JENNIFER2
EU grant n.822070

Containers for ND280 software

(and others use cases)

Mathieu Guigue

Jennifer-II Computing Workshop — December 12th 2019

SCIENCES SORBONNE UNIVERSITÉ

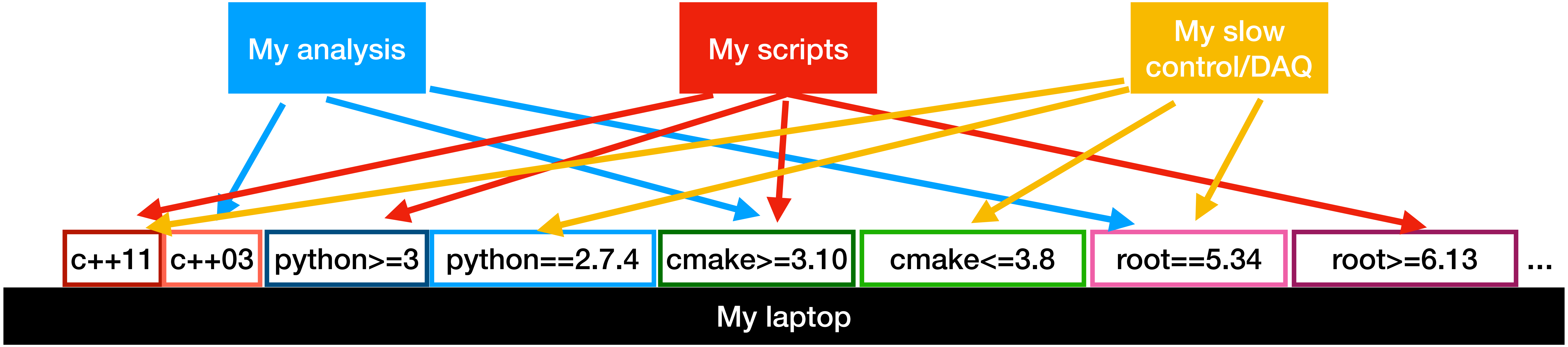
LPNHE PARIS

cnrs

IN2P3 Les deux infinis

Nowadays software issues

Conflicting dependencies between projects
Development on networked hosts
Development by multiple agents/groups
Manage and deploy software on heterogeneous systems
Deploy many applications (slow control, DAQ, web interface…)



Moving to containerized code

“Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes.”
https://aws.amazon.com/containers/

What is it good for?

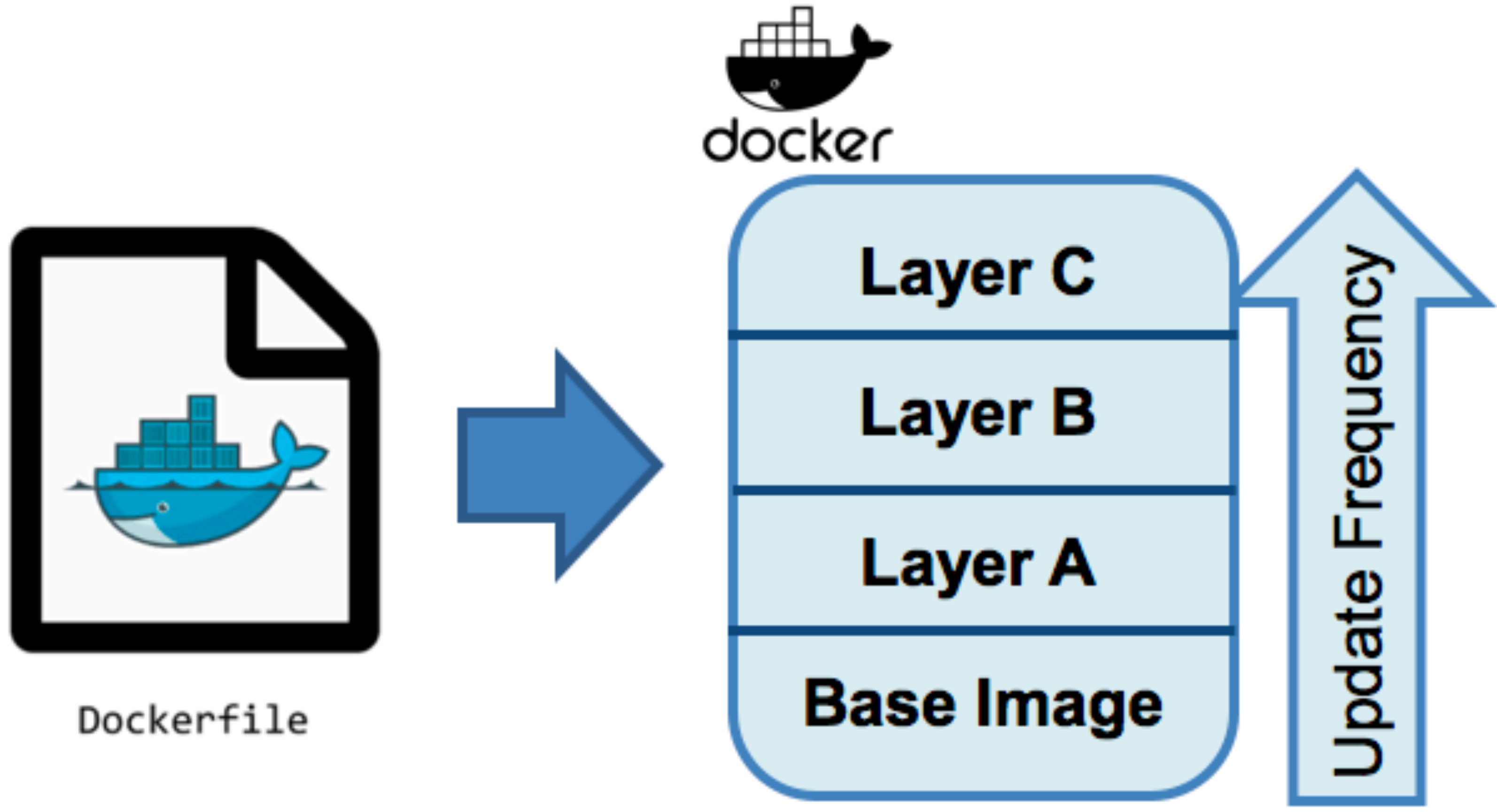
- **uniform** environment (developers on Mac/Ubuntu, cluster on Centos/SLX)*
- **reproducible** installation and code testing
- processes isolation
- **controlled** networking capabilities
- manage dependencies while isolating package code
- facilitate software packaging and **sharing**
- provide control over **resource usage** and dynamic resource allocation

