

easybuild

building software with ease

VSC tutorial
20130221

kenneth.hoste@ugent.be
jens.timmerman@ugent.be

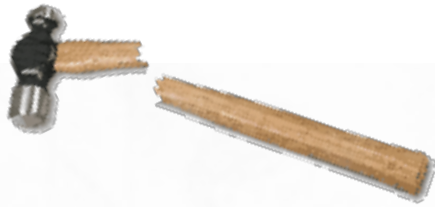
Building scientific software is... fun!

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

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

- ❑ **incomplete** build procedure
- ❑ requiring human **interaction**
- ❑ heavily **customized**
- ❑ use **hard-coded** settings
- ❑ poor and/or outdated **documentation**

Very **time-consuming** for user support teams!



Current tools are lacking

- ❏ building from **source** is preferred in an HPC environment
 - ❏ **performance** is critical, instruction selection is key (e.g. AVX)
- ❏ not a lot of packaged scientific software available (RPMs, ...)
 - ❏ requires **huge effort**, which is duplicated across distros
- ❏ existing build tools are
 - ❏ hard to **maintain** (e.g., bash scripts)
 - ❏ stand-alone, **no reuse** of previous effort
 - ❏ **OS-dependent** (HomeBrew, *Ports, ...)
 - ❏ **custom** to (groups of) software packages
e.g., Dorsal (DOLFIN), gmckpack (ALADIN)



Our build tool wish list

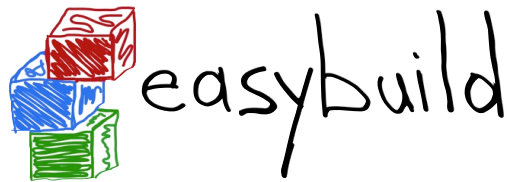
- ▶ **flexible** framework
- ▶ allows for **reproducible** builds
- ▶ supports **co-existence** of versions/builds
- ▶ **automated** builds and **dependency** resolution
- ▶ enables **sharing** of build procedure implementations

Building software with ease



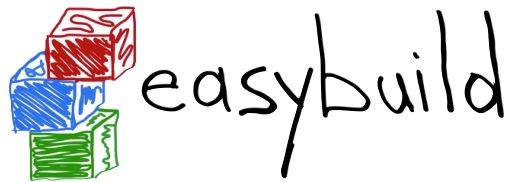
a software build and installation framework

- ❏ written in **Python**
- ❏ developed in-house (HPC-UGent) for 2.5 years
- ❏ **open-source (GPLv2)** since April 2012
- ❏ **stable API** since Nov. 2012 (v1.0.0)
- ❏ continuously enhanced and extended
- ❏ *<http://hpcugent.github.com/easybuild>*



What does EasyBuild need?

- **Linux / OS X**
 - used daily on Scientific Linux 5.x/6.x (Red Hat-based)
 - also tested on Fedora, Debian, Ubuntu, CentOS, SLES, ...
 - a couple of known issues on OS X
 - no Windows support (and none planned for now)
- **Python 2.4** or more recent 2.x
- **environment modules** (lmod support planned)
- system C/C++ compiler to bootstrap a GCC toolchain



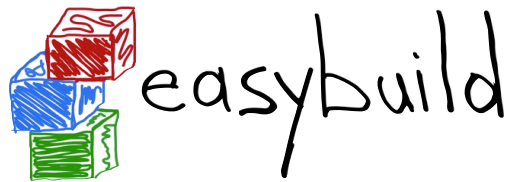
'Quick' demo for the impatient

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

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




- downloads all required sources (best effort)
- constructs *goalf* toolchain, and builds HPL with it
goalf: GCC, OpenMPI, ATLAS, LAPACK, FFTW, ScaLAPACK, BLACS
- default: source/build/install dir in `$HOME/.local/easybuild`

note: we need a better *quick* demo



Terminology



framework

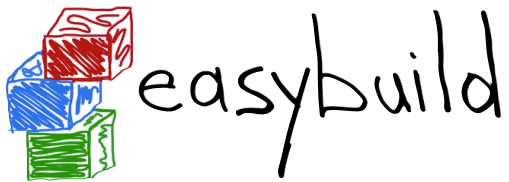
-  Python packages forming the heart of EasyBuild
-  provides (loads of) supporting functionality
-  very modular design w.r.t. toolchains and easyblocks

