REAL WORLD

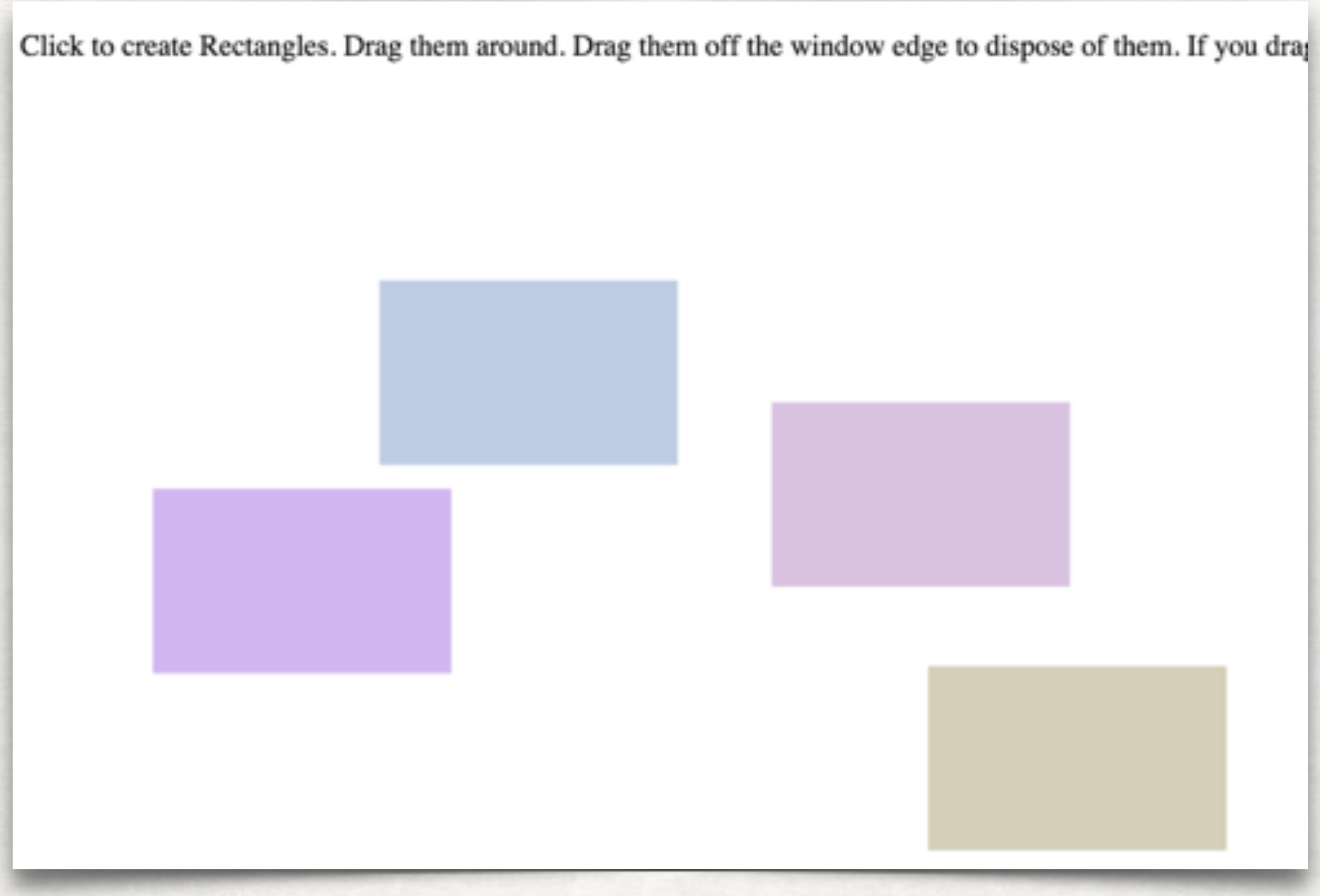
STATE MONADS IN GESTURE RECOGNITION

BEN HUTCHISON, FEB 2016, MELBOURNE SCALA

PART 1
THE PROBLEM

Dragging On An Html Canvas

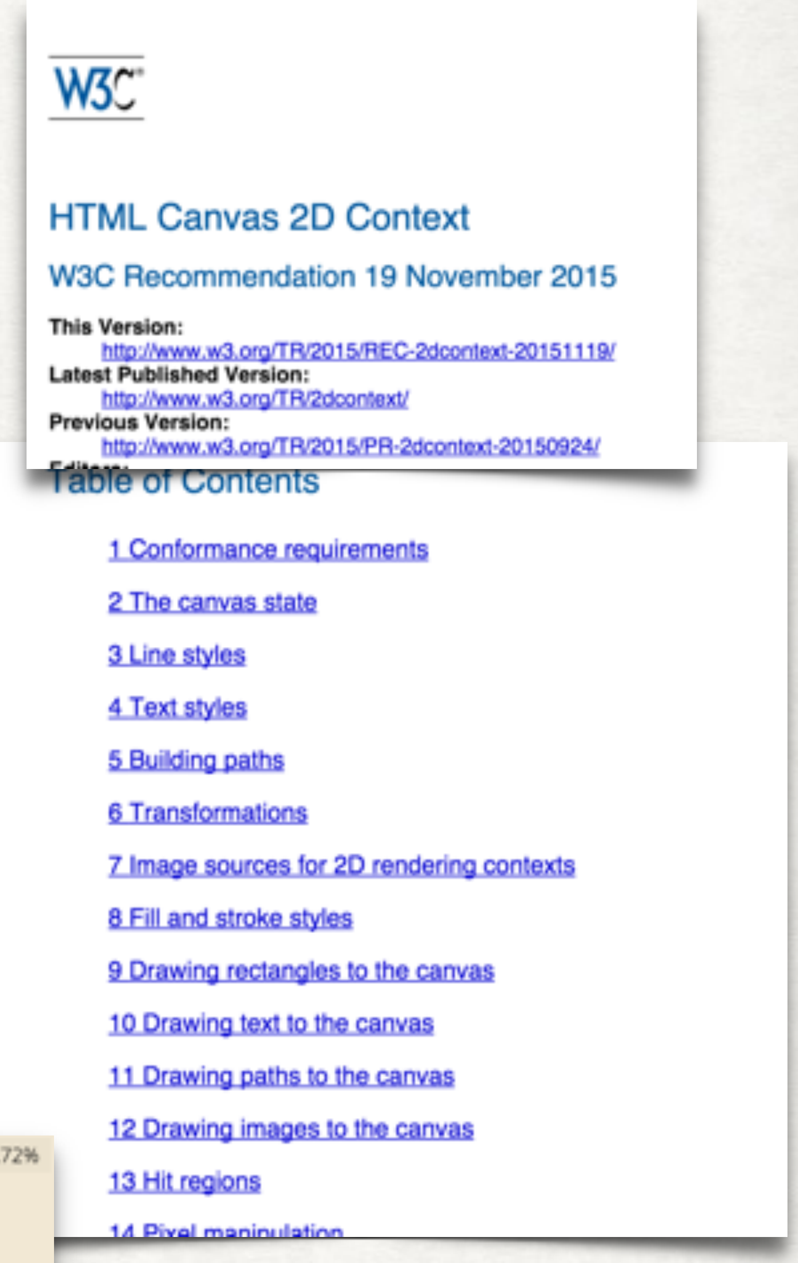
- I wanted to support drag and drop type behaviors in an animation library I'm building with Scala.js and the HTML Canvas



Click to create Rectangles. Drag them around. Drag them off the window edge to dispose of them. If you drag

Html Canvases (Canvii?)

- Performant, portable and versatile API for drawing vector graphics
  - Its imperative, based on commands e.g.
    - fillRect(x, y, w, h)
    - lineTo(x, y)
    - fillStyle=(cssColor)
    - transform(a, b, c, d, e, f)
  - A canvas has no internal structure other than what the programmer creates
  - No drag support



W3C

HTML Canvas 2D Context

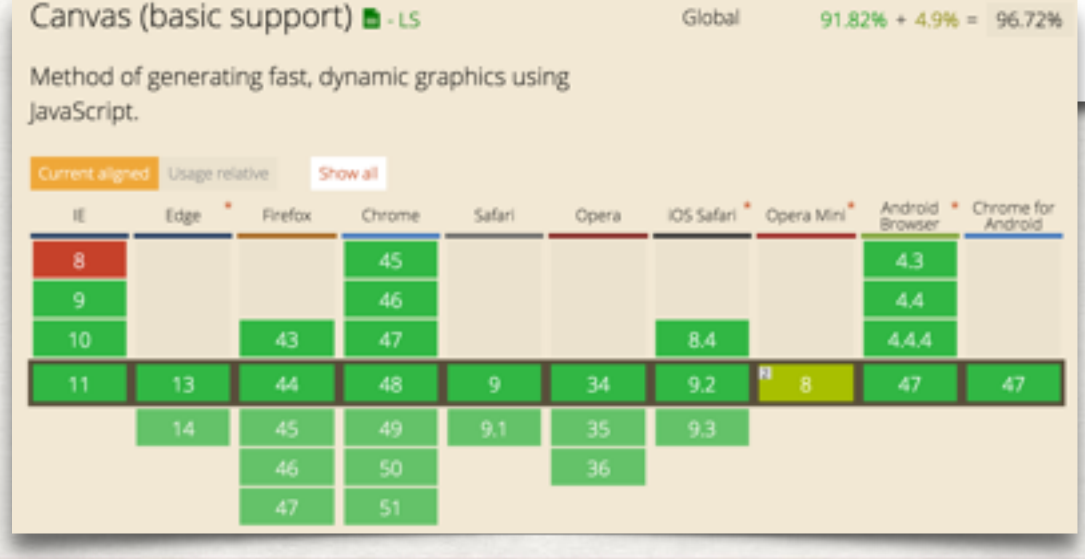
W3C Recommendation 19 November 2015

This Version:
http://www.w3.org/TR/2015/REC-2dcontext-20151119/
Latest Published Version:
http://www.w3.org/TR/2dcontext/
Previous Version:
http://www.w3.org/TR/2015/PR-2dcontext-20150924/

