REACTIVE PROGRAMMING IN FINANCIAL DOMAIN

April 25, 2018

Twitter: @rashidnoorani
AGENDA

- Growth Of Internet By 2020
- Seen Problems?
- What Is Reactive Programming?
- How Reactive Programming Can Help?
  - The Reactive Manifesto
- Reactive Programming Use Case In Financial Domain
  - Learnings, Pros & Cons
- Q & A
GROWTH OF INTERNET BY 2020
SEEN PROBLEMS?

• Queue Overflow
• Resource Exhaustion
• Memory Limit
• Connection Limit
• Denial Of Service
THE REACTIVE MANIFESTO

VALUE

Responsive

FORM

Elastic

Resilient

MEANS

Message Driven

Source: https://www.reactivemanifesto.org/
REACTIVE GENERALLY MEANS

• Reactive Systems (architecture & design)
• Reactive Programming (declarative event-based)
• Functional Reactive Programming (FRP)
HOW REACTIVE SYSTEM CAN HELP?

• Handle “LIVE” streams of data
• Help exchange of stream data across an asynchronous boundary
• Handle backpressure
  ▪ Ensuring that receiver is not forced to buffer data
• Optimize
  ▪ Multiple CPUs/Hosts
LIBRARIES SUPPORTING REACTIVE PROGRAMMING
REACTIVEX IS A COMBINATION OF

- Observer Pattern
- Iterator Pattern
- Functional Programming
FR- LEGACY REPORTING SYSTEM
THE PROBLEM

• Under utilized resources
• Inter-dependency of data
  • Resulted in many serialization points
• Blocking calls impacting performance
• Backpressure
• Difficult to scale
THE PLAN

• Migrate to reactive architecture from top to bottom
• Use RxJava for reactive stream processing
• Migrate blocking REST end-points to micro services
• Replace callbacks with differed/asynchronous response
• Use RabbitMQ for messaging
• Reactively handle backpressure & Async processing
RXJAVA – INSPIRED BY REACTIVEX

```java
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```
EVOLUTION CONTINUES...
RABBITMQ
HANDLE REACTIVE BACKPRESSURE

- Buffering
- Batching
- Skipping
- Throttling
NEW SYSTEM
BENEFITS

• Increased utilization
  ▪ Multi-CPUs and/or hosts
• Reduced serialization points
  ▪ Increased performance
• Isolation of state, space, time, failure
  ▪ Decoupled in time allows concurrency
  ▪ Decoupled in space allows distribution and mobility
LEARNINGS

• Callback paradigm
  ▪ Change in mindset & learning curve
• Multi-threading is not reactive
• Can’t be done over night

Everything is data!
Want to hear a reactive joke?

I'll CALLBACK!
Thank You!
&
Questions?

Twitter: @rashidnoorani