
toptal™

RxYourself - think big, think reactive

Aleksandar Simić

Android & iOS engineer

aleksandar.simic@toptal.com

+381 69 111 57 96


ReactiveX

Zašto bismo koristili Rx?

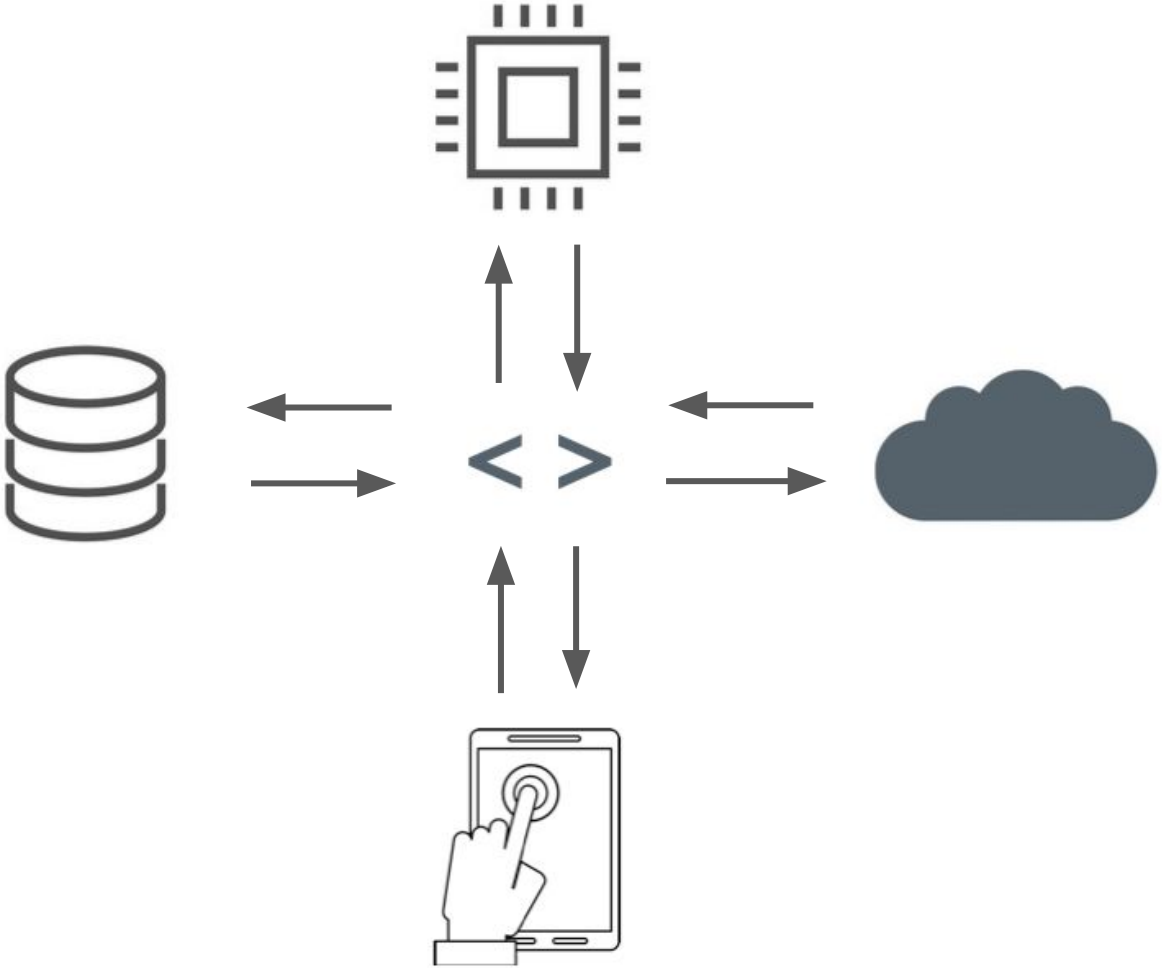
```
public interface CarRepository {
   Car getCar();
   void paint(int color);
   void addGas(int liters);
}

CarRepository carRepository = new CarRepository();
Car car = carRepository.getCar();
System.out.println(car);

carRepository.addGas(28);
carRepository.paint(Color.WHITE);
System.out.println(car);
```

