



# Driven by Events

Stephan Wissel, HCL Labs

Paul Withers, HCL Labs

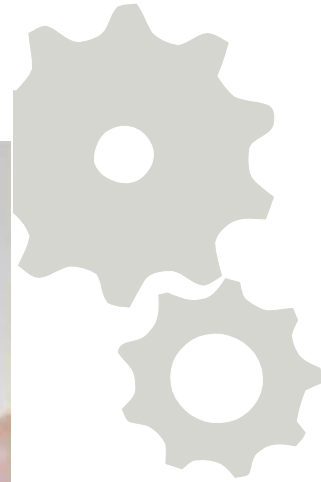
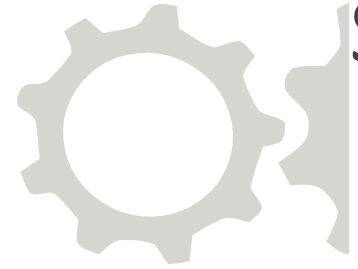
engage 



HCL Digital Solutions

## Stephan Wissel

- NotesSensei
- Not a champion
- Not a master
- Manila / Singapore





## Paul Withers

- Technical Architect, HCL Labs
- Lifetime IBM Champion
- Former HCL Grandmaster
- OpenNTF Board Member





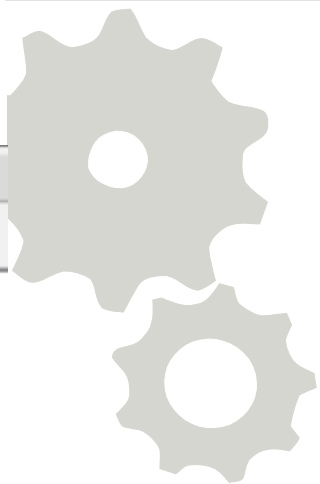
## The Problem...and Solutions

- Different chunks of code perform differently
- Single-threaded (synchronous) programming locks up processes and threads
- Multi-threading streamlines performance
  - Runnables and Callables
  - Imperative programming
  - Cannot apply back-pressure
- Event-driven code is similar but:
  - Can be reactive
  - Can apply back-pressure
  - Is harder to read



## LotusScript

- Linear
- Full-stack
- Modal
  - Client locks on Prompt()
- NotesAgent.runOnServer
  - Redirects to server thread
  - Client locks while agent runs
  - Linear progress through server agent
  - Redirects back to calling function and continues

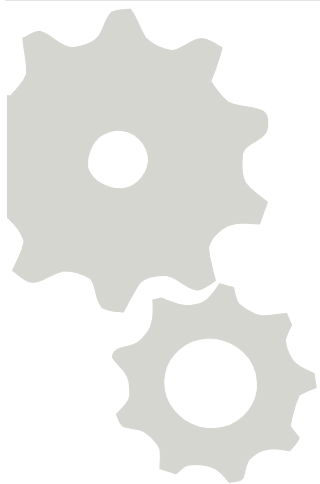


```
Hotspot : Click
Run Client LotusScript
Sub Click(Source As Button)
  Dim ws As New NotesUIWorkspace
  Dim uidoc As NotesUIDocument
  Dim doc As NotesDocument

  Call ws.Prompt([OK], "Continue?", "Do you want to continue")
  |
  Set uidoc = ws.CurrentDocument
  Set doc = uidoc.Document
  doc.completed="true"
  Call doc.save(True, False)
  Call ws.ReloadWindow
End Sub
```



## XPages

- Single-threaded
  - Linear
  - NotesAgent.run()
    - Redirects to LotusScript
    - XPages waits
    - Linear progress through server agent
    - Redirects back to SSJS / Java
- 



## How to Improve Performance?

- But this is all synchronous code
- To improve performance we must:
  - Use profiling to identify performance
  - Choose better-performing APIs
    - ViewNavigators
    - `ViewCollection.getFirstEntry() == null` instead of `ViewCollection.count()`
    - DQL instead of `db.search()`
- Or progress to asynchronous code...



