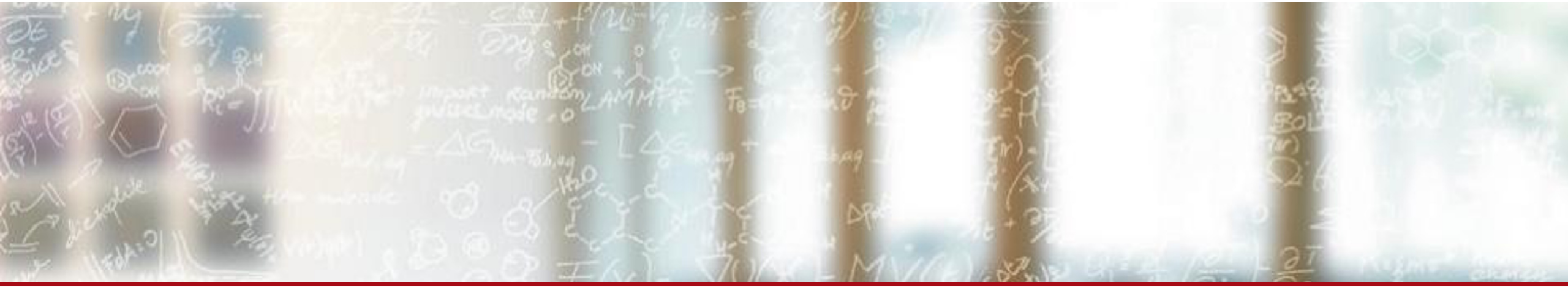
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Spack: an efficient workflow in 30 minutes

CSCS User Lab Day 2022 – Meet the Swiss National Supercomputing Centre

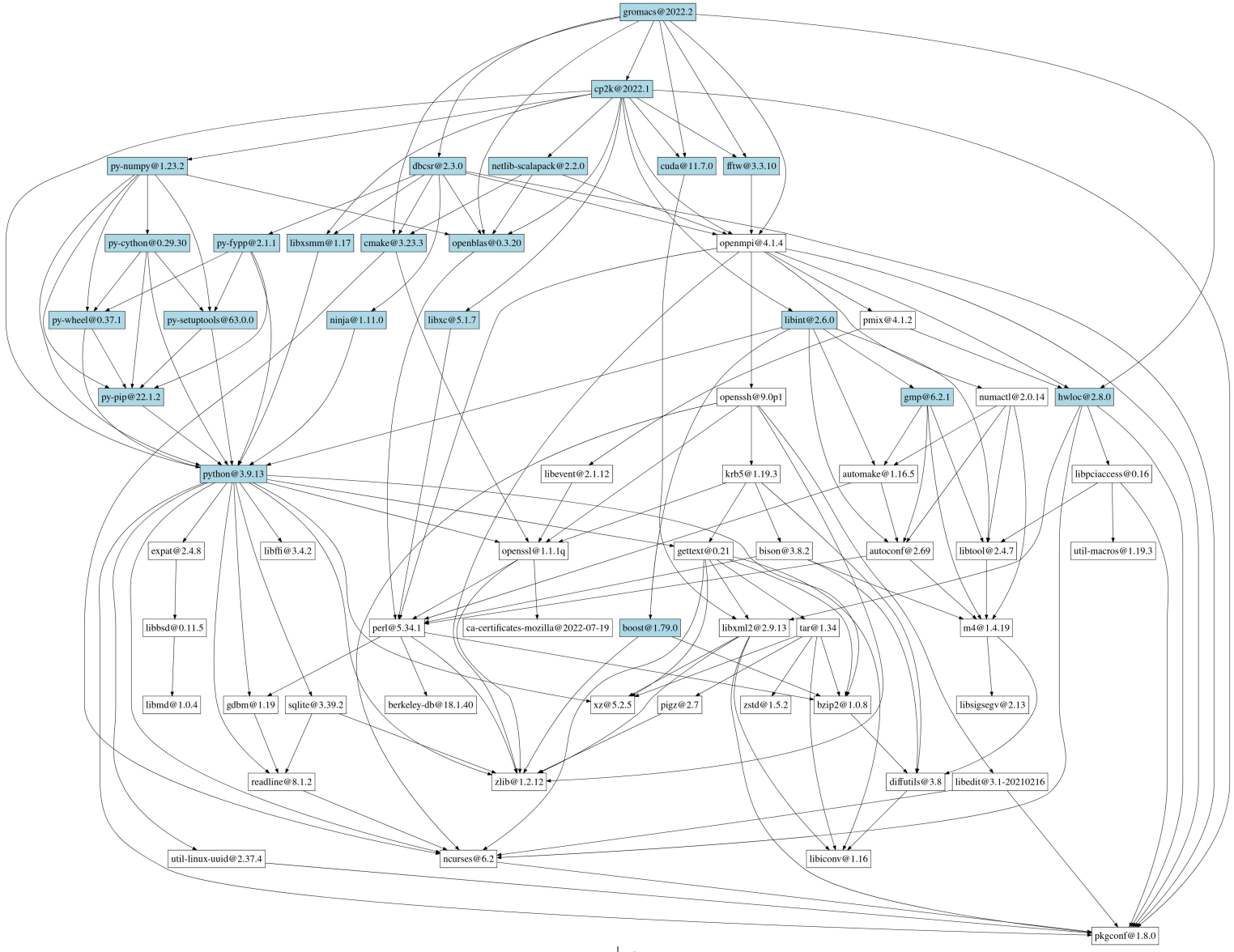
Harmen Stoppels and Alberto Invernizzi, CSCS

