

Modes of Composition in functional Scala programming

Ben Hutchison

Lambdajam Online 2020

Pure Functional Programming

- Functional Programming == “Programming with Functions”
 - $f: (P1, P2 \dots Pn) \Rightarrow R$
 - Only inputs are parameters $P1, P2 \dots Pn$. Not affected by anything else.
 - Computes or evaluates a result R . Has no other effect on the world.
 - Deterministic & side-effect free
 - Referentially transparent - a function invocation is equivalent to, and replaceable by, its result
 - $f(P1, P2.. Pn) == R$
- Great conceptual model
 - But how to extend to applied programming where side-effects are desired?

Effectful Functional Programming

- Effectful FP: a variant for expressing *effects & actions*
 - $f: (P_1, P_2 \dots P_n) \Rightarrow F[R]$
 - $F[R]$ is an *effectful* value
- When an effectful value is run
 - The effects or actions described by F happen. These might be “side-effects”, external actions upon the world. But there are other effects like errors, or asynchrony, that can be included in F without affecting the world.
 - A value R is yielded or computed

A Spectrum of Effectful Values

An effectful value can be a pure action, where the result value is trivial

```
IO(println("hello world")): IO[Unit]
```

An effectful value can be a pure value, where the effect is trivial

```
Id("hello world"): Id[String]
```

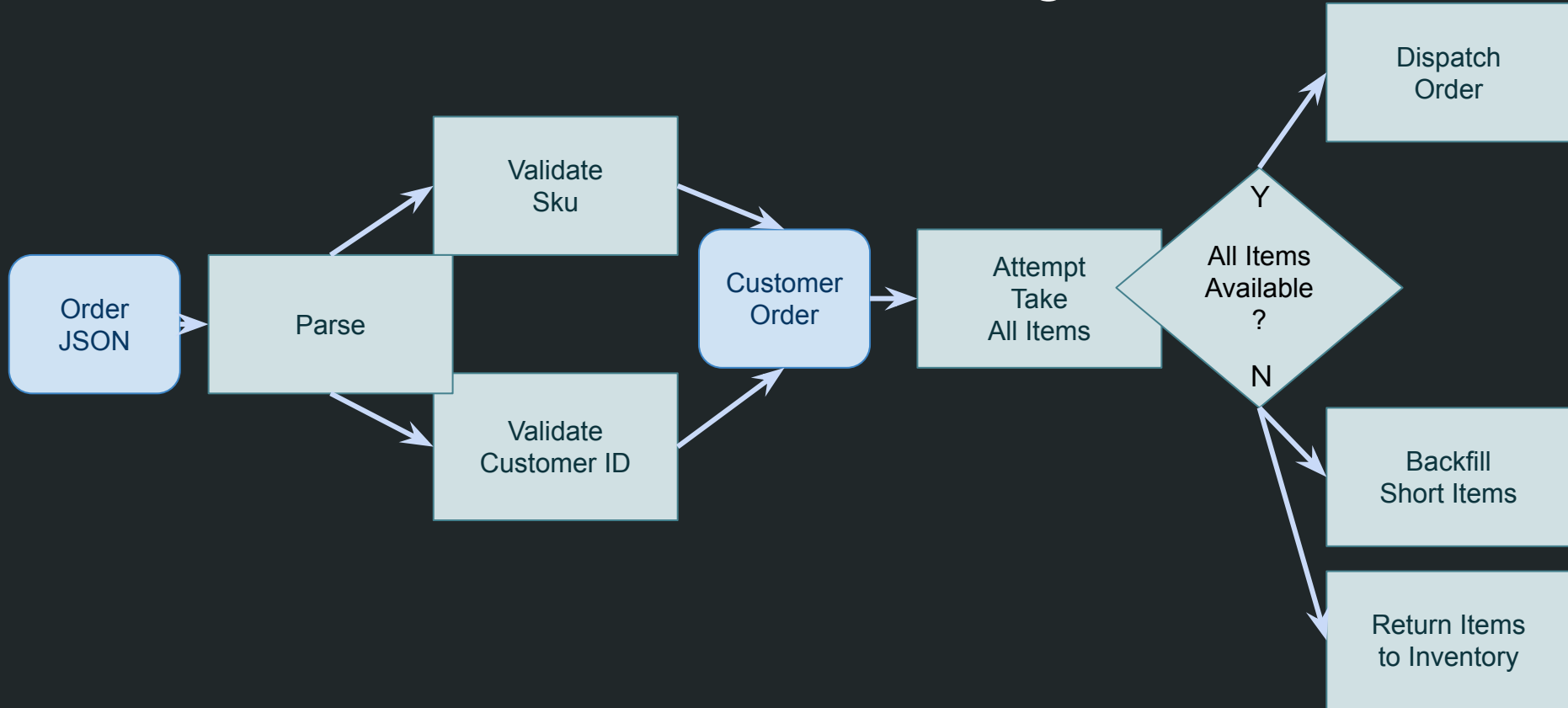
An effectful value can represent some effects that yield a value

```
IO(datastore.readCustomer(12546)): IO[Customer]
```

