

easybuild

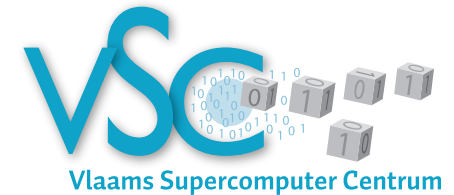
building software with ease

kenneth.hoste@ugent.be



HPC UGent

- ▶ central contact for HPC at Ghent University
- ▶ part of central IT department (DICT)
- ▶ member of Flemish supercomputer centre (VSC)
 - ▶ collaboration between Flemish university associations



- ▶ six Tier2 systems, one Tier1 system
 - ▶ #163 in Top500
- ▶ team consists of 7 FTEs
- ▶ tasks include system administration of HPC infrastructure, user support, user training, ...



Scientific software

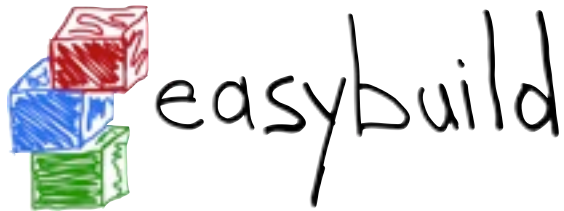
Scientists (generally) spend their time and effort in developing (and testing) their code, not in maintaining it.

Build procedures for scientific software are often:

- ❏ ***incomplete***: e.g., no actual *install* step, only build-in-src-dir
- ❏ ***non-standard***: e.g., requiring human interaction
- ❏ ***customized***: custom scripts for configuration, building, ... instead of configure, cmake, make, etc.
- ❏ ***hard-coded***: no configure option for libraries, compiler commands and flags, ...






Very time-consuming for HPC user support teams!





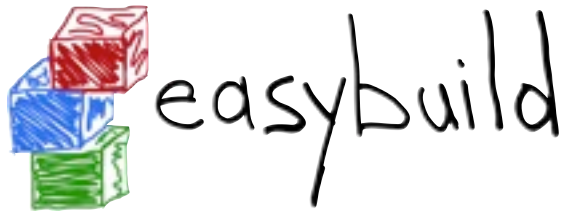
What we need...

flexibility

-  implement funky build procedures with min. effort
-  easily switch to different compiler and/or libraries
-  specify custom compiler options, ...

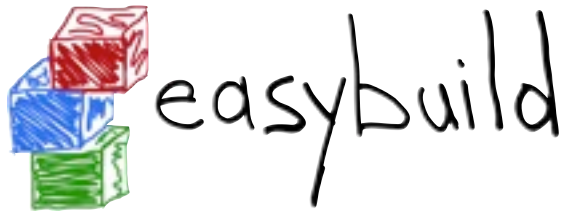
reproducibility

-  easily reproduce exact same build procedure
-  required for version updates, reinstallations, ...



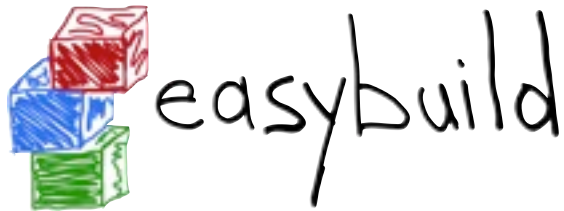
What we need...

- ❏ co-existence of versions/builds
 - ❏ scientists have conflicting requirements
 - ❏ builds available for a long time
 - ❏ latest and greatest version available as well
 - ❏ different build parameters, compilers, ...
 - ❏ e.g., build in prefix path, use environment modules



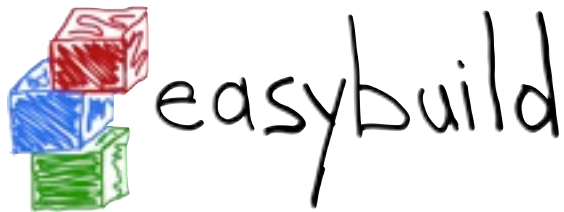
What we need...

- 📦 dependency handling
 - 📦 relieve dependency hell (e.g., DOLFIN)
 - 📦 automate dependency resolution
 - 📦 indispensable in any relevant build framework
 - 📦 cfr. package managers (yum, dpkg, ...)



What we need...

- sharing implementations of install procedures
 - collaborate to tackle this ubiquitous problem
 - enable forming of a community
 - solve the issue once, and share the solution



Current tools are lacking

Package managers like yum and dpkg do not provide a lot of scientific software because of this.
writing .spec files is often a nightmare

Existing tools are:

- ❑ hard to maintain (e.g., huge bash scripts)
- ❑ stand-alone, no reuse of previous efforts
- ❑ OS-dependent (e.g., HomeBrew, Portage, Arch, ...)
- ❑ specific to groups of software packages
(e.g., Dorsal for FEniCS)



easybuild

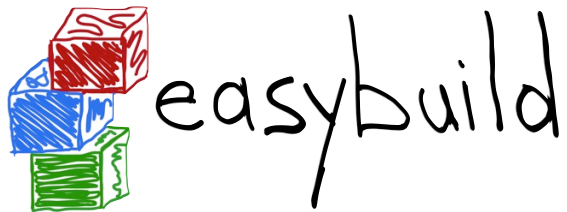
Building software with ease

EasyBuild is a software build and installation framework written in Python.

open-source (GPLv2), available via PyPi and GitHub

It provides:

- a robust framework for implementing build procedures
- lots of supporting functionality
 - extracting, patching, executing shell commands, creating module files, ...
- modular support for compilers, libraries (MPI, BLAS/LAPACK, ...)
- modular support for custom software build procedures



Dependencies

- ❑ Linux (or OS X)
 - ❑ well tested on Red Hat-based systems (SL, Fedora)
 - ❑ also used on Debian, some issues on OS X in v1.0.0
 - ❑ no plans for Windows support yet, should we?
- ❑ Python 2.4 or more recent 2.x
 - ❑ no Python 3.x support yet, but planned
- ❑ environment modules
 - ❑ important bug fix in upcoming 3.2.10 release
- ❑ system C/C++ compiler for building GCC



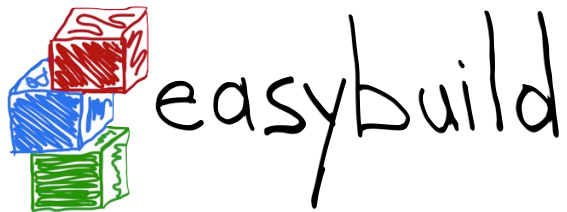
easybuild

Quick demo for the impatient

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

```
$ eb HPL-2.0-goalf-1.1.0-no-OFED.eb --robot
```

