easybuild

building software with ease

Jülich Supercomputer Centre (JSC)

Feb 19th 2014

Kenneth Hoste - kenneth.hoste@ugent.be

easybuild@lists.ugent.be

HPC-UGent, Ghent University, Belgium

- central contact for High Performance Computing at UGent
- established in 2008, part of central IT department (DICT)
- member of Flemish Supercomputer Centre (VSC)
  - virtual centre, collaboration between Flemish university associations

Vlaams Supercomputer Centrum

The HPC-UGent team

Stijn De Weirdt
technical lead

Ewald Pauwels
team lead

Kenneth Hoste
user support, EasyBuild

Wouter Depypere
system administration (Tier1)

Jens Timmerman
user support & system administration

Ewan Higgs
user support

Andy Georges
user support & system administration

Kenneth Waegeman
system administration (storage)

HPC-UGent infrastructure: 7 Tier2 systems

3 Tier2 ‘batch’ clusters

- for single-node, non IO-intensive jobs
- Gigabit Ethernet, no shared scratch filesystem
- IBM - Intel Nehalem (2x) - 56x8 + 168x8 cores
- HP - Intel Sandy Bridge - 64x16 cores

(2009)

(2010)

(2012)

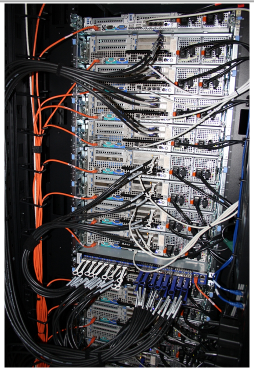
(2009)

(2011)

(2013)

3 Tier2 MPI clusters

- *intended* for multi-node MPI jobs
- Infiniband, dedicated shared scratch filesystem
- IBM - Intel Harpertown - IB DDR - ~1,200 cores
- Dell - AMD Magny Cours - IB QDR - 1,088 cores
- Dell - Intel Sandy Bridge - IB FDR - 2,560 cores


(2011)

1 Tier2 special-purpose cluster

- ‘reverse virtualization’ via ScaleMP vSMP
- Dell - Intel Westmere - 16x dual 6-core
- virtual systems: 3x 48-core ~350GB RAM, 2x 24-core ~190GB RAM

VSC infrastructure: Tier1

first Tier1 system in Flanders, hosted at UGent

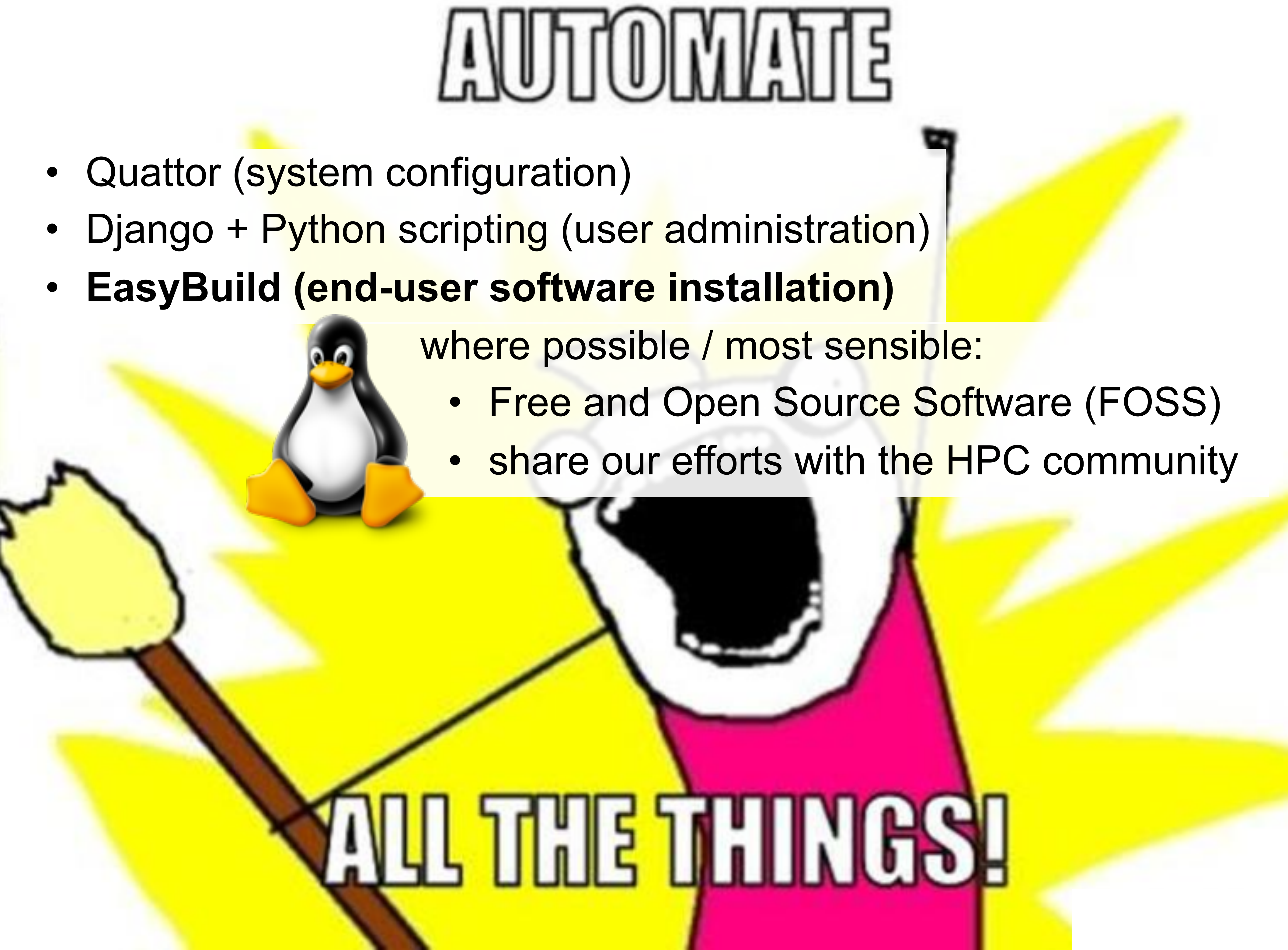
*€4,2M - HP - 528 nodes - 8,448 cores - **152.3 TFlops***

Infiniband FDR - 450TB scratch - 223 kW peak

***Top500**: #118 (June 2012), #163 (Nov 2012),*
*#239 (June 2013), **#306 (Nov 2013)***

HPC-UGent in a nutshell

- team of 8 people (7 technical)
    - ~ 4 FTEs for system administration
    - ~ 3 FTEs for user support
- 7 Tier2 clusters (~600 servers), 1 Tier1 cluster (500+ servers)
    - + peripherals (storage, master nodes, monitoring, ...)
- over 1000 user accounts
- researchers from all over the university (all research domains)
- our tasks
    - purchasing new systems
    - installing and configuring systems + maintenance (OS updates, etc.)
    - keeping things running (efficiently)
    - user support on various levels (dummy to advanced)
    - user training (getting started, OpenMP, MPI, ...)
    - collaboration with other VSC members, training, outreach, ...



- Quattor (system configuration)
- Django + Python scripting (user administration)
- **EasyBuild (end-user software installation)**

where possible / most sensible:

- Free and Open Source Software (FOSS)
- share our efforts with the HPC community

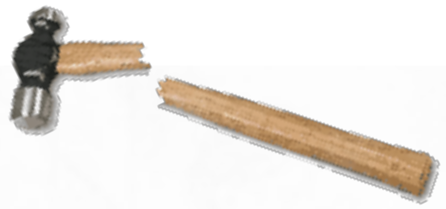
Building scientific software is... fun!

Scientists focus on the **functionality** of their software,
not on portability, build system, ...

Common **problems** with build procedures of scientific software:

- **incomplete**, e.g. no install step
- requires human **interaction**
- heavily **customized** and **non-standard**
- uses **hard-coded** settings
- poor and/or outdated **documentation**

Very time-consuming for user support teams!



