RxJava, Ratpack, Couchbase

Laurent Doguin
Couchbase Developer Advocate
@ldoguin

$whoami

Laurent Doguin

Couchbase Developer Advocate

@ldoguin | laurent.doguin@couchbase.com

2

Ratpack ?

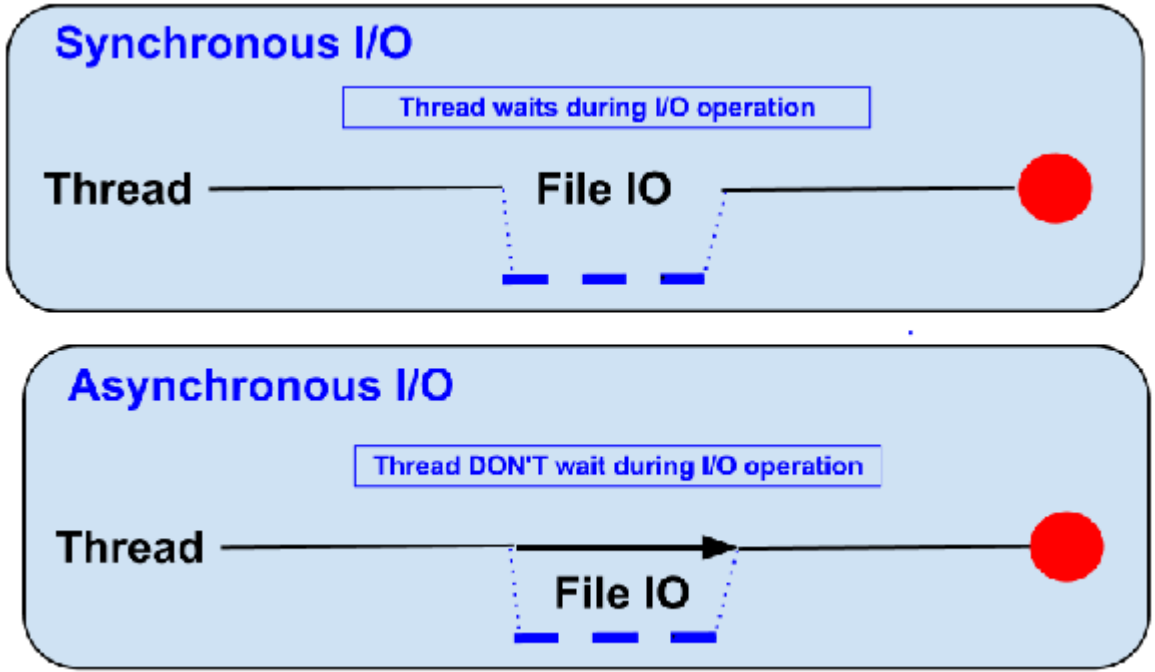
Not an actor/singer club

Netty based, fullstack, non-blocking, web framework

In a Nutshell…

- Asynchronous
- Non-Blocking
- Build on top of Netty
- Unopinonated
- Think about NodeJS or Vert.X
- Promises
- Java 8
- Top Notch Gradle integration
- Strong Groovy Support
- Great for I/O Bound applications

Sync, blocking V.S. Async, non-blocking



Handlers

- Core of Ratpack
- This is where the magic happens
- RatpackServer.*start(*
  - *server -> server.handlers(*
    - *chain -> chain.all(*
      - *ctx -> ctx.render("Hello World")*
    - *)*
  - *)*
- *);*

Plugins and Modules

- GUICE based Modules system
- Can be switched to others
  - Spring Boot support
- Module can be as low level as DI Framework
- Think of it like decorations

Why Ratpack?

- If you spend to much time on I/O
- Code in an async non-blocking style with a determinist/synchronous execution style

RxJava

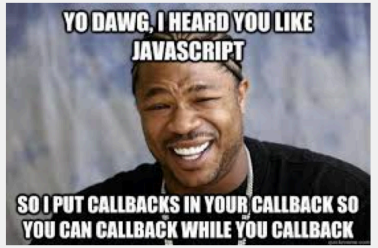
Couchbase

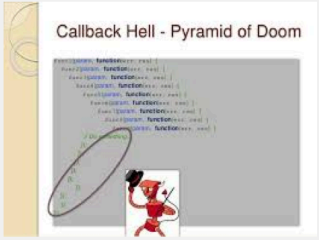
Why RxJava

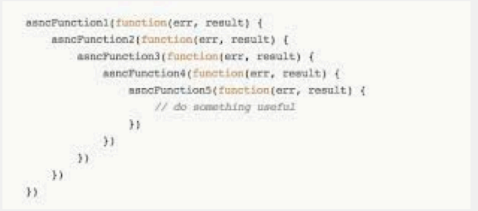
- Blocking is Evil
- Async is good, Better when
  - Reactive
  - Parallelizable
  - Composable
  - Readable

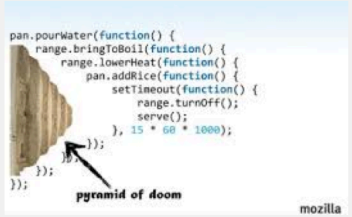
But I can do this already

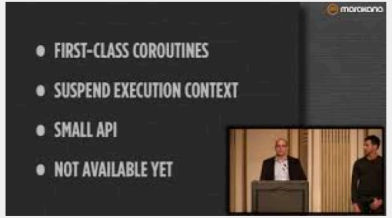
- Callbacks?
  - No composition
  - Callback Hell
- Futures<T>?
  - ‘lol’
  - Too easy to block
  - Hard to compose

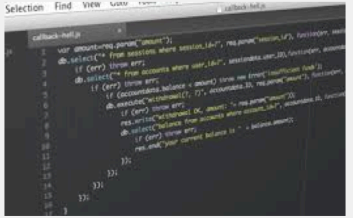


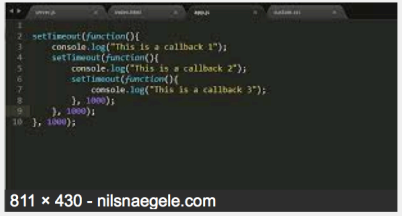


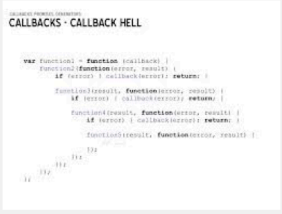


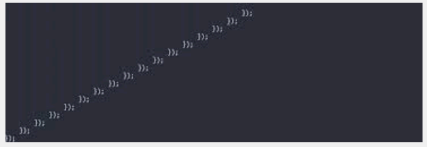












RxJava ?

- Netflix OpenSource
- From Iterator – Iterable (Pull)
- To Observable – Observer(Push)

- Allow to Compose
- Asynchronous code
- Based on events
- Using Observable Sequences

