

# Quality assurance for CORAL and COOL

within the LCG software stack  
for the LHC experiments

Andrea Valassi  
(CERN IT-SDC)

“Developers@CERN” Forum – 29<sup>th</sup> September 2015



# Outline

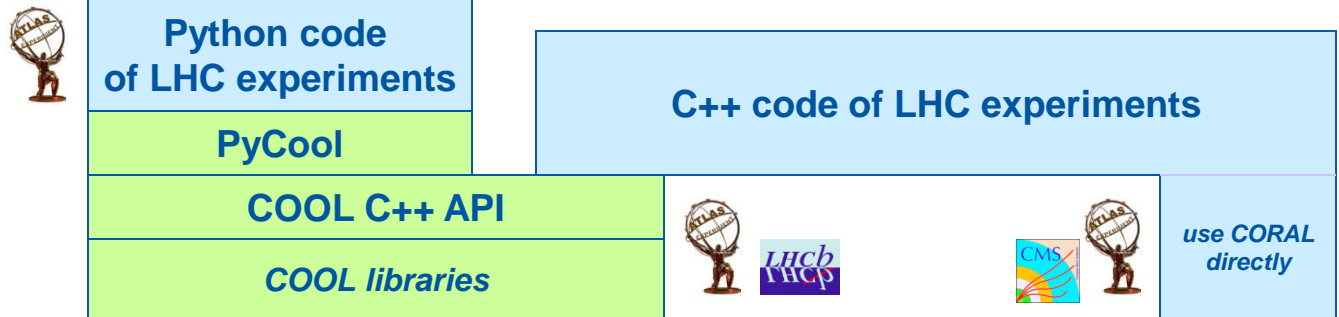
- Introduction to CORAL and COOL
- Software quality and testing for CORAL and COOL
  - Functional tests
  - Performance tests
- Conclusions

# CORAL & COOL – introduction

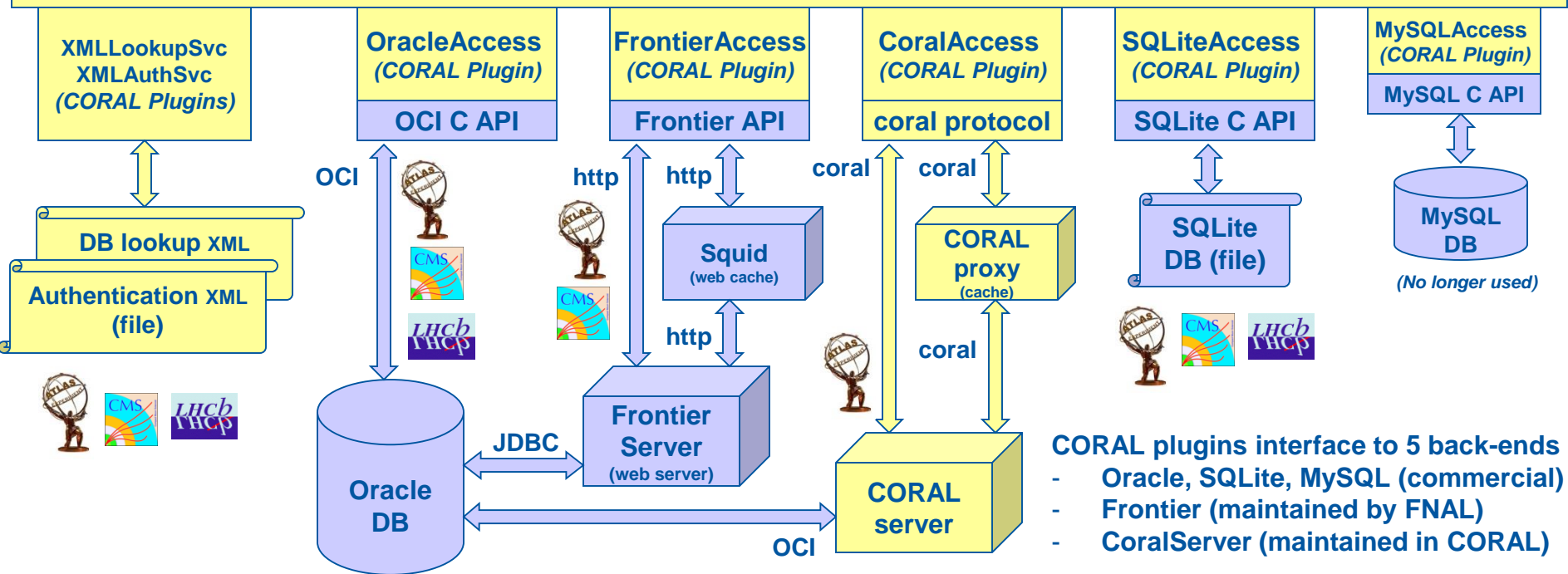
- **CORAL**: a generic *relational database* access layer
  - Used by ATLAS, CMS and LHCb and internally by COOL
    - Conditions data, trigger configuration data, geometry data...
    - Main entry point to physics data in Oracle (directly or via Frontier)
- **COOL**: a set of libraries and tools for handling time variation and versioning in “*conditions databases*”
  - Used by ATLAS and LHCb throughout LHC data taking
    - Example: detector calibration for Sep 2015 from latest algorithm
- Both COOL and CORAL are written in **C++**
  - Bindings for **Python** also exist (PyCool and PyCoral)
  - COOL also implies developing and optimizing **SQL** queries

