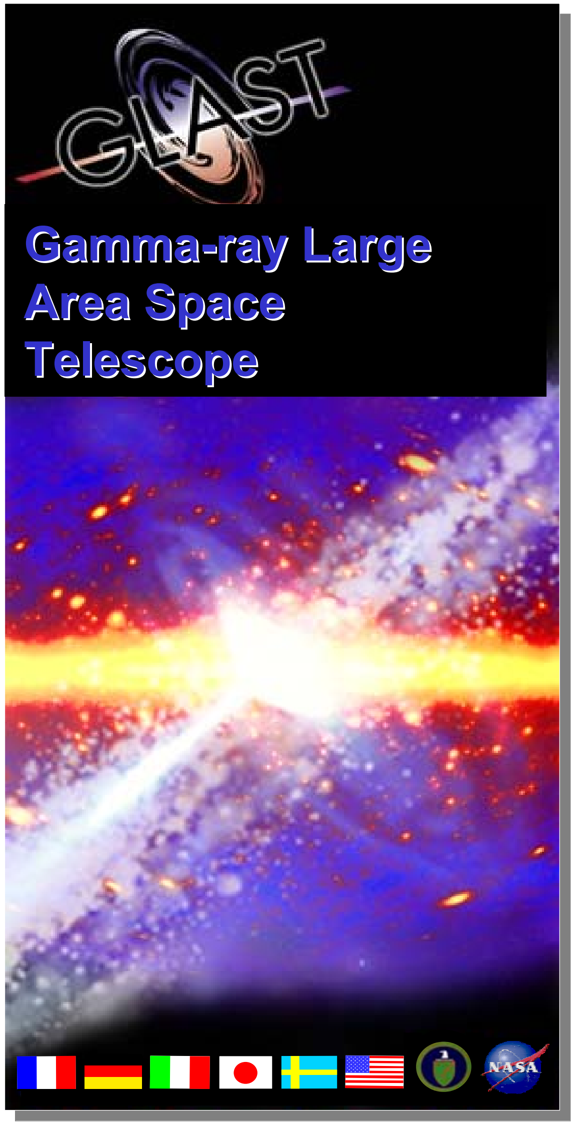


Calibration Infrastructure for the GLAST LAT

Joanne Bogart
Stanford Linear Accelerator Center
jrb@slac.stanford.edu

http://www-glast.slac.stanford.edu/software

Contents

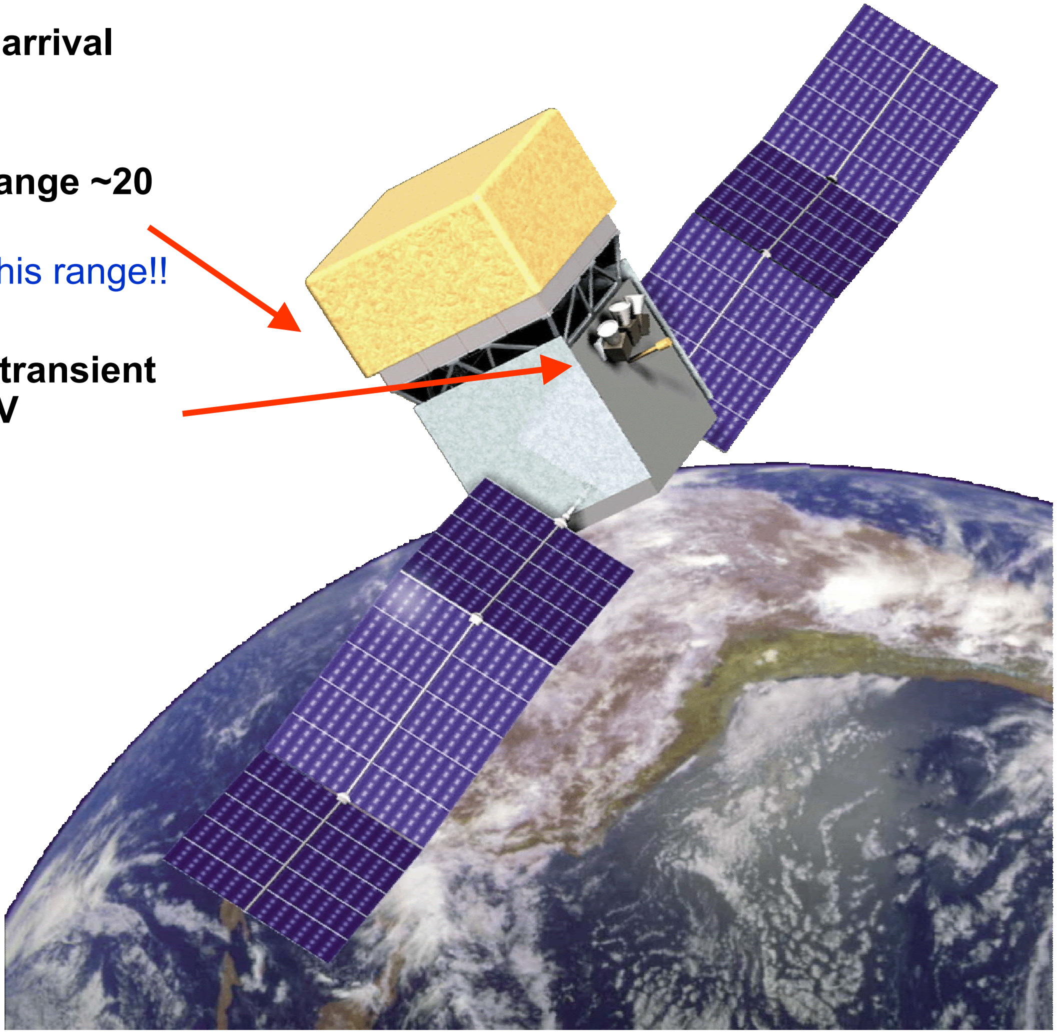
