WHAT JAVA CAN LEARN FROM HASKELL

AND VICE VERSA

WHAT JAVA CAN LEARN FROM HASKELL AND VICE VERSA

Nick Linker @nick_linker

EXAMPLES (SOURCE)

Anchor Applikativ Assertible Borders Beautiful_Destinations ByteAlly Capital_Match Chordify CircuitHub Commonwealth Bank DigitalX Elsen Extensibl Facebook FP_Complete FPInsight Front_Row_Education Helium_Systems Hooky,_Inc Infinipool Iris_Connect Keera_Studios Kite_&_Lightning Least_Fixed LexisNexis_Risk_Solutions Lumi_Guide Madriska_Inc. Microsoft Midroll MyFansDemand Picus_Security Pivot_Cloud Prezi Rheo_Systems Scoompa Scrive Scyfy_Technologies Silk SimplyRETS Snowdrift.coop Soostone Stack_Builders Standard_Chartered Stitcher Suite_Solutions SumAll Swift_Navigation Systor_Vest thoughtbot Tree.is Tsuru_Capital Turing_Jump UpHere VaryWell VFILES Virtual_Forge Wagon Wellposed Well-Typed Zalore

HASKELL IS ...

- Functional
- Pure
- Lazy (by default)
- With advanced type system
- GHC
  - 25 years old, but moves fast
  - last release 2016-05-21

WHAT JAVA CAN LEARN

- Expressive syntax
- Purity
- Expressive Type System
- GHCi (REPL)
- ...

EXPRESSIVE SYNTAX

EXPRESSIVE SYNTAX (1)

```
size xs = loop xs 0
  where
    loop [] acc = acc
    loop (_ : xs) acc = loop xs (acc + 1)

-- Usage
size [1,2,3]
```

EXPRESSIVE SYNTAX (2)

The type of size above

```
size :: [t] -> Integer     -- ?

size :: Num a => [t] -> a  -- actuall type
```

In general

```
f :: (C1 a) => a -> b -> c -> d -> e
(f a1) ::           b -> c -> d -> e
(f a1 b1) ::             c -> d -> e
(f a1 b1 c1) ::               d -> e
(f a1 b1 c1 d1) ::                 e
```

EXPRESSIVE SYNTAX (3)

A spherical program in vacuum

```
module My.Foo where

import Data.Time hiding (Day)

foo :: IO ()
foo = do
  ct <- getCurrentTime
  putStrLn ("UTC time = " ++ show ct)
```

PURITY

PURITY (1)

CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Seriously, what about file system? Network? Random?

PURITY (2)

```
long getLength(String str) {
        return str.length();
}

long getFileLength(String path) {
         return new File(path).length();
}
```

They have the same Java type.

However, these functions are not interchangeable!

PURITY (3)

In Haskell they'd have different types:

```
getLength :: String -> Integer

getFileLength :: String -> IO Integer
```

(Monad tutorial goes here...)

EXPRESSIVE TYPES

EXPRESSIVE TYPES (1)

NEWTYPES

```
// call this as runScript("sql/RunStuff.sql")
Result runScript(String script) { ...}
```

- Milliseconds vs seconds
- Username vs password
- Paths vs contents
- Indices

In Haskell wrapping can be free!

EXPRESSIVE TYPES (2)

NEWTYPES

```haskell
-- typesafe runScript
newtype Path = Path String

runScript :: Path -> IO Result
```

Looks like a separate type, but low-level representation is the same.

