Tips and Recap

Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata

Module Overview

Key points, tips and common issues

Debugging Observables

Brief recap

Learning more

Key points, tips and common issues

RxJS Operators

There are over 100 RxJS operators

Documentation: rxjs.dev

Pipe emitted items through a sequence of operators

An operator subscribes to its input Observable

And creates an output Observable

```
of(2, 4, 6)
  .pipe(
     map(item => item * 2),
     tap(item => console.log(item))
  );
```

Use the **async** Pipe

Handles subscribe and unsubscribe

Emits into the variable defined in the as clause

```
<div *ngIf="products$ | async as products">
 <table>
   <tr *ngFor="let product of products">
     <td>{{ product.productName }}</td>
     <td>{{ product.productCode }}</td>
   </tr>
 </table>
</div>
```

Data Streams

Emits one item, the response

After emitting the response, the stream completes

The response is often an array

To transform the array elements:

- Map the emitted array
- Map each array element
- Transform each array element

Action Streams

Only emits if it is active

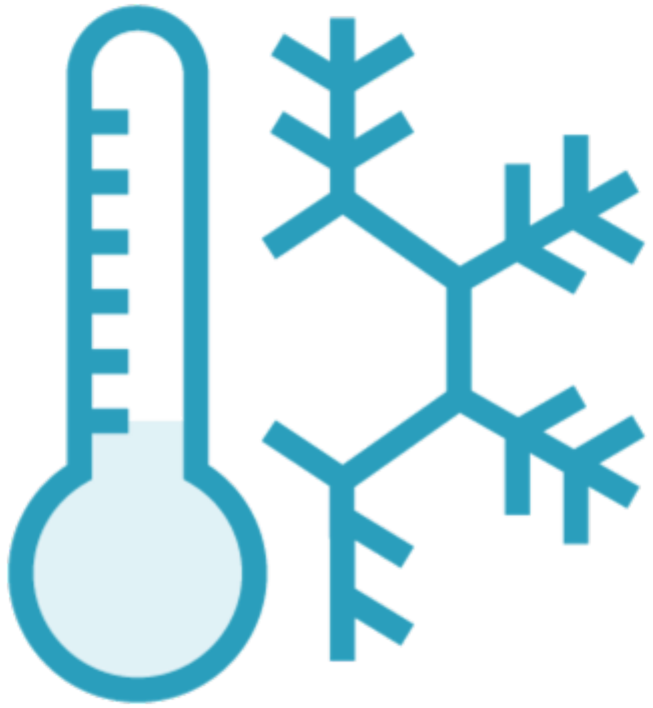
If the stream is stopped, it won't emit

An unhandled error causes the stream to stop

Catch the error and replace the errored Observable

- Don't replace an errored action Observable with EMPTY
- Replace with a default or empty value

Hot and Cold Observables



Cold Observable
Doesn't emit until subscribed to
Unicast

```
products$ = this.http.get<Product[]>(url)
                  .subscribe();
```

Hot Observable
Emits without subscribers
Multicast

```
productSubject = new Subject<number>();
this.productSubject.next(12)
```

Combination Operators/Creation Functions

Don't emit until each input Observable emits at least once

- `combineLatest`
- `forkJoin`
- `withLatestFrom`

Action stream created with a `Subject` does not immediately emit

When combining an action stream, consider using a `BehaviorSubject` since it emits a default value

Nested Subscriptions

Replace nested subscriptions

```
of(3, 7)
 .pipe(
   map(id => this.http.get<Supplier>
              (`${this.url}/${id}`)
 )).subscribe(o => o.subscribe());
```

With higher-order mapping operators

```
of(3, 7)
 .pipe(
   concatMap(id => this.http.get<Supplier>
              (`${this.url}/${id}`)
 )).subscribe();
```

Large Pipelines

```
dataForUser$ = this.userSelectedAction$.pipe(
  // Handle the case of no selection
  filter(userName => Boolean(userName)),
  // Get the user given the username
  switchMap(userName => this.http.get<User>(`${this.userUrl}?username=${userName}`)
    .pipe(
      switchMap(user =>
        // Get the todos and posts for the user
        combineLatest([
          this.http.get<ToDo[]>(`${this.todoUrl}?userId=${user.id}`),
          this.http.get<Post[]>(`${this.postUrl}?userId=${user.id}`)
        ])
          .pipe(
            // Map the data into the desired format for display
            map(([todos, posts]) => ({
              name: user.name,
              todos: todos,
              posts: posts
            }) as UserData)
          )
      ))
  ));
```

Break up Large Pipelines

Break a large pipeline into smaller pieces

- Easier to read
- Easier to debug
- Easier to maintain
- More readily reusable
- Can be bound in the UI

Large Pipelines

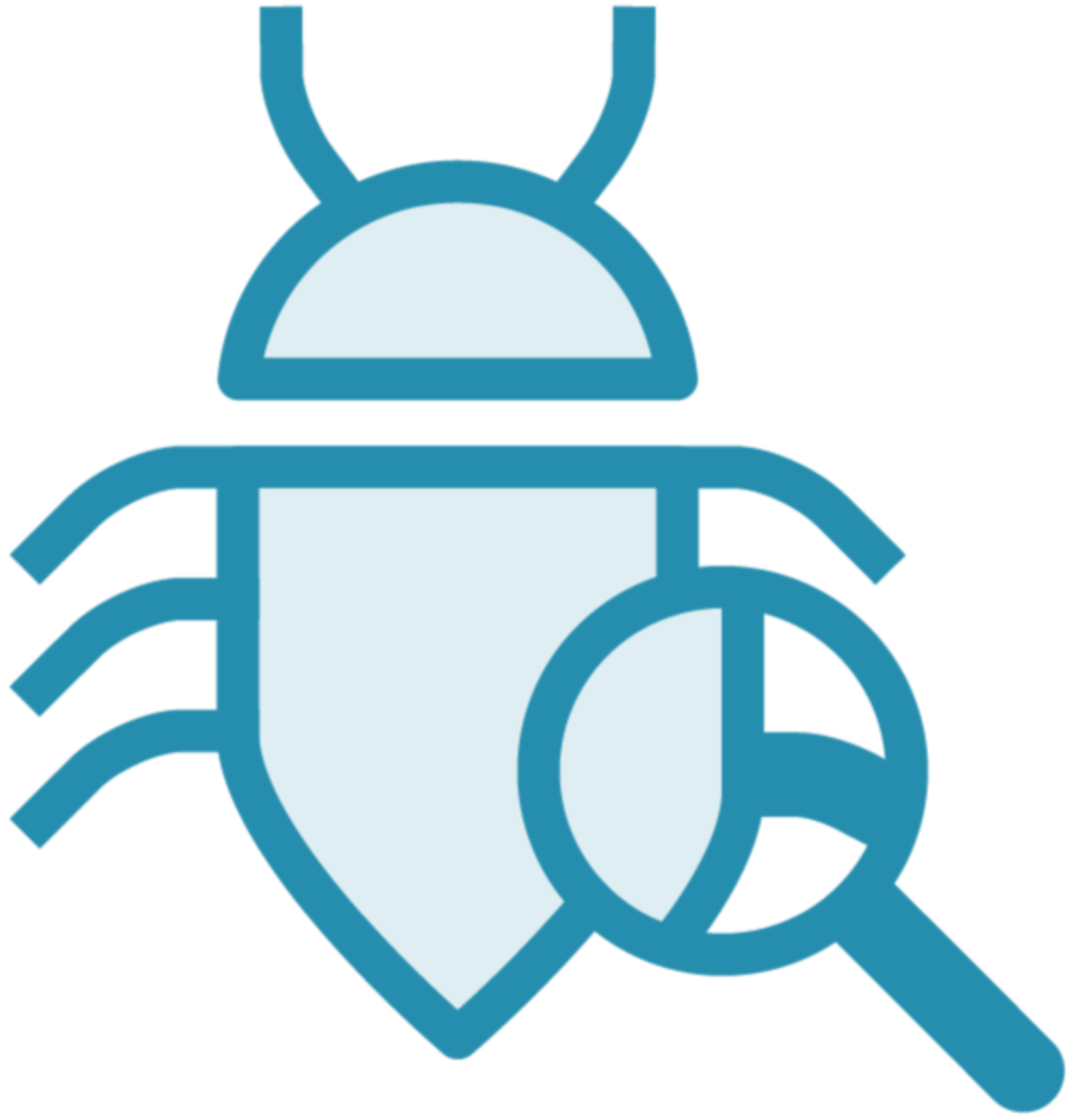
```
productSupplier$ = this.selectedProduct$
  .pipe(
    switchMap(productId =>
      this.http.get<Product>(`${this.productsUrl}/${productId}`)
    )
    switchMap(product =>
      this.http.get<Supplier>(`${this.suppliersUrl}/${product.supplierId}`)
    ),
    catchError(this.handleError)
  );
```

Breaking up Large Pipelines

```
selectedProduct$ = this.productSelectedAction$
  .pipe(
    switchMap(productId =>
      this.http.get<Product>(`${this.productsUrl}/${productId}`)
    )
    catchError(this.handleError)
  );
```

```
productSupplier$ = this.selectedProduct$
  .pipe(
    switchMap(product =>
      this.http.get<Supplier>(`${this.suppliersUrl}/${product.supplierId}`)
    ),
    catchError(this.handleError)
  );
```

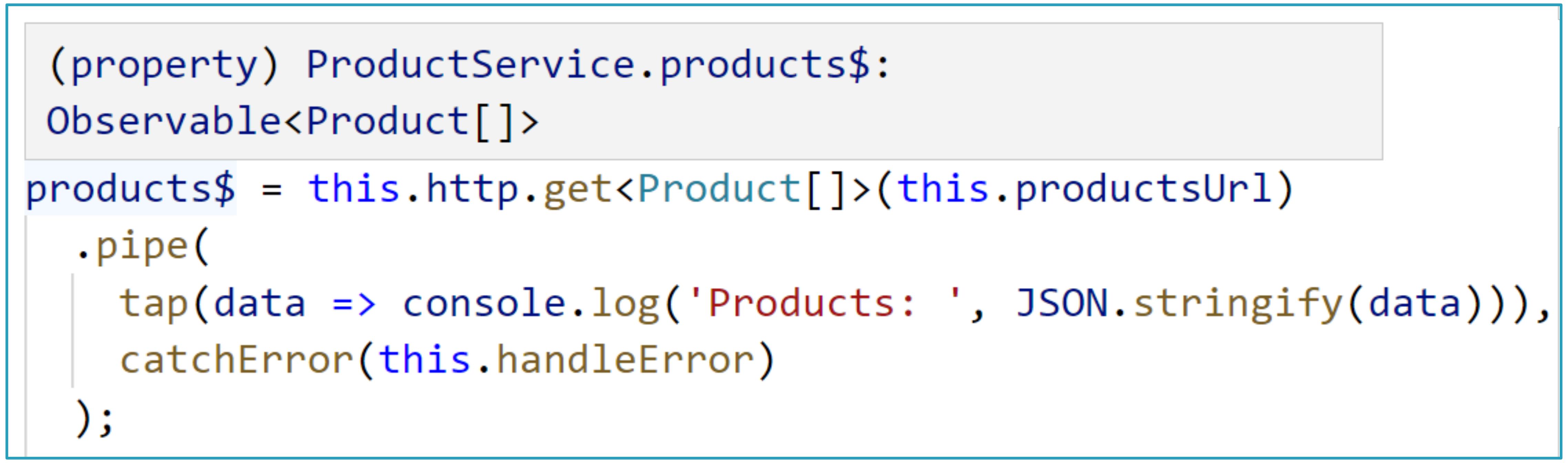
Debugging Observables



Use the `tap` operator

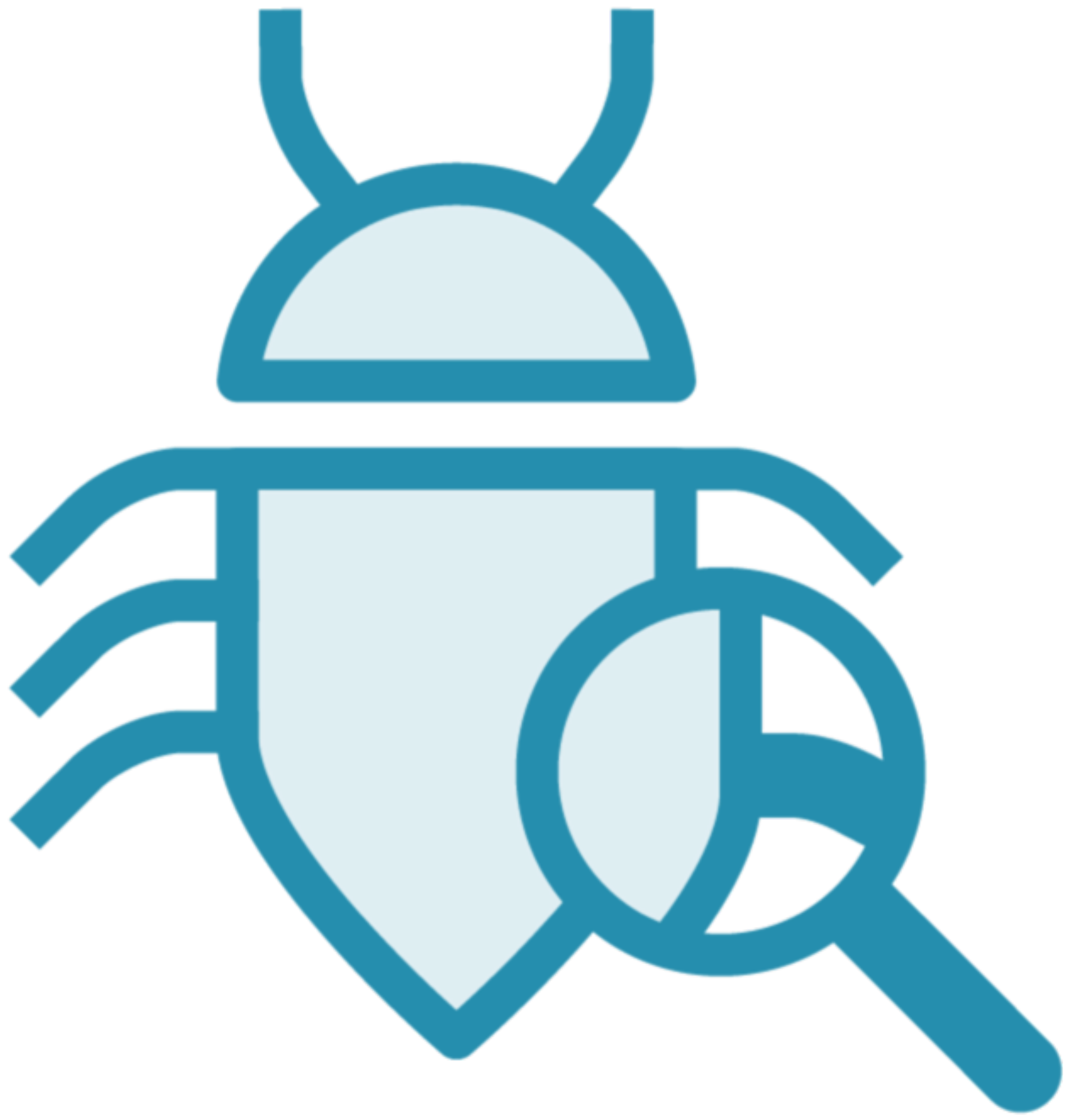
```
tap(data => console.log(JSON.stringify(data)))
```

Hover over the Observable to view the type



```
(property) ProductService.products$:
Observable<Product[]>
products$ = this.http.get<Product[]>(this.productsUrl)
  .pipe(
    tap(data => console.log('Products: ', JSON.stringify(data))),
    catchError(this.handleError)
  );
```

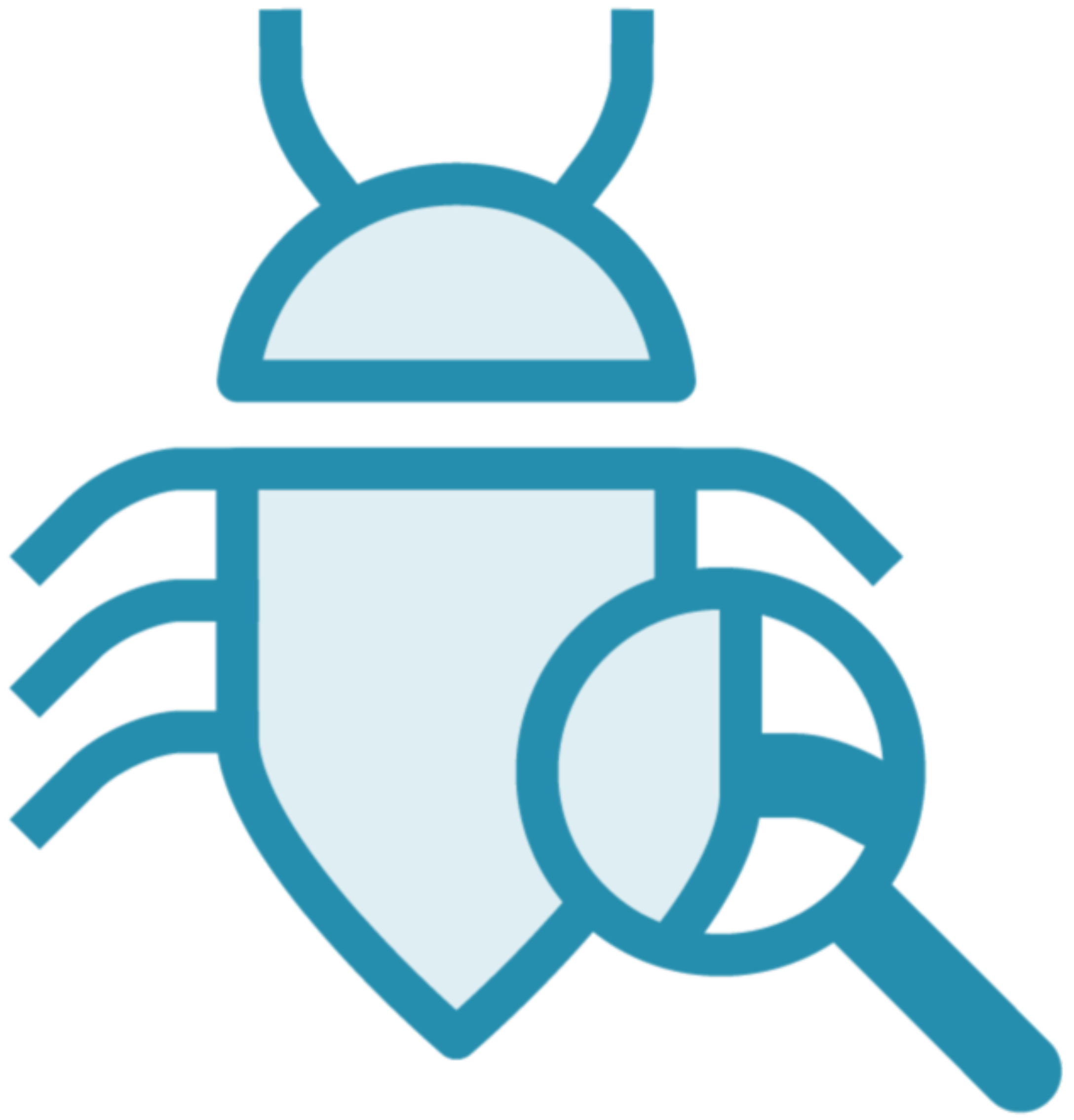
Common Debugging Issues



Is there a subscription?

Is there an operator waiting for completion?

Walk through the Flow

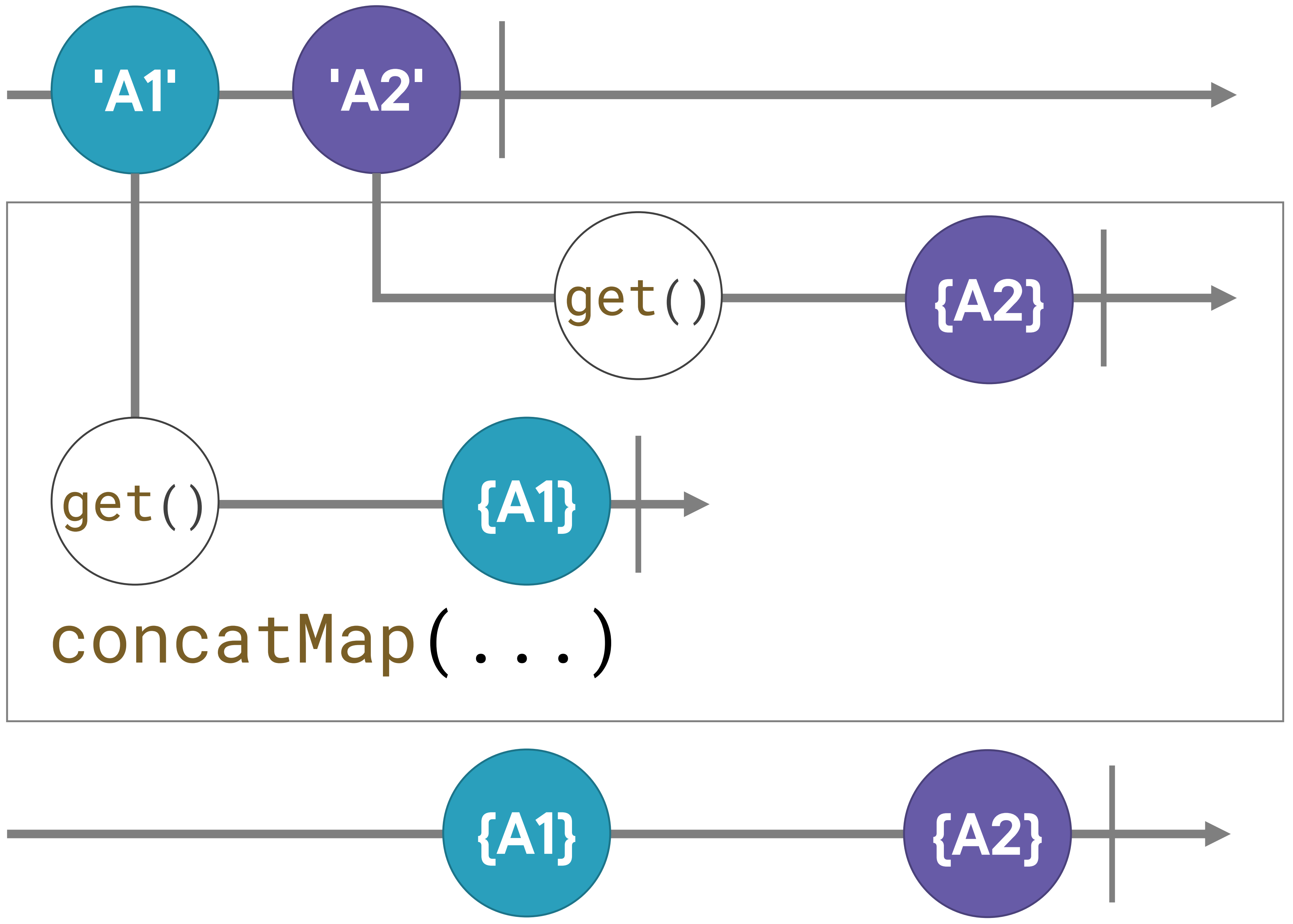


Start at the source (HTTP request)

Walk through each operator in the pipeline

Follow through to the UI

Draw a Picture



Goals for This Course

Add clarity

Examine reactive patterns

Improve state management

Merge RxJS streams

React to user actions

Minimize subscriptions

Improve UI performance

Learning More

Pluralsight courses

- Angular Component Communication
- Angular NgRx: Getting Started
- RxJS: Getting Started
- Learning RxJS Operators by Example Playbook

RxJS documentation

- rxjs.dev

@deborahkurata