

Frege

purely functional programming
on the JVM

GOTO Berlin 2015

Dierk König
canoo



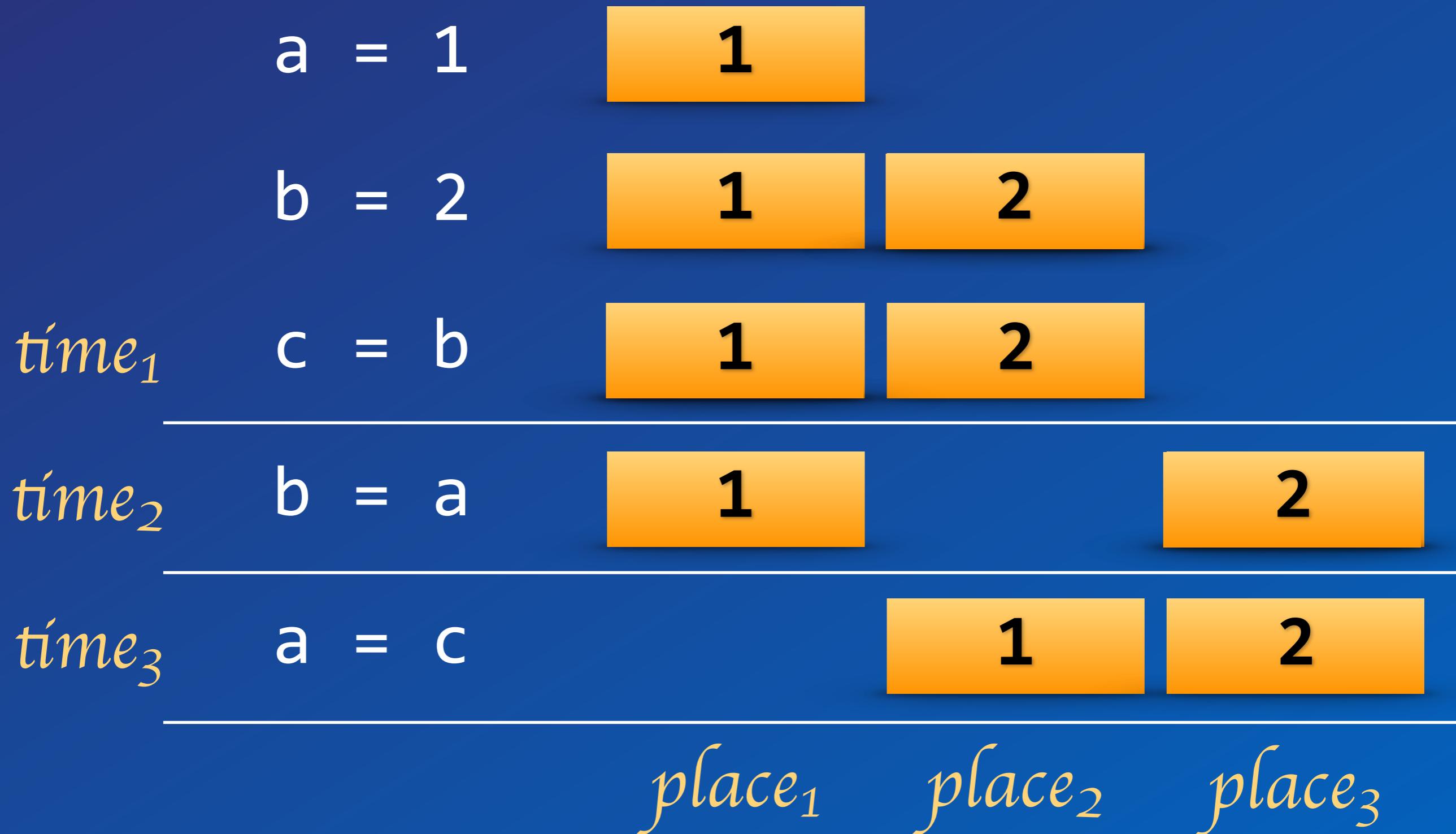
