



easybuild


building software with ease

EasyBuild workshop @ Jülich Supercomputer Centre
October 21st 2014

*Kenneth Hoste - kenneth.hoste@ugent.be
easybuild@lists.ugent.be*



HPC-UGent: in a nutshell

- ▶ HPC team at central IT dept. of Ghent University (Belgium)
 - ▶ 9 team members: 1 manager, ~3 user support, ~5 sysadmin
 - ▶ 6 Tier2 clusters + one Tier1 (8.5k cores), ~1k servers in total
 - ▶ ~1.5k user accounts, all research domains
 - ▶ tasks incl. hardware, system administration, user support/training, ...
- ▶ member of Flemish Supercomputer Centre (VSC) 
 - ▶ virtual centre, collaboration between Flemish university associations



“Please install this software on the cluster?”

Scientists focus on the *science* of the software they produce, not on build procedure, portability, ...

This makes building/installing (lots of) scientific software painful: *very time-consuming, error-prone, hard to get right, ...*

Common issues:

- ▶ non-standard build tools
- ▶ incomplete build procedure, e.g. no install step
- ▶ interactive scripts
- ▶ hardcoded parameters
- ▶ poor/outdated documentation
- ▶ ...



Existing tools are not what we require

Standard packaging solutions (RPM, .deb) are not a good fit.

- ▶ building *from source* is preferred in an HPC context
- ▶ packaging scientific software requires huge amounts of effort
- ▶ packaging formats (e.g. .spec) don't fit peculiarities well
- ▶ collection of build scripts is hard to maintain (by ourselves)

Lots of duplication of work across HPC sites!

Some solutions are (too) OS-dependent:
Portage (Gentoo), Homebrew, ...

Others are software-specific:
Dorsal (DOLFIN), gmckpack (ALADIN), ...

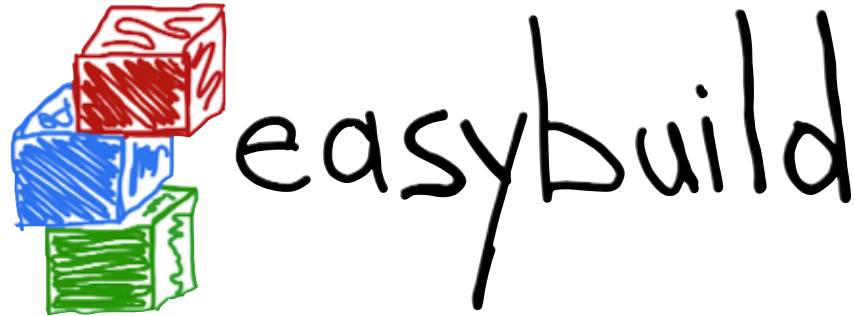


Our build tool wish list

- ▶ a **flexible framework** for building/installing (scientific) software
- ▶ fully **automates** software builds
- ▶ allows for **reproducible** builds
- ▶ supports **co-existence** of versions/builds
- ▶ enables **sharing** with the HPC community (double-edged sword!)
- ▶ **automagic dependency** resolution



EasyBuild: building software with ease



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

EasyBuild is a *software build and installation framework*.

- ▶ written in **Python 2**
- ▶ started in 2009, in-house for ~2.5 years, **GPLv2** since Apr'12
- ▶ **stable API** w/ EasyBuild v1.0 (Nov'12), latest is v1.15.2 (Oct'14)
- ▶ continuously enhanced and extended, thoroughly tested
- ▶ *release early, release often strategy* (major version every 4-6 weeks)
- ▶ development is highly **community-driven**

Requirements

- **Linux x86 / OS X**
- used daily on Scientific Linux (Red Hat-based)
- also tested on Fedora, Debian, Ubuntu, CentOS, SLES, ...
- some known issues on OS X, *focus is on Linux (HPC)*
- no Windows support (and none planned for now)
- **Python** 2.4 or more recent 2.x (no Python 3 support yet)
- modules tool: **Tcl(/C) environment modules** or **Lmod**
- (system C/C++ compiler to bootstrap a GCC toolchain)

Key features

- build & install software **fully autonomously**
also interactive installers, code patching, generating module file, ...
- easily **configurable**: config file/environment/command line
including aspects like module naming scheme
- thorough **logging** and archiving
entire build process is logged thoroughly, logs stored in install dir;
easyconfig file used for build is archived (install dir + file/svn/git repo)
- automatic **dependency resolution**
build entire software stack with a single command, using `--robot`
- building software in **parallel**
e.g., on a (PBS) cluster, using `--job`
- comprehensive **testing**: unit tests, regression testing
- thriving, growing **community**

'Quick' demo for the impatient

```
eb HPL-2.0-goolf-1.4.10-no-OFED.eb --robot
```

- **downloads** all required sources (best effort)
- **builds/installs** *goolf* toolchain (be patient) + HPL with it
goolf: GCC, OpenMPI, LAPACK, OpenBLAS, FFTW, ScaLAPACK
- **generates module file** for each software package
- default: source/build/install dir in `$HOME/.local/easybuild`
can be changed by configuring EasyBuild differently

Some terminology

■ **framework**

- core of EasyBuild: Python packages & modules, *eb* wrapper script
- provides (lots of) supporting functionality

■ **easyblock**

- Python module implementing a particular build procedure
- can be *generic* (e.g., *make* & *cp*) or *software-specific* (e.g., *WRF*)
- talks to framework API, can be viewed as a ‘plugin’

