RxJava – Reactive EXtensions for the JVM

Some reflection…

- **Only recently - Legacy…**
  - **Large applications with 10s of servers**
  - **Seconds of response times**
  - **Hours of offline maintenance**
  - **Gigabytes of Data**
- **Today's Applications**
  - **Deployed on anything from Mobile to Cloud Clusters with thousands of multi core processors**
  - **Millisecond response times**
  - **100% uptime**
  - **Petabytes of Data**
- **Traditional Architectures fail to meet today's demands**

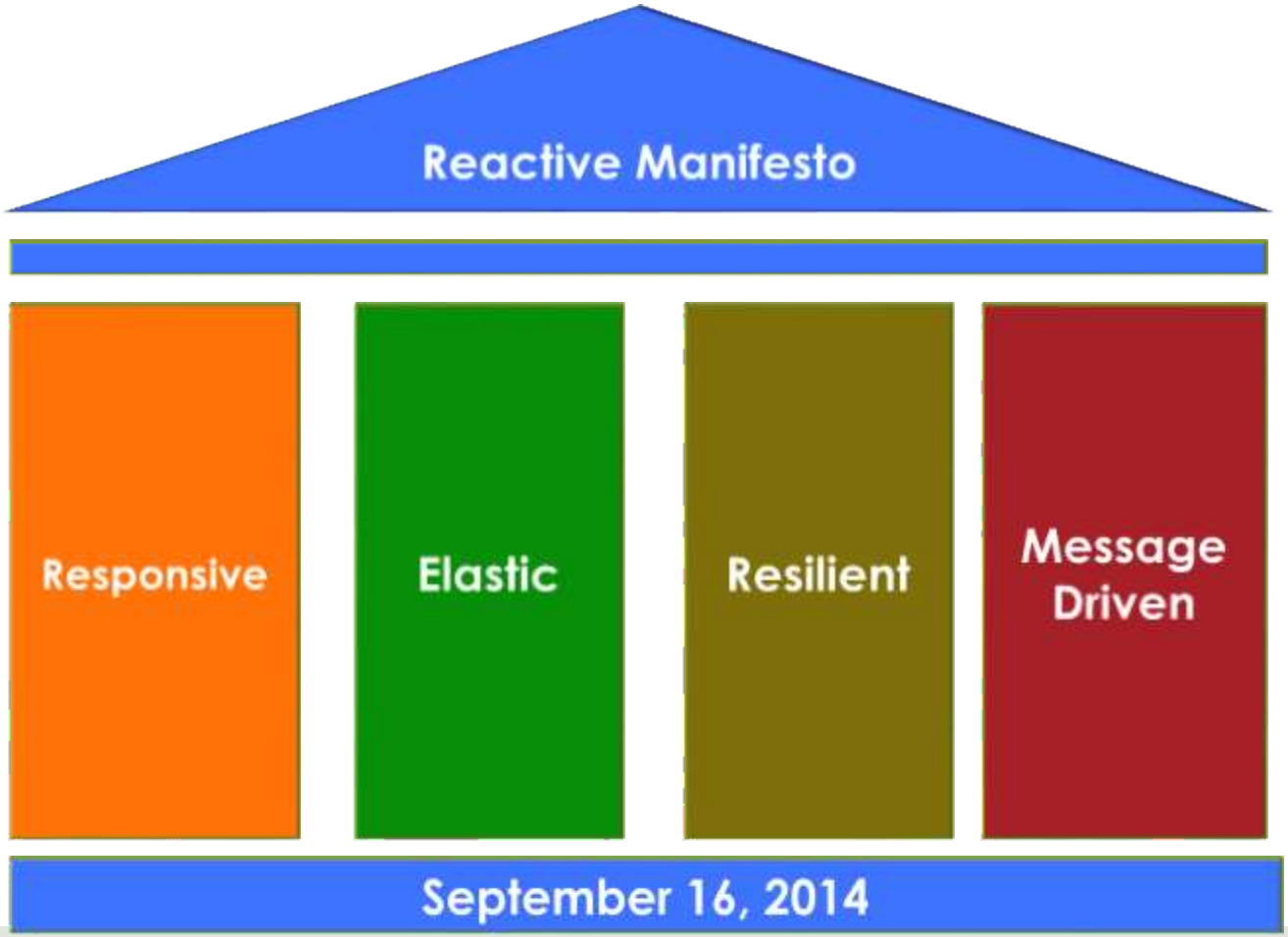
So How do we meet today's demands ?

