ATLAS Metadata Interface

Looking back on 10 years of the ATLAS Metadata Interface. Reflections on architecture, code design and development methods.

CHEP 2013

AMSTERDAM

J.Fulachier, O.Aidel, S.Albrand, F.Lambert

introduction

- development of the **ATLAS Metadata Interface (AMI)** framework
- part of the offline software of the **ATLAS experiment at CERN**
- designed for **generic cataloguing** using **relational databases**
- basis of two tools:
  - **Dataset Discovery**: principally to **catalogue** the **datasets** available for **analysis**
  - **TagCollector**: highly specialized application, part of the **management** of the various releases of **ATLAS software**
- we will concentrate on the history, architecture and evolution of the project

history

- the first **prototype** of AMI was an **electronic bookkeeping** application for the **ATLAS** Liquid Argon detector component in **2001**
- in parallel, the Grenoble team was working on **TagCollector**, a rather different application for **release management**
- both applications **used databases**, both had **web interfaces** and they would be both used by **ATLAS collaboration members**
- much of their underlying software could **become common**

history

- AMI Framework, designed and developed between 2002 and 2004
- 2006: **review** by the ATLAS collaboration
  - framework was chosen for **physics metadata catalogues**
- starting point for the expansion of the **Dataset Discovery** application
- **TagCollector** has also been considerably enhanced
  - increasing complexity of ATLAS releases

architecture and technology

- **architectural and technological choices**
  - context of very **large** and **widely distributed** scientific community over at least **a decade**
- **we chose to implement a central web service used by lightweight clients**
- **Java/Apache Tomcat** or **XML/SOAP** have become well supported standards

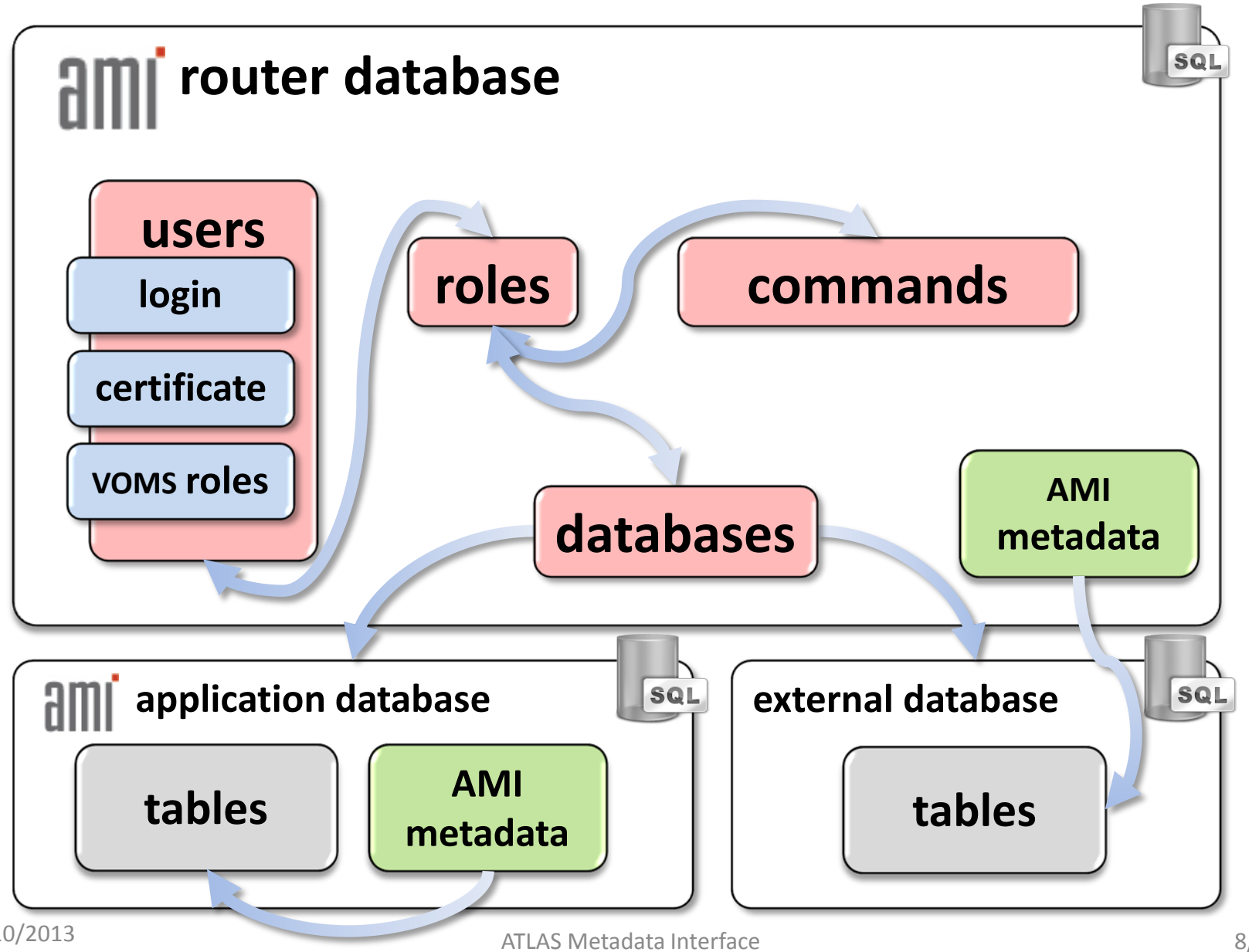
architecture and technology

- AMI has **a multi-tier client–server** architecture
- layers are logically separated
  - presentation
  - application processing
  - data management functions
- **Open-Closed Principle** (OCP)
- **dependency injections**
  - "**inversion of control**" design patterns
- **JAVA**: **server side software**
  - encapsulation
    - limits overall system complexity
    - increases robustness
  - limit the inter-dependencies between software components

