Tagless Final in Scala

A functional programming practice in Scala

Sean Sun @ 23th March, 2023

Agenda

- Preface
- What and Why is “tagless final”?
- Example: A simple example application with call and cache functionality
- Wrapping Up
- Q and A

Preface

About me

- Data Engineer @ GoGoX
- Enthusiast of Scala and FP and have been learning these since 2015
- Mathematical Background in college

- https://github.com/TseEnSun
- https://www.linkedin.com/in/tseensun/

Preface

About this Talk

- Something I learned recently, and it works fine for me.
- Most of the content and practice come from a book, Practical FP in Scala.
- No need to wait to become a master and then share.

Practical FP in Scala

By Gabriel Volpe

https://leanpub.com/pfp-scala

What is Tagless Final?

From Papers

- A solution to the Expression Problem.
- Describes the initial and final encoding.
- The problem of Tag and Tagless.
- Examples in Haskell or OCaml.
- Reference: https://okmij.org/ftp/tagless-final/

What is Tagless Final?

From Papers

- A technique for representing typed higher-order languages in a typed metalanguage, along with type-preserving interpretation, compilation, and partial evaluation.
- The final encoding alternative to the traditional, or ‘initial’ encoding.
- The final encoding represents all and only typed object terms without resorting to generalized algebraic data types, dependent, or other fancy types.
- The final encoding lets us add new language forms and interpretations without breaking the existing terms and interpreters.

What is Tagless Final?

From Papers

- A technique for representing typed higher-order languages in a typed metalanguage, along with type-preserving interpretation, compilation, and partial evaluation.
- typed higher-order languages => algebras
- type-preserving interpretation… => interpreters

What is Tagless Final?

From Book

- Not a new concept; it could be called interface with a higher-kinded type.
- Tagless Final Algebra fit for encoding business concepts.
- Algebra is a simple interface abstract over the effect type using a type construct F[_]

What is Tagless Final?

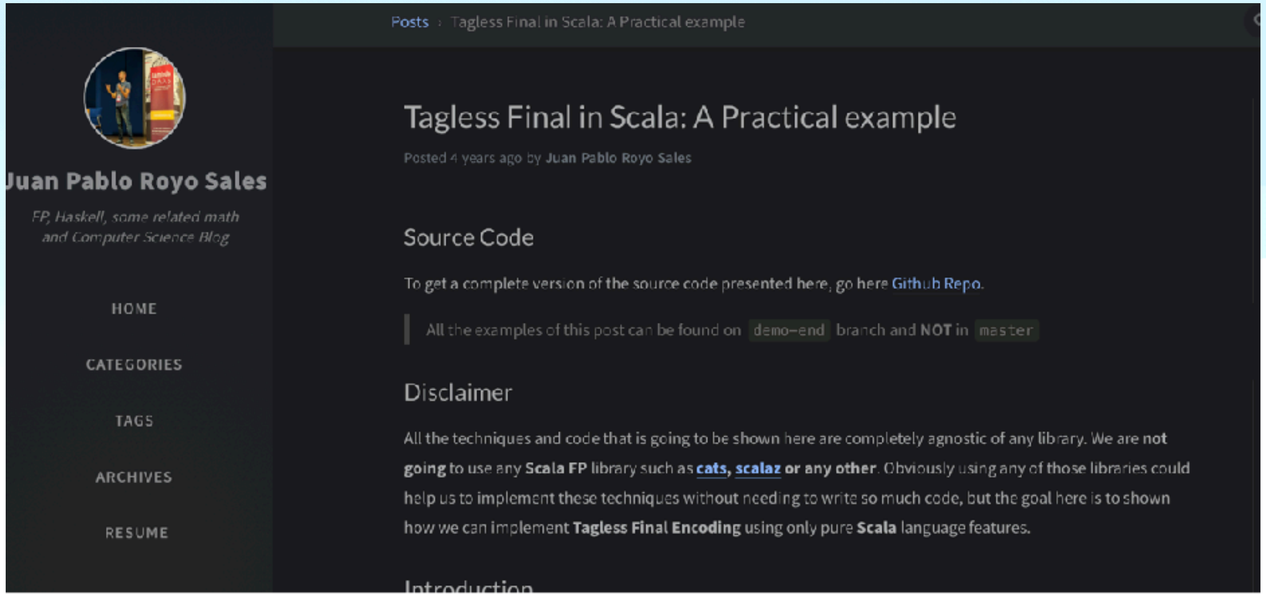
From Book

- Final tagless encodings have little to do with type classes
- Type classes should have coherent instances.
- Tagless Final Algebra could have many implementations, commonly called interpreters.