W3C Recommendation

Table of Contents

1 Conformance requirements
2 The canvas state
3 Line styles
4 Text styles
5 Building paths
6 Transformations
7 Image sources for 2D rendering contexts
8 Fill and stroke styles
9 Drawing rectangles to the canvas
10 Drawing text to the canvas
11 Drawing paths to the canvas
12 Drawing images to the canvas
13 Hit regions
14 Pixel manipulation



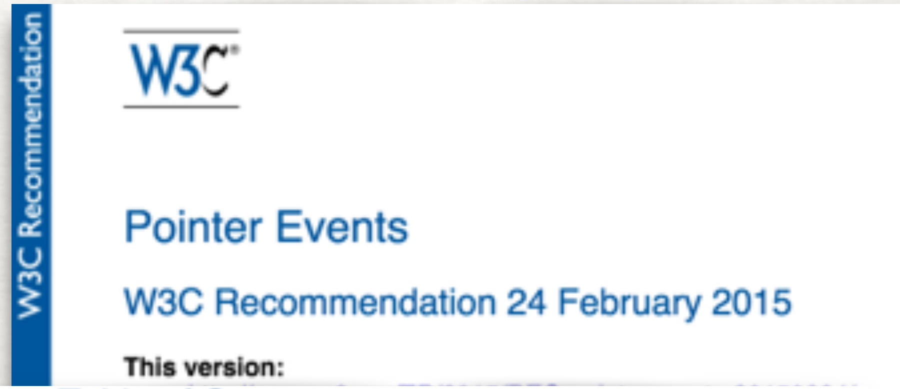
Canvas (basic support) - LS Global 91.82% + 4.9% = 96.72%

Method of generating fast, dynamic graphics using JavaScript.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
8			45					4.3	
9			46					4.4	
10		43	47			8.4		4.4.4	
11	13	44	48	9	34	9.2	8	47	47
	14	45	49	9.1	35	9.3			
		46	50		36				
		47	51						

Pointer Events

- I want to target all web-capable devices, whether touch or mouse based
- The emerging W3C Pointer Events API abstracts over mouse & touch events
    - Polyfills required to support most browsers. I used Points.js
- Because a canvas has no internal structure, the only events we'll get inside are "raw" e.g.
    - pointerdown
    - pointerup
    - pointermove
    - pointerleave



Pointer Events

W3C Recommendation 24 February 2015

This version:

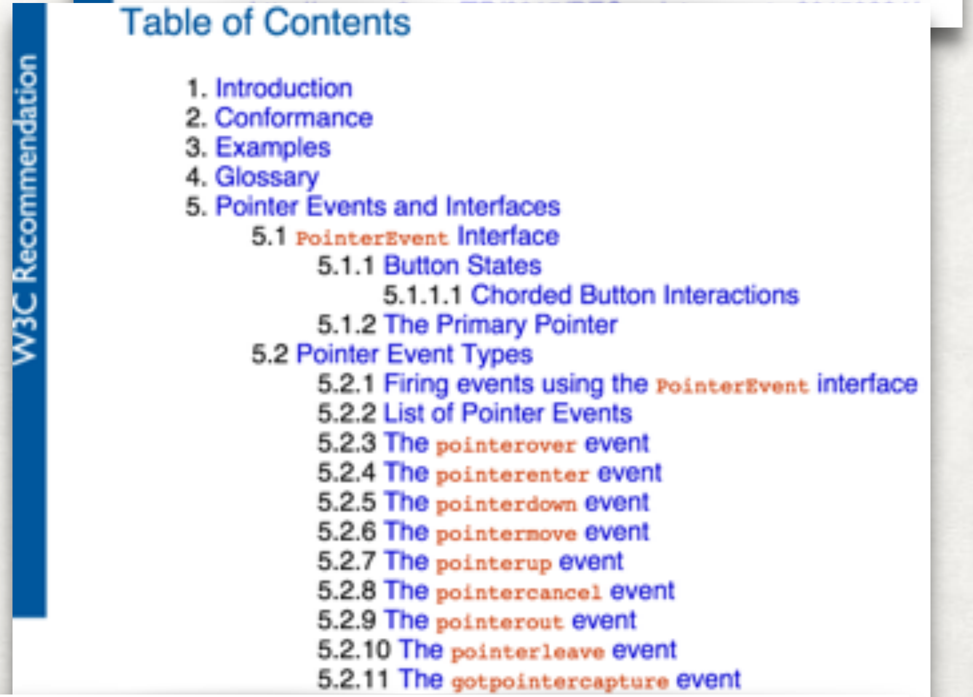
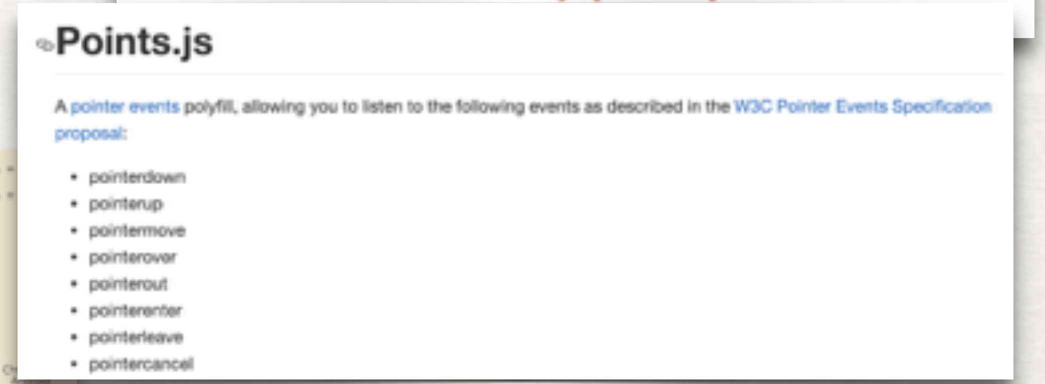


Table of Contents

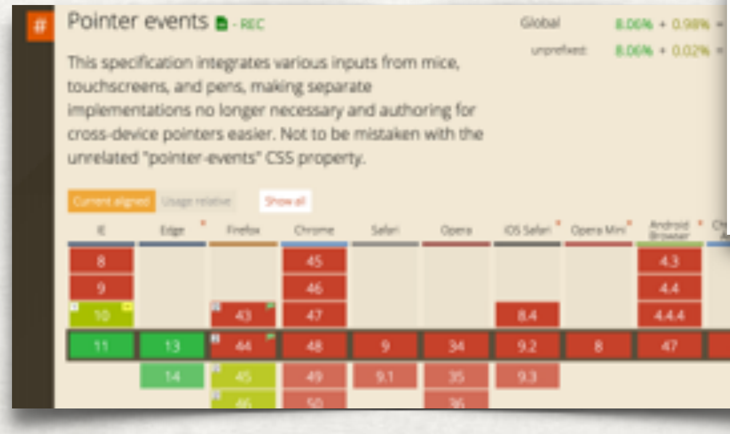
1. Introduction
2. Conformance
3. Examples
4. Glossary
5. Pointer Events and Interfaces
5.1 `PointerEvent` Interface
5.1.1 Button States
5.1.1.1 Chorded Button Interactions
5.1.2 The Primary Pointer
5.2 Pointer Event Types
5.2.1 Firing events using the `PointerEvent` interface
5.2.2 List of Pointer Events
5.2.3 The `pointerover` event
5.2.4 The `pointerenter` event
5.2.5 The `pointerdown` event
5.2.6 The `pointermove` event
5.2.7 The `pointerup` event
5.2.8 The `pointercancel` event
5.2.9 The `pointerout` event
5.2.10 The `pointerleave` event
5.2.11 The `gotpointercapture` event



Points.js

A pointer events polyfill, allowing you to listen to the following events as described in the W3C Pointer Events Specification proposal:

- pointerdown
- pointerup
- pointermove
- pointerover
- pointerout
- pointerenter
- pointerleave
- pointercancel



Pointer events - REC

Global 8.06% + 0.98% =

unprefixed: 8.06% + 0.02% =

This specification integrates various inputs from mice, touchscreens, and pens, making separate implementations no longer necessary and authoring for cross-device pointers easier. Not to be mistaken with the unrelated "pointer-events" CSS property.