mittie

canoo

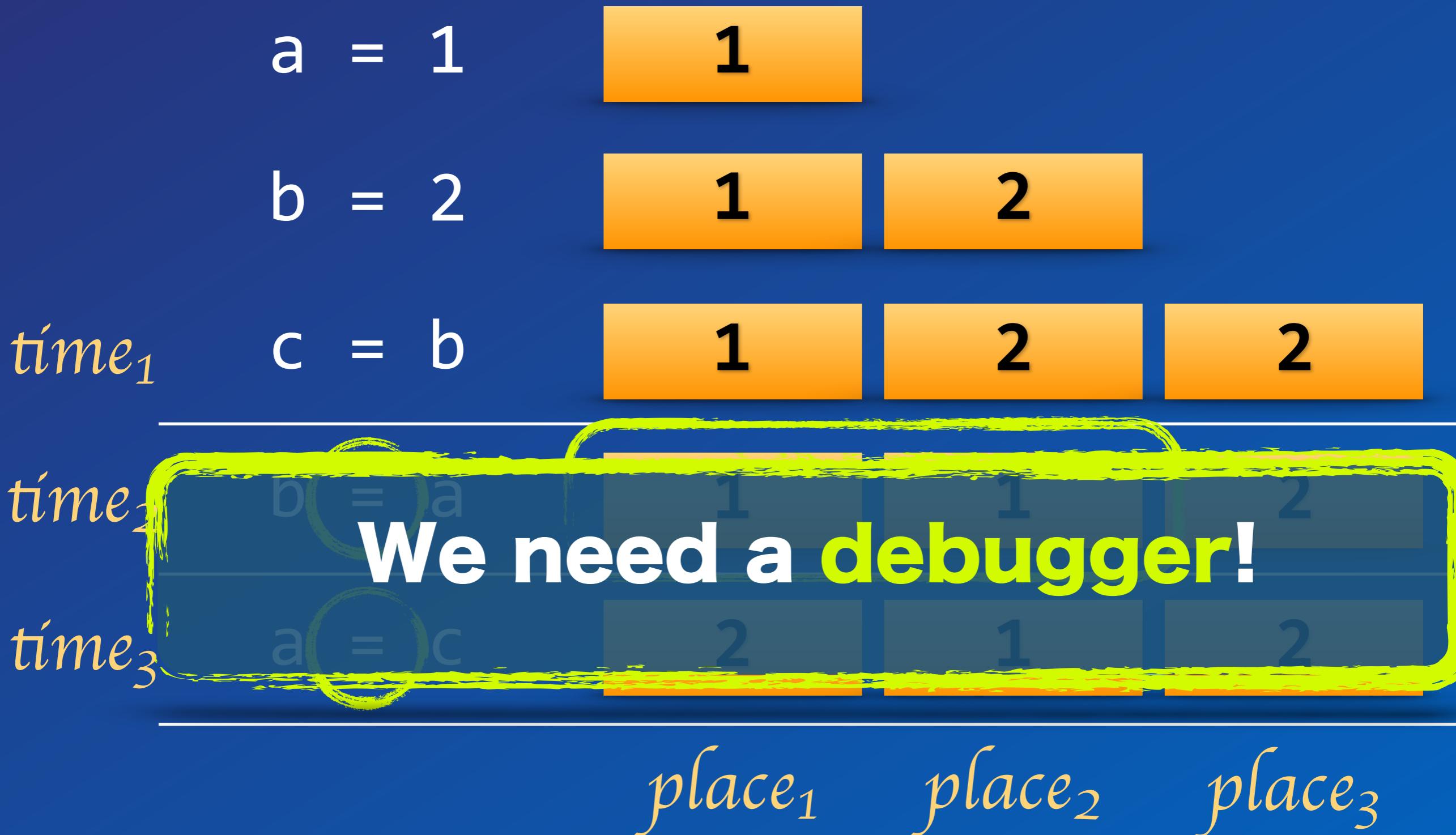


Dreaming of code

Why do we care?