easyblock

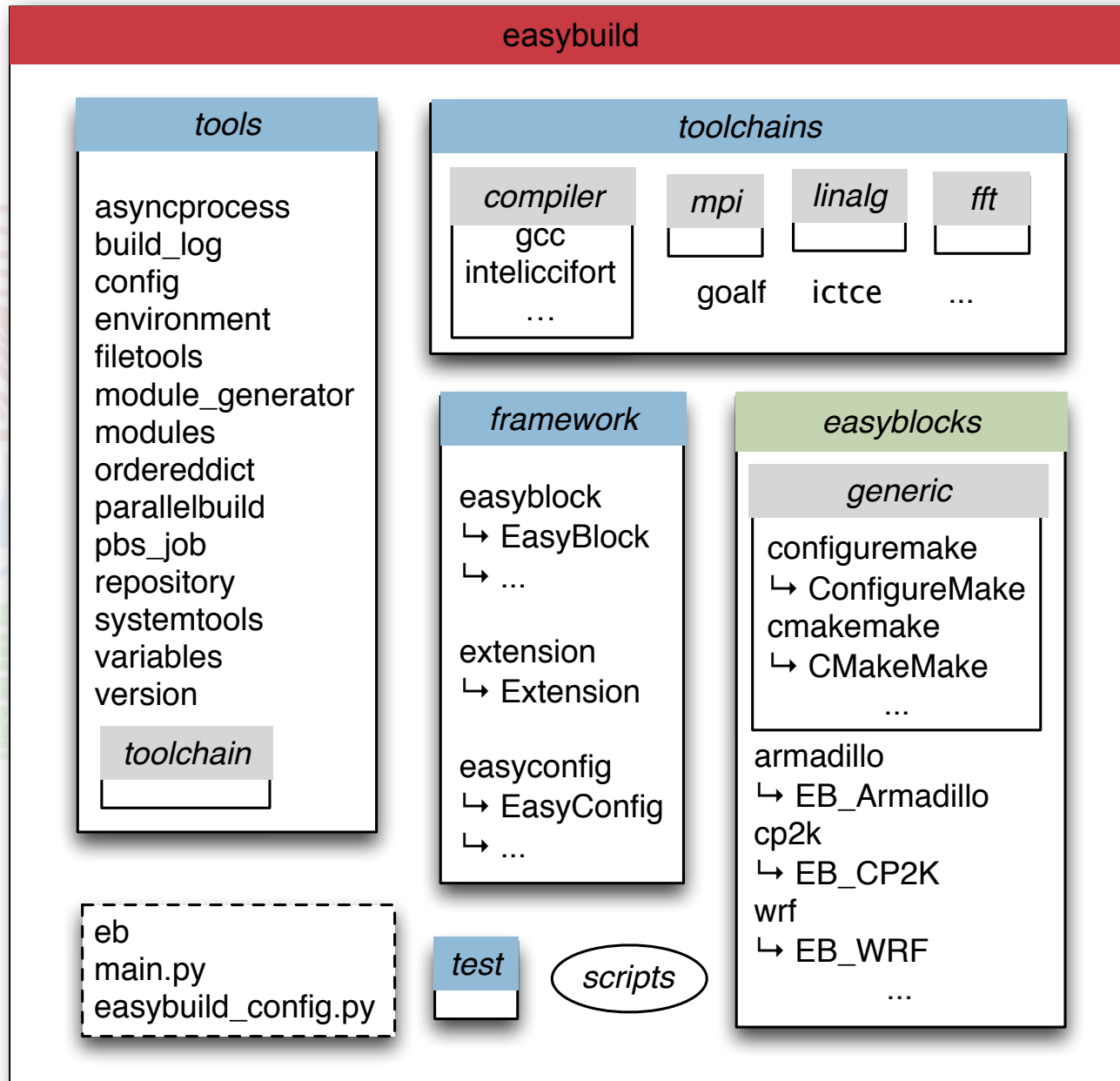
-  Python module providing an implementation of a build procedure
-  generic or software-specific

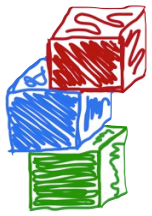
easyconfig file (.eb)

-  build specification: name, version, toolchain, build options, ...
-  simple text files, Python syntax



High-level design



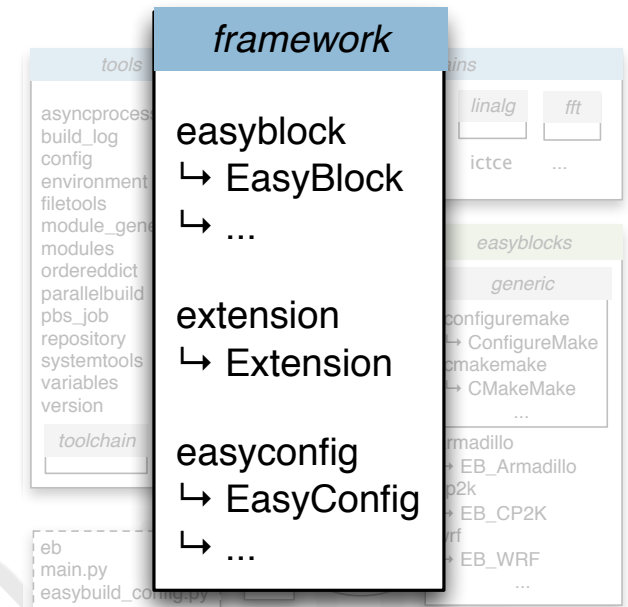


easybuild

High-level design

framework package

- core of EasyBuild
- 'abstract' class Easyblock
 - should be subclassed
- EasyConfig class
- Extension class
 - e.g., to build and install Python packages, R libraries, ...