# Component architecture

CORAL and COOL mainly provide *client software components* (CORAL server / proxy are the only exceptions)



## CORAL C++ API (DB technology independent)



- CORAL plugins interface to 5 back-ends
- Oracle, SQLite, MySQL (commercial)
  - Frontier (maintained by FNAL)
  - CoralServer (maintained in CORAL)

# CORAL – generic DB access layer

**CORAL provides a DB-independent and largely (but not completely) SQL-free C++ API for accessing data stored using relational DB technologies. It takes care of executing user requests using the appropriate SQL.**

```
coral::ConnectionService svc;  
session = svc.connect( URL, coral::Update );  
session.transaction().start( false );  
coral::TableDescription tableDescription;  
tableDescription.setName( "myTable" );  
tableDescription.insertColumn( "ID", "long long" );  
tableDescription.insertColumn( "Data", "double" );  
tableDescription.setPrimaryKey( "ID" );  
session.nominalSchema().createTable( tableDescription );
```

*Example: table creation through the CORAL API*

**SQLite**

*URL = "sqlite\_file:mydatabase.db"*

```
CREATE TABLE "myTable"  
( "ID" SLONGLONG ,  
  "Data" DOUBLE ,  
  PRIMARY KEY("ID") )
```

**Oracle**

*URL = "oracle://server/myschema"*

```
CREATE TABLE MYSCHEMA."myTable"  
( "ID" NUMBER(20),  
  "Data" BINARY_DOUBLE,  
  CONSTRAINT "myTable_PK"  
  PRIMARY KEY ( "ID" ) )
```

**MySQL**

*URL = "mysql://server/myschema"*

```
CREATE TABLE "myschema"."myTable"  
( "ID" BIGINT NOT NULL,  
  "Data" DOUBLE PRECISION,  
  CONSTRAINT "myTable"  
  PRIMARY KEY ( "ID" ) )
```

# COOL – conditions data model

While CORAL is a generic access layer, COOL includes the *design of a relational schema* and the optimization of SQL queries for a specific data model for conditions data

## Each COOL conditions object has

### – Metadata (system-controlled)

- Data item identifier
- Interval-of-validity [since, until]
- Version information

### – Data “payload” (user-defined)

- Physics values (temperatures, calibration parameters...)
- Separate columns or a CLOB

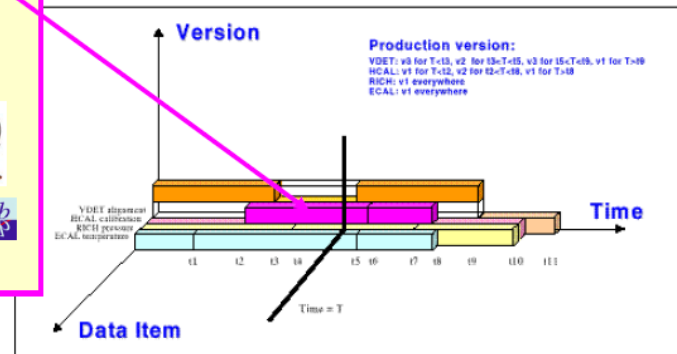


Figure 1 The three axes for identifying uniquely each data item in the condition database

COOL provides a technology-independent C++ API to handle the time variation and versioning of the conditions data of the LHC experiments. This is implemented using relational databases, based on CORAL.

# Lifecycle and collaborations

- CORAL and COOL **started over 10 years ago** (2004)
  - As separate subprojects of LCG Persistency Framework
    - Two teams, overall O(10) developers from **IT and LHC experiments**
    - Example of successful common software project of LHC experiments
  - Now managed together and **largely in maintenance mode**
    - Single team with minimal manpower, overall less than O(1 FTE)
- External collaborations for infrastructure and testing
  - **PH-SFT**: external packages, nightly/release builds and tests
    - See [yesterday's presentation](#) by Patricia Mendez Lorenzo
  - **IT-DB and experiment DBAs**: Oracle service and support
    - Essential interaction during the optimization and testing phases

# Development process overview

- Activity is **driven by LHC experiment requirements**
  - Release on demand, with no predefined schedule
  - CORAL and COOL largely in maintenance mode now
    - Releases mainly due to external software upgrades (e.g. Boost, ROOT), new compiler ports, new infrastructure (e.g. cmake)...
      - The move from ROOT5 to ROOT6 especially triggered a lot of work
    - Essentially no new feature requests, only the occasional bug fixes
- **CORAL & COOL built as part of LCG software stack**
  - Release installation on AFS by PH-SFT (for ATLAS & LHCb)
    - Agreed set of software versions (e.g. LCG\_79) and platforms
    - Internal build with cmake, integration with lcgcmake
    - Nightly builds and tests using Jenkins and CDash



