Conditions Database Project
COOL Status and Plans

Andrea Valassi

(CERN IT-ADC)

Outline

1. Status Report

- Project Overview
- Work in progress and plans in the short term

2. Feature requests

- Some common questions and requests

Project scope

- **Conditions data: non-event data that vary with time**
  - And may also exist in different versions
- **Data producers from both online and offline**
  - *Geometry, readout, slow control, alignment, calibration…*
- **Data used for event processing and more**
  - Detector experts, alignment/calibration, event reconstruction/analysis…
- **Scope of the common project?**
  - Common software and tools for time-varying and versioned data
  - Not the generic access to distributed relational data (RAL, 3D…)
  - Not the problems specific to one experiment or one data type
  - Scope has grown larger in time (from offline to online use case)

COOL – documentation and organization

- **Project web page – http://lcgapp.cern.ch/project/CondDB**
- **Mailing lists and archives**
  - Developers (high traffic) - http://simba2.cern.ch/archive/project-lcg-peb-conditionsdb-developers
  - General (low traffic) - http://simba2.cern.ch/archive/project-lcg-peb-conditionsdb
- **Savannah page – https://savannah.cern.ch/projects/cool**
  - User support (feature requests)
  - User bug reports
  - Internal task list – heavily used
- **Weekly phone meetings (every Monday at 4.30 pm)**
  - 5 to 10 people on average
  - Minutes - http://lcgapp.cern.ch/project/CondDB/snapshot/minutes_list.html
- **DOxygen documentation – http://lcgapp.cern.ch/doxygen**
  - Feedback? Only two (positive!) comments so far
  - First draft of user guide is based on the same DOxygen sources

COOL - people

- **Core development and testing team (~2-3 FTEs for common development)**
  - A. Valassi (IT/ADC) - core development, SCRAM config, release management
  - S. A. Schmidt (Atlas) - core development, PyCool
  - M. Clemencic (LHCb) – SQLite, CLOB, PyCool, testing, CMT config, LHCb integration
  - D. Front (IT/LCG) - performance validation and stress tests
- **Contrib packages and experiment integration**
  - N. Barros, J. Cook, R. Hawkings, A. Perus, S. Roe, RD Schaffer, T. Wenaus, F. Zema…
    - Naming only the people who show up most often at the Monday phone meetings
- **Expected new contributors**
  - U. Moosbrugger (Atlas) – core development (from September 1st)
  - S. Stonjek (Atlas) – distribution tests (presently working on Atlas/3D)
- **Collaboration with other LCG projects and teams**
  - POOL (relational abstraction layer (RAL), infrastructure (SCRAM)
  - 3D and IT-ADC-DP: database deployment, distribution, Oracle consultancy
  - SPI: infrastructure (CVS, tools, AFS…)
  - SEAL: plugins, int64, Time, …
- **There is still room for new people and contributions!...**
  - Contrib packages may later be integrated in the COOL release (e.g. PyCool)

Scope of the "common" project

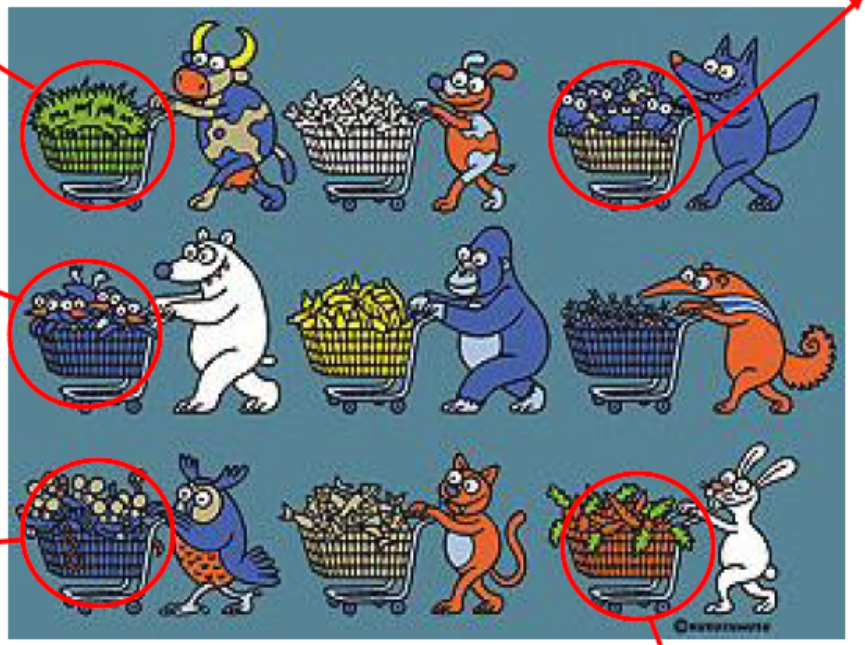
About the "common" project…

- Is there anything *in common* between what you want?

