(Functional) Reactive Programming with ReactiveCocoa

Florent Pillet

fpillet@gmail.com

@fpillet

Reactive programming?

Reactive programming is essentially working with **asynchronous data streams**

Data changes over **time** and flows in **streams** processed with an asynchronous, operator-based **logic**.

They are all **separate**, yet usually found at the same place in the code.

Build modern, highly interactive, dynamic applications

with minimal pain.

Learning curve is **steep**[1]

Need to let go of imperative habits, learn new abstractions and patterns

but...

[1] If you read only one, it should be The introduction to reactive programming you've been missing

Pays off big time

Modular

Reusable

Expressive

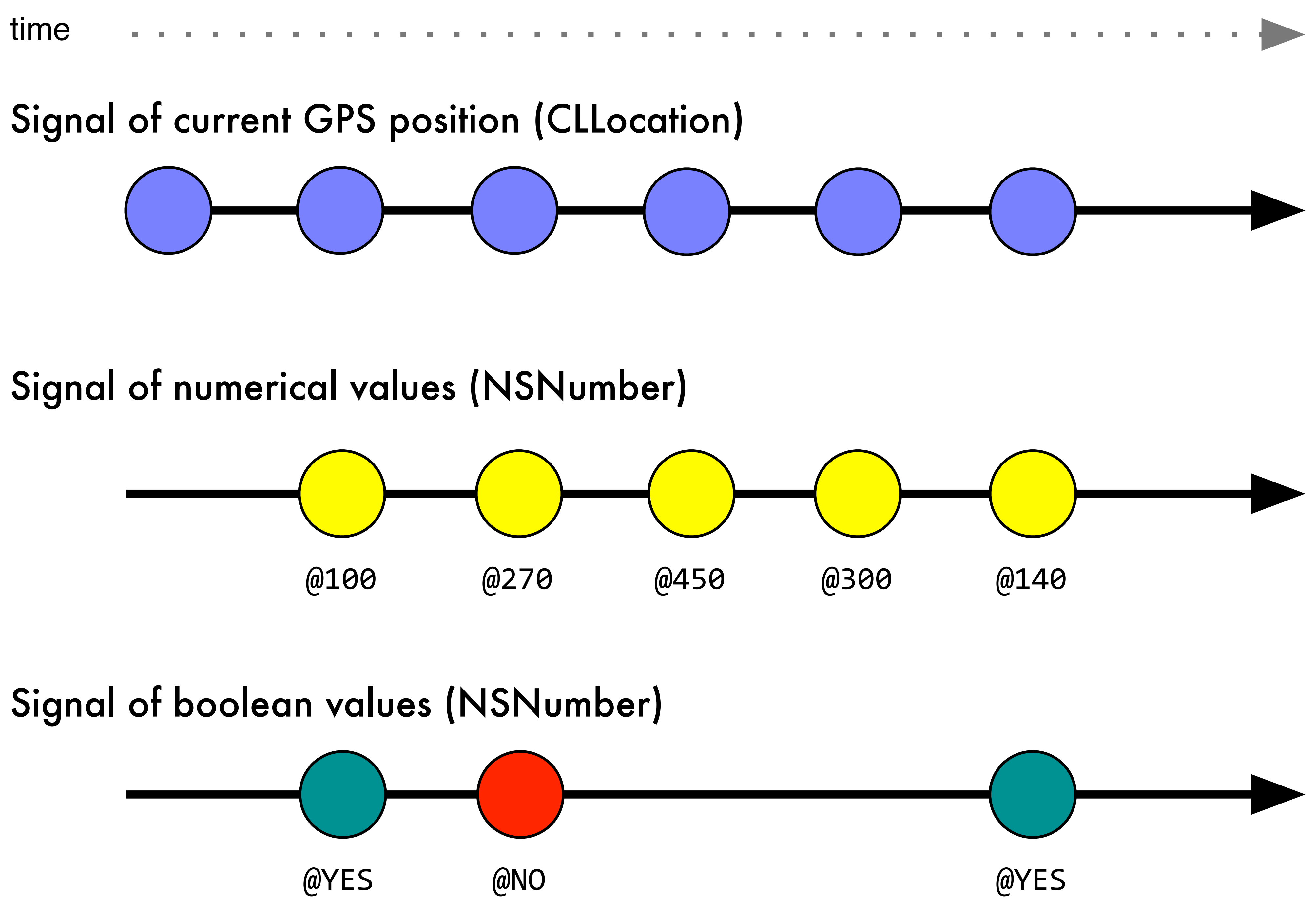
Testable code

Reactive programming paves the way to **eliminating state and mutability** from your code.[2]

[2] Suggested reading: Enemy of the state presentation by Justin Sparh-Summmers.

So what does this reactive thing look like?