What is Tagless Final?

From Blogs

- https://jproyo.github.io/posts/2019-02-07-practical-tagless-final-in-scala.html
- A pure Scala example of Tagless Final Encoding
- **Algebras**: Set of operations over a Structure.
- **Interpreter**: The way those operations behave according to a specific Type.

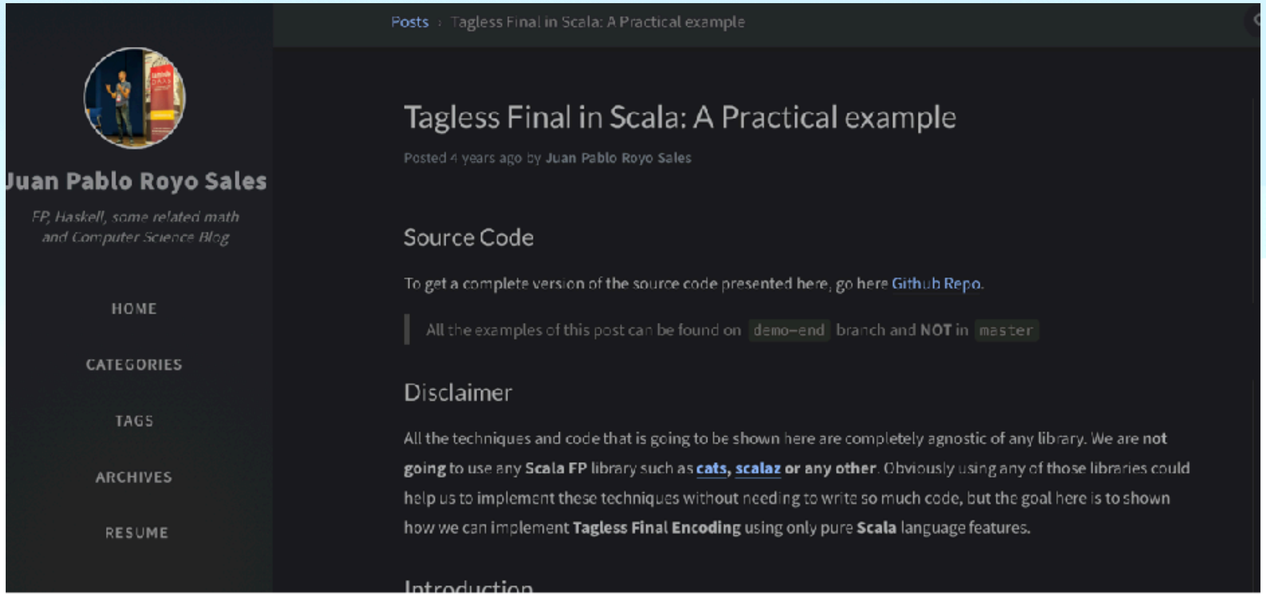


What is Tagless Final?

From Blogs

- **Algebras**:
  In programming, the idiom could be a set of functions that operates over some Type.

