Purely functional programming - the red pill

JFokus Tutorial 2016

Dierk König

canoo

mittie

Dreaming of code

Why do we care?

	a = 1	1		
	b = 2	1	2	
time$_1$	c = b	1	2	
time$_2$	b = a	1		2
time$_3$	a = c		1	2
		place$_1$	*place*$_2$	*place*$_3$

Operational Reasoning

		place$_1$	place$_2$	place$_3$
	a = 1	1		
	b = 2	1	2	
time$_1$	c = b	1	2	2
time$_2$	b = a	1	1	2
time$_3$	a = c	2	1	2

We need a debugger!

Using functions

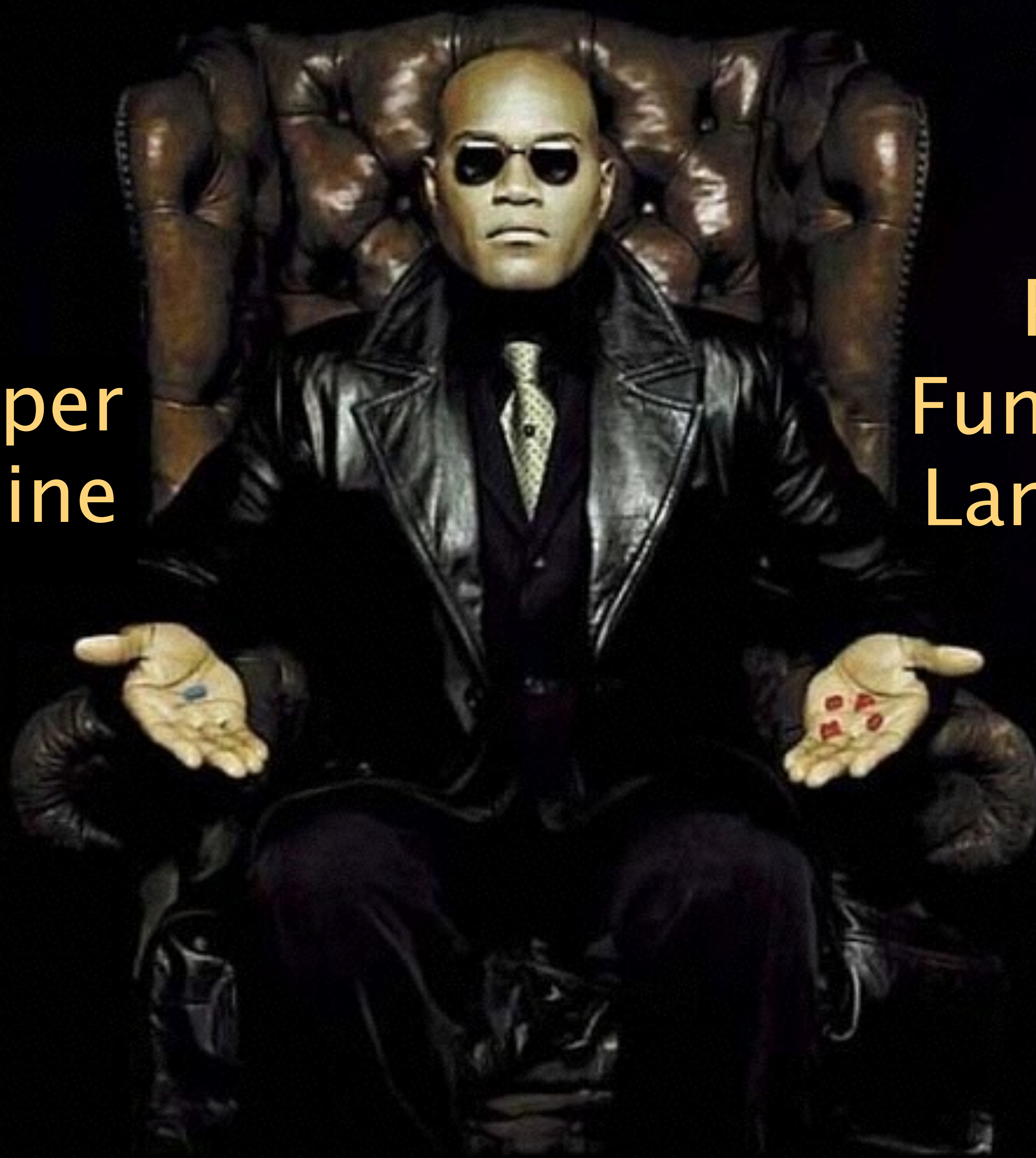
```
a = 1
b = 2
```

Using functions

```
a = 1
b = 2
```

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

Let's just program without assignments or statements!



