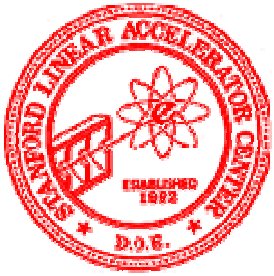
DBProxy:
The Database Communication Agent for HLT Configuration

SLAC ATLAS Forum
2008-08-20

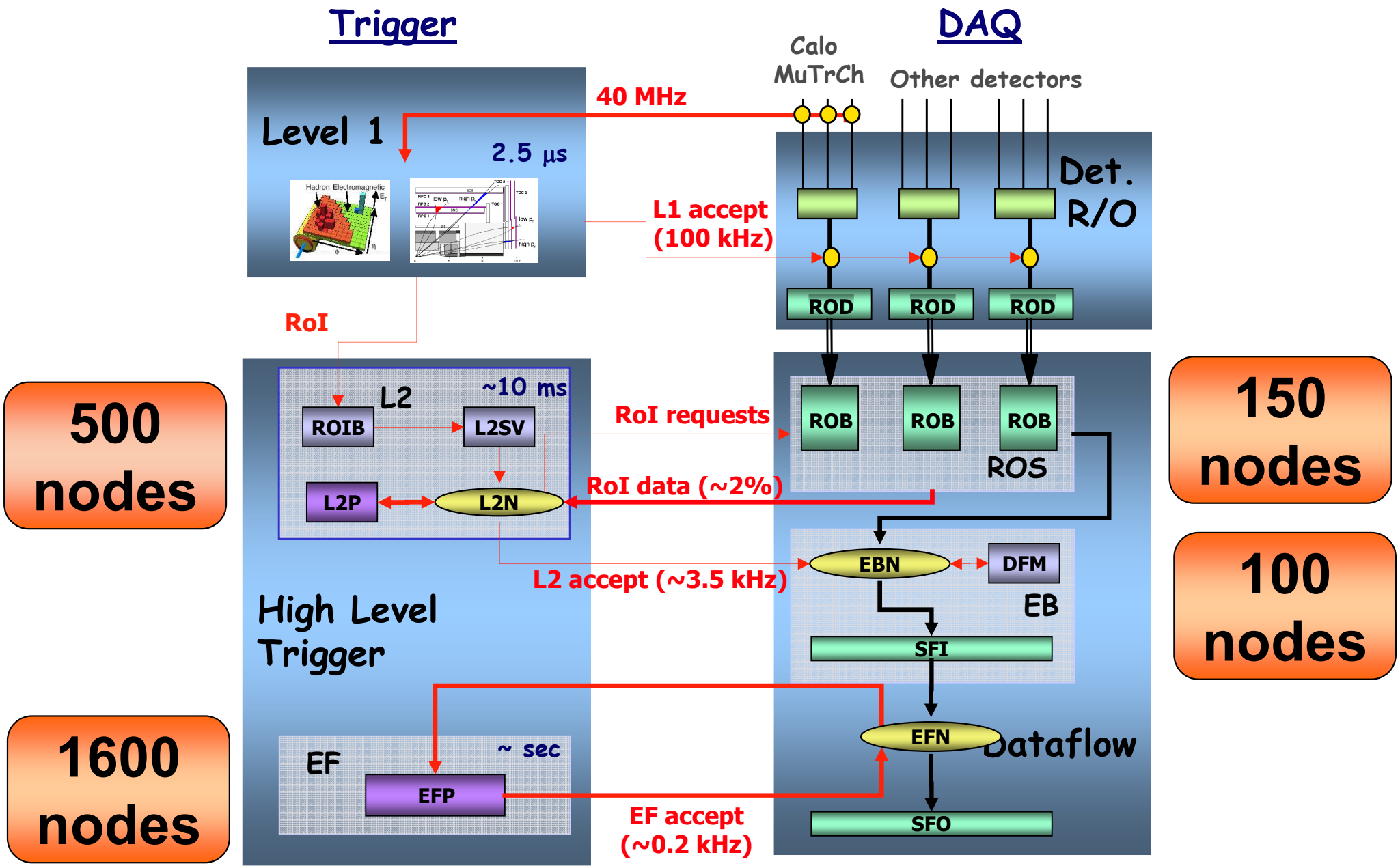
Andy Salnikov



Outline

- HLT farm
- HLT configuration problem
- Using database proxies for configuration
  - MySQL proxy
  - CORAL Server / CORAL proxy
  - MySQL-to-ORACLE bridge
- POOL files access

DAQ/HLT Architecture (2006)



HLT Farm @ Point1

- TDR assumptions (2003):
  - 500 LVL2 “processors”
  - 1600 EF “processors”
  - (assuming 8GHz clock speed)
  - 20+80 racks
- Current setup
  - 27 “XPU” racks
  - 31 nodes per rack
  - dual quad-core CPU
  - ~820 nodes, ~6500 processes
- Final setup
  - depends on first beam experience
  - farm will certainly grow

HLT Configuration

- Every HLT application (LVL2/EF) has to read configuration data to be able to process event data
  - Trigger algorithms, lines, and prescales
  - Geometry data, complete detector description
  - Conditions data: calibrations, reconstruction algorithms, etc.
- The configuration data comes from several database instances
  - ATLAS C++ code uses CORAL library for database access
  - MySQL, SQLite, ORACLE
  - production will use ORACLE exclusively
- Some configuration data come from POOL files (ROOT)
  - used by COOL to store condition database objects

Configuration Problem

- HLT configuration moves a lot of data
  - around 2000 nodes, up to 8 processes per node, tens to hundred MB of configuration data

$$2000 \times 8 \times 10MB = 160GB$$

(~30 minutes over 1Gbps connection)

