SMARTRHINO

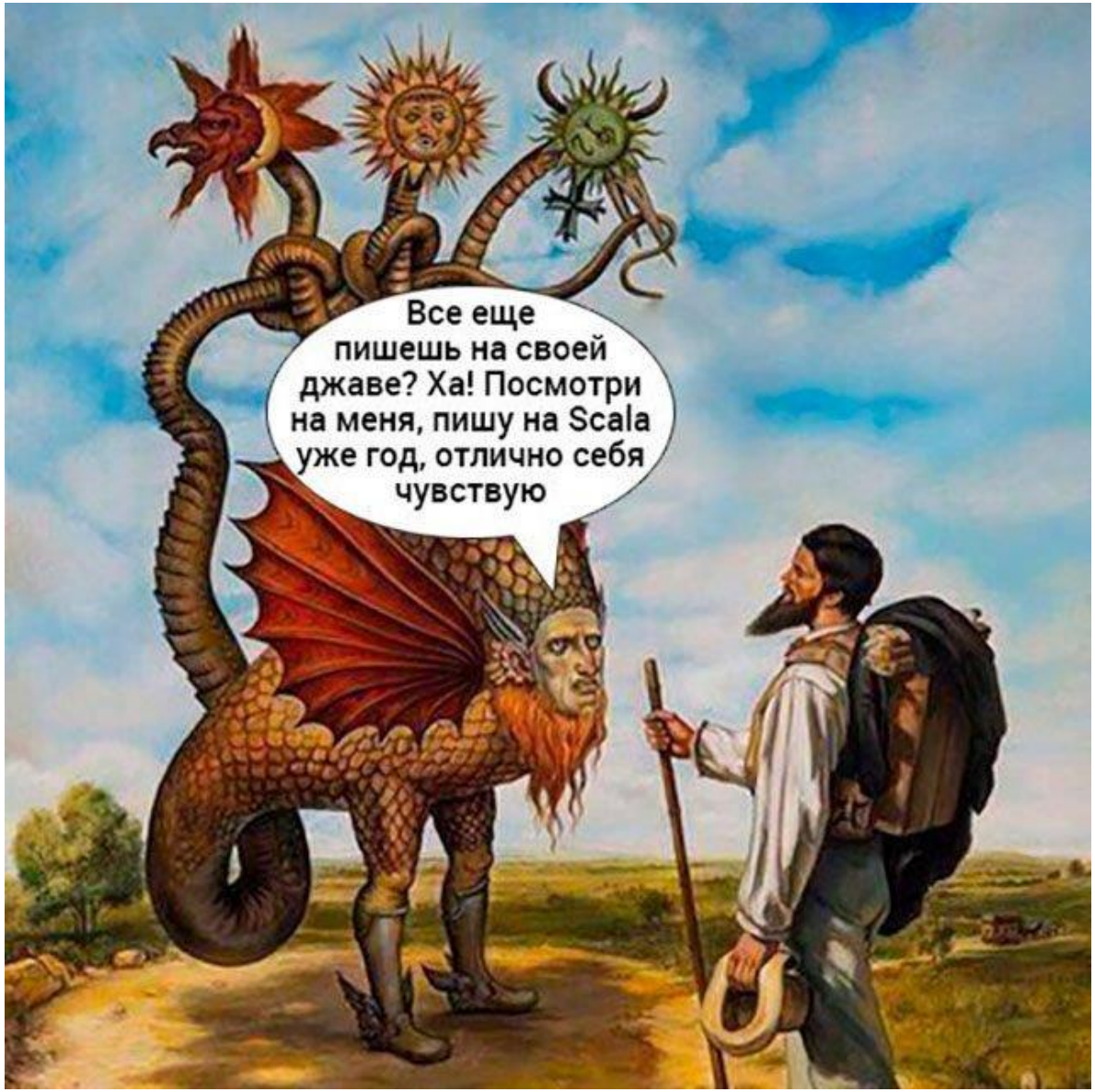
2019

BEST PRACTICES

по программированию

ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ: ПРАКТИЧНЫЙ ПОДХОД

Влад Чесноков


Все еще пишешь на своей джаве? Ха! Посмотри на меня, пишу на Scala уже год, отлично себя чувствую