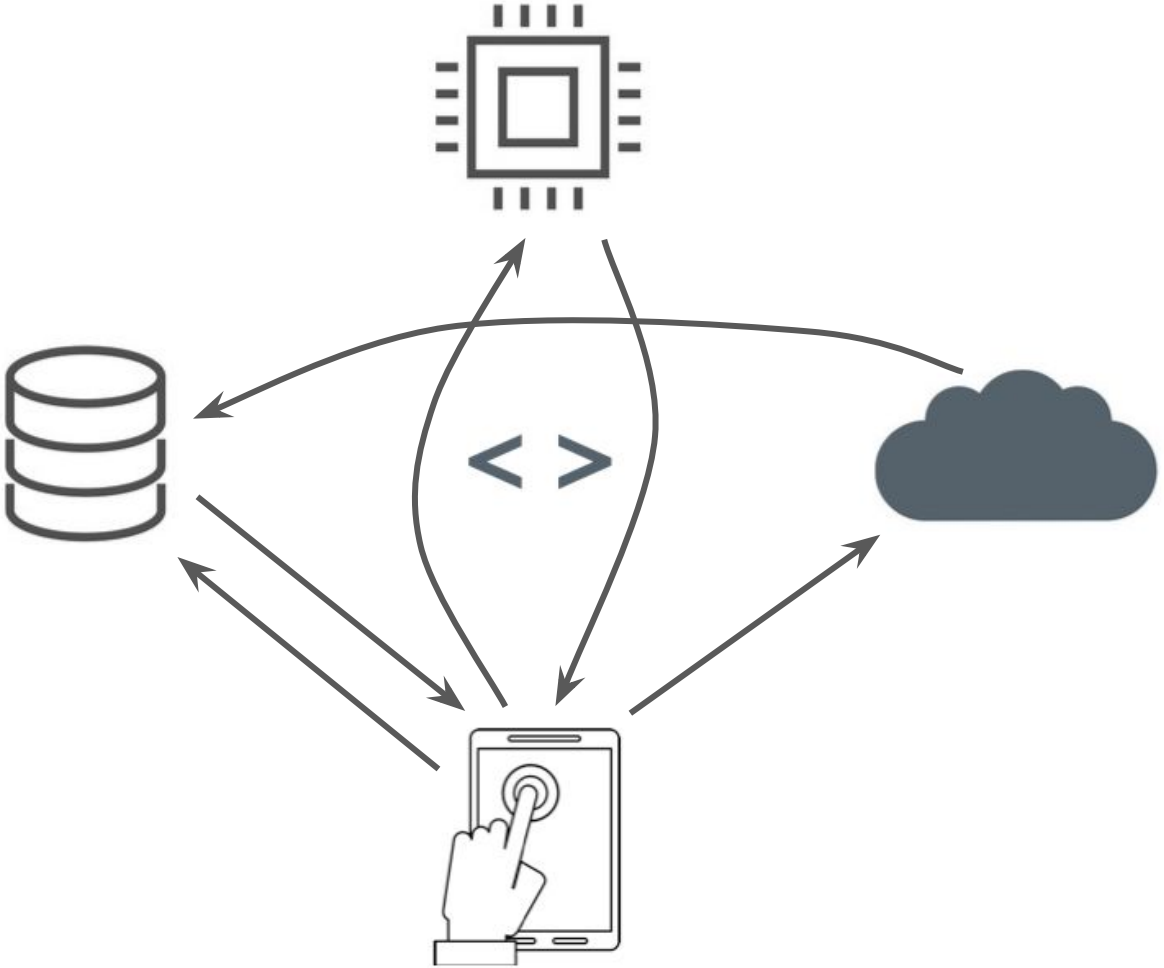
```
public interface CarRepository {
   Car getCar();
   void paintAsync(int color);
   void addGasAsync(int liters);
}

CarRepository carRepository = new CarRepository();
Car car = carRepository.getCar();
System.out.println(car);

carRepository.addGasAsync(28);
carRepository.paintAsync(Color.WHITE);
System.out.println(car);
```

```
public interface CarRepository {
   Car getCar();
   void paintAsync(int color, Runnable callback);
   void addGasAsync(int liters, Runnable callback);
}

CarRepository carRepository = new CarRepository();
Car car = carRepository.getCar();
System.out.println(car);

carRepository.addGasAsync(28, new Runnable() {
   @Override
   public void run() {
      System.out.println(car);
   }
});
```

```
public interface CarRepository {
    Car getCar();
    void paintAsync(int color, Listener listener);
    void addGasAsync(int liters, Listener listener);

    interface Listener {
        void onSuccess();
        void onFailure(Exception e);
    }
}

carRepository.addGasAsync(28, new CarRepository.Listener() {
    @Override public void onSuccess() {
        System.out.println(car);
    }

    @Override public void onFailure(Exception e) {
        System.out.println(e.getLocalizedMessage());
    }
});
```

```
carRepository.addGasAsync(28, new CarRepository.Listener() {
  @Override public void onSuccess() {
    carRepository.paintAsync(Color.WHITE, new CarRepository.Listener() {
      @Override public void onSuccess() {
        System.out.println(car);
      }

      @Override public void onFailure(Exception e) {
        System.out.println(e.getLocalizedMessage());
      }
    });
  }

  @Override public void onFailure(Exception e) {
    carRepository.paintAsync(Color.WHITE, new CarRepository.Listener() {
      @Override public void onSuccess() {
        System.out.println(car);
      }

      @Override public void onFailure(Exception e) {
        System.out.println(e.getLocalizedMessage());
      }
    });
  }
});
```