Signals

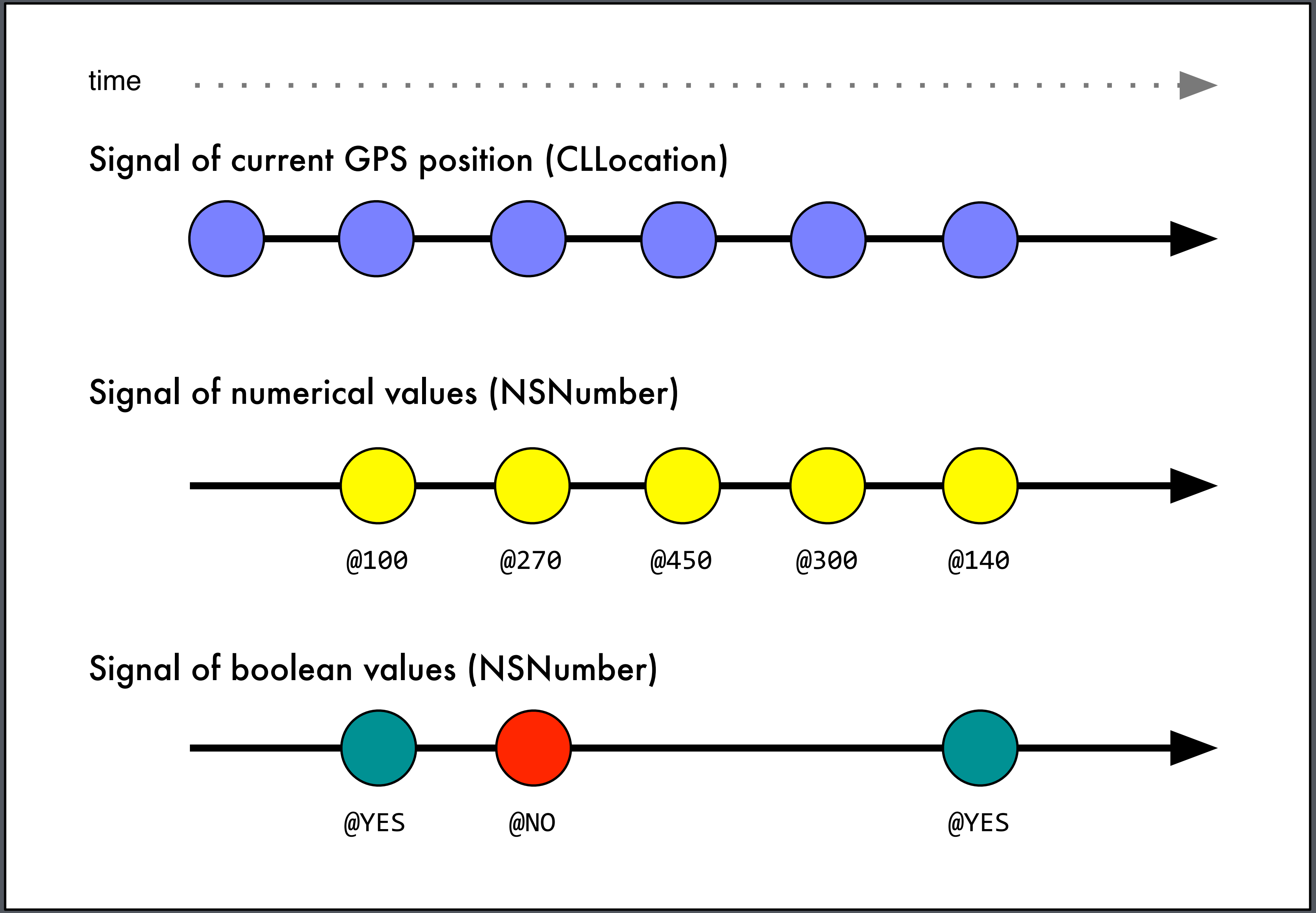

time
Signal of current GPS position (CLLocation)
Signal of numerical values (NSNumber)
@100
@270
@450
@300
@140
Signal of boolean values (NSNumber)
@YES
@NO
@YES

Signals send **events** over **time**

A signal is a push-driven stream.

It produces three types of events:

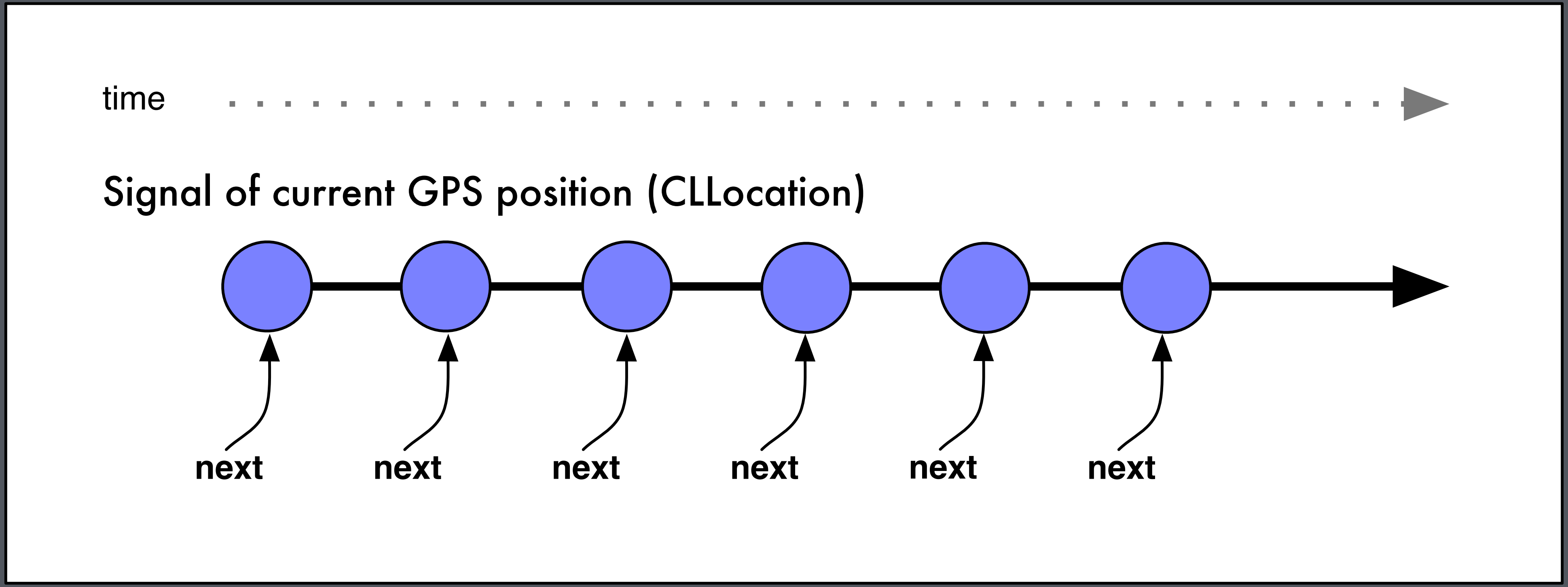
- **next** values
- an **error**
- a **completion**


time
Signal of current GPS position (CLLocation)
Signal of numerical values (NSNumber)
@100
@270
@450
@300
@140
Signal of boolean values (NSNumber)
@YES
@NO
@YES

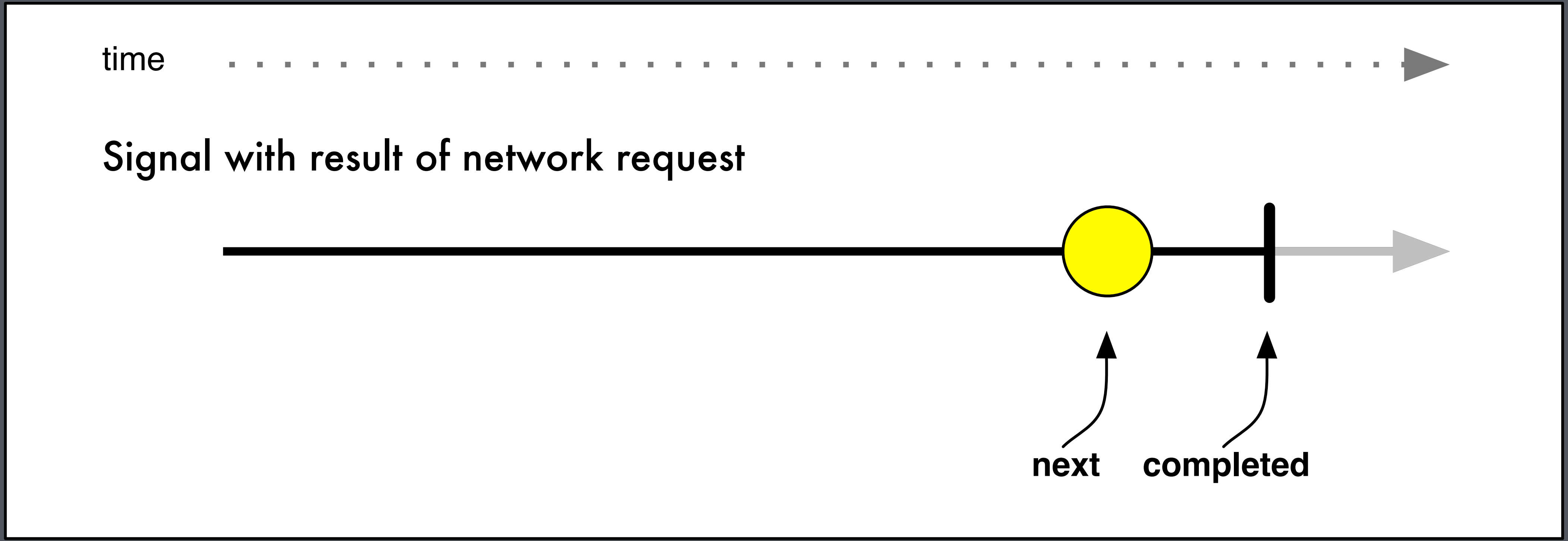
next values

A stream of values delivered over time.

