JetBRAINS

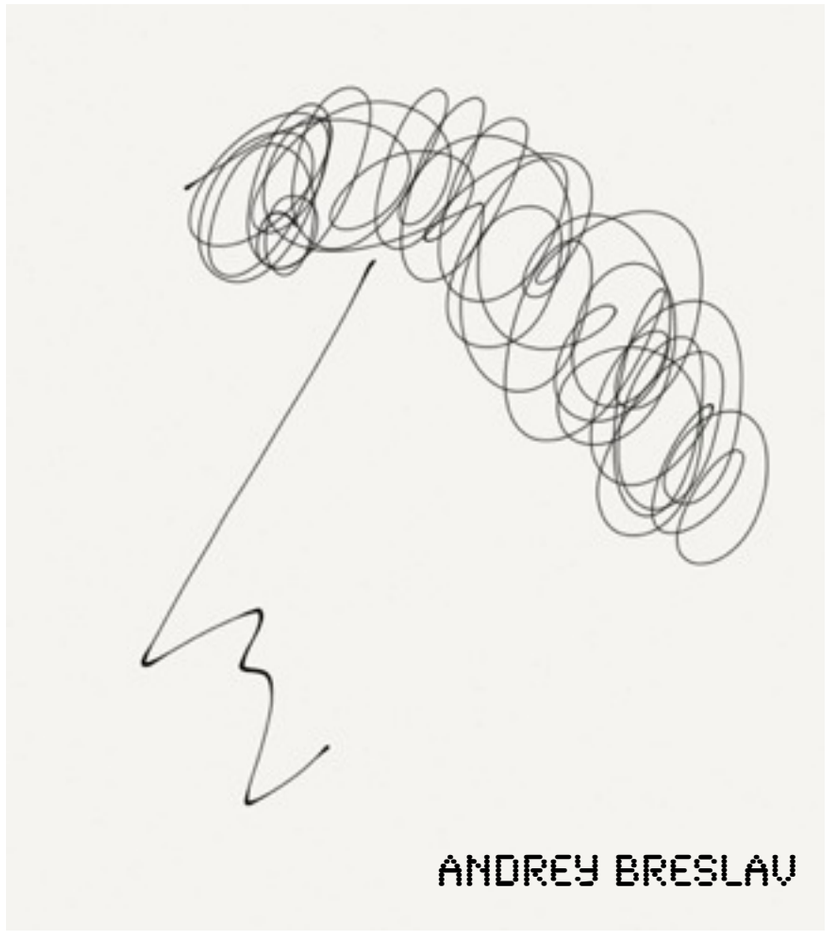
Who's More Functional

Groovy, Kotlin, Scala or Java?

Andrey Breslav

Kotlin

About Me



- Project lead of **Kotlin**
  - ➡ at JetBrains since 2010
- EG member of JSR-335
  - ➡ Project **Lambda**

Prologue

Prologue

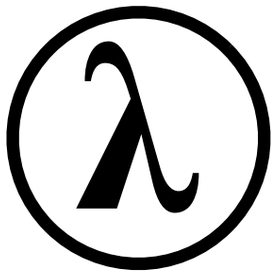
- Dad (or Mom), is Java a functional language?

Prologue

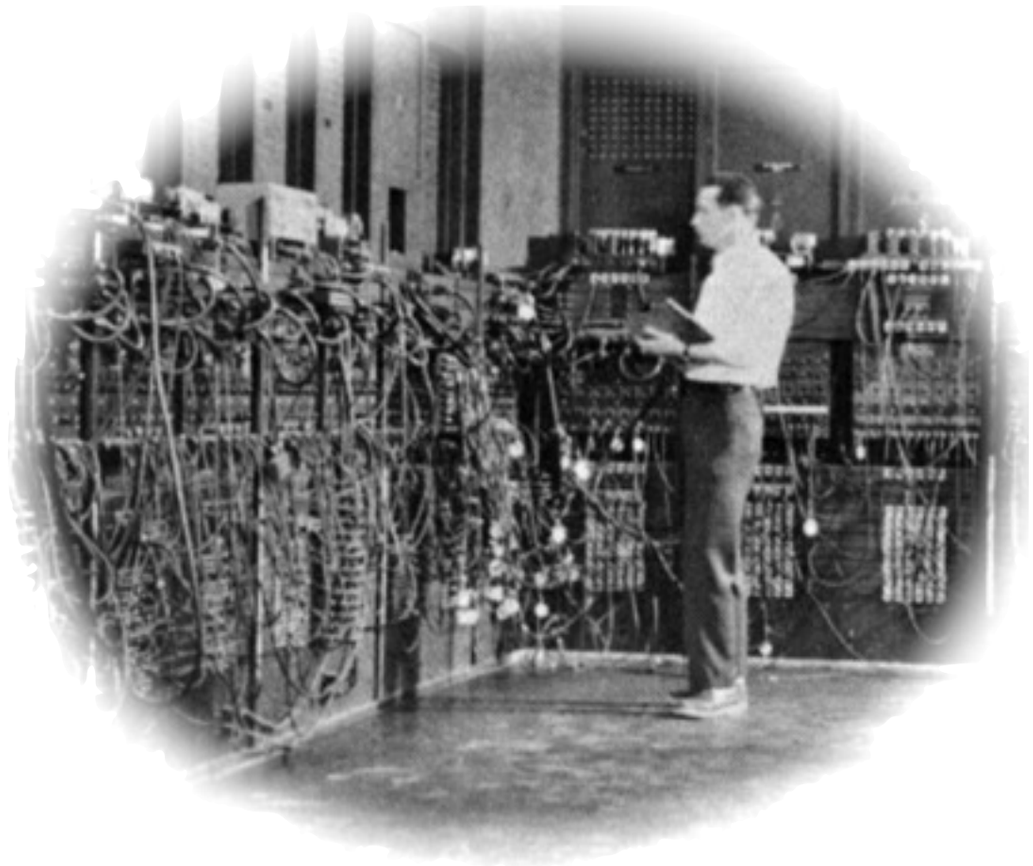
- Dad (or Mom), is Java a functional language?
- Don't you know your dad from your mom?!

What is FP like?

