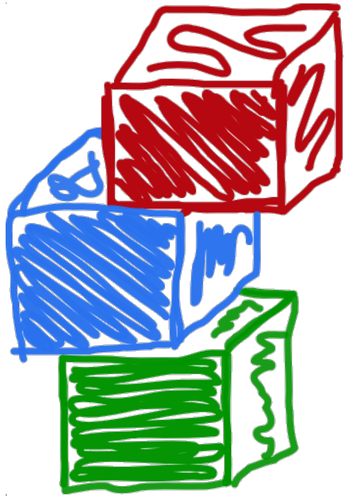
GHENT UNIVERSITY

https://www.ugent.be/hpc

Vlaanderen is computing

https://www.vscentrum.be



easybuild

Kenneth Hoste (HPC-UGent)

kenneth.hoste@ugent.be

LUMI - Dec 17th 2020

Installing (scientific) software on HPC systems

- key aspects:
  - large-scale **multi-user** system (100s to 1000s of active users)
  - **wide variety of users**, scientific domains, use cases, ...
  - results in **huge software collection** to (centrally) maintain & update
  - **different hardware architectures** (old & new processor generations)
  - important **system libraries** (IB network, GPU, ...) should also be taken into account
  - **conflicting user requirements/expectations** make things even more complex
    - stable software versions vs bleeding edge, Python 2 vs 3, ...
  - performance is key, so software preferably **built from source !**

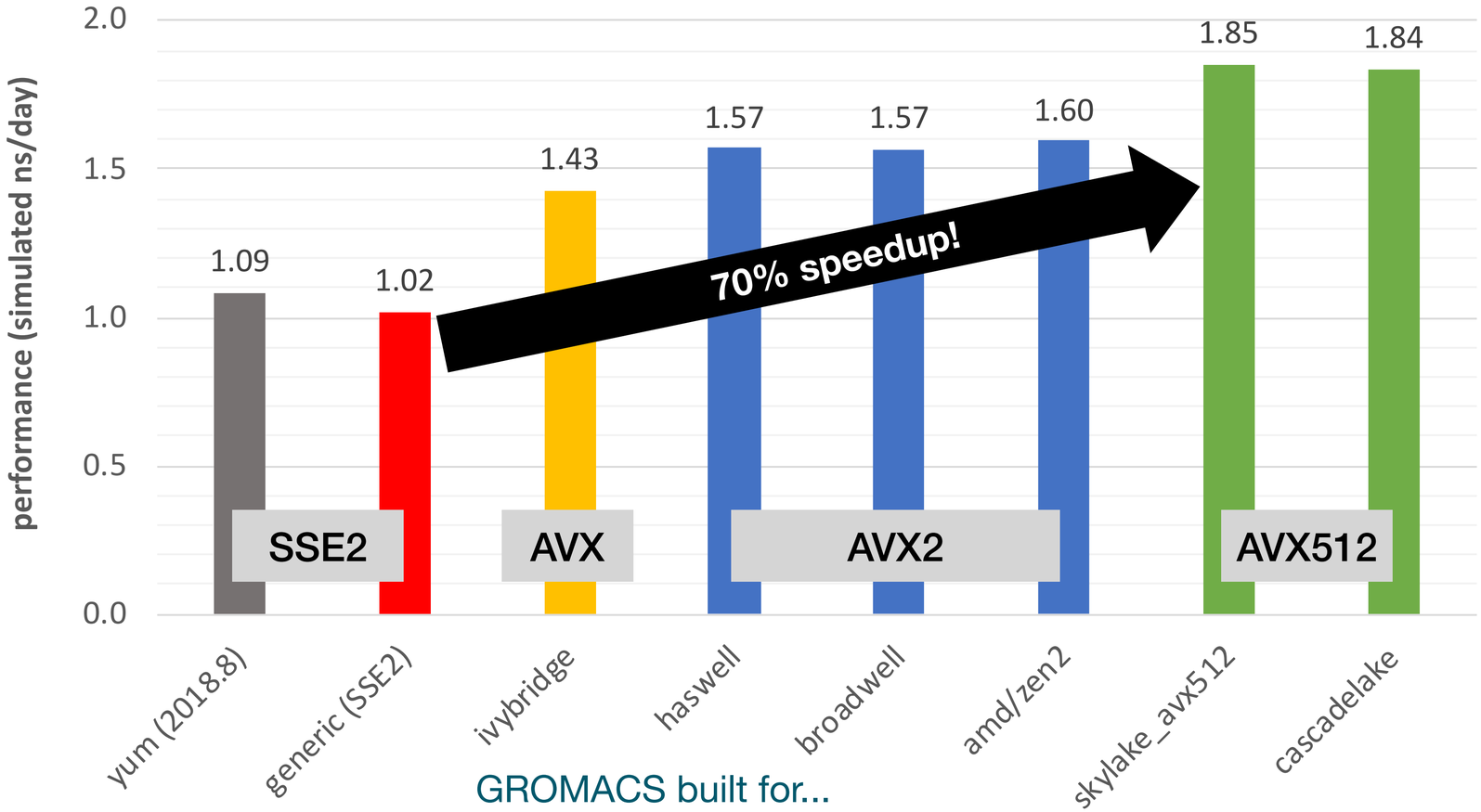
- **user-friendly interface should be provided to users** to leverage installed software
  - well-established solution is the "environment modules" tool (Lmod is quite popular now)
  - `module avail example`, `module load example/1.2.3`, `module list`, ...

