Compositional programming

Rúnar Bjarnason (@runarorama)
Scala Wave 2018, Gdańsk

Category Theory

Category Theory:
the abstract study of
compositionality

Software is ***compositional*** to the extent that we can understand the **whole** by understanding the **parts** and the rules of **composition**.

- A ***compositional*** expression is a nested structure.
- Each subexpression has a ***meaning***.
- The meaning of the whole is ***composed*** of the meanings of the parts.

The composition of the meanings
is the meaning of the composition.

Composability vs *Compositionality*

A Small Example

```scala
val fr = new FileReader("thefile.txt")
val br = new BufferedReader(fr)

var line = br.readLine()

var count = 0

while (line ≠ null) {
  val words = line.split("\\s")
  for (w ← words) {
    count += 1
  }
  line = br.readLine()
}

br.close()

println(count)
```

```
io.linesR("thefile.txt")
   .flatMap(s ⇒ emits(s.split("\\s")))
   .map(_ ⇒ 1)
   .fold(0)(_ + _)
   .to(stdout)
```

```
io.linesR("thefile.txt")
  .flatMap(s ⇒ emits(s.split("\\s")))
  .map(_ ⇒ 1)
  .fold(0)(_ + _)
  .to(stdout)
```

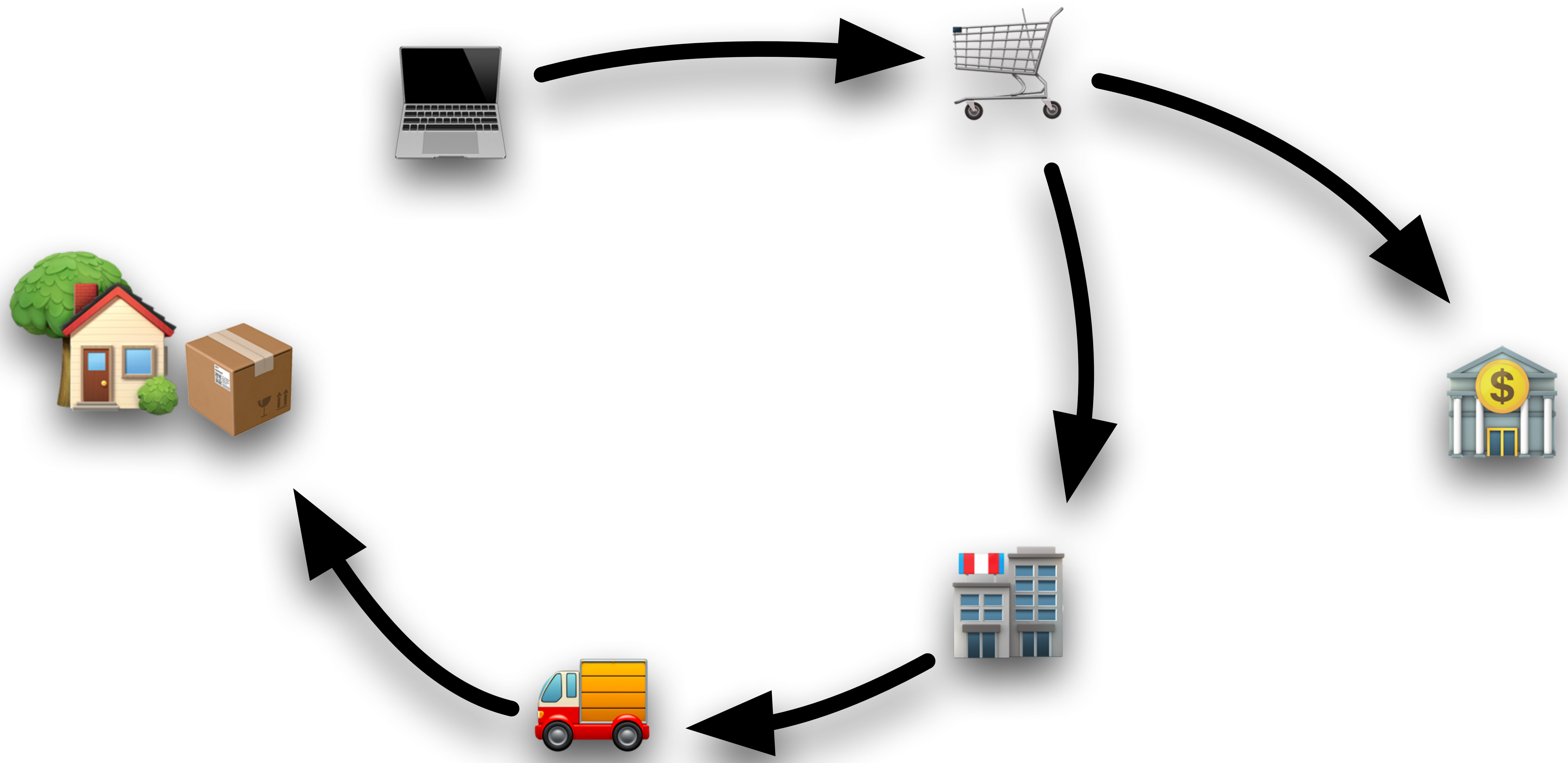
```
io.linesR("thefile.txt")
  .flatMap(s ⇒ emits(s.split("\\s")))
  .map(_ ⇒ 1)
  .fold(0)(_ + _)
  .to(stdout)
```

```
io.linesR("thefile.txt")
  .flatMap(s ⇒ emits(s.split("\\s")))
  .map(_ ⇒ 1)
  .fold(0)(_ + _)
  .to(stdout)
```

```
io.linesR("thefile.txt")
  .flatMap(s ⇒ emits(s.split("\\s")))
  .map(_ ⇒ 1)
  .fold(0)(_ + _)
  .to(stdout)
```

```
io.linesR("thefile.txt")
  .flatMap(s ⇒ emits(s.split("\\s")))
  .map(_ ⇒ 1)
  .fold(0)(_ + _)
  .to(stdout)
```

```
io.linesR("thefile.txt")
   .flatMap(s ⇒ emits(s.split("\\s")))
   .map(_ ⇒ 1)
   .fold(0)(_ + _)
   .to(stdout)
```



```scala
val lines = io.linesR("thefile.txt")
val words = _.flatMap(s ⇒ emits(s.split("\\s")))
val ones = _.map(_ ⇒ 1)
val sum = _.fold(0)(_ + _)
val print = _.to(stdout)

val prg = print(sum(ones(words(lines))))
val f = print
  compose sum
  compose ones
  compose words

val prg = f(lines)
```

