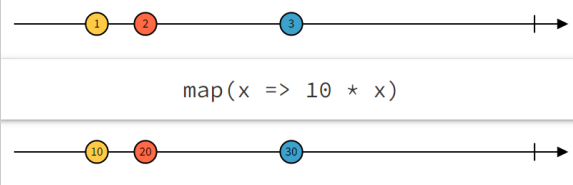


Rxjs Useful Operators

Map

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Transform / modify value of one observable according to the requirement• Map transform value not type

```
const demo = this.http.get('./assets/sample.json').pipe(
  tap(res => console.log('original res => ', res)),
  map(res => res['data']));
demo.subscribe(res => console.log(res));
```

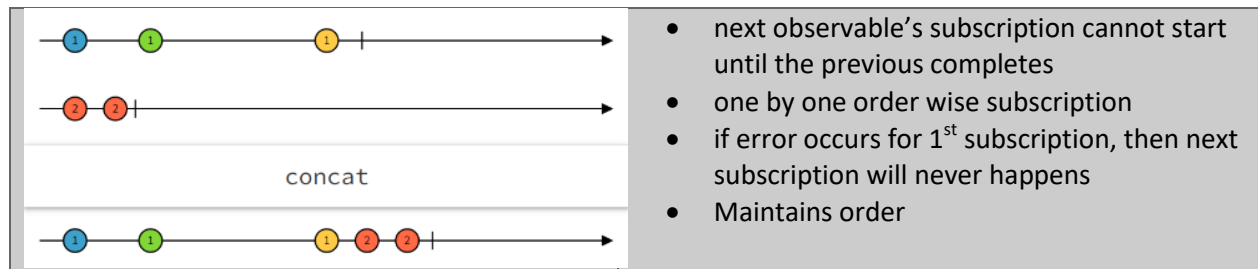
```
original res =>
▼ {meta: {...}, data: Array(1), links: {...}}
  ▶ meta: {totalPages: 13}
  ▶ data: [{...}]
    ▶ links: {self: 'http://example.com/articles?page[number]=3&page[size]=1', first: 'http://example.com/articles?page[number]=1&page[size]=1', prev: 'http://example.com/articles?page[number]=3&page[size]=1', next: 'http://example.com/articles?page[number]=3&page[size]=1'}
    ▶ meta: {totalPages: 13}
    ▶ [[Prototype]]: Object
  ▶ [[Prototype]]: Array(0)
  ▶ 0: {type: 'articles', id: '3', attributes: {...}}
    length: 1
    ▶ [[Prototype]]: Array(0)
```

Real time Examples

- Mapping Returned HTTP response
- If you want to transform/modify 1st API response to pass as request to 2nd API to fetch more data related to the results