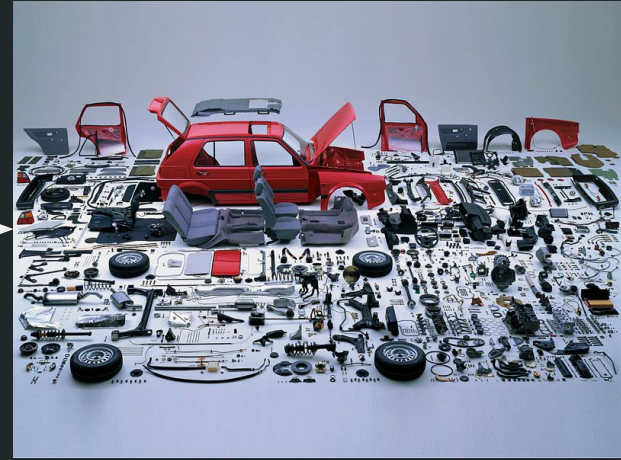
Turning Effectful Values into Effects

- Effectful functional programs compose effectful values together into a complete program that causes the effects in the desired sequence
 - All reads, writes and interactions with the environment, the screen, network, storage, the system clock etc
 - *How* they are composed is the topic of this workshop - we'll examine that in detail
- Running Effectful Functional Programs
 - At the top level, the value representing the whole program is set in motion with a single call that triggers effects (in these examples, `unsafeRunSync`)

Business Scenario: Order Processing



Composition and Decomposition



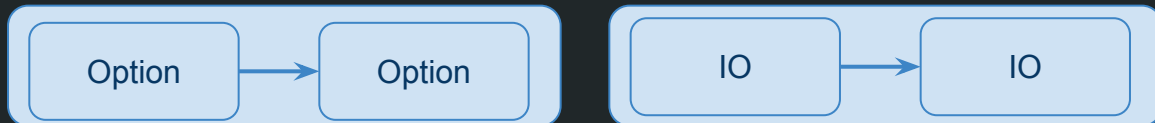
- How are large systems assembled from smaller components?
- How can large problems be broken down into many smaller independent ones?

Modes of Composition

- An effectful value depends upon the result of another, or must happen after
 - Monadic operators
- Independent effectful values can be run together concurrently
 - Applicative (or Semigroupal) operators
- An effectful value can incorporate several different effects
 - Tagless Final style and/or Monad Transformers
- An effectful value is composed of repeated actions or computations
 - FS Streams
- An effectful value and it's (approximate) inverse are paired together to create an effectful isomorphism
 - Resources, Codecs, Functional Optics, Sagas
- An effectful value is internally composed of layers
 - Different rules or invariants apply in each layer

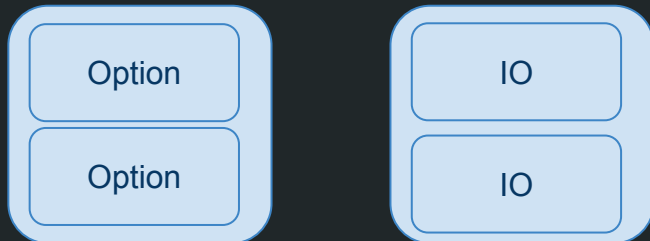
Modes of Composition

- *An effectful value depends upon the result of another, or must happen after*
 - *Monadic operators*
- **FlatMap**, aka **Bind**, composes two effectful values into a larger value of the same type
 - The second operations happens after the first and may depend upon the value it yields
 - Flatmap operator is written symbolically as `>>=`
- The “meaning” of the composition depends upon the type of effect present in the value



