Databases in BABAR

Igor Gaponenko
SLAC National Accelerator Laboratory, USA

Super-B Computing R&D Workshop
Ferrara, Italy
March 11, 2010

What's in this talk

DISCLAMER: It's been almost 2 years since BABAR stopped data taking. Most people have left the Collaboration. My own memory is fading. Certain details may be incorrect. The general picture is still (hopefully) true.

- **Database applications in BABAR**
- **Integration with the Computer Model**
- **Evolution of databases**
  - Distributed Expansion
  - Better models, designs and implementations
  - Technology transition
- **Database distribution**
- **Experience with technologies (benefits, problems, lessons)**
- **Some ideas for Super-B:**
  - What can be inherited from BABAR
  - What can be done better (R&D suggestions)
  - Extreme ideas
- **Final thoughts**

DATABASE APPLICATIONS

Classes of databases

- **BABAR was the first HEP experiment in which almost every bit of data was in a database**
  - **1 PB in 2003** : "World record" for the biggest non-classified "hybrid" (non disk resident due to HPSS) database (see backup slides for more details)
  - No discussion of the Objectivity/DB based Event Store in this talk
- **Two main classes of databases (the terminology is arbitrary):**
  - "**internal**": to support the DAQ, production, reconstruction and analysis
  - "**environment**": to store an auxiliary information which was needed to interpret the detector events
- **"Internal" databases had limited use:**
  - All but one (OPR – to support Reprocessing in Padova) we located at SLAC
  - Limited number of users; no concurrency/scalability issues
  - Available mostly via Perl scripts or Web interfaces
- **"Environment" databases:**
  - Distributed to remote sites
- **More information on the databases on the next slides…**

Some "Internal" Databases

- **Ambient : EPICS records in ONLINE**
  - Objectivity/DB, later ROOT files, 0.5 TB compressed
- **ELOG : Electronic Log Book**
  - ORACLE, a fraction of 36 GB of the BBRO installation (one of a few ORACLE installations at SLAC)
- **IR2 : DAQ runs, shifts, num events, luminosity, beam parameters**
  - ORACLE, X of 36 GB (of BBRO)
- **OPR : processing cycles for runs**
  - Used software releases, and a lot of various metadata associated with that
  - ORACLE, Y of 36 GB (of BBRO)
  - Was also replicated in Padova
- **DQG : Data Quality Group data**
  - ORACLE, Z of 36 GB (of BBRO)
- **SPATIAL & TEMPORAL : Support "Rolling Calibrations"**
  - Accumulate intermediate/raw calibration data in space (from parallel processes processing events of one run) and in time (over a sequence of runs). The data were used to produce calibrations (rolling over runs)
  - Objectivity/DB, later ROOT files, x1 GB

"Environment" Databases

- **Config/DB: DAQ configuration records in ONLINE**
  - Objectivity/DB, later MySQL and ROOT files, 10 MB compressed
  - Updated in ONLINE, converted into ROOT and distributed elswhere
- **BBK: BABAR Book Keeping (a registry/catalog of event collections)**
  - ORACLE
  - Two installations: BBKR18 (21 GB) and BBKR24 (8GB)
  - Distributed around the Collaboration as SQL dumps to be loaded into MySQL
  - See more information in the backup slides area of the talk
- **CDB: The Conditions Database**
  - Objectivity/DB, later MySQL+ROOT, and ROOT files, 20+ GB compressed
  - Distributed around the Collaboration as ROOT files
  - Probably less than 100 installations (counting personal installations on laptops, etc.)
  - Most complicated database in its design, amount of code involved (nearly 10% of BABAR software packages implement or use its API) and use
  - See more details on the next slides

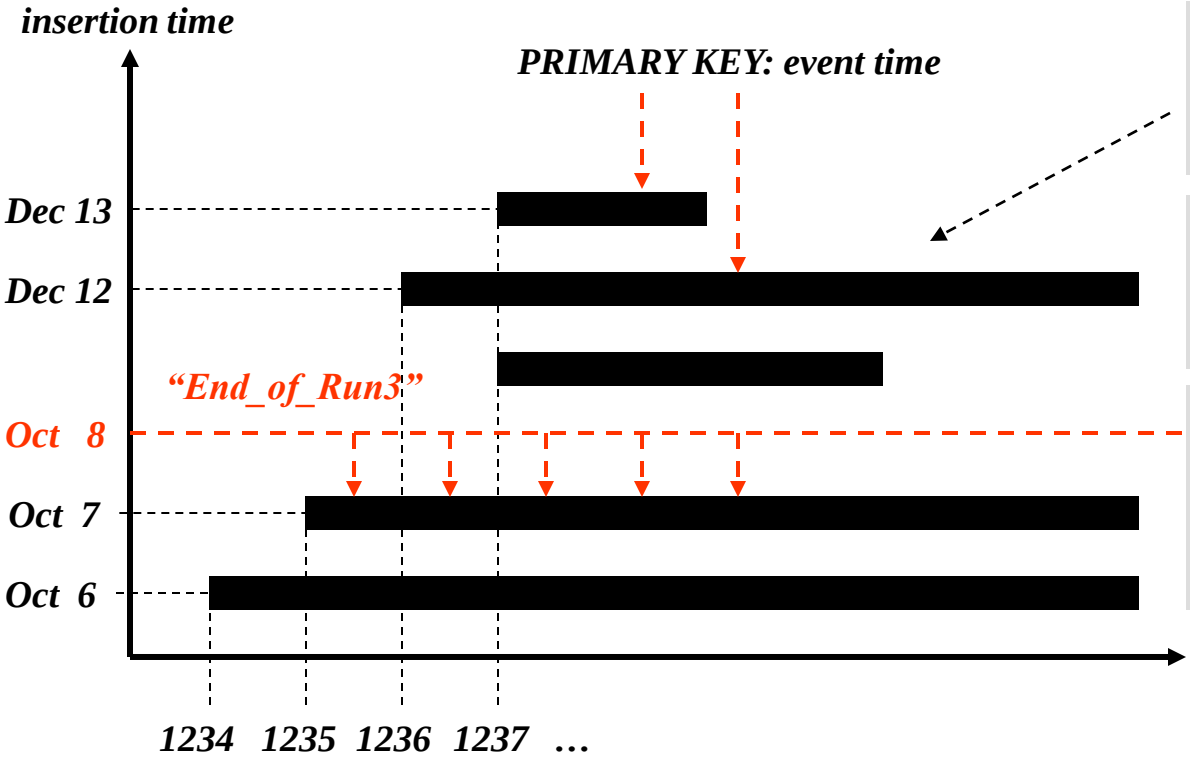
Conditions/DB (CDB)

CDB was designed and got its first implementation in 2002. Many thanks to Dave Brown (LBL) for the support and inspiration!

- **Technology neutral design**
  - Allows any implementations
- **Rich conceptual model targeting distributed computing models of contemporary HEP collaborations:**
  - An explicit ownership model for various persistent artifacts
  - Provisions for automated no-deep-merging data distribution (aka "data plug-ins")
- **Two layered namespace for conditions:**
  - "**Views**": Hierarchical namespace for conditions (like UNIX file systems)
  - "Physical Layer"
- **Geometric 2-D "object storage" space within each condition:**
  - Validity time dimension
  - Insertion time dimension
- **Partitioning of conditions:**
  - A controlled mechanism for sharing a single condition/calibration by parallel (possibly distributed) activities meant to update parts of the condition
  - Transparent to users
  - Easy to understand management model, interfaces and tools for administrators
- **StateID**
  - A small data structure to encapsulate a unique state of a database installation
  - "Increments" when major changes are made to the database
  - Was meant to be used as a mechanism for enforcing a correct use of the database

2-D space of conditions

For conditions, CDB introduces a simple geometric model in which conditions are containers providing 2-D space (INSERTION and VALIDITY timelines) for objects.



