CURE WHAT AILS YOU WITH THIS

RX

an intro to the reactive extensions

MATT SOUCOUP

» Code Mill Technologies

» Xamarin MVP

» Blog: codemilltech.com

» Email: msoucoup@codemilltech.com

» Twitter: @codemillmatt

» https://github.com/codemillmatt/Rx-Cure

WE LIVE IN A PUSH BASED WORLD

» Tweets

» Taps & Touches

» Async I/O - DBs, REST services

» Web Sockets

» Device Services - GPS, BLE, etc

WE NEED A SOLID WAY TO HANDLE AN ASYNC PUSH BASED WORLD!

HOW ABOUT ... EVENTS?

HOW ABOUT ... EVENTS?

THEY HAVE ISSUES...

HOW ABOUT ... EVENTS?

THEY HAVE ISSUES...

» Difficult to implement / understand

» Stateful code

» Not async

» Memory leaks

HOW ABOUT ... EVENTS?

THEY HAVE ISSUES...

```
var wedgesOfCheeseAte = 0;

eatCheese.Click += async(s, e) => {
    if (wedgesOfCheeseAte < 3) {
        await devourDeliciousCheese();

        wedgesOfCheeseAte += 1;
    }

    // Possible memory leak!
}
```

ISN'T THIS BETTER?

```
eatCheese
    .Take(3)
    .Subscribe(async (_) => {
      await devourDeliciousCheese();
    });
```

RX SAVES THE DAY!

AGENDA

» What is Reactive Programming & Rx

» Observables & Data Streams

» Thinking Reactive

WHAT IS REACTIVE PROGRAMMING?

» Takes Observer Pattern - makes it better

» Push based

» Turns events into data streams

» A means to react to changes over time

WHAT ARE THE REACTIVE EXTENSIONS?

“Rx creates a better event”
Paul Betts

WHAT ARE THE REACTIVE EXTENSIONS?

A better question is...

WHAT ARE THE REACTIVE EXTENSIONS?

A better question is...

WHAT ARE EVENTS?

» A collection of things that happen over time

» A stream of data ...

» Much like a list

WHAT ARE THE REACTIVE EXTENSIONS?

- Provides a toolset to create event/data streams
  - May or may not end
  - Signal if they end
  - May or may not throw exception

WHAT ARE THE REACTIVE EXTENSIONS?

- Provides a toolset to manipulate those streams
  - Transform (select)
  - Filter (where)
  - Aggregate (max, min, etc)
  - Combine (concat, zip)
  - Time-based (buffer, window)

RX TURNS EVENTS INTO AN ENUMERABLE

SOME BENEFITS OF REACTIVE EXTENSIONS?

Declaratively layout code

```
eatCheese
    .Take(3)
.Subscribe(async (_) => {
    await devourDeliciousCheese();
});
```

SOME BENEFITS OF REACTIVE EXTENSIONS?

Express intent without implementation details

```
eatCheese
    .Take(3)
    .Subscribe(async (_) => {
      await devourDeliciousCheese();
    });
```

» Focus on business domain

SOME BENEFITS OF REACTIVE EXTENSIONS?

Implementations for many languages

* .Net
* Java
* Javascript
* Python
* Ruby
* C++
* Swift ...

ENTER THE OBSERVABLE

ENTER THE OBSERVABLE

...BUT WHAT EXACTLY IS IT??

AN OBSERVABLE IS A STREAM OF DATA

» Async Cousin of Enumerable

» Instead of pulling data - push data

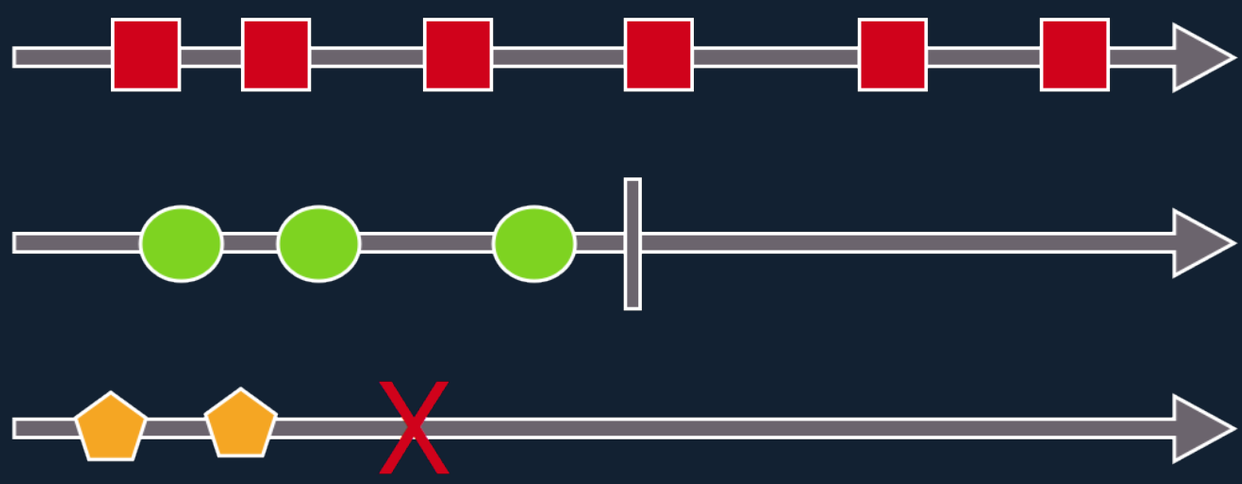
» Asynchronous, non-blocking

» Subscribe to perform some action

  » Similar to Enumerable.ForEach

AN OBSERVABLE DOES THINGS THROUGH TIME

1. Emit a value, and continue
2. Complete
3. Error



ANYTHING CAN BE AN OBSERVABLE

» Enumerable

» Tasks (async/callbacks)

» Events

OBSERVABLES ARE BETTER THAN EVENTS

» Stream through time

» Notify when complete, exception

» LINQ manipulations!

