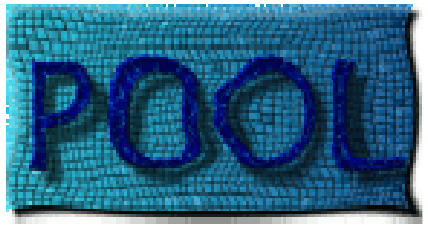


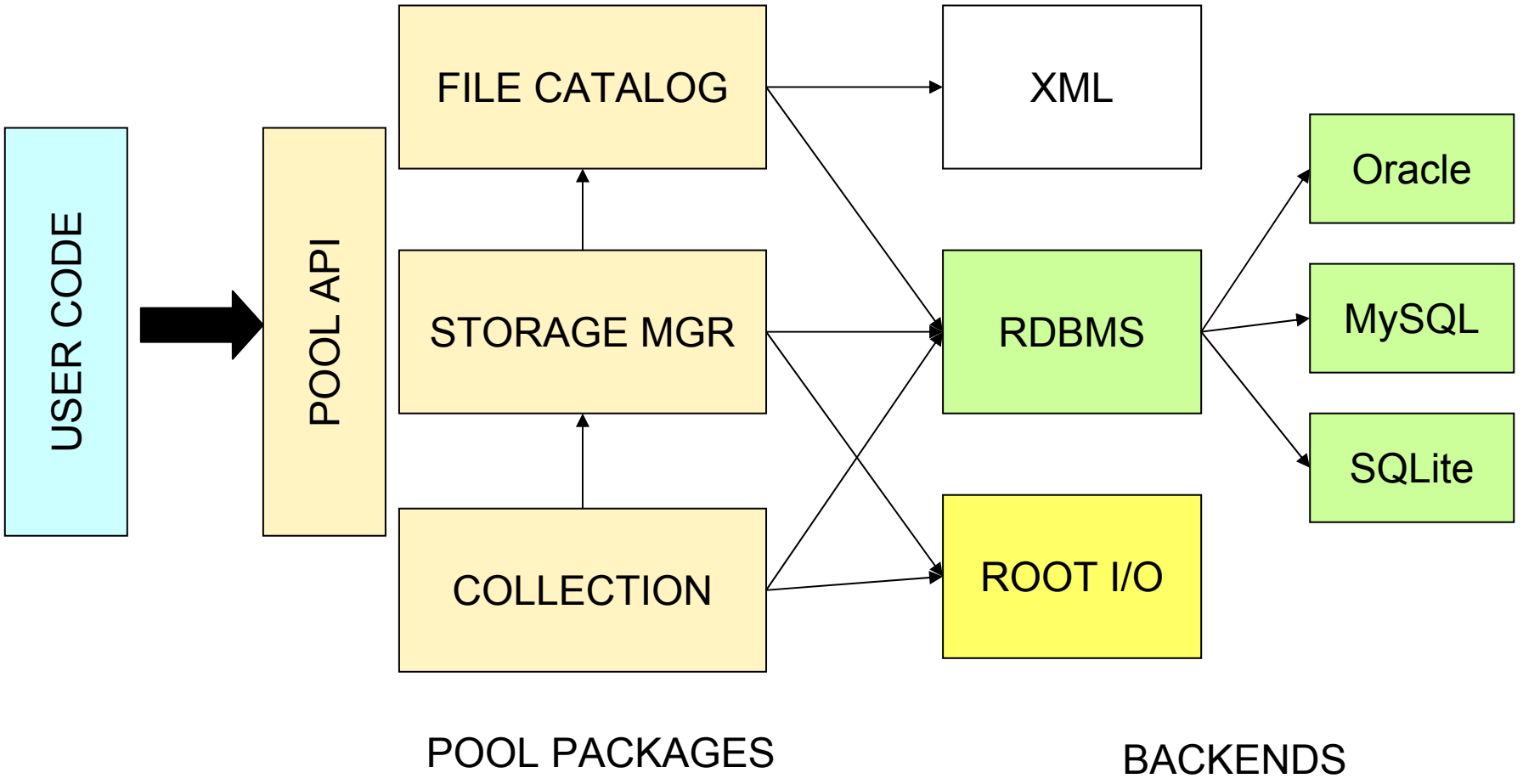
POOL/CORAL
Status and Plans

Giacomo Govi

CERN IT

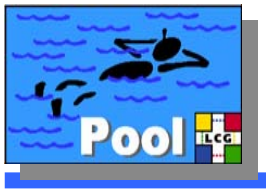
On behalf of POOL project

POOL Domains



Data categories & storage backends

- **ROOT I/O** based backend targeted for complex data structure
  - event data
  - analysis data
- **XML** used for simple data structure in local computing environment
  - catalogue data
- **RDBMS** more natural choice for non-event data
  - conditions, calibration, alignment, detector description
  - possibly produced by online systems
  - frequently involved in selection queries
- POOL provides two levels of RDBMS access (C++)
  - API for general data accessing and manipulating ->CORAL
  - Interface to handle storage of C++ objects -> ORA



CORAL

COmmon Relational Abstraction Layer

- a **C++**, **SQL-free**, **technology-independent** API for accessing and manipulating RDBMS schemata and data

Usage of RAL extended beyond the scope of POOL

- COOL: 'internal' client released separately
- ATLAS: direct access of relational data

Package (and release) the libraries independently of the rest of POOL components

With new features addressing **deployment and distribution** of relational data:

- service indirection, secure authentication mechanisms, client-side monitoring, client-side connection pooling, etc.)

CORAL design concepts

- Technology insulation achieved with abstract interfaces
  - with a minimal, complete set of functionality
  - client components only depends on them
- (New) AttributeList interface used for the description and the handling of the relational data
  - only C++ (no SQL) types exposed
  - type converters responsible for default and user-defined type conversion
- SQL Fragments left only in the WHERE clauses of queries and DML
  - variable binding allowed (and recommended!)

SQL-free access

Creating a table

MYSQL

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

ORACLE

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

CORAL (C++)

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

Issuing a query

MYSQL

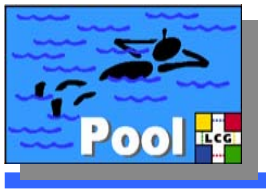
```
SELECT X FROM T_t ORDER BY I LIMIT 5
```

ORACLE

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

CORAL (C++)

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



RDBMS manipulation

- Schema definition and manipulation
  - Creation and manipulation of tables and views, indices, keys (single and multi columns), constraints
  - Describe existing schema elements
- Data manipulation
  - Insert, modify and delete rows
  - Support of I/O for LOBs
- Queries
  - Involving one or more tables
  - Row ordering, limiting
  - Sub queries

Highlights of the CORAL API

- Bulk operations
  - round-trips to the server are minimized in insert/update/delete operations.
- Using 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 plugins
  - users may concentrate on the functionality of their own use cases.

DB connection

Uniform connection protocol

- Explicit contact string specifying technology and protocol:
- No authentication parameters
- Logical database service name: a lookup service provides the corresponding contact string
- An internal service selects transparently the plugin to use

Client-side connection pooling

Authentication

- Explicit, specifying user name and passwords
- Implicit, via a dedicate service
- Integration of authentication based on Grid certificates may follow

