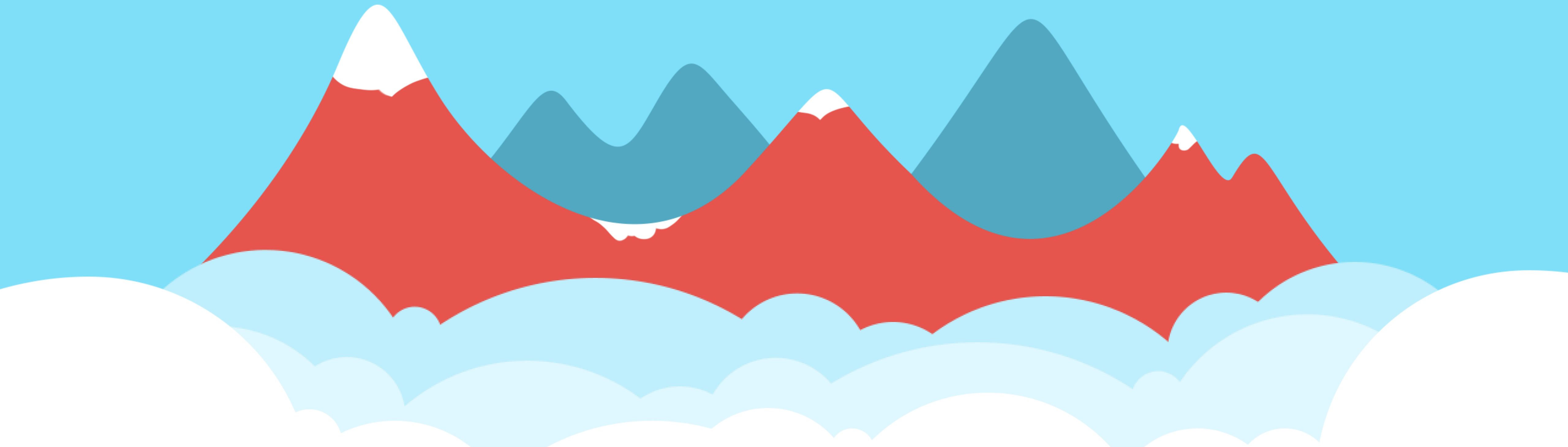
Scala **Enthusiasts** BS

Philipp Wille

Beyond Scala‘s Standard Library



OO or Functional Programming?

- Martin Odersky:

“Systems should be composed from modules. Modules should be simple parts that can be combined in many ways to give interesting results.”

About Scala: The Simple Parts

Modular Programming

- atomic module

```
object Greeting {
  def apply: String = "Hello"
}
```

- templates to create modules

```
class Greet(name: String) {
  def apply: String = "Hello "+name+"!"
}

object Greeting extends Greet("Martin")
```

Modular Programming

- atomic module

```
val Greeting: () => String = {
  () => "Hello"
}
```

- templates to create modules

```
val Greet: String => () => String = {
  case name => () => "Hello "+name+"!"
}

val Greeting: () => String = Greet("Martin")
```

Modular Programming

- mixable slices of behavior

```
trait Politeness {
  val bePolite = " How are you today?"
}

class Greet(name: String) extends Politeness {
  def apply = "Hello "+name+"!"+bePolite
}

val greeting = new Greet("Martin")
```

Modular Programming

- Martin Odersky:

> “Modular Programming is putting the focus on how modules can be combined, not so much what they do.”

About Scala: The Simple Parts

- in this talk we will focus on modules

Scalaz

- “Scalaz is a Scala library for functional programming. It provides purely functional data structures to complement those from the Scala Standard Library.”

http://github.com/scalaz/scalaz

- provided modules: i.a. data structures and methods

Scalaz Memos

- speed up function calls by *memoization*
- think: *caching*
- **example:**

```
val fibonacci: Int => Int = {
  case 0 => 0
  case 1 => 1
  case n => fibonacci(n - 2) + fibonacci(n - 1)
}
```

- **problem:** recomputation of fibonacci values

Scalaz Memos

- speed up function calls by *memoization*
- think: *caching*
- **solution:**

```
val fibonacci: Int => Int = Memo.mutableHashMapMemo {
  case 0 => 0
  case 1 => 1
  case n => fibonacci(n - 2) + fibonacci(n - 1)
}
```

