An Architectural Blueprint for the Common LHC Physics Application Software

Torre Wenaus, BNL/CERN

LCG Applications Area Manager

http://cern.ch/lcg/peb/applications

LCG Seminar

November 6, 2002

Outline

- Introduction to the LHC Computing Grid (LCG) Project and Applications Area
- Architecture Blueprint RTAG
- An architectural blueprint for LCG physics application software
- RTAG recommendations and outcomes
- The blueprint and POOL
- Concluding remarks
- Discussion

LHC Software – General Principles

From the Hoffmann review

- Object oriented software using C++
  - Modularity makes collaborative development easier
  - ‘Implementation hiding’ eases maintenance and evolution
- Take software engineering seriously
  - Treat as a project: planning, milestones, resource management
  - Build for adaptability over a ~20yr project lifetime
  - Careful management of software configuration
    - Reproducibility of analyses
- Employ common solutions
  - HENP community-developed tools; open source; commercial
  - Minimize in-house development
    - But when necessary, develop common solutions **(LCG Project)**

Hence the LCG Applications Area…

The LHC Computing Grid (LCG) Project

Goal – Prepare and deploy the LHC computing environment

- Approved (3 years) by CERN Council, September 2001
  - meanwhile extended by 1 year due to LHC delay
- Injecting substantial new facilities and personnel resources
- Scope:
  - Common software for physics applications
    - Tools, frameworks, analysis environment
  - Computing for the LHC
    - Computing facilities (fabrics)
    - Grid middleware, deployment

→ Deliver a global analysis environment

LCG Areas of Work

Computing System

- Physics Data Management
- Fabric Management
- Physics Data Storage
- LAN Management
- Wide-area Networking
- Security
- Internet Services

Grid Technology

- Grid middleware
- Standard application services layer
- Inter-project coherence/compatibility

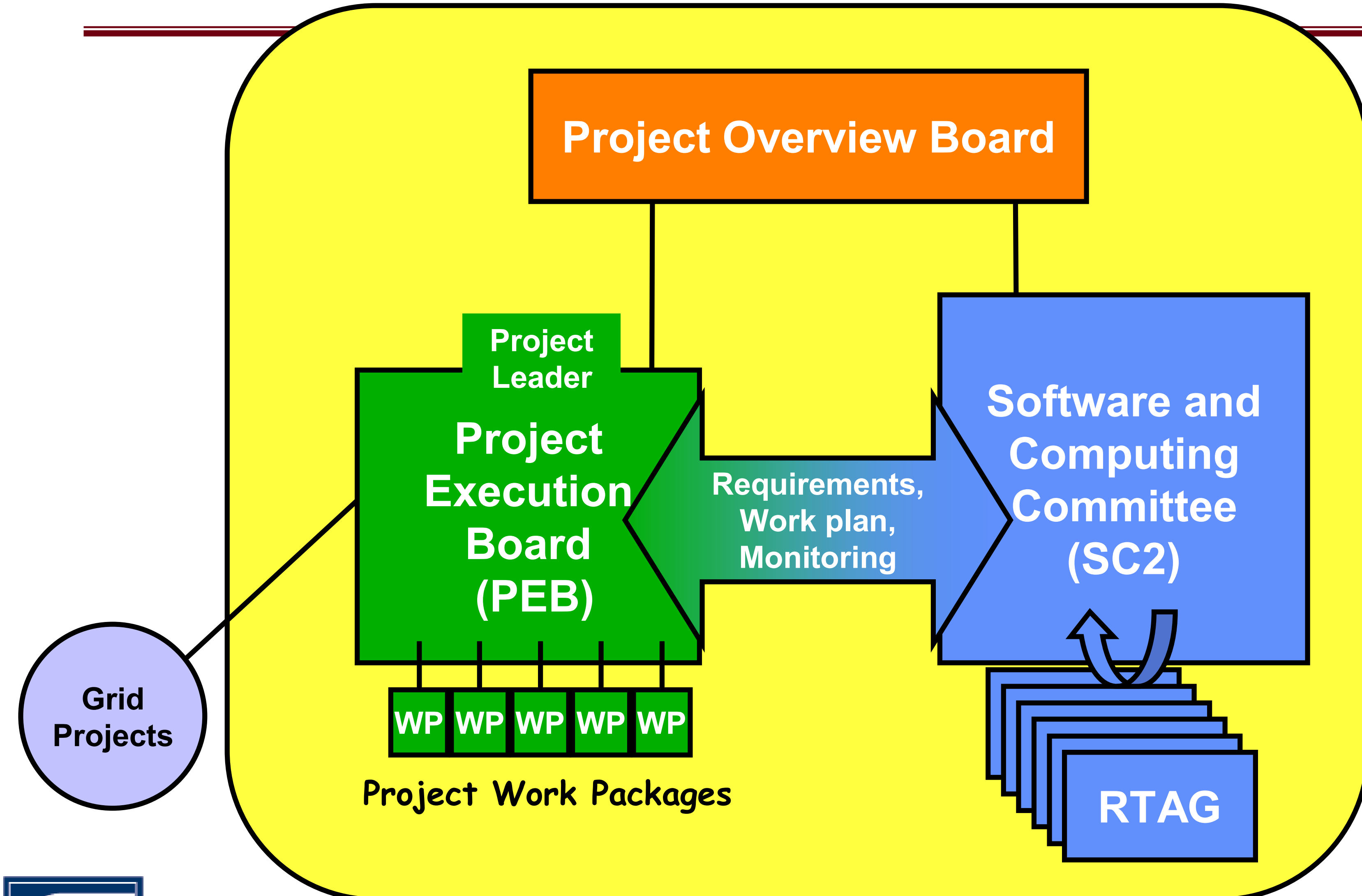
Physics Applications Software

- Application Software Infrastructure – libraries, tools
- Object persistency, data management tools
- Common Frameworks – Simulation, Analysis, ..
- Adaptation of Physics Applications to Grid environment
- Grid tools, Portals

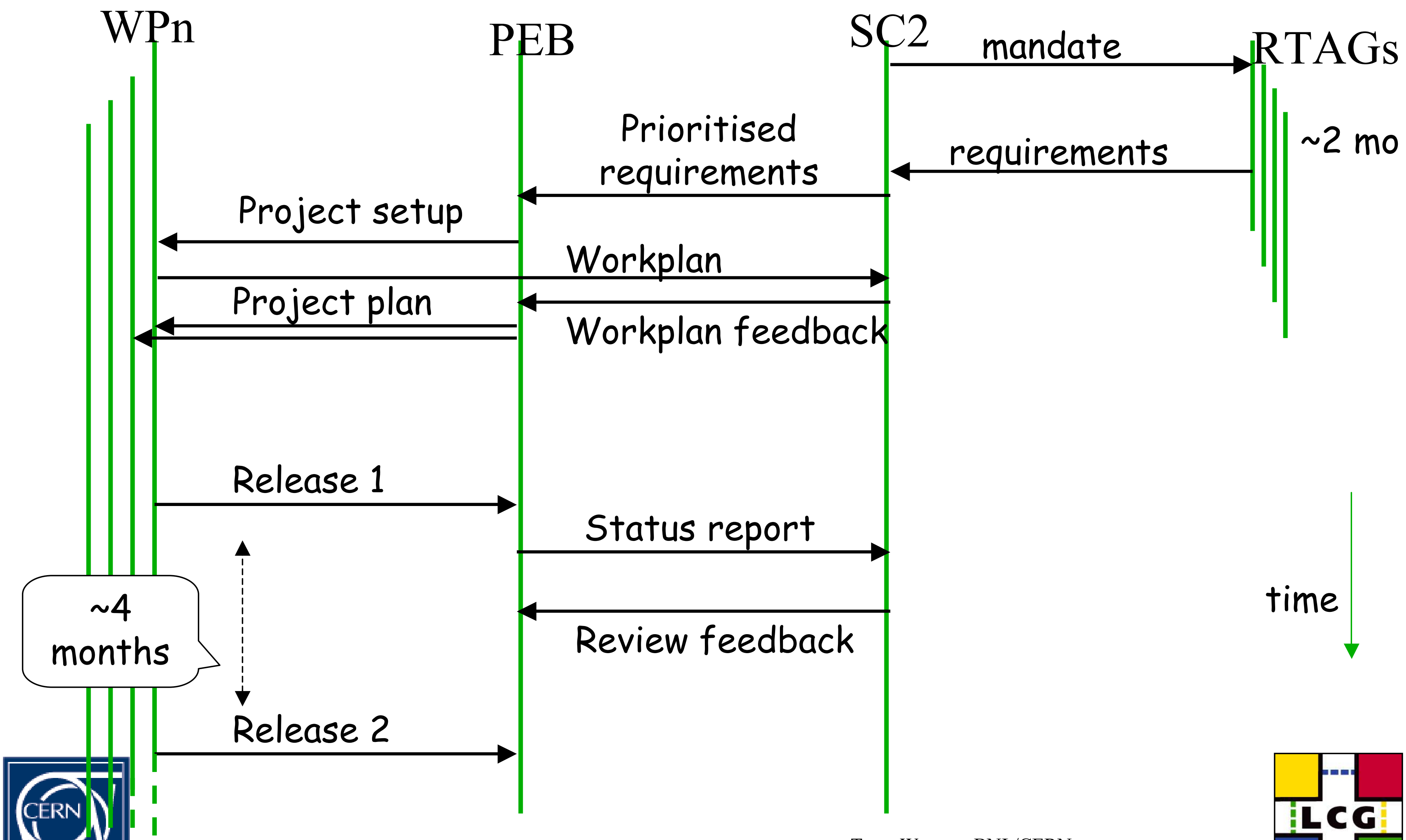
Grid Deployment

- Data Challenges
- Grid Operations
- Network Planning
- Regional Centre Coordination
- Security & access policy

