FP and new Teams

Michael Neale
@michaelneale
developer.cloudbees.com

Brisbane

Background

4 years hard labour in Brisbane

Work @cloudbees: developer.cloudbees.com

FP on and off since uni (first programming language)

Refused entry on several occasions to Treasury Casino for wearing **camouflage**.

Context

New Company
New Team
New Product

You might not be this “lucky”
But I hope you get some ideas or inspiration

Or at least know not to wear cammo

John Arundel
@bitfield

Following

Be a little sceptical of anyone speaking at a conference. After all, this was the guy whose employer could spare him to go to a conference.

Reply Retweeted Favorited More

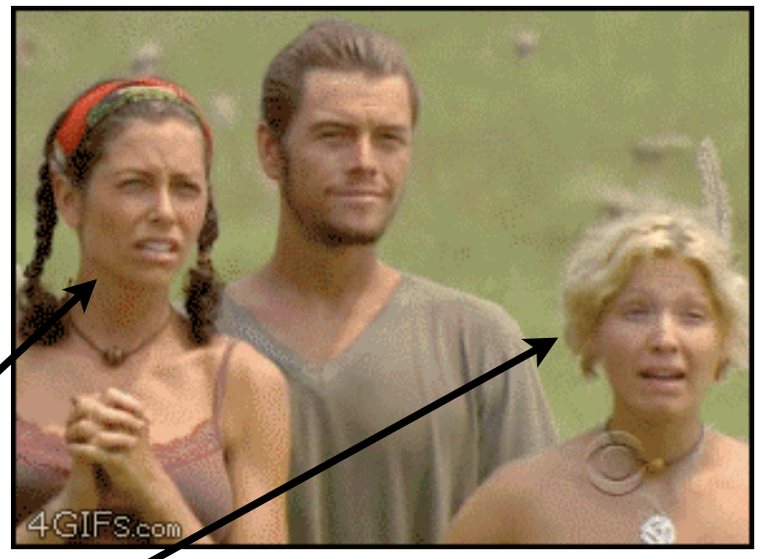
History

2010 Started: JVM stack parts - Scala not controversial (I had experience) - working mostly “lone wolf”

2011 - another team added - brought Erlang

Erlang!

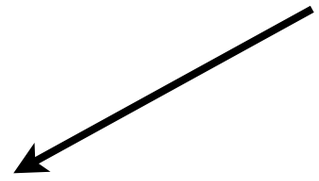
Me



other team members

Warning

Me



Anecdotal evidence ahoy*

**Sometimes all we have*

More FP

Given “success” with Scala, another “FP language” was not a great risk.

The teams so far: two lone wolves. The one man wolfpack is now two one wolfman wolfpacks... what?

Since then

More developers hired

Existing developers start to care about FP

I learned Erlang (avoiding the one person wolfpack)

2012 - some Clojure “micro services” introduced

Observation

FP means one person can do more &&
People like me like to work alone

∴ risk of staying with one-person-per app
(aside: is this a bad thing?)

Objections and

Ask why:
- resistance to the “new” (unnecessary risk)
- resistance to FP ideas (hype?)
- polyglot fear

(realistically multiple languages will be used)

Not everyone enjoys the things we do.

Weirdos

Maintainability

It goes:
- how will anyone be able to maintain this after you?

My Experience:
- 2 projects featuring FP handed over successfully
- new developers able to pick up FP easily
- seasoned developers too

My anecdotes are clearly science!

Where we are today

Several systems involving:

- Erlang (on every server and core services)
- Scala (cloud controllers)
- Clojure (micro services - talk to github api)

Where we may differ

Small teams - many systems.

∴ Little overlap in jobs.

