LCG Persistency Framework Status Report

Dirk Duellmann, CERN IT
(on behalf of the POOL and ConditionsDB teams)

http://pool.cern.ch and http://lcgapp.cern.ch/project/CondDB/

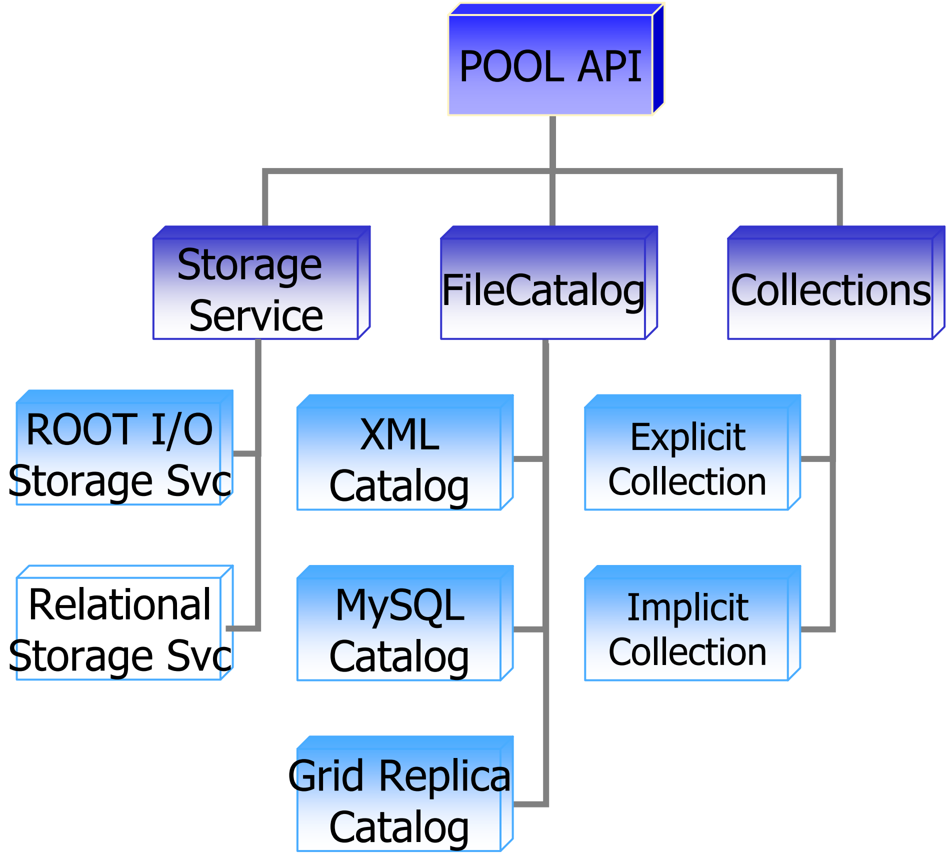
LHCC review,
November 23, 2004

The LCG Persistency Framework

- The LCG persistency framework project consists of two parts
  - Common project with CERN IT and strong experiment involvement
- POOL
  - Hybrid object persistency integration object streaming (using ROOT I/O for event data) with Relational Database technology (for meta data and collections)
  - Established baseline for three LHC experiments
  - Has been successfully integrated into the software frameworks of ATLAS, CMS and LHCb
  - Successfully deployed in three large scale data challenges
- Conditions Database
  - Conditions DB was moved into the scope of the LCG project
    - To consolidate different independent developments
    - and integrate with other LCG components (SEAL, POOL)
  - Should share storage of complex objects into Root I/O and RDBMS backend with POOL

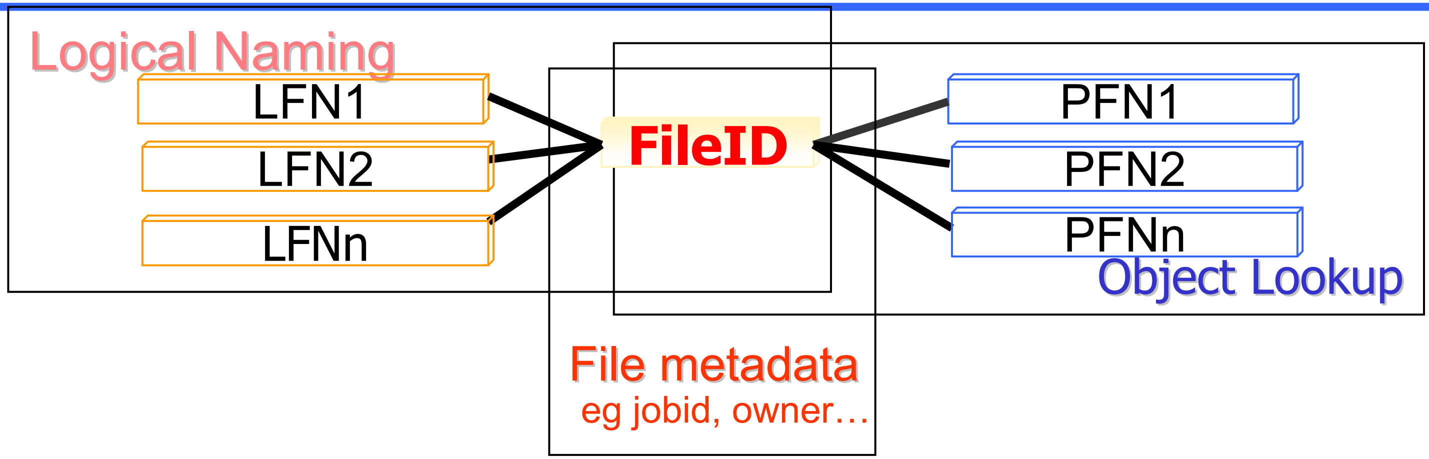
POOL Component Breakdown

- Storage Manager
  - Streams transient C++ objects to/from disk
  - Resolves a logical object reference to a physical object
- File Catalog
  - Maintains consistent lists of accessible files together with their unique identifiers (FileID), which appear in the object representation in the persistent space
  - Resolves a logical file reference (FileID) to a physical file



- Collections
  - Provides the tools to manage potentially large ensembles of objects stored via POOL
    - **Explicit**: server-side selection of object from queryable collections
    - **Implicit**: defined by physical containment of the objects

POOL File Catalog Model



- POOL adds system generated **FileID** to standard Grid m-n mapping
  - Allows for stable inter-file reference even if lfn and pfn are mutable
  - Several grid file catalogs implementation have since then picked up this model (EDG-RLS, gLite, LFC)

- POOL model includes optional file-level **meta-data** for production catalog administration
  - several grid implementations provide this service (eg EDG-RLS, gLite)
  - used mainly for administration of query large file catalogs
    - not for generic physics meta data storage
  - e.g. extract partial catalogs (fragments) based on production parameters

- Fragments can be shipped (+ referenced files) to other sites / decoupled production nodes
  - POOL command line tools allow cross-catalog +cross-implementations operations
  - End-users can connect to several catalogs at once
    - Different implementations can be mixed; Only one can be updated.