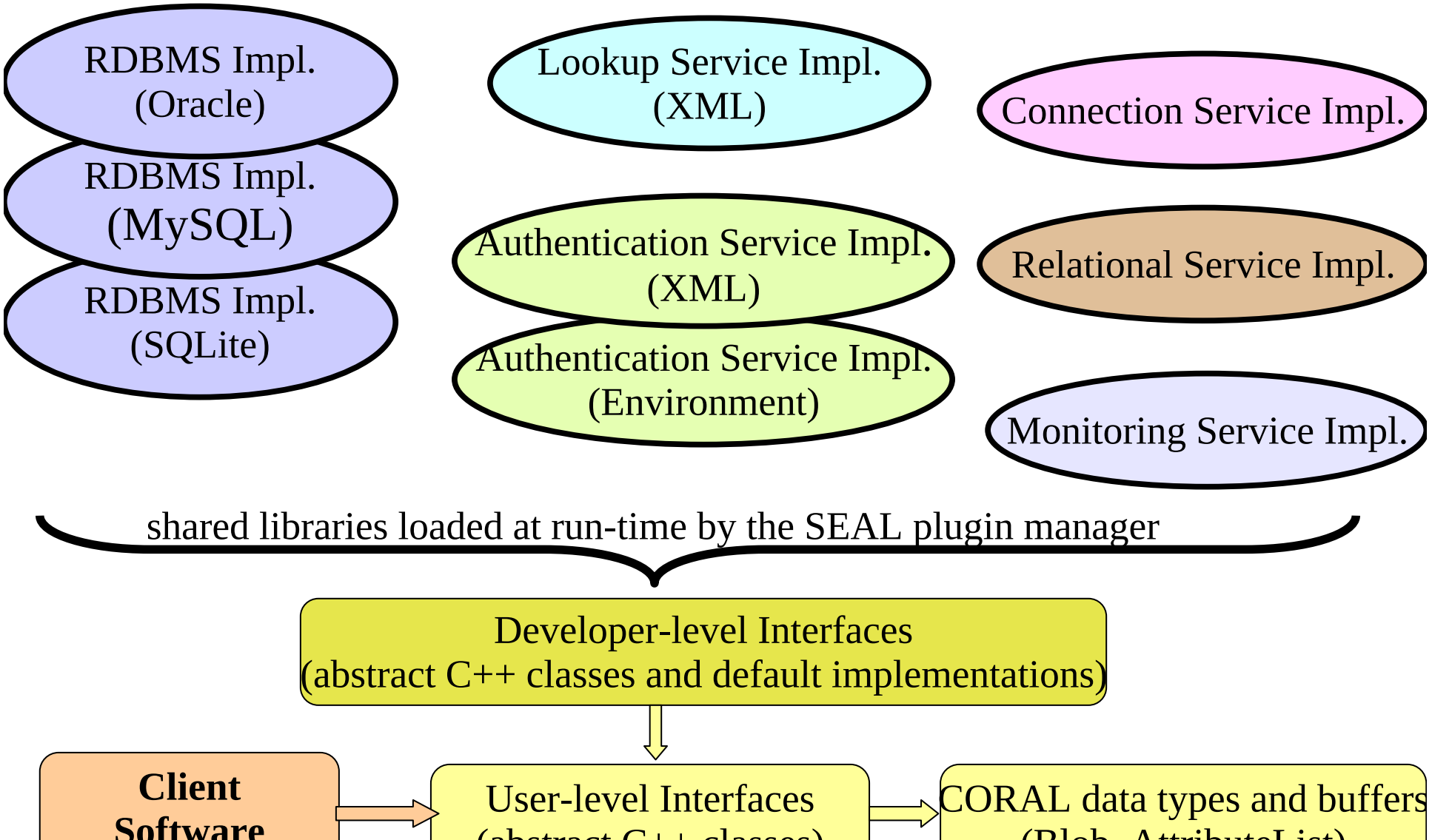
-see talk by Kuba Zajaczkowski

Monitoring

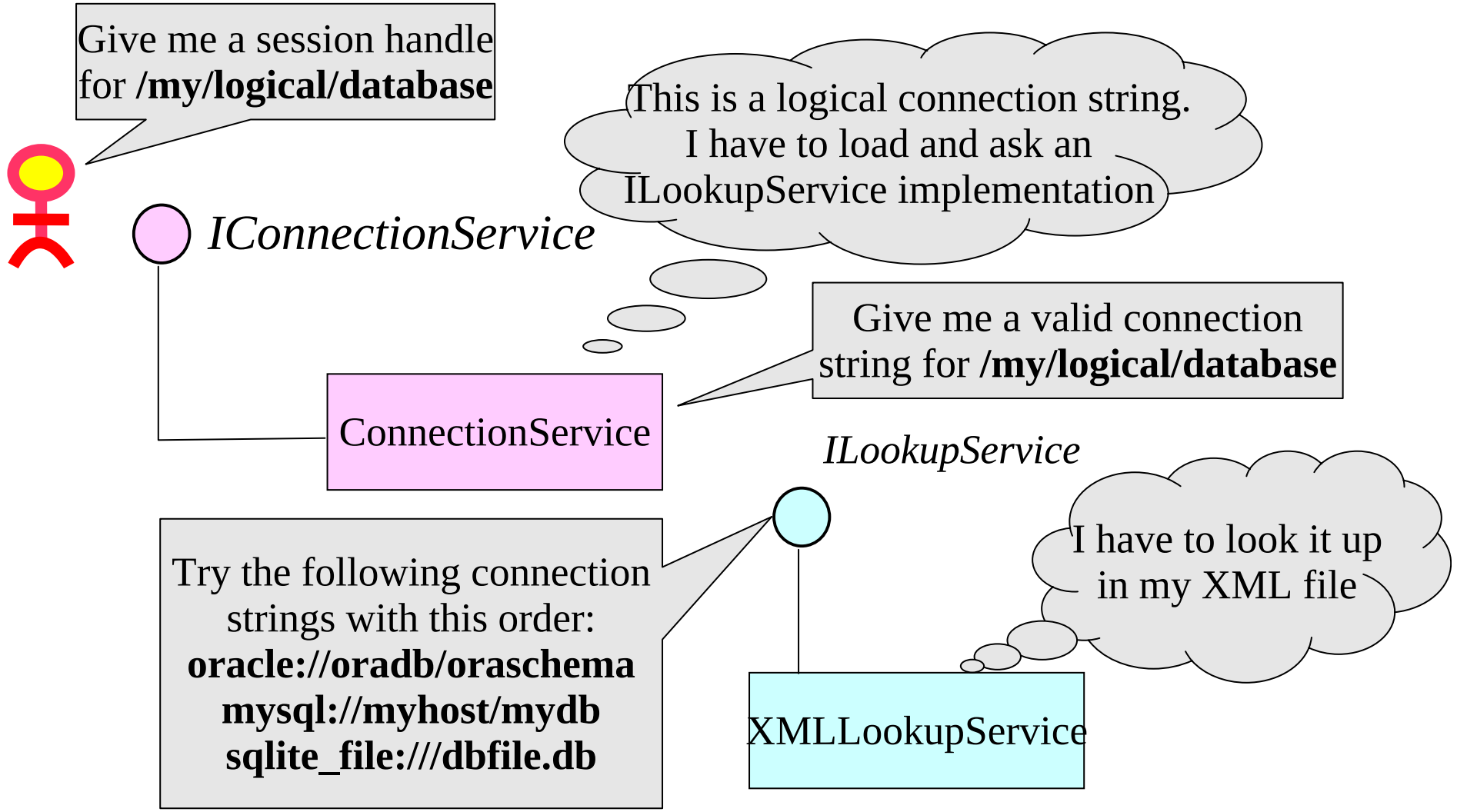
Client-side monitoring

- Flexible verbosity level
- General interface for monitoring service to register information about interesting events:
  - begin-end of sessions
  - transactions
  - response time
  - …
- Collected data can be pushed into any monitoring system by implementing the interface
- A default implementation is provided as a simple data place holder

