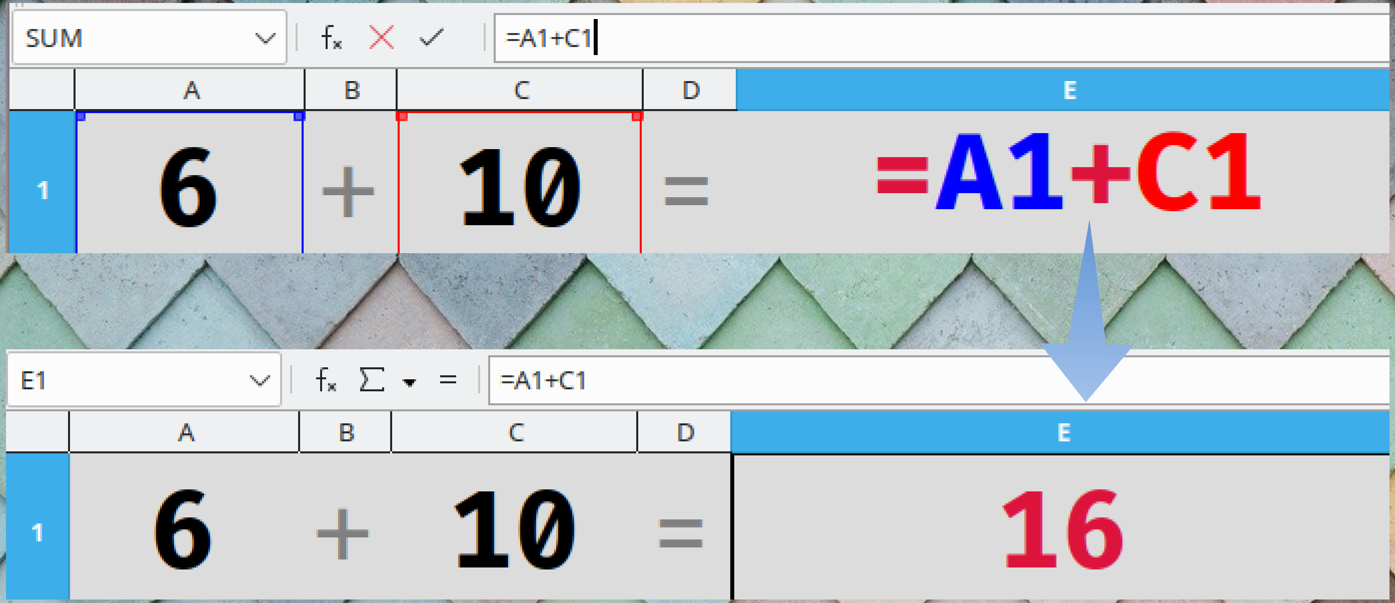
Reactive programming



	A	B	C	D	E
1	6	+	10	=	=A1+C1

	A	B	C	D	E
1	6	+	10	=	16

Imperative programming

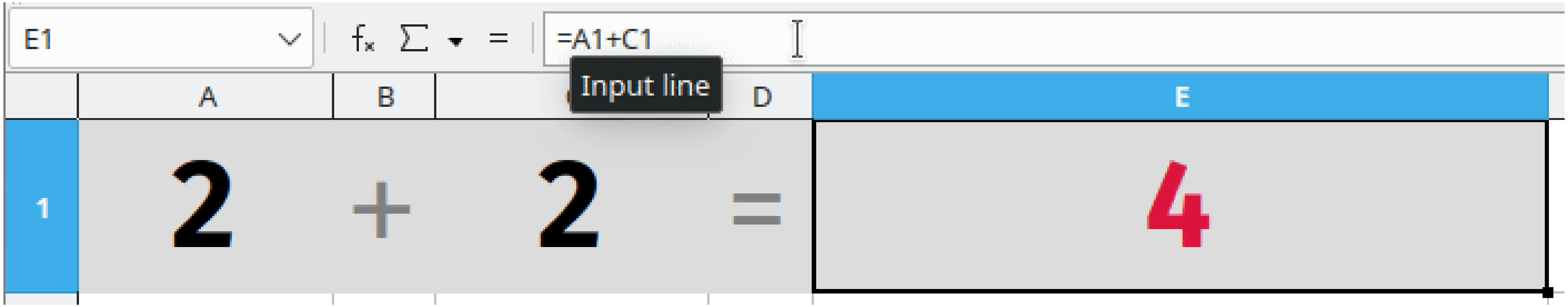
```
int x = 10;
int y = 20;
int a = x + y; //a = 30
x++; //a = 30
```