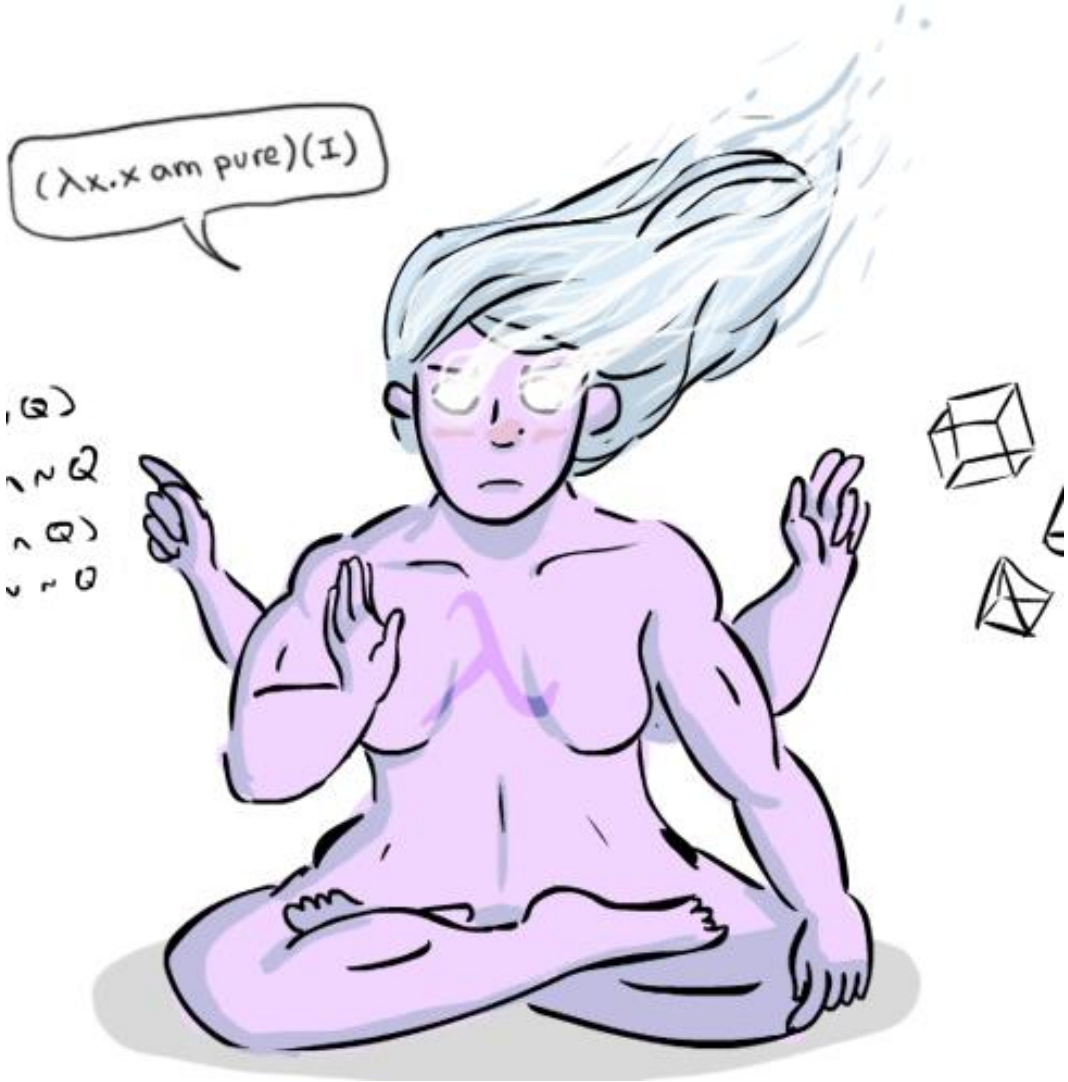
ФП в компании

- Java 8 – с 2015 года;
- Scala – с 2015 года (2.10);
- Kotlin – с 2016 года (1.0-beta);
- Python – примерно с 2012;
- немного C# - 2012-2015;

ФП в компании

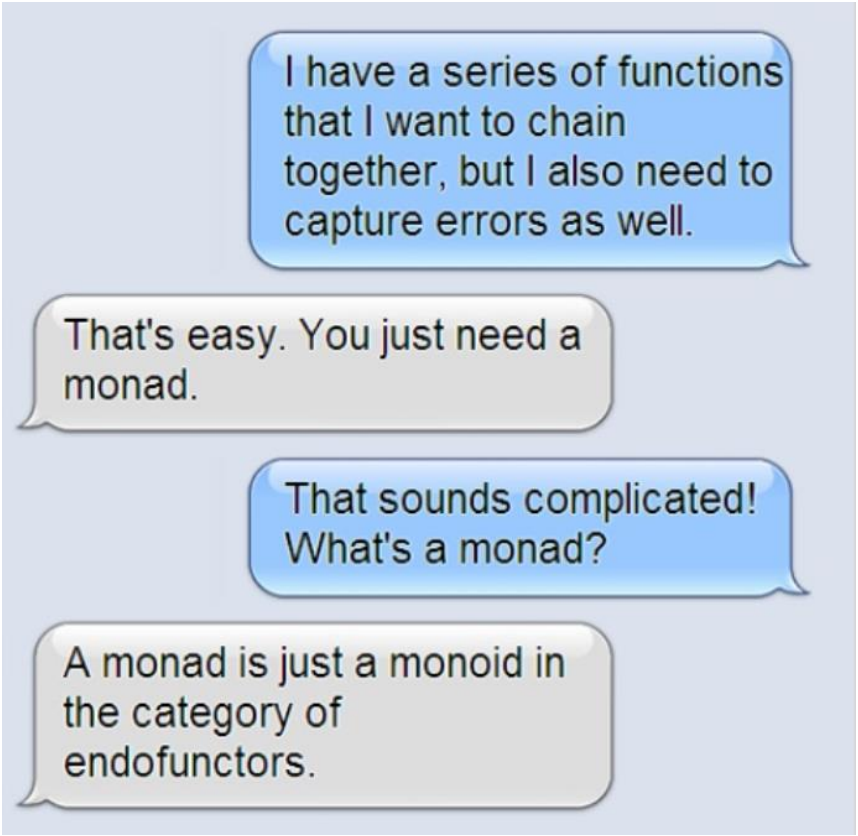
- Java 8 – с 2015 года;
- Scala – с 2015 года (2.10);
- Kotlin – с 2016 года (1.0-beta);
- Python – примерно с 2012;
- немного C# - 2012-2015;
- JavaScript - :D

Все думают, что ФП – это страшно



Страшные слова

- теория категорий;
- монада;
- функтор;
- каррирование;
- контр-вариантность;
- трейт;
- паттерн-матчинг;
- …



