Reactive Programming

Chaoran Chen, Jörn von Henning, Lyubomir Stoykov, Martin Mihaylov

24. March 2017

Person are reactive, why shouldn’t our apps be?

Outline

1. Current Approach with Async Programming
2. Reactive Programming
3. Research Topics

Asynchronous Programming

Current Approach

Callbacks

Promises

Callback vs Promise

```
api(function(result){
    api2(function(result2){
        api3(function(result3){
            // do work
        });
    });
});
```

```
api().then(function(result){
    return api2();
}).then(function(result2){
    return api3();
}).then(function(result3){
    // do work
}).catch(function(error) {
    //handle any error that may occur before this point
});
```

Introduction: Reactive Programming

Data Streams

Observers

Functions

Understanding Streams

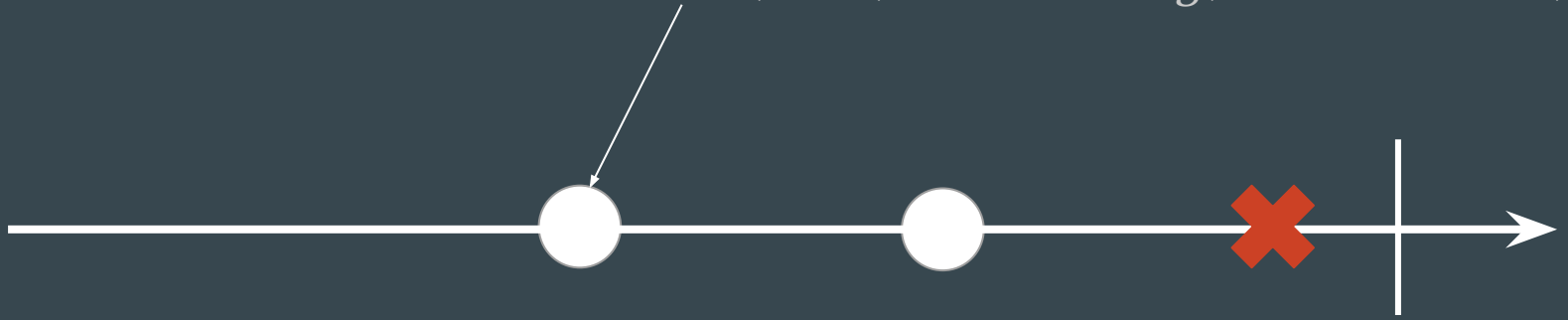
Understanding Streams

Understanding Streams

Understanding Streams

Understanding Streams

F (event) => console.log(“event occurred”);



Understanding Streams

F (event) => console.log(“event occurred”);

Understanding Streams

F (error) => console.log(“error occurred”);

Creating Streams

```
const stream = Rx.Observable.create(observer => {



});
```

Creating Streams

```
const stream = Rx.Observable.create(observer => {

  observer.onNext(1);

});
```

Creating Streams

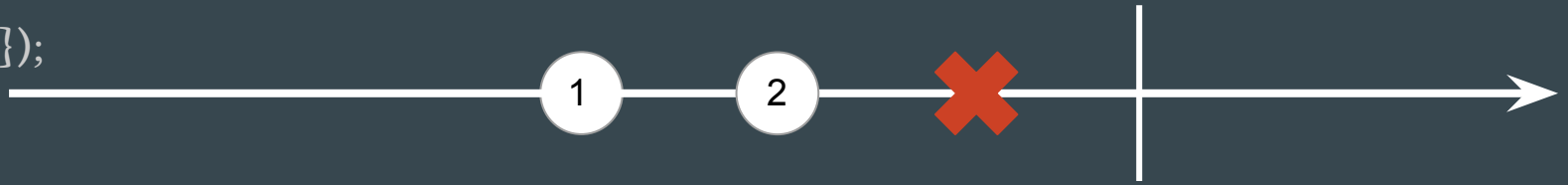
```
const stream = Rx.Observable.create(observer => {
  observer.onNext(1);
  observer.onNext(2);
});
```

