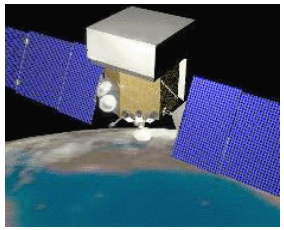
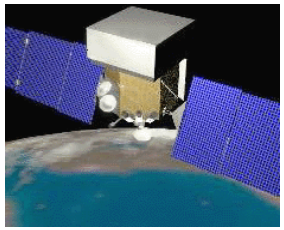
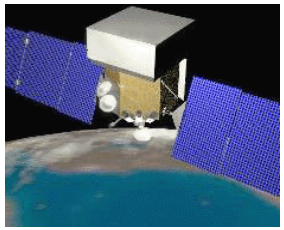


GLAST Event Data Model and Persistency



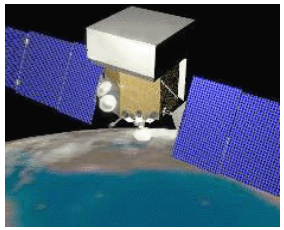
Data Stores

- *"All data which comes from persistent storage, or which is transferred between algorithms, or which is to be made persistent must reside within a data store."*
- GAUDI implements four data stores:
  - event
  - detector
  - histogram
  - n-tuple



Event Data Store

- Of primary interest is the Event Data Store, often referred to as the TDS (transient data store).
- As the name implies, this store deals with Event data. At the end of each event, the TDS is cleared and all allocated memory is released.
- Our Event Data structure strongly resembles the LHCb model.

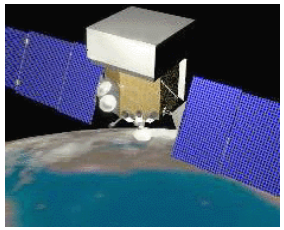


GLAST TDS Structure

/Event

- **/MC**
  - /McParticleCol
  - /McPositionHitCol
  - /McIntegratingHitCol
- **/Digi**
  - /AcdDigiCol
  - /CalDigiCol
  - /TkrDigiCol
- **/AcdRecon**
- **/CalRecon**
  - /CalXtalRecCol
  - /CalClusterCol
- **/TkrRecon**
  - /TkrClusterCol
  - /TkrPatRecCol
  - /TkrFitTrackCol
  - /TkrVertexCol

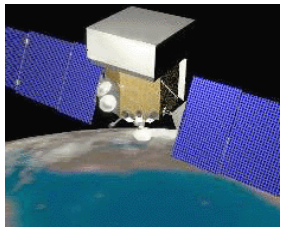
All TDS classes reside in the Event package



What Can be put on the Data Store?

- All objects must derive from Gaudi's DataObject or ContainedObject class.
- Two Implemented ContainedObject classes: ObjectVector and ObjectList
- Each class has a classId.
- Checklist

  Do not delete objects you have registered
  Do not delete contained objects you have registered
  Do not register local objects (must use new)
  Do not delete objects retrieved via retrieveObject
  Delete objects allocated on heap which are not registered



How does Data get on the TDS?

- Beginning of each event, /Event object is created and placed on TDS.
- Branches of the TDS tree are created during event execution
  - A Gaudi Component calls registerObject to put an object on the TDS.

```
Event::McParticleCol* pTdsCol = new Event::McParticleCol;
sc = eventSvc()->registerObject(EventModel::MC::McParticleCol, pTdsCol);
```

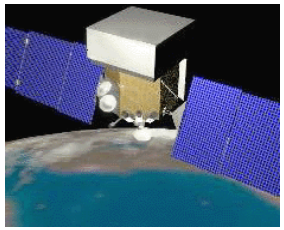
  - A request is made via a retrieveObject or SmartDataPtr call

If the data is on the TDS, the object is returned to the caller.
If the data is currently unavailable, the appropriate Persistency Service is called to retrieve the data from a persistent store.

```
DataObject* pnode =0;
sc = eventSvc()->retrieveObject(EventModel::TkrRecon::Event, pnode);
```