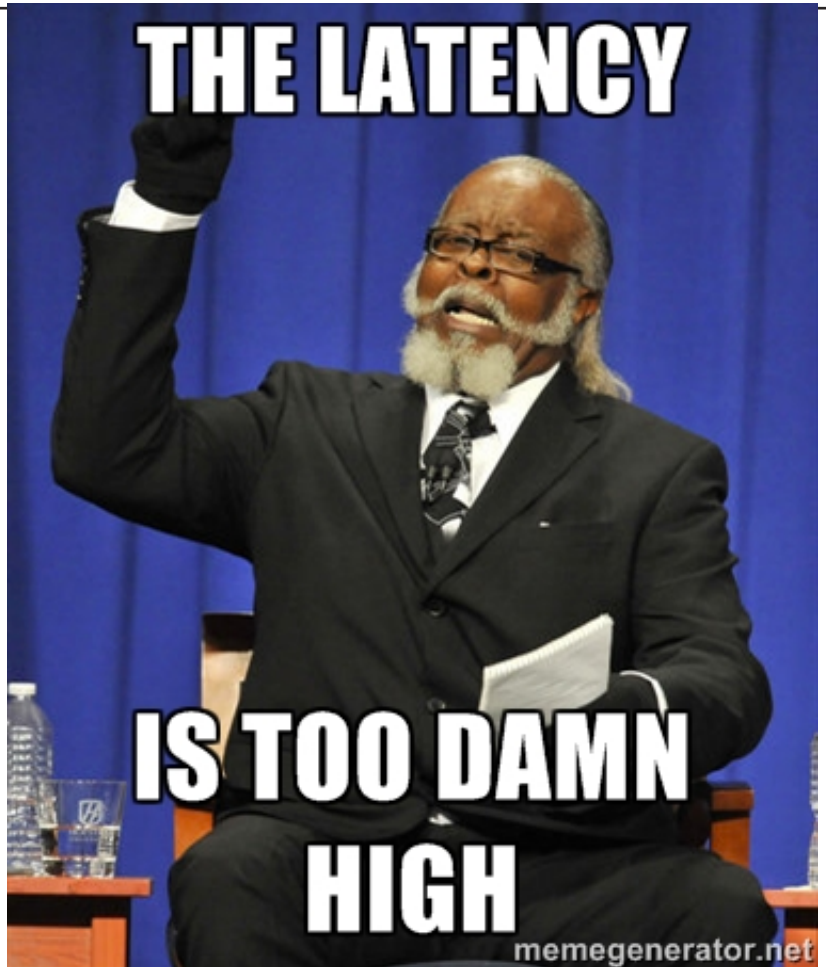
NETFLIX
OSS

Why Reactive?

- New challenges
  - React to user load
  - React to failure
  - Be responsive under load and failure

- Need new solutions
  - Decoupled, event-driven architectures
  - Optimal resource utilization

Why Reactive?



17

RxJava 101

A Gentle Introduction

Couchbase

RxJava: Introduction

- Java implementation for Reactive Extensions
  https://github.com/ReactiveX

- A library to compose **asynchronous** and **event-driven** programs through observable sequences.

	single	multiple
sync	T	Iterable<T>
async	Future<T>	Observable<T>

RxJava: Introduction

- Observables are the duals of Iterables
- They describe both Latency and Error side effects.

event	Iterable<T> (pull)	Observable<T> (push)
data retrieval	T next()	onNext(T)
error discovery	throws Exception	onError(Exception)
completion	returns	onCompleted()

Consuming Observables

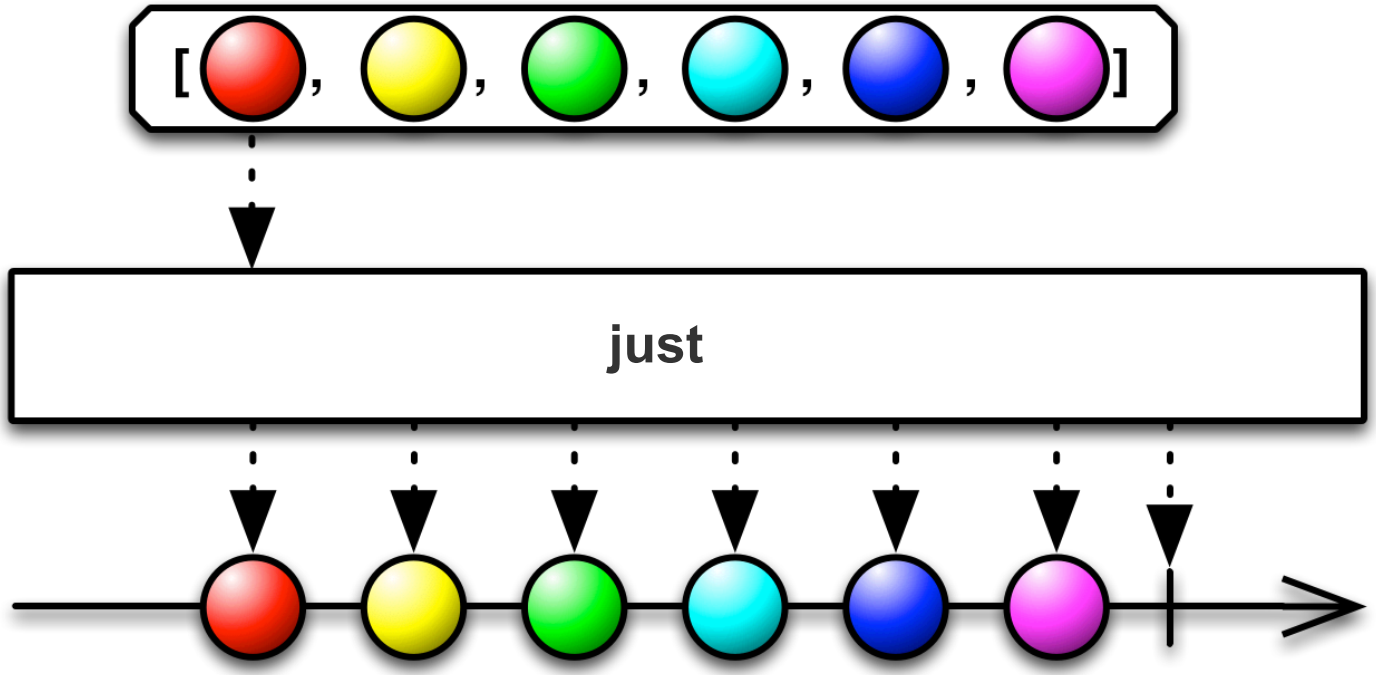
- The Observer subscribes and receives events.
- A cold Observable starts when subscribed.
- onNext can be called o..N times

```
bucket
    .async()
    .get("doc")
    .subscribe(new Observer<JsonDocument>() {
        @Override
        public void onCompleted() {
            System.out.println("Done");
        }

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

        @Override
        public void onNext(JsonDocument doc) {
            System.out.println("Found: " + doc);
        }
    });
```

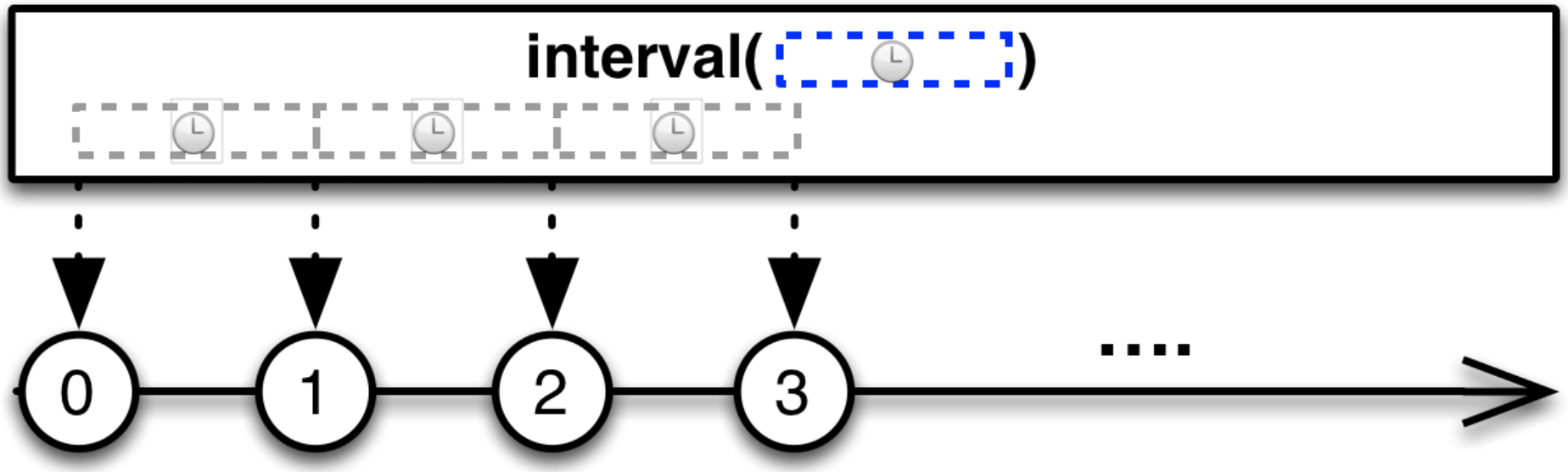
RxJava: Creating Observables



RxJava: Creating Observables

```
1  Observable
2      .just("A", "B", "C")
3      .subscribe(new Action1<String>() {
4          @Override
5          public void call(String s) {
6              System.out.println("Got: " + s);
7          }
8      });
```

