Why I Like Functional Programming

Adelbert Chang
Machine Learning @ Box, Inc.

Let's go back..

http://gauss.cs.ucsb.edu/home/images/UCSB-from-air.jpg

2011

2011

```
void insert(Node* &tree, int value) {
    if (!tree) return;

    Node *trav = tree, *parent;
    while (trav) {
        parent = trav;
        if (value < trav->data) trav = trav->left;
        else if (value > trav->data) trav = trav->right;
        else break;
    }
    if (value < parent->data)
        parent->left = new Node(value);
    else
        parent->right = new Node(value);
}
```

Physics and Math

Physics and Math

- Simplicity and consistency across problems

Physics and Math

- Simplicity and consistency across problems
- Example: Deriving laws of motion from first principles

Physics and Math

- Simplicity and consistency across problems
- Example: Deriving laws of motion from first principles
  - $v_f = v_0 + a \Delta t$

Physics and Math

- Simplicity and consistency across problems
- Example: Deriving laws of motion from first principles
  - $v_f = v_0 + a\,\Delta t$
  - $v_{avg} = v_0 + \frac{1}{2}a\Delta t$

Physics and Math

- Simplicity and consistency across problems
- Example: Deriving laws of motion from first principles
  - $v_f = v_0 + a\,\Delta t$
  - $v_{avg} = v_0 + \frac{1}{2}a\Delta t$
  - $x = x_0 + v_0 t + \frac{1}{2}at^2$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$v_{avg} = \Delta x/\Delta t$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$v_{avg} = \Delta x/\Delta t$

$\Delta x = v_{avg} * \Delta t$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$$v_{avg} = \Delta x/\Delta t$$

$$\Delta x = v_{avg} * \Delta t$$

$$(x - x_0) = v_{avg} * \Delta t$$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$$v_{avg} = \Delta x / \Delta t$$

$$\Delta x = v_{avg} * \Delta t$$

$$(x - x_0) = v_{avg} * \Delta t$$

$$x = x_0 + (v_{avg} * \Delta t)$$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$v_{avg} = \Delta x / \Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$a = \Delta v / \Delta t$

$v_{avg} = \Delta x / \Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$a = \Delta v / \Delta t$

$a = (v_f - v_0) / \Delta t$

$v_{avg} = \Delta x / \Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$a = \Delta v / \Delta t$

$a = (v_f - v_0) / \Delta t$

$v_f - v_0 = a \Delta t$

$v_{avg} = \Delta x / \Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$$v_{avg} = \frac{1}{2}(v_f + v_0)$$

$$a = \Delta v \,/\, \Delta t$$

$$a = (v_f - v_0) \,/\, \Delta t$$

$$v_f - v_0 = a\,\Delta t$$

$$v_f = v_0 + a\,\Delta t$$

$$v_{avg} = \Delta x / \Delta t$$

$$\Delta x = v_{avg} * \Delta t$$

$$(x - x_0) = v_{avg} * \Delta t$$

$$x = x_0 + (v_{avg} * \Delta t)$$

Physics and Math

Deriving $x = x_0 + v_0t + ½at^2$

$v_{avg} = ½(v_f + v_0)$

$½(v_0 + a\Delta t + v_0)$

$a = \Delta v / \Delta t$

$a = (v_f - v_0) / \Delta t$

$v_f - v_0 = a \Delta t$

$v_f = v_0 + a \Delta t$

$v_{avg} = \Delta x/\Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0 t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$\frac{1}{2}(v_0 + a\Delta t + v_0)$

$\frac{1}{2}(2v_0 + a\,\Delta t)$

$a = \Delta v \,/\, \Delta t$

$a = (v_f - v_0) \,/\, \Delta t$

$v_f - v_0 = a\,\Delta t$

$v_f = v_0 + a\,\Delta t$

$v_{avg} = \Delta x/\Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0t + ½at^2$