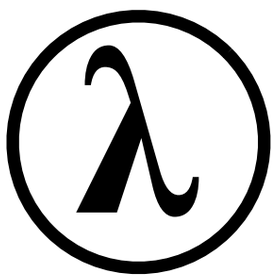
Alonzo Church



1936

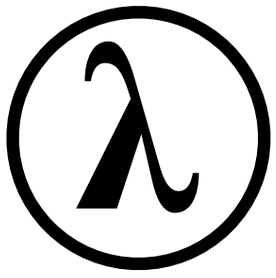


Alonzo Church

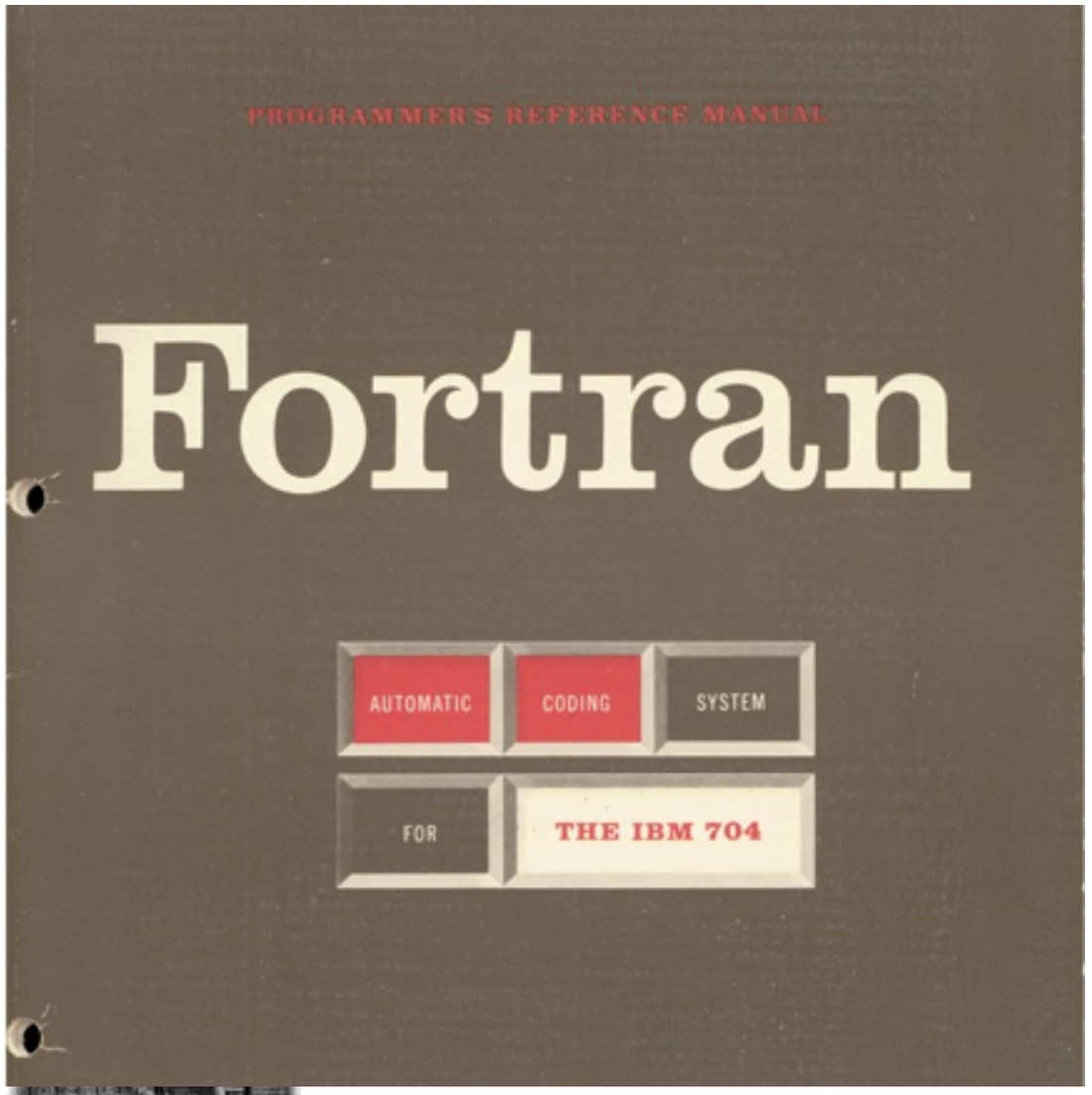


1936 1946

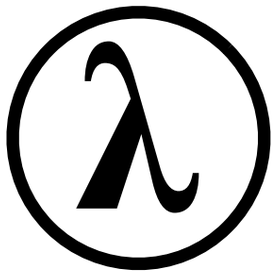
Alonzo Church



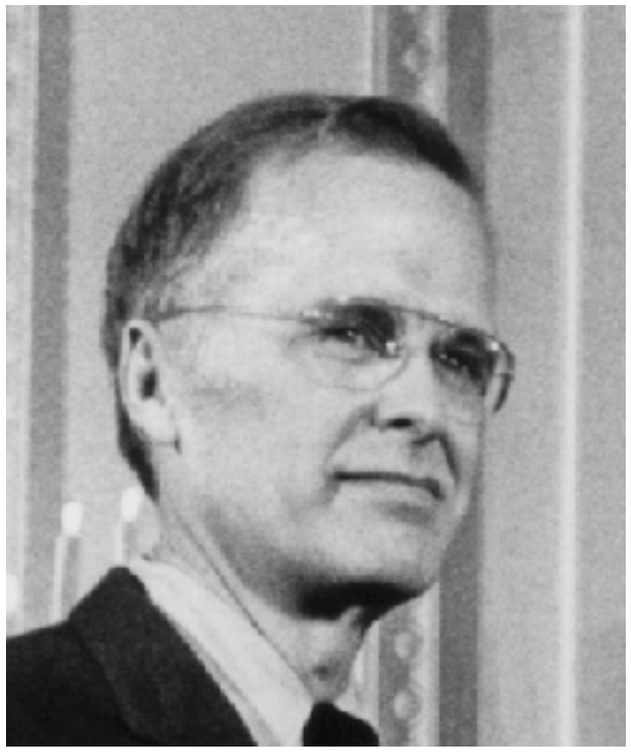
1936 1946



Alonzo Church

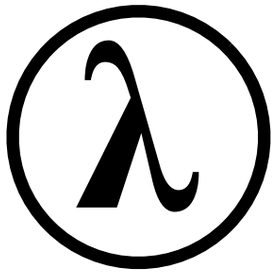


1936 1946 1956

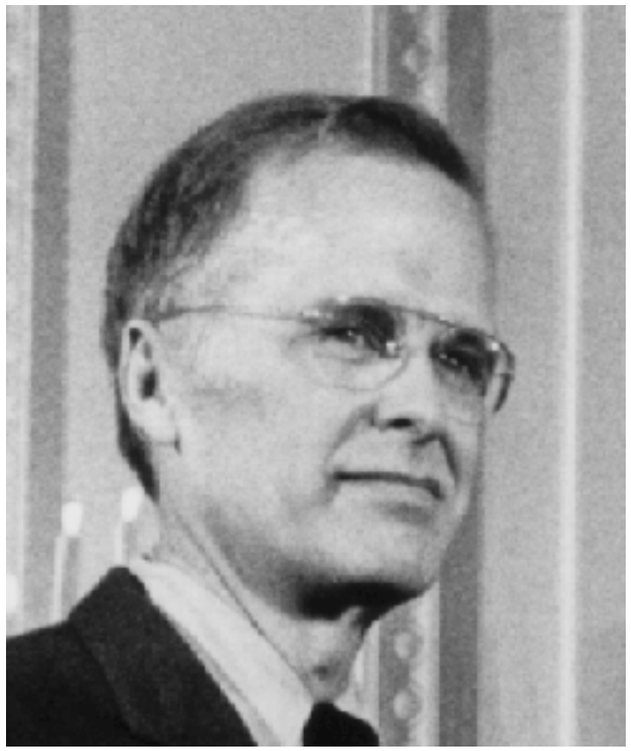


John Backus

Alonzo Church



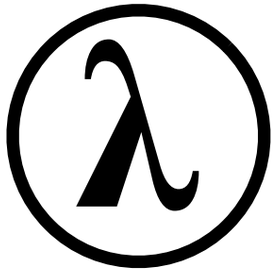
1936 1946 1956



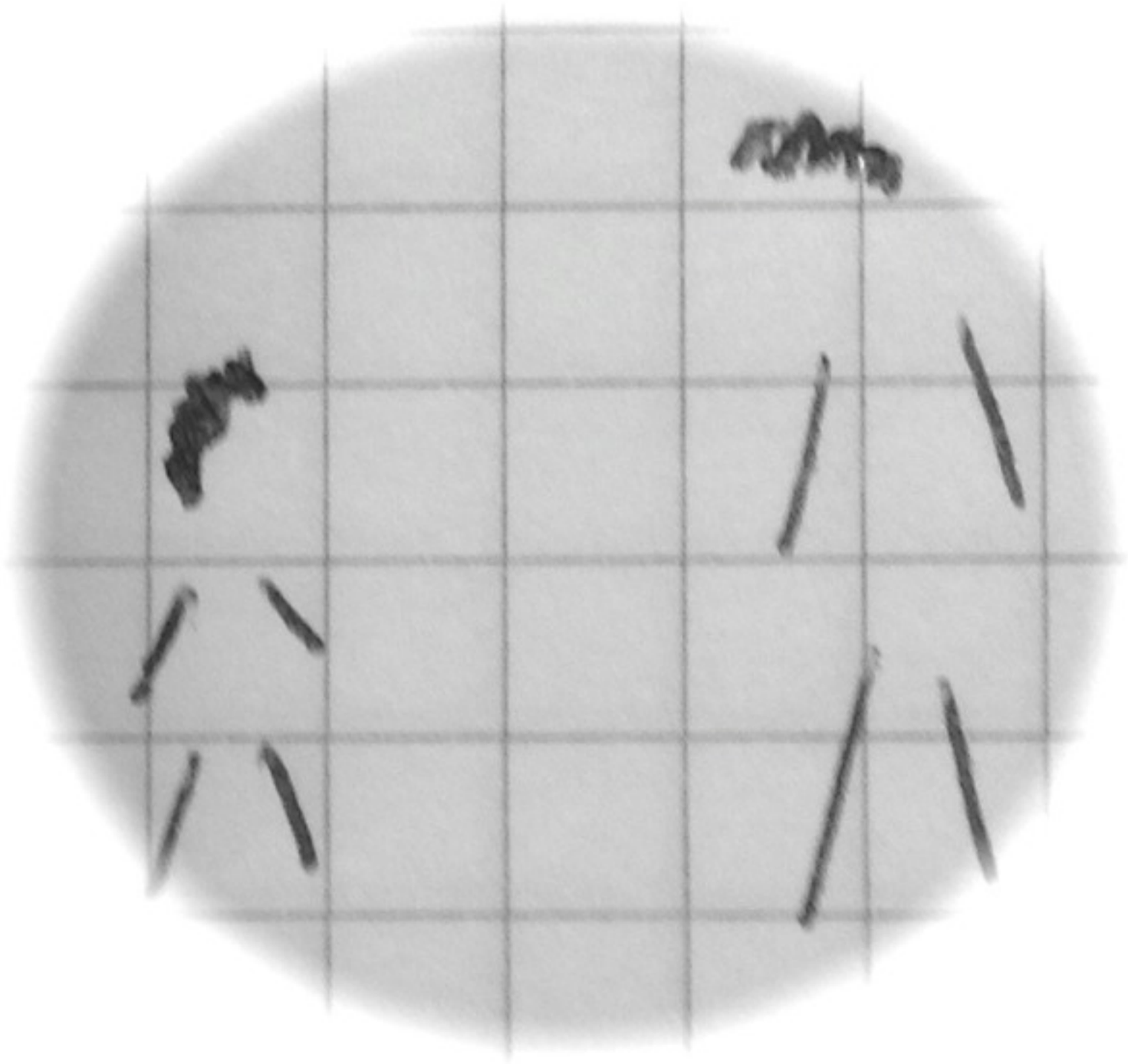
John Backus

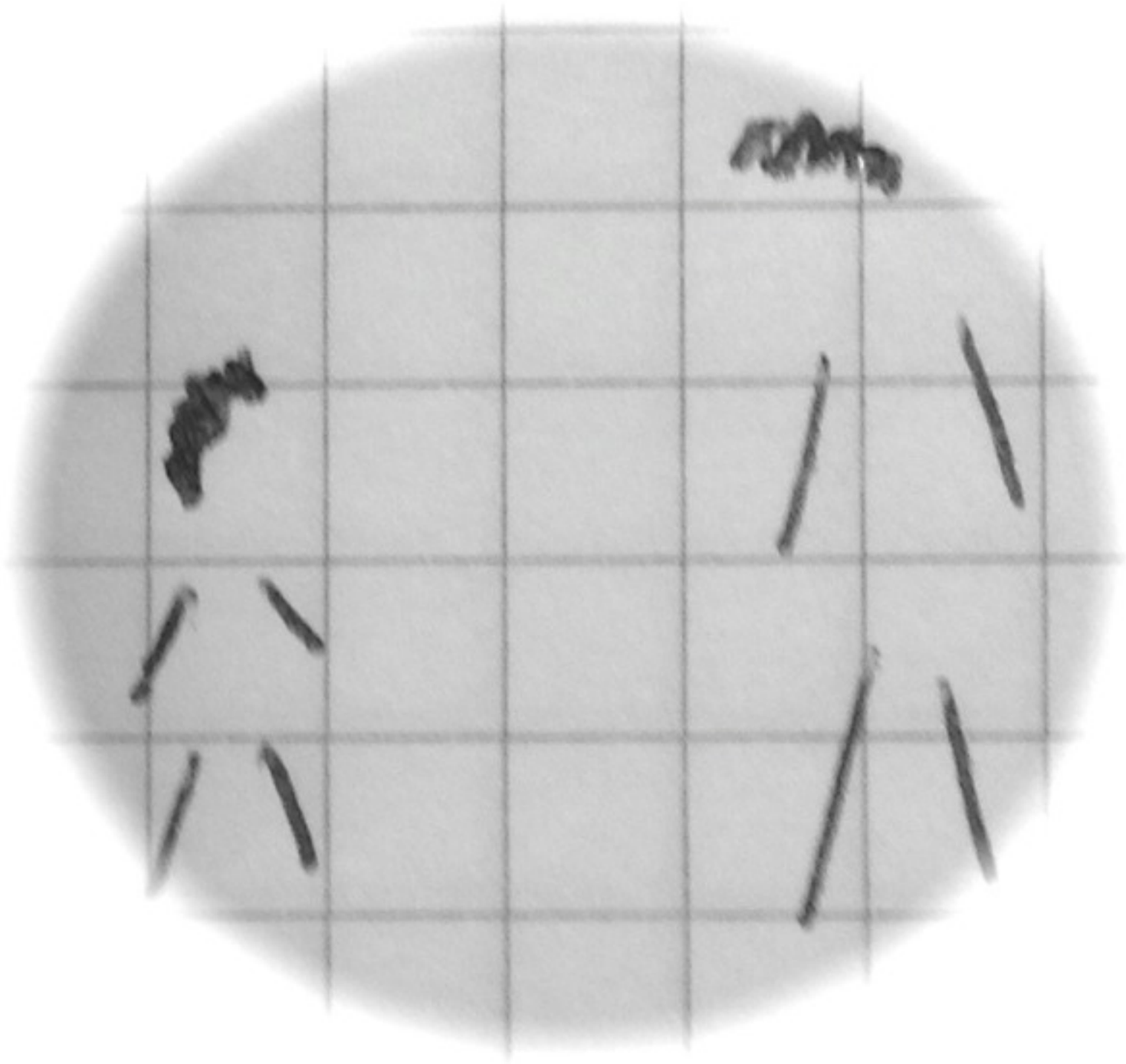
Can Programming be Liberated from the von Neumann Style?

