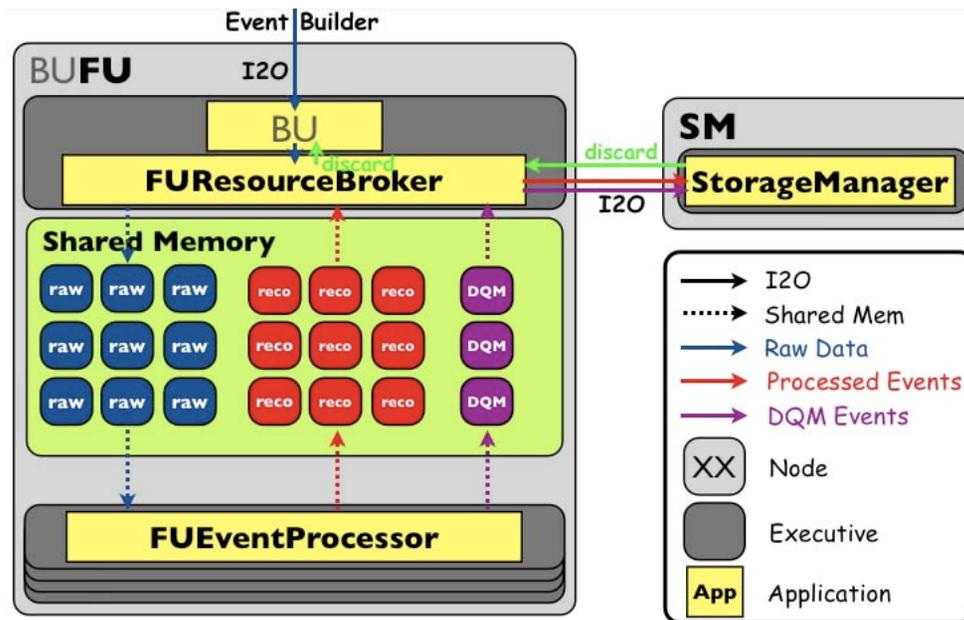
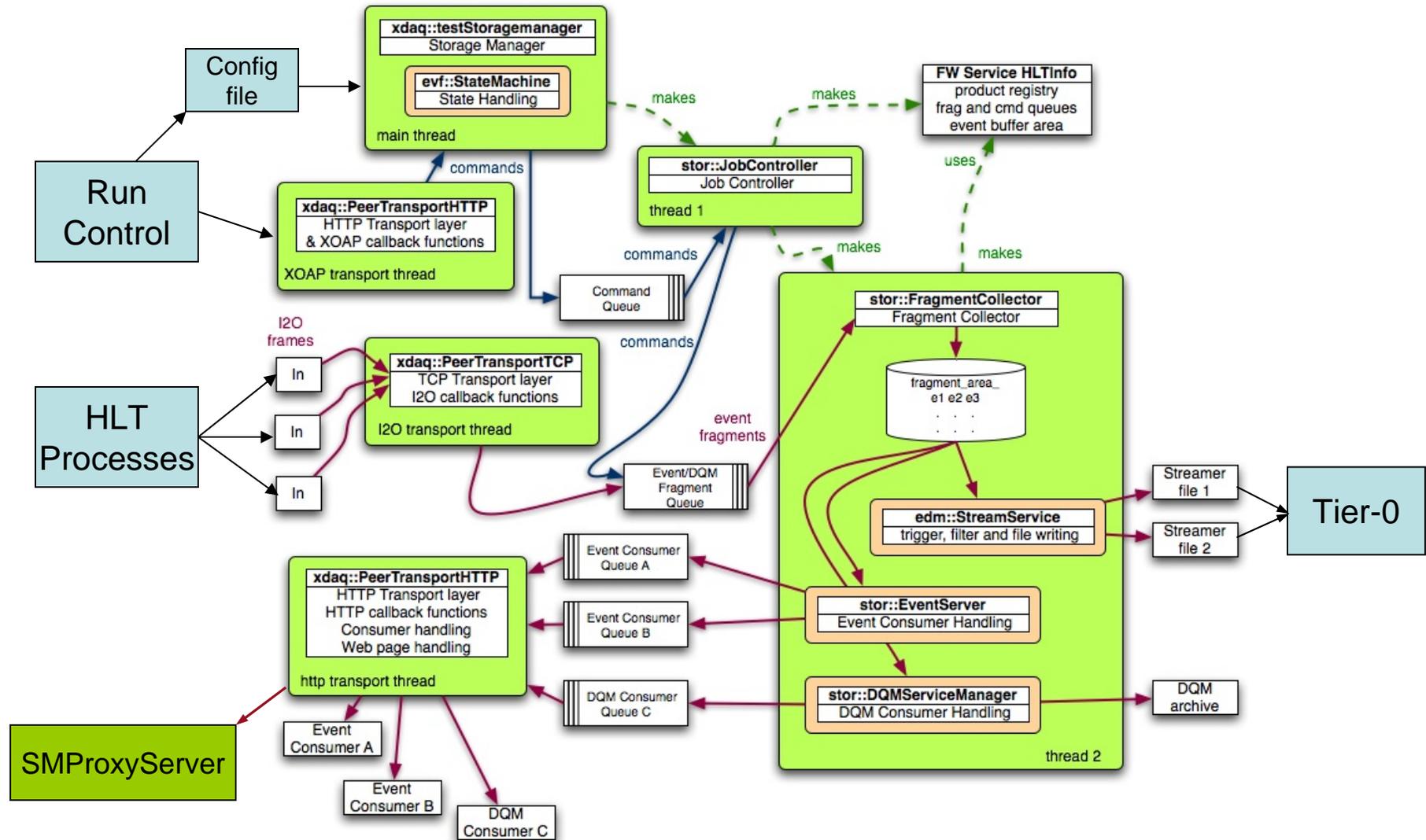