EXPRESSIVE TYPES (3)

ALGEBRAIC DATA TYPES AND PATTERN MATCHING

```
data Void
data X = X
data Y = Y Int Text X
data Z = Zx X | Zy Y
data Day = Mon | Tue | Wed | Thu | Fri | Sat | Sun
data User = User { id :: Int, name :: Text, day :: Day }
```

EXPRESSIVE TYPES (4)

ALGEBRAIC DATA TYPES AND PATTERN MATCHING

Constructing values and matching

```
let z = Zy (Y 123 "Hey" X)
let u1 = User { id = 1, name = "Vasya", day = Mon }
let u2 = User 2 "Petya" Tue
let d = Sat

wd :: Day -> String
wd d | d `elem` [Mon, Tue, Wed, Thu] -> "Working day"
wd d | d `elem` [Sat, Sun]           -> "Weekend day"
wd Fri                               -> "Friday"

case u1 of
  User id name day -> ...
```

EXPRESSIVE TYPES (5)

TYPE CLASSES

Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple				
Orange				

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (5)

TYPE CLASSES

Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple	Eq			
Orange				

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (5)

TYPE CLASSES

Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple	Eq	Ord		
Orange				

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (5)

TYPE CLASSES

Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple	Eq	Ord		
Orange	Eq			

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (5)

TYPE CLASSES

Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple	Eq	Ord		
Orange	Eq		ToJSON	

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (5)

TYPE CLASSES

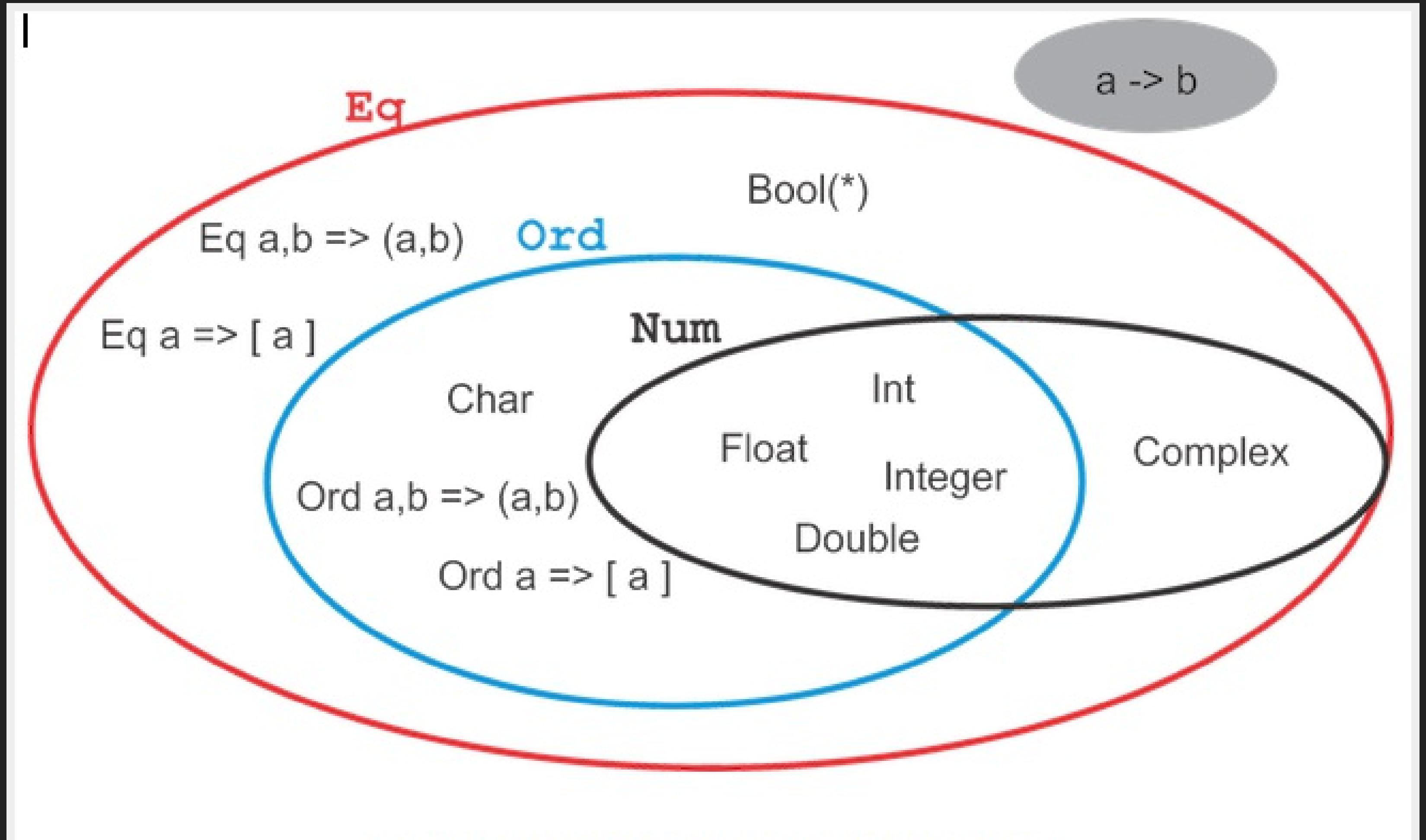
Typeclasses decouple the declaration that a type implements an interface from the declaration of the type itself

