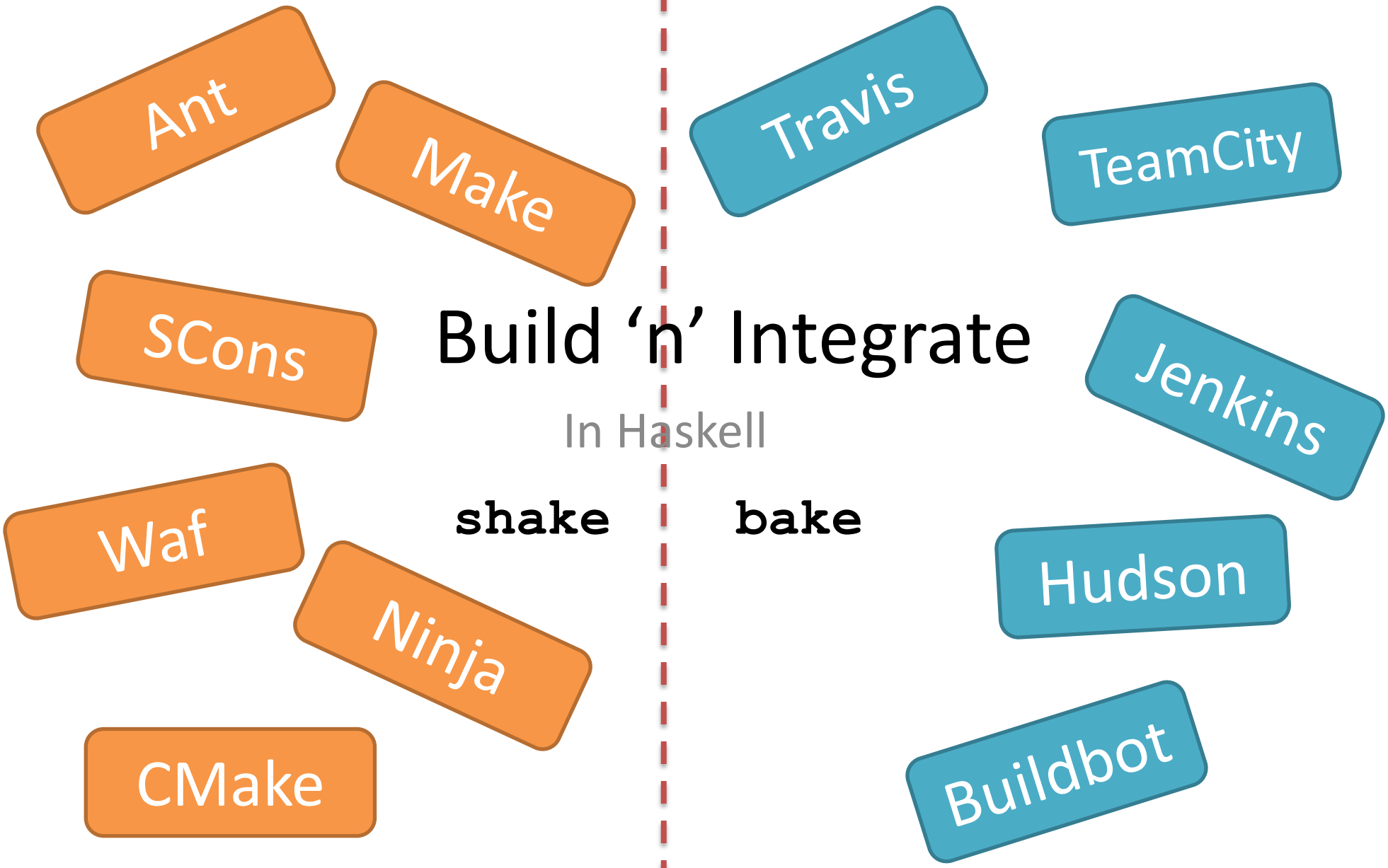
Shake ‘n’ Bake

Neil Mitchell

https://github.com/ndmitchell/{shake,bake}


Ant
Make
Travis
TeamCity
SCons
Build ‘n’ Integrate
In Haskell
Jenkins
shake
bake
Waf
Ninja
Hudson
CMake
Buildbot