Reactive Programming

A program that :

- is event-based
- acts in response to input
- is viewed as a flow of data

Spreadsheets as an example

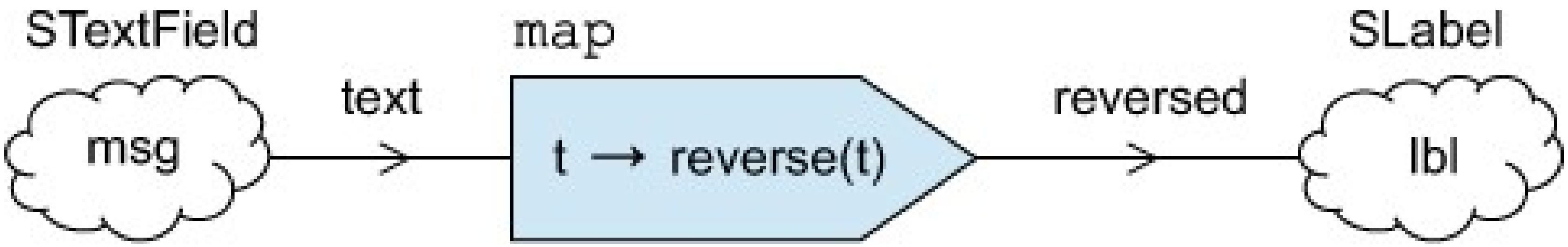


E1

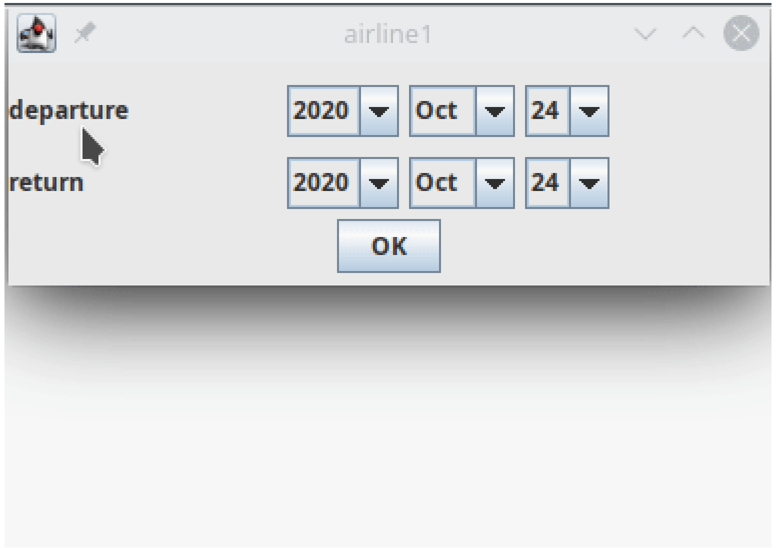
=A1+C1

	A	B	C	D	E
1	2	+	2	=	4

Example 1



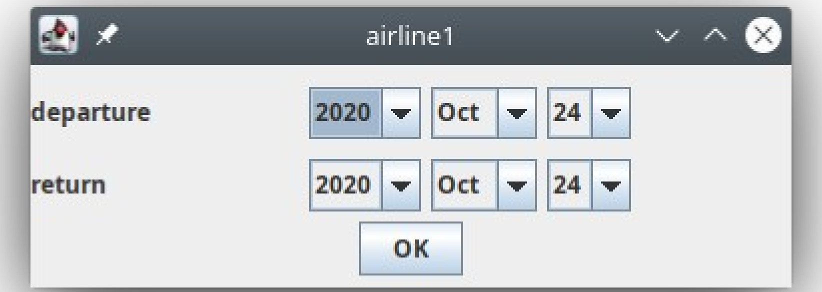
Example 2



departure 2020 Oct 24

return 2020 Oct 24

Example 2



airline1

departure 2020 Oct 24

return 2020 Oct 24

```
SDateField dep = new SDateField();
SDateField ret = new SDateField();

Cell<Boolean> valid = dep.date.lift(ret.date,
    (d, r) → d.compareTo(r) ⩽ 0);

SButton ok = new SButton("OK", valid);
```

