ASYNCHRONOUS PROGRAMMING MADE EASY

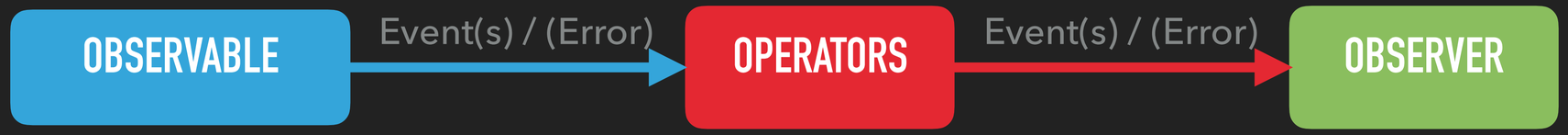
REACTIVE X

RX + {LANGUAGE}

- RxJava
- RxJS
- Rx.NET
- UniRx (Unity)
- RxScala
- RxClojure
- RxCpp
- RxLua
- Rx.rb
- RxPy
- RxGo
- RxGroovy
- RxHRuby
- RxKotlin
- RxSwift
- RxPHP
- reaxive (Elixir)
- RxDart
- RxNetty
- RxAndroid
- RxCocoa

Insert <wow.gif> here

OBSERVABLE




```
Observable<Integer> observable = Observable.range( start: 0,  count: 100);
observable = observable.map(number -> number * 2);
observable.subscribe(number -> Log.d( tag: "", msg: "Received a number: "+ number));
```

Count from 0 to 99

Multiply each number by 2

Display each number


```
Observable<Integer> observable = Observable.range( start: 0,  count: 100);
```

```
Observable<Integer> observable = Observable.create(subscriber -> {
    for (int i = 0 ; i < 100 ; i++)
        subscriber.onNext(i);
    subscriber.onComplete();
});
```

```
public Single<T> makeCall(String url, String params){
    return Single.create(subscriber -> new HttpClient().makeCall(url, params, new Callback<T>() {
        @Override
        public void onSuccess(T t) {
            subscriber.onSuccess(t);
        }

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


```
int[] intArray = {1, 2, 3};
ArrayList<Integer> intList = new ArrayList<>();

Observable.range( start: 0,  count: 100);
Observable.just(0, 1, 2, 3);
Observable.fromArray(intArray);
Observable.fromCallable(() -> computePiDecimals( precision: 10000));
Observable.fromIterable(intList);

Observable.merge(
        Observable.just(1, 2),
        Observable.just(3, 4)
);
```

Creating Observables

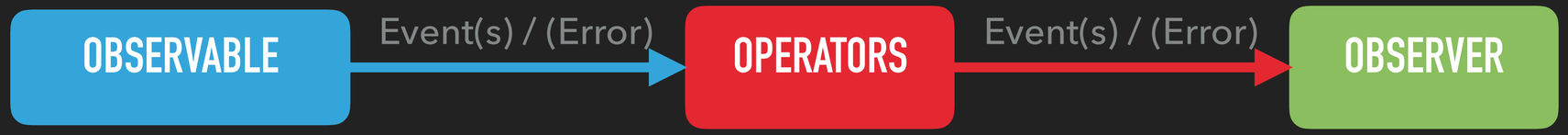
Operators that originate new Observables.

- `Create` — create an Observable from scratch by calling observer methods programmatically
- `Defer` — do not create the Observable until the observer subscribes, and create a fresh Observable for each observer
- `Empty` / `Never` / `Throw` — create Observables that have very precise and limited behavior
- `From` — convert some other object or data structure into an Observable
- `Interval` — create an Observable that emits a sequence of integers spaced by a particular time interval
- `Just` — convert an object or a set of objects into an Observable that emits that or those objects
- `Range` — create an Observable that emits a range of sequential integers
- `Repeat` — create an Observable that emits a particular item or sequence of items repeatedly
- `Start` — create an Observable that emits the return value of a function
- `Timer` — create an Observable that emits a single item after a given delay

http://reactivex.io/documentation/operators.html

Insert <wow.gif> here

OBSERVABLE




