Developing C++ applications using Oracle OCI

Lessons learnt from CORAL

Andrea Valassi
Raffaello Trentadue
(IT-ES)

CERN Oracle tutorial, 8th June 2012

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Agenda

- **What is CORAL?**
- **Why OCI (instead of OCCI) from C++?**
- **A walk-through the OCI features used in CORAL**
  - Will only cover (some of!) the OCI features used in CORAL
    - Focus on 'tricky' issues rather than present a proper tutorial – sorry ☹
  - Will also point out a few Oracle client bugs affecting CORAL

- ***Disclaimer: the present CORAL team does not include the original authors of the OracleAccess plugins***
  - *We learnt a lot by 'reverse engineering' it during its maintenance…*
  - *Some strengths and weaknesses of the original implementation have only become apparent with several years of production use*

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

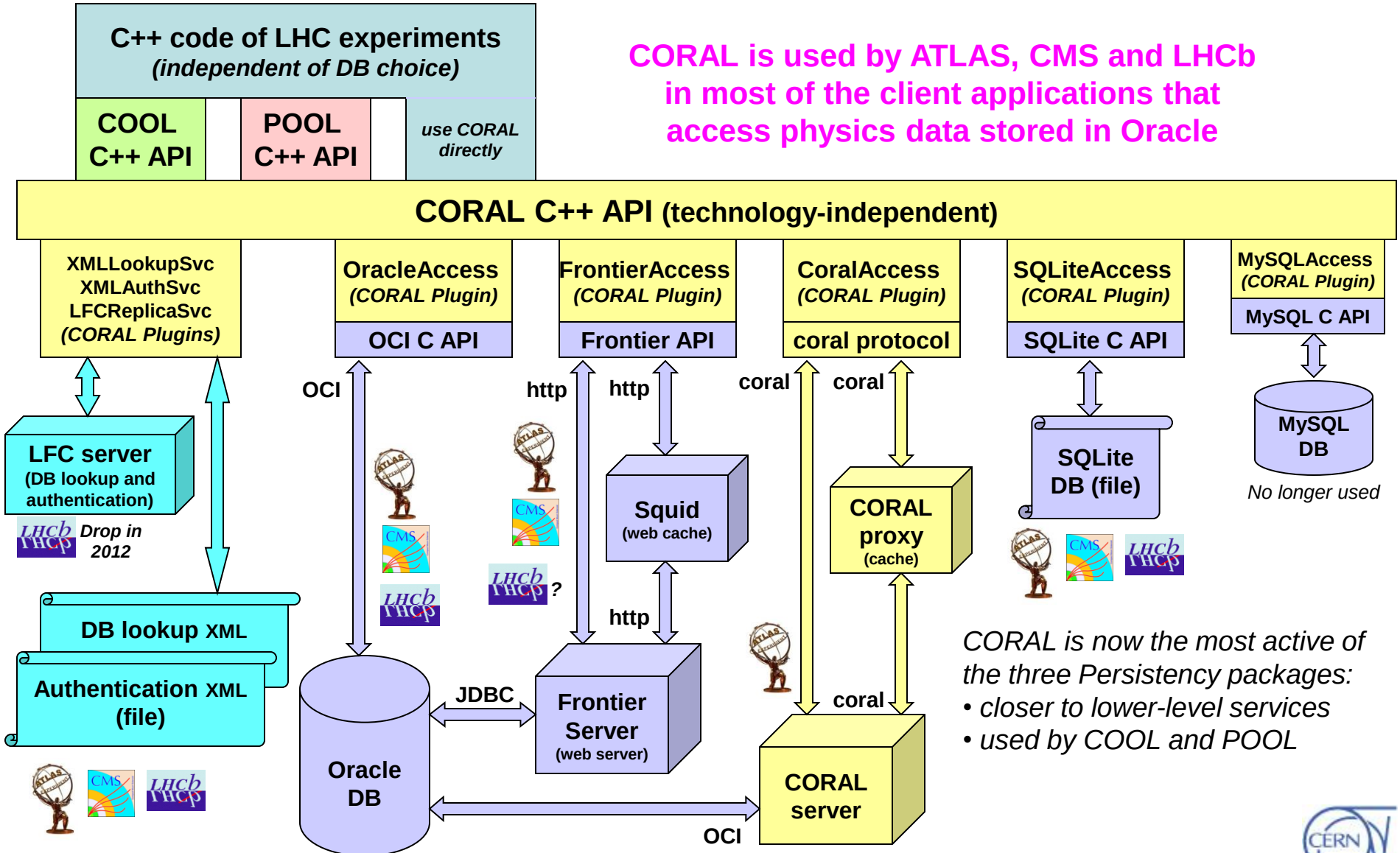
What is CORAL?

