

Getting Scientific Software Installed Using easybuild

11th CÉCI Scientific Meeting

25 April 2019 - Université Libre de Bruxelles

https://users.ugent.be/~kehoste/ceci_20190425.pdf

Kenneth Hoste

HPC-UGent

email: kenneth.hoste@ugent.be - GitHub: @boegel - Twitter: @kehoste



whoami

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

- Masters & PhD in Computer Science from Ghent University
- PhD topic: machine learning applied to software performance, compilers, ...
- joined HPC-UGent team in October 2010
- main tasks: user support & training, *software installations*
- slowly also became  *easybuild* **lead developer & release manager**
- likes family, beer, loud music, FOSS, helping people, dad jokes, stickers, ...
- doesn't like CMake, SCons, Bazel, TensorFlow, OpenFOAM, ...

HPC-UGent



- part of central IT department of Ghent University (Belgium)
- centralised scientific computing services, training & support
- for researchers of UGent, industry & knowledge institutes
- 6 Tier-2 clusters (> 15k cores in total), ~2 PB shared storage
- 7+1 team members, > 3000 user accounts



Flemish Supercomputer Center (VSC)

(Vlaams Supercomputer Centrum)

<https://www.vscentrum.be>



Vlaanderen
is computing

- virtual center, collaboration between Flemish universities
- funded by Flemish government (FWO)
- VSC user account can be used on each VSC site
 - home & data directories are mounted everywhere
- specialised resources per site (GPUs, SMP, visualisation, ...)
- collaboration on large Tier-1 infrastructure



VSC Tier-1 – BrENIAC (@ KUL)

<https://www.vscentrum.be/en/access-and-infrastructure/tier-1>

Hardware

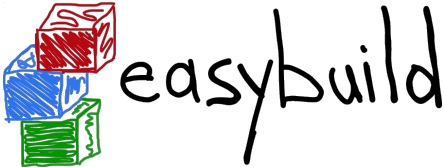
- 580 computing nodes (16,240 cores in total)
 - Two 14-core Intel Xeon processors (Broadwell, E5-2680v4)
 - 128 GiB RAM (435 nodes) or 256 GiB (145 nodes)
- EDR InfiniBand interconnect
 - High bandwidth (11.75 GB/s per direction, per link)
 - Slightly improved latency over FDR
- Storage system
 - Capacity of 634 TB
 - Peak bandwidth of 20 GB/s



Extension is being installed currently, which should bring total compute power to ~1.5 PF

- 408 additional workernodes,
each with 2x Intel Skylake 14-core processors
- double the scratch storage volume