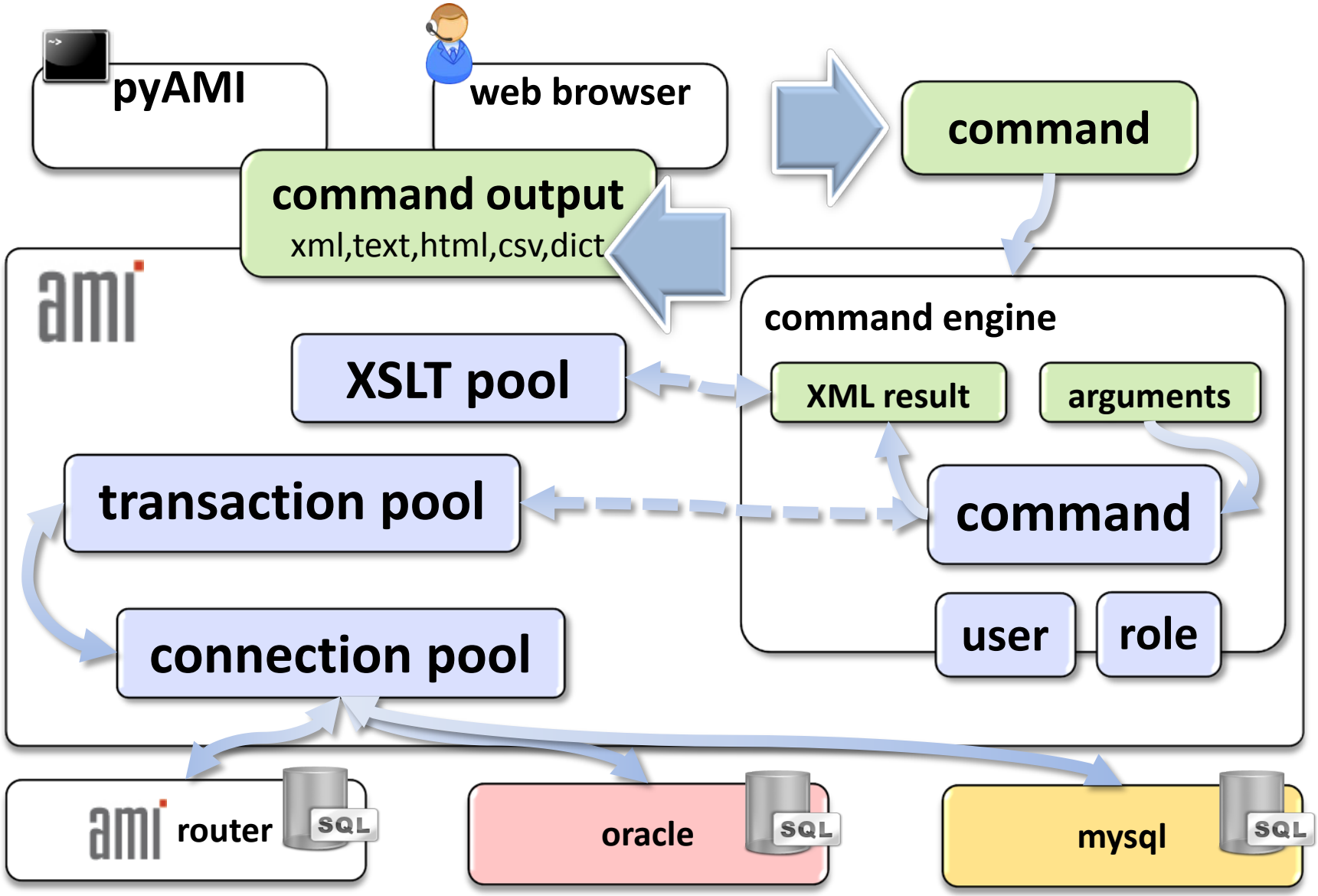
architecture and technology

- **architectural decisions** which have given the framework both **strength** and **flexibility**

