



Что это за слово, которым пугают ООП-разработчиков?



Хакимов Айнур

- Scala разработчик в Tinkoff
- До Tinkoff писал на C/C++

План курса ФИЛП 2024

1. Вводная

2. Базовый синтаксис

3. Функции

4. ADT, Collections

5. *Implicits,
Typeclasses*

6. Common Typeclasses

7. Monad

8. Асинхронное
выполнение

9. Future

10. Монада IO

План лекции

План лекции

- Поймем, что такое монада и для чего она нужна

План лекции

- Поймем, что такое монада и для чего она нужна
- Увидим, какие нам знакомые структуры являются монадами

План лекции

- Поймем, что такое монада и для чего она нужна
- Увидим, какие нам знакомые структуры являются монадами
- Рассмотрим монады Reader, Writer, State

Monad

Что это такое?

Monad



A monad is a
monoid in the category
of endofunctors

Monad

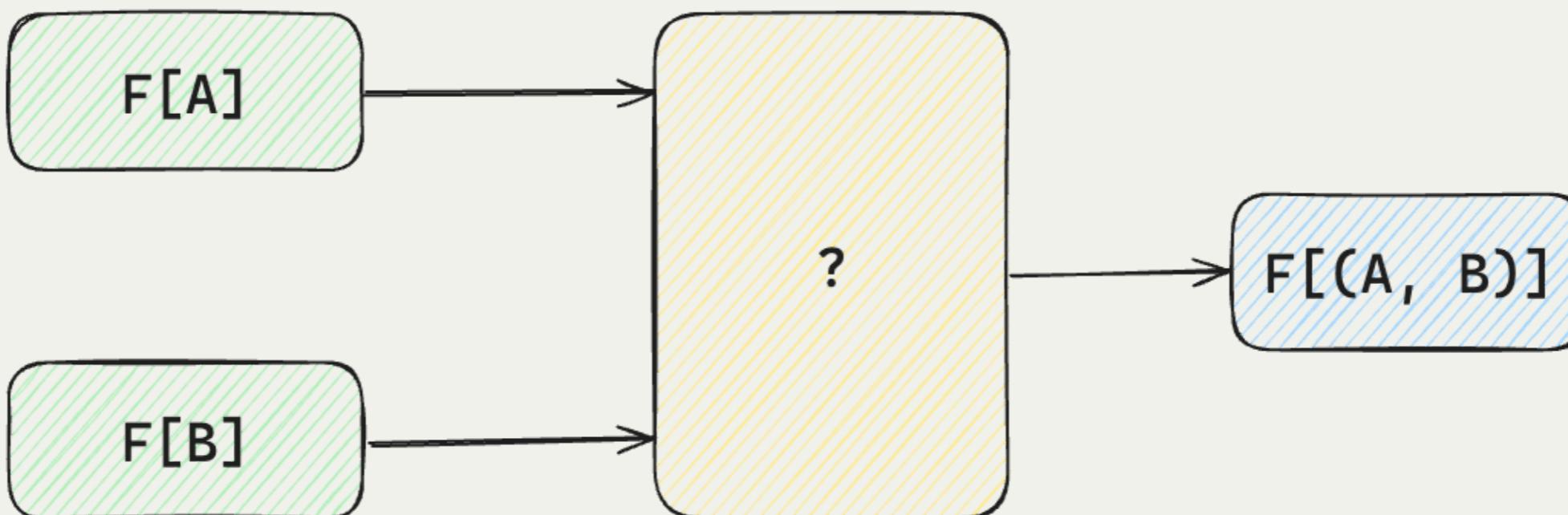


A monad is a
mechanism for
sequencing computations

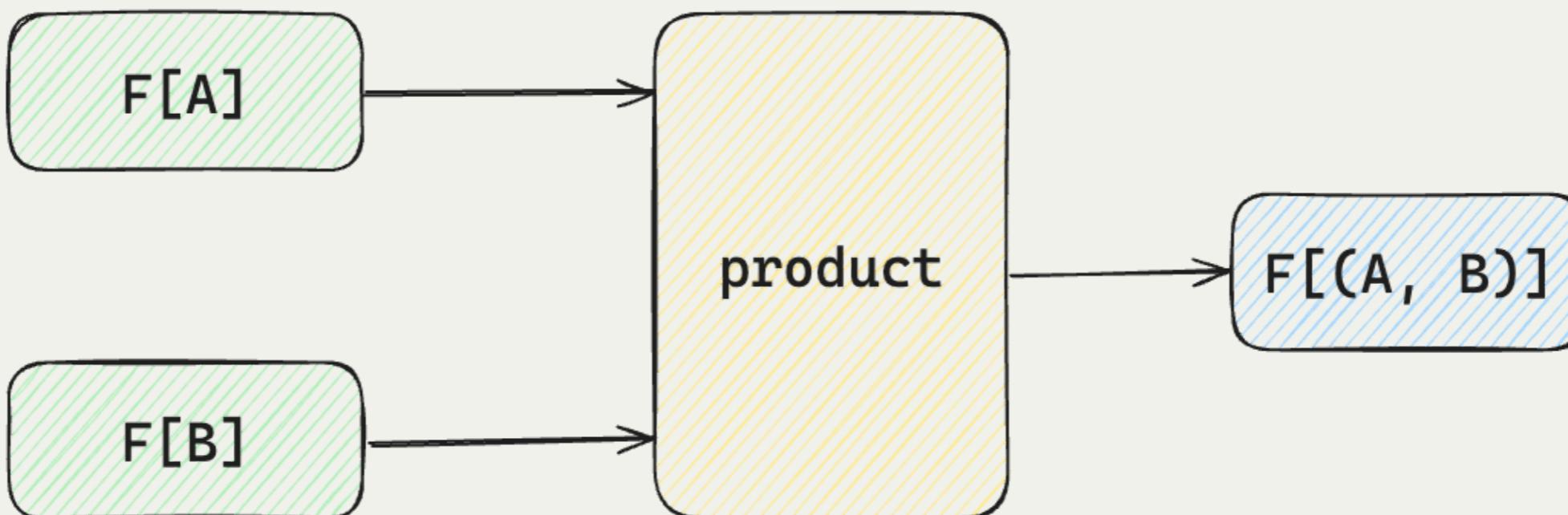
Вычисления

- Независимые
- Последовательные

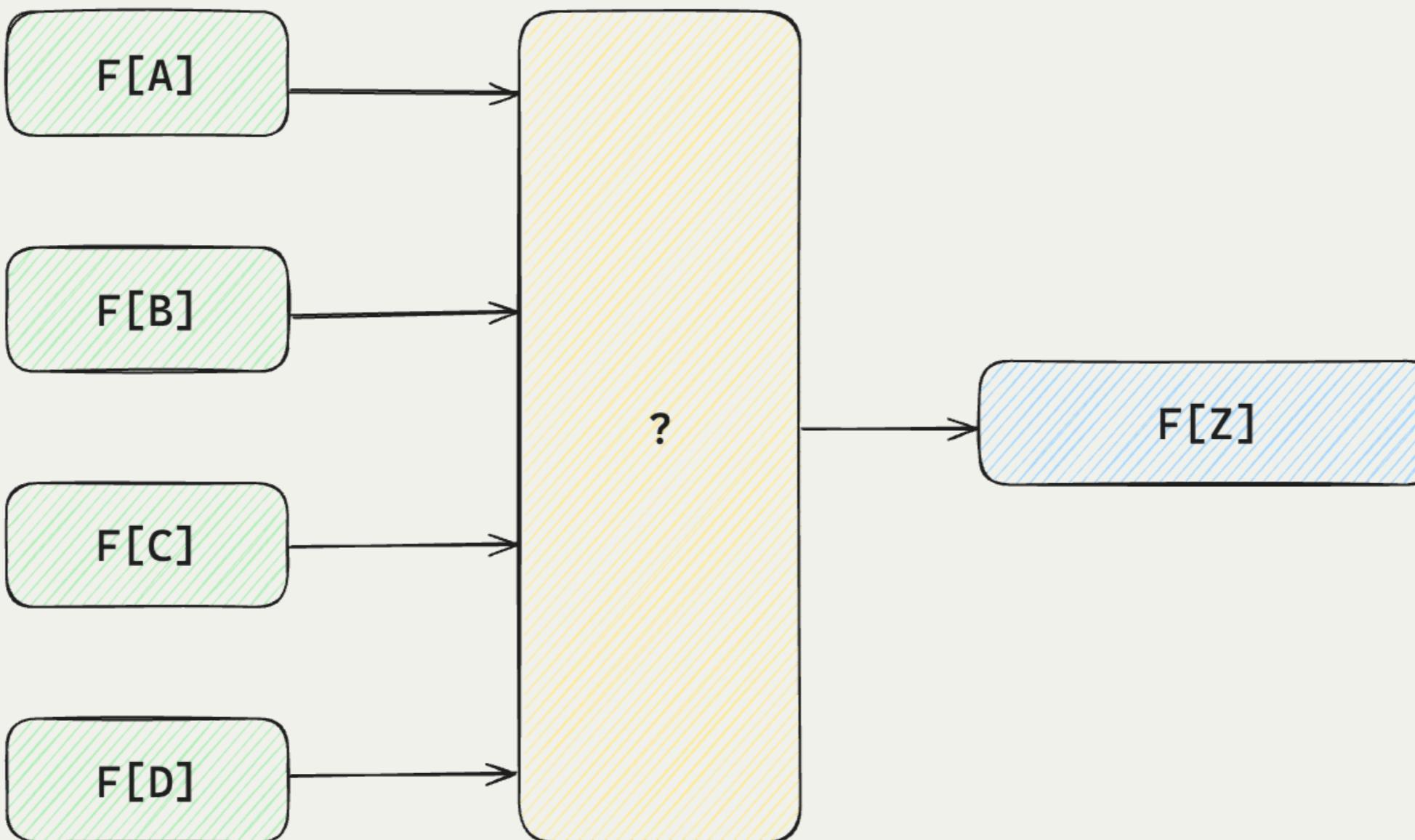
Независимые вычисления



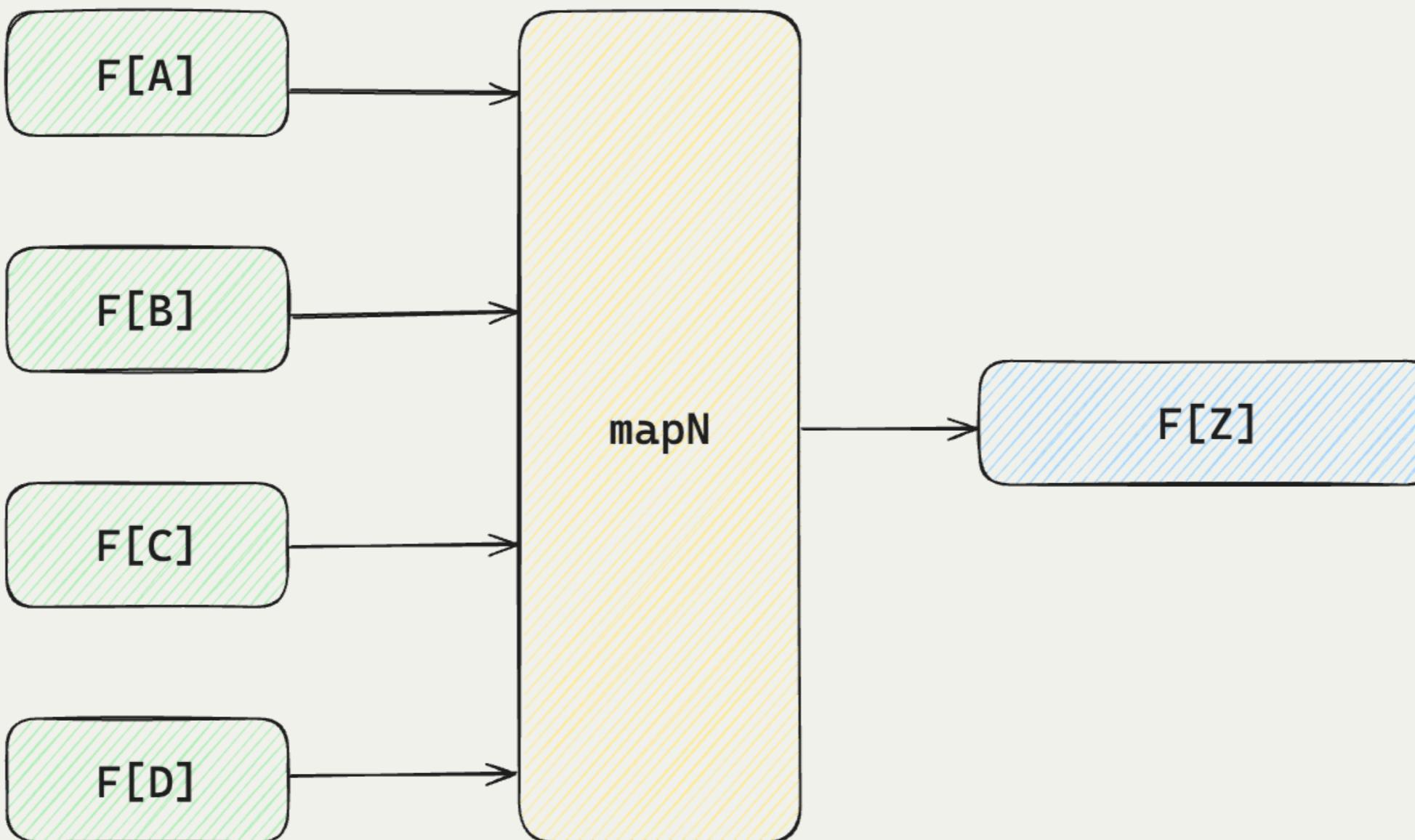
Независимые вычисления



