Using Multiple Persistent Technologies in the Condition Database of BaBar

Igor A. Gaponenko

LBL

(for BaBar Computing Group)

Presented by

Douglas Smith

SLAC

Contributors

- **Many people have contributed into the project at its various stages**
- **A general design, discussions, and project coordination**
  - Rainer Bartoldus (SLAC)
  - Jacek Becla (SLAC)
  - David Brown (LBL)
  - Antonio Ceseracciu (SLAC)
  - Gregory Dubois-Feldsman (SLAC)
  - Peter Elmer (Princeton Univ.)
  - Igor Gaponenko (LBL)
  - Stephen Gowdy (SLAC)
  - Mateo Melani (SLAC)
  - Andy Salnikov (SLAC)
- **Core CDB design and implementation**
  - Igor Gaponenko (LBL)
- **Migration of payload classes and transient client code**
  - Alex Korol (BINP, Novosibirsk, Russia)
  - Elisa Stevanato (INFN, Padova, Italy)
  - Jane Tinslay (SLAC)
- **RDB (dynamic loading of MySQL client library)**
  - Andy Salnikov (SLAC)
- **Data Distribution**
  - Wilko Kroeger (SLAC)
  - Douglas Smith (SLAC)

What's in this talk

- **Major milestones in a history of CDB**

- **Phasing out Objectivity/DB**
- **Multiple-technology approach: ROOT and MySQL**
- **New CDB implementations:**
  - ROOT I/O
  - MySQL+ROOT

- **Migrating persistent classes from DDL to RDL**
- **Deploying new databases**

- **Status of the project**

Major Milestones of CDB

- **Initially (before Oct 2002) it was "just" ConditionDB**
  - Using **Objectivity/DB** as an underlying persistent technology
- **Major redesign undertaken in 2001/2002 resulted in CDB – "Distributed Condition Database of BaBar":**
  - Rich conceptual model and functionality
  - New API
    - Little and confined dependency on a persistent technology, only for user defined classes
  - Supports multiple persistent technologies and API implementations!!!
  - Performance and scalability improvements
  - First implementation was on a top of Objectivity/DB
    - Inherited user defined "payload" classes and objects
  - More info on the present CDB design:
    - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/Chep2004Paper316.pdf
- **A decision to phase out Objectivity/DB was made in 2004**
  - Keep the same basic design of CDB deployed >3 years ago
  - Choose ROOT I/O and MySQL to replace Objectivity/DB
    - Objectivity/DB has already been replaced with ROOT I/O in BaBar's Event Store
- **At the moment we have all three technologies at various stages of use**

Persistent Technologies
&
CDB Implementations

Why to phase out Objectivity/DB?

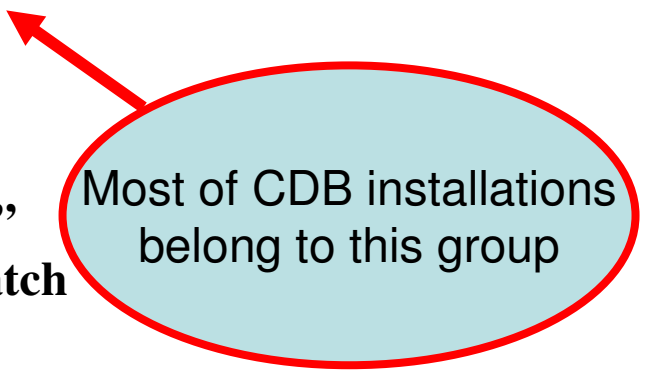
- **Pros**
  - **OODB**, a natural fit for applications written in C++
  - **ACID** compliant (transactions, recovery)
  - Great **performance** (once you understand how to get the one)
  - **Server** based (limited standalone mode also possible)

- **Cons**
  - **Not free**
  - Substantial efforts to **manage** server installations (end users may also be exposed to this)
  - Troubles with some aspects of **client-server interactions**
    - Proprietary client-server protocol
    - No dynamic load balancing, redirections, proxies (all the neat things we have with XROOTD)
  - Hard to deal with the **data distribution**
    - The technology isn't flexible enough to meet all our needs
    - Have to "steal" database files beneath a server
  - Priorities and a **development cycle of the company** have not always met BaBar's needs
    - Slow in some essential (to us) OS-s and compiler ports
    - Delays with some features we badly needed

- **Is a great technology in many instances, but "..one size does NOT fit all.."**
  - Is an **overkill** in some cases (for read-only CDB installations)
  - Is **insufficient** in others (for data distribution)

In search for a replacement

- **In an analysis made in 2004 we identified two major kinds of CDB clients:**
  - Those who require ACID, **read-write** access, backups:
    - ONLINE
    - Prompt Reconstruction and Reprocessing (OFFLINE calibrations)
    - Loading OFFLINE constants
    - Centralized management/planning of distributed CDB
  - And those who need **read-only**, often server-less installations, which would be easy to distribute, would require nearly zero management efforts, and have great performance and scalability:
    - Physics analysis applications
    - Simulation production
    - Monitoring, etc.