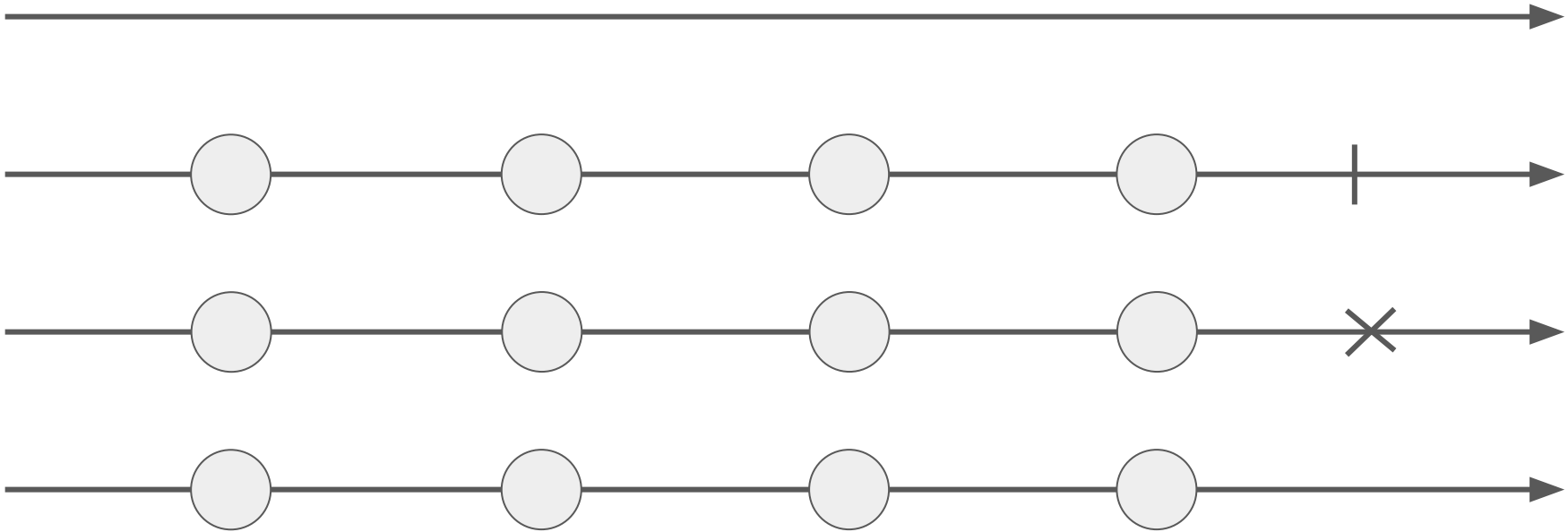




Zašto bismo koristili Rx?

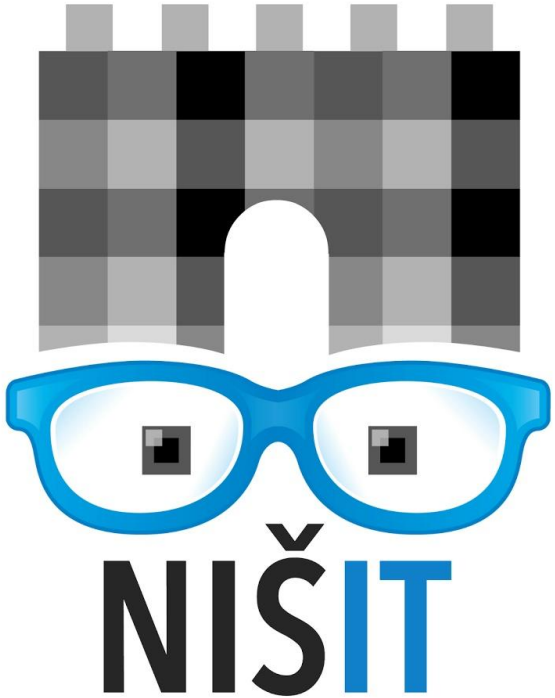
“Zato što reaktivno programiranje dozvoljava da se fokusirate na ono što želite da postignete a ne na tehničke detalje takvog pristupa. Ovo vodi do jednostavnog i čitkog koda i eliminiše veliki deo nepotrebnog koda koji vam skreće pažnju sa namene originalne logike. Kada je kod kratak i jasan ima manje grešaka i lakši je za razumevanje.” - Rx.NET in Action