# CORAL & COOL – testing and QA

- A **priority** – but no formal method, just common sense
  - Believe that early tests/QA save you time in the long term!
  - Mainly functional tests, but also other tests and QA policies
- COOL used ~ *test-driven development* from the start
  - Write unit tests as (or before) you write the implementation
    - Deterministic approach – you know what your code should do
    - This also helps the design of interfaces – what should code look like

# Multi-platform support & SW quality

- CORAL & COOL are built on a range of platforms
  - Using *different C++ compilers/linkers* enhances SW quality
    - Same code for all, only minimal platform-dependent sections
  - As a general rule, *all build warnings are fixed*, not ignored

P. Mendez Lorenzo's talk

	Compilers						OS					Release type	
BUILD CHAINS	gcc48	gcc49	gcc51	clang35	icc15	native	CentOS7	SLC6	mac1010	mac109	ubuntu14	Release	Debug
Experimental	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
dev2	✓	✓		✓		✓		✓			✓	✓	✓
dev3	✓	✓				✓	✓	✓			✓	✓	✓
dev4		✓	✓			✓	✓	✓			✓	✓	✓
Releases	✓	✓					✓	✓				✓	✓

Currently: gcc4, gcc5, clang, icc (Linux, Mac)  
In the past: MSVC (Microsoft)

# Nightly builds and tests – CDash

- CDash dashboard: main access to see nightly results
  - Individual CORAL or COOL logs available on failures

LCGSoft

Dashboard Calendar Previous Current Next Project

No file changed as of **Thursday, September 24 2015 - 22:00 UTC**

**39 minutes ago:** 134 tests failed on experimental-x86\_64-slc6-clang35-opt  
**40 minutes ago:** 1 warning introduced on experimental-x86\_64-slc6-clang35-opt  
**40 minutes ago:** 28 errors introduced on experimental-x86\_64-slc6-clang35-opt  
**3 hours ago:** 1 test failed on dev3-x86\_64-slc6-gcc49-dbg  
**4 hours ago:** 1 test failed on dev3-x86\_64-slc6-gcc48-opt