OR

```
SmartDataPtr<Event::McPositionHitVector> posHits(eventSvc(), EventModel::MC::McPositionHitCol);
```

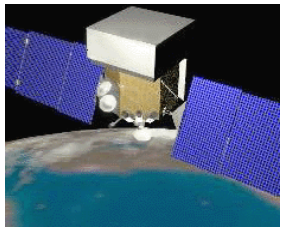


Conversion Process

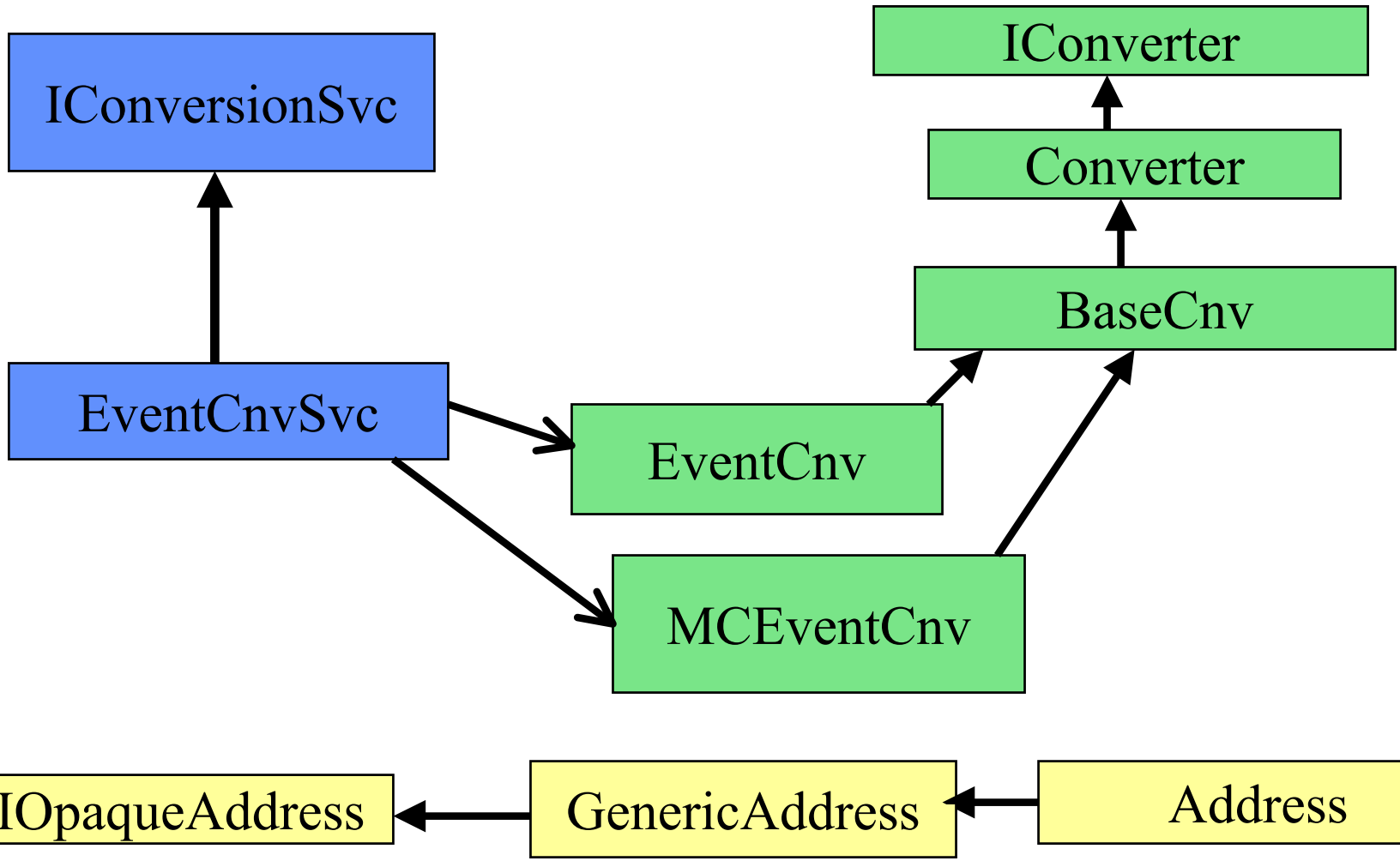
- Collaboration between
  - A service derived from IConversionSvc
    coordinates conversion by calling appropriate IConverter
  - A set of converters each derived from IConverter
    does the work of converting transient type <==> persistent type
  - IOpaqueAddresses
    creates association between converters and paths on TDS

