

Streaming Data - Deep Dive

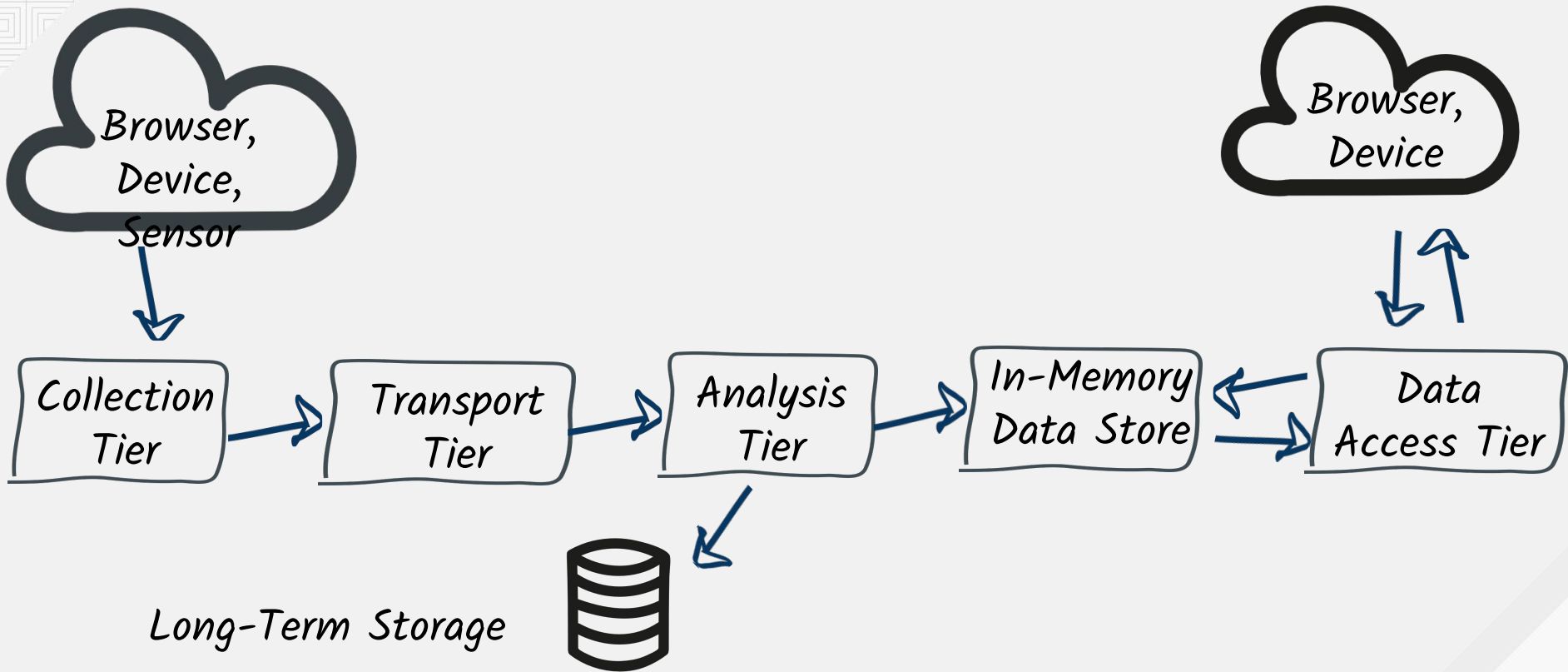
Galder Zamarreño, Clement Escoffier, Katia Aresti,
Thomas Segismont



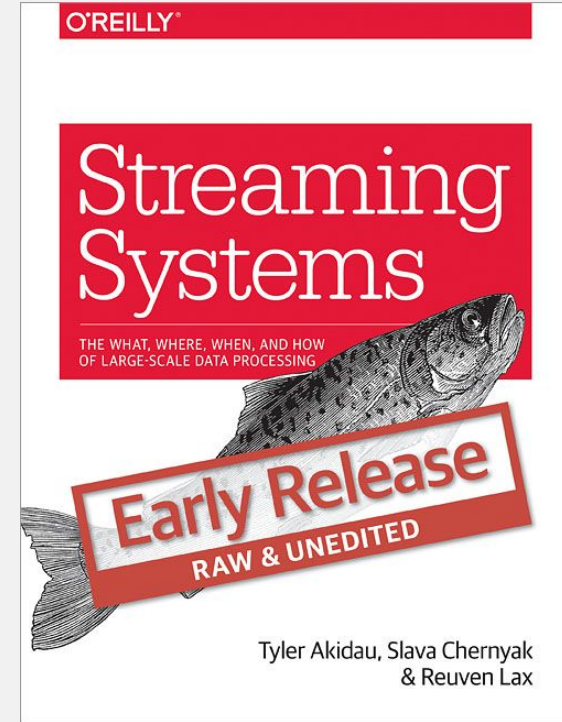
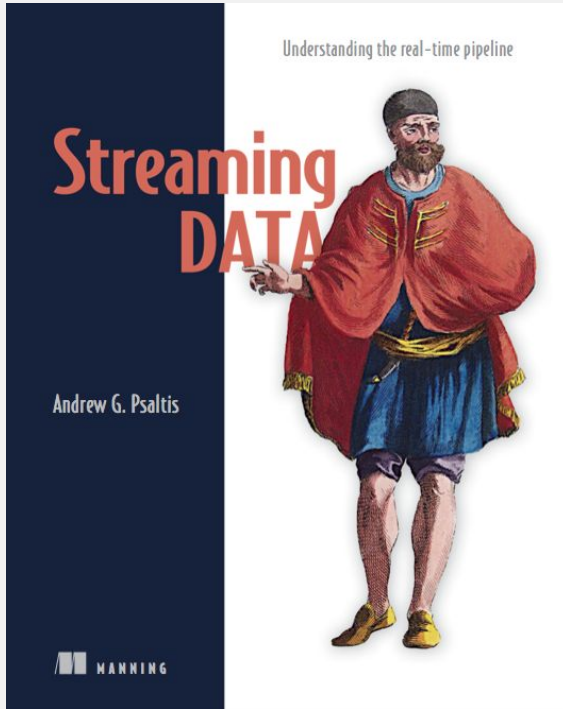
Introduction

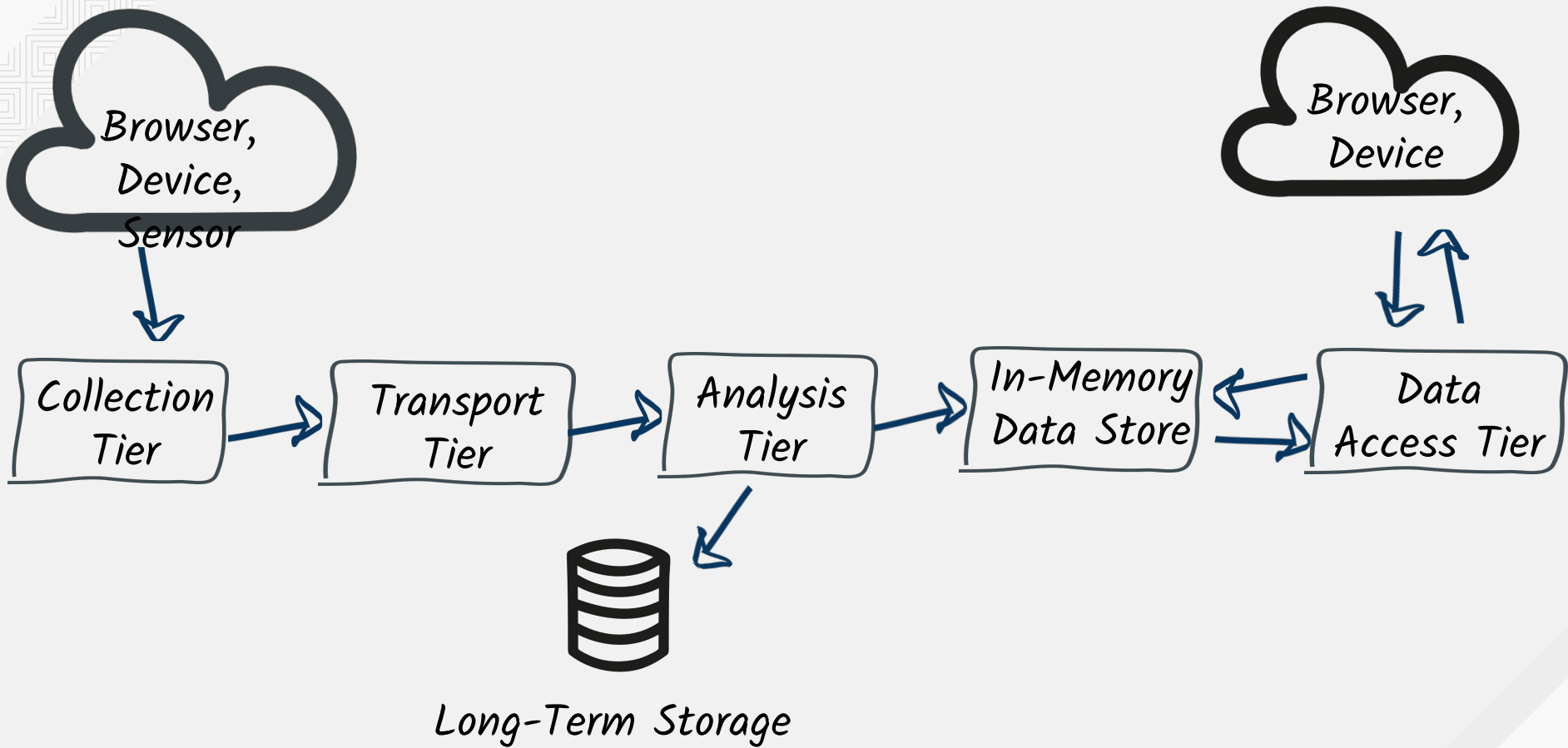
Data Streaming, Microservice, Reactive, and
Containers...