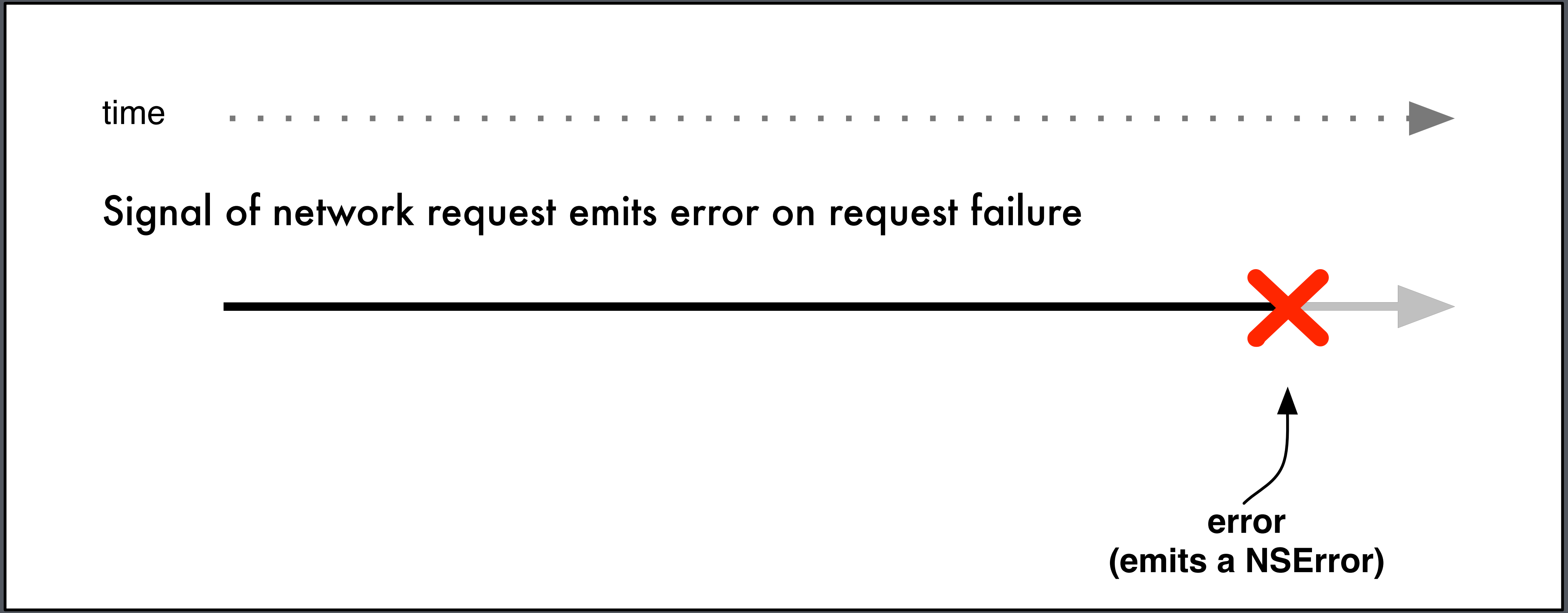
Signals produce zero, one or many **next** values before either **completing** or producing an **error**.