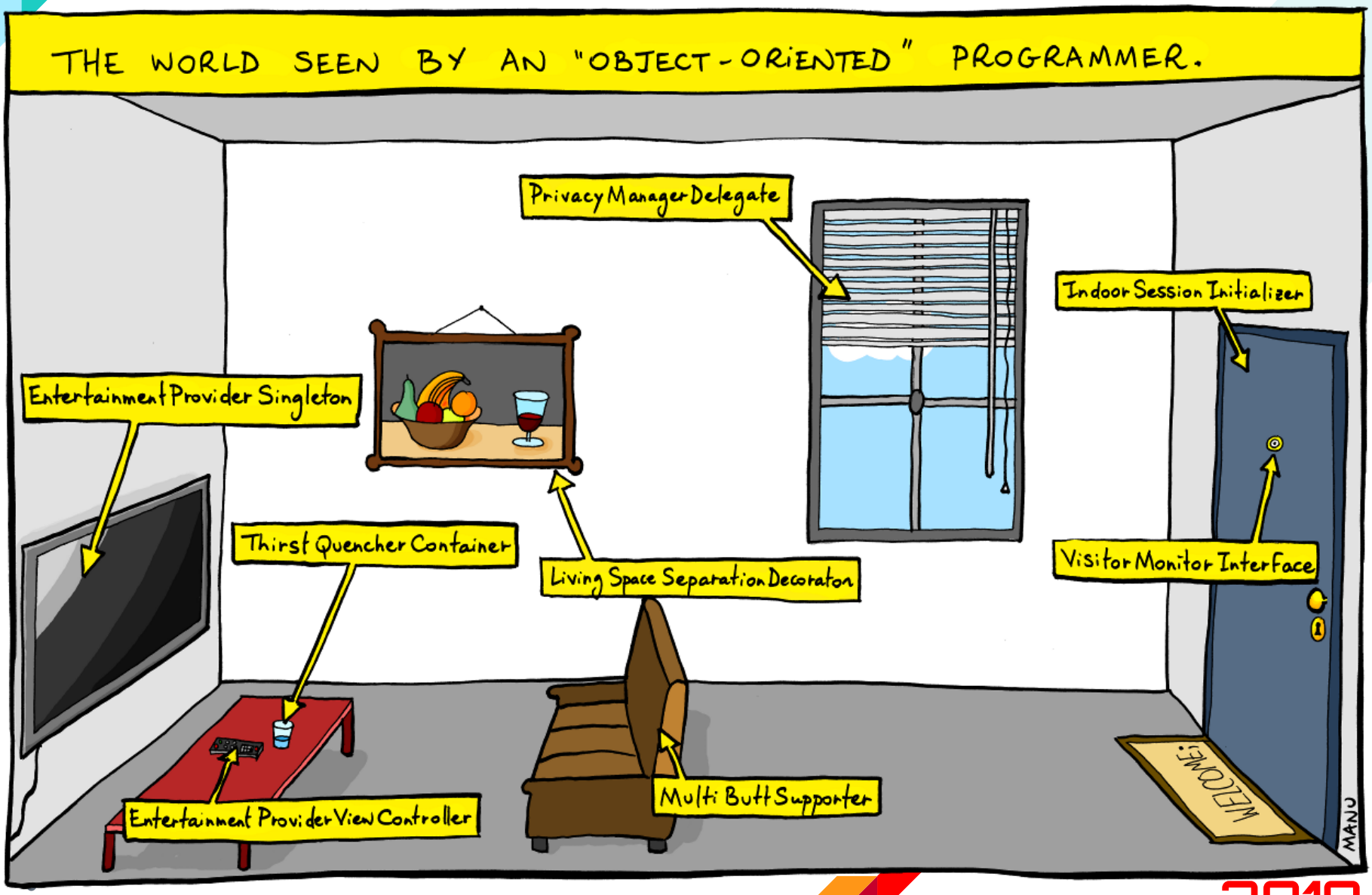
Страшные слова ООП

- инкапсуляция;
- полиморфизм;
- наследование;
- инъекция зависимостей;
- композиция;
- абстрактная фабрика;
- принцип Барбары Лисков;
- …

На самом деле

- другой уровень абстракции;
- писать плохо можно с любой парадигмой;
- писать хорошо сложно с любой парадигмой (см. enterprise FizzBuzz);
- можно знать не всю теорию;
- можно использовать не полностью (особенно в мультипарадигменных языках).

Где хорошо применять ФП?

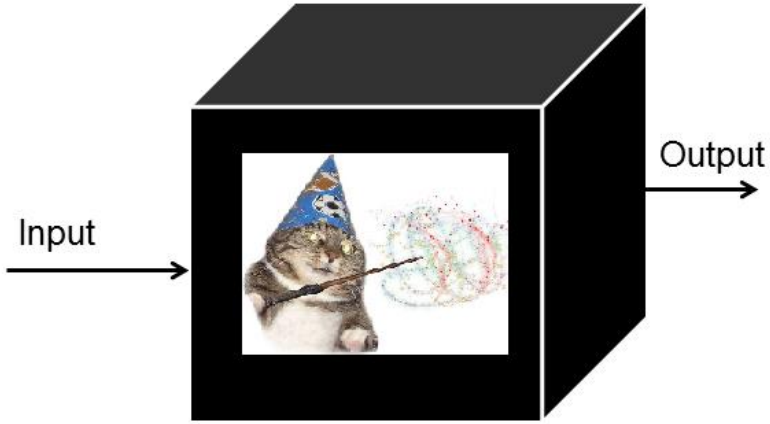


2019

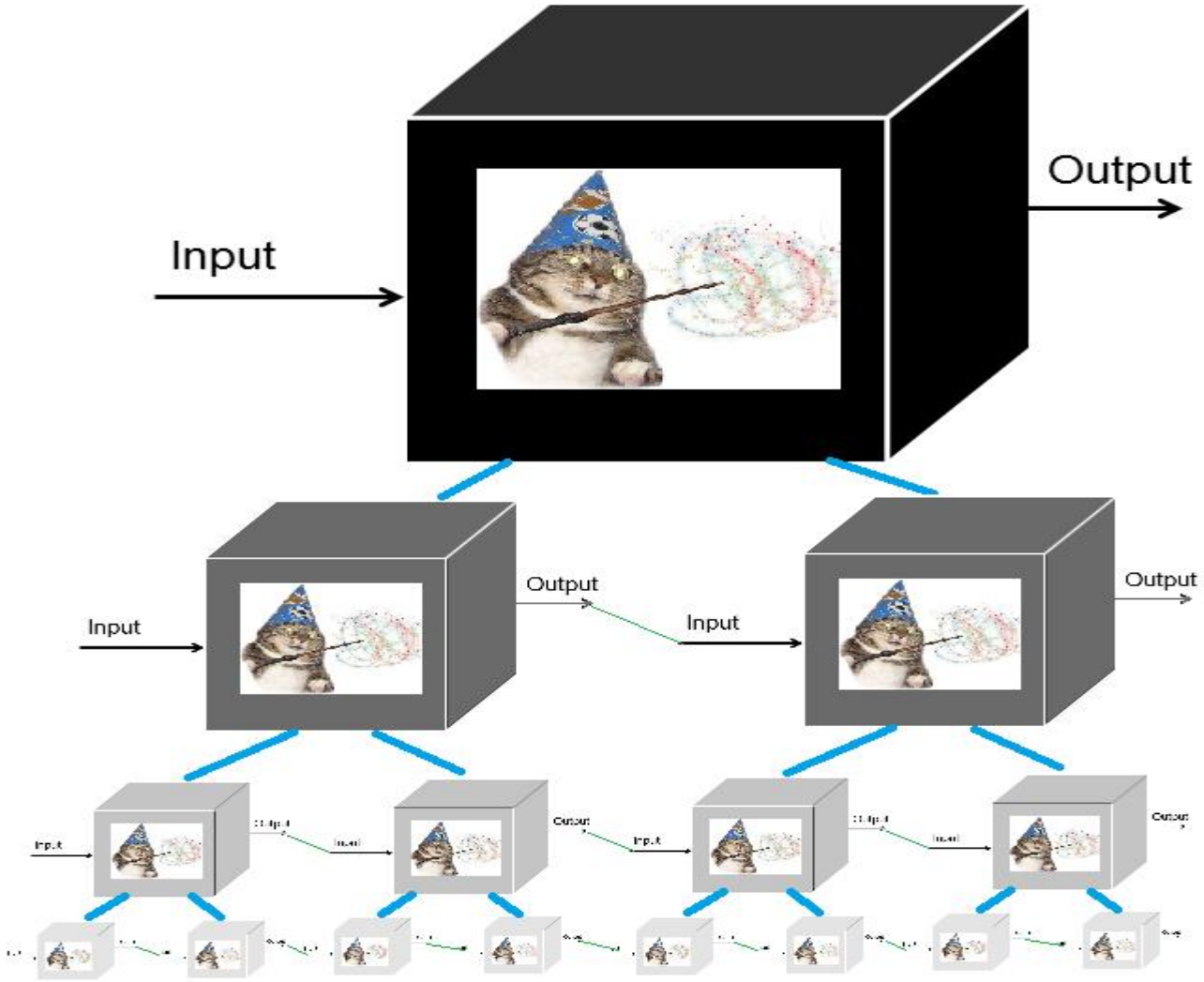
Где хорошо применять ФП?

