HPC UGENT

UNIVERSITEIT GENT

VSC Vlaams Supercomputer Centrum

easybuild

building software with ease

kenneth.hoste@ugent.be

About 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!

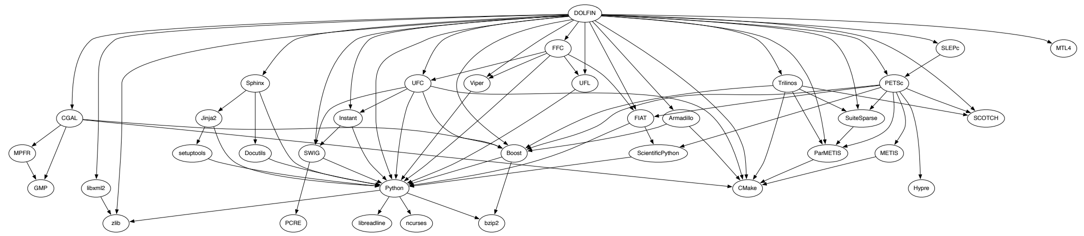
What we need...

- flexibility
  - implement any build procedure with min. effort
  - easily switch between different compilers & libraries
  - specify custom compiler options, install prefix, ...
- 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
  - hide dependency hell (e.g., DOLFIN)
  - automate dependency resolution
  - indispensable in any relevant build framework
  - cfr. package managers (yum, dpkg, portage, ...)

easybuild

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

easybuild

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)

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

easybuild

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

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)
- builds and installs *goalf* compiler toolchain
  GCC, OpenMPI, ATLAS, LAPACK, FFTW, BLACS, ScaLAPACK
- builds and installs HPL benchmark (Linpack) with *goalf* toolchain
- default: everything under $HOME/.local/easybuild
  - easy to configure differently with own config file, environment variables, ...

EasyBuild terminology

framework

- supporting Python packages and Python modules
- provides common required functionality
- enables writing concise plugins (Python modules)
- very modular design, plug-and-play

EasyBuild terminology

compiler toolchain

- compiler + a 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, ...)