For more information, see Chapter 13 of the Gaudi Developers Guide:
http://proj-gaudi.web.cern.ch/proj-gaudi/GDG/v2/Output/GDG_Converters.html#1010951

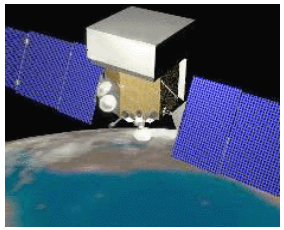
- Our persistency service is a placeholder.



GLAST Persistency Service

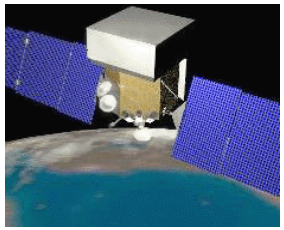


Resides in GlastSvc package



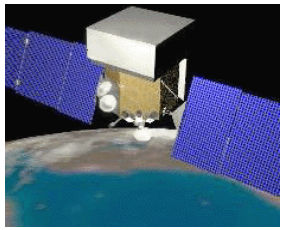
How does it work?

- Add EventCnvSvc to the list of EventPersistencySvc in the jobOptions
  EventPersistencySvc.CnvServices = {"EventCnvSvc"};
- EventCnvSvc::initialize will load all converters that match its storage type.
- As each converter is loaded, an association between TDS path and converter is created using the IOpaqueAddress interface.
- When the Persistency Service is asked to retrieve data for the TDS – the appropriate converter is called based on the IOpaqueAddress.



TDS and Converter Questions

- Our implementation is strongly based on LHCb's.
- There are parts that are unnecessary for our purposes.
- We should be able to simplify our implementation.
- Gaudi Object Description and Introspection
  Can this be used to avoid the explicit definition DataObject classes?
- Should we consider using Atlas' StoreGate?