Available tools are insufficient

- not a lot of packaged scientific software available (RPMs, ...)
  - requires **huge effort,** which is duplicated across distros
  - packaging by distro doesn’t make such sense anyway
  - **building from source** is preferred in an HPC environment
- existing build tools are:
  - **not well suited** to scientific software (think RPM spec files)
  - **hard to maintain** (e.g., ‘object-oriented’ bash)
  - stand-alone, **no reuse** of previous efforts (!)
  - **OS-dependent** (HomeBrew, *Ports, ...)
  - **custom** to (groups of) software packages

    e.g., Dorsal (DOLFIN), gmkpack (ALADIN), Sage, ...

Our build tool wish list

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

Building software with ease

a software build and installation framework

- written in **Python**
- started in 2009, developed in-house for about 2.5 years
- **open-source (GPLv2)** since April 2012
- EasyBuild v1.0: **stable API** (November 2012)
- ~ **monthly releases** (latest: v1.11.0, Feb 16th 2014)
- continuously enhanced and extended
- *http://hpcugent.github.io/easybuild*

Release schedule

- first public release in April 2012 (v0.5)
- **stable** since EasyBuild v1.0
  - released November 13th 2012 (at Supercomputing'12)
- **release early, release often** strategy
  - release whatever is there, don't stall waiting for major features
  - monthly releases since v1.1.0 (Jan'13)
  - occasional bug fix releases, as deemed necessary
  - most recent release is v1.11.0 (Feb 16th 2014), v1.12.0 is planned for Mar'14
- **feature freeze** strategy, **well-planned** releases we (try and) stick to
  - only open PRs with stable contributions are eligible for inclusion
- **careful testing** for every merge/release
  - unit test suite is run on every merge: try and make sure nothing breaks
  - full regression test is run for every release: build *everything* from scratch

Requirements

- **Linux** / OS X
  - used daily on Scientific Linux 5.x/6.x (Red Hat-based)
  - also tested on Fedora, Debian, Ubuntu, CentOS, SLES, ...
  - some known issues on OS X, focus is on Linux
  - no Windows support (and none planned for now)
- **Python v2.4** or more recent version (2.x, no Python 3 support yet)
- **environment modules** (or Lmod)
- (system C/C++ compiler to bootstrap a GCC toolchain)

Key features

- execute software build procedures **fully autonomously**
  - also interactive installers, code patching, generating module file, ...
- thorough **logging** and archiving
  - entire build process is logged thoroughly, logs stored in install dir
  - easyconfig file used for build is archived (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, by using `--job`
- comprehensive **testing**: unit tests, regression testing
- thriving, growing **community**

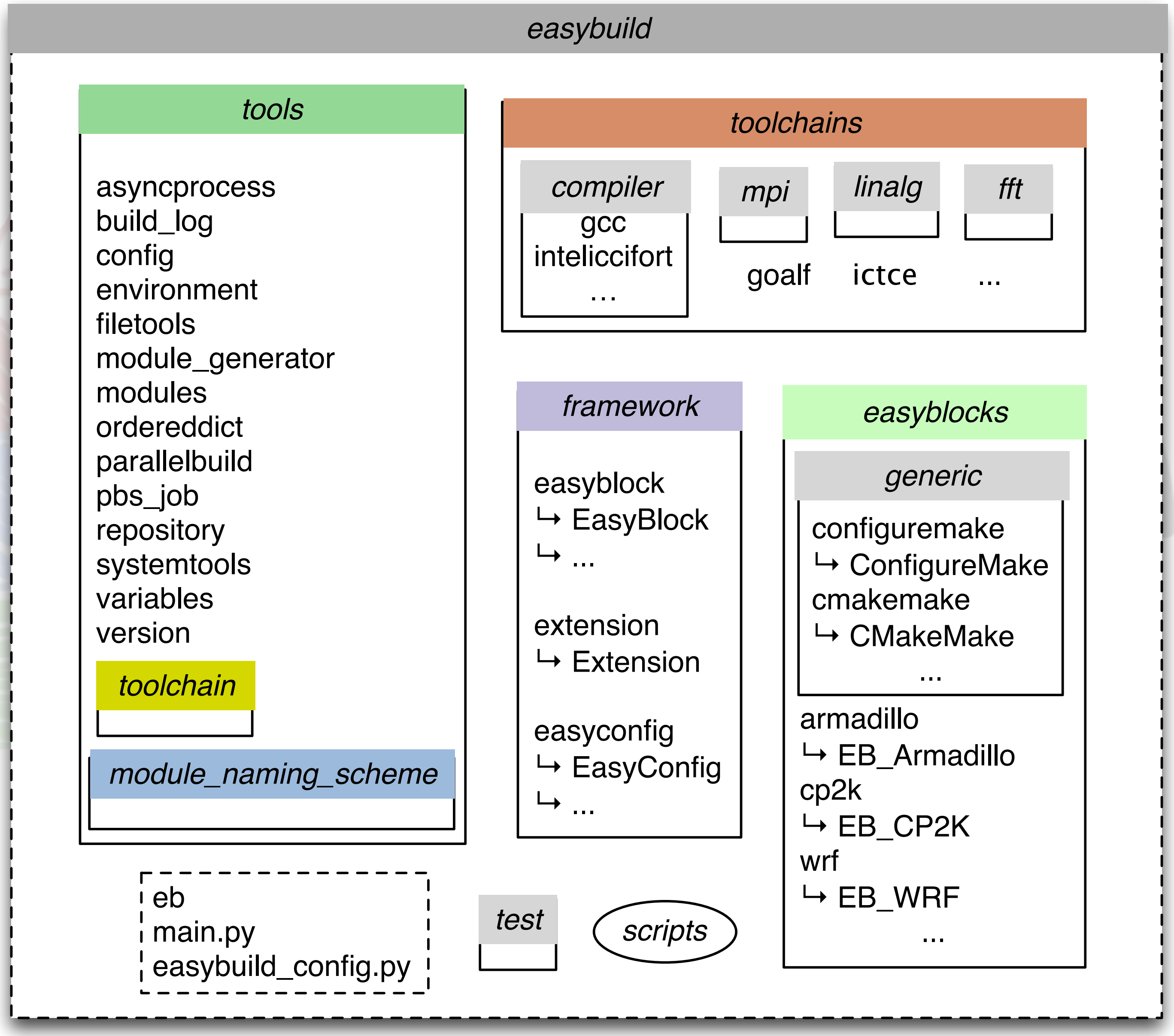
‘Quick’ demo for the impatient

```
eb HPL-2.0-goolf-1.4.10.eb --robot
```

- downloads all required sources (best effort)
- builds *goolf* toolchain (be patient), and builds HPL with it

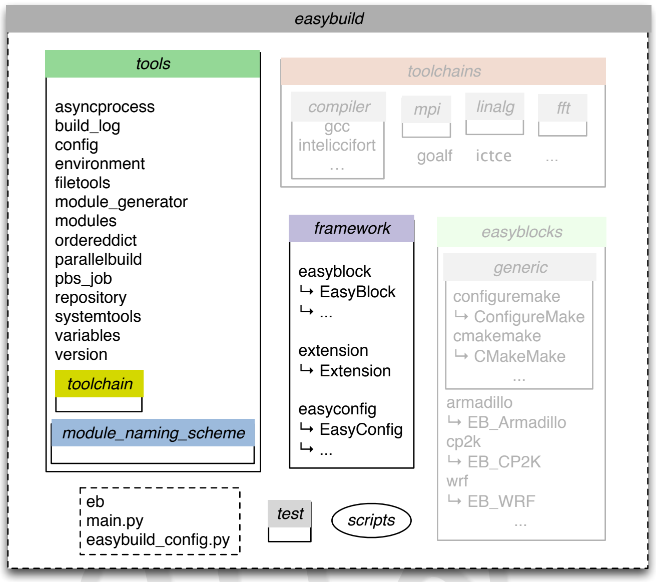
  goolf: GCC, OpenMPI, LAPACK, OpenBLAS, FFTW, ScaLAPACK
- default: source/build/install dir in `$HOME/.local/easybuild`
- all defaults can be tweaked via configuration (and if not, we’ll fix it)

