RxJava

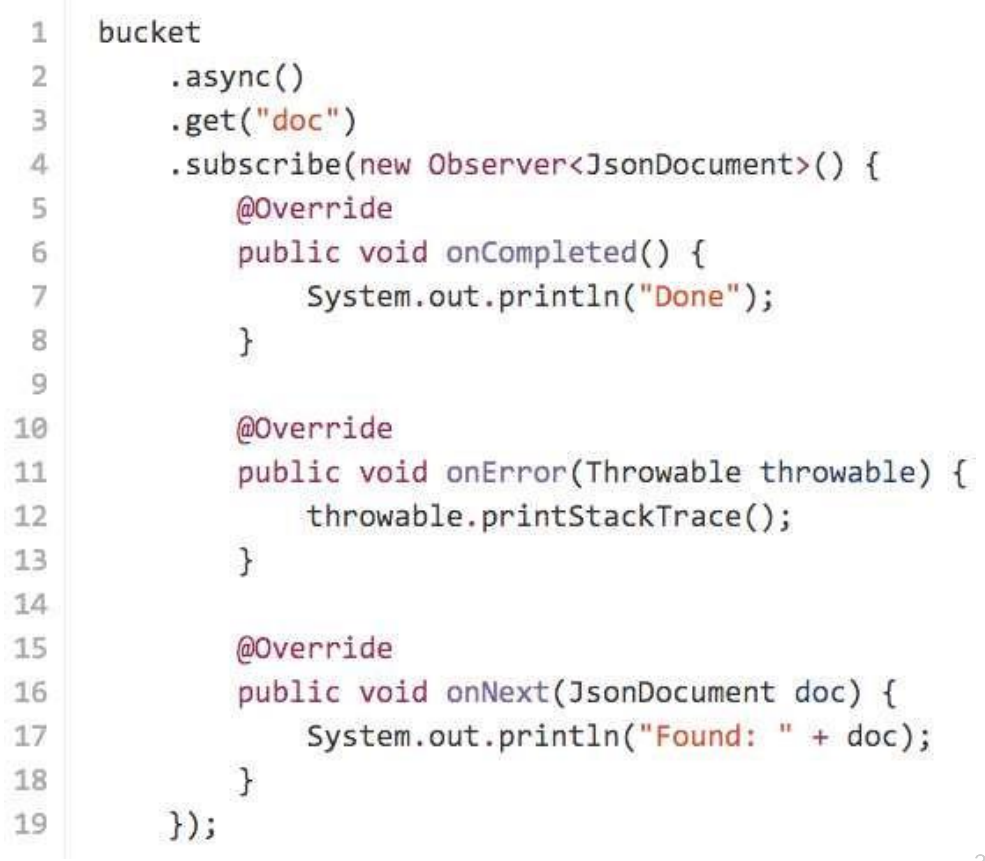
RxJava: Introduction

- 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()

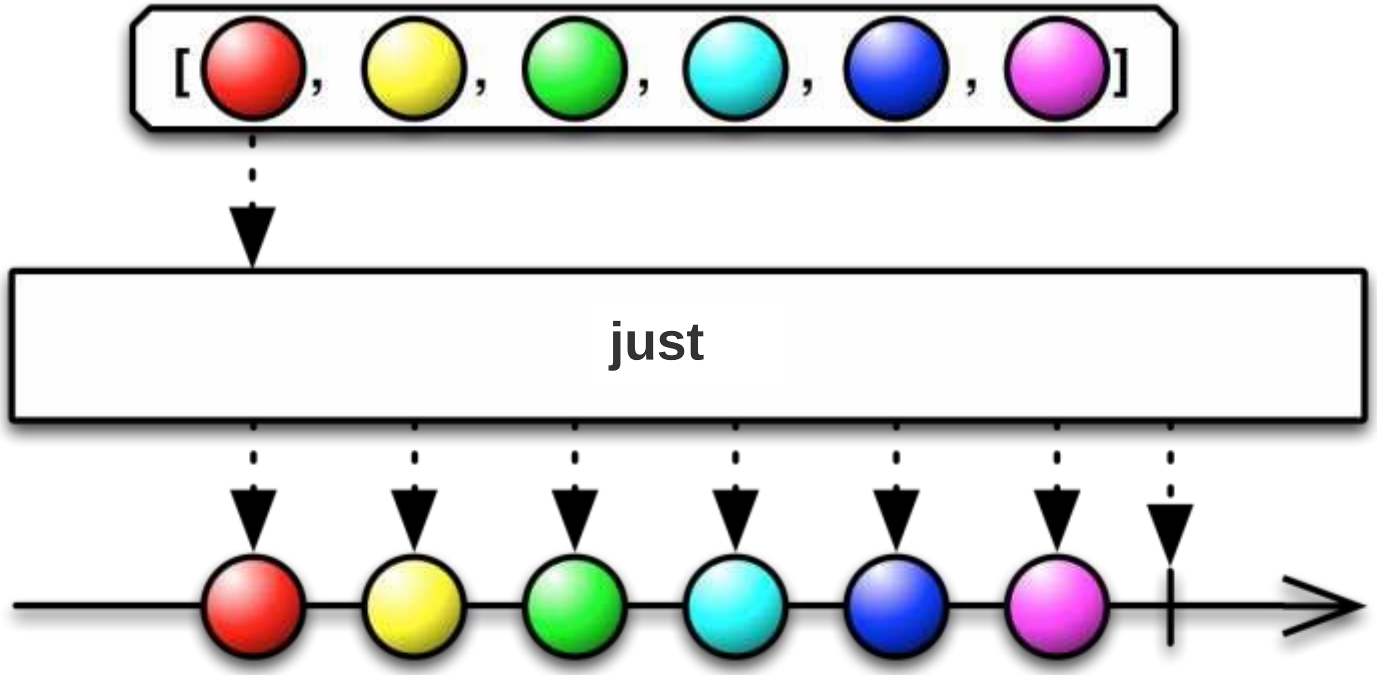
Consuming Observables

- The Observer subscribes and receives events.
- A cold Observable starts when subscribed.
- onNext can be called 0..N times

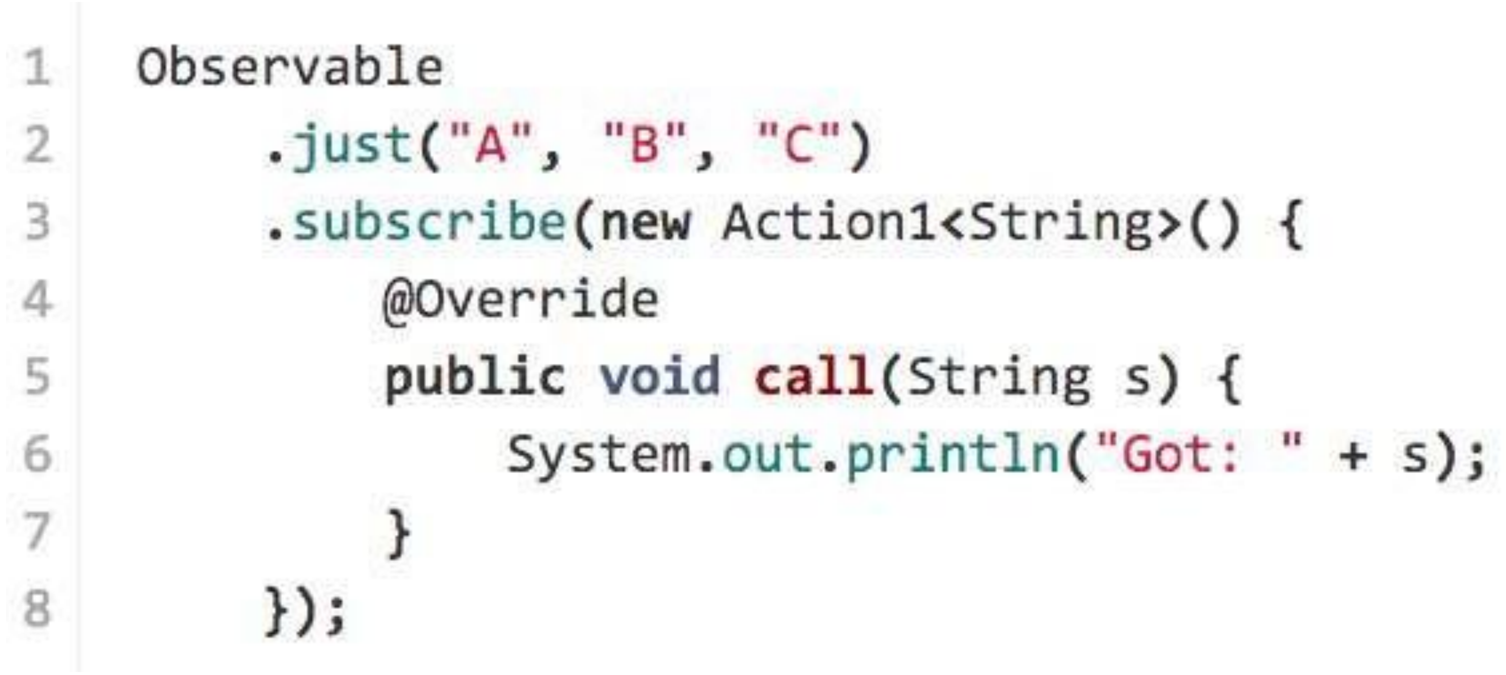


```java
bucket