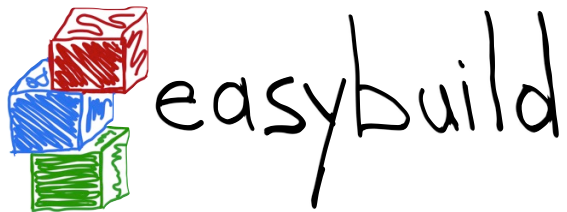
- downloads all required sources
- builds and installs *goalf* compiler toolchain
GCC, OpenMPI, ATLAS, LAPACK, FFTW, BLACS, ScaLAPACK
- builds and installs HPL benchmark (Linpac) with *goalf* toolchain
- default: everything under `$HOME/.local/easybuild`
 - but easy to configure EasyBuild otherwise



EasyBuild terminology

easyblock

- ❏ Python module implementing a build procedure
- ❏ specific to a (group of) software application(s)
- ❏ plugs into EasyBuild, just drop it in Python search path
- ❏ example easyblock later in this talk for WRF

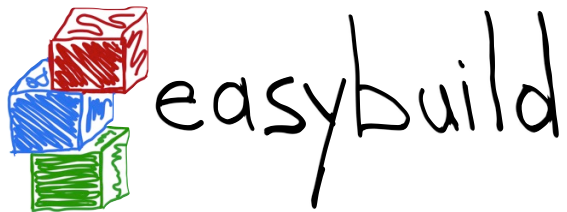


EasyBuild terminology

easyconfig (file)

- ❏ e.g., HPL-2.0-goalf-1.1.0-no-OFED.eb
- ❏ simple text file (Python syntax)
- ❏ specifies software to build, version, build parameters, ...

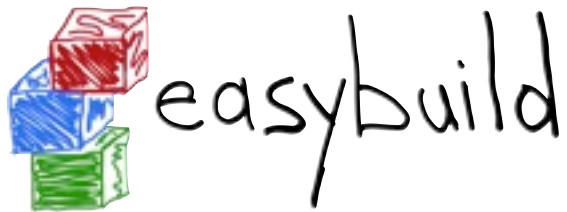
```
1 name = 'HPL'
2 version = '2.0'
3
4 homepage = 'http://www.netlib.org/benchmark/hpl/'
5 description = "High Performance Computing Linpack Benchmark"
6
7 toolchain = {'name': 'goalf', 'version': '1.1.0-no-OFED'}
8 toolchainopts = {'optarch': True, 'usempi': True}
9
10 sources = ['%s-%s.tar.gz' % (name.lower(), version)]
11 source_urls = ['http://www.netlib.org/benchmark/%s' % name.lower()]
12
13 # fix Make dependencies, so parallel build also works
14 patches = ['HPL_parallel-make.patch']
```



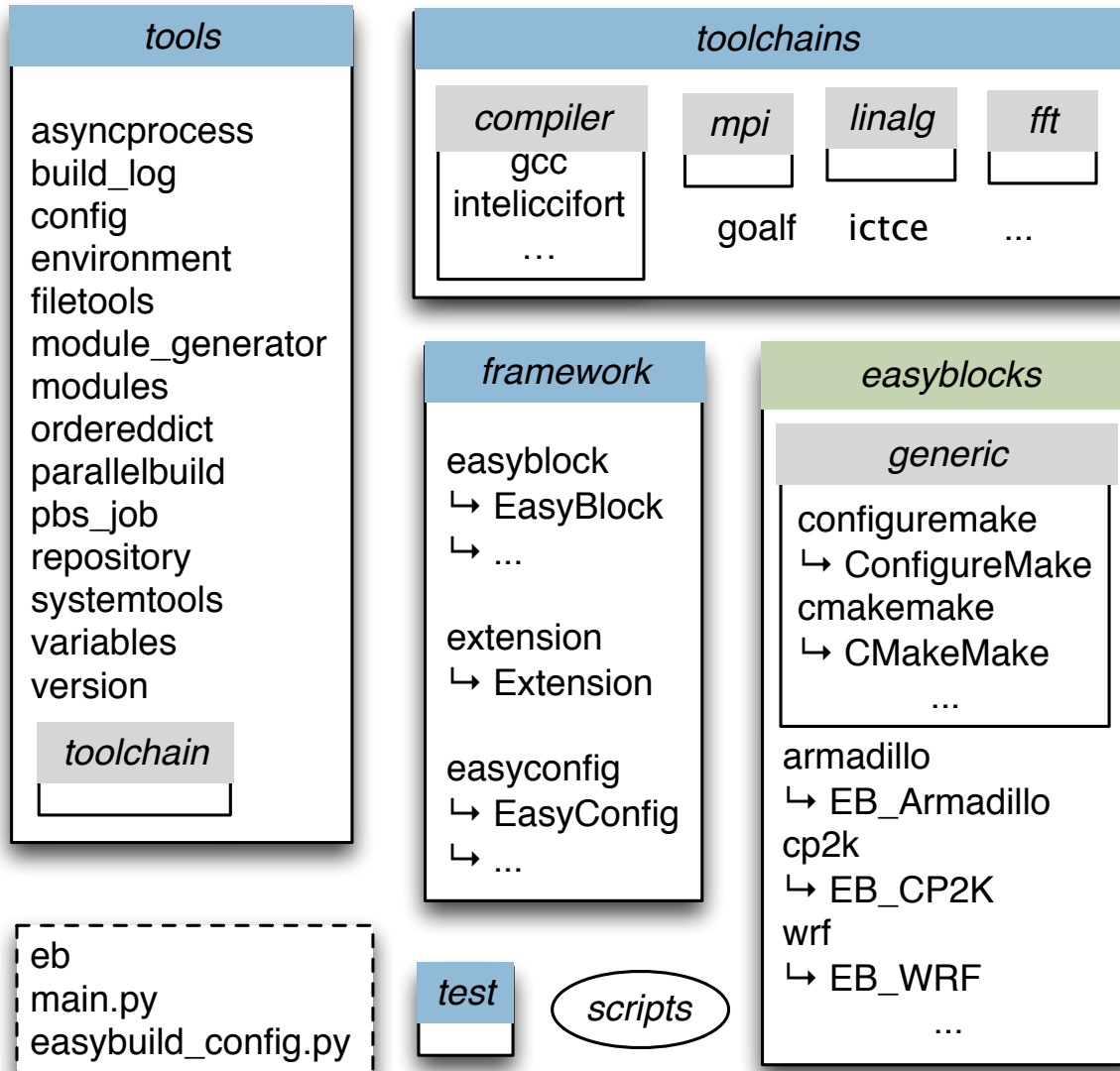
EasyBuild terminology

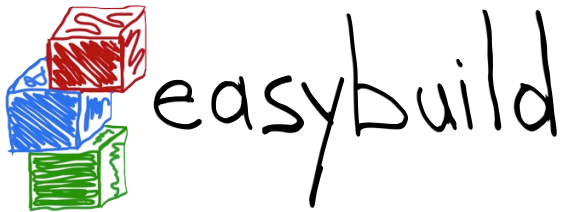
compiler toolchain

- ❏ compiler + set of libraries to support software builds
- ❏ typically MPI, BLAS, LAPACK, FFT libraries for HPC
- ❏ EasyBuild takes care of:
 - ❏ making compiler and libraries available for building
 - ❏ setting environment variables for compiler
 - ❏ CC, MPICC, F90, CFLAGS, CXXFLAGS, etc.
 - ❏ setting include flags, e.g. -I/path/to/include
 - ❏ setting linker flags, e.g. -L/path/to/LAPACK -llapack
- ❏ abstracts away compiler-specific stuff (toolchain options)
 - ❏ e.g. OpenMP (-fopenmp, -openmp, ...)