Creating Streams

```
const stream = Rx.Observable.create(observer => {
  observer.onNext(1);
  observer.onNext(2);
  observer.onError("Oops!");
});
```

Creating Streams

```
const stream = Rx.Observable.create(observer => {
  observer.onNext(1);
  observer.onNext(2);
  observer.onError("Oops!");
  observer.onCompleted();
});
```



Creating Streams

```
const stream.subscribe((event) => {
  // do stuff
});
```

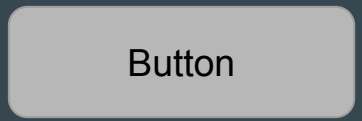
Creating Streams

```
const stream = Rx.Observable.from([1, 2]);
```

Transforming Streams

```
const stream = Rx.Observable.from([1, “2”, “3”, 4, 5, 6])


stream
    .map(x => parseInt(x))
    .filter(x => x % 2 === 0)
    .subscribe(() => alert(“I’m an even number”))
```

Event Streams



<button class="buttonClass">Button</button>

```
const button = document.querySelector(".buttonClass");

const clickStream = Rx.Observable.fromEvent(button, "click");

clickStream.subscribe(() => alert("ReactiveX is so cool!"));
```

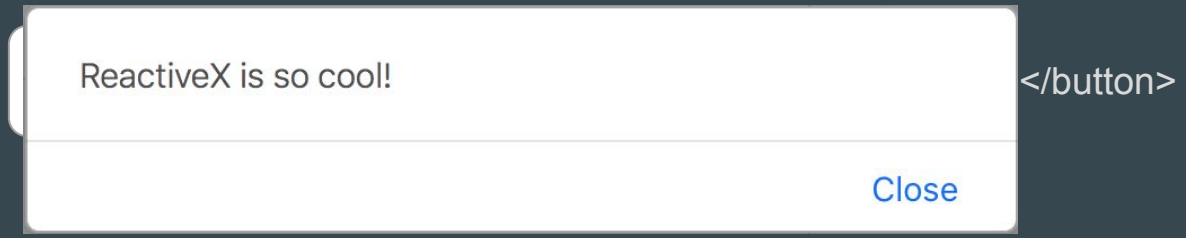
Event Streams

Button

<button class=”buttonClass”>Button</button>

Click Stream

Event Streams



Click Stream

Request & Response

Remember: Everything can be a stream!

```
const requestStream = Rx.Observable.of('https://example.com/api/');

requestStream.subscribe(function(requestUrl) {

    // Do a request here

})
```

Request & Response

```
requestStream.subscribe(function(requestUrl) {

    const responseStream = Rx.Observable.create(function (observer) {




    });

}
```

Request & Response

```
requestStream.subscribe(function(requestUrl) {

    const responseStream = Rx.Observable.create(function (observer) {

        jQuery.getJSON(requestUrl)



    });

}
```

Request & Response

```
requestStream.subscribe(function(requestUrl) {

    const responseStream = Rx.Observable.create(function (observer) {

        jQuery.getJSON(requestUrl)

            .done(response => observer.onNext(response))



    });

}
```

Request & Response

```
requestStream.subscribe(function(requestUrl) {

    const responseStream = Rx.Observable.create(function (observer) {

        jQuery.getJSON(requestUrl)

            .done(response => observer.onNext(response))

            .fail((jqXHR, status, error) => observer.onError(error))


    });

}
```

Request & Response

```
requestStream.subscribe(function(requestUrl) {
    const responseStream = Rx.Observable.create(function (observer) {
        jQuery.getJSON(requestUrl)
            .done(response => observer.onNext(response))
            .fail((jqXHR, status, error) => observer.onError(error))
            .always(() => observer.onCompleted());
    });
}
```