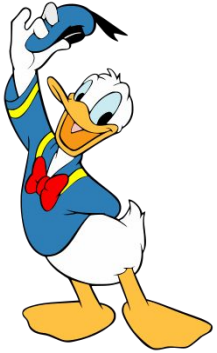
Marble dijagrami

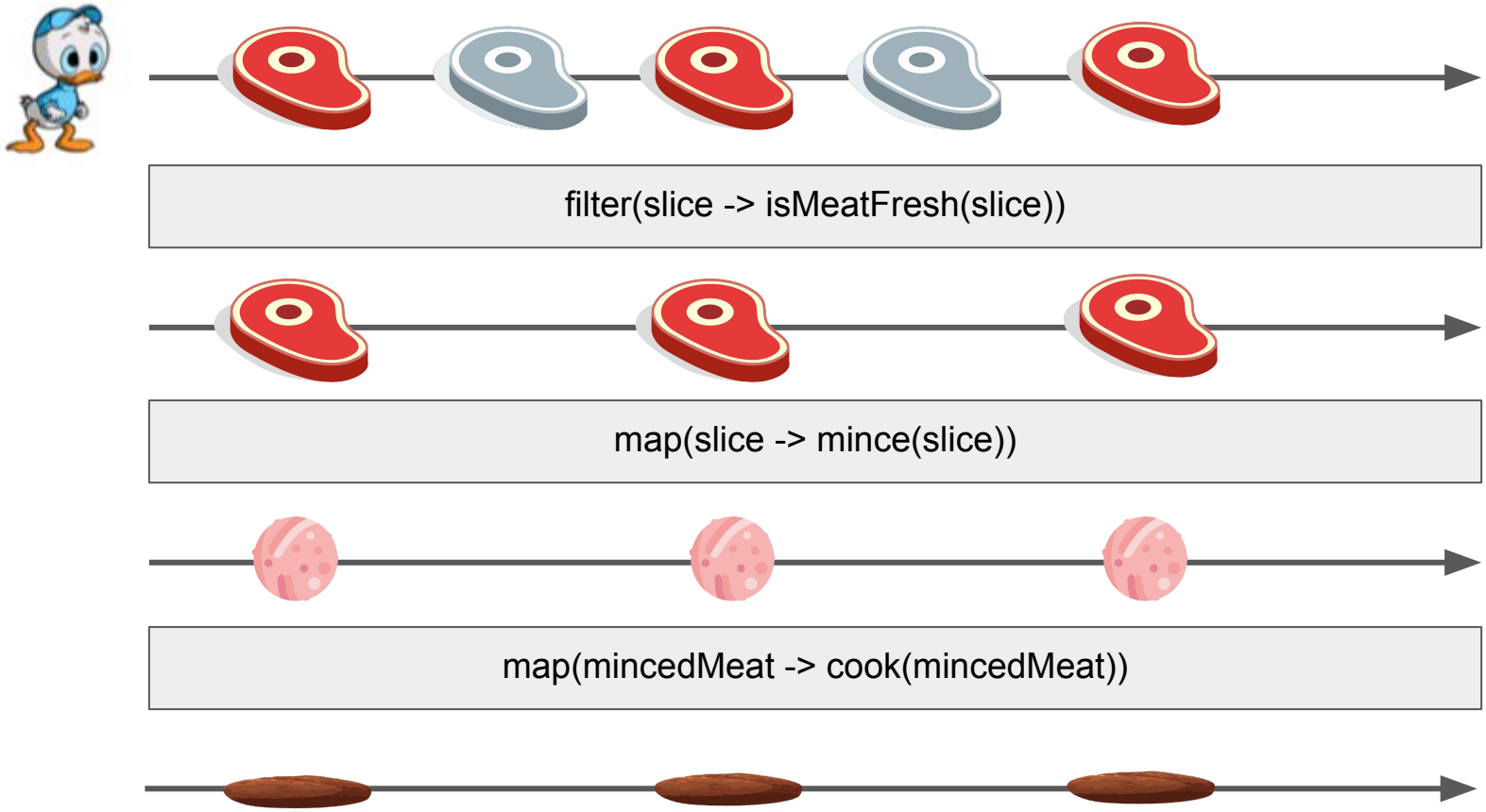


Elementi Rx-a

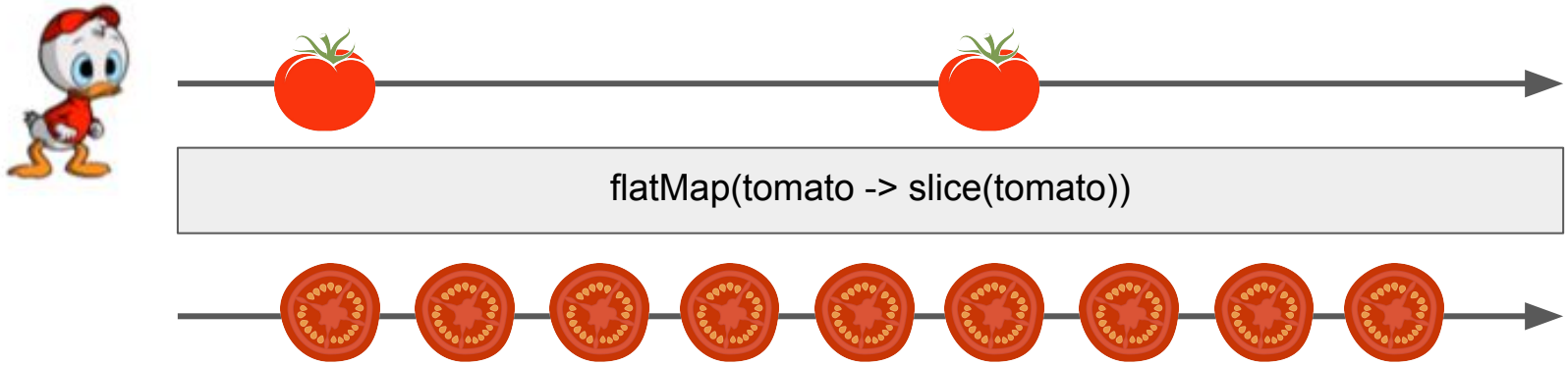
- Observable - tok (stream) podataka
  - Operator - modifikatori
- Observer/Subscriber - krajnji korisnici podataka (onNext, onError i onComplete)