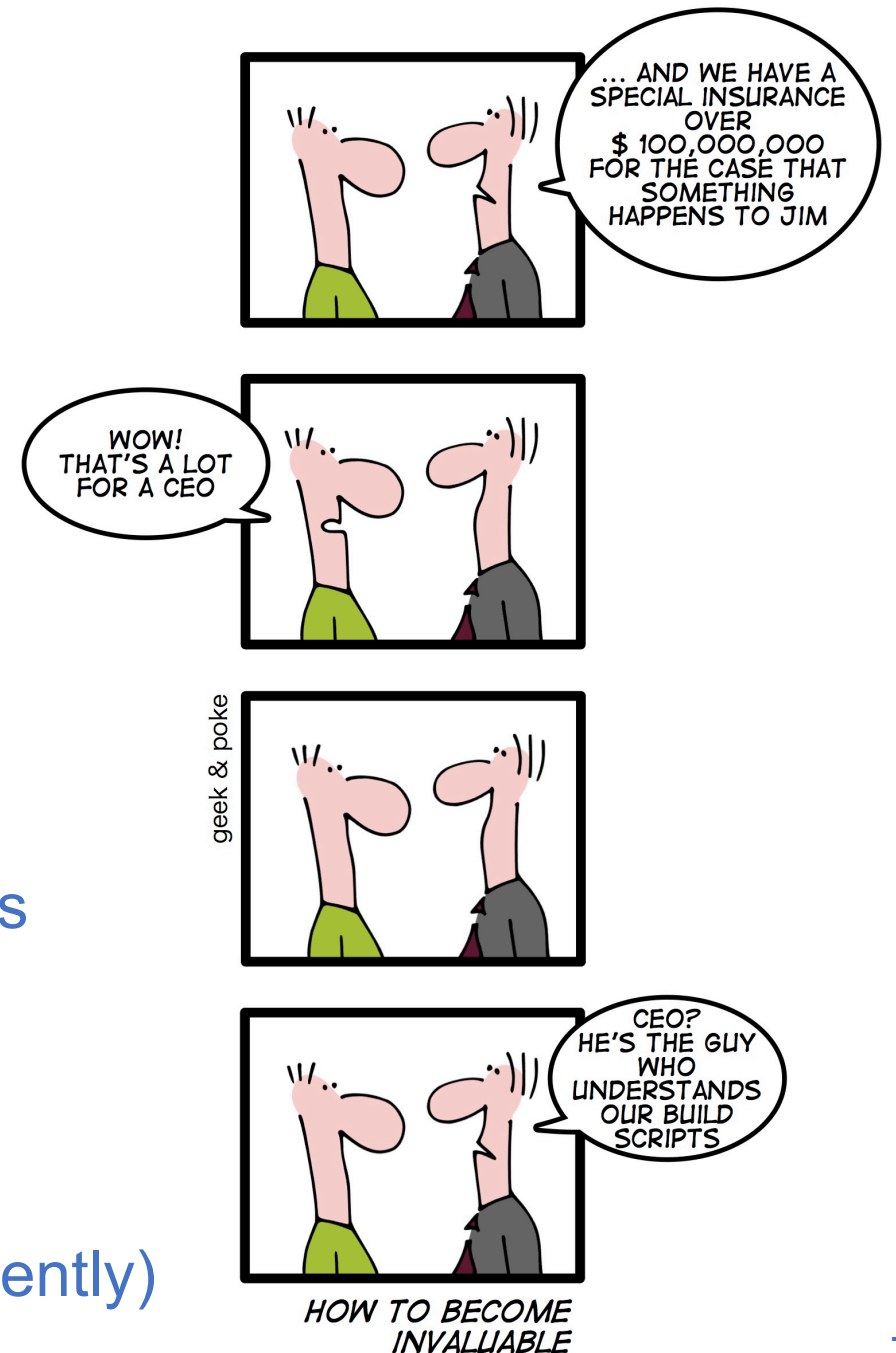
Setting the stage

- Installing scientific software on HPC systems is getting messier
- Current commonly applied "solutions" are mostly workarounds
- 5 tips for making your software difficult/annoying to install !
- How  can help to keep us sane
- Q&A

Getting scientific software installed

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

- ideally built from source (performance is key!)
- tedious, time-consuming, frustrating, sometimes simply not worth the (manual) effort...
- huge burden on researchers & HPC support teams
 - over 25% of support tickets at HPC-UGent, but consumes way more than 1/4th of support time...
- very little collaboration among HPC sites (until recently)



Common issues with scientific software

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

Scientists are (typically) not software developers or system administrators (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 systems (or broken ones)
- "creative" software versioning (or no versions at all)
- dependency hell on steroids
- interactive installation scripts
- hardcoded parameters (compilers, libraries, paths, ...)
- poor/outdated/missing/incorrect documentation



Prime example: TensorFlow



popular open-source software for Deep Learning (<https://www.tensorflow.org>)

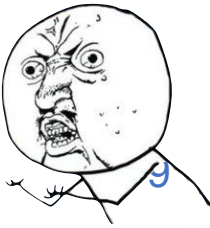
- auto-installs most dependencies, but not all... (Python, CUDA, ...)
- 'configure' script is a custom interactive script
 - silent configuration possible, but only if you *know* which `$TF_*` environment variables to set!
- uses.  Bazel (<http://bazel.io>) as build tool (there is was a contributed CMake alternative...)
- *resets environment*, may result in unsetting important env. variables (e.g., `$PYTHONPATH`)

- quite different from other build tools; e.g. to build TensorFlow:

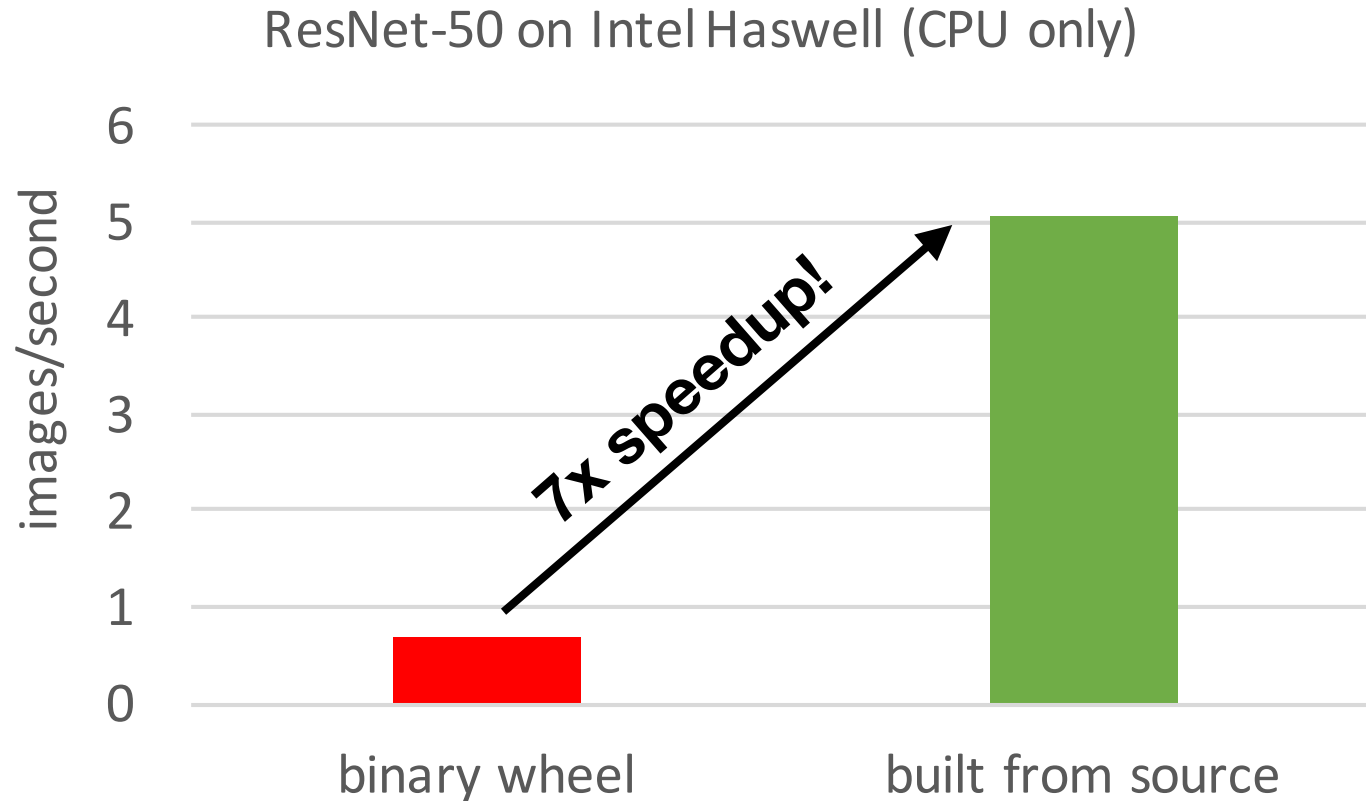
```
bazel build --config=opt //tensorflow/tools/pip_package:build_pip_package
```