Developer
Discipline

Pure
Functional
Language

Online REPL

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
```

function appl. left associative

no comma

```
6

frege> :type times

Num α => α ->(α -> α)
```

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

typeclass constraint

only 1 parameter!

return type is a function!

Reference a Function

```
frege> twotimes    = times 2

frege> twotimes 3

6

frege> :t twotimes

Int -> Int
```

Reference a Function

```
frege> twotimes x = times 2 x

frege> twotimes 3

6

frege> :t twotimes

Int -> Int
```

No second arg!

„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

12
```

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

12
```

Pattern Matching

```
frege> times 0 (threetimes 2)

0

frege> times 0 b = 0
```

Pattern Matching

```
frege> times 0 (threetimes 2)

0

frege> times 0 b = 0
```

unnecessarily evaluated

pattern matching

shortcutting

Lazy Evaluation

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

0
```

endless sequence

evaluation would never stop

Pattern matching and non-strict evaluation to the rescue!

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) {…}
```

Everything!

State changes,

file or db access,

missile launch,…

Frege

```
foo :: (α,β) -> α
```

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?

```
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,
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.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, RuntimeBeanR
org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveValueIfNecessary(Object, Ob
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.applyPropertyValues(
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(Strir
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(
```

Let the type system find out!

Magic (?)

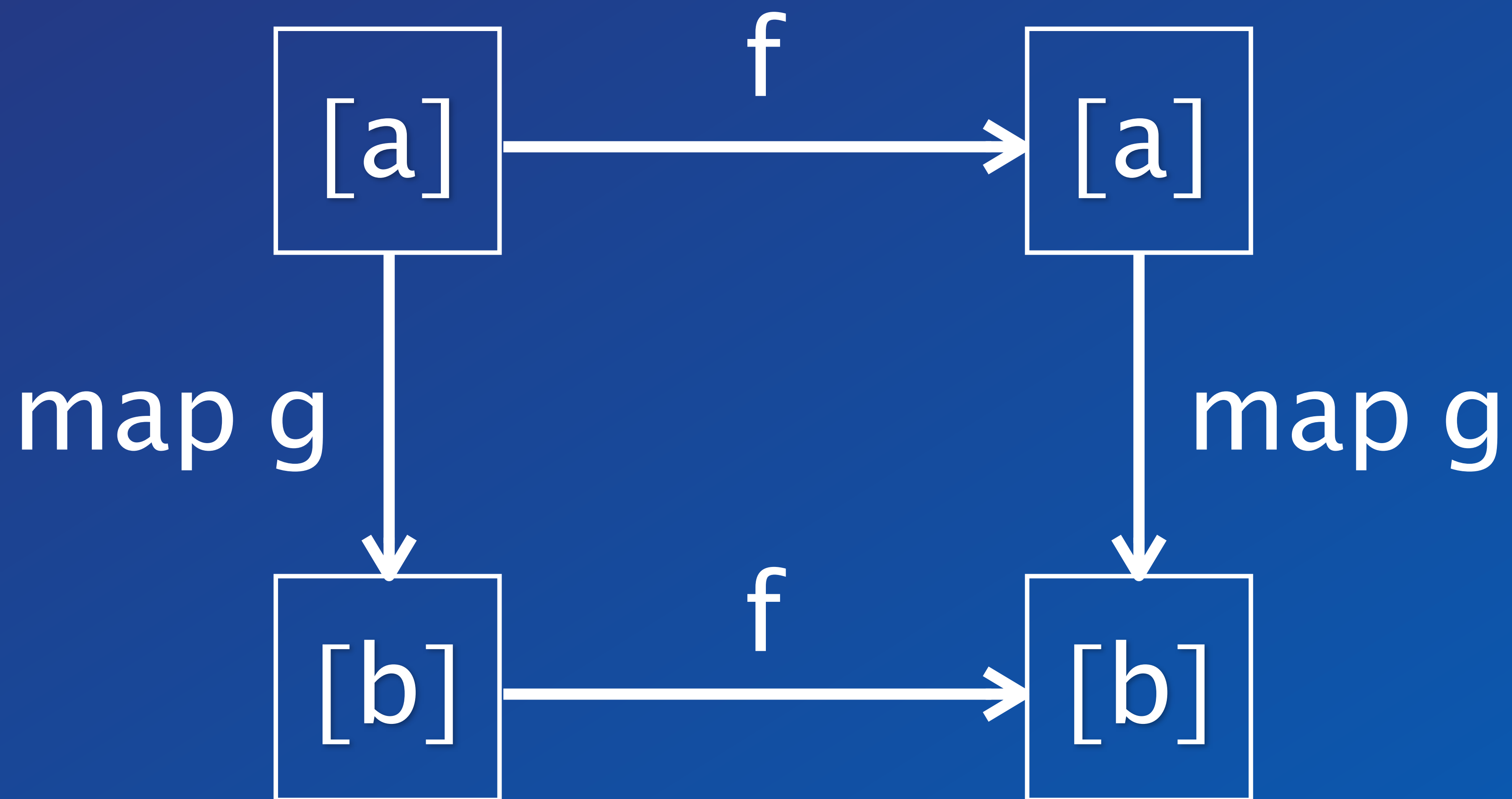
Think of a generic function

