Do-It-Yourself Functional Reactive Programming

Manuel M T Chakravarty
Applicative &
Tweag I/O

mchakravarty
TacticalGrace
justtesting.org
haskellformac.com

Demystifying Functional Reactive Programming

What is the purpose of Functional Reactive Programming?

“React to change in a structured and safe manner.”

Change

Change

is easy!

Change

is easy!

```
model.goalsArray[currentGoalIndex].name = newName
```

Change

is easy!

```
model.goalsArray[currentGoalIndex].name = newName
```

Structure?

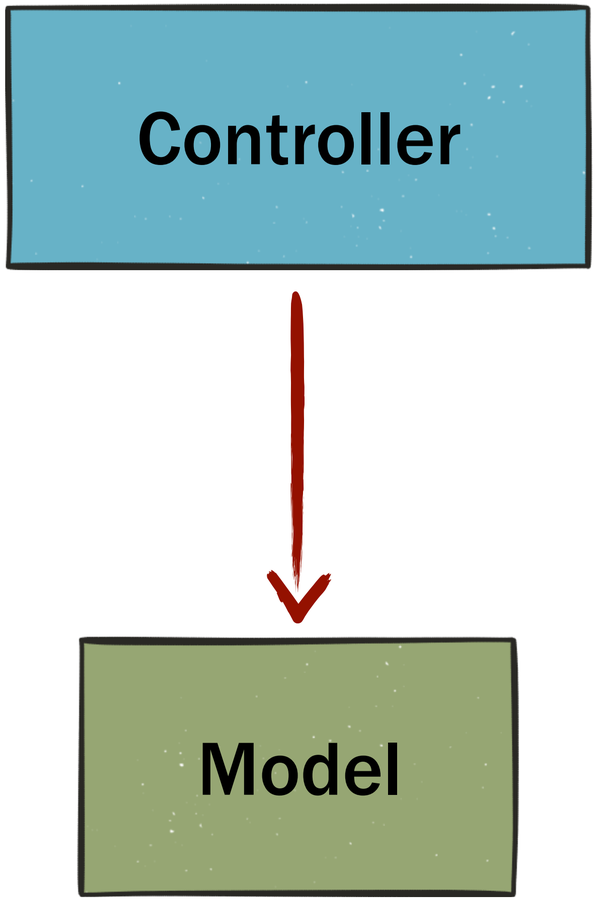
Safety?

Structure?

Safety?

Structure?

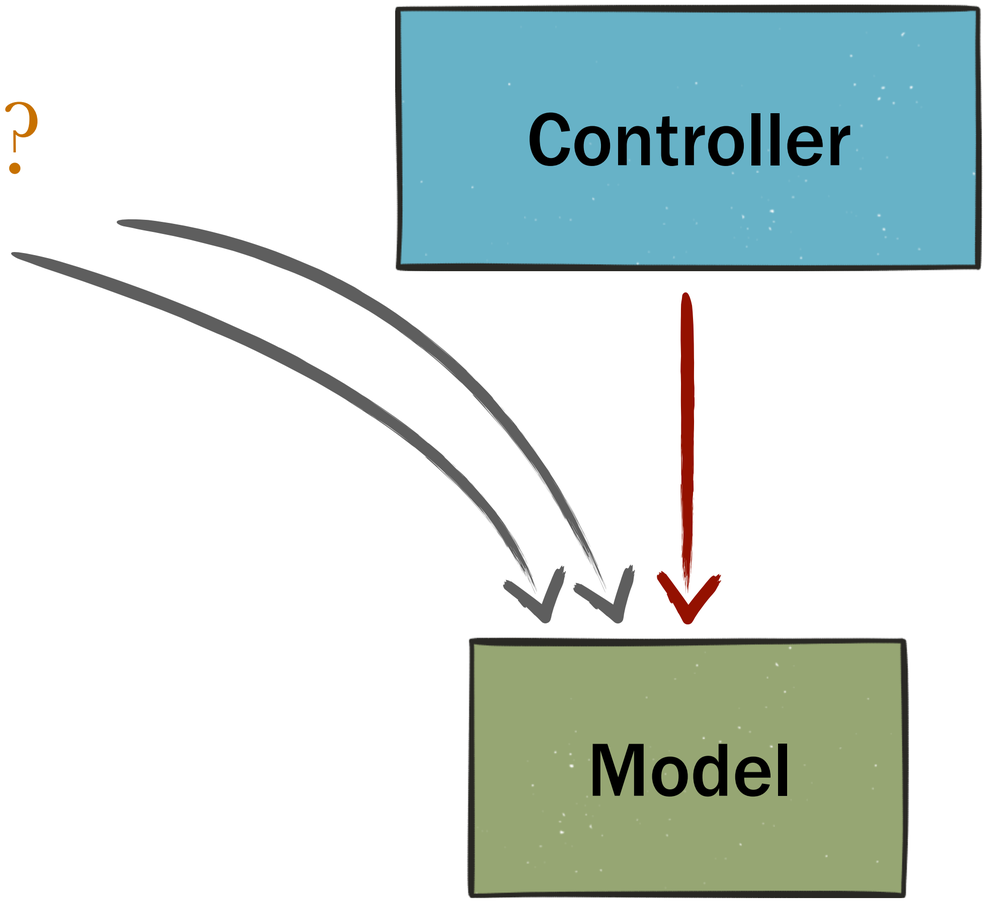
Safety?



Structure?

Safety?

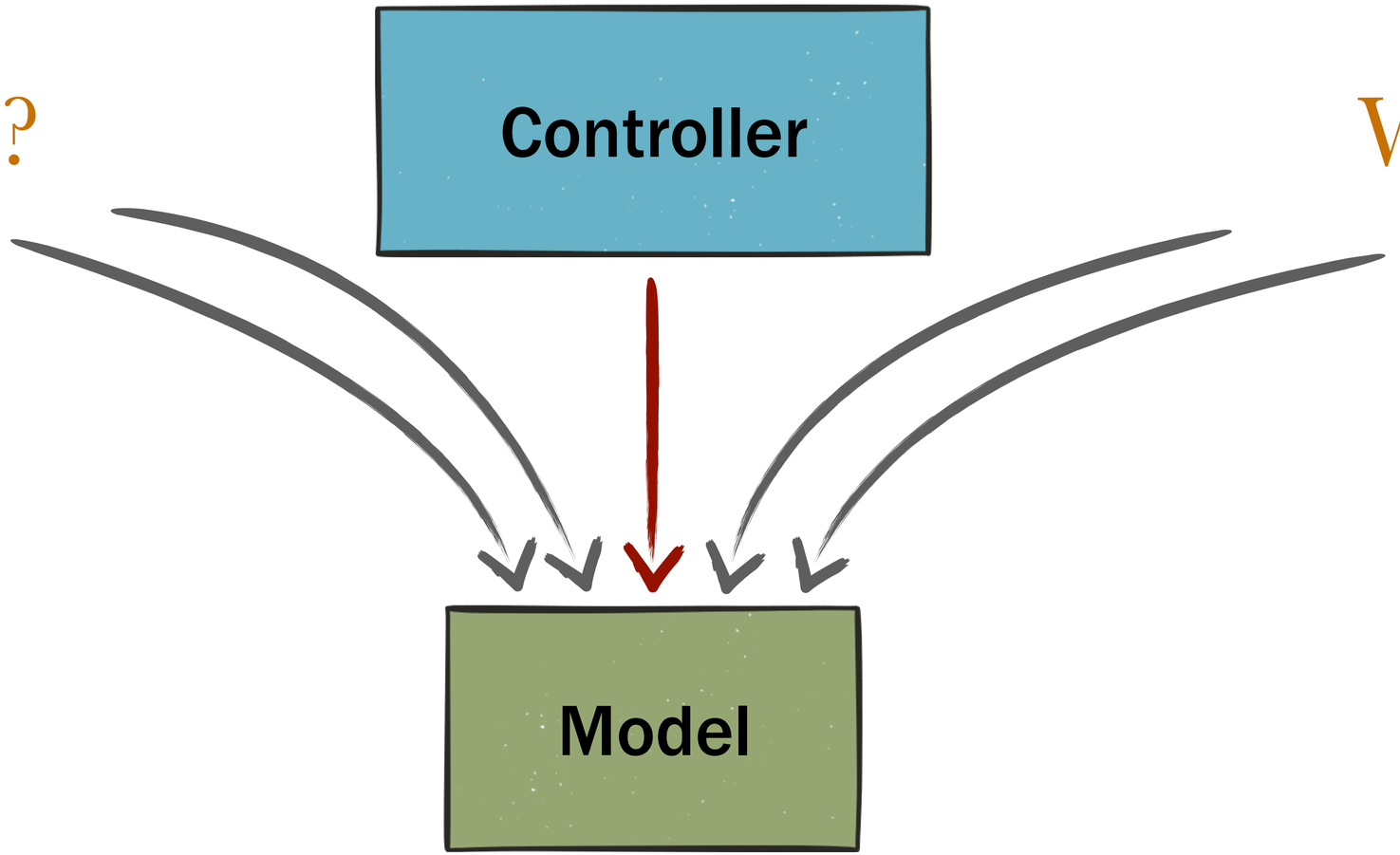
Where?



Structure?

Safety?

Where?



What?

Functional Reactive Programming

Change

Functional Reactive Programming

Localise

Assign only local properties

Change

Functional Reactive Programming

Localise

Assign only local properties

Structure

Explicit streams of change propagation

Change

Functional Reactive Programming

Localise

Assign only local properties

Structure

Explicit streams of change propagation

Type

Track kinds of changes with types

Change


Part 1
What is FRP?

What is Change?

Δt: 0s

Mutable Variable

Δt: 1s

Mutable Variable

Δt: 2s

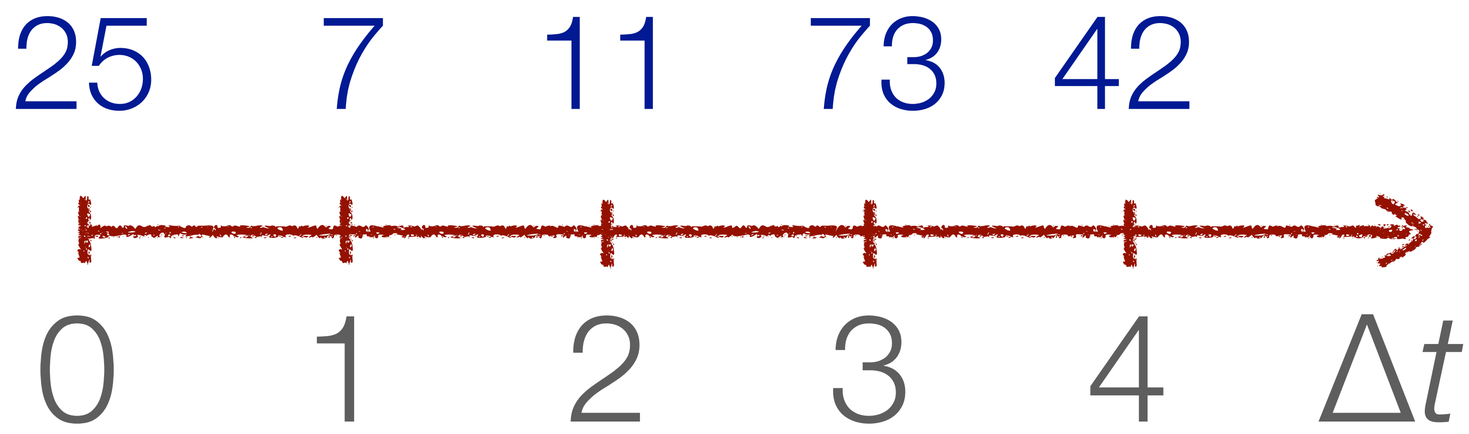
Mutable Variable

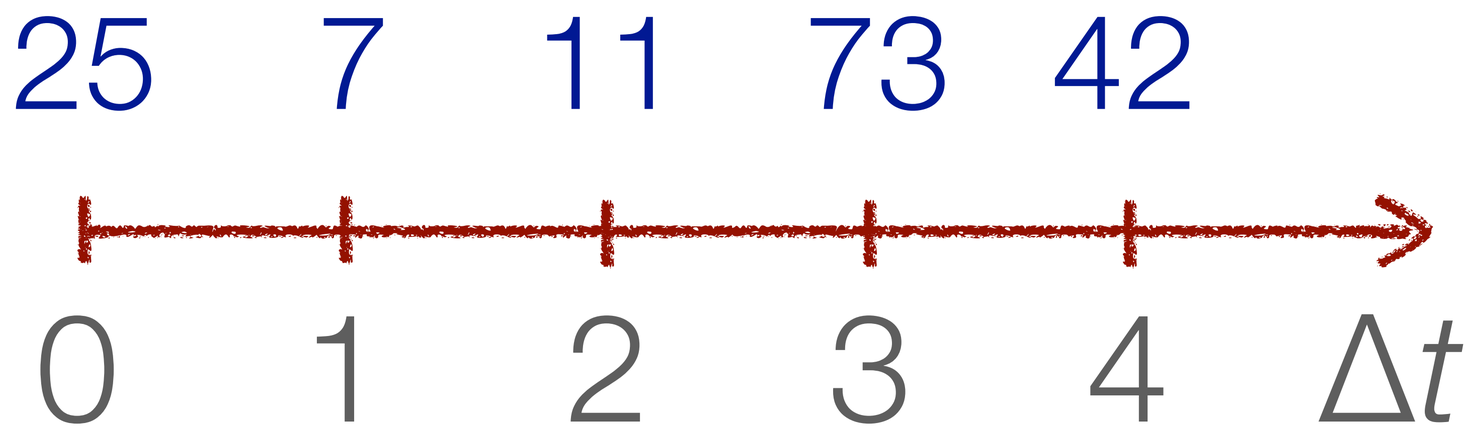
Δt: 3s

Mutable Variable

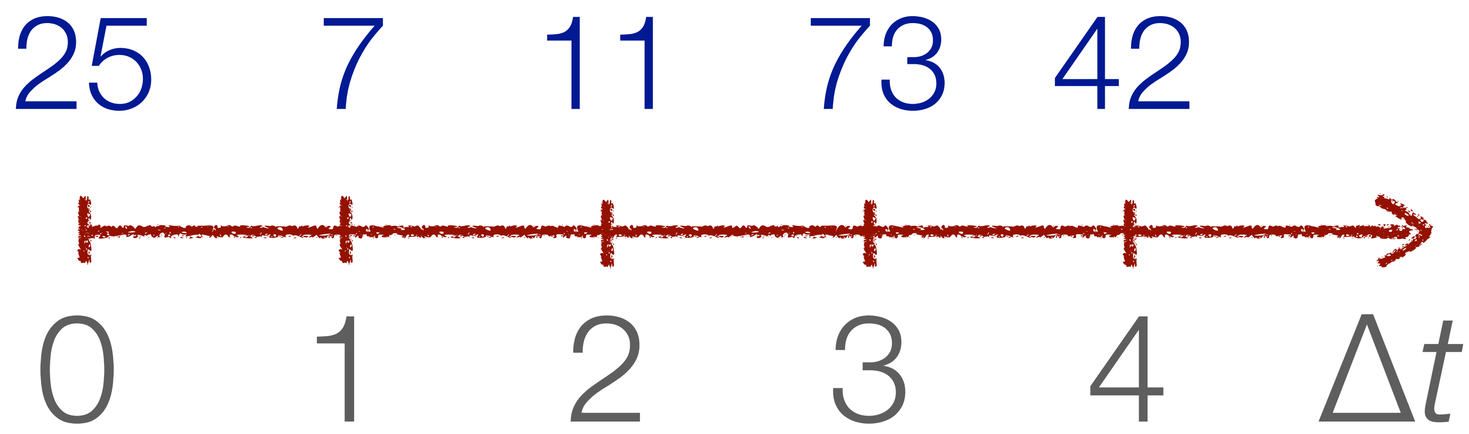
Δt: 4s

Mutable Variable


25
7
11
73
42
0
1
2
3
4
Δt



Variables change destructively



Variables change destructively

Timeline merely unfolds monotonically

Imperative Point of View

“Change is a value varying over time.”

Functional Point of View

"Change is a function from time
to a value."

Where It All Began: Continuous FRP

Where It All Began: Continuous FRP

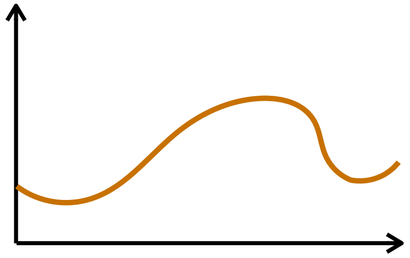
Conal Elliott

Paul Hudak

Where It All Began: Continuous FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```



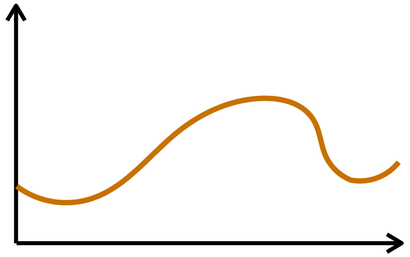
Conal Elliott

Paul Hudak

Where It All Began: Continuous FRP

Continuous behaviour

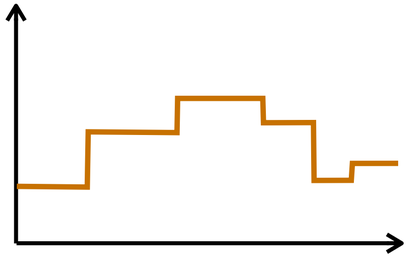
```
typealias Behaviour<a> = Time -> a
```



Conal Elliott

Discrete event stream

```
typealias Event<a> = Stream<(Time, a)>
```

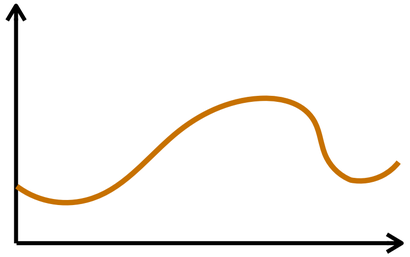


Paul Hudak

Where It All Began: Continuous FRP

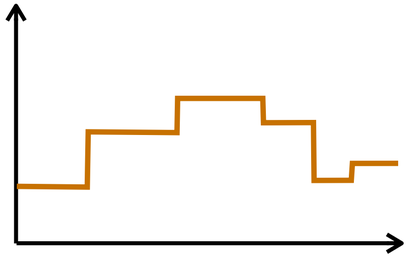
Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```



Discrete event stream

```
typealias Event<a> = Stream<(Time, a)>
```



Fran

Frob

FranTk

FVision

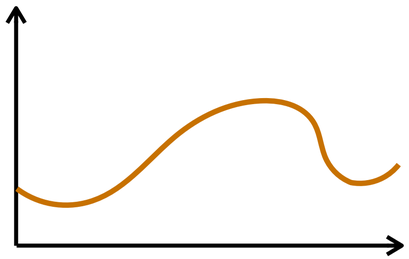
Conal Elliott

Paul Hudak

Improving Efficiency: Push-Pull FRP

Continuous behaviour

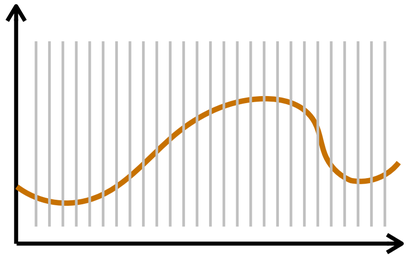
```
typealias Behaviour<a> = Time -> a
```



Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```

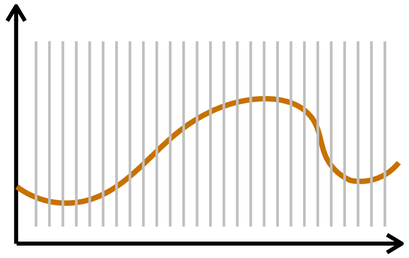


Requires sampling

Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```

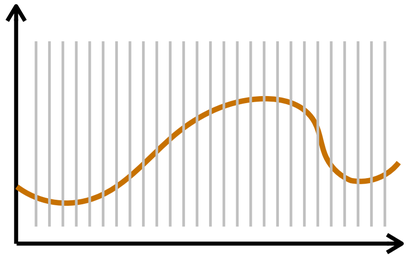


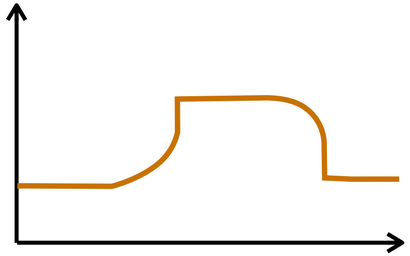
Requires sampling aka pull

Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```



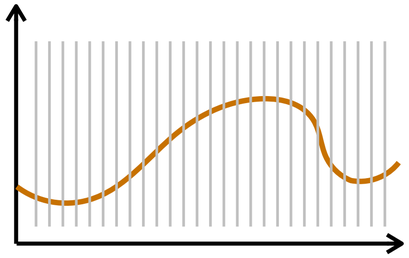


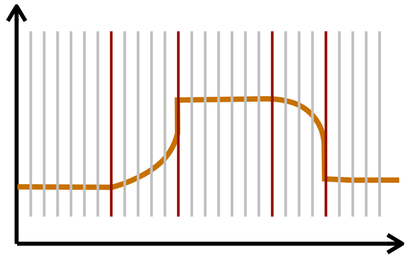
Requires sampling aka pull

Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```





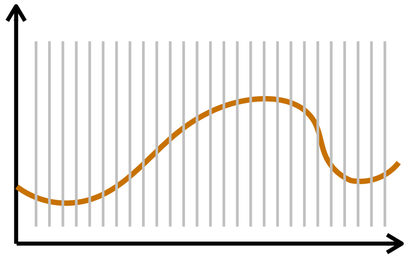
Requires sampling aka pull

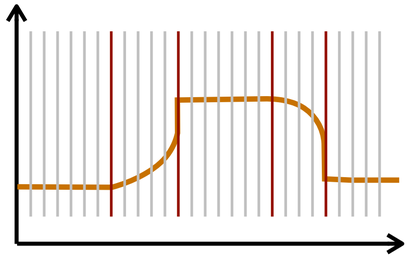
unnecessary sampling in some sections between events

Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```





Requires sampling aka pull

unnecessary sampling in some sections between events

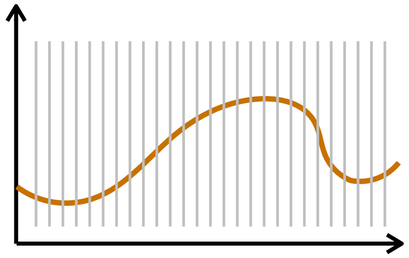
Reactive behaviour

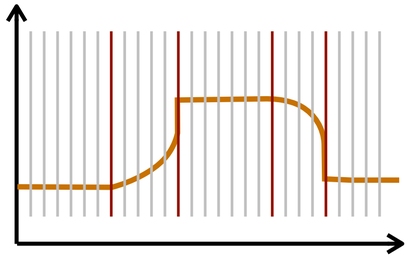
```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
```

Improving Efficiency: Push-Pull FRP

Continuous behaviour

```
typealias Behaviour<a> = Time -> a
```





Requires sampling aka pull

unnecessary sampling in some sections between events

Reactive behaviour

```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
struct Event<Thing> { let when:  Time
                      let thing: () -> Reactive<Thing> }
```

```
typealias Behaviour<a> = Time -> a
```

```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
struct Event<Thing> { let when:  Time
                      let thing: () -> Reactive<Thing> }
```

```
typealias Behaviour<a> = Time -> a
```

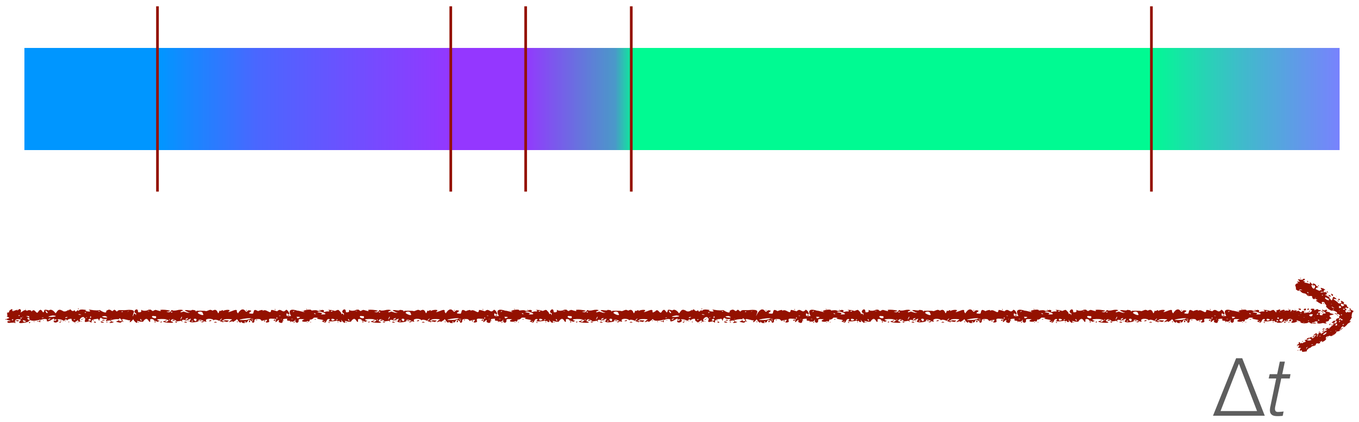
```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
struct Event<Thing> { let when:  Time
                      let thing: () -> Reactive<Thing> }
```

Reactive<Behaviour<UIColor>>

```
typealias Behaviour<a> = Time -> a
```

```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
struct Event<Thing> { let when:  Time
                      let thing: () -> Reactive<Thing> }
```

Reactive<Behaviour<UIColor>>

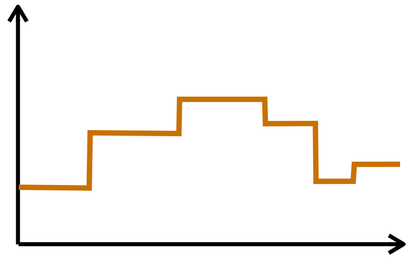


Efficiency Over Expressiveness: Discrete/Event FRP

```
struct Reactive<Thing> { let value: Thing
                         let next:  Event<Thing> }
struct Event<Thing> { let when:  Time
                      let thing: () -> Reactive<Thing> }
```

Efficiency Over Expressiveness: Discrete/Event FRP

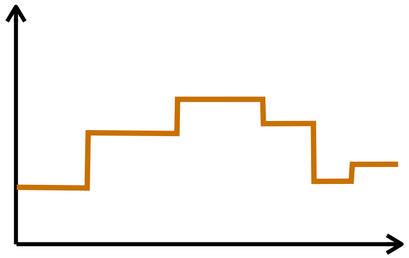
```
struct Reactive<Value> { let value: Value
                         let next:  Event<Value> }
struct Event<Value> { let when:  Time
                      let thing: () -> Reactive<Value> }
```



Values are constant
between events

Efficiency Over Expressiveness: Discrete/Event FRP

```
struct Reactive<Value> { let value: Value
                         let next:  Event<Value> }
struct Event<Value> { let when:  Time
                      let thing: () -> Reactive<Value> }
```



Values are constant between events

Implicit time

continuation

```
struct Reactive<Value> { let value: Value
                         let next:  Event<Value> }
struct Event<Value> { let cont: (Reactive<Value> -> ()) -> () }
```

Observer FRP

Observer FRP

From continuations to observers

Observer FRP

From continuations to observers

Rx

ReactiveCocoa

RxSwift

ReactiveSwift

Observer FRP

From continuations to observers

Rx

ReactiveCocoa

RxSwift

ReactiveSwift

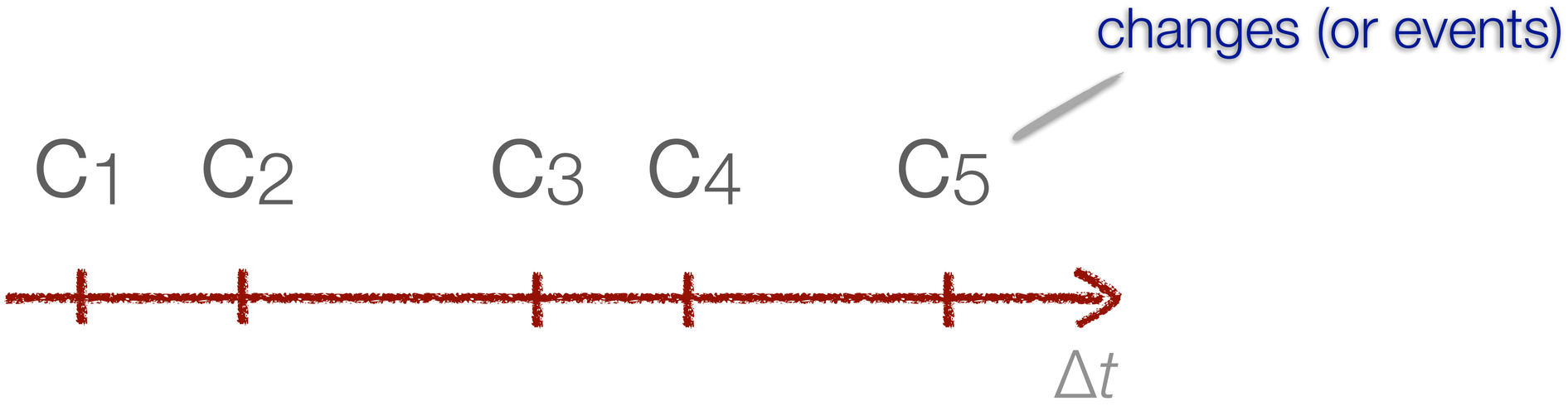
Adding significant complexity

Part 2

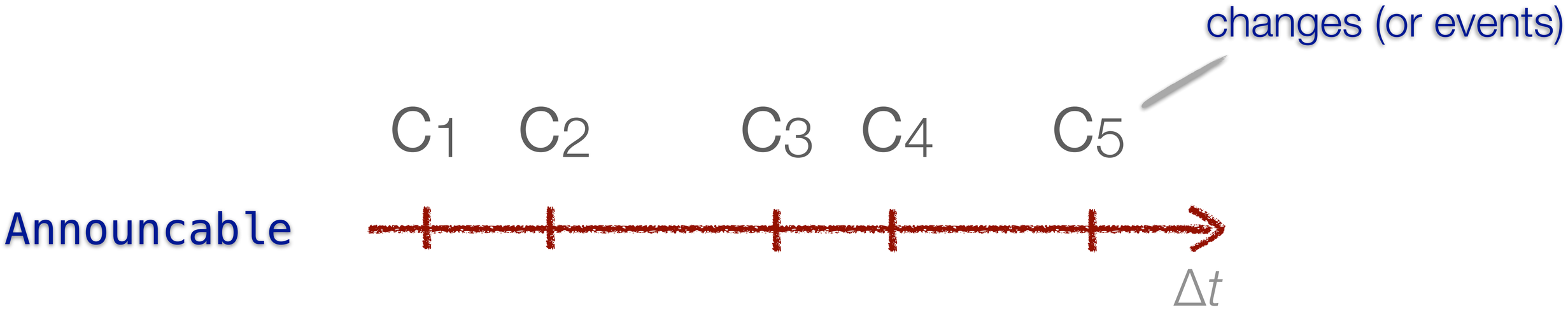
Implementing FRP

Ephemeral Stream of Changes

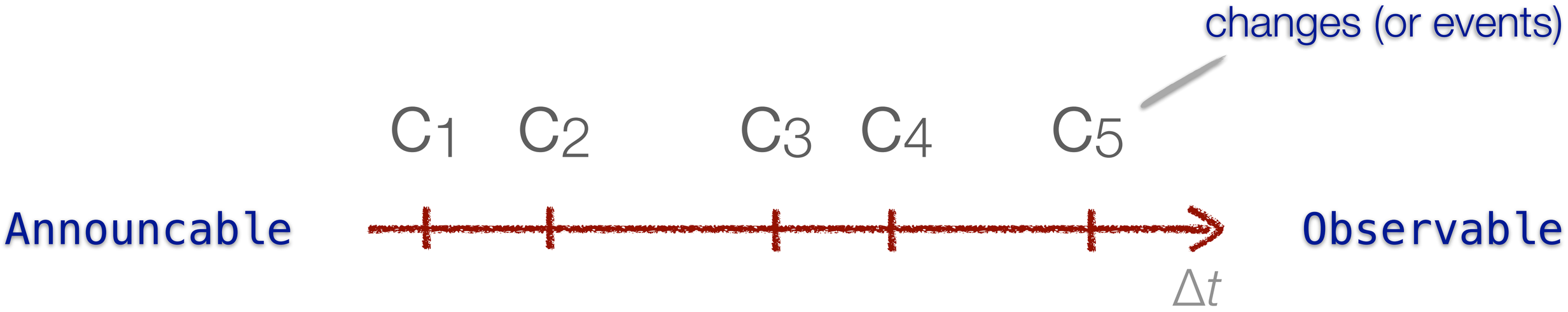
Ephemeral Stream of Changes



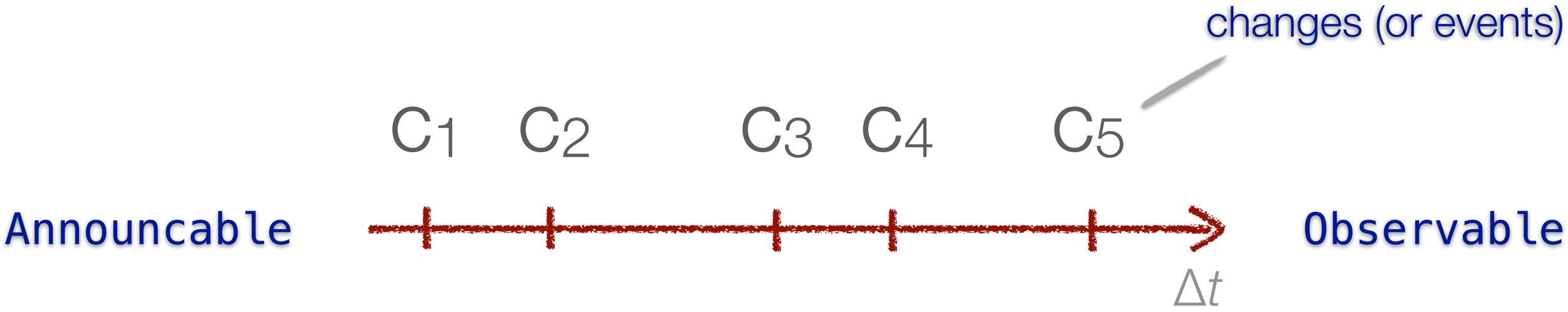
Ephemeral Stream of Changes



Ephemeral Stream of Changes



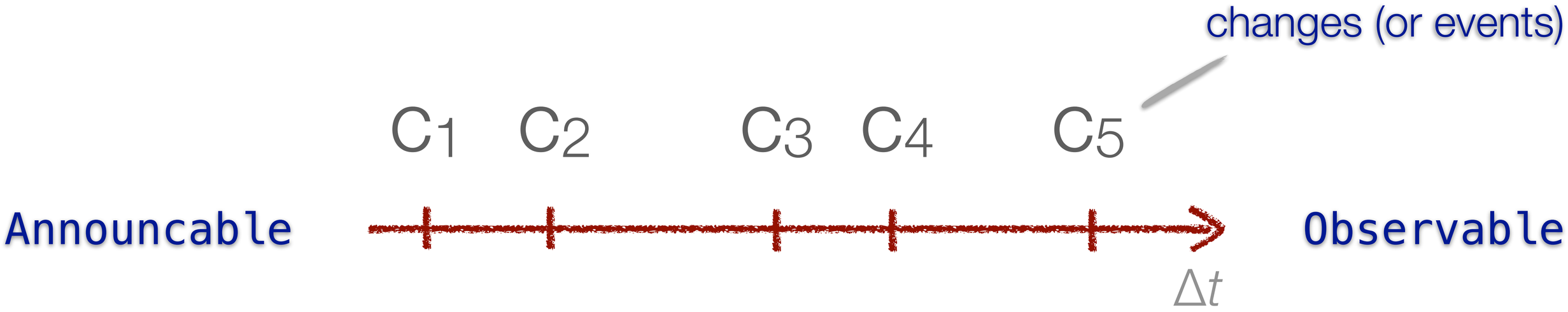
Ephemeral Stream of Changes



the type of change value

```
protocol Announcable {
  associatedtype AnnouncedValue
  func announce(change: AnnouncedValue)
}
```

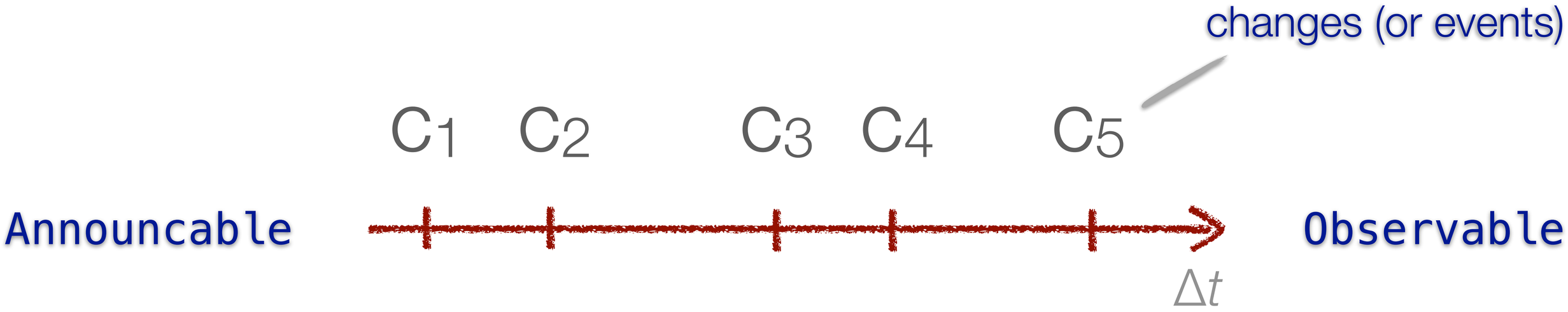
Ephemeral Stream of Changes




```
protocol Observable {
  associatedtype ObservedValue
  func observe<Context: AnyObject>
    (withContext context: Context,
                 observer: Observer<Context, ObservedValue>)
    -> Observation<ObservedValue>
}
typealias Observer<Context, ObservedValue> =
  (Context, ObservedValue) -> ()
```

again: the type of change value

Ephemeral Stream of Changes



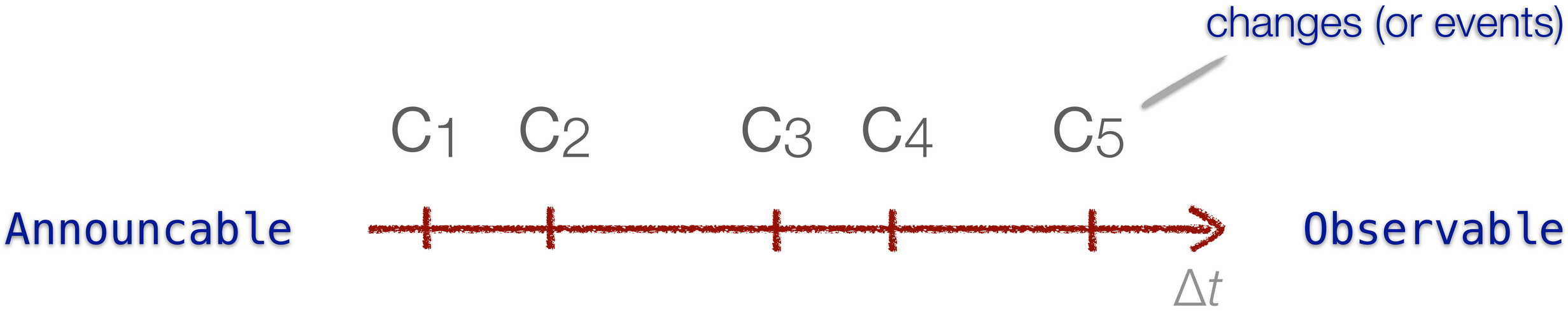

```
protocol Observable {
  associatedtype ObservedValue
  func observe<Context: AnyObject>
    (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
    -> Observation<ObservedValue>
}
typealias Observer<Context, ObservedValue> =
  (Context, ObservedValue) -> ()
```

again: the type of change value

determines lifetime

avoids strong reference

Ephemeral Stream of Changes




```
protocol Observable {
  associatedtype ObservedValue
  func observe<Context: AnyObject>
    (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
    -> Observation<ObservedValue>
}
typealias Observer<Context, ObservedValue> =
  (Context, ObservedValue) -> ()
```

again: the type of change value

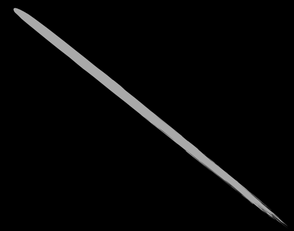
determines lifetime
avoids strong reference

observer callback

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
```

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
```

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
```



records observer callback and its context
manages their lifetimes
applies callbacks to change values

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
```

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
```

apply observer callback
to this change value

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
```

remove observers
whose context disappeared

apply observer callback
to this change value

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
 func observe<Context: AnyObject>
   (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
   -> Observation<Value>
 {
   let observation = Observation(observer: observer,
                                 context: context)