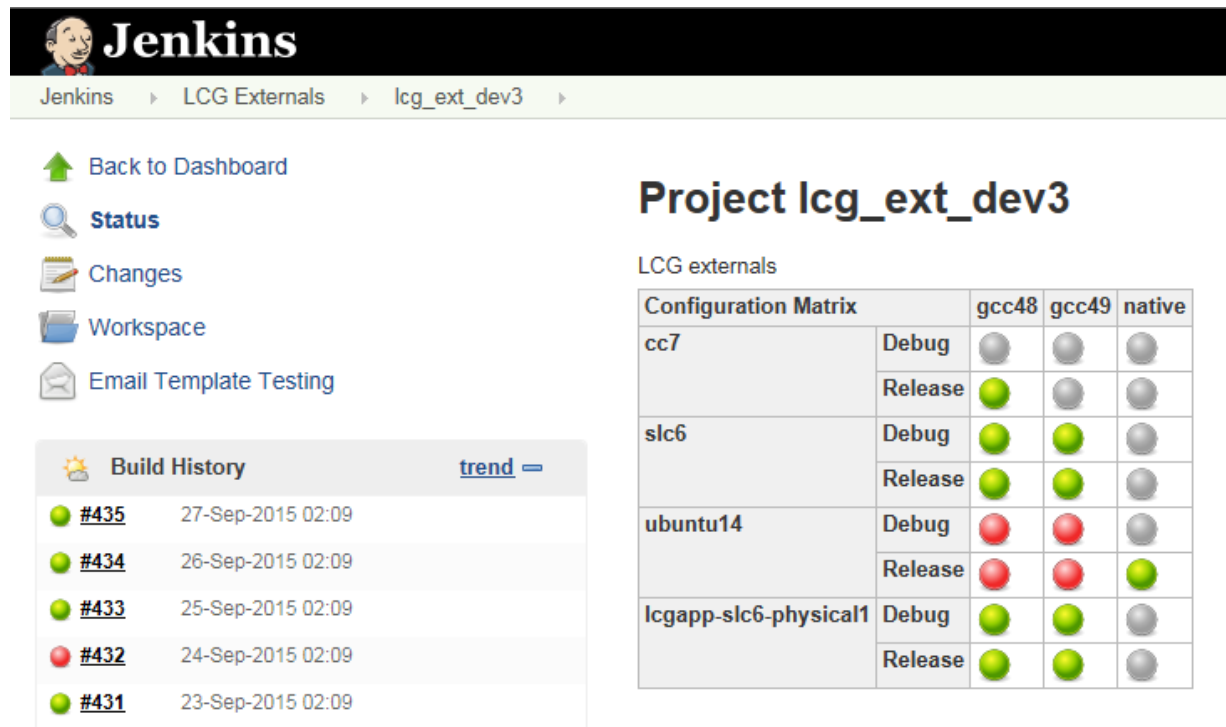
See full feed

Experimental										
Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
lcgapp-slc6-physical1.cern.ch	experimental-x86_64-slc6-icc15-opt	0	0	0	46	1	0	174 <sup>+11</sup>	172 <sub>-11</sub>	Sep 25, 2015 - 16:36 UTC
p01001533x71310.cern.ch	experimental-x86_64-slc6-gcc51-opt	0	0	0	10	1	0	122	224	Sep 25, 2015 - 16:22 UTC
lcgapp-slc6-physical1.cern.ch	experimental-x86_64-slc6-gcc48-opt	0	0	0	7 <sup>+6</sup> <sub>-79</sub>	1	0	96 <sup>+96</sup>	250 <sup>+250</sup>	Sep 25, 2015 - 14:44 UTC
lcgapp-slc6-physical1.cern.ch	experimental-x86_64-slc6-gcc49-dbg	0	0	0	7 <sup>+1</sup> <sub>-280</sub>	1	0	97 <sup>+97</sup>	249 <sup>+249</sup>	Sep 25, 2015 - 12:38 UTC
lcgapp-slc6-physical1.cern.ch	experimental-x86_64-slc6-clang37-opt				49	1				Sep 25, 2015 - 08:09 UTC
macitois13.cern.ch	experimental-x86_64-mac1010-clang61-opt	6	0	1	57 <sup>+9</sup> <sub>-3</sub>	11	0	153 <sup>+38</sup>	158 <sup>+10</sup> <sub>-38</sub>	Sep 25, 2015 - 05:14 UTC
macitois11.cern.ch	experimental-x86_64-mac109-clang60-opt	6	0	1	48	1	0	159 <sup>+34</sup>	152 <sup>+10</sup> <sub>-30</sub>	Sep 25, 2015 - 05:13 UTC
ec-ubuntu-14-04-x86-64-1	experimental-x86_64-ubuntu14-gcc48-opt	0	0	0	9	1	0	111 <sup>+43</sup> <sub>-1</sub>	235 <sup>+11</sup> <sub>-42</sub>	Sep 25, 2015 - 05:04 UTC
lcgapp-slc6-physical1.cern.ch	experimental-x86_64-slc6-clang35-opt	0	0	0	28	1	0	134 <sup>+36</sup>	212 <sup>+10</sup> <sub>-35</sub>	Sep 25, 2015 - 05:04 UTC
dev4										
Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
lcgapp-cc7-x86-64-6.cern.ch	dev4-x86_64-cc7-gcc49-dbg	6	0	0	0	0	0	0	6	Sep 24, 2015 - 23:12 UTC
lcgapp-cc7-x86-64-10.cern.ch	dev4-x86_64-cc7-gcc49-opt	6	0	0	0	0	0	0	6	Sep 24, 2015 - 23:13 UTC
lcgapp-slc6-x86-64-2.cern.ch	dev4-x86_64-slc6-gcc49-dbg	0	0	0	0	0	0	0	6	Sep 24, 2015 - 23:08 UTC
lcgapp-slc6-x86-64-3.cern.ch	dev4-x86_64-slc6-gcc49-opt	6	0	0	0	0	0	0	6	Sep 24, 2015 - 23:35 UTC
lcgapp-slc6-x86-64-8.cern.ch	dev4-x86_64-slc6-gcc51-opt	6	0	0	7	1	0	1	5	Sep 24, 2015 - 23:57 UTC
ec-ubuntu-14-04-x86-64-2	dev4-x86_64-ubuntu14-gcc48-opt	7	0	0	0 <sub>-1</sub>	0 <sub>-1</sub>	0	0 <sub>-2</sub>	6 <sup>+2</sup>	Sep 24, 2015 - 23:22 UTC

Get details of failing package (not necessarily CORAL or COOL!)

# Nightly builds and tests – Jenkins

- The Jenkins dashboard provides additional low-level information that has not been propagated to CDash
  - The full lcgcmake driver log is available here



**Jenkins**

Jenkins > LCG Externals > lcg\_ext\_dev3 >

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Email Template Testing](#)

**Build History** [trend](#)

Build ID	Timestamp
<a href="#">#435</a>	27-Sep-2015 02:09
<a href="#">#434</a>	26-Sep-2015 02:09
<a href="#">#433</a>	25-Sep-2015 02:09
<a href="#">#432</a>	24-Sep-2015 02:09
<a href="#">#431</a>	23-Sep-2015 02:09