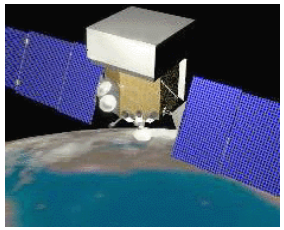


RootIo – The rest of the story

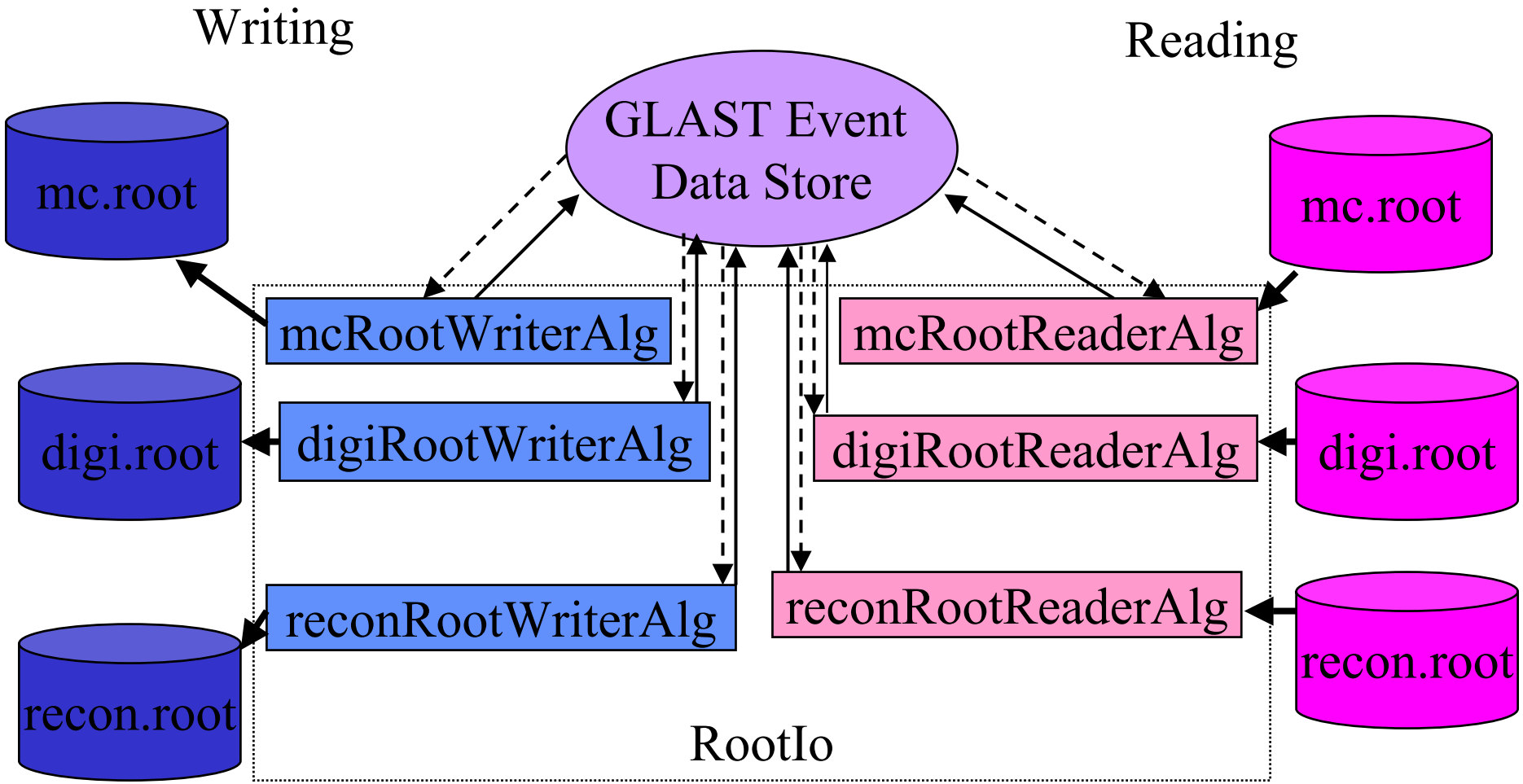
- GLAST Persistency Service not fully utilized
- Why?

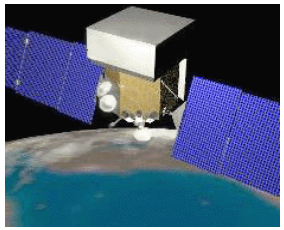
  We want real ROOT I/O.

  Gaudi's default support for ROOT uses TBlobs.
- Short-term solution
  Gaudi algorithms handle ROOT I/O
  The package is called RootIo.
- Our ROOT classes mirror our TDS classes.



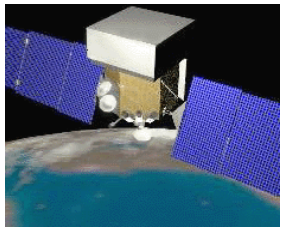
RootIo


Writing
Reading
GLAST Event Data Store
mc.root
mc.root
mcRootWriterAlg
mcRootReaderAlg
digi.root
digiRootWriterAlg
digiRootReaderAlg
digi.root
reconRootWriterAlg
reconRootReaderAlg
recon.root
recon.root
RootIo



Problems with RootIo

- Use of algorithms is inconsistent with the spirit of Gaudi's Persistency Service.
- Does not provide fine control over what is read/written – it's all or nothing as currently implemented.
- Monolithic algorithms are more difficult to maintain versus light weight converters.



ROOT Persistency Service

- Athena has produced a "real" ROOT service.
  http://www.usatlas.bnl.gov/computing/software/db/rootio.html
  - ROOT I/O
  - ROOT interactive session by demand
  - ROOT share library dynamic loading by demand
  - ROOT control over the Gaudi algorithms
- Using this example, we plan to develop our own ROOT Persistency Service.