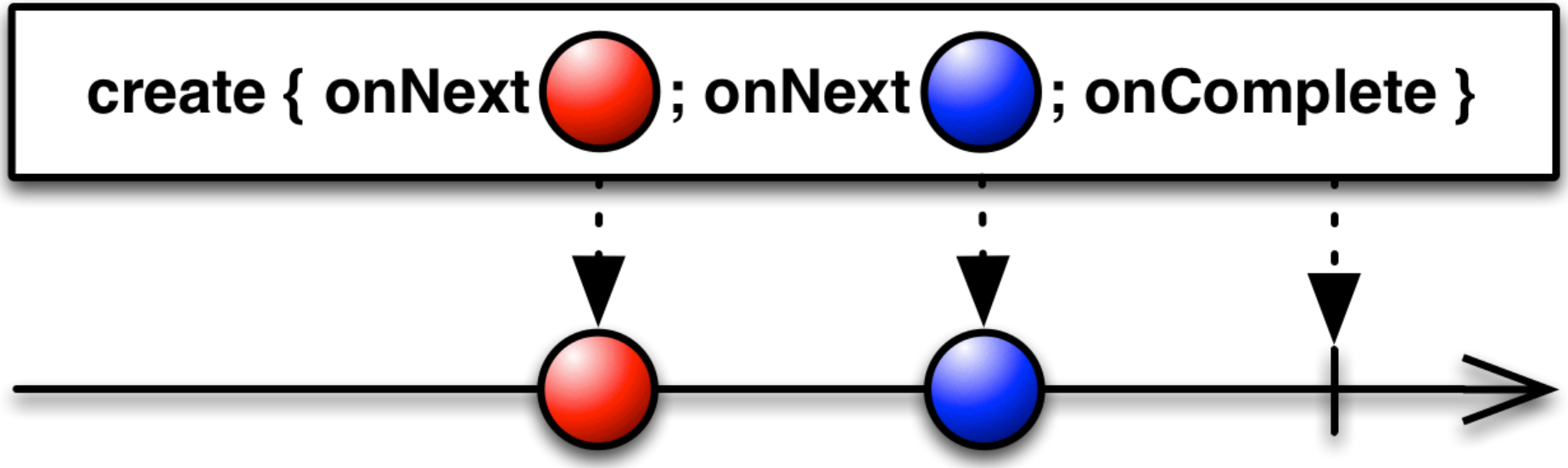
RxJava: Creating Observables



RxJava: Creating Observables

```
1  // 0 ..... 1 ..... 2 ....
2  Observable
3      .interval(2, TimeUnit.SECONDS)
4      .subscribe(System.out::println);
```

RxJava: Creating Observables

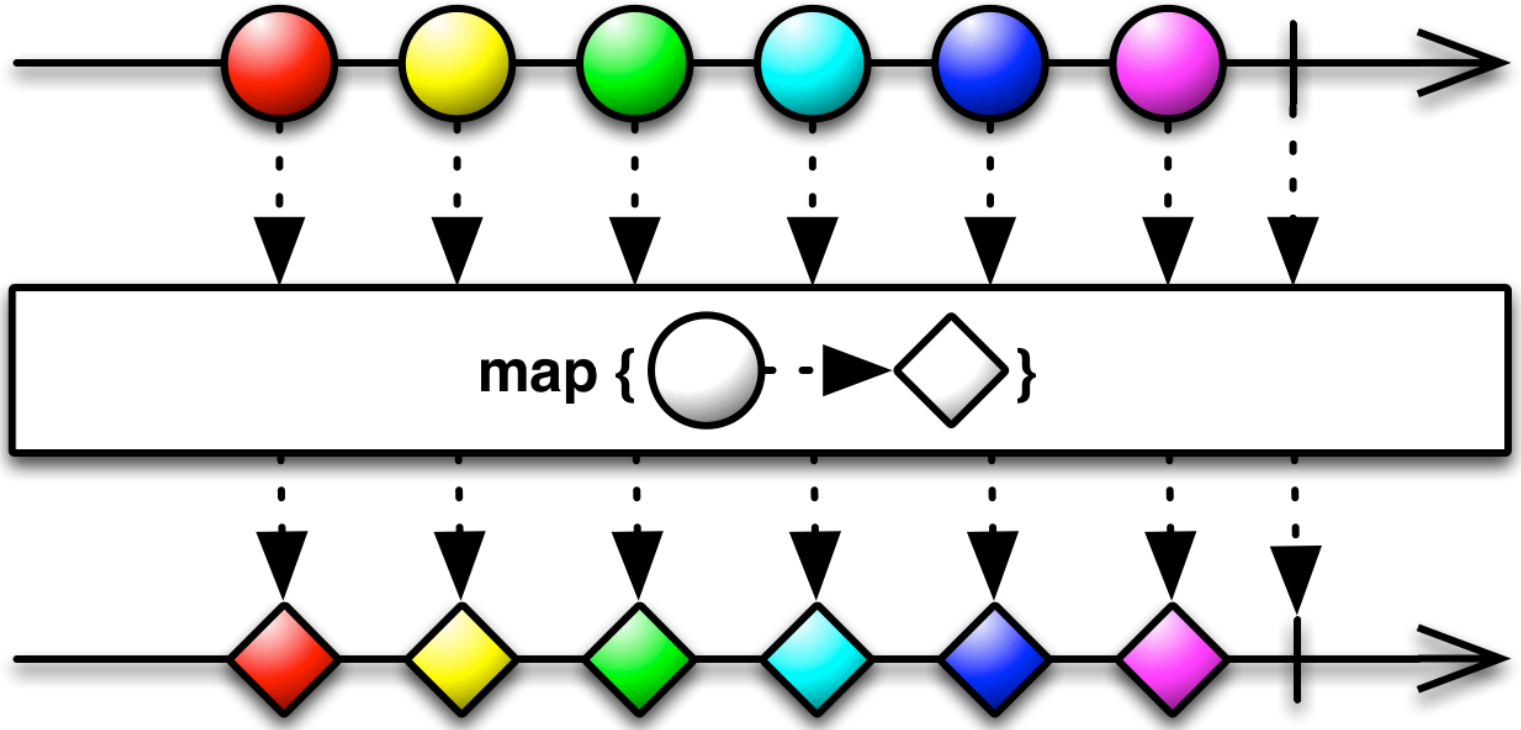


26

RxJava: Creating Observables

```
1  Observable.create(new Observable.OnSubscribe<String>() {
2      @Override
3      public void call(Subscriber<? super String> subscriber) {
4          try {
5              subscriber.onNext("Hello Subscriber!");
6              subscriber.onCompleted();
7          } catch(Exception ex) {
8              subscriber.onError(ex);
9          }
10     }
11 }).subscribe();
```