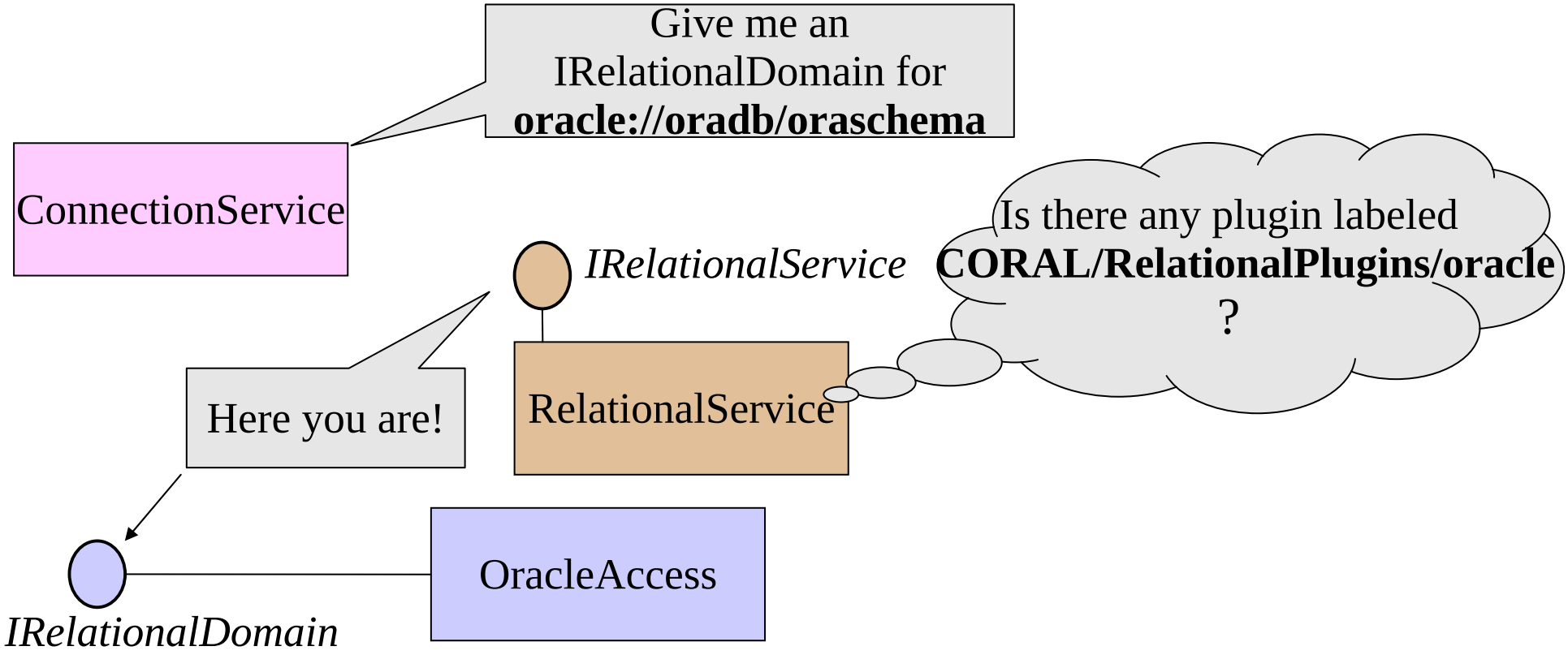
CORAL components