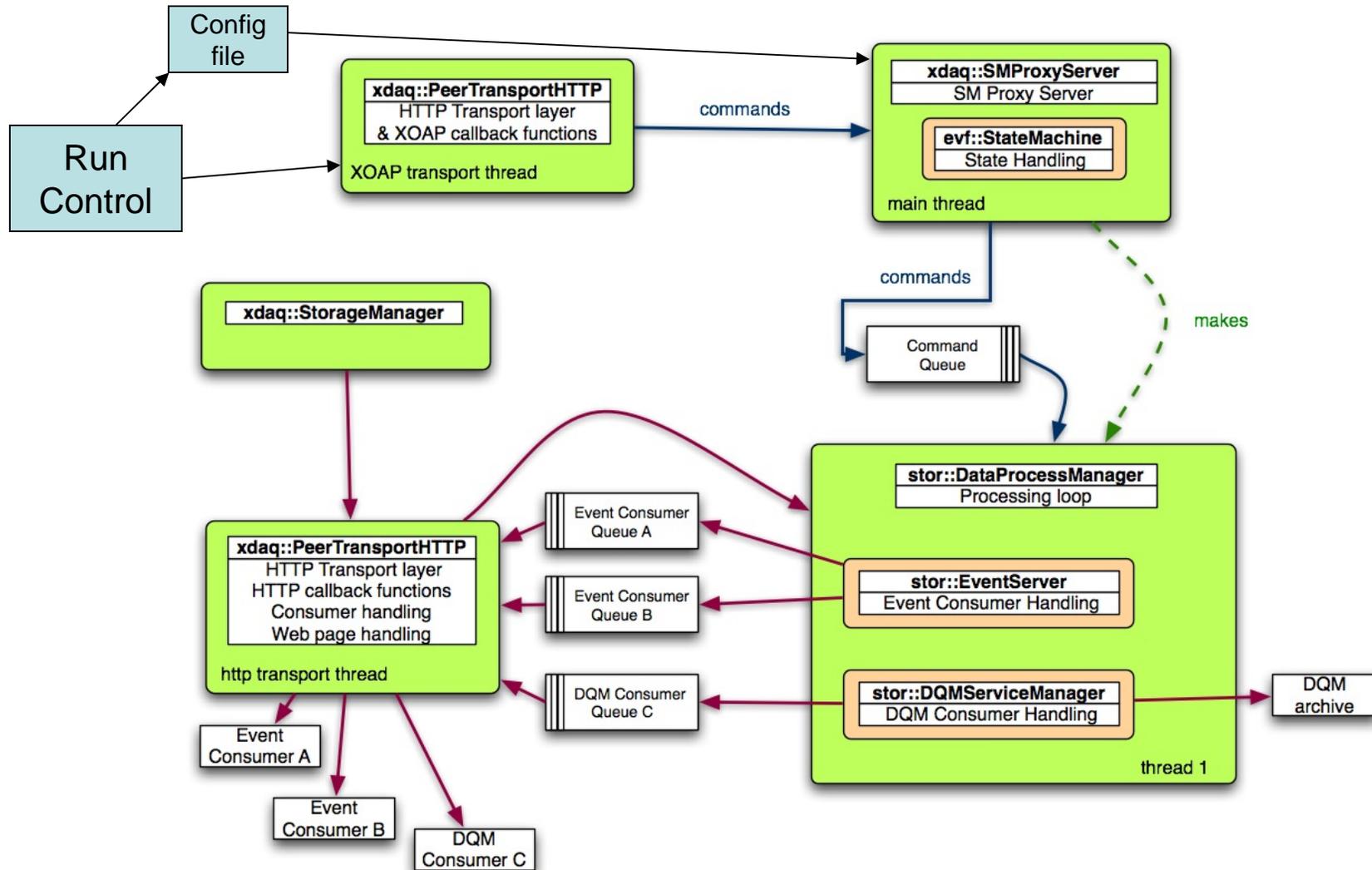