» 1st class citizen

  » Chain operators

  » Pass as variables

» Subscribe to handle

THE RX MINDSET

RX VS IMPERATIVE

THE IMPERATIVE WAY

» Listen for changes

» Implement details of handling

» Call a method

» Handle results immediately / inline

  » even with async and await or callbacks

RX VS IMPERATIVE

THE IMPERATIVE WAY

```
var keyStrokeTimer = new Timer (500);
var timeElapsedSinceChanged = true;

keyStrokeTimer.Start ();

keyStrokeTimer.Elapsed += (sender, e) => {
    timeElapsedSinceChanged = true;
};

var searchText = "";
searchField.EditingChanged += async (sender, e) => {
    keyStrokeTimer.Stop ();
    if (timeElapsedSinceChanged) {
        // Probably should do some locking
        timeElapsedSinceChanged = false;
        keyStrokeTimer.Stop ();

        if (!string.IsNullOrEmpty (searchField.Text)) {
            if (!searchText.Equals (searchField.Text)) {
                searchText = searchField.Text;

                var results = await SearchCheeses (searchText);

                foreach (var cheeseName in results) {
                    Console.WriteLine (cheeseName);
                }
            }
        }
    }

    keyStrokeTimer.Start();
};
```

RX VS IMPERATIVE
THE RX WAY

» Define what will happen asynchronously

  » Observable

» Define any manipulations

  » Operators

» Subscribe to its results

  » Observer

RX VS IMPERATIVE
THE RX WAY

```
var editing = searchField.Events ().EditingChanged;

var searchSteam = editing
    .Select (_ => searchField.Text)
    .Where (t => !string.IsNullOrEmpty (t))
    .DistinctUntilChanged ()
    .Throttle (TimeSpan.FromSeconds (0.5))
    .SelectMany (t =>
        SearchCheeses (t));

searchSteam.Subscribe (
    r =>
        r.ForEach(cheeseName =>
            Console.WriteLine($"Cheese name: {cheeseName}"))
);
```

DEMO TIME!

KEYUP -> SEARCH

THINKING IN RX

LET GO...

» Let go of imperative

» Let go of state

» Let go of pulling

THINKING IN RX

INSTEAD

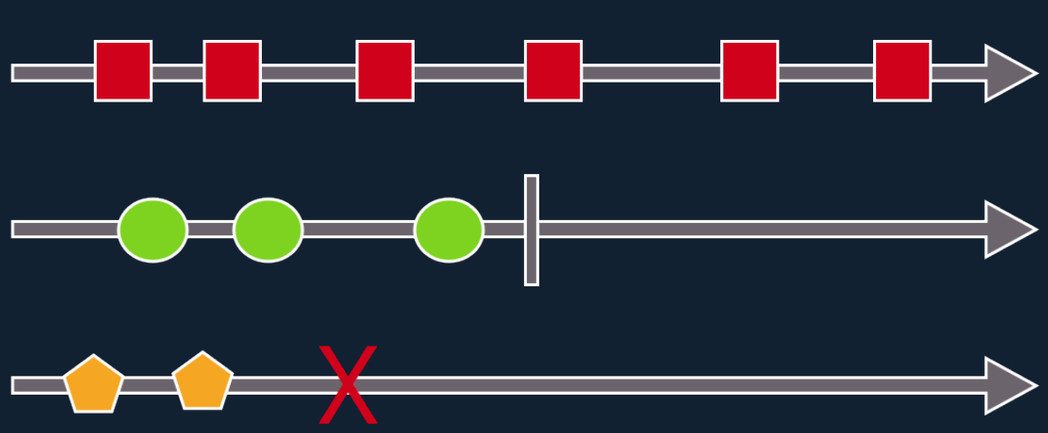
» Think of events through time

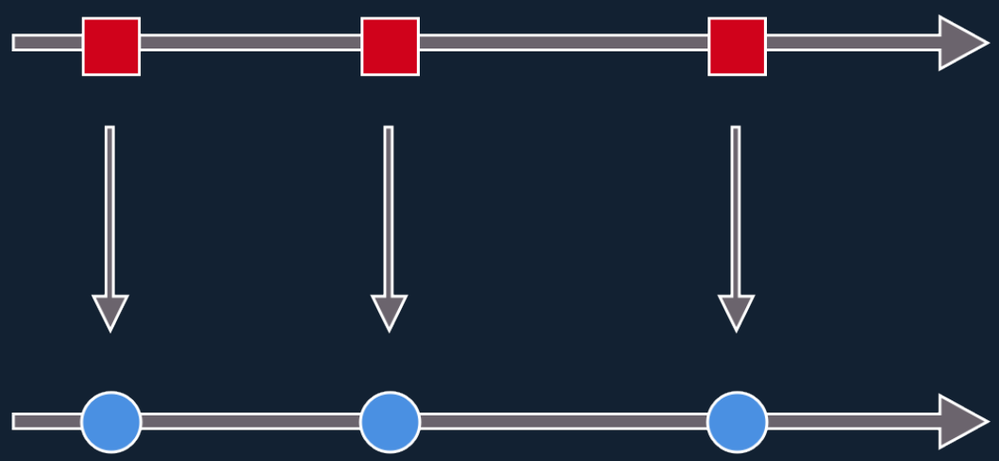
» Think in functional operators

MARBLE DIAGRAMS

» Timeline of events

» Timeline of event operations





FUNCTIONAL OPERATORS
KEY TO UNDERSTANDING RX

» Transform (select)

