Work groups & developing libraries

1. Scientific Libraries (Numpy, Scipy, Quantities, Pandas, etc.)

2. Module libraries (Neo, ElePhant)

3. Unified Portal (task-sdk: bbp-clients, task-service, etc.)

4. Integration (scripts for integration of all modules)
    - install script
    - update script (search and update version of all libraries)
    - start script (main script to call and operate)
    - interface scripts (bind all scripts together)

ElephantCLI

Software Development Life Cycle for ElePhAnt

1. Plan : Requirement Analysis
2. Define : functional/ non-functional Specification for design
   → Software Requirements Specification SRS
3. Design : develop Architecture bases on specification
   → Design Document Specification DDS
4. Build : Implementation
5. Test : all kind of tests
6. Deploy :
   → Patch/ fix/ update/ upgrade/ maintenance
   → Software Release Life Cycle SRLC (Alpha/ Beta/ Final releases)
      → major – long release : many new features, new structure/ improvements
      → minor – short release : bug fix, additional features

ElephantCLI - Requirements

Plan: Requirements Analysis/ Use Cases

1. Requirements

   1.1 Set of commands CLI

   → run user's data analysis as local tasks
   (without ipython and programming)

   → run user's data analysis as remote tasks on Unified Portal via WebUI

   → decorate & commit & register & start new function to U.P.

   → possibly as runnable-standalone application on different Platforms

   1.2 Programming Python-module

   → install package from PyPi

   → enable programming with ipython

   → easily importable with other programming-libraries

2. Scenarios

   → User wants to use set of commands CLI

   → User wants to do programming using ipython (notebook)

ElephantCLI - Specification

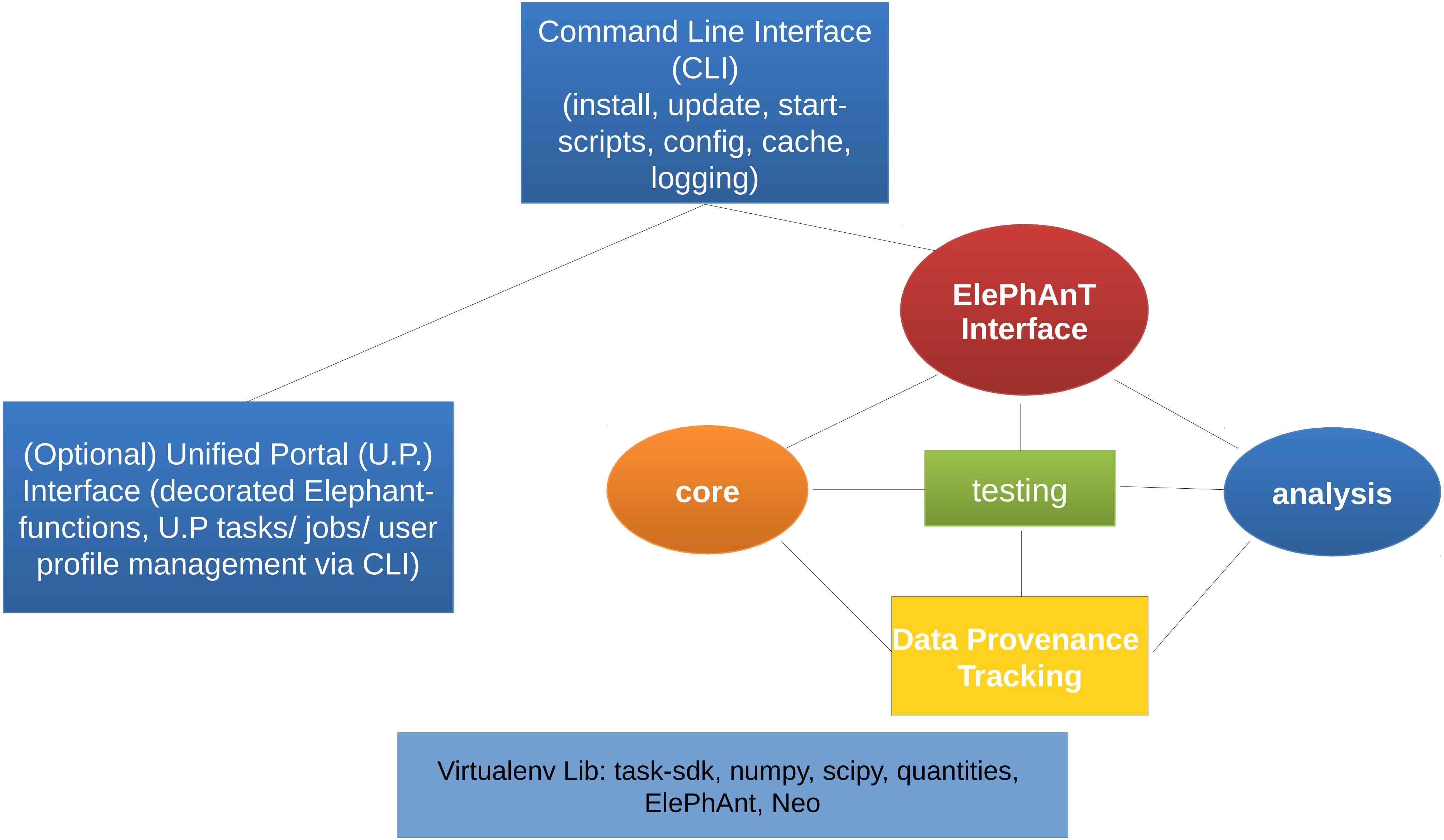
Define: Specification

1. Functional specification
   - set of commands to run all ElePhAnt-functions at local machine
   - set of commands to check result, get infos of tasks & jobs from U.P.
   - logging mechanism (Error, Installation)
     → simple local-cache to save infos (input_values, result) of last run tasks
   - input-data should be in 2 ways
     → IO API that fits many format specifications (using 'neo.io')
     → raw_input (from Command Lines)
   - install & update & test mechanism
   - (binary)-standalone application
2. Non-functional specification
   - friendly CLI
   - documents files (INSTALL, ChangeLog, News, Authors, etc.), autodoc

To let the developers know what to build.
To let the testers know what tests to run.
To let stakeholders know what they are getting.

