



# Gluing things together with Haskell

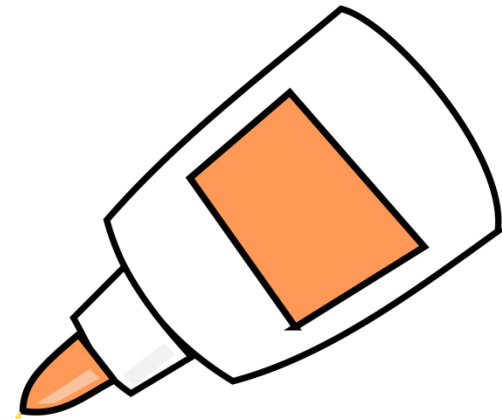
Neil Mitchell

<http://nmitchell.co.uk/>

# Code

Elegantly designed

- Build system
- Test harness
- Continuous integration
- Release bundling
- Installer generator
- Release distribution
- ...



# Release

Thoroughly tested



“A rats nest of Bash”

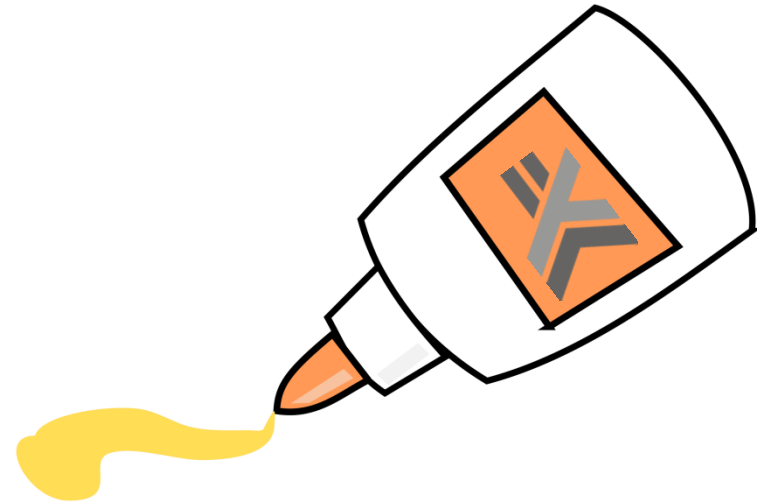
- Would your team write 10K lines of Bash?
- Lots of people write > 10K of Makefile
  - Standard Chartered, GHC developers

# What to do?

1. Accept regular failures
2. Invest lots of time and money on an ongoing basis
3. Do it right  
(elegantly designed and thoroughly tested)

**SHAKE**

Build system



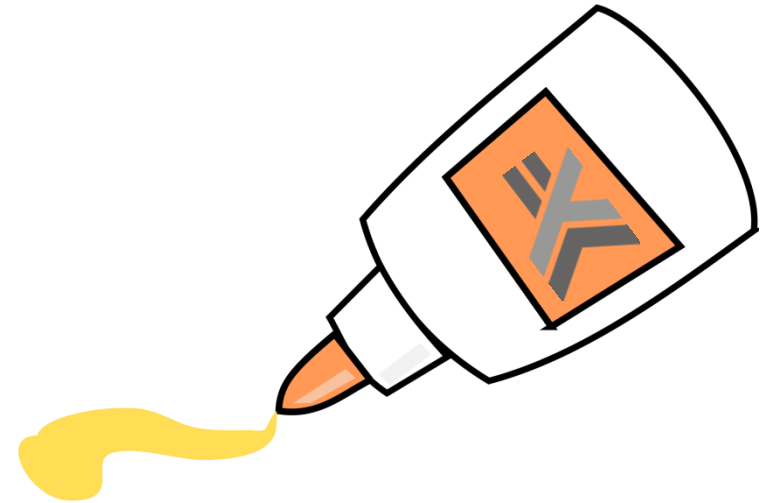
**BAKE**

Continuous integration

**NSIS**

Installer generator

All open source Haskell libraries



# **SHAKE**

Build system

# Shake for managers

- Build system - alternative to Make, Scons...
- Reliable and robust
- Powerful dependencies
- Fast to run

