Athena and HEC Test Beam Reconstruction Tutorial

October 12, 2001

CERN

N. Kanaya, M. Marino, S. Rajagopalan, M. Wielers

Aims for the Tutorial

- At the end of the day you should:
  - Be more familiar with terms and concepts in Athena
  - Be able to understand code development of Algorithms
  - Know some basics about HEC TB software and data structure in Athena
  - Be able to write simple analysis programs

- We will do that using a simple LArg calorimeter cell reconstruction code and the HEC TB software as example
- Tutorial based on Frascati Athena Tutorial and LArg Tutorial

Tentative Schedule

9:00	1. Introduction	
9:10	2. Configuration & Build System Introduction	
10.10	3. Working in the Athena Framework	
10:30 Coffee Break		
10:45	4. Printing and Job Options	
11:45	5. Accessing Event Data	
12:30	6. Histograms and Ntuples	
1:00 Lunch Break		
14:00	7. N-tuple example	
14:30	8. HEC Test beam Introduction	
14:45	9. Run HEC Athena code without building	
15:30 Coffee Break		
16:00	10. Example with creation of user package	
17:00	11. You've done it!	

Configuration and Build System

Objectives

- After completing this session, you should be able to:
  - Understand the basics of the Configuration Management
  - Get a copy of a package from the repository
  - Know how to (re)build libraries and programs

The Student Accounts

- Each terminal has a unique student account
  - student1, student2, student3, …
- Log in to your student account
  - use the username pasted on your terminal
- From there you must telnet to the ATLAS Linux boxes:
  - The code will not work on the SUN boxes
- telnet lxplus03N.cern.ch, N=1, 2, … 0
  - Use the username: atltr1, atltr2, atltr3, … atltr9, atltra, atltrb
  - password = ATLtrain

The Students Accounts

- Each account has its own maxidisk space:
  - cd maxidisk
- Create working directory
  - mkdir Tutorial
  - cd Tutorial
- All exercises will be done within this directory and below
- To set up some convenient aliases run the setup script
  - source /afs/cern.ch/atlas/project/training/train0/LArTutorial/scripts/Setup_Tutorial.sh

Convenient symbols

Alias:

goto_source	Changes working directory 'source area'
goto_header	Changes working directory 'header area'
goto_build	Changes working directory to 'build area'
goto_run	Changes working directory to 'run area'

Environment variables:

$source_area	Env. Variable pointing to your source area
$build_area	Env. Variable pointing to your build area
$run_area	Env. Variable pointing to your run area

This should setup in addition the environment variable where our examples are located $LArTutorial

Setting up the ATLAS Software

- Packages are maintained with Software Release Tools – CMT
  - Suite of tools and other facilities for the developers of relatively large and complex software systems.
  - Looks after the process of building and releasing products such as libraries and executables
  - Version management is under CVS
- Warning: CMT is still being developed and has known problems
  - Changes in recommended instructions will occur
- useful documentation
  - CMT Manual:
    - **http://www.lal.in2p3.fr/SI/CMT/CMT.html**
  - Using CMT in ATLAS
    - **http://ghez.home.cern.ch/ghez/Temp/doc**
    - **D. Quarries talk in september SW week**
- CMT itself can establish a login environment
  - You don't have to use it, but it makes life easier

Setting up the CMT Environment

- It involves the presence of a *requirements* file in your home or work directory
- Works with all supported shells (tcsh, zsh, bash, *etc*.)
- Needs to be modified to suit your needs
- Typical things to change in requirements file:
  - Your location (CERN, BNL, etc.)
    - **set CMTSITE "CERN"**
  - The current base release
    - **set ATLAS_RELEASE 2.1.1**

Setting up the CMT environment

- Set up CMT environment
  - cd
  - cp $LArTutorial/scripts/requirements .
    - **Do this in your home directory**
  - source /afs/cern.ch/sw/contrib/CMT/v1r9/mgr/setup.sh
    - **Make CMT commands available**
  - cmt config
    - **Environment set according to requirements files**
    - **setup.[c]sh created in your directory**
  - source setup.sh –tag=egcs
    - **Set compiler option egcs or gcc**
    - **Default option if compiler not specified: egcs**
- Now your CMT path is setup correctly
- The above needs to be done once for your working package

Setting up the CMT/Athena Environment

(for example the next day)

- Each time you login you get the right environment for CMT by
  - source setup.[c]sh
- Rerun the “source setup.[c]sh” each time you change the requirements files
- In case the setup.[c]sh is in your home directory you might put the “source setup.[c]sh” in your login script
- You can put the requirements file as well in another place (might be useful if you run several releases with different CMT version)

Setting up the CMT/Athena Environment
(for example the next day)

- If you want to run again your athena program you have to set up the right environment
  - Get right CMT environment
    - **Source setup.[c]sh // in home directory**
  - Set some more CMT environment variables for your relase
    - **. setCMTTEST // in Tutorial area (was created**
      // by getfirstexercice)
  - Get environment set up for running athena TestRelease/cmt directory (goto_build)
    - **source setup.[c]sh // in TestRelease/cmt**
- Hopefully you are not confused,…. I am
- If you run tcsh you might have the message “word too long”
  - cut your $PATH down to the minimum
  - e.g. use script $LArTutorial/path
  - zsh might reveal other problems, don’t get frustrated….

