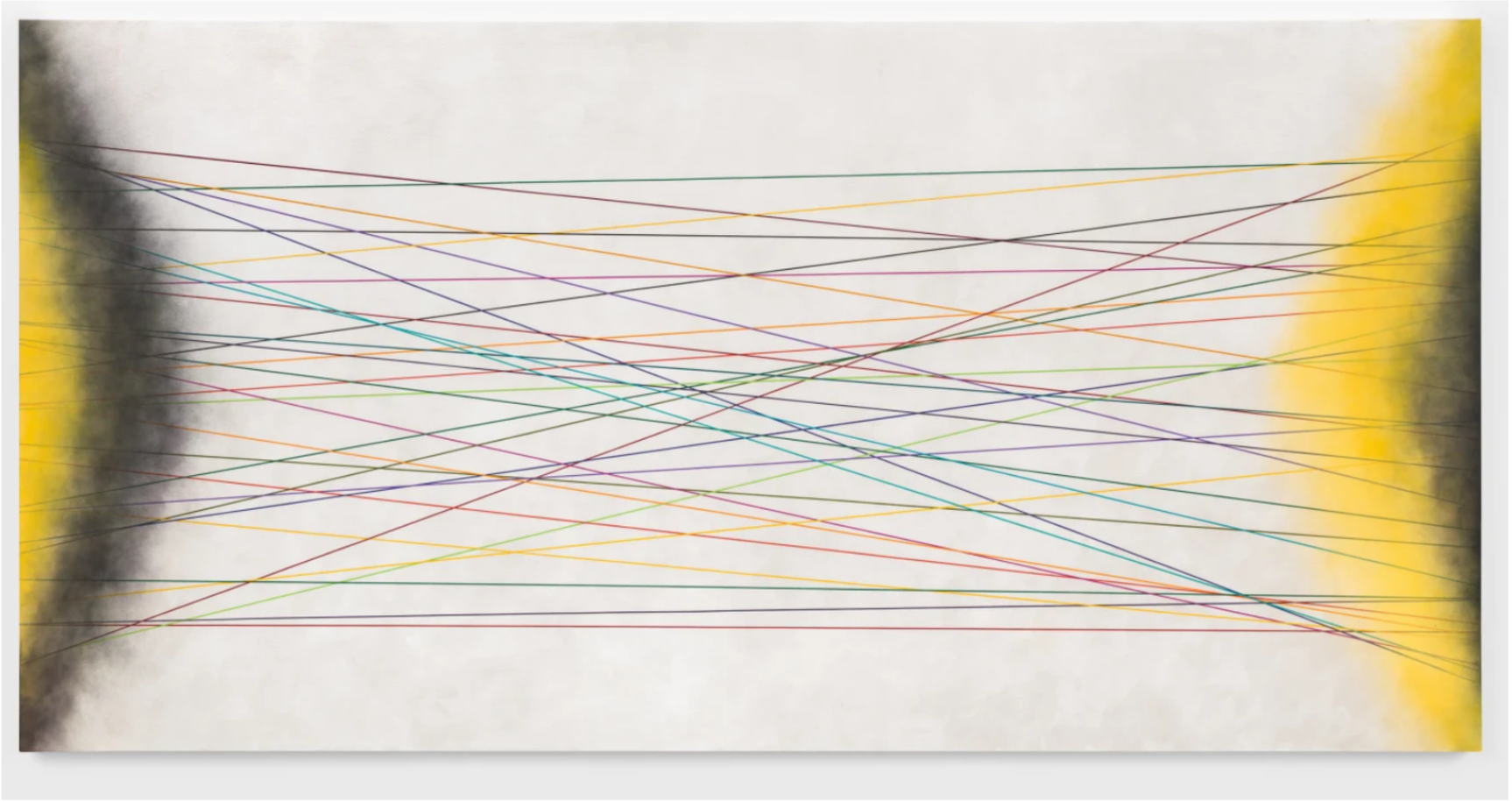
Optimization of derivation jobs and modernization of I/O integration tests for the ATLAS Experiment

March 19th, 2025



Virginia Jamarillo, “Quanta”, 2021

Jack Kraus
Advised by: Jahred Adelman (Northern Illinois University)
Serhan Mete, Peter van Gemmeren (Argonne National Labs)

An outline…

ATLAS Experiment Introduction

Toy Model Investigation into TBranch basket buffer sizes

Applying changes to derivation of real AODs

Creating new unit tests covering new I/O core functionality

Starting with:

ATLAS Experiment Introduction

The ATLAS Detector

Inner Detector (ID)

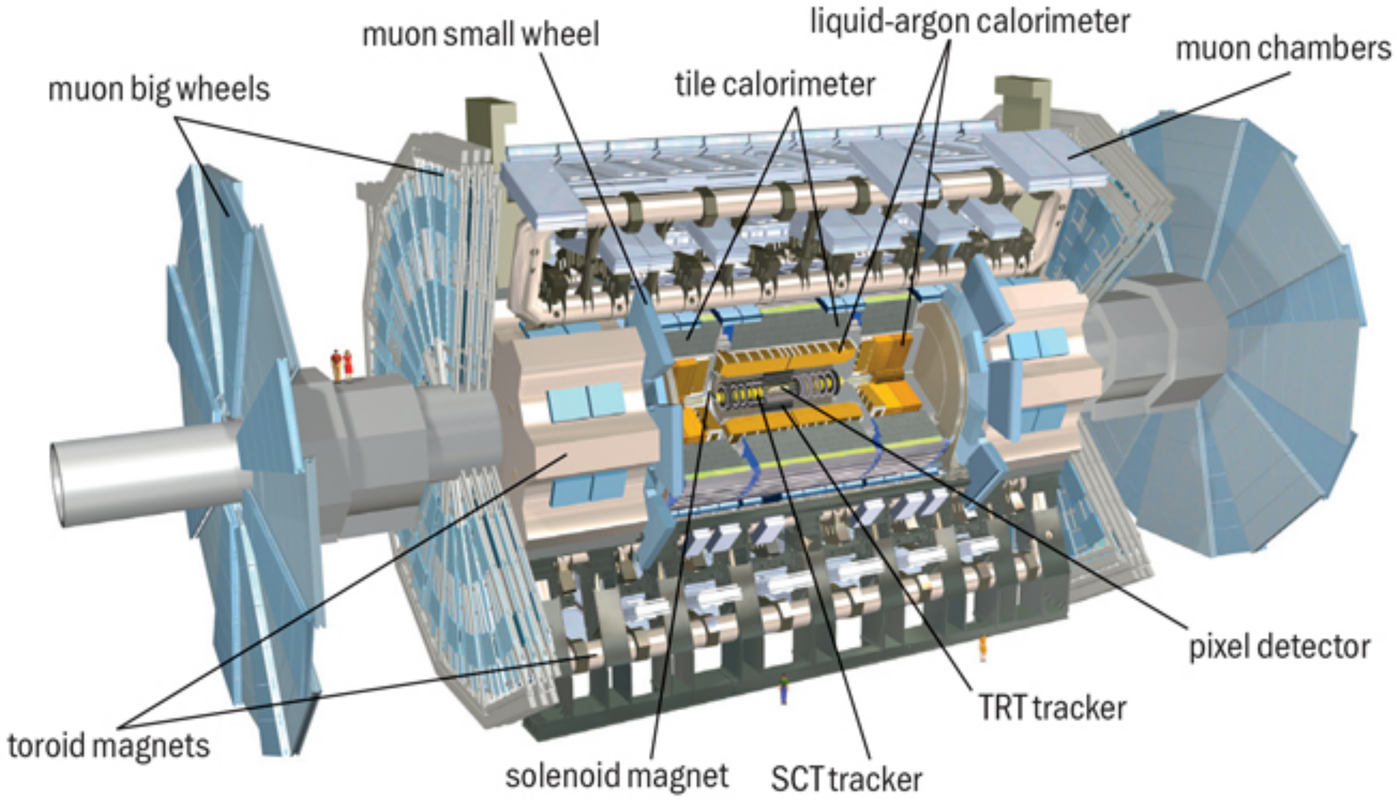
- **Pixel detector** is the first point of contact, ionizes a one-sided silicon wafer to create an electron-hole pair drift, generating an electric current.

- **SemiConductor Tracker (SCT)** uses silicon readout strips, works with the pixel detector in measuring hits to build up tracks.

- **Transition radiation tracker (TRT)** is a collection of tubes filled with a combination of gasses to discriminate between heavier and lighter particles

- Measures direction, momenta, and charge of electrically charged particles.

- Reconstruction of **tracks** can be done from the various **hits** to each part of the ID

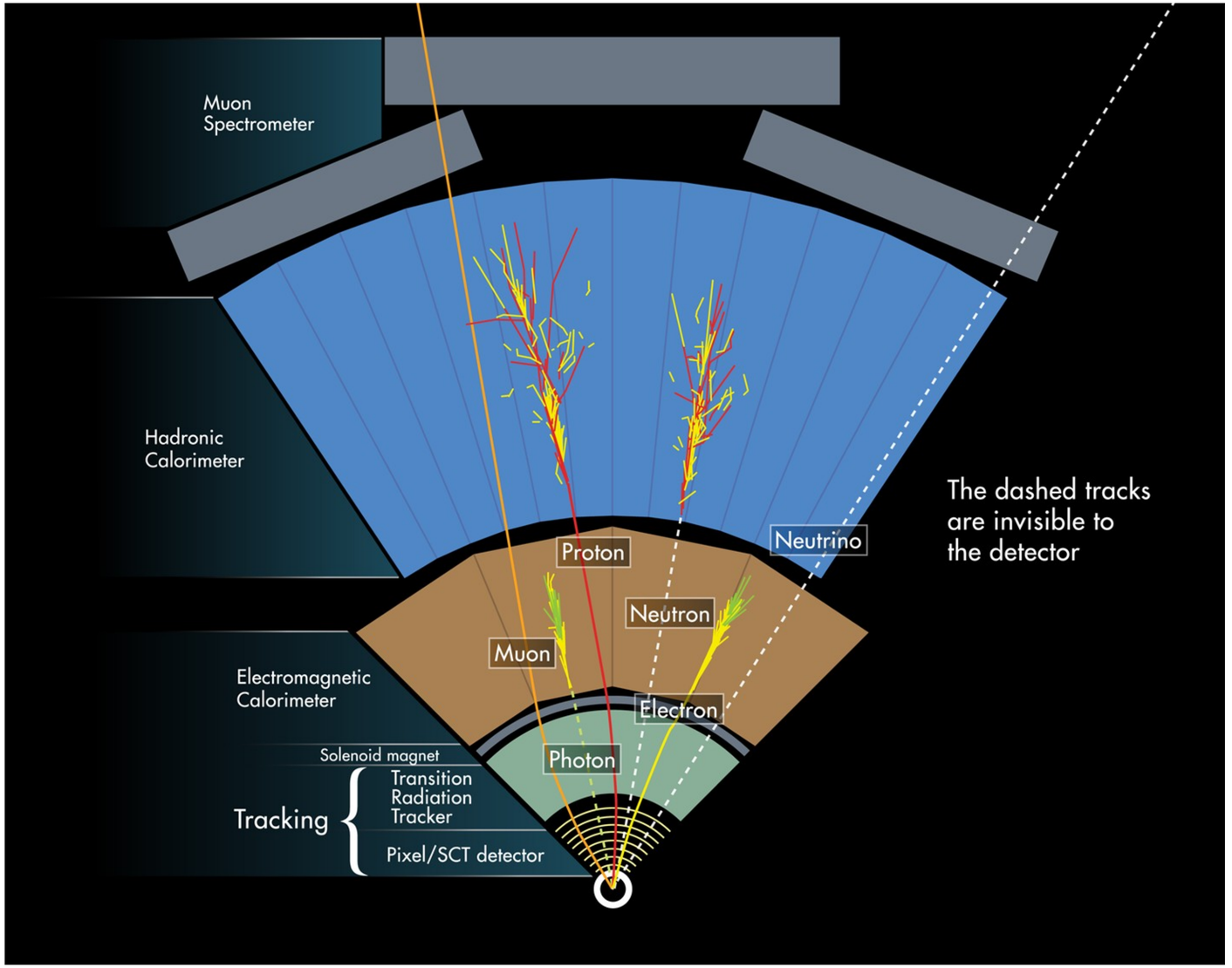


The ATLAS Detector

Calorimeters

- **Liquid Argon (LAr) calorimeter**
  - Surrounds the ID, measures the energy deposits of objects that interact via the electromagnetic force.
- **Tile calorimeter**
  - It's a hadronic calorimeter, so it interacts with particles via the strong nuclear force.
  - When a particle hits the steel, a cascade of secondary protons, neutrons and other hadrons (quark bound states) are produced.

Jets are produced here, along with reconstructed tracks in the ID, two or more tracks will be used to identify a **vertex**.



https://cds.cern.ch/images/CERN-EX-1301009-01

The ATLAS Detector

Muon Spectrometer (MS)

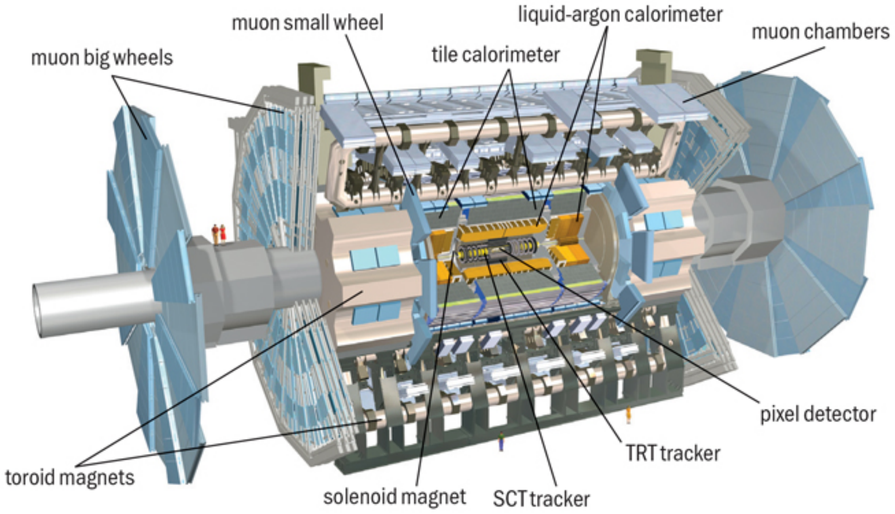
- Designed to identify muon tracks and momenta to high-res

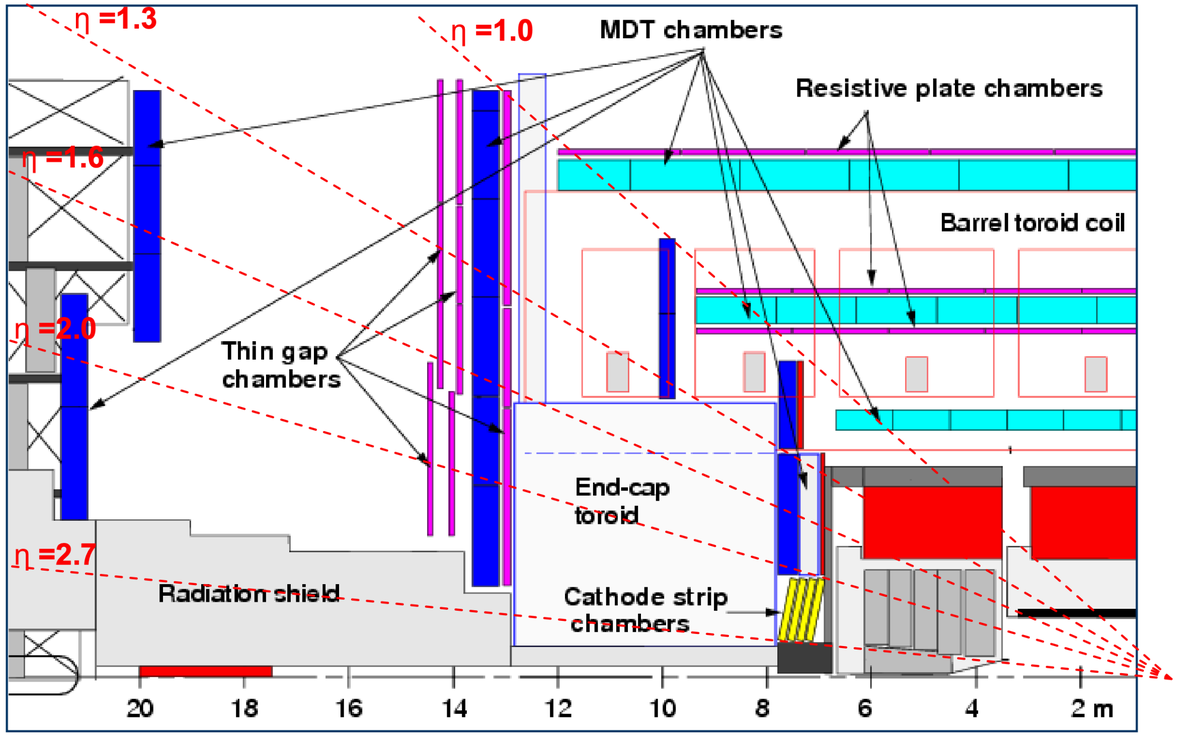
- Monitored Drift Tubes (MDT) lie on the end caps and barrel regions, covering pseudo rapidity regions $0 < |\eta| < 2.7$

