

Reacting to Actions: Examples



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Products

Leaf Rake (Garden)

Garden Cart (Garden)

Hammer (Toolbox)

Saw (Toolbox)

Video Game Controller (Gaming)

Product Detail for: Hammer

Name: Hammer
Code: TBX-0048
Category: Toolbox
Description: Curved claw steel hammer
Price: \$13.35
In Stock: 8

Supplier	Cost	Minimum Quantity
Acme General Supply	\$2.00	24
Acme Tool Supply	\$4.00	12



Product List

- Display All - ▾

Add Product

Product	Code	Category	Price	In Stock
Leaf Rake	GDN-0011	Garden	\$29.92	15
Garden Cart	GDN-0023	Garden	\$49.49	2
Hammer	TBX-0048	Toolbox	\$13.35	8
Saw	TBX-0022	Toolbox	\$17.33	6
Video Game Controller	GMG-0042	Gaming	\$53.93	12



Module Overview



React to selections

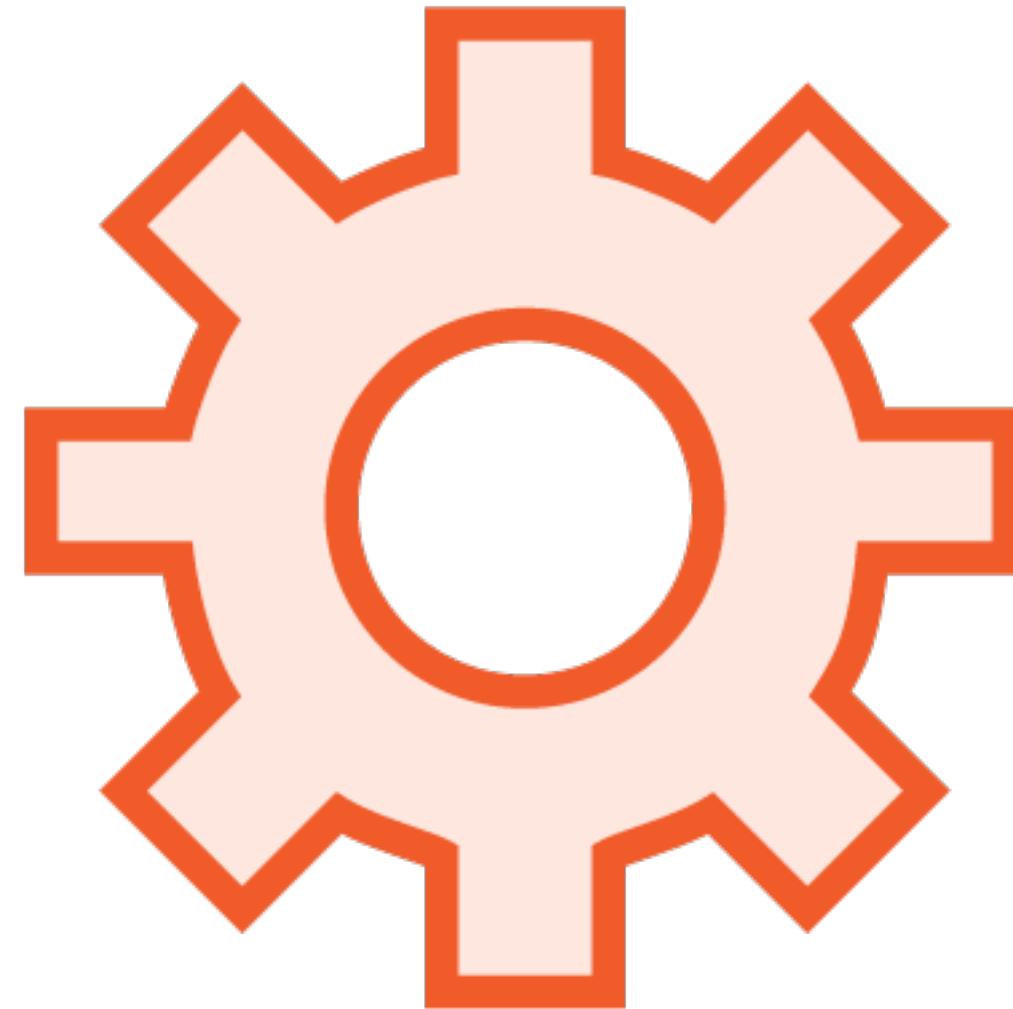
React to errors

Manage state with Observables

React to add operations



RxJS Features



merge
scan



Reacting to a Selection

Acme Product Management [Home](#) [Product List](#) [Product List \(Alternate UI\)](#)

