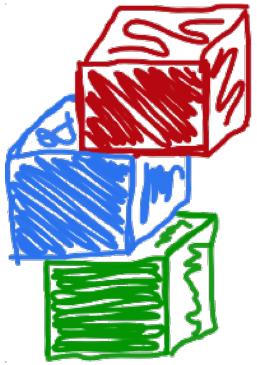


easybuild

Getting Scientific Software Installed with EasyBuild

June 12th 2017 - Francis Crick Institute

remote presentation

http://users.ugent.be/~kehoste/EasyBuild-intro_20170612_FrancisCrick.pdf

kenneth.hoste@ugent.be

http://hpcugent.github.io/easybuild/

GHENT UNIVERSITY

http://www.ugent.be/hpc

https://www.vscentrum.be

Vlaams Supercomputer Centrum

HPC-UGent

Vlaams Supercomputer Centrum

GHENT UNIVERSITY

- part of central IT department of Ghent University (Belgium)
- centralised scientific computing services, training & support
- for researchers of UGent, industry & knowledge institutes
- member of Flemish Supercomputer Centre (VSC)
  https://www.vscentrum.be/
- core values:
  empowerment - centralisation - automation - collaboration

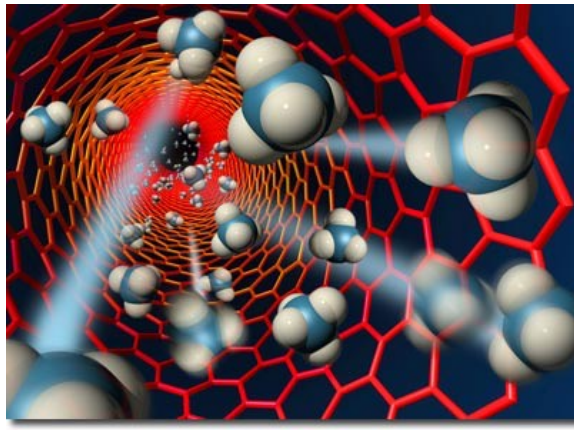
whoami

- Masters & PhD in Computer Science from Ghent University (Belgium)
- joined HPC-UGent team in October 2010
- main tasks: user support & training, software installations
- inherited maintenance of EasyBuild in 2011
- slowly also became lead developer & release manager
- big fan of loud music & FOSS (Free & Open Source Software)

kenneth.hoste@ugent.be - *boegel* (GitHub, IRC) - @kehoste (Twitter)

Installing scientific software for users in HPC systems

- by user request: new software, version updates, more variants, . . .
- usually on a (shared NFS) filesystem available on every workernode
- specifically targeted to the HPC cluster it will be used on
  - built from source (if possible)
  - separate installation per (type of) cluster
  - highly optimised for system architecture
- rebuild when updates for compilers/libraries become available
- installations typically remain available during lifetime of system



"Please install <software> on the HPC?"

The most common type of support request from users is to install (scientific) software.

- over 25% of support tickets at HPC-UGent
- consumes (way) more than 25% of time of HPC-UGent support team

Installing (lots of) scientific software is:

- error-prone, trial-and-error
- tedious, hard to get right
- repetitive & boring (well...)
- time-consuming (hours, days, even weeks)
- frustrating (e.g., dependency hell)
- sometimes simply not worth the effort...

Common issues with scientific software

Researchers focus on the *science* behind the software they implement,
and care little about tools, build procedure, portability, ...

Scientists are not software developers or sysadmins (nor should they be).

"If we would know what we are doing, it would not be called 'research'."

This results in:

- use of non-standard build tools (or broken ones)
- incomplete build procedure, e.g., no configure or install step
- interactive installation scripts
- hardcoded parameters (compilers, libraries, paths, . . .)
- poor/outdated/missing/incorrect documentation

Prime example: WRF

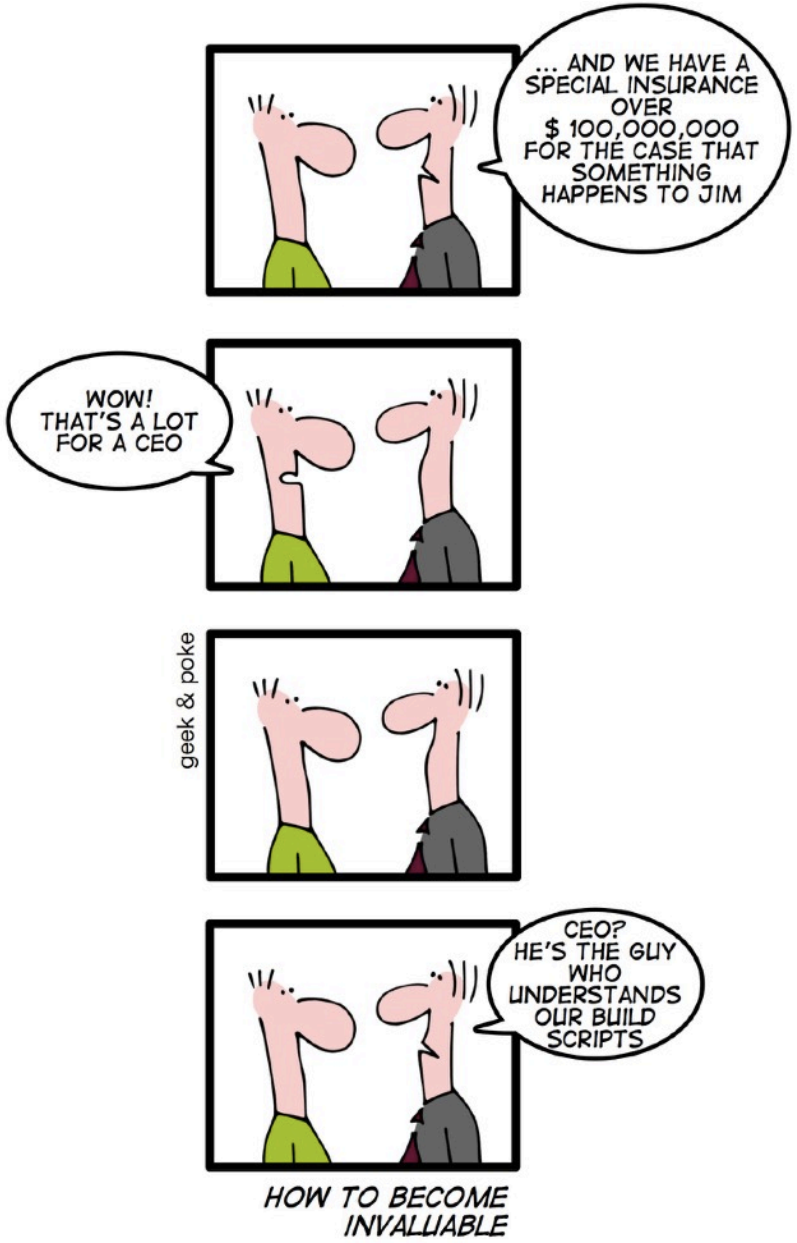
Weather Research and Forecasting Model (http://www.wrf-model.org)

- dozen dependencies: netCDF (C, Fortran), HDF5, tcsh, JasPer, ...
- known issues in last release are (only) documented on website
- no patch file provided, infrequent bugfix releases
- interactive '`configure`' script, options change between versions
- resulting configuration file '`configure.wrf`' needs work:
  fix hardcoded settings (compilers, libraries, . . .), tweaking of options
- custom ‘compile’ script (wraps around ‘make’)
  building in parallel is broken without fixing the Makefile
- no actual installation step

Houston, we have a problem...