EasyBuild terminology

easyblock

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

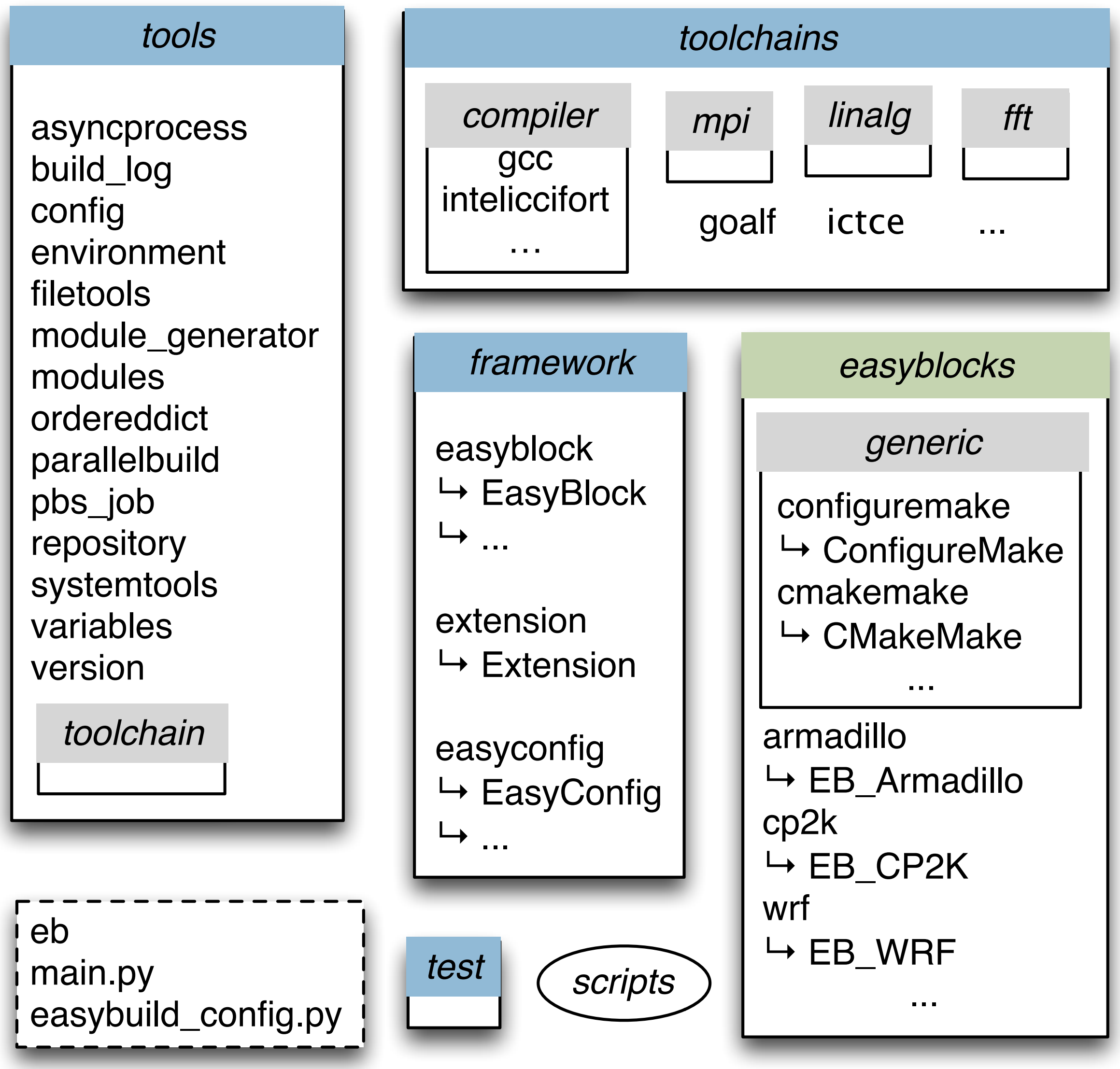
EasyBuild terminology

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, ...

```python
name = 'HPL'
version = '2.0'

homepage = 'http://www.netlib.org/benchmark/hpl/'
description = "High Performance Computing Linpack Benchmark"

toolchain = {'name': 'goalf', 'version': '1.1.0-no-OFED'}
toolchainopts = {'optarch': True, 'usempi': True}

sources = ['%s-%s.tar.gz' % (name.lower(), version)]
source_urls = ['http://www.netlib.org/benchmark/%s' % name.lower()]

# fix Make dependencies, so parallel build also works
patches = ['HPL_parallel-make.patch']
```