(**Note**: η can be expected to be stored as a number between 0 and infinity, but the area with highest flux is between $2 < |\eta| < 2.7$)

Also, η is chosen as a coordinate because it's difference in rapidity is Lorentz invariant under boosts along the beam axis, making it easier to ID tracks due to symmetry of the collision.

$$\eta \equiv -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$$





ATLAS Trigger/Data Acquisition

~40 Million (40 MHz) pp-collision per second

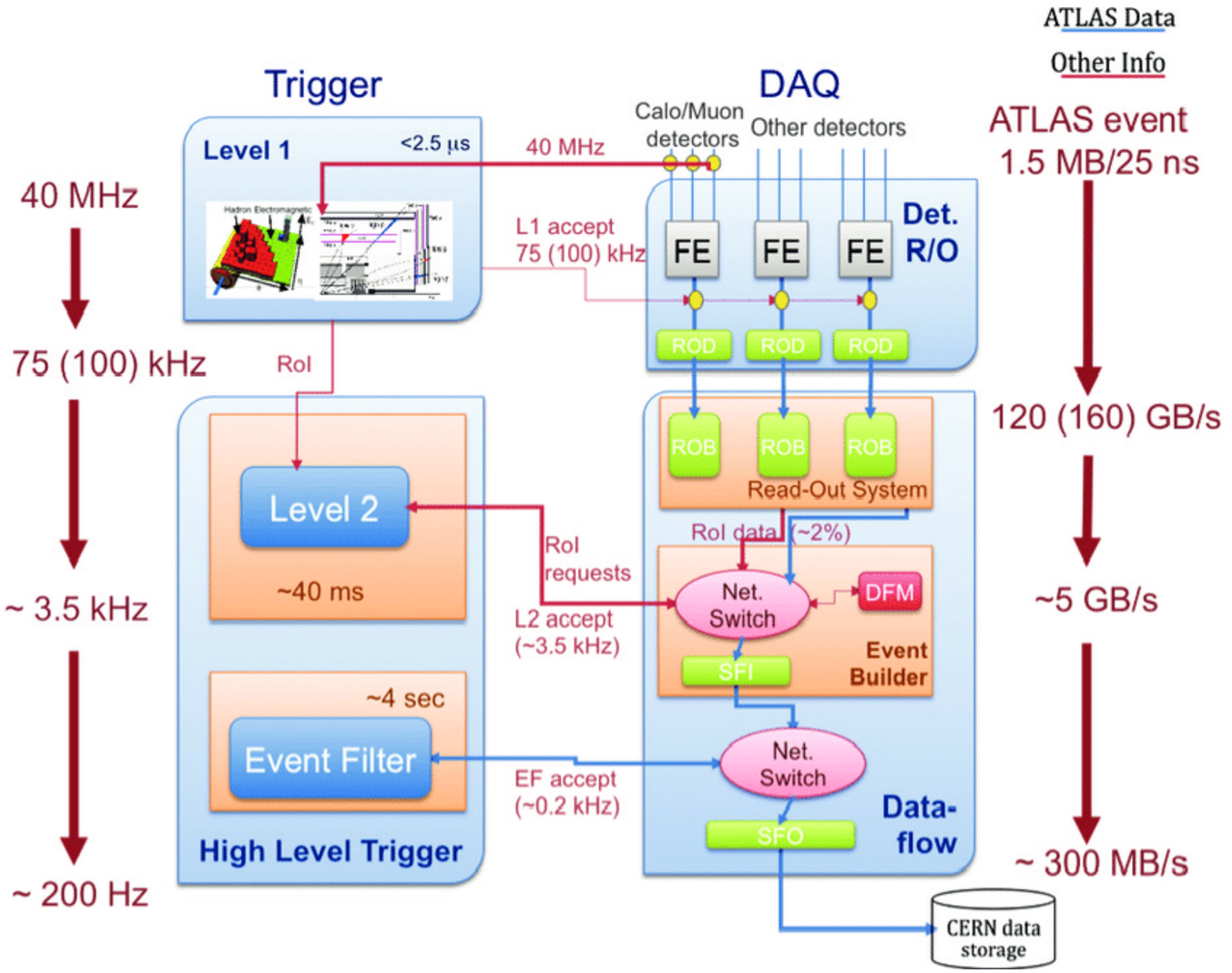
Each collision is an “event.”

- One bunch has O(10^{11}) protons, with ~2800 bunches per proton beam, each bunch is separated 25 ns apart from each other:

- 20 collisions happen at one bunch crossing

Data: ~ 25 MB / raw event, or about ~ 1 PB of raw data per second

“Pileup” is the result of multiple collisions occurring from one bunch crossing.

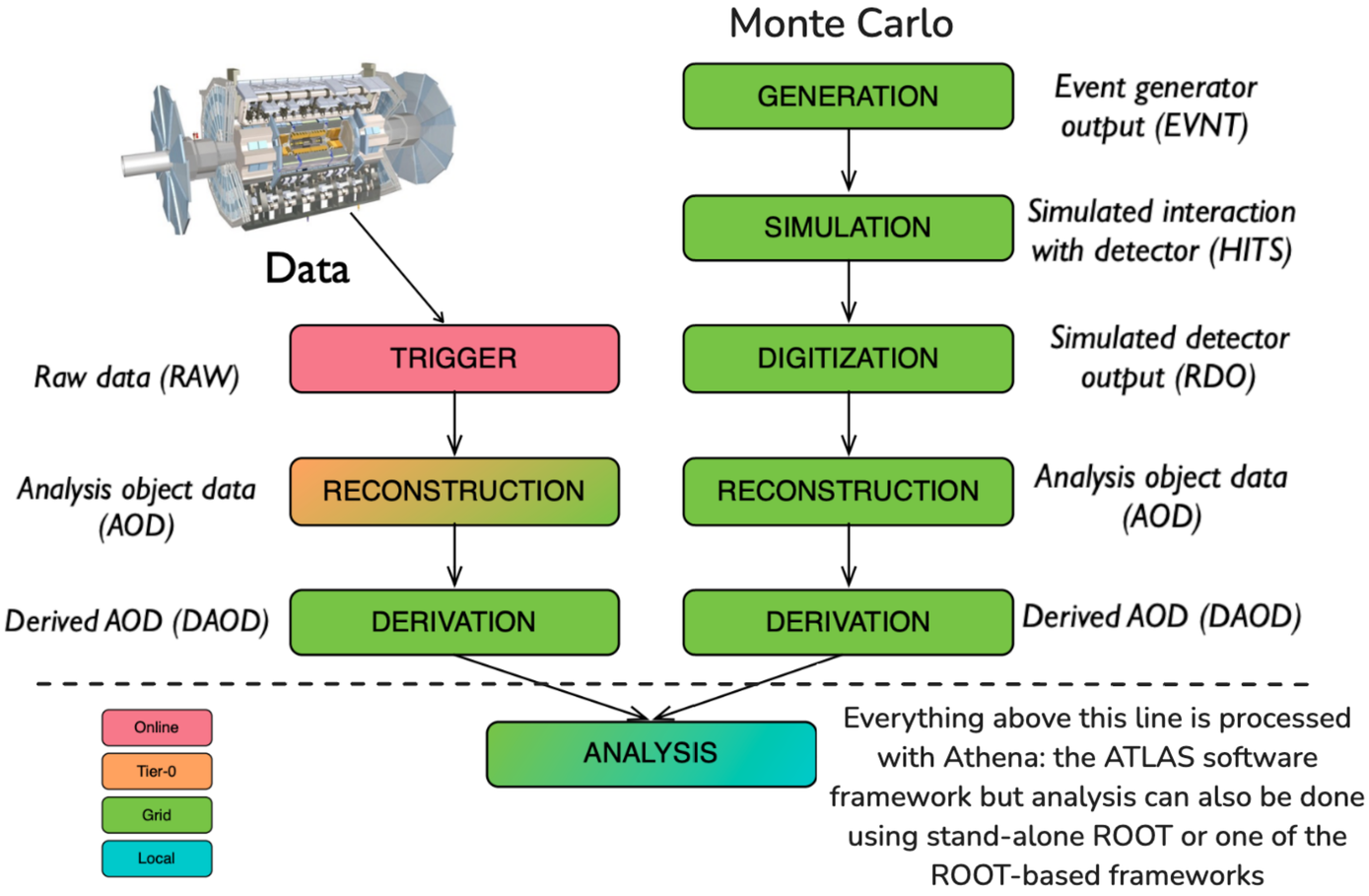


https://doi.org/10.1016/j.phpro.2012.02.503

ATLAS Analysis Pipeline

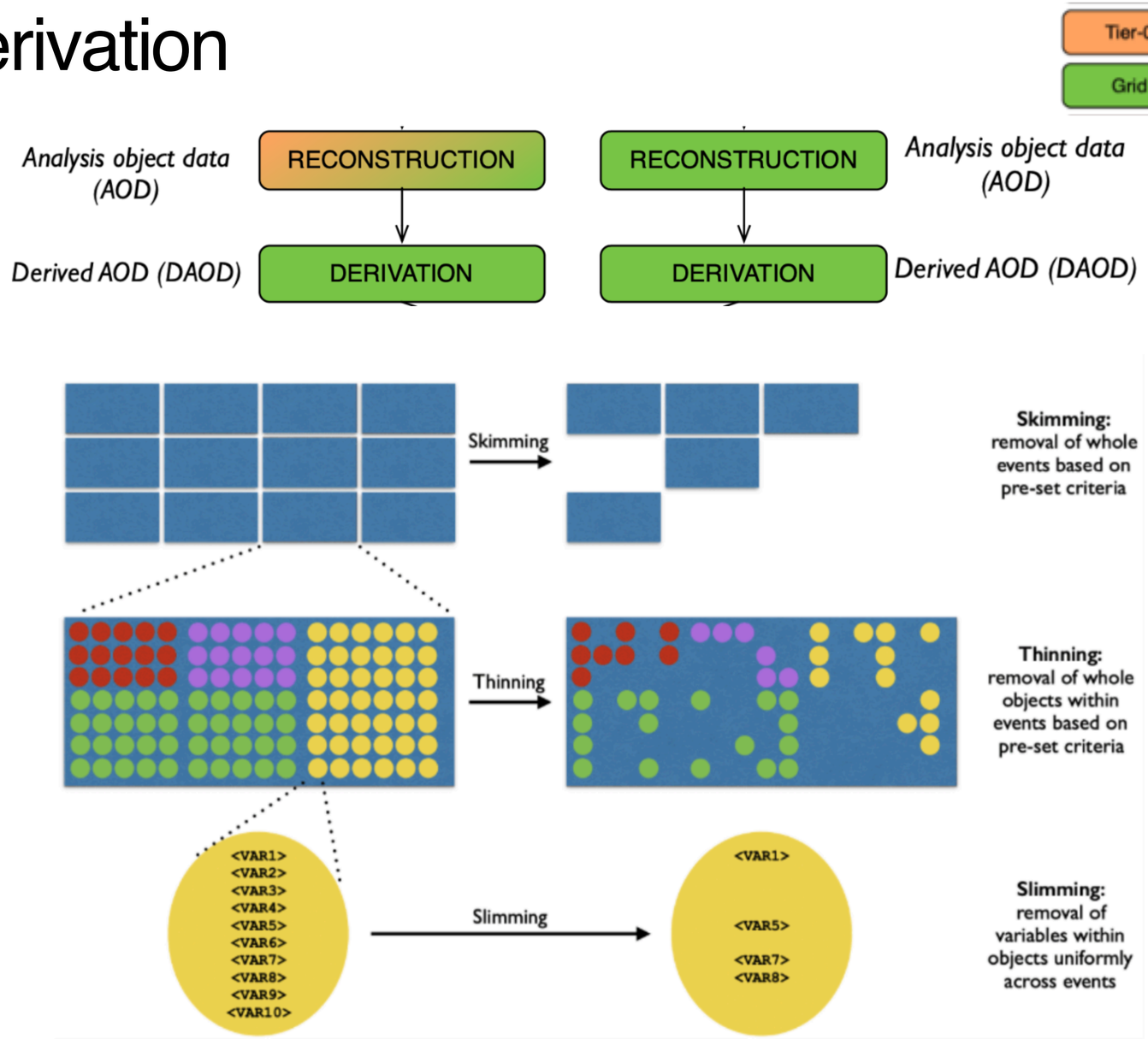
Analysis requires both real data and Monte Carlo simulated data (MC)

- Data:
  - Comes from the detector, trigger decides how interesting an event it comes from
- MC:
  - Files include more information per AOD, i.e. the full Trigger Menu, as the event generation might need to be selected for various detector configurations



ATLAS Analysis Pipeline: Derivation

- Derived Analysis Object Data (or DAOD), or a derivation, is a ROOT file used for analysis
  - From: AOD (**O(1 MB) / event**)
  - To: DAOD (**O(10 kB) / event**)
    - Main formats for ATLAS: PHYS/PHYSLITE
- Grid is used to create these files in a job, these jobs can be resource intensive (primarily memory/disk)
- Grid jobs have a 2 GB / core budget.
- Persistent storage should be minimized whenever possible.



Athena and LCG

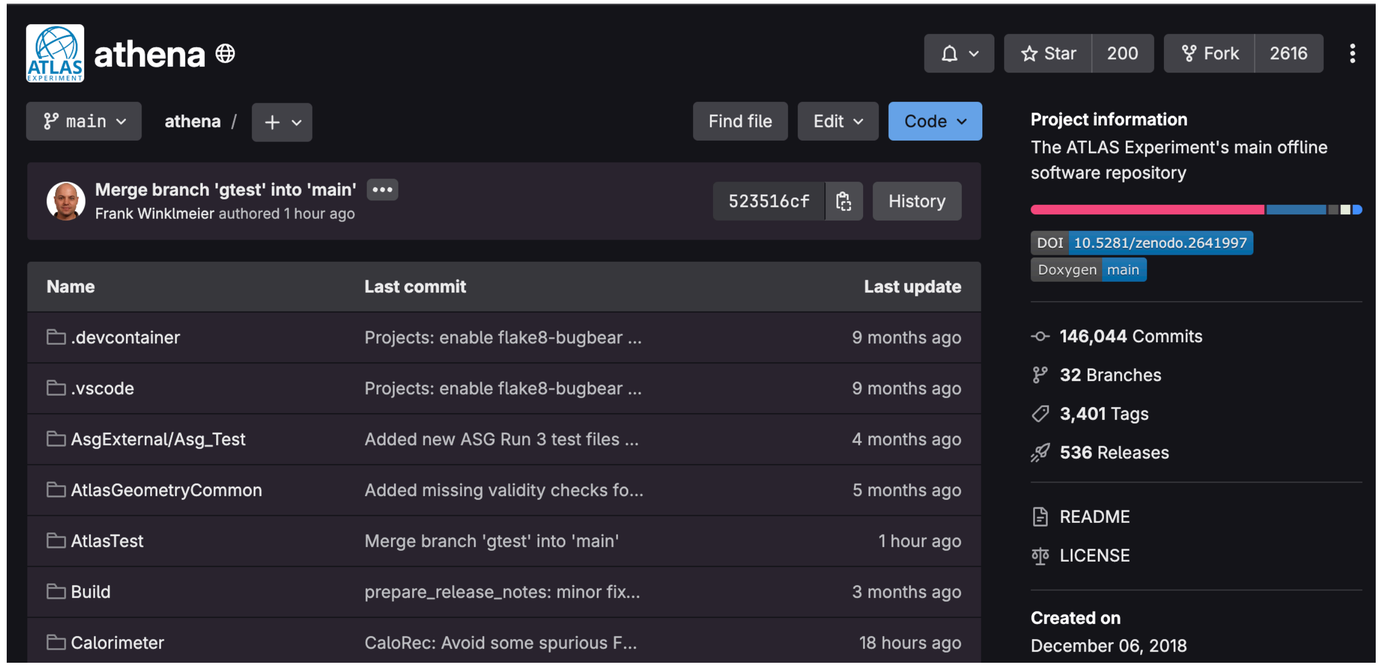
Athena is the open-source software framework for the ATLAS experiment.