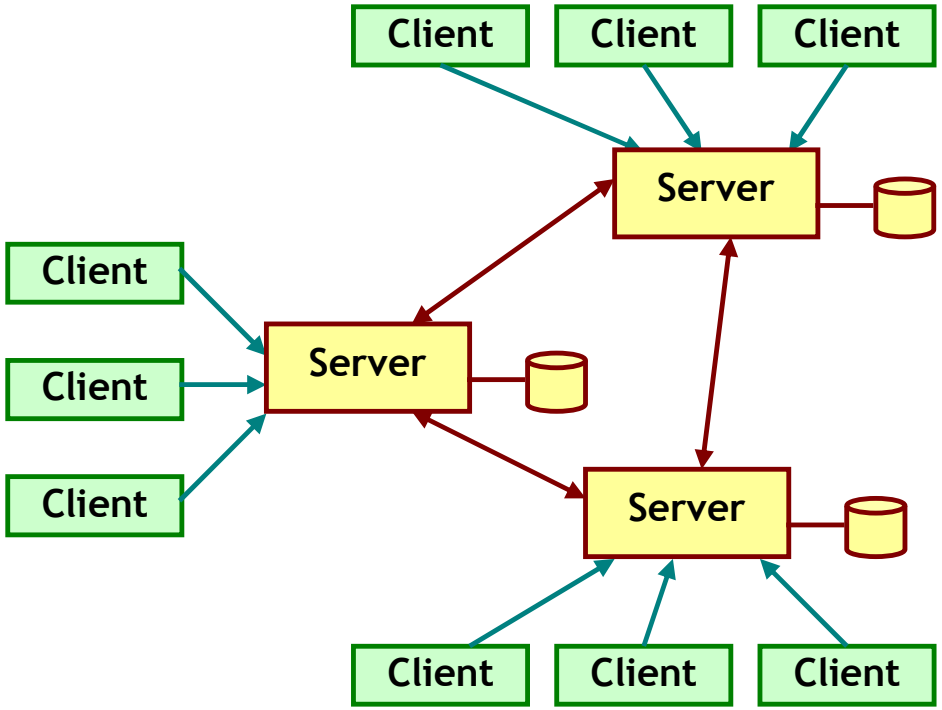
  - all clients request configuration data from database at the same instant
  - single server cannot handle such load
- Positive points:
  - all clients get identical data from database (one set for LVL2 and wider set for EF)
  - database server needs to ship only single copy of the data O(10MB) (fraction of a second over 1Gbps)

Approaches

- Have to reduce both the number of connections from clients to server and the volume of the data
- Simple solution exists
  - increase number of servers to reduce number of connections
  - bring servers “closer” to clients to reduce network traffic
- Servers can be either
  - “real” servers (DB clustering), or
  - specialized “proxy” servers

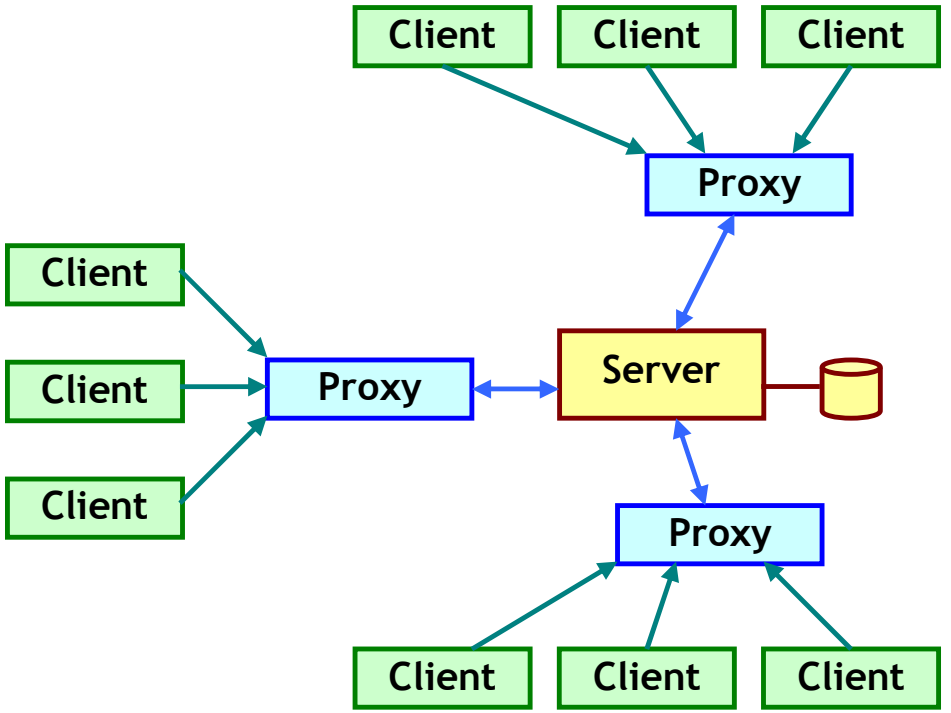
Clustering Approach

- Cluster consists of more than one tightly-connected servers
- Client chooses one (usually less loaded) server
- Cluster servers need very special and expensive hardware
- High management cost
- Solves different problems from what we need