POOL Deployment in the Grid

- Coupling to Grid services
  - In 2004 -Middleware based on the EDG-RLS; Service uses Oracle Application Server + DB
    - Connects POOL to all LCG files
      - Local Replica Catalog (LRC) for GUID <-> PFN mapping for all local files
      - Replica Metadata Catalog (RMC) for file level meta-data and GUID <-> LFN
      - Replica Location Index (RLI) to find files at remote sites (not deployed in LCG)
        - ☹ Resulted in a single centralized catalog at CERN (scalability and availability concerns)
  - Several newer grid catalogs in the queue
    - LFC, gLite, Globus RLS teams plan to provide implementations of the POOL interface

☝ **But Grid-decoupled modes also required by production use-cases**

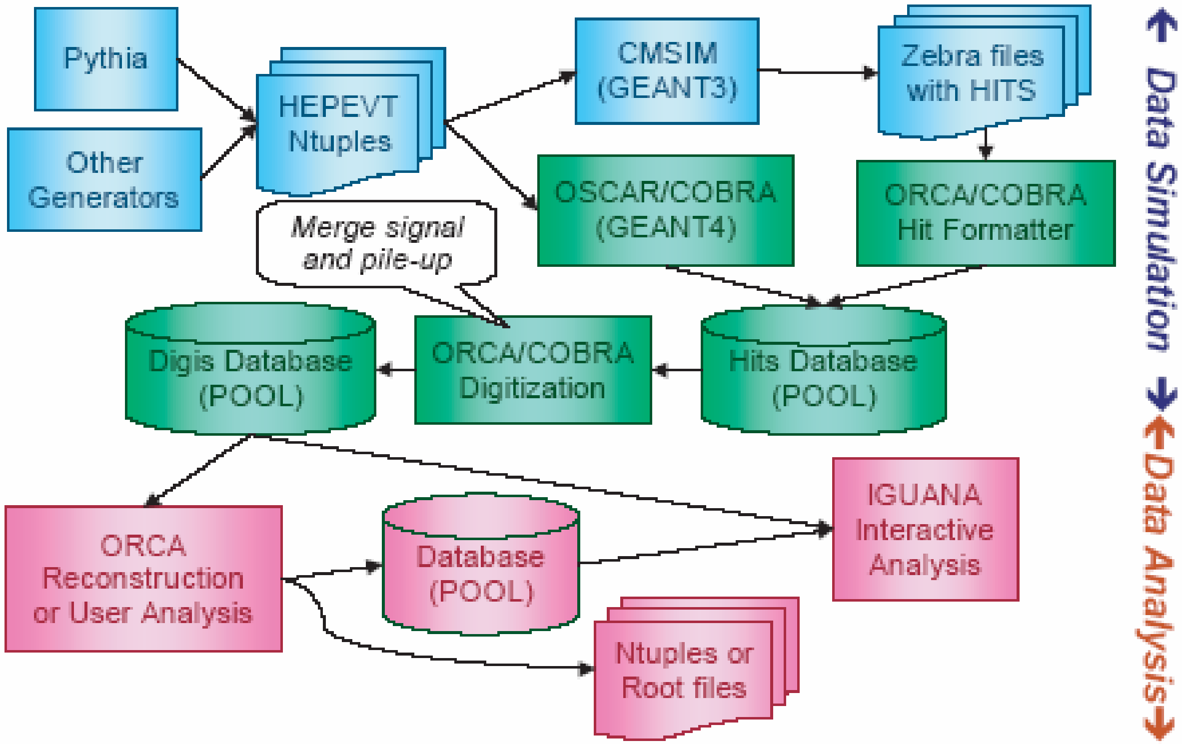
- ➢ XML Catalog
  - Typically used as local file by a single user/process at a time
    - no need for network
    - supports R/O operations via http; tested up to 50K entries
- ➢ Native MySQL Catalog
  - Shared catalog e.g. in a production LAN
    - handles multiple users and jobs (multi-threaded); tested up to 1M entries

CMS DC04

- Demonstrate the capability of the CMS computing system to cope with a sustained rate of 25Hz for one month

- Started in March 2004 based on the PCP04 pre-production (simulation)
  - Reconstruction phase including POOL output concluded in April 2004

- Distributed end-user analysis based on this data is continuing



	digitization	reconstruction
Total amount of data (TB)	24.5	4
Throughput (GB/day)	530	320
Tot num of jobs (1k)	35	25
Jobs/day	750	2200

CMS DC04 Problems

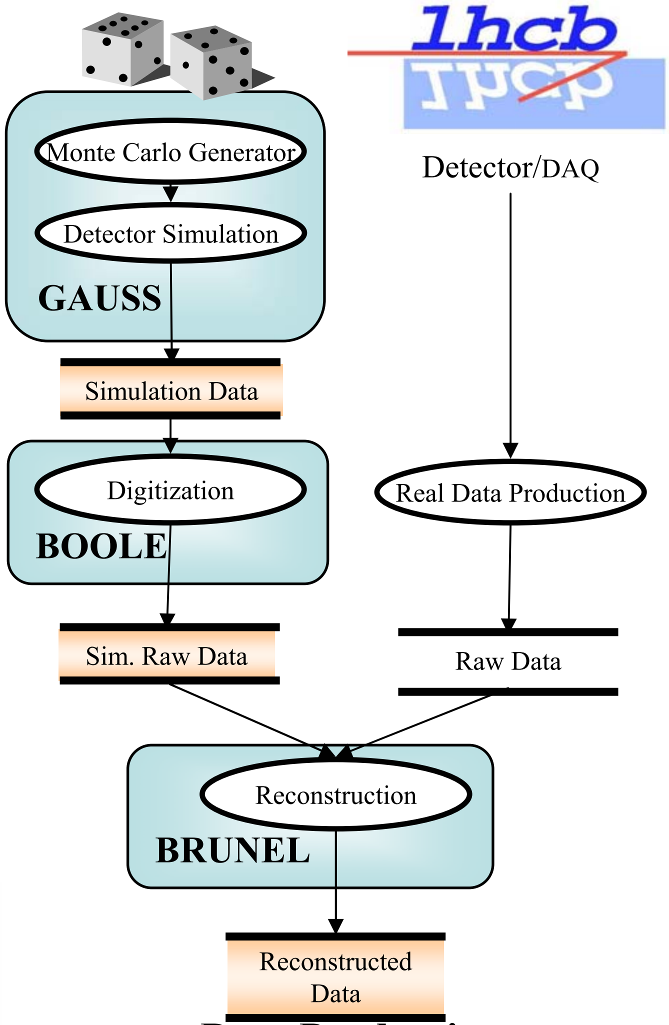
- RLS backend showed significant performance problems in file-level meta-data handling
  - Queries and meta data model became concrete only during the data challenge
    - GUID<->PFN queries 2 orders magnitude faster on POOL MySQL than RLS
    - LRC-RMC cross queries 3 orders magnitude faster on POOL MySQL than RLS
- Main causes:
  - overhead of SOAP-RPC protocol
  - missing support for bulk operations in EDG-RLS catalog implementation
- Transaction support missing
  - Failures during a sequence of inserts/updates require recovery "by hand"
- Basic lookup / insert performance satisfactory
- The POOL model for handling a cascade of file catalogs is still valid
  - Good performance of POOL XML and MySQL backends proves this
  - RLS backend problems being addressed now by IT-Grid Deployment Group
- ☺ Good stability of the RLS service achieved!