Shake build system

Expressive, Robust, Fast

Haskell EDSL
Monadic
Polymorphic
Unchanging

1000's of tests
100's of users
Heavily used

Faster than
Ninja to
build Ninja

Simple example

```
out : in
    cp in out
```

```
(%>) :: FilePattern -> (FilePath -> Action ()) -> Rule ()
```

```
"out" %> \out -> do
    need ["in"]
    cmd "cp in out"
```

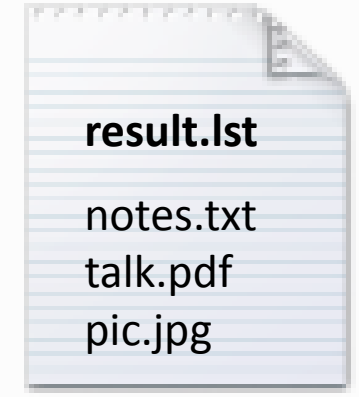
:: Action ()
Monad Action

:: Rule ()
Monad Rule

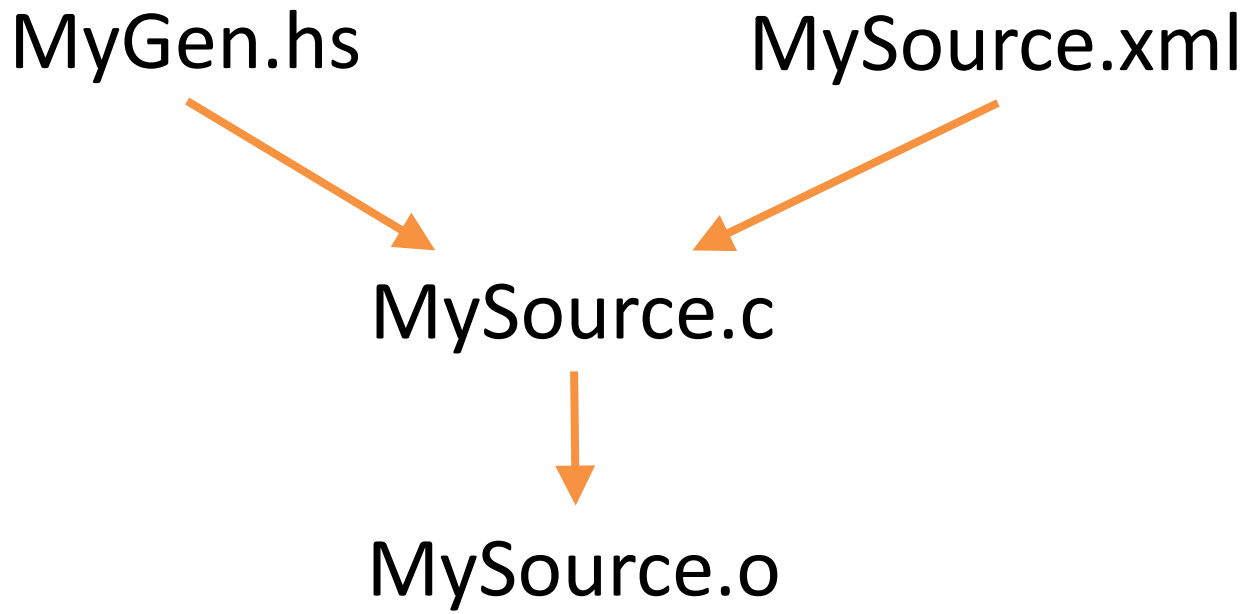
Longer example

```
import Development.Shake
import Development.Shake.FilePath

main = shakeArgs shakeOptions $ do
    want ["result.tar"]
    "*.tar" %> \out -> do
        need [out -<.> "lst"]
        contents <- readFileLines $ out -<.> "lst"
        need contents
        cmd "tar -cf" [out] contents
```



Generated files



What does MySource.o depend on?

