Uses cases for advanced FP at x.ai

A beginner friendly presentation

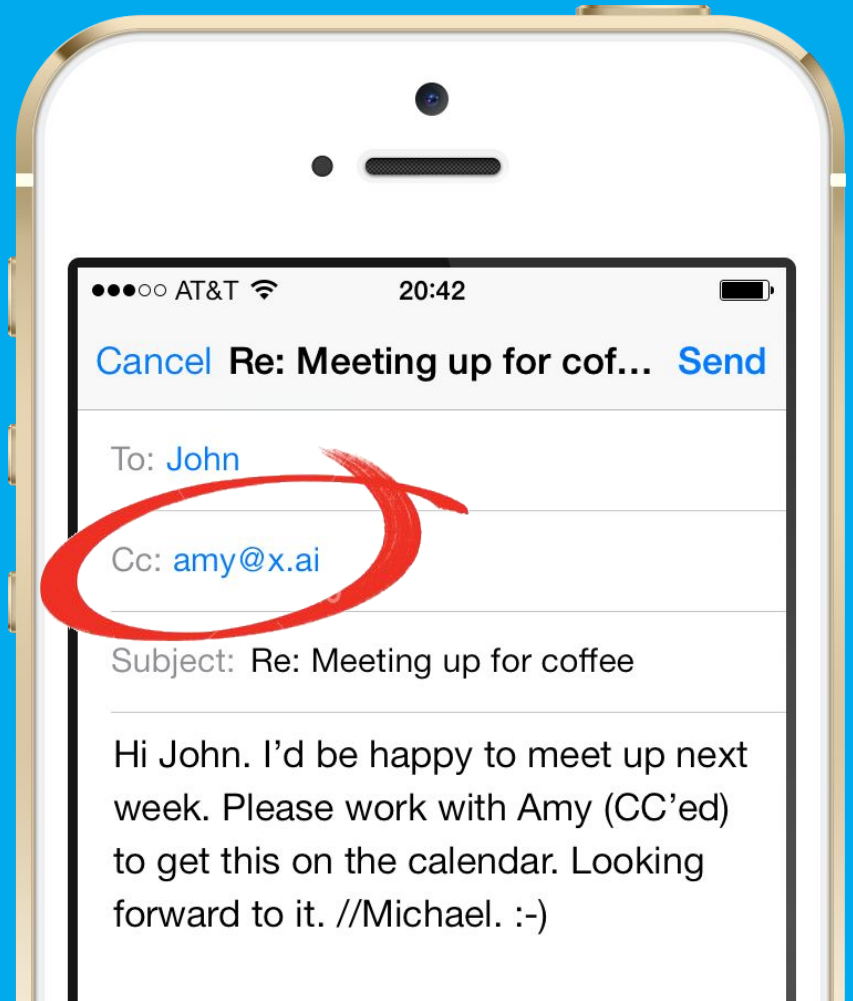
Jan Christopher Vogt / @cvogt

SFSCALA NOVEMBER 7TH 2015 SAN FRANCISCO

What is x.ai?

- Amy and Andrew
- Artificial Intelligence Personal Assistants
- Focused: Meetings - no more no less.
- No app. Email!
- Mostly Scala
- >20 Scala engineers & data scientists

