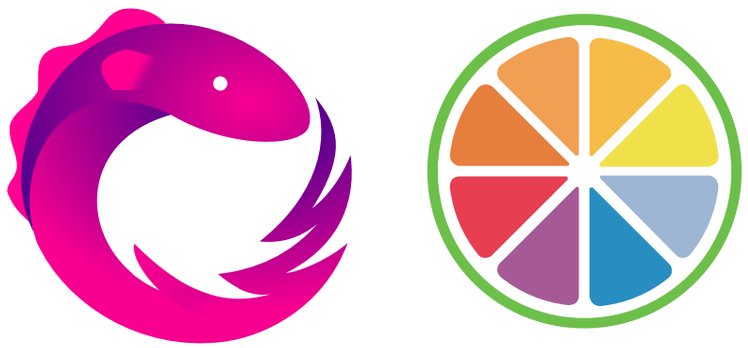


Reactive Extensions in JUCE

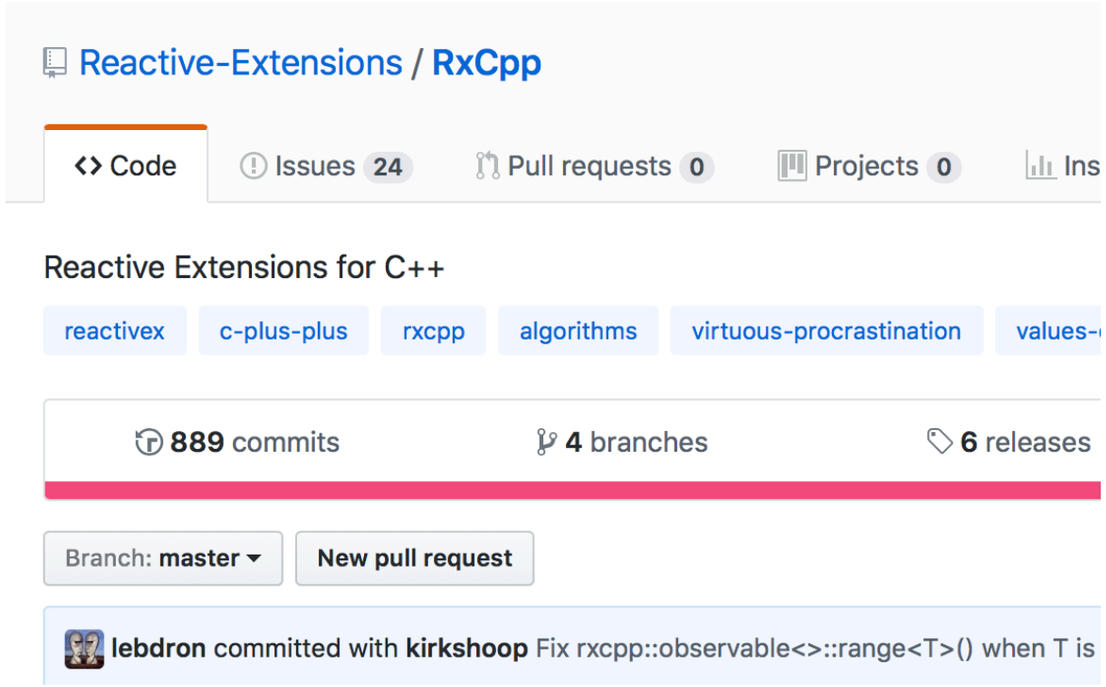
Martin Finke
ADC 2017

What is Rx?

What is Rx?

