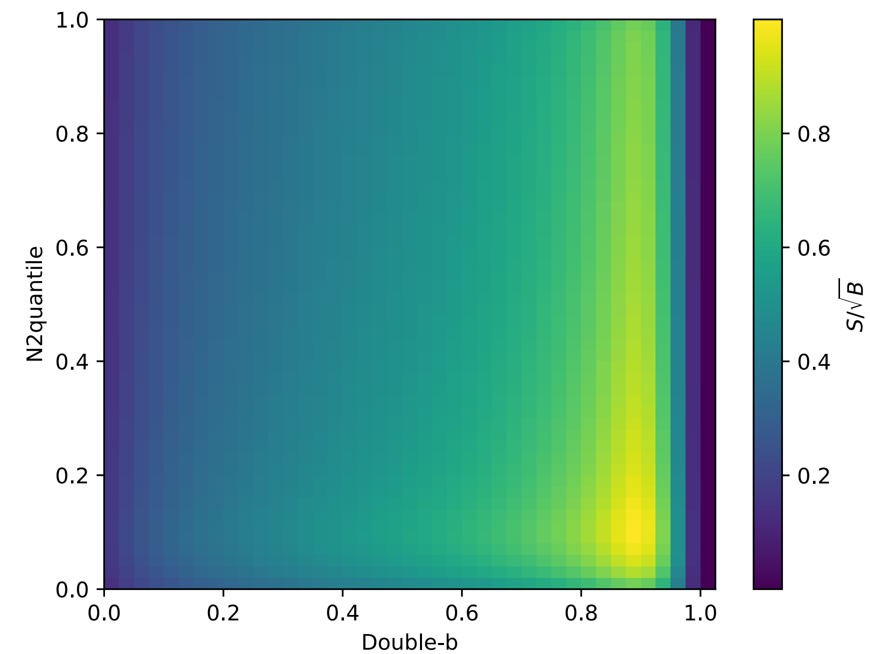
```
dileptons['ee'] = ee[(ee.i0.pdgId*ee.i1.pdgId == -11*11) & (ee.i0.p4.pt > 25)]
dileptons['mm'] = mm[(mm.i0.pdgId*mm.i1.pdgId == -13*13)]
dileptons['em'] = em[(em.i0.pdgId*em.i1.pdgId == -11*13)]
```

- Select good pairs, partitioning by type (per-entry selection on pairs array)