Keeping the P in HPC

(HPC: High-Performance Computing)

- **Software should be optimised for the system it will run on**
- Impact on performance is often significant for scientific software

- Example: GROMACS 2020.1 (PRACE benchmark, Test Case B)
- Metric: (simulated) ns/day, higher is better
- Test system: dual-socket Intel Xeon Gold 6420 (Cascade Lake, 2x18 cores)



easybuild in a nutshell

https://easybuilders.github.io/easybuild - https://easybuild.readthedocs.io

- **framework for installing scientific software** *(built from source when possible)*
- strong focus on Linux & HPC systems (and hence also performance)
- **default is to build/optimise specifically for host architecture**
- implemented in Python (2.6+ or 3.5+), lead development by HPC-UGent
- available under GPLv2 license via PyPI, GitHub
- supports different compilers & MPI libraries, x86_64/ARM/POWER, ...
- currently supports 1,798 different software packages (+ extensions)
- welcoming and helpful worldwide community

terminology

https://easybuild.readthedocs.io/en/latest/Concepts_and_Terminology.html

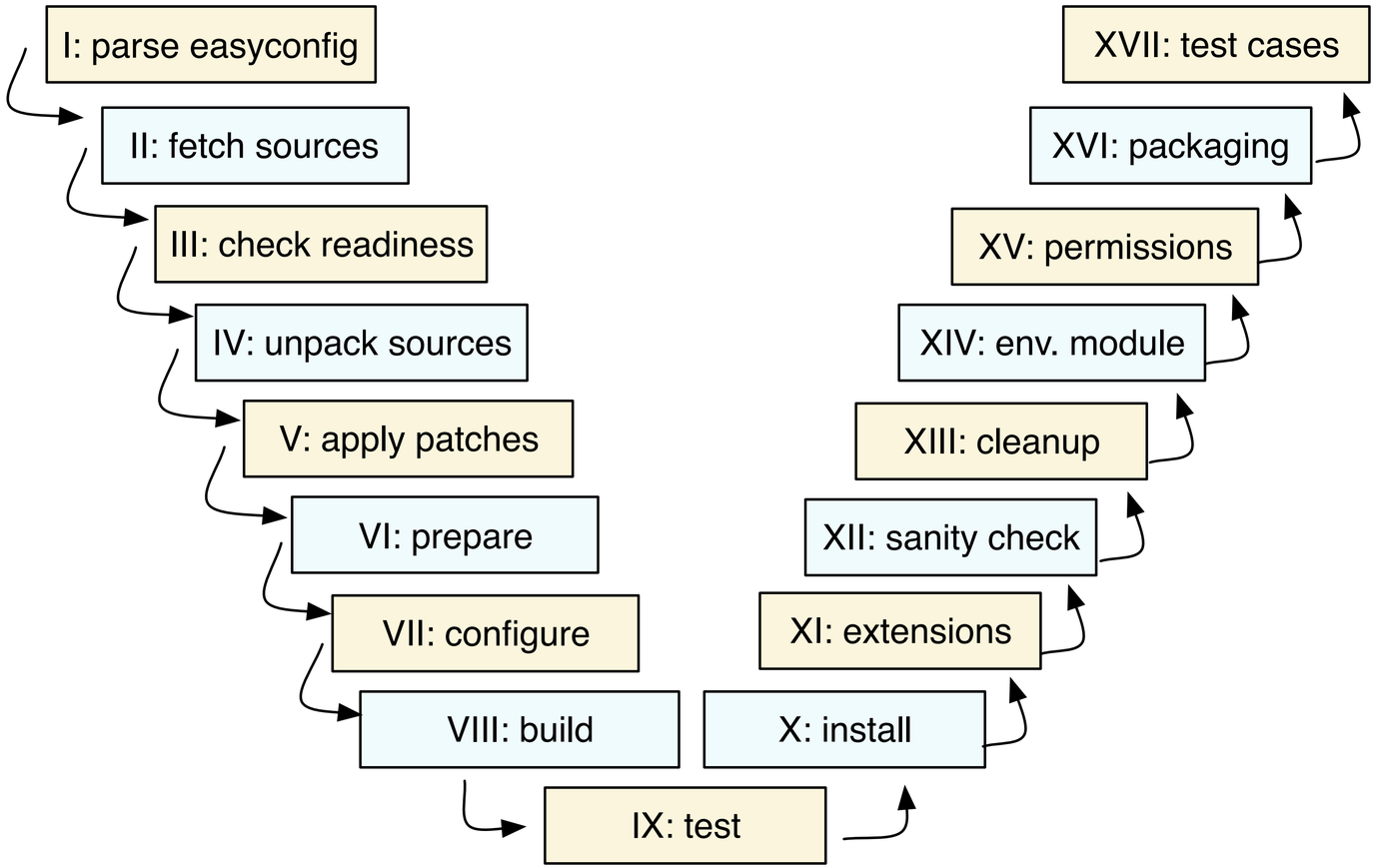
- **framework**
  - core of EasyBuild: Python modules & packages
  - provides supporting functionality for building/installing software, generating modules, ...
- **easyblock**
  - a Python module that serves as a build script, 'plugin' for the EasyBuild framework
  - implements a (generic or software-specific) build/install procedure
- **easyconfig file** (*.eb): build specification; software name/version, compiler toolchain, etc.
- **(compiler) toolchain***:* set of compilers + accompanying libraries (MPI, BLAS/LAPACK, ...)
- **extensions***:* additional packages for a particular applications (e.g., Python, R)