signal **completion**



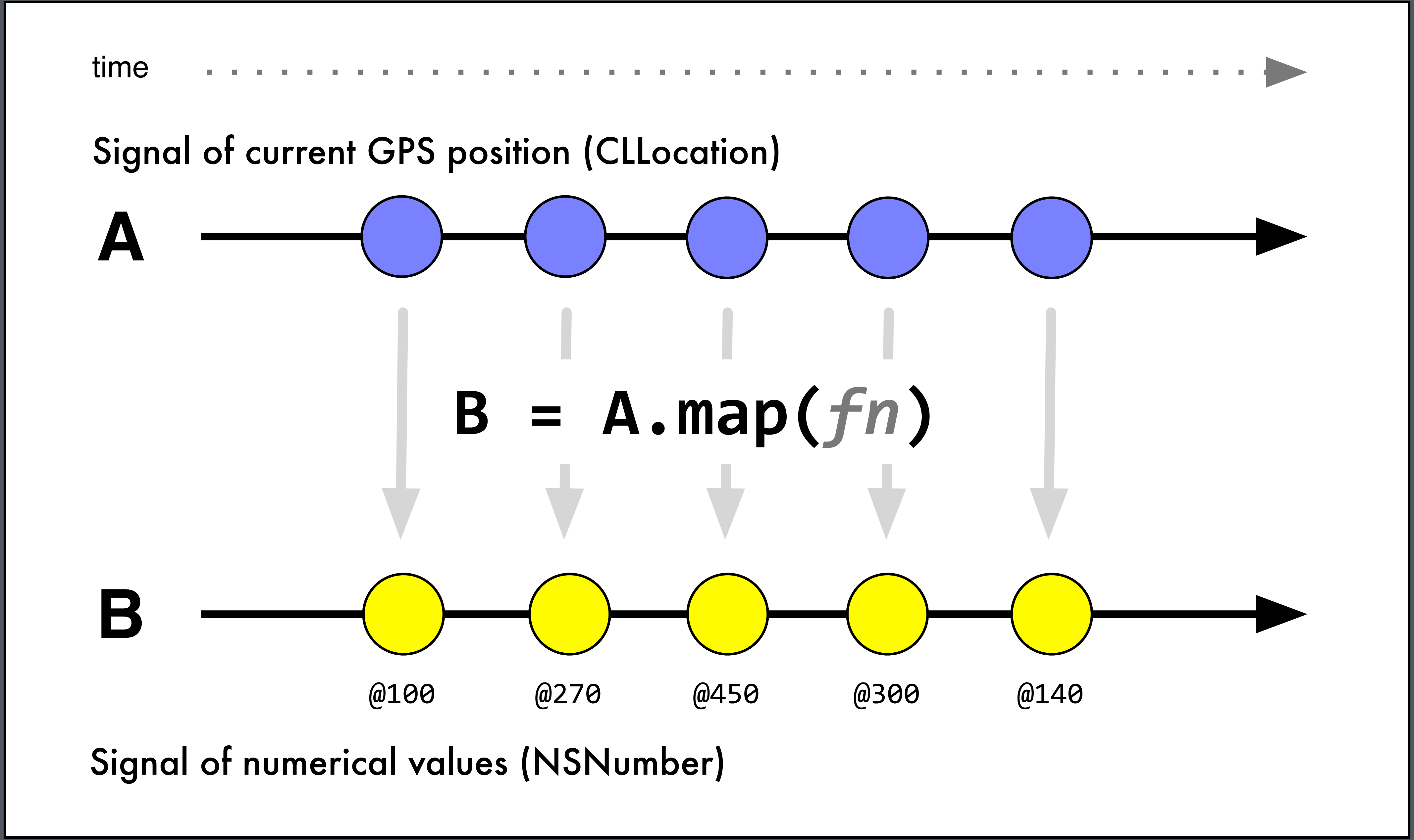
signal **error**



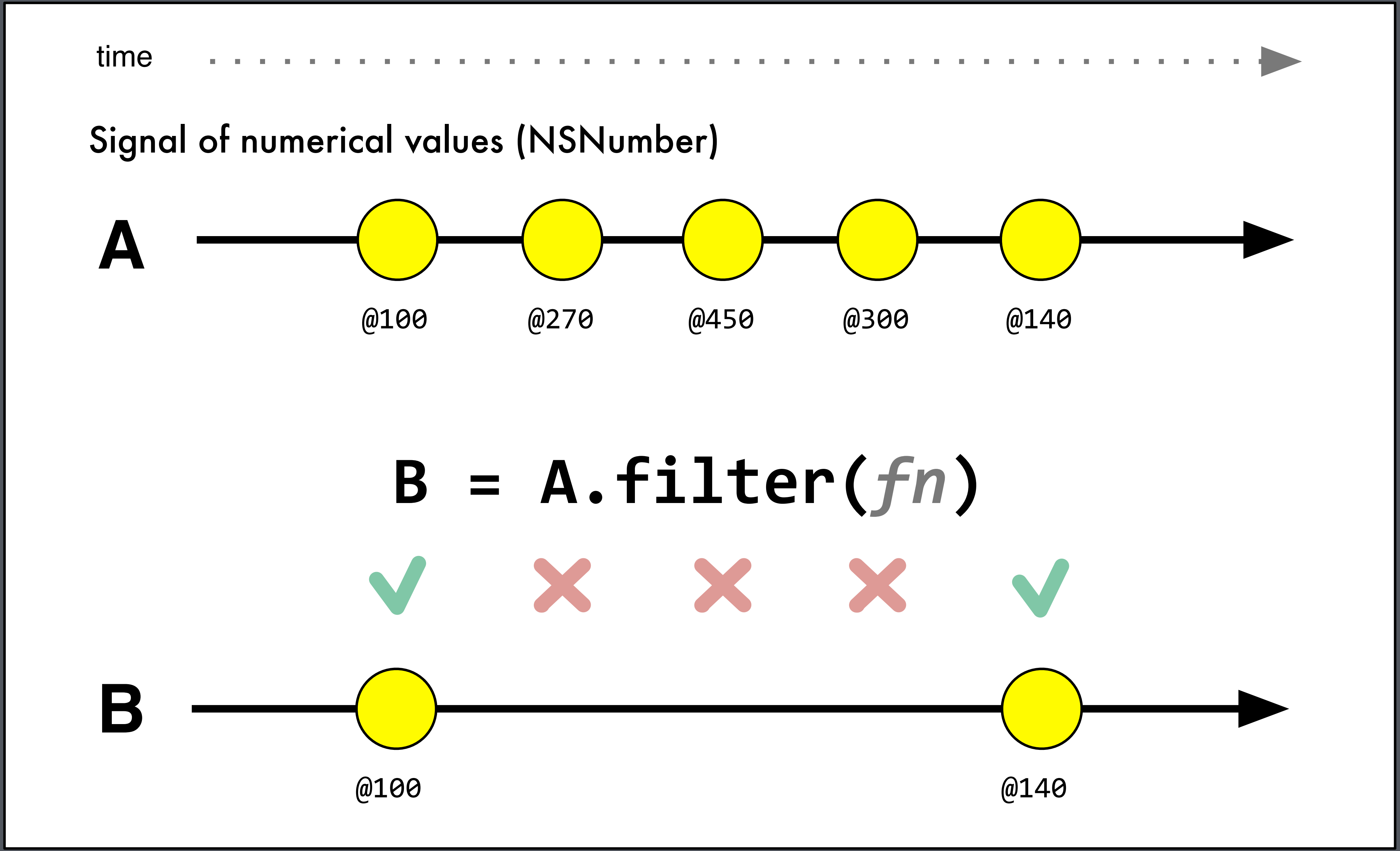
Operations

- A signal can have multiple subscribers
- Operations subcribe to a signal and return a new signal
- Easy multi-staged transformations
- Async, async, async!

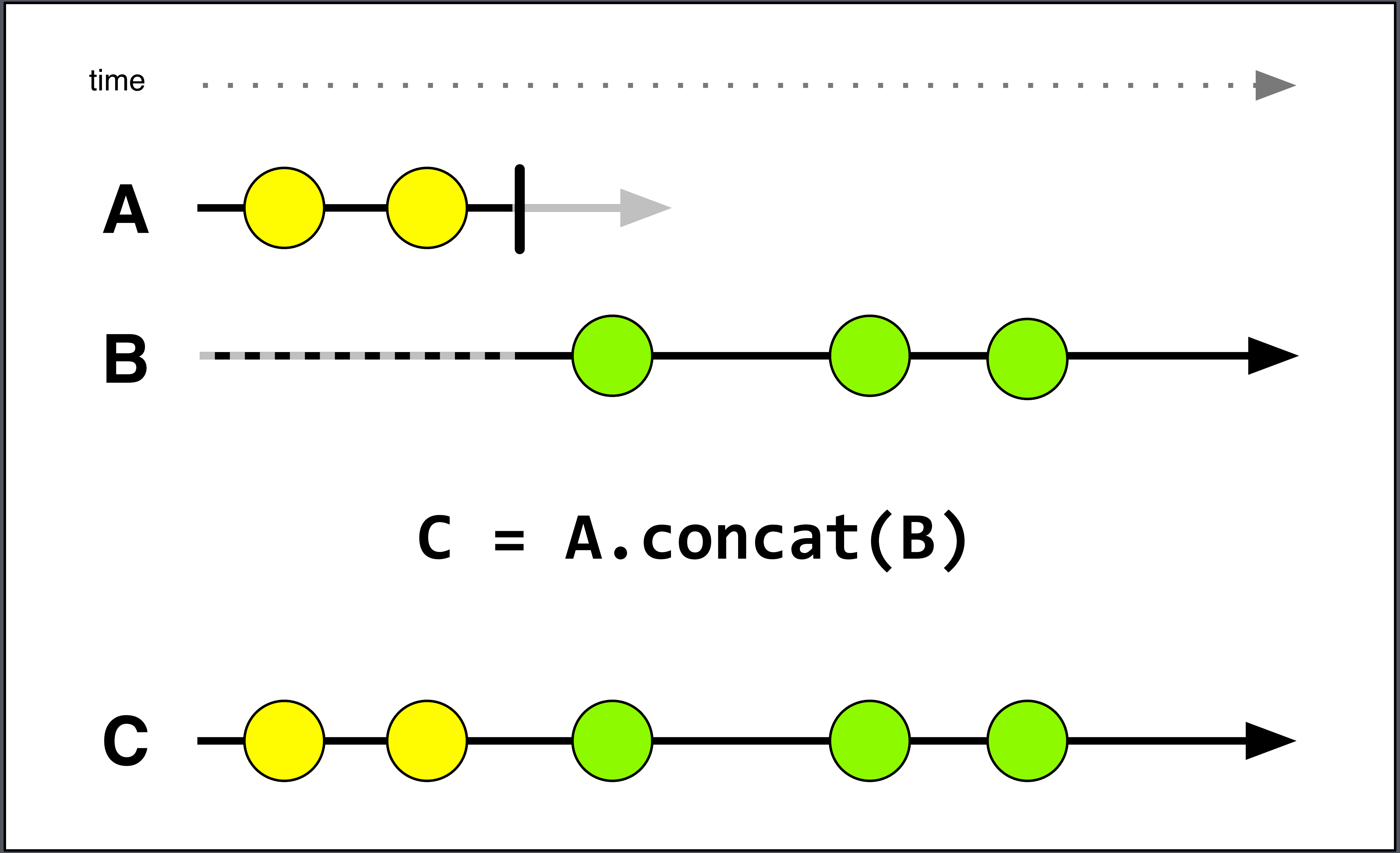
map


time
Signal of current GPS position (CLLocation)
A
B = A.map(fn)
B
@100
@270
@450
@300
@140
Signal of numerical values (NSNumber)

