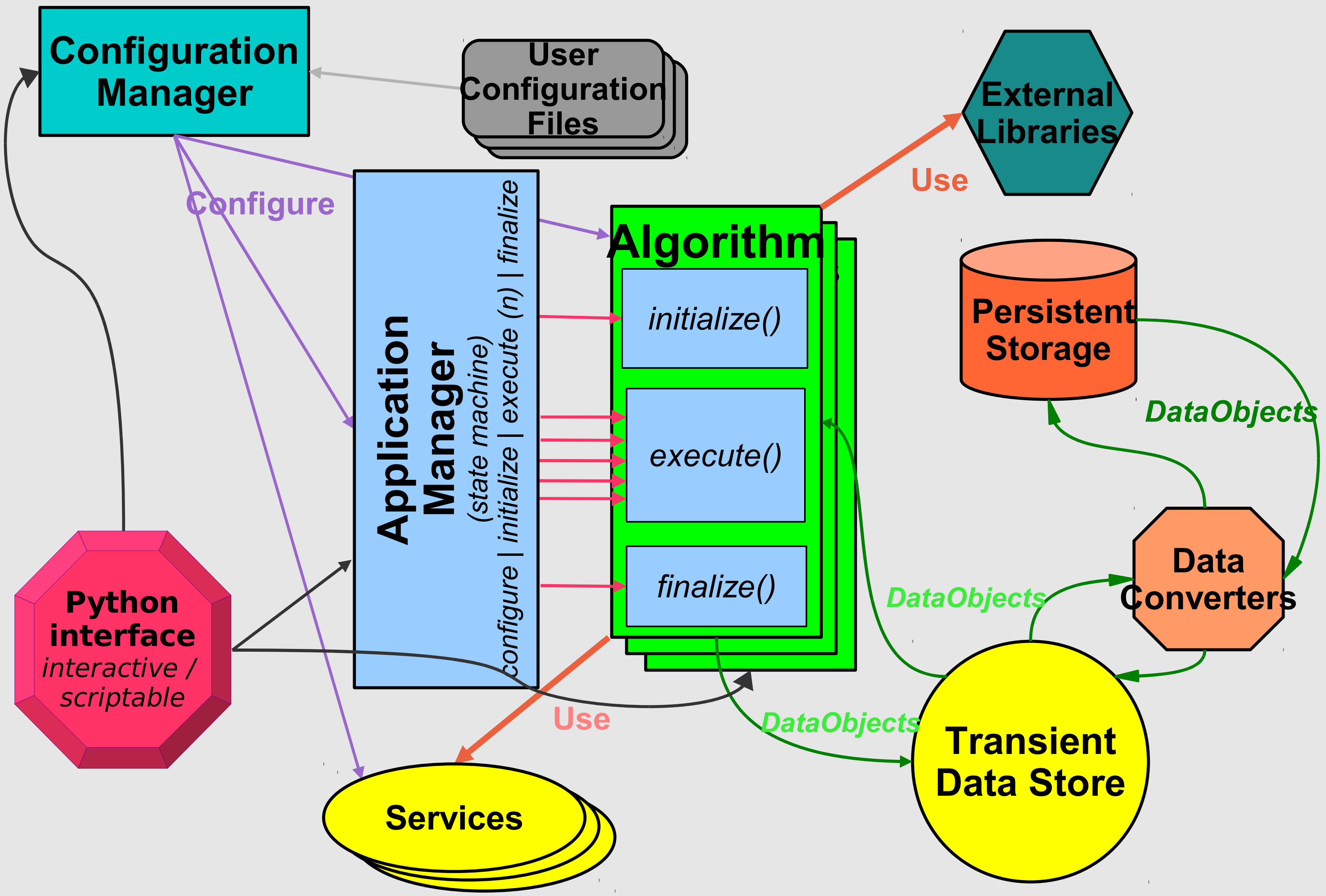
Experiences Running Athena and Gaudi with Multiple Threads