Data Type	Eq	Ord	ToJSON	FromJSON
Apple	Eq	Ord		
Orange	Eq		ToJSON	
Lemon	Eq	Ord	ToJSON	FromJSON

More on Intefaces vs Typeclasses

EXPRESSIVE TYPES (6)

TYPECLASSES



`a->b` (Tipo función) no pertenece a ninguna clase
Recordar que `String` es sinónimo de `[Char]` por ende está incluido en `(Ord a) => [ a ]`
(*) el tipo `Bool` en realidad pertenece a `Ord` (`True` es mayor que `False`)

EXPRESSIVE TYPES (7)

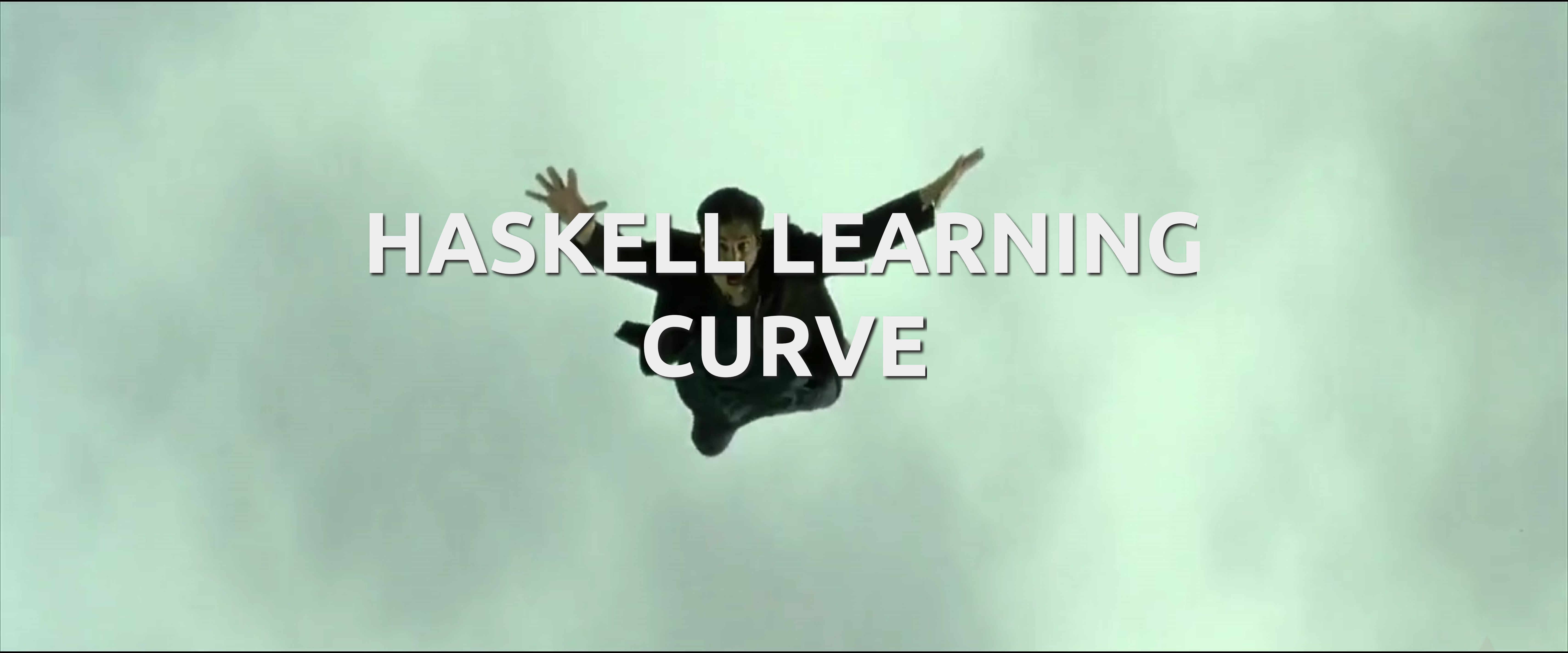
IT IS POSSIBLE TO CHECK AT COMPILE TIME

- Arrays bounds
- Open vs closed files
- Nested transactions
- Guaranteed closing resources
- REST endpoints
- *And test are available too!*

AND MANY MORE

WHAT HASKELL CAN LEARN

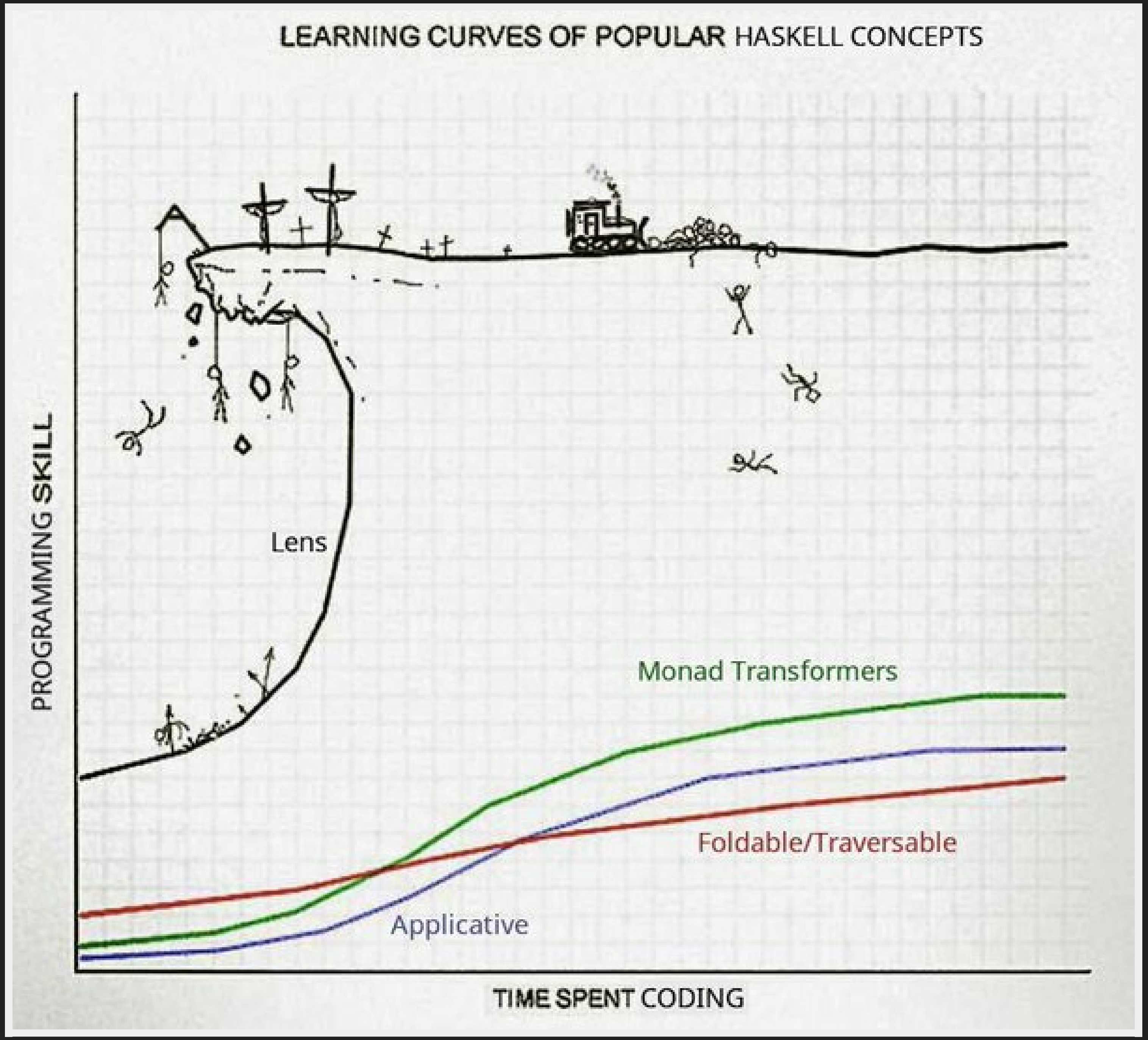
- Flat learning curve
- Intellij IDEA
- Stackoverflow
- ...



