ROOT
Data Bases access

LCG Data Bases deployment workshop
19 October

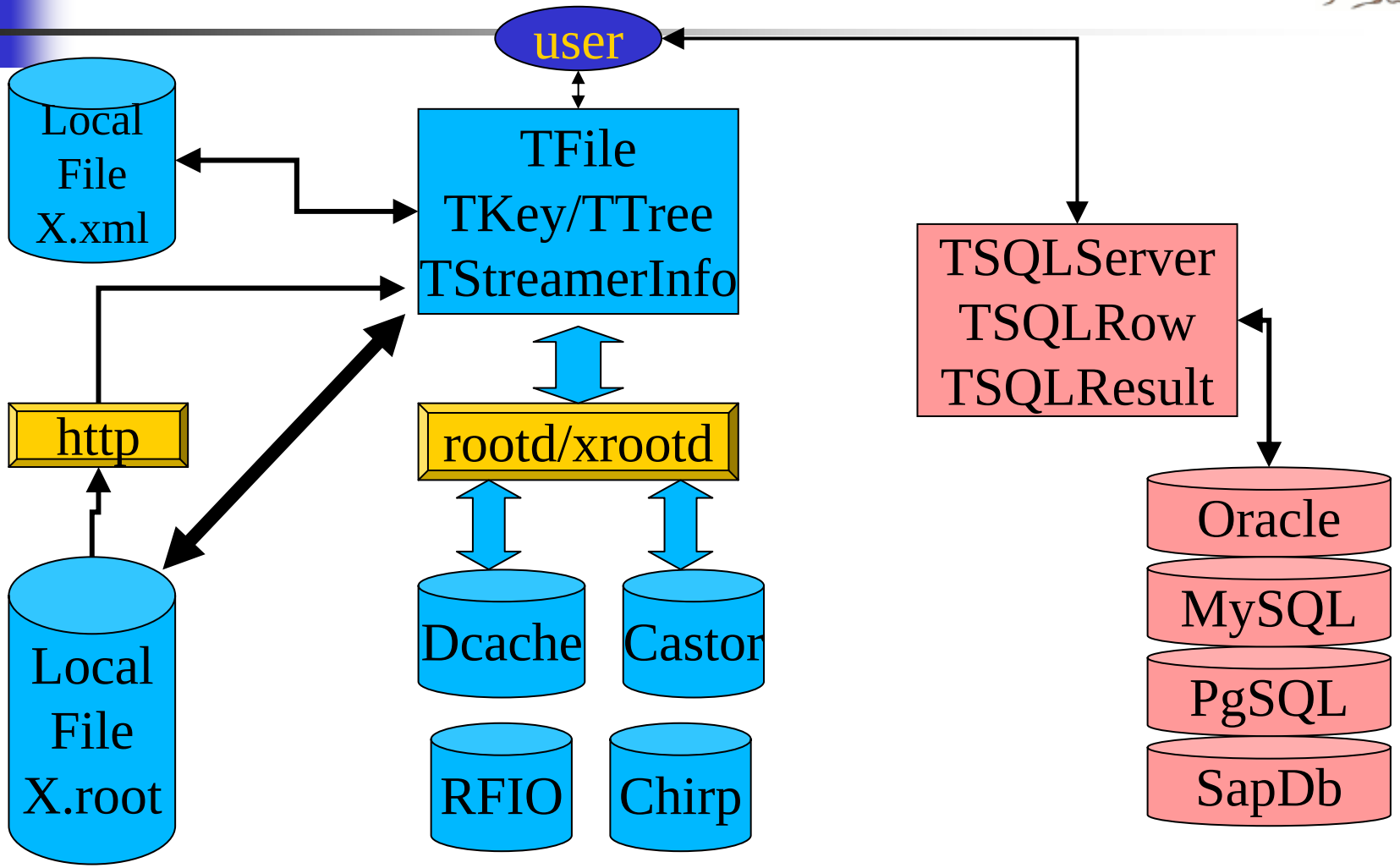
René Brun
CERN

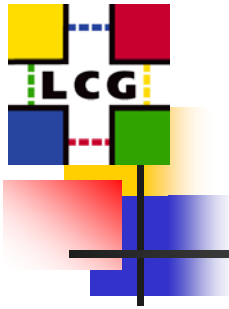
Plan

- Connections between ROOT and RDBMs
- Performance issues
- Utilities

ROOT File types & Access

(SQL implemented in 1999)



