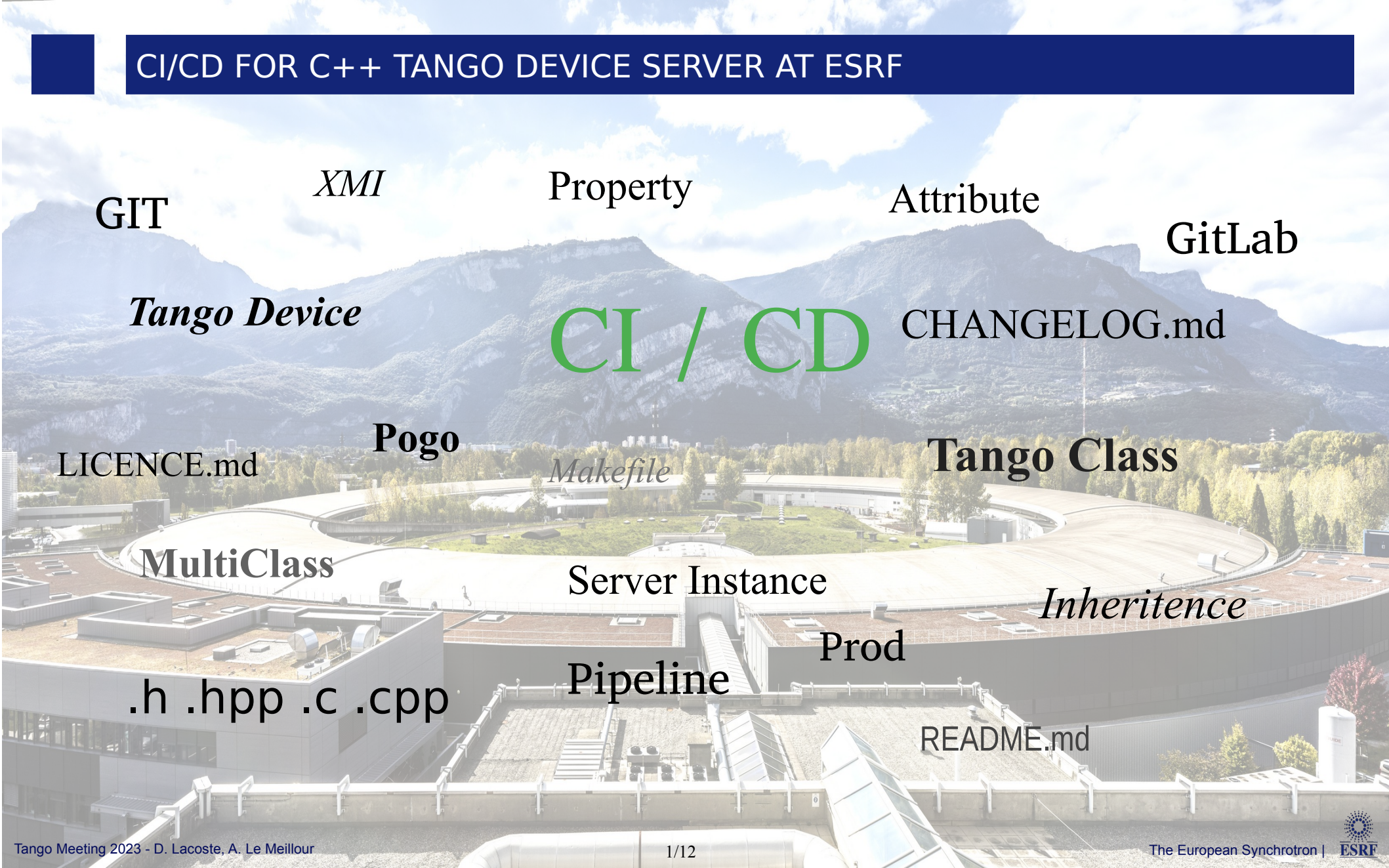
CI/CD FOR C++ TANGO DEVICE SERVER AT ESRF



GOALS OF PROJECT

- Master the life cycle of projects
- Pool resources
- Standardize practices
- Improve maintainability and reusability
- Build and deploy the full control system or part of it
- Keep CI as simple and maintainable as possible

CONTEXT OF THE PROJECT

- Architecture based on centralized file system
- Using Tango Device Server simple-class and multi-class bundled
- Heterogeneity in Gitlab projects organization
- Each developer has his own development methods and habits.