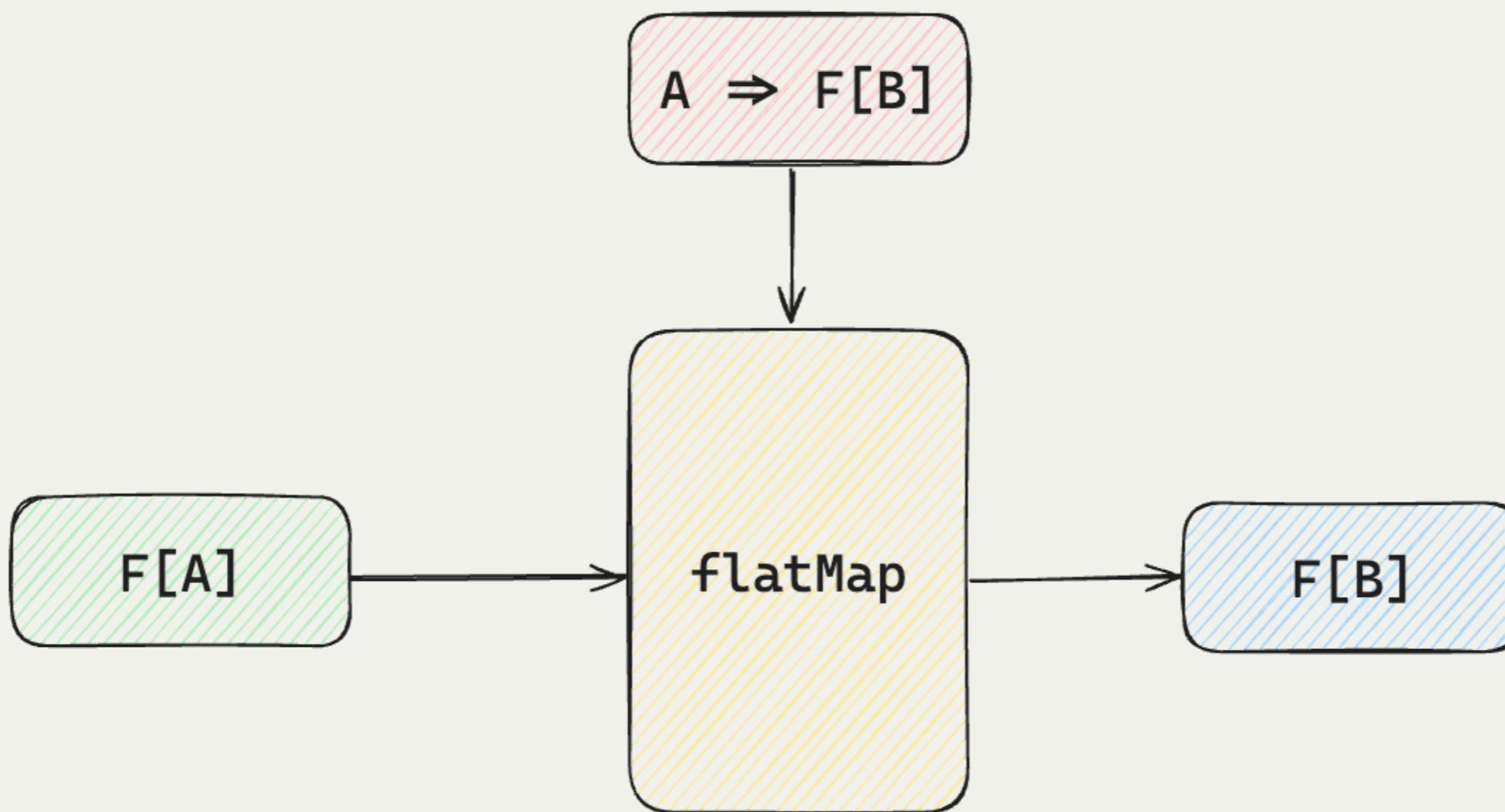
Независимые вычисления



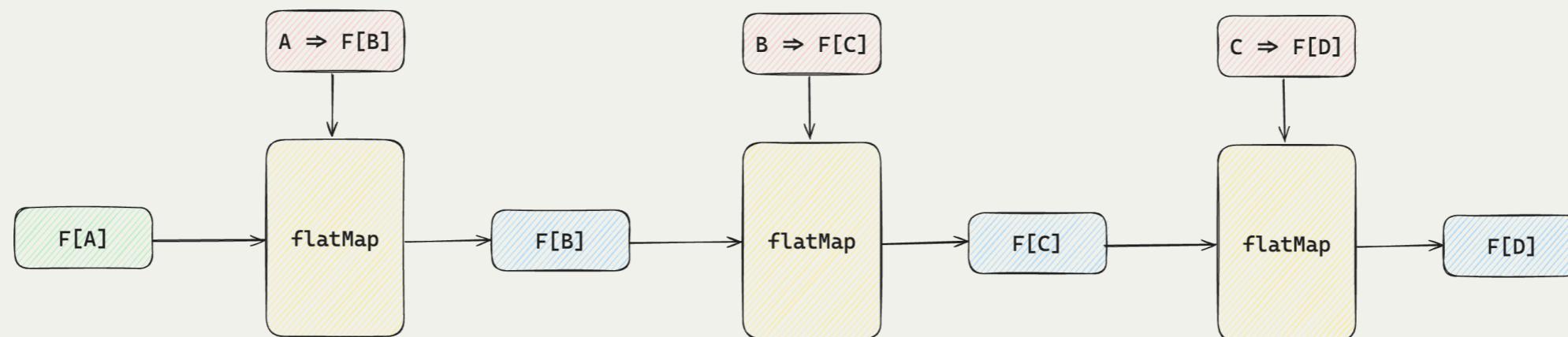
Независимые вычисления



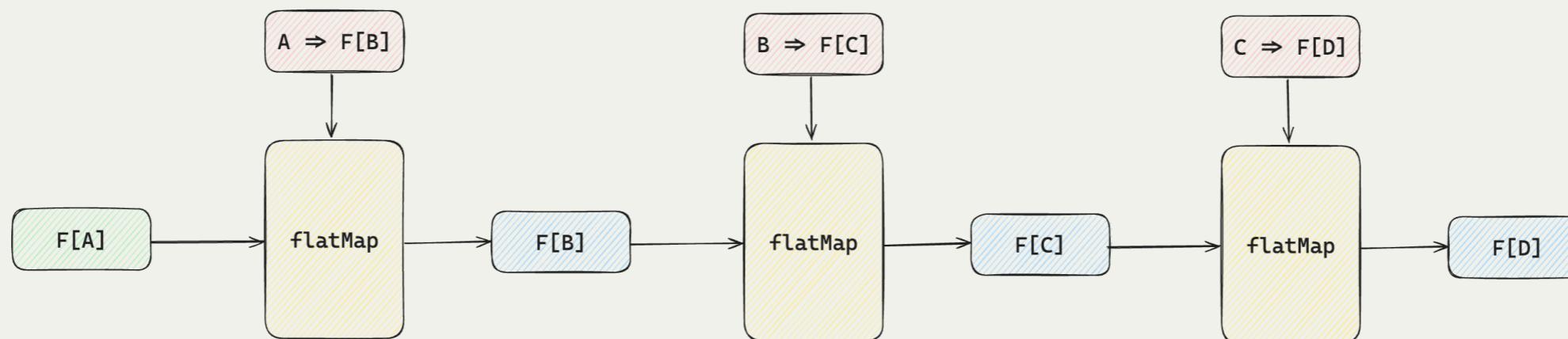
Последовательные вычисления



Последовательные вычисления



Последовательные вычисления



	Method	From	Given	To
Functor	map	$F[A]$	$A \Rightarrow B$	$F[B]$
Monad	flatMap	$F[A]$	$A \Rightarrow F[B]$	$F[B]$

Option Monad

```
Option("42").flatMap(value => value.toIntOption) // Option[Int]
```

Option Monad

```
Option("42").flatMap(value => value.toIntOption) // Option[Int]
```

```
Option("42")
  .map(value => value.toIntOption) // Option[Option[Int]]
  .flatten                      // Option[Int]
```

Option Monad

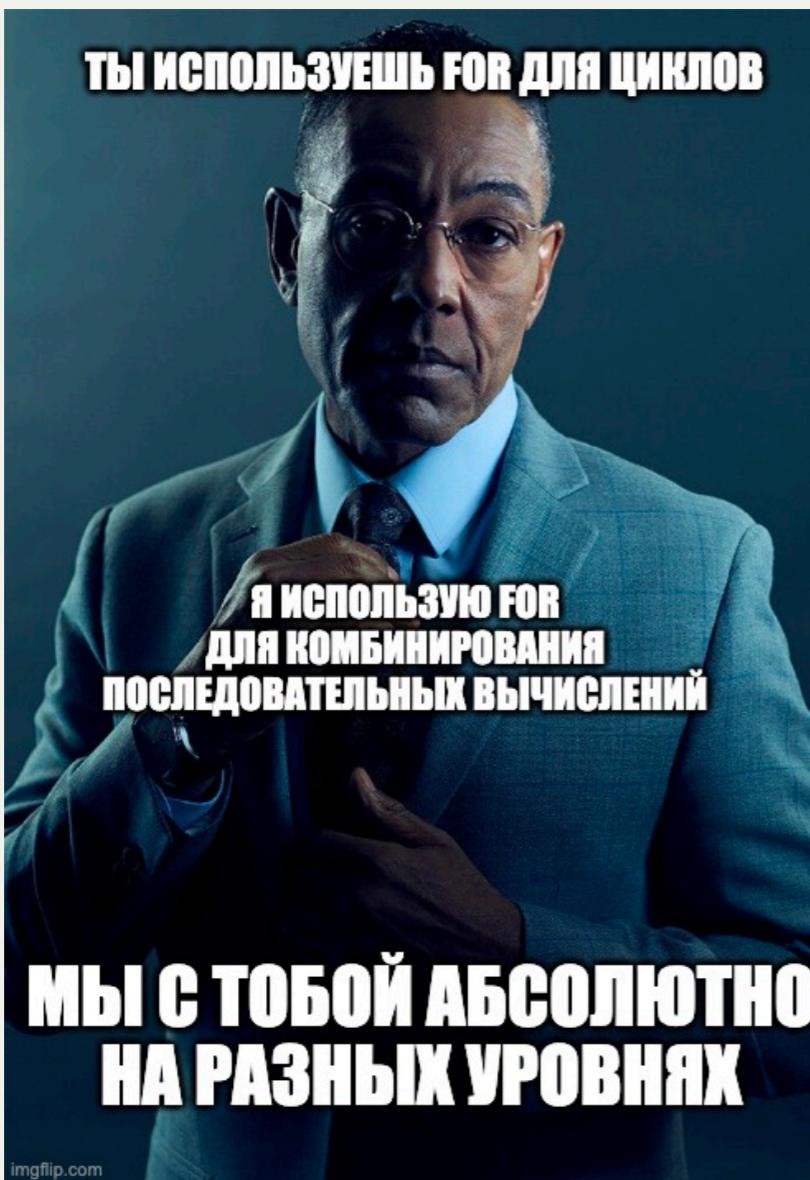
```
1 def loadUserOption(tag: String): Option[User] =  
2   idByTag(tag).flatMap {  
3     id => nameById(id).flatMap {  
4       name => surnameById(id).map {  
5         surname => User(id, name, surname)  
6       }  
7     }  
8   }
```

For Comprehension

```
1 def loadUserOption(tag: String): Option[User] =  
2   idByTag(tag).flatMap {  