Reading the XML BLOB containing the LHCb calibration data valid for the event processed

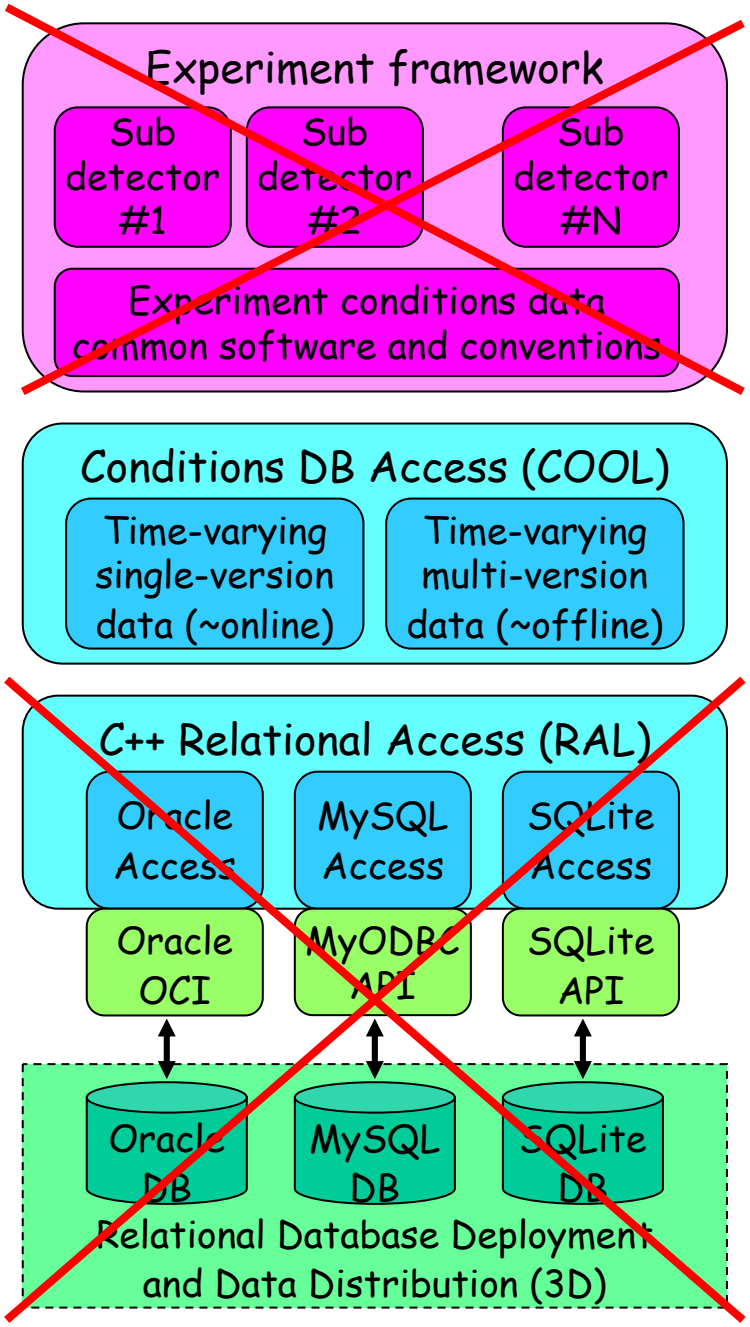
Retrieving the POOL alignment object for the run processed

Registering that the CMS detector geometry in a set of Oracle tables is valid for 2008 and 2009



Storing Atlas high voltages from PVSS into MySQL whenever the values change (every few seconds)

Reading Alice alignment from the ROOT files for the run processed



Scope of the "common" project

NOT the problems specific to one experiment or one data type *(too specific: handled by each experiment)*

Common software tools for time-varying (and optionally versioned) data: a component with a well-defined task

(Note: COOL presently has only a relational implementation because this is what was asked for by the experiments, but non-relational implementations of the same API are possible)

NOT the generic C++ access to relational data *(too generic: handled by RAL)*

NOT the generic deployment and distribution of relational data *(too generic: handled by 3D and IT-ADC)*

Original "common API"

- Designed to handle data "objects" that
  - Can be classified into *independent data items*
  - ***VARY WITH TIME***
  - May have different *versions* (for given time and data item)

This 3-D metadata model is still valid!