filter

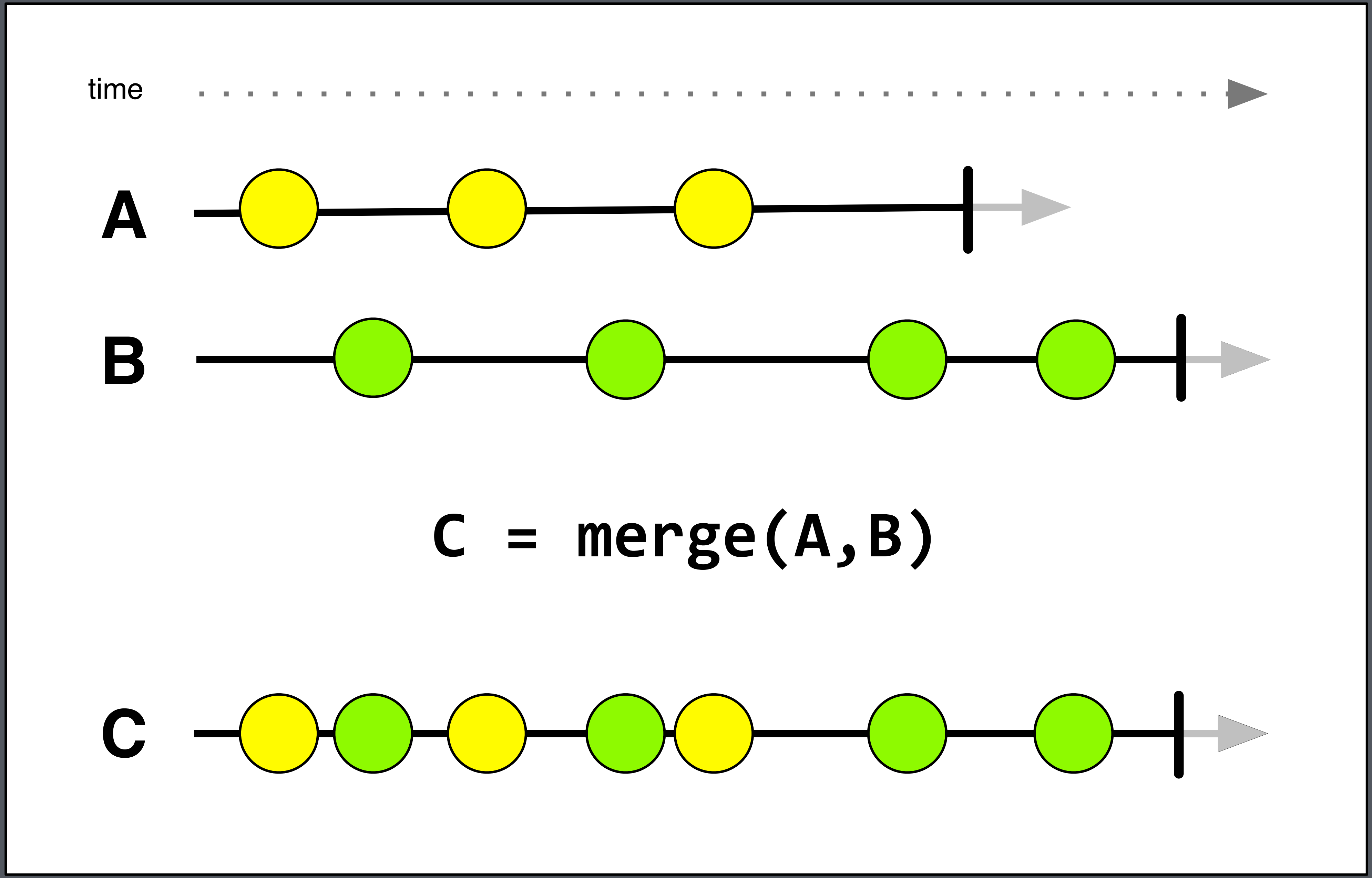

time
Signal of numerical values (NSNumber)
A
@100
@270
@450
@300
@140
B = A.filter(fn)
✓
✗
✗
✗
✓
B
@100
@140