$v_{avg} = ½(v_f + v_0)$

$½(v_0 + a\Delta t + v_0)$

$½(2v_0 + a\,\Delta t)$

$v_{avg} = v_0 + ½a\Delta t$

$a = \Delta v \,/\, \Delta t$

$a = (v_f - v_0) \,/\, \Delta t$

$v_f - v_0 = a\,\Delta t$

$v_f = v_0 + a\,\Delta t$

$v_{avg} = \Delta x/\Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$$v_{avg} = \tfrac{1}{2}(v_f + v_0)$$

$$\tfrac{1}{2}(v_0 + a\Delta t + v_0)$$

$$\tfrac{1}{2}(2v_0 + a\,\Delta t)$$

$$v_{avg} = v_0 + \tfrac{1}{2}a\Delta t$$

$$a = \Delta v \,/\, \Delta t$$

$$a = (v_f - v_0) \,/\, \Delta t$$

$$v_f - v_0 = a\,\Delta t$$

$$v_f = v_0 + a\,\Delta t$$

$$v_{avg} = \Delta x/\Delta t$$

$$\Delta x = v_{avg} * \Delta t$$

$$(x - x_0) = v_{avg} * \Delta t$$

$$x = x_0 + (v_{avg} * \Delta t)$$

$$x = x_0 + ((v_0 + \tfrac{1}{2}a\Delta t) * \Delta t)$$

Physics and Math

Deriving $x = x_0 + v_0t + \frac{1}{2}at^2$

$v_{avg} = \frac{1}{2}(v_f + v_0)$

$\frac{1}{2}(v_0 + a\Delta t + v_0)$

$\frac{1}{2}(2v_0 + a\,\Delta t)$

$v_{avg} = v_0 + \frac{1}{2}a\Delta t$

$a = \Delta v \,/\, \Delta t$

$a = (v_f - v_0) \,/\, \Delta t$

$v_f - v_0 = a\,\Delta t$

$v_f = v_0 + a\,\Delta t$

$v_{avg} = \Delta x/\Delta t$

$\Delta x = v_{avg} * \Delta t$

$(x - x_0) = v_{avg} * \Delta t$

$x = x_0 + (v_{avg} * \Delta t)$

$x = x_0 + ((v_0 + \frac{1}{2}a\Delta t) * \Delta t)$

$x_0 + v_0t + \frac{1}{2}at^2$

2012

2012

```
sealed abstract class Tree
case class Branch(data: Int, left: Tree, right: Tree)
  extends Tree
case class Leaf() extends Tree

def insert(tree: Tree, value: Int): Tree =
  tree match {
    case Leaf() => Branch(value, Leaf(), Leaf())
    case b@Branch(d, l, r) =>
      if (value < d) Branch(d, insert(l, value), r)
      else if (value > d) Branch(d, l, insert(r, value))
      else b
  }
```

Hmm.. functional programming seems pretty cool.

Hmm.. functional programming seems pretty cool.

But is it as cool as physics and math?