CERN IT Department

- **The Common Relational Abstraction Layer (CORAL) is a C++ framework to access data in relational databases**
  - It is used by three LHC experiments (ATLAS, CMS and LHCb) to store and retrieve conditions data and other types of relational data
  - With COOL and CORAL it is part of the Persistency Framework common project (joint development of IT and LHC experiments)
- **Its C++ API is a set of SQL-free abstract interfaces that isolate the user code from the implementation technology**
  - CORAL supports several back-ends (Oracle, MySQL, SQLite…)
    - Users write the same code for all back-ends
    - A detailed knowledge of the many SQL flavors is not required
    - The SQL commands specific to each backend are executed by the relevant library, loaded at run-time by a C++ plugin infrastructure
- **Oracle is accessed in CORAL via the OracleAccess library**
  - *The CORAL OracleAccess library is implemented using OCI*

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

CORAL components



Why OCI? (the problem with OCCI)

- **Why should we develop C++ applications using OCI?**
- **An Oracle C++ library does exist: OCCI**
  - Used extensively for COMPASS and HARP Objectivity migration
  - OCCI does have a very nice API (much nicer in C++ than OCI…)
- **OCCI is a C++ library built only for some C++ compilers**
  - Problem in the past: OCCI for gcc2.96, we needed 2.95.2 or 3.2.3
  - Now better (several compilers), but still limited to gcc4.3 and ???…
    - *It would be different if Oracle released OCCI sources… it does not ☹*
- **OCI is a C library – it can be used with any C++ compiler**
  - OCI is the basic client library, used internally by all Oracle tools
    - e.g. also one of the java drivers for Oracle is based on OCI
  - CORAL OracleAccess, based on OCI, is used and must be built on Linux for gcc4.3, gcc4.6, gcc4.7, icc11, clang3.0…

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

OCI workflow in CORAL

- **Environment bootstrap: OCIEnvCreate**
  - Several options (e.g. to configure multi-threaded clients, as in CORAL)
  - Runtime libraries (libociei or libocicus) are loaded here (see strace)
- ***Setup – allocate handles (all along): OCIHandleAlloc***
  - *Set relations between handles: OCIAttrSet*
- **Connect to the DB server: OCIServerAttach** *(CORAL Connection)*
  - One such physical connection can be used for many user sessions
- **Authenticate user session: OCISessionBegin** *(CORAL Session)*
  - Via username/password in CORAL
    - X509 proxy certificates would need a middle tier (e.g. CORAL server)
- **Start the transaction: OCITransStart**
  - This is only needed for updates, not for read-only use cases…
- **Prepare the SQL statement: OCIStmtPrepare2**

- **Execute the SQL statement: OCIStmtExecute**
  - SQL (SELECT, DDL, DML), PL/SQL (as CALL xxx), control (ALTER SESSION…)
  - Iterate on cursor: OCIStmtFetch2

- **Release the SQL statement: OCIStmtRelease**
- **Commit the transaction: OCITransCommit**
- **Close the session: OCISessionEnd**
- **Disconnect the server: OCIServerDetach**
- ***Cleanup – release handles (all along): OCIHandleFree***
  - *Unset relations between handles: OCIAttrSet*

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

OCI connections/sessions in CORAL

CERN IT Department

0 ***Initialize OCI*** *(e.g. load Oracle client libraries)*
- ***Create OCIEnv*** *(via OCIEnvCreate)*
- ***Create OCIError*** *(via OCIHandleAlloc)*