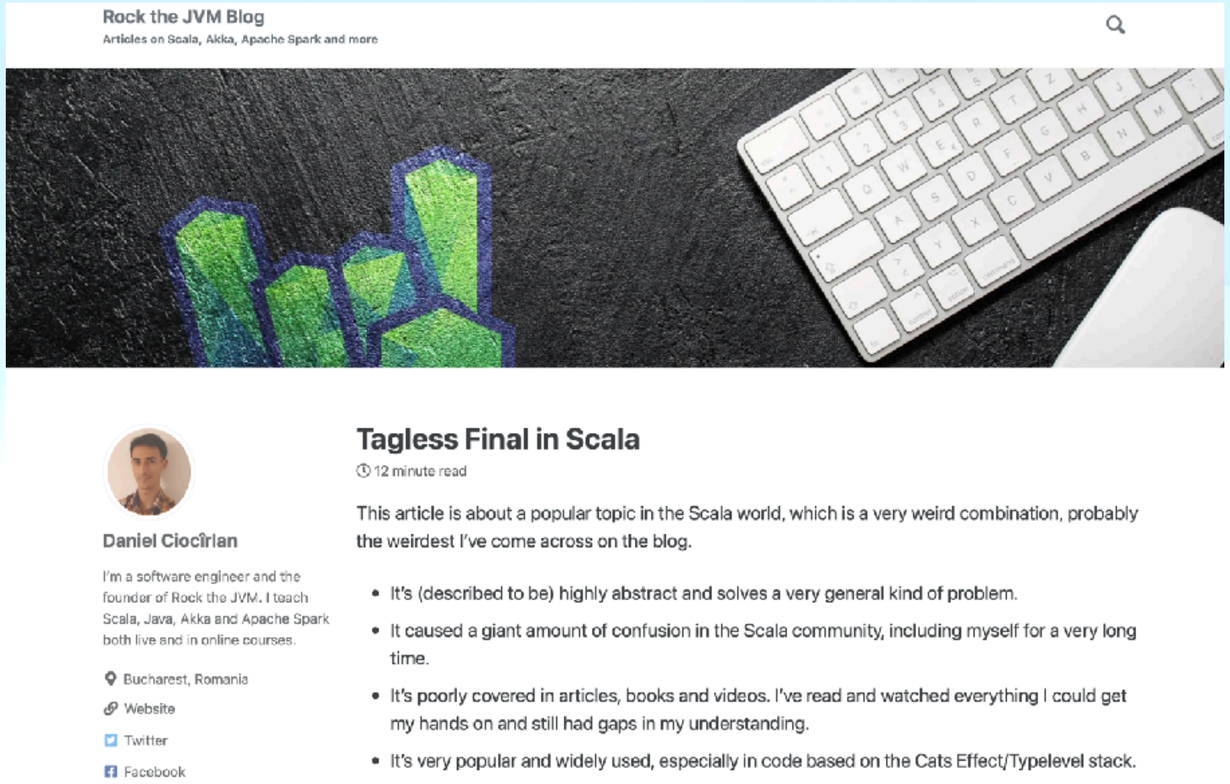
- **Interpreter**:
  In programming, the idiom is the implementation of those functions depending on the specific Type.



What is Tagless Final?

From Blogs

- https://blog.rockthejvm.com/tagless-final/
- Think about TF as a “design pattern.”
- The original paper is much more than a design pattern, it’s a way of creating new “languages.”



What is Tagless Final?

From Blogs

- From Rock the JVM Blog:

  The only overlap is in how the code ends up looking and the functionality restriction:

  - type classes are sets of functionalities that you want to offer to some types and not to others
  - tagless final intends to prove the correctness of expressions of some types and not of others

- They call it tagless-final-with-type-classes.
- We write very general code regarding an effect type, and finally, we plug it in at the “end of the world,” where the type class instances are brought into scope.

What is Tagless Final?

The Practice from Book

- Tagless Final encoding with Type Classes
  - Algebra (encoding the business logic)
  - Interpreters (implementation of algebra)
  - Programs (make use of algebras or other programs)

Why Tagless Final?

- TF is a great technique used to structure purely functional applications.
- With the Cats Effect, there is no ZLayer-like framework.
- A way to construct your Application.