Alonzo Church

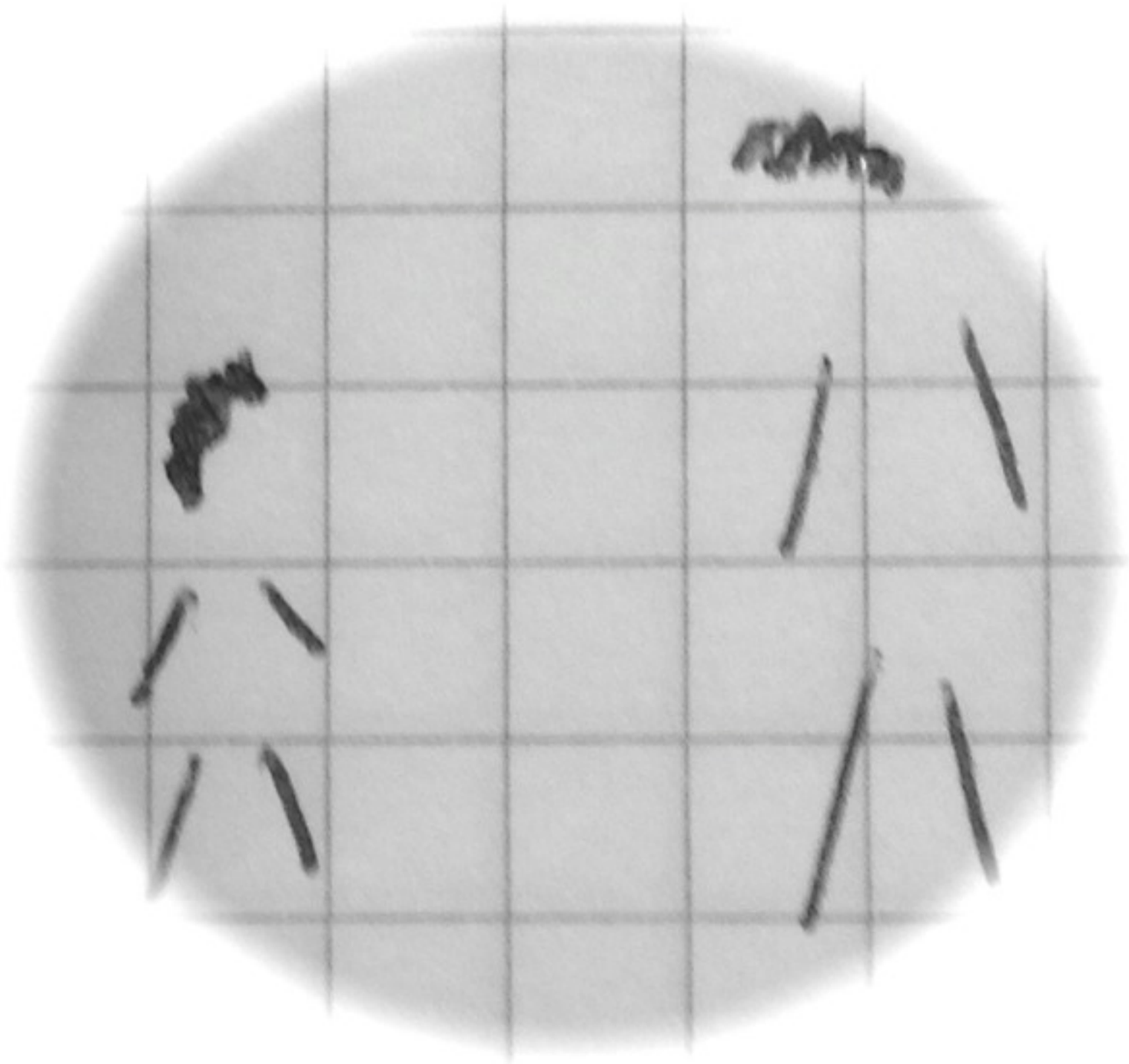


1936 1946 1956 1976



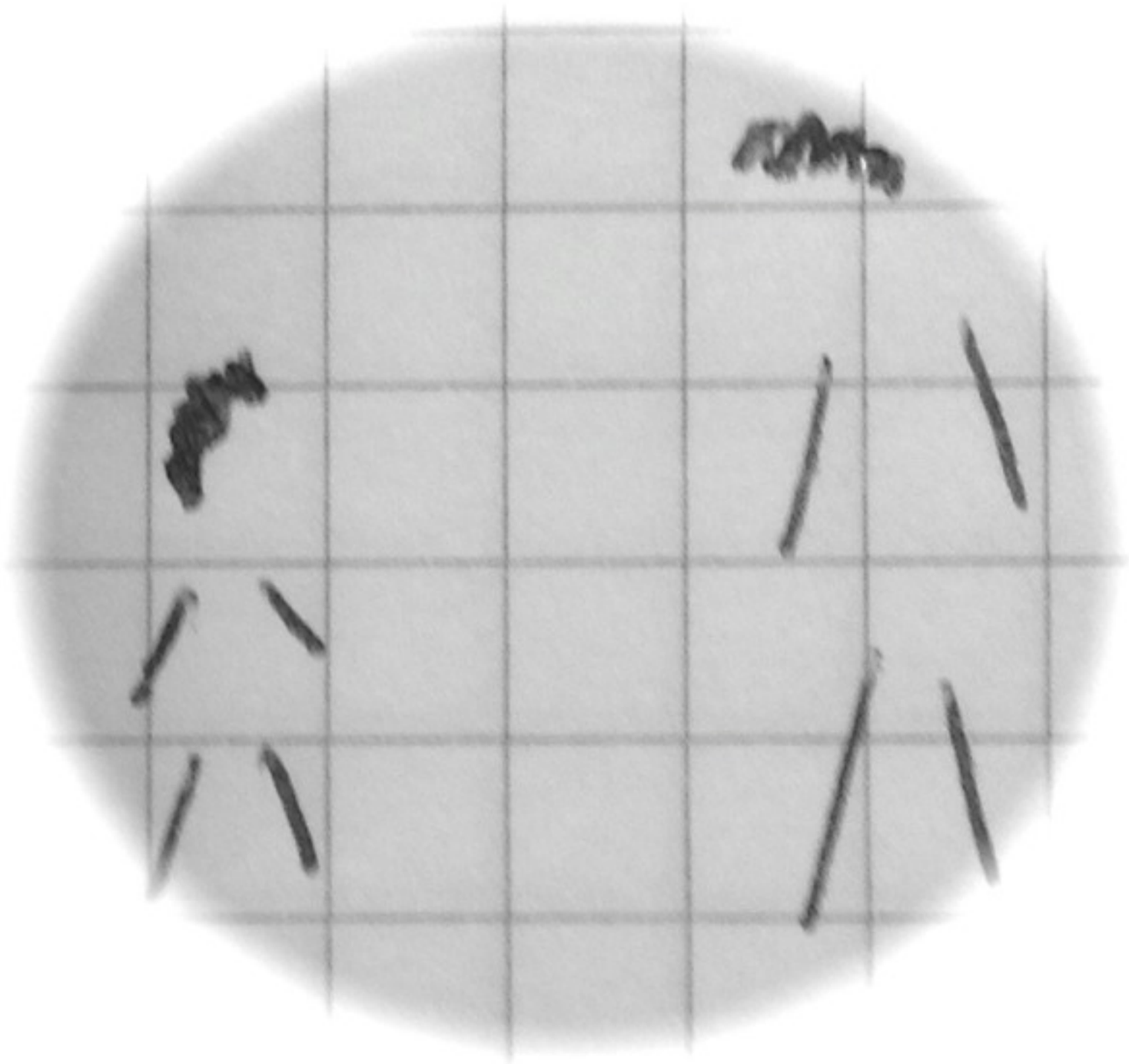


— Dad, what's good about FP?



— Dad, what's good about FP?

— It makes you look smart



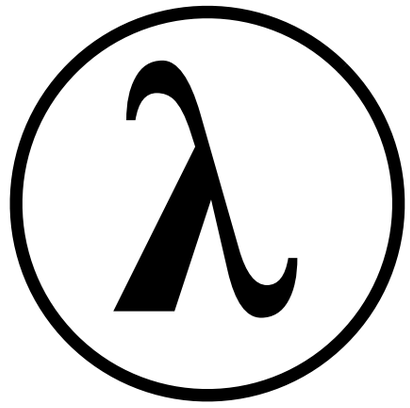
— Dad, what's good about FP?

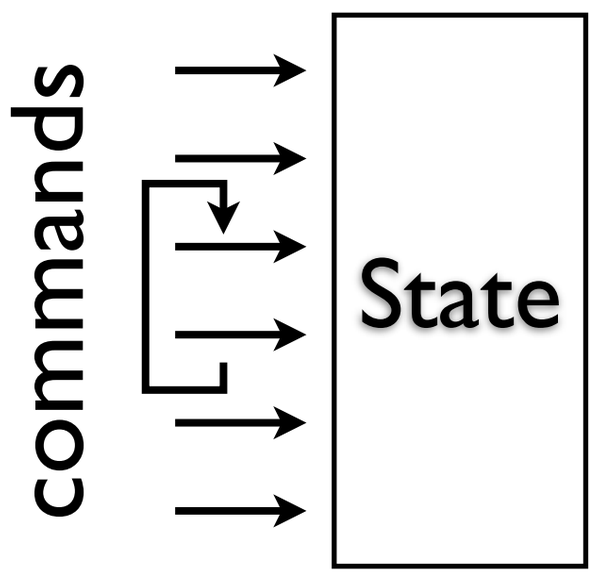
— It makes you look smart

— Like wearing glasses?

Effects (Mutability)

John von Neumann





VS

Inputs

$f(g(x), y)$

Outputs

JetBRAINS

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Leonardo Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
fun fib(n: Int): Int =
    when (n) {
        0, 1 -> 1
        else -> fib(n - 1) + fib(n - 2)
    }
```

Leonardo Fibonacci

1 1 2 3 5 8 13 21 34 .

fun

o Fibonacci

Are you functional?

Kotlin ✓

Groovy ✓

Scala ✓

Java 8 ✓

Recursion

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

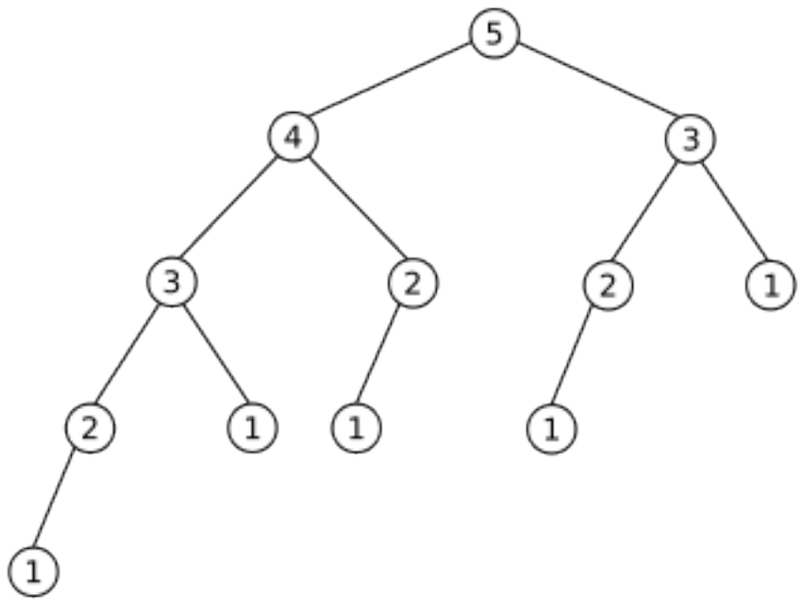
```
fun fib(n: Int): Int =
    when (n) {
        0, 1 -> 1
        else -> fib(n - 1) + fib(n - 2)
    }
```

Leonardo Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
fun fib(n: Int): Int =
    when (n) {
        0, 1 -> 1
        else -> fib(n - 1) + fib(n - 2)
    }
```

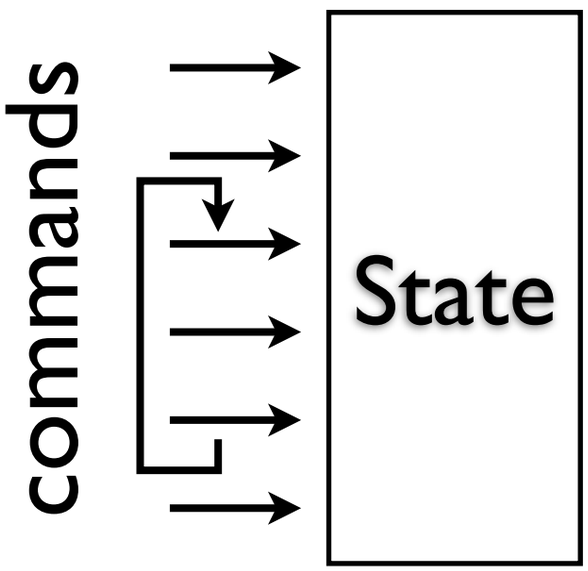
Leonardo Fibonacci



1, 1, 2, 3, 5, 8, 13, 21, 34, ...

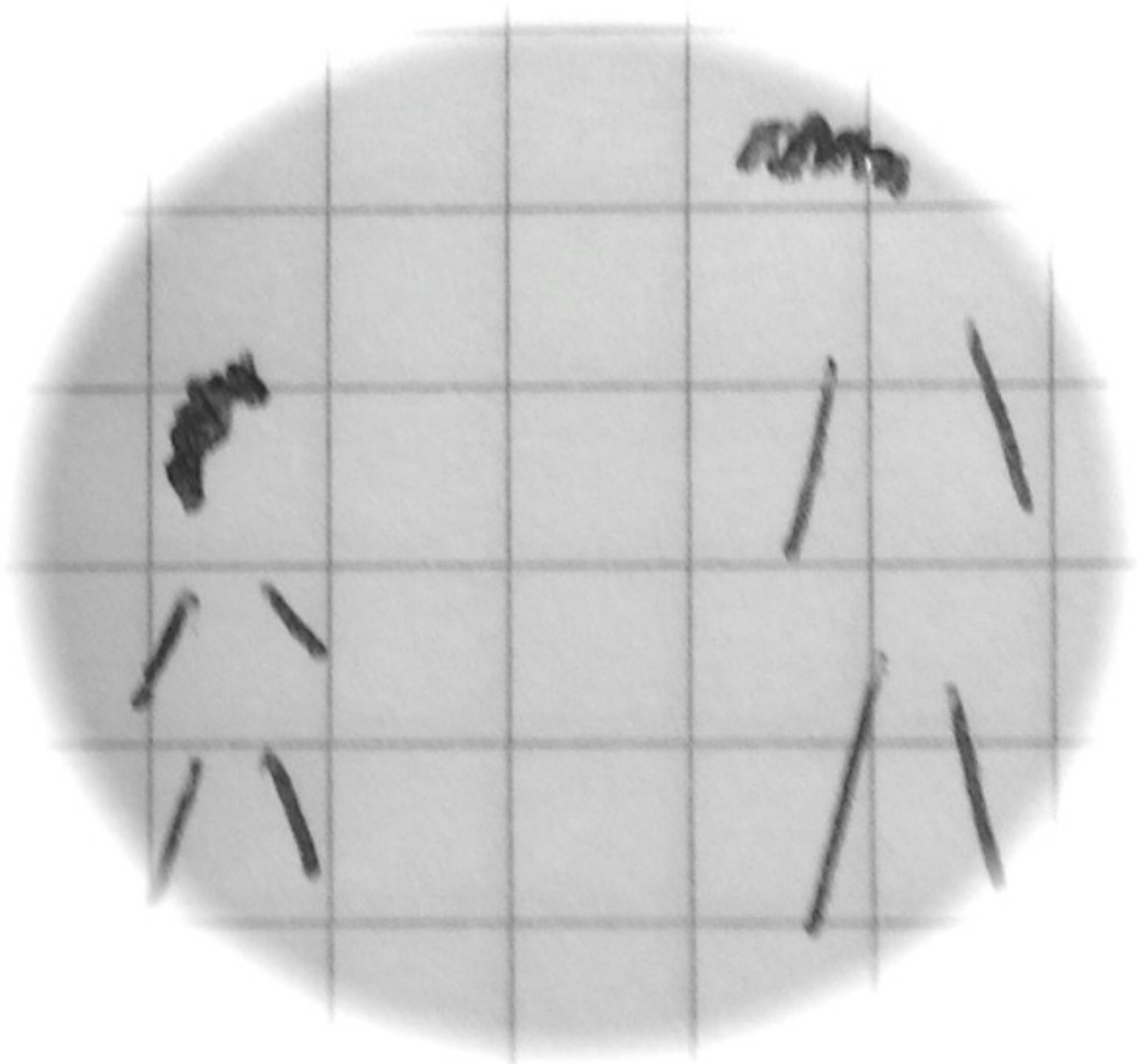
```
fun fib_imp(n: Int): Int {
    var a = 1
    var b = 1
    for (i in 1..n) {
        val t = a + b
        a = b
        b = t
    }
    return a
}
```