A CondDBObject has
- Metadata:
  - Data item identifier
  - Interval-of-validity [since, until]
  - Version information
- Data "payload":
  - Actual data variables (temperatures, calibration parameters…)
  - Separate fields or encoded as a BLOB

A "CondDBObject"

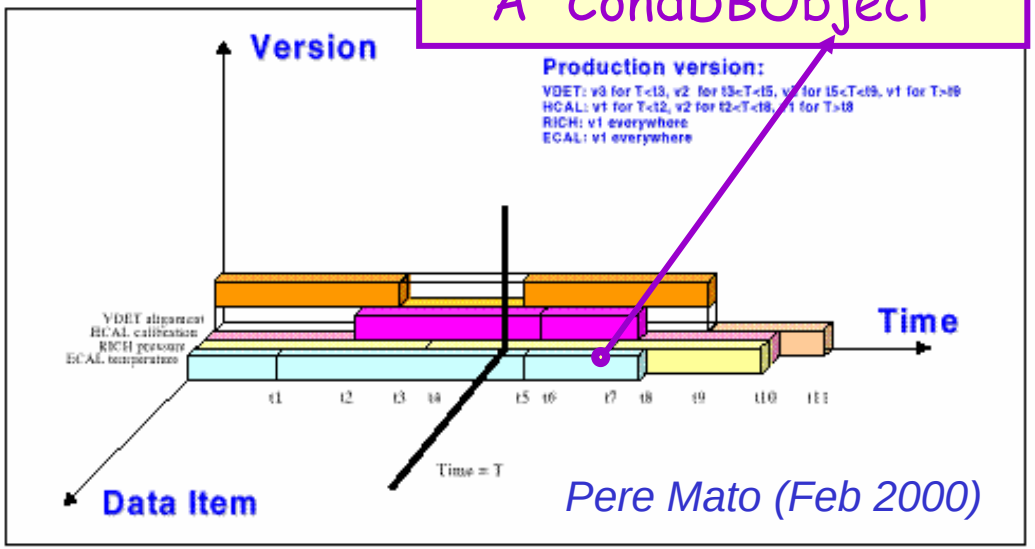


Figure 1 The three axes for identifying uniquely each data item in the condition database

- Main use case: retrieve data payload valid at given time in given tag
  - Inverse lookup (from temperature to time or version) not considered

COOL software overview

- **Maximize integration with and reuse of LCG software (SEAL/POOL)**
  - And merge the best ideas from the previous Oracle/MySQL implementations
- **Single implementation (same schema) for multiple back-ends using RAL**
  - User access encapsulated behind a C++ API (no direct SQL access)
  - Support for Oracle (main focus) as well as MySQL and SQLite
    - *Bulk operations and bind variables used whenever possible/appropriate*
  - Direct implementations or backend specific features may always come later
- **Modeling of Conditions database "objects" (e.g. a calibration set)**
  - System-managed common **"metadata"**
    - Data item id (e.g. calibration of module 1): "folder" (table) name + "channel" number
    - Interval of validity - IOV: since, until
    - Versioning information with handling of interval overlaps (at any given event time, there is never more than one object valid in a given "tagged version" of the data)
      - Two folder "types" for single-version (online) and multi-version (offline) objects
  - User-defined **"data payload"**
    - Support for simple C++ types modeled as pool::AttributeList
      - With a few additions to determine the storage precision (e.g. string: VARCHAR2 or CLOB?)
    - Different tables ("folders") for data items requiring different schemas

COOL – glossary

(does not cover all properties of each hierarchy entity)

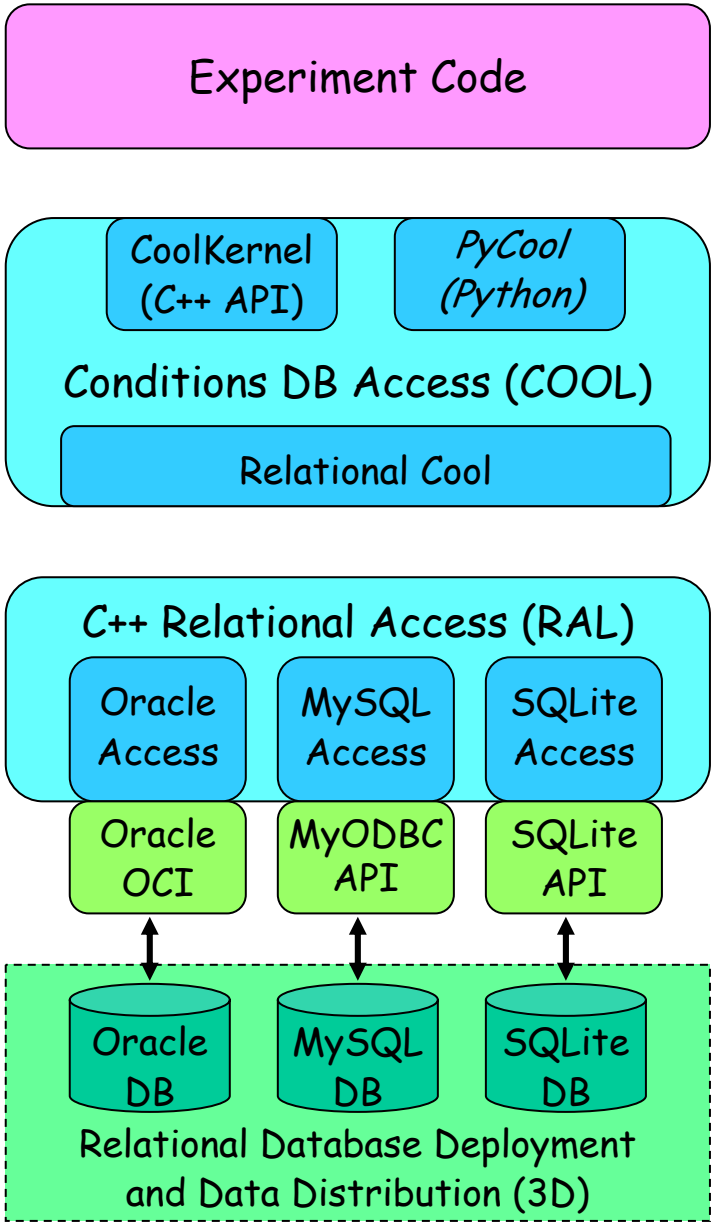
- **Database Service (id: plugin label)**
  - Entry point into the COOL software
  - Software component to manage databases
- **Database (global id: URL)**
  - Entry point into one collection of COOL data
    - of a specific relational technology: Oracle or MySQL
  - 1 database has 1 root folder set (i.e. 1 hierarchy of folder sets and folders)
- **Folder set (id in database: UNIX-like path name)**
  - Contains folders or other folder sets
- **Folder (id in database: UNIX-like path name)**
  - Contains channels
  - Objects in different folders may have different „schemas" (payload specifications)
- **Channel (id in folder: channel number)**
  - Contains objects
  - Objects in different channels in the same folder have the same payload specification
- ***Object (lookup within a channel: by validity time AND version number or tag name)***
  - *Data payload with a specific interval of validity (IOV)*
- **Tag (id in database: tag name)**
  - Object collection that contains only 1 (or 0) object in any channel at any validity time