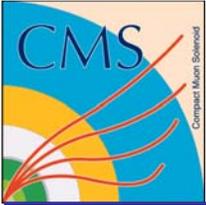
- . - . → I2O messages – push mode

..... → HTTP messages – pull mode









Data Formats



■ Streamer Format

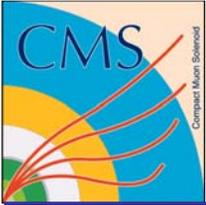
- ◆ In the HLT, the event data is managed by ROOT.
- ◆ The serialized format that is used to transfer events from the HLT to downstream processes is called the streamer format.
- ◆ In order for the data to be converted from streamer format back to ROOT format, information is needed about the structure of the data products that exist in the event. This information is packaged up by the HLT processes at begin run time and sent downstream as INIT messages.

■ FED Raw Data Format

- ◆ To support the transfer of “error events” from the resource broker to the SM, Philipp defined a data format which we have tried to generically label the FED Raw Data format (in case it is ever useful for something other than error events). [Error events have the raw event contents that cause a filter unit hang or crash.]

■ Existing message types: INIT, Event, DQMEvent, ErrorEvent





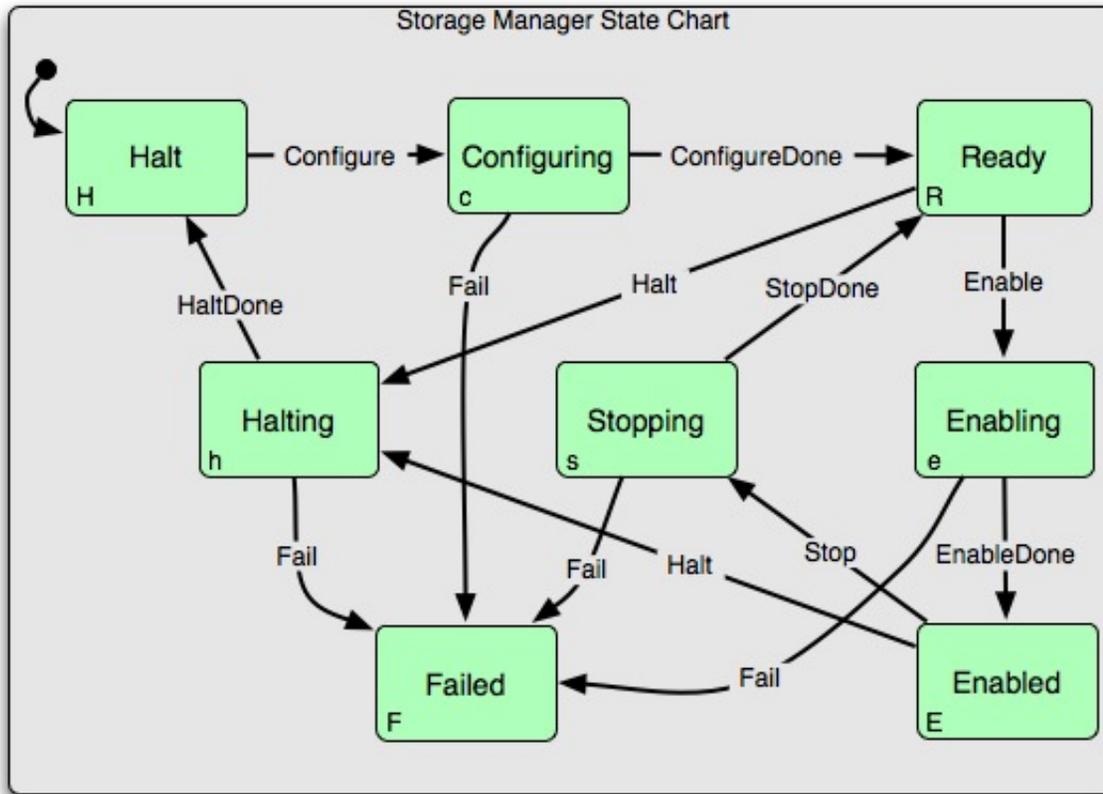
Input Sources & Output Modules



- CMSSW input sources and output modules are provided for reading in and writing out streamer data in various ways:
 - ◆ Writing INIT, Event, and DQMEvent messages to shared memory from the filter unit
 - ◆ Writing Event, DQMEvent, and ErrorEvent data to disk.
 - ◆ Reading Event, DQMEvent, and ErrorEvent data over HTTP from the storage manager or the SM proxy server.

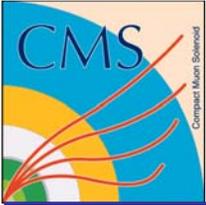


- StorageManager and SMProxyServer State Chart



Enable ~ Begin Run

Stop ~ End Run

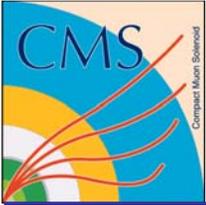


Code Introduction



- The usual place to find a snapshot of the code is http://cmslxr.fnal.gov/lxr/source/?v=CMSSW_2_1_10.
 - ◆ CMSSW_2_1_10 is the most recent version of the software at this time
 - ◆ Other versions are available as you can see when you visit the link
- However, I have also made Doxygen documentation available at <http://home.fnal.gov/~biery/cms/smDoxygen/html/index.html>.
 - ◆ This has more up-to-date code
 - ◆ Doxygen generates some diagrams that might be useful
- If a tar file containing a recent snapshot of the code would be useful, that can be provided as well.
- The following slides describe some of the classes that are responsible for different parts of the SM functionality.



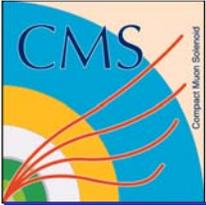


Code Introduction



- The primary class for the storage manager is `stor::StorageManager` in the `EventFilter/StorageManager` package.
 - ◆ Things you will find in `StorageManager.cc...`
 - Methods to process I2O fragments when they are received. `INIT`, `Event`, `DQMEvent`, and `ErrorEvent` messages are transmitted from the filter units to the SM as I2O fragments.
 - Methods to handle transitions between states [`configuring()`, `enabling()`, `stopping()`, and `halting()`]
 - Methods to generate monitoring web pages available through XDAQ.
 - Creation of the threads and worker classes used by the SM.
- The primary class for the proxy server is `stor::SMProxyServer` in the `EventFilter/SMProxyServer` package.
 - ◆ Things you will find in `SMProxyServer.cc...`
 - Methods to handle transitions between states [`configuring()`, `enabling()`, `stopping()`, and `halting()`]
 - Methods to generate monitoring web pages available through XDAQ.