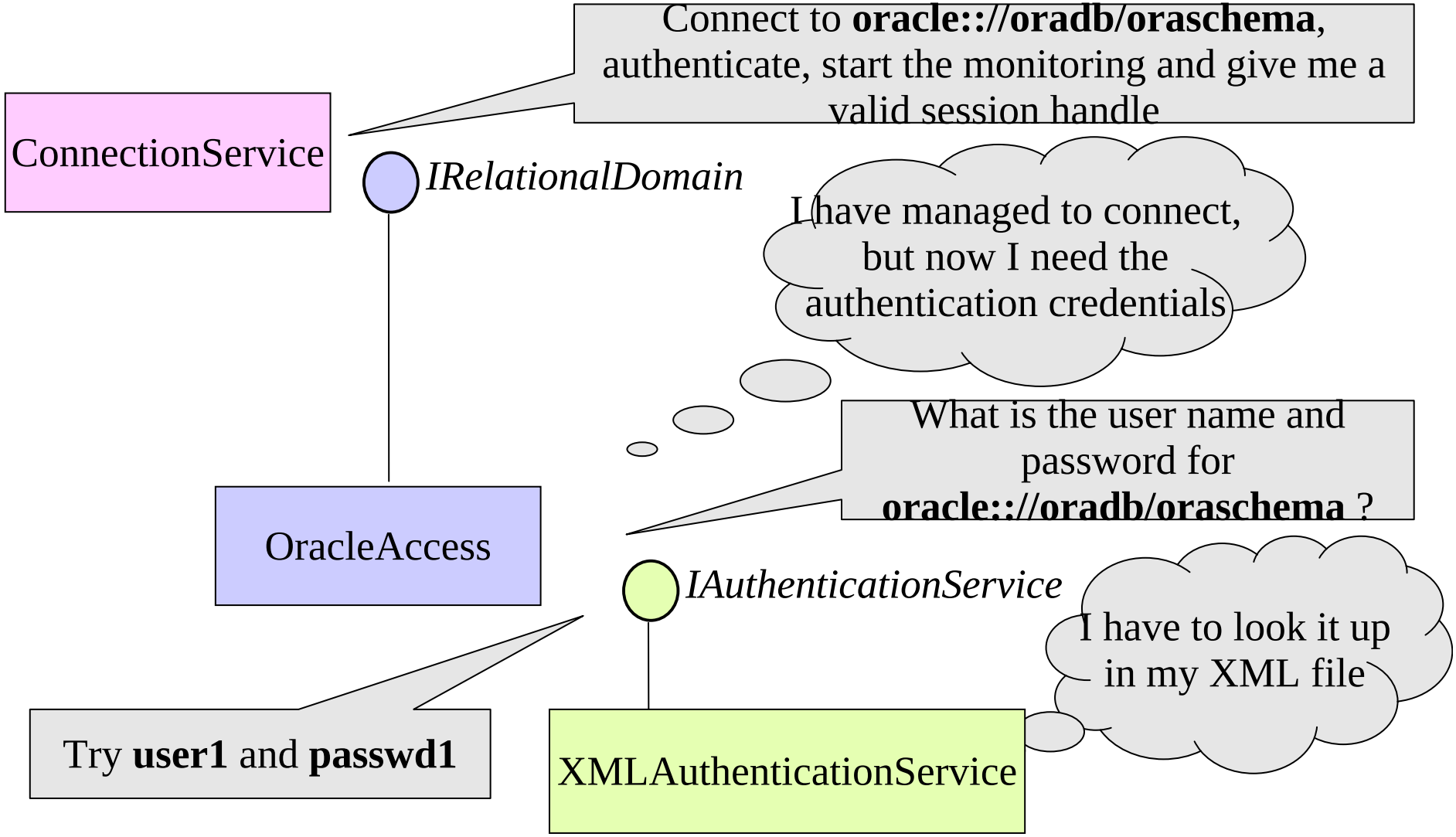
Opening a db session (I)



