CRIC and Rucio integration for CMS Data Management

Fabio Condomitti

Final report – Supervisor: N. Ratnikova, E. Vaandering

25th September 2019

Overview

❑ Intro and project goals

❑ Case study

❑ Generalization

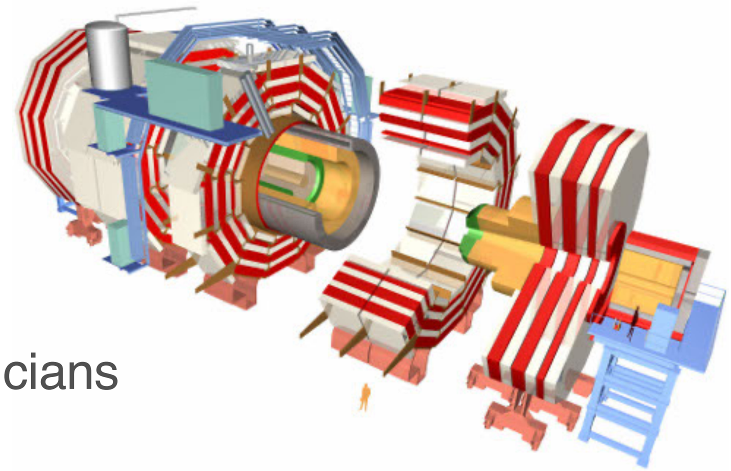
❑ Testing

❑ Conclusions

Compact Muon Solenoid (CMS)

Particle physics detector on the LHC looking for new physics:

- dark matter
- extra dimensions
- Higgs Boson discovery in 2012



CMS Collaboration [1]:

- more than 4000 scientists, engineers and technicians
- students from 220+ institutes and 50+ countries
- 364 PB written to tape (without replicas)

Data management challenges:

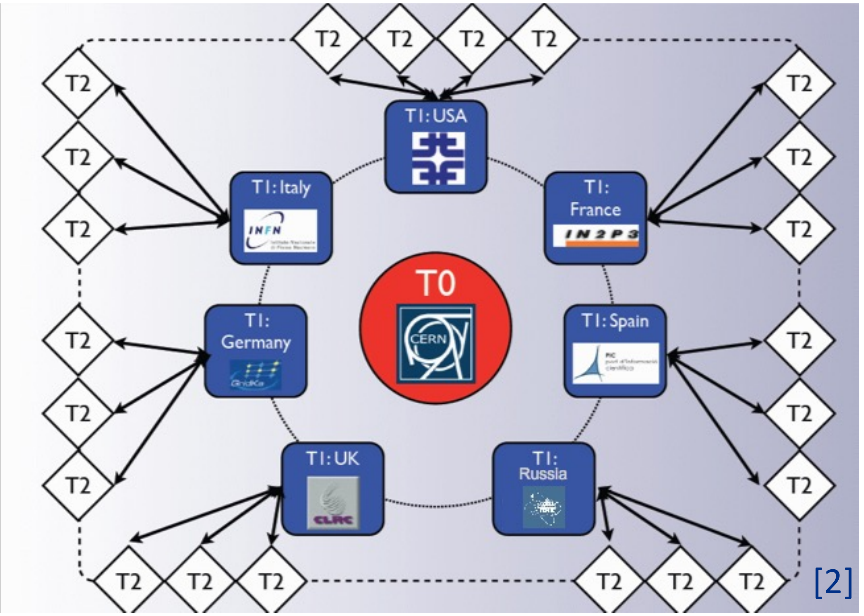
- 1.3 PB/day data transfer rate across the Data Grid
- global data access

CMS Computing infrastructure

- ❑ **Tier-0**: (at CERN) for highly organized production work and quasi-realtime data flows:
  - ○ classify data into 50 datasets according to their physics content
  - ○ keeps a full copy of the raw data

- ❑ **Tier-1**: 7 centers at large regional computing sites:
  - ○ store second copy of CMS raw data
  - ○ data permanent storage allowing high-throughput access for providing selected subsets of data to Tier-2 centers

- ❑ **Tier-2**: 55 centers
  - ○ substantial CPU resources
  - ○ relatively small storage (not permanent)
  - ○ data analysis



[2]

Long Shutdown 2 (LS2)

❑ 2 years of shut down to upgrade (2019-2020):

- accelerator
- detectors
- software and computing tools
- etc.

❑ Transition to a more powerful Data Management system to handle higher data taking rates

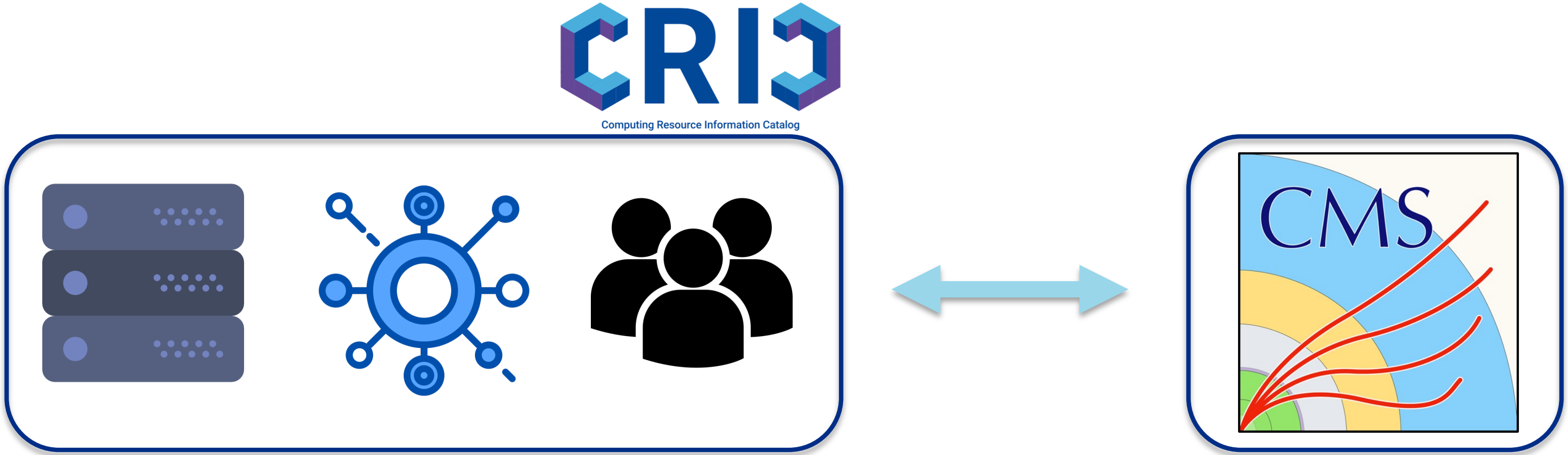
❑ CMS upgrades the tools in favor of community projects:

- CRIC
- Rucio

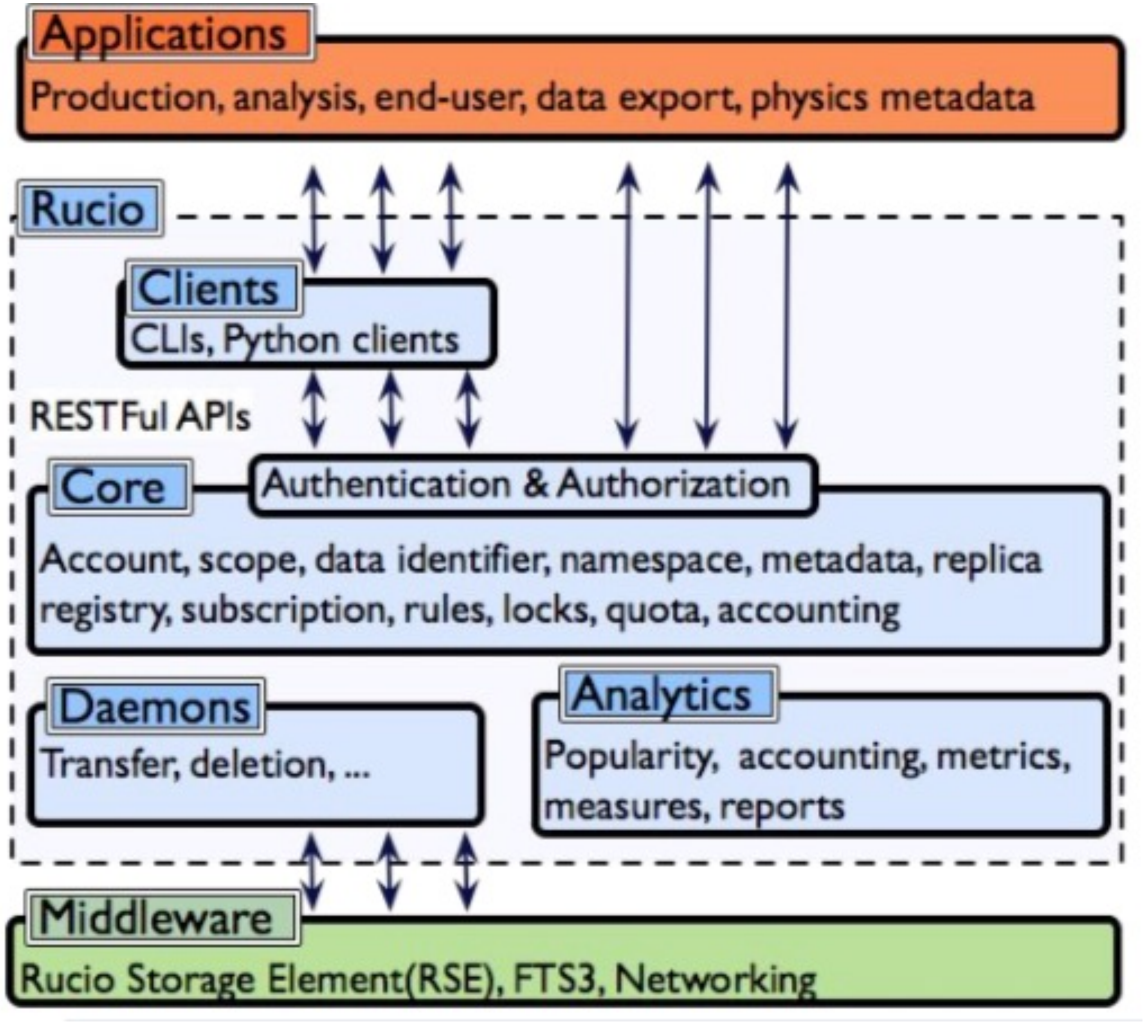
CRIC for computing resource information

High-level middleware to describe the Computing model topology:

- ❑ provides unified description of resources and services [3]
- ❑ holds sites topology
- ❑ stores users information



Rucio for Data Management


Applications
Production, analysis, end-user, data export, physics metadata
Rucio
Clients
CLIs, Python clients
RESTFul APIs
Core
Authentication & Authorization
Account, scope, data identifier, namespace, metadata, replica registry, subscription, rules, locks, quota, accounting
Daemons
Transfer, deletion, ...
Analytics
Popularity, accounting, metrics, measures, reports
Middleware
Rucio Storage Element(RSE), FTS3, Networking

Rucio for Data Management

Stores, manages data transfers in a heterogeneous distributed environment [4]

- ❑ *Rucio Storage Element* (RSE)
  - ○ logical abstraction for physical sites
    - For example → `T1_US_FNAL_Disk (working set), T1_US_FNAL_Tape (archival), T2_US_Florida (working set)`
- ❑ *Replication Rules*
  - ○ allow a user to pin data replicas to RSE for analysis or to archive (e.g. to reproduce results later)
  - ○ every rule is owned by an account

    → "Three copies of this datasets at these sites, one on TAPE, two on DISK"
  - ○ rules expire → data replica with no rules can be deleted
- ❑ Account *quotas* on the RSEs
  - ○ to create a rule an account needs a quota (portion of storage) on an RSE

Objectives

❑ Current state:

- RSEs and quotas are manually assigned through Rucio CLI
- Some policies are documented but not generally formalized in a programmatic way

❑ Automate quotas assignment based on policy:

- default quota (fair)
- site admins can overwrite it

❑ Describe US CMS Policies in a programmatic way

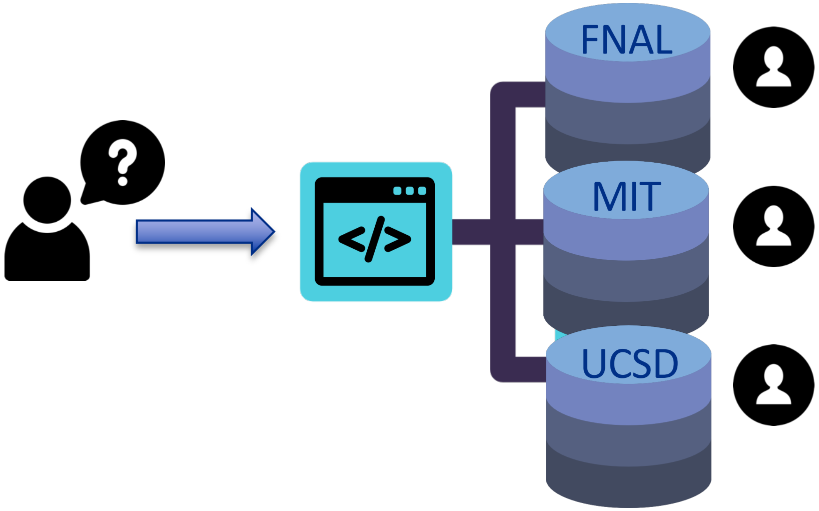
- users are matched with a Tier-2 site on the basis of geography and capacity

Use case

- Download CRIC users list

  1. for each CRIC user:
     i. find the right RSE for him by applying certain policies
     ii. if the user is new in CRIC and not exists in Rucio
        → create a new Rucio account

  2. set a default quota for this user at this RSE



User to site mapping algorithm

❑ Input parameters from CRIC:

- Username
- Distinguished Name (DN)
- Home institute
- Institute country

```
DEFAULT_RSE_QUOTA = 10    # TB
cric_url = 'https://cms-cric.cern.ch/api/accounts/user/query/list/?json'
cric_global_user = json.load(urllib2.urlopen(cric_url))

for key, user in cric_global_user:
    institute_country = user['institute_country']
    if country in institute_country:
        name = key
        dn = user['dn']
        institute = user['institute']
        email = user['institute']
```

User to site mapping algorithm

- Python dictionary (`cric_user`):
  - collection of objects (`<key>:<value>` pairs)
  - objects accessed via keys
  - nested dictionary

```
cric_user = {
        ’email@email.com’ : {
                ’dn’  : ” ”,
                ‘institute’ : “ ”,
                ‘institute_country’ : “”,
                ‘quota’ : {
                        ’some_RSE’  : 0,      # TB
                        ’other_RSE’  : 0,
                }
        }
}
```

User to site mapping algorithm

❑ Python dictionary of lists (`rses_by_country`):

- formalize US CMS Policies
- modular code

```
rses_by_country = {
        ’US’ : {
                ’T2_US_MIT’  : [‘Boston University’,  ‘Brown University’,  ‘MIT’, ‘Boston University’],
                ’T2_US_Florida’ : [‘Florida State University’,  ‘University of Florida’]
                },
        ’IT’ : {
                ’T2_IT_Rome’ : [‘University of Rome’,  ‘Rome 3’],
                ’T2_IT_Pisa’ : [‘University of Pisa’,  ‘University of Florence’]
                }
}
```

User to site mapping algorithm

❑ Output

- each user mapped on a certain RSE
- has a default quota at that site

```
cric_user = {
        ’fabio@email.com’ : {
                ’dn’  : ”DC=ch/DC=cern/OU=Organic ”,
               ‘institute’ : “Boston University”,
               ‘institute_country’ : “US”,
               ‘quota’ : {
                        ’T2_US_MIT’  : 10,   # TB
                        ’other_RSE’  : 0,
               }
        }
}
```

Case study

- ❑ Objectives achieved:
  - ○ automate quota assignment ✓
  - ○ formalize LPC Policies ✓

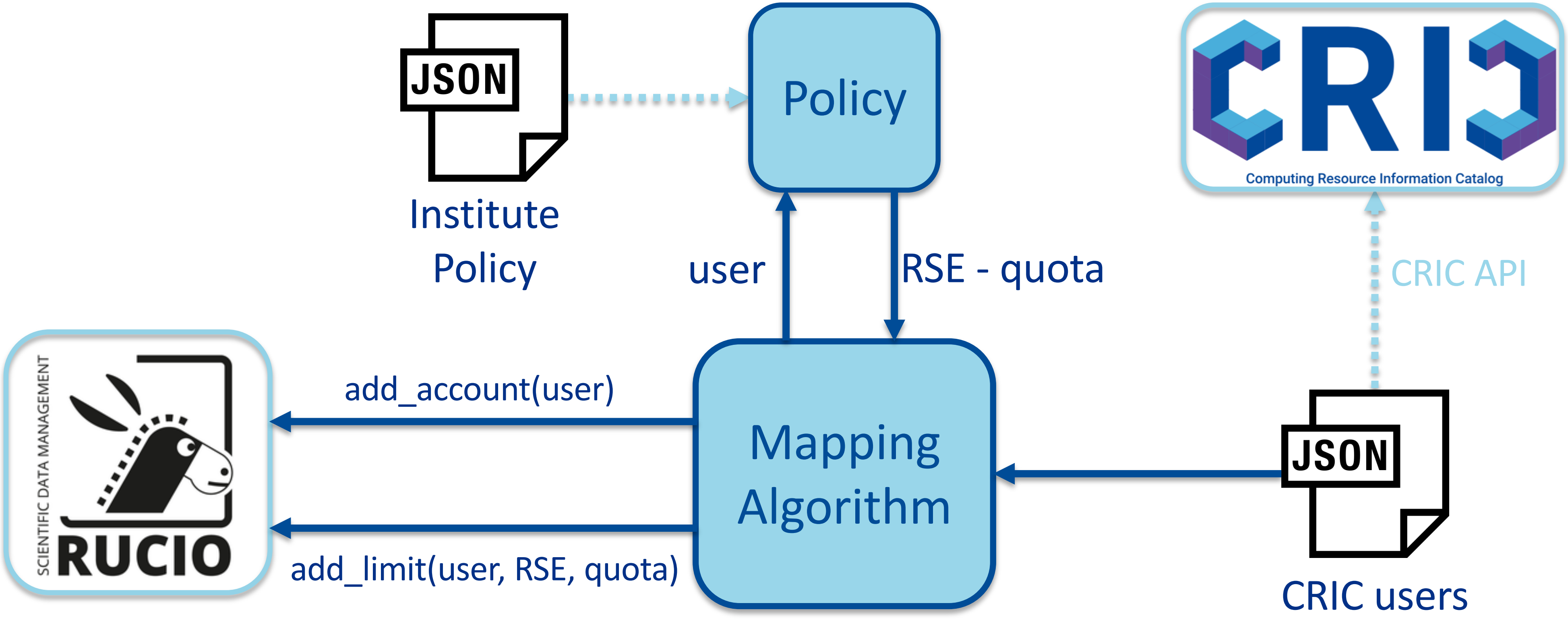
- ❑ Next steps:
  - ○ Generalize the algorithm → Object Oriented Programming
  - ○ Testing phase

