FUNCTIONAL PROGRAMMING & STATE MANAGEMENT

BEN ABRAMSON

Senior Principal Engineer – Stars Group

Functional Programming & State Management

Functional Programming is a stateless programming paradigm, but the World, along with many of the systems that model it, is not stateless. What does this mean for how we should use Functional Programming in the modern IT?

★ The history of Functional Programming

★ The consequences of State Management

★ What does good look like?

A LITTLE ABOUT ME

Who am I and where do I come from?

- ★ I am a Senior Principal Engineer in the Trading Department of the Flutter Group which includes Stars Group (Poker Stars & Bet Stars), Sky Bet, Paddy Power, Bet Fair and others
- ★ We do a lot of data processing, taking fixtures, opinion and game events and producing markets, selections & prices
- ★ This use case combines many applications of Functional Programming with complex State Management

MY BACKGROUND

Who am I and where do I come from?

★ The first programming language I learnt was LISP at University

★ After I left University, I became a Java Developer

★ I never heard anything more about Functional Programming for many years…

HISTORY

IN THE BEGINNING (FOR ME)

THE STARS GROUP

Learning LISP

- I was taught by Dr Phil Green at the University of Sheffield in 1996 (he's still there!)
- What did I learn?
  - Pure Functions – idempotency
  - No shared state
  - Immutability
- A Functional Program approximates to `y = f(x)`
- Functional Programming is awesome!

IN THE BEGINNING (FOR FP)

LISP is a great place to start the story of FP.

- ★ The first incarnations of LISP emerged in the mid-60s
- ★ It was great for data processing – very mathematical, highly recursive
- ★ Why was this important?
  - ★ The main use case – no GUIs, very little commercial IT
  - ★ Most of what people wanted to do with computers approximated to `f(x)`
  - ★ A program starts, it runs and then it stops
  - ★ The users were generally also the developers
  - ★ Very well suited to academia
- ★ Academia tends not to change very much

FUNCTIONAL PROGRAMMING ESTABLISHED

LISP became a standard in academia, IT takes off in the rest of the world

- ★ 1970s, commercial IT starts to take hold
- ★ 1980s, personal computers and IT as entertainment takes off
- ★ Very different use cases from what Functional Programming was being used for
  - ★ Computers were now being used for entertainment and business
  - ★ Holding and managing long living state has become important

CHANGING TIMES

By the 1990s, IT was very different from the 1960s. The Internet had come along.

- IT was no longer just about data processing, it was now about data sharing and data publishing
  - The ‘Information Super-Highway’ was here!
- Java was establishing itself as ‘the language of the Internet’
  - Platform independent
  - Open Source
  - Object-oriented – create objects to represent your state
- Functional programming & associated languages were being viewed as obsolete

THE NEW MILLENNIUM

IT is now about data management

- ★ The first decade of the new millennium is dominated by webapps and desktop apps
  - ★ IT systems are to enable people to manage data and data flows
  - ★ 3-tier architecture & Design Patterns rule the world
- ★ Just when it looked like Functional Programming might be consigned to the history books….
  - ★ Web 2.0
  - ★ Google's 2005 white paper on MapReduce

WHAT'S WEB 2.0 GOT TO DO WITH THIS?

Prior to Web 2.0, the internet was predominantly static content

- ★ Before 2.0, you needed techie skills to create content – people had HTML Home Pages
- ★ Web 2.0 allowed lay-people with few or no technical skills to create content
  - ★ Blogs
  - ★ Social Media
  - ★ Digital photography
  - ★ Smart phones
- ★ Massive increase in the amount of data being created
- ★ Big changes in how we use technology

BIG DATA

Google's white paper on MapReduce heralded the Big Data era

- ★ Fairly quickly, the technology existed to store & crunch data in volumes previously unimagined
  - ★ Distributed, multi-node systems
  - ★ Data storage and data processing could be parallelized
  - ★ (Careful of CAP theorem)
- ★ Many other tools followed
  - ★ Streaming tech, Spark, Storm, Flink
  - ★ Heavyweight messaging, Kafka
- ★ The emergence of JavaScript GUI frameworks separated a lot of the user interface complexity from the back-end systems
- ★ IoT meant we could now gather large amounts of data about our habits and our world

THE RETURN OF FUNCTIONAL PROGRAMMING

Now we have massive data sets and powerful tools to process them

- ★ Use cases that suited Functional Programming were suddenly everywhere
- ★ Back-end processing became more loosely coupled from the front end
- ★ Massive data sets meant massive processing overheads, not great for OO style
- ★ Very much suited to things Functional Programming does well:
  - ★ Recursion
  - ★ Immutable data
  - ★ No shared state – `y = f(x)`

STATE MANAGEMENT

WHAT IS STATE MANAGEMENT?