Functional programming is really the study of ***compositional software***

Functions are compositional

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\ compose\ f"'] & B \arrow[d, "g"] \\
& C
\end{tikzcd}$$

```
(x:A) => g(f(x))
```

Category

- Objects
- Arrows between objects
- Composition of arrows
    - Which is associative
    - And has an identity

The Scala Category

- Objects: Scala types
- Arrows: Scala functions
- Composition: function composition
  - `f compose g compose h = (x ⇒ f(g(h(x))))`
  - `identity = (x ⇒ x)`

Another Scala Category

- Objects: Scala types
- Arrows: Subtype relationships
- Composition: transitivity

```
A <: B <: C

A <: A
```

```scala
trait Monoid[M] {
  def empty: M
  def append(m1: M, m2: M): M
}
```

Monoid

1. A type
2. An associative binary operation
3. An identity element for that operation

Examples

- `Int` with (`+`, `0`)
- `Int` with (`*`, `1`)
- `Boolean` with (`&&`, `true`)
- `String` with (`++`, `""`)
- `A => A` with (`compose`, `identity[A]`)

A monoid is a category with one object

- Objects: The type `M`
- Arrows: Values of type `M`
- Composition: `append`
- Identity: `empty`

Lorem ipsum dolor sit amet, consectetuer adipiscing elit; Aenean commodo ligula eget dolor! Aenean massa? Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem! Nulla consequat massa quis enim; Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu? In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo; Nullam dictum felis eu pede mollis pretium! Integer tincidunt? Cras dapibus! Vivamus elementum semper nisi; Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus; Phasellus viverra nulla ut metus varius laoreet; Quisque rutrum? Aenean imperdiet; Etiam ultricies nisi vel augue! Curabitur ullamcorper ultricies nisi; Nam eget dui; Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor? Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