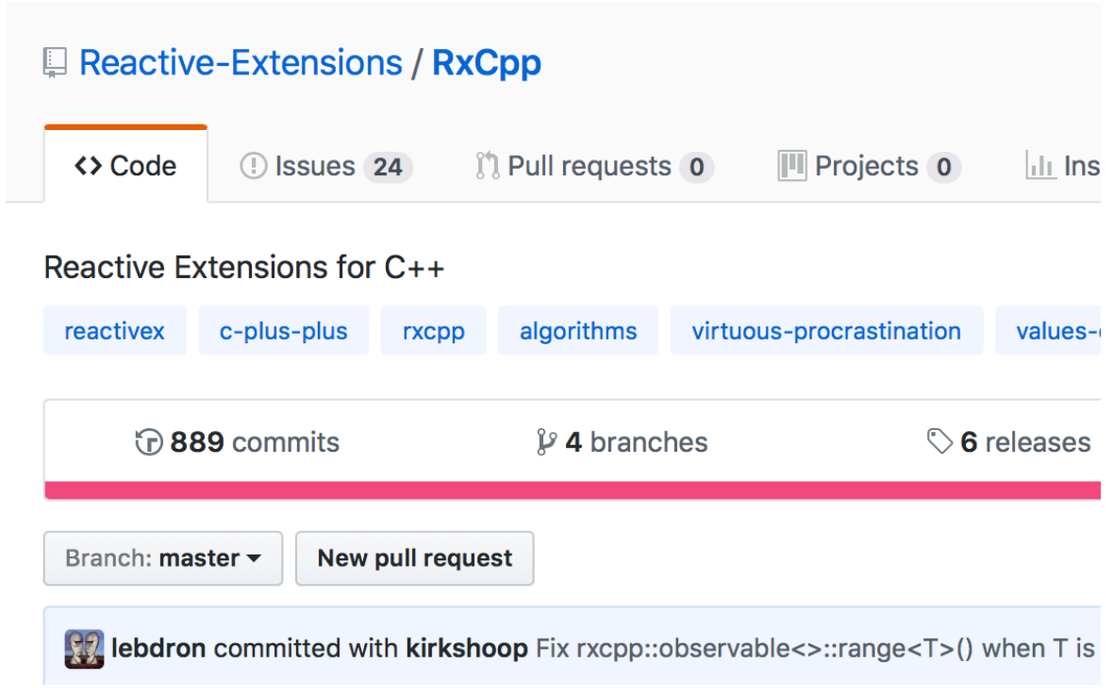
- Programming Style for Bindings (think `Value::Listener`)
- `Observable`, `Observer`
- Language-independent (RxJava, RxJS, Rx.NET, …)

RxCpp (http://github.com/Reactive-Extensions/RxCpp)



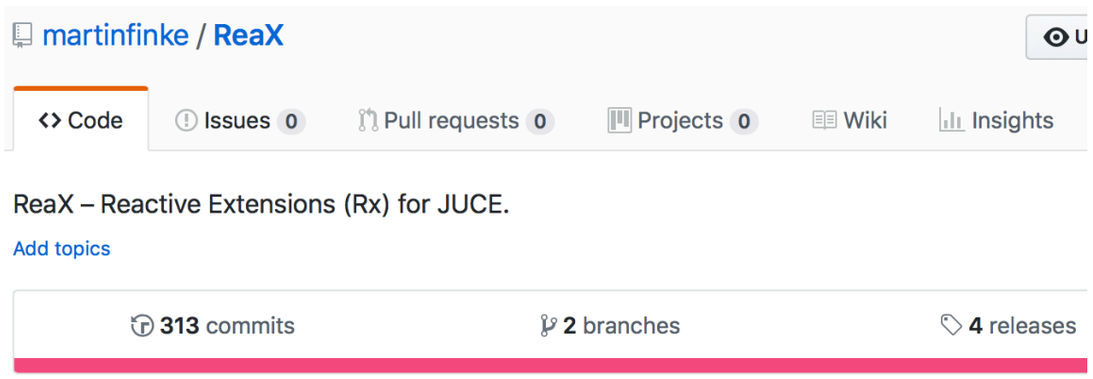
- Microsoft Open Technologies (Kirk Shoop, …)
- Header-only, Template Metaprogramming
- Compile Times