 }
}
```

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
 func observe<Context: AnyObject>
   (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
   -> Observation<Value>
 {
   let observation = Observation(observer: observer,
                                 context: context)


 }
}
```

wrap observer callback and its context

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
 func observe<Context: AnyObject>
   (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
   -> Observation<Value>
 {
   let observation = Observation(observer: observer,
                                 context: context)
   observers.append(observation)
   return observation
 }
}
```

```
class Changes<Value>: Announcable, Observable {
 typealias AnnouncedValue = Value
 typealias ObservedValue  = Value
 private var observers: [Observation<ObservedValue>] = []
 func announce(change: Value) {
  observers = observers.filter{ $0.apply(change) }
 }
 func observe<Context: AnyObject>
   (withContext context: Context,
                observer: Observer<Context, ObservedValue>)
   -> Observation<Value>
 {
   let observation = Observation(observer: observer,
                                 context: context)
   observers.append(observation)
   return observation
 }
}
```

add it to the current observer list

```
class Changes<Value>: Announcable, Observable {
…
var inlet: ChangesInlet<Value>
  { return ChangesInlet(changes: self) }
}
```

```
class Changes<Value>: Announcable, Observable {
…
var inlet: ChangesInlet<Value>
  { return ChangesInlet(changes: self) }
}
```

```
struct ChangesInlet<Value>: Announcable {
 typealias AnnouncedValue  = Value

 private let changes: Changes<Value>

 func announce(change: Value) {
   changes.announce(change: change)
 }
}
```

promote associated type
to a generic type parameter

```
class Changes<Value>: Announcable, Observable {
…
var inlet: ChangesInlet<Value>
  { return ChangesInlet(changes: self) }
}
```

```
struct ChangesInlet<Value>: Announcable {
 typealias AnnouncedValue  = Value

 private let changes: Changes<Value>

 func announce(change: Value) {
   changes.announce(change: change)
 }
}
```

promote associated type
to a generic type parameter

forward announcement

```
class Changes<Value>: Announcable, Observable {
…
var inlet: ChangesInlet<Value>
  { return ChangesInlet(changes: self) }
}
```

```
struct ChangesInlet<Value>: Announcable {
 typealias AnnouncedValue  = Value

 private let changes: Changes<Value>

 func announce(change: Value) {
   changes.announce(change: change)
 }
}
```

promote associated type
to a generic type parameter

forward announcement

```
class Changes<Value>: Announcable, Observable {
…
var inlet: ChangesInlet<Value>
  { return ChangesInlet(changes: self) }
}
```

```
struct ChangesInlet<Value>: Announcable {
 typealias AnnouncedValue  = Value

 private let changes: Changes<Value>

 func announce(change: Value) {
   changes.announce(change: change)
 }
}
```

promote associated type
to a generic type parameter

forward announcement

```
struct ChangesOutlet<Value>: Observable {
…
}
```

Changing Value

aka reactive value

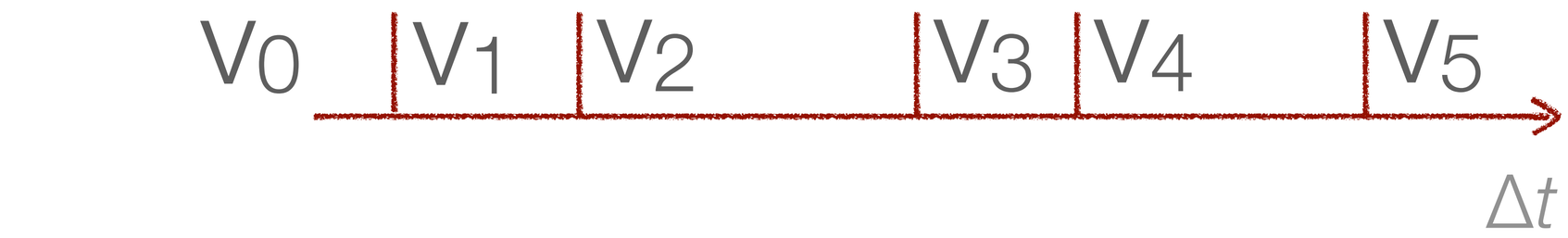
Changing Value

initial value

aka reactive value

Changing Value

initial value

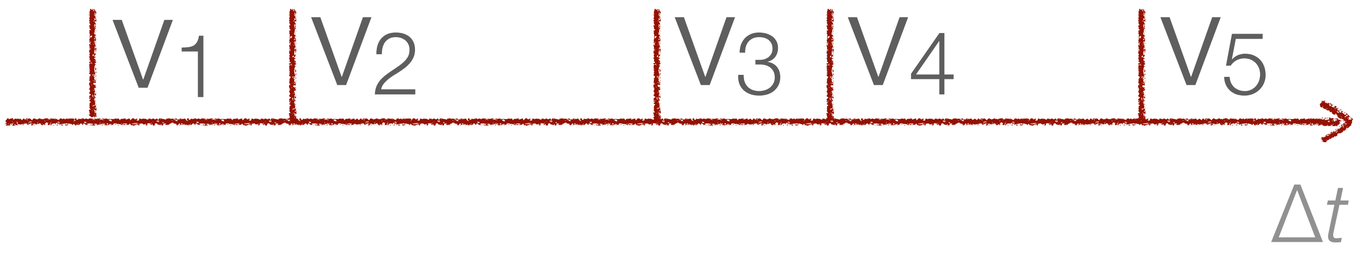


aka reactive value

Changing Value

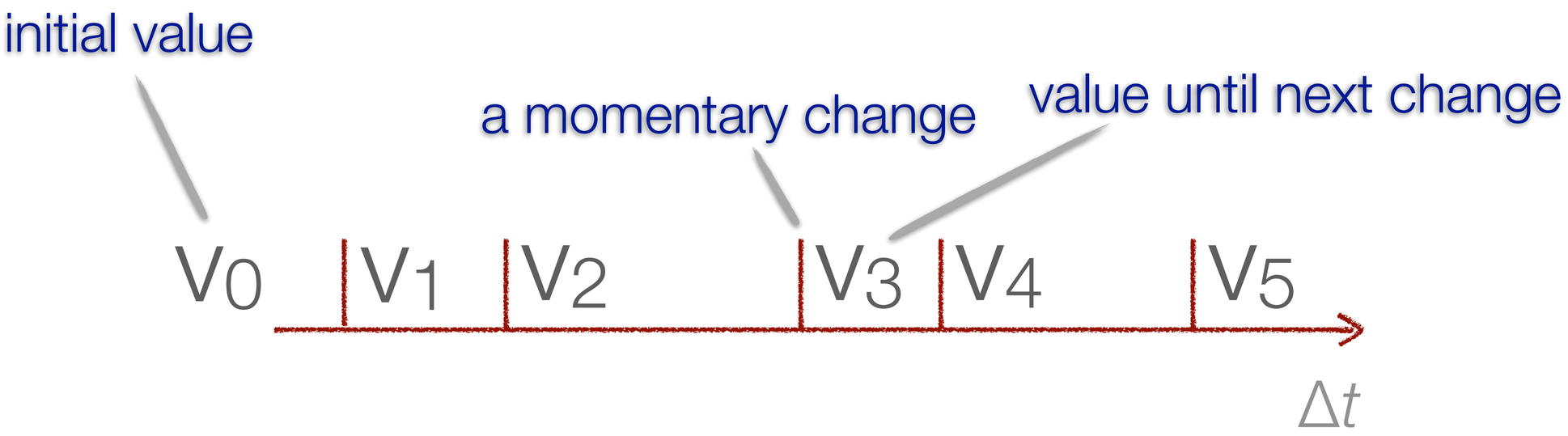
initial value

a momentary change



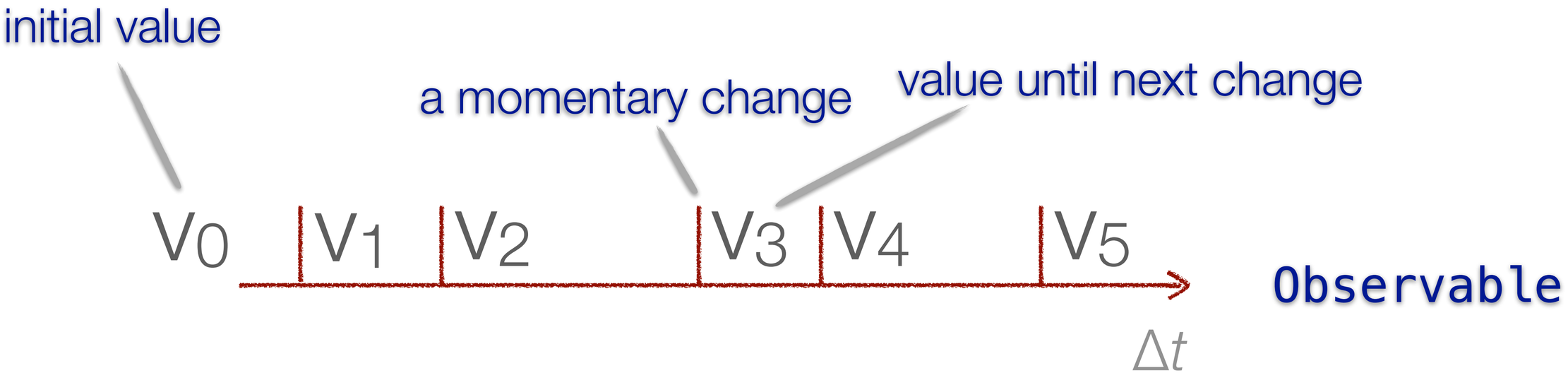
aka reactive value

Changing Value



aka reactive value

Changing Value



2

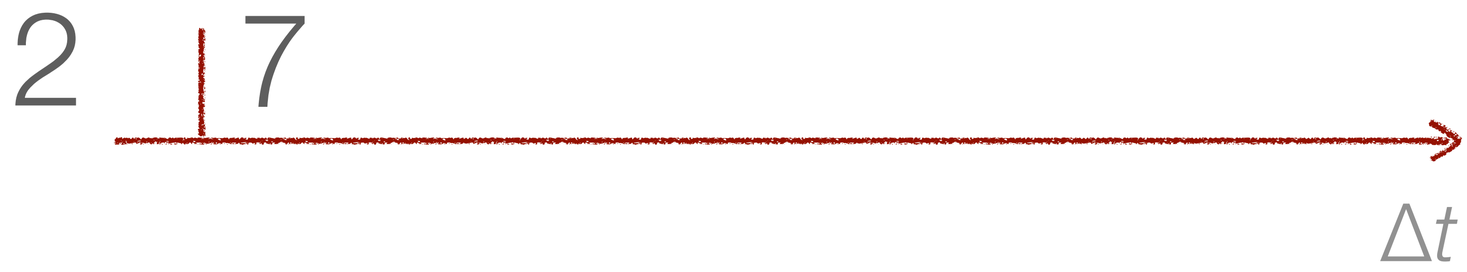

```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

accumulation function

2

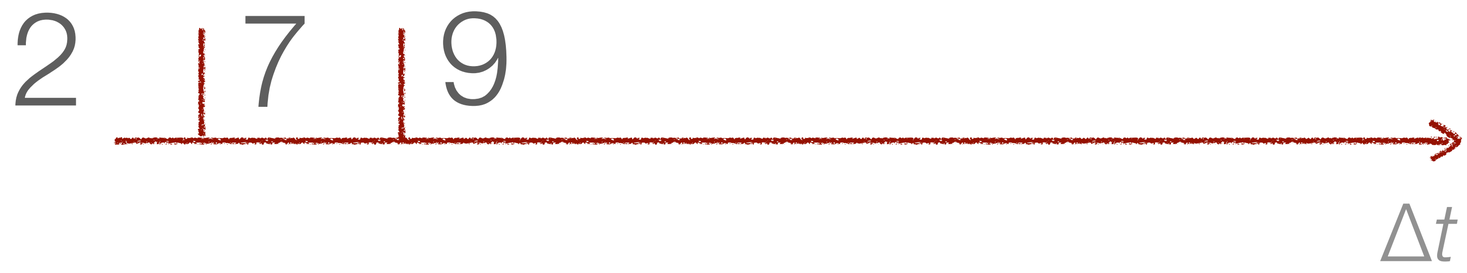

```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

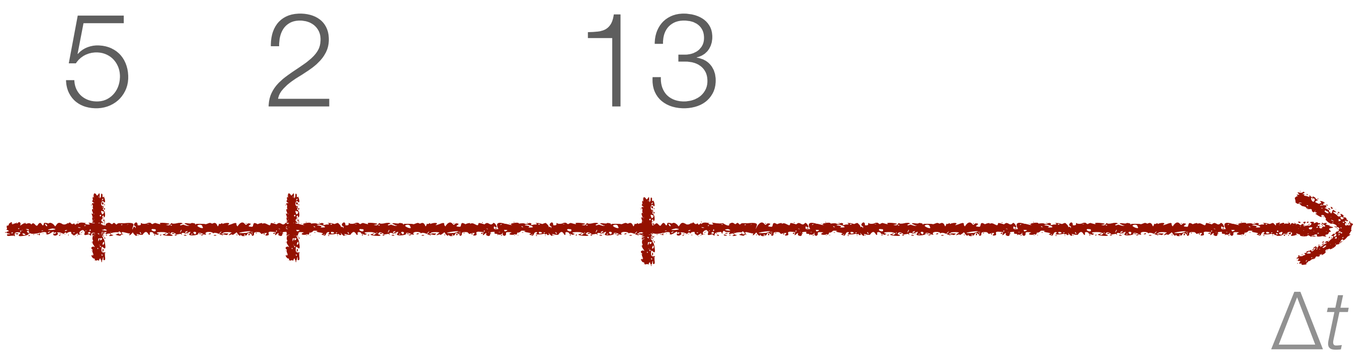
accumulation function




```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

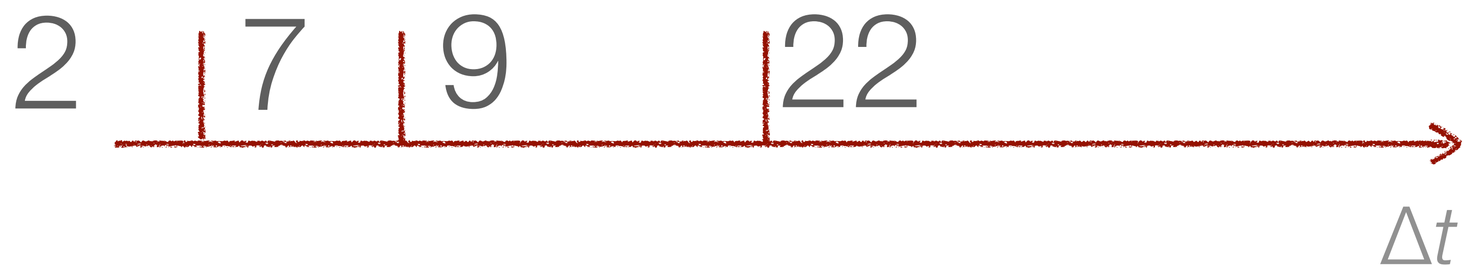
accumulation function

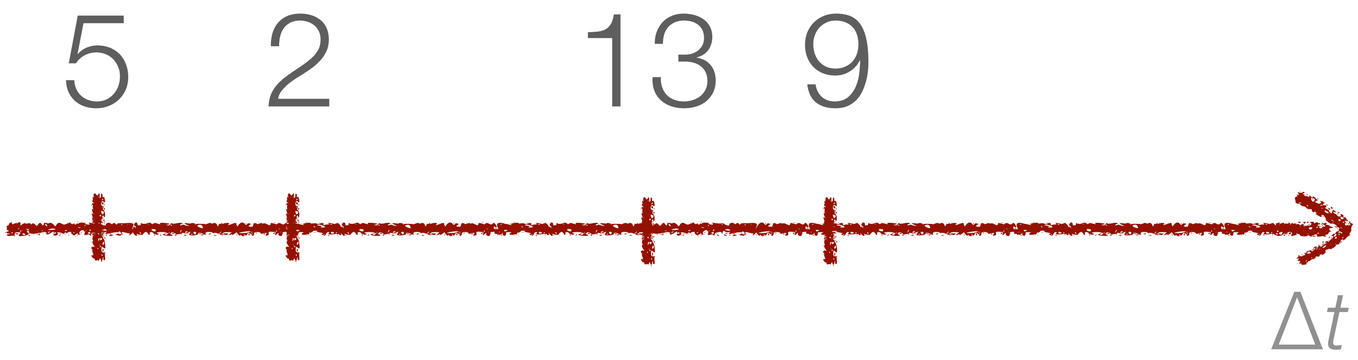





```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

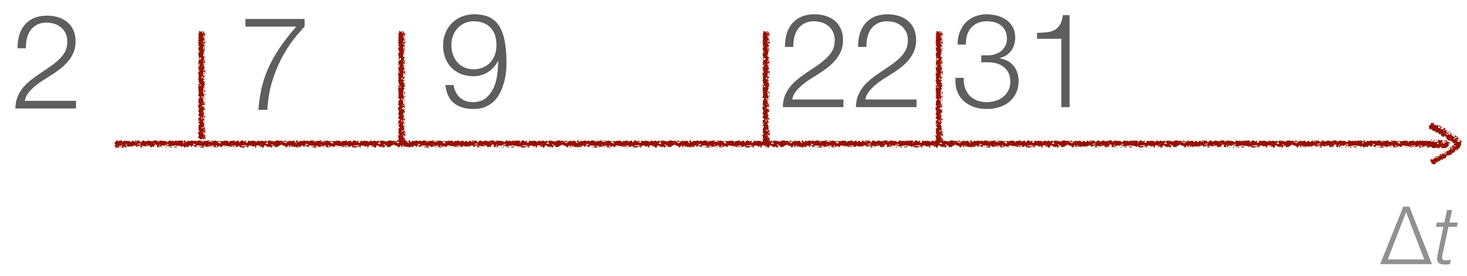
accumulation function

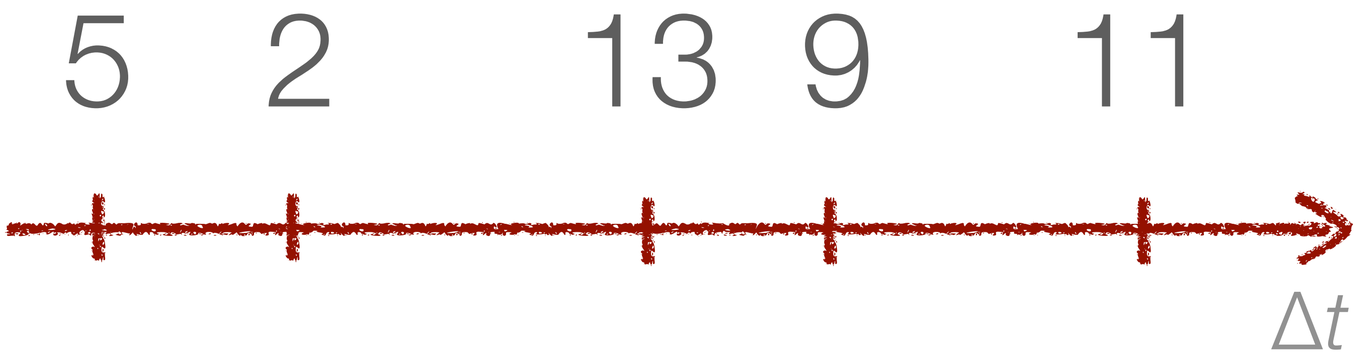