Functional Programming

Functional Programming

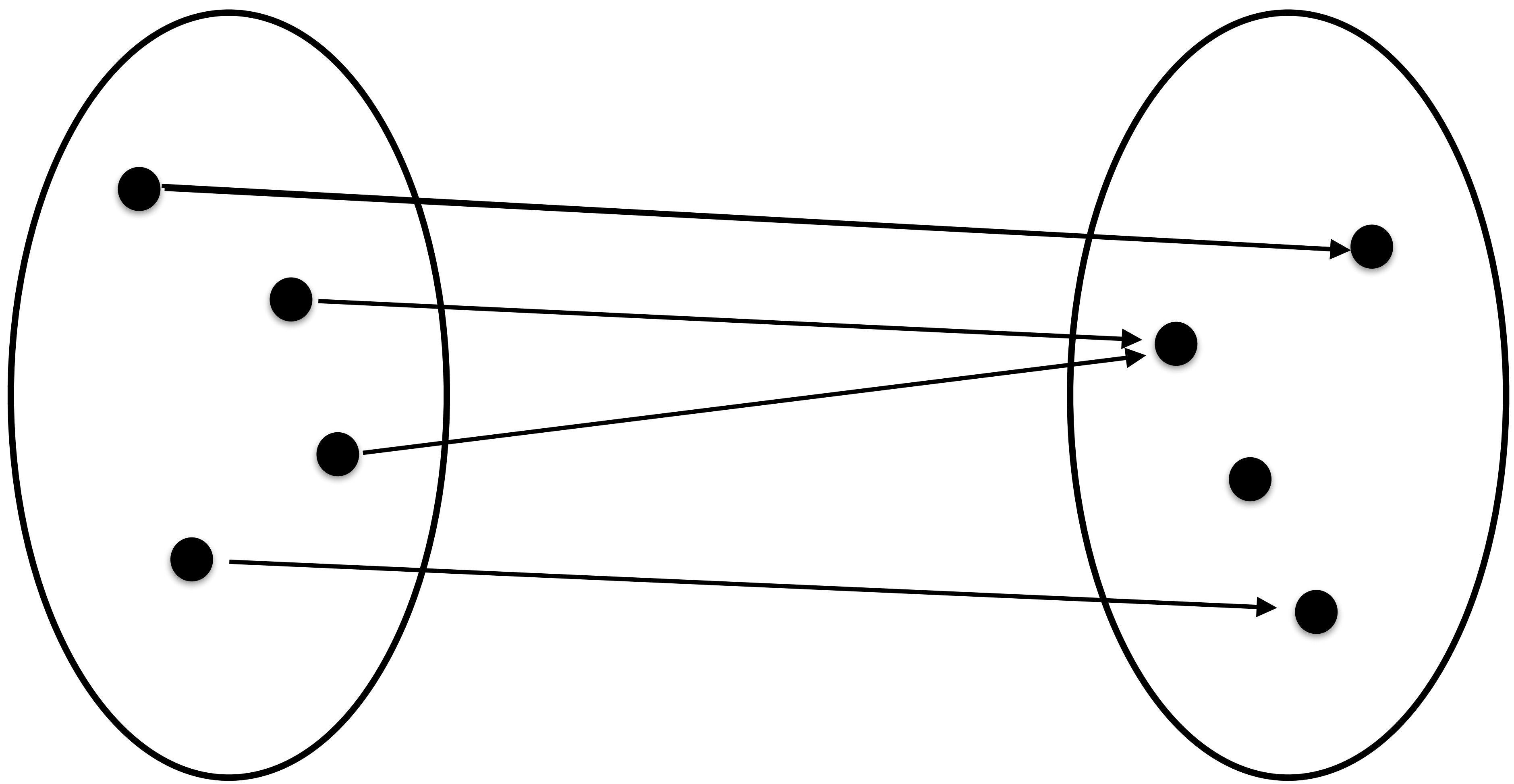
programming with functions

Functional Programming

programming with **pure** functions

Functional Programming

programming with **pure** functions



Functions

Functions

```scala
def parseIntPartial(s: String): Int =
  s.toInt // can throw NumberFormatException
```

Functions

```
def parseIntPartial(s: String): Int =
  s.toInt // can throw NumberFormatException


def parseIntTotal(s: String): Option[Int] =
  try {
    Some(s.toInt)
  } catch {
    case nfe: NumberFormatException => None
  }
```

Functions

Functions

```
class Rng(var seed: Long) {
  def nextInt(): Int = {
    val int = getInt(seed)
    mutate(seed)
    int
  }
}
```

Functions

```
class Rng(var seed: Long) {
  def nextInt(): Int = {
    val int = getInt(seed)
    mutate(seed)
    int
  }
}

case class Rng(seed: Long) {
  def nextInt: (Rng, Int) = {
    val int = getInt(seed)
    val newSeed = f(seed)
    (Rng(newSeed), int)
  }
}
```

Referential Transparency

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

An expression **e** is referentially transparent if

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

An expression **e** is referentially transparent if
for all programs **p**

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

An expression **e** is referentially transparent if
for all programs **p**
every occurrence of **e** in **p**

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

An expression **e** is referentially transparent if
for all programs **p**
every occurrence of **e** in **p**
can be replaced with the result of evaluating **e**

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

An expression **e** is referentially transparent if
for all programs **p**
every occurrence of **e** in **p**
can be replaced with the result of evaluating **e**
without changing the result of evaluating **p**.*

* Taken from Higher Order blog post "What Purity Is and Isn't" May 29, 2015 http://blog.higher-order.com/blog/2012/09/13/what-purity-is-and-isnt/

Referential Transparency

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

Referential Transparency

$$x^4 - 12x^2 + 36 = 0$$

$$y = x^2$$

$$y^2 - 12y + 36 = 0$$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$ax^2 + bx + c = 0$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1$ $b = -12$ $c = 36$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1 \qquad b = -12 \qquad c = 36$

$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1$ $b = -12$ $c = 36$

$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

$12 / 2$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$y = 6$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1$ $b = -12$ $c = 36$

$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

$12 / 2$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$y = 6$

$x^2 = 6$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1$ $b = -12$ $c = 36$

$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

$12 / 2$

Referential Transparency

$x^4 - 12x^2 + 36 = 0$

$y = x^2$

$y^2 - 12y + 36 = 0$

$y = 6$

$x^2 = 6$

$x = \pm \sqrt{6}$

$ax^2 + bx + c = 0$

$(-b \pm \sqrt{b^2 - 4ac}) / 2a$

$a = 1 \qquad b = -12 \qquad c = 36$

$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

$12 / 2$

Referential Transparency

Referential Transparency

```
def userInfo(id: UserId): Future[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => fetchData
}
```

Referential Transparency

```
def userInfo(id: UserId): Future[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => fetchData
}
```

Referential Transparency

```
def userInfo(id: UserId): Future[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => userInfo(userId)
}
```

Referential Transparency

```
def userInfo(id: UserId): Future[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => userInfo(userId)
}
```

???