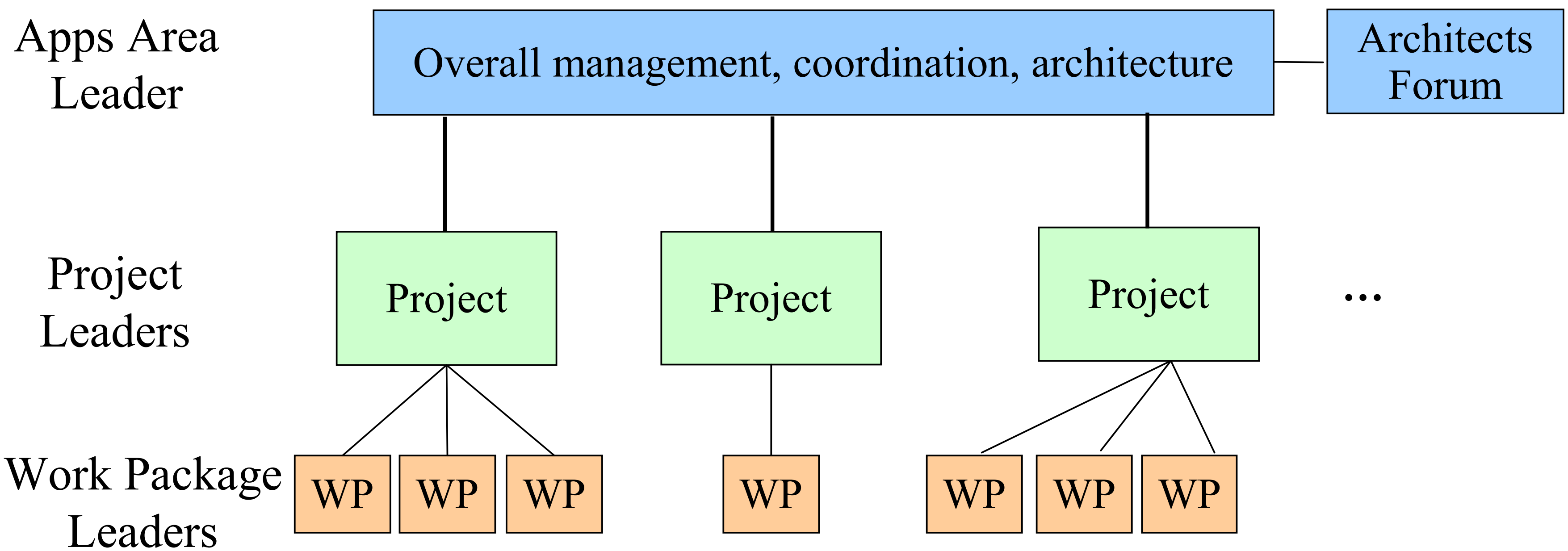
The LHC Computing Grid Project Structure



LCG Workflow



Applications Area Organization

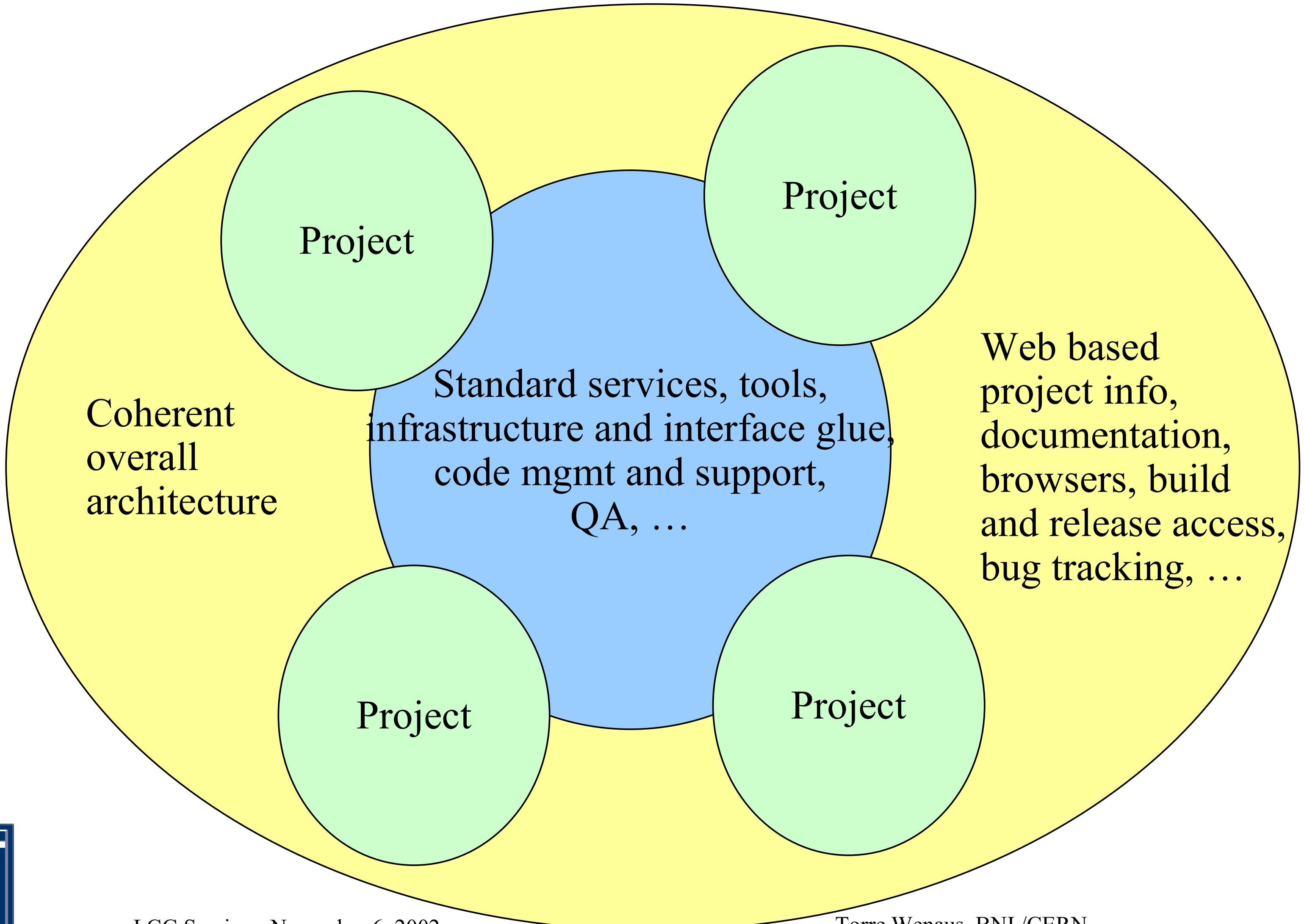


Direct technical collaboration between experiment participants,
IT, EP, ROOT, LCG personnel

Architects Forum

- Members: experiment architects + applications area manager (chair)
  - Plus invited (e.g. project leaders)
  - Architects represent the interests of their experiment
- Decides the difficult issues
  - Most of the time, AF will converge on a decision
  - If not, applications manager takes decision
    - Such decisions can be accepted or challenged
- Challenged decisions go to full PEB, then if necessary to SC2
  - We all abide happily by an SC2 decision
- Meetings at ~2 week intervals
- Forum decisions, actions documented in public minutes