» Filter (where)

» Aggregate (max, min, etc)

» Combine (merge, zip)

» Time-based (buffer, window)

OPERATING ON AN OBSERVABLE

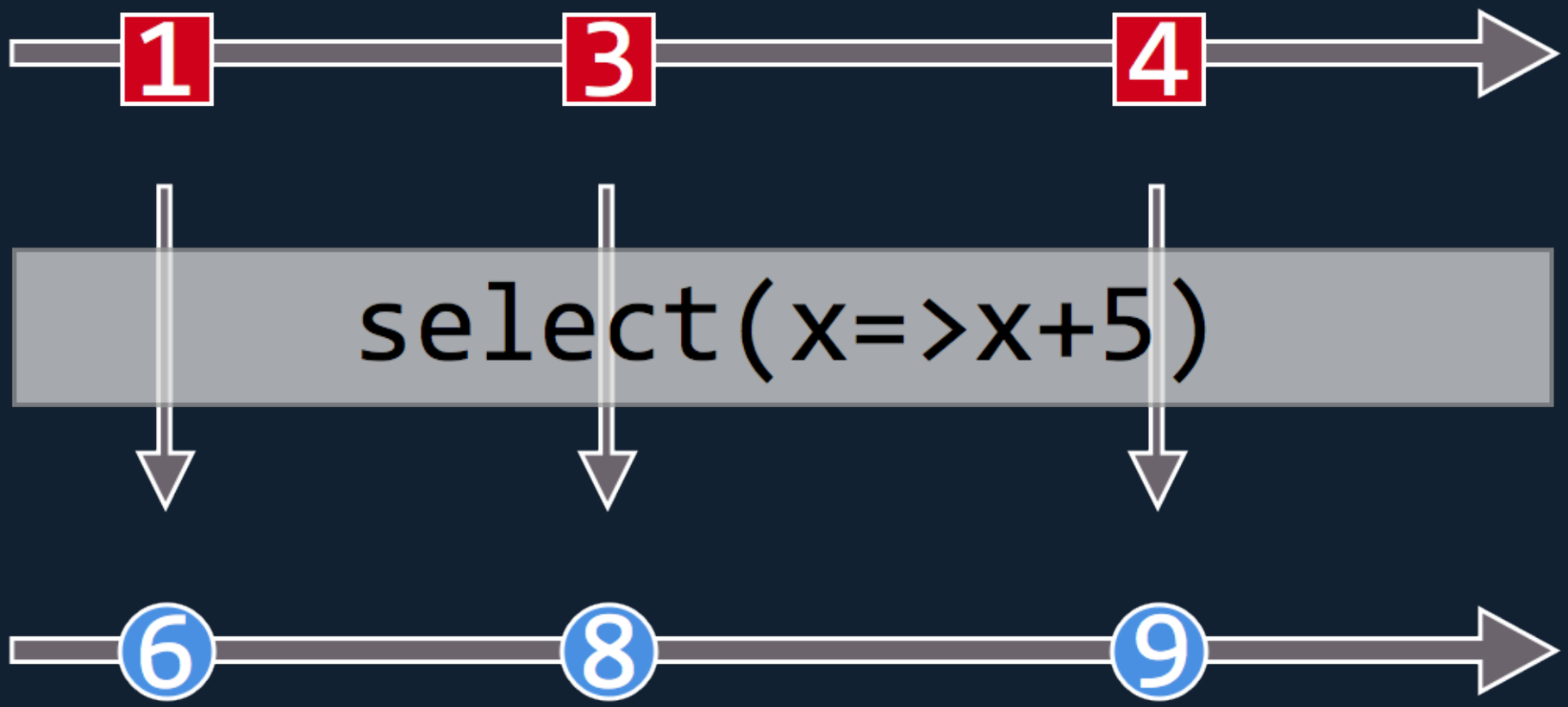
» Operators take Observable

» Operators return Observable (sometimes)

» Chain 'em together!