Products

- Leaf Rake (Garden)
- Garden Cart (Garden)
- Hammer (Toolbox)**
- Saw (Toolbox)
- Video Game Controller (Gaming)

Product Detail for: Hammer

Name: Hammer
Code: TBX-0048
Category: Toolbox
Description: Curved claw steel hammer
Price: \$13.35
In Stock: 8

Supplier	Cost	Minimum Quantity
Acme General Supply	\$2.00	24
Acme Tool Supply	\$4.00	12



Demo



Reacting to a selection



Demo



Reacting to an error



Managing state
is an important part of any
application.



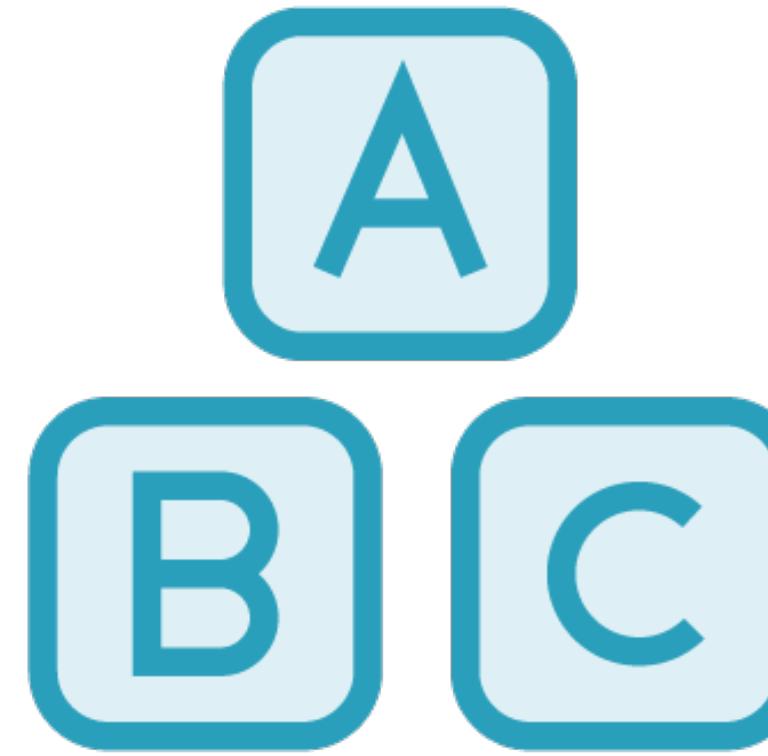
What Is State?



View State



User Information



Entity Data



**User Selection
and Input**



Read-only Data

```
*ngIf="products$ | async as products"
```

```
products$ = this.productService.products$  
  .pipe(  
    catchError(err => {  
      this.errorMessage = err;  
      return EMPTY;  
    })  
  );
```

```
products$ = this.http.get<Product[]>(this.productsUrl)  
  .pipe(  
    tap(data => console.log(JSON.stringify(data))),  
    catchError(this.handleError)  
  );  
}
```



Updating Data

Let the backend handle it

Treat the data as read-only

Issue a PUT, POST, or DELETE

GET to get the current data

Keeps the data "fresh"

Could have performance impacts

Let our Observable handle it

Define an action for update operations

On each update:

- Issue a PUT, POST, or DELETE

- Incorporate the change

- UI automatically updates



Incorporate a Change in an Observable?

```
products: Product[] = [];

ngOnInit(): void {
  this.sub = this.productService.getProducts()
    .subscribe({
      next: products => this.products = products,
      error: err => this.errorMessage = err
    });
}
```

```
products$ = this.productService.products$
  .pipe(
    catchError(err => {
      this.errorMessage = err;
      return EMPTY;
    })
  );
```