```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

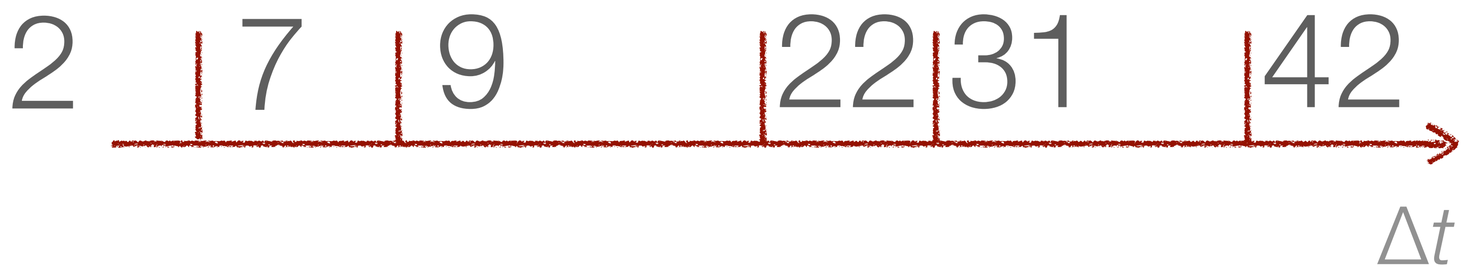
accumulation function



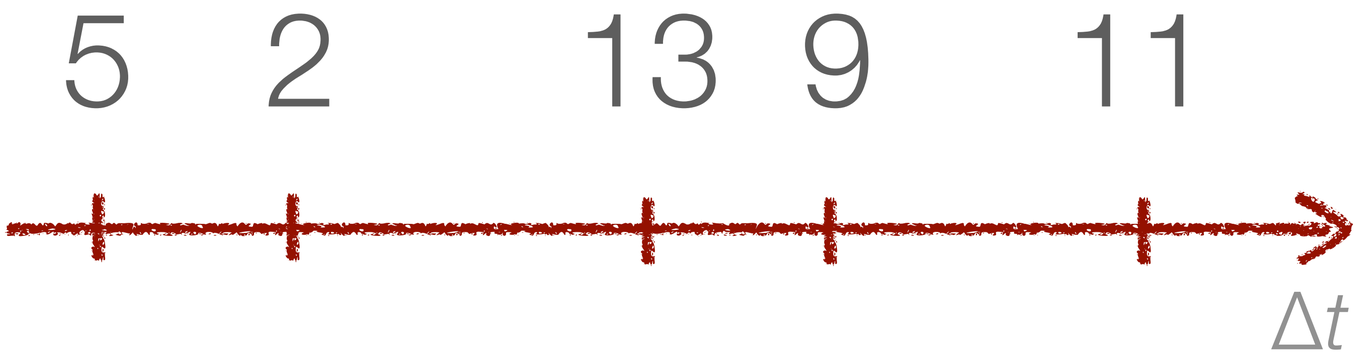



```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

accumulation function

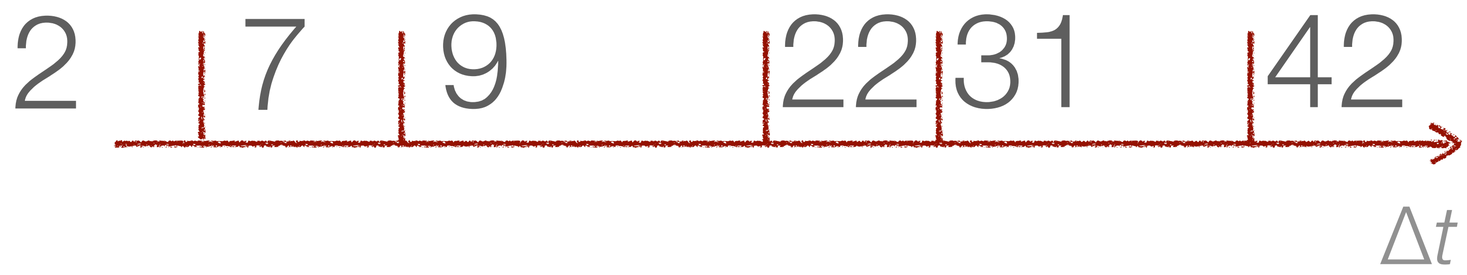


momentary

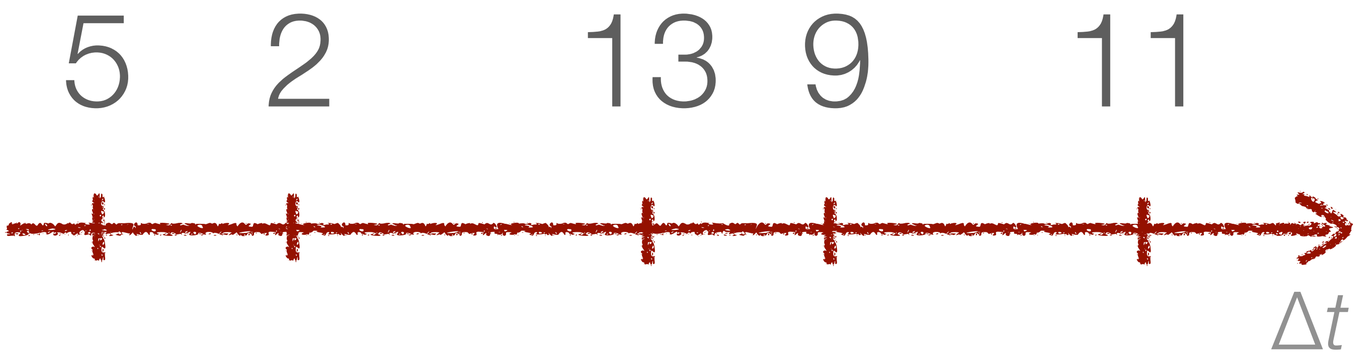



```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

accumulation function



momentary




```
.accumulate(startingFrom: 2){ $0 + $1 } =
```

accumulation function

continuing accumulator

Δt

```
.accumulate(startingFrom: 2){ $0 } =
```

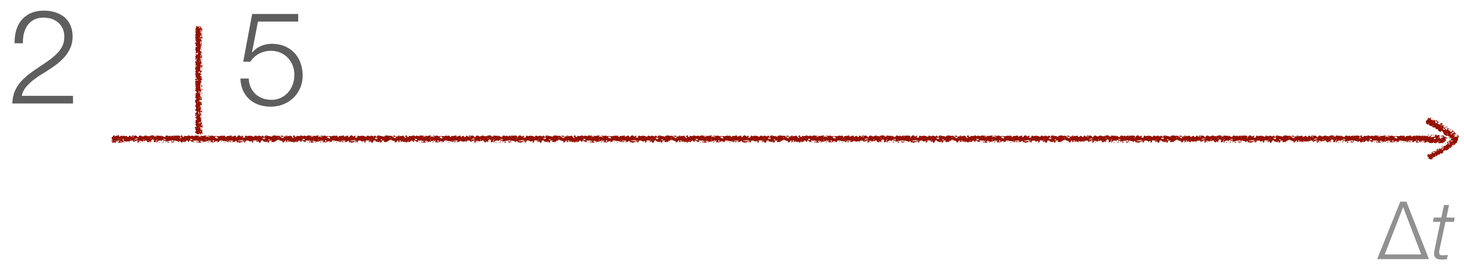
only the change value

2

Δt

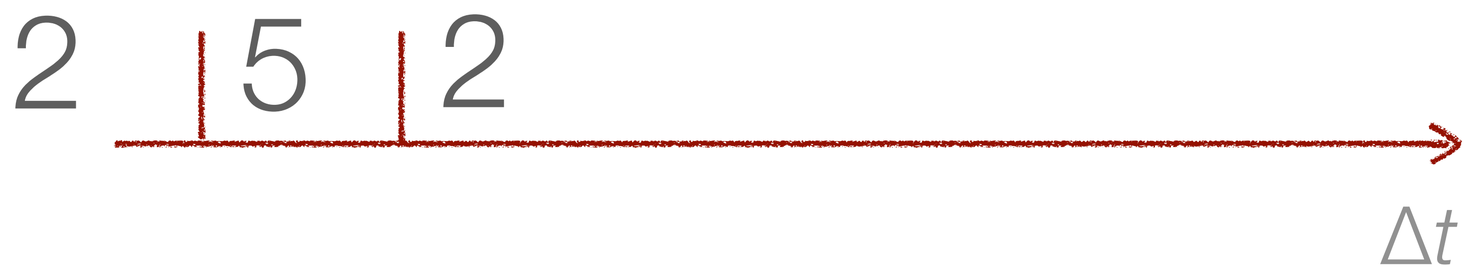

```
.accumulate(startingFrom: 2){ $0 } =
```

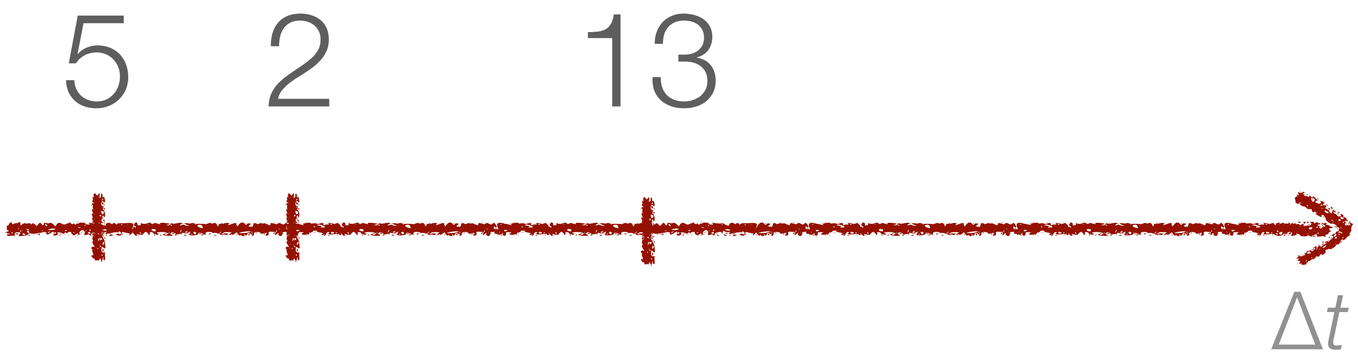
only the change value




```
.accumulate(startingFrom: 2){ $0 } =
```

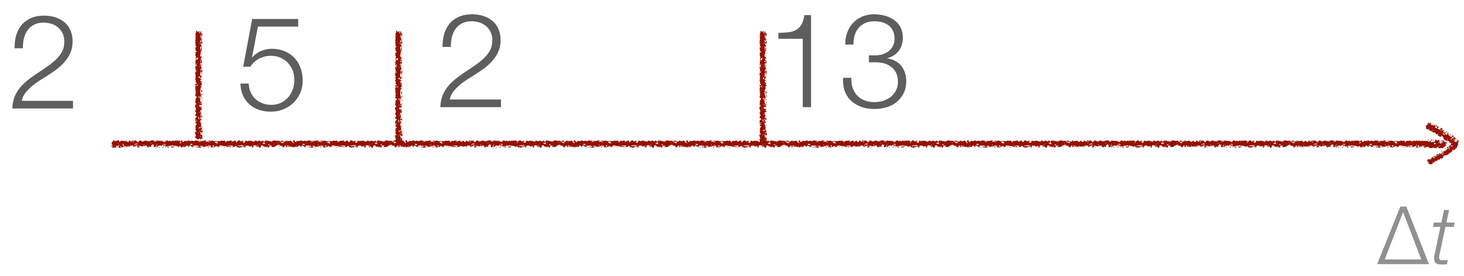
only the change value

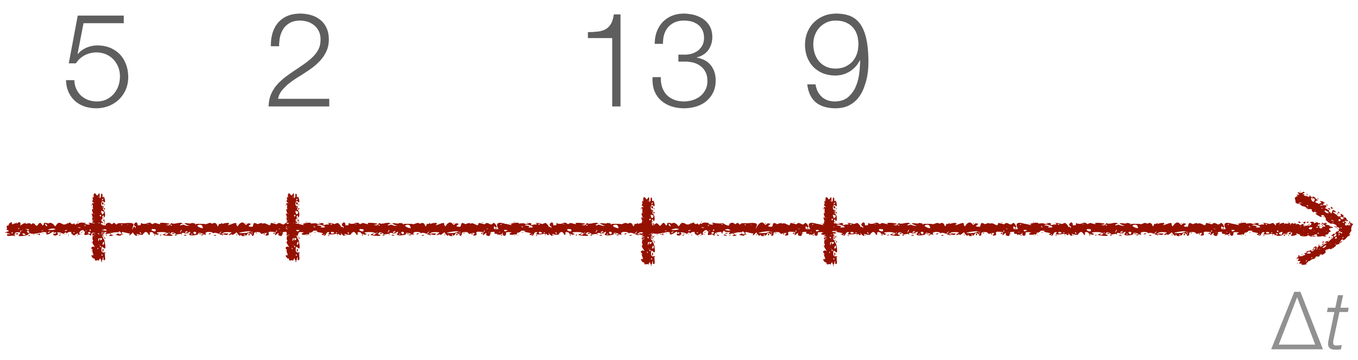





```
.accumulate(startingFrom: 2){ $0 } =
```

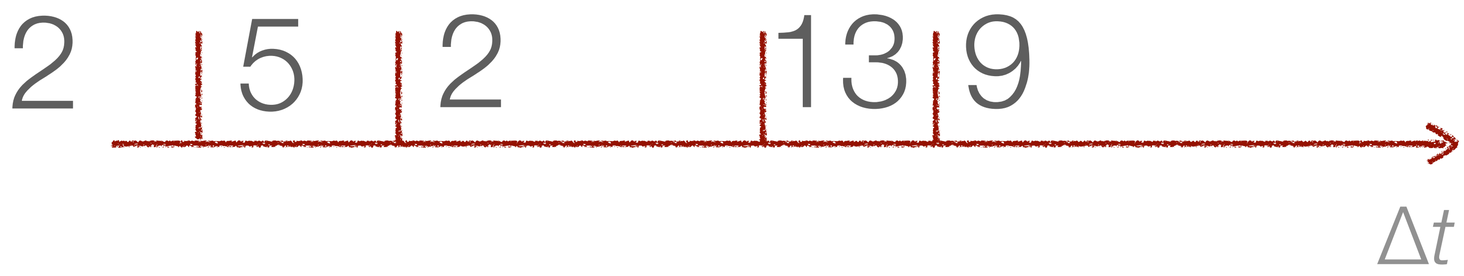
only the change value

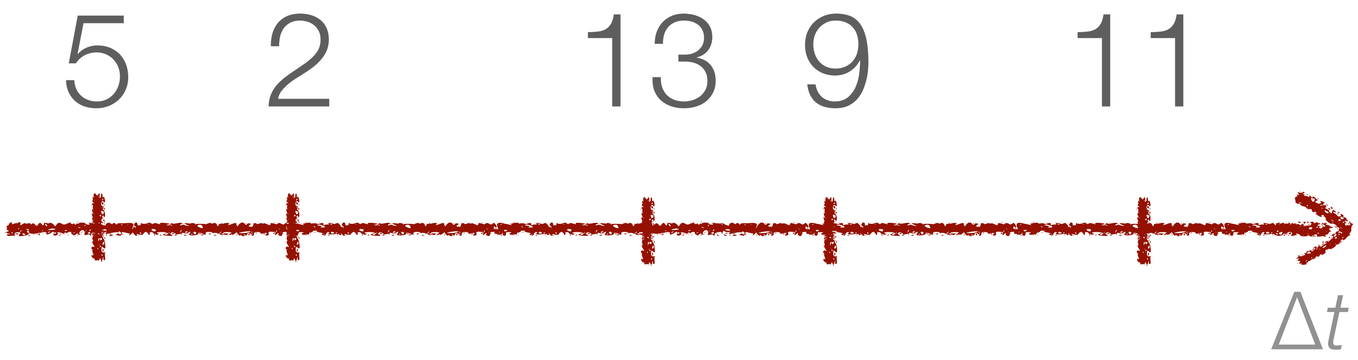





```
.accumulate(startingFrom: 2){ $0 } =
```

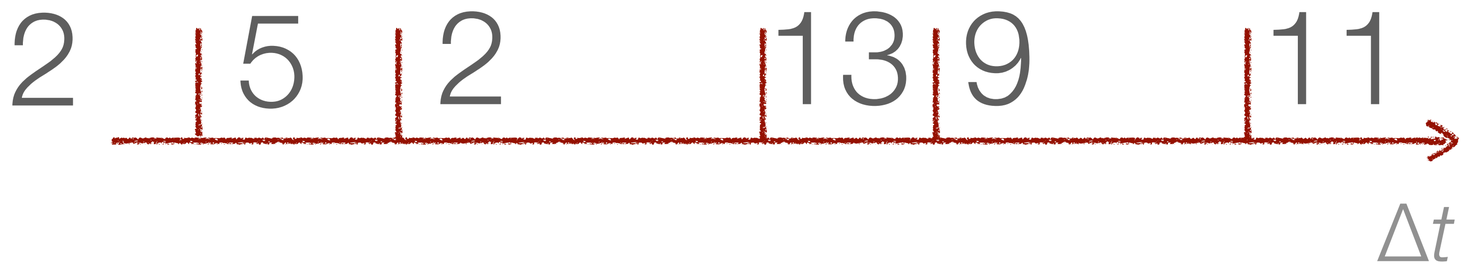
only the change value






```
.accumulate(startingFrom: 2){ $0 } =
```

only the change value



Δt

```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

m_0

Δt

Δt

```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model m_0

Δt

UI changes

Δt

```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model m_0

Δt

UI changes


```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model

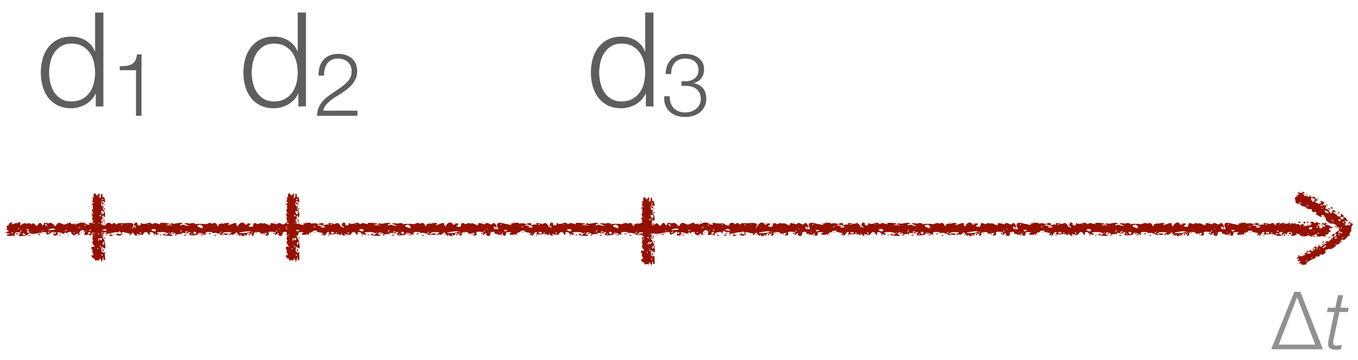
UI changes


```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model

UI changes




```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model

UI changes


```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model

UI changes


```
.accumulate(startingFrom: m₀){ diff, model in
  return diff.transform(model) } =
```

compute updated model

MVC model

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
```

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
```

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
```

current value

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
```

current value

stream of accumulator values over time
(whenever the accumulator changes, it is announced here)

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    …




  }
}
```

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    …
  }
}
```

consumed/accumulated

stream of changes/observations

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    …
  }
}
```

consumed/accumulated

stream of changes/observations

initial accumulator value

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    …
  }
}
```

consumed/accumulated stream of changes/observations

initial accumulator value

accumulation function

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    accumulator       = initial
    changes           = Changes<Value>()
    …


    }
  }
}
```

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    accumulator       = initial
    changes           = Changes<Value>()
    observed.observe(withContext: self){ (context, value in
      context.accumulator = accumulate(value, context.accumulator)
      context.changes.announce(change: context.accumulator)
    }
  }
}
```

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    accumulator      = initial
    changes          = Changes<Value>()
    observed.observe(withContext: self){ (context, value in
      context.accumulator = accumulate(value, context.accumulator)
      context.changes.announce(change: context.accumulator)
    }
  }
}
```

update accumulator

```
class Changing<Value>: Observable {
  typealias ObservedValue = Value
  private var accumulator: Value
  private var changes:     Changes<Value>
  init<Observed: Observable>
    (observing observed: Observed,
     startingFrom initial: Value,
     accumulateWith accumulate:
       (Observed.ObservedValue, Value) -> Value)
  {
    accumulator       = initial
    changes           = Changes<Value>()
    observed.observe(withContext: self){ (context, value in
      context.accumulator = accumulate(value, context.accumulator)
      context.changes.announce(change: context.accumulator)
    }
  }
}
```

update accumulator

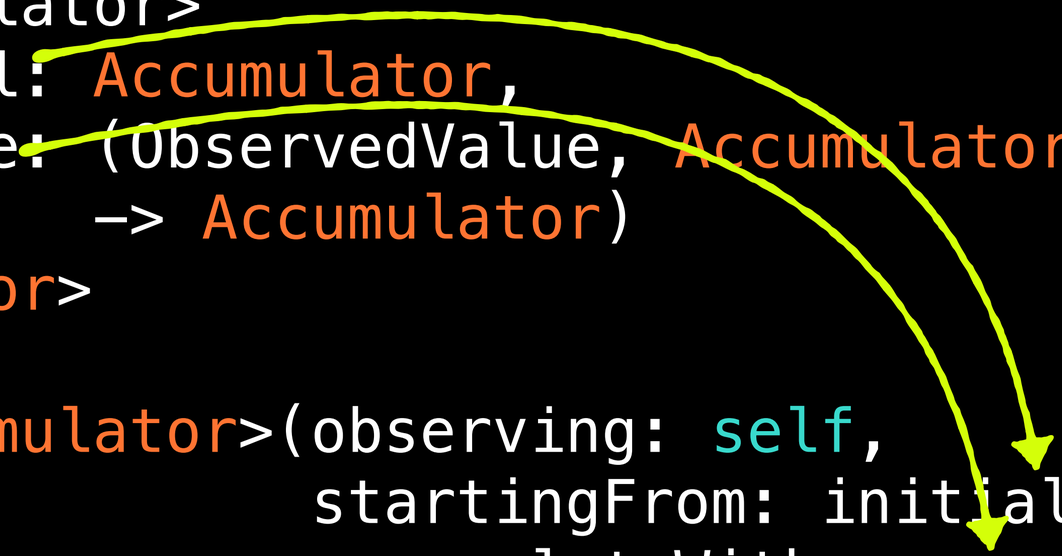
announce updated accumulator value

Accumulating Observations

```
extension Observable {

  func accumulate<Accumulator>
    (startingFrom initial: Accumulator,
accumulateWith accumulate: (ObservedValue, Accumulator)
                           -> Accumulator)
   -> Changing<Accumulator>
  {
    return Changing<Accumulator>(observing: self,
                                 startingFrom: initial,
                                 accumulateWith: accumulate)
  }
}
```

