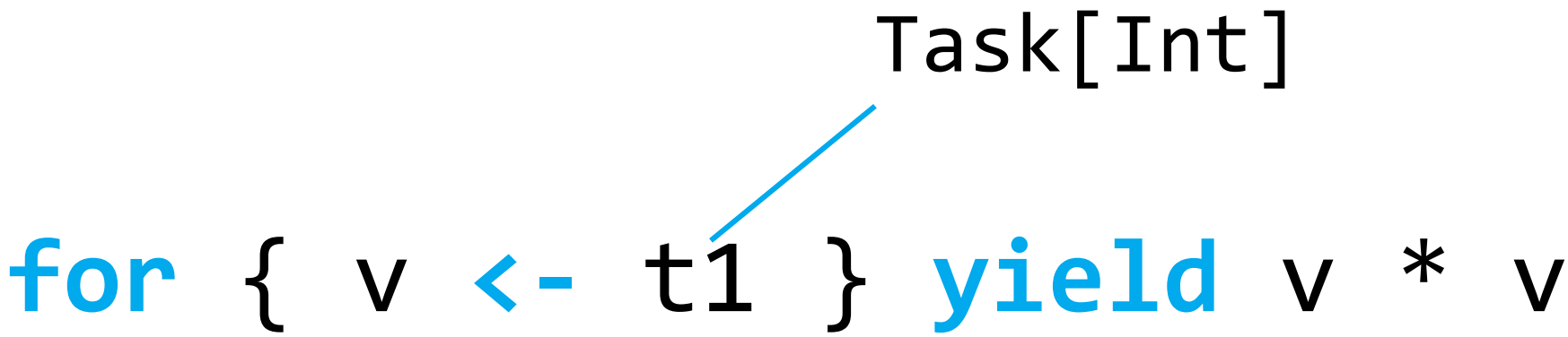
Direct-style Monad Comprehensions

Monadic super-powers through less syntax

Christopher Hodapp, Rocket Lawyer / @clhodapp
Jan Christopher Vogt, x.ai / @cvogt

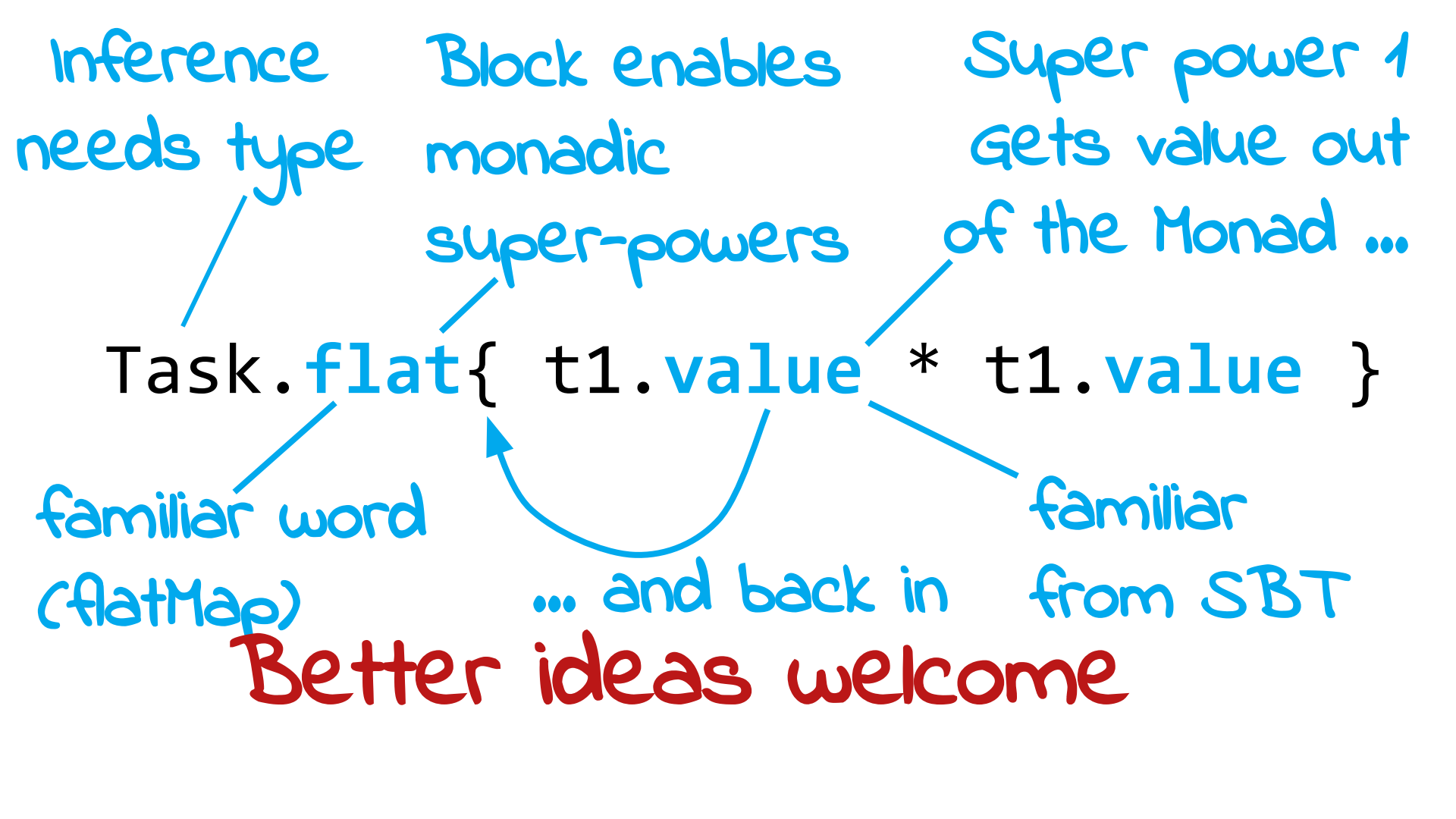
TYPELEVEL SUMMIT MARCH 2ND 2016 PHILADELPHIA

https://github.com/cvogt/flow-comprehensions


Task[Int]
for { v <- t1 } yield v * v

Classic Comprehensions use specialized syntax

Alternative:
“direct-style” syntax


Inference needs type
Block enables monadic super-powers
Super power 1 Gets value out of the Monad ...
Task.flat{ t1.value * t1.value }
familiar word (flatMap)
... and back in
familiar from SBT
Better ideas welcome

Live demo

The Typelevel Mission

“Let the Scala Compiler Work For You”

Yes!