→ **Singularity — Docker**

EU grant n.822070

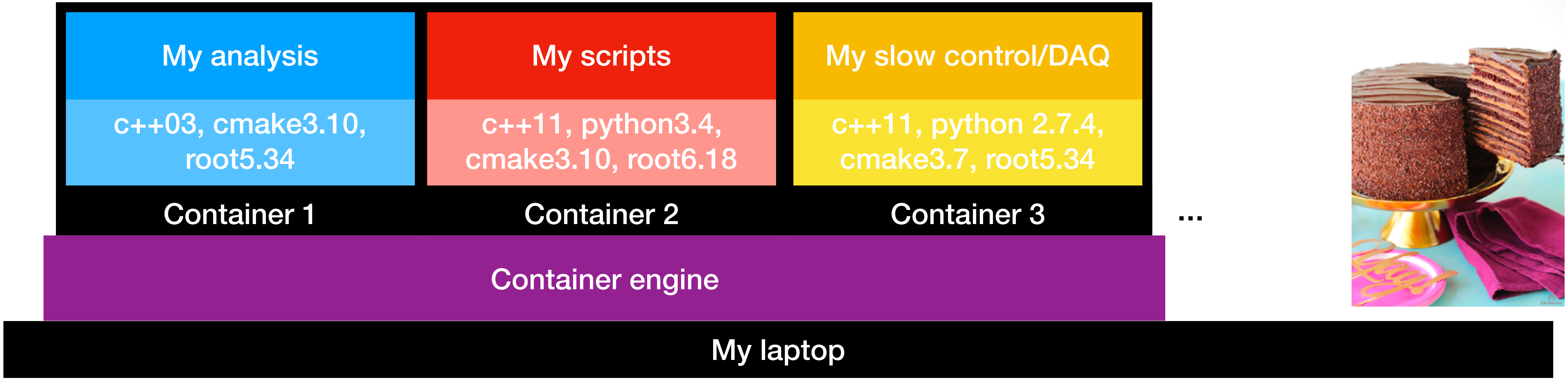
“Layer cake” approach

- Containers built in layers
  - Starting from a base/OS image
- Each layer contains a piece of software
  - Depth related to update frequency
- “Cake recipe” given by Dockerfile



Layer by update frequency

https://openliberty.io/


My analysis
c++03, cmake3.10, root5.34
Container 1

My scripts
c++11, python3.4, cmake3.10, root6.18
Container 2

My slow control/DAQ
c++11, python 2.7.4, cmake3.7, root5.34
Container 3

...

Container engine

My laptop

Nowadays usage of containers

Continuous Integration/Continuous Deployment:

Pushed code changes tested (building of containers and tests)

