Coding: collaborative, trustworthy, reproducible

Alessandro Corbetta

Plan of the lectures

Structured collaboration

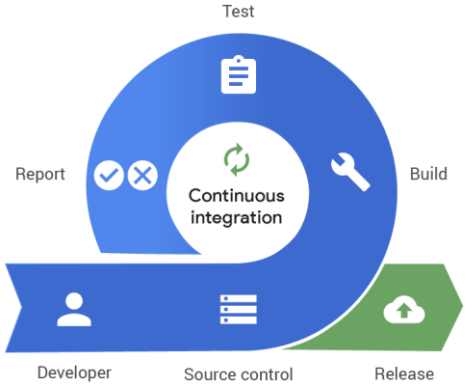
Trusting changes

Immediate reproducibility

Gitlab
Conversational development

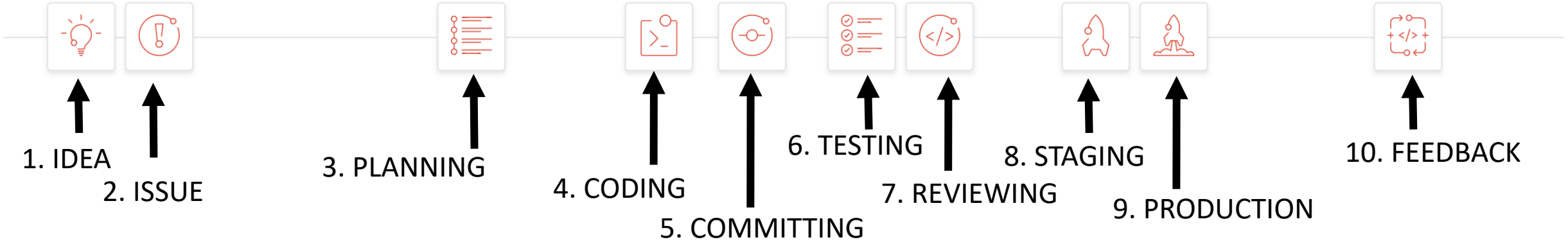
Continuous integration
Continuous deployment

Containers
Docker
Virtualenvs



git ➡ GitLab

The conversational development paradigm



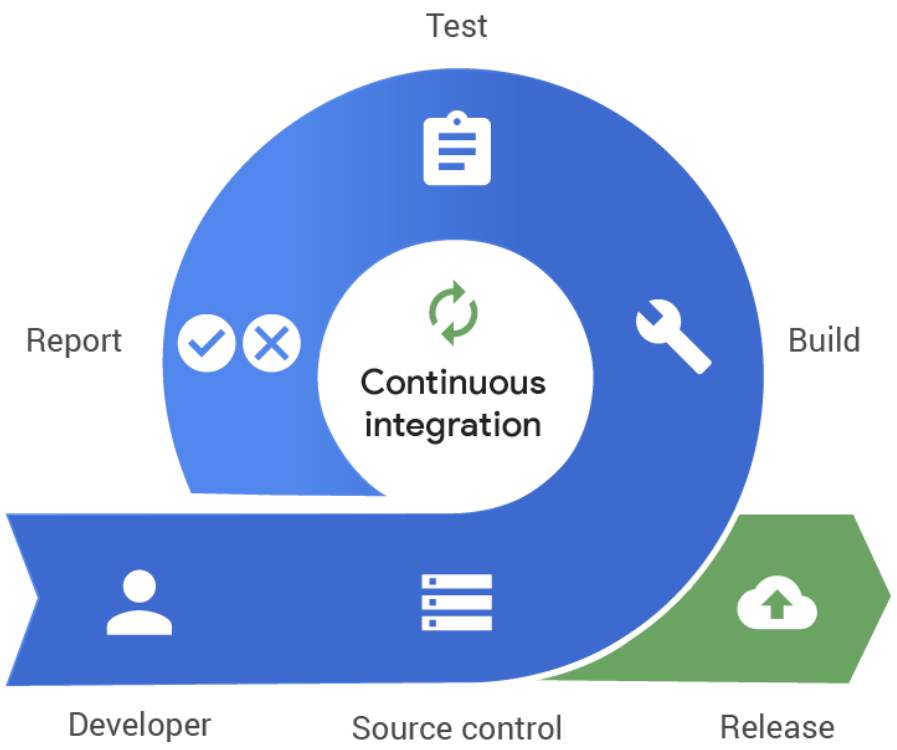
Gitlab/github: meant to support this approach

Development model => content & conversations between developers
Fosters collaborations w/o centralized entities

In my experience: very scalable also in research (codes & execution)!

Continuous integration

- **Each push: automated** remote **testing**
- Every user fully aware of the code state



- If testing quick, dev cycle & master merge: <u>very frequent</u>
  - Many github repos: <u>hundreds</u> merge per day after remote testing

Minimal python example: CI

.gitlab-ci.yml

```
image: python:latest
```

Running in a linux sandbox with up-to-date python (Docker container)

```
test-only:
  script:
    - pip install pytest
    - pytest -vvv
```

Installing dependencies, here put numpy etc…
(better with requirements.txt)

Testing

Search GitLab

P **Pytest Example**

- Project information
- Repository
- Issues 0
- Merge requests 0
- **CI/CD**
  - **Pipelines**
  - Editor
  - Jobs
  - Schedules
- Security & Compliance
- Deployments
- Packages and registries
- Infrastructure
- Monitor
- Analytics
- Wiki
- Snippets
- Settings

passed **Job test-only** triggered 5 minutes ago by Administrato

test-only
test

```
Running with gitlab-runner 15.6.1 (133d7e76)
  on Ci runner NS9smqSy
Preparing the "docker" executor
Using Docker executor with image python:latest ...
Pulling docker image python:latest ...
Using docker image sha256:ee4e7a0f1c354d9996229a765d0785df2671252c1822ae111015d37dcf5f765b for
Preparing environment
Running on runner-ns9smqsy-project-2-concurrent-0 via smr3696-1...
Getting source from Git repository
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /builds/gitlab-instance-59615060/pytest-example/.git/
Checking out 2733dd46 as main...
$ pytest -vvv
============================= test session starts ==============================
platform linux -- Python 3.11.0, pytest-7.2.0, pluggy-1.0.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /builds/gitlab-instance-59615060/pytest-example
plugins: cov-4.0.0
collecting ... collected 2 items
test_code.py::test_summing_f1 PASSED                                     [ 50%]
test_code.py::test_summing_f2 PASSED                                     [100%]
============================== 2 passed in 0.01s ===============================
Cleaning up project directory and file based variables
Job succeeded
```

Coding: collaborative, trustworthy, **reproducible**

Alessandro Corbetta

Reproducibility

extent to which consistent results are obtained when an experiment is repeated.

Reproducibility

extent to which consistent results are obtained when an experiment is repeated.

Yo, man..
Your code is broken

Reproducibility

extent to which consistent results are obtained when an experiment is repeated.

Yo, man..
Your code is broken

Psst..
it **runs perfectly** on **my machine**!

Reproducibility

extent to which consistent results are obtained when an experiment is repeated.

Yo, man..
Your code is broken

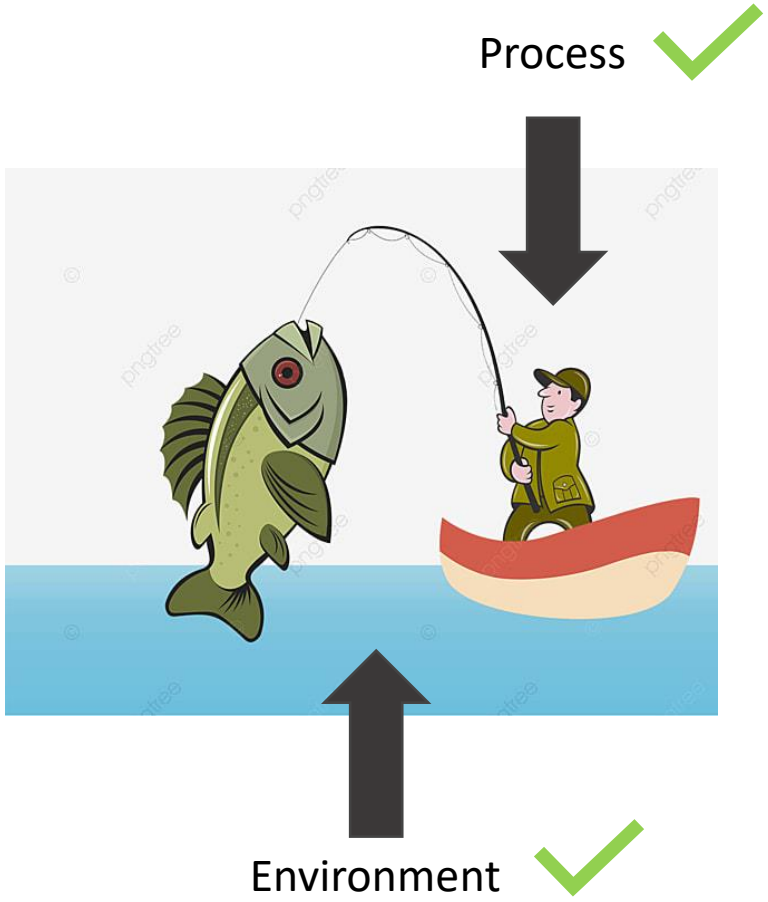
Psst..
it **runs perfectly** on **my machine**!