```
Observable<Integer> observable = Observable.range( start: 0,  count: 100);
observable = observable.map(number -> number * 2);
observable.subscribe(number -> Log.d( tag: "", msg: "Received a number: "+ number));
```

Count from 0 to 99

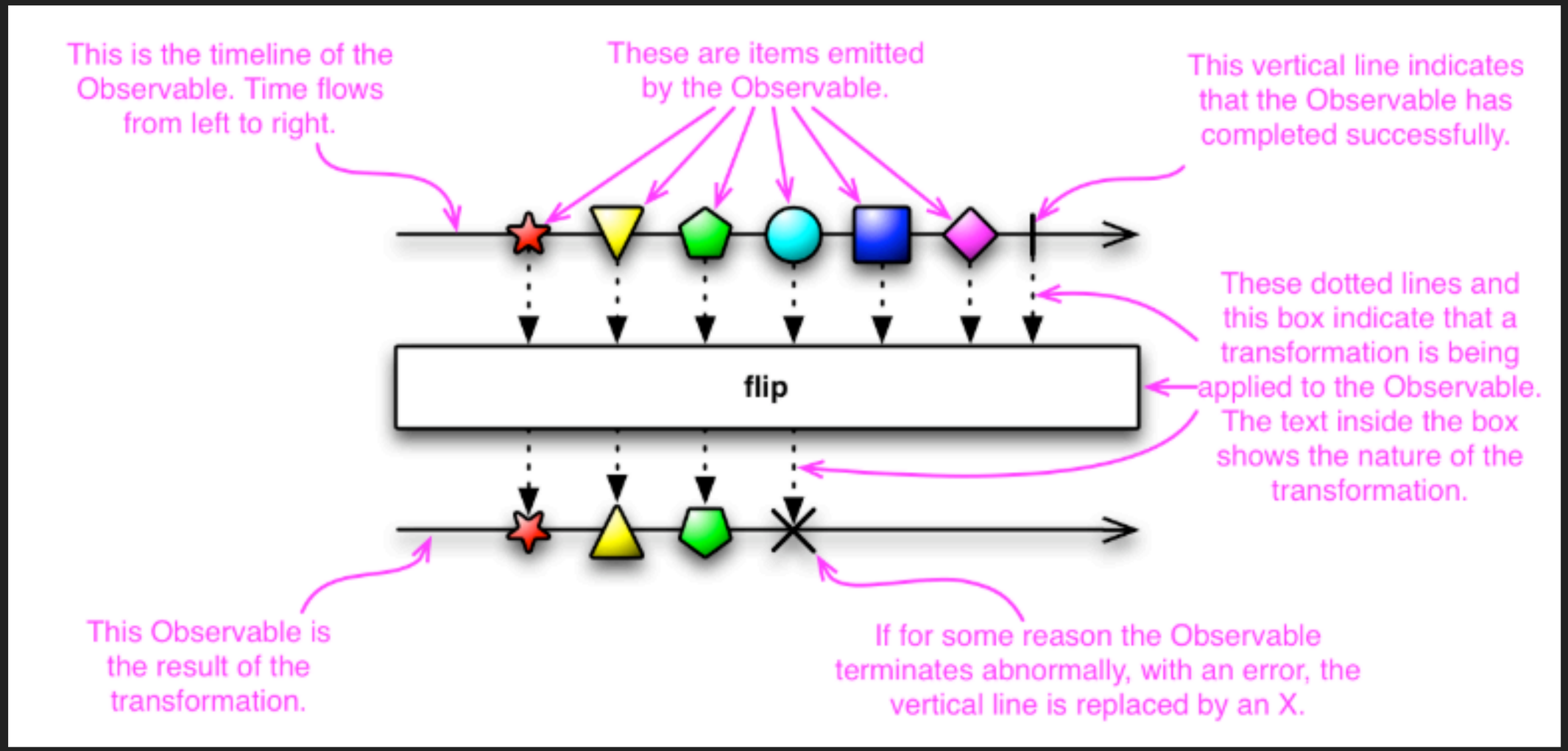
Multiply each number by 2

Display each number

REACTIVEX