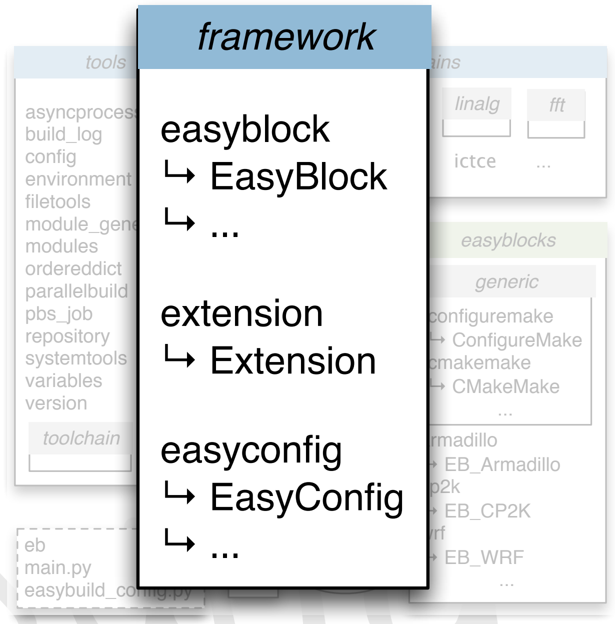
High-level design



High-level design

framework package

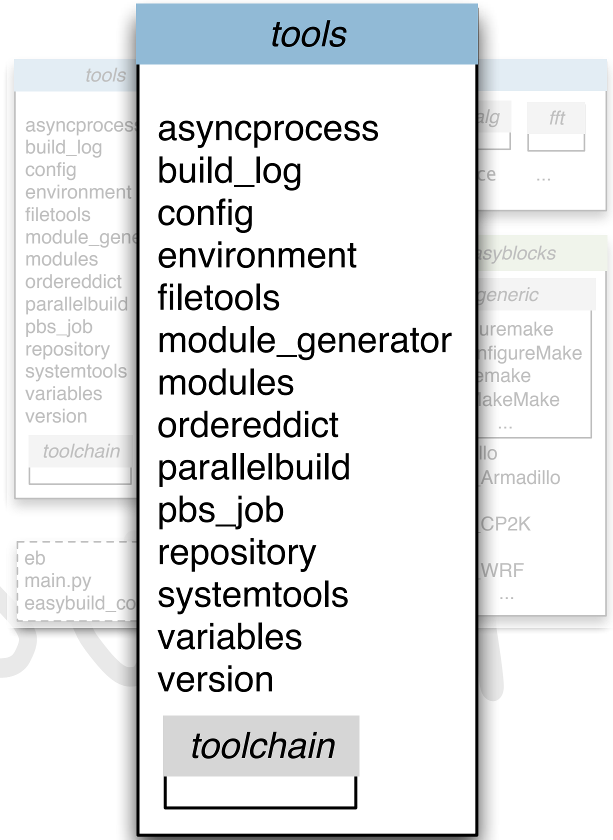
- core of EasyBuild
- 'abstract' class Easyblock
  - should be subclassed
- EasyConfig class
- Extension class
  - e.g., to build and install Python packages, R libraries, ...



High-level design

tools package

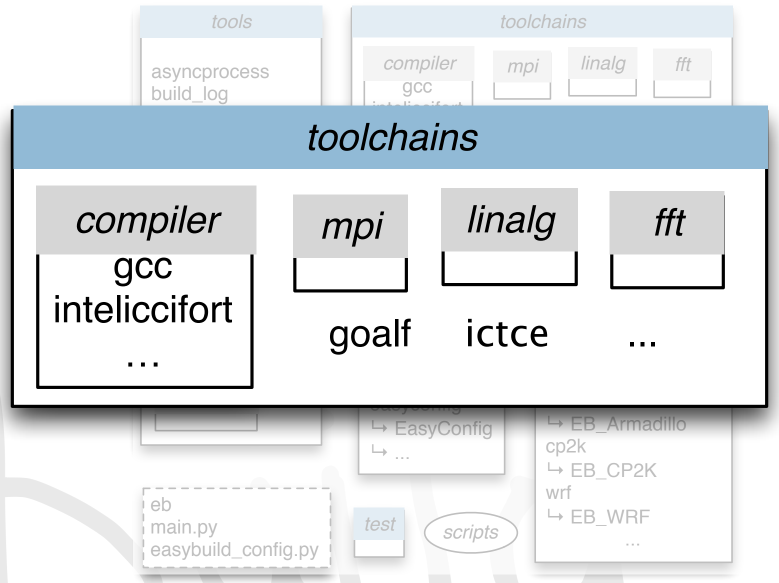
- supporting functionality, e.g.:
  - `run_cmd` for shell commands
  - `run_cmd_qa` for interaction
  - `extract_file` for unpacking
  - `apply_patch` for patching
- *tools.toolchain* for compiler toolchains