- Based off of the Gaudi project, it uses ROOT and other software included in the LHC Computing Grid (LCG) software stack.

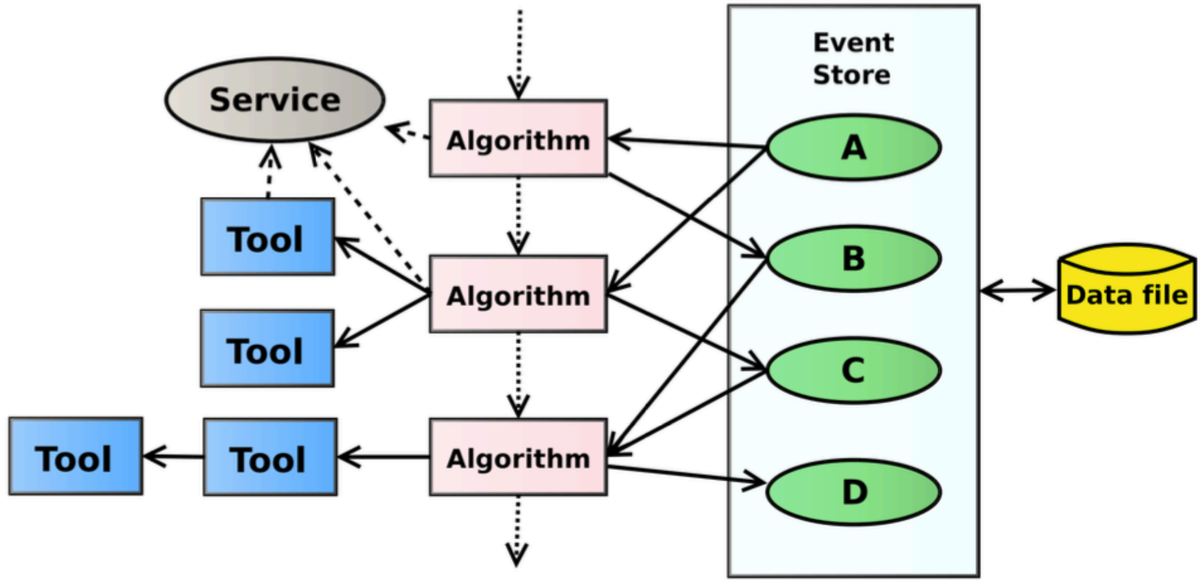
- LCG stack has a set of software frameworks that provide general solutions to LHC experiments computing needs.

~ 500 packages:

  - binary builders, compilers, language libraries, dependencies, simulation and analysis software, and more.



https://gitlab.cern.ch/atlas/athena



https://arxiv.org/abs/2404.06335

Athena Applications

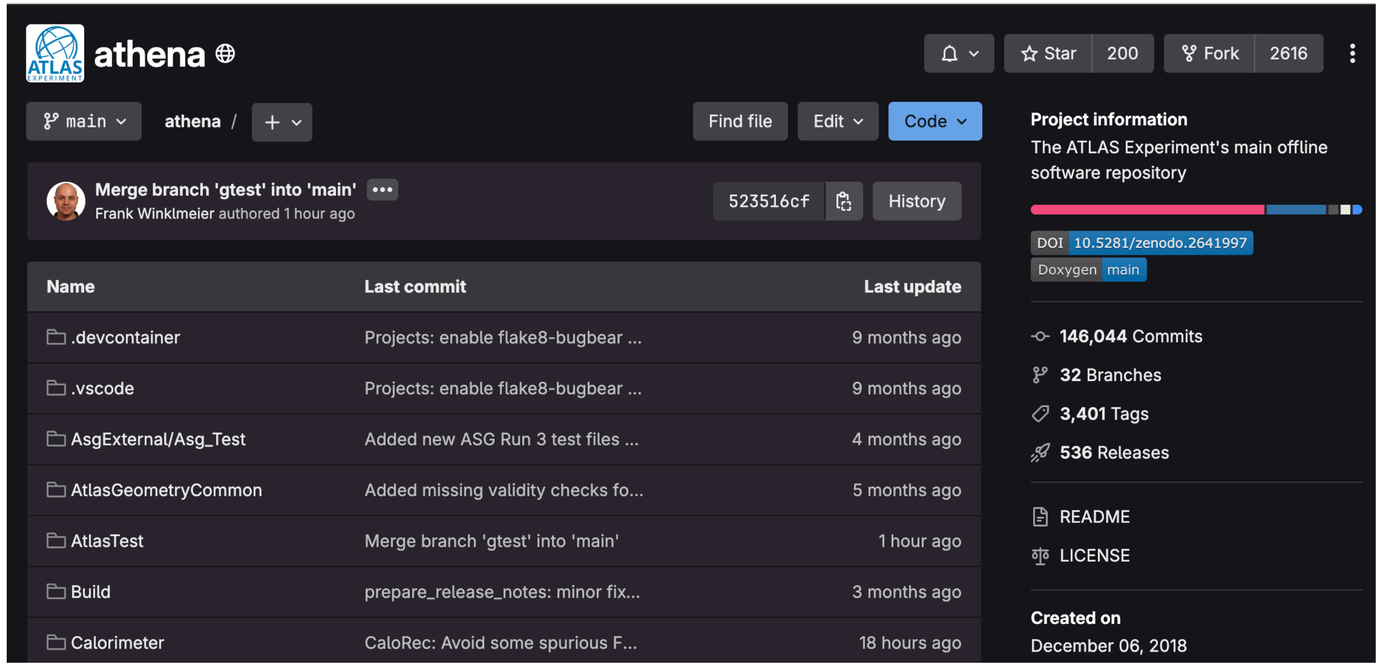
An Athena application relies on components such as Algorithms, Tools, Services, and Properties.

Python is used for the applications job steering, a high-level configuration for the direction a user may want to take the job.

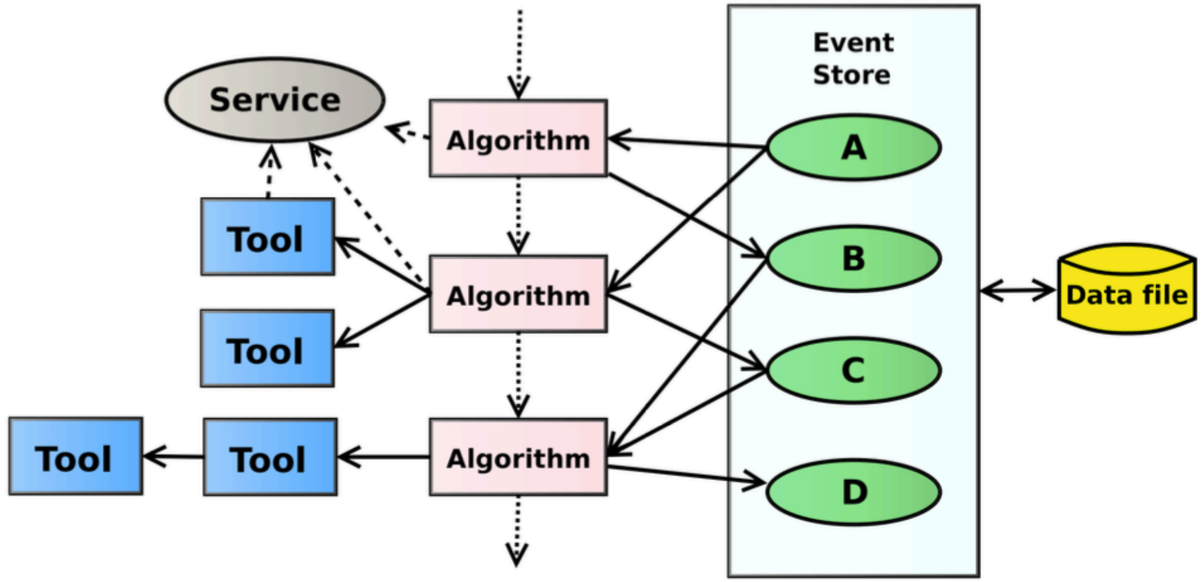
The **Event Store** is a Service

Tools can be anything Algorithms need, i.e. random number generators

All of this sits on top of ROOT, using its own classes, Tools and Services, though it primarily acts on ROOT files.



https://gitlab.cern.ch/atlas/athena



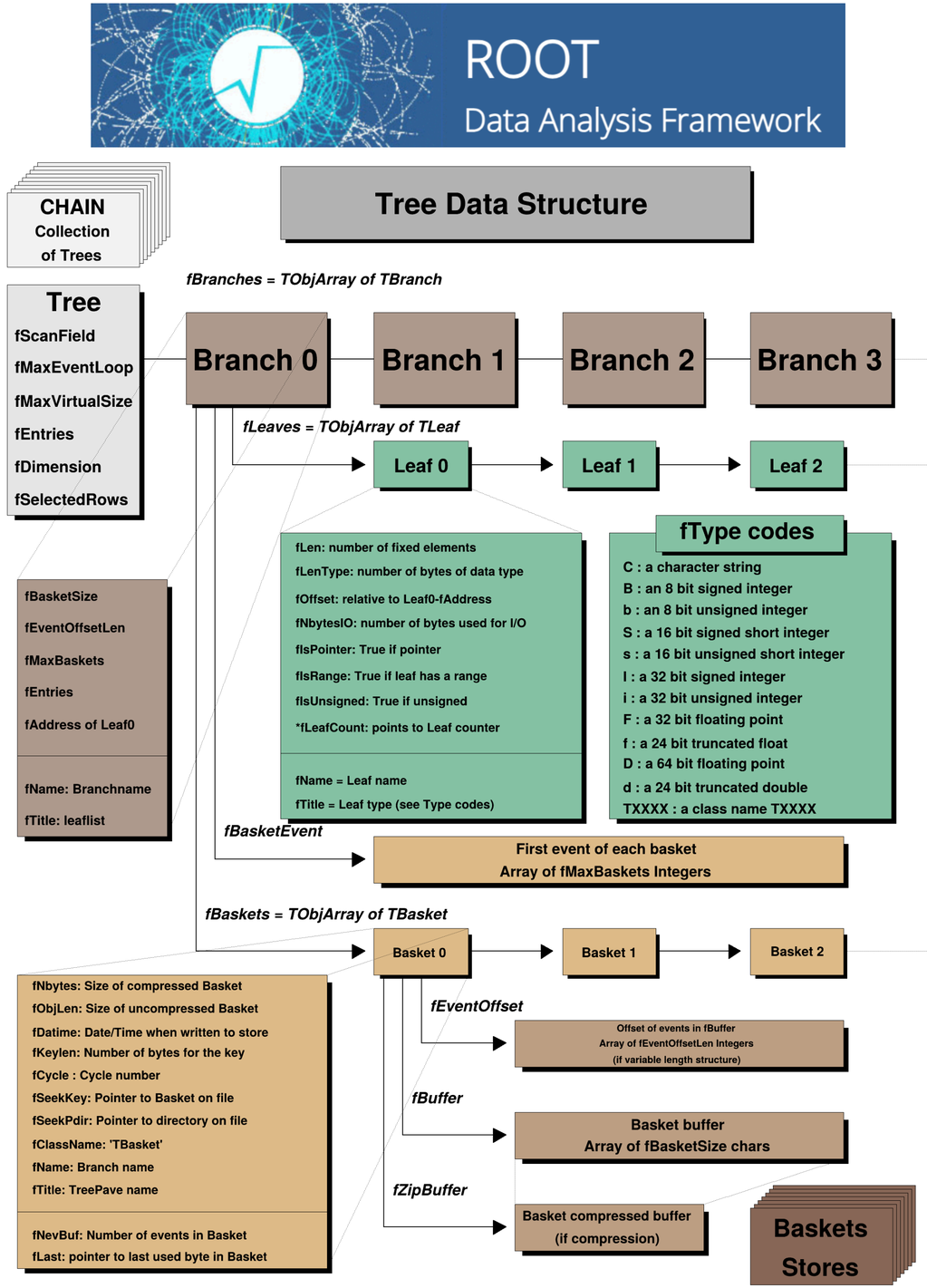
https://arxiv.org/abs/2404.06335

ROOT

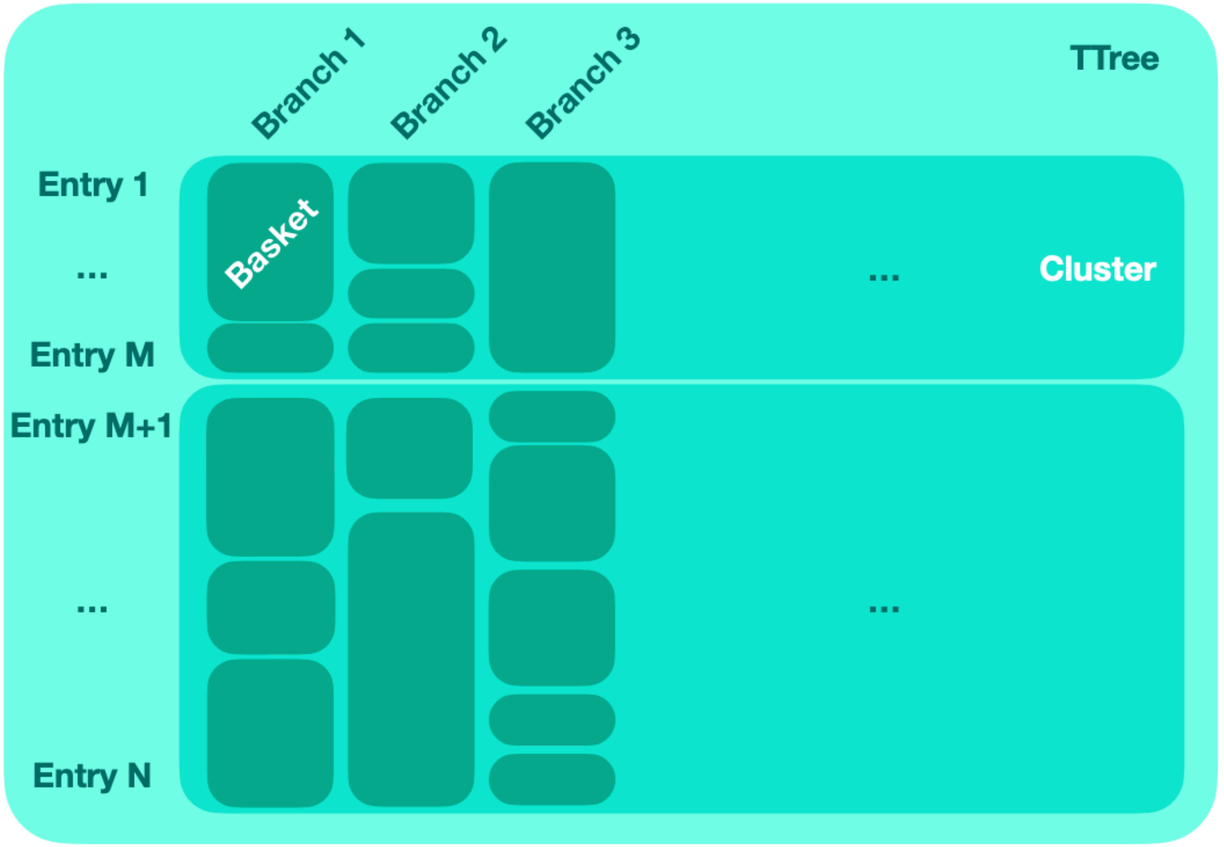
ROOT is one of the primary open-source software framework for high-energy physics analysis.

The primary structure ROOT uses is TTree, which includes a series of nested objects of varying size.

TTree is still used and will continue to be used for a while, but RNTuple is around the corner and has shown a lot of promise.



How are ROOT files organized?



DAODs are ROOT files, so they contain TTrees

TTrees have branches which can correspond to physics objects (i.e. Electrons, Muons, etc.)