September 2nd, 2022

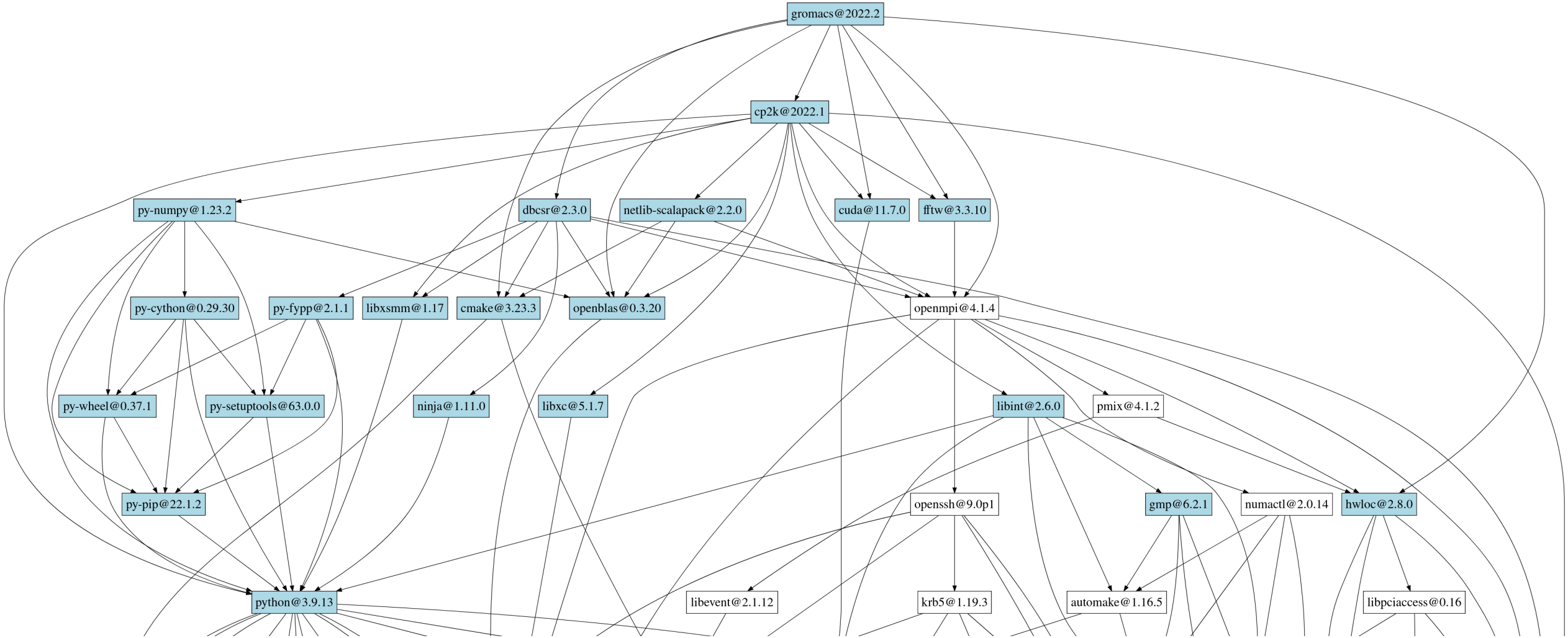
Motivation

- Scientific software can have a complex set of dependencies
- The same version of a software package can come in many flavors
  - Enabling/disabling features
  - Different build types (debug vs release)
  - Different compilers (gcc, clang, …)
- Manual installation is a tedious task

