HPC Café

Howto on using the Cx services based on the RRZE Gitlab instances

Recap: What is Cx?

- Continuous Integration (CI) is the practice of automatically integrating code changes into a software project. It relies on a code repository that supports automated building and testing. Often, CI also involves setting up a build system from scratch, including all dependencies.
- Continuous Testing (CT) is the practice of executing automated tests as an integral part of the software development process. It tries to make sure that no functionality is lost and no errors are introduced during development.
- Continuous Benchmarking (CB) can be seen as a variant of CT, where not only functionality but also performance is tested in order to avoid regressions, i.e., unwanted performance degradation due to code changes.
- Continuous Deploying (CD) is the automatic deployment of the software coming out of the other Cx processes. This can be the installation on a particular system, rolling out a revision within a whole organization, pushing installation packages to public repositories, etc.

Recap: Why Cx?

- Build: Does it compile?
- Unit Tests: Produces correct results?
- Coverage: Are more tests needed?
- Lint: Is code “well written”?
- Deploy to production

- It’s “free”
- Find bugs earlier
- Encourages test-driven development (write test before code and fore every bug found)
- Find regressions (reintroduction of already fixed bugs)
- Helps contributors get engaged
- (Standardized environment)

Prerequisites using NHR@FAU Cx services

- Project must be hosted on one of the two RRZE Gitlab instances
  - gitlab.rrze with Enterprise features for FAU-internal projects
  - gitos.rrze for projects in the German/European research community (attached to DFN-AAI)
- A valid HPC user account at HPC4FAU and/or NHR@FAU
- Create an SSH-key (no passphrase)

```
$ ssh-keygen -t ed25519 -f id_ssh_ed25519_gitlab
$ ls
id_ssh_ed25519_gitlab    id_ssh_ed25519_gitlab.pub
```

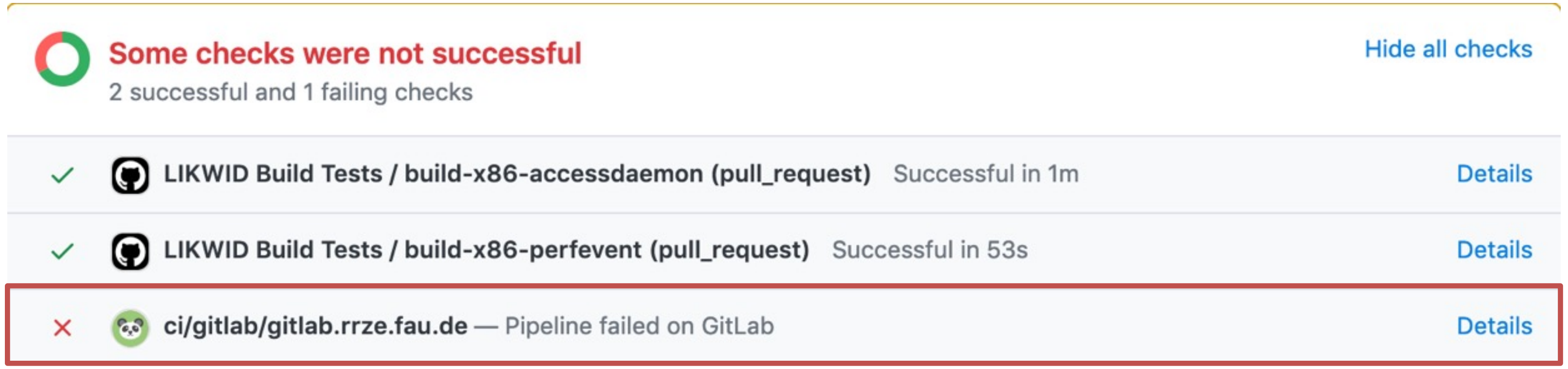
Private key

Public key

Sync remote repository to local Gitlab

- Development repository already at a remote hoster (Github, Gitlab.com)
- Create a synced repository at local Gitlab
- For Github use gitlab.rrze.fau.de
  - New project/repository → Run CI/CD pipelines for external repositories
  - Generate Personal Access Token at Github and copy to Gitlab
  - Select remote repository and local group/name
  - Syncs repo and configures bi-directional integration

Handle local repo as read-only copy. No changes!