GLAST Mission

GLAST measures the direction, energy and arrival time of celestial gamma rays

-LAT measures gamma-rays in the energy range ~20 MeV - >300 GeV

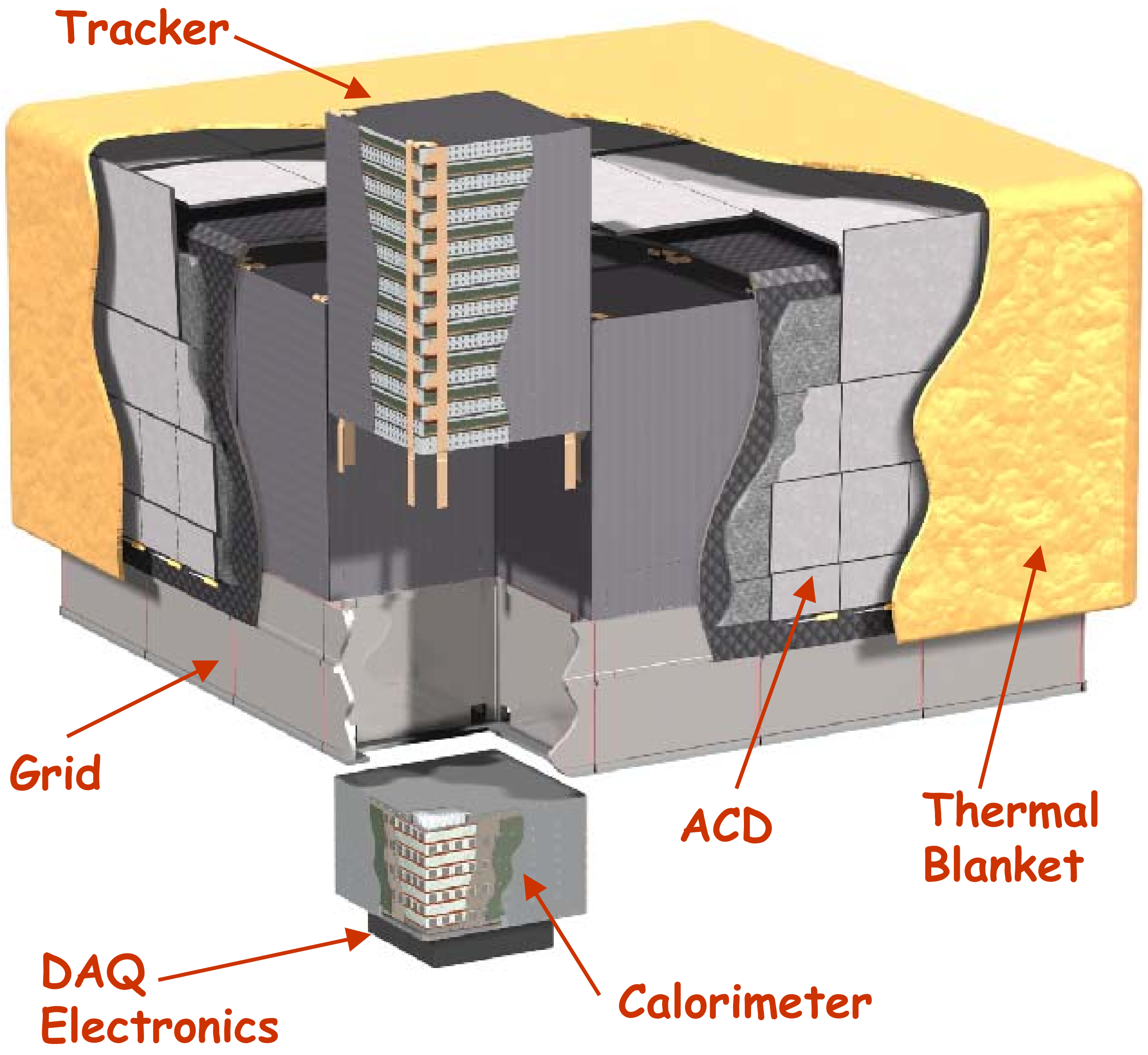
- There is no telescope now covering this range!!

