# The WAF build system

Sebastian Jeltsch

Electronic Vision(s)
Kirchhoff Institute for Physics
Ruprecht-Karls-Universität Heidelberg

17. November 2010

# The WAF Tool

- project configuration, building, installation, uninstallation

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script
- supports build variants

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script
- supports build variants
- good documentation & active development

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script
- supports build variants
- good documentation & active development
- fast and small memory footprint

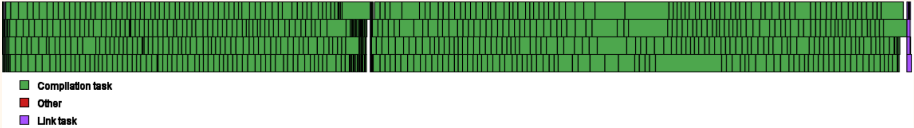
# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script
- supports build variants
- good documentation & active development
- fast and small memory footprint
  - as fast as make and 15x faster than SCons

# The WAF Tool

- project configuration, building, installation, uninstallation
- Python (WAF comes with batteries)
- WAF is only a 80kb script
- supports build variants
- good documentation & active development
- fast and small memory footprint
  - as fast as make and 15x faster than SCons
  - 10x less function calls than SCons

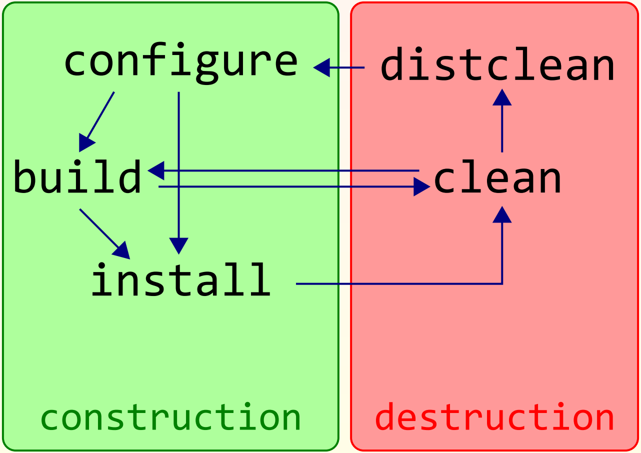
# Samba 4




Parallel build representation for samba/source3 with 4 jobs on a dual-core cpu

- build time 5min $\Rightarrow$ 35s
- build size reduction
  - check object file duplication
  - extensive shared-object and rpath use
- full dependency checks
- cleaner build rules

# Commands


configure
distclean
build
clean
install
construction
destruction

# Basic wscript Structure

```
#!/usr/bin/env python

def configure(context):
    pass

def build(context):
    pass
```

# wscript Used for this Presentation

```
#!/usr/bin/env python
# encoding: utf-8

APPNAME="WAF_Turotial"

top='.'

def configure(context):
    context.load("tex")

def build(context):
    context.new_task_gen(
        features = "tex",
        source   = "main.tex",
        )
```

# Configuration Phase

```
def configure(context):
    context.find_program('touch', mandatory=True)
    context.find_program('echo', var='ECHO', mandatory=True)
    print context.env['ECHO']

    # load custom WAF tool
    context.load('my_tool', tooldir='.')

    # define compiler flags
    context.env.CXXFLAGS_TARGET = '-g -O0 -fPIC'
```

- check for requirements
- configure build flags
- ...

# Option Parser

```
def options(context):
    context.add_option('--foo', action='store', \
            default=False, help='Silly test')

    # C++ compiler and Python
    context.load('compiler_cxx')
    context.load('python')

def configure(context):
    import Options
    print('the value of foo is %r' % Options.options.foo)
```

- easy to add options (optparse wrapper)
- values are stored in the context dictionary

# Task System

```
def build(context):
    task = context.new_task_gen(
        features     = 'cxx cxxprogram',
        source       = [ 'foo.cpp', ],
        use          = 'boost_thread-mt boost_system-mt'
        includes     = '.',
        target       = 'foo',
        install_path = 'bin')
```

commands: `build`, `clean`, `install` and `uninstall` call `build()`
$\Rightarrow$ isolate targets from actual code

| | |
|---|---|
| Execution control: | targets are evaluated lazily |
| Parallel: | task scheduling |
| FS abstraction: | e.g. distributed build |
| Language abstraction: | flexibility and extensibility |

# Task Abstraction Layer

abstraction layer between code execution (task) and declaration (task generators):

- `Task`:
    - abstract transformation unit
    - sequential constraints
    - require scheduler for parallel execution
- `Task Generator`:
    - factory task creation
    - handles global constraints (across tasks)
        - configuration set access
        - data sharing
        - OS abstraction

# Command Line Build Rules

- shell abstraction (e.g.: PowerShell ⇔ ZSH)

```
#!/usr/bin/env python
APPNAME='example4'      # shell usage & task translation

def configure(context): pass

def build(context):
    context(rule='cp ${SRC} ${TGT}', source='wscript',
                  target='f1.txt', shell=False)
    context(rule='cp ${SRC} ${TGT}', source='wscript',
                  target='f2.txt', shell=True)

    # commands containing '>','<' or '&' can not be executed
    #  => FALLBACK: shell usage (altough shell=False)
    context(rule='cat ${SRC} > ${TGT}', source='wscript',
              target='f3.txt', shell=False)
```

# More Abstract Build Rules

```
#!/usr/bin/env python
APPNAME='example2a'      # Task Generator

brule='gcc ${SRC} -o ${TGT}'

import TaskGen
TaskGen.declare_chain(
        rule      = brule,
        ext_in    = '.c',
        ext_out   = '.a',
        reentrant = False)

def configure(context): pass

def build(context):
    context(source='t0.c', target='t0', rule=brule)
    context(source='t1.c')
```

# Automated C/C++ Build Rules

```
#!/usr/bin/env python
APPNAME='example2b'      # Task Generator

def options(context):
    context.load('compiler_c')

def configure(context):
    context.load('compiler_c')

def build(context):
    context(target='t', source='t.c', features='c cprogram')
```

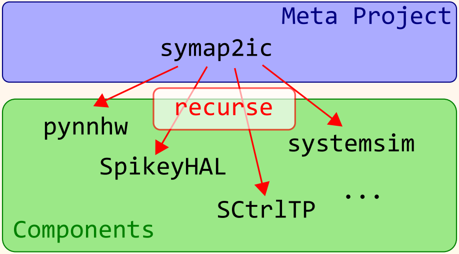
# FS interaction

```
def build(context):

    etc       = context.root.find_dir('/etc')
    fstab     = context.root.find_resource('/etc/fstab')
    local_dir = context.path.find_or_declare('a').get_src()
    local_dir = context.path.find_or_declare('b').get_bld()

    txts      = context.root.ant_glob('etc/**/*.txt')
```

- two different entry points ('.', '/')
- three different access functions
- two different target locations

Ant Globs (http://ant.apache.org/manual/dirtasks.html)

# Import of Sub-Projects




Problems:

1. projects are spread over several repositories
2. dependencies between sub-projects
   $\Rightarrow$ resolution requires fixed fs location or env variables

# Include Sub-Projects into Build Flow

```
#!/usr/bin/env python
APPNAME='example0'
VERSION='0.1337'
top='.'

import os
COMPONENTS=[os.path.join('modules', i) \
        for i in ['module0', 'module1',]]

def options(context):
    context.recurse(COMPONENTS)

def configure(context):
    print "= %s =" % APPNAME
    context.recurse(COMPONENTS)

def build(context):
    context.recurse(COMPONENTS)
```