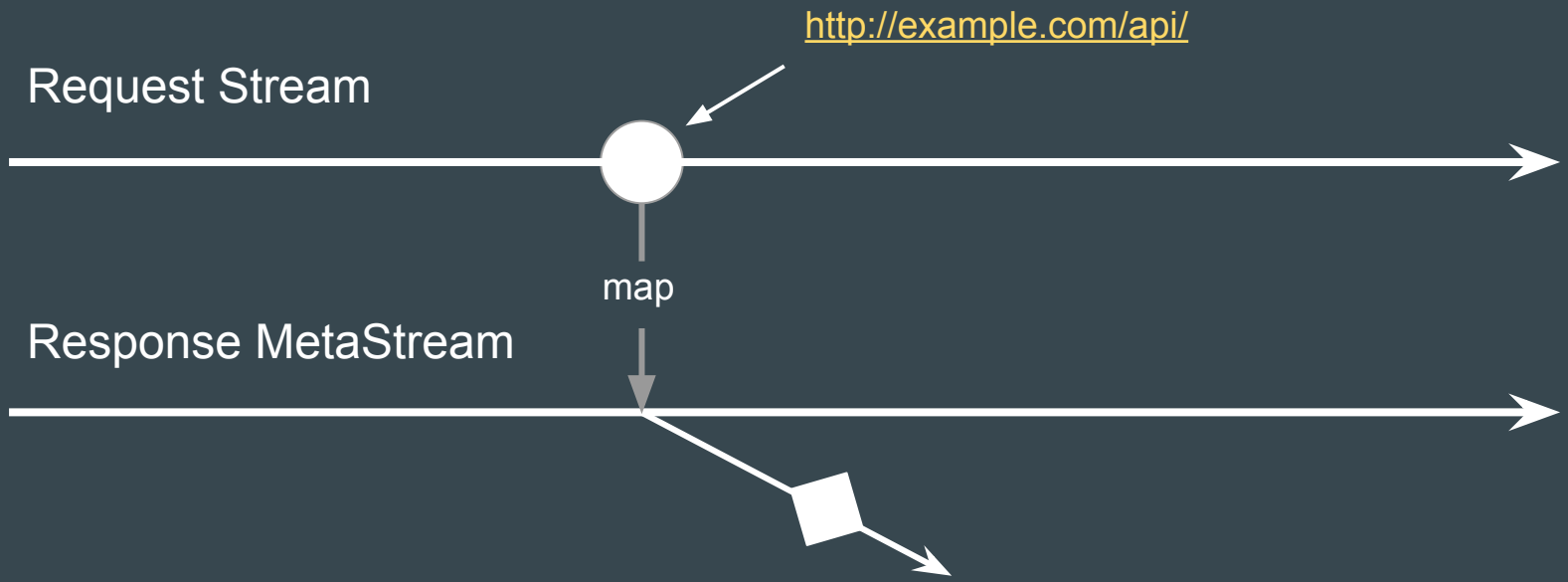
Request & Response

```
requestStream.subscribe(function(requestUrl) {

    const responseStream =

        Rx.Observable.fromPromise(jQuery.getJSON(requestUrl));

}
```