- Scheduler - odgovorni za paralelnu obradu podataka
- Subject - i observable i observer

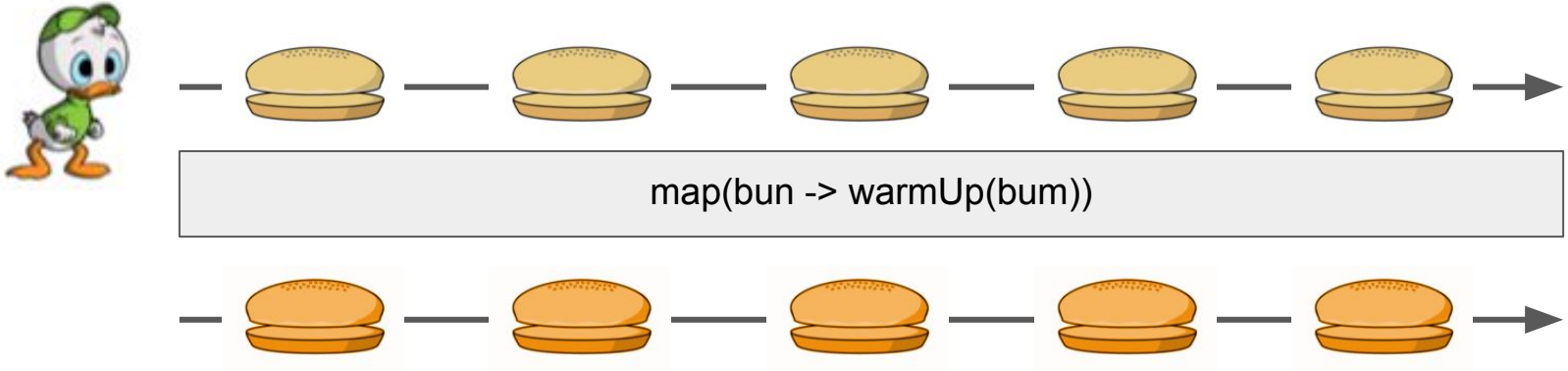

NIŠIT



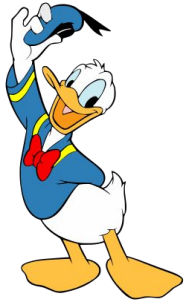

filter(slice -> isMeatFresh(slice))
map(slice -> mince(slice))
map(mincedMeat -> cook(mincedMeat))