Example: gromacs (with cp2k & cuda support)



Example: gromacs (with cp2k & cuda support)



How does Spack solve this issue?

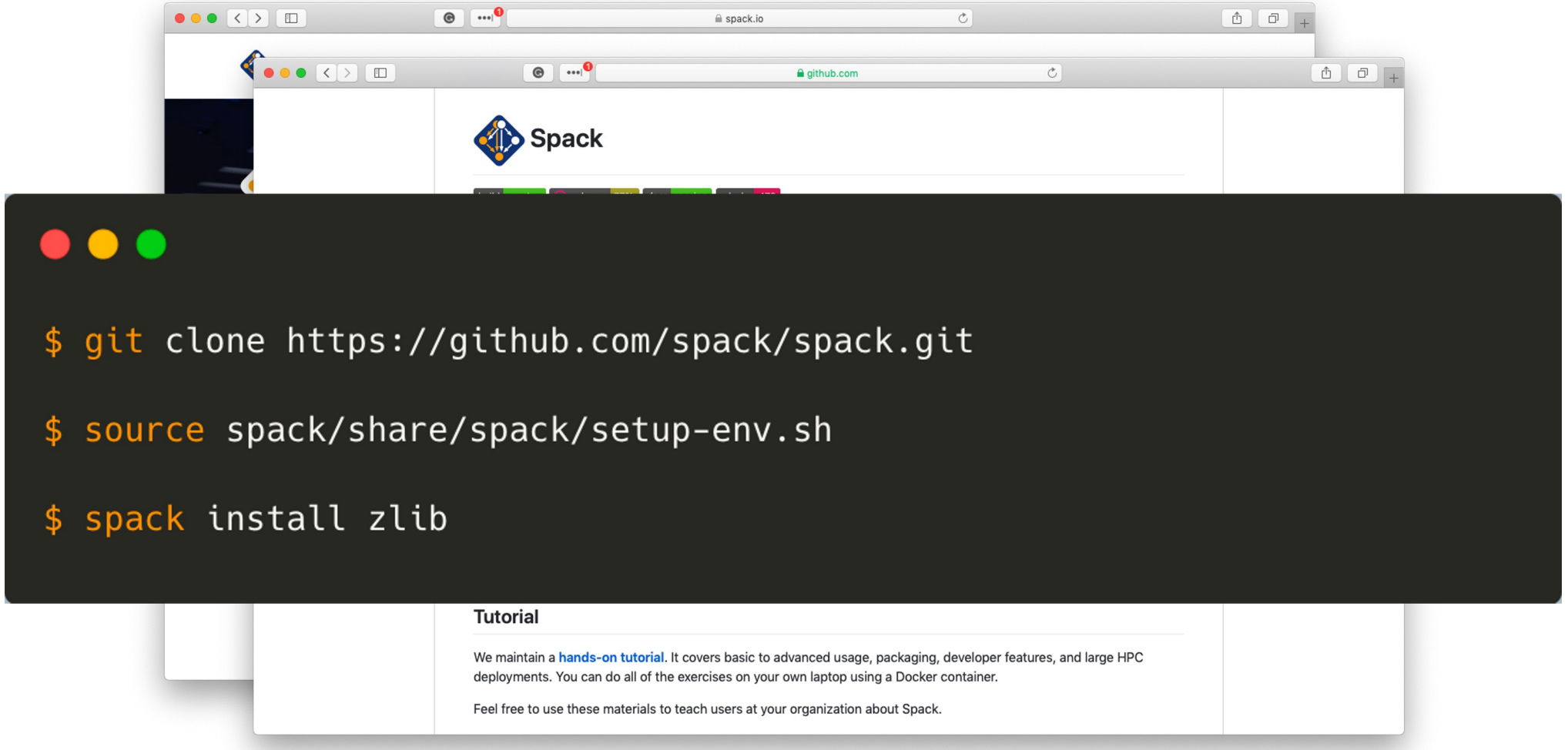
- Spack is a package manager
- It resolves dependencies automatically
- It is customizable with a simple `spec` syntax

```
$ spack install gromacs +cp2k +cuda ^cp2k +cuda cuda_arch=70
```