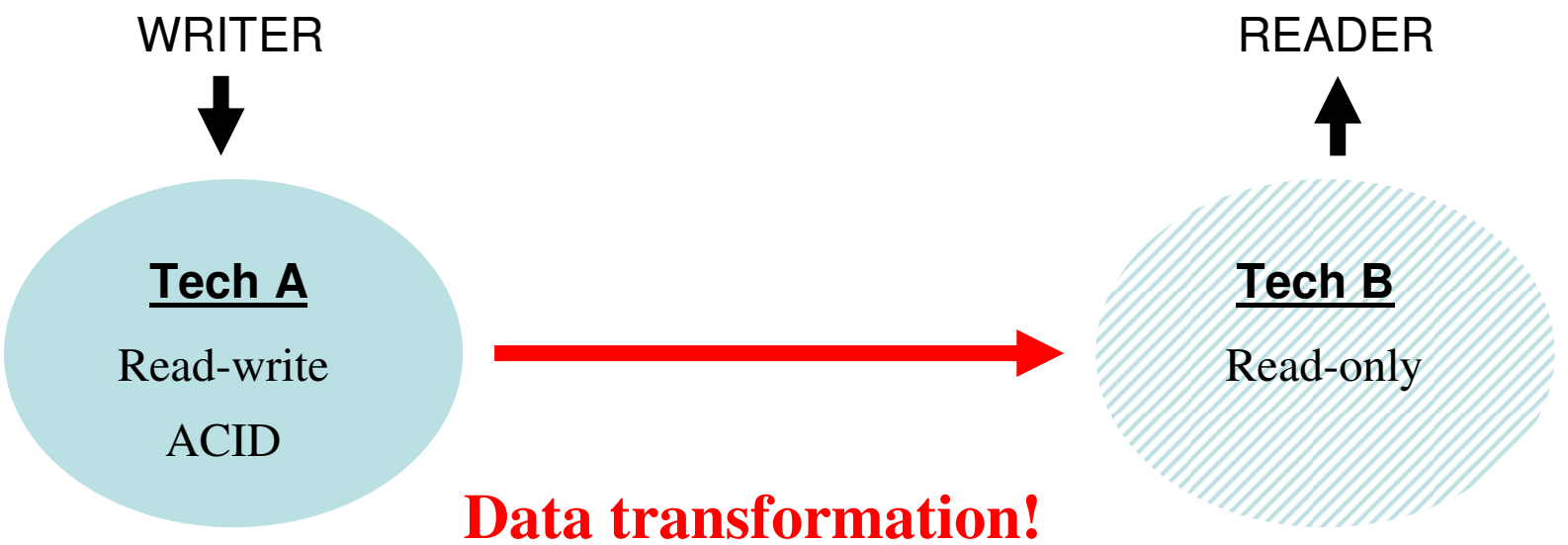


- **A candidate persistent technology had to be "free"**
- **And we didn't want to develop our own from scratch**

- **No single technology met our requirements**

- **In the end, a possibility to use more than one technology or a combination of those, was considered as the best solution**

Two technologies approach



Potential problem: using two different technologies may mean two representations of the same user defined data (and schema). A subsequent **data transformation** may be costly (in terms of time), and it may be difficult to implement. In the later case one would need to maintain two schemas and make sure they're coherent (at least) at a time of each transformation.

See next slide for more details…

More On Data Transformation Problem

Origin of the problem: a model of interactions between user code and CDB API is based on **"object" paradigm** - what's stored in CDB and what's retrieved back is a "payload object". Objects have an identity (OID-s) in the database. User data are modeled using a data definition language of the corresponding persistent technology. Meta-data (intervals of validity, etc.) implementations are hidden behind a technology-neutral CDB API.