State Management is something that we probably don't think enough about

- ★ The World has a state, and it has events happening
- ★ The state of the World dictates what events happen, the events change the state
- ★ State is mutable – it changes
- ★ Some state is volatile and changes often and unpredictably, other state is pseudo-constant
- ★ Most of the complexity in your system will come from managing edge cases around changing state

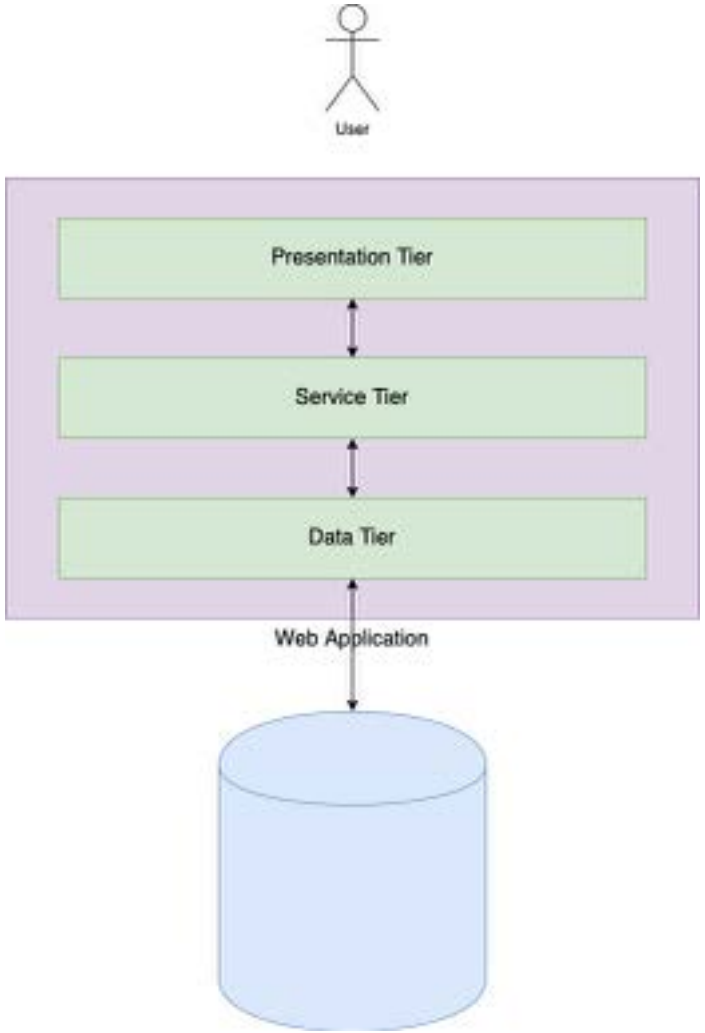
STATE MANAGEMENT & COMPLEXITY

Complexity in your system is likely to stem from issues around state management

- To state the obvious, reducing complexity is the most important principle of development
- Out of the Tar Pit (Moseley & Marks, 2006)
  - Essential Complexity: The complexity that cannot be avoided
  - Accidental Complexity: Avoidable complexity, often arising from bad state management
- Eradicating accidental complexity is the key to keeping them simple
- The best way to do that is to get your State Management right

WHY IS STATE MANAGEMENT IMPORTANT NOW?

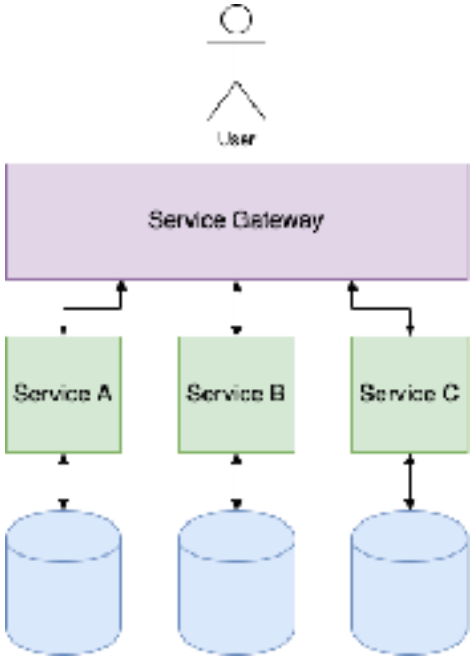
In older monolith WebApps, state management was easy – the database did it