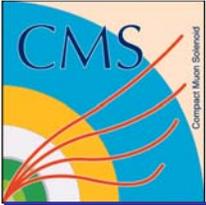


Event Flow – HLT to SM



- Events are written to shared memory in the filter unit using the `edm::StreamerOutputModule` class in the `IOPool/Streamer` package and the `edm::FUShmOutputModule` class in the `EventFilter/Modules` package.
 - ◆ The CMSSW output module that is specified in filter unit configurations is `ShmStreamConsumer`, and this output module is a typedef of `edm::StreamerOutputModule<edm::FUShmOutputModule>`
- `edm::StreamerOutputModuleBase::write()`
 - ◆ → `edm::StreamerOutputModule::doOutputEvent()`
 - ◆ → `edm::FUShmOutputModule::doOutputEvent()`
 - ◆ → `evf::FUShmBuffer::writeRecoEventData()`
- From the shared memory pool, the resource broker breaks the events into I2O fragments and sends them to the storage manager.



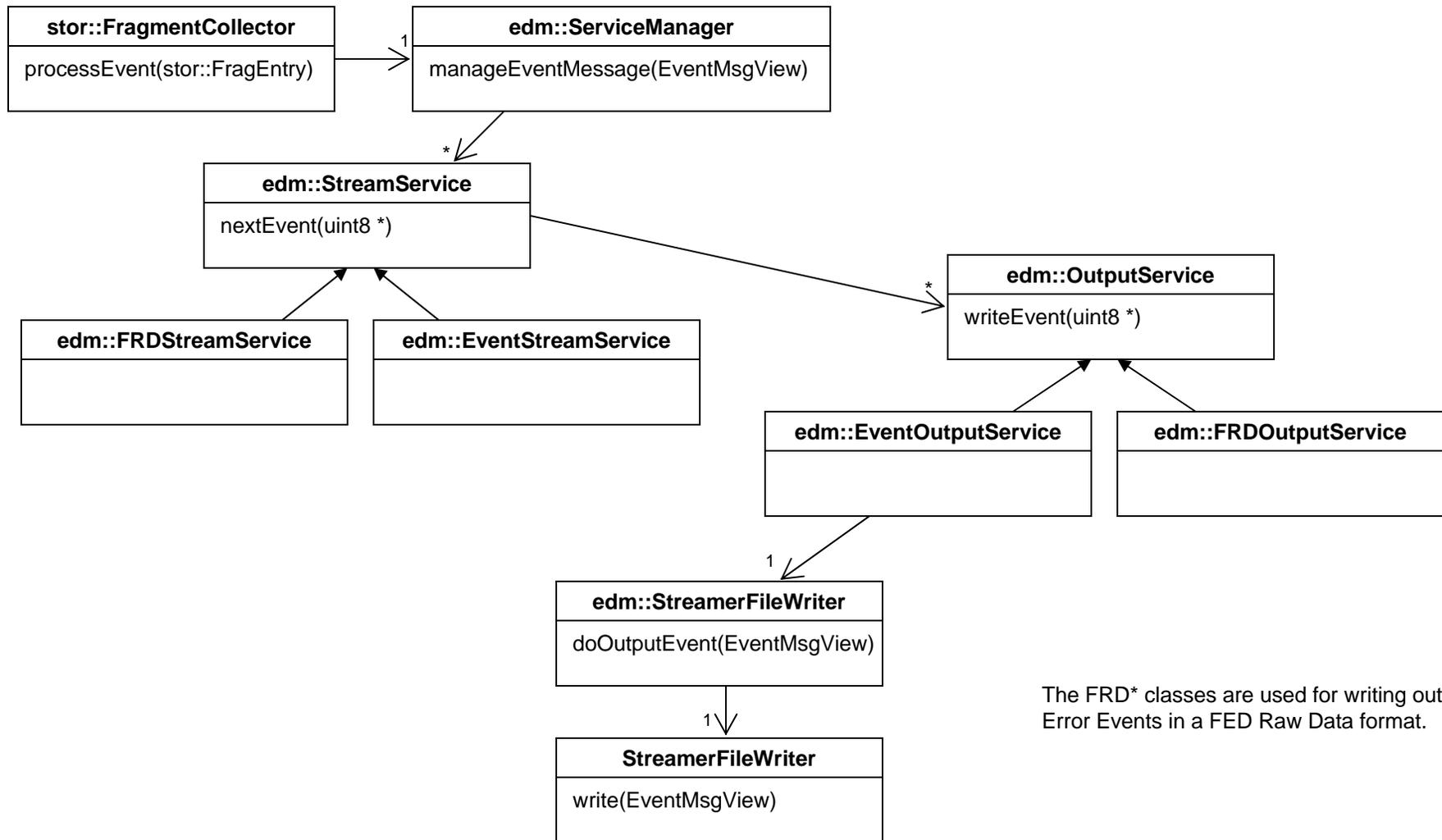


Event Flow – SM to Disk

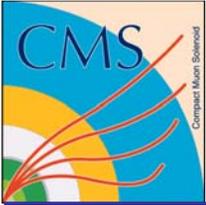


- The event fragments are received by `stor::StorageManager::receiveDataMessage()` (EventFilter/StorageManager).
- The fragments are placed on the fragment queue.
- The `stor::FragmentCollector` class (EventFilter/StorageManager) assembles the fragments (see `processFragments()` and `processEvent()`).
- `stor::FragmentCollector::processEvent()`
 - ◆ → `edm::ServiceManager::manageEventMessage()`
 - ◆ → `edm::EventStreamService::nextEvent()`
 - ◆ → `edm::EventOutputService::writeEvent()`
 - ◆ → `edm::StreamerFileWriter::doOutputEvent()`
 - ◆ → `StreamerOutputFile::write()`





The FRD* classes are used for writing out Error Events in a FED Raw Data format.

