esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

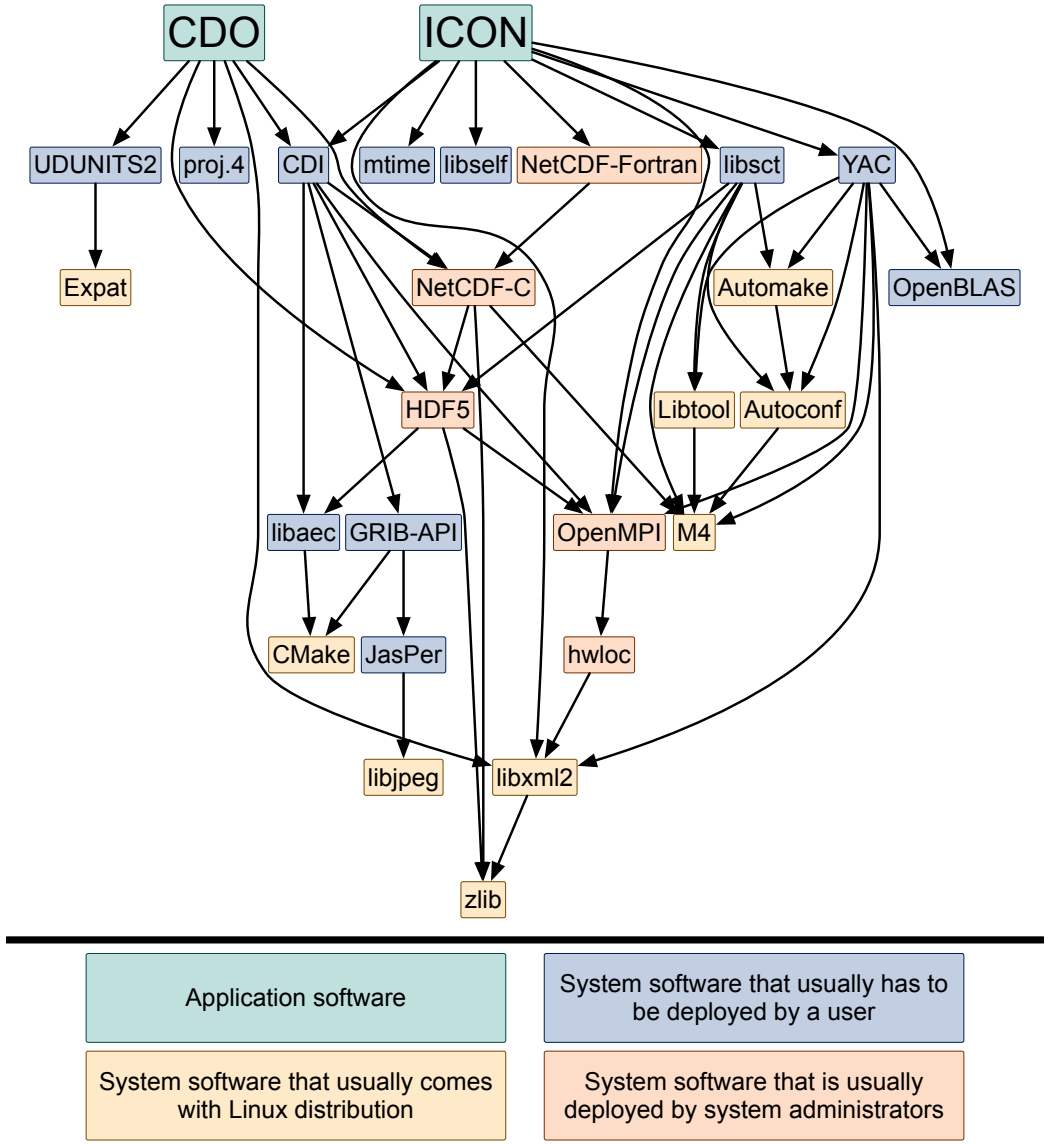
Spack for ESM: Experiences and Plans

Sergey Kosukhin

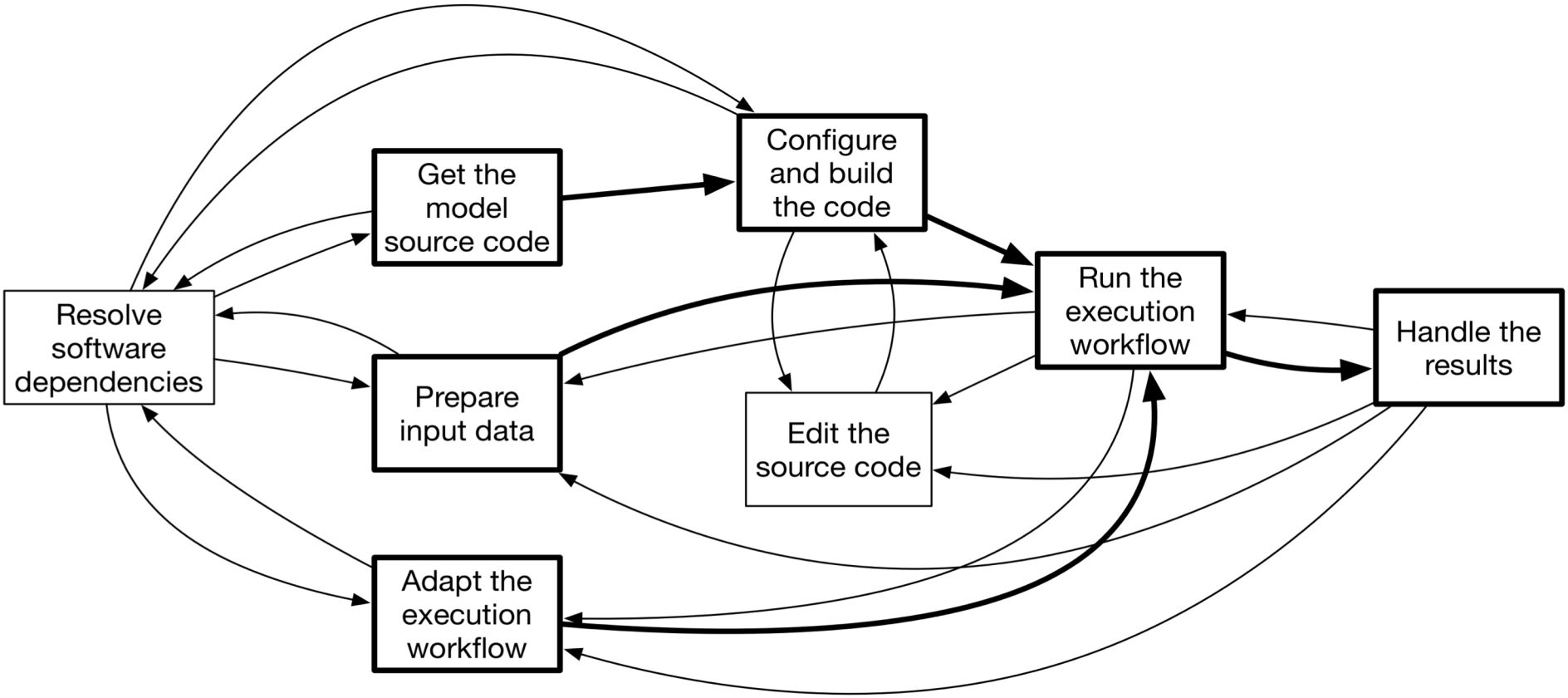
Max Planck Institute for Meteorology

The ESiWACE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 675191

Software stack



The technical part of the research workflow



Problem

Fuzzy questions that need to be answered before an experiment can be started

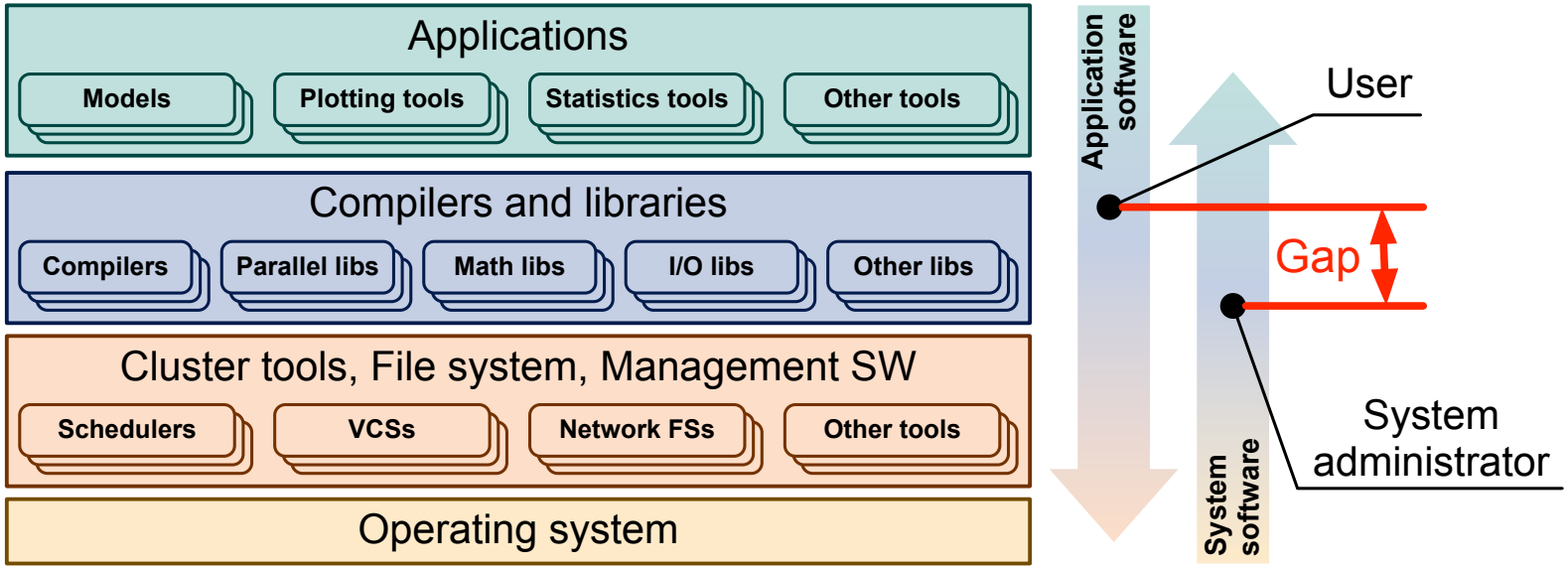
Different environments

Single machine or HPC site?
What software is already there?
What privileges do I have?

Different requirements

Which versions?
Which features to enable?
Which API implementation?

Transition between areas of responsibility



Solutions

- Install Manually
- Binary package managers
  - Designed to manage a single, stable and well tested stack.
  - Install one version of each package in a single prefix (/usr).
- Port systems
  - Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.
- Virtual Machines and Linux Containers
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
- Other HPC package managers (EasyBuild)
  - Intended for system administrators

Spack: a Tool for Automatic Software Deployment

Easy to install and to use

```
Get from git repository:
$ git clone https://github.com/spack/spack.git

Setup environmental variables:
$ . ./spack/share/spack/setup-env.sh
```

Relatively large community

~350 contributors (~100 a year ago)