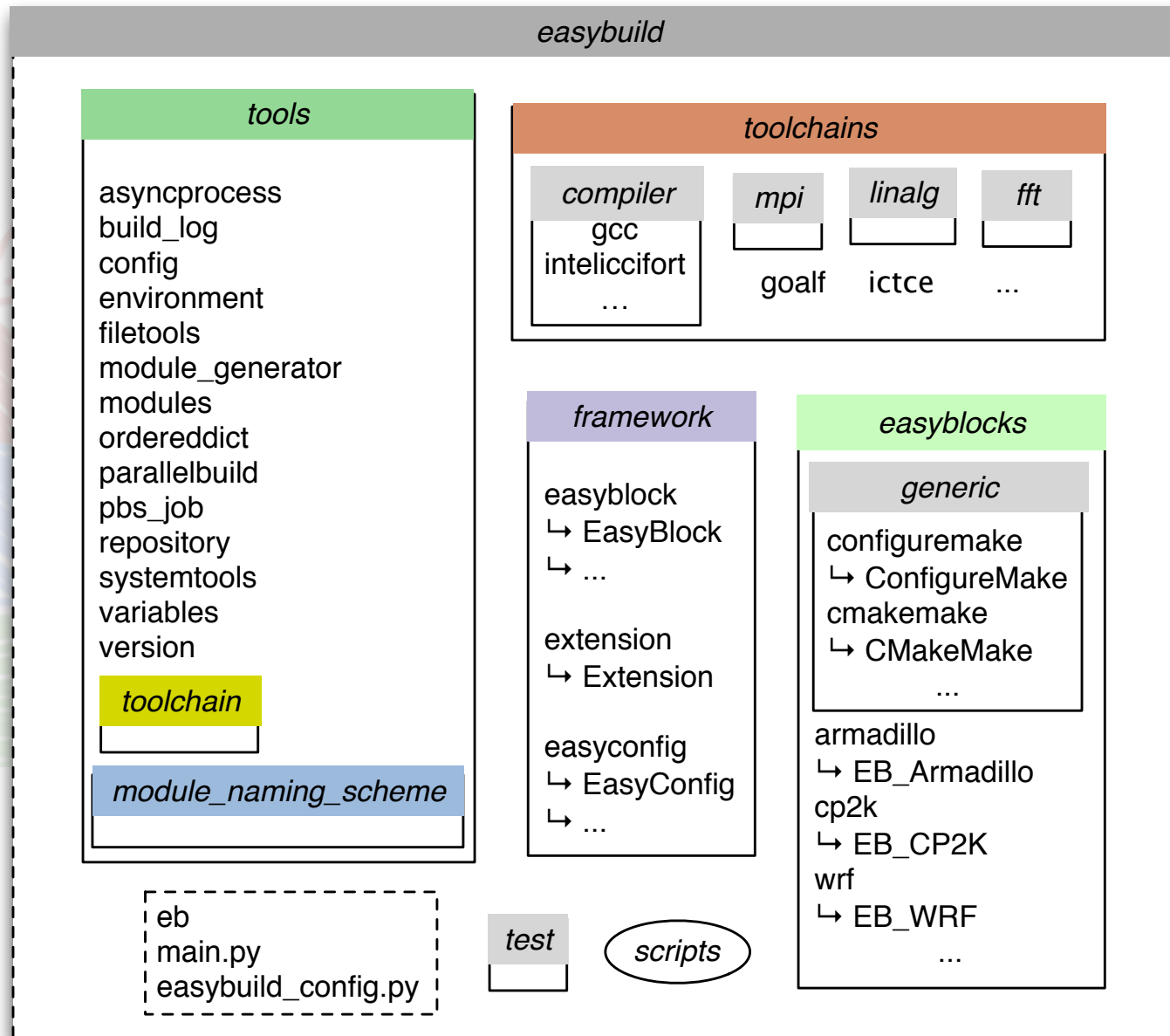
■ **easyconfig (file)**

- text file (Python syntax) with build specifications, mostly key-value
- supplied to EasyBuild, either directly or via ‘robot’ (for dependencies)
- new easyconfigs with slightly different specs can be *generated*

■ **compiler toolchain**

- collection of compilers & libraries for building software with
- usually C/C++/Fortran compilers + libs for MPI, BLAS, LAPACK, FFT
- EasyBuild sets up build environment for used toolchain (*\$CC*, ...)

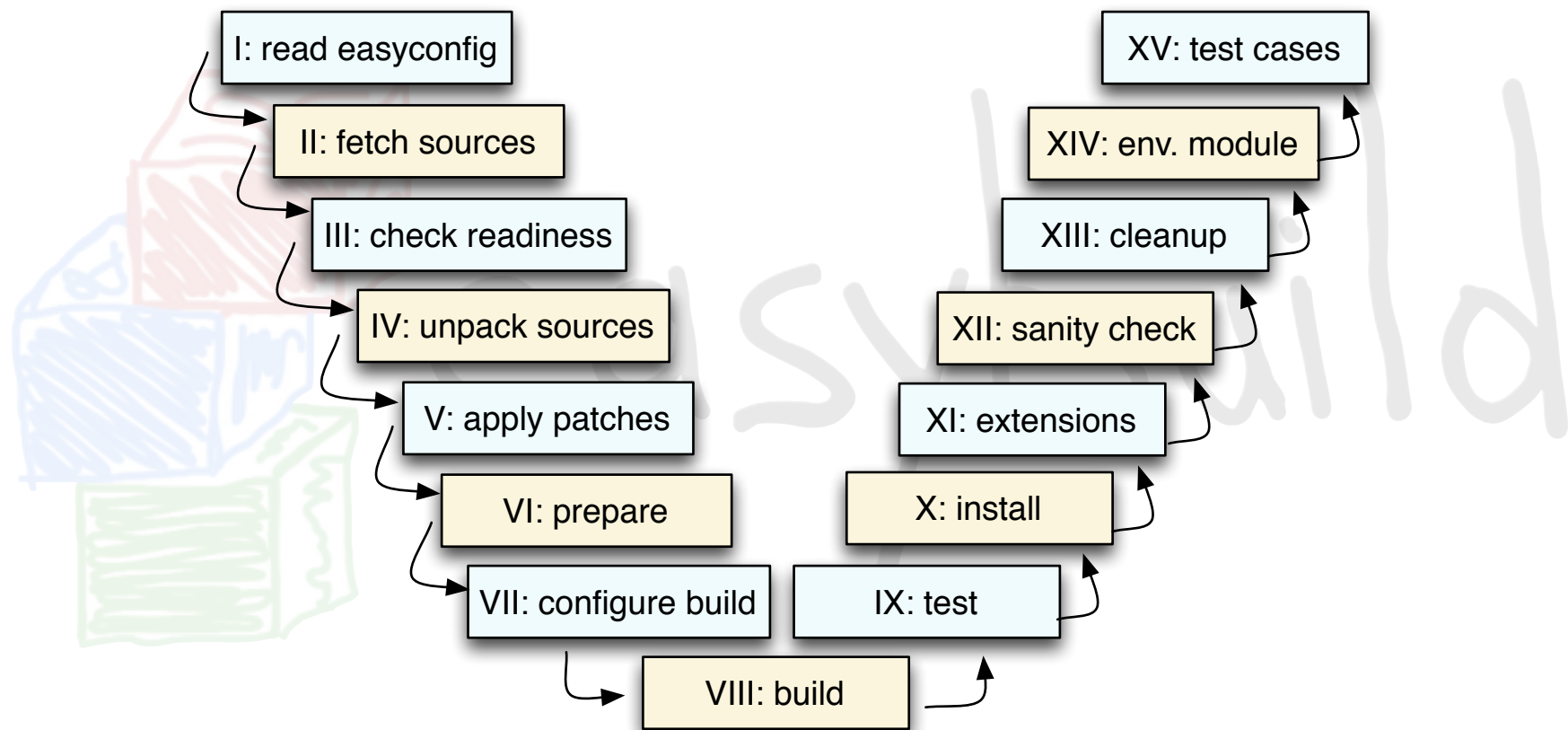
High-level design (framework)





Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customised if required,
via *easyconfig* parameters or a *custom easyblock*



List of supported software (v1.15.2)

511 different software packages (2,809 example easyconfigs)

a2ps ABAQUS ABINIT ABySS ACML **ALADIN** Allinea ALLPATHS-LG AMOS AnalyzeFMRI ANSYS ant APBS ARB argtable aria2 Armadillo arpack-ng ASE ATLAS Autoconf Automake bam2fastq BamTools Bash BayesTraits bbcp bbFTP bbftpPRO bc beagle-lib Beast BEDTools BFAST binutils biodeps BioPerl Biopython BiSearch Bison BitSeq BLACS BLAST BLAT BOINC Bonnie++ Boost Bowtie Bowtie2 BWA byacc bzip2 cairo CAP3 CBLAS ccache CCfits CD-HIT CDO CEM CFITSIO cflow CGAL cgdb cgmpich cgmpolf cgmvapich2 cgmvolv cgompi cgoolf Chapel CHARMM Circos Clang ClangGCC CLHEP CLooG Clustal-Omega ClustalW2 CMake Coreutils Corkscrew **CP2K** CPLEX CRF++ Cube CUDA Cufflinks cURL cutadapt CVS CVXOPT Cython DB DB_File Diffutils DL_POLY Classic Docutils **DOLFIN** Doxygen **EasyBuild** ECore ed Eigen ELinks ELPA ELPH Emacs EMBOSS EPD ErlangOTP ESMF ESPResSo evmix expat eXpress FASTA fastahack FastTree FASTX-Toolkit FCM FDTD_Solutions Ferret FFC FFTW FIAT file findutils fixesproto flex FLTK FLUENT fmri FoldX fontconfig foss FRC_align freeglut FreeSurfer freetype FSL g2clib g2lib GATE GATK gawk GCC gcccuda GD GDAL GDB Geant4 GEM-library GEMSTAT GenomeAnalysisTK GEOS gettext GHC Ghostscript gimkl gimpi GIMPS git GLib GLIMMER GLPK glproto gmacml GMAP GMP gmpich2 gmpolf GMT gmvapich2 gmvolf gnuplot gnutls Go goalf gomp gompic google-sparsehash goolf goolfc GPAW gperf gperftools Greenlet grep grib_api GROMACS GSL gsl GTI guile gzip h4toh5 h5py h5utils Harminv HDF HDF5 HH-suite HMMER horton HPCBIOS_Bioinfo HPCBIOS_Debuggers HPCBIOS_LifeSciences HPCBIOS_Math HPCBIOS_Profilers HPCG HPL HTSeq hwloc Hypricc icc iccifort ictce ifort iimpi iiqmpi imake imkl impi Infernal inputproto Inspector Instant intel iomkl iompi IOR lperf ipp IPython iqacml Isolnfer ispc itac JAGS Jansson JasPer Java Jellyfish Jinja2 JUnit kbproto LAPACK less lftp libcircle libctl libdrm libffi libgd libgtextutils libharu libibmad libibumad libibverbs libICE libidn Libint libint2 libjpeg-turbo libmatheval libpciaccess libpng libpthread-stubs libreadline libSM libsmm LIBSVM LibTIFF libtool libungif libunistring libunwind libX11 libXau libXaw libxc libxcb libXext libXfixes libXi libxml2 libXmu libXp libXpm libxslt libXt libyaml likwid Lmod Lua LWM2 lxml lynx LZO M4 MAFFT make makedepend Maple MariaDB Mathematica MATLAB matplotlib mc MCL mcpp MDP mdtest Meep MEME Mercurial Mesa Mesquite MetaVelvet MethPipe METIS MMSEQ Modeller Molden Molekel molmod Mothur motif MPFR mpi4py mpiBLAST MPICH MPICH2 MrBayes MTL4 MUMmer MUMPS MUSCLE MUST MUSTANG MVAPICH2 NAMD nano NASM NCBI-Toolkit ncdf4 **NCL** ncurses ncview NEdit netaddr netCDF netCDF-C++ netCDF-C++4 netCDF-Fortran netcdf4-python netifaces netloc nettle **NEURON** nodejs ns numactl numexpr numpy NWChem O2scl Oases OCaml Oger OPARI2 OpenBabel OpenBLAS **OpenFOAM** **OpenFOAM-Extend** OpenIFS OpenMPI OpenPGM OpenSSL ORCA orthomcl otcl OTF OTF2 packmol PAML pandas PANDaseq PAPI parallel Paraview ParFlow ParMETIS ParMGridGen Pasha patch paycheck PCC PCRE PDT Perl **PETSc** petsc4py phonopy PhyML picard pixman pkg-config PLINK PnMPI popt PP PRACE PRANK Primer3 printproto problog protobuf pscom PSI psmpi2 PyQuante pysqlite pyTables **Python** python-dateutil python-meep PyYAML PyZMQ QLogicMPI Qt qtop QuadProg++ **QuantumESPRESSO** R RAXML RCS RDP-Classifer RNAz ROOT Rosetta rSeq RSEQtools Ruby Sablotron SAMtools ScaLAPACK Scalasca ScientificPython scikit-learn scipy SCons SCOOP Score-P SCOTCH SDCC SDPA sed segemehl setuptools Shapely SHRiMP SIBELia sickle Silo slalib-c SLEPc SOAPaligner SOAPdenovo SOAPdenovo2 SOAPec SPAdes Sphinx SQLite SRA-Toolkit Stacks stemming Stow Stride SuiteSparse SURF SWIG sympy Szif TAMkin Tar tbb TCC Tcl tccl tcsh Tesla-Deployment-Kit texinfo Theano TiCCutils TiMBL TinySVM Tk TopHat Tornado TotalView TREE-PUZZLE Trilinos Trinity UDUNITS UFC UFL util-linux Valgrind VCFtools Velvet ViennaRNA Vim Viper vsc-base vsc-mypirun vsc-mypirun-scoop vsc-processcontrol VSC-tools VTK VTune WHAM **WIEN2k** wiki2beamer **WPS** **WRF** xbitmaps xcb-protocol XCrySDen xextproto XML XML-LibXML XML-Simple xorg-macros xproto xtrans XZ yaff YamCha YAML-Syck Yasm YAXT ZeroMQ zlib zsh zsync



