An Introduction to Athena

Karsten Köneke
Uni Freiburg

UNI
FREIBURG

The absolute basics

Online software:

- Runs during the data taking (trigger, control, DAQ, etx)

Offline software:

- Processes data once it is committed to storage
  - This tutorial is solely concerned with the offline software

Athena:

- C++ control *framework* in which data processing and analysis is performed
- Athena is based on Gaudi (originally from LHCb)
- Used for Simulation, Reconstruction, even High Level Trigger, DPD making, analysis
- Most software written in C++
- Scripting and configuration in PYTHON

✧ *Athena*: Greek goddess of wisdom, war, the arts, industry, justice and skill; sprang fully grown out of her father's head (Zeus)

✧ *Gaudi*: Antoni Gaudi, Barcelona architect 1852 – 1926, famed for the design of Barcelona's La Sagrada Familia, in immense Basilica which has been under construction since the beginning of the 20th century and is not scheduled for completion until 2026

What is a Software Framework?

- Skeleton for all applications into which developers plug in their software
- Predefines a high level “architecture” or software organization
- Provides functionality common to several applications
- Provides communication between different components
- Controls the configuration, loading and execution of the software

Further reading:

- http://en.wikipedia.org/wiki/Software_framework

… which in practice means

The framework ensures that your code:

- Runs at the right time
- With the right input data
- Using the correct conditions data
- And correctly writes results to disk

All of the offline processing is done in the Athena framework:

- This power and flexibility comes at the cost of a rather large code base
  - …but users don't need to understand every single line of code
  - You have to be able to easily use it!

What are we doing?

We are experimental particle physicist!

- Thus, we are dealing with *lots of data*
- Our data is structured:
  - We have individual "**<u>events</u>**" recorded/generated/simulated!
  - We also have **<u>metadata</u>**, i.e., "data about data", like:
    - Which trigger stream is this data coming from?
    - Is it simulated data?
    - Which trigger menu was used?
    - Which detector geometry was used?
    - …
- We usually want to access every such event and do something with each event:
  - run it through our detector simulation
  - reconstruct it
  - analyze it
  - fill some histograms

We are a collaboration!

We are a big collaboration,

thus, we very often work with many people on a certain project:

- We need to agree on a basic structure of how we can best achieve our goals
- Since many people are working on individual aspects of a whole project:
  - Our software infrastructure needs to support this
  - Our software needs to be flexible to allow for new ideas
  - We need to be able to easily put several pieces of code together
  - Our software needs to be "*plug-and-play*"

Basic concepts: Algorithm

Since our data has the special "event" structure:

- We loop over all events and do a bunch of things for all these events
- Actually, more precisely:
  - We set up our job ***before the event loop starts*** (**initialize**)
  - We run the loop over all events and ***do something for each event*** (**execute**)
  - We do something ***after the event loop is done*** (**finalize**)

Basic concepts: Algorithm

Since our data has the special "event" structure:

- We loop over all events and do a bunch of things for all these events
- Actually, more precisely:
  - We set up our job ***before the event loop starts*** (**initialize**)
  - We run the loop over all events and ***do something for each event*** (**execute**)
  - We do something ***after the event loop is done*** (**finalize**)

In Athena, this is precisely what an Algorithm is:

- A piece of code that:
  - Initializes before the event loop starts
  - Executes once per event
  - Finalizes after the event loop finishes

Basic concepts: Algorithm (2)

Actually, more precisely:

- Athena ***calls*** the Algorithm method `StatusCode initialize()` ***before the event loop starts***
- Athena ***calls*** the Algorithm method `StatusCode execute()` ***once per event***
- Athena ***calls*** the Algorithm method `StatusCode finalize()` ***after the event loop finishes***

Basic concepts: Algorithm (2)

Actually, more precisely:

- Athena ***calls*** the Algorithm method `StatusCode initialize()` ***before the event loop starts***
- Athena ***calls*** the Algorithm method `StatusCode execute()` ***once per event***
- Athena ***calls*** the Algorithm method `StatusCode finalize()` ***after the event loop finishes***

In Athena, when you write your own algorithm, you should:

- Inherit your own algorithm implementation from:
  - AthAlgorithm:

    http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/AthenaBaseComps/html/classAthAlgorithm.html
  - AthFilterAlgorithm (if your algorithm makes a decision if an event should be kept or thrown away):

    http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/AthenaBaseComps/html/classAthFilterAlgorithm.html
  - AthHistogramAlgorithm (for easier handling of histograms):

    http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/AthenaBaseComps/html/classAthHistogramAlgorithm.html

Basic concept: Reporting back

What is this `StatusCode`?