Definitions

- **Cell** : represent a value that changes over time
- **Streams** : represent a stream of events

github.com/SodiumFRP

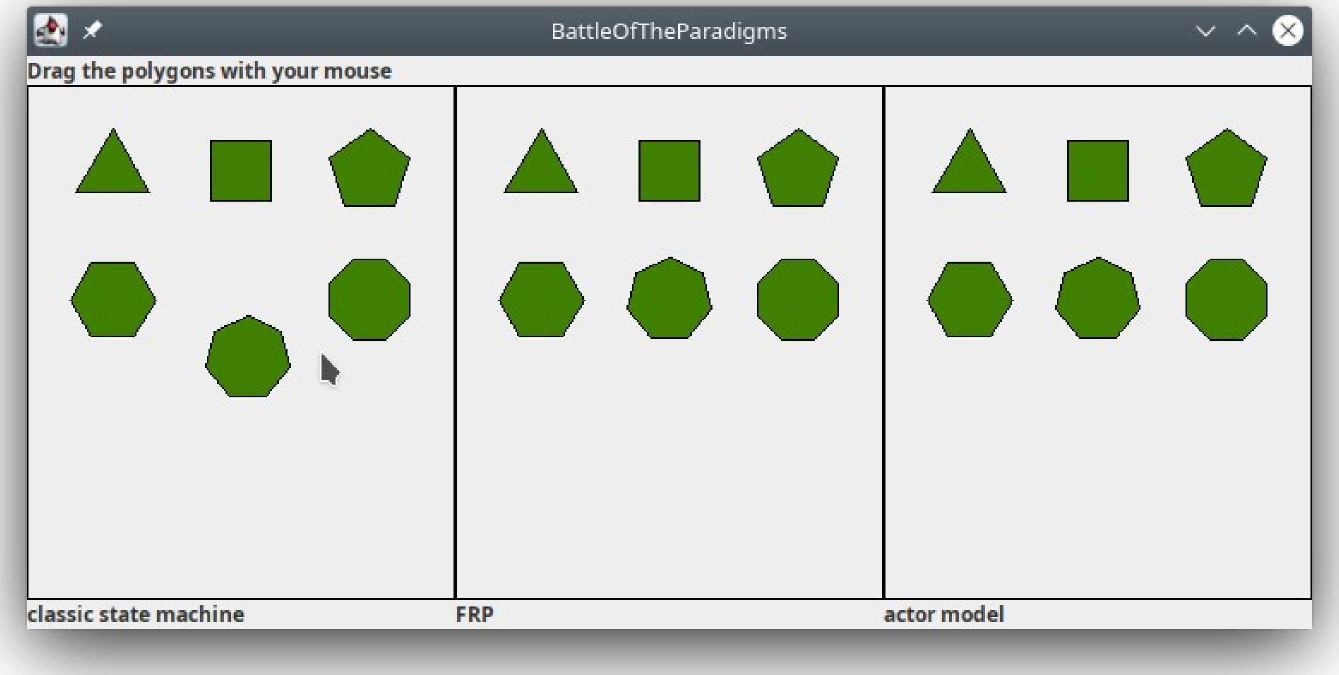
⚠ *Every reactive system has its own definitions*

```java
CellSink<Integer> a = new CellSink<>(1);

Cell<Integer> a3 = a.map(x -> x * 3);
Cell<Integer> a5 = a.map(y -> y * 5);
Cell<String> b = a3.lift(a5, (x, y) -> x + " " + y);

List<String> out = new ArrayList<>();
Listener l = b.listen(out::add);

a.send(2);
a.send(5);

l.unlisten();
assertEquals(Arrays.asList("3 5", "6 10", "15 25"), out);
```

