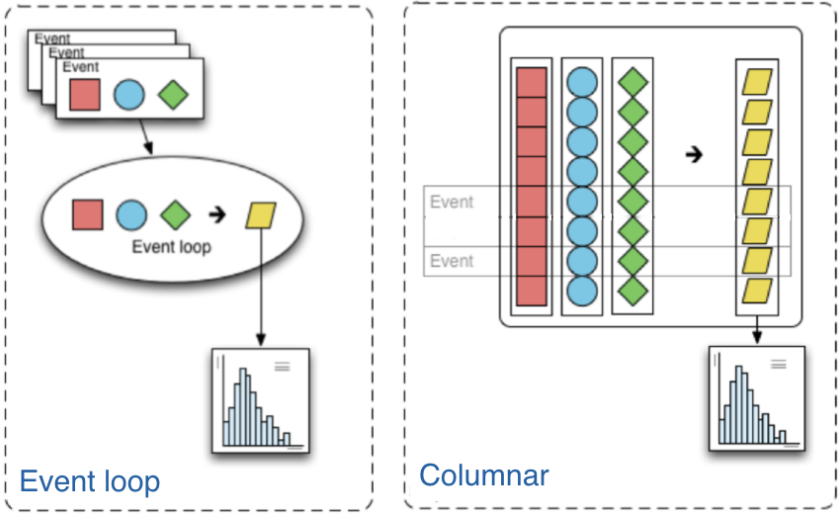
Columnar data analysis with ATLAS analysis formats

Nikolai Hartmann[1], Guenter Duckeck[1], Johannes Elmsheuser[2]
on behalf of the ATLAS collaboration

[1]Ludwig-Maximilians-Universität München (LMU)
[2]Brookhaven National Laboratory (BNL)

May 19, 2021, vCHEP 2021

Columnar data analysis - Motivation



[1] Plot from https://coffeateam.github.io/coffea/concepts.html

Columnar data analysis - Motivation

Operate on columns - "array-at-a-time" instead of "event-at-a-time"

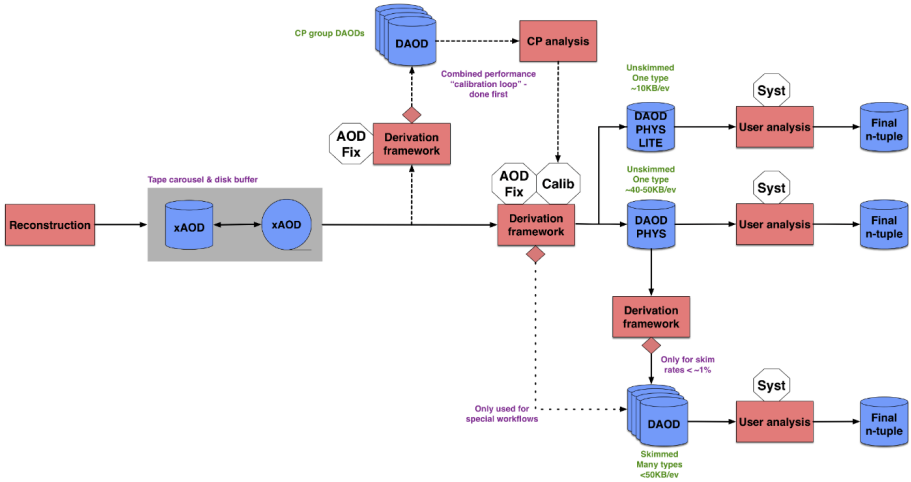
Advantages:

- Predefined operations, no for loops! Most prominent example: numpy
  → Move slow bookkeeping out of the event loop
  → **Write analysis code in python instead of c++**
- Run on contiguous blocks in memory
  → fast (good for CPU cache, vectorizable)
- Advances in tools in recent years
  → data science/machine learning
  → also in HEP: uproot, awkward array, coffea