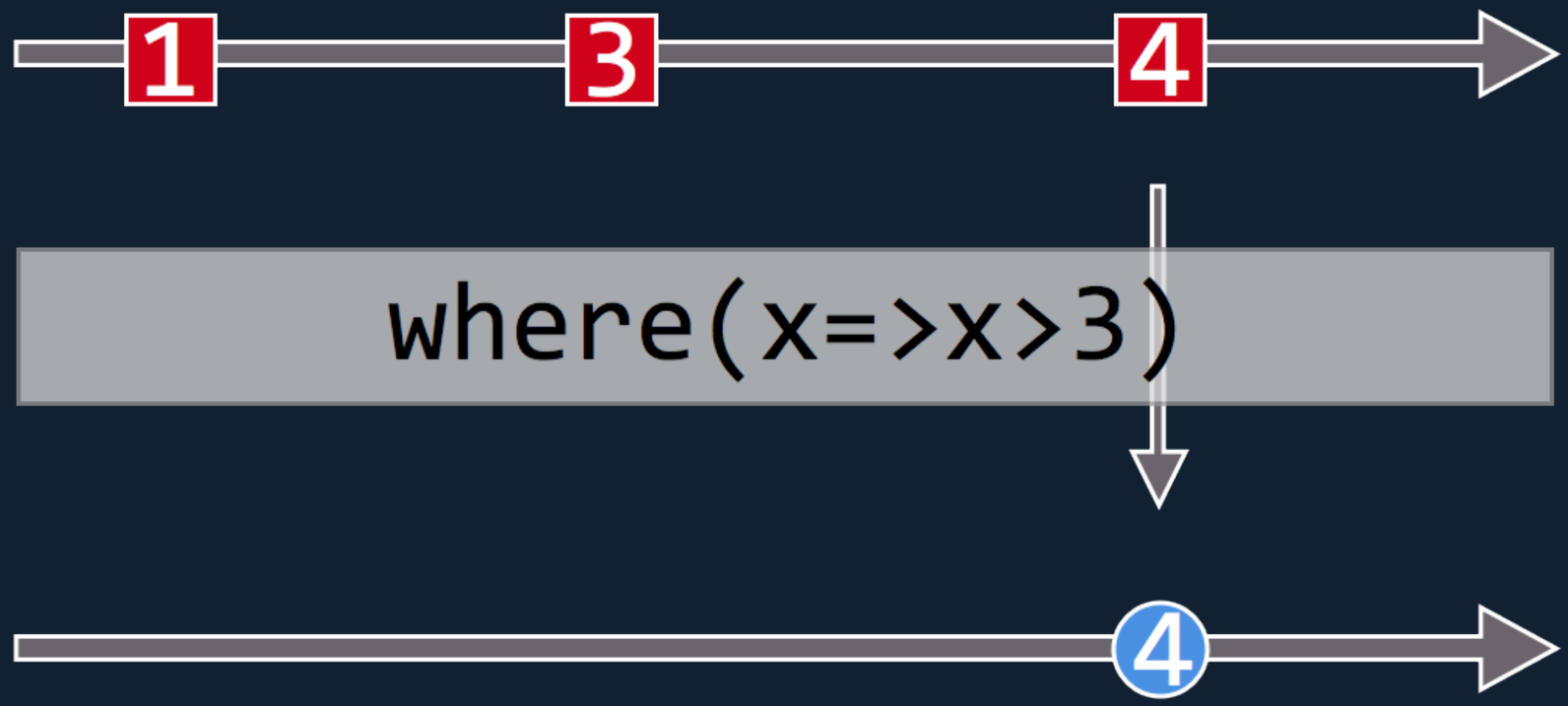
» Operate in turn - one after the other

TRANSFORM DIAGRAM



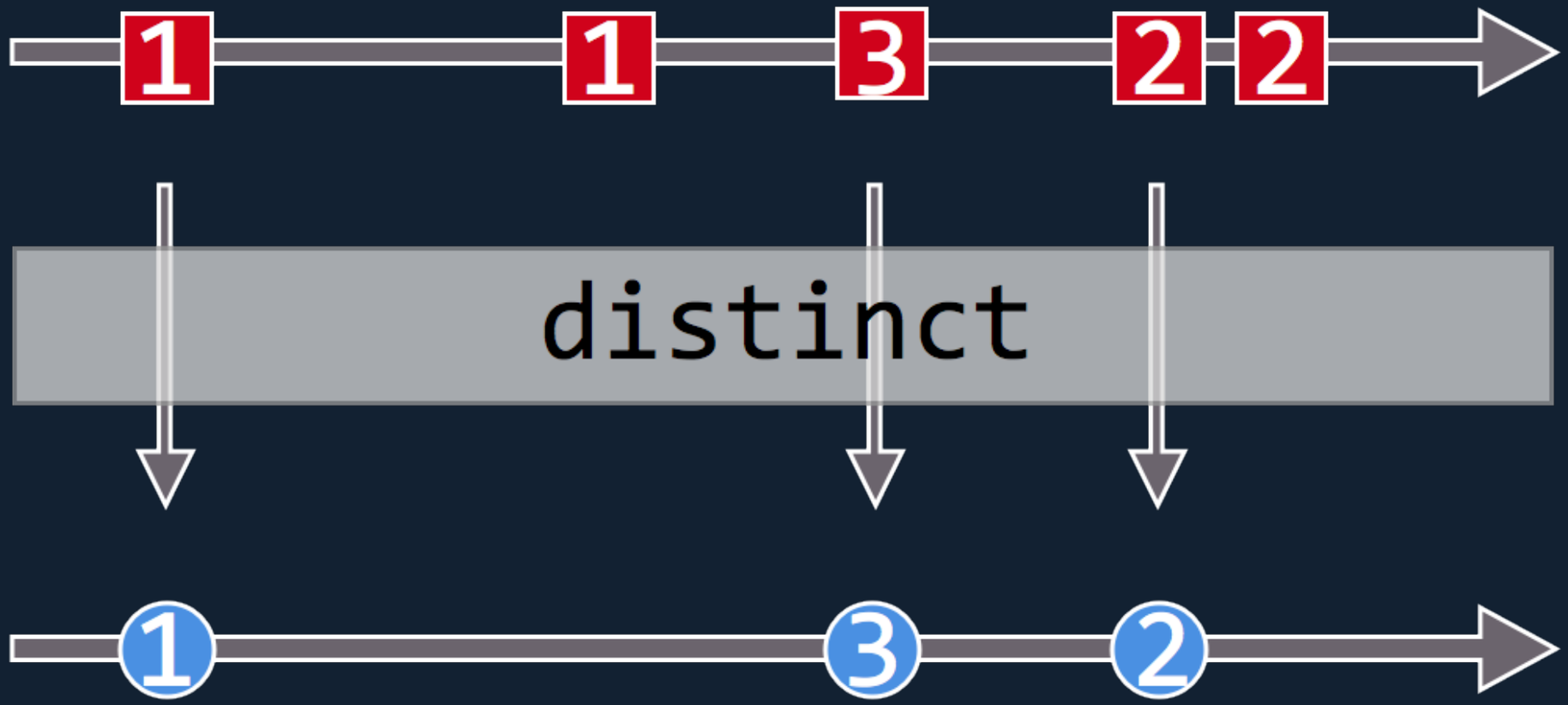
* Transform an item by applying a function to it
* aka: select, map, cast