3     id => nameById(id).flatMap {  
4       name => surnameById(id).map {  
5         surname => User(id, name, surname)  
6       }  
7     }  
8   }
```

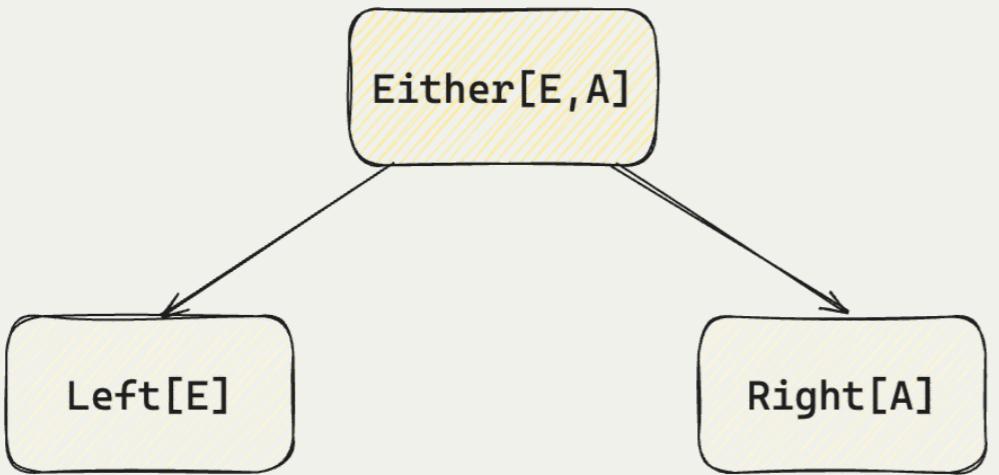
```
=  
1 def loadUserOption(tag: String): Option[User] =  
2   for {  
3     id      <- idByTag(tag)  
4     name    <- nameById(id)  
5     surname <- surnameById(id)  
6   } yield User(id, name, surname)
```

For Comprehension



```
1 def loadUserOption(tag: String): Option[User] =  
2   for {  
3     id      <- idByTag(tag)  
4     name    <- nameById(id)  
5     surname <- surnameById(id)  
6   } yield User(id, name, surname)
```

Either Monad



Either Monad

```
1 Right("42").flatMap(  
2   value => value.toIntOption.toRight("Parse failed")  
3 )
```



Either Monad

```
1 def loadUserEither(tag: String): Either[Error, User] =  
2   for {  
3     id      <- idByTag(tag)  
4     name    <- nameById(id)  
5     surname <- surnameById(id)  
6   } yield User(id, name, surname)
```



Monad

```
1 trait Monad[F[_]] extends Applicative[F] {  
2     def flatMap[A, B](fa: F[A])(f: A => F[B]): F[B]  
3 }
```