(Source: http://en.wikipedia.org/wiki/Functional_specification)

ElephantCLI - Architecture



Command Line Interface (CLI)
(install, update, start-scripts, config, cache, logging)

ElePhAnT Interface

(Optional) Unified Portal (U.P.) Interface (decorated Elephant-functions, U.P tasks/ jobs/ user profile management via CLI)

core

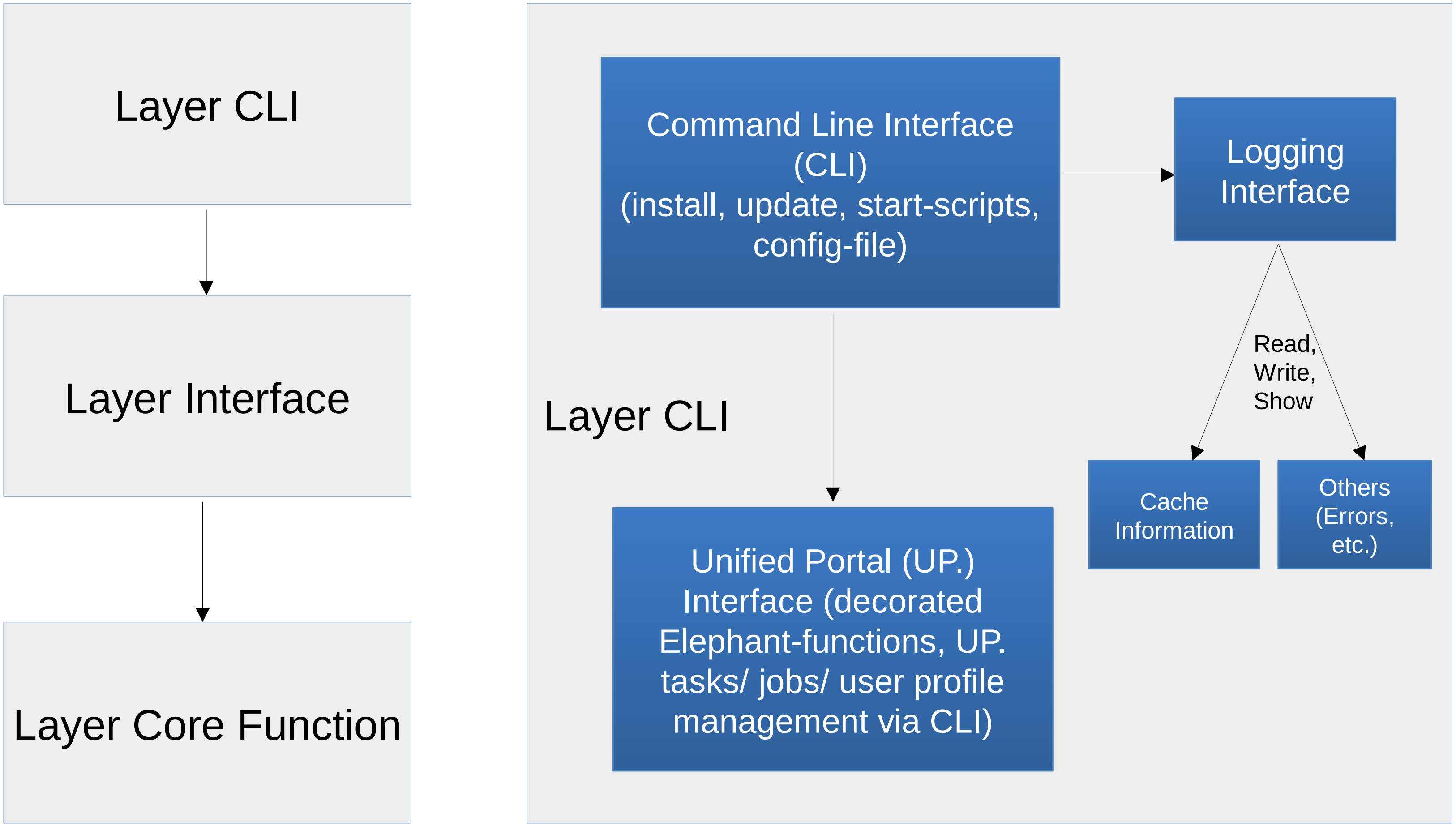
testing

analysis

Data Provenance Tracking

Virtualenv Lib: task-sdk, numpy, scipy, quantities, ElePhAnt, Neo

ElephantCLI - Architecture



Layer CLI

Layer Interface

Layer Core Function

Command Line Interface (CLI) (install, update, start-scripts, config-file)

Logging Interface

Read, Write, Show

Cache Information

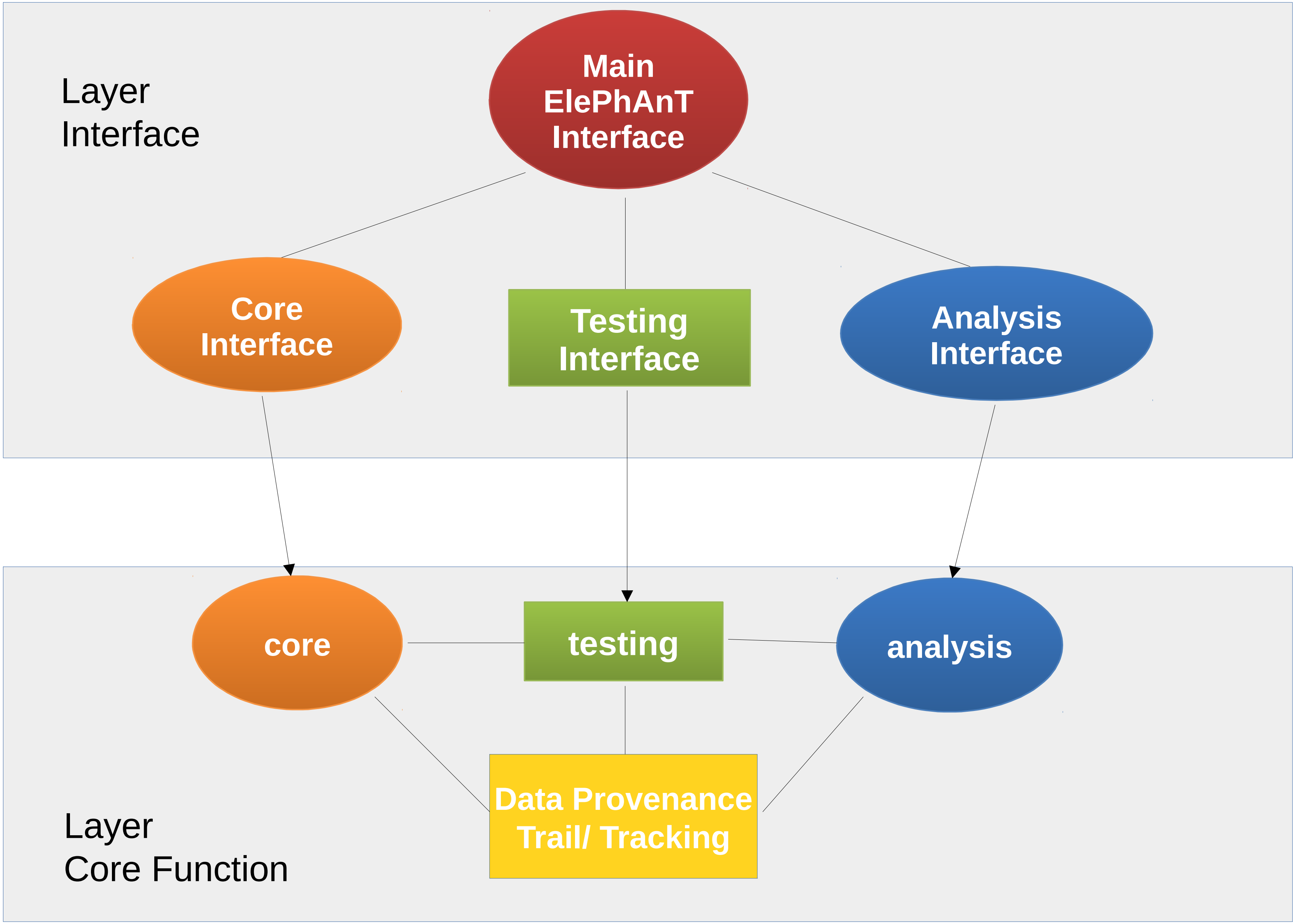
Others (Errors, etc.)