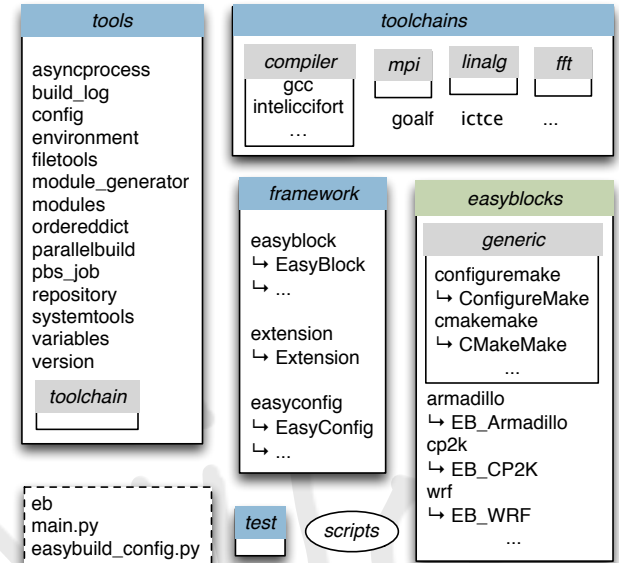


High-level design

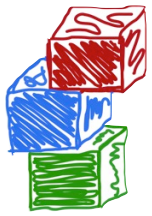




High-level design



easybuild

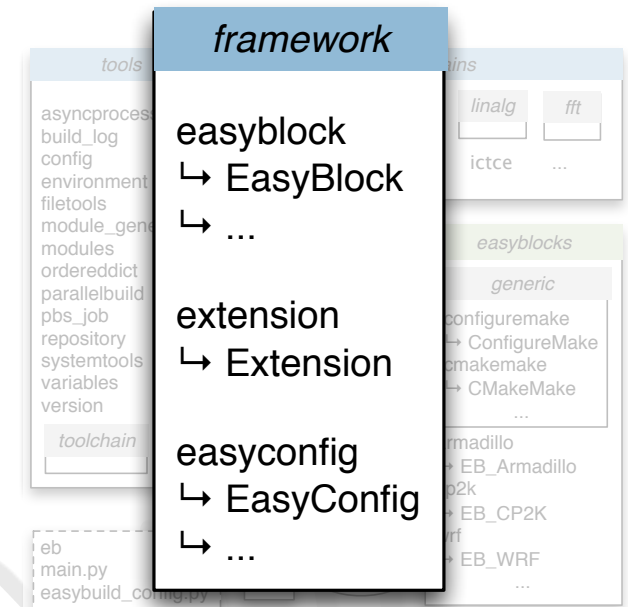


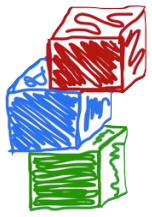
easybuild

High-level design

framework package

- core of EasyBuild
- 'abstract' class Easyblock
 - should be derived
- EasyConfig class
- Extension class
 - e.g. Python packages



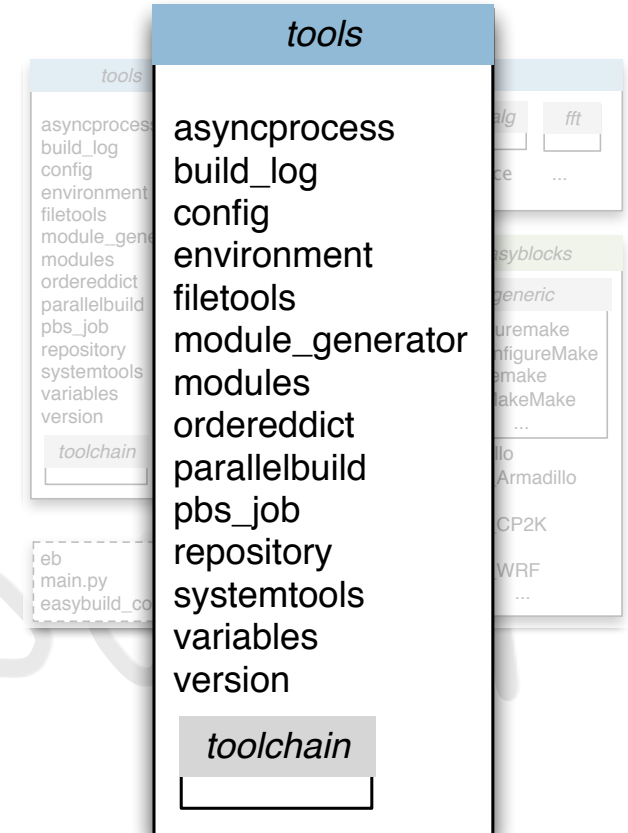


easybuild

High-level design

tools package

- supporting functionality, e.g.:
 - `run_cmd` for shell commands
 - `run_cmd_qa` for interaction
 - `extract_file` for unpacking
 - `apply_path` for patching
- tools.toolchain* for compiler toolchains

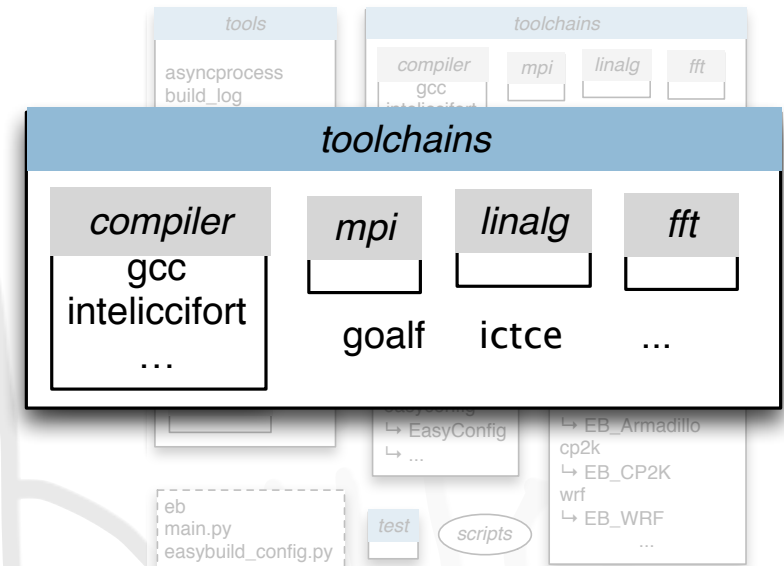




High-level design

toolchains package

- support for compiler toolchains
- relies on *tools.toolchain*
- toolchains are defined in here
- organized in subpackages:
 - toolchains.compiler*
 - toolchains.mpi*
 - toolchains.linalg* (BLAS, LAPACK, ...)
 - toolchains.fft*
- very modularly designed
- plug in a Python module for compiler/library to extend it