Installation of scientific software is a tremendous problem for HPC sites all around the world.

- huge burden on HPC user support teams
- researchers lose lots of time (waiting)
- sites typically resort to crude in-house scripting
- very little collaboration among HPC sites



http://geekandpoke.typepad.com/geekandpoke/2010/05/how-to-become-invaluable.html

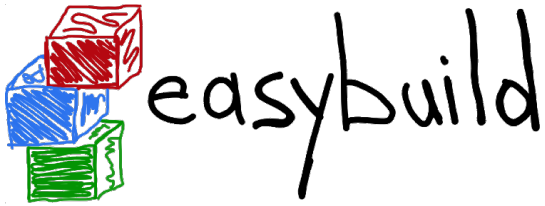
What about existing software installation tools?

Existing tools are not well suited to scientific software and HPC systems.

- package managers: yum (RPMs), apt-get (.deb), ...
- Homebrew (Mac OS X), http://brew.sh/ ; Linuxbrew, http://brew.sh/linuxbrew/
- Portage (Gentoo), http://wiki.gentoo.org/wiki/Project:Portage
- pkgsrc (NetBSD & (a lot) more), http://pkgsrc.org/
- Nix, http://nixos.org/nix ; GNU Guix, https://www.gnu.org/s/guix

Common problems:

- usually poor support for old/multiple versions and/or to have builds side-by-side
- not flexible enough to deal with idiosyncrasies of scientific software
- little support for scientific software, non-GCC compilers, MPI



- website: http://hpcugent.github.io/easybuild/
- documentation: http://easybuild.readthedocs.io
- **framework to build & install scientific software on HPC clusters**
- implemented in Python 2, lead development by HPC-UGent
- supports different compilers & MPI libraries, **> 1,200 different software packages**, ...
- **active & helpful worldwide community**

5 years of EasyBuild

- (in-house development at HPC-UGent since 2009)
- **first public release in April 2012 (EasyBuild v0.5)**
  - initial intention was to get feedback, but a community emerged rather quickly...
- **first stable release in November 2012 (EasyBuild v1.0)**
- frequent stable releases since then (both feature and bugfix)
- latest: EasyBuild v3.2.1 (May 12th 2017)
- development is highly community-driven (bug reports, feature requests, contributions)

Feature highlights (1)

- fully **autonomously** building and installing (scientific) software
  - automatic dependency resolution
  - automatic generation of module files (Tcl or Lua syntax)
- thorough **logging** of executed build/install procedure
- **archiving** of build specifications
- 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
- **comprehensively tested**: lots of unit tests, regression testing, ...
- actively developed, frequent stable releases
- **collaboration** between various HPC sites
- worldwide **community**

What EasyBuild is (not)

EasyBuild is:

- ***not*** YABT (Yet Another Build Tool); it does not replace tools like `make` (but wraps around them)
- ***not*** a replacement for your favourite package manager (`yum`, `apt-get`, ...)
- ***not*** a magic solution to all your (software installation) problems...
- a **uniform interface** that wraps around software build 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 world-wide
- a way to **empower users to self-manage their software stack** on HPC systems

Supported software

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

- EasyBuild v3.2.1 supports installing **over 1,200 software tools & applications**
  - including CP2K, NAMD, NWChem, OpenFOAM, QuantumESPRESSO, WRF, WPS, ...
  - also a lot of bioinformatics software is supported out of the box...
  - in addition: hundreds of R libraries, Python packages, Perl modules, ...
- diverse toolchain support:
  - compilers: GCC, Intel, Clang, PGI, IBM XL, Cray
  - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, ...
  - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ATLAS, ACML, Cray LibSci, ...

Terminology

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

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

In numbers

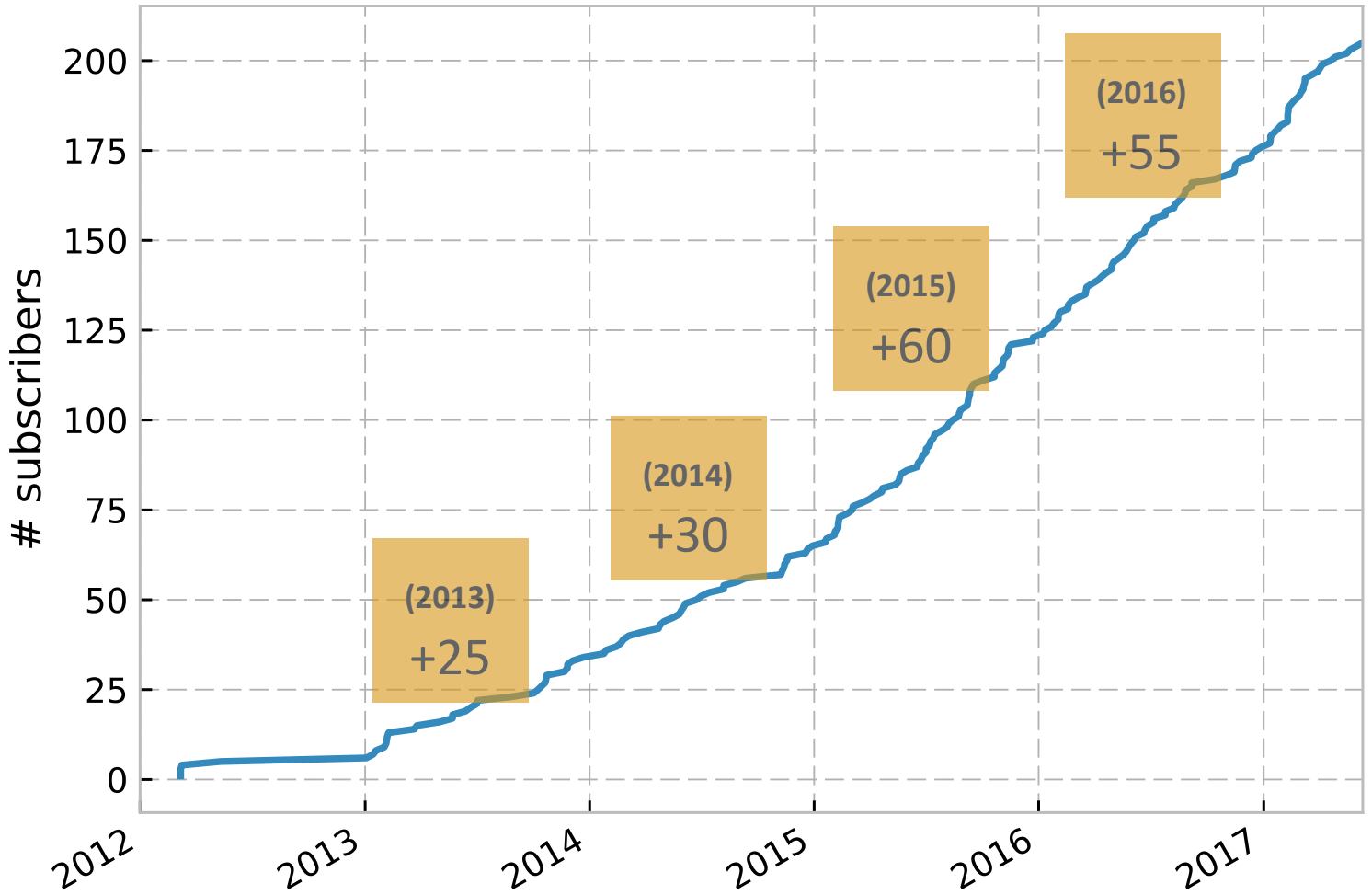
included in EasyBuild v3.2.1:

- about 50k LoC of Python (incl. framework, easyblocks, and logging/option parsing support)
- about 15k LoC more for unit/integration test suites
- 1,228 supported software applications (excl. extensions, software versions)
- 25 different toolchain definitions (excl. toolchain versions)
- 185 software-specific easyblocks, 30 generic easyblocks
- 7,036 easyconfig files
  - different software versions, variants, using different toolchains, ...

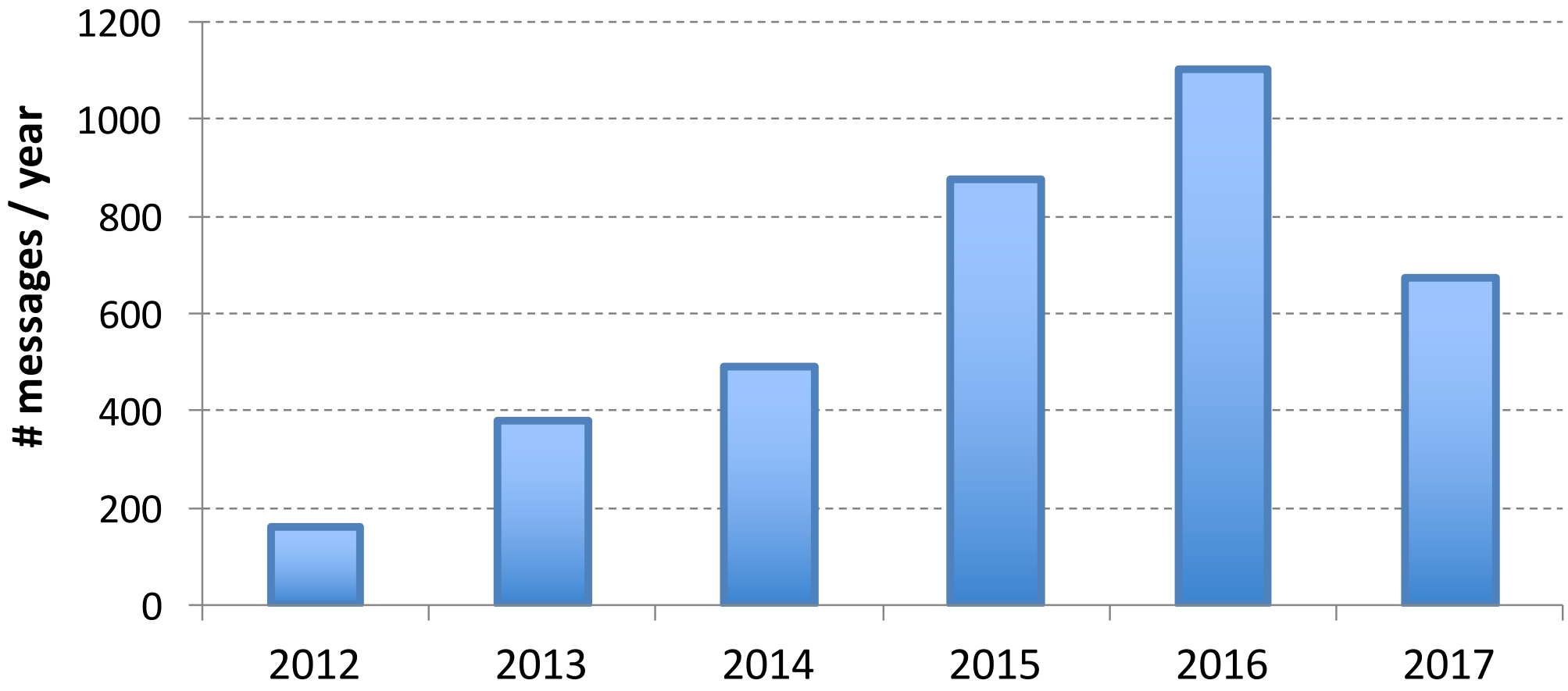
Community aspects

- EasyBuild community has been growing rapidly the last couple of years
- **hundreds of HPC sites worldwide, large and small**
- very welcoming & supportive to newcomers
- significant overlap between EasyBuild & Lmod communities
- active mailing lists: https://lists.ugent.be/wws/info/easybuild
- EasyBuild also has active IRC and Slack channels
- users are also contributing: bug reports, feature requests, code contributions, ...