Streaming Data - Architecture Blueprint

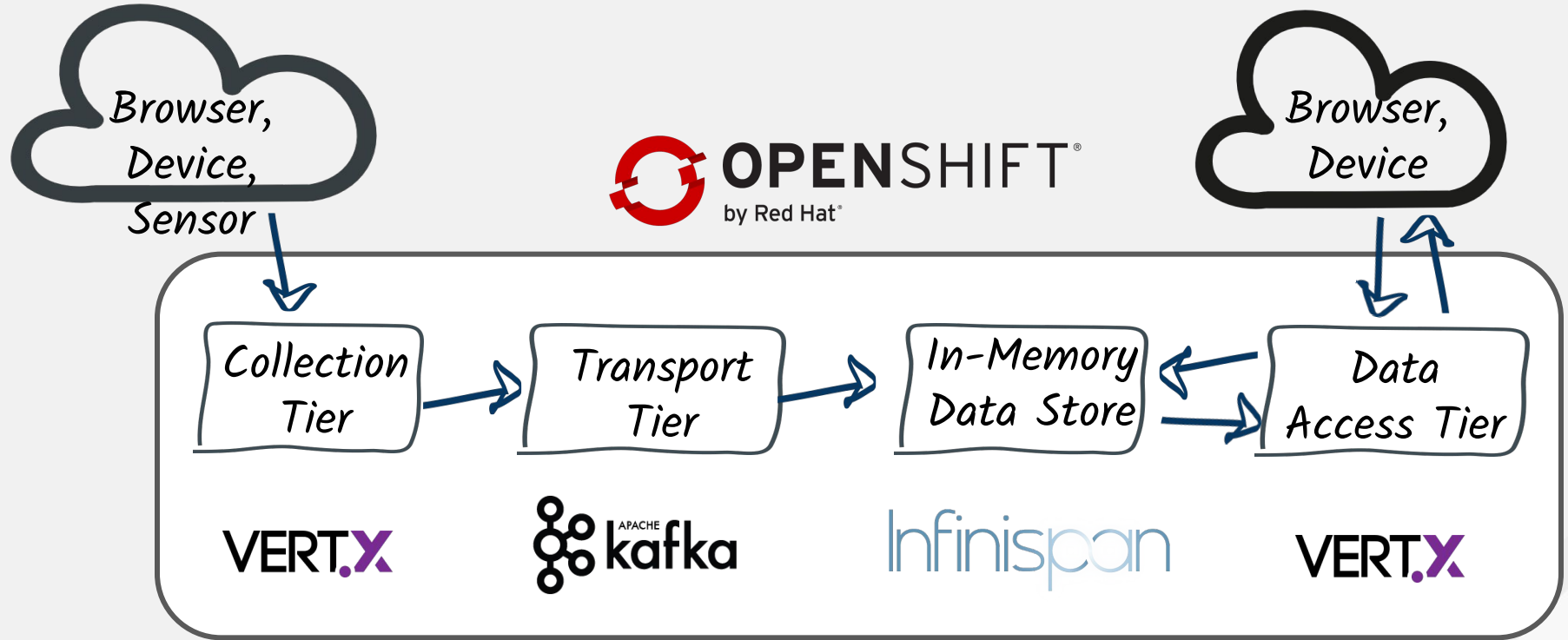


Streaming Data - Books





Streaming Data - Deep Dive



*Microservices - Welcome to a
not so micro-world*

Microservice

The microservice architectural style is an approach to developing a single application as *a suite of small services*, each running in its *own process* and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and *independently deployable by fully automated deployment* machinery. There is a bare *minimum* of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
(Martin Fowler)

Each service runs in its own process

So they are *distributed applications*

- *Lightweight interactions*
- *Loose-coupling*

Not only HTTP

- *Messaging*
- ***Streams***
- *(async) RPC*

A suite of independent services

Independently developed, tested and deployed

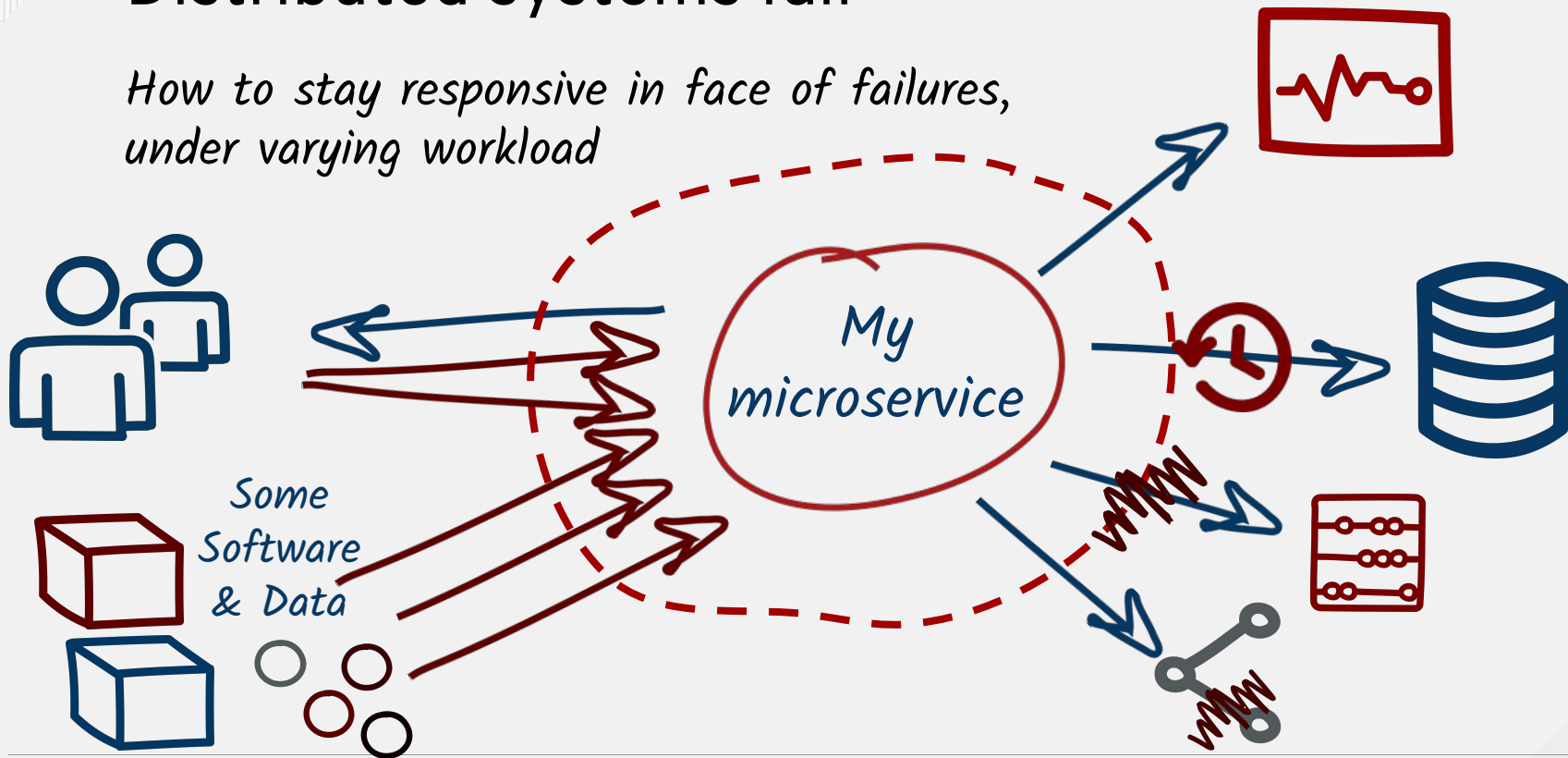
- Automated process
- (Liskov) substitutability