Object Oriented Programming (OOP)

- ❑ Programming paradigm based on *objects*:
  - ○ an *object* is an instance of a class
  - ○ each one can contain data (*attributes*)
  - ○ these data are accessible through procedures (*methods)*

- ❑ Advantages:
  - ○ more abstraction levels and to hide the complexity of inner implementations
  - ○ modularity for easier to modify and troubleshoot
  - ○ reuse of code through inheritance

CRIC and Rucio integration



JSON

Institute Policy

Policy

user

RSE - quota

CRIC

Computing Resource Information Catalog

CRIC API

SCIENTIFIC DATA MANAGEMENT

RUCIO

add_account(user)

add_limit(user, RSE, quota)

Mapping Algorithm

JSON

CRIC users

Workflow

1. Download CRIC users list and store it in a JSON file

2. Open the JSON file and for each CRIC user:

   i. find the right RSE for him by using the *get_rse(username)* of InstitutePolicy instance

   ii. create a *CricUser* object and put it in a list

   iii. if the user is new in CRIC and not exists in Rucio

   → create a new Rucio account

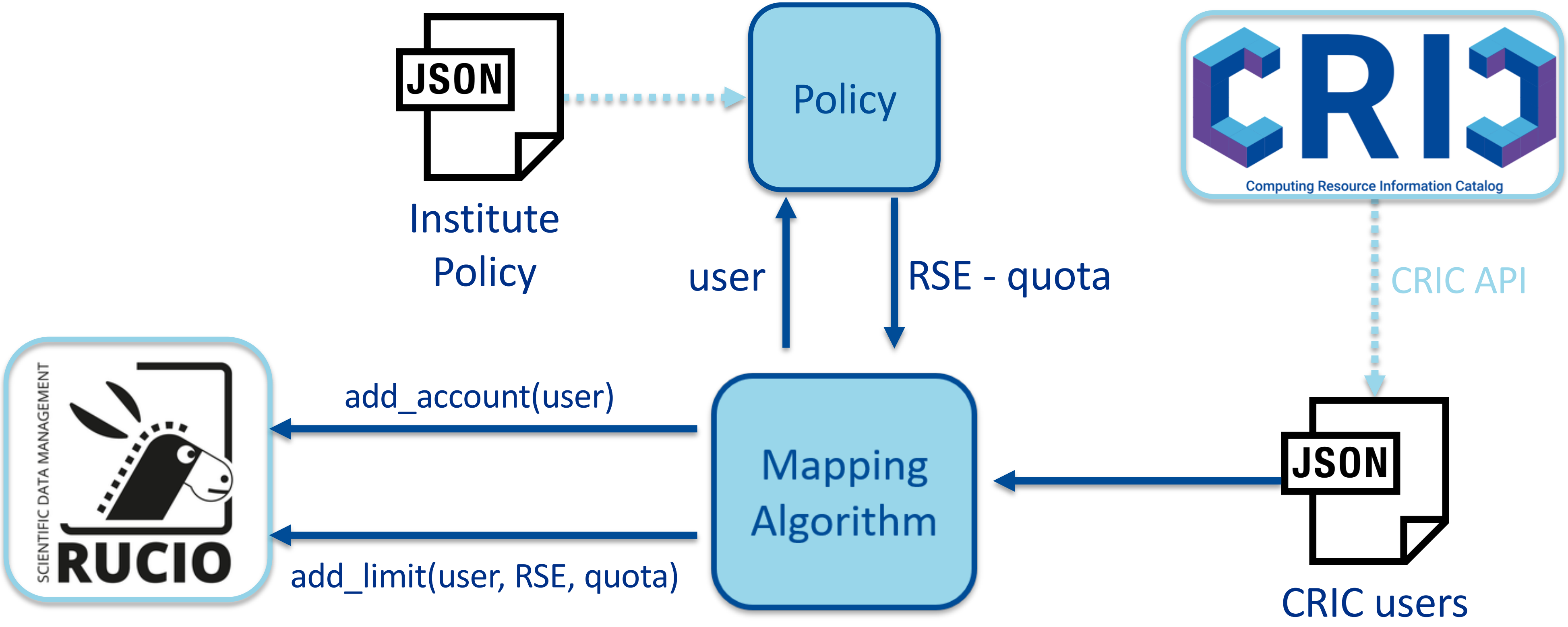
3. set a default quota for this new user at the given RSE