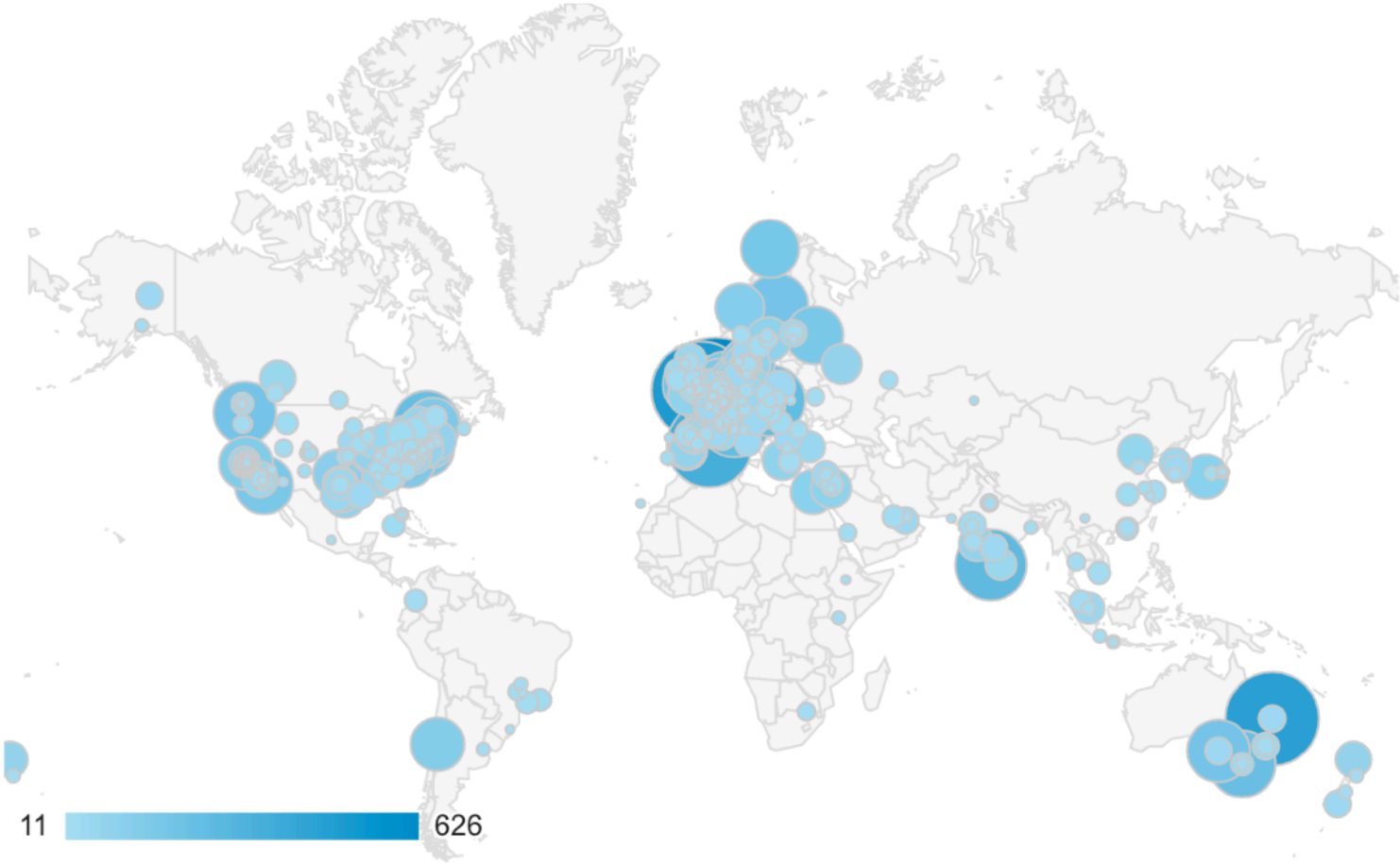
EasyBuild mailing list (subscribers)



EasyBuild mailing list (traffic)

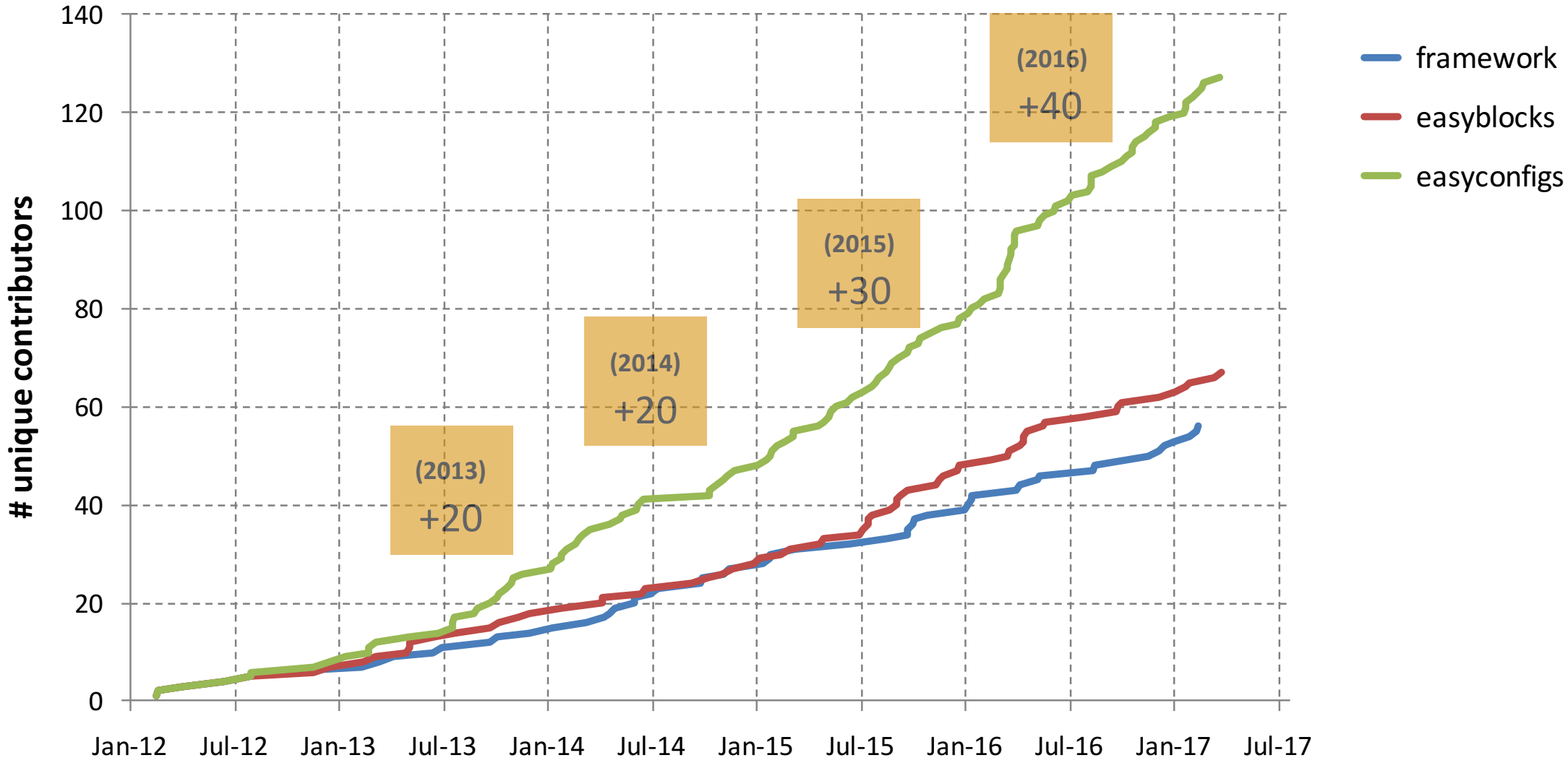


EasyBuild community



cities from where EasyBuild documentation was visited at least 10 times during the last year
source: Google Analytics for easybuild.readthedocs.io

Code contributions to EasyBuild



EasyBuild requirements

http://easybuild.readthedocs.io/en/latest/Installation.html#requirements

- focus is on Linux x86_64 HPC systems
  - RedHat-based (CentOS, SL, ...), also on Debian and SuSE derivatives
  - also (kind of) works on macOS, but not a target platform
  - Windows support only via new Windows Subsystem for Linux (WSL)
  - ongoing efforts to enhance support on AARCH64 (ARM) and PowerLinux
- Python 2.6 or more recent Python 2, incl. setuptools (no Python 3 support yet)
- system C/C++ compiler (GCC), used to kickstart installation of compiler toolchain
- an environment modules tool (Lmod is highly recommended!)