FILTERING DIAGRAM



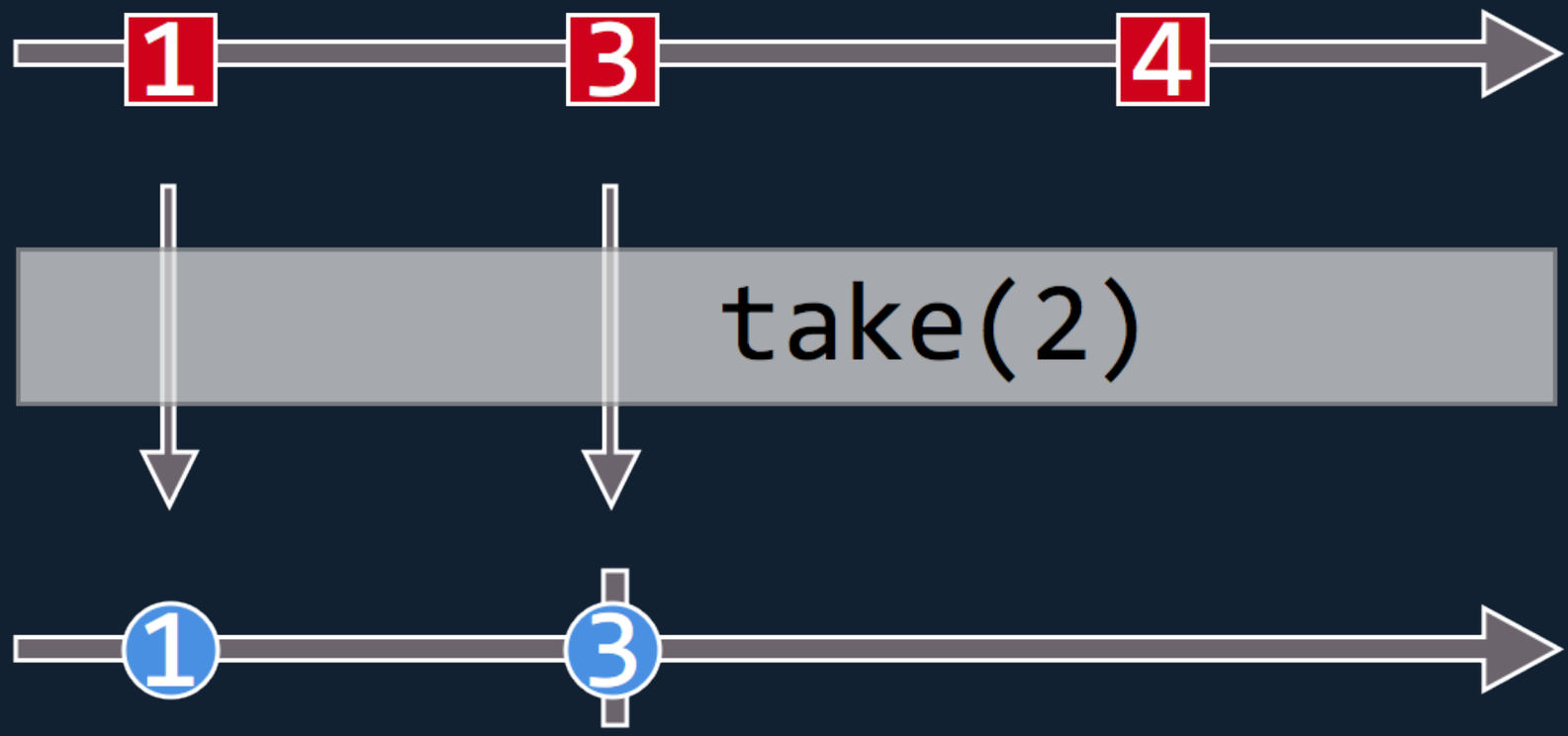
* Filter the results based on a function
* Only emit results that pass
* aka: where, filter

FILTERING DIAGRAM



* Distinct
* Only emit items which have not been seen already

FILTERING DIAGRAM



* Take
* Emit 'x' amount of items
* Complete after 'x'

DEMO TIME!

MOVE IMAGE AROUND SCREEN

DEMO TIME!

MOVE IMAGE AROUND SCREEN

* Constrain image to left half of screen
* Constrain image to top half of screen
* Make event args easier to deal with

DEMO TIME!


```
select(loc=>new{X=loc.X, Y=loc.Y})
```


```
where(l=> l.X < screenWidth / 2)
```


```
where(l=> l.Y < screenHeight / 2)
```

DEMO TIME!