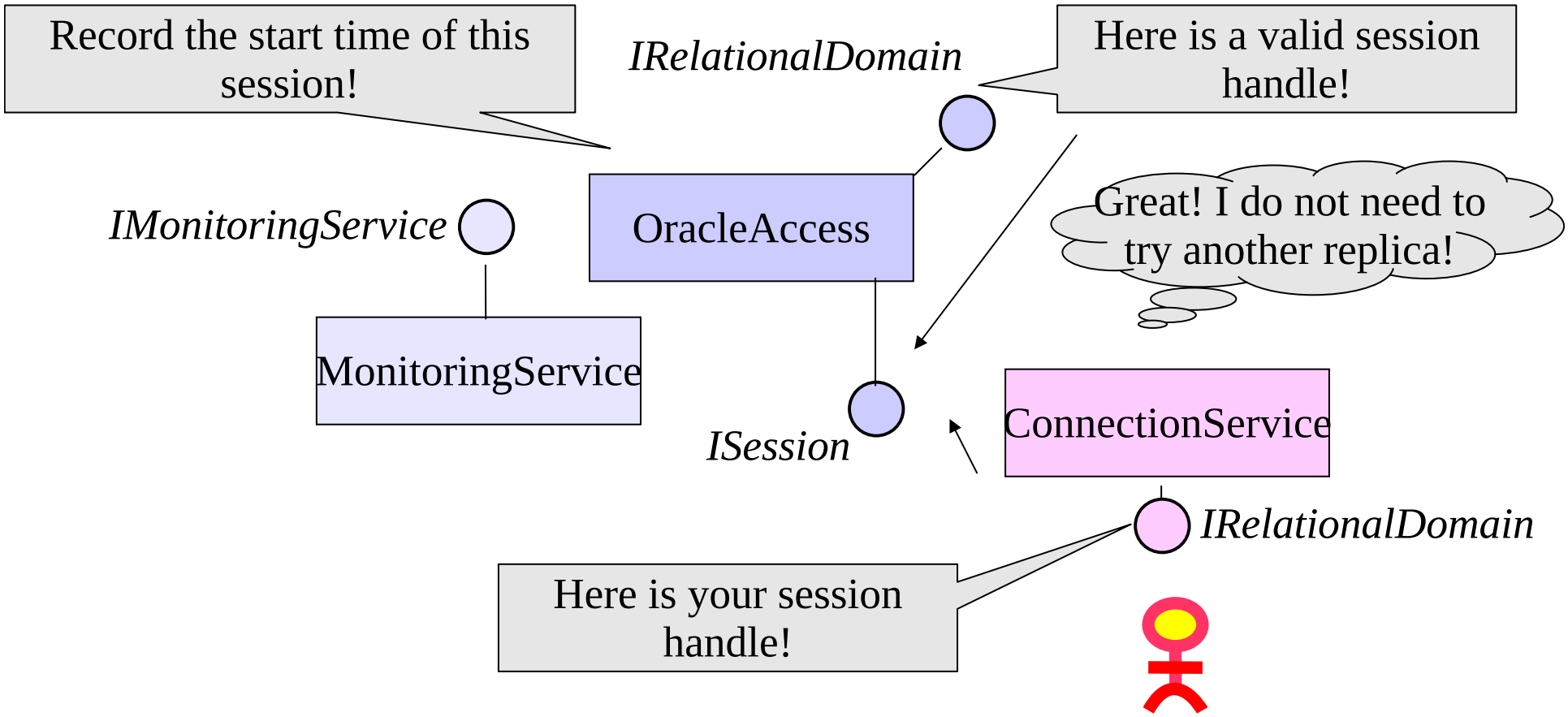
Opening a db session (II)



