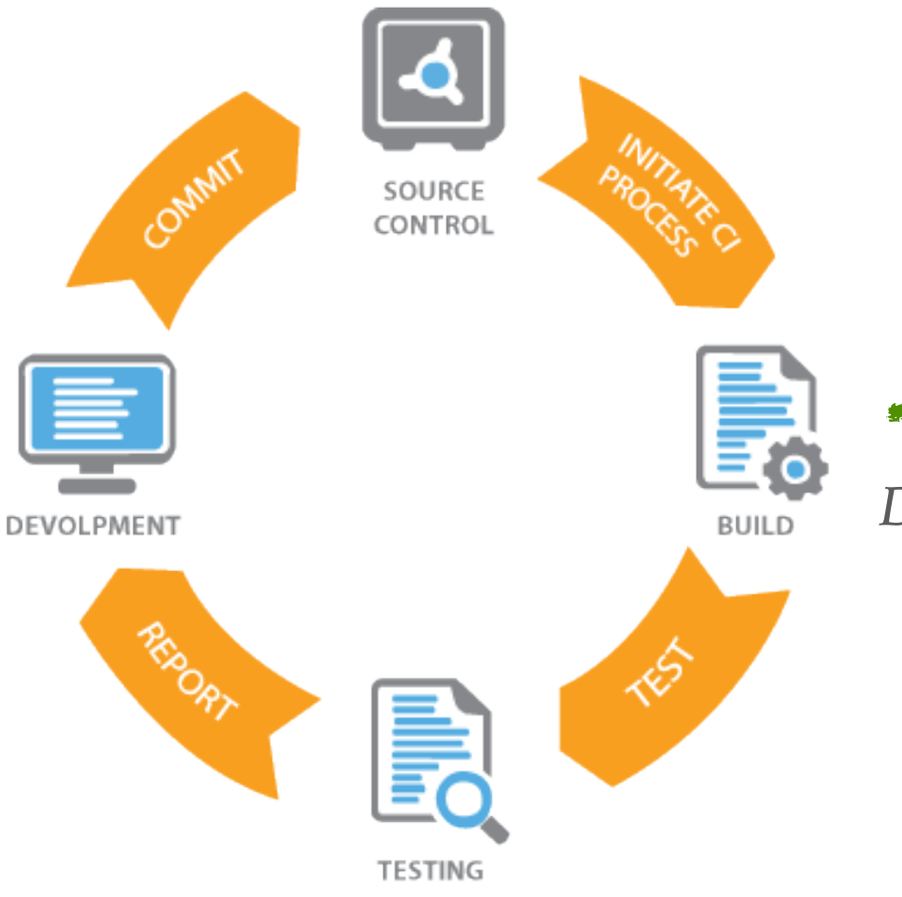
BUILDING ALICE SOFTWARE STACK

Giulio Eulisse (CERN)