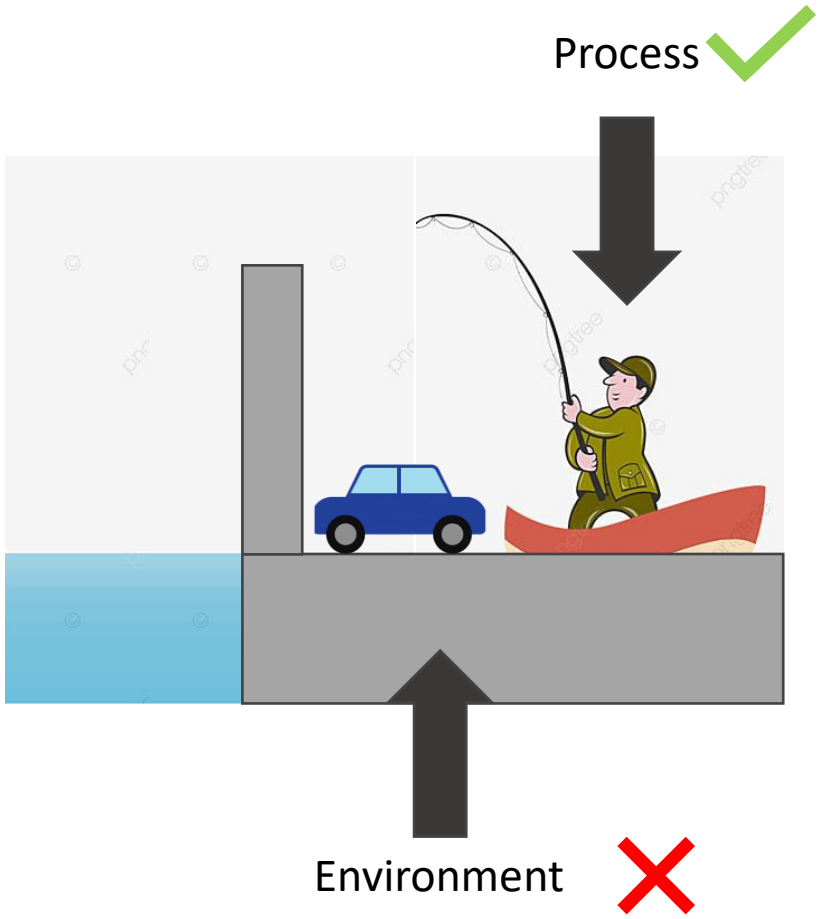
Yo…for real??
But which versions…

Reproducibility

- **Process**
  Makefile

- **Environment**
  ??





Aims

- Predictable code deployment
- Streamline execution in collaborative environment
- Reproducible science
- Byproducts:
  - Sandboxed execution
  - Preventing conflicts with system-wide dependencies

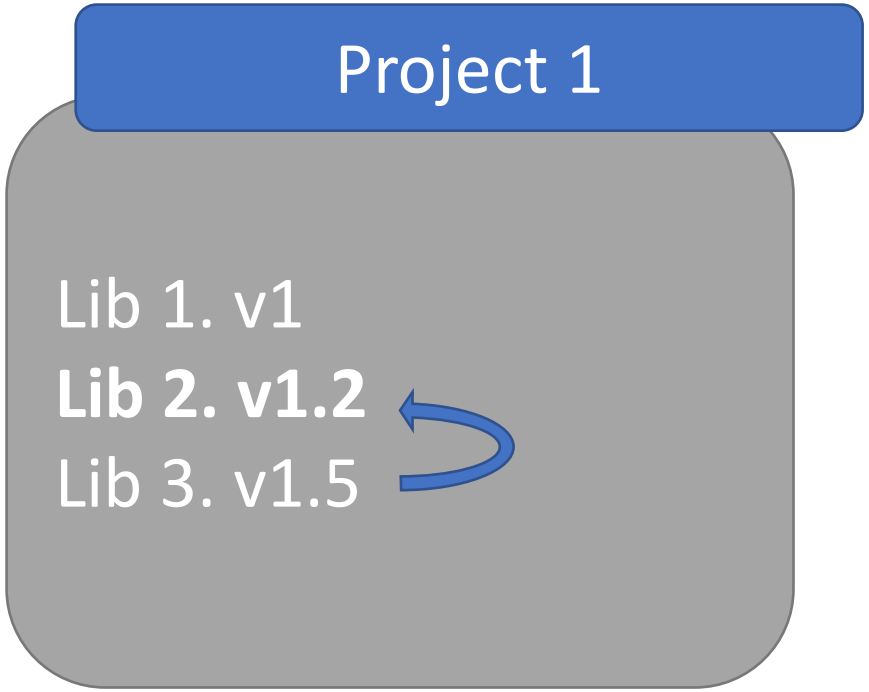
Reproducibility

- Environment notion:
  python dependencies
  - Venv, virtualenv, conda env

- Environment notion:
  OS / complete environment
- Light-weight sytem-level sandboxing

Python virtual environments

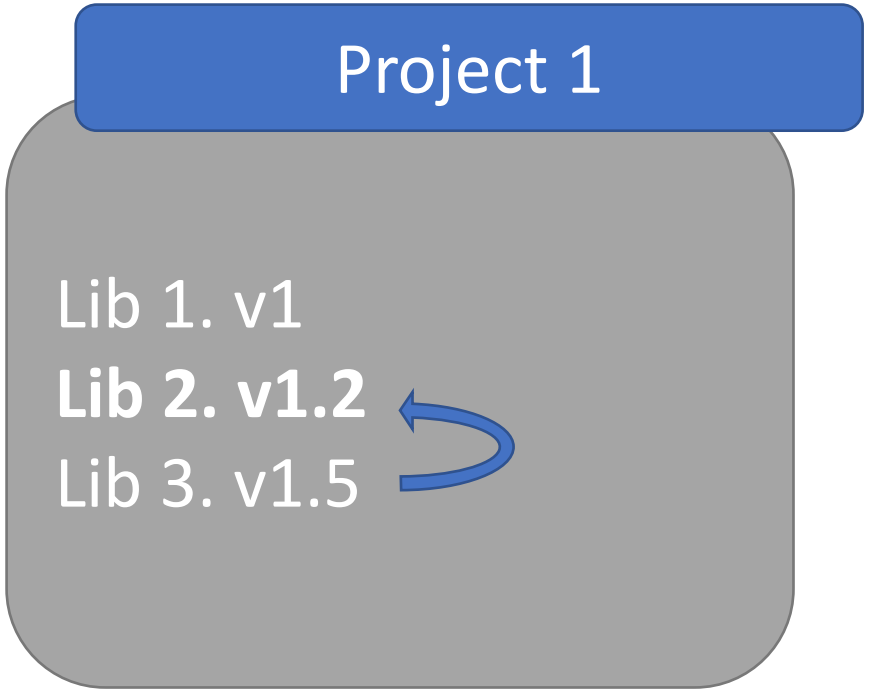
ISSUE 1: having control on the actual project dependencies (and sub-dependencies)