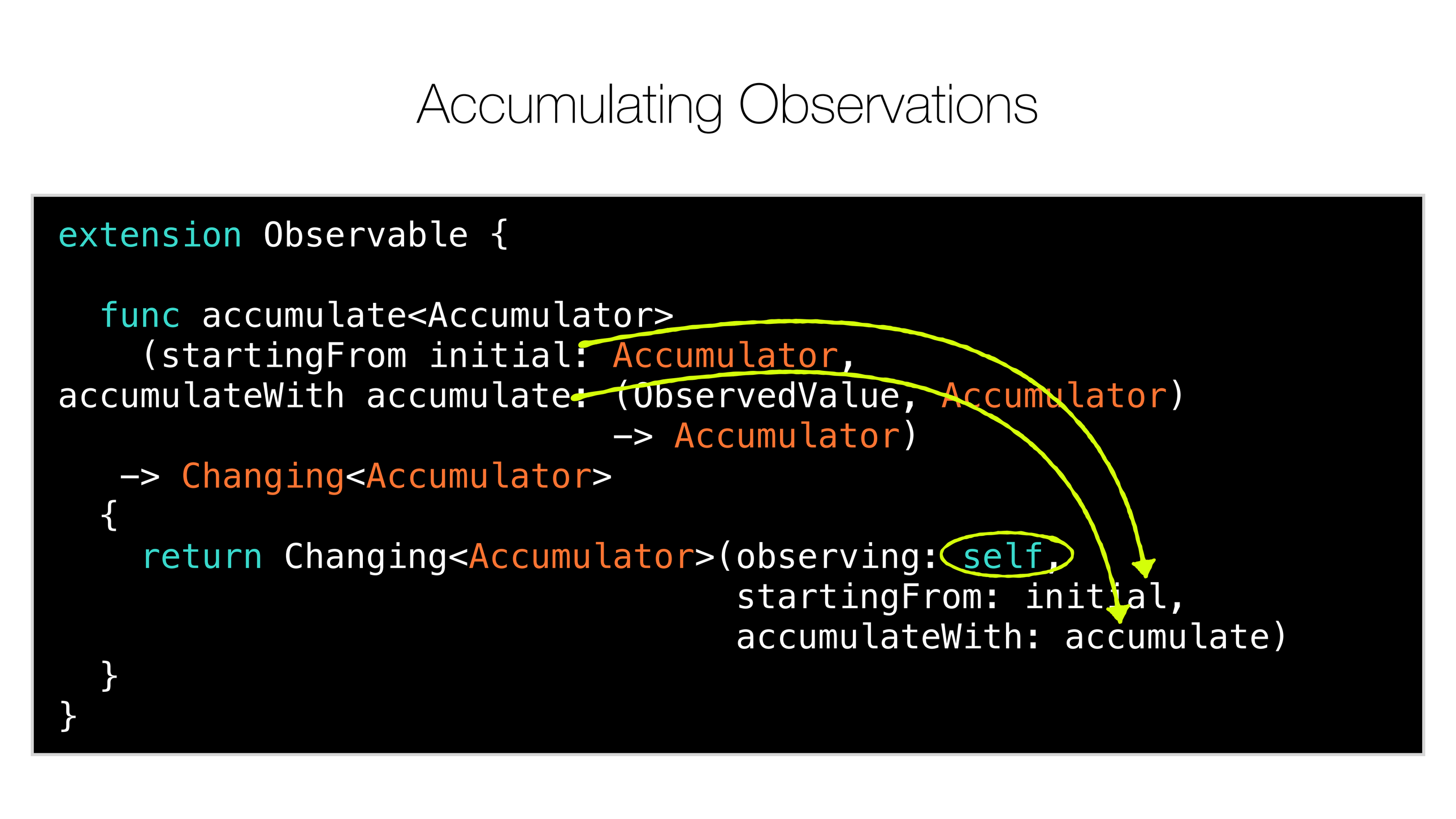
Accumulating Observations

```
extension Observable {

  func accumulate<Accumulator>
    (startingFrom initial: Accumulator,
accumulateWith accumulate: (ObservedValue, Accumulator)
                             -> Accumulator)
   -> Changing<Accumulator>
  {
    return Changing<Accumulator>(observing: self,
                                 startingFrom: initial,
                                 accumulateWith: accumulate)
  }
}
```



Accumulating Observations

```
extension Observable {

  func accumulate<Accumulator>
    (startingFrom initial: Accumulator,
accumulateWith accumulate: (ObservedValue, Accumulator)
                           -> Accumulator)
   -> Changing<Accumulator>
  {
    return Changing<Accumulator>(observing: self,
                                 startingFrom: initial,
                                 accumulateWith: accumulate)
  }
}
```



Mapping Observations

Mapping Observations

Δt

```
.map{ $0 * 2 } =
```

Δt

Mapping Observations

5

Δt

```
.map{ $0 * 2 } =
```

10

Δt

Mapping Observations

5 2

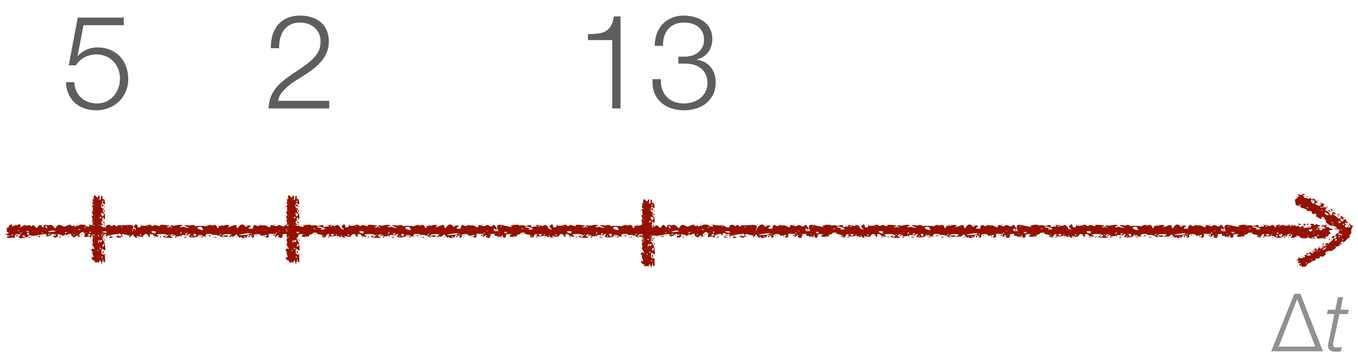
Δt

```
.map{ $0 * 2 } =
```

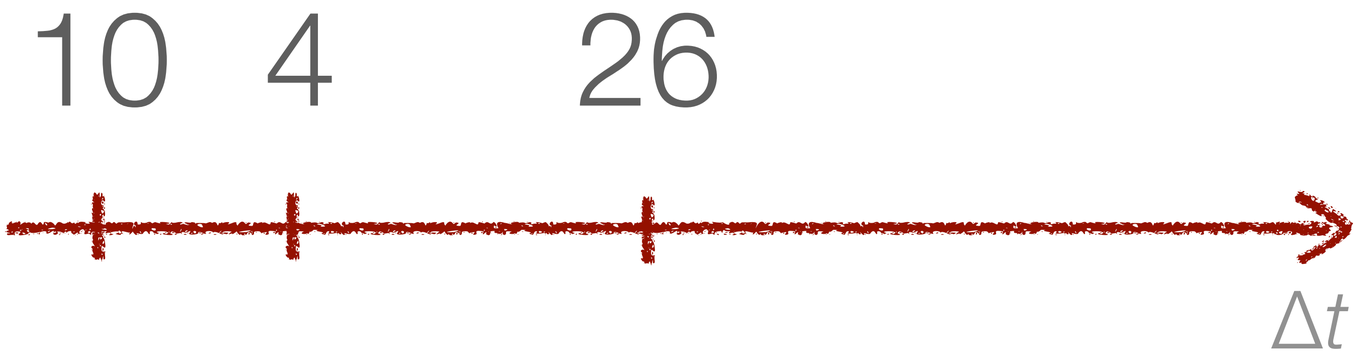
10 4

Δt

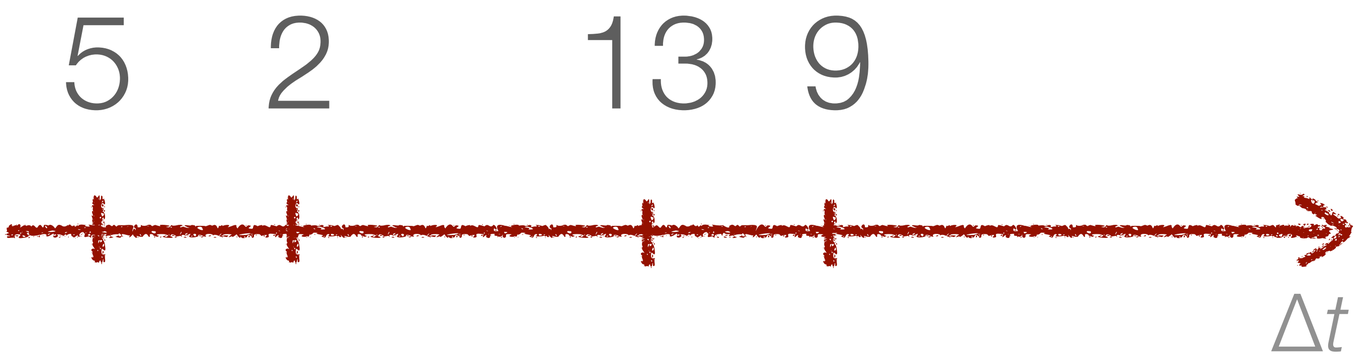
Mapping Observations



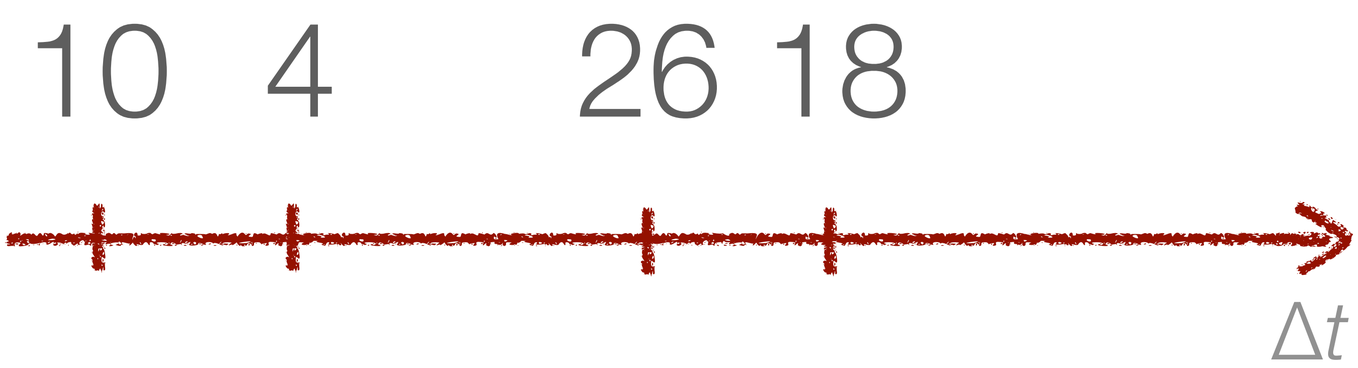

```
.map{ $0 * 2 } =
```



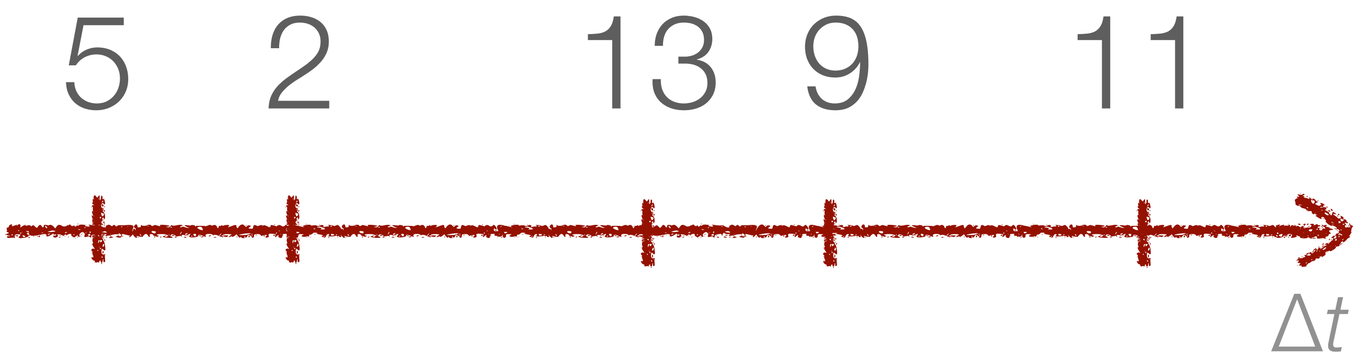
Mapping Observations



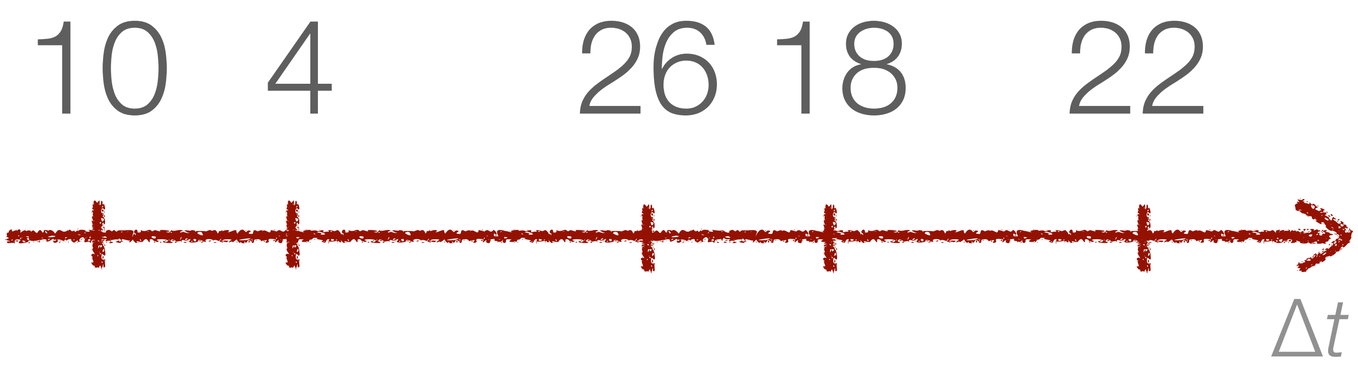

```
.map{ $0 * 2 } =
```



Mapping Observations




```
.map{ $0 * 2 } =
```



```
extension Observable {

 func map<MappedValue>(transform: (ObservedValue) -> MappedValue)
   -> Changes<MappedValue> {

   let changes = Changes<MappedValue>()
   observe(withContext: changes,
           observer: { changesContext, change in
            changesContext.announce(change: transform(change)) })
   return changes
 }
}
```

```
extension Observable {

 func map<MappedValue>(transform: (ObservedValue) -> MappedValue)
   -> Changes<MappedValue> {

   let changes = Changes<MappedValue>()
   observe(withContext: changes,
           observer: { changesContext, change in
            changesContext.announce(change: transform(change)) })
   return changes
 }
}
```

observe self

```
extension Observable {

 func map<MappedValue>(transform: (ObservedValue) -> MappedValue)
   -> Changes<MappedValue> {

   let changes = Changes<MappedValue>()
   observe(withContext: changes,
           observer: { changesContext, change in
            changesContext.announce(change: transform(change)) })
   return changes
 }
}
```