Leonardo Fibonacci

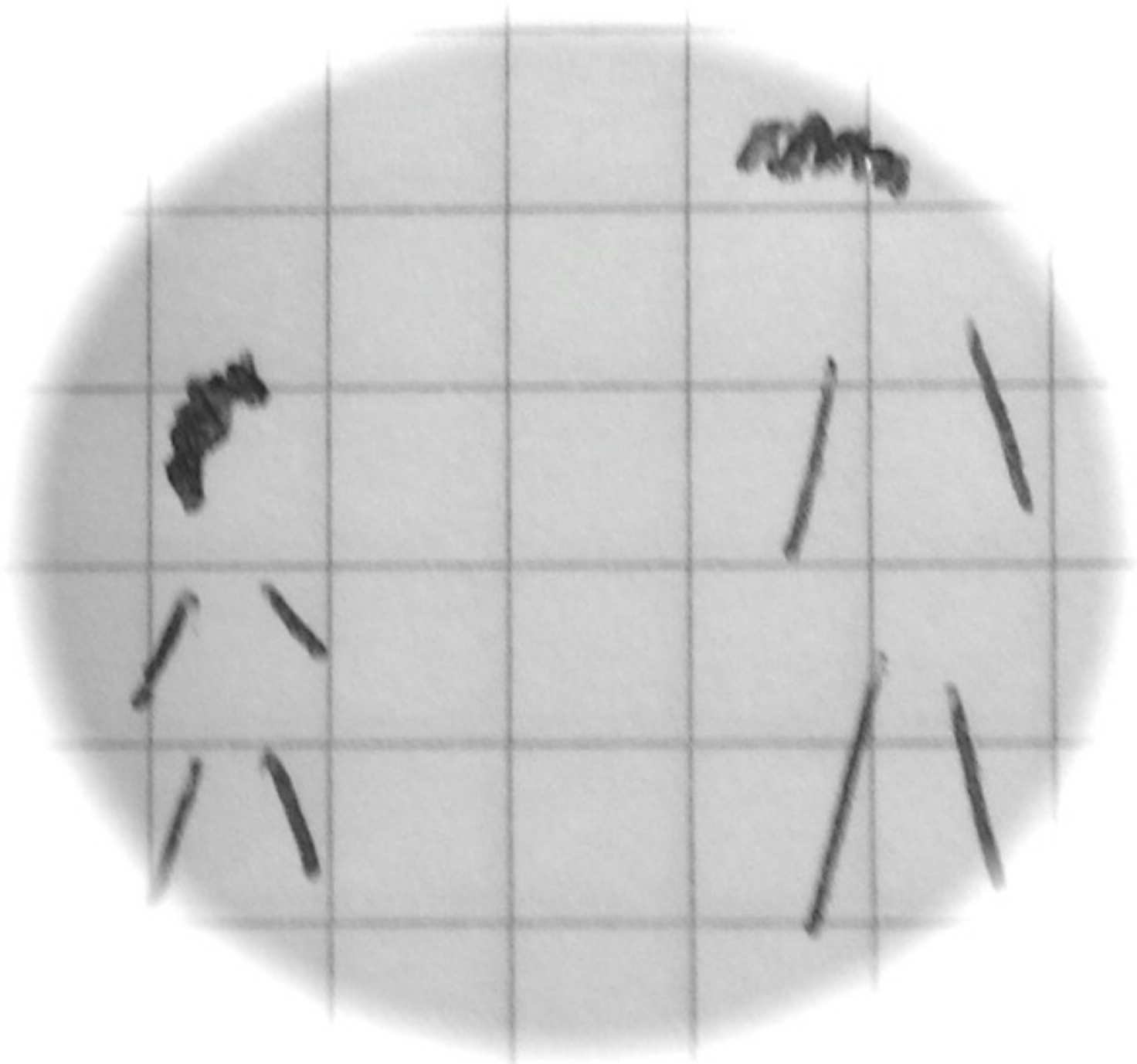


Effect-Free

Effect-Free

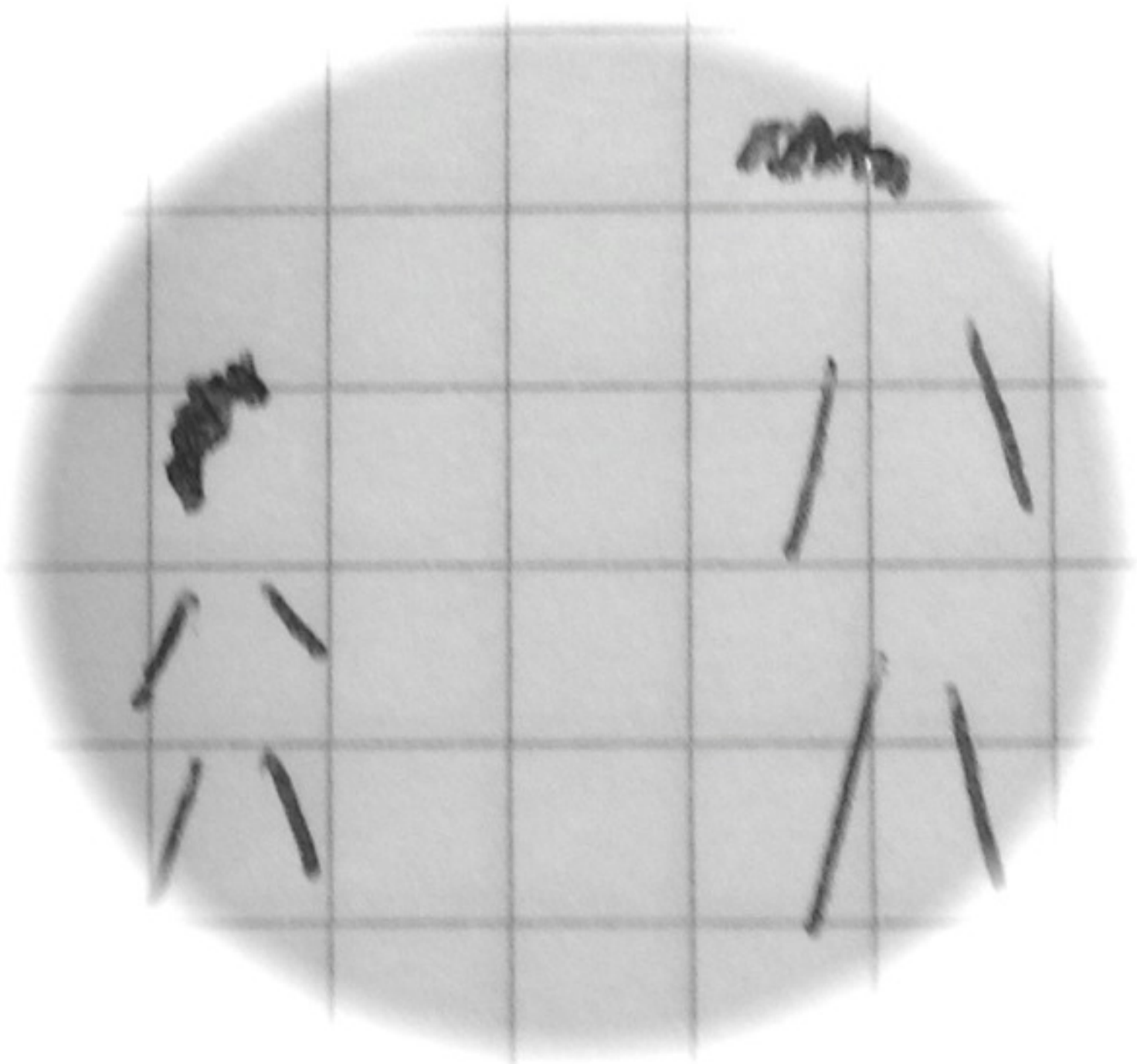


Effect-Free



— Dad, how do I

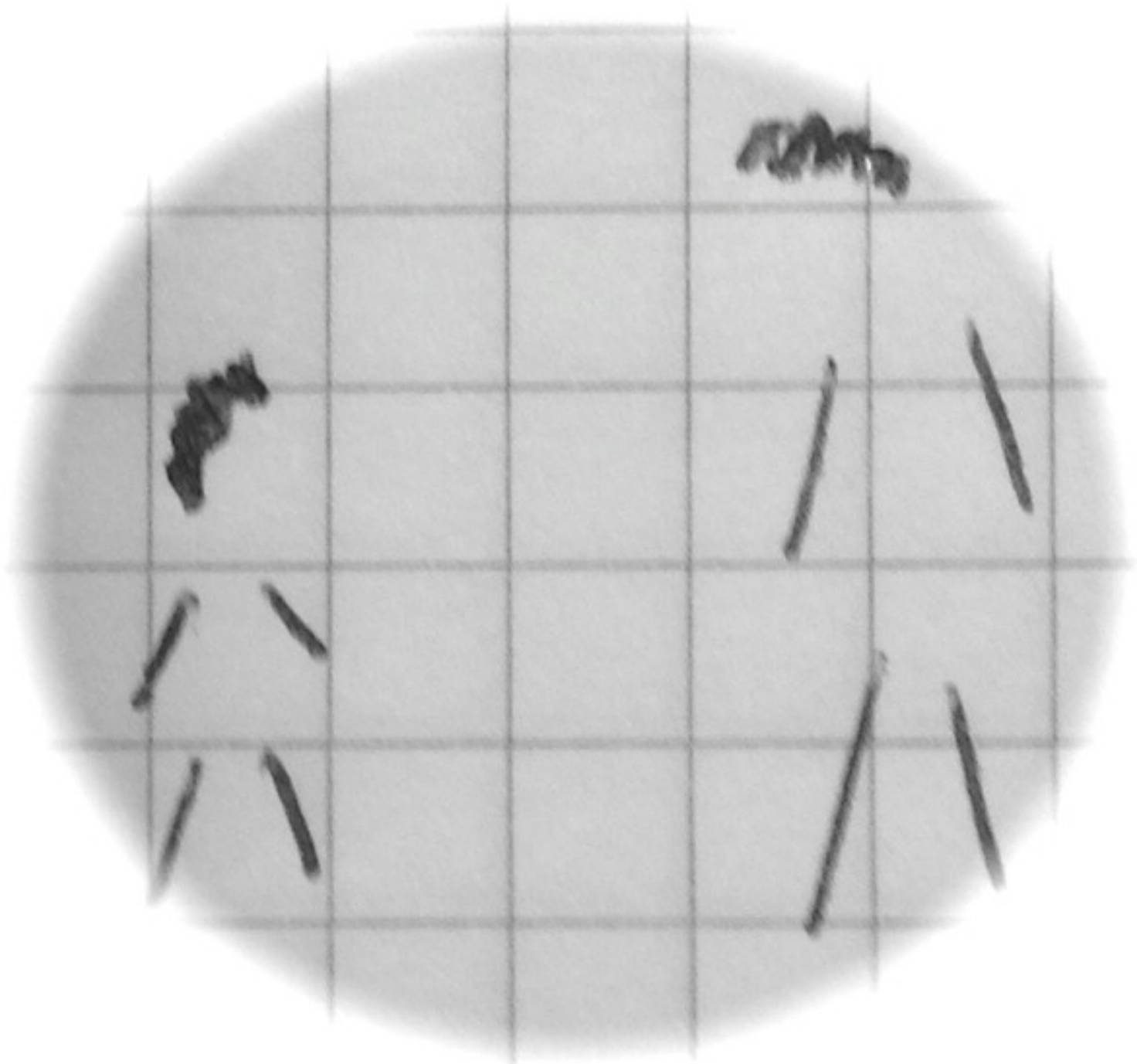
Effect-Free



— Dad, how do I

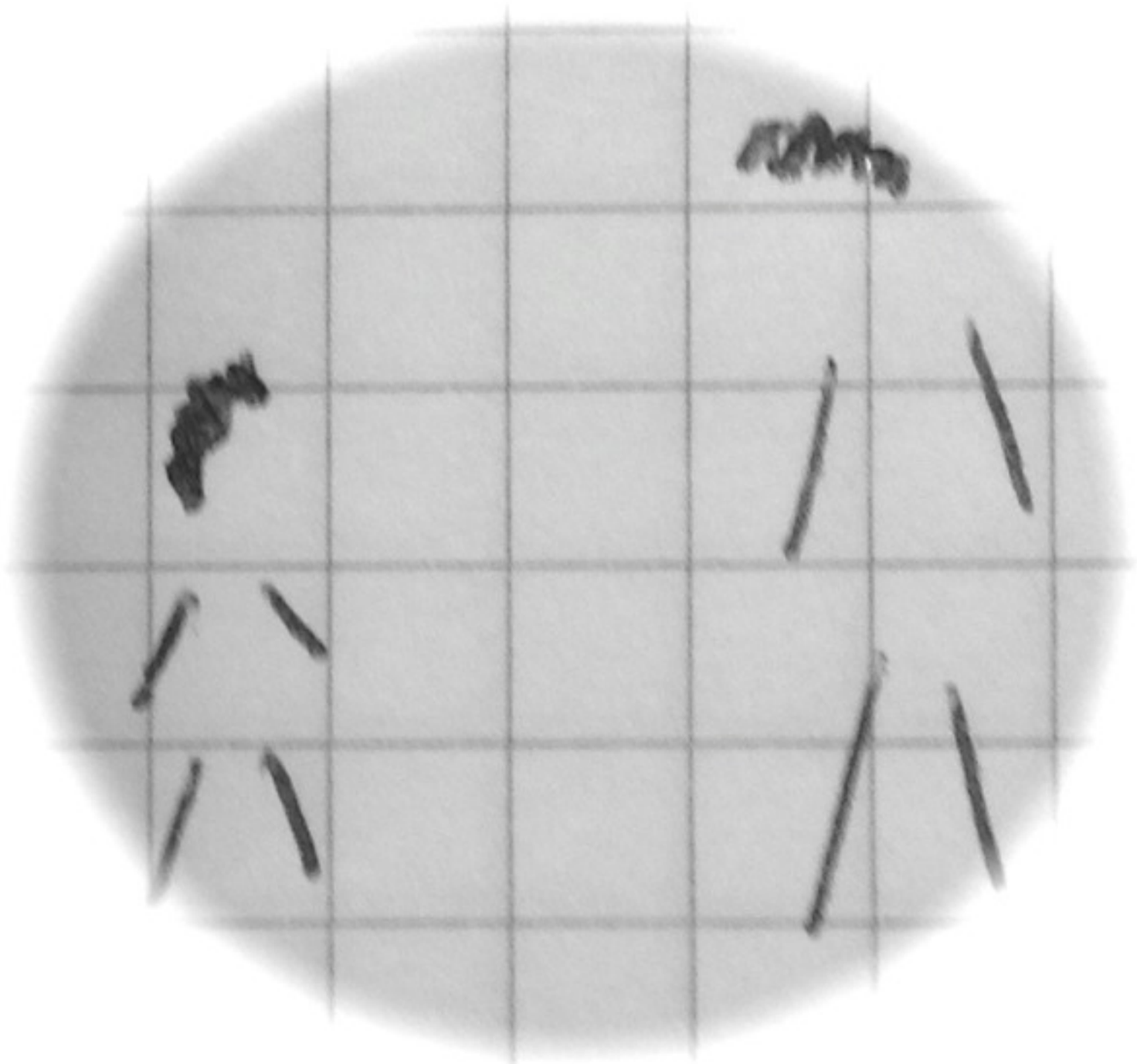
- print("Hello")?

Effect-Free



- Dad, how do I
  - print("Hello")?
  - write to a file?

Effect-Free



- Dad, how do I
  - print("Hello")?
  - write to a file?
  - do both?

Effect-Free

Are you functional?

	Recursion	Pure
Kotlin	V	X
Groovy	V	X
Scala	V	X
Java 8	V	X

v do I

ello")?

a file?

?

Summary

- FP makes things simpler
- Sometimes at a huge price

- Our languages are **not purely functional**

Higher Order

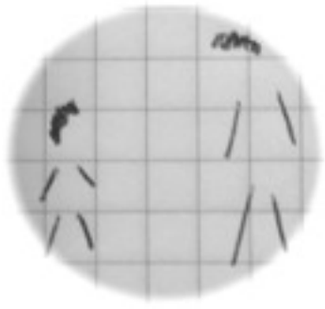
FP brings order :)

FibonacciTest

```
assertEquals(1, fib(0))
assertEquals(1, fib(1))
assertEquals(2, fib(2))
assertEquals(3, fib(3))
assertEquals(5, fib(4))
assertEquals(8, fib(5))
assertEquals(13, fib(6))
assertEquals(21, fib(7))
```