Installing TensorFlow from source with **one command**...


```
$ eb TensorFlow-2.1.0-fosscuda-2019b-Python-3.7.4.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig TensorFlow-2.1.0-fosscuda-2019b-Python-3.7.4.eb
== building and installing TensorFlow/2.1.0-fosscuda-2019b-Python-3.7.4...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== permissions...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /software/TensorFlow/2.1.0-...
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-GyvPHx/easybuild-U1TkEI.log* have been removed.
== Temporary directory /tmp/eb-GyvPHx has been removed.
```

Step-wise installation procedure

EasyBuild performs a step-wise installation procedure for each software:



- download sources (best effort)
- set up build directory & environment
  - unpack sources (& apply patches)
  - load modules for toolchain & deps
  - define toolchain-related env vars (`$CC`, `$CFLAGS`, ...)
- configure, build, (test), install, (extensions)
- perform simple sanity check on installation
- generate environment module file

each step can be customised via easyconfig parameters or an easyblock

feature highlights (1)

- fully **autonomously** building and installing (scientific) software
  - automatic dependency resolution (via `--robot`)
  - automatic generation of environment module files (Tcl or Lua syntax)
- **no admin privileges required** (only write permission to installation target)
- thorough **logging** of executed build/install procedure
- **archiving** of easyconfigs, patches, easyblocks that were used
- highly **configurable**, via config files/environment/command line
- **dynamically extendable** with additional easyblocks, toolchains, etc.

feature highlights (2)

- support for **custom module naming schemes** (incl. hierarchical)
- **transparency** via support for 'dry run' installation & trace output
- **comprehensively tested**: lots of unit tests, frequent regression testing, ...
- actively developed, **frequent stable releases** (every 6-8 weeks)
- **collaboration** between various HPC sites large & small
- integration with Torque/SLURM, FPM, Singularity, Docker, Cray PE, ...
- worldwide **community**

Supported software

https://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html

- latest EasyBuild (v4.3.2) supports installing **2,176 different software packages**
  - including CP2K, LAMMPS, NAMD, NWChem, OpenFOAM, TensorFlow, WRF, ...
  - a lot of bioinformatics software is also supported out of the box
  - + >1,500 extensions: Python packages, R libraries, Perl modules, X11 libraries, ...
  - built from source when possible, optimised by host architecture by default
- diverse toolchain support:
  - compilers: GCC, Intel, Clang, PGI, IBM XL, **Cray PE (GNU, Intel, CCE, PGI)**, CUDA
  - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, Cray MPI, ...
  - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ScaLAPACK, BLIS, Cray LibSci, ...

Supported software

(counts are without extensions)