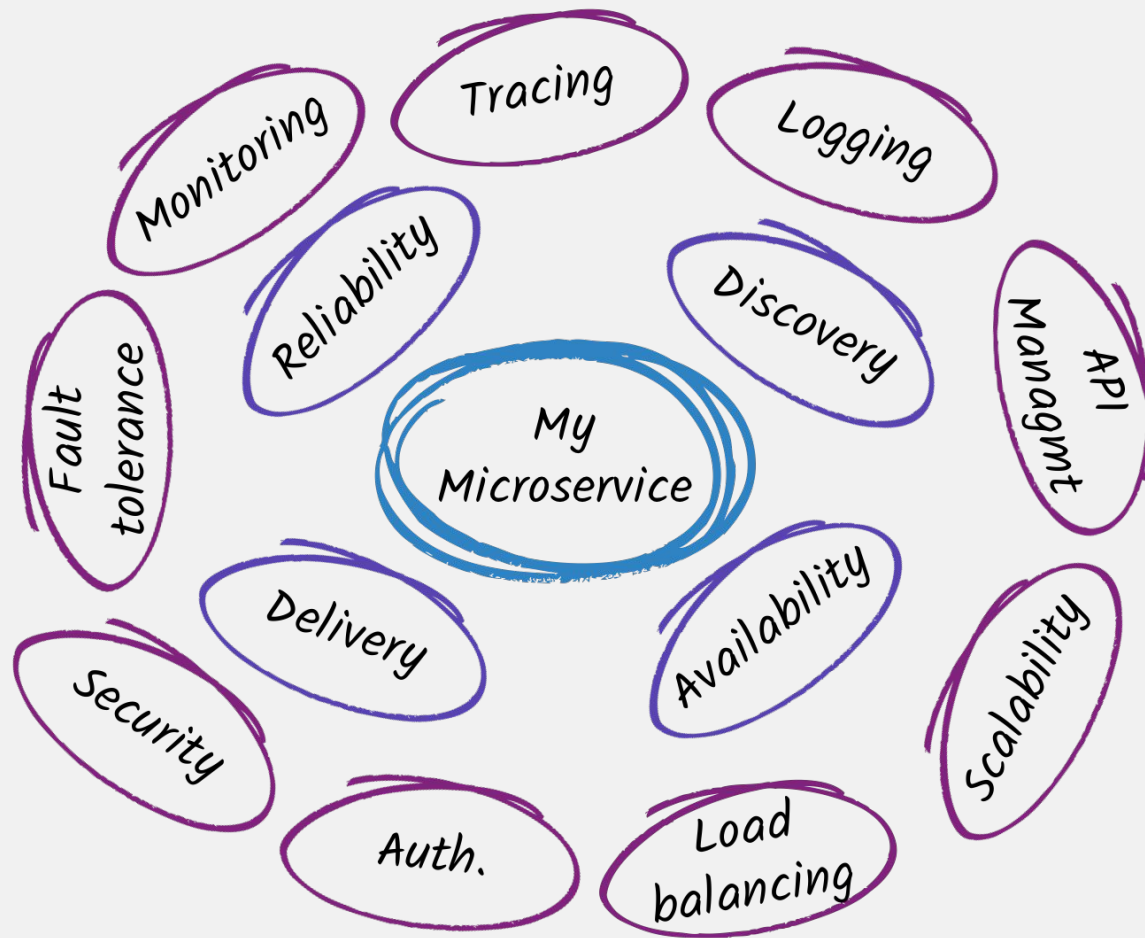
It's all about *agility*

- Agility of the composition
- You can replace any microservice
- You can decide who uses who