- It tells the caller if whatever the callee was supposed to do went all right
- It is a small class that holds a few enums (name-> unsigned int) and knows if its outcome was checked or not
- The enums are:
  - `StatusCode::SUCCESS`
  - `StatusCode::FAILURE`
  - `StatusCode::RECOVERABLE`

Basic concept: Sequence

Many pieces of code have been written:

- Inside a given event, one piece of code sometimes needs to be executed after another:
  - *e.g.*, because it needs something the first one computed
    - *e.g.*, to reconstruct an electron, you first need to reconstruct a cluster in the calorimeter

Basic concept: Sequence

Many pieces of code have been written:

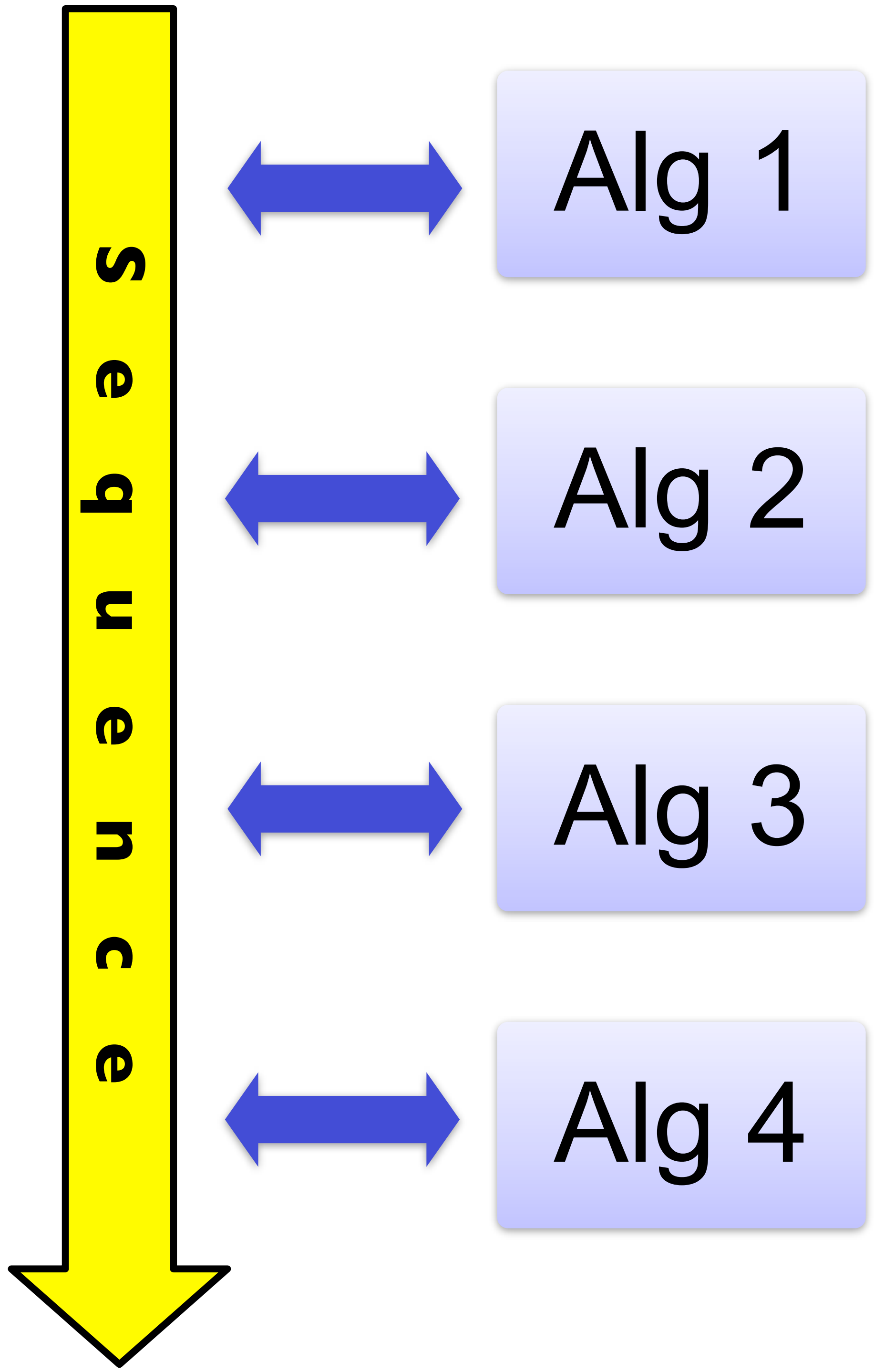
- Inside a given event, one piece of code sometimes needs to be executed after another:
  - *e.g.*, because it needs something the first one computed
    - *e.g.*, to reconstruct an electron, you first need to reconstruct a cluster in the calorimeter

This defines the concept of a sequence of code executions:

- It is just an ordered list of algorithms that tells Athena what to run and in what order
- Athena goes through the whole sequence:
  - once before the event loop to call all `initialize()` methods,
  - then once for every event to call all `execute()` methods
  - and then once after the event loop to call all `finalize()` methods
- The main sequence in Athena is called `topSequence`
  - Basically, this is where you can add all your algorithms