COOL – software organization

- **Software infrastructure (SPI and CERN IT)**
  - CVS service for LCG, viewcvs, LXR, dOxygen
  - Platforms: SLC3 (gcc323 +dbg), Win32 (vc71_dbg)
  - SCRAM build system used internally (evaluate CMT to speed up Windows build)

- **Six packages (as of COOL_1_2_3)**
  - CoolKernel (public API)
    - Dependency on SealBase (exceptions, Time, int64) and POOL AttributeList
    - Non-relational interface (keep door open for non-relational implementations)
  - RelationalCool (implementation plugin using RAL – loaded on demand)
    - Relational database service is a loadable SEAL service
    - Runtime loading on demand of RAL plugins for Oracle or MySQL
    - Extensive unit test suite (may also provide additional user examples)
  - CoolApplication (link library – database service factory for standalone use)
    - Optional: SEAL applications can load the RelationalCool plugin directly
  - Examples (user examples and use cases)
  - Utilities (optional tools for performance testing)
  - PyCool (python bindings for interactivity and scripting)

COOL Dependencies



POOL packages

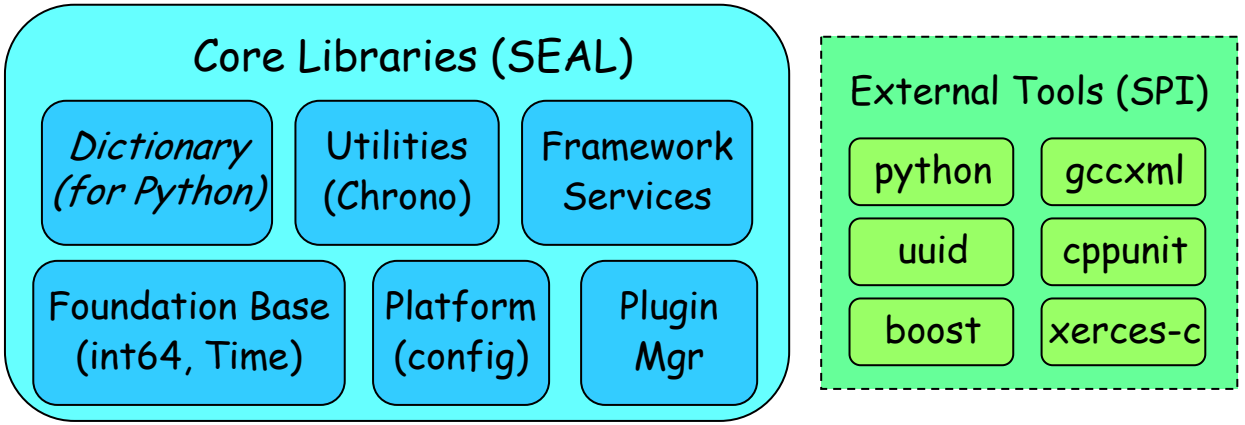
- AttributeList, POOLCore, RelationalAccess; Oracle/ODBC/SQLiteAccess. AuthenticationService