Last 200 commits made by 55 contributors

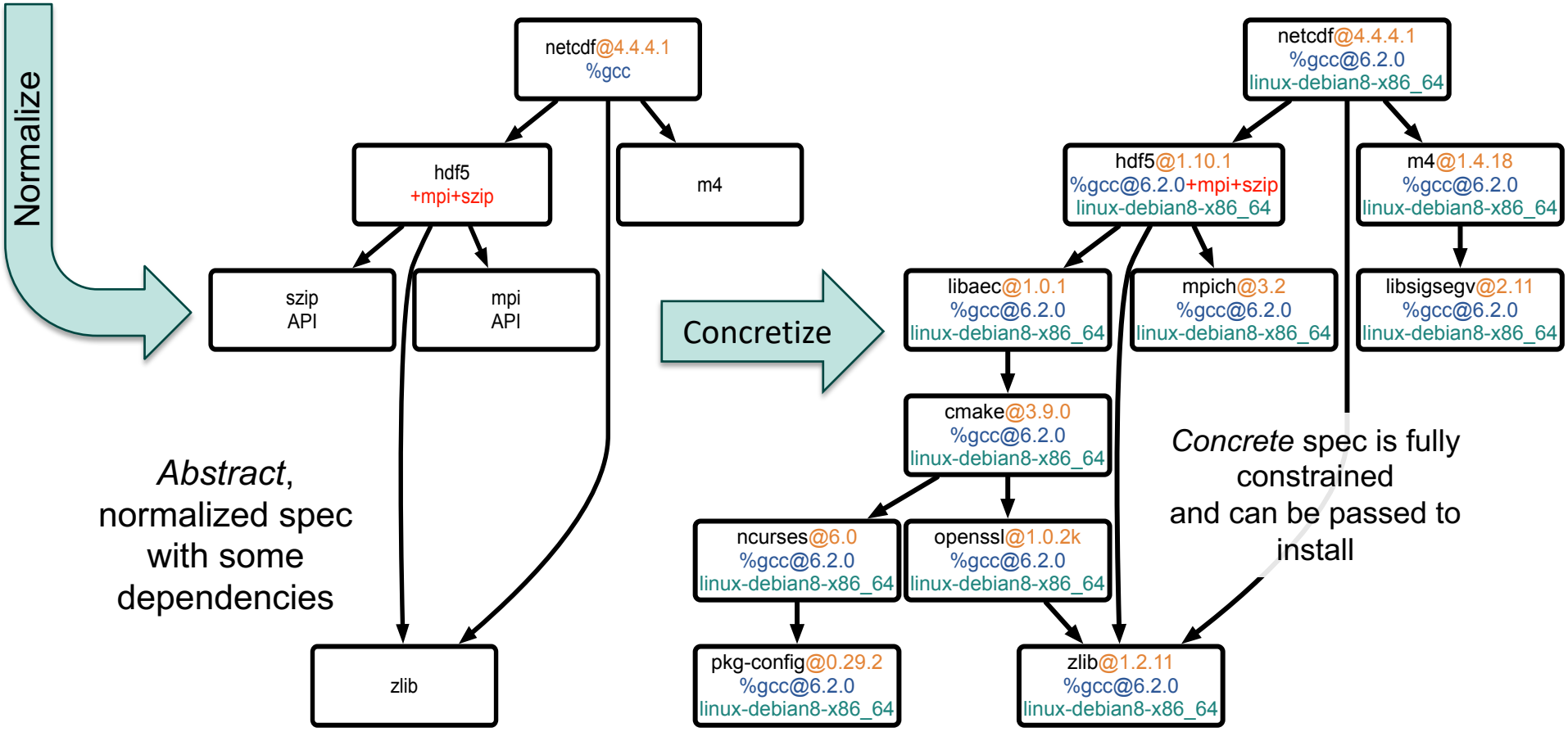
Switching from LGPL-2.1 to MIT/Apache-2.0

```
Common installation
$ spack install cdo

Custom version
$ spack install cdo@1.9.0

Custom compiler
$ spack install cdo@1.9.0 %gcc@6.2.0

Custom build option
$ spack install cdo@1.9.0 %gcc@6.2.0 +grib_api

Cross compilation
$ spack install cdo@1.9.0 %gcc@6.2.0 +grib_api target=haswell
```