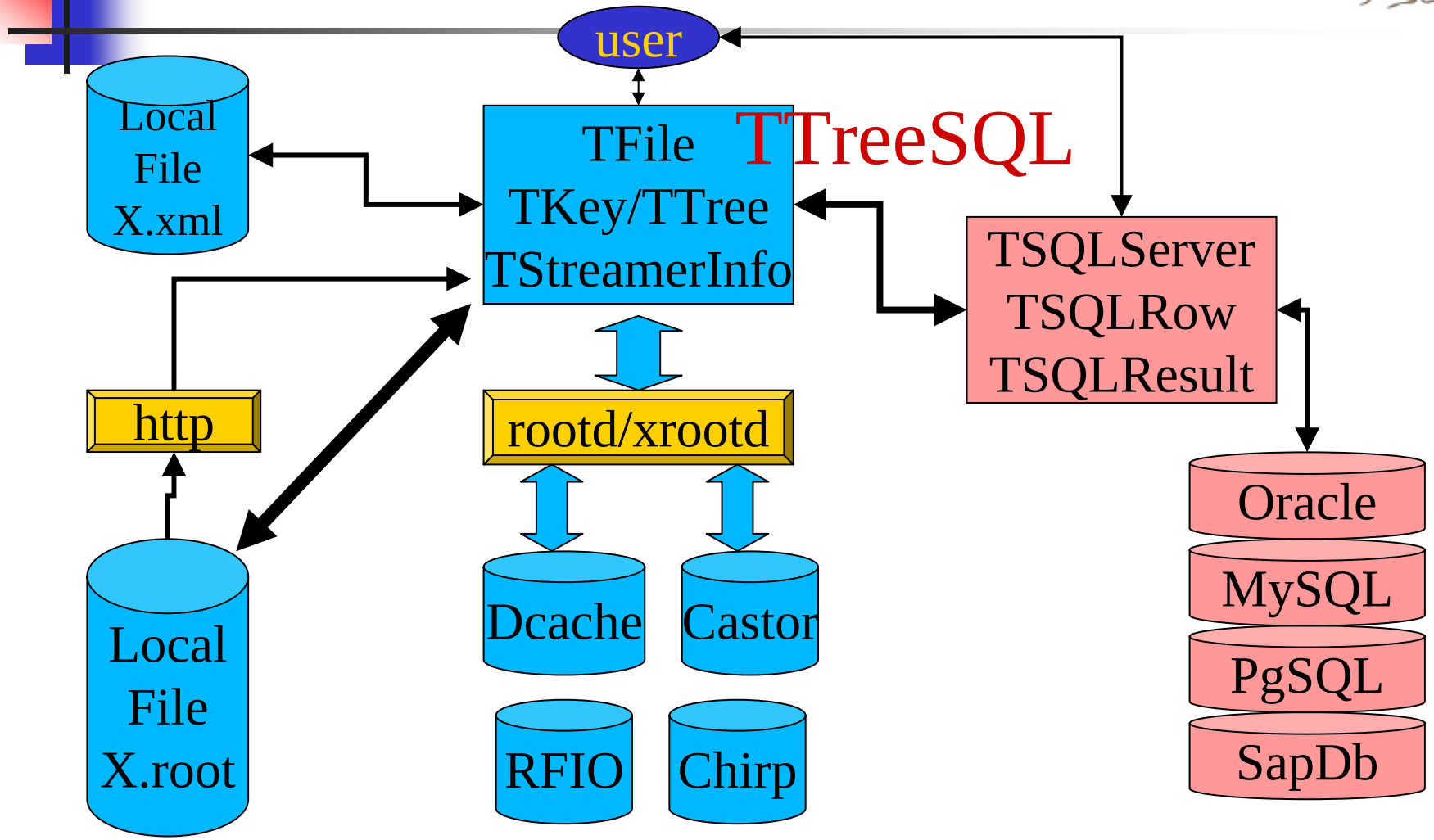


RDBC (from V.Onuchin) (implemented in 2000)

used by Phenix and Minos

- The RDBC aims for JDBC 2.0 compliance.
  - It contains the set of classes corresponding to JDBC 2.0 one
  - TSQLDriverManager, TSQLConnection, TSQLStatement, TSQLPreparedStatement,
  - TSQLCallableStatement, TSQLResultSet, TSQLResultSetMetadata, TSQLDatabaseMetadata
- The RDBC aims for ROOT SQL compliance, e.g. TSQLResult is subclass of TSQLResultSet
- RDBC implementation is based on libodbc++ library (http://orcane.net/freeodbc++) developed
- by Manush Dodunekov manush@stendahls.net
- Connection string can by either JDBC style i.e. <dbms>://<host>[:<port>][/<database>], or
  - ODBC style (as DSN) e.g. "dsn=minos;uid=scott;pwd=tiger"
- Exceptions handling is implemented via ROOT signal-slot communication mechanism.
- RDBC has an interface which allows to store ROOT objects in relational database as BLOBs.
  - For example, it is possible to store ROOT histograms, trees as a cells of SQL table.
- RDBC provides connection pooling, i.e. reusing opened connections during ROOT session.
- RDBC has an interface which allows to convert TSQLResultSets to ROOT TTrees
- RDBC with Carrot (ROOT Apache Module) allows to create three-tier architecture.

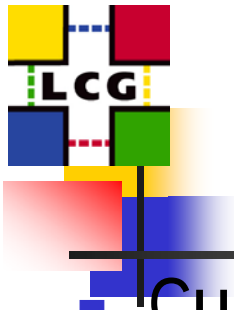
File types & Access in 5.04



New RDBMS interface in v5

- New class TTreeSQL
  - support the TTree containing branches created using a leaf list (eg. hsimple.C).
- Access any RDBMS tables from TTree::Draw

- Create a TTree in split mode
  - ➔ creating a RDBMS table and filling it.
- The table can be processed by SQL directly.
- The interface uses the normal I/O engine
  - including support for Automatic Schema Evolution.



TTreeSQL Syntax

- Currently:
  - ROOT:

```
TFile *file = new TFile("simple.root","RECREATE");
TTree *tree; file->GetObject("ntuple",tree);
```

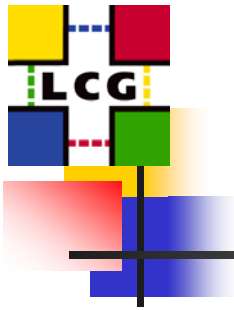
  - MySQL:

```
TSQLServer*dbserver = TSQLServer::Connect("mysql://…",db,user,passwd);
TTree *tree = new TTreeSQL(dbserver,"rootDev","ntuple");
```

- Coming:

```
TTree *tree = TTree::Open("root:/simple.root/ntuple");
```

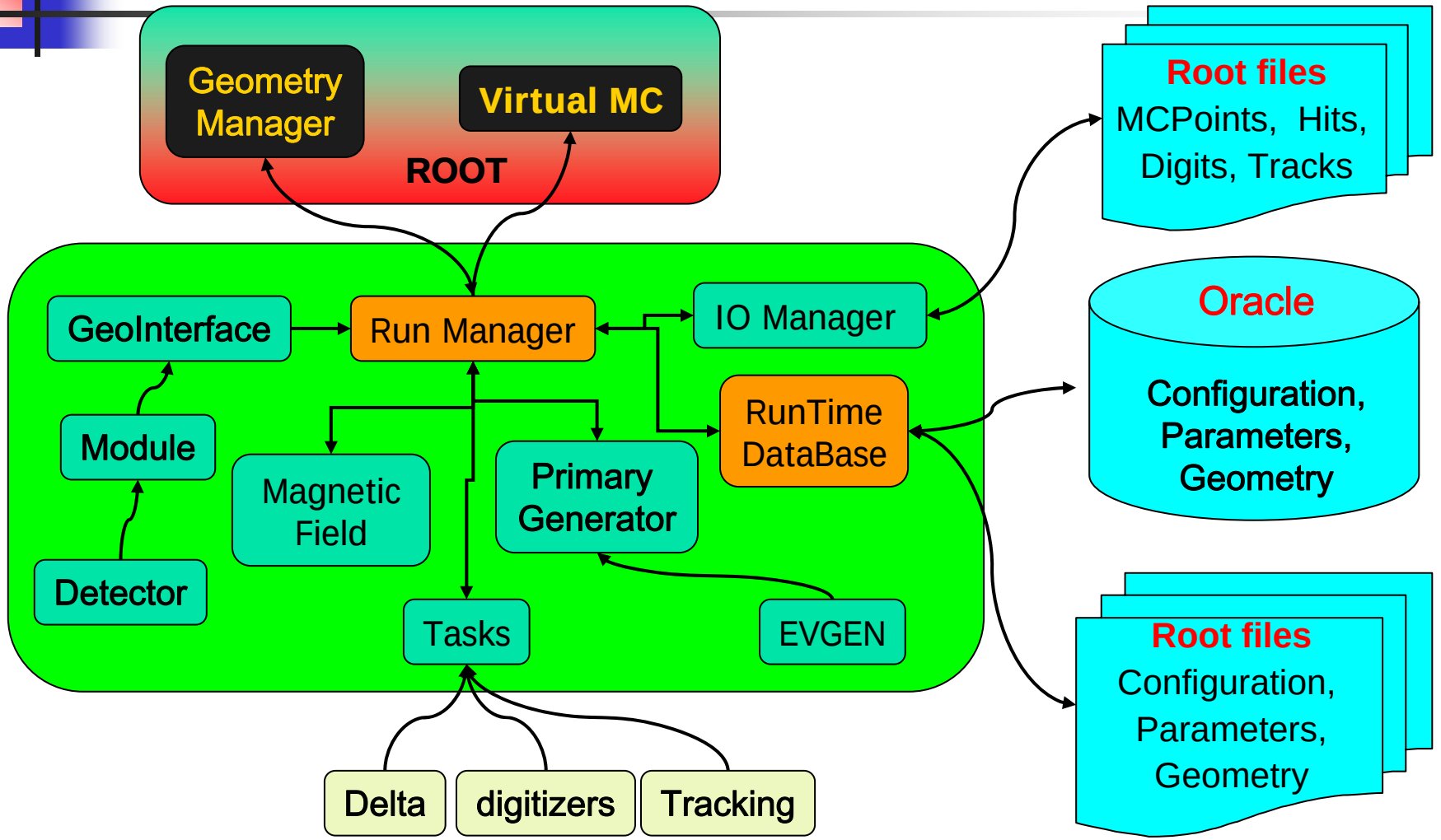
```
TTree *tree = TTree::Open("mysql://host../rootDev/ntuple");
```