EasyBuild: high-level design

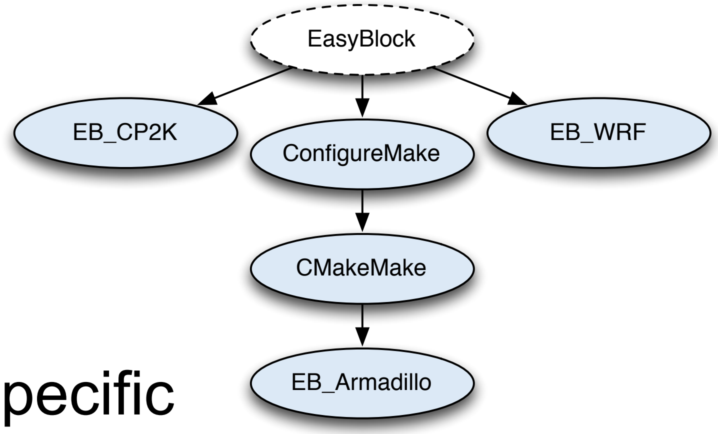


EasyBuild framework & easyblocks

- **framework**
  - *core of EasyBuild*
  - Python packages & modules (~25k LOC)
  - provides (lots of) supporting functionality
  - very modular and dynamic design
  - ‘plug and play’ support for new software, compiler toolchains, ...

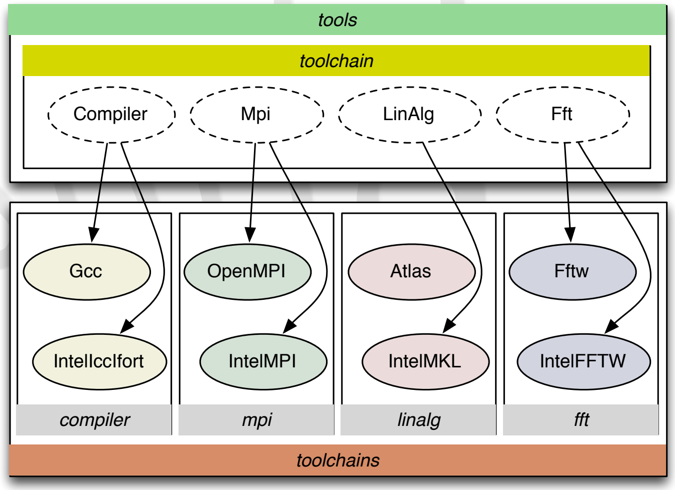


- **easyblocks**
  - *implementation of build procedures*
  - Python classes, hierarchically organized
  - an easyblock can be generic or software-specific
  - tie into EasyBuild framework via API (`easybuild.tools`, `EasyBlock`, ...)

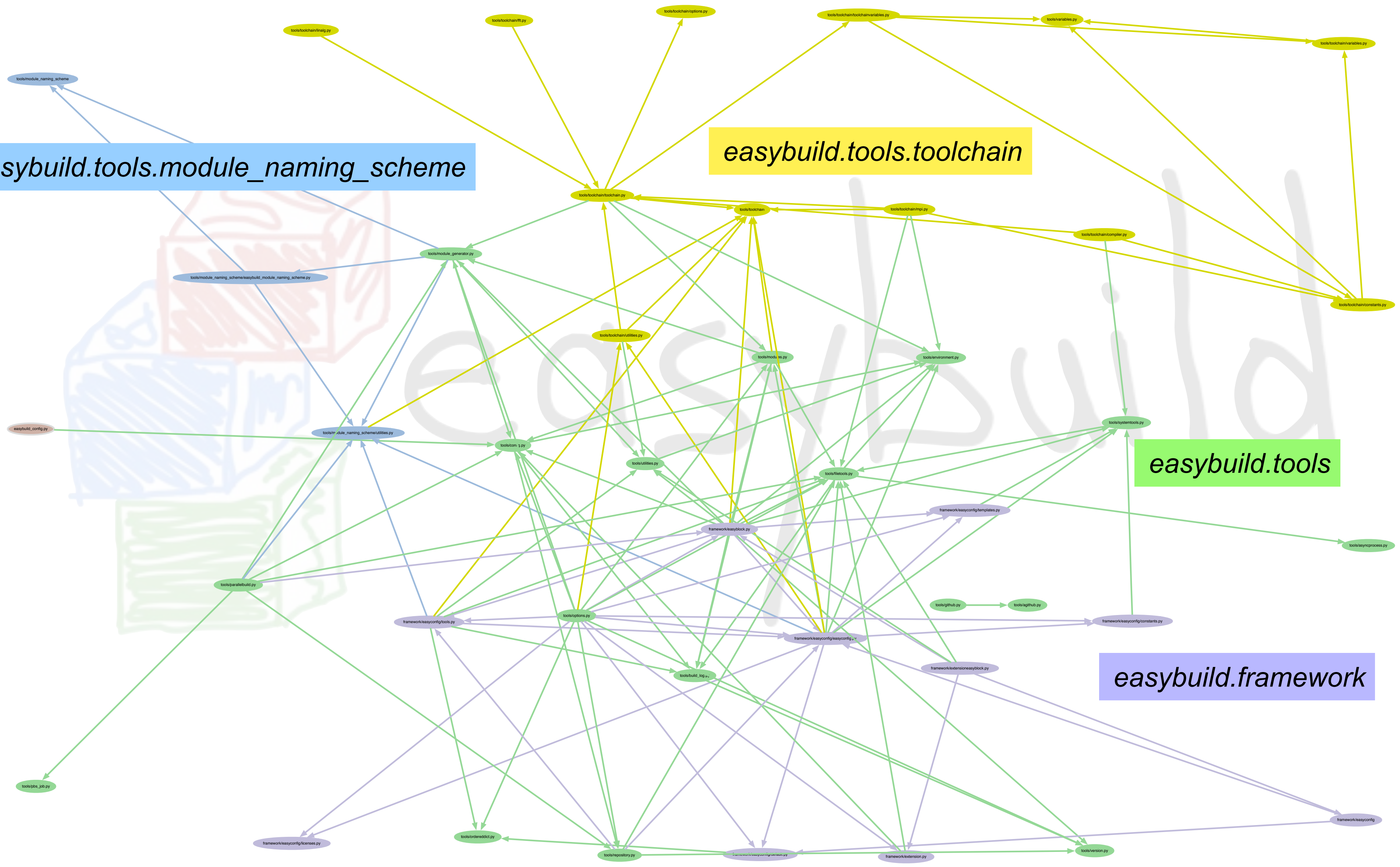


easyconfig files & compiler toolchains

- **easyconfig files (.eb)**
  - *build specifications*
    software name/version, compiler toolchain, build options, ...
  - simple text files
    - format v1: executed as Python code
    - format v2: move towards proper parsing (WIP)
- **(compiler) toolchains**
  - set of compilers and common libraries
    - usually includes MPI, BLAS, LAPACK, FFT
    - grouped together for convenience (e.g. shorter module names, ...)
  - ictce: Intel tools (icc, ifort, impi, imkl)
  - goolf: all open-source (GCC, OpenMPI, OpenBLAS, FFTW)
  - ... (define your own!)