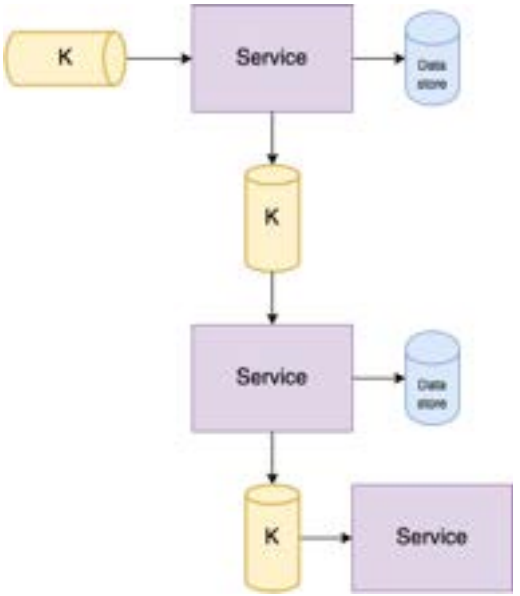


- ★ In a single component system, with a single relational data store, the state is handled by the DB
- ★ Any changes to persistent data is handled by the DB
- ★ Locking, transactions, isolation levels
- ★ Any transient state could be held in memory

STATE MANAGEMENT NOW

Modern systems have different challenges





- ★ Maintaining logical state across multiple services
- ★ Try to avoid distributed transactions

- ★ Where is the Golden Copy?
- ★ Recovering state
- ★ Performance

STATE MANAGEMENT PROBLEMS

With modern systems, state management needs to be explicitly designed

- ★ In multi-component, multi-datastore system, without planning, accidental complexity accrues
- ★ Remember, state is long living:
  - ★ State must be fault tolerant and able to recover from failure
  - ★ Very little of it is truly immutable
- ★ Classic design principles become very important:
  - ★ Single responsibility – avoid God-like state
  - ★ Clear integration points – interfaces and data resources must be well defined, good fences make good neighbours
  - ★ Avoid global state mutation

IMMUTABILITY

IMMUTABILITY

Immutability is where Functional Programming and state management can clash

- ★ In classic Functional Programming, data should be immutable
- ★ State is never completely immutable
- ★ You can’t handle state by pretending it doesn’t exist or it never changes
- ★ Very little data is truly immutable
- ★ There are 5 pieces of immutable data….

THE IMMUTABLES

$$e^{i\pi} + 1 = 0$$

c

IMMUTABILITY & SCOPE

OK, extreme example……

- ★ The immutability of data is defined by the scope of that data
- ★ For a specific, discrete calculation, quite a lot of data can be thought of as immutable
- ★ For the lifetime of the data resource, most of it is probably mutable
- ★ Data is probably more mutable than it looks
- ★ When creating data structures for functions, consider the Scope of Immutability

THE STARS GROUP

When modelling a process, consider the Scope of Immutability of your data.

Most of your state will be mutable during its lifetime, but is it going to mutate within the scope of the process you are modelling?

If it won't, then consider taking a Functional approach, but do so with the view that the state is going to change later on.

THE SCOPE OF IMMUTABILITY

Simple real-world example, how much of the identity information of a person is immutable:

```
public class Person {
        private UUID personId;
        private String firstName;
        private String lastName;
        private Date dateOfBirth;
        private String gender;
}
```

SCOPE OF IMMUTABILITY

Depending on the scope of our data process, we can think of two types of data

- Identity Data
  - Data that is fundamental to identifying what instance of data this is
  - IDs, Timestamps, Origin data
  - Immutable
- State Data
  - Data that represents the state of this instance
  - ‘Current’ values of state fields
  - Is expected to change, often on a regular basis
- What we define as being identity and state may change according to the scope of our processing
- This defines our Scope of Immutability

THE IMMUTABILITY ANTIPATTERN

Immutability is often abused

```
public MyObject doSomething(MyObject oldObject, MyValue2 newValue) {

        return MyObject.builder()
                        .myValue1(oldObject.getMyValue1())
                        .myValue2(newValue)
                        .myValue3(oldObject.getMyValue1())
                        .build()
}
```

This is not really Functional Programming!