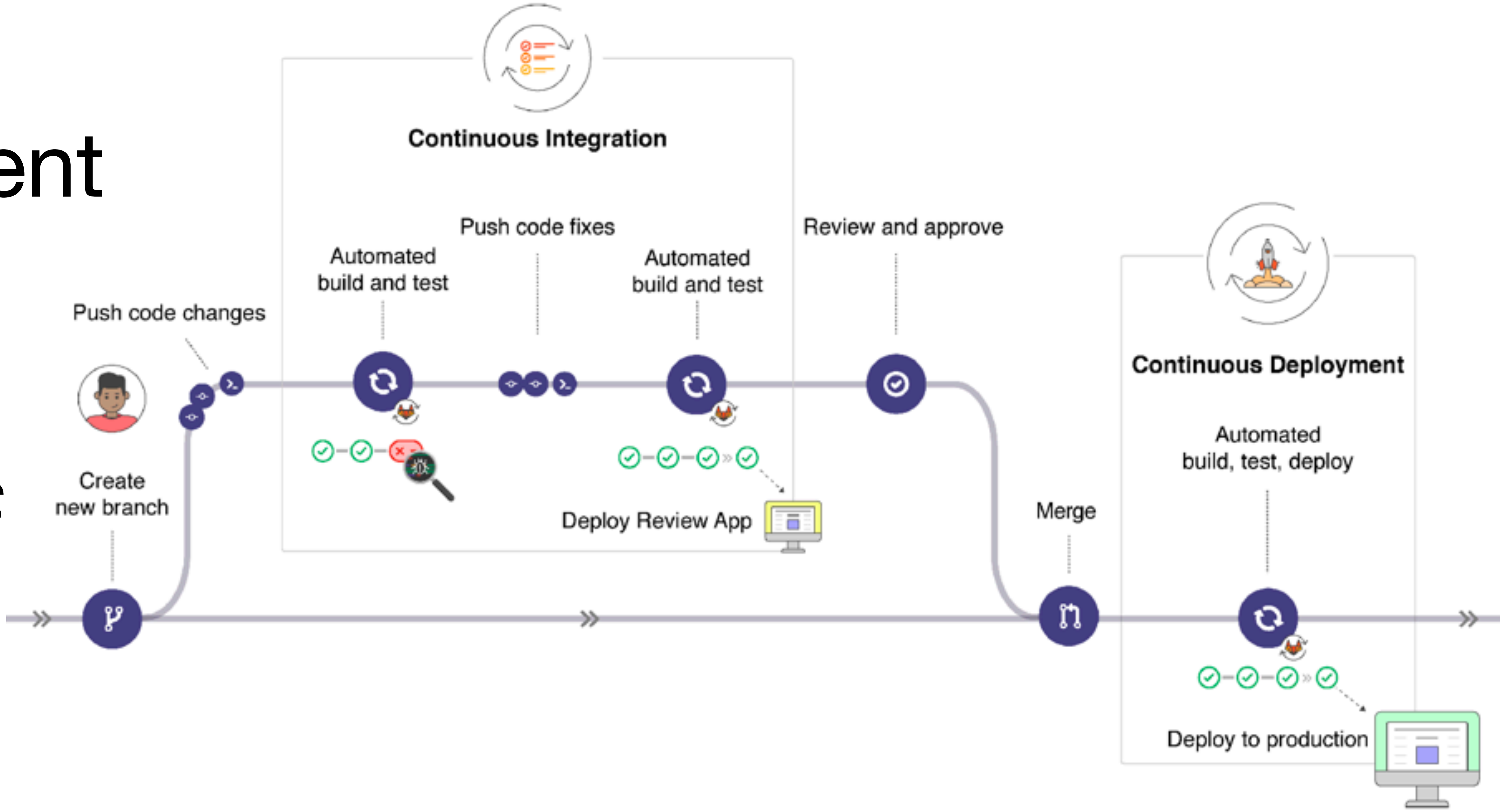
→build/tests successful before merge

Once merged, automated image build containing latest “stable” code

Trigger other actions (build/tests of dependent packages)

→Docker provides uniform environment

Intensive usage in software industry and fundamental physics experiments



Nowadays usage of containers

Continuous Integration/Continuous Deployment

Job environment and job submission:

Heterogeneous clusters (various OS, installed libraries…)

Potentially strong constraints on environment set by software

→Use of containers to run jobs on cluster

Dirac client installation and configuration can be messy

→Provide maintained job submission container (e.g. Dirac client)

EU grant n.822070

Nowadays usage of containers

Continuous Integration/Continuous Deployment

Job environment and job submission

DAQ/slow control software testing and deployment:

Multiple identical deployed services

→Run connected containerized code on host via docker-compose

→Scalability and control using Kubernetes

Impact of cascading failures & recovery, network delay…

Dynamic service discovery

Continuous Integration elsewhere

Basically every experiment e.g. ATLAS

One image per package
Used for development, testing, CI
One image for entire software stack (simulation, analysis)

Nightly builds of every package and stack
Integrated in most of Git frameworks e.g. Gitlab, Github

Jobs elsewhere

PNNL (Richland - USA)*

Job environment

Conversion of software stack image to Singularity image**

Upload image on CVMFS

Job spin container up and run commands***

Job submission

Image with Dirac client**** and specific configuration

Job submission and data retrieval from File Catalog

Sharing with host via docker-compose → **plug-and-play!**

* Poster

** Conversion from docker to singularity maintained by Singularity people: https://github.com/singularityhub/docker2singularity

*** Singularity container support: https://github.com/DIRACGrid/DIRAC/pull/3476

**** An example: https://github.com/mariojmdavid/docker-dirac

EU grant n.822070

DAQ/slow control elsewhere

Project 8 - ADMX (Seattle), Memphyno (France):

Software testing

Experimental room not always accessible to developers
SC/DAQ software in separate images
Experiment-like environment for development/debugging

Software deployment

Containers management by “orchestrator” (Kubernetes)
Number of replicas configurable
Container failure recovery
Management via API and browser
Successfully deployed on these experiments!

