# Usage of and Requirements for Gaudi in LHCb

Sébastien Ponce
sebastien.ponce@cern.ch

with input from :

- Gerhard Raven
- Marco Clemencic
- Hadrien Grasland
- Benjamin Couturier

# Context

- LHCb has started to work Run 3 upgrade
- This includes software upgrade
- And a full software trigger @ 40MHz
- TDR due end of 2017
  - so it's demonstrator time

# Outline

Framework

Event Model

Condition Data

Detector Description

# Framework

# Main goal : improve scaling

- Memory usage
  - less is better
  - be cache friendly
  - avoid pointers-to-pointers-to-pointers-to...
    - no pointer to container on the event store
    - use C++11 move semantics
- Multi-thread friendly code
  - using a task model

# Items to consider

1. Data dependencies & control flow
   - necessary for dynamic scheduling
2. Proper const usage
   - const methods are safe to call from multiple threads
3. No non-const static (i.e. global) state
4. Incidents
   - ban usage of beginEvent/endEvent
     - there may be multiple events in flight
5. No (explicit) new/delete!

# Handles everywhere

- Directed a-cyclic graph of data dependencies needed
  - must know before algorithm (tool) executes what it will ask for so that the producer is scheduled prior to its consumers
- must also know which tools will be used by algorithms, and what data those tools request !
- $\rightarrow$ more introspection $\rightarrow$ handles everywhere !

# Handles on anything

- That is using DataHandle on any object
  - not necessarily inheriting from DataObject
- Allows to hide completely the transient event store from users
- And thus to modify it deeply (drop it ?) in the back of the users

```
AnyDataHandle<std::vector<int>> ids
  ("/Event/Test/Ids", Writer, this);
ids.put(vector<int>({42,84}));
```

# Gaudi::Functional

- Many algorithms look like "data in $\rightarrow$ data out"
- Standardize this pattern, and factor out getting and putting the data
  - less code to write
  - more uniform code, easier to understand
  - move maintenance of annoying details to the framework
  - fix bottlenecks once and for all
- Patterns available
  - Consumer, Producer, Filter, Transformer, MultiTransformer, ScalarTransformer

# Gaudi::Functional practical code

```
class MySum: public TransformAlgorithm
    <OutputData(const Input1&, const Input2&)> {
  MySum(const std::string& name, ISvcLocator* pSvc)
  : TransformAlgorithm(name, pSvc,
                       { KeyValue("Input1Loc", "Data1"),
                         KeyValue("Input2Loc", "Data2") },
                       KeyValue("OutputLoc", "Output/Data") )
  {}
  // ...
  OutputData operator()(const Input1& in1,
                        const Input2& in2) const override {
    return in1 + in2;
  }
  // ...
}
```

# Timing

- currently several ways to time code in Gaudi
  - GaudiSequencer
  - Auditors
- they both work and give same result
- but do we want to keep duplication ?
- Most importantly : do they work in multithreaded environment ?

# Event Model

# Ranges and DataHandles

- Ranges are widely used in Gaudi code to avoid duplication of data
- They are a light weight proxies to a subpart of a container
- However they need to be adapted to the usage of DataHandles
- This work has almost been completed by Ben

# Consequences of functionnal approach

- no direct access to TES anymore (no get/put)
- objects stored in TES are unmodifiable
- so cannot be modified/extended
- $\rightarrow$ need for object composition ?

# What remains from the TES ?

- from user's point of view : it's gone
- thanks to Handles, life time of objects can be controlled
  - and objects can be ref counted and deleted when not used anymore
- so objects used by a single algorithm ("consumer") do not even need to enter the TES, they can be moved to their consumer

# Back to object modifications

Case 1 : single consumer

- as seen, the object does not enter the TES
- a new object can thus be created from it with move constructor
- then modified and returned

Case 2 : multiple consumer

- aka concurrent modifications of objects
- simply forbidden as objects are not thread safe

# Condition Data

# A vision for Gaudi

- Gaudi should be aware of conditions
- It should provide a standard interface to them
- It should not dictate how exactly conditions are to be loaded, stored, computed...
  - Can provide architecture + default implementation
  - Must enable a progressive migration from current experiment-specific infrastructure
  - And must allow experiment-specific tuning

# Conditions usage in LHCb

- condition access need to be thread safe
- conditions for different IOVs may be used in parallel
  - but not many (actually, max 2)
  - and this is seldom (every many 1000s events)
- so we do not need an optimized solution

# Detector Description

# Thinking about reworking our geometry

- Current geometry in production for 15 years
  - but too detailed/slow for tracking and simulation
- Simplified geometries implemented by hand with no support from the framework

# Possible evolution

- DD4HEP is an interesting replacement
  - Gaudi integration done for FCC
- However
  - Difficult to map LHCb Detector description to DD4HEP
  - Direct mapping may not even be what we want...

*Geometry migration would be a significant effort*
*With huge return if we can ease multi resolution geometry*