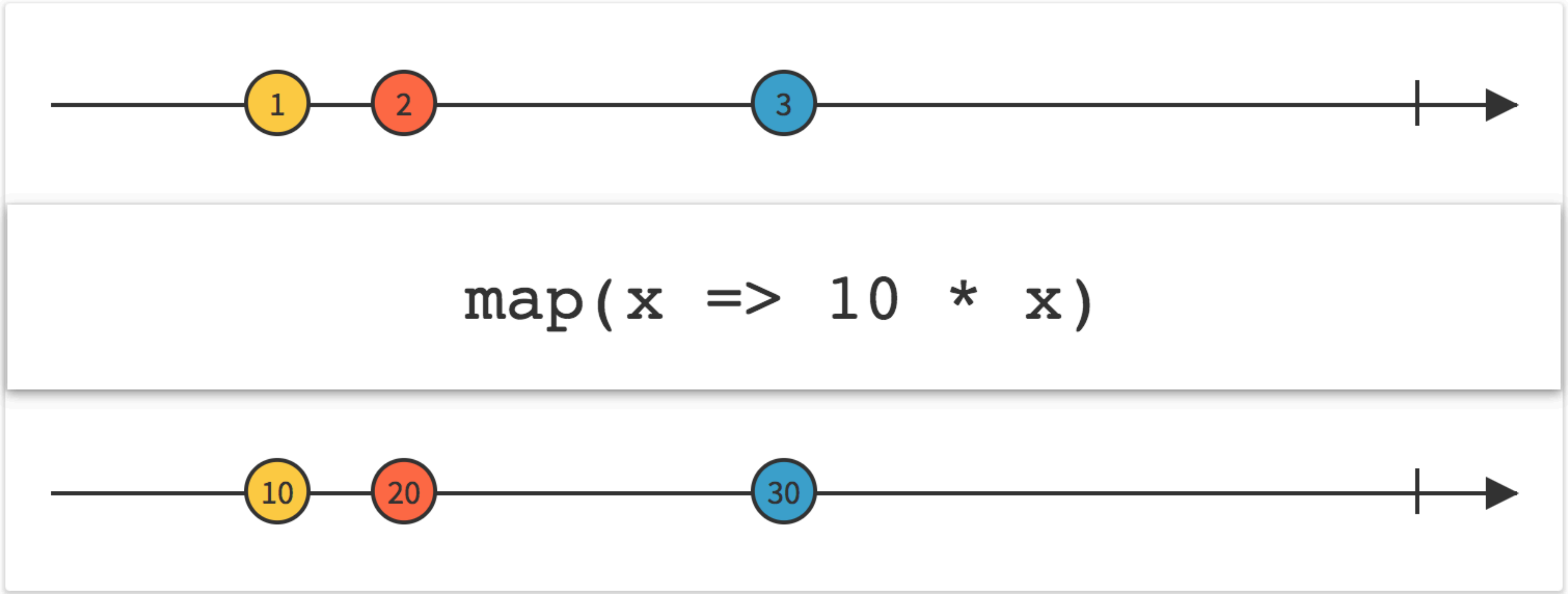
RxJS

RxJS



ilionx
Your partner in digital business

RxJS

- Reactive Programming
- RxJS & Angular
- Observable
- Operators

What is Reactive Programming?

- *a declarative programming paradigm concerned with data streams and the propagation of change*
- a way of responding to sets of events over a period of time
- a new mindset of thinking about events

Why Reactive Programming?

- solve complex problems with a few lines of code

When to use Reactive Programming?

Responding to:

- User events
- Data changes
- State changes
- All other kinds of events

Example: double click (non reactive)

```
previousClick = null;

onClick() {
  const thisClick = Date.now();
  if (thisClick - this.previousClick <= 250) {
    this.previousClick = null;
    double();
  } else {
    this.previousClick = thisClick;
    setTimeout(() => {
      if (this.previousClick) {
        this.previousClick = null;
        single();
      }
    }, 250);
  }
}
```

imperative

Example: double click (reactive)

streams
(declarative)

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

Reactive Libraries

- RxJS
- RxJava
- Rx.NET
- RxSwift
- …

RxJS & Angular

ilionx
Your partner in digital business

RxJS in Angular

- Angular Http uses Observables

```
private http: HttpClient;

this.http
      .get<Contact[]>(‘/api/contacts’)
      .subscribe()
         contacts => {

           // do something with contacts

         }
      );
```