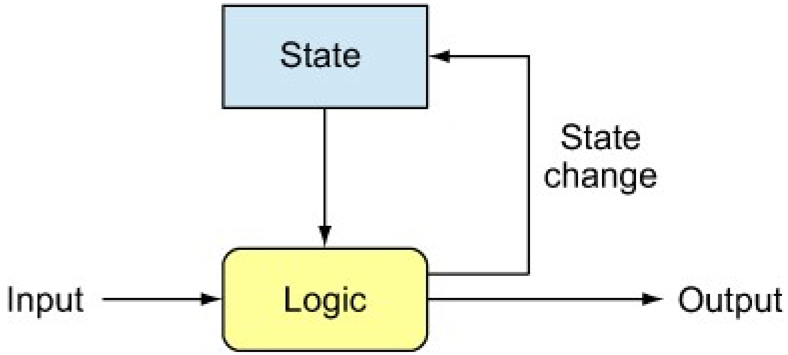
```java
StreamSink<Character> e = new StreamSink<>();

List<Integer> out = new ArrayList<>();

Listener l = e.filter(Character::isUpperCase)
        .map(Integer::valueOf)
        .listen(out::add);

e.send('H');
e.send('o');
e.send('I');
l.unlisten();
assertEquals(Arrays.asList(17/*H*/,18/*I*/), out);
```