Creating a Test Release

- Even if you just want to run an Athena version without building, the safest way (at the moment) is to checkout the TestRelease package
  - ⌘ cmt co TestRelease
- This checkouts a skeleton package from CVS which you modify to control building other packages that you checkout
- In this tutorial we do not check out the TestRelease but copy it from the tutorial area!!!
- Note that packages are checked out from CVS using the “cmt co” or “cmt checkout” command
  - ⌘ In general don’t use the “cvs co” command
- You need not check out any other packages, unless you need to rebuild the libraries. Otherwise you can use the libraries from the release area.
- Sometimes the A-Team may require you to check out and rebuild packages such as: **EventAthena, AthenaCommon, Zebra TDRCnv, McEventSelector, GaudiInterface ….** Watch out for e-mail

Creating a Test Release

- To create a TestRelease for this Tutorial go to your Tutorial area
  - cp $LArTutorial /scripts/getfirstexercise .
  - . getfirstexercise
- Look at the build script
  - In TestRelease/cmt you have to add the packages you want to use in the requirements file
  - Note, here you have another requirements file + setup.[c]sh. This one is for Athena!
  - Typical things you might change in here
    - **A location for your "default" test release**

      macro CMTTEST "${HOME}/maxidisk/Tutorial"
    - **The version of the GAUDI release (We're trying to avoid having to specify this)**

      macro

      CMTGAUDI "/afs/cern.ch/atlas/offline/external/Gaudi/0.7.3"

Creating a Test Release

- Build all the checked out packages using
  - **cmt broadcast cmt config**
  - **cmt broadcast gmake**
  - **Alternatively you could run "cmt config" and "gmake" in the cmt directory of the package you want to build**
- Set up your run directory and and link some "standard" run files

Exercise 1

- Perform the general setup for CMT
- Run the script to build your TestRelease
  - This will create the shared libraries for all the packages that you have checked out and install it. You need to do this only once for this Tutorial.
  - Note: you will get quite some messages for incompatible versions for CLHEP, HTL and some missing libraries
    - **Ignore them and hope it runs….**
  - Try to understand what's done in the script
- Run the athena executable
  - athena
- Does it work?
  - Yes: Great you got over the 1st hurdle!
  - No: Cry for help

Working in the Athena Framework

What is a Framework?

- Framework Definition [1,2]
  - **Architectural pattern that codifies a particular domain. It provides the suitable knobs, slots and tabs that permit clients to use and adapt to specific applications within a given range of behavior.**

- In practice
  - **A skeleton of an application into which developers plug in their code and provides most of the common functionality.**