<http://shakebuild.com/>



# Shake for developers

- A Haskell library for writing build systems
  - Your code is in Haskell, but calling compilers etc
- Monadic dependencies (generated code)
- Polymorphic dependencies (not just files)
- Optimised and tested (faster than Ninja)

<https://github.com/ndmitchell/shake>



# An example

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



**result.lst**

notes.txt  
talk.pdf  
pic.jpg

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



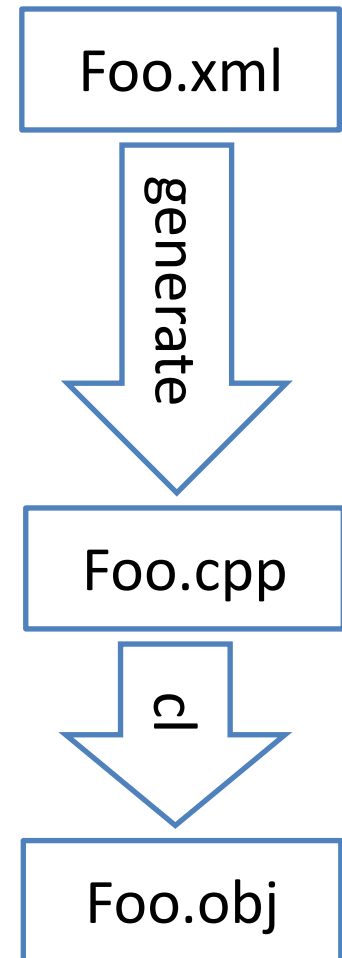
**result.tar**

notes.txt  
talk.pdf  
pic.jpg

# Monadic dependencies

What does Foo.obj depend on  
(what does Foo.cpp #include)

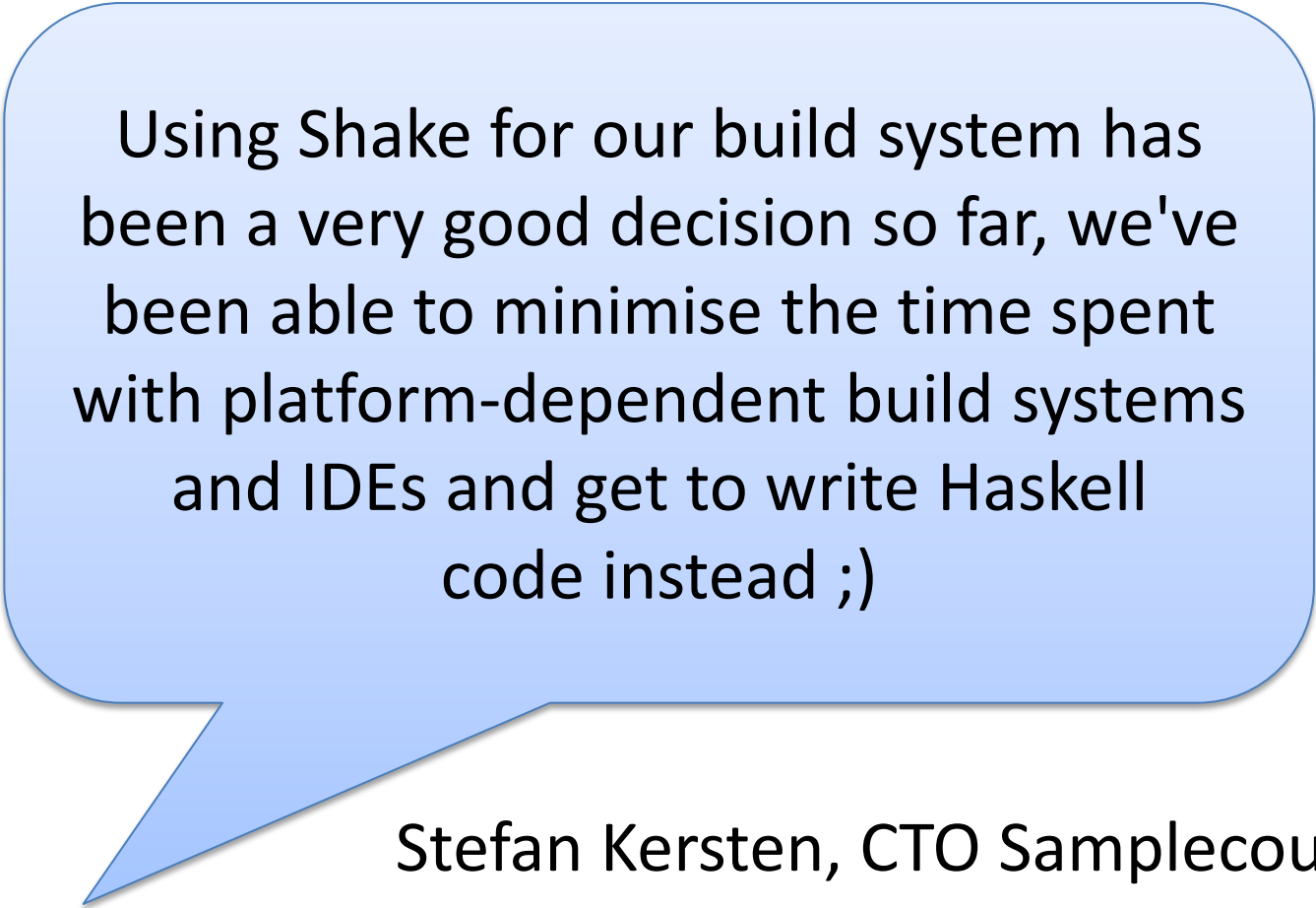
- Applicative
  - Tell me the dependencies up front
  - Phases? Guess from Foo.xml?
- Monadic
  - New dependencies later
  - Generate Foo.cpp. Look at it.