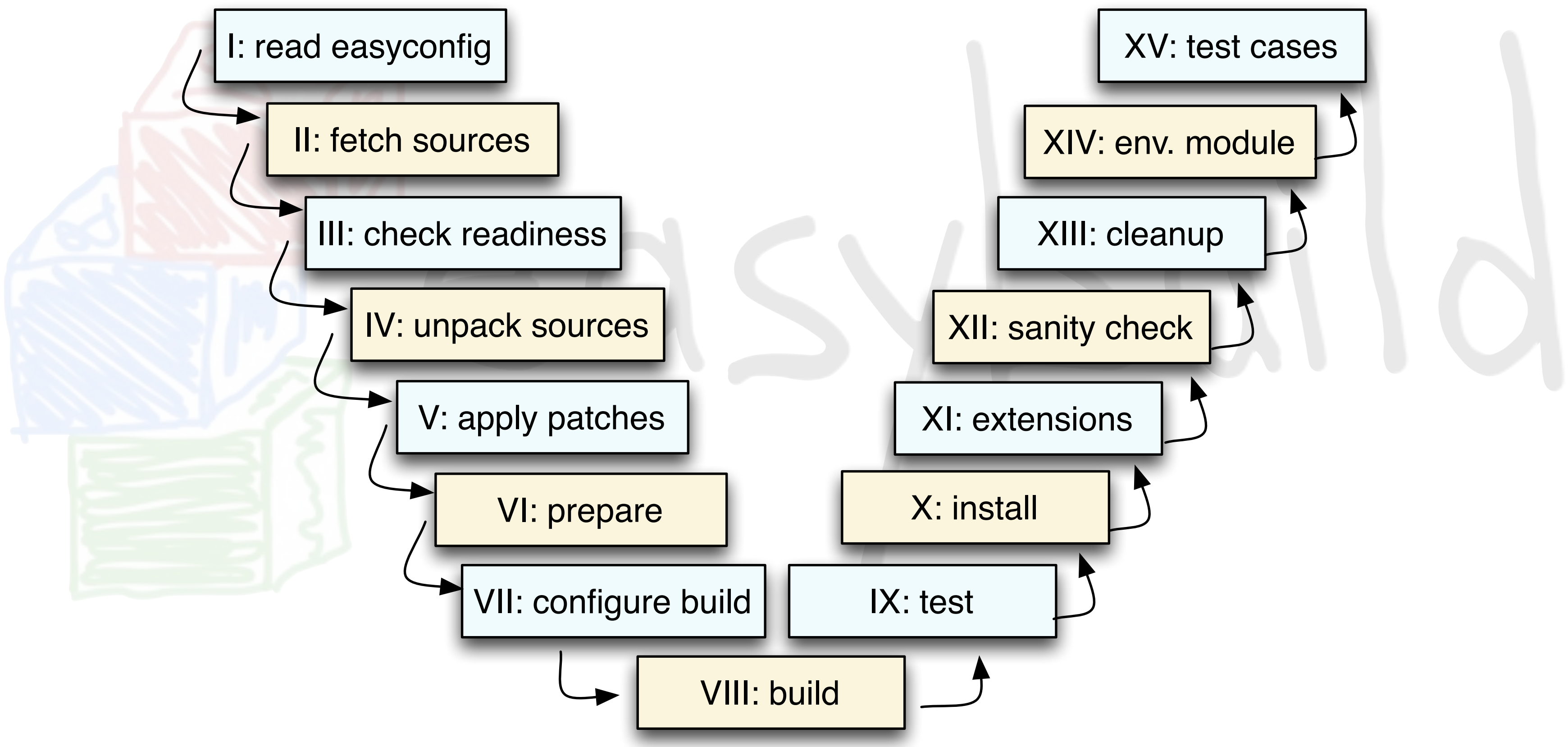
High-level design: framework



tools/toolchain/linalg.py
tools/toolchain/fft.py
tools/toolchain/options.py
tools/toolchain/toolchainvariables.py
tools/variables.py
tools/toolchain/variables.py
tools/module_naming_scheme
easybuild.tools.module_naming_scheme
easybuild.tools.toolchain
tools/toolchain/toolchain.py
tools/toolchain
tools/toolchain/mpi.py
tools/toolchain/compiler.py
tools/module_generator.py
tools/module_naming_scheme/easybuild_module_naming_scheme.py
tools/toolchain/constants.py
tools/toolchain/utilities.py
tools/modules.py
tools/environment.py
easybuild_config.py
tools/module_naming_scheme/utilities.py
tools/config.py
tools/systemtools.py
tools/utilities.py
tools/filetools.py
easybuild.tools
framework/easyconfig/templates.py
framework/easyblock.py
tools/asyncprocess.py
tools/parallelbuild.py
tools/github.py
tools/agithub.py
framework/easyconfig/tools.py
tools/options.py
framework/easyconfig/constants.py
framework/easyconfig/easyconfig.py
framework/extensioneasyblock.py
tools/build_log.py
easybuild.framework
tools/pbs_job.py
framework/easyconfig
framework/easyconfig/licenses.py
tools/ordereddict.py
tools/repository.py
framework/easyconfig/default.py
framework/extension.py
tools/version.py

Step-wise install procedure

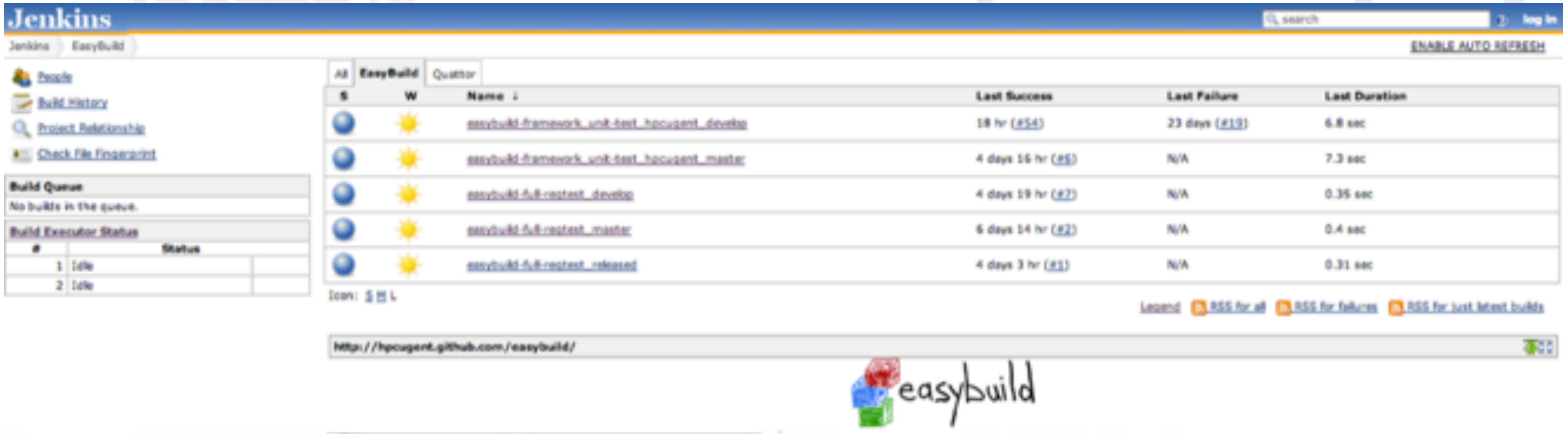
build and install procedure as implemented by EasyBuild



most of these steps can be customized if required

Comprehensive testing

- **unit tests** are run automagically by Jenkins on every 'push'
- **regression tests**: before every release, test rebuilding *everything*
- results publicly availale: *https://jenkins1.ugent.be/view/EasyBuild*



Build Queue

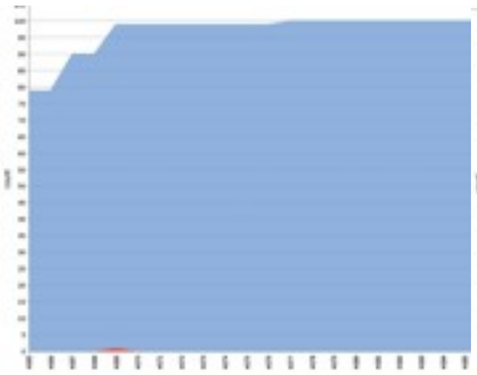
No builds in the queue.