The above does not install just gromacs, but also its dependencies, and dependencies of dependencies…

Quick Start

Let’s start using it!



SPEC…in Spack

«[…] SPECS ARE MORE THAN A PACKA TO SPECIFY THE COMPILER, COMPILER VERSION, ARCHITECTURE, COMPILE OP »

official docs (spack.readthedocs.io)

```
hpx@1.3.0 %gcc@8.4                                    ne ~tools +examples
```

- **@**
- **%** n optional compiler version (gcc or gcc@4.7.3)
- **+** or **-** or **~** g, -qt, or ~qt) for boolean variants
- **name=<value>** cted to boolean variants
- **name=<value>** ags, cxxflags, fflags, cppflags, ldflags, and ldlibs})
- **target=<value> os=<value>**)
- **^<dependency-spec>** dependency on another package

```
➜  spack git:(develop) ✗ spack help --spec
spec expression syntax:

  package [constraints] [^dependency [constraints] ...]

  package                          any package from 'spack list', or
  /hash                            unique prefix or full hash of
                                   installed package


  constraints:
    versions:
      @version                     single version
      @min:max                     version range (inclusive)
      @min:                        version <min> or higher
      @:max                        up to version <max> (inclusive)

    compilers:
      %compiler                    build with <compiler>
      %compiler@version            build with specific compiler version
      %compiler@min:max            specific version range (see above)

    variants:
      +variant                     enable <variant>
      -variant or ~variant         disable <variant>
      variant=value                set non-boolean <variant> to <value>
      variant=value1,value2,value3 set multi-value <variant> values

    architecture variants:
      platform=platform            linux, darwin, cray, bgq, etc.
      os=operating_system          specific <operating_system>
      target=target                specific <target> processor
      arch=platform-os-target      shortcut for all three above

    cross-compiling:
      os=backend or os=be          build for compute node (backend)
      os=frontend or os=fe         build for login node (frontend)

    dependencies:
      ^dependency [constraints]    specify constraints on dependencies
      ^/hash                       build with a specific installed
                                   dependency

  examples:
      hdf5                         any hdf5 configuration
      hdf5 @1.10.1                 hdf5 version 1.10.1
      hdf5 @1.8:                   hdf5 1.8 or higher
      hdf5 @1.8: %gcc              hdf5 1.8 or higher built with gcc
      hdf5 +mpi                    hdf5 with mpi enabled
      hdf5 ~mpi                    hdf5 with mpi disabled
      hdf5 +mpi ^mpich             hdf5 with mpi, using mpich
      hdf5 +mpi ^openmpi@1.7       hdf5 wtih mpi, using openmpi 1.7
      boxlib dim=2                 boxlib built for 2 dimensions
      libdwarf %intel ^libelf%gcc
          libdwarf, built with intel compiler, linked to libelf built with gcc
      mvapich2 %pgi fabrics=psm,mrail,sock
          mvapich2, built with pgi compiler, with support for multiple fabrics
```

A successful workflow: environments setup

Creating an environment

1. Create an environment
2. Add a few packages or specs
3. Concretize
    a. Resolves dependencies
    b. Shows what will get installed
4. Install

Creating an environment

1. `$ spack env create my_env`
2. `$ spack -e my_env add py-scipy ^python@3.10`
3. `$ spack -e my_env concretize`
4. `$ spack -e my_env install`