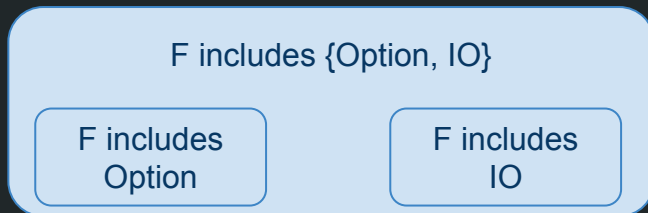
Modes of Composition

- *Independent effectful values can be run together concurrently*
 - *Applicative (or Semigroupal) operators*
- Multiple independent effectful values can be composed in parallel into a larger value of the same type using `Semigroupal.tupledN` operators
 - Operations can't depend upon each other's outputs and happen in an undefined order
- The “meaning” of the composition still depends upon the type of effect present in the value



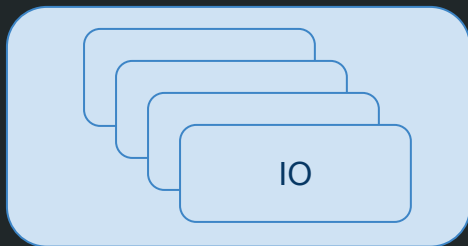
Modes of Composition

- *An effectful value can incorporate several different effects*
 - *Tagless Final style and/or Monad Transformers*
- If part of our program includes an Option effect, and another includes an IO effect, how can we compose them together?
 - The effects present in our program are a *Set of Constraints*. We can add to the set when needed.



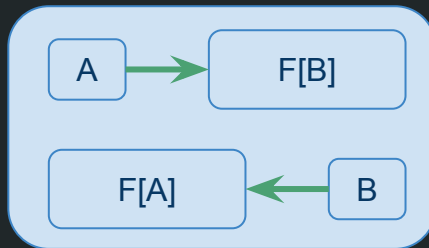
Modes of Composition

- *An effectful value is composed of repeated actions or computations*
 - *FS Streams*
- Functional Streams model effectful values that repeat a variable or infinite number of times, either by
 - polling or reading the environment to pull each new value
 - generating each value from an internal computation process



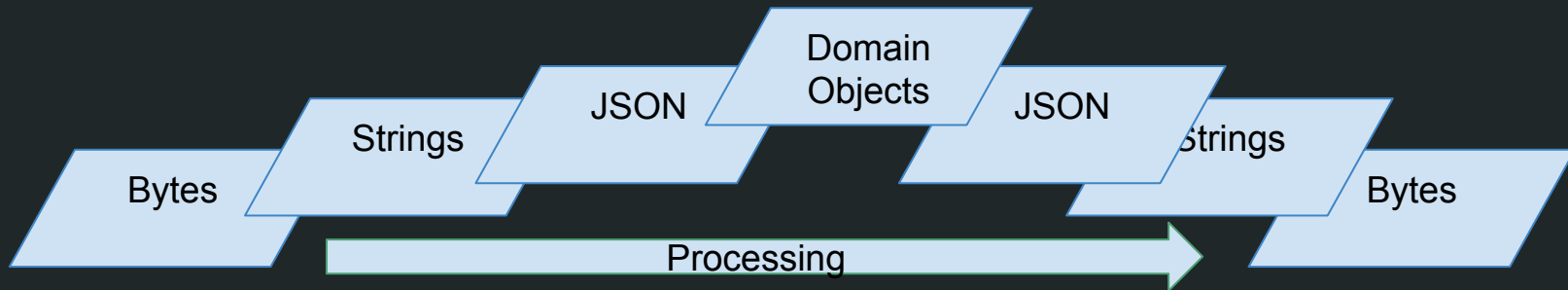
Modes of Composition

- *An effectful value and its (approximate) inverse are paired together to create an effectful isomorphism*
 - *Resources, Codecs, Functional Optics, Sagas*
- The inverse of a function reverses, or undoes, its effect
- When paired, a function and its inverse enable a program to “travel” between a mode where the function applies, and one where its doesn’t
 - Turns out useful in a broad variety of superficially unrelated situations



Modes of Composition

- An effectful value is internally composed of layers
 - Different rules or invariants apply in each layer
- The typical structure sees weakly typed layers on the outer rings of an application and one to many more strongly typed interior layers
 - Data representation is different in each layer



Workshop Approach: Type-driven Development

- The hands-on aspect of this workshop takes a “color-by-numbers” approach
- Existing code scaffolds have “typed-holes” which participants need to complete
- Aka “Type-Driven Development”
 - approach advocated by Edwin Brady (creator of the Idris language) where missing parts of a program, described by a type, are gradually filled in
- Using compositional operators to combine parts into a working whole