SEAL packages

- Foundation (SealPlatform, SealBase, SealUtil, PluginMgr, PluginChecker), Framework (SealKernel, SealServices), Dictionary (Reflection, *ReflectionBuilder, DictionaryGenerator*), *Scripting (PyLCGDict2)*

External tools

- sockets, pcre, uuid, boost, cppunit, xerces-c, *python, gccxml* (+ platform-dependent compilers)



COOL - evolution

- **Initial developments**
  - Oct-04: decision to develop new software merging best of 2 existing packages
  - *Nov-04: start of COOL development*
  - Dec-04: single-object, then bulk, read/write for single-version (online) mode
  - Feb-05: performance studies (useful feedback for RAL – no scrollable cursors)
  - Feb-05: multi-version (offline) mode with tagging
  - Mar-05: int64, SQL types (foresee CLOB), better API, bug fixes
- **Software releases**
  - COOL 1.0.0 (Apr-05): first public release
  - COOL 1.0.1 (Apr-05): bug fixes
  - COOL 1.0.2 (May-05): listChannels, listTags, bug fixes, new CVS
  - COOL 1.1.0 (May-05): repackage, basic multi-channel bulk insertion
  - COOL 1.2.0 (Jun-05): IFolderSet, setDescription, seal::Time workaround, performance improvements, AuthenticationService, bug fixes
  - COOL 1.2.1 (Jul-05): untag/retag, SQLite, Examples, Oracle privilege management, bug fixes in POOL (pthread lock, #open cursors)…
  - COOL 1.2.2 (Jul-05): bug fixes in SEAL
  - COOL 1.2.3 (Aug-05): PyCool, multi-channel bulk retrieval, CLOB, user guide, get software version, get COOL table names, get tag scope, bug fixes…

COOL Milestones for 2005

Conditions Database Project Milestones for 2005
Source: AA Plans for LCG Phase 2 (Draft 5, July 2005)

ID	Delivery Date	Description	
CDB01	30/04/2005	First COOL production release	OK
CDB02	*31/07/2005*	*Define reference performance requirements for COOL software and COOL service (ATLAS online exists)*	~ OK – High priority
CDB03	*31/07/2005*	*COOL tier 0 deployment starts for ATLAS*	OK
CDB04	*31/07/2005*	*Complete support for SQLight backend*	OK
CDB05	*31/08/2005*	*User examples and basic user guide*	OK
CDB06	*31/08/2005*	*Support for large character objects (CLOBS)*	OK
CDB07	*30/09/2005*	*COOL software review*	NOT YET
CDB08	30/09/2005	COOL passes all tier 0 validation tests	NOT YET – High priority
CDB09	*30/09/2005*	*COOL integration into ATLAS and LHCb offline framework finished*	NOT YET
CDB10	*30/11/2005*	*COOL release based on new RAL release*	NOT YET
CDB11	31/12/2005	API and command line tools for data extraction and cross-population between COOL databases	NOT YET – High priority

COOL – work in progress and short-term plans (1.3.0)

- **Performance validation for realistic experiment workload**
  - Service issues: assess whether database server is properly sized
  - Software issues: identify software bottlenecks (see details in next slide)
    - Also involves addition of a CHANNELS table for each folder
  - *Definition of "realistic" experiment workload still not entirely complete*
    - *Only need a clearly defined first guess, not the exact final LHC number ☺*
- **Data extraction and replication tools and tests**
  - At the COOL API level: need API to selectively clone data slices
    - Also start the design of a more distributed data model for a COOL "database"
  - At the back-end level: tests in the context of the 3D project (replicate full databases)
- **Improved API for CLOB support**
- **Improved Examples and user documentation**
  - Then organise a software review
- ***Integration into experiments' computing models and software***
  - *Milestone for the experiments, but would give useful feedback to COOL team*

Have a look at the Savannah tasks page for more details

COOL performance studies

objectID	channelID	since	until	pressure	temperature

- **Data insertion**
  - Achieved rates > 1k rows/sec in a single channel (satisfactory for the moment)
  - Performance for data insertion affected by data retrieval
    - Interval overlap analysis performed in C++, no straightforward modeling for interval overlap constraints in SQL
    - Minor issue: only one row retrieved when bulk inserting a batch (~1k-10k) of rows
  - Known issue: bulk insertion slower if into many "channels" in the same table
    - *Planned solution: re-optimize when bulk delete/update operations supported in RAL*

- **Data retrieval**
  - Presently studying a use case requiring 20 MB/sec and 20k rows/sec rates
  - A few issues already solved (e.g. ban bidirectional database cursors)
  - Known issue: retrieval performance non-uniform
    - Query/Indexes not optimal for WHERE SINCE <= time AND time < UNTIL
    - *Planned solution: query only one variable, retrieve first row WHERE SINCE <= time*
  - Known issue: memory leak observed (still unclear if in COOL, RAL or SEAL)

