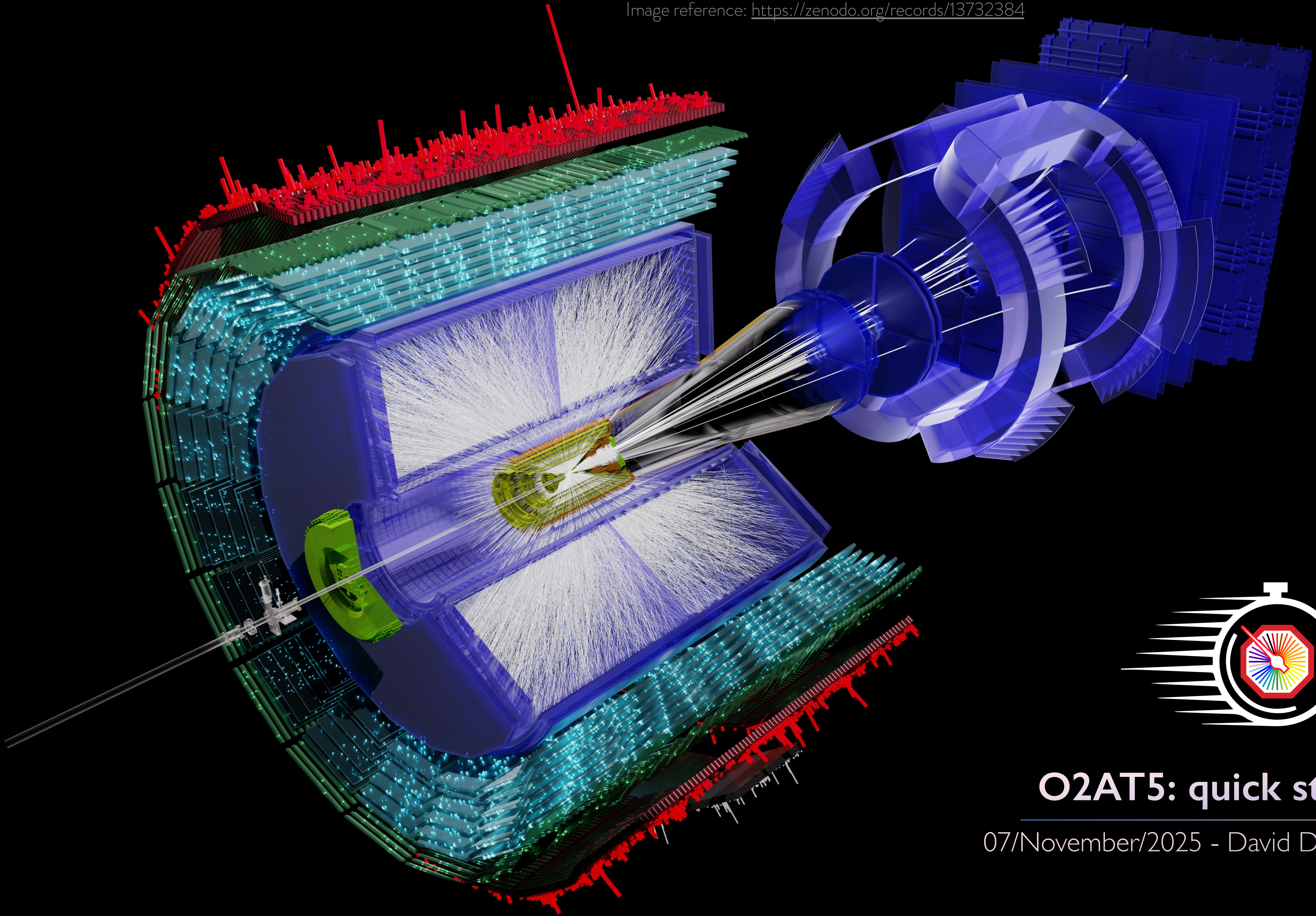
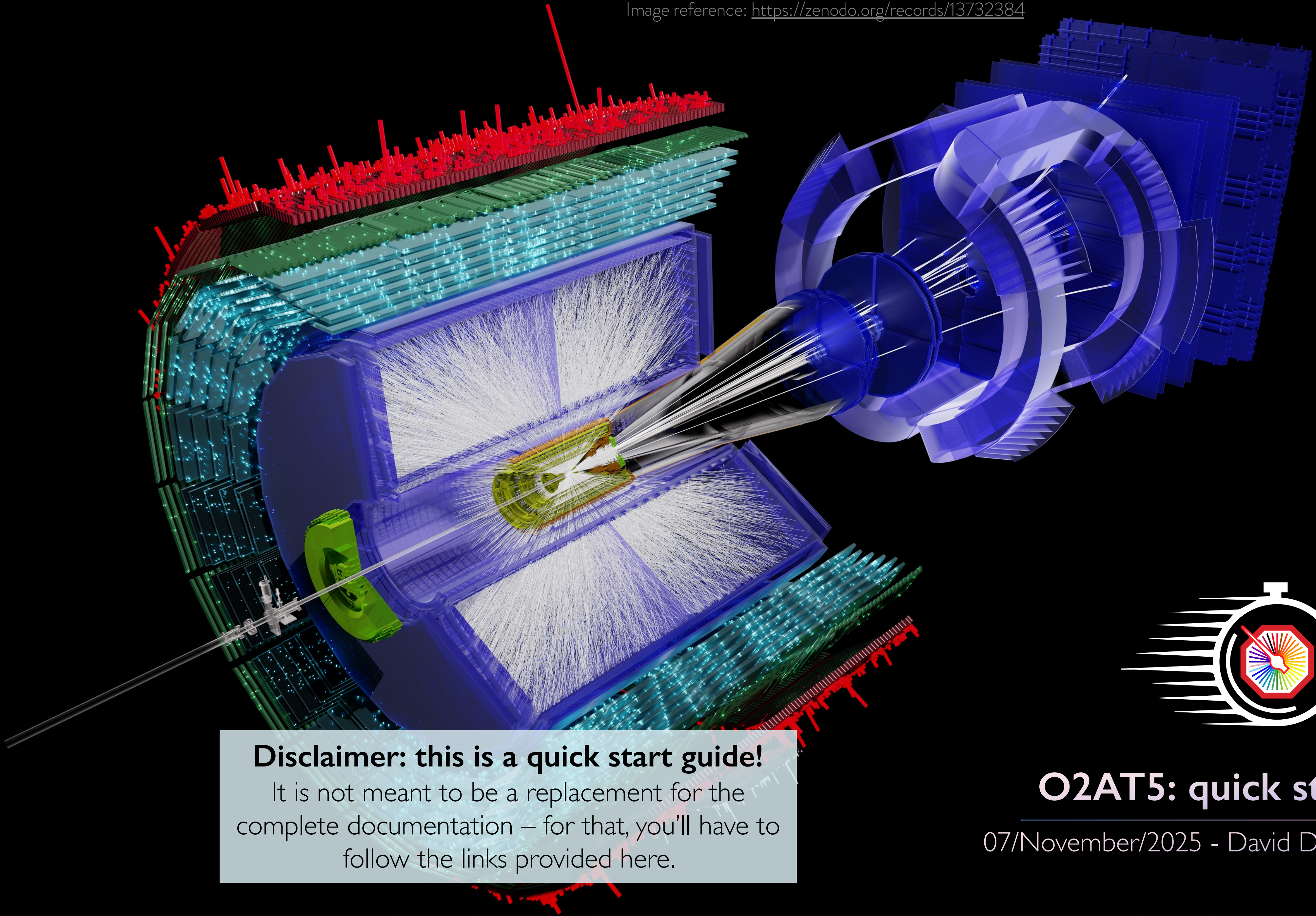
Image reference: https://zenodo.org/records/13732384