- GBM provides correlative observations of transient events in the energy range ~20 keV – 20 MeV



Launch:	September 2006 Florida
Orbit:	550 km, 28.5^o inclination
Lifetime:	5 years (minimum)

GLAST Instrument: Large Area Telescope (LAT)



- Array of 16 identical "Tower" Modules, each with a tracker (Si strips) and a calorimeter (CsI with PIN diode readout) and DAQ module.
- Surrounded by finely segmented Anti-Coincidence Detector (plastic scintillator with PMT readout).

What do we mean by "calibration" ?

Roughly speaking, we mean information about the detector which can vary with time and is required to interpret the raw readout. The boundaries are still fuzzy, but will include at a minimum

- Hardware status (hot, dead, etc.) for components of the three subdetectors: calorimeter, tracker, and ACD.
- Tracker alignment constants
- Parameters needed to convert from electronic readout to physical units (thresholds, gains, position-dependent light attenuation in calorimeter crystals, ...)

and will *not* include description of the ideal detector geometry, which is managed by a separate facility.

Infrastructure Requirements

- **Accommodate variety of data types, as in previous slide**
- **Handle data for prototypes as well as flight instrument**
  - Prototypes have the same components, but in different numbers (including zero)
- **Support, to varying degrees**
  - clients adding new datasets (coach)
  - clients wishing to track hardware performance (coach)
  - [Gaudi] event reconstruction and analysis clients (1st class)
- **For event analysis,**
  - require transparent update as event data timestamp leaves validity interval of in-memory constants
  - support access to multiple “flavors” of a single data type concurrently
- **Support at least XML and ROOT persistent forms**
- **Portability for readers: full capability for anyone with network access; limited support for development on desert islands.**

Infrastructure Non-requirements

- Don't need to provide easy access to subset of a particular calibration data set.
  - Simplifies TDS structure and conversion process
- Although must handle prototype instruments as well as flight instrument, may assume that any single analysis job only cares about one instrument.
- Conversion in the Gaudi* sense from in-memory form to persistent form is not required (though Gaudi applications may generate persistent calibration data sets other ways).
  - Conversion *from* persistent form must happen transparently during event analysis. Conversion *to* persistent form is the result of an explicit request.

*Software framework designed for HEP or HEP-like event analysis. Some familiarity is assumed for this talk.

Data and Metadata

- The ability to look things up looms large in this system. To expedite this, we distinguish the bulk data from the metadata.

- Bulk data is what typical applications care about: which strips are dead, what is the gain of each calorimeter channel, etc.

- Metadata is information *about* a particular calibration bulk data set. It comes in several categories:
  - selection information used to determine which is the desired dataset, such as calibration type, instrument, validity interval
  - conversion information used to find and read in the bulk data such as file spec and physical format type
  - **miscellany**: other information primarily for browsing or for creating summaries for human readers, such as description of conditions when calibration was done.

Most Metadata Fields

Calibration type	TKR alignment, CAL gains
Flavor	vanilla, ideal, digi, ...
Serial number	automatically assigned; unique
Data format version	for schema evolution (someday)
Data identifier	Where to find the data; e.g., file spec.
Validity start time	Compare to event time
Validity end time	ditto
Completion time	When procedure generating data completed
Instrument	One LAT (flight), CU, EM, ... (prototypes)
Procedure level	Production, development, test, superseded..
Calibration status	Completion status: OK, aborted, ...
Data format	XML or ROOT
Input description	String describing input to the calibration procedure
Comment field	String for anything else

Searching the Metadata

Typically want to find the “right” calibration for a particular event and a particular kind of analysis...

```
/** Return serial number for calibration which is best match
  to criteria. */
Metadata::eRet calibUtil::Metadata::findBest
(unsigned int *       ser,         // serial # (output)
 const std::string&  calibType,
 const facilities::Timestamp& ts,// validity interval must
                                 // include it
 unsigned int        levelMask,  // acceptable proc. level
 const std::string&  instrument, // e.g. LAT, EM, CU, ...
 const std::string&  flavor = "VANILLA"
)
```