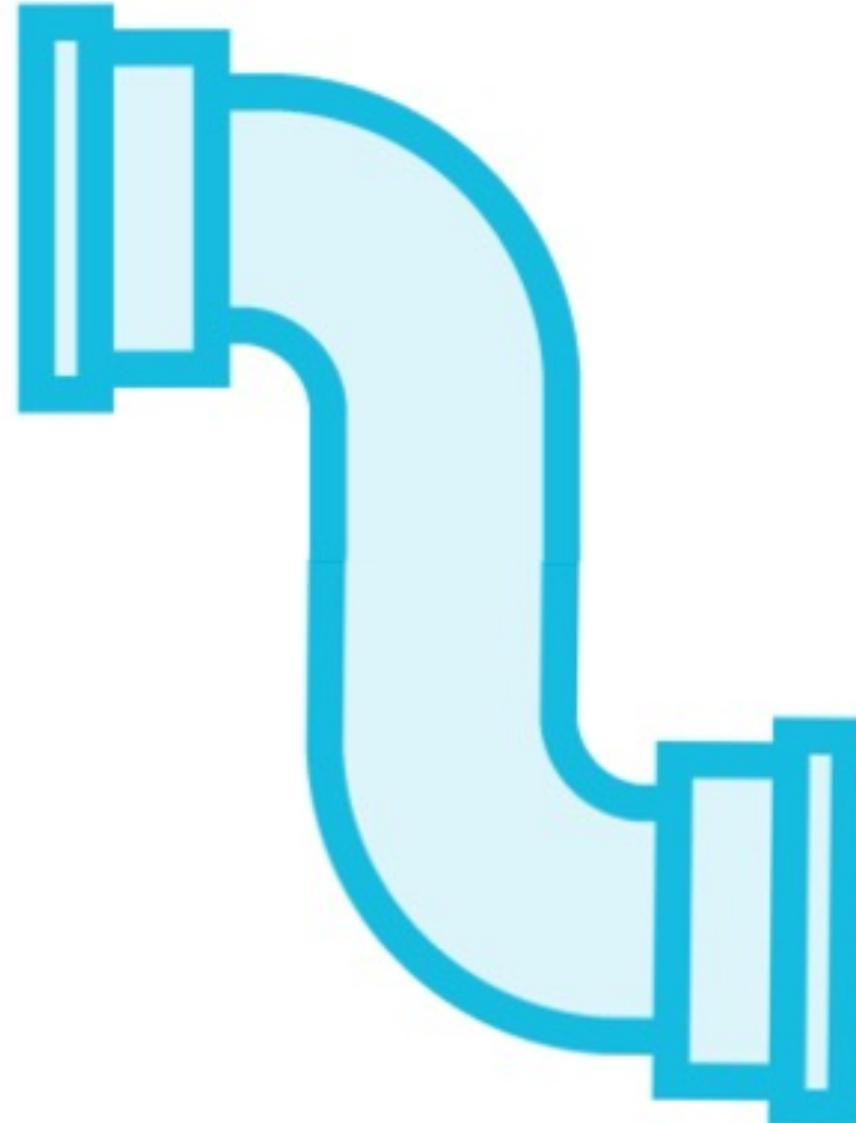
Scan: A key operator when managing state

Retains an accumulated value

- Sum of values
- Array of items



RxJS Operator: scan



Accumulates items in an Observable

```
scan((acc, curr) => acc + curr)
```

For each emitted item

- The accumulator function is applied
- The result is buffered and emitted

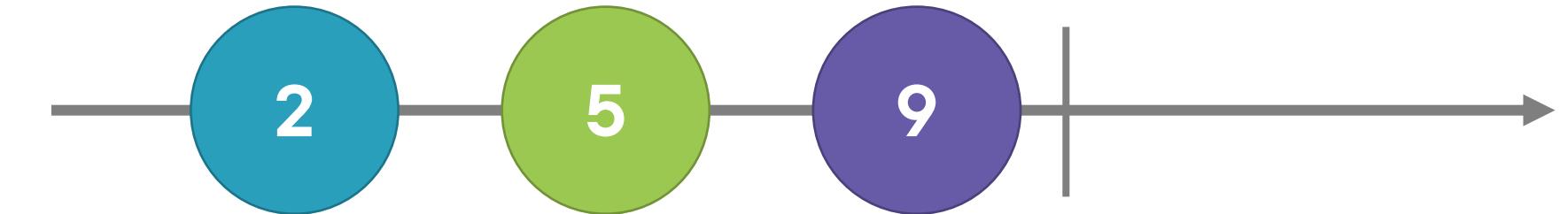
Used for

- Encapsulating and managing state
- Totaling amounts
- Accumulating items into an array

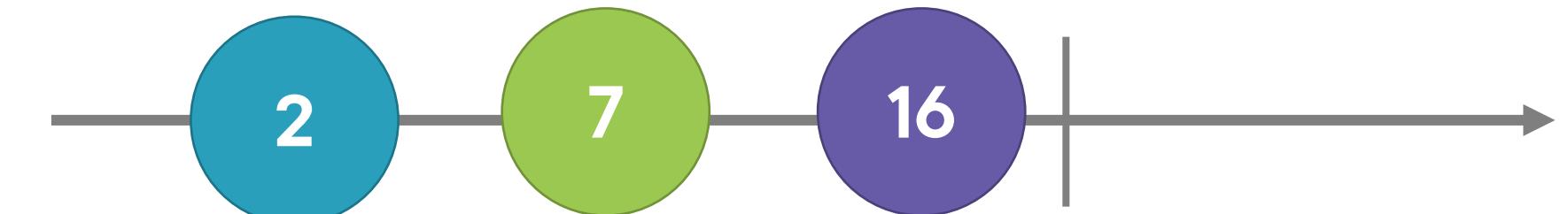


Marble Diagram: scan

```
of(2, 5, 9)
  .pipe(
    scan((acc, curr) => acc + curr)
  )
  .subscribe(x => console.log(x));
// 2, 7, 16
```



scan((acc, curr) => acc + curr)



Initial State

```
of(2, 5, 9)
  .pipe(
    scan((acc, curr) => acc + curr, 10)
  )
  .subscribe(x => console.log(x));
// 12, 17, 26
```

Uses the provided seed value as the initial state

```
of(2, 5, 9)
  .pipe(
    scan((acc, curr) => acc + curr)
  )
  .subscribe(x => console.log(x));
// 2, 7, 16
```

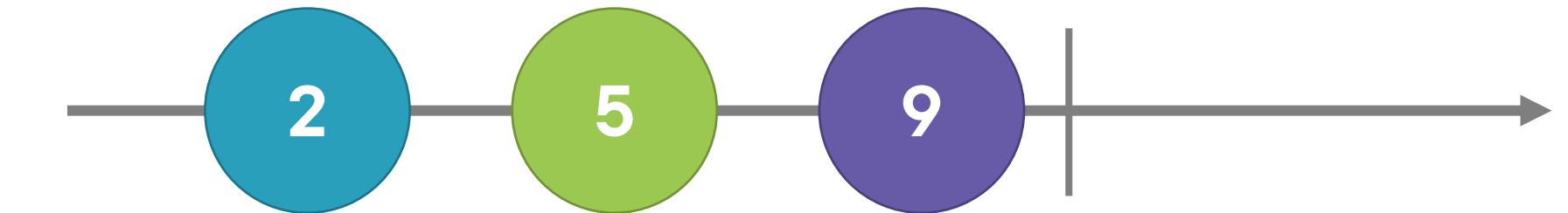
If no seed value is provided, uses the first value from the source as the initial state