COOL performance – user recommendations

- **Do NOT artificially inflate data granularity!**
  - All data that have the same IOV should be grouped together and stored in the same channel as a single "object": do not define more channels than you need!
    - This is the correct data model if the data intrinsically have the same IOV
    - More important: **failing to do so may cause a serious performance overhead**
    - *Rule of thumb: define as few channels as possible*
  - Translated into relational terms: do not store more independent table ***rows*** than you need or than your data model intrinsically possesses
    - Whether or not a channel with 100 parameters with the same IOV is represented by a table row with 100 **columns** or with a single large column is largely irrelevant instead
    - The data insertion/retrieval rate to/from the database in #rows/second is as relevant as the rate in kilobytes/second…

- **Example**
  - A detector has 100k tubes. The calibration of all tubes is done every day at midnight. One calibration parameter per tube is computed, but all tubes are always calibrated at the same time.
    - A single IOV should be used to describe all 100k tubes!
    - The payload may be stored as a single column (LOB or reference to external object)
      - or for instance it may be split in 100 columns, one for each sector containing 1k tubes

COOL status report - summary

- **Latest production release COOL_1_2_3**
  - Support for single and multi version time-varying data
    - Retrieval by data item (folder + channel), time point/range and tag
  - Development/testing team of 4 (~2-3 FTEs), new people joining
- **Work in progress and current priorities**
  - Performance optimization
  - Replication and slicing API and tools
- **Progress essentially on schedule so far**
  - Also managed to support a few new requirements on the way (e.g. PyCool)
- **Deployment in Atlas has started (accounts on Atlas RAC)**
  - *Waiting for feedback (on the software and on the deployment)!*

Outline

1. Status Report

- Project Overview
- Work in progress and plans in the short term

2. Feature requests

- Some common questions and requests

Some common feature requests

- **Improved data item addressing**
  - Identify channels by more than an integer id: e.g. chamber 3 in module 5
- **Tags across different folders**
  - E.g. associate tag "PRODUCTION" to "CALIB-V1" and "ALIGN-V5"
- **Payload queries**
  - Inverse queries: e.g. in which time range was the temperature above 35°?
  - Sometimes expressed as freedom to do arbitrary SQL queries on COOL tables
- **COOL as a configuration database**
  - Retrieval by version but not by time
  - Performance adequate for uploading data to the detector online in real time
- **Additional relational features**
  - E.g. freedom to define foreign key constraints using COOL tables
  - E.g. "support for relational triggers" in COOL

Example: TGC group schema

Redundant: this could be elsewhere!

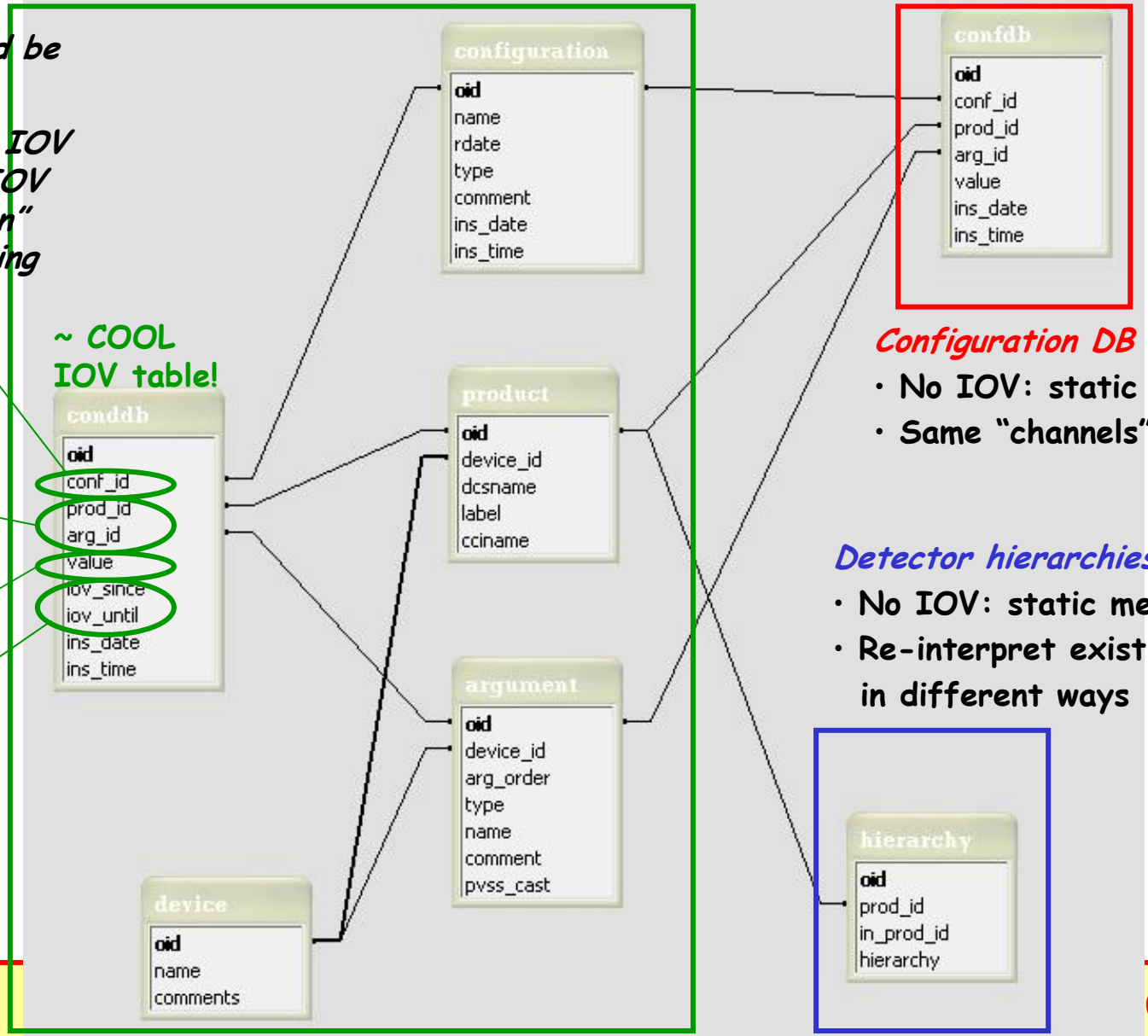
Every condition data IOV belongs to a larger IOV for the "configuration" used during data taking