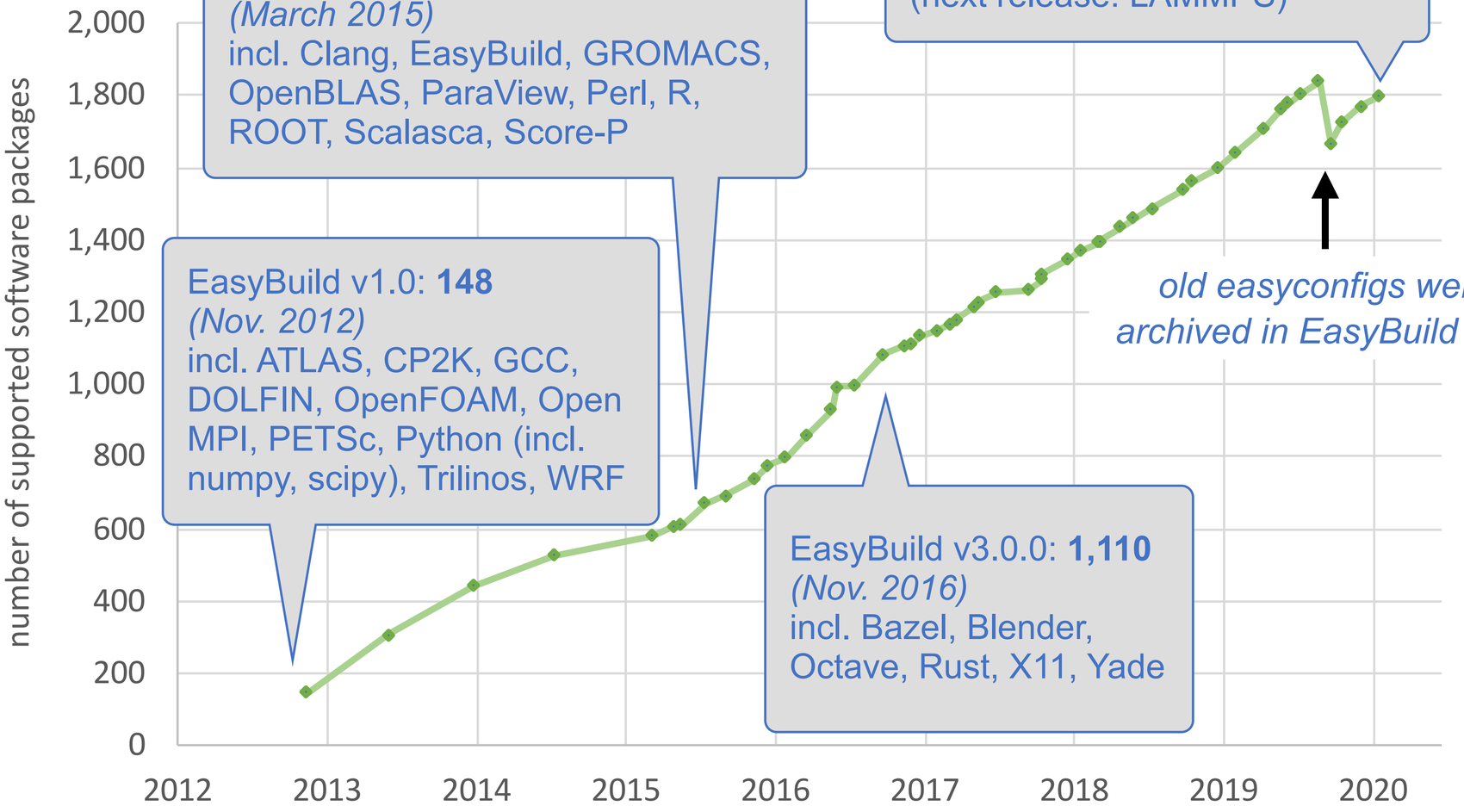
EasyBuild v4.1.2: 1,798
(Jan. 2020)
incl. TensorFlow, PyTorch, ...
(next release: LAMMPS)

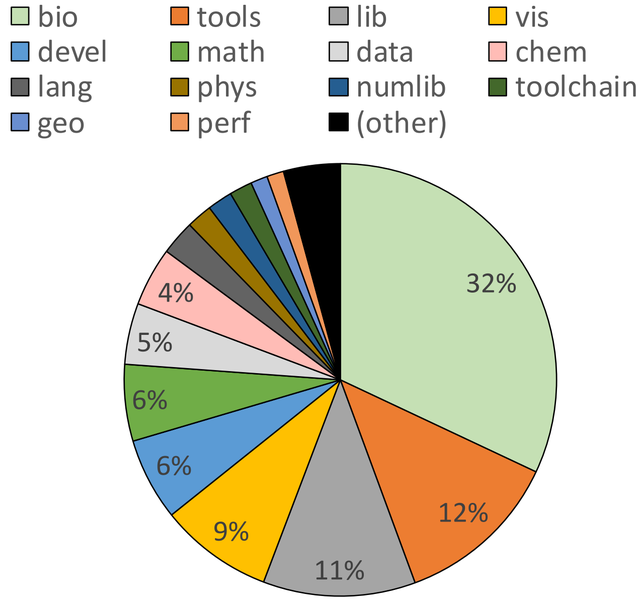
EasyBuild v2.0.0: **581**
(March 2015)
incl. Clang, EasyBuild, GROMACS, OpenBLAS, ParaView, Perl, R, ROOT, Scalasca, Score-P

EasyBuild v1.0: **148**
(Nov. 2012)
incl. ATLAS, CP2K, GCC, DOLFIN, OpenFOAM, Open MPI, PETSc, Python (incl. numpy, scipy), Trilinos, WRF

EasyBuild v3.0.0: **1,110**
(Nov. 2016)
incl. Bazel, Blender, Octave, Rust, X11, Yade

old easyconfigs were archived in EasyBuild v4.0





- 1/3rd of supported software is bioinformatics
- 23% is (generic) libraries & tools
- > 2000 supported software in 2020?