Distributed systems fail

*How to stay responsive in face of failures,
under varying workload*



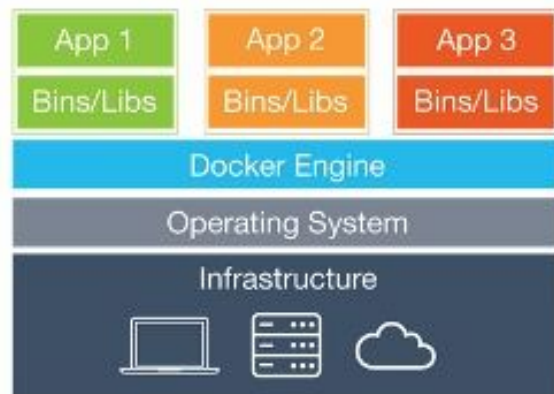


Containers

Containers are **NOT** light VMs

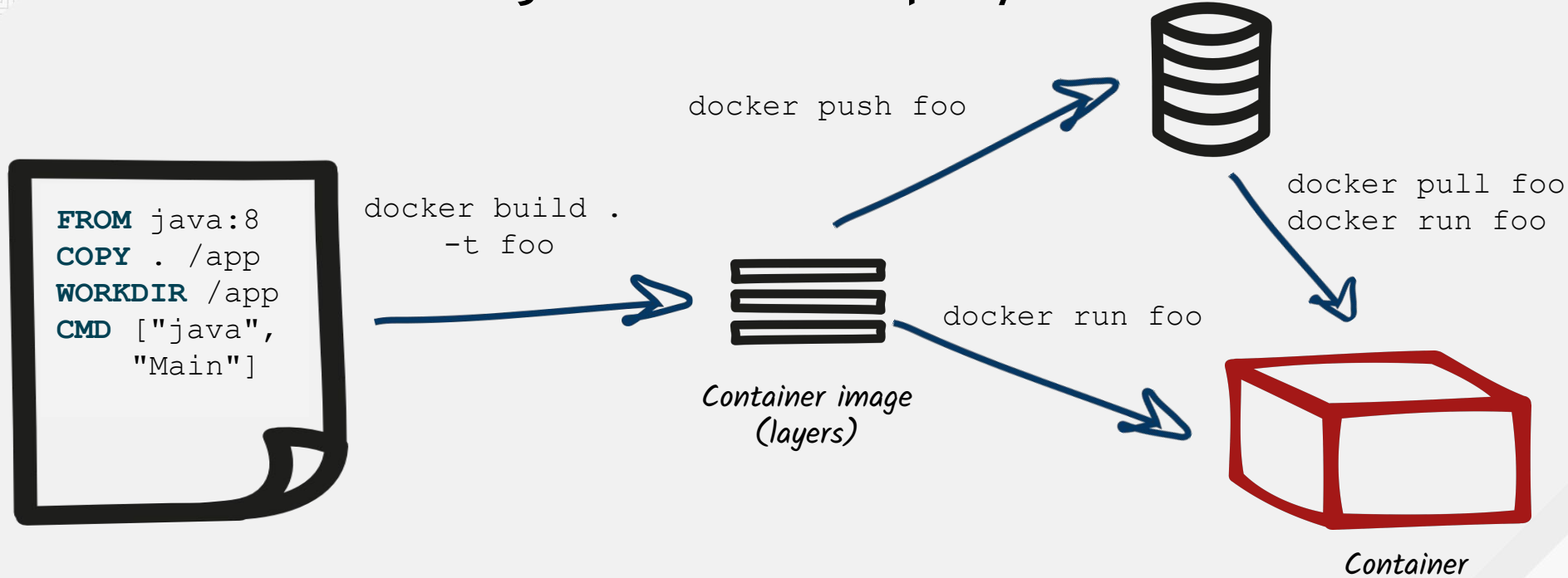


Virtual Machines



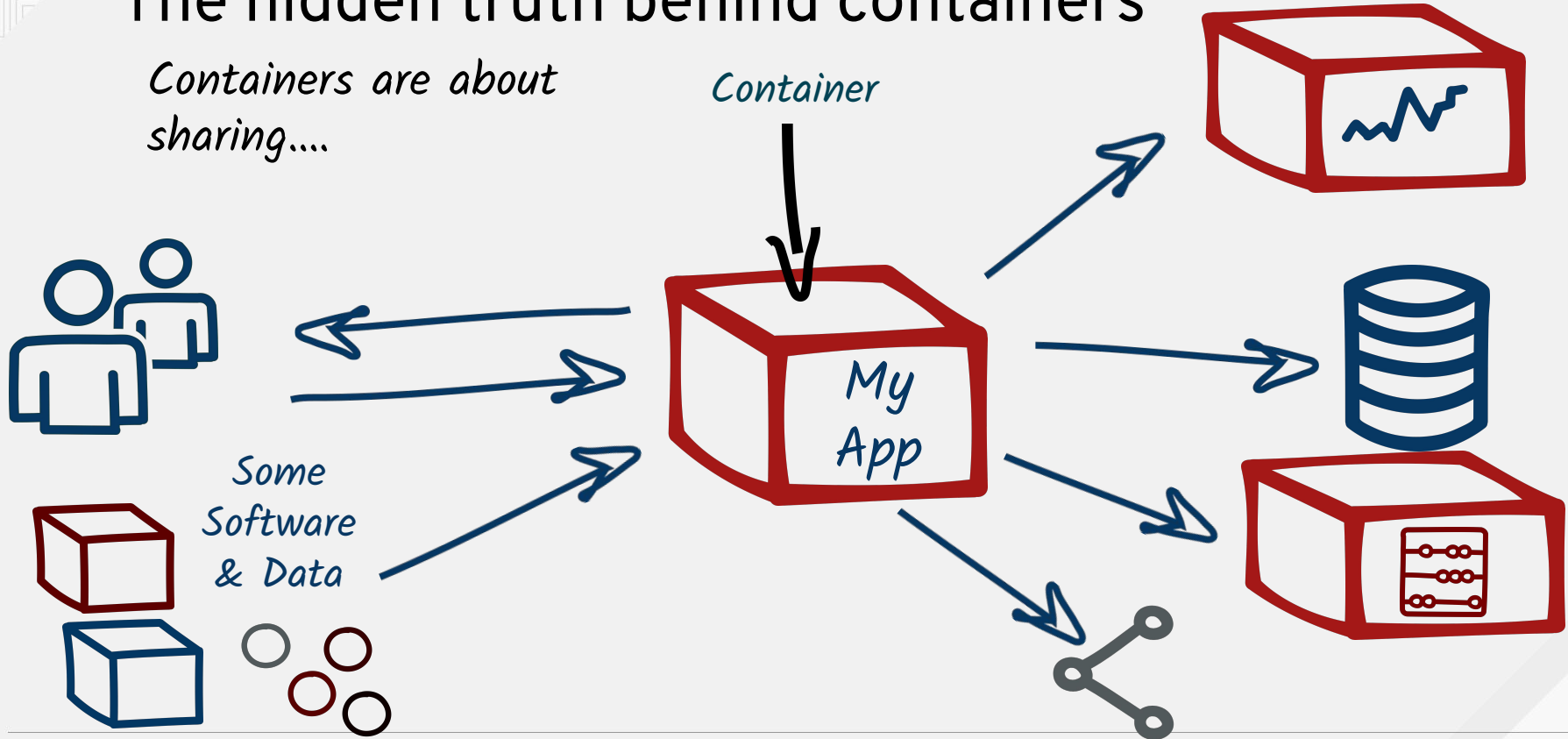
Containers

Container image: the new deployment unit



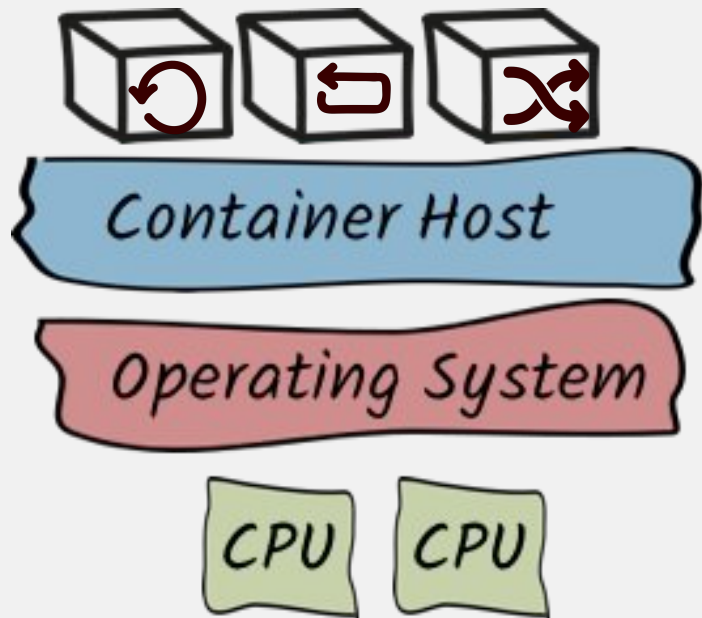
The hidden truth behind containers

Containers are about sharing....



The hidden truth being containers

Containers are about sharing



Thread-based execution model are not efficient in containers:

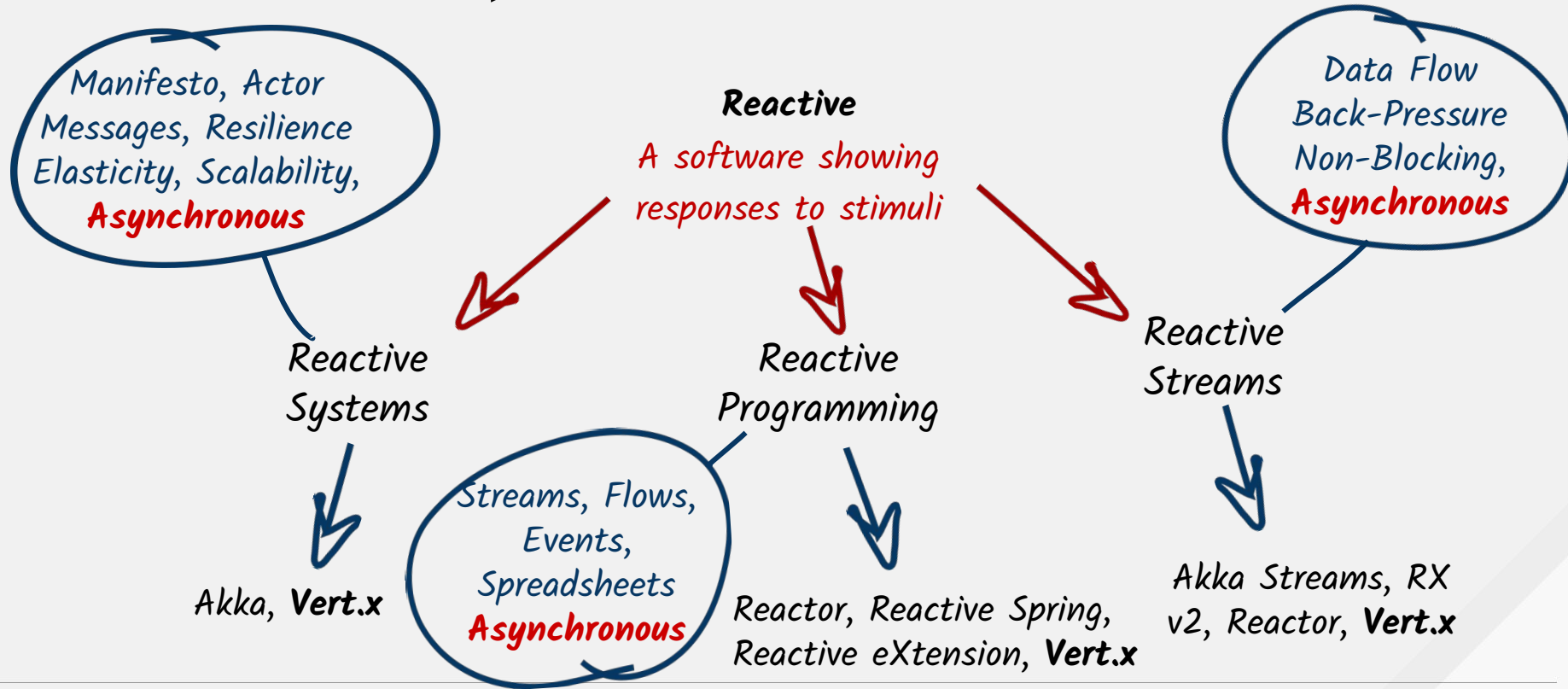
- Too much memory (threads are expensive)
- CPU quotas used to manage thread switch
- Tuning thread pool is hard

Reactive all the things...

Reactive all the things ???

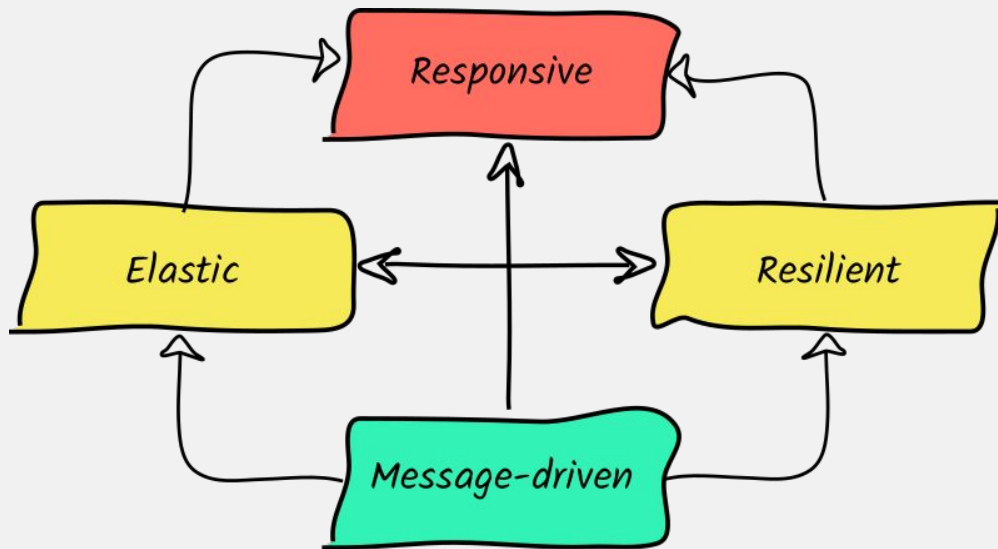
Message Resilience Back-Pressure
Elasticity Streams
Manifesto System
Asynchrony Scalability Data Flows
Actor eXtensions Asynchronous
Programming Observable
Events Spreadsheets RX Java
Reactor Spring