RxCpp (http://github.com/Reactive-Extensions/RxCpp)



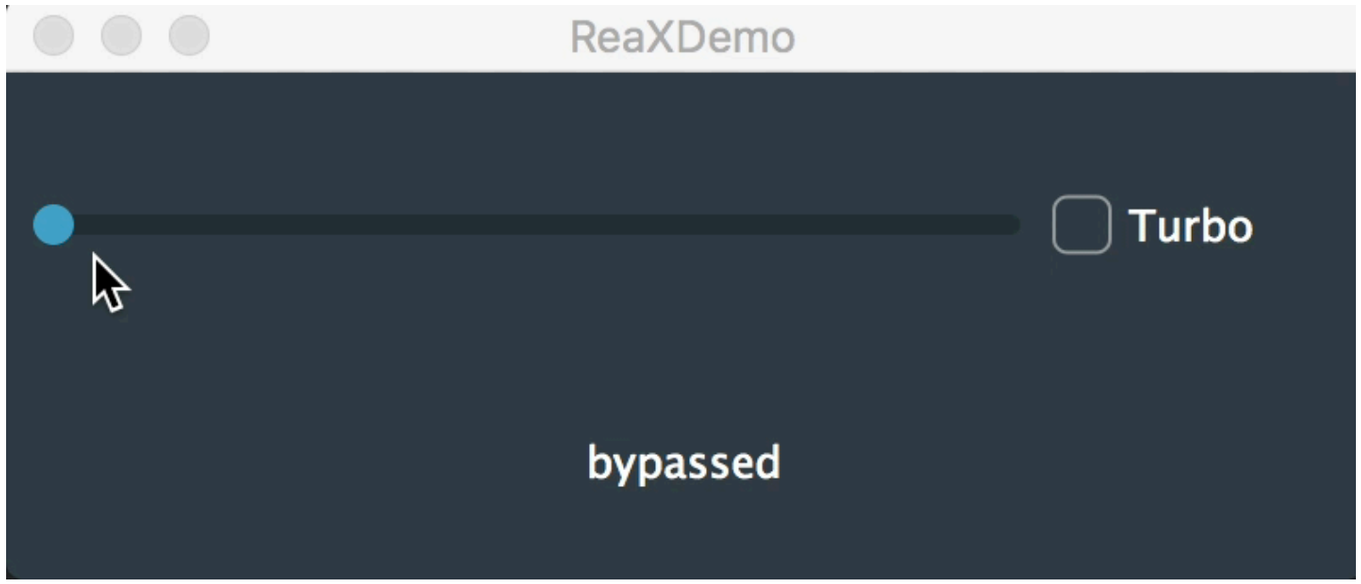
- Microsoft Open Technologies (Kirk Shoop, …)
- Header-only, Template Metaprogramming
- Compile Times

ReaX (http://github.com/martinfinke/ReaX)