```
f :: [a] -> [a]
```

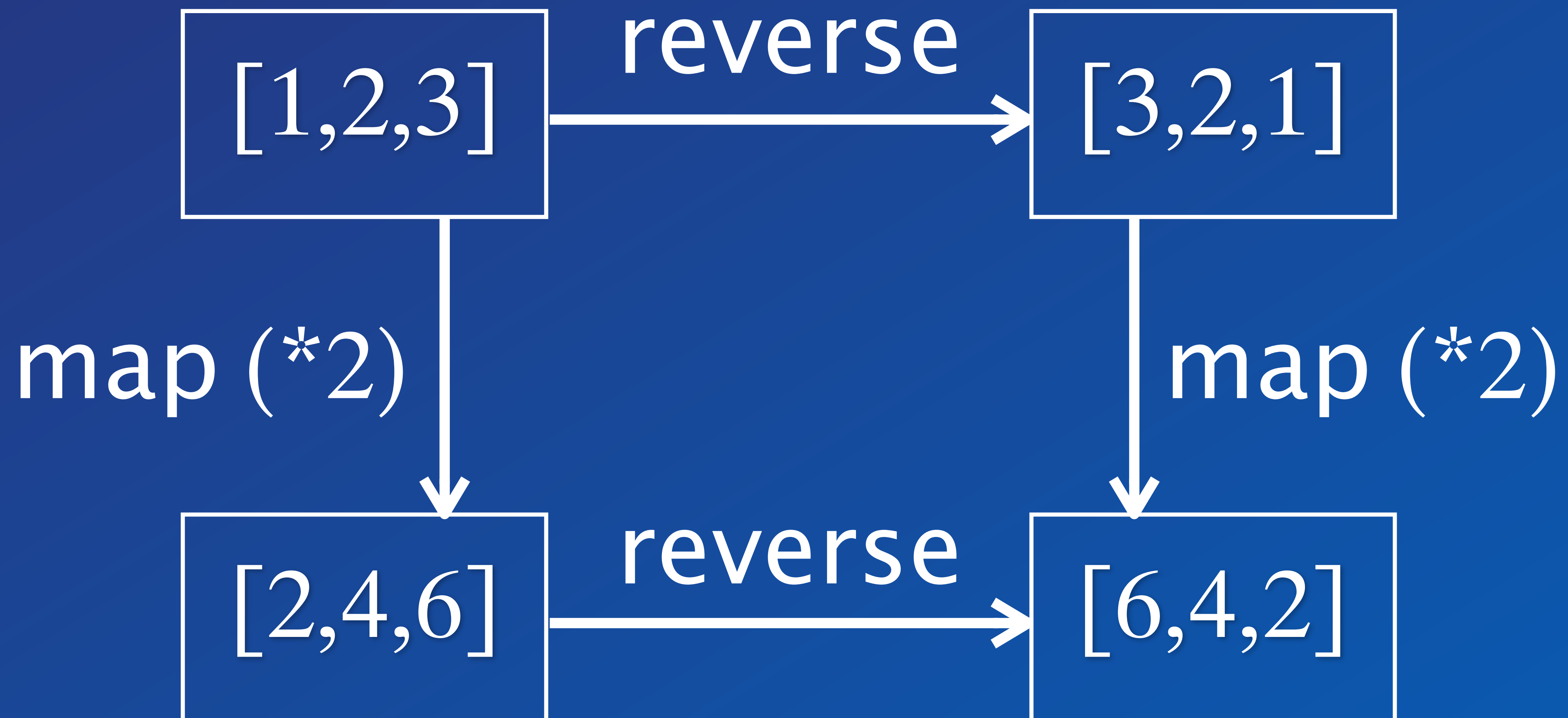
and any specific function