Lorem ipsum dolor sit amet, consectetuer adipiscing elit; Aenean commodo ligula eget dolor! Aenean massa? Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem! Nulla consequat massa quis enim; Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu? In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo; Nullam dictum felis eu pede mollis pretium! Integer tincidunt? Cras dapibus! Vivamus elementum semper nisi; Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus; Phasellus viverra nulla ut metus varius laoreet; Quisque rutrum? Aenean imperdiet; Etiam ultricies nisi vel augue! Curabitur ullamcorper ultricies nisi; Nam eget dui; Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor? Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

```
append(wc(s1), wc(s2)) == wc(s1 ++ s2)
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit; Aenean commodo ligula eget dolor! Aenean massa? Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem! Nulla consequat massa quis enim; Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu? In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo; Nullam dictum felis eu pede mollis pretium! Integer tincidunt? Cras dapibus! Vivamus elementum semper nisi; Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus; Phasellus viverra nulla ut metus varius laoreet; Quisque rutrum? Aenean imperdiet; Etiam ultricies nisi vel augue! Curabitur ullamcorper ultricies nisi; Nam eget dui; Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor? Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

`WC("Lorem", 39, "")`

massa quis enim; Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu? In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo; Nullam dictum felis eu pede mollis pretium! Integer tincidunt? Cras dapibus! Vivamus elementum semper nisi; Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus; Phasellus viverra nulla ut metus varius laoreet; Quisque rutrum? Aenean imperdiet; Etiam ultricies nisi vel augue! Curabitur ullamcorper ultricies nisi; Nam eget dui; Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor? Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

```
WC("Lorem", 39, "")

WC("massa", 49, "vi")
```

tae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus; Phasellus viverra nulla ut metus varius laoreet; Quisque rutrum? Aenean imperdiet; Etiam ultricies nisi vel augue! Curabitur ullamcorper ultricies nisi; Nam eget dui; Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor? Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

```
WC("Lorem", 39, "")

WC("massa", 49, "vi")

WC("tae", 55, "pena")
```

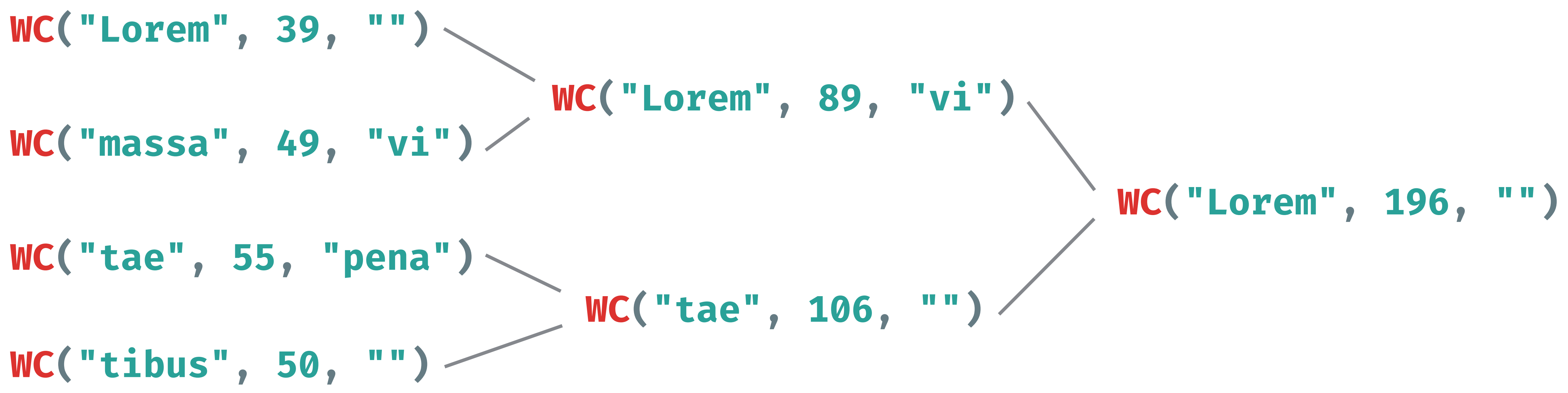
tibus et magnis dis parturient montes, nascetur ridiculus mus! Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem? Nulla consequat massa quis enim! Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu! In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo! Nullam dictum felis eu pede mollis pretium!

```
WC("Lorem", 39, "")

WC("massa", 49, "vi")

WC("tae", 55, "pena")

WC("tibus", 50, "")
```


WC("Lorem", 39, "")
WC("massa", 49, "vi")
WC("tae", 55, "pena")
WC("tibus", 50, "")
WC("Lorem", 89, "vi")
WC("tae", 106, "")
WC("Lorem", 196, "")

Compositional reasoning

```
append(wc(s1), wc(s2)) == wc(s1 ++ s2)

append(wc(s), wc("")) == wc(s)
append(wc(""), wc(s)) == wc(s)
```

Monoid homomorphism