High-level design

toolchains package

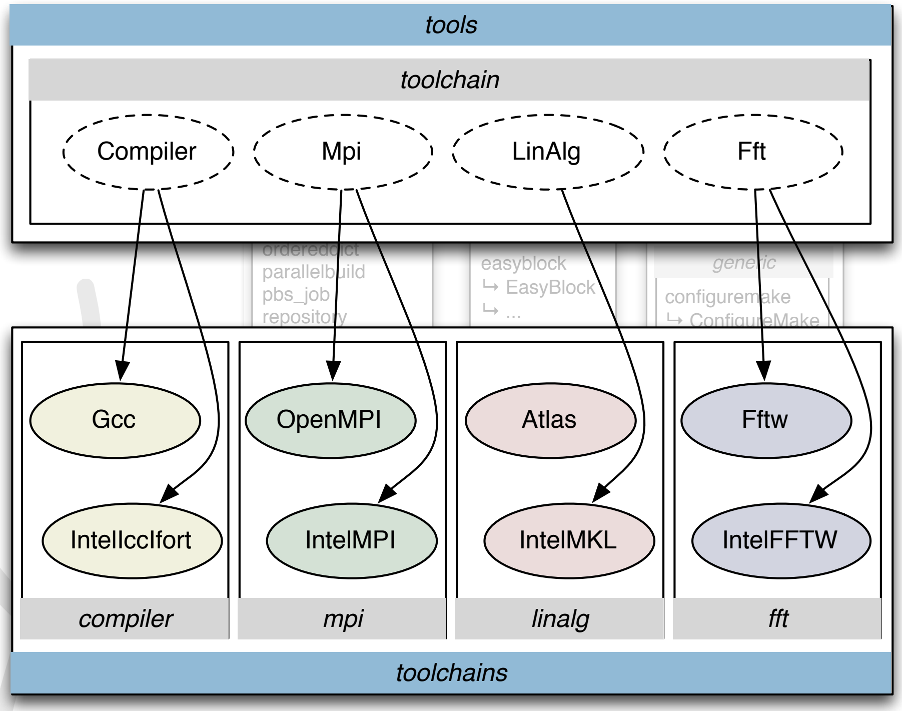
- support for compiler toolchains
- relies on *tools.toolchain*
- toolchains are defined in here
- organized in subpackages:
  - *toolchains.compiler*
  - *toolchains.mpi*
  - *toolchains.linalg* (BLAS, LAPACK, ...)
  - *toolchains.fft*
- very modular design for allowing extensibility
- plug in a Python 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