HASKELL LEARNING CURVE

LEARNING CURVE (1)

HASKELL LEARNING CURVE, COMPARISON



LEARNING CURVE (2)

THINGS TO LEARN IN HASKELL

1. Syntax, functions from Prelude
2. Monads
3. Concurrency & parallelism, STM
4. Libraries

---

1. Monad transformers
2. Free monads, recursive schemes
3. Arrows, Lens, Type families
4. Type safe DSLs, TH
5. *Whatever you want*

LEARNING CURVE (4)

COMPARE WITH C++

> *The new book released, the translation of the C++17 Standard Draft to Russian. 888 pages. You say, Haskell is too complex? Okay... @dshevchenko*

INTELLIJ IDEA

There is no analog for Haskell.

There are plugins/extensions for

- Atom
- Vim
- Emacs
- Sublime

However, Haskell's stepping debugger (GHCi) is not universal.

STACKOVERFLOW AND DOCS

For Java it is easy to find examples and good documentation.

For Haskell

- The documentation is often poor
- Needed to look into the libraries' code
- Fortunately I could ask my colleagues directly.

OUR CASE

OUR CASE

1. There were communication problems inside the team
2. There was a split between haskellers and javaists
3. ...Despite the pretty good quality of the services itself
4. The productivity would not save us :-/

OUR CASE

1. There were communication problems inside the team
2. There was a split between haskellers and javaists
3. ...Despite the pretty good quality of the services itself
4. The productivity would not save us :-/

One should take the social aspects into account during the introduction of the new technologies.

SO?

IS IT WORTH TO USE HASKELL IN PRODUCTION?

1. Only if all team members are eager to learn Haskell
2. Maybe for some separate task, e.g. compiler
3. I believe one can grow a team of Haskellers
4. .. but cannot easy switch team to Haskell (you cannot force people to learn)

Examples: GHC, Corrode, Elm, PureScript, Agda, Kaleidoscope, Pandoc

---

Haskell is not for Production and Other Tales, Katie Miller
Video and Slides

IS IT WORTH TO LEARN HASKELL? (1)

YES!