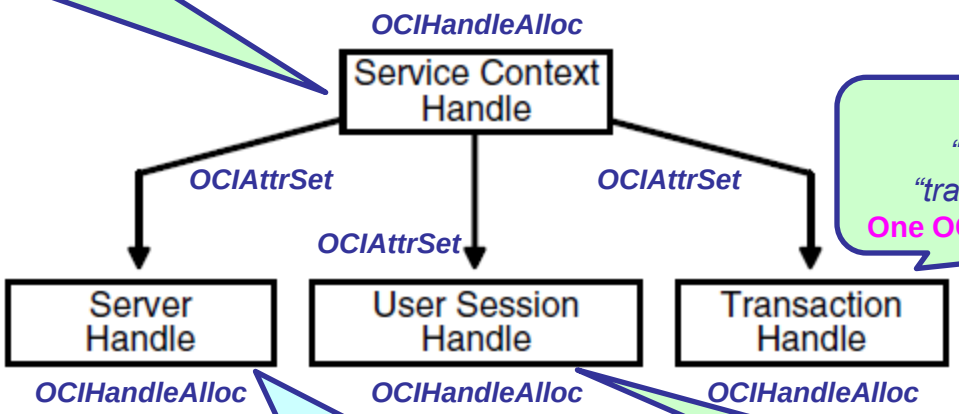
2 ***Set up logical user session on DB server***
- ***Create OCISvcCtx*** *(via OCIHandleAlloc)*
- ***Link server connection to it*** *(via OCIAttrSet)*
- ***Create OCISession*** *(via OCIHandleAlloc)*
- ***Link it to service context*** *(via OCIAttrSet)*
- ***Set session's user and password*** *(via 2 x OCIAttrSet)*
- ***Begin user session*** *(via OCISessionBegin)*
- ***Create OCITrans*** *(via OCIHandleAlloc)*
- ***Link it to service context*** *(via OCIAttrSet)*
- ***Start transaction*** *(via OCITransStart)*

OCISvcCtx
"service context handle"
One OCISvcCtx per CORAL Session

Connection sharing?

- One OCIServer
(One CORAL Connection)

- Many sets of (OCISvcCtx, OCISession, OCITrans) using the same OCIServer
(Many CORAL Sessions)



OCITrans
"transaction handle" or "transaction context handle"
One OCITrans per CORAL Session

OCIServer
"server handle" or *"server context handle"*
One OCIServer per CORAL Connection

OCISession
"session handle" or "session context handle"
One OCISession per CORAL Session

1 ***Set up physical connection to DB server***
- ***Create OCIServer*** *(via OCIHandleAlloc)*
- ***Establish connection*** *(via OCIServerAttach)*

Transactions – comments

CERN IT Department

- **CORAL presently calls OCITrans also for read-only use cases**
  - This is equivalent to 'SET TRANSACTION READ ONLY'
  - These are serializable read-only transactions
    - A SELECT statement issued at time t1 will always return a result set describing the state of the database at the time t0 (< t1) when the transaction started
  - Advantages: all data can be cached (e.g. used in COOL)
  - Disadvantages: several
    - Heavy operation on the redo logs (has been seen to lead to ORA-01555)
    - Querying newly created tables in COOL tests leads to OCI-01466 (Oracle bug)
    - Sometimes you do want to read newly added data (e.g. ATLAS HLT)

- **Summary: calling OCITrans is NOT needed in read-only use cases**
  - Not clear to me if originally this was intentional or unintentional in CORAL
  - We plan to properly support three use cases in CORAL eventually:
    - RW transactions, serializable RO transactions, "non-serializable RO" transactions
    - You may simply skip OCITrans and execute your SQL directly (e.g. ATLAS HLT)
    - See https://twiki.cern.ch/twiki/bin/view/Persistency/CoralReadOnlyModes

Data handling – comments

CERN IT Department

- **Bind variables *(recommended in CORAL to reduce server load!)***
  - OCIBindByPos, OCIBindByName, OCIBindArrayOfStruct, OCIBindDynamic
- **Data types**
  - Simple data types (C++ numbers/strings)
    - Described in OCI calls by SQLT_INT, SQLT_FLT, SQLT_STR…
  - Oracle special data types (e.g. DATE)
  - BLOBs and CLOBs *(for C++ strings longer than 4000 characters)*
    - OCILobOpen, OCILobClose, OCILobRead, OCILobWriteAppend…
  - No C++ "object" types in CORAL
    - More generally, no user defined types and no OTT (Object Type Translator)
- **Mapping C++ and SQL/OCI data types?**
  - Custom implementation in CORAL (based on C++ types, e.g. "long")
    - And a slightly different one in COOL (based on storage precision, e.g. int32, int64)
  - Personally I am somewhat suspicious of wildly generic auto-translations
    - The C++ program flow and *SQL query optimization* are two separate areas that both need a lot of care
    - Developing C++ code 'letting another layer take care of all SQL translations' brings the risk of poor schema design and poor query performance
    - You are the developer, you are in charge of SQL query performance!