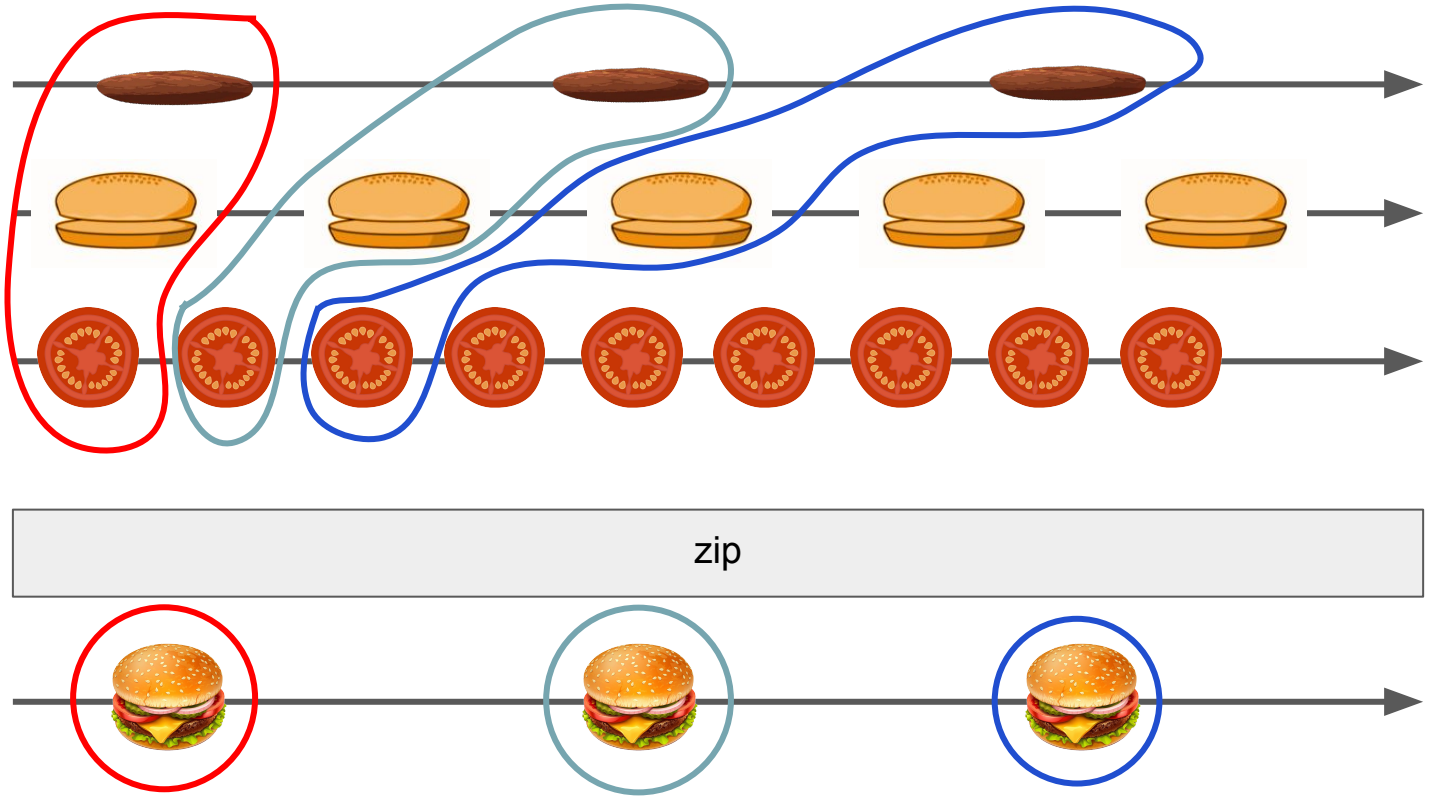
```
Observable<Meat> meatStream = Observable.from(meatSlicesSource)
                                        .filter(meatSlice -> meatSlice.isFresh())
                                        .map(meatSlice -> meatSlice.mince())
                                        .map(mincedMeat -> mincedMeat.cook());
```


flatMap(tomato -> slice(tomato))


map(bun -> warmUp(bum))

```
Observable<TomatoSlices> tomatoStream = Observable.from(tomatoSource)
                                                  .flatMap(tomato -> Observable.from(tomato.slice()));

Observable<Bun> bunStream = Observable.from(bunSource)
                                      .map(bun -> bun.warmUp());
```