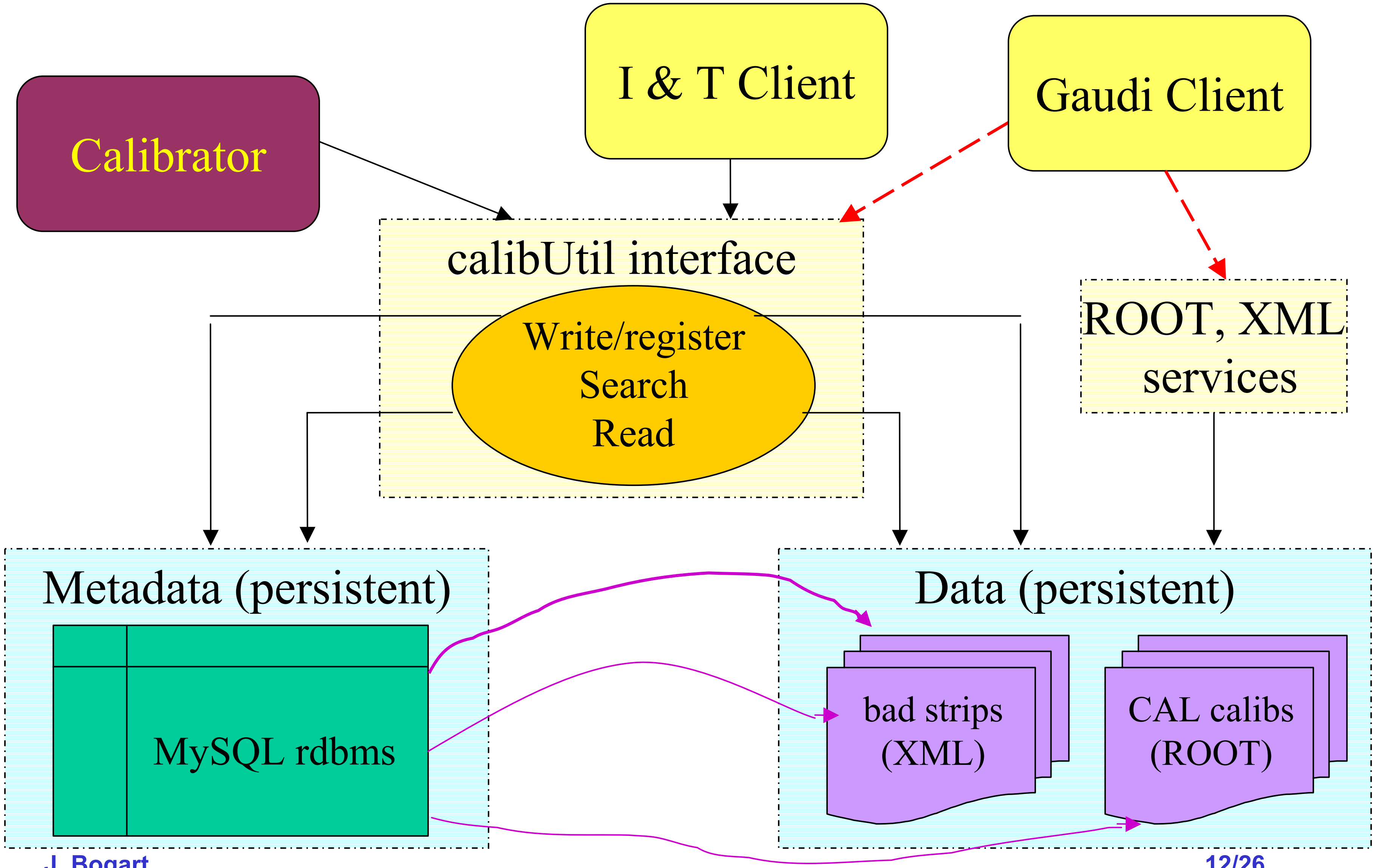
Using the Metadata

...and read it in.

```
/** Given a calibration serial number, return information needed
    for caller to read in the data.
    Returns : true if serialNo exists in dbs and "filename" has
    non-null value;  else false.
  */
Metadata::eRet calibUtil::Metadata::getReadInfo
(     unsigned int    serialNo,       // input
      eDataFmt &      dataFmt,        // XML or ROOT
      std::string &   fmtVersion,
      std::string &   dataIdent  )    // Typically, file spec.
```

findBest and getReadInfo can be viewed as implementing an abstract interface for metadata, independent of the underlying MySQL implementation.