No, that's not a typo...

- `--config=opt`, `-c opt` and `-copt=...` \leq these 3 options all mean different things...
- installation via 'pip install' of locally built Python wheel file (.whl)



"pip install tensorflow" works fine for me...



(disclaimer: old version of TensorFlow (1.4.x?), but point is still valid)

- installing pre-built generically optimised binaries is easy
- ... but **you may be paying a *significant* prize w.r.t. performance**

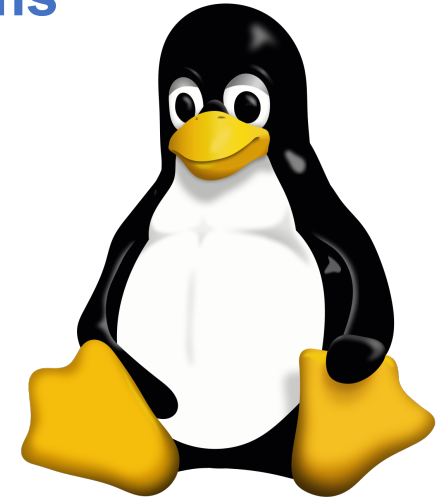
Installing (scientific) software on HPC systems



- key aspects:
 - large-scale **multi-user** system (100s to 1000s of active users)
 - **wide variety of users**, scientific domains, use cases, ...
 - results in **huge software collection** to (centrally) maintain & update
 - **different hardware architectures** (old & new generations)
 - important **system libraries** (IB network, GPU, ...) should also be taken into account
 - **conflicting user requirements/expectations** make things even more complex
 - stable software versions vs bleeding edge, Python 2 vs 3, ...
 - **performance is key !**
- **user-friendly interface should be provided to users** to leverage installed software
 - well-established solution is the "environment modules" tool (**Lmod** is quite popular now)
 - `module avail example, module load example/1.2.3, module list, ...`

What about existing software installation tools? (1/2)

- **traditional package managers in Linux(-like) operating systems**
 - *yum* to install RPMs (Red Hat derivatives)
 - *apt* to install .deb files (Debian & derivatives)
 - *Homebrew* (macOS/Linux)
 - *Portage* (Linux), *pkgsrc* (*nix), ...
- **problems in context of scientific software & HPC:**
 - not well suited to idiosyncrasies of scientific software
 - not aware (enough) of MPI/BLAS/LAPACK/GPUs, other compilers (Intel, PGI, ...)
 - bad fit for multi-user HPC systems in general
 - little support for having multiple versions installed side-by-side compilers
 - strong focus on generically optimised binaries (portability is important to limit effort)



What about existing software installation tools? (2/2)