Options of the mapping algorithm

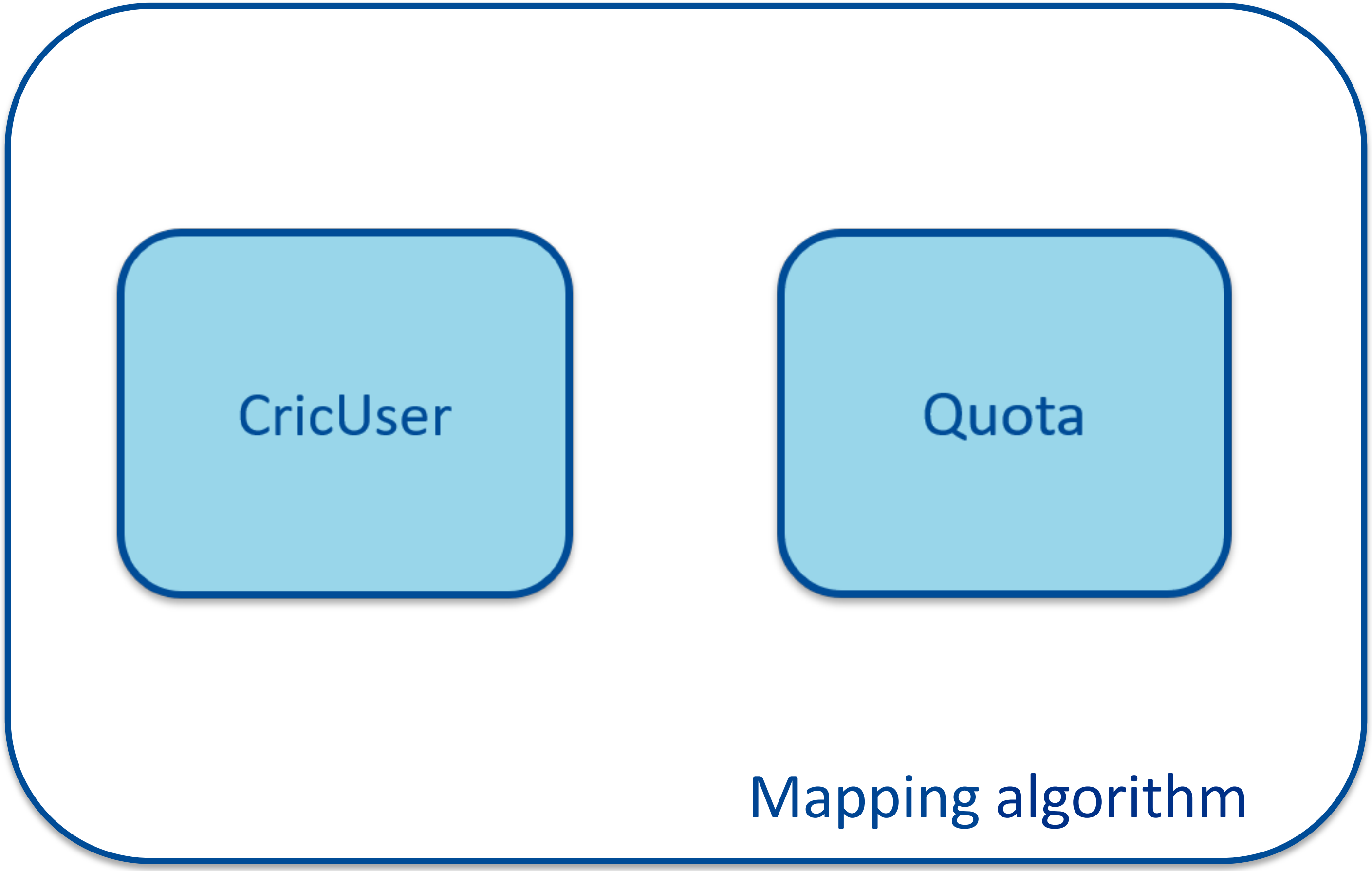
The mapping algorithm can be executed in different configurations by passing some command line arguments:

- Mode
  1. *set-new-only*: allows to add new CRIC users to Rucio and set the default quotas for them only, others are not modified
  2. *reset-all*: allows to reset all the quotas to the default value of all users
  3. *delete-all*: allows to delete all the default quotas of all users
- Dry run:
  1. *off*: normal operating mode
  2. *on*: to test the case in which a new user joins CRIC at any time

CRIC and Rucio integration



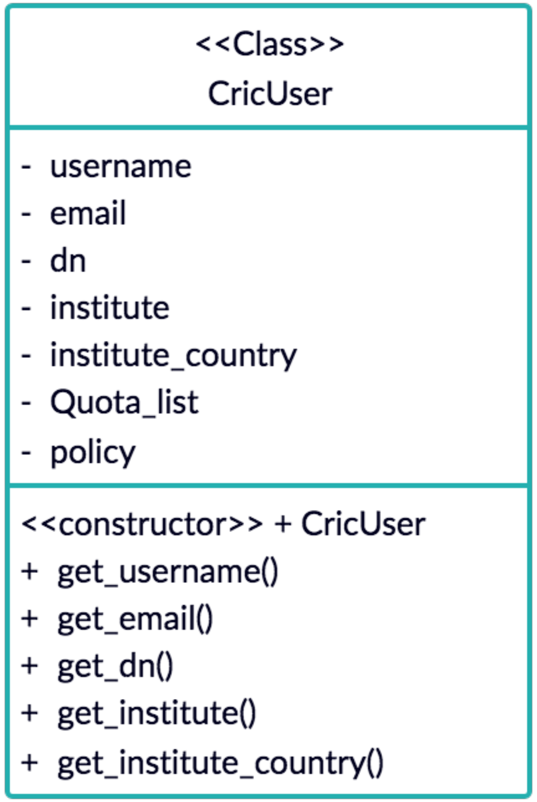
Classes of the Mapping Algorithm



CricUser class

- ❑ Load from JSON CRIC users and store them in a list (*cric_user_list*)

- ❑ A *CricUser* object stores all the information needed about one user to run the mapping algorithm:
  - ○ username
  - ○ email
  - ○ distinguished name
  - ○ institute
  - ○ institute country
  - ○ list of *Quota* objects
  - ○ policy



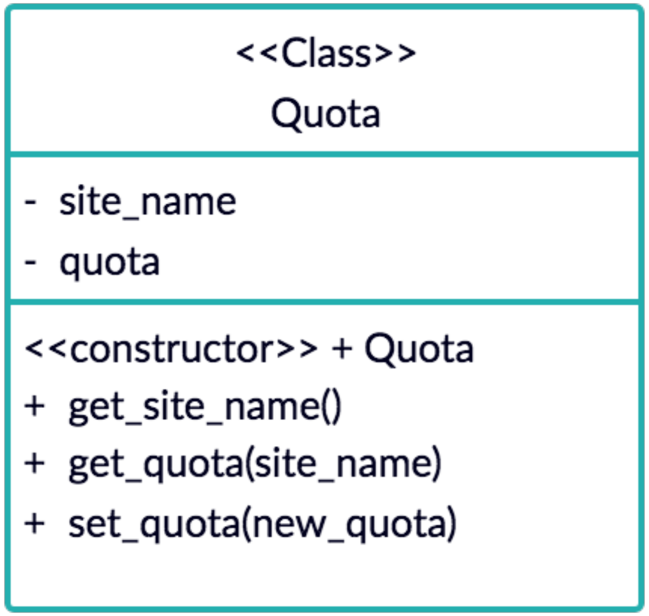
```
<<Class>>
CricUser

- username
- email
- dn
- institute
- institute_country
- Quota_list
- policy

<<constructor>> + CricUser
+ get_username()
+ get_email()
+ get_dn()
+ get_institute()
+ get_institute_country()
```