# Polymorphic dependencies

Way more than just files  
(but files are still the norm)

- Does a file exist (perfect for `$INCLUDE` paths)
- Contents of a directory (good for VS projects)
- Compiler/library versions
  - Upgrade a library, have the right things rebuild



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

# At Standard Chartered

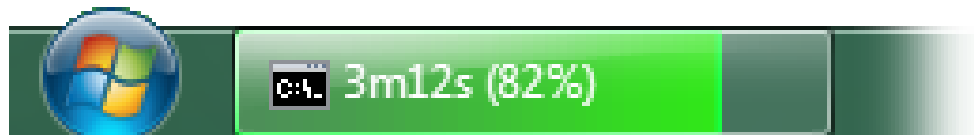
- > 10,000 lines Makefile became < 1,000 Shake
- Compiled more than 2x faster
- More malleable – no global phases
- A fantastic success
  - Our project keeps growing
  - Same structure as at the beginning
  - Monadic = more compositional

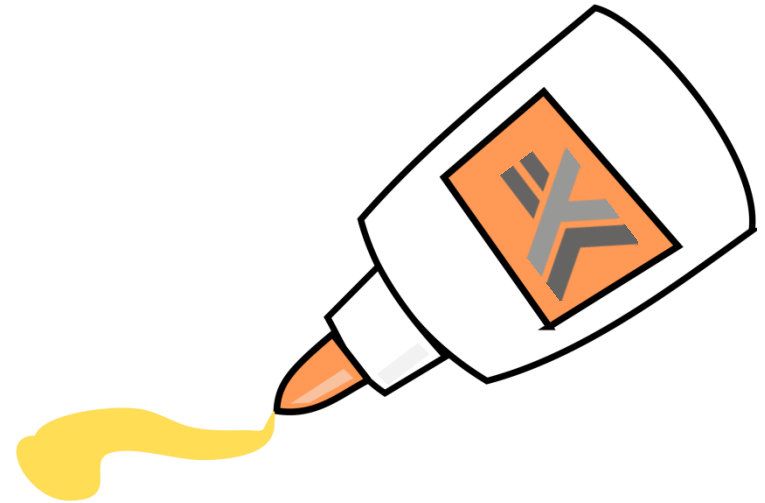
# Stealing from Haskell

- 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
- ...