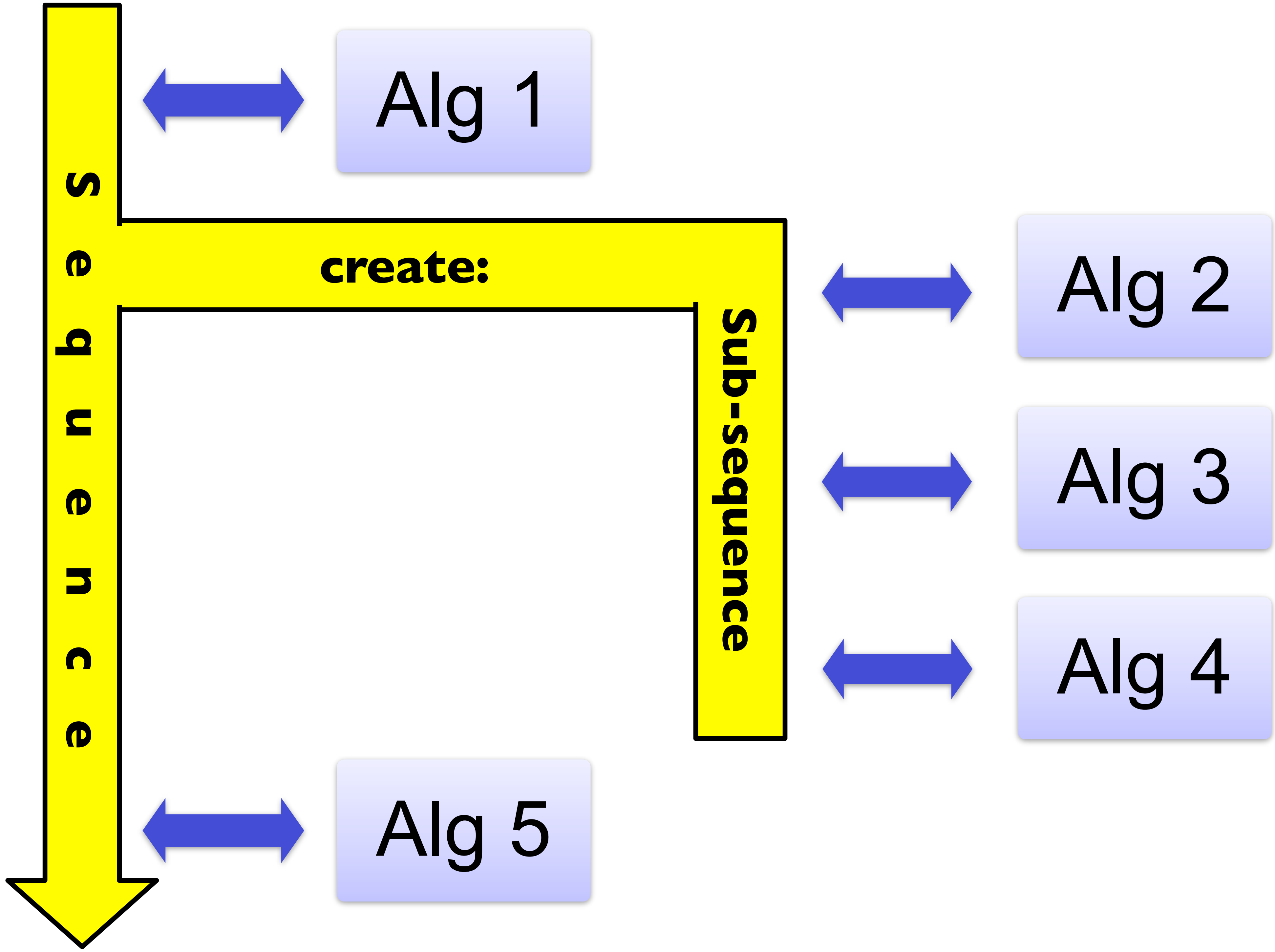
Basic concept: Sequence (2)

So a basic sequence of algorithms can look like this:



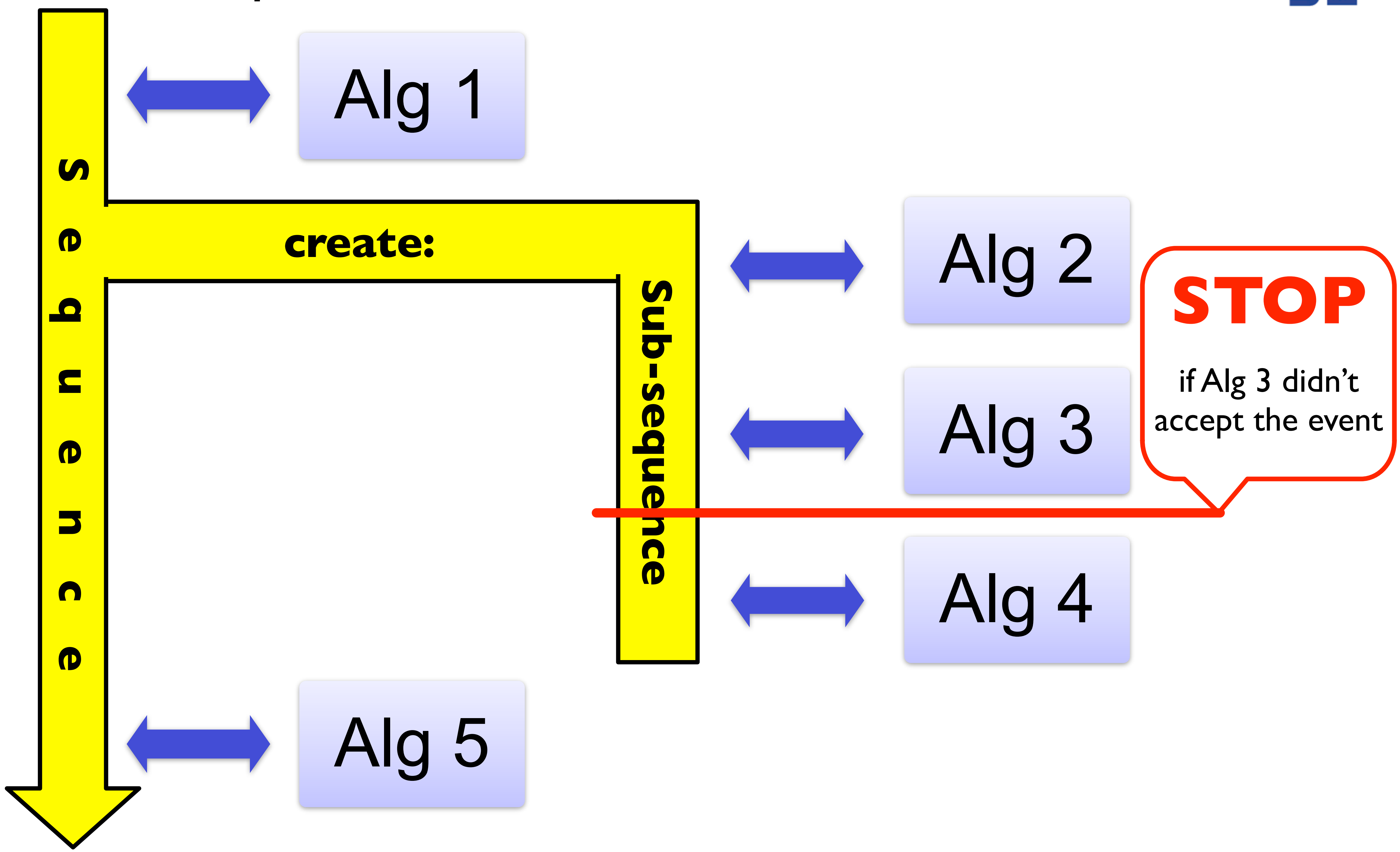
Basic concepts: Sequence (2)

You can create sub-sequences:



Basic concepts: Sequence (2)

You can create sub-sequences:



Basic concepts: Whiteboard

So if the different algorithms don't talk directly with each other, how do they exchange they produced or modified?

Via a whiteboard!

- All data is "published" on a central whiteboard, including the data read in from disk
- Each algorithm can then "**retrieve**" certain data from the whiteboard
- If an algorithm has produced new data, it can "**record**" it to the whiteboard

Basic concepts: Whiteboard

So if the different algorithms don't talk directly with each other, how do they exchange they produced or modified?