Cohesion Across Projects



Applications Area Projects

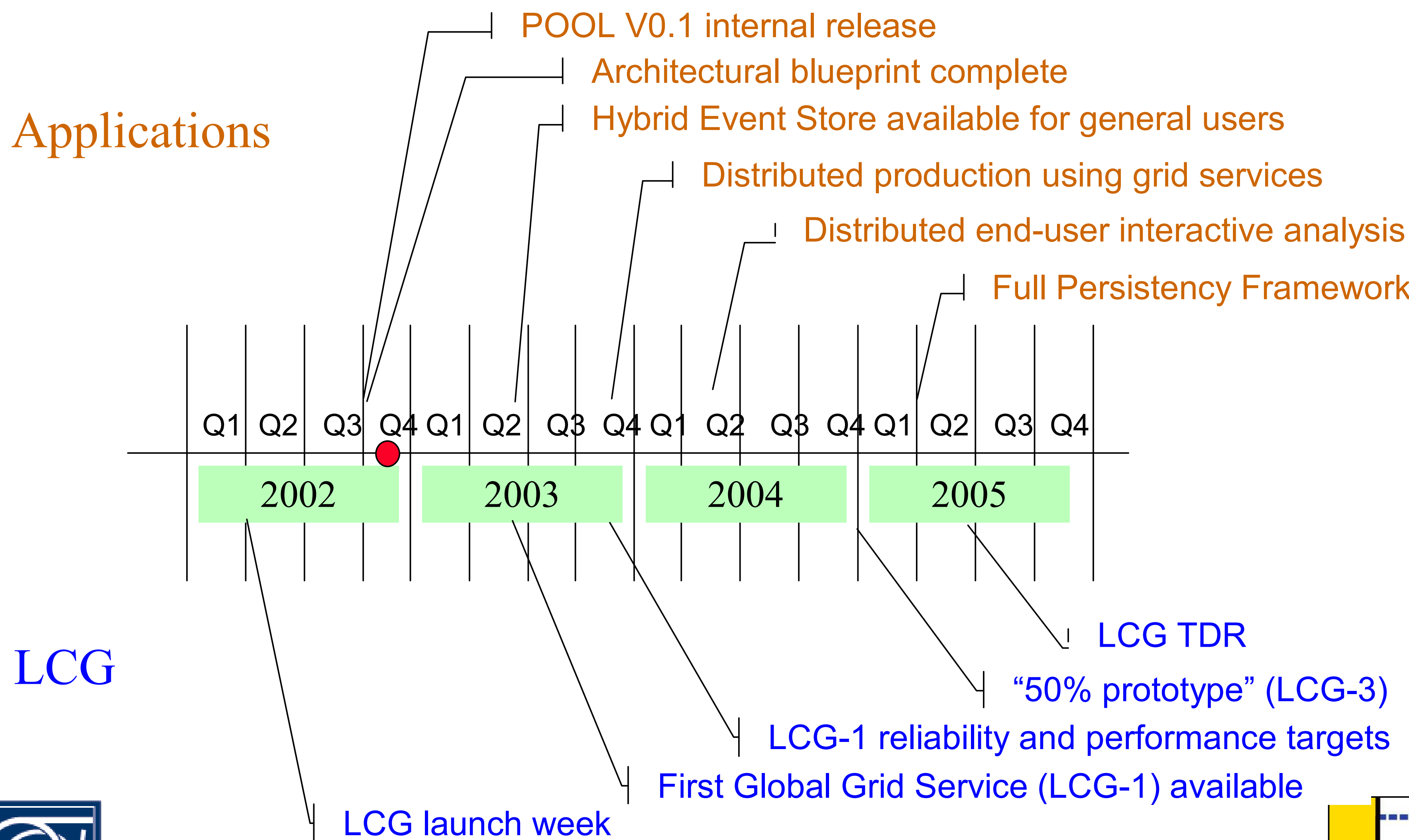
- Software Process and Infrastructure *(operating)*
  - Librarian, QA, testing, developer tools, documentation, training, …
- Persistency Framework *(operating)*
  - POOL hybrid ROOT/relational data store
- Mathematical libraries *(operating)*
  - Math and statistics libraries; GSL etc. as NAGC replacement
- Core Tools and Services *(just launched)*
  - Foundation and utility libraries, basic framework services, system services, object dictionary and whiteboard, grid enabled services
- Physics Interfaces *(being initiated)*
  - Interfaces and tools by which physicists directly use the software. Interactive (distributed) analysis, visualization, grid portals
- Simulation *(coming soon)*
  - Geant4, FLUKA, virtual simulation, geometry description & model, …
- Generator Services *(coming soon)*
  - Generator librarian, support, tool development

Candidate RTAG timeline from March

	02Q1	02Q2	02Q3	02Q4	03Q1	03Q2	03Q3	03Q4	04Q1	04Q2
Simulation tools		X								
Detector description & model		X								
Conditions database			X							
Data dictionary		X								
Interactive frameworks		X								
Statistical analysis			X							
Detector & event visualization				X						
Physics packages			X							
Framework services			X							
C++ class libraries			X							
Event processing framework								X		
Distributed analysis interfaces					X					
Distributed production systems				X						
Small scale persistency						X				
Software testing			X							
Software distribution				X						
OO language usage						X				
LCG benchmarking suite					X					
Online notebooks							X			

Blue: RTAG/activity launched or (light blue) imminent

LCG Applications Area Timeline Highlights



Personnel status

- 15 new LCG hires in place and working
- A few more starting over the next few months
- Manpower ramp is on schedule
- Contributions from UK, Spain, Switzerland, Germany, Sweden, Israel, Portugal, US
- Still working on accruing enough scope via RTAGs to employ this manpower optimally
  - But everyone is working productively

- ~10 FTEs from IT (DB and API groups) also participating
- ~7 FTEs from experiments (CERN EP and outside CERN) also participating, primarily in persistency project at present
- Important experiment contributions also in the RTAG process

Software Architecture Blueprint RTAG

- **Established early June 2002**
- **Goals**
  - Integration of LCG and non LCG software to build coherent applications
  - Provide the specifications of an architectural model that allows this, i.e. a ‘blueprint’
- **Mandate**
  - Define the main **domains** and identify the principal components
  - Define the **architectural relationships** between these ‘frameworks’ and components, identify the main requirements for their **inter-communication**, and suggest possible **first implementations.**
  - Identify the high level **deliverables** and their order of priority.
  - Derive a set of **requirements** for the LCG

RTAG Participants

- RTAG members
  - John Apostolakis, Guy Barrand, Rene Brun, Predrag Buncic, Vincenzo Innocente, Pere Mato, Andreas Pfeiffer, David Quarrie, Fons Rademakers, Lucas Taylor, Craig Tull, Torre Wenaus (Chair)
- Invited experts
  - Paul Kunz (SLAC), Tony Johnson (SLAC), Bob Jacobsen (LBNL), Andrea Dell'Acqua (CERN/ATLAS – Simulation RTAG)
- End-user readers of the draft
  - LHCb: Gloria Corti, Philippe Charpentier, Olivier Schneider
  - ATLAS: Fabiola Gianotti, Marge Shapiro
  - CMS: Paris Sphicas, Stephan Wynhoff, Norbert Neumeister
  - ALICE: Yves Schutz, Karel Safarik, Marek Kowalski

RTAG Meetings and Presentations

June 12	Pere Mato, Rene Brun
June 14	Rene Brun, Vincenzo Innocente
July 3	Torre Wenaus
July 8	Pere Mato, Vincenzo Innocente
July 12	Pere Mato, Vincenzo Innocente
July 23	Paul Kunz
August 5	Tony Johnson
August 6	Bob Jacobsen, Lassi Tuura
August 8	
August 9	Andrea Dell'Acqua
September 9	
September 10	
October 1	
October 7	Readers from the experiments

Response of the RTAG to the mandate

- Establish a high level 'blueprint' for LCG software which will provide sufficient **architectural guidance** for individual projects to ensure that LCG software
  - Conforms in its architecture to a coherent **overall architectural vision**
  - Makes consistent use of an identified set of **core tools, libraries and services**
  - **Integrates** well with other LCG software and experiment software
  - Functions in the **distributed** environment of the LCG
- Document a full **domain decomposition** of the applications area
- Clearly establish the **relationship between LCG software and ROOT**, and address the architectural implications
- Take account of the context: all four experiments have **established software** infrastructures and an existing user base requiring **continually functional** software
- Results will be presented and discussed in a **broad public meeting**

Architecture Blueprint RTAG Report

- Executive summary
- Response of the RTAG to the mandate
- Blueprint scope
- **Requirements**
- **Use of ROOT**
- **Blueprint architecture design precepts**
  - High level architectural issues, approaches
- **Blueprint architectural elements**
  - Specific architectural elements, suggested patterns, examples
- **Domain decomposition**
- **Schedule and resources**
- **Recommendations**

After 14 RTAG meetings,
much email...
A 36-page final report
Accepted by SC2 October 11

http://lcgapp.cern.ch/project/blueprint/BlueprintReport-final.doc
http://lcgapp.cern.ch/project/blueprint/BlueprintPlan.xls

