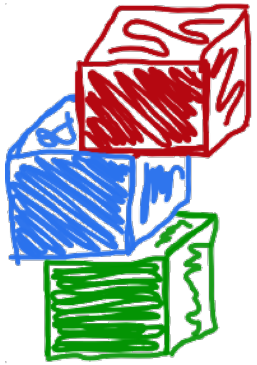


easybuild

Modern Scientific Software Management using EasyBuild & co

PRACE-VI-SEEM 2017 Spring School - System Administration Track

April 25th 2017 - The Cyprus Institute

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)

Scope

- focus is on **tools for HPC systems**, mostly related to user support
- in particular, **installing (scientific) software** for users of HPC clusters
- only actively maintained, well-documented **open source** software projects
- important things I will *not* be talking about (in detail):
  - configuration management (tips: Ansible, Warewulf, Quattor)
  - security, monitoring, cloud services, ...
  - MPI, performance tuning, accelerators, ...
  - the latest hype (Big Data, Deep Learning, ...)

Outline

- **Birds-eye view of tools & projects**
  - EasyBuild, Lmod, Singularity, ClusterShell, XALT/OGRT, OpenHPC
- **Introduction to EasyBuild & Lmod**
  - Scope, motivation, terminology & features
  - Installation and configuration
  - Typical usage
  - Advanced topics

Birds-eye view of tools & projects

- **EasyBuild**
- **Lmod**
- Singularity
- ClusterShell
- XALT & OGRT
- OpenHPC

Use case: installing (scientific) software

artwork on VSC BrENIAC Tier-1 system

- significant part of user support services
- traditional package managers are not sufficient
- installation should be done in an optimised way (from source)
- users often lack required knowledge to handle this themselves (efficiently)
- scientific software in particular can be... 'challenging'
- little collaboration between HPC sites, even though this is a common burden

framework for building & installing (scientific) software on HPC clusters

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

will be covered in detail later...

Use case: providing easy access to software

- lots of (scientific) software is typically provided on HPC clusters
- usually in non-standard locations, different versions & variants, ...
- users need to query what software is readily available and get easy access to it
- well established solution on HPC clusters: *environment modules*
- traditional environment modules tool is outdated, no longer actively maintained...

Lmod

a modern environment modules tool

https://www.tacc.utexas.edu/research-development/tacc-projects/lmod

will be covered in detail later...

Birds-eye view of tools & projects

- EasyBuild: getting (scientific) software installed
- Lmod: providing easy access to software installations
- **Singularity**
- ClusterShell
- XALT & OGRT
- OpenHPC

Use case: same workflow on local & HPC systems

- switching to HPC cluster usually requires users to change their workflow
  - different environment to work in (env. modules, etc.)
  - required tools are missing or installed differently
  - maybe also different versions, configuration, etc.
- confusing and time-consuming for users
- reproducing the environment they are used to may require a lot of effort
  - users may be unable to take care of it themselves
    (lack permissions & technical expertise, dependency hell, ...)
  - support teams are busy, may lack domain knowledge

Singularity

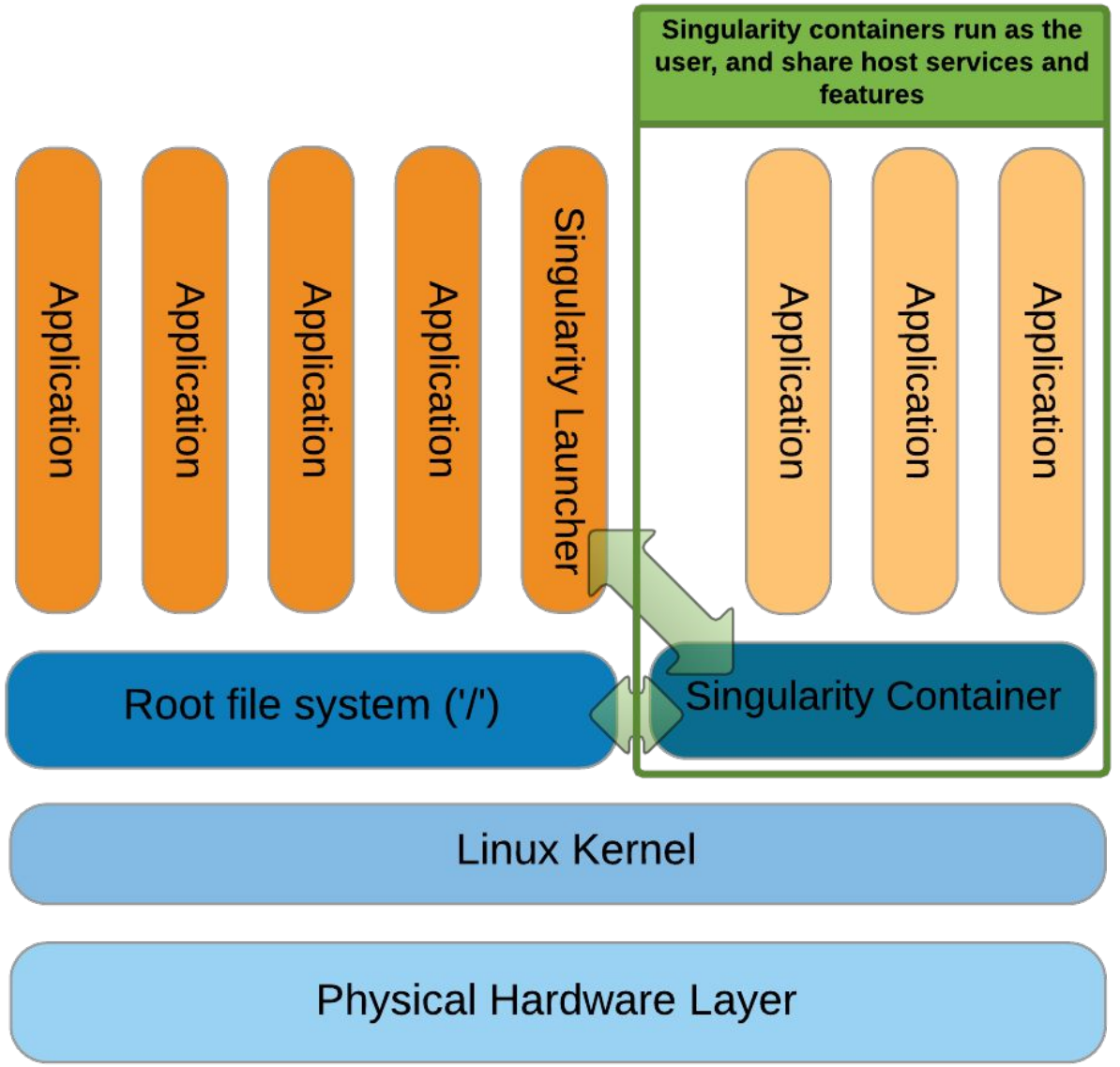
- http://singularity.lbl.gov/ - http://singularity.lbl.gov/user-guide
- **container solution for HPC environments**
- provides *"mobility of compute"*: container images can be used on any (Linux) system
- enables reproducible science: container images include everything that is required
- deals with security concerns of other container solutions like Docker
- good support for accelerators (GPUs, ...), MPI, Infiniband, etc.

Singularity

- developed by HPC team at Lawrence Berkeley National Laboratory (LBNL)
- project lead: Gregory M. Kurtzer (also known from CentOS Linux, Warewulf)
- first public release in April 2016, *massive* uptake since
- latest release: Singularity v2.2.1 (Feb 14th 2017)
- download from: https://github.com/singularityware/singularity/releases
- to be included in OpenHPC v1.3.1

Singularity architecture

- similar to other container solutions
- limited use of namespaces
- lower overhead than Docker & co
- easy to integrate



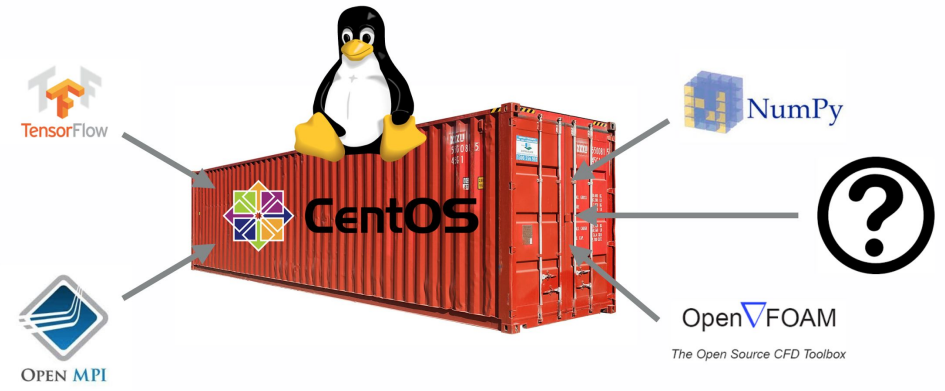
(figure courtesy of Gregory M. Kurtzer (LBNL)