Layer CLI

Unified Portal (UP.) Interface (decorated Elephant-functions, UP. tasks/ jobs/ user profile management via CLI)

Virtualenv Lib: task-sdk, numpy, scipy, quantities, ElePhAnt, Neo

ElephantCLI - Architecture



Layer
Interface

Main
ElePhAnT
Interface

Core
Interface

Testing
Interface

Analysis
Interface

core

testing

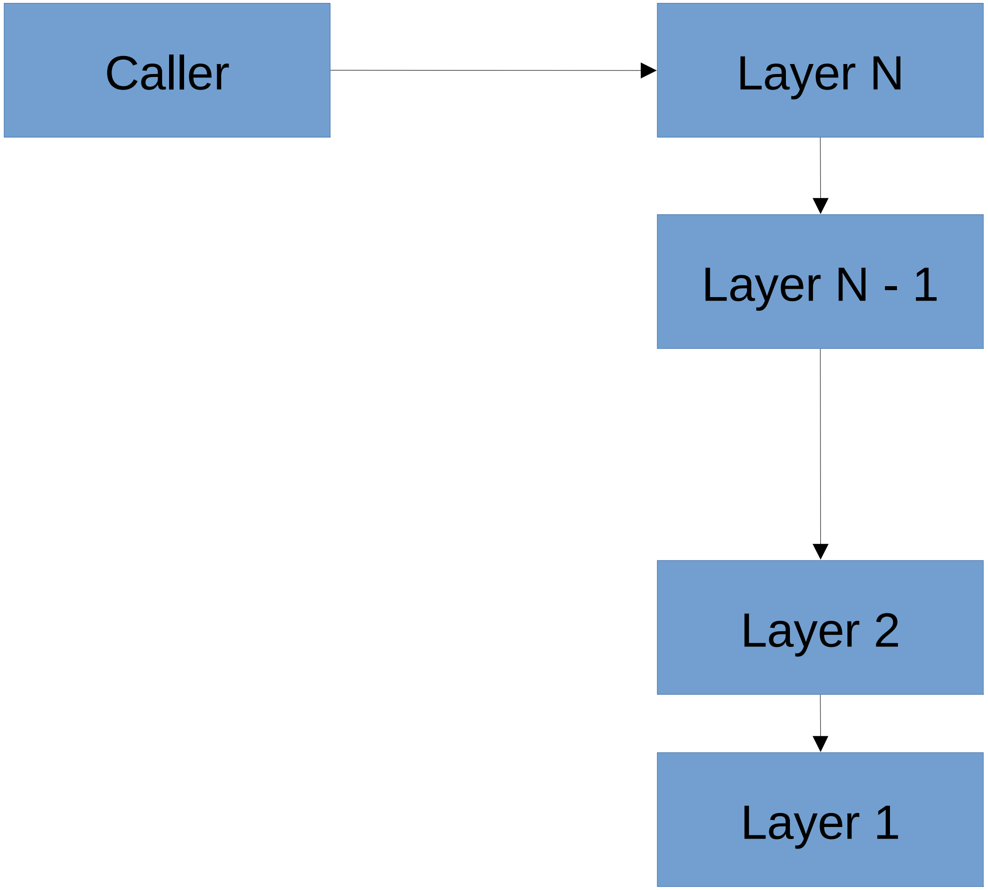
analysis

Data Provenance
Trail/ Tracking

Layer
Core Function

ElephantCLI - Implementation

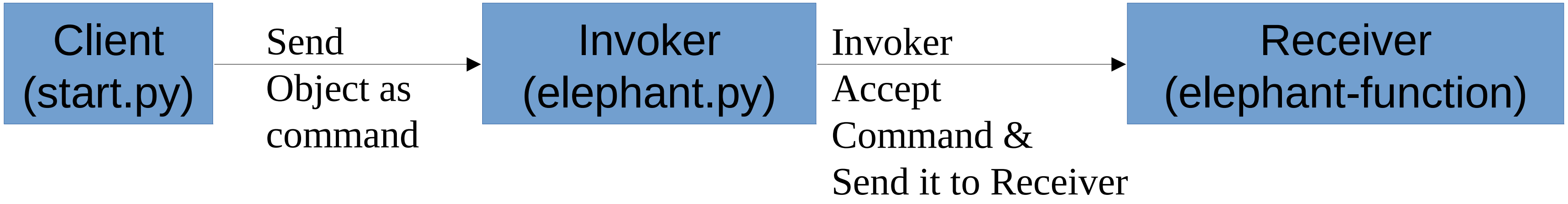
Architectural Pattern Layer



ElephantCLI - Implementation

Design Patterns

1. Command:



Client (start.py)

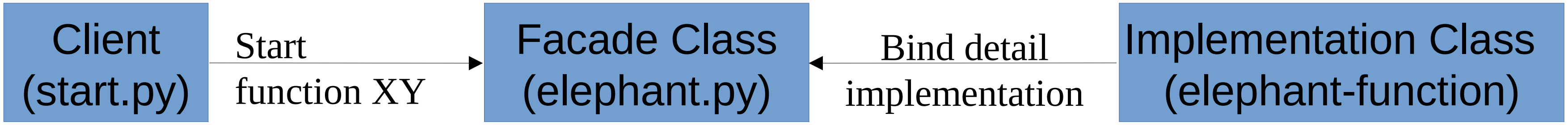
Send Object as command

Invoker (elephant.py)

Invoker Accept Command & Send it to Receiver

Receiver (elephant-function)

2. Facade:



Client (start.py)

Start function XY

Facade Class (elephant.py)

Bind detail implementation

Implementation Class (elephant-function)

Call main-functions at facade-elephant (simplified interface), hide complexity of implementation, bind and move the detail implementation of main-functions to core-functions.