Different Models

- Classic state machine
- FRP (functional reactive programming)
- Actor



Classic state machine



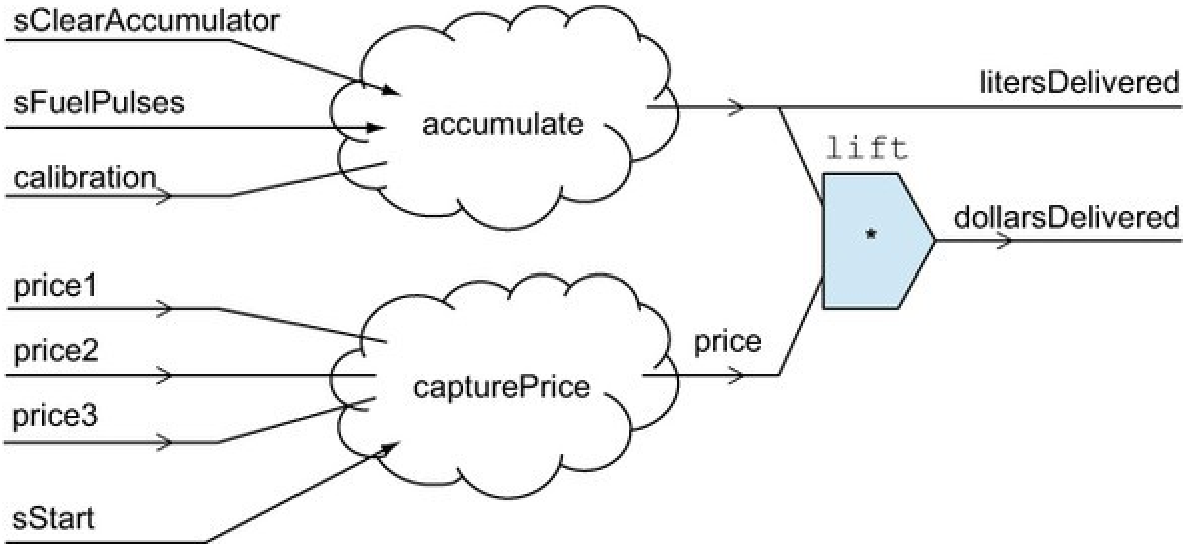
Classic state machine

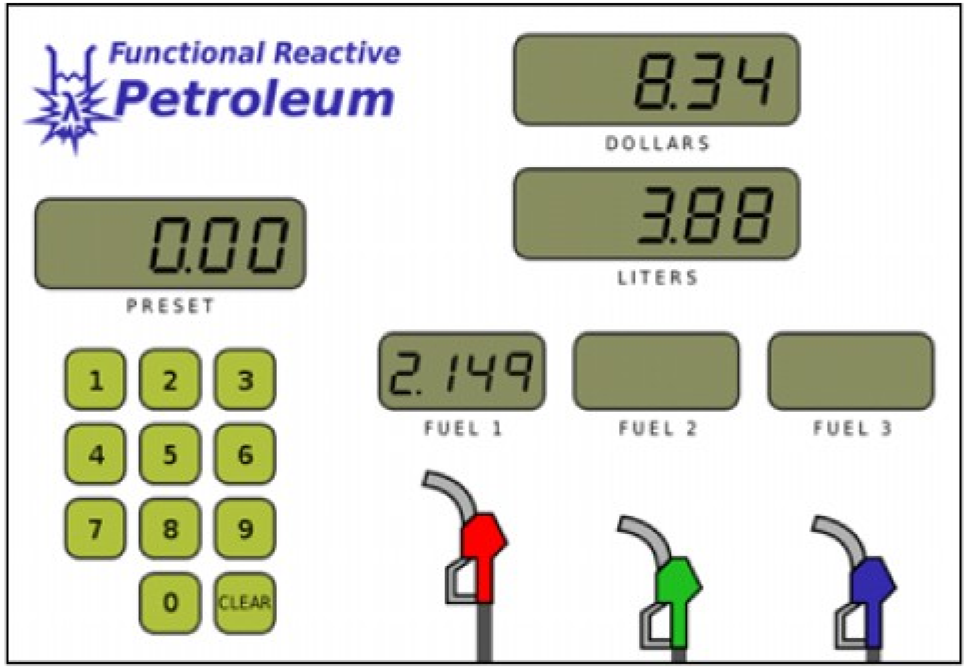
- Listeners/Callbacks → Observer design pattern
- Bug prone
  - Unpredictable order
  - Missed first event
  - Messy state
  - Threading issues
  - Leaking callbacks

FRP

- event propagation with functional programming
- It’s a composable, modular way to code event-driven logic
- complete embedded language for stateful logic
- Thinking in terms of dependency rather than sequences

FRP

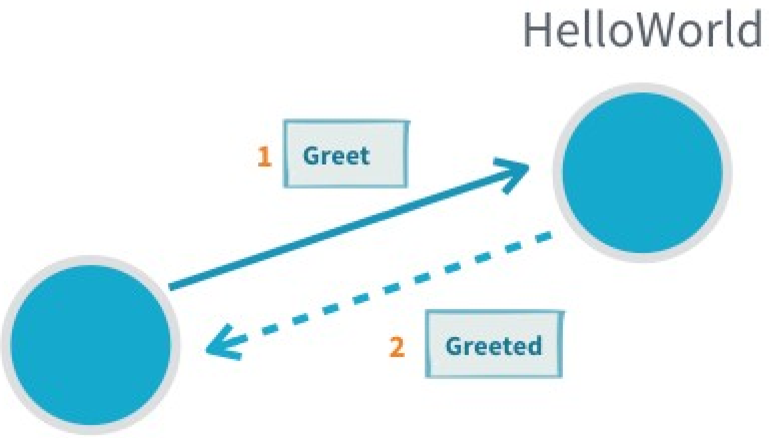

sClearAccumulator
sFuelPulses
calibration
accumulate
litersDelivered
lift
*
dollarsDelivered
price1
price2
price3
capturePrice
price
sStart


Functional Reactive
Petroleum
8.34
DOLLARS
3.88
LITERS
0.00
PRESET
1
2
3
4
5
6
7
8
9
0
CLEAR
2.149
FUEL 1
FUEL 2
FUEL 3

Actor

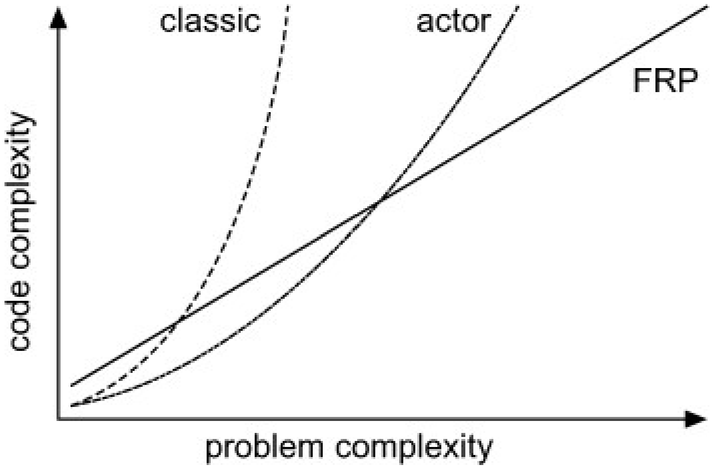
- An actor is a process whose job is to handle incoming messages from a single asynchronous input queue
- Each actor has a public address, and other actors that know the address can send the actor messages
- Actors commonly use a reply mechanism that sends a message to the originator of an input message
- Actors as they’re commonly implemented have a thread-like flow of control