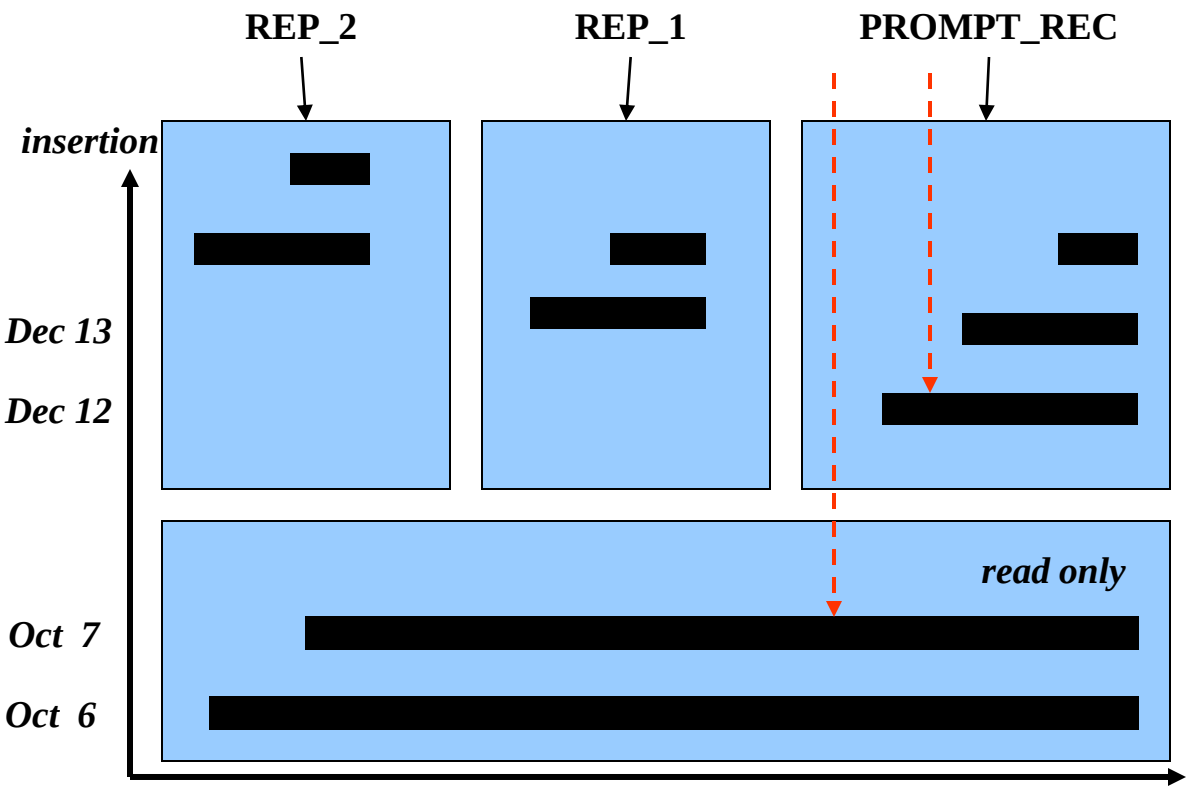
Bars are representing individual CONDITON OBJECTS

Each such object is valid during certain VALIDITY INTERVAL

Each object is assigned an INSERTION (creation) time when stored

A desired object is found by looking "from the top" at an intersection of the horizontal EVENT TIME and vertical REVISION.

Distributed Conditions & Partitions



PARTITIONS are sub-spaces in the 2-D space of certain conditions.

Not all conditions are partitioned. Only those ones to be modified by different activities at a time or at different locations.

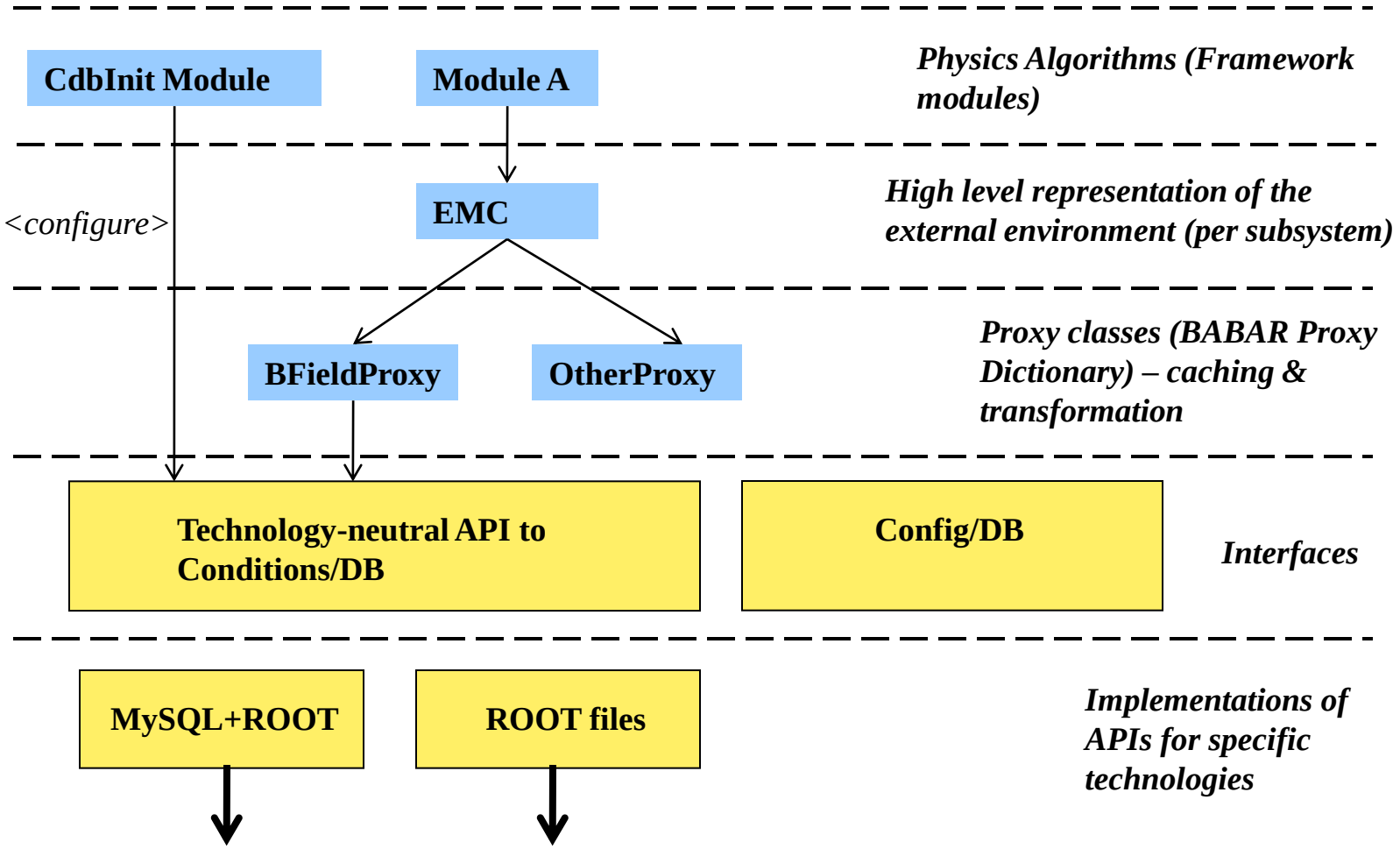
It's a foundation for **DISTRIBUTED** conditions.

Partitions are owned by **ORIGINS**. All storing operations for new objects are confined within the limits of the corresponding partition.

Meanwhile, partitions are transparent to the data access operations.

INTEGRATION WITH THE COMPUTER MODEL

Integration with Framework



Application Configuration

- Configure Framework applications to use a specific database using either:
  - An environment variable (default method)
  - Or, via special Framework modules
- The modules would also allow change a behavior of a database interfaces/implementations, or inquiry additional information:
  - Such as the current value of the CDB StateID

Event Store Integration

- **Using event timestamps or run number as explicit primary keys when accessing data from databases:**
  - The linkage is made by the Framework

- **[extra for CDB] Relying on an implicit "current state" of a particular Condition/DB installation used by an application:**
  - No formal check was made by the Framework that the database is in the "right" state
  - This makes applications extremely vulnerable to misconfiguration

- **Limited use of CDB StateID:**
  - Values of the ID are stacked in Event Headers by a special Framework module when an event gets created (DAQ or SP) and for each reprocessing cycle
  - The keys can be extracted from the headers and used for the "forensic" analysis
  - THOUGH, INITIAL IDEA WAS TO ENFORCE THE USE OF THE RIGHT DATABASE (STATE)
  - **Can be considered as a failed attempt to implement the Data Provenance "light"**

EVOLUTION

3 Vectors of the evolution

There was a strong "mixing" (correlation) between these directions.

