easybuild

building software with ease

ORNL conference call
September 3rd 2014

kenneth.hoste@ugent.be
easybuild@lists.ugent.be

HPC-UGent: in a nutshell

- HPC team at central IT dept. of Ghent University (Belgium)
  - 9 team members: 1 manager, ~3 user support, ~5 sysadmin
  - 6 Tier2 clusters + one Tier1 (8.5k cores), ~1k servers in total
  - ~1.2k user accounts, all research domains
  - tasks incl. hardware, system administration, user support/training, …
- member of Flemish Supercomputer Centre (VSC)
  - virtual centre, collaboration between Flemish university associations

“Please install this software on the cluster?”

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

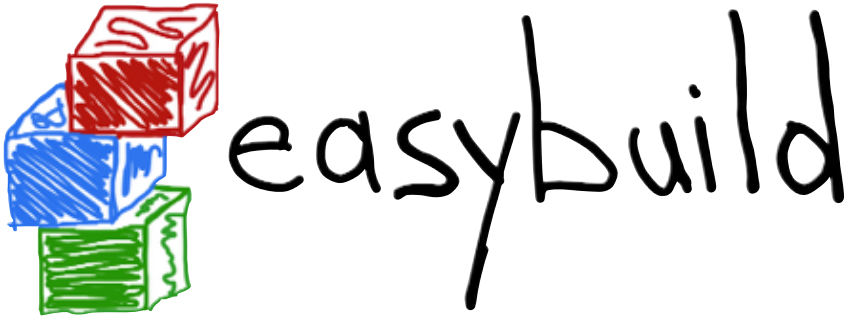
This makes building/installing (lots of) scientific software **painful**:
very time-consuming, error-prone, hard to get right, …

Common issues:

- non-standard build procedures
- tons of (obscure) dependencies
- incomplete build procedure
  e.g., no install step
- interactive scripts
- hardcoded parameters
- poor/outdated documentation
- …

Lots of duplication of work across HPC sites!

EasyBuild: building software with ease



http://hpcugent.github.io/easybuild

EasyBuild is a *software build and installation framework*.

- written in **Python 2**
- started in 2009, in-house for ~2.5 years, **GPLv2** since 2012
- EasyBuild v1.0 w/ **stable API** (Nov'12), latest is v1.14.0 (June'14)
- continuously enhanced and extended, thoroughly tested
- *release early, release often strategy* (major version every 4-6 weeks)
- development is highly **community-driven**

Requirements

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

Key features

- execute software build procedures **fully autonomously**

  incl. interactive installers, dependencies, patching, creating module files, ...

- 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`

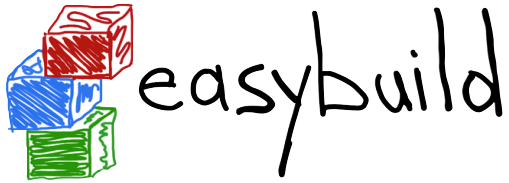
- default behaviour can be customised via **simple configuration**

- building software in **parallel**

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

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

- thriving, growing **community**



EasyBuild: basic usage

0) Install EasyBuild by bootstrapping it (easy!)

```
$ curl -O http://hpcugent.github.io/easybuild/bootstrap_eb.py
$ python bootstrap_eb.py <prefix>  # install EasyBuild here
```

1) Set module path, load EasyBuild module, basic configuration

```
$ export MODULEPATH=<prefix>/modules/all:$MODULEPATH
$ module load EasyBuild
$ export EASYBUILD_PREFIX=<prefix>  # install software here
```

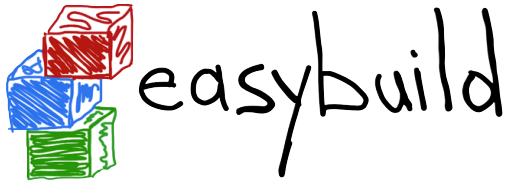
2) Example: *build and install WRF & all dependencies (!)*
using Intel compilers/libraries (!)

```
$ eb WRF-3.5-ictce-5.3.0.eb –robot
...
$ module available WRF
   WRF/3.5-ictce-5.3.0
```

List of supported software (v1.14.0)