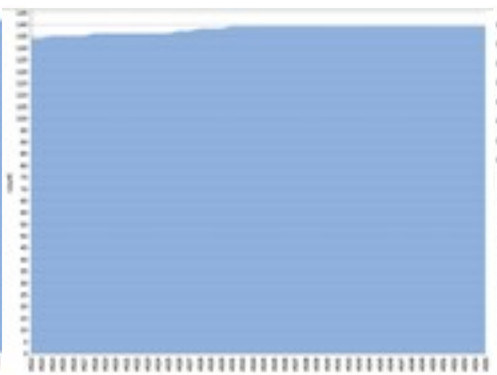
Build Executor Status

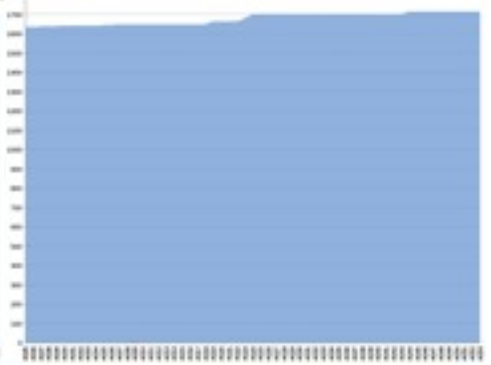
#	Status
1	Idle
2	Idle

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	easybuild-framework_unit-test_hpcugent_develop	18 hr (#54)	23 days (#19)	6.8 sec
●	☀	easybuild-framework_unit-test_hpcugent_master	4 days 16 hr (#6)	N/A	7.3 sec
●	☀	easybuild-full-regtest_develop	4 days 19 hr (#7)	N/A	0.35 sec
●	☀	easybuild-full-regtest_master	6 days 14 hr (#2)	N/A	0.4 sec
●	☀	easybuild-full-regtest_released	4 days 3 hr (#1)	N/A	0.31 sec

http://hpcugent.github.com/easybuild/


unit tests (framework)


unit tests (easyblocks)


unit tests (easyconfigs)

full regression test

List of supported software (v1.11.0)

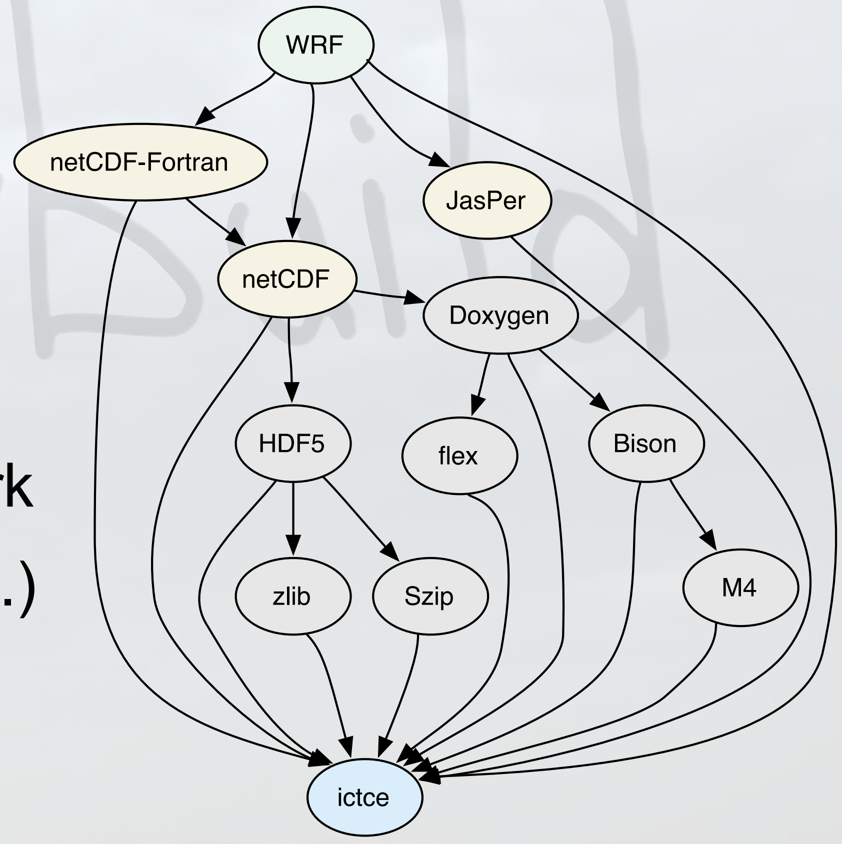
444 different software packages (2,319 example easyconfigs)

a2ps ABAQUS ABINIT ABySS ACML **ALADIN** Allinea ALLPATHS-LG AMOS AnalyzeFMRI ant ARB argtable aria2 Armadillo arpack-ng ASE ATLAS Autoconf Automake bam2fastq BamTools Bash bbcp bbFTP bbftpPRO beagle-lib BEDTools BFAST binutils biodeps BioPerl Biopython BiSearch Bison BLACS BLAST BLAT BOINC Bonnie++ Boost Bowtie Bowtie2 BWA byacc bzip2 cairo CAP3 CBLAS ccache CCfits CD-HIT CDO CFITSIO cflow CGAL cgdb Chapel CHARMM Clang CLHEP Clustal-Omega ClustalW2 CMake Coreutils Corkscrew **CP2K** CPLEX CRF++ Cube CUDA Cufflinks cURL cutadapt CVXOPT Cython DB Diffutils DL_POLY_Classic Docutils **DOLFIN** Doxygen **EasyBuild** ECore ed Eigen ELinks EMBOSS EPD ErlangOTP ESMF ESPResSo expat FASTA fastahack FASTX-Toolkit FCM FDTD_Solutions Ferret FFC FFTW FIAT findutils fixesproto flex FLTK FLUENT fmri FoldX fontconfig FRC_align freeglut FreeSurfer freetype FSL g2clib g2lib GATE GATK gawk **GCC** GDAL GDB Geant4 GenomeAnalysisTK GEOS gettext GHC Ghostscript GIMPS git GLib GLIMMER GLPK glproto GMP GMT gnuplot gnutls google-sparsehash GPAW gperf gperftools Greenlet grep grib_api GROMACS GSL GTI guile gzip h4toh5 h5py h5utils Harminv HDF HDF5 HH-suite HMMER horton HPL HTSeq hwloc Hypre icc ifort imake imkl impi Infernal inputproto Inspector Instant Iperf ipp IPython itac Jansson JasPer Java Jinja2 JUnit kbproto LAPACK lftp libctl libdrm libffi libgtextutils libharu libibmad libibumad libibverbs libICE libidn Libint libint2 libjpeg-turbo libmatheval libpciaccess libpng libpthread-stubs libreadline libSM libsmm 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 MDP Meep MEME Mercurial Mesa Mesquite MetaVelvet METIS Molden Molekel molmod Mothur motif MPFR mpi4py mpiBLAST MPICH MPICH2 MrBayes MTL4 MUMmer MUMPS MUSCLE MUST MVAPICH2 nano NASM NCBI-Toolkit ncdf4 **NCL** ncurses NEdit netCDF netCDF-C++ netCDF-Fortran netcdf4-python netloc nettle **NEURON** nodejs ns numactl numexpr numpy **NWChem** O2scl Oases OCaml Oger OPARI2 OpenBabel OpenBLAS **OpenFOAM** 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 PP Primer3 printproto problog protobuf PSI PyQuante pysqlite pyTables Python python-dateutil python-meep PyYAML PyZMQ QLogicMPI Qt qtop **QuantumESPRESSO** R RAxML RCS RNAz ROOT Rosetta Sablotron SAMtools ScaLAPACK Scalasca ScientificPython scikit-learn scipy SCons SCOOP Score-P SCOTCH SDCC sed setuptools Shapely SHRiMP sickle Silo SLEPc SOAPdenovo Sphinx SQLite Stacks Stow Stride SuiteSparse SURF SWIG sympy Szip TAMkin Tar tbb TCC Tcl tclcl tcsh Tesla-Deployment-Kit 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-mympirun vsc-mympirun-scoop vsc-processcontrol VSC-tools VTK VTune **WIEN2k** wiki2beamer **WPS WRF** xbitmaps xcb-proto XCrySDen xextproto XML XML-LibXML XML-Simple xorg-macros xproto xtrans XZ yaff YamCha YAML-Syck Yasm ZeroMQ zlib zsh zsync