```
append(wc(s1), wc(s2)) == wc(s1 ++ s2)

append(wc(s), wc("")) == wc(s)
append(wc(""), wc(s)) == wc(s)
```

Homomorphism

Monoid homomorphism

```
s1.length + s2.length == (s1 ++ s2).length

"".length == 0
```

Category Theory is really the study of ***homomorphisms***

The category $\mathbf{\mathit{Mon}}$ of monoids

- Objects: monoids
- Arrows: monoid homomorphisms
- Composition: function composition

The category $\mathcal{C}at$ of categories

- Objects: categories
- Arrows: ***category homomorphisms***
- Composition: ?

The category $\mathcal{C}at$ of categories

- Objects: categories
- Arrows: ***functors***
- Composition: functor composition

Functor

$F: C \to D$

- Takes every object in C to an object in D
- Takes every arrow in C to an arrow in D
- Composition and identity are preserved

```scala
trait Functor[F[_]] {
  def map[A,B](h: A ⇒ B): F[A] ⇒ F[B]
}
```

```
trait Functor[F[_]] {
  def map[A,B](f: A ⇒ B): F[A] ⇒ F[B]
}

map(f compose g) = map(f) compose map(g)
map(identity) = identity
```

```scala
implicit val optionF = new Functor[Option] {
  def map[A,B](f: A ⇒ B): Option[A] ⇒ Option[B] =
    { case Some(a) ⇒ Some(f(a))
      case None ⇒ None
    }
}
```

$$f : A \Rightarrow B$$

If f has a side effect, composition is impossible.

```
f: A ⇒ Option[B]
```

Effect: the function **f** might not return any **B**

```
f: A ⇒ Option[B]
g: B ⇒ Option[C]
```

Problem:

```
f andThen g
```

```
f: A ⇒ Option[B]
g: B ⇒ Option[C]
```

Solution:

```
f andThen (_ flatMap g)
```

```
f: A ⇒ Option[B]
g: B ⇒ Option[C]

f >=> g : A ⇒ Option[C]
```

Kleisli Category

- Objects: Scala types
- An arrow from **A** to **B** is a function of type
  `A ⇒ Option[B]`
- Composition: Kleisli composition
  - `f >=> g >=> h =`
    `(x => h(x) flatMap g flatMap f)`
  - `identity(x) = Some(x)`

Kleisli Category

- Objects: types `A`, `B`, `F[T]` etc.
- An arrow from `A` to `B` is a function of type `A => M[B]` for some functor `M`.
- Composition: Kleisli composition (`flatMap`)
- Identity: `unit: A => M[A]`

```
trait Monad[M[_]] {
  def flatMap[A,B](h: A ⇒ M[B]): M[A] ⇒ M[B]
  def unit[A]: A ⇒ M[A]
}

flatMap(f ⟹ g) = flatMap(f) compose flatMap(g)
flatMap(unit) = identity
```

Things that prevent compositionality

Side effects

```scala
class Cafe {
  def buyCoffee(cc: CreditCard): Coffee = {
    val cup = new Coffee()
    cc.charge(cup.price)
    cup
  }
}
```

Side effects

```scala
class Cafe {
  def buyCoffee(cc: CreditCard): (Coffee, Charge) = {
    val cup = new Coffee()
    (cup, new Charge(cc, cup.price))
  }
}
```

Side effects

```
map(f compose g) = map(f) compose map(g)


f(x) + f(y)      = f(x + y)
```

Connected sequences

The meaning of the whole is not a combination the meaning of the parts

