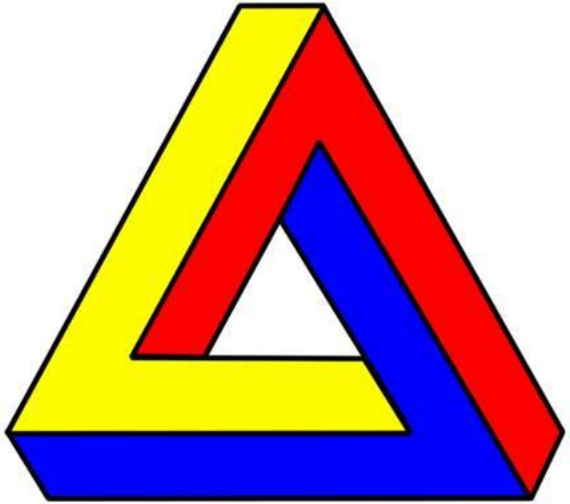
A Playful Introduction To Rx

Hacking should be fun



Erik Meijer
www.applied-duality.com

Manifesto | Become a Programmer, Motherfucker | Buy a T-Shirt, Asshole!

Programming, Motherfucker

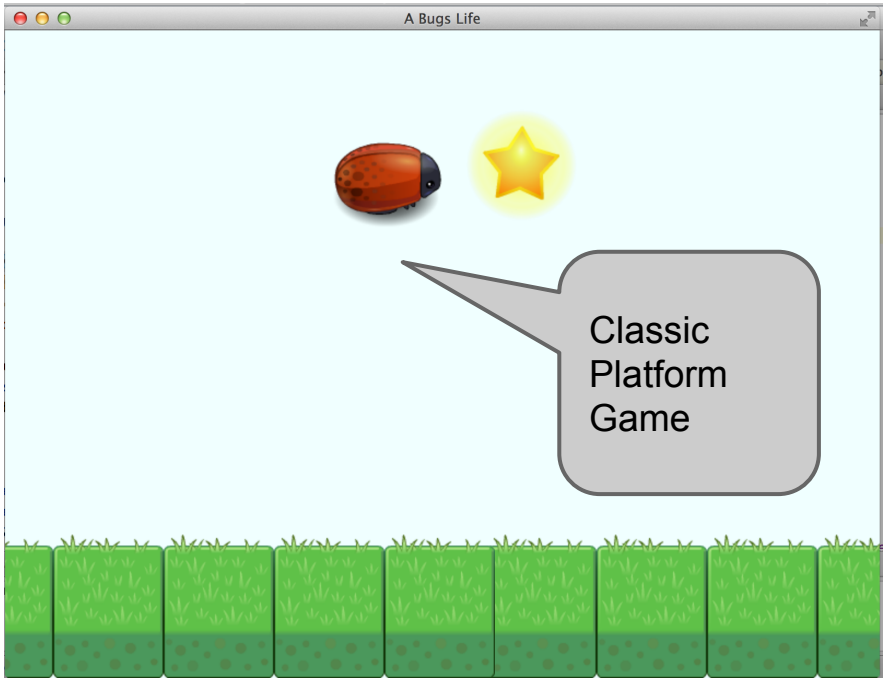
Do you speak it?

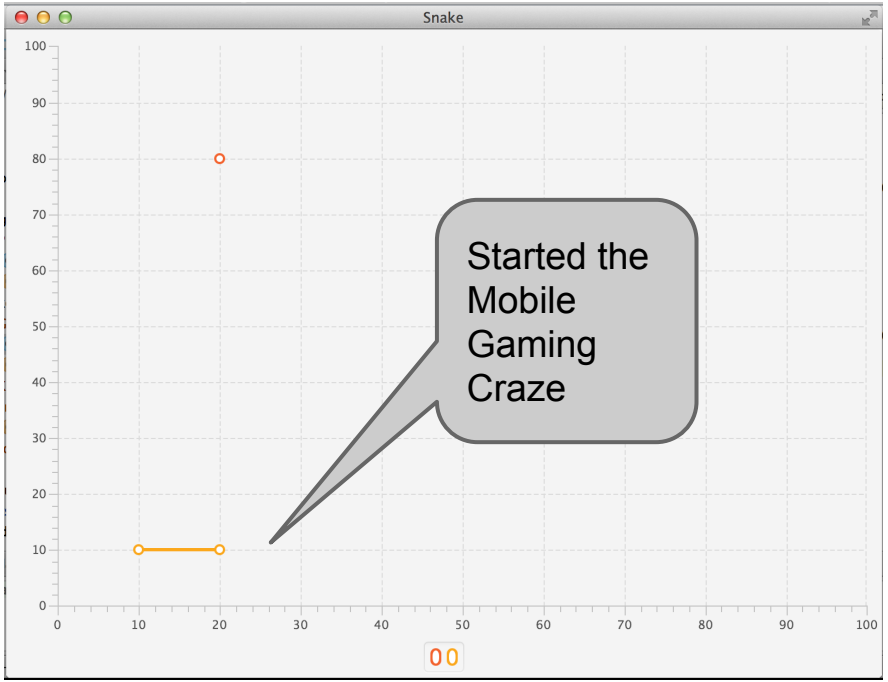
We are a community of motherfucking programmers who have been **humiliated** by software development methodologies for years.

We are tired of *XP*, *Scrum*, *Kanban*, *Waterfall*, *Software Craftsmanship* (aka *XP-Lite*) and anything else getting in the way of...**Programming, Motherfucker.**

http://programming-motherfucker.com/

Two Games


A Bugs Life
Classic Platform Game


Snake
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
00
Started the Mobile Gaming Craze

Source Code

A Bug's Life

```
val clock = Observable.timer(initialDelay = 0 seconds, period = (1/60.0) second, scheduler)
  .observeOn(PlatformScheduler())

val sky = new Canvas(screenWidth,screenHeight) {
  val context = getGraphicsContext2D
  context.setFill(Color.AZURE)
  context.fillRect(0, 0, screenWidth,screenHeight)
  root.getChildren.add(this)
}

val grass = new {
  val tile = new Image(s"$resourceDir/GrassBlock.png")
  val height = tile.getHeight
  val nrTiles = math.ceil(screenWidth/tile.getWidth).asInstanceOf[Int]+1

  val tiles = (0 to nrTiles-1).map(i =>
    new ImageView {
      setImage(tile)
      setTranslateX(i*getImage.getWidth)
      root.getChildren.add(this)
    }).toList

  val v0 = 1
  clock.scan(v0)((dX,_)=>dX).subscribe(dX =>
    tiles.foreach(tile =>
      tile.setTranslateX(
        if(tile.getTranslateX <= -tile.getImage.getWidth) {
          screenWidth-dX
        } else {
          tile.getTranslateX-dX
        }))))
}

val sun = new ImageView {
  def showHeart(showHeart: Boolean): Unit = {
    setImage(
      if(showHeart)
        new Image(s"$resourceDir/Heart.png")
      else
        new Image(s"$resourceDir/Star.png")
    )
  }

  showHeart(false)
  setTranslateY(-(screenHeight-200))
  root.getChildren.add(this)

  val v0 = 3
  clock.scan(v0)((vi,_)=>vi).subscribe(dX =>
    setTranslateX(
      if(getTranslateX <= -getImage.getWidth) {
        screenWidth-dX
      } else {
        getTranslateX-dX
      }))
}

val bug = new ImageView {

  val homeY = (-grass.height/2)-5
  val gravity = 0.1

  setImage(new Image(s"$resourceDir/EnemyBug.png"))
  setTranslateY(homeY)
  setTranslateX(screenHeight/2)
  root.getChildren.add(this)

  val jumps: Subject[Double] = Subject()

  jumps.flatMap(v0 => clock.scan(v0)((vi,_)=>vi-gravity).takeUntil(jumps)).subscribe(dY =>
    setTranslateY(
      if(getTranslateY < homeY+dY) {
        getTranslateY-dY
      } else {
        homeY
      }))
```

Snake

```
val gameOver = Subject[Boolean]()
val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 seconds)
  .takeUntil(gameOver)
  .observeOn(PlatformScheduler())
  ._share

val preyModel: ObservableList[Data[Number, Number]] = FXCollections.observableArrayList((0,0))
lineChart.getData.add(new XYChart.Series[Number, Number](preyModel))
val prey: Observable[List[(Int,Int)]] = preyModel.toObservableX.filter(change => {
  change.wasAdded
}).map(change => {
  asScalaBuffer(change.getList).toList.map(data =>
    (data.getXValue.intValue, data.getYValue.intValue))
})._share

val snakeModel: ObservableList[Data[Number, Number]] = FXCollections.observableArrayList(
  (10,10),(20,10))
lineChart.getData.add(new XYChart.Series[Number, Number](snakeModel))
val snake = snakeModel.toObservableX.filter(change => {
  change.wasAdded
}).map(change => {
  asScalaBuffer(change.getList).toList.map(data =>
    (data.getXValue.intValue, data.getYValue.intValue))
})._share

val eaten = preyModel.toObservableX.filter(change => change.wasRemoved)._share

val food = {
  val random = new Random()
  val delay = 5
  eaten.flatMapWith(_ => clock.drop(random.nextInt(delay)).take(1))((_,_) =>
    ((random.nextInt(8)+1)*10, (random.nextInt(8)+1)*10))
}

val directions = keyPresses(scene).map(_.getText).publish((keyBoard: Observable[String]) =>
  clock.join(keyBoard)(_ => empty, _ => keyBoard, (_,k) => k)
)._share

val hit = prey.map(p => snake.filter(s => s.last == p.last)).switch.map(_ => true)._share

val updates = directions.tumbling{ hit }.flatMap(directions =>
  directions.zipWithIndex.map({ case (d,i) => (d, i!=0)}))
)._share

hit.subscribe(x => {
  preyModel.clear()
})

food.subscribe(position => {
  preyModel.add(position)
})

updates.subscribe(d => {
 val (x: Int, y:Int) = dataToTuple(snakeModel.get(snakeModel.size-1))
 if(x == 0 || x == 100 || y == 0 || y == 100) {
   gameOver.onCompleted()
 } else {
   val z = d._1 match {
     case "u" => snakeModel.add((x, y+10))
     case "d" => snakeModel.add((x, y-10))
     case "l" => snakeModel.add((x-10, y))
     case "r" => snakeModel.add((x+10, y))
   }
   if(d._2) snakeModel.remove(0)
 }
})

preyModel.clear()
```

Programming with Streams

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

The mother of all streams

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

In Java 8

java.util.stream

Interface Stream<T>

Type Parameters:

T - the type of the stream elements

All Superinterfaces:

AutoCloseable, BaseStream<T,Stream<T>>

Down the Rabbit Hole

```
interface BaseStream<T, S extends BaseStream<T,S>>
interface Stream<T> extends BaseStream<T, Stream<T>>

Stream<T>
BaseStream<T, Stream<T>>
BaseStream<T, BaseStream<T, Stream<T>>>
BaseStream<T, BaseStream<T, BaseStream<T, Stream<T>>>>
…
BaseStream<T, BaseStream<T, BaseStream<T, …>>>>>>>>>>>>>>>>>>>>>>>>
```

Compared to this, everything else is simple

BaseStream Is An Iterable

```
interface BaseStream<T, S extends BaseStream<T,S>>
  extends AutoClosable {
    Iterator<T> iterator()
}
```

Denial of fact
In this form of denial, someone avoids a fact by utilizing deception. This lying can take the form of an outright falsehood (commission), leaving out certain details to tailor a story (omission), or by falsely agreeing to something (assent, also referred to as "yessing" behavior). Someone who is in denial of fact is typically using lies to avoid facts they think may be painful to themselves or others.

http://en.wikipedia.org/wiki/Denial#Denial_of_fact

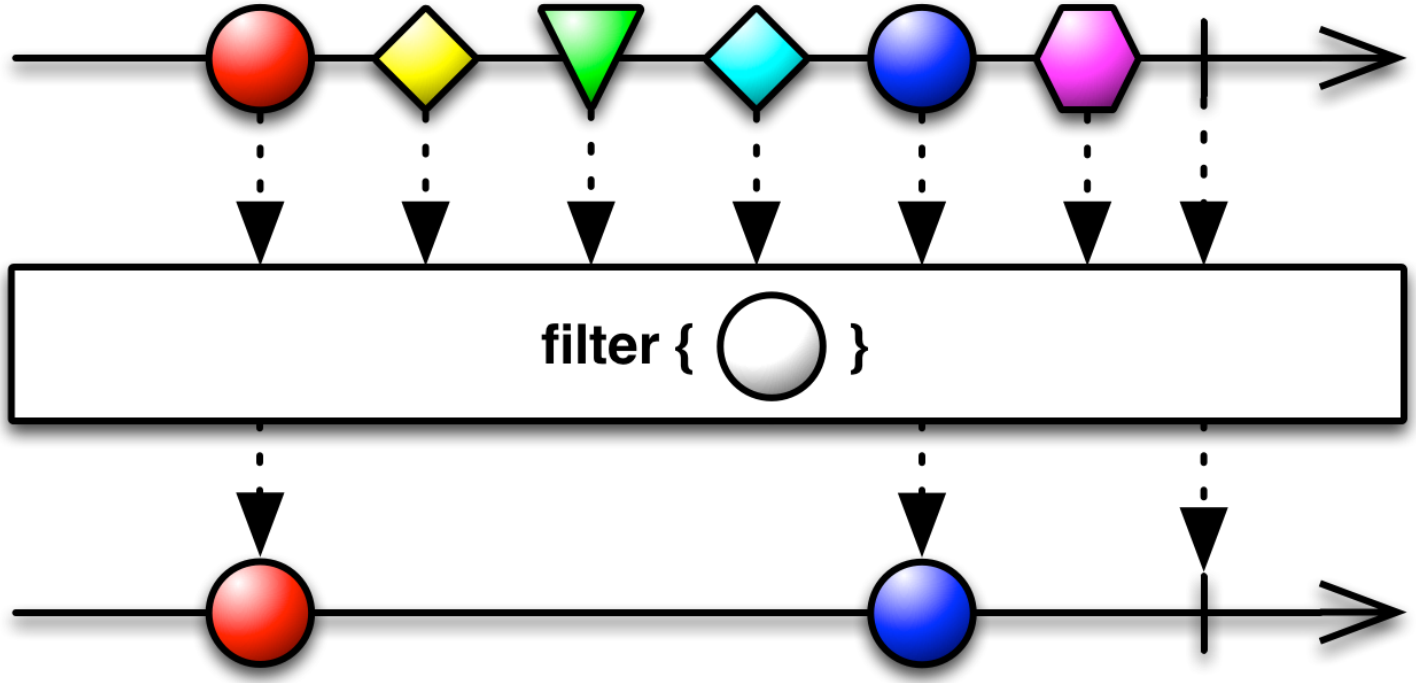
Stream Pipeline

```
int sum = widgets.stream()
            .filter(w -> w.getColor() == RED)
            .mapToInt(w -> w.getWeight())
            .sum();
```