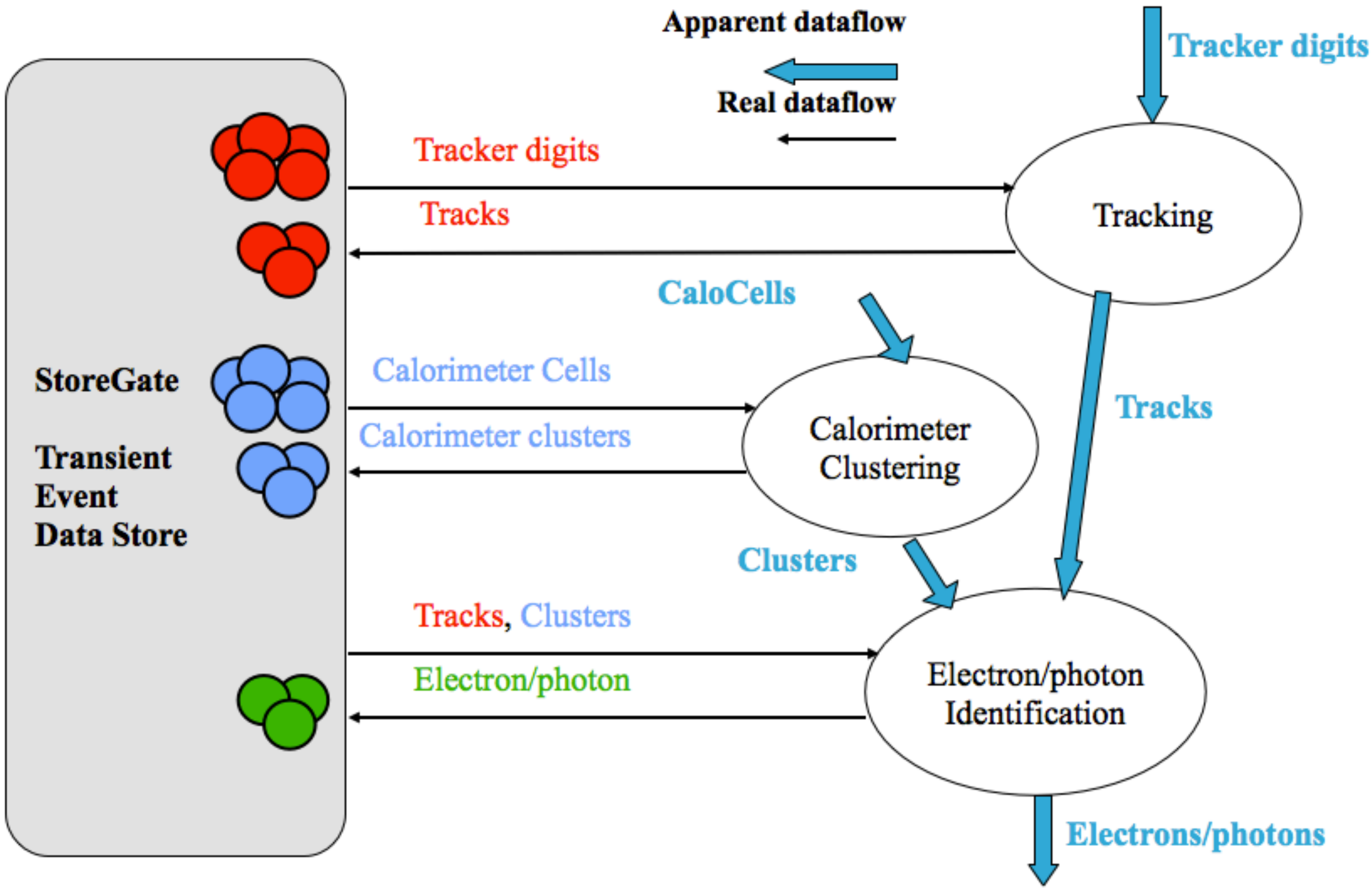
Via a whiteboard!

- All data is "published" on a central whiteboard, including the data read in from disk
- Each algorithm can then "**retrieve**" certain data from the whiteboard
- If an algorithm has produced new data, it can "**record**" it to the whiteboard

In Athena, the software that does the retrieve from and the record to the whiteboard is called "StoreGate"

- There are several whiteboards:
  - One that holds the event data and is wiped clean every event, the event store:
    - The StoreGate instance accessing this event store is (in AthAlgorithms):
      - `evtStore()->retrieve(…); evtStore()->record(…);`
  - One that holds the detector information and remains over many events:
    - The StoreGate instance accessing this detector store is (in AthAglorithms):
      - `detStore()->retrieve(…); detStore()->record(…);`

Athena scheme (a bit less simplified)



Basic concepts: Service

There is also a type of software that provides a service to you:

- You already met one: StoreGateSvc!
- You don't necessarily have to ask it directly to do something for you
- A service gets initialized and finalized, but *NOT called ever event* by the framework, i.e., it **doesn't have an execute() method!**
- There are other important services:
  - Message service: provides the service of handling the printout of messages for you
  - Decision service: handles the decisions of all event filtering algorithms if an event should be kept or not; it makes the final decision
  - CutFlow service: keeps track of all the filtering decisions and selections and bookkeeps them automatically
  - Histogram service: handles the booking and usage of histograms (and other ROOT objects) and hands them over to the output file

Basic concepts: Tool

Our data has even finer granularity than just events:

- E.g., we can have several electrons in an event

Basic concepts: Tool

Our data has even finer granularity than just events:

- E.g., we can have several electrons in an event

There are often tasks that you want to perform an operation per object that is useful in many cases:

- Refit the track of an electron
- Calculate the calorimetric isolation of a muon
- Determine if this muon would pass quality selection criteria
- …

Basic concepts: Tool

Our data has even finer granularity than just events:

- E.g., we can have several electrons in an event

There are often tasks that you want to perform an operation per object that is useful in many cases:

- Refit the track of an electron
- Calculate the calorimetric isolation of a muon
- Determine if this muon would pass quality selection criteria
- …