DELIVERABLE

Continuous process



Daily build

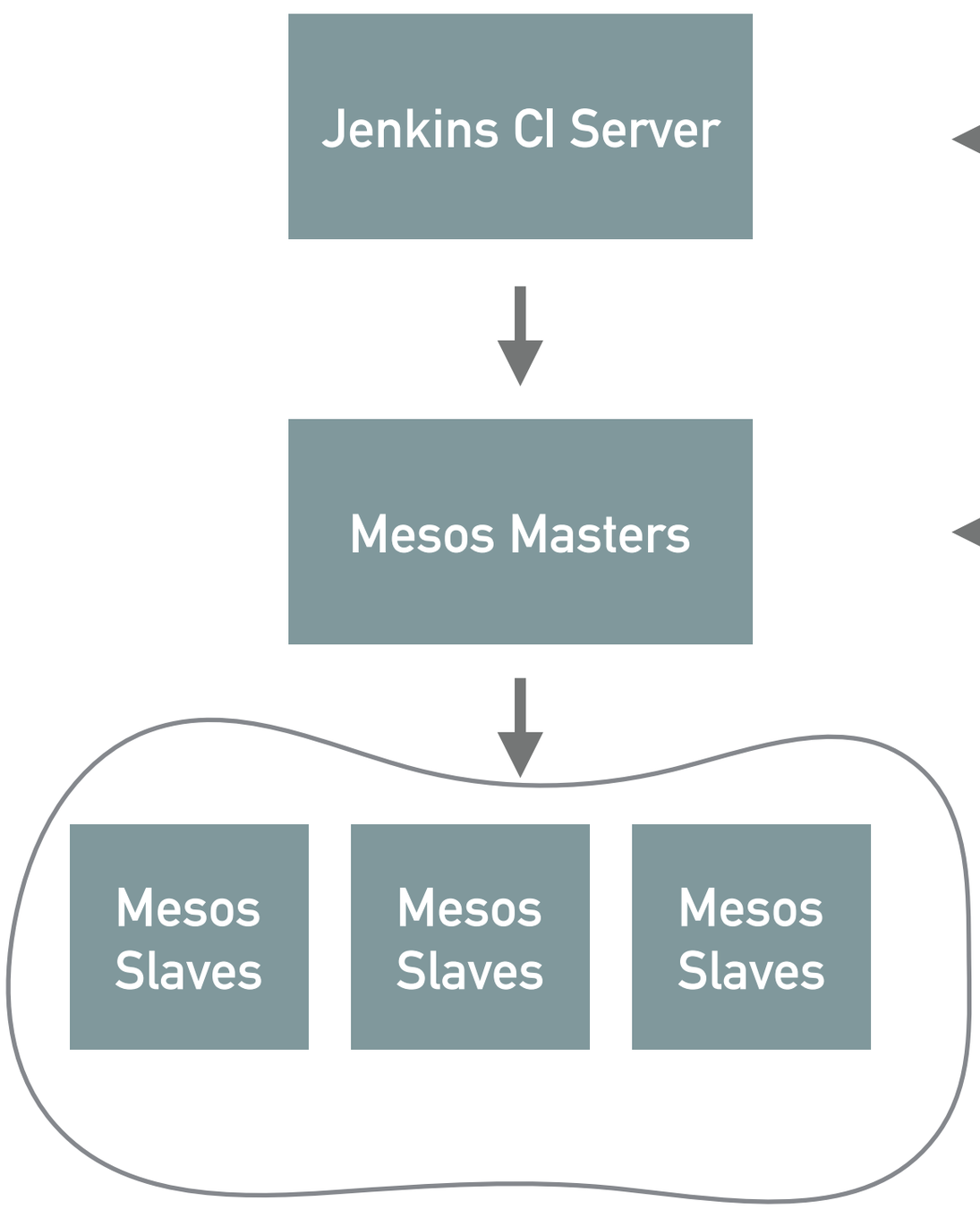
Deployment on CVMFS

Deployment on Packman

INFRASTRUCTURE

BUILD TOOL
&
RECIPES

INFRASTRUCTURE

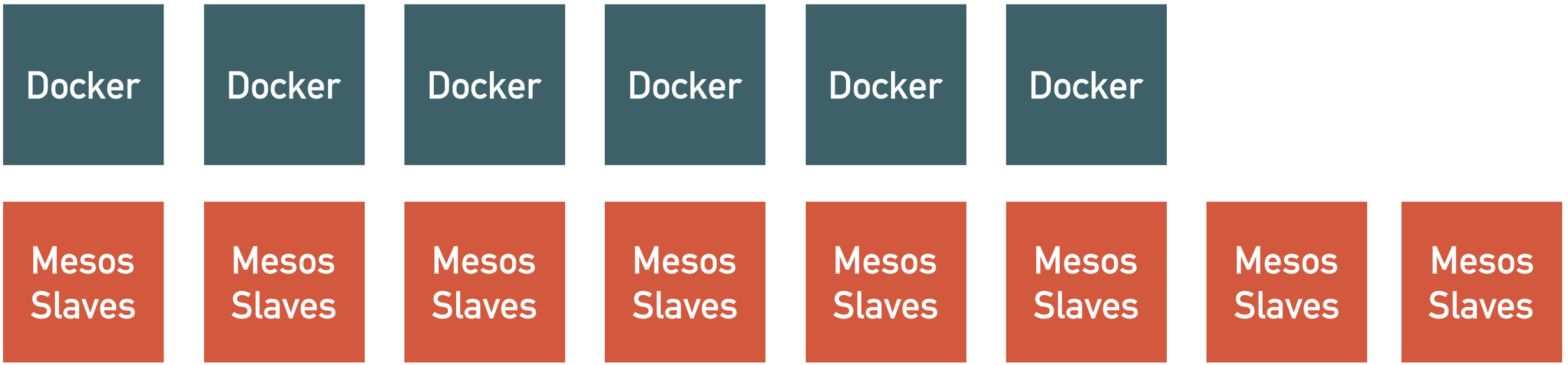


Industry standard, opensource, CI web application

We use Mesos cluster management software, providing HA, dynamic setup.

Workers are Docker containers running on Mesos Slaves. ***No more need for specially installed machines (e.g. slc5)***

INFRASTRUCTURE



OpenStack

CERN/IT provided. Roughly 150 cores, 300 GB of RAM in 20 or so VMs. All running Centos 7.