Security concerns with containers on HPC systems

- regular users must *not* be able to gain admin privileges in any way...
  - sensitive issue because of shared filesystems & lots of resources
- **no privilege escalation allowed in Singularity**
  - calling user is maintained within the container
  - if you want to be root inside container, you *must* be root outside the container
- no root owned daemon process (as opposed to Docker)
- **containers are started as user processes**

Mobility of compute, reproducible science & BYOE

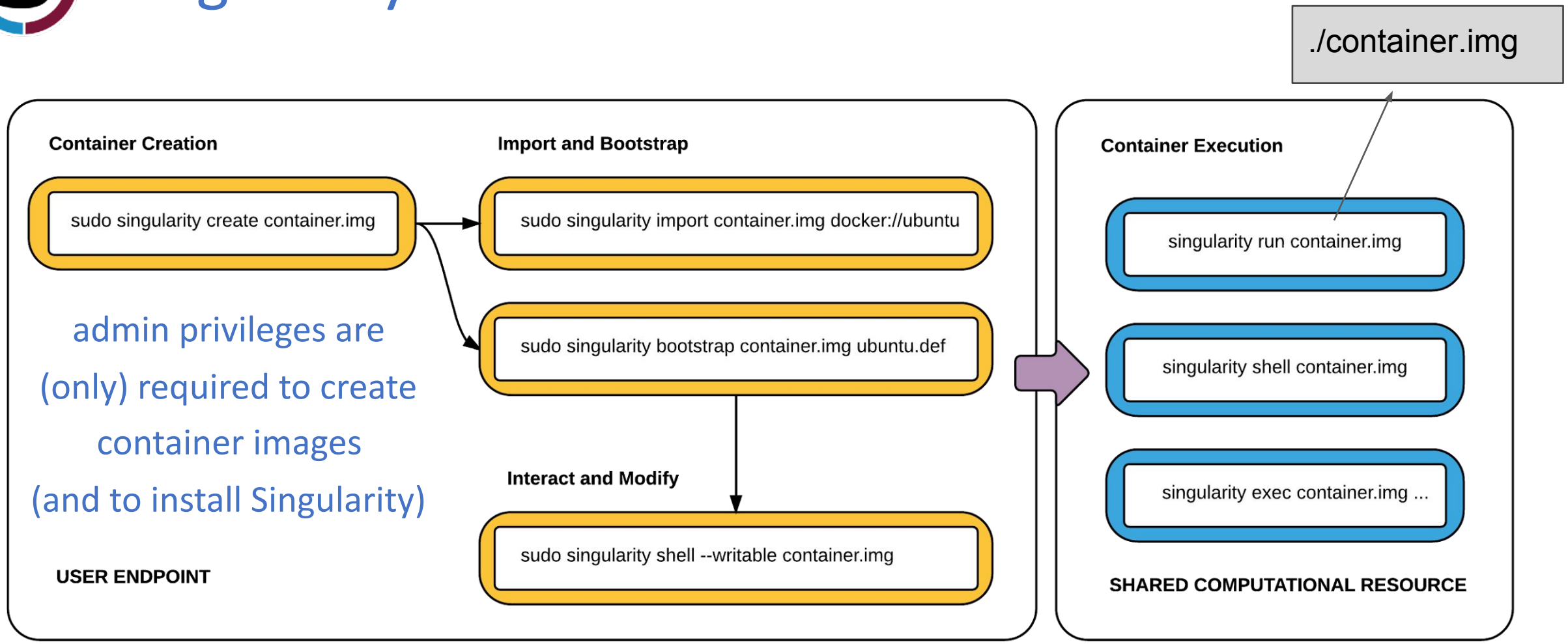
- container images are *single files*
  - easy to distribute & share
  - very efficient on parallel filesystems
- self-contained container images: "it just works"
- container images can be used on any Linux system
- Bring Your Own Environment (BYOE)
  - user is familiar with environment encapsulated in container



Integration with available resources

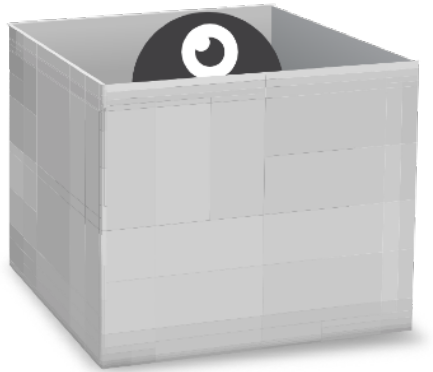
- Singularity is designed from the ground up to integrate easily with available resources
- containers are launched via resource manager, just like regular jobs
  - no changes required to system configuration, it just works
- access to accelerators (GPU, Xeon Phi, ...) is easy to come by
- some effort is required to use MPI and leverage interconnects like Infiniband
  - requires necessary libraries to be available *in* the container image...
- shared filesystems can be bound into the container through Singularity configuration
- supports multiple architectures (x86_64, POWER, AARCH64, ...)

Singularity workflow



(figure courtesy of Gregory M. Kurtzer (LBNL)

Singularity Hub

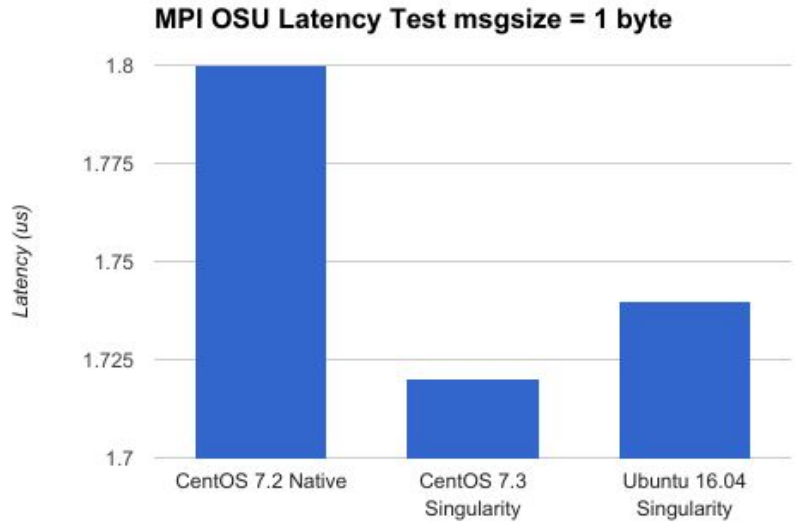


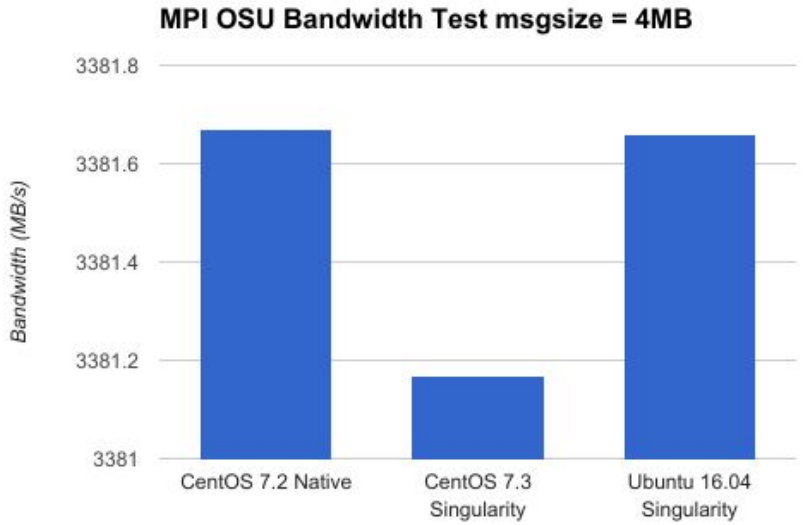
- https://singularity-hub.org/
- created by Vanessa Sochat (Stanford's Research Computing group)
- platform to build & archive Singularity container images
- designed to facilitate reproducible research, publications & peer-review
- integrated with GitHub to trigger builds and automate containerisation
- container images hosted on Singularity Hub can be used directly:

```
singularity run shub://singularityhub/$USER/$CONTAINER:$TAG
```

Singularity: performance

- benchmarking experiments show **very low overhead** compared to native execution
- lower overhead compared to Docker & co
- sometimes even (slightly) better performance than native! (note scale on Y-axis, diff. is *small*)





data provided by Paulo Souza (paulo7@gmail.com), via Gregory M. Kurtzer

Singularity: more information

- http://singularity.lbl.gov - https://github.com/singularityware/singularity
- presentations:
  - FOSDEM'17 (by Michael Bauer, University of Michigan)
    - https://fosdem.org/2017/schedule/event/singularity
    - https://fosdem.org/2017/schedule/event/singularityhpc
  - HPCAC Stanford conference '17 (by Gregory M. Kurtzer)
    - http://insidehpc.com/2017/02/video-singularity-containers-science-reproducibility-hpc
- https://www.nextplatform.com/2017/04/10/singularity-containers-hpc-reproducibility-mobility

Birds-eye view of tools & projects

- EasyBuild: getting (scientific) software installed
- Lmod: providing easy access to software installations
- Singularity: containers for HPC
- **ClusterShell**
- XALT & OGRT
- OpenHPC

Use case: running commands on remote systems

- HPC system administrators often need to run commands on remote systems
  - for example: workernodes in an HPC cluster, login nodes, manager nodes, etc.
- poor solution: for-loop to run a command via SSH on each system
  - very slow for large number of systems (even though it is embarrassingly parallel...)
  - hard to compare command outputs
  - specifying list of target nodes may be tricky (e.g., excluding nodes that are down)
- support for running remote commands via a custom Python script would be useful
- `pdsh`: no Python integration, no integrated CLI (`pdsh`, `dshbak`, `pdcp`), scaling issues, ...

