



Introduction to Reactive

Daniel Hinojosa

 @dhinojosa

 dhinojosa@evolutionnext.com

**Slides and code is
available at:**



https://github.com/dhinojosa/intro_to_reactive

What is **reactive**?

Streamlining Synergy Win-Win
Paradigm Best of Breed Turnkey
Touch Base Best Practices Cloud
Service Oriented Architecture Level
Latcom Mission Critical ROI Push
Holistic Approach Eating our own dog
Food Horizon Push the Envelope

Reactive?

What is **reactive**?

**What is reactive
programming?**

functional



What is **reactive**
programming?

According to...



Functional reactive programming (FRP) is a programming paradigm for reactive programming (asynchronous dataflow programming) using the building blocks of functional programming (e.g. map, reduce, filter). FRP has been used for programming graphical user interfaces (GUIs), robotics, and music, aiming to simplify these problems by explicitly modeling time.

According to...



In computing, **reactive programming** is a **programming paradigm** oriented around **data flows** and the propagation of change. This means that it should be possible to express static or dynamic **data flows** with ease in the **programming languages** used, and that the underlying execution model will automatically propagate changes through the data flow.



In the year, 2051, anthropologists discovered...

```
var a = 10;  
  
var b = a + 1;  
  
a = 11;  
  
b = a + 1;
```

<http://paulstovell.com/blog/reactive-programming>

Example of P#

```
var a = 10;  
  
var b <= a + 1; // <= Destiny Operator  
  
a = 20;  
  
Assert.AreEqual(21, b);
```

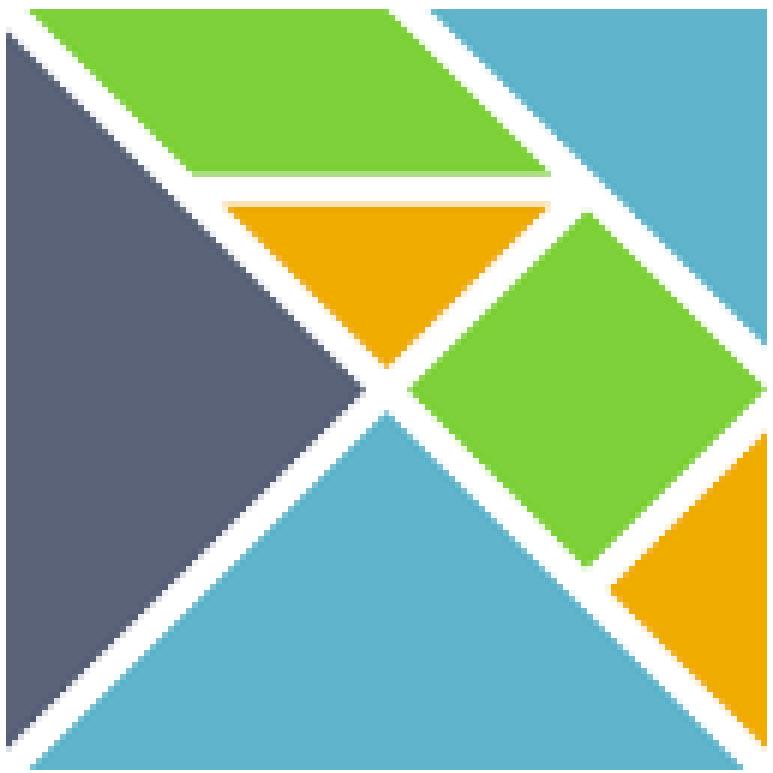
	A	B
1	Invoices	
2	ABC Corp	\$3,000.00
3	XYZ Tech	\$1,500.00
4	Right on Inc.	\$2,300.00
5	Total	\$6,800.00

	A	B
1	Invoices	
2	ABC Corp	\$3,000.00
3	XYZ Tech	\$1,500.00
4	Right on Inc.	\$2,300.00
5	Total	\$6,800.00



= SUM(B2:B4)

**Is there a language
like P#, today?**



**Can we do this FRP
on the JVM... easily?**

A photograph of a forest scene. In the foreground, there is a dirt path or clearing covered in fallen leaves. The path leads towards a dense stand of tall, thin trees in the background. The surrounding area is filled with green foliage and bushes.

**Not in a natural
way....**

**What technologies
can we use on the
JVM... for “Reactive”?**



ReactiveX

An API for asynchronous programming
with observable streams

Choose your platform





Actors and Resiliency





Reactive Extensions for Async Programming



RX
for Async Programming

RX is FRP on the JVM



RX is FRP **in the JVM**

"There's a whole lot debate to what we should be calling this. We started using Functional Reactive to name it before any of us had any clue about the academia behind it...if anyone can give me a better name for what we are doing...I'd like to hear it"

- Ben Christensen @ Goto Conference

https://youtu.be/_t06LRX0DV0?t=46

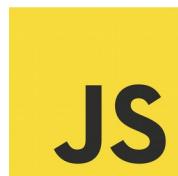


A photograph of a tropical forest stream. The water flows over several large, mossy rocks. Sunlight filters through the dense canopy of green leaves and branches above, creating bright highlights on the water and the surrounding foliage. The overall atmosphere is serene and natural.

Reactive Streams



Reactive Extensions for Async Programming



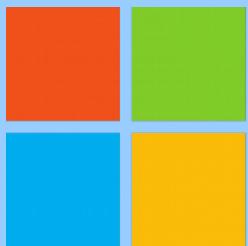


RX for Async Programming





formerly of....



Microsoft

“Mouse is a database”

PUSH

Observer Iterator



What pains does Rx alleviate?