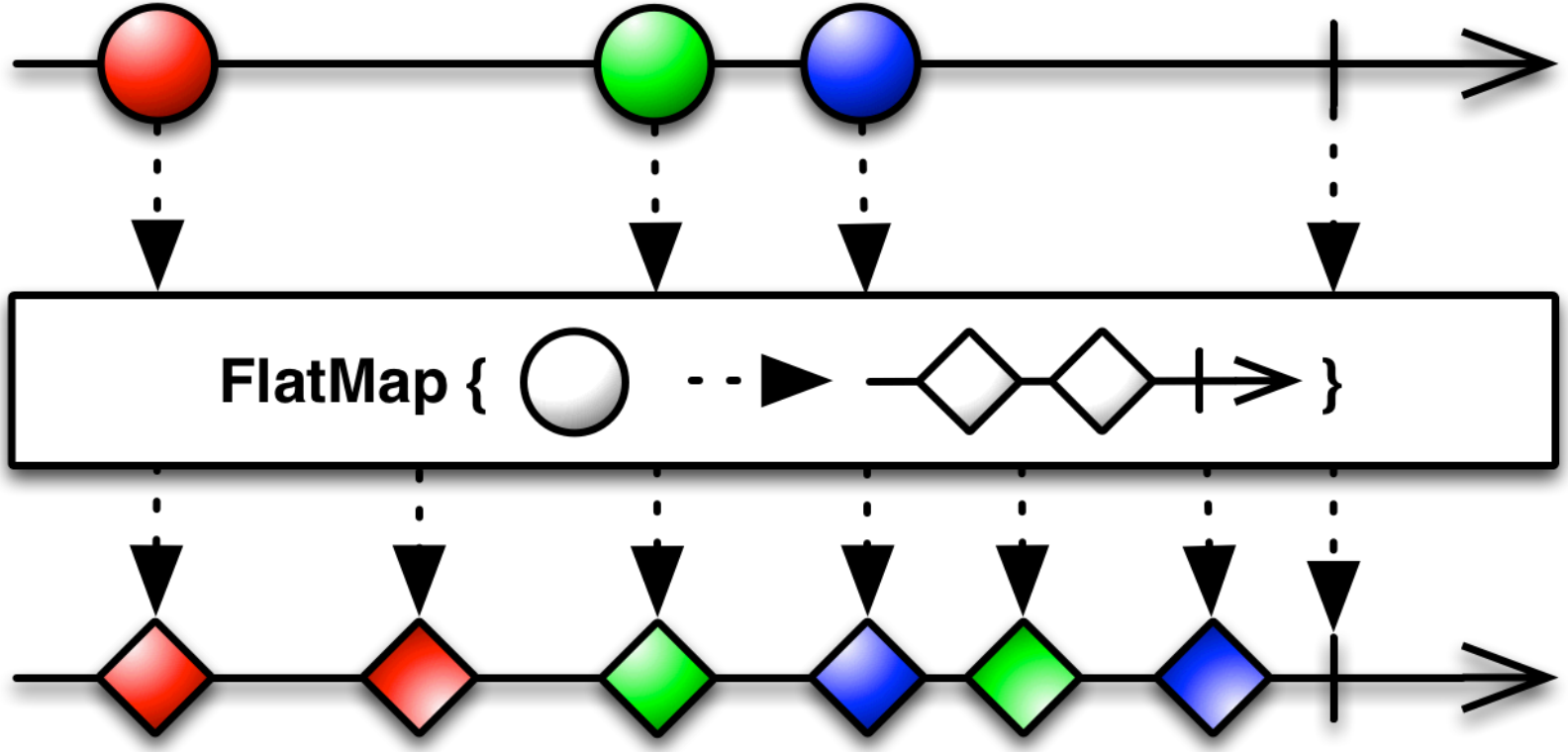
RxJava: Transforming Observables



RxJava: Transforming Observables

```
1  bucket
2      .async()
3      .get("doc")
4      .map(doc -> doc.content())
5      .subscribe(new Action1<JsonObject>() {
6          @Override
7          public void call(JsonObject content) {
8              System.out.println("Found: " + content);
9          }
10     });
```

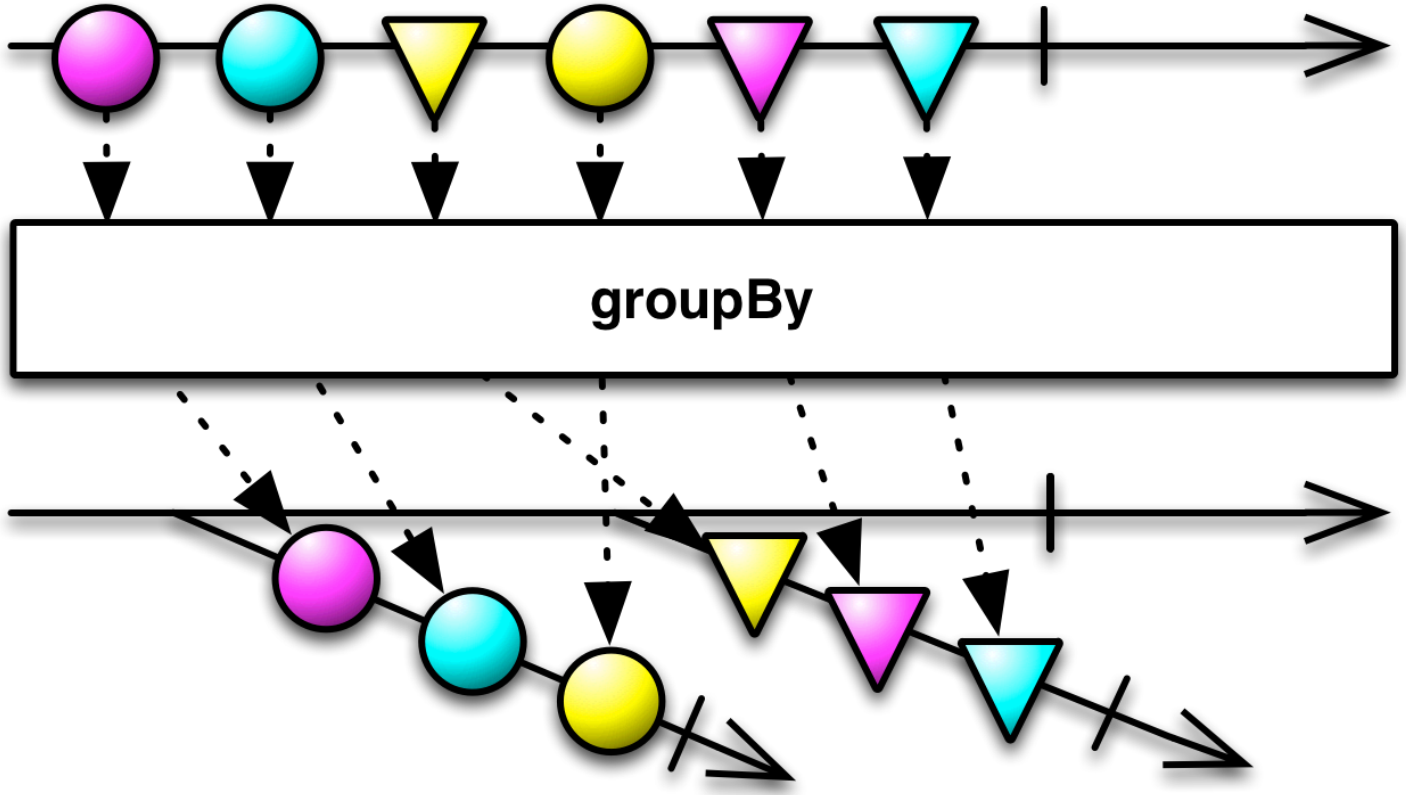
RxJava: Transforming Observables



RxJava: Transforming Observables

```
bucket
    .query(ViewQuery.from("beers", "by_name").limit(100)) // query
    .flatMap(AsyncViewResult::rows) // stream each row
    .flatMap(AsyncViewRow::document) // grab doc for each row
    .filter(doc -> doc.content().getDouble("abv") > 5.0) // filter beer by abv
    .count() // count all filtered beers
    .timeout(10, TimeUnit.SECONDS) // overall timeout
    .subscribe(System.out::println); // print
```