ATLAS Data Challenge 2 scale

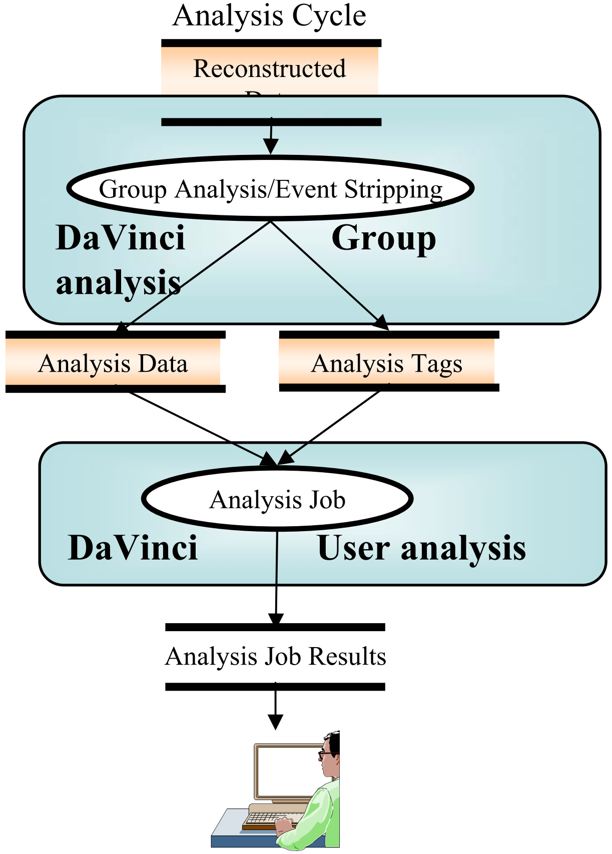
- Phase I: Started beginning of July and still running
- 10^7 events
- Total amount of data produced in POOL: **~30TB**
- Total number of files: **~140K**
- Digitization output is in bytestream format, not POOL
  - This is the format of data as it comes off the ATLAS detector
- Anticipated ESD (October 2004): 700 KB/event→7 terabytes in POOL
  - ESD is currently ~1.5 MB/event, but this will decrease soon
  - 2 copies distributed among Tier 1s implies 14 TB ESD in POOL
- Anticipated AOD (October 2004): 22 KB/event→220 gigabytes in POOL, to be replicated N places (N>6)
- TAG databases: MySQL-hosted POOL collections replicated at many sites
  - "All events" collection ~6 gigabytes; physics collections will be smaller (10-20% of this size)

Data Processing in LHCb

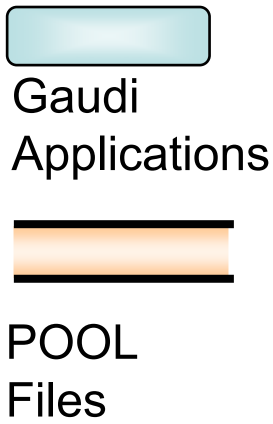
File type	# files	# events	Data Volume [TB] produced	Data Volume [TB] kept in mass storage
Simulation data	791 k	319 M	116	7
Digitized data	604 k	226 M	128	6
Reconstructed data	348 k	225 M	66	64



Data Production



Data analysis



POOL Deployment 2004

- Experience gained in Data Challenges is **positive!**
  - No major POOL-related problems
  - Close collaboration between POOL developers and experiments invaluable!

  - EDG-RLS deployment based on Oracle services at CERN
    - Stable throughout the 2004 Data Challenges!

  - File Catalog experience in 2004
    - Important input for the future Grid-aware File Catalogs

- **Successful integration and use in LHC Data Challenges!**

- **Data volume stored: ~400TB!**
  - Similar to that stored in / migrated from Objectivity/DB!