Disadvantages

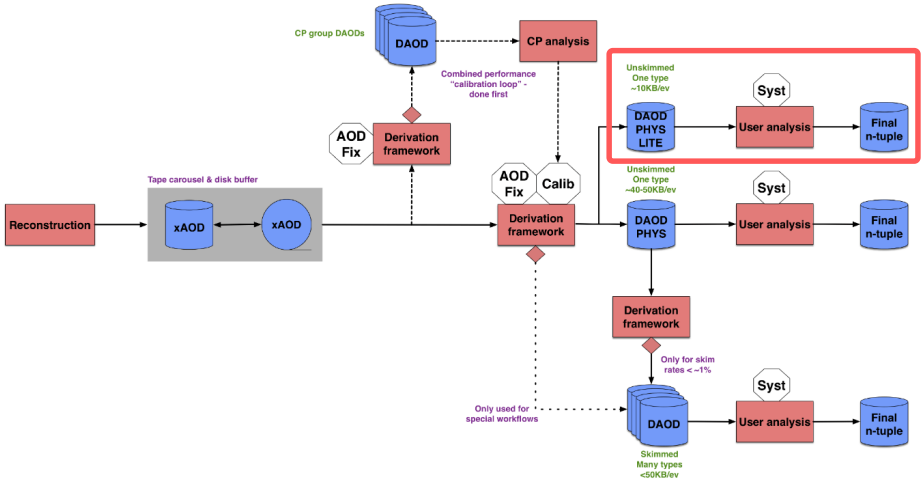
- Arrays need to be loaded into memory
  → need to process chunk-wise if amount of data too large
- Some operations complex to implement
  (e.g combinatorics, nested selections, variable length lists per event)

ATLAS analysis model starting from Run 3



[1]See J. Elmsheuser et al., EPJ Web Conf. 245, 06014 (2020)

ATLAS analysis model starting from Run 3



[1]See J. Elmsheuser et al., EPJ Web Conf. 245, 06014 (2020)

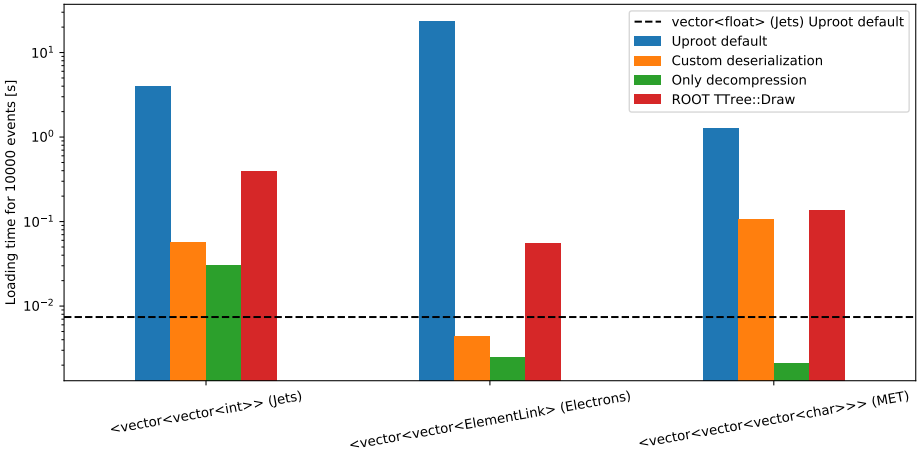
Read DAOD_PHYSLITE with uproot

DAOD_PHYSLITE ROOT format has most data split into columns, but

- Some branches have higher level of nesting (`vector<vector<...>>`)
  → typical case: Link into other collection (an electron can have multiple tracks associated)
- Those can't be split by ROOT
- Also, need to loop through data to "columnize"
  - slow in python (current Uproot implementation)
  - developed a solution based on Numba (python functions JIT compiler) for now
  - in the future a Forth machine in the Awkward array package will handle this
    → Next presentation: "AwkwardForth: accelerating Uproot with an internal DSL"

Read DAOD_PHYSLITE with uproot

Branches with more than one level of nested lists



→ efficient columnar reading of DAOD_PHYSLITE ROOT files possible!