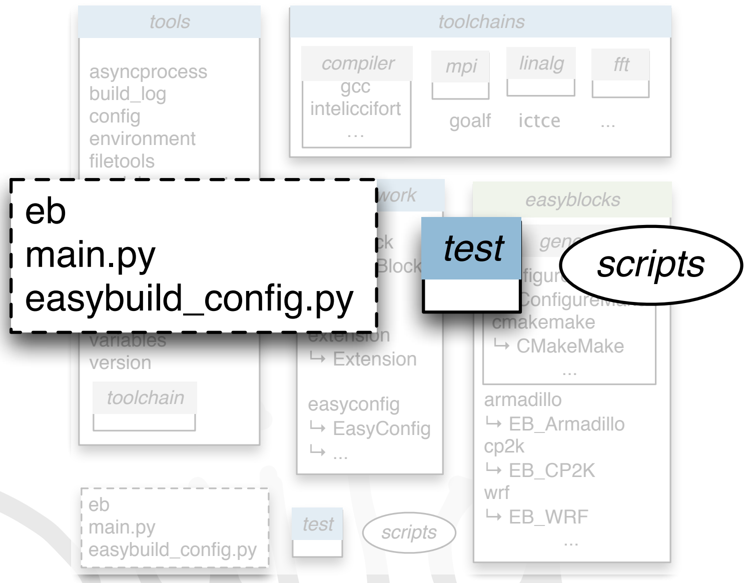
collection of scripts

- mainly for EasyBuild developers

main.py script + *eb* wrapper

default EasyBuild configuration file

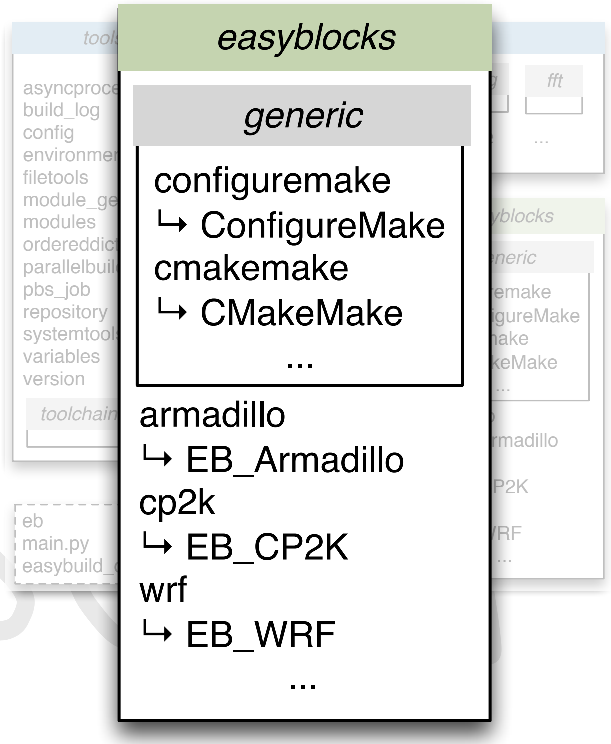
- can be used as a template for your config file