ElephantCLI - Implementation

ElePhAnt directory structure
→ install.py, setup.py, start.py, start_up, Doc-files (authors, license, changes, etc.)
→ elephant/
    ----> core-functions
→ scripts/   # other independent-, test_scripts
    ----> profiling # optimized beta-version ElePhAnt,
→ up/
    ----> query_up.py, unified-portal tasks
    ----> all up_tasks
→ logging/
    ----> different log-files (result, install, task, job, cache)
→ interface/
    ----> elephant.py
    ----> sub-interface to different core-functions
→ test/
    ----> functional tests, performance test
→ doc
→ build, ci

ElephantCLI - Implementation

```
./start.py # script to start ElephantCLI
        --help (-h)                        # show all functional interfaces
                → show : --el & --up & --te
        --elephant (-el)                   # call elephant_interface (elephant.py)
                → -el --help (-h)          # show list of functions
                → -el -c (conversion)      # call func elephant_conversion
                → -el -s (statistic)       # call func elephant_statistic
                → -el …
        --test (-te)                       # call test_interface
                → -te --help(-h)           # show list of functions
                → -te -p  funcXY (run performance test for function XY)
                → -te -el funcXY (run elephant test for function XY)
                → -te -m moduleZ (run elephant test for module Z)

./start_up.py # script with additional unified_portal functions
        --unifiedportal (-up)              # call up_interface (query_up.py)
                → -up --help(-h)           # show  list of functions
                → -up [get_task/ get_all_task/ get_job/ get_all_jobs]
                → -up func_name            # run local task
```

ElephantCLI Demo

Short Video Demonstration (2')

- Create virtualenv
- Installation (run install script)
- Download (download source code from github)
- Setup (setup source code to pypi)
- Run some functions from elephant module
- Run UP_task at local
- Run UP_function to get infos of remote tasks & jobs

Summary

- Standalone data analysis (portable) application, runnable CLI-scripts (capable to run on different platforms independently or Linux-derived platform)
- Scientific library pyElephant for using with ipython
- Open Source Software and is licensed under the GNU General Public License v2 https://github.com/lphan/ElephantCLI

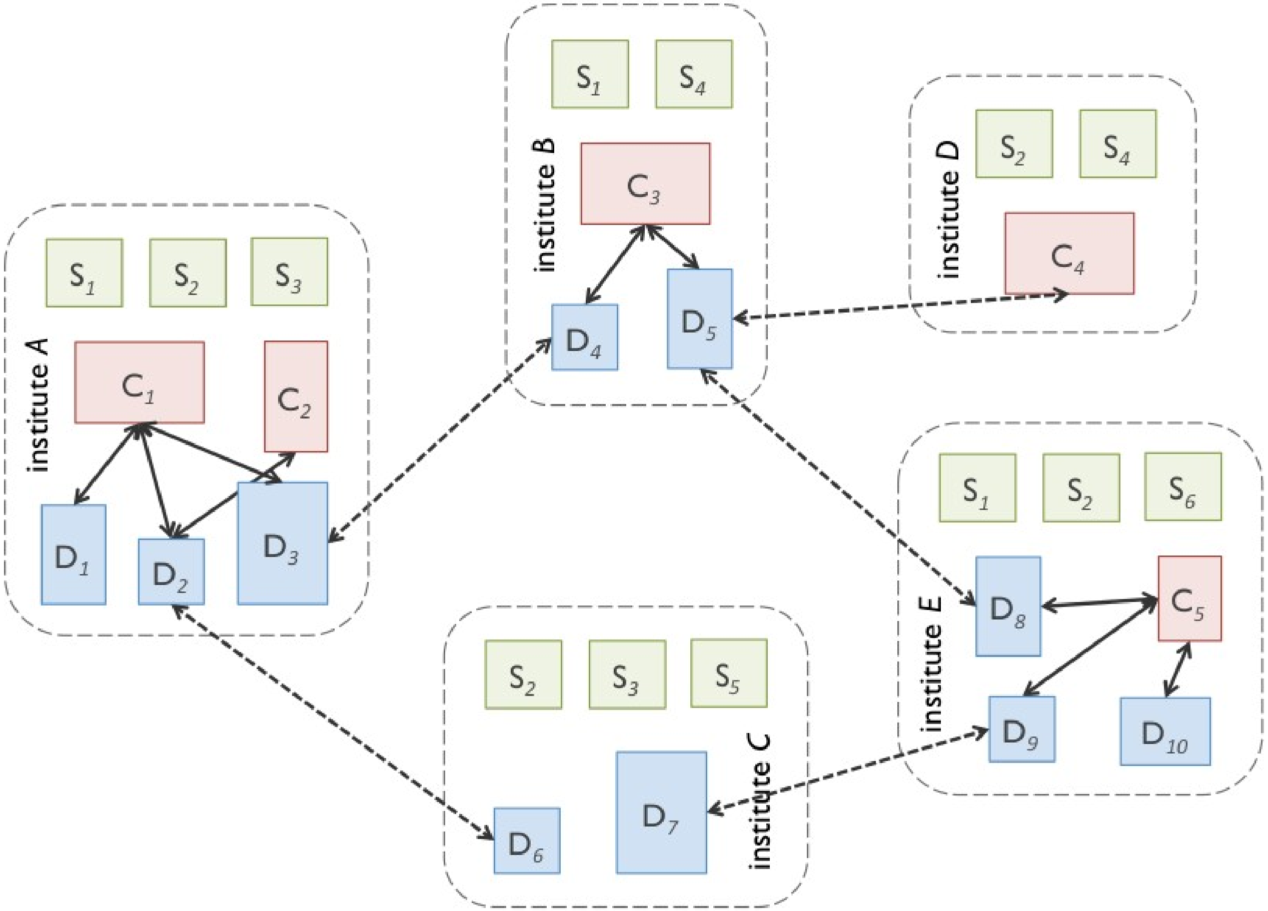
Reference:

Design Patterns in Python Rahul Verma, Chetan Giridhar http://kennison.name/files/zopestore/uploads/python/DesignPatternsInPython_ver0.1.pdf

IEEE Software Life Cycle http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159431

Questions ??