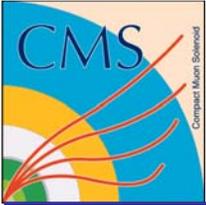


Event Flow – SM to Consumers

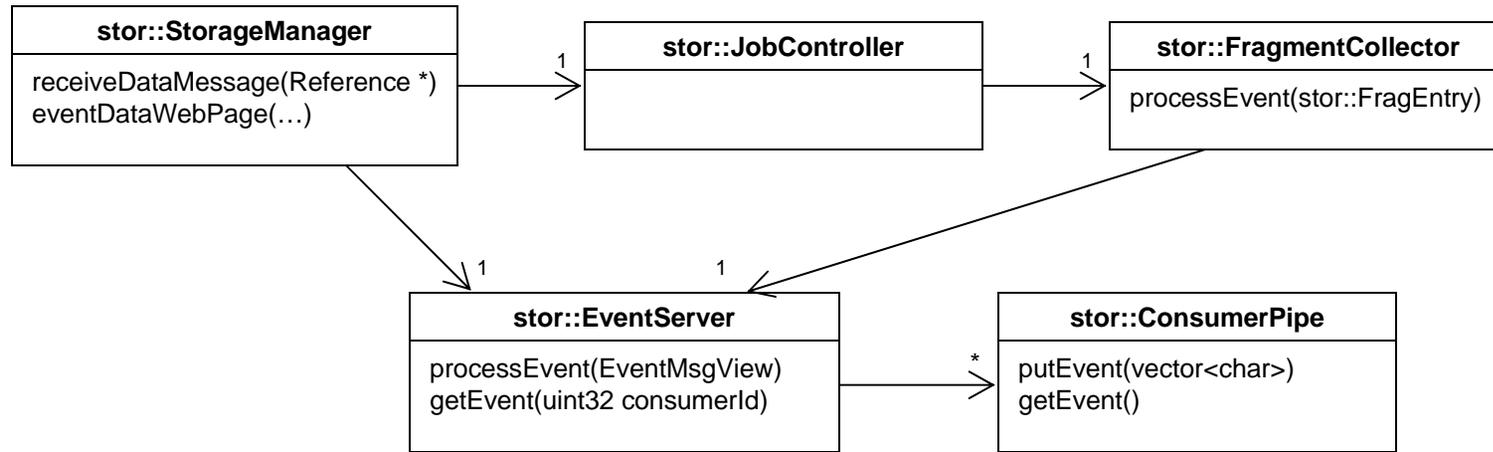


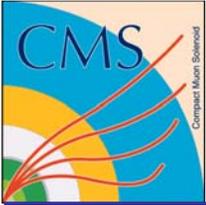
- Events travel from the filter units to the SM fragment queue as listed on the previous pages.
- In the SM, they are stored on queues for serving to consumers with the following calls:
 - ◆ `stor::FragmentCollector::processEvent()`
 - ◆ `→ stor::EventServer::processEvent()`
 - ◆ `→ stor::ConsumerPipe::putEvent()`
- Asynchronous calls to the SM from consumers are processed in the following way:
 - ◆ `stor::StorageManager::eventdataWebPage()`
 - ◆ `→ stor::EventServer::getEvent()`
 - ◆ `→ stor::ConsumerPipe::getEvent()`
- The Proxy Server (SMPS) acts like a consumer to the SM. Inside the SMPS, events are fetched by the DataProcessManager:
 - ◆ `stor::DataProcessManager::processCommands()`
 - ◆ `→ stor::DataProcessManager::getEventFromAllSM()`
 - ◆ `→ stor::DataProcessManager::getOneEventFromSM()`
 - ◆ `→ stor::EventServer::processEvent()`
 - ◆ `→ stor::ConsumerPipe::putEvent()`
- Asynchronous calls to the SMPS from consumers are processed in the following way:
 - ◆ `stor::SMProxyServer::eventDataWebPage()`
 - ◆ `→ stor::EventServer::getEvent()`
 - ◆ `→ stor::ConsumerPipe::getEvent()`
- Event Consumers fetch events using the EventStreamHttpReader input source:
 - ◆ `edm::EventStreamHttpReader::getOneEvent()`