Opening a db session (III)



Opening a db session (IV)



POOL Object Relational Access

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

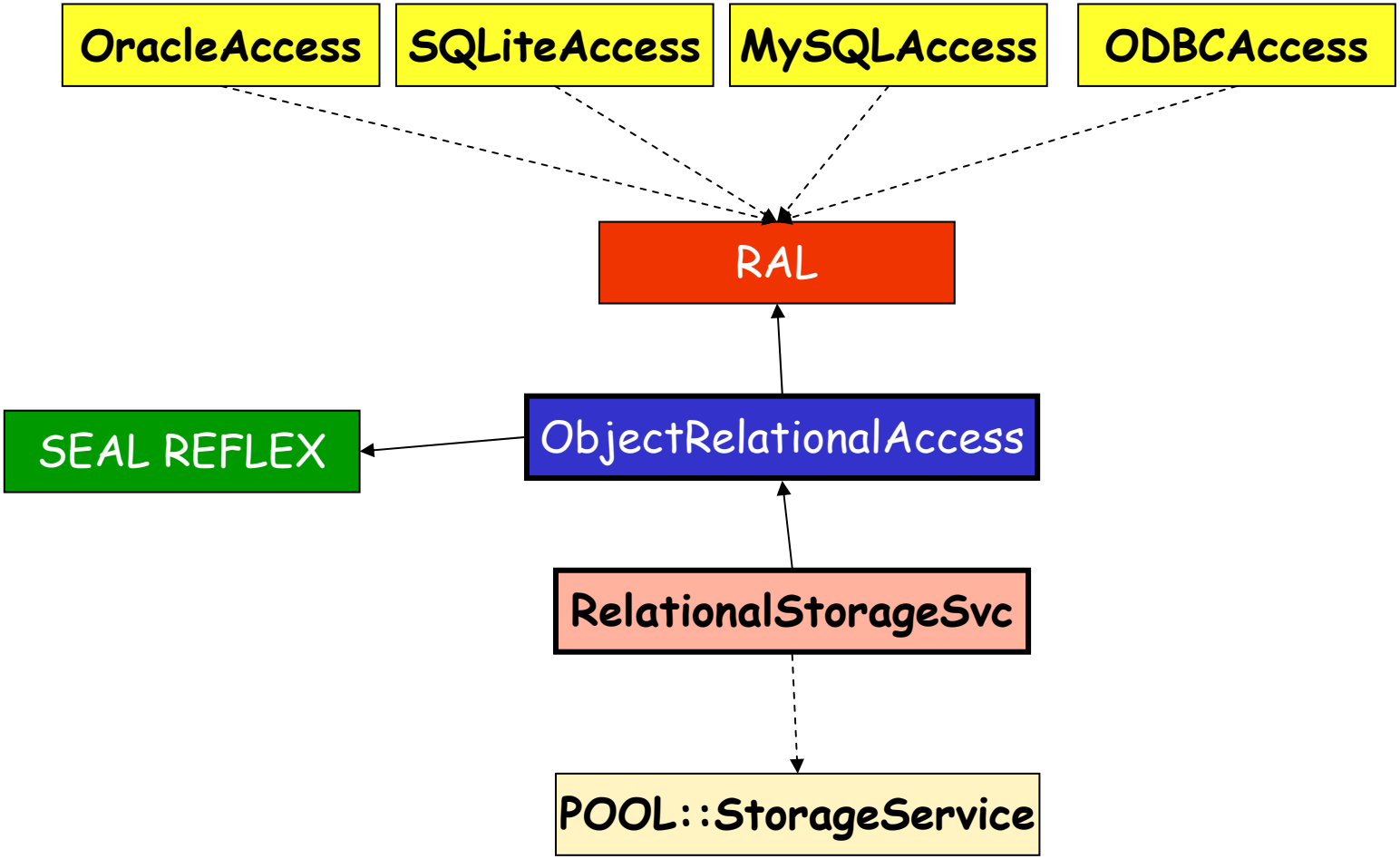
Write and Read complex data structures into/from relational DBs