Linux bare metal

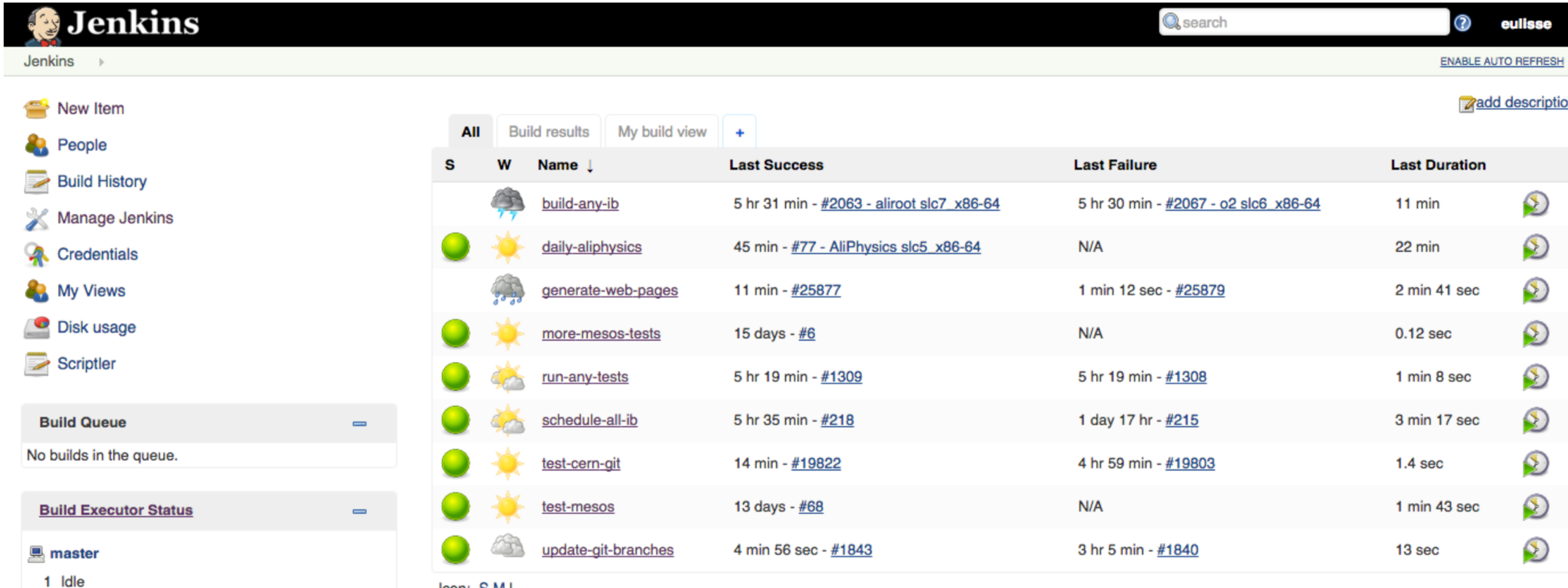
Few big boxes under ALICE control, kudos to Costin. All running latest Ubuntu.

OSX bare metal

One Mac Mini under ALICE control.

WHY?

TECHNOLOGY	MOTIVATION	NET RESULT
Jenkins	Workflow management, logging, reports	Click on a web page to define a job / browse build results.
Docker	Isolate application from infrastructure	slc5 builds on centos7 machines. Infrastructure machine configuration minimal.
Mesos	Avoid static partitioning of resources	Exploit of "peak only" usage patterns. Common layer between different schedulers.
OpenStack	Simplify hardware ownership and purchases	Get new VMs with a one line command / click on a web page.



https://alijenkins.cern.ch

Over 2K build jobs in the last 3 months, 4 different architectures plus special builds. Used both for test builds and ***production releases****.*

Provisioning & scheduling

openstack™

Apache MESOS™

Configuration management / deployment

puppet labs®

ANSIBLE

docker

Continuous integration

Jenkins

Log parsing & mining

elasticsearch

Monitoring & Results

MonALISA
MONitoring Agents using a Large Integrated Services Architecture

kibana

CDash

BUILD TOOL: ALIBUILD

Open Source

*Tool itself can be found at **https://github.com/alisw/alibuild**, actual recipes to build externals are at **https://github.com/alisw/alidist**.*

Standalone

Python is the only dependency. Does not depend on packaging technology, produces tarballs.

No magic, compact, maintainable

*Tool itself is **527 SLOCs of Python** + **116 SLOCs of Bash**. Recipes are simple Bash scripts with a YAML header.*

Git based workflow, reproducible builds

*Configuration management happens in git. **A given checkout of alidist corresponds to a given configuration**.*

Not a CMake / make / (pick your favourite tool) replacement

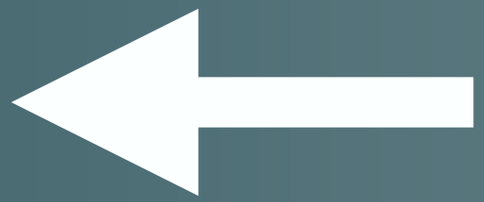
INSTANT GRATIFICATION

*Full documentation at **http://alisw.github.io/alibuild***

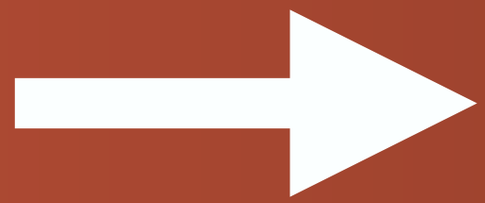
```
git clone https://github.com/alisw/alibuild
git clone https://github.com/alisw/alidist
alibuild/aliBuild -d -j 40 -a slc7_x86-64 build AliRoot
```

Any resemblance to other experiments naming conventions is purely fictional.