Referential Transparency

```
def userInfo(id: UserId): Task[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => fetchData
}
```

Referential Transparency

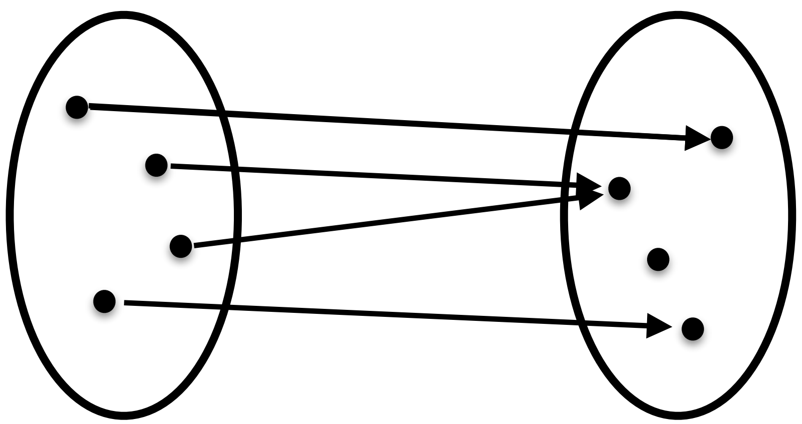
```
def userInfo(id: UserId): Task[UserData]

val userId = . . .

val fetchData = userInfo(userId)

fetchData.retry {
  case StatusCode(429) => fetchData
}
```

```
trait Task[A] {
  def unsafeRun(): A
}
```