500 different software packages *(2,596 easyconfigs)*

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



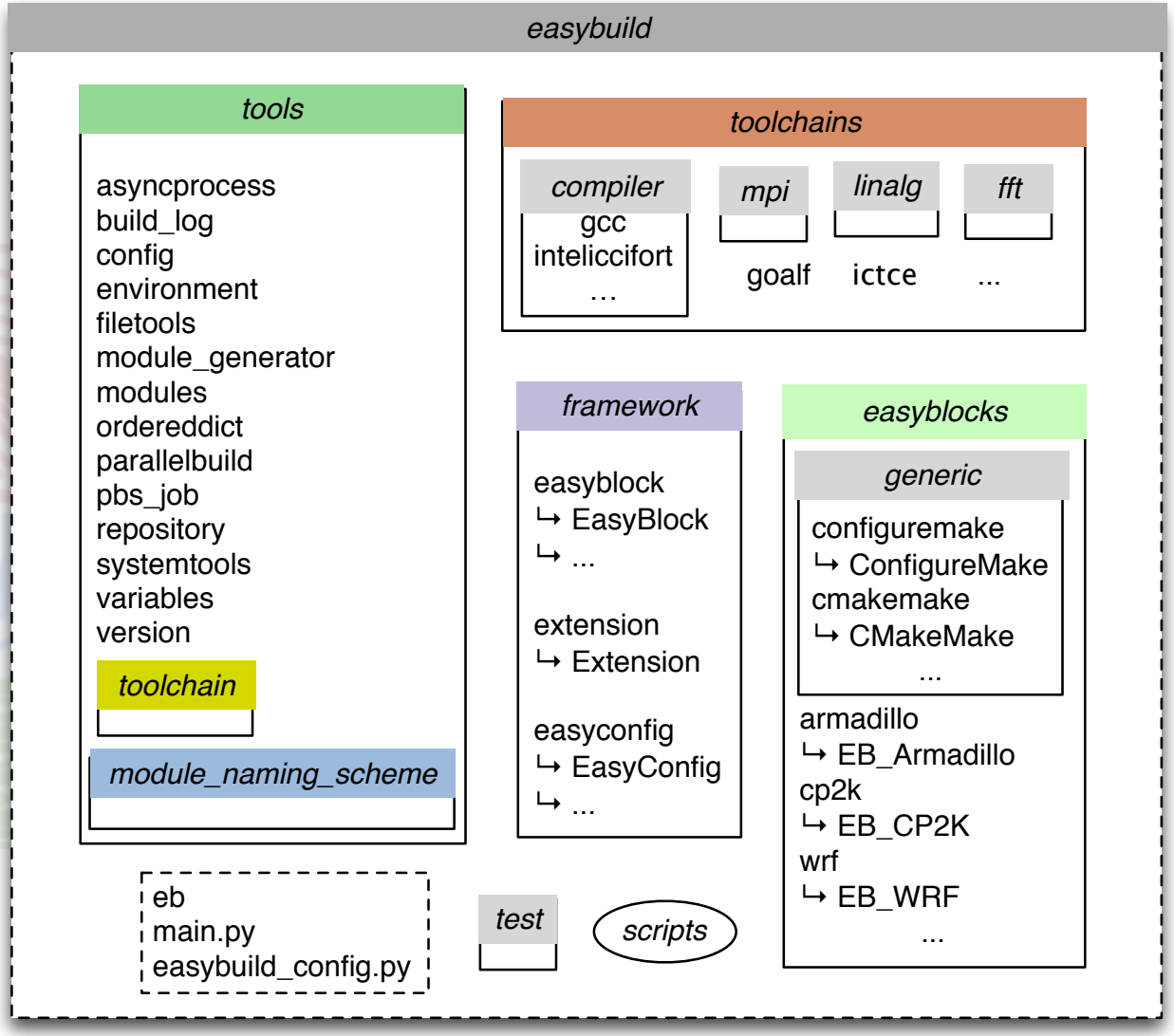
EasyBuild terminology

EasyBuild is a collection of *Python* modules that interact with each other, dynamically picking up additional Python modules as needed for *building and installing a (stack of) software package(s)* specified via simple specification files.