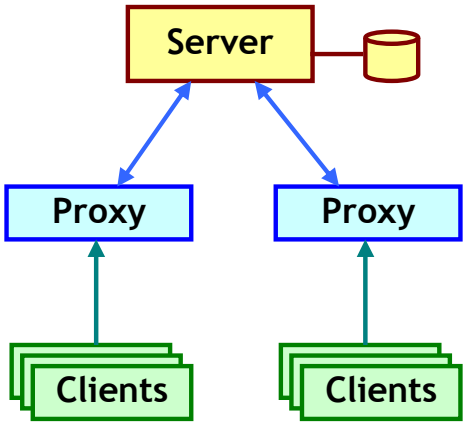
Proxy Approach

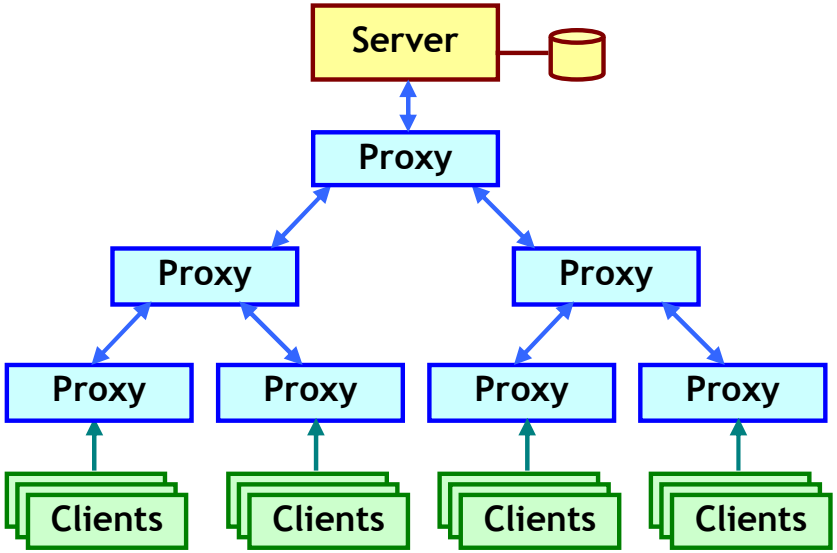
- One central database server
- Several proxy servers connect to database server
- Proxies cache the results returned by server, reduce repeated queries
- Clients connect to a “closest” proxy server



DbProxy - Design

- Two keywords — ***caching*** and ***multiplexing***
  - caching eliminates duplicate queries going to the server
  - multiplexing reduces number of connections from clients to server
- Should be possible to build hierarchies of the proxies
  - essential for scalability beyond several hundred clients

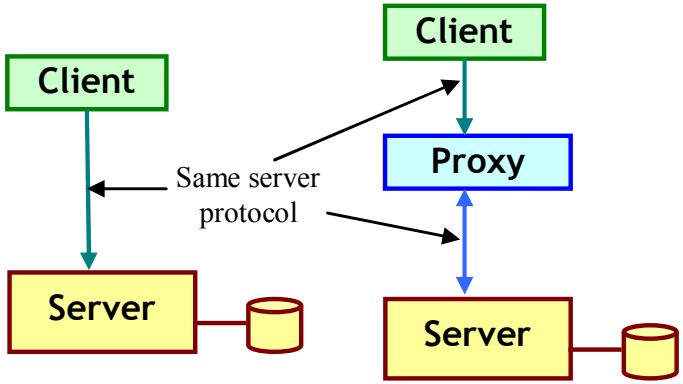


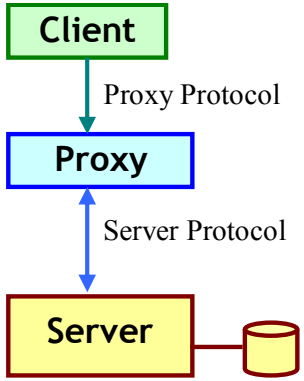


DbProxy – Proxy Transparency

Two possible types of proxies:

- *Transparent proxy*
  - does not need any modifications on the client side (except possibly configuration such as host name or port number)
  - big benefit, client code is not touched, do not need debugging on client side
- *Non-transparent proxy*
  - client has to talk different "proxy language", new code has to be added on client side (debugged, tested, etc.)
  - can be more optimal w.r.t. caching or multiplexing



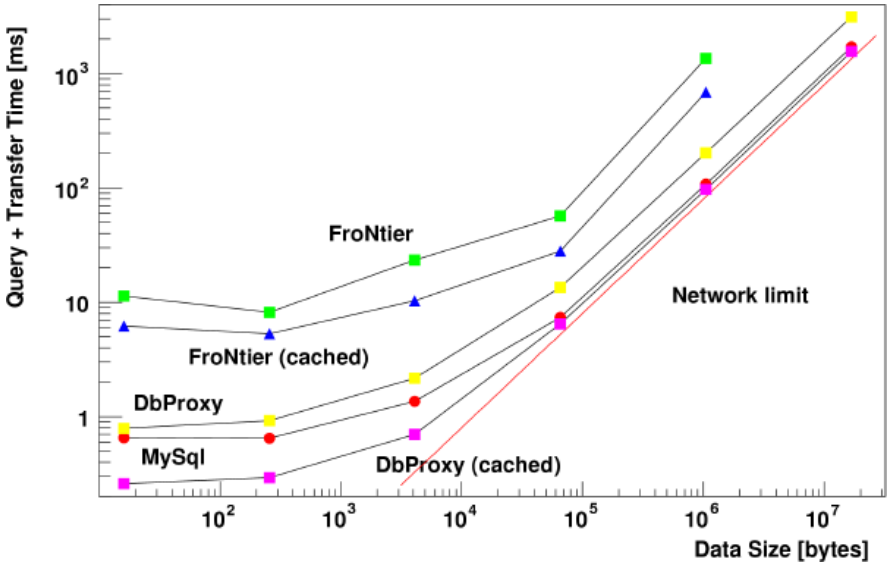


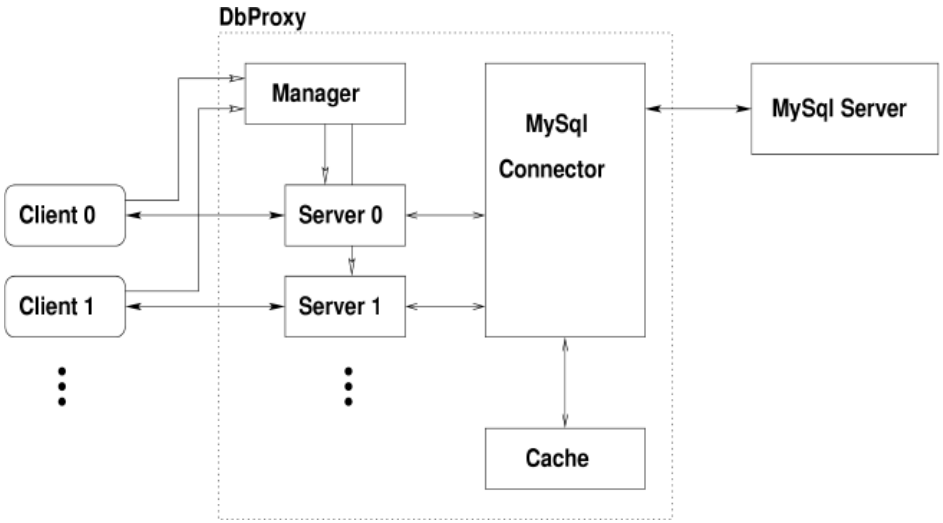
DbProxy – MySQL Implementation

- MySQL is a popular open-source database system
- Protocol is open, easy to build transparent proxy which looks exactly like MySQL server
- Some limitation though, MySQL protocol was not designed to support multiplexing
  - SQL requests have to be self-contained, no relying on external context (such as "**`USE DATABASE`**")
  - special care needed, even small modifications to the CORAL client library code