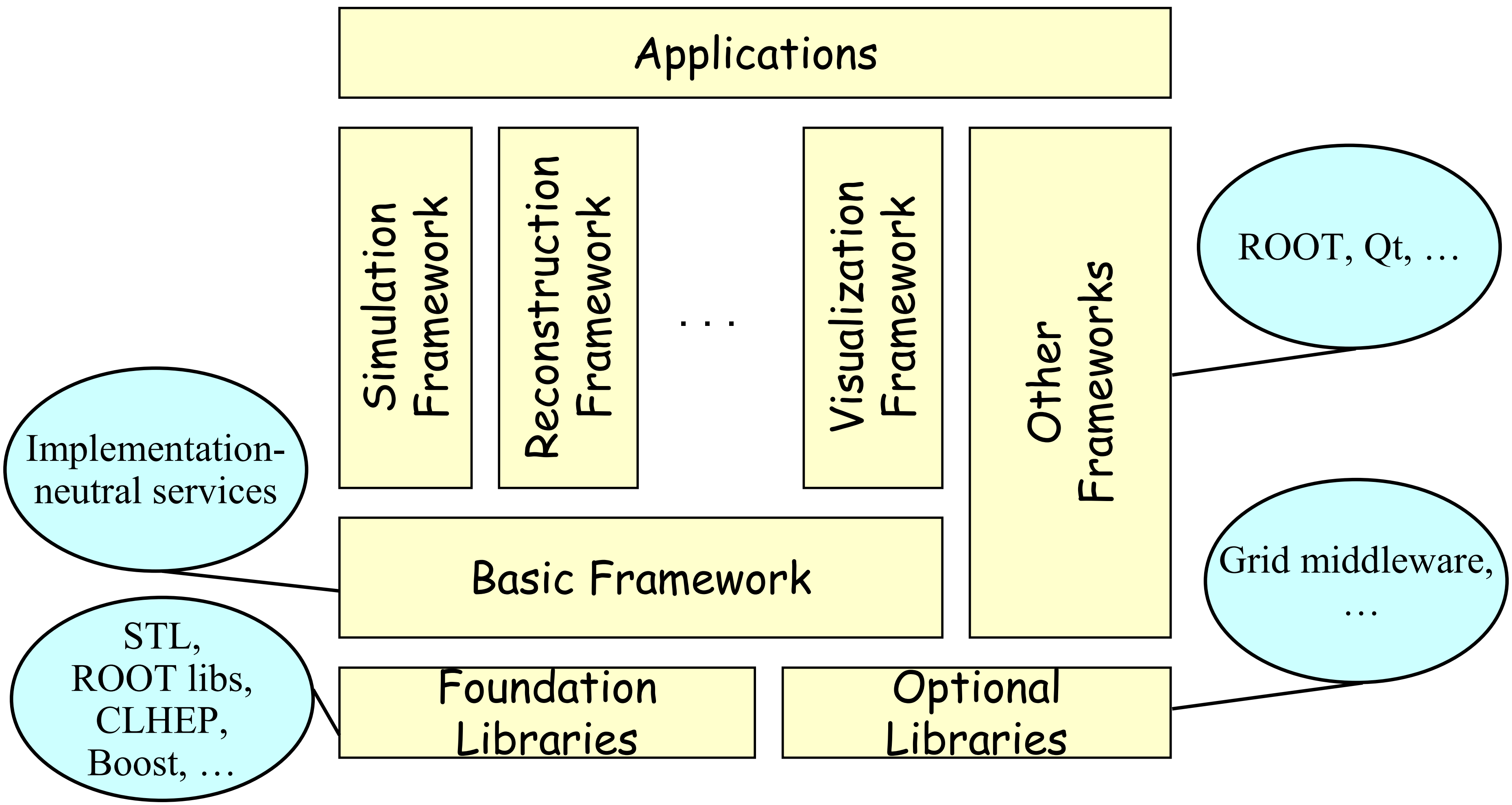
Architecture requirements

- Long lifetime: support technology evolution
- Languages: LCG core sw in C++ today; support language evolution
- Seamless distributed operation
- TGV and airplane work: usability off-network
- Modularity of components
- Component communication via public interfaces
- Interchangeability of implementations

Architecture Requirements (2)

- Integration into coherent framework and experiment software
- Design for end-user's convenience more than the developer's
- Re-use existing implementations
- Software quality at least as good as any LHC experiment
- Meet performance, quality requirements of trigger/DAQ software
- Platforms: Linux/gcc, Linux/icc, Solaris, Windows

Software Structure



Component Model

- *Addresses several requirements: modularity, communication, interchangeable implementation, integration, re-use, existing sw base*
- Granularity driven by component replacement criteria; development team organization; dependency minimization
- Communication via public interfaces
- Plug-ins
  - Logical module encapsulating a service that can be loaded, activated and unloaded at run time
- APIs targeted not only to end-users but to embedding frameworks and internal plug-ins
- Plug-in model is master/slave; foresee also peer-to-peer (e.g. peer components plugging into a common framework)

Component Model (2)

- Services
  - Components providing uniform, flexible access to basic framework functionality
  - E.g. a logical to physical filename mapping service may encapsulate distinct implementations appropriate to different circumstances: local lookup, grid lookup, fetch on demand, …
- Configuration
  - Coherent, scalable management of configurations
- Other issues
  - Composition vs. inheritance; interface versioning; component lifetime management (reference counting); class granularity

Distributed Operation

- Architecture should enable but not require the use of distributed resources via the Grid
- Configuration and control of Grid-based operation via dedicated services
  - Making use of optional grid middleware services at the foundation level of the software structure
    - Insulating higher level software from the middleware
    - Supporting replaceability
  - Apart from these services, Grid-based operation should be largely transparent
  - Services should gracefully adapt to ‘unplugged’ environments
    - Transition to ‘local operation’ modes, or fail informatively

Distributed Character of Components

- POOL
  - Naming based on logical filenames
  - Replica catalog and management
- Interactive frameworks
  - Grid-aware environment; ‘transparent’ access to grid-enabled tools and services
- Statistical analysis, visualization
  - Integral parts of distributed analysis environment
- Framework services
  - Grid-aware message and error reporting, error handling, grid-related framework services
- Distributed production tools, portals

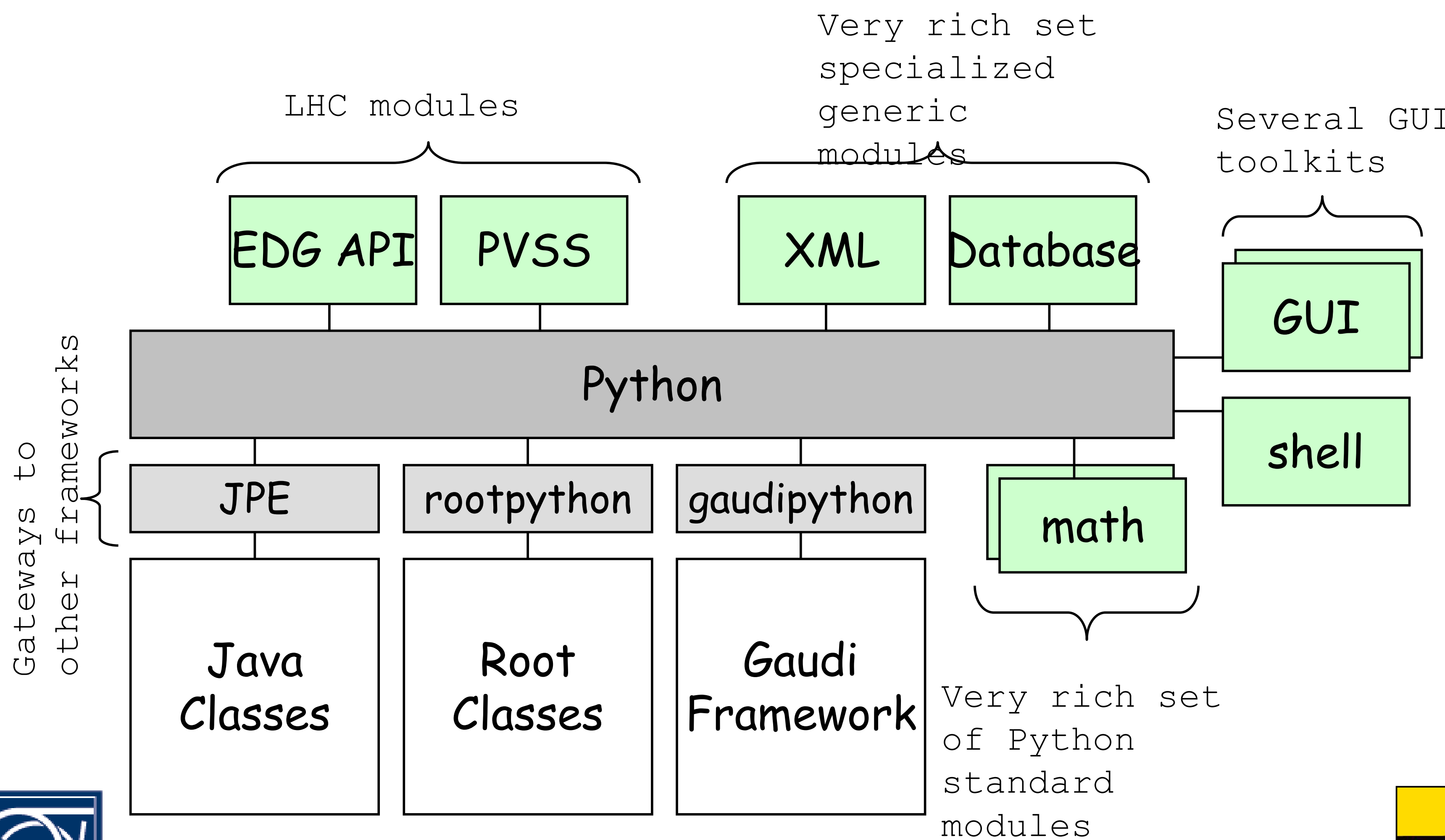
Other Design Elements

- Object model
  - Support both ‘dumb’ (data) and ‘smart’ (functional) objects
  - Clear and enforced object ownership
- Role of abstract interfaces (pure virtual classes)
  - ‘Documents’ an API used by multiple implementations
  - Use (only) where appropriate; e.g. services-yes, data-no
- Dependency minimization
- Exception handling via C++ exceptions
  - Graceful and informative handling required
- Interface to external components via generic adapters
- Identify metrics measuring quality, usability, maintainability, modularity

Managing Objects

- Object Dictionary
  - To query a class about its internal structure (Introspection)
  - Essential for persistency, data browsing, interactive rapid prototyping, etc.
  - The ROOT team and LCG plan to develop and converge on a *common dictionary* (common interface and implementation) with an interface anticipating a C++ standard (XTI)
    - To be used by LCG, ROOT and CINT
    - Timescale ~1 year
- Object Whiteboard
  - Uniform access to application-defined transient objects

Python as a "Component Bus"