Alternative storage formats

Loading times for all columns (≈1000) of 10k DAOD_PHYSLITE events:

Format	Compression	Dedup. offsets	Size on disk	Execution time
(Up)root	zlib	No	117 MB	6.0 s
(Up)root (large baskets)	zlib	No	116 MB	5.0 s
Parquet	snappy	No	121 MB	0.6 s
Parquet	snappy	Yes	118 MB	0.6 s
HDF5	gzip	No	101 MB	2.0 s
HDF5	gzip	Yes	89 MB	1.6 s
HDF5	lzf	No	137 MB	1.5 s
HDF5	lzf	Yes	113 MB	1.1 s
npz	zip	No	92 MB	2.0 s
npz	zip	Yes	82 MB	1.5 s

Parquet seems especially promising, but all tested formats faster than Up(root)

(Note: constant overhead for Uproot, will be less significant for larger number of events)

Dedup. offsets: store event offsets for collections only once instead of for each column

Python packages used: Uproot (root), PyArrow (parquet), h5py (HDF5), Numpy (npz) and Awkward for conversion

Event data model using awkward array

Prototype for DAOD_PHYSLITE

Currently prototype
→ Working to include DAOD_PHYSLITE schema into the coffea NanoEvents module
→ CoffeaTeam/coffea#527

```
>>> import awkward as ak
>>> events[ak.num(events.Electrons) >= 1].Electrons.pt[:, 0]
<Array [7.36e+03, 8.84e+04, ... 3.27e+04] type='20194 * float32'>
```

→ filtering on different levels

```
>>> events.Electrons.trackParticles.z0
<Array [[[-47]], [], ... ] type='50000 * var * var * float32'>
```

→ dynamically create cross references from indices

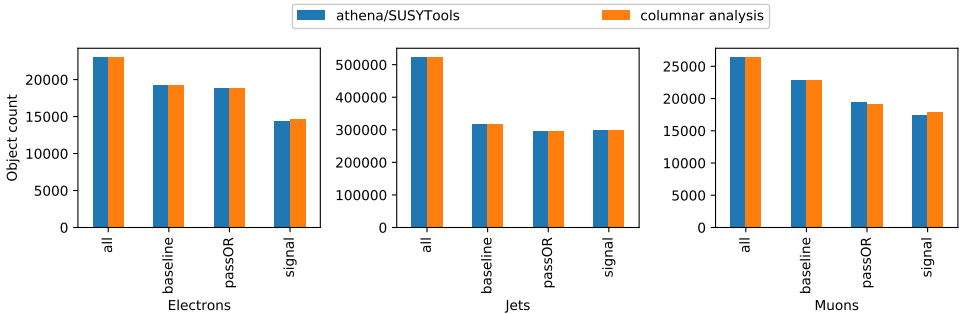
```
>>> events.Electrons.trackParticles
<xAODTrackParticleArray [[[{chiSquared: 56.4, ... z0: 0.239}]]]
type='50000 * va...'>
>>> events.Electrons.trackParticles.pt
<Array [[[7.24e+03]], ... 3.09e+04, 4.88e+03]]]
type='50000 * var * var * float32'>
```

→ dynamically calculate momenta from track parameters

```
>>> electrons = Events.electrons
>>> jets = Events.jets
>>> electrons.delta_r(electrons.nearest(jets)) < 0.2
<Array [[True], [], [], ... True], [], [True]]
type='50000 * var * ?bool'>
```

→ more advanced LorentzVector calculations

Trying to do an actual analysis



- Start with some simple object selections on Electrons, Muons, Jets
  → most challenging part: getting all the overlap removal logic correct
- Compare with SUSYTools framework (athena analysis)
  → largely reproduced selected object counts
- Many things still missing/to be developed
  → e.g. MET calculation, Pileup reweighting, Systematics

Performance

Measurement	Total time [s]	average no. events / s
Athena/SUSYTools	22	2300
Columnar	3.8	13000
Columnar (cached)	1.2	42000