The reactive spectrum



Reactive Manifesto

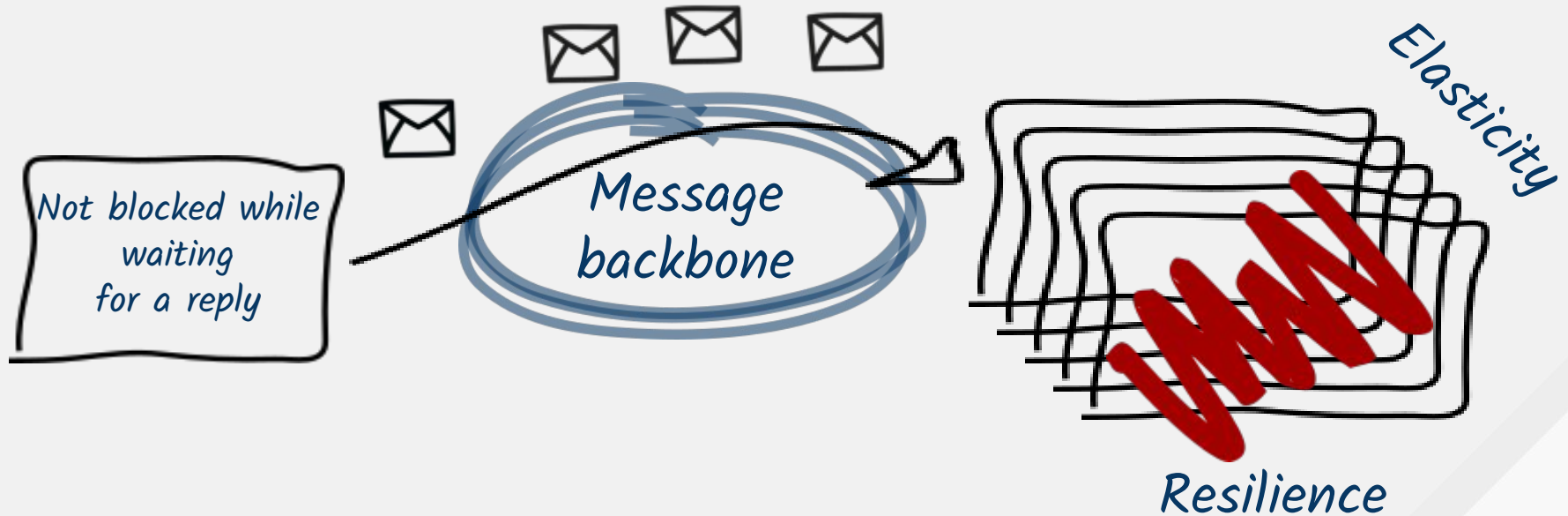
<http://www.reactivemanifesto.org/>



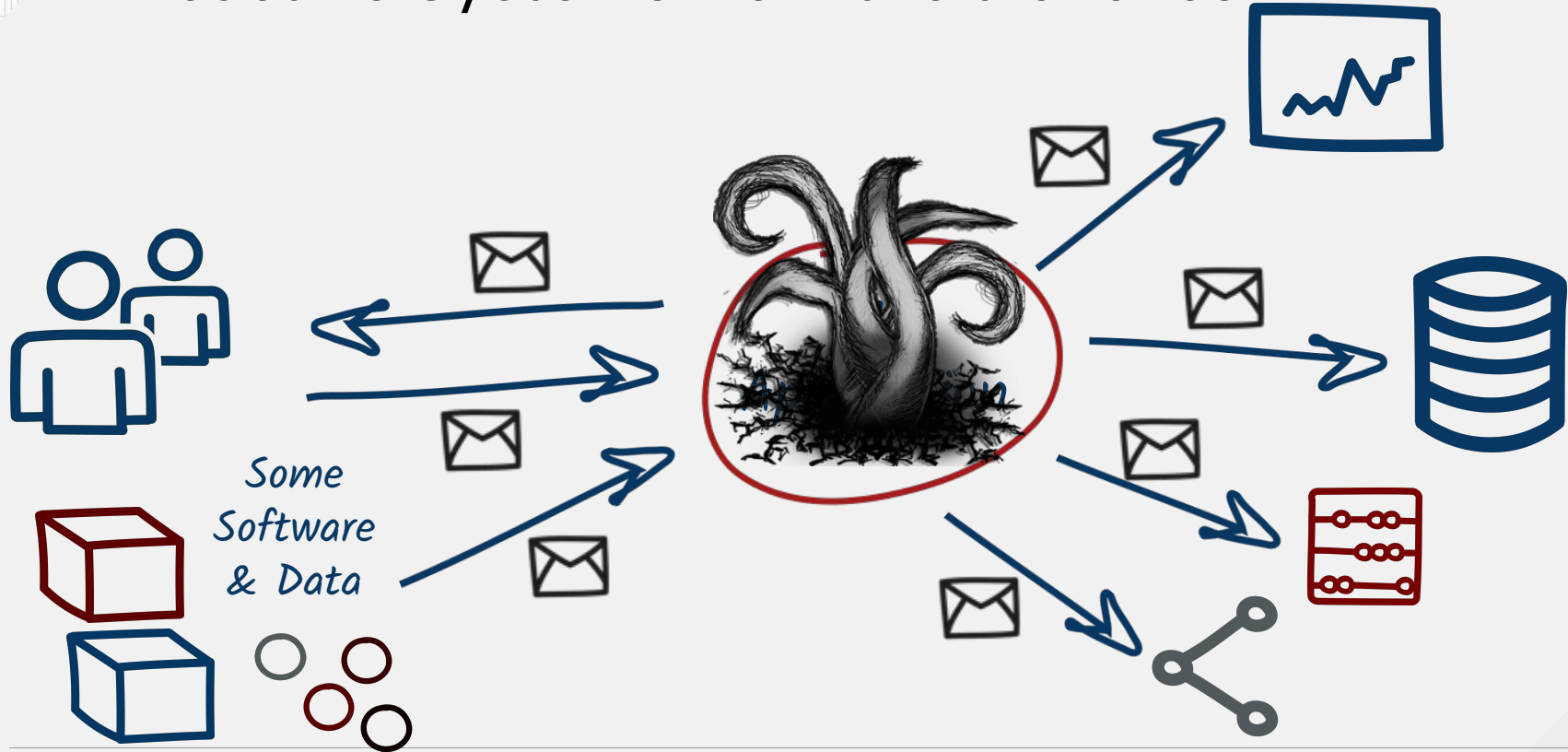
Asynchronous message passing



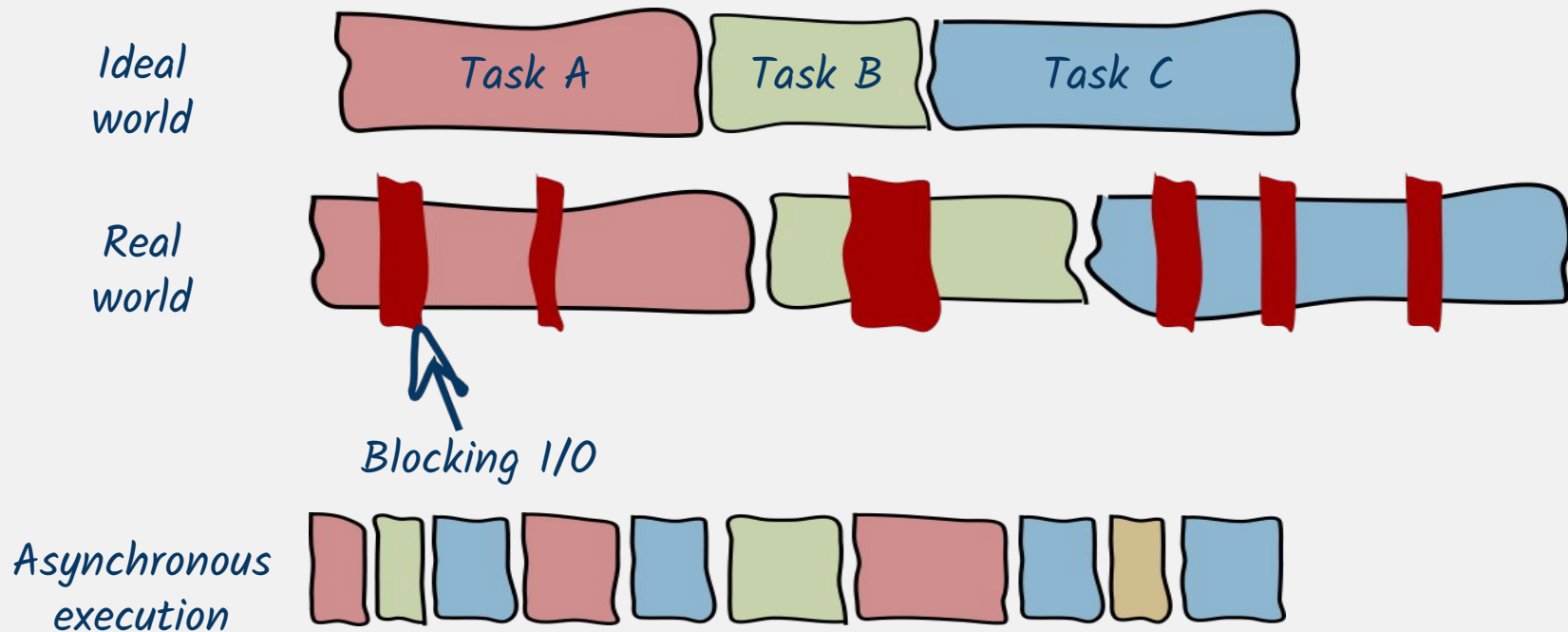
Asynchronous message passing



Reactive Systems from the trenches

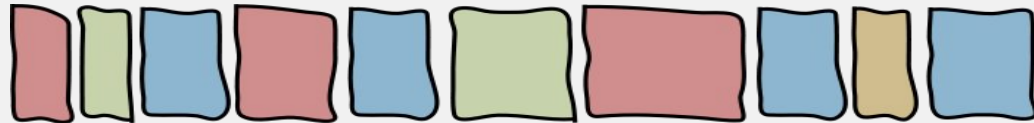


Asynchronous execution



Asynchronous execution

*Asynchronous
execution*

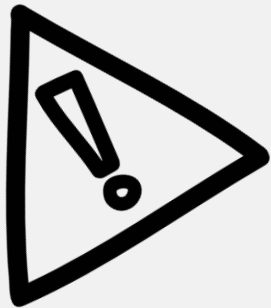
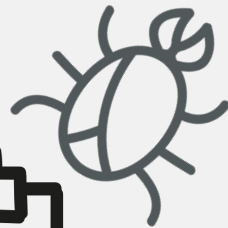
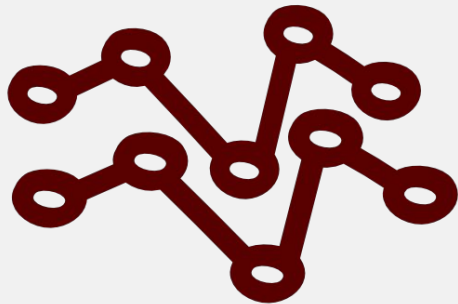


Async programming model

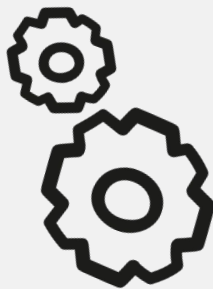
Non-blocking IO

Task-based concurrency

This is what Eclipse Vert.x offers



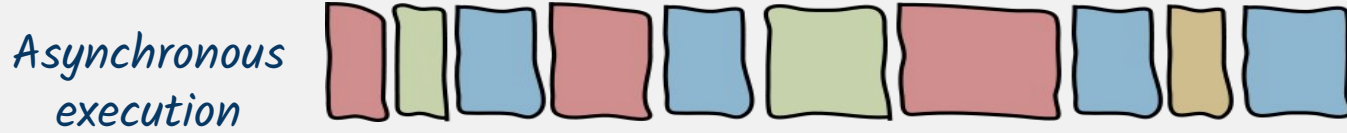
DEMO TIME!



Taming the async beast with RX Java 2



Asynchronous execution



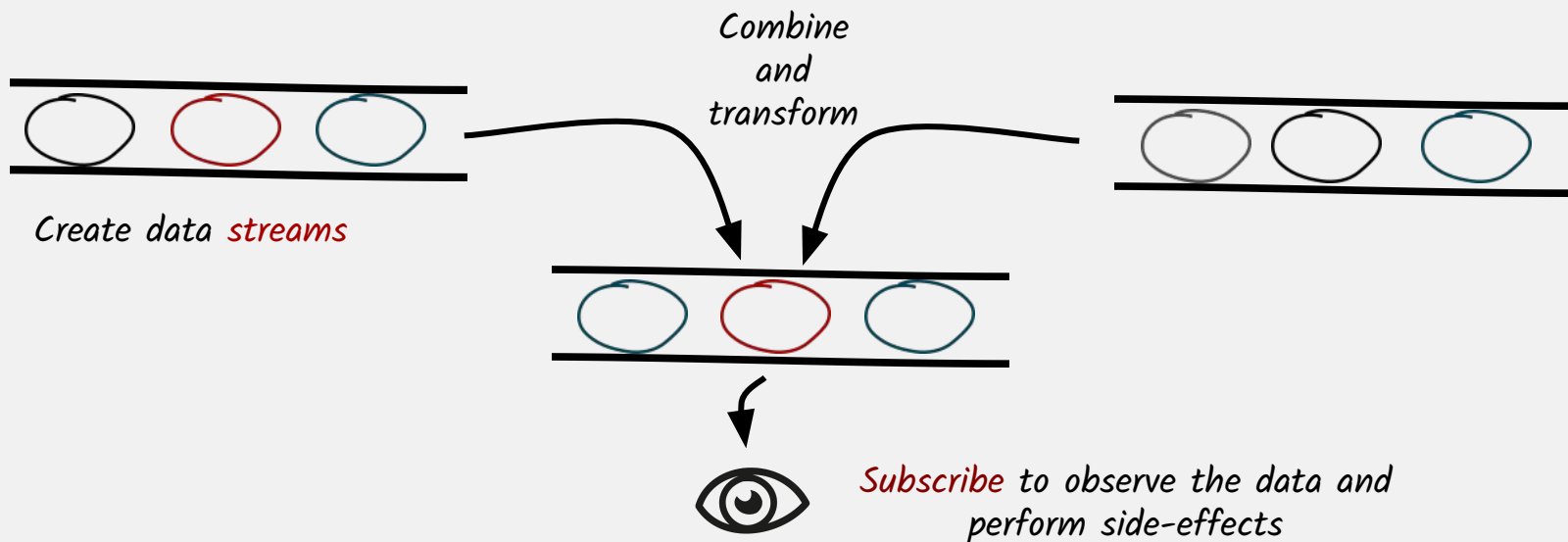
Async programming model => RX Java 2

Non-blocking IO => Vert.x

Task-based concurrency => Vert.x

Reactive eXtension - <http://reactivex.io>

Combination of the best ideas from the *Observer* pattern, the *Iterator* pattern, and *functional programming*



3 types of notifications

stream

```
.doOnNext(item -> System.out.println(item))  
.doOnError(err -> err.printStackTrace())  
.doOnComplete(() -> System.out.println("End of  
stream"));
```


Transformation: map

Synchronous computation, extraction....



```
stream  