- Expansion towards the distributed computing
- Better models, new designs and implementations
- Technology transition

Expansion…

Rather early switch to the distributed production/processing in BABAR

- **Stage I**: Initial setup (Spring 1999)
  - All (but a some) data in a single Objectivity/Db federation shared within SLAC by ONLINE and OFFLINE
- **Stage II**: Split into ONLINE and OFFLINE domains (Summer 1999)
  - Separate federations
  - Trivial data transfer from ONLINE into OFFLINE works
  - Still no external installations
- **Stage III**: Split within OFFLINE (Fall 1999 – 2000)
  - Separate federations for Prompt Reconstruction, Re-Reconstruction, Simulation Production and Analysis
  - Mounting issues with carrying data over federations
  - Very serious problems with the Condition/DB => merging database being in different states (AKA "CVS branches"): slow and error prone => lots of headaches
- **Stage IV**: Going distributed off-SLAC (probably in 2000 or 2001)
  - Off-SLAC Simulation Production, Analysis
- **Stage V**: Off-SLAC Reconstruction in Padova (probably 2002 or 2003)
  - This is a model of the distributed processing which we ended up in BABAR in Spring 2008
  - The only significant changes were in CM2, tools and persistent technologies for databases

CDB improvements

- **Not everything was ready for the expansion from Stage II to Stages III and on – the main road block was a lack of support for databases distribution at the design and implementation levels:**
  - Was especially difficult for the Condition/DB due to its use at all stages of the data production, processing and analysis
  - Poor performance of the initial implementation of Conditions/DB
- **Using CORBA servers as an intermediate cache**
  - Used as a domain-level cache to cache results of most expensive operations with the database
  - A significant speedup of the reprocessing farms
  - Details in the backup slides
- **Significant redesign of CDB made in 2001-2002:**
  - Rich conceptual model was introduced
  - An explicit support for a distributed multi-origin use
  - Sophisticated version control for timeline (revisions, views, etc.)
  - Hierarchical namespace for conditions
  - The idea of the **CDB State ID** – the first attempt to implement the Data Provenance "light"

Technology transition

- **Before 2002:** Original Computer Model
  - Everything in Objectivity/DB (except ORACLE for a few support databases)
  - Total amount of data 1 PB
- **2002-2003**: CM2
  - Event Store migration from Objectivity/DB ROOT (the topic covered in other talks)
  - No similar migration for other databases was planned at that time
- **2004**: Discussing scenarios for getting rid of Objectivity/DB in all databases; replace with MySQL and ROOT
- **2005**: Begin working on an implementation
- **2006**: Most but Condition/DB were migrated and deployed
- **2007**: The final migration for CDB
- **2008: End of data taking**
- **2010**: Still having one Objectivity/DB installation at SLAC for older releases

Event Store and Databases migrated separately; two technologies coexisted for a while

Technology choices

- **Migrate all non-ORACLE databases:**
  - Ambient, Configuration, Conditions, Temporal/Spatial
- **Stay within the Object-oriented paradigm at the persistent level:**
  - As it was in the original computer model of BABAR
- **Use ROOT/CINT as the new Data Definition Language:**
  - Great match and replacement for **Objectivity/DDL**
  - Simplified the migration
- **ROOT files:**
  - Where no update/read concurrency required
  - Ambient in ONLINE
  - All off-SLAC databases
- **Hybrid MySQL + serialized ROOT objects in BLOBs:**
  - Where the concurrency is an issue
  - Configuration/DB in ONLINE
  - Core Conditions/DB installations in ONLINE and OFFLINE at SLAC

CDB "Grand" Migration

- **Was most difficult and lengthy because of:**
  - Very complex and diverse user-defined schema; users were directly contributing into the schema by supplying DDL classes and C++ code which would use the classes to store and interpret subsystem specific applications; over 200 classes; no single design pattern; difficult to re-implement, test; had to write converters and unit tests; limited feedback/assistance from original developers (due to a shift of interest of the community towards LHC)
- **Additional complications:**
  - Not to interfere with the production code & data
  - Working for 1 year on a frozen snapshot of the whole BABAR code and then deploy it at once was NOT an option;
- **Migration strategy:**
  - **Migration-through-refactoring**
  - Re-implement the Condition/DB interfaces to be purely transient with technology-specific persistent-transient converters on the back end; still use Objectivity/DB as the back-end
  - Migrate user code in Framework and stand-alone applications to the **transient API**; test and deploy on the package-by-package basis
  - Finally, provide the MySQL & ROOT based API & schema implementation, convert data and switch.

Lessons learned

- **BABAR wouldn't have that much pain if:**
  - The underlying persistent technology was hidden behind interfaces
  - There was a clear persistent/transient separation
  - User-defined persistent classes had a simple straightforward design
    - In most occasions the schema complications were barely justified
    - Overdesign was not so uncommon as some people were overexcited with the "power and importance of OO")

- **Would it be better to introduce some sort of the Quality Control for the public schema design by a core group of experts?**

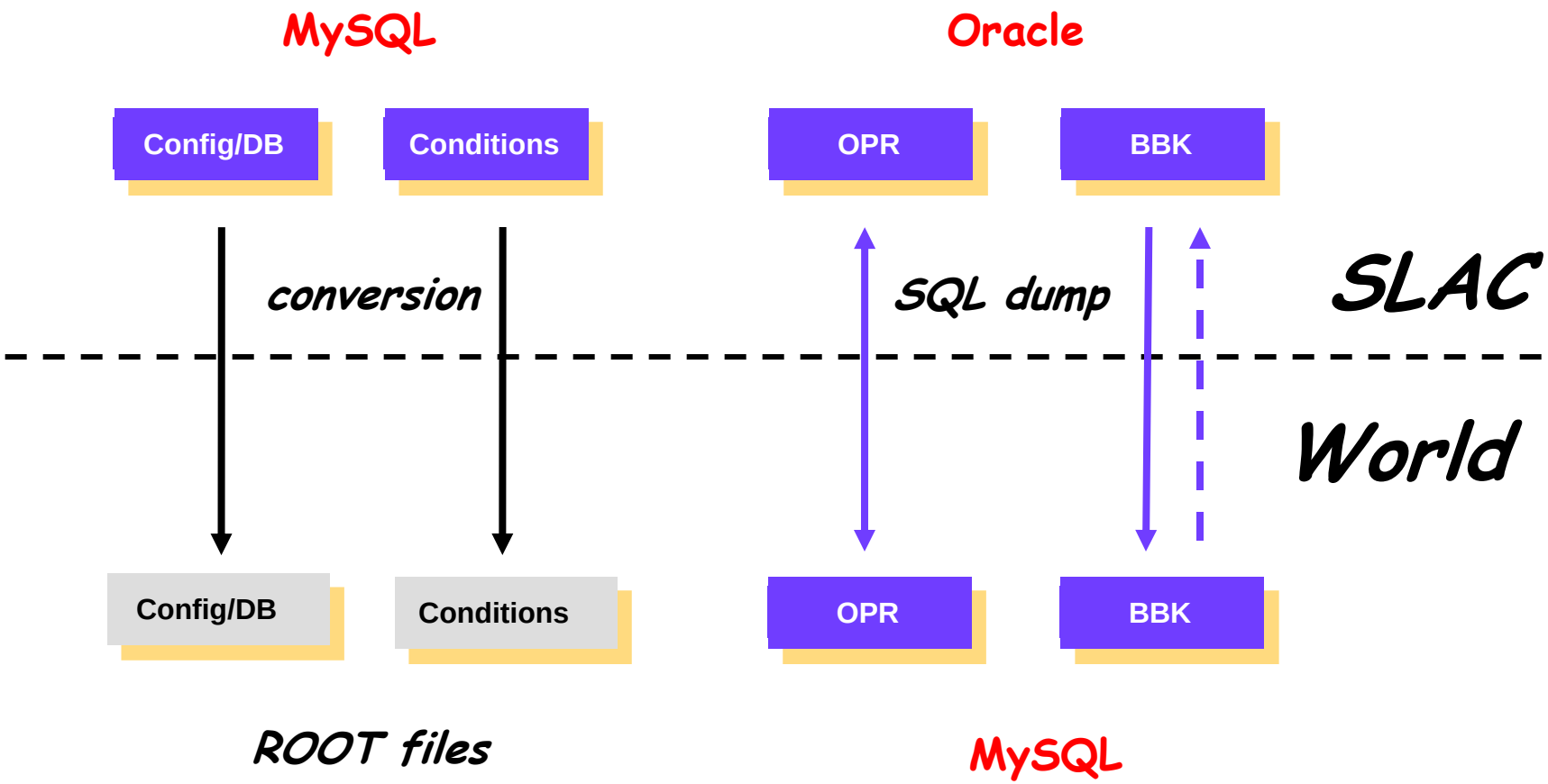
- **Or, if we take it to an extreme, better not to allow ordinary users to contribute into the schema at all?**