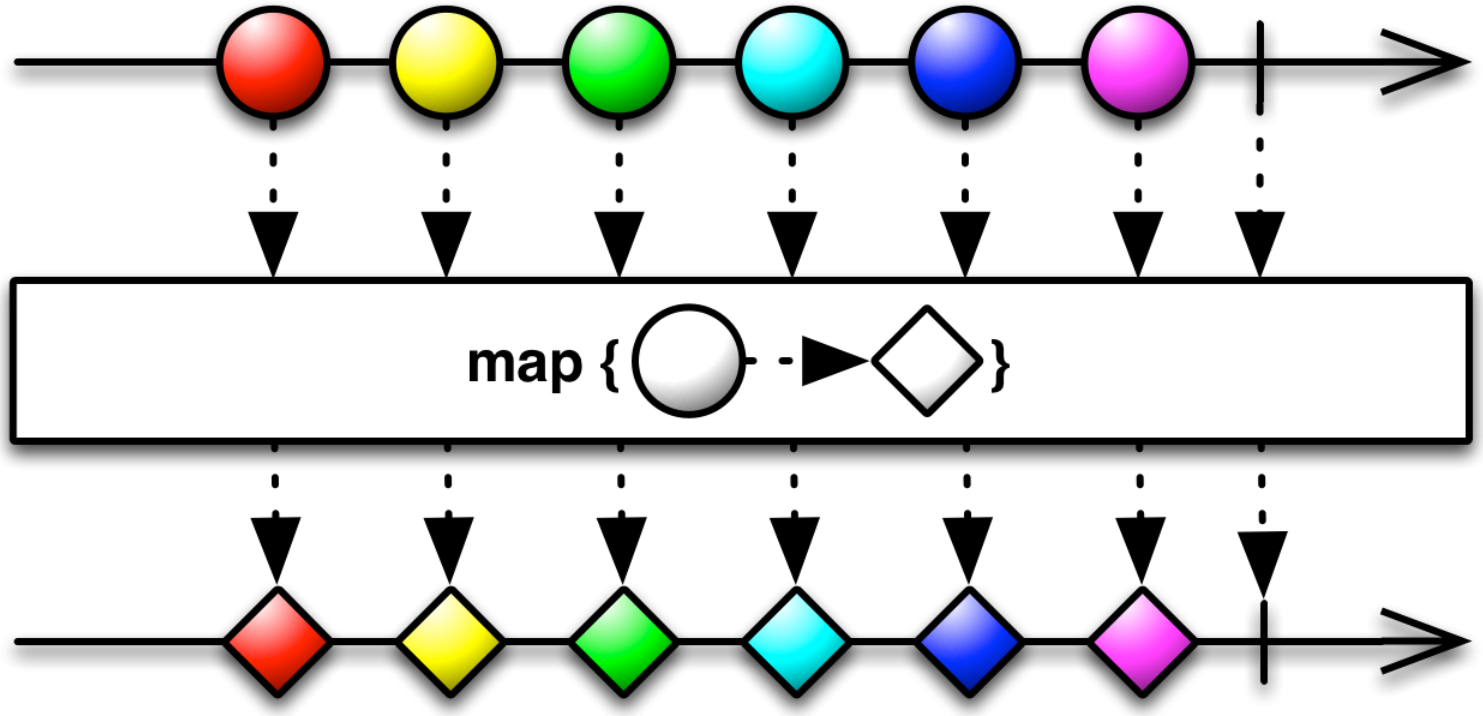
Because Java lacks extension methods

“Collections are primarily concerned with the efficient management of, and access to, their elements. By contrast, streams do not provide a means to directly access or manipulate their elements, and are instead concerned with declaratively describing their source and the computational operations which will be performed in aggregate on that source.”

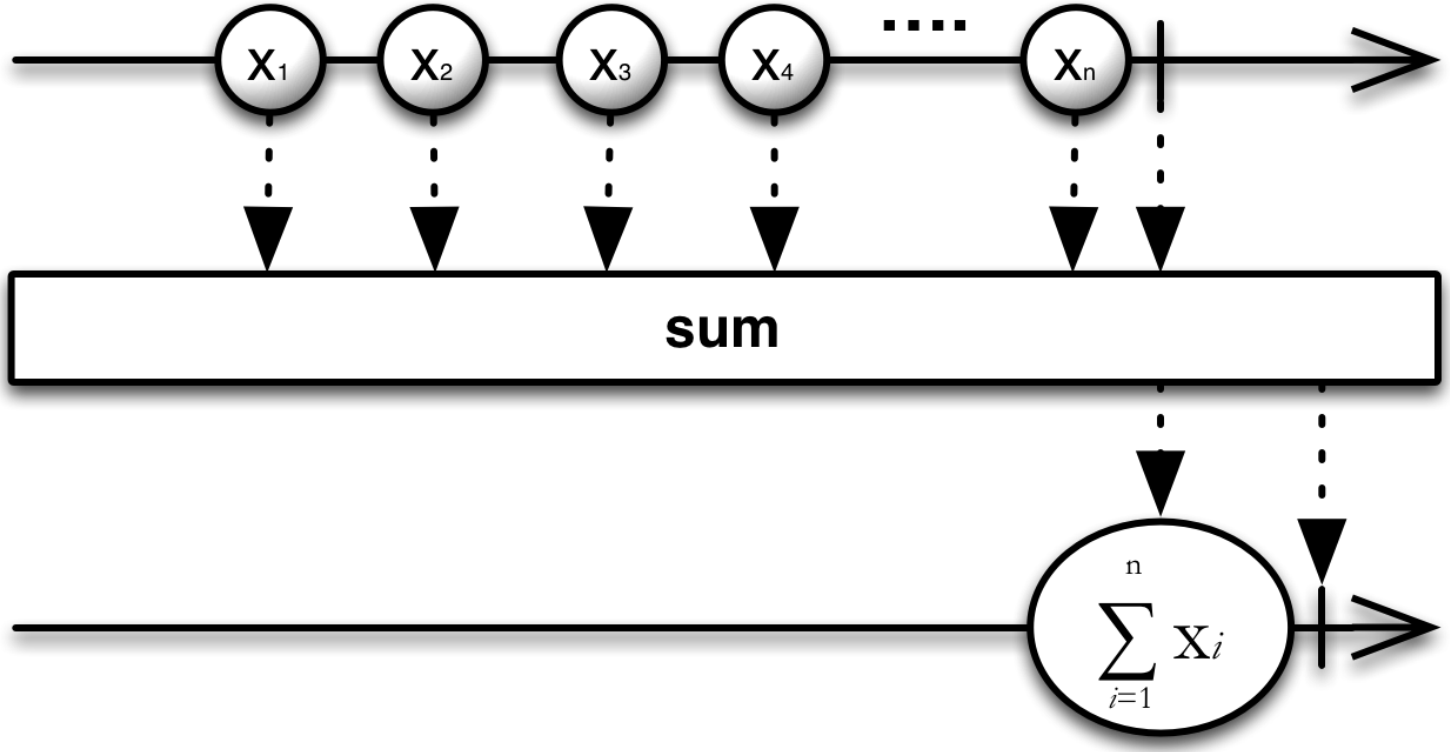
Filter



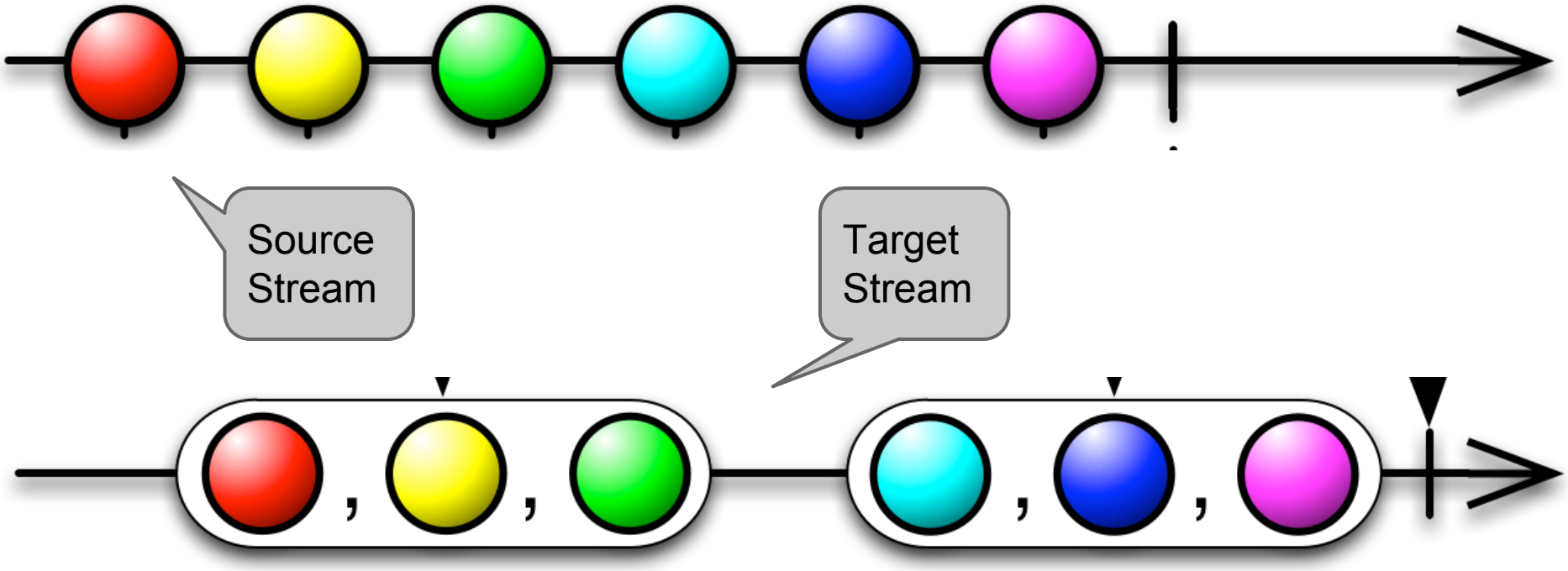
Map



Sum



Bulk Operations Over Streams

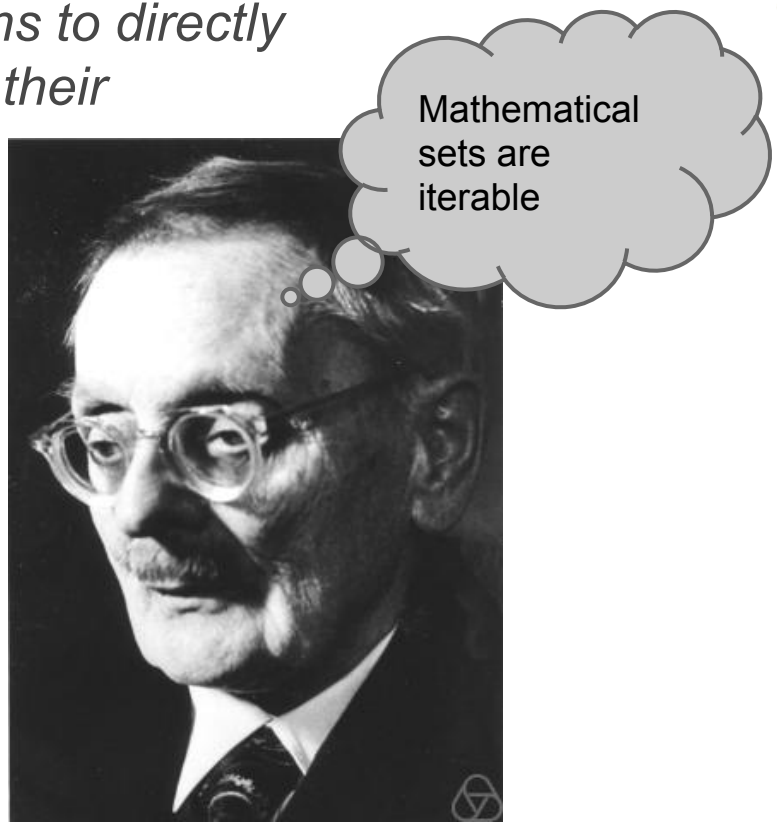


https://github.com/Netflix/RxJava/wiki

Streams

do not provide a means to directly access or manipulate their elements

```
interface Iterable<T> {
   Iterator<T> iterator()
}

interface Iterator<T> {
   boolean hasNext()
   T next()
}
```



Ernst Friedrich Ferdinand Zermelo (1871–1953)

http://en.wikipedia.org/wiki/Ernst_Zermelo#mediaviewer/File:Ernst_Zermelo.jpeg

```
class MainJava {

  public static void main(String[] args) {
      RandomStream integers = new RandomStream();
      integers.stream().limit(5).forEach(System.out::println);
  }
}

class RandomStream implements Iterable<Integer> {
    public Stream<Integer> stream() {
        return StreamSupport.stream(this.spliterator(), false);
    }

    @Override
    public Iterator<Integer> iterator() {
        final Random random = new Random();
        return new Iterator<Integer>(){
            @Override public boolean hasNext() { return true;}
            @Override public Integer next() { return random.nextInt(); }
        };
    }
}
```