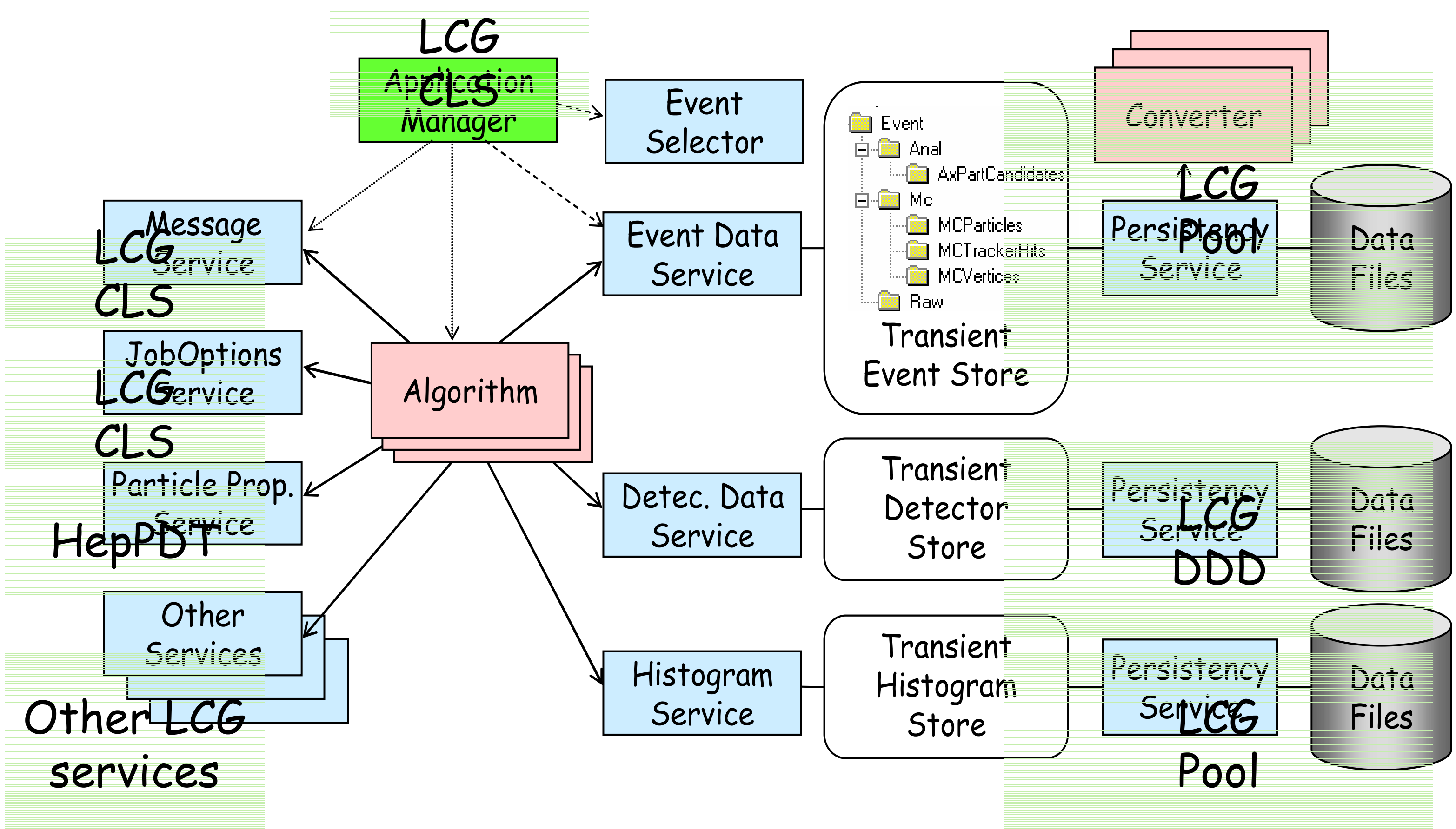
Basic Framework Services

- Component/Plug-in management
- Factories (object creation), Registries (discovery)
- Component configuration
- Smart pointers
- Incident ('event') management
- Monitoring and reporting
- GUI manager
- Exception handling
- Consistent interface to system services

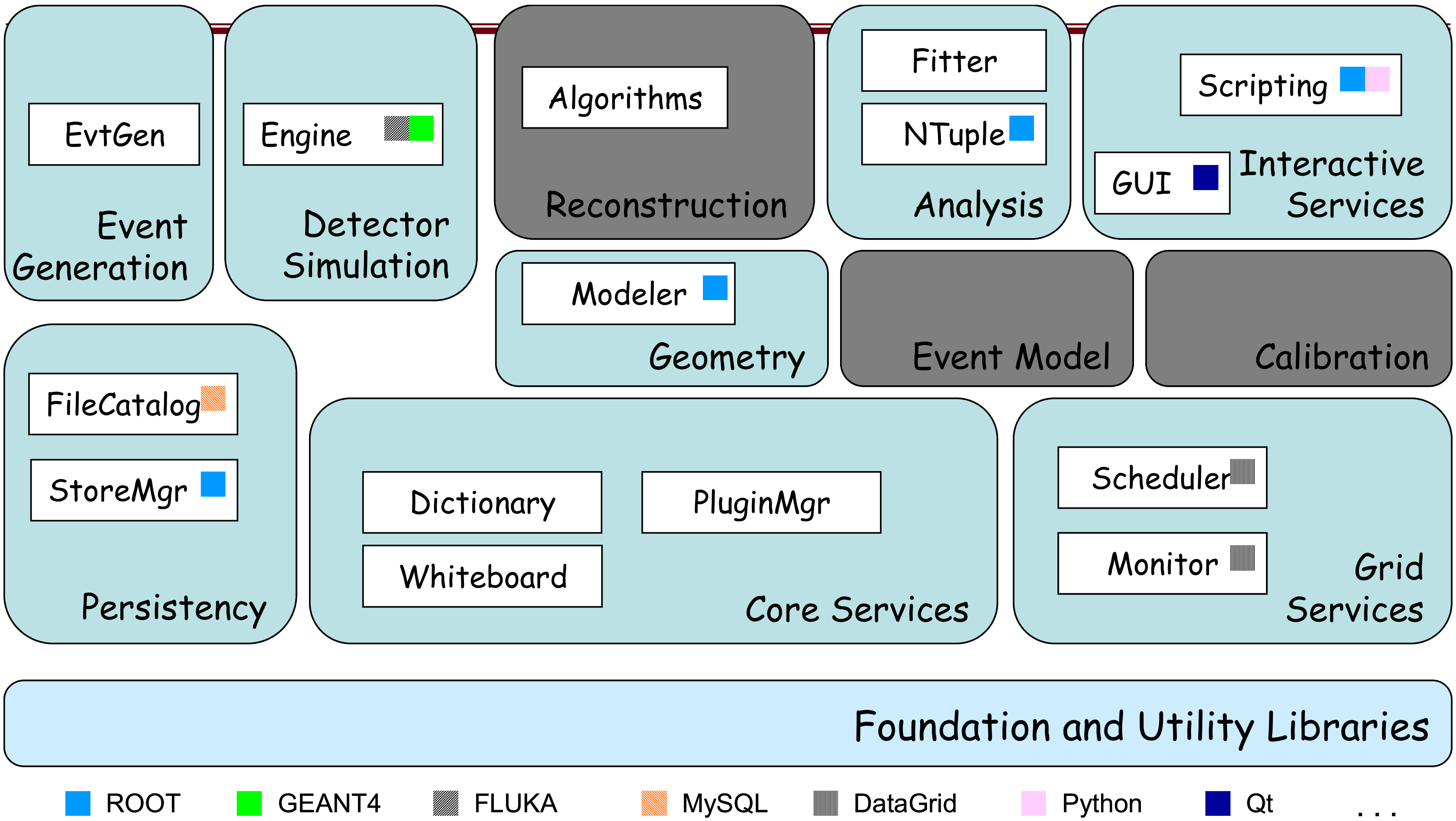
Other Architectural Elements

- Component Bus
  - Plug-in integration of components providing a wide variety of functionality
  - Component interfaces to bus derived from their C++ interfaces
- Scripting Language
  - Python and CINT (ROOT) to both be available
  - Access to objects via object whiteboard in these environments
- Interface to the Grid
  - Must support convenient, efficient configuration of computing elements with all needed components

(LHCb) Example of LCG–Experiment SW Mapping



Domain Decomposition



Products mentioned are examples; not a comprehensive list

Grey: not in common project scope
(also event processing framework, TDAQ)

Use of ROOT in LCG Software

- Among the LHC experiments
  - ALICE has based its applications directly on ROOT
  - The 3 others base their applications on components with implementation-independent interfaces
    - Look for software that can be encapsulated into these components
- All experiments agree that ROOT is an important element of LHC software
  - Leverage existing software effectively and do not unnecessarily reinvent wheels
- Therefore the blueprint establishes a **user/provider relationship** between the LCG applications area and ROOT
- Will draw on a great ROOT strength: users are listened to very carefully!
  - The ROOT team has been *very* responsive to needs for new and extended functionality coming from the persistency effort