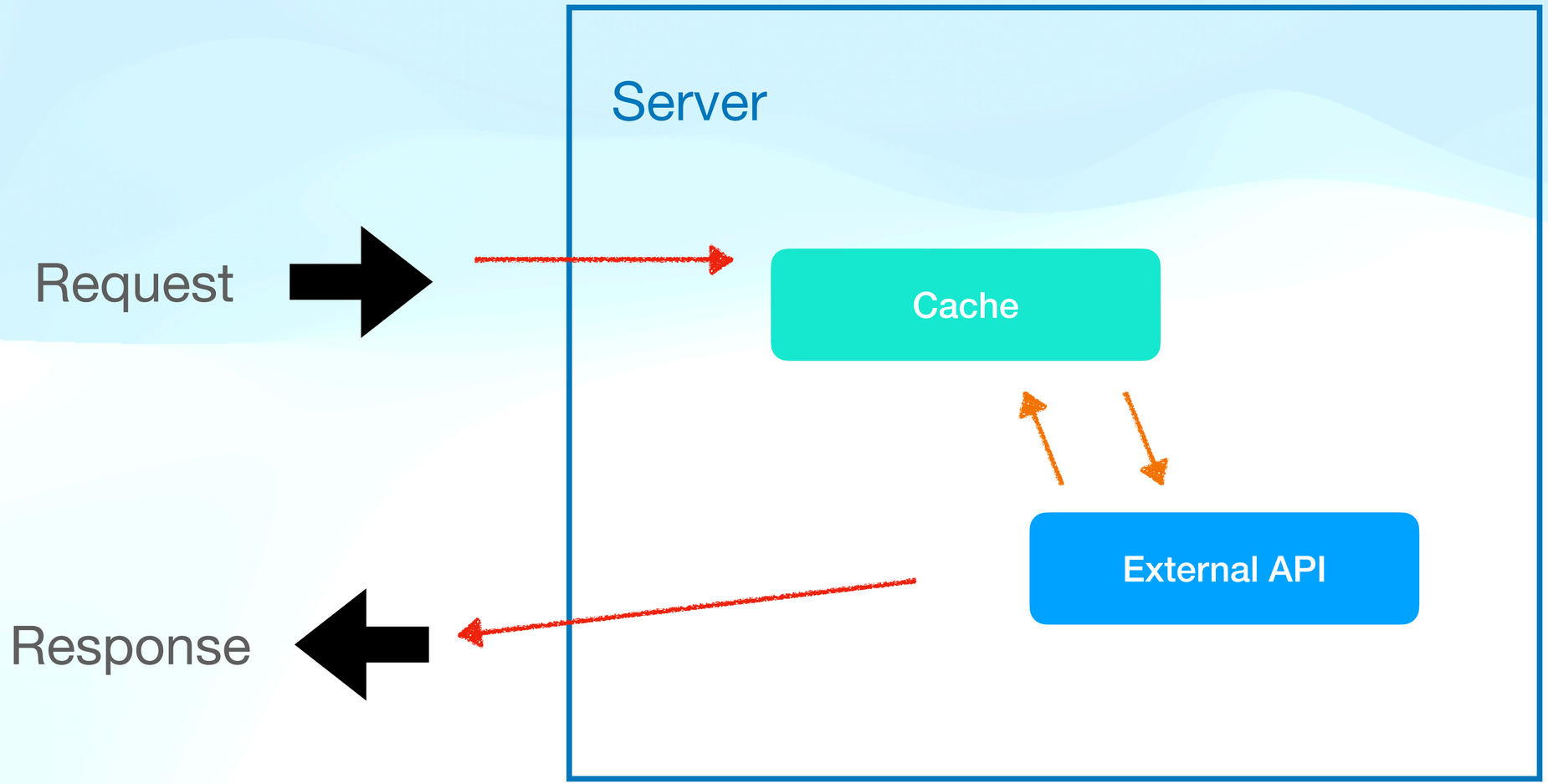
Weather Application

- Requirement:
  An HTTP API for querying the current temperature in celsius and humidity.
  Source the weather information from external API, but we don’t want to cost too much.

- Simple solution:
  Since the weather does not change too much in a short period, we could cache the result to reduce the real external API call.