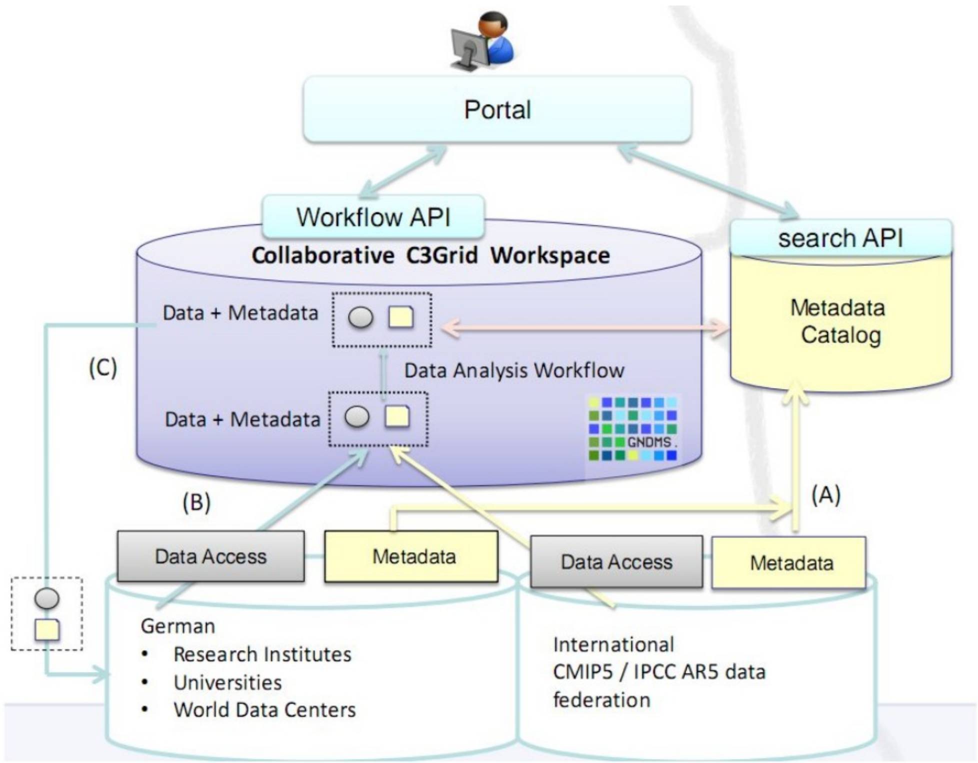
Design of distributed work environment



S: (User) Scientist, C: Computing (Cluster) Resource, D: Data (Storage) Resource

(*Source: LP-Thesis https://github.com/lphan/ElephantCLI/blob/master/doc/Thesis_in_German.pdf*)

Scientific Data processing in distributed Environment



C3Grid - A Collaborative Data Management Infrastructure for Climate Data Analysis. Geophysical Research Abstracts, 14 (EGU2012), p. 10569, 2012. C3-Grid Poster

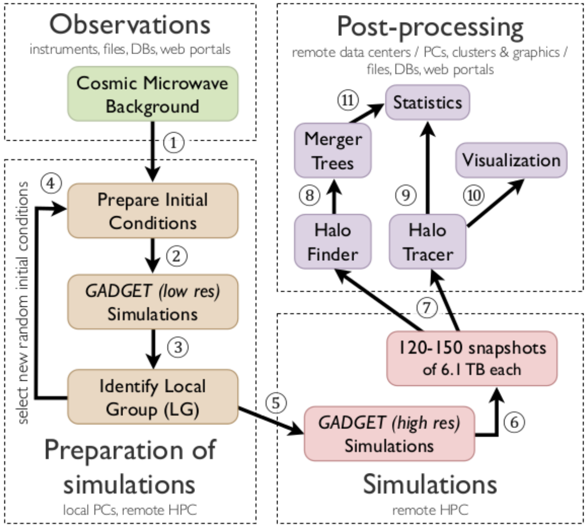
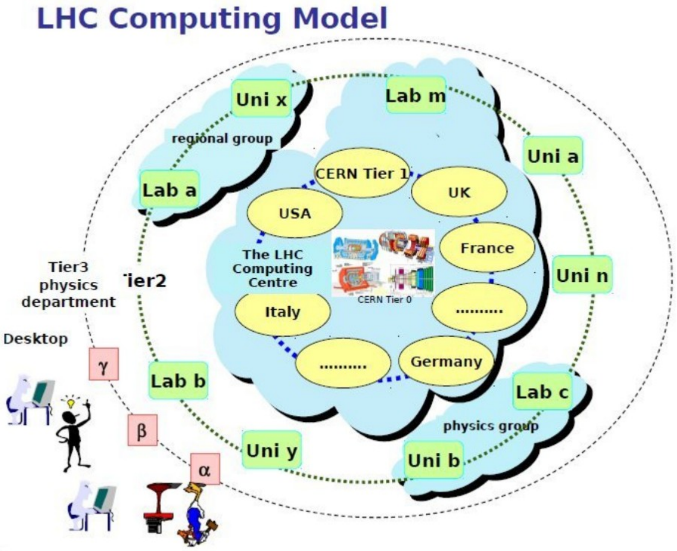

Observations
instruments, files, DBs, web portals
Cosmic Microwave Background
1
Prepare Initial Conditions
2
GADGET (low res) Simulations
3
Identify Local Group (LG)
4
select new random initial conditions
Preparation of simulations
local PCs, remote HPC
5
GADGET (high res) Simulations
6
120-150 snapshots of 6.1 TB each
7
Simulations
remote HPC
Post-processing
remote data centers / PCs, clusters & graphics / files, DBs, web portals
Halo Finder
8
Merger Trees
11
Statistics
Halo Tracer
9
10
Visualization

Figure 1: CLUES workflow.

Vision of a Virtual Infrastructure for Storing & Processing Scientific Data



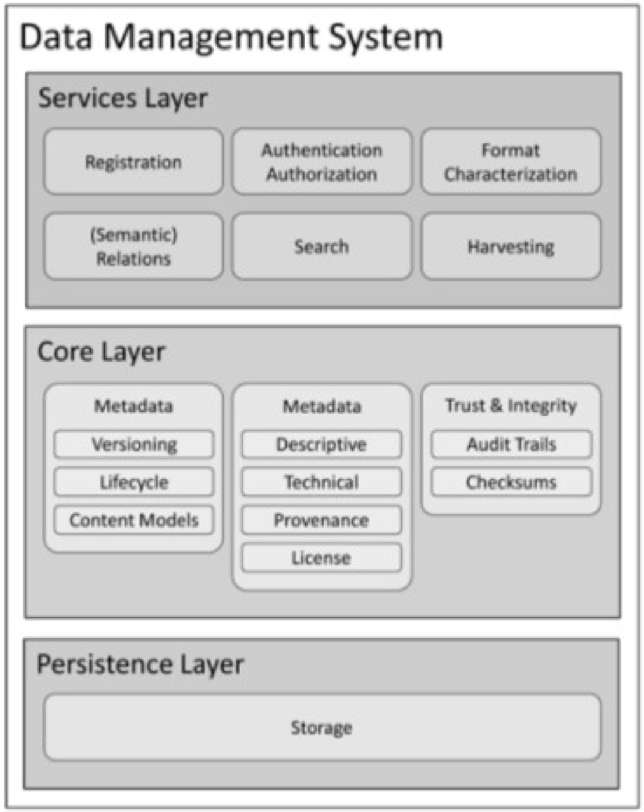
LHC Computing Model - Deploying the LHC Computing Grid The LCG Project - Ian Bird IT Division, CERN CHEP 2003 27 March 2003

Use cases

→ A way to **manage** & use our infrastructure resource efficiently:
- Computing resource
- Storage Capacity resource
- Network bandwidth

→ A way to **share** our computing results with other partners
- data replication
- authentication, authorization
- data grid for knowledges sharing, etc.

→ A way to maintain and reuse our data & computing results **reproducibly**
- data management, data processing-pipeline
- data search, metadata, database, data policies
- digital library of research results, etc.