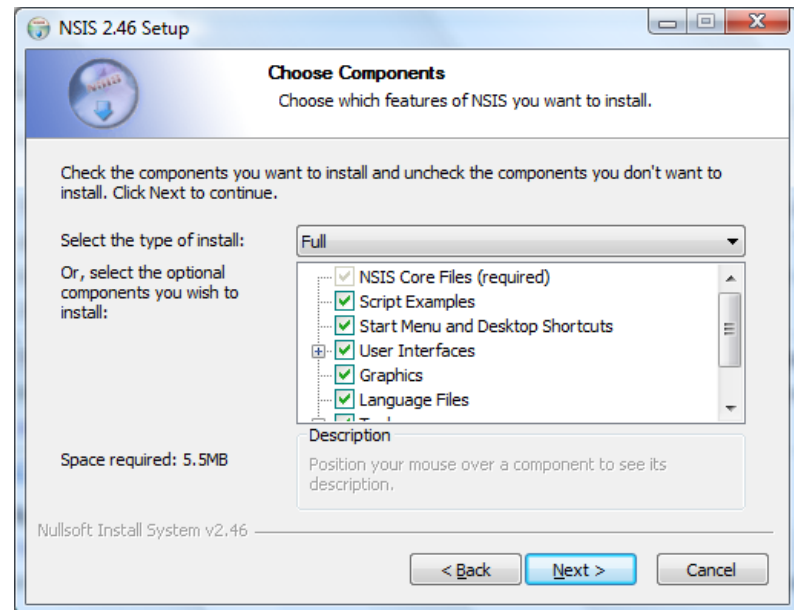
# NSIS

Installer generator



# NSIS – NullSoft Install System

- Originally the WinAmp installer (pre 2002)
- Generates small, fast Windows installer
- Lots of plugins



<http://nsis.sourceforge.net/>

# Quirky (understatement)

- Bad language
  - Scripted with a programming language
  - Twenty registers (\$0, \$R0), plus a stack, plus mem
  - Goto only, plus functions, no block if/for
  - Everything is a string (< 1Kb, or it segfaults)
- Bad structure
  - Nice user interface requires MUI2
  - A set of preprocessor defines over NSIS

# Solution: Haskell

- Define a DSL for writing NSIS libraries
- Generates NSIS code underneath
- Expression/Statement orientated
  - Very imperative

<https://github.com/ndmitchell/nsis>

# Comparison

```
StrCmp $WINDIR $INSTDIR bad 0
StrCmp $SYSDIR $INSTDIR bad 0
Goto skip
bad:
MessageBox MBOK | MB_ICON_EXCLAMATION "Bad idea"
skip:
```

Vs

```
iff_ (" $INSTDIR" %== "$WINDIR" %||
      "$INSTDIR" %== "$SYSDIR" ) $
      alert "Bad idea"
```

# Comparison

```
!Include MUI2.nsh
Name "Example1"
!insertmacro MUI_PAGE_DIRECTORY
!insertmacro MUI_PAGE_INSTFILES
!insertmacro MUI_LANGUAGE "English"
Section "" _sec1
    SetOutPath "$INSTDIR"
    File "Example1.hs"
SectionEnd
```

Vs

```
name "Example1"
page Directory
page InstFiles
section "" [] $ do
    setOutPath "$INSTDIR"
    file [] "Example1.hs"
```

# Add types and structure

```
data S = S Unique [NSIS]
data Action a = Action (State S a)
data Value ty = Value Val
type Exp ty = Action (Value ty)
-- ty is String, Int or Bool
```

Monad Action

Enum (Exp Int)

Integral (Exp Int)

Real (Exp Int)

Bits (Exp Int)

Functor Action

Eq (Exp a)

Num (Exp Int)

Show (Exp a)

Typeable a => IsString (Exp a)

Applicative Action

Fractional (Exp Int)

Ord (Exp Int)

Monoid (Exp String)

# Reduce expressions

```
iff_ :: Exp Bool -> Action () -> Action ()
iff_ test true = do
  thn <- newLabel
  end <- newLabel
  Value t <- test
  emit $ StrCmpS t (lit "") end thn
  label thn
  true
  label end
```