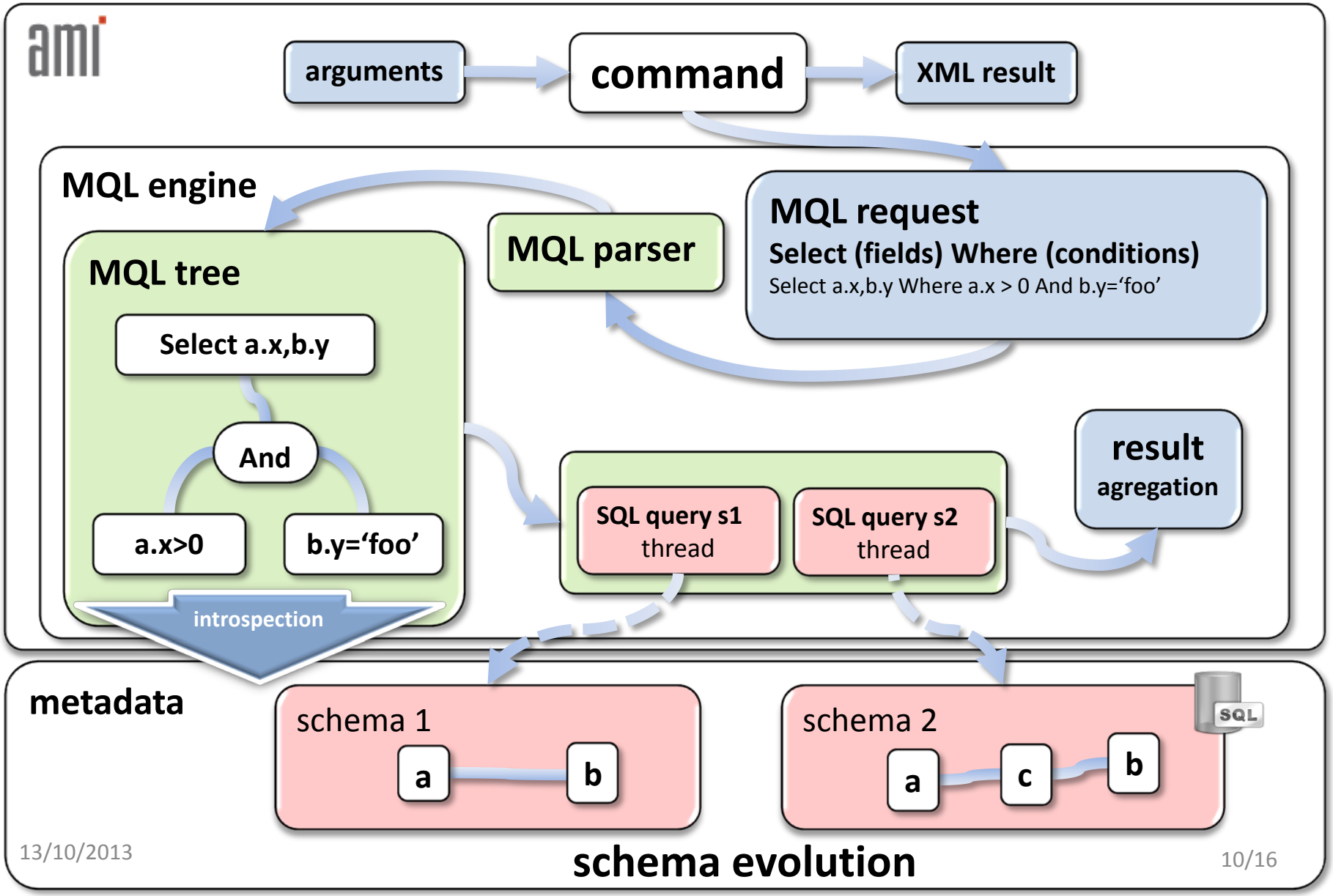
users and roles



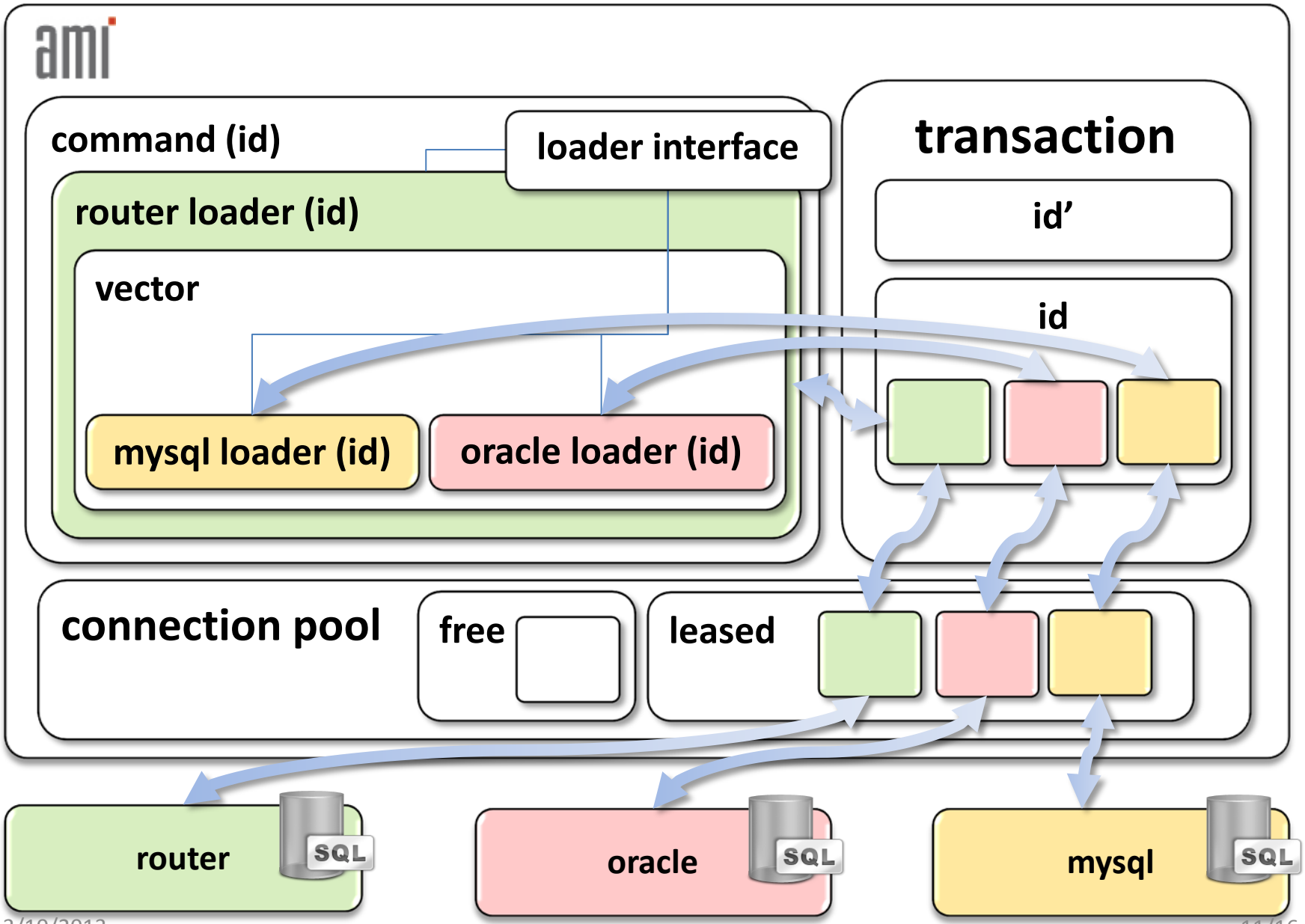
commands and layers



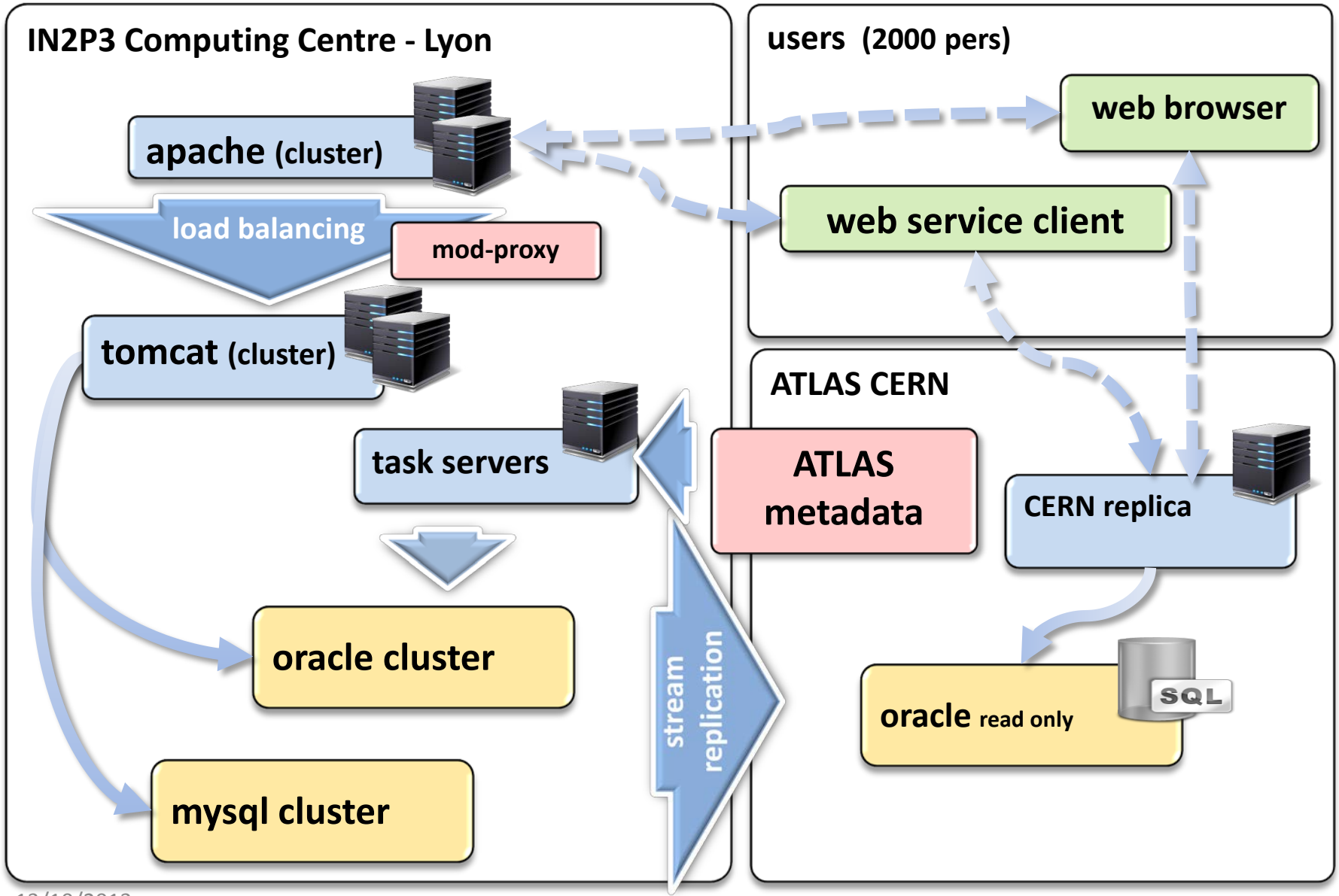
metadata query language



connections and transactions



production environment



development methods and tools

- **Agile model**
  - extend development responsiveness
  - reduce maintenance cost
  - well adapted to **a project with a long life cycle** and frequent feature requests

- some important points are:
  - avoiding programming of features until they are actually needed
  - programming in pairs
  - including unit testing

development methods and tools

- **standard tools** often **linked** to one another:

Subversion (SVN)	code repository & version control
Eclipse	integrated development environment
Firebug	web development tool (JavaScript debugger)
Twitter Bootstrap	CSS framework for the application web pages
Sphinx	pyAMI documentation generation
Redmine	project management
Jenkins	continuous integration & deployment
Joomla	content management – used for the portal pages.

- **project management: keeping track** of all **issues** (bugs and feature requests) and to have **a global vision** of the **future evolution**

- **deployment procedure: linking** the **resolution** of bugs or **introduction** of new features with code and release **versions**

future and prospective

- the **LHC** is currently **stopped** for a major upgrade, and is scheduled to restart at the beginning of 2015

- during this time many operations will undergo major changes

  - the **ATLAS production system** (main source of metadata)
    - is being **rewritten**

  - **redefinition** of the **ATLAS dataset nomenclature**
    - repository of reference tables
    - software configurations