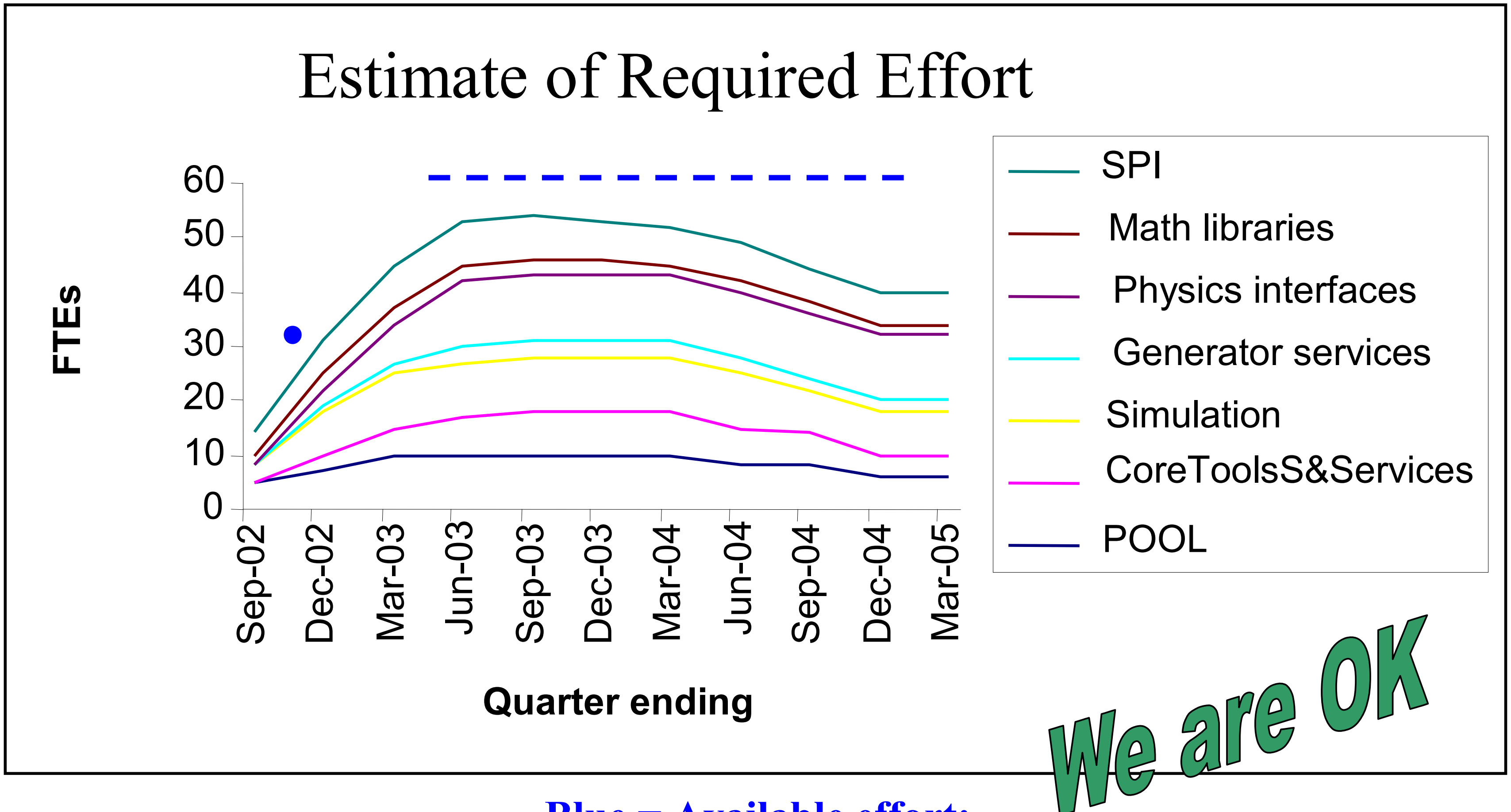
ROOT in the LCG Blueprint Architecture

- User/provider relationship should allow experiments to take full advantage of the capabilities of ROOT
  - Implementation technology choices will be made on case by case basis
- Strong CERN support for ROOT
  - Fully staffed ROOT section in new LCG-focused EP/SFT group
- We can expect that LCG software may place architectural, organizational or other demands on ROOT
  - e.g. to ensure no interference between components used in different domains
- For specific components we can expect that different implementations will be made available
  - e.g. CINT and Python will both be available, to exploit complementary capabilities

ROOT Development Areas of LCG Interest

- Support for ‘foreign classes’ in ROOT I/O
- Locatable persistent references
- Expanding support for STL
- Convenient external access to CINT dictionary (feed/retrieve info)
- Eventual migration to a standard C++ introspection interface
- Support for automatic schema evolution
- PROOF and grid interfacing
- Interfaces to Python and Java
- Qt based GUI
- Histogramming, fitting, n-tuples, trees
- Interface to GSL

Personnel Resources – Required and Available



Blue = Available effort:

FTEs today: 15 LCG, 10 CERN IT, 7 CERN EP + experiments

Future estimate: 20 LCG, 13 IT, 28 EP + experiments

RTAG Conclusions and Recommendations

- **Use of ROOT** as described
- **Start common project on core tools and services**
- **Start common project on physics interfaces**
- **Start RTAG on analysis**, including distributed aspects
- Tool/technology recommendations
  - CLHEP, CINT, Python, Qt, AIDA, …
- Develop a clear process for adopting third party software

Core Libraries and Services Project

- **Pere Mato (CERN/LHCb) is leading the new Core Libraries and Services (CLS) Project**
- Project being launched now, developing immediate plans over the next week or so and a full work plan over the next couple of months
- Scope:
  - Foundation, utility libraries
  - Basic framework services
  - Object dictionary
  - Object whiteboard
  - System services
  - Grid enabled services
- Many areas of immediate relevance to POOL
- Clear process for adopting third party libraries will be addressed early in this project

Physics Interfaces Project

- Expect to launch this project very soon
- Covers the interfaces and tools by which physicists will directly use the software
- Should be treated coherently, hence coverage by a single project
- Expected scope once analysis RTAG concludes:
  - Interactive environment
  - Analysis tools
  - Visualization
  - Distributed analysis
  - Grid portals

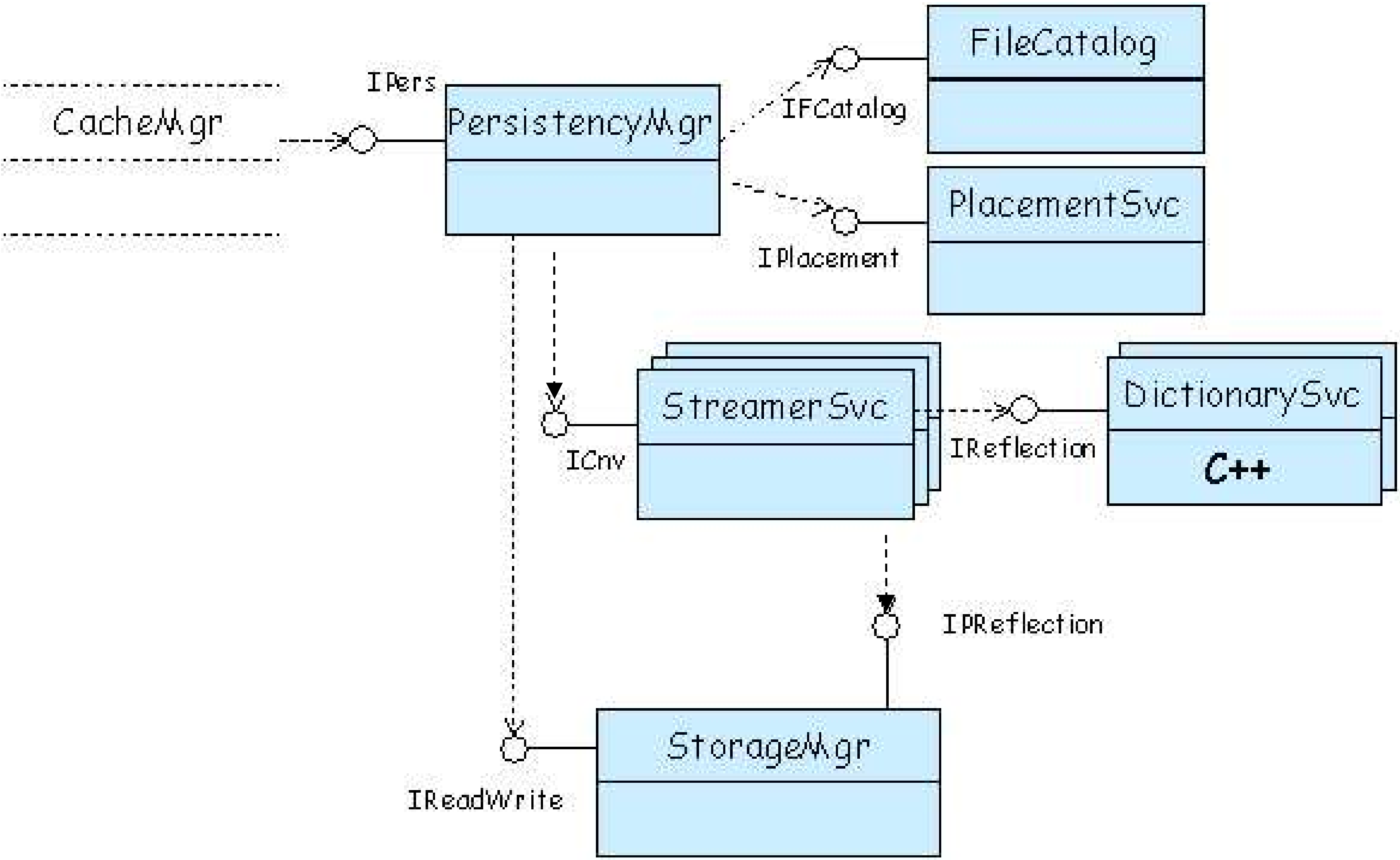
Analysis RTAG

- LHC analysis environment & tools including distributed aspects
- Get LCG involved now to bring some coherence to the already diverse efforts
- Expectation: RTAG launch before end of year
  - In the hands of the SC2

