How to use spack – solving the dependency problem

- Building your model with netcdf-fortran (modules)

```
$ module load netcdf-fortran/4.5.3-openmpi-4.1.2-intel-2021.5.0
```

- Problem: long list of dependencies like MPI, hdf5, parallel-netcdf and netcdf-c not accessible with the module system
- Solution: use spack to list them all

```
$ module show netcdf-fortran/4.5.3-openmpi-4.1.2-intel-2021.5.0
$ spack find -dp /k6xq5g
```

How to use spack – module vs. spack

- Module system is easy to use but does not help much for building your own software
  - Users need to set compilation and linking commands
  - Might lead to unstable binaries, which depend on the loaded modules at runtime
- Spack is more complex but allows to build packages with stable dependencies
  - Linking paths are written into the binaries so they know where to look for their needed libraries
  - Software can be build completely in user-space -> user can be more independent of the software tree (similar to conda)

How to use spack – build your own package (I)

Example: you want an older CDO version

- Create your own installation directory and set up the spack config accordingly:

docs.dkrz.de/doc/levante/code-development/building-with-spack.html

- Check the existing compilers

```
$ spack compilers
```

- Check the available cdo versions

```
$ spack info cdo
```

How to use spack – build your own package (II)

- Select compiler and version and check what spack would build

```
$ spack spec -I cdo@1.9.2 %gcc
```

- Since spack is designed for HPC centers, using MPI is default. Can be disabled to speed up building process

```
$ spack spec -I cdo@1.9.2 %gcc ^fftw~mpi ^hdf5~mpi ^eccodes
```

- If everything looks ok, start the build

```
$ spack install cdo@1.9.2 %gcc ^fftw~mpi ^hdf5~mpi ^eccodes
```