Two-column Channel ID

Payload

IOV

~ COOL IOV table!



Configuration DB

- No IOV: static data!
- Same "channels" as CondDB

Detector hierarchies

- No IOV: static metadata!
- Re-interpret existing "channels" in different ways

Improved data item addressing

- **IMO: propose to add a "channels table" to COOL**
  - With the corresponding C++ API
  - See https://savannah.cern.ch/task/?func=detailitem&item_id=2392
  - Priority: medium-high (i.e. most likely early next year?)
- **Each channel may be identified by more than one parameter**
  - TGC case: device "Chamber", product "[Chamber] #3", argument "Temperature"
- **Each channel also gets a unique integer ID anyway**
  - Either user-defined or system-assigned
  - The channels table simply makes the static mapping between multi-column channel metadata and integer channel ID
    - Suggestion to TGC group: try to modify schema this way already
  - The integer ID continues to be used in the COOL IOV table
- **Suggestions are welcome**
  - Aim is to describe a general model, not a specific detector

Tags across different folders

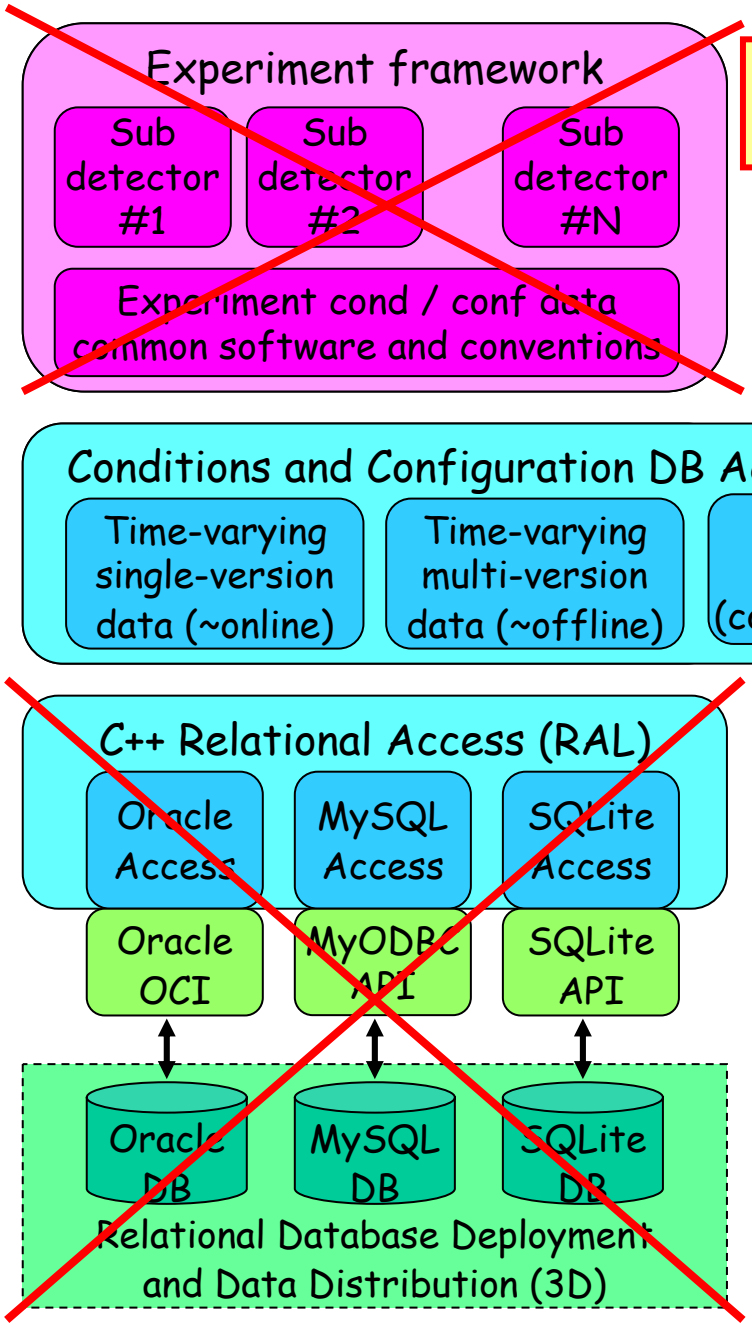
- **IMO: propose to implement it a la "HVS"**
  - Store persistently the association between "tag PRODUCTION in /Atlas" and "tag ALIGN-V1 in /Atlas/Alignment"
  - Alternative: propagate global tags as local tags associated to each IOVs in the leaf folders (a la CVS), actually easier…
  - See https://savannah.cern.ch/task/?func=detailitem&item_id=2393
  - Priority: medium (i.e. next year?)

