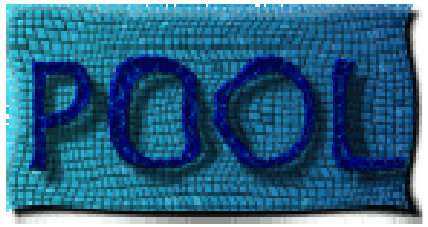


Databases developments in the LCG Persistency Framework

Giacomo Govi

CERN IT/PSS

On behalf of POOL project

Project scope

- Mandate: data persistency of LHC physics application
- Two main technology domains
  - **FILES - based on ROOT I/O**
    - Targeted for complex data structure: event data, analysis data
    - Main focus of the project developments during the first two years
    - Migration to Reflex LCG dictionary completed
    - No big changes planned. Support & maintenance
  - **Relational Databases – Oracle, MySQL, SQLite**
    - Suitable for conditions, calibration, alignment, detector description data - possibly produced by online systems
    - Complex use cases and requirements, multiple ‘environments’ – difficult to be satisfied by a single, solution
    - Completed first phase of developments
    - Focus moved to deployment and experiment support

Project deliverables

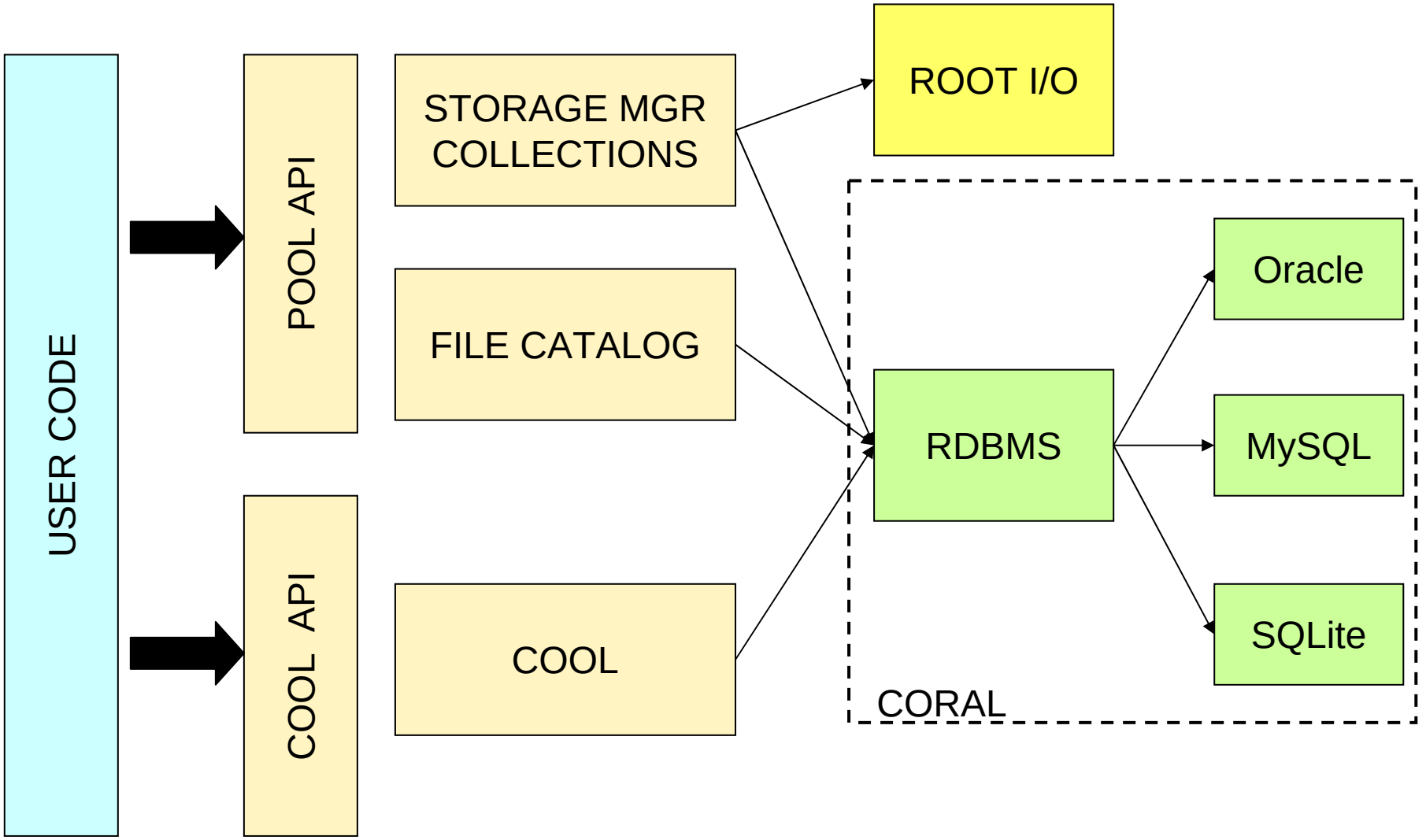
- POOL
  - framework for the persistency of arbitrary C++ objects, with File-based (Root) or RDBMS back-ends
  - Users: ATLAS, CMS, LHCb
- CORAL
  - general, technology independent interface to Relational Database
  - Users: COOL, POOL, ATLAS
- COOL
  - framework for the handling of condition data associated to a time validity
  - Users: ATLAS, LHCb