- once a value is computed it is *cached* in a mutable HashMap and will be reused

Scalaz Ordering

- monadic way for defining orderings for types
- defines types LT, GT, and EQ
- defines functions ?|?, lt, gt, lte, gte, min, and max

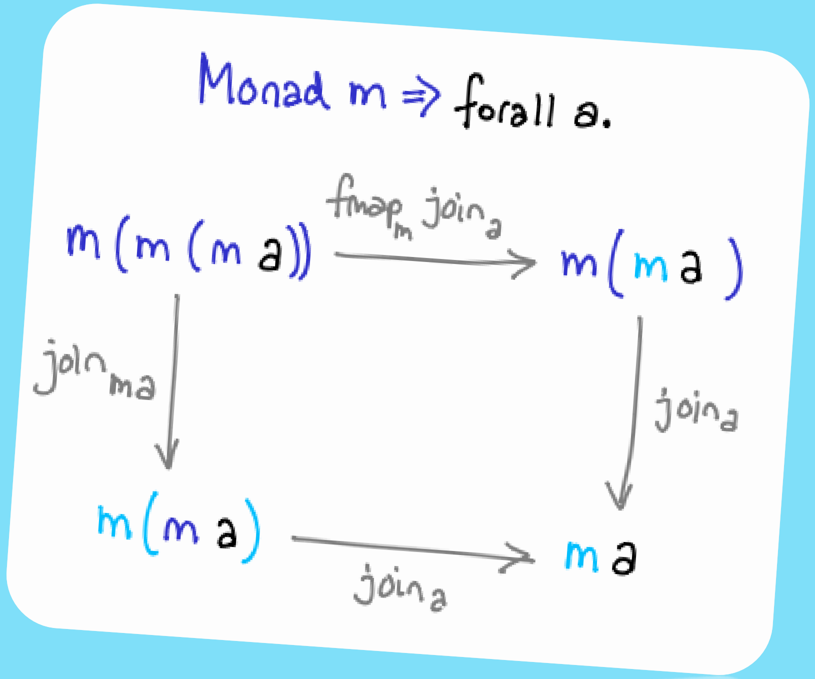
```
1.0 ?|? 2.0 // scalaz.Ordering = LT
1.0 gt 2.0  // Boolean = false

def compare(a: String, b: String): Ordering =
  (a.length ?|? b.length) |+| (a ?|? b)

compare("viktor", "martins") // scalaz.Ordering = LT
compare("viktor", "martin") // scalaz.Ordering = GT
```

Scalaz – additional features

- Type classes
  - Functor
  - Applicative
  - Monad
  - Zipper
  - Lenses
  - Free Monad
  - State Monad



Monad m ⇒ forall a.

$$\begin{tikzcd}
m(m(m\,a)) \arrow[r, "fmap_m\ join_a"] \arrow[d, "join_{ma}"'] & m(m\,a) \arrow[d, "join_a"] \\
m(m\,a) \arrow[r, "join_a"'] & m\,a
\end{tikzcd}$$

Shapeless

- “Shapeless is a type class and dependent type based generic programming library for Scala.”

http://github.com/milessabin/shapeless

- provided modules: i.a. data structures and additional methods for Standard Library types

Shapeless

- Scala's Standard Library does not provide any collection methods for tuples (for a reason)
- Shapeless adds support for them

```
val tuple = ("dog", true)

tuple.head     // String = dog
tuple.drop(1)  // (Boolean,) = (true,)
tuple.split(1) // ((String,), (Boolean,)) = ((dog,), (true,))
23 +: tuple    // (Int, String, Boolean) = (23, dog, true)
```

Shapeless

- Scala’s Standard Library does not support polymorphic function values
- Shapeless adds them

```scala
val tuple = ("dog", true)

object AsList extends (Id ~> List) {
  def apply[A](a: A) = List(a)
}

tuple.map(elem => AsList(elem))
// (List[String], List[Boolean]) = (List(dog), List(true))
```

Shapeless – additional features

- type specific polymorphic function values
- heterogenous lists (including map over polymorphic function values)
- generic representation of case classes