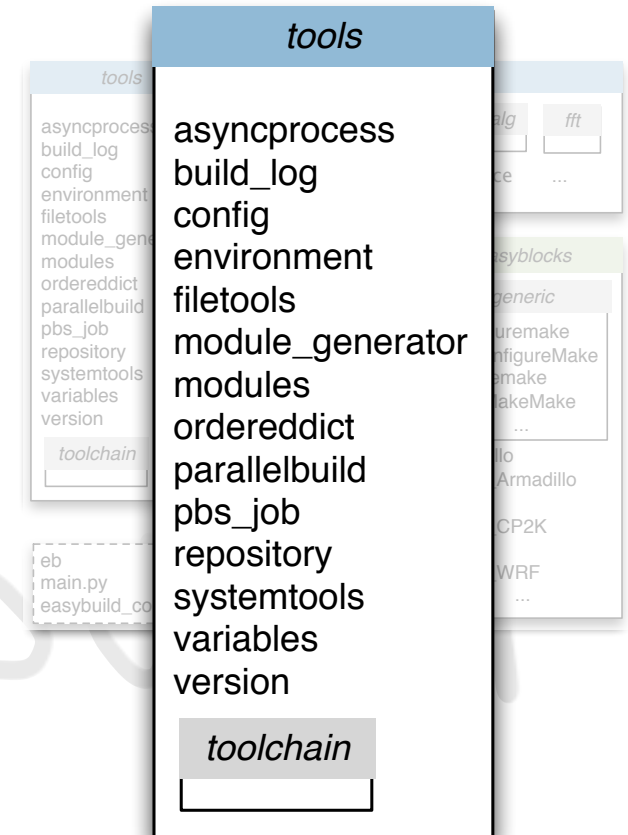


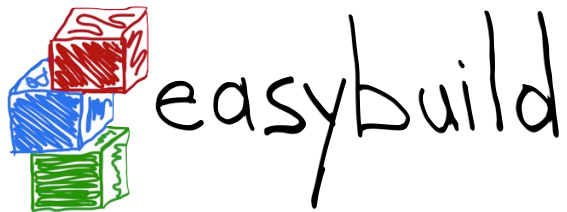


High-level design

tools package

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

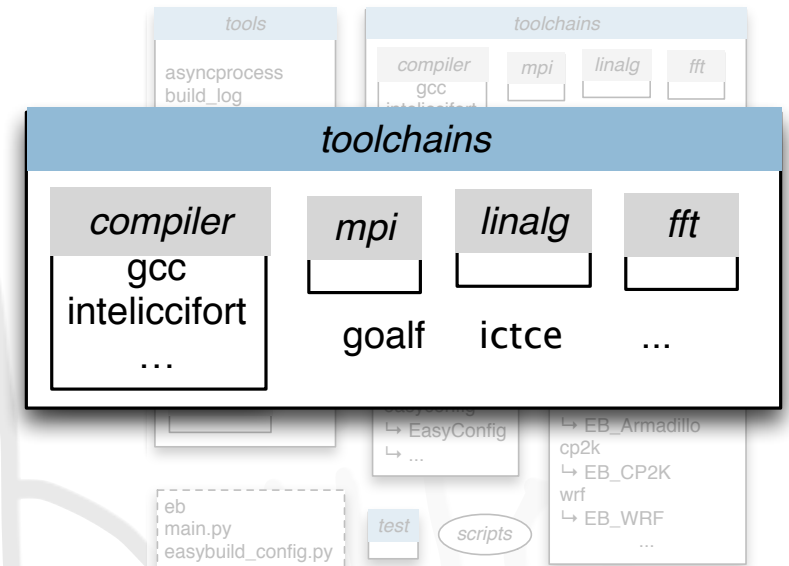


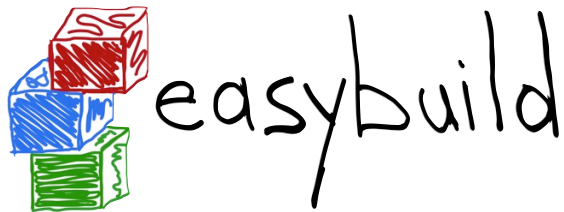


High-level design

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 modular design for allowing extensibility
- 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*