Quota class

- □ Each one of this object stores the RSE – quota pair
- □ An user may have a list of *Quota* objects
- □ By iterating on this list quotas on different RSEs can be assigned
- □ Extendable implementation for different policies

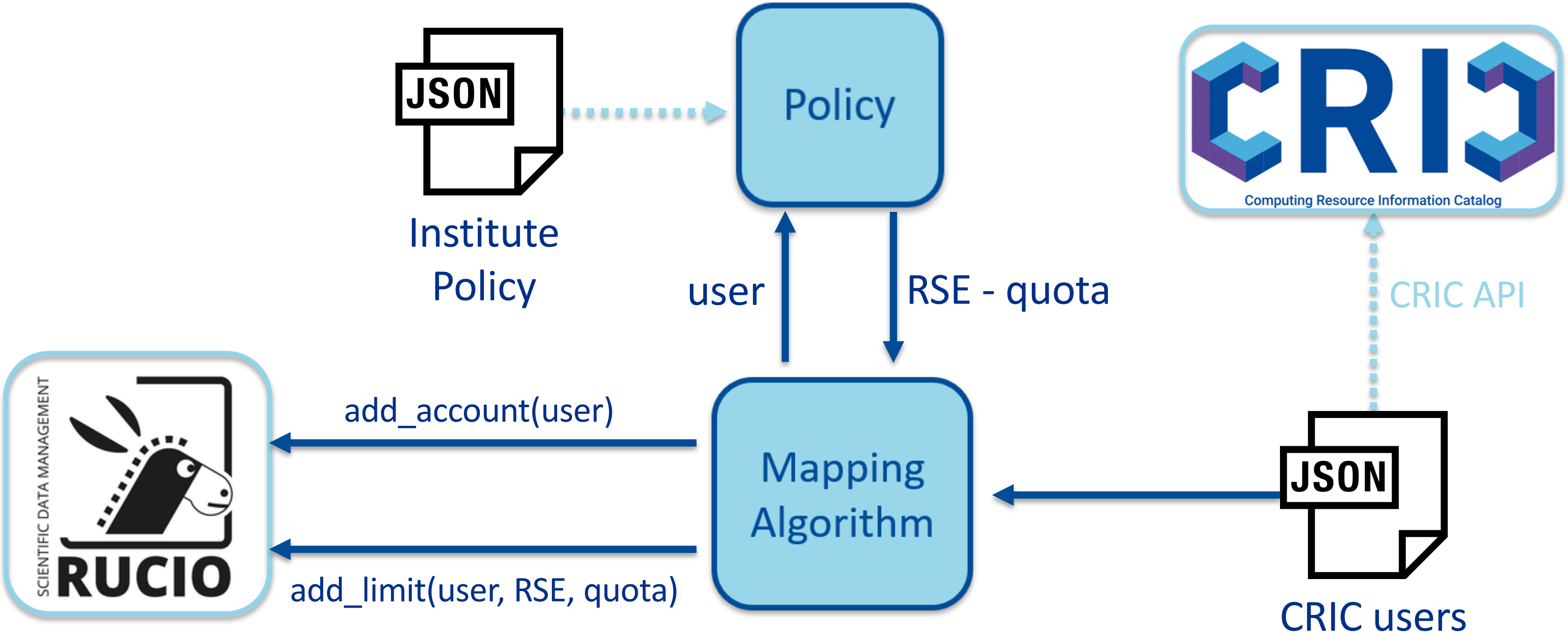


<<Class>>
Quota

- site_name
- quota

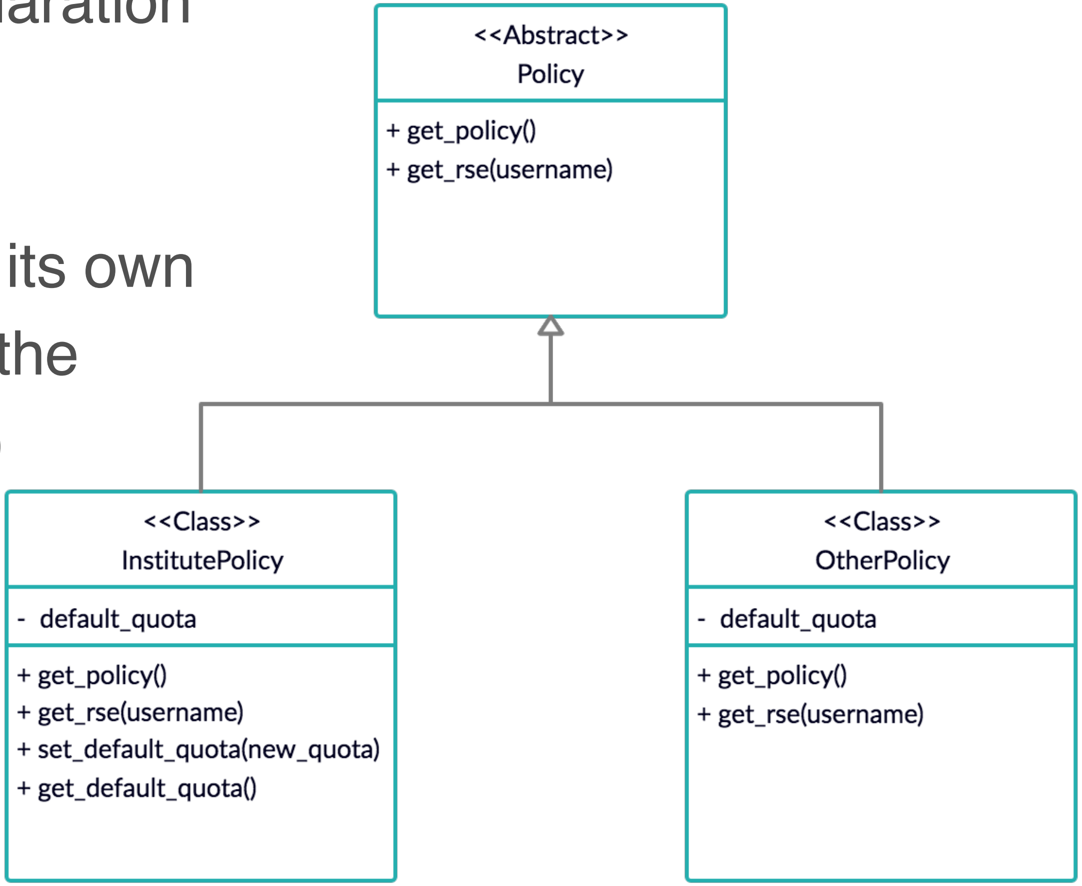
<<constructor>> + Quota
+ get_site_name()
+ get_quota(site_name)
+ set_quota(new_quota)