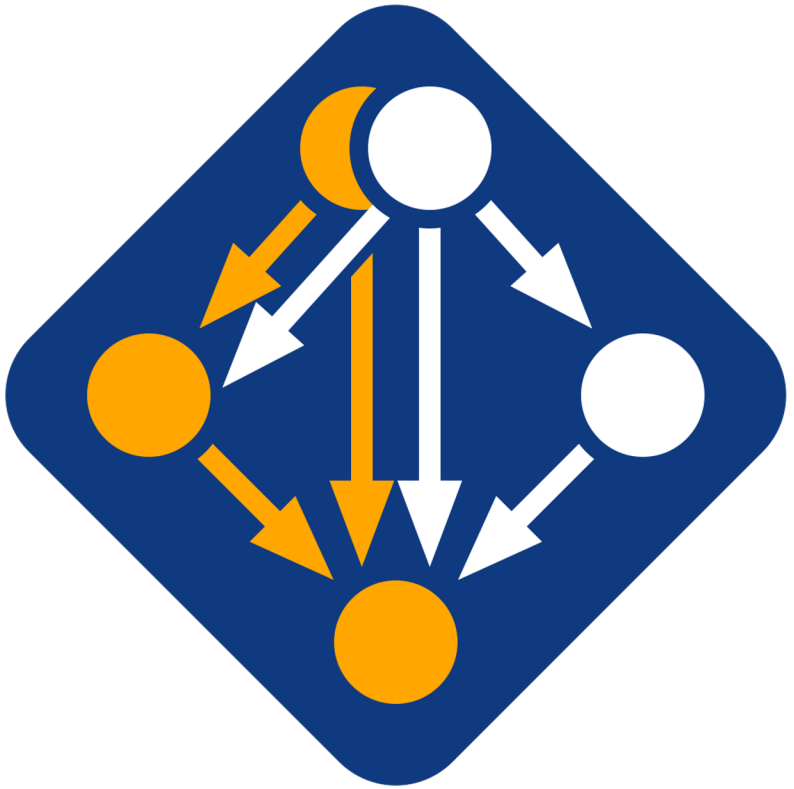
NOTE: `spack -e my_env` *makes subcommands work within the specified environment*

```
[my_env] $ spack -e my_env concretize
 -   ldnsr6x  py-scipy@1.8.1%gcc@11.1.0 arch=linux-ubuntu20.04-zen2
 -   n4b7nxt      ^openblas@0.3.20%gcc@11.1.0~bignuma~consistent_fpcsr~ilp64+locking...
[+]  7eix2j5          ^perl@5.30.0%gcc@11.1.0~cpanm+shared+threads arch=linux-ubuntu...
 -   y3k6mww      ^py-cython@0.29.30%gcc@11.1.0 arch=linux-ubuntu20.04-zen2
 -   x37gqeq          ^py-pip@22.1.2%gcc@11.1.0 arch=linux-ubuntu20.04-zen2
 -   glduetc              ^python@3.10.6%gcc@11.1.0+bz2+ctypes+dbm~debug+libxml2+lzm...
[+]  7bdbbqq                  ^bzip2@1.0.8%gcc@11.1.0~debug~pic+shared arch=linux-ub...
[+]  d2y4dtt                      ^diffutils@3.8%gcc@11.1.0 arch=linux-ubuntu20.04-z...
[+]  eudajht                          ^libiconv@1.16%gcc@11.1.0 libs=shared,static a...
[+]  355ji4i                  ^expat@2.4.8%gcc@11.1.0+libbsd arch=linux-ubuntu20.04-...
[+]  n3req45                      ^libbsd@0.11.5%gcc@11.1.0 arch=linux-ubuntu20.04-z...
[+]  vpv4x3u                          ^libmd@1.0.4%gcc@11.1.0 arch=linux-ubuntu20.04...
[+]  dvjhmhy                  ^gdbm@1.19%gcc@11.1.0 arch=linux-ubuntu20.04-zen2
[+]  sepfikp                      ^readline@8.1.2%gcc@11.1.0 arch=linux-ubuntu20.04-...
[+]  yv6danh                          ^ncurses@6.2%gcc@11.1.0~symlinks+termlib abi=n...
[+]  qi3rmzy                              ^pkgconf@1.8.0%gcc@11.1.0 arch=linux-ubunt...
[+]  dgk5wrw                  ^gettext@0.21%gcc@11.1.0+bzip2+curses+git~libunistring...
[+]  s4imhr2                      ^libiconv@1.16%gcc@11.1.0 libs=shared,static arch=...
[+]  p7orov2                      ^libxml2@2.10.1%gcc@11.1.0~python arch=linux-ubunt...
[+]  l4oscwa                          ^pkgconf@1.8.0%gcc@11.1.0 arch=linux-ubuntu20....
```

What's in an environment?