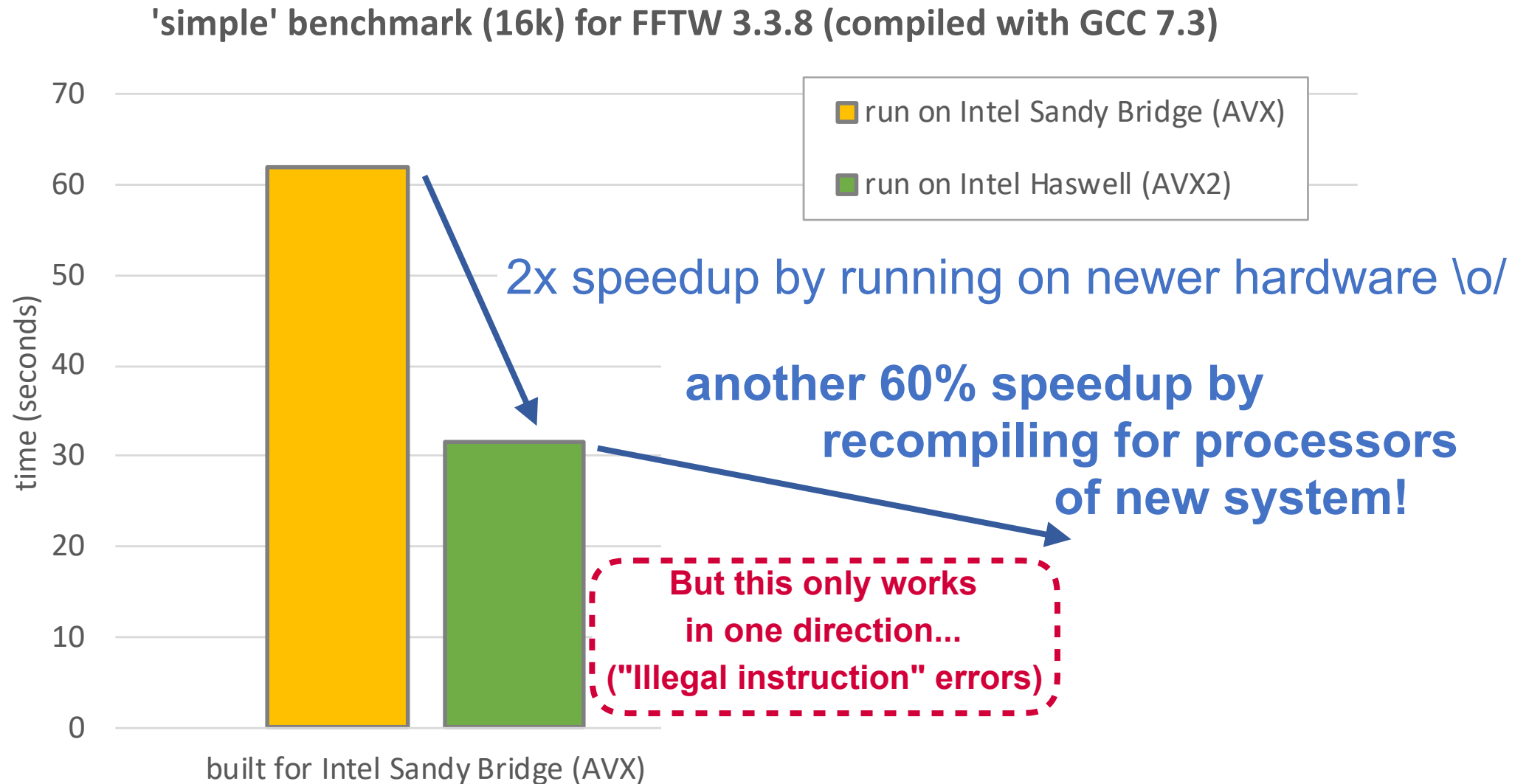
 **CONDA** (installation tool for Anaconda distribution)

- very popular tool for installing scientific software, but mainly **targeted to workstations**
- **not well suited for centrally managing software** installations on multi-user HPC systems
- installing software with conda *usually* results in running **generically optimised binaries**
- hard to combine with software installed in other ways (e.g. centrally provided modules)

Containers (Docker/Singularity)  

- **blatant lack of attention to optimising for underlying hardware** ("mobility of compute")
- **integration with system resources** (MPI, CUDA, ...) is **still a problem**
- **someone still needs to be build/maintain/update the container image you need/want !**
- in my opinion, containers are a *symptom*, not a cure...

Current practice vs "performance is key" in HPC



(FFTW 3.3.8 installed in Singularity container, built from source)

So *why* did tools like conda & containers become so popular?

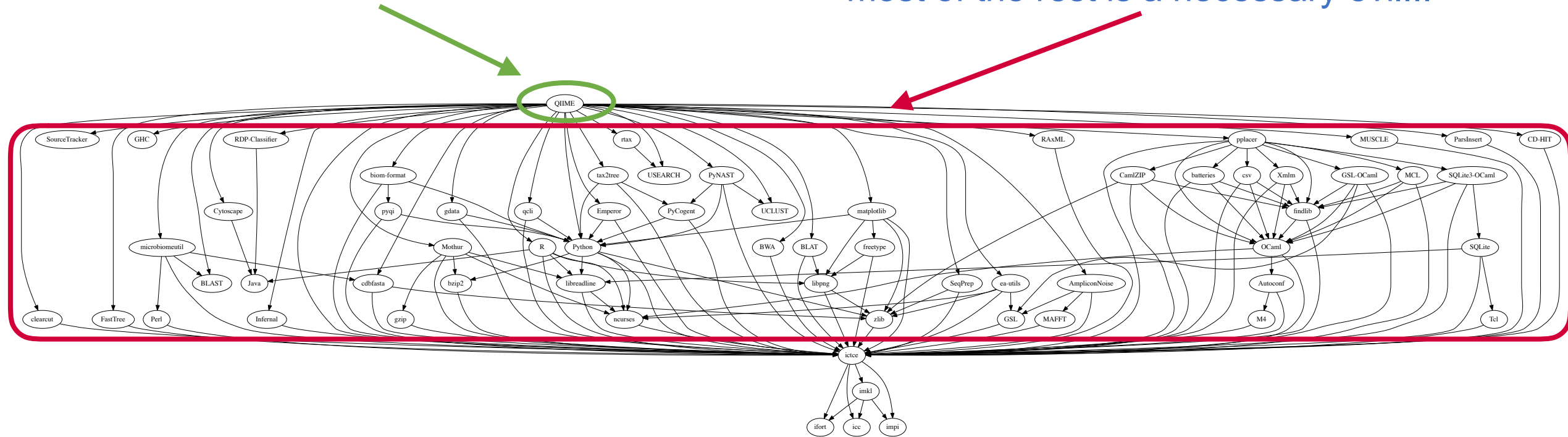
- **installing scientific software (correctly/efficiently) is becoming more complex**
 - dependency hell is getting worse (**fast**)
 - growing variety in target platforms (x86_64, GPUs, ARM, POWER, ...)
 - specifically targeting host system is becoming *more* important for performance
- **scientists using/developing scientific software lack expertise in**
 - (sensible) software release management
 - picking the right tool for the job (tip: not a custom Perl script named 'configure')
 - diagnosing & fixing compilation/installation problems
- some popular programming languages & applications are not great examples...
- also, **people are lazy/impatient** (and StackOverflow doesn't always have an easy answer)