What did we do with FP

Manage horrendous public cloud APIs

Server controlling (agent) - manage horrendously misbehaving apps

Automatic scaling

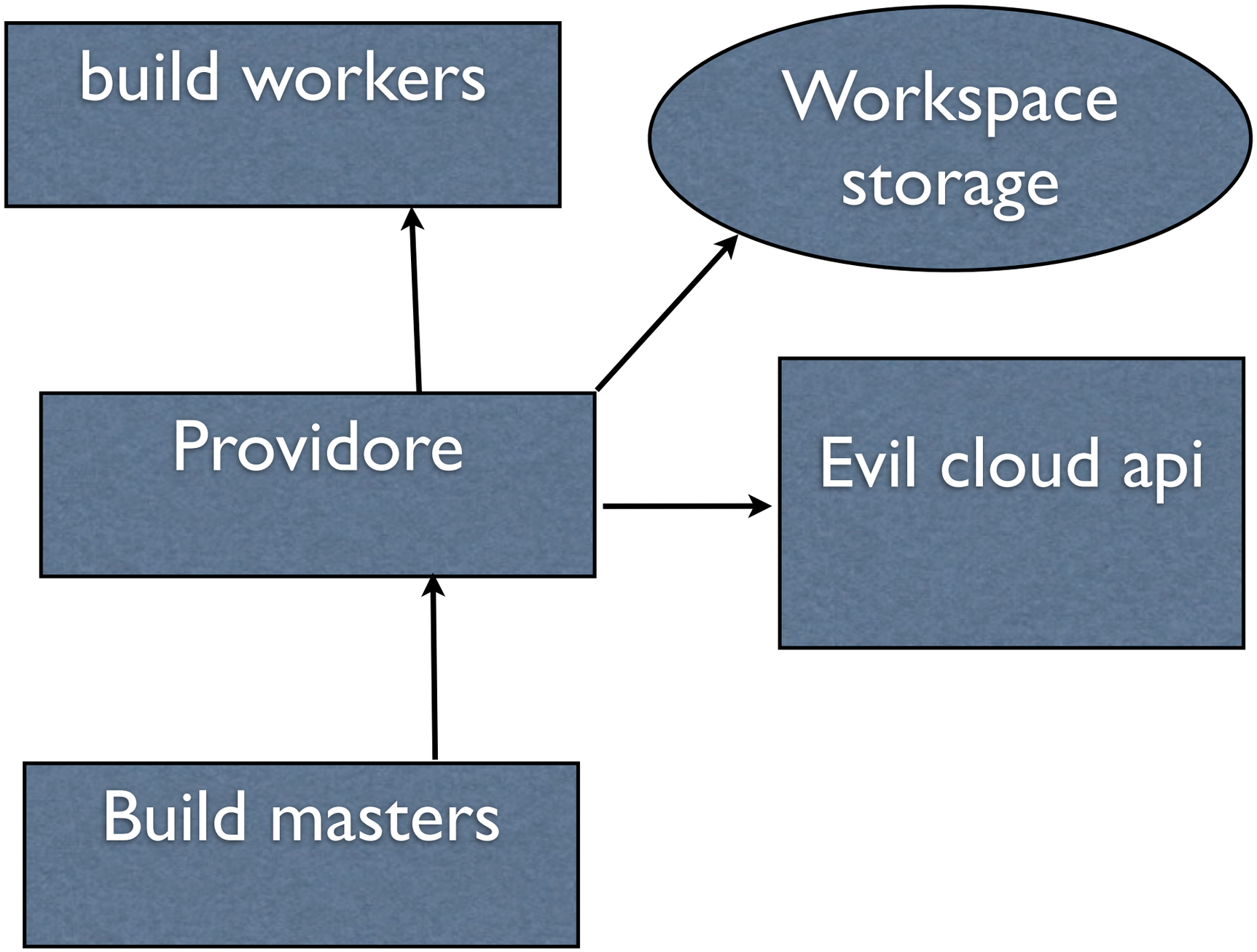
Github crawling

Surprisingly practical

No real calculations, no explicit maths.

Just boring every day error prone stuff.

For example


build workers
Workspace storage
Providore
Evil cloud api
Build masters

The problem:

New build required. Check the pool, talk to the Evil api, ask for a new server, it fails, ask again. Wait for server to be up, no, I mean really up.

Ask for a new disk, wait for a new disk, it fails, try again, attach the new disk, is the disk attached? fails, try again, damn it the snapshot isn’t available.

How can we solve this?

TDD? problems only manifest under load/in-production. APIs are buggy, change over time.

Industry best practice: hack something together as a script and some of the time it works.

Can FP help us?