concat

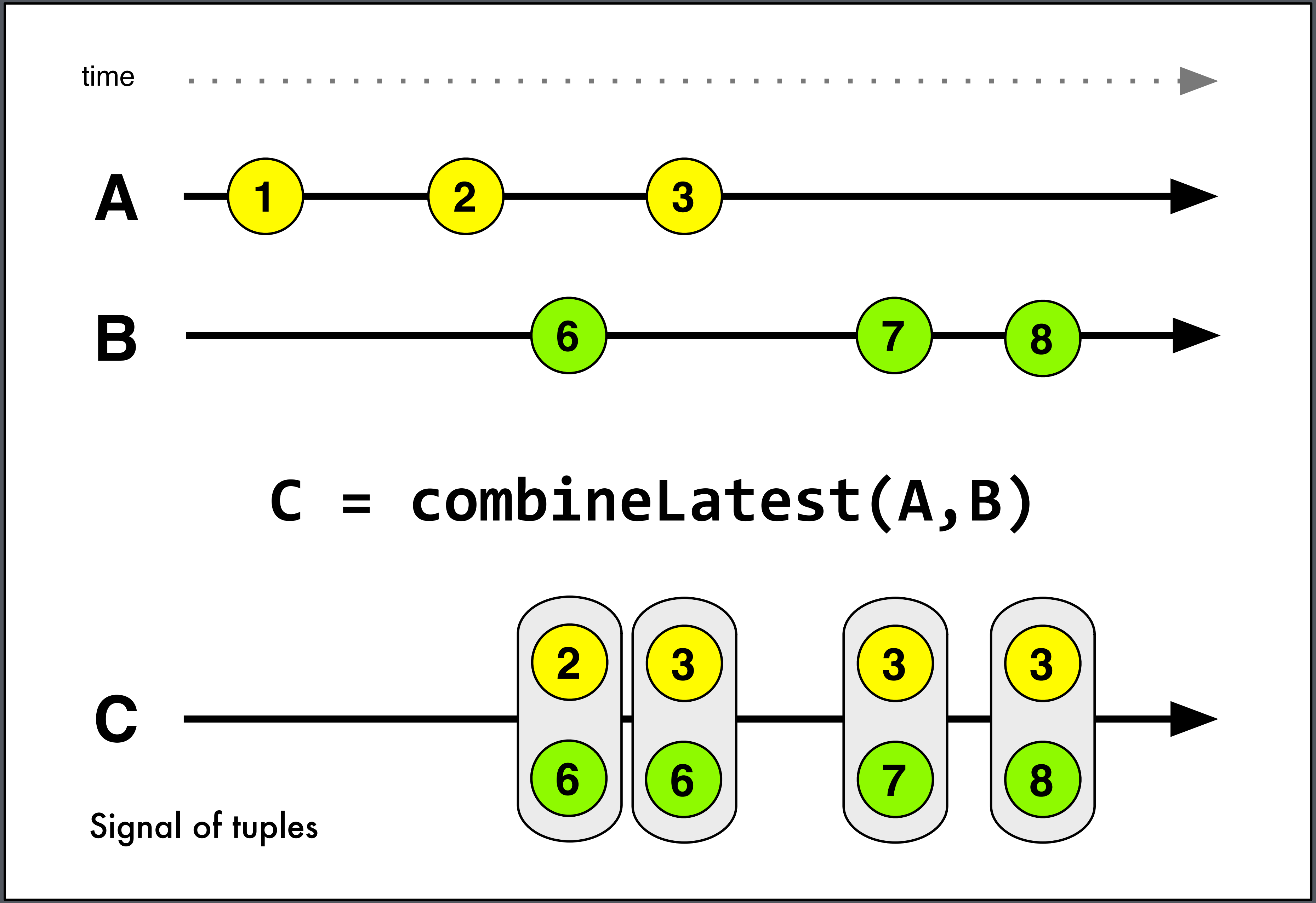


```
C = A.concat(B)
```

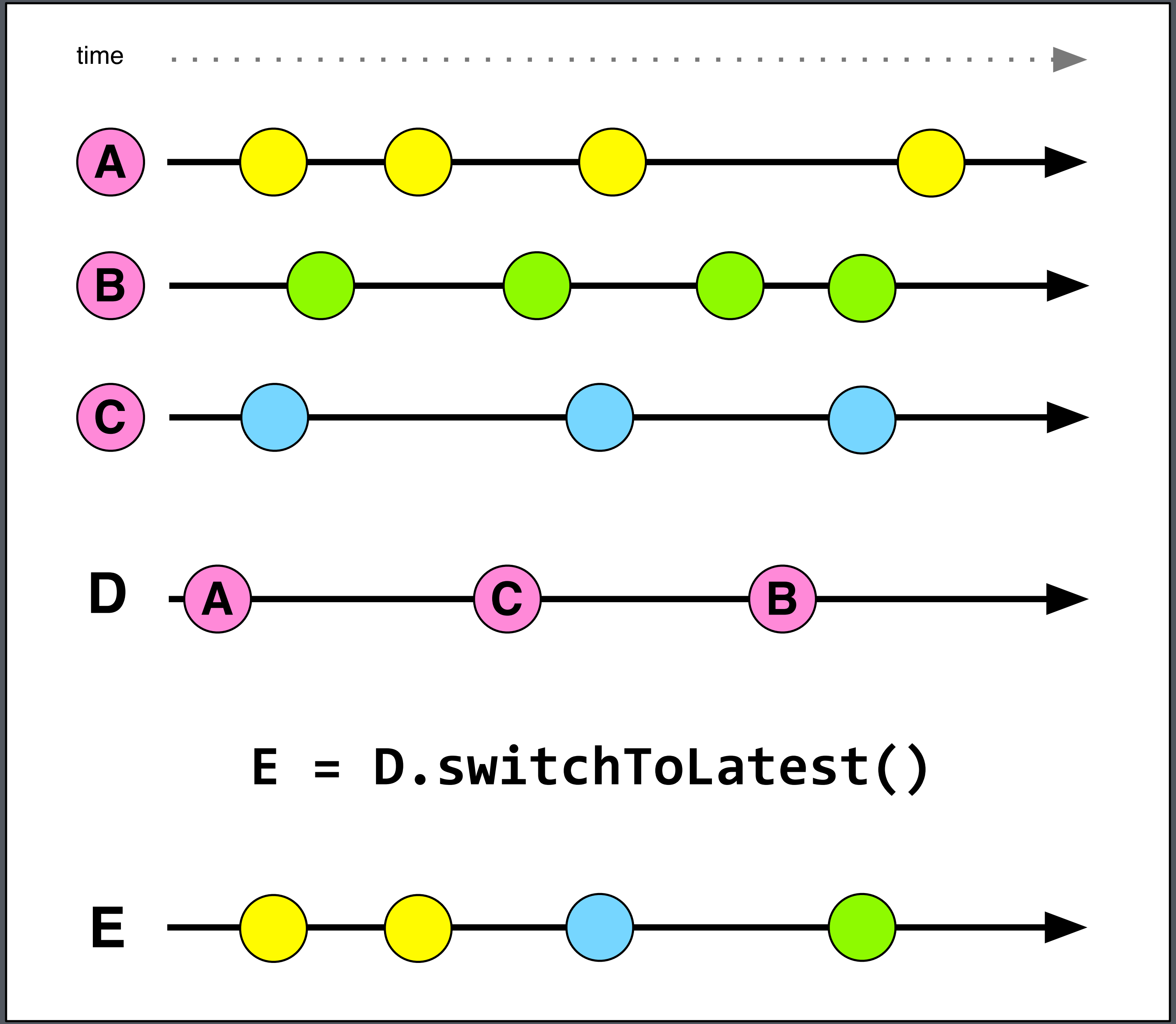
merge



combineLatest



switchToLatest



Other operations

- `flatten  flattenMap  zip  zipWith  startWith`
- `delay  throttle  sample timeout`
- `take  takeUntil  skip  ignoreValues`
- `try  catch  then  switch  if  and  or  not`
- `replay  replayLast  repeat  retry`

… and a **lot** more …

ReactiveCocoa

Main classes in ReactiveCocoa

- Signals: `RACSignal`
- Tuples: `RACTuple`
- Disposables: `RACDisposable`
- Schedulers: `RACScheduler`
- Commands: `RACCommand` (not covered here)

Subscribing to signals

```
[someSignal subscribeNext:^(id nextValue) {

    // this is called asynchronously, every time a new value
    // is available on the signal

    NSLog(@"Signal emitted new value= %@", nextValue);

}];
```

Subscribing to signals

```
[someSignal subscribeNext:^(id nextValue) {

    NSLog(@"Signal emitted new value= %@", nextValue);

} error:^(NSError *error) {

    NSLog(@"Signal emitted error %@", error);

} completed:^{

    NSLog(@"Signal completed");

}];
```

Unsubscribing from signals

```
// subscribe
RACDisposable *disposable =
    [someSignal subscribeNext:^(id nextValue) {

        NSLog(@"Signal emitted new value= %@", nextValue);

    }];


// can cancel subscription at any time
[disposable dispose];
```

Creating signals

- Using KVO
- Transforming existing signals into a new signal
- Dynamically with a generator block
- Lifting from the imperative world
- Manually producing events

Signals from variables (KVO)

Use the **`RACObserve(object,path)`** macro to update our unread count label[3]:

```
[RACObserve(self.model, unreadCount) subscribeNext:^(NSNumber *value) {
    self.unreadCountLabel.text = [value stringValue];
}];
```

