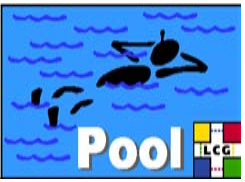



# POOL Integration, Testing and Release Procedure


## Integration

- Packages structure
- External dependencies
- Configuration and build system

## Testing

- Plan
- Policy
- External tools

## Release procedure

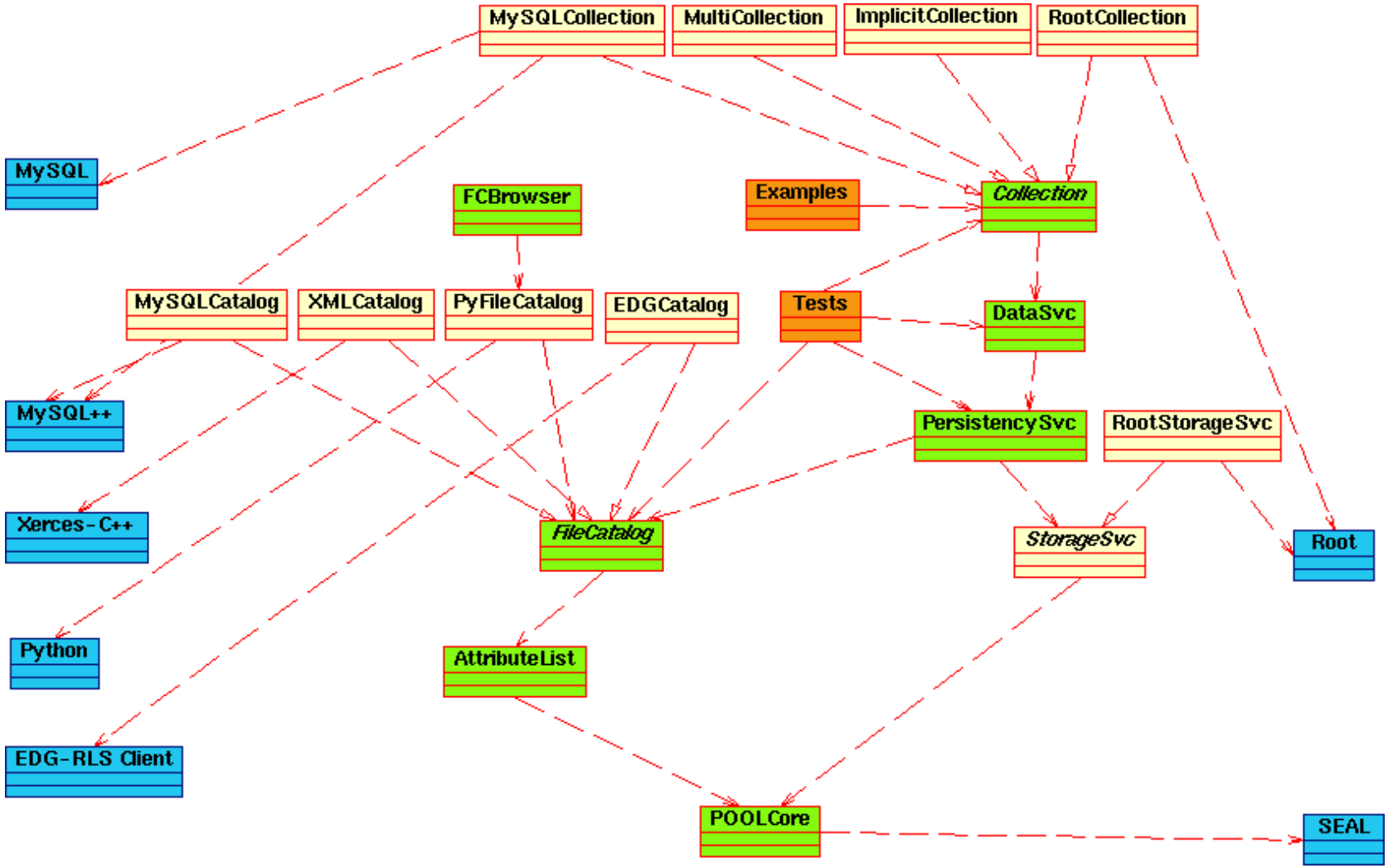
- Component release
- Automatic building