Actor



```
object HelloWorld {
  final case class Greet(whom: String, replyTo:ActorRef[Greeted])
  final case class Greeted(whom: String, from: ActorRef[Greet])

  def apply(): Behavior[Greet] = Behaviors.receive
{ (context, message) ⇒
    context.log.info("Hello {}!", message.whom)
    //#hello-world-actor
    println(s"Hello ${message.whom}!")
    message.replyTo ! Greeted(message.whom, context.self)
    Behaviors.same
  }
}
```

Different Models



Different Architectures

Most applications are architected around one of two programming models, or a mix of the two:

- Threads
    - There are two types:
    - Non-Blocking – asynchronous execution is supported and is allowed to unsubscribe at any point in the event stream.
    - Blocking
- Events

Reactive system

Sometimes the term reactive programming refers to the architectural level of software engineering, where individual nodes in the data flow graph are ordinary programs that communicate with each other.

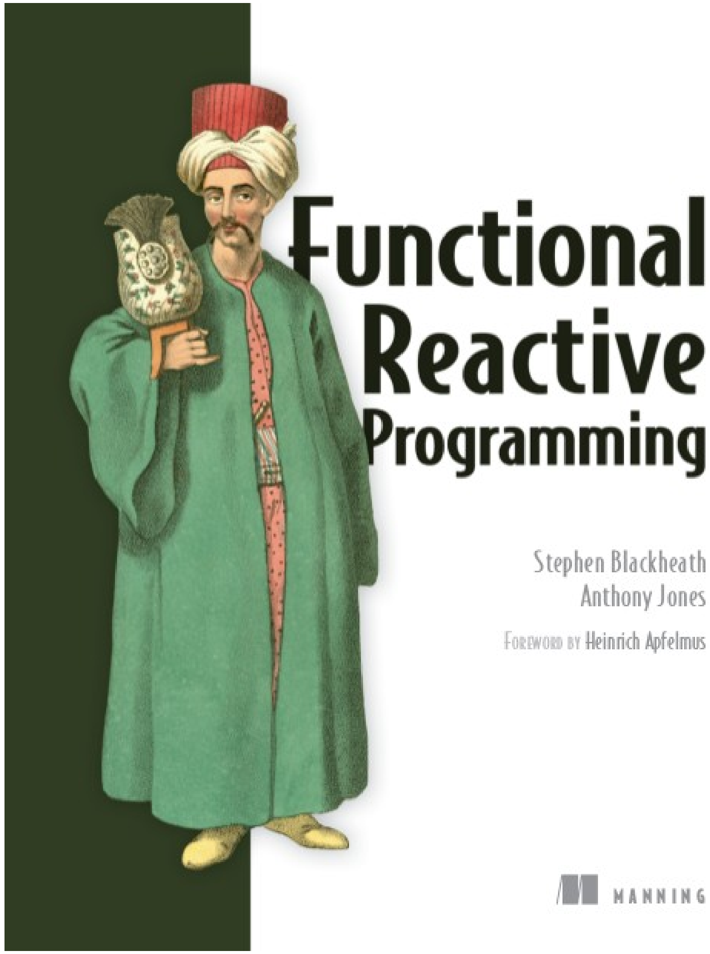
Some libraries

- ReactiveX – Microsoft (.NET)
  - Supports most of the languages : RxJava, RxJs , ...
- Project Reactor – Pivotal (Java)
- Sodium FPR (supports many languages)
- Lisp Cells

Reactive Stream

- It provides a standard for asynchronous stream processing with non-blocking backpressure
  - Reactive Streams in Java 9
  - Akka Streams
  - Ratpack
  - Vert.x

References


Functional Reactive Programming
Stephen Blackheath
Anthony Jones
Foreword by Heinrich Apfelmus
MANNING

- https://theartofservice.com/reactive-programming.html
- https://www.baeldung.com/java-reactive-systems