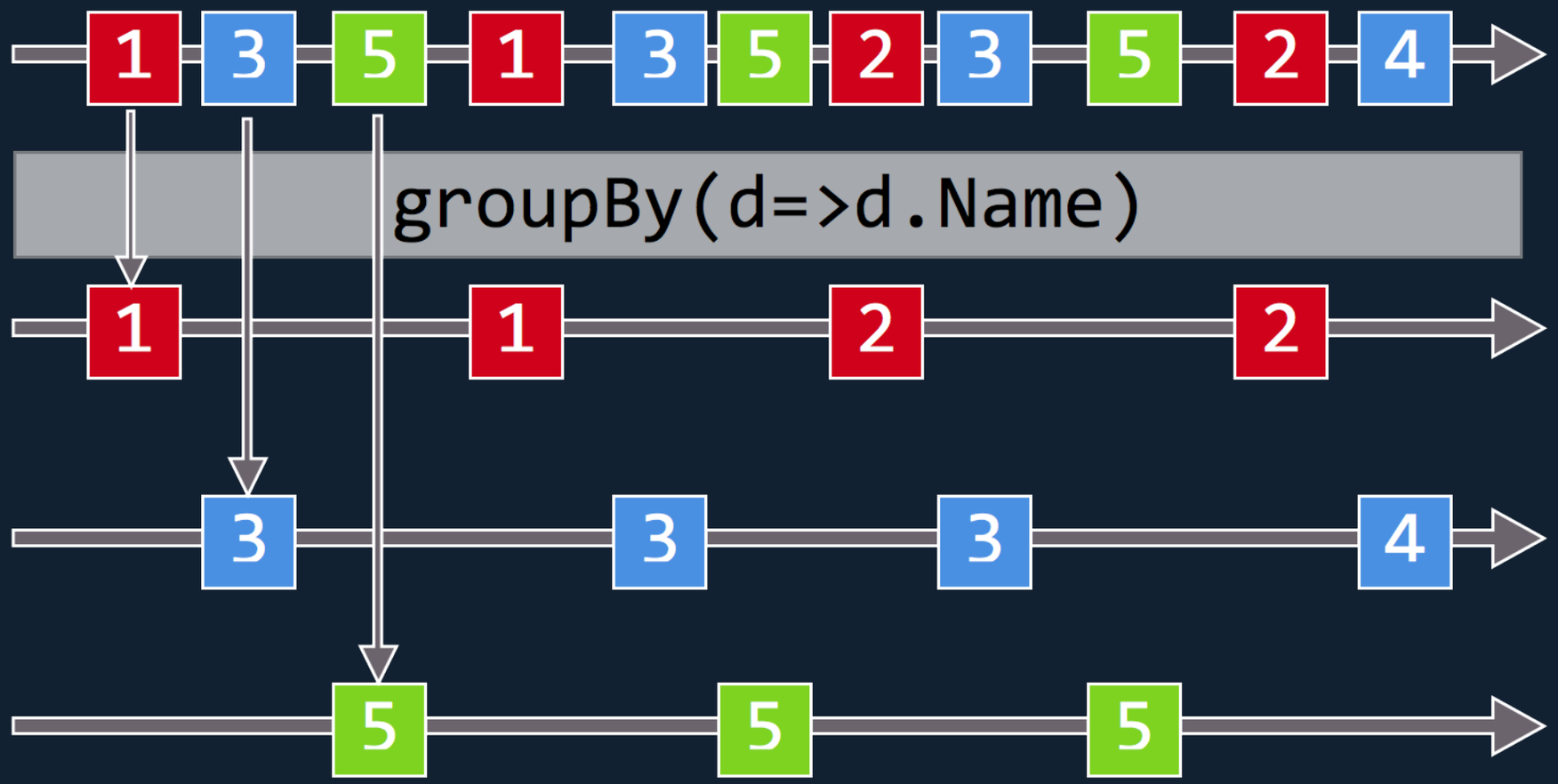
HANDLE BLE ADVERTISEMENTS

DEMO TIME!