Generated approaches

- Hardcode it?
  - Very fragile.
- Hack an approximation of MyGen?
  - Slow, somewhat fragile, a lot of effort.
- Run MyGen.hs and look at MySource.c
  - Easy, fast, precise.
  - Requires *monadic* dependencies

Monadic dependencies

Determine future dependencies
based on the results
of previous dependencies

Monadic dependencies in code

```
"MyHeader.h" %> \out -> do
    need ["MyGen.hs","MyHeader.xml"]
    cmd "runhaskell MyGen.hs"
```

```
"MySource.o" %> \out -> do
    need =<< readFile' "MySource.c.deps"
    cmd "gcc -c MySource.c"
```

See user manual for .deps rule

Unchanging

- Assume you change whitespace in MyHeader.xml and MySource.c doesn't change
  - What rebuilds?
  - What do you want to rebuild?
  - (*Very* common for generated code)

Unchanging consequences

- Assume you change whitespace in MyHeader.xml
  - Using file hashes: MyGen.hs runs and nothing
  - Using modtimes: Stops if MyGen.hs checks for Eq first
- Always build children before their parents
- What if a child fails, but the parent changed to no longer require that child?
  - Must rebuild the parent and fail on demand

Polymorphic dependencies

- Can dependency track more than just files

```
"_build/run" <.> exe %> \out -> do
    link <- fromMaybe "" <$> getEnv "C_LINK_FLAGS"
    cs <- getDirectoryFiles "" ["//*.c"]
    let os = ["_build" </> c -<.> "o" | c <- cs]
    need os
    cmd "gcc -o" [out] link os
```

Polymorphic dependencies

- 8 built in Rule instances

```
type ShakeValue a = (Show a, Typeable a, Eq a,
                     Hashable a, Binary a, NFData a)

class (ShakeValue k, ShakeValue v) => Rule k v where
    storedValue :: k -> IO (Maybe v)
```

Using Shake for our build system has been a very good decision so far, we've been able to minimise the time spent with platform-dependent build systems and IDEs and get to write Haskell code instead ;)

Stefan Kersten, CTO Samplecount
Cross-platform music stuff in C/Haskell
Using Shake for > 2 years

Ready for primetime!

- **Standard Chartered** have been using Shake since 2009, 1000's of compiles per day.
- **factis research GmbH** use Shake to compile their Checkpad MED application.
- **Samplecount** have been using Shake since 2012, producing several open-source projects for working with Shake.
- **CovenantEyes** use Shake to build their Windows client.
- **Keystone Tower Systems** has a robotic welder with a Shake build system.
- **FP Complete** use Shake to build Docker images.

Don't write a build system unless you have to!

Stealing from Haskell

- Syntax, reasonable DSLs
- Some use of the type system (not heavy)
- Abstraction, functions/modules/packages
- Profiling the Haskell functions

Extra features

- HTML profile reports
- Very multithreaded
- Progress reporting
- Reports of live files
- Lint reports
- …

Why is Shake fast?

- What does fast even mean?
  - Everything changed? Rebuild from scratch.
  - Nothing changed? Rebuild nothing.
- In practice, a blend, but optimise both extremes and you win

Fast when everything changes

- If everything changes, rule dominate (you hope)
- One rule: Start things *as soon as you can*
  - Dependencies should be fine grained
  - Start spawning before checking everything
  - Make use of multiple cores
  - Randomise the order of dependencies (~15% faster)
- Expressive dependencies, Continuation monad, cheap threads, immutable values (easy in Haskell)

Fast when nothing changes

- Don’t run users rules if you can avoid it
- Shake records a *journal*, [(k, v, …)]

```
unchanged journal = flip allM journal $ \(k,v) ->
    (== Just v) <$> storedValue k
```

- Avoid lots of locking/parallelism
  - Take a lock, check storedValue a lot
- Binary serialisation is a bottleneck

Shake Questions?