Installing EasyBuild

http://easybuild.readthedocs.io/en/latest/Installation.html

- installing Python packages can be messy, cfr. plethora of Python installation tools
- **bootstrap script** for EasyBuild was created out of frustration

```
# download bootstrap script from
# https://raw.github.com/hpcugent/easybuild-framework/develop/easybuild/scripts/bootstrap_eb.py

$ python bootstrap_eb.py $PREFIX
$ module use $PREFIX/modules/all
$ module load EasyBuild
```

- standard installation tools like `easy_install`, `pip`, etc. work too
- packaging efforts under way (see OpenHPC; WIP in Fedora)

Updating EasyBuild

http://easybuild.readthedocs.io/en/latest/Installation.html#updating-an-existing-easybuild-installation

- new releases are fully backwards-compatible (except for new major versions)
- to update EasyBuild:
  - re-bootstrap to obtain a module for latest EasyBuild release
  - or use '`eb --install-latest-eb-release`' to install module for latest release
  - install a specific EasyBuild version with EasyBuild using an easyconfig file

    https://github.com/hpcugent/easybuild-easyconfigs/tree/develop/easybuild/easyconfigs/e/EasyBuild

Configuring EasyBuild (1)

http://easybuild.readthedocs.io/en/latest/Configuration.html

You should configure EasyBuild to your preferences, via:

- configuration file(s): key-value lines, text files (e.g., `prefix=/tmp`)
- environment variables (e.g., `$EASYBUILD_PREFIX` set to `/tmp`)
- command line parameters (e.g., `--prefix=/tmp`)

Each configuration option can be set on each of these configuration levels.

Priority among different options: cmdline, env vars, config file. For example:

- `--prefix` overrules `$EASYBUILD_PREFIX`
- `$EASYBUILD_PREFIX` overrules `prefix` in configuration file

Configuring EasyBuild (2)

http://easybuild.readthedocs.io/en/latest/Configuration.html

By default, EasyBuild will (ab)use `$HOME/.local/easybuild`.

Use '`eb --show-config`' to get an overview of the current configuration.

```
$ EASYBUILD_PREFIX=/tmp eb --buildpath /dev/shm --show-config
#
# Current EasyBuild configuration
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
#
buildpath       (C) = /dev/shm
installpath     (E) = /tmp
packagepath     (E) = /tmp/packages
prefix          (E) = /tmp
repositorypath (E) = /tmp/ebfiles_repo
robot-paths     (D) = /Users/kehoste/work/easybuild-easyconfigs/easybuild/easyconfigs
sourcepath      (E) = /tmp/sources
```

Basic usage

http://easybuild.readthedocs.io/en/latest/Using_the_EasyBuild_command_line.html

http://easybuild.readthedocs.io/en/latest/Typical_workflow_example_with_WRF.html

- specify software name/version and toolchain to 'eb' command
- commonly via easyconfig filename(s):

```
eb GCC-4.9.2.eb Clang-3.6.0-GCC-4.9.2.eb
```

- check whether required toolchain & dependencies are available using `--dry-run/-D`:

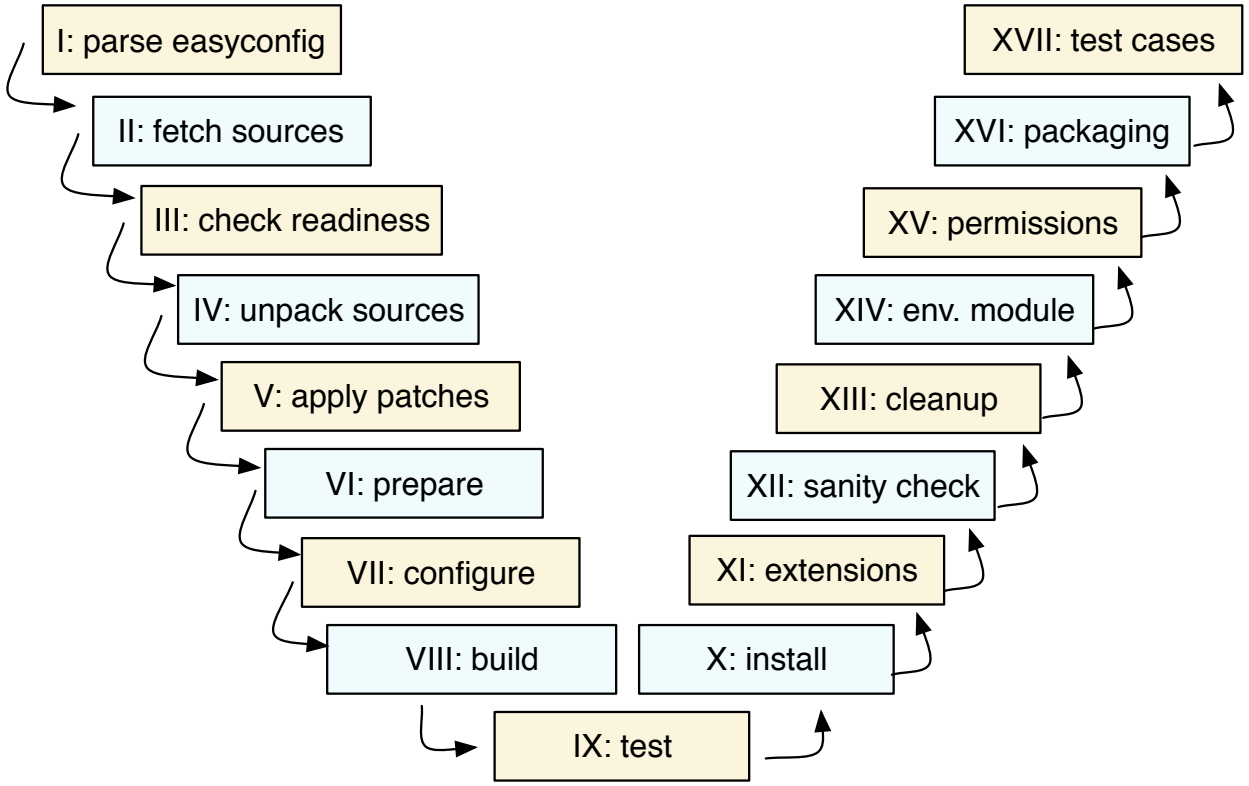
```
eb Python-2.7.13-intel-2017a.eb -D
```

- enable dependency resolution via `--robot/-r`:

```
eb WRF-3.8.0-intel-2016b-dmpar.eb -dr
```

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 a simple sanity check on installation
- generate module file

each step can be customised via easyconfig parameters or an easyblock

Example output of using 'eb' command

```
$ eb GCC-4.9.3.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig GCC-4.9.3.eb
== building and installing GCC/4.9.3...
== 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 /opt/easybuild/software/GCC/4...
== 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.
```