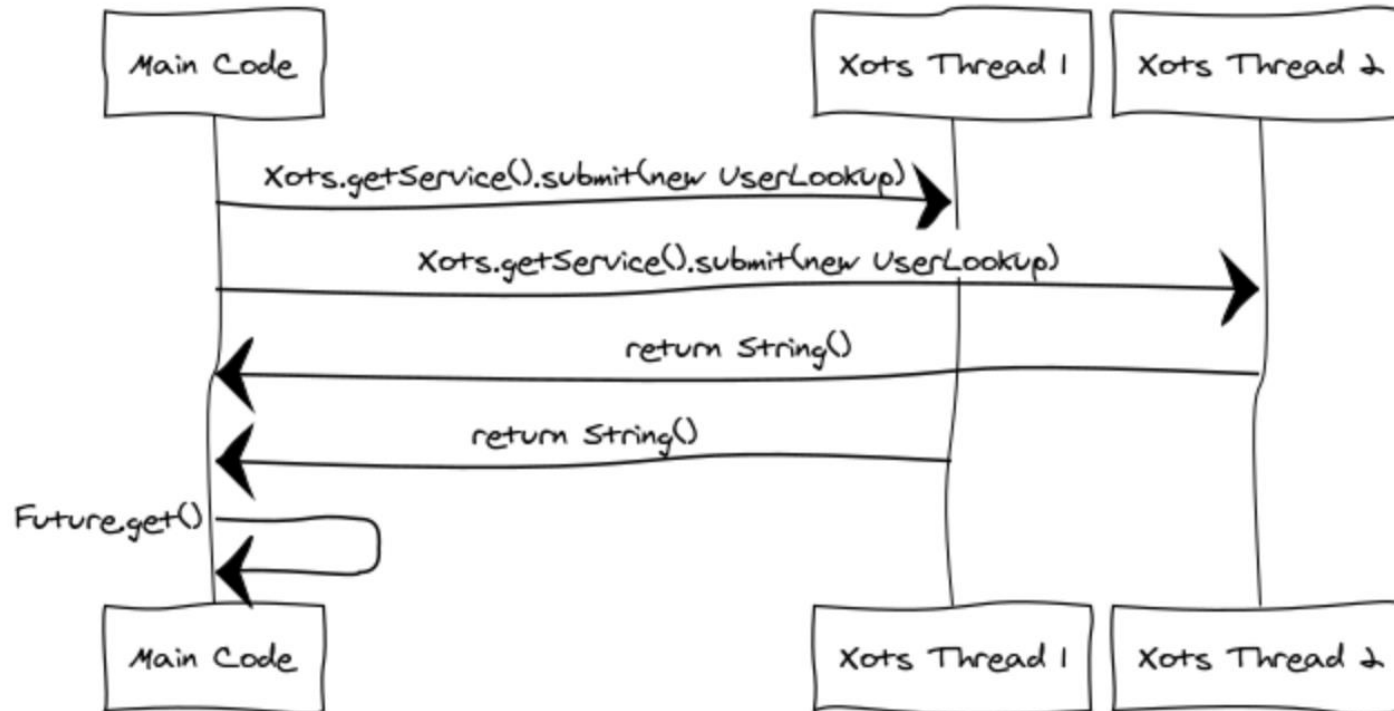
## Asynchronous in Domino

- Not possible in Formula Language
- Not possible in SSJS
- Agent.RunInBackgroundThread in LotusScript
  - But like a Java Runnable, no UI interaction
- [“Threads and Jobs”](#) project on OpenNTF
- [XOTS](#)
  - Runnables and Callables
  - Imperative programming



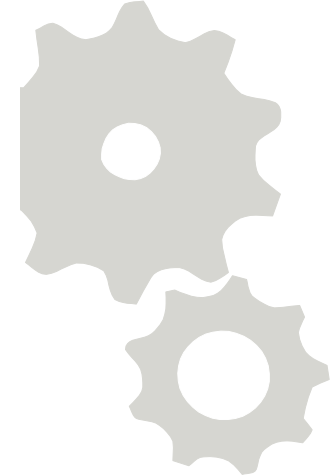
# Xots.getService().submit()

Callables





## The Problems

- Responsible for submitting Future / Promise
  - ExecutorService responsible for taking tasks off the queue
  - `get ()` method blocks while awaiting response(s)
  - Then code continues
  - Gets messy when nested
- 

## Asynchronous in JavaScript

- First came callbacks...and “callback hell”
- Promises made popular by jQuery Deferred Objects
  - Accepted into ECMAScript 2015 spec
  - Nested / chained promises gets hard to read (as we’ll see)
- Async / await introduced in NodeJS 7.6
  - Part of ECMAScript 2017 spec
  - Built on Promises
  - Async function = function that returns a Promise
  - await expects a Promise and unwraps it
    - Code looks synchronous, but runs asynchronous