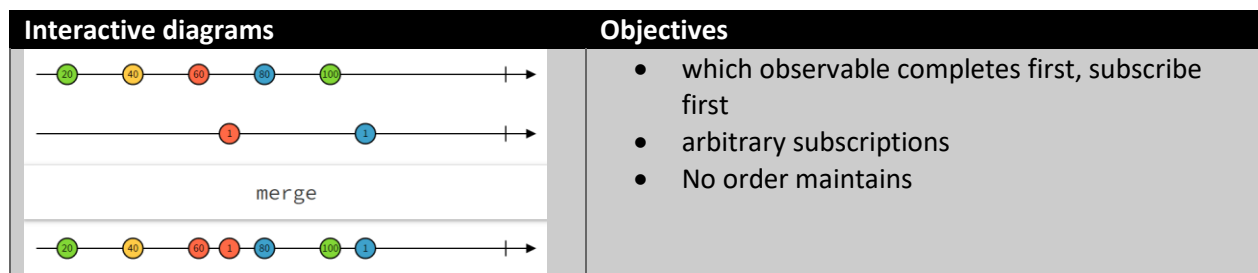
Concat

Interactive diagrams	Objectives
----------------------	------------



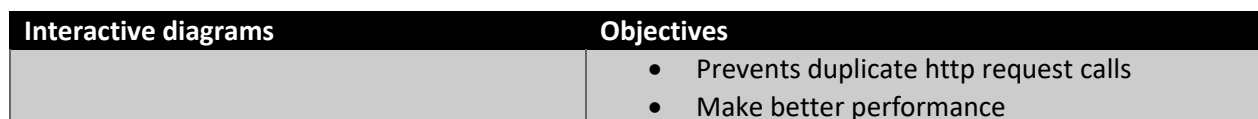
Real time Examples

Merge



Real time Examples

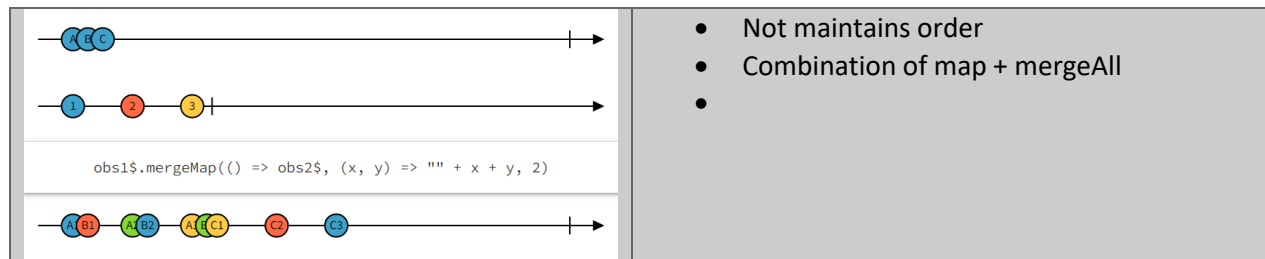
ShareReplay



Real time Examples

MergeMap





Real time Examples

1. Let's say you show a live list of movies (via push notifications, like netflix).

For each movie, you might want to also call a 3rd party service, like IMDB to show it's "up to date" movie rank.

So you can mergeMap the movie into an http request to IMDB and by that, to enhance your movie cards with this additional data.

2. CEO wants the developers name who has knowledge in these technologies [backbone, angular]

```
const frameworkTweetsObservable = from(['Backbone', 'Angular'])
  .pipe(
    tap(fwName => console.log(`*** "${fwName}" tweet pops up ***`))
  ); // don't mind this ^ logging code,
// we just use it to show in the console when a new tweet comes
inframeworkTweetsObservable.pipe( map(framework => getAgency(framework) ),
  mergeMap(agency => agency.getRecruitsObservable() )).subscribe( recruit =>
  console.log(recruit) ); // CONSOLE:
```

```
// *** "Backbone" tweet pops up ***
// Backbone Developer #1
// Backbone Developer #2
// *** "Angular" tweet pops up ***
// Angular Developer #1
// Backbone Developer #3 (<-- still happening)
// Angular Developer #2
```

concatMap

	<ul style="list-style-type: none"> • maintains order • combination of map + concatAll • subscribing to the next Observable in the queue, only when the last one is completed
--	---

Real time Examples

CEO, was polite (and had tons of spare time), he could have chosen to wait until he finished all the interviews with the Backbone developers, before he started interviewing the Angular developers.

Even if he saw that Angular was the hot new framework, he would have waited until there were no more Backbone developers to interview.

```
const frameworkTweetsObservable = from(['Backbone', 'Angular'])
  .pipe(
    tap(fwName => console.log(`*** "${fwName}" tweet pops up ***`))
  );
frameworkTweetsObservable.pipe(
  map(framework => getAgency(framework) ),
  concatMap(agency => agency.getRecruitsObservable() )
).subscribe( recruit => console.log(recruit) );//
CONSOLE:
```

```
// *** "Backbone" tweet pops up ***
// Backbone Developer #1
// *** "Angular" tweet pops up ***
// Backbone Developer #2
// Backbone Developer #3 <-- this is the last one
// Angular Developer #1 <-- only now the Angular devs appear
// Angular Developer #2
// Angular Developer #3
```

switchMap

Interactive diagrams	Objectives
	<ul style="list-style-type: none"> • only care about the latest Api calls • cancel previous API calls

Real time Examples

1. If you Google something, you press a key on the big input box, and then you get suggestions for things you might mean to write.

So every new input triggers a new ajax request for that search term.

If you just used mergeMap you'll get suggestions for every key stroke
("m", "my ", "my p" "my parrot is looking at me like I owe him money!")

But switchMap will make sure that the ongoing http request is being canceled on every new search input, and only the newest http request is live.

```
<input type="text" #txtField>

this.searchSubscription = fromEvent<any>(this.txtField.nativeElement, 'input')
  .pipe(
    // Time in milliseconds between key events
    debounceTime(1000),
    map(event => event.target.value),
    distinctUntilChanged(),
    switchMap(val => this.apiService.getUsersByLocation(val)),
    takeUntil(this.unsubscribe$)
  )
  .subscribe(data => {
    console.log(data);
    console.log('data is collected');
  });
```

2. CEO was just interested in getting only the developers who knew how to code in the latest technology. We have one observable(behavioursubject) with multiple technologies [backbone, angular]

```
const frameworkTweetsObservable = from(['Backbone', 'Angular'])
  .pipe(
    tap(fwName => console.log(`*** "${fwName}" tweet pops up ***`))
  ); frameworkTweetsObservable.pipe( map(framework => getAgency(framework) ),
  switchMap(agency => agency.getRecruitsObservable() )).subscribe( recruit =>
  console.log(recruit) );// CONSOLE:

// *** "Backbone" tweet pops up ***
// Backbone Developer #1
// *** "Angular" tweet pops up ***
// Angular Developer #1
```

```
// Angular Developer #2
// Angular Developer #3
```

exhaustMap

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Combination of exhaust + map• Ignore/cancel other values until that observable completes• Prevents continues calling of API before one finish• Make better performance

Real time Examples

1. Saving data in database

Let's say user is continuously clicking on save button, then exhaustMap prevents continuous calling service / send requests until first request is not completed.