    .async()
    .get("doc")
    .subscribe(new Observer<JsonDocument>() {
        @Override
        public void onCompleted() {
            System.out.println("Done");
        }

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

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

RxJava: Creating Observables

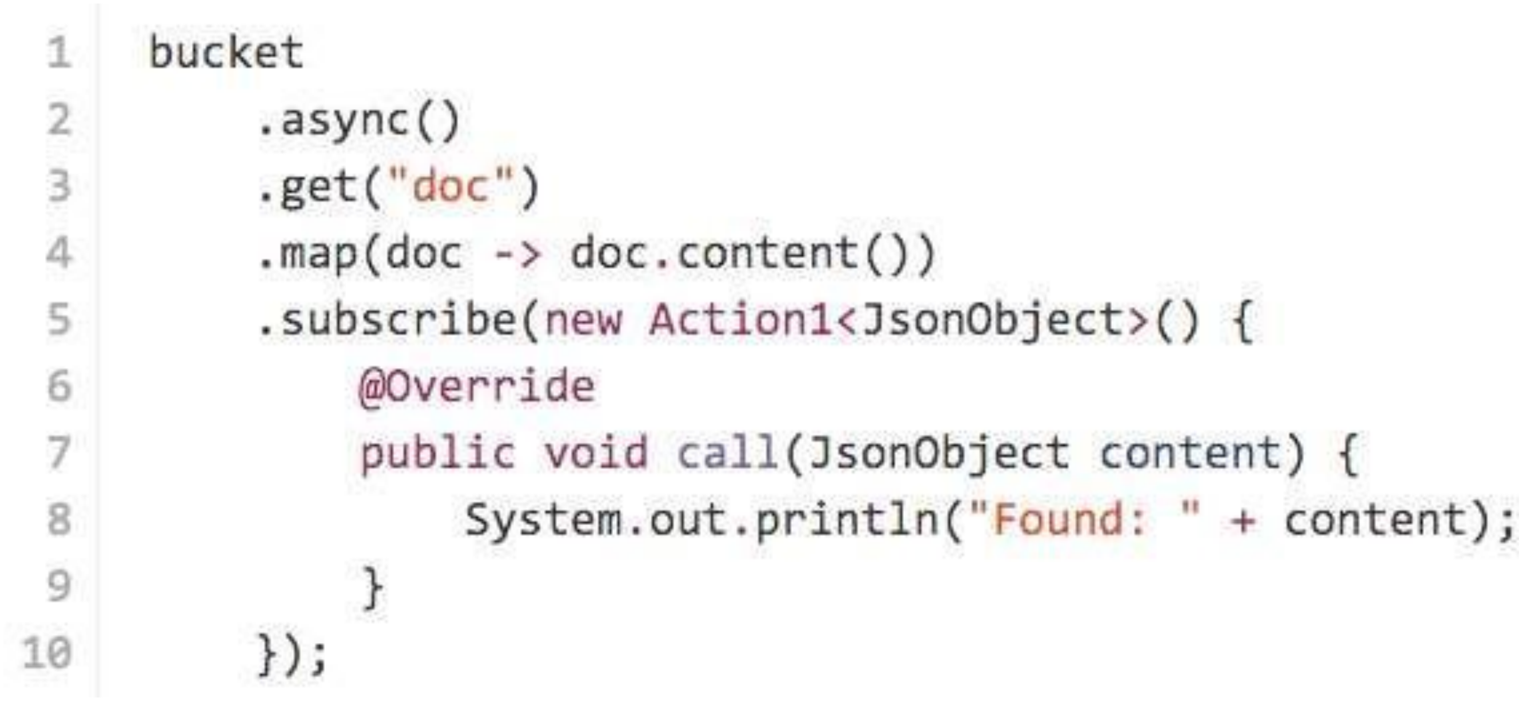


