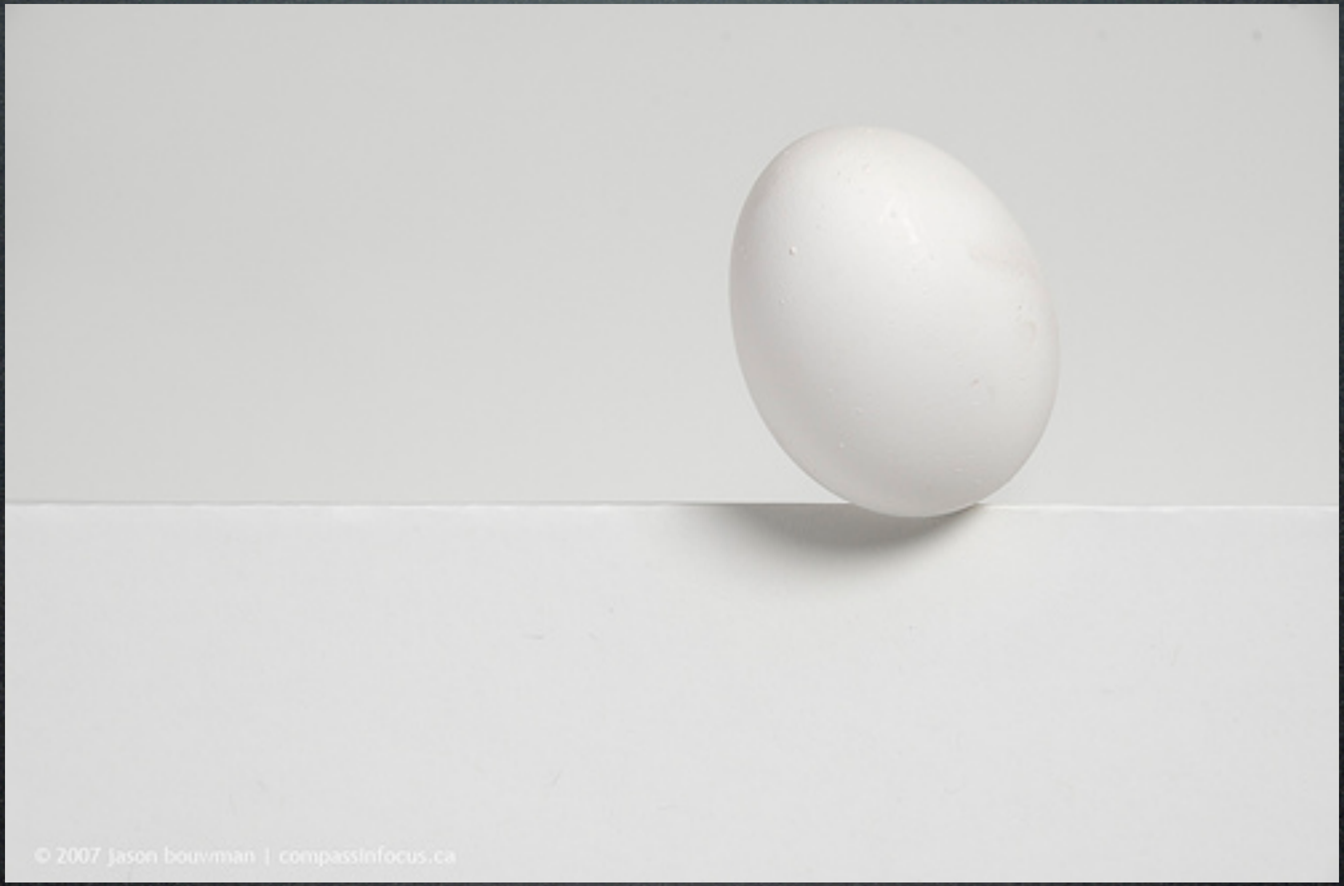
Life after Make

Build systems **suck!**

- They’re fragile



- They’re complicated

- Only one person understands it

- ...but he's not here

- On top of that...

Make has a few issues...

Make has a few issues...

- Tabs and spaces?!

Make has a few issues...