Callback Hell



Retaining a Source



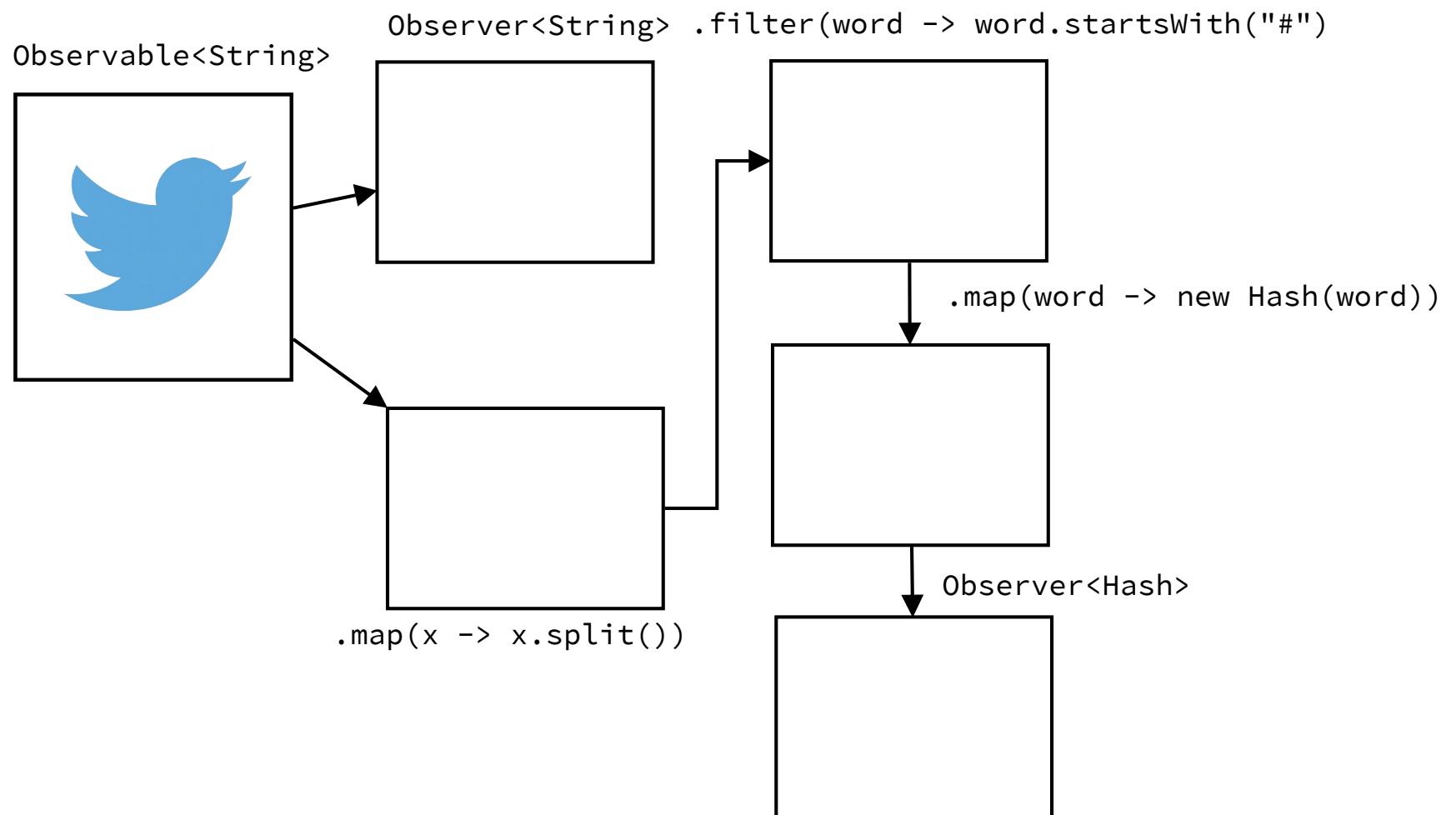
Backpressure Remedies

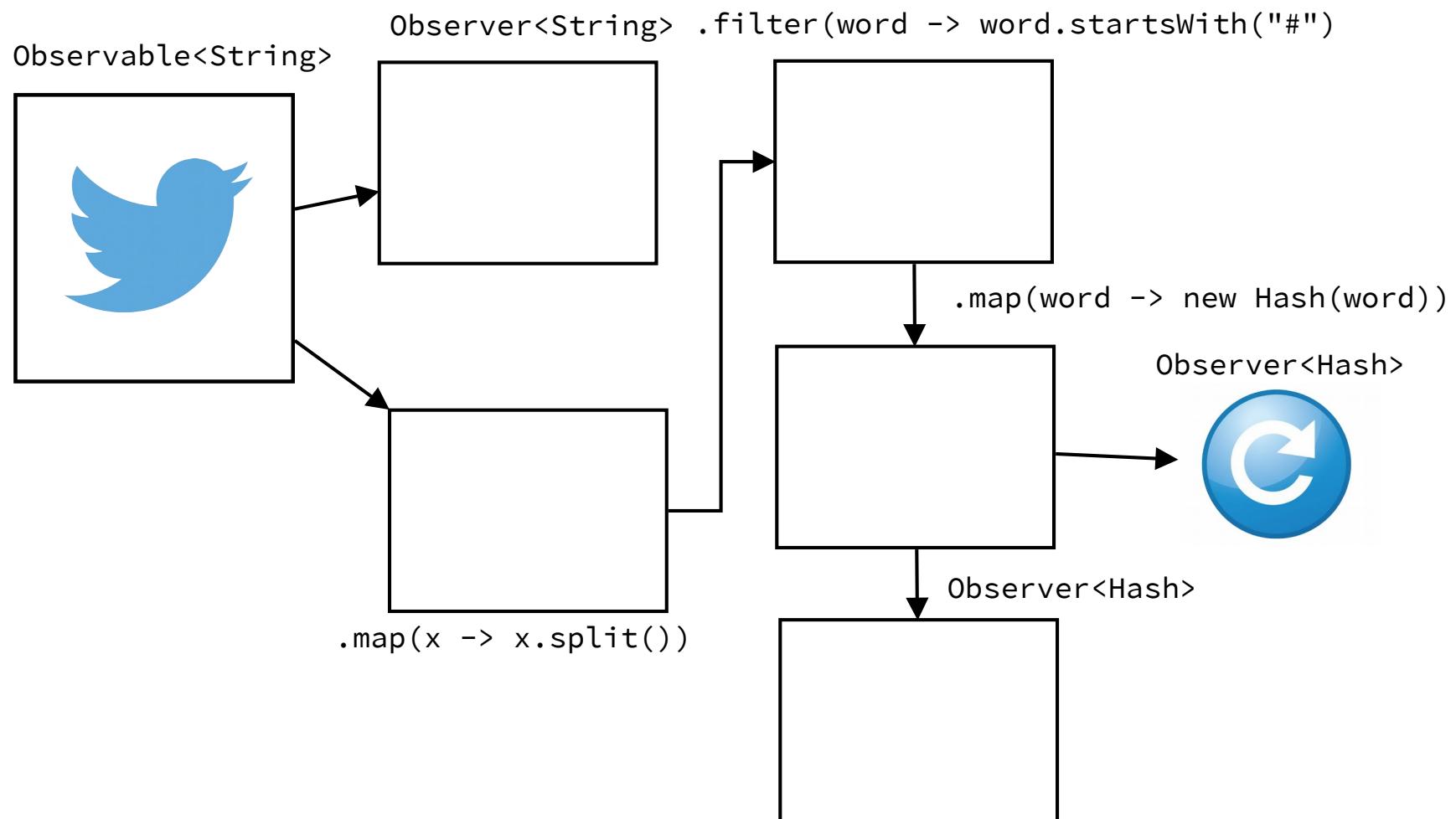


	One	Many
Sync	T	Iterator<T>
Async	Future<T>	Observable<T>



Future Demo





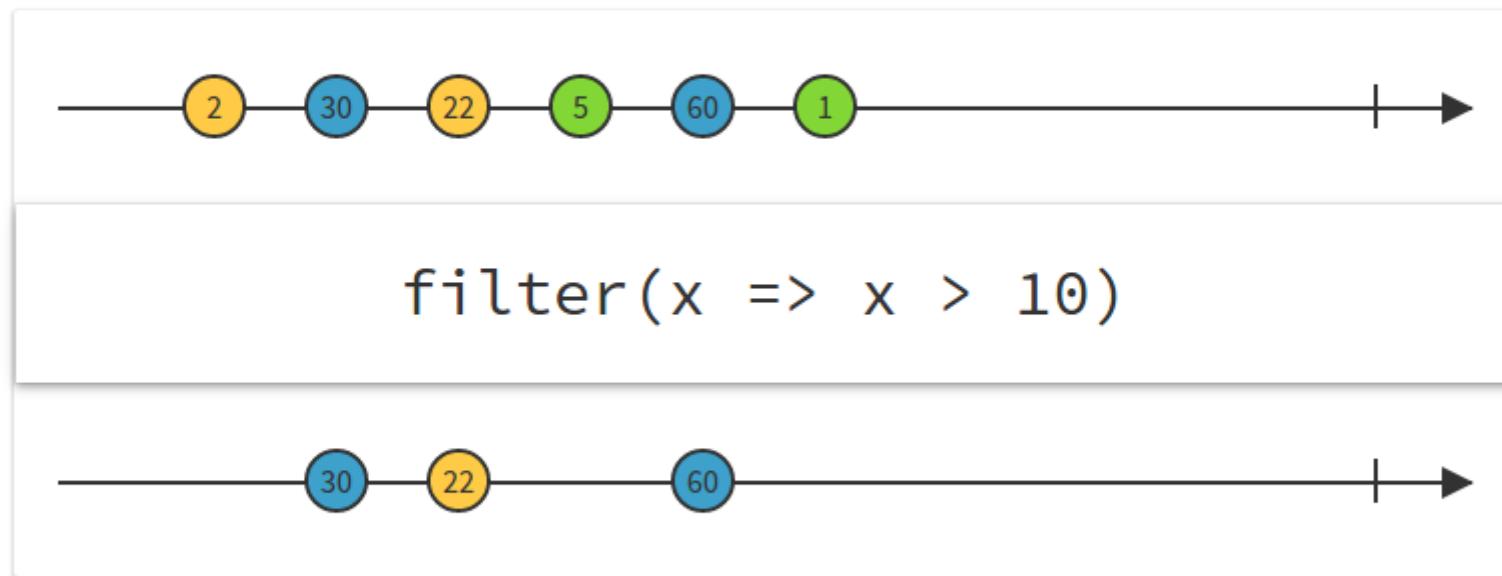


Thinking of Observables as Pipes

Marble Diagrams

Filter

emit only those items from an Observable that pass a predicate test



The `Filter` operator filters an Observable by only allowing items through that pass a test that you specify in the form of a predicate function.



Find all your favorite operators at:

<http://reactivex.io/documentation/operators>



RXJava Observable Demo



Hot vs. Cold Observable