In EasyBuild terminology:

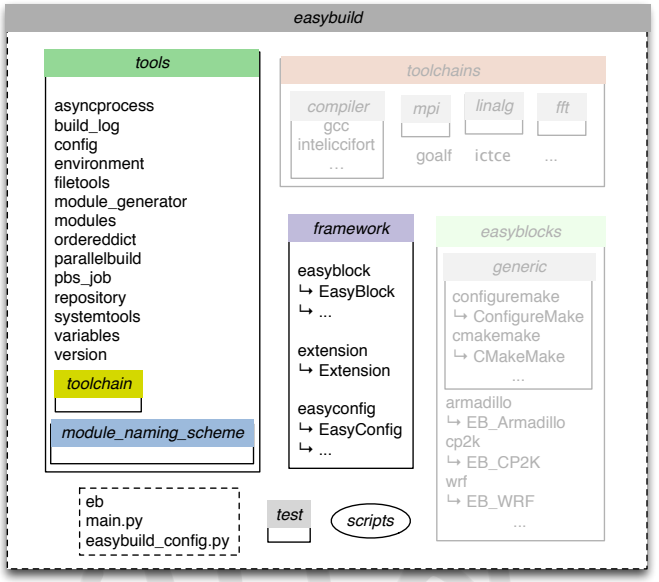
the EasyBuild ***framework*** leverages ***easyblocks*** to *automatically build and install software (stacks)* using a particular ***compiler toolchain***, as specified by one or multiple ***easyconfig files***

EasyBuild: high-level design

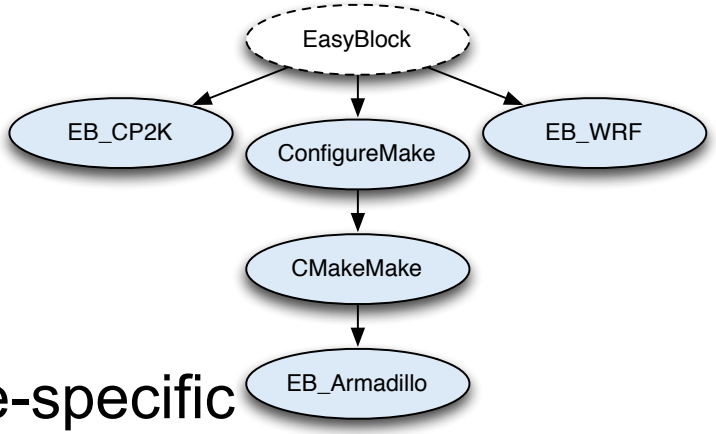


EasyBuild v1.14.0: 18,000 LOC in 125 Python modules + 157 easyblocks (20 generic)

EasyBuild framework & easyblocks



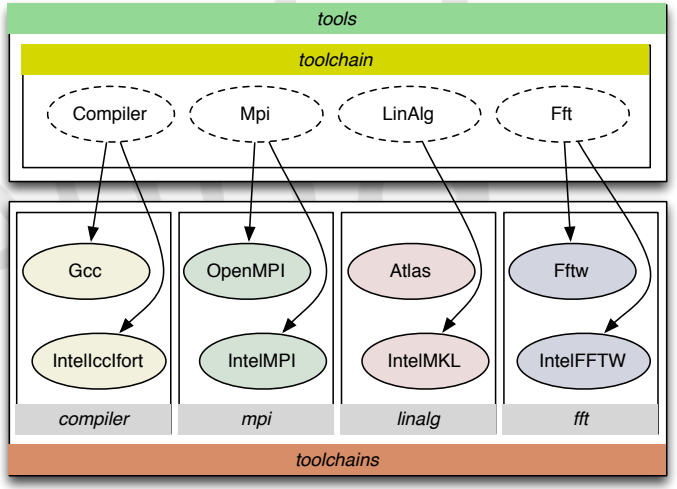
- **framework**
- *core of EasyBuild*
- Python packages & modules
- provides (lots of) supporting functionality
- very modular and dynamic design
- 'plug and play' support for new software, compiler toolchains, ...



- **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`, ...)

