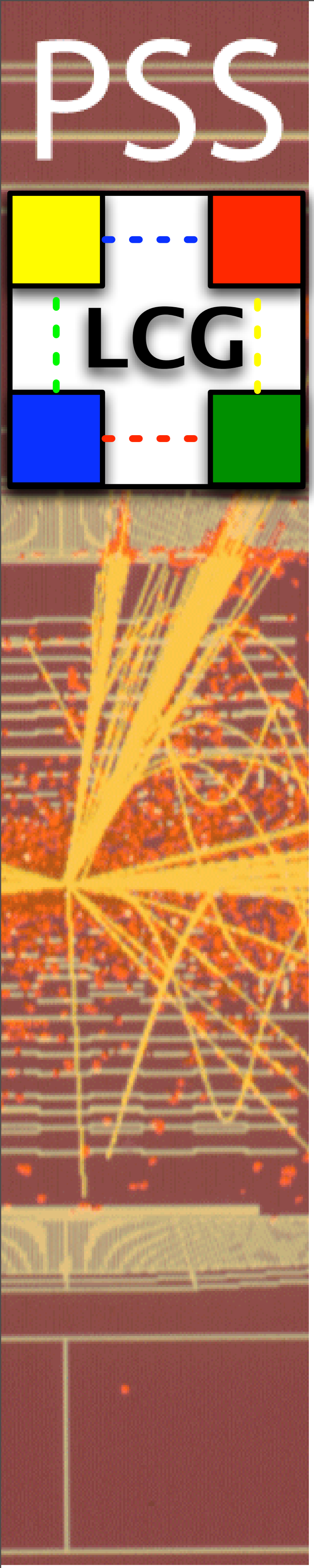


Distributed Database Access in the LHC Computing Grid with CORAL

Dirk Duellmann, CERN IT

on behalf of the CORAL team

(R. Chytracek, D. Duellmann, G. Govi, I. Papadopoulos, Z. Xie)

http://pool.cern.ch & http://pool.cern.ch/coral

CHEP '07, September 6th

Victoria, Canada

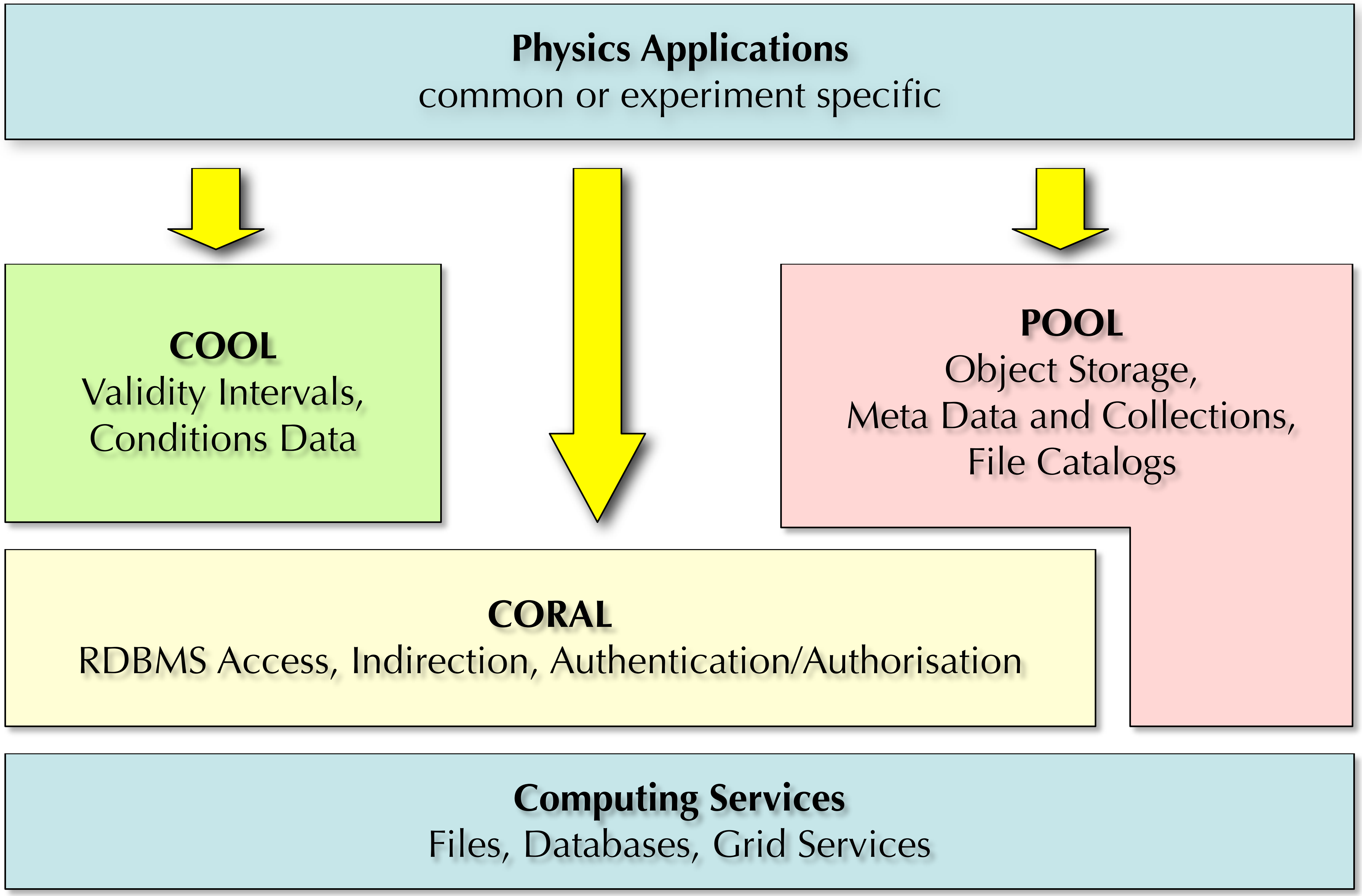
Other connected contributions at CHEP'07