Source: Handbuch Forschungsdatenmanagement (in German)
http://www.forschungsdatenmanagement.de/

ElePhAnt - v2

2. Version: iElephant (= iRODS + Elephant)

Client-Server architecture (web-based application)

Database and applications running on remote-Server

User-Client sends query-request to Server:

→ To retrieve information stored at Database (Meta data, result-cache, logging)

→ To execute single computing-task or multi Workflow-tasks

→ To update new information to their authorized data

iRODS: Rule Framework, Data Grid Middleware, tool used for management of data life cycle (Data Discovery, Workflow automation, Secure Collaboration, Data Virtualization)

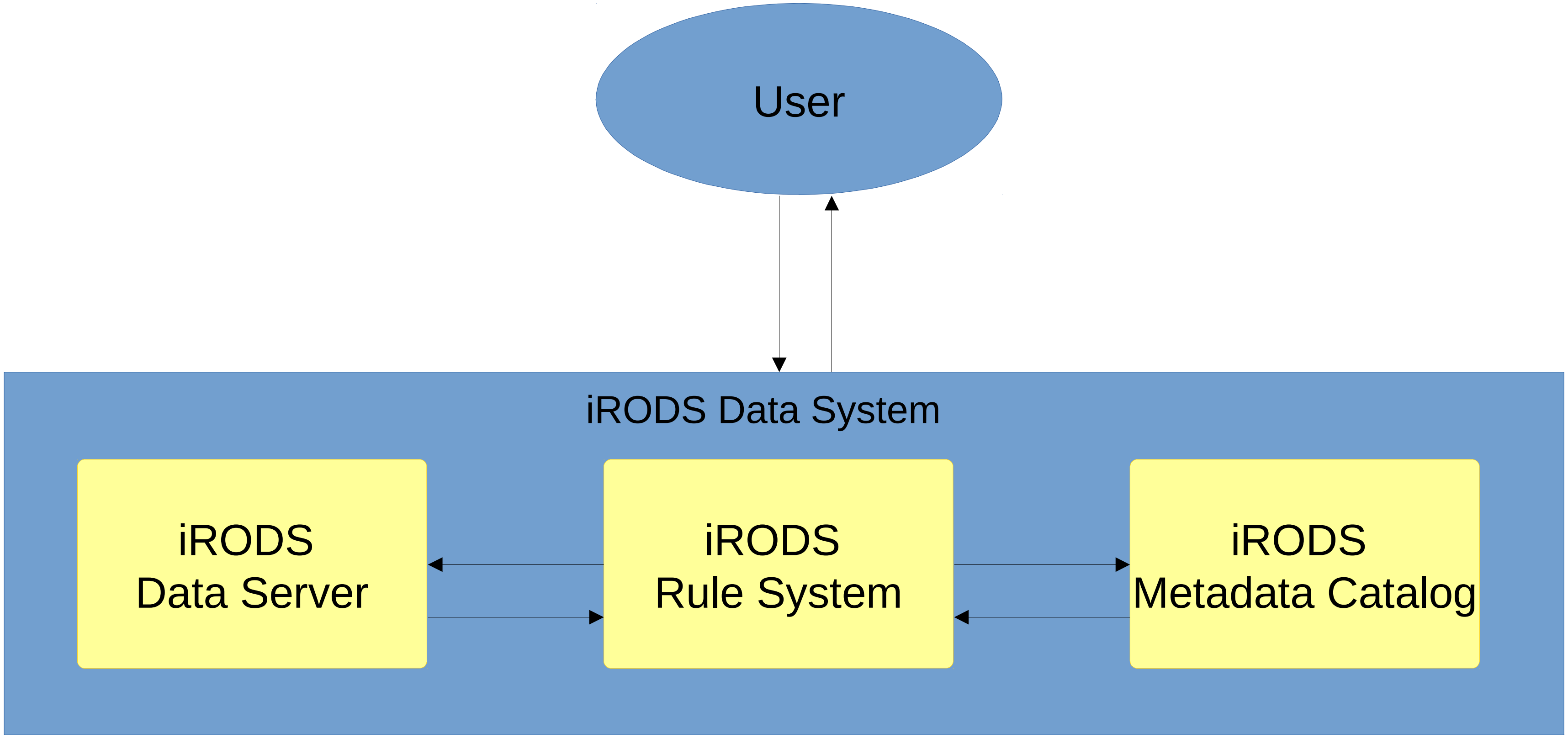
iRODS + Elephant = a environment of data and computing management for data analysis toolkit

→ Data exchange between irods-servers at different scientific institutes

→ Others: UP is working on integration irods into UP-backend

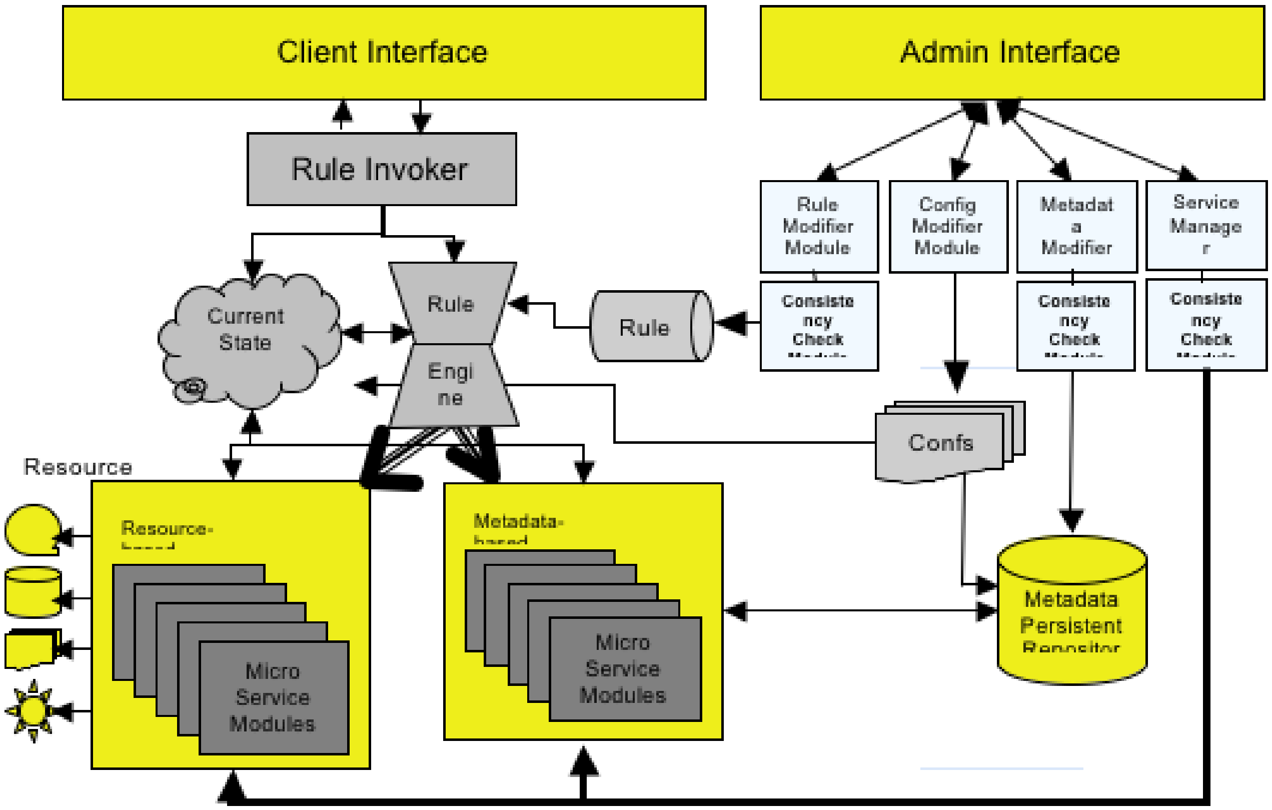
iRODS Integrated Rule-Oriented Data System