Event Flow – SM to Consumers



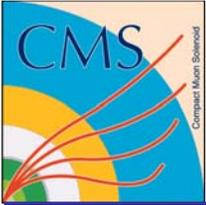


DQM Event Flow – SM to SMPS to Disk



- For simplicity, the following notes assume that the SM passes DQMEvents directly to the SM proxy server instead of summing updates and sending the result.
- DQMEvents travel from the filter units to the SM fragment queue in a similar fashion to physics events.
- In the SM, they are stored on a queue for retrieval by the proxy server:
 - ◆ `stor::FragmentCollector::processDQMEvent()`
 - ◆ `→ stor::DQMServiceManager::manageDQMEventMsg()`
 - ◆ `→ stor::DQMEventServer::processDQMEvent()`
 - ◆ `→ stor::DQMConsumerPipe::putDQMEvent()`
- Asynchronous calls to the SM from proxy server are processed in the following way:
 - ◆ `stor::StorageManager::DQMeventdataWebPage()`
 - ◆ `→ stor::DQMEventServer::getDQMEvent()`
 - ◆ `→ stor::DQMConsumerPipe::getDQMEvent()`
- Inside the SMPS, events are fetched by the DataProcessManager:
 - ◆ `stor::DataProcessManager::processCommands()`
 - ◆ `→ stor::DataProcessManager::getDQMEventFromAllSM()`
 - ◆ `→ stor::DataProcessManager::getOneDQMEventFromSM()`
 - ◆ `→ stor::DQMServiceManager::manageDQMEventMsg()`
 - ◆ `→ stor::DQMInstance::updateObject()`
 - ◆ `→ DQMServiceManager::writeAndPurgeDQMInstances()`
 - ◆ `→ stor::DQMInstance::writeFile()`



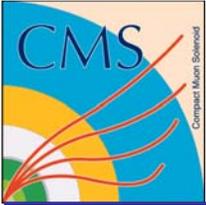


Web Pages for Monitoring



- Variables of StorageManager and SMPProxyServer accessible through XDAQ Infospace or XDAQ application web pages
- In the SM, we have the following web monitoring pages:
 - ◆ `stor::StorageManager::defaultWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::StorageManager::rbsenderWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::StorageManager::streamerOutputWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::StorageManager::eventServerWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::StorageManager::consumerListWebPage(xgi::Input, xgi::Output)`
- In the Proxy Server (SMPS) we have:
 - ◆ `stor::SMPProxyServer::defaultWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::SMPProxyServer::smsenderWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::SMPProxyServer::eventServerWebPage(xgi::Input, xgi::Output)`
 - ◆ `stor::SMPProxyServer::consumerListWebPage(xgi::Input, xgi::Output)`



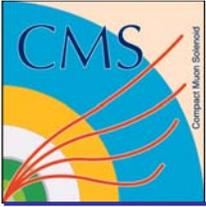


Message Definition Classes



- Classes for constructing and parsing streamer messages (all of these are in the IOPool/Streamer package):
 - ◆ **INIT messages:**
 - InitMsgBuilder
 - InitMsgView
 - ◆ **Event messages:**
 - EventMsgBuilder
 - EventMsgView
 - ◆ **DQMEvent messages**
 - DQMEventMsgBuilder
 - DQMEventMsgView
 - ◆ **ErrorEvent messages:**
 - FRDEventMsgView





Backup Slides

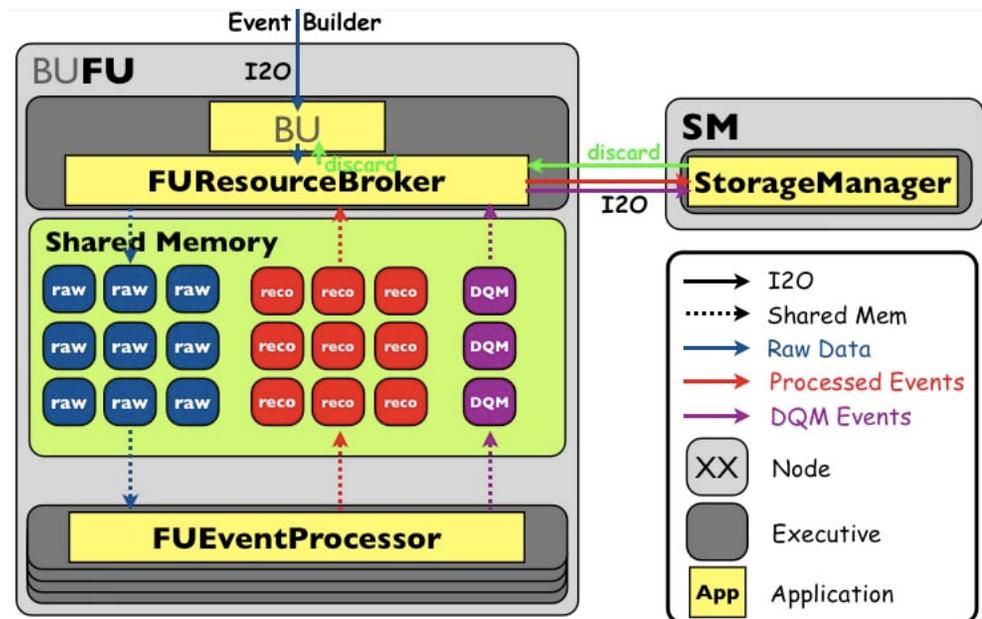


- Output modules, Input modules, and Output service all use standard CMS EDM framework base classes, supported by the CMS EDM Framework Group
 - ◆ Major components are the serialize and deserialize utilities, and format
- Streamer output module writes to shared memory handled by Resource Broker, (supported by DQM group). Network transfer via I2O (XDAQ)
- Other data network transfer uses http (cURL and XDAQ http server)