Example Application

Architecture



Weather Application

Algebras

- Cache Algebra

```
trait WeatherCacheAlgebra[F[_]] {
  def set(key: String, weather: Weather): F[Unit]
  def get(key: String): F[Option[Weather]]
}
```

- External API Algebra

```
trait WeatherApiAlgebra[F[_]] {      You, 5 days ago • Bui
  def get(city: City): F[Either[String, Weather]]
}
```

Weather Application

Interpreters

- Cache Interpreter

```scala
object WeatherCache {
  def make[F[_]: Sync](
    inMemoryStore: Ref[F, Map[String, (Long, Weather)]]
  ): WeatherCacheAlgebra[F] =
    new WeatherCacheAlgebra[F] {

      override def set(key: String, weather: Weather): F[Unit] = ???

      override def get(key: String): F[Option[Weather]] = ???

    }
}
```

Weather Application

Interpreters

- External API Interpreter

```
object WeatherApi{

  def make[F[_]: Concurrent](client: Client[F]): WeatherApiAlgebra[F] =

    new WeatherApiAlgebra[F] {
      val weatherApiToken = "25393e6d0e674ae0a1e54447231203"
      val baseUri = uri"https://api.weatherapi.com"
      val uriWithPath = baseUri.withPath(path"/v1/current.json")

      def get(city: City): F[Either[String, Weather]] = ???
    }

}
```

Weather Application

Programs

- Algebra

```
trait WeatherProgramAlgebra[F[_]] {
  def getCityWeather(city: City): F[Either[String, Weather]]
}
```