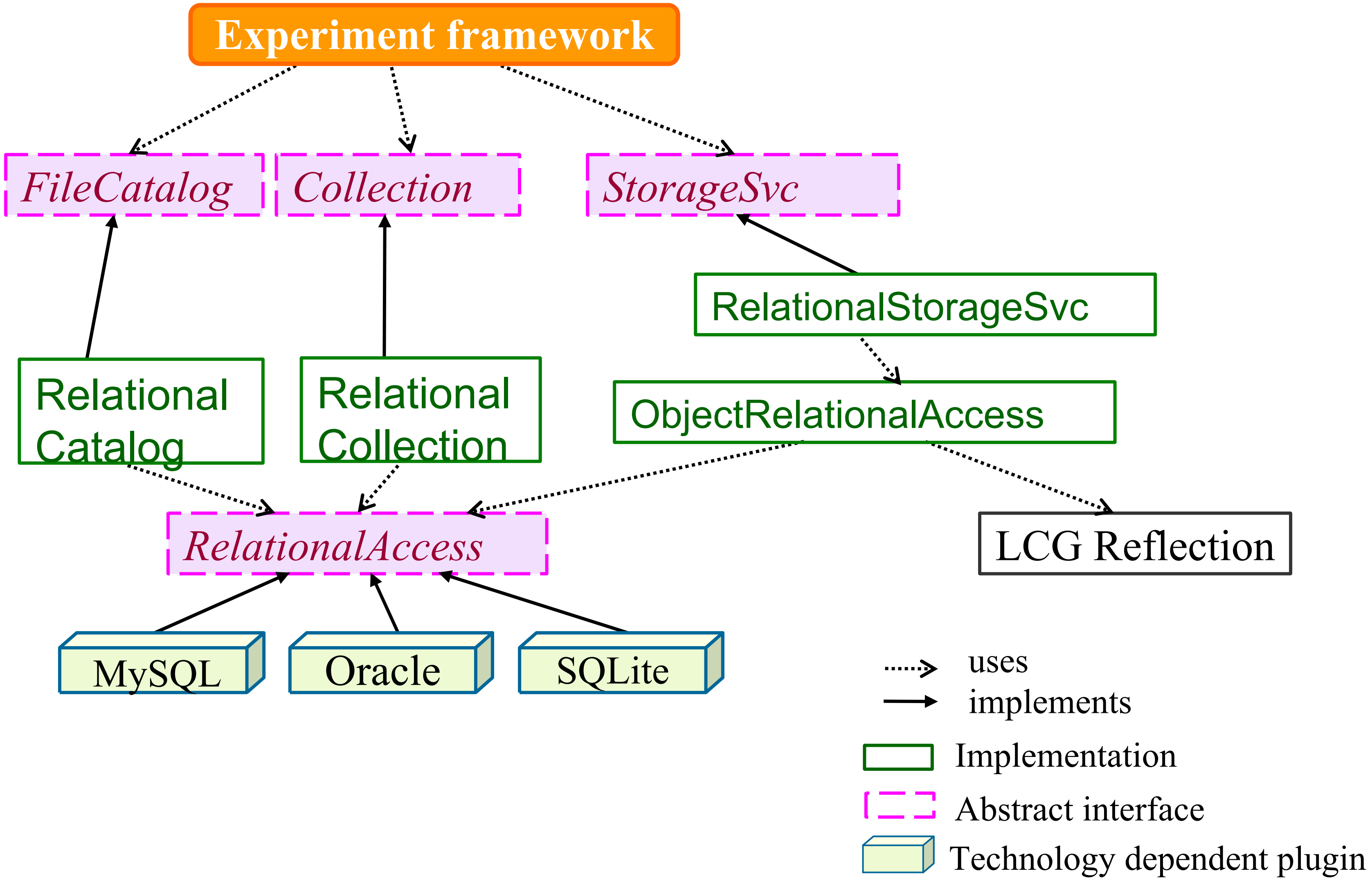
Developments this Year

- Move to ROOT4 (POOL2.0 Line)
  - To take advantage of automatic schema evolution and simplified streaming of STL containers
    - Need to insure backward compatibility for POOL 1.x files
  - Currently undergoing validation by the experiments
    - Will release two branches until POOL 2 is fully certified
- File Catalog deployment issues
  - DC productions showed some weaknesses of grid catalog implementations
    - Several new/enhanced catalogs coming up
    - Changes in the experiment computing models need to be taken into account
  - POOL tries to generalise from specific implementations and provides an open interface to accommodate upcoming components
- Collections
  - Several implementations of POOL collections exist
  - Collection cataloguing has been added in response to experiment requests
    - Similar to file catalogs
    - re-use of catalog implementation and commandline tools
  - Experiment analysis models are still being concretized
  - Expect experience from concrete analysis challenges

Why a Relational Abstraction Layer (RAL)?

- Goal: Vendor independence for the relational components of POOL, ConditionsDB and user code
  - Continuation of the component architecture as defined in the LCG Blueprint
  - File catalog, collections and object storage run against all available RDBMS plug-ins
- To reduced code maintenance effort
  - All RDBMS client components can use all supported back-ends
  - Bug fixes can be applied once centrally
- To minimise risk of vendor binding
  - Allows to add new RDBMS flavours later or use them in parallel and are picked up by all RDBMS clients
  - RDBMS market is still in flux..
- To address the problem of distributing data in RDBMS of different flavours
  - Common mapping of application code to tables simplifies distribution of RDBMS data in a generic application independent way

How does this fit into POOL?



Object to Relational Mapping

- How to map classes ↔ tables ?
  - Both C++ and SQL allow to describe data layout
  - But with very different constraints/aims
    - no single unique mapping
- Need for fast object navigation an unique Object identity (persistent address)
  - requires unique index for addressable objects
  - part of mapping definition
- POOL stores mapping with the object data
  - including mapping versions
- First prototype exists which stores simple objects with vectors and maps as described in the LCG Dictionary

Conditions DB

- **Project launched in summer 2003 (within LCG Persistency Framework)**
  - Background in 2000-2003:
    - C++ API definition and Objectivity implementation
    - Oracle implementation of the original API ("BLOB" data payload)
    - API extensions and MySQL implementation (user-defined relational data payload)
  - Two goals for the common project:
    - Integrate the existing Oracle and MySQL packages into LCG Application Area
    - Coordinate new development of API, software and tools

- **Status overview**
  - Kick-off workshop at CERN in December 2003
  - Activity along two directions in parallel
    - Integrate the existing software into LCG Application Area
    - Review two APIs and implementations, coordinate discussion about new developments
  - *Main problem so far: lack of committed manpower*
    - *New developments also slowed down by the divergence in the two APIs*

CondDB software releases

- **Release CONDDB_0_1_0 (April 2004) – first public release**
  - Most recent Oracle and MySQL implementations (integrated in LCG CVS and SCRAM)
  - CondDBOracle: original common API (only BLOBs) – only for gcc2.95.2
  - CondDBMySQL: Lisbon extended API (BLOBs and ICondDBTable)
  - Separate API and examples for the two packages
- **Release CONDDB_0_1_1 (May 2004)**
  - Full support for gcc3.2.3 (Oracle OCCI for gcc3.2.3), functionality as in CONDDB_0_1_0
- **Release CONDDB_0_2_0 (July 2004)**
  - Common dependency on API package ConditionsDB (~original API, only BLOBs)
  - Lisbon extensions (ICondDBTable and others) in CondDBMySQL
  - Same functionality and packaging as CONDDB_0_1_1, only packaging changed
- **Next release CONDDB_0_3_0 (October 2004?)**
  - Common dependency on library package CondDBCommon (SimpleTime implementation)
  - DataCopy and Utilities packages with libraries/tools to extract/copy MySQL data
  - Maybe: possibility to link together both packages and copy data across implementations?
- **Future releases CONDDB_0_4_x**
  - Integration (common dependency?) with SEAL: CondDBOracle/MySQL as SEAL plugins
  - Integration with POOL: POOL string token example, copy POOL data too from DataCopy