EU grant n.822070

Applications to T2K software

Moving to Gitlab for version control and transition to CMake

See Alex’s presentation

Applications of containerization for T2K software:

- Enable local development environment*
- Continuous Integration via Gitlab*
- Jobs execution across heterogeneous clusters**
- DAQ and slow control in containers***

*today

**”tomorrow”

***maybe… one day?

Containerizing T2K analysis and simulation software

(1) developer testing

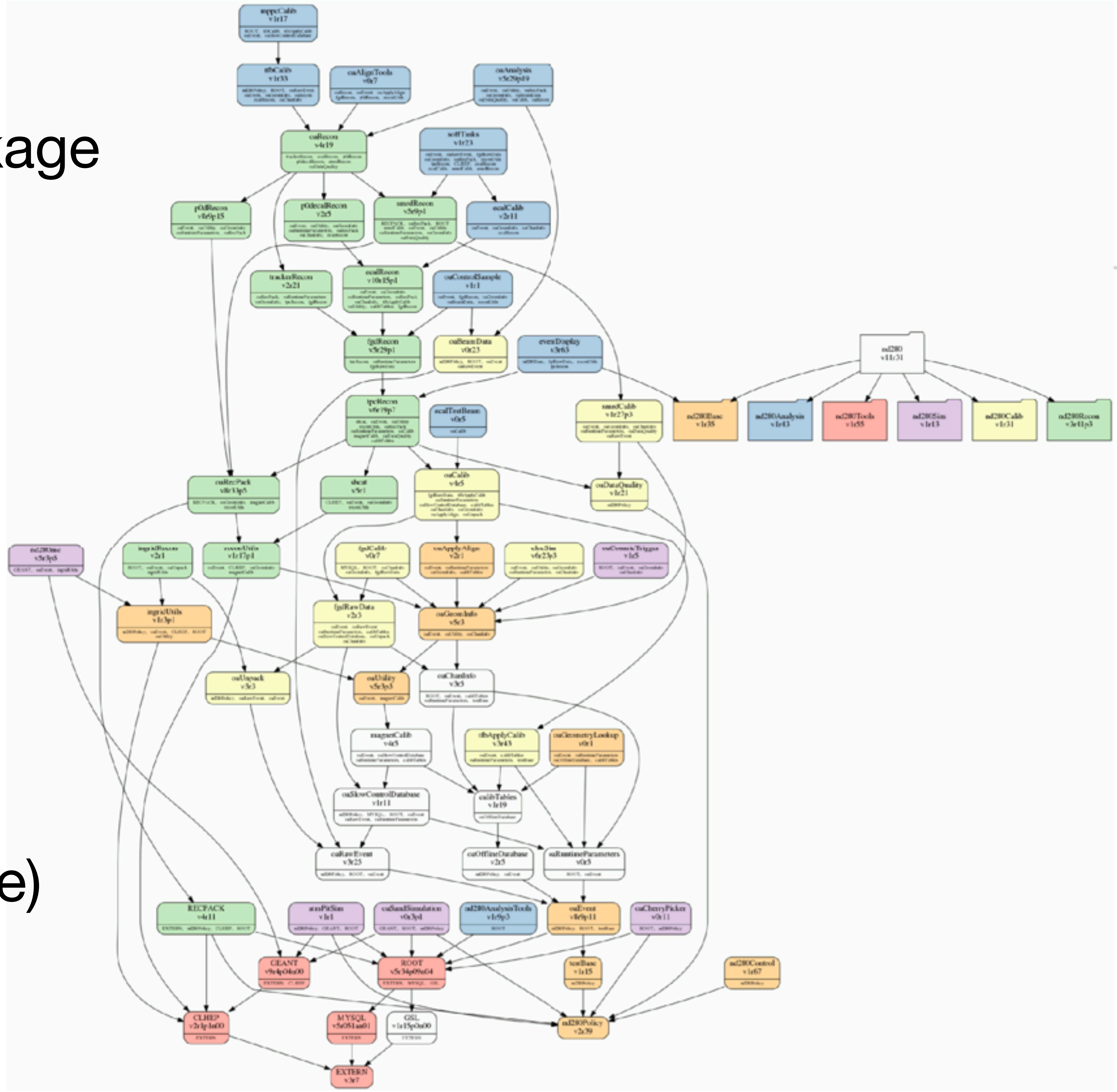
→ build/test only (if possible) the relevant package

(2) production testing

→ build/test several packages at once (using master packages)

(3) production deployment

→ build/test all packages at once (using nd280SoftwareMaster)



For each level:

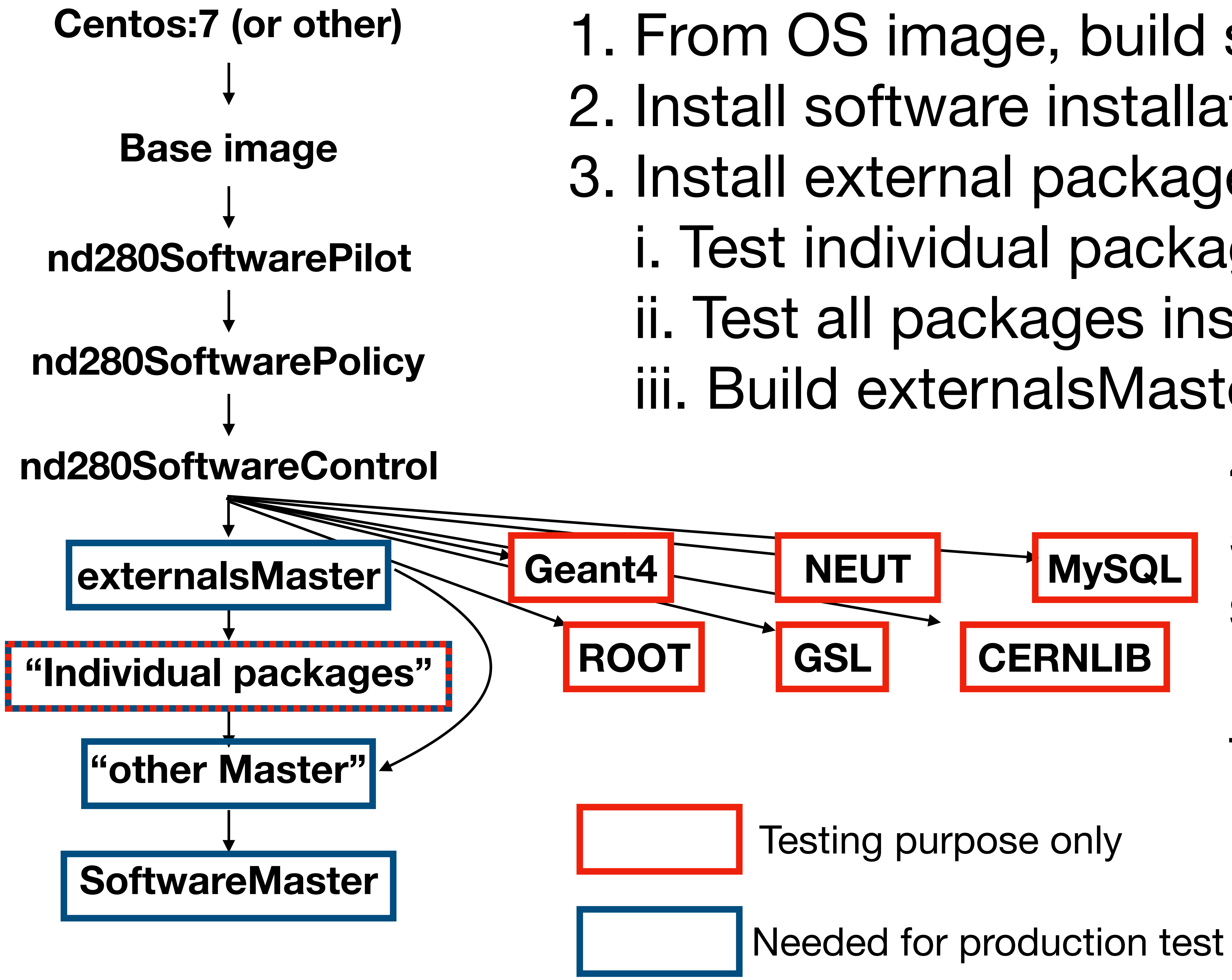
- one Dockerfile
- one image (reusing layers produced elsewhere)

(- one CI configuration file)

EU grant n.822070

Executing a layered approach with ND280 software

1. From OS image, build system dependencies (one image per OS)
2. Install software installation scripts and CMake logic
3. Install external packages (ROOT, GSL, Geant4…)
   i. Test individual package installation using CI
   ii. Test all packages installation using “externalsMaster”
   iii. Build externalsMaster image as base image for other packages
4. Same for individual packages
5. Build software stack using softwareMaster
   i. Produce image for high-level software tests (small simulations), users and jobs