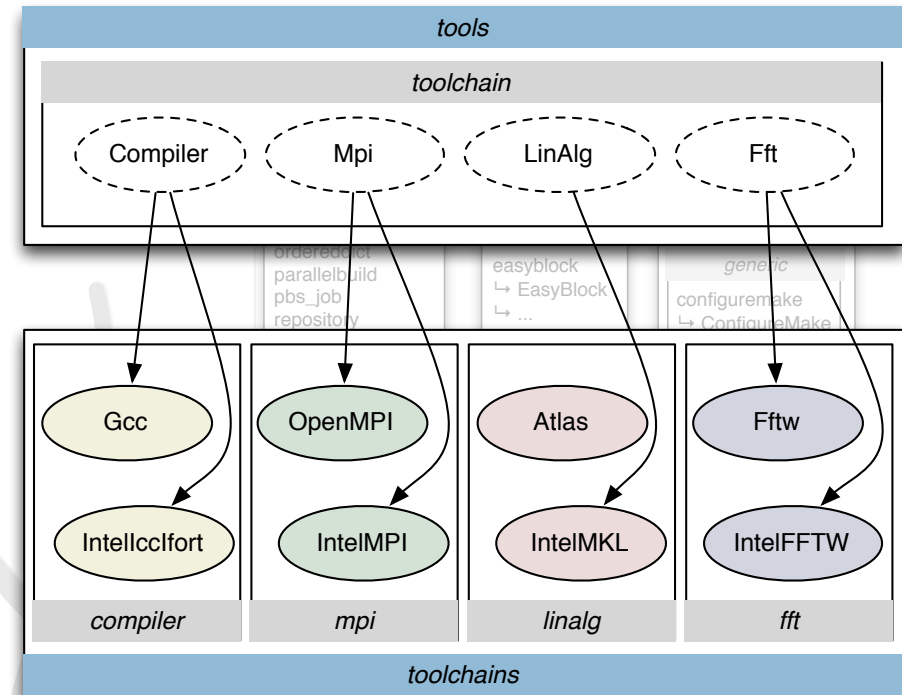
 - toolchains.mpi*

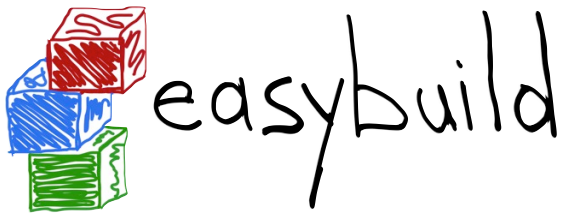
 - toolchains.linalg* (BLAS, LAPACK, ...)

 - toolchains.fft*

- very modular design for allowing extensibility

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





High-level design

test package

- unit testing of EasyBuild

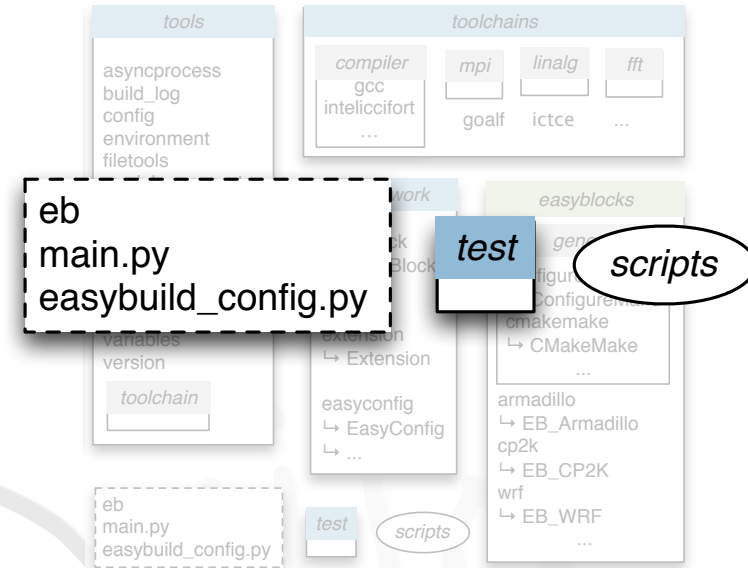
collection of scripts

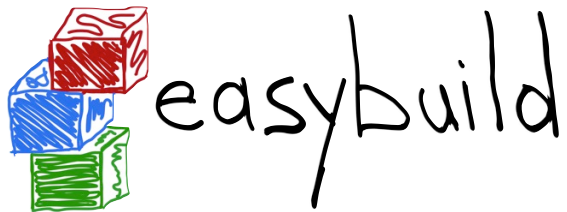
- mainly for EasyBuild developers

main.py script + eb wrapper

default EasyBuild configuration file

- can be used as a template for your config file

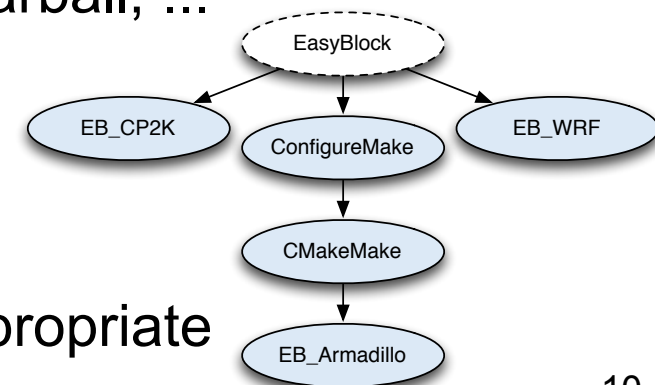
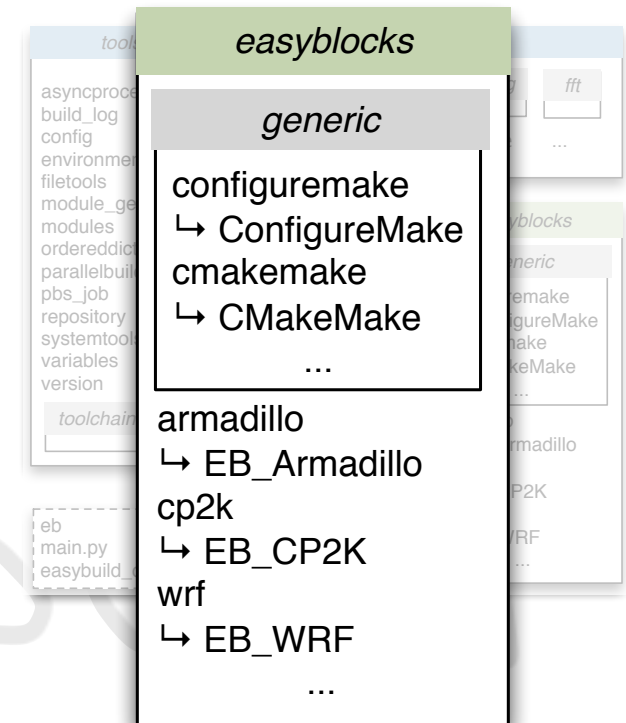