# Promise

```
99  const getCustomer = function () {
100    return new Promise((resolve, reject) => {
101      axios.post(baseUrl + '/auth', identity)
102        .then(function (response) {
103          console.log(response.status);
104          if (response.status < 200 || response.status > 299) {
105            reject(new Error('Failed with status code: ' + response.status));
106          } else {
107            let bearer = response.data.bearer;
108            axios.get(baseUrl + '/lists/Customers?db=demo-small', { headers: { "Authorization": "Bearer " + bearer }})
109              .then(function (response) {
110                if (response.status < 200 || response.status > 299) {
111                  reject(new Error('Failed with status code: ' + response.status));
112                } else {
113                  resolve(response.data);
114                }
115              })
116            }
117          });
118    });
119  };

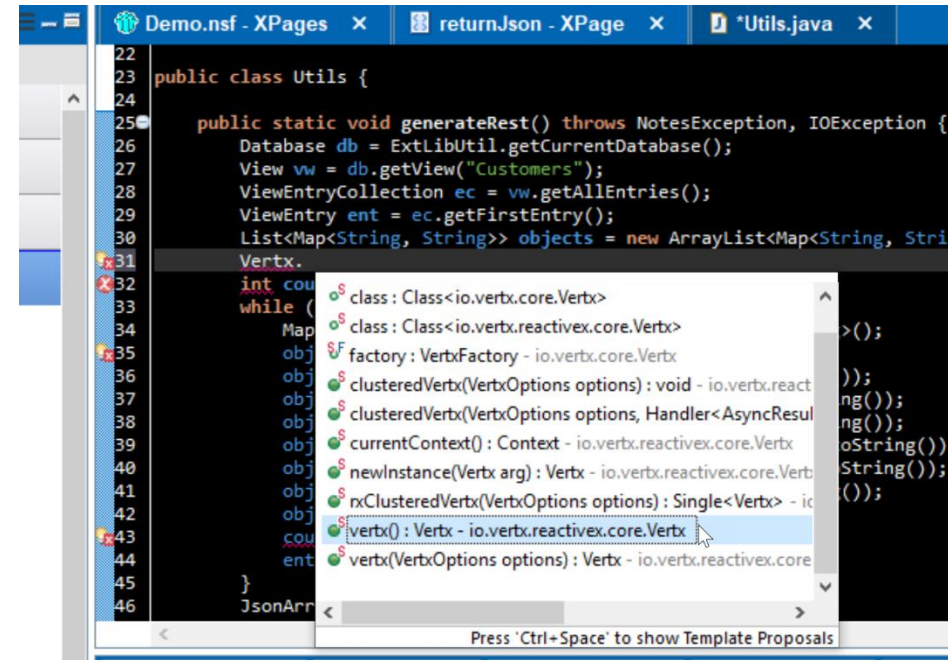
```

# Async / Await

```
21  async handle(handlerInput) {
22      let outputSpeech = 'This is the default message.';
23
24      let response = await getCustomer();
25      try {
26          let items = response.length;
27          let elemNum = Math.floor(Math.random() * items);
28          let firstItem = response[elemNum];
29          let elemForResponse = (elemNum + 1).toString();
30          outputSpeech = `In the Customers view there are ${items} customers. Your random customer is number ${elemForResponse},
31      } catch (err) {
32          //set an optional error message here
33          console.log(err);
34          outputSpeech = err.message;
35      }
36
37      return handlerInput.responseBuilder
38          .speak(outputSpeech)
39          .withSimpleCard("Project Keep Demo Card", outputSpeech)
40          .getResponse();
41
42  },
43  };
```

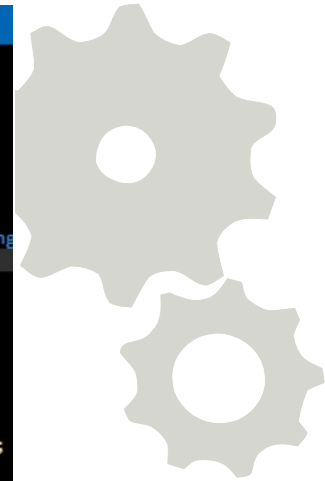
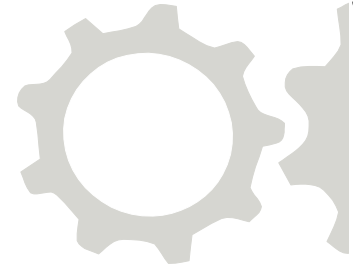
# What is Vert.x?