PRODUCTION



DEVELOPMENT



RECIPE EXAMPLE

YAML formatted metadata at the top.

```
package: AliRoot
version: %(commit_hash)s
requires:
  - ROOT
  - fastjet
build_requires:
  - CMake
source: http://git.cern.ch/pub/AliRoot
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT \
      -DROOTSYS=$ROOT_ROOT                            \
      -${FASTJET_ROOT:+-DFASTJET=$FASTJET_ROOT}
make -j 20
make install
```

RECIPE EXAMPLE

Bash recipe at the bottom. ***Conventions*** *over template magic / special languages.*

```
package: AliRoot
version: %(commit_hash)s
requires:
  - ROOT
  - fastjet
build_requires:
  - CMake
source: http://git.cern.ch/pub/AliRoot
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT \
      -DROOTSYS=$ROOT_ROOT                            \
      -${FASTJET_ROOT:+-DFASTJET=$FASTJET_ROOT}
make -j 20
make install
```

RECIPE EXAMPLE

Sources are always expected in git repositories. It simplifies source fetching logic a lot, allows automatic rebuilds when tip of a given branch changes.

```
package: AliRoot
version: %(commit_hash)s
requires:
  - ROOT
  - fastjet
build_requires:
  - CMake
source: http://git.cern.ch/pub/AliRoot
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT \
      -DROOTSYS=$ROOT_ROOT                            \
      -${FASTJET_ROOT:+-DFASTJET=$FASTJET_ROOT}
make -j 20
make install
```

DEPENDENCIES HANDLING

Consistent builds

When you change something in the recipe or in the tool itself, it notices and acts accordingly on a subsequent build of the tool and its dependencies. E.g. if you change ROOT recipe and try to rebuild AliRoot, it will notice.

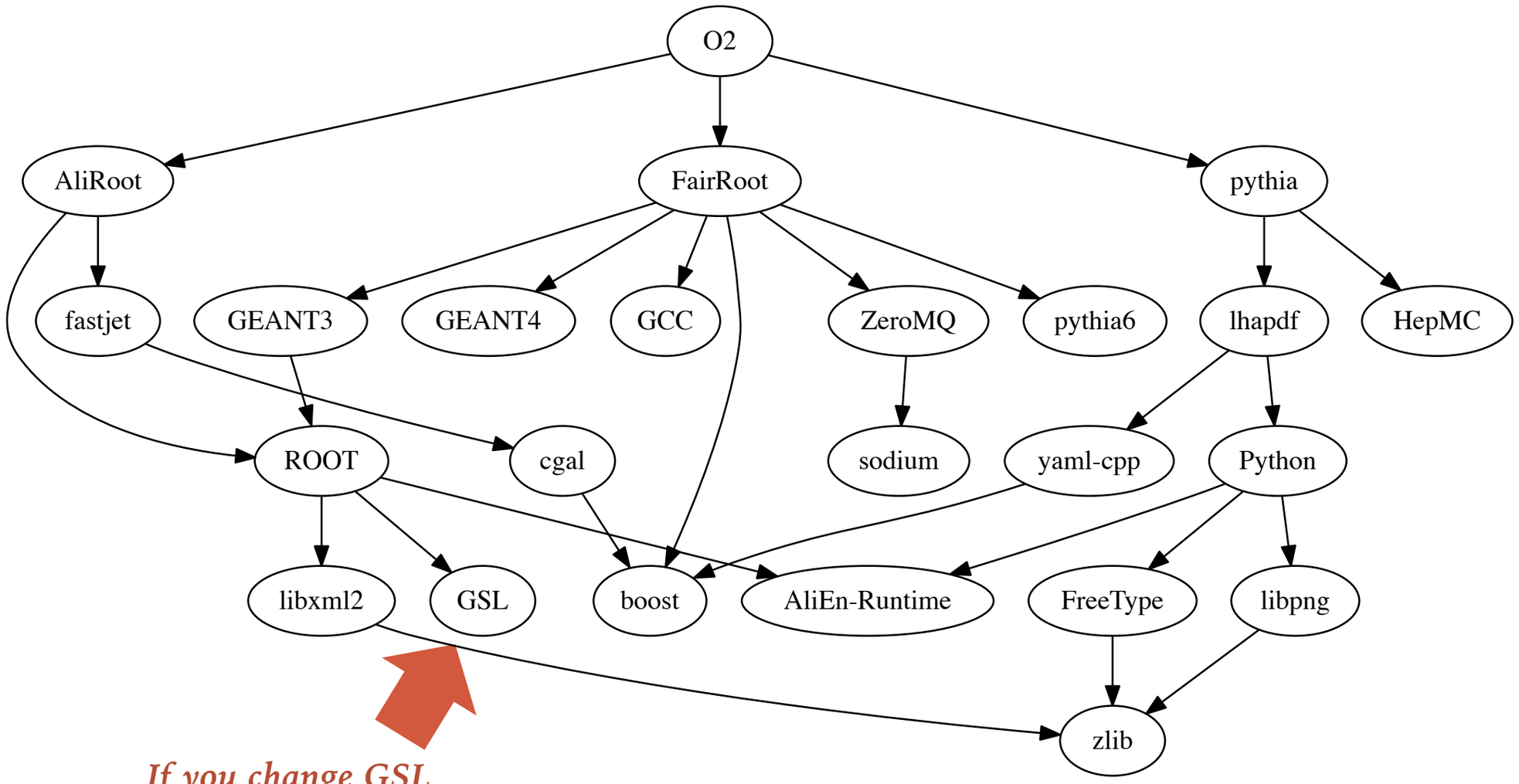
Flexible (and correct) handling of dependencies