Expressive, Robust, Fast

Haskell EDSL
Monadic
Polymorphic
Unchanging

1000's of tests
100's of users
Heavily used

Faster than
Ninja to
build Ninja

Bake Continuous Integration

- A lot less applicable and mature than Shake
  - Not suitable for everyone
  - And those who it is suitable for might find it sucks
  - But already used in production at 3 or 4 places

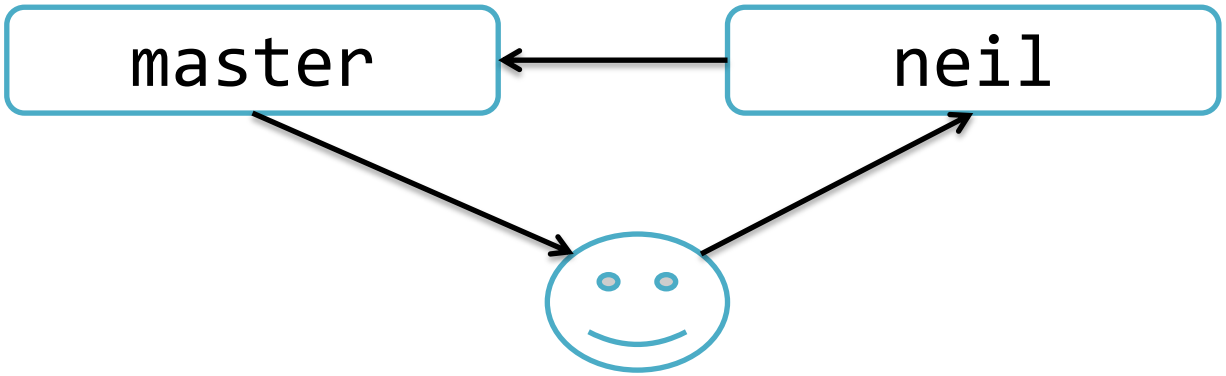
DISCLAIMER

Bake for Managers

- Continuous integration – Travis, Jenkins…
- Designed for teams which are:
  - Large: ~5-50 people
  - Semi-trusted: Not always advance code review
  - Productive: Writing lots of code
- Never break the build

https://github.com/ndmitchell/bake

Bake for Developers



- Master branch *always* works perfectly
- When code is ready, tell Bake
- Bake compiles it, runs the tests, merges it
- Bad code is rejected

Not enough time in the day

- 50 patches are promoted per day
- Compile & test = 10 hours (multithreaded)
- 20+ servers testing is infeasible
  - 2 might be reasonable, Windows & Linux

- Bake's solution
  - Assume if p1+p2 pass the tests, that's fine
  - If a test fails, then identify whether p1 or p2 fails

Bake Continuous Integration

Patches

Patch	Status
779ff62 by tony Main.hs	Testing (passed 5 of 8) Retrying Linux Run 10, Windows Run 10
90e55ac by bob Main.hs	**Rejected** Linux Run 10, Windows Run 10
a684325 by bob Main.hs	Testing (passed 6 of 8) Retrying Linux Run 10, Windows Run 10
c9a3ac6 by tony Main.hs	**Merged**
3064bb2 by bob Main.hs	**Merged**