CRIC and Rucio integration



Policy

- ❑ Abstract Base Class (ABC) Python module to support different policies
  - ○ abstract method has declaration but no implementation
  - ○ each country may define its own policies by implementing the abstract methods (*derive*)



InstitutePolicy class

❑ Implement the US CMS policies

- *get_rse(username)*: given an username, returns the RSE site
    1. if the user is new in CRIC, find the right RSE for him
    2. if the user is not new and already has a quota set by a site admin → do not overwrite it
- *get_default_quota()*: return the current default quota
- *set_default_quota(new_quota):* set a new default quota

Testing phase - preliminary actions

❑ obtain an empty Rucio development instance (*cms-rucio-testbed*)

❑ populate with worldwide CRIC users through an import script

❑ populate with some RSEs through an already existing import script:

- ○ export the RSEs from the *cms-rucio-dev* istance
- ○ import them into the empty *testbed*

❑ develop tests to asses the correctness of this import/export tool → help the Rucio dev team to test new features

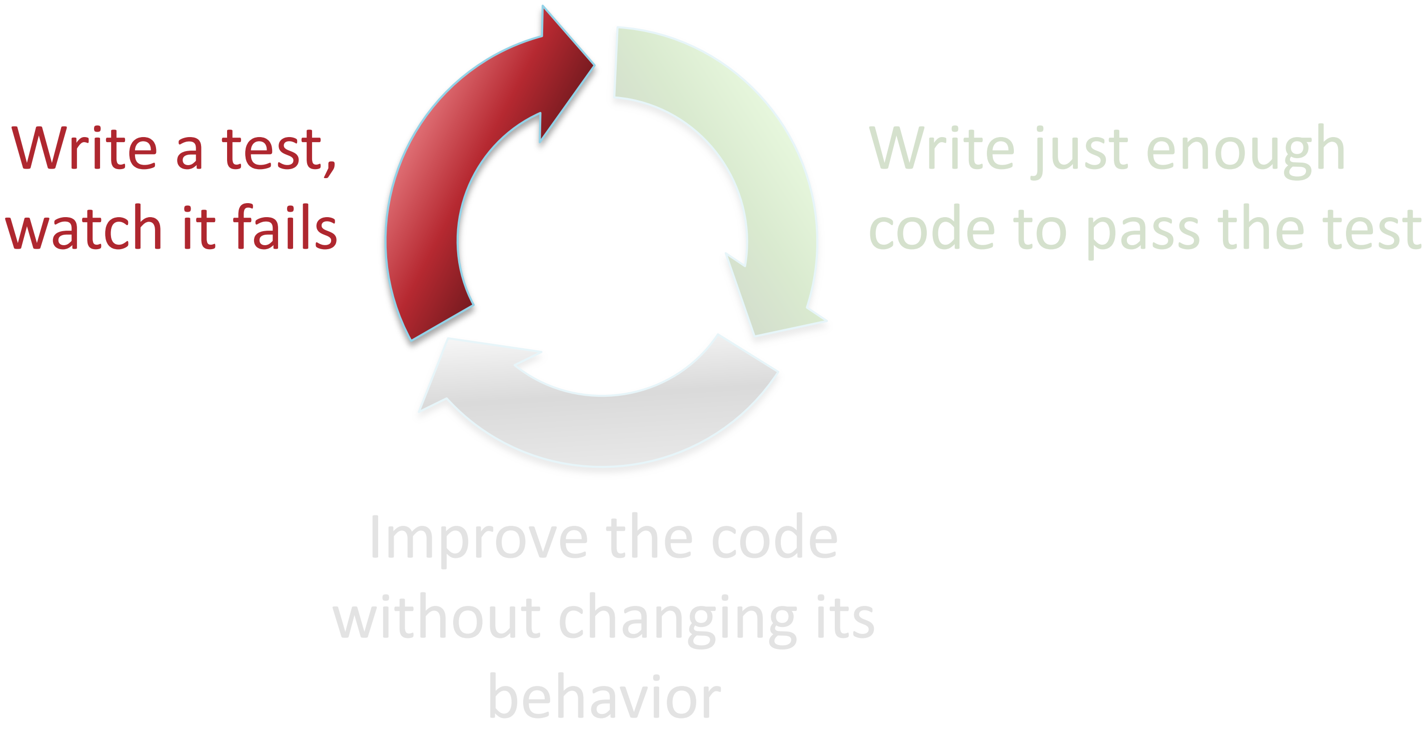
Testing phase - Test-Driven Development

❑ requirements are turned into very specific test cases, then the software is improved so that the tests pass

❑ relies on the repetition of a very short development cycle

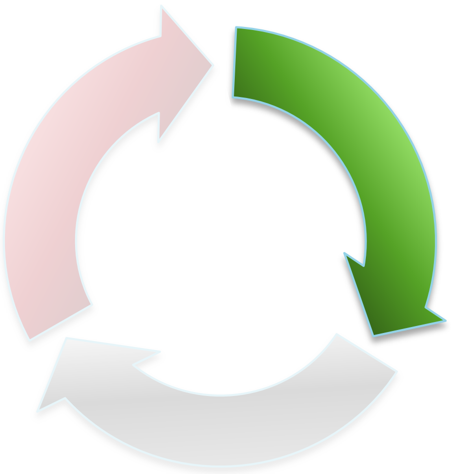
Testing phase - Test-Driven Development

- ❑ requirements are turned into very specific test cases, then the software is improved so that the tests pass
- ❑ relies on the repetition of a very short development cycle