Topological sort of the dependency graph for correct build order, even in the case of implicit dependencies. Run-time and build-time dependencies. Ability to disable dependencies. Platform (and soon experiment) specific dependencies. Parallel installations of externals.

Parallel installations

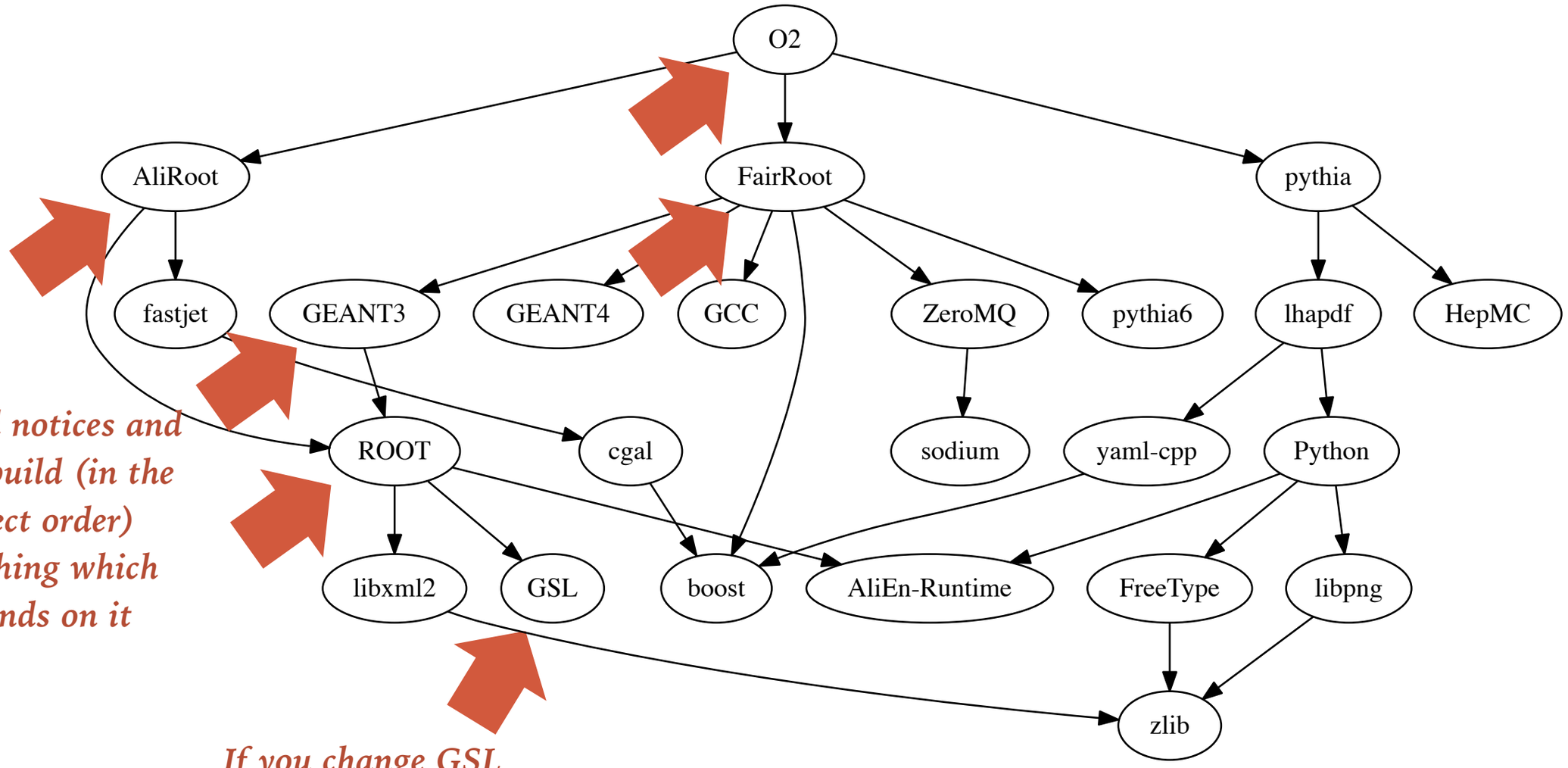
The requirement to have one common namespace for all our builds is not going away any time soon. Same for the ability to reuse dependencies between different builds. However, nothing prevents to regroup the packages at a later stage to simplify distribution.