Each branch stores entries into “baskets” for better I/O handling

Baskets are dedicated memory buffers that contain data which are compressed and then stored to disk

How can we reduce the load (memory/disk) to the Grid?

Currently, **ROOT during runtime**:

- Optimizes baskets after tree->Fill() is called, which assigns a basket buffer size optimized for the tree at that time. Maximum of 1.3 MB

Athena before runtime:

- Has several settings, but the two we're looking at are:
  Maximum Basket Buffer Size: 128 kB
  Minimum Number of Entries to the Buffer: 10 entries

For both ROOT and Athena: What happens to performance when we fix the maximum buffer limit **prior** to runtime memory allocation for the job occurs?

Now we start:

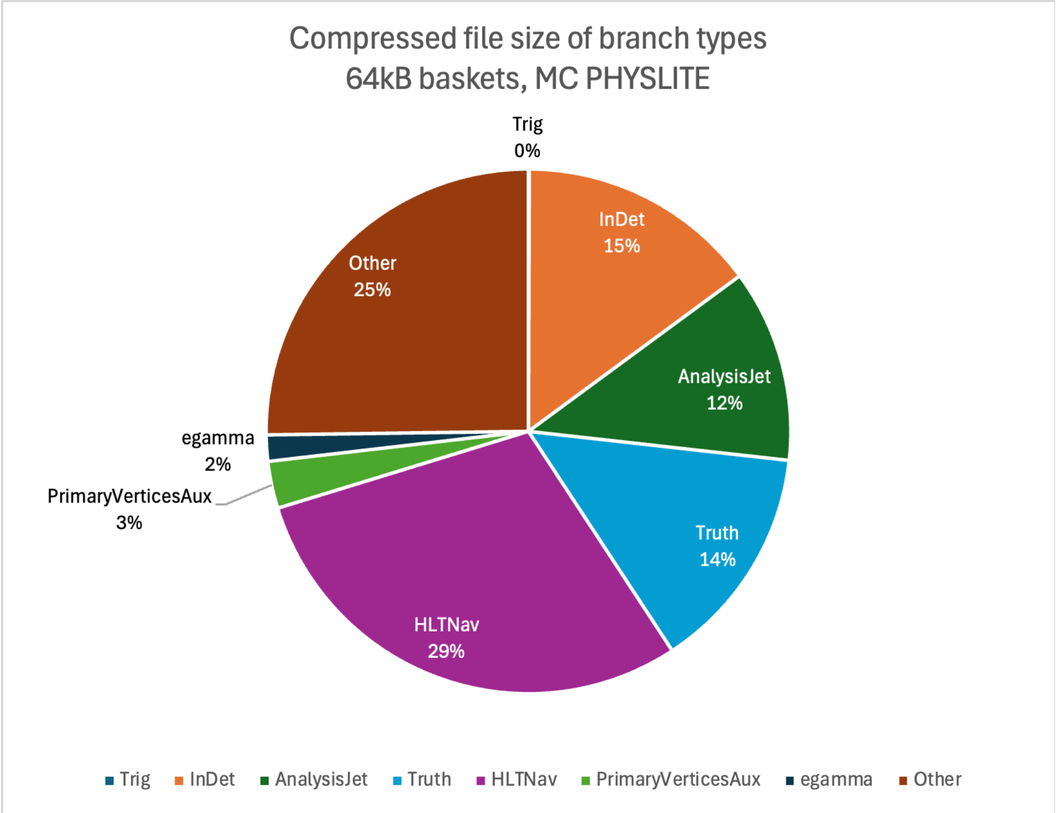
Toy Model Investigation into TBranch basket buffer sizes

Toy Model I/O Tests: What to look out for

Can we mimic what will be put into branches?

What we know:
- MC AODs are typically larger than Data AODs (i.e. Full TriggerMenu for MC)
- Branch data consists of things like:
  - Charges (Electrons: q = -1, Muons: q=-1, etc)
  - pT (best represented by floating point numbers)
  - η values (range from 0 -> Infinity, but majority are in that $2 < |\eta| < 2.7$ range)
  - ϕ values

Given an example MC derivation we might expect to see an average compression ratio of about ~5.5



Tag	Compressed File Size (MB) (out of 2.5 GB)
Trig	1.13
InDet	336.94
AnalysisJet	269.24
Truth	317.48
HLTNav	669.33
PrimaryVerticesAux	65.81
egamma	36.68
Other	572.08

Toy Model I/O Tests: What to look out for

	branch-name	tag	num-of-entries	basket-size	total-file-size	compressed-file-size	compression-ratio
891	TruthForwardProtonsAuxDyn.classifierParticleOutCome	Truth	140000	10752	2034540	403401	5.04
892	TruthForwardProtonsAuxDyn.polarizationPhi	Truth	140000	10752	2029090	397102	5.11
893	TruthForwardProtonsAuxDyn.polarizationTheta	Truth	140000	10752	2030180	400159	5.07
894							
895						AVG:	5.77

Just one example

Tag	Compressed File Size (MB)
Trig	1.13
InDet	336.94
AnalysisJet	269.24
Truth	317.48
HLTNav	669.33
PrimaryVerticesAux	65.81
egamma	36.68
Other	572.08

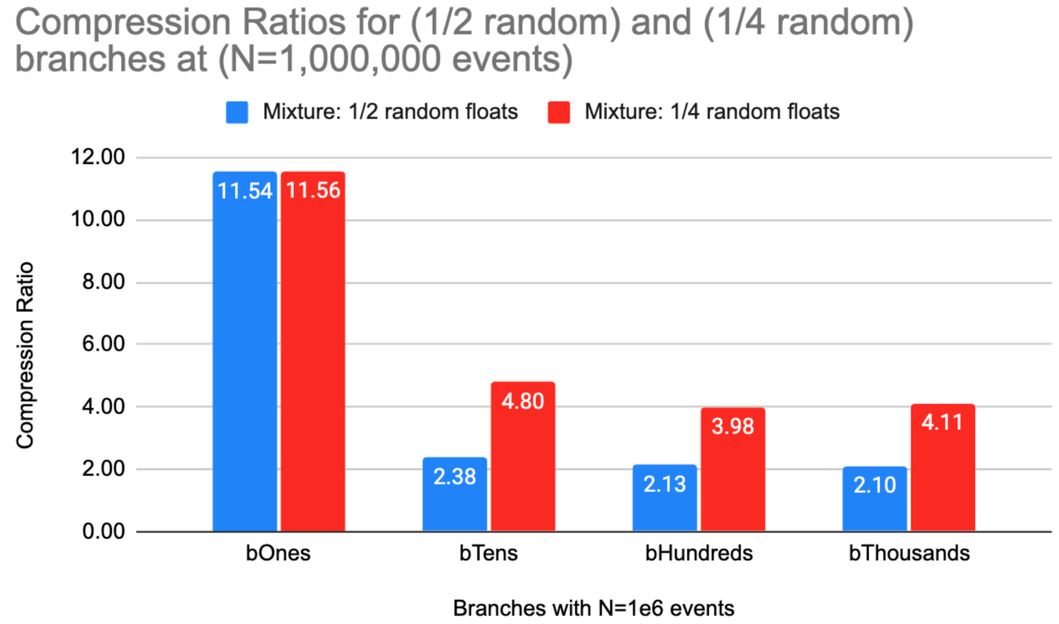
Toy Model I/O Tests

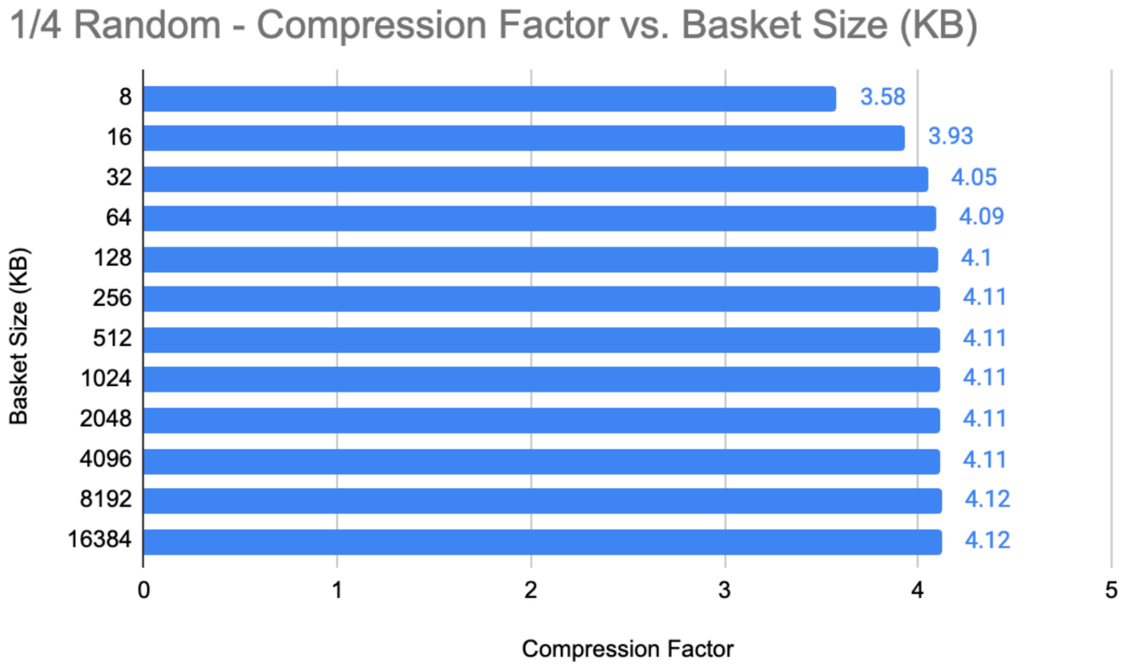
Early toy model derivations test the role of basket sizes with compression factors

For any toy model where branches are:

- Filled with some fraction of (random floats & the same constant integer)
- Mimic what data looks like (mixed bag of easy/ hard compressible data)

We do see that increasing basket size does not scale the compression linearly



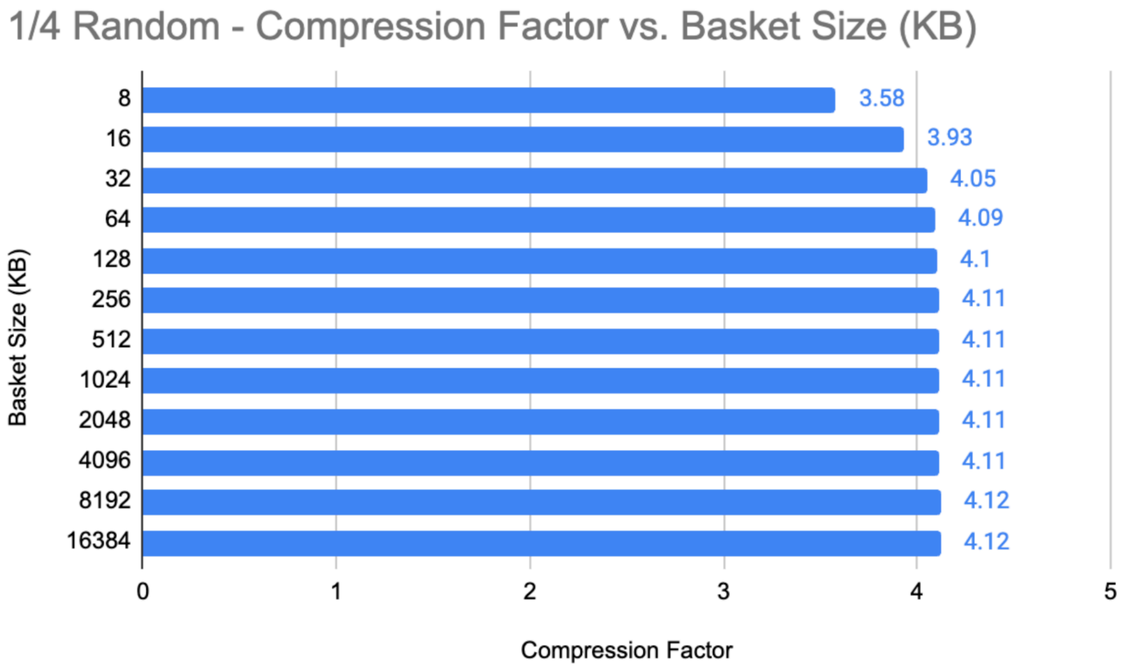


Toy Model I/O Tests

During the writing of these branches, the setting was done on within the script that filled the TTree full of values with:

```
tree->SetBasketSize(*, 8000) // 8 kB
```

But this is done during runtime… What if we wanted to set or remove the basket buffer size beforehand?



How can we reduce the load (memory/disk) to the Grid?

Currently, **ROOT during runtime**:

- Optimizes baskets after tree->Fill() is called, which assigns a basket buffer size optimized for the tree at that time. Maximum of 1.3 MB

Athena before runtime:

- Has several settings, but the two we're looking at are:
  Maximum Basket Buffer Size: 128 kB
  Minimum Number of Entries to the Buffer: 10 entries

For both ROOT and Athena: What happens to performance when we fix the maximum buffer limit **prior** to runtime memory allocation for the job occurs?

For this we'll need to run real derivation jobs…

Then we can get to:

Applying changes to derivation of real AODs

After running derivation jobs: `128k-Basket / min-entries`

Key [+] [−] = `128kB_basket` present [not present]
[+] [−] = `min_entries=10` present [not present]

Configs	Athena v22.0.16 Data Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
	PHYS + PHYSLITE	**PHYS**	**PHYSLITE**
[+] [+]	27.11	3.216	1.034
[−] [−]	27.81 [+ 2.5]	3.222 [+ 0.2]	1.036 [+ 0.2]
[+] [−]	27.81 [+ 2.5]	3.216 [- 0.0]	1.030 [- 0.4]
[−] [+]	27.30 [+ 0.7]	3.221 [+ 0.2]	1.042 [+ 0.7]

Configs	Athena v22.0.16 MC Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
	PHYS + PHYSLITE	**PHYS**	**PHYSLITE**
[+] [+]	14.1	5.8	2.59
[−] [−]	16.1 [+ 12.1]	6.0 [+ 2.9]	2.7 [+ 5.1]
[+] [−]	16.0 [+ 11.5]	5.7 [- 2.8]	2.5 [- 5.6]
[−] [+]	14.2 [+ 0.4]	6.2 [+ 5.4]	2.9 [+ 9.9]

Default configuration

Testing: `128k-Basket / min-entries`

Key (+) (−) = `128kB_basket` present [not present]

(+) (−) = `min_entries=10` present [not present]

Configs	Athena v22.0.16 Data Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
	PHYS + PHYSLITE	**PHYS**	**PHYSLITE**
(+) (+)	27.11	3.216	1.034
(−) (−)	27.81 [+ 2.5]	3.222 [+ 0.2]	1.036 [+ 0.2]
(−) (+)	27.81 [+ 2.5]	3.216 [- 0.0]	1.030 [- 0.4]
(+) (−)	27.30 [+ 0.7]	3.221 [+ 0.2]	1.042 [+ 0.7]

Configs	Athena v22.0.16 MC Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
	PHYS + PHYSLITE	**PHYS**	**PHYSLITE**
(+) (+)	14.1	5.8	2.59
(−) (−)	16.1 [+ 12.1]	6.0 [+ 2.9]	2.7 [+ 5.1]
(−) (+)	16.0 [+ 11.5]	5.7 [- 2.8]	2.5 [- 5.6]
(+) (−)	14.2 [+ 0.4]	6.2 [+ 5.4]	2.9 [+ 9.9]

- Data derivation jobs see little effect. Except a bit of memory increase when removing the basket limit.