Some checks were not successful

2 successful and 1 failing checks

Hide all checks

✓ LIKWID Build Tests / build-x86-accessdaemon (pull_request) Successful in 1m Details

✓ LIKWID Build Tests / build-x86-perfevent (pull_request) Successful in 53s Details

✗ ci/gitlab/gitlab.rrze.fau.de — Pipeline failed on GitLab Details

Also possible with gitos.rrze but much more manual work

Request NHR@FAU Cx service for a repository

- Send email to hpc-support@fau.de with
  - Repository URL
  - HPC account name
  - Public SSH key (should be a key used only for Cx)

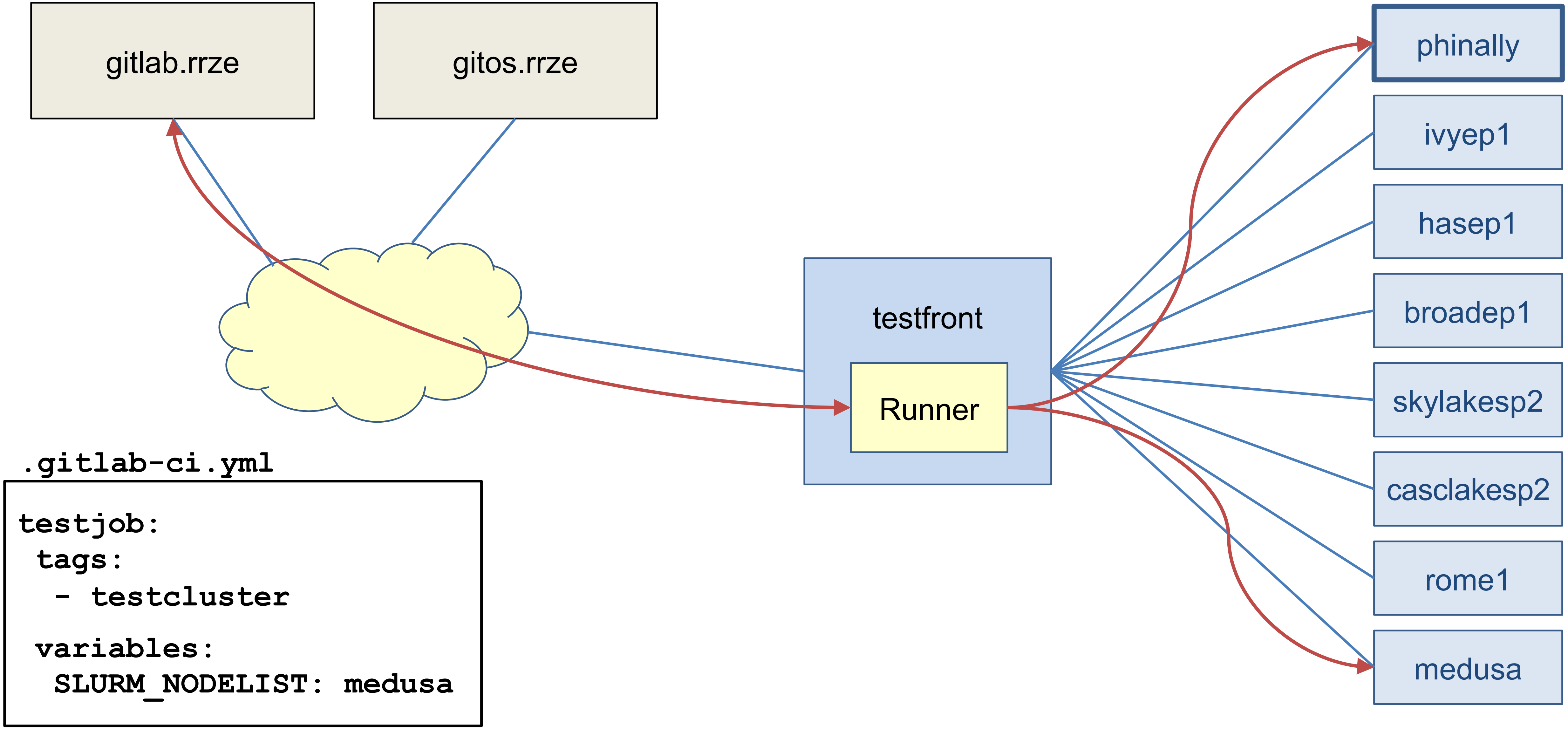
If you use your common SSH key, others might be able to login with your credentials