👩 spack.yaml → spack concretize → 🤖 spack.lock

```yaml
spack:
  concretizer:
    unify: true
  specs:
  - py-scipy
  - python@3.10
```



```json
{
  "_meta": {
    "file-type": "spack-lockfile",
    "lockfile-version": 4,
    "specfile-version": 3
  },
  "roots": [
    {
      "hash": "ldnsr6xrkr3dfskktawbep7y5rekcfak",
      "spec": "py-scipy"
    },
    {
      "hash": "glduetcfm24cxra64jy5lbtilpspok2z",
      "spec": "python@3.10"
    }
  ],
  "concrete_specs": {
    "ldnsr6xrkr3dfskktawbep7y5rekcfak": {
      "name": "py-scipy",
      "version": "1.8.1",
```

Modifying an environment

Edit the **`spack.yaml`** file directly:

```
$ spack -e my_env config edit
```

Or

```
$ spack location -e my_env
/path/to/my_env

$ vim /path/to/my_env/spack.yaml
```

Modifying an environment (different ways, same result)

```yaml
spack:
  concretizer:
    unify: true
  specs:
  - py-scipy
  - python@3.10
  - intel-oneapi-mkl
```

```yaml
spack:
  specs:
  - py-scipy
  - intel-oneapi-mkl
  concretizer:
    unify: true
  packages:
    python:
      require: '@3.10'
```

```yaml
spack:
  specs:
  - py-scipy ^python@3.10 ^intel-oneapi-mkl
```

Iterating concretization

1.
```
$ spack -e my_env config edit
```
2.
```
$ spack -e my_env concretize --force
```
3.
```
$ spack -e my_env install
```

Iterate until happy!

A successful workflow: environments usage

Using the installed environment

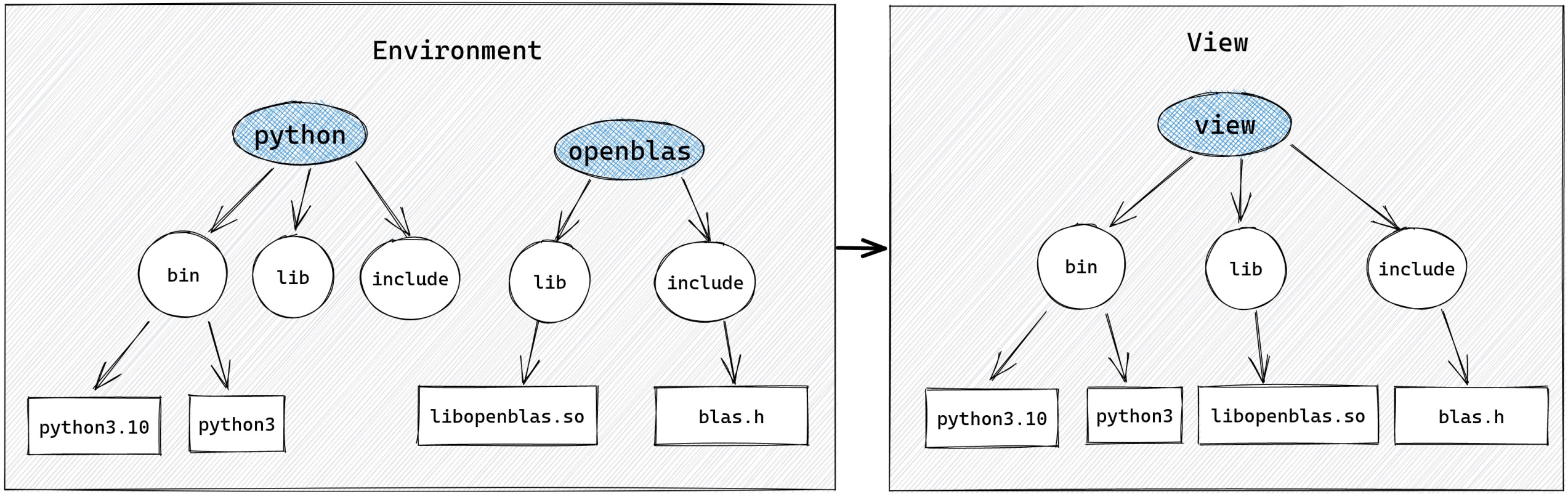
Typical interaction with a programming environment:

```
module swap PrgEnv-cray PrgEnv-gnu
module load gcc/10
module load …
```

Not without issues

- **Consistency**: compatibility & dependencies across modules
- **Opaqueness**: many, many environment variables
- **Persistence**: what modules did I load last time?