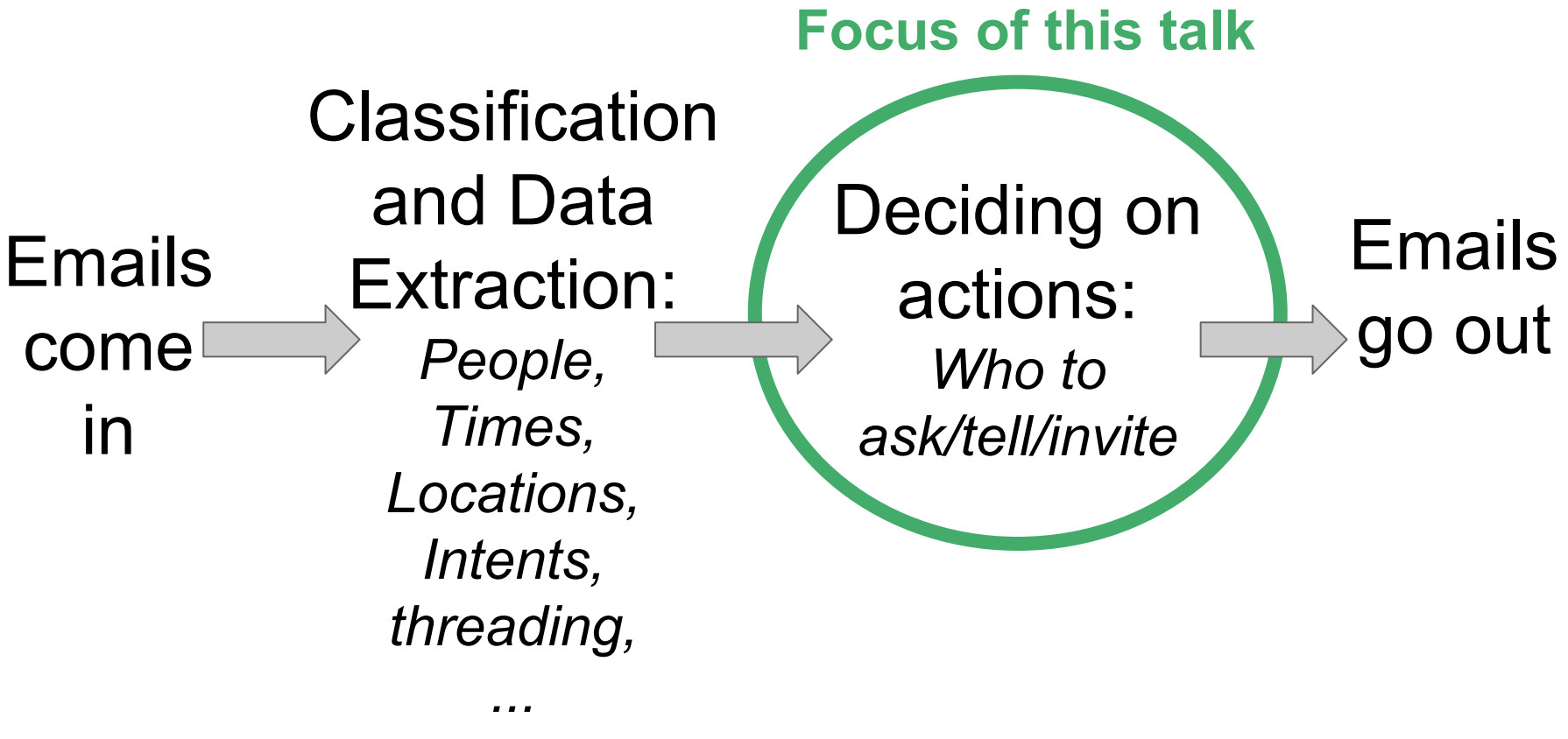
x.ai


●●●○○ AT&T
20:42
Cancel
Re: Meeting up for cof...
Send
To: John
Cc: amy@x.ai
Subject: Re: Meeting up for coffee
Hi John. I'd be happy to meet up next week. Please work with Amy (CC'ed) to get this on the calendar. Looking forward to it. //Michael. :-)

Who am I?

- Jan Christopher Vogt
- worked at Martin’s lab in Switzerland
- co-author of Slick
- Senior Software Engineer at x.ai