Operational Reasoning



Using functions

a = 1

1

b = 2

1

2

Using functions

a = 1

1

b = 2

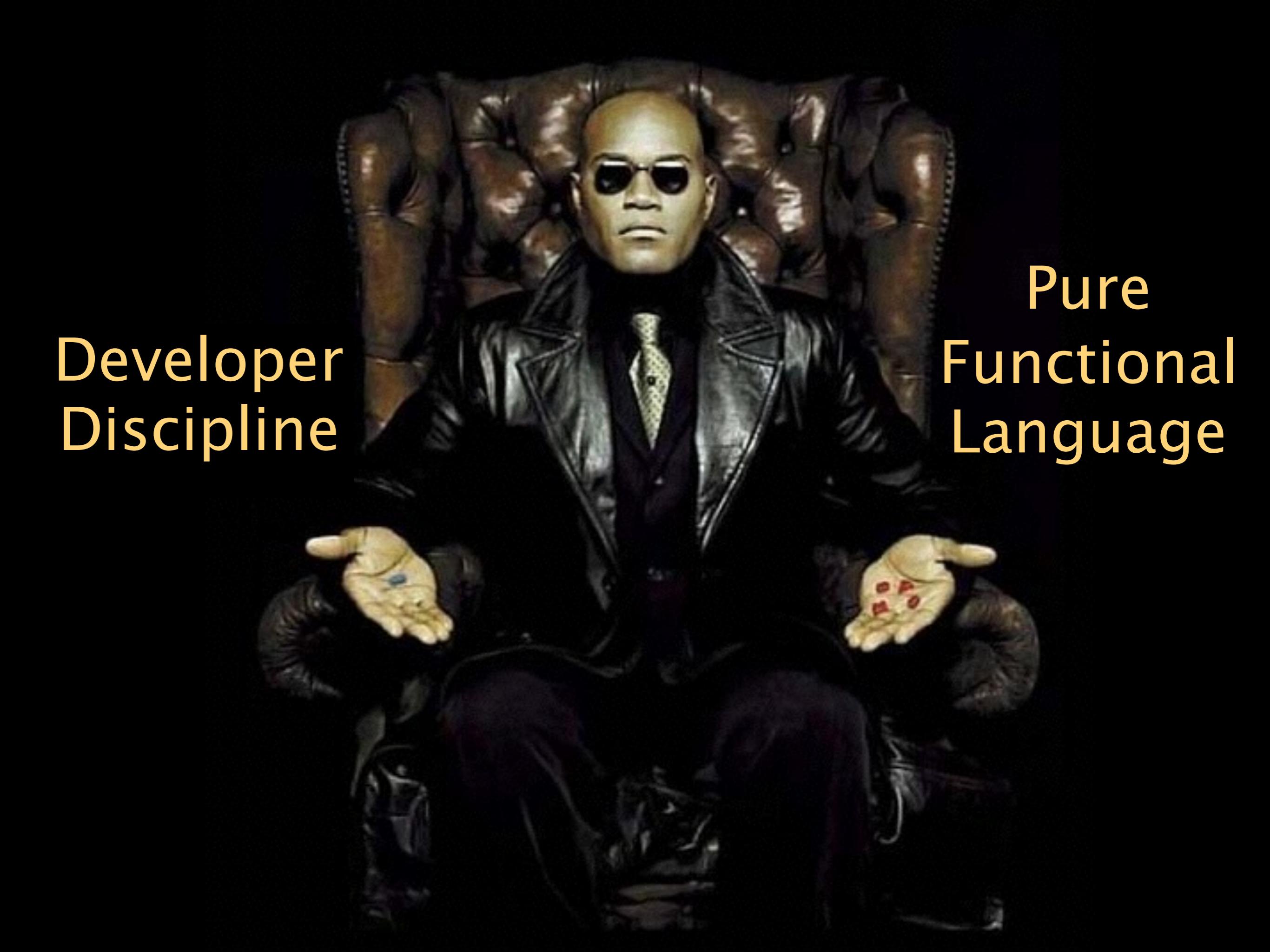
1 2

2

1

swap(a,b) = (b,a)

Let's just program
without
assignments or
statements!

A dramatic, low-key lighting photograph of a man in a dark suit and sunglasses, looking directly at the viewer. He is framed by several large, gloved hands, some wearing leather bracers, which are positioned behind his head and in front of him, creating a sense of confinement or control. The background is dark and moody.