Easyconfig files as build specifications

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

- **simple text files including 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
  - (build) dependencies
  - options for configure/build/install commands
  - files and directories that should be present (sanity check)

Example easyconfig file

```
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'),
]
```

Example easyconfig file

software name and version

software metadata

```
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'),
]
```

Example easyconfig file

software name and version

software metadata

toolchain name & version

sources & patches

```
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'),
]
```

Example easyconfig file

software name and version

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'),
]
```

Example easyconfig file

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'),
]
```

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'),
]
```

Log files

http://easybuild.readthedocs.io/en/latest/Logfiles.html

- **EasyBuild thoroughly logs executed installation procedure**
  - active EasyBuild configuration
  - easyconfig file that was used
  - modules that were loaded + resulting changes to environment
  - defined environment variables
  - output of executed commands
  - informative log messages produced by easyblock
- log is file is copied to software installation directory for future reference
- can be used to debug build problems or see how installation was performed exactly

Log files: example

```
== 2016-04-24 13:34:31,906 main.EB_HPL INFO This is EasyBuild 3.1.2 (framework:
3.1.2, easyblocks: 3.1.2) on host example.
...
== 2016-04-24 13:34:35,503 main.EB_HPL INFO configuring...
== 2016-04-24 13:34:48,817 main.EB_HPL INFO Starting configure step
...
== 2016-04-24 13:34:48,823 main.EB_HPL INFO Running method configure_step part of
step configure
...
== 2016-04-24 13:34:48,823 main.run DEBUG run_cmd: running cmd /bin/bash
make_generic (in /tmp/user/easybuild_build/HPL/2.0/goolf-1.4.10/hpl-2.0/setup)
== 2016-04-24 13:34:48,823 main.run DEBUG run_cmd: Command output will be logged
to /tmp/easybuild-W85p4r/easybuild-run_cmd-XoJwMY.log
== 2016-04-24 13:34:48,849 main.run INFO cmd "/bin/bash make_generic" exited with
exitcode 0 and output:
...
```

Adding support for additional software

- for each software installation, an **easyconfig file** is required
  - defines easyconfig parameters that specify to EasyBuild what to install, and how
  - existing easyconfig files can serve as examples
  - for version or toolchain updates, a tweaked easyconfig can be *generated* via `eb --try-*`
  - see http://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html
- for 'standard' installation procedures, a **generic easyblock** can be used
  - generic installation can be controlled where needed via easyconfig parameters
- for custom installation procedures, a **software-specific easyblock** is must be implemented
  - see http://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 Python packages ('`python setup.py install`', '`pip install`', ...)
- **`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

Common toolchains

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

- `intel` and `foss`[1] toolchains are most commonly used in EasyBuild community
- helps to focus efforts of HPC sites using one or both of these toolchains
- updated twice a year, clear versioning scheme (`2016b`, `2017a`, ...)
- latest version:
  - `foss/2017a`

    binutils 2.27, GCC 6.3.0, OpenMPI 2.0.2, OpenBLAS 0.2.19, LAPACK 3.7.0, FFTW 3.3.6
  - `intel/2017a`

    binutils 2.27 + GCC 6.3.0 as base

    version 2017.1.132 of Intel compilers, Intel MPI and Intel MKL

(1) FOSS: Free and Open Source Software

Transparency of performed install procedure

http://easybuild.readthedocs.io/en/latest/Extended_dry_run.html

- **'`eb --extended-dry-run`', '`eb -x`' reveals planned installation procedure**
- runs in a matter of seconds
- shows commands that will be executed, build environment, generated module file, ...
- any errors that occur in used easyblock are ignored but clearly reported
- not 100% accurate since easyblock may require certain files to be present, etc.
- very useful when debugging easyblocks, instant feedback as a first pass
- implementation motivated by requests from the community
- helps to avoid impression that EasyBuild is a magic black box for installing software

Example output of --extended-dry-run (1)


```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x

== temporary log file in case of crash /tmp/eb-Dh1wOp/easybuild-0bu9u9.log

== processing EasyBuild easyconfig /home/example/eb/easybuild-easyconfigs/easybuild/
easyconfigs/w/WRF/WRF-3.8.0-intel-2016b-dmpar.eb

...

*** DRY RUN using 'EB_WRF' easyblock (easybuild.easyblocks.wrf @ /home/example/eb/easybuild-
easyblocks/easybuild/easyblocks/w/wrf.py) ***

== building and installing WRF/3.8.0-intel-2016b-dmpar...
fetching files... [DRY RUN]

[fetch_step method]
Available download URLs for sources/patches:
  * http://www2.mmm.ucar.edu/wrf/src//$source
  * http://www.mmm.ucar.edu/wrf/src//$source

List of sources:
  * WRFV3.8.0.TAR.gz will be downloaded to /home/example/eb/sources/w/WRF/WRFV3.8.0.TAR.gz
```

Example output of `--extended-dry-run` (2)

```
...

building... [DRY RUN]

[build_step method]
  running command "tcsh ./compile -j 4 wrf"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsh ./compile -j 4 em_real"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsh ./compile -j 4 em_b_wave"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
...

[sanity_check_step method]
Sanity check paths - file ['files']
  * WRFV3/main/libwrflib.a
  * WRFV3/main/real.exe
  * WRFV3/main/wrf.exe
Sanity check paths - (non-empty) directory ['dirs']
  * WRFV3/main
  * WRFV3/run
Sanity check commands
  (none)
```

Example output of --extended-dry-run (3)

```
...

[make_module_step method]
Generating module file /home/example/eb/modules/all/WRF/3.8.0-intel-2016b-dmpar,
with contents:

    #%Module
    proc ModulesHelp { } {
        puts stderr { The Weather Research and Forecasting (WRF) Model }
    }
    module-whatis {Description: WRF - Homepage: http://www.wrf-model.org}