RxJava: Creating Observables



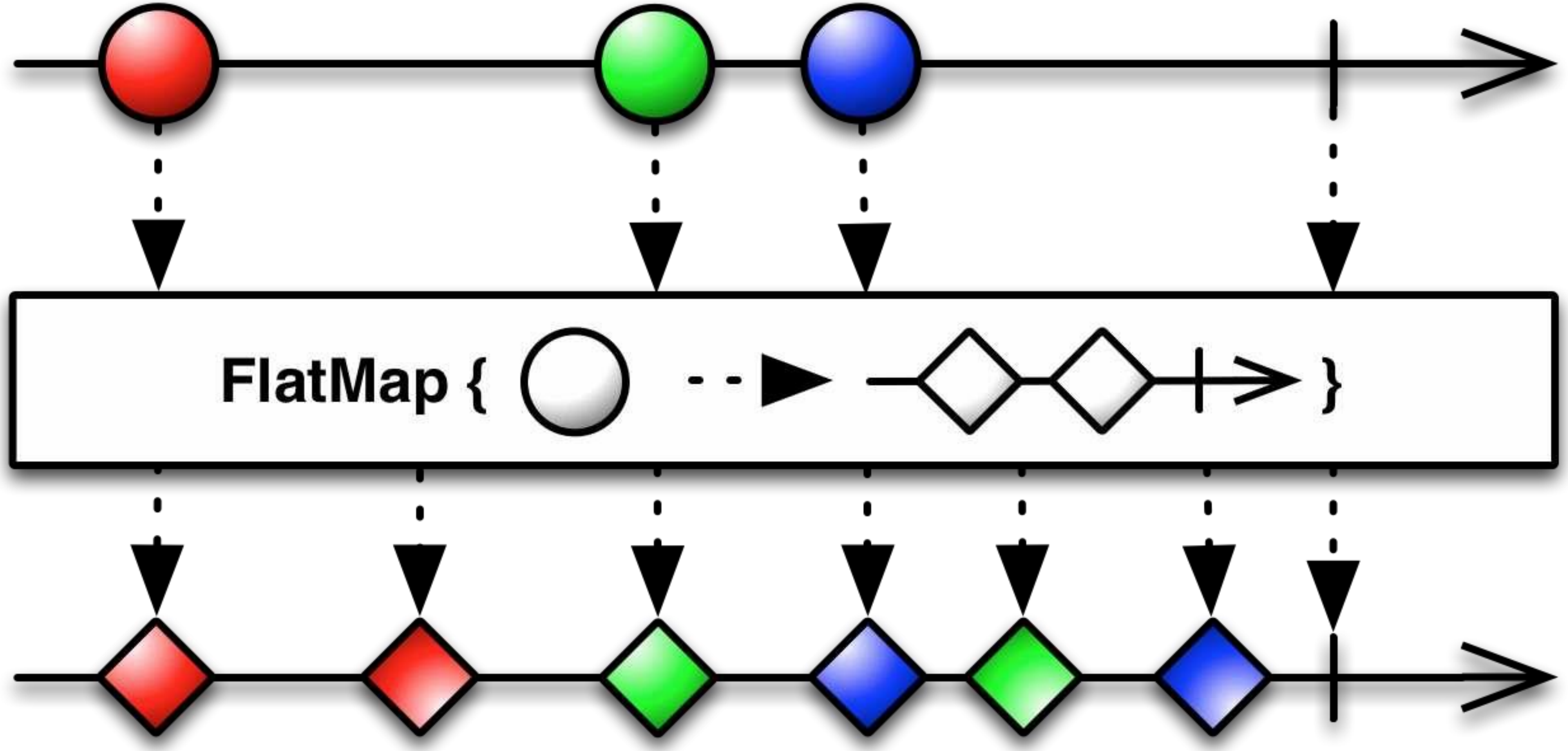
```
Observable
        .just("A", "B", "C")
        .subscribe(new Action1<String>() {
            @Override
            public void call(String s) {
                System.out.println("Got: " + s);
            }
        });
```

RxJava: Transforming Observables

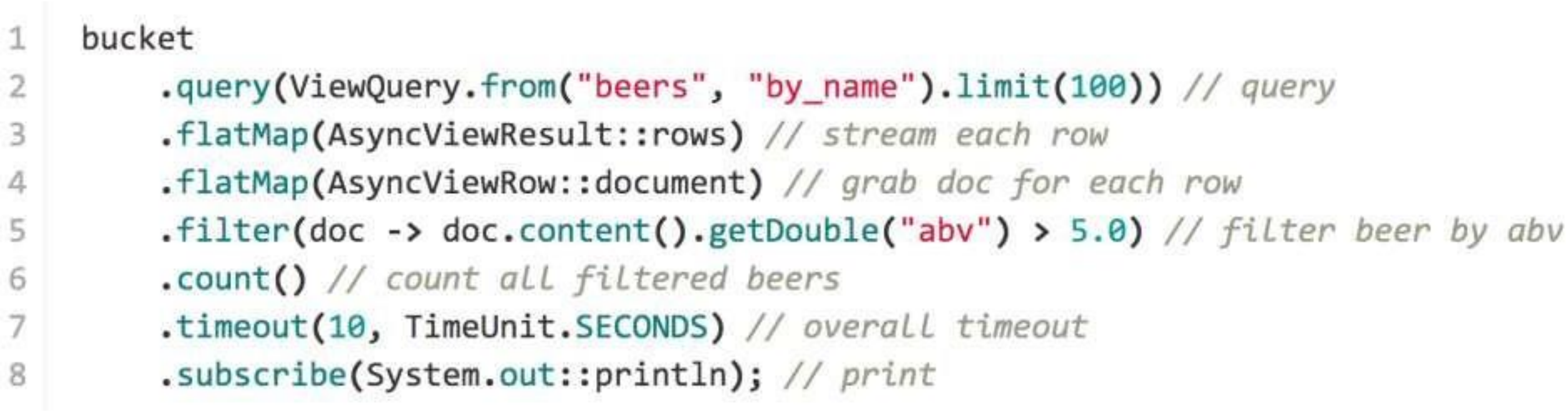


```java