Weather Application

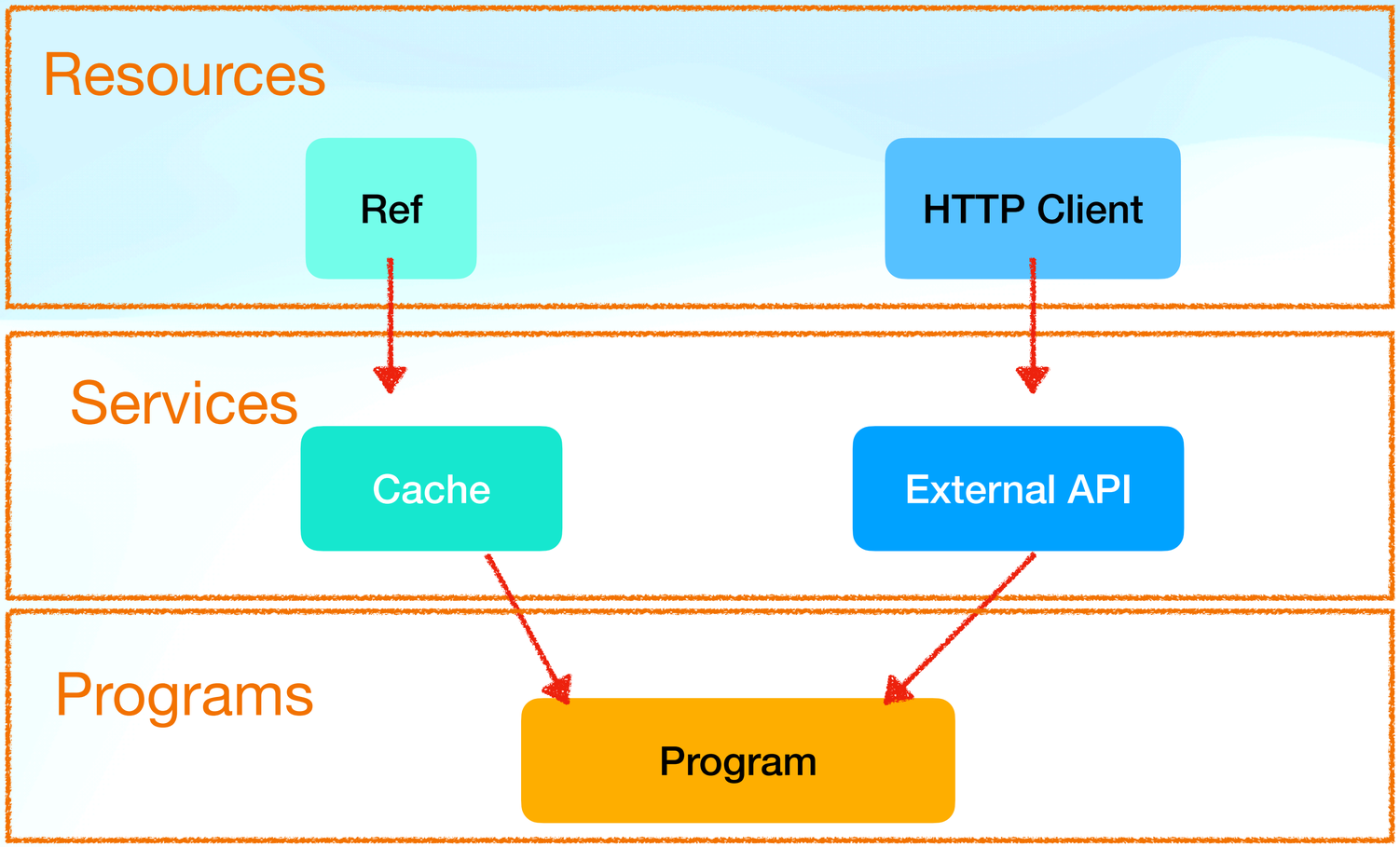
Programs

- Interpreter

```scala
object WeatherProgram {
  def make[F[_]: Monad : Logger](
    cacheService: WeatherCacheAlgebra[F],
    apiService: WeatherApiAlgebra[F]
  ): WeatherProgramAlgebra[F] = new WeatherProgramAlgebra[F] {

    override def getCityWeather(city: City): F[Either[String, Weather]] =
      for {
        cacseResult <- cacheService.get(city.name)
        result <- cacseResult match {
          case None => ???
            /*
              Cannot get result from Cache, make a call to external API.
              If there something, make a cache of it
            */
          case Some(weather) => ???
            // Get the result from Cache and return.
        }
      } yield result
  }
}
```

Weather Application

Resources



Weather Application

Testing

- An algebra should permit multiple interpreters so it is easy to create one for testing.
- For instance, the weather program depends on cache and external API services.
- Note: the program and interpreter know only the algebras

Weather Application

Testing

- Let's create good and bad cache interpreters

```scala
def successCache(weather: Weather): WeatherCacheAlgebra[IO] =
  new WeatherCacheAlgebra[IO] {
    override def set(key: String, weather: Weather): IO[Unit] = IO.unit
    override def get(key: String): IO[Option[Weather]] = IO.pure(Some(weather))
  }

def noHitCache: WeatherCacheAlgebra[IO] =
  new WeatherCacheAlgebra[IO] {
    override def set(key: String, weather: Weather): IO[Unit] = IO.unit
    override def get(key: String): IO[Option[Weather]] = IO.pure(None)
  }
```

Weather Application

Testing

- Test Case: If the cache is hit, then always return the right result.

```
test("If cache hit, then always right result") {

  val program = WeatherProgram.make(
    successCache(dummyWeather),
    successApi(dummyWeather)
  )
  assertIO(program.getCityWeather(dummyCity), Right(dummyWeather))

  val programWithErrorApi = WeatherProgram.make(
    successCache(dummyWeather),
    errorApi
  )
  assertIO(program.getCityWeather(dummyCity), Right(dummyWeather))
}
```

Weather Application

HTTP Server

- Routes

```
final case class WeatherRoutes[F[_]: Sync](
  weather: WeatherProgramAlgebra[F]
) extends Http4sDsl[F] {

  private[routes] val prefixPath = "/currentWeather"

  private val httpRoutes: HttpRoutes[F] = HttpRoutes.of[F] {
    case GET -> Root :? CityQueryParamMatcher(city) => ???
      // Call the weather query to the program, then construct the Response
  }

  val routes: HttpRoutes[F] = Router(
    prefixPath -> httpRoutes
  )
}
```