Infrastructure Diagram (simplified)



How to Add a Calibration Dataset

- Generate the bulk data. This is entirely independent of Calibration Infrastructure except for help in some cases in writing it out in the proper form.
- Store the resulting information in an appropriate place.
- Make an entry in the production MySQL dbs pointing to it and including validity interval. calibUtil provides support for this
  - Access to MySQL is automatic for clients of calibUtil, but controlled.
  - Also possible to write a row to MySQL table directly, but without benefit of any sanity check.

How to Track Hardware Status (NYI)

- **When a new calibration of an interesting type has been entered into the system, run a job which**
  - Reads the entire new bulk data set
  - Outputs to a separate hardware dbs, organized by channel rather than by calibration procedure instance
- **Initially do this manually; later it could be triggered automatically.**
- **The hardware database will have its own set of services, probably including histogramming relevant quantities, report generation,… all of which are outside the scope of “Calibration Infrastructure” as used here.**

Gaudi and Calibration

- **As for any TDS (Transient Data Store) object, may associate converter/conversion service with calibration data.**
- **Gaudi has built-in support for calibration/conditions data: data whose validity is a function of time.**
  - IValidity abstract interface for data classes
  - IDetDataSvc interface for data services* needing to check validity
- **However, Gaudi-provided implementation DetDataSvc was unsuitable (initialize() makes some rather specific assumptions) so wrote a variant CalibDataSvc class.**
- **CalibBase class inherits from IValidity, DataObject. Also keeps track of serial number.**
- **Since a (conceptually) single calibration dataset comes in two physical pieces which may be in different formats, the conversion process is most naturally implemented in two stages*: metadata "conversion" and bulk data conversion.**

* **Data service**: something responsible for providing access to data in a TDS.

* See **Acknowledgements**.

Transient (Calibration) Data Store [T(C)DS]

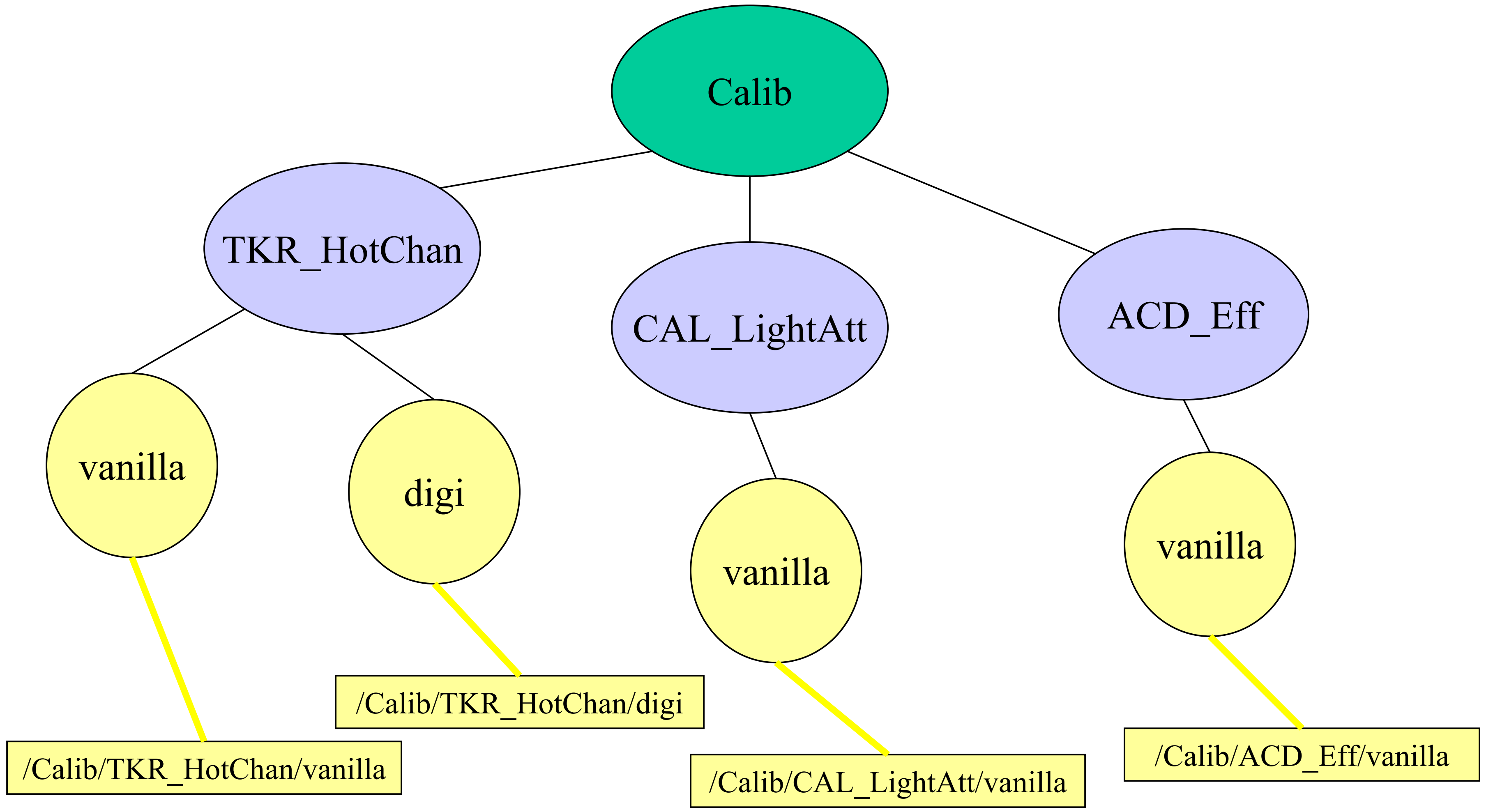
- Datatypes are “simple” in the sense that anyone wanting calibration data gets the whole dataset. There is no hierarchy of data in the TCDS; all the actual data is in the leaf nodes.
- Early on we realized different applications might want different flavors of the same calibration type, covering the same time interval. Might even want more than one flavor available concurrently to different parts of the same job.