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Ch
8			45					4.3	
9			46					4.4	
10		43	47			8.4		4.4.4	
11	13	44	48	9	34	9.2	8	47	47
	14	45	49	9.1	35	9.3			
		46	50		36				

Scala.Js And Scala-Js-Dom

- Scala.js compiles Scala code to Javascript
  - Near-transparent interop with native JS code
- Facade libraries put a typed interface over JS APIs
  - scala-js-dom covers a lot of core W3C browser APIs including canvas
  - Not Pointer Events - Gesture lib defines them directly

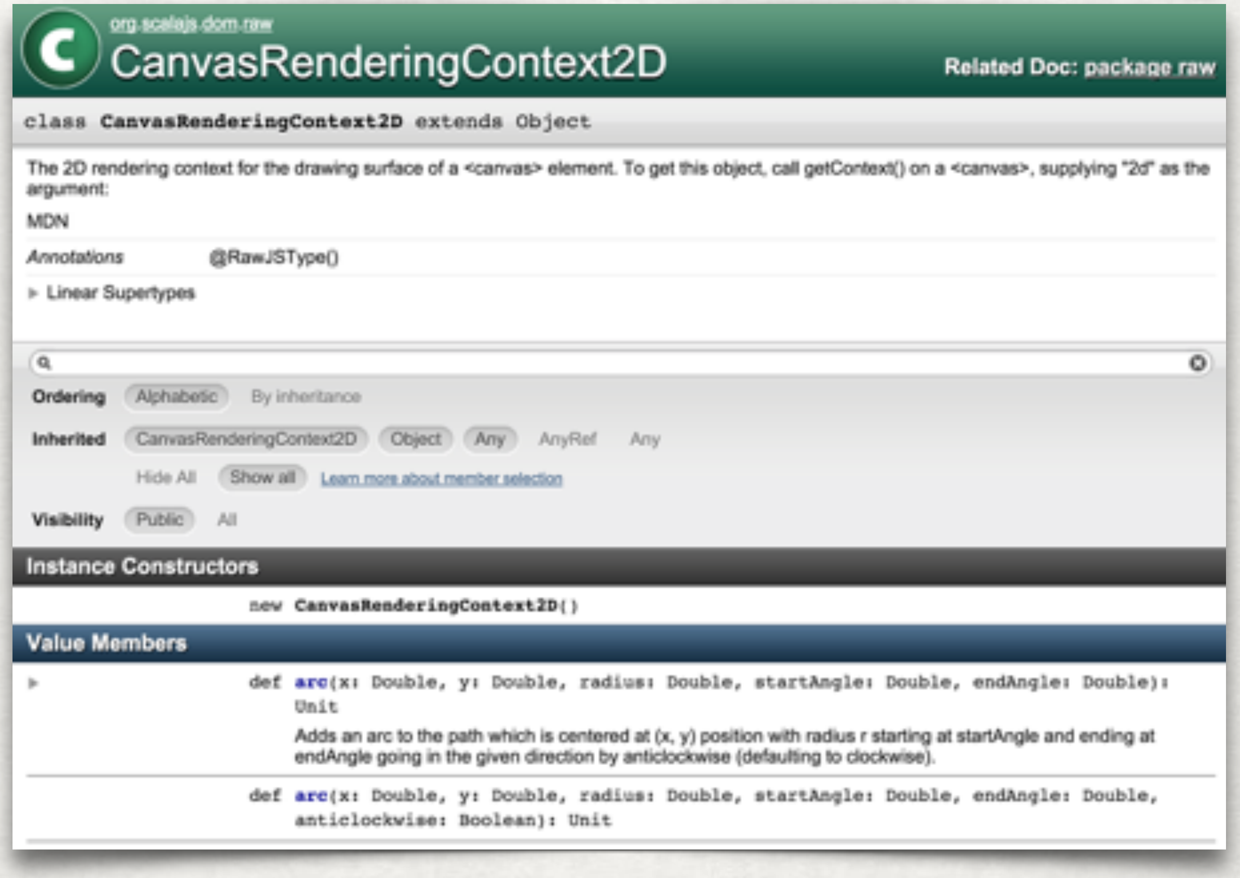
Scala.js

0.6.7

A safer way to build robust front-end web applications!

Correctness

Strong typing guarantees your code is free of silly mistakes; no more mixing up strings or numbers, forgetting what



CanvasRenderingContext2D

```
class CanvasRenderingContext2D extends Object
```

The 2D rendering context for the drawing surface of a <canvas> element. To get this object, call getContext() on a <canvas>, supplying "2d" as the argument:

MDN

Annotations @RawJSType()

Linear Supertypes

Instance Constructors

```
new CanvasRenderingContext2D()
```

Value Members

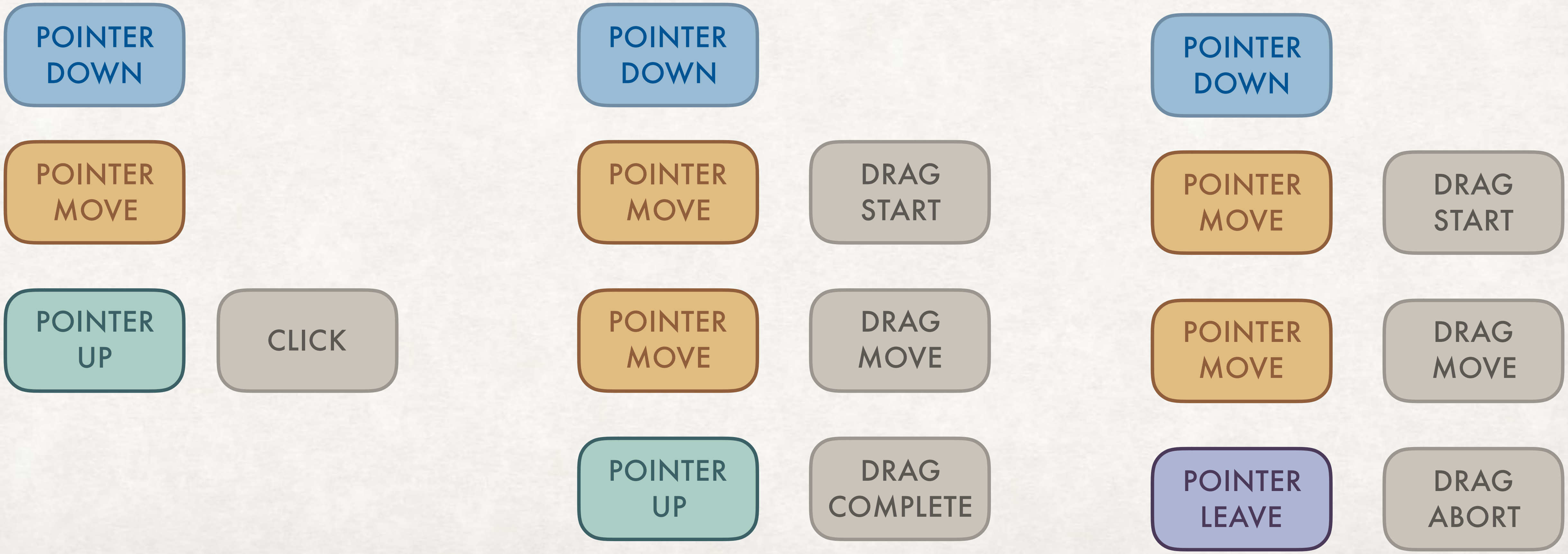
```
def arc(x: Double, y: Double, radius: Double, startAngle: Double, endAngle: Double):
    Unit
```

Adds an arc to the path which is centered at (x, y) position with radius r starting at startAngle and ending at endAngle going in the given direction by anticlockwise (defaulting to clockwise).

```
def arc(x: Double, y: Double, radius: Double, startAngle: Double, endAngle: Double,
    anticlockwise: Boolean): Unit
```

Gesture Recognition

- The process of recognising patterns in a stream of low-level events and emitting higher-level gesture events
  - Inherently *stateful* process
  - An incomplete gesture may be ambiguous



POINTER DOWN
POINTER MOVE
POINTER UP — CLICK

POINTER DOWN
POINTER MOVE — DRAG START
POINTER MOVE — DRAG MOVE
POINTER UP — DRAG COMPLETE

POINTER DOWN
POINTER MOVE — DRAG START
POINTER MOVE — DRAG MOVE
POINTER LEAVE — DRAG ABORT

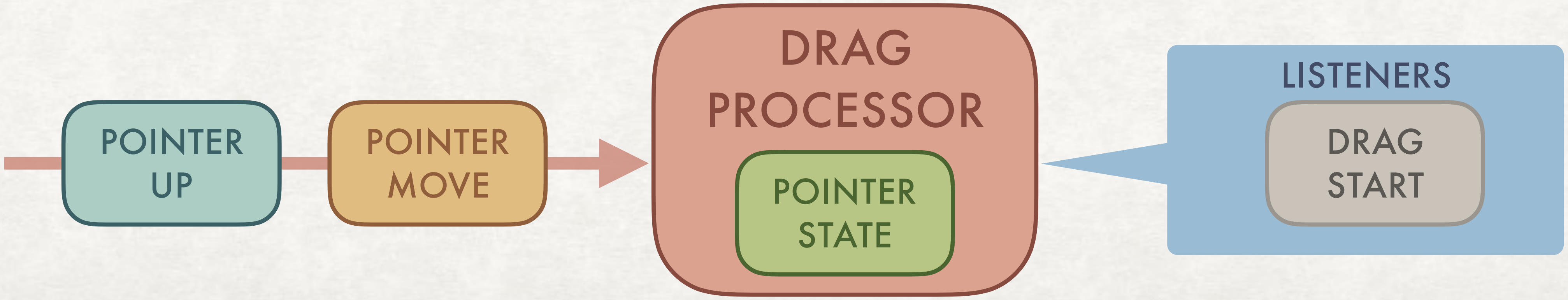
States And Gestures

```
sealed trait PointerState
case class Up() extends PointerState
case class Down(p: Vec2d, timestamp: Long) extends PointerState
case class Drag(from: Vec2d, fromTimestamp: Long,
                to: Vec2d, toTimestamp: Long) extends PointerState

sealed trait GestureEvent
case class Click(p: Vec2d, timestamp: Long) extends GestureEvent
case class DragStart(from: Vec2d, fromTimestamp: Long,
  to: Vec2d, toTimestamp: Long, delta: Vec2d) extends GestureEvent
case class DragMove(from: Vec2d, fromTimestamp: Long,
  to: Vec2d, toTimestamp: Long, delta: Vec2d) extends GestureEvent
case class DragComplete(from: Vec2d, fromTimestamp: Long,
  to: Vec2d, toTimestamp: Long, delta: Vec2d) extends GestureEvent
case class DragAbort(from: Vec2d, fromTimestamp: Long,
  to: Vec2d, toTimestamp: Long) extends GestureEvent
case class Invalid(msg: String, pointerEvent2: PointerEvent) extends
  GestureEvent
case object Noop extends GestureEvent
```

Why Not The Classic Object-Oriented Approach?