Почти все ПО – это ETL:

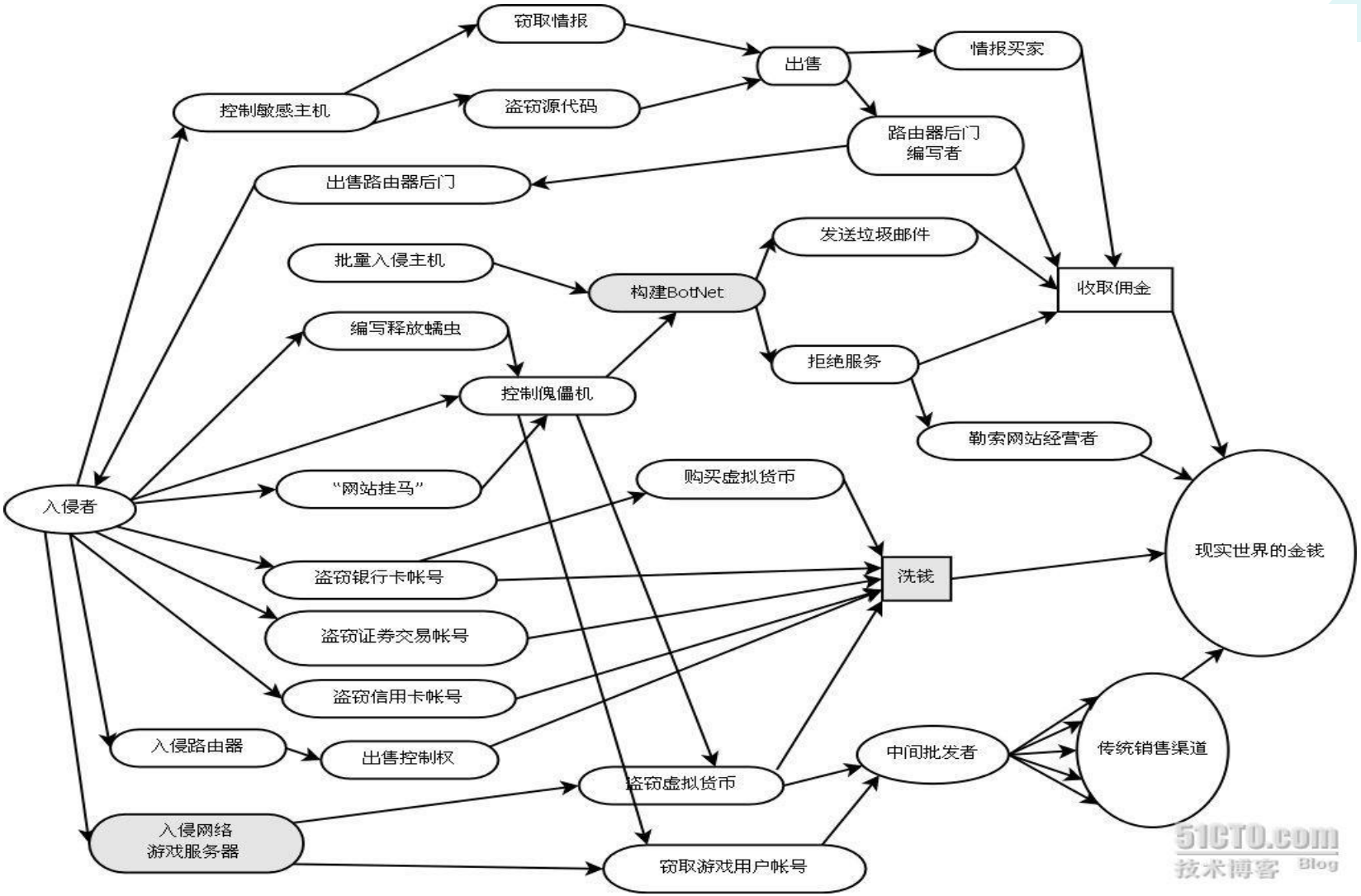
- взять данные (extract);
- преобразовать (transform);
- загрузить куда-нибудь (load).



Декомпозиция



Декомпозиция (реальность)



Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Do one thing and do it well

Хорошая функция:

- адекватное название;
- делает одну вещь;
- без побочных эффектов (чистая).

Do one thing and do it well

Хорошая функция:

- адекватное название;
- делает одну вещь;
- без побочных эффектов (чистая).

Другими словами:

- адекватное название;
- SRP;
- инкапсуляция.