```
g :: a -> b
```

Magic (?)



Magic (?)



Commutative square of natural transformations.

Robust Refactoring

Changing the order of operations

Requires purity.
100% safe if the type system
can detect natural transformations

Applied Category Theory

credits: Phil Wadler, Tech Mesh 2012 – Faith, Evolution, and Programming Languages

QuickCheck

```
import Test.QuickCheck
f = reverse
g = (*2)

commutativity = property (\xs ->
 map g (f xs) == f (map g xs) )
```

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 ()
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

Prerequisite to safe concurrency and deterministic parallelism!

Type System

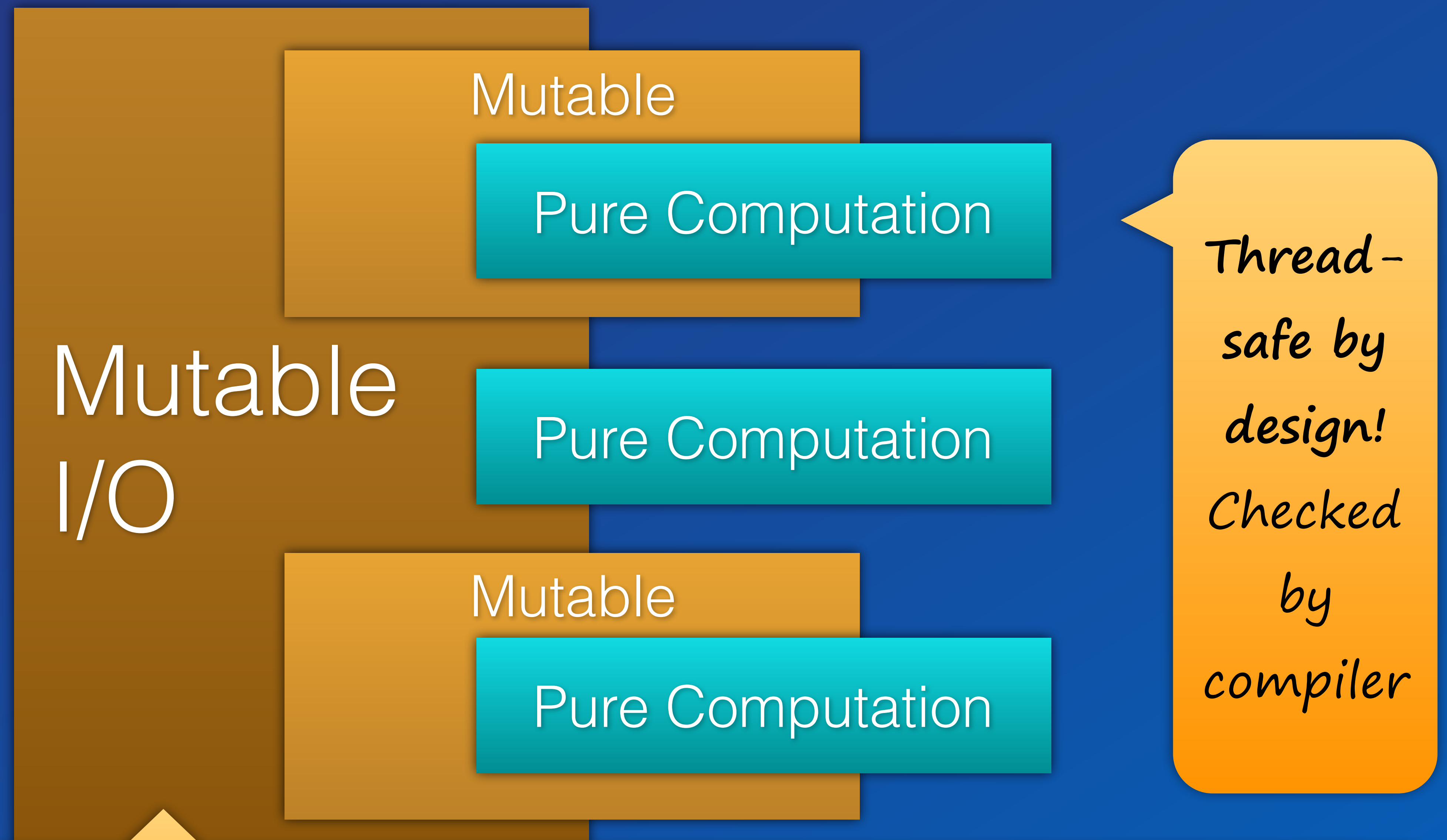
Global type inference

More safety and less work for the programmer

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

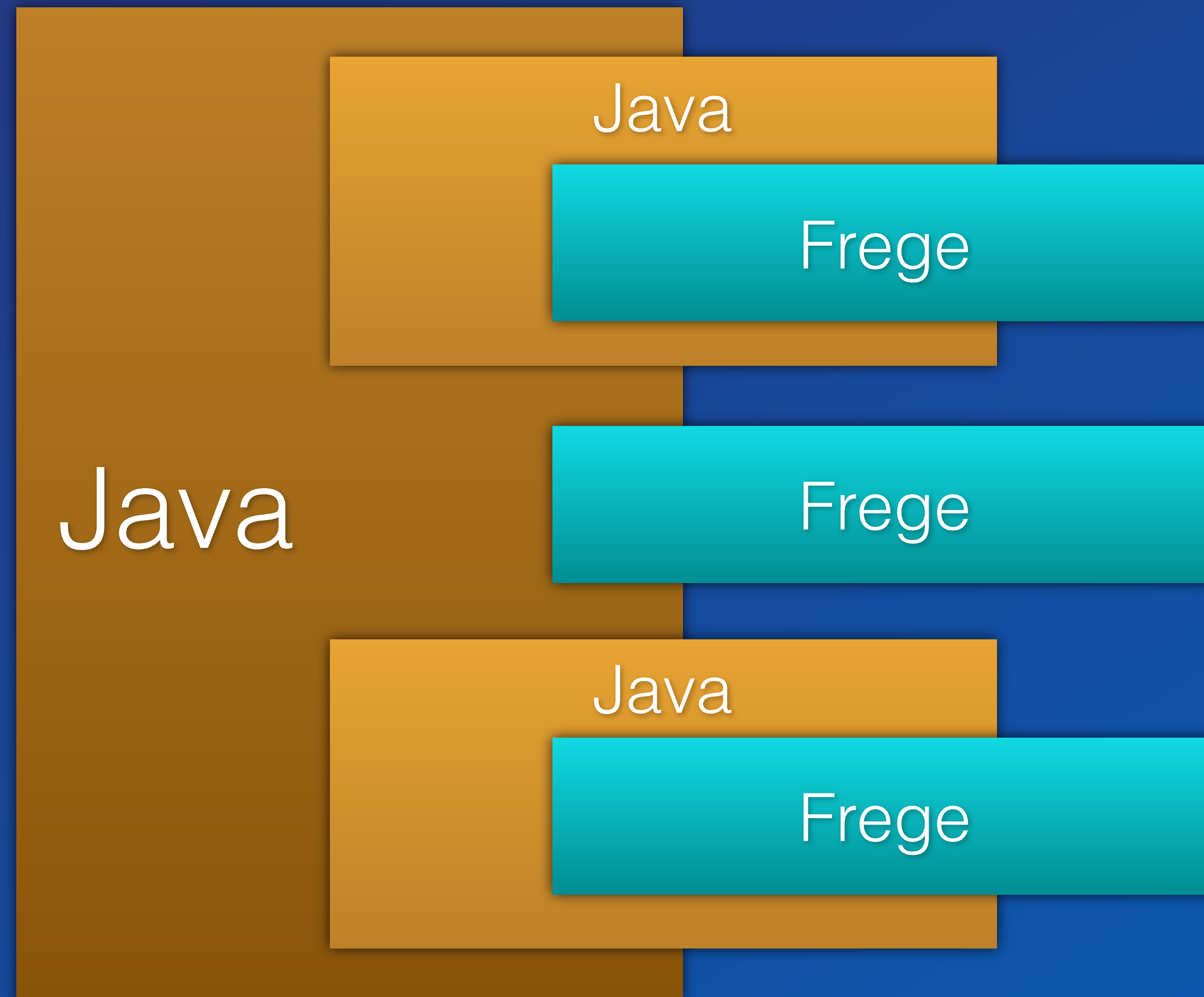
Keep the mess out!

Keep the mess out!



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

Service Based Design



A typical integration option: use Frege code for services

Some Cool Stuff

Zipping