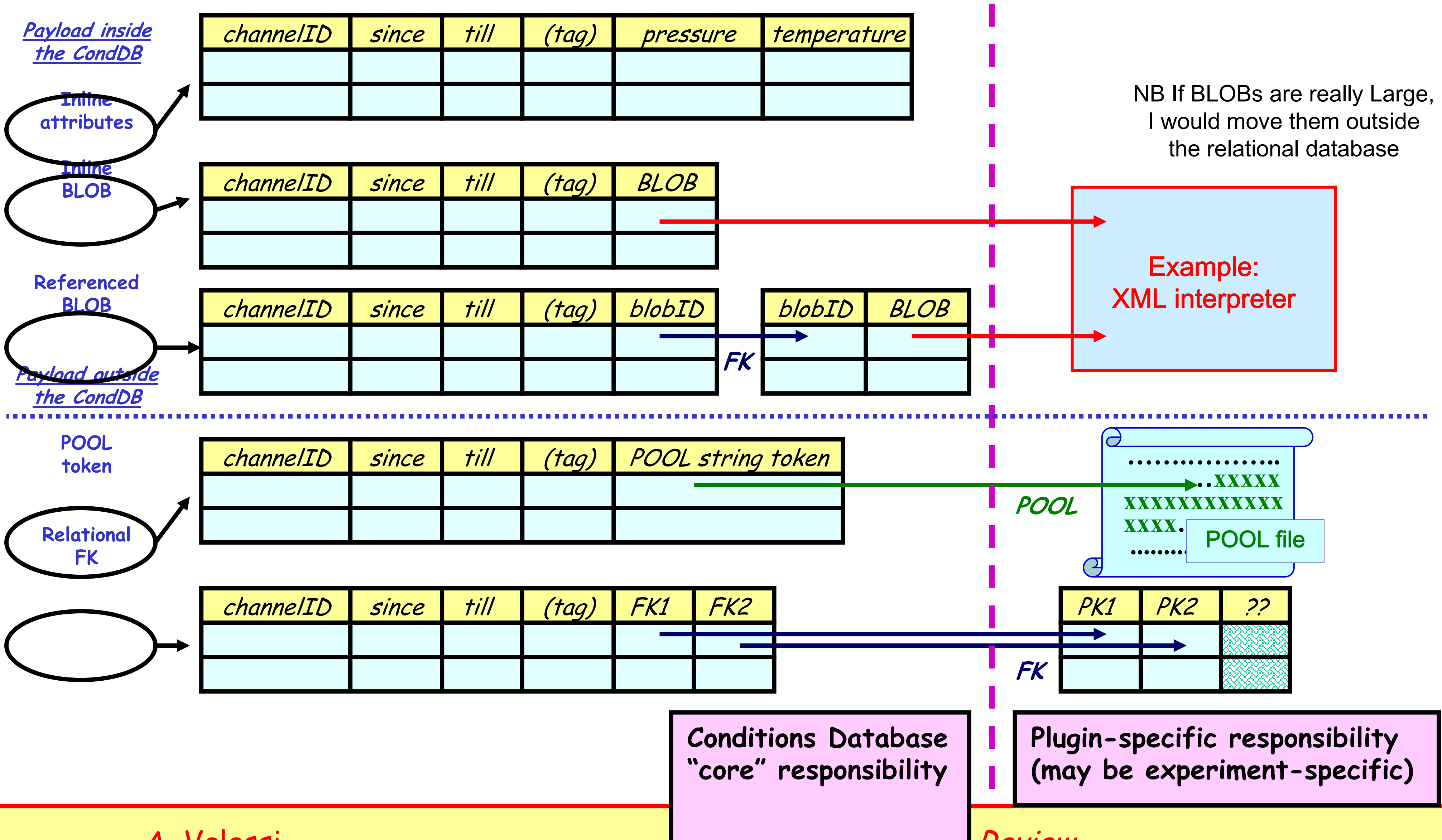
Goals and non-goals

- **Project non-goals (experiment-common, not conditionsDB-specific)**
  - Generic C++ access to relational databases (→ POOL project: RAL)
  - Generic relational database deployment and data distribution (→ 3D project)
    - Integration with data distribution infrastructure, however, is a project goal
- **Project goals (experiment-common, conditionsDB-specific)**
  - Common software and tools for non-event time-varying versioned data
    - NB 1: you will need to work a lot to customize the common solution to your needs!
    - NB 2: even this may still be too generic! (see the next slide)
- **Project non-goals (experiment-specific)**
  - Specific data models for calibration/geometry/… (→ experiments)
  - Specific payload format encoding (→ experiments)
    - That is to say: how you use relational databases, RAL or POOL is up to you!
  - Specific time encoding and other conventions (→ experiments)

Main limitations of current software

- **CondDBOracle (original API)**
  - *Data model:* only BLOBs, *no user-defined data payload* (~ à la CondDBMySQL)
    - BLOBs also imply performance overhead if you only need to store POOL *string* tokens
  - *Implementation:* *slow*, need reengineering (bulk inserts, speed up versioning)
- **CondDBMySQL (original API, plus Lisbon API extensions)**
  - *Data model & C++ API:* too many ad-hoc solutions, lacks a consistent approach
    - *for instance: BLOBs and relational attributes handled by two different APIs*
    - for instance: versioning/tagging and "channel ID" not provided for all "folder types"
  - *C++ API:* ICondDBTable interface is confusing, many concepts mixed up
    - *schema vs. contents; one vs. many objects*; persistent table vs. transient objects
  - *Duplication of effort:* large overlap with POOL relational access
- **CondDBOracle and (vs.) CondDBMySQL:**
  - *Differences in data model & C++ API: new common developments very difficult*
  - *Implementation:* schemas differ even in tables providing same functionality
    - data copy between CondDBOracle and CondDBMySQL more complex than it could be
  - *Duplication of effort:* code/schema implemented separately in Oracle/MySQL
    - any new features (e.g. partitioning, user tags) would need to be implemented twice
  - *Data distribution:* lack consistent data model and API for partitioning/cloning
  - *Integration:* foresee components to handle referenced data in POOL or tables

Data payload: typical use cases



Consolidation Proposal (main points)