- The state monad based design was in fact my *second* attempt
- Initially I used a DragProcessor written in object-oriented style
- Main problem: it was harder to test
  - PointerState encapsulated away inside processor
  - Need to mock out listeners to verify emitted events



What Does “Easy To Test” Look Like?

- IMO the easiest code to test would be
  - Specify the current state
  - Specify the input PointEvent
- and it returns
  - The new state
  - The recognised gesture, if any

Input, Current State => (Next State, Gesture)

What Does "Easy To Test" Look Like?

- IMO the easiest code to test would be
  - Specify the current state
  - Specify the input PointEvent
- and it returns
  - The new state
  - The recognised gesture, if any

Input, Current State => (Next State, Gesture)

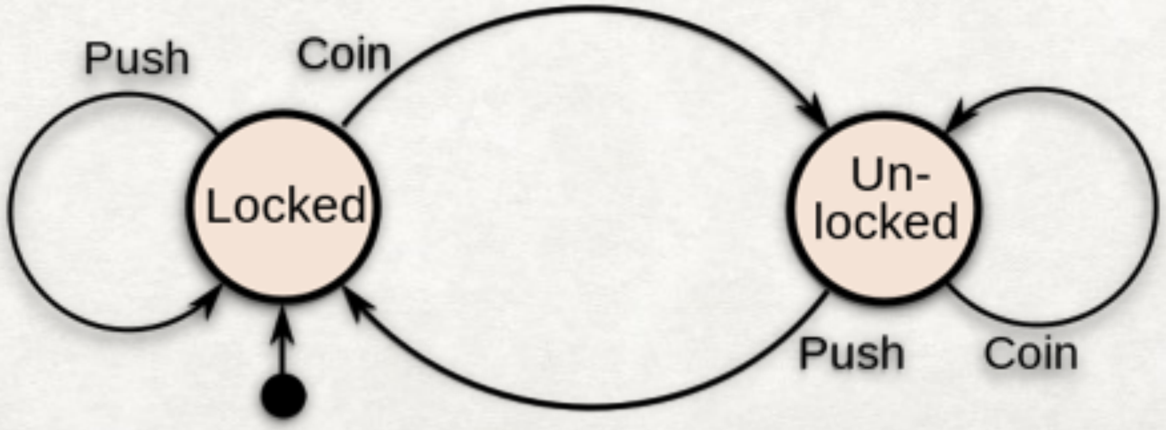
```
val (s, g) = eventSequence(initialState = Up())(
  PointerDown((0, 0), 0L), PointerMove((20, 20), 10L), PointerUp((30, 30), 20L))

(s must_== Up()) and (g must_== DragComplete((0, 0), 0L, (30, 30), 20L, (10,
10)))
```

PART 2
FINITE STATE MACHINES
STATE MONADS

Finite State Machines

- FSMs are my favourite way to think about state
  - What are all the states the system can be in?
  - How does it transition between states?
  - What should happen upon transition?
  - State diagram for a turnstile



Modelling States & Transitions

- A finite set of state are well modelled using a case class hierarchy

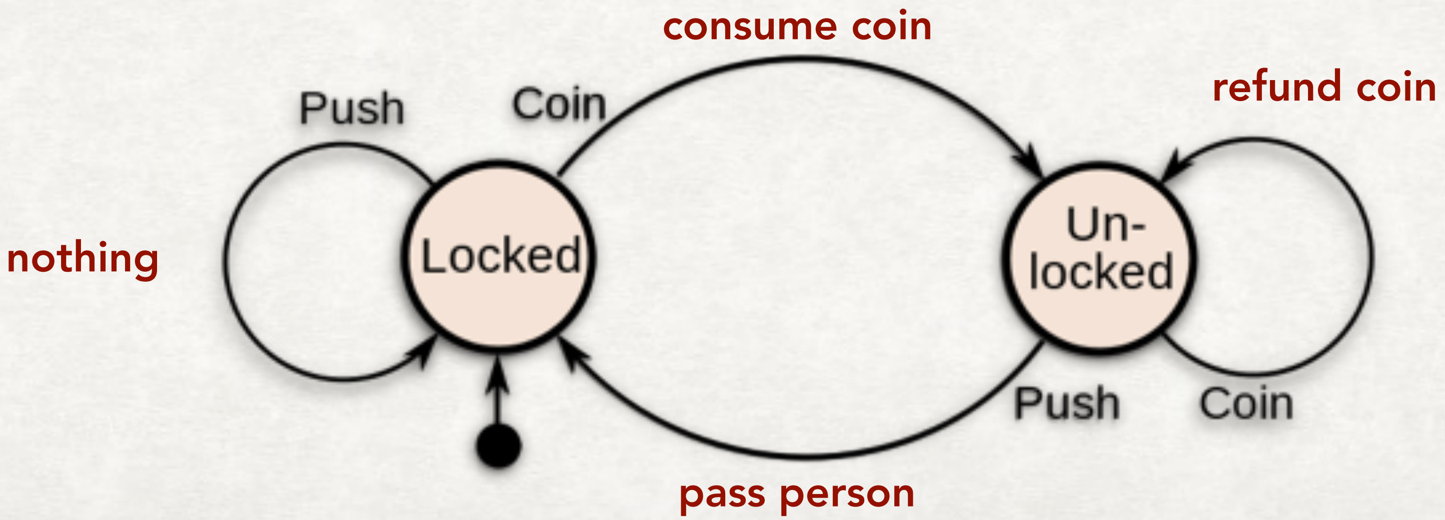
```
sealed trait TurnstileState
case object Locked extends TurnstileState
case object Unlocked extends TurnstileState
```

- Transitions could be modelled as

```
    f: (Input, TurnstileState) => TurnstileState
```

- ..but we also want output actions to occur upon state change, so

```
  f: (Input, TurnstileState) => (TurnstileState, Output)
```



State Monads Are State Transitions

- A state monads is a function f: State => (State, Action)

- It represents a *transition path* in a FSM, *not a state*. Be warned the terminology can be confusing because libraries typically wrap f in a data type called State. But its not a state.

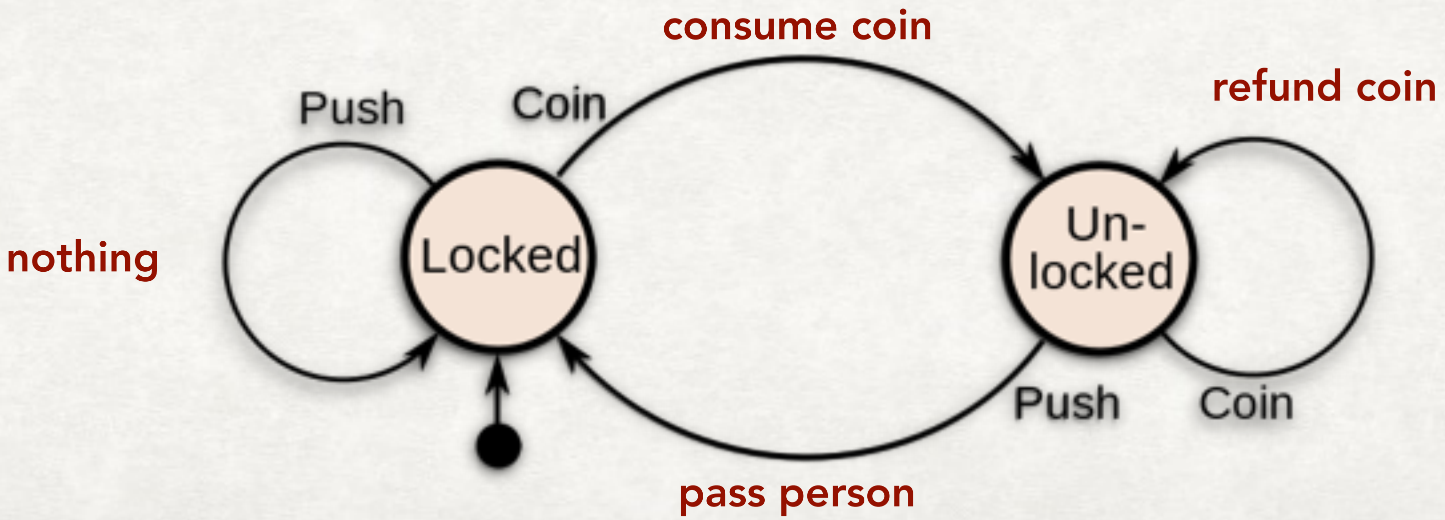
- Any inputs are assumed to have already been provided. Using currying, we can write functions that accept required input and yield the State monad

```
def insertCoin(c: Coin): TurnstileState => (TurnstileState, Action)
```