- In the repository (Settings → CI/CD)
  - Runners: Activate shared HPC runner
  - Variables:
    - Create **`AUTH_USER`** with HPC account name
    - Create **`AUTH_KEY`** with private SSH key (just copy&paste)
  - Create **`.gitlab-ci.yml`** from scratch or use the CI editor

SSH-Key pair
(no passphrase)

All MAINTAINERS in the repo can read variables!

NHR@FAU Cx topology

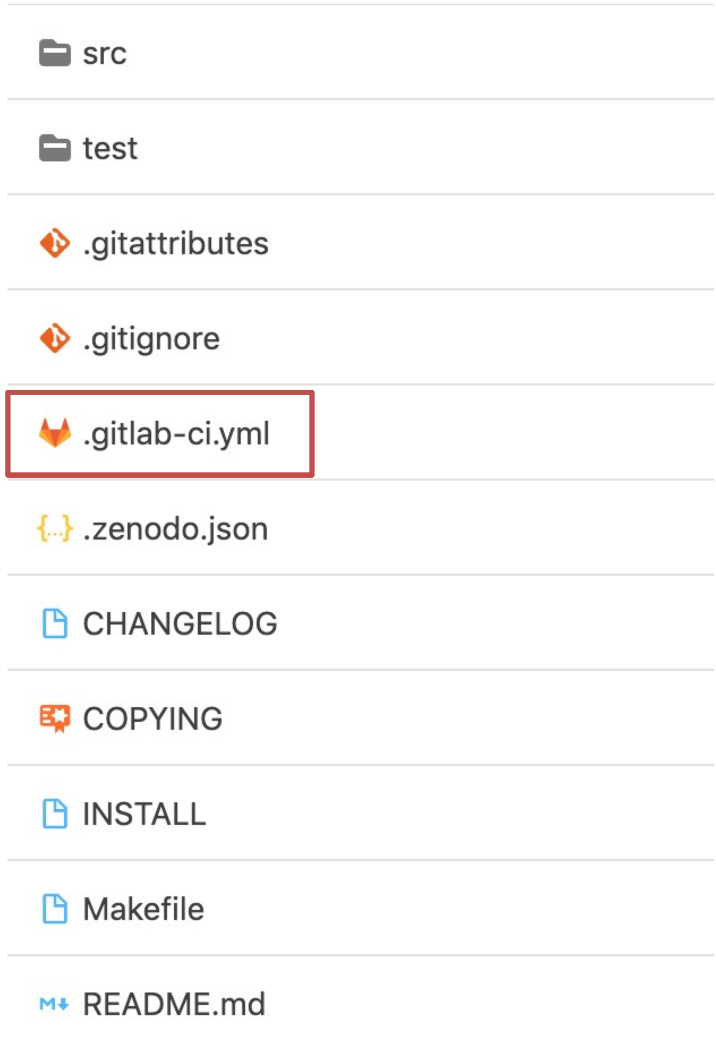


.gitlab-ci.yml

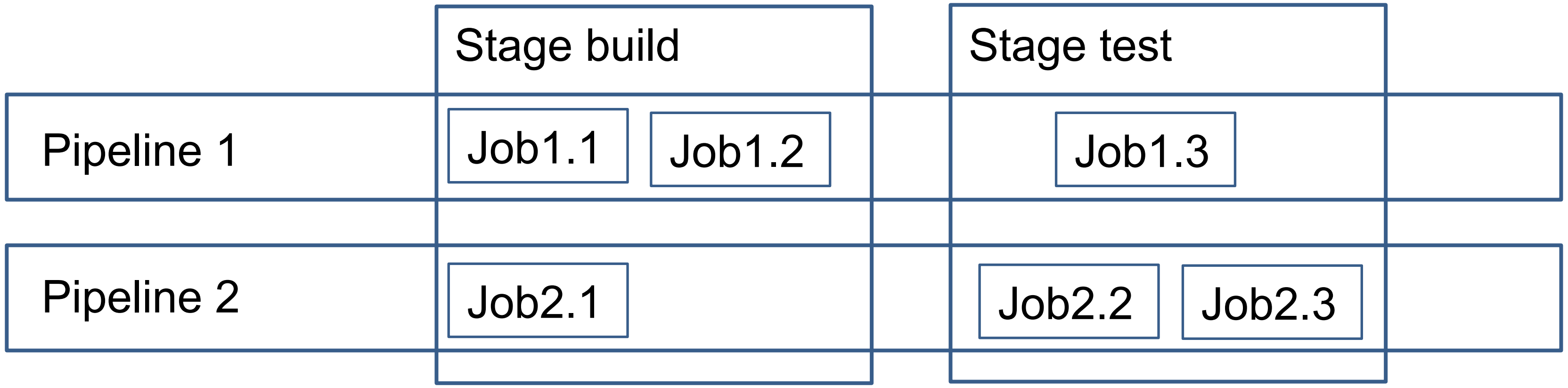
```
testjob:
 tags:
  - testcluster
 variables:
  SLURM_NODELIST: medusa
```

The `.gitlab-ci.yml` file

- Central management file for Gitlab CI
- MUST be in the root of the repository
- Contents:
  - Define & run scripts
    (manually triggered and/or automatically)
  - Include other (`gitlab-ci.yml` compatible) YAML files
  - Control serial and/or parallel execution of CI jobs
  - Configure deployment



- src
- test
- .gitattributes
- .gitignore
- .gitlab-ci.yml
- .zenodo.json
- CHANGELOG
- COPYING
- INSTALL
- Makefile
- README.md



	Stage build	Stage test
Pipeline 1	Job1.1 Job1.2	Job1.3
Pipeline 2	Job2.1	Job2.2 Job2.3

My first `.gitlab-ci.yml`

```
variables:
 SLURM_NODELIST: phinally
 SLURM_TIMELIMIT: 120

job1:
 tags:
  - testcluster
 script:
  - make
  - ./runtests
```

Global variables
(overwritten by job-specific variables)

Runs `make && ./runtests` using NHR@FAU Cx services

→ Runs on host phinally for maximal 2h

For starters: Use Gitlab CI editor, it does syntax checks

Official Gitlab CI/CD documentation

What about the build system?

- NHR@FAU Cx services run on bare-metal hardware (BIOS and OS settings might change without notice!)

- The job is submitted with the given HPC account → Job script can access the user‘s data (**`$HOME`**, **`$WORK`**, …)

- All modules are usable inside jobs (**`module use X`**)
- Dependency installation only into user‘s directories

```
$ pip install --user X
```

In the future: Spack package manager for user-local installations

- Best Practice: Install everything below **`$CI_PROJECT_DIR`** OR cleanup all installed files in **`after_script`** section