DIRECTIONS CHOSEN

- Redesign of the Tango Device Server structure
- Generate Tango Classes as shared libraries
- Launching Tango Device Servers via Shell scripts that specify the resources used

- Using a centralized CI/CD template that generates dynamic jobs
- We keep a centralized file system but with complete separation of the development environment from the production environment

PREREQUISITE TASKS

- Gitlab Runners
- Docker Images
- Central Filesystem
- Dependencies

CONTENT OF THE GITLAB PROJECT

- New Pogo generator
- Use Cmake target for each component
  - external libraries
  - internal and external Tango classes
  - Tango device servers
- Multiple user configuration files
  - root CmakeLists.txt
  - config-class(es).cmake
  - config-server(s).cmake
- Overload some configurations by environment variables: (Compiler, C++ vers..)
- Headers and sources files are separate
- CHANGELOG.md at Tango class level
- Generation of <lib>Config.cmake at deployment
- Add custom configuration by arg:
  - -DUSER_CONFIG=xxx

git clone -b dyn_rev1 https://gitlab.esrf.fr/accelerators/ci-cd/templates/template-ds.git

GITLAB PIPELINE

Need:

- A solution for hundreds of Device Server C++ projects
- Be able to manage pipelines centrally
- Ability to customize tasks at the project level
- Being able to choose the operating system(s).
- Be able to choose the type or types of build (debug, release..)
- Be able to choose the version or versions of Tango (9.3, 9.4)

Difficulty:

- Complexity of CI Yaml files, to manage

Solution:

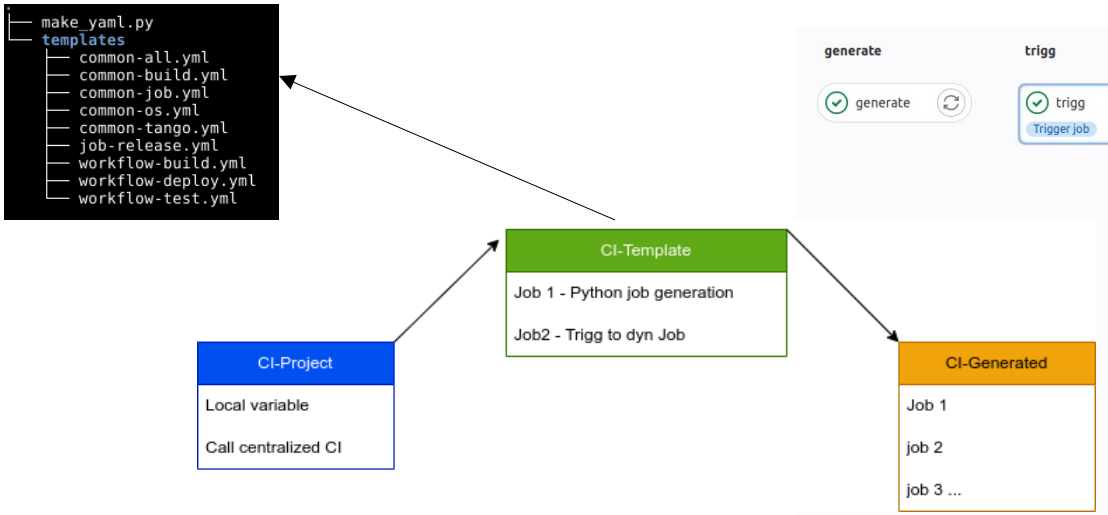
- Use a centralized JOB file
- Use Gitlab's Parent-Child pipeline technique which triggers a downstream pipeline from a job in the main pipeline