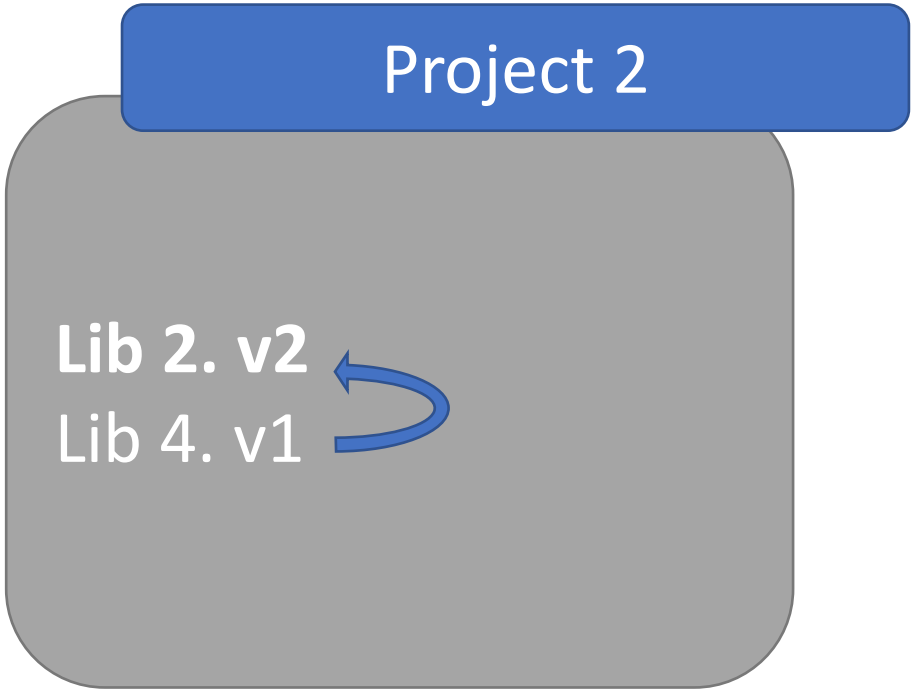
pip install lib_2==1.2

Python virtual environments

ISSUE 2: Allowing projects with conflicting dependencies



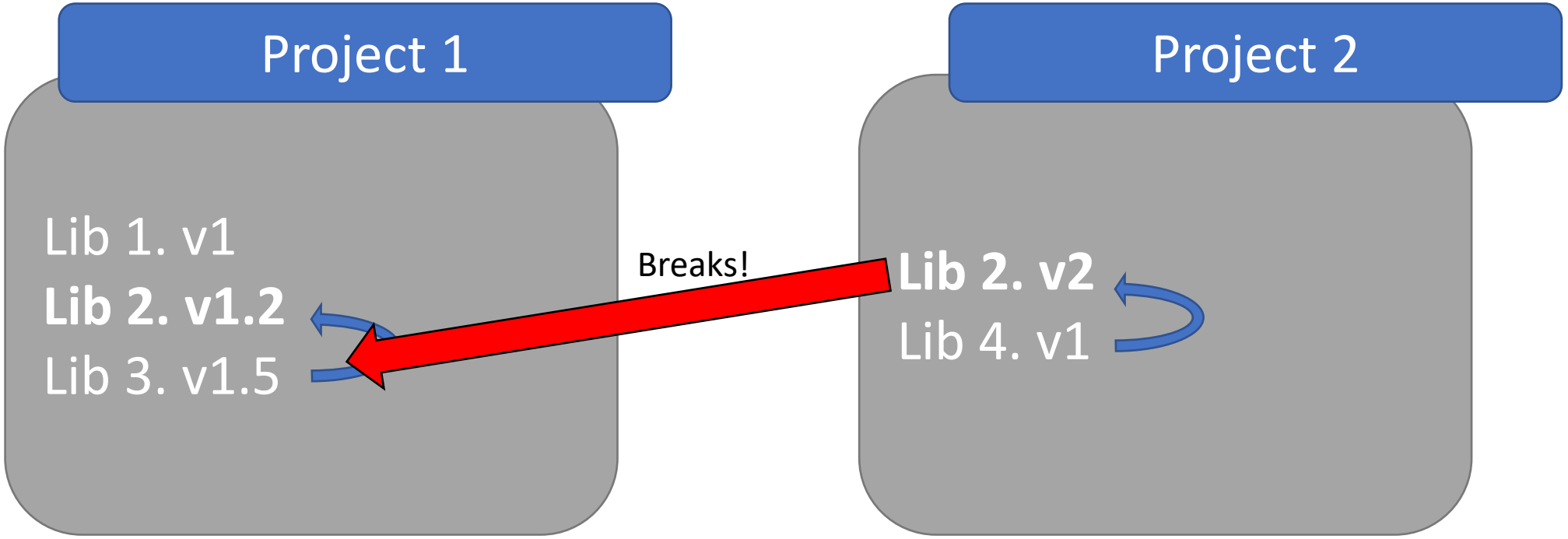
pip install lib_2==1.2



pip install lib_2==2.0

Python virtual environments

ISSUE 2: Allowing projects with conflicting dependencies

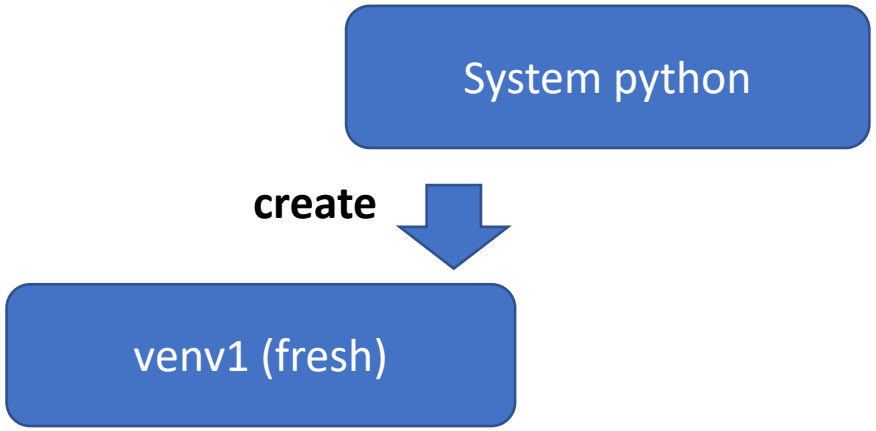


pip install lib_2==1.2

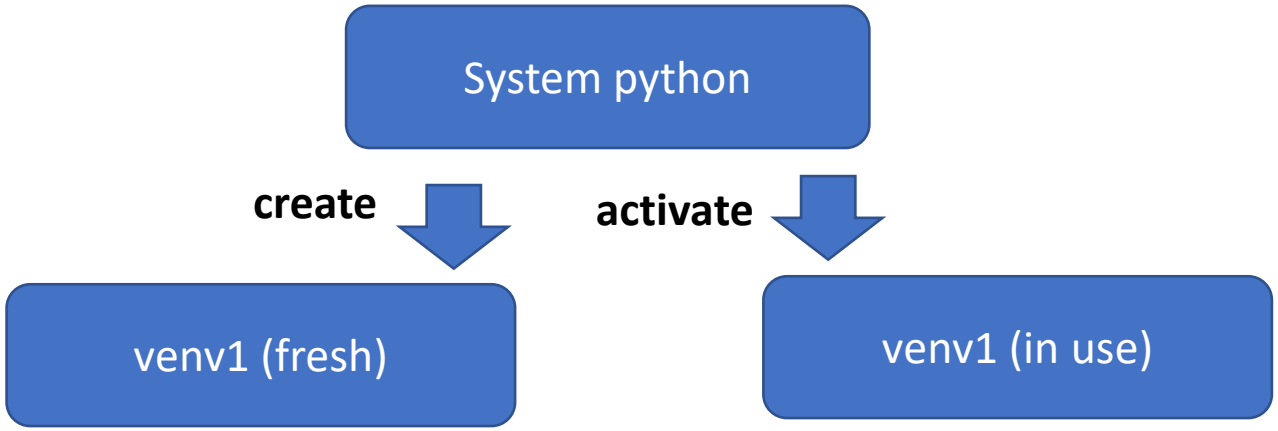
pip install lib_2==2.0

Virtual environment pattern

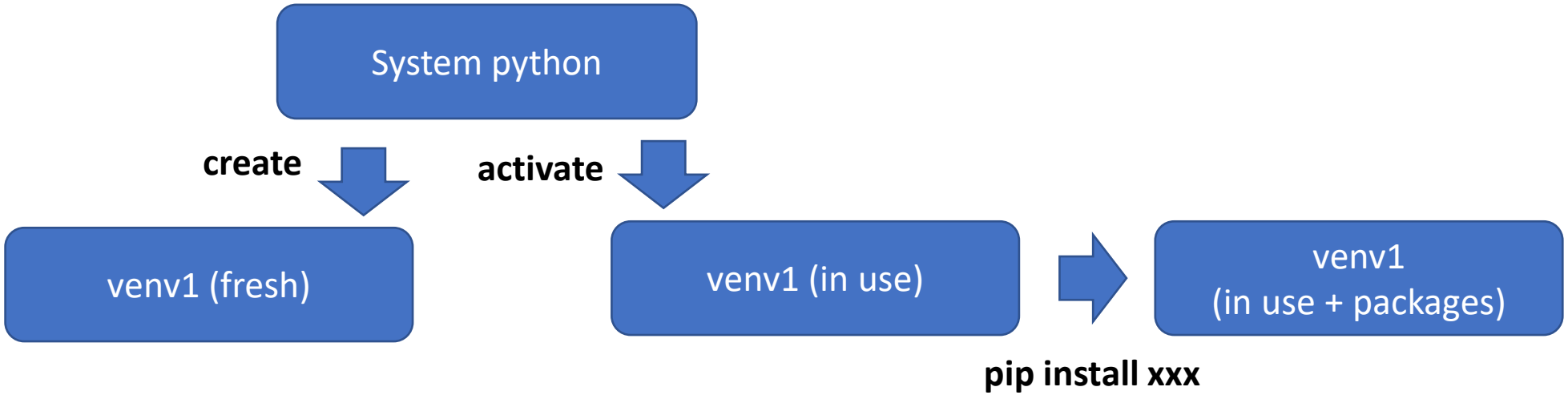
Virtual environment pattern



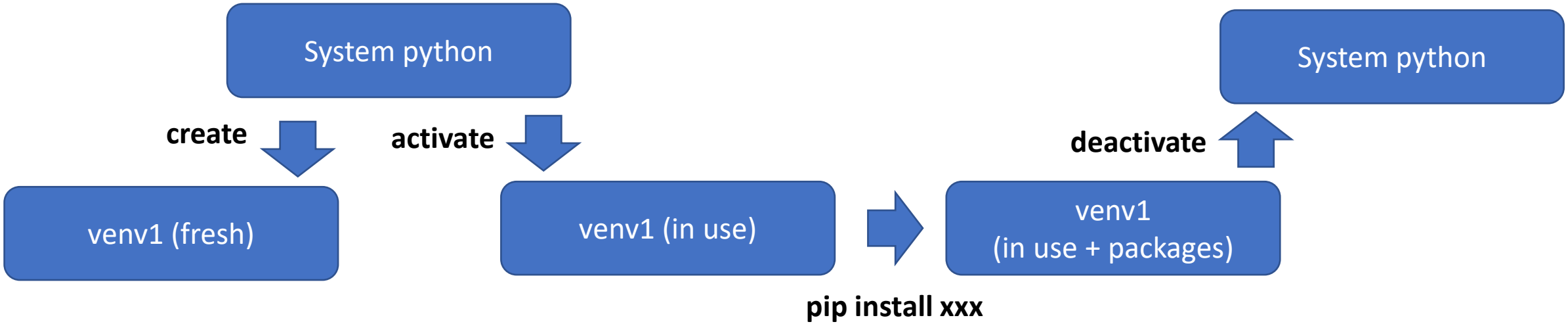
Virtual environment pattern



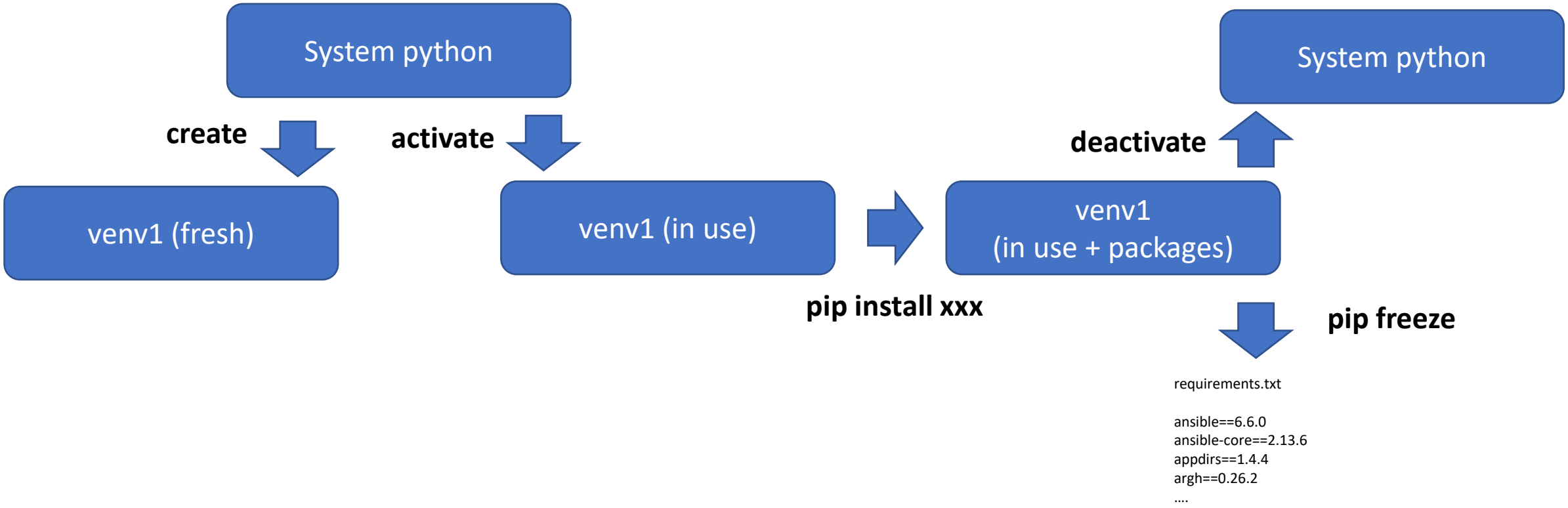
Virtual environment pattern