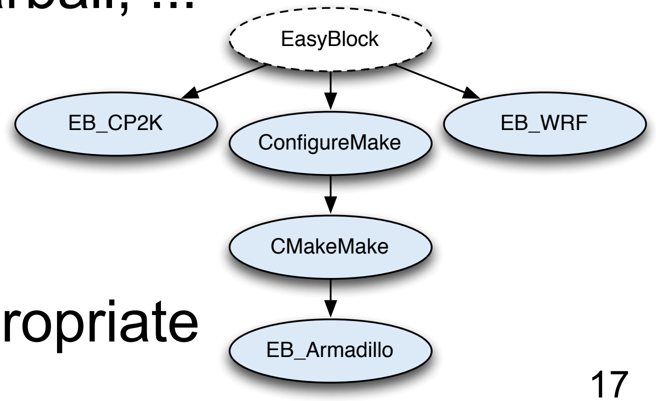


High-level design

easyblocks package

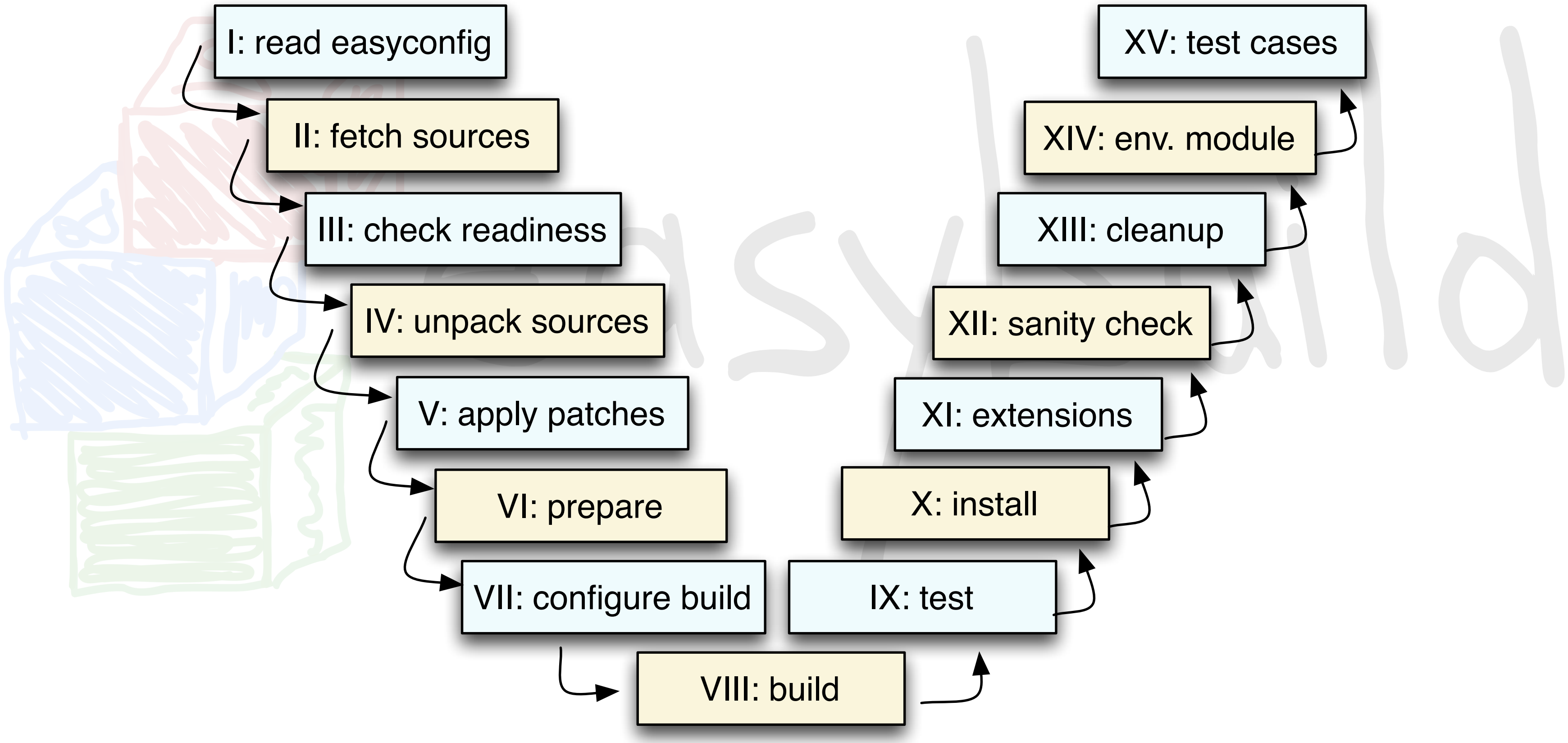
- build procedure implementations
- very modular design
  - add yours in the Python search path
  - EasyBuild will pick it up
- *easyblocks.generic*: generic easyblocks
  - custom support for groups of applications
  - e.g., ConfigureMake, CMakeMake, Binary, Tarball, ...
- application-specific easyblocks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

Features: logging and archiving

EasyBuild thoroughly logs the build process

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

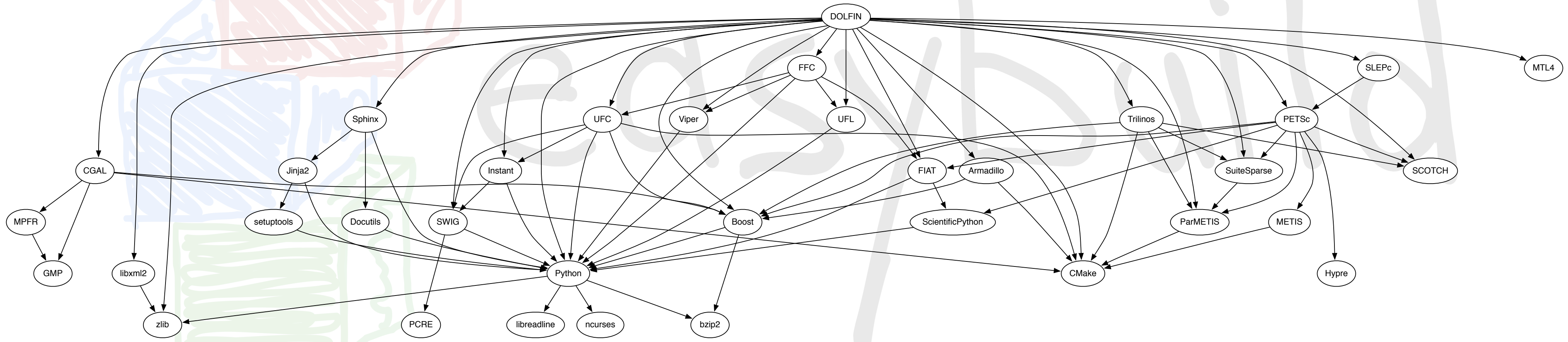
Easyconfig file used for the build is archived