RxJava: Transforming Observables

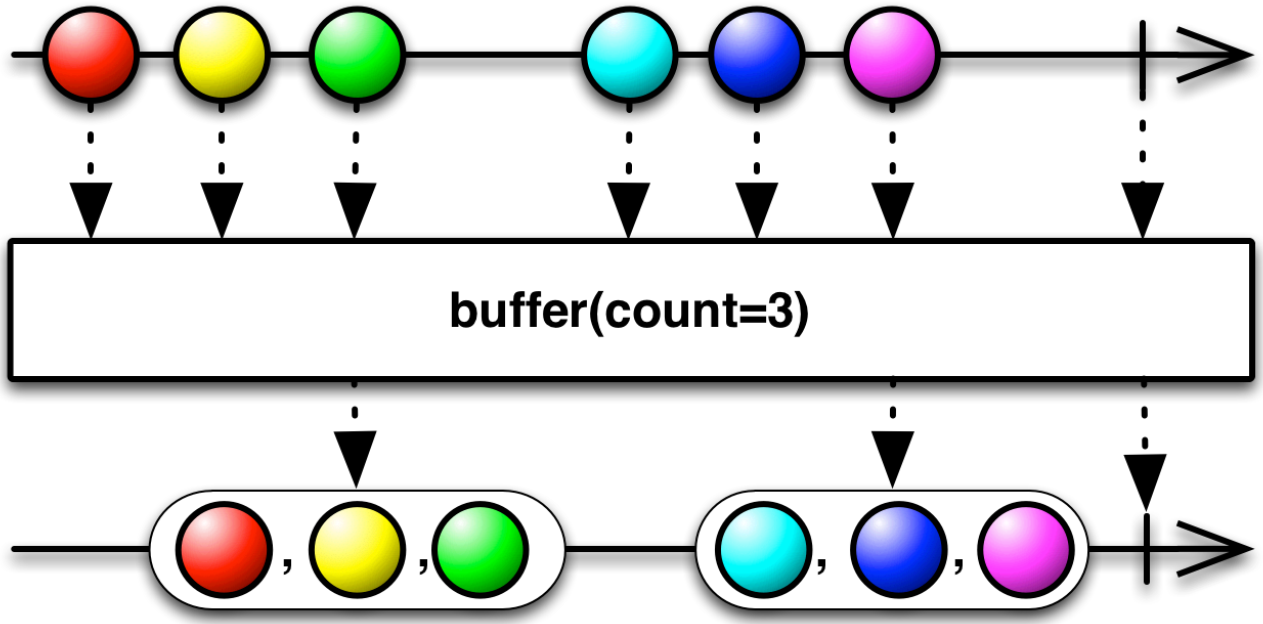


RxJava: Transforming Observables

```
bucket
    .query(select("*").from("beer-sample"))
    .flatMap(AsyncQueryResult::rows)
    .groupBy(result -> result.value().getString("type"))
    .subscribe(grouped ->
          grouped
              .count()
              .subscribe(cnt -> System.out.println(grouped.getKey() + ": " + cnt))

    );
```

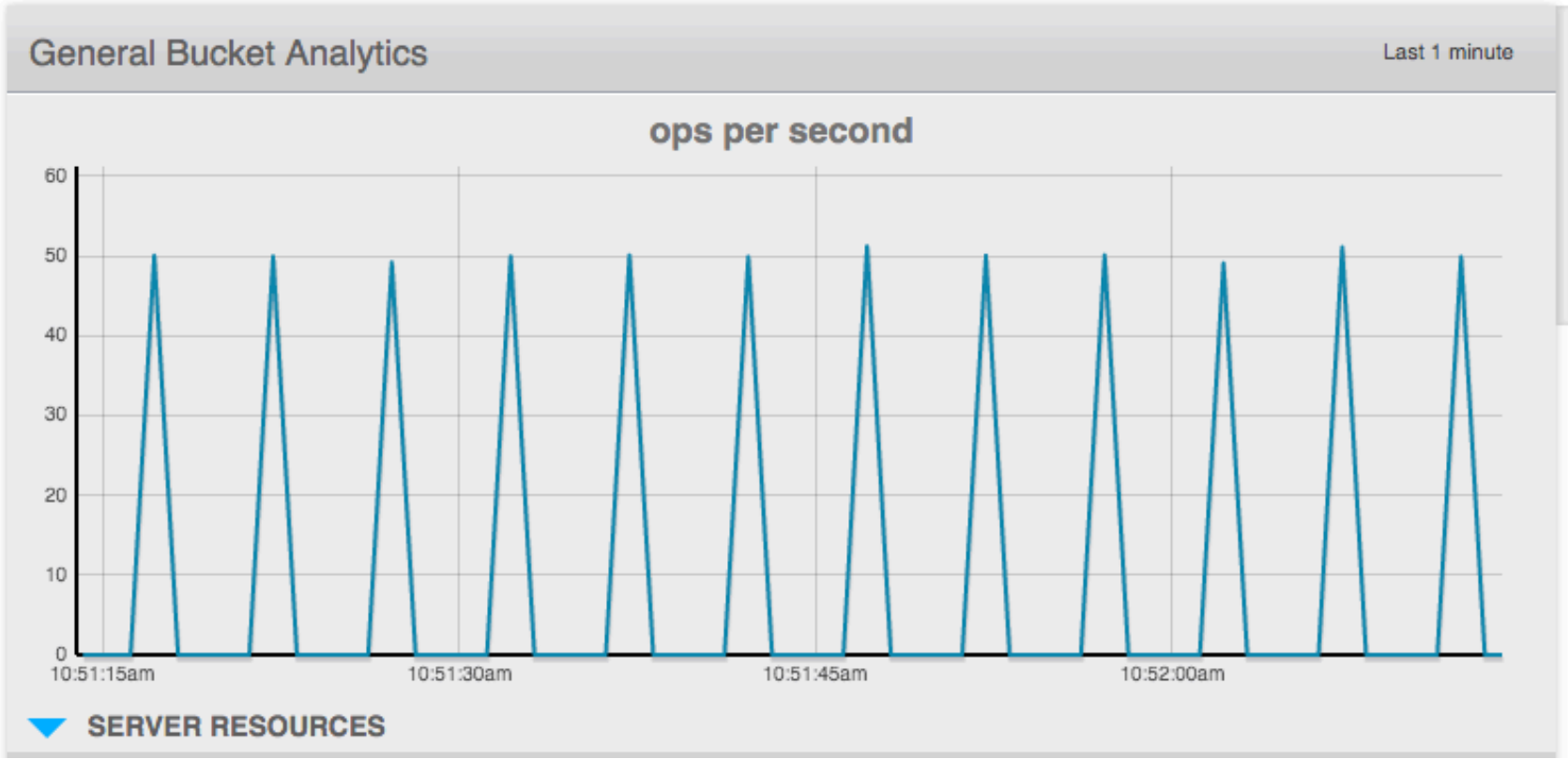
RxJava: Transforming Observables



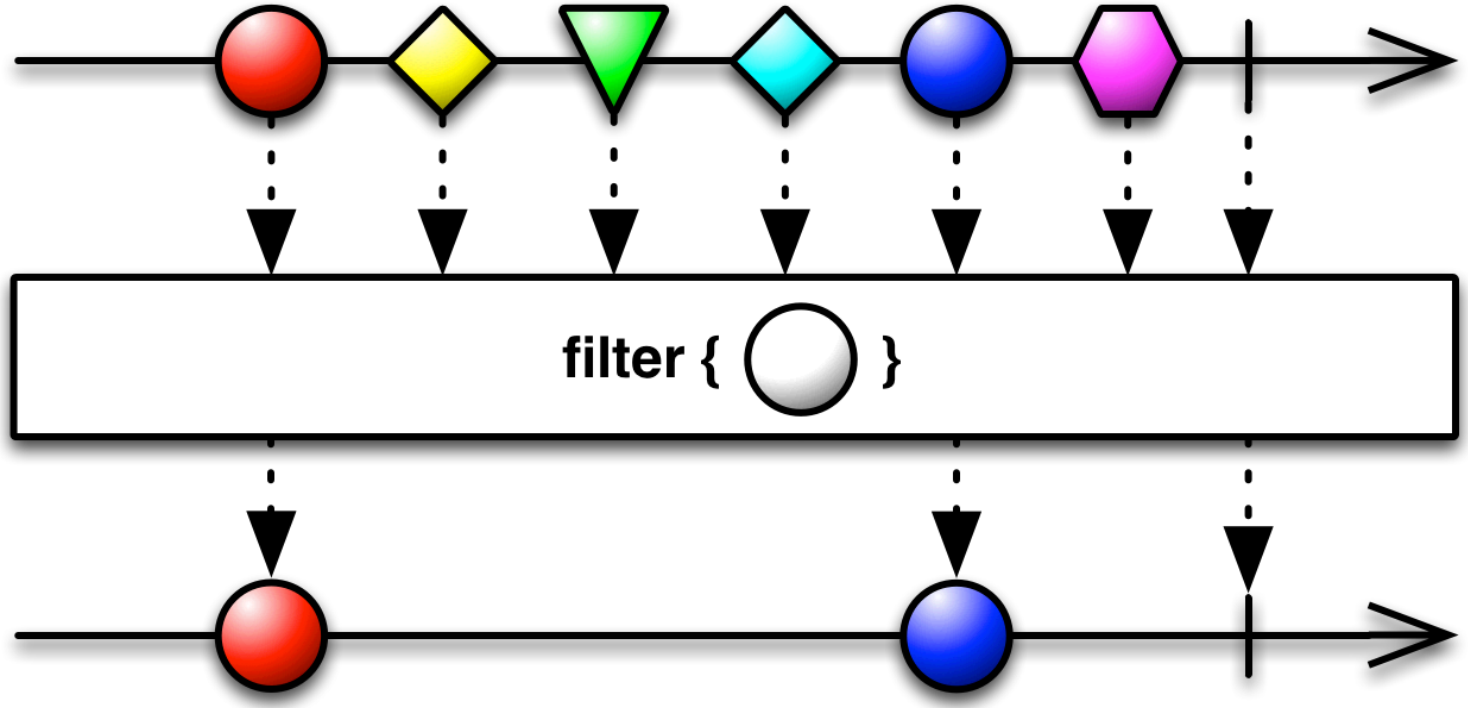
RxJava: Transforming Observables

```
1  Observable
2      .interval(100, TimeUnit.MILLISECONDS)
3      .buffer(5, TimeUnit.SECONDS)
4      .flatMap(ids -> Observable.from(ids).map(i -> "id::" + i))
5      .flatMap(bucket.async()::get)
6      .toBlocking()
7      .last();
```