Monad

```
import Monad.Syntax._

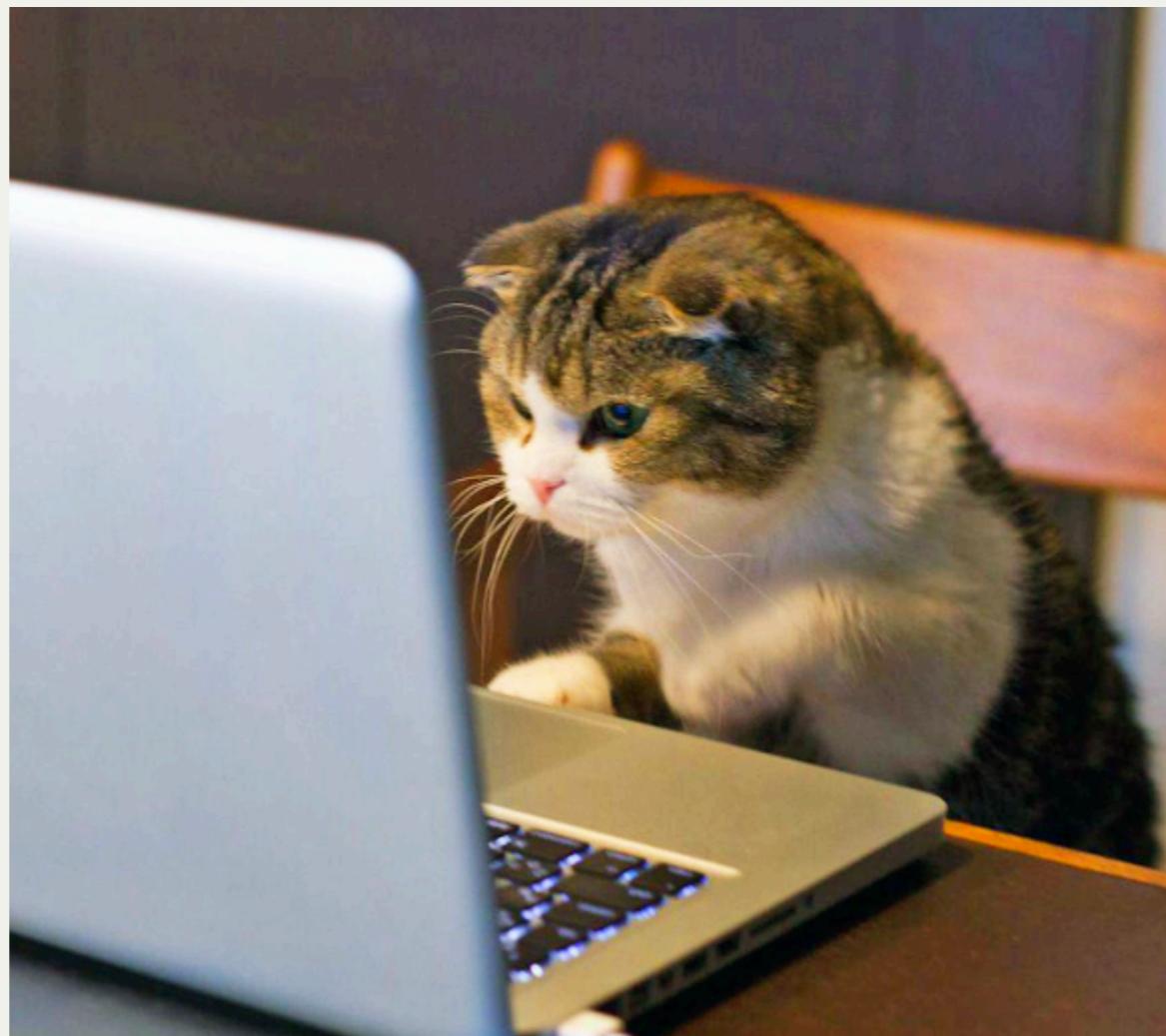
def loadUser[F[_]: Monad](tag: String): F[User] = {
  for {
    id      <- idByTag(tag)
    name   <- nameById(id)
    surname <- surnameById(id)
  } yield User(id, name, surname)
}
```

Monad

```
import Monad.Syntax._

def loadUser[F[_]: Monad](tag: String): F[User] = {
  for {
    id      <- idByTag(tag)
    name   <- nameById(id)
    surname <- surnameById(id)
  } yield User(id, name, surname)
}
```

```
loadUser[Option]("123")
loadUser[Either[String, *]]("444")
loadUser[Try]("456")
loadUser[CustomType]("456")
```



Посмотрим на примеры кода с монадами

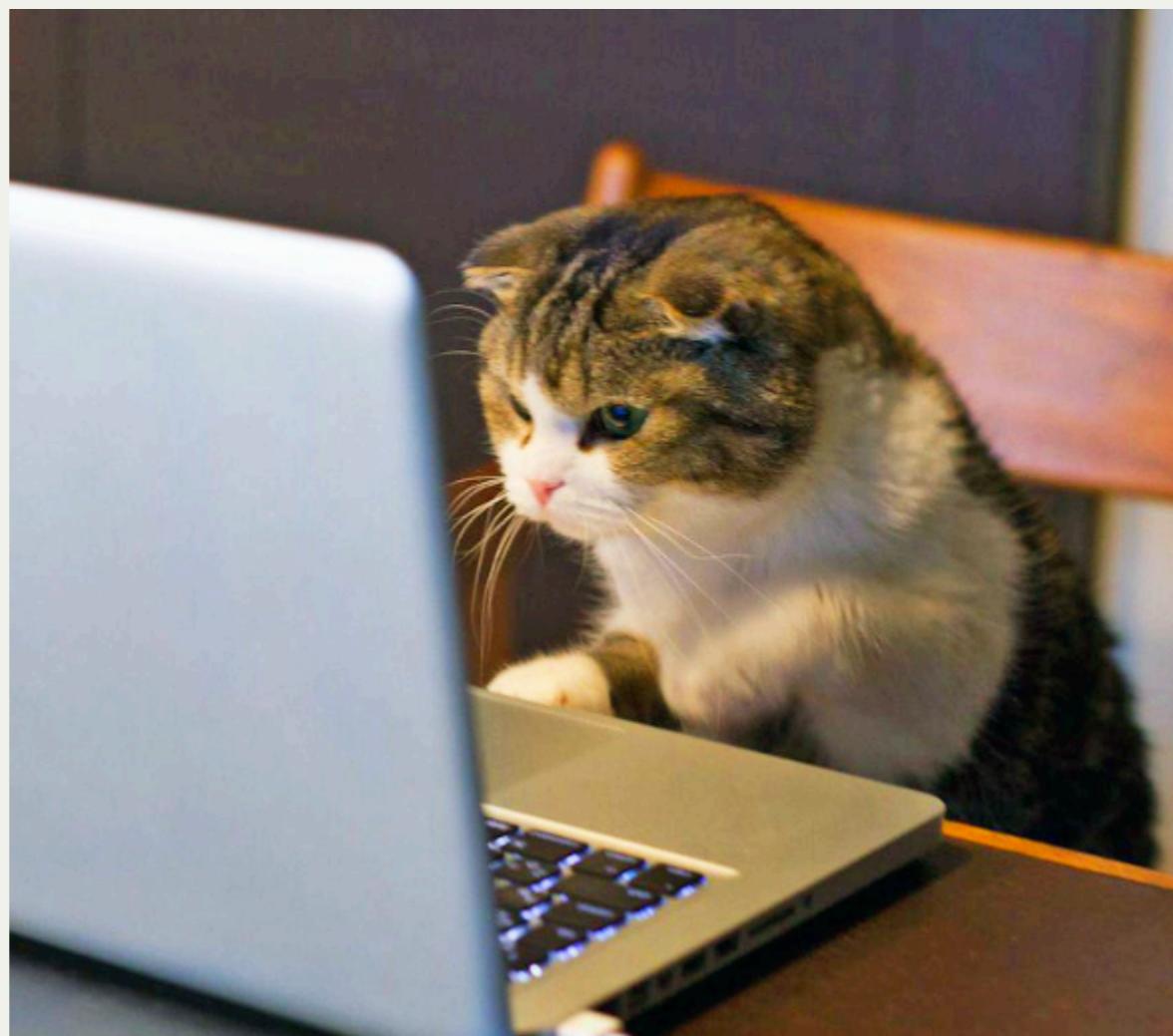
Вопросы?

Monad Laws

```
// Left identity
Monad[F].pure(a).flatMap(f) == f(a)

// Right identity
fa.flatMap(Monad[F].pure) == fa

// Associativity
fa.flatMap(f).flatMap(g) == fa.flatMap(a => f(a).flatMap(g))
```



Посмотрим на примеры кода

Вопросы?

MonadError

```
1 trait MonadError[F[_], E] extends Monad[F] {
2   def throwError[A](e: E): F[A]
3
4   def handleErrorWith[A](fa: F[A])(f: E => F[A]): F[A]
5 }
```



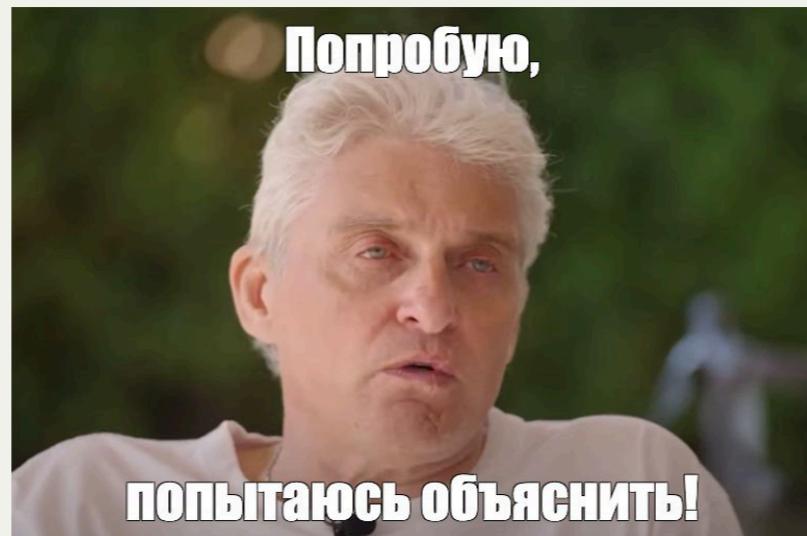
Переключимся в IDEA и посмотрим примеры кода

Вопросы?