Yes

Types (providore is written in scala)
 - Specifically: Maybe/Option, Either
 - Closed Data Types (servers only in so many states)

The M word: Monads

Currying

Cloud API

launch_server: Server

at best hopes to be:

launch_server: Maybe[Server]
launch_server: Either[Server, OhGodWhyWhyWhy]

(not an actual pure function of course)

Cloud Monad

Cloud APIs are like IO

Slow, horrible, misbehaving IO

...and then the APIs other people write

All want to be monadic

```
val validation: String \/ Subscription = (for {
  account <- extractAccount
  _ <- validateSubscription(account)
  callback <- extractCallBack
  plan <- validatePlan
  billing_day <- extractBillingDay
  subscription <- createSub(account, plan, callback, ...
} yield subscription).run(req.body)
```

(scala) ReaderT to help you compose

http://debasishg.blogspot.com.au/2011/07/monad-transformers-in-scala.html

Need to "organise code in monadic way"
See Jed's talk!

Jed

See his talk. He explains it and has a great beard.

Infrastructure devs, “devops”: pay attention!

Correct code matters here: hard to test -> correctness can help

Types

So hard to catch things without them

Monadic IO + types mean you catch things before you try

Trying/experimenting can be $$ expensive...

(still learning this, all new to me)

Aside

Our deepest excursions into FP are by a new hire/ recent graduate**

He builds and maintains some of our most important systems.

** the future is bright, I hope I can keep up.

Types helped with

Ignored messages
Bad pattern matching
Misconfiguration/timing of server creation
Avoiding “stringly typed” messages

All “real world” things types have help us catch with a friendly compile error**

** may not actually be friendly

Types didn’t help with…

```ruby
def ec2
  Fog::Compute.new(:provider => 'AWS',
                   :aws_secret_access_key => ENV['EC2_SECRET_KEY'],
                   :aws_access_key_id => ENV['EC2_ACCESS_KEY'])
end

def tenured? (instance)
  instance.created_at && (instance.created_at < Chronic.parse('50 minutes
ago'))
end

def alive? (instance)
  instance.state == 'running' or instance.state == 'stopped'
end

zombies = ec2.servers.select { |i| i.tags.empty? && tenured?(i) && alive?(i) }
Parallel.each(zombies, :in_threads => 15) do |zombie|
  begin
    puts "Terminating zombie node #{zombie.id}"
    ec2.servers.get(zombie.id).destroy
  end
end
```

True story

Currying

Server lifecycle: Reserved->Launching->Update (user data)->Volume Create->Volume Attach->initialise/start

Accumulate setup data via partial application

Instead of an object that has mutating state, a function you partially apply to accumulate data.

