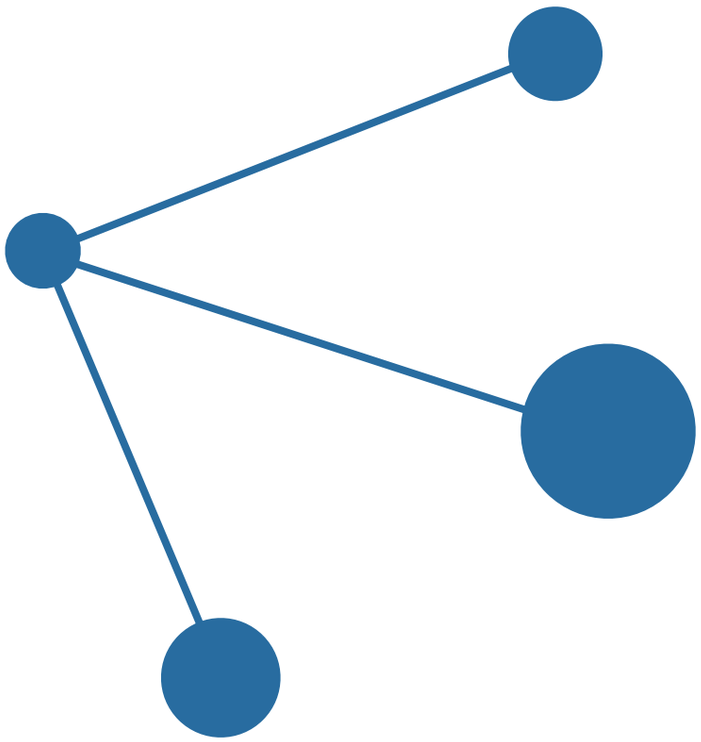


Why I Like FP

Adelbert Chang
Box, Inc.

Let's go back..

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

Timeline

Timeline

2010
Entered UCSB
as CS major.

Timeline

Took first two college physics courses. Considered changing to physics.

2011

2010

Entered UCSB as CS major.

Timeline

2011

Took first two college physics courses. Considered changing to physics.

2010

Entered UCSB as CS major.

2012

Took upper-division math courses. Considered minoring in mathematics. Started learning Scala.

Timeline

2010
Entered UCSB as CS major.

2011
Took first two college physics courses. Considered changing to physics.

2012
Took upper-division math courses. Considered minoring in mathematics. Started learning Scala.

2013
Started learning Haskell. Took graduate static analysis course.

Timeline

2010
Entered UCSB as CS major.

2011
Took first two college physics courses. Considered changing to physics.

2012
Took upper-division math courses. Considered minoring in mathematics. Started learning Scala.

2013
Started learning Haskell. Took graduate static analysis course.

2014
Graduated UCSB as a CS major. Working at Box in a functional codebase.

Timeline

2010
Entered UCSB as CS major.

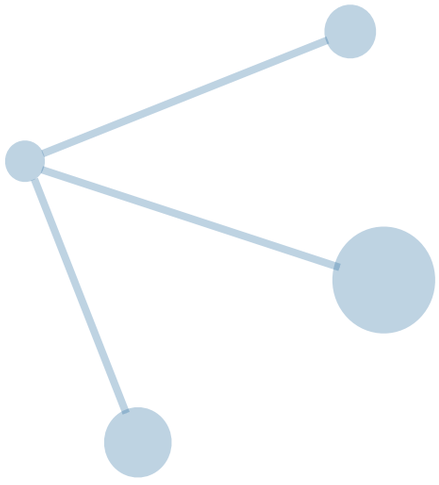
2011
Took first two college physics courses. Considered changing to physics.

2012
Took upper-division math courses. Considered minoring in mathematics. Started learning Scala.

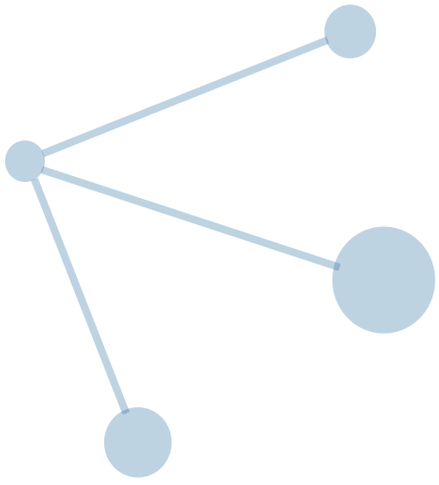
2013
Started learning Haskell. Took graduate static analysis course.

2014
Graduated UCSB as a CS major. Working at Box in a functional codebase.

2015
Presenting at an awesome FP conference.

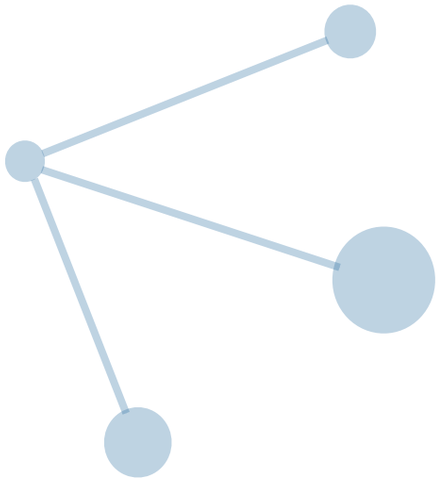


Why CS



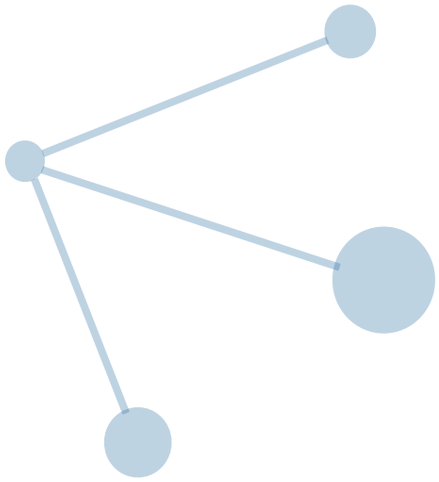
Why CS

- Computers are pretty cool



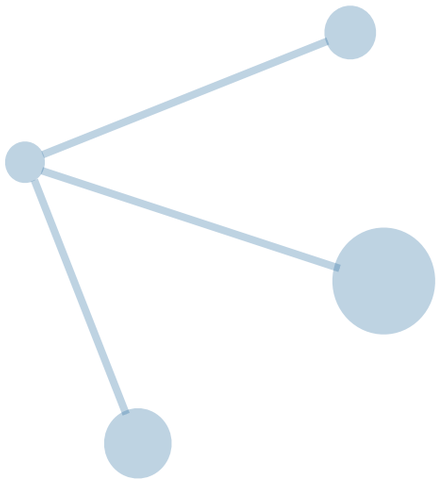
Why CS

- Computers are pretty cool
- Applied problems to solve



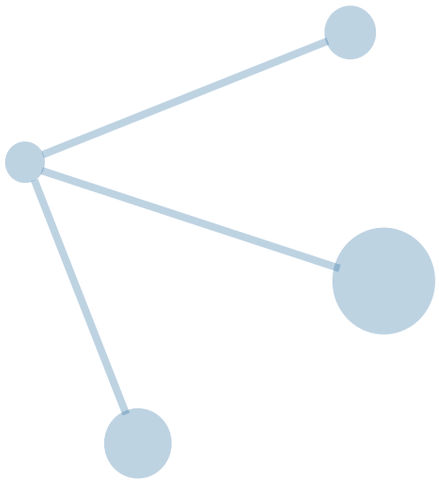
Why CS

- Computers are pretty cool
- Applied problems to solve
- Fairly interactive



Why CS

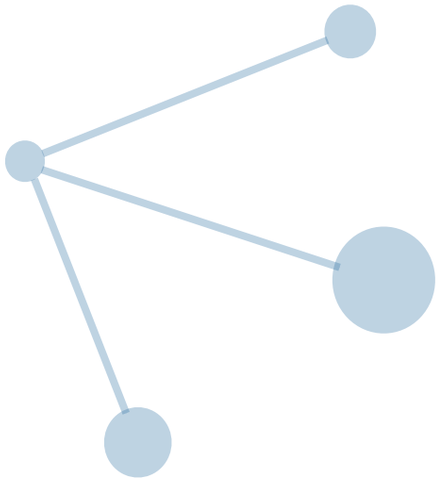
- Computers are pretty cool
- Applied problems to solve
- Fairly interactive
- Rooted in logic



Why CS

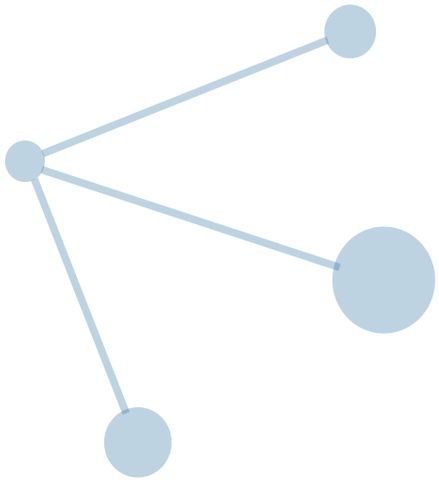
- Computers are pretty cool
- Applied problems to solve
- Fairly interactive
- Rooted in logic
- Can be done pretty much anywhere

2011



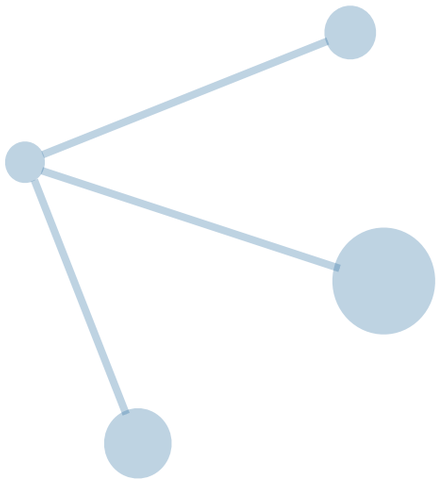
2011

- Languages learned/used: Python, C, C++



2011

- Languages learned/used: Python, C, C++
- Took first two college physics courses



2011

- Languages learned/used: Python, C, C++
- Took first two college physics courses
    - Considered switching major to physics

2011

- Languages learned/used: Python, C, C++
- Took first two college physics courses
    - Considered switching major to physics

```
swapped = True
while swapped:
    swapped = False
    for i in range(len(xs) - 1):
        if xs[i] > xs[i + 1]:
            xs[i], xs[i + 1] = xs[i + 1], xs[i]
            swapped = True
```

Physics and Math

Physics and Math

- Problem solving activity

Physics and Math

- Problem solving activity
- Can be done pretty much anywhere

Physics and Math

- Problem solving activity
- Can be done pretty much anywhere
- Simplicity and consistency across problems

Physics and Math

- Problem solving activity
- Can be done pretty much anywhere
- Simplicity and consistency across problems
- Example: Deriving law of motion from first principles

Physics and Math

- Problem solving activity
- Can be done pretty much anywhere
- Simplicity and consistency across problems
- Example: Deriving law of motion from first principles
  - $x = x_0 + v_0t + \frac{1}{2}at^2$

Physics and Math

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

$a = \Delta v / \Delta t$

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Physics and Math

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

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

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

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

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

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

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

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

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

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

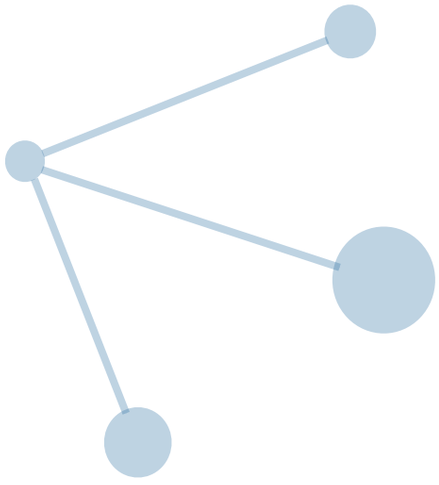
$$\Delta x = v_{avg} * \Delta t$$

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

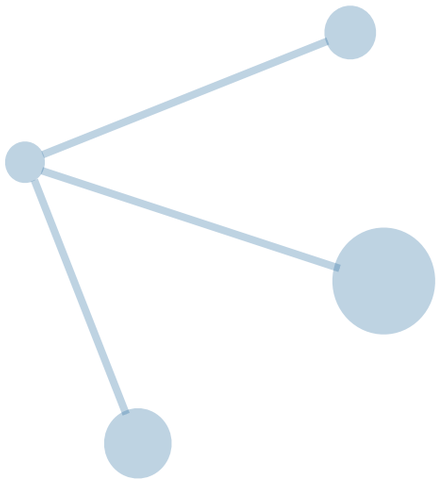
$$x = x_0 + (v_{avg} * \Delta t)$$

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

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

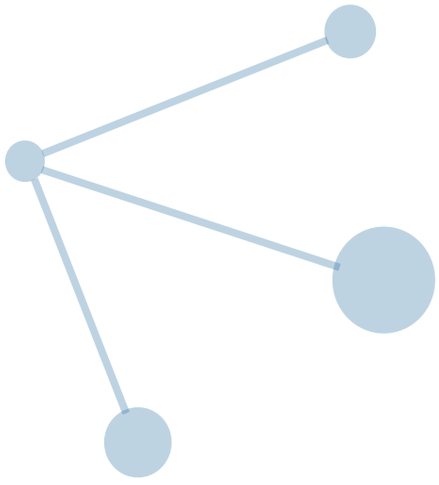


2012

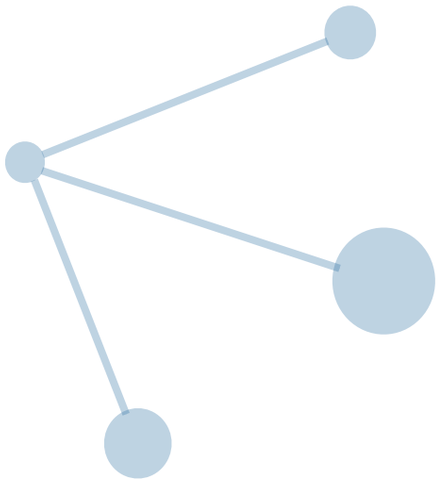


2012

```
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);
}
```



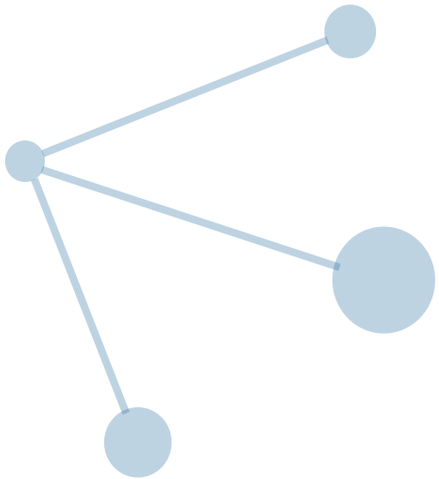
2012



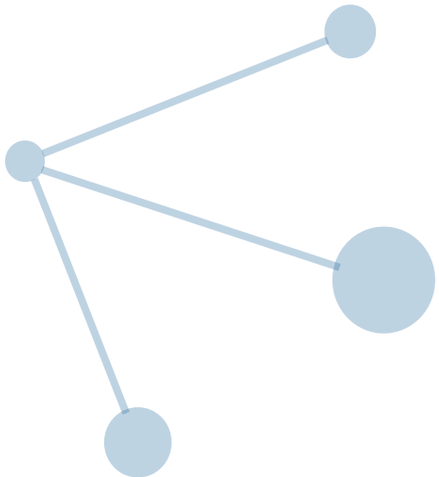
2012

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

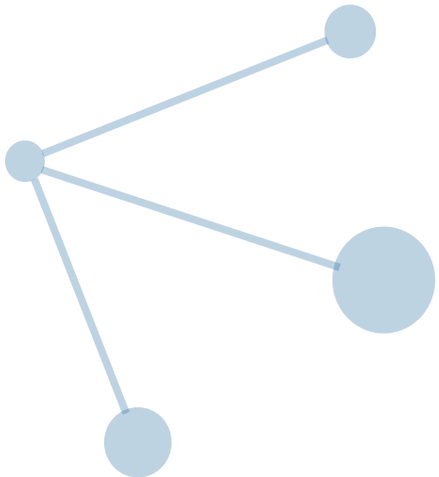
```
def insert(tree: Tree, value: Int): Tree =
  tree match {
    case Leaf() => Branch(value, Leaf(), Leaf())
    case b@Branch(d, l, r) =>
      if (value < d) Branch(d, insert(l, value), r)
      else if (value > d) Branch(d, l, insert(r, value)
      else b
  }
```



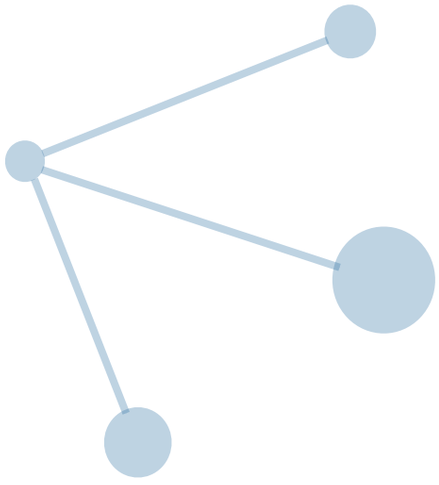
Hmm.. functional programming is pretty cool.