**Giacomo Govi CERN IT/DB**

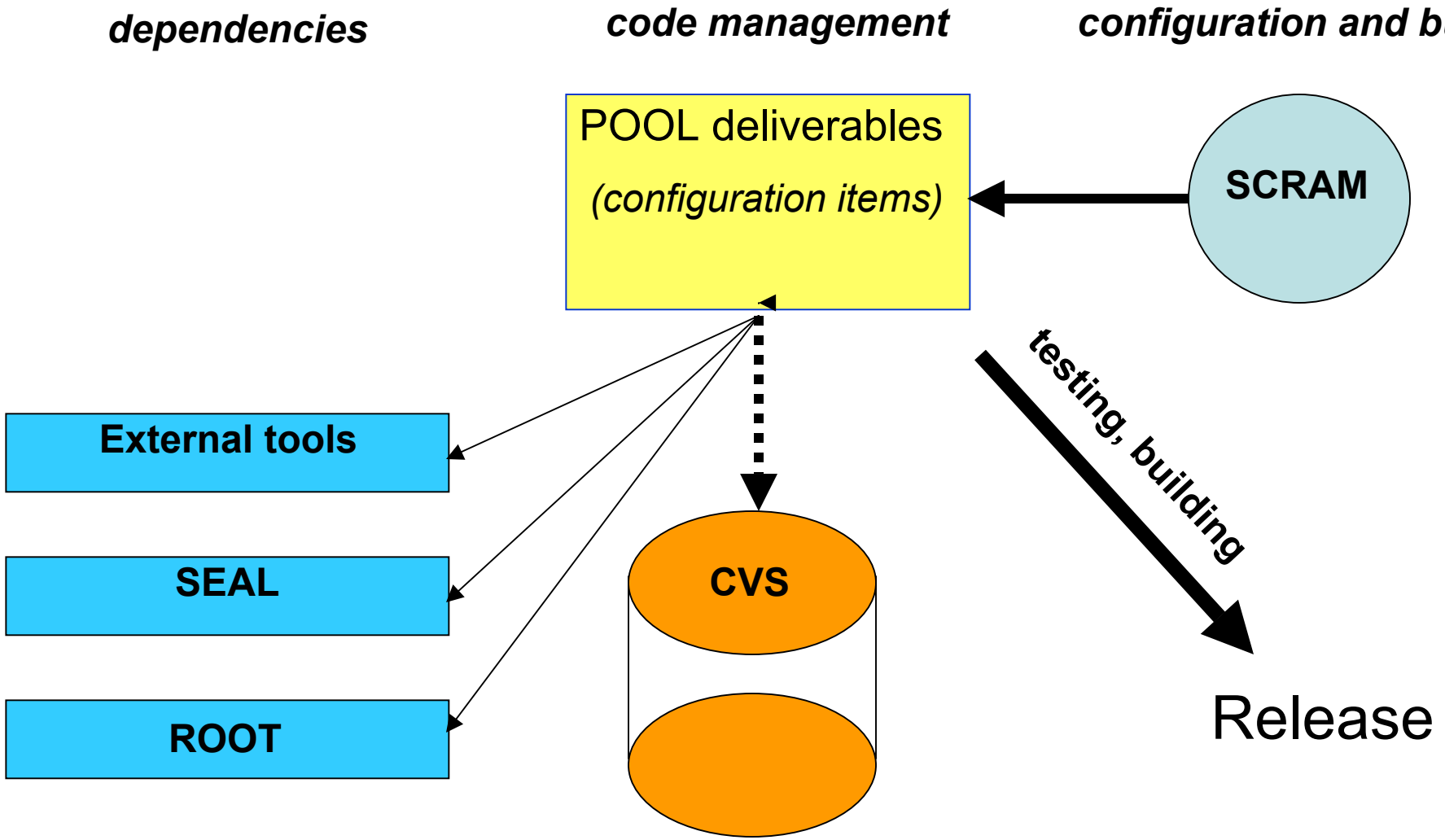
**pool/Infrastructure WP**

**(T.Barras,M.Girone,G.Govi,I.Papadopoulos)**

# Packages dependencies



# POOL as a LCG project



# Configuration items

| | |
|---|---|
| Core components: | -shared library |
| | -unit tests |
| | -command line tools |
| Testing Libraries: | -shared library |
| | -runtime loaded dictionary library |
| Test cases: | -application |
| | -runtime loaded dictionary library |
| Utilities | |
| Scripts | |
| Example Libraries: | -shared library |
| | -runtime loaded dictionary library |
| Example Applications.: | -application |
| | -runtime loaded dictionary library |

# CVS repository structure

Explicit mapping of the *configuration items*:

pool/<Package>/<Package>
/src — *Core component*
/tests

/Tests/Libraries/<Library>/<Library>
/src — *Testing library*
/dict

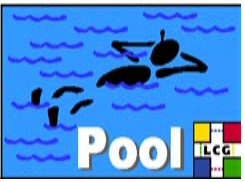
/<TestCase>/src
/dict — *Test cases*

/Examples/Libraries/<Library>/<Library>
/src — *Example library*
/dict

/<ExampleAppl>/src
/dict — *Example application*

/Utilities/<Package> — *Utilities*

/Scripts/<Package> — *Scripts*




# LCG common infrastructure


- Development area, CVS repository and delivery area share the same structure.

The layout of the structure is compliant with a common LCG AA policy (Discussed with SPI and projects and approved by AF).