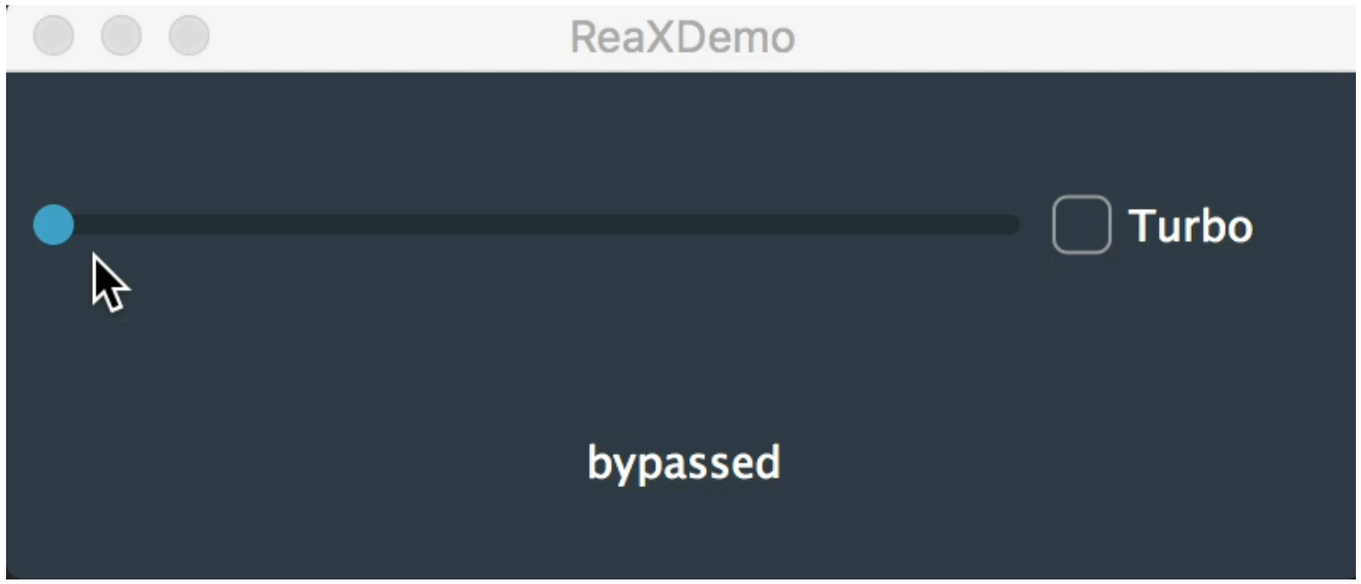


- Uses RxCpp
- Connects with JUCE Classes

Example



Example



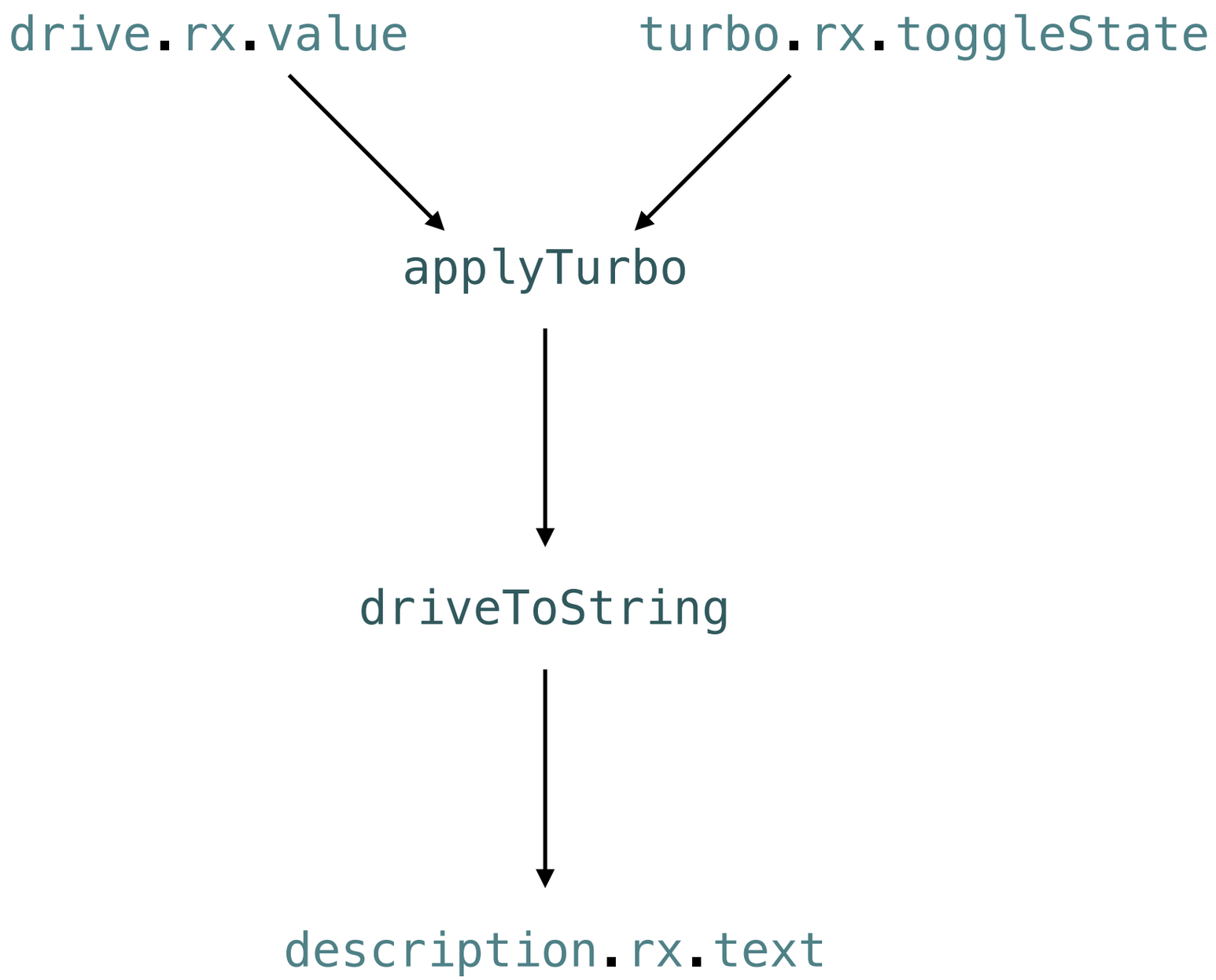
```
class MainComponent : public Component,
                      private Slider::Listener {
public:
  MainComponent() {
    drive.addListener(this);
  }

private:
  Slider drive;

  void sliderValueChanged(Slider*) override {}
};
```

```
class MainComponent : public Component,
                      private Slider::Listener {
public:
  MainComponent() {
    drive.addListener(this);
  }

private:
  Label description;
  Slider drive;

  void sliderValueChanged(Slider*) override { update(); }

  void update() {
    description.setText(driveToString(drive.getValue()));
  }

  static String driveToString(double drive);
};
```

```
class MainComponent : public Component,
                      private Slider::Listener {
public:
  MainComponent() {
    drive.addListener(this);
    update();
  }

private:
  Label description;
  Slider drive;

  void sliderValueChanged(Slider*) override { update(); }

  void update() {
    description.setText(driveToString(drive.getValue()));
  }

  static String driveToString(double drive);
};
```

```
class MainComponent : public Component,
                      private Slider::Listener,
                      private Button::Listener {
public:
  MainComponent() {
    drive.addListener(this);
    turbo.addListener(this);
    update();
  }

private:
  Label description;
  Slider drive;
  ToggleButton turbo;

  void sliderValueChanged(Slider*) override { update(); }

  void buttonStateChanged(Button*) override { update(); }
```

```
  void update() {
    double combined = applyTurbo(drive.getValue(),
                                 turbo.getToggleState());
    description.setText(driveToString(combined));
  }

  static String driveToString(double drive);
  static double applyTurbo(double drive, bool turbo);
};
```