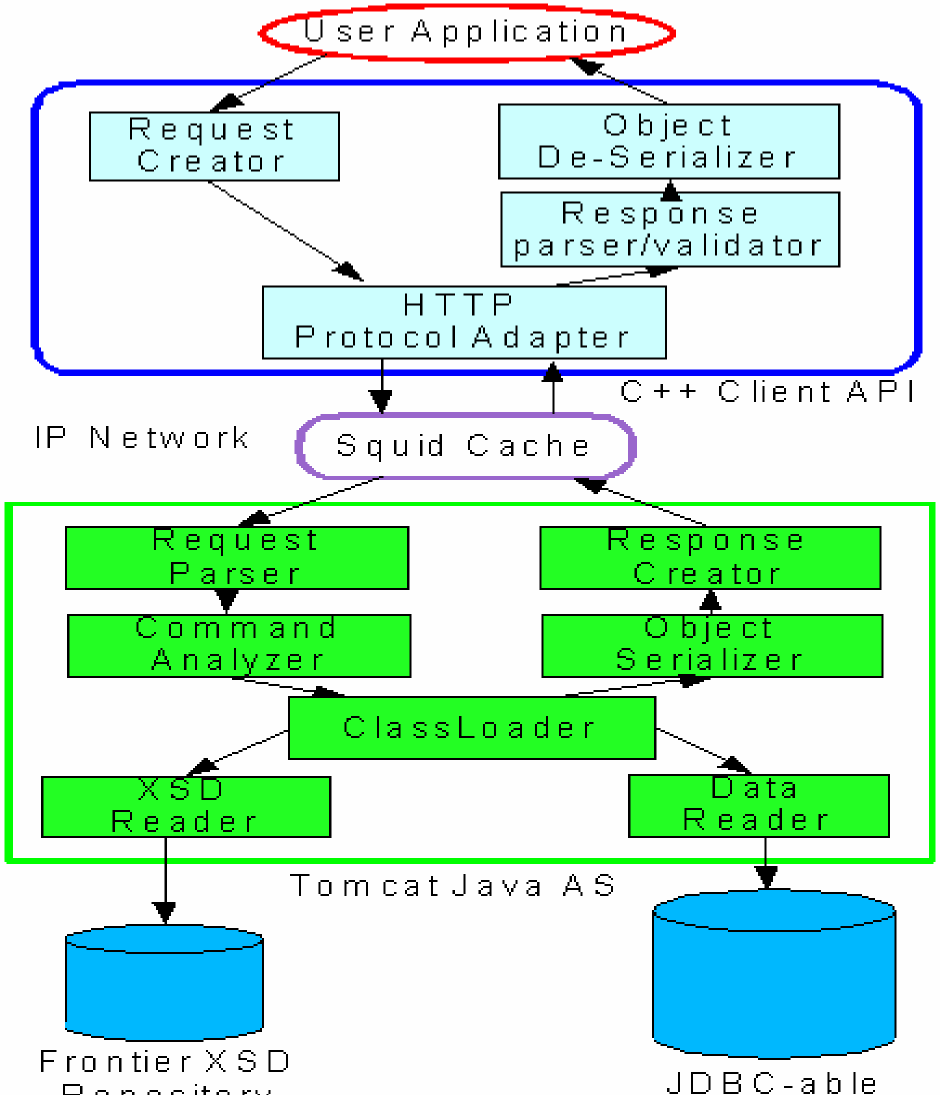
- **Extend original API: user-defined data payload (AttributeList)**
  - Different payload schemas in different folders (AttributeListSpecification)
- **Drop the "ICondDBTable" Lisbon API, extend the ICondDBObject API**
  - Keep the same API and metadata model for BLOBs and user-defined data
  - Clean separation of schema vs. data and of one vs. many objects
- **Extend original API: foresee partition management and data cloning**
  - Many physical partitions may be created within the same logical folder
  - Add special methods to insert "cloned" data (user-specified insertion time)
- **Component decomposition: develop components above new extended API**
  - Handlers of specific payload types (POOL tokens, relational FKs, BLOBs…)
  - Slicing and copy tools (including deep copy of referenced data, eg POOL)
  - Synchronization manager: keep registered data items in sync with event time
  - Browsing and visualization tools (accepting plugins for user payload)
- **Maximise integration with existing LCG solutions**
  - Infrastructure: support only SCRAM builds on the official LCG platforms
  - Software: take AttributeList and "generic" relational tables from RAL