    set root /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar

    conflict WRF

    if { ![ is-loaded intel/2016b ] } {
        module load intel/2016b
    }

    if { ![ is-loaded JasPer/1.900.1-intel-2016b ] } {
        module load JasPer/1.900.1-intel-2016b
    }
```

Using a custom module naming scheme

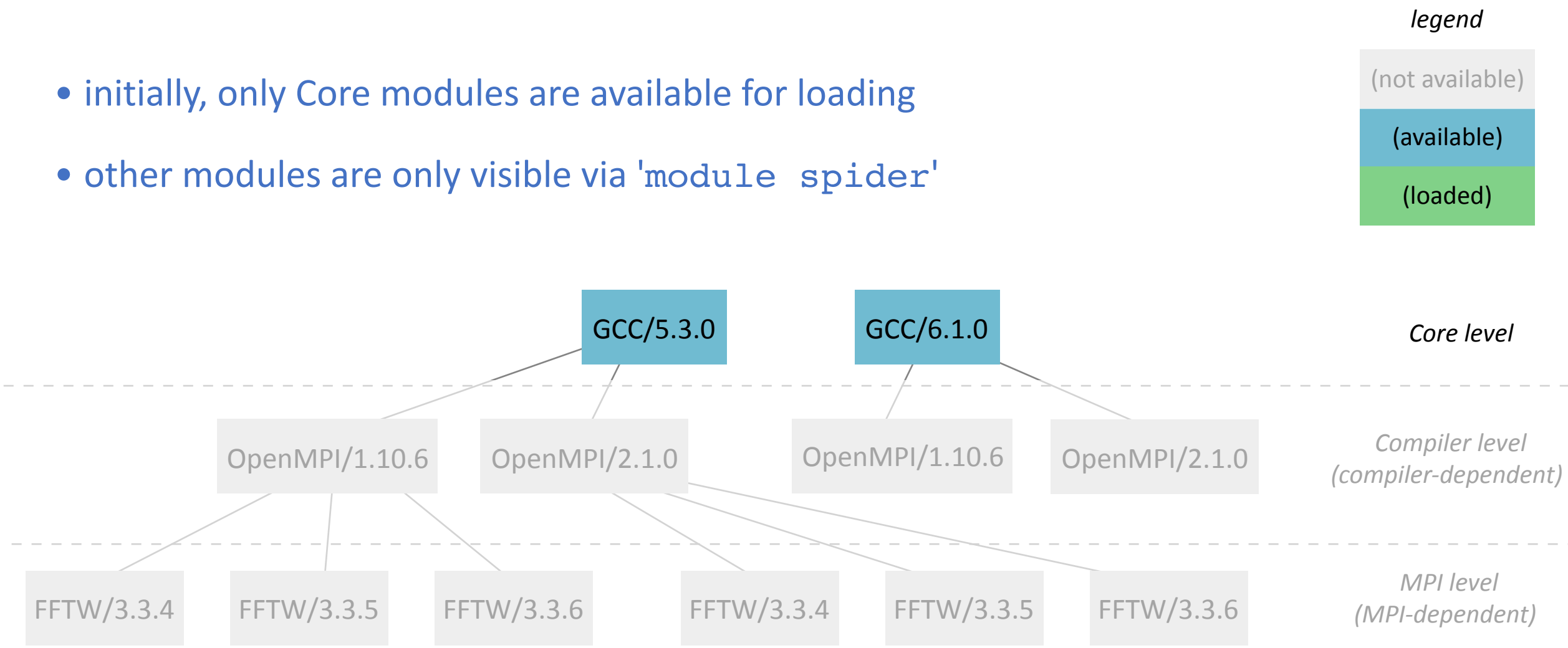
- a couple of different module naming schemes are included in EasyBuild
  - see `--avail-module-naming-schemes`
  - specify active module naming scheme via `--module-naming-scheme`
  - default: `EasyBuildMNS` (`<name>/<version>-<toolchain>-<versionsuffix>`)
- **you can implement your own module naming 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 vs hierarchical module naming scheme

- **flat module naming scheme**:
  - all existing modules are always available for loading
  - downsides:
    - long (confusing) module names to distinguish build with different toolchains
    - users can easily shoot themselves in the foot by (trying) to load incompatible modules
- **hierarchical module naming scheme**:
  - modules are organised in a tree-like fashion
  - a 'core' module must be loaded first to make more (compatible) modules available
  - typically used with different hierarchy levels: Core - Compiler - MPI

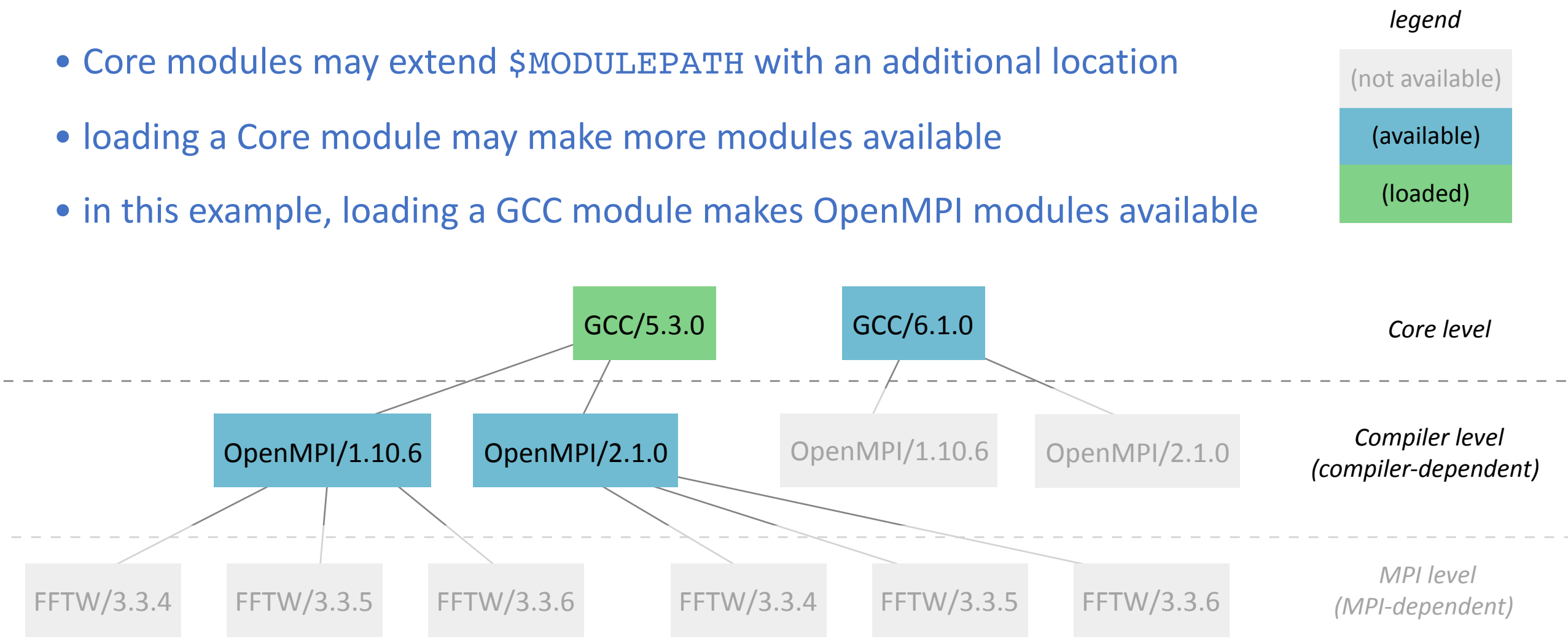
Hierarchical module naming scheme in detail

- initially, only Core modules are available for loading
- other modules are only visible via '`module spider`'



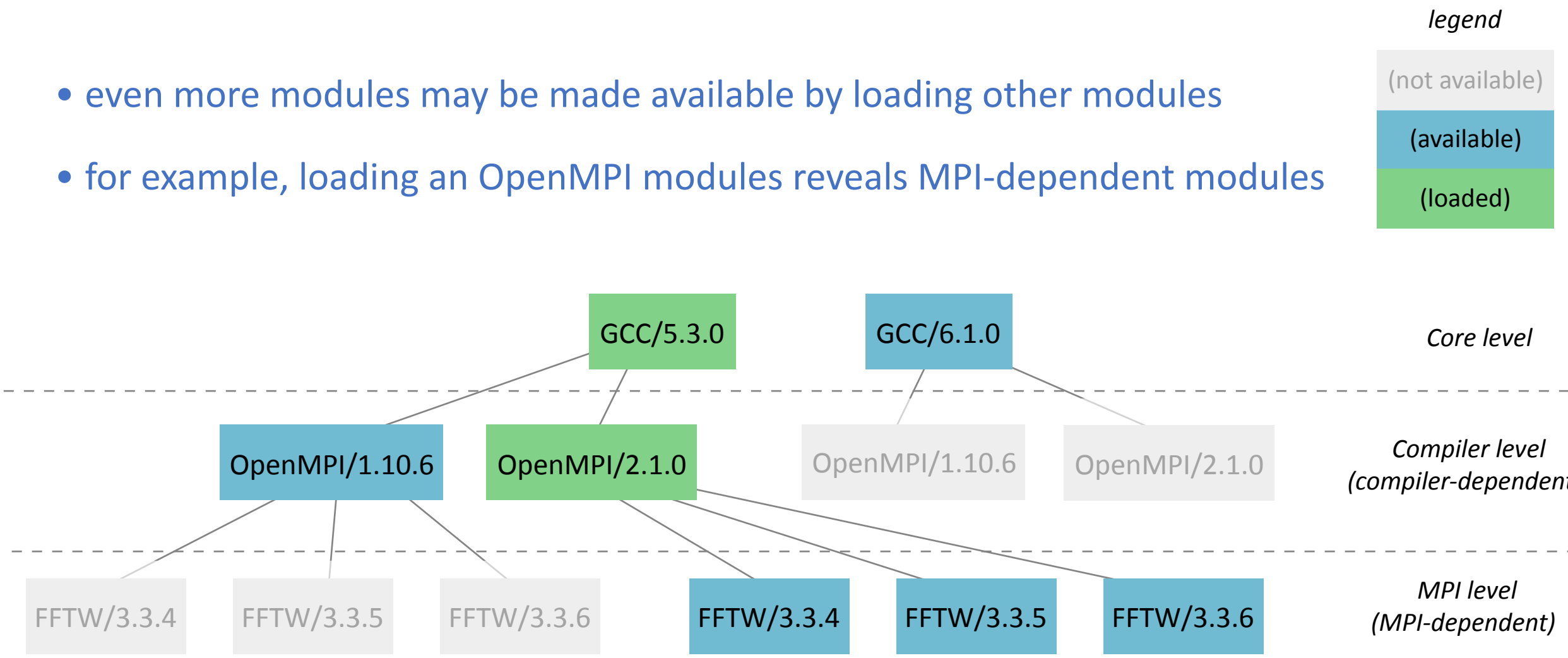
Hierarchical module naming scheme in detail

- 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 in detail

- even more modules may be made available by loading other modules
- for example, loading an OpenMPI modules reveals MPI-dependent modules



Integration with GitHub

http://easybuild.readthedocs.io/en/latest/Integration_with_GitHub.html

- **access to GitHub straight from EasyBuild command line (!)**
- '`eb --from-pr`' to pull in easyconfigs from a specific pull request
- `--upload-test-report` to add comment with test report into a pull request
- '`eb --new-pr`' to create a new pull request for contributing (changes to) easyconfigs
- '`eb --update-pr`' to add updates to existing pull requests
- **significantly lowers the bar for contributing,** no need to be familiar with `git`
- also turns out to be a **huge time saver** for experienced contributors
- for now only supported for easyconfig files, soon also for easyblocks & framework

Other features that were not covered...

http://easybuild.readthedocs.io