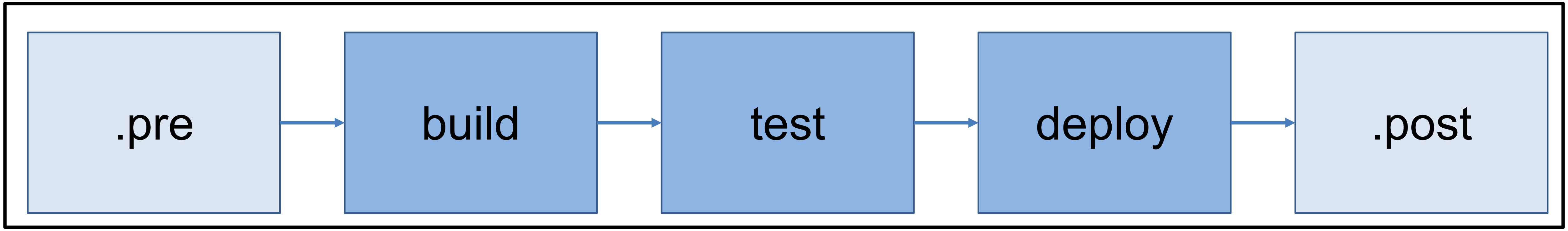
My second `.gitlab-ci.yml`

```
job-intel-AVX512:
 variables:
  SLURM_NODELIST: skylakesp2
 tags:
  - testcluster
 script:
  - module load intel64
  - icc -O3 -xAVX512 tests.c
  - ./a.out out.log
  - ./verify_result_avx512.sh
```

```
job-gcc-AVX:
 variables:
  SLURM_NODELIST: broadep2
 tags:
  - testcluster
 script:
  - module load gcc
  - gcc -O3 -mavx tests.c
  - ./a.out out.log
  - ./verify_result_avx.sh
```

Two jobs that build and test the code on two nodes with different compilers and vectorization.