- Polyglot toolkit framework
  - Java, JS, Groovy, Ruby, Scala, Kotlin
- Modular and Lightweight
  - Vert.x core is 650kB
- Fast
- Flexible for everything from HTTP/REST microservices to sophisticated web applications
- Integrated into other frameworks, like Quarkus



The screenshot shows an IDE with a Java file named 'Utils.java'. The code includes a 'generateRest' method that uses Vert.x to interact with a database. An autocomplete menu is open over the 'Vertx' variable, showing suggestions like 'io.vertx.core.Vertx', 'io.vertx.reactivecore.Vertx', and 'io.vertx.reactivecore.VertxFactory'. The code snippet is as follows:

```
22  
23  
24  
25 public static void generateRest() throws NotesException, IOException {  
26     Database db = ExtLibUtil.getCurrentDatabase();  
27     View vw = db.getView("Customers");  
28     ViewEntryCollection ec = vw.getAllEntries();  
29     ViewEntry ent = ec.getFirstEntry();  
30     List<Map<String, String>> objects = new ArrayList<Map<String, String>  
31     Vertx.  
32     int coun  
33     while (  
34     Map  
35     obj  
36     obj  
37     obj  
38     obj  
39     obj  
40     obj  
41     obj  
42     obj  
43     cou  
44     ent  
45  
46     }  
    jsonArray
```



## Who's using Vert.x?

vmware®  
airwatch®

 **BOSCH**  
Invented for life

CYANOGEN

 DEUTSCHE BÖRSE  
GROUP

 Fraunhofer  
IGD

 **GROUPON**

hopscotch

**hulu**

 LIFERAY

 **RBS**<sup>TM</sup>  
The Royal Bank of Scotland

 **Red Hat**

SWISS POST 

 swisscom

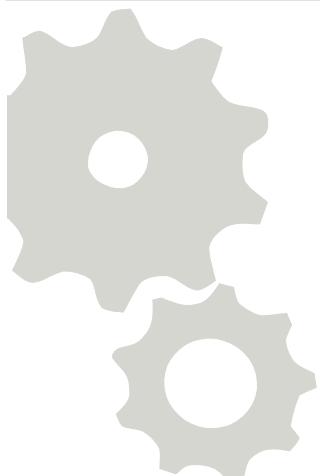
**TESCO**

ticketmaster®

 zalando



## Vert.x is Event Driven

- Vert.x is *event driven* and *non blocking*
  - Vert.x is multi-threaded (like Domino but not like Node.JS)
  - Means lots of concurrency with few threads
  - Uses Event Loop with multiple threads (Multi-Reactor)
    - Number of threads depends on CPU
  - *Blocking* code offloaded to worker threads
    - 20 threads by default
  - Verticles are actors on threads
- 

```
Thread vertx-eventloop-thread-3 has been blocked for 20458 ms
```



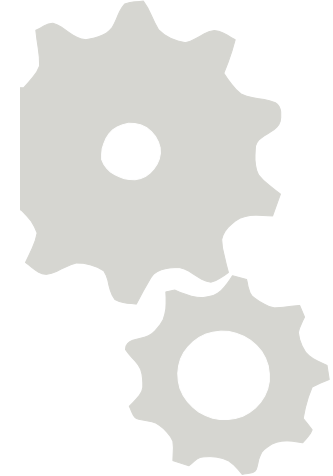


## The Event Bus

- Communication between verticles via *Event Bus*
- One Event Bus per Vert.x instance
- Can be clustered across Vert.x instances
- Can allow JavaScript and Java verticles to communicate with one another
- Supports different messaging options:
  - Publish/Subscribe
  - Point-to-point messaging
  - Request-response messaging