CONSISTENT BUILDS



If you change GSL recipe or metadata...

CONSISTENT BUILDS



aliBuild notices and will rebuild (in the correct order) everything which depends on it

If you change GSL recipe or metadata...

PARALLEL INSTALLATIONS

```
../osx_x86-64/
        ../AliRoot/
                ../v1-1
                ../v2-1
                ../latest
        ../boost/
                ../latest
                ../v1.57.0-1
                ../v1.59.0-1
        ../fastjet/
                ../latest
                ../v3.1.3_1.017-1
                ../v3.1.3_1.017-2
                ../v3.1.3_1.020-1
../slc7_x86-64/...
```

Usual:

`<architecture>/<package>/<version>`

hierarchies.

Architecture is just a string, no platform auto-detection. It identifies the build host, not the installation or runtime requirements.

Changes in build recipes result in different "revisions".

BINARY REPOSITORY

Reuse builds

Packages built by one builder can be reused by other builders, ***even on a separate machine.*** *This comes handy when you cannot guarantee that you will always rebuild on the same machine.*

K.I.S.S.

The repository is just an object store with a bunch of symlinks to keep track of reproducibility.

Authoritative source

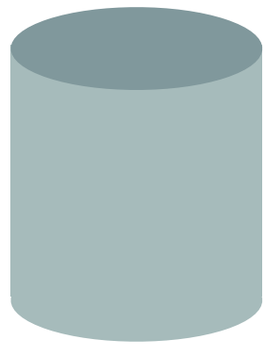
The binary repository acts as authoritative source to then do deployments.

BINARY REPOSITORY

time

Build ROOT 6.X.Y-1

Build package is uploaded to store



Later rebuilds of the same recipe (on a different machine / docker container) do not rebuild but download the binary package from the store.

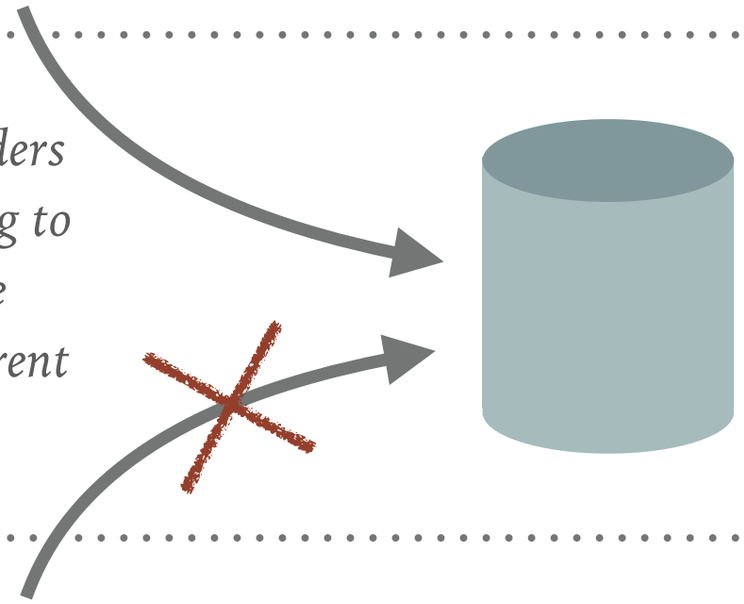
Reuse ROOT 6.X.Y-1

BINARY REPOSITORY

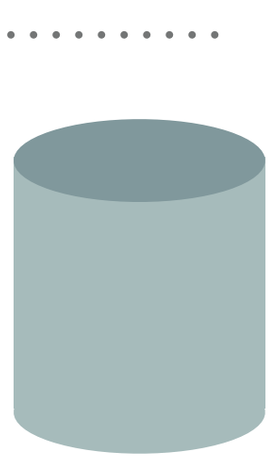
time

Build ROOT 6.X.Y-1
using recipe A

Two different builders might end up trying to upload the same package from different recipes.



Only the first one succeeds, the other one does not rebuild but repackages everything and tries again.