O2AT5: quick start guide

07/November/2025 - David Dobrigkeit Chinellato

Image reference: https://zenodo.org/records/13732384



Disclaimer: this is a quick start guide!
It is not meant to be a replacement for the complete documentation – for that, you'll have to follow the links provided here.

O2AT5: quick start guide

07/November/2025 - David Dobrigkeit Chinellato

Getting started in practice: resources in a nutshell

- **General documentation** page:
  - https://aliceo2group.github.io/analysis-framework/docs/
- **Install O2 and O2Physics** following these instructions:
  - https://aliceo2group.github.io/analysis-framework/docs/gettingstarted/installing.html
- Once installed, you may want to **contribute to the repository**. This involves using a git fork, see Anton's talk
- **Support**: when installing, working, analysing, scratching your head, … you may want to ask people questions!
  - For that, you can use mattermost: https://mattermost.web.cern.ch/
  - Mattermost is a communication platform that can be used in a web browser or you can download an app (PC/smartphone)
  - The first step in using mattermost is to join **the ALICE team** here:
    - https://mattermost.web.cern.ch/signup_user_complete/?id=jt4btzi83trgi83jpjnwpnkrfo&md=link&sbr=su
  - The second step is to join the O2 analysis tutorial channel (for tutorial-related support) here:
    - https://mattermost.web.cern.ch/alice/channels/o2-analysis-tutorial
  - The third step is to join **the O2 analysis channel** (for general support) here:
    - https://mattermost.web.cern.ch/alice/channels/o2-analysis

Installation: basic requisites and getting started

- Hardware requirements: none are specified in our guides. However, we:
  - Recommend a multi-core CPU to shorten compilation times
  - Recommend a computer with ample memory to avoid running out of memory when compiling
  - Rule of thumb: memory / CPU core must be generous to be safe (4GB/core is recommended)
  - If memory / CPU core is insufficient, compile with the "-j N" option (more on this later!)

- Supported platforms:
  - Primary: CentOS/AlmaLinux 8, AlmaLinux9,
  - Best-effort: macOS Ventura, macOS Sonoma, Ubuntu 20.04, 22.04, 24.04, Fedora, Linux Mint

- Build manager: aliBuild
  - Needs to be installed in the system once all requisites are installed (Ubuntu example here)
  - In ubuntu, Installed easily as a package via:
  - Other operating systems: look at docs

```
sudo add-apt-repository ppa:alisw/ppa
sudo apt update
sudo apt install python3-alibuild
```

Using alibuild to compile O2/O2Physics

- To get started with O2 and O2Physics, you have to create a directory to keep all O2/O2Physics code:

```
mkdir -p ~/alice          # creates an "alice"
cd ~/alice                # directory in your home
```

- You can then initialize the code repositories:

```
aliBuild init O2@dev                # initializes O2 (if interested in core dev)
aliBuild init O2Physics@master      # initializes O2Physics
```

- This downloaded O2, O2Physics and "alidist", a third git repository that bookkeeps the recipes of how aliBuild should compile (build) this software. Then all that is needed is:

```
aliBuild build O2Physics      # compiles O2Physics and all
                              # it needs (including O2!)
```

(this will take time!)

You can enter the O2Physics env with:

```
alienv enter O2Physics/latest
```

(new shell session)

```
alienv load O2Physics/latest
```

(load in current session)

MARIETTA BLAU INSTITUTE FOR PARTICLE PHYSICS

ALICE

Using alibuild to compile O2/O2Physics

- To get started with O2 and O2Physics, you have to create a directory to keep all O2/O2Physics code:

```
mkdir -p ~/alice
cd ~/alice
```

```
# creates an "alice"
# directory in your home
```

- You can then initialize the code repositories:

```
aliBuild init O2@dev
aliBuild init O2Physics@master
```

```
# initializes O2 (if interested in core dev)
# initializes O2Physics
```

- This downloaded O2, O2Physics and “alidist”, a third git repository that bookkeeps the recipes of how aliBuild should compile (build) this software. Then all that is needed is:

```
aliBuild build O2Physics
```

(this will take time!)

```
# compiles O2Physics and all
# it needs (including O2!)
```

If you have little memory: `aliBuild build -j N O2Physics` will use only N cores and, if you choose N smaller than the number of cores you have, you’ll have more memory per used core!