- Monads are characterised by their join (aka flatten) operation:

```
def join: State[S, State[S, A]] => State[S, A]
```

  For state monads, join means “chain” the state transitions together



PART 3
USING STATE MONADS IN GESTURE

Responding To Pointerdown

- The State[S, A] data type just wraps a function you define with signature f: S => (S, A), providing some useful state monad operations
  - S means "state", action means "Action"
  - Typically you pattern match on the initial state
- PointerDown rules in prose and then in code
  - "if we're in an Up state, transition to Down state, recording when and where, and emit no gesture"
  - "a PointerDown event doesn't make sense if we're already down or dragging"

```
def pointerDown(pe: PointerDown) = State[PointerState, GestureEvent](ps => ps match {
  case Up() =>
    (Down(pe.p, pe.timestamp), Noop)
  case _ => invalid(pe, ps)
})
```

Responding To Pointermove

- PointerMove rules in prose and then in code
    - “if we’re in an Up state, stay there and emit no gesture”
    - “if we’re Down, check how far we’ve travelled since we went down. If its enough to count as a drag, enter Drag state and emit a DragStart gesture”
    - “if we’re already Dragging, extend the Drag to the new location and emit a DragMove”

```
def pointerMove(pe: PointerMove) = State[PointerState, GestureEvent](ps => ps match {
  case Up() => (Up(), Noop)
  case Down(p, timestamp) =>
    if (p.distanceTo(pe.p) > dragThreshold)
      (Drag(p, timestamp, pe.p, pe.timestamp), DragStart(p, timestamp, pe.p, pe.timestamp, pe.p - 
p))
    else
      (ps, Noop)
  case Drag(from, fromTimestamp, to, toTimestamp) =>
    (Drag(from, fromTimestamp, pe.p, pe.timestamp), DragMove(from, fromTimestamp, pe.p, 
pe.timestamp, pe.p - to))
})
```

The Imperative Rind

- An remark by Simon Peyton Jones early in my FP journey left a mark on me
    - roughly “functional programs have a functional interior and an imperative rind (exterior)”
- The updated state computed by a state monad needs to be stored somewhere mutable
- The action emitted by the state monad needs to be executed

Gesture Demo

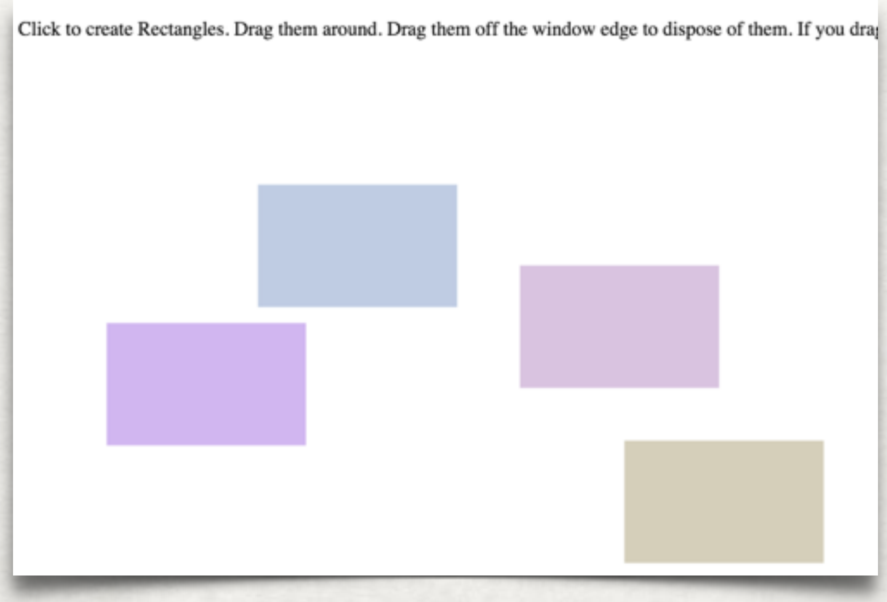
```scala
def handlePointerEvent(pe: PointerEvent) = {
  val (newState, gestureAndRegions) = gestureRegionProcessor.
    handlePointerEvent(pe, search).run(pointerAndRegionState).run

  pointerAndRegionState = newState
  interpret(gestureAndRegions)
}

def interpret(gr: GestureAndRegions[Rect]) = {
  gr match {
    case GestureAndRegions(Click(p, timestamp), None, None) =>
      def randLightValue = 180 + Random.nextInt(60)
      val randomColor = s"rgb($randLightValue, $randLightValue, $randLightValue)"
      val r = new Rect(p, Width, Height, randomColor)
      rectangles = rectangles :+ r
      draw()
    case GestureAndRegions(d: DragMove, Some(Rect(_, _, _, _, id)), _) =>
      rectangles = rectangles.map(r =>
        if (r.id == id)
          r.copy(topLeft = r.topLeft + d.delta)
        else r)
    case GestureAndRegions(d: DragAbort, Some(Rect(_, _, _, _, id)), _) =>
      rectangles = rectangles.filterNot(_.id == id)
    case GestureAndRegions(d: DragComplete, Some(Rect(_, _, _, _, srcId)), Some(Rect(_, _, _, _,
targetId))) =>
      rectangles = rectangles.map(r =>
        if (srcId != targetId && r.id == targetId)
          r.copy(cssColorString = RedColorString)
        else r)
    case _ => Noop
  }
  draw()
}
```

Tracking Drag Regions

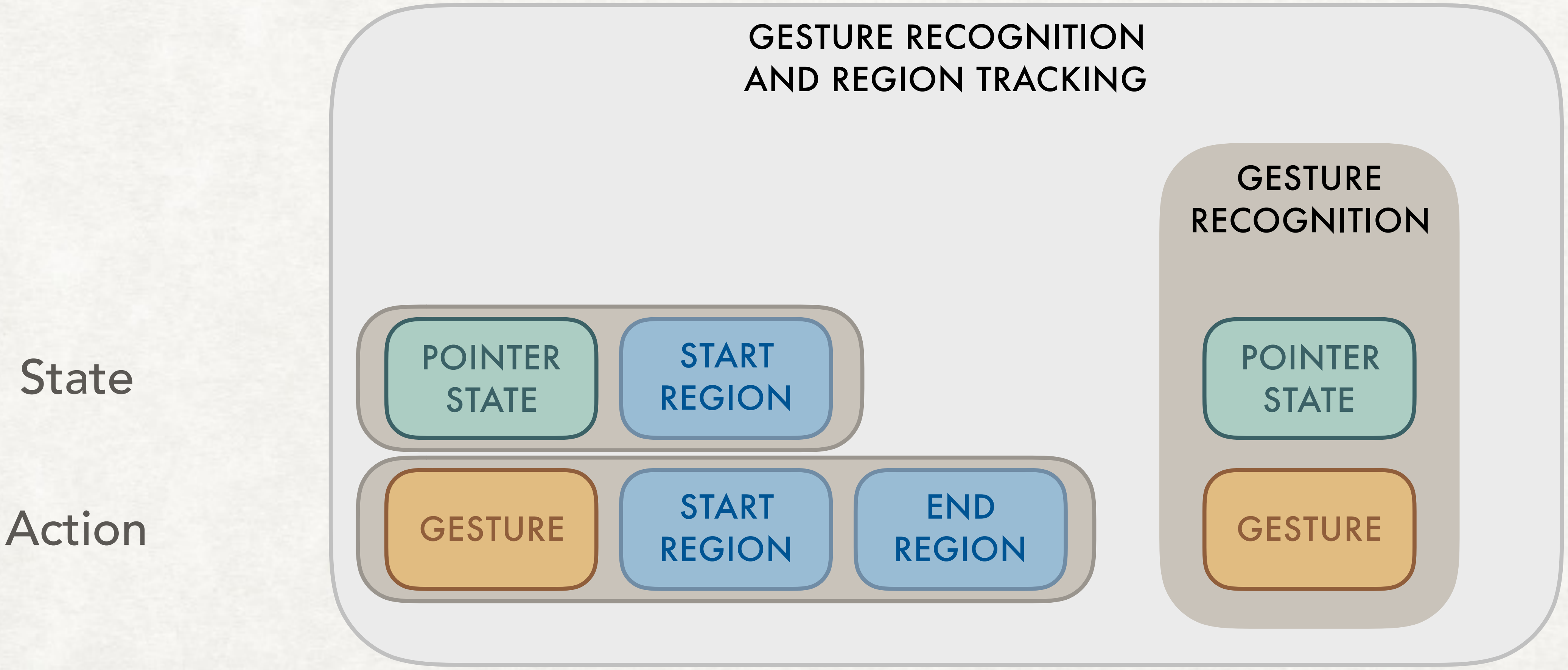
- For most applications, its not enough to know that drags have happened
  - We want to know about the object where they began, completed or passed over
  - Gesture keeps track of these objects, which it calls "Regions", and it actually doesn't care what they are
    - You give it a function to convert a point into a region of arbitrary type R, and it calls it and tracks the values



Click to create Rectangles. Drag them around. Drag them off the window edge to dispose of them. If you drag