Simplified iRODS-Architecture



iRODS Integrated Rule-Oriented Data System

Architecture



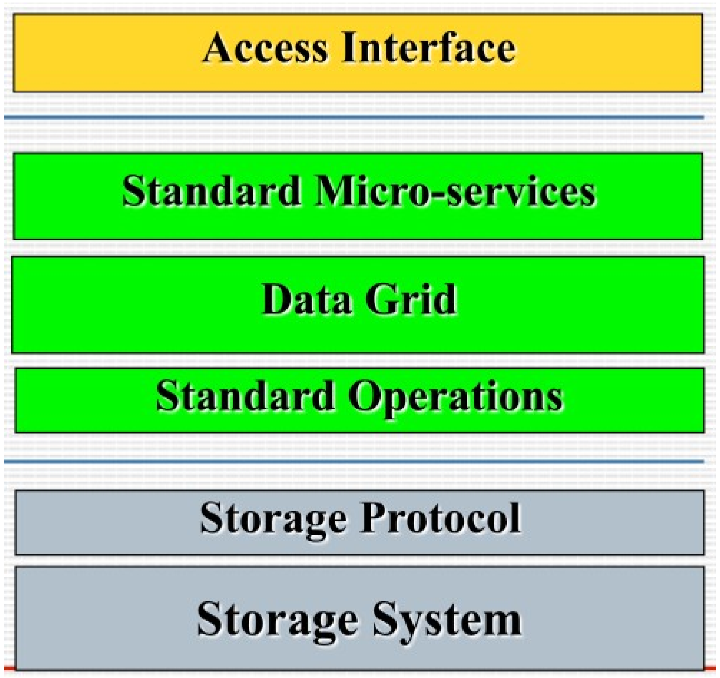
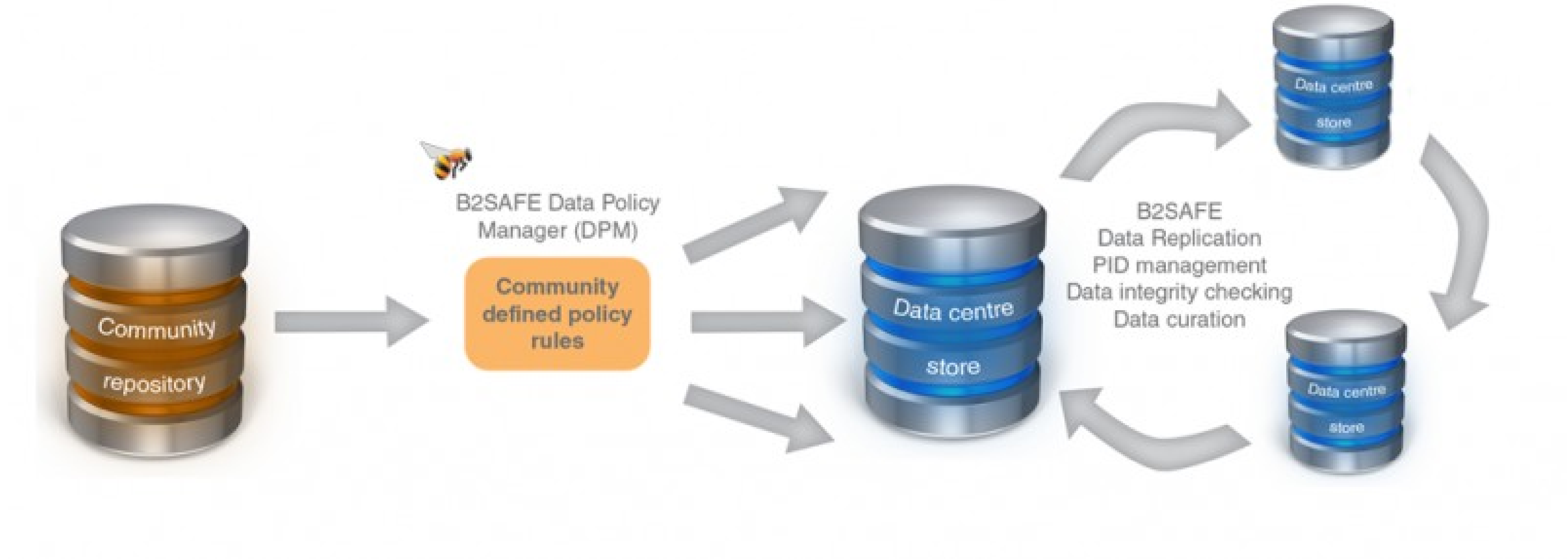


Figure 5. iRODS Architecture Components

Source : https://wiki.irods.org/index.php/iRODS_Components
iRODS_workshop_ISGC_Taiwan_03-2010 Reagan Moore