Чистота функций

Плюсы:

- легче читать код (не надо думать о текущем состоянии);
- легче рефакторить;
- легче тестировать;
- легче оптимизировать – ленивые вычисления и мемоизация;
- легче распараллелить;

Чистота функций – работа с кодом

- нет состояния => меньше надо помнить;
- можно подставить результат => работа только с текущим уровнем абстракции (ссылочная прозрачность);
- нет состояния => меньше связность => легче рефакторить и изменять.

```
Parser parser = new Parser();
...
parser.setConfigProperty("line.ending", "\r\n")
...
parser.init(input);
...
ParserResult parserResult = parser.getResult();
```

Чистота функций – легче тестировать

- TDD;
- не надо тянуть всю иерархию вызовов;
- интеграционные тесты делать почти так же легко как модульные;
- можно повторно использовать входные данные – легче тестировать почти одинаковые сценарии;

Чистота функций - оптимизация

- ответ зависит только от входа => можно закэшировать/запомнить;
- ответ не зависит от внешних условий => можно давать когда попросят (лениво);
- можно переставлять вызовы функций;
- можно разбросать по потокам.

```
$ python2
...
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
...
$ python3
...
>>> range(10)
range(0, 10)
```

Параллелизм

- нет состояния => нечего синхронизировать

Параллелизм

- нет состояния => нечего синхронизировать;
- выше абстракции – меньше менять кода

```
val text = "..."
val wordCount = text
  .split(' ')
  .map(_.trim.toLowerCase)
  .filter(_.nonEmpty)
  .groupBy(identity)
  .mapValues(_.length)
```

```
val text = "..."
val wordCount = text
  .split(' ')
  .par
  .map(_.trim.toLowerCase)
  .filter(_.nonEmpty)
  .groupBy(identity)
  .mapValues(_.length)
```

- функции можно передавать по сети;

Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Иммутабельность

- никто не нагадит вам из другого потока;
- легко копировать;
- уже в вашем языке!

  - const, final, …

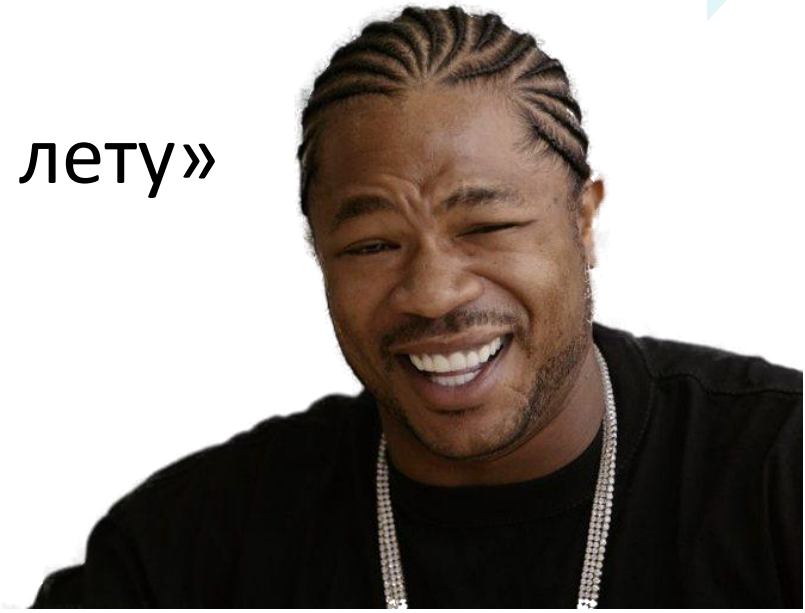
  - строки в Python, Java, C# …

Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Функция как значение

- стандартный подход на высоком уровне, детали – в передаваемой функции (паттерн «Стратегия»);
- создание поведения «на лету» («фабрика»);
- каррирование (это уже хардкорнее);
- частичное применение.



Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Система типов

- теория категорий;
- вывод типов;

```
InternationalCustomerOrderProcessor<AnonymousCustomer, SimpleOrder<Book>> orderProcessor =
createInternationalOrderProcessor(customer, order);

val orderProcessor = createInternationalOrderProcessor(customer, order);
```

- выявление ошибок на стадии компиляции;
- формальная верификация;
- композиция типов (ADT);

```
val httpResponse: Success | Error | Timeout = ...
```

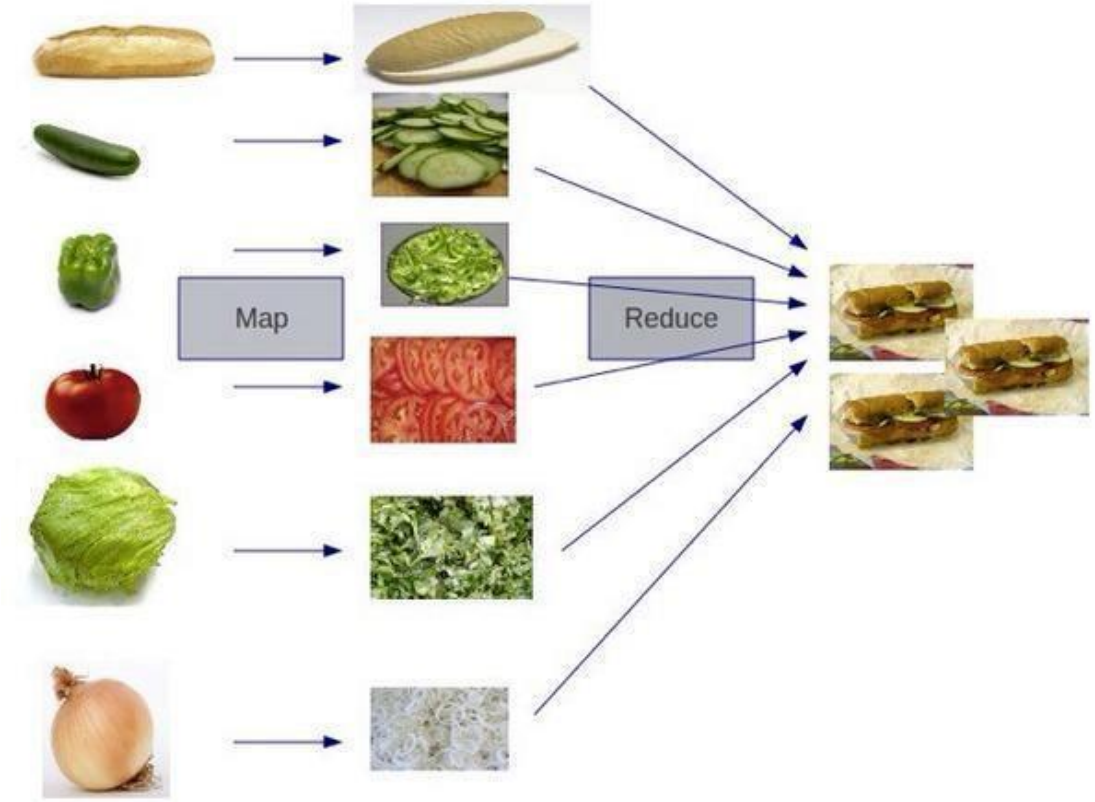
- pattern matching (switch на стероидах);
- data classes.

Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Работа с коллекциями

- map;
- filter;
- fold;
- reduce;
- groupBy;
- flatMap;
- …
- тысячи их!



Функциональный подход

- «чистые» функции;
- иммутабельность;
- функция – это тоже значение;
- мощная система типов;
- куча методов для работы с коллекциями;
- railway-oriented programming.

Railway-oriented programming

```
string UpdateCustomerWithErrorHandling()
{
  var request = receiveRequest();
  validateRequest(request);
  canonicalizeEmail(request);
  db.updateDbFromRequest(request);
  smtpServer.sendEmail(request.Email)

  return "OK";
}
```

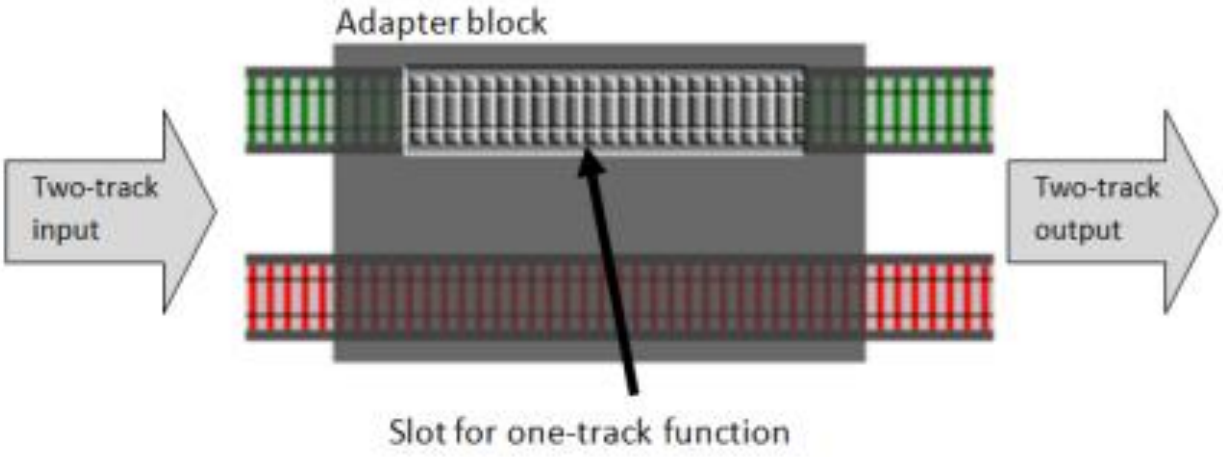
Railway-oriented programming

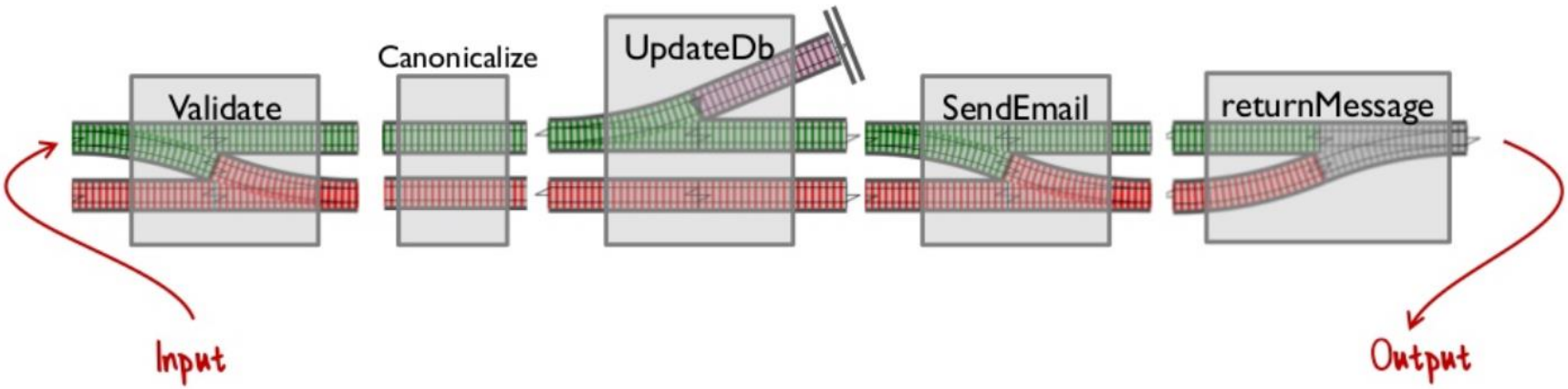
```
string UpdateCustomerWithErrorHandling()
{
  var request = receiveRequest();
  var isValidated = validateRequest(request);
  if (!isValidated) {
     return "Request is not valid"
  }
  canonicalizeEmail(request);
  try {
    var result = db.updateDbFromRequest(request);
    if (!result) {
      return "Customer record not found"
    }
  } catch {
    return "DB error: Customer record not updated"
  }

  if (!smtpServer.sendEmail(request.Email)) {
    log.Error "Customer email not sent"
  }

  return "OK";
}
```