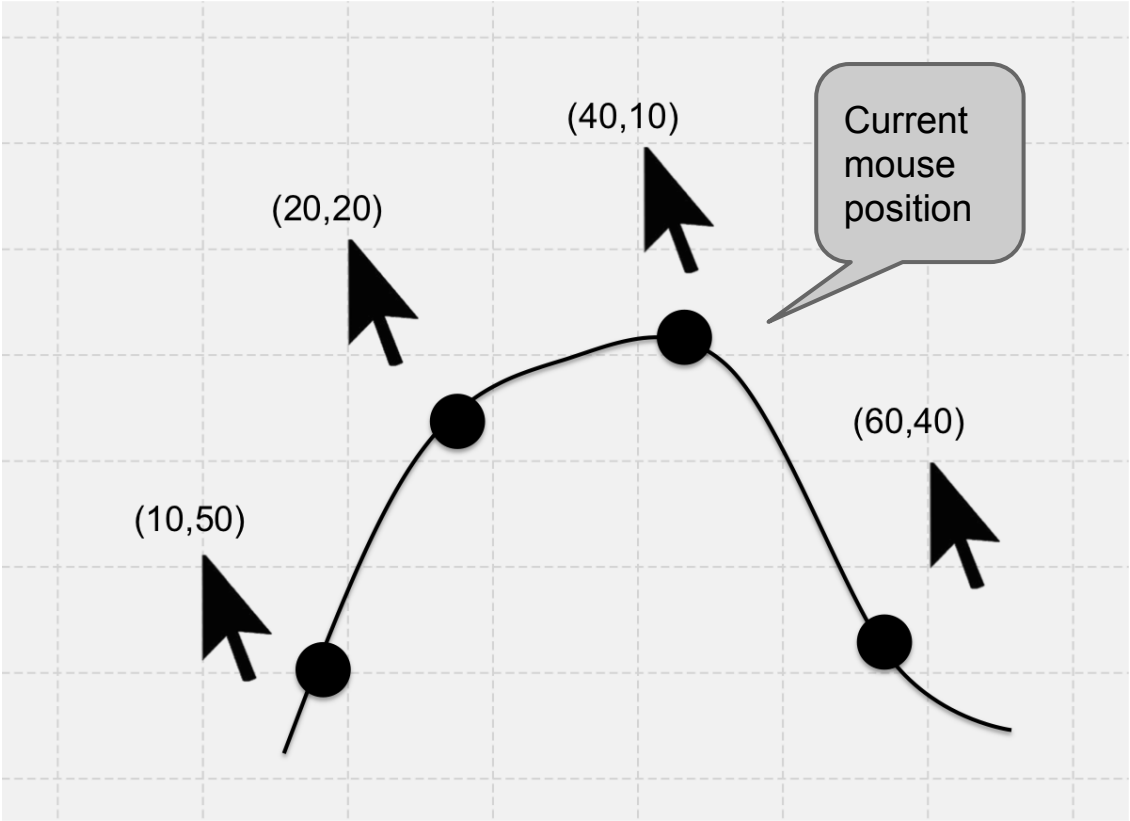
Elements of the stream do not exist before (and after) they are accessed.

Dalai Lama About Programming With Streams

http://en.wikipedia.org/wiki/Dalai_Lama

"*Live in the moment. Do not forget nor dwell on the past, but do forgive it. Be aware of the future but do no fear or worry about it.* ***Focus on the present moment, and that moment alone****.*"

See Streams Everywhere

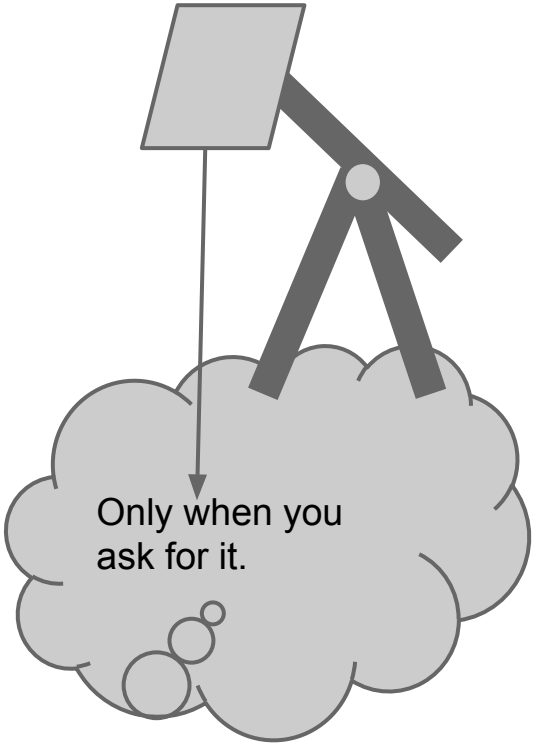


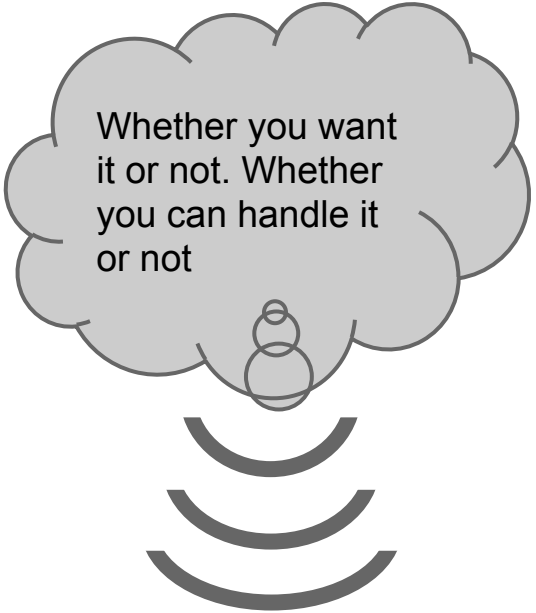
See Streams Everywhere

Screaming Babies

http://en.wikipedia.org/wiki/Baby_colic

These Streams Don’t Wait For You





Observable Streams

```
public class Observable <T> {
  Subscription subscribe(Observer<T> observer)
}
public interface Observer <T> {
    void onCompleted();
    void onError(Throwable error);
    void onNext(T value);
}
public interface Subscription {
    void unsubscribe();
    boolean isUnsubscribed();
}
```

Event source

Event handler

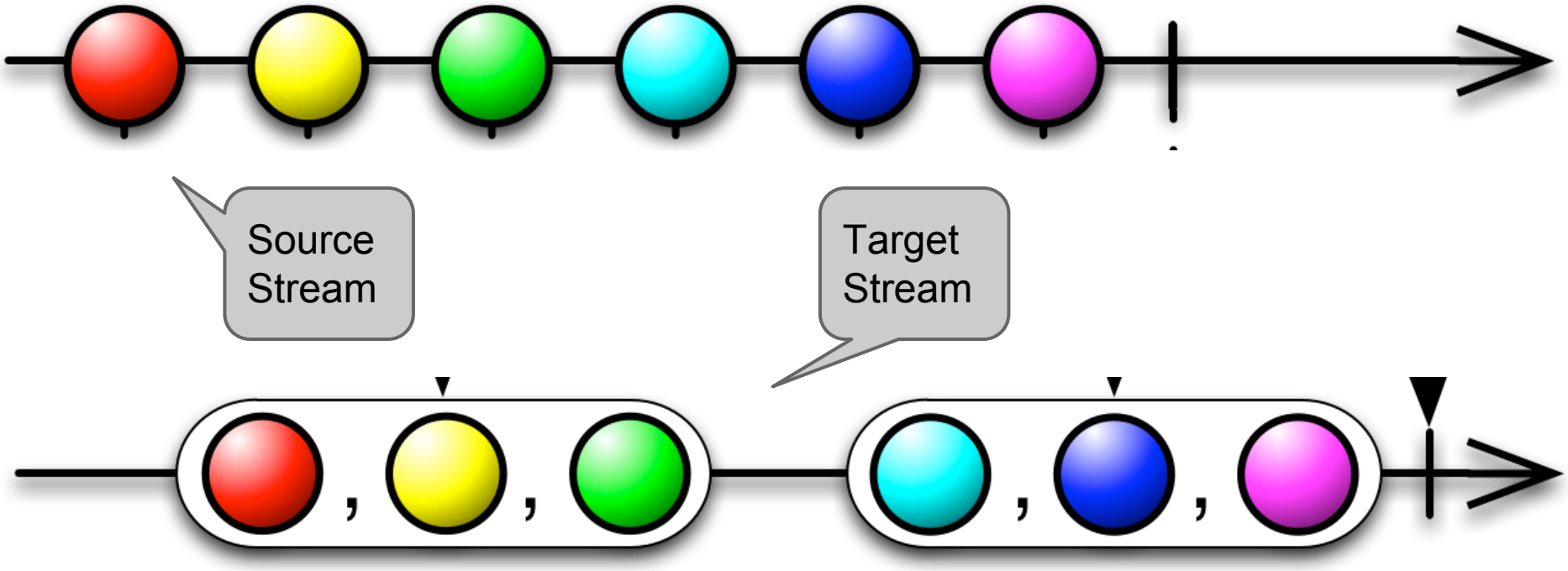
Autoclosable

callback invoked every time a new “present” happens

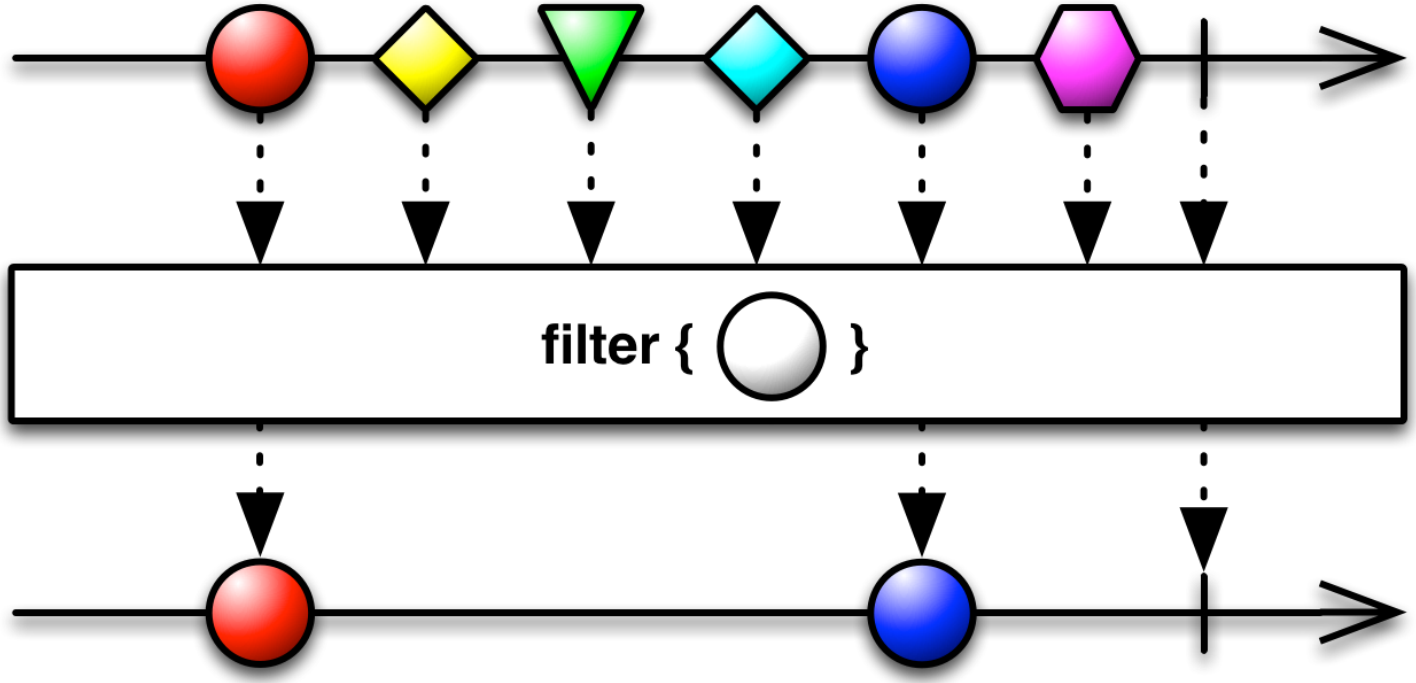
```
class MainJava {

  public static void main(String[] args) {
      RandomObservable integers = new RandomObservable();
      integers.take(5).subscribe(System.out::println);
  }
}

class RandomObservable extends Observable<Integer> {
    public RandomObservable() {
        super(subscriber -> {
            Random random = new Random();
            while(!subscriber.isUnsubscribed()) {
                subscriber.onNext(random.nextInt());
            }
        });
    }
}
```

Elements of the stream do not exist before (and after) they are accessed.