Virtual environment pattern

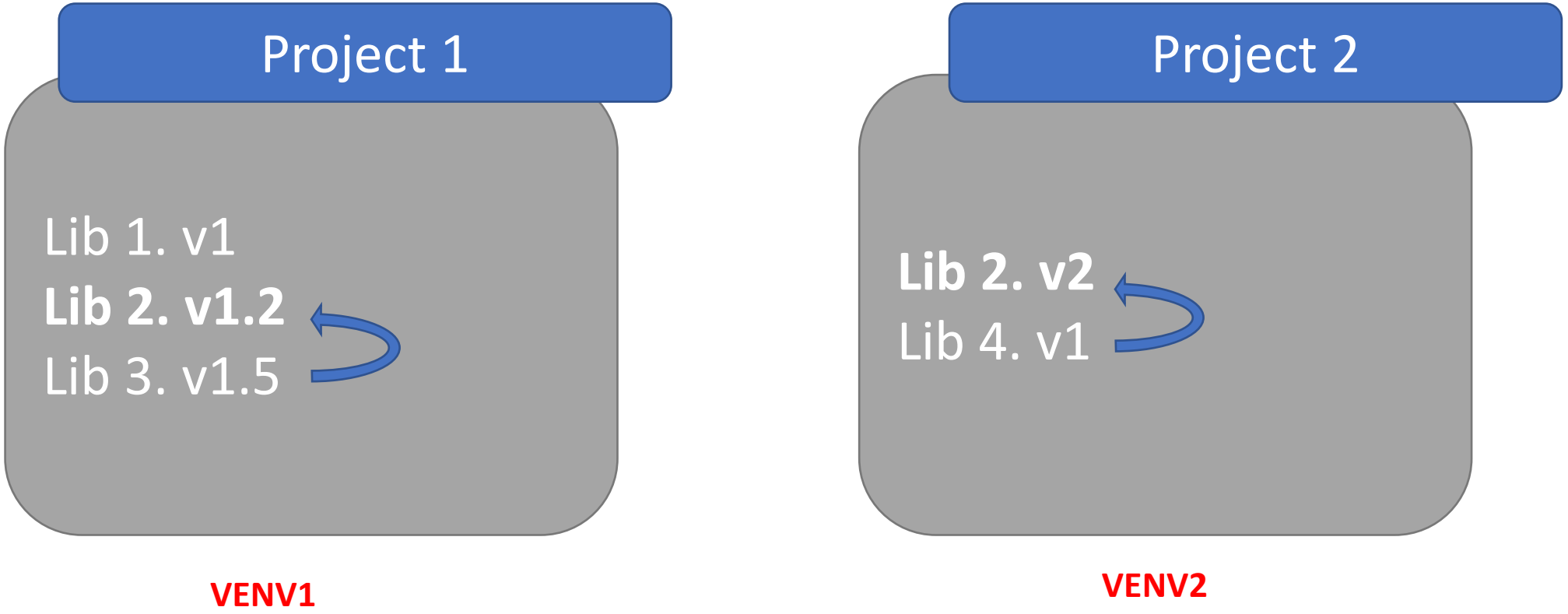


Virtual environment pattern



pip install -r requirements.txt

Python virtual environments



Basic implementation: **venv**

- **python3 -m venv env1**
  - **source env1/bin/activate**

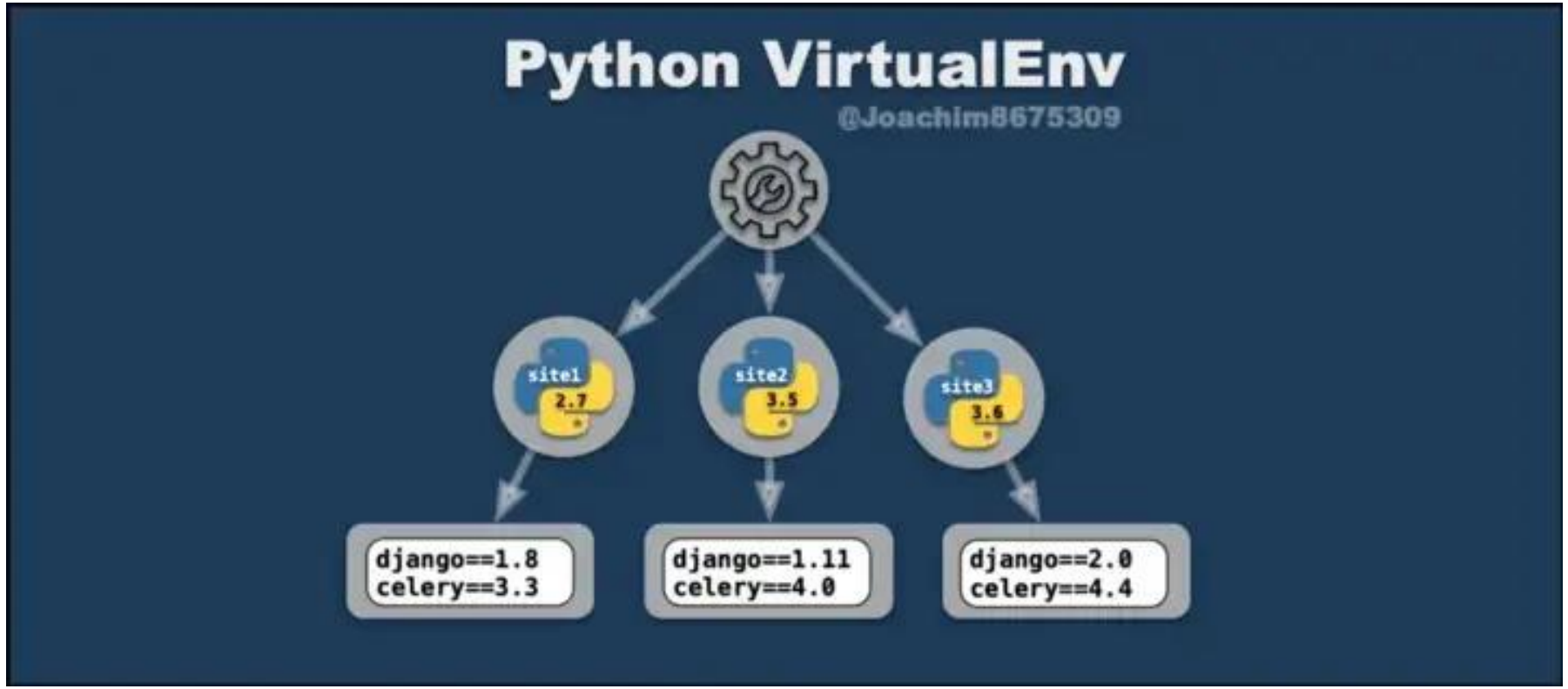
  - **pip** installs in dedicated path
  - Dump all the packages:
    - pip freeze > requirements.txt

```
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── easy_install
│   ├── easy_install-3.5
│   ├── pip
│   ├── pip3
│   ├── pip3.5
│   ├── python -> python3.5
│   ├── python3 -> python3.5
│   └── python3.5 -> /Library/Frameworks/Python.framework/Versions/3.5/bin/p
├── include
├── lib
│   └── python3.5