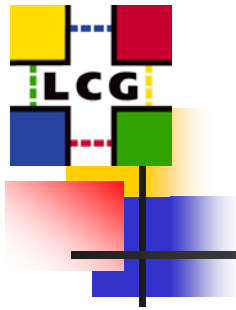


TTreeSQL Optimization

- On a simple test with a local MySQL database:
  - Reading is 5x slower than with ROOT I/O
  - Writing is functional but requires significant optimization of the code.
- Current implementation of the SQL communication (text oriented) could be greatly improved.
  - Could use some expertise in MySQL and odbc (to reinvigorate RDBC)
- Following discussions at the ROOT workshop, this will become low priority.

CBM Analysis at GSI

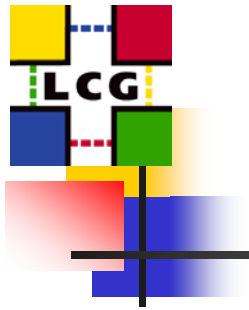




Reading Parameters: Oracle

- gSystem->Load ("libOra");
- CbmRunAna * fRun = new CbmRunAna();
- CbmRuntimeDb* rtdb=fRun->GetRuntimeDb();
- CbmParOraIo* ora=new CbmParOraIo();
- ora->open();
- rtdb->setFirstInput(ora);
- CbmGenericParOraIo* genio=
- (CbmGenericParOraIo*)(ora->getDetParIo("CbmGenericParIo"));
- CbmParTest* par=(CbmParTest*)(rtdb->getContainer("CbmParTest"));

- genio->readFromLoadingTable(par,RunId);
- par->print();
- par->printParams();
-



ROOT & RDBMS go & nogo

- ROOT interface with RDBMS is minimal
- Because there are many different use cases, we see many users with their own interface that seems appropriate in most cases.
- Because of scalability issues, the move to read-only files in a distributed environment is becoming obvious.
- We prefer to invest in a direction that we believe is very important for data analysis:
  - Optimize the use of read-only files in a distributed environment: size, read speed, read ahead & cache, selective reads (rows &columns) with Trees.
  - Optimize the performance: xrootd, load balancing, authentication with caching for interaction, robustness.

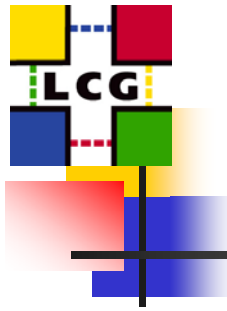
Scalability issues

- It is not the same problem any more if the CONDB is 10 Terabytes instead of 10 Gigabytes.
- CDF(300 Gb), Compass (1Tb), Alice (nTb)
- 1000 jobs accessing simultaneously a central DB looks crazy. This does not fit well with distributed computing.
- Independent read only files with load balanced servers (a la xrootd) seems the way to go.
- Keep the info in larger and structured objects, eg a RUN data structure instead of an access per event.
- Keep the RDBMS for what they are good at.