**Project lcg\_ext\_dev3**

LCG externals

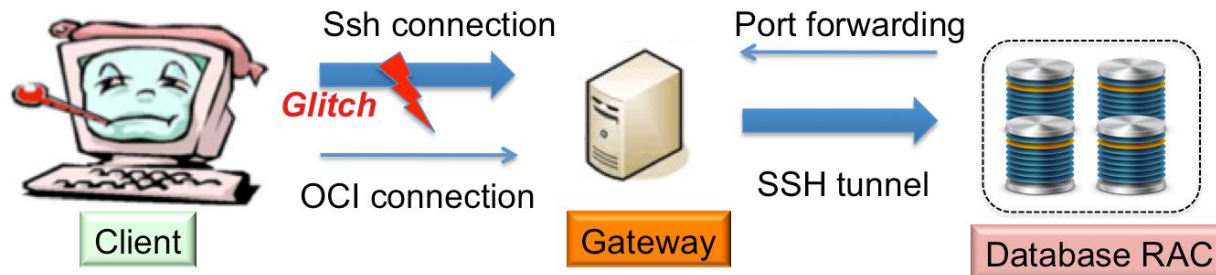
Configuration Matrix		gcc48	gcc49	native
cc7	Debug	⚪	⚪	⚪
	Release	⚪	⚪	⚪
slc6	Debug	⚪	⚪	⚪
	Release	⚪	⚪	⚪
ubuntu14	Debug	⚪	⚪	⚪
	Release	⚪	⚪	⚪
lcgapp-slc6-physical1	Debug	⚪	⚪	⚪
	Release	⚪	⚪	⚪

# Functional test suites

- Mainly **CppUnit (C++)** and **unittest (Python)** assert-based tests
  - Blurred border between “unit” and “integration” tests – name is irrelevant
    - Most tests involve external software packages (ROOT) and services (Oracle)
- Rather **“large” coverage**, but no explicit metric
  - Years ago we used gcov for a while to improve coverage
  - *Bugs reported by users systematically lead to new tests (reproducibility!)*
- All database back-ends are systematically tested
  - Oracle (also with Frontier or CoralServer read-back), SQLite, MySQL
    - Also requires maintenance of server infrastructure (partly dedicated, partly shared)
    - Careful test speed optimization for Oracle (slow DDL from table creation/deletion)
- Functional test suites assembled using **QMTest**
  - Very old, but does the job! – might move to ctest at some point
  - Long suites for full tests (~ 2 hours), shorter for Jenkins/CDash nightlies

# CORAL “network glitch” tests

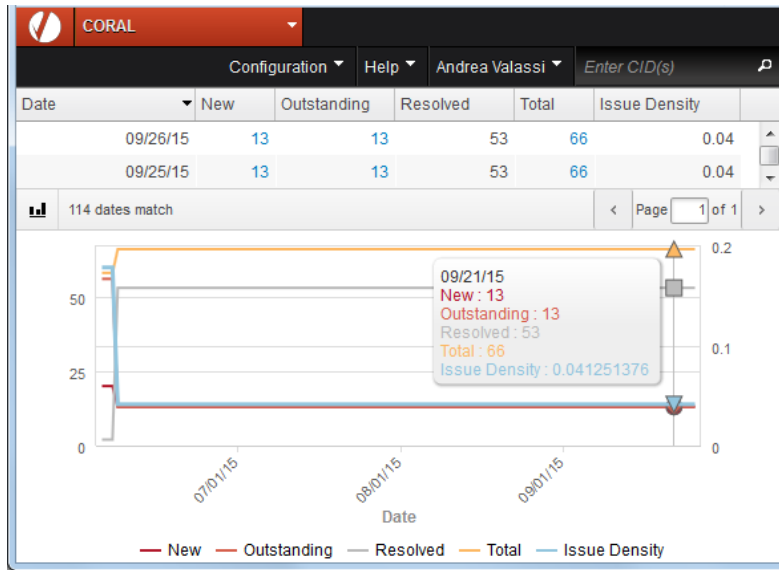
- Complex implementation issues require complex test design!
  - Here’s just one representative example
- A few CORAL segmentation faults observed around 2010
  - Due to buggy legacy implementation of reaction to lost connections
  - **Ad-hoc testing framework was developed to reproduce these issues**
    - Using synchronized killing of SSH tunnels to simulate a lost connection
    - This test framework was essential to allow a proper fix / reimplementaion
    - Now part of the standard functional test suite



# CoralServer tests for ATLAS HLT

- CoralServer is an essential component of the ATLAS High Level Trigger system since data-taking in October 2009
  - From design through implementation to deployment in only 9 months
  - *Aggressive timescale was only possible thanks to rigorous testing!*
- Two complementary sets of tests
  - Artificial ad-hoc unit tests
  - Standalone integration test application using full ATLAS HLT software
- *Need for testing included upfront in modular component design*
  - Clean separation of networking, marshalling and RDBMS components
    - Allowed them to be implemented and tested independently of each other

# Static code analysis



- Coverity (PH-SFT service)
  - Advantage of having own instance: keep history and triage defects!
- One-off scans – *all CORAL or COOL defects now fixed*
  - Ignore issues from external packages (Boost, gcc, Qt...)