Weather Application

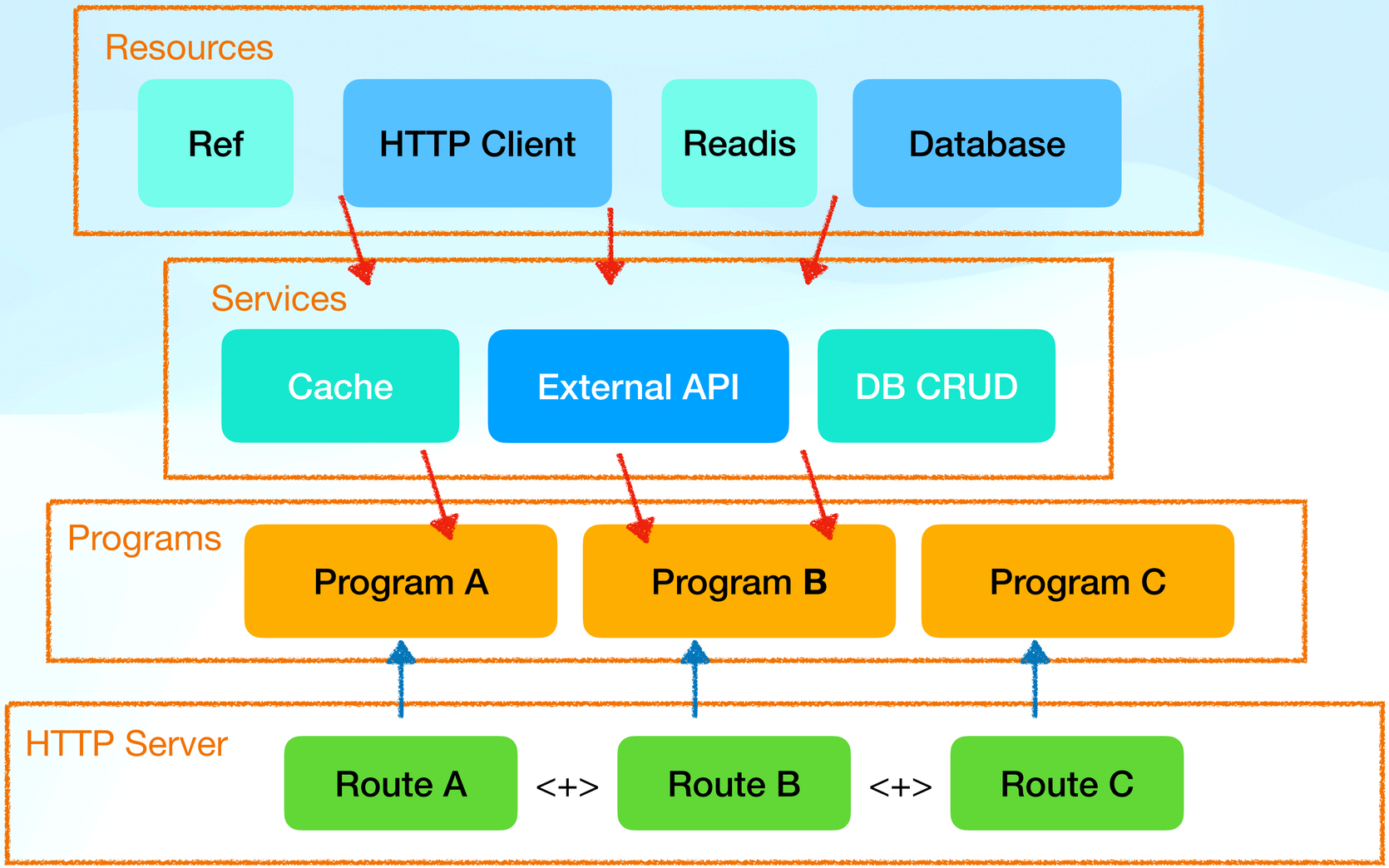
HTTP Server

- Server

```
object WeatherServer:

  def run[F[_]: Async: Logger]: F[Nothing] = {
    for {
      client <- EmberClientBuilder.default[F].build
      inMemoryCache <- Resource.eval(Ref.of[F, Map[String, (Long, Weather)]](Map.empty))
      weatherCache = WeatherCache.make[F](inMemoryCache)
      weatherAPi = WeatherApi.make[F](client)
      weatherProgram = WeatherProgram.make[F](weatherCache, weatherAPi)

      httpApp = (
        WeatherRoutes[F](weatherProgram).routes
      ).orNotFound

      finalHttpApp = MiddleWareLogger.httpApp(true, true)(httpApp)

      _ <-
        EmberServerBuilder.default[F]
          .withHost(ipv4"0.0.0.0")
          .withPort(port"8080")
          .withHttpApp(finalHttpApp)
          .build
    } yield ()
  }.useForever
```