Write a test, watch it fails
Write just enough code to pass the test
Improve the code without changing its behavior

Testing phase - Test-Driven Development

- □ requirements are turned into very specific test cases, then the software is improved so that the tests pass
- □ relies on the repetition of a very short development cycle



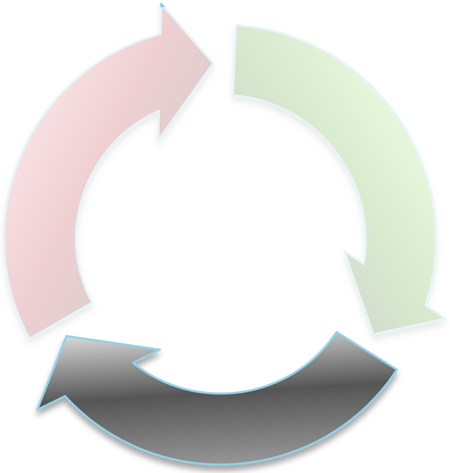
Write a test, watch it fails

Write just enough code to pass the test

Improve the code without changing its behavior

Testing phase - Test-Driven Development

- ❑ requirements are turned into very specific test cases, then the software is improved so that the tests pass
- ❑ relies on the repetition of a very short development cycle

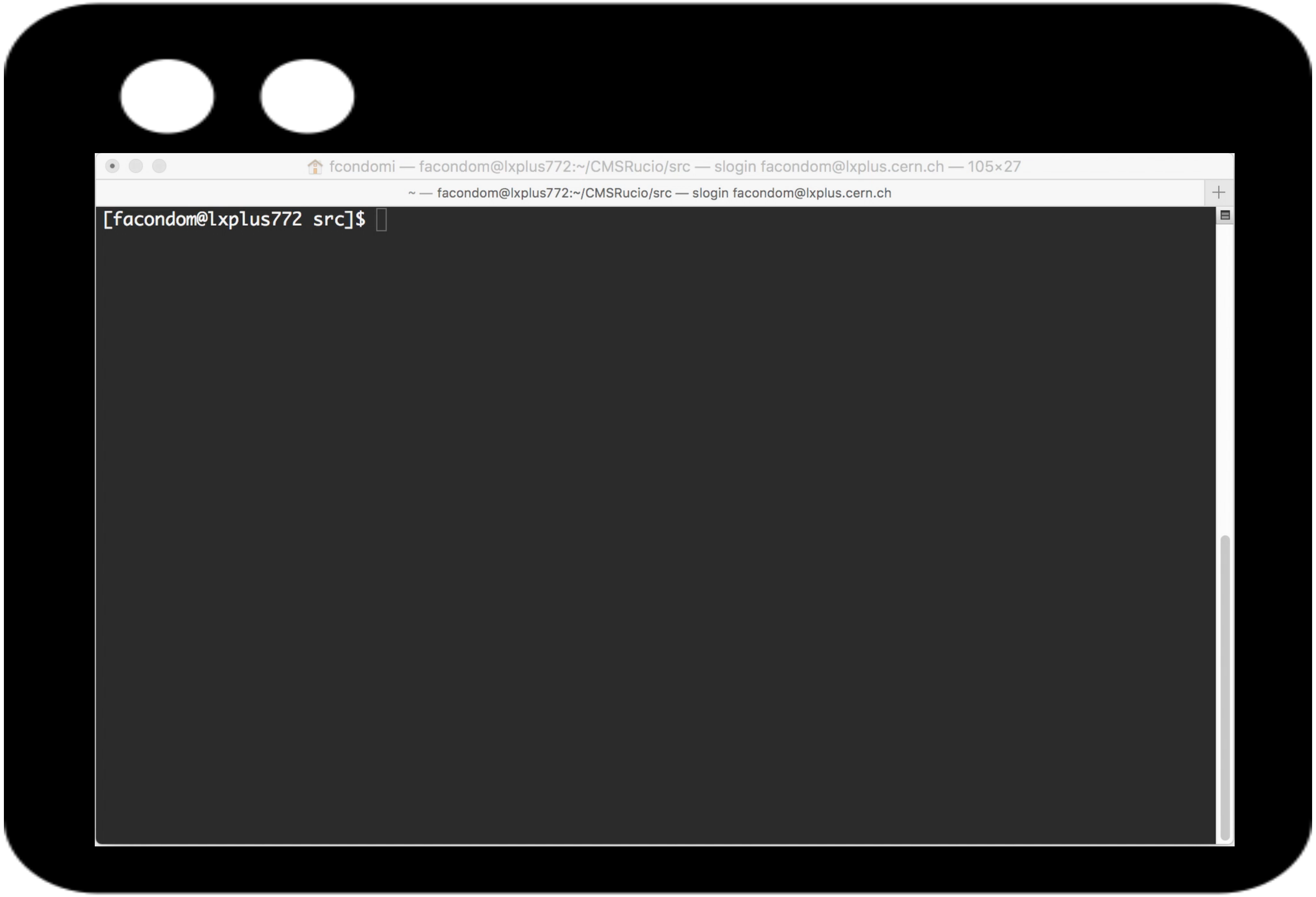


Write a test, watch it fails

Write just enough code to pass the test

Improve the code without changing its behavior

Demo



fcondomi — facondom@lxplus772:~/CMSRucio/src — slogin facondom@lxplus.cern.ch — 105×27

~ — facondom@lxplus772:~/CMSRucio/src — slogin facondom@lxplus.cern.ch

```
[facondom@lxplus772 src]$
```

Summary

❑ Objectives achieved:

- automate quota assignment ✓
- formalize LPC Policies ✓
- generalization for different policies ✓
- tests ✓

❑ How the mapping algorithm will be used:

- now → first testing with Rucio
- mid term → setting user quotas for Rucio in production
- long term → incorporated in CRIC directly (requires CRIC development)

Thanks for the attention.

Questions?

References

1) https://cms.cern/collaboration/people-statistics

2) *D. Bonacorsi / Nuclear Physics B (Proc. Suppl.) 172 (2007) 53–56*

3) *https://indico.cern.ch/event/578991/contributions/2738744/attachments/1538768/2412065/20171011_GDB_CRIC_sameNEC.pdf*

4) *https://indico.fnal.gov/event/16010/contribution/2/material/slides/0.pdf*