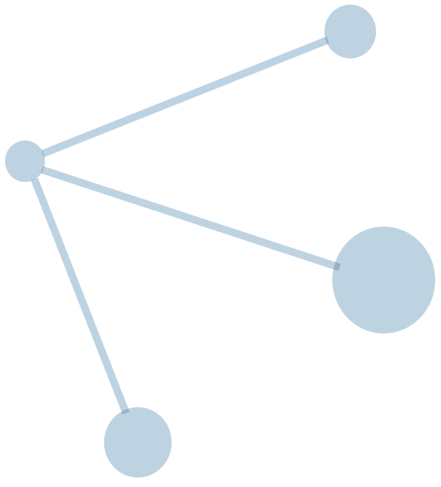
But is it at cool as physics and math?



Let's see..

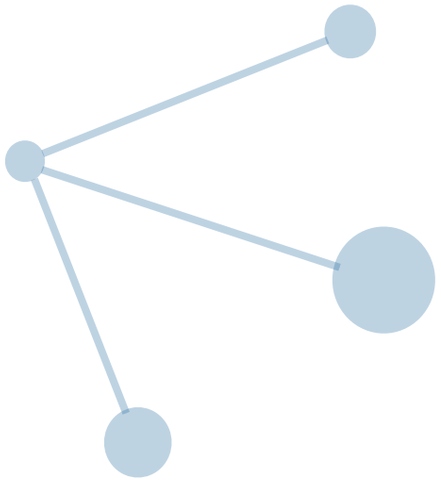


Functional Programming



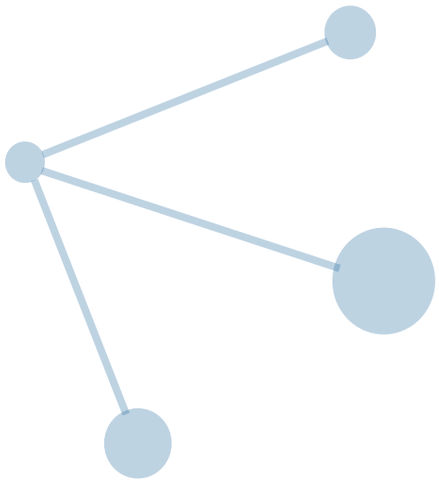
Functional Programming

programming with functions



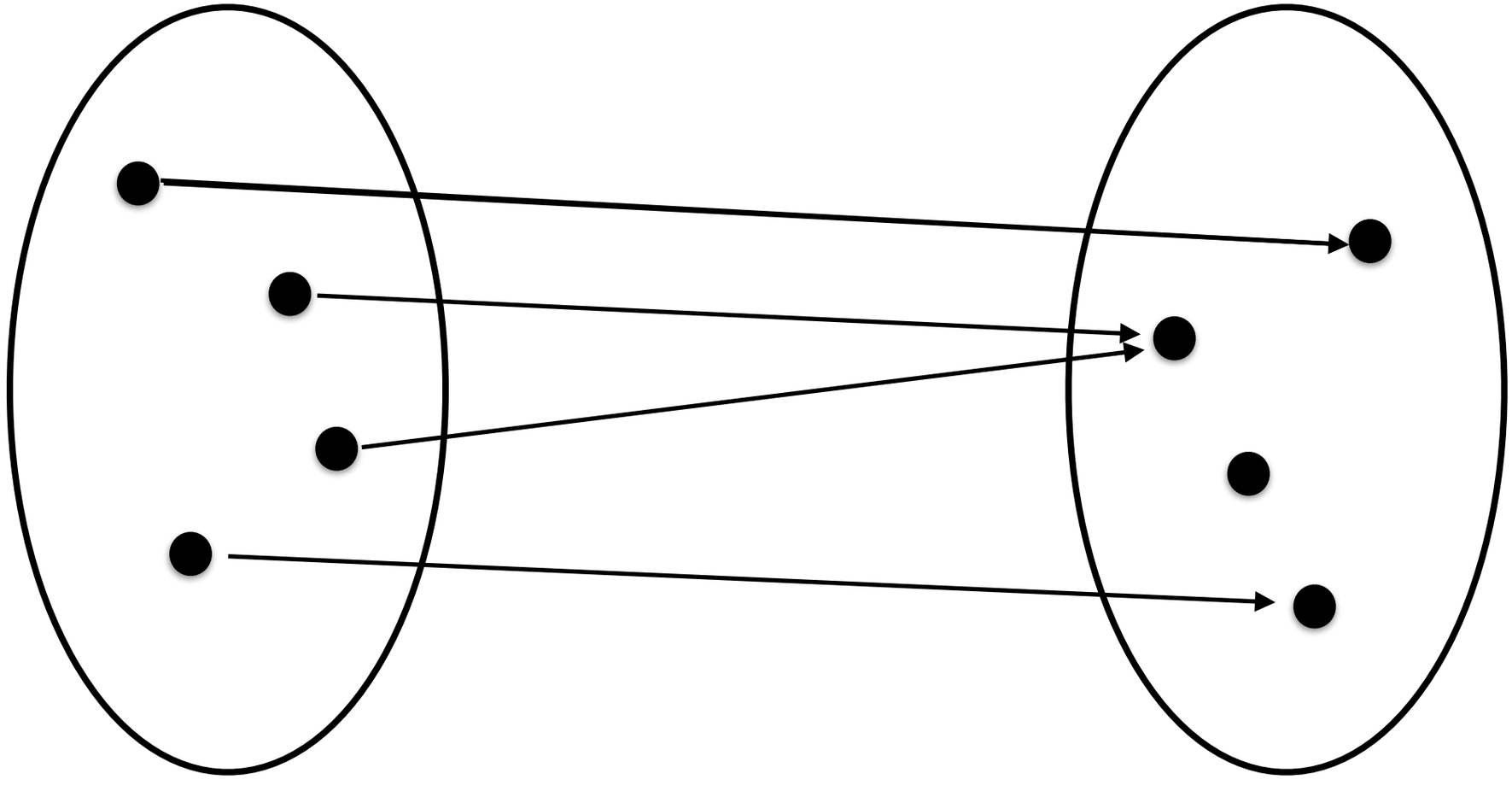
Functional Programming

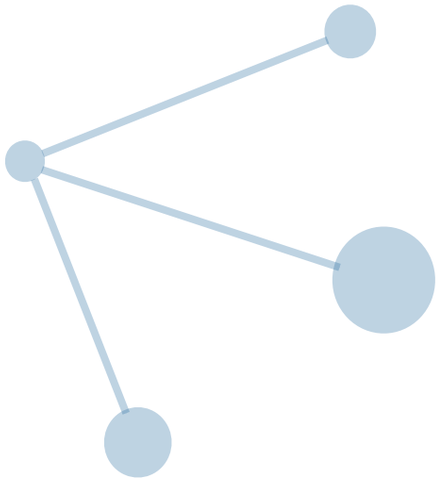
programming with **pure** functions



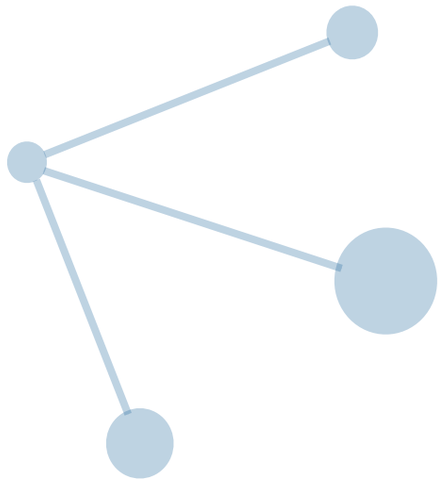
Functional Programming

programming with **pure** functions



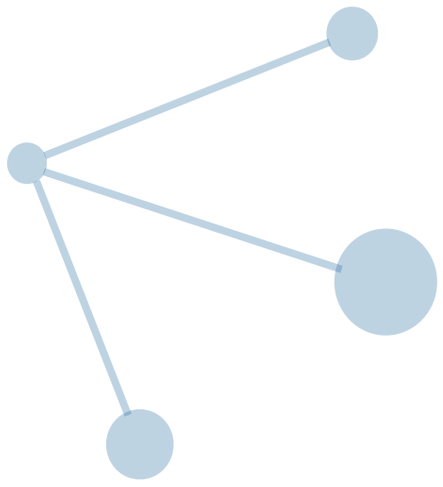


Functions



Functions

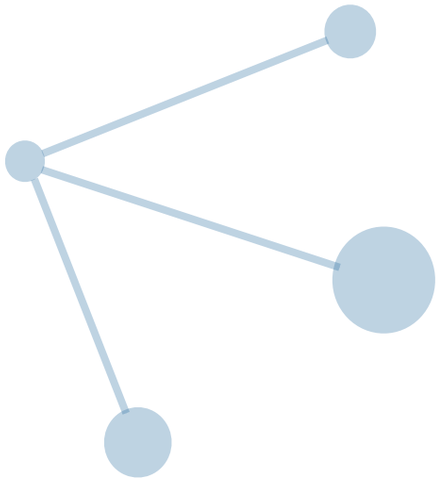
```
def parseIntPartial(s: String): Int =
  s.toInt // can throw NumberFormatException
```



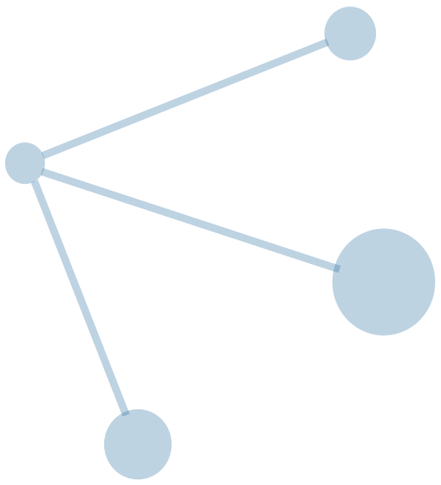
Functions

```
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
  }
```

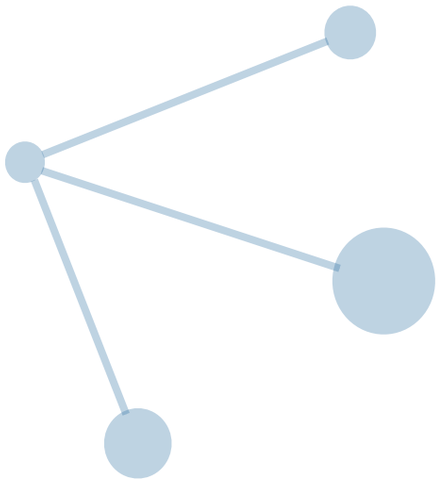


Functions



Functions

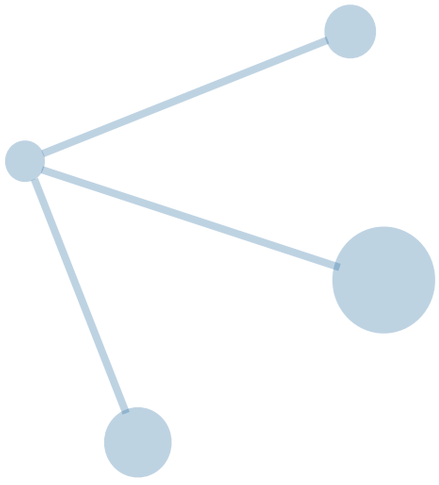
```
class Rng(var seed: Long) {
  def nextInt(): Int = {
    val int = getInt(seed)
    mutate(seed)
    int
  }
}
```



Functions

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

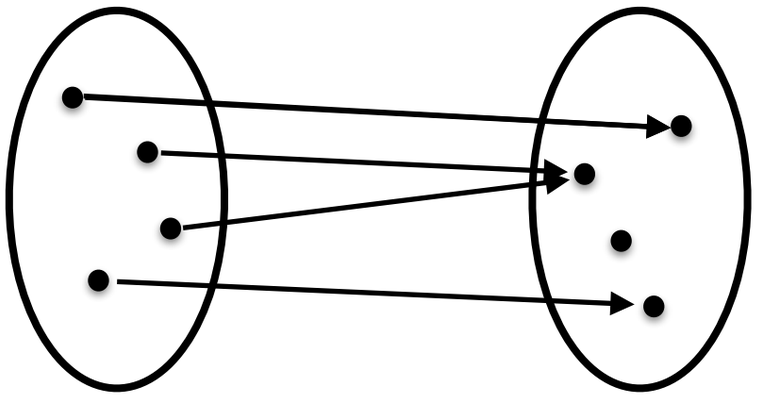
```
case class Rng(seed: Long) {
  def nextInt: (Rng, Int) = {
    val int = getInt(seed)
    val newSeed = f(seed)
    (Rng(newSeed), int)
  }
}
```