벌 떨게 한 시베리아 도강 도전기 大 공개! ◆ **월요일마다 〈김병만의 정글의 법칙〉** 오늘 오후 5시 방송

PSY- Gangnam Style (Official Music Video)



DanceGangnamStyle

[Subscribe](#) 46,824

44,983,176

Add to Share More

88,792 23,430



DICK
Pritchett
REAL ESTATE, INC.

Southwest Florida Eagle Cam



Southwest Florida Eagle Cam

[Subscribe](#) 6,394

2592 watching now

Add to Share More

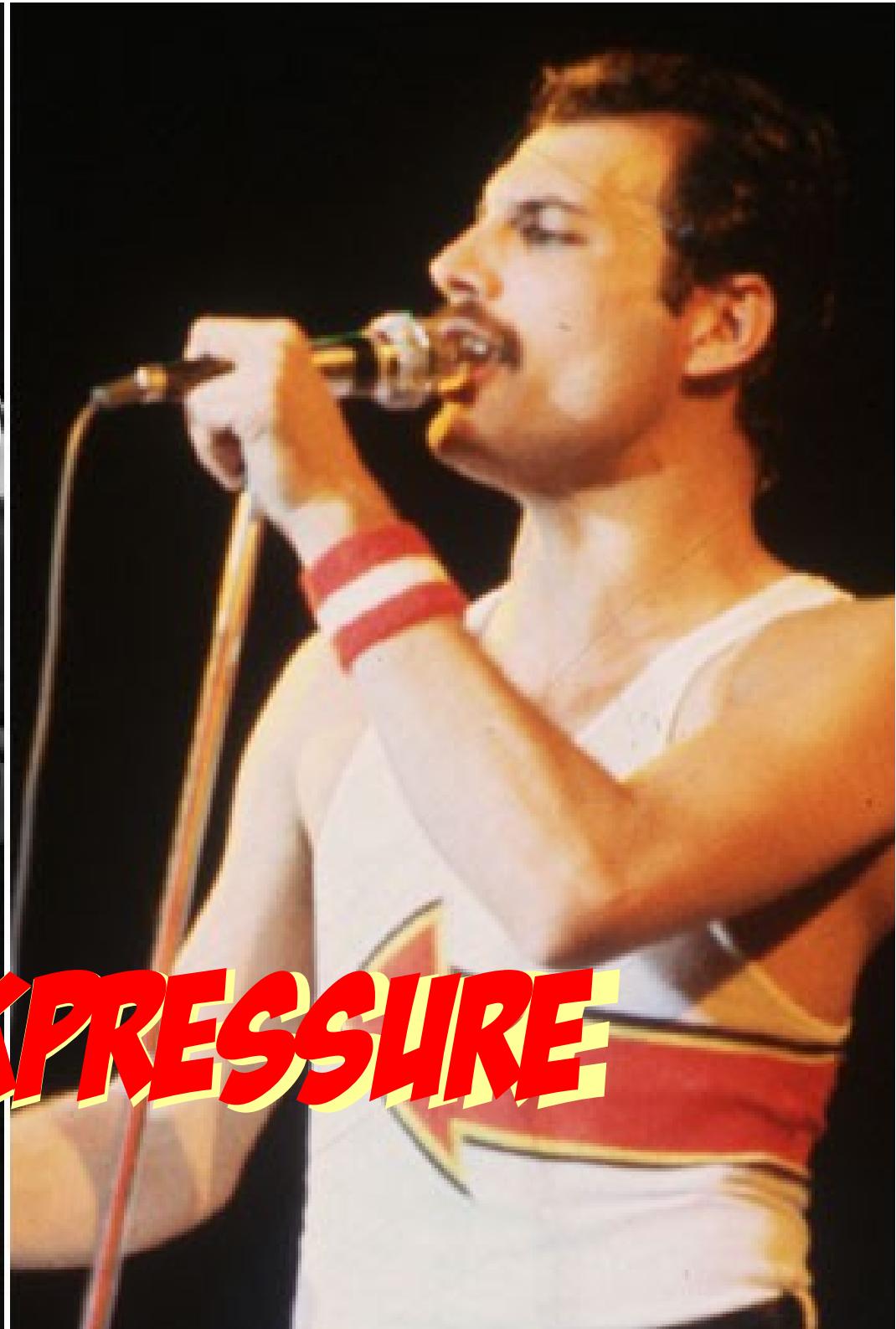
4,599 392



RXJava Hot/Cold Demo



UNDER BACKPRESSURE



Backpressure *def*:

Back pressure refers to pressure opposed to the desired flow of a fluid in a confined place such as a pipe. It is often caused by obstructions or tight bends in the confinement vessel along which it is moving, such as piping or air vents.