DbProxy – MySQL Implementation

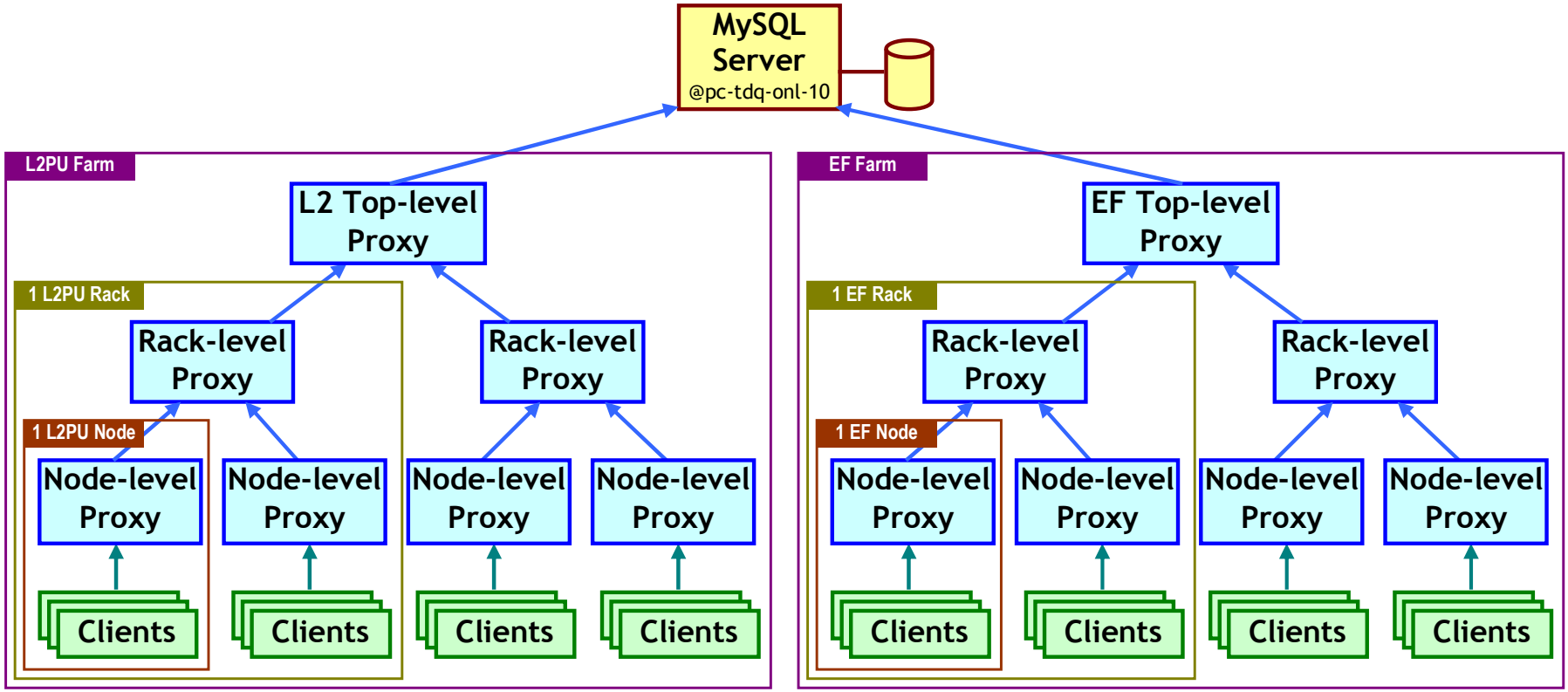
- First proxy created at SLAC
- Extensive studies with MySQL and Frontier during 2006 by Amedeo Perazzo
- Initial design and implementation by Amedeo
- Successfully tested at Point1 in the course of several Technical Runs during 2007
- Today is essential part of the TDAQ system
- Even at scale of 5 racks it was not possible to configure partition without proxy