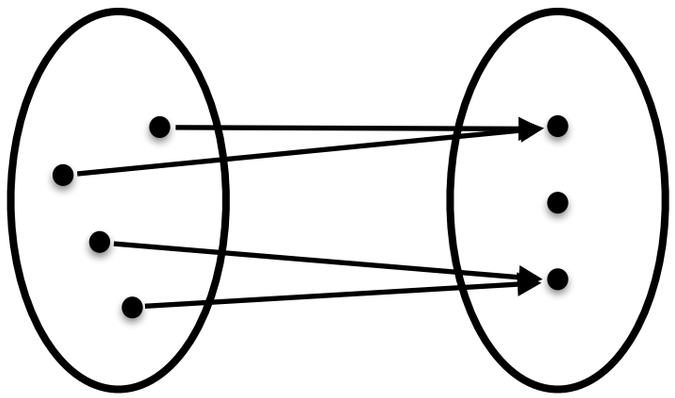
Functions

Functions

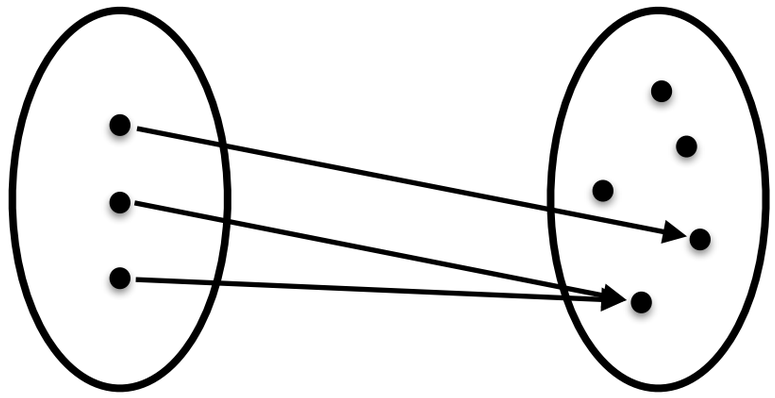
foo: A => B



bar: B => C

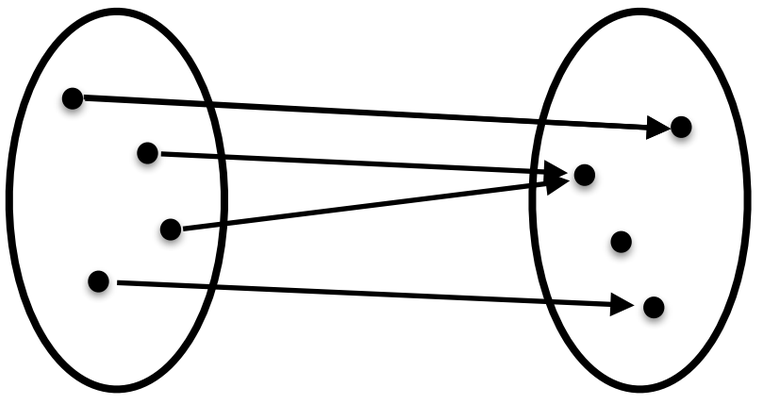


baz: C => D

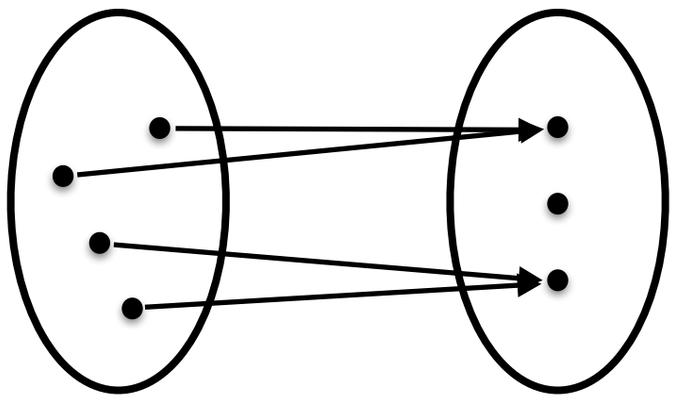


Functions

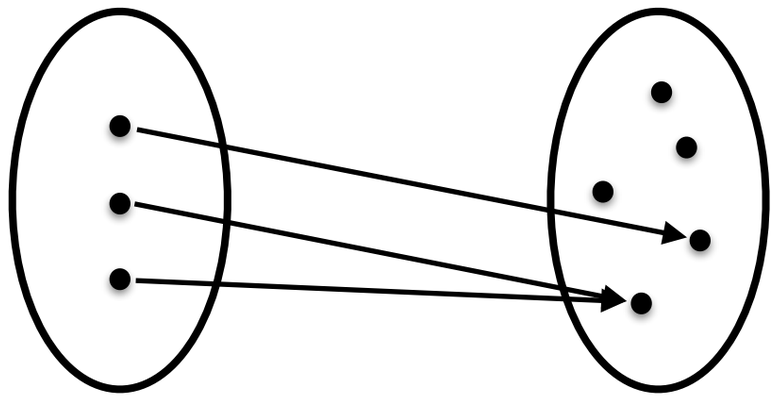
foo: A => B



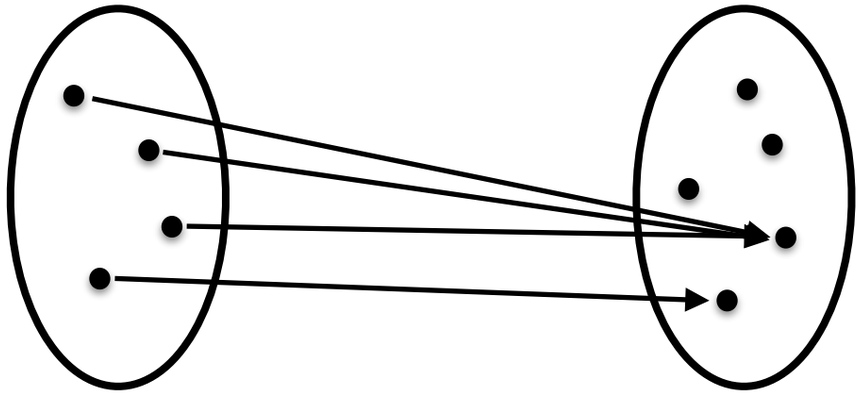
bar: B => C

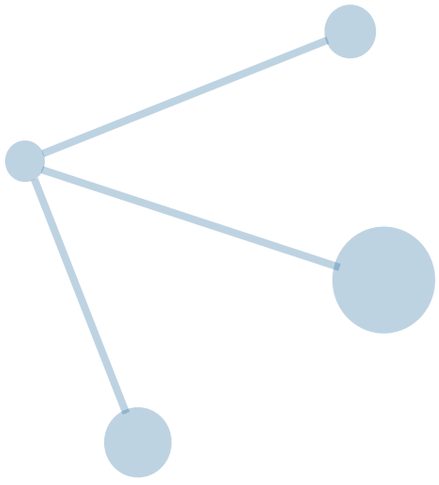


baz: C => D

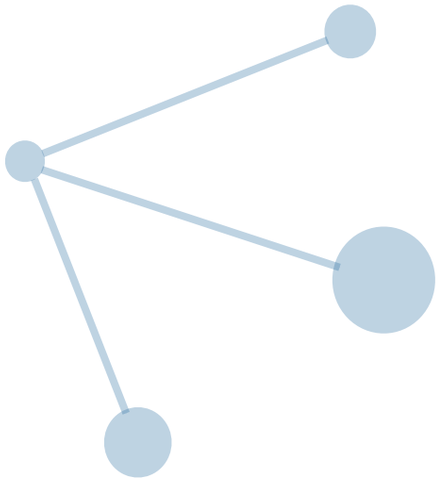


baz.compose(bar).compose(foo): A => D



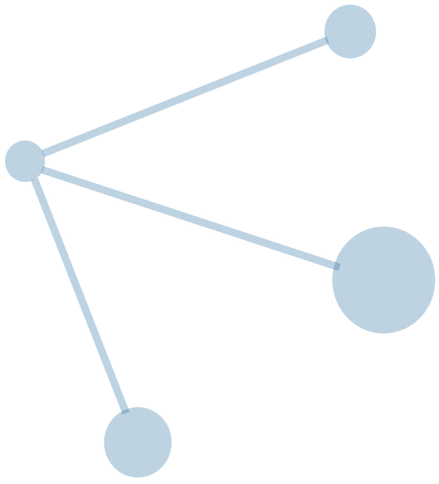


Reducing Values



Reducing Values

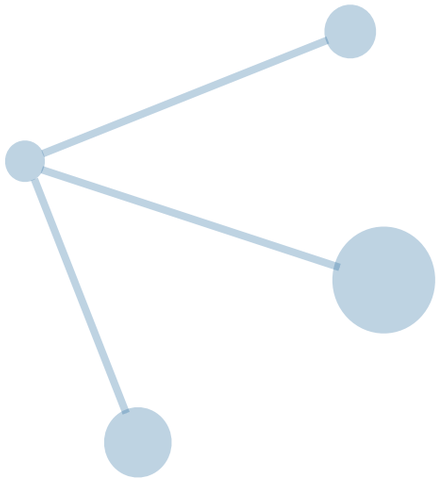
```
def sum(list: List[Int]): Int =
  list.foldLeft(0)(_ + _)
```



Reducing Values

```
def sum(list: List[Int]): Int =
  list.foldLeft(0)(_ + _)
```

```
def concat(list: List[String]): String =
  list.foldLeft("")(_ ++ _)
```

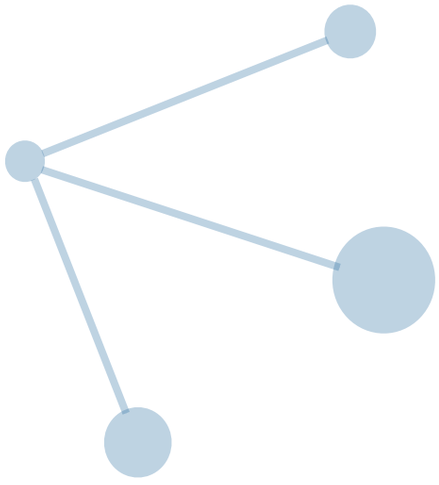


Reducing Values

```scala
def sum(list: List[Int]): Int =
  list.foldLeft(0)(_ + _)
```

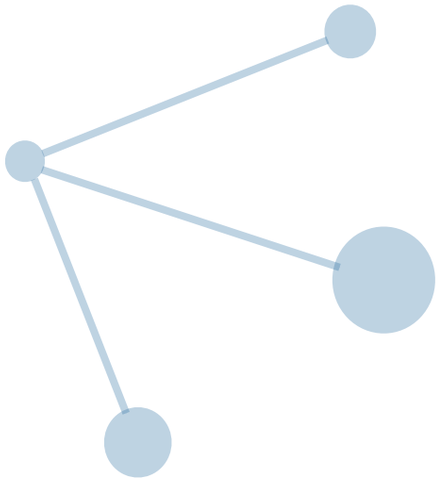
```scala
def concat(list: List[String]): String =
  list.foldLeft("")(_ ++ _)
```

```scala
def union[A](list: List[Set[A]]): Set[A] =
  list.foldLeft(Set.empty[A])(_ union _)
```

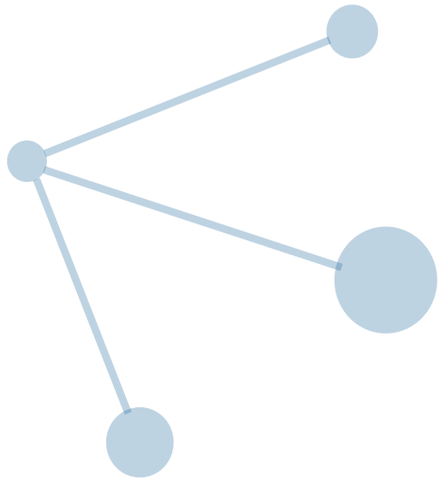


Reducing Values

```
def fold[A](list: List[A]): A =
  list.foldLeft(IDENTITY)(BINARY_OP)
```

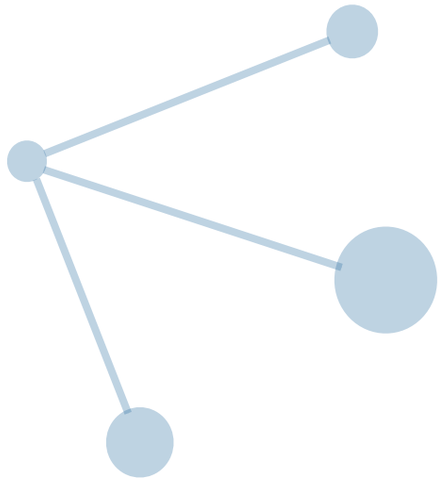


Reducing Values



Reducing Values

```
trait Monoid[A] {
  def append(x: A, y: A): A
  def zero: A
}
```

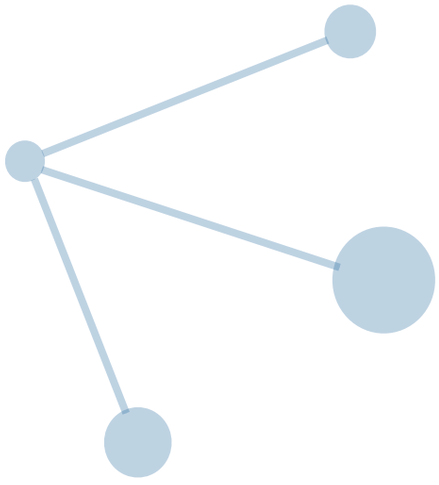


Reducing Values

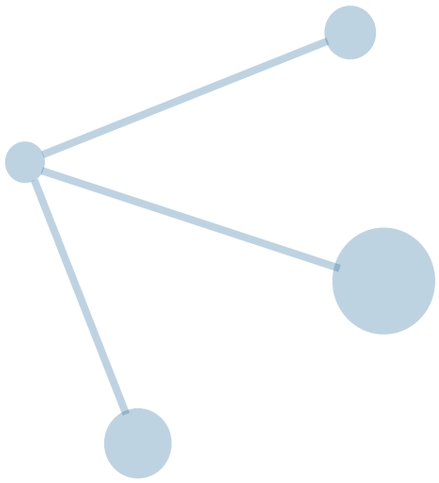
```
trait Monoid[A] {
  def append(x: A, y: A): A
  def zero: A
}

trait Foldable[F[_]] {
  def foldMap[A, B : Monoid](fa: F[A])(f: A => B): B

  def fold[A : Monoid](fa: F[A]): A =
    foldMap(fa)(identity) // identity: x => x
}
```

Reducing Values

Set	Op	Identity
Numbers	+	0
Numbers	x	1
Collections	++	Empty collection
Boolean	&&	TRUE
Boolean	||	FALSE
A => A	∘	identity
(A, B)	position-wise	position-wise

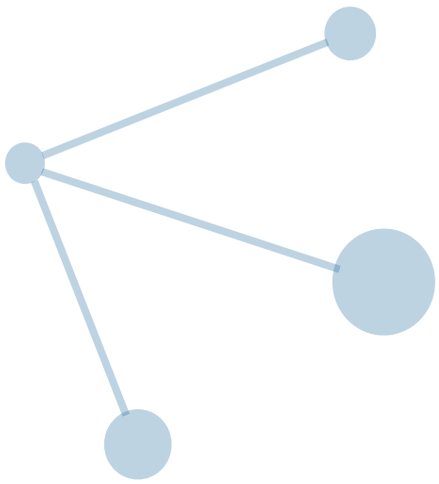


Reducing Values



Reducing Values

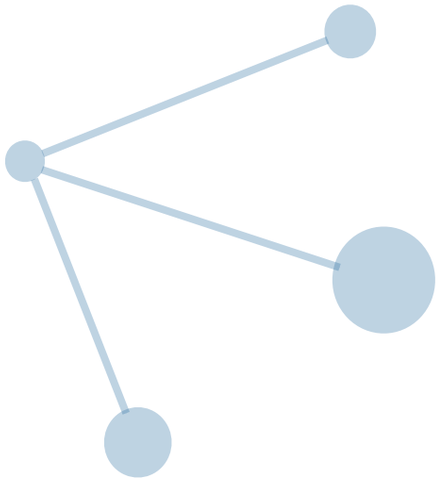
```
trait Foldable[F[_]] {
  def foldMap[A, B : Monoid](fa: F[A])(f: A => B): B

  def forall[A](fa: F[A])(p: A => Boolean): Boolean =
    foldMap(fa)(a => Conjunction(p(a)))

  def exists[A](fa: F[A])(p: A => Boolean): Boolean =
    foldMap(fa)(a => Disjunction(p(a)))

  def toList[A](fa: F[A]): List[A] =
    foldMap(fa)(a => List(a))

  def length[A](fa: F[A]): Int =
    foldMap(fa)(const(1))
}
```