Testing: 128k-Basket / min-entries

Key		
	+	= 128kB_basket present [not present]
	+	= min_entries=10 present [not present]

	Athena v22.0.16 Data Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
Configs	**PHYS + PHYSLITE**	**PHYS**	**PHYSLITE**
+ +	27.11	3.216	1.034
− −	27.81 [+ 2.5]	3.222 [+ 0.2]	1.036 [+ 0.2]
+ −	27.81 [+ 2.5]	3.216 [- 0.0]	1.030 [- 0.4]
− +	27.30 [+ 0.7]	3.221 [+ 0.2]	1.042 [+ 0.7]

	Athena v22.0.16 MC Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
Configs	**PHYS + PHYSLITE**	**PHYS**	**PHYSLITE**
+ +	14.1	5.8	2.59
− −	16.1 [+ 12.1]	6.0 [+ 2.9]	2.7 [+ 5.1]
− +	16.0 [+ 11.5]	5.7 [- 2.8]	2.5 [- 5.6]
+ −	14.2 [+ 0.4]	6.2 [+ 5.4]	2.9 [+ 9.9]

- MC derivation sees a much larger effect

Reduction in the output file-size by (~5.6%) but at the price of much higher memory usage (> 11% for MC derivation)

Removing the basket cap in ROOT doesn't make sense, cannot assume every ROOT user can afford the memory increase this large. Ignore ROOT

Testing: 128k-Basket / min-entries

Key		
	+	= 128kB_basket present [not present]
	+	= min_entries=10 present [not present]

	Athena v22.0.16 Data Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
Configs	**PHYS + PHYSLITE**	**PHYS**	**PHYSLITE**
+ +	27.11	3.216	1.034
− −	27.81 [+ 2.5]	3.222 [+ 0.2]	1.036 [+ 0.2]
+ −	27.81 [+ 2.5]	3.216 [- 0.0]	1.030 [- 0.4]
− +	27.30 [+ 0.7]	3.221 [+ 0.2]	1.042 [+ 0.7]

	Athena v22.0.16 MC Derivations		
	Max PSS (GB) [Δ% default]	Output File Size (GB) [Δ%]	
Configs	**PHYS + PHYSLITE**	**PHYS**	**PHYSLITE**
+ +	14.1	5.8	2.59
− −	16.1 [+ 12.1]	6.0 [+ 2.9]	2.7 [+ 5.1]
− +	16.0 [+ 11.5]	5.7 [- 2.8]	2.5 [- 5.6]
+ −	14.2 [+ 0.4]	6.2 [+ 5.4]	2.9 [+ 9.9]

- MC derivation sees a much larger effect

When both features were first implemented, it just so happened to have the opposite effect—**hit a happy medium! Keep Both!**

When only changing Athena: just how good is choosing 128 kB as a basket limit?

MC Derivation by Altered Basket Sizes			
Basket sizes	**MaxPSS (GB) [Δ% default]**	**PHYS Size (GB) [Δ%]**	**PHYSLITE Size (GB) [Δ%]**
(**default**) 128k_basket	15.0	5.88	2.59
512k_basket	16.4 [+ 8.6]	5.74 [- 2.5]	2.46 [- 5.1]
no_limit	16.9 [+ 11.3]	5.72 [- 2.8]	2.45 [- 5.6]

Pretty good!

Any larger and we immediately start heading toward high-memory usage territory. Need to stay under budget for these jobs.

Interesting branches [MC PHYSLITE DAOD]

MC PHYS-LITE branch: HLTNav_Summary_DAODSlimmedAuxDyn.decisions					
Configs	Baskets	Basket Size (KB)	Total Size (MB)	File Size (MB)	Compression
128kB_basket	11211	128	1862.8	417.0	4.47
no_limit	5223	1293.5	1861.9	342.4	5.44

Again: Basket size beyond 128kB doesn't massively improve compression [See Toy Model]

Interesting Branches	Basket size (kB)	Total-file-size (MB)	Compressed-file-size (MB)	Compression
PrimaryVerticesAuxDyn.trackParticleLinks	1293.5	2145.5	22.9	93.5
HLTNav_Summary_DAODSlimmedAuxDyn.decisions	1293.5	1861.8	342.4	5.44
HardScatterVerticesAuxDyn.incomingParticleLinks	693.0	118.5	1.3	90.1
HardScatterVerticesAuxDyn.outgoingParticleLinks	635.5	108.5	1.5	74.0
AnalysisTauJetsAuxDyn.tauTrackLinks	447.0	74.9	1.9	39.2
	Max = 1293.5 kB = 1.3 MB			

Interesting branches [MC PHYSLITE DAOD (no-lim)]

Contains mostly repeated numbers, highly compressible

Interesting Branches	Basket size (kB)	Total-file-size (MB)	Compressed-file-size (MB)	Compression
PrimaryVerticesAuxDyn.trackParticleLinks	1293.5	2145.5	22.9	93.5
HLTNav_Summary_DAODSlimmedAuxDyn.decisions	1293.5	1861.8	342.4	5.44
HardScatterVerticesAuxDyn.incomingParticleLinks	693.0	118.5	1.3	90.1
HardScatterVerticesAuxDyn.outgoingParticleLinks	635.5	108.5	1.5	74.0
AnalysisTauJetsAuxDyn.tauTrackLinks	447.0	74.9	1.9	39.2
	Max = 1293.5 kB = 1.3 MB			

Poorly compressed, Large file, reaches ROOT basket limit!

Other highly compressible branches

Conclusion to branch buffer size studies:

Removing the basket limit, a greater decrease in DAOD output file size is achieved at the expense of greater memory usage. However, what we're currently doing is good, which wasn't explored in detail before.

Finding these highly compressible branches leads us to want to look further into these branches to see what can be done to further optimize beyond derivation production.

Finally

Creating new unit tests covering new I/O core functionality

Transient/Persistent EDM and the xAOD EDM

Transient/Persistent (T/P):

- Transient data is located in memory and is readily accessible to the user.

- While Persistent data is saved to disk and need to be converted into transient data before it can be accessible by the user.

Some problems:

- Transient data would often gain in complexity the more information attached to it, and ROOT had a hard time dealing with this complexity.

- Developers would have to convert data samples from the full EDM to simpler ones (more readily available to ROOT) which would lead to duplication of data.

Transient/Persistent EDM and the xAOD EDM

Transient/Persistent (T/P):

- Transient data is located in memory and is readily accessible to the user.

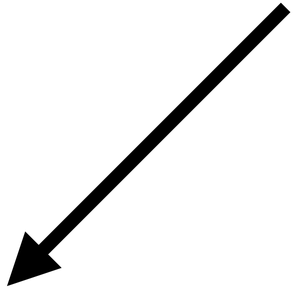
- While Persistent data is saved to disk and need to be converted into transient data before it can be accessible by the user.

Some problems:

- Transient data would often gain in complexity the more information attached to it, and ROOT had a hard time dealing with this complexity.

- Developers would have to convert data samples from the full EDM to simpler ones (more readily available to ROOT) which would lead to duplication of data.

(From what I could find, they put ‘x’ in-front of the AOD to distinguish it as the new AOD, though I can’t find the source)



xAOD EDM:

- Seeks to simplify things further in to interface objects and payload objects.

Transient/Persistent EDM and the xAOD EDM

Transient/Persistent (T/P):

- Transient data is located in memory and is readily accessible to the user.

- While Persistent data is saved to disk and need to be converted into transient data before it can be accessible by the user.

Some problems:

- Transient data would often gain in complexity the more information attached to it, and ROOT had a hard time dealing with this complexity.

- Developers would have to convert data samples from the full EDM to simpler ones (more readily available to ROOT) which would lead to duplication of data.

xAOD EDM:

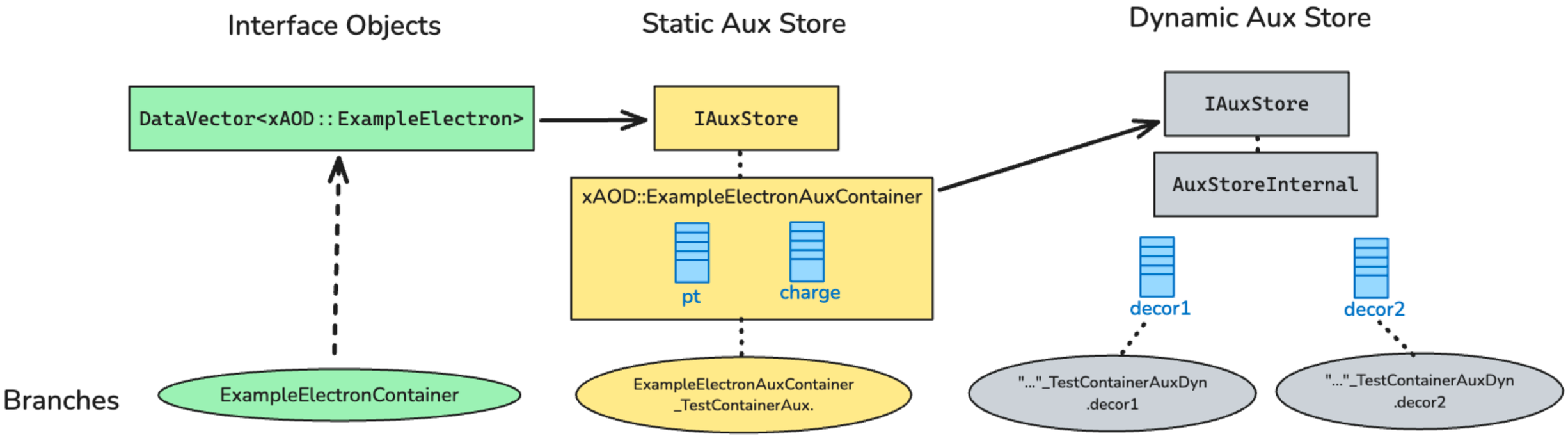
- Seeks to simplify things further in to interface objects and payload objects.

- Interfaces allow the user to delay loading data into memory

- While the payload objects contain the data for the interface objects and allocate contiguous blocks of memory. Payload objects are synonymous with auxiliary storage.

Note: xAOD EDM was adopted during Run 2, so it's not *that* new—and T/P EDM is still used for certain things.

Transient/Persistent EDM and the xAOD EDM



xAOD EDM:

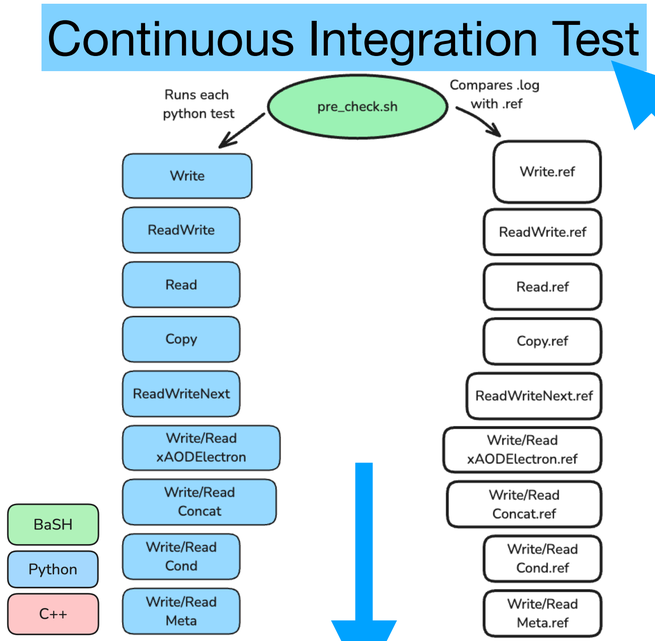
- Seeks to simplify things further in to interface objects and payload objects.

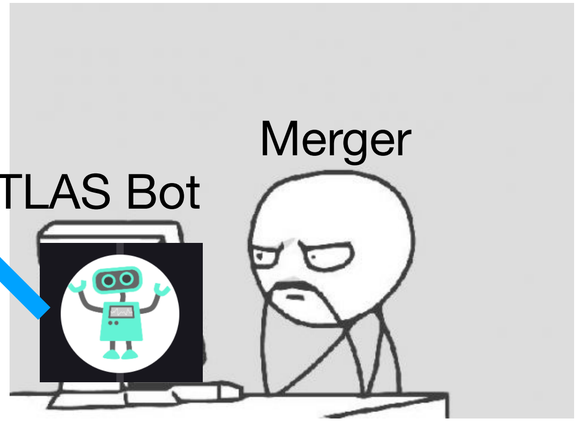
- Allows for decorations which can be readily accessed by ROOT and don't have as much of the complexity problem.

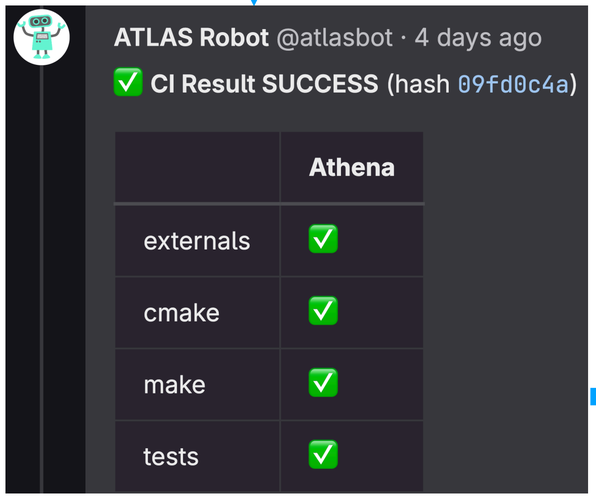
Continuous Integration Testing

- Athena is updated every night (a **nightly**) and built during continuous integration, but each time it's built it needs to reference unit tests to ensure safety of core features

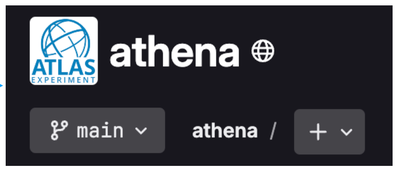
- Unit tests can test core features across many areas, but the package we're focused on is core I/O functionality in AthenaPoolExample