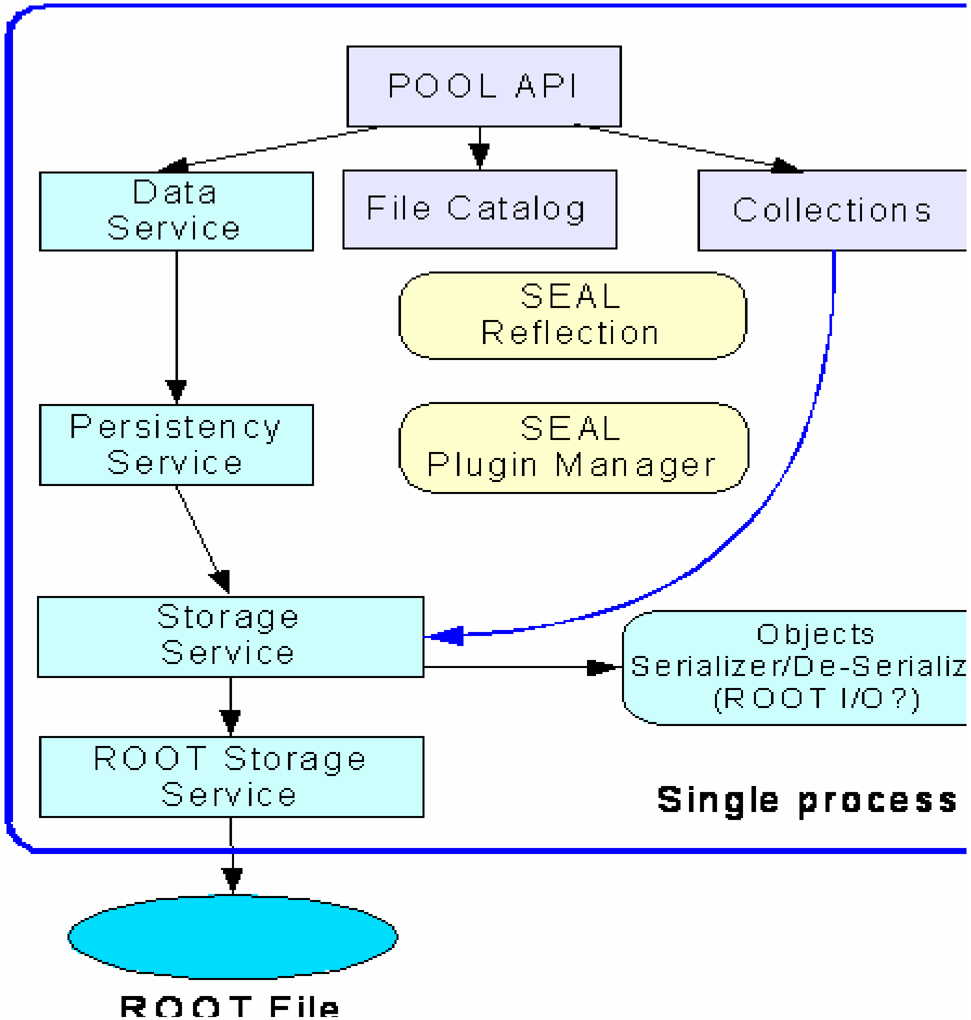
POOL RAL and LCG 3D

- Propose POOL RAL as reference implementation interfacing to a distributed database service (LCG 3D)
  - Location independent connection to a database replica
    - Prototyping work using the POOL (File) catalog components underway
    - Within limited development resources on POOL and 3D sides
  - Mapping of grid identity to local database user and role
    - Consistent VOMS backend implementation for DB (and POOL files) required
- Will keep RAL independent from other POOL components
  - Possibly package separately for applications which have no other POOL dependency…
  - …but would profit from a simplified LCG 3D integration

Slide prepared by the FroNTier team

Frontier & POOL (Simplified)





FroNtier Integration

- Access to remote Databases is provided by the FroNtier system developed at FNAL
  - http based data transfer with light-weight, database independent client
  - Full encapsulation of database details (schema, queries, physical storage) behind application server
  - Simplified deployment option for read-only data in particular for higher level tiers with only limited resources to provide a database service
- Proof of concept POOL integration has been done by FroNtier team at FNAL
  - Frontier objects appear as normal POOL objects as described by the LCG Dictionary
  - More work required to setup a test deployment infrastructure (within LCG 3D) but very promising option eg for conditions data
  - Integration with ConditionsDB project being discussed
- Integration of the pre-existing FroNtier system with was possible in relatively short time

Developments vs. Work Plan

- Since the work plan presentation to PEB
  - Relational Abstraction Layer has been delivered and is used by several experiments
  - Development on the object-relational is progressing and first working prototype exists
  - ROOT4 migration is progressing and POOL 2 pre-releases have been build to allow for experiment validation
  - Several file catalog upgrades and workarounds have been introduced to limit the RLS performance problems
- But without doubt, POOL fell behind schedule
  - Use of RAL in all POOL components and relational storage manager
  - Automated dictionary loading
  - Schema evolution test suite
  - Additional OS ports
  - Collection and Ref integration into interactive ROOT
- Main reason: decreasing resources available to POOL development since beginning of this year
  - Move of developers into service provision and experiment integration
- Both IT and the experiments are now taking measures to fix this

POOL Project Evolution

- POOL is entering its third year of active development
  - Joint development between CERN and experiments
  - During the last 2 years we managed to follow the proposed work plan and met the rather aggressive schedule to move POOL into the experiment production
  - This year POOL has been proven in the LCG data challenges with volumes ~400TB
- Changing from pure development mode to support, deployment and maintenance
  - Several developers moved their effort into experiment integration or back-end services
    - This is healthy move and insures proper coupling between software and deployment!
    - Affects the available development manpower
  - Task profile changing from design and debugging to user support and re-engineering
- Need to maintain stable and focused manpower from CERN and the experiments
  - This close contact has made POOL a successful project
  - Both Experiments and CERN have confirmed their commitment to the project

Summary

- The LCG POOL project provides a hybrid store integrating object streaming with RDBMS technology
  - POOL has been successfully integrated into three quite different LHC experiments software frameworks
  - Successfully deployed as baseline persistency mechanism for CMS, ATLAS and LHCb at the **scale of ~400TB**
- POOL continues the LCG component approach by abstracting database access in a vendor neutral way
  - A Relational Abstraction Layer has been released and is being picked up by several experiments
  - Minimised risk of vendor binding, simplified maintenance **and data distribution** are the main motivations
- POOL as a project is (slowly) migrating to a support and maintenance phase
  - Need keep remaining manpower focused in order to complete remaining developments and to provide adequate support to user community
  - Maintaining a significant experiment contribution is required insure the the tight feedback loop which made POOL an effective project

- The LCG Conditions DB project has produced several releases of the Oracle and MySQL based implementations within the LCG Application Area
  - After an interface and extension review a concrete plan to consolidate the implementations has been discussed
  - Manpower also from the experiments is now becoming available to the project allowing to re-factor the package based on the Relational Abstraction Layer
- New complementary technologies such as FroNtier are being integrated into the LCG persistency framework as distributed access to database data gets more interest

POOL catalogs in ATLAS DC2

- Production jobs read from and write to local .xml POOL file catalogs
  - Content of .xml catalogs is imported into grid replica catalogs when output files are published
  - Conversely, minimal .xml catalogs are created and shipped with input files when jobs are submitted
- Using EDG-RLS for master POOL file catalog on LCG
  - Other grids (e.g., Grid3) use Globus RLS as master catalog
- An ATLAS data management tool (Don Quijote) knows how to communicate with multiple catalog flavors
  - Don Quijote adjusts for the fact that the Globus RLS does not support file GUIDs natively
  - *Ad hoc* solution stores GUID as metadata in Globus RLS
- Metadata are maintained in a separate ATLAS bookkeeping service (AMI) that supports queries on datasets and returns LFN lists
  - No use of POOL for file- or dataset-level metadata
  - No pattern-matching queries on LFNs in POOL file catalogs

POOL Components in Use in LHCb

- StorageSvc + ROOT backend
- FileCatalog + XML backend
  - Catalog used without metadata functionality
  - Create XML catalog slices from LHCb bookkeeping
- PersistencySvc
- LHCb does not use RLS
- LHCb does not use POOL event tag collections
  - Though Gaudi event tag collections are POOL files
- POOL is mostly hidden from LHCb users
  - Dynamically loaded Gaudi module
  - What users see is a configuration file and input/output file specification(s)

A Mapping Example

```
class A {
  int x;
  float y;
  std::vector<double> v;
  class B {
    int i;
    std::string s;
  } b;
};
```

A Mapping Example

p.k. — *f.k. constraint*

T_A

ID	**X**	**Y**	**B_I**	**B_S**
1	10	1.4	3	“Hello”
2	22	2.2	3	“Hi”
.	.	.	.	.