ClusterShell

- website: http://cea-hpc.github.io/clustershell/
- documentation: http://clustershell.readthedocs.io/
- **tool for running shell commands on remote systems, in parallel and scalable**
- support for managing node sets, merging/diffing command outputs, ...
- can also be used as a Python library (`import ClusterShell`)
- (originally) developed by CEA HPC center (France)
- latest release: ClusterShell v1.7.3 (Dec 20th 2016)
- download from: https://github.com/cea-hpc/clustershell/releases

ClusterShell: `clush` command

- **tool for remote execution of shell commands**, compatible with `pdsh`
- supports SSH, RSH + custom connection modes using Python plugins
- target nodes are specified via node sets (`-w`) or predefined/dynamic node groups (`-g`)
- supports copying of files (in parallel, both ways) via `-c`/`--copy` and `--rcopy`
- smart display of command results using `-b`/`-B`/`-L` options
  - integrated output gathering, sorting by node, node set or node groups
- degree of parallelism can be controlled via `-f`/`--fanout`
- supports configurable 'tree mode' for better scalability

ClusterShell: `nodeset` and `clubak` commands

- `nodeset`: **tool for manipulation of node sets, node groups**
  - expanding (`-e`), folding (`-f`), counting (`-c`) of node names/sets/groups
  - node groups: YAML files, or via external sources (e.g. SLURM's `sinfo` command)
  - output format can be tweaked according to needs (`-O`)
  - supports nodeset arithmetic: joining sets, excluding elements, etc.
- `clubak`: **utility to summarise output and highlight differences**
  - backwards compatible with `dshbak` utility that is available with `pdsh`
  - leveraged by `clush` when `-b`/`-B`/`-L` is used

ClusterShell as Python library

- nodeset manipulation:

```
from ClusterShell.NodeSet import NodeSet

ns1 = NodeSet('node24[01-10]')
ns2 = NodeSet('@login')

for node in ns1 | ns2:
    print node
```

- command execution:

```
from ClusterShell.Task import task_self

task_self().run("hostname", nodes='node24[01-10]')

for output, nodes in task_self().iter_buffers():
    for node in nodes:
        print "%s: %s" % (node, output)
```

ClusterShell: demo

```
# example usage of nodeset command
$ nodeset -e 'node24[01-5]'
node2401 node2402 node2403 node2404 node2405
# predefined group of 200 workernodes in golett cluster
$ nodeset -c @golett
200

# output for 'module --version' on all golett workernodes (10 nodes in parallel)
$ clush -B -f 10 -w @golett module --version
---------------
node[2400-2599] (200)
---------------

Modules based on Lua: Version 6.6  2016-10-13 13:28 -05:00
    by Robert McLay mclay@tacc.utexas.edu
```

ClusterShell: more information

- website: http://cea-hpc.github.io/clustershell/
- documentation: http://clustershell.readthedocs.io/
- wiki: https://github.com/cea-hpc/clustershell/wiki
- support:
  - mailing list: https://lists.sourceforge.net/lists/listinfo/clustershell-devel
  - issue tracker: https://github.com/cea-hpc/clustershell/issues
- article: http://slashdot.org/journal/266980/A-Quick-Look-on-ClusterShell
- presentation: https://archive.fosdem.org/2016/schedule/event/hpc_bigdata_clustershell/

Birds-eye view of tools & projects

- EasyBuild: getting (scientific) software installed
- Lmod: providing easy access to software installations
- Singularity: containers for HPC
- ClusterShell: executing commands on remote systems
- **XALT & OGRT**
- OpenHPC

Use case: tracking software usage on HPC systems

- **knowing which (scientific) software is being used is important**
  - which scientific software applications consume most of available compute time?
  - provided optimised installations *vs* available via OS *vs* user-installed
  - are provided software installations actually used long-term?
  - are old software installations still being used?
- just checking which modules are being loaded is not reliable
  - loading modules does not imply the software they provide will actually be used
  - what if multiple modules are loaded in jobs?
- usage tracking should be very light-weight and should never cause any problems...

XALT

- https://github.com/Fahey-McLay/xalt - http://xalt.readthedocs.io
- latest stable release: XALT v1.1.2 (Jan 4th 2017), XALT v2 is under active development
- **tool to track which programs/libraries users are using on HPC systems**
- successor of ALTD (http://www.nersc.gov/users/software/programming-libraries/altd/)
- XALT v2 is implemented in C/C++, developed by Robert McLay (TACC)
- light-weight, transparent & robust
- hijacks linker (`ld`) to fingerprint executables so they can be tracked in detail
- runtime tracking by XALT triggered via `init`/`fini` in ELF header of binaries

OGRT

- https://github.com/georg-rath/ogrt
- latest release: OGRT v0.4.1 (Sept 28th 2016)
- **(another) tool to track which programs/libraries users are using on HPC systems**
- implemented in C/Go, by Georg Rath (freelancer)
- light-weight (tracking all in-memory) and transparent, resistant to outside influence
- no runtime dependencies, easy to deploy
- only works with dynamic binaries, relies on `$LD_PRELOAD`

OGRT vs XALT side-by-side comparison

OGRT v0.4.1

- implemented in C/Go
- backends: SPLUNK, ElasticSearch
- heritage: track many small jobs/executables
- real-time data collection,
  sent to OGRT server (via UDP)
- tracking via `$LD_PRELOAD`
- tracks *all* MPI ranks
- used at: IMBA, <private company>

XALT v1.6.1-devel (pre-v2)

- implemented in C/C++
- backends: database, file, syslog
- heritage: track large MPI jobs (also tracks non-MPI)
- data to file/syslog & digested nightly to DB,
  can also do direct-to-DB in real-time
- tracking via ELF `init`/`fini` or `$LD_PRELOAD`
- only tracks MPI rank 0 (ignores other ranks)
- used at: TACC, NICS, Univ. Florida, KAUST, ...
- commercial support by Ellexus

(Big) Data collection with XALT/OGRT

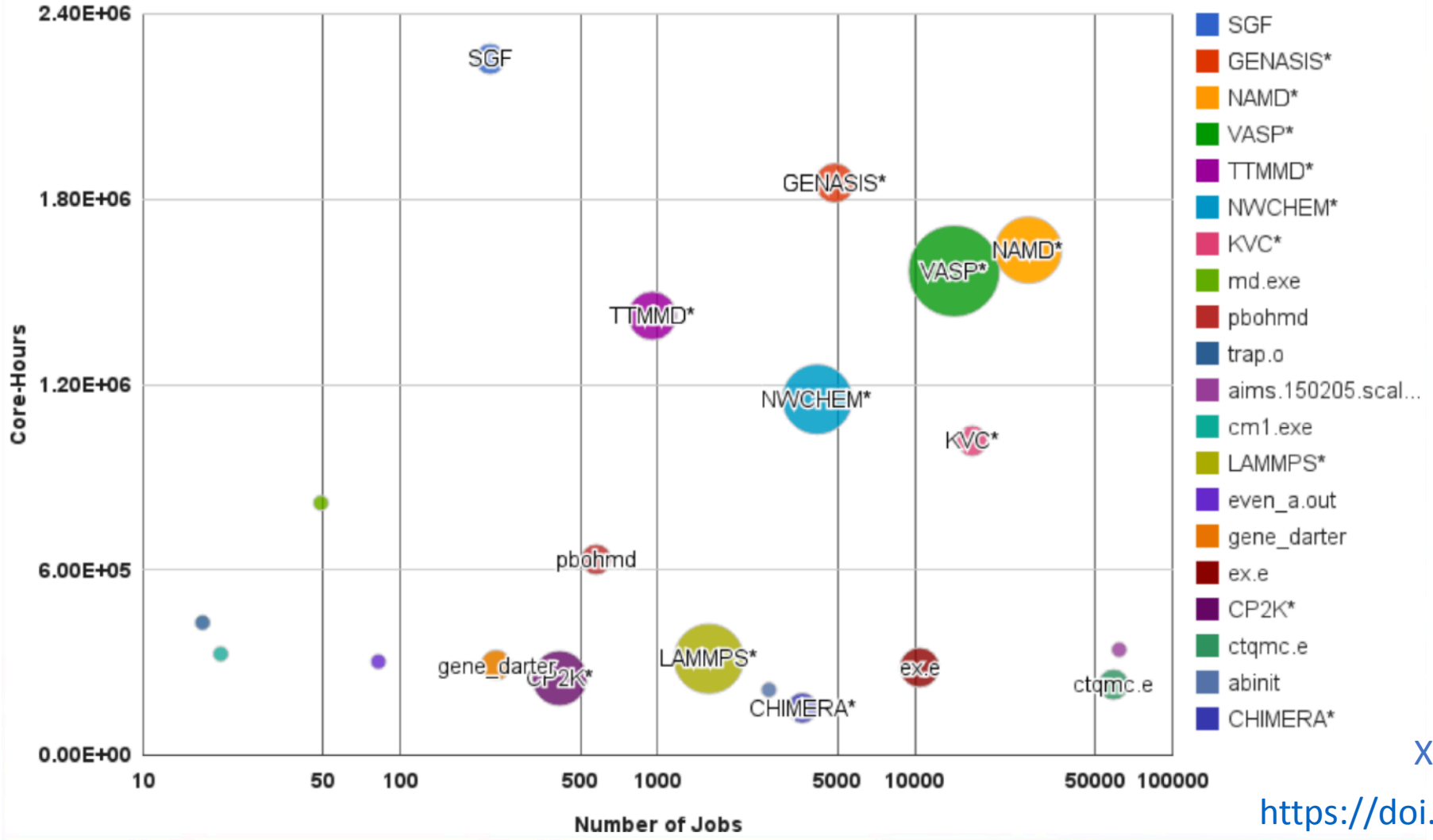
- quote by Robert McLay: *"Big Data sucks!"*
- easy to collect *a lot* of data on software being used in jobs
  - by default *all* executables are tracked, incl. standard tools like `ls`, `cd`, `cp`, etc.
  - also captures all defined environment variables (typically ~250 per event)
  - ~17KB of data per event, easily >1GB /day in total...
- filtering is required to focus on events of interest:
  - only executables & environment variables that you care about
  - only on hosts that really matter
- extra care should be taken with MPI jobs (e.g., XALT only tracks rank 0)