Backpressure *def*:

Back pressure is when an Observable is emitting items more rapidly than an operator or subscriber can consume them. This presents the problem of what to do with such a growing backlog of unconsumed items.



RXJava Backpressure Demo



JavaTM 8

Streams?

Reusing Streams



31

I'd like to duplicate a Java 8 stream so that I can deal with it twice. I can `collect` as a list and get new streams from that;



2

```
// doSomething() returns a stream
List<A> thing = doSomething().collect(toList());
thing.stream()... // do stuff
thing.stream()... // do other stuff
```

but I kind of think there should be a more efficient/elegant way. **Is there a way to copy the stream without turning it into a collection?**

Reusing Streams



38

I think your assumption about efficiency is kind of backwards. You get this huge efficiency payback if you're only going to use the data once, because you don't have to store it, and streams give you powerful "loop fusion" optimizations that let you flow the whole data efficiently through the pipeline.



If you want to re-use the same data, then by definition you either have to generate it twice (deterministically) or store it. If it already happens to be in a collection, great; then iterating it twice is cheap.

We did experiment in the design with "forked streams". What we found was that supporting this had real costs; it burdened the common case (use once) at the expense of the uncommon case. The big problem was dealing with "what happens when the two pipelines don't consume data at the same rate." Now you're back to buffering anyway. This was a feature that clearly didn't carry its weight.

If you want to operate on the same data repeatedly, either store it, or structure your operations as Consumers and do the following:

```
stream()...stuff....forEach(e -> { consumerA(e); consumerB(e); });
```

You might also look into the RxJava library, as its processing model lends itself better to this kind of "stream forking".

[share](#) [edit](#) [flag](#)

edited May 26 '14 at 4:44

answered May 26 '14 at 0:00

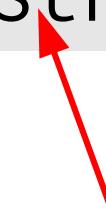


Brian Goetz

25.5k ● 7 ● 53 ● 71

	One	Many
Sync	T	Iterator<T>
Async	Future<T>	Observable<T>

	One	Many
Sync	T	Iterator<T> Stream<T>
Async	Future<T>	Observable<T> ParallelStream<T>



One Time Process

	One	Many
Sync	T	Iterator<T> Stream<T>
Async	Future<T>	Observable<T> ParallelStream<T>

Multiuse Process

One Time Process



	One	Many
Sync	T	Iterator<T> Stream<T>
Async	Future<T> CompletableFuture<T>	Observable<T> ParallelStream<T>



More messy
overloaded
"Reactive"
Terms

The Reactive Manifesto

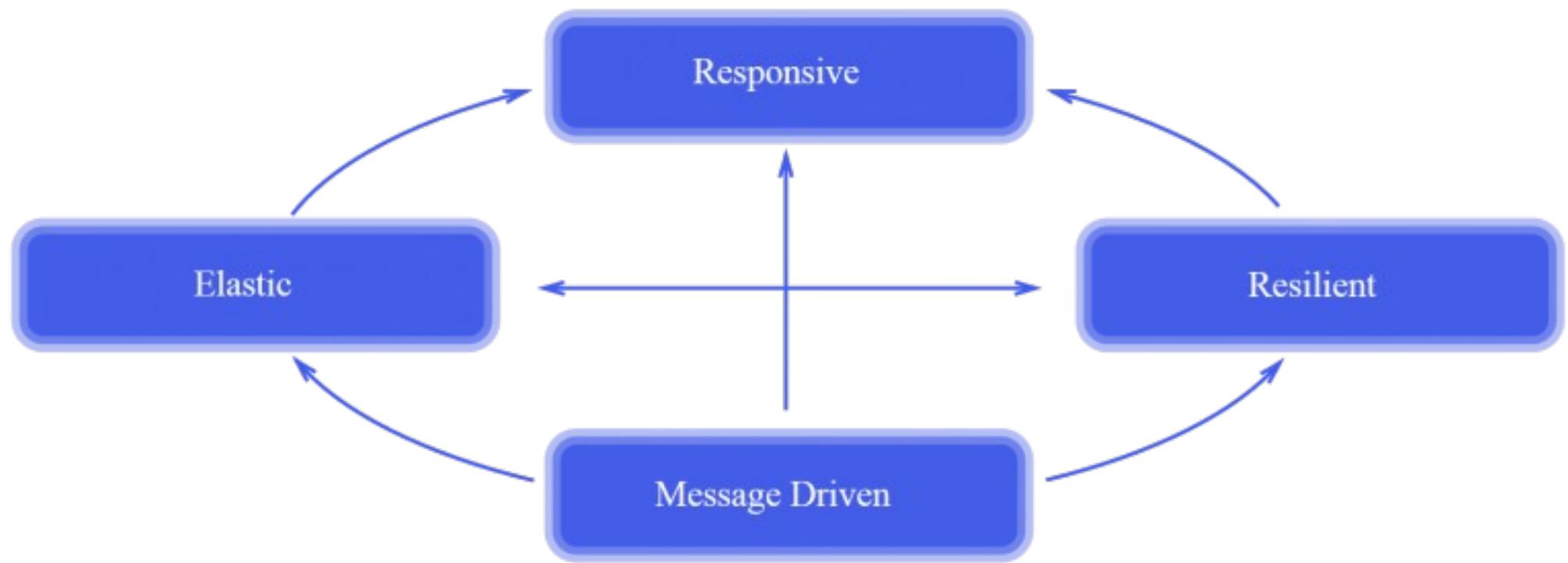
Published on September 16 2014. (v2.0)

Organisations working in disparate domains are independently discovering patterns for building software that look the same. These systems are more robust, more resilient, more flexible and better positioned to meet modern demands.

These changes are happening because application requirements have changed dramatically in recent years. Only a few years ago a large application had tens of servers, seconds of response time, hours of offline maintenance and gigabytes of data. Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and 100% uptime. Data is measured in Petabytes. Today's demands are simply not met by yesterday's software architectures.

We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems.

Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.