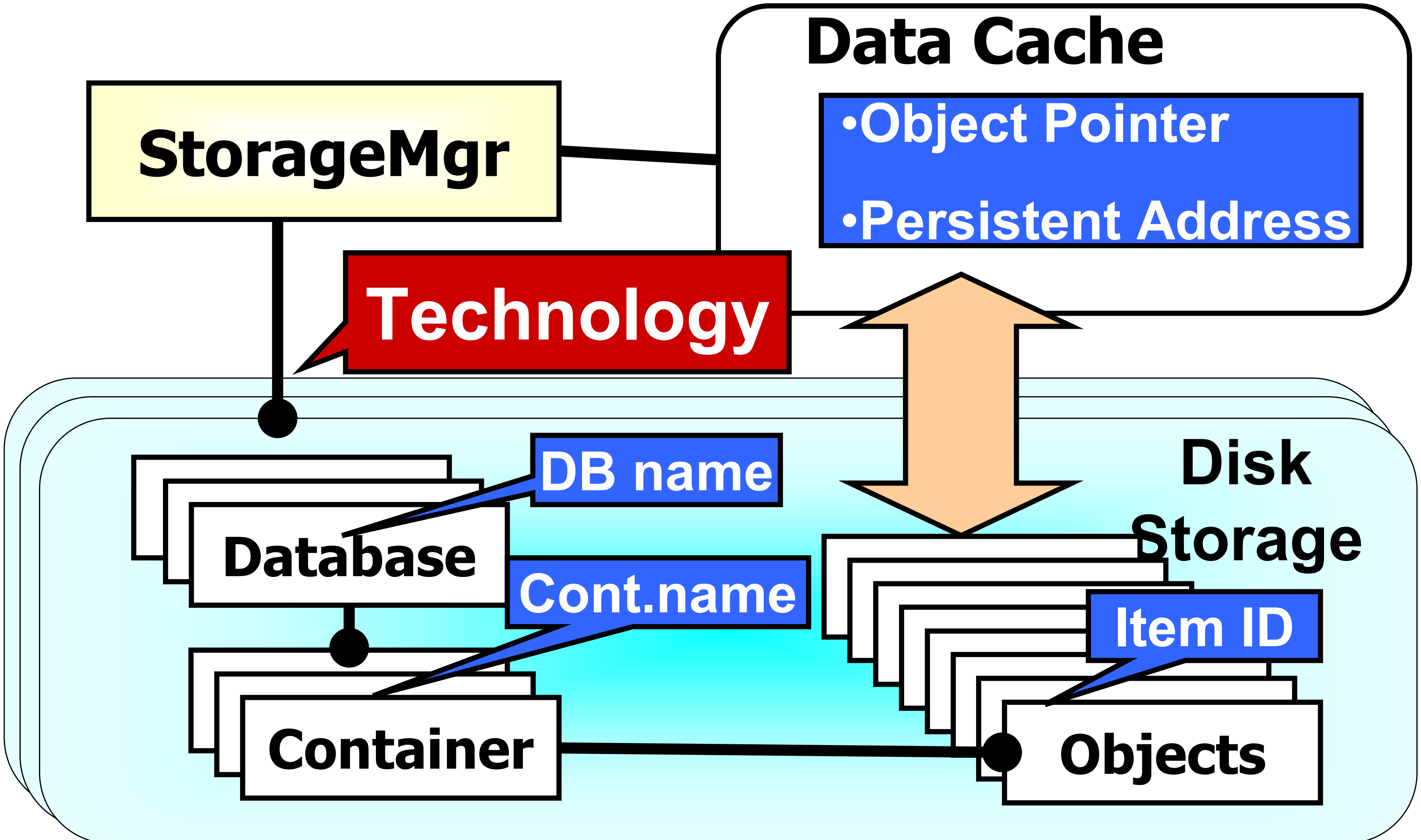
POOL and the Blueprint

- Pool of persistent objects for LHC, currently in prototype
  - Targeted at event data but not excluding other data
  - Hybrid technology approach
    - Object level data storage using file-based object store (ROOT)
    - RDBMS for meta data: file catalogs, object collections, etc (MySQL)
  - Leverages existing ROOT I/O technology and adds value
    - Transparent cross-file and cross-technology object navigation
    - RDBMS integration
  - Integration with Grid technology (eg EDG/Globus replica catalog)
    - network and grid decoupled working modes
- Follows and exemplifies the LCG blueprint approach
    - Components with well defined responsibilities
    - Communicating via public component interfaces
    - Implementation technology neutral