- In all cases: Read with "warm" page cache
- "Cached" for columnar analysis means data already decompressed and deserialized

Summary

- Columnar data analysis attractive for DAOD_PHYSLITE
  → read all needed data efficiently using Uproot
  → represent event data model using Awkward Array
  → fast analysis in python (potentially faster than traditional analysis)

- Alternative storage formats or tuning of ROOT basket sizes might bring benefits
  → faster reading for columnar access (parquet format promising)
  → potential storage savings (for same or still faster throughput)

- First proof-of-concept of a simple analysis
  → working, but more R&D needed to cover all aspects

Backup

Event data models and awkward array

Represent event data model in a columnar fashion using awkward array

- Nested records
  e.g. `Events -> [Electrons -> pt, eta, phi, ..., Jets -> pt, eta, phi ...]`
- Behavior/Dynamic quantities
  e.g. LorentzVector - can add vectors, calculate invariant masses etc.
- Cross references via indices (`IndexedArray`)
  e.g. `Events.Electrons.trackParticles` represent each electron's track particles (via indices into the `Events.trackParticles` array)

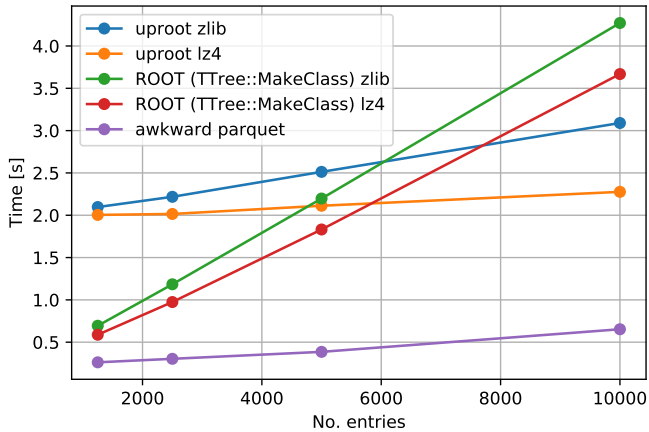
Technical aspects of this

- Class names are attached as metadata to the arrays
  → separation of data schema and behaviour
- Dynamic cross references need a reference to the top level object
  (`events.Electrons` needs to know about `events`)
- Lazy loading of columns
  → very useful for interactive working
  → using awkward's VirtualArray
- Cross references also have to work after slicing/indexing the array
  → need "global" indices

All these exist in the coffea NanoEvents module
→ working on including DAOD_PHYSLITE schema into coffea

columnar ROOT I/O for “flat” TTrees

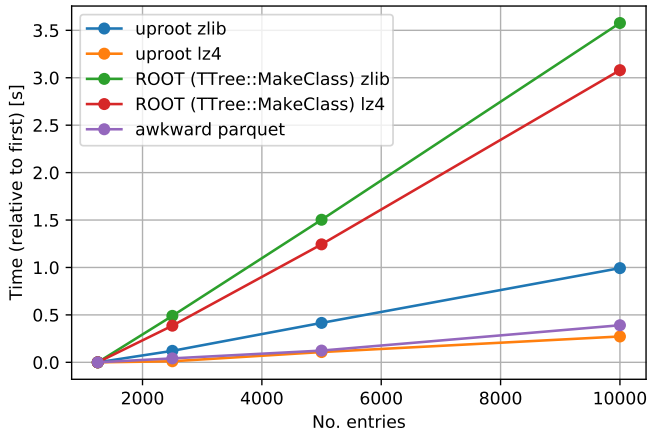
Time to read all columns



- “Flat” means branches of only fundamental types or arrays thereof (no `std::vector`)
- Comparison: EventLoop in ROOT vs. Columnar access in uproot/awkward.

columnar ROOT I/O for “flat” TTrees

Time to read all columns (relative to first measurement)



- “Flat” means branches of only fundamental types or arrays thereof (no `std::vector`)
- Comparison: EventLoop in ROOT vs. Columnar access in uproot/awkward.