Clients

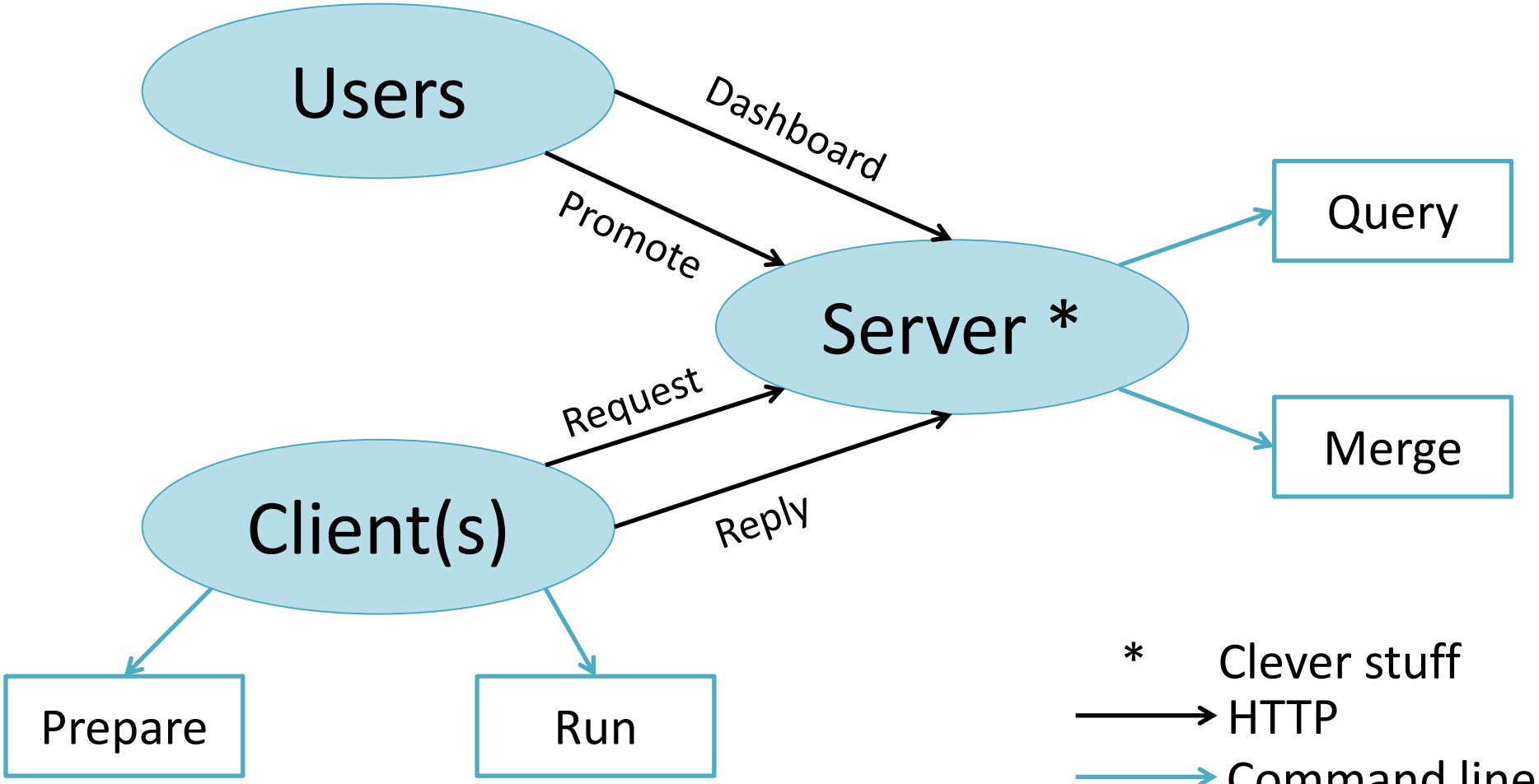
Name	Running
Linux	Linux Compile
Windows	*None*

Configure in Haskell

```
data Action = Compile | Test

main = bake $
    ovenGit repo "master" Nothing $
    ovenTest (return [Compile,Test]) exec
    defaultOven

exec Compile = run $ cmd "shake"
exec Test = after [Compile] $ run $ cmd "test"
```



90% string passing

String passing the Haskell way

```
data Stringy s = Stringy
    {stringyTo :: s -> String
    ,stringyFrom :: String -> s
    ,stringyPretty :: s -> String
    }

stringyTo . stringyFrom == id
stringyFrom . stringyTo == id

check :: Stringy s -> Stringy s
```

Stealing from Haskell

- Parameterisable and configurable
  - Parameterised over version control
  - Parameterised over tests
- Use types to safely pass different strings
- A bit of pure “clever” stuff in the middle

Optimisation

- First version was *way* too slow
  - Directory copy on Windows is very slow
  - Git checkout from scratch is very slow
- Use a single directory for all building
- Tarballs of each compiled state (distribution only)
- Extract tarballs to do a bisection on test failure
- Use exhaustive search near the leaves

Should you use Bake?

- Are you in a large tech firm? Google/Facebook?
  - Probably have lots of CPU years dedicated to testing
- Are you an individual or a small organisation?
  - Probably can use Travis just fine and fix your mistakes
- Are you in the middle? With hours of tests?
  - Bake might be suitable here.

Questions?

Or beer?