High-level design

toolchains package

- support for compiler toolchains

- relies on *tools.toolchain*

- toolchains are defined in here

- organized in subpackages:

 - toolchains.compiler*

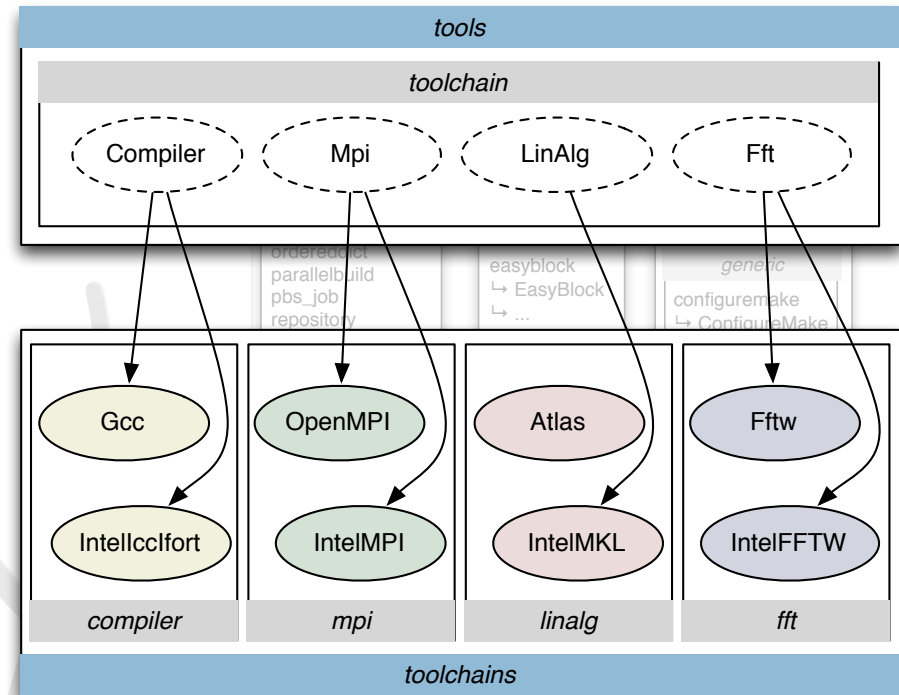
 - toolchains.mpi*

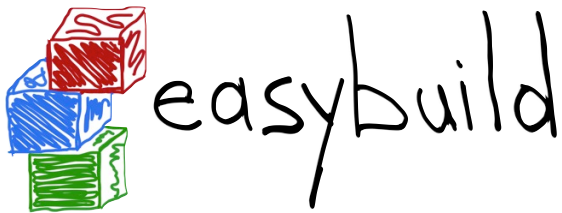
 - toolchains.linalg* (BLAS, LAPACK, ...)

 - toolchains.fft*

- very modularly designed

- plug in a Python module for compiler/library to extend it





High-level design

test package

- unit testing of EasyBuild

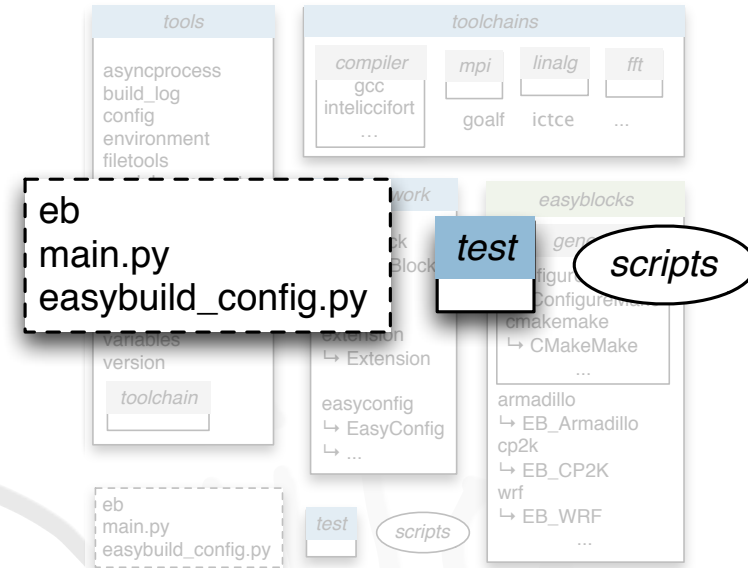
collection of scripts

- mainly for EasyBuild developers

main.py script + eb wrapper

default EasyBuild configuration file

- can be used as a template for your config file

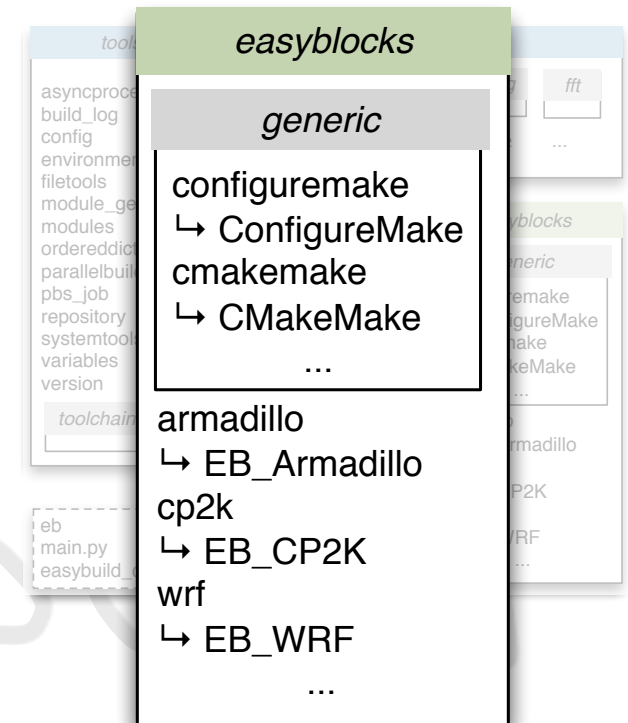


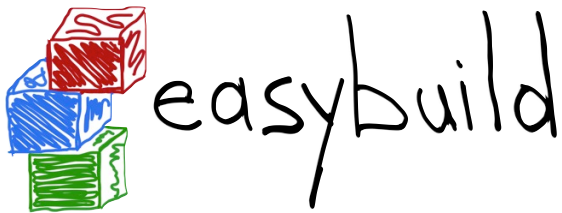


High-level design

easyblocks package

- ❏ build procedure implementations
- ❏ very modular design
 - ❏ add yours in Python search path
 - ❏ EasyBuild will pick it up
- ❏ *easyblocks.generic*: generic *easyblocks*
 - ❏ custom support for groups of applications
 - ❏ e.g. ConfigureMake, CMakeMake, Binary, Tarball, ...
- ❏ application-specific *easyblocks*
- ❏ object-oriented
 - ❏ derive from existing *easyblocks* where appropriate