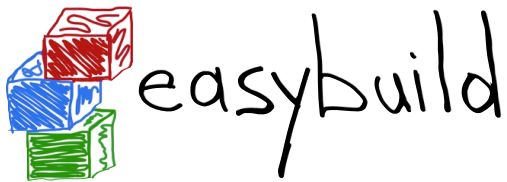


High-level design

easyblocks package

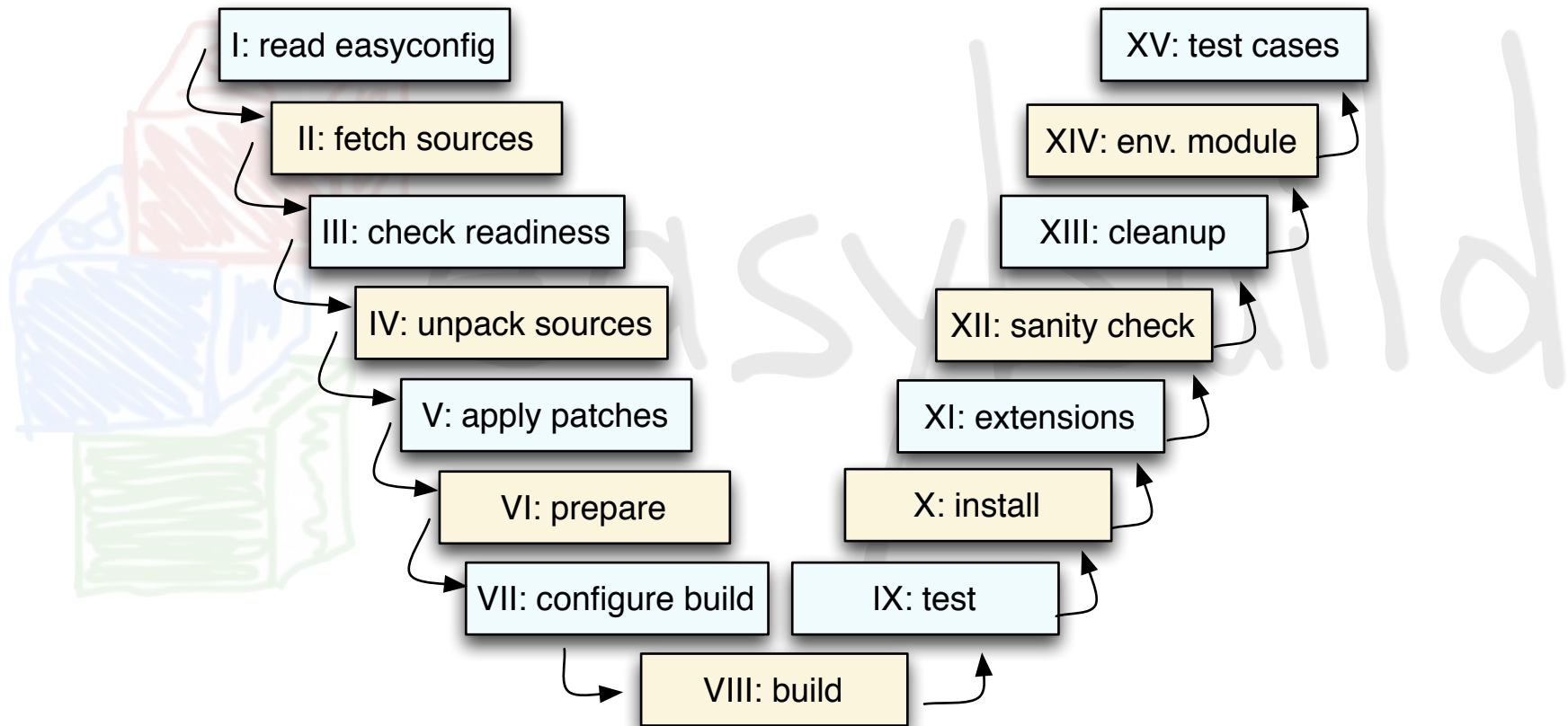
- build procedure implementations
- very modular design
 - add yours in the Python search path
 - EasyBuild will pick it up
- *easyblocks.generic*: generic *easyblocks*
 - custom support for groups of applications
 - e.g., ConfigureMake, CMakeMake, Binary, Tarball, ...
- application-specific *easyblocks*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