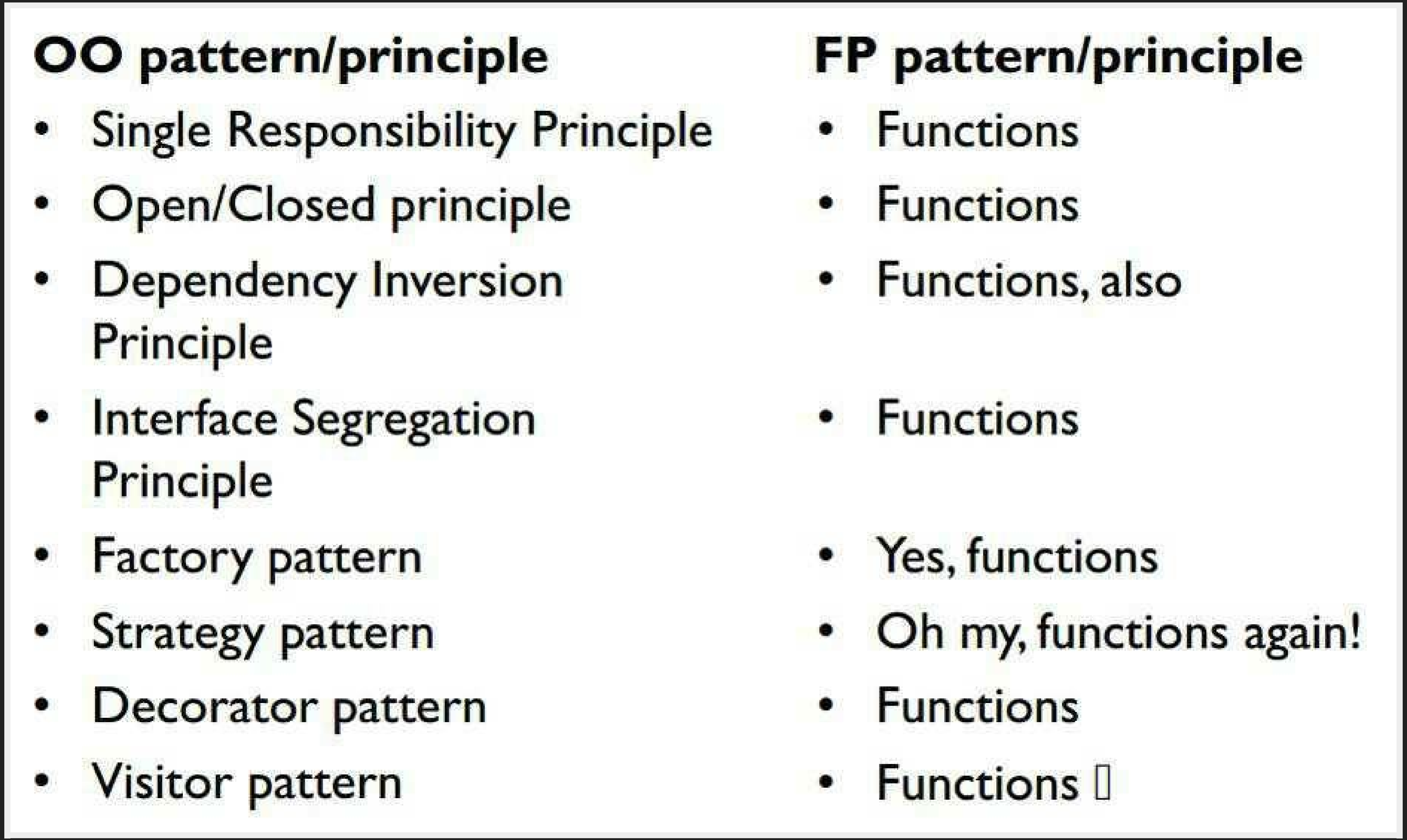
1. to get a new way of thinking and to push the boundaries
2. to know how to structure things without inheritance
3. to know the alternatives to the buzzwords: DDD, Anemic, Patterns, IOC, SOLID, DI, ...
4. to finally understand monads

IS IT WORTH TO LEARN HASKELL? (2)

OO is full of design problems

- is hard to get it done right
- a lot of buzzwords
- as opposite to algorithms there is no clear criteria whether is one solution better than another
  - cow.eat(grass)
  - grass.beEatenBy(cow)
  - field.eatingInteraction(cow, grass)

IS IT WORTH TO LEARN HASKELL? (2)



OO pattern/principle	FP pattern/principle
Single Responsibility Principle	Functions
Open/Closed principle	Functions
Dependency Inversion Principle	Functions, also
Interface Segregation Principle	Functions
Factory pattern	Yes, functions
Strategy pattern	Oh my, functions again!
Decorator pattern	Functions
Visitor pattern	Functions ▯

QUESTIONS?