DATABASE DISTRIBUTION

Overview

- **Distributing databases to support:**
  - Simulation Production: 10+ sites
  - Event Reconstruction: 1 site (Padova)
  - Analysis: many x10 sites, laptops, etc.
- **4 Databases:**
  - Conditions
  - Configuration
  - OPR (runs and processing info)
  - BBK (event collections info)
- **Simple SLAC centric distribution model**
  - New data were created at SLAC (exceptions: local collections in BBK, and processing records for event reconstruction in Padova)
  - Read-only "snapshots" (ROOT) or SQL "dumps" (RDBMS) for using off-SLAC
  - Daily cron job to synchronize OPR database between SLAC and Padova
- **The PULL synchronization model:**
  - Produce updated snapshots at SLAC (frequency and methods varied for different databases and processing cycles)
  - Clients discover new snapshots and deploy locally (XROOTD, RDBMS)

SLAC-centric model

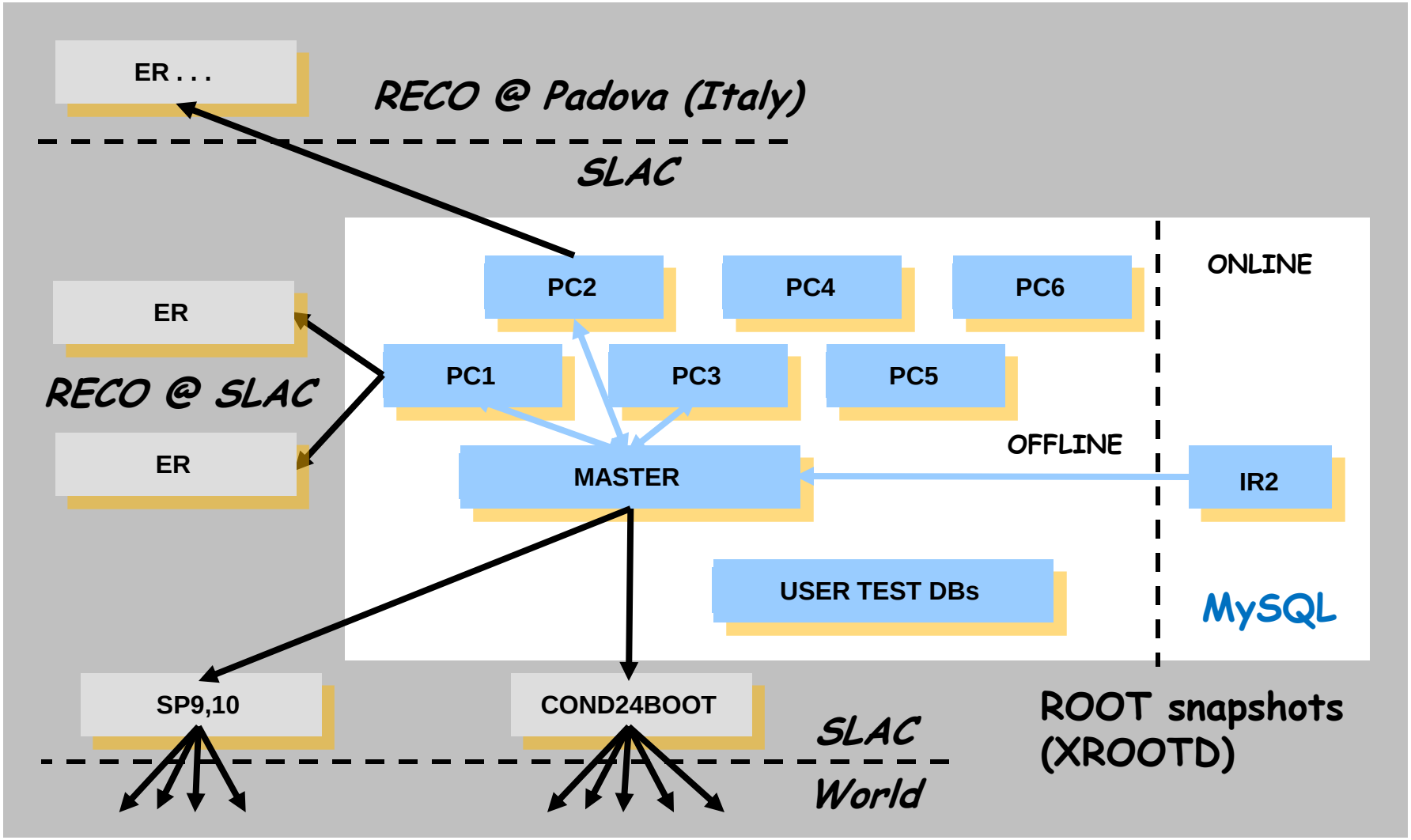


Details

- **Configuration database:**
  - **Usage**: simulation, reconstruction and analysis
  - **What's distributed**: A single ROOT file
  - **Distribution size**: ~1 MB

- **Conditions database:**
  - **Usage**: simulation, reconstruction and analysis
  - Most complicated distribution model (see next slide):
    - Home grown MySQL-to-MySQL synchronization within SLAC
    - Read-only ROOT snapshots for SLAC analysis and all off-SLAC uses
  - **What's distributed** : a few x100 ROOT files
  - **Distribution size** varied:
    - Full database (20 GB) for SP and analysis
    - Partial database (a few x1 GB) for Event Reconstruction in Padova

Distributed "Hybrid" CDB



More Details

- **OPR database:**
  - **Usage**: event reconstruction at SLAC and in Padova
  - **Distribution method**: A custom Perl script run on the daily basis (during an active reprocessing cycle) to keep SLAC and Padova databases in sync

- **BBK (Bookkeeping) database:**
  - **Usage**: simulation, reconstruction and analysis
  - **What's distributed**: compressed SQL dumps of the database to be installed/mirrored at remote sites
  - **Distribution size** varied:
    - Full uncompressed database for the latest reprocessing was 4 GB
    - Compressed size: 1.5 GB

Problems

- **Conditions/DB**
  - Unlike other databases it had an implicit secondary key (in CDB jargon: "**revision**") representing a version of an object
  - The ambiguity wasn't properly reflected in the Computer Model (including CM2), even though the database itself had sufficient provisions for that
  - Mainstream client application couldn't tell if a database instance were in the right state or not
- **What was at risk:**
  - Most applications run either at or off SLAC
- **Worst accident (May 2008 – July 2008):**
  - Simulation production at some remote sites were using old snapshots
  - Wrong results
  - Difficult to detect (CDB **StateID** didn't help)
  - 3-6 months to investigate it and fix consequences

Unfinished developments

- **Problem: A size of Conditions/DB snapshots**
  - Full snapshots were too big: ~20 GB
  - Non-negligible time to produce and publish a snapshot
  - Even more time needed to transfer and install at remote sites
- **Idea: Time slices for distributed Simulation Production**
  - Operated on the monthly basis (one month worth of background events)
  - 1/50th of the full CDB was actually needed for each month
- **Idea: Subsets of the database for distributed analysis**
  - Only a small fraction of all conditions/calibrations/constants needed
  - 1/10 - 1/100 reduction was possible
- **Benefits:**
  - Would be easier to run SP and analysis on GRID
- **Realization:**
  - Already supported by a conceptual design of CDB
  - Extra work on the data distribution tools and procedures needed
  - Extra work on core CDB implementation to prevent clients from using incomplete instances of the database

Started looking at this just before the end of the experiment

EXPERIENCE WITH TECHNOLOGIES (SUMMARY)

Objectivity/DB

For those in HEP who still cares about ODBMS.