Use cases for data collected with XALT/OGRT

Collected data open up whole range of potential use cases:

- **targeted outreach**
  - find out which users are using old software versions, buggy libraries, etc.
- **verifying how software is being used**
  - do jobs really use the requested resources (cores, memory, ...)?
  - is software being run in the correct environment (e.g., used MPI libraries)?
- **detailed statistics** on which software is consuming most of available compute time
  - where to focus optimisation efforts (e.g., `awk` use case with OGRT)
  - can help with targeted benchmarking for new systems

Example analysis of data collected with XALT



top 20 executables

on Darter (NICS)

October'14 - June'15

source:

XALT paper @ HUST'15

https://doi.org/10.1145/2834996.2835000

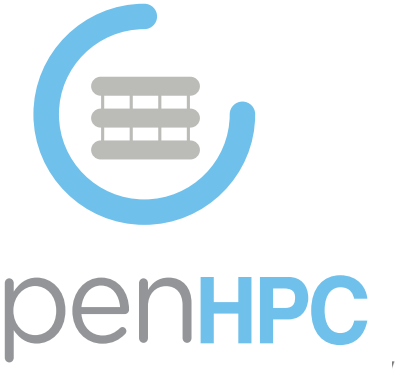
XALT/OGRT: more information

- GitHub: https://github.com/Fahey-McLay/xalt - https://github.com/georg-rath/ogrt
- documentation: http://xalt.readthedocs.io/en/latest - https://github.com/georg-rath/ogrt
- XALT presentations
  - FOSDEM'16: https://archive.fosdem.org/2016/schedule/event/hpc_bigdata_xalt/
  - EUM'17: http://hpcugent.github.io/easybuild/files/EUM17/20170208-5_XALT.pdf
- OGRT presentations:
  - EUM'16: http://goo.gl/zbvChr - http://youtu.be/3l0eJq0nrOU
  - FOSDEM'17: https://fosdem.org/2017/schedule/event/ogrt/
- **joint Birds-of-a-Feather session at ISC'17 on XALT & OGRT**

Birds-eye view of tools & projects

- EasyBuild: getting (scientific) software installed
- Lmod: providing easy access to software installations
- Singularity: containers for HPC
- ClusterShell: executing commands on remote systems
- XALT & OGRT: track which software is being used
- **OpenHPC**

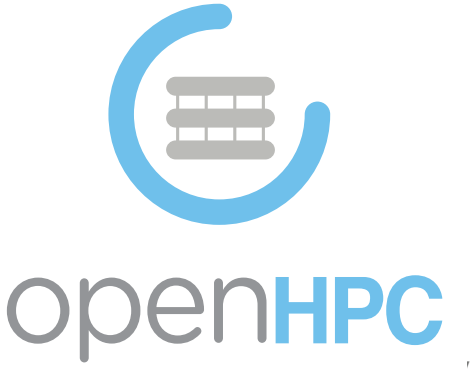
OpenHPC



- **reference collection of open-source HPC software components**
- includes SLURM, Warewulf, *Lmod*, *EasyBuild*, *ClusterShell*, etc.

  (soon also Singularity)
- supports both CentOS & SLES, x86_64 and AARCH64 (currently in tech preview)
- pre-built packages for compilers, MPI, libraries & tools for HPC, etc.
- also includes best practices, guidelines for configuration, etc.
- intends to lower the barrier to entry for HPC
- initial release at SC'15, latest release v1.3.0 (March'17)

OpenHPC: more information



- website: http://openhpc.community
- GitHub repository: https://github.com/openhpc/ohpc
- presentation at FOSDEM'16 (Feb'16)

  OpenHPC: Community Building Blocks for HPC Systems

  https://fosdem.org/2016/schedule/event/hpc_bigdata_openhpc/
- paper at System Professionals Workshop (Nov'16)

  Cluster Computing with OpenHPC

  https://github.com/openhpc/ohpc/files/619162/HPCSYSPROS16_OpenHPC.pdf

Outline

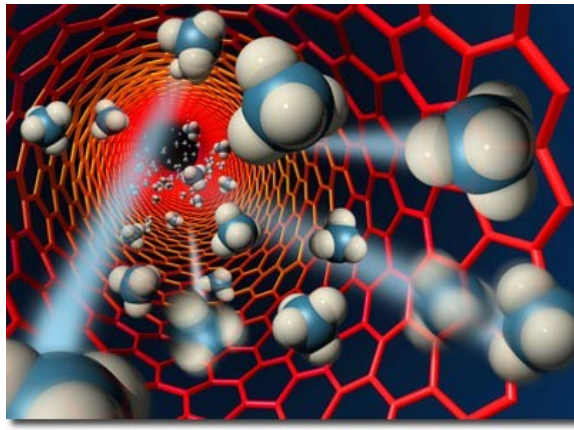
- Getting Scientific Software Installed with EasyBuild & Lmod
  - **Context & motivation for tools & automation**
  - **Introduction to EasyBuild & Lmod**
  - Installation & configuration
  - Usage & workflow
  - Advanced topics

Tasks for HPC user support teams

- resolving problems that occur when using the HPC system(s)
  - login problems, crashing jobs, things that "stopped working", ...
- answering questions, from simple to very technical
- **installing (scientific) software tools/libraries/applications**
- helping users improve their workflow (not necessarily by request)
- training: Linux basics, OpenMP, MPI, Python, etc.
- ~~performance analysis and optimisation of large scientific applications~~
- ~~consultancy services w.r.t. developing scientific software~~