- 171 - Production Experience with Distributed Deployment of Databases for the LHC Computing Grid
- 172 - CERN Database Services for the LHC Computing Grid
- 122 - Relational databases for conditions data and event selection in ATLAS
- 161 - Building a Scalable Event-Level Metadata System for ATLAS
- 186 - Development, Deployment and Operations of ATLAS Databases
- 319 - Alignment data streams for the ATLAS Inner Detector
- 333 - Large Scale Access Tests and Online Interfaces to ATLAS Conditions Databases
- 90 - ATLAS Conditions Database Experience with the LCG COOL Conditions Database Project
- 236 - Control and monitoring of alignment data for the ATLAS endcap Muon Spectrometer at the LHC
- 294 - An Inconvenient Truth: file-level metadata and in-file metadata caching in the (file-agnostic) ATLAS distributed event store
- 462 - Database architecture for the calibration of ATLAS Monitored Drift Tube Chambers
- 358 - Distributed Interactive Access to Large Amount of Relational Data
- 430 - The ATLAS METADATA INTERFACE
- 110 - Experience and Lessons learnt from running high availability databases on Network Attached Storage
- 109 - Oracle RAC (Real Application Cluster) application scalability, experience with PVSS and methodology
- 265 - LHCb Distributed Conditions Database
- 213 - LHCb experience with LFC database replication
- 89 - LHCb Online Interface to a Conditions Database
- 322 - CMS Conditions Data Access using FroNTier
- 325 - The CMS Dataset Bookkeeping Service
- 182 - Distributed Database Access in the LHC Computing Grid with CORAL
- 181 - Development Status and Plans for the LCG Common Database Access Layer (CORAL)
- 204 - COOL Software Development and Service Deployment Status
- 205 - COOL Performance Tests and Optimization
- 350 - Nightly builds and software distribution in the LCG / AA / SPI project
- 447 - Implementing a Modular Framework in a Conditions Database Explorer for ATLAS
- 292 - Explicit state representation and the ATLAS event data model: theory and practice

CORAL - A Database Foundation Layer

- Started as component of POOL- now packaged independently
  - POOL/COOL use CORAL
  - CORAL does not require either (online use)
- CORAL goals
  - database foundation for physics applications
    - local area and grid
    - all LCG AA platforms and database back-ends
  - high level interface
    - C++ and Python API
    - abstraction from SQL dialects of db vendors and connection technologies, avoids risk of vendor binding
  - high level services
    - authentication, authorisation, db service look-up, retry and fail-over (see talk #182 for details)

Between Applications and Services



Physics Applications
common or experiment specific
COOL
Validity Intervals,
Conditions Data
POOL
Object Storage,
Meta Data and Collections,
File Catalogs
CORAL
RDBMS Access, Indirection, Authentication/Authorisation
Computing Services
Files, Databases, Grid Services

Applications Using CORAL

- LCG Persistency framework components POOL and COOL
  - fully replaced direct database dependencies with CORAL
    - POOL File catalogs
    - POOL Event collections
    - COOL conditions database
- CORAL is also used directly
  - ATLAS: detector geometry several online
  - CMS: conditions data, POPCON framework
  - LHCb: online applications
  - Astrophysics: LSST project is evaluating CORAL for storage of large volume image data