Installing EasyBuild

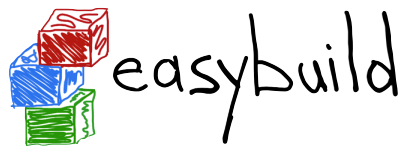
<https://github.com/hpcugent/easybuild/wiki/Bootstrapping-EasyBuild>

Install EasyBuild using the **bootstrap script** (highly recommended):

```
$ curl -O https://raw.githubusercontent.com/hpcugent/easybuild-framework/  
develop/easybuild/scripts/bootstrap_eb.py  
  
$ python bootstrap_eb.py /tmp # specify your install prefix  
  
$ export MODULEPATH=/tmp/modules/all:$MODULEPATH  
  
$ module load EasyBuild
```

Updating:

```
$ module load EasyBuild/1.15.1  
$ eb EasyBuild-1.15.0.eb --try-software-version=1.15.2  
$ module swap EasyBuild EasyBuild/1.15.2
```



Configuring EasyBuild

<https://github.com/hpcugent/easybuild/wiki/Configuration>

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

You *should* configure EasyBuild to your preferences, using:

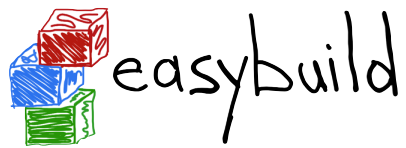
- **configuration files:** key-value lines, text files (e.g., `prefix='/foo'`)
- **environment variables** (e.g., `EASYBUILD_PREFIX=/foo`)
- **command line parameters** (e.g., `--prefix=/foo`)

Consistency across these options is guaranteed (see `eb --help | tail`)

Different options in order of preference: cmdline, env vars, config file

`--prefix` overrides `$EASYBUILD_PREFIX`,

`$EASYBUILD_PREFIX` overrides `prefix` in configuration file



First steps with *eb*

installing bzip2 v1.0.6 with system compiler:

```
eb bzip2-1.0.6.eb  
module load bzip2/1.0.6
```

or (equivalent), install latest version (that EasyBuild knows about):

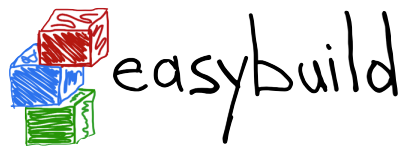
```
eb --software-name=bzip2 --toolchain-name=dummy
```

Install *goolf* compiler toolchain (be patient):

```
eb goolf-1.4.10-no-OFED.eb --robot # no-OFED indicates no IB support
```

Install gzip v1.6 on top of *goolf* toolchain, log with debug info to stdout:

```
eb gzip-1.6-goolf-1.5.14-no-OFED.eb -l dr
```



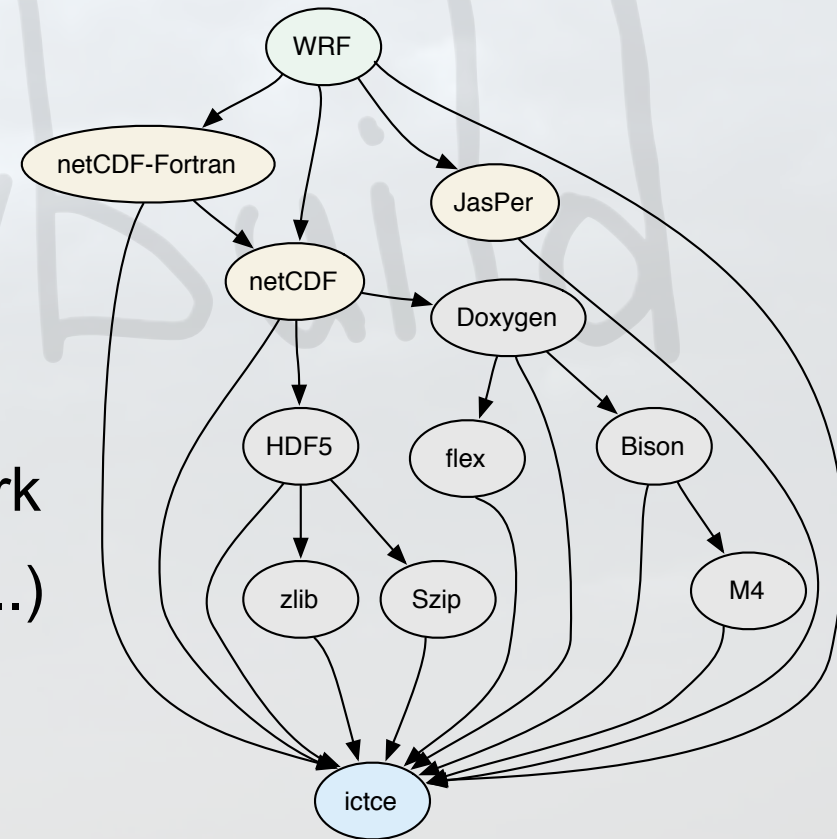
EasyBuild command line

- getting help, overview of options: *eb --help*
- list of available easyconfig parameters: *eb -a*
- robot build, debug log to stdout: *eb bzip2-1.0.6.eb -ldr*
- overview of required/available modules: *eb --dry-run* or *eb -D*
- list of known toolchains + their definition: *eb --list-toolchains*
- generating easyconfigs: *eb bzip2-1.0.6.eb --try-toolchain=ictce,6.2.5 -r*
- searching for easyconfigs: *eb -S* or *eb --search*
- use easyconfigs available in an (open) pull request: *eb --from-pr <PR#>*
- test pull request and upload test report:
eb --from-pr <PR#> --upload-test-report --github-user=boegel

Use case: building WRF

*building and installing **WRF** (Weather Research and Forecasting Model)*

- ▶ <http://www.wrf-model.org>
- ▶ complex(ish) **dependency graph**
- ▶ **very non-standard build procedure**
 - ▶ interactive configure script (!)
 - ▶ resulting `configure.wrf` needs work (hardcoding, tweaking of options, ...)
- ▶ `compile` script (wraps around `make`)
- ▶ no actual installation step