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

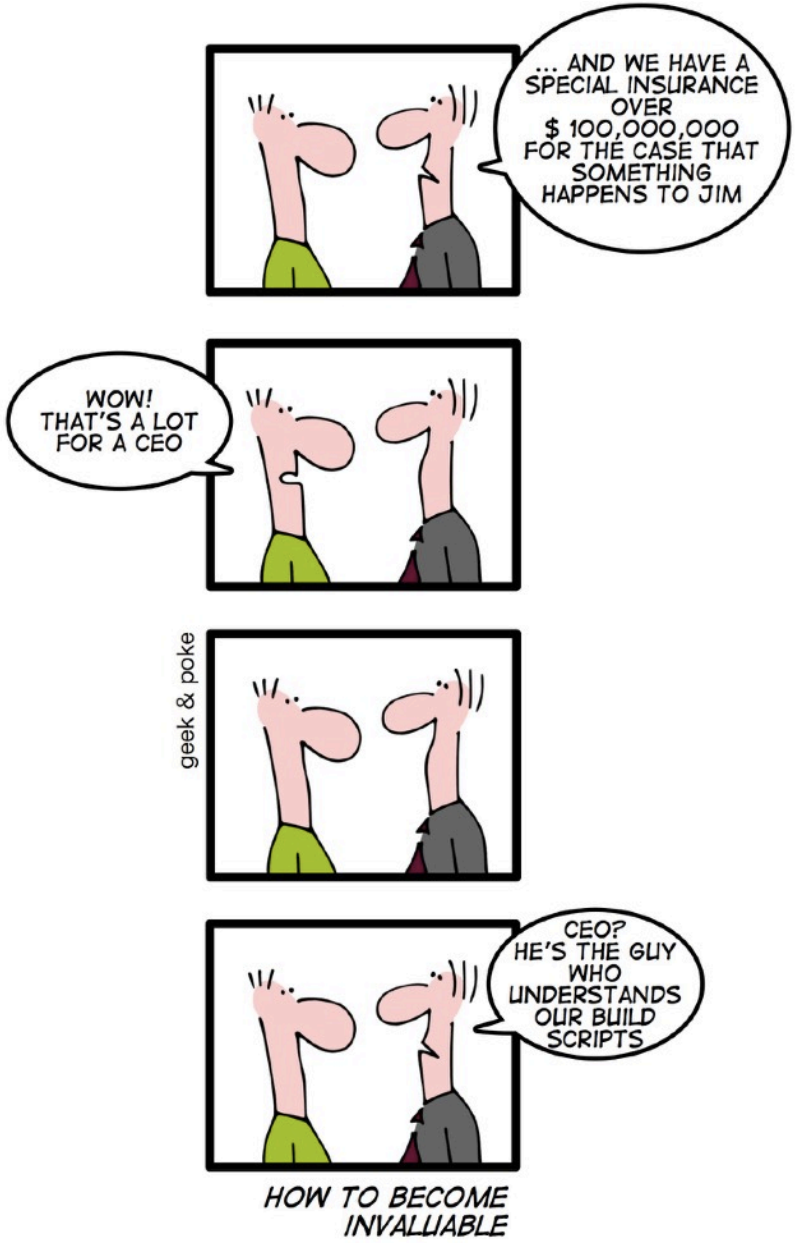
Weather Research and Forecasting Model (http://www.wrf-model.org)

- 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

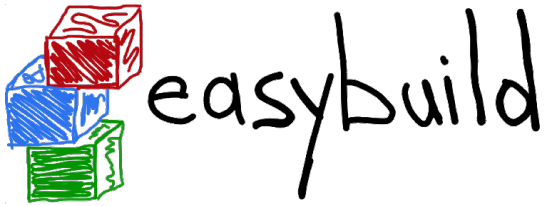
What about existing software installation tools?

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, **> 1,150 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 v3.1.2 (March 20th 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`, it wraps around them
- ***not*** a replacement for your favourite package manager
- ***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 v3.1.2 supports installing **over 1,150 software tools & applications**
  - including CP2K, NAMD, NWChem, OpenFOAM, QuantumESPRESSO, WRF, WPS, ...
  - 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 v3.1.2:

- about 50k LoC of Python (incl. framework, easyblocks, and logging/option parsing support)
- about 15k LoC more for unit/integration test suites
- 1,182 supported software applications (excl. extensions, software versions)
- 25 different toolchain definitions (excl. toolchain versions)
- 181 software-specific easyblocks, 30 generic easyblocks
- 6,718 easyconfig files
  - different software versions, variants, using different toolchains, ...

Providing (easy) access to software installations

- lots of (scientific) software is typically installed on HPC clusters
  - potentially wide range of scientific domains
  - multiple different versions, builds, variants, ...
  - installed with different generations of compilers/libraries
  - usually in a non-standard location (shared filesystem)

- environment needs to be modified to use these software installations
  - `$PATH`, `$LD_LIBRARY_PATH`, etc. + custom software-specific environment variables
- **how can we easily let users access these software installations?**

Environment modules

- well established solution in HPC community
- shell-agnostic 'scripts' to set up environment
- intuitive user interface via `module` command
- used on most HPC systems (>80%[1]), since mid 90's
- Tcl-based environment modules system is ubiquitous, but:
  - last release of Tcl/C implementation (v3.2.10) was Dec 2012
  - pure Tcl version is actively maintained, but (a lot) less used
  - lacks features supported by more recent implementations...

(1) http://hpcugent.github.io/easybuild/files/SC15_BoF_Getting_Scientific_Software_Installed.pdf

Environment modules: user interface

- checking which modules are available is done using `module avail`
- to update the environment for a particular software installation,

  a module should be loaded using `module load`
- an overview of currently loaded modules can be obtained with `module list`
- reversing the changes to the environment can be done with `module unload`
- resetting the environment by unloading all loaded modules can be done

  using `module purge`

Environment modules: user interface

```
$ which gcc
/usr/bin/gcc
$ module avail GCC
---------------------- /apps/modules/all ----------------------
GCC/4.7.4 GCC/4.8.5 GCC/4.9.4 GCC/5.1.0  GCC/5.2.0  GCC/5.3.0
$ module load GCC/5.3.0
$ which gcc
/apps/software/GCC/5.3.0/bin/gcc
```

Lmod: a modern environment modules tool

- website: https://www.tacc.utexas.edu/research-development/tacc-projects/lmod
- documentation: http://lmod.readthedocs.io/
- latest: Lmod v7.4.4 (April 17th 2017) - https://github.com/TACC/Lmod/releases
- implemented in Lua, lead development by dr. Robert McLay (TACC)
- *mostly drop-in replacement for Tcl-based environment modules tool ('Tmod')*
- actively maintained, (very) frequent updates, vibrant & helpful community
- slowly taking over the (HPC) world...

Lmod: feature highlights (1)

- same `module` user interface as Tmod
- consumes module files written in either Tcl or Lua, mixing of both is supported
- module files are **cached** to make user interface more responsive
  - Lmod's *spider cache* must be kept up-to-date! (cron job)
- also provides `ml` shorthand command (`moudle` is (too) easy to mistype)
  - `ml` is the same as `module list`
  - `ml foo` is the same as `module load foo`
  - `ml av` is the same as `module available`
  - `ml -foo` is the same as `module unload foo`

Lmod: feature highlights (2)

- some minor **functional differences with Tmod**:
  - *one name rule*: only one single module can be loaded for a particular software
    - can also be enforced in Tmod via '`conflict`' statements in module files
    - Lmod automatically unloads previously loaded module on name clash (by default)
  - correct ordering of software versions (5.6 is older than 5.10)
  - proper support for hierarchical modules
    - can search through all existing modules via '`module spider`',
      even if they are not (yet) available for loading

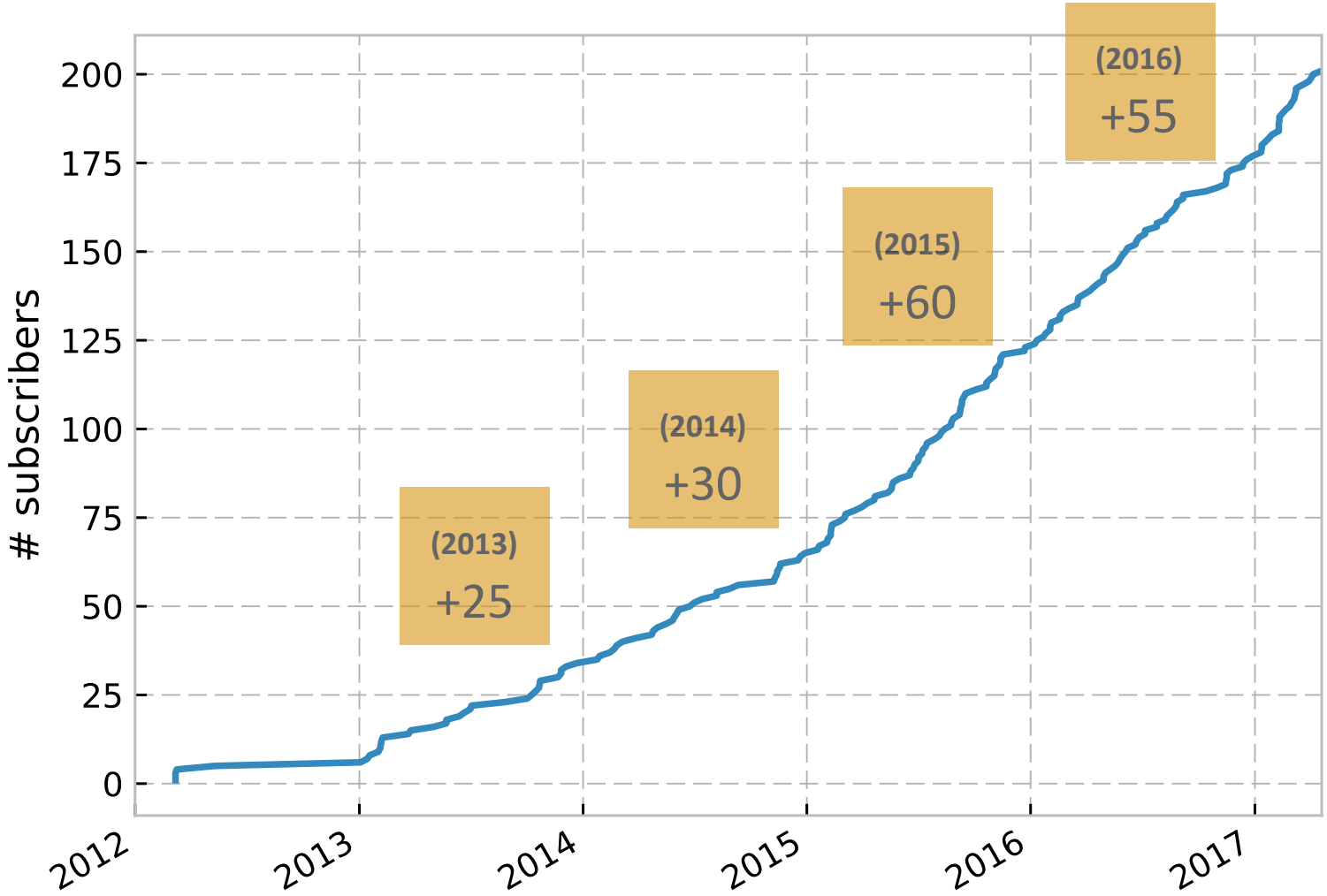
Lmod: feature highlights (3)