This is a workaround to manage state while trying to be faithful to Functional principles

(Basically replacing thread safety with a race condition)

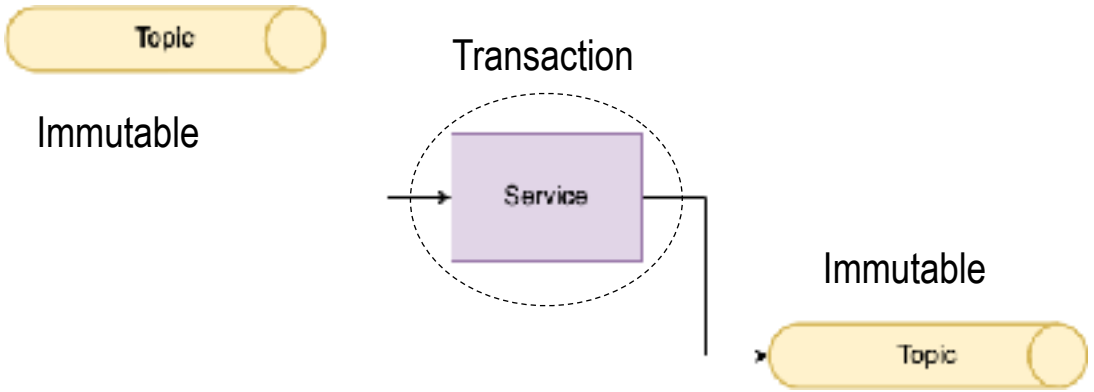
STATE & IMMUTABILITY

State is not and never will be immutable

- ★ If your state is immutable, your system is pointless
- ★ Do not pretend that your state is immutable and model it as such
  - ★ It will result in far more boilerplate code than the boilerplate supposedly saved by builder patterns in certain languages
  - ★ It's inefficient as new instances of the same object are constantly created taking up more memory than is necessary

DATA PIPELINE IMMUTABILITY

In data pipeline systems, immutability is often handled for us



IMMUTABILITY

In order to prevent immutability from hampering state management, consider the following:

- ★ Across the lifetime of a data resource, it's likely that a lot of the data is mutable
- ★ State is never completely immutable
- ★ In order to utilise immutability, consider the Scope of Immutability of your data within that process
- ★ Thinking about managing changing state through transactions is an alternative to immutability
- ★ Creating a data resource as immutable and then creating new instances of it every time it needs to mutate is not really proper Functional Programming

FUNCTIONAL PROGRAMMING IN MODERN DEVELOPMENT

WHAT DOES FP LOOK LIKE NOW?

THE STARS GROUP

Functional Programming has taken off, but no-one is writing LISP, right?

```
public void printObjects(List<MyObject> myStuff) {

    for (MyObject myo : myStuff) {
        if (myo.getThreshold() > 100) {
            System.out.println(myo.getName());
        }
    }
}
```

★ Old style imperative programming

```
public void printObjects(List<MyObject> myStuff) {

    myStuff.stream()
           .filter(myo -> myo.threshold > 100)
           .forEach(myo ->
                 System.out.println(myo.name));
}
```

★ Now I'm a functional programmer!

DON’T BELIEVE THE HYPE!

THE STARS GROUP

A streaming API does not make a Functional Program

(Quite good for creating mudballs though!)

WHAT DOES MODERN FP LOOK LIKE?

THE STARS GROUP

So it's not just a streaming API…..

★ It's not immutable data structures

★ It's not inextensible classes

★ It's not Kotlin (or Scala, or Erlang, or Clojure) – Functional Programming is not a language choice

Accept the Functional Epiphany…..

All code is Functional!

WHAT??

Huh??

- ★ Everything you've ever written has a functional context – which is usually its primary context
- ★ Think about how you test your code
- ★ All unit tests mock out external inputs and therefore expect the same output, no matter what, every time
- ★ This is how functional programming behaves
- ★ (Maybe all languages are essentially a LISP derivative!)

SO WHAT DOES MODERN FP LOOK LIKE?

THE STARS GROUP

When we look at modern codebases, what signs of Functional Programming do we see?