- also stored in install directory
- archived into svn/git/file repository
  - enables easy sharing of easyconfigs (public repo)

Features: dependency resolution

automatic dependency resolution

the `--robot` command line option provides support for building a full 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
```

Features: interactive installers

Some software package haves 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.

Comprehensive testing

unit tests available for the EasyBuild framework

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

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

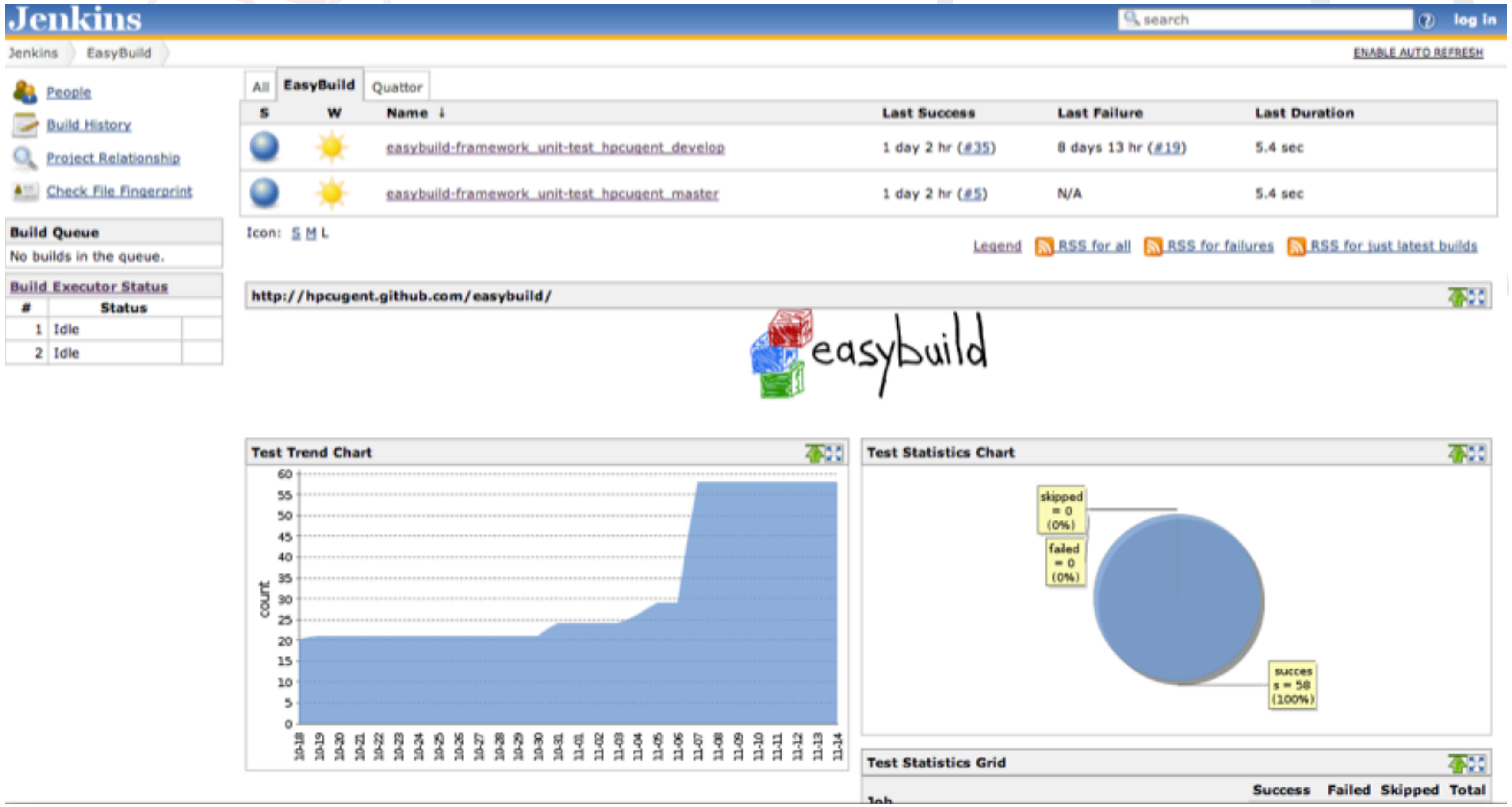
regression testing available via `--regtest`

- build ALL available easyconfig files

Comprehensive testing

unit tests are run automagically by Jenkins

- https://jenkins1.ugent.be/view/EasyBuild



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

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

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

Test Statistics Grid

Job	Success	Failed	Skipped	Total

Use case: WRF

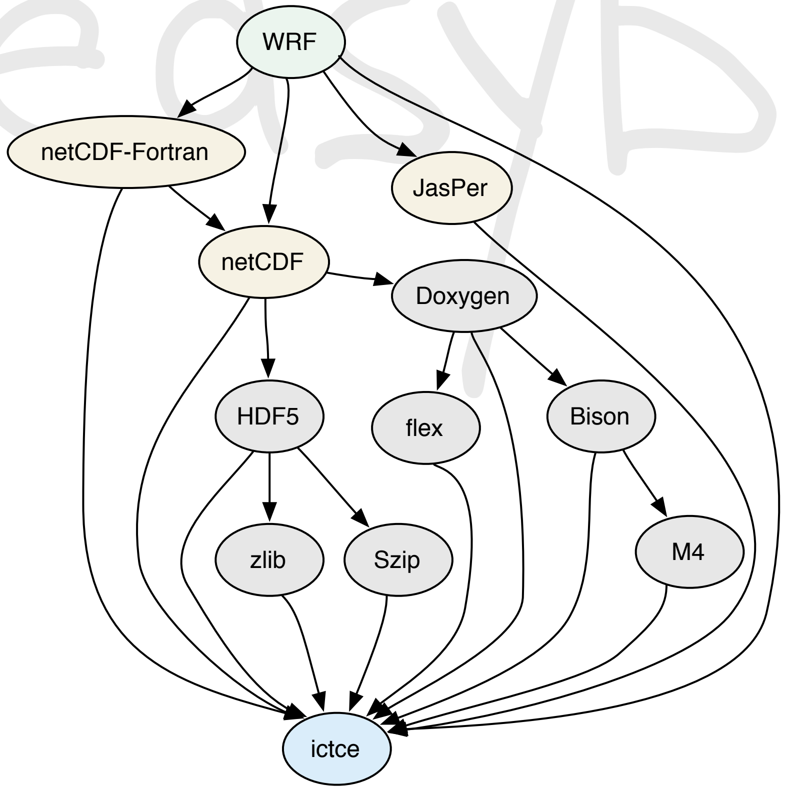
Weather Research and Forecasting Model (WRF)

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

Use case: WRF

Weather Research and Forecasting Model (WRF)

- complex(ish) dependency graph
  - dependencies also feature nasty build procedures



Use case: WRF - easyblock (1/3)

imports, class constructor,
custom easyconfig parameter

```python
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)

```

Use case: WRF - easyblock (2/3)

configuration (part 1/2)

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

  jasper = get_software_root('JasPer')
  jasperlibdir = os.path.join(jasper, "lib")
  if jasper:
    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))
```

Use case: WRF - easyblock (2/3)

configuration (part 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)
```

Use case: WRF - easyblock (3/3)

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

```
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

```

Use case: installing WRF

specify build details in easyconfig file

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

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

tcver = '3.2.2.u3'
toolchain = {'name': 'ictce','version': tcver}
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
```

easybuild

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 lmod
- 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
  - do not like it? why?
  - what features are missing that you require?
- report problems (mail, GitHub issue, IRC, ...)
- help verify correctness of easyblocks and builds
- contribute back
  - additional compilers, libraries for toolchains
  - easyblocks and/or easyconfig files

Let's build a community to tackle this problem together!

building software with ease

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

GitHub: *http://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 @ irc.freenode.net*