Project organization

- Work packages
  - Object storage and references
  - Collections and Metadata
  - Database access and distribution
  - Catalog and Grid integration
  - Conditions Database

- Current resources involved: 7.1 FTEs
  - From experiments: 3.4 FTEs

Project Framework



CORAL

COmmon Relational Abstraction Layer

A C++, SQL-free, technology-independent API for RDBMS accessing

- Schema definition and manipulation
- Data manipulation
- Queries
- ORACLE, MySQL and SQLite plug-ins available

- Insulates in a common layer all the code accessing the DB
  - Allows efficiently code (and knowledge…) re-use

- Usage extended beyond the scope of POOL
  - COOL: self contained package independently released
  - ATLAS: direct access of relational data
- API was reviewed and validated by early adopters
- New features addressing deployment and distribution of relational data

Highlights of the CORAL API

- Bulk operations
  - Round-trips to the server are minimized in insert/update/delete operations
- Internal use of bind variables
  - SQL parsing on the server is avoided
- Client-side caching of query results (row pre-fetching)
  - Round-trips to the server are minimized when fetching the result set of a query
- Support for BLOB I/O.
- Optimizations and “best practices” implemented in the RDBMS plug-ins
  - Users may concentrate on the functionality of their own use cases

Addressing deployment (I)

Database connection

- Uniform connection protocol for different RDBMS flavours
  - An internal service selects transparently the plug-in to use
- Logical database service name: a lookup service provides the corresponding contact string
  - Service indirection to access distributed replicas
  - Provides connection failover across RDBMS technologies
- Client-side connection pooling
  - Client can access data through a connection proxy
- Authentication
  - Explicit/Implicit, via a dedicate service
  - Exploring authentication based on Grid certificates

Addressing deployment (II)

Client-side monitoring

- General interface for monitoring service to register information about interesting events:
  - Time and duration of sessions
  - Transactions
  - Response time
  - …
- Collected data can be pushed into existing monitoring system by implementing the interface
  - E.g. MonALISA
- A default implementation is provided as a simple data place holder

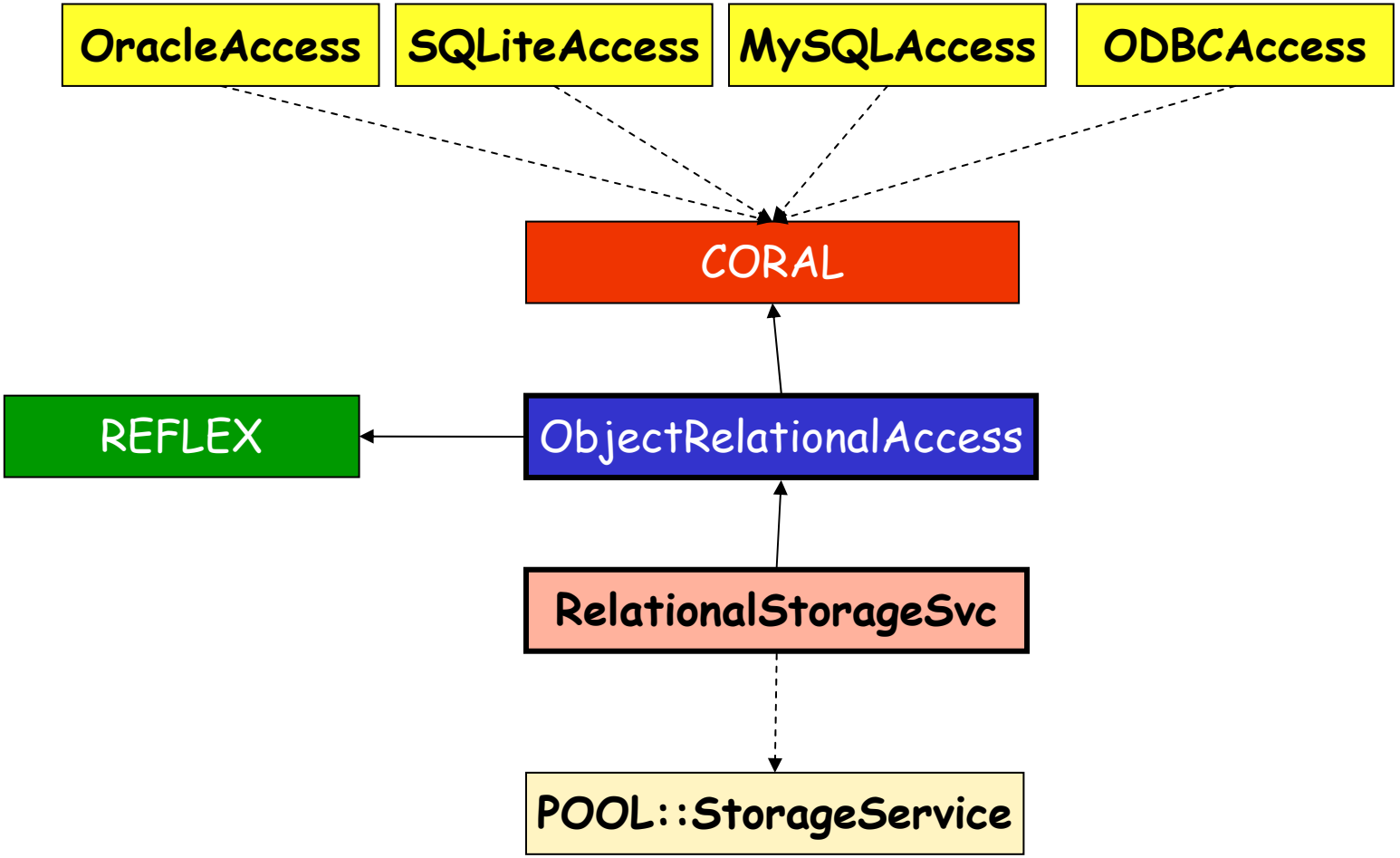
POOL Object Relational Access (ORA)