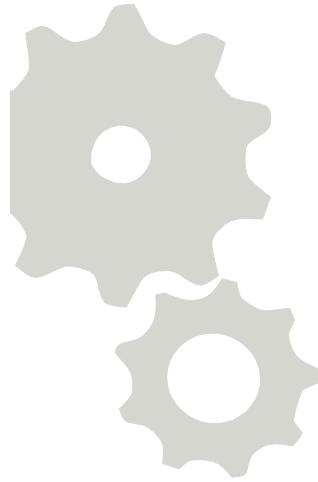
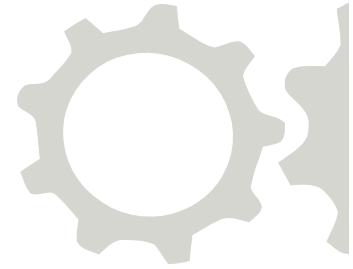
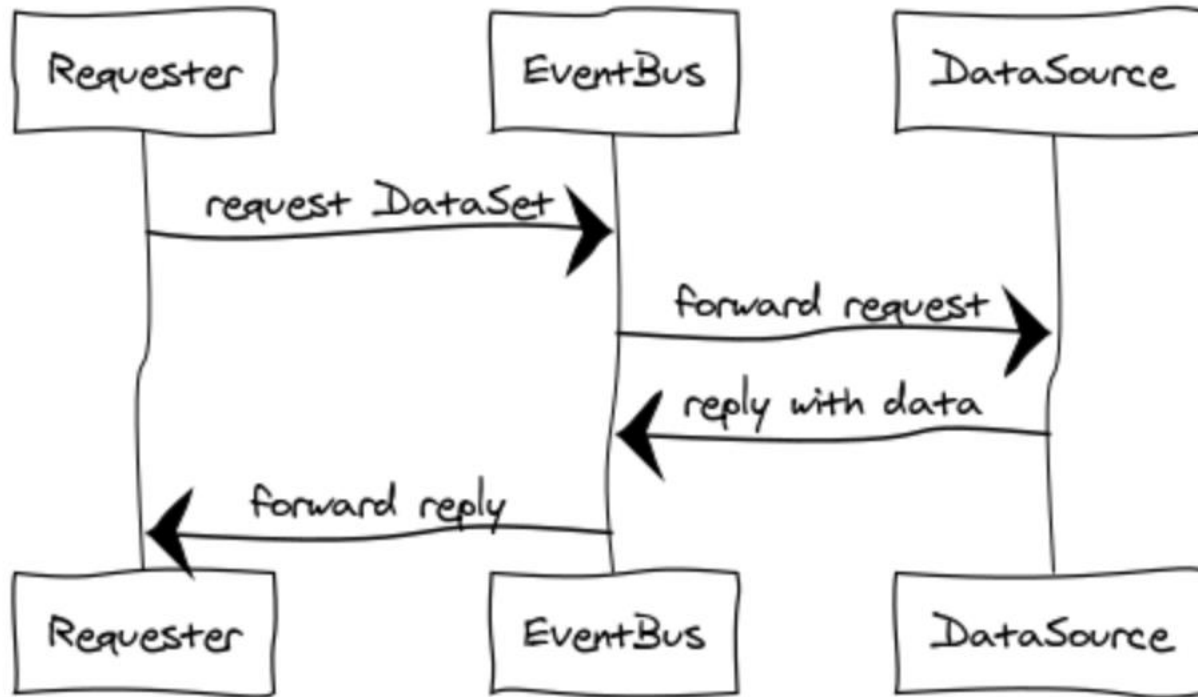


## The Event Bus

- Messages sent to an address, e.g. “keep.request.fetchviews”
  - Handlers are registered to listen for an address
  - Messages can be:
    - Strings
    - Buffers
    - JsonObjects (in-built JSON support built on Jackson)
    - Any other object for which a codec is registered
      - Message codecs have a name and define the class
- 

Event Bus

Vert.x EventBus





## Domino's "Distributed Message Bus"

- Messages = email
- Sends message to an address
- Mail-in database receives email
- "After mail arrives" agent processes message
- However, no in-built communication back
  - You could have a mail-in database at the other end as well
  - "After mail arrives" finally sends email back



## HTTP “Message Bus”

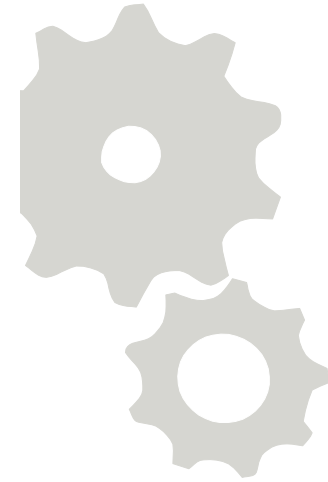
- Browser sends request
- Request has an *address* (URI)
- Sends a *message* (body + headers)
- Subscribes for response
- Receives response from server and processes it

engage 



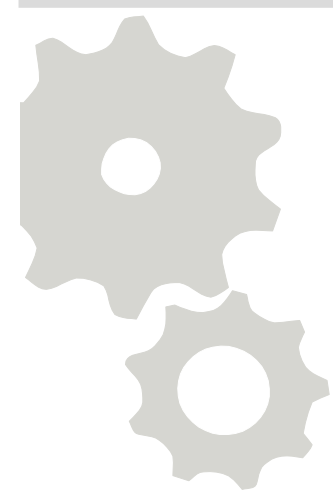
HCL Digital Solutions

So What?