Performance Issues

ROOT compared to RDBMs

Making ROOT access faster



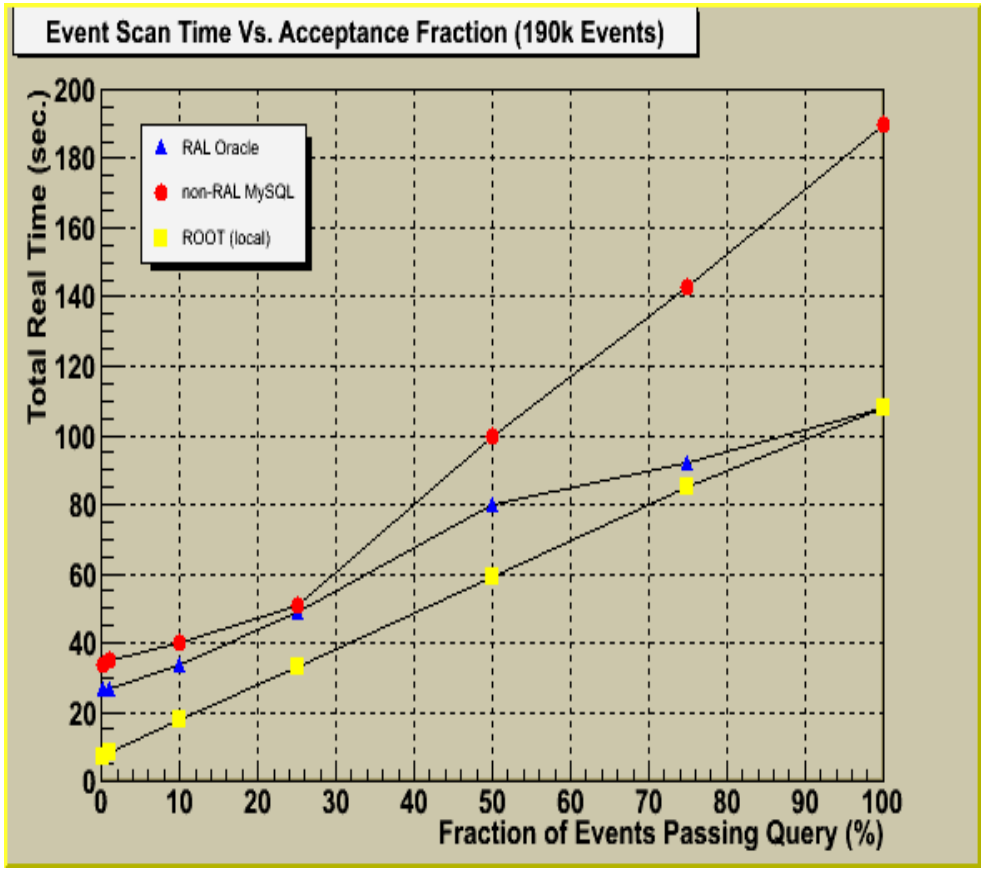
ROOT compared to RDBMs

- Several comparisons between ROOT and SQL
- Size
  - Typically 3 to 10 times smaller
- Selectivity in rows or/and columns
  - 2 to 30 times faster on local file
  - Same speed as MySQL on remote file with no cache
  - 10 times faster than mySQL when UseCache

Next two examples are from Atlas

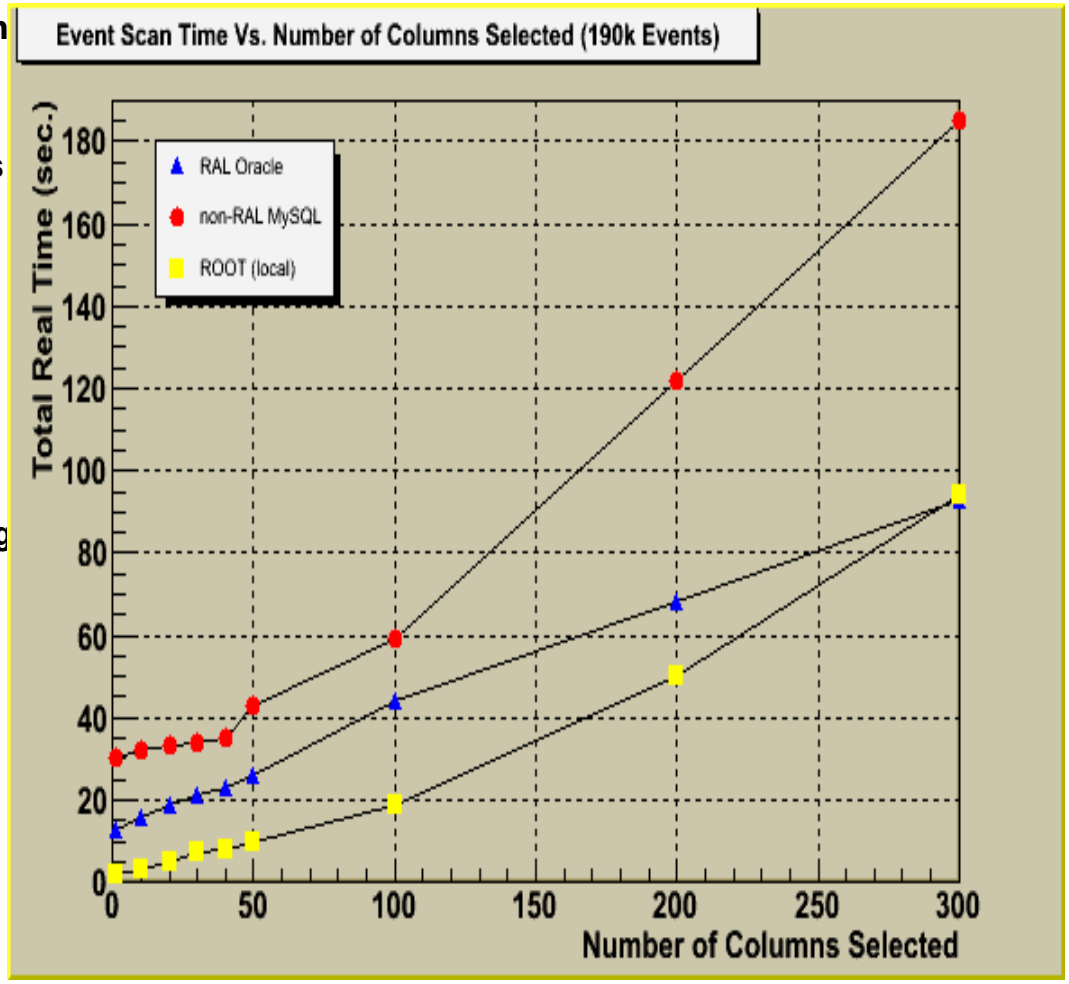
ROOT Collections: Query Time Vs. Acceptance Fraction

- Query time measured as a function of acceptance fraction.
- ROOT and Oracle query times still comparable at 75% but ROOT decreases linearly at faster rate.
- Oracle and MySQL reach minimum values ~27 sec. and ~34 sec. respectively at about 1% but ROOT continues to decrease linearly.
- Minimum values again refer to minimum amount of time needed for relational databases to scan the event sample.



ROOT Collections: Query Time Vs. Column Selection

- Query time measured as function of number of columns selected.
- ROOT and Oracle query times equivalent when all 335 columns selected but ROOT query time decreases more rapidly with decreasing column selection.
- Oracle and MySQL query times reach minimum values of ~ 15 sec. and ~ 30 sec., respectively near a selection of 10 columns.
- ROOT query times continue to decrease linearly with decreasing number of columns.
- For a selection of only one column ROOT is 7 times faster than Oracle and 15 times faster than MySQL.