observe self

announce transformed value
on resulting stream of changes

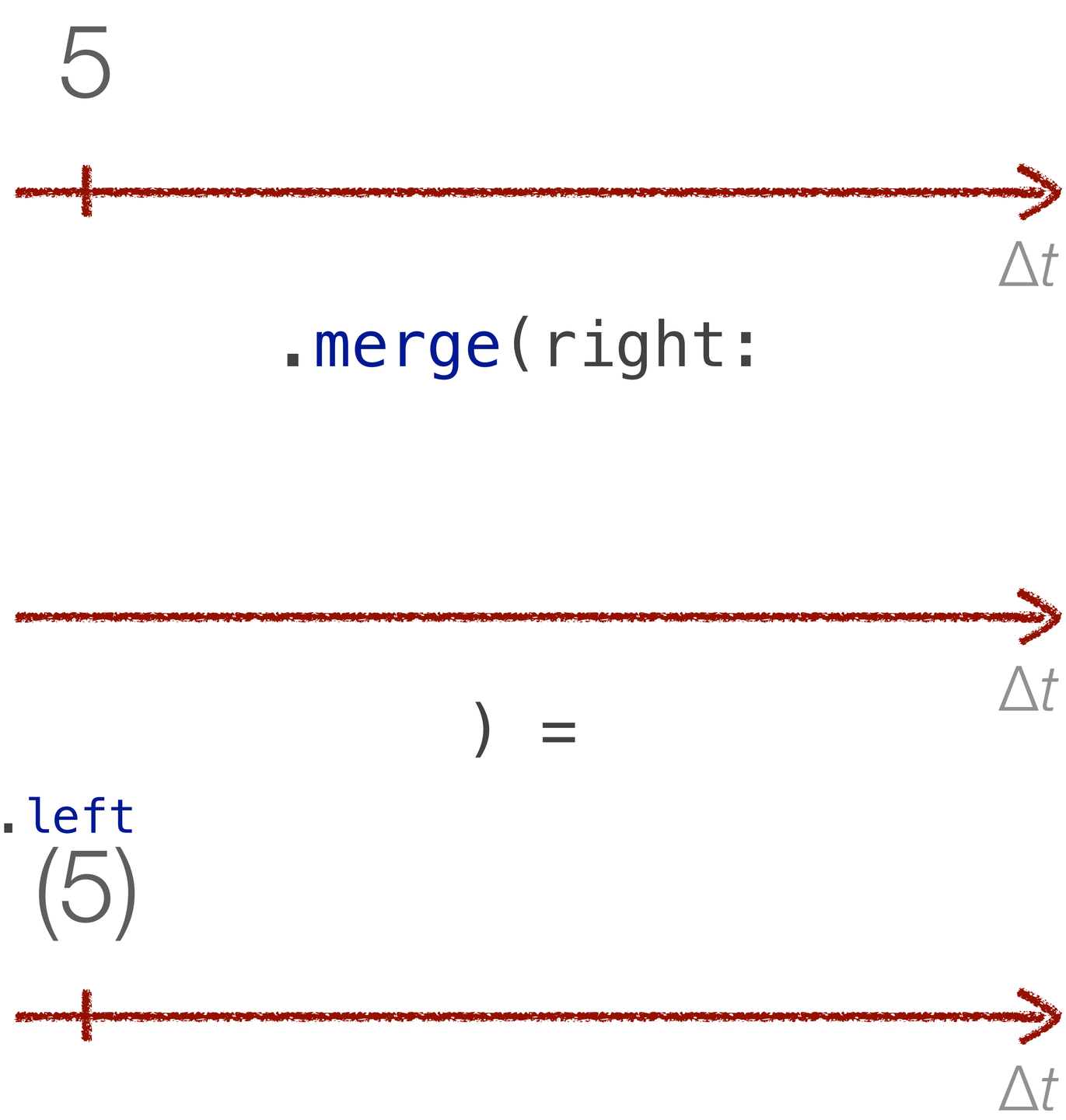

Δt

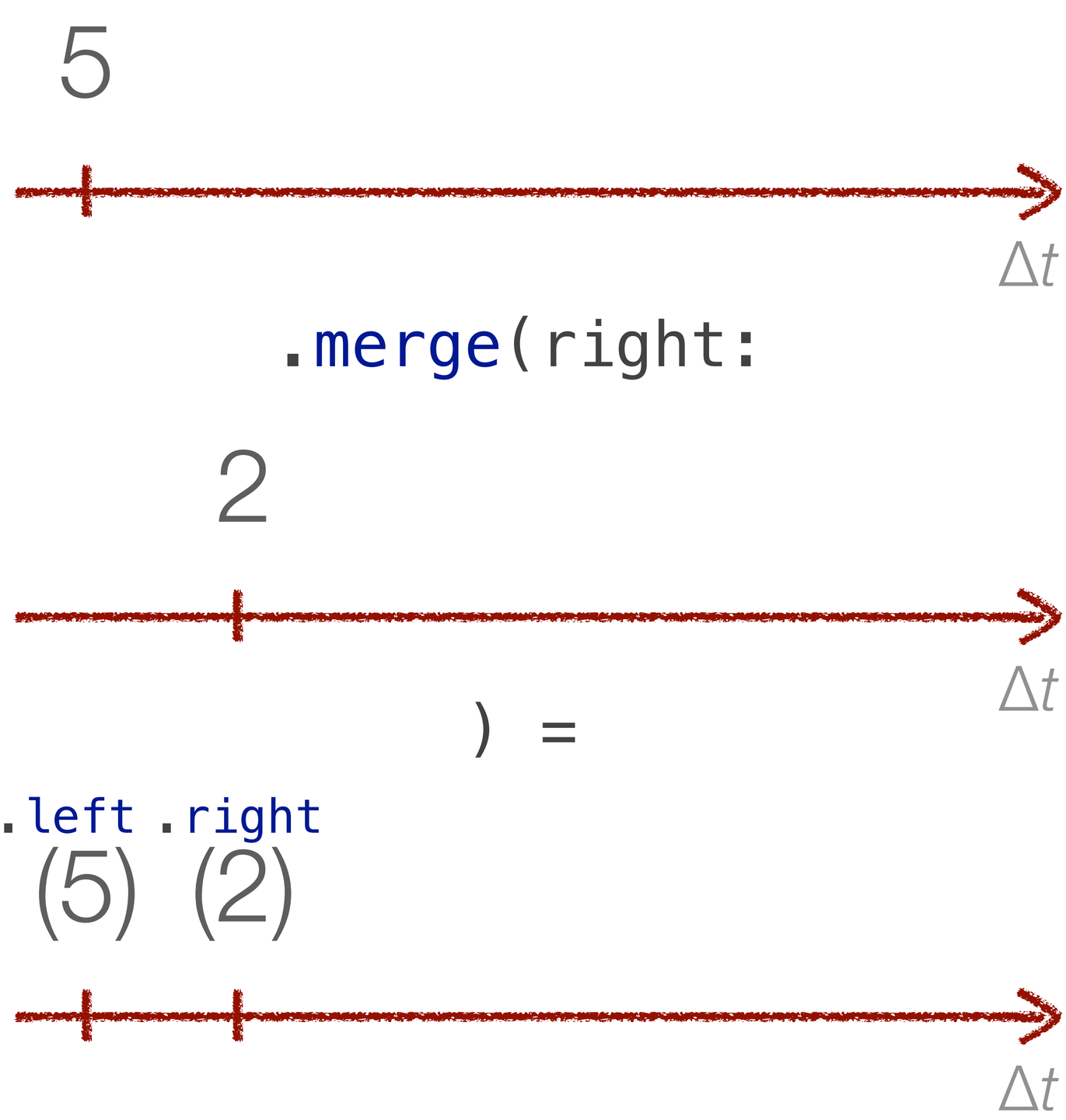

Δt

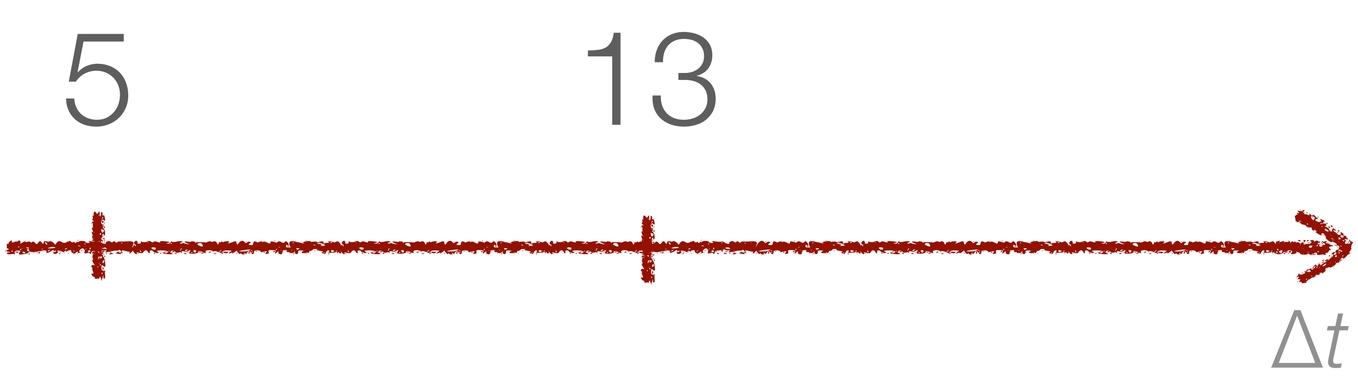

Δt


```
.merge(right:
```


```
) =
```


5
Δt
.merge(right:
Δt
) =
.left
(5)
Δt


5
Δt
.merge(right:
2
Δt
) =
.left .right
(5) (2)
Δt

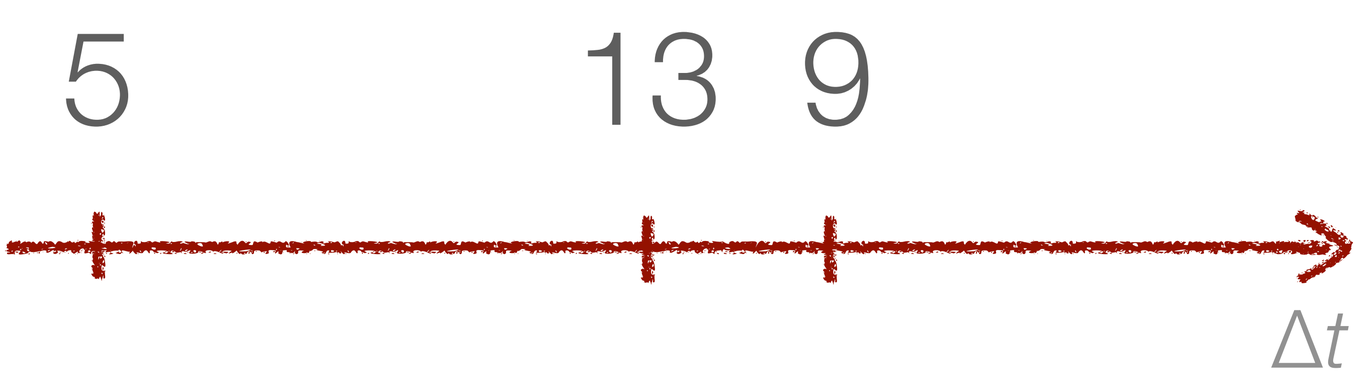



```
.merge(right:
```


```
) =
```

```
.left .right    .left
```

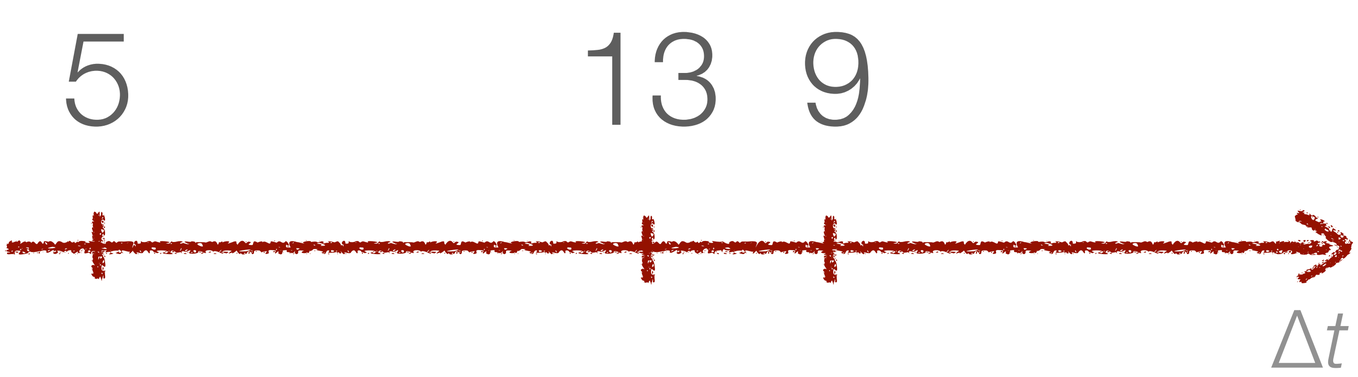
(5) (2) (13)




```
.merge(right:
```


```
) =
.left .right    .left .left
```

(5) (2) (13) (9)




```
.merge(right:
```

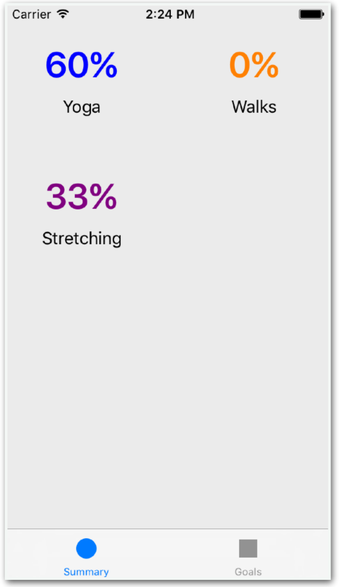

```
) =
```

```
.left .right    .left .left    .right
```

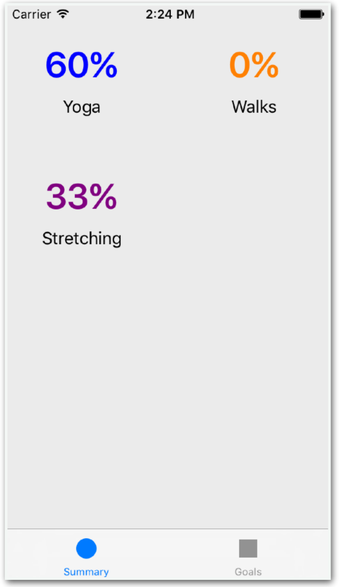
(5) (2) (13) (9) (11)

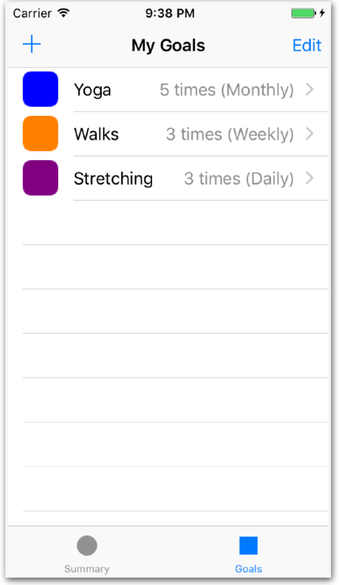
Part 3
Putting FRP to use

Goals.app

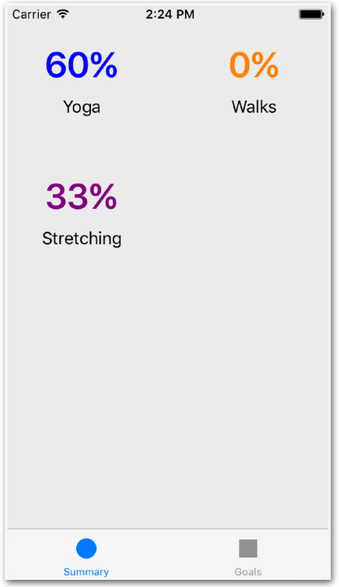


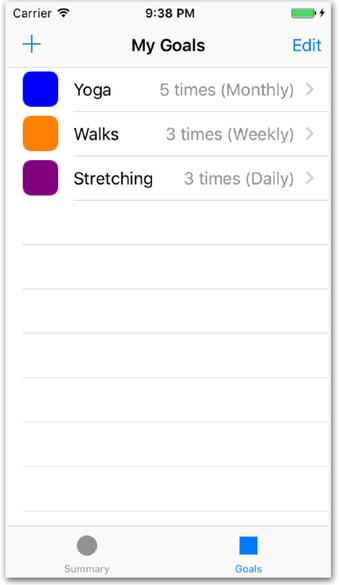
Goals.app

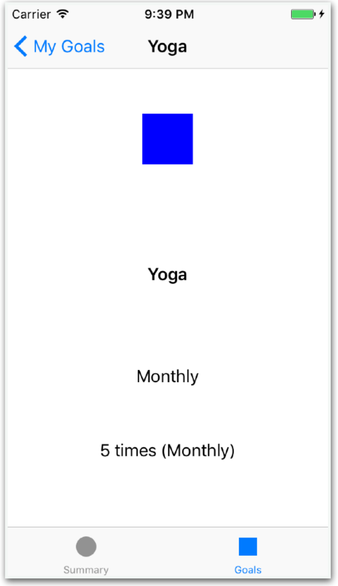

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals


Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

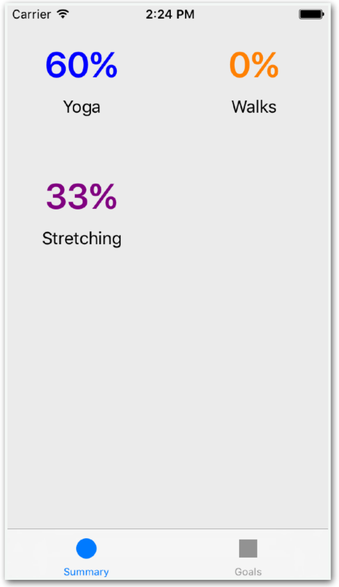
Goals.app

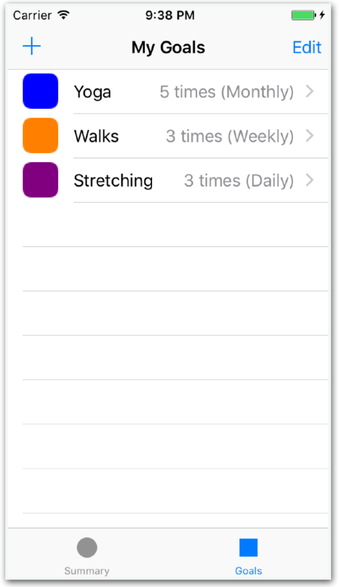

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals

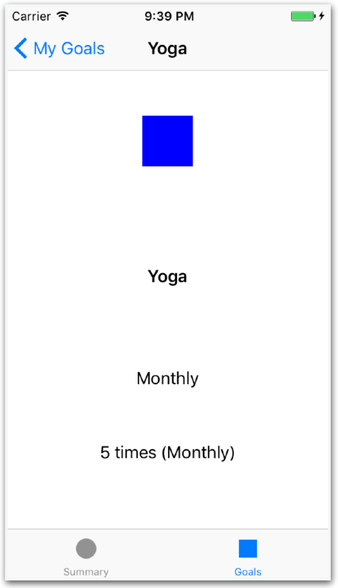

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

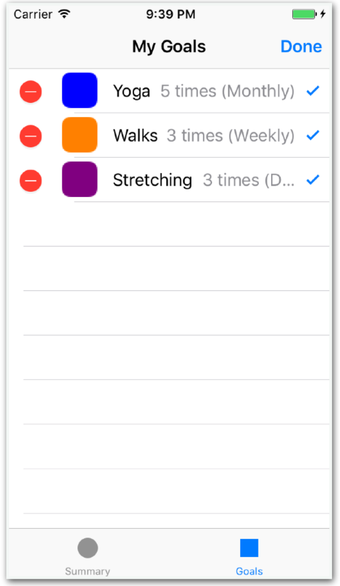

Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals

Goals.app

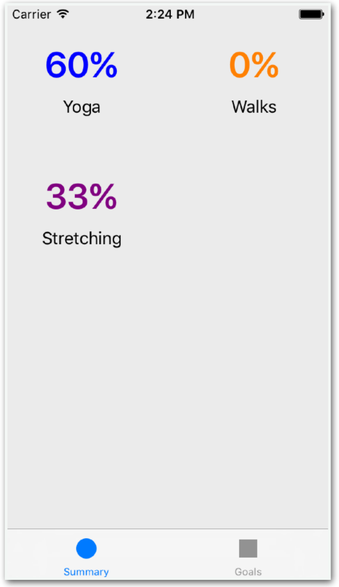

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals

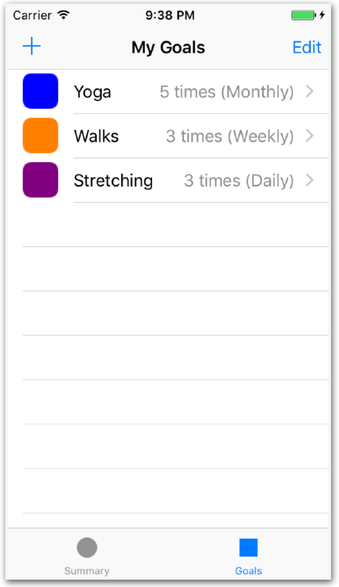

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

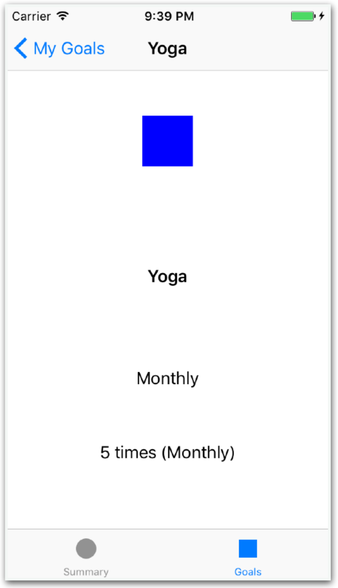

Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals

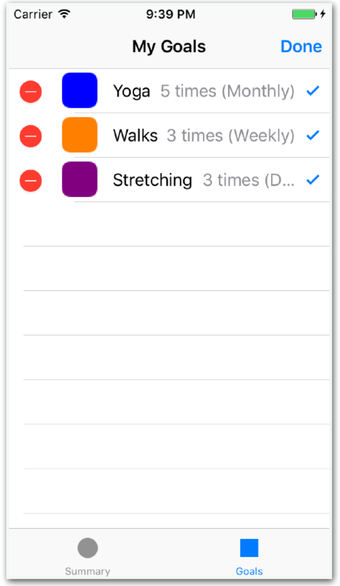

Carrier
9:39 PM
My Goals
Done
Yoga 5 times (Monthly)
Walks 3 times (Weekly)
Stretching 3 times (D...
Summary
Goals