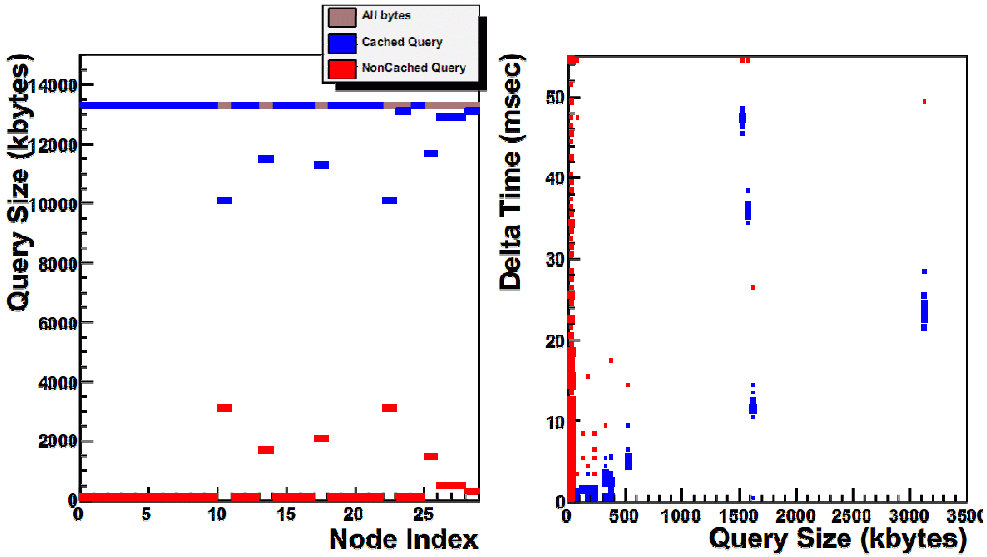
Proxy Tree at Point1

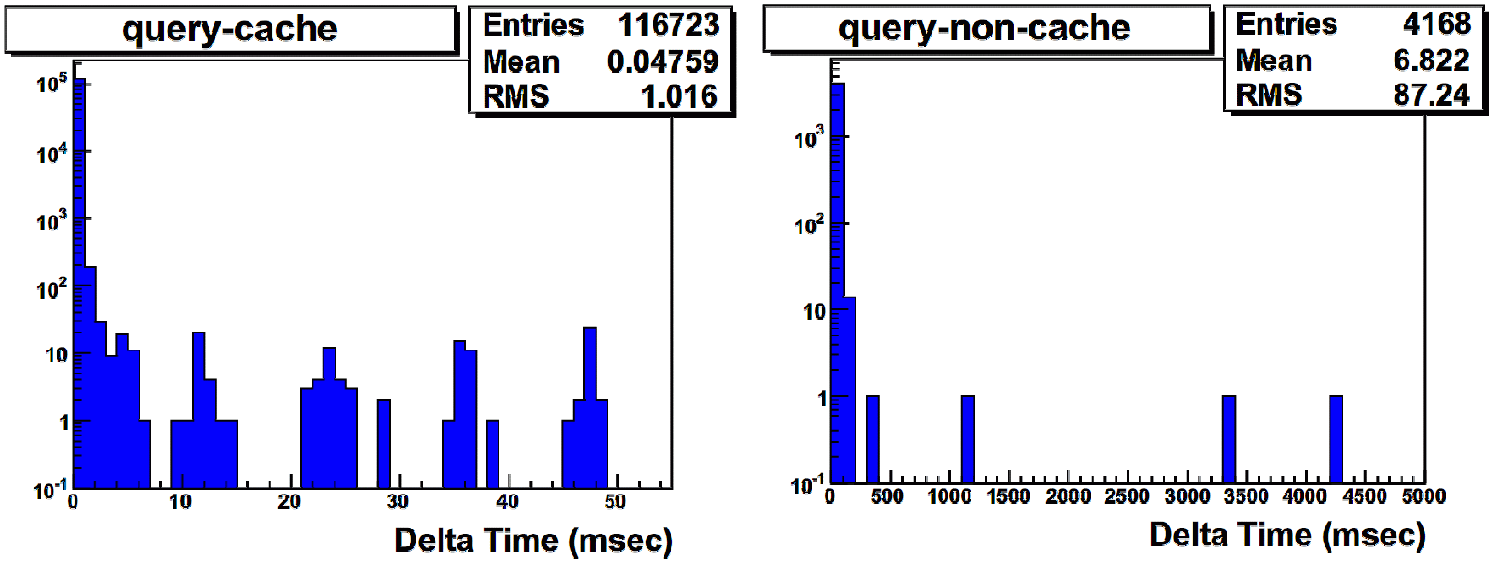
- Three-layer proxy setup during the Technical Runs at Point1
- Node-level proxy serves up to 8 L2PU/EF processes on the same node
- Rack-level proxy serves all node-level proxies in the same rack
- Top-level proxy serves all rack-level proxies in the L2PU or EF segment
- MySQL server has only two clients