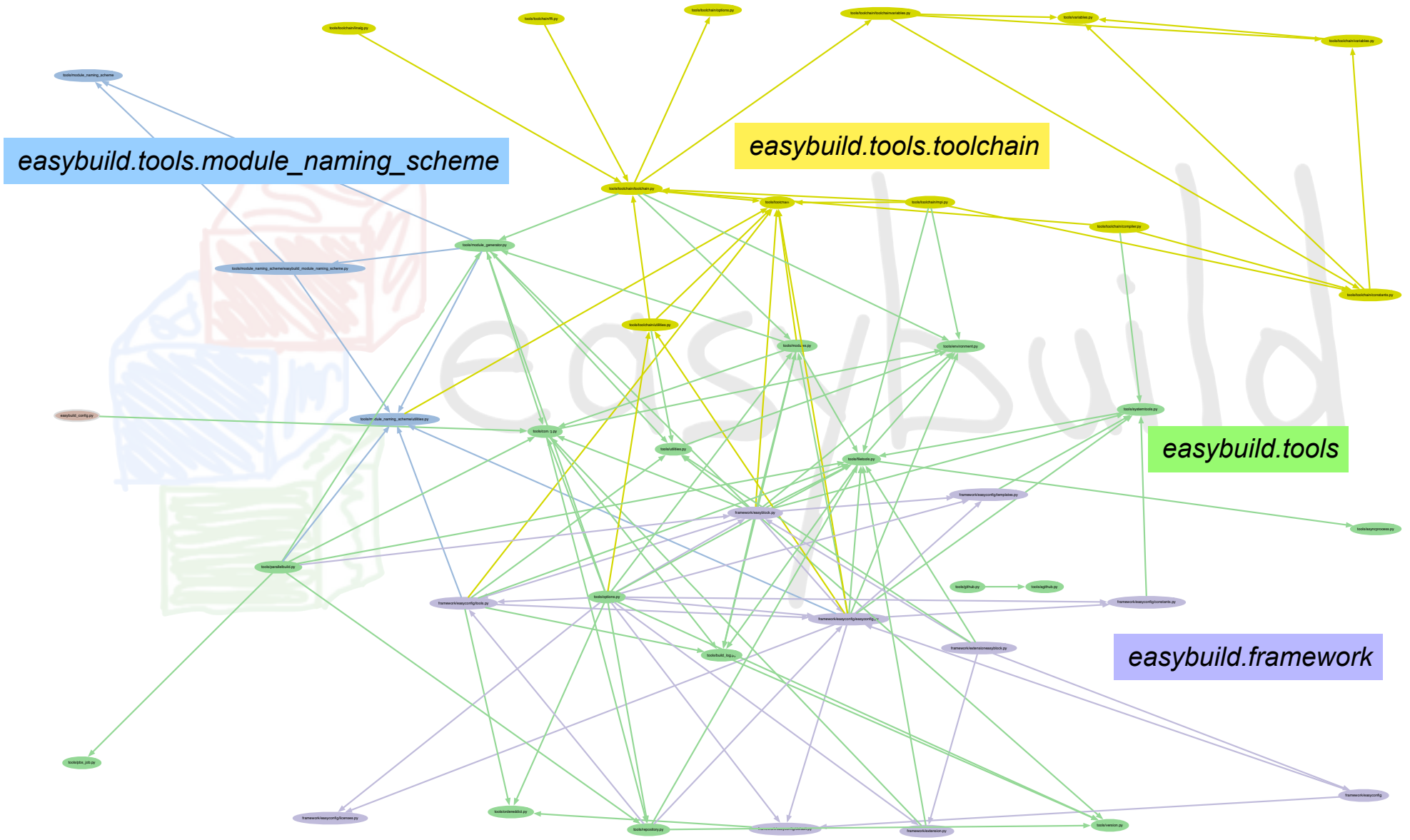
easyconfig files & compiler toolchains

- **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)