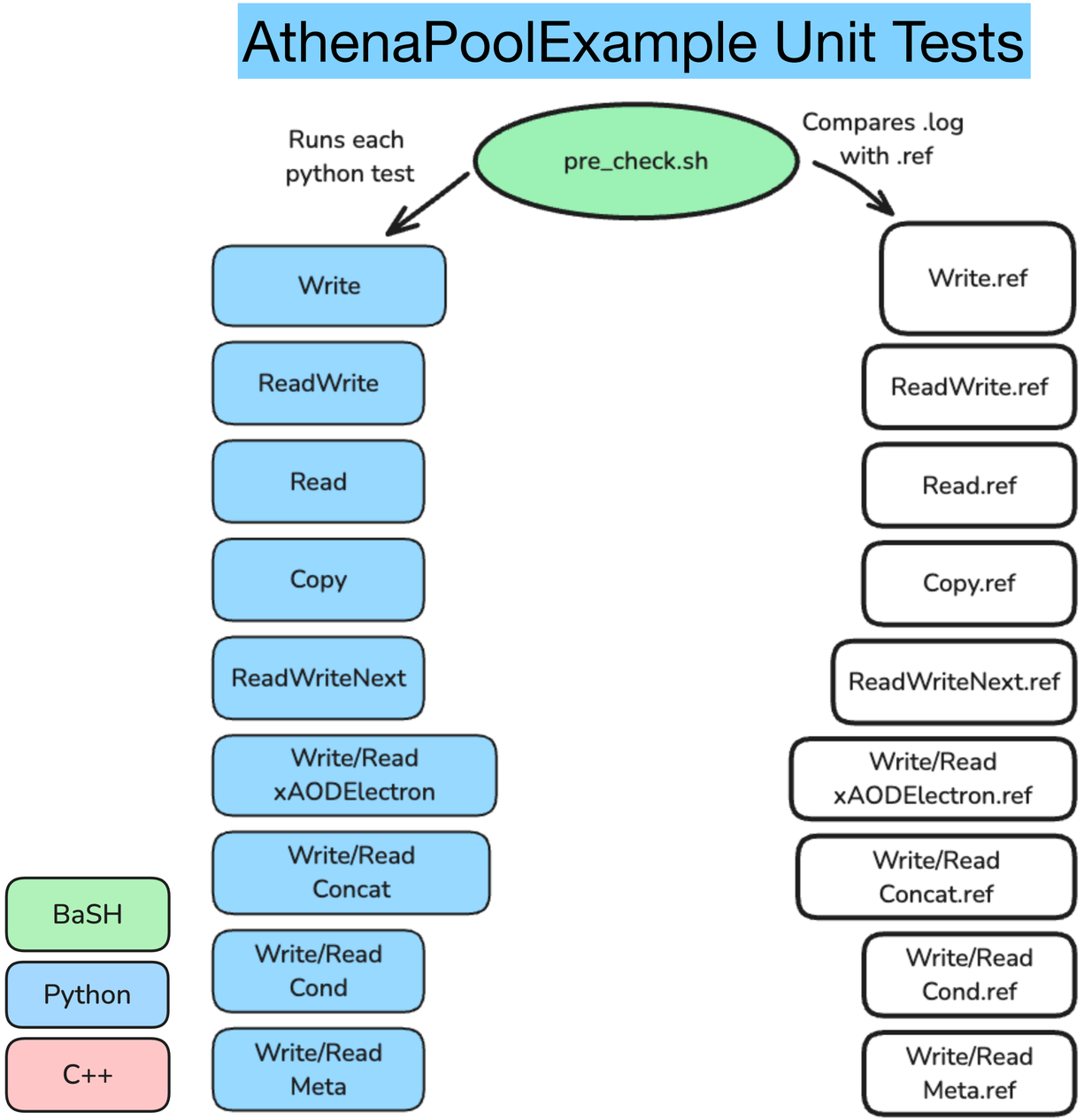


Merges with

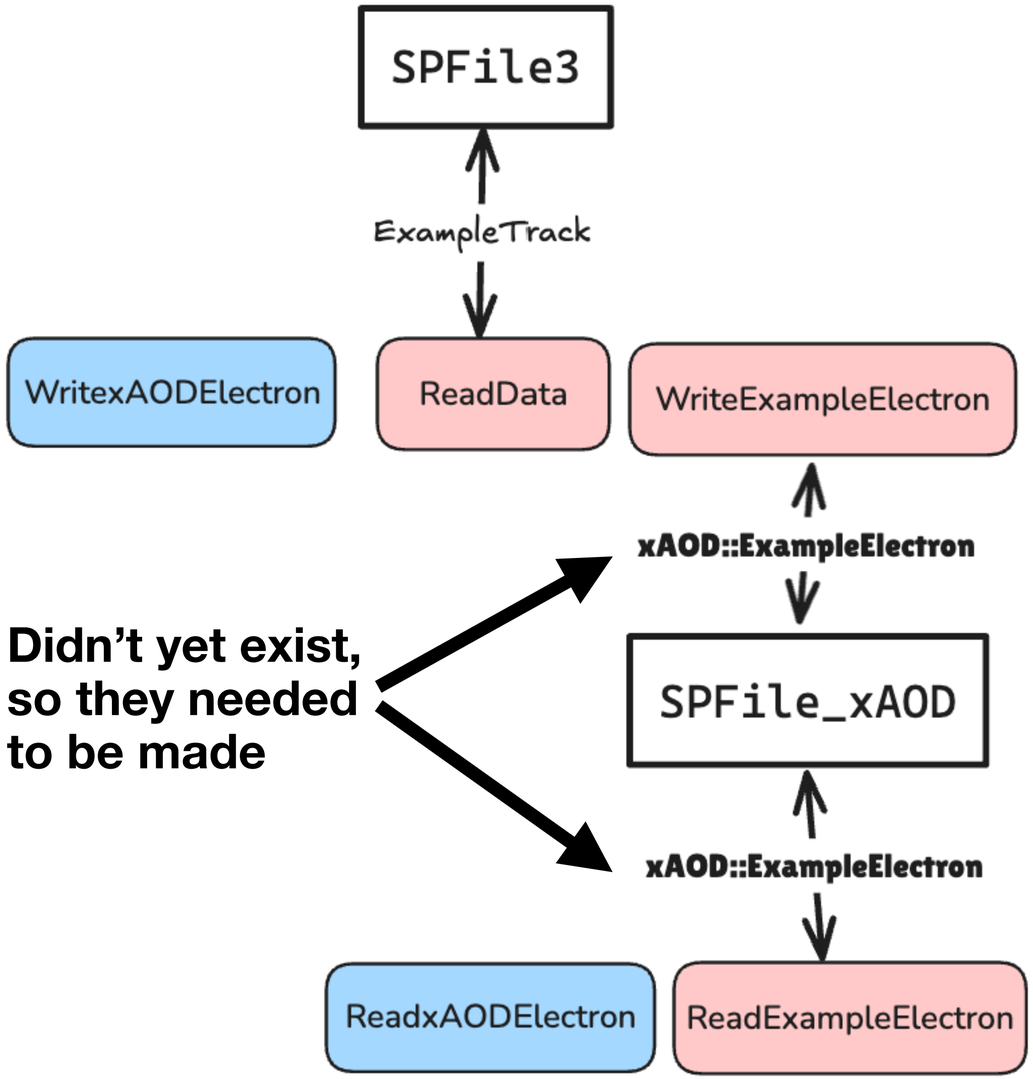


AthenaPoolExample

- There are a number of unit tests in the AthenaPoolExample package that test core I/O functionality, but we want to include one that tests the xAOD EDM specifically.

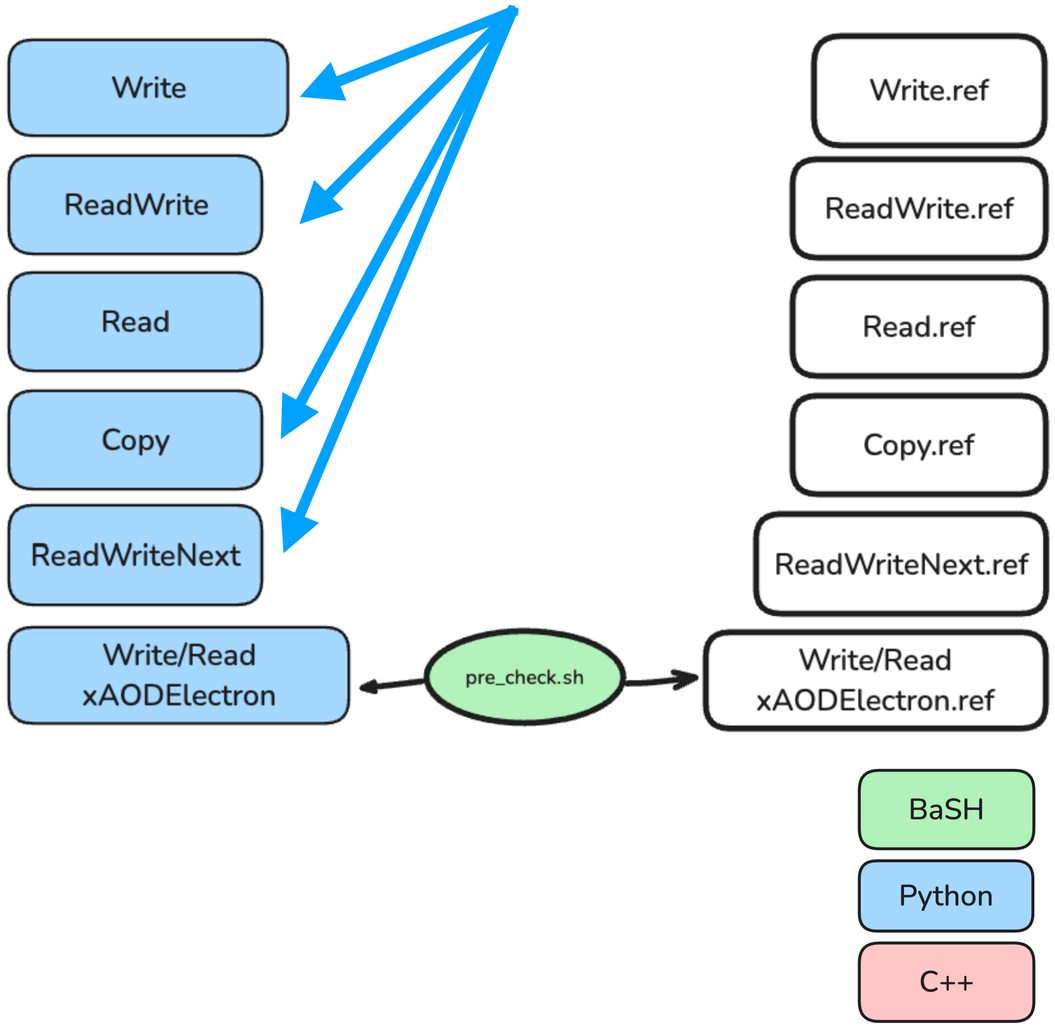


Write/ReadxAODElectron



Didn't yet exist, so they needed to be made

All these SimplePool(SP) Files are created by the preceding unit tests.



xAOD::ExampleElectron Architecture

xAOD::ExampleElectrons are bespoke xAOD objects that try to achieve simplicity

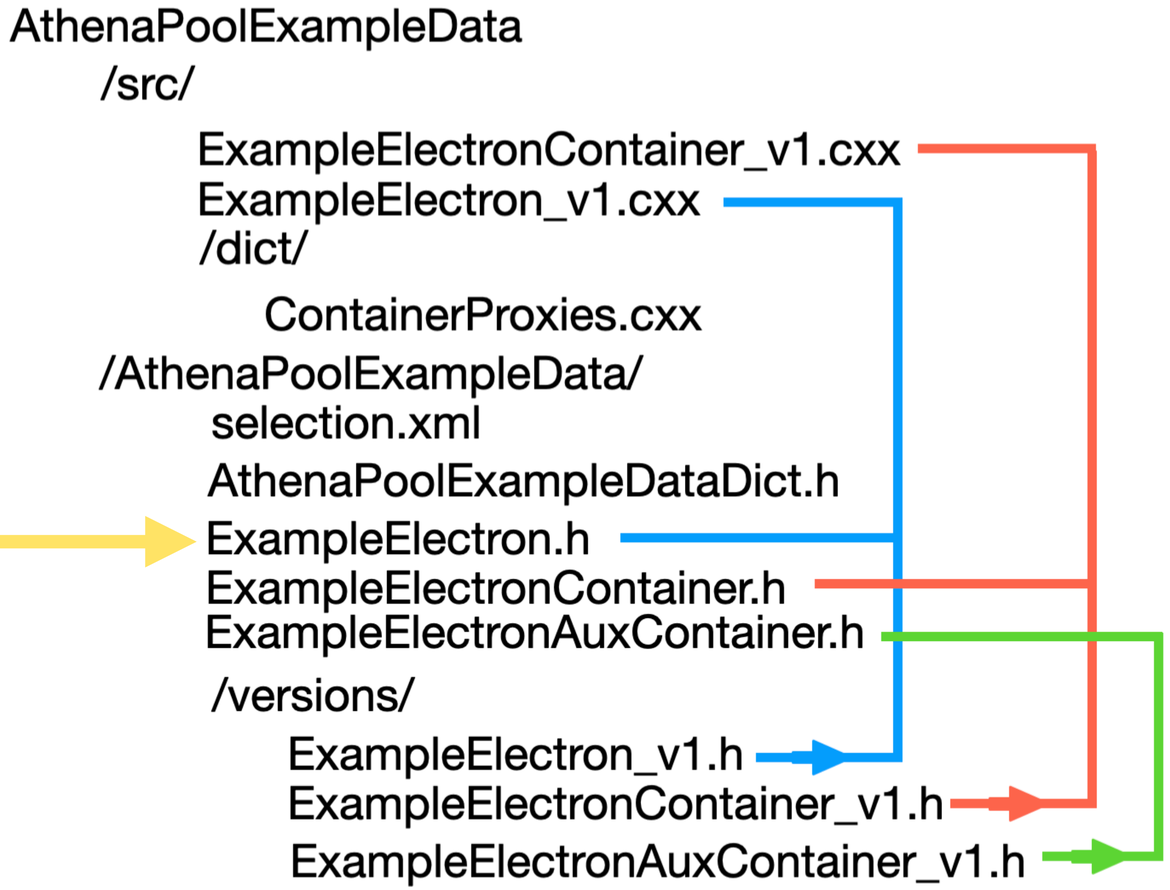
Each of these have unique identifiers that link their ‘xAOD::ExampleElectron___’ to the dictionary that’s accessible by ROOT

They also identify versioning by typed which version of the class to use. I.e.:

```
namespace xAOD {

typedef ExampleElectron_v1 ExampleElectron;

}  // namespace xAOD
```

[LXR link]

[Athena Gitlab link]



xAOD::ExampleElectron Architecture

These could be enveloped in the non-versioned headers above, but this setup works and helps with versioning control a bit.

These versioned header files declare all necessary object types for the xAOD EDM. I.e.

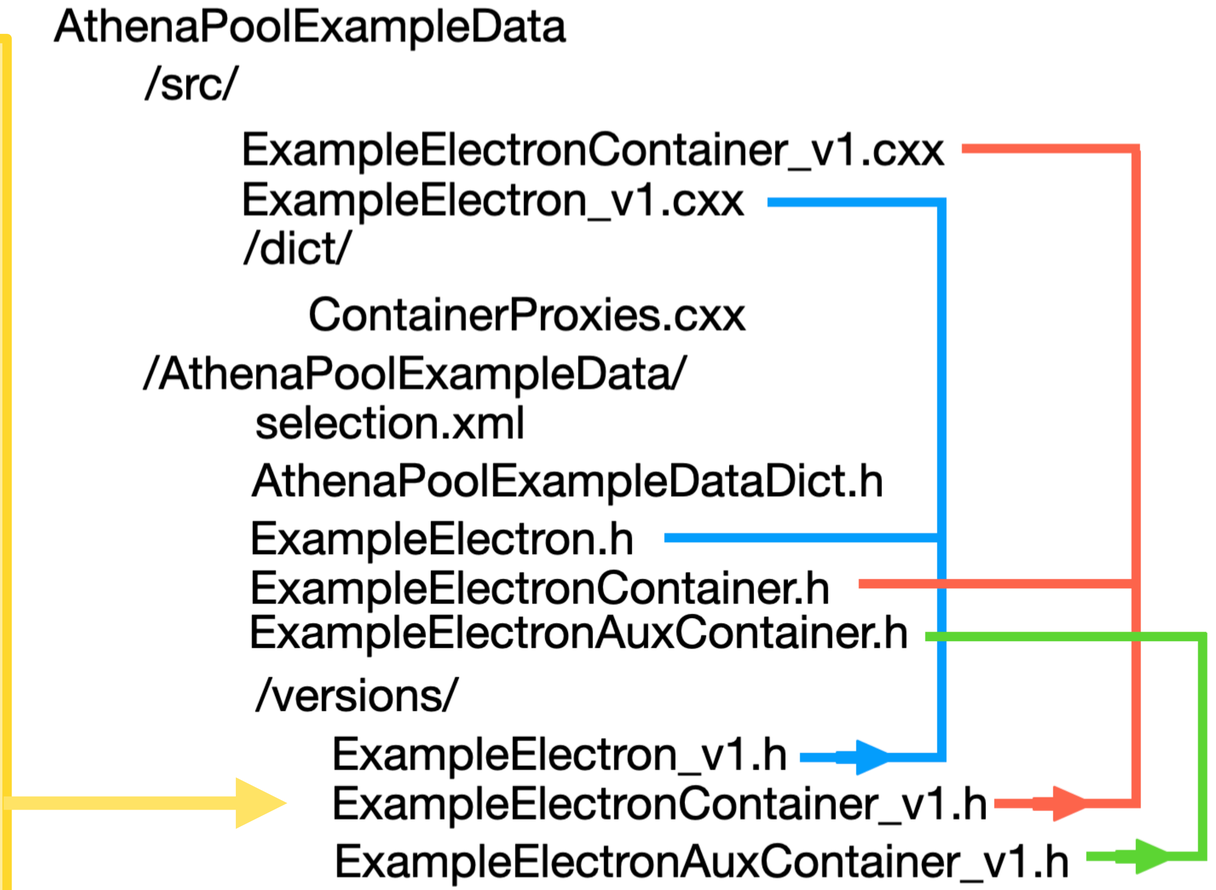
Electron: getter/setters for pt and charge

ElectronContainer:

```
namespace xAOD {
typedef DataVector<ExampleElectron_v1> ExampleElectronContainer_v1;
}
```

ElectronAuxContainer: Declaring AUX_VARS

```
 private:
  AUXVAR_DECL(double, pt);
  AUXVAR_DECL(float, charge);
```