bucket
      .async()
      .get("doc")
      .map(doc -> doc.content())
      .subscribe(new Action1<JsonObject>() {
          @Override
          public void call(JsonObject content) {
              System.out.println("Found: " + content);
          }
      });
```

RxJava: Transforming Observables

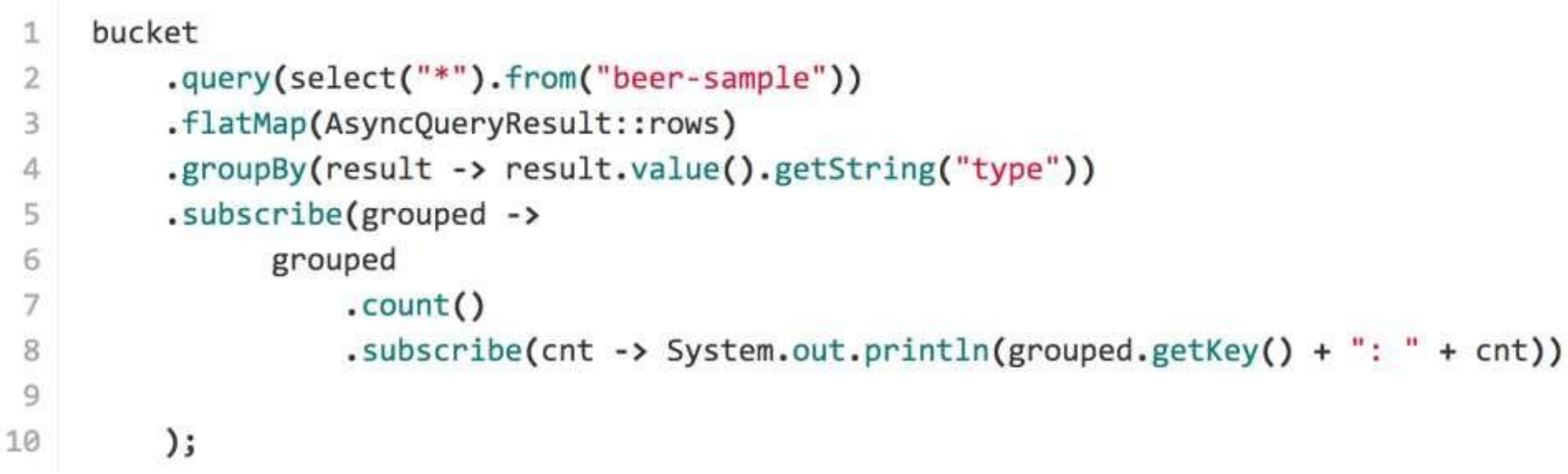


RxJava: Transforming Observables