What  easybuild is *not*

- EasyBuild is ***not*** **YABT** (Yet Another Build Tool)

  it does *not* replace build tools like `cmake` or `make`; it wraps around them

- it is ***not*** **a replacement for package managers** (`yum`, `apt`, ...)

  it leverages some tools & libraries provided by the OS (glibc, OpenSSL, IB drivers, ...)

- it is ***not*** **a magic solution** to all your (software installation) problems...

  you will still run into compiler errors (unless somebody has already taken care of it)

What  easybuild is

- a **uniform interface** that wraps around software installation procedures
- a huge **time-saver**, by automating tedious/boring/repetitive tasks
- a way to provide a **consistent software stack** to your users
- an **expert system** for software installation on HPC systems
- a **platform for collaboration** with HPC sites worldwide
- a way to **empower *users* to self-manage their software stack** on HPC systems
- a tool that can be leveraged for **building *optimised* container images**

Getting started: installing a compiler toolchain

https://easybuild.readthedocs.io/en/latest/Common-toolchains.html

- most easyconfig files use a particular compiler (sub)toolchain
- commonly used toolchains are:
  - `foss`: GCC, OpenMPI, OpenBLAS + ScaLAPACK, FFTW
  - `intel`: Intel C/C++/Fortran compilers, Intel MPI, Intel Math Kernal Library (MKL)
- compiler toolchain must be installed first (for example via "`eb --robot`")
- this may take a while, depending on available system resources...
- system compilers & MPI/BLAS/LAPACK libraries are usually not used
- tools provided via EasyBuild are strongly preferred (makes reproducing installations easier)

Community (common) toolchains

http://easybuild.readthedocs.io/en/latest/Common-toolchains.html

- **`intel` and `foss`* toolchains** are most commonly used in EasyBuild community

  (*) FOSS: Free and Open Source Software

- helps to focus efforts of HPC sites using one or both of these toolchains
- updated twice a year, clear versioning scheme: `<year>{a,b}` (`2018b`, `2019a`, ...)
- latest version:
  - `foss/2020b`

    binutils 2.35, GCC 10.2.0, OpenMPI 4.0.5, OpenBLAS 0.3.12, FFTW 3.3.8
  - `intel/2020b`

    binutils 2.32 + GCC 8.3.0 as base

    Intel compilers 2020.4.304, Intel MPI 2019.9.304, Intel MKL 2020.4.304

Easyconfig files as build specifications

https://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html

- for each software installation, there is a corresponding easyconfig file
- **simple text files defining a set of easyconfig parameters** (in Python syntax)
- some are mandatory: software name/version, toolchain, metadata (homepage, descr.)
- other commonly used parameters:
  - easyblock to use
  - list of sources & patches
  - list of (build) dependencies
  - options for configure/build/install commands
  - files/directories that should be present
  - trivial commands that should work (sanity check)

Example easyconfig file

no easyblock specified, which implies using
a software-specific easyblock (`EB_WRF`)

software name and version

build variant (specific to WRF)
('dmpar': distributed, MPI)

software metadata

toolchain name & version

sources & patches

list of (build) dependencies
note: all versions are *fixed*!

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Adding support for additional software

- for each software installation, an **easyconfig file** is *required*
  - see ***https://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html***
  - existing easyconfig files can serve as examples
  - tweaked easyconfig files can be *generated* via `eb --try-*`
- for 'standard' installation procedures, a **generic easyblock** can be used
  - installation can be controlled where needed via easyconfig parameters
- for custom installation procedures, a **software-specific easyblock** is required
  - see ***https://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html***

Common generic easyblocks

http://easybuild.readthedocs.io/en/latest/version-specific/generic_easyblocks.html

- **`ConfigureMake`**
  standard '`./configure`' - '`make`' - '`make install`' installation procedure
- **`CMakeMake`**
  same as `ConfigureMake`, but using '`cmake`' for configuring
- **`PythonPackage`**
  installing an individual Python package ('`python setup.py install`', '`pip install`', ...)
- **`PythonBundle`**
  installing a bundle of Python packages
- **`MakeCp`**
  no (standard) configuration step, build with '`make`', install by copying binaries/libraries
- **`Tarball`**: just unpack sources and copy everything to installation directory
- **`Binary`**: run binary installer (specified via '`install_cmd`' easyconfig parameter)

Easyconfig files vs easyblocks

http://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html

- thin line between using custom easyblock and 'fat' easyconfig with generic easyblock
- custom easyblocks are "do once and forget", **central solution to build peculiarities**
- reasons to consider implementing a software-specific easyblock include:
  - 'critical' values for easyconfig parameters required to make installation succeed
  - toolchain-specific aspects of the build and installation procedure (e.g., configure options)
  - interactive commands that need to be run
  - custom (configure) options for dependencies
  - having to create or adjust specific (configuration) files
  - 'hackish' usage of a generic easyblock