Use case: building WRF with *eb*

*building and installing **WRF** (Weather Research and Forecasting Model)*

- ▶ easyblock that comes with EasyBuild implements build procedure
 - ▶ running interactive `configure` script **autonomously**
 - ▶ **patching** `configure.wrf`
 - ▶ **building** with `compile` script
 - ▶ **testing** build with standard included tests/benchmarks
- ▶ easyconfig files for different versions, toolchains, build options, ...
- ▶ building and installing WRF becomes child's play, for example:

```
eb --software=WRF,3.4 --toolchain-name=ictce --robot
```

Use case: easyblock for WRF

part I: imports, class constructor,
custom easyconfig parameter

```
1 import fileinput, os, re, sys
2
3 import easybuild.tools.environment as env
4 from easybuild.easyblocks.netcdf import set_netcdf_env_vars
5 from easybuild.framework.easyblock import EasyBlock
6 from easybuild.framework.easyconfig import MANDATORY
7 from easybuild.tools.filetools import patch_perl_script_autoflush, run_cmd, run_cmd_qa
8 from easybuild.tools.modules import get_software_root
9
10 class EB_WRF(EasyBlock):
11
12     def __init__(self, *args, **kwargs):
13         super(EB_WRF, self).__init__(*args, **kwargs)
14         self.build_in_installdir = True
15
16     @staticmethod
17     def extra_options():
18         extra_vars = [('buildtype', [None, "Type of build (e.g., dmpar, dm+sm).", MANDATORY])]
19         return EasyBlock.extra_options(extra_vars)
20
21     def configure_step(self):
22         # prepare to configure
23         set_netcdf_env_vars(self.log)
24
```

import required functionality

class definition

class constructor, specify building in installation dir

define custom easyconfig parameters

Use case: easyblock for WRF

part II: configuration (1/2)

```
21 def configure_step(self):
22     # prepare to configure
23     set_netcdf_env_vars(self.log)
24
25     jasper = get_software_root('JasPer')
26     if jasper:
27         jasperlibdir = os.path.join(jasper, "lib")
28         env.setvar('JASPERINC', os.path.join(jasper, "include"))
29         env.setvar('JASPERLIB', jasperlibdir)
30
31     env.setvar('WRFIO_NCD_LARGE_FILE_SUPPORT', '1')
32
33     patch_perl_script_autoflush(os.path.join("arch", "Config_new.pl"))
34
35     known_build_types = ['serial', 'smpar', 'dmpar', 'dm+sm']
36     self.parallel_build_types = ["dmpar", "smpar", "dm+sm"]
37     bt = self.cfg['buildtype']
38
39     if not bt in known_build_types:
40         self.log.error("Unknown build type: '%s' (supported: %s)" % (bt, known_build_types))
41
```

configuration step function

set environment variables for dependencies

set WRF-specific env var for build options

patch configure script to run it autonomously

check whether specified build type makes sense

Use case: easyblock for WRF

part II: configuration (2/2)

```

42 # run configure script
43 bt_option = "Linux x86_64 i486 i586 i686, ifort compiler with icc"
44 bt_question = "\s*(?P<nr>[0-9]+).\s*%s\s*\(%s\)" % (bt_option, bt)
45
46 cmd = "./configure"
47 qa = {"(1=basic, 2=preset moves, 3=vortex following) [default 1]:" : "1",
48       "(0=no nesting, 1=basic, 2=preset moves, 3=vortex following) [default 0]:" : "0"}
49 std_qa = {r"%s.*\n(.*)\n*Enter selection\s*\[[0-9]+-[0-9]+\]\s*:" % bt_question: "%(nr)s"}
50
51 run_cmd_qa(cmd, qa, no_qa=[], std_qa=std_qa, log_all=True, simple=True)
52
53 # patch configure.wrf
54 cfgfile = 'configure.wrf'
55
56 comps = {
57     'SCC': os.getenv('CC'), 'SFC': os.getenv('F90'),
58     'CCOMP': os.getenv('CC'), 'DM_FC': os.getenv('MPIF90'),
59     'DM_CC': "%s -DMPI2_SUPPORT" % os.getenv('MPICC'),
60 }
61
62 for line in fileinput.input(cfgfile, inplace=1, backup='.orig.comps'):
63     for (k, v) in comps.items():
64         line = re.sub(r"^(%s\s*=%s*).*$" % k, r"\1 %s" % v, line)
65     sys.stdout.write(line)
66

```

prepare Q&A for configuring

run configure script autonomously

patch generated configuration file

Use case: easyblock for WRF

part III: build step & skip install step (since there is none)

build step function

```
67 def build_step(self):
68     # build WRF using the compile script
69     par = self.cfg['parallel']
70     cmd = "./compile -j %d wrf" % par
71     run_cmd(cmd, log_all=True, simple=True, log_output=True)
72
73     # build two test cases to produce ideal.exe and real.exe
74     for test in ["em_real", "em_b_wave"]:
75         cmd = "./compile -j %d %s" % (par, test)
76         run_cmd(cmd, log_all=True, simple=True, log_output=True)
77
78 def install_step(self):
79     pass
80
```

**build WRF
(in parallel)**

**build WRF
utilities as well**

**no actual installation step
(build in installation dir)**

Use case: installing WRF

specify build details in easyconfig file (.eb)

```
1 name = 'WRF'
2 version = '3.4'
3
4 homepage = 'http://www.wrf-model.org'
5 description = 'Weather Research and Forecasting'
6
7 toolchain = {'name': 'ictce', 'version': '5.3.0'}
8 toolchainopts = {'opt': False} # no -O2
9
10 sources = ['%sV%s.TAR.gz' % (name, version)]
11 patches = ['WRF_parallel_build_fix.patch',
12            'WRF-%(version)s_known_problems.patch',
13            'WRF_tests_limit-runtimes.patch',
14            'WRF_netCDF-Fortran_separate_path.patch']
15
16 dependencies = [('JasPer', '1.900.1'),
17                 ('netCDF', '4.2.1.1'),
18                 ('netCDF-Fortran', '4.2')]
19
20 buildtype = 'dmpar'
```