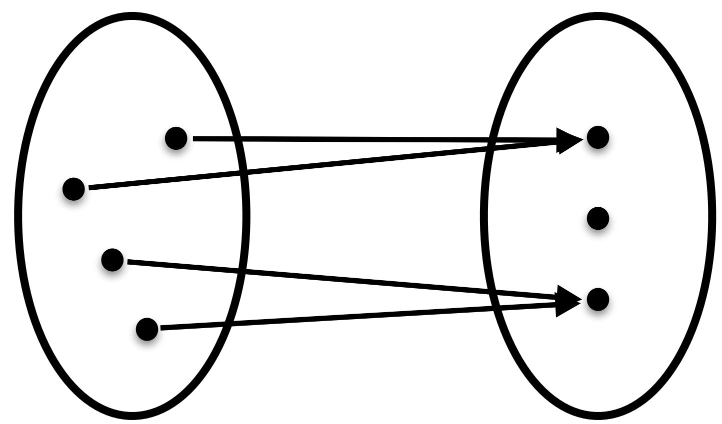
Functions

Functions

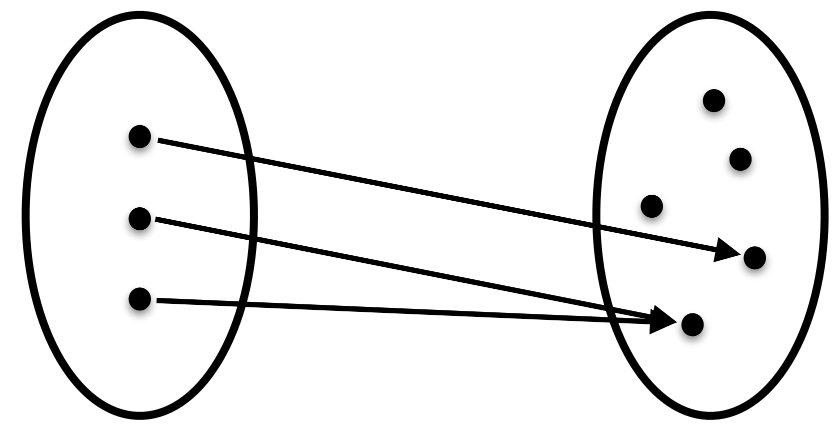
foo: A => B



bar: B => C

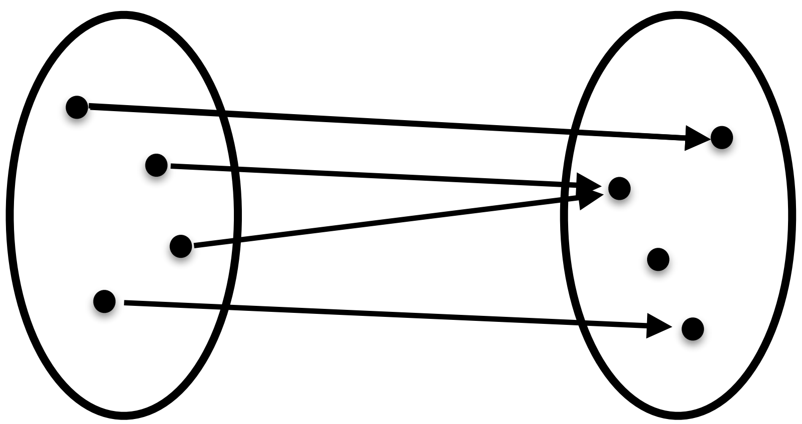


baz: C => D

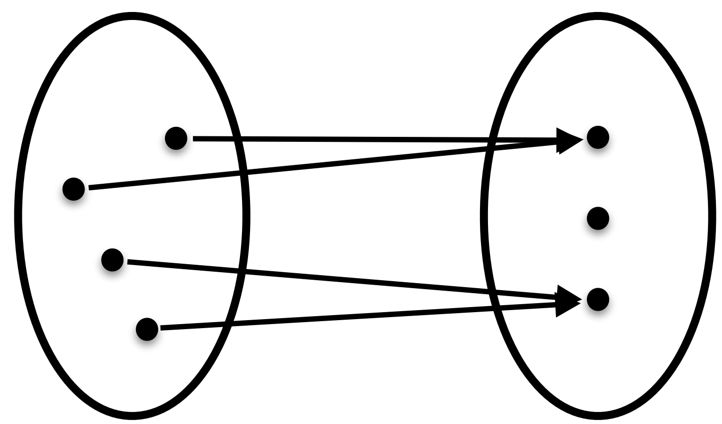


Functions

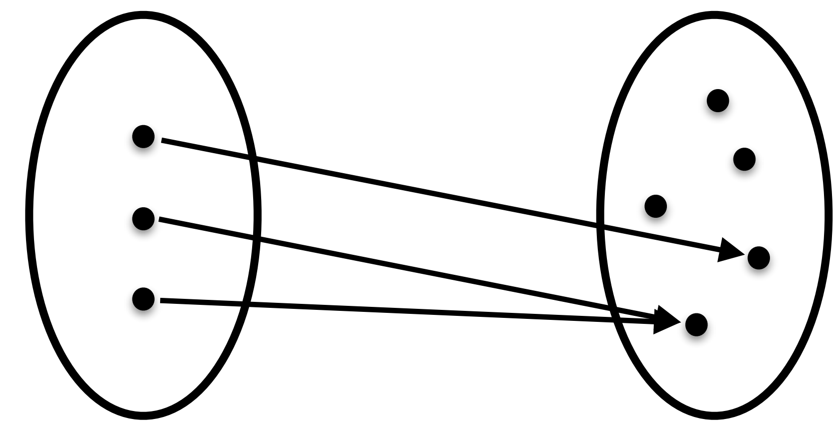
foo: A => B



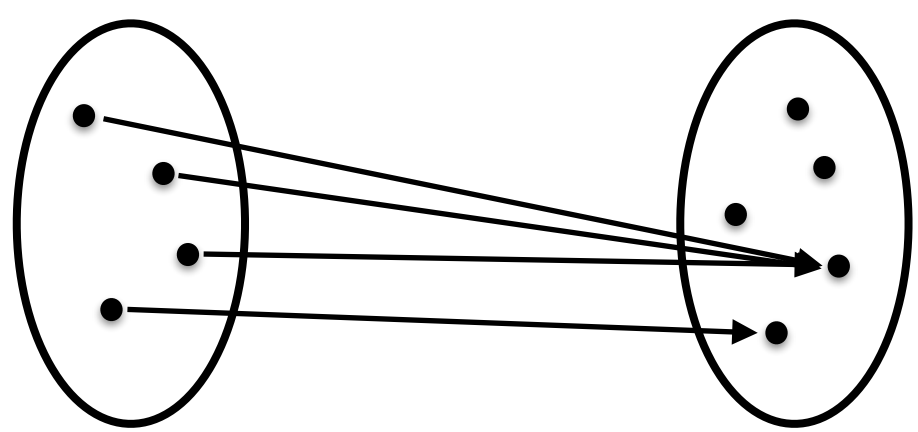
bar: B => C



baz: C => D



baz.compose(bar).compose(foo): A => D



Tracking Effects

Tracking Effects

- Interesting programs will have effects
  - Optional values, error handling, asynchronous computation, input/output
  - Usual means aren’t quite nice

Tracking Effects

- Interesting programs will have effects
  - Optional values, error handling, asynchronous computation, input/output
  - Usual means aren’t quite nice

Solution:

Tracking Effects

- Interesting programs will have effects
  - Optional values, error handling, asynchronous computation, input/output
  - Usual means aren't quite nice

Solution:

Reify effects as values.

Missing values

Missing values

```scala
// A value that might exist is either..
sealed abstract class Option[A]

// ..there
final case class Some[A](a: A) extends Option[A]

// .. or not there
final case class None[A]() extends Option[A]
```

Missing values

```
def lookup(map: Map[Foo, Bar], foo: Foo): Option[Bar] =
  if (map.contains(foo)) Some(map(foo))
  else None
```

Errors

```scala
sealed abstract class Either[+E, +A]

final case class Success[+A](a: A) extends Either[Nothing, A]

final case class Failure[+E](e: E) extends Either[E, Nothing]
```

Errors

```
sealed abstract class Error
final case object UserDoesNotExist extends Error
final case object InvalidToken     extends Error

def userInfo(uid: UserId,
             tk: Token): Either[Error, UserInfo] = . . .
```

Manipulating Effects

Manipulating Effects

- The type signature of our functions reflect the effects involved

Manipulating Effects

- The type signature of our functions reflect the effects involved

```scala
def tokenFor(uid: UserId): Option[EncryptedToken]

def decrypt(token: EncryptedToken): Token

/** Client side */
val encrypted: Option[EncryptedToken] =
  tokenFor(. . .)