Flavors*

- **Potential uses**
  - Handy way to dispense with calibration altogether; use an “ideal” detector, all of whose (flavor = ideal) calibrations are perfect and valid for all time.
  - Can have one set of bad channels at digi step, a different set at recon (which is in fact what happens with real rather than MC data)
  - Can simulate failure modes
- **How dynamic is it?**
  - Code can discover flavors at initialization time; but specifying them (in job options) is a bit clumsy.
  - It's probably adequate for our needs.

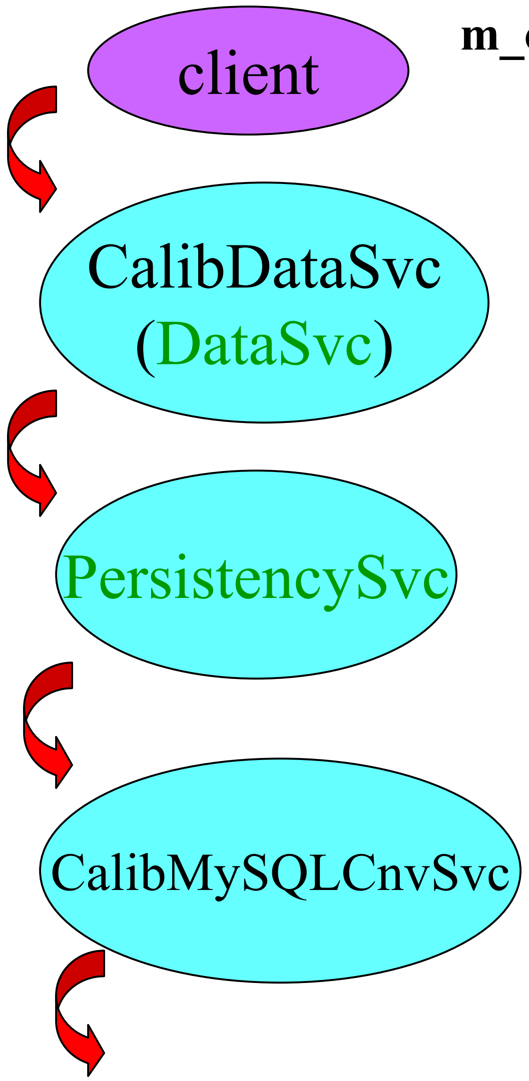
* See **Acknowledgements**.

TCDS Structure



Part of TCDS node hierarchy. Only the leaf nodes have calibration data associated with them.

