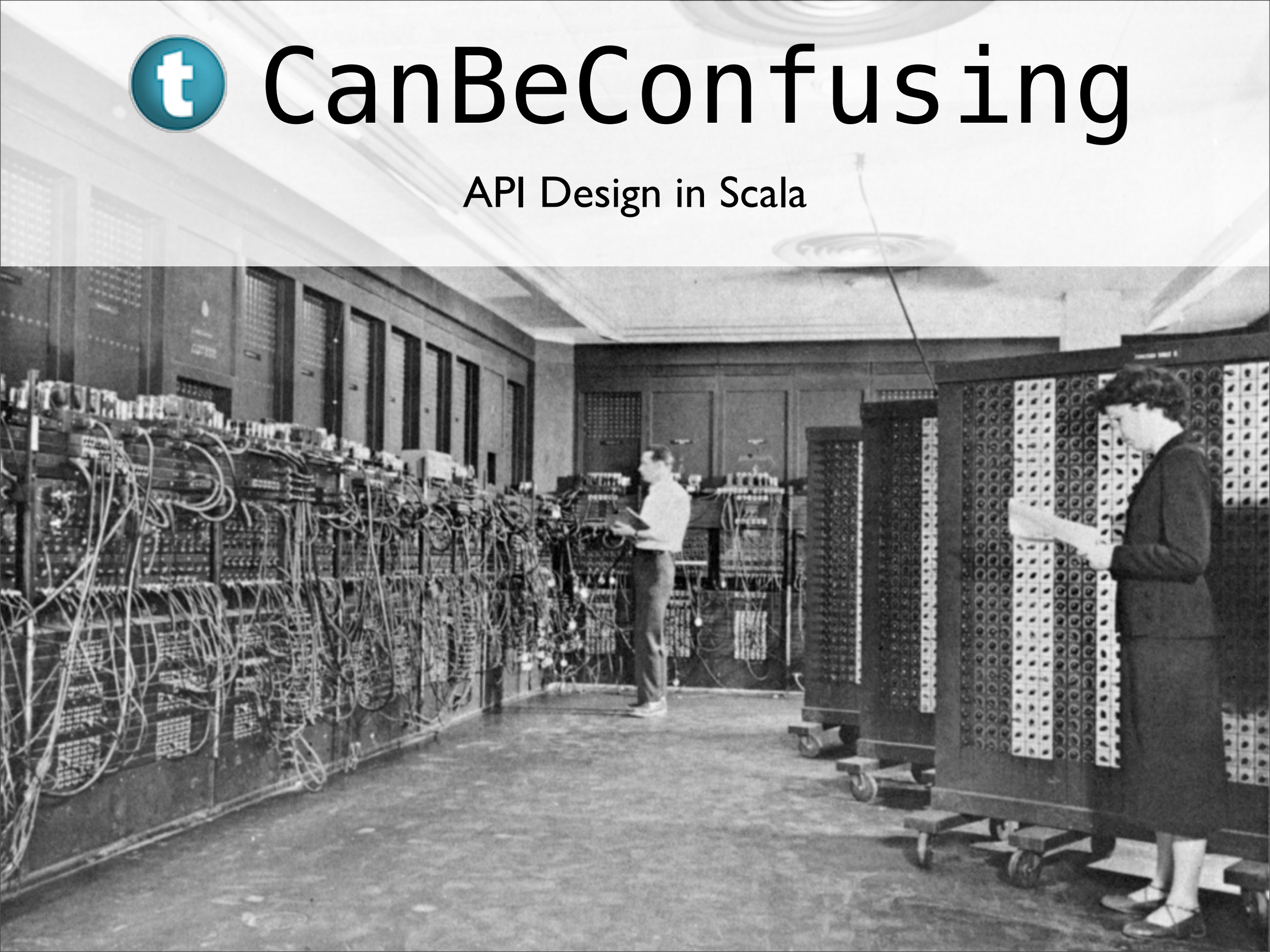

CanBeConfusing

API Design in Scala

Outline

- Operators and Functions
- Implicits
- Java Interop
- Traits

Operator Overloading

Operator Overloading

- **Don’t do it!**

Operator Overloading

- **Don’t do it!**
  - No really, don’t do it!
    - *(yes I’m serious)*

def $#!+

- Some operator meanings are well-known

def $#!+

- Some operator meanings are well-known
  - Seriously, what does ! ! ! mean?

def $#!+

- Some operator meanings are well-known
  - Seriously, what does ! ! ! mean?
- Explanation == #fail

def $#!+

- Some operator meanings are well-known
  - Seriously, what does ! ! ! mean?
- Explanation == #fail
- Most sane operators are mathematical

def $#!+

- Some operator meanings are well-known
  - Seriously, what does ! ! ! mean?
- Explanation == #fail
- Most sane operators are mathematical
  - Domains with well-defined symbols