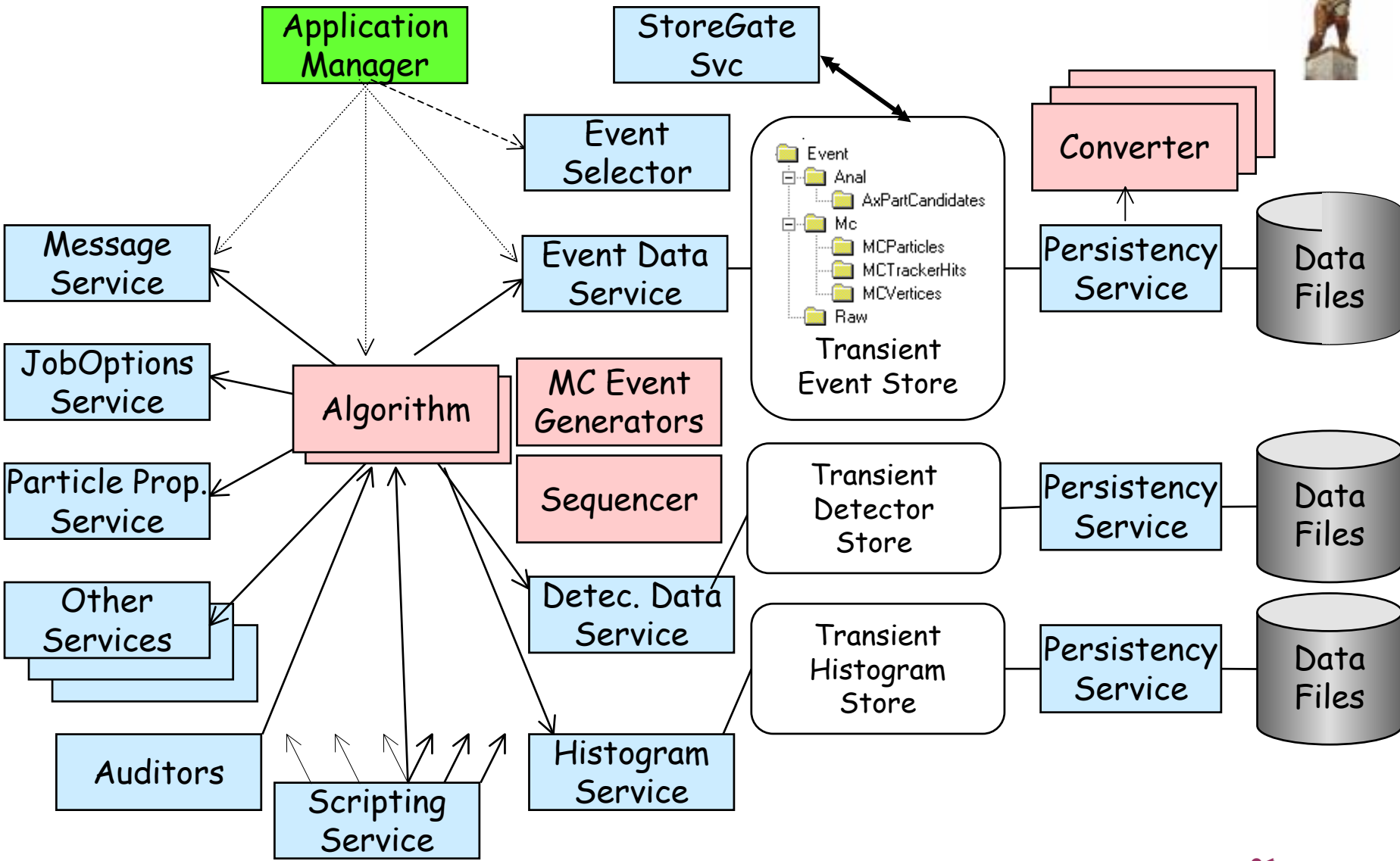
[1] G. Booch, "Object Solutions", Addison-Wesley 1996

[2] E. Gamma, et al., "Design Patterns", Addison-Wesley 1995

Framework Benefits

- Common basis
  - Everyone plugs in code in same framework
  - Everyone uses the same services (no need to reinvent the wheel again and again)
- Low coupling between concurrent developments
  - Smoother integration & organization of the development.
- Robustness
  - Resilient to change (change-tolerant).
- Fosters code re-use

Athena-Gaudi Object Diagram



Definition of Terms (1)

- Algorithms:
  - Users write Concrete Algorithms
  - Must inherit from Algorithm class
  - Implements three methods for invocation by framework :
    - **initialize(), execute(), finalize()**
  - Can be simple or composite Algorithm
    - **Composite Algorithms are made up of several sub-algorithms**
- Data Object (Collection)
  - Produced by Algorithms
  - Atomic unit (visible & managed by transient data store) of data
  - e.g., a Collection of Clusters produced by an Algorithm
    - **Collection containing several Cluster Objects (Contained Object)**