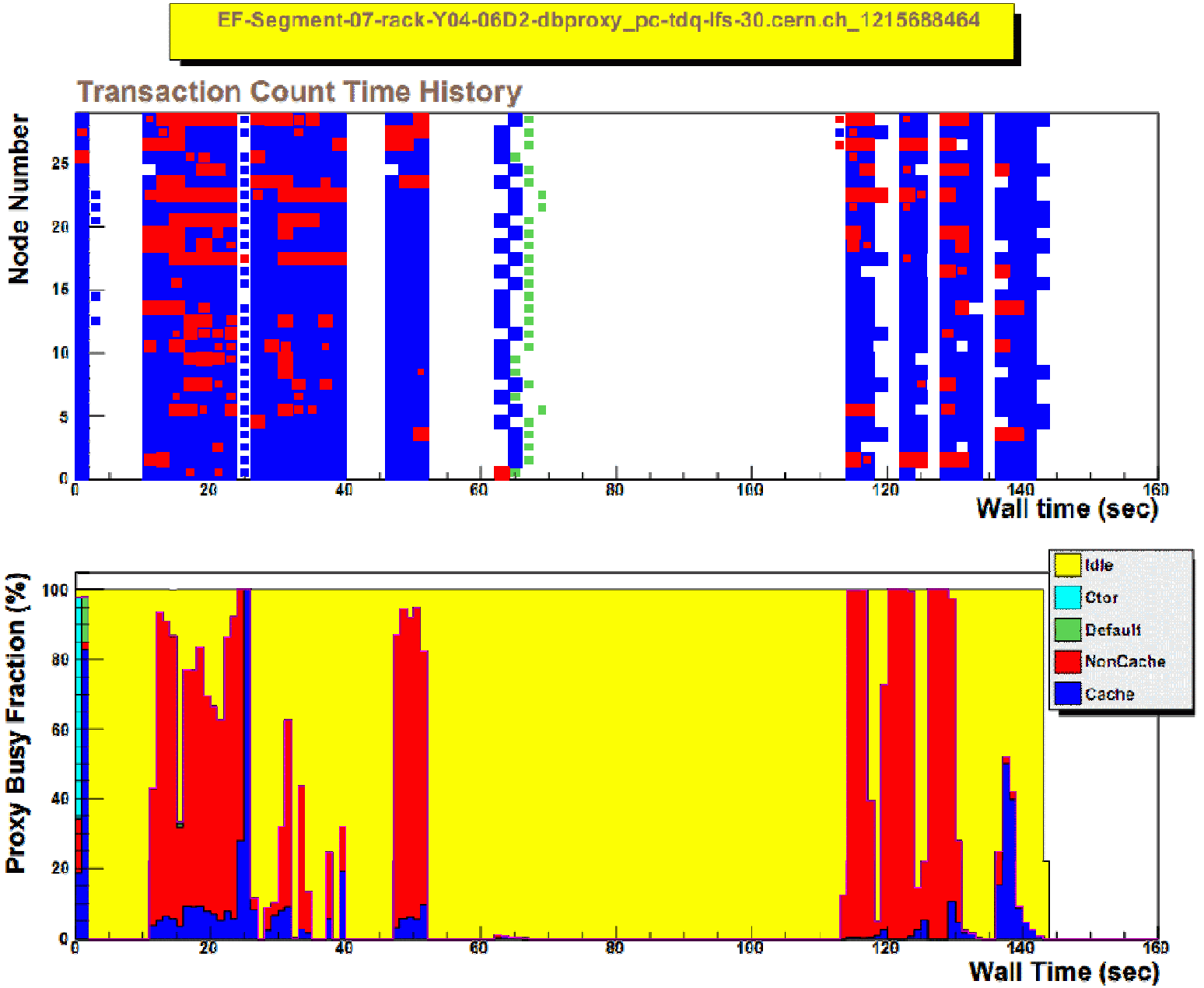
Monitoring Tool

- Performance plots regularly produced from the proxy logs
- Wealth of useful information about databse access patterns, response time, and cache behavior





Monitoring Tool



DbProxy and ORACLE

- DbProxy has to support ORACLE database
- Transparent proxy talking ORACLE protocol is not feasible, protocol is closed and proprietary
- Other options exist:
  - Keep MySQL protocol for proxy tree, translate to ORACLE at topmost level
    - may be easy to implement for subset of SQL used by CORAL
    - difficult to support in the long term, have to watch all CORAL changes for potential changes in SQL requests
    - have all drawbacks of MySQL protocol
  - Better option – non-transparent proxy
    - new protocol optimized for caching/multiplexing
    - translated to ORACLE or MySQL via existing CORAL plug-ins

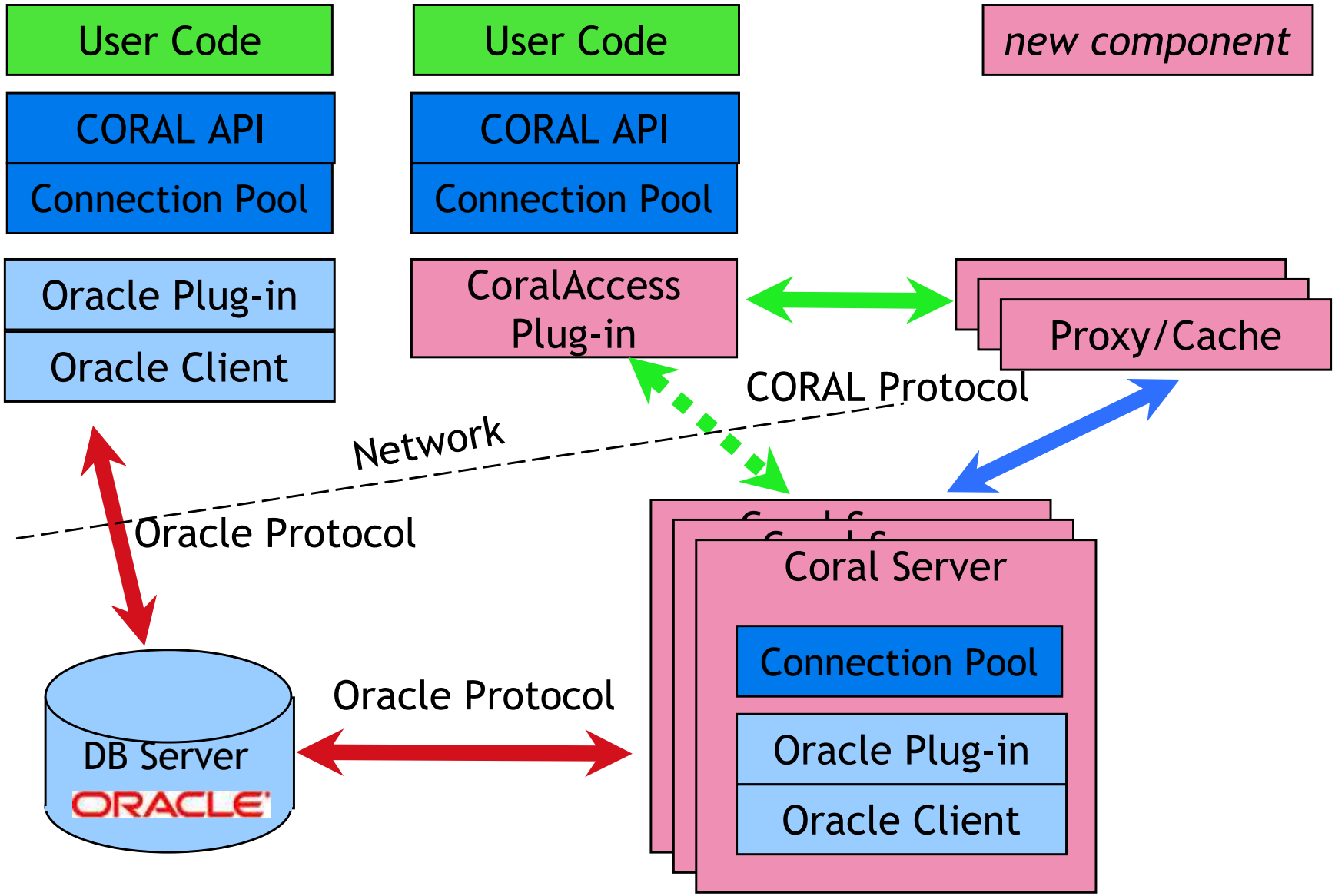
CORAL Server

- Non-transparent proxy for ORACLE access
- LHC offline world has its own potential uses for non-transparent proxy, main considerations are security and large number of clients
- Project “CORAL Server”:
  - CORAL team — Dirk Duellmann, Alexander Kalkhof, Zsolt Molnar, Andrea Valassi (CERN/IT division)
  - SLAC team with experience on caching/multiplexing side