## ReactiveX

- Reactive eXtensions
  - Asynchronous programming with observable streams
  - Polyglot – RxJava, RxJS, RxGroovy, RxCpp, RxPY, RxSwift, RxScala
    - RxJava 2 is latest Java version
  - Combines Observer pattern, Iterator pattern and functional programming
- 

## We use ReactiveX







## Project Reactor

- Spring uses Project Reactor
  - Java 8+
  - Doesn't have to support Android
  - Directly interacts with Java functional API, Completable Future, Streams
  - Based on joint reactive research effort also implemented by RxJava 2
  - Started as Rx lite but now almost the same

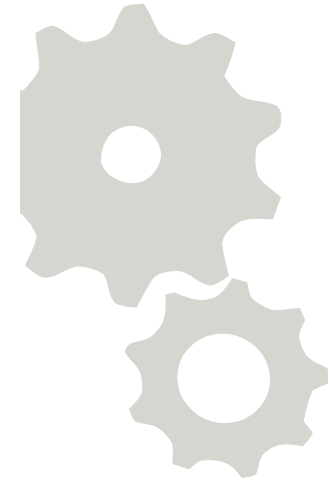
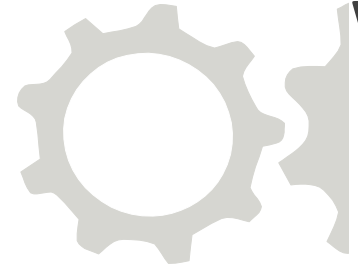
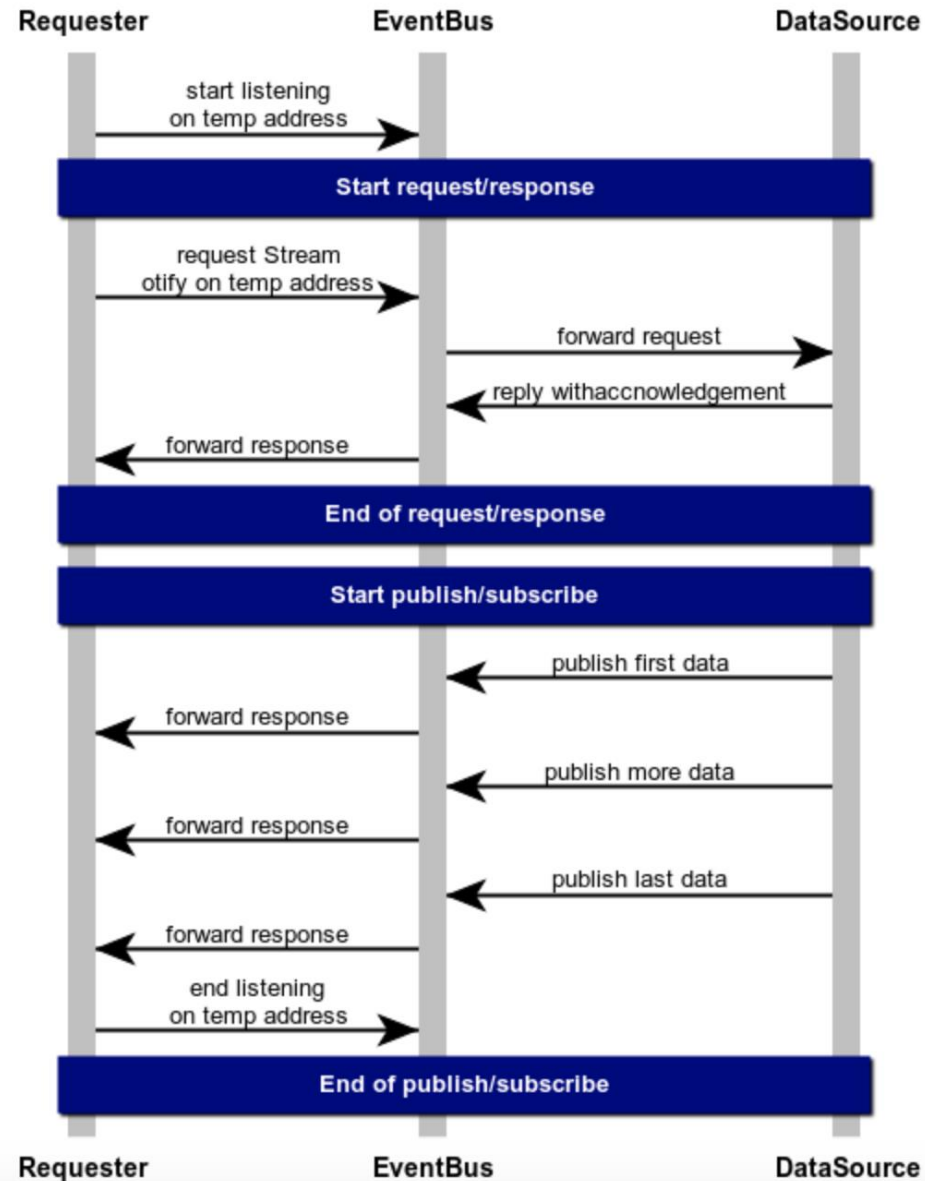