```
addzip [] _  = []
addzip _  [] = []

addzip (x:xs) (y:ys) =
       (x + y : addzip xs ys )
```

Zipping

```
addzip [] _   = []
addzip _  [] = []

addzip (x:xs) (y:ys) =
       (x + y : addzip xs ys )
```

Pattern matching feels like Prolog

use as

```
addzip [1,2,3]
       [1,2,3]
    == [2,4,6]
```

Why only for the (+) function? We could be more general...

High Order Functions

```
zipWith f [] _  = []
zipWith f _  [] = []

zipWith f (x:xs) (y:ys) =
       (f x y : zipWith f xs ys )
```

invented by Gottlob Frege

High Order Functions

```
zipWith f [] _   = []
zipWith f _  [] = []

zipWith f (x:xs) (y:ys) =
        (f x y : zipWith xs ys )
```

use as

```
zipWith (+) [1,2,3]
            [1,2,3]
         == [2,4,6]
```

and, yes we can now define

```
addzip =
     zipWith (+)
```

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 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	-	+++

Fibonacci

```
fib = 0: 1: addzip fib (tail fib)
```

use as

```
take 60 fib
```

a new solution approach

```
fib  0:1 …
tail 1 …
zip  1 …
```

Fibonacci

```
fib = 0: 1: addzip fib (tail fib)
```

use as

```
take 60 fib
```

a new solution approach

```
fib   0 1:1 …
tail    1 …
zip     2 …
```

Fibonacci

```
fib = 0: 1: addzip fib (tail fib)
```

use as

```
take 60 fib
```

a new solution approach

```
fib   0 1 1:2 …
tail        2 …
zip         3 …
```

Fibonacci

```
fib = 0: 1: addzip fib (tail fib)
```

use as

```
take 60 fib
```

a new solution approach

```
fib   0 1 1 2:3 …
tail          3 …
zip           5 …
```

List Comprehension

Pythagorean triples: $a^2 + b^2 = c^2$

```
pyth n = [
 (x,y,z)
 | x <- [1..n], y <- [1..n], z <- [1..n],
 x*x + y*y == z*z
 ]
```

List Comprehension

Pythagorean triples: $a^2 + b^2 = c^2$

```
pyth n = [
 (x,y,z)
 | x <- [1..n], y <- [1..n], z <- [1..n],
 x*x + y*y == z*z
 ]
```

select

from

where

„brute force“ or „executable specification“.
A more efficient solution:

List Comprehension

Pythagorean triples: $a^2 + b^2 = c^2$