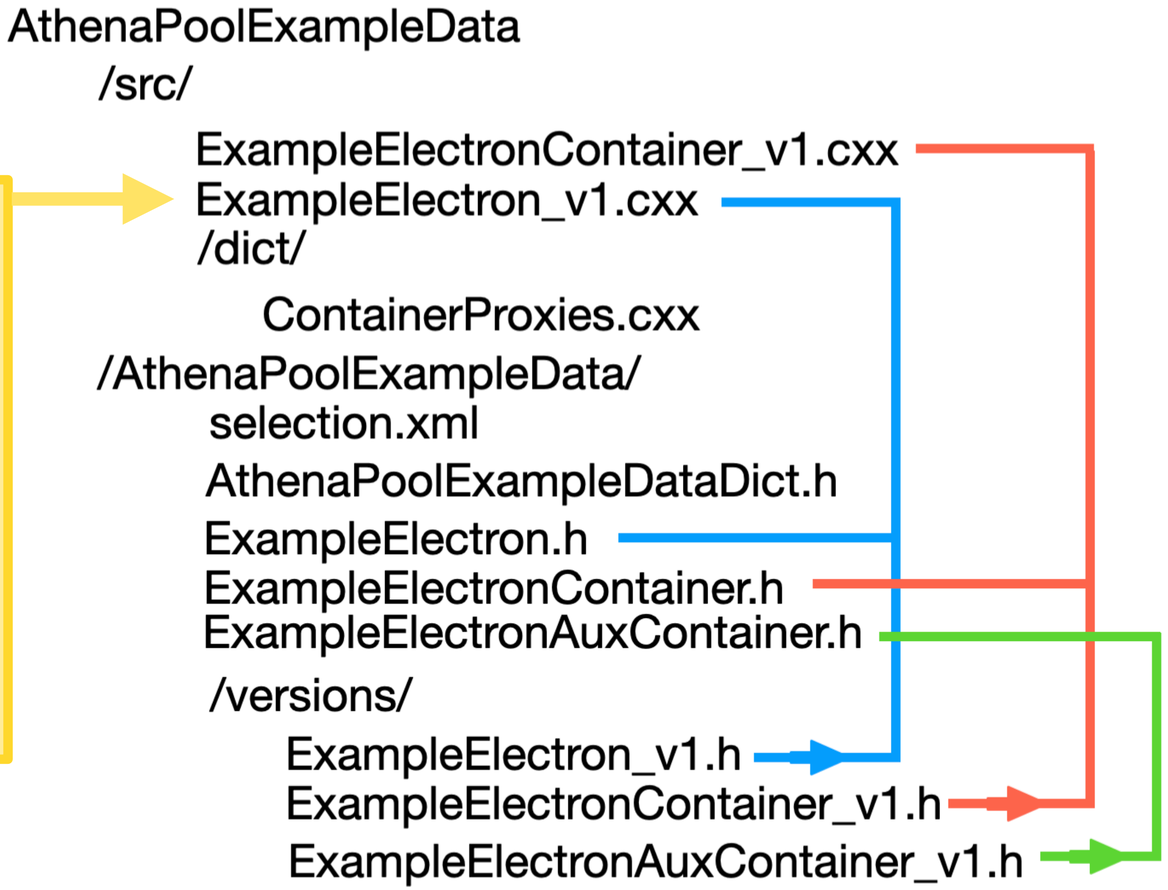
[LXR link]

[Athena Gitlab link]

xAOD::ExampleElectron Architecture

You would probably need a **setter** and **getter** specific to the AuxStore.

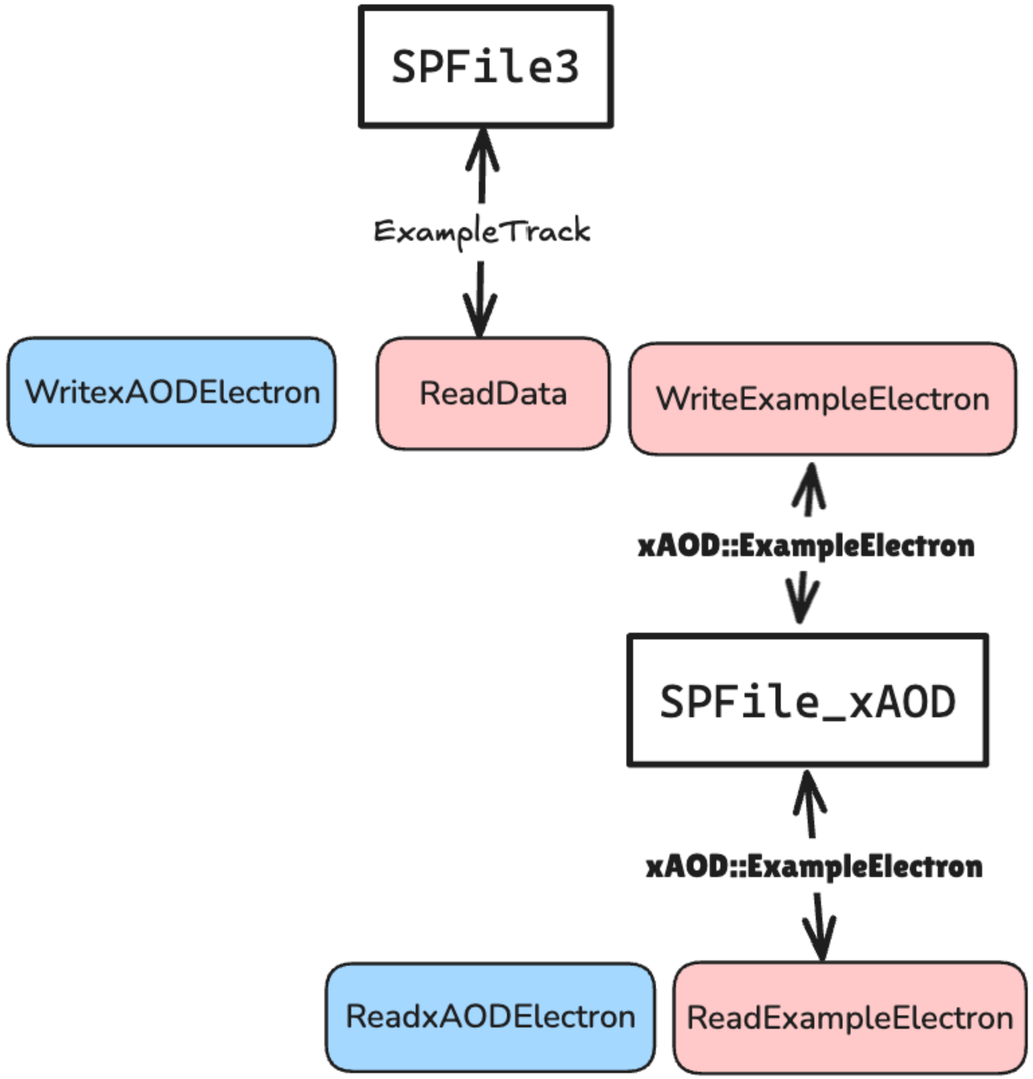
```
#include "AthenaPoolExampleData/versions/ExampleElectron_v1.h"

#include "xAODCore/AuxStoreAccessorMacros.h"

namespace xAOD {
AUXSTORE_PRIMITIVE_SETTER_AND_GETTER(ExampleElectron_v1, double, pt, setPt)
AUXSTORE_PRIMITIVE_SETTER_AND_GETTER(ExampleElectron_v1, float, charge,
                                     setCharge)
}  // namespace xAOD
```

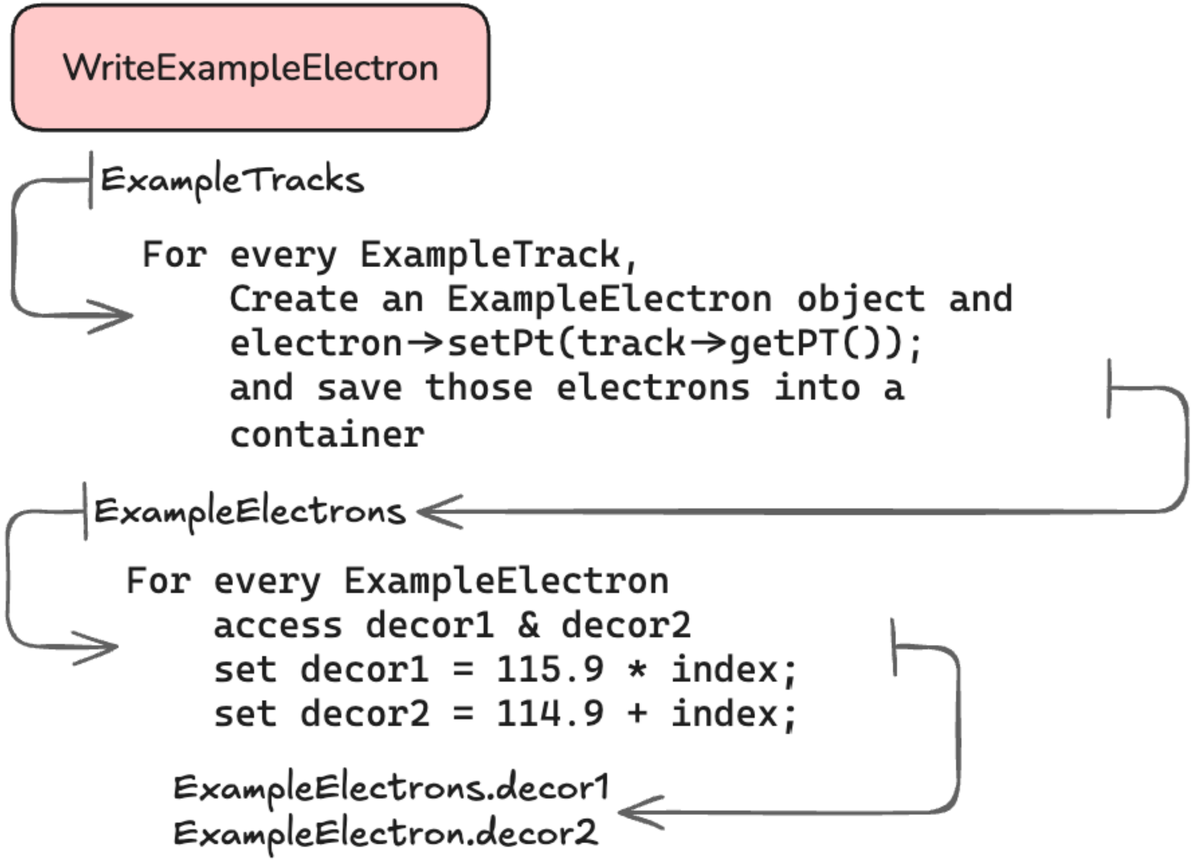


[LXR link]

[Athena Gitlab link]

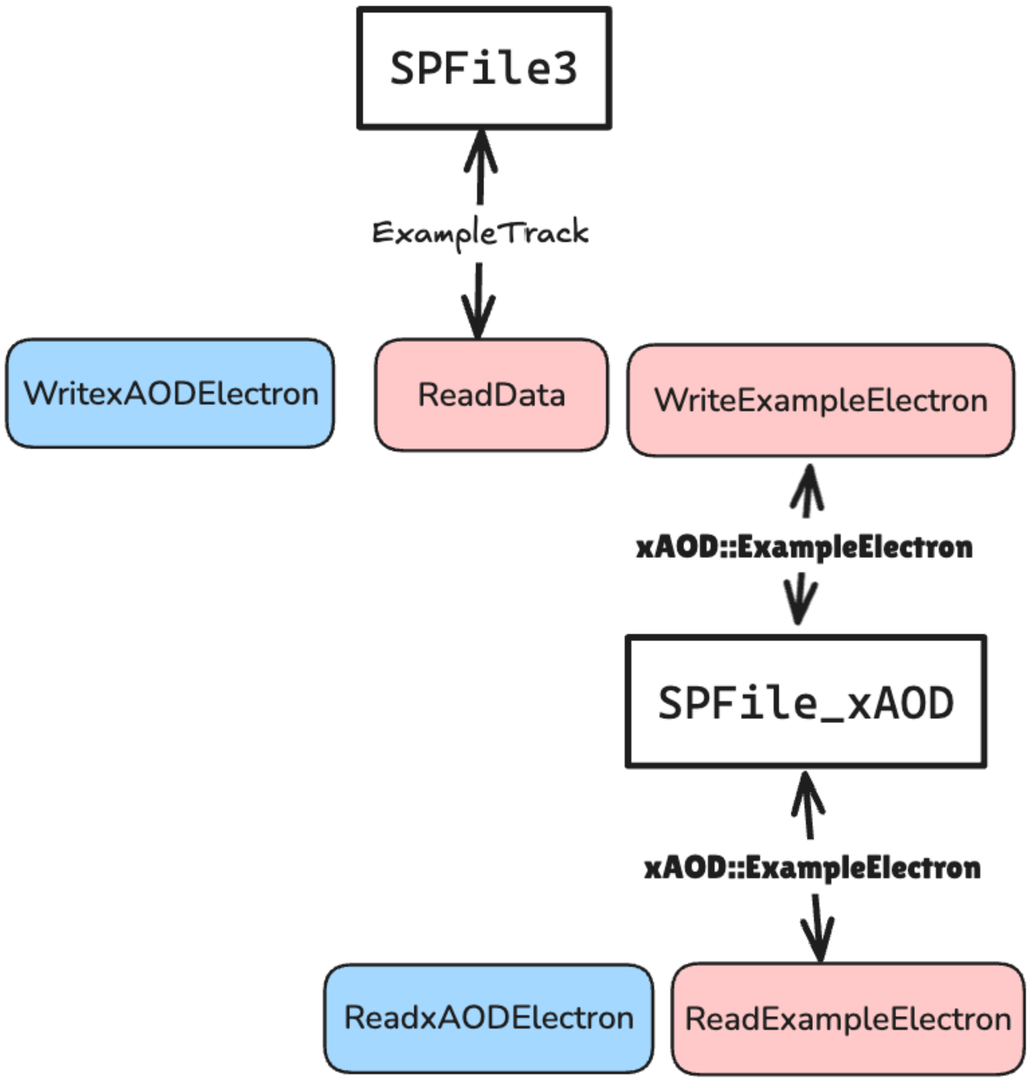
Write/ReadxAODElectron

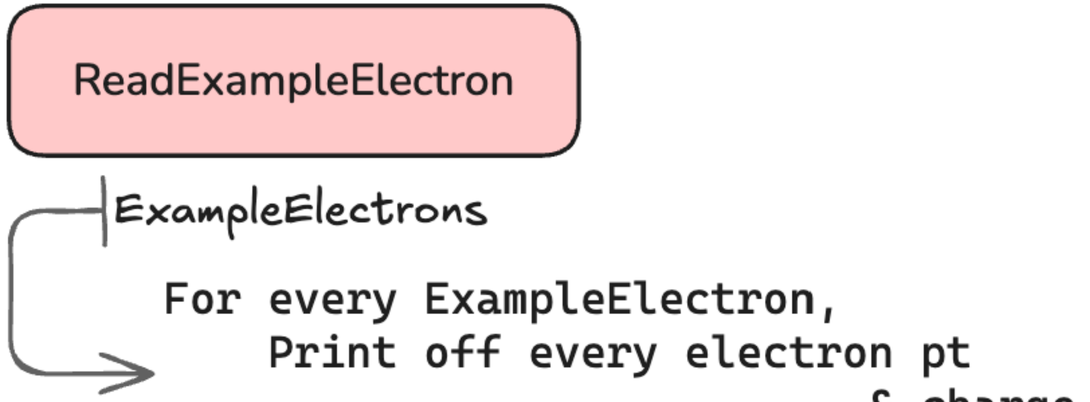




Save this for later so we can test: access decor1 but not decor2.