## Spring

```
45 String authToken = (String) UI.getCurrent().getSession().getAttribute("authToken");
46 WebClient client = getClient(authToken);
47 try {
48     Mono<Map<String, State>> respStates = client
49         .get()
50         .uri("/lists/AllStates?db=contacts")
51         .retrieve()
52         .bodyToFlux(State.class)
53         .collectMap(State::getKey);
54     states = respStates.block();
55 } catch (Throwable t) {
```

## What Does It Mean?

- Send request
- Request acknowledged
- Publish data
- Publish more data
- Complete



## Give Us The Stats!

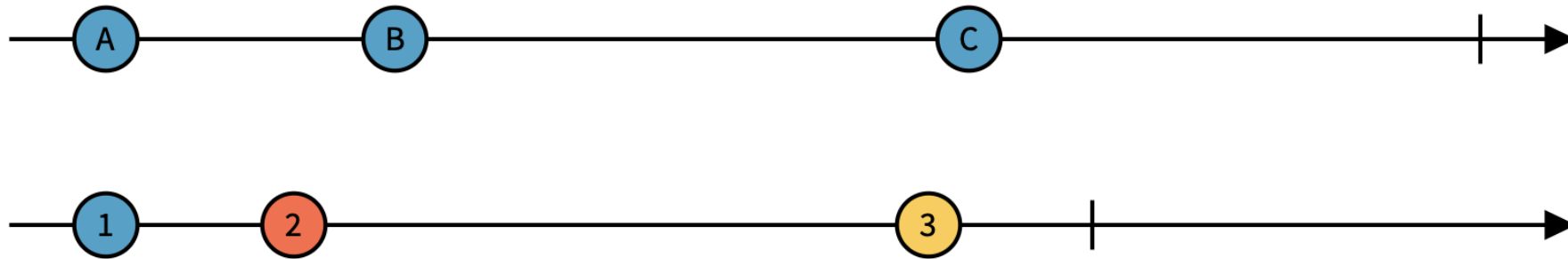
REST Method	Worst	Best	Average
XAgent View AllEntries	1.971 secs	1.519 secs	<b>1.6582 secs</b>
XAgent ViewNavigator	1.562 secs	1.397 secs	<b>1.4598 secs</b>
LS Agent View All Entries	1.557 secs	1.325 secs	<b>1.424 secs</b>
LS Agent ViewNavigator	1.14 secs	1.274 secs	<b>1.1908 secs</b>
<b>Project Keep API</b>	<b>0.641 secs</b>	<b>0.537 secs</b>	<b>0.6068 secs</b>



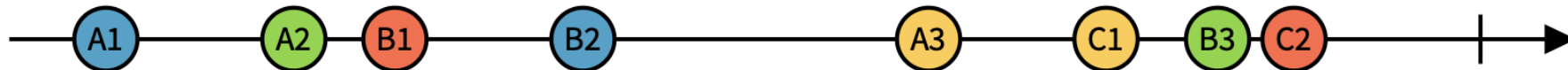
## Advantages

- Calling code can start work quicker
- Can perform additional processes
  - Merge
  - Order
  - Filter
  - Count