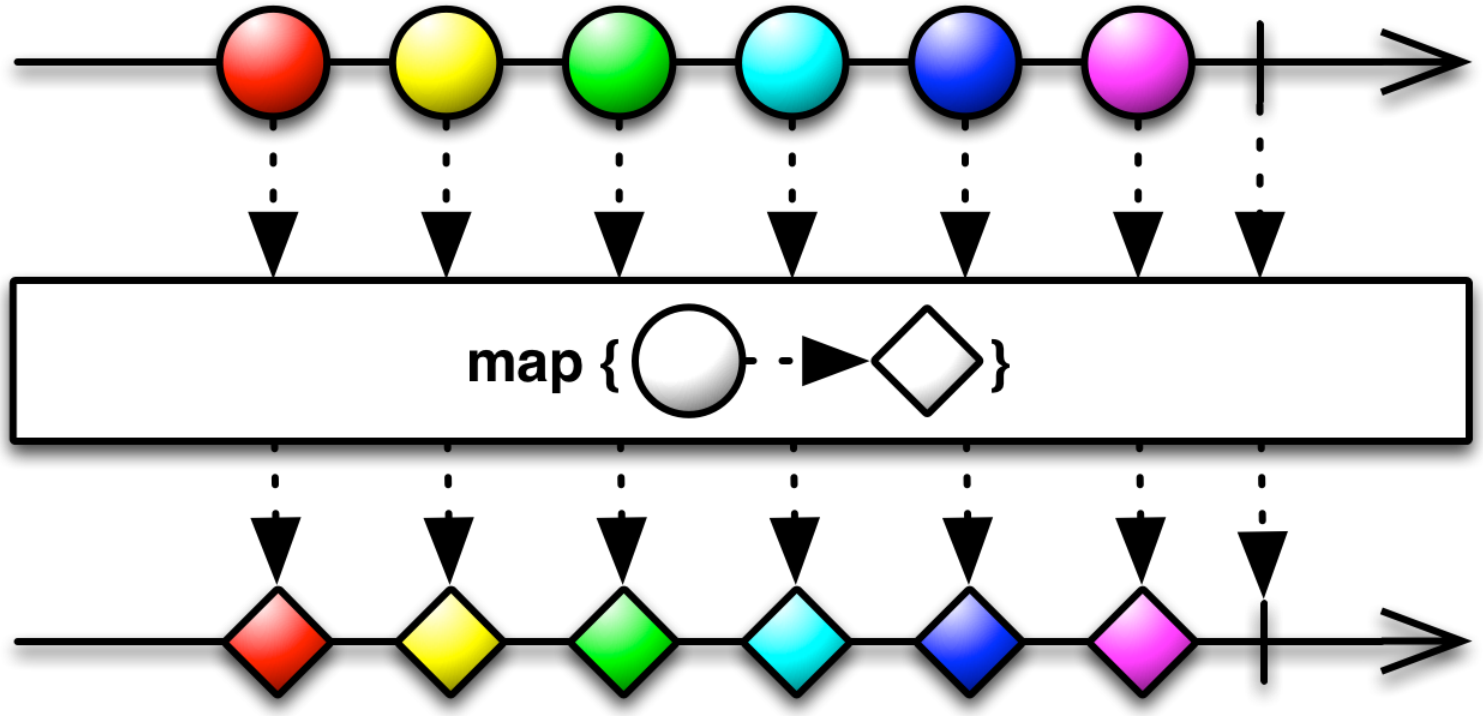
Bulk Operations Over Streams



Filter



Map



Observable Collections

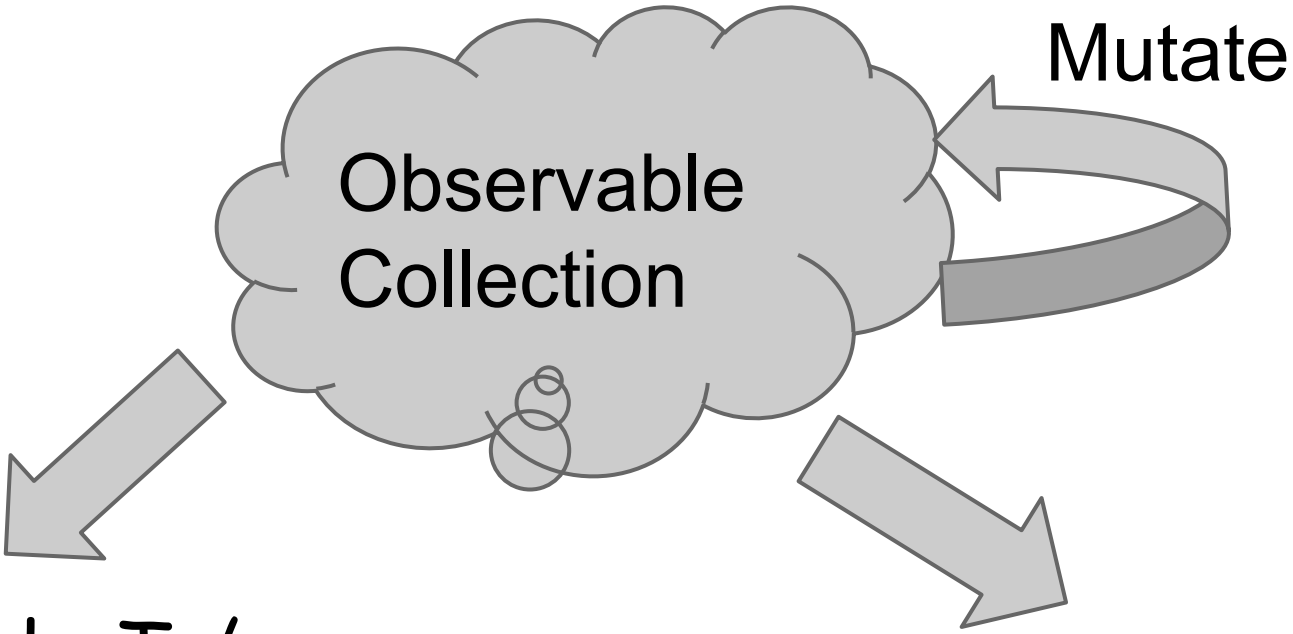
```
class ObservableCollection<T> : Collection<T>,
    INotifyCollectionChanged, INotifyPropertyChanged
```

Represents a dynamic data collection that provides notifications when items get added, removed, or when the whole list is refreshed.

```
public interface ObservableList<E>
extends java.util.List<E>, Observable
```

A list that allows listeners to track changes when they occur.

Cross The Streams



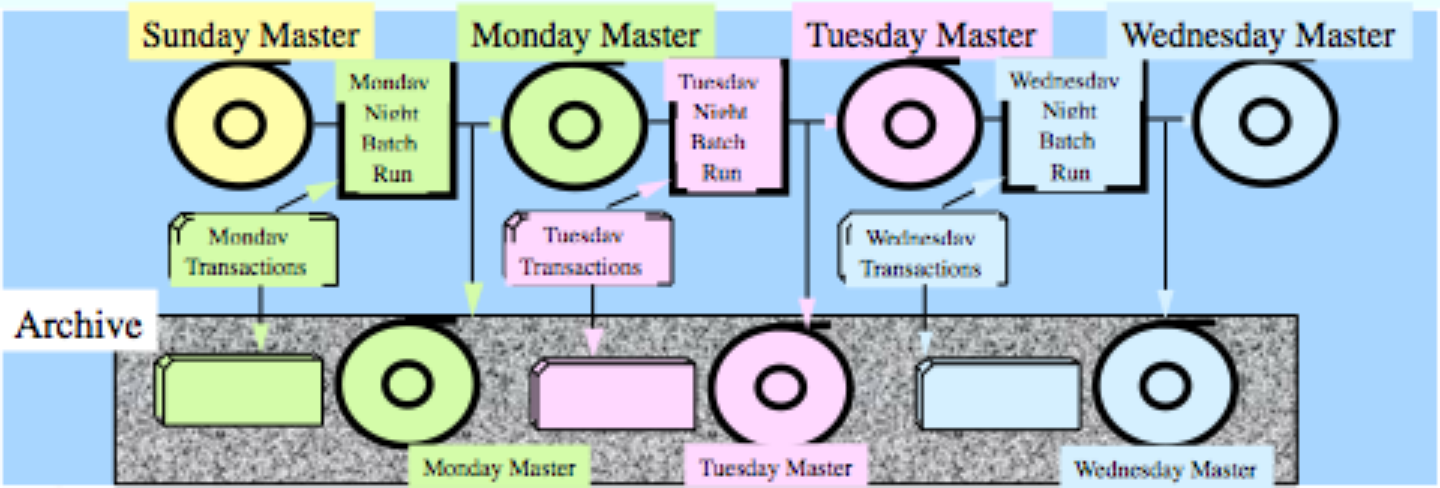
Iterable<T>/
Stream<T>

Observable<Change<T>>

Log Concept

Jim Gray & Andreas Reuter

- Log is a history of all changes to the state.
- Log + old state gives new state
- Log + new state gives old state (not in this picture)
- Log is a sequential file.
- Complete log is the complete history
- Current state is just a "cache" of the log records.



Why use a database when you have a log?

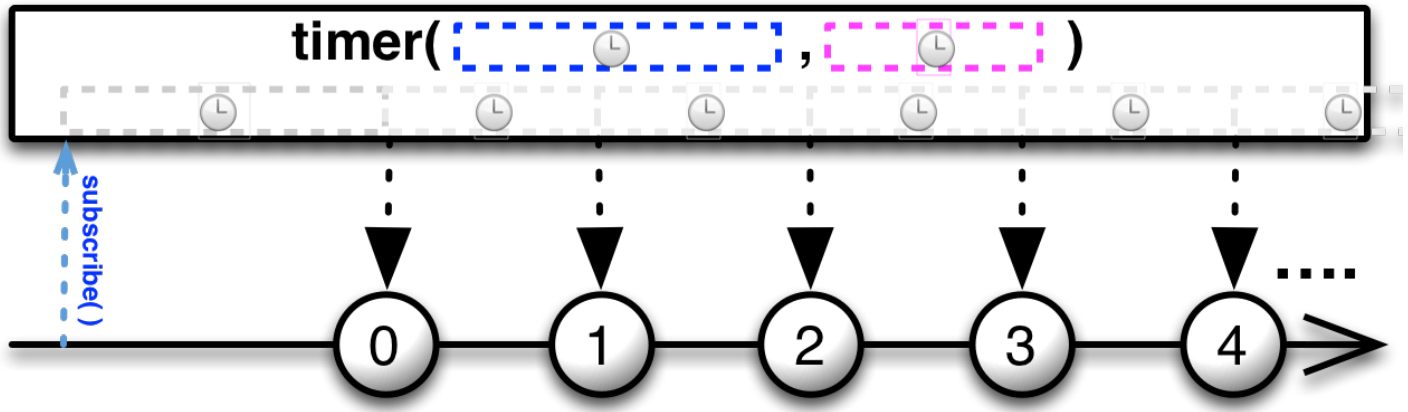
the past

The future mutates the present

the present

Time Is A Stream

```
val clock: Observable[Long] =
   Observable.timer(
      initialDelay = 0 seconds,
      period = 1 second
)
```



Computing Primes (naive)

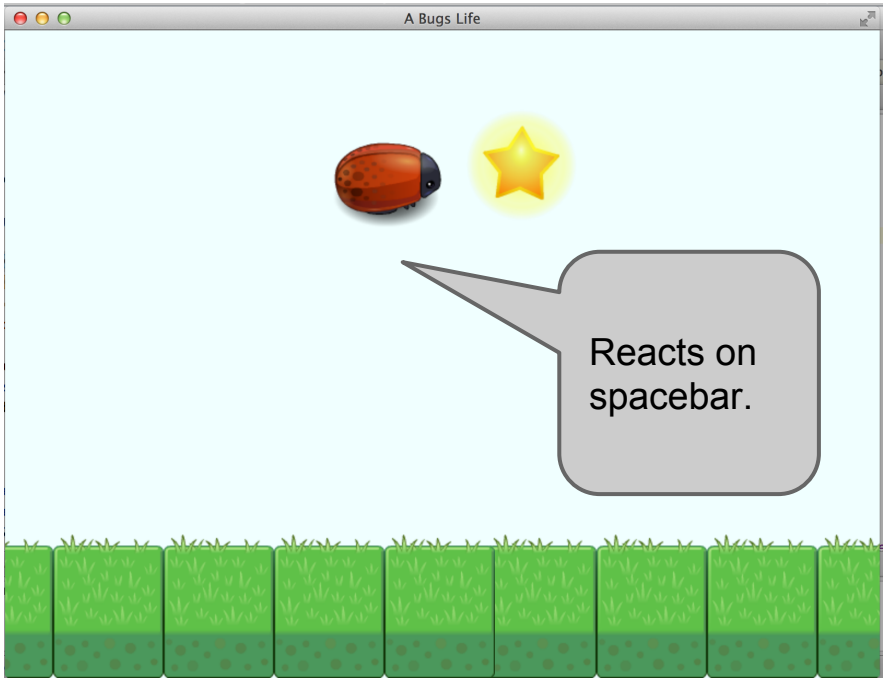
```
def factors(n: Long): Iterable[Long] =
    Seq.range(1L, n+1).filter(x => n%x == 0)
```

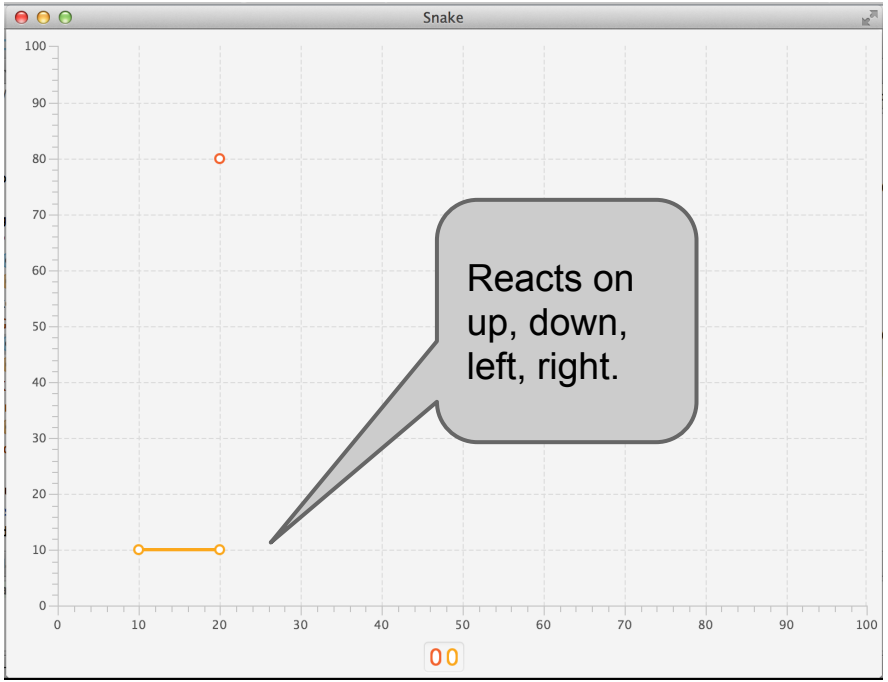
Iterable

```
def main(args: Array[String]) {
   val clock: Observable[Long] = Observable.timer(0 seconds, 1 second)

    val primes: Observable[Long] = clock.filter(n => factors(n) == Seq(1,n))

   primes.foreach(println)
   readLine()
 }
```