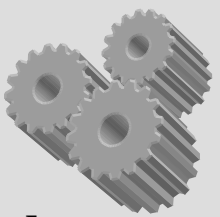
Client code deals with

Technology neutral CDB API for most parts of interactions with CDB

Complex metadata implemented in a technology specific way

hidden

visible



Limited and controlled "breaches" in CDB API to deal with payload objects expressed in a technology specific way (user defined schema and data)

Technology choice: ROOT and MySQL

- **A solution to the data transformation problem (for many-technology approach):**
  - Finding a common language to model user defined payload
  - **ROOT/CINT** was chosen

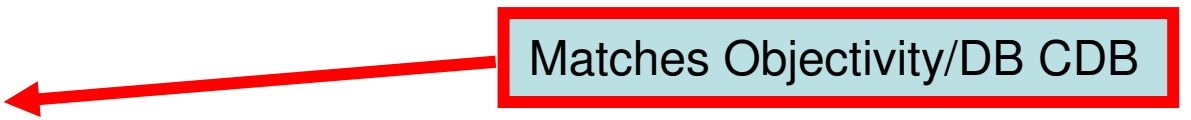
- **We ended up with two separate CDB implementations:**
  - **read-only**: ROOT I/O files for both metadata and payload objects
  - **read-write**:
    - Serialized ROOT objects are hosted in a MySQL database as **BLOB-s**
    - MySQL tables are used for metadata

- **That eliminates the data transformation problem because:**
  - New user objects get loaded into MySQL+ROOT databases only
  - It's easy to **translate** "stable" metadata from MySQL tables into ROOT I/O
  - It's easy to **copy** "serialized" payload objects from MySQL BLOB-s into ROOT I/O
  - **Universal** tools can be written to accommodate these operations

- **Other benefits:**
  - We stay with our **original ("open") OO approach** to model user defined data
  - It makes an ongoing schema and data conversion from Objectivity/DDL into ROOT simpler (due to a more or less **straightforward mapping** between DDL and RDL classes). It's still a tough problem though.

CDB Implementation: ROOT I/O

- **From users' point of view it's just a set of files**
  - And it can be treated as such – copied onto a notebook, etc.
  - Easy to distribute around
    - Fits well into an existing Data Distribution system developed for Event Data
- **Great performance**
  - As long as you know how to get the one with ROOT I/O ☺



- **Can be managed by XROOTD for:**
  - Greater performance of client applications
  - Unlimited scalability
- **GRID ready**
  - No need for special database services (and shared libraries) at GRID sites

CDB Implementation: MySQL+ROOT

- **Why MySQL?**
  - BaBar already uses MySQL for Event Data Bookkeeping
  - Other RDBMS can also be used
- **Fully ACID compliant solution**
- **Good performance**
  - Though, in the current implementation of our code it's still **worse** compared to Objectivity/DB or ROOT I/O implementations of CDB.
    - MySQL schema and transient code tune up is still needed
  - **Sufficient** for our use cases
- **Data Distribution**
  - Is much **less an issue** compared to Objectivity/DB CDB
    - A number of CDB installation is very small, all in controllable "production" areas
    - Ready to use tools exist for MySQL replication
  - **MySQL replication** is planned
    - **Third party** solutions are also available

2xCPU PIII 1.2 GHz 1 GB server
~4 MB/s for large objects
~2 kHz of read requests
~400 Hz of write requests

Specialized CDB Server? Why not?

- **Choosing ROOT/CINT to model user-defined data opens a path for this option!**

- **Specialized CDB server:**
  - Serves payload objects defined in RDL (***ROOT Definition Language in BaBar***)
  - Hides all details of its internal implementation
  - May use various persistent technologies to store data
  - Possibly caches the whole database in memory (if enough memory available)
  - Provides transactions (if needed)
  - Allows dynamic load balancing, redirections, proxies, etc.

- **A *federation* of CDB servers would effectively solve a problem of synchronizing distributed database installations**
  - The server would also support a management protocol
  - Servers would be able to talk to each other and exchange data

- **Client-server protocol options**
  - CORBA?
  - TCP/IP (for maximum performance)

That's just an idea!
No implementation exists yet.

Migration & Deployment

DDL to RDL Migration

- **Payload migration is the most complex problem in replacing Objectivity/DB**
  - A variety of DDL classes; different designs; "composite" objects.
  - A total number of persistent classes to be migrated: ~200.
  - A total number of persistent objects to be converted ~3M.
  - Significant amount of transient client code having direct or indirect dependency on Objectivity/DB API.