Create a Calibration Object



m_calibDataSvc->retrieveobject("/Calib/ACD_Eff/vanilla",pObj)

Ask TCDS data service for pointer to object

pLoader->createObject(pAddress ,pObj)

If object not already in TCDS, ask loader (Persistency service) to load it. So-called address is a descriptor including enough information to guide conversion

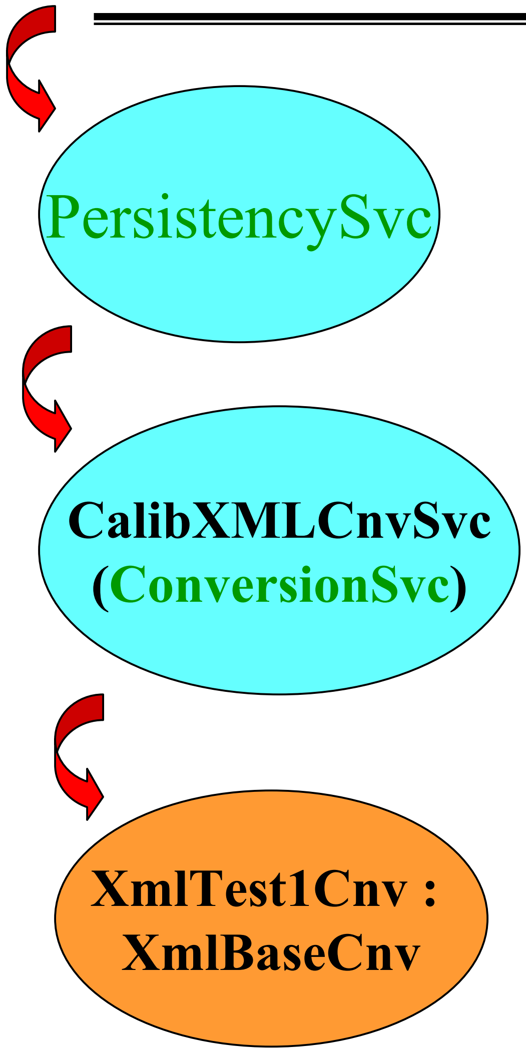
pSvc->createObject(pAddress ,pObj)

Ask format-specific conversion service (MySQL) to create object

```
m_meta->findBest(&ser, calibType, eventTime, ...);
m_meta->getReadInfo(ser, &physFmt, &version, &ident);
m_persSvc->createObj(tmpAddress, pObj);
```

Search meta dbs for best match; get info needed to retrieve calib bulk data, ask Persistency Svc to create corresponding obj.

Create a Calibration Object (cont'd)



pSvc->createObject(pAddress ,pObj)

Ask format-specific conversion service (XML, later also ROOT) to create object

pCnv->createObject(pAddress ,pObj)

Find the right converter for this data type, this physical format, and ask it to create the object

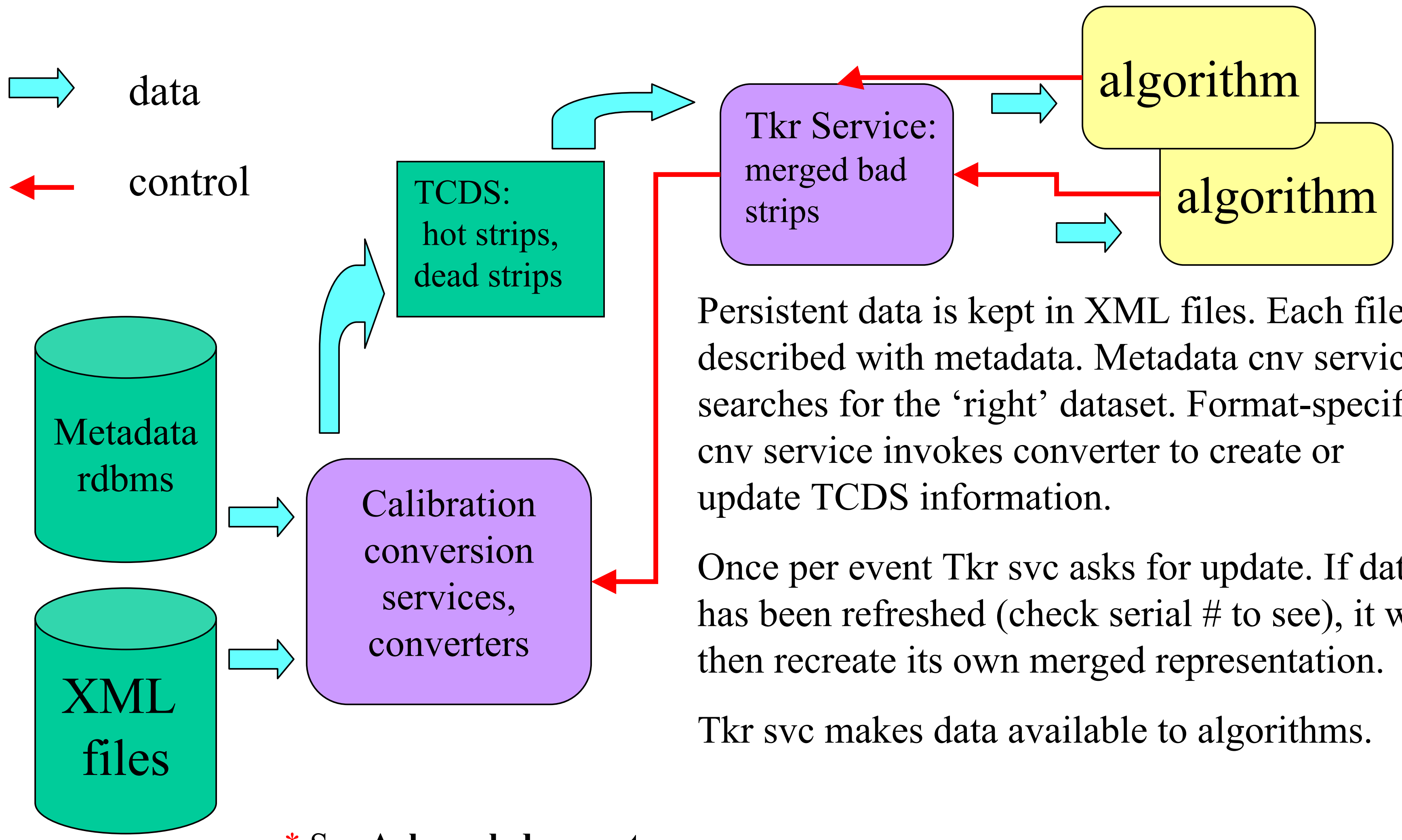
XmlBaseCnv fills info common to all calib data (validity interval, metadata serial number), parses XML file and passes DOM_Document node to specific converter, which fills in remainder of data.