# Optimise

```
Goto foo  
foo:
```

```
dullGoto :: [NSIS] -> [NSIS]
```

```
dullGoto = transform f
```

```
  where
```

```
    f (Goto l1:Label l2:xs)
```

```
      | l1 == l2 = Label l2 : xs
```

```
    f x = x
```

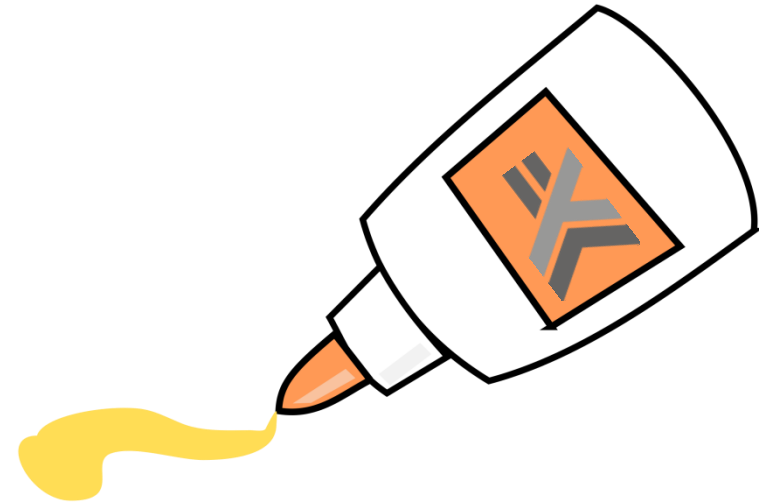


# Stealing from Haskell

- Syntax, reasonable DSLs
- Phantom types to eliminate lots of errors
- Abstraction, build up in layers
- Standard compiler techniques
- Symbolic manipulation for optimisation
  - (which is pretty much totally unnecessary)

# The Result

- Doesn't define an installer, wraps an installer
- Polish off the rough edges, fix a few bugs
- Hide all the complexity
- Keep all the good stuff



# **BAKE**

Continuous integration

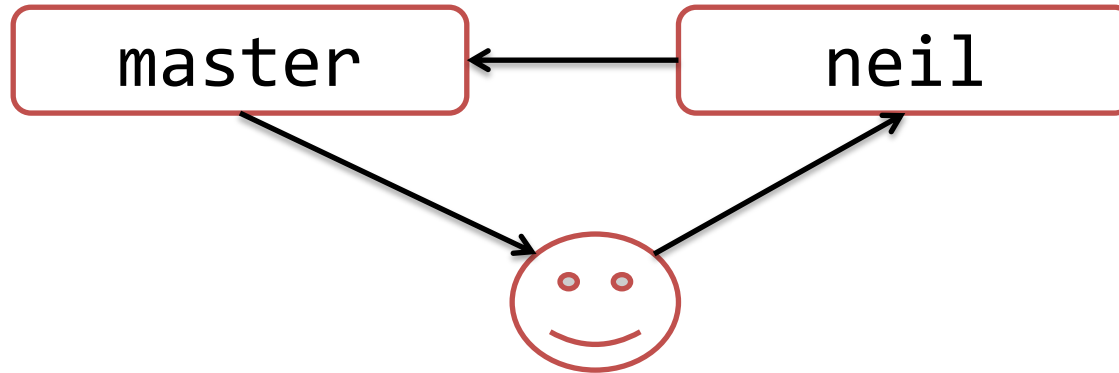
# Bake for managers

- 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
<a href="#">779ff62</a> by tony Main.hs	Testing (passed 5 of 8) Retrying Linux Run 10, Windows Run 10
<a href="#">90e55ac</a> by bob Main.hs	<b>Rejected</b> Linux Run 10, Windows Run 10
<a href="#">a684325</a> by bob Main.hs	Testing (passed 6 of 8) Retrying Linux Run 10, Windows Run 10
<a href="#">c9a3ac6</a> by tony Main.hs	<b>Merged</b>
<a href="#">3064bb2</a> by bob Main.hs	<b>Merged</b>