Step-wise install procedure

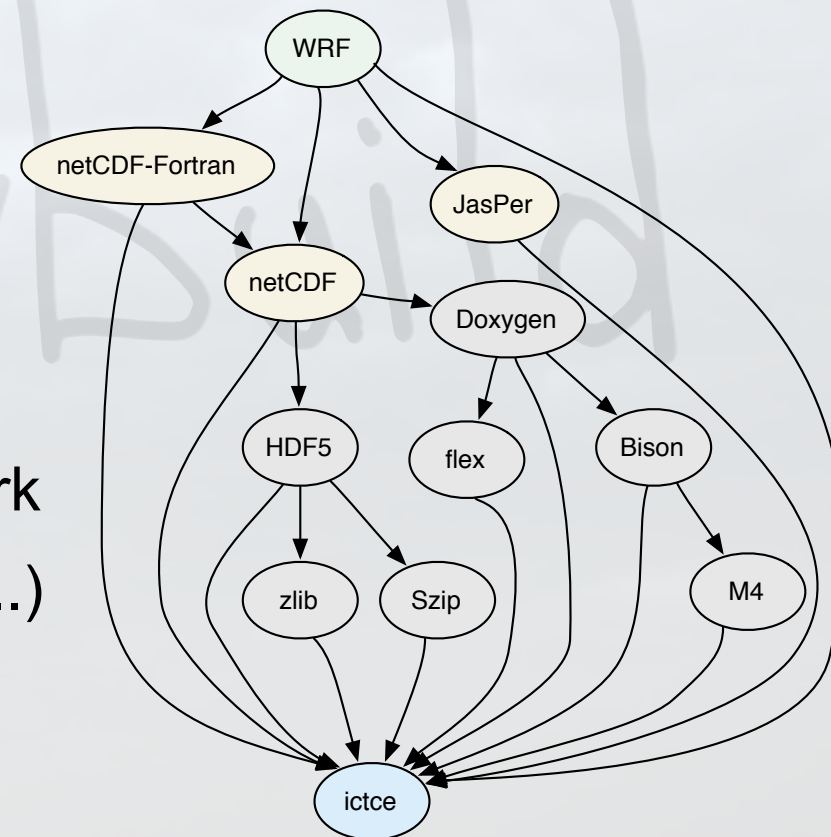
build and install procedure as implemented by EasyBuild



most of these steps can be customized if required

*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



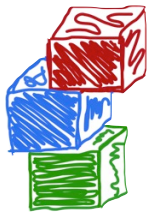


Example use case (2/2)

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

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

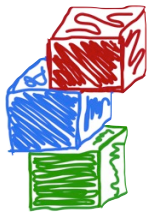
```
eb --software=WRF,3.4 --toolchain-name=ictce --robot
```



Use case: WRF - easyblock (1/3)

imports, class constructor,
custom easyconfig parameter

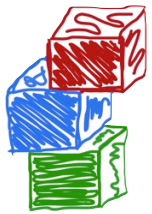
```
1 import fileinput, os, re, sys
2
3 import easybuild.tools.environment as env
4 from easybuild.easyblocks.netcdf import set_netcdf_env_vars
5 from easybuild.framework.easyblock import EasyBlock
6 from easybuild.framework.easyconfig import MANDATORY
7 from easybuild.tools.filetools import patch_perl_script_autoflush,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

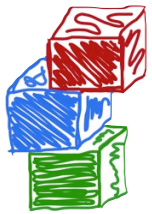
```
21 def configure_step(self):
22     # prepare to configure
23     set_netcdf_env_vars(self.log)
24
25     jasper = get_software_root('JasPer')
26     if jasper:
27         jasperlibdir = os.path.join(jasper, "lib")
28         env.setvar('JASPERINC', os.path.join(jasper, "include"))
29         env.setvar('JASPERLIB',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



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
```



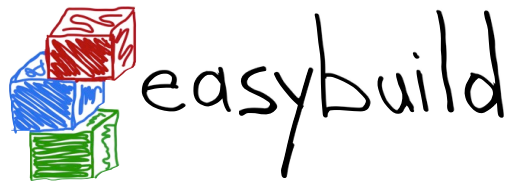
easybuild

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 = ['WRF_parallel_build_fix.patch',
13           'WRF-3.4_known_problems.patch',
14           'WRF_tests_limit-runtimes.patch',
15           'WRF_netCDF-Fortran_separate_path.patch']
16
17 dependencies = [('JasPer', '1.900.1'),
18               ('netCDF', '4.2'),
19               ('netCDF-Fortran', '4.2')]
20
21 buildtype = 'dmpar'
```

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



Features

■ **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**

- software stack be built with a single command, using `--robot`

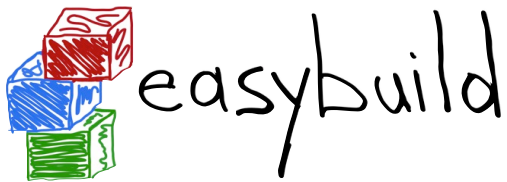
■ **running interactive installers autonomously**