│       └── site-packages
└── pyvenv.cfg
```

- Major limitation: same python version of the system

Python-level reproducibility: powerful alternatives

- **Virtualenv** or **conda environments**



Reproducibility

- **Environment notion:**
  python dependencies

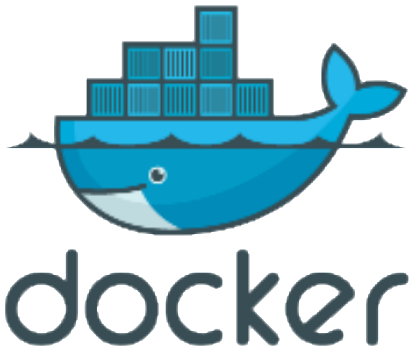
  - Venv, virtualenv, conda env

- Environment notion:
  OS / complete environment

- Light-weight sytem-level sandboxing

System-level reproducibility: containers

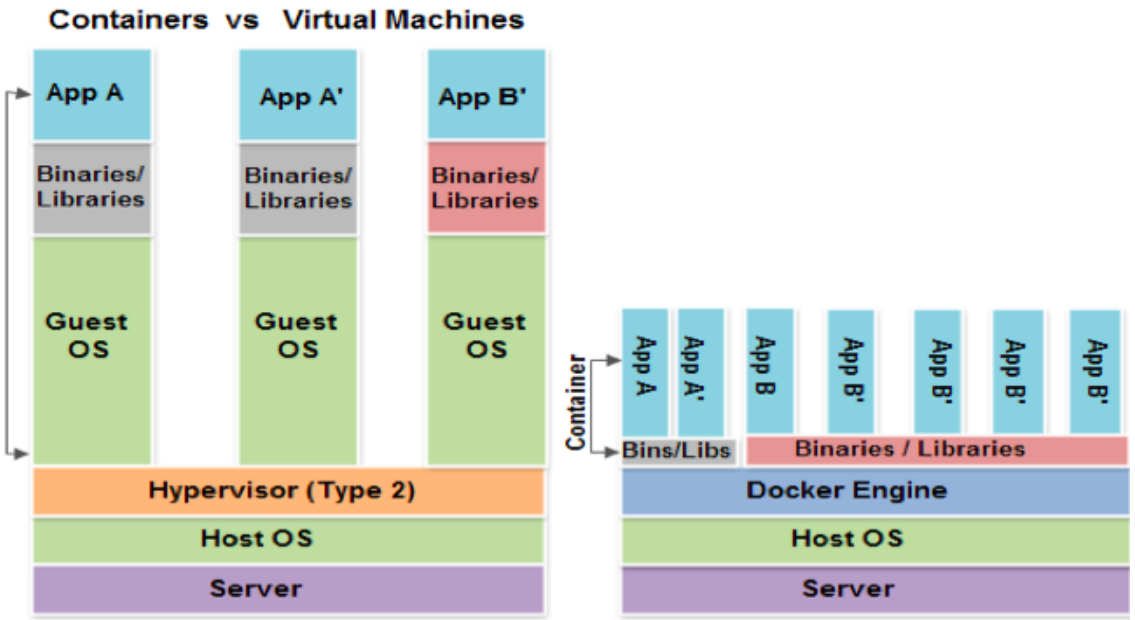
- Sandboxes
  - Run code in an “independent system” (nested)
  - Full config from “fresh” in one file (`Dockerfile`)
  - No scope on filesytem, ports, resources, unless specified binding
  - Full root rights