- Tabs and spaces?!
  - Even Python doesn’t recommend that

Make has a few issues...

- Tabs and spaces?!
  - Even Python doesn’t recommend that
- All work is done by external tools!

Make has a few issues...

- Tabs and spaces?!
  - Even Python doesn’t recommend that
- All work is done by external tools!
  - Hope you have good documentation

Make has a few issues...

- Tabs and spaces?!
  - Even Python doesn't recommend that
- All work is done by external tools!
  - Hope you have good documentation
- Dependency tracking is full-manual

Make has a few issues...

- Tabs and spaces?!
    - Even Python doesn’t recommend that
- All work is done by external tools!
    - Hope you have good documentation
- Dependency tracking is full-manual
- Not cross-platform

Cygwin is **not** a solution

Enter SCons

SCons

SCons

- Written entirely in Python

SCons

- Written entirely in Python
- “Makefiles” are Python scripts

SCons

- Written entirely in Python
- “Makefiles” are Python scripts
- Does dependency tracking for you!

- You still describe the steps to build your project

```
   # SConstruct (SCons Makefile equivalent)

env = Environment()

env.Program(target = 'helloworld', source = ['main.c'])
```

- Supports build variants

```
# SConstruct (SCons Makefile equivalent)
releaseEnv = Environment(BUILDNAME = 'release')

debugEnv = Environment(BUILDNAME = 'debug')
debugEnv.Append(CFLAGS = ['-g'])

for env in [debugEnv, releaseEnv]:
    Export('env')
    env.SConscript('src/SConscript',
                   variant_dir = 'build/$BUILDNAME')

# src/SConscript
Import('env')

env.Program(target = 'helloworld', source = ['main.c'])
```

- Each environment can be configured differently
    - Debug
    - Release
    - Test
    - Host vs. Target

- You can also add your own builders...

```
# SConstruct (SCons Makefile equivalent)

env = Environment()

# My custom builder
def build_foo(target, source, env):
    # ... Some tedious process to build whatever
    return 0 # Successfully built target

env.Append(BUILDERS = {'Foo': build_foo})

env.Foo(target = 'bar', source = ['main.foo'])
```

- ..that’s really nice for reducing your reliance on external tools

Helper Methods

Helper Methods

- Extraordinarily powerful!

Helper Methods

- Extraordinarily powerful!
- Makes it easy to introduce additional build steps

Helper Methods

- Extraordinarily powerful!
- Makes it easy to introduce additional build steps
  - Stripping release builds

Helper Methods

- Extraordinarily powerful!
- Makes it easy to introduce additional build steps
  - Stripping release builds
  - Munging data

Helper Methods

- Extraordinarily powerful!
- Makes it easy to introduce additional build steps
  - Stripping release builds
  - Munging data
  - Reduces redundancy

```python
# SConstruct (SCons Makefile equivalent)

env = Environment()

# You can group a series of steps into a helper
def export(env, source):
    env.Install("#export/bin", source)
    env.Install("#export/local/bin", source)

env.AddMethod(export, "ExportBin")

prog = env.Program(target = 'helloworld', source = ['main.c'])

env.ExportBin(prog)
```

- These features have allowed us to:

- These features have allowed us to:
  - Incorporate coverage testing

- These features have allowed us to:
    - Incorporate coverage testing
    - Support building apps in both the host and target environments (cross-compiling)

- These features have allowed us to:
  - Incorporate coverage testing
  - Support building apps in both the host and target environments (cross-compiling)
  - Simplify our build scripts

- These features have allowed us to:
    - Incorporate coverage testing
    - Support building apps in both the host and target environments (cross-compiling)
    - Simplify our build scripts
    - Make it super easy to add to our build process

SCons is not without fault...

SCons is not without fault...

- The code base is over-engineered

SCons is not without fault...

- The code base is over-engineered
- Despite the plethora of documentation, it could use more

SCons is not without fault...

- The code base is over-engineered
- Despite the plethora of documentation, it could use more
- Targets are built according to dependency, not the order listed

Where to find it...

- http://scons.org/
- Latest version is 1.2.0
- User’s Guide and Man page on the site