CORAL Server – Main Components

- CORAL plug-in [CORAL team]
  - client-side plug-in library which talks new CORAL protocol
- CORAL server [CORAL team]
  - standalone server application which understands new CORAL protocol and translates it into calls to CORAL API
  - uses existing ORACLE or MySQL plug-ins to communicate to real database server
- DbProxy (CoralServerProxy) [SLAC]
  - complete re-write of the current DbProxy which understands new CORAL protocol
  - does not need to understand all details of CORAL protocol, only small part sufficient for caching and multiplexing

CORAL Server Components



Picture courtesy Dirk Duellmann (CERN/IT)

Caching in CORAL Proxy

- Held series of meetings to define transport-layer protocol for CORAL server
- New protocol is very flexible w.r.t. caching and multiplexing
  - All three components can make decision about caching of particular request
  - Multiplexing is a core part of the protocol
- Right mixture of features, optimal for solving ATLAS HLT configuration problems

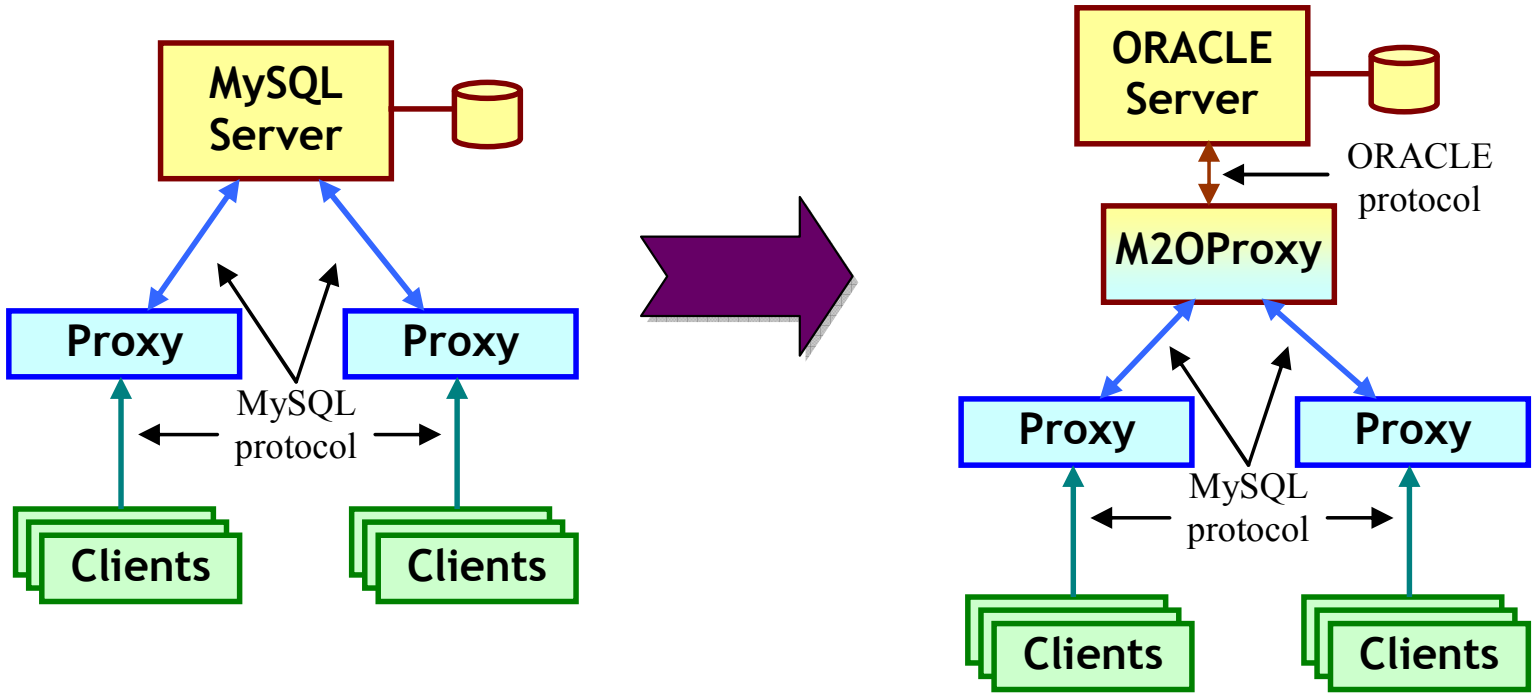
CORAL Server Status

- Weekly developers meetings
  - Main focus on minimal working prototype supporting ATLAS online needs
- Good progress so far, but slower than anticipated
  - manpower, reorganizations at CERN IT

MySQL-to-ORACLE bridging

- CORAL Server development may take longer, better to have backup plan
- ***Short-term*** solution until CORAL server is fully functional
- New proxy server which speaks MySQL wire-level protocol on one end and ORACLE on another
- Depends on several CORAL features
  - MySQL ANSI mode, most queries follow SQL standard and can be given to ORACLE without rewriting
  - Type conversion between ORACLE and MySQL is done after the rules used by CORAL plug-ins for ORACLE and MySQL
- Few MySQL-specific queries need rewriting