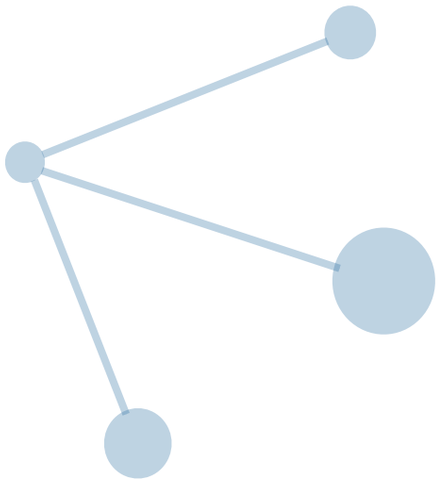
Reducing Values

```
trait Foldable[F[_]] {
  def foldMap[A, B : Monoid](fa: F[A])(f: A => B): B

  def forall[A](fa: F[A])(p: A => Boolean): Boolean =
    foldMap(fa)(a => Conjunction(p(a)))

  def exists[A](fa: F[A])(p: A => Boolean): Boolean =
    foldMap(fa)(a => Disjunction(p(a)))

  def toList[A](fa: F[A]): List[A] =
    foldMap(fa)(a => List(a))

  def length[A](fa: F[A]): Int =
    foldMap(fa)(const(1))
}
```

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

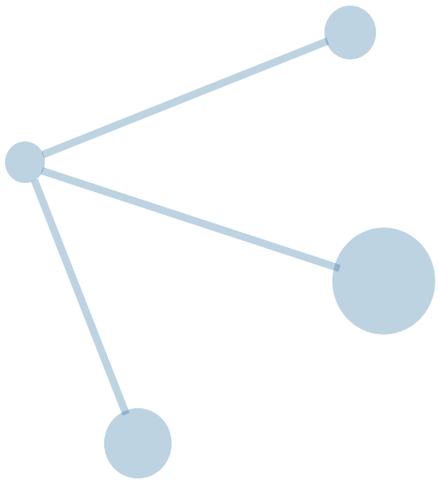


Reducing Values



Reducing Values

```
def sum(list: List[Int]): Int = list.foldMap(identity)

def length(list: List[Int]): Int = list.foldMap(const(1))

def average(list: List[Int]): Int =
  sum(list) / length(list)
```

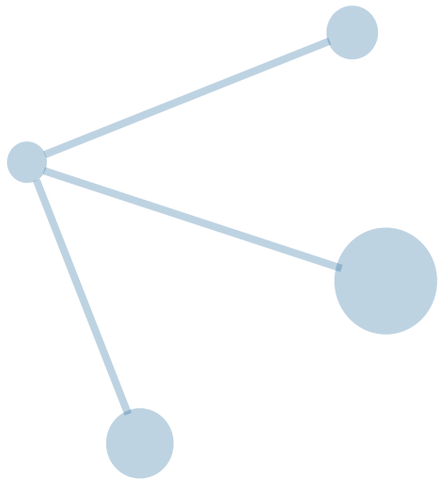


Reducing Values

```
def sum(list: List[Int]): Int = list.foldMap(identity)

def length(list: List[Int]): Int = list.foldMap(const(1))

def average(list: List[Int]): Int =
  sum(list) / length(list)
```

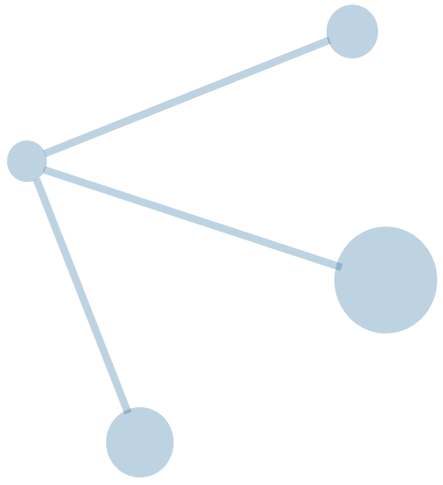
```
def average(list: List[Int]): Int = {
  val (s, l) = list.foldMap(i => (i, 1))
  s / l
}
```

Tracking Effects



Tracking Effects

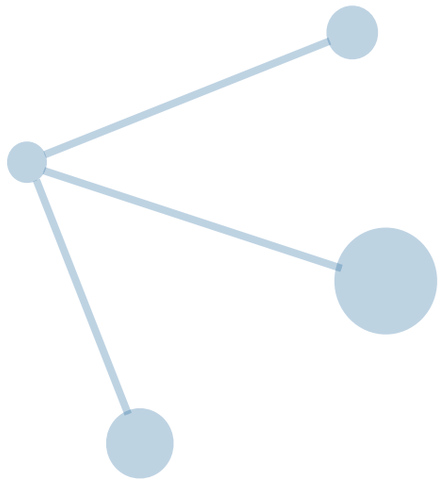
- Interesting programs will have effects
  - Optional values, error handling, asynchronous computation, input/output
  - Usual means aren't quite nice



Tracking Effects

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

```
def lookup(map: Map[Foo, Bar], foo: Foo): Bar =
  if (map.contains(foo)) map(foo) else null

val bar = lookup(someMap, someFoo)
if (bar != null) {
  . . .
} else {
  . . .
}
```

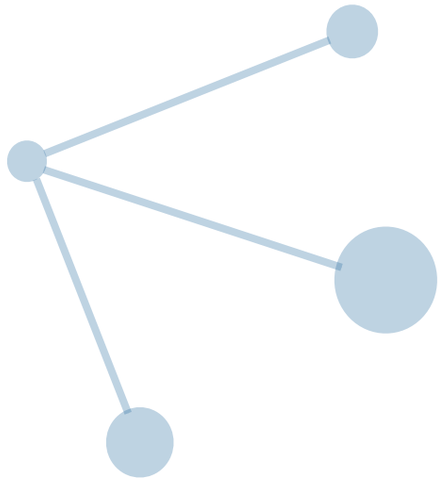


Tracking Effects

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

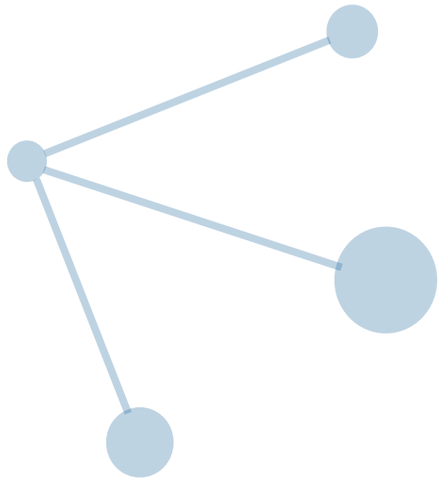
```
def lookup(map: Map[Foo, Bar], foo: Foo): Bar =
  map(foo)

try {
  val bar = lookup(map, someFoo)
} catch {
  case e: NoSuchElementException => . . .
}
```

Tracking Effects



Tracking Effects

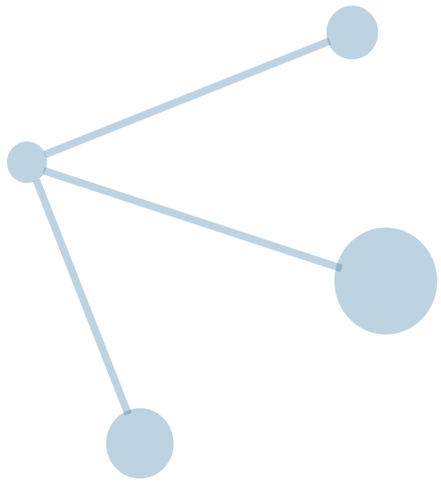
- Reify effects as values



Tracking Effects

- Reify effects as values

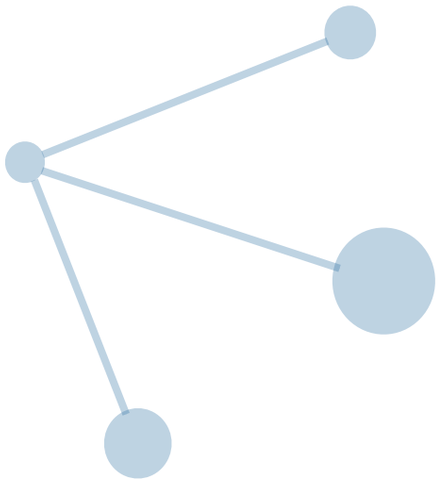
```scala
// A value that might exist is either..
sealed abstract class Option[A]

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

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



Tracking Effects

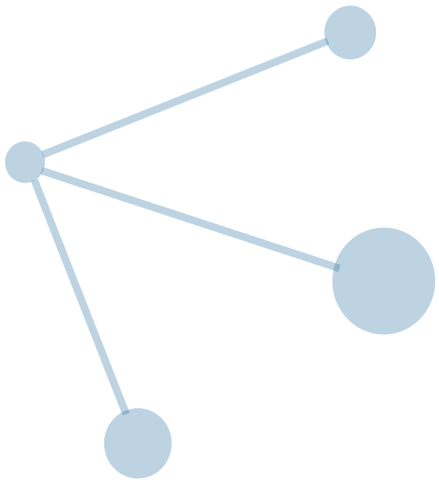
- Reify effects as values

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



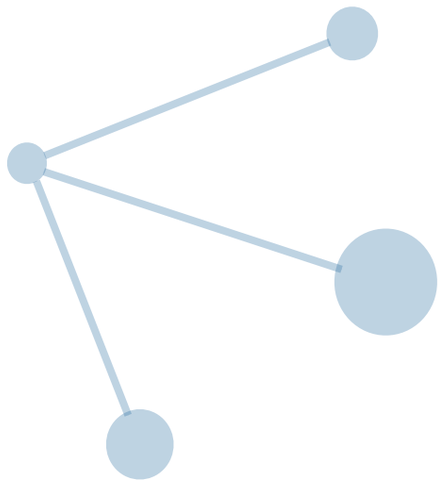
Missing values

```
sealed abstract class Option[A]

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

final case class None[A]() extends Option[A]
```



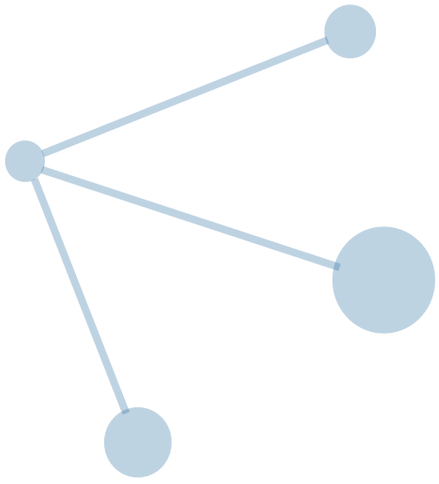
Errors

```
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]
```



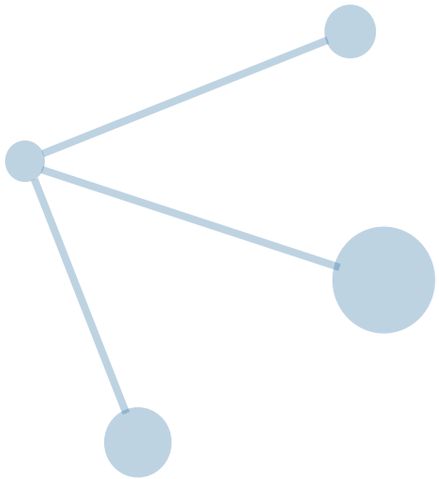
Ex. Errors

```
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] = . . .
```



Input/Output

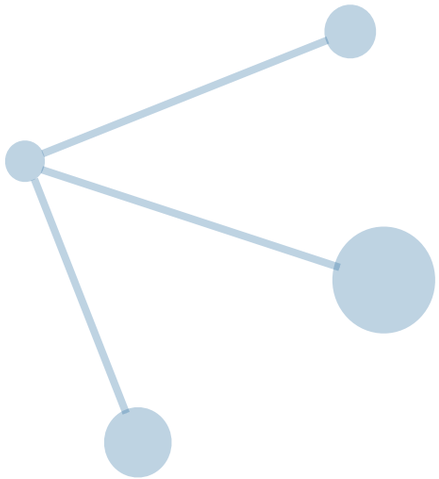
```
trait IO[A] {
  def unsafePerformIO(): A
}
```



Input/Output

```
trait IO[A] {
  def unsafePerformIO(): A
}
```

- Input/output now becomes a value that can be manipulated and more importantly, tracked

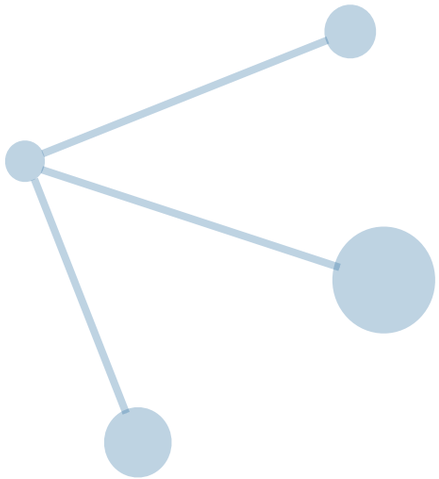


Input/Output

```
trait IO[A] {
  def unsafePerformIO(): A
}
```

- Input/output now becomes a value that can be manipulated and more importantly, tracked

```
val action: IO[Data] = . . .
val retryPolicy = . . .

retryPolicy.retry(action) {
  case HttpError(429) => action
}
```

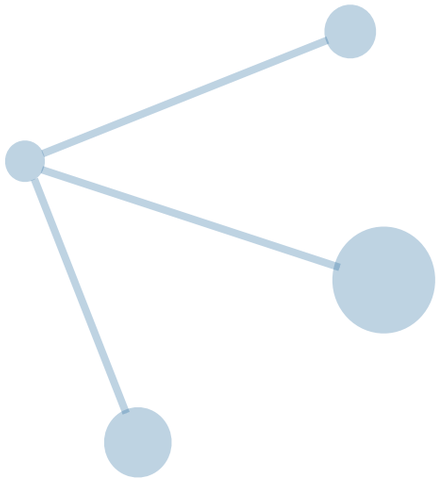


Input/Output

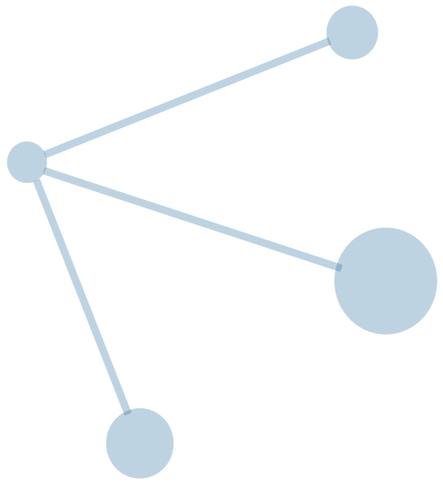
```
trait IO[A] {
  def unsafePerformIO(): A
}
```

- Input/output now becomes a value that can be manipulated and more importantly, tracked