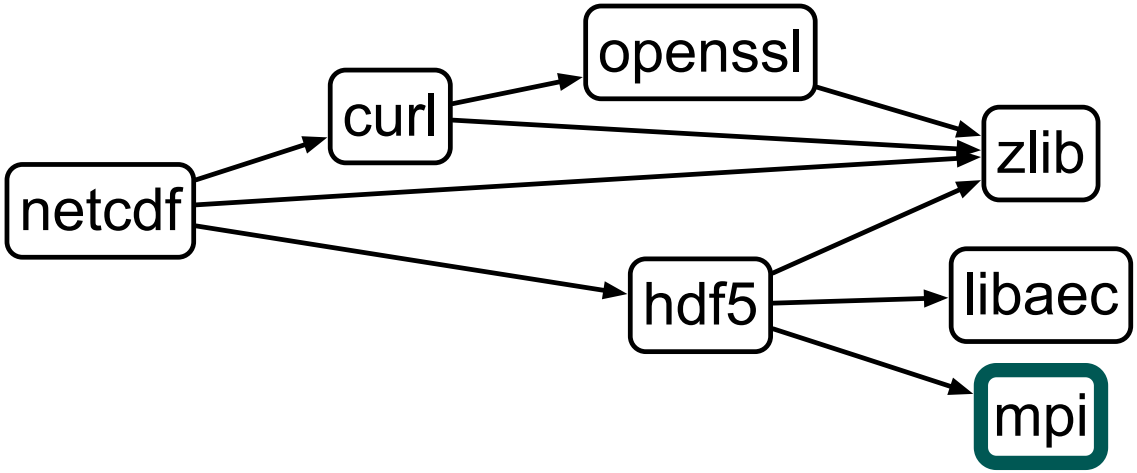
Spack: resolves dependencies

User input: *abstract* spec with some constraints

```
$ spack install netcdf@4: %gcc ^hdf5+mpi+szip
```



Spack: supports API dependencies



Install the same package built with two different MPI implementations

```
$ spack install netcdf ^mvapich@1.9
```

```
$ spack install netcdf ^openmpi@1.4:
```

Let Spack choose MPI version, as long as it provides MPI 2 interface:

```
$ spack install netcdf ^mpi@2
```

Spack: hides complexity

Installation recipe is a Python script

```
from spack import *

class Magics(Package):
    homepage = "https://software.ecmwf.int/wiki/display/MAGP/Magics"
    url = "https://software.ecmwf.int/wiki/download/attachments/3473464/Magics-2.29.0-Source.tar.gz"

    version('2.29.4', '91c561f413316fb665b3bb563f3878d1')
    version('2.29.0', 'db20a4d3c51a2da5657c31ae3de59709', preferred=True)

    patch('no_hardcoded_python.patch')
    patch('resolve_isnan_ambiguity.patch', when='@2.29.0')

    variant('bufr', default=False, description='Enable BUFR support')
    # More variants here...

    depends_on('grib_api')
    # More dependencies here...
    depends_on('libemos', when='+bufr')

    def install(self, spec, prefix):
        options = []
        options.extend(std_cmake_args)
        options.append('-DGRIB_API_PATH=%s' % spec['grib_api'].prefix)

        if '+bufr' in spec:
            options.append('-DENABLE_BUFR=ON')
            options.append('-DLIBEMOS_PATH=%s' % spec['libemos'].prefix)
        else:
            options.append('-DENABLE_BUFR=OFF')
```