RxJava: Transforming Observables



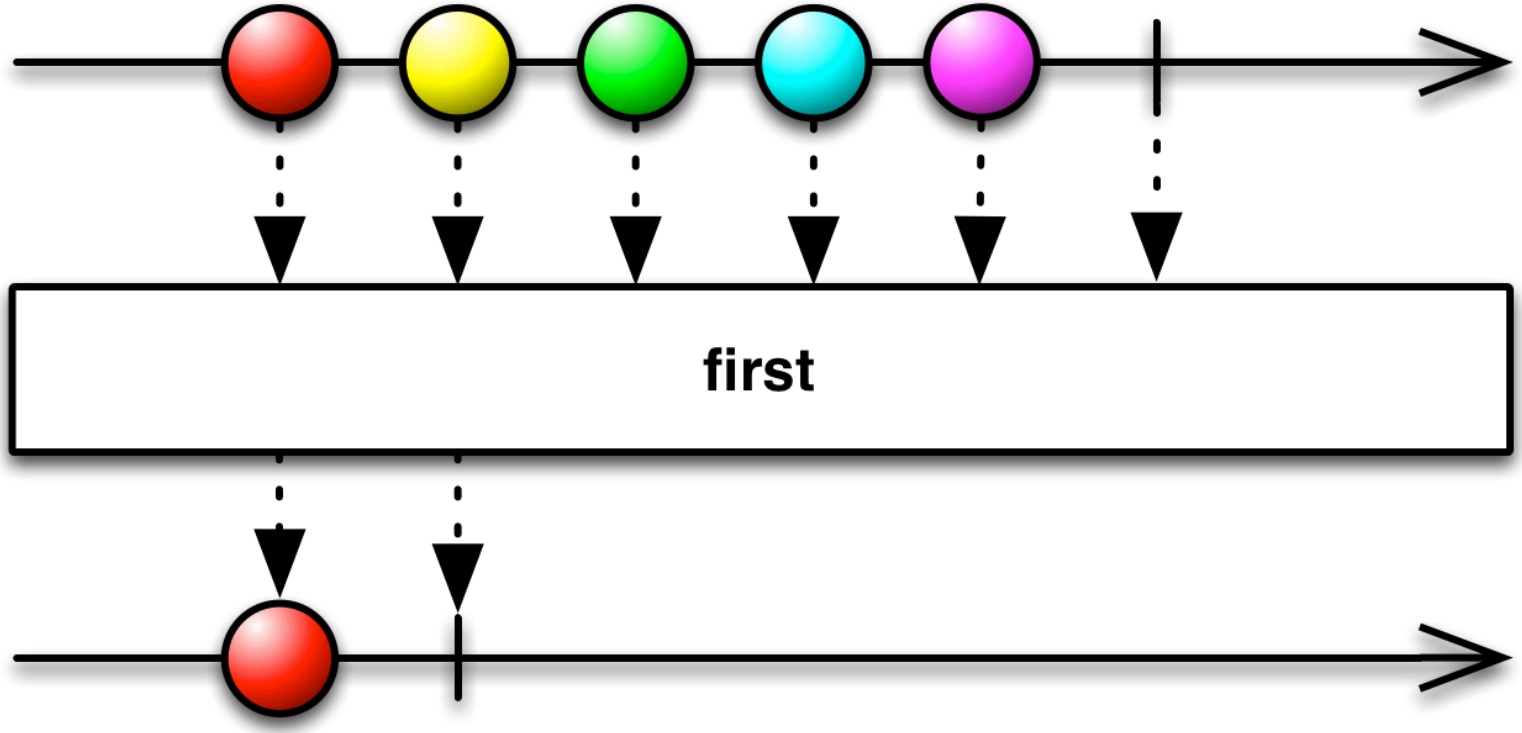
RxJava: Filtering Observables



RxJava: Filtering Observables

```
Observable
    .just("doc1", "doc2", "doc3")
    .flatMap(new Func1<String, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(String s) {
            return bucket.async().get(s);
        }
    })
    .filter(new Func1<JsonDocument, Boolean>() {
        @Override
        public Boolean call(JsonDocument document) {
            return document.content().containsKey("foo");
        }
    })
    .subscribe();
```

RxJava: Filtering Observables



RxJava: Filtering Observables

```
1  bucket
2     .async()
3     .getFromReplica("doc", ReplicaMode.ALL)
4     .first()
5     .subscribe();
```