---

[3] Real developers use **NSNumberFormatter**

Transforming existing signals

```
- (RACSignal *)colorForUnreadCount {

    return [RACObserve(self.model,unreadCount) // this is a signal

        // 'map' subscribes to the signal above and returns
        // a new signal that sends a color for each new unreadCount

        map:^id(NSNumber *unread) {
            NSInteger count = unread.integerValue;
            return  count < 10 ? [UIColor blackColor] :
                    count < 20 ? [UIColor orangeColor] :
                                 [UIColor redColor];
        }];
}
```

Transforming existing signals

```
// using the signal created in the previous slide

[[model colorForUnreadCount] subscribeNext:^(UIColor *color) {

   self.unreadLabel.textColor = color;

}];

// a shorter way to write this (for simple binding cases)
// see <ReactiveCocoa/RACSusbscriptingAssignmentTrampoline.h>

RAC(self.unreadLabel, textColor) = model.colorForUnreadCount;
```

Dynamic signals

```
- (RACSignal *)post:(NSDictionary *)formData toURL:(NSURL *)url {

    return [RACSignal createSignal:^(id<RACSubscriber> subscriber)] {
        // use AFNetworking to post form data
        NSURLSessionDataTask *task = [self.sessionManager POST:url parameters:data
                success:^(NSURLSessionDataTask *t, NSDictionary *responseObject) {
                    if (responseObject)
                        [subscriber sendNext:responseObject];
                    [subscriber sendCompleted];
                }
                failure:^(NSURLSessionDataTask *t, NSError *error) {
                    [subscriber sendError:error];
                }
            ];

        return [RACDisposable disposableWithBlock:^{
            [task cancel];
        }];
    }];
}
```

Dynamic signals

```
// use signal defined in previous slide
RACSignal *postSignal = [manager post:@{@"name": @"Florent"} toURL:someURL];

[postSignal subscribeNext:^(NSDictionary *response) {

    NSLog(@"Server answered POST with %@", response);

} error:^(NSError *error) {

    NSLog(@"POST failed with error: %@", error);

} completed:{

    NSLog(@"POST was successful");

}]
```

Lifting to the reactive world

```
[[[[[self
    rac_signalForSelector:@selector(locationManager:didRangeBeacons:inRegion:)
             fromProtocol:@protocol(CLLocationManagerDelegate)]

    reduceEach:^(CLLocationManager *manager, NSArray *beacons, CLBeaconRegion *region) {
        return [[beacons sortedArrayUsingFunction:proximityComparator context:NULL]
                        firstObject] ?: [NSNull null];
    }]

    filter:^BOOL(id value) {
        return [value isKindOfClass:[CLBeacon class]];
    }]

    distinctUntilChanged]

    subscribeNext:^(CLBeacon *beacon) {
        NSLog(@"Last closest beacon: %@.%@", beacon.major, beacon.minor);
    }];
```

Manual signals

```
@property (strong) RACSubject *manualSignal;

- (id)init {
    if (self = [super init]) {
        self.manualSignal = [[RACSubject alloc] init];
    }
    return self;
}

- (void)dealloc {
    [self.manualSignal sendCompleted];
}

- (RACSignal *)dataSignal {
    return self.manualSignal;
}

- (void)newDataObtained:(id)data {
    [self.manualSignal sendNext:data];
}
```

Manual signals

Note that:

- `**RACSubject**` doesn't automatically emit a `completed` event on dealloc. You must do it manually.
- Use `**RACReplaySubject**` to create a subject that can resend one or more of the last `next` values to new subscribers.
- Avoid using subjects if you have alternatives.

Disposables

Any subscription returns a **`RACDiposable`**. Use it to cancel the subscription

Schedulers

- Based on serial queues
- Makes **cancellation** easy!
- Use for timers and to control delivery of signals

```
RACSignal *onMainThread =
    [signal deliverOn:[RACScheduler mainThreadScheduler]];

RACSignal *onSomeSerialQueue =
    [signal deliverOn:[[RACTargetQueueScheduler alloc]
                        initWithName:@"My queue scheduler"
                        targetQueue:someSerialQueue]]
```

Schedulers

```
// A one-time timer that fires after 1 second on main thread

RACDisposable *timer = [[RACScheduler mainThreadScheduler]
        afterDelay:1.0
          schedule:^{
            NSLog(@"Delayed logging");
        }];

// We can cancel this at any time

[timer dispose];
```

Schedulers

```
// A cancellable periodic action

RACDisposable *timer =  [[RACScheduler schedulerWithPriority:RACSchedulerPriorityDefault]
         after:[NSDate dateWithTimeIntervalSinceNow:1.0]
repeatingEvery:0.5
    withLeeway:0.1
      schedule:^{
          NSLog(@"Running periodic action on private queue");
      }];

// Later: stop repeating

[timer dispose];
```

Other ReactiveCocoa gems

- `@weakify @strongify @unsafeify`
- `@onExit`

```
#import <ReactiveCocoa/RACExtScope.h>

@weakify(self);

[signal subscribeNext:^(id value) {

    @strongify(self);

    [self doSomethingWith:value];

}];
```

… and there is **a lot more** …

Commands, sequences, signal multicasting, side effects, channels, backtraces & debugging features, event materialization and dematerialization, testing are among topics not covered here.

Framework source code and the docset for Dash are useful resources.

More usage examples

```
- (RACSignal *)numberOfUnreadItems
{
    @weakify(self);
    return [[[[[self
            itemsUpdated]
            startWith:@YES]
            map:^(id updated) {
                @strongify(self);
                return self.unreadItemsCount;
            }]
            distinctUntilChanged]
            deliverOn:RACScheduler.mainThreadScheduler];
    }];
}
```