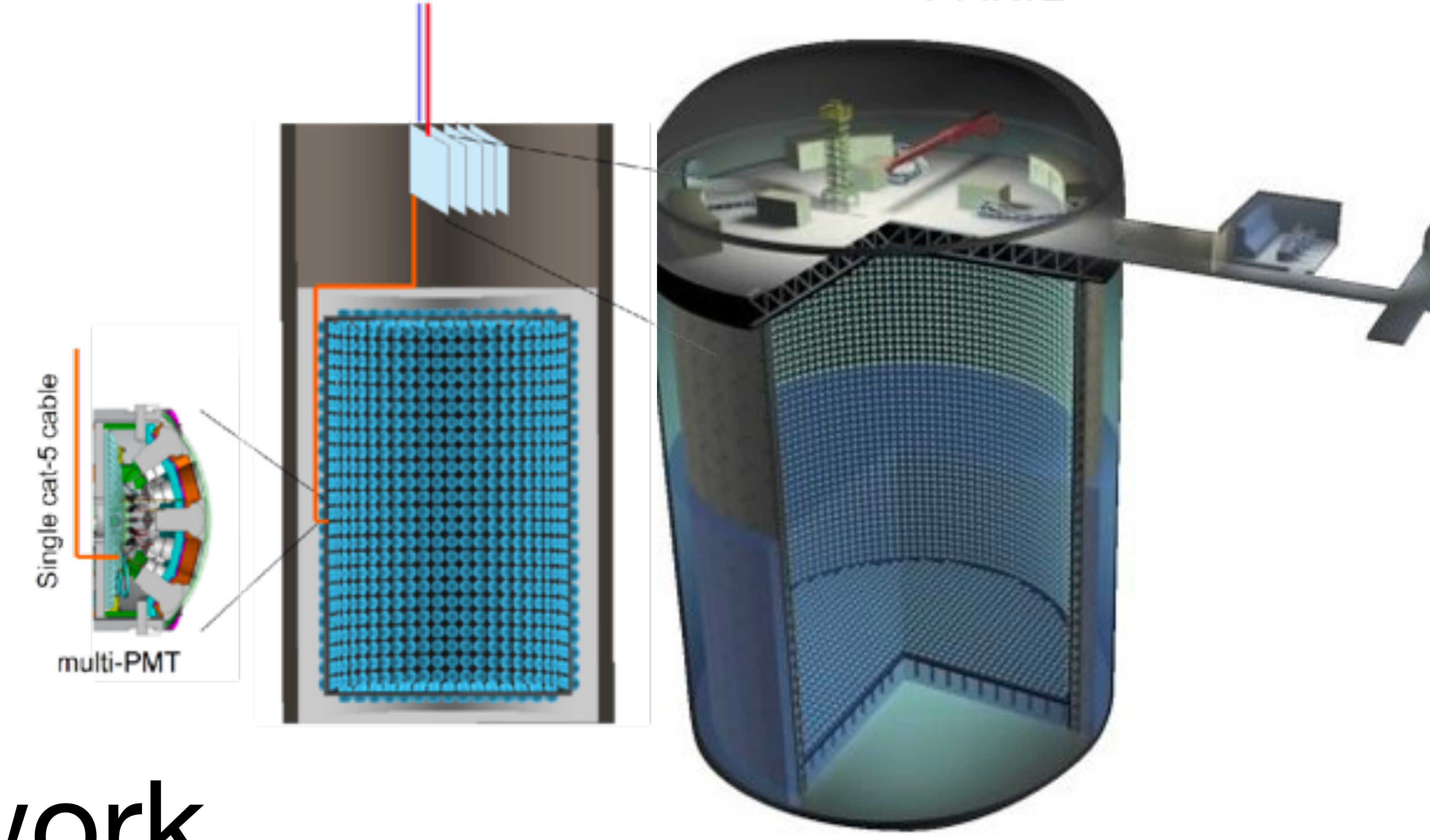


🟥 Testing purpose only

🟦 Needed for production test

Work in progress: currently debugging some off-road installation procedures

Applications to HK



New experiment, new possibilities!

Far detector DAQ model very different from ND280

- ToolDAQFramework as DAQ system
- Decentralized system communicating through network
- Multiple processes to be controlled (online triggers, event builders, SC…)
- Need for crash recovery
- A lot of components to develop and test

Containers could have very positive impact on testing and deployment

Kubernetes as "orchestrator"…?

→**Developing expertise within Jennifer-II would be highly beneficial!**

EU grant n.822070

Conclusions

- Containers are widely used in software industry
- Useful features and properties for small and large-scale experiments
  - Isolated development environment
  - Uniform environment regardless of hosts heterogeneity
  - Resources and network control
  - Useable as Continuous Integration executor
- Work-in-progress in the T2K collaboration
  - Containerization of ND280 analysis and simulation software
  - Continuous Integration in Gitlab
- Potential other usages
  - As part of job submission process (base image, Dirac client…)
  - Slow control and DAQ testing and deployment

Jennifer-II is a great place
to exchange ideas and develop common frameworks!

EU grant n.822070

Backup

Docker terminology

Docker: open-source project to create, deploy and run applications via containers

Docker Inc.:

- provides applications to run on Mac/Linux/Windows
- provides free hosting and automatic builds of images