Reducing compilation scope of O2Physics

- When compiling the current O2Physics repository as of today: many objects need to be compiled!
- This takes time: typical laptops may need O(hours) to compile the entire repository
- A good option to work very efficiently is to **reduce the scope of O2Physics compilation** by selecting only a few subdirectories to compile
- Let's use the tutorial example! For the general hands-on, **you will only need code from 'Common' and 'Tutorials' to be compiled**, so you can skip all PWG directories. To do so, use the command line:

(do this in the ~/alice directory)

```
O2PHYSICS_COMPONENTS="Tutorials/install Common/install" aliBuild build O2Physics
```

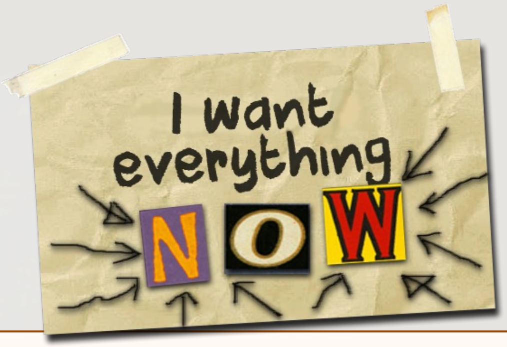
Technically: O2PHYSICS_COMPONENTS is an environmental variable that is a space-separated list of components that one wants aliBuild to compile. A typical approach may be to put this entire line in a "o2build.sh" script inside ~/alice and possibly also add further necessary directives such as "-j [N]", where "N" is the number of cores to use for compilation, to this o2build.sh script so they're always used.

- Reducing to Common + Tutorial will **reduce number of compiled objects by ~5x**
  - Add your favorite PWG here too so you're covered for the corresponding hands-on!
- **Further compilation directives** that should be very useful and can be added before 'aliBuild':
  - `ENABLE_UPGRADES=ON` enables the compilation of ITS3, ALICE 3 code that would otherwise not be compiled

MARIETTA BLAU INSTITUTE FOR PARTICLE PHYSICS

Instant gratification:

The recommended installation for the tutorial



```
mkdir -p ~/alice                      # creates an "alice"
cd ~/alice                            # directory in your home
```

```
aliBuild init O2@dev                  # initializes O2 (if interested in core dev)
aliBuild init O2Physics@master        # initializes O2Physics
```

```
export O2PHYSICS_COMPONENTS="Tutorials/install Common/install"   # small compilation
aliBuild build O2Physics                                         # compile!
```

- IMPORTANT: do this before the actual tutorial and especially before the Hands-on!
  - Compilation takes time! Make sure it finished successfully so you're set
- **If you're following PWG hands-on session(s),** add also "PWGxx/install" to O2PHYSICS_COMPONENTS
- Add also "-j N" after "aliBuild" if you do not have a lot of memory
  - The parameter N should be at most (available RAM) / 4GB (mac) or (available RAM) / 6GB (linux)

(for completeness: this assumes memory is more limiting than N_{cores}; ideally, N should be min[(available RAM) / 6GB, N_{cores}]

Surgical compilation: ninja to the rescue

- **If you have a compiled O2Physics installation** and just want to recompile one subdirectory or even one particular executable, this can be done using a tool called “ninja” – it will help you be faster!

- Go to your build directory using:

Bear in mind: different path needed for any non-standard installation path

```
cd ~/alice/sw/BUILD/O2Physics-latest/O2Physics
```

- Once in that directory, load the build environment doing:

```
direnv allow
```

- You can then rebuild only one specific subdirectory of O2Physics:

```
ninja <directory>/install
```

```
# Example: ninja PWGCF/Tasks/install
```

- You can even rebuild only one single executable:

```
ninja stage/bin/<target>
```

```
# Example: ninja stage/bin/o2-analysis-cf-correlations
```

Important information

- The directories alidist, O2 (if initialized) and O2Physics will appear under ~/alice
- These are **git clones** of the respective git repositories: alidist, O2, O2Physics (these links: web browsing)
- They contain the code you have compiled and you may want to develop code in O2Physics
  - …or even in O2 if you eventually do core O2 business (typically expert use, but hey, you can do that too!)