// Want EncryptedToken, have Option[EncryptedToken]
val decrypted = decrypt(???)
```

Manipulating Effects

```scala
/** Apply a pure function to an effectful value */
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

Manipulating Effects

```scala
/** Apply a pure function to an effectful value */
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}

new Functor[Option] {
  def map[A, B](fa: Option[A])(f: A => B): Option[B] =
    fa match {
      case Some(a) => Some(f(a))
      case None()  => None()
    }
}
```

Manipulating Effects

```scala
def tokenFor(uid: UserId): Option[EncryptedToken]

def decrypt(token: EncryptedToken): Token

/** Client side */
val encrypted: Option[EncryptedToken] =
  tokenFor(. . .)

val decrypted = encrypted.map(tk => decrypt(tk))
```

Manipulating Effects

```
def tokenFor(uid: UserId): Option[EncryptedToken]

def decrypt(token: EncryptedToken): Token

/** Client side */
val encrypted: Option[EncryptedToken] =
  tokenFor(. . .)

val decrypted = encrypted.map(tk => decrypt(tk))
```

- Similar mechanisms for manipulating multiple effects

Lens

Lens

- Often want getters/setters when working with data

Lens

- Often want getters/setters when working with data
- If getters/setters are per-object, cannot compose

Lens

- Often want getters/setters when working with data
- If getters/setters are per-object, cannot compose
- As with effects, reify as data
  - Getter: get an **A** field in an object **S**
  - Setter: change an **A** field in an object **S**

Lens

- Often want getters/setters when working with data
- If getters/setters are per-object, cannot compose
- As with effects, reify as data
  - Getter: get an **A** field in an object **S**
  - Setter: change an **A** field in an object **S**

```
trait Lens[S, A] {
  def get(s: S): A

  def set(s: S, a: A): S
}
```

Lens

```scala
trait Lens[S, A] { outer =>
  def get(s: S): A

  def set(s: S, a: A): S

  def modify(s: S, f: A => A): S =
    set(s, f(get(s)))

  def compose[B](other: Lens[A, B]): Lens[S, B] =
    new Lens[S, B] {
      def get(s: S): B = other.get(get(s))

      def set(s: S, b: B): S =
        set(s, other.set(outer.get(s), b))
    }
}
```

Lens