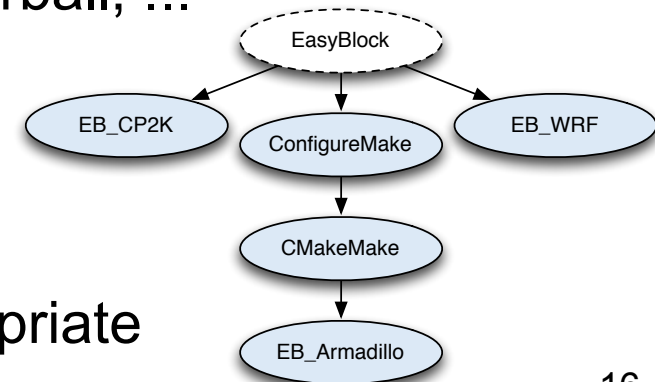
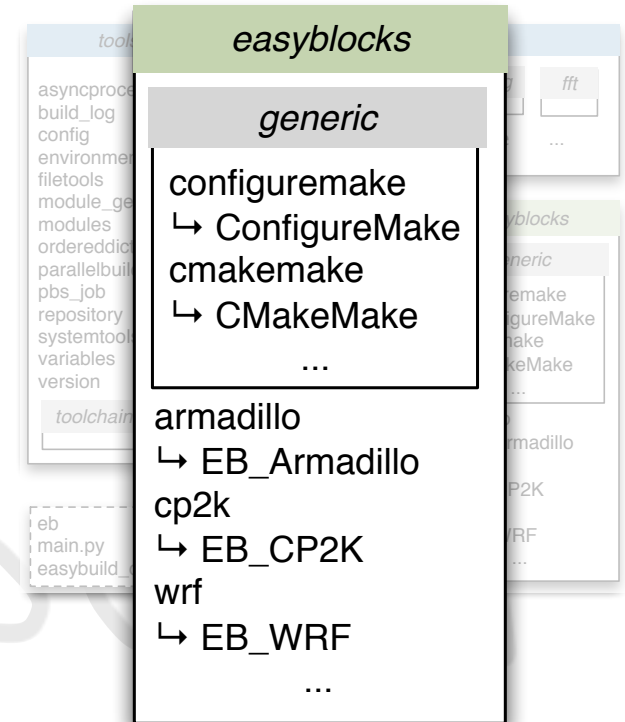




High-level design

easyblocks package

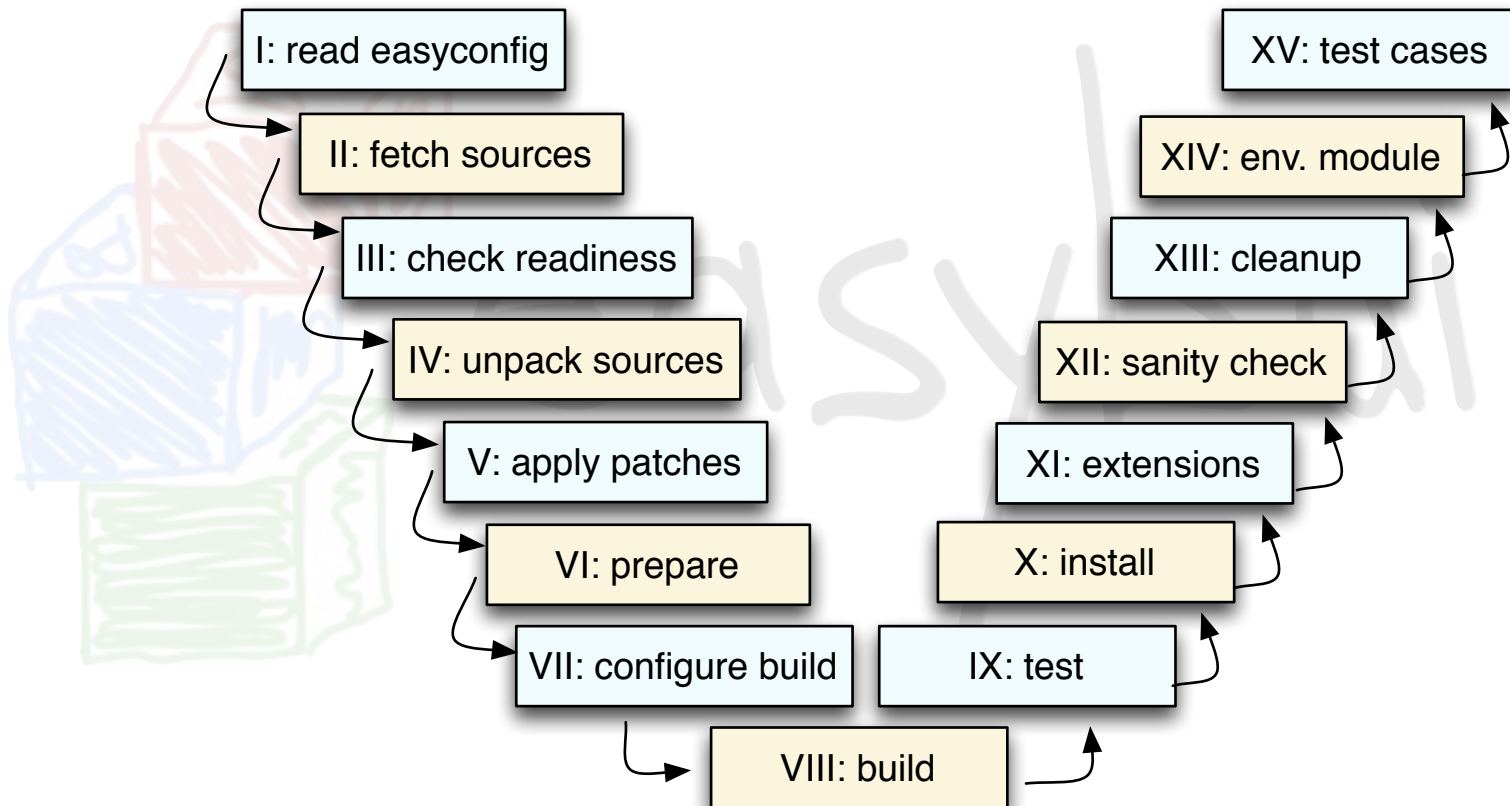
- build procedure implementations
- very modular design
 - add yours in Python search path
 - EasyBuild will pick it up
- easyblocks.generic*: generic easyblocks
 - custom support for groups of applications
 - e.g. ConfigureMake, CMakeMake, Binary, Tarball, ...
- application-specific easyblocks
- object-oriented
 - derive from existing easyblocks where appropriate