Dependency hell in scientific software

this is the part we actually care about

most of the rest is a necessary evil...

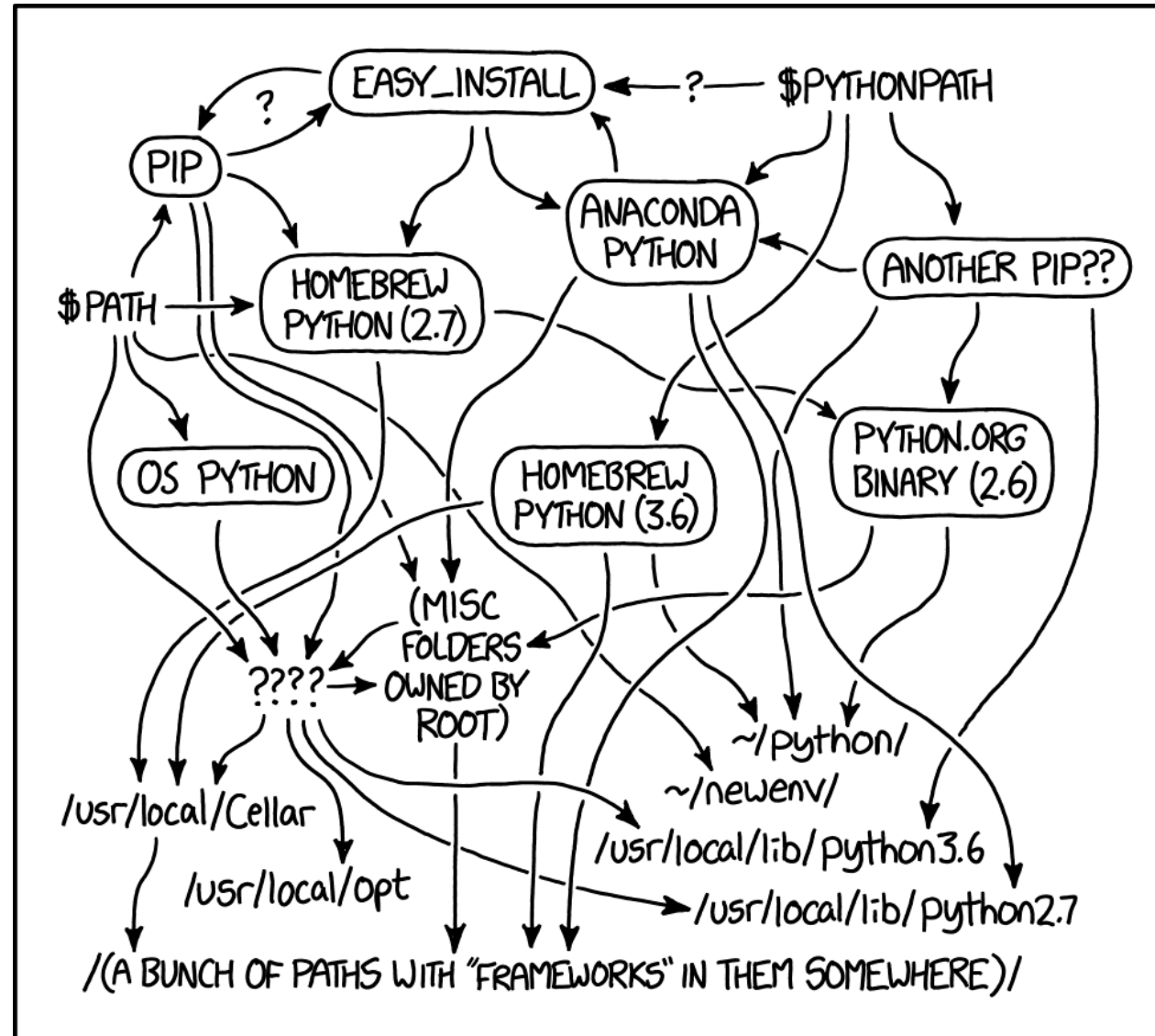


dependency graph for (old version) of QIIME (<http://qiime.org>)