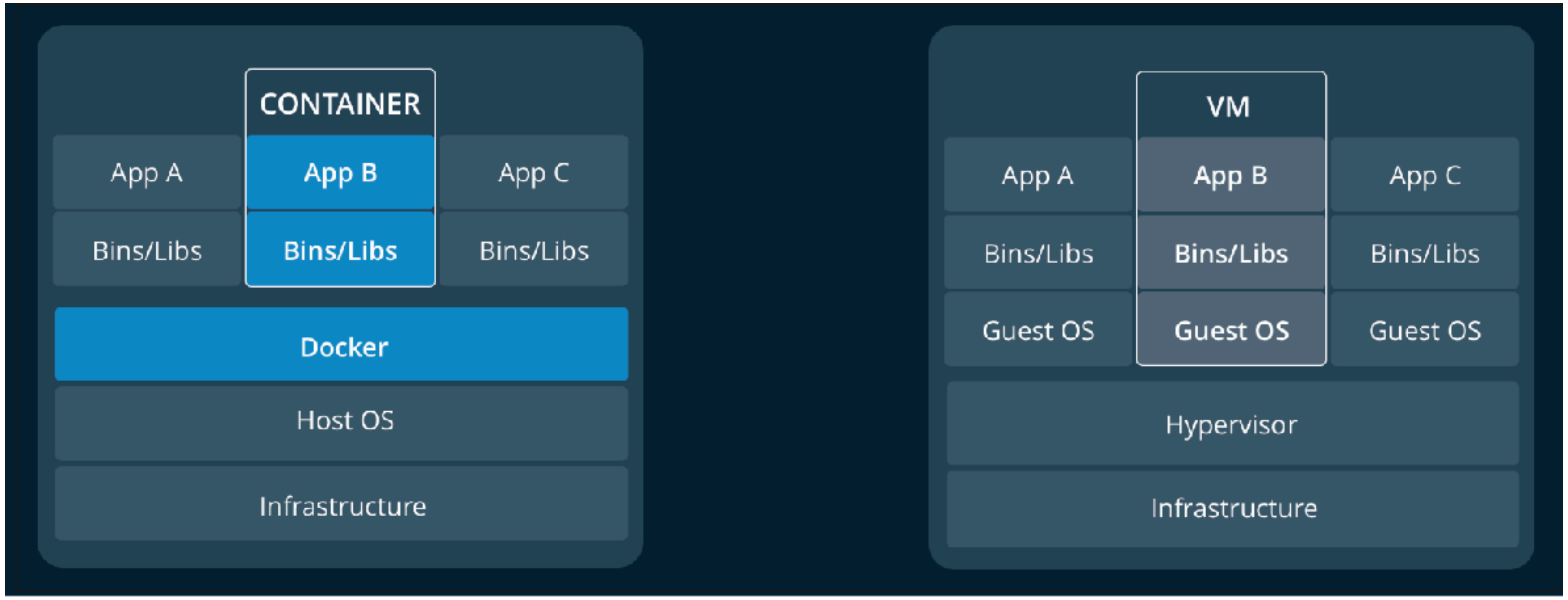
Container: self-container application easily deployable in an environment

Container image: compressed container used to create functioning containers

Docker engine: back-end of Docker software running on computing element (laptop, server…) and managing containers

Docker client: interface that communicates with the Docker engine

Difference between containers and VM



Container runs on Linux as a **process** and share host machine kernel

→direct access to host resources

VM runs as “independent” **guest operating system**

→virtual access to the host resources

→VM need more resources than containers (CPU, memory, disk space)

CI testing

Multiple runner-servers (allowing individual downtime)

Instance set up at CC-IN2P3 using their runner servers

Issues between CC-IN2P3 and git.t2k.org (solved 2 weeks ago)

One runner in Poland

Could setup dedicated Linux desktop machines at LPNHE (easy!)

What's next?

Deploy first version on some key packages

Define pipelines

Should a new version of X trigger something? How to release?

Evaluate actual computing needs (depends on pipeline/users)

Wait for more finalized version before full-deployment (minimal version…)

EU grant n.822070

From the beginning… a base image

Starts from a base Docker image

- contains all needed dependencies (dev tools, cmake3, wget…)
- defines common location for all packages: /usr/local/t2k/current
- defines some convenience env. variables

Currently, centos:7 is used, but others/multiple OS are possible

Note:

If we want several Docker image (depending on OS), could use manifest

Only framework and master package would need image for each OS

Dockerfile, images… → here

Framework images

Several “framework” packages:

- pilot/nd280SoftwarePilot
- framework/nd280SoftwarePolicy
- framework/nd280SoftwareControl

Need to be installed before installing packages (control not actually needed, but good to have earliest on)

Need to change install order?

For each, need the CI to build/test/generate Docker image

Master branch → “latest” exists

Tag → “X.Y.Z” exists

Release branch → tagged docker image “X.Y.*_latest”? (useful for debugging)

Merge request → dedicated tagged image? (useful for debugging)

Any branch → dedicated tagged image “feature_XXX”?

EU grant n.822070

externalsMaster

Based on nd280SoftwareControl image
Contains GSL, MySQL, ROOT, GEANT4, CERNLIB, CLHEP, NEUT
Used as base image (FROM statement) of all other packages

Using find-dependencies pilot script for getting packages/version
- Careful with what username/password is used…

Libraries/data rather large
- Dominated by Geant4, CERNLIB and ROOT
- Post-installation cleanup help reducing size
- Need cleaning NEUT installation (misplaced header files…)

Building packages

externalsMaster as base image

Currently copy files from already existing Docker images

ex:
```
ENV OAGEOMINFO_VERSION 5.9
ENV OAGEOMINFO_PATH $COMMON_BUILD_PREFIX/oaGeomInfo_${OAGEOMINFO_VERSION}
COPY --from=git.t2k.org:8088/nd280/base/oageominfo:5.9 ${OAGEOMINFO_PATH} ${OAGEOMINFO_PATH}
```