Using a custom module naming scheme

- a couple of different module naming schemes are included in EasyBuild
  - see `--avail-module-naming-schemes`
  - specify active module naming scheme via `--module-naming-scheme`
  - default: `EasyBuildMNS` (`<name>/<version>-<toolchain>-<versionsuffix>`)
- **you can implement your own module naming scheme relatively easily**
  - specify how to compose module name using provided metadata
  - via Python module that defines custom derivative class of `ModuleNamingScheme`
  - make EasyBuild aware of it via `--include-module-naming-schemes`
- decouple naming of install dirs vs modules via `--fixed-installdir-naming-scheme`

Flat module naming scheme

- all modules are always available for loading
- long(er) module names
- '`module avail`' may be overwhelming for users
- too easy to load incompatible modules together

legend

(not available)

(available)

(loaded)

GCC/5.3.0

GCC/6.1.0

OpenMPI/1.10.2-GCC-5.3.0

OpenMPI/2.1.0-GCC-5.3.0

OpenMPI/1.10.3-GCC-6.1.0

OpenMPI/2.1.0-GCC-6.1.0

FFTW/3.3.4-gompi-2016.04

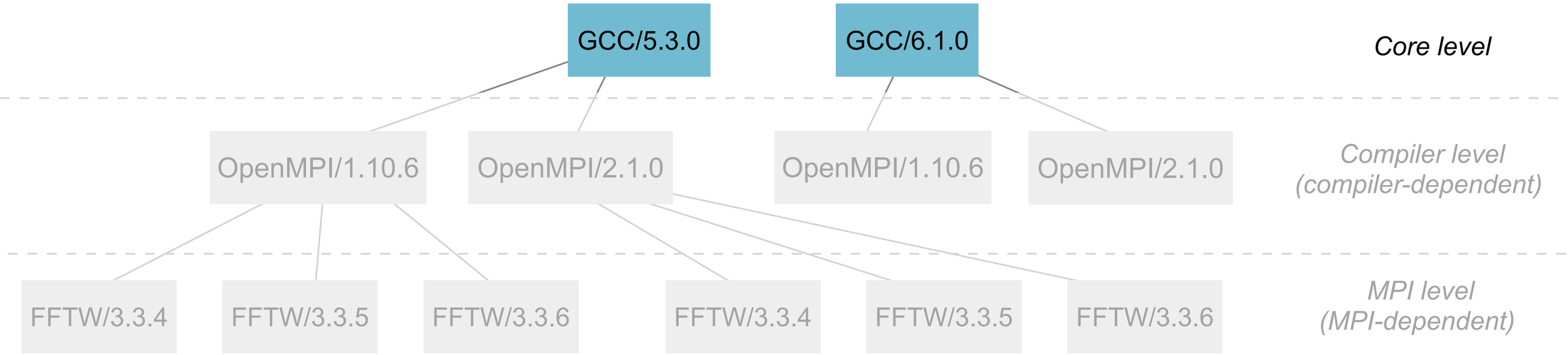
FFTW/3.3.6-gompi-2016.04

FFTW/3.3.4-gompi-2016.07

FFTW/3.3.6-gompi-2016.07

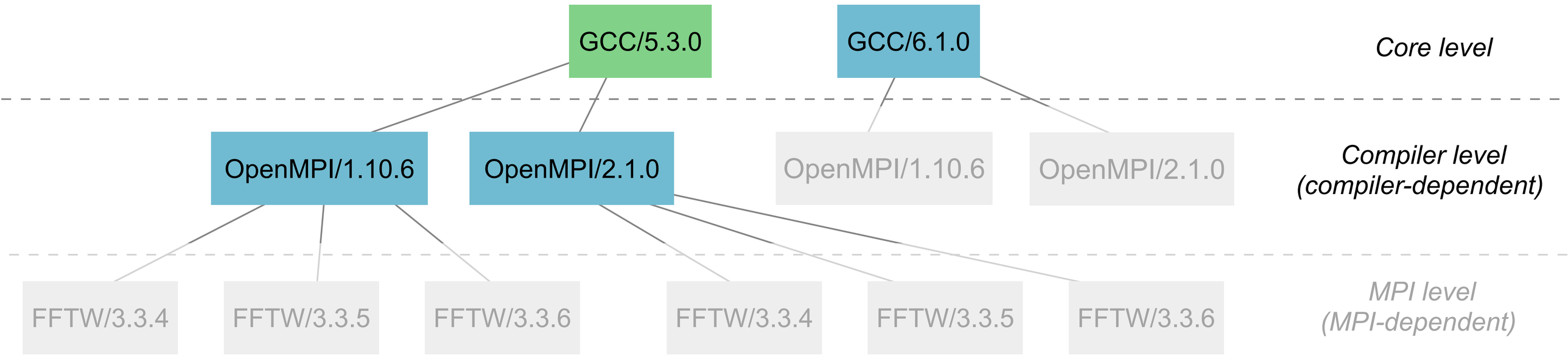
Hierarchical module naming scheme (1)