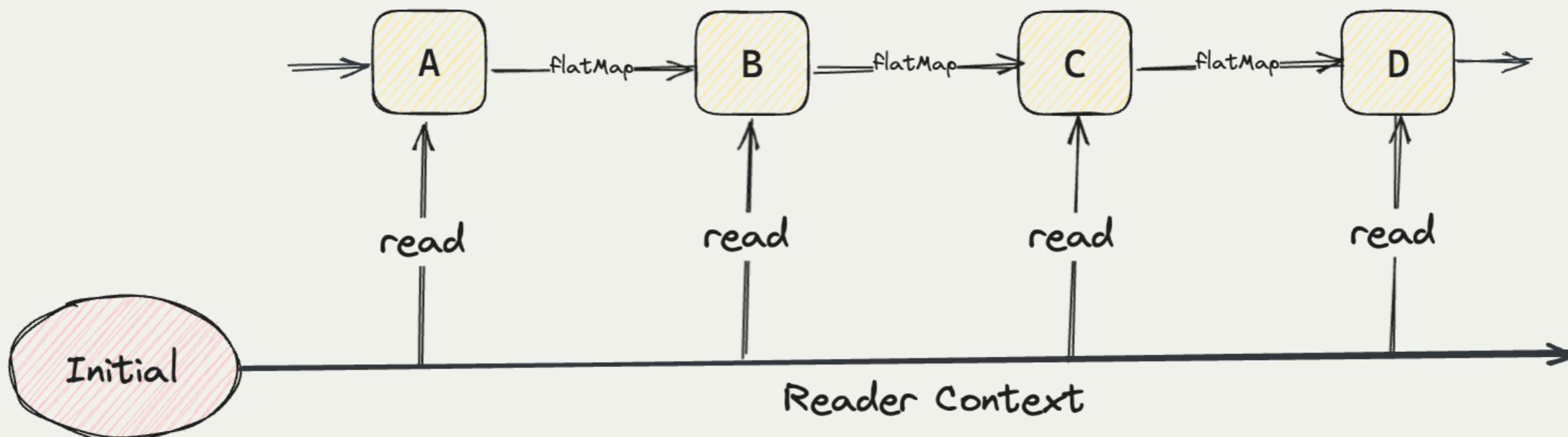


(начинается сложная часть лекции)

Reader Monad



Reader Monad





Переключимся в IDEA и посмотрим примеры кода

Reader Monad

```
1 // На самом деле это просто функция
2 final case class Reader[Ctx, Out](run: Ctx => Out)
3
4 object Reader {
5     // Позволяет извлечь информацию из контекста
6     def ask[Ctx, CtxInfo](f: Ctx => CtxInfo): Reader[Ctx, CtxInfo] = Reader(f)
7
8 // ...
```

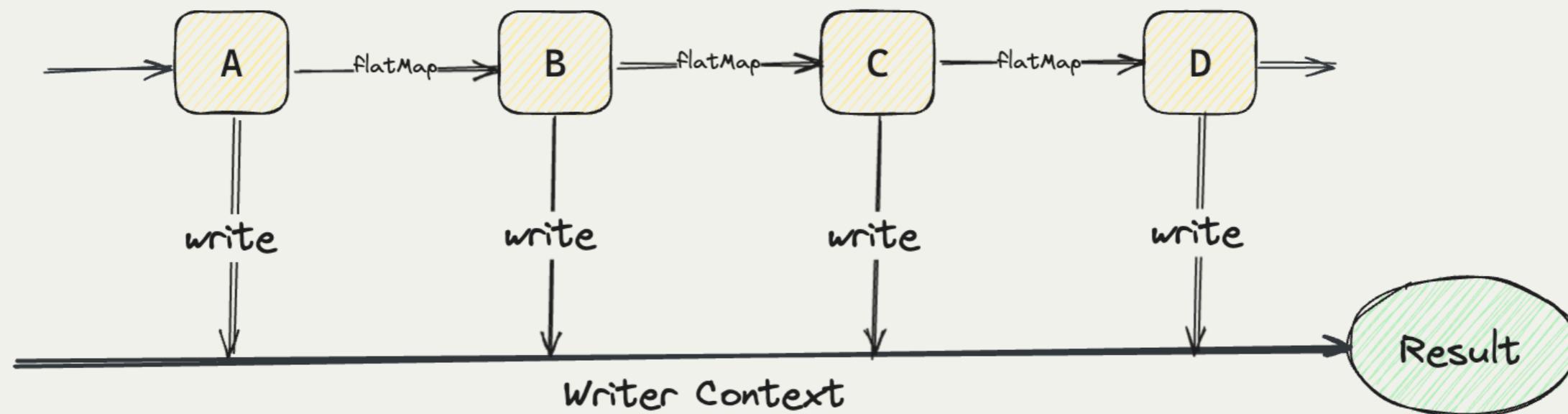
Reader Monad

```
1 def httpRequest(url: String): Reader[Config, String] = {
2   println(s"Http request $url")
3   "Bob".pure[Reader[Config, *]]
4 }
5
6 def getUserName(id: Int): Reader[Config, String] = for {
7   url  <- Reader.ask[Config, String](_.url)
8   name <- httpRequest(s"$url/$id")
9 } yield name
10
11 getUserName(42).run(Config("http://service.ru/users"))
```

Вопросы?

Writer Monad

Writer Monad



Writer Monad

```
1 case class Writer[Log, Value](run: (Log, Value)) {  
2  
3     // Позволяет делать записи в Log  
4     def tell(message: Log)(implicit monoid: Monoid[Log]): Writer[Log, Value] =  
5         run match {  
6             case (log, value) => Writer((log |+| message, value))  
7         }  
8     }  
9  
10    // ...
```

Writer Monad

```
1 type Logs = List[String]
2
3 def httpRequest(url: String): Writer[Logs, String] = for {
4     _ <- Writer.tell[Logs](List(s"Http request $url"))
5     response <- "Bob".pure[Writer[Logs, *]]
6 } yield response
7
8 def getUserName(id: Int): Writer[Logs, String] = for {
9     name <- httpRequest(s"http://some-service/$id")
10    _ <- Writer.tell[Logs](List(s"Fetched name $name"))
11 } yield name
12
13 val (logs, result) = getUserName(42).run
14 // Http request http://some-service/42
15 // Fetched name Bob
```

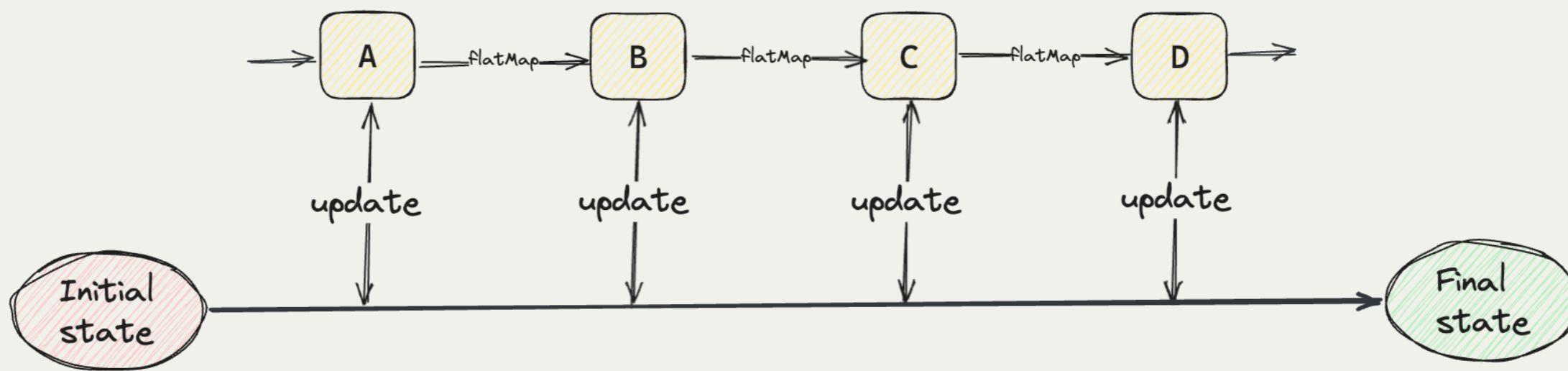


Переключимся в IDEA и посмотрим на примеры кода

Вопросы?

State Monad

State Monad



State Monad

```
1 // Тоже просто функция
2 final case class State[S, V](run: S => (V, S))
3
4 object State {
5     // Запрашивает состояние
6     def ask[S, StateCtx](f: S => StateCtx): State[S, StateCtx] =
7         State(current => (f(current), current))
8
9     // Обновляет состояние
10    def update[S](f: S => S): State[S, Unit] =
11        State(current => (((), f(current))))
12
13 // ...
```

State Monad

```
1 def updateStateExample(str: String): State[Int, String] = for {
2   current <- State.ask[Int, Int](identity)
3   _ <- str match {
4     case "increment" => State.update[Int](_ + 1)
5     case "decrement" => State.update[Int](_ - 1)
6     case _ => State.update[Int](identity) // do nothing
7   }
8   updated <- State.ask[Int, Int](identity)
9 } yield if (current != updated) "updated" else "unchanged"
10
11 val (result, state) = updateStateExample("increment").run(100)
12 // result=updated, state=101
```



Переключимся в IDEA и посмотрим на примеры кода

Вопросы?