Metadata

Versions

Patches

Variants

Dependencies

Commands for installation

Spack: other features

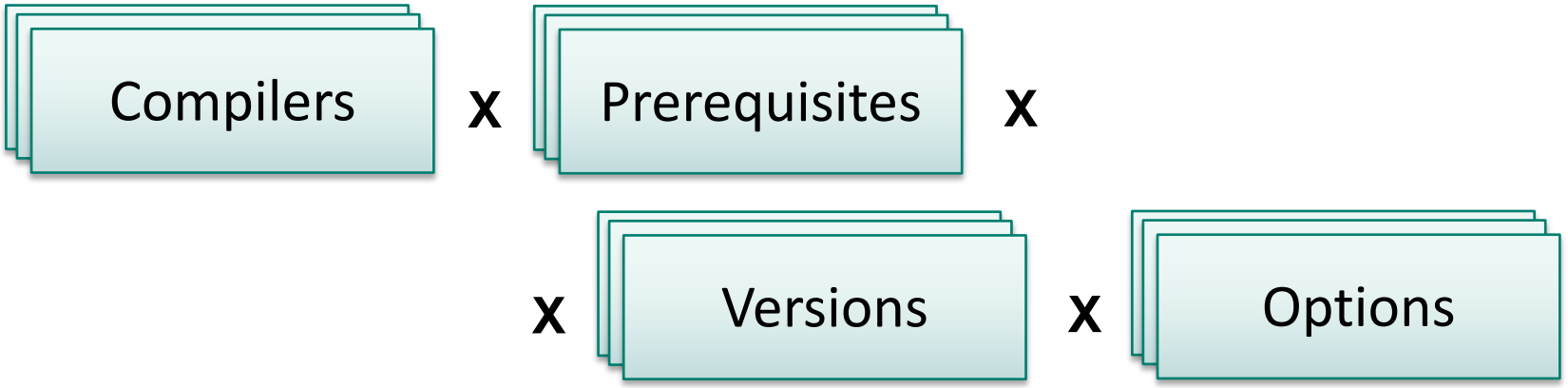
- Installed packages automatically find dependencies
  No need to use modules or set LD_LIBRARY_PATH
- Sanitizes the building environment
  Packages are likely NOT to get unexpected dependencies
- Supports package extensions
  You can configure your own collection of Python packages
- Packages are installed in separate directories
  Multiple configurations of the same package can coexist
- Generates module files
  But does not force to use them
- Stores detailed provenance with the installed package
- Does not require root privileges
- Well documented

Spack: target audience

For users: a way to get the productive software environment for their workflows with minimal lag

For system administrators: a way to understand the requirements of the scientific software and automatization of the maintenance of the software stack with the support of a large (and growing) community

For developers: a way to simplify the maintenance of the code (no need to provide bundled libraries and complicate installation scripts) and to ensure portability of their code: a tool for testing their software with different compilers, libraries, etc.



Commonly used packages

CDO, CMOR, GRIB-API, ecCodes, Emoslib, Magics, NCL, libAEC, Extrae, Paraver, OpenBLAS, LAPACK, NetCDF, HDF5, Python (with modules), NCO, and more…

Tested machines:

- Mistral (DKRZ)
- MareNostrum (BSC)
- Piz Daint (CSCS)
- Marconi (CINECA)
- ARCHER
- XCE (DWD)
- Various UNIX/Linux workstations

Spack: issues

- Spack is under continuous development without explicit stable versions.
- Works out of the box in common UNIX/Linux desktop environments, but often requires additional configuration for clusters, mainly due to their significant customization (wrappers and MPI).

ICON-bootstrap: a project (early stage) within ICON community to address these issues. The project aims to provide Spack configuration files for most commonly used machines (environments).

Requirements to a building system

1. A straightforward way to build software on a new machine. It's preferable to support a standard workflow of a commonly used building system ('./configure && make && make install').
2. No interactive scripts (they still often ask users to specify low-level information like linking flags).
3. Out-of-source build (same code, many configurations at the same time).
4. As much parallelization as much as possible (tricky with Fortran).
5. Support cross-compilation by just setting CC and FC to the cross-compilers.
6. Minimal set of tools, no custom tools that require additional compilation.
7. "In-house" and third-party libraries are treated as separate packages (no bundled libraries).
8. All intermediate files (e.g. *.mod files) and directories must be automatically regenerated when (accidentally) deleted.

Conclusions

- Users will be able to easily customise the software environment on their own, thus being more productive and reducing the workload put on system administrators.
- Spack allows for formal description of the software stack available on a supercomputer, which simplifies identification of the list of missing software dependencies of a modelling workflow.
- Spack can also help system administrators to easily test various usage scenarios of the basic elements of the software environments, such as compiler toolchains and MPI libraries.
- A good way to document the installation procedures and maintain a collection of patches (e.g. for libtool).

Thank you!
Questions?

Our deliverables

- Application Software Framework: A White Paper
- Handbook for System Administrators (and Users)