- 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

POOL relational layer



POOL persistency concepts

- A POOL “object”:
  - Mapping version
  - Value(s) of indexed parameter(s)
- A POOL “container”
  - A table holding the values of the “object” structure
- A POOL “database”
  - Oracle user schema
  - MySQL database
  - SQLite file

Object storing objects into RDBMS

- How to map classes ↔ tables ?
  - C++ and SQL describe data layout with very different constraints/aims
- Objects need an unique identifier (persistent address)
  - allows fast navigation
  - requires unique index for addressable objects
  - part of mapping definition
- Vectors can be stored
  - Currently with the elements in individual records
  - BLOB-based storage coming soon
- Mapping has to be stored with the object data
  - more mapping versions may be needed
- Natural support of schema evolution
  - Adapting the reading of previously written data through a proper user-defined mapping

Mapping example

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

T_A

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

f.k. constraint

T_A_V

ID	PO	V
1	1	0.12
1	2	12.2
2	1	32.1
2	2	0.3

This is only one of the possible mappings!

Mapping elements

- A mapping :
  - Version
  - Hierarchical tree of mapping elements
- An element:
  - Element type (“Object”, “Primitive”, “Array”, “POOL reference”, “Pointer”)
  - Associated table name
  - Associated variable name
  - Associated variable type
  - Associated columns
  - Associated mapping elements
- Everything is stored in 3 relational tables

Mapping generation

- Prerequisites :
  - C++ class(es) already defined
  - The SEAL dictionary libraries already generated
- A tool is provided for the user-driven mapping generation:
  - XML input file to
    - Select the C++ classes
    - Override default mapping rules
    - Define the mapping version
  - Mapping gets “materialized” and stored in the database

Guided object storage

- Object I/O via the ObjectRelationalPersistency interface
  - For every object I/O operation the client has to supply:
    - the corresponding SEAL dictionary for the object's class
    - the object/relational mapping
    - the “persistent address” (eg. the value of the primary key in the table corresponding to the object's class)
  - Object data stored/retrieved following the SEAL dictionary information, and finding the corresponding entries in the mapping
    - Many schema evolution cases can be treated transparently through this mechanism

Reading existing data as C++ objects

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!
- 'Dry run' option

IOV and payload data can be treated as associated objects

Issues:

- updates from the online side could be asynchronous.
  An automatic mechanism to trigger the update on the POOL side has to be studied.
- deletion of rows potentially breaks references

Summary

POOL framework provides transparent access to RDBMS based persistency

- CORAL API for general data manipulation
- Object Relational Storage manager for storage and retrieve of data as C++ objects

Strong focus on features addressing deployment issues

- Connection with indirection to support multi-replica across technologies
- Transparent authentication
- Client-side connection pooling
- Interface for client-side monitoring
- Use and promote best practices in the DB client code

Concrete use case of C++ objects I/O into RDBMS

- Including the capability to read existing data

CORAL backend plugins

- Oracle
  - Based on OCI
  - CORAL interface fully supported natively
  - Linux platforms (Win32 will follow)
- SQLite
  - a light-weight embeddable SQL database engine
  - file based (zero configuration, administration)
  - Linux and Win32 platfroms
- MySQL
  - available implementation based on MyODBC driver
  - implementation based on the native MySQL driver coming soon…