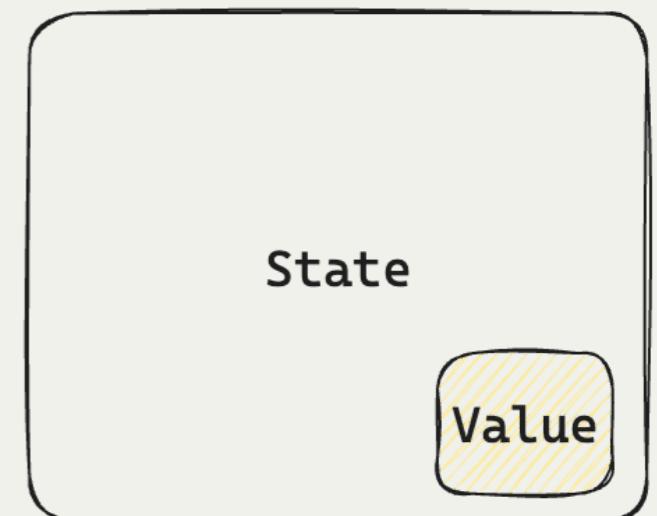
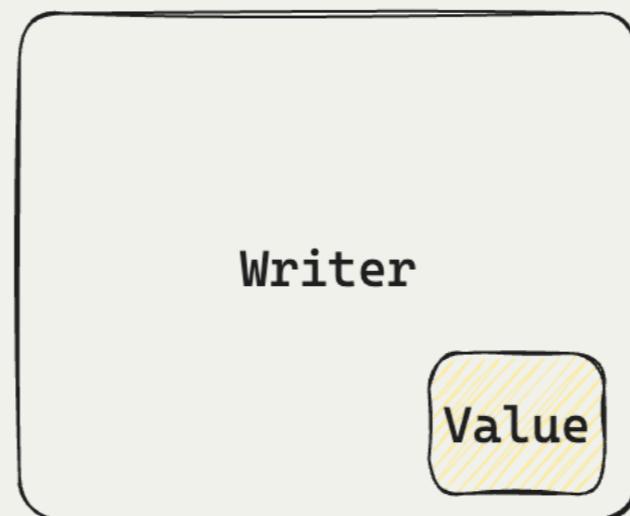
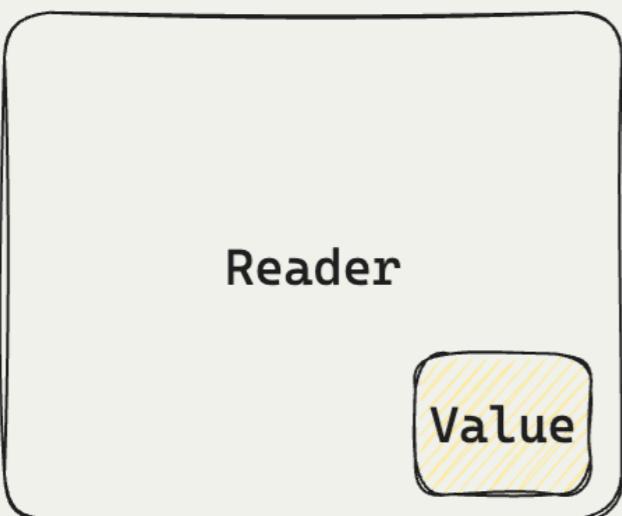
Monads

Reader

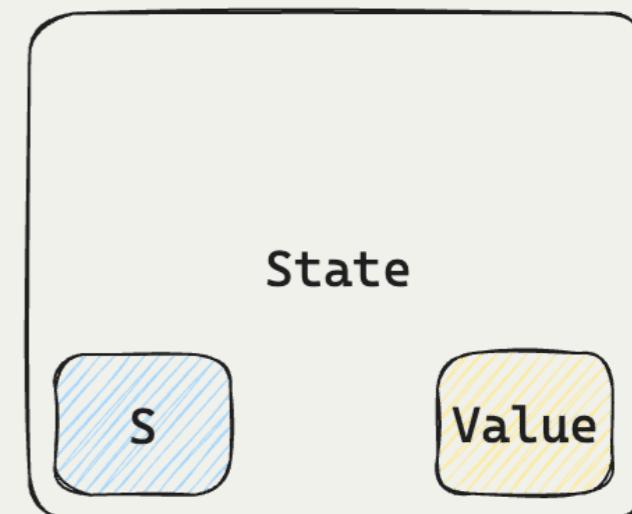
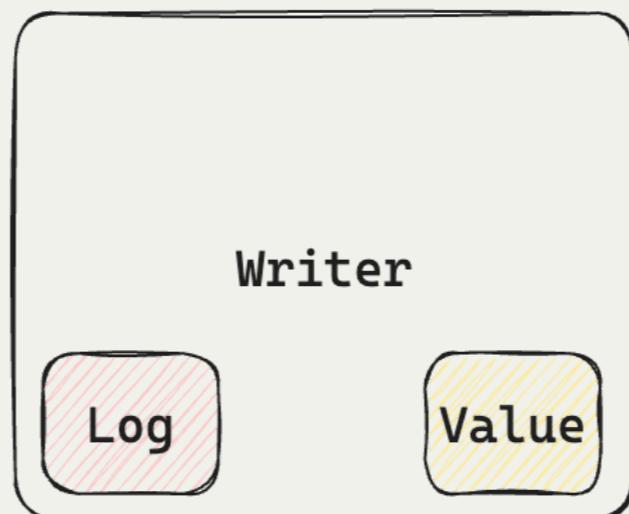
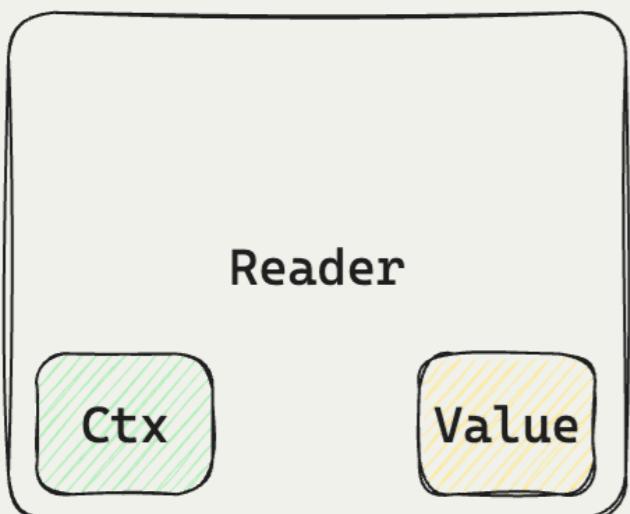
Writer

State

Monads

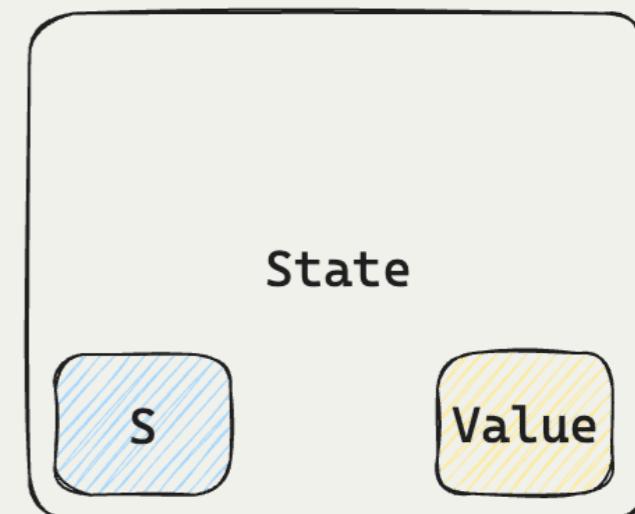
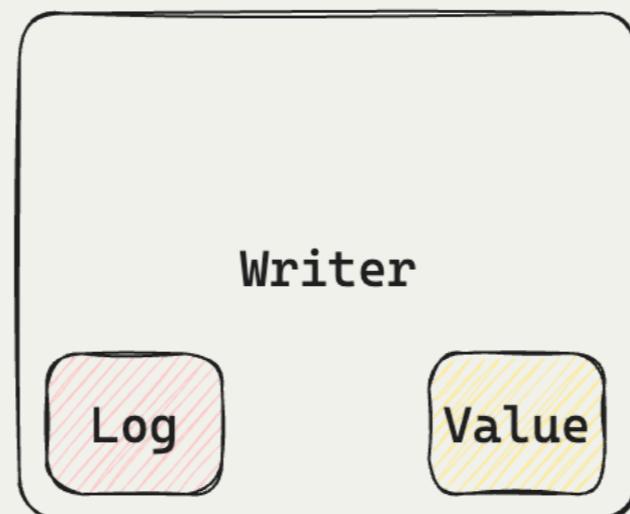
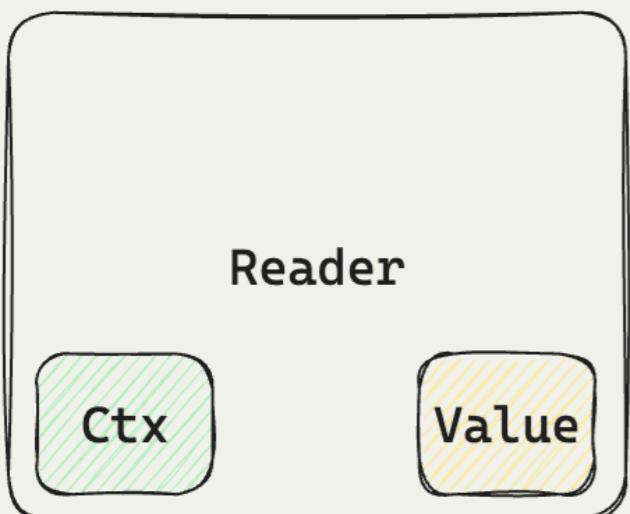