Idiotic Idiom Alert

Apparently, the ~ operator means
“combine in sequence”

Functions

- This ain't Haskell

Functions

- This ain't Haskell
- Arguments are in the inverse order
- (from most specific to least)

```
foldl :: (a -> b -> a) -> a -> [b] -> a
```

```
foldl :: (a -> b -> a) -> a -> [b] -> a
```

```
foldLeft: A => ((A, B) => A) => A
```

Functions

- This ain't Haskell
- Arguments are in the inverse order
  - (from most specific to least)
- Function params need to be curried

Functions

- This ain't Haskell
- Arguments are in the inverse order
  - (from most specific to least)
- Function params need to be curried
- Side-effecting methods *need* parentheses

Implicits

- **Conversions**
  - Triggered with *any* implicit function value
- **Typeclasses**
  - Everything else that uses `implicit`

Implicit Horror

- 21st Century Monkey Patching

Implicit Horror

- 21st Century Monkey Patching
- So easy to produce spaghetti code

Implicit Horror

- 21st Century Monkey Patching
- So easy to produce spaghetti code
- *Agonize* over every implicit conversion

Implicit Horror

- 21st Century Monkey Patching
- So easy to produce spaghetti code
- *Agonize* over every implicit conversion
- Protip: you need them less than you think!

Jorge's Laws

Jorge's Laws

Jorge's Laws

Thou shalt onl  conversions for one of *two* (2)

I. Pimping members onto an existing type

II. "Fixing" a broken type hierarchy

Implicit Resolution

1. Current scope
2. Explicit imports
3. Wildcard imports
4. Same scope in other files

Implicit Resolution

5. Companion objects of a type
6. Companion objects of type parameters
7. Outer objects for nested types
8. *Other dimensions…*

Typeclasses

- `mkOps`
- Configuration
- DSL guards
- Static scope control

mkOps

```
def sum[A : Numeric](xs: Seq[A]) = {
  val tc = implicitly[Numeric[A]]
  import tc.mkNumericOps

  (tc.zero /: xs) { _ + _ }
}
```

Configuration

```
implicit val config =
  DbConf("jdbc:hsqldb:mem", "user", "pass")
```

```
val query = SELECT (*) FROM "attendees"

for (row <- query.execute) {
  ...
}
```

DSL Guards

```
implicit def pimpInt(i: Int)(implicit g: Guard) = new {
  def foo = ...
  def bar = ...
}
```

Static Scope Control

```
trait Ref[A] {
  def apply()(implicit txn: Transaction) = ...
  def update(a: A)(implicit txn: Transaction) = ...
}
```

```
val x = Ref(42)

val result = atomic { implicit txn =>
  x() = x() + 1
  x() % 2
}
```

Java Interop

- Understand name mangling
- Traits *barely* work, try to avoid them
- Beware type signatures involving `Nothing`
- Accessor / mutator patterns

```scala
import scala.collection.JavaConverters._

class Useless {
  val boys = List("Daniel", "Chris", "Joseph")
  val girls = List("Renee", "Bethany", "Grace")

  def getBoys = boys.asJava
  def getGirls = girls.asJava
}
```

Traits

- Beware evaluation order!

Traits

- Beware evaluation order!
- Patterns
  - Mixins
  - Modules
  - Abstract Classes

Traits

- Beware evaluation order!
- Patterns
  - Mixins
  - Modules
  - Abstract Classes
- Binary Compatibility

```scala
trait A {
  val x: Int
  val y = x + 10
}

class B extends A {
  val x = 42
}

println(new B().y)          // any guesses?
```

```
trait A {
  val x: Int
  lazy val y = x + 10
}

class B extends A {
  lazy val x = 42
}

println(new B().y)          // => 52
```

Patterns

- **Mixin** – Container for utilities
  - (think: Lift's `Helpers`)

Patterns

- **Mixin** – Container for utilities
  - (think: Lift's `Helpers`)
- **Module** – Cake Pattern

Patterns

- **Mixin** – Container for utilities
  - (think: Lift's `Helpers`)
- **Module** – Cake Pattern
- **Abstract Class**
  - You almost never need constructor args!

Binary Compatibility

- Traits get compiled into…

Binary Compatibility

- Traits get compiled into…
  - An interface

Binary Compatibility

- Traits get compiled into…
  - An interface
  - Inner class with static methods

Binary Compatibility

- Traits get compiled into…
  - An interface
  - Inner class with static methods
- Linking works fine!

Binary Compatibility

- Traits get compiled into…
  - An interface
  - Inner class with static methods
- Linking works fine!
- Cross-references to new APIs don't

Conclusion

- Don't use operator overloading
- Be (much) more careful with implicits
  - Jorge's Laws
- Lazy vals in traits

Questions?

TODO: insert humorous Google Image result here…