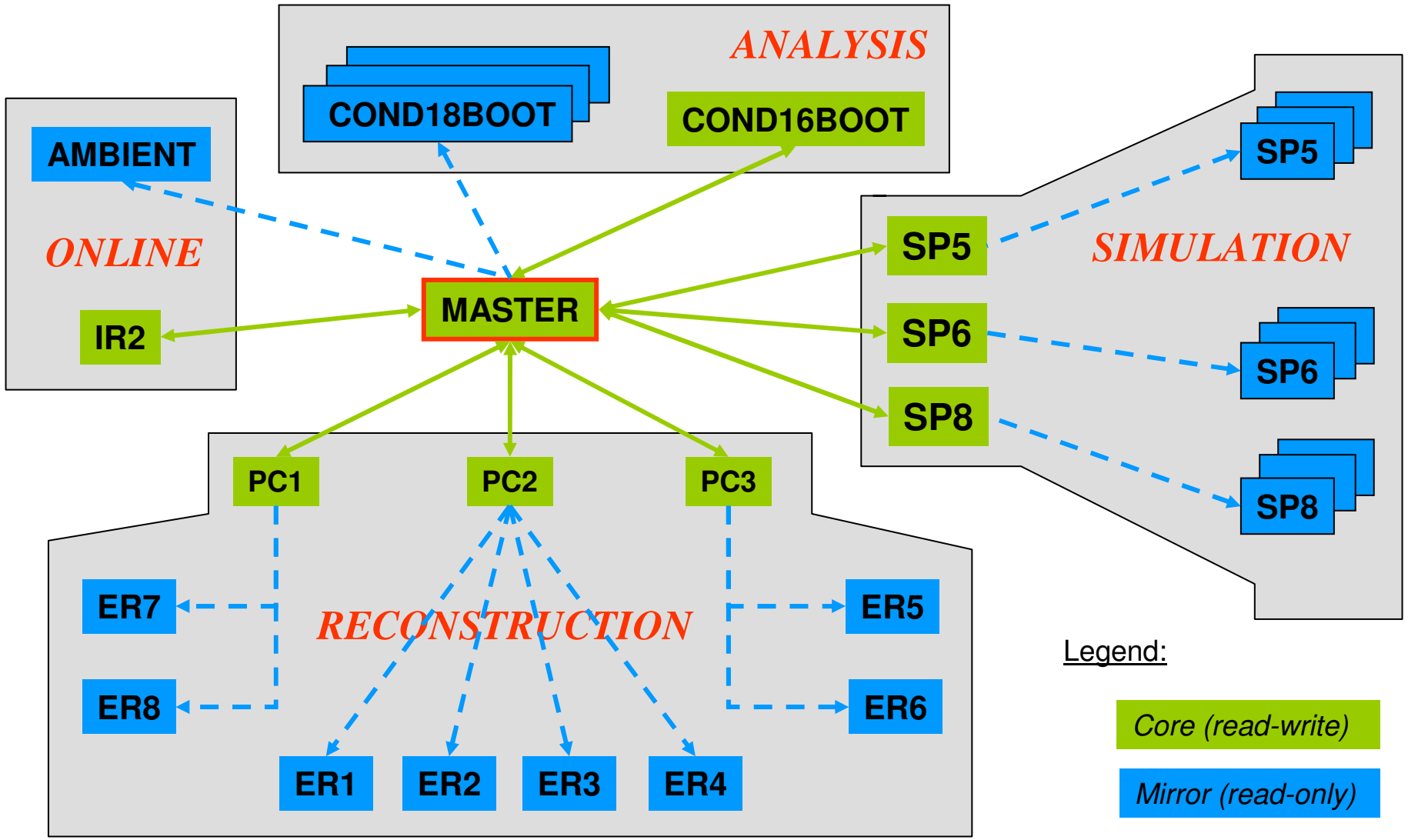
- **Is being done in three incremental steps:**
  - A subset of classes for analysis applications (25%)
  - A subset of classes for simulation production applications (over 50%)
  - Full set of classes

- **A problem of verifying the quality of the data conversion**
  - A low-level test suit had to be developed for object-to-object comparison
  - An application level comparison (*Does my analysis produce the same results?*)