"Swedified" from Áhkká

<http://akka.io>

**Actors, Location
Transparency, Finite
State Machines,
Persistence,
Clustering, Agents**

Failure Tolerance

Scaling up and out

About **Actors**

ER LANG

```
0,14,0> [number_analysis:number_analysis/0
? analyse([1])
! analysis_result(get_more_digits)
<0,14,0> [number_analysis:number_analysis
analysis/1
0,14,0> [number_analysis:number_analysis/0
? analyse([1,6])
! analysis_result(get_more_digits)
<0,14,0> [number_analysis:number_analysis
analysis/1
0,14,0> [number_analysis:number_analysis/0
? analyse([1,6,7])  I
! analysis_result(port(67))
<0,14,0> [number_analysis:number_analysis
analysis/1
switch:switch_group/1]: exit(normal) in
```



Erlang the Movie!

<https://www.youtube.com/watch?v=xrIjfIjssLE>

```
0,14,0> [number_analysis:number_analysis/0
? analyse([1])
! analysis_result(get_more_digits)
<0,14,0> [number_analysis:number_analysis
analysis/1
0,14,0> [number_analysis:number_analysis/0
? analyse([1,6])
! analysis_result(get_more_digits)
<0,14,0> [number_analysis:number_analysis
analysis/1
0,14,0> [number_analysis:number_analysis/0
? analyse([1,6,7])  I
! analysis_result(port(67))
<0,14,0> [number_analysis:number_analysis
analysis/1
switch:switch_group/1]: exit(normal) in
```



Erlang the Movie!

<https://www.youtube.com/watch?v=xrIjfIjssLE>

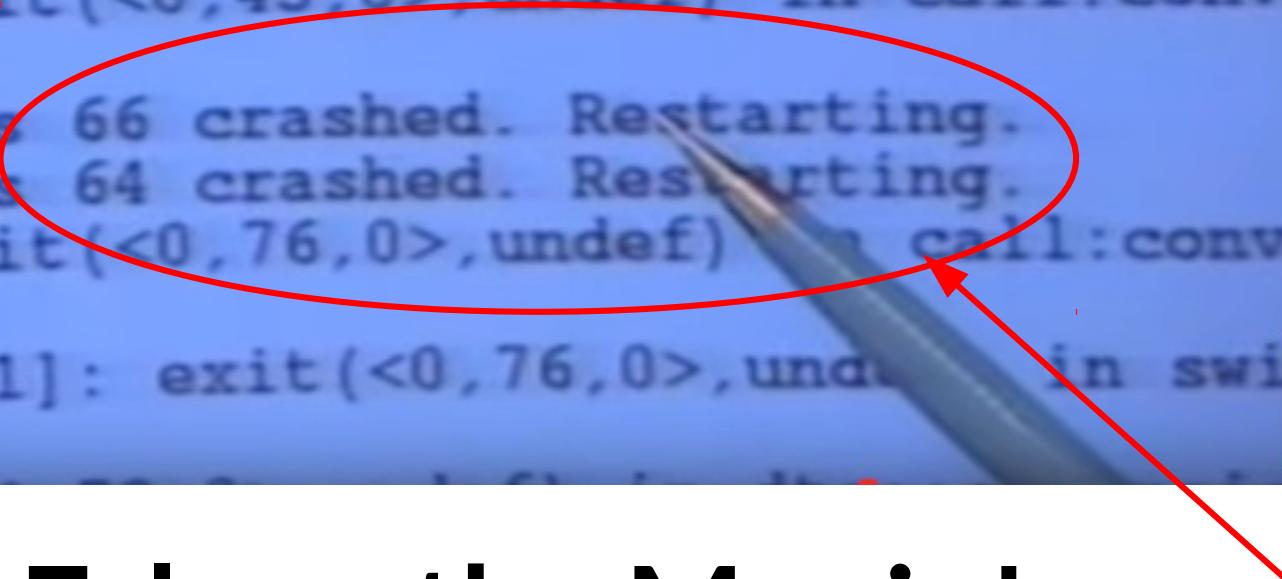
```
utgoing/3]: undefined feature::multi_n  
[7]: exit(<0,80,1>,undef) in call:conver  
_group/1]: exit(<0,80,1>,undef) in switc  
exit(<0,80,1>,undef) in dts::extension/  
exit(<0,81,0>,undef) in dts::extension/  
[3]: exit(<0,43,0>,undef) in call:conver  
address 66 crashed. Restarting.  
address 64 crashed. Restarting.  
[7]: exit(<0,76,0>,undef) in call:conver  
_group/1]: exit(<0,76,0>,undef) in switc
```



Erlang the Movie!

<https://www.youtube.com/watch?v=xrIjfIjssLE>

```
utgoing/3]: undefined feature::mmulti_n  
[7]: exit(<0,80,1>,undef) in call:conver  
_group/1]: exit(<0,80,1>,undef) in switc  
exit(<0,80,1>,undef) in dts::extension/  
exit(<0,81,0>,undef) in dts::extension/  
[3]: exit(<0,43,0>,undef) in call:conver  
address 66 crashed. Restarting.  
address 64 crashed. Restarting.  
[7]: exit(<0,76,0>,undef) in call:conver  
_group/1]: exit(<0,76,0>,undef) in switc
```



10:

Erlang the Movie!

<https://www.youtube.com/watch?v=xrIjfIjssLE>

**Encapsulate state and
behavior**

**Concurrent processors that
exchange messages!**

Inside the Actors studio!