Reactive Systems

Reactive Definition

- Reactive is being readily responsive to a stimulus

http://www.reactivemanifesto.org



Event Stream(s)

Stocks

splunk>

Splunk Events

Mouse Clicks

Tweets

Functional Reactive Programming

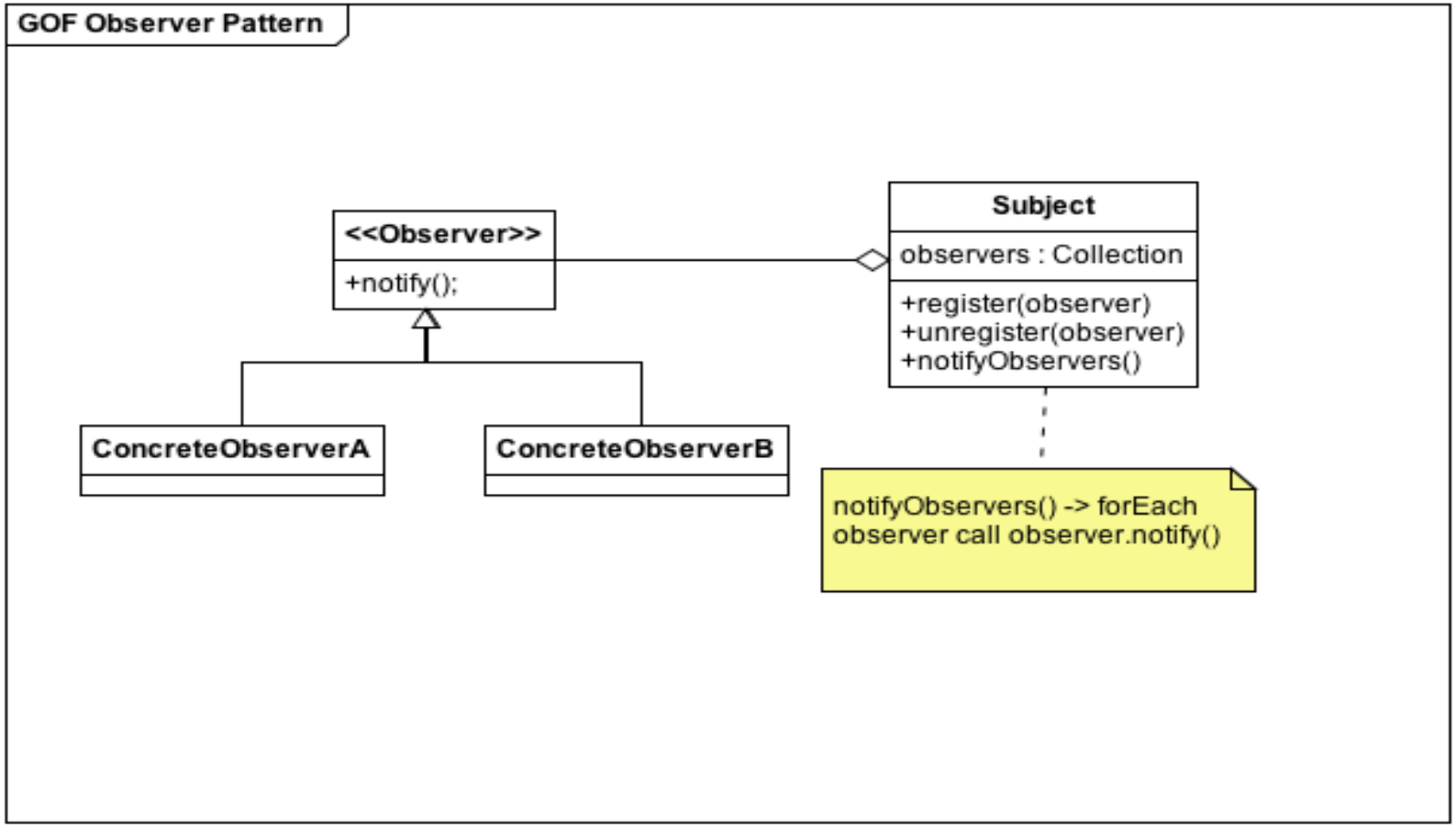
Reactive Model is **Push** rather than Pull