Avoid rebuilding code

But manually updating version in Dockerfiles needed

Will use of minimal version feature by Alex

ex:
```
ND280_USE(oaEvent 8.16+ )
```

Versions needed for building code in XXXND280_USE.cmake file

Easier maintenance of packages dependencies

See backup slides for more details about progress

Gitlab-ci for nd280SoftwarePilot

```
.job_template: &job_definition  # Hidden key that defines an anchor named 'job_definition'
  image: docker:latest
  stage: build
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY

docker-build-master:
  <<: *job_definition
  script:
    - docker build --pull --build-arg GIT_T2K_TOKEN=$CI_REGISTRY_PASSWORD --build-arg GIT_T2K_USERNAME=$CI_REGISTRY_USER -t
"$CI_REGISTRY_IMAGE:latest" .
    - docker push "$CI_REGISTRY_IMAGE:latest"
  only:
    - master

docker-build-tags:
  <<: *job_definition
  script:
    - docker build --pull --build-arg GIT_T2K_TOKEN=$CI_REGISTRY_PASSWORD --build-arg GIT_T2K_USERNAME=$CI_REGISTRY_USER -t
"$CI_REGISTRY_IMAGE:$CI_COMMIT_TAG" .
    - docker push "$CI_REGISTRY_IMAGE:$CI_COMMIT_TAG"
  only:
    - tags

docker-build-mr:
  <<: *job_definition
  script:
    - docker build --pull --build-arg GIT_T2K_TOKEN=$CI_REGISTRY_PASSWORD --build-arg GIT_T2K_USERNAME=$CI_REGISTRY_USER -t
"$CI_REGISTRY_IMAGE" .
  only:
    - merge_requests
```

Typical content of a Dockerfile

```
FROM git.t2k.org:8088/nd280/master-packages/externalsmaster:5.3.5.0 as pre_oaEvent

ENV OAEVENT_VERSION 8.15
ENV TESTBASE_VERSION 1.17
ENV OAEVENT_PATH $COMMON_BUILD_PREFIX/oaEvent_${OAEVENT_VERSION}
ENV TESTBASE_PATH $COMMON_BUILD_PREFIX/testBase_${TESTBASE_VERSION}

COPY --from=git.t2k.org:8088/nd280_old/base/testbase:1.17 ${TESTBASE_PATH} ${TESTBASE_PATH}

########################################
FROM pre_oaEvent as interm_oaEvent

COPY . ${OAEVENT_PATH}

RUN mkdir ${OAEVENT_PATH}/${LINUX_INSTALL_FOLDER}
WORKDIR ${OAEVENT_PATH}/${LINUX_INSTALL_FOLDER}
ENV ND280_NJOBS 3
RUN source $COMMON_BUILD_PREFIX/setup.sh &&\
    source $ROOT_PATH/$LINUX_INSTALL_FOLDER/bin/thisroot.sh &&\
    cmake ../cmake &&\
    make -j3

########################################

FROM pre_oaEvent

COPY --from=interm_oaEvent $OAEVENT_PATH/$LINUX_INSTALL_FOLDER/*.sh $OAEVENT_PATH/
$LINUX_INSTALL_FOLDER/
...
```

Base image (all external dependencies)

Package and dependencies versions

Path definitions

Dependencies copy

Package installation

Clean container creation
(no intermediate file)

Make Docker images lean

Important for CI

Base image (Centos7) is 1.59GB (unnecessary dependencies?)
Installed externals dependencies:

```
275M	/usr/local/t2k/current/NEUT_5.3.5.00
160M	/usr/local/t2k/current/MYSQL_5.6.20.01
36M	/usr/local/t2k/current/GSL_1.15.0.00
773M	/usr/local/t2k/current/ROOT_5.34.34.00
90M	/usr/local/t2k/current/CLHEP_2.1.1.0
469M	/usr/local/t2k/current/CERNLIB_2005.8
560K	/usr/local/t2k/current/externalsMaster_1.74
880M	/usr/local/t2k/current/Geant4_10.1.03.00
3.3M	/usr/local/t2k/current/nd280SoftwarePolicy_v3.1.2
2.8M	/usr/local/t2k/current/nd280SoftwarePilot
2.7G	/usr/local/t2k/current
```

→Last layer is 2.6GB

Not save intermediate files (CMakeCache, objects…) from installation
- Use additional intermediate folder for these files
- “Only” libraries/exe/headers installed in output folder (Linux-…)
- Doable for C++ dependencies, more complex for CERNLIB/NEUT
- Need changes in policy for our packages

Careful with using cpp files as headers (need copy)
- Only headers should be copied over

Packages status

Package name	Docker status	Docker-compose	Gitlab-CI
base	Final	None	None
nd280SoftwarePilot	Final	None	Final
nd280SoftwarePolicy	Final	None	Final
nd280SoftwareControl	Final	None	Final
externalsMaster	Final	None	Final
testBase	Final	None	Final
oaEvent	Working	None	Final
oaGeomInfo	Working	None	None
oaChanInfo	Working	None	None
oaUtility	Working	None	None
oaRuntimeParams	Working	None	None
oaMagnetCalib	Working	None	None
oaOfflineDatabase	Working	None	None
oaRawEvent	Working	None	None
oaCalibTables	Working	None	None
oaSlowcontrolDatabase	Working	None	None
detResponseSim	Working	Final	None
neutGeant4CascadeInterface	Final	None	Final
nd280Geant4Sim	None	None	None

EU grant n.822070