Observable

Your Keyboard is a Stream

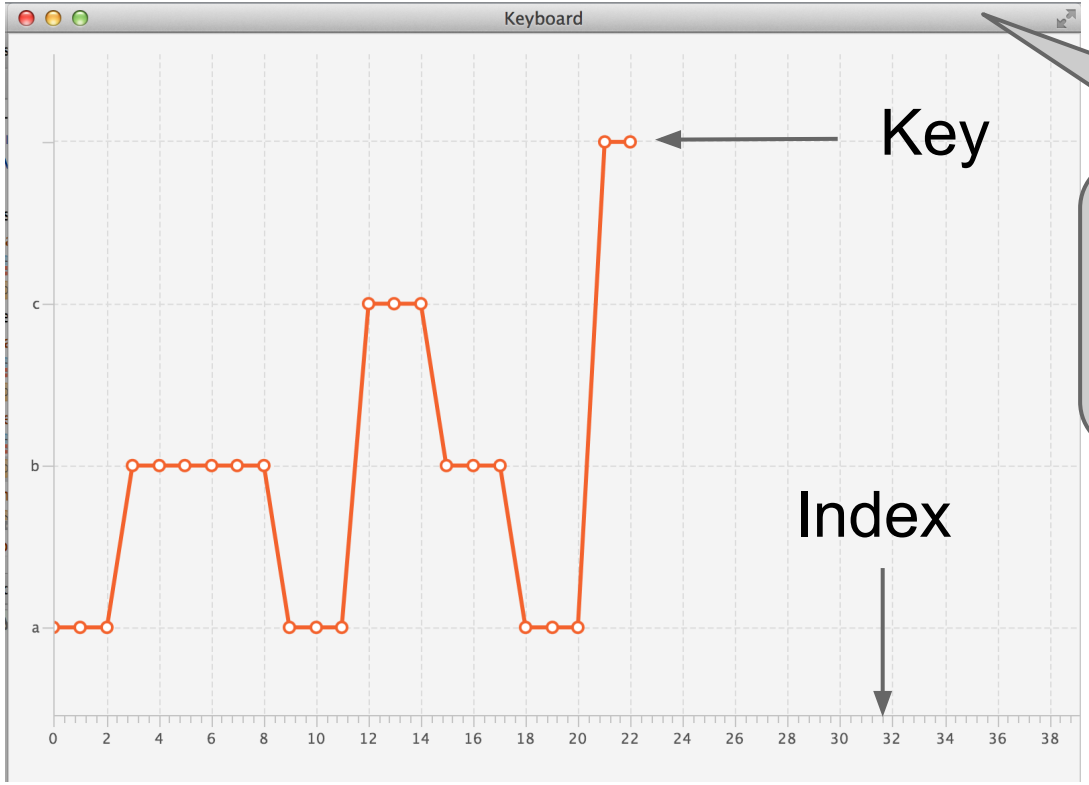




Events Are Streams

```
implicit def actionToEventHandler[E <: javafx.event.Event](f: E => Unit): EventHandler[E] =
  new EventHandler[E] { def handle(e: E): Unit  = f(e) }

 def keyPresses (scene: Scene): Observable[KeyEvent] = Observable[KeyEvent](observer => {
  val handler: EventHandler[KeyEvent] = (e:KeyEvent) => observer.onNext(e)
  scene.addEventHandler(KeyEvent.KEY_PRESSED, handler)
  observer.add {
    scene.removeEventHandler(KeyEvent.KEY_PRESSED, handler)
  }
 })

 def spaceBars(scene: Scene): Observable[KeyEvent] =
   keyPresses(scene).filter(_.getCode == KeyCode.SPACE)
```

Showing Key Events In A Graph



JavaFx line chart of

Data[Number, String]

Show me the code

Line that is shown on the chart

```
val keys: Series[Number, String] = {
  val ks = new XYChart.Series[Number, String](); chart.getData.add(ks); ks
}
val keyboard: Observable[String] = keyPresses(scene).map(_.getText)

val withIndex: Observable[(Int, String)] = keyboard.zipWithIndex.map(_.swap)
```

Pair each keystroke with its index

Add each pair to the graph

```
withIndex.subscribe(s => keys.getData.add(s))
```

Show me the code

Model

```
val keys: ObservableList[Data[Number, String]] =
                                    FXCollections.observableArrayList()
```

View

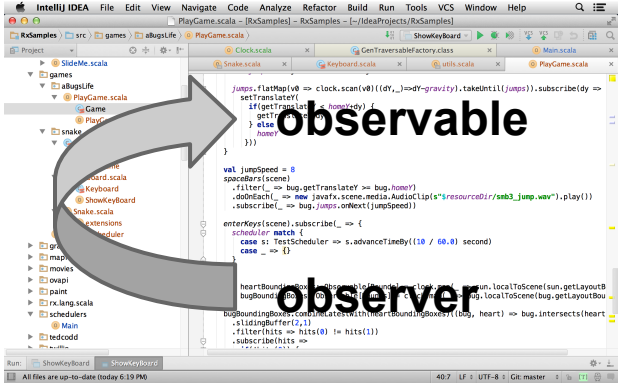
```
lineChart.getData.add(new XYChart.Series[Number, String](keys))
```

```
val keyboard: Observable[String] = keyPresses(scene).map(_.getText)
val withIndex: Observable[(Int, String)] = keyboard.zipWithIndex.map(_.swap)
```

Controller

```
withIndex.subscribe(s => keys.add(s))
```

Rx For Astronaut Architects

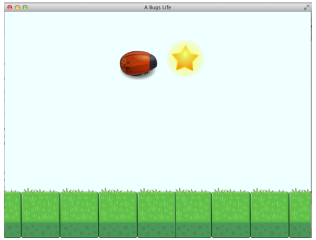


mutate state

observe changes

Observable Collection

UI



data binding

Sampling keyboard

```
val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 second)
val sample = keyboard.map(key => clock.map(i => (i,key))).switch
```

for each keystroke

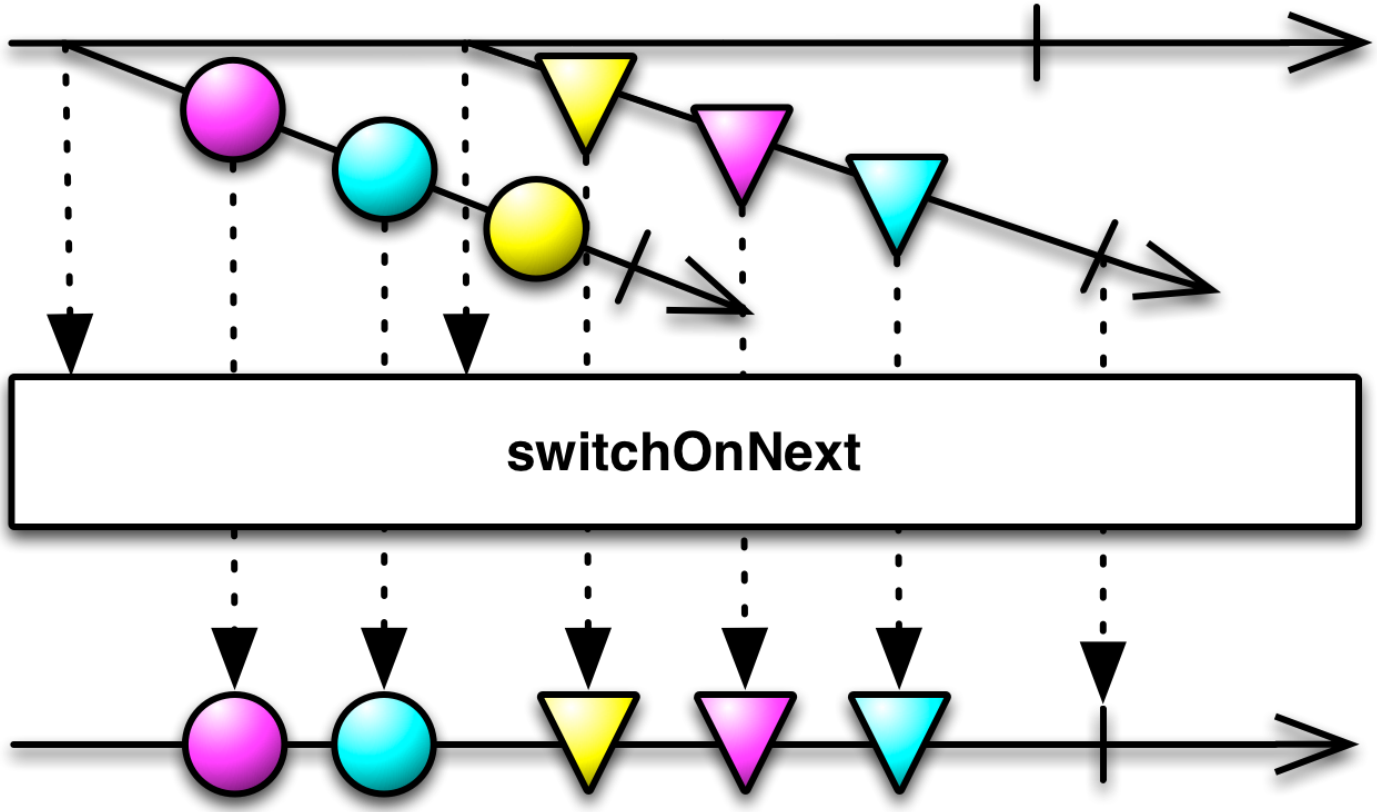
Pair keystroke with clock ticks

Now we have a nested stream

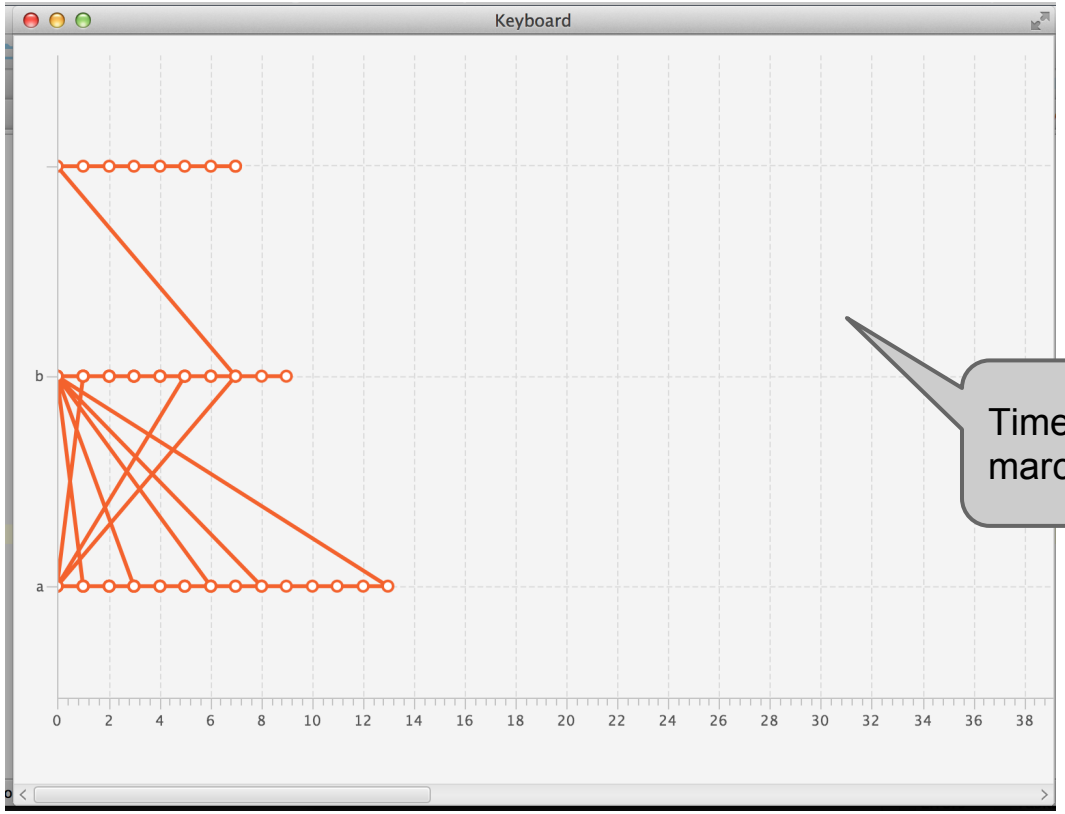
Switch whenever a new keystroke happens

```
sample.subscribe(s => keys.getData.add(s))
```

Switching Streams



Sampling Every Tick (With Bug)



Time does not
march forward :-(

Cold Observable

```
val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 second)
val sample = keyboard.map(key => clock.map(i => (i,key))).switch
```