Amy's & Andrew's brain

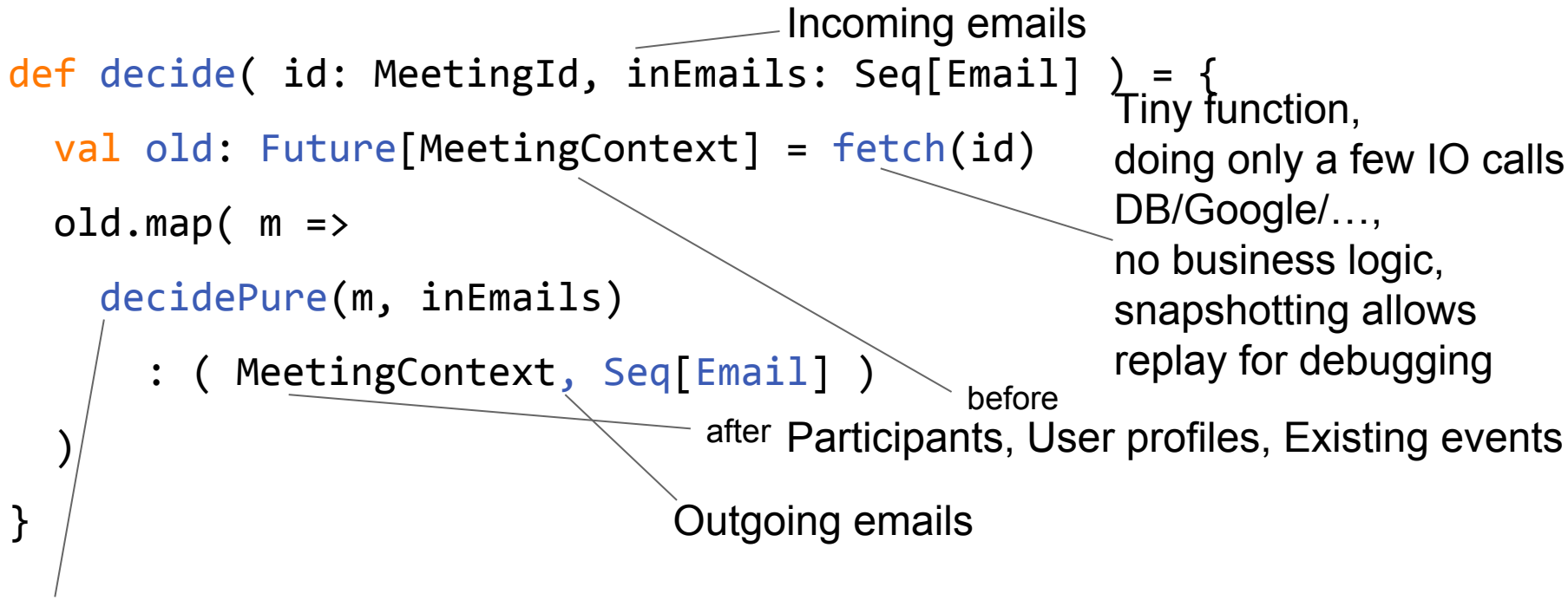


This talk

1. Functional architecture
   mutability, IO

2. Functional pattern / idioms / interfaces
   simple concepts with alien names

x.ai's Functional Architecture



Pure function, complete decision making business logic, no Future, no DateTime.now, no id generation, easy testing as a whole, no complication through IO, easy to speed test and optimize, good stack traces, we do logging, println for debugging

Context or rather … mutable State

- sometimes became inconsistent -> bugs
- made it harder to change our minds
  - who is coming
    - if we save, hard to change afterwards
    - if we compute on the fly easy
    - however: harder if we sent out emails
  - no access to older info
    - e.g. the times we suggested before
      - introduction of timeSuggestionsPrevious
      - easier: history

Solution?

Use incoming and outgoing emails as a log of what happened. Store data there, e.g. Amy’s outgoing emails contain the times she suggested.

Functional patterns / interfaces

This talk:

- informal
- imprecise
- incomplete
- BUT: explains why to care
- because when you care the rest emerges

All problems in computer science can be solved by another level of indirection

...except for the problem of too many level of indirection.

(https://en.wikipedia.org/wiki/Indirection)

Abstraction via Indirection (auxiliary)

many basic FP concepts gain power this way

unified api, regardless of location, existence, time, repetition, …

Used by Futures, Options, Lists, Slick, etc.

Transformation / Sequencing

Indirect[A] => (A => B) => Indirect[B]

In Scala: .map

Where do we use it? e.g. Future, Options, List, ...