2. Login screen

Let's say you have a login screen with a login button, where you map each click to a login service API request.

If the user clicks more than once on the login button, it will cause multiple calls to the server, and you probably don't want that...

So you can use exhaustMap to temporarily "disable" the mapping while the first http request is still on the go — this makes sure you never call the server while the current request is running.

forkJoin

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• If you want response of multiple APIs at one time together• forkJoin accepts a variable number of observables and subscribes to them in parallel. When all of the provided observables complete, forkJoin collects the last emitted value from each and emits them as an array• If any input observable errors at some point, forkJoin will error as well and all other observables will be immediately

	unsubscribed. • Maintains order
--	------------------------------------

It's worth noting that the forkJoin operator will preserve the order of inner observables regardless of when they complete.

To demonstrate this let's create 2 delayed observables where the second observable will complete first:

```
const joinedAndDelayed$ = forkJoin(
  of('Hey').pipe(delay(2000)),
  of('Ho').pipe(delay(3000)),
  of('Lets go!').pipe(delay(1000))
);

joinedAndDelayed$.subscribe(console.log);
```

The first observable, despite completing last, will still be the first result in the output array as forkJoin preserves the order of observables that were passed in - giving us:

```
['Hey', 'Ho', 'Lets go!'];
```


```
const color$ = from(['white', 'green', 'red', 'blue']);
const logo$ = from(['fish', 'dog', 'bird', 'cow']);
forkJoin(color$, logo$).subscribe(([color, logo]) => console.log(`${color} =====> ${logo}`))
blue =====> cow
```

```
const color1$ = from(['white1', 'green1', 'red1', 'blue1']);
const logo1$ = from(['fish1', 'dog1']);
forkJoin(color1$, logo1$).subscribe(([color, logo]) => console.log(`${color} =====> ${logo}`));
blue1 =====> dog1
```

Real time Examples

- if you wish to issue multiple requests on page load (or some other event) and only want to take action when a response has been received for all. In this way it is similar to how you might use Promise.all.
- if you have an observable that emits more than one item, and you are concerned with the previous emissions forkJoin is not the correct choice. In these cases you may be better off with an operator like combineLatest or zip.
- If you have more than one Highchart in a single component and has dependency on 3 different API responses. You need to get all 3 API responses at one time to bind the data to the Highcharts.

Zip

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• zip doesn't start to emit until each inner observable emits at least one value• zip emits as long as emitted values can be collected from all inner observables• zip emits values as an array• zip maintains exact sequence of emitted response in the array as like the observables are zipped

```
const color$ = from(['white', 'green', 'red', 'blue']);
const logo$ = from(['fish', 'dog', 'bird', 'cow']);
zip(color$, logo$).subscribe(([color, logo]) => console.log(`${color} =====> ${logo}`));
```

white =====> fish

green =====> dog

red =====> bird

blue =====> cow

```
const color1$ = from(['white1', 'green1', 'red1', 'blue1']);
const logo1$ = from(['fish1', 'dog1']);
zip(color1$, logo1$).subscribe(([color, logo]) => console.log(`${color} =====> ${logo}`));
```

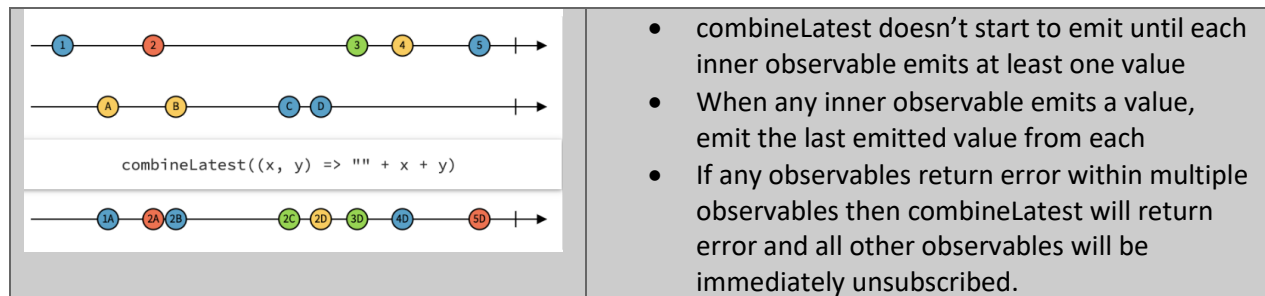
```
white1 =====> fish1
```

green1 =====> dog1

[illegible]