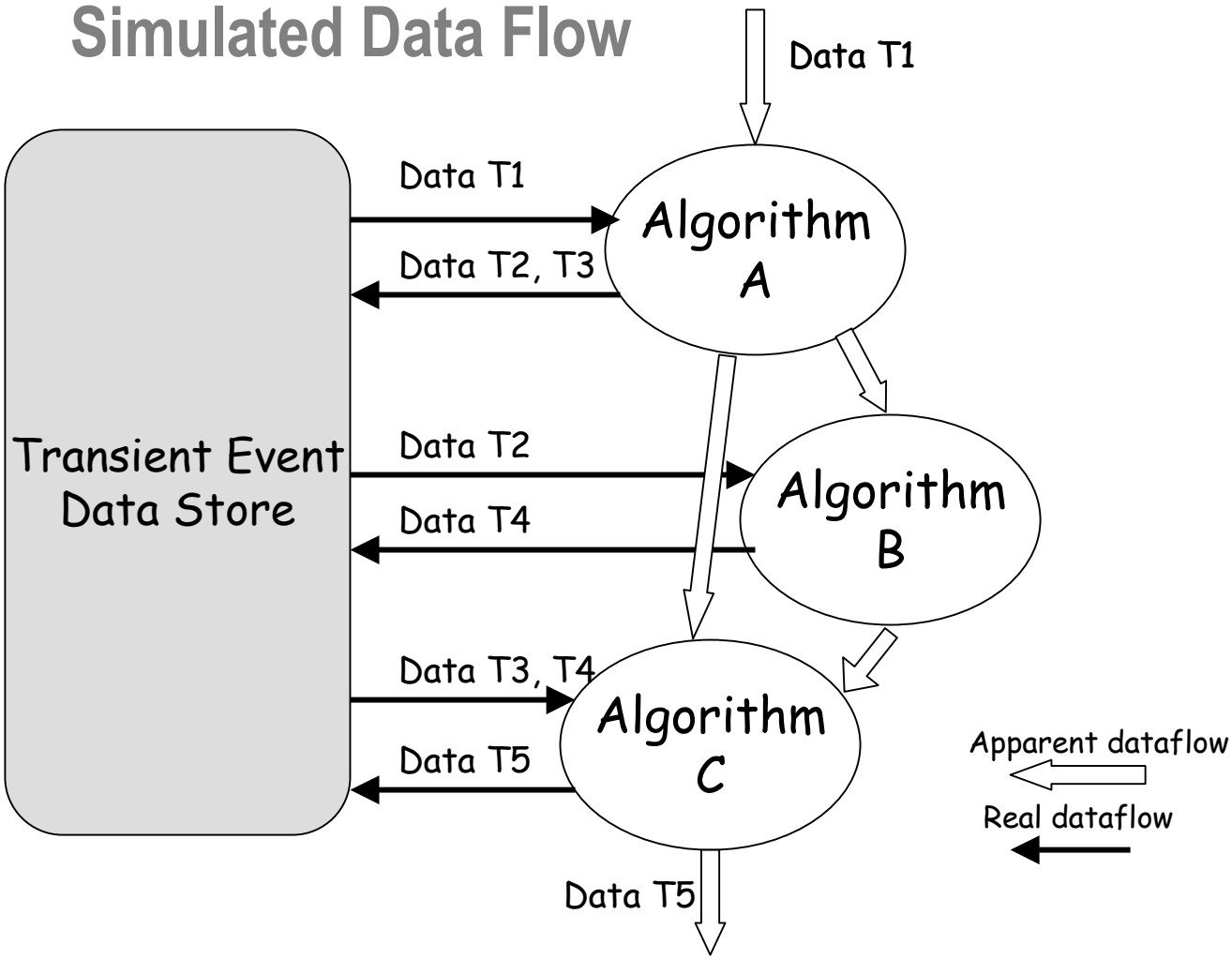
Definition of Terms (2)

- Transient Data Store(s)
  - Central service and repository for data objects (data location, life cycle, load on demand, …)
    - **Event store, detector data store, histogram store**
    - **Should be accessed via Storegate**
- Services
  - Globally available software components providing specific framework capabilities, e.g. Histogram service
- Data Converter
  - Provides explicit/implicit conversion from/to persistent data format to/from transient data
  - Decouple Algorithm code from underlying persistency mechanism(s)

Definition of Terms (3)

- Properties
  - Control and data parameters for Algorithms and Services. Allow for run-time configuration. Specified via a startup text file (jobOption file) or Python script or from the scripting language shell
- Job Options files
  - Conventional text file (default jobOptions.txt) used to control an Athena application configuration at run-time
- Auditors, sequences, Filters…
  - See Athena documentation

Algorithm & Transient Store



Accessing Services

- Within the Algorithm services are readily accessible.
- The most common are:
  - msgSvc( ) [or messageService( )]
  - eventSvc( ) [or eventDataService( )]
  - histoSvc( ) [or histogramDataService( )]
  - ntupleSvc( ) [or ntupleService( )]
  - detSvc( ) [or detDataService( )]
  - service<T>(…) generalized access to Services
  - serviceLocator( )
- And more…
-

Documentation

- Athena User Guide v1.3.0 - new version in preparation & release notes
- Gaudi Developer Guide v7
  - Both from http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/index.html
- Athena Examples
  - http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Tech.Doc/Build/ReleaseNotes.txt
- LAr Tutorial
  - http://atlas.web.cern.ch/Atlas/GROUPS/LIQARGON/software/Reconstruction
- Athena Tutorial
  - http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Tutorial/18may01

Printing and Job Options

Objectives

- After completing this session, you should be able to:
  - Know how to print.
  - Know how to steer algorithms with job options.
  - Know basic job options.