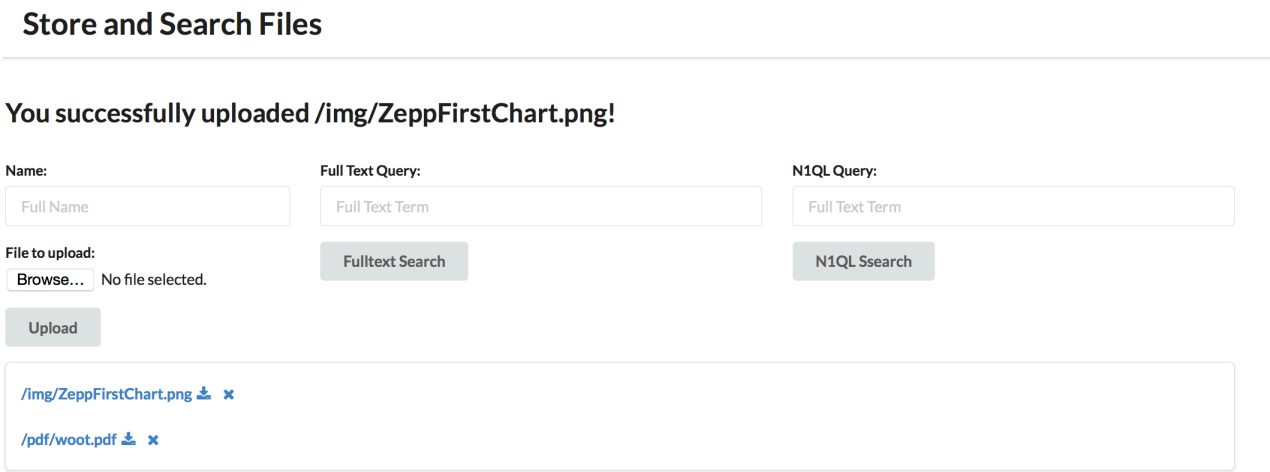
Brain Break

Example

Store, index and search files

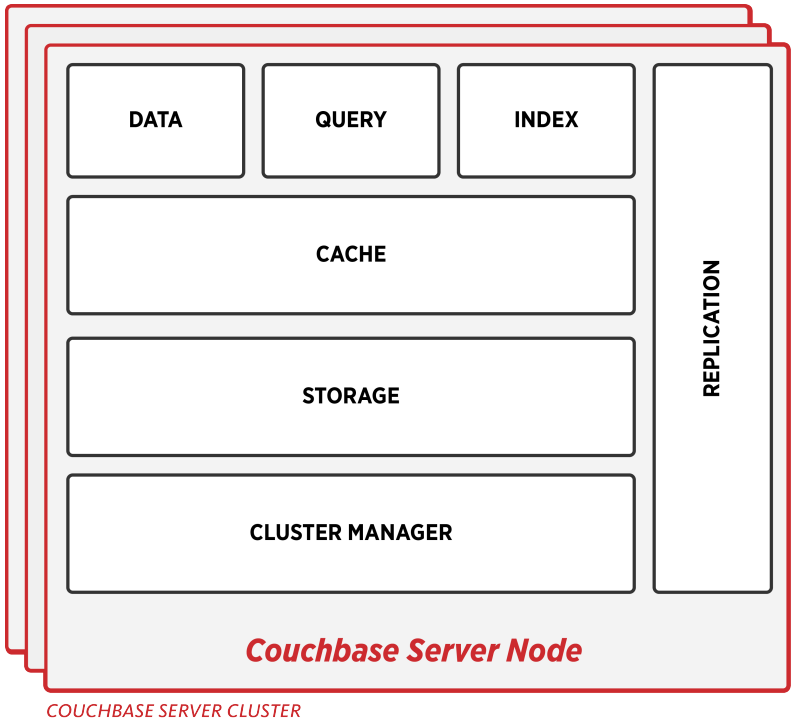
The Application

- Using Ratpack as web framework
- Using Couchbase as database
- User can Upload a file
- User can list all the files
- User can search files
  - Using N1QL
  - Using fulltext search



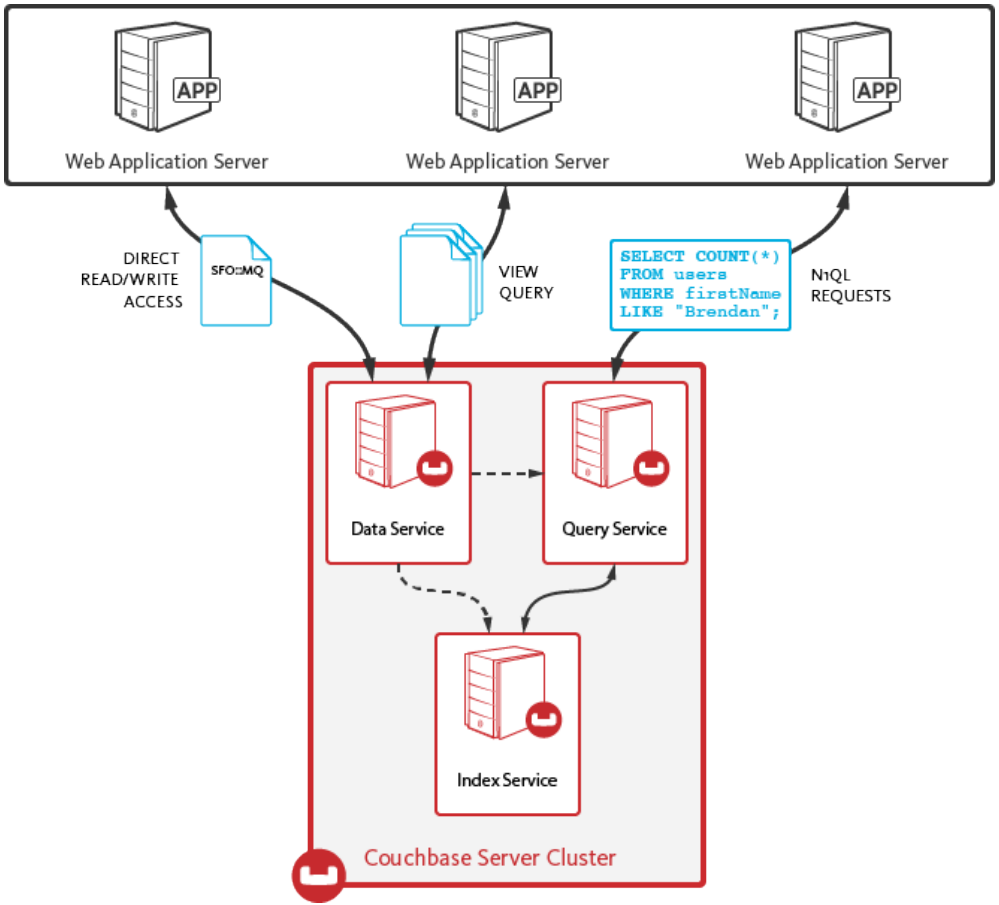
About Couchbase

Couchbase Server – Single Node Architecture

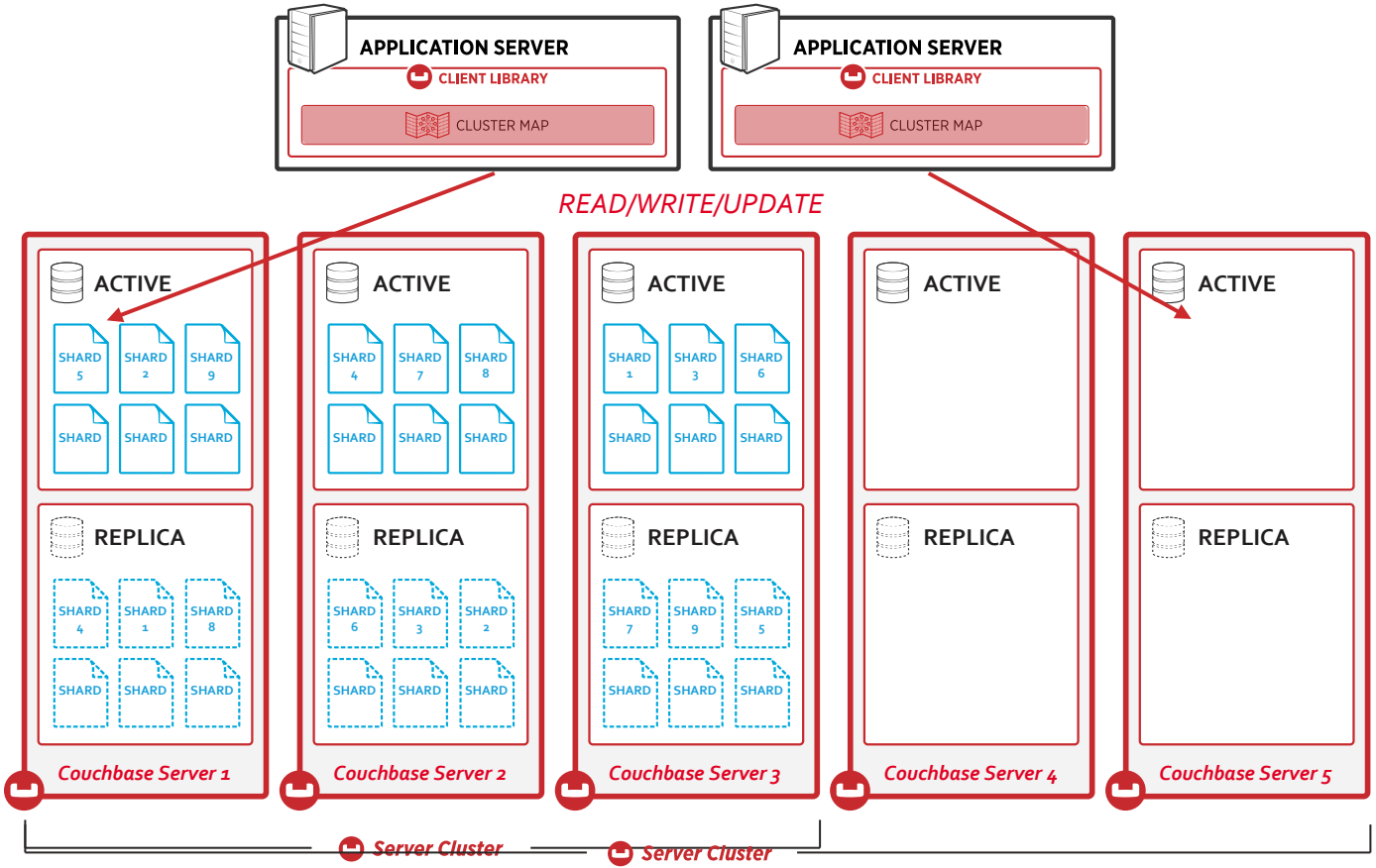


- **Data Service** – builds and maintains local view indexes
- **Indexing Engine** – builds and maintains Global Secondary Indexes
- **Query Engine** – plans, coordinates, and executes queries against either Global or Local view indexes
- **Cluster Manager** – configuration, heartbeat, statistics, RESTful Management interface