TBitmapIndex: An attempt to introduce FastBit to ROOT

See J.Wu talk at ROOT workshop

Kurt Stockinger[1], Kesheng Wu[1], Rene Brun[2], Philippe Canal[3]

(1) Berkeley Lab, Berkeley, USA

(2) CERN, Geneva, Switzerland

(3) Fermi Lab, Batavia, USA

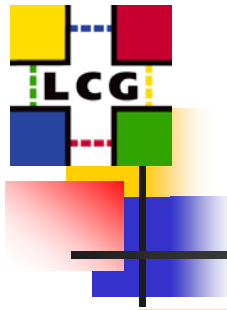
Bitmap Indices

- Bitmap indices are efficient data structures for accelerating multi-dimensional queries:
  - E.g. pT > 195 AND nTracks < 4 AND muonTight1cm > 12.4
- Supported by most commercial database management systems and data warehouses
- Optimized for read-only data
- However, because an efficient index may be a substantial fraction of the data, we think that it is only appropriate for things like event meta data catalogues

FastBit: A compressed bitmap indexing technology for efficient searching of read-only data

http://sdm.lbl.gov/fastbit

LBNL holds the copyright of the FastBit software and a US patent on the core compression technique used in FastBit. LBNL intends to seek opportunities to commercialize the searching technology and the compression technique. However, since the ROOT framework is essential to high-energy physics experiments funded by US Department of Energy, which also funded the development of FastBit and the related compression technique, LBNL has agreed to develop a license to grant ROOT users free use of the FastBit searching technology as long as FastBit is only accessed through ROOT framework. FastBit source code may also be distributed with ROOT so long as it is only used through ROOT.



Example - Build Index

```
// open ROOT-file
TFile f("data/root/data.root");
TTree *tree = (TTree*) f.Get("tree");

TBitmapIndex bitmapIndex;
char indexLocation[1024] = “/data/index/";

// build indices for all leaves of a tree
bitmapIndex.BuildIndex (tree, indexLocation);

// build index for two attributes “a1”, “a2” of a tree
bitmapIndex.BuildIndex(tree, "a1", indexLocation);
bitmapIndex.BuildIndex(tree, "a2", indexLocation);
```

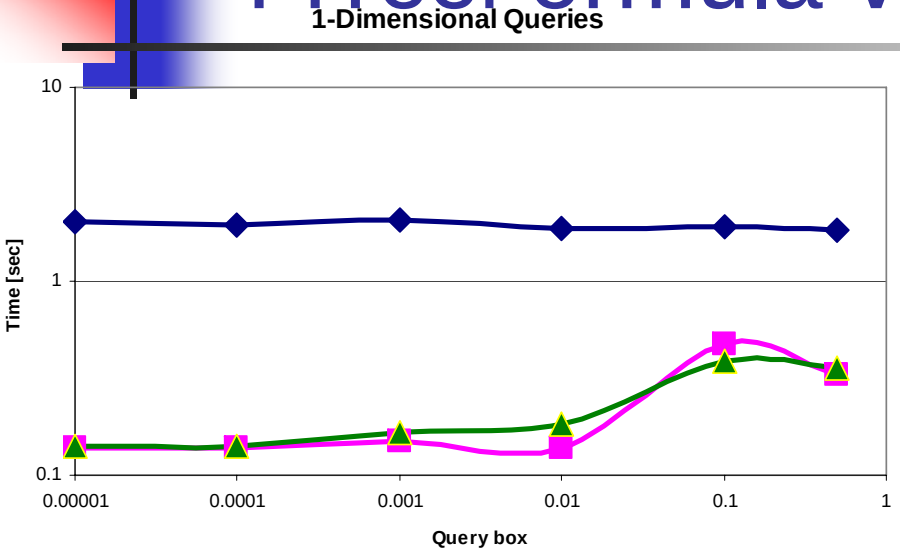
Example - Tree::Draw with Index

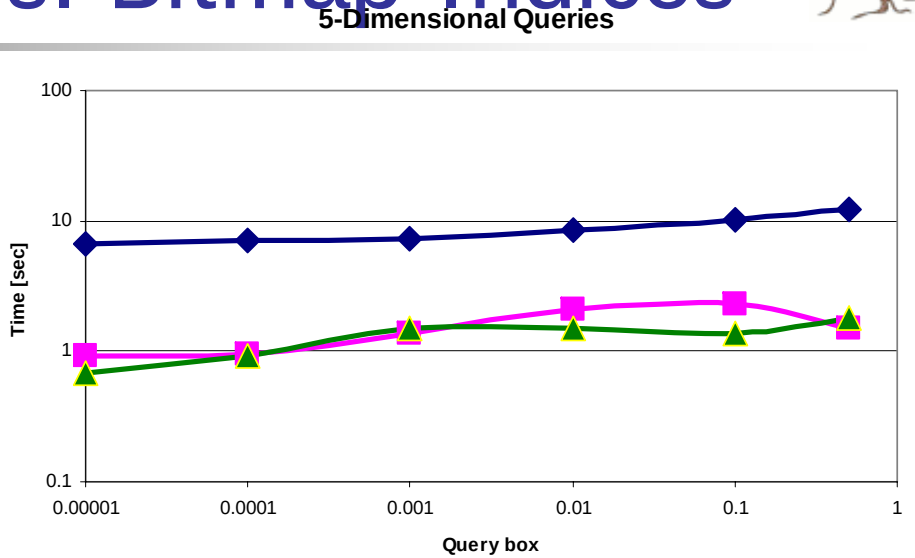
```
// open ROOT-file
TFile f("data/root/data.root");
TTree *tree = (TTree*) f.Get("tree");

TBitmapIndex bitmapIndex;
bitmapIndex.Draw(tree, "a1:a2", "a1 < 200 && a2 > 700");
```

Experiments With BaBar Data

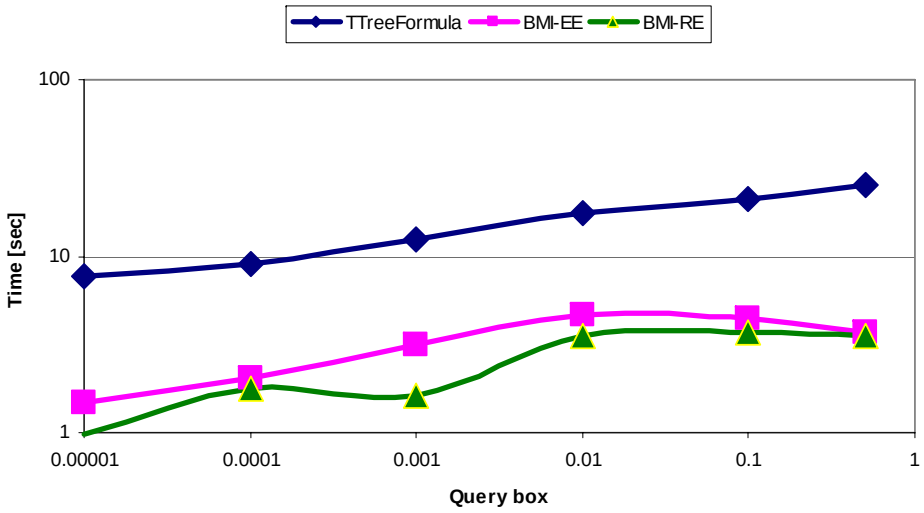
- Software/Hardware:
  - Bitmap Index Software is implemented in C++
  - Tests carried out on:
    - Linux CentOS
    - 2.8 GHz Intel Pentium 4 with 1 GB RAM
    - Hardware RAID with SCSI disk
- Data:
  - 7.6 million records with ~100 attributes each
  - Babar data set:
- Bitmap Indices (FastBit):
  - 10 out of ~100 attributes
  - 1000 equality-encoded bins
  - 100 range-encoded bins