Database Back-ends

- Oracle - 10g R2
  - based on C level interface (OCI)
    - performance and flexibility in compiler choice
  - all API concepts / optimisations supported natively
    - bind variables, bulk operations, session pooling
- MySQL - V5 (and 4)
  - small to medium sized services
  - standalone development set-ups
- SQLight - V3.4
  - server-less, file based access
  - popular transport/replication medium for read-only data
- FroNTier/SQUID - caching layer
  - read-only access to Oracle via http
  - multi-level caching between server and client to avoid network and server latencies for repeated queries

SQL-free API

Example 1: Table creation

```
coral::ISchema& schema = session.nominalSchema();
coral::TableDescription tableDescription;
tableDescription.setName( “T_t” );
tableDescription.insertColumn( “I”, “long long” );
tableDescription.insertColumn( “X”, “double” );
schema.createTable( tableDescription);
```

Oracle

```
CREATE TABLE “T_t”     ( I NUMBER(20),
                         X BINARY_DOUBLE)
```

MySQL

```
CREATE TABLE T_t       ( I BIGINT,
                         X DOUBLE PRECISION)
```

SQL-free API cont.

Example 2: Issuing a query

```
coral::ITable& table = schema.tableHandle( “T_t” );
coral::IQuery* query = table.newQuery();
query->addToOutputList( “X” );
query->addToOrderList( “I” );
query->limitReturnedRows( 5 );
coral::ICursor& cursor = query->execute();
```