```
MOV    AH, 01h
INT    21
```

Dependencies

The meaning of one part depends on the meaning of some or all the other parts.

Rhe1	**d4**
Nd5	**Nbd5**
ed5	**Qd6**
Rd4	**cd4**

Leaky abstractions

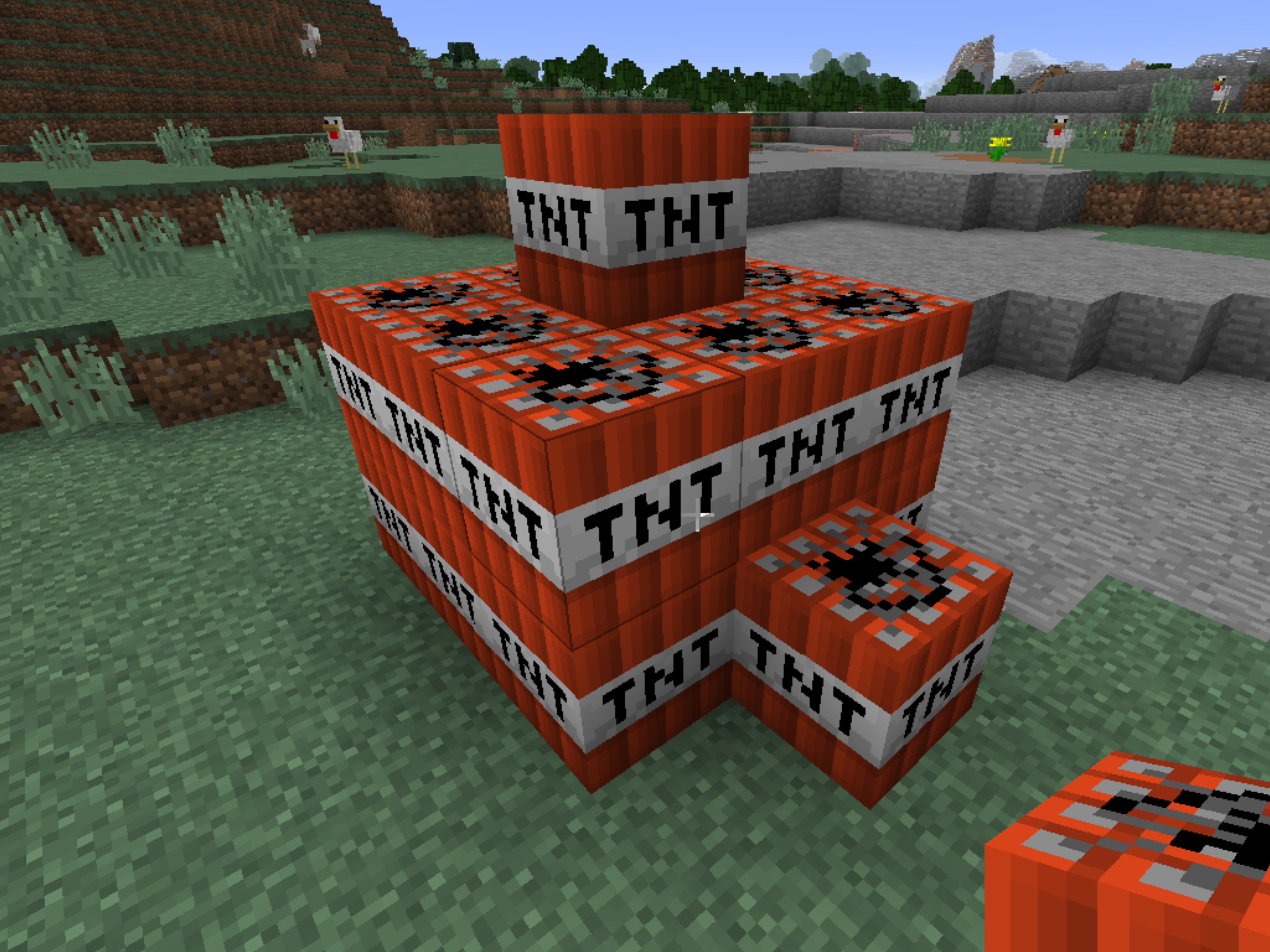
```
val query = """
  select a, b, c
  from foo
  where a = ?
  """
```

Leaky abstractions

```
val query = """
  select a, b, c, d
  from foo
  where a = ?
  and b = ?
  """
```

Entropy and Perplexity

TNT
TNT


TNT

Without compositionality, language is literally meaningless.

Without compositionality, software is literally meaningless.

Language without composition

- Totally nonuniform
- Absolutely unambiguous
- Maximally perplexed
- Literally meaningless

Big Wins

Productivity

Understand things we've never seen before by understanding the the components.

Productivity

- Break a problem into parts.
- Solve the parts with simple programs.
- Compose the solution from the smaller programs.

Compositionality lets us reason about really big systems and ideas.

Systematicity

If we understand `f(x)` and `g(y)`,
we also understand `f(y)` and `g(x)`.

Systematicity

If we can solve problems with ***p***, ***q***, and ***r*** individually, we can solve any problem whose solution is any combination of of ***p***, ***q***, and ***r***.

Pragmatics:
Compositionality
works.

Pragmatics:
Compositionality is the ***only*** thing that works.

Æsthetics:
Compositional software
is delightful.

Functional Programming in scala

Paul Chiusano
Rúnar Bjarnason

Foreword by Martin Odersky

MANNING

Write delightful,
meaningful,
compositional code.