Query Performance - TTreeFormula vs. Bitmap Indices



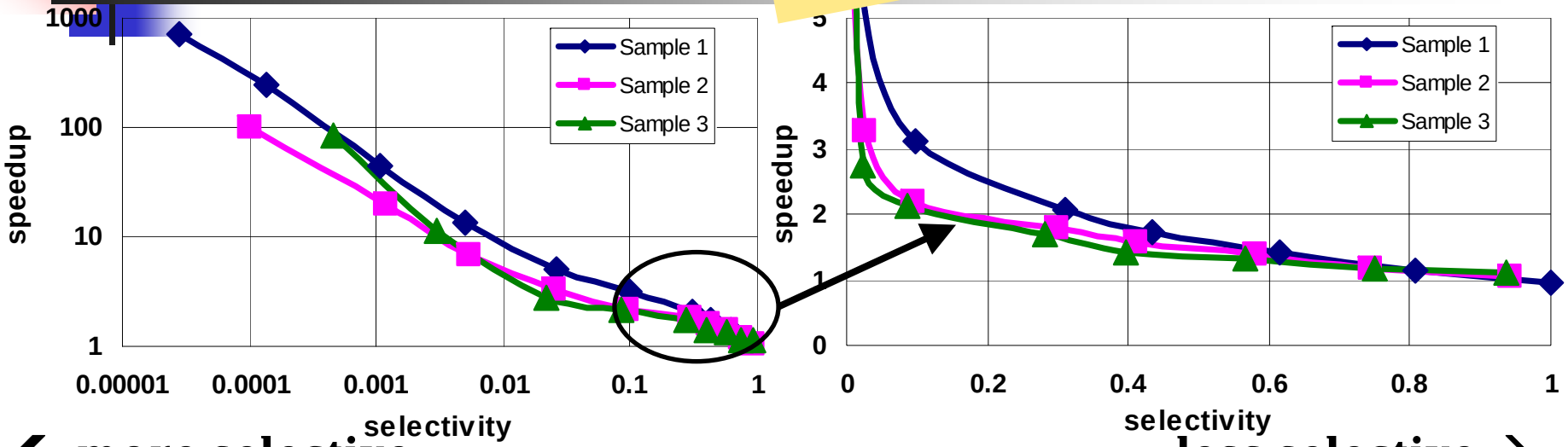


Bitmap indices 10X faster than TTreeFormula



Grid Collector Speeds up Analyses

Used in STAR Grid Collector

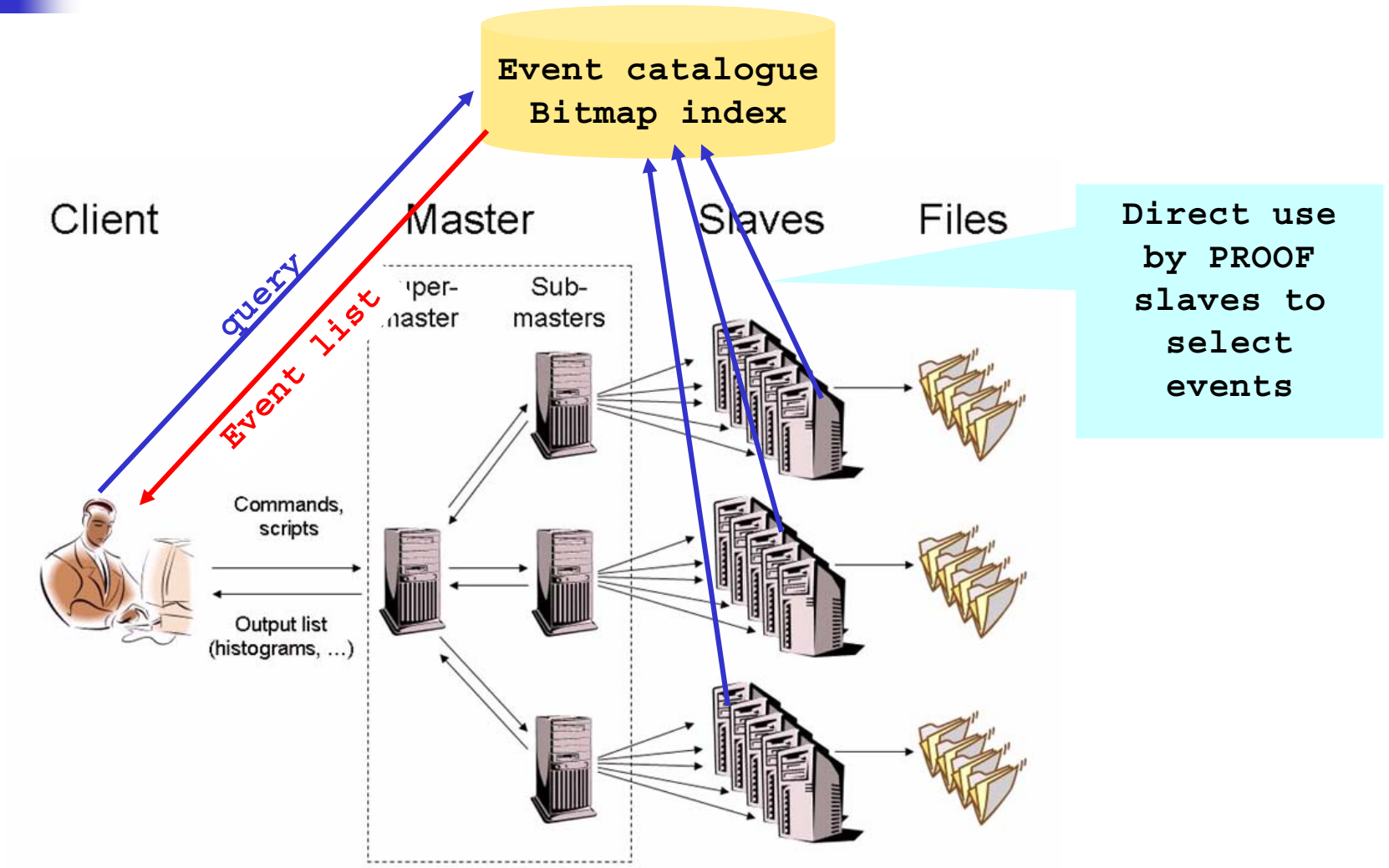


← more selective

less selective →

- Legend
  - Selectivity: fraction of events selected for an analysis
  - Speedup = ratio of time to read events without GC and with GC
  - Speedup = 1: speed of the existing system (without GC)
- Results
  - When searching for rare events, say, selecting one event out of 1000 (selectivity = 0.001), using GC is 20 to 50 times faster
  - Even using GC to read 1/2 of events, speedup > 1.5