Composition

(Indirect[A], Indirect[B]) => Indirect[(A,B)]

Missing from Scala. In Scalaz: |@|

Where we use it

(fetchPerson() |@| fetchAddress()){ (p,a) => p.name + “ lives on “ + a.street }

Fusing

Indirect[Indirect[A]] => Indirect[A]

in Scala: .flatten

Sequential Fusing: Fusing + Transformation

Indirect[A] => (A => Indirect[B]) => Indirect[B]

In Scala: .flatMap

Surprise - official terms

- Transformation: **Functor**
- Composition: **Applicative**
- Sequential fusing (Fusing + Transformation): **Monad**

What is the problem?

...the problem of too many level of indirection.

Why does that happen?

if all you have is a hammer, everything looks like a nail…

(https://en.wikipedia.org/wiki/Law_of_the_instrument)

Over-engineering

it’s easy:

choose any pattern that solves the business problem

Not over-engineering

it’s hard:

choose the pattern that solves the business problem most elegantly

Why do we overengineer?

humans love to recognize patterns

(“this reminds me of”, “first thing comes to mind”, “look like elephant”)

not easy finding the simplest

(“let’s take a step back”, “think this through”)

Why is that a problem?

too complicated pattern ->
too many indirections ->
more mental steps in understanding

human brain is limited

Avoid or recover from over-engineering

- use concepts very carefully - only when they pay off
- try to really understand their properties and the properties of your problem
- we learn as we go - be ready to recover from mistakes

Intuition vs Strategy

Intuition

human pattern recognition optimized for speed

Strategy

human pattern recognition optimized for accuracy

Also: Hack your brain

Kill your darlings

(Origin unclear, but used by William Faulkner, Oscar Wilde, Stephen King, and many more)

http://www.slate.com/blogs/browbeat/2013/10/18/_kill_your_darlings_writing_advice_what_writer_really_said_to_murder_your.html

Concepts we applied not well in some cases

- Reader: we delayed application with no gain

Careful: Stateful FP

A => A

val chained: A => A = Function.chain(l: List[A => A])

requires each function to update A. It’s flexible, but can be hard to track. Do you need the flexibility?

“Can be seen as” aka. Type Classes

- piggy bag extra stuff onto types
  - operations
  - meta data
  - laws
  - properties
- can do X (by providing the implementation)

Scalaz: Scala’s Types “can be seen as” Monads, Functors, Applicatives, etc

Scala: Ordering, CanBuildFrom

Play-json: “can be serialized”

Aggregator - official term: Semigroup / Monoid

```
case class Aggregator[A]( f: (A, A) => A )
val plus = Aggregator(_ + _)
plus.f(1,2) == 3, or List(1,2,3).reduce(plus.f) = 6
```

But why?!? Automatic composition using implicits!

```
Option(3) |+| Option(4) == Option(7)
```

We made our lives harder than necessary sometimes

- Monoid: when Semigroup was enough
- Semigroup: when there was no nested types

Lenses: updating nested case classes made easy

are pairs (getter/setter)

get: person.address.street
set: person.copy(address=person.address.copy(street = “Park Lane”))

x.ai’s ah-hoc setters (based on Monocle):

person.lens(_.address.street).set(“Park Lane”) : Person

person.lens(_.address.street).modify(old => old ++ “ Lane”) : Person

```
    ...
    .lens(_.age).set(999)
```

Summary

- Self-contained, easy to introduce, immediately useful FP concepts:
  import scalaz._; import Scalaz._
  Equal **===,** Applicative **|@|**, Semigroup **|+|**, **.lens(...).modify(...)** Lenses
- Worth investigating in more detail as time permits:
  Type Classes, Monads, Functors
- BUT, resist too much intuition, use strategy, try to avoid concept misuse
- Still, experimentation means making mistakes but learning from them
- So: make sure you recover

chris @ human.x.ai

Senior Backend Engineer

Twitter: @cvogt @xdotai
Github: @cvogt
Slides, etc **https://github.com/cvogt/talk-2015-11-10**