State[PointerRegionState, GestureAndRegions[R]]

- The region-tracking State wraps the simpler gesture recognition State



State[PointerRegionState, GestureAndRegions[R]]

- The region-tracking State wraps the simpler gesture recognition State

```
case class GestureAndRegions[R](
    gesture: GestureEvent, from: Option[R], to: Option[R])

type PointerRegionState = (PointerState, Option[R])

def handlePointerEvent(pe: PointerEvent, regionSearch: Vec2d => Option[R]) =
  State[PointerRegionState, GestureAndRegions[R]] {
  case (ps, optRegion) =>
    val (ps2, g) = gestureProcess.handlePointerEvent(pe).run(ps).run
    g match {
      case DragStart(from, _, to, _, _) =>
        val fromR = regionSearch(from)
        val s = (ps2, fromR)
        val a = GestureAndRegions(g, fromR, regionSearch(to))
        (s, a)
      case DragMove(_, _, to, _, _) =>
        val s = (ps2, optRegion)
        val a = GestureAndRegions(g, optRegion, regionSearch(to))
        (s, a)
//more cases for other gestures…
}}
```

Scaling Up With State Monads

- What if the whole client application was purely functional?
  - Input: PointerEvent | Server Messages | Time
  - State: ApplicationState(PointerRegionState, AnimationState, ...)
  - Actions..? ..maybe UpdateView | ServerCall | SetCookie ..
- Need a way to compose local state monads (like State[PointerRegionState, GestureAndRegions[R]]) into app-wide State[ApplicationState, AppAction]

Scaling Up With State Monads

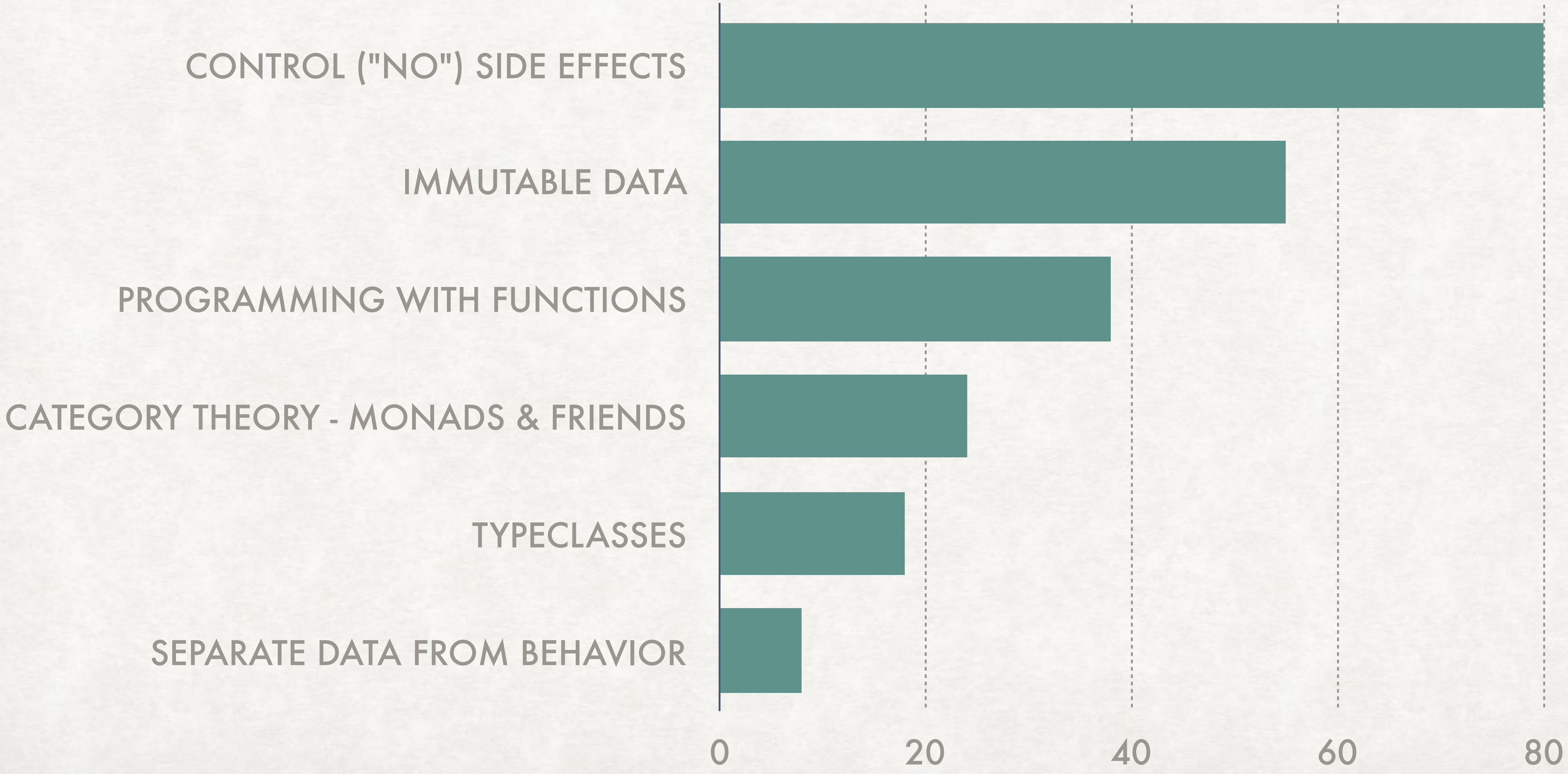
- A *Lens* is a pair of functions
  - get: S => T
  - set: (S, T) => S
- Interpretation: if S is global state, T is the subsystem state, the lens *extracts* the local state with get, and *updates* the local state with set

```
def lift[T, S, A](
  s: State[T, A],
  l: Lens[S, T]
): State[S, A] = State { inputS =>
    val inputT = l.get(inputS)
    val (outputT, a) = s.run(inputS).run
    val outputS  = l.set(inputS, outputT)
    (outputS, a)
})
```

EPILOG
THE TWIN PEAKS HYPOTHESIS

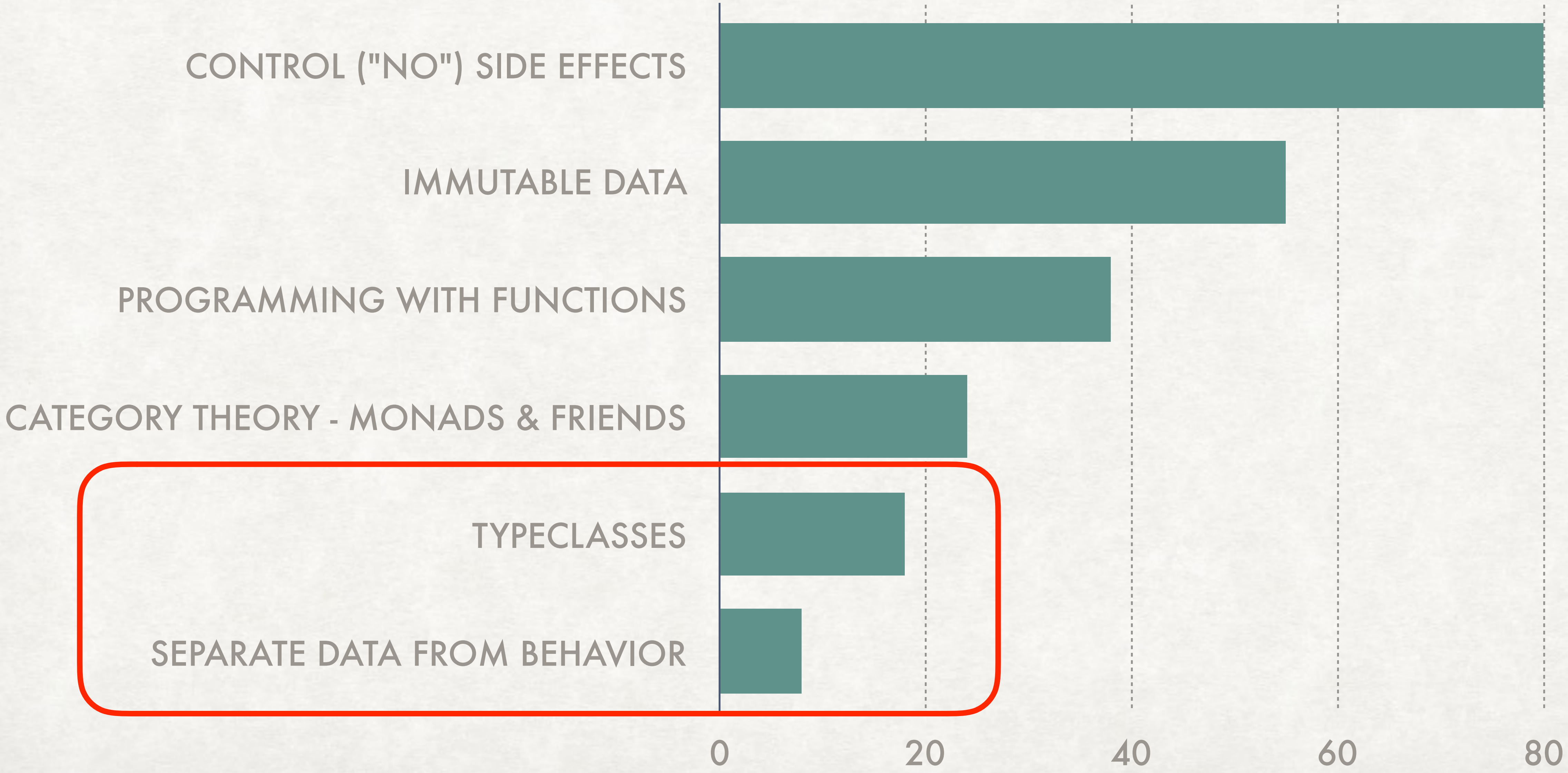
“What’s Functional Programming About?”