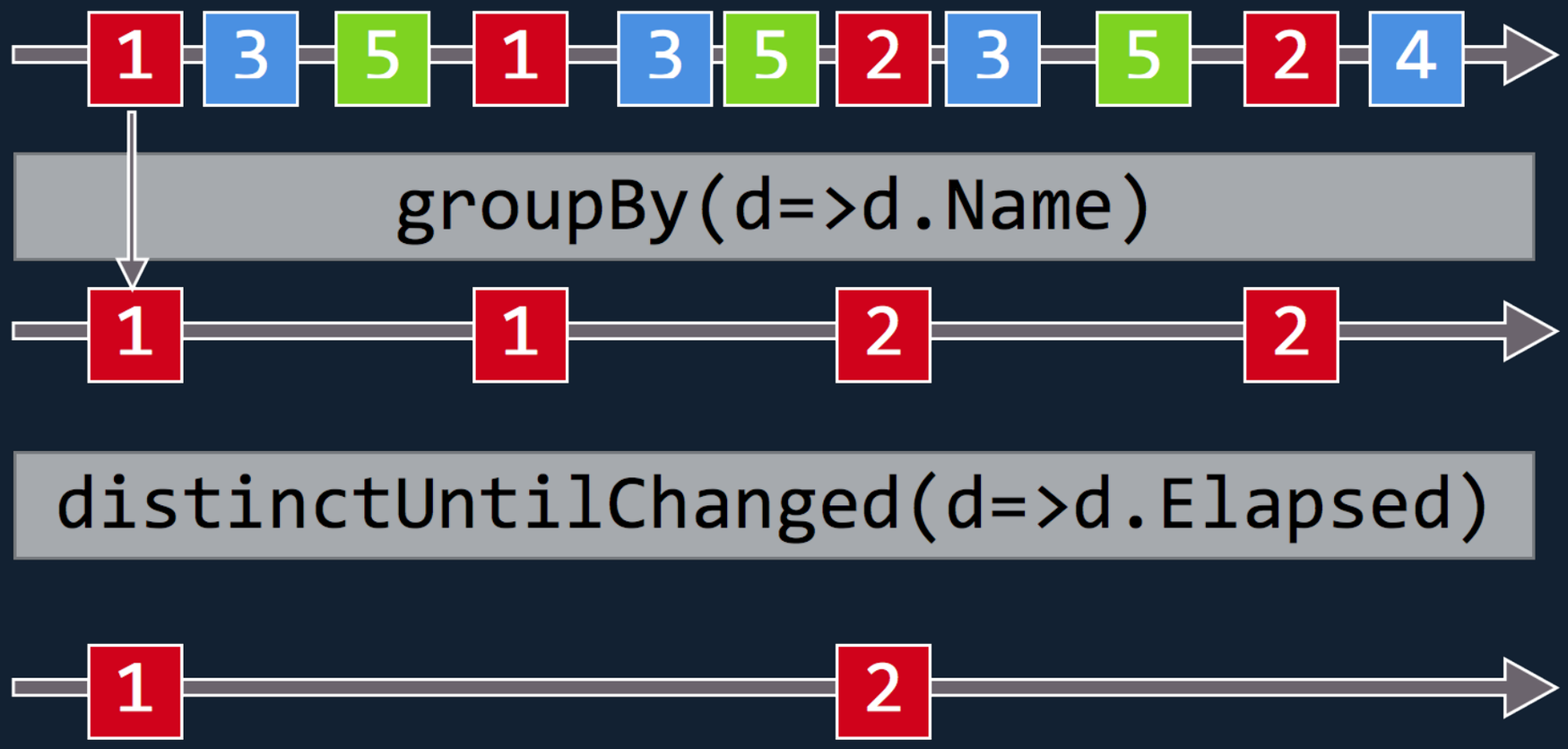
HANDLE BLE ADVERTISEMENTS

* Many ads come in
* Separate by device name
* Only record distinct readings

DEMO TIME!



DEMO TIME!


1 3 5 1 3 5 2 3 5 2 4
groupBy(d=>d.Name)
1 1 2 2
distinctUntilChanged(d=>d.Elapsed)
1 2

TIME BASED OPERATORS

» Events through time

» Operators to deal with those

  » Throttle

  » Window

  » Buffer

  » Delay

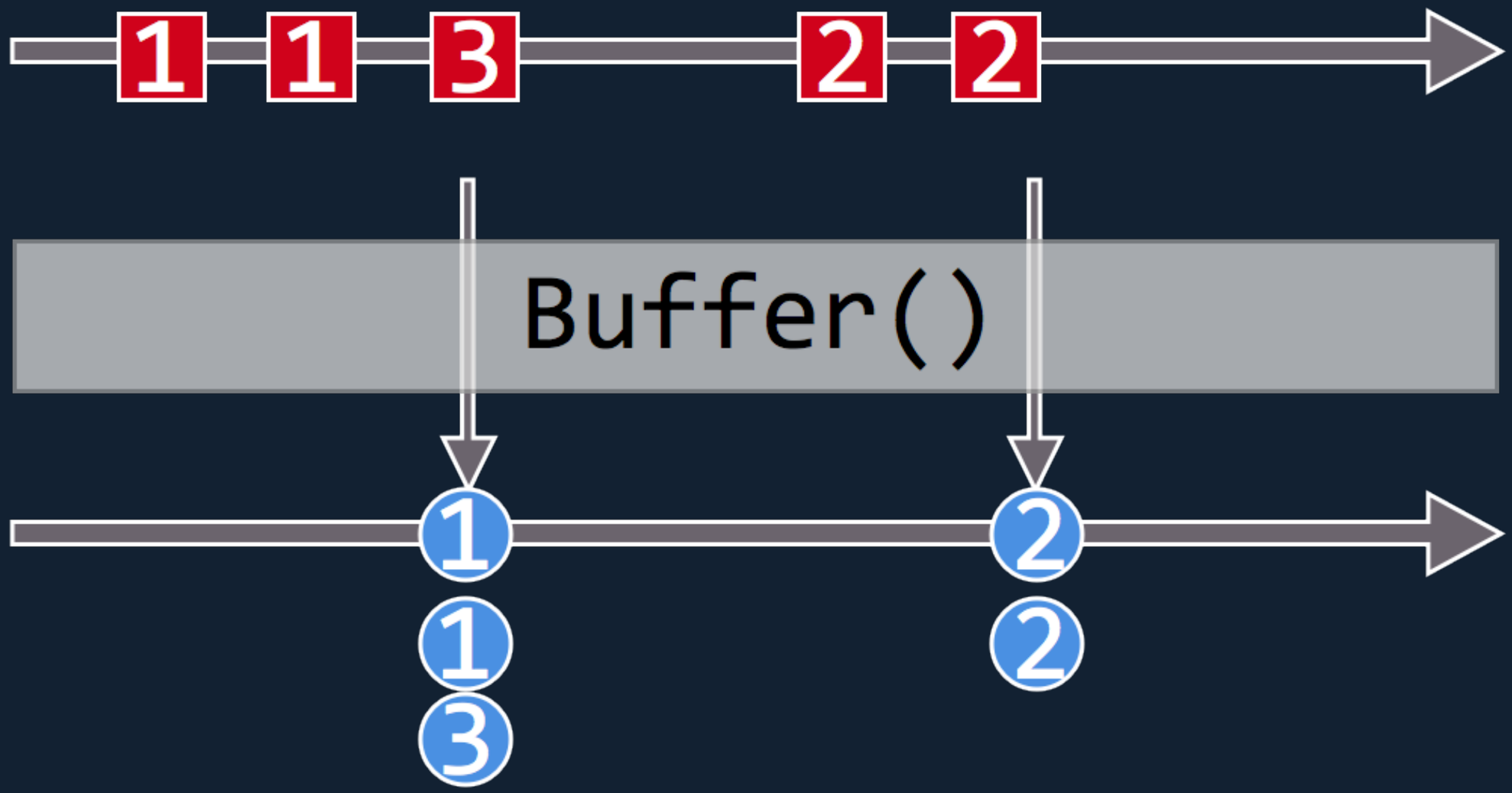
  » TimeInterval

TIME BASED DIAGRAM

Throttle()