Message Service

- Include header file:
  - #include “Gaudi/MessageSvc/MsgStream.h”
- To print message:
  - MsgStream log(msgSvc(), name());
  - log << MSG::INFO << “Hello there” << endreq;
- Use of the MsgStream class
  - Different levels of printing - OutputLevel:

MSG::NIL	**(=0) DEFAULT**	**(not yet avail.)**
MSG::VERBOSE	**(=1)**	**(not yet avail.)**
MSG::DEBUG	**(=2)**	
MSG::INFO	**(=3)**	
MSG::WARNING	**(=4)**	
MSG::ERROR	**(=5)**	
MSG::FATAL	**(=6)**	

- Message Level specified in jobOptions file:
  - MessageSvc.OutputLevel = 3;
- Settable per Algorithm:
  - MyAlgorithm.OutputLevel = 2;

Athena job configuration

- Job is essentially steered by a conventional text file. Future options are:
  - Database
  - Scripting (interactive session) under development but already in working shape
- Options & properties are accessed through framework interfaces, IJobOptionsSvc or IScriptingSvc - when scripting is activated - .

With reference to the past, think of *data cards*. But it is more than that!

JobOptions details

In jobOptions.txt

Standard Configuration

```
#include "Atlas_TDR.UnixStandardJob.txt"
```

Maximum number of events to execute

```
ApplicationMgr.EvtMax      = <integer>
```

Component libraries to be loaded

```
ApplicationMgr.DLLs += {<comma separated array
of string>}
```

Top level algorithms: "Type/ObjectName"

```
ApplicationMgr.TopAlg += {<comma separated Array of
string>}
```

Comments

```
Preceded by //
```

Include other jobOptions

```
#include "jobOptions_SimpleCell.txt"
```

Exercises

- In all exercise
  - After each code modification, issue:
    - `goto_build`
    - `cmt broadcast gmake`
    - `goto_run`
    - `athena`

Exercise 2a: MsgStream

- Print messages with MsgStream
  - Goto source area and edit SimpleCellBuilder.cxx
  - Add some prints using different print levels
    - **in initialize(), execute() and finalize()**

```
MsgStream log(msgSvc(), name());
log    << MSG::INFO
       << "SimpleCellBuilder"
       << endreq;
```

  - Compile in build area

```
cmt broadcast gmake
```

  - Goto run area

```
athena
```

  - Change print level in jobOptions.txt file and try again

Declare algorithms properties

Declare property variable as data member (*.h)

```
class SimpleCellBuilder : public Algorithm  {
  private:
  ...
double m_Ethreshold;
string m_CellContainerName;
  ...
};
```

Declare the property in the Constructor (*.cxx)

```
SimpleCellBuilder::SimpleCellBuilder( <args> )
  : <initialization>
{
declareProperty("EThreshold", m_EThreshold);
declareProperty("SimpleCellContainerName",
m_CellContainerName);
...
}
```

Hands On: Set Properties

- Set properties in jobOptions file
  - C++ like syntax
    - `SimpleCellBuilder.EThreshold = 0.5;`
    - `SimpleCellBuilder.SimpleCellContainerName = "MyCellContainer"`