Build ROOT 6.X.Y-1
using recipe B

Repackage to
ROOT 6.X.Y-2

CVMFS / YUM / APT SUPPORT

time

Build ROOT 6.X.Y-1

Build package is uploaded to store

TARS

*Tarballs are converted to RPMs / DPKG via FPM, store metadata is used to construct a **standard** yum / apt repository*

aliPublish

CVMFS

YUM / RPMS

APT / DEBS

DEVELOPER MODE

aliBuild can pick up sources from local checkouts for the builds. After the first build, one can go inside the build directory and type "make install". Handy for those who need to develop externals while improving the application code (e.g. patching ROOT fixes).

```
git clone https://github.com/alisw/alibuild
git clone https://github.com/alisw/alidist
git clone https://github.com/root-mirror/root ROOT
alibuild/aliBuild -a slc7_x86-64 --devel ROOT build AliRoot
...
cd sw/BUILD/ROOT-latest/ROOT
make install
```

DOCKER SUPPORT

Simplify cross platform builds

*Handy for cross platform builds, i.e. using your Mac laptop to test builds in the slc7 environment. Just add "**--docker**" to the command line and the build will happen inside a container matching the provided architecture.*

Create docker containers (not implemented, yet)

It's trivial to extend the above to allow building and uploading of the containers with the results of the build itself.

DEFAULTS & DISABLING DEPENDENCIES

Support for common options

*A special recipe, called "**defaults-release.sh**" is added as a build requirement to each other. This recipe can be used to specify common options which affect global behaviour of the build, e.g. CXXFLAGS.*

Command line overwriteable

*Which defaults should be used can be specified on the command line via the "`--defaults <name>`" option. E.g. "**--defaults debug**" will add **defaults-debug.sh** everywhere as a dependency, setting CXXFLAGS="-g -O0" everywhere.*

Disabling dependencies

*It's possible to disable dependencies by using the command line option "**--disable**". Also in this case consistency of the build is ensured.*

ARCHITECTURE CUSTOMIZATIONS

$ARCHITECTURE

The system exports the command line provided architecture to the recipes as an environment variable.

Architecture specific dependencies

aliBuild supports architecture specific dependencies in the YAML preamble by adding a regular expression which needs to match for the requirements to be valid. E.g.:

```
name: AliRoot-tests
requires:
  - AliRoot
  - IgProf:slc7.*
```

You can then use `$<package>_ROOT` to detect if the dependency was included or not.

EXPERIMENT CUSTOMIZATIONS

aliBuild... alfaBuild... anyBuild!

*You can get experiment specific customisations (at the moment limited to a few details, like the name of the project hosting recipes) by simply using a symlink with the correct name. E.g. GSI people are experimenting with "**alfaBuild**" which uses the "**alfadist**" repository (i.e. "Florian is happy").*

Build customizations (idea)

*Build customisations can be driven the same way as the architecture ones. A **$FLAVOUR** can be defined as part of the build environment, depending on the tool name:*

E.g.: "aliBuild" ⇒ "FLAVOUR=ali", "alfaBuild" ⇒ "FLAVOUR=alfa".

```
name: ROOT
requires:
  - Alien: flavour=ali
  - Python: flavour=(panda|cbm)
```

HOW TO USE

To build a package:

```
git clone https://github.com/alisw/alibuild.git
git clone https://github.com/alisw/alidist.git
alibuild/aliBuild -d -a slc7_x86-64 -j 16 build AliRoot
```

To build a package in developer mode:

```
git clone https://github.com/root-mirror/ROOT
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --devel ROOT build AliRoot
```

To build a package in docker mode:

```
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --docker build AliRoot
```

To disable a package (and drop all its dependencies):

```
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --disable GEANT4 build O2
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --disable simulation build O2
```

SOURCECODE HANDLING

Git(hub/lab) based

The tool handles directly git repositories only. This simplifies enormously the code which takes care or managing the sources and provides nice, uniform, web based views.

Benefits

- Support for "moveable" builds without extra code.

- Support for changing repository without rebuilding (assuming the commit hash is the same).

- Easy backup / proxying / mirroring of sources. Fast downloads for daily builds if local "reference" clone is available.

SOURCECODE AND PATCHES

Patches are inevitable

Some bugfixes just cannot wait for the next ROOT release. Sometimes ZeroMQ does not compile on Mac and we are the first ones to find out (e.g. yours truly: https://github.com/zeromq/libzmq/pull/1483). The ***goal is to simplify contributing upstream, not to fork****.*

Policy on how to handle external sources

Policy over tools. Current one I wrote and we use:

https://github.com/alisw/alidist#guidelines-for-handling-externals-sources

Policy is NOT mandatory

Of course there are cases where we cannot redistribute sources. Preferred option would be use a protected git repository, if not even is an option, "curl inside the recipe" is of course not forbidden. You simply lose the benefits of dealing with git.