- These directories do not update themselves!
  - In case O2Physics stops compiling after a git pull (update), consider updating O2
  - In case O2 stops compiling after a git pull (update), consider updating alidist and aliBuild (should be rare!)
  - If it fails, please check the typical problems page [1] and these instructions [2] for finding errors.
  - If compilation still fails, you can ask for help on mattermost.
    - If so: please provide basic details about your installation. You can use this tool [3] to do that

  - **Important**: be cautious when issuing git commands! Recompilation takes time
    - Being careful with that will also be important during the tutorial!
    - if you trigger a full recompile, you will be stuck for a while and won't be able to do the exercises
  - **But**: we have a tool that can help with that!

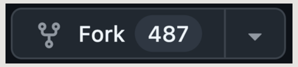
[1] https://aliceo2group.github.io/analysis-framework/docs/troubleshooting/#typical-compilation-problems-and-solutions
[2] https://aliceo2group.github.io/analysis-framework/docs/troubleshooting/#compilation-problems
[3] https://aliceo2group.github.io/analysis-framework/docs/tools/#setup-diagnostic-tool

MARIETTA BLAU INSTITUTE FOR PARTICLE PHYSICS

ALICE

Contributing to O2Physics: crash instructions to git work

- Create your own fork of O2Physics: initializes a repository to track your own work
- Done via web browsing by folllowing: https://github.com/AliceO2Group/O2Physics/fork
  - …or by clicking on the “fork” icon in the repository browsing



- Once you have a fork, you should add that fork to the git clone you have:

```
cd ~/alice/O2Physics
git remote add origin git@github.com:<your-github-username>/O2Physics.git
```

- Here, “origin” identifies how this fork will be identified locally. To visualize the forks you have as remote sources, do:

```
daviddc$ git remote -v
origin    git@github.com:ddobrigk/O2Physics.git (fetch)
origin    git@github.com:ddobrigk/O2Physics.git (push)
upstream  https://github.com/AliceO2Group/O2Physics (fetch)
upstream  https://github.com/AliceO2Group/O2Physics (push)
```

The remote add worked: You can now switch to edit your fork!

Fetching your work and handling branches

- You can fetch the latest version of your remote by doing:

```
daviddc$ git fetch origin
```

- By default, you'll be editing master (of the ddobrigk fork). You can then edit and do a commit and pull request:

```
daviddc$ git commit -a -m "My first commit"
daviddc$ git push origin
```

- This will push the commit to your remote fork only. You can then visualize it in the web browser and, if desired, also do a PR to the upstream repository (the "official" one) there.

- You can also organize your work into branches inside your fork by doing:

```
daviddc$ git switch -c newbranch #-c indicates 'create this branch'
# ………… do your work …………
daviddc$ git commit -a -m "My first commit"
daviddc$ git push -u origin newbranch
```

→ More information in Anton's talk

Handling pull requests: getting code approved

- Managing pull requests (PRs) will usually go through the web interface:
  - https://github.com/AliceO2Group/O2Physics
- We run a series of **"Continuous Integration" (CI) tests** to ensure that all PRs meet some quality criteria
- For a certain pull request to be approved, **two conditions must be met:**
  - All "required" checks must be successful. These include: formatting, macOS arm and linux compilation
  - Formatting can be made easier (automatic!) with pre-commit hooks: see documentation!
  - An approver (listed in O2Physics/CODEOWNERS) must approve your pull request
- **Optional**: linter checks (O2 and megalinter) are done on all PRs
  - You are encouraged to look at the warnings and errors reported by these tools
  - Some of the errors may point to code that misbehaves! Examples:
    - Use of "abs" instead of "fabs" or "std::abs" -> in some situations, "abs" typecasts float to int!
    - Always-true or always-false conditionals ("if(a = b)" instead of "if(a == b)", etc)
    - Uninitialized variable use could lead to undefined behaviour

      +…many more! → see also the O2 linter documentation page

Thank you!

Try it out yourself!

And get back to us in case of troubles…