Recap: Cx stages in Gitlab

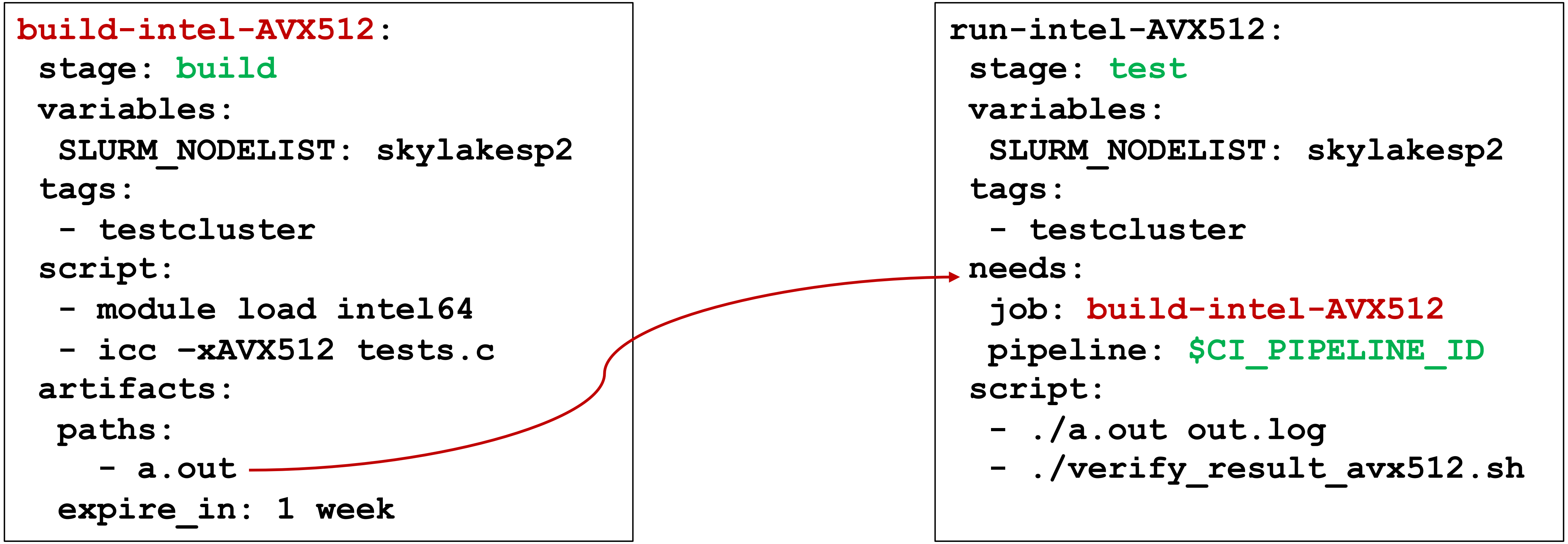


.pre → build → test → deploy → .post

- CI Pipelines consist of multiple stages
- Stages and their order can be self defined with **`stages`** keyword
- My stages:
  - **`.pre`** : Do basic checks like input file formats (JSONlint, YAMLlint, …)
  - **`build`** : Setup build system, build application, store as artifact, cleanup
  - **`test`** : Get artifact, setup runtime(!) system, run application tests
  - (**`deploy`** : push to package indices like PyPI)

Storing intermediate results as artifacts

- All outcome of a Cx job can be stored as artifact at the Gitlab server
- Reuse artifact by job **`needs`** or **`depends`**



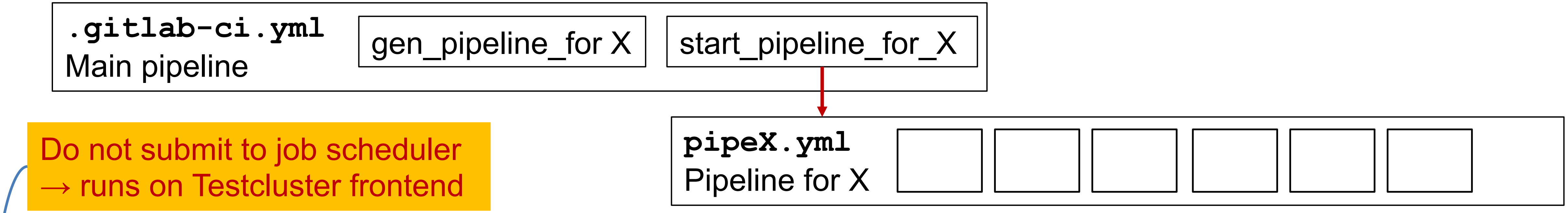
```
build-intel-AVX512:
 stage: build
 variables:
  SLURM_NODELIST: skylakesp2
 tags:
  - testcluster
 script:
  - module load intel64
  - icc -xAVX512 tests.c
 artifacts:
  paths:
    - a.out
  expire_in: 1 week
```

```
run-intel-AVX512:
 stage: test
 variables:
  SLURM_NODELIST: skylakesp2
 tags:
  - testcluster
 needs:
  job: build-intel-AVX512
  pipeline: $CI_PIPELINE_ID
 script:
  - ./a.out out.log
  - ./verify_result_avx512.sh
```

- Recommendation: Use **`expire_in`** with reasonable length

Do I need a job for each X (X={system, cuda, intel64})

- Tedious to write a job for each X and keep it up-to-date
- How to dynamically create jobs?