Yet Another Proxy



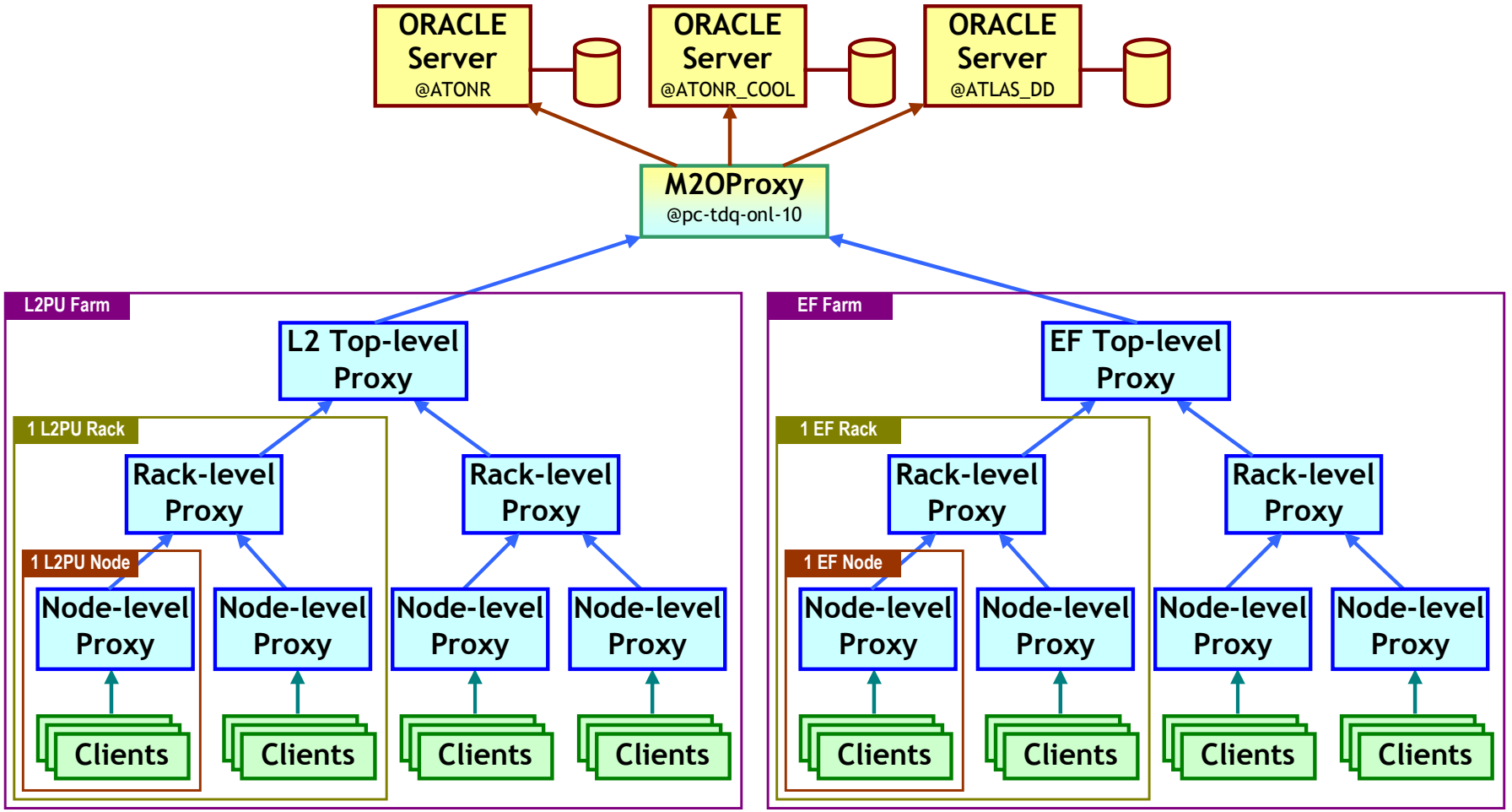
- M2OProxy is a drop-in replacement for MySQL server
- Mostly transparent, except for schema and user names
  - simple change to CORAL configuration files

M2OProxy Status

- First prototype implemented in Python last year
  - Successful tests, but needed better multi-threading and more performance than Python can provide
- Rewritten completely in C++ with ORACLE OCI library
  - Added caching of schema meta-data to speed-up MySQL schema queries
- Tested extensively on SLAC TDAQ farm and on preseries
- Integrated into TDAQ system at Point1, part of the initial ATLAS partition
  - Works with all three databases: COOL, ATLASDD, TriggerDB
  - Partition configures slightly faster than with direct ORACLE connection due to schema caching

Proxy Tree with M2OProxy

- Top-level proxies connect to the M2OProxy
- M2OProxy gets data from three Oracle Servers



POOL files

- Significant volume of conditions data is not in the databases, but in POOL files (~2MB per application, can be 200MB)
- Proxy-like mechanism for accessing POOL files from central location is possible via **`xrootd`**
  - tested successfully at SLAC farm, some speed issues
  - would need a separate tree of xrootd servers similar to DbProxy, one more entity to manage in the partition
- Better solution would be to move POOL objects into COOL database directly
  - data distribution and concurrency are serious issues with POOL
  - ORACLE replication uses ORACLE streams, POOL files are distributed with Grid DDM, extra work to synchronize them
  - already seen accidents when POOL data replication lagged behind ORACLE replication

POOL to Database

- At least two possible options here
  - *Relational POOL* – storage of POOL object in ORACLE instead of ROOT files
    - one more separate database to manage
    - may be straightforward or very difficult depending on the structure of the objects
  - *Inline BLOBs* storage for object payload in COOL
    - BLOBs contain serialized object in interchangeable format, could be the same serialized ROOT object
    - data live “close” to IOVs, no indirections and external services involved
- We plan to investigate further both options and work with the subsystems to reduce POOL files content
  - frequently changing data is most problematic
  - stable data may stay in POOL

Last Slide

- Proxy servers provide efficient and scalable solution for the HLT configuration problem
- MySQL DbProxy
  - is an integral part of online system
- MySQL-to-ORACLE bridging proxy
  - short-term solution for ORACLE access via MySQL DbProxy
  - small-scale project completely under our control
  - integrated into TDAQ
- For ORACLE access CORAL server is a natural solution
  - in active development
  - slowly moving to a working prototype
- POOL files
  - definitely a source of many problems
  - better to store content in ORACLE
  - studying possible options