```
[ (m*m-n*n, 2*m*n, m*m+n*n)
| m <- [2..], n <- [1..m-1]
]
```

„select“ functions

dynamic „from“

empty „where“

endless production

think „nested loop“

History

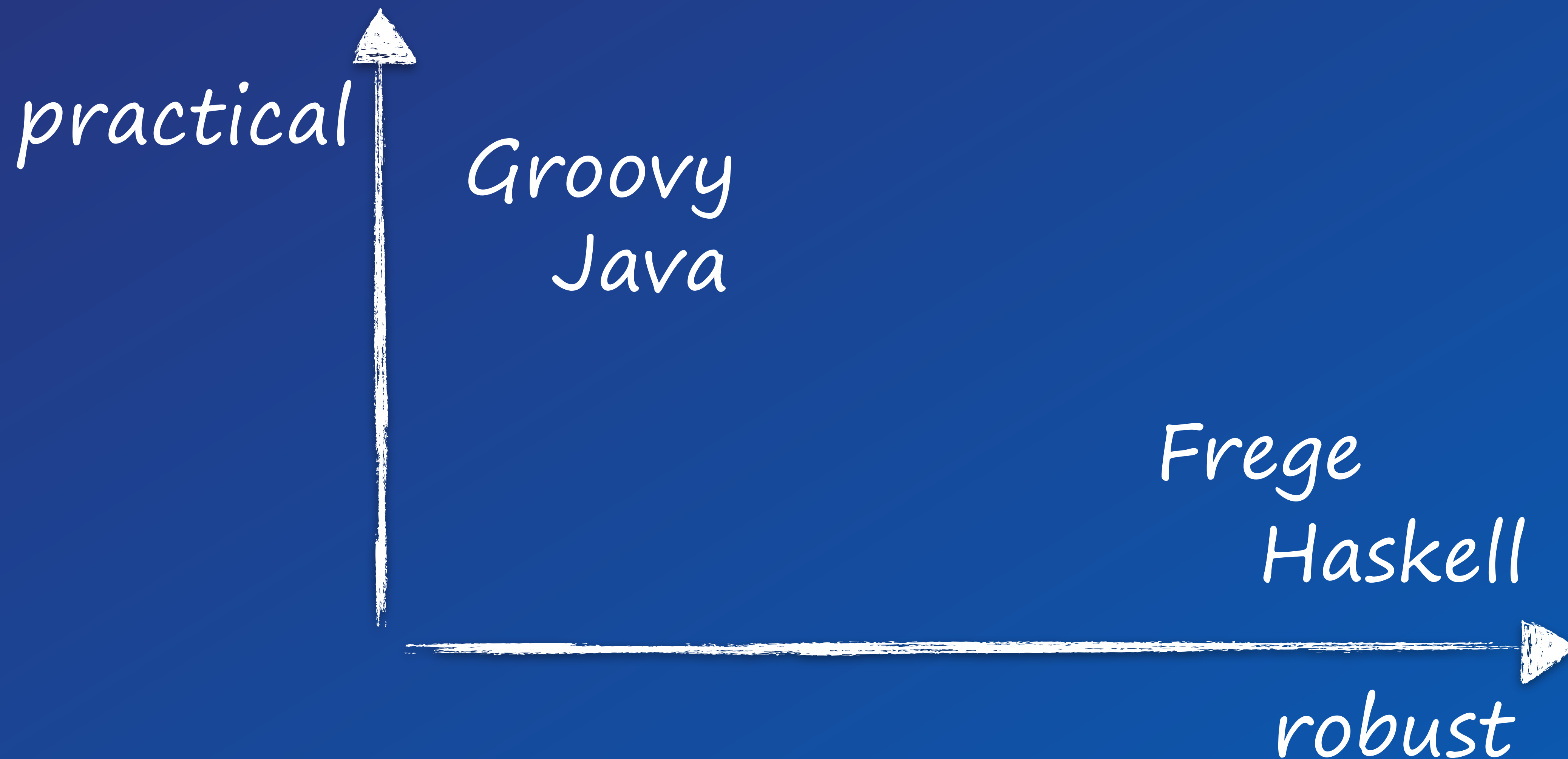
Java promise: „No more pointers!“

But Null**Pointer**Exceptions (?)

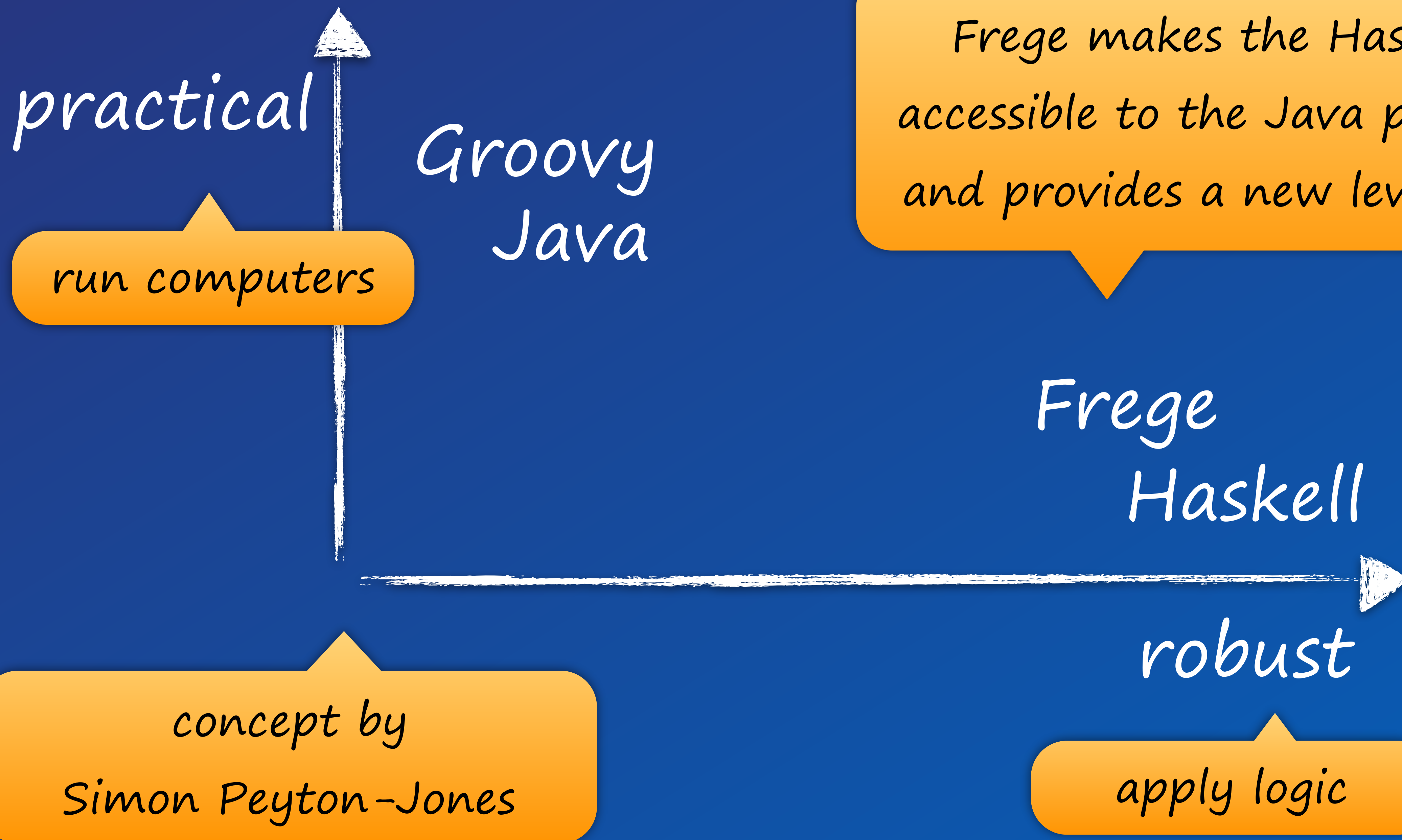
Frege is different

No More	But
state	no state (unless declared)
statements	expressions (+ „do“ notation)
assignments	definitions
variables	ST monad as „agent“
interfaces	type classes
classes & objects	algebraic data types
inheritance	parametric polymorphism
null references	Maybe
NullPointerExceptions	Bottom, error

Frege in comparison



Frege in comparison



Frege makes the Haskell spirit accessible to the Java programmer and provides a new level of safety.

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 FP matters