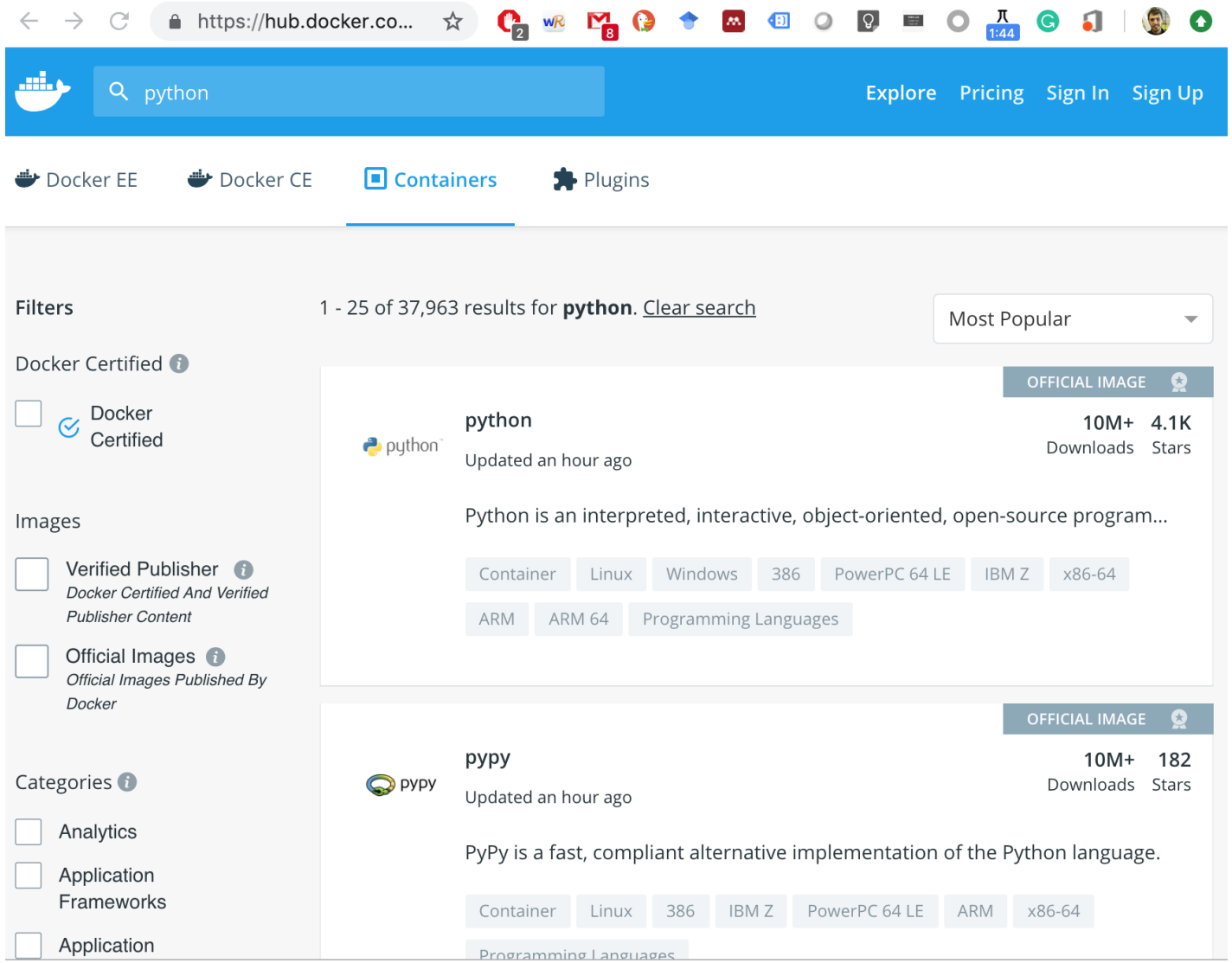
System-level reproducibility: containers

- Sandboxes
  - Run code in an “independent system” (nested)
  - Full config from “fresh” in one file (`Dockerfile`)
  - No scope on filesytem, ports, resources, unless specified binding
  - Full root rights

- Lighter than a VM
  - No nested kernel
  - Full definition in few bytes
  - Runs also on Win



Docker container: “fresh” system images



https://hub.docker.com

Docker container: pull & run

Pulling official image for python 3.7

docker run -i --rm -t python:3.7 /bin/bash

Automated download

```
acorbe@nb-20-145 2019-collaborative-computing-ictp  [master] $ docker run -i -t python:3.7 /bin/bash
Unable to find image 'python:3.7' locally
3.7: Pulling from library/python
e79bb959ec00: Extracting [==================================================> ]   44.5MB/45.34MB
d4b7902036fe: Download complete
1b2a72d4e030: Download complete
d54db43011fd: Download complete
69d473365bb3: Downloading [================================>                  ]  136.3MB/215MB
7dc3a6a0e509: Download complete
68cd774d0852: Download complete
2ef86095a118: Download complete
bd9da5a171e0: Download complete
```

```
root@559da25be6b2:/# python --version
Python 3.7.3
```

Docker container: pull & run

Pulling official image for python 3.7

docker run -i -t python:3.7 /bin/bash

```
root@559da25be6b2:/# python --version
Python 3.7.3
```

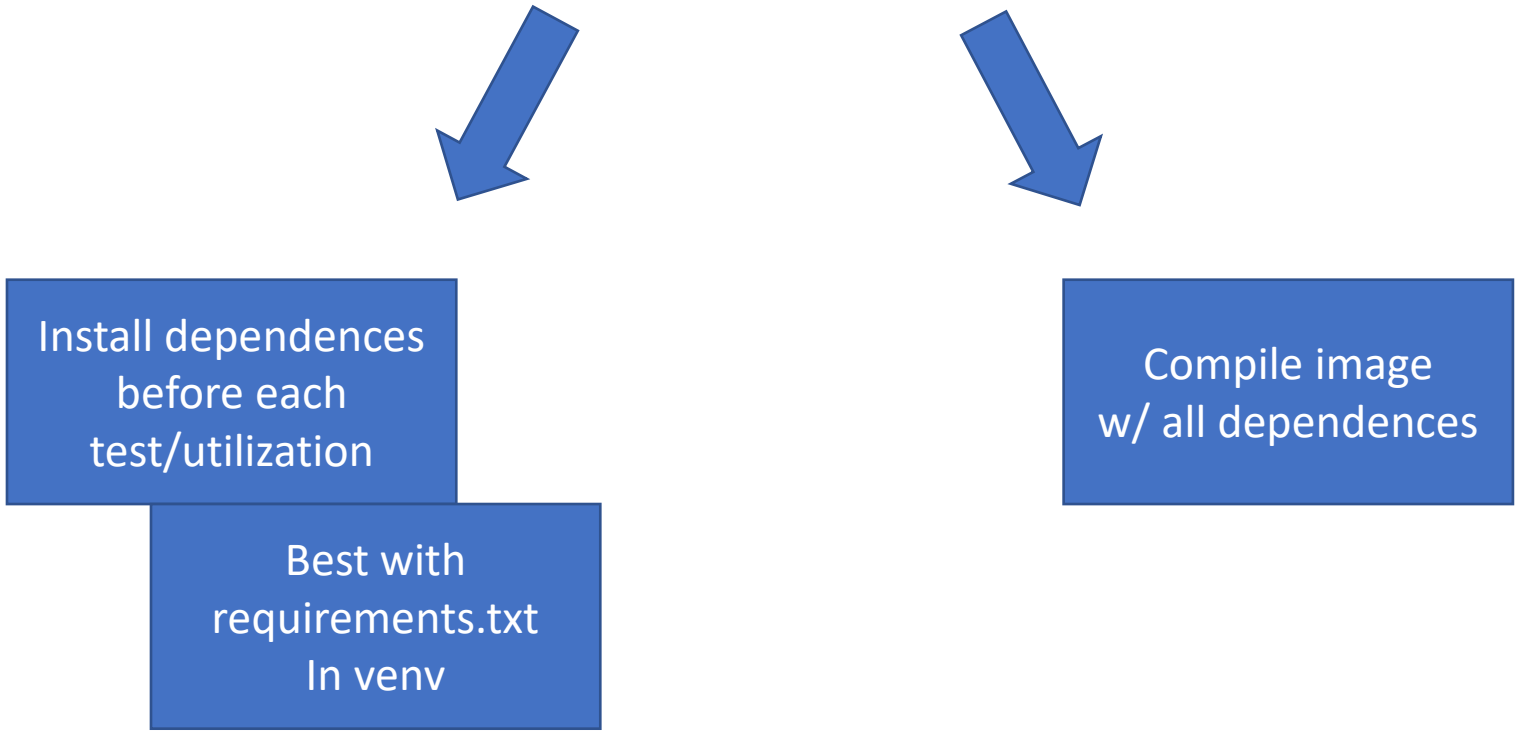
```
root@559da25be6b2:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

We have a fresh Debian to run on!

Containers: fully reproducible environments

Ideal for “3rd” parties testing

What about dependencies? Libraries, etc.?



Case 1

Reconfig at every usage w/ script

```
image: python:3.8
before_script:
  - python -V                    # Print out python version for debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install numpy nose
  - # OR
  - pip install -r requirements.txt
test:
  script:
  - cd  binary_str_2_float
  - nosetests -v
```

Case 2
Containers: fully reproducible environments

dependencies embedded (Dockerfile)

Similar to Makefile -> Dockerfile

```
docker build .