Data analysis with bitmap indices



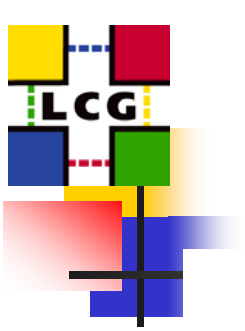
Some Utilities

Interface with GRID stuff (eg FC)

Grouping many small files

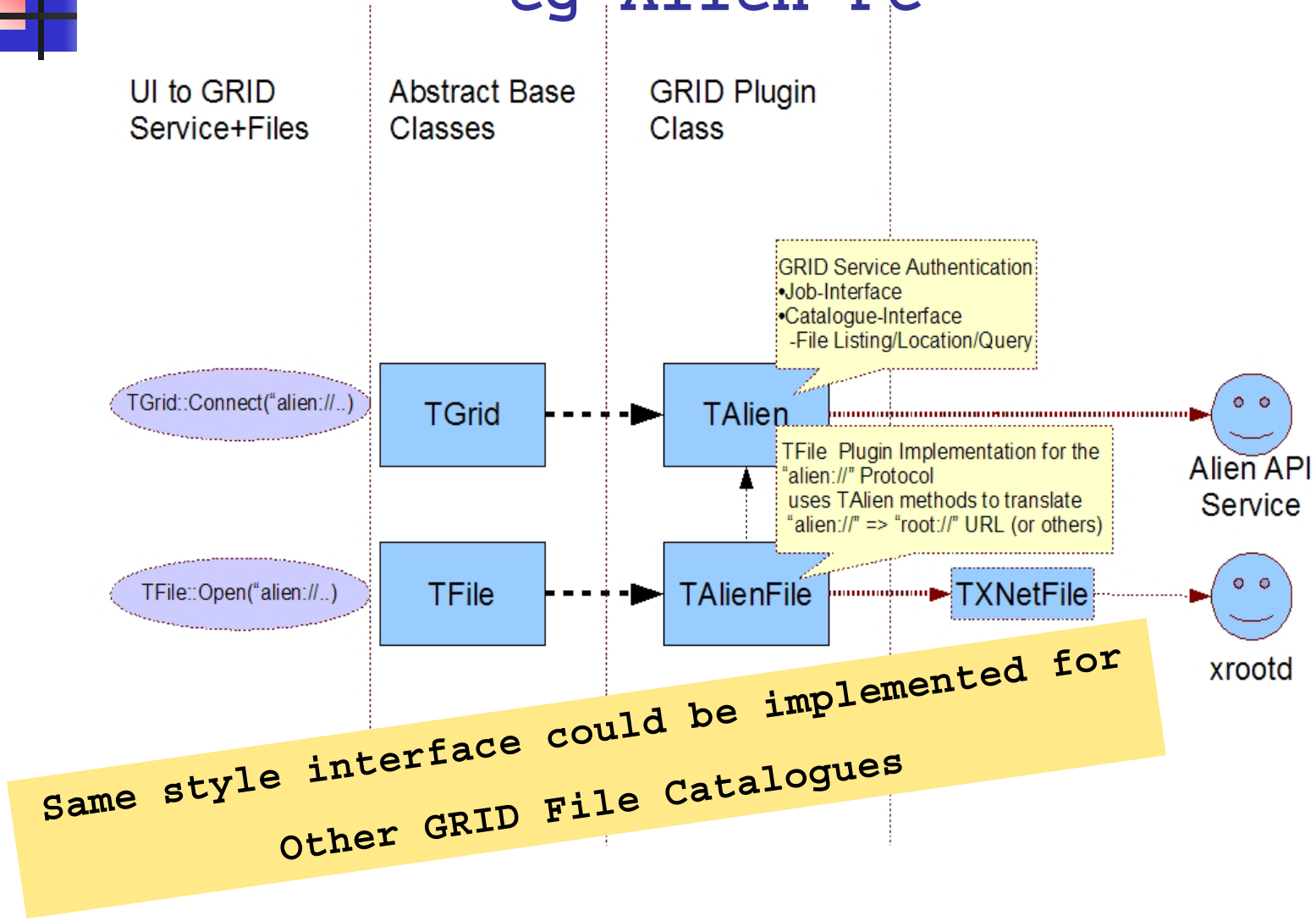
Class TGrid (abstract interface)

```
//--- General GRID
 const char    *GridUrl() const
 const char    *GetGrid() const
 const char    *GetHost() const
 const char    *GetUser() const
 const char    *GetPw()   const
 const char    *GetOptions() const
 Int_t          GetPort() const

 //--- Catalogue Interface
 virtual TGridResult *Command(const char *command,
                              Bool_t interactive = kFALSE,
                              UInt_t stream = kFALSE)

 virtual TGridResult *Query(const char *path, const char *pattern,
                            const char *conditions,
                            const char *options)

 virtual TGridResult *LocateSites()

 virtual TGridResult *ls(const char*ldn ="", Option_t*options ="")
 virtual Bool_t cd(const char*ldn ="",Bool_t verbose =kFALSE)
 virtual Bool_t mkdir(const char*ldn ="", Option_t*options ="")
 virtual Bool_t rmdir(const char*ldn ="", Option_t*options ="")
 virtual Bool_t register(const char *lfn , const char *turl , Long_t size,
                         const char *se, const char *guid)
 virtual Bool_t rm(const char*lfn , Option_t*option ="")

 //--- Job Submission Interface
 virtual TGridJob *Submit(const char *jdl)
 virtual TGridJDL *GetJDLGenerator()

 //--- Load desired plugin and setup conection to GRID
 static TGrid *Connect(const char *grid, const char *uid,
                       const char *pw, const char *options)
```