- **General impressions (benefits of ODBMS):**
  - Great match for C++
  - Great performance for applications which were properly written
  - Infinitively extendable should non-standard indexes were needed
  - Worked great for most (but Event Store) applications
- **Unfortunately, too many shortcomings:**
  - No interactive language like SQL (to fix/debug problems when nothing else helps)
  - No authentication/authorization (was introduced too late for BABAR)
  - Limitations of the static schema (before the Active Schema API was introduced in 2003/2004)
  - Fundamental troubles each time new platforms were introduced (BABAR ran into this a number of times)
- **Cost & Maintenance:**
  - Pricy and not a mainstream technology => no market for trained DBAs => problems for the Collaboration members
- **Objectivity <-> BABAR relationships**
  - Mutual interest (BABAR served as a big advertisement for Objectivity)
  - We initiated (participated in) a development of three important features: **read-only** databases, **multi-federation** architecture and extended **security architecture**

MySQL (1)

- **Let's skip the well known benefits and focus on issues...**

- **Authentication model**
  - Password/host based authentication is very inconvenient for users
  - **Ugly workaround**: "*security through obscurity*" -> hidden accounts and passwords (in code or special files)
  - Things are getting more complicated as systems are getting more distributed
- **Authorization**
  - Is defined on SQL operations (SELECT, etc.) on databases and/or specific tables.
  - In most cases a desired authorization model of a domain level can't always be simply mapped onto the MySQL authorization model.
  - **Negative consequence**: a technology makes a **backpressure** onto a high level design.
  - **Solution**: developed an internal authorization layer within the application code (the above mentioned "*security through obscurity*")

MySQL (2)

- **Scalability**
  - **DISCLAIMER**: We didn't do any thorough testing. All said below is based on informal tests and real use in production
  - A single server won't scale beyond x100 of connections per an installation (observed on x2 CPU x4 core server, MySQL 5.0.* "Community Edition", using BABAR Conditions/DB clients)
  - **Solution**: MySQL master-slave mirroring/replication (unfortunately this doesn't have any dynamic load balancing and it's not trivial to set up and maintain)
- **Poor performance for certain critical operations:**
  - Operations with time series (intervals) in the Condition/DB (and alike)
  - **Reason**: MySQL is not able to recognize and cache similar (from our point of view) requests to avoid a costly query processing and execution
  - Consider this query:

    SELECT * FROM intervals **WHERE** begin >= event-time **AND** even-time < end
  - The results of the query won't be cached by MySQL even if all events belonged to the same interval
  - **Implication**: requests from (for example) x100 of processes working on different events of the same event collection (the same run) will be treated as different
  - **Possible solution**: application-domain external caching

MySQL (3)

- **MySQL replication is too restrictive to be used as a foundation for any serious database distribution in HEP**

- **Only three simple modes are supported (see next slide):**
  - Master-master
  - A ring of masters
  - A tree
- **No support for arbitrary graphs of interconnected servers**
- **Home-grown solution:**
  - Developed tools to dump databases/table into files => compress => transfer => decompress => upload
  - Benefiting from multi-core architectures by taking a multi-file snapshot of many tables simultaneously (x5 speedup on the x8 core machine)

  - **Issue #1**: not terribly efficient as the amount of data keeps growing over time
  - **Issue #2**: interferes with clients' activities (upload would destroy/replace tables)

ROOT I/O

- **Good match/replacement for Objectivity/DDL as a persistent data modeling language**
- **Simplified a transition of user defined schema during the technology migration**
- **Most likely to be supported for another 10+ years because of LHC:**
  - Which is good for Super-B should the experiment consider the same approach
- **Added benefits:**
  - Serialized objects can be "embedded" as byte strings into any structures (stored in RDBMS BLOBs, ROOT files, any arbitrary files, etc.)
  - The objects can be sent over network
  - Possible to implement specialized servers (for various optimization and caching purposes)
- **Issues:**
  - Schema stability problems when upgrading to newer versions of ROOT
  - (forward) Incompatibility of the data format between ROOT versions

IDEAS FOR SUPER-B

What can be inherited?

Very conservative strategy

- **Unfortunately very little because:**
  - None of the databases mentioned in this talk was designed/implemented as an independent or portable product
    - This has never been encouraged/discussed in BABAR
  - Very little documentation
  - Dissipation of the expertise with specific designs/implementations (and getting worse each year)
- **For things which can be reused the waiting time matters:**
  - Waiting for a couple more years won't make it easier
  - If anything has to be inherited directly – it must be done today, not tomorrow
- **What to consider:**
  - Experience, models, approaches (ROOT as a data modeling language?)
  - Probably Conditions/DB design, interfaces and some implementations (MySQL+ROOT); probably Config/DB
  - Other code (yet to be identified by contacting developers)

What can be done better?

Moderate strategy

- **R&D topic: "Distributed Databases"**
  - Better integration of databases into the Computer Model; provisions for consistency and data integrity in a distributed environment; Data Provenance "light"?
  - Design database applications (conceptual models, interfaces, protocols) as if they were to be used in a distributed environment, even if that's not a requirement now.
  - Investigate design scenarios to increase **mobility** of applications so that they would less depend on a specific database environment
  - Consider specialized servers/services (possibly Web based) in front of databases (see next slides)
  - Consider cooperative data caching on many core architectures (see next slides)

- **R&D topic: "Abstraction layers, Interfaces, Tools"**
  - Study various options for decoupling database applications from being directly dependant (by a design) on underlying persistent technologies; proper abstraction layers and interfaces.
  - Investigate a possibility of using standard portable data formats in the interfaces, such as XML, JSON

Extreme ideas (1)

Progressive strategy

This is inspired by a success of XROOTD

- **Investigate a possibility of developing a specialized server (service architecture) for the Conditions/DB (and alike):**
  - Basically, this is the same concept as "*..extra level of indirection solves all problems..*"; this is how most Web users interact with back-end databases behind Web servers; consider this as a model
  - Decouple client code from any technology-specific libraries (except those which are distributed along with the application); less requirements for an exec environment
  - Redefine the "**atomicity**" of database operations at the domain level, not at the level of a persistent technology
  - Implement authorization models which would suit specific database application domains better (not trying to fit into the low-level ones enforced by technologies);
  - Optimize operations with the "backend" databases; more options to implement better caching of results (see an example of caching interval requests in MySQL)
  - Correct blocking of clients during data distribution/synchronization on the server's backend; no service shutdown during the distributed data synchronization operations
  - Dynamic load balancing, cleints' redirection and more…

Extreme ideas (2)

Progressive strategy

- **An interesting variation of the idea from the previous page would be to use Web services as an indirection layer between client applications and databases:**
  - Should work for both reading and updating databases
  - Will increase the **mobility** of applications in a distributed environment
  - Leverage of certain interesting Web technologies, such as portable object serialization using XML and JSON, caching, etc.

- **The idea was inspired by the US/DOE/SBIR proposal:**
  - "Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases", Tech X Corporation
  - http://www.sc.doe.gov/sbir/awards_abstracts/sbirsttr/cycle25/phase2/087.htm

Extreme ideas (3)

Targeting an explosion of parallelism

- **Investigate a possibility of implementing a dynamic cooperative caching of the read-only data by a group of application processes run on a multi- or many-core system:**
  - Consider a scenario of x1000 processes run on a many-core system and processing in parallel different events of the same event collection; each process would probably need to bring the same data from some remote database server (x1000 similar requests to that server); what would happen to the server?
  - The processed could cooperate to cache (on a local disk or in memory) the data read from a remote database server either by delegating one of them as a caching server, or launching a dedicated server, or employing a distributed logic to use a shared disk as a local data cache
  - Will decrease the load onto database servers
  - Will decrease (due to caching) the service latency for applications in a distributed environment
  - Should be easy to do for read-only access to databases

Final thoughts

- **BABAR experience shows an interesting trend:**
  - A steadily diminished direct use of real (RDBMS, ODBMS) database systems for most (but internal or production support) applications
- **For a variety of reasons traditional server-based RDBMSs already do not scale (also are difficult to use) for the systems we have in HEP.**
- **The trend is most likely to continue due to:**
  - An explosion of parallelism (x1000 in 15 years according to Moor's Law)
  - An expansion of HEP systems into more distributed environment
- **Besides we don't seem to be using database systems for what they were designed. What we do instead in most cases is like calling functions:**
  - Store that object + parameters
  - Fetch that object + parameters
- **Are we seeing a diversification of ways we would handle databases for "internal" and "external" uses?**
- **Is there any need to maintain a very complicated and costly RDBMS infrastructure in HEP for the external uses?**
- **Should we stop using traditional RDBMS in some applications? Time to step back?**