T_A_V

ID	**POS**	**V**
1	1	0.12
1	2	12.2
1	3	4.1
1	4	5.452
2	1	32.1
2	2	0.1
2	3	0.1

This is only one of the possible mappings!

ConditionsDB Use

- <u>**Only one production user so far: Atlas test beams**</u>
  - *Essentially using only the work done by the Lisbon group*
    - Only *MySQL version* used in production
    - Only *extended API* used in production (no BLOBs)
    - Atlas-specific software installation (not from central LCG installation)
    - Software integration with Athena and PVSS
  - Writers: online and offline
    - <u>Online (PVSS interface):</u> all data from DCS, no filtering, stored when values change
    - Offline: output from Muon alignment program
    - Data size ~10 GB in 2000 folders/tables
  - Readers: online and offline
    - Online: experts debugging their detector (CondDB used as/instead of PVSS archive)
    - Offline: input to Muon alignment program
    - Offline: Athena code reading output from Muon alignment program
- **Other activities**
  - Tests in LHCb: plan offline readers only, BLOBs or POOL only (Oracle/MySQL)
  - Tests in CMS; also ideas on registering in CondDB data from preexisting tables
  - No production use of CondDBOracle (except for pre-LCG version in Harp)

Proposal: extend CondDBObject API

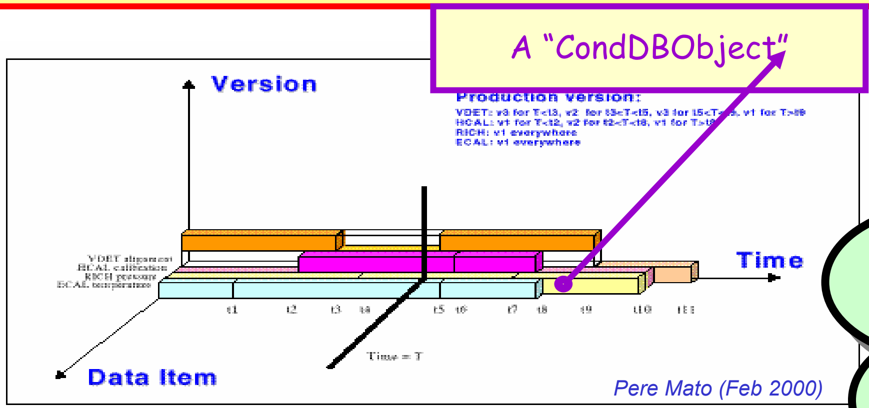
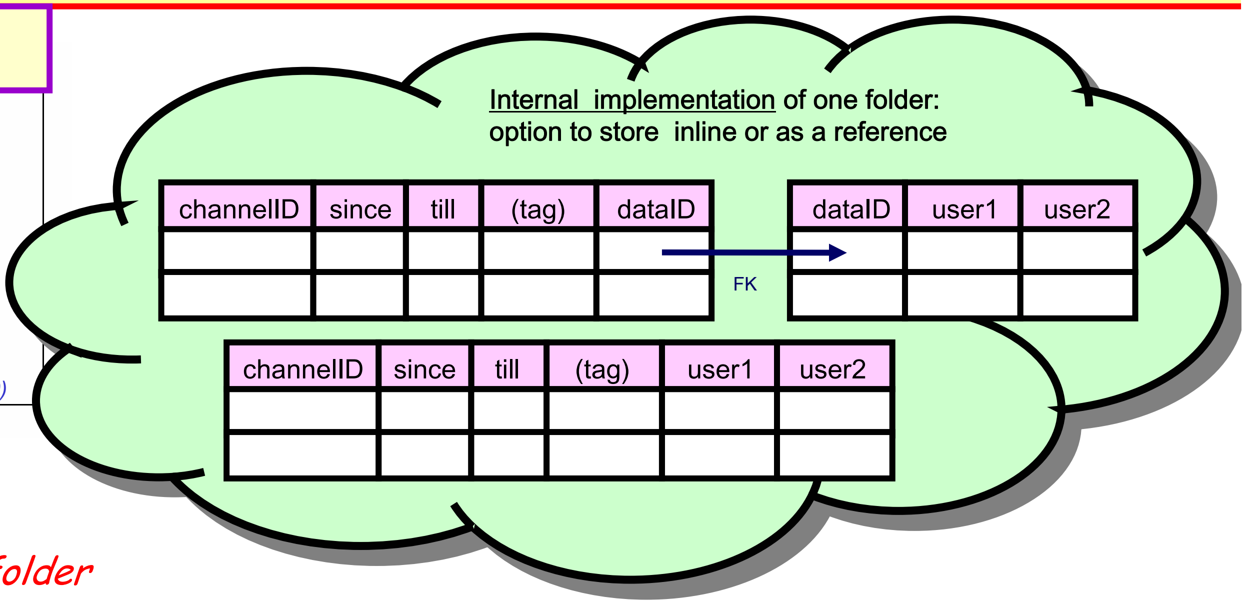


Figure 1 The three axes for identifying uniquely each data item in the condition database



Metadata for one CondDBObject:

1. Data item id: folder name + *channelID in folder*
   - Options at folder creation: specify channelID schema (AttributeListSpecification); no channelID (only one channel)
2. Interval of validity: [since, till]
3. Version info: *insertion time* (not layer number)
   - Options at folder creation: *no versioning*; versioning with *inline user data*; versioning with referenced user data (stored only once)

Payload for one CondDBObject:

1. User data (AttributeList)
   - Simple C++ types, BLOB; no arrays
   - At folder creation: specify user data AttributeListSpecification
   - Different folders have different schemas; different channels in the same folder have the same schema)

POOL Ref/Collections in ROOT

- Work plan has been discussed between POOL/ROOT and SEAL after last years review
  - Development effort for a first prototype was estimated to be rather modest
  - Development started only late and was made difficult by communication problems discussing POOL storage manager
- Lack of resources in this area has been pointed out to the experiments
  - A single person in POOL is handling several complex requests (ROOT4, schema loading, etc.)
  - Not many people can effectively contribute in this area
- Need a firm commitment in terms of manpower for this essential area from all experiments to insure that POOL
  - receives detailed experiment feedback and requests
  - has sufficient manpower to implement them
  - POOL evolution is properly taken into account on the experiment side