docker build -t tag:version .
```

Containers: fully reproducible environments

Where to start

What to add

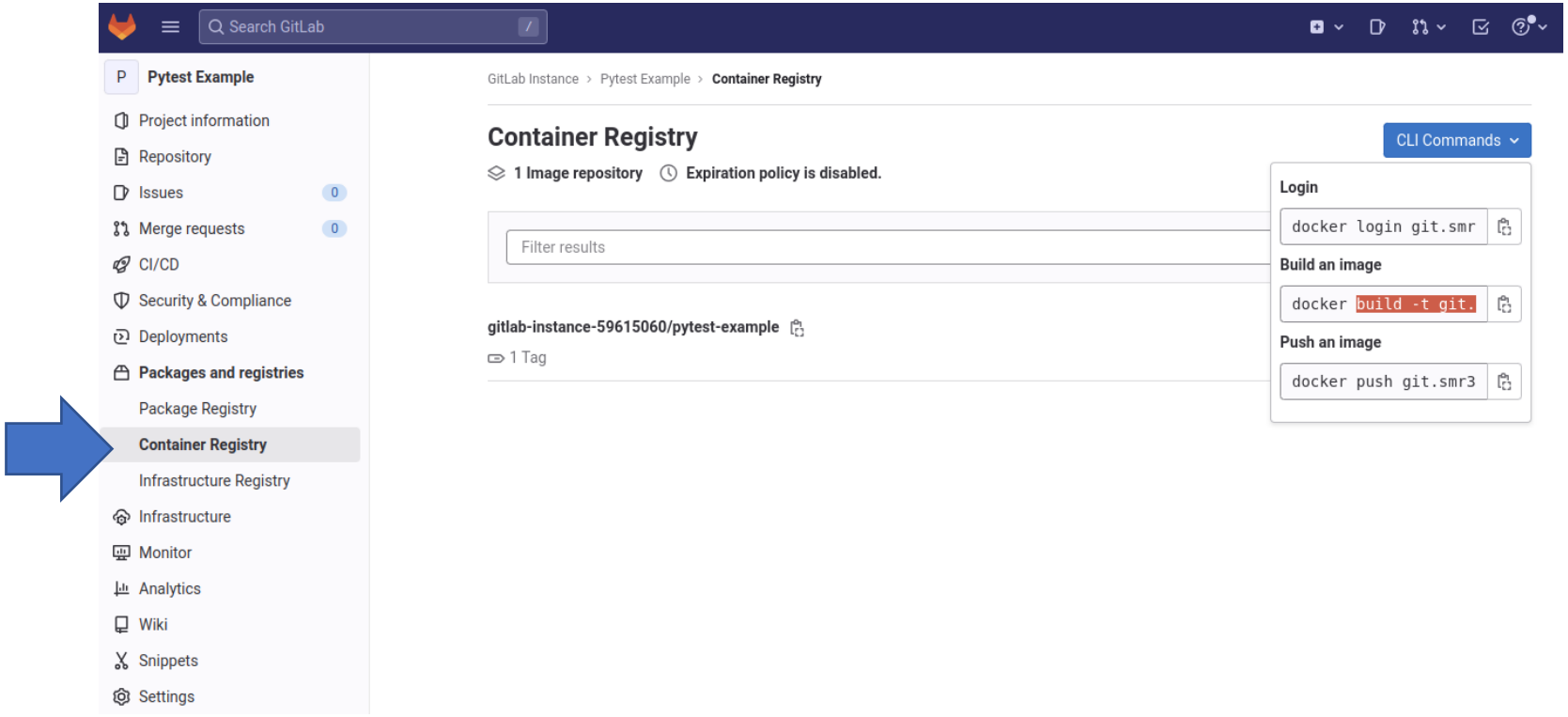
Dockerfile 711 Bytes

```
FROM ubuntu:18.04


# Install.
RUN \
  sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \
  apt-get update && \
  apt-get -y upgrade && \
  apt-get install -y build-essential && \
  apt-get install -y software-properties-common && \
  apt-get install -y byobu curl git htop man unzip vim wget

ENV DEBIAN_FRONTEND=noninteractive

RUN \
    apt-get install -y texlive-full

RUN \
  apt-get install -y gnuplot

RUN \
  rm -rf /var/lib/apt/lists/*

```

docker

Gitlab container registry

- Built images can be pushed/pulled from the gitlab container registry



Some CI examples

Demos index links

- Pytest, pycov & report deployment
  https://git.smr3696.ictp.it/gitlab-instance-59615060/pytest-example
  http://gitlab-instance-59615060.pages.smr3696.ictp.it/pytest-example/

- Compiling latex in CI
  https://gitlab.com/acorbe/ci-latex-example

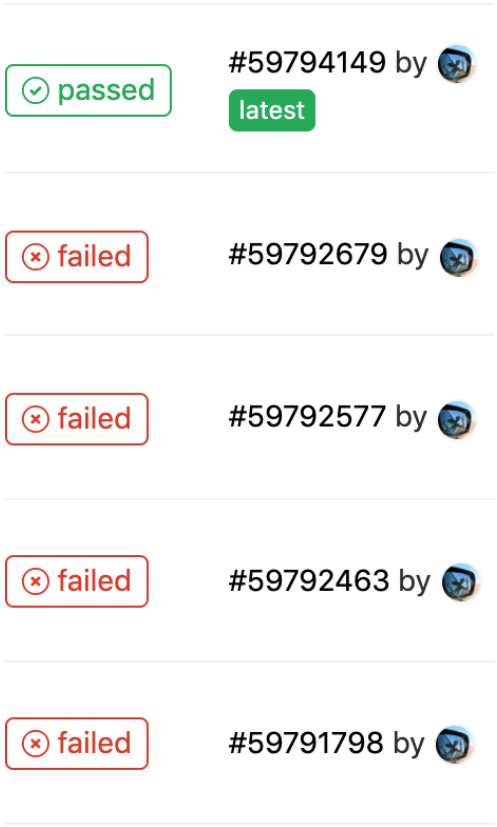
- Serving static pages
  https://gitlab.com/acorbe/acorbe-pages
  Result: https://acorbe.gitlab.io/acorbe-pages

Usecase 1: CI for testing - pytest

git push

.gitlab-ci.yml

docker



- At every push
  - Repo is cloned remotely
  - Tests are run
  - Should test fail one can receive emails

https://gitlab.com/acorbe-ci-tests/SMR3199_FP_ex

CI for testing - pytest

.gitlab-ci.yml

```
image: python:3.6
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache"
cache:
  paths:
    - .cache/pip
    - venv/
before_script:
  - python -V               # Print out python version for debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install numpy pytest
test:
  script:
  - cd  binary_str_2_float
  - pytest -v
```

CI for testing - pytest

passed	#59794149 by latest	master e391face fixed converter		00:01:08 2 minutes ago
failed	#59792679 by	master ab342d87 reduce		00:01:36 23 minutes ago
failed	#59792577 by	master 7284bc04 fixed print		00:01:08 26 minutes ago
failed	#59792463 by	master c42b89c3 reverted to normal pyt...		00:01:06 29 minutes ago
failed	#59791798 by	master 854dd228 update		00:00:35 41 minutes ago

CI for testing – pytest (here nosetest)

```
$ cd  binary_str_2_float
$ nosetests -v
tests that the trivial case 0.1e1 == 1. ... ok
tests that the trivial case 0.1111010101010e-4 == 0.0598907470703 up to 1e-10 accuracy ... ok
tests for the correct interpretation of negative sign ... ok
tests for the correct interpretation of positive sign ... ok
tests for the correct interpretation of negative exponent ... ok
tests for the correct interpretation of positive exponent ... ok
checks whether line command responds properly ... ok
Test that we deal only with normalized representations ... ok
Test that we deal only with normalized representations ... ok

----------------------------------------------------------------------
Ran 9 tests in 0.075s