References

- **Design documents and manuals at BaBar Databases Web Site:**
  - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/experts/designDocs.shtml
- **Talks:**
  - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/talks/Igor/Chep2004Talk316.pdf
  - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/talks/Igor/CdbApiOverview.pdf
  - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/talks/Igor/BaBarExperienceWithCDB/BaBarExperienceWithCDB.html
- **Articles:**
  - http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/Chep2004Paper316.pdf
- **Source in BaBar WebCVS:**
  - http://babar-hn.slac.stanford.edu:5090/cgi-bin/internal/cvsweb.cgi/
- **My personal talks (mostly about databases):**
  - http://www.slac.stanford.edu/~gapon/TALKS/

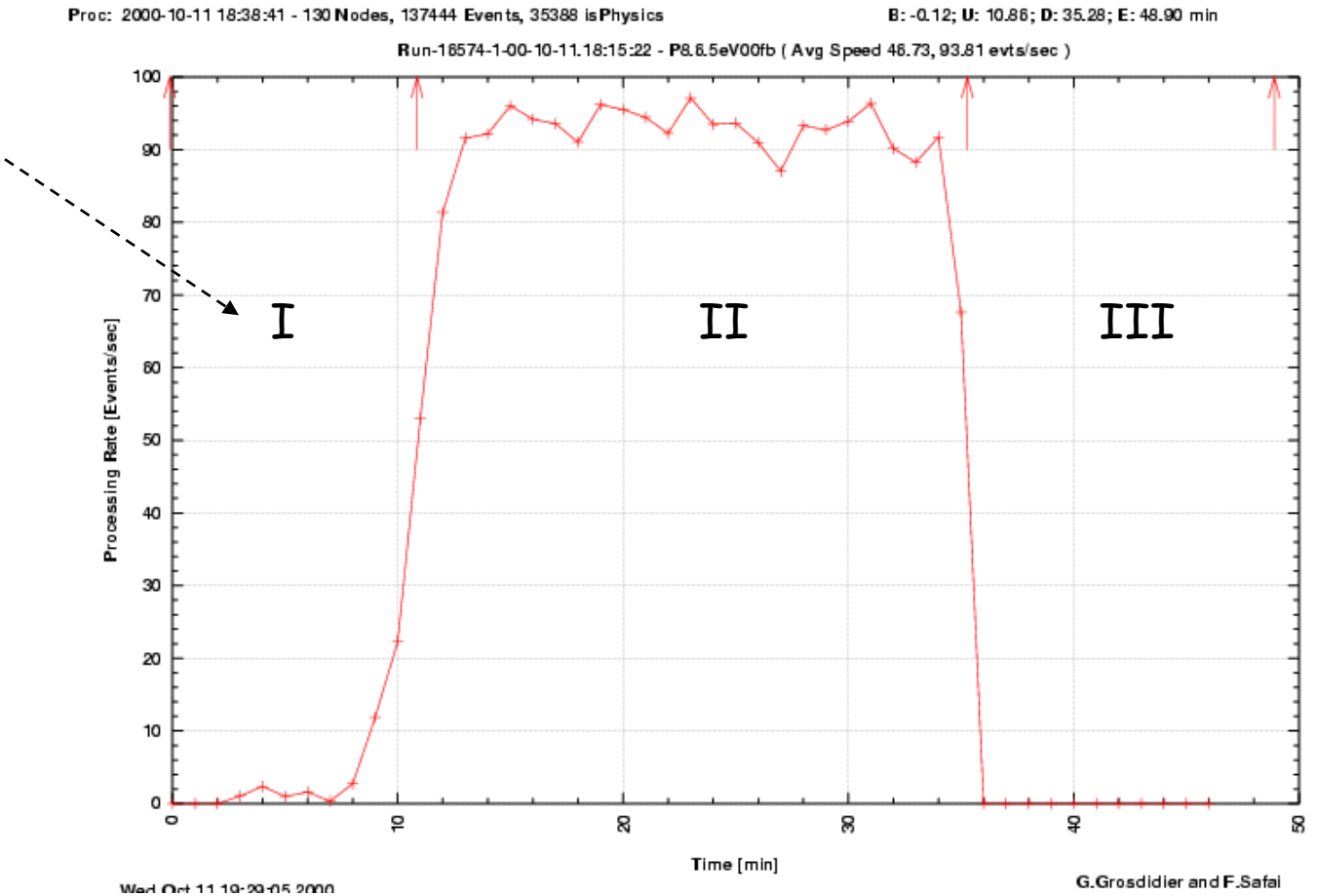
BACKUP MATERIAL

The Startup Time of Reco

OID Server

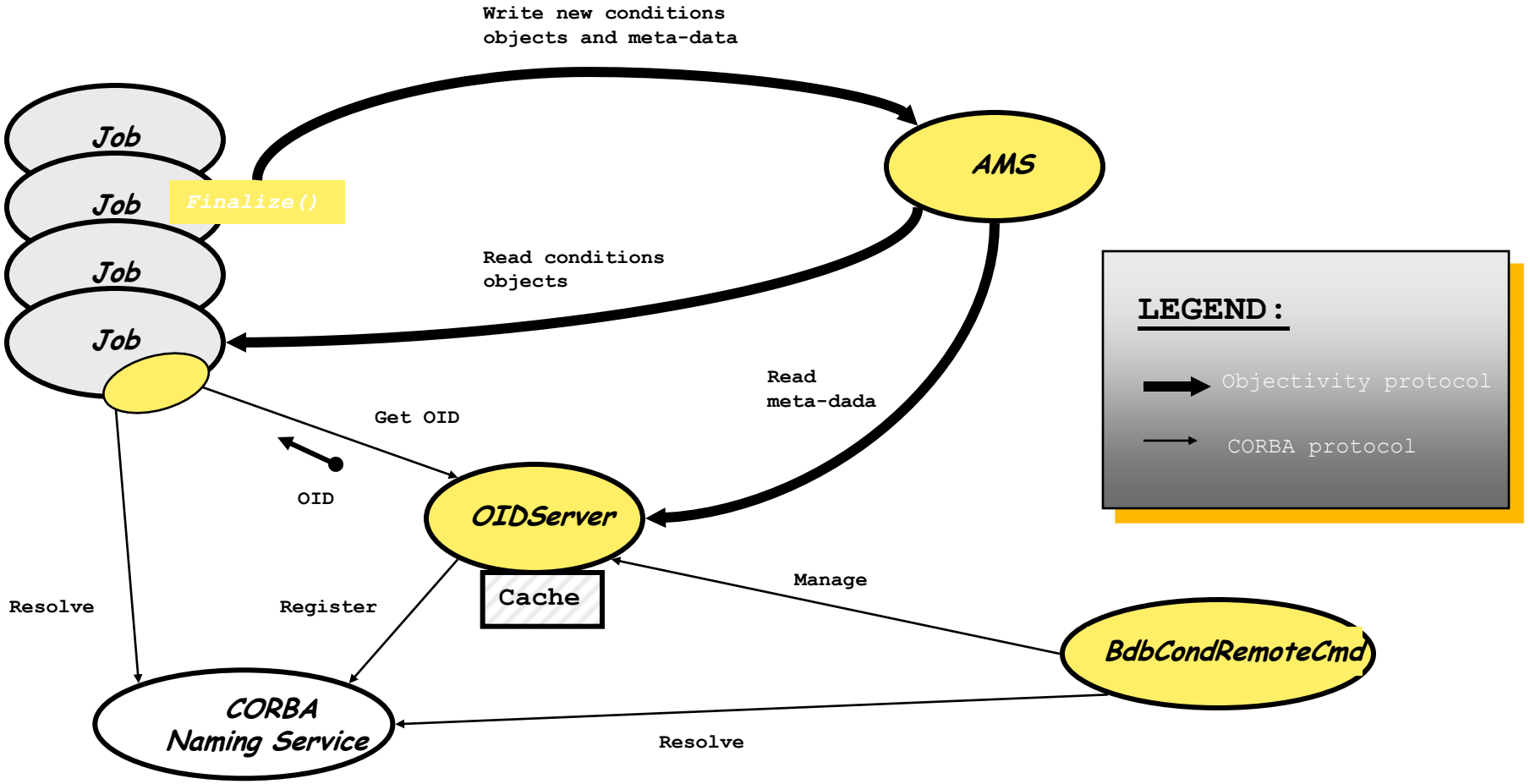
Problem Definition: *It takes too long for (150) jobs to "warm up".*

Here is a problem.