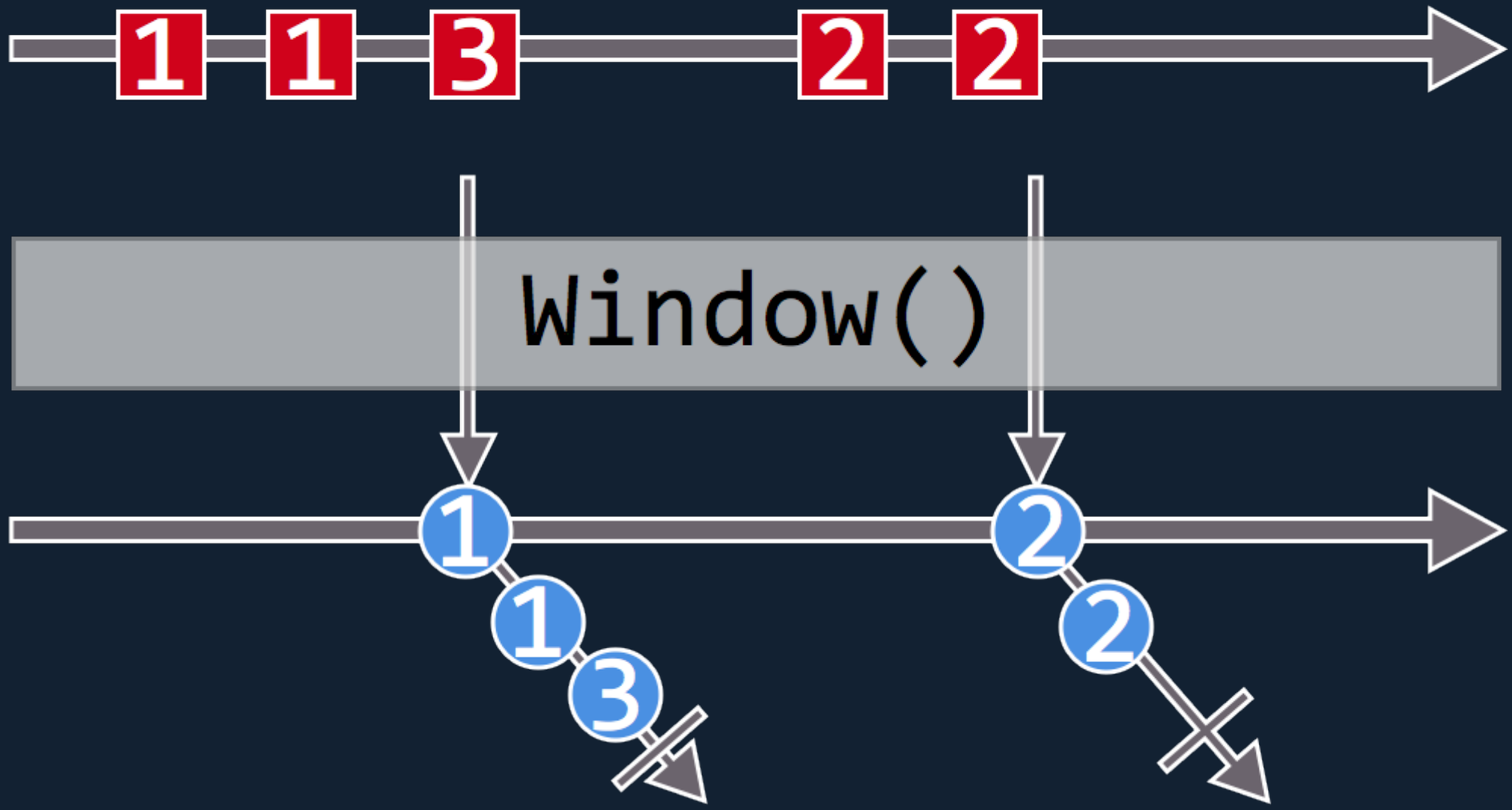
* aka: throttle, debounce
* Only emit an observable value if a specified amount of time has passed

TIME BASED DIAGRAM



* Emit a collection of items from an observable from a time span

TIME BASED DIAGRAM



* Emits observables instead of collections of items in a time span
* Each observable "packet" is completed

TIME BASED DIAGRAM

TimeInterval()

* Outputs time since last observable emission

WHEN TO USE RX

» Anything in UI

» Events

» Network requests

» Callbacks

» Push Async!

SUMMARY

» Rx deals with events

» Turns them into data streams

» Allows familiar manipulation of streams

» Anything Enumerable can do Observable can do!

» Think with marble diagrams

MATT SOUCOUP

» https://github.com/codemillmatt/Rx-Cure

» Code Mill Technologies

» Xamarin MVP

» Blog: codemilltech.com

» Email: msoucoup@codemilltech.com

» Twitter: @codemillmatt