- by passing a Q&A Python dictionary to the `run_cmd_qa` function

■ **building software in parallel**

- e.g., on a (PBS) cluster, by using `--job`

■ **comprehensive testing: unit tests, regression testing**



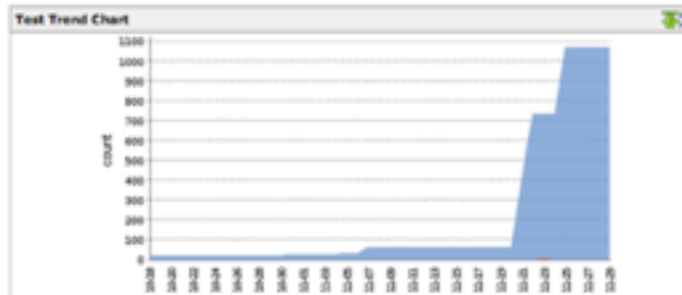
Comprehensive testing

- unit tests are run automagically by Jenkins
- regression test results are pulled in
- <https://jenkins1.ugent.be/view/EasyBuild>

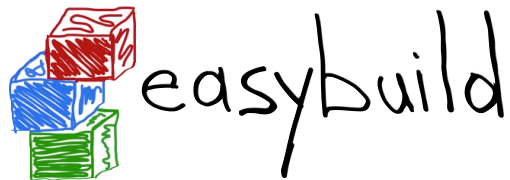
The screenshot shows the Jenkins EasyBuild dashboard. On the left is a sidebar with navigation links: Home, Build history, Project Relationship, and Check File Fingerprint. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status (2 idle). The main area displays a table of build history for the EasyBuild job. The table has columns for Success (S), Warning (W), Name, Last Success, Last Failure, and Last Duration. The builds listed are:

S	W	Name	Last Success	Last Failure	Last Duration
●	☀	easybuild.framework_unit_test_hpcugent_devel	18 hr (#14)	23 days (#13)	6.8 sec
●	☀	easybuild.framework_unit_test_hpcugent_master	4 days 16 hr (#5)	N/A	7.3 sec
●	☀	easybuild.full_regression_devel	4 days 19 hr (#2)	N/A	0.35 sec
●	☀	easybuild.full_regression_master	6 days 14 hr (#2)	N/A	0.4 sec
●	☀	easybuild.full_regression_released	4 days 3 hr (#1)	N/A	0.31 sec

Below the table is a legend: Legend: ● ASS for all, ☀ ASS for failures, ● ASS for just latest builds. At the bottom of the dashboard is the URL <http://hpcugent.github.com/easybuild/>.



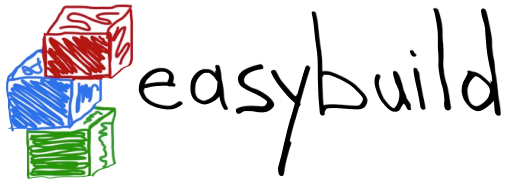
Job	Success		Failed		Skipped		Total
	#	%	#	%	#	%	#
easybuild.framework_unit_test_hpcugent_devel	30	100%	0	0%	0	0%	30
easybuild.framework_unit_test_hpcugent_master	29	100%	0	0%	0	0%	29
easybuild.full_regression_devel	337	100%	0	0%	0	0%	337



List of supported software (v1.1.0)

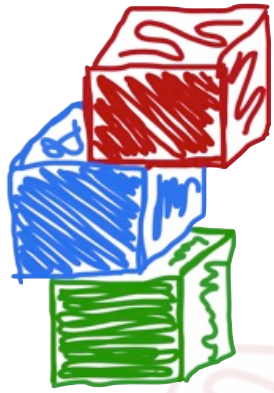
198 different software packages (488 example easyconfigs)

ABINIT ABySS ACML **ALADIN** AMOS ant Armadillo ASE ATLAS Autoconf BEAGLE BFAST
BiSearch Bison BLACS BLAST Boost Bowtie2 BWA byacc bzip2 CGAL Chapel CLHEP
ClustalW2 CMake **CP2K** CPLEX Cufflinks cURL CVXOPT Cython Docutils DOLFIN
Doxygen ECore Eigen ESPResSo expat FASTX-Toolkit FFC FFTW FIAT flex freetype FSL
g2clib g2lib GATE GCC Geant4 GEOS GHC git glproto gmacml GMP gmvapich2 goalf
gompi google-sparsehash GPAW Greenlet grib_api GSL guile h5py h5utils Harminv HDF
HDF5 HMMER HPL hwloc Hypre icc iccifort ictce ifort imkl impi Infernal Instant iomkl ipp itac
JasPer Java Jinja2 JUnit LAPACK libctl libdrm libffi libgtextutils libibmad libibumad libibverbs
Libint libmatheval libpciaccess libpng libpthread-stubs libreadline libsmm libtool libunistring
libxcb libxml2 libxslt M4 makedepend Maple matplotlib MCL MDP Meep MEME Mercurial
Mesa MetaVelvet METIS MPFR mpiBLAST MrBayes MTL4 MUMmer MVAPICH2 NCBI-
Toolkit **NCL** ncurses netCDF netCDF-Fortran NEURON numpy **NWChem** Oger **OpenFOAM**
OpenMPI OpenPGM OpenSSL ORCA PAPI ParMETIS Pasha paycheck PCRE **PETSc**
petsc4py pkg-config Primer3 Python python-meep PyZMQ **QuantumESPRESSO** R RNAz
ROOT SAMtools ScaLAPACK ScientificPython scipy SCOOP SCOTCH setuptools Shapely
SHRiMP SLEPc SOAPdenovo Sphinx SuiteSparse SWIG Szip tbb Tcl Theano Tk Tophat
Tornado Trilinos Trinity UFC UFL util-linux Velvet ViennaRNA Viper VSC-tools **WIEN2k**
wiki2beamer **WPS WRF** xcb-proto XCrySDen XML xorg-macros xproto ZeroMQ zlib