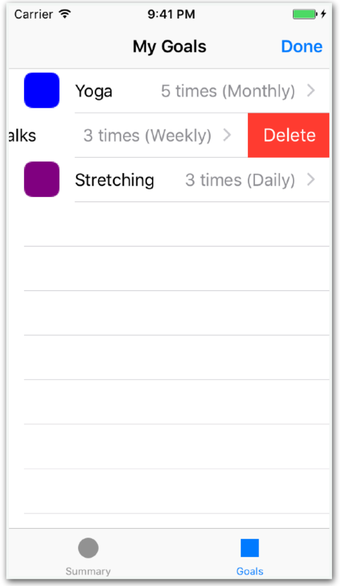
Goals.app


Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals

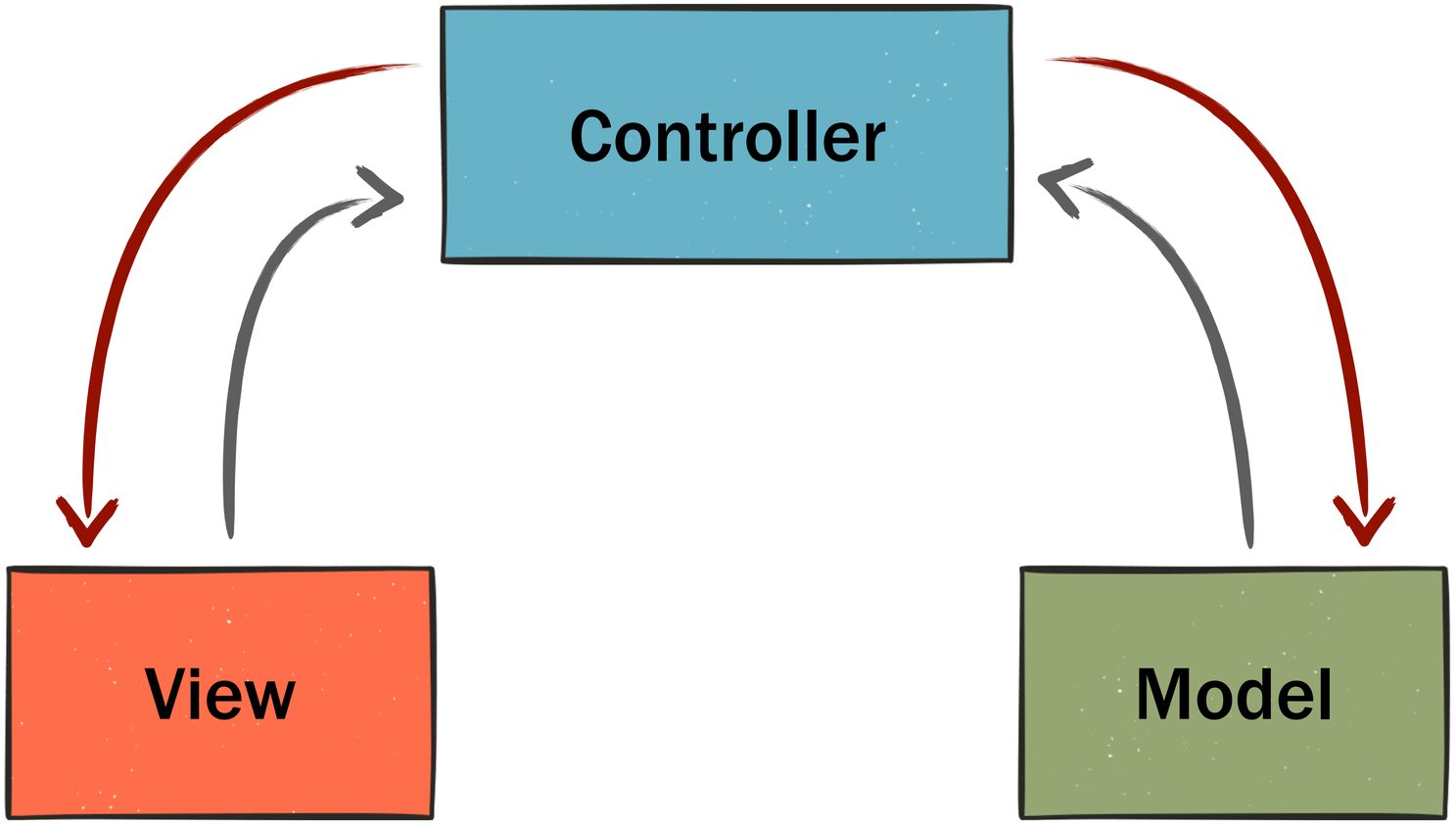

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals


Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals

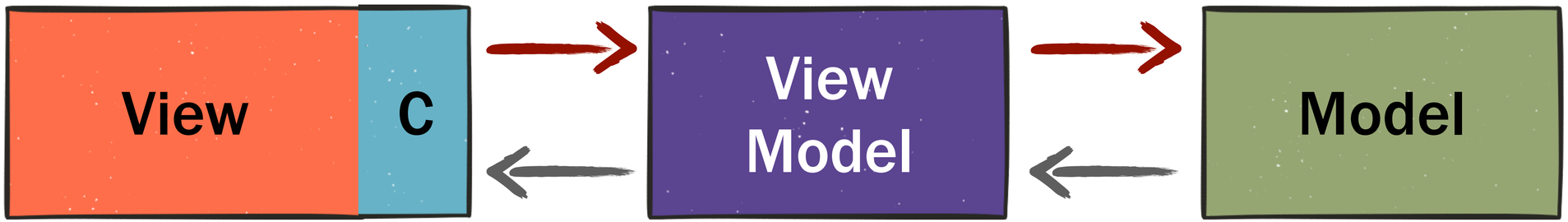

Carrier
9:39 PM
My Goals
Done
Yoga 5 times (Monthly)
Walks 3 times (Weekly)
Stretching 3 times (D...
Summary
Goals


Carrier
9:41 PM
My Goals
Done
Yoga
5 times (Monthly)
alks
3 times (Weekly)
Delete
Stretching
3 times (Daily)
Summary
Goals

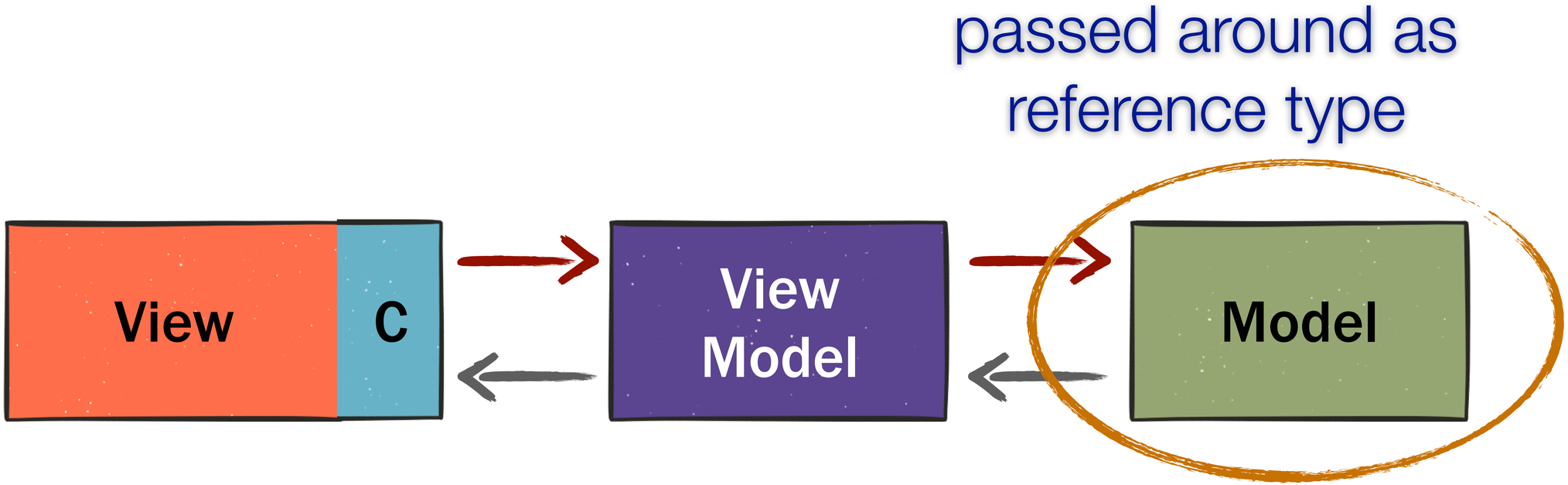
MVC & MVVM Architectures



MVC & MVVM Architectures

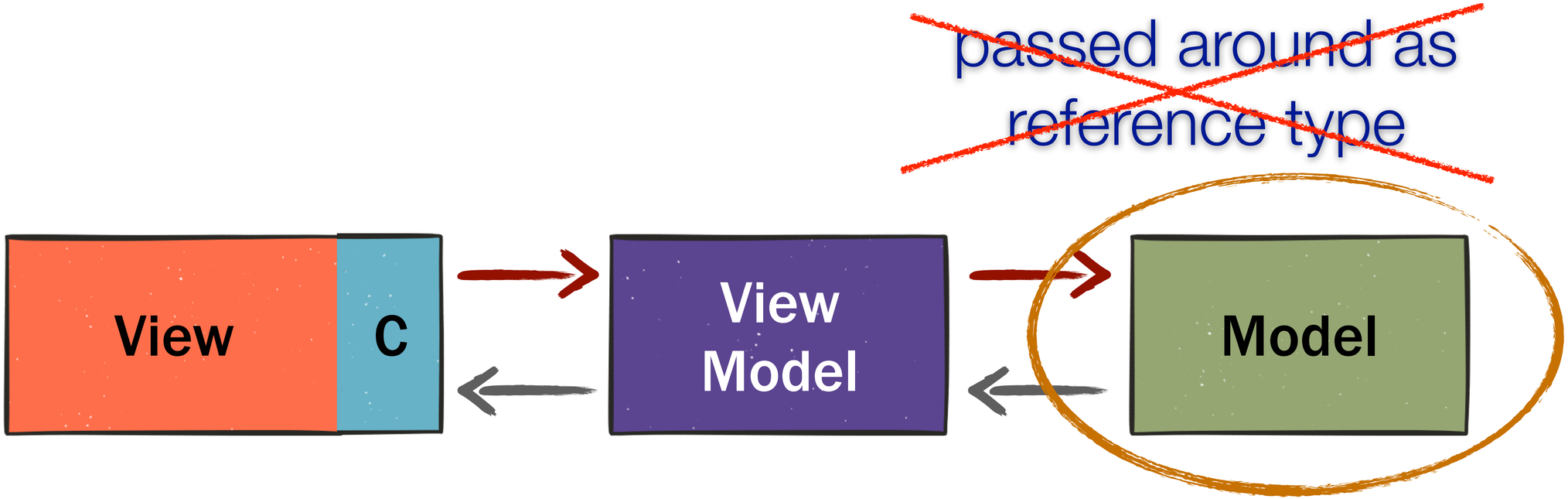


MVC & MVVM Architectures



MVC & MVVM Architectures

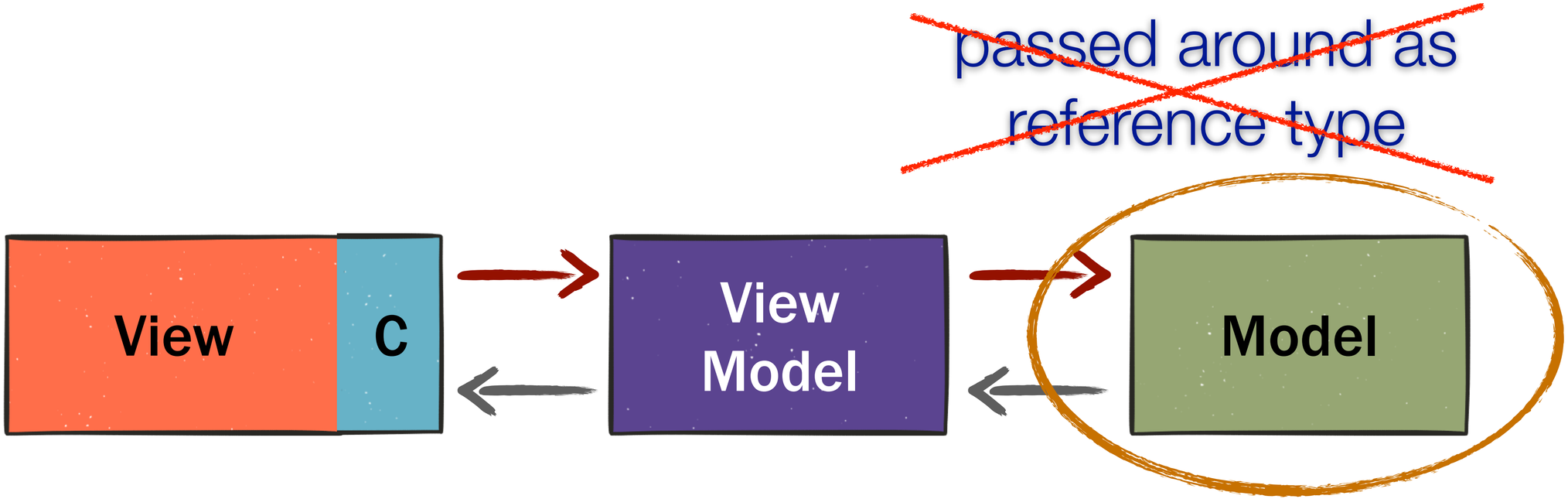
✓ Streams of changes instead of direct references



MVC & MVVM Architectures

✓ Streams of changes instead of direct references

✓ Change values are usually immutable value types



Immutable Goals

```
struct Goal {
  let uuid:      UUID                  // fast equality
  var colour:    UIColor
  var title:     String
  var interval:  GoalInterval
  var frequency: Int
}
```

Immutable Goals

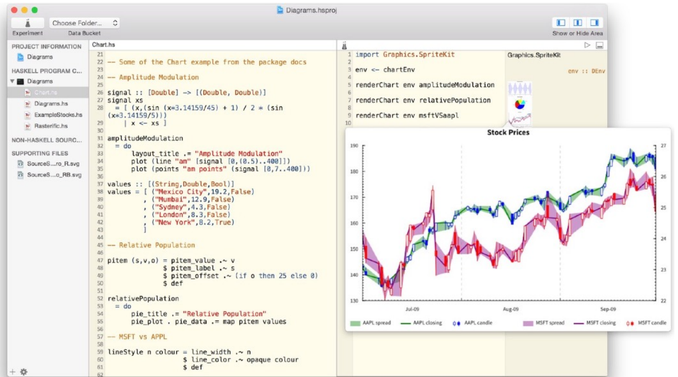
```
struct Goal {
  let uuid:      UUID                  // fast equality
  var colour:    UIColor
  var title:     String
  var interval:  GoalInterval
  var frequency: Int
}

typealias GoalProgress = (goal: Goal, count: Int?)
```

Immutable Goals

```
struct Goal {
  let uuid:      UUID                  // fast equality
  var colour:    UIColor
  var title:     String
  var interval:  GoalInterval
  var frequency: Int
}

typealias GoalProgress = (goal: Goal, count: Int?)

typealias Goals = [GoalProgress]  // array
```

Immutable Models Do Scale!

Haskell for Mac



"Functional Programming in a Stateful World"

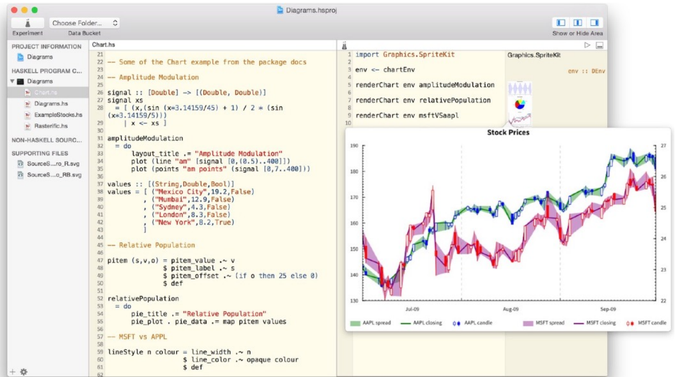
YOW! Lambda Jam 2015

(on speakerdeck.com)

Immutable Models Do Scale!

“A Type is Worth a Thousand Tests”
YOW! Connected 2016
(on speakerdeck.com)

Haskell for Mac



“Functional Programming in a Stateful World”
YOW! Lambda Jam 2015
(on speakerdeck.com)

Immutable Models Do Scale!

Redux

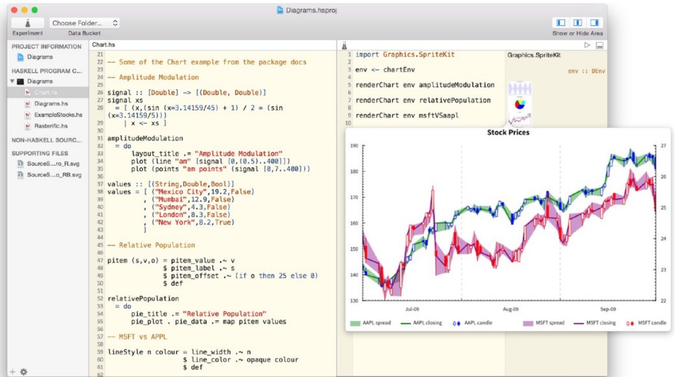
redux.js

“A Type is Worth a Thousand Tests”

YOW! Connected 2016

(on speakerdeck.com)

Haskell for Mac



“Functional Programming in a Stateful World”

YOW! Lambda Jam 2015

(on speakerdeck.com)

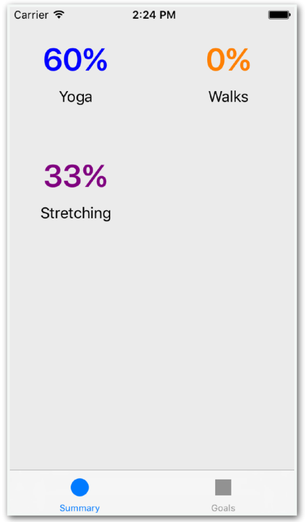
Change Propagation in Goals

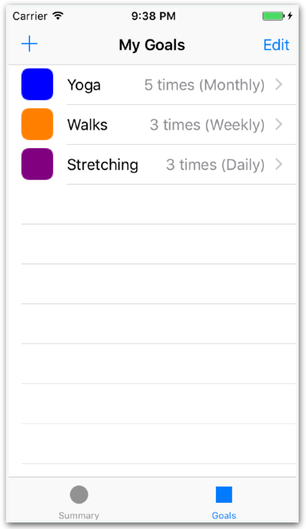
G0	3
G1	nil
G2	1

Views

Model

Change Propagation in Goals





observe

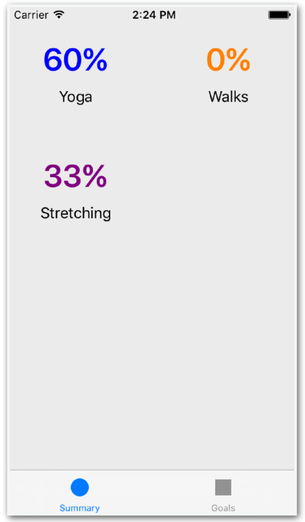
G0	3
G1	nil
G2	1

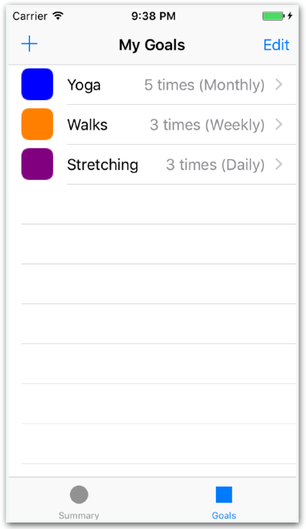
observe

Views

Model

Change Propagation in Goals





observe

merge

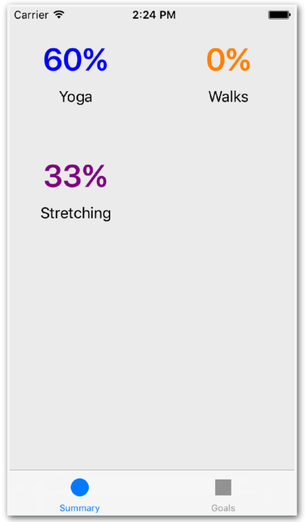
observe

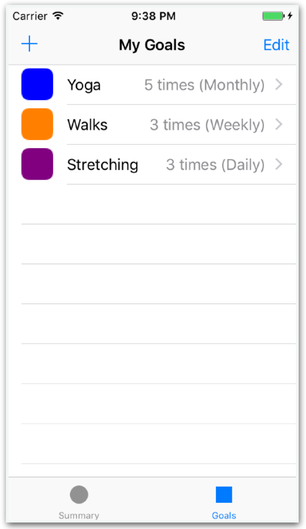
G0	3
G1	nil
G2	1

Views

Model

Change Propagation in Goals





observe

Changes<ProgressEdit>

merge

Changes<GoalEdit>

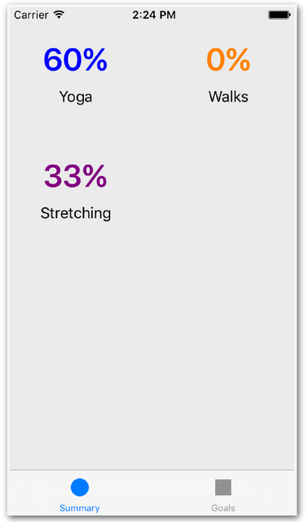
observe

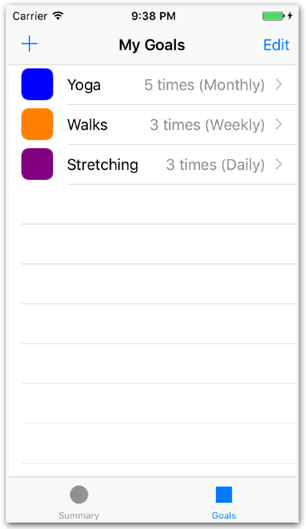
G0	3
G1	nil
G2	1

Views

Model

Change Propagation in Goals





observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

G0	3
G1	nil
G2	1

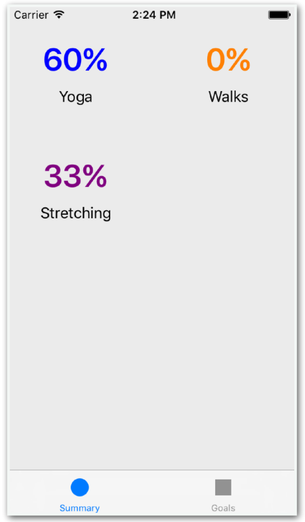
Changes<GoalEdit>

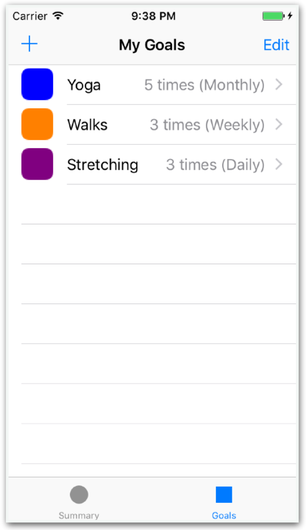
observe

Views

Model

Change Propagation in Goals





observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

G0	3
G1	nil
G2	1

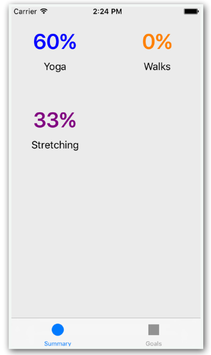
Changing<Goals>

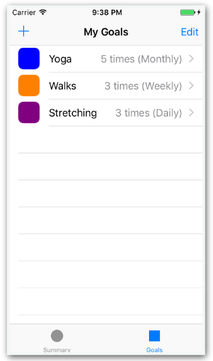
Changes<GoalEdit>

observe

Views

Model





observe

Changes<ProgressEdit>

Changes<Edit>

merge

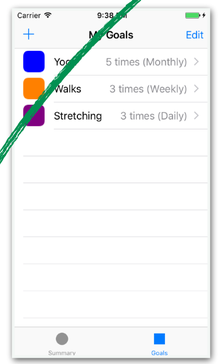
accumulate

G0	3
G1	nil
G2	1

Changes<GoalEdit>

Changing<Goals>

observe



observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

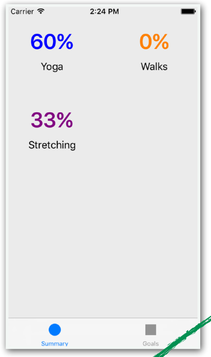
G0	3
G1	nil
G2	1

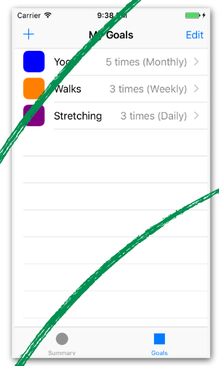
Changes<GoalEdit>

Changing<Goals>

observe

```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
```





observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

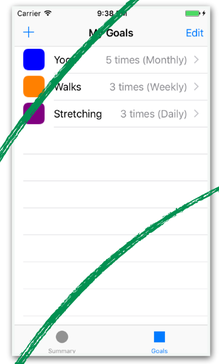
G0	3
G1	nil
G2	1

Changes<GoalEdit>

Changing<Goals>

observe

```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
```



observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

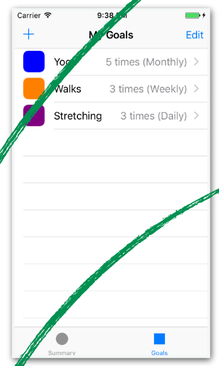
G0	3
G1	nil
G2	1

Changes<GoalEdit>

Changing<Goals>

observe

```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
  edits:          Changes<Edit>
```