This type of work can be done with a Tool:

- A Tool gets initialized and finalized, but *NOT called ever event by the framework*, i.e., it **doesn't have an execute() method!**
- Rather, *it has some special method that you can call from your algorithm* for every electron or muon or jet or … (the method depends on the tool at hand)

Basic concepts: Tool (2)

An example of a Tool are the StandardSelectorTools:

- https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StandardSelectorTools
- They have a standard method that you need to call for every particle:
  - `accept( myParticle )`

Basic concepts: Tool (2)

An example of a Tool are the StandardSelectorTools:

- https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StandardSelectorTools
- They have a standard method that you need to call for every particle:
  - `accept( myParticle )`

But when I create an instance of a tool, how do I find it? Who keeps track of it?

- A service: the **ToolSvc**
- When you create an instance of a tool, you add it to the ToolSvc and you can always retrieve it back from there

Basic concepts: Tool (2)

An example of a Tool are the StandardSelectorTools:

- https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StandardSelectorTools
- They have a standard method that you need to call for every particle:
  - `accept( myParticle )`

But when I create an instance of a tool, how do I find it? Who keeps track of it?

- A service: the **ToolSvc**
- When you create an instance of a tool, you add it to the ToolSvc and you can always retrieve it back from there

How should I write my own tool?

- If it should be used only in Athena, then you should inherit from `AthAlgTool`:

  http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/AthenaBaseComps/html/classAthAlgTool.html

- Or inherit from `AthHistogramTool` (to allow for easier histogram handling):

  http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/AthenaBaseComps/html/classAthAlgTool.html

How do I setup and run Athena

Once and for all, put these two lines into your shell startup script (if you are using zsh, this is ~/.zshrc):

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export DQ2_LOCAL_SITE_ID=DESY-HH_SCRATCHDISK
alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh'
setupATLAS
```

(you need to source this file once again, or log back in: source ~/.zshrc)

How do I setup and run Athena

Once and for all, put these two lines into your shell startup script (if you are using zsh, this is ~/.zshrc):

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export DQ2_LOCAL_SITE_ID=DESY-HH_SCRATCHDISK
alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh'
setupATLAS
```

(you need to source this file once again, or log back in: source ~/.zshrc)

Then, every time you want to setup Athena:

- Go to a directory of your choice (create a new one if you are starting something new)
- Type: asetup ReleaseNumber,here
  - For example: `asetup 17.2.7.4.1,here`

How do I setup and run Athena

Once and for all, put these two lines into your shell startup script (if you are using zsh, this is ~/.zshrc):

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export DQ2_LOCAL_SITE_ID=DESY-HH_SCRATCHDISK
alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh’
setupATLAS
```

(you need to source this file once again, or log back in: source ~/.zshrc)

Then, every time you want to setup Athena:

- Go to a directory of your choice (create a new one if you are starting something new)
- Type: asetup ReleaseNumber,here
  - For example: `asetup 17.2.7.4.1,here`

To run Athena:

- Go to a wherever you want to run something (usually a run/ directory)
- Type in:
  - `athena MyTopOptions.py`
- That’s it ☺

Checking out packages, compile, change,...

After you have set up your Athena environment:

- Find the package and its path that you want to check out (see next slide)
- Go to your $TestArea (the directory where you typed "*asetup 17.2.7.4.1,here*")
- Type (to check out the trunk/head of a package):
  - `cmt co Path/To/InterestingPackage`
  - `pkgco.py -A InterestingPackage`
- or (if you need a specific version):
  - `cmt co Path/To/InterestingPackage –r InterestingPackage-00-00-12`
  - `pkgco.py InterestingPackage-00-00-12`
- Then, go to:
  - `cd Path/To/InterestingPackage/cmt/`
  - `cmt make`
- ... easy ☺

D3PD Athena-based analysis model

- Functionalities and analyses implemented as `AthAlgorithms` (or `AthHistogramAlgorithms`)
  - C++ and/or python (mix-and-match, whatever you like)
- Data is read, published, and exchanged via `StoreGate`
- Data is always read **on demand**
  - when one does `evtStore()->retrieve(…)`
  - fast-path for data retrieval (faster than a simple `TTree::GetEntry`)
- Each `TTree` branch can be read by different algorithm classes and multiple instances thereof
  - Data is ONLY exchanged via `StoreGate`, algorithms don't directly talk to each other
  - Thus, very easy and straight-forward to work on the same analysis with several people without obstructing each other; e.g.:
    - one person works on selecting "good" muons and publishes the result via StoreGate
    - another person is working on building "good" Z boson candidates out of muon pairs; use as input simply the "good" muons selected by the other guy
- Leverage Athena job options to assemble a sequence of algorithms
  - can reuse Athena/Gaudi toolset: sequences, filters, pre-scalers,…

A glimpse at how it looks like

Configuring the job to read D3PD ntuples:

```
import AthenaRootComps.ReadAthenaRoot
svcMgr.EventSelector.InputCollections = [
    "ntuple.0.root",
    "ntuple.1.root",
    "root://eos/.../ntuple.10.root",
    ]
svcMgr.EventSelector.TupleName = "physics"
```

A glimpse at how it looks like

Configuring the job to write ntuples:

```
import AthenaRootComps.WriteAthenaRoot as arcw
out = arcw.createNtupleOutputStream(
        "StreamD3PD",
        fileName="filtered.d3pd.root",
        tupleName="MySkimmedD3PD"
      )
out.ItemList += [
    "el_n",
    "el_pt",
    # or just:
    "el_*",
    ...]
```

A glimpse at how it looks like

Retrieving event data for each event (in your C++ algorithm):

```
StatusCode
MyAlg::execute()
{
  uint32_t *runnbr = 0;
  ATH_CHECK(evtStore()->retrieve(runnbr, "RunNumber"));
  ATH_MSG_DEBUG("runnbr: " << *runnbr);
  float *data = new float(42.0f);
  ATH_CHECK(evtStore()->record(data, "my-data"));
  return StatusCode::SUCCESS;
}
```

A glimpse at how it looks like

Retrieving event data for each event in a more modern way, using var-handles:

```
class MyAlg
{
  SG::RVar<uint32_t> m_runnbr;
  SG::WVar<float> m_data;
};
```

In the class declaration/ header (.h file)

```
StatusCode
MyAlg::execute()
{
  ATH_MSG_DEBUG("runnbr: " << *m_runnbr);
  m_data = new float(42.0);
  *m_data += 666.0f;
  return StatusCode::SUCCESS;
}
```

In the implementation (.cxx file)

A glimpse at how it looks like

Retrieving event data for each event in a more modern way, using var-handles:

```
MyAlg::MyAlg(...)
{
  declareProperty("RunNbrHandle",
      m_runnbr = SG::RVar<int>("RunNumber"));
  declareProperty("MyDataHandle",
      m_data = SG::WVar<float>("my-data"));
}
```

In the class constructor in the implementation (.cxx file)