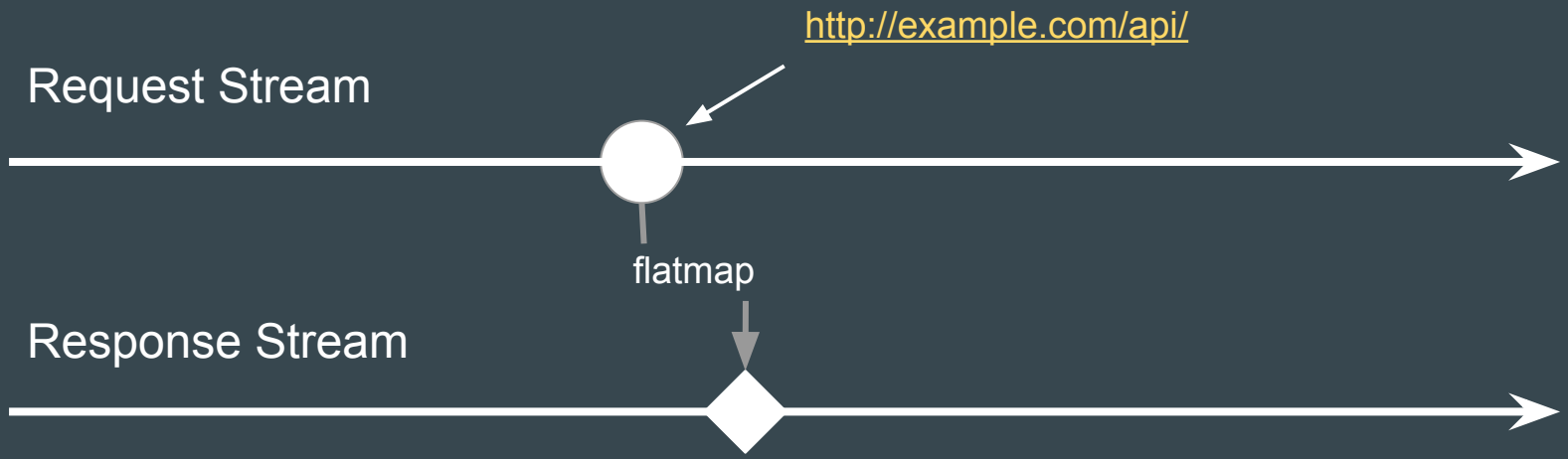
Request & Response

```
const requestStream = Rx.Observable.of('https://example.com/api/');

const responseStream = requestStream

  .map(requestUrl =>

      Rx.Observable.fromPromise(jQuery.getJSON(requestUrl))

  );
```

Request & Response



Request & Response

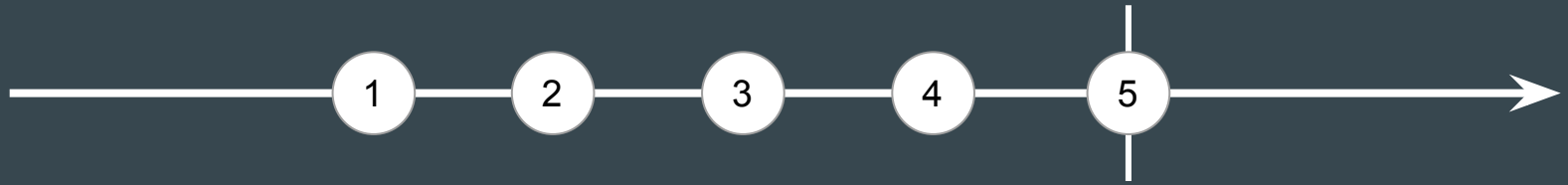


Request & Response

```
const requestStream = Rx.Observable.of('https://example.com/api/');

const responseStream = requestStream

  .flatMap(requestUrl =>

        Rx.Observable.fromPromise(jQuery.getJSON(requestUrl))

  );

responseStream.subscribe(renderToDOM);
```

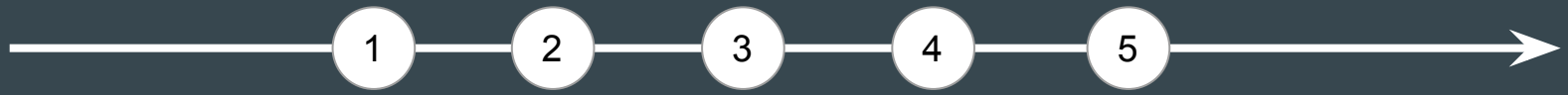
Other useful methods

```
Rx.Observable.range(1, 5);
```



Other useful methods

Rx.Observable.interval(1000);