```
trait SafeApp {
  def run(args: List[String]): IO[Unit]

  final def main(args: Array[String]): Unit =
    run(args.toList).unsafePerformIO()
}
```

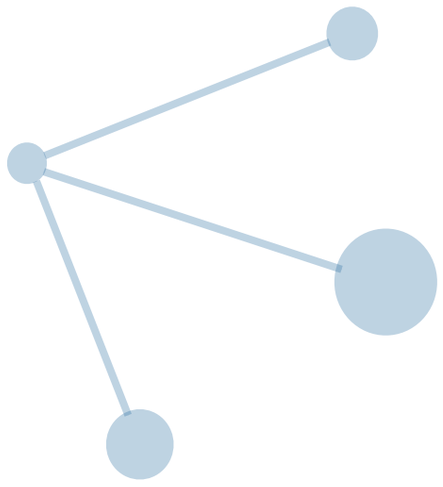


Referential Transparency



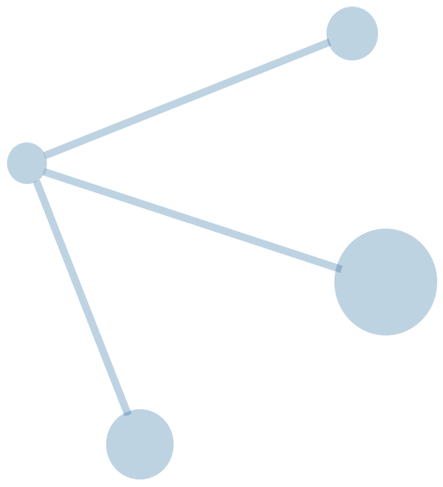
Referential Transparency

An expression **e** is referentially transparent if



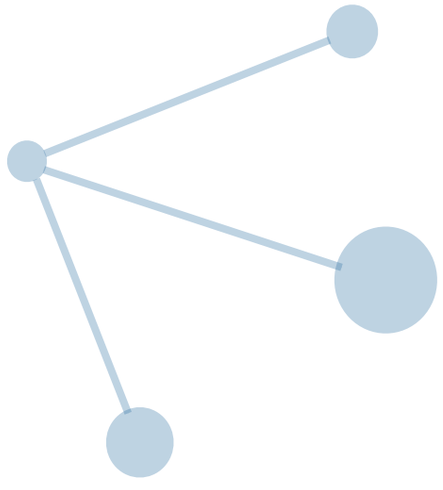
Referential Transparency

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



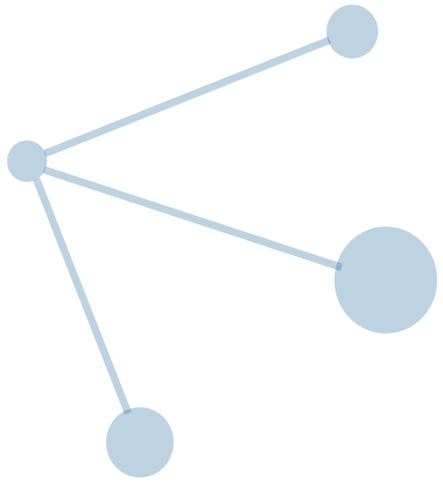
Referential Transparency

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



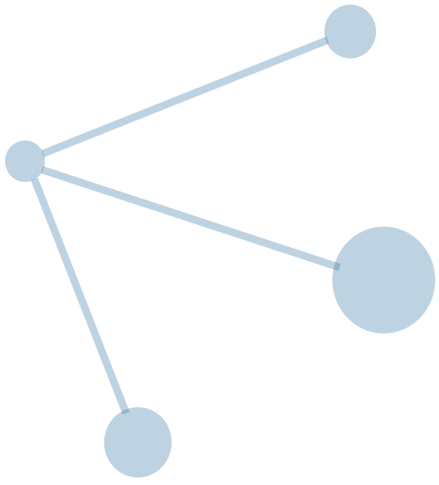
Referential Transparency

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**

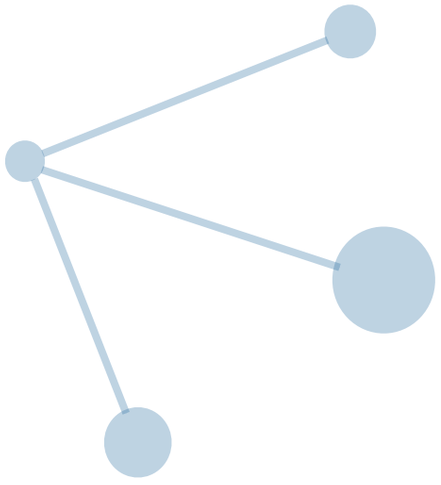


Referential Transparency

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**.

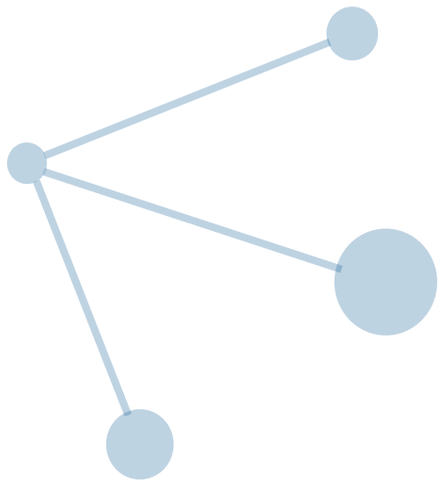


Referential Transparency



Referential Transparency

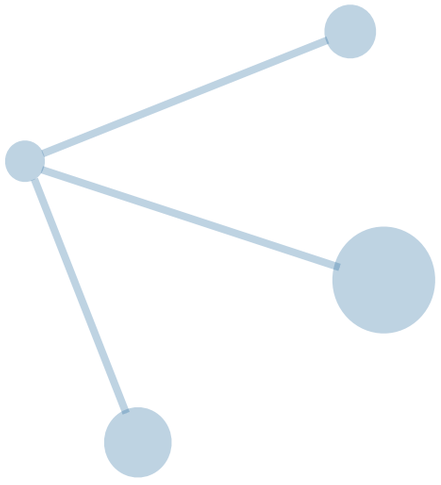
$x^4 - 12x^2 + 36 = 0$



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

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

$y = x^2$

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

$ax^2 + bx + c = 0$

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

$a = 1$

Referential Transparency

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

$y = x^2$

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

$ax^2 + bx + c = 0$

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

$a = 1$

$b = -12$

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

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

$y = x^2$

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

$ax^2 + bx + c = 0$

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

$a = 1$

$b = -12$

$c = 36$

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

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

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

$y = x^2$

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

$y = 6$

$x^2 = 6$

$x = \pm \sqrt{6}$

$ax^2 + bx + c = 0$

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

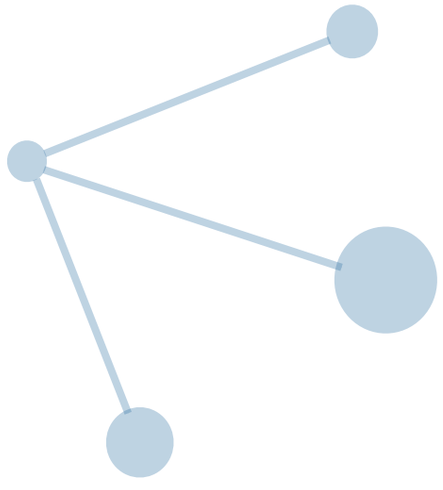
$a = 1$

$b = -12$

$c = 36$

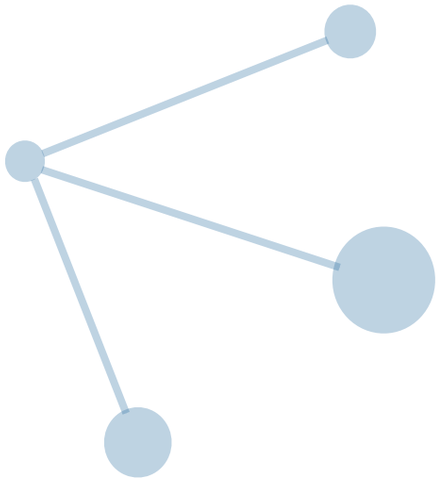
$(-(-12) \pm \sqrt{(-12)^2 - 4(1)(36)}) / 2(1)$

$12 / 2$



Referential Transparency

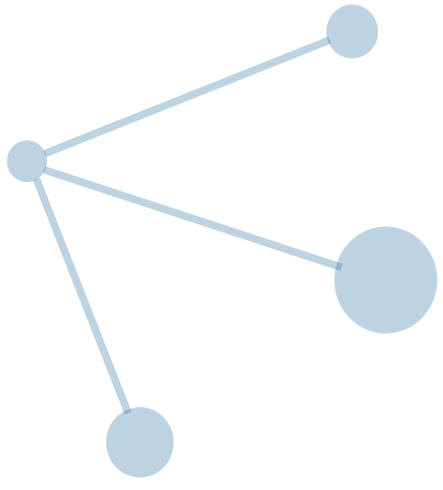
An expression **e** is referentially transparent if
for all programs **p**
every occurrence of **e** in **p**
can be replaced with the result of evaluating **e**
without changing the result of evaluating **p**.



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**.

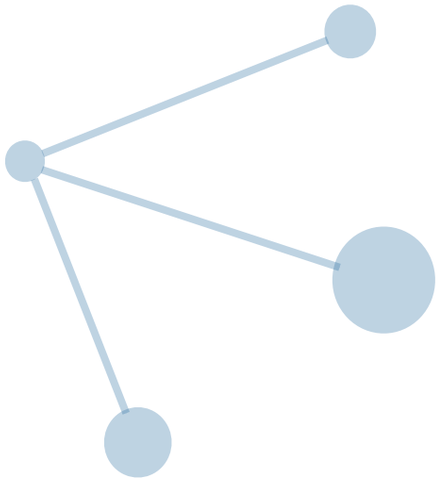
```
val sb = new StringBuilder("Lambda ")

val lc = sb.append("Conf")

val s1 = lc.toString // Lambda Conf

val s2 = lc.toString // Lambda Conf
```



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**.

```
val sb = new StringBuilder("Lambda ")

val lc = sb.append("Conf")

val s1 = lc.toString // Lambda Conf

val s2 = lc.toString // Lambda Conf
```

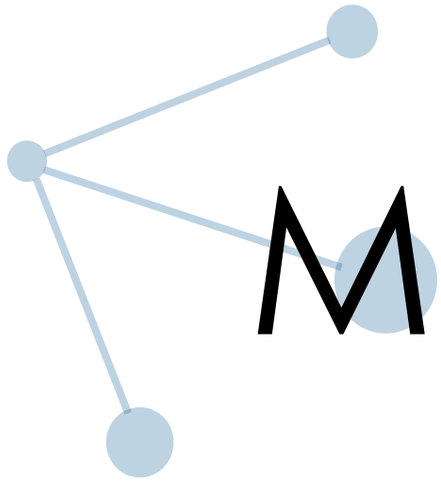


Referential Transparency

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**.

```
val sb = new StringBuilder("Lambda ")

val s1 = sb.append("Conf").toString // Lambda Conf

val s2 = sb.append("Conf").toString // Lambda ConfConf
```

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

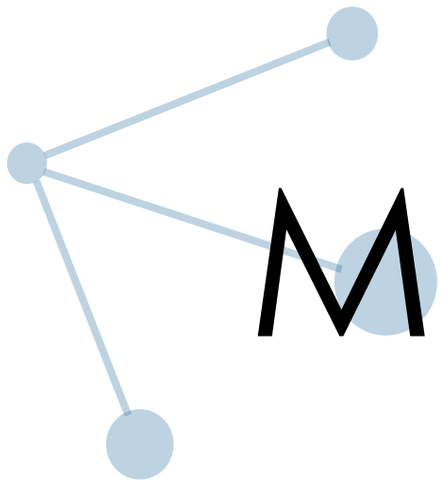
```
def tokenFor(uid: UserId): IO[EncryptedToken]

def decrypt(token: EncryptedToken): Token

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

// Want EncryptedToken, have IO[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]
}
```

```scala
new Functor[IO] {
  def map[A, B](fa: IO[A])(f: A => B): IO[B] =
    new IO[B] {
      def unsafePerformIO(): B =
        f(fa.unsafePerformIO())
    }
}
```

Manipulating Effects



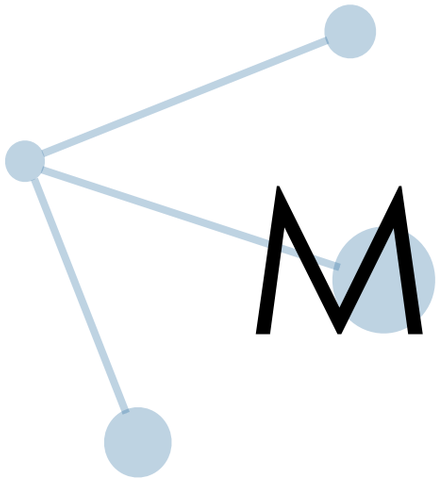
Manipulating Effects

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

def decrypt(token: EncryptedToken): Token

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

// Want EncryptedToken, have IO[EncryptedToken]
val decrypted: IO[Token] =
  encrypted.map(et => decrypt(et))
```

Manipulating Effects

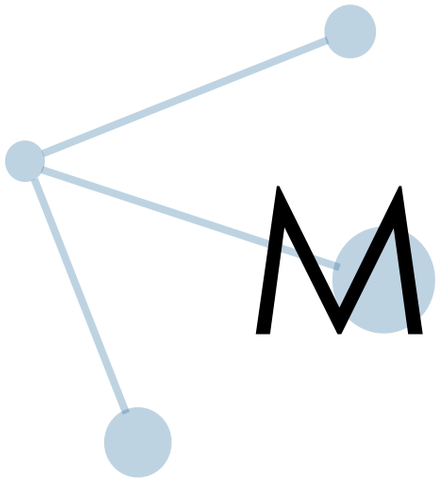
```
def tokenFor(uid: UserId): IO[EncryptedToken]

def decrypt(token: EncryptedToken): Token

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

// Want EncryptedToken, have IO[EncryptedToken]
val decrypted: IO[Token] =
  encrypted.map(et => decrypt(et))
```

- Similar mechanisms for manipulating multiple effects



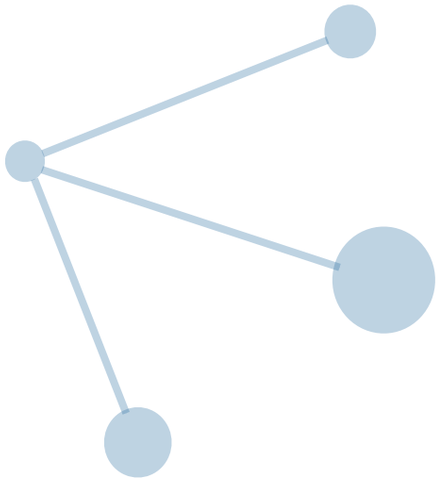
Manipulating Effects

```scala
/** Apply a pure binary function to 2 effectful values */
trait Applicative[F[_]] extends Functor[F] {
  def map2[A, B, C](fa: F[A], fb: F[B])(f: (A, B) => C): F[C]

  def pure[A](a: A): F[A]

  def map[A, B](fa: F[A])(f: A => B): F[B] =
    map2(fa, point(())) { case (a, _) => f(a) }
}
```

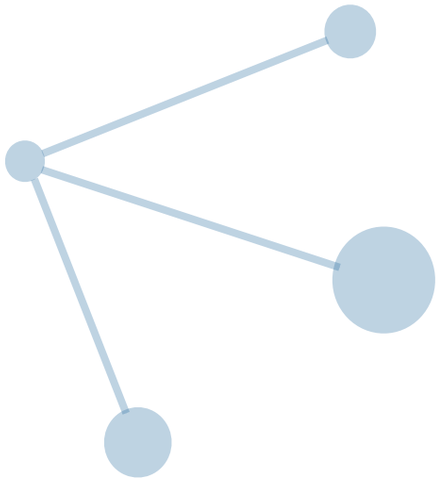


Manipulating Effects

```
/** Apply a pure binary function to 2 effectful values */
trait Applicative[F[_]] extends Functor[F] {
  def map2[A, B, C](fa: F[A], fb: F[B])(f: (A, B) => C): F[C]

  def pure[A](a: A): F[A]
}

new Applicative[IO] {
  def map2[A, B, C](fa: IO[A], fb: IO[B])
                   (f: (A, B) => C): IO[C] =
    new IO[C] { def unsafePerformIO(): C =  f(fa.un.., fb.un..) }

  def pure[A](a: A): IO[A] =
    new IO[A] { def unsafePerformIO(): A = a }
}
```