Monads



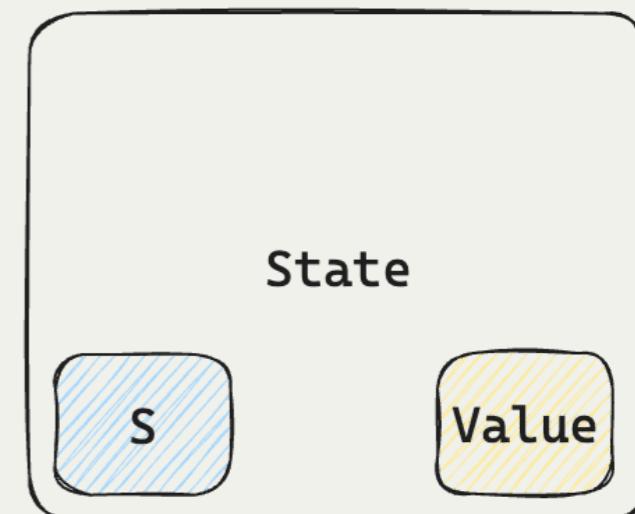
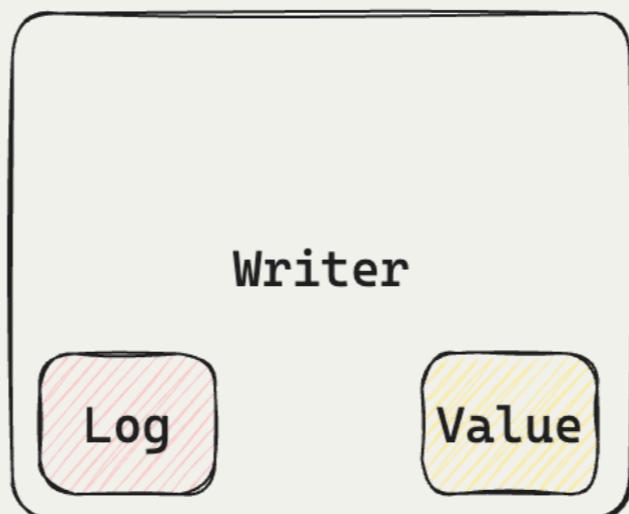
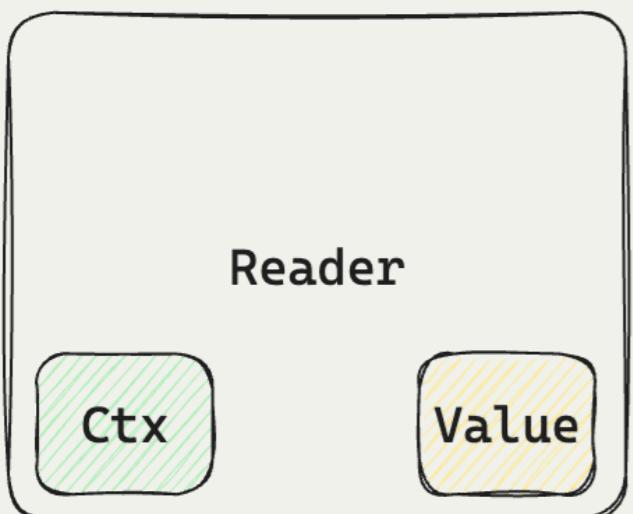
Monads

Позволяют композировать вычисления



Monads

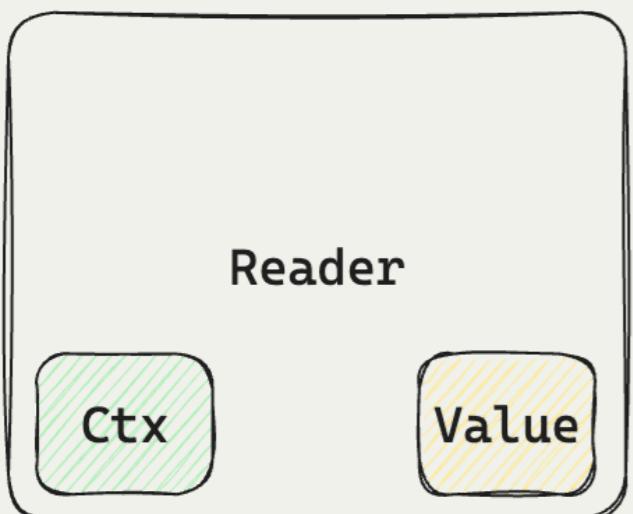
Позволяют композировать вычисления



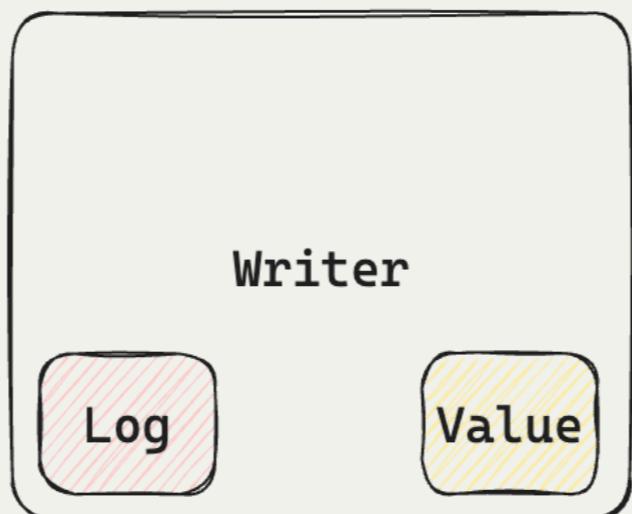
Позволяет передавать общий
контекст через последовательность
вычислений

Monads

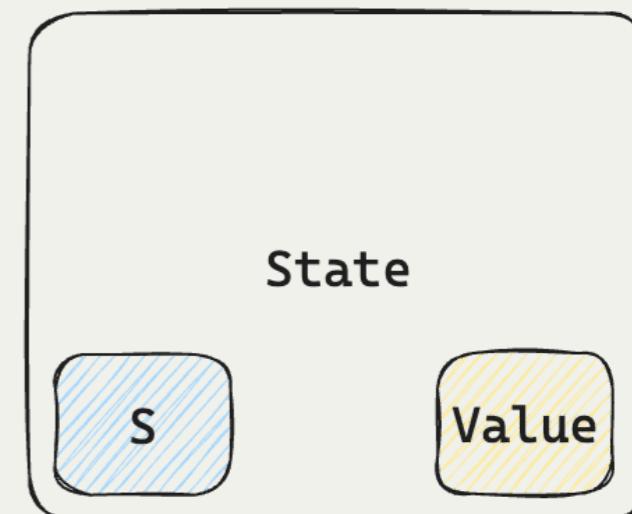
Позволяют компоновать вычисления



Позволяет передавать общий контекст через последовательность вычислений

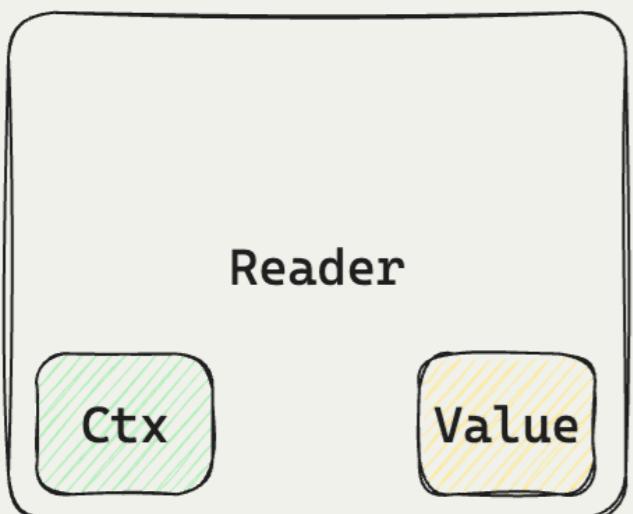


Позволяет собирать лог или другие побочные данные, которые генерируются в процессе вычислений

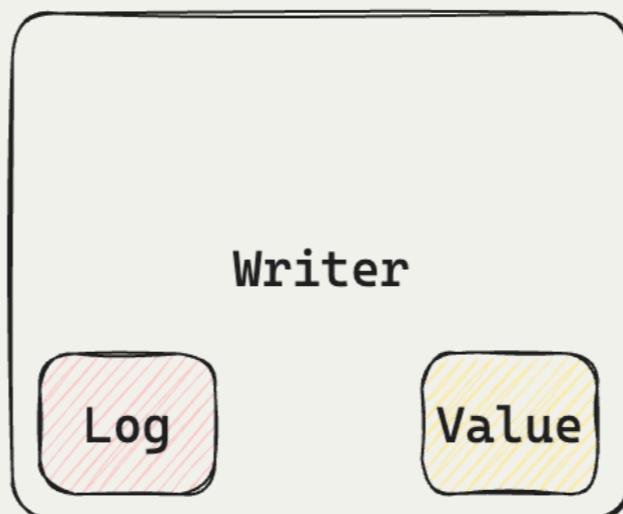


Monads

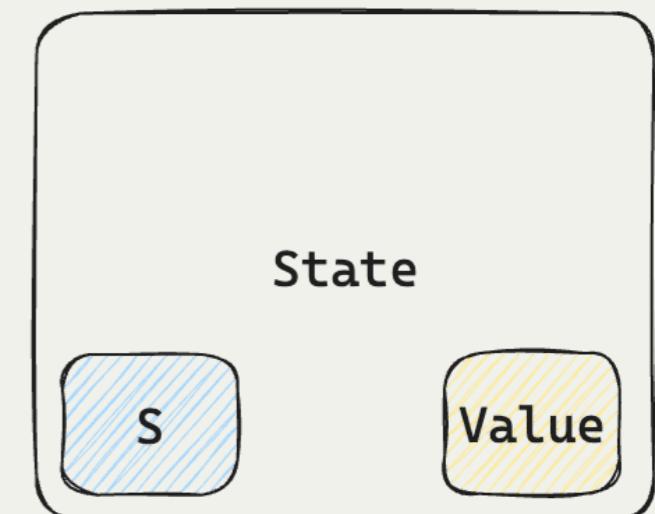
Позволяют компоновать вычисления



Позволяет передавать общий контекст через последовательность вычислений



Позволяет собирать лог или другие побочные данные, которые генерируются в процессе вычислений