  - new requests for very specific metadata functions

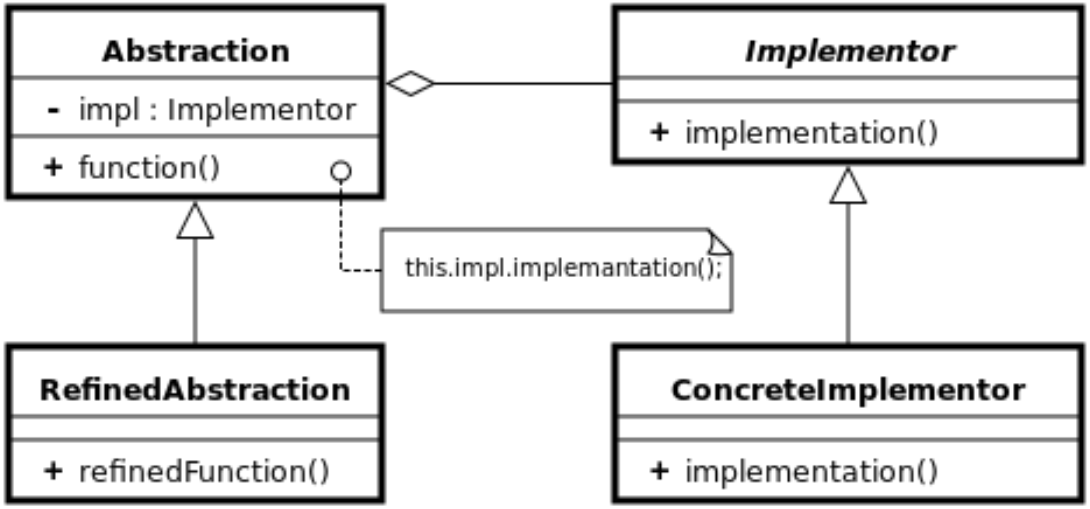
future and prospective

- **TagCollector** web interface **review**
  - implementation of a complete **new version (AJAX)**
- we **do not foresee changes to the lower level AMI framework** beyond some optimization of code
- greatly improved **performance monitoring tools**
- better **package the AMI framework**
  - easily **shared with other experiments**

AMI

extra slides

bridge/plugin pattern

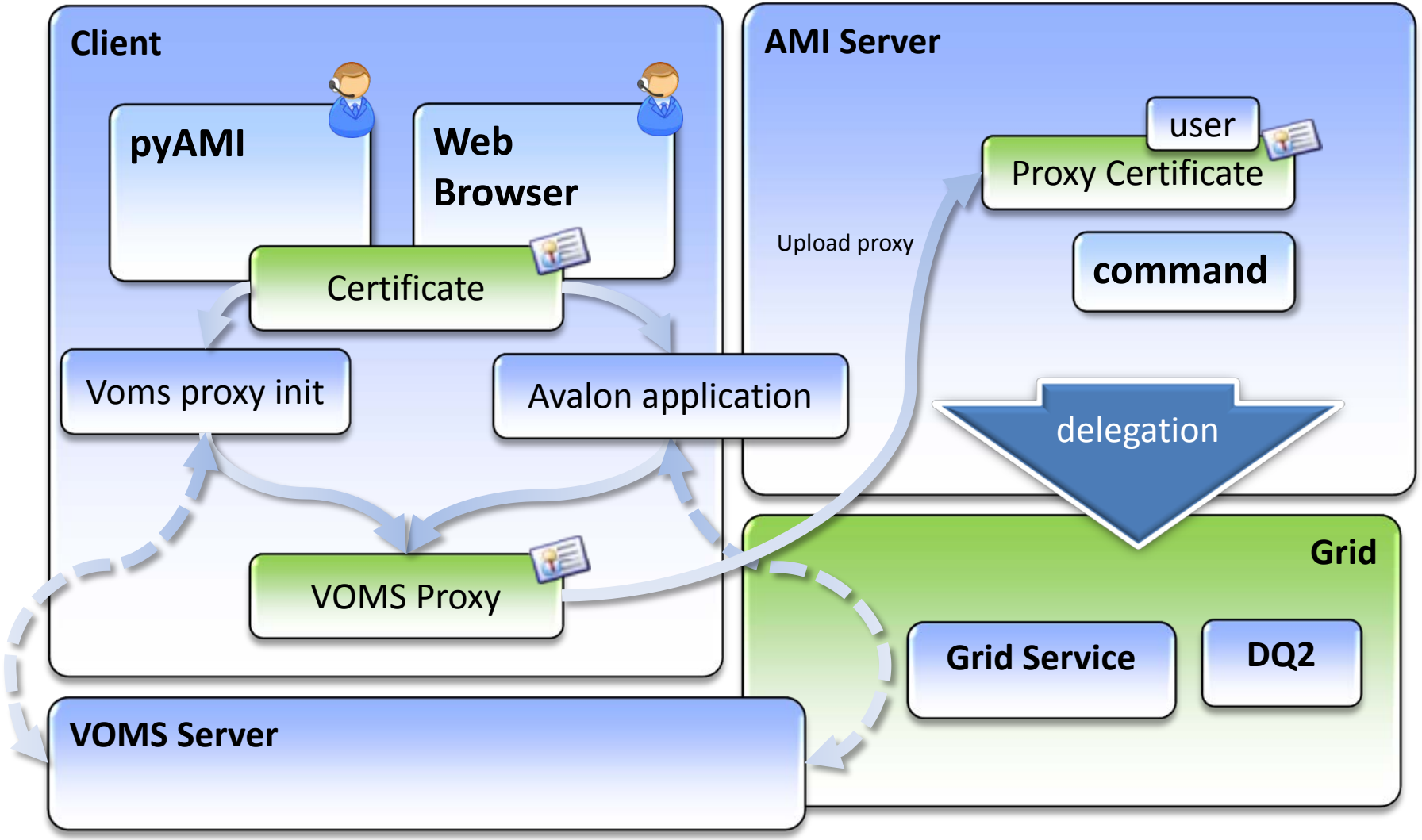


- **Abstraction**
  - defines the abstract interface
  - maintains the Implementor reference
- **RefinedAbstraction**
  - extends the interface defined by Abstraction
- **Implementor**
  - defines the interface for implementation classes
- **ConcreteImplementor**
  - implements the Implementor interface