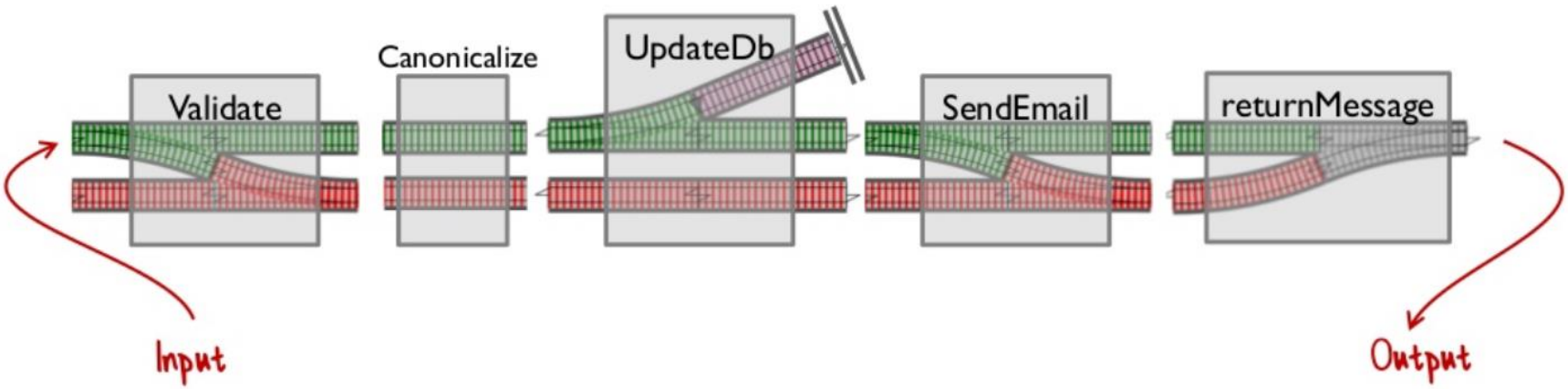
Railway-oriented programming





Railway-oriented programming

```
let updateCustomer =
  receiveRequest
  >> validateRequest
  >> updateDbFromRequest
  >> sendEmail
  >> returnMessage
```



ROP vs исключения

Варианты развития событий:

- все хорошо;
- ожидаемая ошибка;
- неожиданная ошибка; <---- тяжело!

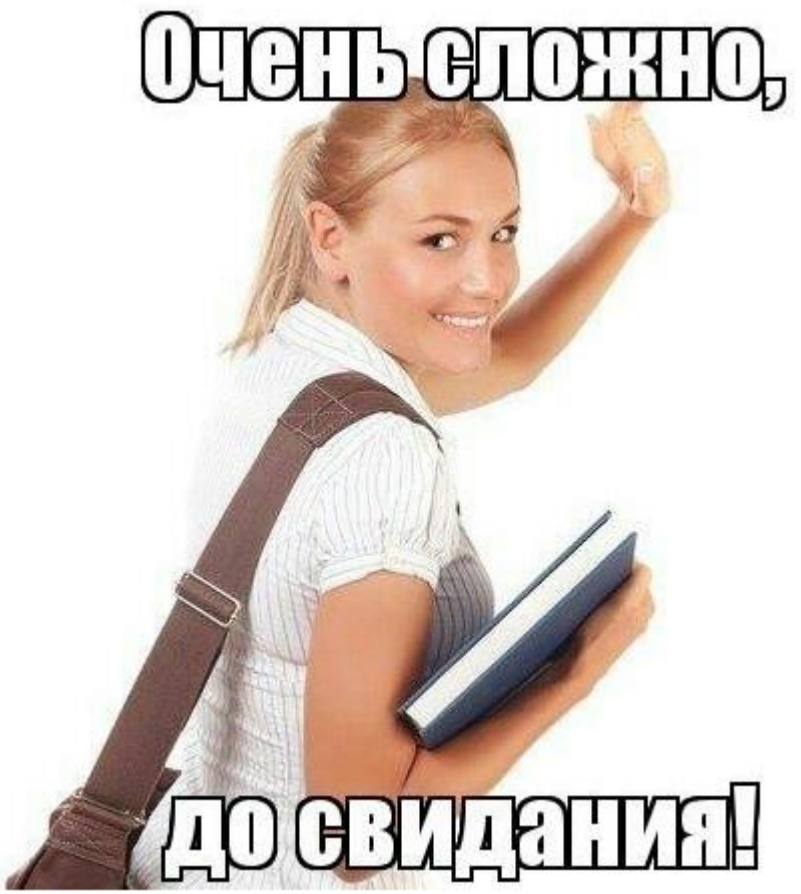
Даже в мире ФП есть пользовательский ввод и чужое API => неисчерпаемый источник неожиданностей;

Минусы

- сложно;
- производительность;
- работает не всегда;

Сложна

- надо учиться новому;
- надо думать;
- надо знать паттерны, и методы коллекции;
- если по хардкору – иногда придется решать ребусы;



Производительность

- нагрузка на GC;
- часто упирается в другое;
- оптимизация – отдельная задача, не всегда нужна;
- компиляторы умнеют;
- зависит от ситуации;
- железо дешевле программистов.

Не всегда работает

- 100% чистота – нереально;
- иногда дешевле решить «в лоб», чем решать ребус;
- не все задачи легко выразимы: одноразовые скрипты, эмуляторы…