software name and version →

software website and description (informative) ←

compiler toolchain specification and options →

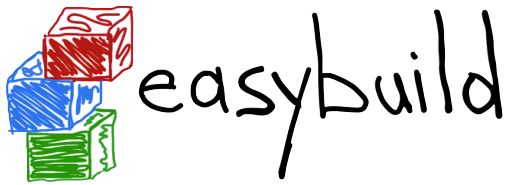
list of source files ←

list of patches for sources ←

list of dependencies ←

custom parameter for WRF →

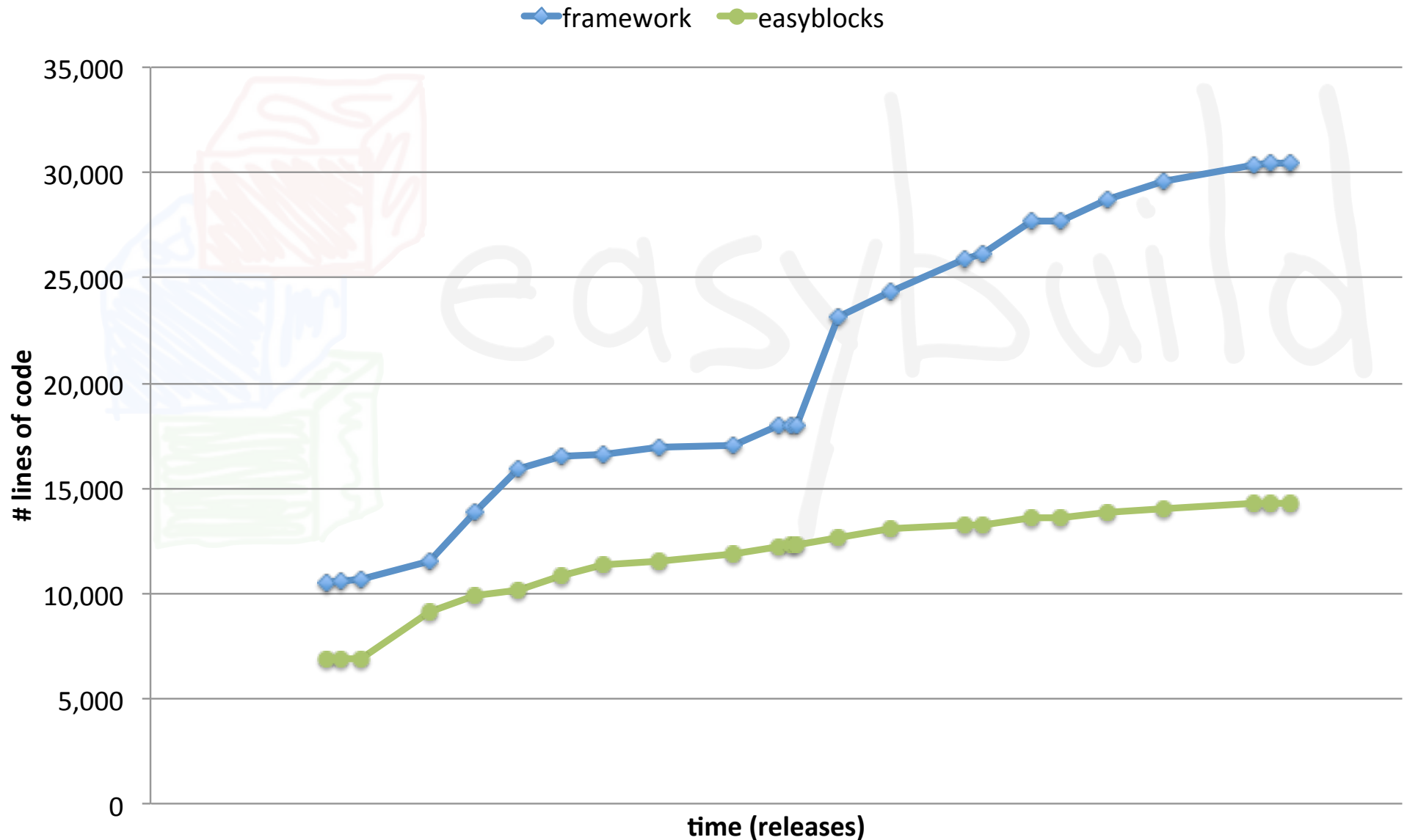
eb WRF-3.4-ictce-5.3.0-dmpar.eb --robot