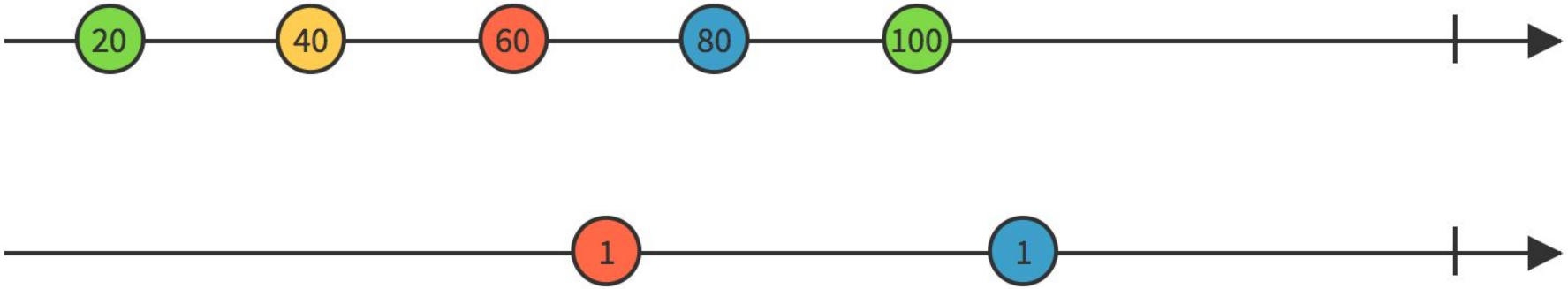

zip

```
Observable<Burger> burgerStream = Observable.zip(meatStream,
        tomatoStream,
        bunStream,
        (meat, tomatoSlice, bun) -> new Burger(meat, tomatoSlice, bun));

burgerStream.subscribe(burger -> eat(burger),
        error -> complain(error),
        () -> leave());
```

```
Observable<Burger> burgerStream = Observable.zip(meatStream,
        tomatoStream,
        bunStream,
        (meat, tomatoSlice, bun) -> new Burger(meat, tomatoSlice, bun));

burgerStream.subscribeOn(Schedulers.computation())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(burger -> eat(burger),
        error -> complain(error),
        () -> leave());
```

Chat

```
public class ChatEvent { /* */ }

public class ChatMessageEvent extends ChatEvent { /* */ }

public class ChatStatusEvent extends ChatEvent { /* */ }
```



merge

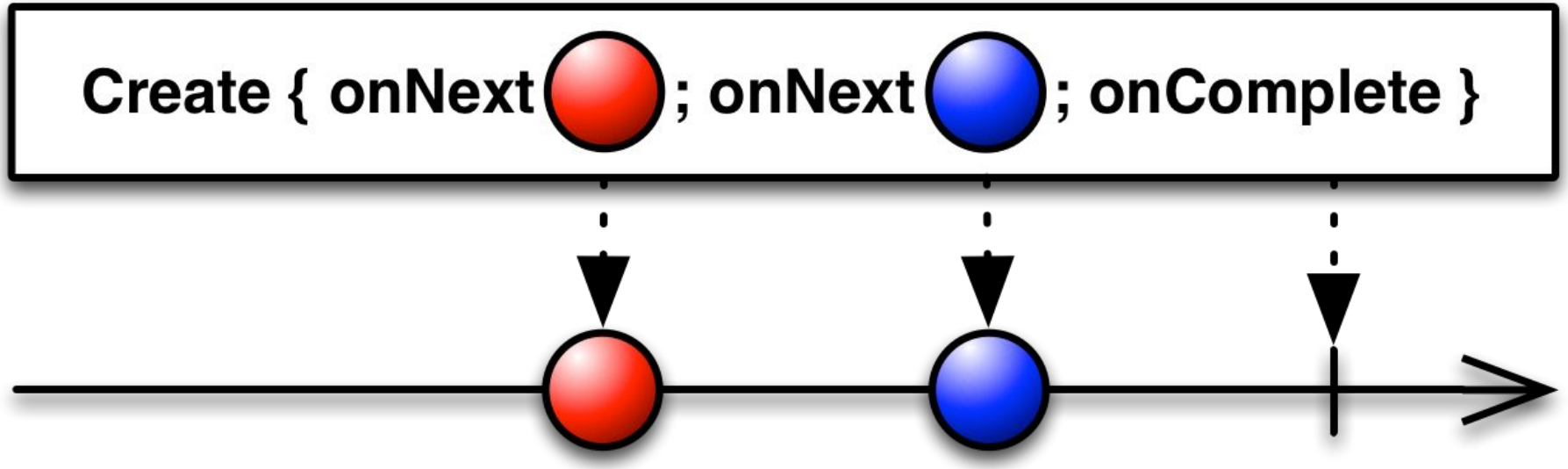
```
void onCreate() {
  Observable<ChatEvent> chatEvents = Observable.merge(RxPubnub.instance().getEvents(),
                                                      FirebaseChatService.getEvents())
        .distinct();

  chatMessagesDisposable = chatEvents
        .ofType(ChatMessageEvent.class)
        .subscribe(message -> appendMessageToUi(message),
              error -> showError(error),
              () -> closeScreen());

  chatStatusDisposable = chatEvents
        .ofType(ChatStatusEvent.class)
        .subscribe(status ->  /*handle status.isOnline() */,
                    error -> showError(error));
}

void onDestroy() {
  chatMessagesDisposable.dispose();
  chatStatusDisposable.dispose();
}
```

```
public class FirebaseChatService extends FirebaseMessagingService {

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);
    }
```

```
public class FirebaseChatService extends FirebaseMessagingService {

    private static PublishSubject<ChatEvent> eventsSubject = PublishSubject.create();

    public static Observable<ChatEvent> getEvents() {
        return eventsSubject;
    }

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);
        ChatMessageEvent event = ChatMessageEvent.fromFirebase(remoteMessage);
        eventsSubject.onNext(event);
    }

}
```

```
public class RxPubnub {

   private PubNub pubnub;
   private Observable<ChatEvent> messagesStream;

   private RxPubnub() {
      initPubnub();
      initMessagesStream();
   }
}
```

```
private void initMessagesStream() {
  SubscribeCallback callback = new SubscribeCallback() {
     @Override  public void status(PubNub pubnub, PNStatus status) {

     }

     @Override public void message(PubNub pubnub, PNMessageResult message) {

     }

     @Override public void presence(PubNub pubnub, PNPresenceEventResult presence) {
        //no operation
     }
  };

  pubnub.addListener(callback);
  pubnub.subscribe()
        .channels(Arrays.asList("awesomeChannel"))
        .execute();
}
```