Eye candy





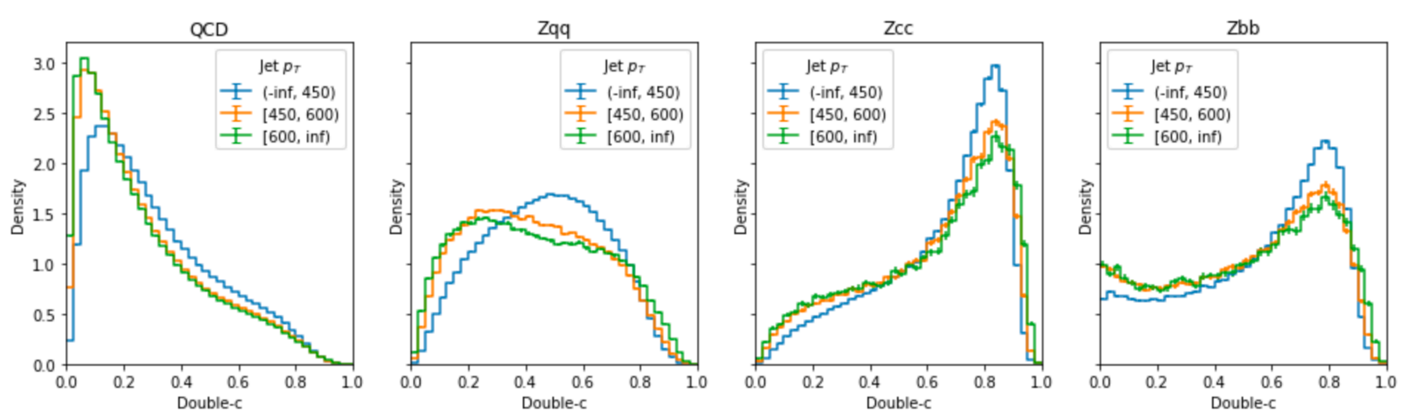


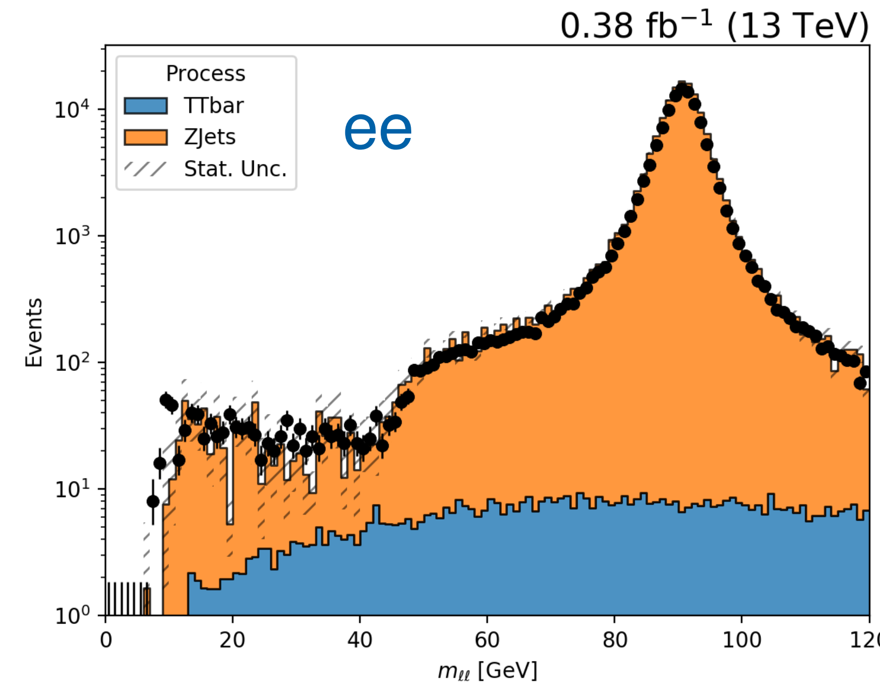

```
plot_opts = {'error_opts': {}, 'overflow': 'none', 'overlay_overflow': 'all', 'density': True}

print("msd bin:", hists['htagtensor'].axis("AK8Puppijet0_msd")[1])
htagtensor = hists['htagtensor'].project("AK8Puppijet0_msd", overflow='none')
htagtensor.label = "Density"

hc = htagtensor.sum("AK8Puppijet0_deepdoubleb", "AK8Puppijet0_deepdoublecvb")
fig2,_ = plot.plotgrid(hc, col="process", overlay="AK8Puppijet0_pt", **plot_opts)
fig2.savefig("plots/doublec.pdf")
```