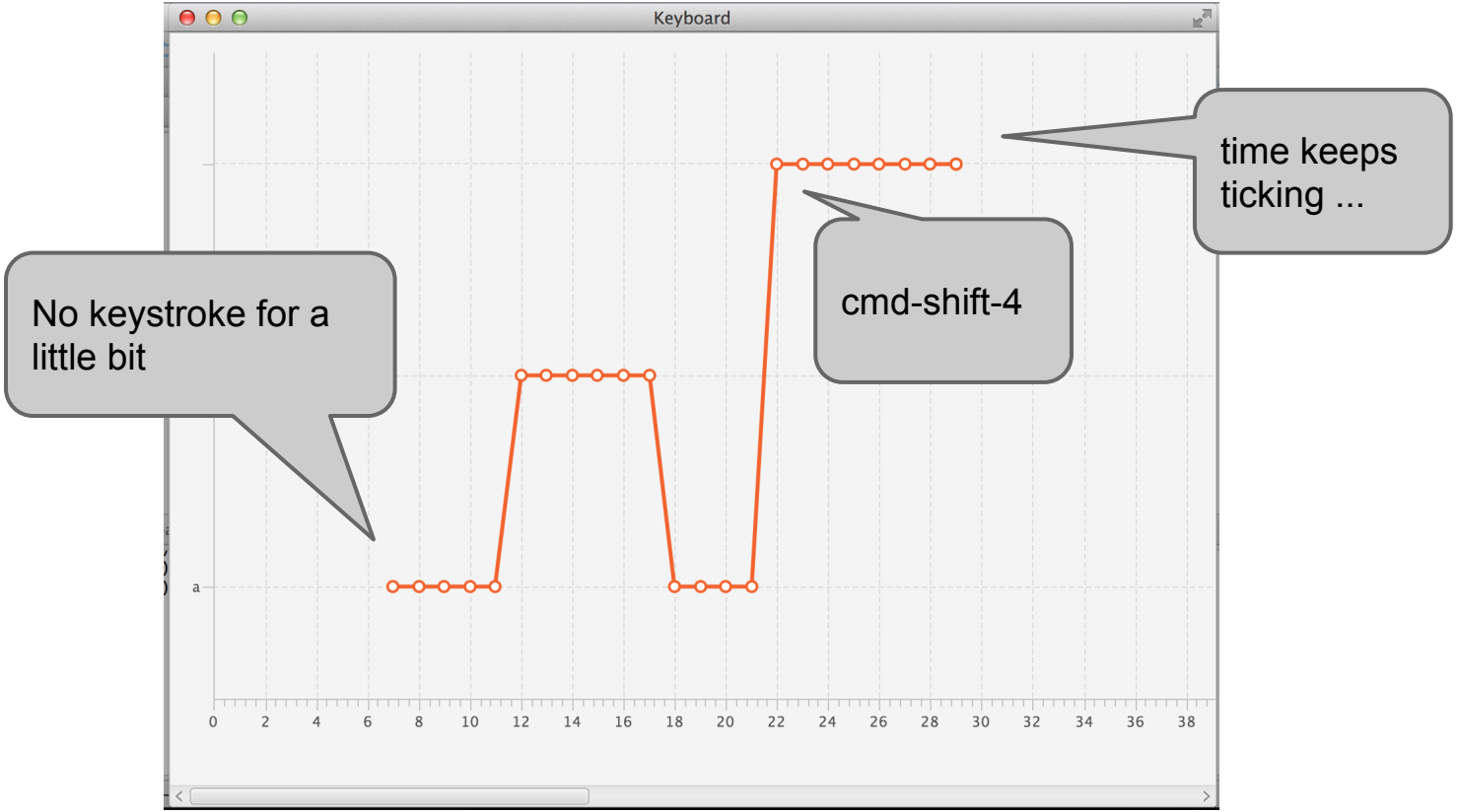
Every time we switch streams, ***we resubscribe to the clock***, starting at time t=0.

Hot Observable

```
val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 second)

val sample = clock.publish(_clock =>
                    keyboard.map(key => _clock.map(i => (i,key)))).switch
```

Every time we switch streams, ***we share the subscription to the clock***, continueing with time t = i.

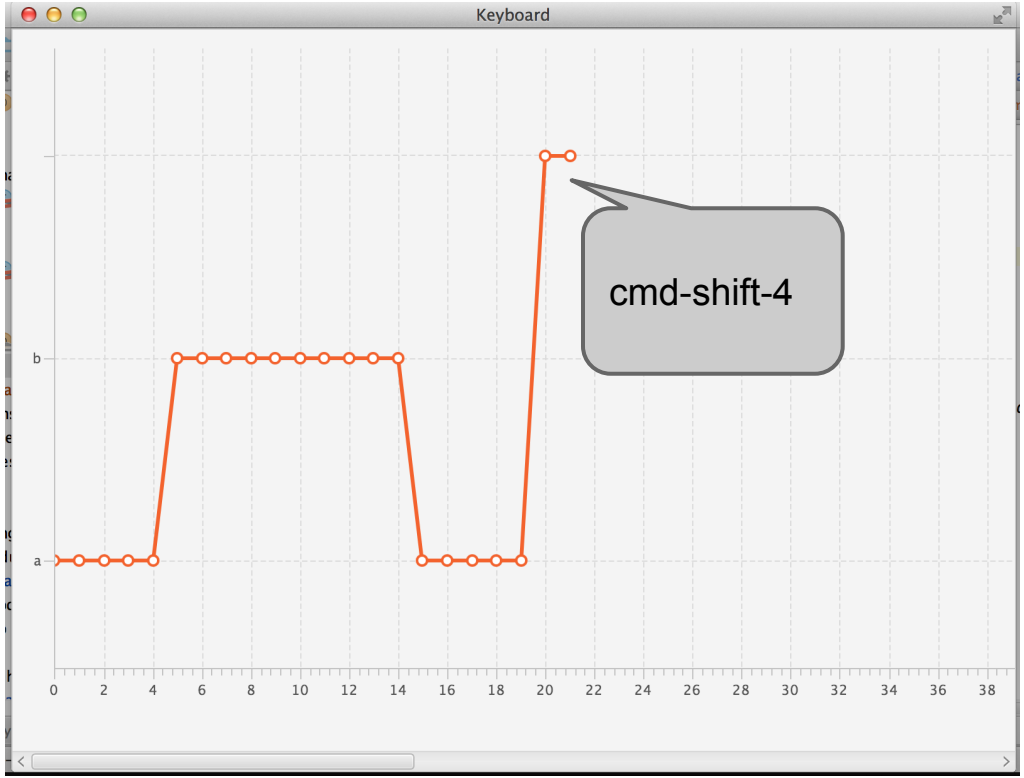
Sampling Every Tick



Virtual Time

```
val testScheduler = TestScheduler()

val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 second,
  testScheduler)

val sample = clock.publish(_clock =>
                    keyboard.map(key => _clock.map(i => (i,key)))).switch


keyboard.subscribe(key => {
  testScheduler.advanceTimeBy(0.5 second))
}
```

Say Cheese!


Keyboard
cmd-shift-4
b
a
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38

Keyboards Are Naturally Hot

```
val testScheduler = TestScheduler()

val clock: Observable[Long] = timer(initialDelay = 0 seconds, period = 1 second,
  testScheduler)

val sample = clock.publish(_clock =>
                    keyboard.map(key => _clock.map(i => (i,key)))).switch

keyboard.subscribe(key => {
  testScheduler.advanceTimeBy(0.5 second))
}
```

Ensuring an Observable is Hot

```
val keyboard: Observable[String] = keyPresses(scene).map(_.getText).publish

val sample = clock.publish(_clock =>
                    keyboard.map(key => _clock.map(i => (i,key)))).switch

keyboard.subscribe(key => {
  testScheduler.advanceTimeBy(0.5 second))
}

keyboard.connect
```

Ensuring an Observable is Hot

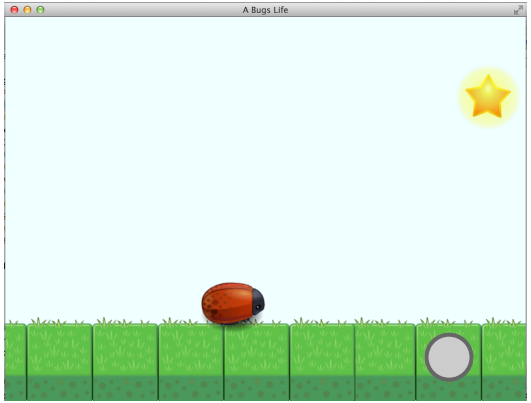
```
val keyboard: Observable[String] = keyPresses(scene).map(_.getText).share

val sample = clock.publish(_clock =>
                    keyboard.map(key => _clock.map(i => (i,key)))).switch

keyboard.subscribe(key => {
  testScheduler.advanceTimeBy(0.5 second))
}
```

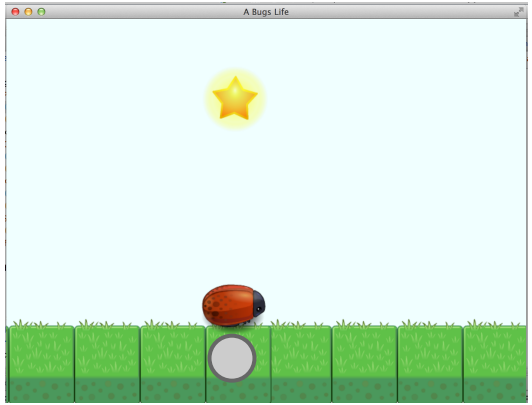
WARNING: (re) subscribes when refcount goes 0 -> 1

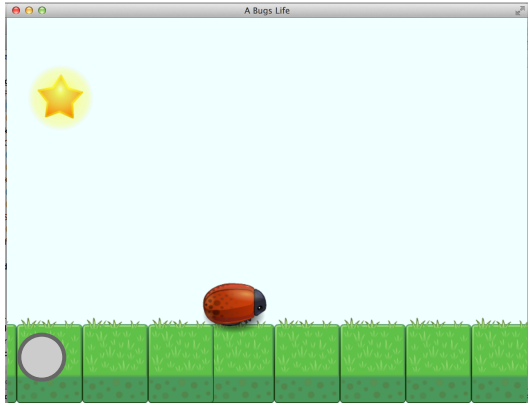
```
class MainJava {

 class HotIterable implements Iterable<T> {

   public Stream<T> stream() {
      return StreamSupport.stream(this.spliterator(), false);
   }

   final State perIterableState = new State();

   @Override
   public Iterator<T> iterator() {
      … perIterableState …
      final State perIteratorState = new State();
      return new Iterator<T>(){
         @Override public boolean hasNext() { … }
         @Override public Integer next() {
            … perIterableState … perIteratorState …
         }
      };
   }
}
```

State shared between all iterators ⇒ Hot

Everything Is Relative







The grass moves right to left,
the bug stays put.

Show me the code

```
val grass = new {
  val tile = new Image(s"$resourceDir/GrassBlock.png")
  val height = tile.getHeight
  val nrTiles = math.ceil(screenWidth/tile.getWidth).asInstanceOf[Int]+1

  val tiles = (0 to nrTiles-1).map(i =>
    new ImageView {
        setImage(tile)
        setTranslateX(i*getImage.getWidth)
       root.getChildren.add(this)
     }).toList
```

Create list of tiles using bulk operations on iterable

Where Is Dr Beckman When You Need Him Most

http://channel9.msdn.com/Series/Beckman-Meijer-Overdrive

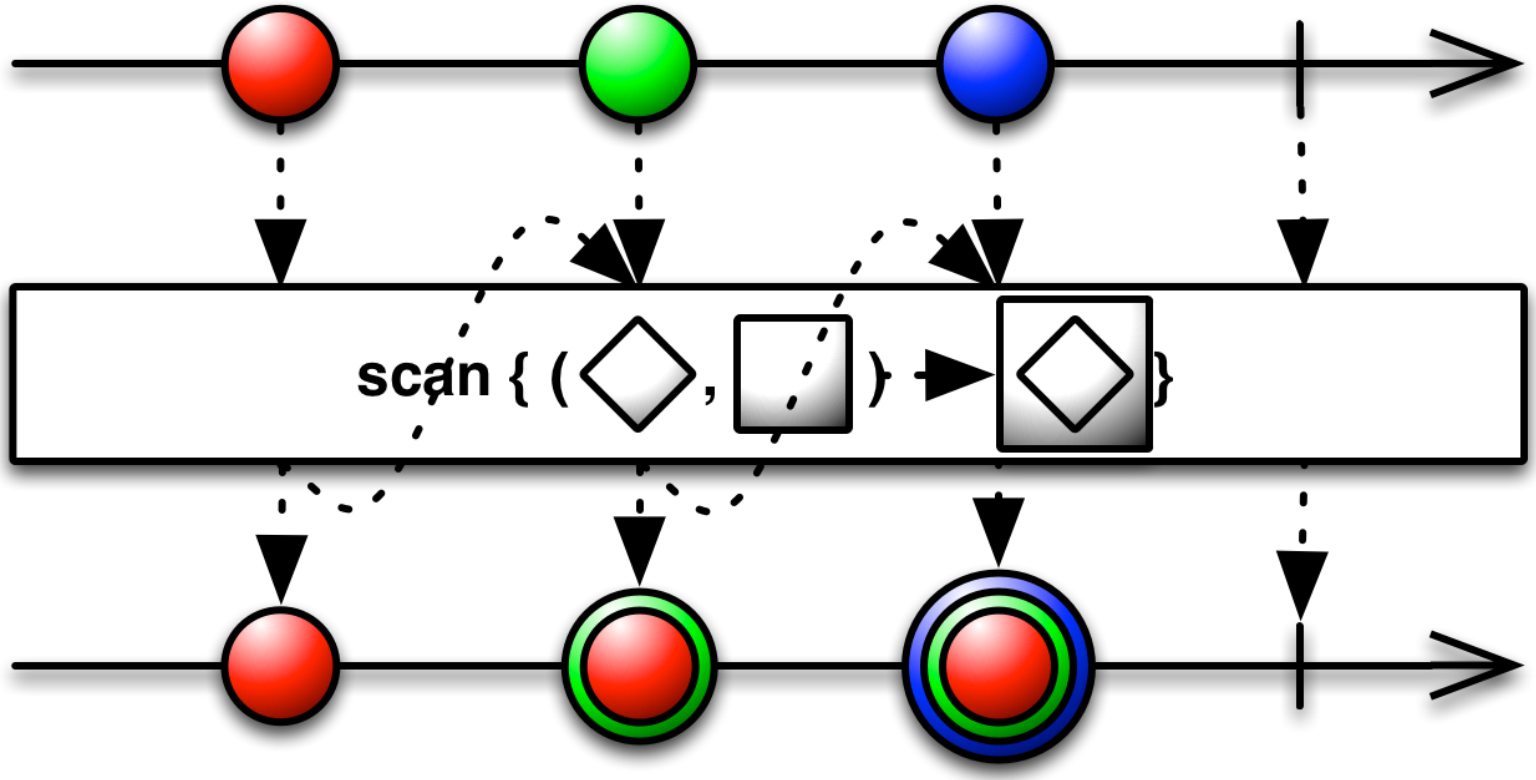
Show me the code

```
val grass = new {
  val v0 = 3
  clock.scan(v0)((vi,_)=>vi).subscribe(dX =>

    setTranslateX(
      if(getTranslateX <= -getImage.getWidth) {
        screenWidth-dX
      } else {
        getTranslateX-dX
      }))
}
```