```
observable.take(10) Observable<Integer>
        .filter(number -> number > 5) Observable<Integer>
        .skip(2) Observable<Integer>
        .buffer(4) Observable<List<Integer>>
        .flatMap(numberList -> Observable.fromIterable(numberList)) Observable<Integer>
        .reduce((a, b) -> a + b) Maybe<Integer>
        .onErrorReturnItem(5) Maybe<Integer>
```

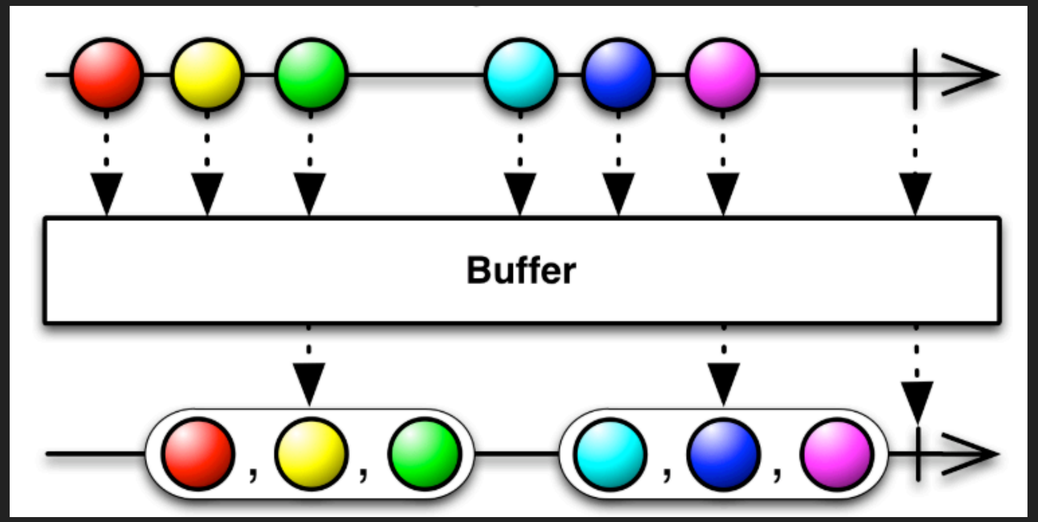
REACTIVEX

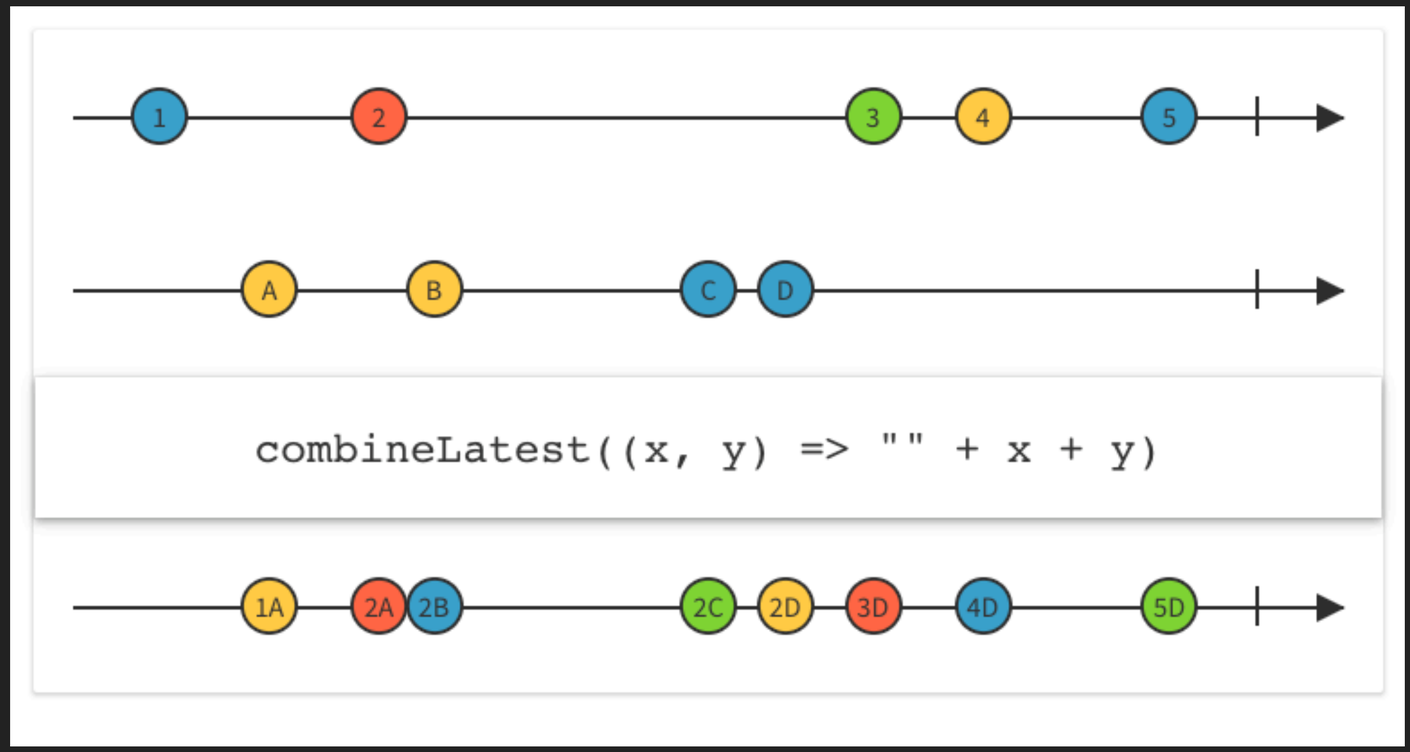


http://reactivex.io/documentation/observable.html

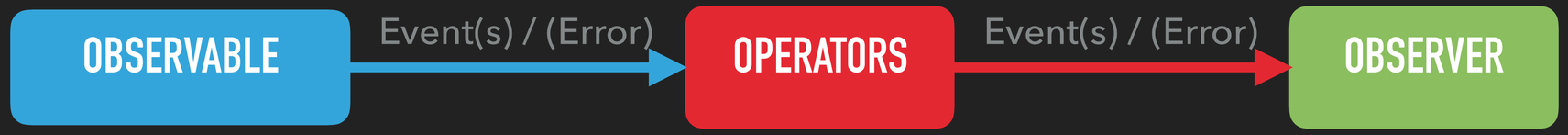
REACTIVEX

Insert <wow.gif> here





OBSERVABLE



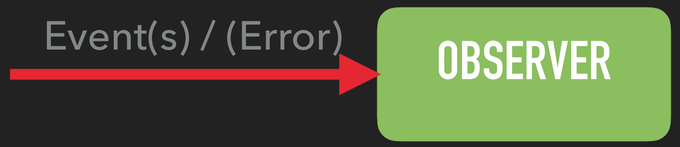