Still Valid?

A typical client algorithm for calibration data has to make at least two calls to the Calibration Data Service to insure that a dataset appropriate for the current event time will be in the Calibration TDS:

- **The first to get a pointer to the data in the TDS (and create it from its persistent form if it's not already there). This is what was portrayed in the previous slides.**
- **The second to verify that data already present in the TCDS is still applicable to current event (and if not update it with a new data set which is). The details are similarly convoluted, but mercifully omitted.**

TKR Bad Strips Architecture*


data
control
algorithm
algorithm
Tkr Service: merged bad strips
TCDS: hot strips, dead strips
Metadata rdbms
XML files
Calibration conversion services, converters

Persistent data is kept in XML files. Each file is described with metadata. Metadata cnv service searches for the 'right' dataset. Format-specific cnv service invokes converter to create or update TCDS information.

Once per event Tkr svc asks for update. If data has been refreshed (check serial # to see), it will then recreate its own merged representation.

Tkr svc makes data available to algorithms.

* See **Acknowledgements**.

CAL per-range data

- Unlike TKR bad strips, there are several CAL calibration types for which every data set (for a given instrument) is of the same size, and all such calibration types have the same organization: there is some fixed amount of data per channel.
- Design nearly-uniform XML (later will be ROOT) description for all such calib types.
- Design helper class CalFinder which knows how to find the right dataset for a particular range.
- End up with parallel class hierarchies, one for per-range data and one for full data set. Template implementation also considered.

Status

- MySQL database exists in a usable form, perhaps even final form. Future changes, if any, will not be sweeping.
- MySQL and XML conversion services exist in final form or something close to it.
- ROOT conversion service is on its way.
- Have several working examples of calibration data types:
  - TKR hot and dead strips
  - CAL pedestals and gains
  - Simple test data type
- For each, have
  - Persistent representation (all XML for now)
  - Corresponding TCDS class
  - Converter taking the former to the latter

Conclusions

So far, the system is living up to expectations. The design effort was long and difficult; implementation and debugging haven't been bad. However, there is plenty left to do:

- **Get ROOT conversion service going (preferably one which will handle event data as well as calibration data)**
- **Add a bunch more calibration data types**
- **Finalize location for persistent form of production calibration data sets.**
- **Implement scheme for getting event time from the event (event has to *contain* a sensible event time field first!)**
- **Design and implement tools for maintaining metadata, in particular for updating validity intervals and checking their properties.**
- **Design and implement alternative to MySQL conversion service for isolated users**
- **Design and implement mirror strategy**
- **Enhance infrastructure or clone it to handle program parameters.**

Acknowledgements

Many thanks to...

- Andrea Valassi, CERN, for help with the two-stage conversion including
  - pointing me to his somewhat similar implementation
  - answering my questions
  - (not least!) encouraging me to follow this route, which seemed rather daunting at first
- Leon Rochester, SLAC (GLAST), for contributing the concept of “flavor” and for helping with design and implementation of bad strips architecture.
- Just about everyone else in the GLAST LAT Software group, one way or another.