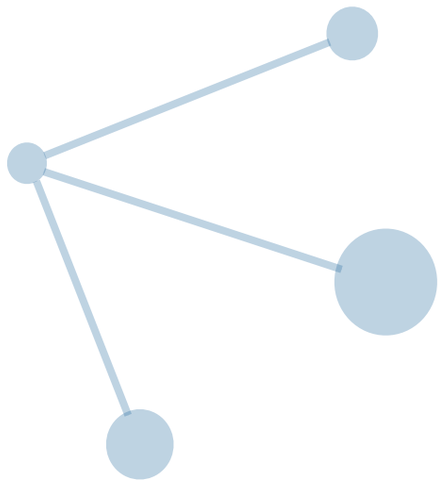


Lens



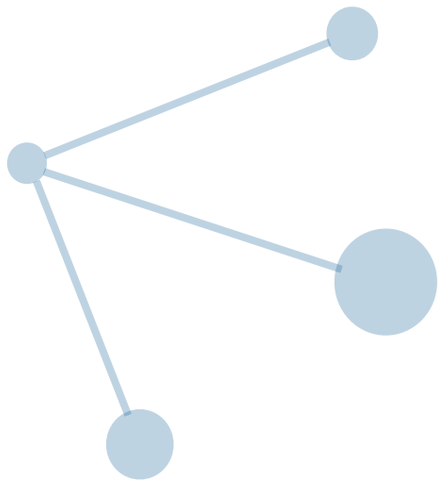
Lens

- Often want getters/setters when working with data



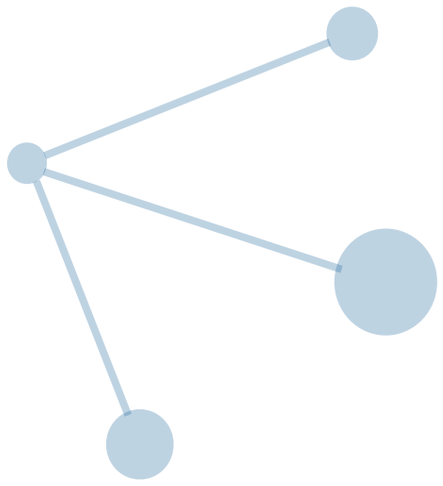
Lens

- 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**

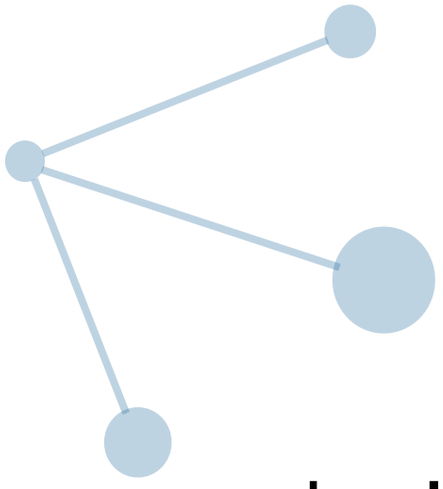


Lens

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

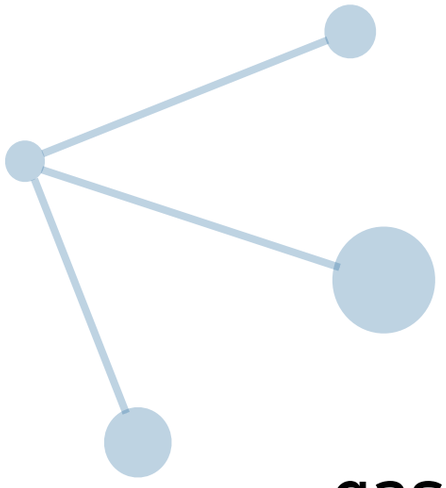
- Getter: get an **A** field in an object **S**
- Setter: change an **A** field in an object **S**

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

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



Lens

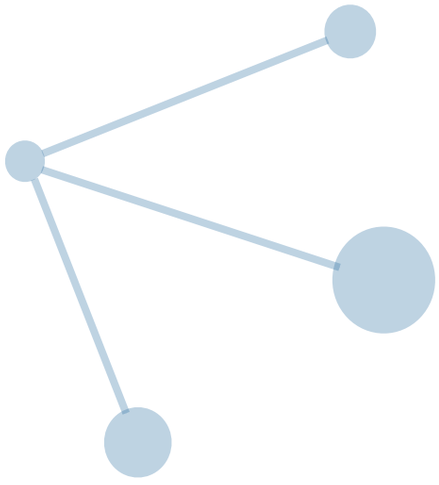
```
trait Lens[S, A] { outer =>
  def get(s: S): A

  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))
    }
}
```



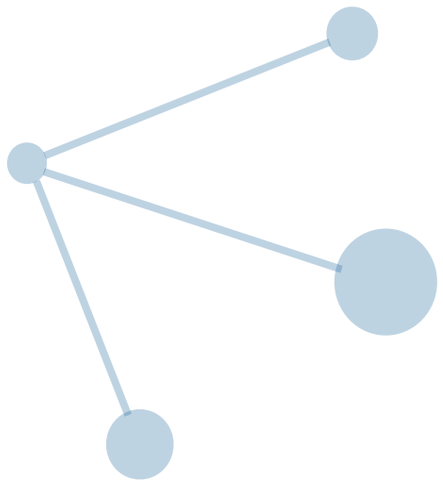
Lens

```
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)
```

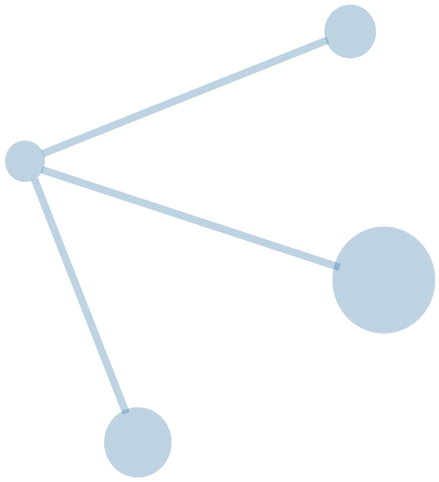


Lens



Lens

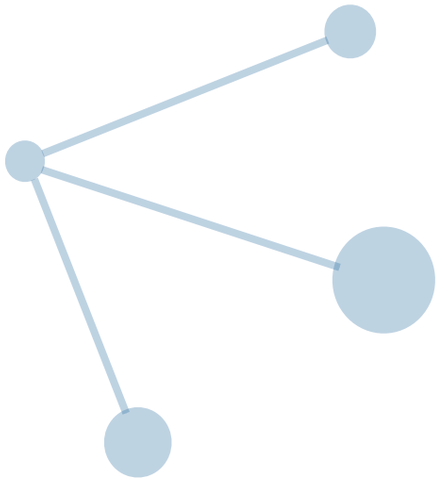
- Effectful modifications can be useful
  - Possibly failing: A => Option[A]
  - Many possible values: A => List[A]
  - IO: A => IO[A]



Lens

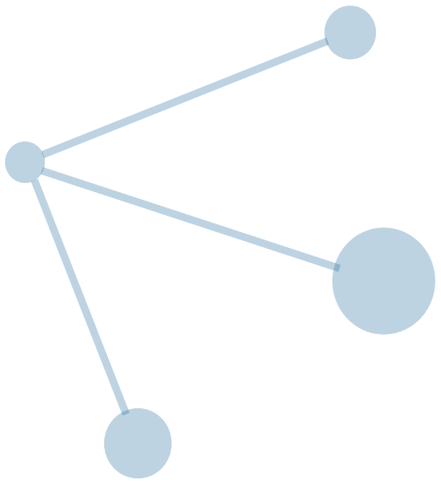
- Effectful modifications can be useful
  - Possibly failing: A => Option[A]
  - Many possible values: A => List[A]
  - IO: A => IO[A]

```
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 modifyIO(s: S, f: A => IO[A]): IO[S]
}
```



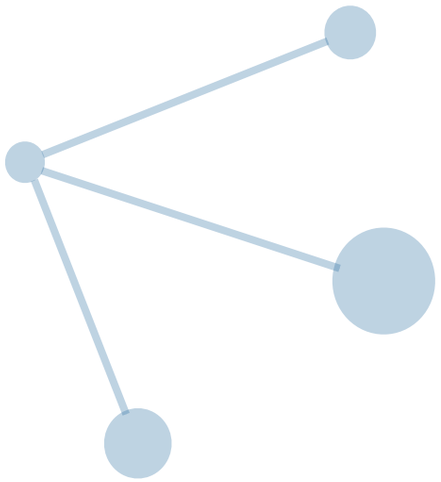
Lens

```
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 modifyIO(s: S, f: A => IO[A]): IO[S] =
    f(get(s)).map(a => set(s, a))
}
```



Lens

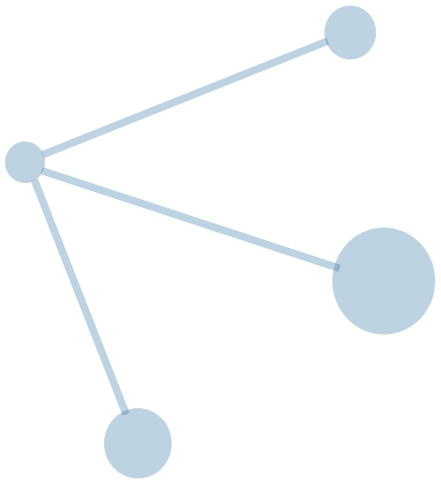
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```



Lens

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

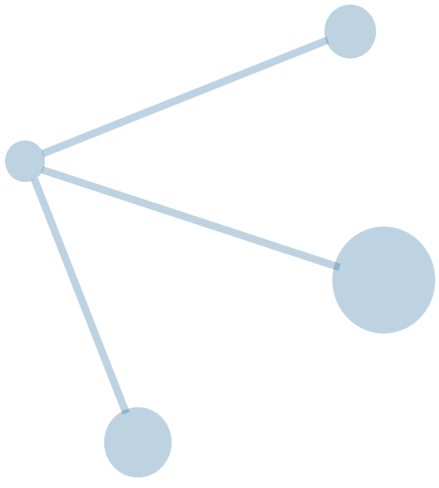
- Example functors:
  - Option, Either, Future, List, IO



Lens

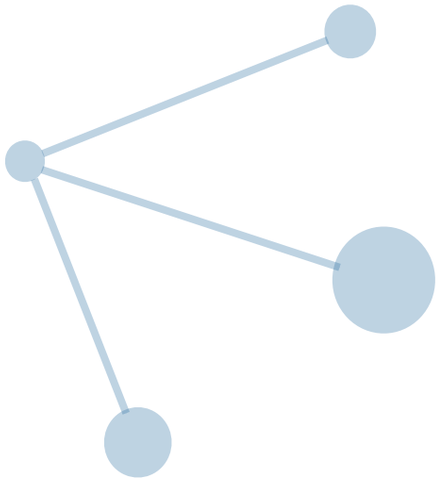
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))
}
```

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



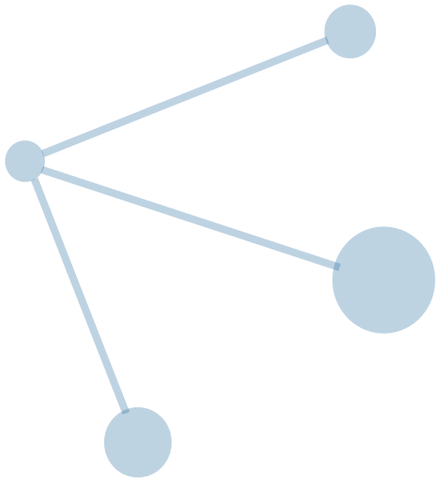
Id

```
type Id[A] = A

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



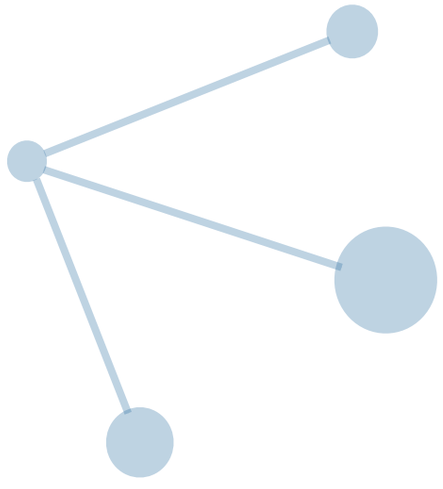
Id

```
type Id[A] = A

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



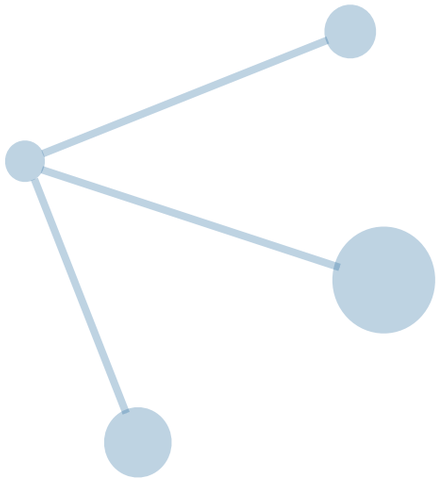
Id

```
type Id[A] = A

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



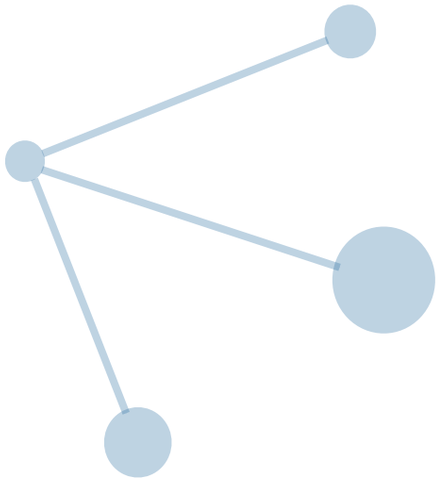
Id

```
type Id[A] = A

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



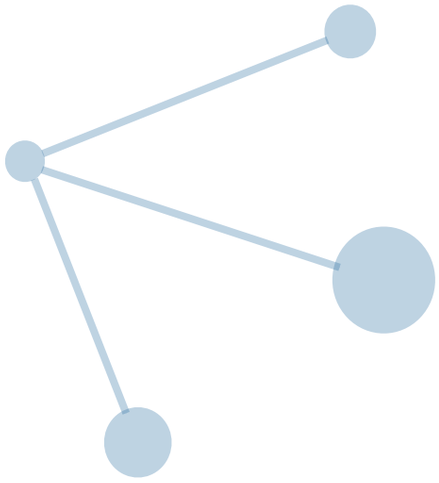
Id

```
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

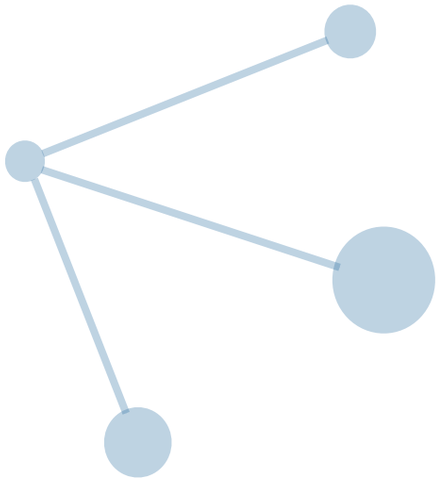
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S] =
    f(get(s)).map(a => set(s, a))

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

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



Setting

```
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))
}
```

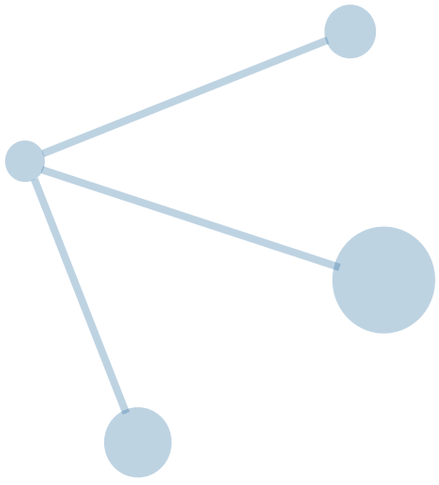
```
def const[A, B](a: A)(b: B): A = a
```