## Clients

Name	Running
Linux	Linux Compile
Windows	None

# Configure in Haskell

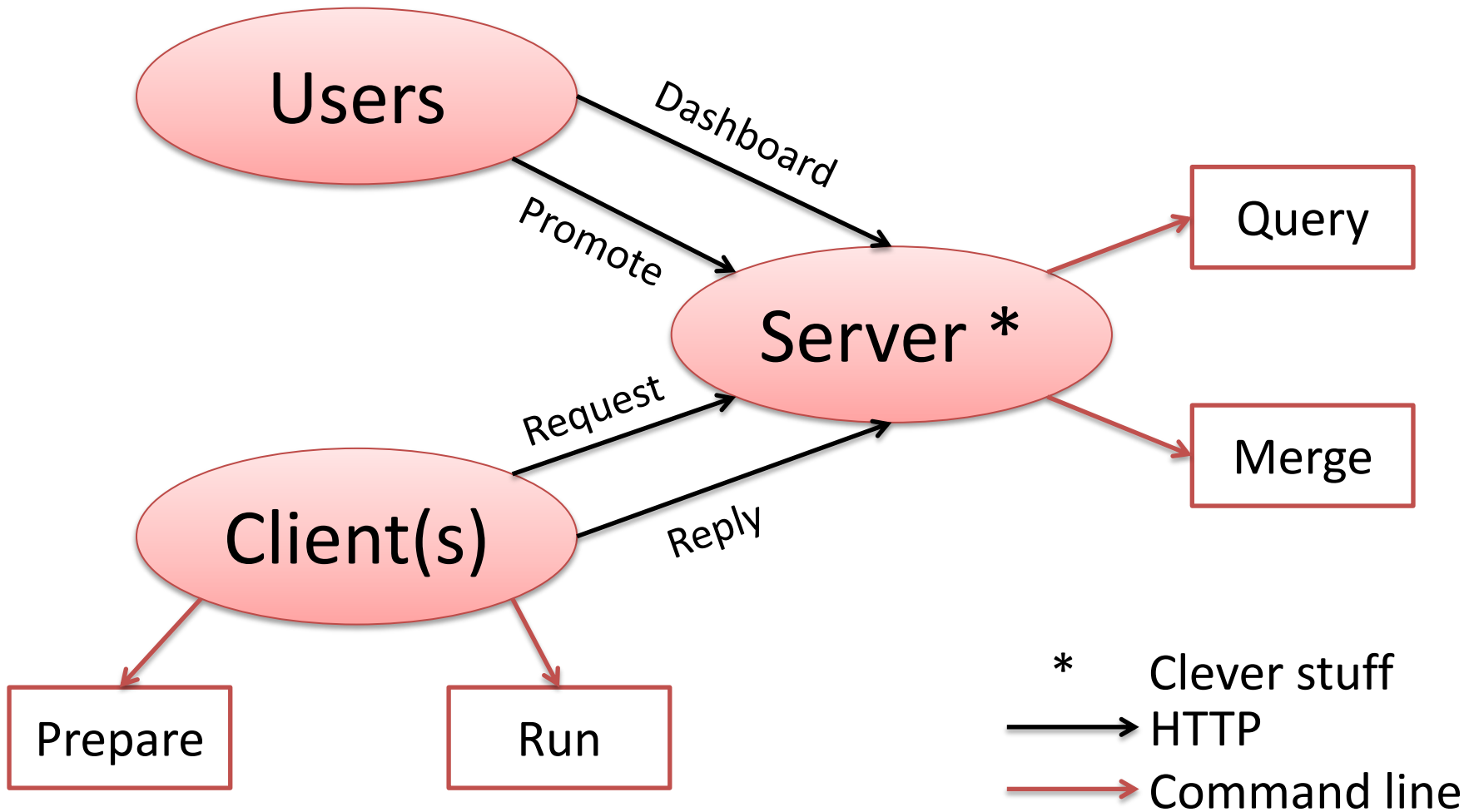
```
data Action = Compile | Test
```

```
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

# The Result

- Too early to say!
- Bake is only 6 weeks old
- Looks promising...

**SHAKE**

**NSIS**

**BAKE**

**TODO**

Lots more