- **Values are pushed when ready in a non blocking manner**
- **Facilitates parallelism**
- **Application of 'functions' on data stream to transform data – Examples -** ***map, filter, zip***

ReactiveX

- A library for composing asynchronous and event based programs by using observable sequences
- Created by Microsoft initially for the .NET platform
  By Erik Meijer
- Extends Observer Pattern to
  - Support sequence of data and/or events
  - Adds operators that allow you to compose sequences together declaratively
  - Abstracts away concerns around
    - Threading and Thread Safety
    - Concurrent data structures and non blocking I/O.
- RxJava is a port of Reactive Extensions created by Netflix

GOF Observer Pattern



Reactive Components

Emits Events

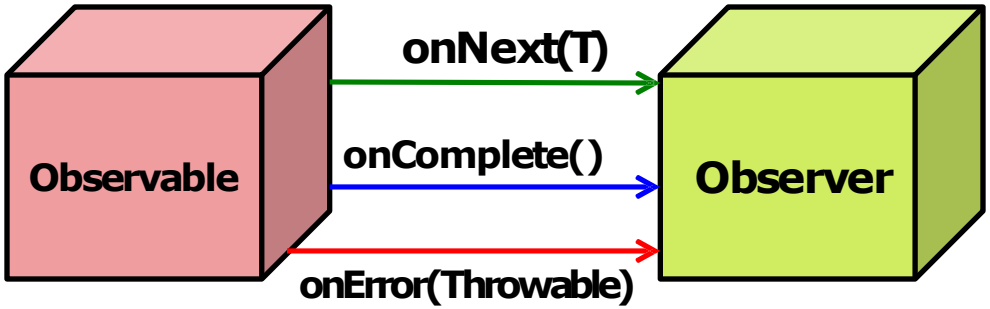
Observes Emitted Events

Observes and Emits Events

rx.Observable

```
public class Observable<T> { … }
```

- Emits zero or more values
- Life Cycle
  - Notifies Observer
    - onNext element
  - Completes with
    - onError
    - Or onCompletion



Observable Creation

```
Observable<Integer> o = Observable.create(new ObservableOnSubscribe<Integer >() { // Action
    @Override
    public void subscribe(ObservableEmitter<Integer> subscriber) throws Exception {
        // TODO Auto-generated method stub
        try {
            for (int i = 0; i < 10 ; i++) {
                subscriber.onNext(i);
            }

            subscriber.onComplete();

        }
        catch (RuntimeException e) {

            subscriber.onError(e);

        }
    }
});
```

Subscribing (Observer)

```
Observable<Integer>  o = Observable.create(…);

Subscription  s = o.subscribe(new  Observer<Integer>()  {
        @Override
        public void onNext(Integer  i)  {
          System.out.print(i);
        }

        @Override
        public void onCompleted()  {
          System.out.println("\nAll  Done!");
        }

        @Override
        public void onError(Throwable  t)  {
          t.printStackTrace();
        }
});
```

Observable from previous slide

Prints – 0 to 9!

Prints – All Done!

```
0123456789
All Done!
```

Operator	Description	Example
Observable.just(T)	Simple values wrapped	Observable.just("HelloWorld");
Observable.empty()	Empty Sequence that completes right away	Observable.empty();
Observable.from(Iterable<T>)	Sequence from an Iterable	Observable.from(Lists.newArrayList("A", "B"));
Observable.from(T[])	Sequence from Array	Observable.from(new Integer[] {0, 1, 2, 3, 4});
Observable.defer(ObservableFactory)	Defer emitting of items for each subscriber	Example shown later.
Observable.interval(long, TimeUnit)	Emit a sequence of numbers separated by an interval	Observable.interval(100, TimeUnit.MILLISECONDS);
Observable.range(start, count)	Emits a sequence of integers in a range	Observable.range(100, 20);
Observable.create(OnSubscribe<T>)	Most flexible and powerful	Observable.create(new OnSubscribe…());

Observable Vs Iterator

Event	Iterable (PULL)	Observable (PUSH)
Retrieve Data	Tnext();	onNext(T);
Discover the Error	throws Exception	onError(Throwable)
Complete	!hasNext()	onCompleted();