drive.rx.value
turbo.rx.toggleState
applyTurbo
driveToString
description.rx.text

```
drive.rx.value
  .combineLatest(applyTurbo, turbo.rx.toggleState)
```



```
driveToString
```



```
description.rx.text
```

```
drive.rx.value
  .combineLatest(applyTurbo, turbo.rx.toggleState)
  .map(driveToString)
```

```
description.rx.text
```

```
drive.rx.value
  .combineLatest(applyTurbo, turbo.rx.toggleState)
    .map(driveToString)

.subscribe(description.rx.text);
```

```
drive.rx.value
  .combineLatest(applyTurbo, turbo.rx.toggleState)
  .map(driveToString)
  .subscribe(description.rx.text);
```

```
MainComponent() {
  drive.rx.value
    .combineLatest(applyTurbo, turbo.rx.toggleState)
    .map(driveToString)
    .subscribe(description.rx.text);
}
```

```
class MainComponent : public Component {
public:
  MainComponent() {
    drive.rx.value
      .combineLatest(applyTurbo, turbo.rx.toggleState)
      .map(driveToString)
      .subscribe(description.rx.text);
  }

private:
  Reactive<Label> description;
  Reactive<Slider> drive;
  Reactive<ToggleButton> turbo;

  static String driveToString(double drive);
  static double applyTurbo(double drive, bool turbo);
};
```

```
class MainComponent : public Component {
public:
  MainComponent() {
    drive.rx.value
      .combineLatest(applyTurbo, turbo.rx.toggleState)
      .map(driveToString)
      .subscribe(description.rx.text);
  }

private:
  Reactive<Label> description;
  Reactive<Slider> drive;
  Reactive<ToggleButton> turbo;

  static String driveToString(double drive);
  static double applyTurbo(double drive, bool turbo);
};
```

```
drive.rx.value
  .combineLatest(applyTurbo, turbo.rx.toggleState)
  .map(driveToString)
  .subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
dr.combineLatest(applyTurbo, turbo.rx.toggleState)
  .map(driveToString)
  .subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
dr.combineLatest(applyTurbo, tb)
  .map(driveToString)
  .subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
dr.combineLatest(applyTurbo, tb)
  .map(driveToString)
  .subscribe(description.rx.text);
```

An Observable emits values over time.

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
dr.combineLatest(applyTurbo, tb)
  .map(driveToString)
  .subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
cd.map(driveToString)
  .subscribe(description.rx.text);
```

```
static double applyTurbo(double drive, bool turbo);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
static String driveToString(double drive);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
static String driveToString(double drive);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
static String driveToString(double drive);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

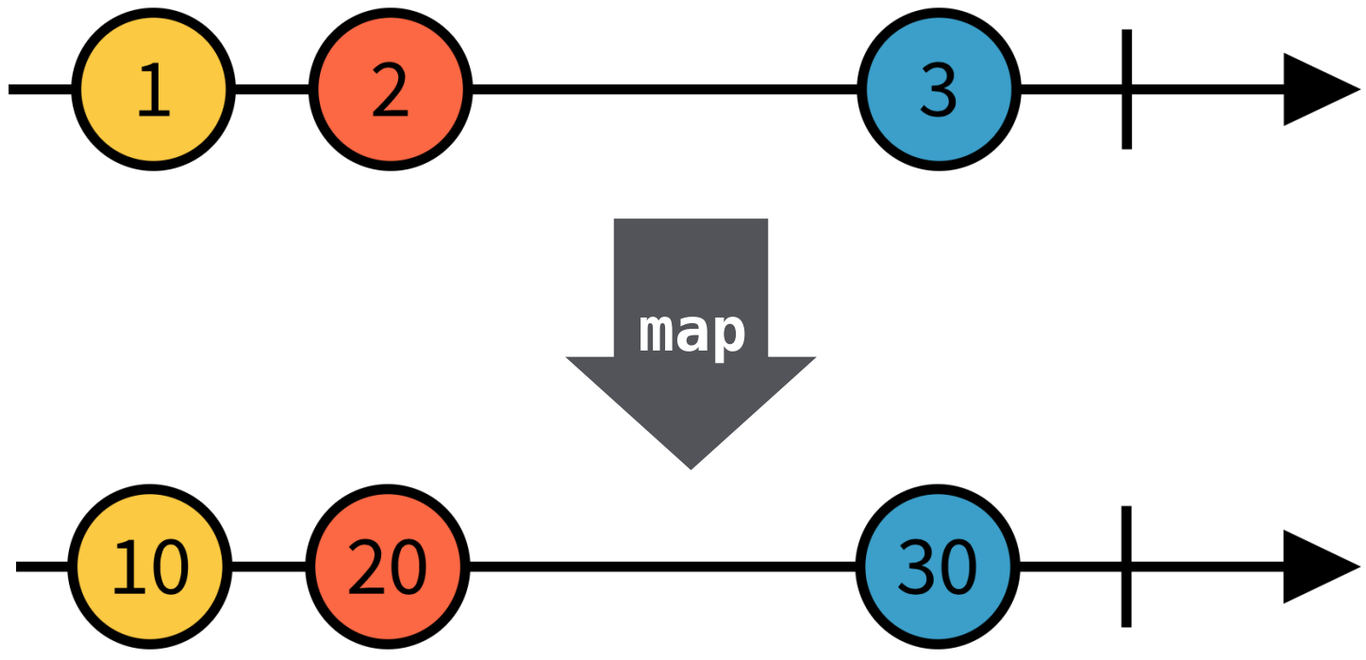
```
static String driveToString(double drive);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
static String driveToString(double drive);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

Operators create new Observables from Observables.