The screenshot shows the CORAL web interface displaying a list of outstanding defects. The table has columns: CID, Type, Impact, Status, First Detected, Owner, Classification, Severity, Action, Component, Category, File. The table lists 12 issues, all with a status of 'Triaged'.

CID	Type	Impact	Status	First Detected	Owner	Classification	Severity	Action	Component	Category	File
59117	Recursion in included headers	Low	Triaged	10/14/14	Unassigned	Pending	Minor	Ignore	Boost	Build system issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
57274	Uninitialized pointer field	Medium	Triaged	07/03/14	Unassigned	Pending	Minor	Ignore	gcc	Uninitialized members	/releases/gcc4.9.1/x86_64-slc6/include/c++/4.9.1/functional
57255	Uninitialized pointer field	Medium	Triaged	07/03/14	Unassigned	Pending	Minor	Ignore	gcc	Uninitialized members	/releases/gcc4.9.1/x86_64-slc6/include/c++/4.9.1/functional
54793	Same on both sides	Medium	Triaged	05/13/14	Unassigned	Pending	Minor	Ignore	gcc	Incorrect expression	/releases/gcc4.9.1/x86_64-slc6/include/c++/4.9.1/bits/stl_algo_base.h
43434	Logically dead code	Medium	Triaged	06/27/12	Unassigned	Pending	Minor	Ignore	Boost	Control flow issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
27253	Not restoring ostream format	Medium	Triaged	06/12/12	Unassigned	Pending	Minor	Ignore	Boost	API usage errors	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
27182	Data race condition	Medium	Triaged	06/12/12	Unassigned	Pending	Minor	Ignore	Boost	Concurrent data access violations	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
27162	'Constant variable guards dead code	Low	Triaged	06/12/12	Unassigned	Pending	Minor	Ignore	Boost	Possible Control flow issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
43217	Recursion in included headers	Low	Triaged	06/12/12	Unassigned	Pending	Minor	Ignore	Boost	Build system issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
27378	Recursion in included headers	Low	Triaged	04/16/11	Unassigned	Pending	Minor	Ignore	Boost	Build system issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
25496	Recursion in included headers	Low	Triaged	03/25/11	Unassigned	Pending	Minor	Ignore	Boost	Build system issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost
25495	Recursion in included headers	Low	Triaged	03/25/11	Unassigned	Pending	Minor	Ignore	Boost	Build system issues	/releases/LCG_75root6/Boost/1.55.0_python2.7/x86_64-slc6-gcc49-dbg/include/boost



# Profilers and other tools

- Various profilers have been used for CORAL and COOL
  - Most recently and effectively: *valgrind*, *IgProf* and *gperftools*
  - Mainly ad-hoc “campaigns”, no systematic use of any of these
- Memory profiling – valgrind
  - Successfully used in the past to **fix a few CORAL memory leaks**
  - May run the full test suite through it but seldom do so
- CPU and elapsed time profiling – gperftools
  - Successfully used in the past to **fix CORAL unneeded DB roundtrips**
  - But SQL speed optimization is generally more relevant (next slide)
- And then of course other very different debugging tools
  - When hope is almost lost: *gdb*, *strace*...

# COOL – Oracle performance tests (1/2)

- COOL involves the design and optimization of the relational schema and SQL queries for its data model
  - SQL speed is more relevant than C++ speed for COOL
- Main issue: unstable Oracle SQL execution plans
  - *Comprehensive test and QA strategy ([Oracle tutorial 2013](#))*
    - Stabilization of execution plans using SQL hints
    - One-click tool to **run client tests and build a detailed performance report** including plots and info gathered from server-side trace files
    - No large-scale stress tests: small-scale tests on O(100k) rows are enough to extrapolate scalability (it's only the slope that matters...)

# COOL – Oracle performance report

SQL execution plan  
(from server-side trace file)

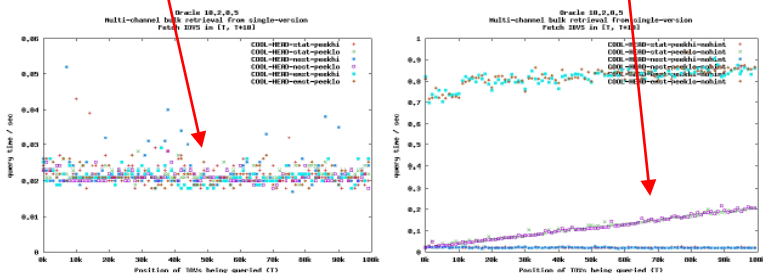
Flat = OK!  
(Production, with hints)

Slope = NOT OK!  
(Control test, no hints)

## 1. Use case SV\_R

Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bit Production  
Instance test11 (Linux x86\_64 on itrac507) on 2011-12-13

(COOL-preview on Oracle 10.2.0.5.0)