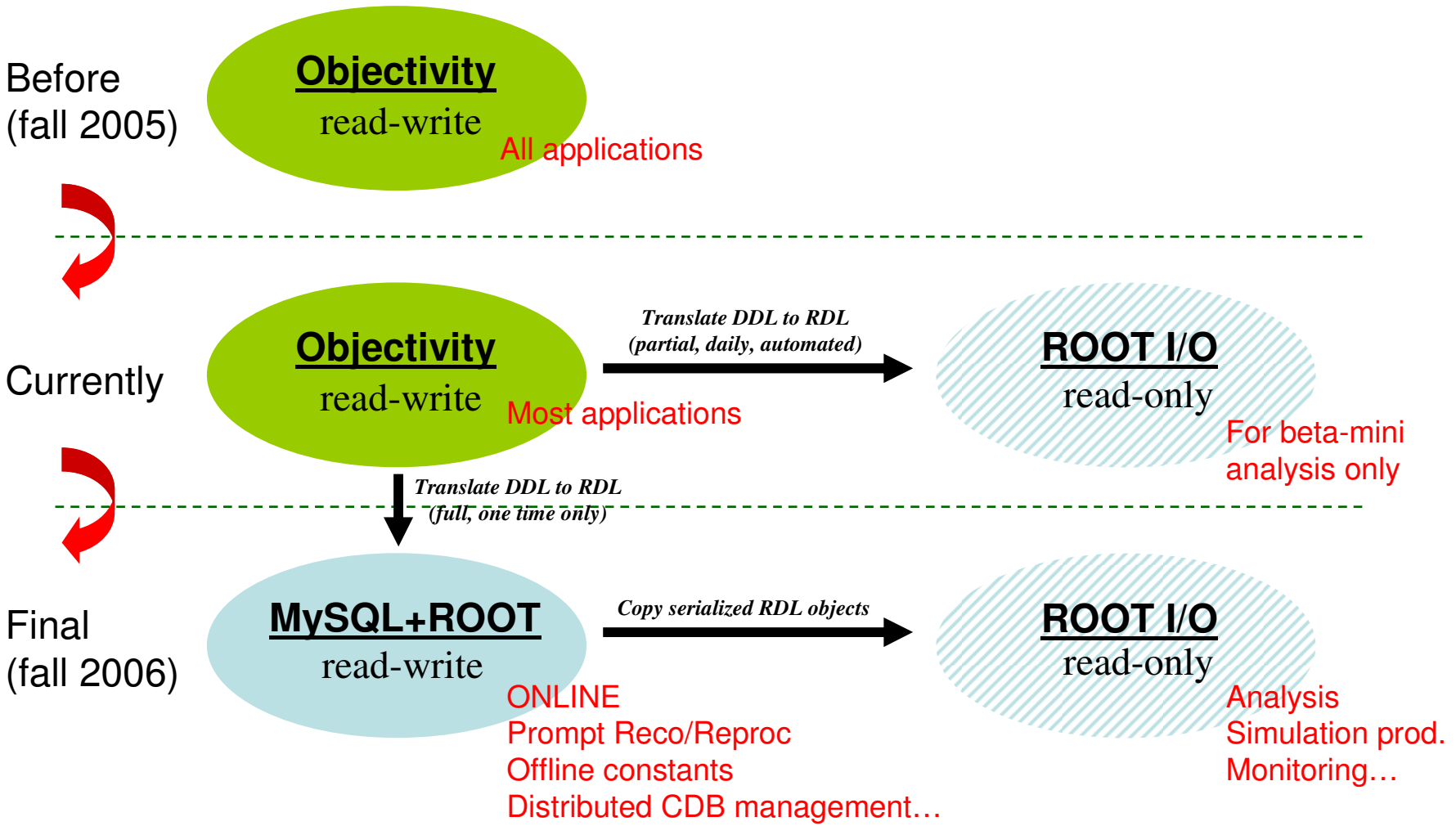
- **A "social" problem**
  - Many original developers (of payload classes and code) have left the experiment.
  - Have to migrate most of the classes ourselves.

- **LESSON: better to enforce stricter policies on a design of payload classes!**

Distributed Setup to Be Migrated



Deploying in two major steps



Current Status

- **Core CDB implementations for all three persistent technologies mentioned before exist**
  - MySQL+ROOT still needs some tune up (SQL schema)
- **A subset of payload classed has been migrated from Objectivity/DDL into ROOT (RDL)**
  - Sufficient to run BaBar's "**beta-mini**" analysis
- **An automated conversion procedure established for "beta-mini"**
- **Work is going on to finish migrating the rest of payload classes to RDL**
- **In a process of deploying the ROOT I/O based CDB for distributing around the Collaboration**
  - This will effectively solve a problem of accessing CDB from small sites and universities where Objectivity/DB wasn't available or difficult to use.
- **The final switch from Objectivity/DB is planned to be accomplished by Fall 2006**