Setting

```
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))
}
```

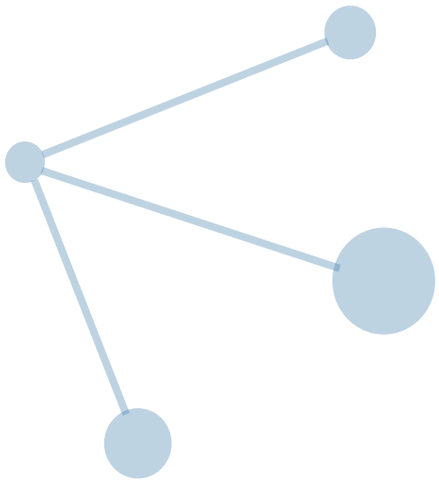
```
def const[A, B](a: A)(b: B): A = a
```



Setting

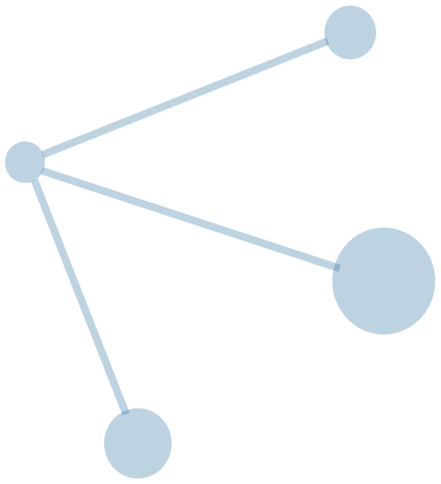
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  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))
}
```

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



Getting

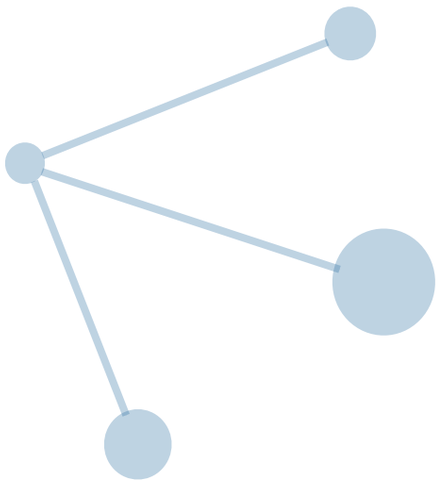
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

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



Getting

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

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

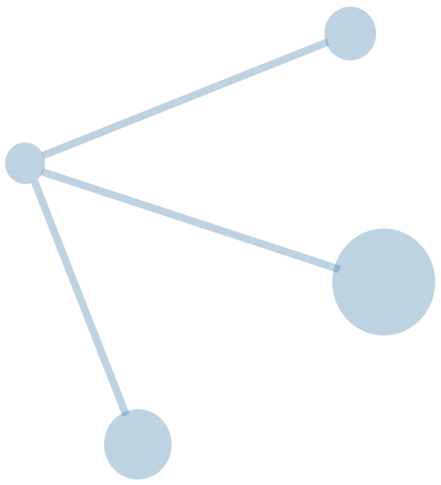
- Can we define `get` in terms of `modify`?



Getting

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

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

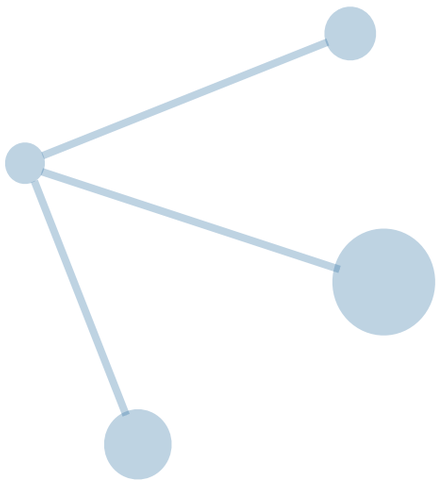
- Can we define `get` in terms of `modify`?
  - `modifyF` gives us some F[S].. but we want an A



Getting

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

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

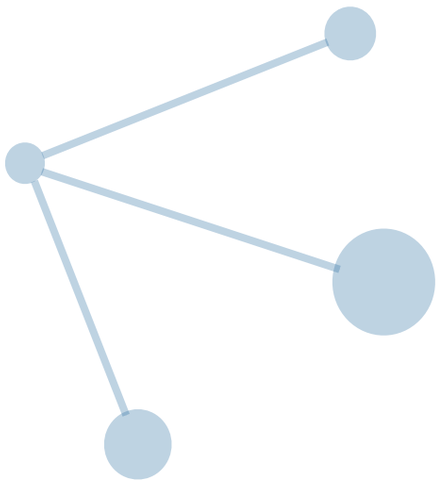
- Can we define `get` in terms of `modify`?
  - `modifyF` gives us some F[S].. but we want an A
  - We need some way of "ignoring" the S parameter and still get an A back



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
    - Reify constant function to type-level

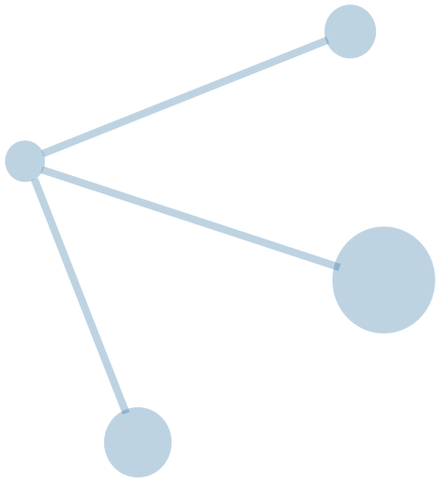


Getting

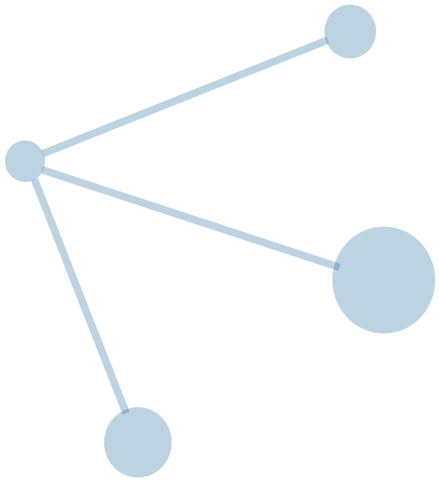
```
trait Lens[S, A] {
  def modifyF[F[_] : Functor](s: S, f: A => F[A]): F[S]

  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
    - Reify constant function to type-level

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

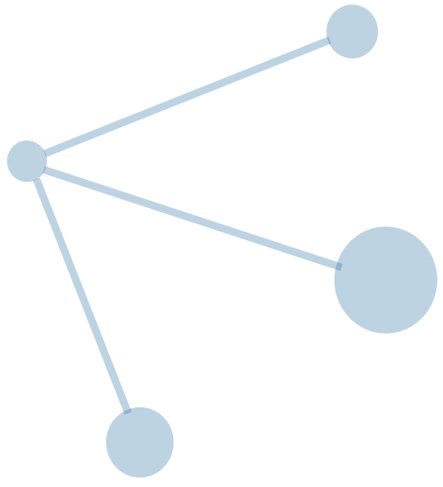


Getting



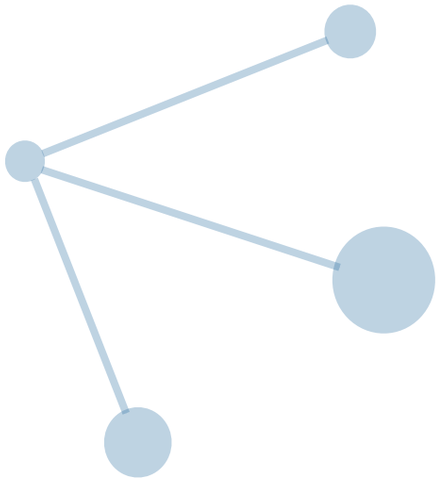
Getting

```
case class Const[Z, A](get: Z)
```



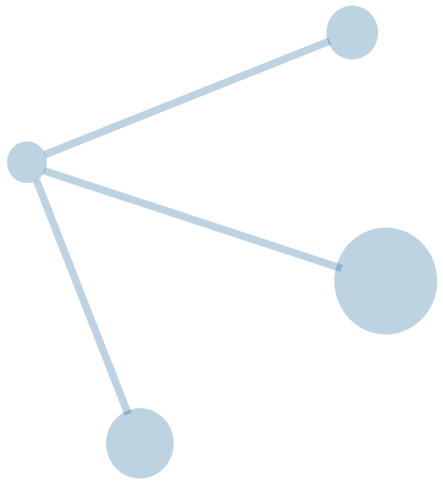
Getting

```
case class Const[Z, A](get: Z)


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



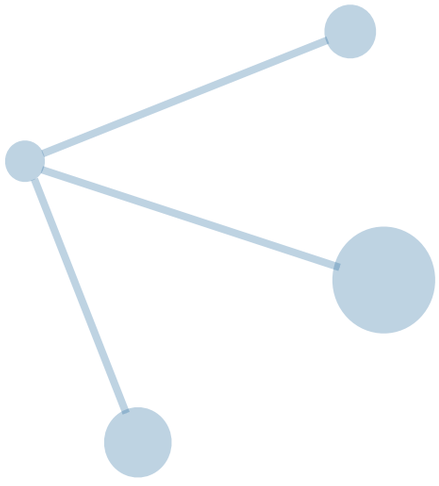
Getting

```
case class Const[Z, A](get: Z)


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



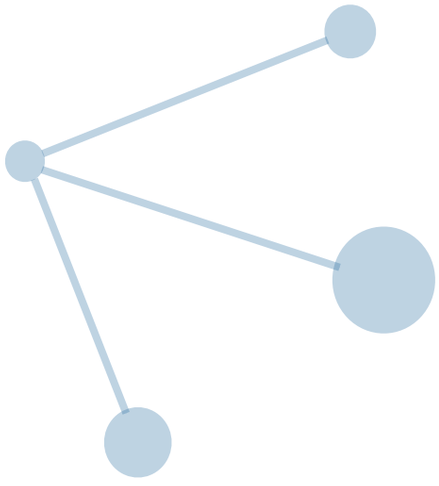
Getting

```
case class Const[Z, A](get: Z)


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



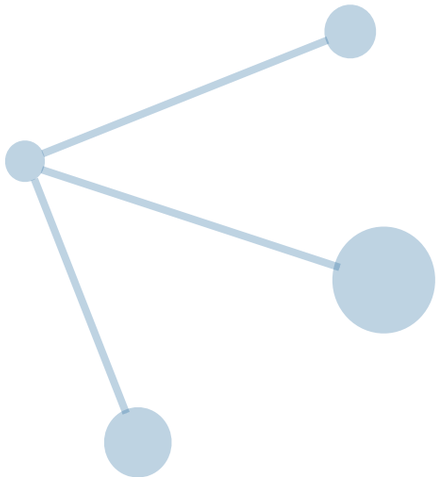
Getting

```
case class Const[Z, A](get: Z)


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

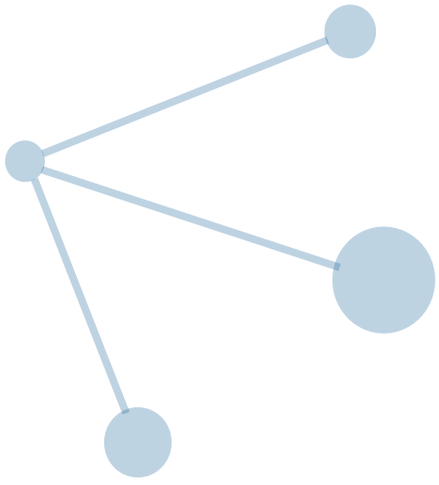


Getting

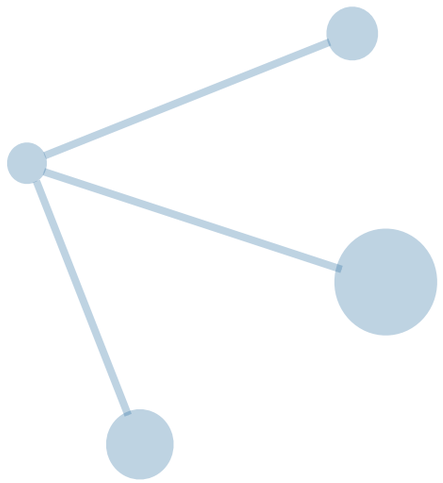
```
case class Const[Z, A](get: Z)


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


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 => Const(z))
    const.get
  }
}
```



Id and Const are party tricks!

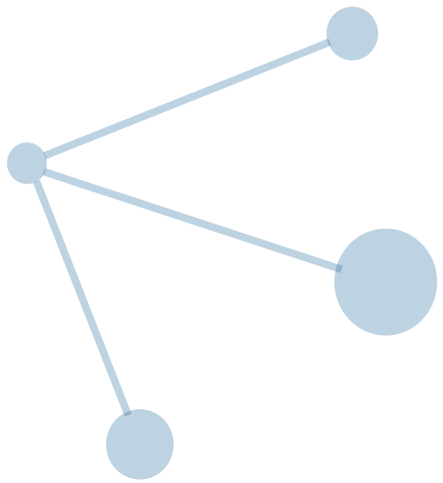


Traversal



Traversal

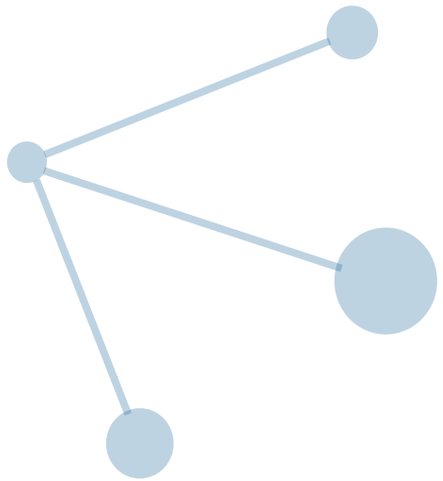
```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}
```



Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}
```

```
def map2[A, B](fa: F[A], fb: F[B])(f: (A, B) => C): F[C]
```

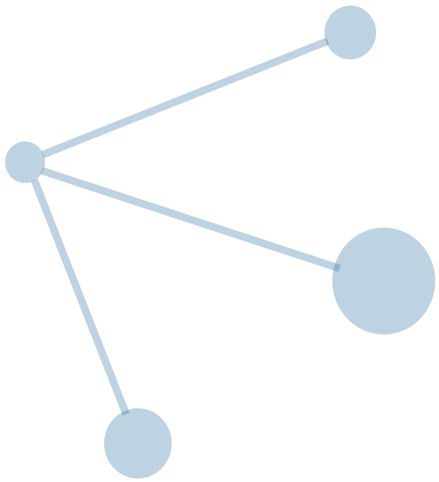


Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}
```

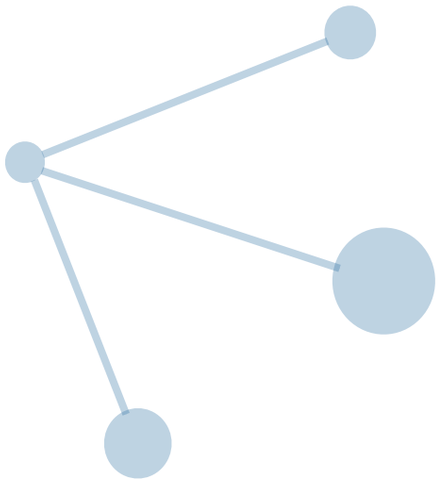
```
def scatterGather[A, B]
    (list: List[A])(f: A => Future[B]): Future[List[B]]
```

```
def map2[A, B](fa: F[A], fb: F[B])(f: (A, B) => C): F[C]
```