- support for additional features, including:
  - defining **module families** (e.g. MPI), only one module can be loaded per family
  - **module properties**, which are shown in output of '`module avail`'
  - ***sticky* modules**, which do not get unloaded on '`module purge`' (unless forced)
  - user-defined **module collections** via '`module save`' and 'module restore'
  - **deprecating modules**, user can get a custom message when a module is loaded
  - **hooks** to customise Lmod's behaviour, e.g., to track which modules get loaded
  - easy **debugging** of Lmod itself via '`-D`' option to `module`/`ml` commands
  - ...

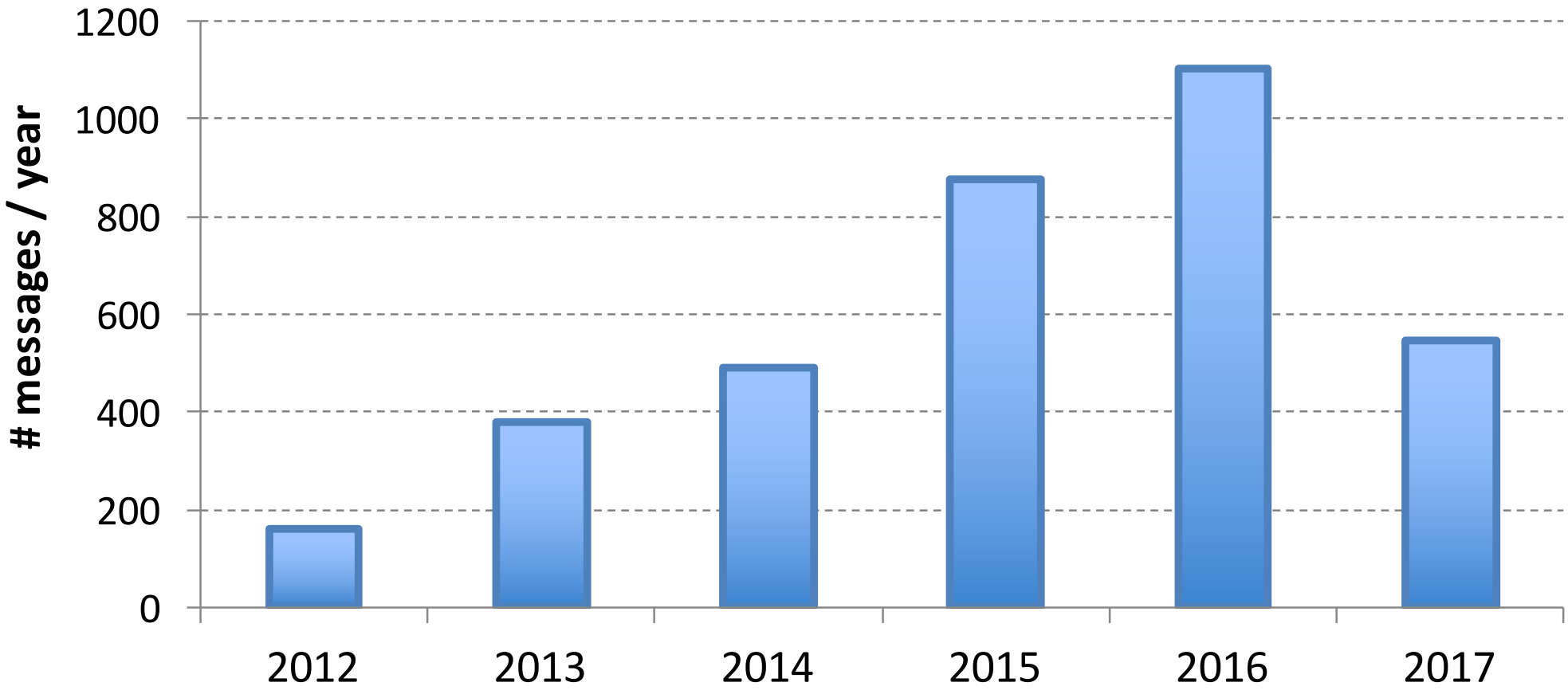
Community aspects

- EasyBuild & Lmod communities have 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:
  - EasyBuild: https://lists.ugent.be/wws/info/easybuild
  - Lmod: https://sourceforge.net/p/lmod/mailman/lmod-users/
- EasyBuild also has active IRC and Slack channels
- users also start contributing: bug reports, feature requests, code contributions, ...

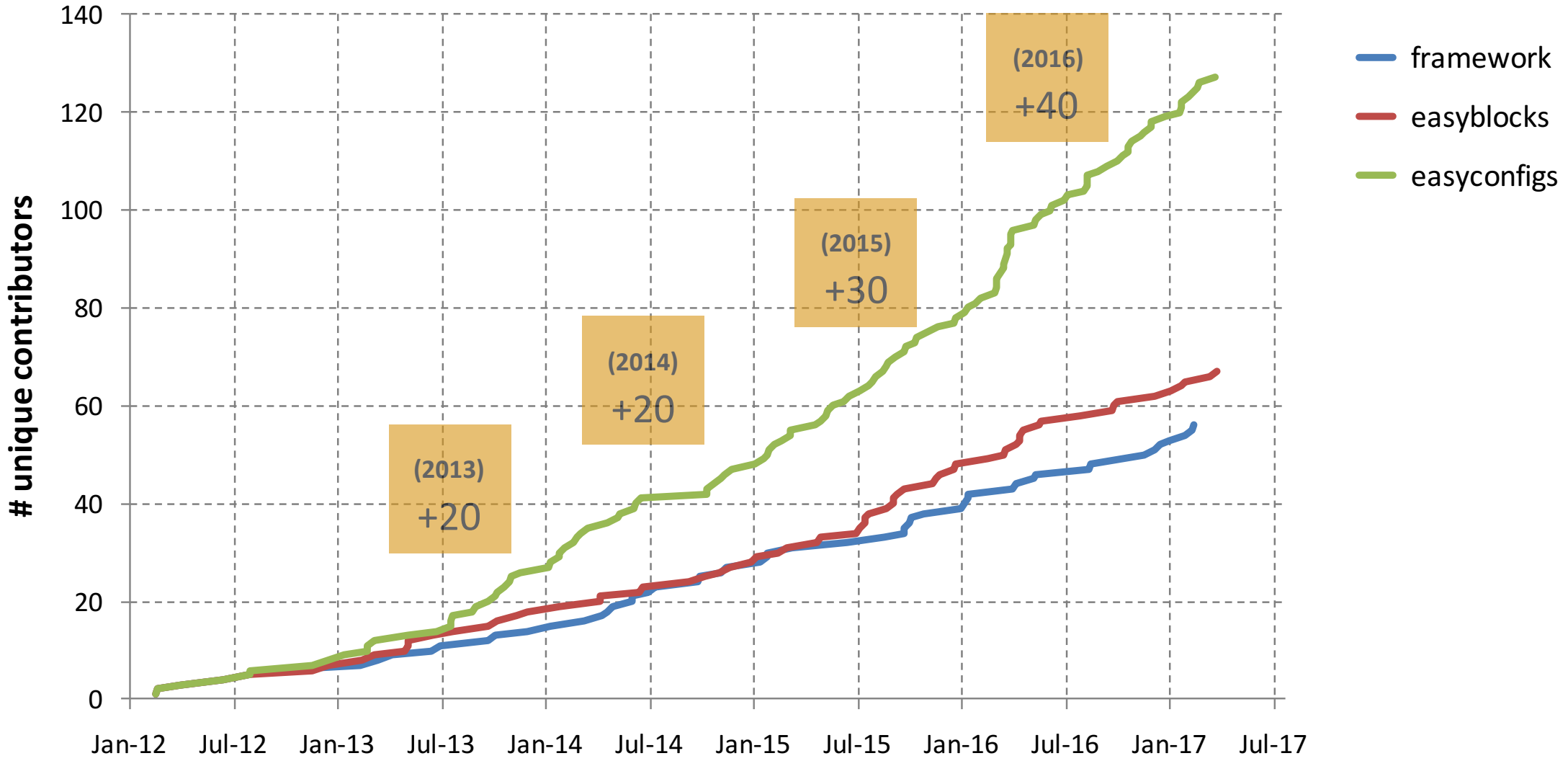
EasyBuild mailing list (subscribers)