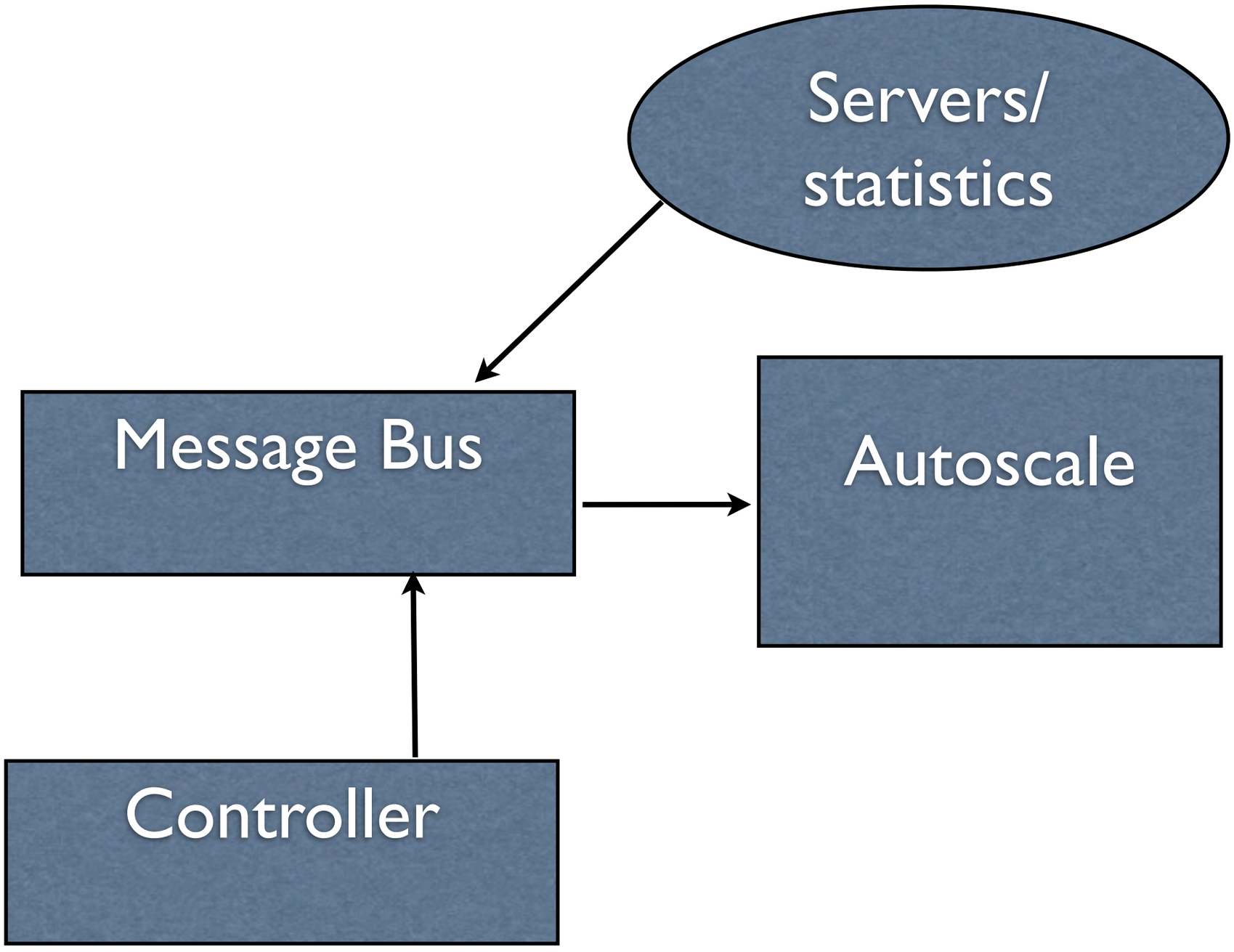
Good “beginner” FP concept (powerful, simple)

Small things

But every little bit helps.

The pure FP is my ideal, rarely reached (so far)

Another example (autoscale)


Servers/
statistics
Message Bus
Autoscale
Controller

Auto scaling

`F(last minute stats, previous data window) -> “Suggestion”` to scale up, or down (or in or out)

Fundamentally calculation, fundamentally functional.

Built in Erlang. Never restarted.

Side effects

Push out side effects to other side of the message bus

Let a nasty app handle the side effects

Messages are signals, suggestions, idempotent

Didn’t have to go the whole Monad for this

Developers Developers

Don’t “sell” to your developers by:
 - saying monad too often
 - saying “it’s easy”
 - showing that it can be just as easy/familiar as what they have

Instead...

Find the functions

Find the functions in what they do
Find the calculations (eg core of autoscaling)

Advanced:
Separate the program definition from the execution (Jeds talk)

Intermediate:
Currying, Higher order functions, Types (good ones)

Unlearning OO

Erlang and Clojure: both excellent at teaching people to forget about OO.

Scala: challenge. Temptation always there.
∴ Use object/package as namespace (avoid classes)

Lack of implicit state allows “Erlang Magic”

Advocating FP ...