More usage examples

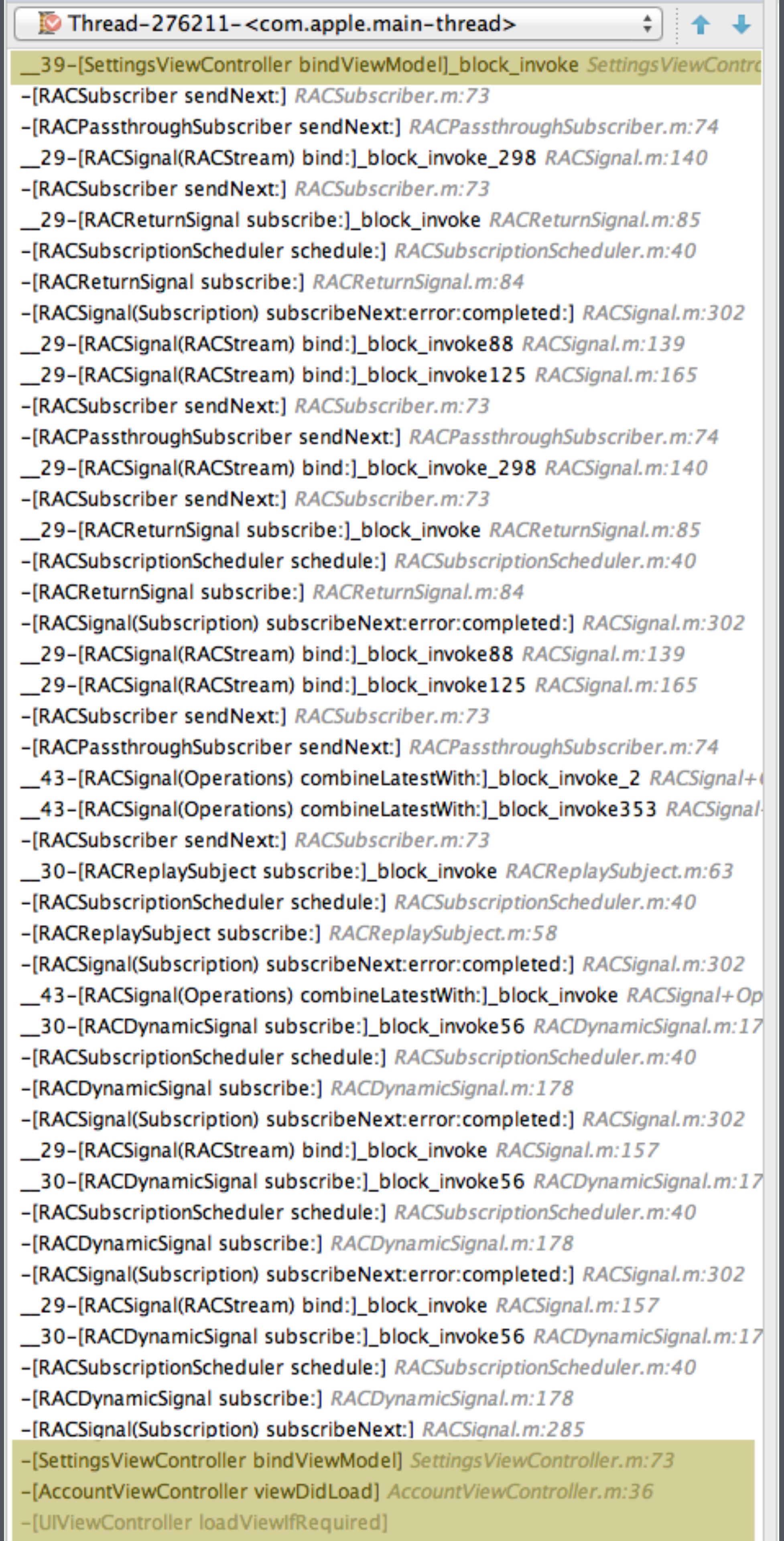
```
// Automatically update a badge on tab bar
// when the count of unread items changes

- (void)keepNewsItemUpToDate:(UITabBarItem *)newsItem {
    @weakify(newsItem);
    [self.model.numberOfUnreadItems subscribeNext:^(NSNumber *count) {
        @strongify(newsItem);
        if (count.integerValue)
            newsItem.badge = count.stringValue;
        else
            newsItem.badge = @"";
    }];
}
```

Links

- ReactiveCocoa framework
- A demo project: ReactiveWeather
- The introduction to reactive programming you've been missing
- Enemy of the state
- Reactive MVVM (Model-View-ViewModel): perfect match

Also look up 'Reactive' on Github and filter by langage (Obj-C).



```
Thread-276211-<com.apple.main-thread>
__39-[SettingsViewController bindViewModel]_block_invoke SettingsViewContro
-[RACSubscriber sendNext:] RACSubscriber.m:73
-[RACPassthroughSubscriber sendNext:] RACPassthroughSubscriber.m:74
__29-[RACSignal(RACStream) bind:]_block_invoke_298 RACSignal.m:140
-[RACSubscriber sendNext:] RACSubscriber.m:73
__29-[RACReturnSignal subscribe:]_block_invoke RACReturnSignal.m:85
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACReturnSignal subscribe:] RACReturnSignal.m:84
-[RACSignal(Subscription) subscribeNext:error:completed:] RACSignal.m:302
__29-[RACSignal(RACStream) bind:]_block_invoke88 RACSignal.m:139
__29-[RACSignal(RACStream) bind:]_block_invoke125 RACSignal.m:165
-[RACSubscriber sendNext:] RACSubscriber.m:73
-[RACPassthroughSubscriber sendNext:] RACPassthroughSubscriber.m:74
__29-[RACSignal(RACStream) bind:]_block_invoke_298 RACSignal.m:140
-[RACSubscriber sendNext:] RACSubscriber.m:73
__29-[RACReturnSignal subscribe:]_block_invoke RACReturnSignal.m:85
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACReturnSignal subscribe:] RACReturnSignal.m:84
-[RACSignal(Subscription) subscribeNext:error:completed:] RACSignal.m:302
__29-[RACSignal(RACStream) bind:]_block_invoke88 RACSignal.m:139
__29-[RACSignal(RACStream) bind:]_block_invoke125 RACSignal.m:165
-[RACSubscriber sendNext:] RACSubscriber.m:73
-[RACPassthroughSubscriber sendNext:] RACPassthroughSubscriber.m:74
__43-[RACSignal(Operations) combineLatestWith:]_block_invoke_2 RACSignal+(
__43-[RACSignal(Operations) combineLatestWith:]_block_invoke353 RACSignal
-[RACSubscriber sendNext:] RACSubscriber.m:73
__30-[RACReplaySubject subscribe:]_block_invoke RACReplaySubject.m:63
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACReplaySubject subscribe:] RACReplaySubject.m:58
-[RACSignal(Subscription) subscribeNext:error:completed:] RACSignal.m:302
__43-[RACSignal(Operations) combineLatestWith:]_block_invoke RACSignal+Op
__30-[RACDynamicSignal subscribe:]_block_invoke56 RACDynamicSignal.m:17
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACDynamicSignal subscribe:] RACDynamicSignal.m:178
-[RACSignal(Subscription) subscribeNext:error:completed:] RACSignal.m:302
__29-[RACSignal(RACStream) bind:]_block_invoke RACSignal.m:157
__30-[RACDynamicSignal subscribe:]_block_invoke56 RACDynamicSignal.m:17
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACDynamicSignal subscribe:] RACDynamicSignal.m:178
-[RACSignal(Subscription) subscribeNext:error:completed:] RACSignal.m:302
__29-[RACSignal(RACStream) bind:]_block_invoke RACSignal.m:157
__30-[RACDynamicSignal subscribe:]_block_invoke56 RACDynamicSignal.m:17
-[RACSubscriptionScheduler schedule:] RACSubscriptionScheduler.m:40
-[RACDynamicSignal subscribe:] RACDynamicSignal.m:178
-[RACSignal(Subscription) subscribeNext:] RACSignal.m:285
-[SettingsViewController bindViewModel] SettingsViewController.m:73
-[AccountViewController viewDidLoad] AccountViewController.m:36
-[UIViewController loadViewIfRequired]
```

Scary interlude

Yellow parts are my app.

Not always like this.
This shouldn't put you off!

Summary

Reactive programming is a **logical** way to model and react to **asynchronous** information flow

Reactive code clearly[5] exposes **logic** and **transformations** deriving from new data

[5] finding the syntax weird? remember your first steps with Objective-C and all these square brackets...

Enforcing the separation between data producers and consumers, reactive code is **more testable**

Once trained to think reactively,

reducing state and mutability

is the next logical step towards code safety, stability and predictability.

ReactiveCocoa is a **rich** and **powerful** reactive programming framework that will bring your code to a new level

ReactiveCocoa is a **rich** and **powerful** reactive programming framework that will bring your code to a new level

works with Swift, too[4]

---

[4] See Colin Eberhardt's posts for Swift, ReactiveCocoa and MVVM pr0n.

Thank You !

Q & A

fpillet@gmail.com

@fpillet