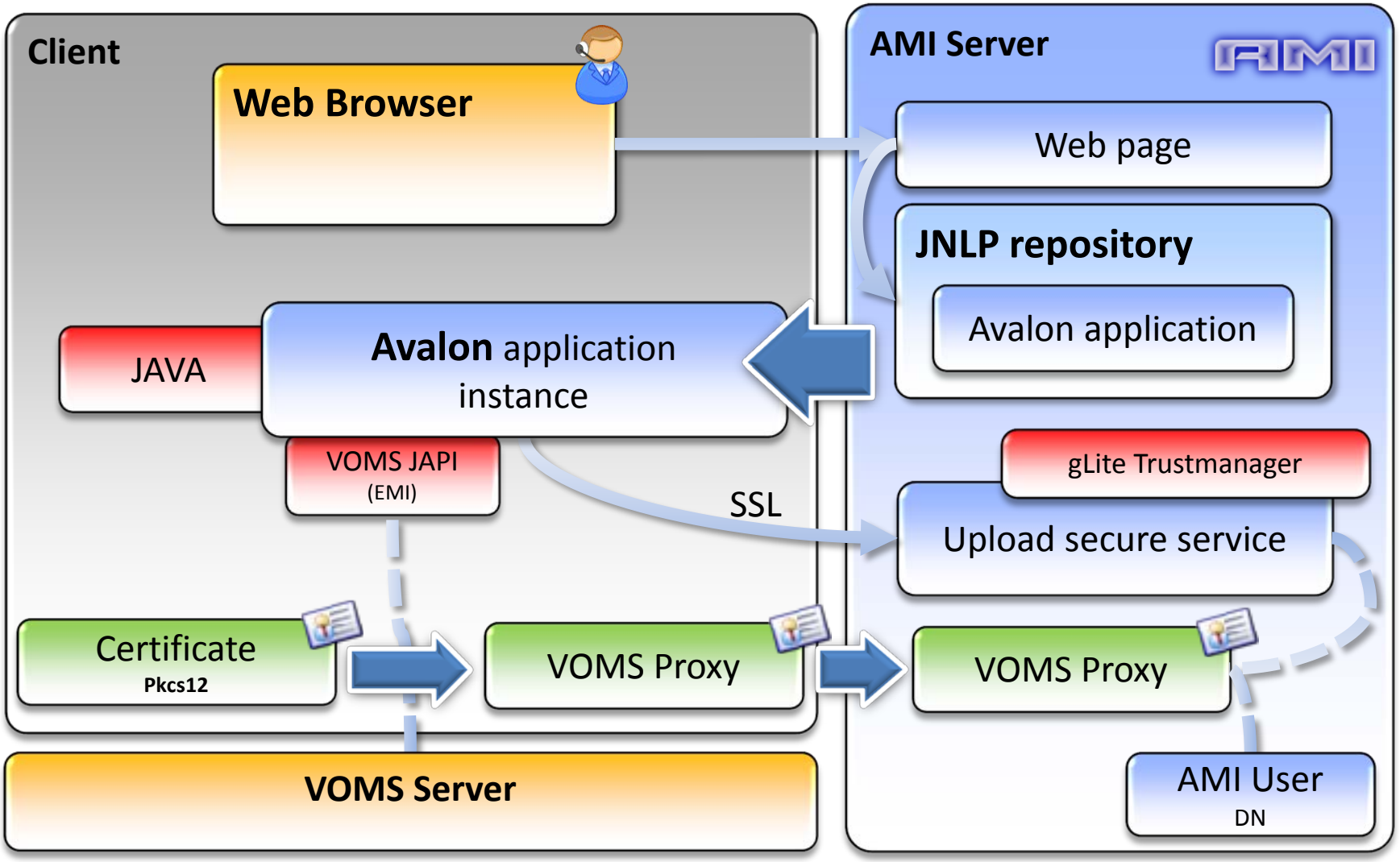
users and roles

- the users are mapped to a set of roles
- each role is linked to a set of commands
- the command execution may be limited to a sub-set of catalogues
- users must possess a certificate known to VOMS, but as long as these credentials remain valid they may also authenticate with a username and password
- AMI also has a method for VOMS proxy forwarding, so that GRID middleware operations can be performed by AMI for users

proxy delegation



avalon – proxy management



commands and layers

- gives an overview of the interaction of AMI layers
- client commands arrive either from a browser or the python client
- after user authentication and authorization the command is executed by the command engine, possibly using several RDBMS through the transaction and the connections pools
- connections are made using the physical connection parameters in the router