<Object name>.<Property name>=<Property value>

Exercise 2b

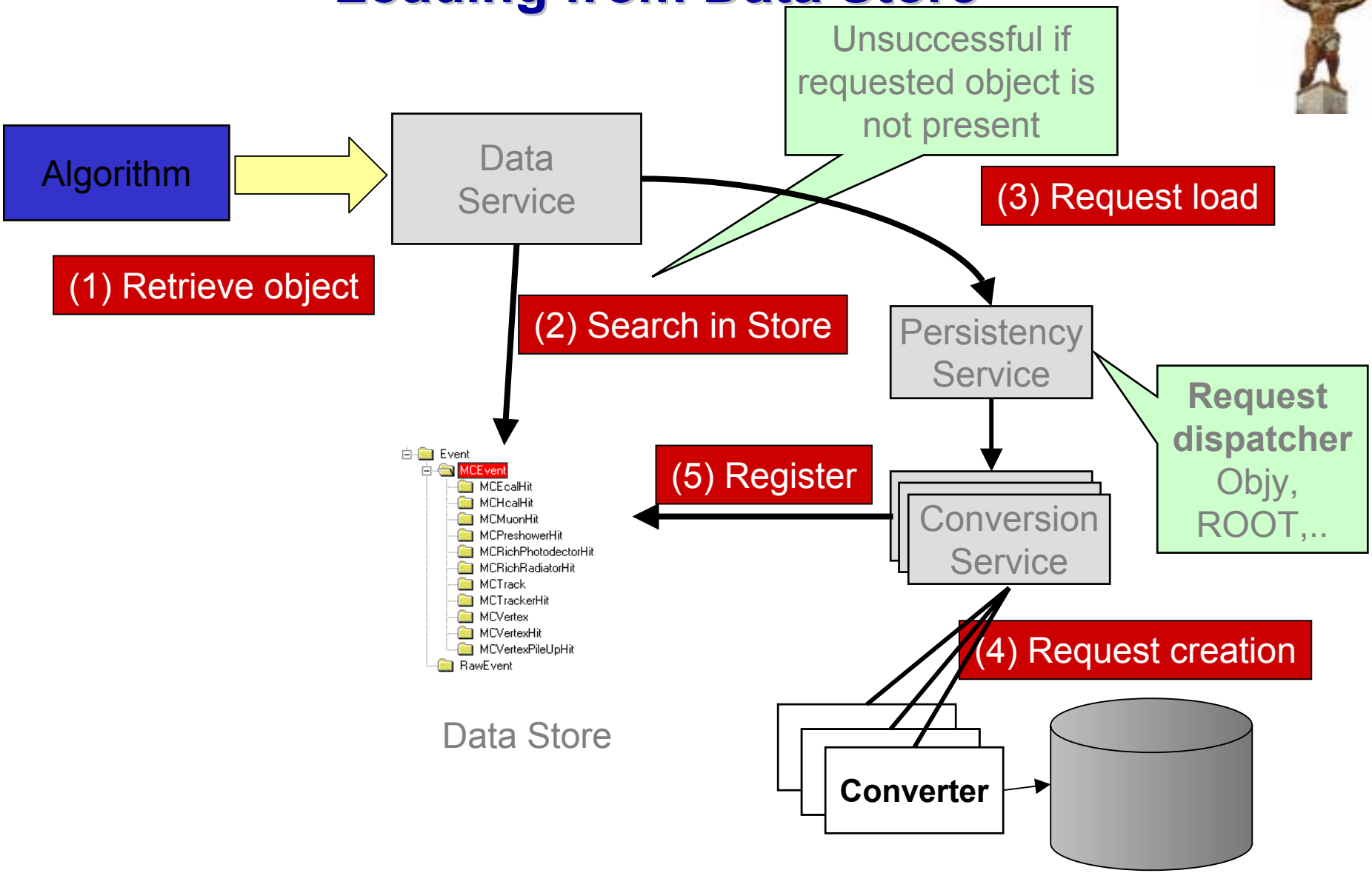
- Add a property to SimpleCellBuilder, namely the name of the SimpleCellContainer. We'll fill this container in the next part of the exercise
- Add a message to print out the name of the container name
- Declare a name for the SimpleCellContainer in jobOptions.
- Compile and run

Data Access via StoreGate

Objectives

- After completing this lesson, you should be able to:
  - Understand how objects are delivered to user algorithms.
  - Retrieve objects from the event data store within an algorithm.
  - Operate on objects collections.

Loading from Data Store



StoreGate: the Athena Transient Store

- Objects of arbitrary type can be posted to SG
  - no need to inherit from DataObject or ContainedObject (it does not hurt either)
  - works with STL and STL-like containers (including ObjectVector). Can support custom ones (HepMC)