**Flooding Matt Damon
with messages!**



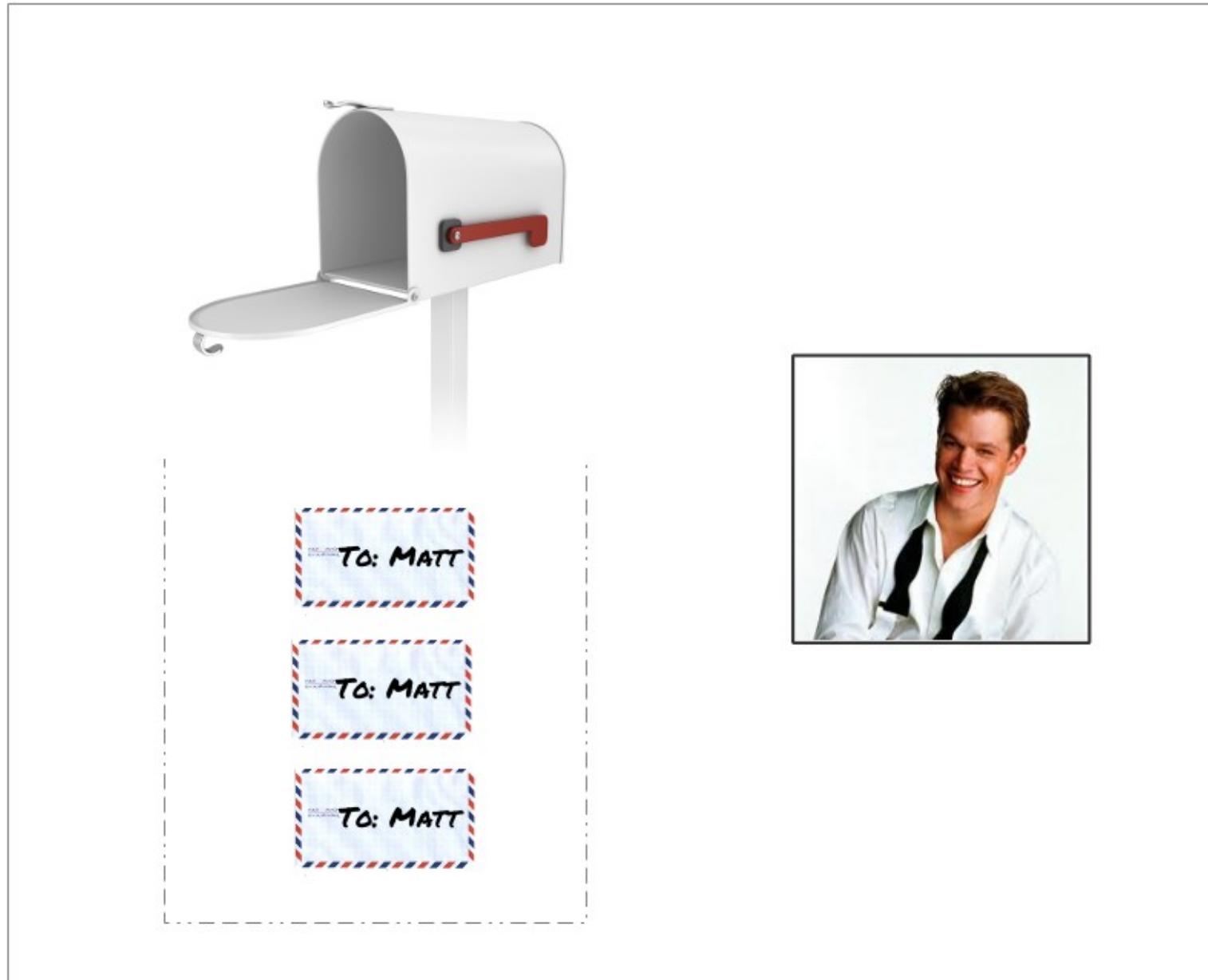


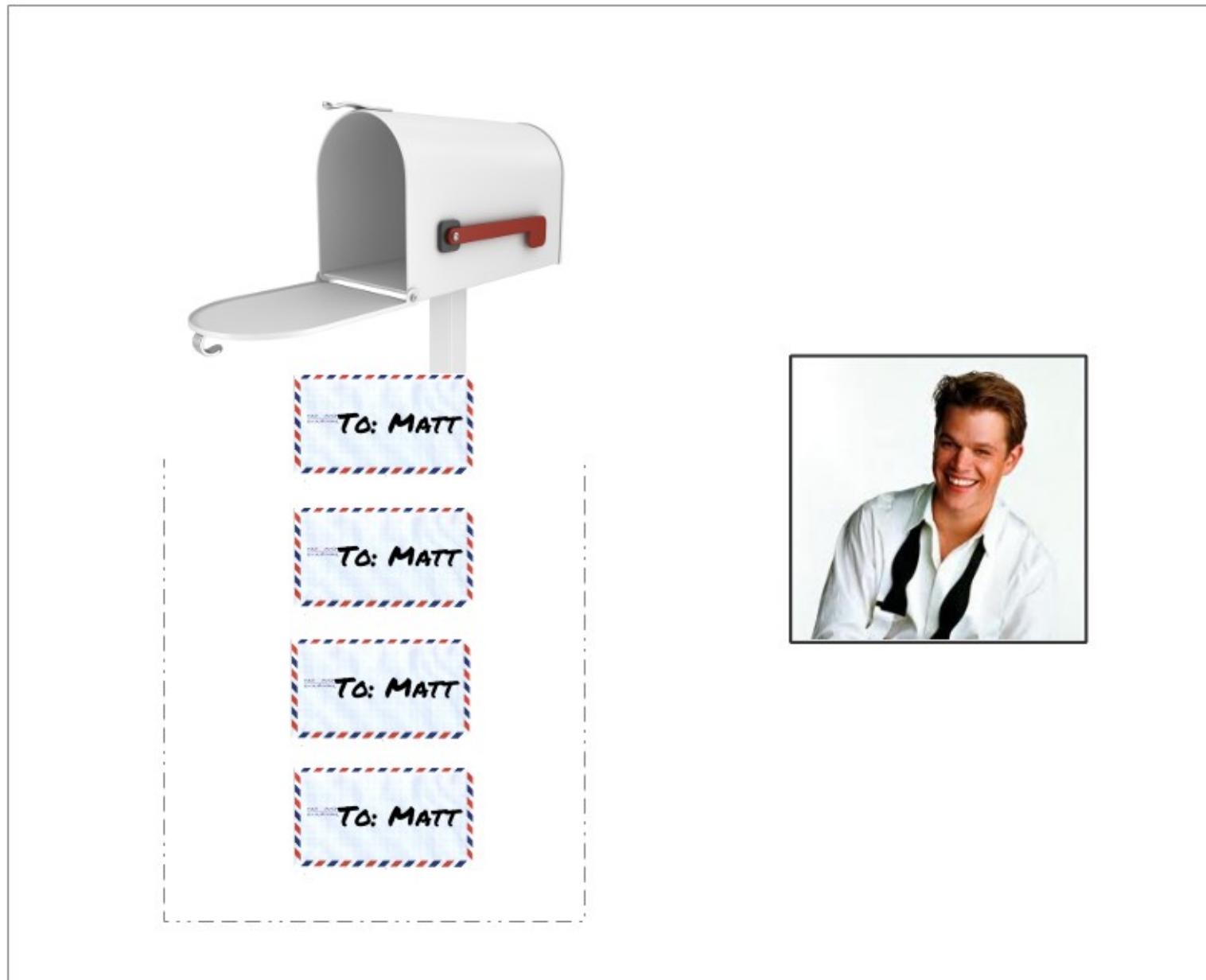


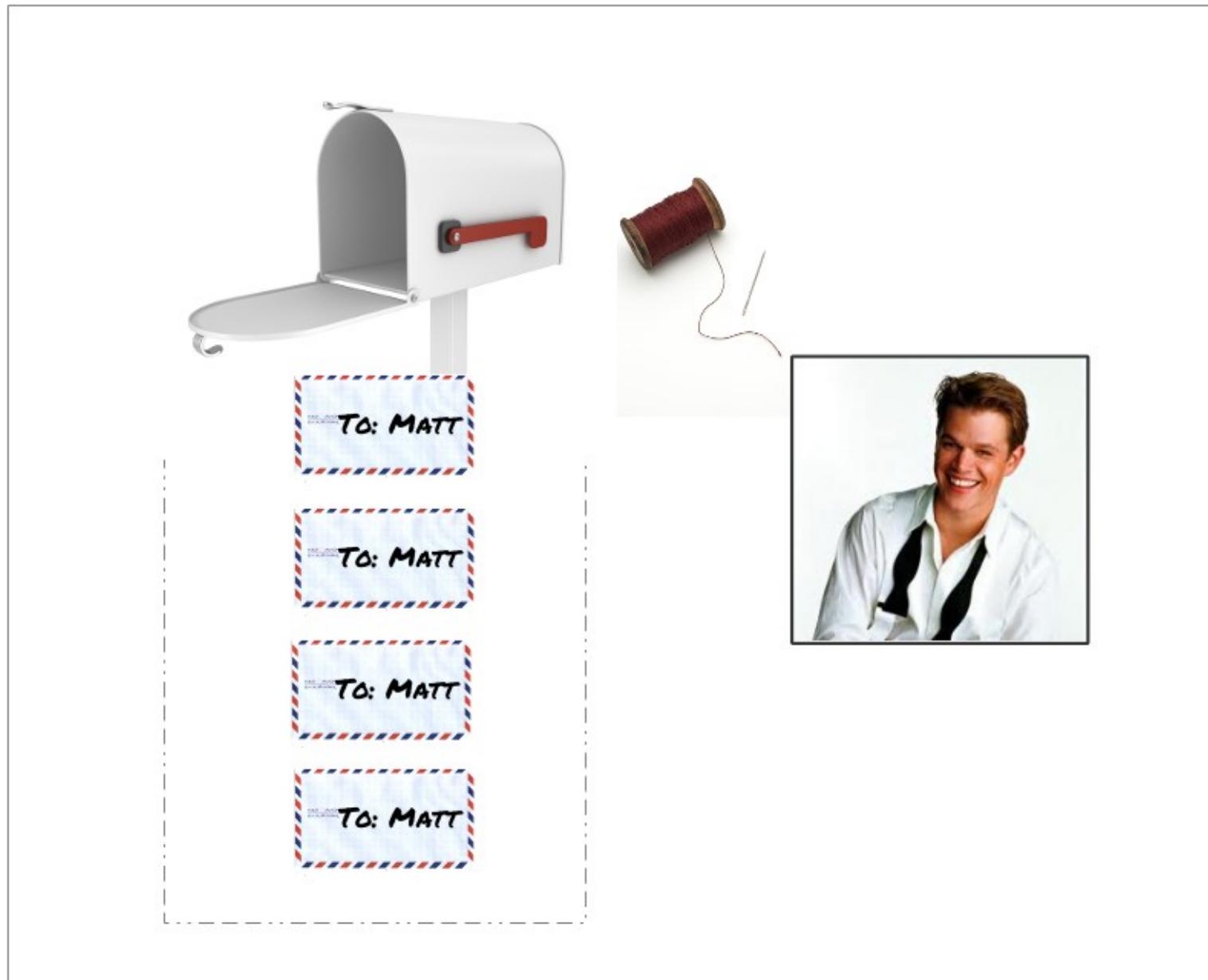


