```
# configuration: override output key
alg = CfgMgr.MyAlg()
alg.MyDataHandle = {"key": "my-other-data"}
```

In the python job option (.py) file

How can I find code?

There are several options:

- To just browse the svn repositories:
  - TRAC: https://svnweb.cern.ch/trac/atlasoff/browser
- To search the software that is in a release:
  - LXR: http://acode-browser.usatlas.bnl.gov/
- To look at class hierarchies, automatically generated documentation, and more:
  - Doxygen: http://atlas-computing.web.cern.ch/atlas-computing/links/nightlyDocDirectory/globalDoxySearch.php

Where can I get help?

Google!

…and there are also several email lists that you can (and should) subscribe to:

- The most important are in the hypernews:
  - https://espace.cern.ch/atlas-forums/default.aspx
  - **`hn-atlas-offlineSWHelp`**: All kinds of Athena and software problems (not grid)
  - **`hn-atlas-PATHelp`**: Help with specific physics analysis tools
  - **`hn-atlas-PATDevelopment`**: If you are developing PAT tools
  - **`hn-atlas-dist-analysis-help`**: Any grid-related problems
- There are also the egroup lists for more specialized topics:
  - https://e-groups.cern.ch/e-groups/EgroupsSearchForm.do
  - **`atlas-sw-pat-d3pd-developers`**: D3PD developers

More information about Athena

…there are twiki pages and the ATLAS Computing TDR:

- https://twiki.cern.ch/twiki/bin/viewauth/Atlas/AthenaFramework

Documentation about NTUPLE reading:

- AthenaD3pdReading:
  - https://twiki.cern.ch/twiki/bin/viewauth/Atlas/AthenaRootD3pdReading
- Mana (install Athena for D3PD-reading on your laptop):
  - https://twiki.cern.ch/twiki/bin/viewauth/Atlas/Mana
- AthenaRootComps:
  - https://svnweb.cern.ch/trac/atlasoff/browser/Database/AthenaRoot/AthenaRootComps/trunk

Summary

Questions?

backup

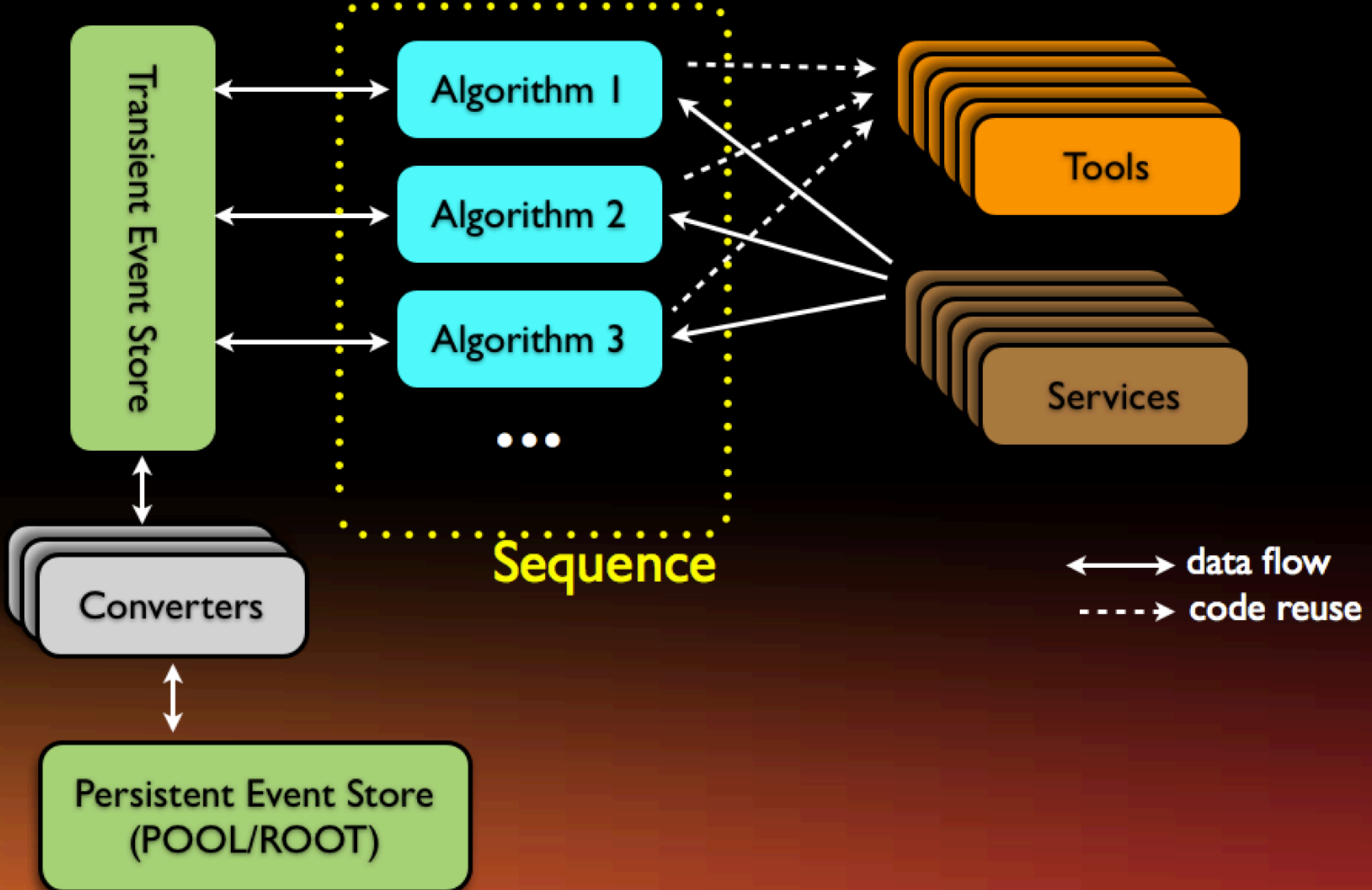
Athena components (i)

- **Algorithm**: an application - a piece of code that "does something"
  - All algorithms inherit from the Algorithm class, which contains three methods:
    - *Initialize*() - run once at the start
    - *Execute*() - run n times
    - *Finalize*() - run once at the end
  - Algorithms are invoked centrally by the framework
  - Many algorithms can be run in a single job - one after the other
- **Data object**: result of an algorithm, or the input to it
  - E.g. Track, Cluster, Muon, Electron, McEvent
- **Service**: globally available software entity which performs some common task
  - Message printing
  - Histogram drawing
- **Event**: a single pass of the execute() method, roughly corresponding to a physics event
- **JobOptions**: Python script which passes user instructions to Athena
  - Which algorithms to run, what order, configuration
  - Control of number of cycles, input/output files, runtime variables etc

Athena components (ii)

- **Tool**: piece of code that is shared between algorithms - it can be executed as many times as you need in the execute() method of your algorithms
- **Auditors**: software which monitors the other components of the framework
- **Sequence**: execution order of the algorithms
- **Filters**: software which allows or forbids an event from passing to the next algorithm in the sequence or being written to disk
- **Transient Store (StoreGate)**: service which stores results of algorithms (data objects) and passes them to the next algorithm.
  - The data is held in the computer memory
- **Persistent Store (POOL)**: format in which the data objects are written to disk
- **Converter**: software which enables the data objects used in the code to be written to and read from POOL without the details of the persistency being included in the objects themselves

... in Practice ?


Transient Event Store
Algorithm 1
Algorithm 2
Algorithm 3
...
Sequence
Tools
Services
Converters
Persistent Event Store
(POOL/ROOT)
data flow
code reuse