- Imagine if you asked random programmers on the street


CONTROL ("NO") SIDE EFFECTS
IMMUTABLE DATA
PROGRAMMING WITH FUNCTIONS
CATEGORY THEORY - MONADS & FRIENDS
TYPECLASSES
SEPARATE DATA FROM BEHAVIOR
0
20
40
60
80

“What’s Functional Programming About?”

- Imagine if you asked random programmers on the street


CONTROL ("NO") SIDE EFFECTS
IMMUTABLE DATA
PROGRAMMING WITH FUNCTIONS
CATEGORY THEORY - MONADS & FRIENDS
TYPECLASSES
SEPARATE DATA FROM BEHAVIOR
0
20
40
60
80

Separate Data And Behaviour

- I want to look at this principle
- It stands in stark contrast to object-orientation
    - “One of the fundamental principles of object-oriented design is to combine data and behavior” - Martin Fowler
  - ...and its not a *defining* characteristic of FP as a whole
    - what I’d call the *Object-Functional* school of thought combines immutability with object-orientation
      - Egs scala standard library, Typesafe stack
  - ..but its normal in typeclass-centric languages and libraries
    - Haskell, Scalaz, Cats

(Ad-Hoc) Polymorphism Is The Universal Good

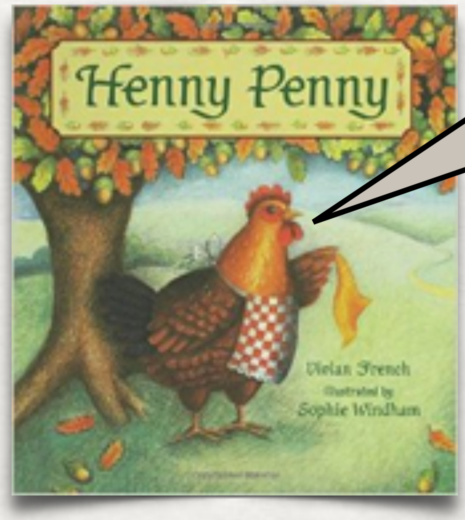
- "Ad-hoc polymorphism occurs when a function is defined over several types, acting in a different way for each type"

  How to make ad-hoc polymorphism less ad-hoc, Wadler & Blott, 1988

- *Object-orientation implements ad-hoc polymorphism by*
  - *wrapping up data into an "object", along with all methods that refer to the data*
  - *using a runtime pointer lookup to the appropriate implementation for that data*
    - eg trait Showable {def show: String}; object Foo extends Showable {def show = "Foo"}
- *Typeclasses implement adhoc polymorphism by*
  - *making functions over data parametric in the data type*
  - *defining behaviour in typeclasses that are polymorphic in the data they operate on*
    - eg def foo[A: Show](a: A): String = a.show //..further pesky details omitted!

Integrating Data+Behavior: Object-Oriented Vs Typeclass Approach

- In object-oriented systems, combining data & behaviour in a class is the Right Thing because that's how you'll achieve polymorphism over the data
  - Behaviour will change appropriately as the data is varied
- In typeclass based systems, data and behaviour can be defined in separate classes ("un-encapsulated"), and the behaviour will still change appropriately as the data is varied
  - The compiler "zips" data & behavior together during compilation using the static type of the data and the implicit search process



IF YOU DON'T ENCAPSULATE YOUR DATA THE SKY WILL FALL IN!

Is There One Scala Way To Rule Them All?

- Martin Oderksy:
  - “Scala is not opinionated; you can use it with any style you prefer.”
- Thoughtworks Tech Radar:
  - “To successfully use Scala, you need to research the language and have a very strong opinion on which parts are right for you, creating your own definition of Scala, the good parts.”
- Stack Overflow Question:
  - “I have been doing Java for a long time and started Scala about 6 months ago. I love the language. One thing that I discovered is that there are multiple ways to do things. I don't know if it is because of the nature of the language or because it's still young and evolving and idioms and best practices haven't emerged.”

The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

Trait Composition
OO-polymorphism

Mixed Mutability
Moderate typing

Object-Functional

Trait Composition
OO-polymorphism

Immutability
Strong typing

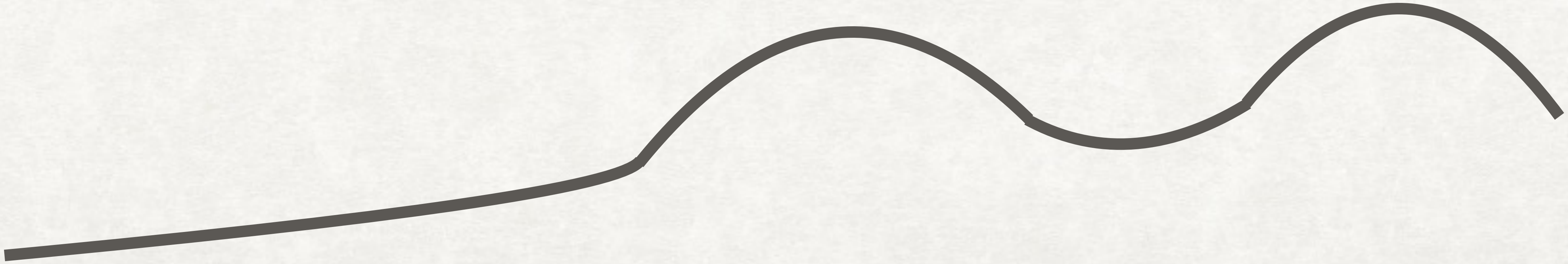
Control Effects
Interpreters

Haskell-School

Typeclasses
Separate Data

Immutability
Stronger typing

Control Effects
Interpreters



The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

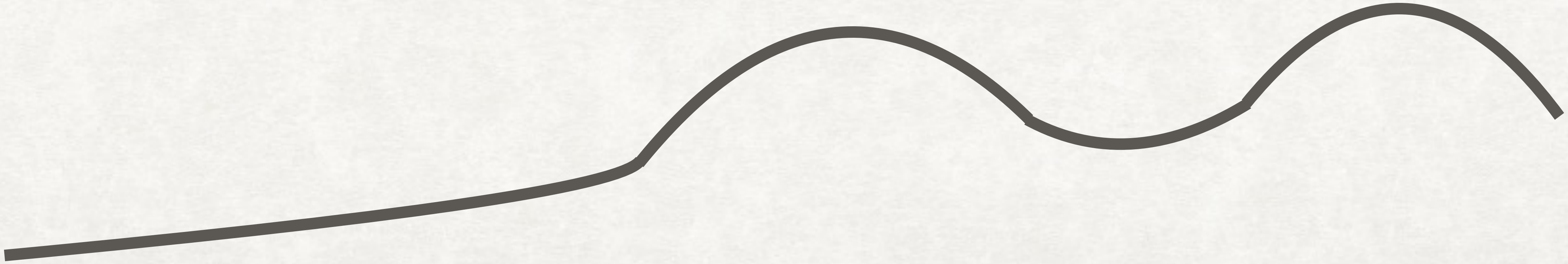
accessible on-ramp for large installed-base of OO programmers

Object-Functional

Typesafe
Odersky
Scala Std Lib
SBT

Haskell-School

Typelevel
Scalaz
Shapeless



The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

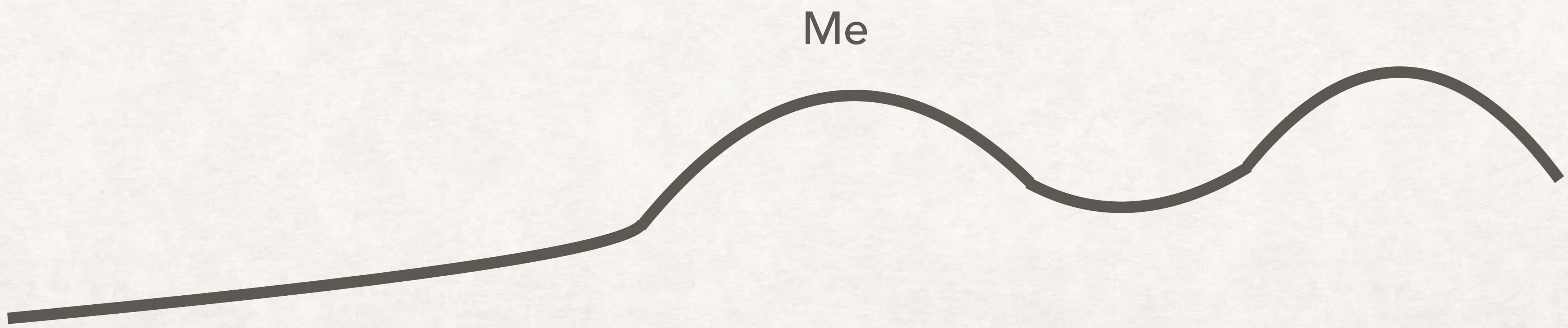
accessible on-ramp for large installed-base of OO programmers