Charles Leggett

Gaudi / Athena Framework



Multi-Threading in Gaudi

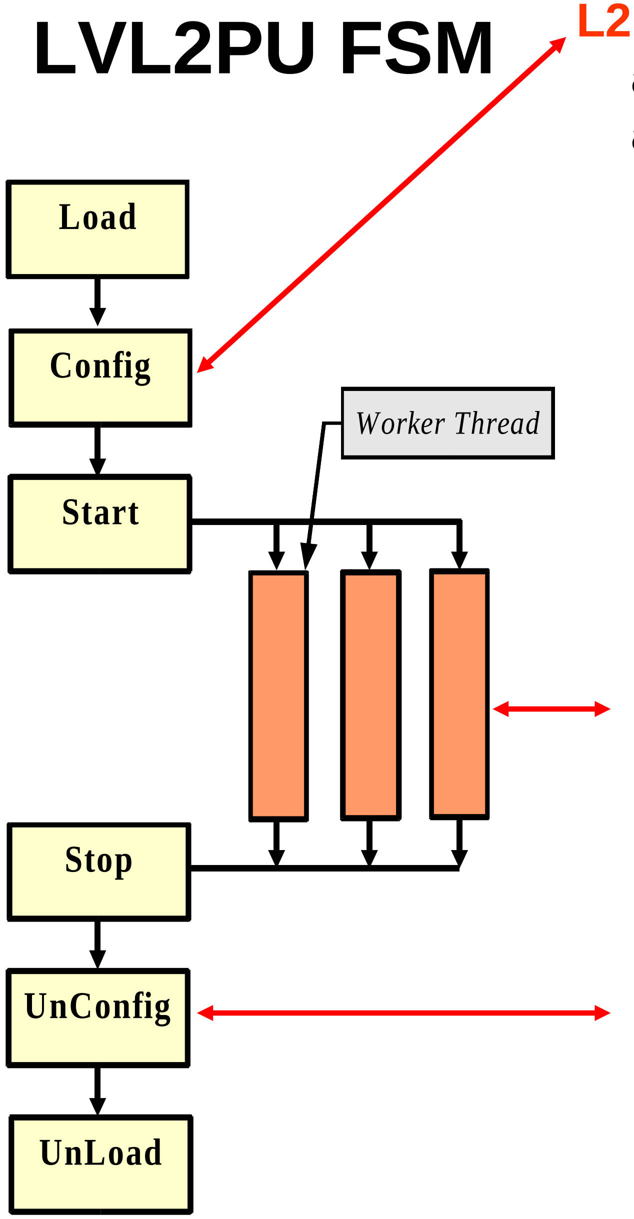
- Hide multi-threading from algorithm developers
  - process entire event in a single thread
- Use normal Gaudi naming convention of <type>/<name> for Algorithms and Services with an appended _<thread#> to distinguish between threads:
  - HelloWorldAlg/MyHelloWorld_1
- All Services and Algorithms which modify data must be thread specific
- Services that wish to share data between threads must be declared at initialization
- Default is to make any automatically created Algorithm or Service thread specific.

GaudiMT and AthenaMT

- GaudiExamples contains a trivial multi-threading example GaudiMT
  - replace main executable that spawns individual worker threads after global configuration
  - specify which services to be shared between threads as a job property

- AthenaMT: Used by HLT level 2 trigger
  - follows TDAQ state machine

Mapping of Gaudi on LVL2PU FSM



L2PU configure: initialize basic Gaudi framework and configure/initialize all requested services and algorithms.

- Configuration = jobOptions.py file
- for every thread an instance of a Gaudi "EventLoopMgr" is created, configured and initialized. The "EventLoopMgr" creates and configures then for its thread all algorithms.
- for every thread all thread specific services are created (specified in configuration)
- AppMgr→configure(); AppMgr→initialize();

L2PU start: in each worker thread the following PESA code is executed

- clear and initialize event store
- store Lvl1 result as root object of the event store
- EventLoopMgr__(threadID)→executeEvent(Lvl1Res)

L2PU UnConfigure: finalize all PESA algorithms and terminate Gaudi application manager.

AppMgr→finalize(); AppMgr→terminate();

Issues with Multi-Threading

- Non-thread safe code:
  - objects shared between services/algorithms to “save” memory
  - globals and statics
- Exception handling with threads
- STL implementations with different compiler versions, especially with strings and allocators
  - solved in more recent compilers (probably?)
- Non-thread safe external libraries

- Tools for debugging

- Users need to be aware of thread safe programming
- Use of shared objects in stores

Issues with Multi-Threading

- Initialization:
  - AthenaMT called Initialize() on each thread separately
    - very time inefficien
  - really want to instantiate Algorithm in “mother”, initialize it, then clone it in each worker thread
- Message passing and Incidents between threads
- Histogramming
- AlgTools: private vs public
  - public shared between threads?
- Detector description (especially LAr) contained shared objects that would be touched by different threads
  - would normally want to have the Detector Description read only and shared between threads
- Could make things work for one release, but would be broken in next one

AthenaMP	**AthenaMT**
sharing done via copy-on-write	must decide *a-priori* what to share between threads
configuration/initialization performed in mother, workers start	no mother process, initialization occurs multiple time
fast-merge of output on finalize	post-finalize merging never addressed
exceptions ok	exceptions not well behaved
message passing / incidents non-trivial	message passing / incidents non-trivial
	need to write thread-safe code from the ground up

Future Prospects of Multi-Threading

- ATLAS abandoned multi-threading 4 years ago, for a reason
- Adding multi-threading to Athena for any non-trivial job will be very, very difficult, if it is to be “maintenance free” between releases
  - a large number of different packages will need to be fixed
- Users will have to be educated in writing thread-safe code
- Framework needs to be written from the ground up with multi-threading / thread safety in mind