Ruby Slippers

Concept from _why

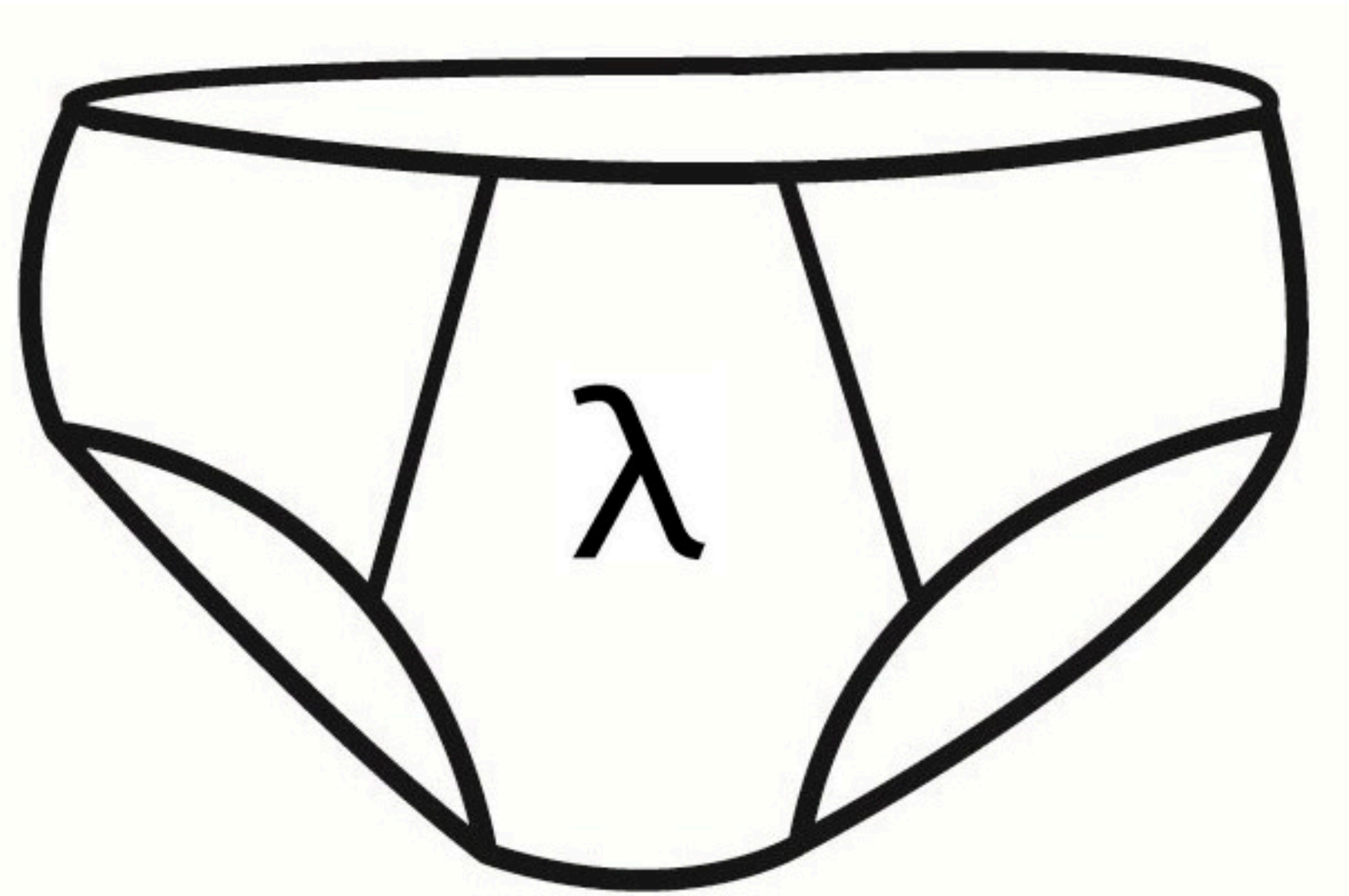
You can’t wear anything you want, but you can wear ruby slippers

http://viewsourcecode.org/why/hacking/wearingRubySlippersToWork.html

Use ruby for small, but necessary, tools/scripts

Introducing ...

Lambda Underpants



Lambda Underpants 

http://lambdaunderpants.com

Use FP for small, but necessary services, scripts

Disposable, possibly forgettable, sneaky.

example: Kit by @nkpart
me: small clojure services that deal with github api

lambdaunderpants.com

Lambda Underpants | About

Lambda Underpants

You used to wear ruby slippers, now wear lambda underpants

Manifesto

- Sneak in FP when you can
- Composition over coupling
- Libraries over frameworks
- Blah blah blah
- Don't take this seriously
- Purity over side effects

What?

Read about ruby slippers and think of ways you could make things better by applying FP in the smallest possible way in an analagous fashion

Micro Services

Small http services (typically) - REST

perfect for FP: F(request) -> response

Examples:

- github crawling
- monitoring cloud usage (instances)
- admin interfaces and utilities

Micro Services

Pick ones that are “transforming” data from one service to a client

Low risk, uncontroversial

Just do it anyway. As forgiveness later.