PROPOSED POLICY

If Sources hosted on git, used unmodified:

- *Directly refer to the Upstream repository.*

If Sources hosted on git, need patching:

- *Fork / mirror the **relevant parts** of the Upstream repository*
- *Pick a tag / commit which will be used as `<fork-point>`, create a branch "`alice/<fork-point>`". Apply Patches on top.*

If Sources are not hosted on git:

- *Create an ALICE mirror in some agreed location, e.g. https://github.com/alisw/*
- *Import a tar-ball with one Upstream version, commit it to git, tag it with the original tag.*
- *Create a branch "`alice/<fork-point>`" and apply Patches on top.*

PROPOSED POLICY: BENEFITS

Sources history can be browsed:

https://github.com/alisw/root

Patched Sources can be pin-pointed:

https://github.com/alisw/root/tree/alice/v5-34-30

Upstream Sources can be pin-pointed:

https://github.com/alisw/root/tree/v5-34-30

Changes between w.r.t. Upstream can be diff-ed

https://github.com/alisw/root/compare/v5-34-30...alice/v5-34-30

PROPOSED POLICY: BENEFITS

Sources history can be browsed:

https://github.com/alisw/geant4/

Patched Sources can be pin-pointed:

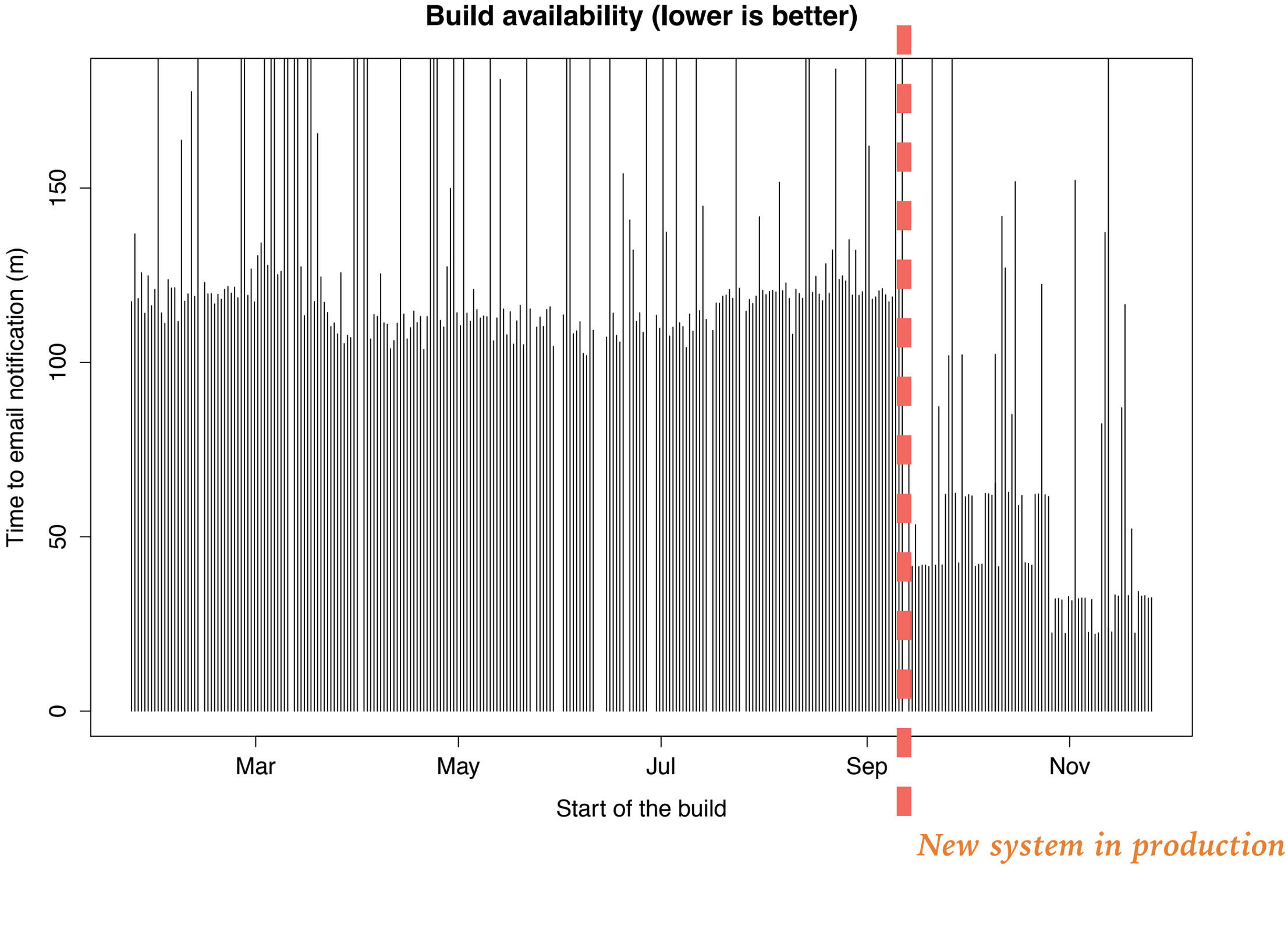
https://github.com/alisw/geant4/tree/alice/v4.10.01.p02

Upstream Sources can be pin-pointed:

https://github.com/alisw/geant4/tree/v4.10.01.p02

Changes between w.r.t. Upstream can be diff-ed:

https://github.com/alisw/geant4/compare/v4.10.01.p02...alice/v4.10.01.p02


Build availability (lower is better)
Time to email notification (m)
0
50
100
150
Mar
May
Jul
Sep
Nov
Start of the build
New system in production