```
Observable<Integer> observable = Observable.range( start: 0,  count: 100);
observable = observable.map(number -> number * 2);
observable.subscribe(number -> Log.d( tag: "", msg: "Received a number: "+ number));
```

Count from 0 to 99

Multiply each number by 2

Display each number

REACTIVEX



Insert <wow.gif> here

```
.subscribe(new Observer<Integer>() {
    @Override
    public void onSubscribe(Disposable d) {

    }

    @Override
    public void onNext(Integer integer) {

    }

    @Override
    public void onError(Throwable e) {

    }

    @Override
    public void onComplete() {

    }
});
```

```
Observable.create(new ObservableOnSubscribe<Integer>() {
    @Override
    public void subscribe(ObservableEmitter<Integer> emitter) throws Exception {

    }
});
```

ASYNC CALLS

```
public Single<T> makeCall(String url, String params){
    return Single.create(subscriber -> new HttpClient().makeCall(url, params, new Callback<T>() {
        @Override
        public void onSuccess(T t) {
            subscriber.onSuccess(t);
        }

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

```
new HttpClient<Integer>().makeCall( url: "http://monapi.com/clientId",  params: "")
        .subscribe(
                clientId -> {

                },
                error -> {

                }
        );
```

ASYNC CALLS

```
new HttpClient<Integer>().makeCall( url: "http://monapi.com/clientId",  params: "") Single<Integer>
        .flatMap(clientId -> new HttpClient<Client>().makeCall( url: "http://monapi.com/client/" + clientId,  params: "")) Single<Client>
        .filter(client -> client.getAge() > 30) Maybe<Client>
        .subscribe(
                client -> {

                }
        );
```

Bonus: each call for the client details is in parallel

Insert <wow.gif> here

ERROR HANDLING

```
public class Cache<K, V> {

    private final HashMap<K, V> cache = new HashMap<>();

    public Single<V> get(K key){
        return Single.fromCallable(() -> {
            V cachedObject = cache.get(key);
            if(cachedObject != null) {
                return cachedObject;
            }
            throw new Exception("Object not in cache");
        });
    }

    public Observable<V> getAll(){
        return Observable.fromIterable(cache.values());
    }

    public Completable add(K key, V value){
        return Completable.fromAction(() -> {
            synchronized (cache) {
                cache.put(key, value);
            }
        });
    }
}
```

```
cache.get(2)
        .onErrorResumeNext(error ->
                new HttpClient<String>().makeCall( url: "http://monapi.com/client/2", params: ""))
        .subscribe(
                s -> {},
                error -> {}
        );
```

Insert <wow.gif> here

LINKS

- http://reactivex.io/ -> Official website
- https://rxviz.com/ -> Animated playground for operators
- https://rxmarbles.com/ -> Interactive diagrams (RxJS)
- http://blog.jimbaca.com/essential-rxjava-guide-for-android-developers/

Name	Units of Data Produced
Observable	Multiple or None
Flowable	Multiple or None
Single	One
Maybe	One or None
Completable	None

```
Observable.interval( period: 1, TimeUnit.SECONDS) // by default this runs on the computation scheduler
        .map(it -> it * 2 ) // runs on the computation scheduler
        .observeOn(Schedulers.io())
        .filter(it -> it % 3L == 0L ) // runs on the io scheduler
        .observeOn(Schedulers.computation())
        .subscribe(); // runs on the computation scheduler
```

TEXTE

https://www.youtube.com/watch?v=HnbNcQlzV-4