Akka

- “Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM.”

http://akka.io

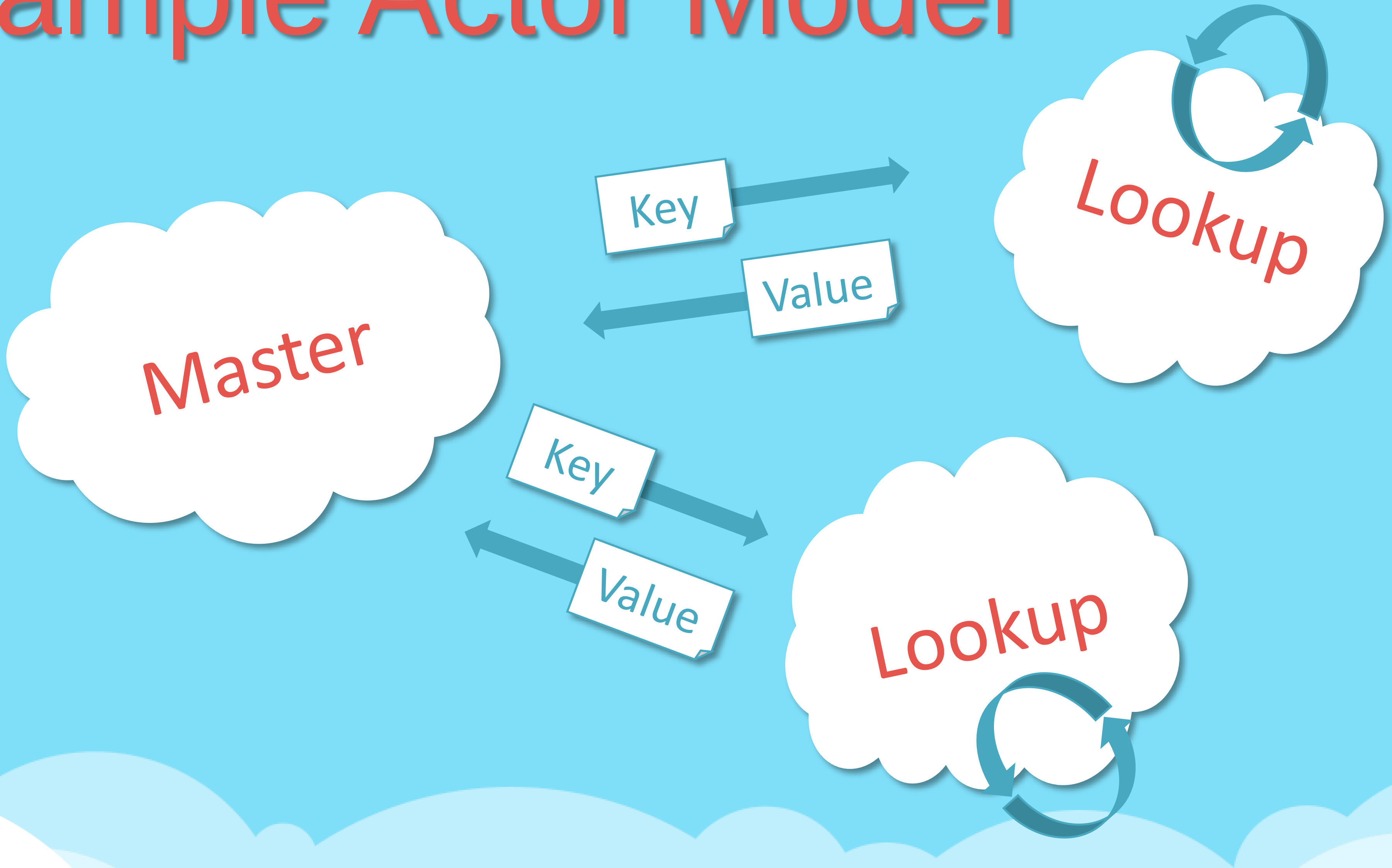
- provided modules: i.a. actors, data structures

Akka Actors

- concurrent entities (think: each actor runs in its own thread)
- send and receive messages (think: they act like mailboxes)
- they have a message queue and process one message at a time