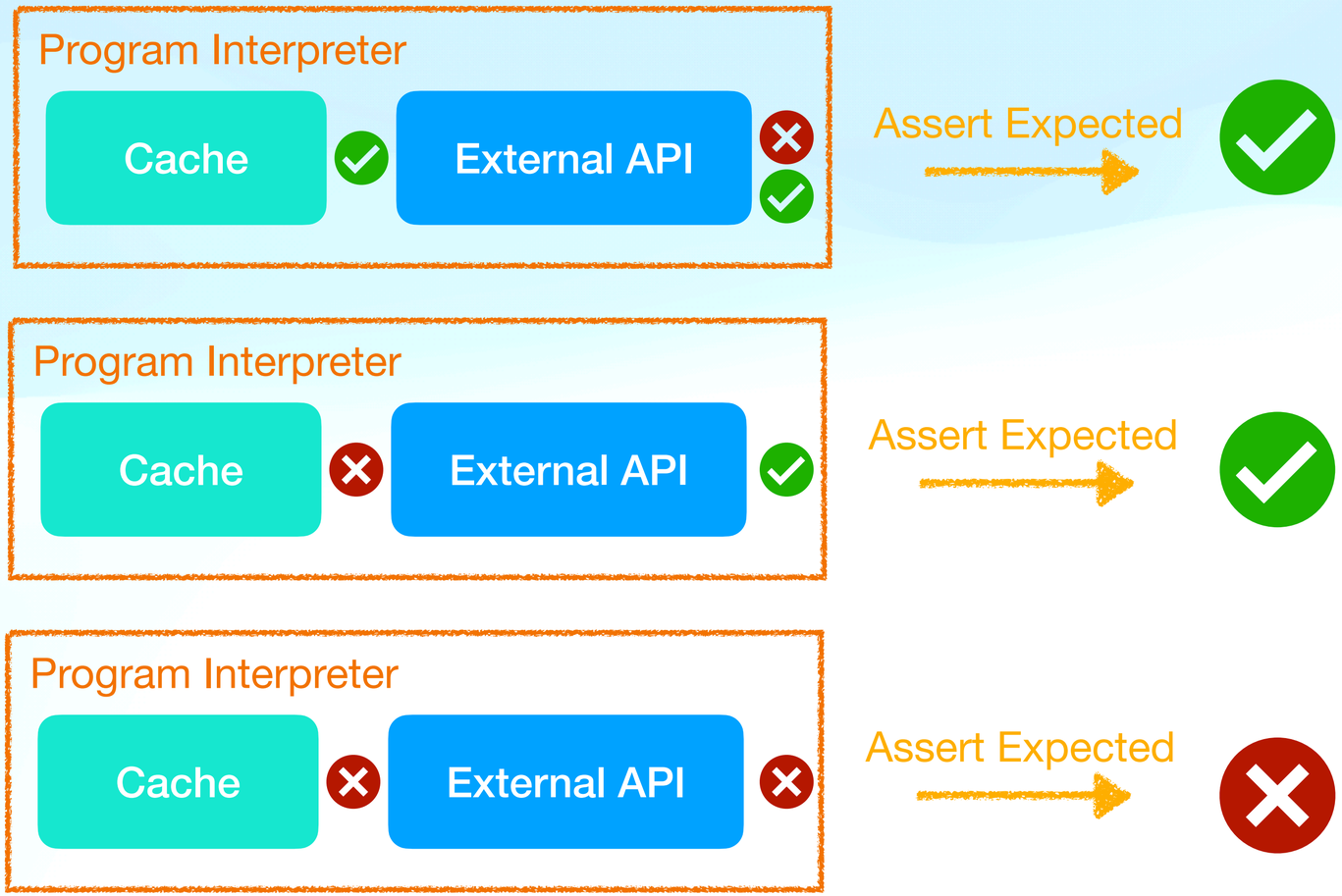
Wrapping Up

Summary



Wrapping Up

Summary



Wrapping Up

Takeaway

- Tagless Final in Scala is a design pattern about algebra, interpreters, and programs.
- Normally, an algebra has at least two interpreters: a production interpreter and a test interpreter.
- A program is a piece of code that uses algebras and interpreters to implement business logic.

Q and A