Current status

- ❏ **EasyBuild v1.1.0** released January 27th 2013
 - ❏ planned monthly releases (v1.x.0), bugfix releases as needed
 - ❏ v1.2.0 planned for Feb 28th, *develop* branch now in code freeze
- ❏ various features pending:
 - ❏ **more flexibility**, e.g., module naming scheme, lmod support
 - ❏ bring **documentation** wiki up-to-date
 - ❏ support for **more software** and additional **compiler toolchains**
 - ❏ generate **packages** for supported software (RPMs, .deb, ...)
- ❏ **small community**, growing steadily
 - ❏ UGent + VSC partners
 - ❏ University of Luxembourg
 - ❏ Gregor Mendel Institute (Austria)
 - ❏ Cyprus Institute (next EasyBuild hackathon in Nicosia, March 11-13th)



easybuild

building software with ease

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

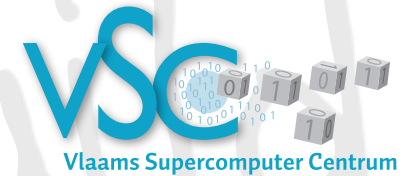
GitHub: [https://github.com/hpcugent/easybuild\[-framework|-easyblocks|-easyconfigs\]](https://github.com/hpcugent/easybuild[-framework|-easyblocks|-easyconfigs])

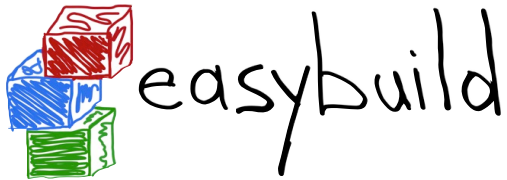
PyPi: [http://pypi.python.org/pypi/easybuild\[-framework|-easyblocks|-easyconfigs\]](http://pypi.python.org/pypi/easybuild[-framework|-easyblocks|-easyconfigs])

mailing list: easybuild@lists.ugent.be

Twitter: @easy_build

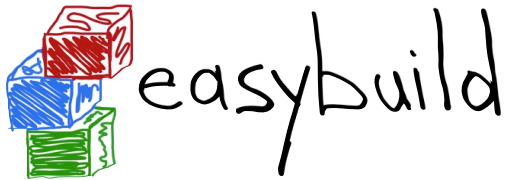
IRC: #easybuild on freenode.net





Module naming scheme

- ❏ currently still hard-wired in EasyBuild
- ❏ more flexibility planned, to allow tailoring to site setup
- ❏ support for customization hopefully by v1.3 (end of March)
- ❏ current status:
 - ❏ trying to understand what is needed
 - ❏ thinking about best way to incorporate this into current design
- ❏ most likely solution:
 - ❏ modular/object-oriented, just plug in a custom naming scheme
 - ❏ customization via self-defined ModuleNamingScheme (sub)class



Module naming scheme

@HPC-UGent: flat (non-hierarchical) naming scheme

<name> / [verpre] **<version>** [-<tcname-tcver>] [<versuf>]

Examples:

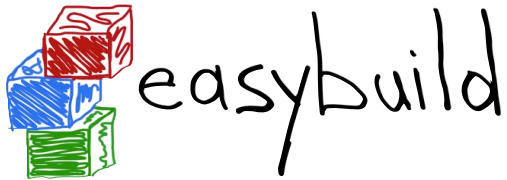
GCC/4.6.3

icc/2013.1.117

ictce/4.0.6

bzip2/1.0.6-ictce-4.0.6

WRF/3.4.1-iqacml-3.7.3-dmpar



Hands-on with v1.2.0dev

Install EasyBuild v1.2.0dev (v1.2.0 pre-release):

```
export X=/your/path
export PYVER=2.7
mkdir -p $X/lib/python$PYVER/site-packages
export PYTHONPATH=$X:$PYTHONPATH
for pkg in framework easyblocks easyconfigs;
do
    easy_install --prefix=$X http://github.com/hpcugent/
    easybuild-$pkg/archive/develop.tar.gz
done
export PATH=$X/bin:$PATH
```