Что по факту у нас

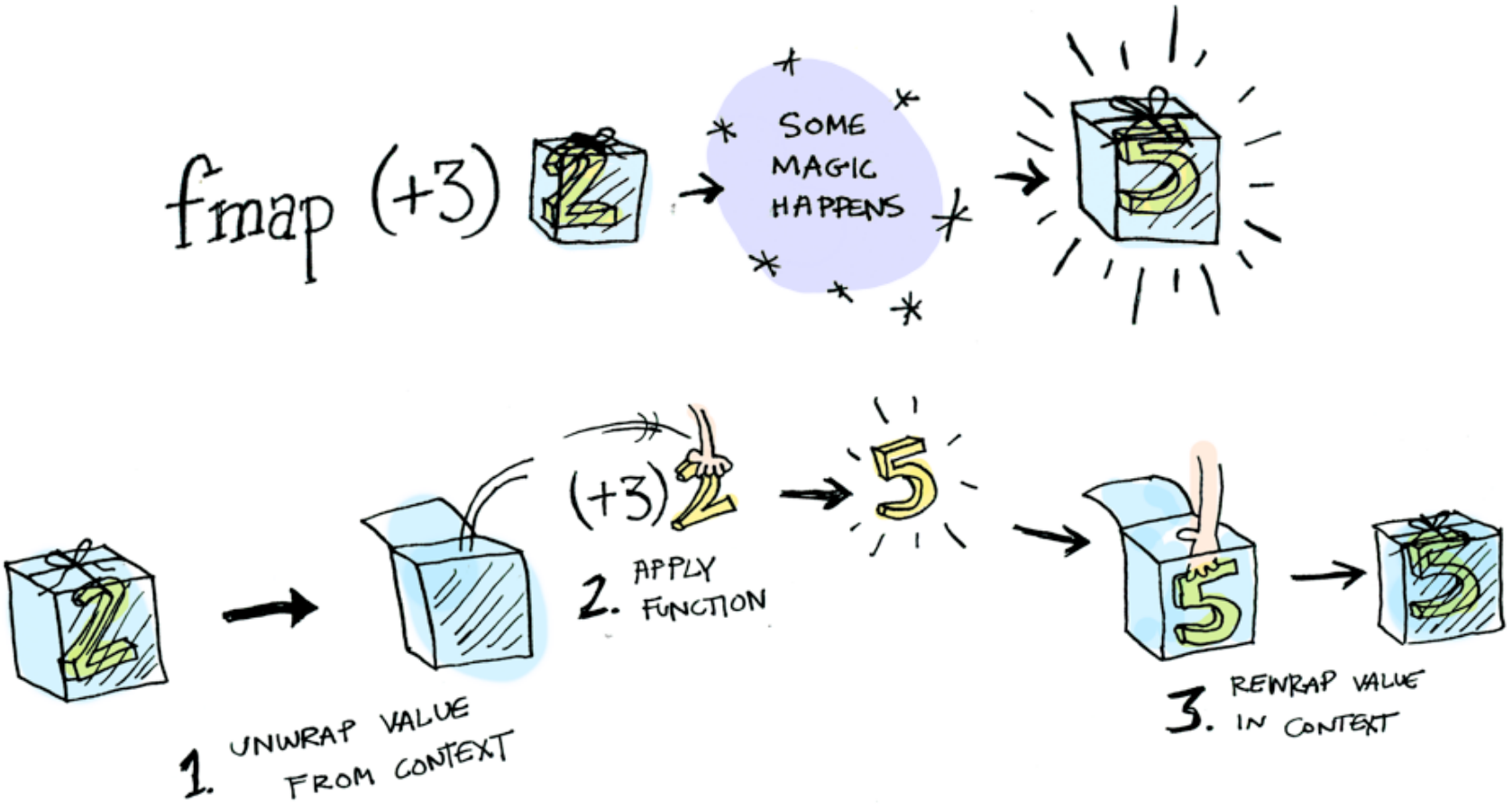
- «серая зона»;
- комбинация ООП и ФП;
- стараемся делать «чисто»;
- работа с коллекциями;
- data classes;
- трейты («собери класс по кусочкам»);
- паттерн-матчинг;
- с исключениями.

Функтор и монада

Функтор: штука, для которой есть map.

Функтор и монада

Функтор: штука, для которой есть map.



Функтор и монада

Функтор: штука, для которой есть map.

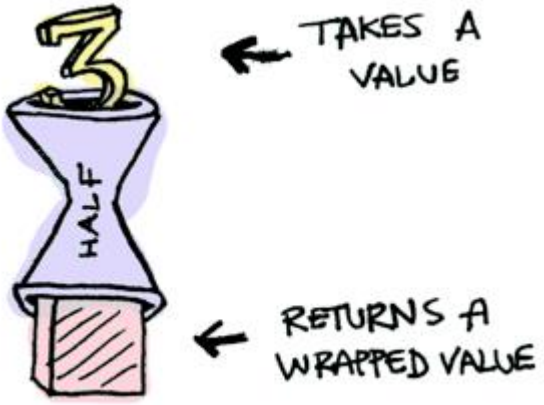
2 закона:

functor.map(id) == functor

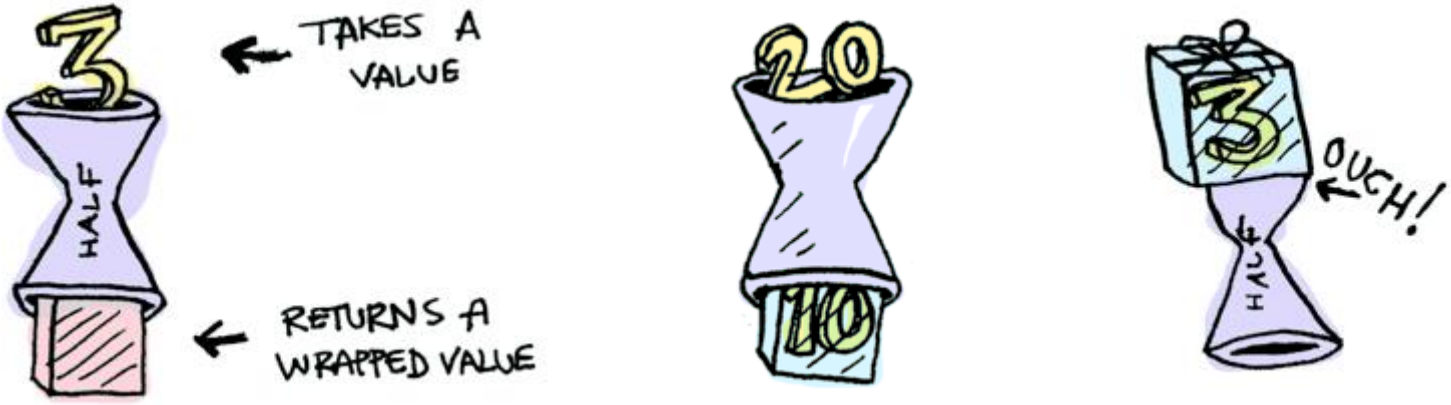
functor.map(f).map(g) == functor.map(g ◦ f)

Функтор и монада

Функтор и монада



Функтор и монада



Монада: штука, для которой есть flatMap.

```
val valueInBox: Monad[T] = ...
def func: T => Monad[R] = ...

val result: Monad[R] = valueInBox.flatMap(func)
```

Заключение

- ФП – это не страшно;
- тренд на включение элементов ФП в языки;
- можно использовать в продакшене;
- не перебарщивать;
- лучше использовать совместно с другими парадигмами.

Спасибо за внимание

Вопросы?

Литература

http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html

https://medium.com/@lettier/your-easy-guide-to-monads-applicatives-functors-862048d61610

https://www.slideshare.net/ScottWlaschin/fp-patterns-ndc-london2014

https://www.slideshare.net/ScottWlaschin/railway-oriented-programming

https://habr.com/ru/post/337880/

http://steve-yegge.blogspot.com/2006/03/execution-in-kingdom-of-nouns.html

http://leftoversalad.com/c/015_programmingpeople/

https://hackernoon.com/why-functional-programming-matters-c647f56a7691

https://l-infinity.space/pssst-hey-kid-wanna-be-a-functional-programmer-cefcb5a48a8

https://www.slideshare.net/vseloved/can-functional-programming-be-liberated-from-static-typing

https://twitter.github.io/scala_school/index.html