Cleanup phase – comments

- **Release OCI handles and break relations in the right order**
  - reverse order to that used for allocating and associating handles
- **Crashes observed in an older CORAL implementation**
  - if a connection is broken and the corresponding handle is released before an associated (e.g. statement) handle is used/released
  - patched by fixes/workarounds in CORAL since then
  - strictly speaking some of these are Oracle client bugs
    - the crash (segmentation fault) is deep inside the Oracle client
    - the Oracle client should do its internal bookkeeping to be able to tell if a first OCI handle has gone out of scope when a second OCI handle tries to access it inside the implementation of an Oracle method…
    - I would expect an ORA-24xxx or ORA-031xx error rather than a crash

The OCI client library – comments

- **The OCI library is definitely more 'portable' than the OCCI library**
  - e.g. no need to respect a specific libstdc++ dependency
- **But also the OCI library (libclntsh.so) does have some limitations**
  - if you run your client software on non-standard O/S (e.g. SLC)
  - if client applications have multiple dependencies (e.g. LHC experiments)
  - different issues from those seen when installing the server software on very controlled environments (e.g. strict RedHat Enterprise Linux)
- **Example 1 (Oracle bug – fixed): SELinux**
  - out-of-the-box, the 11.2.0.1.0 library could not be loaded due to SELinux
  - patched version (for OCI and OCCI too) is installed for the LHC experiments
    - see /afs/cern.ch/sw/lcg/external/oracle/11.2.0.1.0p3/doc/README_11.2.0.1.0p3.txt
- **Examples 2/3 (Oracle bugs – not yet fixed): GSSAPI and Kerberos**
  - the OCI library redefines its own custom versions of some security libraries (GSSAPI and Kerberos), conflicting with the version provided by the O/S
  - user applications that need Oracle AND another security-aware component (e.g. xrootd) may fail one or the other depending on the loading order…
  - we were lucky so far and we could find workarounds… for how long?
    - a strategy to fix exists: using 'versioned symbols' (recently used by EMI middleware to fix a similar bug in Globus due to GSSAPI)

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

11g advanced topics – not yet used

CERN IT Department

- **OCI callbacks for Transparent Application Failover (TAF)**
  - CORAL has its own custom implementation to react to “network glitches”
    - Oracle TAF is mainly about instance crashes – failover to another instance
    - CORAL implementation is mainly about temporary network/database glitches
- **OCI client result cache**
  - Would be an interesting area to test
  - CORAL caches data via Frontier/Squid and CoralServer/Proxy middle tiers

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Python access: PyCoral and PyCool

CERN IT Department

- **PyCool – Python bindings for COOL**
  - Implemented automatically via ROOT (Reflex, PyCintex, PyRoot)
    - ROOT will internally move this from CINT to llvm by the end of 2012
  - Easy to maintain, ~nothing to do when a class is added/modified
- **PyCoral – Python bindings for CORAL**
  - Implemented manually (class by class) via low-level C calls
  - Very painful to maintain, may merge with PyCool eventually
- **Both PyCoral and PyCool allow Oracle access from Python**
  - Ultimately via the OCI implementation in the OracleAccess C++

Conclusions

CERN IT Department

- **To access Oracle from C++:**
  - you could use OCCI (but this may limit your set of compilers)
  - or you can use OCI
  - or you can also use CORAL directly ☺
- **References**
  - Oracle Call Interface (OCI) 11g Programmer's Guide (html, pdf)
  - CORAL twiki, savannah and CVS
    - All OCI in CORAL can be found in the OracleAccess library in CVS
    - A standalone example of OCI connection sharing is in the COOL CVS