EasyBuild mailing list (traffic)



Code contributions to EasyBuild



Outline

- Getting Scientific Software Installed with EasyBuild & Lmod
  - Context & motivation for tools & automation
  - Introduction to EasyBuild & Lmod
  - **Installation & configuration**
  - Usage & workflow
  - Advanced topics

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
  - basic Windows support 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 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
```

Installing Lmod

http://lmod.readthedocs.io/en/latest/030_installing.html
https://github.com/TACC/Lmod/blob/master/INSTALL

- requirements:
  - Lua 5.1 or newer, incl. `luaposix` and `luafilesystem` libraries
  - `luajson` and `luaterm` are also recommended (but not strictly required)
- Lmod installation procedure is standard `configure` - `make` - `make install`
- profile scripts are included, required to define `module`/`ml` commands on login
- **typically installed system-wide as an OS package, with site-specific configuration**
- .spec file for (our) Lmod is available at https://github.com/hpcugent/Lmod-UGent

Configuring Lmod (1)

http://lmod.readthedocs.io/en/latest/155_lmodrc.html
http://lmod.readthedocs.io/en/latest/170_hooks.html

- default Lmod behaviour can be configured at build time using `configure` options
- can also be tweaked at runtime using `$LMOD_*` environment variables
- some aspects are configured via `lmodrc.lua` file
  - location of Lmod spider cache, known module properties, ...
- hooks can be implemented in `SitePackage.lua` file
- **active Lmod configuration can be queried via '`module --config`'**

Configuring Lmod (2)

- some noteworthy configuration options:
  - `--with-redirect=yes`: redirect messages to *stdout* (other than warnings/errors)
  - `--with-disableNameAutoSwap=yes`: produce an error rather than automatically unloading module on name clash (one name rule)
  - `--with-shortTime=<N>`: with large N, avoid building of user spider cache file
  - `--with-pinVersions=yes`: always pin module versions in collections
  - `--with-cachedLoads=yes`: also use spider cache for '`module load`'

Lmod: spider cache

http://lmod.readthedocs.io/en/latest/130_spider_cache.html

- **making Lmod use an up-to-date system spider cache file is *very* important**
- required for responsive '`module`'/'ml'
- location of system spider cache can be customised:
  - via `configure` options: `--with-spiderCacheDir`, ...
  - via active `lmodrc.lua` configuration file (see `scDescriptT` entry)
  - verify configuration via '`ml --config`'
- use `update_lmod_system_cache_files` to update system spider cache via cron
- avoid building of user spider cache by specifying large value to `--with-shortTime`

Outline

- Getting Scientific Software Installed with EasyBuild & Lmod
  - Context & motivation for tools & automation
  - Introduction to EasyBuild & Lmod
  - Installation & configuration
  - **Usage & workflow**
  - Advanced topics

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`:

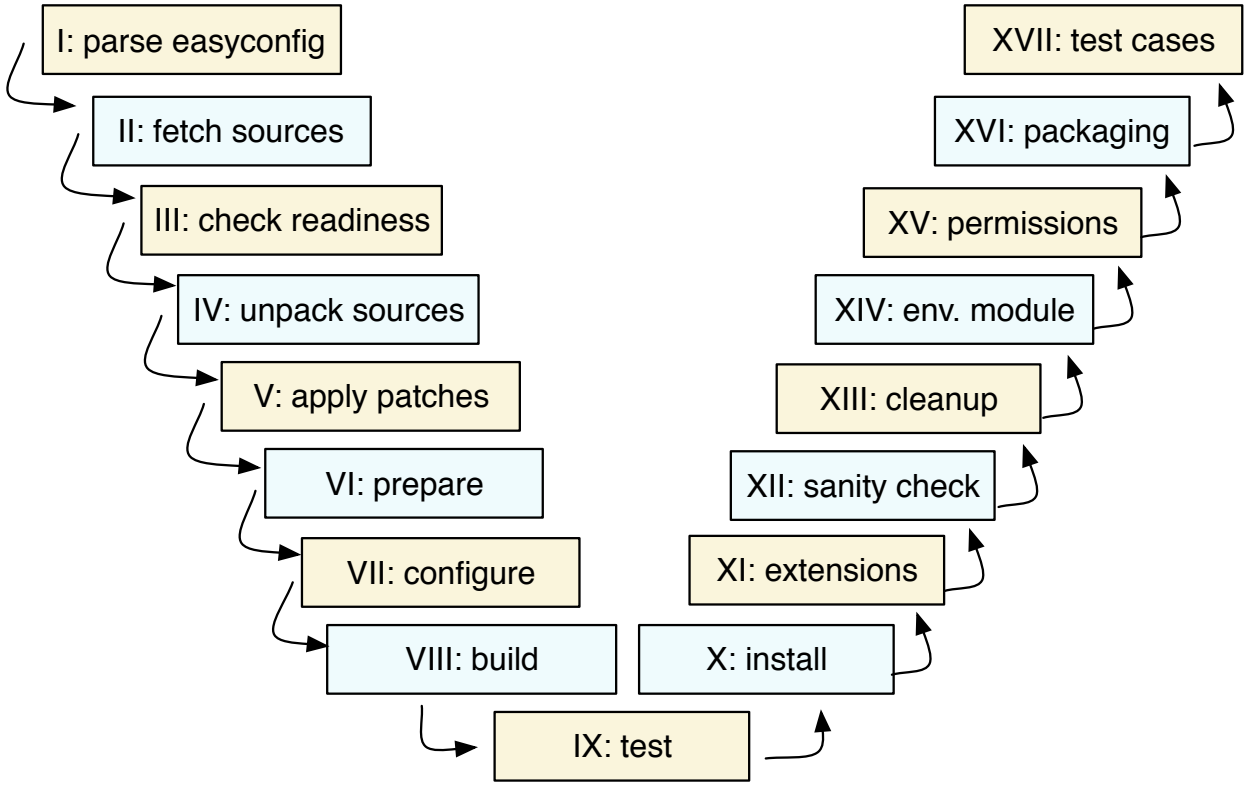
```
eb Python-2.7.13-intel-2017a.eb -D
```

- 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

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'),
]
```

Outline

- Getting Scientific Software Installed with EasyBuild & Lmod
  - Context & motivation for tools & automation
  - Introduction to EasyBuild & Lmod
  - Installation & configuration
  - Typical usage & workflow
  - **Advanced topics**

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

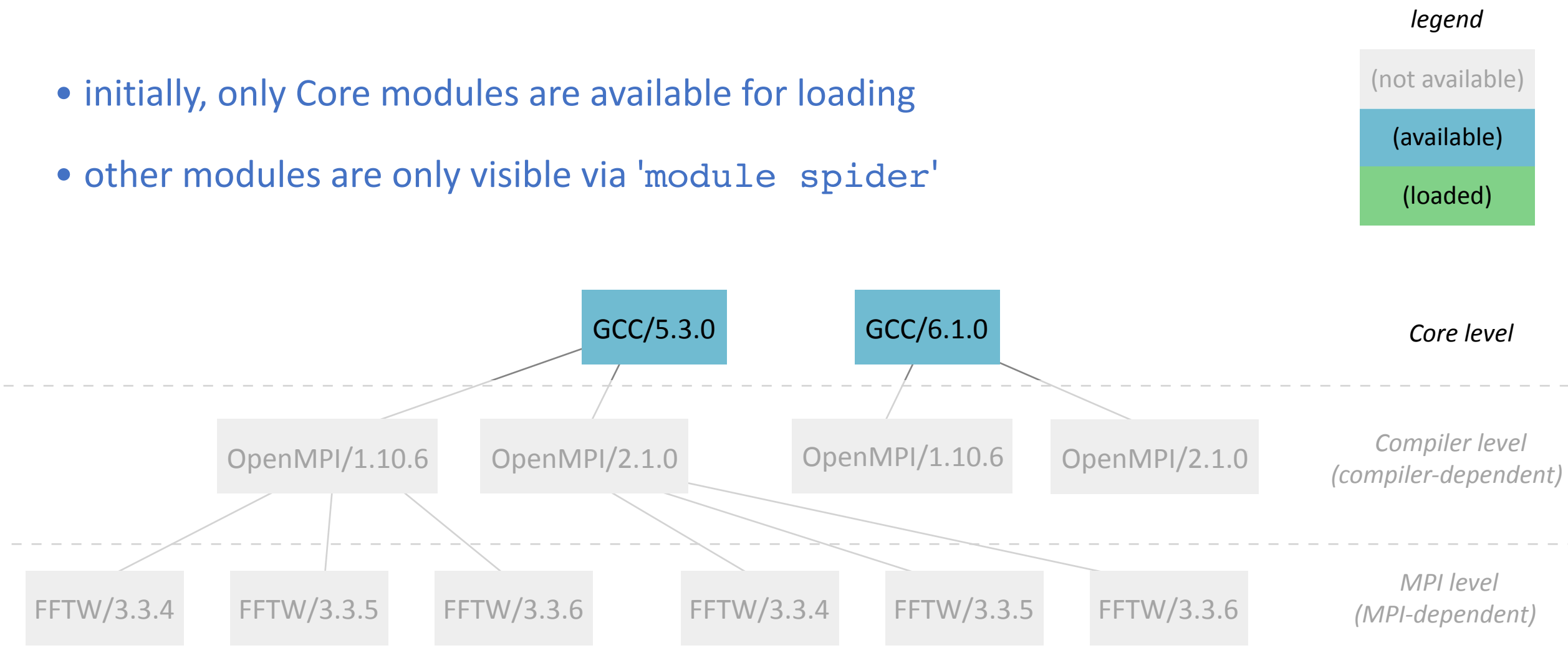
- a couple of different module naming schemes are included in EasyBuild
  - see `--avail-module-naming-schemes`
  - specify active module naming scheme via `--module-naming-scheme`
  - default: `EasyBuildMNS` (`<name>/<version>-<toolchain>-<versionsuffix>`)
- **you can implement your own module naming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