- External packages are hosted and maintained by SPI

Version upgrade and new installation on request

- SCRAM is the LCG AA configuration and building tool.

POOL only takes care of the configuration for the specific use cases.

External dependencies handled through an LCG AA shared *ToolBox*

# SCRAM

- Scram provides the cross-project and internal configuration:

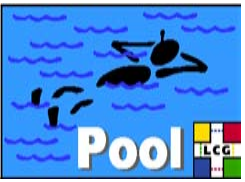
include and library paths, indirect dependencies

- Scram is configured to distinguish the various *configuration items* and associate specific classpath and makefile

Dependencies are exploited in each package *BuildFile.*

Use cases:

- **Pool development**. Only the specific package under development has been checked out. 'Satellite' installation pointing to the release area.
- **External development (a)**. No explicit checkout is in principle needed. 'Satellite' installation.
- **External development (b).** Complete Pool compilation. All the code is checked out.




# Test plan


## Scopes and objectives clearly defined:

### Functionality test:

- *Unit test*: verify single class functionalities
- *Component test*: verify use cases restricted to the component
- *Integration test*: verify functionalities of a subset of components
- *System test*: verify functionalities of the framework as a whole
- *Acceptance test*: verify concrete use cases for which the framework has been designed.

### Performance test:

- CPU usage, memory usage. Comparison with benchmarks

### Regression test on data format

- Ensure compatibility of data format along the pool releases

# Testing policy (I)

Responsibilities

- Test related to a specific component (unit, component, performance):

  the owner of the component

- Integration test of a subset of components:

  the owner of the top level package

- System, acceptance, regression test, planning, infrastructure

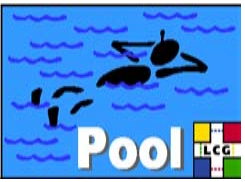
  the infrastructure WP

Dependencies

- Unit test and component test must be designed ‘stand alone’, using a stub or dummy implementation for all the interfaces which is depending on.

Validation criteria

- Assertion of ‘expected’ conditions
- Comparison of output with reference
- Check memory leakage




# Testing policy (II)

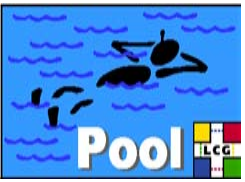

## External tools

- Oval used to steering of test execution
  Allows the comparison with reference output
- CppUnit recommended for condition assertion in the test procedure
- QMtest (soon integrated) allows the common LCG/AA validation (in terms of test execution) of a release

## Deliverables

- An executable (binary exe, scritps)
- A set of input data (optional)
- A reference output for the validation
- A description of the test (aim, use case, set up)

This policy complies with a general LCG/AA testing policy.




# Component release policy


## Component (*configuration unit*) release

- Adopt tagging policy: *UnitName-x-y-z*

  Digits are increasing according to changes in the implementation (z), public interface (y), or in dependencies (x)

  Tags on other formats are private

  Tagging policy explicitly declares dependencies that need to be updated

  Any other tagging format is considered private

- Notification

  Using mail to developer list

  Message includes: tag, dependencies tag, changes

# Pool release procedure

A pool release is a set of CVS tag for each deliverable (*configuration unit*).

Scram does not handle package versioning.

-> POOL has a set of python scripts to assist the release procedure

The procedure is fully automatic consists in:

- Selection of configuration units to be part of the release + CVS checkout of related tags
- Definition of dependencies among units
- Build the whole software ("scram b") for the desired OS/compiler/flags
- Run the tests selected for the validation

-> Integrated in NICOS nightly build system!

A successful set of tags (no compiler errors or test failures) becomes a candidate -> global tag *pool_X-Y-Z_cand_N*

Candidates are produced frequently- delivered as internal releases

They are used:

- By pool developer to get aligned with other component updates
- By the experiment framework integrator for 'beta' testing. Useful to quick feedback.

Public release usually contains major changes. Typically each 6 wks

# Documentation

## General information:

POOL web site:

- Project internal structure, general announcements, meeting minutes
- Release history

Savannah portal

- Bugs, support requests, task lists

## Public documentation:

Workbook

- Howto's, Faq's

File catalogue user guide

## Internal documentation:

Component descriptions

- Design strategies, Implementation specifics, API description

Policies

- Component release and tagging policy, testing policy

## Coming soon:

User guide + Detailed class documentation (for the exposed interfaces)

# Conclusion remarks

The pool team is spending a relevant effort in providing and maintaining the infrastructure for the project.

Policies have been introduced in some common activities (release, testing).

Common LCG/AA policies have been discussed and approved among the projects. Pool complies with them and contributes to their definition.

An automatic release procedure has been implemented internally and contributed very much to the efficient and regular delivery of the new software productions.

Pool has strong interaction with other projects as provider of services (Spi) or software deliverables (Seal, Root), and with the 'final user' experiment software teams.

We think we have established so far a fruitful collaboration with them.

Many thanks to all the people involved!