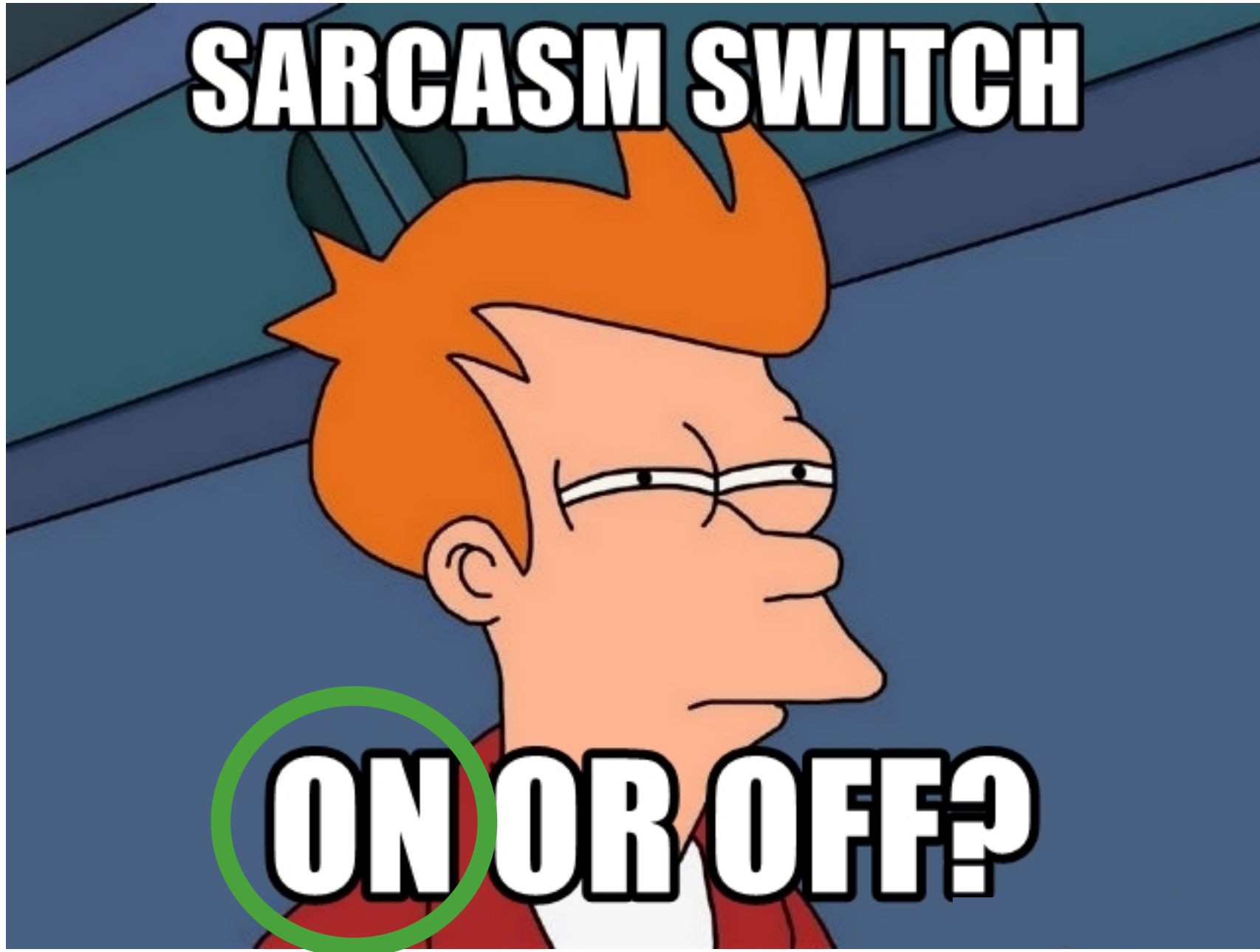


the
Python
mess...

<https://xkcd.com/1987>



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.



Tips & tricks to make your software difficult to install

- But... why should my software be difficult to install?!
- Implementing scientific software is fun, but supporting it is not...
- People need to be able to install your software before they can use it.



Tip #1: creative software naming

- try to come up with a silly/annoying/stupid software name
- goal: cause confusion
- examples:
 - hard to Google: R, HOME, ROOT
 - weird capitalisation: MaCH, netMHCIIPan, NIfTI, SHRiMP, StaMPS
 - special characters: Open|Speedshop, QuadProg++, X!Tandem, X!!Tandem
 - implying things: EasyBuild, Free*, Open*
 - cultural aspects: Spack (slang for "idiot" in the UK)

Tip #2: creative software versioning

- try to make the versions as useless as possible, remove all implied meaning...
- do not use the established practice of semantic versioning (semver.org)
- only pretend you are using semantic versioning (cfr. Boost)
- calendar versioning (calver.org): 20190425 (or even better: "Tue_04_25_2019")
- switch versioning scheme every now and then (for no good reason)
- re-release under same version multiple times, after making some changes
- actively remove old versions, pretend they've never existed (cfr. Bioconductor)
- don't define any versions at all (why would you)
- use commit IDs as versions, they contain numbers too! (example: 6e2bf9a)