Other useful methods

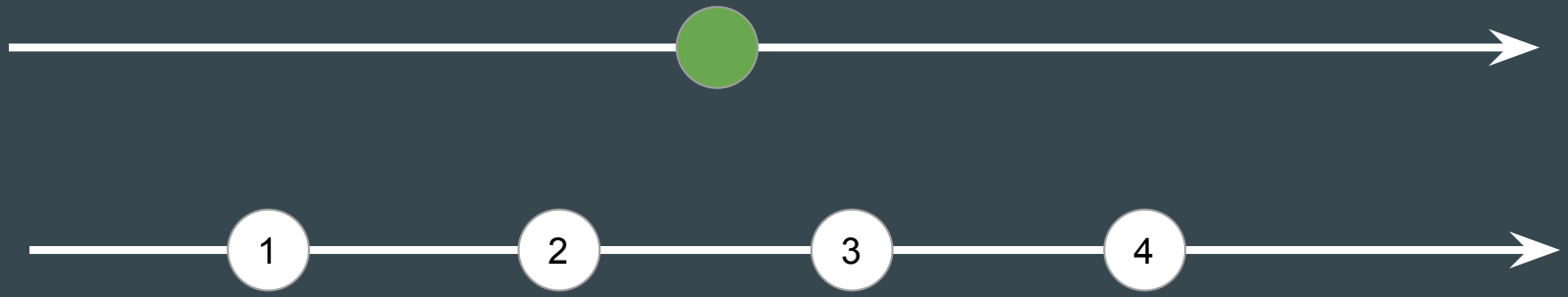
```
const clicks = Rx.Observable.fromEvent(document, ‘click’);

const timer = Rx.Observable.interval(1000);

const clicksOrTimer = Rx.Observable.merge(clicks, timer);
```

Other useful methods

```
const timer = Rx.Observable.interval(1000);
```

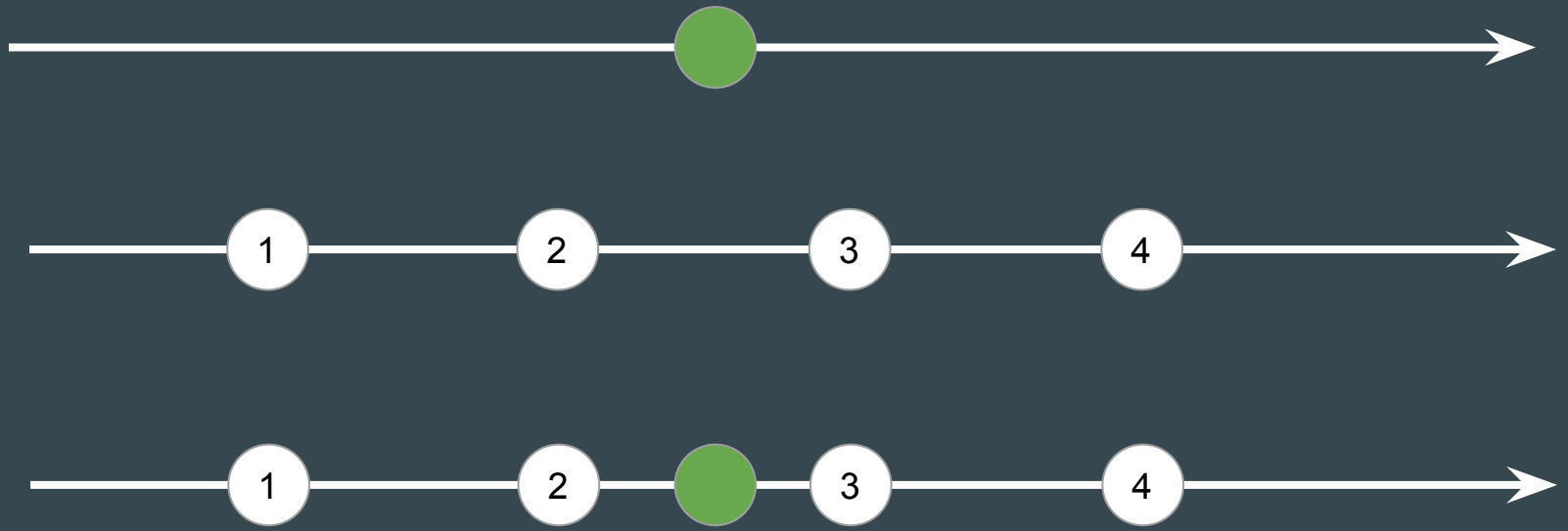
```
const clicksOrTimer = Rx.Observable.merge(clicks, timer);
```