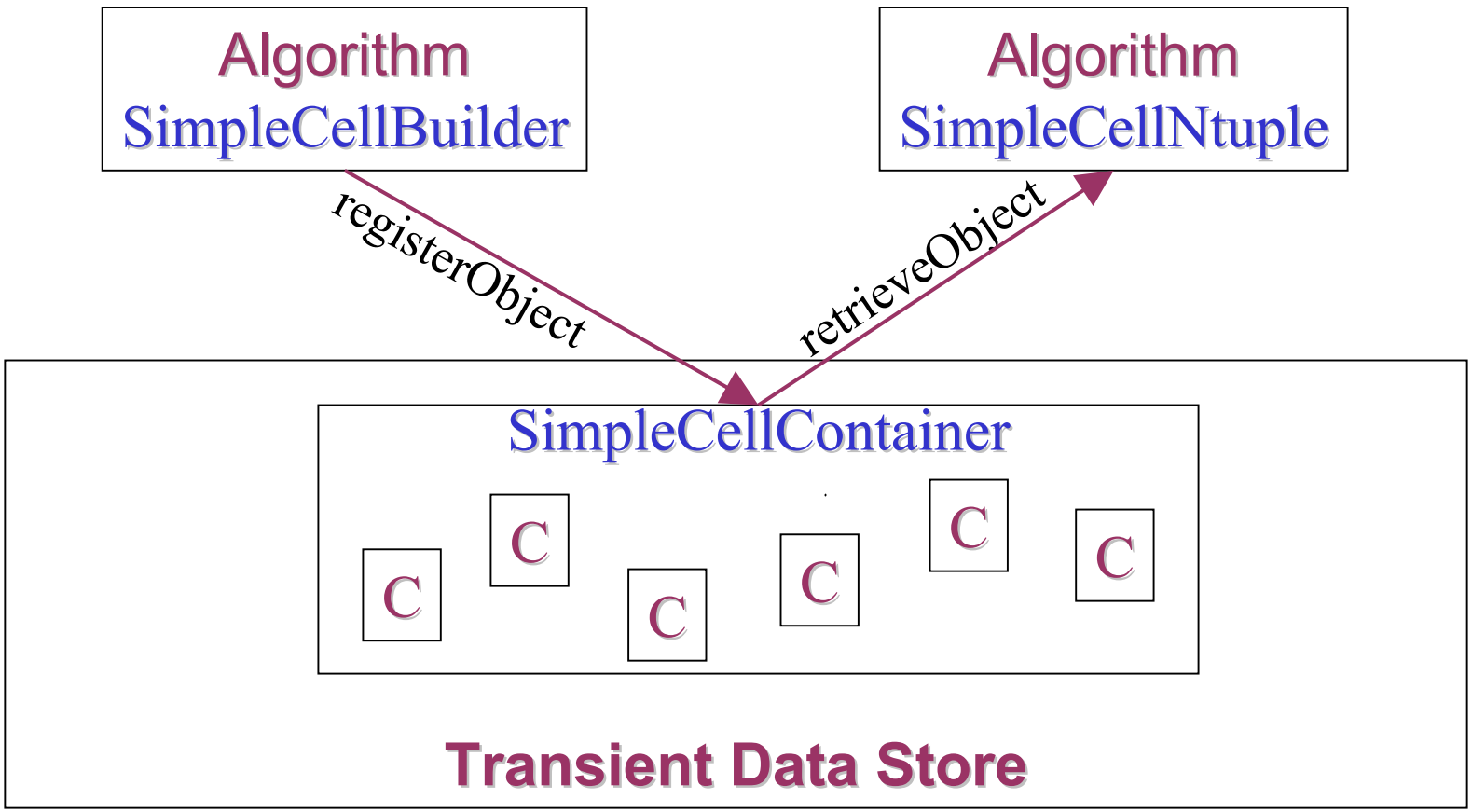
- Type-based hierarchy, type-safe access
  - no need to specify (and propagate!) object "paths"
  - object type is primary key,
    can retrieve *all* CellCollections or the *default* one
  - compiler checks type of retrieved objects
  - optional "secondary key" (not only strings) allows to identify a specific data object

StoreGate Access

- StoreGate, like Gaudi TDS, own recorded objects, The TDS expects full ownership.
  - Do not destroy existing objects! SG will delete them for you at the right time
  - It's not yours!
    - **typically results in an access violation!**
  - the data store is responsible for calling the corresponding *delete* operator at the end of each event.
- "Almost const" access
  - You can't modify an object you retrieve
    - **compiler error!**
  - you can, when recording an object, allow downstream algos to modify it
    - **use sparingly!**

The SimpleCellBuilder Example

This Example will demonstrate how to store and retrieve data objects from the Transient Data Store.



Registering & Retrieving from TDS via SG

- In the initialize step get pointer to the StoreGate service

```
StatusCode sc=service("StoreGateSvc", m_storeGate)
If (sc.isFailure()) { print error message;
                      return StatusCode::FAILURE;}
```

Declare m_storeGate as a private data member in your header file

- To register (record) a SimpleCellContainer:

```
StatusCode sc = m_storeGate→record(ptr_container,
    m_CellContainerName);
if ( sc != StatusCode::SUCCESS)    { print some error message
    }
```

ptr_container = pointer to the SimpleCellContainer object

CellContainerName is a name given to this CellContainer

Registering & Retrieving from TDS via SG

- To retrieve this object:

  Const DataHandle<SimpleCellContainer> cellcontainer;
  StatusCode sc=m_storeGate→retrieve(cellcontainer, m_CellContainerName)

  Use the DataHandle as a pointer

  CellContainerName is the given name while registering this object

Exercise 2c:

- In SimpleCellBuilder you have two data objects:
  - SimpleCellContainer & SimpleCell
- In the initialize pointer to StoreGate Service is already set
- In the execute method of SimpleCellBuilder():
  - Register the SimpleCellContainer in the TDS
    - Container name specified in last example 2b in the jobOptions; take that one
  - Check the return status and issue a success message or an error
  - Look at the loop that makes SimpleCell and puts them in the container: In particular, look at the use of a gaussian random number generator.

Exercise 2c

- You have two algorithms:
  - SimpleCellBuilder & SimpleCellNtuple
  - Add SimpleCellNtuple in the program execution
  - Make sure the CellContainerName in the SimpleCellBuilder and SimpleCellNtup are the same
- Run
- Start PAW and look at the produced ntuple SimpleCell.ntup

Histograms & Ntuples

Histograms & Ntuples

- One of they key tools in HEP
  - Wheel not being reinvented
  - Usage and function closely matching HBOOK
    - **book & fill**
- Kept in TDS in their own areas
  - /NTUPLES and /stat
  - managed by NTupleSvc and HistogramSvc
- Persistency
  - HBOOK
  - ROOT

Histograms - Good To Know...