Interaction Graph

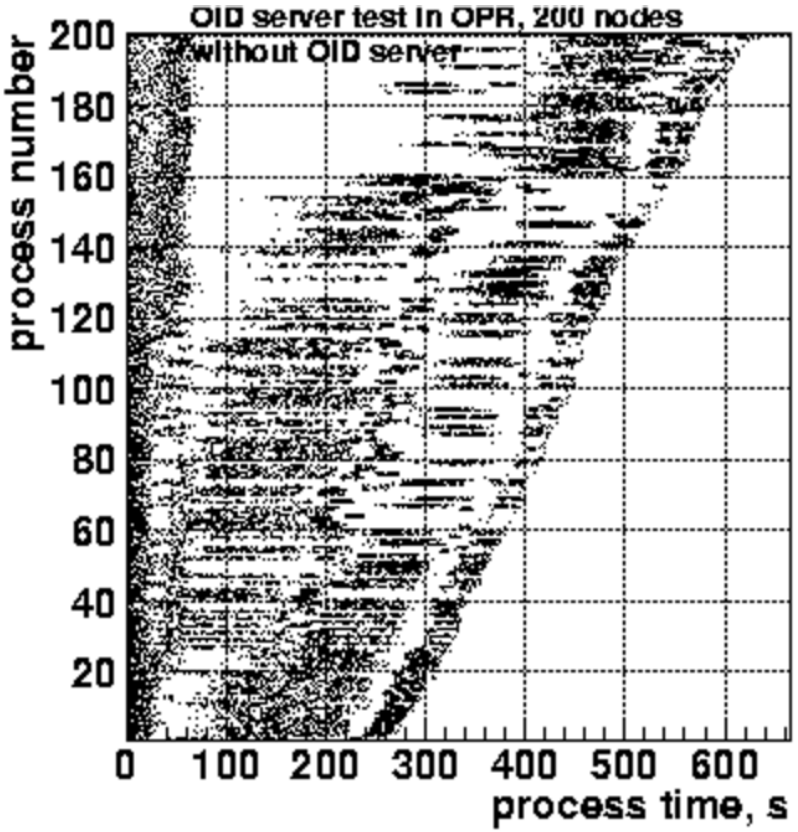
OID Server

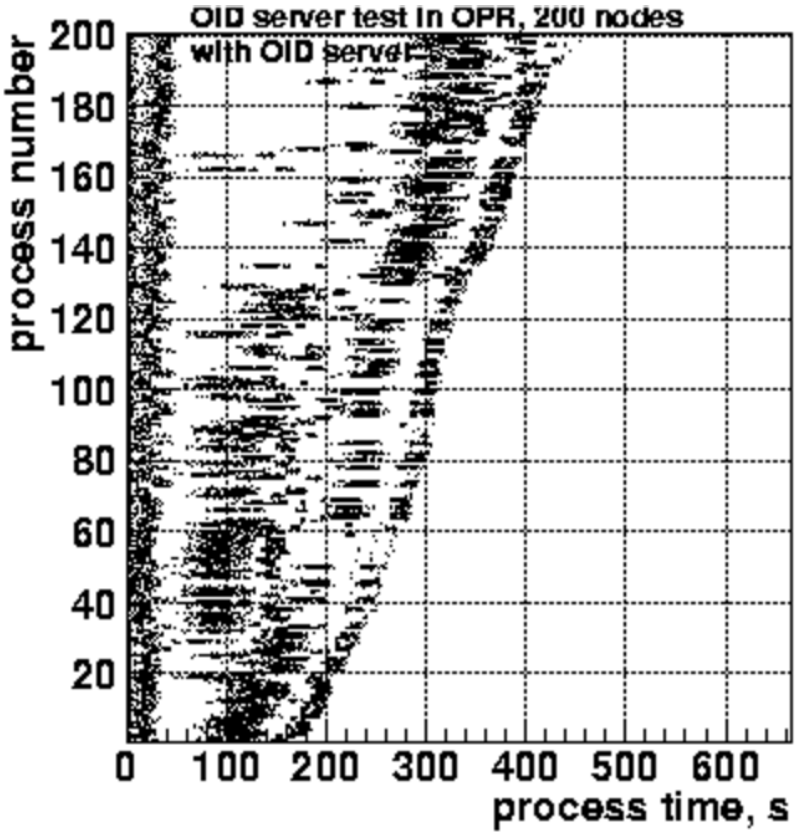


Performance: OID Server (I)

OID Server

Compare: *Conditions loading requests are shown without and with the OID Server.*

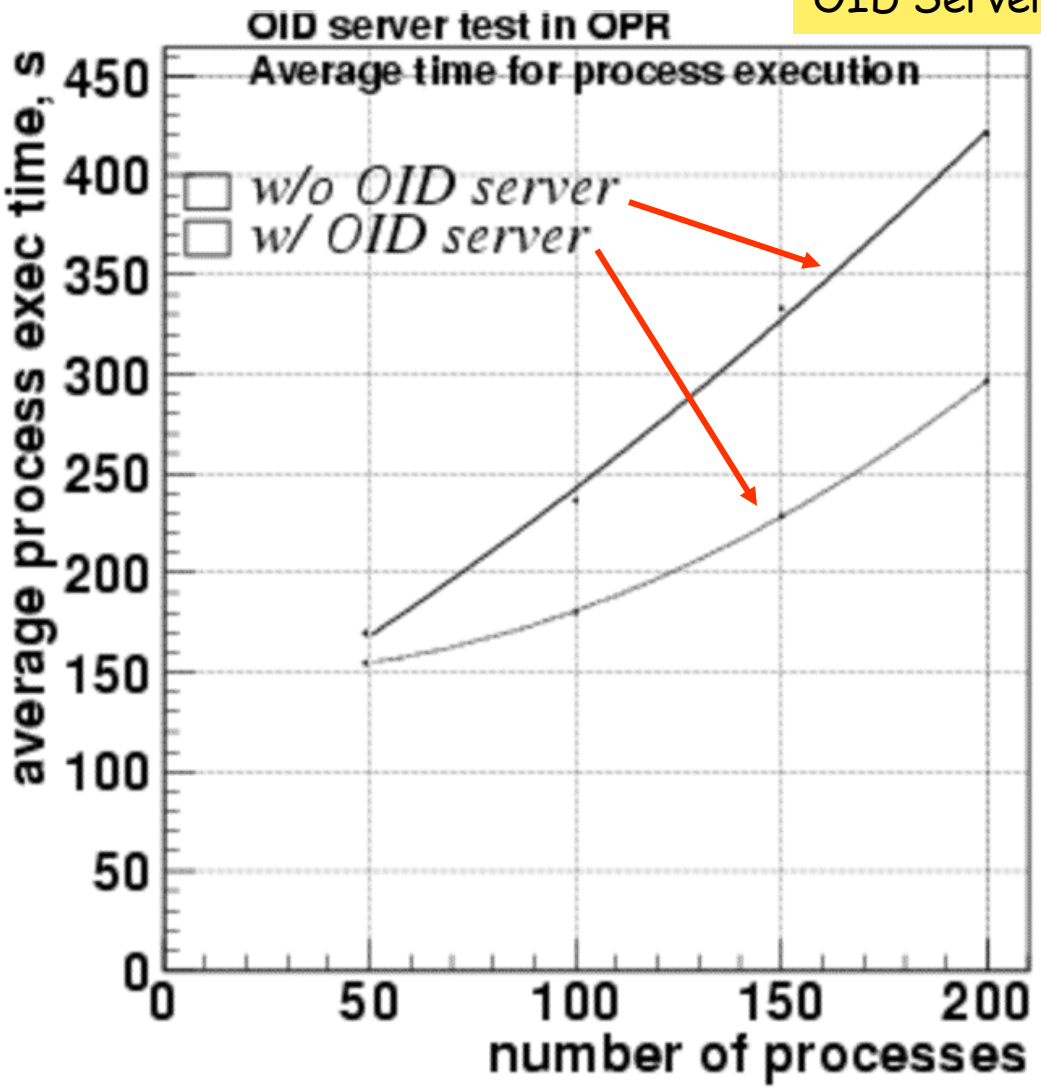




Performance: OID Server (II)

OID Server

Note: *The reduction of the average startup time for reconstruction jobs with the server is up to 30%.*



World Record 2003

In 2003 BABAR had the biggest non-classified database in the "hybrid" (non disk resident due to HPSS) category.