- letting users manage their software stack on top of centrally provided modules
- installing hidden modules, hiding certain dependencies & toolchains
- support for using RPATH linking *(experimental)*
- partial installations: only (re)generate module file, install additional extensions
- dynamically extending EasyBuild via `--include-*`
- submitting installations as jobs to an HPC cluster via `--job`
- creating packages (RPMs, ...) for software installations done with EasyBuild
- using EasyBuild on Cray systems, integration with Cray Programming Environment

Contributing to EasyBuild

http://easybuild.readthedocs.io/en/latest/Contributing.html

EasyBuild has improved significantly thanks to the community.

You too can contribute back, by:

- sending feedback
- reporting bugs
- joining the discussion (mailing lists, EasyBuild conf calls)
- sharing suggestions and ideas for changes & additional features
- contributing easyconfigs, enhancing easyblocks, adding support for new software...
- extending & enhancing documentation

vs Spack

- Spack (http://spack.io/) is a package manager focused on scientific software and HPC
- lead development by Todd Gamblin (LLNL)
- significant growth in user base in recent months
- similar yet different approach compared EasyBuild
  - **very flexible command line interface w.r.t. specifying what to install**
  - concretises partial specification, dependency versions don't need to be fixed
  - focus is more on software developers
  - some experience with Git & Python is assumed/required

Papers on EasyBuild (& Lmod)

Modern Scientific Software Management Using EasyBuild and Lmod

Markus Geimer (JSC), Kenneth Hoste (HPC-UGent), Robert McLay (TACC)

http://hpcugent.github.io/easybuild/files/hust14_paper.pdf

Making Scientific Software Installation Reproducible On Cray Systems Using EasyBuild

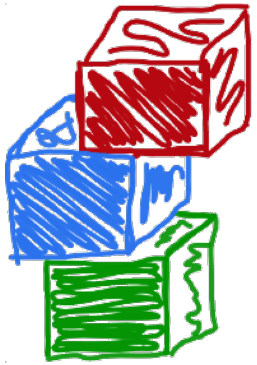
Petar Forai (IMP), Guilherme Peretti-Pezzi (CSCS), Kenneth Hoste (HPC-UGent)

https://cug.org/proceedings/cug2016_proceedings/includes/files/pap145.pdf

Scientific Software Management in Real Life: Deployment of EasyBuild on a Large Scale System

Damian Alvarez, Alan O'Cais, Markus Geimer (JSC), Kenneth Hoste (HPC-UGent)

http://hpcugent.github.io/easybuild/files/eb-jsc-hust16.pdf



easybuild

Getting Scientific Software Installed with EasyBuild

June 12th 2017 - Francis Crick Institute

remote presentation

http://users.ugent.be/~kehoste/EasyBuild-intro_20170612_FrancisCrick.pdf

kenneth.hoste@ugent.be

http://hpcugent.github.io/easybuild/

GHENT UNIVERSITY

http://www.ugent.be/hpc

https://www.vscentrum.be

Vlaams Supercomputer Centrum