  .map(item -> item + 1);
```



Transformation: flatMap

Async composition



stream

```
.flatMap(item -> Observable  
  .fromArray(item, item));
```



Subscription

*If you don't **subscribe**, nothing happen*

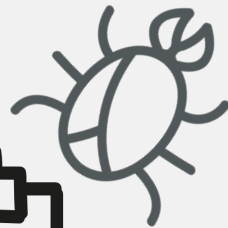
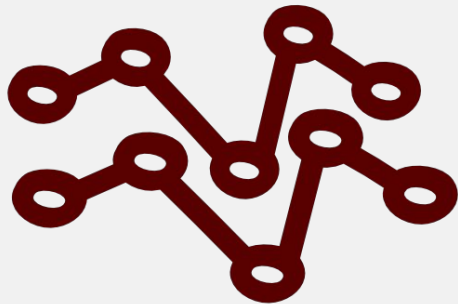
```
stream
  .subscribe(
    item -> System.out.println(item),
    err -> err.printStackTrace(),
    () -> System.out.println("End of stream")
  );
```

Reactive Types

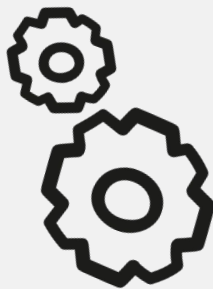
Type	Type of stream	Notifications	Use case
Completable	Stream without items	Error, End of stream	Asynchronous action without a result
Single	Stream with 1 item	Item, Error, End of stream	Asynchronous action
Maybe	Stream with 0 or 1 item	Item, Error, No Item	Asynchronous lookup
Observable	Several items	Each item, Error, End of stream	Notification, Stream without back-pressure
Flowable	Several items + Back pressure	Each item, Error, End of stream	Flow of data

Reactive Types

Calling a service	<code>Single<JsonObject> invoke();</code>
Flushing a storage	<code>Completable flush();</code>
Query / Lookup	<code>Maybe<Person> findByName();</code>
Keystrokes	<code>Observable<Integer> keyStrokes();</code>
Read a file	<code>Flowable<Buffer> read();</code>



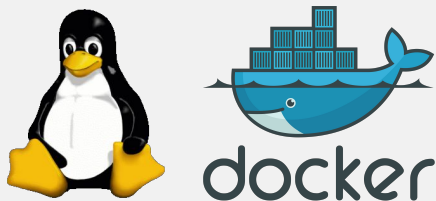
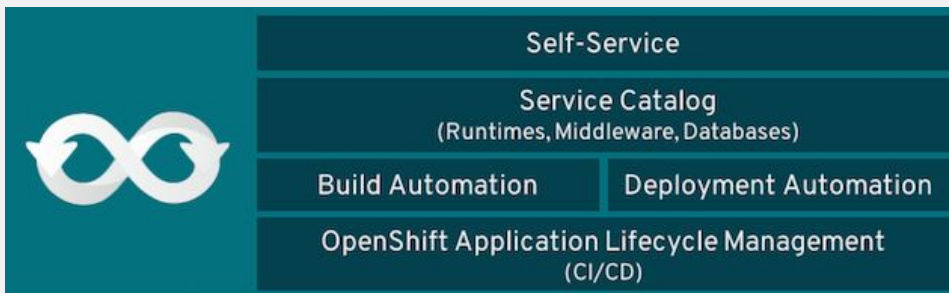
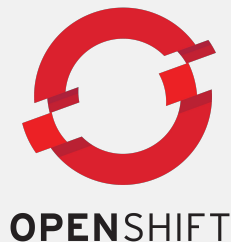
DEMO TIME!