Tip #3: aim for dependency hell

- try to use a lot of dependencies
- prefer using dependencies that:
 - have a lot of dependencies
 - are difficult to install by themselves (make people cry!)
 - use different programming languages (combine R, Python, Perl, Haskell, OCaml, ...)
- include the code of some dependencies in your code repository
- even better: ship pre-compiled binaries for (some) of your required dependency
- example success case: left-pad Javascript module


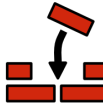

https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos

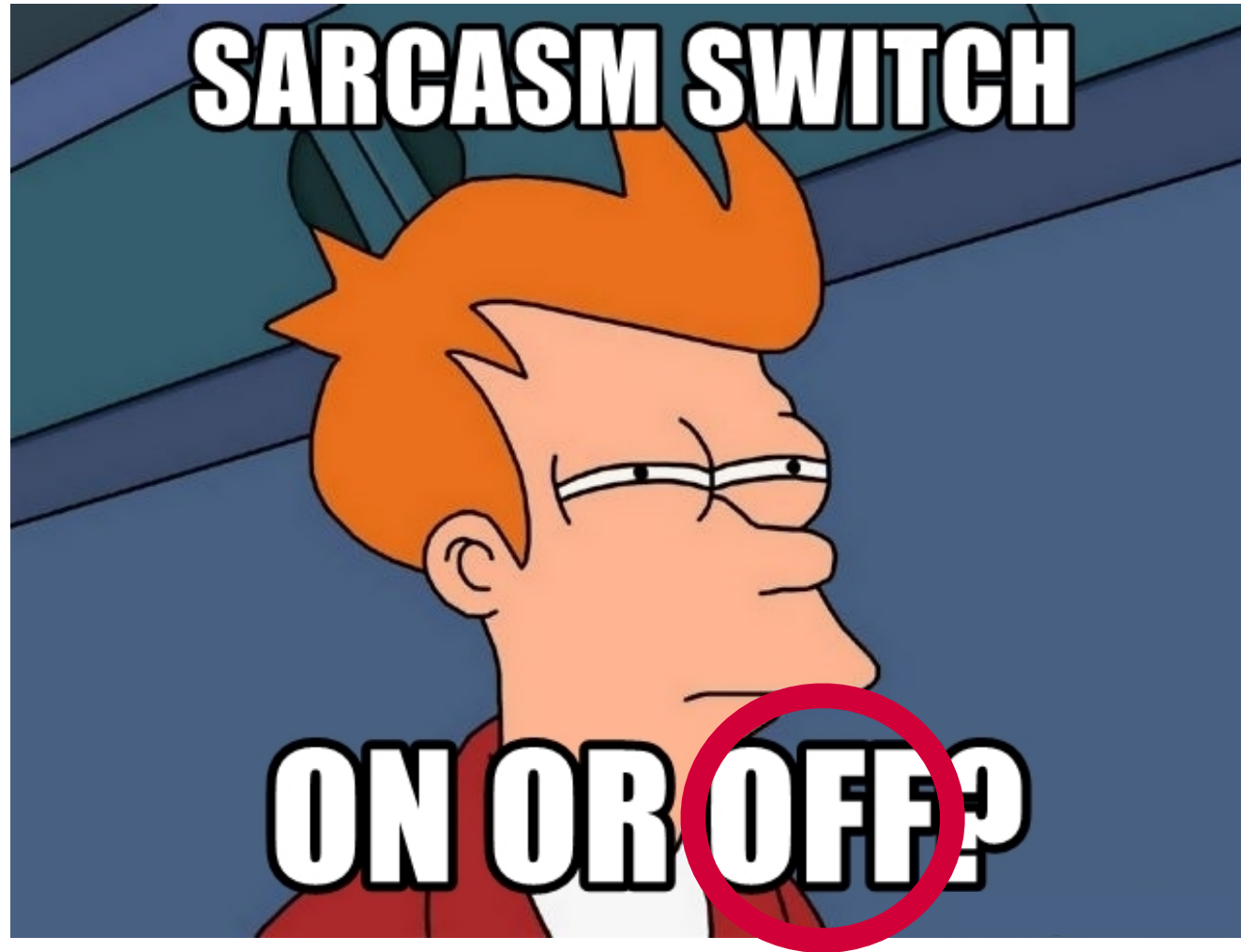


Tip #4: hardcode as much as possible

- **hardcode (locations of) compilers, tools, libraries**
 - `/usr/bin/gcc`
 - `-I/usr/include/... -L/usr/lib/...`
 - `--prefix /home/yourusername/`
- use 'sudo' in your installation scripts, it's there for a reason!
- bonus points if you make it difficult to change the hardcoded values
 - spread them around in different files/scripts as much as you can
 - make compilation fail with cryptic error messages if something is changed

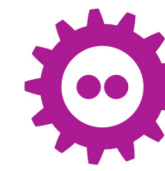
Tip #5: choose your tools wisely

- prefer using build tools that most people are not familiar with (not 'make')
- try to use build tools with funky behaviour:
 - hard to debug/diagnose build problems (let alone fixing them)  **CMake**
Cross-platform Make
 - building in an isolated environment  **SCONS**  **Bazel**
- even better: use a self-written (interactive!) installation script
- use a 'fun' or new programming language: C++17 (use templates!), Julia, Nim
- do not release via standard mechanisms like PyPI (Python), CPAN (Perl), CRAN (R)



 TensorFlow score: 3+ out of 5!
(3 out of 5 tips taken into account, in very creative ways!)