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

```
import fileinput, os, re, sys

import easybuild.tools.environment as env
from easybuild.easyblocks.netcdf import set_netcdf_env_vars
from easybuild.framework.easyblock import EasyBlock
from easybuild.framework.easyconfig import MANDATORY
from easybuild.tools.filetools import patch_perl_script_autoflush, run_cmd, run_cmd_qa
from easybuild.tools.modules import get_software_root

class EB_WRF(EasyBlock):

  def __init__(self, *args, **kwargs):
    super(EB_WRF, self).__init__(*args, **kwargs)
    self.build_in_installdir = True

  @staticmethod
  def extra_options():
    extra_vars = [('buildtype', [None, "Type of build (e.g., dmpar, dm+sm).", MANDATORY])]
    return EasyBlock.extra_options(extra_vars)
```

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)

```python
def configure_step(self):
  # prepare to configure
  set_netcdf_env_vars(self.log)

  jasper = get_software_root('JasPer')
  if jasper:
    jasperlibdir = os.path.join(jasper, "lib")
    env.setvar('JASPERINC', os.path.join(jasper, "include"))
    env.setvar('JASPERLIB', jasperlibdir)

  env.setvar('WRFIO_NCD_LARGE_FILE_SUPPORT', '1')

  patch_perl_script_autoflush(os.path.join("arch", "Config_new.pl"))

  known_build_types = ['serial', 'smpar', 'dmpar', 'dm+sm']
  self.parallel_build_types = ["dmpar", "smpar", "dm+sm"]
  bt = self.cfg['buildtype']

  if not bt in known_build_types:
    self.log.error("Unknown build type: '%s' (supported: %s)" % (bt, known_build_types))

```

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)

```
    # run configure script
    bt_option = "Linux x86_64 i486 i586 i686, ifort compiler with icc"
    bt_question = "\s*(?P<nr>[0-9]+).\s*%s\s*\(%s\)" % (bt_option, bt)

    cmd = "./configure"
    qa = {"(1=basic, 2=preset moves, 3=vortex following) [default 1]:": "1",
          "(0=no nesting, 1=basic, 2=preset moves, 3=vortex following) [default 0]:": "0"}
    std_qa = {r"%s.*\n(.*\n)*Enter selection\s*\[[0-9]+-[0-9]+\]\s*:" % bt_question: "%(nr)s"}

    run_cmd_qa(cmd, qa, no_qa=[], std_qa=std_qa, log_all=True, simple=True)

    # patch configure.wrf
    cfgfile = 'configure.wrf'

    comps = {
             'SCC': os.getenv('CC'), 'SFC': os.getenv('F90'),
             'CCOMP': os.getenv('CC'), 'DM_FC': os.getenv('MPIF90'),
             'DM_CC': "%s -DMPI2_SUPPORT" % os.getenv('MPICC'),
            }

    for line in fileinput.input(cfgfile, inplace=1, backup='.orig.comps'):
        for (k, v) in comps.items():
            line = re.sub(r"^(%s\s*=\s*).*$" % k, r"\1 %s" % v, line)
        sys.stdout.write(line)
```

prepare Q&A for configuring

run configure script autonomously

patch generated configuration file

Use case: WRF - easyblock (3/3)

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

build step function

```
def build_step(self):
  # build WRF using the compile script
  par = self.cfg['parallel']
  cmd = "./compile -j %d wrf" % par
  run_cmd(cmd, log_all=True, simple=True, log_output=True)

  # build two test cases to produce ideal.exe and real.exe
  for test in ["em_real", "em_b_wave"]:
      cmd = "./compile -j %d %s" % (par, test)
      run_cmd(cmd, log_all=True, simple=True, log_output=True)

def install_step(self):
  pass
```

build WRF (in parallel)

build WRF utilities as well

no actual installation step (build in installation dir)

easybuild

Use case: installing WRF

specify build details in easyconfig file (.eb)

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

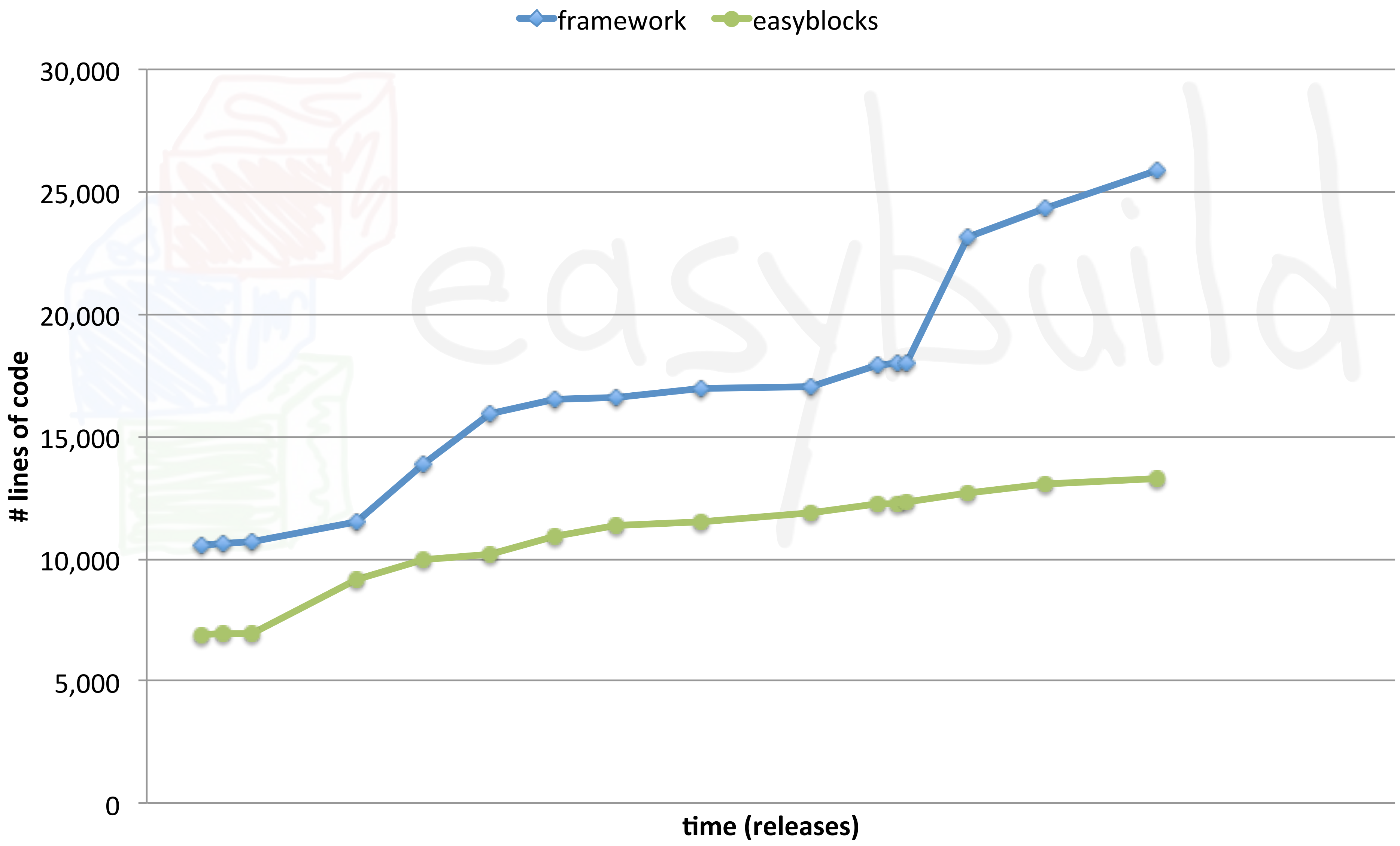
```
name = 'WRF'
version = '3.4'

homepage = 'http://www.wrf-model.org'
description = 'Weather Research and Forecasting'

toolchain = {'name': 'ictce','version': '3.2.2.u3'}
toolchainopts = {'opt': False, 'optarch': False}

sources = ['%sV%s.TAR.gz' % (name, version)]
patches = ['WRF_parallel_build_fix.patch',
           'WRF-3.4_known_problems.patch',
           'WRF_tests_limit-runtimes.patch',
           'WRF_netCDF-Fortran_separate_path.patch']

dependencies = [('JasPer', '1.900.1'),
                ('netCDF', '4.2'),
                ('netCDF-Fortran', '4.2')]

buildtype = 'dmpar'
```

```
eb WRF-3.4-ictce-3.2.2.u3-dmpar.eb --robot
```