FibonacciTest

```
assertEquals(1, fib(0))
assertEquals(1, fib(1))
assertEquals(2, fib(2))
assertEquals(3, fib(3))
assertEquals(5, fib(4))
assertEquals(8, fib(5))
assertEquals(13, fib(6))
assertEquals(21, fib(7))
```



– How do I test both implementations?

test(f)

```
fun testFib(fib: (Int) -> Int) {

    assertEquals(1, fib(0))
    assertEquals(1, fib(1))
    assertEquals(2, fib(2))
    assertEquals(3, fib(3))
    assertEquals(5, fib(4))
    assertEquals(8, fib(5))
    assertEquals(13, fib(6))
    assertEquals(21, fib(7))

}


testFib({ n -> fib(n) })
testFib { n -> fib_imp(n) }
```

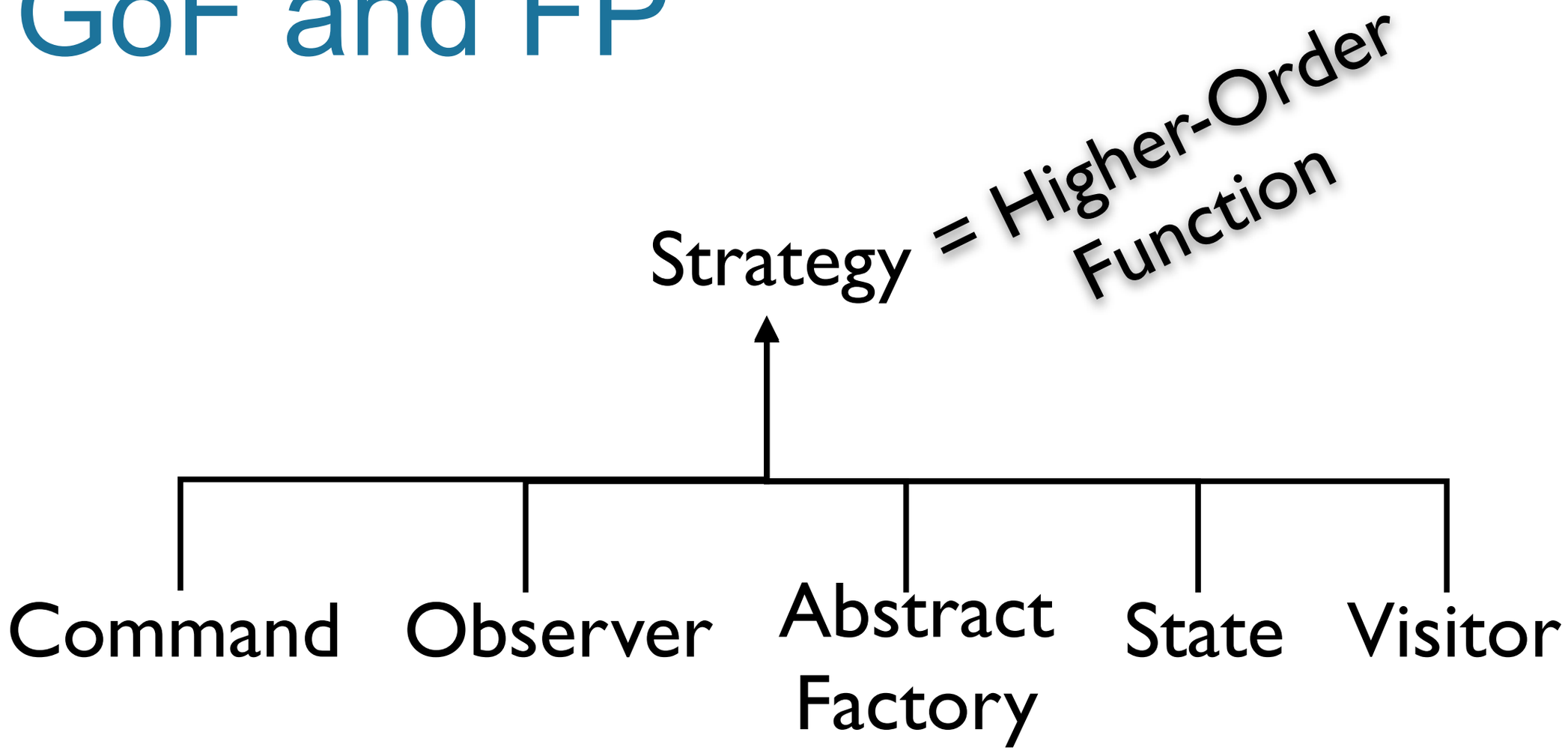
test(f)

Are you functional?

	Recursion	Pure	HO
Kotlin	V	X	V
Groovy	V	X	V
Scala	V	X	V
Java 8	V	X	V

```
testFib({ n -> fib(n) })
testFib { n -> fib_imp(n) }
```

GoF and FP


Strategy ≡ Higher-Order Function
Command
Observer
Abstract Factory
State
Visitor

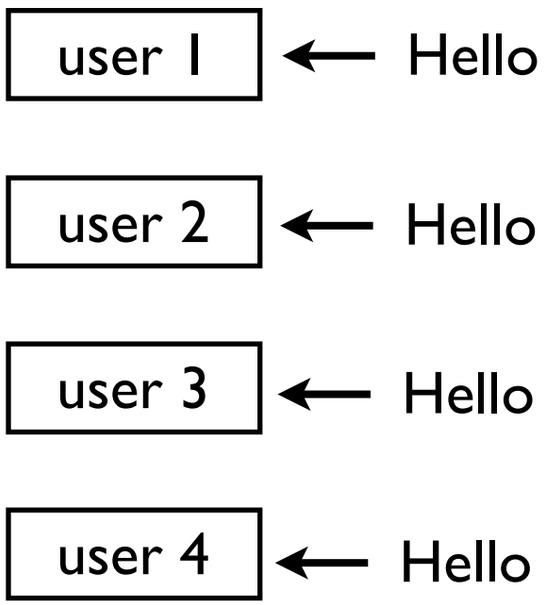
Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