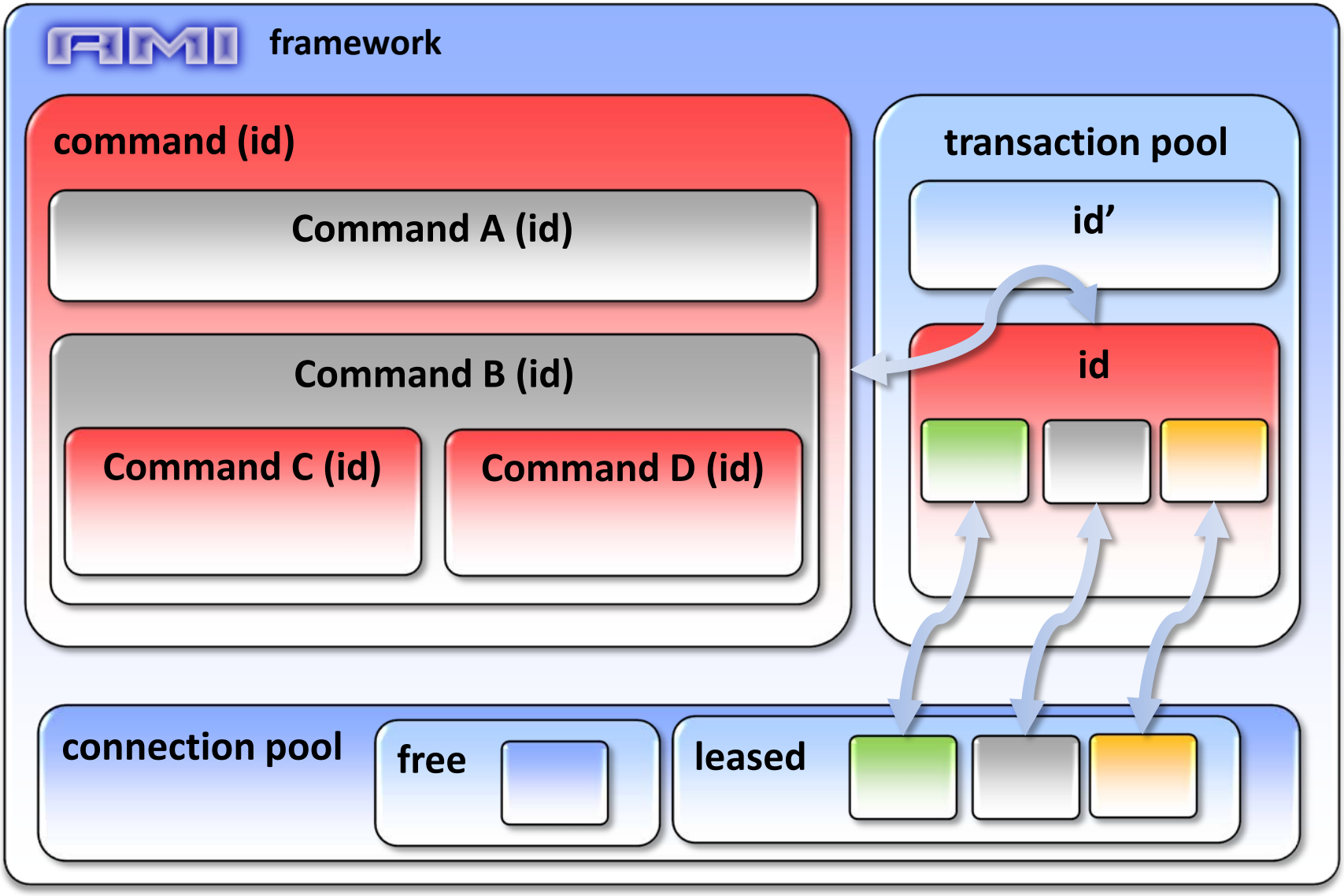
metadata query language

- outlines the transformation of a query in MQL to catalogue specific SQL queries
- using database schema introspection
- the results are aggregated to produce a simple row data output
- this mechanism can be used as an abstraction layer in case of database schema evolution

connections and transactions

- transaction pool supporting a set of transactions
- each AMI transaction manages a pool of RDBMS connections and ensures atomicity at the command level
- a command can contain calls to other commands belonging to the same transaction