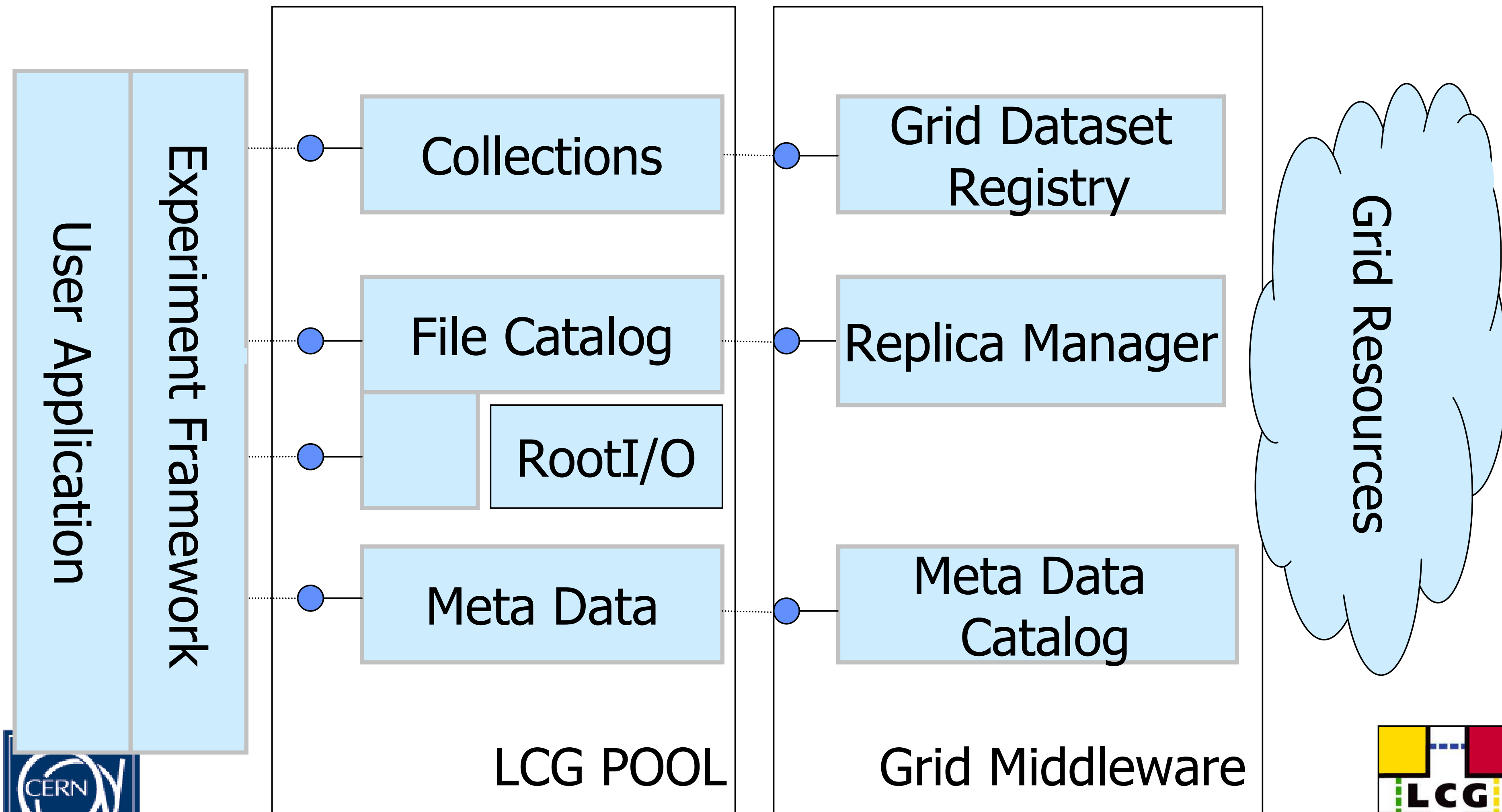
POOL Components



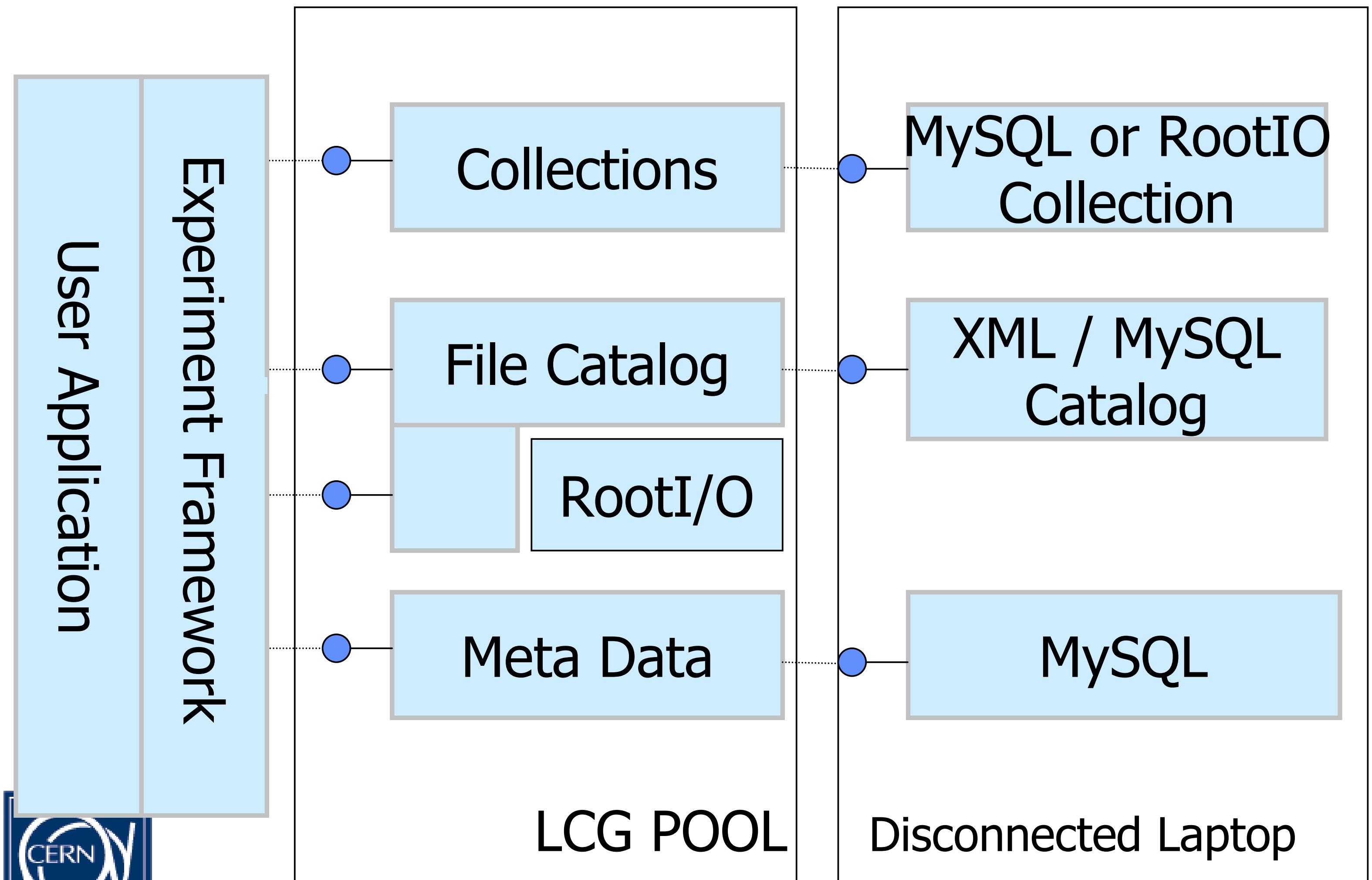
Data Storage on the Object Store



Pool on the Grid



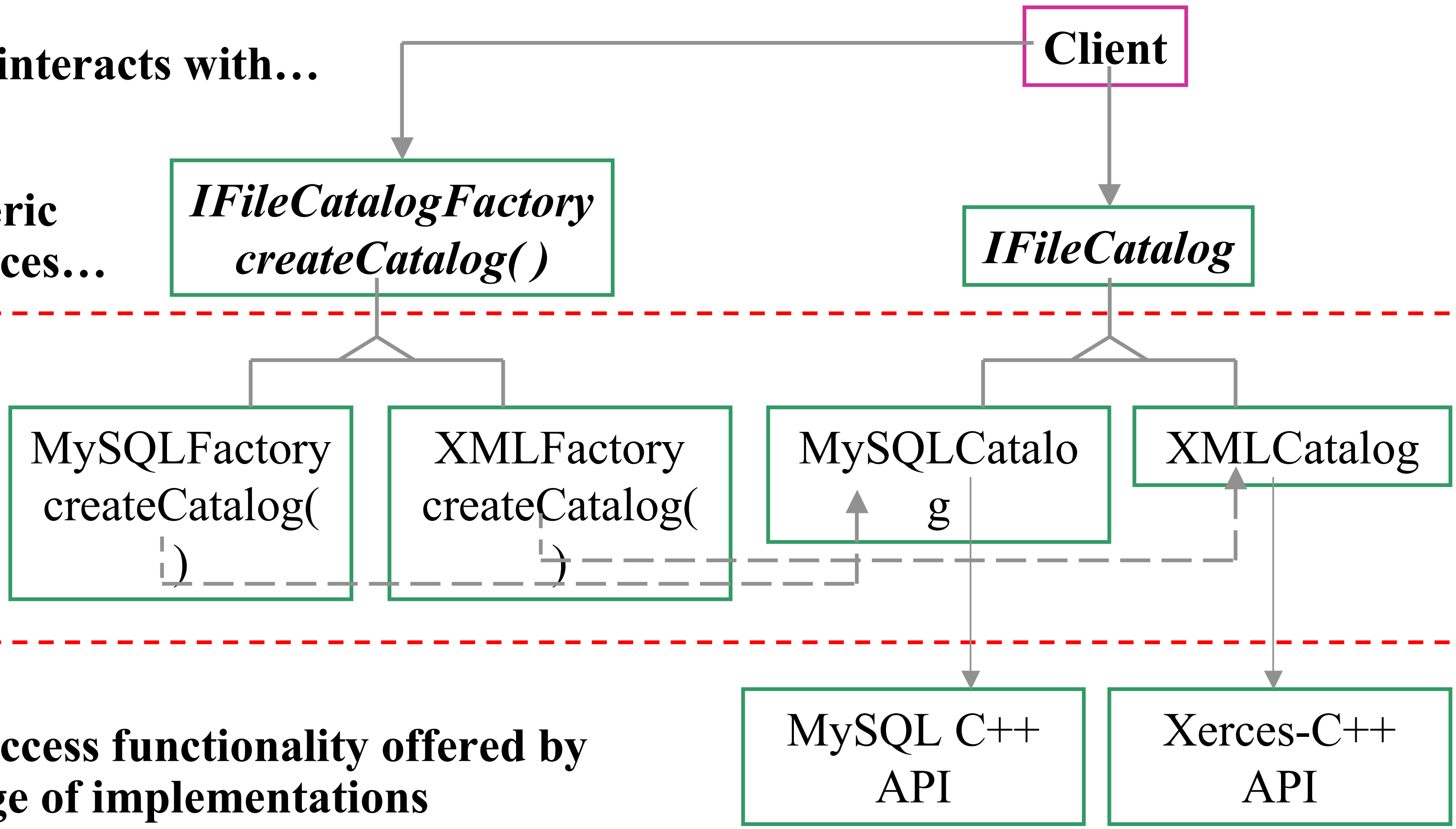
Pool off the Grid



Diverse Implementations via Generic Interfaces

Client interacts with…

…generic interfaces…



…to access functionality offered by a range of implementations

Pool Release Schedule

- End September - V0.1 **(Released on schedule)**
  - All core components for navigation exist and interoperate
  - Assumes ROOT object (TObject) on read and write
- End October - V0.2
  - First collection implementation
- End November - V0.3 **(First public release)**
  - EDG/Globus FileCatalog integrated
  - Persistency for general C++ classes (not instrumented by ROOT)
  - Event meta data annotation and query
- June 2003 – Production release

Four Experiments, Four Viewpoints, Two Paths

- Four viewpoints (of course) among the experiments; two basic positions
  - ATLAS, CMS, LHCb similar views; ALICE differing
- The blueprint report establishes the basis for a good working relationship among all
  - LCG applications software is developed according to the blueprint
    - To be developed and used by ATLAS, CMS and LHCb, with ALICE contributing mainly via ROOT
    - Making use of ROOT in a user/provider relationship
  - ALICE continues to develop their line making direct use of ROOT as their software framework
    - Of which you will hear more tomorrow!

Concluding Remarks

- The LCG fosters and hosts cooperative development of common LHC physics applications software
- The LCG Architecture Blueprint
  - establishes how LCG software should be developed to facilitate integration of components developed by different parties across a long span of time
  - takes a component approach that is widely used and practical
  - recognizes and defines a special role for ROOT
  - has all experiments as participants and signatories.
  - is directed at meeting the real needs of users in the experiments
- The first LCG software deliverable, POOL, is on schedule, and is a good example of
  - the blueprint being realized in software
  - the LCG/ROOT 'working relationship' in action

Another component software example

- http://www.openoffice.org/white_papers/tech_overview/tech_overview.html
- A technical overview of Sun's open source office suite (formerly StarOffice)
- Quoting it:
- *The OpenOffice.org suite employs a component-based development system that exemplifies all the important characteristics of Component Ware…*
- *OpenOffice.org's component technology is open, object oriented, interface based, and independent of both platform and development system.*
- *The OpenOffice.org API is version independent, scalable, durable, and re-applicable.*
- *Because the component technology is used in its implementation, the OpenOffice.org API is programming language independent.*