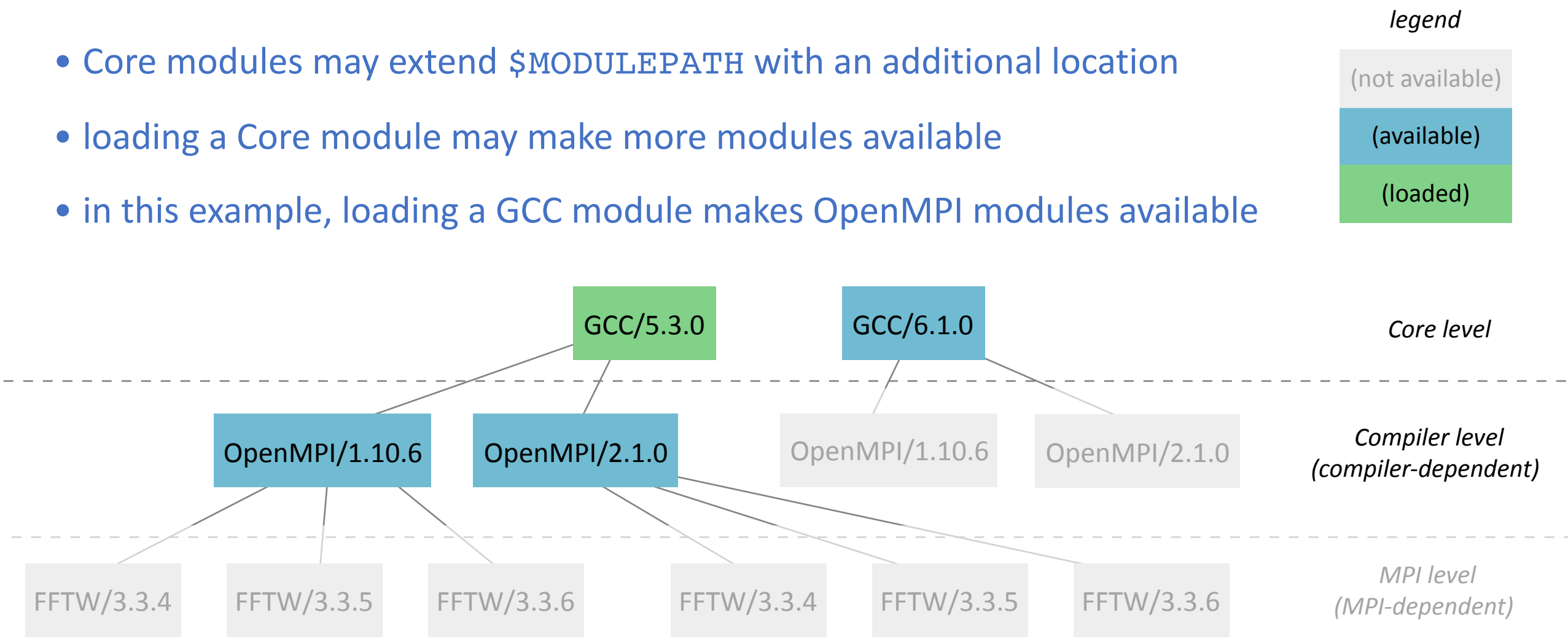
Hierarchical module naming scheme in detail

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



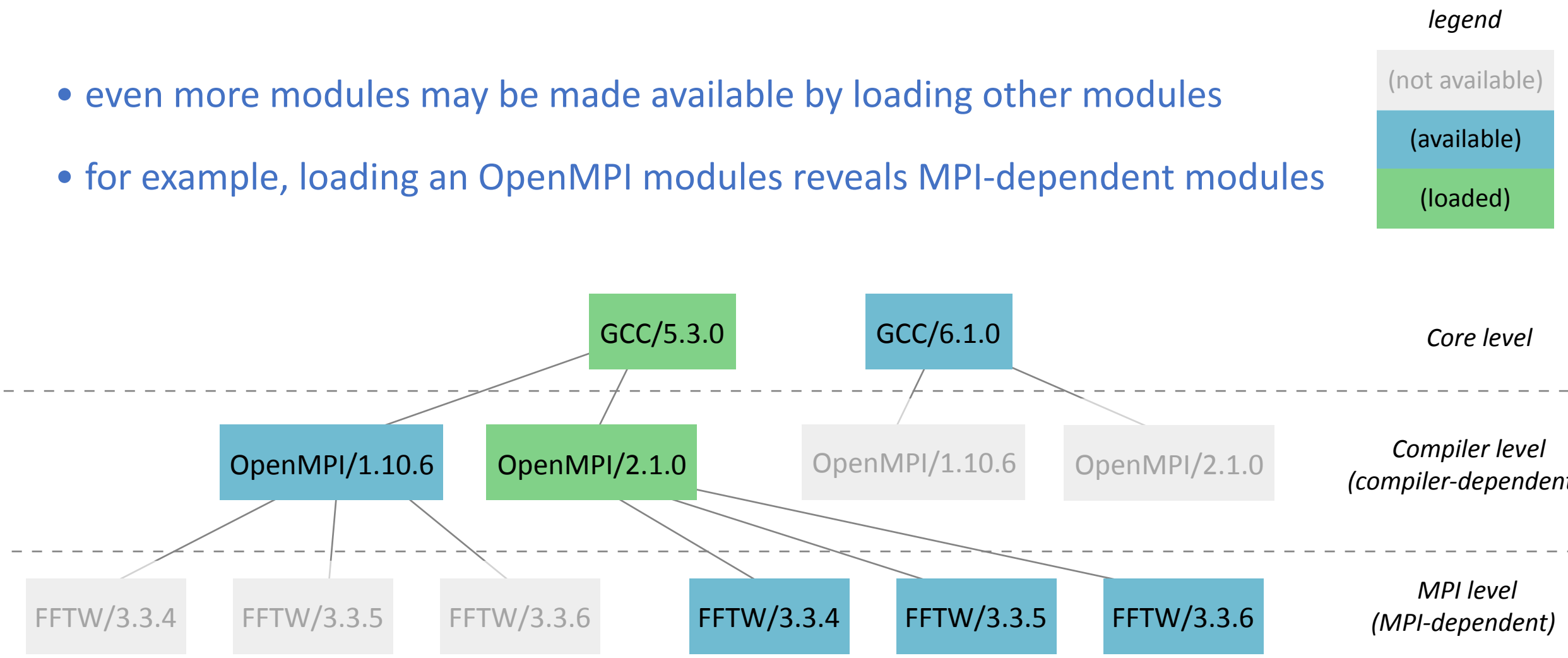
Hierarchical module naming scheme in detail

- 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
- 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 & Lmod

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

Both EasyBuild and Lmod have improved significantly thanks to their communities.

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

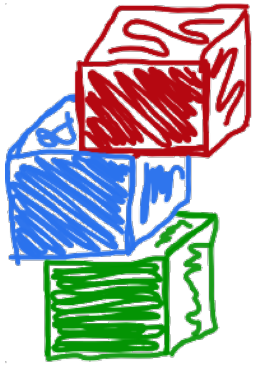
Petar Forai (IMP), Guilherme Peretti-Pezzi (CSCS), Kenneth Hoste (HPC-UGent)

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

Modern Scientific Software Management using EasyBuild & co

PRACE-VI-SEEM 2017 Spring School - System Administration Track

April 25th 2017 - The Cyprus Institute

kenneth.hoste@ugent.be

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

GHENT UNIVERSITY

http://www.ugent.be/hpc

https://www.vscentrum.be

Vlaams Supercomputer Centrum