- ★ Most developers are trying to write in the functional style:
  - ★ Immutable data structures
  - ★ Use of streaming APIs
  - ★ A lot of final classes
- ★ Data crunching and data mapping tends to be a little easier
- ★ Often at the cost of state management
- ★ Practical, effective functional programming in an enterprise environment is not very well defined

WHAT DOES GOOD LOOK LIKE?

WHAT DOES GOOD LOOK LIKE?

The over-arching principles of Functional Programming seem a little idealistic for modern development

- ★ When Functional Programming came along, most systems approximated to `f(x)`
  - ★ There was no outside state once programs started with given inputs
  - ★ There was little or no user-interaction with a running program
  - ★ This makes sticking to Functional Programming principles very easy
- ★ Some applications can function in this way
- ★ Many cannot – they rely on either outside inputs or long living state

STATE MANAGEMENT

The key to using Functional Programming successfully in an Enterprise environment comes down to State Management. Consider the following:

- ★ Durability
- ★ Performance
- ★ Persistence
- ★ Representation

DURABILITY

Do not try to handle state by ignoring it

- ★ Many developers will try to implement a system functionally by stripping out all traditional elements of state management
- ★ Make everything immutable – just create new ones when state changes
- ★ Downside is it introduces a lot of real time dependencies and data to feed in
- ★ Caching of data then tends to get introduced, which works until there is a failure
- ★ Long term state cannot be dealt with in a transient manner

PERFORMANCE

Be aware of the importance of Real Time

- ★ Trying to infer state on the fly every time it is required will result in performance issues
- ★ Most notable with Kafka – “Just read the topic from the start”
- ★ Depending on your use case that can introduce performance issues
- ★ Trying to pass full state around everywhere also becomes problematic – it has a high overhead
- ★ A Golden Copy of State must always be available in real time

PERSISTENCE

State is long-lived, often over multiple inputs. Functions generally are not.

- ★ Long living state needs somewhere to persist – this is necessary to solve performance & durability issues
- ★ It must be available in real time
- ★ It's usually mutable, ensure that it is not globally mutable – global state mutation is bad
- ★ Databases solve this problem very well
- ★ Take care with flat, nosql style data!

REPRESENTATION

What does your state look like?

- ★ In any system that is does not approximate to `f(x)` your state is mutable
- ★ (So don't make it immutable)
- ★ From a good practice point of view, also consider the following:
  - ★ Is it hierarchical?
  - ★ What is the Scope of Immutability?
  - ★ How is it going to be used in processing?

GOOD DESIGN

THE STARS GROUP

When designing a system, consider State Management first, it's the core of your system

- ★ Think about durability, performance, persistence and representation
- ★ Where's your Golden Copy of State? It must be available in real time
- ★ Once the State Management is sorted out, then consider the functional aspects
- ★ Consider the Scope of Immutability for each process that needs to be modelled
- ★ When it comes to modelling the actual functional computations, Functional Programming is the most powerful paradigm

GOOD DESIGN

THE STARS GROUP

Functional or Not?

- ★ Functions are very good at data crunching – it is what they are best at
- ★ Modelling process flows, workflows and state management are not quite the same thing
  - ★ It is possible to apply Functional Programming to these parts of a system
  - ★ The principles of Functional Programming may need to be subtly manipulated to do this
- ★ OO/Imperative programming is quite good at modelling flows and doing state management
  - ★ Objects are good for modelling state
  - ★ Open for extension & mutability make it easier to model flows & state
- ★ If you mix & match, no-one's going to arrest you!
- ★ Using an OO style to model the wider behavior, mixed with functions for data processing is a very powerful combination
- ★ You've probably already done this

LESSONS LEARNED

What have we learned?

- ★ Functional Programming is better for pure data processing, however, most modern systems are not pure functions, and you may fall foul of state management problems
- ★ Your state is going to mutate – just accept it and work with it
- ★ You will always need a Golden Copy of State, available in real time
- ★ When thinking about Functional use cases, always consider the Scope of Immutability

SUMMARY

THE STARS GROUP

What to take away

- ★ Being a technology purist might be great in a textbook or classroom, but it often comes undone in the reality of Enterprise Systems development
- ★ Technology is only as good as how you use it, so make sure you know the use case
- ★ Don’t think about a choice between FP and OO – that’s a simplification, there’s room for both
- ★ I don’t consider myself a Functional Programmer, I’m a developer who writes Functions

QUESTIONS

THANK YOU