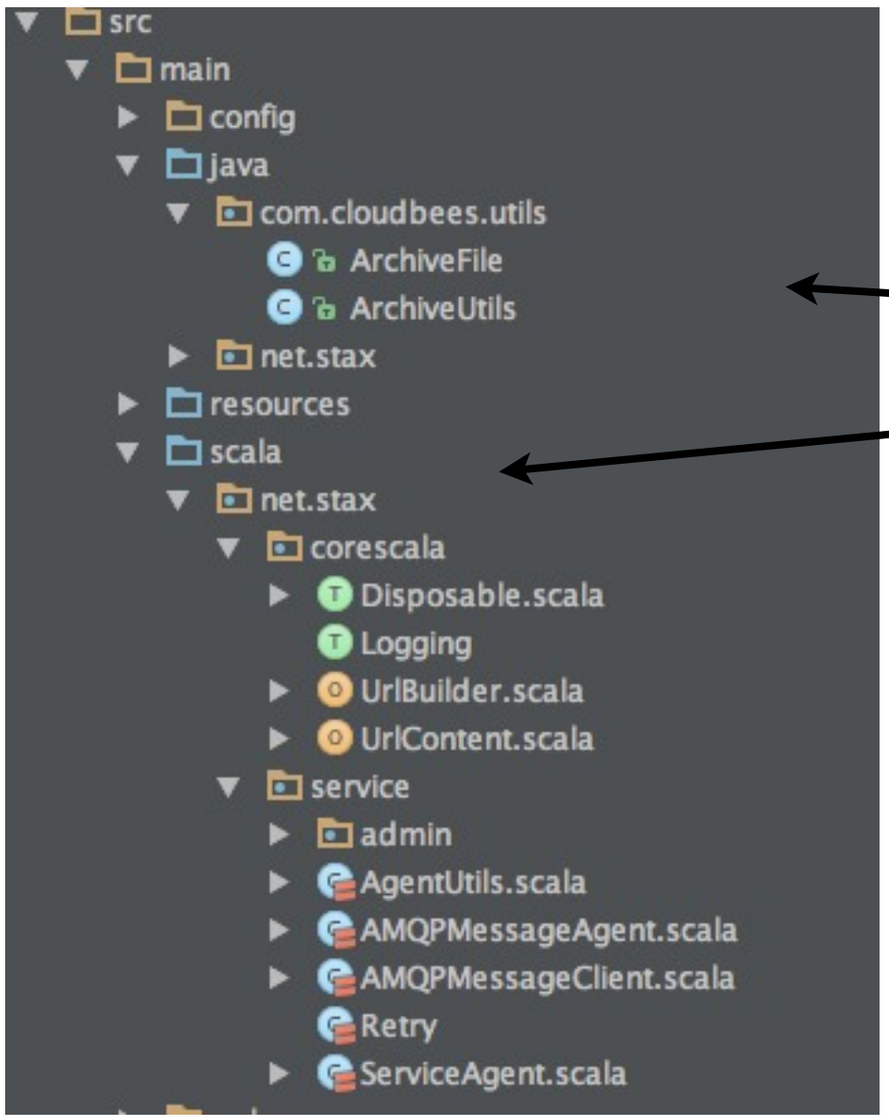
To the Clown!

Cloud Platforms (hint! vendor shilling!) make it easy to try things out that may be used seriously.

Haskell on Heroku (or CloudBees*)
Clojure anywhere, Scala anywhere

Say “cloud cloud” a lot and people will listen.

Mixed language projects



Good idea??
Relevant to JVM (and .net) environs only(?)

Ease-into-it
Jury is out...

Bad questions to hear

(when advocating)

If you hear these asked, probably will have a bad time:

How will we hire people?
Does it work for large teams?

Final Observation

Developers who have fondness for emacs/vim (over IDE) find things easier

FP invites “change this small bit, see what happens” exploration

No real tool barriers.

If I can do this, anyone can.

Thank you

Michael Neale
https://twitter.com/michaelneale
https://developer.cloudbees.com