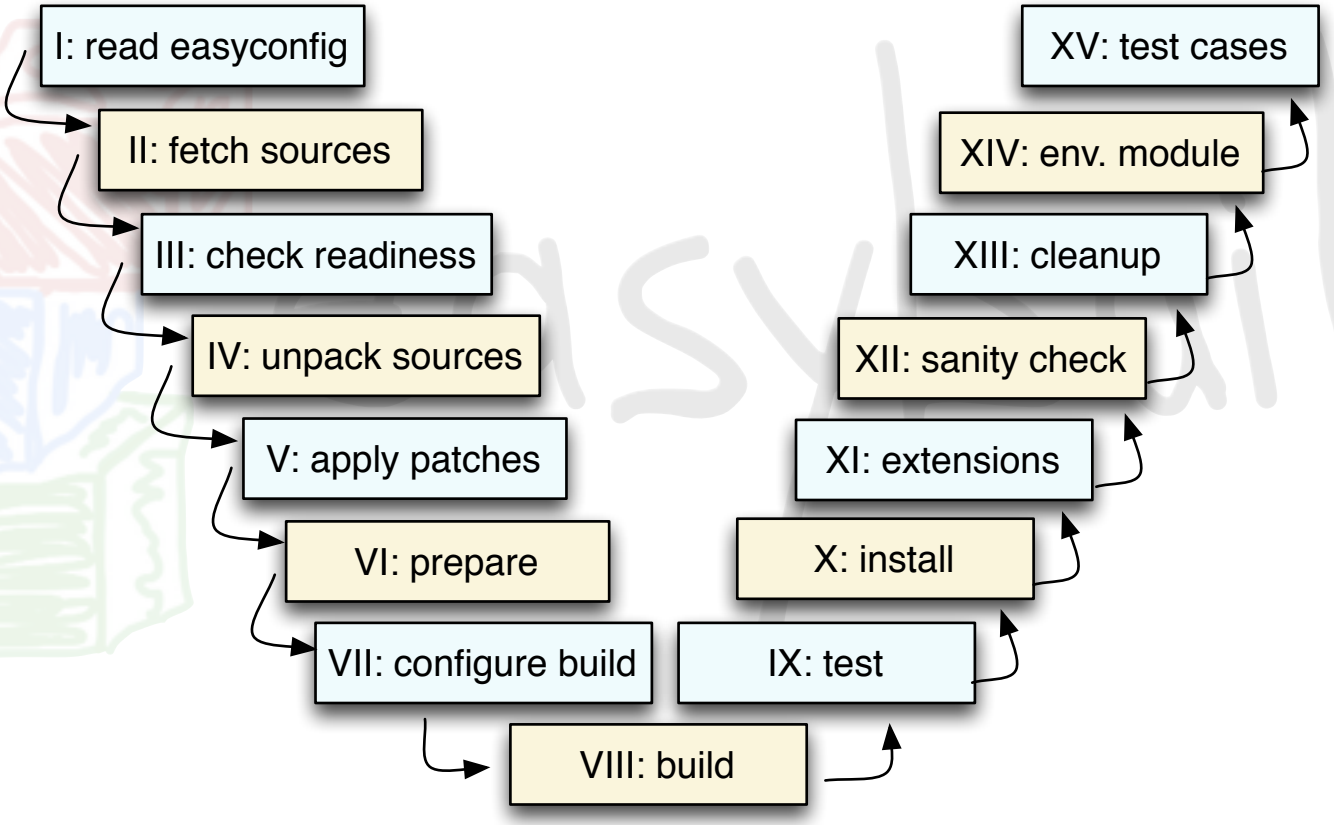
- **(compiler) toolchains**
- set of compilers and common libraries
  - usually includes MPI, BLAS, LAPACK, FFT
  - grouped together for convenience (e.g. shorter module names, ...)
- ictce: Intel tools (icc, ifort, impi, imkl)
- goolf: all open-source (GCC, OpenMPI, OpenBLAS, FFTW)
- ... (define your own!)

High-level design: framework

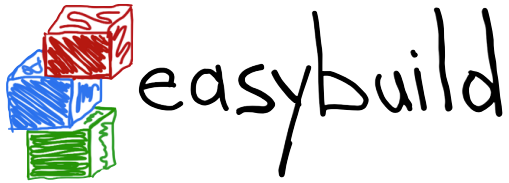


Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customized if required



Current features, future directions

currently (EasyBuild v1.14.0):

- supports GCC/Clang/Intel compilers & various MPI/BLAS/LAPACK/FFT libraries
- robust **framework** providing supporting functionality (~18k LOC, 125 Python modules)
- very **dynamic design**: plugin support for new compiler/MPI/software package
- **generates module files** (Tcl), supports using (C/)Tcl & Lmod module tools
- support for using a **custom module naming scheme** you define yourself
- (initial) support for **hierarchical module naming schemes**

work in progress:

- better integration with Lmod (support for Lua modue files & Lmod-specific aspects)
- (even) better support for site customisation
- more user-friendly error reporting
- support for more compilers, software applications & platforms (e.g., Cray, BG/Q)
- new format for easyconfig files (build specs) to handle explosion of contributions