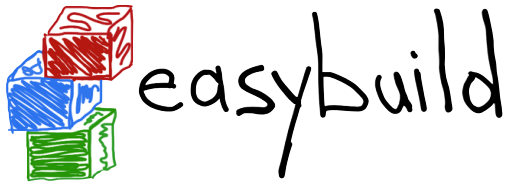


Growth: codebase

lines of code

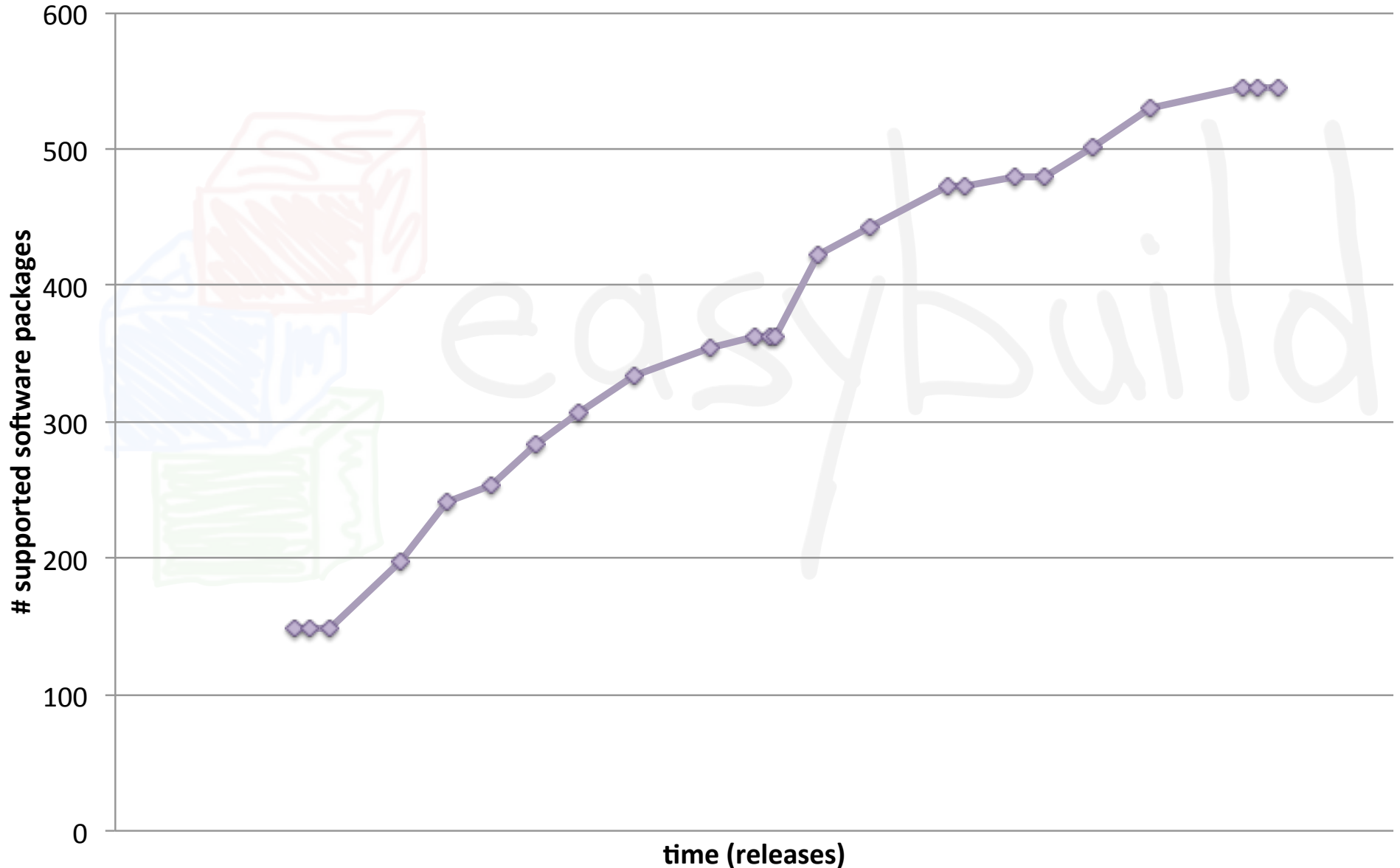
```
find . -name '*.py' | xargs cat | egrep -v '^#|^[\ ]*$' | wc -l
```

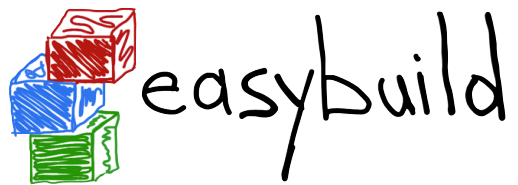




Growth: supported software

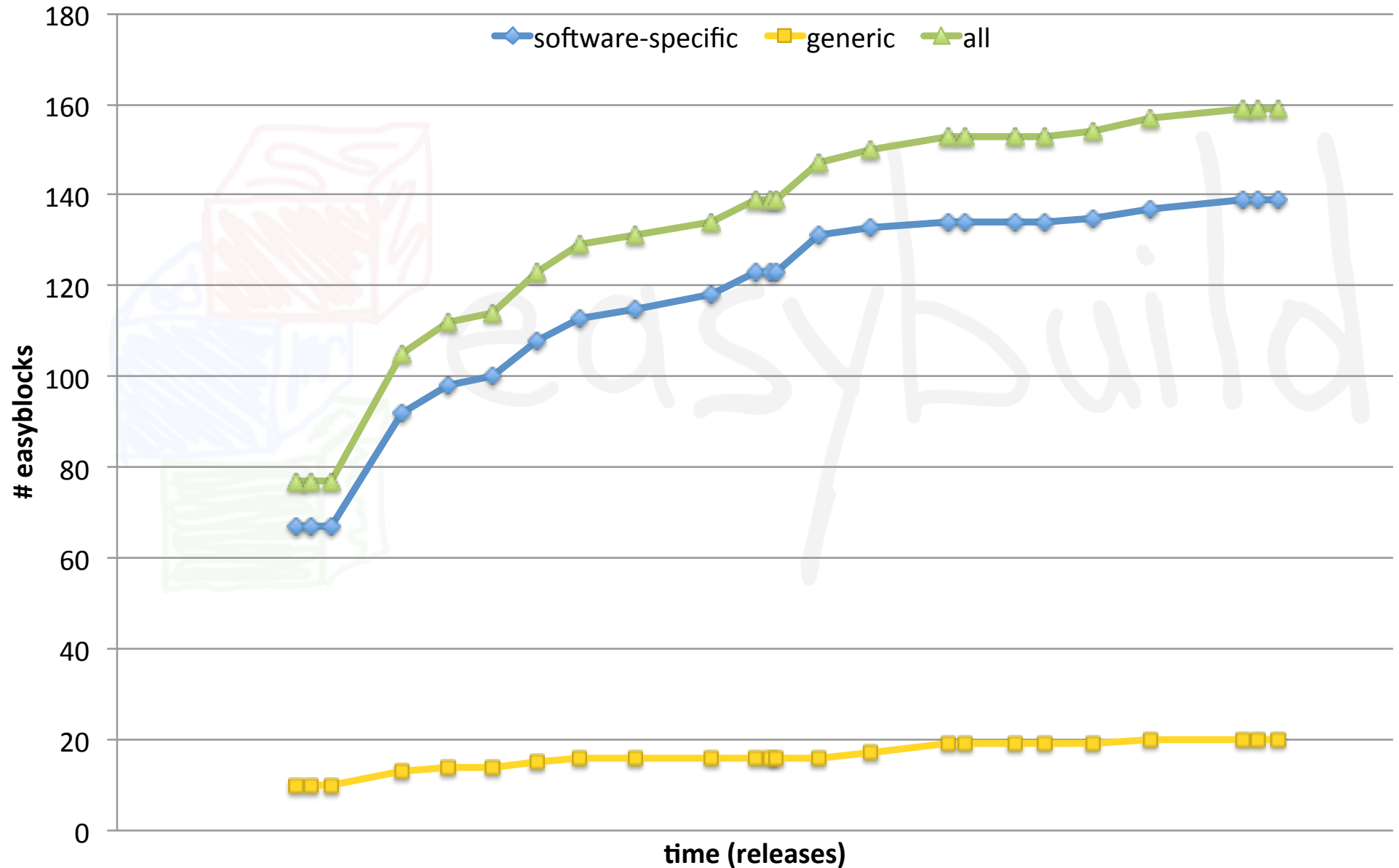
supported software packages (incl. toolchains)