Enabling incremental development

www.canoo.com/blog/fp1

Brush up computational fundamentals

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

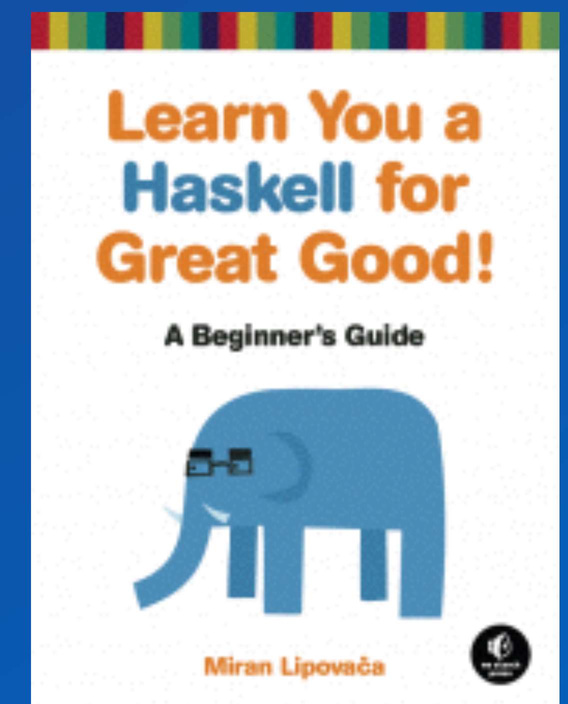
Why Frege

it is just a pleasure to work with

a
c
c
b
c
a
b
c
that's why

How?

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



Gottlob Frege

"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 Russel

"Not many people managed to create a revolution in thought. Frege did. Twice." —Graham Priest

Lecture on Gottlob Frege: http://www.youtube.com/watch?v=foITiYYu2bc

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)
Yes, STM is almost finished.

How it goes on

Practical project work
github.com/Dierk/fregeTutorial.git

Purely functional turtle graphics

Game / Web

Software Transactional Memory