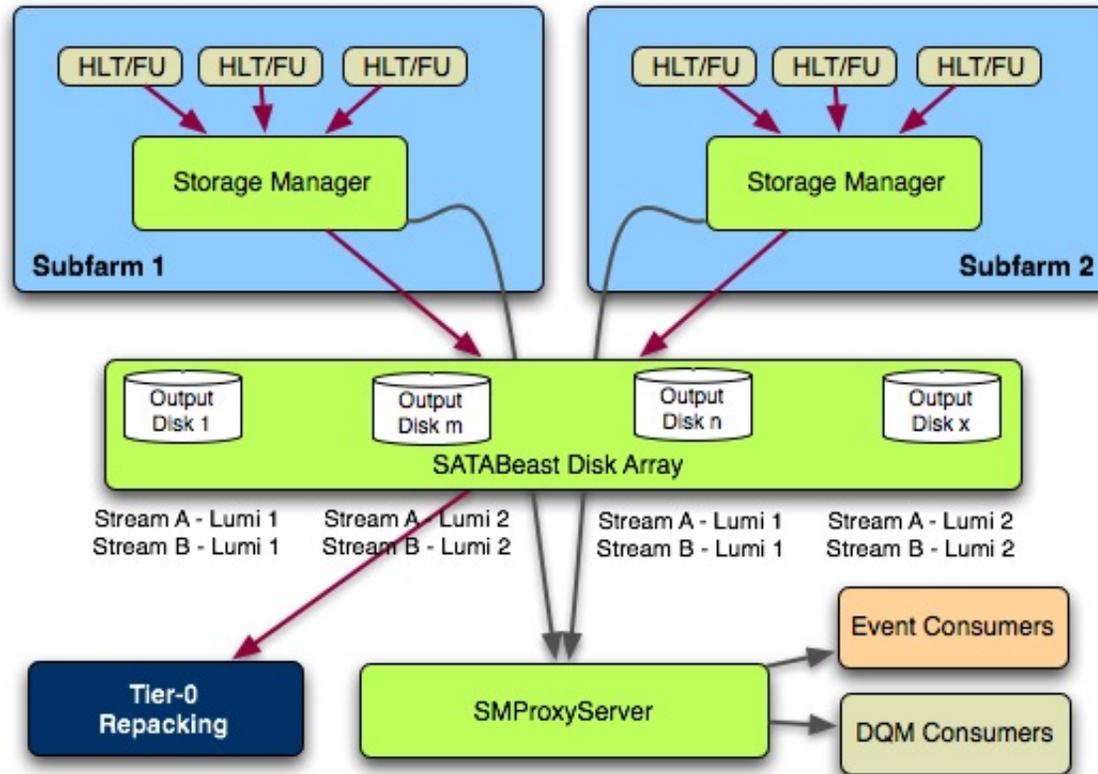
Serialization and deserialization using Root

The Root class descriptions are used to convert event-by-event the object data into a serial byte stream

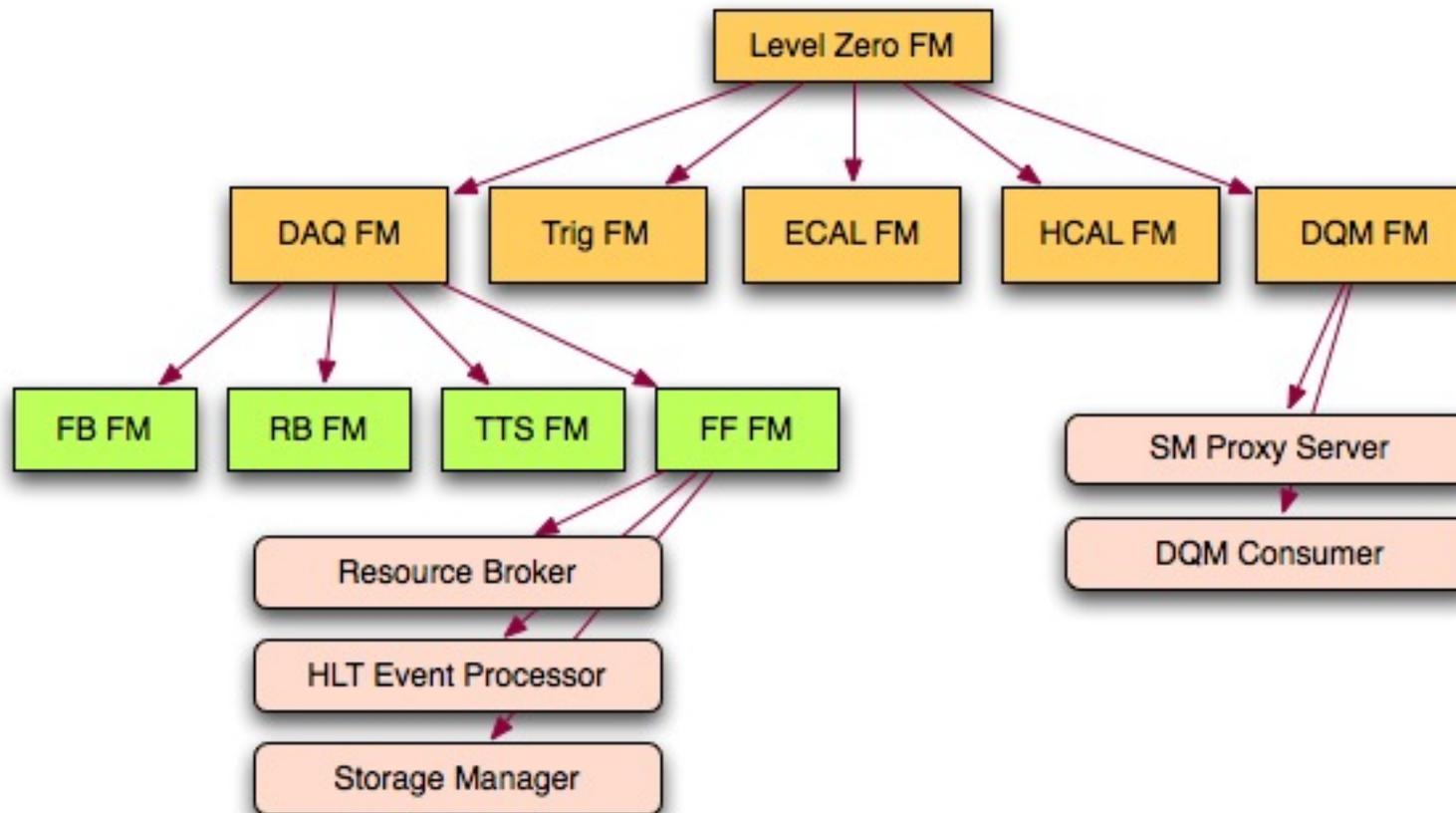
Various “messages” are formed and either written to file or sent over the network