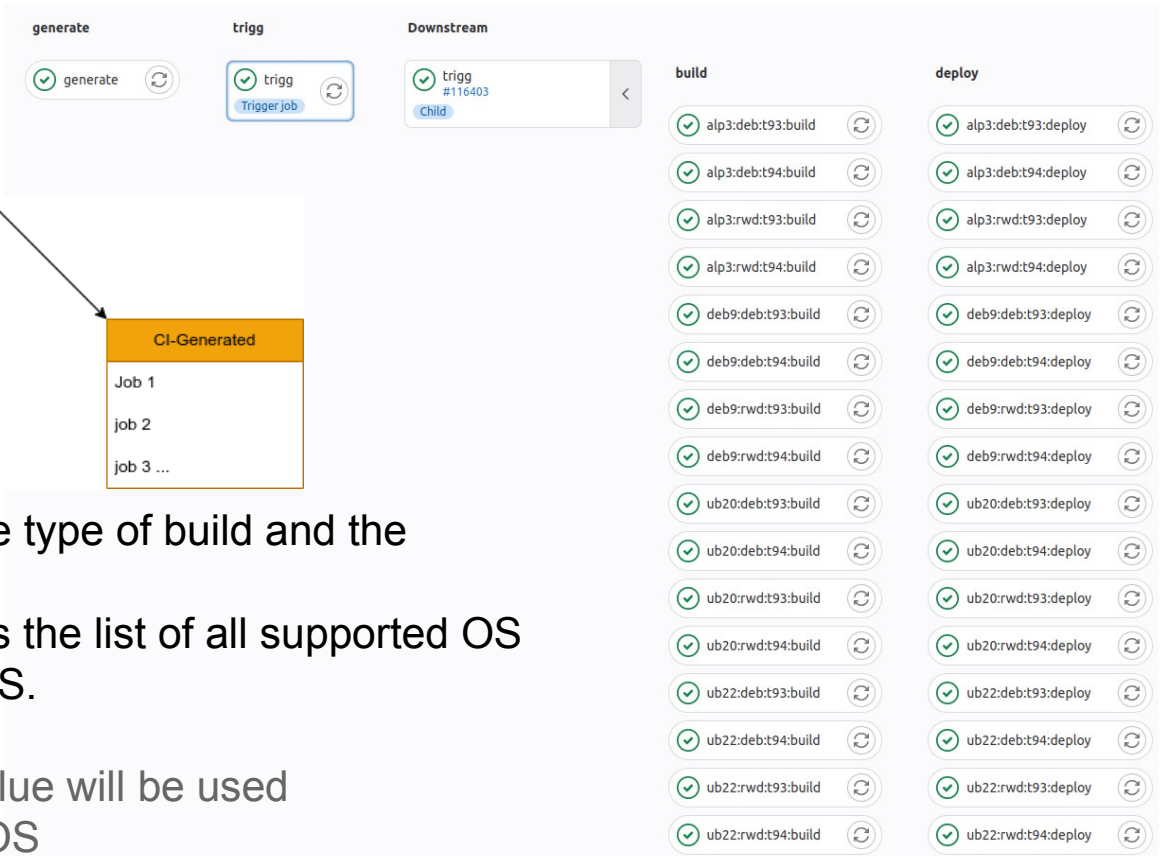
GITLAB PIPELINE

- Limitations of the Parent-Child Pipeline pipeline
  - Loss of some contextual element (MR).

- About our pipeline strategy:
  - Job are triggered during MR (build + test)
  - Job are triggered during Tag = x.x.x (build + deploy)
  - We can force pipeline with Gitlab variables : BUILD_MANUAL, BUILD_AUTO, TEST_MANUEL, TEST_AUTO, DEPLOY_MANUAL, DEPLOY_AUTO, DEPLOY_FORCE.

GITLAB PIPELINE





- It's the same mechanism for the OS, the type of build and the version of Tango.
- In Gitlab we have a variable that defines the list of all supported OS and a second that defines the default OS.
- In the project, the user can
  - do not define anything, the default value will be used
  - Define a list containing a part of the OS
  - or specify 'all' and all OS will be defined.
- At the end, the python script generates a file containing all the jobs.