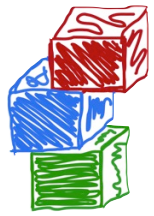


Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customized as needed



easybuild

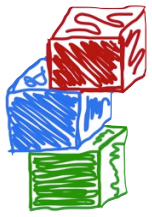
Features: logging and archiving

EasyBuild thoroughly logs the build procedure

- more verbose debug logging enabled with `--debug`
- logs is stored in install directory for future reference

Easyconfig file used for the build is archived

- also stored in install directory
- archived into svn/git/file repository

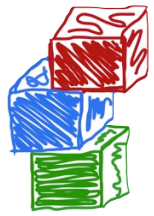


easybuild

Features: dependency resolution



easybuild



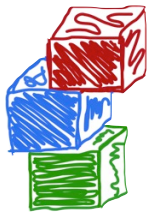
easybuild Features: dependency resolution

automatic dependency resolution

the `--robot` command line option provides support for building a whole software stack with a single command



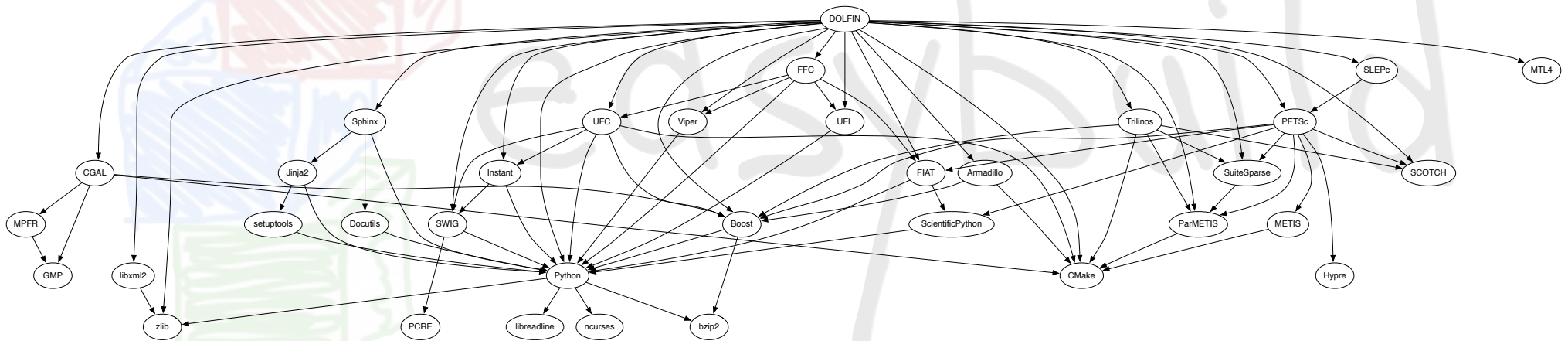
easybuild



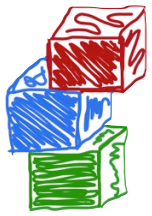
easybuild Features: dependency resolution

automatic dependency resolution

the `--robot` command line option provides support for building a whole software stack with a single command



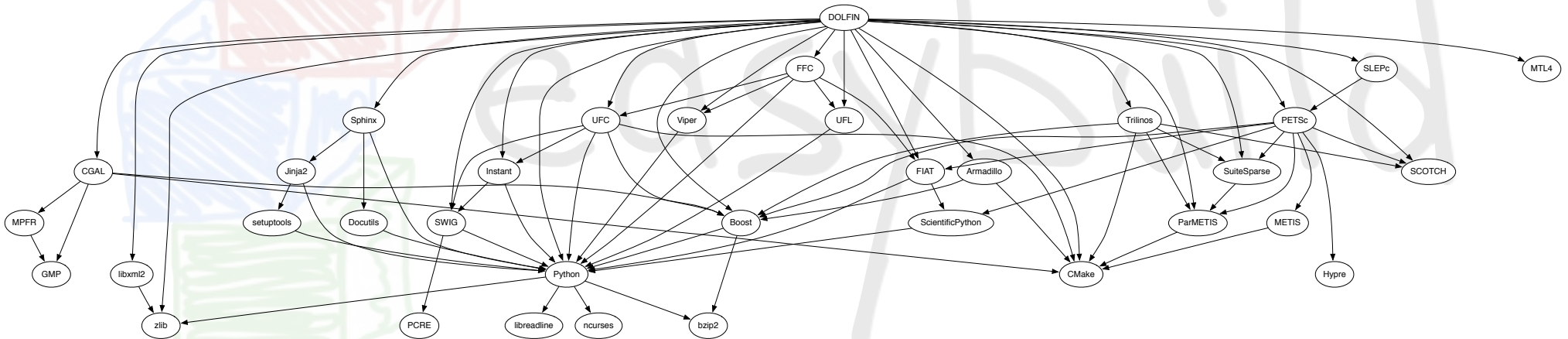
```
eb DOLFIN-1.0.0-ictce-4.0.6-Python-2.7.3.eb --robot
```



easybuild Features: dependency resolution

automatic dependency resolution

the `--robot` command line option provides support for building a whole software stack with a single command



```
eb DOLFIN-1.0.0-ictce-4.0.6-Python-2.7.3.eb --robot
```

or

```
eb --software-name DOLFIN --toolchain-name ictce -r
```



easybuild

Features: interactive installers

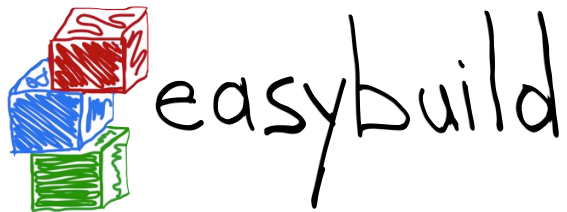
Some software package has a build script that requires human interaction (Q&A).

EasyBuild provides `run_cmd_qa` to handle this, just pass it a question-answer map (dictionary).

```
qa = {  
    '1) Where?': 'SLC',  
    '2) What?': 'SC12',  
    '3) Why?': 'HPC'  
}
```

`run_cmd_qa` will poll for questions and provide answers

will stop build on unknown questions



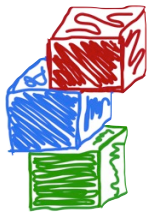
Features: building in parallel

When building a large software stack, you can run builds in parallel, using the `--job` command line option.