Implementation of the existing POOL Storage Manager interface for C++ objects persistency

- Users access data with the same interface used for the ROOT backend
- Write and Read complex data structures into/from relational DBs
  - Wide acceptance of C++ constructs (seal Reflex)
  - Selecting the DB technology according to the requirements
- Retrieve existing relational data as C++ objects in the offline reconstruction/analysis framework
  - Import in the off-line chain condition data taken on-line

- First integration tests from CMS successful!

Object Relational Access (ORA)



ORA: Object mapping

- How to map C++ classes ↔ RDBMS tables?
  - C++ and SQL describe data layout with very different constraints/aims
- Mapping definition :
  - Objects need an unique identifier (persistent address)
  - Reflex dictionary necessary to decompose the class hierarchy in simple elements (primitives/strings)
  - Tables and column associated to the fields with simple ‘default’ rules
  - Everything is stored in 3 relational tables
  - A tool is provided for the user-defined mapping:
  - Mapping gets “materialized” and stored in the database
- Object arrays supported
  - Currently stored with the elements in individual records
  - BLOB-based storage coming soon

ORA: Reading pre-existing data

ORA can read relational data as C++ objects, even if tables and rows are generated by other means

- Prerequisites:
  - Classes describing the object layout are defined with a proper mapping
  - Rows storing objects data are uniquely identified by primary keys
- Command-line tool available
  - Set up the POOL database according to the directives specified by the user (technology, containers, mapping)
  - ‘Soft’ import: configure and update POOL internals to populate pool containers
  - Original data is unchanged!

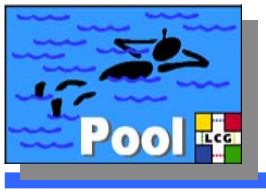
Conditions Database Project (COOL)

Common software for the management of experiment conditions data

- Conditions data: non-event data that vary with time and may also exist in different versions
  - Online producers: slow control (single version)
  - Offline producers: calibration, alignment (multiple versions)
- Minimize duplication of efforts, maximize reuse of LCG software
  - Single implementation (and same relational schema) for multiple back-ends using CORAL – same user code for all backends
  - Free access to CORAL connection pooling, monitoring and retrial
- Modeling of conditions “objects” (e.g. a calibration set)
  - System-managed common *metadata*
  - User-defined *data payload*

COOL Status

- Basic functionalities implemented in several production releases
- Current priorities concern deployment issues for Atlas and LHCb
  - Performance validation and optimization
  - Deployment has started for Atlas (production accounts on RAC)
- Performance validation and optimization
  - Definition of realistic experiment workloads
  - Proper sizing of database server
  - Software optimization (identify and solve performance bottlenecks)
- Data extraction and replication tools and tests
  - Basic tools for data extraction and replication exist
  - Replication and access tests in the context of the 3D project
  - Start the design of a more distributed data model and API for a COOL “database”



Summary

The LCG Persistency framework provides transparent access to RDBMS based persistency

- Experiments needs and requirements fulfilled:
  - CORAL API for general data manipulation
  - COOL Framework specific for time-varying condition data
  - Object Relational Storage manager for storage and retrieve of data as C++ objects
- Strong focus on features addressing deployment issues
  - Access to multi-technology services
  - Authentication and connection handling
  - Client-side monitoring
  - Use and promote best practices in the DB client code
- Attention to experiments concrete use cases
  - Ad hoc tools implemented to meet specific requirements
  - Optimization and performance tuning