observe

Changes<ProgressEdit>

Changes<Edit>

merge

accumulate

G0	3
G1	nil
G2	1

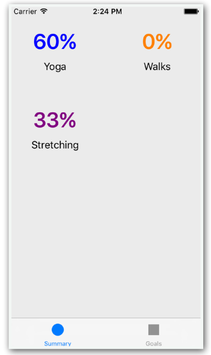
Changes<GoalEdit>

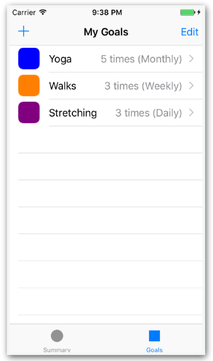
Changing<Goals>

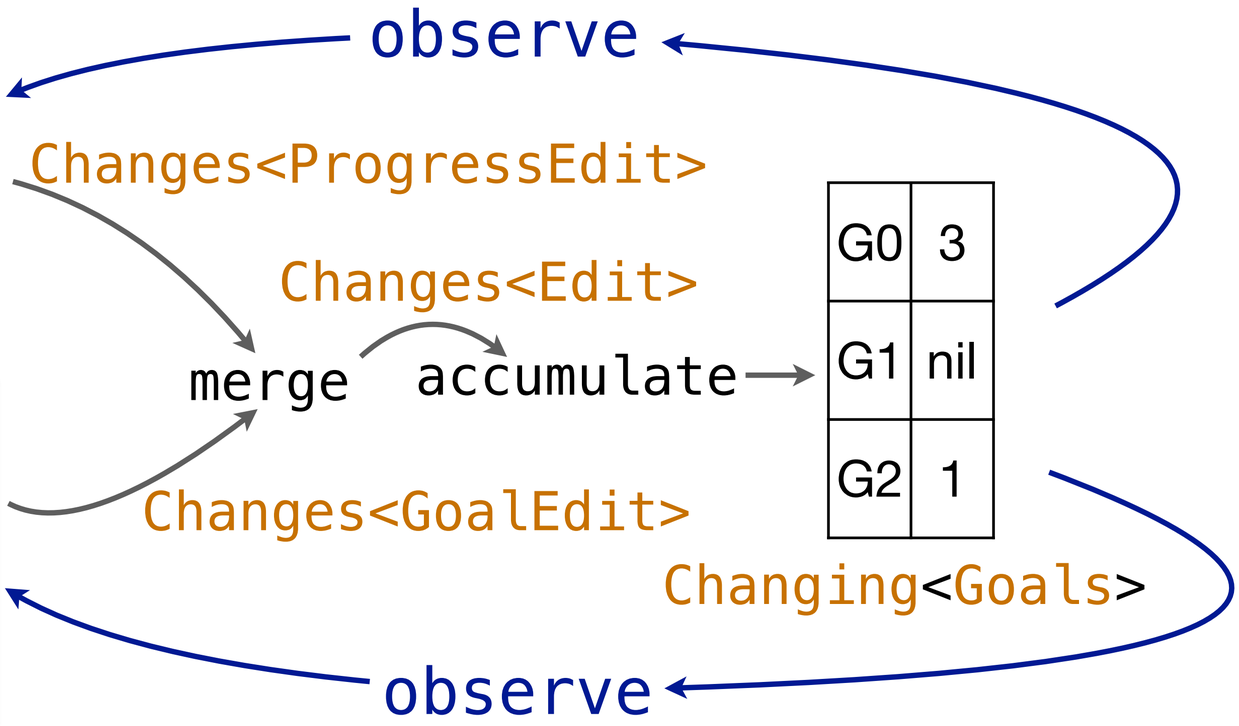
observe

```
struct GoalsModel {
  let
    progressEdits = Changes<ProgressEdit>()
    goalEdits     = Changes<GoalEdit>()
    edits:          Changes<Edit>
    model:          Changing<Goals>
}
```

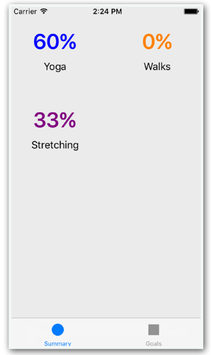
```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
  edits:          Changes<Edit>
  model:          Changing<Goals>
}
```

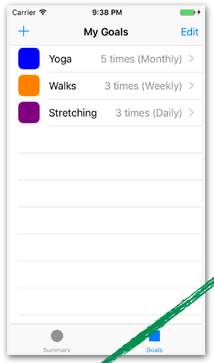


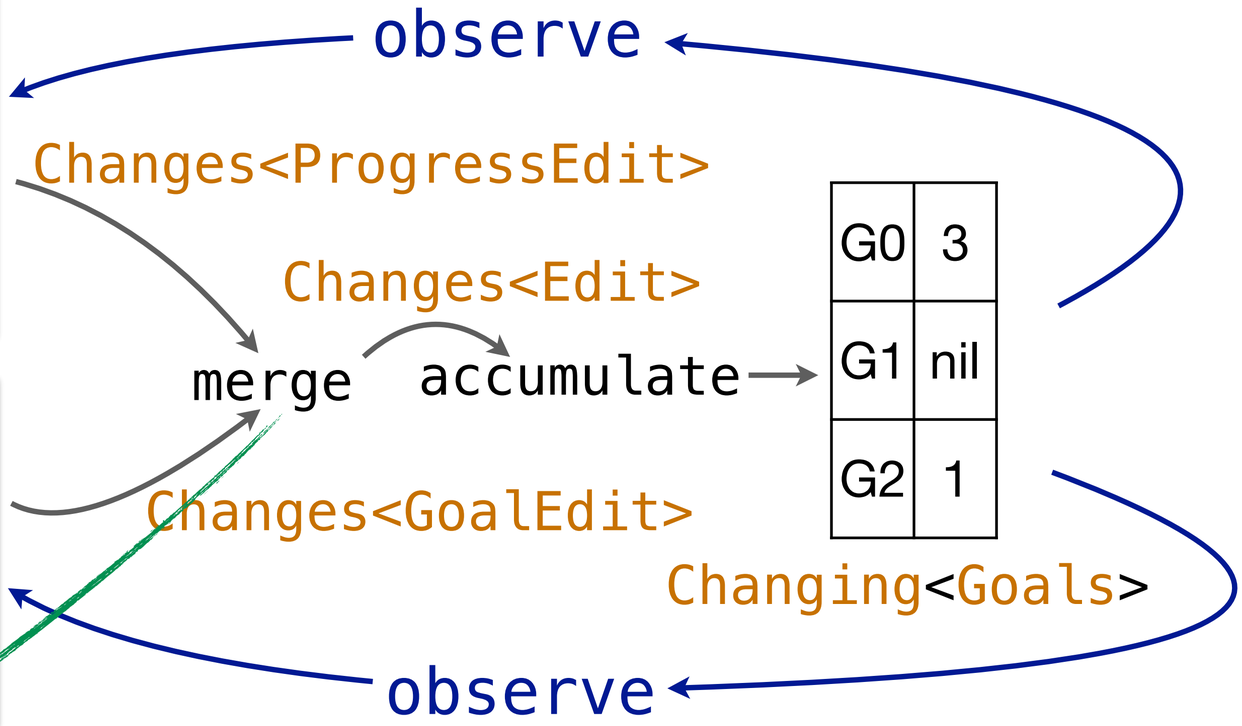




```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
  edits:          Changes<Edit>
  model:          Changing<Goals>
}
```

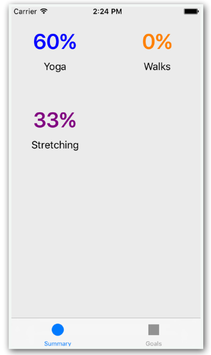


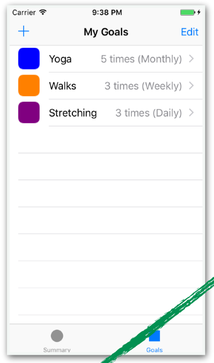


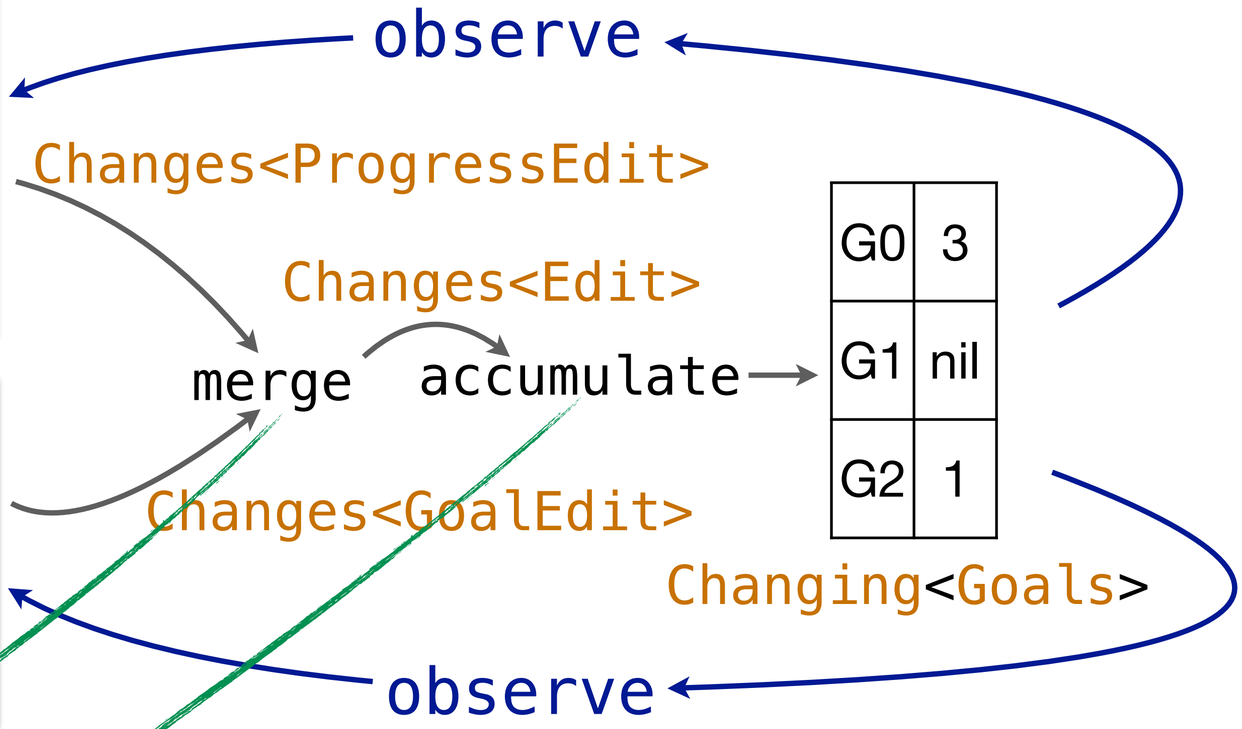



```
extension GoalsModel {
 init(initial: Goals) {
  edits = goalEdits.merge(right: progressEdits)
                   .map{ Edit(goalOrProgressEdit: $0) }
```

```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
  edits:          Changes<Edit>
  model:          Changing<Goals>
}
```

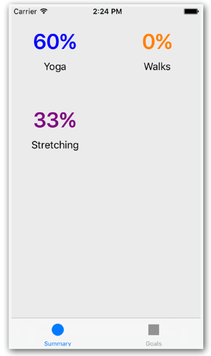


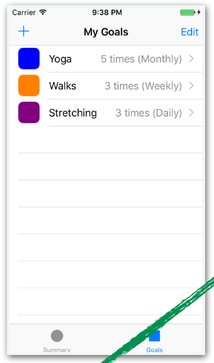


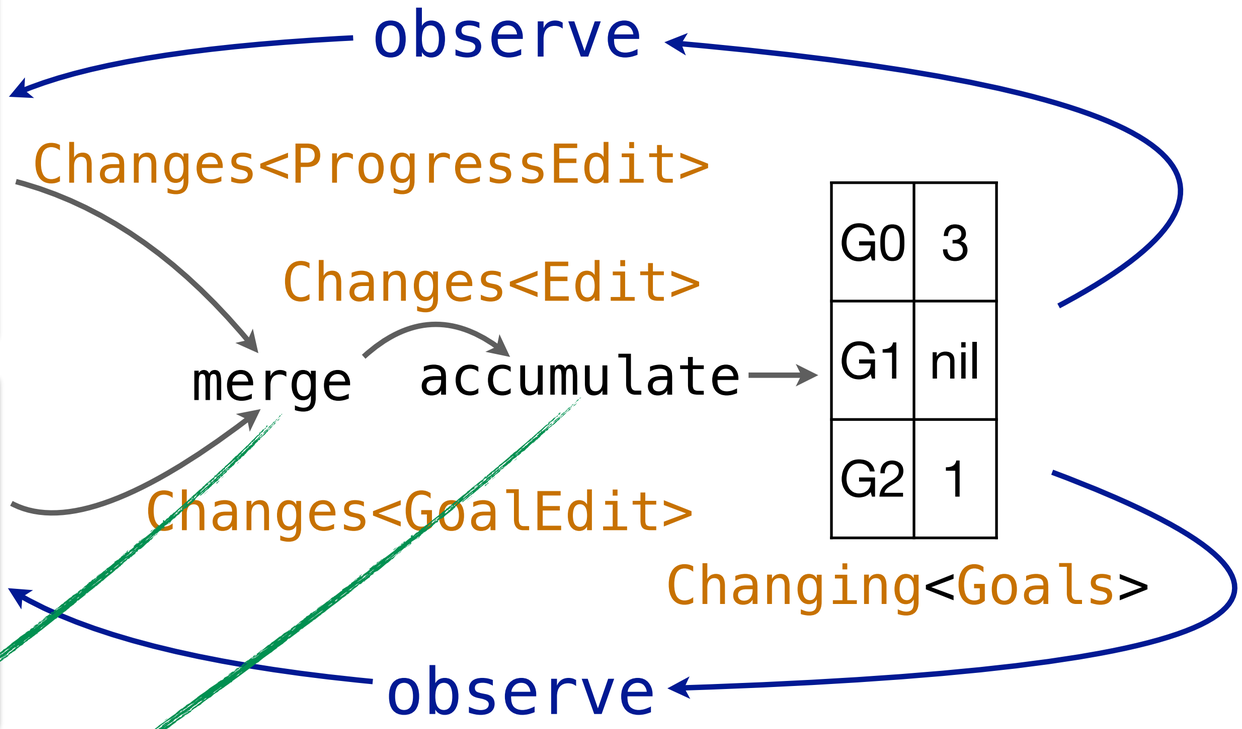



```
 extension GoalsModel {
  init(initial: Goals) {
   edits = goalEdits.merge(right: progressEdits)
                    .map{ Edit(goalOrProgressEdit: $0) }
   model = edits.accumulate(startingFrom: initial){ edit, current in
               return edit.transform(currentGoals)
           }
   } }
```

```
struct GoalsModel {
 let
  progressEdits = Changes<ProgressEdit>()
  goalEdits     = Changes<GoalEdit>()
  edits:          Changes<Edit>
  model:          Changing<Goals>
}
```








```
 extension GoalsModel {
  init(initial: Goals) {
   edits = goalEdits.merge(right: progressEdits)
                    .map{ Edit(goalOrProgressEdit: $0) }
   model = edits.accumulate(startingFrom: initial){ edit, current in
               return edit.transform(currentGoals)
           }
   } }
```

GoalEdits

ProgressEdits

Edits

```
typealias GoalEdits = Changes<GoalEdit>
```

ProgressEdits

Edits

```
typealias GoalEdits = Changes<GoalEdit>
```

ProgressEdits

Edits

```
typealias GoalEdits = Changes<GoalEdit>

enum GoalEdit {
  case add(goal: Goal)
  case delete(goal: Goal)
  case update(goal: Goal)
  case setActivity(activity: [Bool])
}
```

ProgressEdits

Edits

```
typealias GoalEdits = Changes<GoalEdit>

enum GoalEdit {
  case add(goal: Goal)
  case delete(goal: Goal)
  case update(goal: Goal)
  case setActivity(activity: [Bool])
}

extension GoalEdit {
 func transform(_ goals: Goals) -> Goals {
   switch self {
    case .add(let newGoal):
     …
```

ProgressEdits

Edits

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>
```

Edits

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>
```

Edits

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>

enum ProgressEdit {
  case bump(goal: Goal)
}
```

Edits

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>

enum ProgressEdit {
  case bump(goal: Goal)
}

extension ProgressEdit {
 func transform(_ goals: Goals) -> Goals { … }
}
```

Edits

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>

enum ProgressEdit {
  case bump(goal: Goal)
}

extension ProgressEdit {
 func transform(_ goals: Goals) -> Goals { … }
}
```

```
typealias Edits = Changes<Edit>
```

GoalEdits

```
typealias ProgressEdits = Changes<ProgressEdit>

enum ProgressEdit {
  case bump(goal: Goal)
}

extension ProgressEdit {
 func transform(_ goals: Goals) -> Goals { … }
}
```

```
typealias Edits = Changes<Edit>

enum Edit {
  case goalEdit(edit: GoalEdit)
  case progressEdit(edit: ProgressEdit)
}
```

Check out the source code of Goals.app!

`https://github.com/mchakravarty/goalsapp`

 types

 `haskellformac.com`

 `mchakravarty`

 `TacticalGrace`

 `justtesting.org`

>< state

Check out the source code of Goals.app!

`https://github.com/mchakravarty/goalsapp`

Change propagation by way of streams of changes

Changing values instead of mutable variables

Structured & typed change propagation with combinators

❤️ types


`haskellformac.com`


`mchakravarty`


`TacticalGrace`


`justtesting.org`

>< state

Thank you!

Image Attribution

https://pixabay.com/photo-1957451/

https://pixabay.com/photo-36732/

https://pixabay.com/photo-787980/

YouTube

https://www.youtube.com/watch?v=AqmQKEPwYzI

https://openclipart.org/detail/463/heart