Enabled by *tools.parallelbuild* and *tools.pbs_job* packages in EasyBuild.

Jobs will be submitted via Torque, with dependencies set between them to ensure correct build order.

Current regression test (338 builds) builds in 6.5 hours, with a single command.



easybuild

Comprehensive testing



easybuild



Comprehensive testing

unit tests available for the EasyBuild framework

- critical for quality control of the code
- run automatically by Jenkins (continuous integration)

```
$ python -m easybuild.test.suite
.....
-----
Ran 29 tests in 22.837s
OK
Log available at /tmp/easybuild_tests.log
```



Comprehensive testing

unit tests available for the EasyBuild framework

- critical for quality control of the code
- run automatically by Jenkins (continuous integration)

```
$ python -m easybuild.test.suite
.....
-----
Ran 29 tests in 22.837s
OK
Log available at /tmp/easybuild_tests.log
```

regression testing available via `--regtest`

- build ALL available easyconfig files available

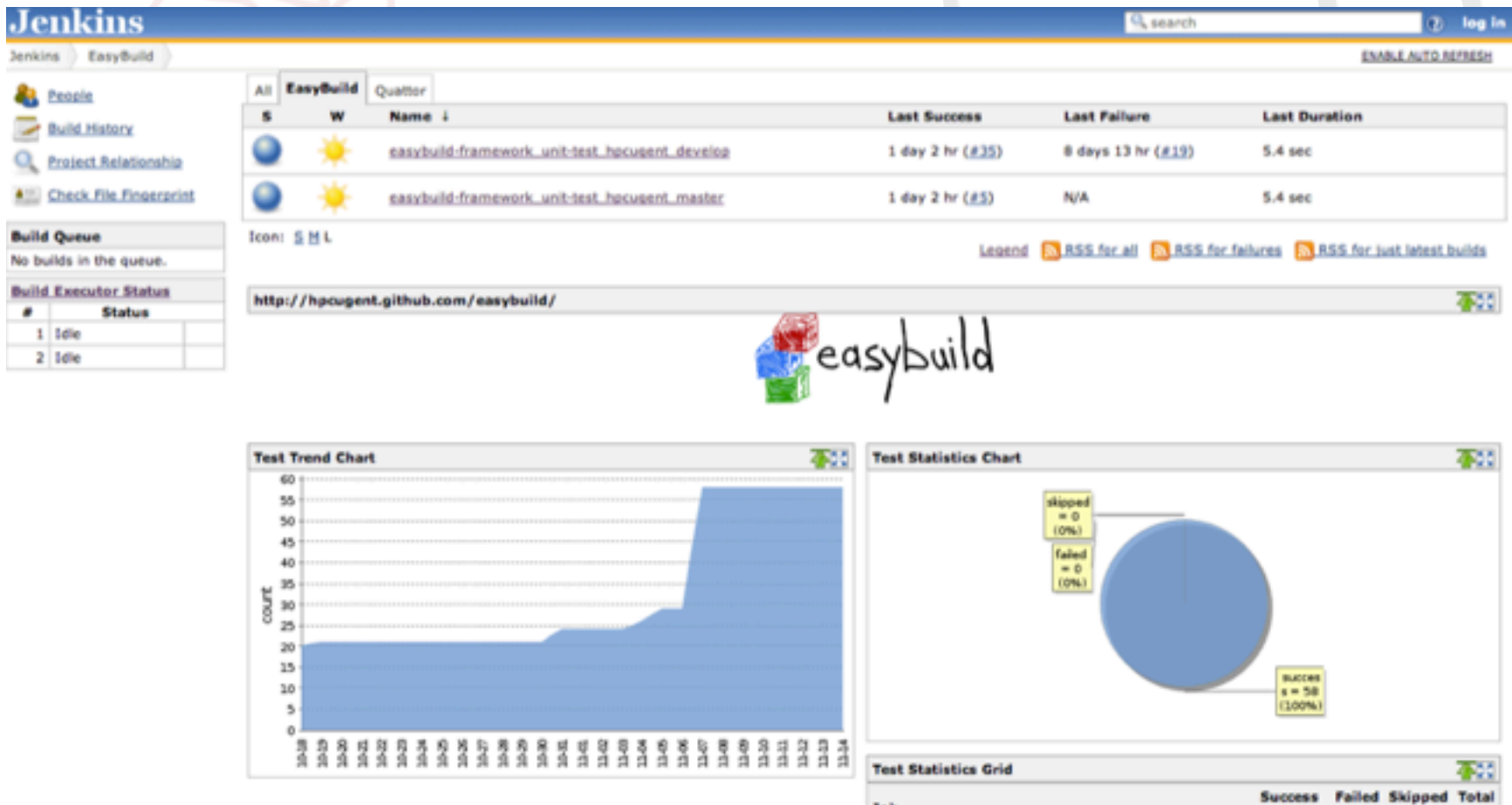


easybuild

Comprehensive testing

unit tests are run automagically by Jenkins

 <https://jenkins1.ugent.be/view/EasyBuild>



Jenkins 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	1 day 2 hr (#35)	8 days 13 hr (#19)	5.4 sec
●	☀	easybuild-framework_unit-test_hpcugent_master	1 day 2 hr (#5)	N/A	5.4 sec

Legend: RSS for all RSS for failures RSS for just latest builds

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

Test Trend Chart

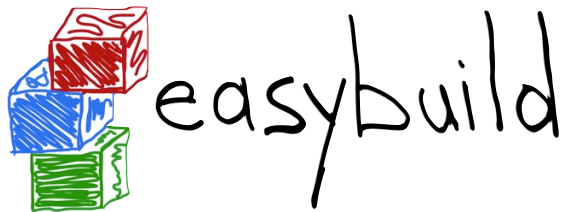
Date	Count
10-18	20
10-19	20
10-20	20
10-21	20
10-22	20
10-23	20
10-24	20
10-25	20
10-26	20
10-27	20
10-28	20
10-29	20
10-30	20
10-31	20
11-01	20
11-02	20
11-03	20
11-04	20
11-05	20
11-06	20
11-07	20
11-08	55
11-09	55
11-10	55
11-11	55
11-12	55
11-13	55
11-14	55

Test Statistics Chart

Category	Count	Percentage
skipped	0	0%
failed	0	0%
success	58	100%

Test Statistics Grid

Task	Success	Failed	Skipped	Total



Use case: WRF

Weather Research and Forecasting Model (WRF)

- very non-standard build procedure
 - interactive `configure` script
 - generates `configure.wrf` file that needs tuning
 - remove hardcoded stuff that you don't want
 - tweak compiler options
 - `compile` script which wraps around `make`
 - no actual installation step
 - need to build in install directory