Using the installed environment: views



Using the installed environment: views

- Persistent on disk
- Just a handful of environment variables
- Consistent set of libraries

```
$ srun ./view/bin/my_app

$ gcc -o app -I view/include app.c -L view/lib -lfoo
```

Configuring views

```
spack:
  view: /path/to/view
  concretizer:
    unify: true
  specs:
  - py-scipy
  - python@3.10
  - intel-oneapi-mkl
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Spack at CSCS

Spack at CSCS: Piz Daint

Many packages are already provided by the system, including optimized libraries (e.g. cray-mpich) and compilers.

CSCS provides spack configuration files for default CDT/CPEs that can be easily used to let your personal spack instance identify main packages, so that they can be used as dependencies in spack.

```
$ module load daint-gpu          # or module load daint-mc

$ module load spack-config
```

Spack at CSCS: Alps

- Many Spack packages come pre-installed
- Spack 0.18+ prefers reuse over the latest the greatest
- Decoupled from the Cray PE

- As a user: less time spent configuring and building from sources

Environments and replicability

👩 `spack.yaml` vs 🤖 `spack.lock`

describes **requirements**

- they can be easily share with users on <u>different machines</u>
- what is needed to get a correct working environment

describes **how to exactly reproduce the full installation**:

- it's safer to use this only on the <u>same machine</u>
- reproduce exactly the same working environment

```
$ spack env create new_env /path/to/spack.{yaml/lock}
```

you can use any file you want, or if you want to clone another environment

```
$(spack location -e other_env)/spack.yaml
```

…and it’s not just for HPC!

It is a useful tool in your local development environment too!

- It keeps libraries organized with all the variants
  - debug/release
  - with or without a specific option
  - different versions
- It eases the installation process
  - and it builds from source with the microarchitecture optimizations

It is not just for libraries! You can find also tools like

- cmake
- tmux
- ...

And also compilers!

- gcc
- clang
- ...

References

Documentation

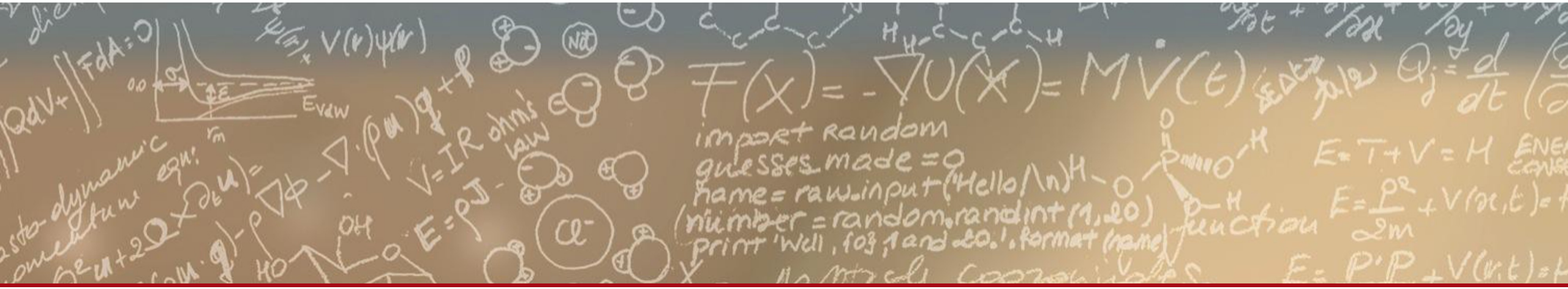
https://spack.readthedocs.io/

GitHub Repo

https://github.com/spack/spack

Slack channel

https://slack.spack.io



~~The end~~ A new beginning

Thank you for your attention.