Write/ReadxAODElectron





WritexAODElectron

```
# Create and attach the algorithms
acc.addEventAlgo( CompFactory.AthPoolEx.ReadData("ReadData", OutputLevel = DEBUG) )
acc.addEventAlgo( CompFactory.AthPoolEx.WriteExampleElectron("WriteExampleElectron", OutputLevel = DEBUG) )

# ----------------  Output Stream configuration
streamCA = OutputStreamCfg( flags, streamName, disableEventTag = True,
                            ItemList = [ "ExampleTrackContainer#MyTracks",
                                "xAOD::ExampleElectronContainer#TestContainer",
                                "xAOD::ExampleElectronAuxContainer#TestContainerAux.-decor2"] )
```

```
# Create and attach the reading algorithm for the ExampleElectron and its decorations
acc.addEventAlgo( CompFactory.AthPoolEx.ReadExampleElectron("ReadExampleElectron",
                                                            OutputLevel = DEBUG) )
```

```
INFO WriteExampleElectron: track # 0 pT = 228.153.
INFO WriteExampleElectron: track # 0is an electron with pT = 228.153;
INFO WriteExampleElectron: DecorationWriter: decor1 = 115.9, decor2 = 114.9.
```

```
INFO ExampleElectron #0 {pt = 137.584, charge = 0}
INFO Decoration reader: decor1 = 115.9
```

```
INFO WriteExampleElectron: track # 0 pT = 2017.04.
INFO WriteExampleElectron: track # 0is an electron with pT = 2017.04;
INFO WriteExampleElectron: DecorationWriter: decor1 = 115.9, decor2 = 114.9.
```

```
INFO ReadExampleElectron: is executing ...
INFO ExampleElectron #0 {pt = 2017.04, charge = 0}
INFO Decoration reader: decor1 = 115.9
```

WritexAODElectron

```
# Create and attach the algorithms
acc.addEventAlgo( CompFactory.AthPoolEx.ReadData("ReadData", OutputLevel = DEBUG) )
acc.addEventAlgo( CompFactory.AthPoolEx.WriteExampleElectron("WriteExampleElectron", OutputLevel = DEBUG) )

# ----------------  Output Stream configuration
streamCA = OutputStreamCfg( flags, streamName, disableEventTag = True,
                            ItemList = [ "ExampleTrackContainer#MyTracks",
                                "xAOD::ExampleElectronContainer#TestContainer",
                                "xAOD::ExampleElectronAuxContainer#TestContainerAux.-decor2"] )
```

ReadxAODElectron

```
# Create and attach the reading algorithm for the ExampleElectron and its decorations
acc.addEventAlgo( CompFactory.AthPoolEx.ReadExampleElectron("ReadExampleElectron",
                                                            OutputLevel = DEBUG) )
```

```
INFO WriteExampleElectron: track # 0 pT = 228.153.
INFO WriteExampleElectron: track # 0is an electron with pT = 228.153;
INFO WriteExampleElectron: DecorationWriter: decor1 = 115.9, decor2 = 114.9.
```

⋮

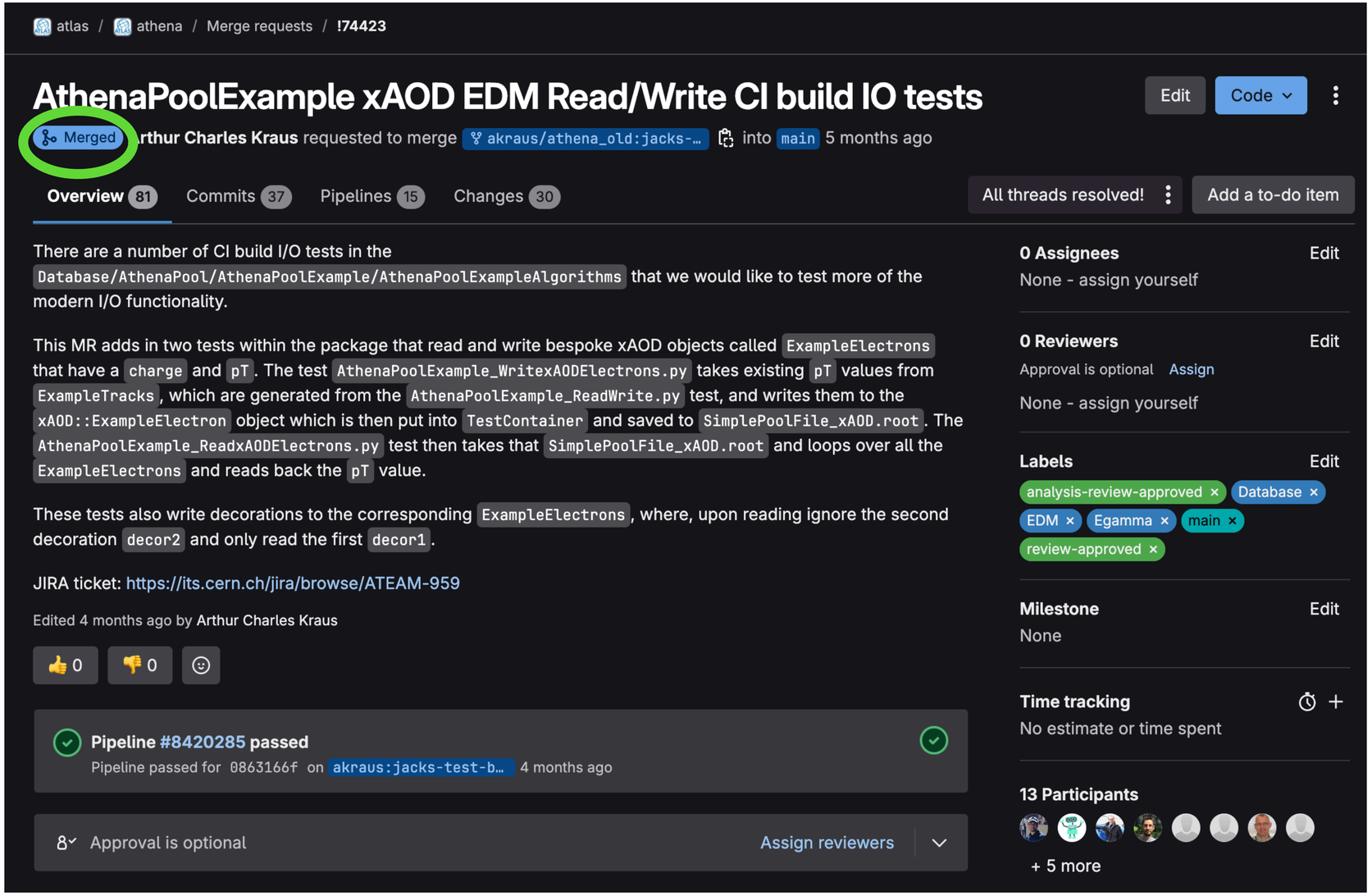
```
INFO WriteExampleElectron: track # 0 pT = 2017.04.
INFO WriteExampleElectron: track # 0is an electron with pT = 2017.04;
INFO WriteExampleElectron: DecorationWriter: decor1 = 115.9, decor2 = 114.9.
```

```
INFO ExampleElectron #0 {pt = 137.584, charge = 0}
INFO Decoration reader: decor1 = 115.9
```

⋮

```
INFO ReadExampleElectron: is executing ...
INFO ExampleElectron #0 {pt = 2017.04, charge = 0}
INFO Decoration reader: decor1 = 115.9
```

CI tests merged with Athena!



Thanks!

Virginia Jamarillo, “Zero Point”, 2023

Backup Slides:

Virginia Jamarillo, “Zero Point”, 2023

AthenaPoolExample_WritexAODElectron.py

```python
from AthenaConfiguration.AllConfigFlags import initConfigFlags
from AthenaConfiguration.ComponentFactory import CompFactory
from AthenaCommon.Constants import DEBUG
from OutputStreamAthenaPool.OutputStreamConfig import OutputStreamCfg, outputStreamName

streamName = "WritexAODElectrons"
outputFileName = "SimplePoolFile_xAOD.root"

# Setup flags
flags = initConfigFlags()
flags.Input.Files = ["SimplePoolFile3.root"]
flags.addFlag(f"Output.{streamName}FileName", outputFileName)
flags.Exec.MaxEvents = -1
flags.Common.MsgSuppression = False
flags.Exec.DebugMessageComponents = [outputStreamName(streamName),
                                     "PoolSvc", "AthenaPoolCnvSvc","AthenaPoolAddressProviderSvc", "MetaDataSvc"]
flags.lock()

# Main services
from AthenaConfiguration.MainServicesConfig import MainServicesCfg
acc = MainServicesCfg( flags )

# Pool reading and writing
from AthenaPoolExampleAlgorithms.AthenaPoolExampleConfig import AthenaPoolExampleReadCfg, AthenaPoolExampleWriteCfg
acc.merge( AthenaPoolExampleReadCfg(flags, readCatalogs = ["file:Catalog1.xml"]) )
acc.merge( AthenaPoolExampleWriteCfg( flags, streamName,
                                      writeCatalog = "file:Catalog1.xml") )

# Create and attach the algorithms
acc.addEventAlgo( CompFactory.AthPoolEx.ReadData("ReadData", OutputLevel = DEBUG) )
acc.addEventAlgo( CompFactory.AthPoolEx.WriteExampleElectron("WriteExampleElectron", OutputLevel = DEBUG) )

# ---------------  Output Stream configuration
streamCA = OutputStreamCfg( flags, streamName, disableEventTag = True,
                            ItemList = [ "ExampleTrackContainer#MyTracks",
                                "xAOD::ExampleElectronContainer#TestContainer",
                                "xAOD::ExampleElectronAuxContainer#TestContainerAux.-decor2"] )
stream = streamCA.getEventAlgo( outputStreamName( streamName ) )
acc.merge( streamCA )

#--------------------------------------------------------------
# Set output level threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR, 6=FATAL)
#--------------------------------------------------------------
stream.WritingTool.OutputLevel = 3
stream.HelperTools[0].OutputLevel = 3

# Run
import sys
sc = acc.run(flags.Exec.MaxEvents)
sys.exit(sc.isFailure())
```

AthenaPoolExample_ReadxAODElectron.py

```
from AthenaConfiguration.AllConfigFlags import initConfigFlags
from AthenaConfiguration.ComponentFactory import CompFactory
from AthenaCommon.Constants import DEBUG

# Setup flags
flags = initConfigFlags()
flags.Input.Files = [ "SimplePoolFile_xAOD.root" ]
flags.Exec.MaxEvents = -1
flags.Common.MsgSuppression = False
flags.Exec.DebugMessageComponents = [ "ReadExampleElectron", "PoolSvc", "AthenaPoolCnvSvc",
                                      "AthenaPoolAddressProviderSvc", "MetaDataSvc", "EventSelector" ]
flags.lock()

# Main services
from AthenaConfiguration.MainServicesConfig import MainServicesCfg
acc = MainServicesCfg( flags )

# Configure AthenaPool reading
from AthenaPoolExampleAlgorithms.AthenaPoolExampleConfig import AthenaPoolExampleReadCfg
acc.merge( AthenaPoolExampleReadCfg(flags, readCatalogs = ["file:Catalog1.xml"]) )

# Create and attach the reading algorithm for the ExampleElectron and its decorations
acc.addEventAlgo( CompFactory.AthPoolEx.ReadExampleElectron("ReadExampleElectron",
                                                            OutputLevel = DEBUG) )

# Run
import sys
sc = acc.run(flags.Exec.MaxEvents)
sys.exit(sc.isFailure())
```

WriteExampleElectron.h

```
#ifndef ATHENAPOOLEXAMPLEALGORITHMS_WRITEEXAMPLEELECTRON_H
#define ATHENAPOOLEXAMPLEALGORITHMS_WRITEEXAMPLEELECTRON_H

/** @file AthenaPoolExampleAlgorithms/src/WriteExampleElectron.h
 *  @brief This file contains the class definition for the WriteExampleElectron
 *class.
 **/

#include "AthenaBaseComps/AthReentrantAlgorithm.h"
#include "AthenaPoolExampleData/ExampleElectronContainer.h"
#include "AthenaPoolExampleData/ExampleHitContainer.h"
#include "AthenaPoolExampleData/ExampleTrackContainer.h"
#include "StoreGate/WriteDecorHandleKey.h"
#include "StoreGate/WriteHandleKey.h"

namespace AthPoolEx {

/** @class AthPoolEx::WriteExampleElectron
 *  @brief This class provides an example for writing event data objects to
 *Pool.
 **/
class WriteExampleElectron : public AthReentrantAlgorithm {
 public:  // Constructor and Destructor
  /// Standard Service Constructor
  WriteExampleElectron(const std::string& name, ISvcLocator* pSvcLocator);
  /// Destructor
  virtual ~WriteExampleElectron() = default;

 public:
  /// Gaudi Service Interface method implementations:
  virtual StatusCode initialize() override final;
  virtual StatusCode execute(const EventContext& ctx) const override final;
  virtual StatusCode finalize() override final;

  // Read in ExampleTracks and ExampleHits
  SG::ReadHandleKey<ExampleTrackContainer> m_exampleTrackKey{
      this, "ExampleTrackKey", "MyTracks"};
  SG::ReadHandleKey<ExampleHitContainer> m_exampleHitKey{this, "ExampleHitKey",
                                                         "MyHits"};

  // We will want to write some of those tracks as an ExampleElectron and place
  // them into a container
  SG::WriteHandleKey<xAOD::ExampleElectronContainer>
      m_exampleElectronContainerKey{this, "ExampleElectronContainerName",
                                    "TestContainer"};
```

```
  // Testing writing decorations, need keys
  SG::WriteDecorHandleKey<xAOD::ExampleElectronContainer> m_decor1Key{
      this, "ExampleElectronContainerDecorKey1", "TestContainer.decor1",
      "decorator1 key"};
  SG::WriteDecorHandleKey<xAOD::ExampleElectronContainer> m_decor2Key{
      this, "ExampleElectronContainerDecorKey2", "TestContainer.decor2",
      "decorator2 key"};

} // namespace AthPoolEx

#endif
```

WriteExampleElectron.cxx

```cpp
#include "WriteExampleElectron.h"

// the user data-class defintions
#include "AthenaPoolExampleData/ExampleElectron.h"
#include "AthenaPoolExampleData/ExampleElectronAuxContainer.h"
#include "AthenaPoolExampleData/ExampleElectronContainer.h"
#include "AthenaPoolExampleData/ExampleHitContainer.h"
#include "AthenaPoolExampleData/ExampleTrackContainer.h"
#include "GaudiKernel/EventContext.h"
#include "StoreGate/WriteDecorHandle.h"
#include "StoreGate/WriteHandle.h"

using namespace AthPoolEx;

//___________________________________________________________________________
WriteExampleElectron::WriteExampleElectron(const std::string& name,
                                           ISvcLocator* pSvcLocator)
    : AthReentrantAlgorithm(name, pSvcLocator) {}

//___________________________________________________________________________
StatusCode WriteExampleElectron::initialize() {
  ATH_MSG_INFO(name() << ": in initialize()");

  ATH_CHECK(m_exampleElectronContainerKey.initialize());
  ATH_CHECK(m_exampleTrackKey.initialize());
  ATH_CHECK(m_exampleHitKey.initialize());
  ATH_CHECK(m_decor1Key.initialize());
  ATH_CHECK(m_decor2Key.initialize());
  ATH_CHECK(m_decor_floatKey.initialize());
  ATH_CHECK(m_decor_longdoubleKey.initialize());

  return StatusCode::SUCCESS;
}

//___________________________________________________________________________
StatusCode WriteExampleElectron::execute(const EventContext& ctx) const {
  ATH_MSG_DEBUG("WriteExampleElectron in execute()");
  std::size_t idx_trk = 0;
  std::size_t MAX = 100;
  float idx_decor = 0.0f;
  float trackPt = 0.0f;

  auto elecCont = std::make_unique<xAOD::ExampleElectronContainer>();
  auto elecStore = std::make_unique<xAOD::ExampleElectronAuxContainer>();
  elecCont->setStore(elecStore.get());
```

```cpp
/*
 * Convert ExampleTrack to xAOD::ExampleElectron
 */

SG::ReadHandle<ExampleTrackContainer> trackCont(m_exampleTrackKey, ctx);

elecCont->push_back(std::make_unique<xAOD::ExampleElectron>());

for (const ExampleTrack* track : *trackCont) {
  trackPt = track->getPT();
  ATH_MSG_INFO(name() << ": track # " << idx_trk << " pT = " << trackPt
                      << ".");

  // Push 10 electrons
  for(size_t idx = 0; idx < MAX; ++idx) {
      // proceed to take this tracks pT
      elecCont->back()->setPt(trackPt);
      ATH_MSG_INFO(name() << ": idx: "<< idx << " Pt= " << elecCont->back()->pt()
                          << ";");

  }


  // Print out the pT that's being saved
  ATH_MSG_INFO(name() << ": track # " << idx_trk
                      << "is an electron with pT = " << elecCont->back()->pt()
                      << ";");


  idx_trk++;
}

SG::WriteHandle<xAOD::ExampleElectronContainer> objs(
    m_exampleElectronContainerKey, ctx);

ATH_CHECK(objs.record(std::move(elecCont), std::move(elecStore)));


/*
 * Writing Decorations
 */

// WriteDecorHandle for the decoration 'TestContainer.decor1'
SG::WriteDecorHandle<xAOD::ExampleElectronContainer, float> hdl1(m_decor1Key,
                                                                 ctx);

// And for the second decoration 'TestContainer.decor2'
SG::WriteDecorHandle<xAOD::ExampleElectronContainer, float> hdl2(m_decor2Key,
                                                                 ctx);
```