- modules are organised in a tree-like fashion
- initially, only 'core' modules are available for loading
- typically 3 hierarchy levels: core, compiler-dependent, MPI-dependent
- other modules are only visible via '`module spider`'



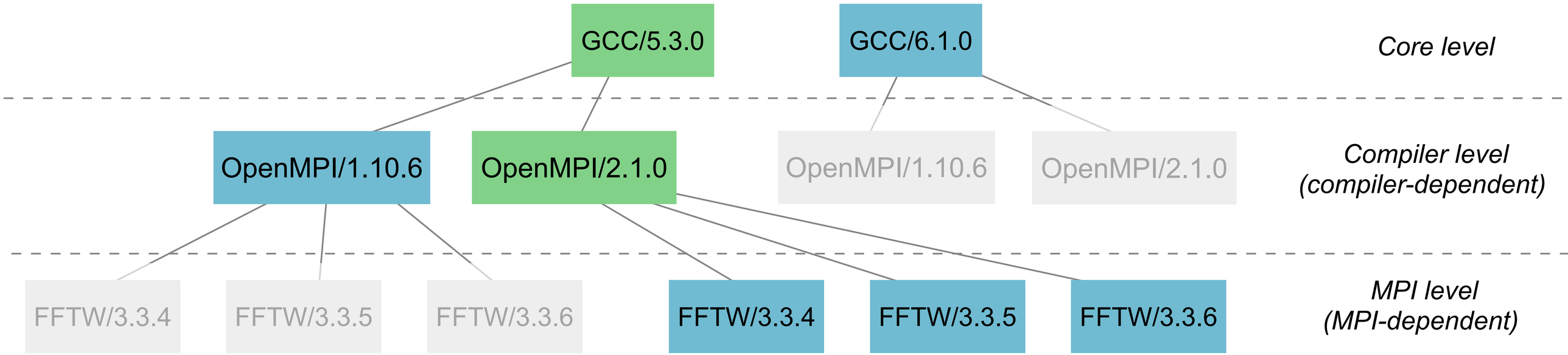
Hierarchical module naming scheme (2)

- Core modules may extend `$MODULEPATH` with an additional location
- loading a Core module may make more modules available
- in this example, loading a GCC module makes OpenMPI modules available



Hierarchical module naming scheme (3)

- even more modules may be made available by loading other modules
- for example, loading an OpenMPI modules reveals MPI-dependent modules
- EasyBuild can organise modules in hierarchy for you!



The sun never sets on EasyBuild...

cities from where https://easybuild.readthedocs.io was visited at least 10 times during the last year

(source: Google Analytics)

Hands-on tutorial

https://easybuilders.github.io/easybuild-tutorial

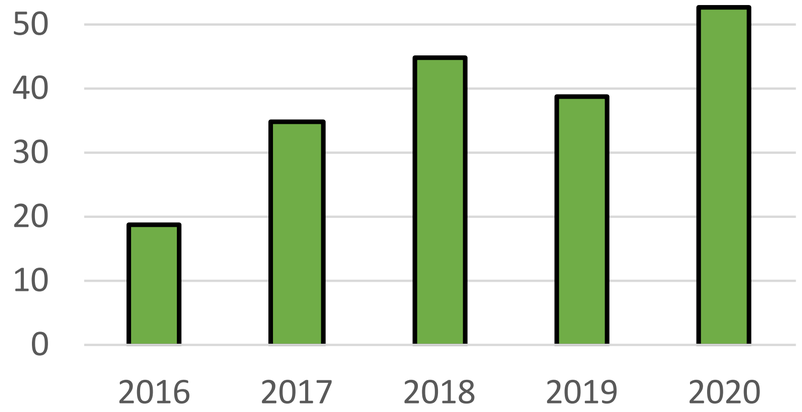
- Introductory tutorial to EasyBuild:
  installation, configuration, basic usage, troubleshooting, etc.
- Including hands-on exercises (+ solutions)
- Container images available with small software stack to get started easily
- ~100 attendees to virtual tutorial in June 2020 (recording available)
- **Accepted tutorial for ISC 2021!**

EasyBuild User Meetings

2018 (SURFsara)


2019 (UCLouvain)


2020 (HPCNow! - Barcelona)



Attendance is growing over the years:

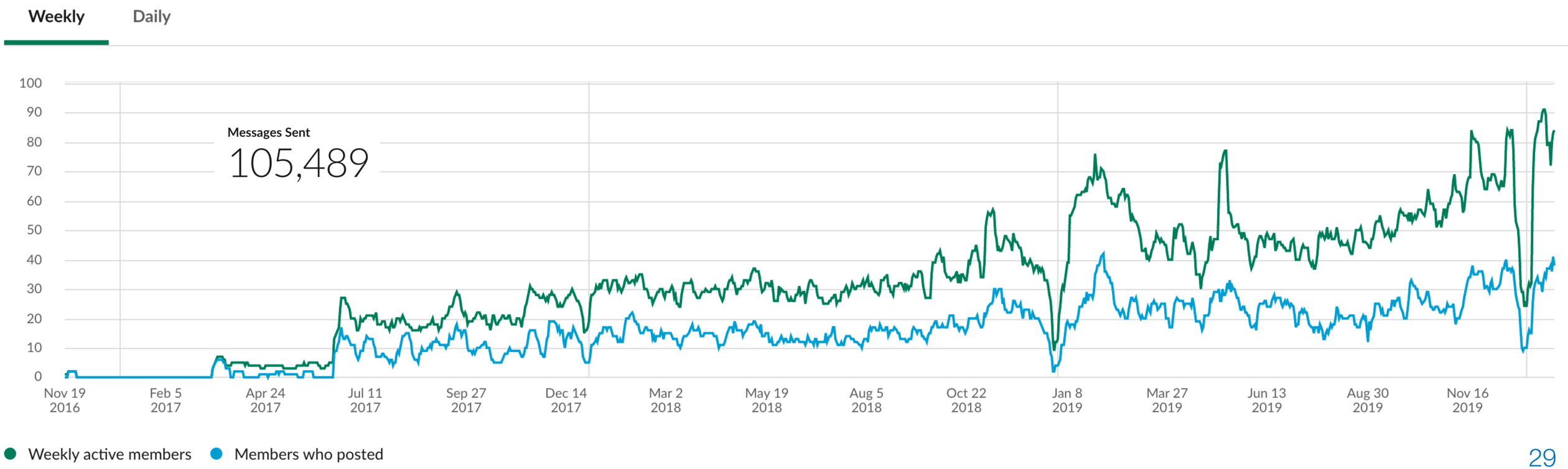
- *2016: 19 (HPC-UGent, Belgium)*
- *2017: 35 (JSC, Germany)*
- *2018: 45 (SurfSARA, Netherlands)*
- *2019: 39 (CECI, Belgium)*
- *2020: 53 (HPCNow!, Spain)*

easybuild slack channel (+ IRC)

- over 220 members on Slack channel (+70 in 2019!)
- steady growth in activity since start of EasyBuild Slack in 2017

(self-invite via *https://easybuild-slack.herokuapp.com*)

- also: EasyBuild mailing list
- 270+ subscribers
- ~700 emails/year



17 easybuild maintainers

Damian Alvarez - `@damianam`
(Jülich Supercomputing Centre)

Miguel Dias Costa - `@migueldiascosta`
(National University of Singapore)

Alex Domingo - `@lexming`
(Free University of Brussels)

Pablo Escobar - `@pescobar`
(sciCORE, University of Basel)

Fotis Georgatos - @fgeorgatos
(SDSC - Swiss Data Science Center)

Kenneth Hoste - `@boegel`
(HPC-UGent)

Adam Huffman - `@verdurin`
(Big Data Institute, University of Oxford)

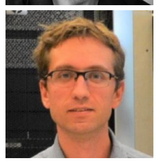
Samuel Moors - `@smoors`
(Free University of Brussels)

Simon Branford - @branfosj
(University of Birmingham)

Alan O'Cais - `@ocaisa`
(Jülich Supercomputing Centre)

Mikael Öhman - `@micketeer`
(Chalmers University of Technology)

Bart Oldeman - `@bartoldeman`
(ComputeCanada)



Ward Poelmans - `@wpoely86`
(Free University of Brussels)

Åke Sandgren - `@akesandgren`
(Umeå University, Sweden)

Caspar Van Leeuwen - `@casparvl`
(SURFsara, Netherlands)

Davide Vanzo - `@vanzod`
(Microsoft HPC)

Lars Viklund - `@zao`
(Umeå University, Sweden)

vs

- EasyBuild: GPLv2 license; Spack: MIT/Apache 2.0 license
- **no stable releases yet for Spack (< 1.0), EasyBuild is stable since 2012**
- ~ on par w.r.t. amount of supported software (but differences w.r.t. which software)
- targeted to different use cases: HPC support teams (EasyBuild) vs developers (Spack)
- fixed dependency/toolchain versions in EasyBuild vs flexible CLI in Spack
- both support running on top of Python 2.6, 2.7, 3.5+ (since EasyBuild v4.0)
- macOS support in EasyBuild is limited (no toolchains/testing for macOS)
- both projects are backed by an active & supportive community!

For a more detailed comparison,
see https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists