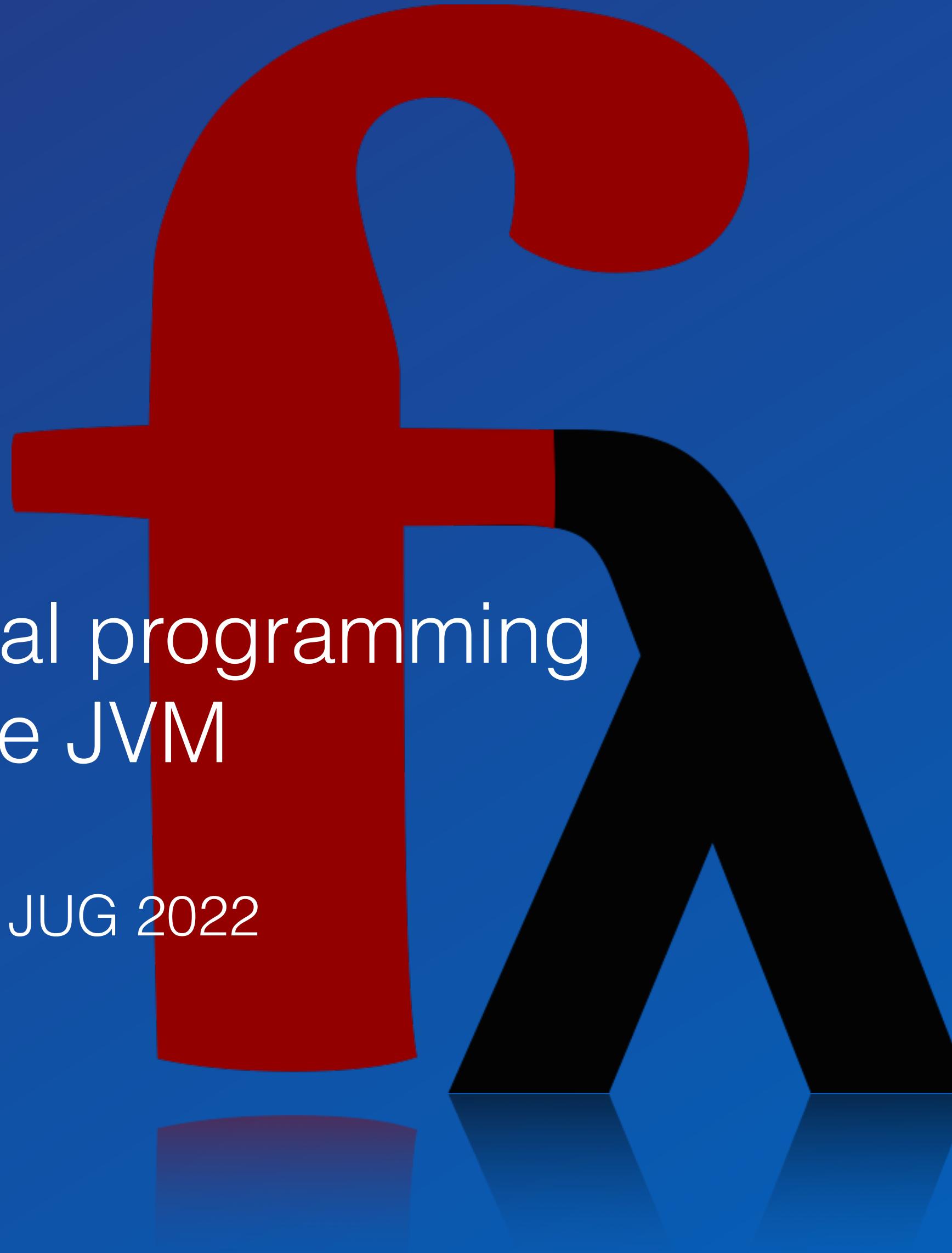


Frege

purely functional programming
on the JVM

St. Louis JUG 2022





Prof. Dierk König

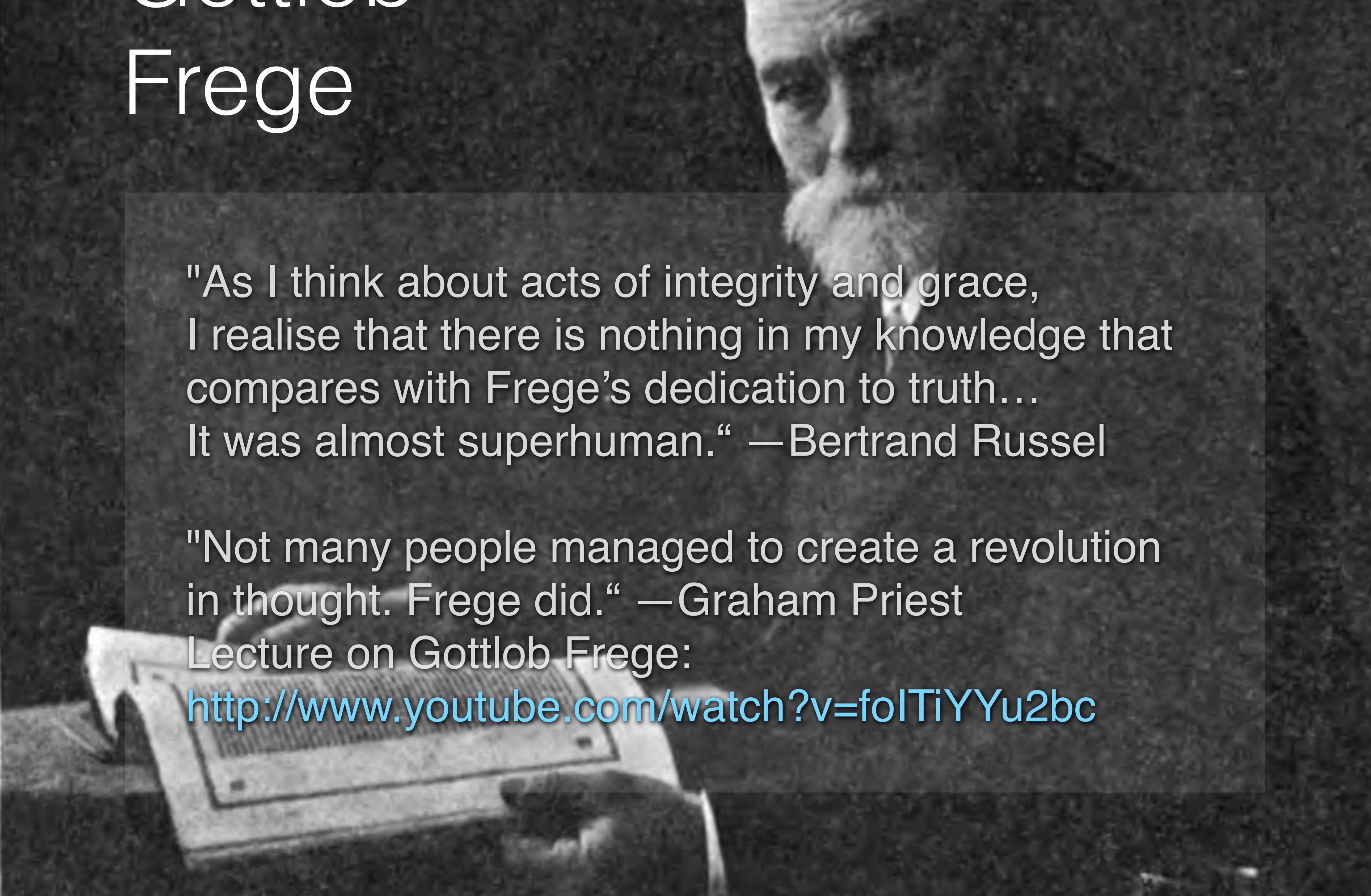
Professor at FHNW

Fellow at Karakun



@mittie

Gottlob Frege

A black and white portrait of Gottlob Frege, an elderly man with a high forehead and receding hairline, wearing a dark suit and white shirt.

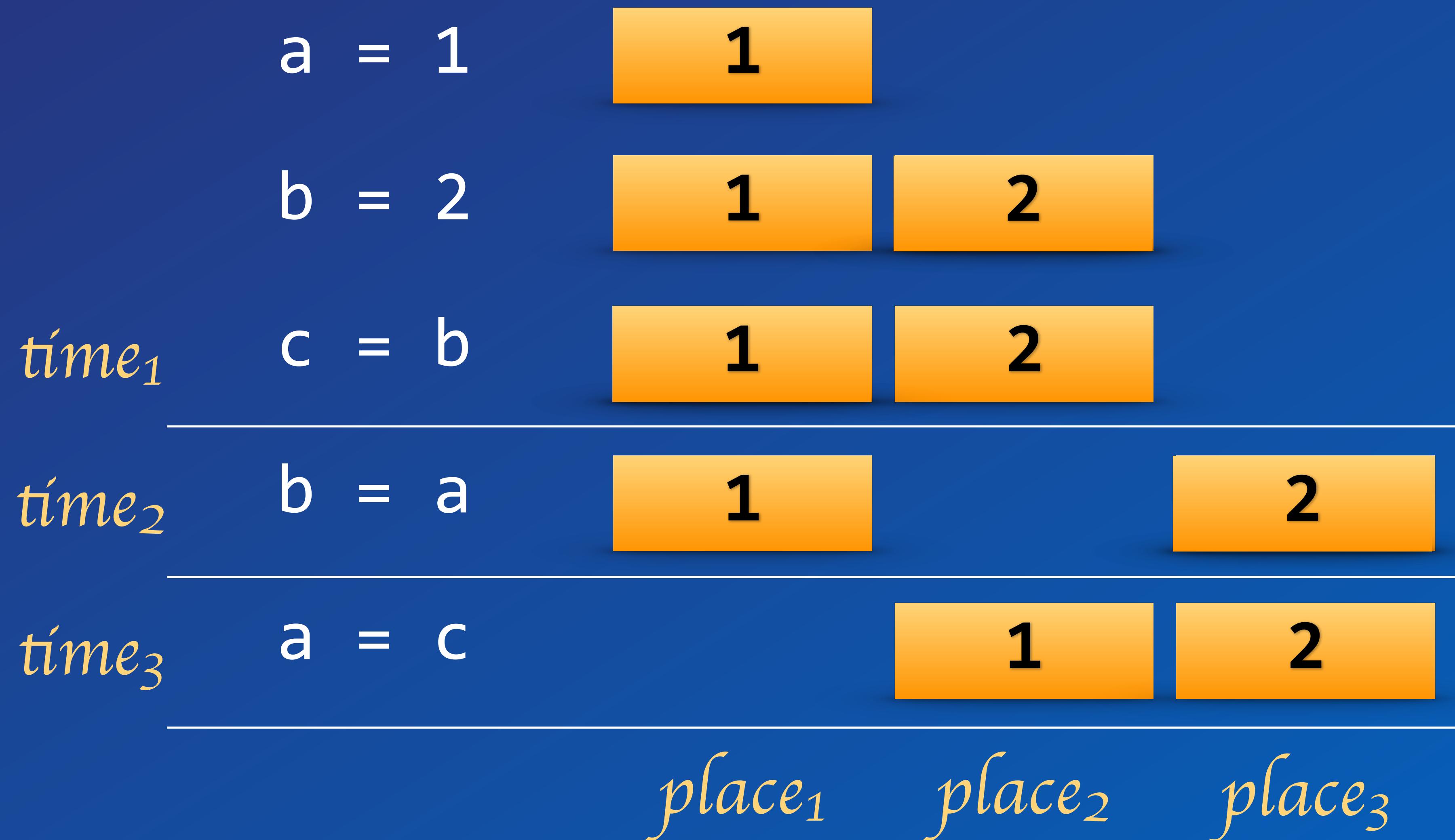
"As I think about acts of integrity and grace,
I realise that there is nothing in my knowledge that
compares with Frege's dedication to truth...
It was almost superhuman." — Bertrand Russell

"Not many people managed to create a revolution
in thought. Frege did." — Graham Priest
Lecture on Gottlob Frege:

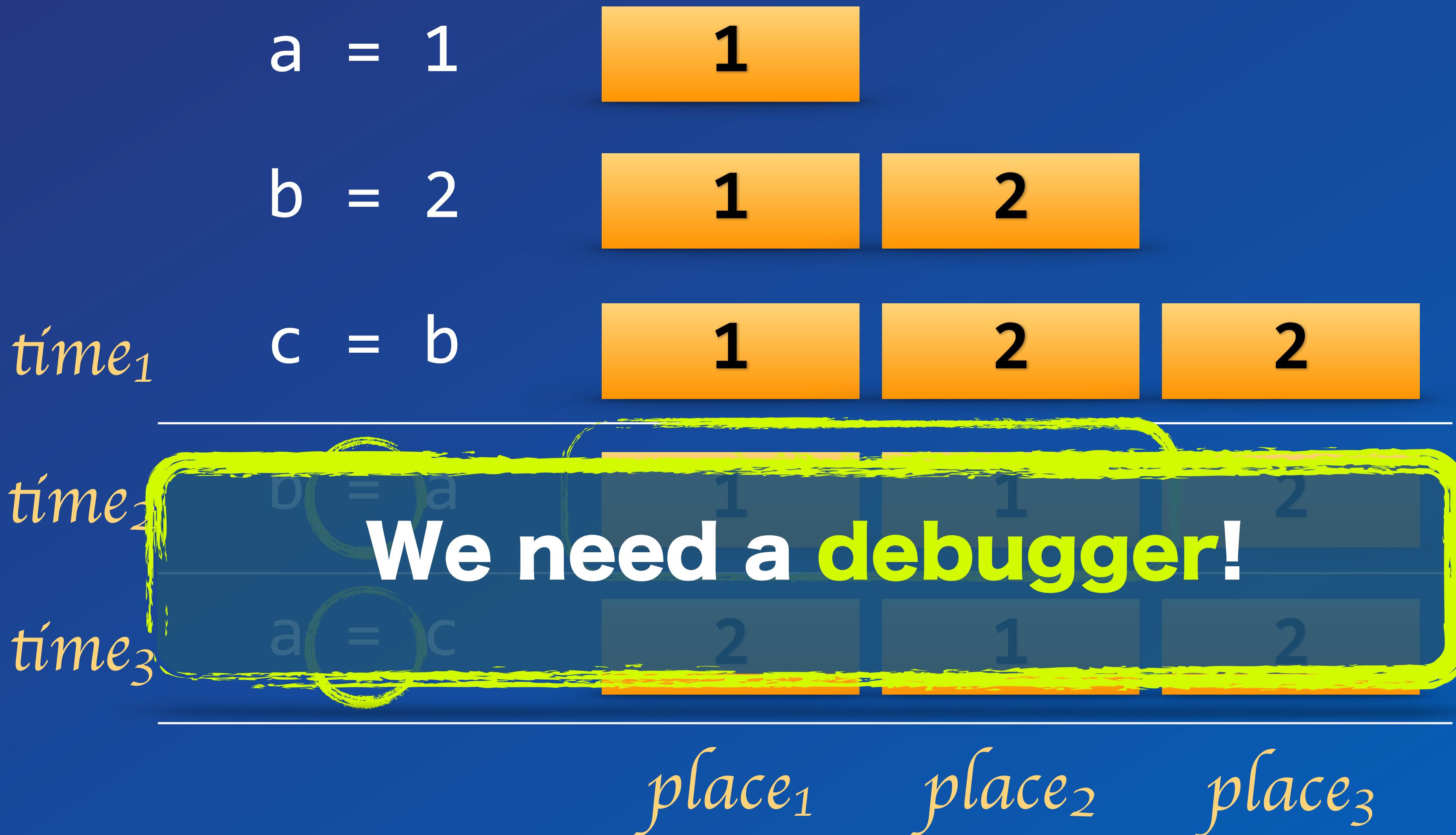
<http://www.youtube.com/watch?v=foITiYYu2bc>

Dreaming of code

Why do we care?



Operational Reasoning



Using functions

a = 1

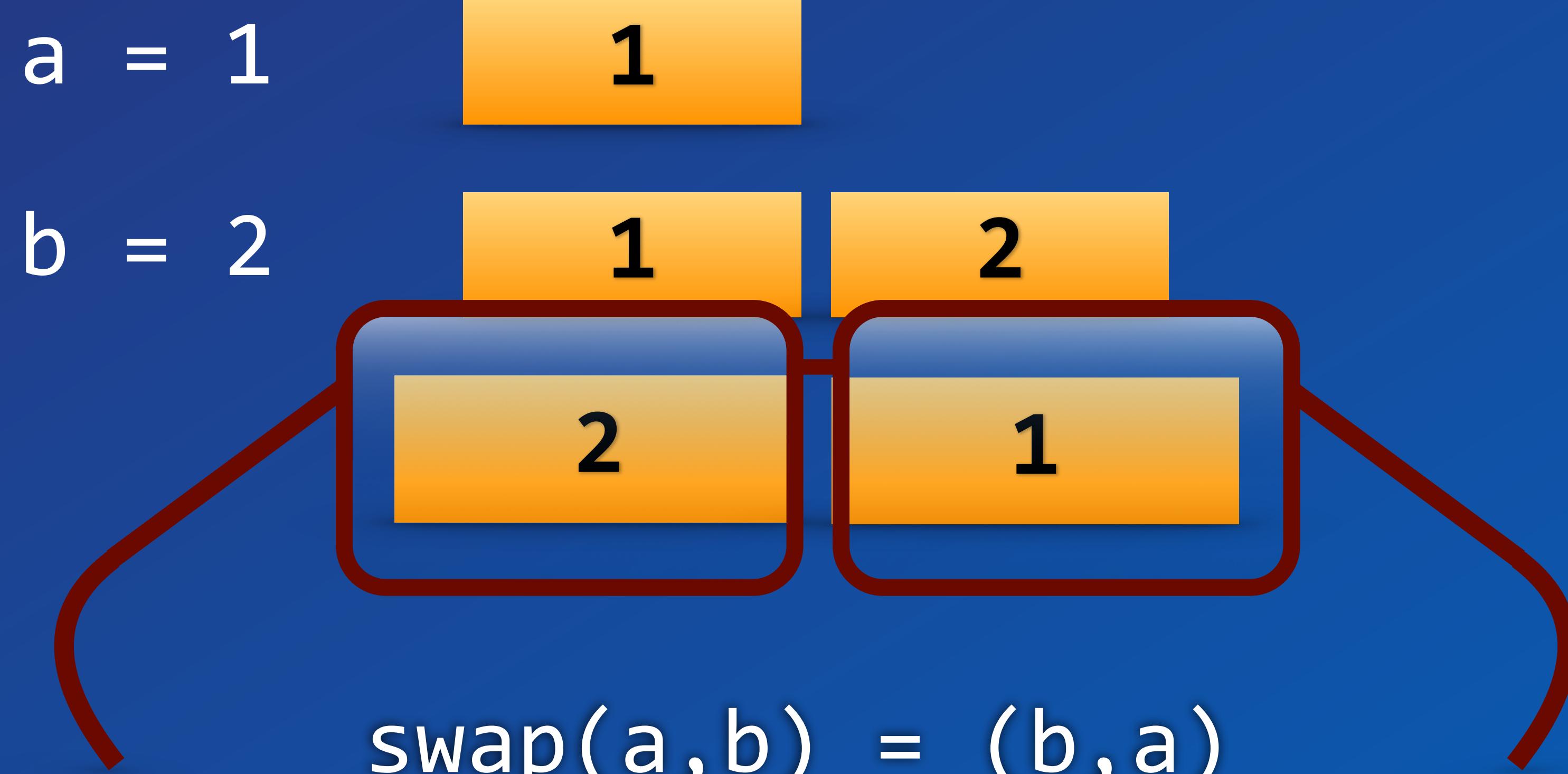
1

b = 2

1

2

Using functions



Let's just program
without assignments
or statements!



Developer
Discipline

Pure
Functional
Language

Go Strong

IDE Support

Safe Refactoring

Design Tool

No silly errors

Less tricky errors

Abstraction

Runtime Efficiency

Communicate

Insight

It's all based on purity

It's all based on purity

Pure functions produce the same result given the same arguments.

No observable side effects.

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?

```
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.getBean(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.doCreateBean(String, RootBeanDefinition, Object[])
                                                              java.security.AccessController.doPrivileged(PrivilegedAction, AccessControlContext)
                                                                org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.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.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.doCreateBean(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.getBean(String, Class, Object[], boolean)
                                                                                    org.springframework.beans.factory.support.AbstractBeanFactory.getBean(String, Class, Object[])
                      Let the type system find out!
```

Less tricky
errors

Pure Types

make **implicit** constraints **explicit**

Streams:

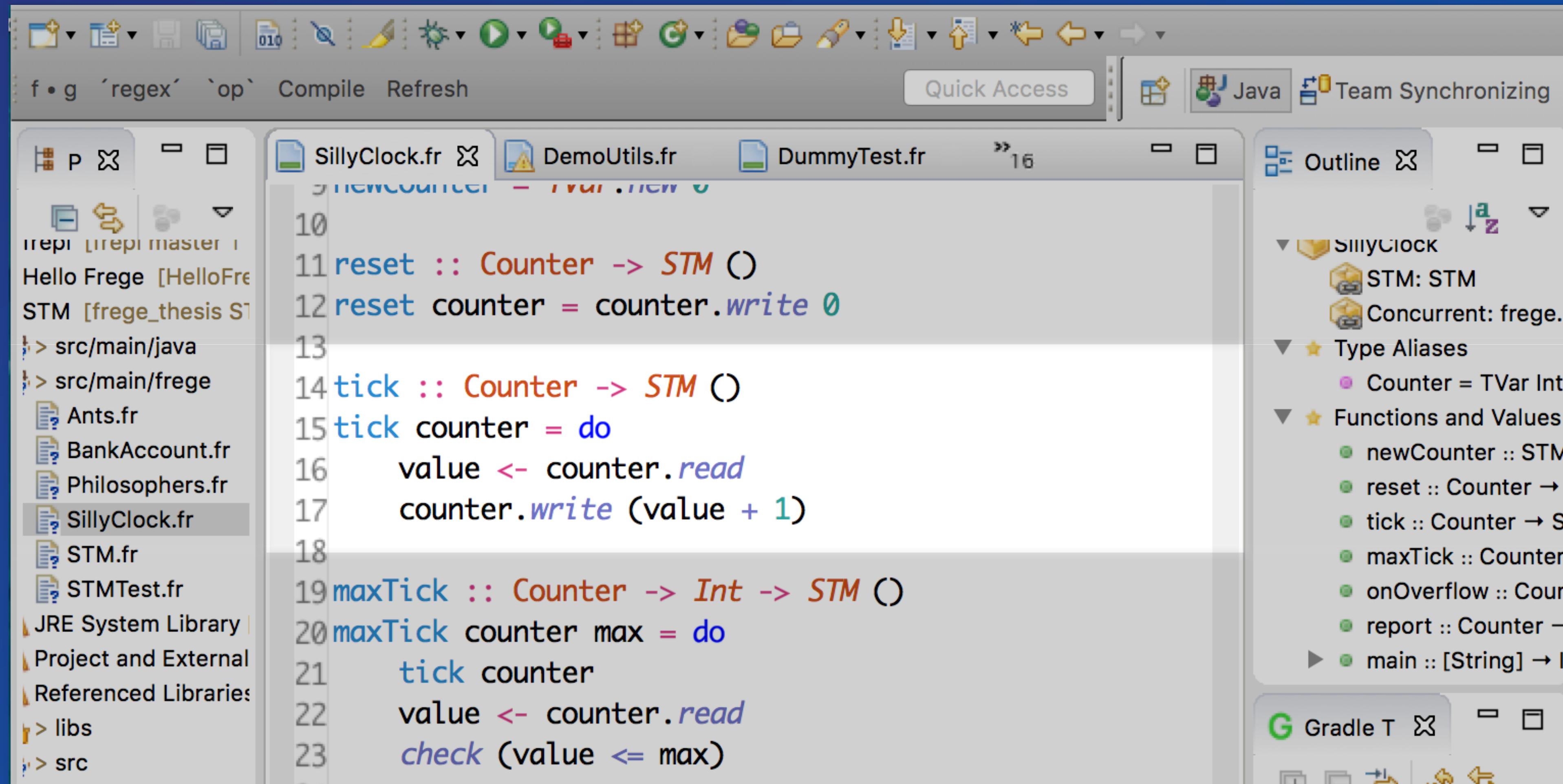
never modify, never do IO
in a (parallel) map/filter/reduce

JavaFX:

only touch nodes **inside** UI thread
only do IO **outside** UI thread

Transactions: no IO at all!

No IO in Transactions!



The screenshot shows an IDE interface with a Java perspective. The central editor window displays Frege code for a counter. The code defines three functions: `reset`, `tick`, and `maxTick`. The `reset` function initializes a counter to 0. The `tick` function increments the counter by 1. The `maxTick` function sets a maximum value for the counter and performs a check if the current value exceeds it. The code uses STM (Software Transactional Memory) operations like `write` and `read`.

```
10
11 reset :: Counter -> STM()
12 reset counter = counter.write 0
13
14 tick :: Counter -> STM()
15 tick counter = do
16     value <- counter.read
17     counter.write (value + 1)
18
19 maxTick :: Counter -> Int -> STM()
20 maxTick counter max = do
21     tick counter
22     value <- counter.read
23     check (value <= max)
```

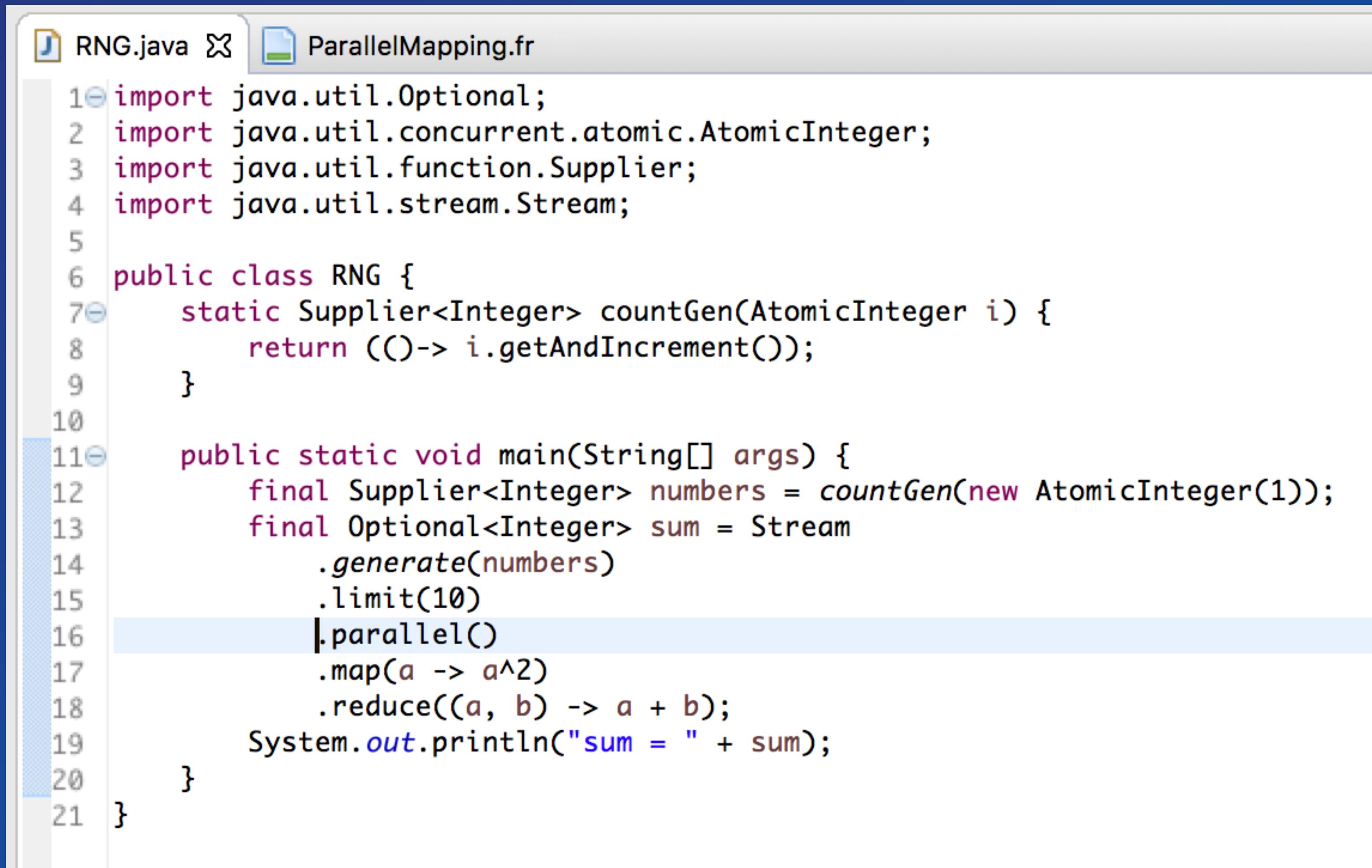
Less tricky
errors

Type inference FTW

The screenshot shows an IDE interface with a Frege project open. The left sidebar displays the file structure of the `SillyClock` project, including files like `STM.fr`, `Ants.fr`, and `SillyClock.fr`. The central editor pane contains Frege code with several error markers (red squiggly lines) underlined. The code defines three functions: `reset`, `tick`, and `maxTick`. The `reset` function initializes a counter. The `tick` function reads the current value, prints a message, and then increments it. The `maxTick` function sets a maximum value for the counter. The right sidebar shows the `Outline` and `Gradle T` panes.

```
10
11 reset :: Counter -> STM()
12 reset counter = counter.write 0
13
14 tick :: Counter -> STM()
15 tick counter = do
16     value <- counter.read
17     println "Hey, I am a side effect"
18     counter.write (value + 1)
19
20 maxTick :: Counter -> Int -> STM()
21 maxTick counter max = do
22     tick counter
23     value <- counter.read
```

Random Numbers



```
RNG.java ✘ ParallelMapping.fr
1 import java.util.Optional;
2 import java.util.concurrent.atomic.AtomicInteger;
3 import java.util.function.Supplier;
4 import java.util.stream.Stream;
5
6 public class RNG {
7     static Supplier<Integer> countGen(AtomicInteger i) {
8         return ()-> i.getAndIncrement();
9     }
10
11    public static void main(String[] args) {
12        final Supplier<Integer> numbers = countGen(new AtomicInteger(1));
13        final Optional<Integer> sum = Stream
14            .generate(numbers)
15            .limit(10)
16            .parallel() // Cursor is here
17            .map(a -> a^2)
18            .reduce((a, b) -> a + b);
19        System.out.println("sum = " + sum);
20    }
21 }
```

Follow along in the
Online REPL

<http://86.119.37.112:9999>
<http://try.frege-lang.org>

Define a Function

```
frege> times a b = a * b
```

```
frege> times 2 3
```

6

```
frege> :type times
```

Num α => α -> α -> α

Define a Function

```
frege> times a b = a * b
```

no types declared

```
frege>(times 2)3
```

6

function appl.
left associative

no comma

```
frege> :type times
```

Num a => a ->(a -> a)

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

```
frege> twotimes x = times 2 x
```

No second
arg!

```
frege> twotimes 3
```

6

```
frege> :t twotimes
```

Int -> Int

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

Concept invented by
Gottlob Frege.

inferred types
are more specific

Function Composition

```
frege> six x = twotimes (threetimes x)
```

```
frege> six x = (twotimes . threetimes)x
```

```
frege> six    = twotimes . threetimes
```

```
frege> six 2
```

Function Composition

fr $f(g(x))$ = twotimes (threetimes x)

fr $(f \circ g) x$ = (twotimes . threetimes)x

fr $f \circ g$ = twotimes . threetimes

frege> six 2

Pattern Matching

```
frege> times 0 (threetimes 2)
```

```
0
```

```
frege> times 0 b = 0
```

Pattern Matching

frege> times 0 (threetimes 2)

0

unnecessarily evaluated

frege> times 0 b = 0

pattern matching

shortcutting

Lazy Evaluation

frege> times 0 (length [1..])

0

Pattern matching and
non-strict evaluation
to the rescue!

endless sequence

evaluation would never stop

Communicate

Reduce to the Max

The designer/implementor cares about
parameter constraints

The consumer/user cares about
result constraints

Check out Rùnar Bjarnason
„Liberties constrain, constraints liberate“

Pure Functions

Java

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

What could possibly happen?

Frege

```
foo :: (α, β) -> α
```

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

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 :: () -> IO Long  
millis ()  
past = millis () - 1000
```

Does not compile!

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

More Native Declarations

```
native println System.out.println :: String -> IO ()  
println "Dierk König"
```

```
native getText :: TextInputControl -> JFX String  
text <- inputField.getText  
println text      -- does not compile!  
inIO text println -- compiles!
```

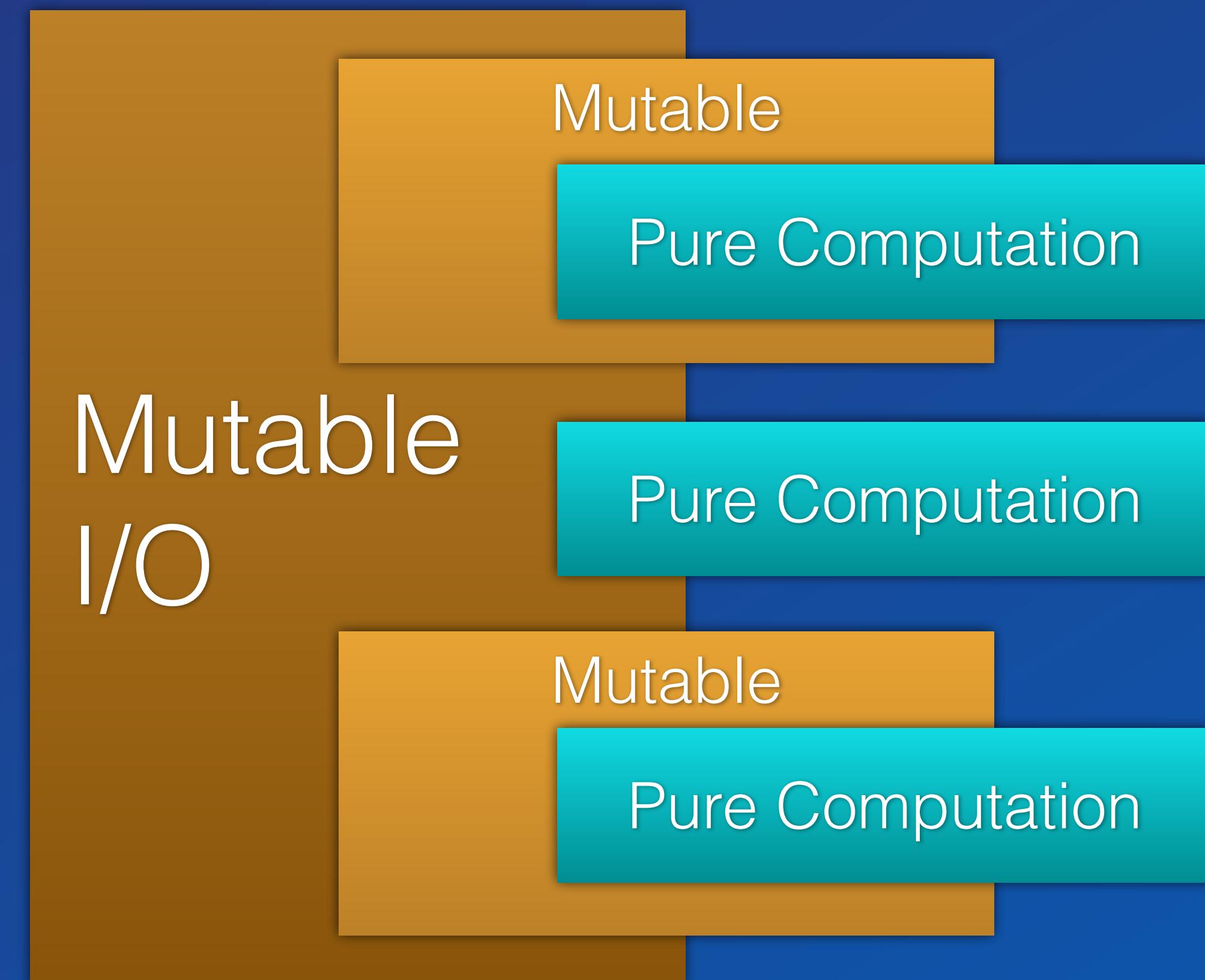
Frege

allows calling Java
but never unprotected!

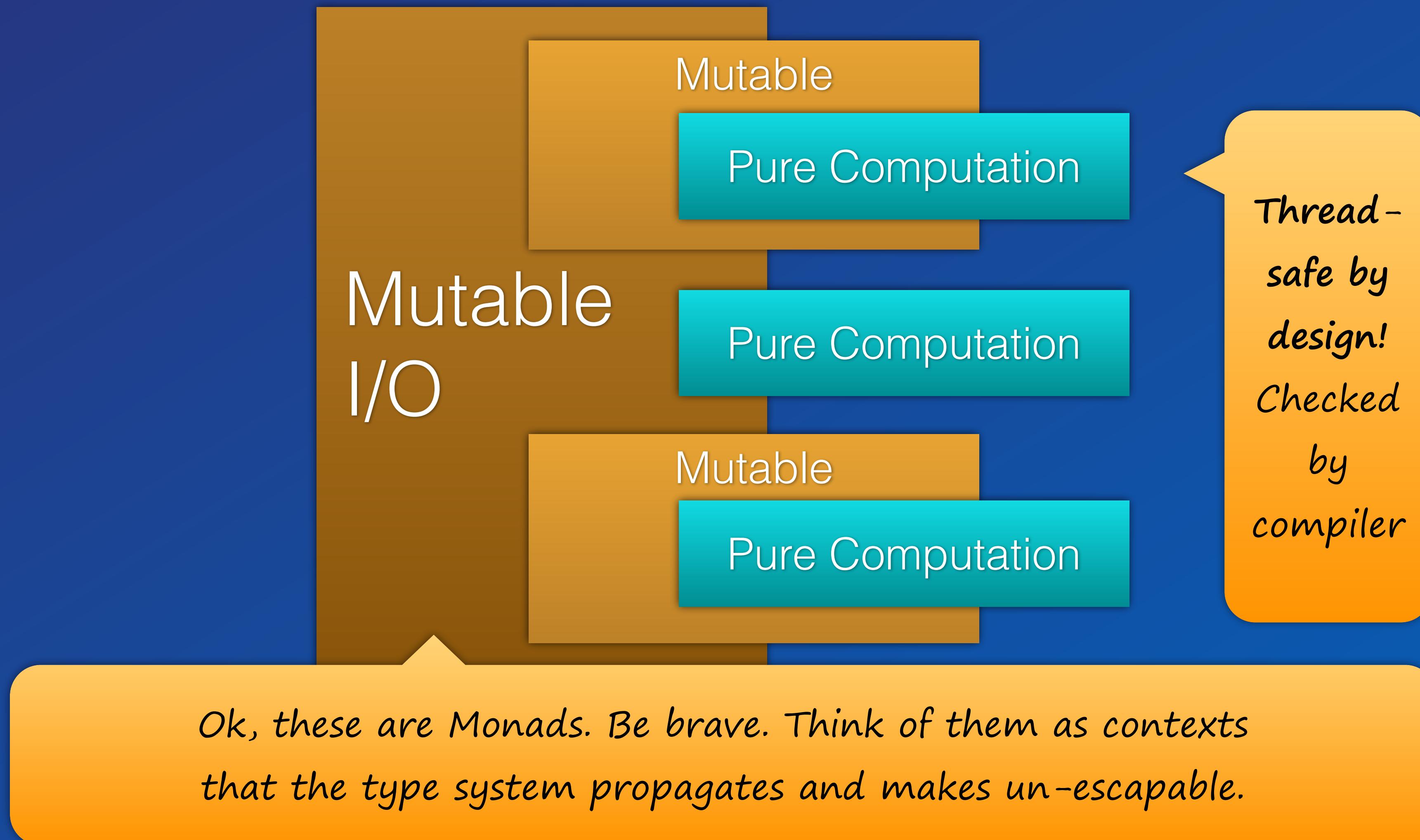
is explicit about effects
just like Haskell

Prerequisite to safe concurrency and
deterministic parallelism!

Keep the mess out!



Keep the mess out!



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 for clarity.

Fizzbuzz

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

<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 bestOf numbers pattern where
            bestOf n "" = n
            bestOf n p  = p

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	-	+++

QuickCheck

-- An AVL tree is balanced so
that the height of the left and
right subtree differ by at most 1

```
p_balance = forAll aTree  
(\tree -> abs tree.balance < 2)
```

QuickCheck will create 500 different trees
covering all corner cases in creation and validate
the invariant. (from Frege source code)

History

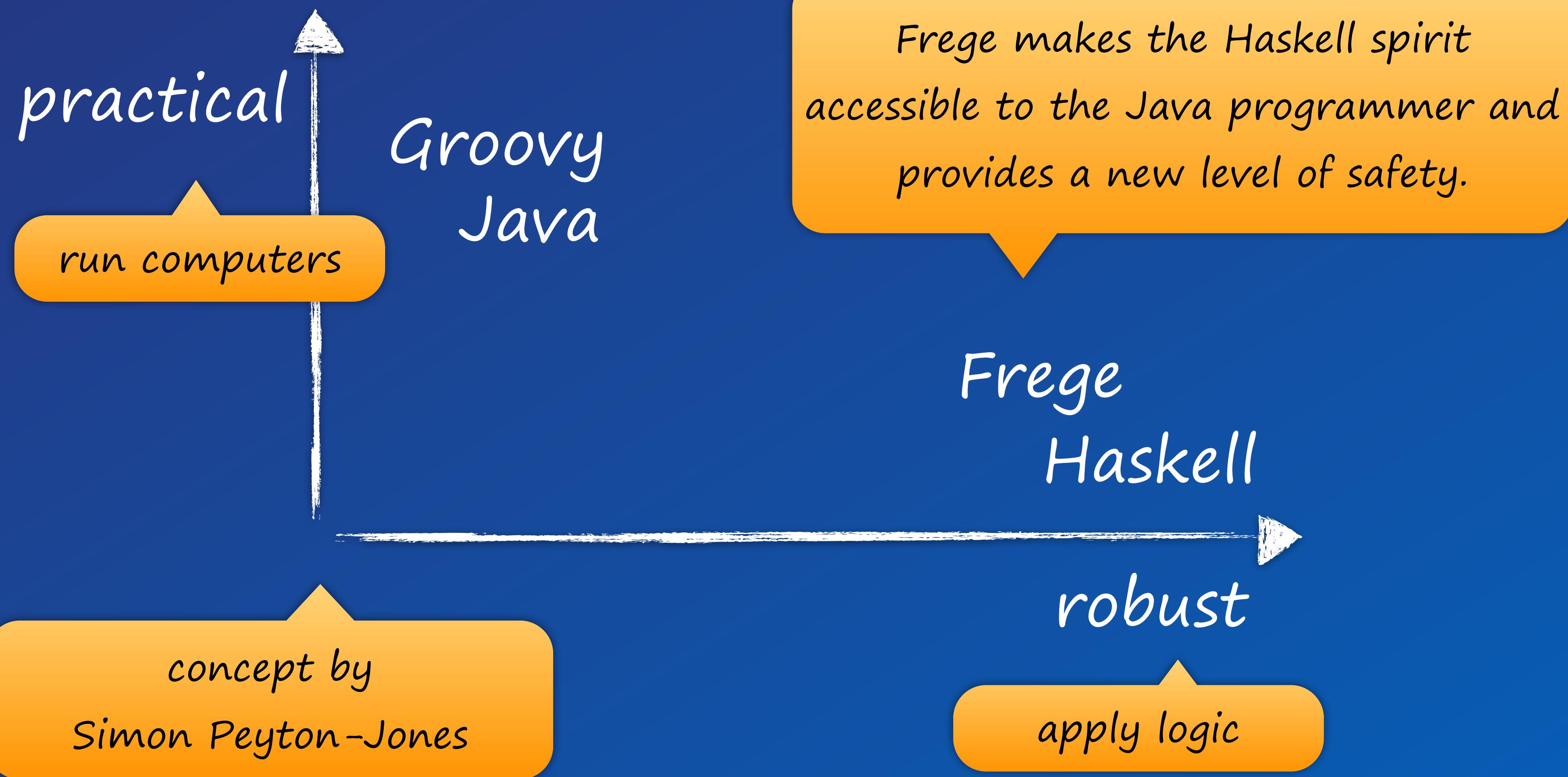
Java promise: „No more pointers!“

But NullPointerException (?)

Frege in comparison



Frege in comparison



Unique in Frege

Global type inference (*requires purity*)

Purity by default

effects are explicit in the type system

Type-safe concurrency & parallelism

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
functional programming**

Why FP matters

Enabling incremental development

see FregeGoodness "Incremental Development"

Brush up computational fundamentals

„An investment in knowledge
always pays the best interest.“

—Benjamin Franklin

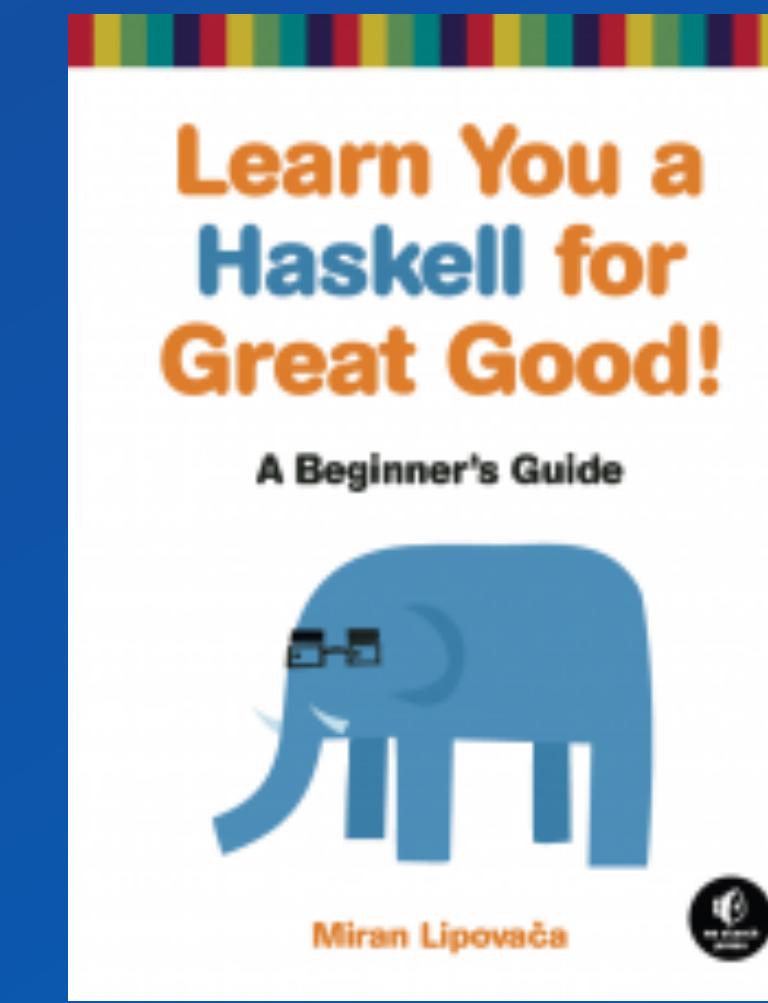
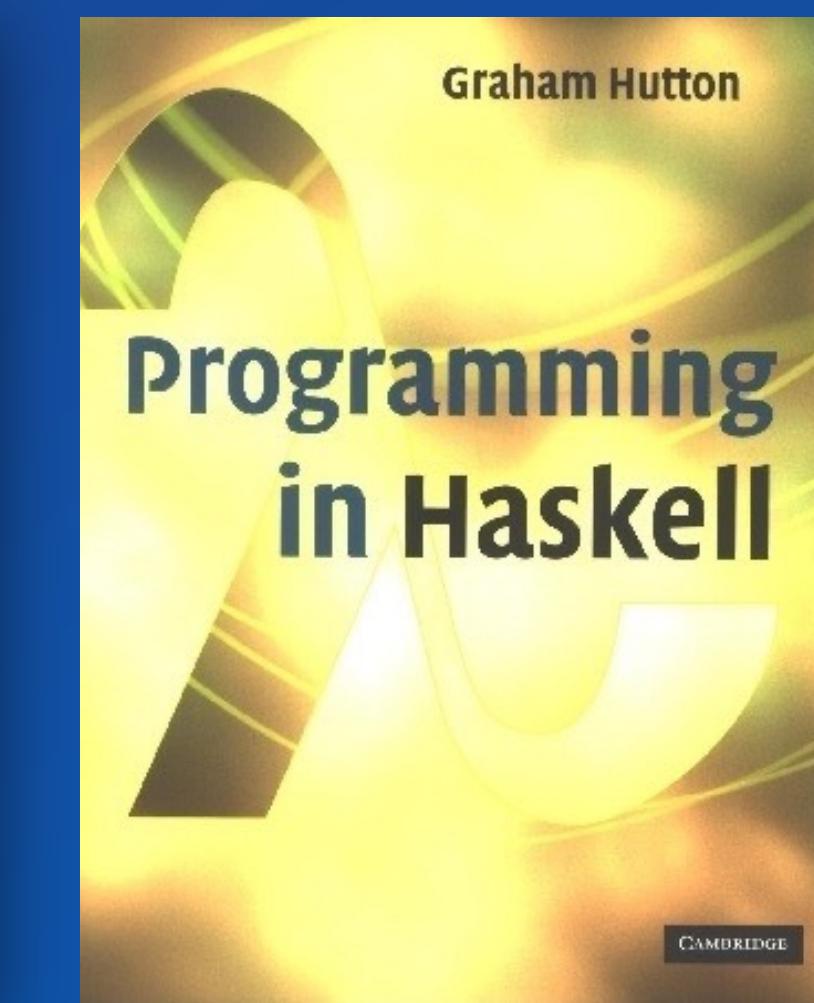
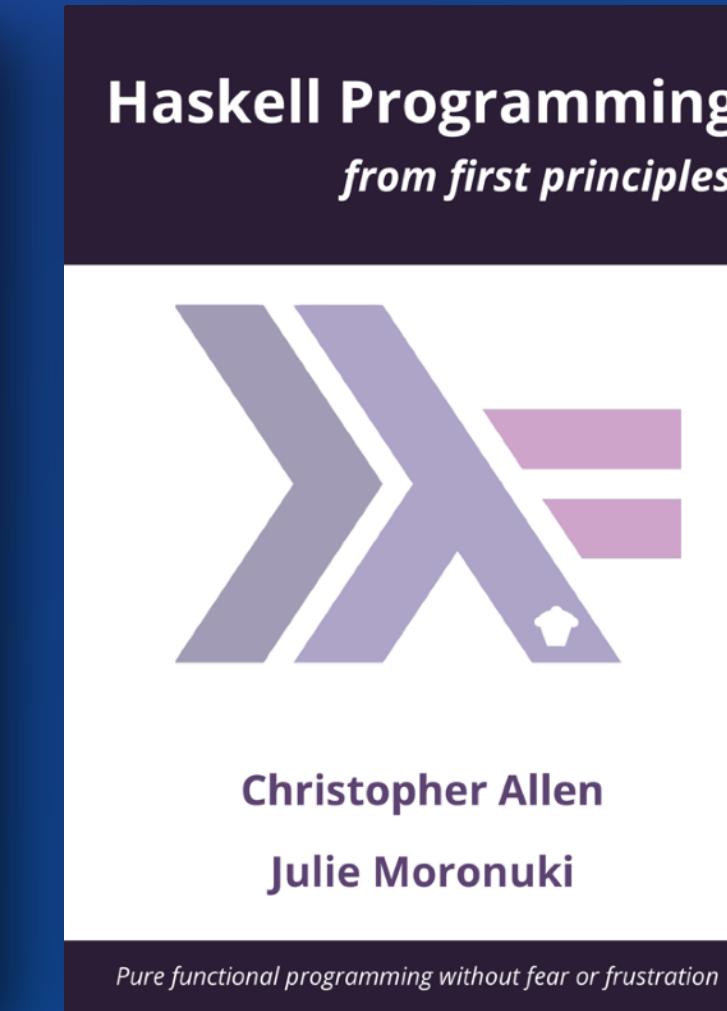
How?

<http://www.frege-lang.org>

@fregelang

stackoverflow „frege“ tag

edX FP101 MOOC



Community

Frege compiles Haskell to Java.

It combines both communities.

Popularity: 3500+ github stars

top 10 of 26k Haskell projects

top 0.03% of 1M Java projects

Tool support

All Java tools!

Frege Eclipse, IDEA, VS Code plugin

Haskell mode in editors/IDEs

Maven, Gradle, Leinigen, Bazel, make

Compiler, doc tool, REPLs, quickcheck

Training Material

All Haskell material applies:

MOOCs, Books, Lectures, Videos

Scientific Papers

Specialized Frege training
at Karakun



Questions?

Prof. Dierk König



@mittie



FGA

Language level is Haskell Report 2010.

Yes, performance is roughly ~ Java.

Yes, the compiler is reasonably fast.

Yes, we have IDE Plugins.

Yes, Maven/Gradle/etc. integration.

Yes, we have HAMT (aka HashMap).

Yes, we have QuickCheck (+shrinking)

Yes, we have STM.