Your partner in digital business

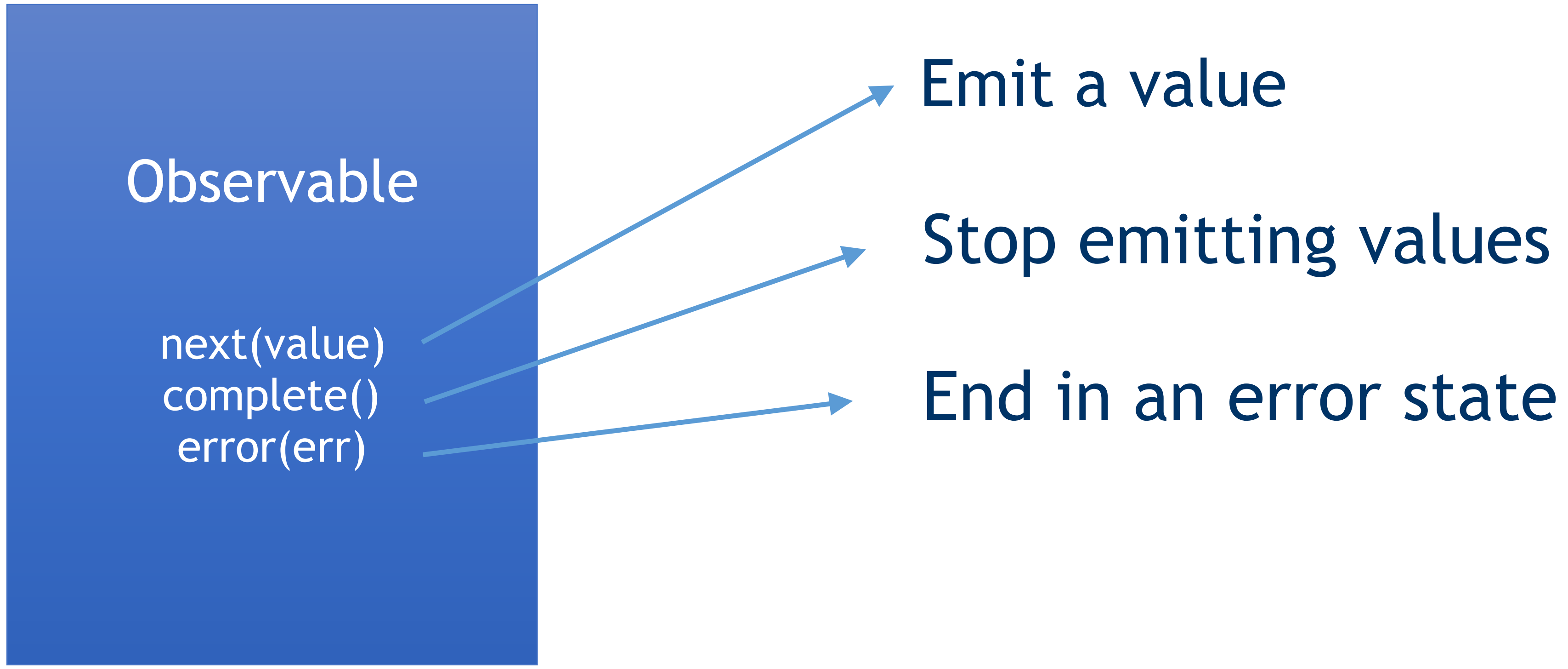
RxJS in Angular

- Angular Http uses Observables
- Change Detection uses Observables
- Router uses Observables

In Angular, Observables are everywhere!

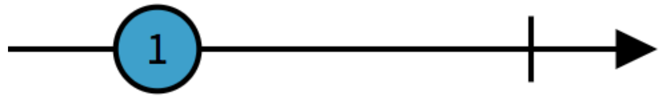
Observable

Observable: something that emits events



ilionx
Your partner in digital business

Example: Http

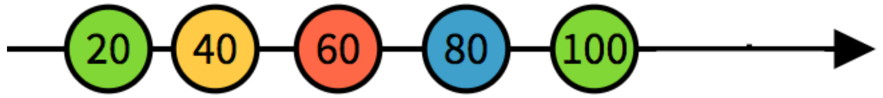


An Http call

- emits a single value
- then completes
- (or just ends in an error)

No need to unsubscribe!