- **Also needed: tag objects in individual channels**
  - See https://savannah.cern.ch/task/?func=detailitem&item_id=2390
  - Priority: medium-high (i.e. most likely early next year?)

Payload queries

- **IMO: understand precise deployment/computing model first**
  - *High probability that an arbitrary query would cause a performance issue*
    - Most likely need an index (for a query on one column)
    - Impossible to predict needed indexes and execution plans for multi-column complex queries that are not defined in advance…
    - COOL optimization is not even yet complete for retrieval by time/tag!
  - Do you need to do these queries in the database or can this be done in the experiment reconstruction framework (just like event analysis…)?

- **Options**
  - Provide simplest payload query functionality in the C++ software (with no performance guarantee), e.g. one-column payload range queries
    - Users responsible for appropriate deployment (eg OK on private SQLite file, not on production Oracle server used to write online data…)
  - Publish relational schema, allow arbitrary SQL queries
  - IMO: first option is better (cannot guarantee a "frozen" relational schema)
    - Priority: low (much lower than COOL optimization and development of slicing tools)



ConfigDB: extend project scope?

- **Handle time-varying and/or versioned data**
  - 1. Time-varying MV folders (e.g. calibration)
  - 2. Time-varying SV folders (e.g. DCS)
  - 3. Static MV folders (e.g. configuration)

Add 3rd type of data: configuration data?

e.g. TGC: upload into detector configuration for "commissioning" or "cosmic" run

- **Is this reasonable?**
  - Functionality: easy to add (within certain constraints)
    - Updating a configuration parameter is treated as inserting a new version of the parameter value
    - Advantage: may reuse some of the metadata model defined for conditions data (detector hierarchy)
  - Performance: computing model for configuration data is different from that of condition data
    - ***Retrieve data online in real time***
  - Performance: data granularity is different from that of condition data for offline reconstruction
    - How many independent table rows do you need to retrieve to configure your detector?
  - IMO: low priority, need to define precise requirements

Additional relational features

- **IMO: need precise requirements (specific task of a specific component)**
  - Project scope is NOT generic C++ relational access layer (it exists: RAL!)
  - If you cannot see/use COOL as a *component* to which you delegate the task it's meant to do (retrieval by time/tag), maybe you need your custom application?
  - Can you define a precise functionality you would need from COOL which is not already provided by RAL? Is this related to time-varying or versioned data?

- **A few specific points**
  - FK constraints for data item id
    - IMO, provide a COOL channels table and basic channel selection functionality
  - FK constraints for payload in referenced external tables
    - IMO, provide possibility to retrieve data from referenced tables (no payload query!)
  - Any other complex FK constraints: need precise definition of functionality!
  - Triggers: IMO, no support for these!

Common feature requests – summary IMO

- **Improved data item addressing – channels table (medium pr.)**
  - Identify channels by more than an integer id: e.g. chamber 3 in module 5
- **Tags across different folders – HVS (medium pr.)**
  - E.g. associate tag "PRODUCTION" to "CALIB-V1" and "ALIGN-V5"
- **Payload queries – simplest functionality (low pr.), performance warning**
  - Inverse queries: e.g. in which time range was the temperature above 35°?
  - Sometimes expressed as freedom to do arbitrary SQL queries on COOL tables
- **COOL as a configuration database – to be defined/understood**
  - Retrieval by version but not by time
  - Performance adequate for uploading data to the detector online in real time
- **Additional relational features – to be defined/understood**
  - E.g. freedom to define foreign key constraints using COOL tables – which ones?
  - E.g. "support for relational triggers" in COOL – NO