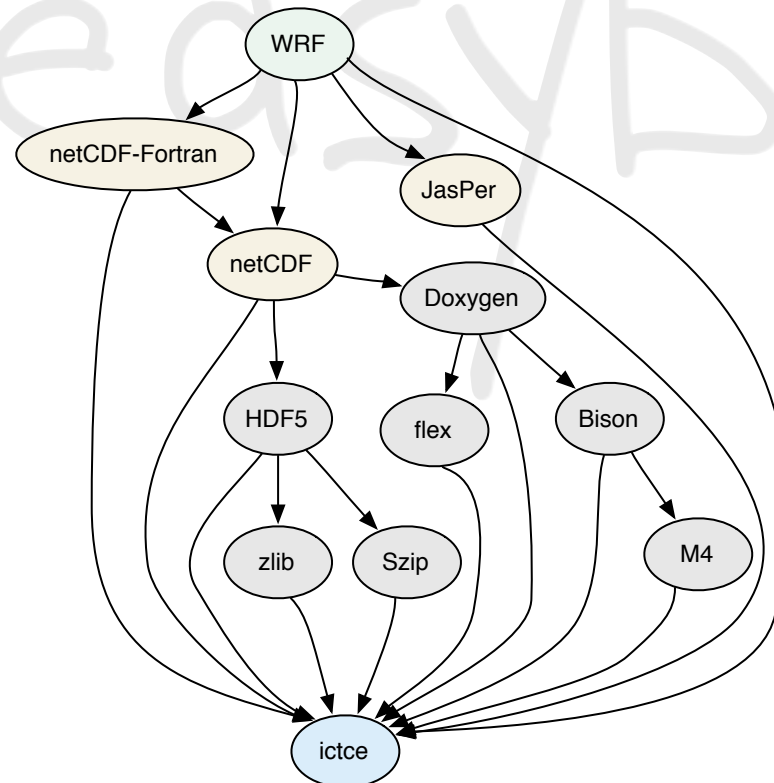


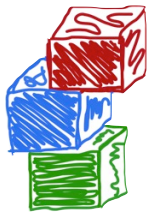
Use case: WRF

Weather Research and Forecasting Model (WRF)

complex(ish) dependency graph

dependencies also feature nasty build procedures

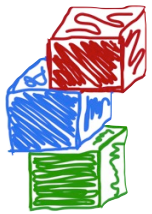




easybuild Use case: WRF - easyblock (1/3)

imports, class constructor,
custom easyconfig parameter

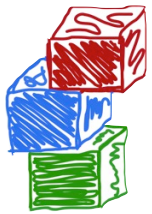
```
1 import fileinput, os, re, sys
2
3 import easybuild.tools.environment as env
4 from easybuild.easyblocks.netcdf import set_netcdf_env_vars
5 from easybuild.framework.easyblock import EasyBlock
6 from easybuild.framework.easyconfig import MANDATORY
7 from easybuild.tools.filetools import patch_perl_script_autoflush, 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
```



easybuild Use case: WRF - easyblock (2/3)

configuration (part 1/2)

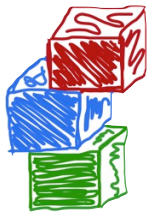
```
21 def configure_step(self):
22     # prepare to configure
23     set_netcdf_env_vars(self.log)
24
25     jasper = get_software_root('JasPer')
26     jasperlibdir = os.path.join(jasper, "lib")
27     if jasper:
28         env.setvar('JASPERINC', os.path.join(jasper, "include"))
29         env.setvar('JASPERLIB', 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
```



easybuild Use case: WRF - easyblock (2/3)

configuration (part 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
```



easybuild Use case: WRF - easyblock (3/3)

build step &
skip install step (since there is none)

```
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
```



Use case: installing WRF

specify build details in easyconfig file

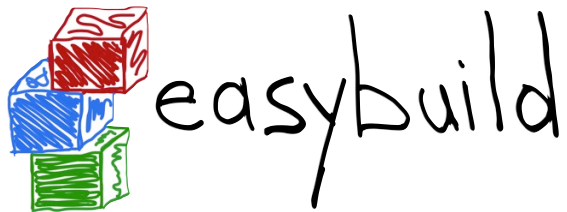
```
1 name = 'WRF'
2 version = '3.4'
3
4 homepage = 'http://www.wrf-model.org'
5 description = 'Weather Research and Forecasting'
6
7 tcver = '3.2.2.u3'
8 toolchain = {'name': 'ictce', 'version': tcver}
9 toolchainopts = {'opt': False, 'optarch': False}
10
11 sources = ['%sV%s.TAR.gz' % (name, version)]
12 patches = [
13     'WRF_parallel_build_fix.patch',
14     'WRF-3.4_known_problems.patch',
15     'WRF_tests_limit-runtimes.patch',
16     'WRF_netCDF-Fortran_separate_path.patch']
17
18 dependencies = [('JasPer', '1.900.1'),
19                 ('netCDF', '4.2'),
20                 ('netCDF-Fortran', '4.2')]
21
22 buildtype = 'dmpar'
```

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



Current status

- developed in-house for over 3.5 years
- available on GitHub (GPLv2) since April 2012
- v1.0.0 (stable API) just released (Nov. 13th 2012)
- support for GCC and Intel compilers, ATLAS, Intel MKL, ...
- custom *easyblocks* available for 77 software packages
 - more being ported from our legacy version in coming weeks/months
- 338 example easyconfigs for 148 different software packages
- used in Scientific Linux (SL) 5/6 day-to-day
- Univ. of Luxembourg uses it on Debian, with great success

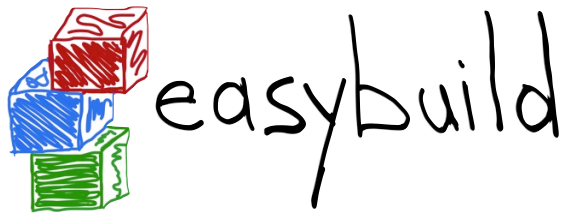


Roadmap

Pending enhancements:

- fix documentation wiki (now bit outdated)
- regression test results available in Jenkins
- flexible module namespace, to tailor it to your setup
- support for Imod
- generate .spec files, RPMs, .deb, etc.
- skipping build procedure steps (e.g. only relink with libs)
- support for more software, more easyblocks/easyconfigs
- support for more compilers (community effort!)

v1.1.0 planned by end of 2012



Contribute!

- see if it fits your needs, do ask questions
- feedback
 - try it, let us know how it goes
 - don't like it? why?
 - what do you need to make it work on your end?
- report problems (mail, GitHub issue, IRC, ...)
- contribute back
 - support for more compilers, libraries for toolchains
 - support for more software by implementing easyblocks
- let's build a community, tackle this together!



easybuild

building software with ease

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

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

IRC: #easybuild @ irc.freenode.net