- Histograms are kept in special TDS area
  - unlike the event, it is not cleared with each new event
  - same access mechanism as other DataObjects
- If not saved - they are lost
  - histograms are kept in memory
  - must be registered with the TDS and saved at end of job

Booking 1-d Histograms

- Through the Histogram service

```
IHistogram1D* multiplicityH1D =
 histoSvc()->book("/stat/myhist/1",
                  "Visible chargedMultiplicty",
                  100,
                    0,
                 1000.0);
```

Number of bins

Low Edge

High Edge

"/stat/myhist/1"

Opt.

RZ-Directory

ID

Theory: Histogram identifier (short name)
Practice: HBOOK histogram ID

Booking 2-d Histograms

- Through the Histogram service

```
IHistogram2D* multiplicityVsEnergyH2D =
 histoSvc()->book("/stat/myhist/2",
                  "MultiplictyvsEnergy(GeV)",
                    100,
                    0.0,
                    1000.0,
                    50,
                    0.0,
                    500.0);
```

- Title
- Number of bins (X)
- Low Edge (X)
- High Edge (X)
- Number of bins (Y)
- Low Edge (Y)
- High Edge (Y)

Filling Histograms

1-D Histograms

```
multiplicityH1D->fill(
```

X - value	mult,
Weight	1.0);

2-D Histograms

```
multiplicityVsEnergyH2D->fill(
```

X - value	mult,
Y - value	energy,
Weight	1.0);

Calls similar to HBOOK HF1 & HF2

Histogram Persistency

- In jobOptions file
  - Load the relevant shared library
    - `ApplicationMgr.DLLs += {"HbookCnv"}/ {"RootHistCnv"};`
  - Specify persistency (accordingly to shared lib)
    - `ApplicationMgr.HistogramPersistency =`
    - `"HBOOK" / "ROOT" / "NONE" (default);`
  - Specify name of output file, e.g.,
    - `HistogramPeristencyService.OutputFile = "myana.hbook" / "myana.rt";`

Ntuples - Good To Know...

- Unlike histograms they cannot be kept in memory
  - rows are constantly added
    - **size of Ntuple is not constant**
- Like all other data - reside in a Data Store
    - **Same access mechanism**
- Both column-wise and row-wise Ntuples are supported

Booking A Ntuple

Through the Ntuple service

```
NTuplePtr nt( ntupleSvc(), "/NTUPLES/FILE1/100”);
if ( !nt )  { //check if already booked
  nt = ntupleSvc()->book (
                    "/NTUPLES/FILE1/100",
                    CLID_ColumnWiseTuple,
                    ”My Ntuple");
   if ( nt )     { m_ntuple = nt; }
}
```

Location on store
Row- or Column wise ?
Title

“/NTUPLES/FILE1/100”

Theory/Practice: HBOOK Ntuple ID

Optional

Logical name of RZ-File

NTUPLE ID

Define Ntuple Columns

- For column-wise, variable must first be declared as data members of the class
  - ⌘ Item, Array, Matrix of type bool, float, long

```
NTuple::Item<long>     m_ntrk;
NTuple::Item<float>    m_energy;
NTuple::Array<float>   m_mom;
NTuple::Tuple* p_nt1;
```

- After the Ntuple is booked

```
if ( nt1 )     {
  status = nt1->addItem("Ntrack, m_ntrk,
  0,5000);
  status = nt1->addItem("Energy",m_energy);
  status = nt1->addItem("Momentum",m_ntrk,
  m_mom);
}
```

Index

Array with index

Filling Ntuples

- Items behave like a number
  - Just assign values:

```
m_energy = 50.67;
```

  - Same for arrays

```
m_mom[i] = 10.25;
```

- Write record

```
//status = ntupleSvc->writeRecord(m_ntuple);
  status = m_ntuple->write();
   if ( status.isFailure() )  {
     log << MSG::ERROR << "Cannot fill Ntuple"
<< endreq;
   }
```

Ntuple Persistency

In job options specify output files

```
NTupleSvc.Output = { "FILE1
                     DATAFILE='myana.ntup'
                     OPT='NEW'
                     TYP='HBOOK'" };
```

Technology: HBOOK

```
NTupleSvc.Output = { "FILE1
                     DATAFILE='myana.ntup'
                     OPT='NEW'
                     TYP='ROOT'" };
```

ROOT

Exercise 3

- Add some more variable in SimpleCell Ntuple
  - Add eta and phi of the cell
  - In .h file, add private data member
  - In initialize(), add variables to ntuple,
  - In execute(), fill eta/phi in ntuple
  - Compile and build
  - Can you find your new variables in ntuple?