Other useful methods



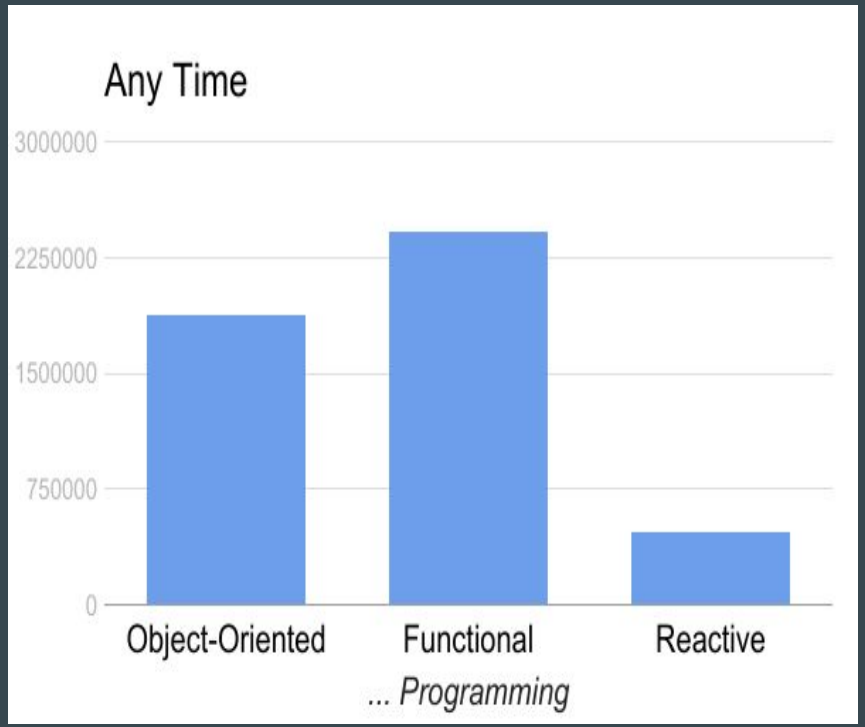

```
const clicksOrTimer = Rx.Observable.merge(clicks, timer);
```

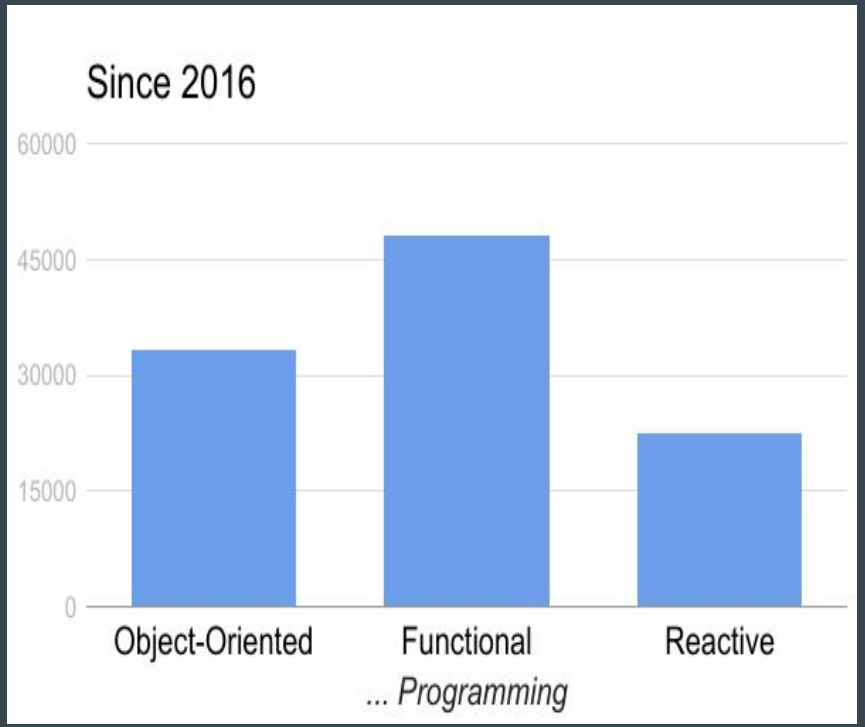
Other useful methods



Research Topics

Hits in Google Scholar





The Beginning of Functional Reactive Programming

C. Elliott und P. Hudak, „Functional Reactive **Animation**“, in Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming, New York, NY, USA, 1997, S. 263–273.