Example: Router parameters



Router parameters

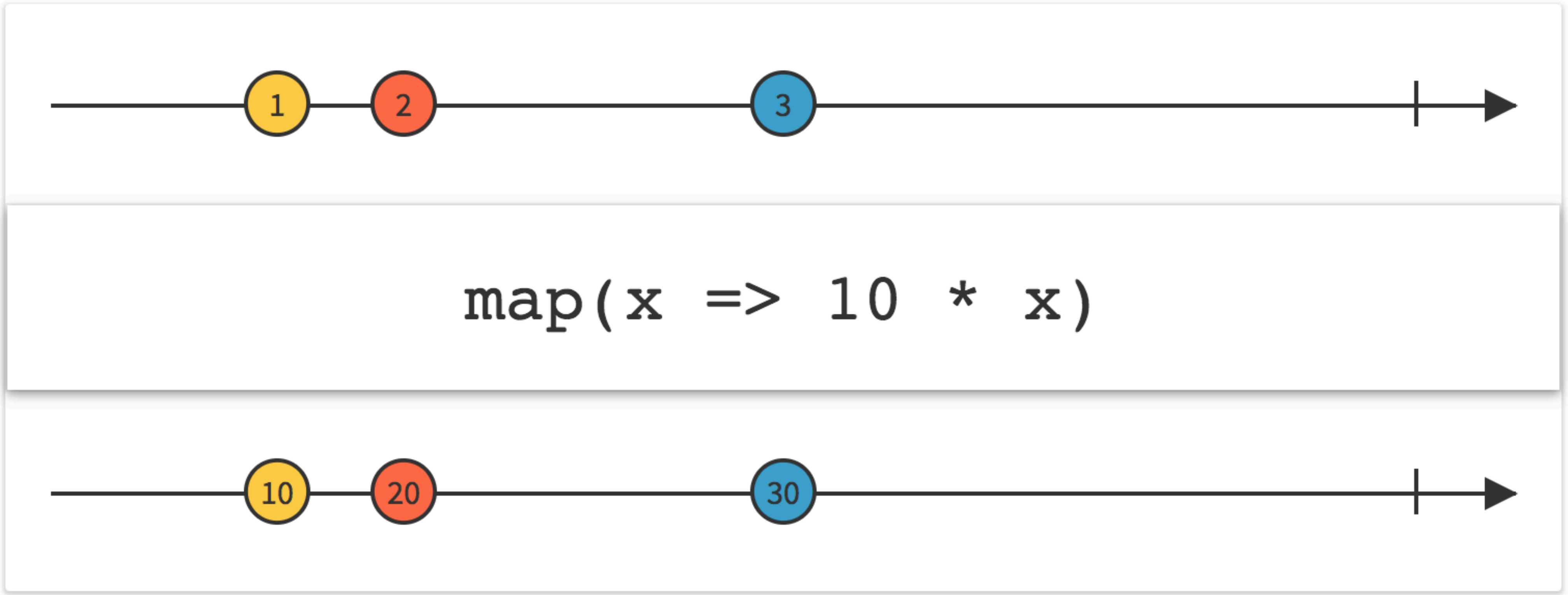
- keeps emitting values

You have to unsubscribe!

More Observables

- fromEvent(htmlElement, ‘event’)
- create your own:
  - Subject
  - BehaviorSubject
  - ReplaySubject
  - AsyncSubject

Operators



ilionx
Your partner in digital business

Operators

- transform an Observable into another
- combine Observables
- operate on Observables

Operators make Observables powerful!

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>('/who-am-i')
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

ilionx
Your partner in digital business

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>('/who-am-i')
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

ilionx
Your partner in digital business

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>('/who-am-i')
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>('/who-am-i')
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

Your partner in digital business

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>('/who-am-i')
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

ilionx
Your partner in digital business

Operator syntax

Pipeable operators (RxJS >= 5.5)

```
this.http.get<Contact>(‘/who-am-i’)
  .pipe(
    map(contact => contact.id),
    mergeMap(id => this.http.get<Details>(`/details/${id}`)),
    tap(details => console.log(details)),
    catchError(error => handle(error))
  )
  .subscribe(details => {
    // do something with contact details
  });
```

TakeUntil

Take one observable,

but terminate it when another observable emits/terminates

```
observable.pipe(
  takeUntil(someOtherObservable)
);
```

ilionx
Your partner in digital business

TakeUntil instead of unsubscribe

```
destroy = new Subject<boolean>();

ngOnInit() {
  interval(1000)
    .pipe(takeUntil(this.destroy))
    .subscribe(console.log);
}

ngOnDestroy() {
  this.destroy.next(true);
}
```

Your partner in digital business

TakeUntil instead of unsubscribe

Create a Subject

```
destroy = new Subject<boolean>();

ngOnInit() {
  interval(1000)
    .pipe(takeUntil(this.destroy))
    .subscribe(console.log);
}

ngOnDestroy() {
  this.destroy.next(true);
}
```

ilionx
Your partner in digital business

TakeUntil instead of unsubscribe

takeUntil the Subject emits

```
destroy = new Subject<boolean>();

ngOnInit() {
  interval(1000)
    .pipe(takeUntil(this.destroy))
    .subscribe(console.log);
}

ngOnDestroy() {
  this.destroy.next(true);
}
```

ilionx
Your partner in digital business

TakeUntil instead of unsubscribe

Send a value on the Subject on destroy

```
destroy = new Subject<boolean>();

ngOnInit() {
  interval(1000)
    .pipe(takeUntil(this.destroy))
    .subscribe(console.log);
}

ngOnDestroy() {
  this.destroy.next(true);
}
```

ilionx
Your partner in digital business

Double click revisited

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

Double click revisited

Observable from html event

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

ilionx

Your partner in digital business

Double click revisited

Second Observable: emits when clickStream silent for 250ms

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

ilionx
Your partner in digital business

Double click revisited

Send a value on the Subject on destroy

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

ilionx
Your partner in digital business

Double click revisited

Send a value on the Subject on destroy

```
@ViewChild('button') button;

ngAfterViewInit() {
  const clickStream = fromEvent(this.button.nativeElement, 'click');

  const finalClick = clickStream.pipe(debounceTime(250));

  clickStream
    .pipe(buffer(finalClick))
    .subscribe(clicks =>
      clicks.length === 1 ? single() : double()
    );
}
```

ilionx
Your partner in digital business

More operators

- filter, distinct, elementAt
- join, merge, zip
- buffer, debounce, delay
- skipWhile, takeUntil
- average, count
- …and many more

More information

- reactivex.io
- rxmarbles.com
- … google

ilionx
Your partner in digital business