EasyBuild community drives most new features, so *get involved!*

EasyBuild community

6th EasyBuild hackathon

June 18-20th 2014, Vienna (Austria)

Ghent University & VSC sites (Belgium)

University of Luxembourg

Gregor Mendel Institute (Austria)

The Cyprus Institute

University of Basel (Switzerland)

Jülich Supercomputer Centre (Germany)

Bayer (Germany)

University of Auckland (New Zealand)

Texas A&M University (US)

Idaho National Lab (US)

Kiev Polytechnic Institute (Ukraine)

Pacific Northwest National Lab (US)

UC Davis (US)

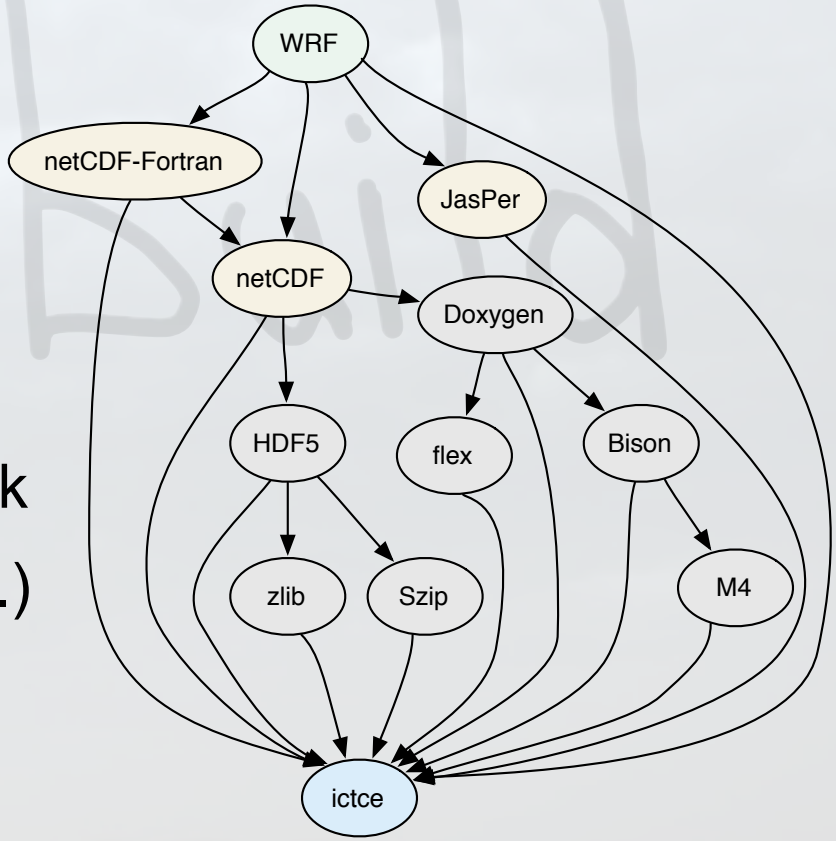
& (many?) more…

+ support from NVIDIA, TACC (Lmod), …

Use case: building WRF

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

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



Use case: building WRF with *eb*

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

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

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

Use case: easyblock for WRF

part I: imports, class constructor, custom easyconfig parameter

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

import required functionality

class definition

class constructor, specify building in installation dir

define custom easyconfig parameters

Use case: easyblock for WRF

part II: configuration (1/2)

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

configuration step function

set environment variables for dependencies

set WRF-specific env var for build options

patch configure script to run it autonomously

check whether specified build type makes sense

Use case: easyblock for WRF

part II: configuration (2/2)

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

prepare Q&A for configuring

run configure script autonomously

patch generated configuration file

Use case: WRF - easyblock (3/3)

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

build step function

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

build WRF (in parallel)

build WRF utilities as well

no actual installation step (build in installation dir)

Use case: installing WRF

specify build details in easyconfig file (.eb)

software name and version

compiler toolchain specification and options

custom parameter for WRF

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

software website and description (informative)

list of source files

list of patches for sources

list of dependencies

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

easybuild

building software with ease

Do you want to know more?

website: https://hpcugent.github.io/easybuild

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

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

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

Twitter: *@easy_build*

YouTube: *search for "EasyBuild intro", "EasyBuild WRF"*

IRC: *#easybuild on* freenode.net