```scala
case class Employee(position: Position)

object Employee {
  val position: Lens[Employee, Position] = . . .
}

case class Team(manager: Employee, . . .)

object Team {
  val manager: Lens[Team, Employee] = . . .
}
```

Lens

```scala
case class Employee(position: Position)

object Employee {
  val position: Lens[Employee, Position] = . . .
}

case class Team(manager: Employee, . . .)

object Team {
  val manager: Lens[Team, Employee] = . . .
}

/** Client side */
val l: Lens[Team, Position] =
  Team.manager.compose(Employee.position)
l.set(someTeam, somePosition)
```

Lens

Lens

- Effect-ful modifications can be useful
  - Possibly failing: A => Option[A]
  - Many possible values: A => List[A]
  - Async: A => Task[A]

Lens

- Effect-ful modifications can be useful
  - Possibly failing: A => Option[A]
  - Many possible values: A => List[A]
  - Async: A => Task[A]

```
trait Lens[S, A] {
  def modifyOption(s: S, f: A => Option[A]): Option[S]

  def modifyList(s: S, f: A => List[A]): List[S]

  def modifyTask(s: S, f: A => Task[A]): Task[S]
}
```

Lens

```
trait Lens[S, A] {
  def modifyOption(s: S, f: A => Option[A]): Option[S] =
    f(get(s)).map(a => set(s, a))

  def modifyList(s: S, f: A => List[A]): List[S] =
    f(get(s)).map(a => set(s, a))

  def modifyTask(s: S, f: A => Task[A]): Task[S] =
    f(get(s)).map(a => set(s, a))
}
```

Lens

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

Lens

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:

Lens

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:
  - Option, Either, Future, List, IO

Lens

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:
  - Option, Either, Future, List, IO
- What if we want to just modify without any effect?

Lens

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:
  - Option, Either, Future, List, IO
- What if we want to just modify without any effect?
  - We want the F[A] to just be a plain A

Lens

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:
  - Option, Either, Future, List, IO
- What if we want to just modify without any effect?
  - We want the F[A] to just be a plain A
    - Type-level identity function

Lens

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

- Example functors:
  - Option, Either, Future, List, IO
- What if we want to just modify without any effect?
  - We want the F[A] to just be a plain A
    - Type-level identity function

```scala
def identity[A](a: A): A = a
```

Id

```
type Id[A] = A
```

Id

```scala
type Id[A] = A

new Functor[Id] {
  def map[A, B](fa: Id[A])(f: A => B): Id[B] =
}
```

Id

```
type Id[A] = A

new Functor[Id] {
  def map[A, B](fa: Id[A])(f: A => B): Id[B] =
}
```

Id

```
type Id[A] = A

new Functor[Id] {
  def map[A, B](fa: A)(f: A => B): B =
}
```

Id

```scala
type Id[A] = A

new Functor[Id] {
  def map[A, B](fa: A)(f: A => B): B = f(fa)
}
```

Id

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))

  def modify(s: S, f: A => A): S = modifyF[Id](s, f)
}
```

Setting

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))

  def modify(s: S, f: A => A): S = modifyF[Id](s, f)

  def set(s: S, a: A): S = modify(s, const(a))
}
```

Setting

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))

  def modify(s: S, f: A => A): S = modifyF[Id](s, f)

  def set(s: S, a: A): S = modify(s, const(a))
}
```

```scala
def const[A, B](a: A)(b: B): A = a
```

Setting

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))

  def modify(s: S, f: A => A): S = modifyF[Id](s, f)

  def set(s: S, a: A): S = modify(s, const(a))
}
```

```
def const[A, B](a: A)(b: B): A = a
```

Setting

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A

  def modify(s: S, f: A => A): S = modifyF[Id](s, f)

  def set(s: S, a: A): S = modify(s, const(a))
}
```

```scala
def const[A, B](a: A)(b: B): A = a
```

Getting

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

Getting

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

- Can we define `get` in terms of `modify`?