See:
http://www.objectivity.com/pages/press/htmlpress.asp?id=98

http://www2.slac.stanford.edu/tip/2004/may21/database.htm

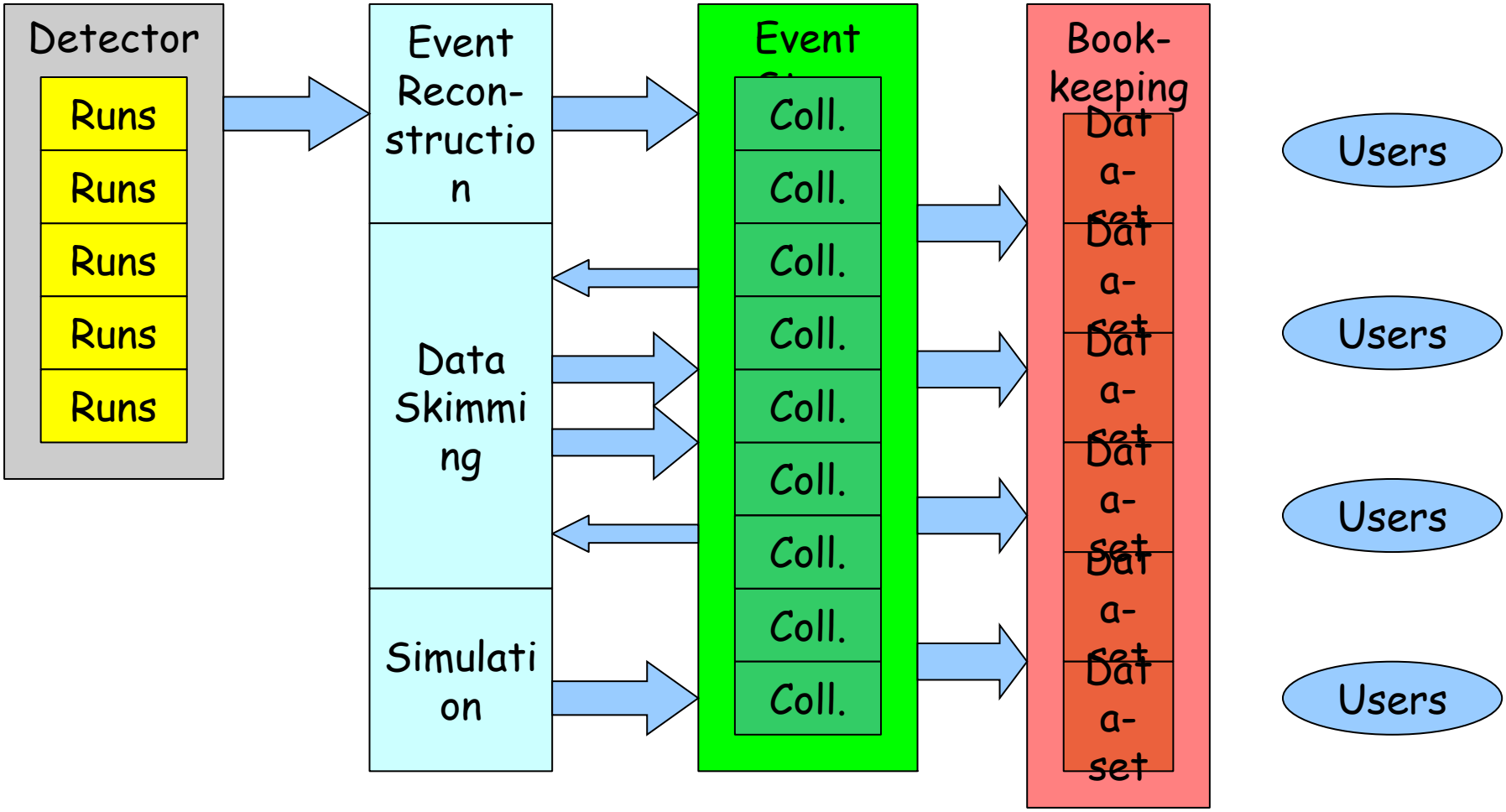
Overview of Bookkeeping

Slides provided by Douglas Smith

- CM2 event store is in ROOT files.
- Different event components can be placed in separate files (header, AOD, ESD).
- Related files of many events called "Collections", these are the unique elements of the event store.
- Collections are organized in lists, called "Datasets" for use, and there is a *n* to *m* relation between them.
- Also the relation between data run number and collection is kept in a *n* to *m* relation.

Cartoon of data management

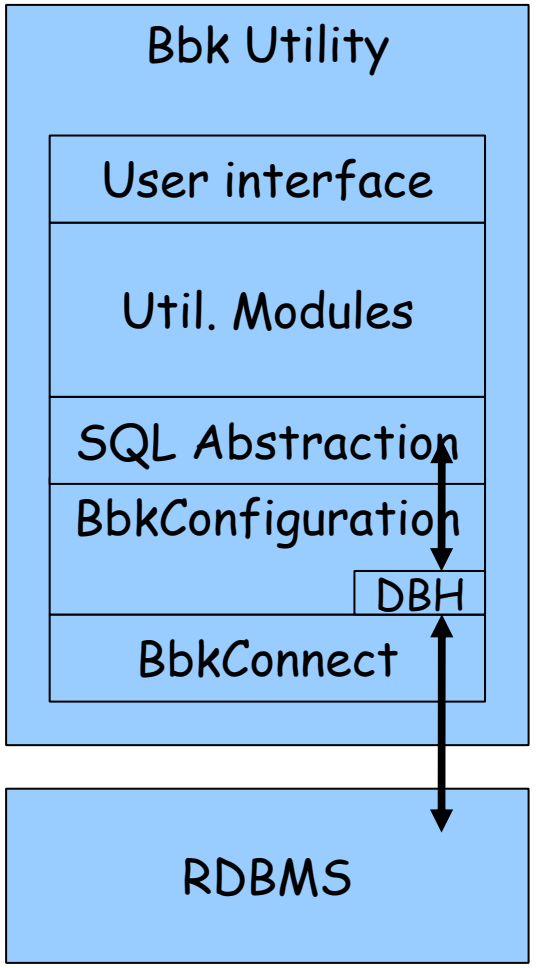
Slides provided by Douglas Smith



Design of a Bookkeeping utility

Slides provided by Douglas Smith

- Utilities built from modular design
  - All information kept in RDB
  - Connection module
  - Configuration module
  - SQL Abstraction module
  - Utility code modules
  - User interface



Multiple database design

Slides provided by Douglas Smith

- Utilities can be used to connect to any number of datasets.
  - For scaling to data size, new reprocessing go into new datasets:
    - First full reprocessing in CM2 with release 14 – bbkr14
    - Next full reprocessing with release 18 – bbkr18
    - Latest full reprocessing with release 24 – bbkr24
  - For scaling to distributed use, the databases are mirrored:
    - Updates to database done to master in SLAC.
    - Updates mirrored to remote sites.
- Default management system can control use of utility:
  - Analysis with release 18 gets bbkr18, release 24 gets bbkr24.
  - Analysis at remote site gets local mirrored database.

Size of current use

Slides provided by Douglas Smith

- Total event store is 2.73 PB with 5.25M files. Latest reprocessing for use 882TB with 1.54M files.
- The Bookkeeping system keeps track of this as 3.76M collections, 1.04M coll for latest reprocessing.
- The collections are organized into 240k datasets, 55k for the latest reprocessing.
- Size of databases is ~20 GB total, ~4GB for latest. Can be downloaded as compressed daily snapshots of ~1.5GB, 419MB for latest.
- Has scaled fine to meet the needs of the experiment.