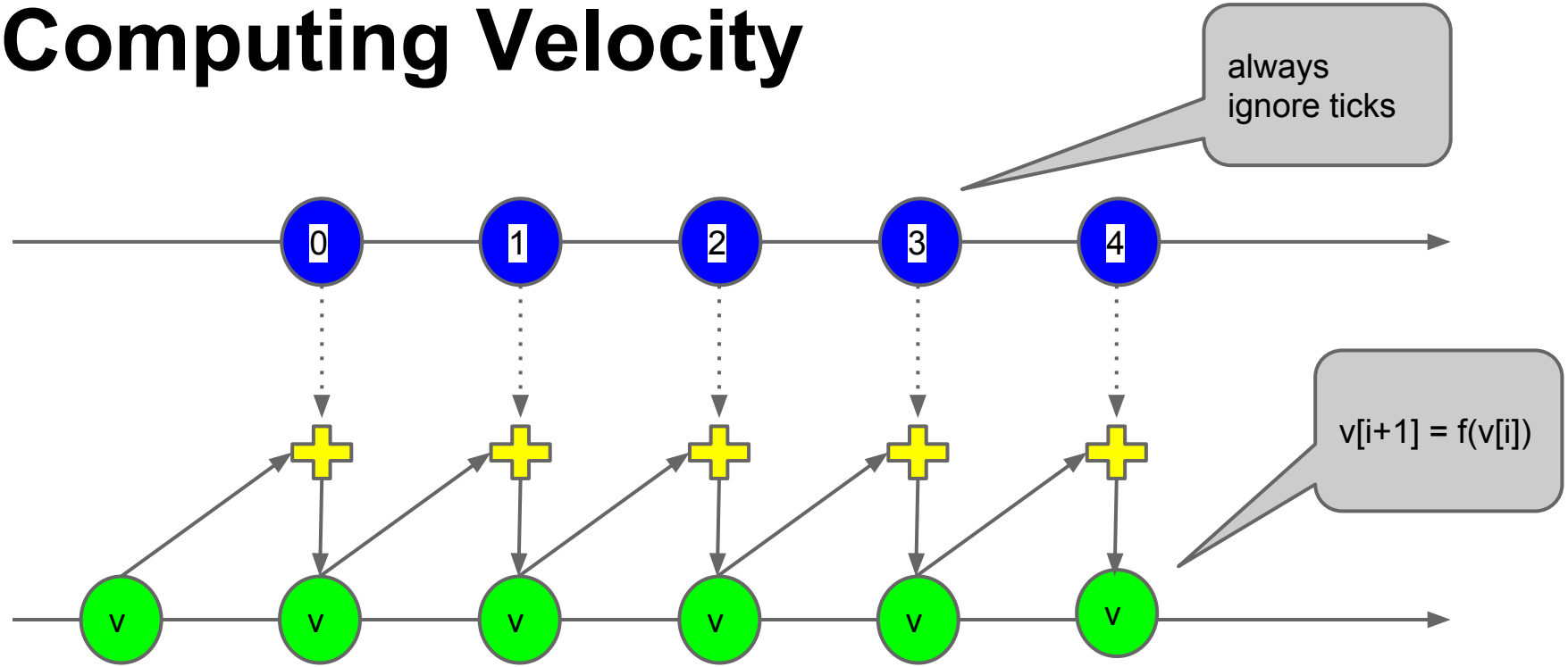
Poor man's physics:
$\Delta x = v*\Delta t$ where $\Delta t = 1$,
so $\Delta x = v$. We'll just take velocity to be pixels/tick and move things accordingly

wrap-around

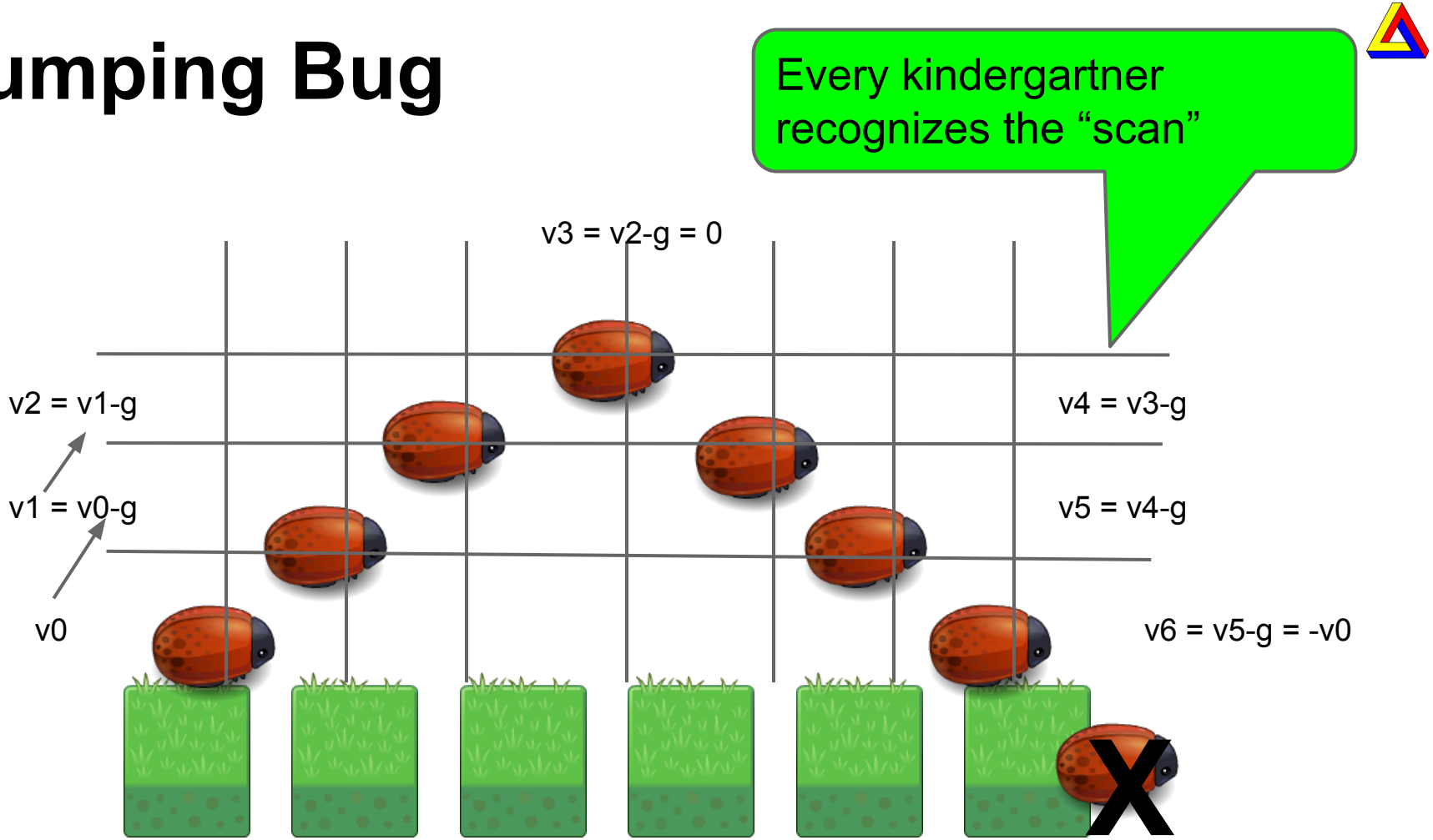
Scan: Accumulate with Trace



Computing Velocity



Jumping Bug


Every kindergartner recognizes the “scan”
v3 = v2-g = 0
v2 = v1-g
v4 = v3-g
v1 = v0-g
v5 = v4-g
v0
v6 = v5-g = -v0

Show Me The Code

```
jumps
  .flatMap(v0 =>
    clock
      .scan(v0)((vi,_)=>vi-gravity)
      .takeUntil(jumps)
  )
  .subscribe(dY => … update position …)
```

When jump happens

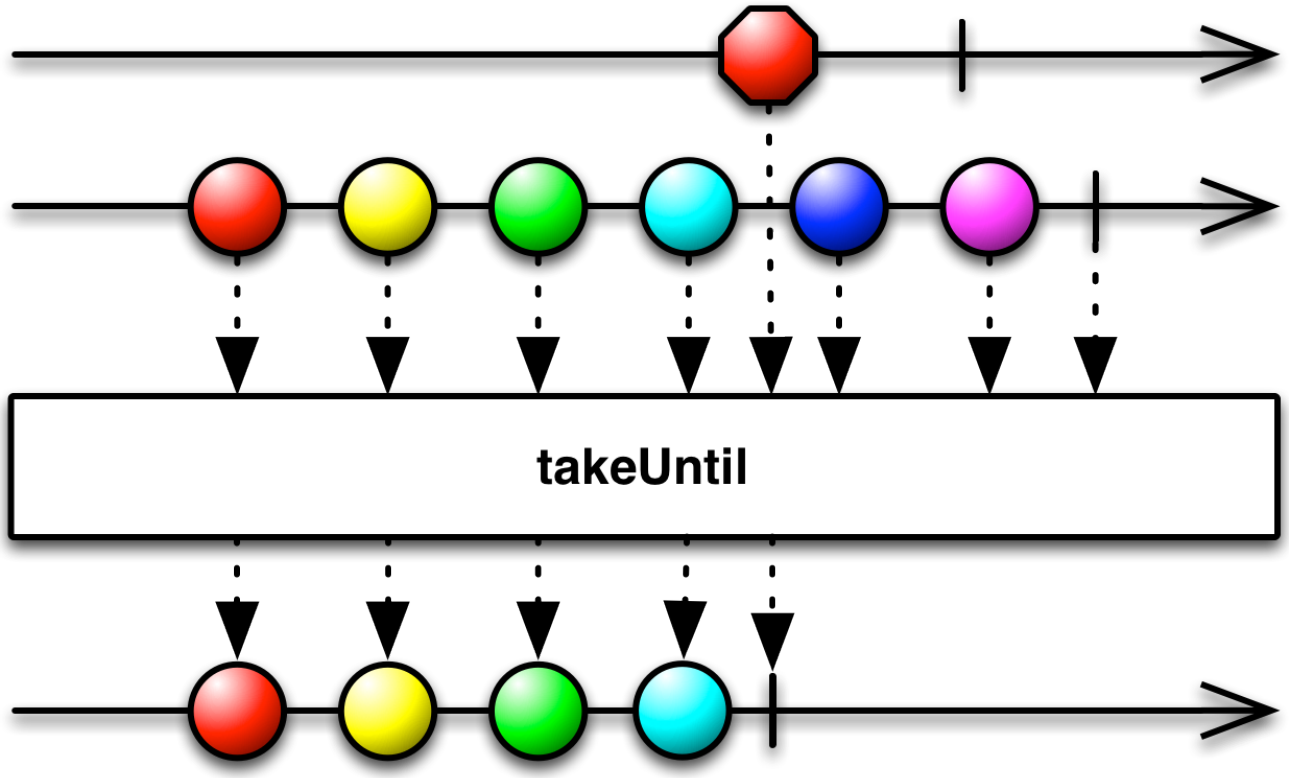
Initital jump velocity

Decrease with “gravity” each tick

Until next jump

Prevent falling through floor by ignoring changes when below sea level

Turn One Stream Off By Another



Show Me The Code

```
val jumpSpeed = 8
    spaceBars(scene)
      .filter(_ => bug.getTranslateY >= bug.homeY)
      .doOnEach(_ => ... play sound ...)
      .subscribe(_ => bug.jumps.onNext(jumpSpeed))
```