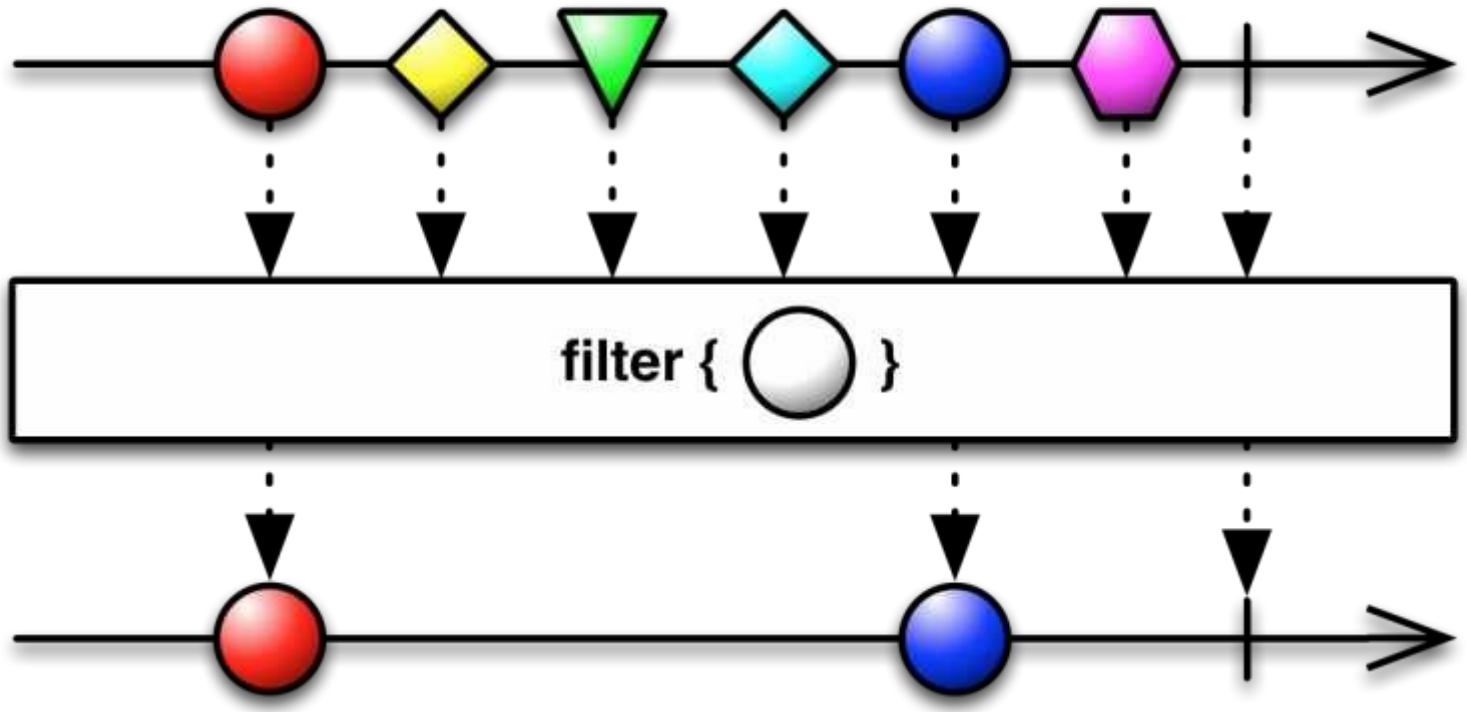
```java
bucket
    .query(ViewQuery.from("beers", "by_name").limit(100)) // query
    .flatMap(AsyncViewResult::rows) // stream each row
    .flatMap(AsyncViewRow::document) // grab doc for each row
    .filter(doc -> doc.content().getDouble("abv") > 5.0) // filter beer by abv
    .count() // count all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