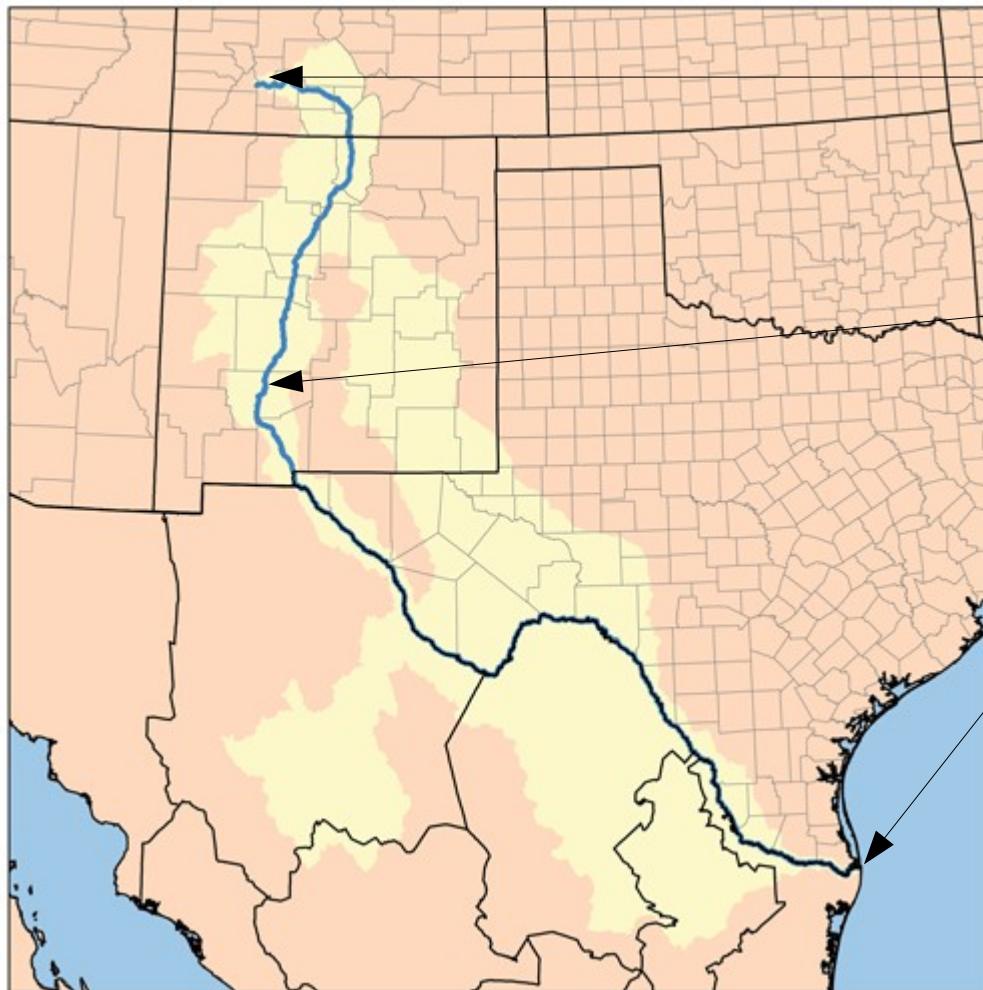




Actor Demo



Reactive Streams



Source – One Output, Emits Data Elements

Flow – One Input and One Output, Connects up and down streams transforming data.