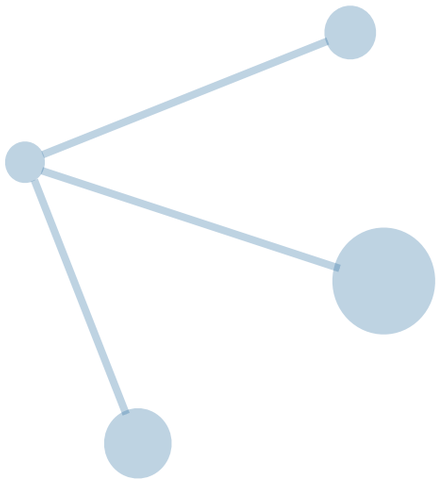
Traversal

```
new Applicative[Id] {
  def map2[A, B, C](fa: Id[A], fb: Id[B])(f: (A, B) => C): Id[C] =

  def pure[A](a: => A): Id[A] =
}
```



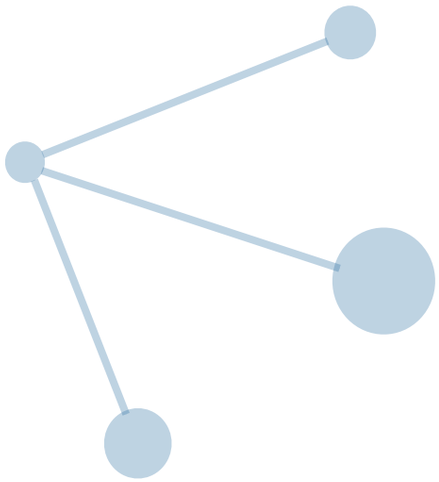
Traversal

```
new Applicative[Id] {
  def map2[A, B, C](fa: Id[A], fb: Id[B])(f: (A, B) => C): Id[C] =

  def pure[A](a: => A): Id[A] =
}
```



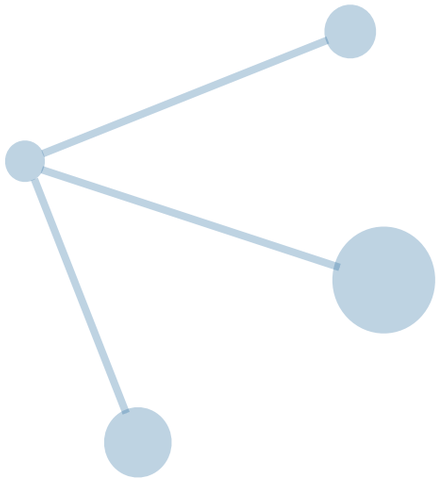
Traversal

```
new Applicative[Id] {
  def map2[A, B, C](fa: A, fb: B)(f: (A, B) => C): C =

  def pure[A](a: => A): A =
}
```



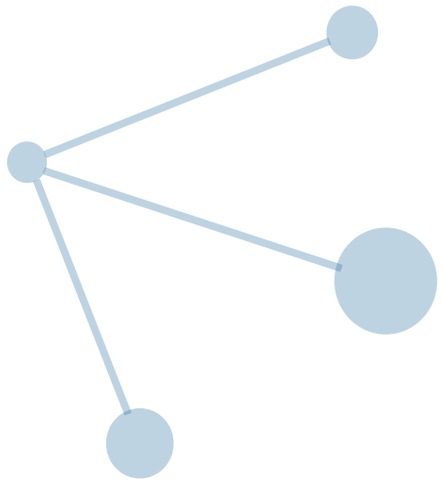
Traversal

```
new Applicative[Id] {
  def map2[A, B, C](fa: A, fb: B)(f: (A, B) => C): C = f(fa, fb)

  def pure[A](a: => A): A = a
}
```



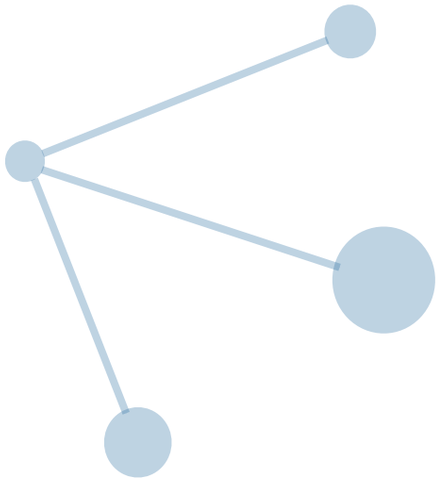
Traversal

```
new Applicative[Const[Z, ?]] {
  def map2[A, B, C]
          (fa: Const[Z, A], fb: Const[Z, B])
          (f: (A, B) => C): Const[Z, C] =

  def pure[A](a: => A): Const[Z, A] =
}
```



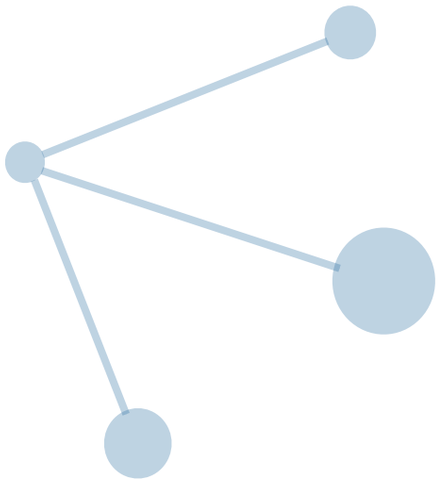
Traversal

```
new Applicative[Const[Z, ?]] {
  def map2[A, B, C]
          (fa: Const[Z, A], fb: Const[Z, B])
          (f: (A, B) => C): Const[Z, C] =

  def pure[A](a: => A): Const[Z, A] =
}
```



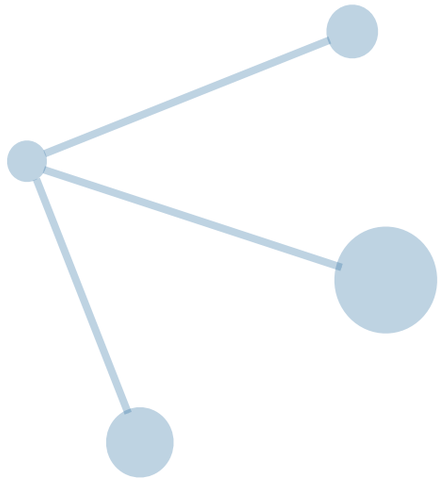
Traversal

```
new Applicative[Const[Z, ?]] {
  def map2[A, B, C]
          (fa: Z, fb: Z)
          (f: (A, B) => C): Z =

  def pure[A](a: => A): Z =
}
```



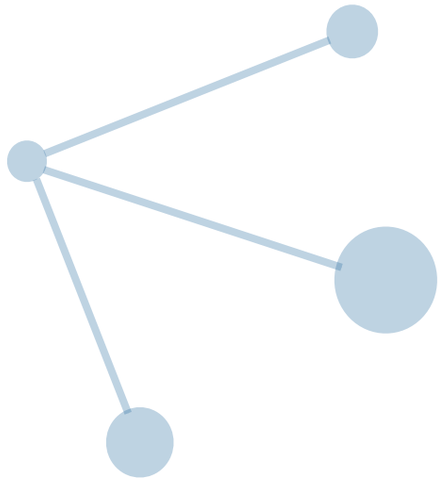
Traversal

```
new Applicative[Const[Z, ?]] { // If Z can form a Monoid..
  def map2[A, B, C]
          (fa: Z, fb: Z)
          (f: (A, B) => C): Z = fa.append(fb)

  def pure[A](a: => A): Z = Monoid[Z].zero
}
```



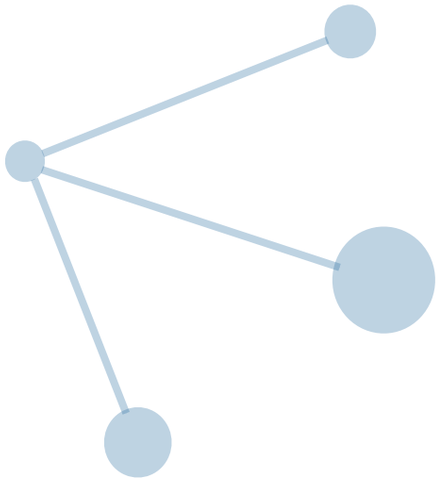
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}
```



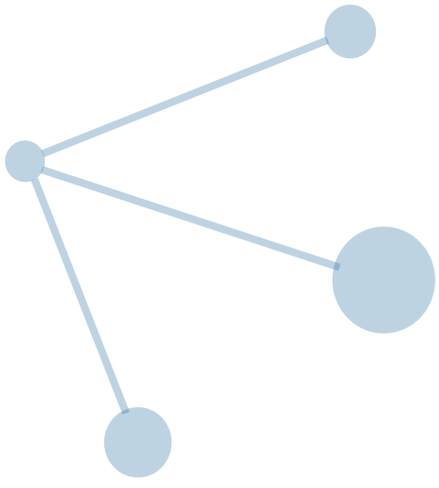
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}

traverse[Id, A, B]
```



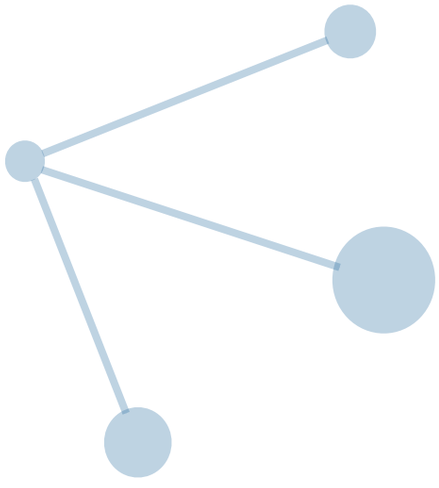
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}


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



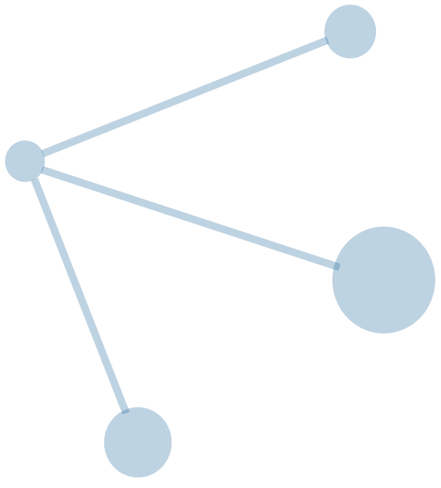
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}


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



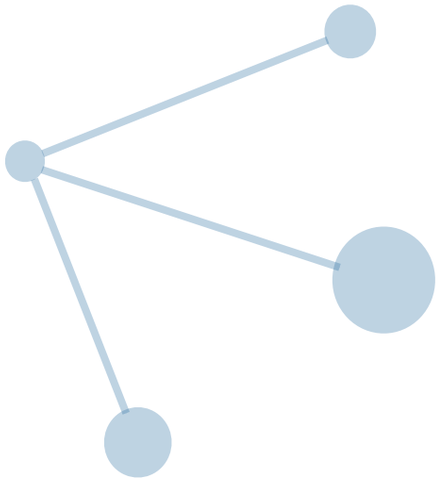
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
              (fa: F[A])(f: A => G[B]): G[F[B]]
}


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



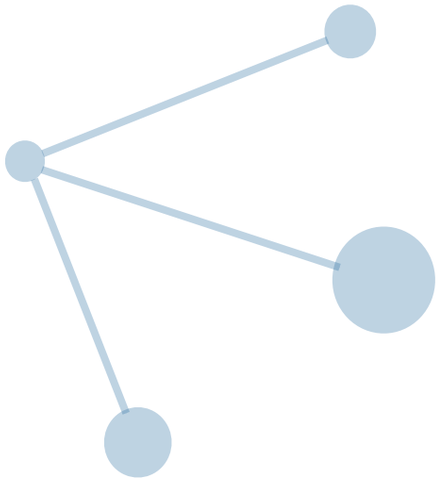
Traversal

```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
                (fa: F[A])(f: 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 form a Monoid
```



Traversal

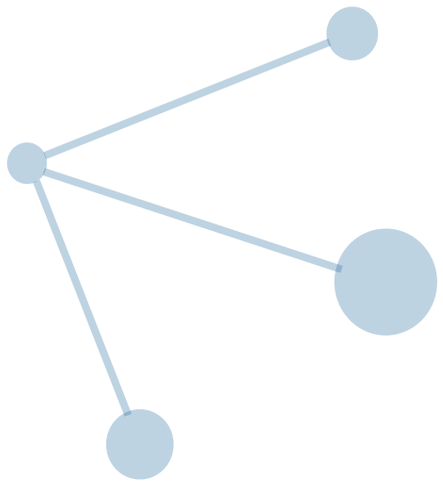
```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
                (fa: F[A])(f: A => G[B]): G[F[B]]
}


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 form a Monoid
    F[A] => (A => Const[Z, B]) => Const[Z, F[B]]
```



Traversal

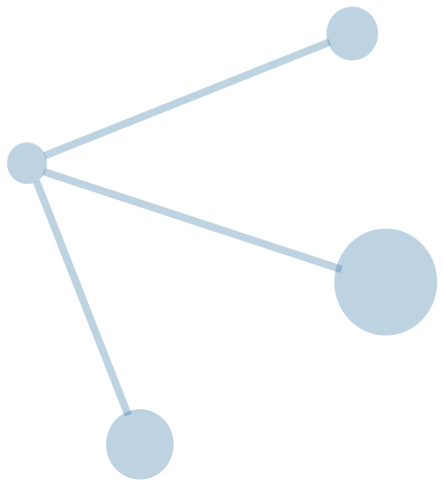
```
trait Traverse[F[_]] { // e.g. F = List, Option, Map
  def traverse[G[_] : Applicative, A, B]
                (fa: F[A])(f: A => G[B]): G[F[B]]
}


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 form a Monoid
    F[A] => (A => Const[Z, B]) => Const[Z, F[B]]
    F[A] => (A => Z) => Z
```

Interested?



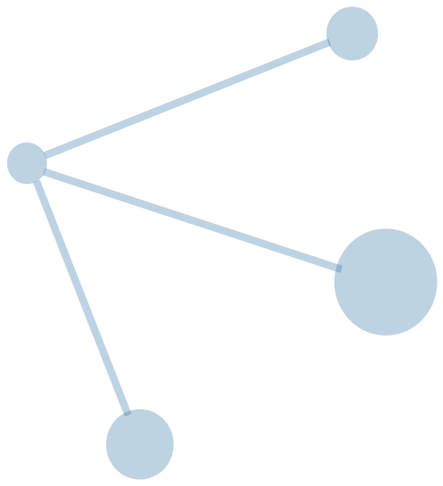
Interested?

- Functional Programming
  - Cats, Scalaz
  - Argonaut, Atto, Doobie, HTTP4S, Monocle, Remotely, Scalaz-stream, Shapeless, Typelevel.org



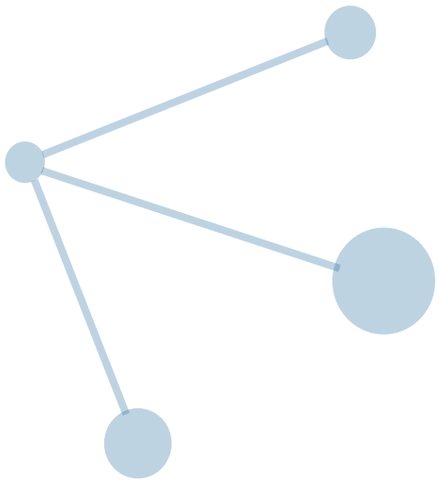
Interested?

- Functional Programming
  - Cats, Scalaz
  - Argonaut, Atto, Doobie, HTTP4S, Monocle, Remotely, Scalaz-stream, Shapeless, Typelevel.org
- Algebra
  - Algebird, Algebra, Breeze, Spire



Interested?

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



EOF