```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
st.subscribe(description.rx.text);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
Observer<String> labelText = description.rx.text;
st.subscribe(labelText);
```

```
Observable<double> dr = drive.rx.value;
Observable<bool> tb = turbo.rx.toggleState;
Observable<double> cd = dr.combineLatest(applyTurbo, tb);
Observable<String> st = cd.map(driveToString);
Observer<String> labelText = description.rx.text;
st.subscribe(labelText);
```

Observers receive values from Observables.

Observable

```
template<typename T>
class Observable {
public:
  template<typename U>
  Observable<U> map(std::function<U(T)> function);

  Subscription subscribe(Observer<T> observer);
  ...
};
```

Observer

```
template<typename T>
class Observer {
public:
    void onNext(T value);
    ...
};
```

Subject

```
template<typename T>
class Subject : public Observer<T>, public Observable<T> {};
```

Subject

```
template<typename T>
class Subject : public Observer<T>, public Observable<T> {};
```

```
Subject<int> numbers;

numbers.subscribe([](int number) { ··· });

numbers.onNext(17); // Calls the lambda
```

Sounds familiar?

JUCE	Rx
`Value`	`Subject`
`Value::Listener`	`Observer`
`addListener()`	`subscribe()`
`valueChanged()`	`onNext()`

Rx and Audio/MIDI

Rx and Audio/MIDI

- Real-time Thread → Real-time Thread: ⛔
  Use `MidiKeyboardState`, `MidiMessageCollector`, …

Rx and Audio/MIDI

- Real-time Thread → Real-time Thread: ⛔
  Use `MidiKeyboardState`, `MidiMessageCollector`, ...

- Real-time Thread → Main Thread: ✅
  `LockFreeSource`

Rx and Audio/MIDI

- Real-time Thread → Real-time Thread: ⛔
  Use `MidiKeyboardState`, `MidiMessageCollector`, …

- Real-time Thread → Main Thread: ✅
  `LockFreeSource`

- Main Thread → Real-time Thread: ✅
  `LockFreeTarget`

LockFreeTarget