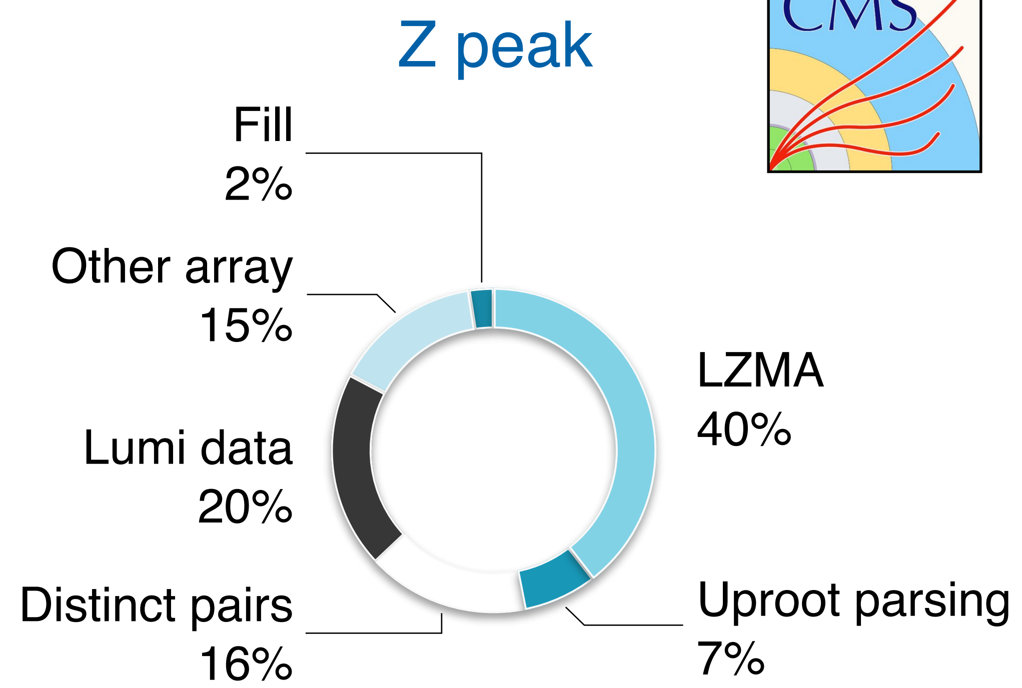
```
msd bin: [40, 200)
```

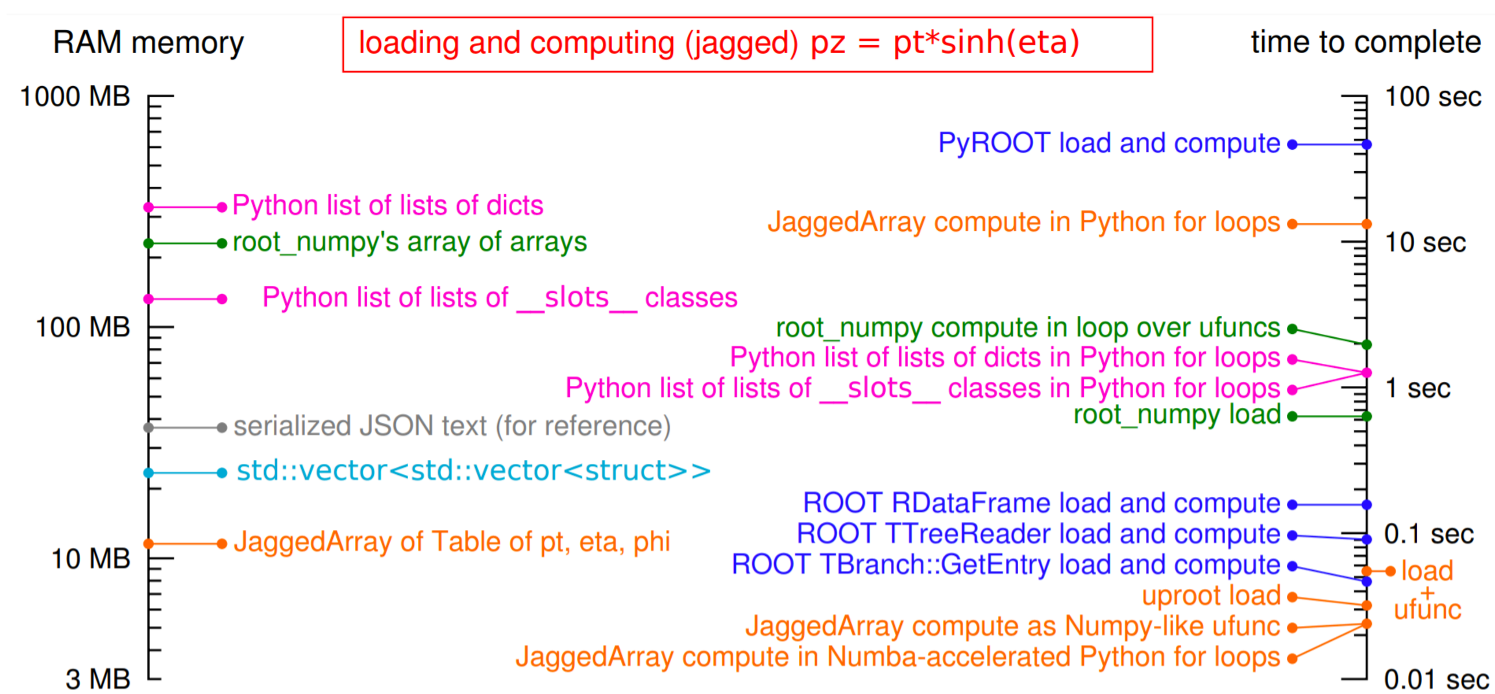




Performance

- Z peak example
  - Includes lumimask, PU correction, ID scale factors, electron & muon categorized
  - 8 μs/evt/thread (125 kHz) wall time
    - ROOT C++ SetBranchAddress: ~1.5x faster
- Boosted Hbb prototype
  - Includes recursive gen parent finding, gen matching, binned corrections, parametric corrections, systematics
  - 30 μs/evt/thread
- More inefficiencies can be removed





Fermilab

Future Directions

- As Coffea (& underlying libraries) matures, invite beta testers
  - I encourage everyone to try uproot+numpy now, it can be very effective for small checks
- Target first release this summer
  - Two full analysis implemented
  - Data delivery mechanisms fully separated
  - User interface improvements and documentation

- Far future: analysis facility
  - This feeds towards the dream of a “short time-to-insight” “analysis as a service” facility
    - Tendering bids for additional buzzwords
  - Array programming allows easier construction of computation graphs
    - Query planning can detect common patterns and execute them once
    - By removing manual cache management, we can optimize throughput and storage

- First, lets see if we are happy and productive with the columnar approach
  - So far, the answer appears to be yes