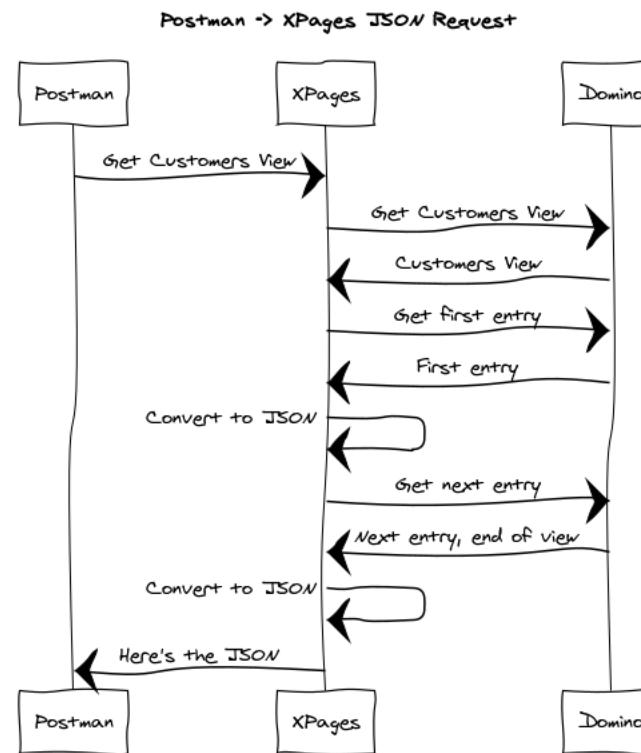
# RxMarbles



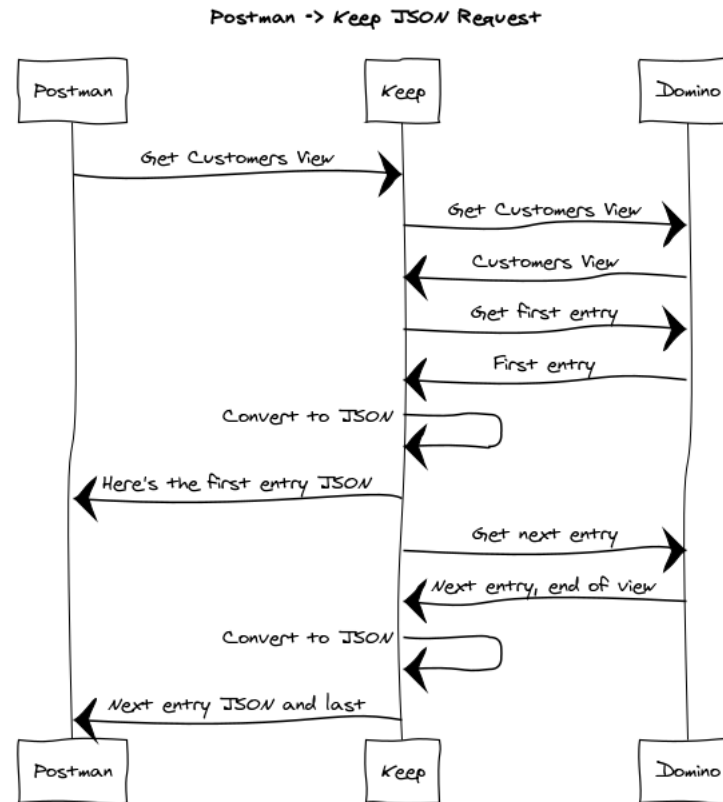
```
obs1$.mergeMap(() => obs2$, (x, y) => "" + x + y, 2)
```



# What XPages / LS -> Postman Does



# What Keep -> Postman Does







## Thank You and Useful Links

- [ReactiveX](#) - various languages
- [RxJS Marbles](#) - interactive reactive diagrams
- [Going Reactive with Eclipse Vert.x and RxJava](#)
- [Vert.x and Reactive](#)
- [Reactive programming in Redux](#)
- [Development of Reactive Applications with Quarkus](#) – Niklas Heidloff
- [Domino and Synchronous / Asynchronous Processing](#)
- [A Streaming Pattern for the Vert.x Event Bus](#)
- [Project Reactor](#)

# Why No Comparison to App Dev Pack