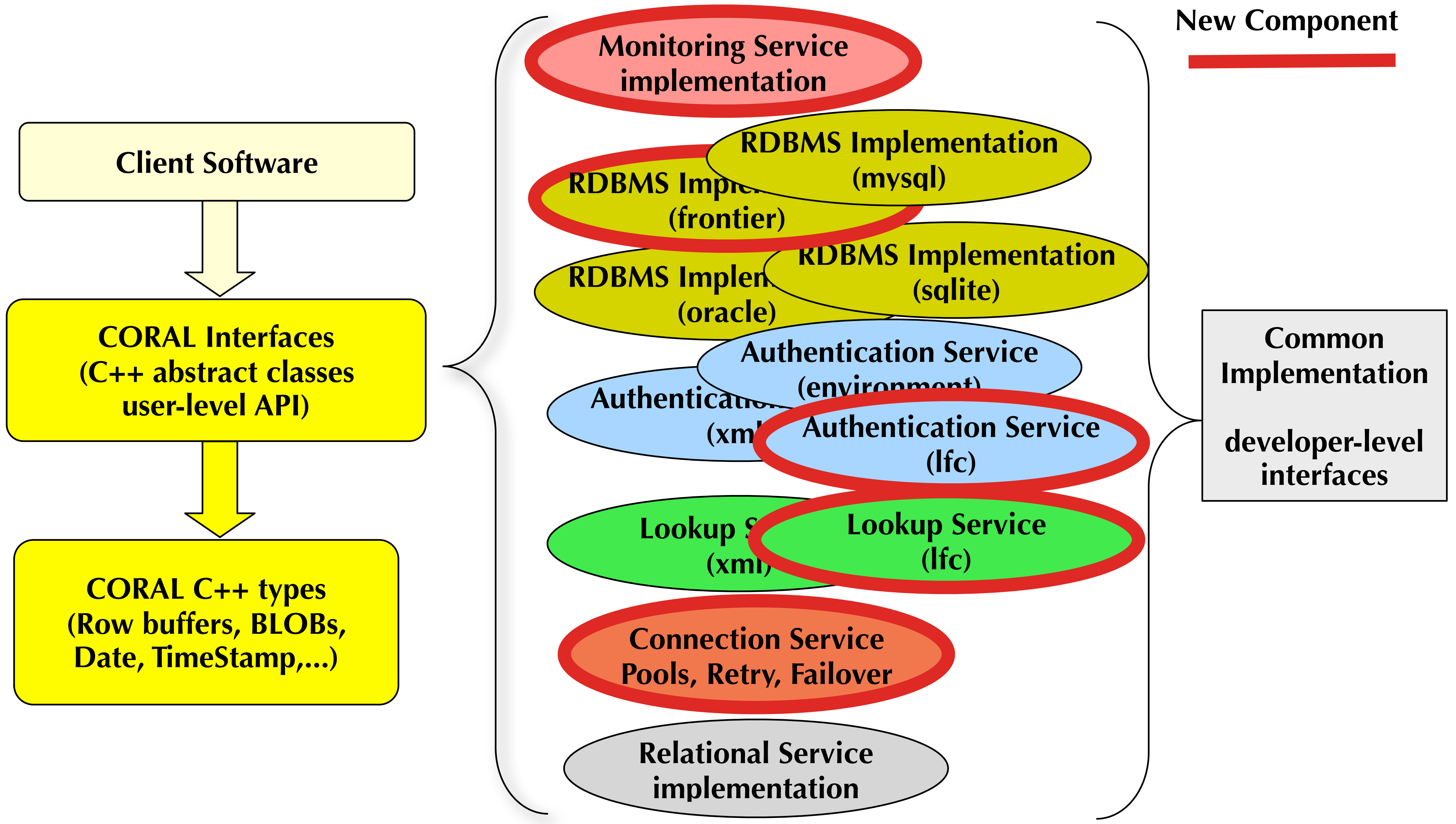
Oracle

MySQL

```
SELECT * FROM
( SELECT X FROM “T_t” ORDER BY I )
WHERE ROWNUM < 6
```

```
SELECT X FROM T_t ORDER BY I LIMIT 5
```

CORAL Components


New Component
Client Software
CORAL Interfaces (C++ abstract classes user-level API)
CORAL C++ types (Row buffers, BLOBs, Date, TimeStamp,...)
Monitoring Service implementation
RDBMS Implementation (mysql)
RDBMS Implementation (frontier)
RDBMS Implementation (sqlite)
RDBMS Implementation (oracle)
Authentication Service (environment)
Authentication Service (xml)
Authentication Service (lfc)
Lookup Service (xml)
Lookup Service (lfc)
Connection Service Pools, Retry, Failover
Relational Service implementation
Common Implementation
developer-level interfaces

Plug-in libraries, loaded at run-time, interacting only through the interfaces

Database Service Lookup & Connection Service

- Problem: DB clients need to obtain physical DB connection parameters (aka “connect string”)
  - more than a host name as technology specific optimisation and retrial parameters are specified
  - handle DB replica selection connection retry and fail-over in case of problems
  - avoid hard-coding of connection parameters or user credentials
- Several CORAL plug-ins can be selected at runtime to handle the above in different computing environments
  - via shell environment (assuming you control machine access)
  - via XML files (assuming you can control file access)
  - via LFC catalog (certificates to protect DB access credentials)
- CORAL maintains a client side pool of connections
  - network connection is shared by several logical DB sessions
  - authentication overhead is minimised

Database Authentication

- Different requirements in different areas of physics computing
  - Online databases - controlled environment, few applications,users and roles
    - User/password pairs still manageable
    - Integration between online and offline accounts is an issue
  - Offline data production - reconstruction/simulation
    - More users (and roles), apps environment shared with grid
    - DB passwords still widely used, but with security issues
  - Grid jobs - reprocessing, end user analysis
    - Large user community, spreading many organisation boundaries
    - User/password pairs can not
      - be shipped with job to a worker node
      - be maintained for a large virtual organisation
- Certificates (X.509 proxy cert's) are used by scientific grids
  - support from commercial vendors still missing
- CORAL support the above via a set of plug-ins coupling to grid services as required

Database Authorisation

- After successful authentication CORAL will match the application read/write or update requests against the available roles for a user
- CORAL enables the appropriate database privileges for a database table or schema based on the VOMS roles in the user certificate
  - Allows to develop general applications insuring proper data protection according to the VO access model
  - Administration of database roles can be based on the existing VO administration tools (VOMS)

Building reliable Applications and Services with CORAL

- The reliability of a composite service depends crucially on core services (eg DB server)
  - but can easily exceed those if a valid retry and fail-over mechanisms are used
  - Unfortunately this often happens only late in the application development or is only incomplete
  - Many different retry and fail-over strategies complicate deployment of db apps
- CORAL implements a consistent retry and fail-over strategy which can be parametrised according to the application needs
  - parameters are kept in one place (eg service look-up file or DB catalog)
- This does not completely eliminate the need for specialised handling of error conditions during data manipulation
- but covers the most frequent problems to obtain a database connection (eg on a overloaded or temporarily unavailable DB server) without loosing the execution state of an application/service

Summary

- Since last CHEP CORAL passed a phase of active development
  - The user API has been very stable, but many DB deployment improvements have been implemented
  - Client-side connection pool, LFC based service look-up and authentication, Python binding and DB Copy Tools
- The package is widely used by Persistency Framework and experiment code, is well integrated with the LCG database and grid services and provides a complete foundation for database applications
- Next Steps and future improvements
  - Remove dependency on SEAL from Persistency Framework
  - Investigating a multi-threaded CORAL server to further improve scalability and security for large scale LHC database deployments
  - Existing CORAL applications are expected to profit with minimal/ no code changes