OK
Creating cache default...
WARNING: .cache/pip: no matching files
venv/: found 2065 matching files
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-
cache/project/12175211/default
Created cache
Job succeeded
```

Duration: 1 m

Timeout: 1h (

Runner: share
5.gitlab.com (

Commit e391

fixed converte

Pipeline #

test

test

Usecase 2: Building binaries -- latex

- Version control on Latex:
  - Stop with paper_0.tex; paper_1; paper_2.tex; paper_final.tex; paper_edits.tex…
  - Simple collaboration with co-authors
  - Simply tracing source

- Issue: we want our paper to build
  - now and in 10 years; regardless the env
  - i.e. all dependencies are present
  - Let’s make it a CI-job

https://gitlab.com/acorbe/ci-latex-example

Building binaries -- latex

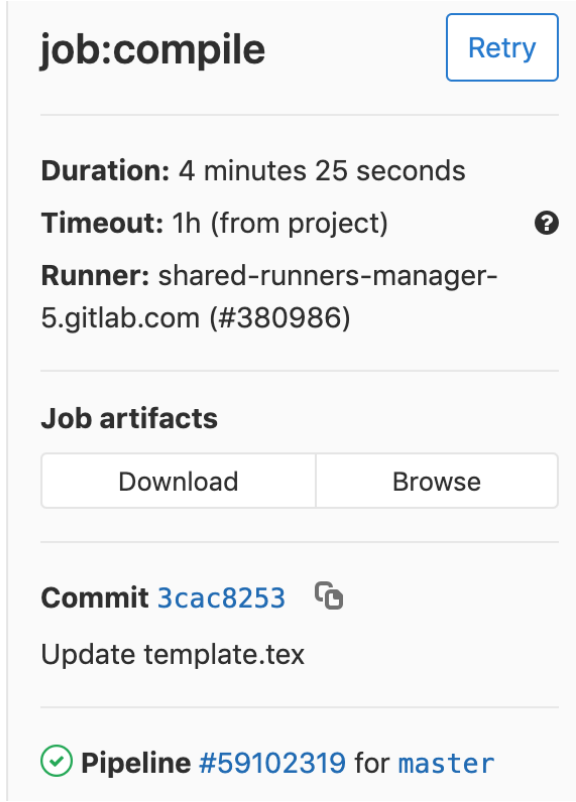
.gitlab-ci.yml 289 Bytes

```yaml


job:compile:
    image: niccokunzmann/ci-latex
    script:
       - latexmk -pdf
       - echo 'renaming target file for artifact upload...'
       - mv template.pdf paper-$CI_COMMIT_SHORT_SHA.pdf
    artifacts:
         paths:
         - paper-$CI_COMMIT_SHORT_SHA.pdf
```

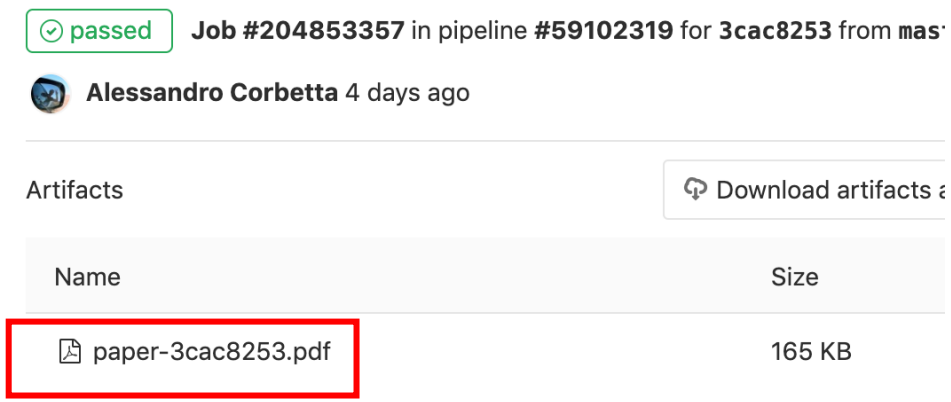
Retrieve product of compilation

- https://gitlab.com/acorbe/ci-latex-example

Building binaries -- latex



- At every push
  - Paper is built – code/dependency check
  - Automating spell check would be possible
  - Served as CI product + hash tracking



Building binaries -- latex

.gitlab-ci.yml 289 Bytes

```yaml
job:compile:
    image: niccokunzmann/ci-latex
    script:
       - latexmk -pdf
       - echo 'renaming target file for artifact upload...'
       - mv template.pdf paper-$CI_COMMIT_SHORT_SHA.pdf
    artifacts:
         paths:
         - paper-$CI_COMMIT_SHORT_SHA.pdf
```

Even better: call makefile

- https://gitlab.com/acorbe/ci-latex-example

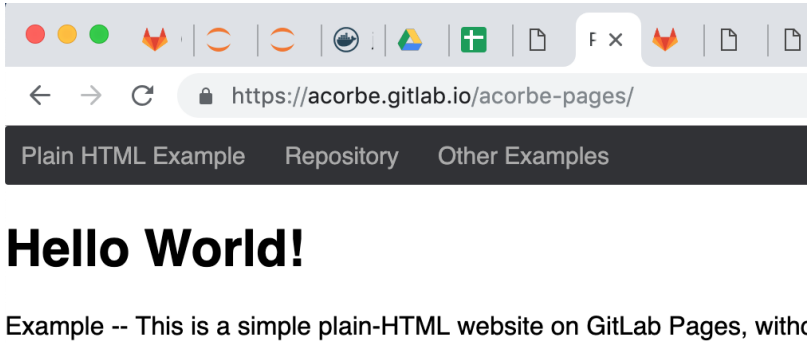
Usecase 3: CI static websites

https://gitlab.com/acorbe/acorbe-pages

Name
public
.gitlab-ci.yml
README.md

.gitlab-ci.yml 139 Bytes

```yaml
image: alpine:latest

pages:
  stage: deploy
  script:
  - echo 'Nothing to do...'
  artifacts:
    paths:
    - public
  only:
  - master
```



Usecase 4: CI for deploying content pages/documentation

Content source

Content built

Content served (static pages) *.gitlab.io/page

Content served by my server

- We can use gitlab-CI to build content:
  - Documentation w/ sphinx
  - Our personal webpage
  - Etc

Assumption: sphynx config in /docs

.gitlab-ci.yml 516 bytes

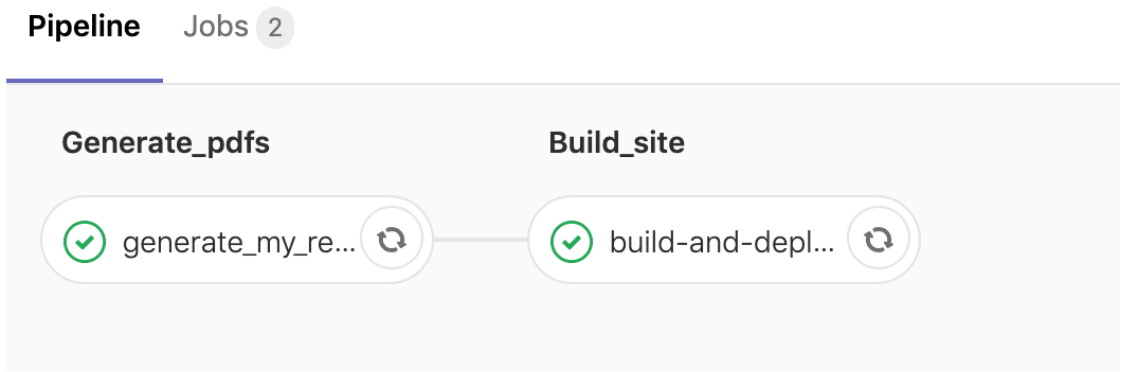
Edit in pipeline editor | Replace | Delete

```yaml
image: python:3.7


stages:
  - docs


pages:
    stage: docs

    script:
        - apt-get update
        - apt-get install -y graphviz
        - pip3 install numpy pandas scipy networkx matplotlib sphinx sphinxcontrib-napoleon pydot dask toolz pyreadr sklearn numba
        - pip3 install dask[dataframe] --upgrade
        - rm -f /usr/bin/python && ln -s /usr/bin/python3 /usr/bin/python
        - cd docs
        - make html
        - mv build/html ../public
    artifacts:
        paths:
        - public
```

Usecase 5: deployment on external server

```
stages:
  - generate_pdfs
  - build_site

generate_my_resume:
   stage: generate_pdfs
   script:
    - apt update
    - apt install -y python-pip - pip install jinja2
    - make my_resume
    - cp resume/resume-corbetta.pdf  content/publ
   artifacts:
      paths:
      - content/publ/resume-corbetta.pdf
```

```
build-and-deploy:
  stage: build_site
  environment:
    name: corbetta-website
    url: http://corbetta.phys.tue.nl
  script:
    - source activate pelican
    - make ssh_upload
```



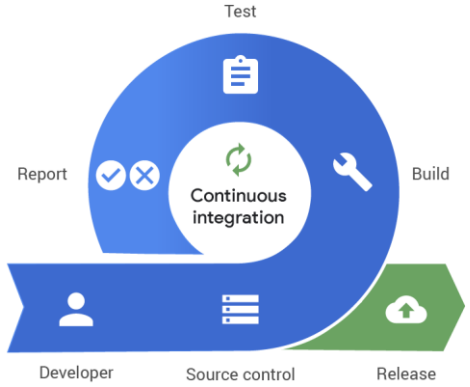
Thanks

Structured collaboration

Gitlab
Conversational development

Trusting changes

Testing
Continuous integration
Continuous deployment



Immediate reproducibility

Makefile
Containers
Docker
Virtualenvs