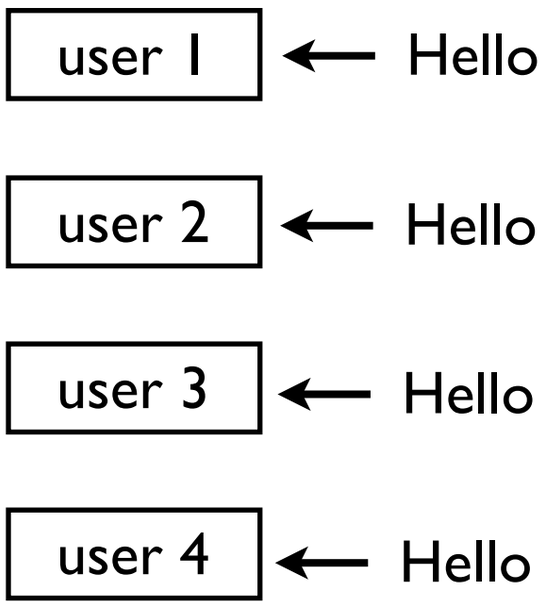
users: ArrayList



Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

users: ArrayList

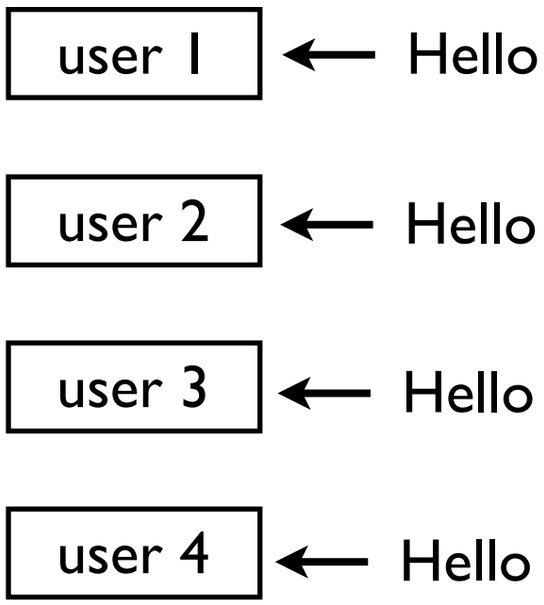


users: ParallelCollection

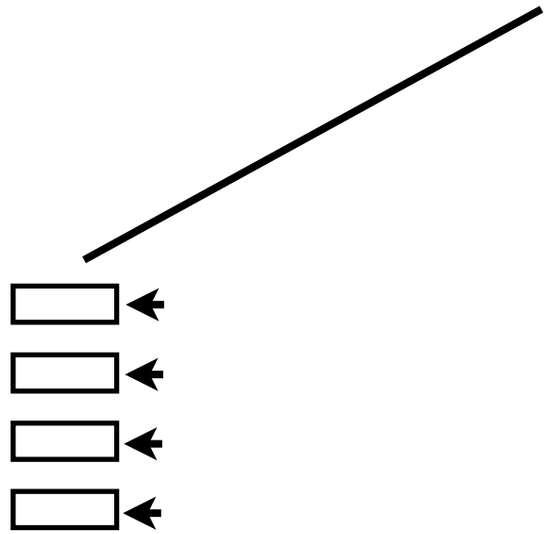
Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

users: ArrayList



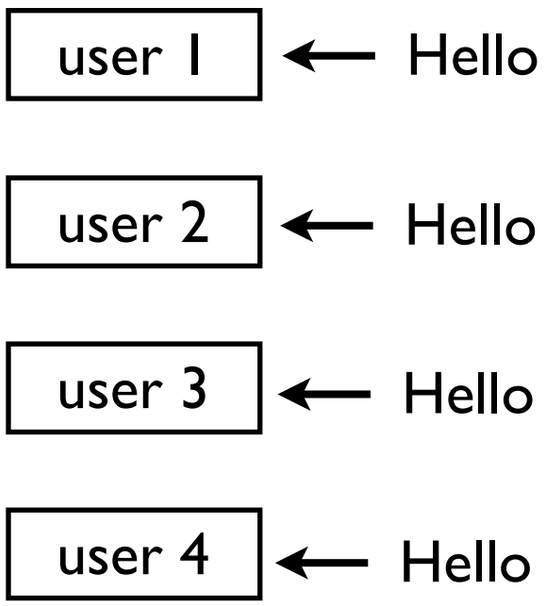
users: ParallelCollection



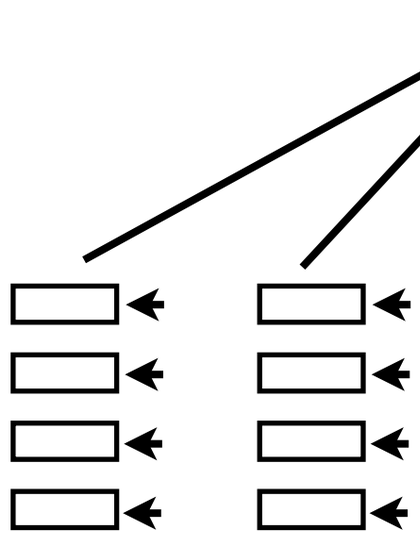
Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

users: ArrayList



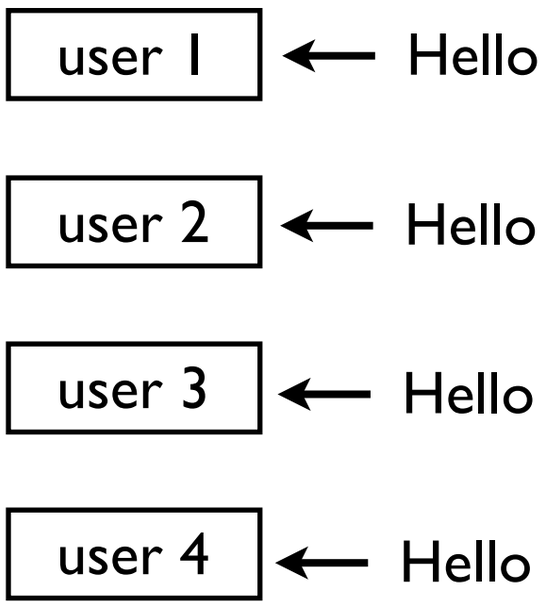
users: ParallelCollection



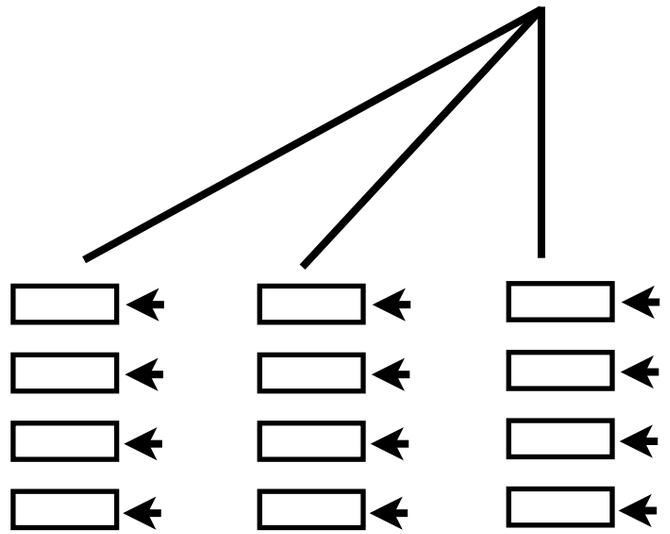
Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

users: ArrayList



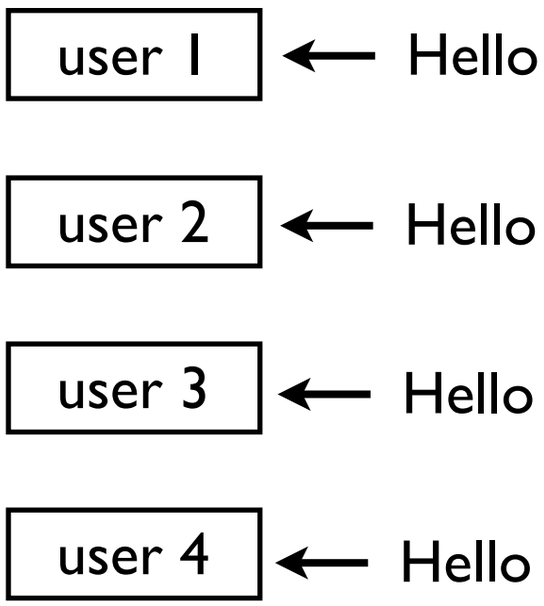
users: ParallelCollection



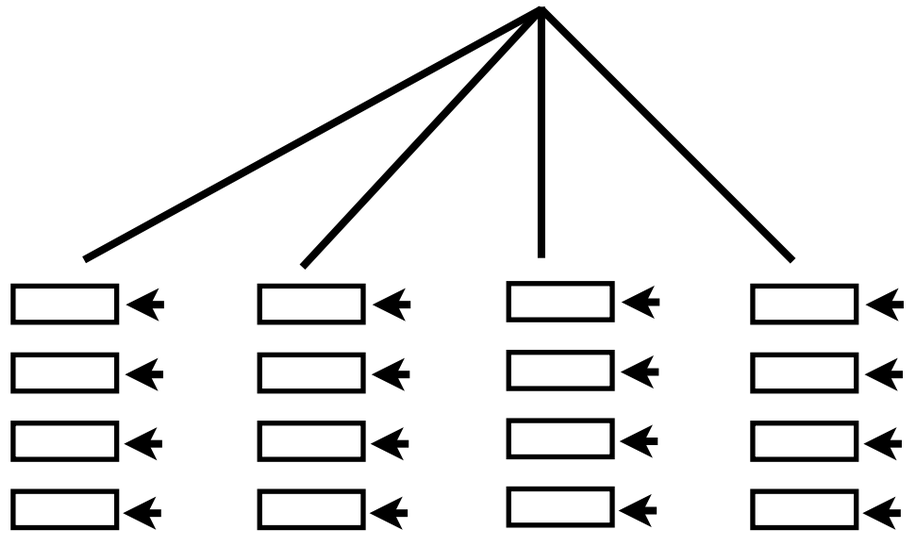
Internal Iteration

```
users.forEach { user ->
    user.sendMessage("Hello from admins!")
}
```