Getting

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

- Can we define `get` in terms of `modify`?
    - `modifyF` gives us some F[S].. but we want an A

Getting

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

- Can we define `get` in terms of `modify`?
  - `modifyF` gives us some F[S].. but we want an A
  - We need some way of "ignoring" the S parameter and still get an A back

Getting

```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

- Can we define `get` in terms of `modify`?
  - `modifyF` gives us some F[S].. but we want an A
  - We need some way of "ignoring" the S parameter and still get an A back
    - Type-level constant function

Getting

```scala
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  def get(s: S): A
}
```

- Can we define `get` in terms of `modify`?
  - `modifyF` gives us some F[S].. but we want an A
  - We need some way of "ignoring" the S parameter and still get an A back
    - Type-level constant function

```scala
def const[A, B](a: A)(b: B): A = a
```

Getting

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```
type Const[Z, A] = Z
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```scala
type Const[Z, A] = Z

new Functor[Const[Z, ?]] { // *
  def map[A, B](fa: Const[Z, A])(f: A => B): Const[Z, B]
}
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```
type Const[Z, A] = Z

new Functor[Const[Z, ?]] { // *
  def map[A, B](fa: Const[Z, A])(f: A => B): Const[Z, B]
}
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```scala
type Const[Z, A] = Z

new Functor[Const[Z, ?]] { // *
  def map[A, B](fa: Z)(f: A => B): Z =
}
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```
type Const[Z, A] = Z

new Functor[Const[Z, ?]] { // *
  def map[A, B](fa: Z)(f: A => B): Z = fa
}
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Getting

```scala
type Const[Z, A] = Z

new Functor[Const[Z, ?]] { // *
  def map[A, B](fa: Z)(f: A => B): Z = fa
}

trait Lens[S, Z] {
  def modifyF[F[_] : Functor](s: S, f: Z => F[Z]): F[S]

  def get(s: S): Z = {
    val const: Const[Z, S] =
      modifyF[Const[Z, ?]](s, z => z) // *
    const
  }
}
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Lens

```scala
trait Lens[S, A] {
  /** Abstract */
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  /** Implemented */
  def modify(s: S, f: A => A): S

  def get(s: S): A

  def set(s: S, a: A): S

  def compose[B](other: Lens[A, B]): Lens[S, B]
}
```

Are Id and Const just party tricks?

Traverse

```
trait Traverse[F[_]] {
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}
```

Traverse

```
trait Traverse[F[_]] {
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}


def validate[A]
    (data: List[A])(f: A => Option[B]): Option[List[B]]
```

Traverse

```scala
trait Traverse[F[_]] {
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}

def validate[A]
    (data: List[A])(f: A => Option[B]): Option[List[B]]

def scatterGather[A]
    (data: List[A])(f: A => Task[B]): Task[List[B]]
```

Traverse

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]
```

```
traverse[Id, A, B]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]
```

```
traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]


traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
    F[A] => (A => B)     => F[B]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]


traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
    F[A] => (A => B)     => F[B]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]



traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
    F[A] => (A => B)     => F[B]

traverse[Const[Z, ?], A, B] // *, If Z can be "reduced"
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]



traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
    F[A] => (A => B)     => F[B]

traverse[Const[Z, ?], A, B] // *, If Z can be "reduced"
    F[A] => (A => Const[Z, B]) => Const[Z, F[B]]
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Traverse

```
traverse: F[A] => (A => G[B]) => G[F[B]]



traverse[Id, A, B]
    F[A] => (A => Id[B]) => Id[F[B]]
    F[A] => (A => B)     => F[B]

traverse[Const[Z, ?], A, B] // *, If Z can be "reduced"
    F[A] => (A => Const[Z, B]) => Const[Z, F[B]]
    F[A] => (A => Z)           => Z
```

* Const[Z, ?] syntax is valid due to kind-projector plugin https://github.com/non/kind-projector

Interested?

Interested?

- Functional Programming
    - Cats, Scalaz
    - Argonaut, Atto, Monocle, Shapeless

Interested?

- Functional Programming
    - Cats, Scalaz
    - Argonaut, Atto, Monocle, Shapeless
- Algebra
    - Algebird, Algebra, Breeze, Spire

Interested?

- Functional Programming
  - Cats, Scalaz
  - Argonaut, Atto, Monocle, Shapeless
- Algebra
  - Algebird, Algebra, Breeze, Spire
- Big Data
  - Scalding, Scoobi, Spark

Interested?

- Functional Programming
  - Cats, Scalaz
  - Argonaut, Atto, Monocle, Shapeless
- Algebra
  - Algebird, Algebra, Breeze, Spire
- Big Data
  - Scalding, Scoobi, Spark
- Systems
  - Doobie, HTTP4S, Remotely, Scalaz-stream

EOF