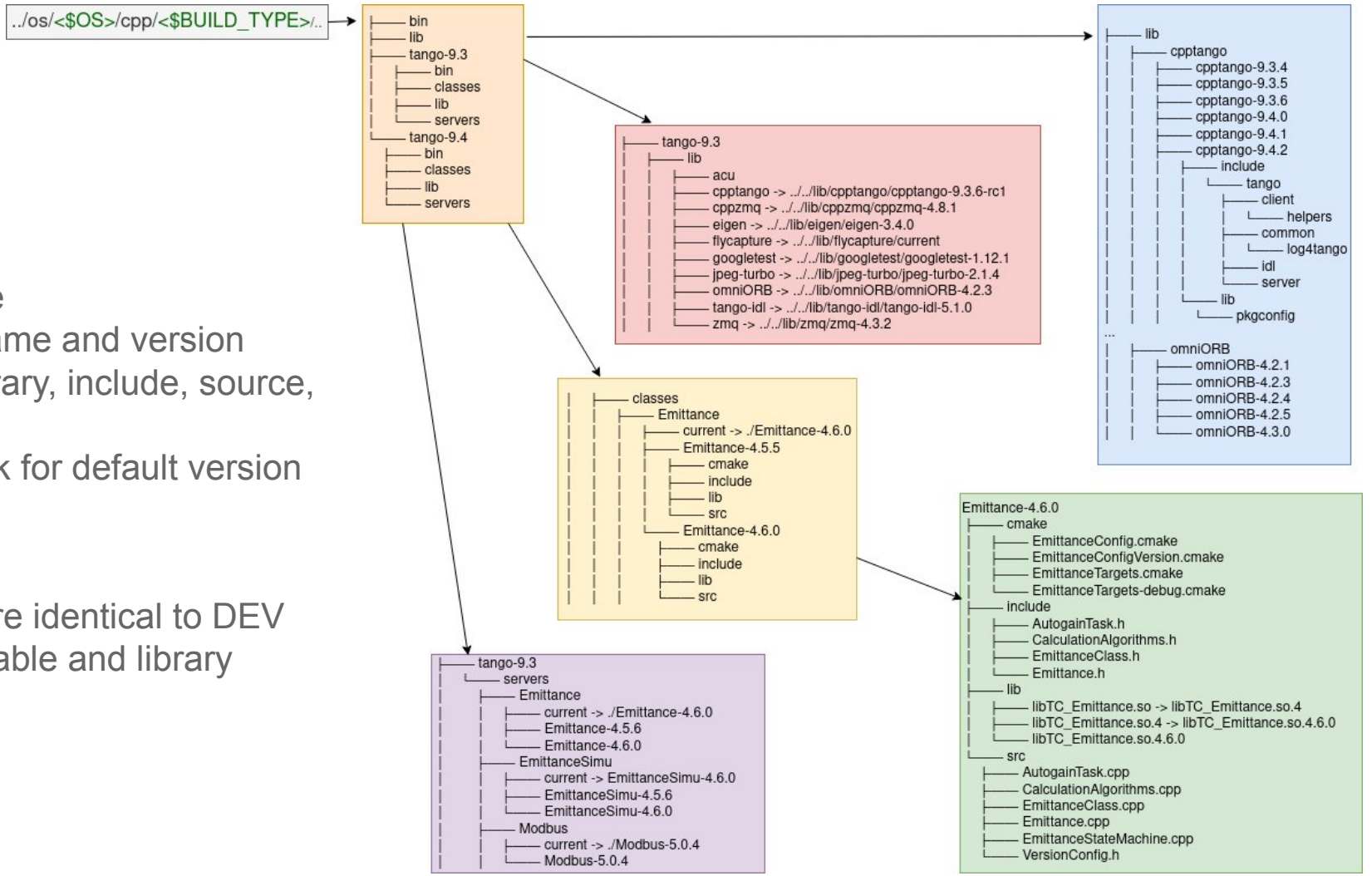
FILESYSTEM DEV / PROD

DEV

- Detailed tree
- Based on name and version
- Contains library, include, source, cmake
- Symbolic link for default version

PROD

- Tree structure identical to DEV
- Only executable and library



TO DO

- Set up a system to easily visualize the Device Server and Tango Classes as well as their dependencies.
- Being able to easily manage the devices versions.
- Trigger Gitlab pipelines on Device Servers dependencies major update.

QUESTION

Thank you !

Any Question ?