Access to File Catalogues

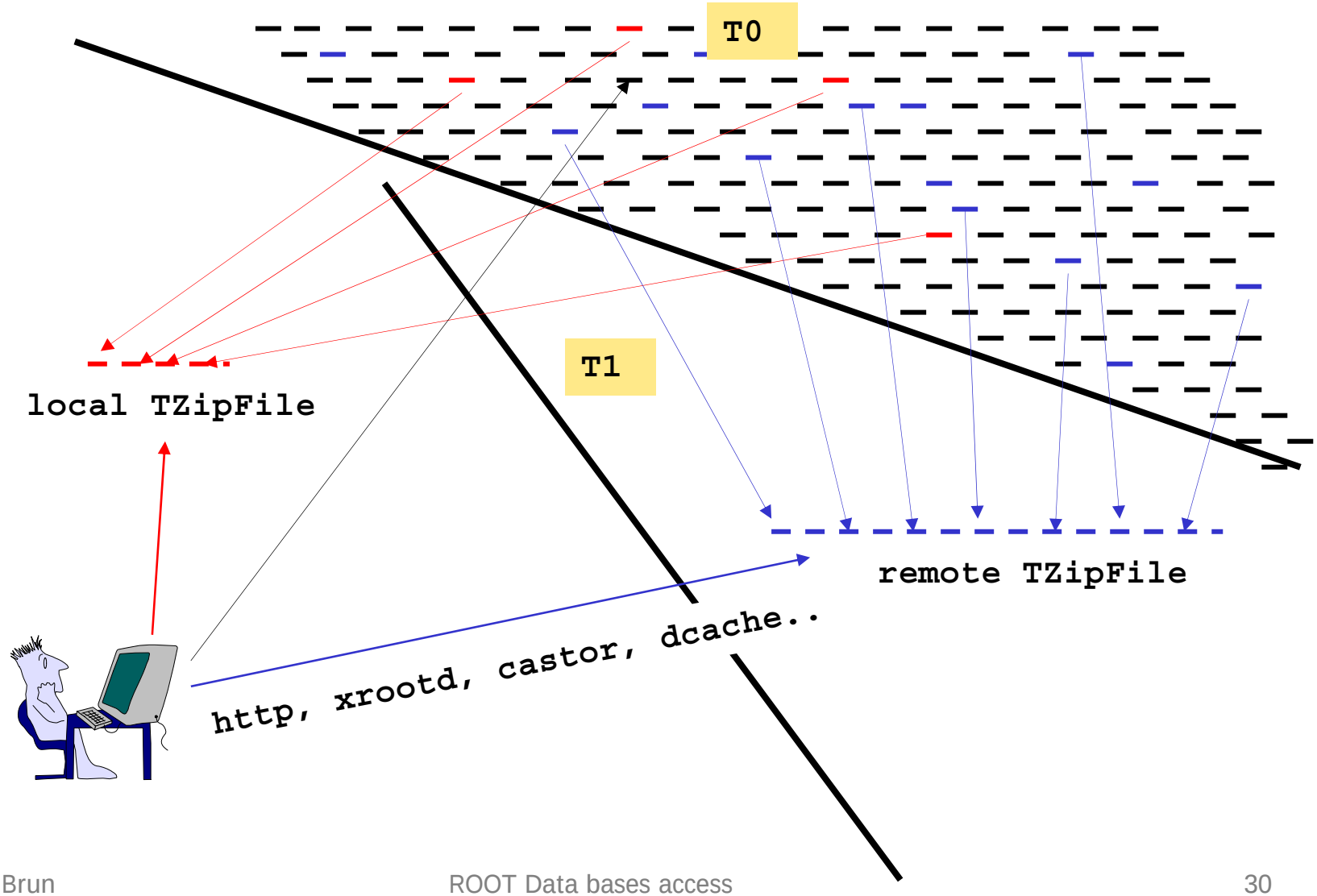
eg Alien FC

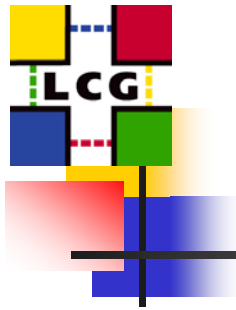


TGrid example with Alien

```
// Connect
TGrid alien = TGrid::Connect("alien://");

// Query
TGridResult *res =alien.Query
("/alice/cern.ch/user/p/peters/analysis/miniesd/",
                "*.root");
// List of files
TList *listf = res->GetFileInfoList();

// Create chain
TChain chain("Events", "session");
Chain.AddFileInfoList(listf);

// Start PROOF
TProof proof("remote");

// Process your query
Chain.Process("selector.C");
```

Replica of a DB subset





TArchiveFile and TZIPFile

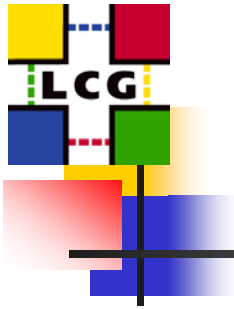
- TArchiveFile is an abstract class that describes an archive file containing multiple sub-files, like a ZIP or TAR archive.
- The TZIPFile class describes a ZIP archive file containing multiple ROOT sub-files. Notice that the ROOT files should not be compressed when being added to the ZIP file, since ROOT files are normally already compressed. To create the file multi.zip do:

```
zip –n root multi file1.root file2.root
```

- The ROOT files in an archive can be simply accessed like this:

```
TFile *f = TFile::Open("multi.zip#file2.root")
  or
TFile *f = TFile::Open("root://mymachine/multi.zip#2")
```

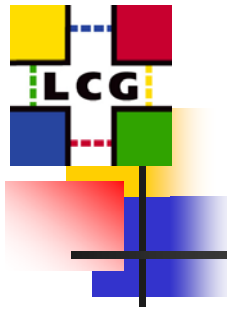
- A TBrowser and TChain interface will follow shortly.



TFileMerger

- This new class allows for easy copying of two or more files using the many TFile plugins (i.e. it can copy from Castor to dCache, or from xrootd to Chirp, etc.).
-
```
TFileMerger m;
m->Cp("srcUrl", "destUrl");
or
m->AddFile("url1");
m->AddFile("url2");
m->Merge();
```

- The AddFile() and Merge() use the Cp() to copy the file locally before making the merge, and if the output file is remote the merged file will be copied back to the remote destination.



New TFile Feature

- Support for opening files in raw mode when the file url contains the option string "filetype=raw", like "anyfile.tgz?filetype=raw".
- This allows TFile and its many remote access plugins to be used to open and read any file.
- This is used by the TFileMerger::Cp() method to copy any file from and to Grid storage elements (e.g. from Castor to dCache, from xrootd to a local file, and all possible permutations).