```
template<typename T>
class LockFreeTarget : public Observer<T> {
public:
  // Call from real-time thread:
  bool tryDequeue(T& value);
  bool tryDequeueAll(T& value);
};
```

Where To Go Next

Do you need to find an operator for your problem? Start by choosing an option from the list below:

- I have one existing Observable, and...
- I have some Observables to combine together as one Observable, and...
- I have no Observables yet, and...

---

http://reactivex.io/rxjs/

Where To Go Next

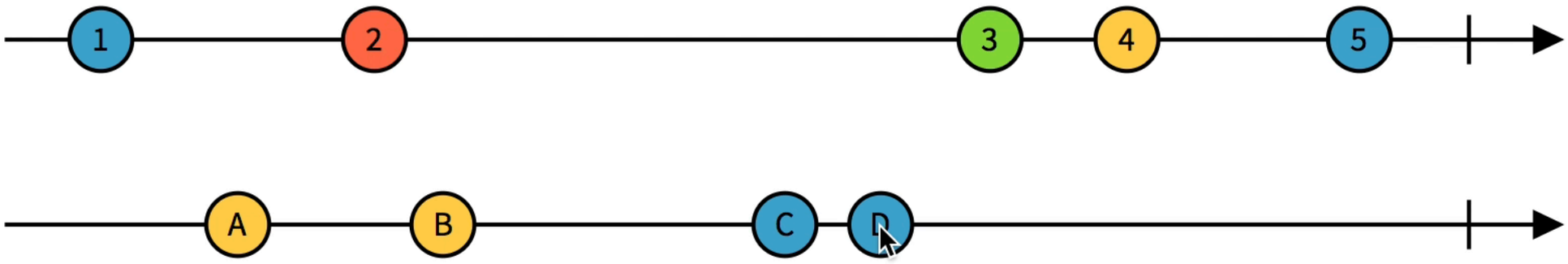
Do you need to find an operator for your problem? Start by choosing an option from the list below:

- I have one existing Observable, and...
- I have some Observables to combine together as one Observable, and...
- I have no Observables yet, and...

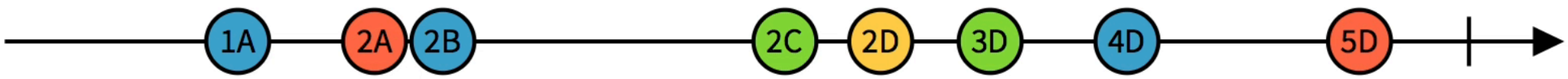
---

http://reactivex.io/rxjs/

Where To Go Next

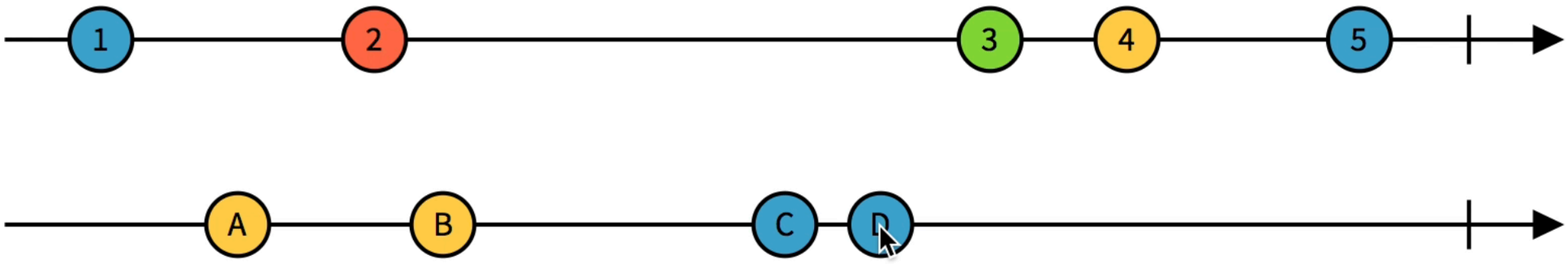



```
combineLatest((x, y) => "" + x + y)
```

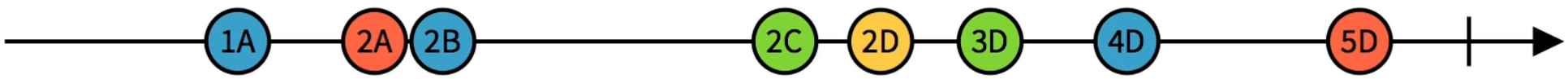


http://rxmarbles.com

Where To Go Next




```
combineLatest((x, y) => "" + x + y)
```



http://rxmarbles.com

Try it, you'll like it!

http://github.com/martinfinke/ReaX