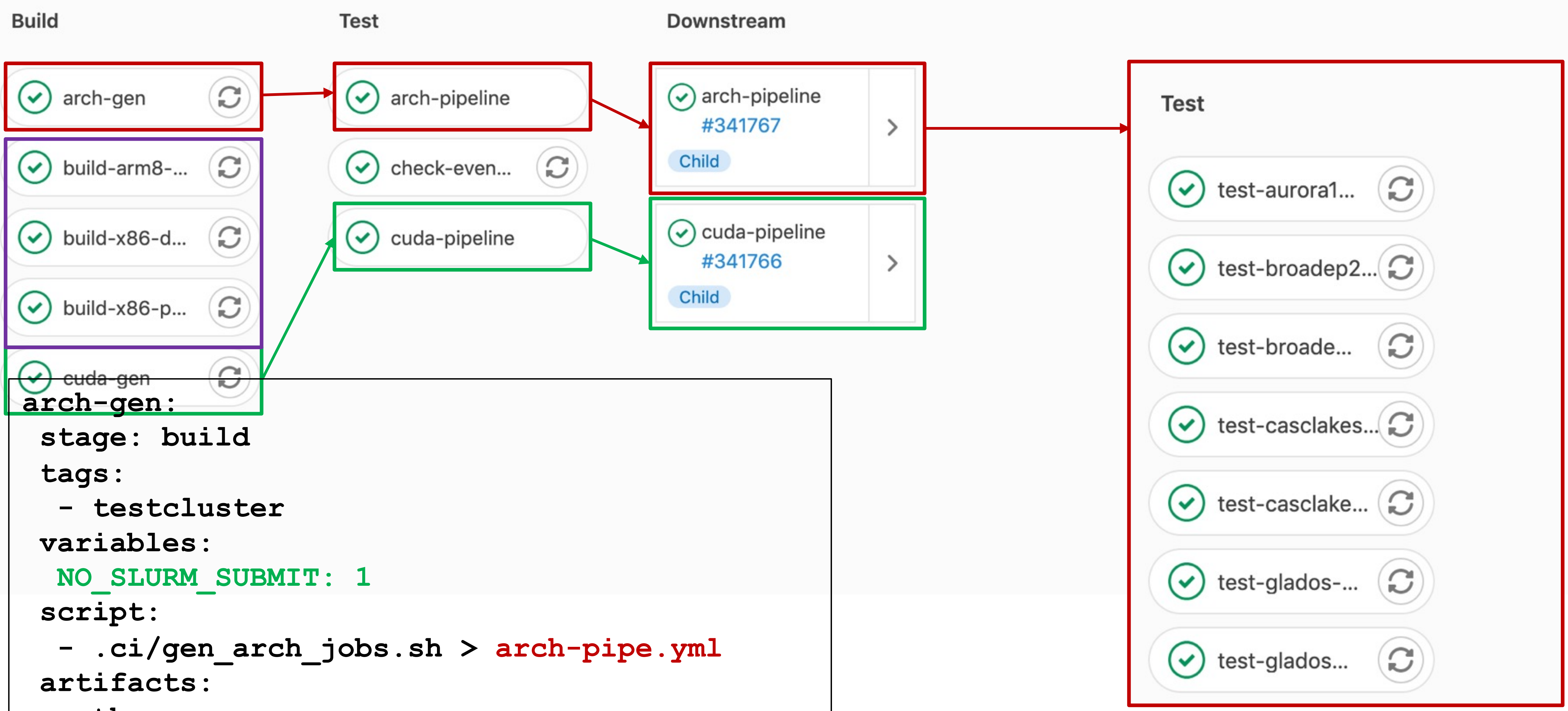
Do not submit to job scheduler
→ runs on Testcluster frontend

```
arch-gen:
 stage: build
 tags:
  - testcluster
 variables:
  NO_SLURM_SUBMIT: 1
 script:
  - .ci/gen_arch_jobs.sh > arch-pipe.yml
 artifacts:
  paths:
   - arch-pipe.yml
```

```
arch-pipe:
 stage: test
 trigger:
  include:
   - artifact: arch-pipe.yml
     job: arch-gen
  strategy: depend
 variables:
  PARENT_PIPELINE_ID: $CI_PIPELINE_ID
```

Required to use artifacts from parent pipeline

Do I need a job for each X (X={system, cuda, intel64})



```
arch-gen:
  stage: build
  tags:
   - testcluster
  variables:
   NO_SLURM_SUBMIT: 1
  script:
   - .ci/gen_arch_jobs.sh > arch-pipe.yml
  artifacts:
   paths:
    - arch-pipe.yml
```

Do I need a job for each X

- For each node:

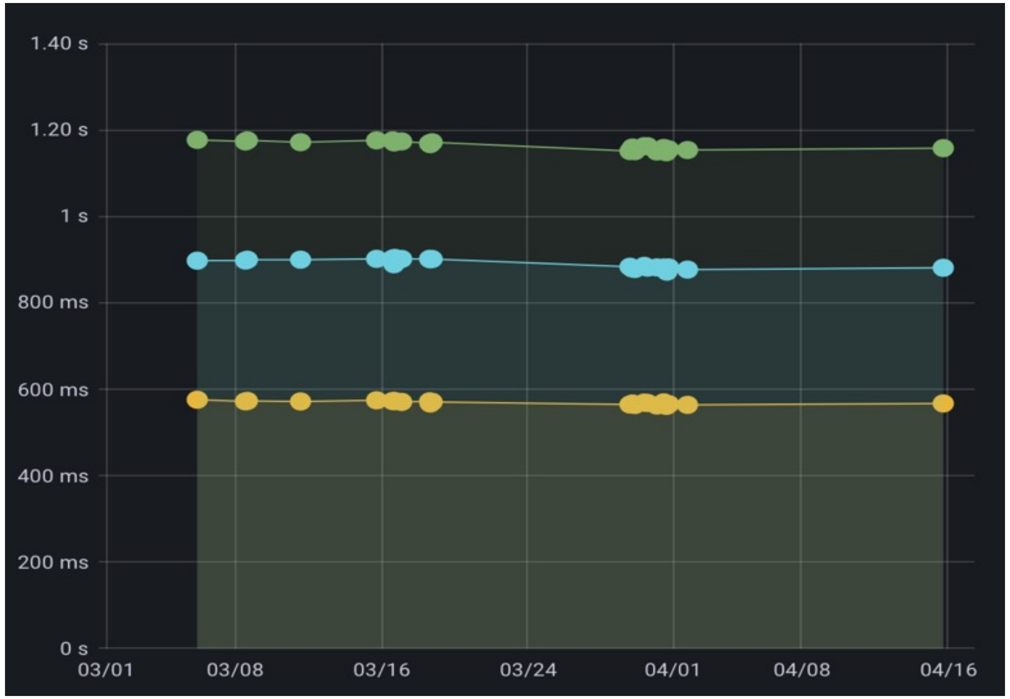
```
for HOST in $(sinfo -h -p work -o "%n"); do
cat << EOF
jobs-$HOST:
 variables:
  SLURM_NODELIST: $HOST
 […]
EOF
done
```

- For each „cuda“ module:

```
for MOD in $(module av -t cuda 2>&1 | grep -E "^cuda" | cut -d ' ' -f 1); do
PMOD=${MOD/\//-} # replace / in module name with - => cuda-X.Y
cat << EOF
job-$PMOD:
 script:
  - module load $MOD
 […]
EOF
done
```

Continuous Benchmarking (Work in progress)

- Benchmarks different parts of the waLBerla software framework developed at the chair for system simulation
  - I.e. Particle dynamics, LBM fluid simulation with generated kernels
- Data persisted in InfluxDB database
- Visualization using Grafana



waLBerla → influxdb → Grafana

Continuous Deployment

- For releases
  - upload the tested version to a registry
  - install to local server system
  - …
- Check Gitlab documentation:
  https://docs.gitlab.com/ee/user/packages/package_registry/
- Recommendation:
  Put all account names and keys as variables in the CI/CD configuration

Summary

- NHR@FAU provides Cx infrastructure for HPC-relevant CI
- Usable from gitlab.rrze and gitos.rrze Gitlab instances
- Test codes that require specific hardware features

- After syncing also available for external repositories
- BUT no virtualized environments
  Install below `$CI_PROJECT_DIR` or cleanup in `after_script`
- Use artifacts to reuse job results

Happy testing!

Probable use-cases

- Architecture-specific software projects
  - Use hardware features
  - Run shared-memory codes
  - Do performance tests
  - Build/Test accelerator code

LIKWID with NHR@FAU Cx
- Synced from Github
- Uses parent-child pipelines

- Build (simple) LaTeX projects (`pdflatex` is installed)

- Maybe in the future: Gitlab runner for MPI jobs