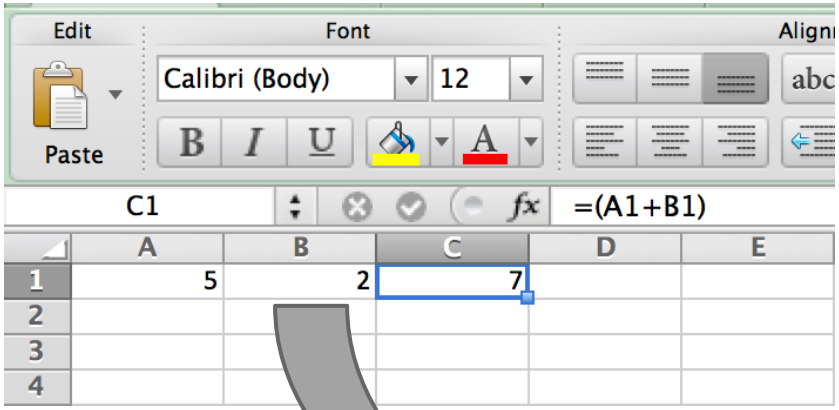
Whenever spacebar is hit

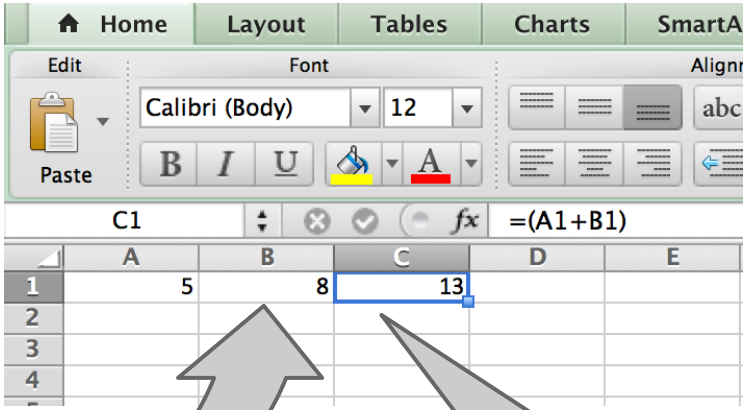
But only when the bug is on the ground

Play some music

Make bug jump

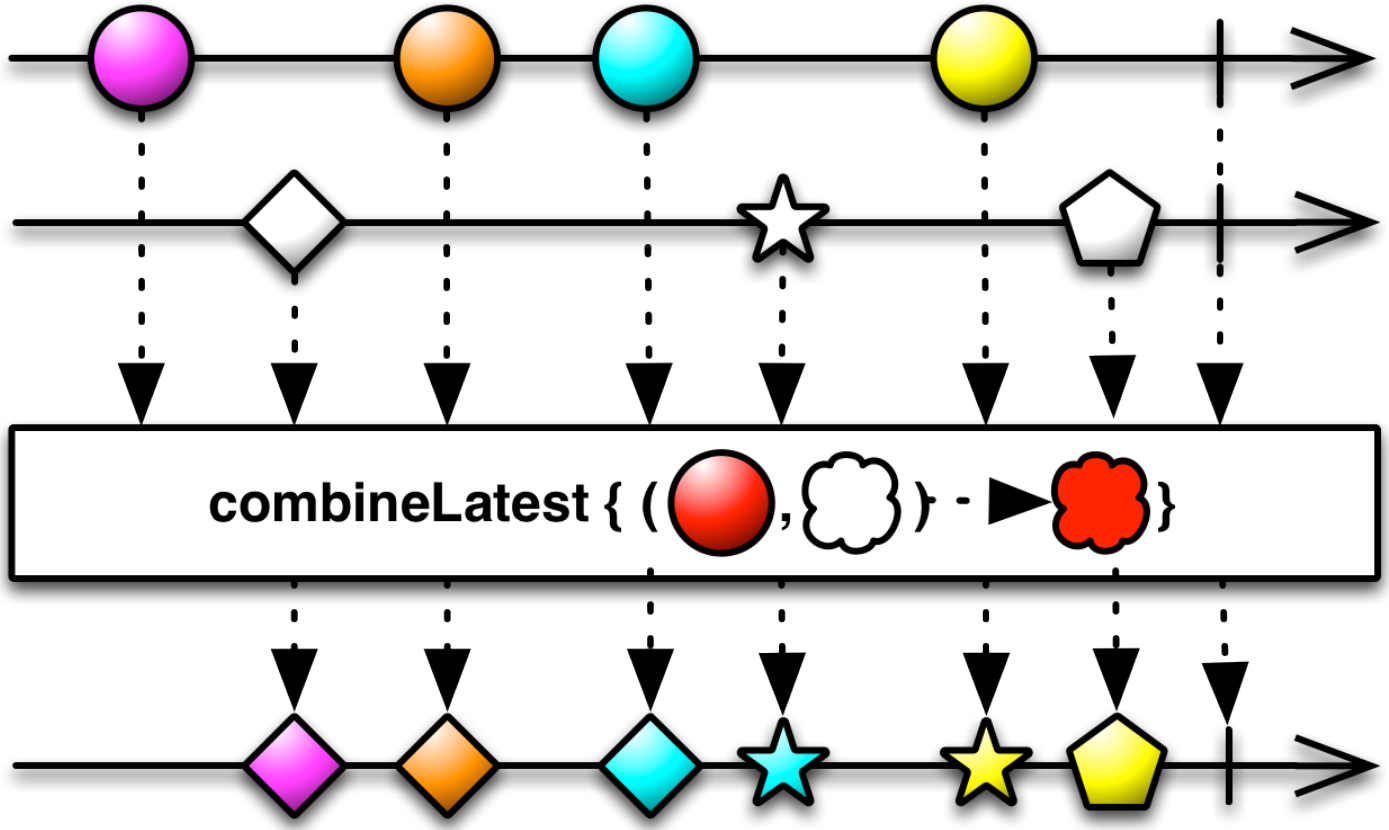
SpreadSheet == Mother of All Reactive Programming





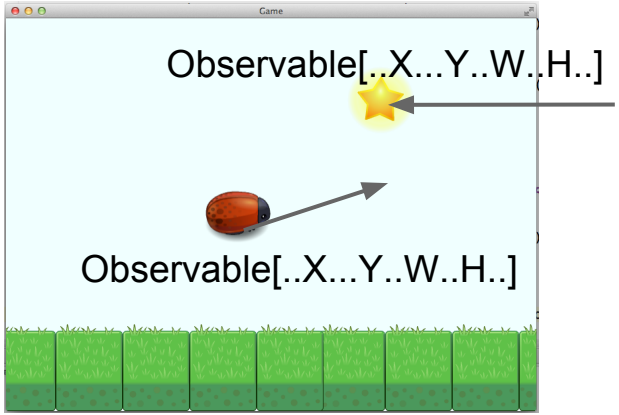
Result cell automatically updated whenever any input cell changes

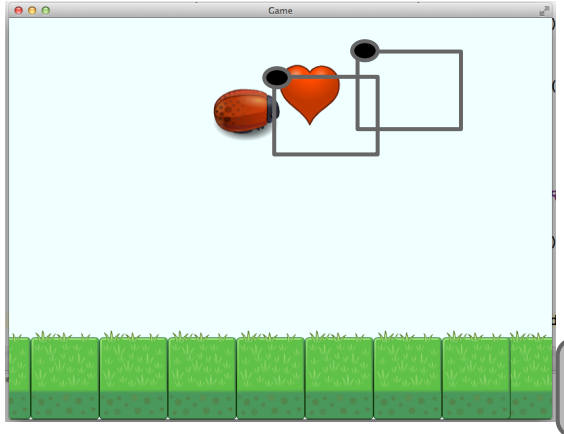
Steal A Trick From Spreadsheets

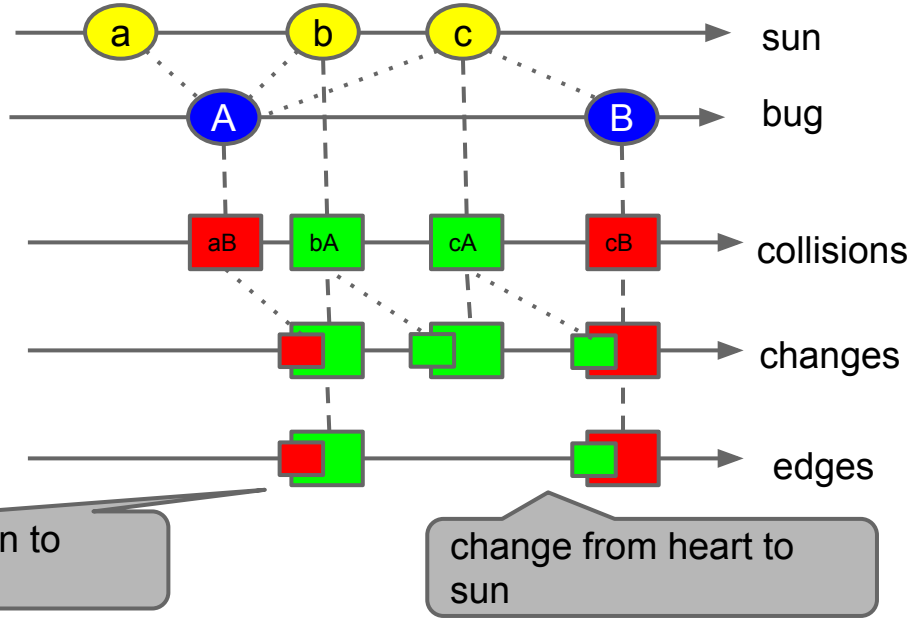


Hit Detection

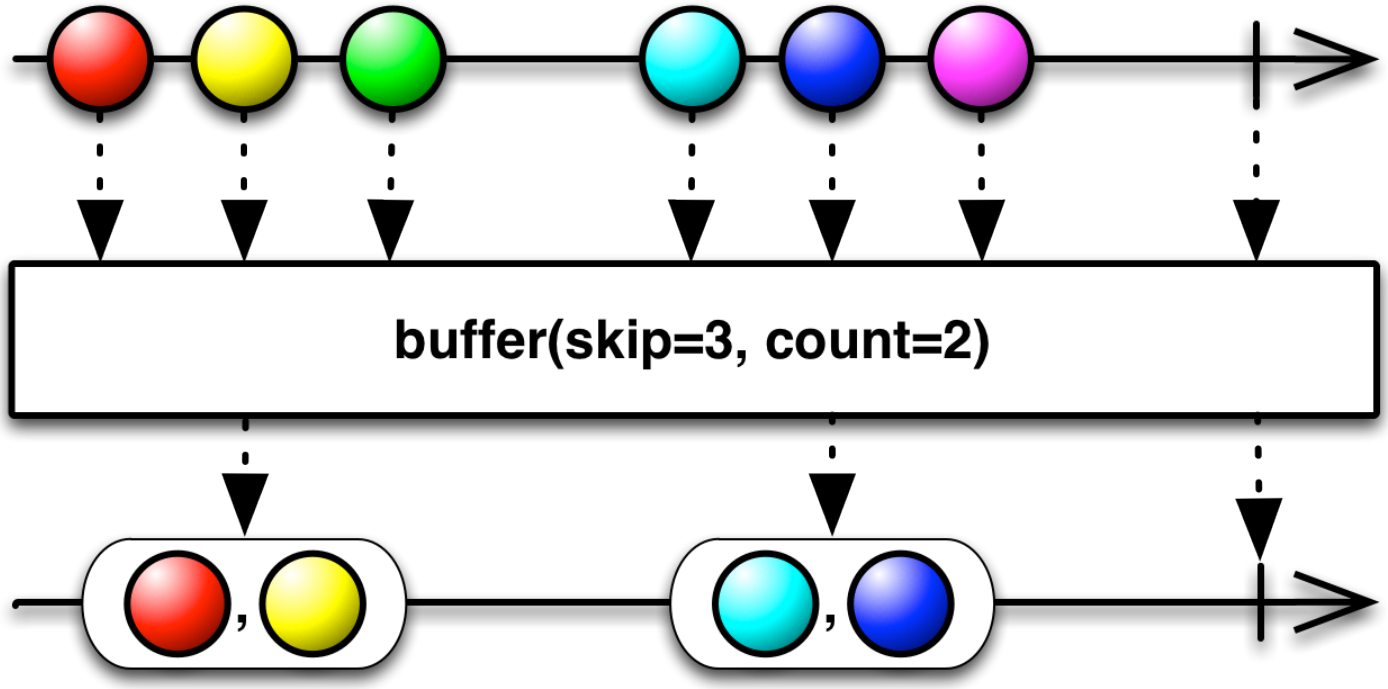
```
bugPosition.combineLatest
 (sunPosition, collides)
    .slidingBuffer(2,1)
    .filter(hits => hits(0) != hits(1))
```







Tumbling Buffers

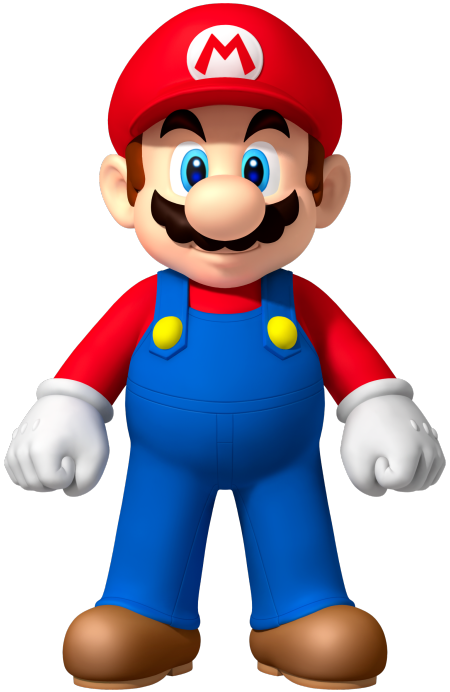


That’s Really All There Is

In the end we are all just plumbers

```
import javafx.geometry._
import javafx.scene.canvas.Canvas
import javafx.scene.paint.Color
import javafx.scene._
import javafx.scene.image._
import javafx.scene.layout.StackPane
import javafx.stage._
import javafx.application._
import games.PlatformScheduler
import rx.lang.scala.schedulers._
import rx.lang.scala._
import scala.language.postfixOps
import scala.concurrent.duration._
import rx.lang.scala.javaFx.utils
```

Import list nearly long than the code



http://www.mariowiki.com/Image:MarioNSMBWii.PNG

More Info

https://github.com/Netflix/RxJava

https://github.com/Reactive-Extensions/RxJS

https://github.com/Reactive-Extensions/RxCpp

https://github.com/Reactive-Extensions/Rx.NET

https://github.com/ReactiveCocoa/ReactiveCocoa

https://www.dartlang.org/docs/tutorials/streams/

http://bodil.org/prez/

And Don’t Forget To Sign Up For

edX HOW IT WORKS COURSES SCHOOLS & PARTNERS REGISTER NOW log in

Introduction to Functional Programming

The aim of this course is to teach the foundations of functional programming and how to apply them in the real

School:	DelftX
Course Code:	FP101x
Classes Start:	October 2014
Course Length:	8 weeks
Estimated effort:	4 - 6 hours/week

https://www.edx.org/course/delftx/delftx-fp101x-introduction-functional-2126#.U8jMmI2Sylo