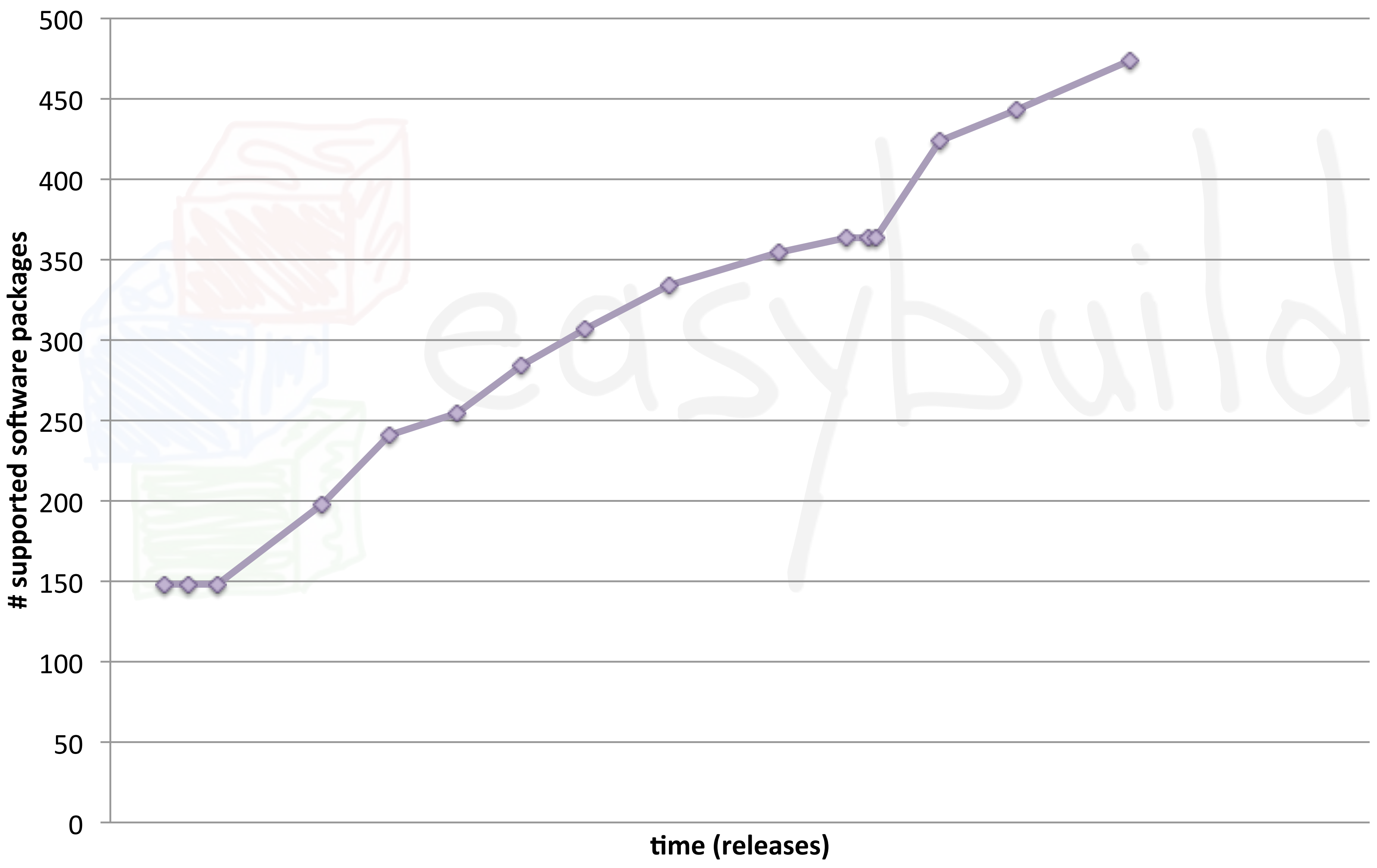
Growth: codebase

# lines of code

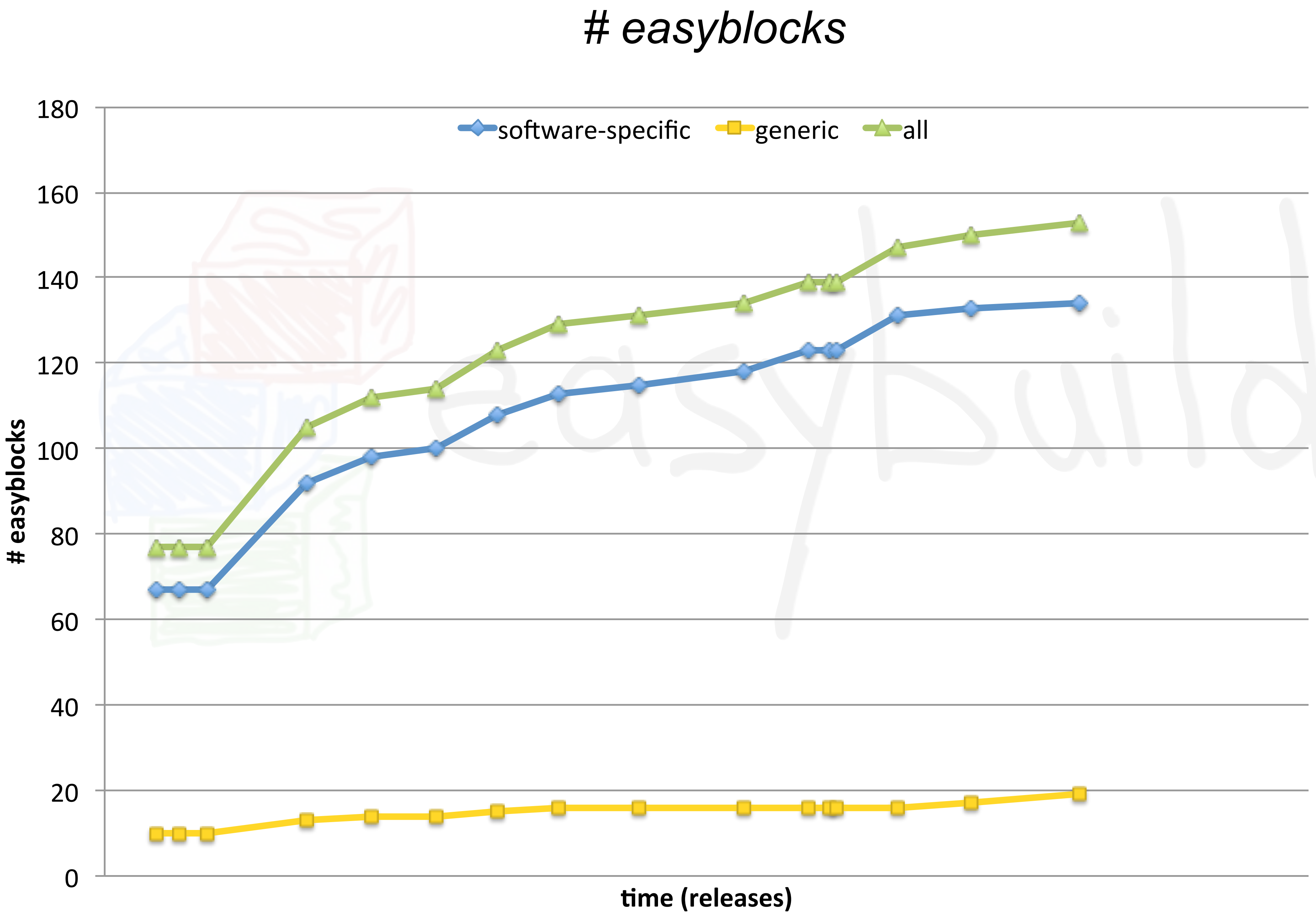
```
find . -name '*.py' | xargs cat | egrep -v '^#|^[ ]*$' | wc -l
```