**NB: different vertical scales**

Primary SQL statement (hint option=""):

```
SELECT /*+ NO_BIND_AWARE QB_NAME(MAIN) INDEX(@MAIN COOL_J3@MAIN (CHANNEL_ID IOV_SINCE IOV_UNTIL)) LEADING(@MAIN COOL_C2@MAIN COOL_J3@MAIN) USE_NL(@MAIN COOL_J3@MAIN) INDEX(@MAX1 COOL_J1@MAX1 (CHANNEL_ID IOV_SINCE IOV_UNTIL)) */ COOL_J3.OBJECT_ID AS "OBJECT_ID", COOL_J3.CHANNEL_ID AS "CHANNEL_ID", COOL_J3.IOV_SINCE AS "IOV_SINCE", COOL_J3.IOV_UNTIL AS "IOV_UNTIL", COOL_J3.USER_TAG_ID AS "USER_TAG_ID", COOL_J3.SYS_INSTIME AS "SYS_INSTIME", COOL_J3.LASTMOD_DATE AS "LASTMOD_DATE", COOL_J3.ORIGINAL_ID AS "ORIGINAL_ID", COOL_J3.NEW_HEAD_ID AS "NEW_HEAD_ID", COOL_J3.I AS "I", COOL_J3."S4k" AS "S4k" FROM AVALASSI."XSV_RCWD.F0001.CHANNELS" "COOL_C2", AVALASSI."XSV_RCWD.F0001.IOVS" "COOL_J3" WHERE COOL_J3.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_J3.IOV_SINCE>=COALESCE((SELECT /*+ QB_NAME(MAX1) */ MAX(COOL_J1.IOV_SINCE) FROM AVALASSI."XSV_RCWD.F0001.IOVS" COOL_J1 WHERE COOL_J1.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_J1.IOV_SINCE<="since1");,"since1"); AND COOL_J3.IOV_SINCE<="until3" AND COOL_J3.IOV_UNTIL>:"since3u" ORDER BY COOL_J3.CHANNEL_ID ASC, COOL_J3.IOV_SINCE ASC
```

Main hint in alternative SQL statement (hint option='nohint'):

```
/*+ NO_BIND_AWARE QB_NAME(MAIN) */
```

Identified 3 execution plan(s) from 12 trace files (in /tmp/avalassi/ALL/10.2.0.5/SV\_R on lxmrra5001):

Trace file (stat)-(peek)-(hint)	Exec plan	Bind variables				Hints	
		:since1	:since3s	:until3	:since3u	Used	Unused
stat-peekhi	#1	99000	99000	99010	99000	6	0
stat-peeklo	#1	0	0	10	0	6	0
nost-peekhi	#1	99000	99000	99010	99000	6	0
nost-peeklo	#1	0	0	10	0	6	0
emst-peekhi	#1	99000	99000	99010	99000	6	0
emst-peeklo	#1	0	0	10	0	6	0
stat-peekhi-nohint	#1	99000	99000	99010	99000	2	0
stat-peeklo-nohint	#2	0	0	10	0	2	0
nost-peekhi-nohint	#1	99000	99000	99010	99000	2	0
nost-peeklo-nohint	#2	0	0	10	0	2	0
emst-peekhi-nohint	#3	99000	99000	99010	99000	2	0
emst-peeklo-nohint	#3	0	0	10	0	2	0

Oracle 10.2.0.5

SV\_R / Execution plan #1

(COOL-preview on Oracle 10.2.0.5.0)

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	TABLE ACCESS BY INDEX ROWID	XSV_RCWD.F0001.IOVS
3	NESTED LOOPS	
4	INDEX FULL SCAN	XSV_RCWD.F0001.CHANNELS_PK
5	SORT AGGREGATE	
6	FIRST ROW	
7	INDEX RANGE SCAN (MIN/MAX)	XSV_RCWD.F0001.IOVS.CSU_3INDX
8	INDEX RANGE SCAN	XSV_RCWD.F0001.IOVS.CSU_3INDX
9	SORT AGGREGATE	
10	FIRST ROW	
11	INDEX RANGE SCAN (MIN/MAX)	XSV_RCWD.F0001.IOVS.CSU_3INDX

```
4 - filter(COALESCE(,"since3s")<="until3")
7 - access("COOL_J1"."CHANNEL_ID"=:B1 AND "COOL_I1"."IOV_SINCE"<="since1")
8 - access("COOL_J3"."CHANNEL_ID"=:B1 AND "COOL_C2"."CHANNEL_ID" AND "COOL_J3"."IOV_SINCE">=COALESCE(,"since3s") AND "COOL_J3"."IOV_UNTIL">:"since3u" AND "COOL_I3"."IOV_SINCE"<="until3" AND "COOL_J3"."IOV_UNTIL" IS NOT NULL)
8 - filter("COOL_J3"."IOV_UNTIL">:"since3u")
11 - access("COOL_I1"."CHANNEL_ID"=:B1 AND "COOL_I1"."IOV_SINCE"<="since1")
```