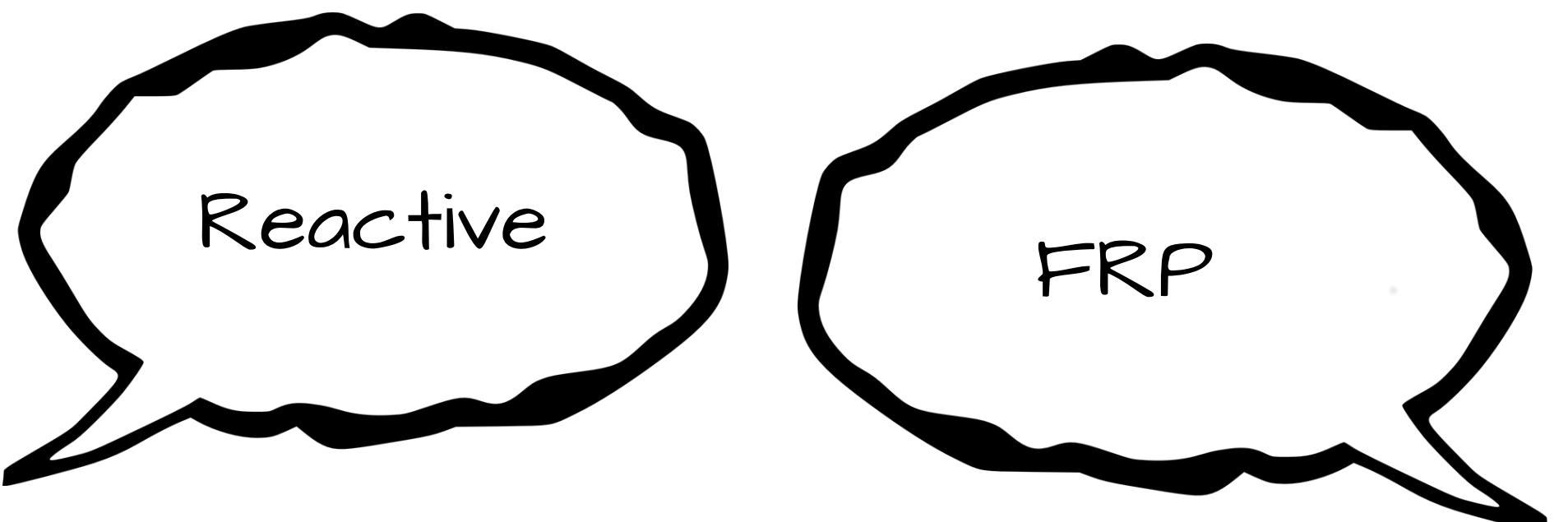
Sink – One Output, requesting and accepting data elements possibly slowing down the upstream producer of elements



Reactive Streams Demo

**What does it mean when
something is
Reactive?**





Reactive

FRP

Reactive

FRP

Reactive
Streams

Reactive
Systems

Reactive

FRP

Reactive
Streams



Reactive
Systems



Reactive

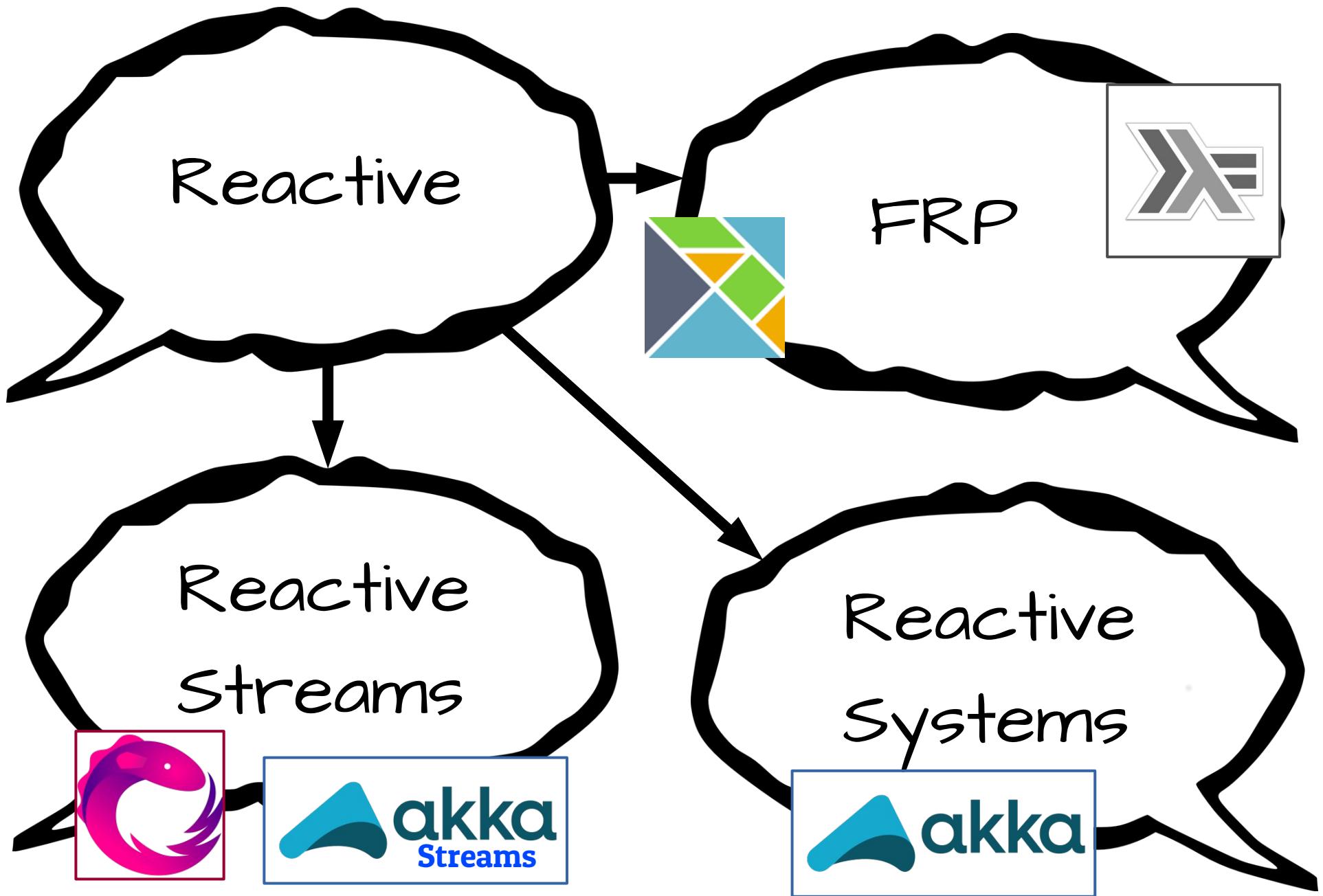
FRP

Reactive
Streams



Reactive
Systems





About knowing the
Differences between...

Reactive Streams *and* Reactive Systems

Thank You!