Object-Functional

Typesafe
Odersky
Scala Std Lib
SBT

Haskell-School

Typelevel
Scalaz
Shapeless



The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

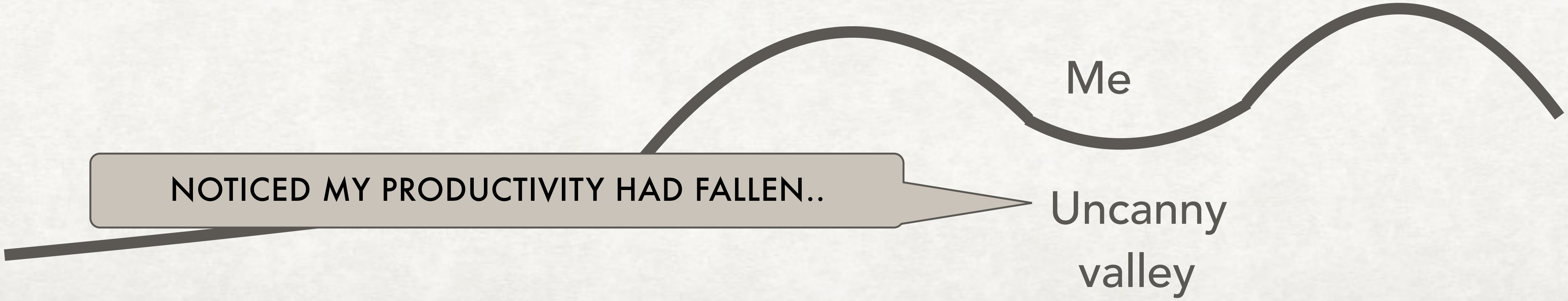
accessible on-ramp for large installed-base of OO programmers

Object-Functional

Typesafe
Odersky
Scala Std Lib
SBT

Haskell-School

Typelevel
Scalaz
Shapeless



NOTICED MY PRODUCTIVITY HAD FALLEN..

The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

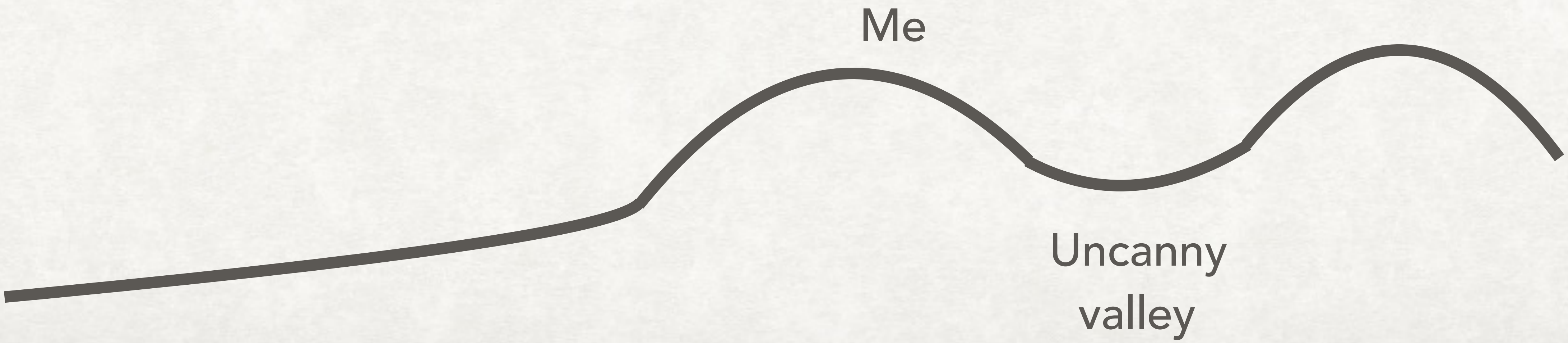
accessible on-ramp for large installed-base of OO programmers

Object-Functional

Typesafe
Odersky
Scala Std Lib
SBT

Haskell-School

Typelevel
Scalaz
Shapeless



The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons

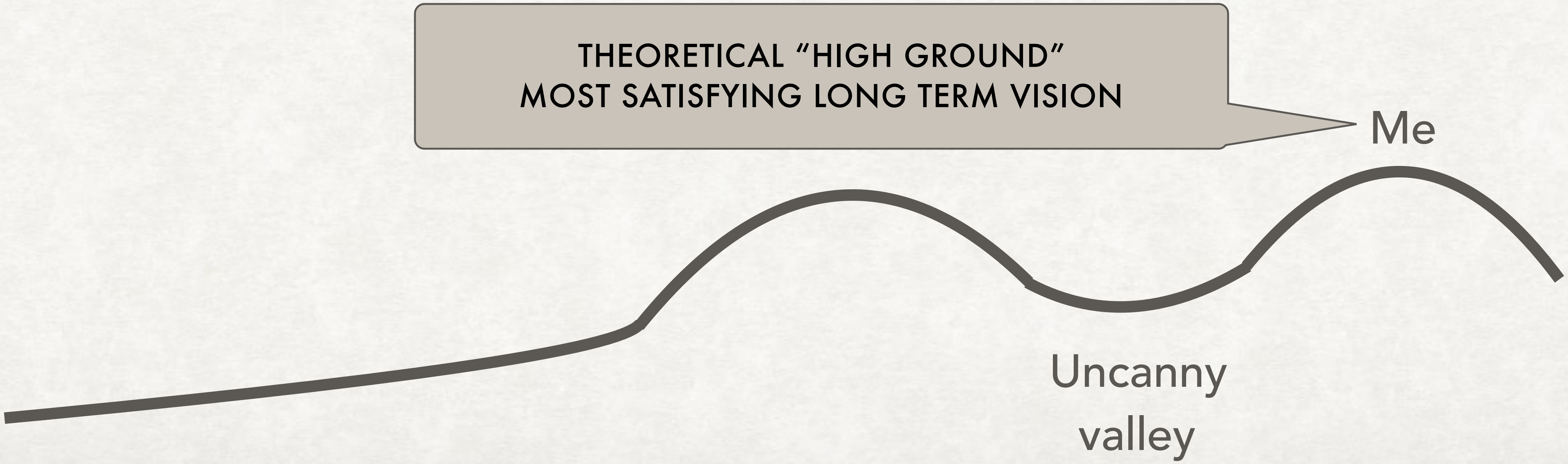
accessible on-ramp for large installed-base of OO programmers

Object-Functional

Typesafe
Odersky
Scala Std Lib
SBT

Haskell-School

Typelevel
Scalaz
Shapeless



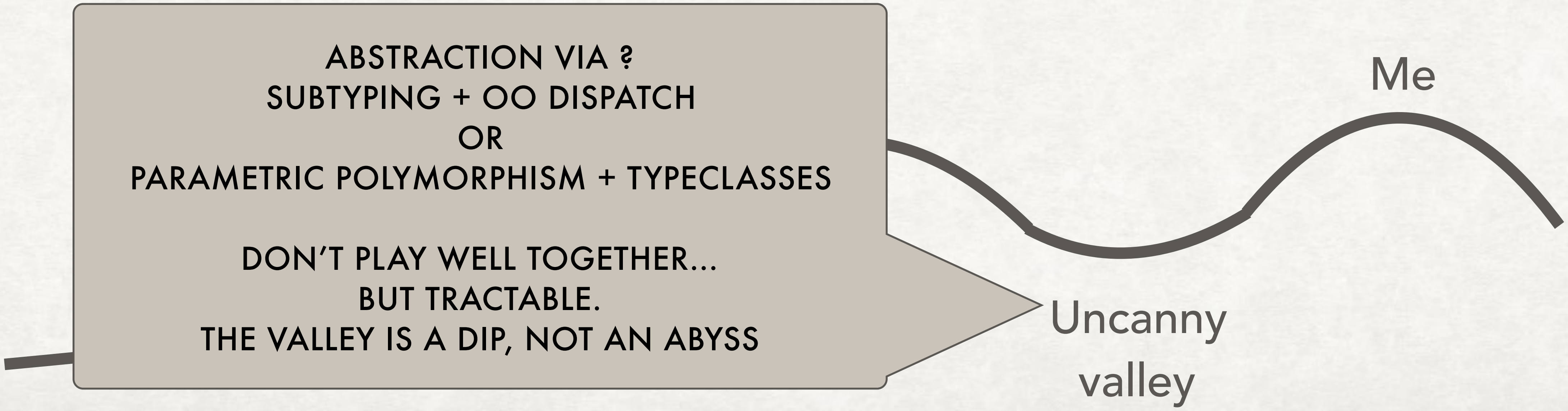
THEORETICAL "HIGH GROUND"
MOST SATISFYING LONG TERM VISION

Me

Uncanny valley

The Twin Peaks Hypothesis:
Scala Has Two Optimal Ways And A Dip Between

Java without Semicolons	Object-Functional	Haskell-School
accessible on-ramp for large installed-base of OO programmers	Typesafe Odersky Scala Std Lib SBT	Typelevel Scalaz Shapeless



ABSTRACTION VIA ?
SUBTYPING + OO DISPATCH
OR
PARAMETRIC POLYMORPHISM + TYPECLASSES

DON'T PLAY WELL TOGETHER...
BUT TRACTABLE.
THE VALLEY IS A DIP, NOT AN ABYSS

THE END

BEN HUTCHISON, FEB 2016, MELBOURNE SCALA