users: ArrayList



users: ParallelCollection



Summary

- Good old callbacks/strategies
- Very important abstraction

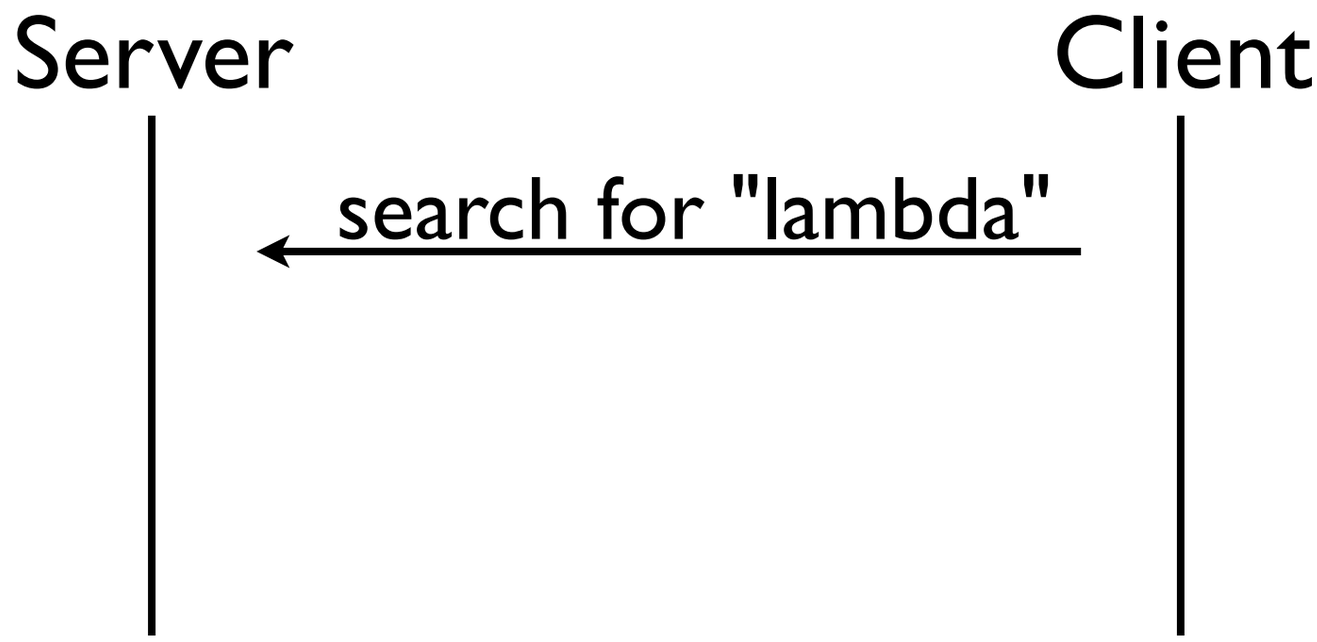
JetBRAINS

ADT

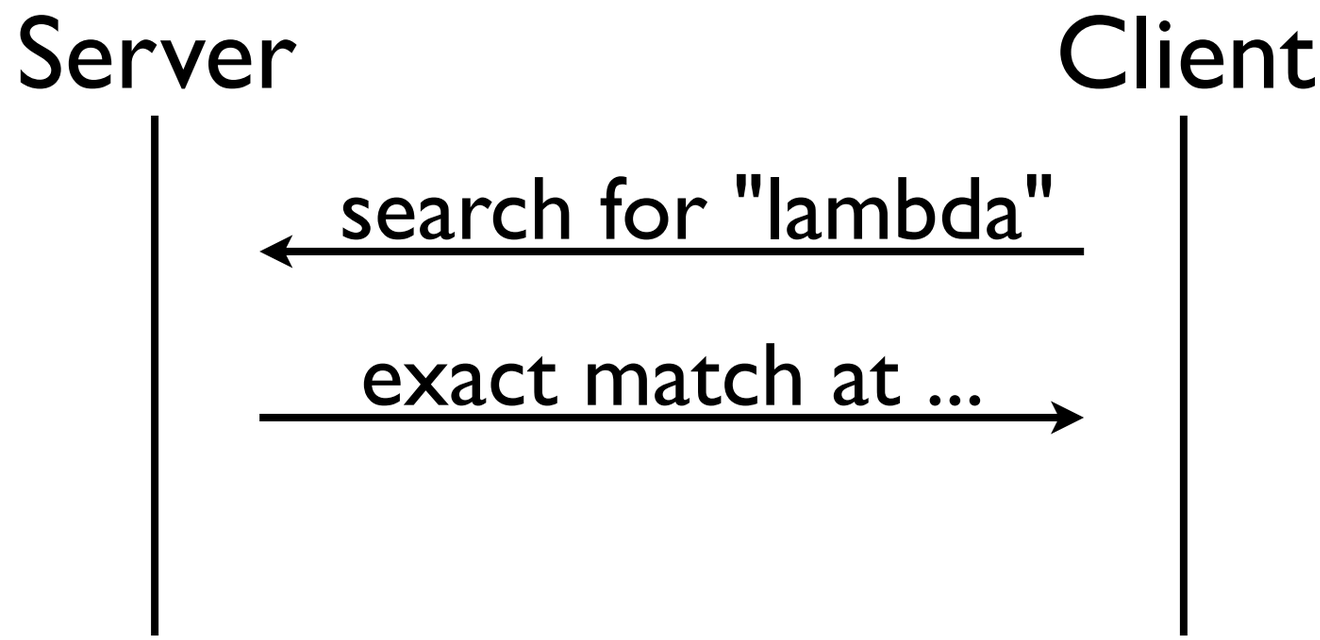
What is your "A" for?

Example: Messages

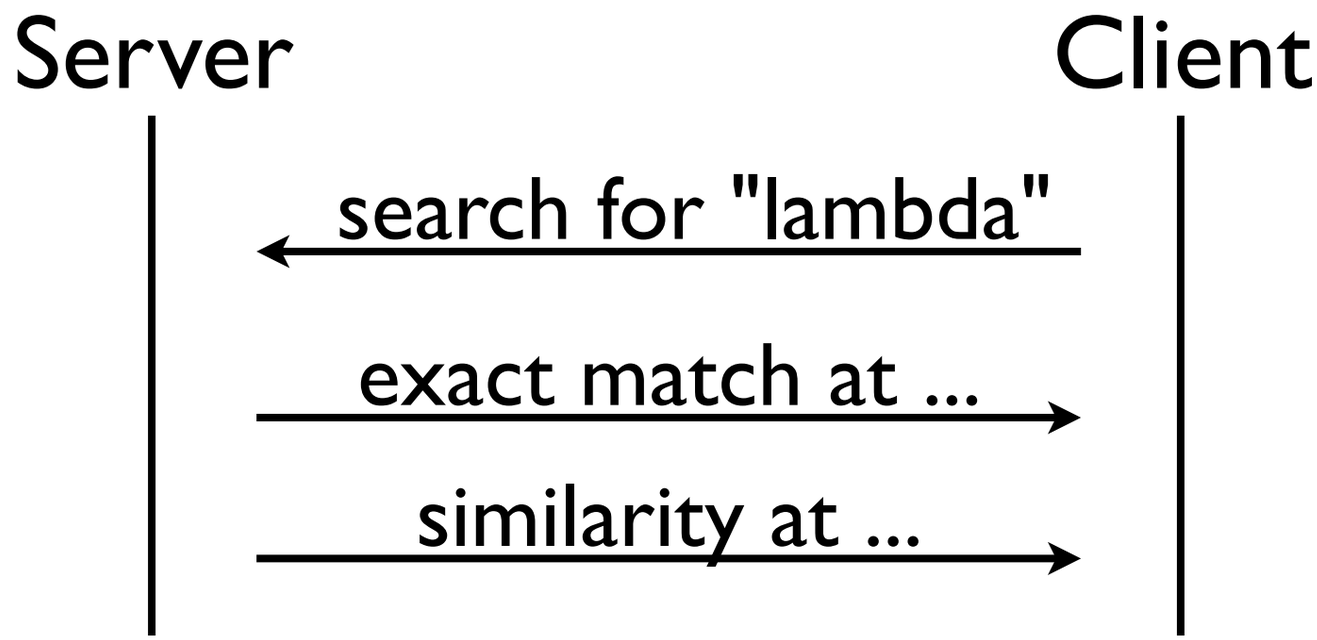
Example: Messages



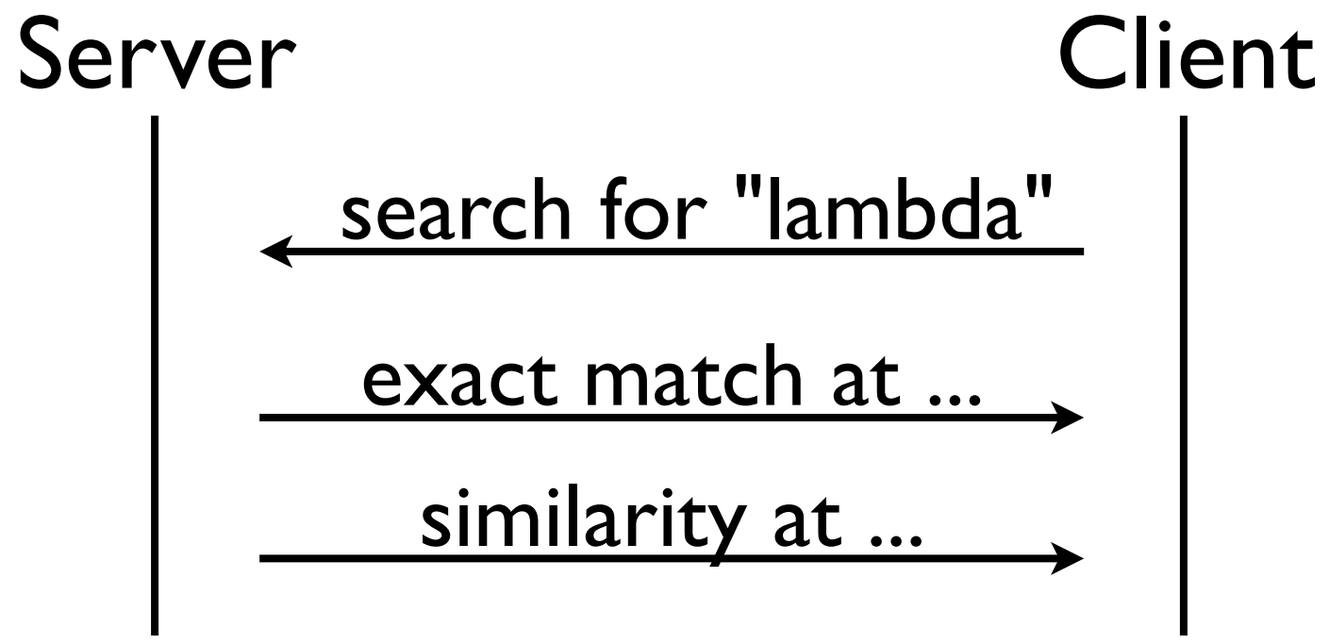
Example: Messages



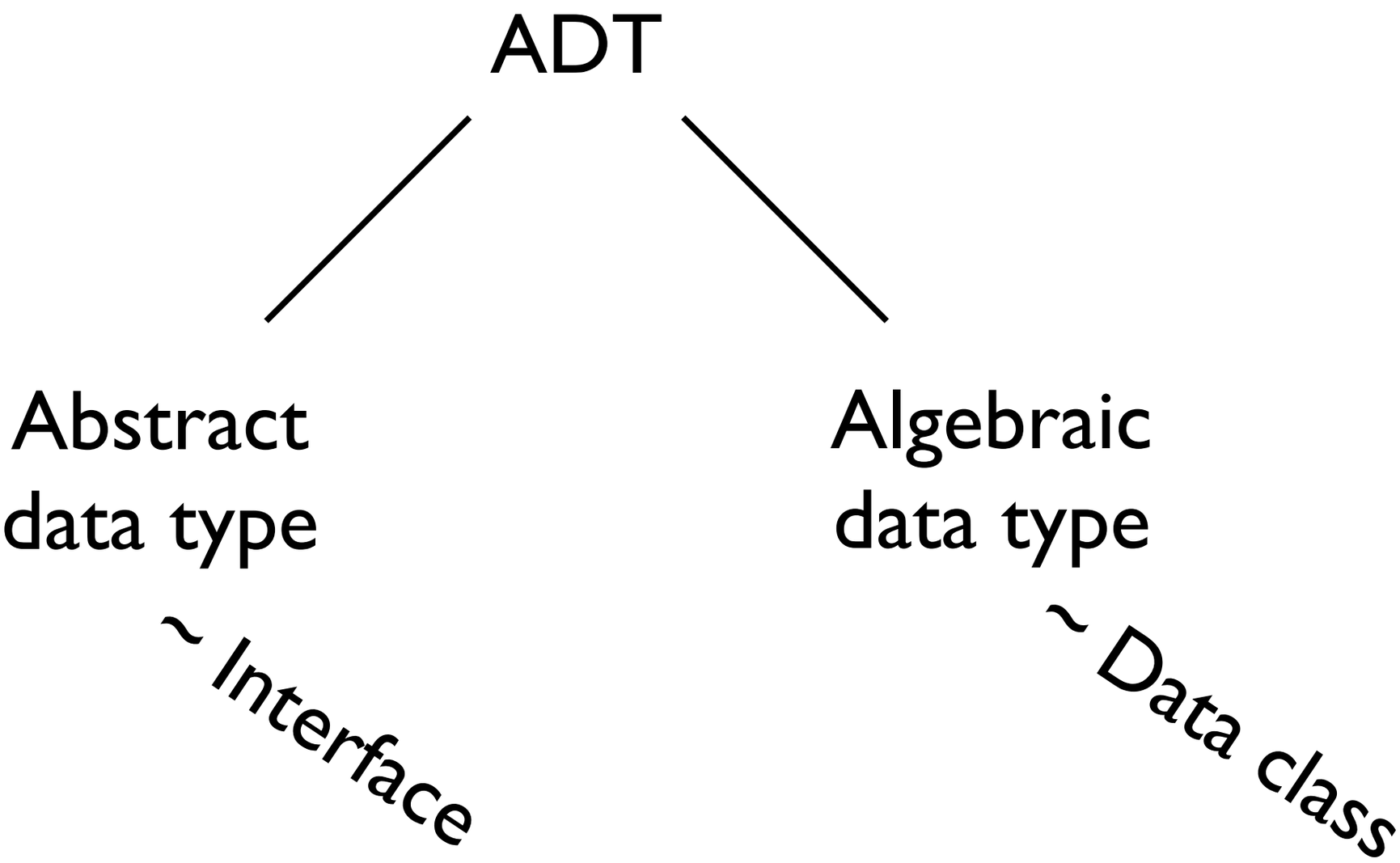
Example: Messages



Example: Messages




```
message ::= search for term
          | exact item
          | similar item
```


ADT
Abstract data type
~ Interface
Algebraic data type
~ Data class

Are you functional?

	Recursion	Pure	HO	ADT
Kotlin	✓	✗	✓	✓
Groovy	✓	✗	✓	✓
Scala	✓	✗	✓	✓
Java 8	✓	✗	✓	✓

ADT in Kotlin

```
abstract class Message

class SearchFor(val term: String) : Message()
class ExactMatch(val at: Location) : Message()
class Similarity(val at: Location) : Message()
```

```
server.send(SearchFor("word")) { m ->
    when (m) {
        is ExactMatch -> println("Found at ${m.at})")
        is Similarity -> println("Similar at ${m.at})")
        else -> println("Unknown message: $m")
    }
}
```

JetBRAINS

Utilization

You may have your cake, but can you eat it too?

JetBRAINS

Up to **5** HO-functions **for free**!

Kotlin

Static Utility Methods

```
map(filter(collection, { x -> x > 5 }), { x -> x * x })
```

backwards!