RxJava: Transforming Observables

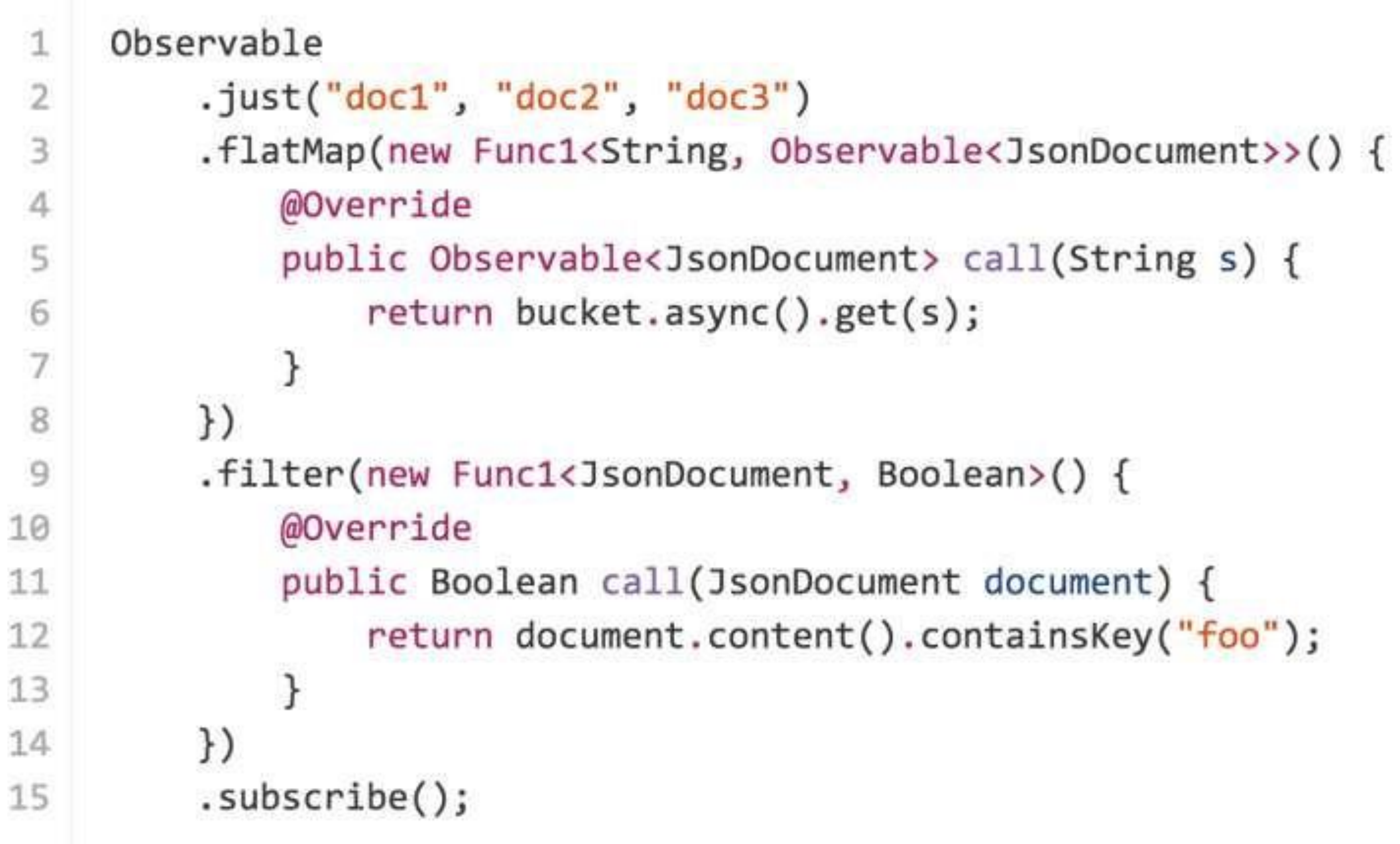


```
bucket
    .query(select("*").from("beer-sample"))
    .flatMap(AsyncQueryResult::rows)
    .groupBy(result -> result.value().getString("type"))
    .subscribe(grouped ->
        grouped
            .count()
            .subscribe(cnt -> System.out.println(grouped.getKey() + ": " + cnt))

    );
```

RxJava: Filtering Observables



RxJava: Filtering Observables



```java
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();
```

Batching Operations

Batching Operations

```
Cluster cluster = CouchbaseCluster.create();
Bucket bucket = cluster.openBucket();

List<JsonDocument> foundDocs = Observable
    .just("key1", "key2", "key3", "key4", "inexistentDoc", "key5")
    .flatMap(new Func1<String, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(String id) {
            return bucket.async().get(id);
        }
    })
    .toList()
    .toBlocking()
    .single();

for (JsonDocument doc : foundDocs) {
    System.out.println(doc.id());
}
```

```
key1
key2
key3
key4
key5
```

Lab : Java Rx – Reactive & N1QL