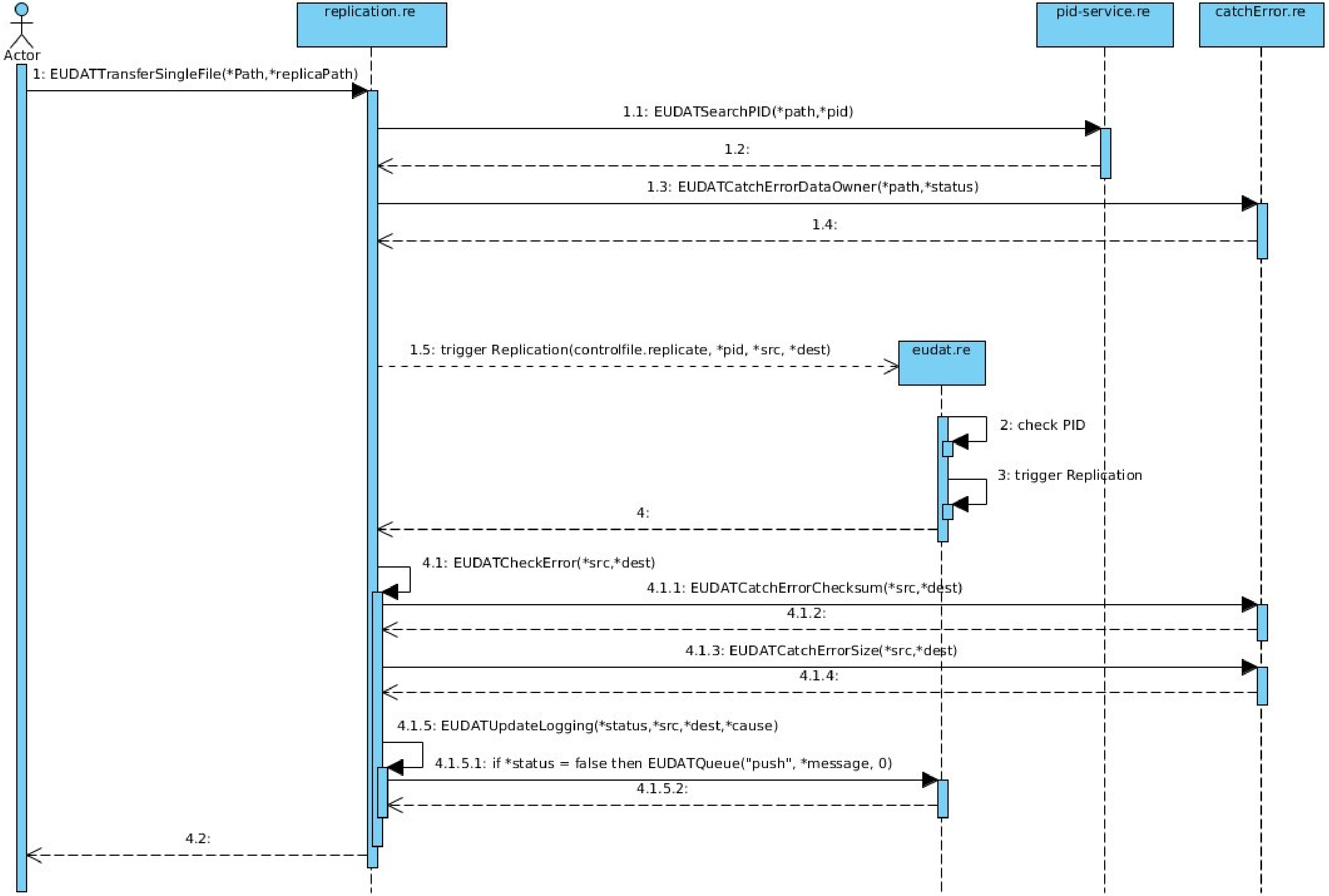
EUDAT-B2Safe

Architecture



Source: http://www.eudat.eu/,
https://github.com/EUDAT-B2SAFE/B2SAFE-core

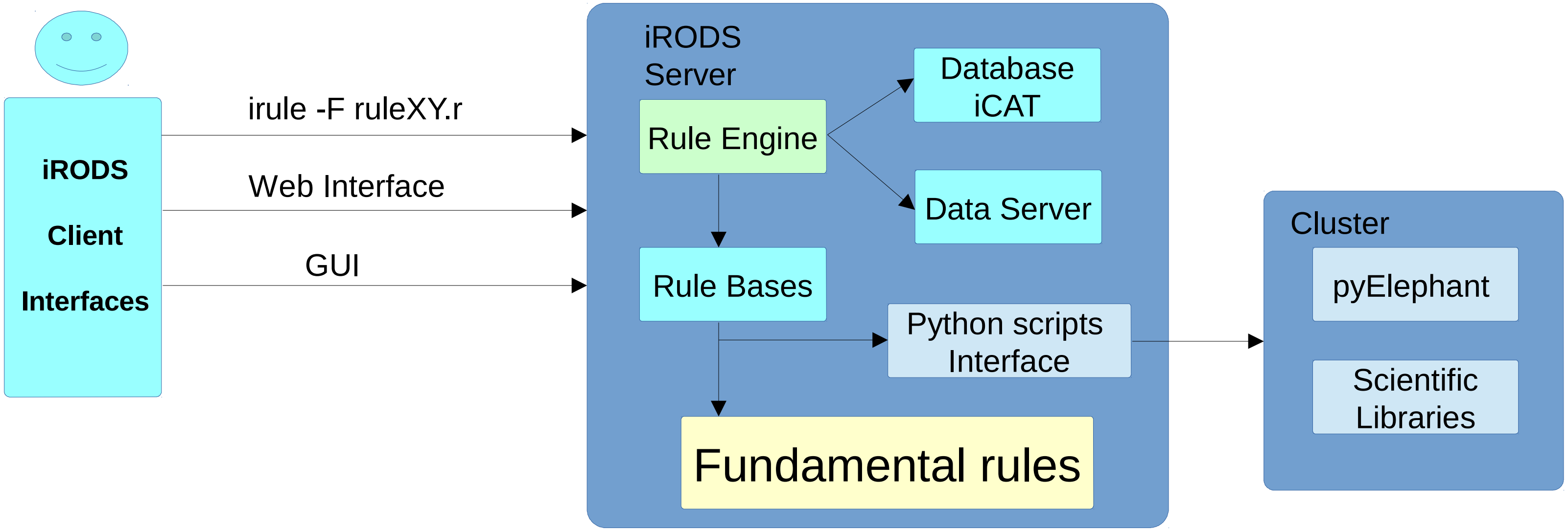
EUDAT-B2Safe


Actor
replication.re
pid-service.re
catchError.re
1: EUDATTransferSingleFile(*Path,*replicaPath)
1.1: EUDATSearchPID(*path,*pid)
1.2:
1.3: EUDATCatchErrorDataOwner(*path,*status)
1.4:
1.5: trigger Replication(controlfile.replicate, *pid, *src, *dest)
eudat.re
2: check PID
3: trigger Replication
4:
4.1: EUDATCheckError(*src,*dest)
4.1.1: EUDATCatchErrorChecksum(*src,*dest)
4.1.2:
4.1.3: EUDATCatchErrorSize(*src,*dest)
4.1.4:
4.1.5: EUDATUpdateLogging(*status,*src,*dest,*cause)
4.1.5.1: if *status = false then EUDATQueue("push", *message, 0)
4.1.5.2:
4.2:

Source:
https://github.com/EUDAT-B2SAFE/B2SAFE-core
https://github.com/EUDAT-B2SAFE/B2SAFE-core/blob/master/docs/integrityCheck.pdf

iElephant Architecture Know-How

Elephant on Cluster, virtual infrastructure for storing & processing Scientific Data



Use case 1:
Make an copy of data set **(replication)** to other different institutes

Use case 2:
User calculates function_X, saves result_Y to folder_Z, converts result_Y into format_F, delivers right access **(sharing)** to group

User-defined case:
User builds up their own chain-of-tasks by assembling different rules from RuleBases

References

Virtual Infrastructure for Storing & Processing Scientific Data
http://www.cs.tu-dortmund.de/nps/de/Forschung/Publikationen/Graue_Reihe1/Ver__ffentlichungen_2011/839.pdf

iRODS Rule Programming
http://wiki.irods.org/index.php/Publications

Riding the wave
http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/hlg-sdi-report.pdf

The Fourth Paradigm
http://research.microsoft.com/en-us/collaboration/fourthparadigm/

Open Handbuch Forschungsdatenmanagement (in German)
http://www.forschungsdatenmanagement.de/

LP-Thesis
https://github.com/lphan/ElephantCLI/blob/master/doc/Thesis_in_German.pdf