combineLatest

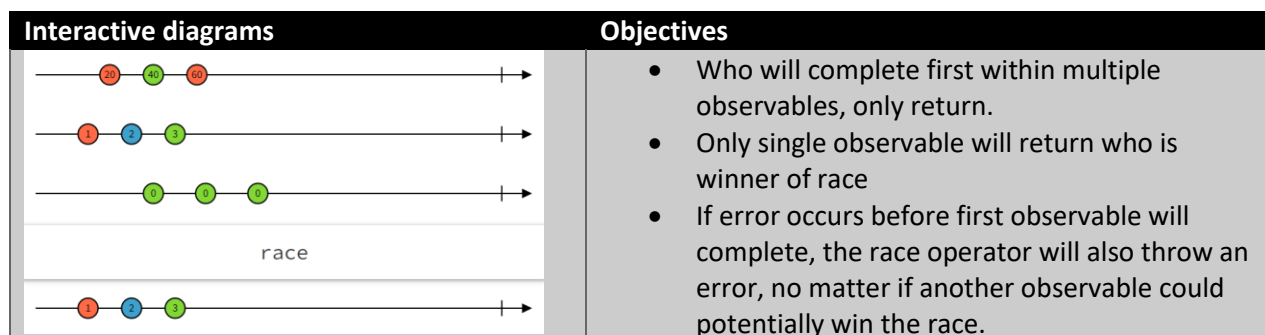
Interactive diagrams	Objectives
----------------------	------------



```
const amount = of(70, 72, 76, 79, 75);
const conversionRate = of(0.06, 0.07, 0.08);
const fees = combineLatest(amount, conversionRate).pipe(
  map(([a, r]) => (a * r)),
);
fees.subscribe(x => console.log('commission is ' + x));
```

commission is 4.5
 commission is 5.2500000000000001
 commission is 6

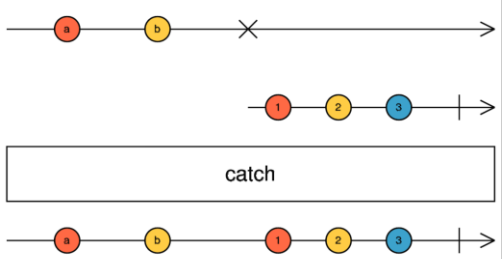
Race



Real time Examples

- race can be useful for selecting the response from the fastest network connection for HTTP or WebSockets.
- race can also be useful for switching observable context based on user input

catchError

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Catches errors on the observable to be handled by returning a new observable or throwing an error

Real time Examples

- Use in interceptor for handling http errors before occurring errors in components
- Handle errors in services

Retry

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Retry an observable sequence a specific number of times• retry can be used to retry a failed network request

Real time Examples

- Retrying HTTP requests if error occurs in api calls in services

retryWhen

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Retry an observable sequence based on a specific criteria• retryWhen can be used to retry a failed network request• can put condition with retryWhen

Real time Examples

- Use `retryWhen` in case of a 404 failure because 404 error means API url is not present, but in case of 500X exceptions it is really mandatory

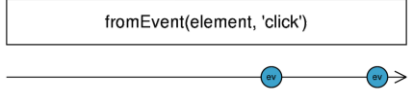
Tap

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• passes values along to the observer (mirroring) without modification• mostly used for side-effects

Real time Examples

- used for changing observable values without hampering original values
- want to display ascending or descending order but need original copy as well
- call another functions from inside http GET/POST call like set user details depending on user response, set theme depending on user type, display dialog, show snackbar, redirect to different route
- debugging purpose like `console.log`
- dispatch actions for store (if you are using any - for example `ngRx` store)
- anything what can be considered as 'side effect' for your stream

fromEvent

Interactive diagrams	Objectives
 <p>The diagram shows a code box with the text <code>fromEvent(element, 'click')</code>. Below the box, a horizontal line represents an event stream. It starts with a blue circle containing the text 'ev', followed by an arrow pointing to the right. At the end of the arrow is another blue circle containing the text 'ev'.</p>	<ul style="list-style-type: none">• <code>fromEvent</code> accepts as a first argument event target, which is an object with methods for registering event handler functions• As a second argument it takes string that indicates type of event we want to listen for.

Real time Examples

- `fromEvent` supports DOM EventTargets with `addEventListener`/ `removeEventListener` methods

Interval

Interactive diagrams	Objectives
	<ul style="list-style-type: none">• Emit numbers in sequence based on provided timeframe• (interval, timer) allow you to pipe conditional operators (takeWhile, skipWhile for example) which allows you to add a stop or implement a stop-start logic by just flipping a boolean flag, instead of adding complicated logic of clearing the interval, and then recreating it.

Real time Examples