Declarative approach (What?) over imperative approach (How?)

Using Haskell:

- Higher-order functions
- Overloading of functions and operators

The Beginning of Functional Reactive Programming

Defined important data types and functions

- Time

The Beginning of Functional Reactive Programming

Defined important data types and functions

- Time
- Behavior

$$at : Behavior \rightarrow Time \rightarrow \alpha$$

The Beginning of Functional Reactive Programming

Defined important data types and functions

- Time
- Behavior

$$at : Behavior \rightarrow Time \rightarrow \alpha$$

- Event

$$occ : Event \rightarrow Time \times \alpha$$

The Beginning of Functional Reactive Programming

Defined important data types and functions

- Event handling

$$(\Rightarrow) : Event \rightarrow (\alpha \rightarrow \beta) \rightarrow Event$$

$$occ[e \Rightarrow f] = (t, f\ x)$$

$$where\ (t, x) = occ[e]$$

Flapjax: Reactive Programming for the Web

L. A. Meyerovich u. a., „Flapjax: A Programming Language for Ajax Applications“, in Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, New York, NY, USA, 2009, S. 1–20.

Comprehensibility

G. Salvaneschi, S. Proksch, S. Amann, S. Nadi, und M. Mezini, „On the Positive Effect of Reactive Programming on Software Comprehension: An Empirical Study“, IEEE Transactions on Software Engineering, Bd. PP, Nr. 99, S. 1–1, 2017.

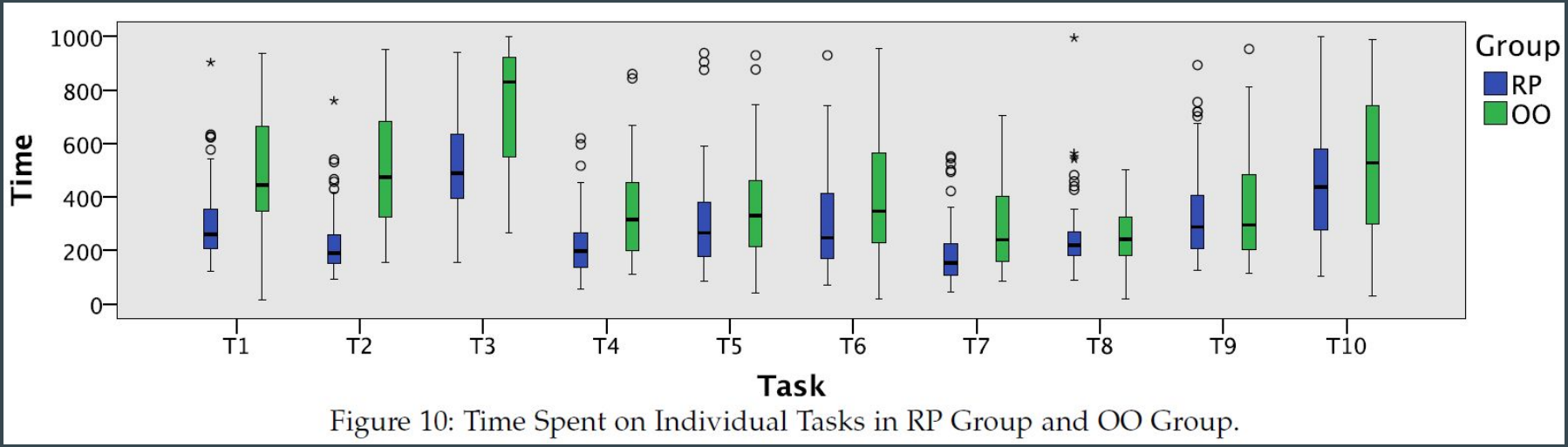


Figure 10: Time Spent on Individual Tasks in RP Group and OO Group.