- difficulties: unordered arrivals and no guaranteed delivery

Sample Actor Model



Actor Model

```
class Lookup extends Actor {
  val data = Map("Martin" -> "Odersky")
  def receive = {
    case Key(key) =>
      val value = data.getOrElse(key, "")
      sender() ! Value(value)
  }
}
```

Actor Model

```
class Master extends Actor {
  override def preStart() {
    val viktor = context.actorOf(Props[Lookup], "lookup-viktor")
    val martin = context.actorOf(Props[Lookup], "lookup-martin")
    viktor ! Key("Viktor")
    martin ! Key("Martin")
  }
  val results = ListBuffer[String]()
  def receive = {
    case Value(value) => results += value
  }
}
```

Actor Model – Local

- deploying the actors on a local machine

```
val local = ConfigFactory.load("local")
val system = ActorSystem("Master", local)
system.actorOf(Props[Master], "master")
```

Actor Model – Remote

- deploying the actors on a remote machine

```
val master = ConfigFactory.load("remote-master")
val system = ActorSystem("Master", master)
system.actorOf(Props[Master], "master")

val worker = ConfigFactory.load("remote-worker")
ActorSystem("Worker", worker)
```

Actor Model – Remote

- configuring the remote machine’s address

```
akka.actor.deployment {
  "/master/*" {
    remote = "akka.tcp://Worker@127.0.0.1:13371"
  }
}
```

Actor Model – Cluster

- deploying the actors in a cluster

```
val master = ConfigFactory.load("cluster-master")
val system = ActorSystem("Master", master)
Cluster(system).registerOnMemberUp {
  system.actorOf(Props[Master], "master")
}

val worker = ConfigFactory.load("cluster-worker")
val system = ActorSystem("Worker", worker)
system.actorOf(Props[Lookup], "lookup")
```

Actor Model – Cluster

- configuring the cluster

```
akka.actor.deployment {
  "/master/*" {
    router = adaptive-group
    metrics-selector = mix
    routes.paths = ["/user/lookup"]
    cluster {
      enabled = on
      use-role = worker
}}}
```

Akka – additional features

- Akka Persistence
- Akka Http (former Spray.io)
- Akka Futures (now in Scala‘s Standard Library)
- Akka Streams
- Akka Finite State Machine

Sbt (Scala/Simple Build Tool)

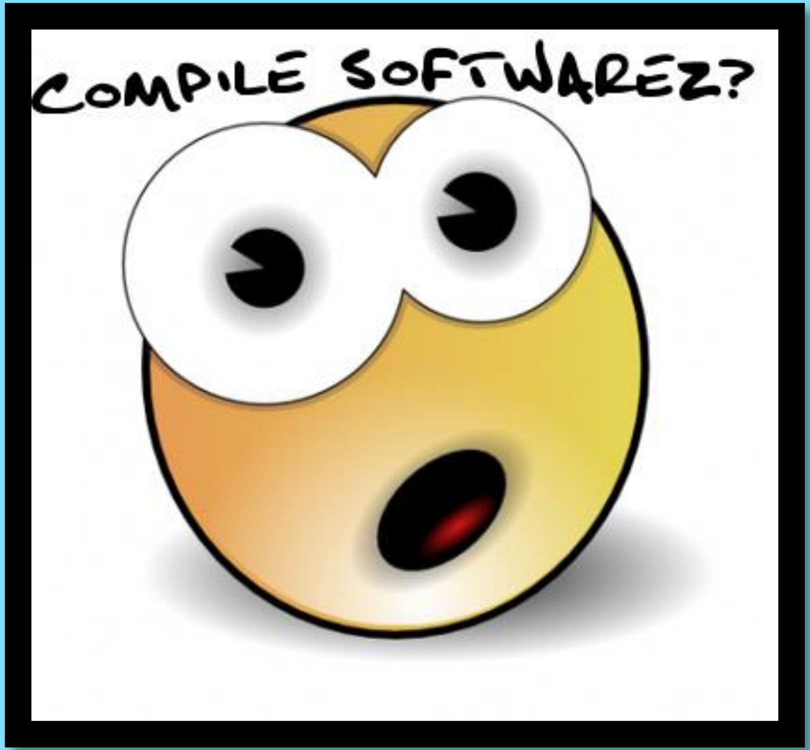
- interactive build tool
- compiles both, Scala and Java
- uses Maven dependencies
- compile, run, package your code
- multi-project builds