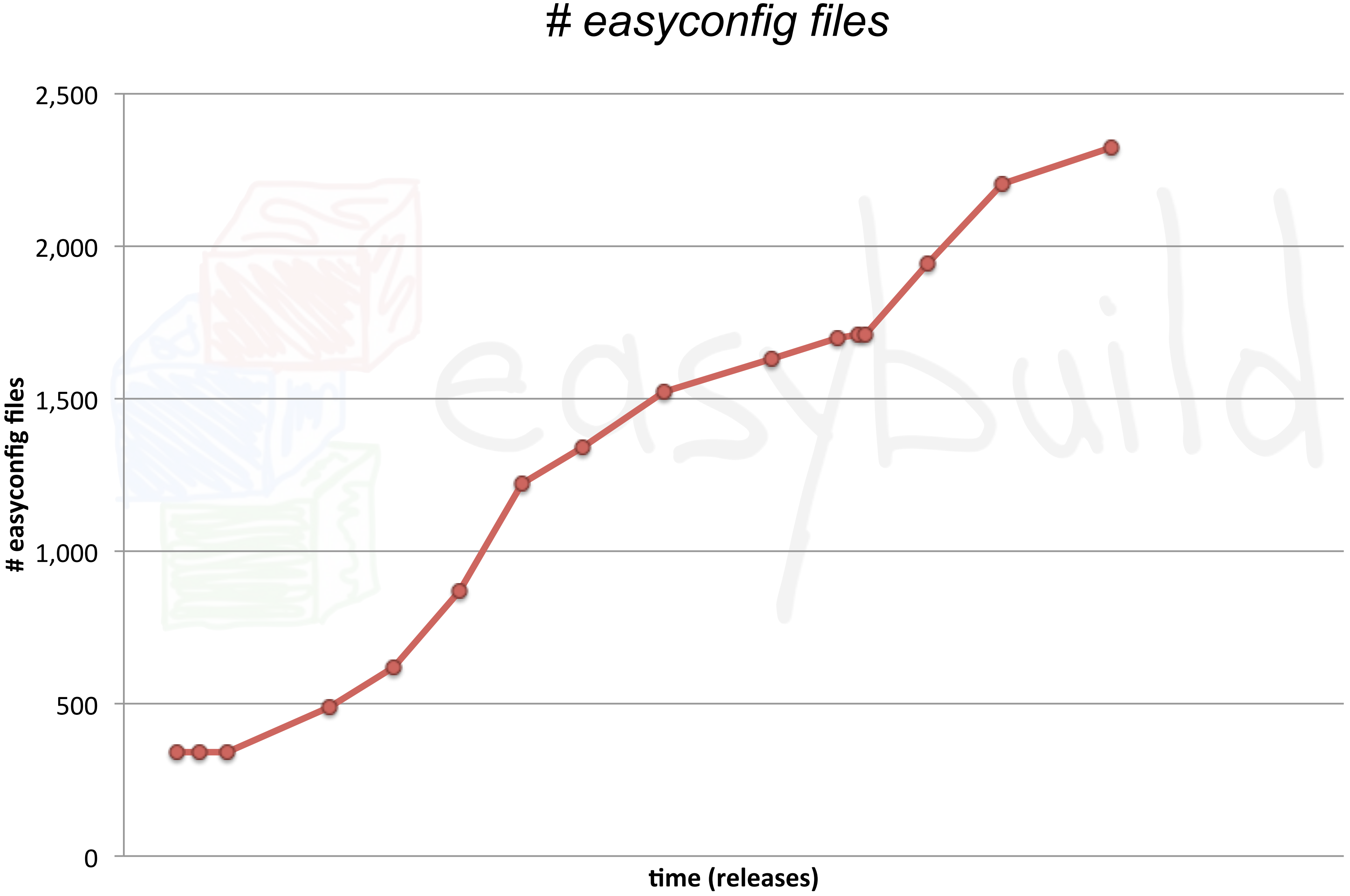
Growth: supported software



Growth: supported software



Growth: supported software



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 Auckland (New Zealand)
- NVIDIA Corp. (cfr. previous hackathon)
- Kiev Polytechnic Institute (NTTU, Ukraine)
- Idaho National Lab (US)
- Jülich Supercomputer Centre (Germany)
- Pacific Northwest National Lab (US)
- UC Davis (US)
- Bayer (Germany)
- ...

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]*

PyPi: *http://pypi.python.org/pypi/easybuild[-framework|-easyblocks|-easyconfigs]*

mailing list: *easybuild@lists.ugent.be*

Twitter: *@easy_build*

IRC: *#easybuild on freenode.net*

easybuild

building software with ease

Jülich Supercomputer Centre (JSC)
Feb 19th 2014

Kenneth Hoste - kenneth.hoste@ugent.be

easybuild@lists.ugent.be

Installing EasyBuild :(

EasyBuild suffers from the mess that is Python packaging...

```
$ easy_install --user easybuild
```

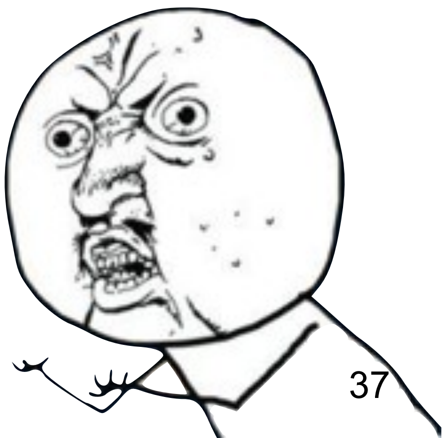
error: option --user not recognized (only for recent versions of easy_install / setuptools)

“You should be using pip!”

```
$ pip install --user easybuild
pip: No such file or directory
```
(pip not installed)

“Just use --prefix with easy_install!”

```
$ easy_install --prefix=$HOME easybuild
$ export PATH=$HOME/bin:$PATH
$ eb --version
ERROR: Failed to locate EasyBuild's main script
```
(`$PYTHONPATH` is not set correctly)



Bootstrapping EasyBuild :)

The easiest way to install EasyBuild is by **bootstrapping** it.

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

```
$ wget http://hpcugent.github.com/easybuild/bootstrap_eb.py
$ python bootstrap_eb.py $HOME
```

This will install EasyBuild using EasyBuild, and produce a module:

```
$ export MODULEPATH=$HOME/modules/all:$MODULEPATH
$ module load EasyBuild
$ eb --version
This is EasyBuild 1.11.0 (framework: 1.11.0, easyblocks: 1.11.0)
```

We're also looking into a packaged release (RPM, .deb, ...).

Configuring EasyBuild

By default, EasyBuild will install software to

`$HOME/.local/easybuild/software`

and produce modules files in

`$HOME/.local/easybuild/modules/all`

You can instruct EasyBuild otherwise by **configuring** it, using:

- a **configuration file**, e.g., `$HOME/.easybuild/config.cfg`
- **environment variables**, e.g., `$EASYBUILD_INSTALLPATH`
- **command line,** e.g. `--installpath`

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