Yes!

Yes!

“Let the Scala Compiler Work For You”

Yes!

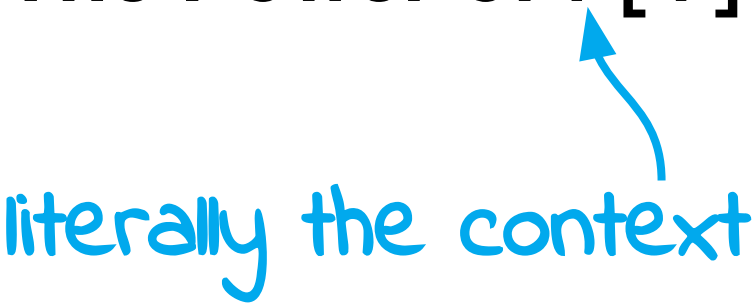
Yes!

Yes!

Profit From Types

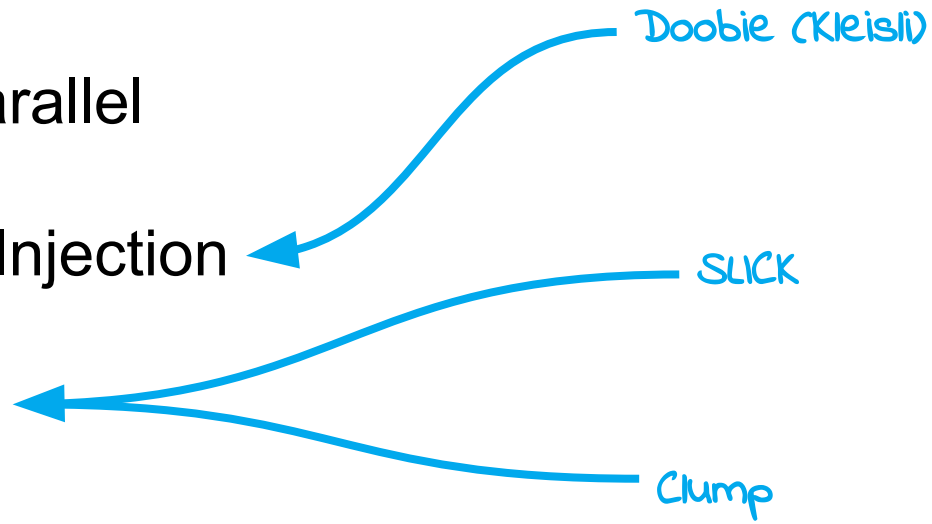
Make Context a Type

The Power of `F[T]`



Super-Powerful Libraries

- Async and Parallel
- Dependency Injection
- Optimization
- Data Passing



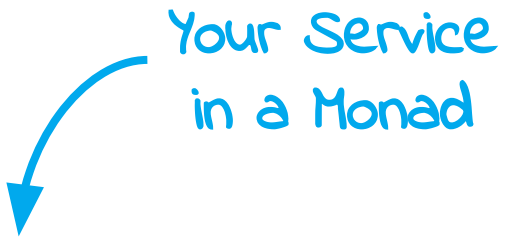
Look, It’s Possible but not Practical

- Half your methods have a `for`
- Things get ugly
- It’s not Scala anymore
- Your team just wants to write blocking code

Demo Code

At the End of the Day...

Make asynchronous code as easy as blocking code



Make it practical to use contexts for more things (e.g. db transactions)

Make it low-cost to drop a context around some code

What’s next

- rock-solid macros
- more of the language
- monad stacks (Emm?)
- context operations, e.g. sortBy
  (aka comprehensive comprehensions)

Isn’t This Like.....

- sbt .value
  - Specific to sbt
  - applicative (not monadic)
- scala async
  - Specific to scala futures
- effectful (closest)
  - Scalaz dependency
  - Trusts macro api too much
  - Postfix !

See Also

- Akka dataflow
- Scala workflow by @aztek
- Computation expressions by @jedesah
- daemonad by @mandubian
- each by Thought Works
- Haskell codo

Rocket Laywer

x.ai

Twitter, Github:
@cvogt @clhodapp

https://github.com/cvogt/flow-comprehensions

https://github.com/cvogt/flow-comprehensions-example

Who are we?

Christopher Hodapp

Rocket Lawyer

legal made simple

Comprehensive Comprehensions

Jan Christopher Vogt

Slick, Martin's lab, Compossible records

Better Syntax! All this + more:

```
(for{
  x <- future1
  _ = logger.debug("x was "+x)
  y <- future2
  z = x * y
  _ <- cloudwatch.record(x * y)
} yield (x, z)).sortBy{
  case (x,_) => x
}.map(_._2)
```

```
List.flat{ c =>
  val x = future.value
  logger.debug("x was "+x)
  val z = x * future2.value
  cloudwatch.record(z)
  c!(_.sortBy(_ => x))
  z
}
```

Super-power comprehension syntax!