Sample build.sbt definition

```
name := "root"
version := "1.0"
scalaVersion := "2.11.1"
libraryDependencies ++= Seq(
  "com.typesafe.akka" % "akka-actor_2.11" % "2.3.4"
)
lazy val hello = ProjectRef(file("../hello"), "hello")
lazy val world = ProjectRef(file("../world"), "world")
lazy val root = project.in(file(".")).dependsOn(hello, world)
```

Sbt – additional features

- user defined tasks
- huge number of plugins (e.g. assembly)
- interactive Scala console
- code deployment
- cross Scala versions building



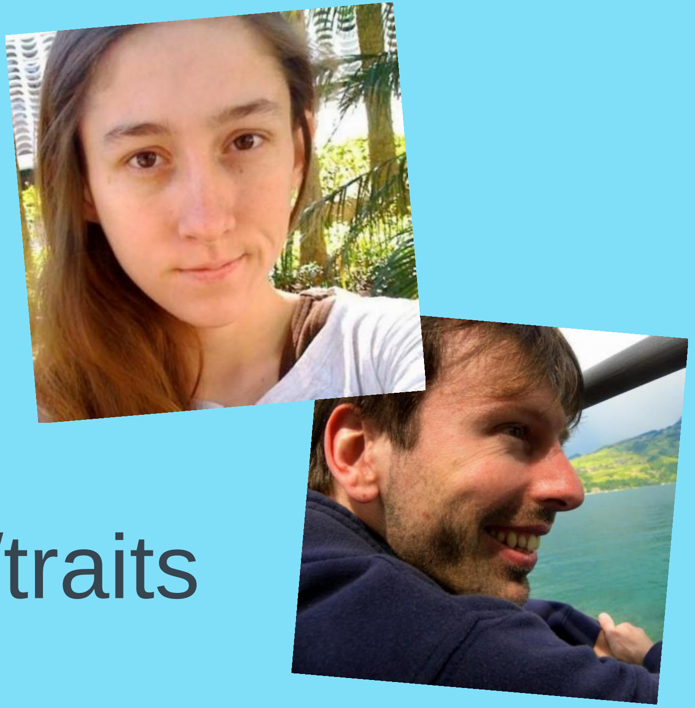
Conclusions

- Modular Programming means thinking about software design first
  - different types of modules
  - different ways to combine them
  - multiple ways of solving a problem
- Modular Programming will make you a better software engineer
  - the more modules you know the better
  - the more combinations you know the better
  - scale your knowledge by scaling your language

Questions?

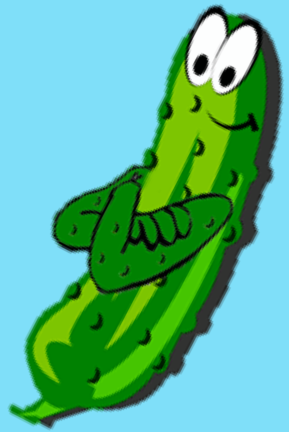
Thank you for your attention

Pickling



- automatic serialization framework
  - works out-of-the-box
  - no need for implementing interfaces/traits
- typesafety
  - compile-time errors for serialization
  - uses Scala Macros
  - runtime errors for deserialization

Pickling



```
import scala.pickling._
import json._

val pckl = List(1, 2, 3, 4).pickle
val list = pckl.unpickle[List[Int]]

case class Cloud(shape: String)
case class Sky(clouds: Set[Cloud])
val pckl = Sky(Set(Cloud("Dog"), Cloud("Banana"))).pickle
val sky = pckl.unpickle[Sky]
```

```
{ "tpe": "List[Int]",
  "elems": [1, 2, 3, 4] }

{ "tpe": "Sky",
  "clouds": {
    "tpe": "Set[Cloud]",
    "elems": [
    { "tpe": "Cloud",
      "shape": "Dog" },
    { "tpe": "Cloud",
      "shape": "Banana" }] }}
```