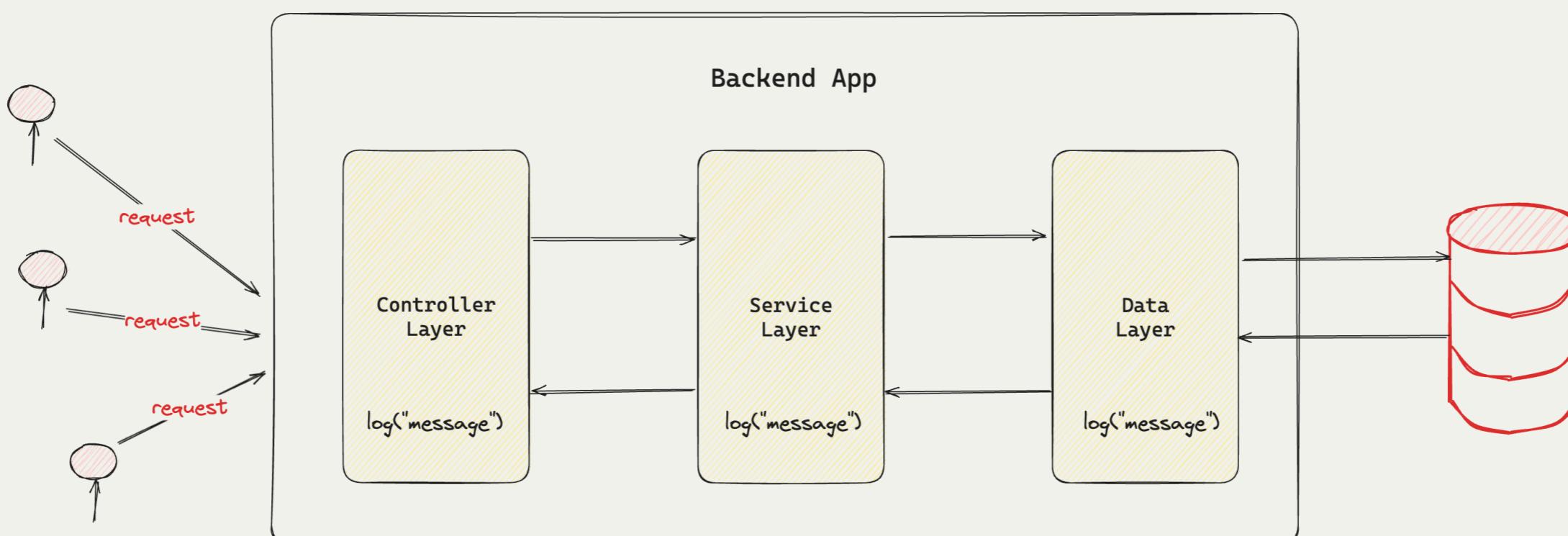


Обеспечивает способ управления изменяемым состоянием

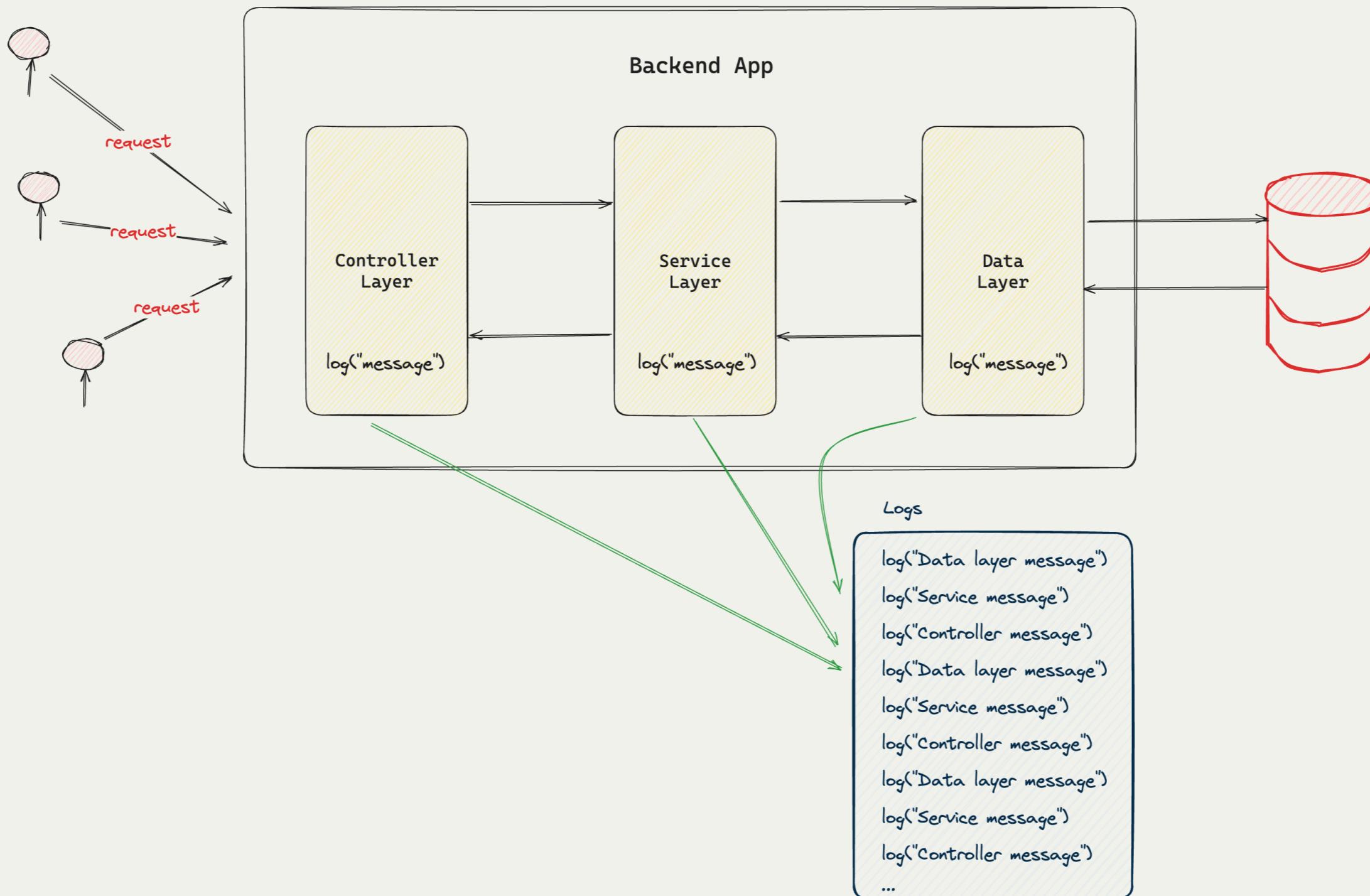
Вопросы?

Пример реального использования Reader монады

Пример реального использования Reader монады



Пример реального использования Reader монады

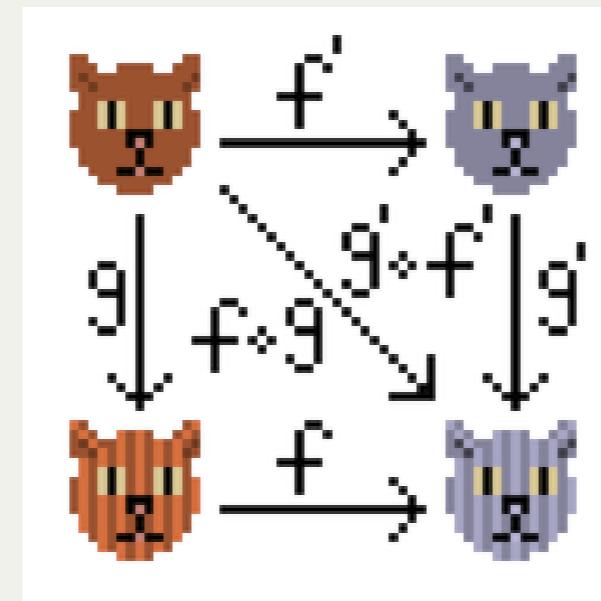




Переключимся в IDEA и посмотрим примеры кода

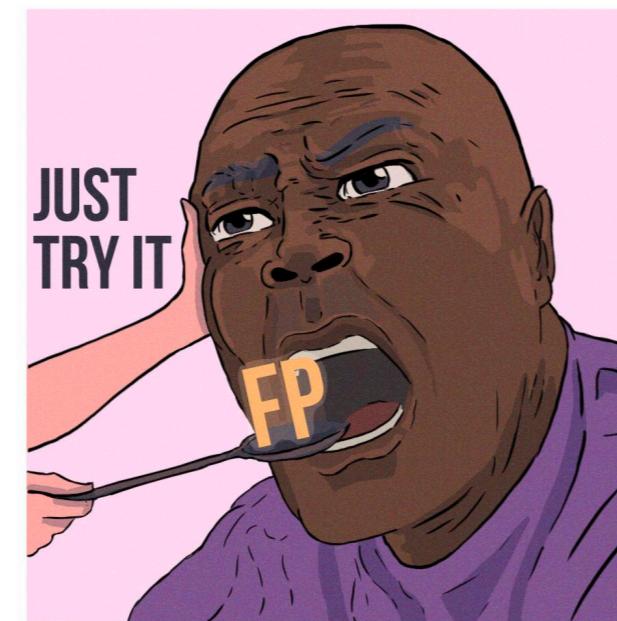
Вопросы?

Scala with Cats



Cats (categories) is a library which provides abstractions for functional programming in the Scala programming language

- <https://typelevel.org/cats/>
- <https://www.scalawithcats.com/>
- <https://eed3si9n.com/herding-cats/>



Спасибо за внимание!