Create { onNext ; onNext ; onComplete }

```
private void initMessagesStream() {
  messagesStream = Observable.create(subscriber -> {

    SubscribeCallback callback = new SubscribeCallback() {
      @Override public void status(PubNub pubnub, PNStatus status) {
      }

      @Override public void message(PubNub pubnub, PNMessageResult message) {
      }

      @Override public void presence(PubNub pubnub, PNPresenceEventResult presence) {
        //no operation
      }
    };

    pubnub.addListener(callback);
    pubnub.subscribe()
        .channels(Arrays.asList("awesomeChannel"))
        .execute();
  });
}
```

```
@Override public void status(PubNub pubnub, PNStatus status) {
  if (subscriber.isDisposed()) { return; }

  switch (status.getCategory()) {
     case PNDisconnectedCategory:
     case PNAccessDeniedCategory:
     case PNUnexpectedDisconnectCategory:
        subscriber.onNext(ChatStatusEvent.offline());
        Break;

     case PNConnectedCategory:
     case PNReconnectedCategory:
        subscriber.onNext(ChatStatusEvent.online());
        Break;

     default:
        break;
  }
}
```

```
@Override public void message(PubNub pubnub, PNMessageResult message) {
  if (subscriber.isDisposed()) { return; }

  subscriber.onNext(ChatMessageEvent.fromPubnub(message.getMessage()));
}
```

```
public class RxPubnub {

   private PubNub pubnub;
   private Observable<ChatEvent> messagesStream;

   private RxPubnub() {
      initPubnub();
      initMessagesStream();
   }
}
```

```
public class RxPubnub {

  private PubNub pubnub;
  private Observable<ChatEvent> receivedMessagesStream;
  private PublishSubject<String> sendMessageStream;

  private RxPubnub() {
     initPubnub();
     initReceivedMessageStream();
     initSendMessageStream();
  }

  private void initSendMessageStream() {
     sendMessageStream = PublishSubject.create();
  }
}
```

```
public void sendMessage(String message) {
  sendMessageStream.onNext(message);
}

private Observable<ChatEvent> constructSendMessageObservable(String message) {
  return Observable.create(subscriber -> {

    PNCallback<PNPublishResult> publishListener = new PNCallback<PNPublishResult>() {

      @Override public void onResponse(PNPublishResult result, PNStatus status) {
        if (subscriber.isDisposed()) { return; }

        ChatMessageEvent event = ChatMessageEvent.sent(message, status.isError());
        subscriber.onNext(event);
      }
    };

    pubnub.publish().channel("awesomeChannel").message(message) .async(publishListener);
  });
}
```

```
public Observable<ChatEvent> getEvents() {
  return Observable.merge(receivedMessagesStream,
        sendMessageStream
              .flatMap(message -> constructSendMessageObservable(message)));
}
```


toptal™