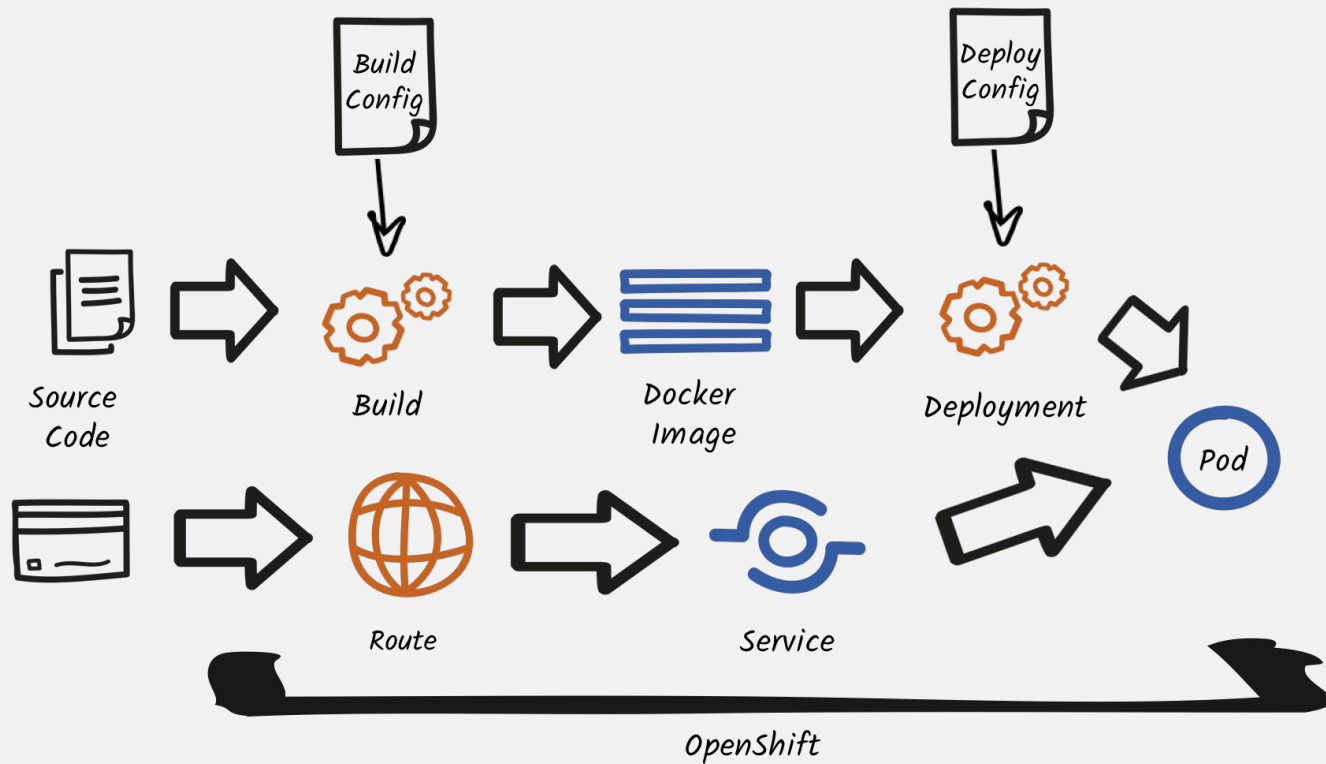
*1 container, 2 containers,
3 containers.... BOOM!*

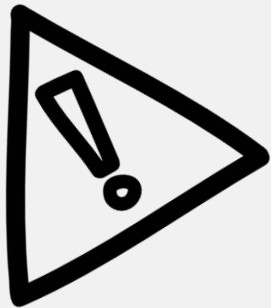
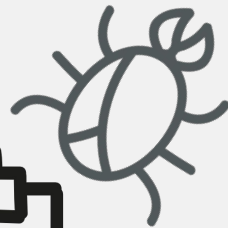
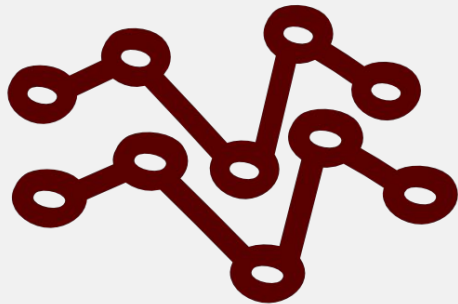