Developer
Discipline

Pure
Functional
Language

Online REPL
try.frege-lang.org

Define a Function

```
freges> times a b = a * b
```

```
freges> times 2 3
```

6

```
freges> :type times
```

```
Num α => α -> α -> α
```

Define a Function

```
freges> times a b = a * b
```

no types declared

```
freges> (times 2) 3
```

6

function appl.
left associative

no comma

```
freges> :type times
```

```
Num α => α ->(α -> α)
```

typeclass
constraint

only 1
parameter!

return type is
a function!

thumb: „two params
of same numeric type
returning that type“

Reference a Function

```
freges> twotimes = times 2
```

```
freges> twotimes 3
```

6

```
freges> :t twotimes
```

Int -> Int

Reference a Function

```
freges> twotimes x = times 2 x
```

No
second
arg!

```
freges> twotimes 3
```

6

```
freges> :t twotimes
```

„Currying“, „schönfinkeling“,
or „partial function
application“.

Concept invented by
Gottlob Frege.

```
Int -> Int
```

inferred types
are more specific

Function Composition

```
frege> twotimes (threetimes 2)
```

12

```
frege> sixtimes = twotimes . threetimes
```

```
frege> sixtimes 2
```

```
frege> :t sixtimes
```

Int -> Int

Function Composition

```
frege> twotimes (threetimes 2) f(g(x))
```

12

```
frege> sixtimes = twotimes . threetimes
```

```
frege> sixtimes 2 (f ∘ g) x
```

```
frege> :t sixtimes
```

Int -> Int

Pure Functions

Java

```
T foo(Pair<T,U> p) {...}
```

What could
possibly happen?

Frege

```
foo :: ( $\alpha$ ,  $\beta$ ) ->  $\alpha$ 
```

What could
possibly happen?

Pure Functions

Java

```
T foo(Pair<T,U> p) {...}
```

Frege

```
foo :: (α,β) -> α
```

Everything!

*State changes,
file or db access,
missile launch,...*

a is returned

Pure Functions

- can be cached (memoized)
- can be evaluated lazily
- can be evaluated in advance
- can be evaluated concurrently
- can be eliminated
in common subexpressions
- can be optimized

Is my method pure?

The screenshot shows a Java call stack with many frames from the Spring framework. A blue rounded rectangle highlights the bottom portion of the stack, which contains several recursive calls to `AbstractAutowireCapableBeanFactory.createBean`. This indicates a potential performance issue or infinite loop if the conditions for creating new beans are not properly handled.

```
org.hibernate.ejb.Ejb3Configuration.addClassesToSessionFactory(Map)
  org.hibernate.ejb.Ejb3Configuration.configure(Properties, Map)
    org.hibernate.ejb.Ejb3Configuration.configure(PersistenceUnitInfo, Map)
      org.hibernate.ejb.HibernatePersistence.createContainerEntityManagerFactory(PersistenceUnitInfo, Map)
        org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean.createNativeEntityManagerFactory()
          org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.afterPropertiesSet()
            org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(String, Object, RootBeanDefinition)
              org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(String, Object, RootBeanDefinition)
                org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(String, RootBeanDefinition, Object[])
                  org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$1.run()
                    java.security.AccessController.doPrivileged(PrivilegedAction, AccessControlContext)
                      org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(String, RootBeanDefinition, Object[])
                        org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject()
                          org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(String, ObjectFactory)
                            org.springframework.beans.factory.support.AbstractBeanFactory doGetBean(String, Class, Object[], boolean)
                              org.springframework.beans.factory.support.AbstractBeanFactory.getBean(String, Class, Object[])
                                org.springframework.beans.factory.support.AbstractBeanFactory.getBean(String)
                                  org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveReference(Object, RuntimeBeanReference)
                                    org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveValueIfNecessary(Object, Object)
                                      org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.applyPropertyValues(String, BeanDefinition, BeanWrapper, PropertyValues)
                                        org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(String, AbstractBeanDefinition, BeanWrapper)
                                          org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(String, RootBeanDefinition, Object[])
                                            org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$1.run()
                                              java.security.AccessController.doPrivileged(PrivilegedAction, AccessControlContext)
                                                org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(String, RootBeanDefinition, Object[])
                                                  org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveInnerBean(Object, String, BeanDefinition)
                                                    org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveValueIfNecessary(Object, Object)
                                                      org.springframework.beans.factory.support.ConstructorResolver.resolveConstructorArguments(String, RootBeanDefinition, BeanWrapper, ConstructorArgumentValues, BeanDefinition)
                                                        org.springframework.beans.factory.support.ConstructorResolver.instantiateUsingFactoryMethod(String, RootBeanDefinition, Object[])
                                                          org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.instantiateUsingFactoryMethod(String, RootBeanDefinition, Object[])
                                                            org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBeanInstance(String, RootBeanDefinition, Object[])
                                                              org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$1.createBean(String, RootBeanDefinition, Object[])
                                                                org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveReference(Object, RuntimeBeanReference)
                                                                  org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveValueIfNecessary(Object, Object)
                                                                    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.applyPropertyValues(String, BeanDefinition, BeanWrapper, PropertyValues)
                                                                      org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(String, AbstractBeanDefinition, BeanWrapper)
                                                                        org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(String, RootBeanDefinition, Object[])

```

Let the **type system** find out!

Java Interoperability

Do not mix
OO and FP,

combine them!

Java -> Frege

Frege compiles Haskell to Java source and byte code.

Just call that.

You can get help by using the `:java` command in the REPL.

Frege -> Java

```
pure native encode java.net.URLEncoder.encode :: String -> String  
encode "Dierk König"
```

even Java can be pure

```
native millis java.lang.System.currentTimeMillis :: () -> I0 Long  
millis ()  
millis ()  
past = millis () - 1000
```

This is a key distinction between Frege and
other JVM languages!

Does not compile!

Frege

allows calling Java
but never unprotected!

is explicit about effects
just like Haskell

Type System

Global type inference

More safety and less work
for the programmer

You don't need to specify any types at all!
But sometimes you do anyway...

Keep the mess out!

Mutable
I/O

Mutable

Pure Computation

Pure Computation

Mutable

Pure Computation

Keep the mess out!



Ok, these are Monads. Be brave. Think of them as contexts that the type system propagates and makes un-escapable.

Fizzbuzz

<http://c2.com/cgi/wiki?FizzBuzzTest>

[https://dierk.gitbooks.io/fregegoodness/
chapter 8 „FizzBuzz“](https://dierk.gitbooks.io/fregegoodness/chapter-8-„FizzBuzz“)

Fizzbuzz Imperative

```
public class FizzBuzz{  
    public static void main(String[] args){  
        for(int i= 1; i <= 100; i++){  
            if(i % 15 == 0{  
                System.out.println(„FizzBuzz”);  
            }else if(i % 3 == 0){  
                System.out.println("Fizz");  
            }else if(i % 5 == 0){  
                System.out.println("Buzz");  
            }else{  
                System.out.println(i);  
            } } } }
```

Fizzbuzz Logical

```
fizzes      = cycle    [ "", "", "fizz"]
buzzes      = cycle    [ "", "", "", "", "buzz"]
pattern     = zipWith (+) fizzes buzzes
numbers    = map       show [1..]
fizzbuzz   = zipWith max pattern numbers

main _     = for (take 100 fizzbuzz) println
```

Fizzbuzz Comparison

	Imperative	Logical
Conditionals	4	0
Operators	7	1
Nesting level	3	0
Sequencing	sensitive	transparent
Maintainability	---	+
Incremental development	-	+++

Unique in Frege

Global type inference
requires a purely functional language
(only expressions and parametric polymorphism)

Purity by default
effects are explicit in the type system

Laziness by default

Values are always immutable

Guarantees extend into Java calls

Why Frege

Robustness under parallel execution

Robustness under composition

Robustness under increments

Robustness under refactoring

Enables local and equational reasoning

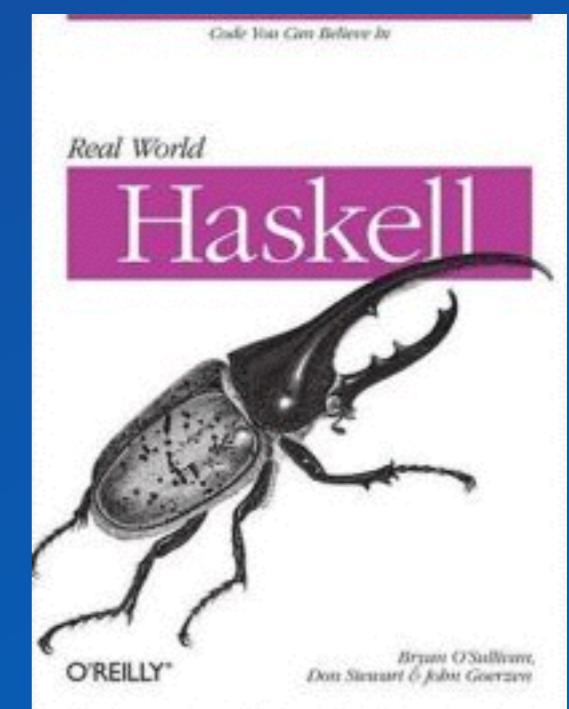
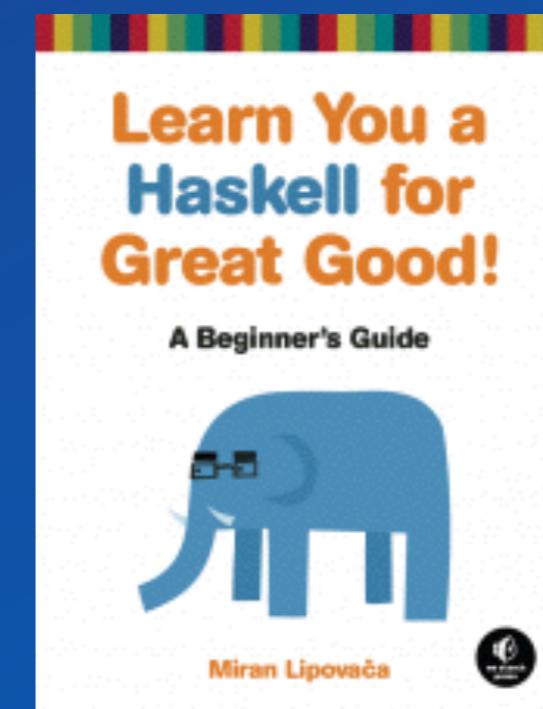
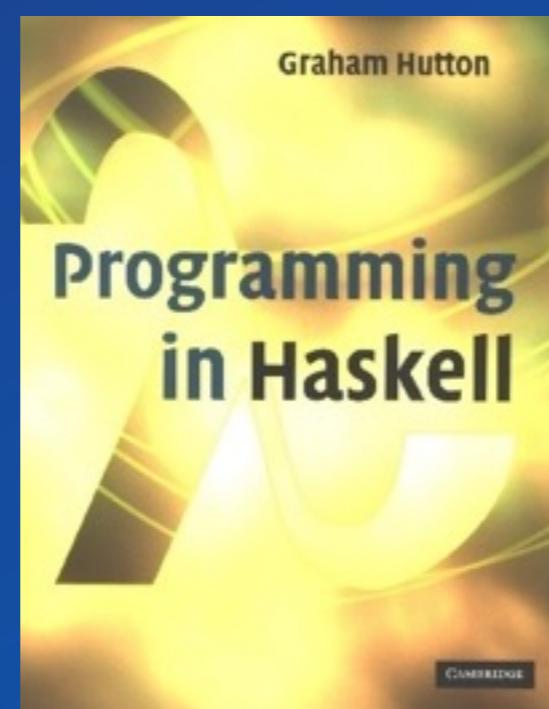
Best way to learn FP

Why Frege

it is just a pleasure to work with

How?

<http://www.frege-lang.org>
@fregelang
stackoverflow „frege“ tag
edX FP101 MOOC



Dierk König
canoo



mittie

canoo

Please give feedback!



FGA

- Language level is Haskell Report 2010.
- Yes, performance is roughly ~Java.
- Yes, the compiler is reasonably fast.
- Yes, we have an Eclipse Plugin.
- Yes, Maven/Gradle/etc. integration.
- Yes, we have HAMT (aka HashMap).
- Yes, we have QuickCheck (+shrinking)
- No, but STM is in the works.