Application to Database Integration



Smart Connectivity with Built in Sharding and Replication



Accessing Data From Couchbase

Key access using Document ID

- Operations are extremely fast with consistent low latency
- Reads and writes are evenly distributed across Data Service nodes
- Data is cached in built-in Managed Caching layer and stored in persistent storage layer

Views using static queries

- Pre-computed complex Map-Reduce queries
- Incrementally updated to power analytics, reporting and dashboards
- Strong for complex custom aggregations

Queries using N1QL

- SQL-like : `SELECT * FROM WHERE`, `LIKE`, `GROUP`, etc.,
- `JOIN`s
- Powerful Extensions (nest, unnest) for JSON to support nested and hierarchical data structures.
- Multiple access paths – Views and global secondary indexes
- ODBC/JDBC drivers available

Building Applications with Java SDK – Synchronous API

```
//connecting to the cluster via known node(s)
CouchbaseCluster cluster =
CouchbaseCluster.create("192.168.1.101");

//opening a bucket, establishing resources
Bucket bucket = cluster.openBucket("customBucket", "password");

//creating JSON and a Document
JsonObject json = JsonObject.create().put("name", "John");
JsonDocument doc = JsonDocument.create("key1", json);

//storing the Document
Document inDb = bucket.insert(doc);
```

Building Applications with Java SDK

```
public List<Map<String, Object>> function(Bucket bucket) {
    String query = "SELECT * FROM `" + bucket.name() + "`";
    N1qlQueryResult result = bucket.query(N1qlQuery.simple(query));
    if (!result.finalSuccess()) {
        throw new DataRetrievalFailureException("Query error: " + result.errors());
    }
    List<Map<String, Object>> content = new ArrayList<Map<String, Object>>();
    for(N1qlQueryRow row : result) {
        content.add(row.value().toMap());
    }
    return content;
}
```

Complex N1QL Query

```
public static List<Map<String, Object>> getAll(final Bucket bucket, String from,
String to) {
    String queryStr = "SELECT faa AS fromAirport, geo " +
                      "FROM `" + bucket.name() + "` r" +
                      "WHERE airportname = " + from +
                      "UNION SELECT faa AS toAirport, geo " +
                      "FROM `" + bucket.name() + "` r" +
                      "WHERE airportname = " + to;
    ParameterizedN1qlQuery query = ParameterizedN1qlQuery.parameterized(queryStr,
                                   JsonArray.create().add(from).add(to));
    N1qlQueryResult queryResult = bucket.query(query);
    return extractResultOrThrow(queryResult);
}
```

Building Applications with Java SDK - Asynchronous API

- The Cluster and Bucket both have async versions, obtained by calling **async()** method.
- Asynchronous API exposes RxJava **Observables**.
- Very rich and expressive API in terms of **combinations** and **transformations**.

Building Applications with Java SDK - Asynchronous API

```
//retrieving a document and extracting data for output
bucket.async()
      .get("key1")
      .map(doc -> doc.content().getString("name"))
      .subscribe(name -> System.out.println("Hello " + name))
```

Building Applications with Java SDK - Asynchronous API

- Async API, exposing an **Observable<JsonDocument>**
- Observable is a stream, can be connected to an Observer

```
//retrieving a document and extracting data for output
bucket.async()
      .get("key1")
      .map(doc -> doc.content().getString("name"))
      .subscribe(name -> System.out.println("Hello " + name))
```

Building Applications with Java SDK - Asynchronous API

- Simple transformation operator from RxJava, `T->R`
- Gets a `JsonDocument`
- Extract String name value
- Gives `Observable<String>`

```
//retrieving a document and extracting data for output
bucket.async()
      .get("key1")
      .map(doc -> doc.content().getString("name"))
      .subscribe(name -> System.out.println("Hello " + name))
```

Error Handling

with RxJava

Couchbase

Timeouts

Timeouts

```
bucket.get("doc", Integer.MAX_VALUE, TimeUnit.SECONDS);
```

Circuit Breakers

- monitor traffic
- open if errors happen
  - Latency
  - Throughput
  - Wrong results
- close in a controlled fashion
- expose metrics

Backpressure

- Allows for coordinated flow control under stress conditions

Preparing to Fail

- Things will go wrong, so better plan for it
- Do not trust integration points (including the SDK)
- Synchronous retry & fallback is too damn hard

- RxJava provides (almost) everything you need to
  - fallback
  - retry
  - fail fast

Timeouts

- The network is unreliable
- Servers fail
- The SDK contains bugs

- Always specify timeouts and deal with them!
- The synchronous wrapper defines them for you all the time.

Timeouts: Simple

```
1  bucket
2      .get("id")
3      .timeout(5, TimeUnit.SECONDS)
4      .subscribe(System.out::println);
```

Timeouts: Synchronous API

```
@Override
public <D extends Document<?>> D get(D document, long timeout, TimeUnit timeUnit) {
  return asyncBucket
       .get(document)
       .timeout(timeout, timeUnit)
       .toBlocking()
       .singleOrDefault(null);
}
```

Timeouts: Complex Example

```
bucket
    .query(ViewQuery.from("beers", "by_name").limit(100)) // query
    .flatMap(AsyncViewResult::rows) // stream each row
    .flatMap(AsyncViewRow::document) // grab doc for each row
    .filter(doc -> doc.content().getDouble("abv") > 5.0) // filter beer by abv
    .count() // count all filtered beers
    .timeout(10, TimeUnit.SECONDS) // overall timeout
    .subscribe(System.out::println); // print
```

Coordinated Retry

- Fail fast
  - Don't let your system get stuck
  - Shed load with backpressure

- Retry
  - immediately won't help
  - either linear or exponential back-off necessary
  - have a strategy if retry also doesn't work (Fallbacks!)

Coordinated Fallback

```
bucket
    .get("beer") // fetch doc
    .onErrorResumeNext(bucket.getFromReplica("beer", ReplicaMode.ALL)) // fallback to replica
    .first() // grab the first doc to arrive
    .map(doc -> doc.content().getString("name")) // extract the name
    .timeout(2, TimeUnit.SECONDS) // only wait 2 secs
    .doOnError(System.err::println) // error log if needed
    .onErrorReturn(error -> "Not found!"); // if failure, return a default string
```

Coordinated Retry: Builder

- Declarative API instead of complicated retryWhen

```
observable.retryWhen(
  anyOf(BackpressureException.class, TemporaryFailureException.class)
    .delay(Delay.exponential(TimeUnit.SECONDS), 5)
    .max(100)
    .build()
);
```

Code

Resources

- RxMarbles - http://rxmarbles.com
- Asynchronous programming with Couchbase
- Ratpack Documentation - https://ratpack.io/manual/current/
- Learning Ratpack by @danveloper Dan wood – Book
- Ratpack: The Story So Far (Phill Barber) - YouTube
- Greach 2016 - Daniel Hyun - Testing in Ratpack – YouTube
- Greach 2016 - Jeff Beck - Ratpack in Practice - YouTube

Where do you find us?

- developer.couchbase.com
- blog.couchbase.com
- @couchbase or @couchbasedev
- forums.couchbase.com
- stackoverflow.com/questions/tagged/couchbase

Couchbase
DEVELOPER COMMUNITY

Thank You!