OpenShift - A Kubernetes distribution

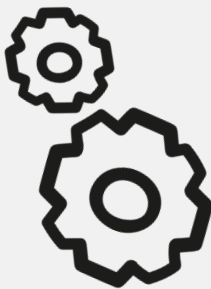


OpenShift - Workflow



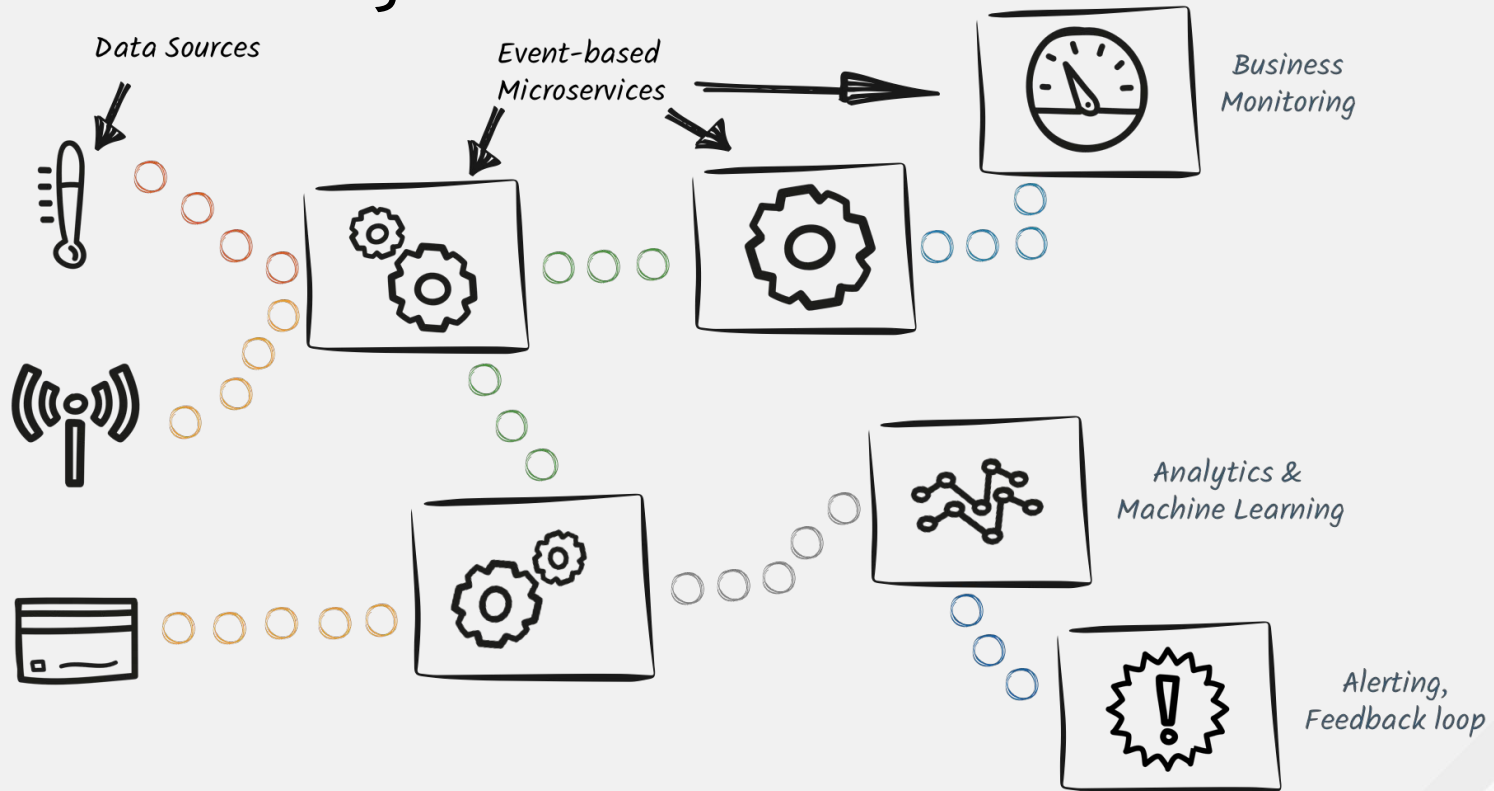


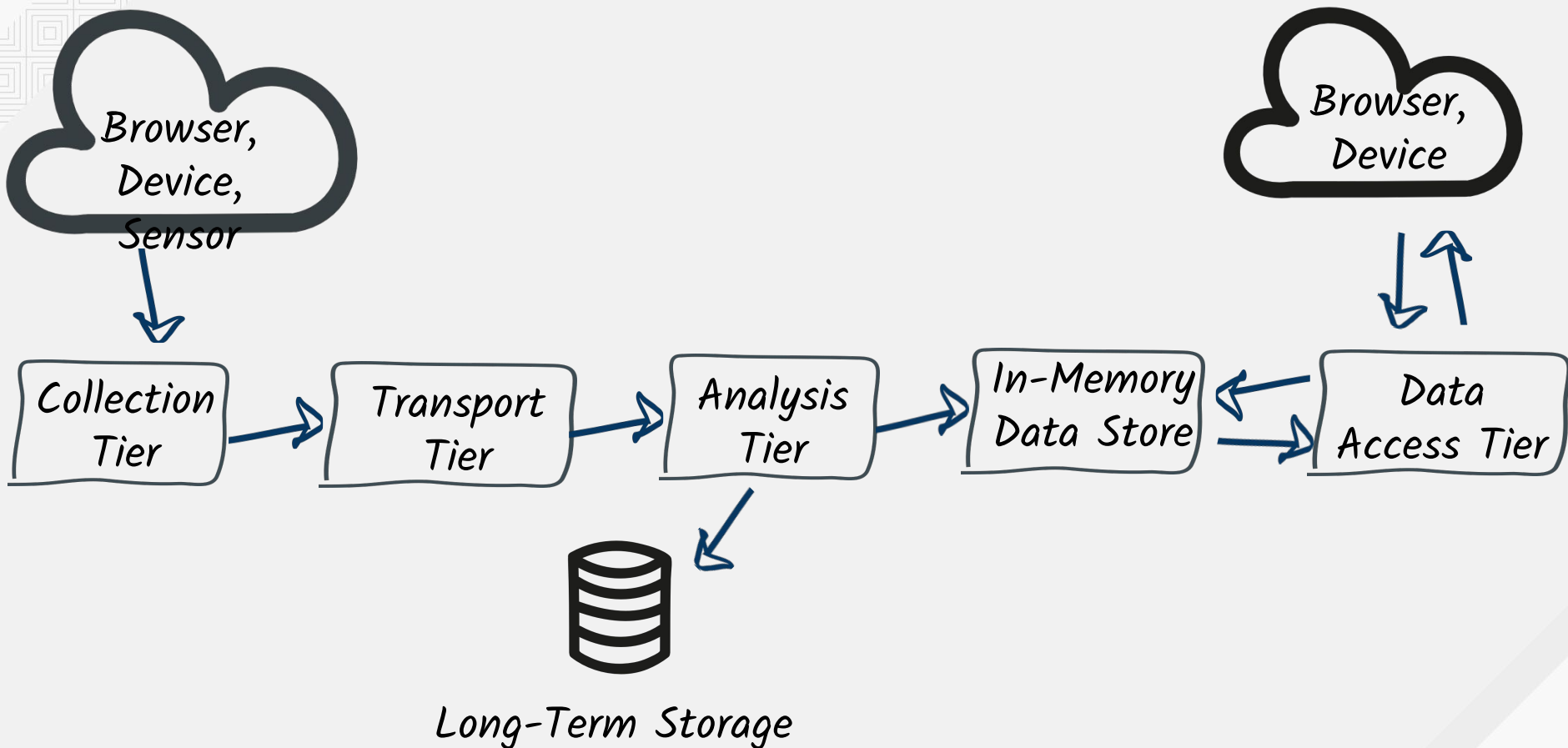
DEMO TIME!



What about streaming?

Data Streaming

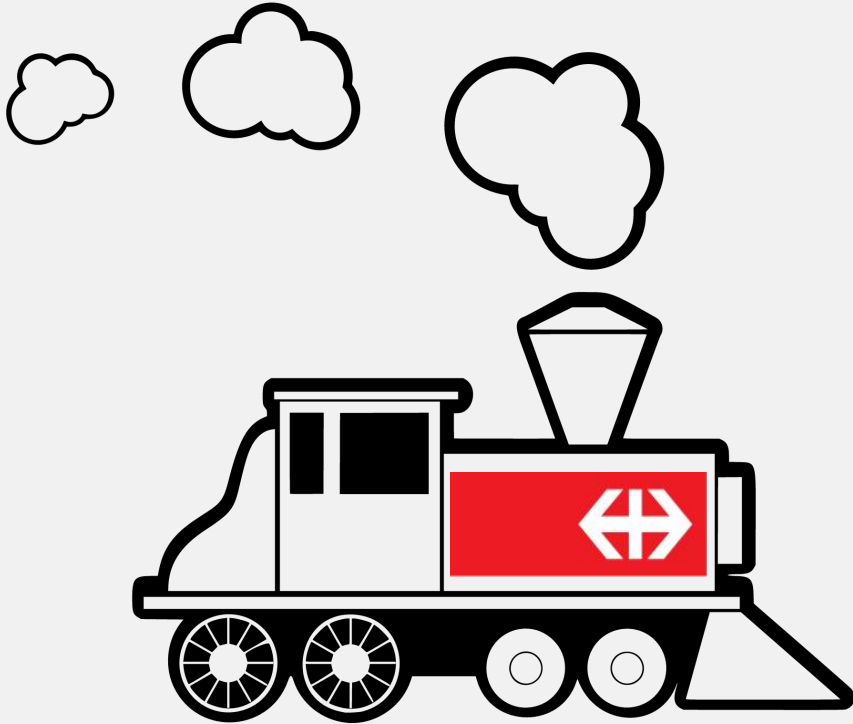




Objectives

*real-time data streams to build robust,
reactive and scalable applications*

Use Case: Swiss Trains



Two data streams, from 2 different providers:

- *Swiss transport timetable data (opendata.ch)*
- *Positions of trains at a given time (sbb.ch)*

Dashboard with delayed trains

Web application monitoring the delayed trains position


```
{"stop":{"station":{"id":"8500301","name":"Rheinfelden","score":null,"coordinate":{"type":"WGS84","x":47.55121,"y":7.792155},"distance":null},"arrival":null,"arrivalTimestamp":null,"departure":"2016-02-29T17:34:00+0100","departureTimestamp":1456763640, "delay":3,"platform":"4","prognosis":{"platform":"4","arrival":null,"departure":"2016-02-29T17:37:00+0100","capacity1st":1,"capacity2nd":1},"realtimeAvailability":null,"location":{"id":"8500301","name":"Rheinfelden","score":null,"coordinate":{"type":"WGS84","x":47.55121,"y":7.792155},"distance":null}}, "name":"IR 1978" , "category":"IR","categoryCode":2,"number":"1978", "operator":"SBB","to":"Basel SBB","capacity1st":null,"capacity2nd":null, "subcategory":"IR","timeStamp":1456761753983,"nextStation":{"station":{"id":"8500301","name":"Rheinfelden","score":null,"coordinate":{"type":"WGS84","x":47.55121,"y":7.792155},"distance":null},"arrival":"2016-02-29T17:34:00+0100","arrivalTimestamp":1456763640,"departure":null,"departureTimestamp":null,"delay":null,"platform":"","prognosis":{"platform":null,"arrival":null,"departure":null,"capacity1st":null,"capacity2nd":null},"realtimeAvailability":null,"location":{"id":"8500301","name":"Rheinfelden","score":null,"coordinate":{"type":"WGS84","x":47.55121,"y":7.792155},"distance":null}},"@version":"1","@timestamp":"2016-02-29T16:02:34.781Z"}
```

```
{ "x": "8290840", "y": "47483629", "name": "IR 1978",  
  "trainrefdate": "29.02.16", "category": "IR", "trainid": "84/25934/18/24/95", "direction": "15",  
  "prodclass": "4", "delay": "7", "passproc": "", "lstopname": "Basel  
SBB", "poly": [{ "x": "8290840", "y": "47483629", "passproc": "",  
    "msec": "0", "direction": "15"}, {"x": "8290193", "y": "47483647", "passproc": "", "msec":  
    "2000", "direction": "15"}, {"x": "8289528", "y": "47483674", "passproc": "", "msec":  
    "4000", "direction": "15"}, {"x": "8288863", "y": "47483701", "passproc": "", "msec":  
    "6000", "direction": "15"}, {"x": "8288198", "y": "47483728", "passproc": "", "msec":  
    "8000", "direction": "15"}, {"x": "8287532", "y": "47483755", "passproc": "", "msec":  
    "10000", "direction": "15"}, {"x": "8286885", "y": "47483773", "passproc": "", ...}], "timestamp":  
1456761728202, "@version": "1", "@timestamp": "2016-02-29T16:02:11.321Z"  
}
```

Jump on the train!