Do you want more?



FOSDEM 2018

How to make package managers cry

https://archive.fosdem.org/2018/schedule/event/how_to_make_package_managers_cry

<https://www.youtube.com/watch?v=NSemlYagjIU>

- me **venting ~7 years of frustration** with getting scientific software installed in HPC context
- TensorFlow & Bazel as main motivators
- **sarcastic tone for dramatic effect** (it worked!)
- lots of feedback (and ideas for an extended version of the talk), **clearly hit a nerve...**
- others discussing similar points:
 - *"Software disenchantment"*, blog post by Nikita Prokopov (Sept 2018)
<http://tonsky.me/blog/disenchantment>
 - *"Don't package your libraries, write packageable libraries!"*, talk at CppCon 2018
by Robert Schumacher (Microsoft) - <https://www.youtube.com/watch?v=sBP17HQAQjk>





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

- **framework for installing scientific software** (*built from source when possible*)
- project started in-house at HPC-UGent in 2009, released publicly in 2012
- strong focus on Linux & HPC systems (and hence also performance)
- **by default builds/optimises specifically for host architecture**
- implemented in Python 2, lead development by HPC-UGent
- available under GPLv2 license via PyPI, GitHub
- supports different compilers & MPI libraries, x86_64/ARM/POWER, ...
- active & helpful worldwide community

Supported software



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

- latest EasyBuild (v3.9.0) supports installing **over 1,700 different software packages**
 - including CP2K, NAMD, NWChem, OpenFOAM, TensorFlow, WRF, ...
 - also a lot of bioinformatics software is supported out of the box
 - + ~1,000 extensions: Python packages, R libraries, Perl modules, X11 libraries, ...
 - built from source when possible, optimised by host architecture by default...
- diverse toolchain support:
 - compilers: GCC, Intel, Clang, PGI, IBM XL, Cray, CUDA
 - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, Cray MPI, ...
 - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ScaLAPACK, BLIS, Cray LibSci, ...

Installing TensorFlow from source with one command...



```
$ eb TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-UlTkEI.log
== processing EasyBuild easyconfig TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb
== building and installing TensorFlow/1.13.1-foss-2018b-Python-3.6.6...
== 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/Tensor...
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-GyvPHx/easybuild-UlTkEI.log* have been removed.
== Temporary directory /tmp/eb-GyvPHx has been removed.
```

What easybuild is *not*

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

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

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

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

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

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

What easybuild is

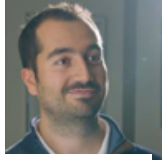
- a **uniform interface** that wraps around software installation procedures
- a huge **time-saver**, by automating tedious/boring/repetitive tasks
- a way to provide a **consistent software stack** to your users
- an **expert system** for software installation on HPC systems
- a **platform for collaboration**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



- EasyBuild community has been growing rapidly the last couple of years
- **hundreds of HPC sites and companies worldwide, incl. JSC, CSCS, SURFsara, Pfizer, ...**
- very welcoming & supportive to newcomers

easybuild maintainers

Since early 2019, there are **11 EasyBuild maintainers**.



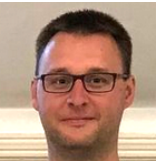
Damian Alvarez - @damianam
(Jülich Supercomputing Centre)



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



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



Kenneth Hoste - @boegel
(HPC-UGent)



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



Samuel Moors - @smoors
(Free University of Brussels)



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



Bart Oldeman - @bartoldeman
(ComputeCanada)



Ward Poelmans - @wpoely86
(Free University of Brussels)



Åke Sandgren - @akesandgren
(Umeå University, Sweden)




Davide Vanzo - @vanzod
(Vanderbilt University)

More information on

- **website:** <https://easybuilders.github.io/easybuild>
- **documentation:** <https://easybuild.readthedocs.io>
- **mailing list:** easybuild@lists.ugent.be (subscribe via <https://lists.ugent.be/www/subscribe/easybuild>)
- **Slack channel:** <https://easybuild.slack.com>
 - self-request invite via <https://easybuild-slack.herokuapp.com>
- **YouTube channel:** <https://www.youtube.com/channel/UCqPyXwACj3sjtOho7m4haVA>

Conclusions

- getting scientific software installed on HPC systems is not all rainbows and pink fluffy unicorns (there is a lot of shit too)
- ... and it's actually getting worse in my experience
- the P in HPC (performance) is being ignored/overlooked more often
- scientists often lack necessary expertise to produce software that follows best practices w.r.t. installation & releasing
- popular tools like **CONDA** and containers are a *workaround*, not a solution
- tools like  **easybuild** are indispensable for HPC support teams & users



Getting Scientific Software Installed Using easybuild

11th CÉCI Scientific Meeting

25 April 2019 - Université Libre de Bruxelles

https://users.ugent.be/~kehoste/ceci_20190425.pdf

Kenneth Hoste

HPC-UGent

email: kenneth.hoste@ugent.be - GitHub: @boegel - Twitter: @kehoste