Static Utility Methods

```
map(filter(collection, { x -> x > 5 }), { x -> x * x })
```

backwards!

as opposed to

```
collection
        .filter { x -> x > 5 }
        .map { x -> x * x }
```

Extension Functions

```
fun Collection<Int>.filter(f: (Int) -> Boolean): List<Int> {
    val r = ArrayList<Int>()

    for (x in this) {
        if (f(x)) {
            r.add(x)
        }
    }

    return r
}
```

Extension Functions

Receiver Type

```
fun Collection<Int>.filter(f: (Int) -> Boolean): List<Int> {
    val r = ArrayList<Int>()

    for (x in this) {
        if (f(x)) {
            r.add(x)
        }
    }

    return r
}
```

Extension Functions

Receiver Type

```
fun Collection<Int>.filter(f: (Int) -> Boolean): List<Int> {
    val r = ArrayList<Int>()

    for (x in this) {
        if (f(x)) {
            r.add(x)
        }
    }

    return r
}
```

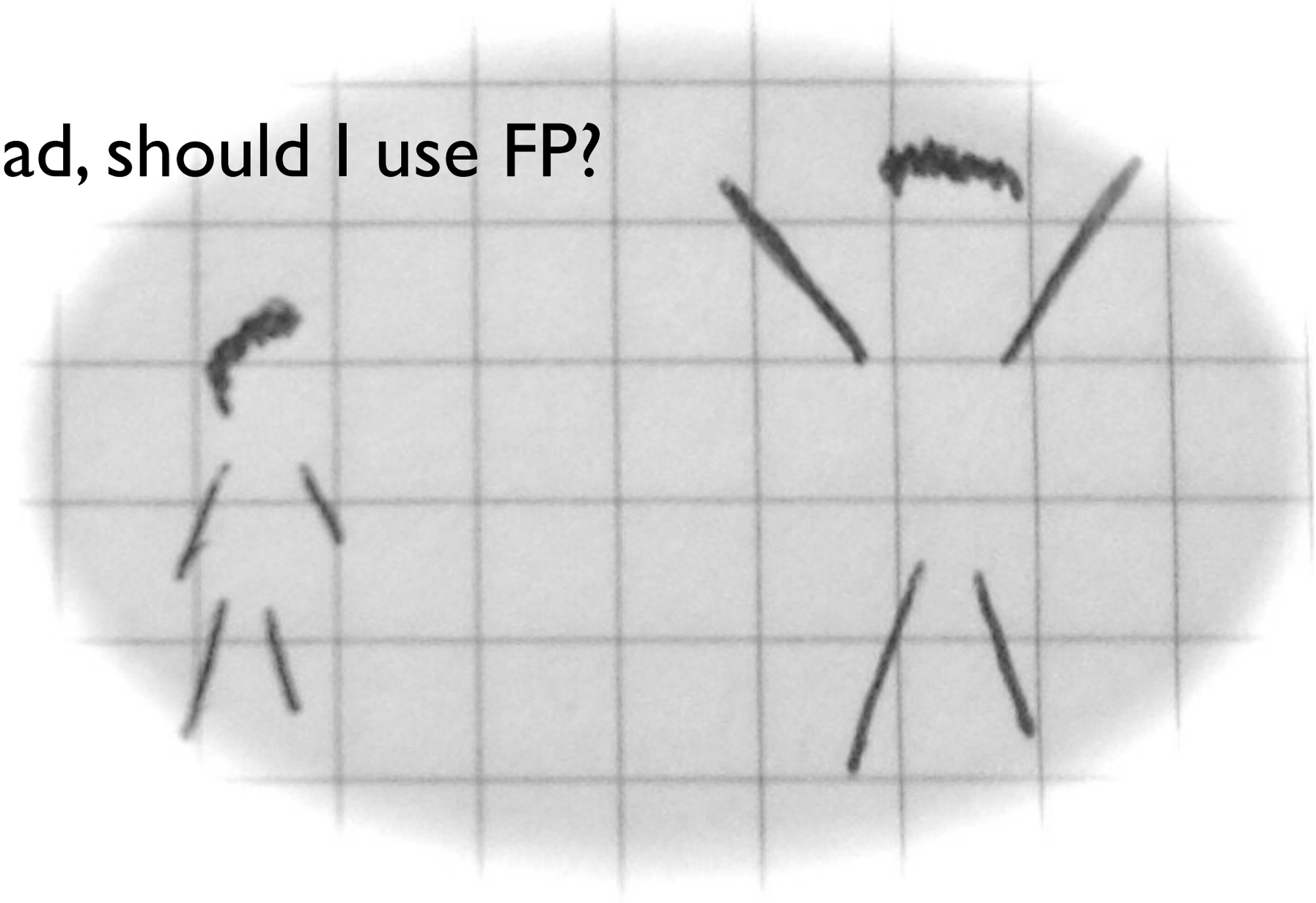
receiver

Summary

- Extending existing types
- Without changing the classes

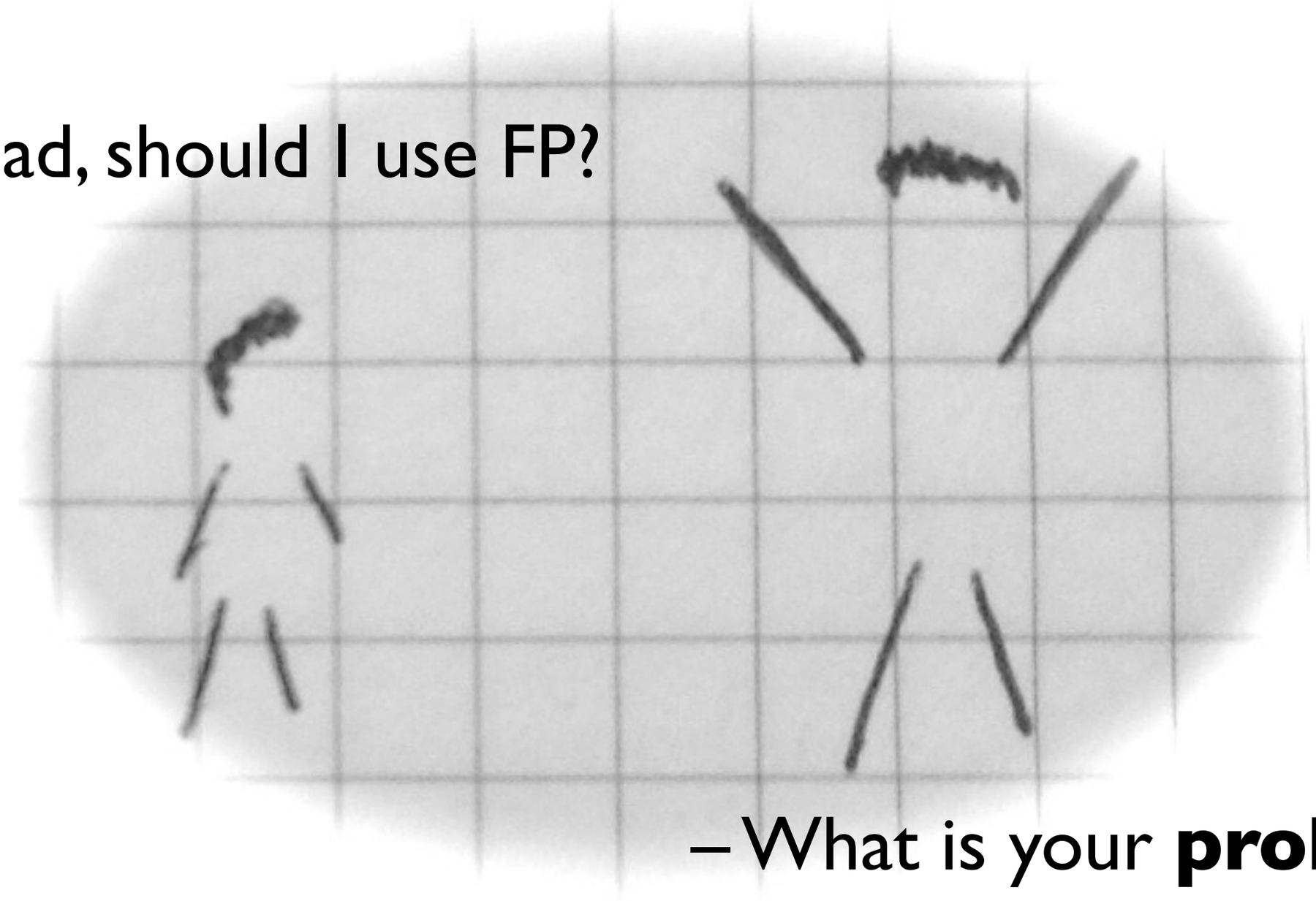
Epilogue

– Dad, should I use FP?



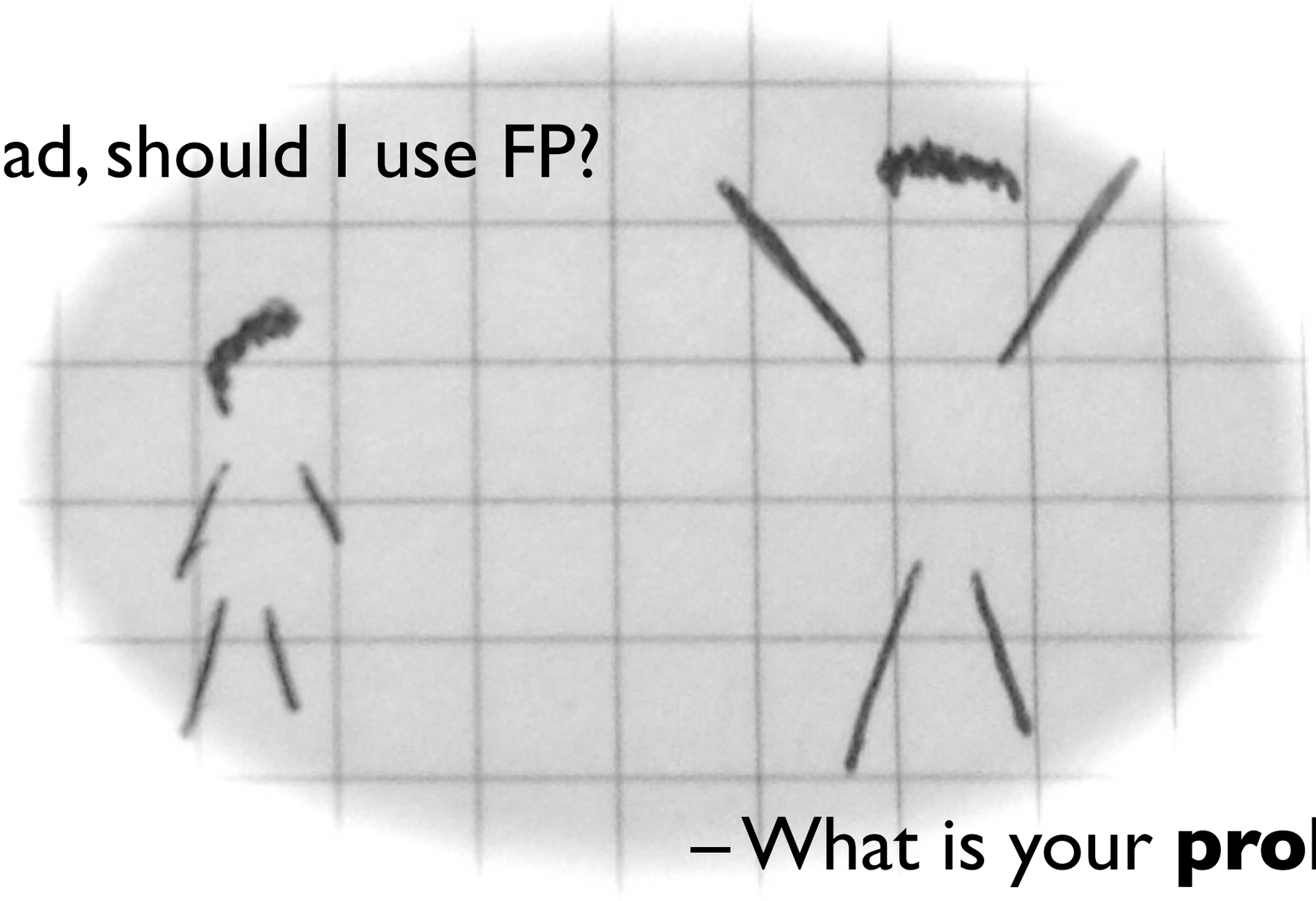
Kotlin

– Dad, should I use FP?



– What is your **problem**?!

– Dad, should I use FP?



– What is your **problem**?!

P.S. Kotlin is cool :) http://kotlin.jetbrains.org

Kotlin Resources

- Docs: http://kotlin.jetbrains.org
- Demo: http://kotlin-demo.jetbrains.com
- Code: http://github.com/jetbrains/kotlin
- Twitter:
  - @project_kotlin
  - @abreslav