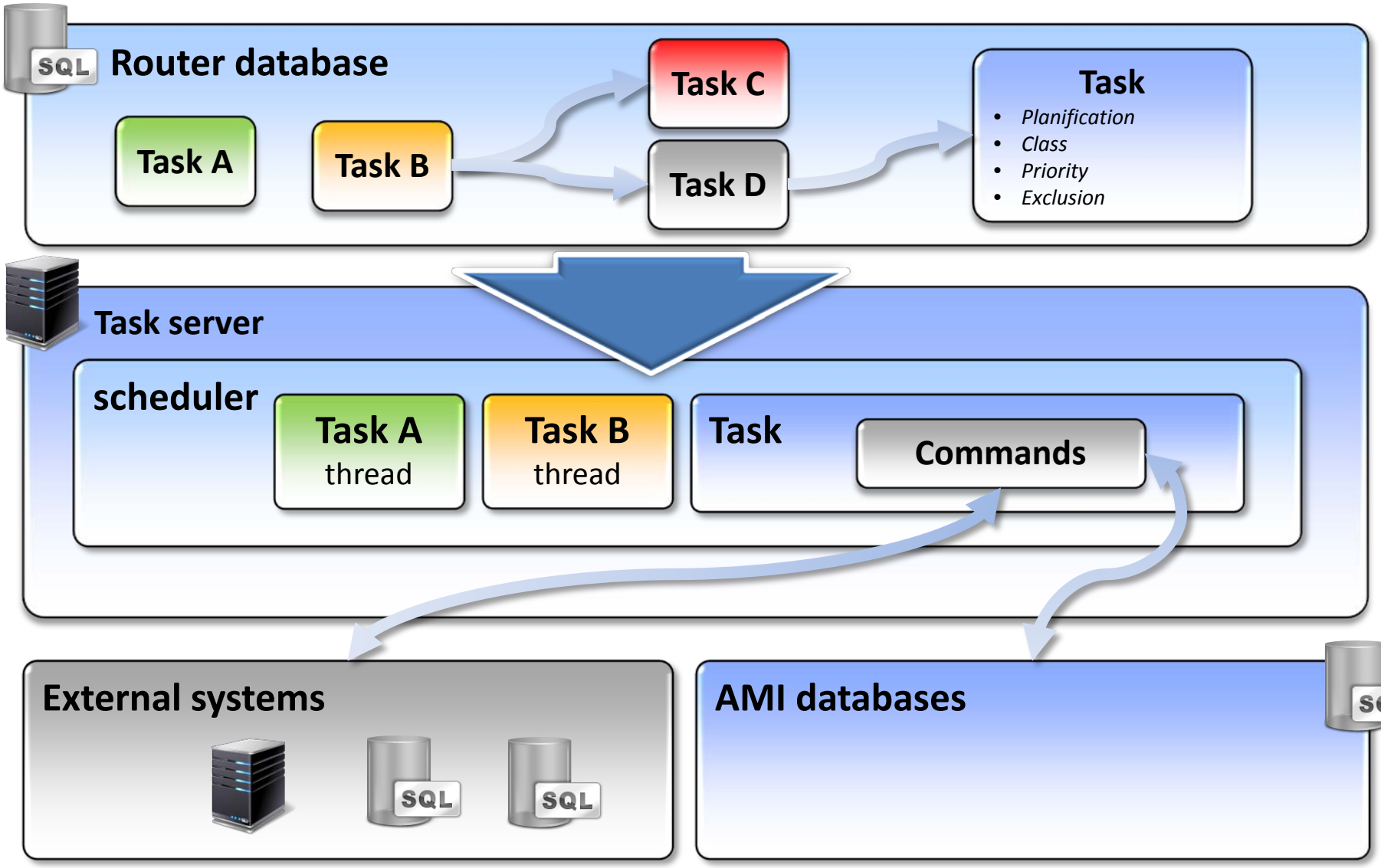
connections - transactions



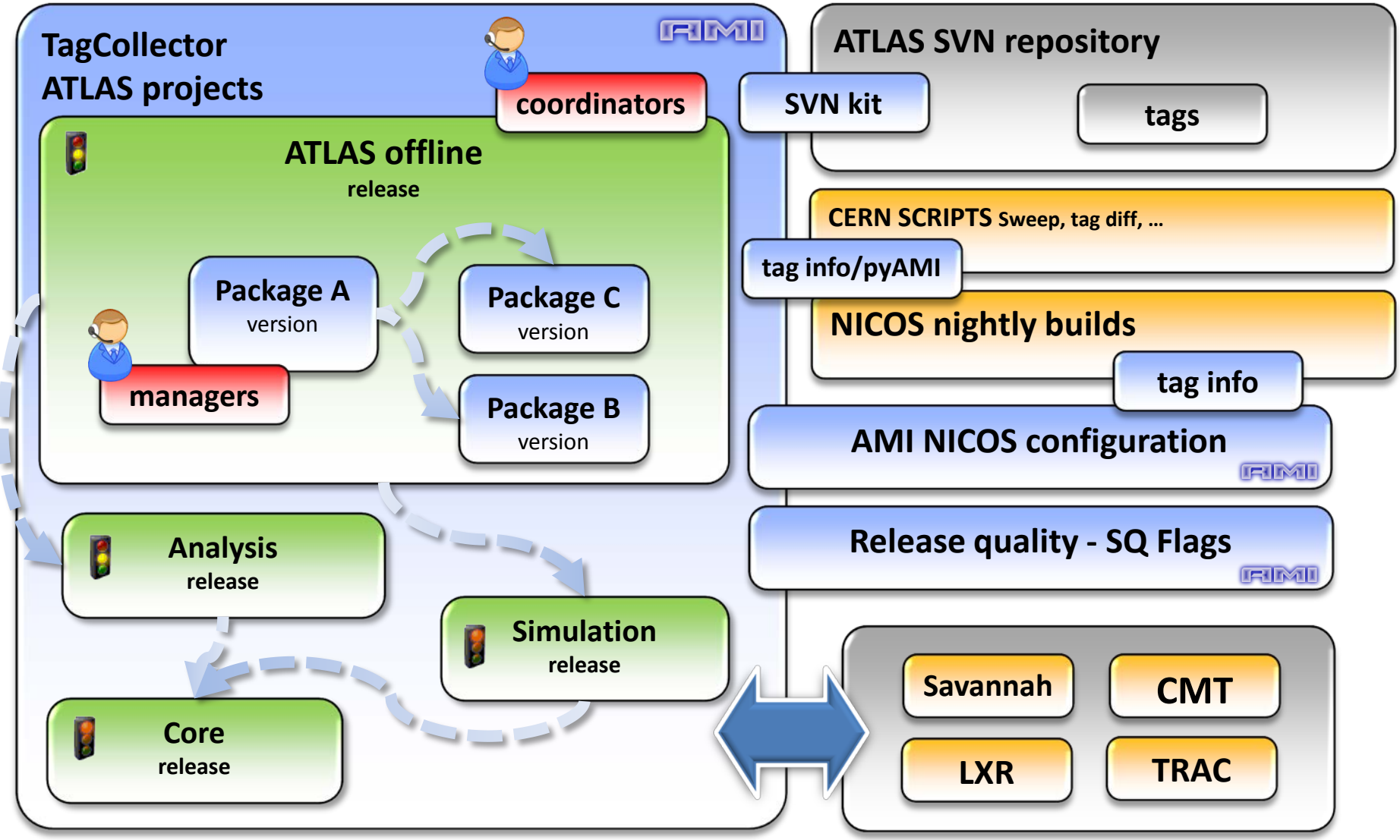
production environment

- the AMI infrastructure is designed to meet strong constraints of service availability and response time as befits an application with almost 2000 users situated all over the world
- summary of the deployment of AMI at the IN2P3 computing centre and at CERN (next slide)
- users connect through Tomcat using either web browsers or the python client
- the task servers are instances of AMI running independently, used for filling the AMI databases from external sources, and for other recurring tasks
- AMI uses both Oracle and MySQL hosted at the French Tier 1

task server


SQL
Router database
Task A
Task B
Task C
Task D
Task
Planification
Class
Priority
Exclusion
Task server
scheduler
Task A
thread
Task B
thread
Task
Commands
External systems
SQL
SQL
AMI databases
SQL

ATLAS Software management



development - deployment