Growth: supported software

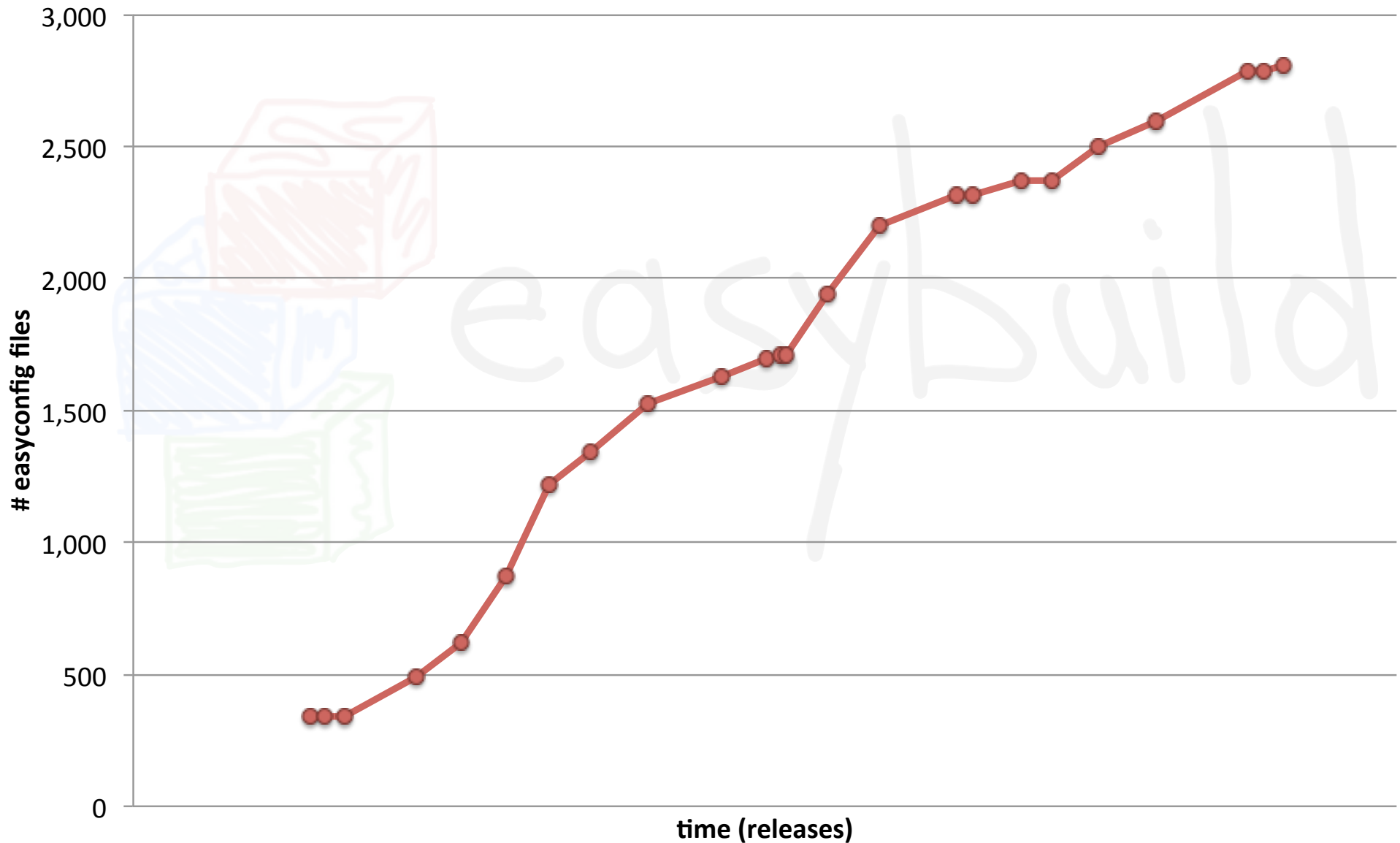
easyblocks

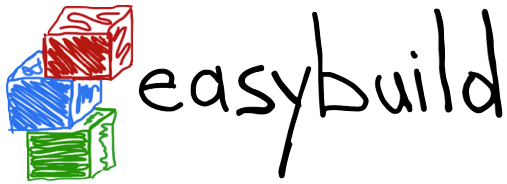




Growth: supported software

easyconfig files





Growth: community

EasyBuild community is growing slowly but steadily:

- Ghent University (HPC-UGent & users)
- K.U. Leuven, Antwerp Univ., Hasselt Univ. (VSC members)
- University of Luxembourg
- Gregor Mendel Institute (Austria)
- The Cyprus Institute
- University of Basel (Switzerland)
- Jülich Supercomputer Centre (Germany)
- University of Auckland (New Zealand)
- Bayer (Germany)
- University of Warwick (UK)
- Stanford University (US)
- NVIDIA Corp.
- Idaho National Lab (US)
- Kiev Polytechnic Institute (NTTU, Ukraine)
- Pacific Northwest National Lab (US)
- UC Davis (US)
- ...



easybuild

building software with ease

Do you want to know more?

website: <http://hpcugent.github.com/easybuild>

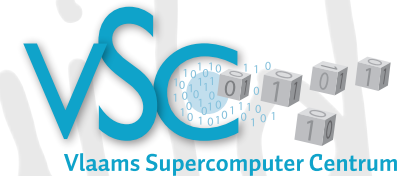
GitHub: [https://github.com/hpcugent/easybuild\[-framework\]-easyblocks\[-easyconfigs\]](https://github.com/hpcugent/easybuild[-framework]-easyblocks[-easyconfigs])

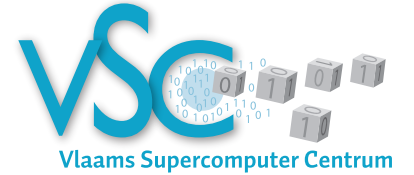
PyPi: [http://pypi.python.org/pypi/easybuild\[-framework\]-easyblocks\[-easyconfigs\]](http://pypi.python.org/pypi/easybuild[-framework]-easyblocks[-easyconfigs])

mailing list: easybuild@lists.ugent.be

Twitter: [@easy_build](https://twitter.com/easy_build)

IRC: [#easybuild](https://freenode.net) on freenode.net





easybuild

building software with ease

EasyBuild workshop @ Jülich Supercomputer Centre
October 21st 2014

*Kenneth Hoste - kenneth.hoste@ugent.be
easybuild@lists.ugent.be*