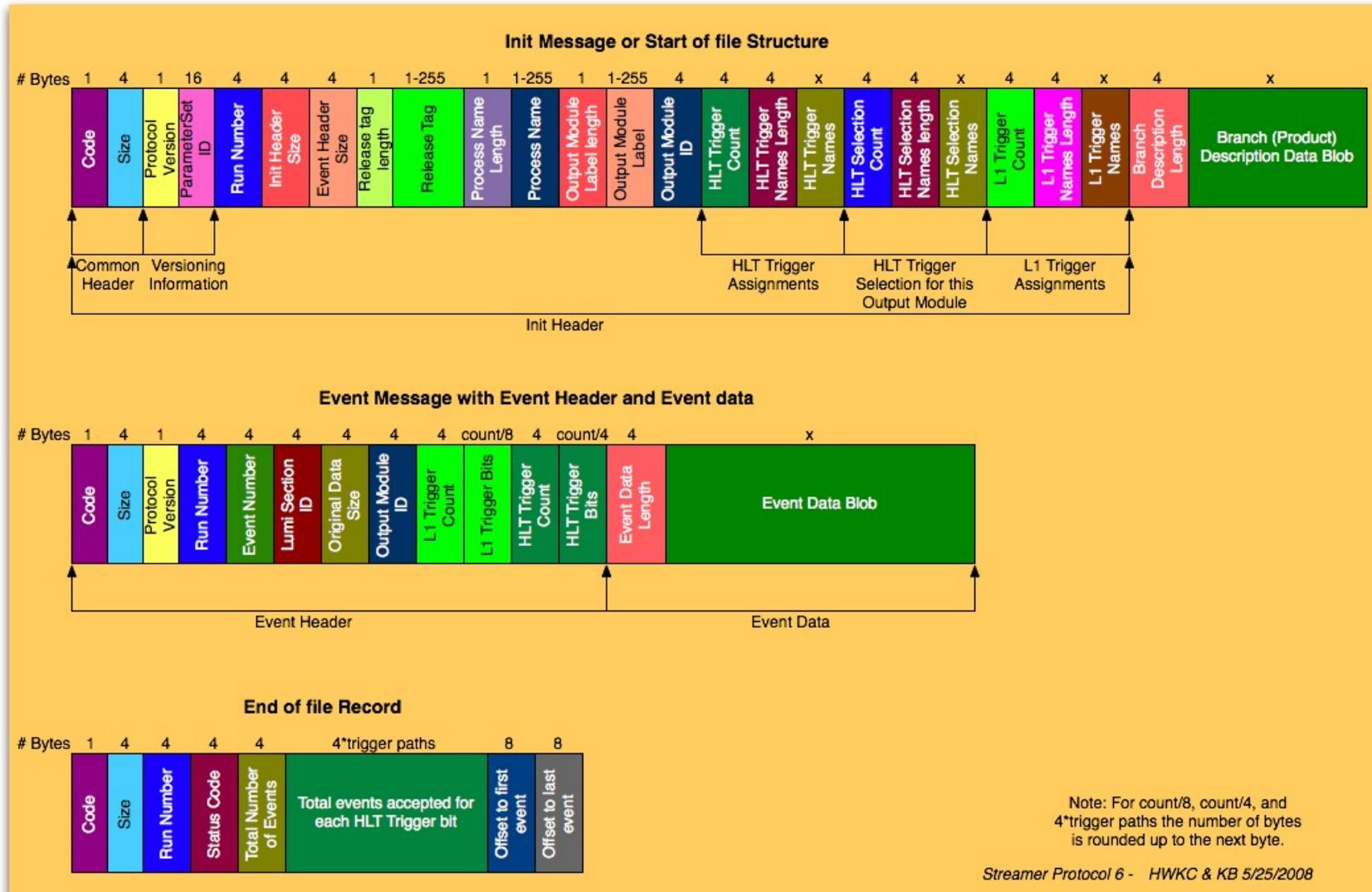


- Two XDAQ applications: StorageManager & SMProxyServer



- Illustration of Function Manager hierarchy





- Streamer event data file format and index file format. (Index files to aid Tier-0 processing)
- Consists of concatenated serialized event data messages (i.e. same code/format for network and files)
- A file format for temporary files used within Tier-0