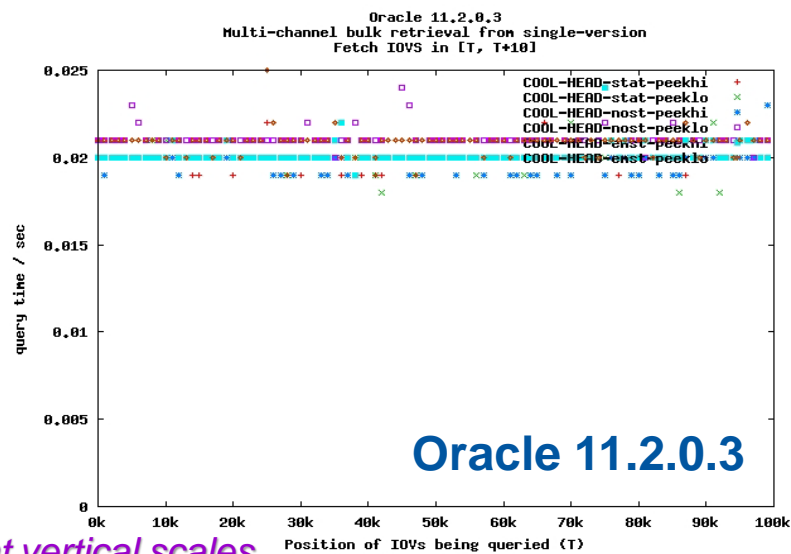
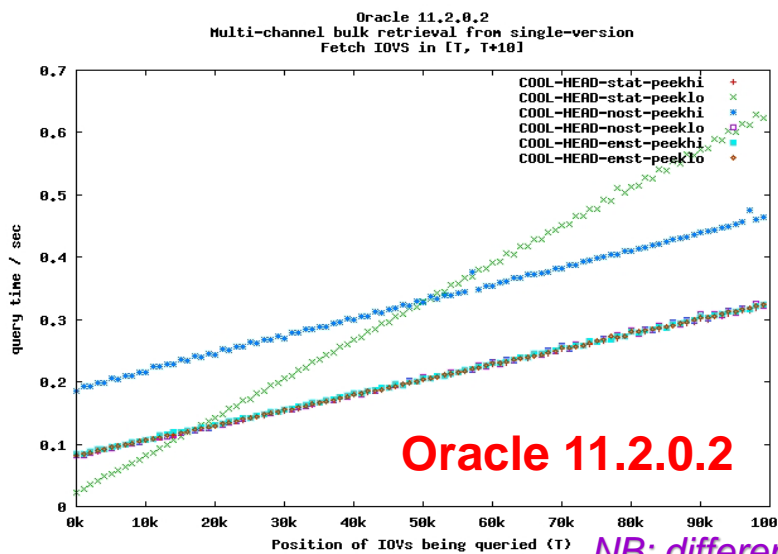
```
/*+
  BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.5')
  ALL_ROWS
  OUTLINE_LEAF(@"MAX1")
  OUTLINE_LEAF(@"MAIN")
  OUTLINE(@"MAX1")
  OUTLINE(@"MAIN")
  INDEX(@"MAIN" "COOL_C2"@"MAIN" ("XSV_RCWD.F0001.CHANNELS"."CHANNEL_ID"))
  INDEX(@"MAIN" "COOL_J3"@"MAIN" ("XSV_RCWD.F0001.IOVS"."CHANNEL_ID"
  "XSV_RCWD.F0001.IOVS"."IOV_SINCE" "XSV_RCWD.F0001.IOVS"."IOV_UNTIL"))
  LEADING(@"MAIN" "COOL_C2"@"MAIN" "COOL_J3"@"MAIN")
  USE_NL(@"MAIN" "COOL_J3"@"MAIN")
  PUSH_SUBQ(@"MAX1")
  INDEX(@"MAX1" "COOL_I1"@"MAX1" ("XSV_RCWD.F0001.IOVS"."CHANNEL_ID"
  "XSV_RCWD.F0001.IOVS"."IOV_SINCE" "XSV_RCWD.F0001.IOVS"."IOV_UNTIL"))
  END_OUTLINE_DATA
*/
```

Oracle 10.2.0.5



# COOL – Oracle performance tests (2/2)

- Was it worth all this effort? Yes, definitely!
  - Test infrastructure was developed to react to issues, but became a **tool for proactive QA (e.g. in software upgrades)**
    - Oracle server migration 10g to 11g: detecting a problem in Oracle 11.2.0.2 (and its resolution in 11.2.0.3) was a matter of days
    - Oracle server migration 11g to 12c: very fast validation checks



*NB: different vertical scales*

# Conclusions

- Testing and QA are top priorities in CORAL / COOL
  - Tests account for 30% of software (100k/300k lines of code), probably for even more of the development time and effort
  - Functional tests (from unit tests to complex integration tests), performance tests, static code analysis and other QA tools...
- Test-driven development pays off (*and is fun!*)
  - Including tests upfront is easier and more efficient than trying to add them a posteriori (and preventing is better than fixing)
- Designing and implementing tests is just as important as developing libraries and applications!