That first value is emitted without going through the accumulator function

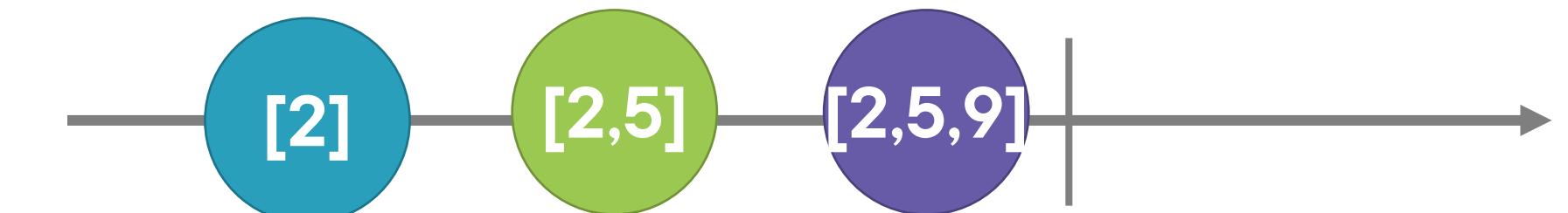


Marble Diagram (array): scan

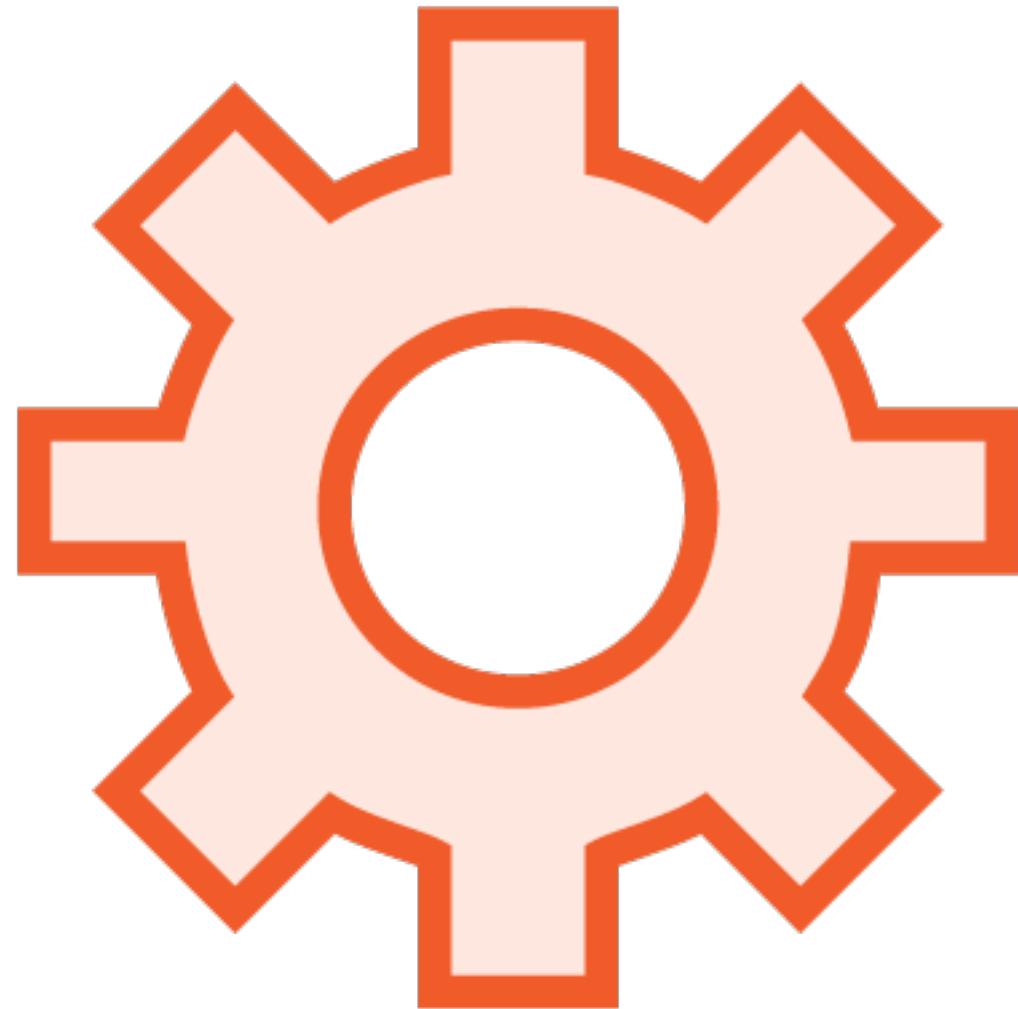
```
of(2, 5, 9)
  .pipe(
    scan((acc, curr) =>
      [...acc, curr], [] as number[])
  )
  .subscribe(x => console.log(x));
// [2], [2,5], [2,5,9]
```



```
scan((acc, curr) =>
  [...acc, curr], [] as number[])
```



RxJS Operator: scan



scan is a transformation operator

- Subscribes to its input Observable
- Creates an output Observable

Seed, if defined, is used as the initial state

Otherwise, the first emitted value is used as the initial state

Once initial state is set, when an item is emitted

- Item is accumulated as specified by the provided accumulator function
- Result is emitted to the output Observable



RxJS Creation Function: merge



Combines multiple Observables by merging their emissions

```
merge( a$, b$, c$ )
```

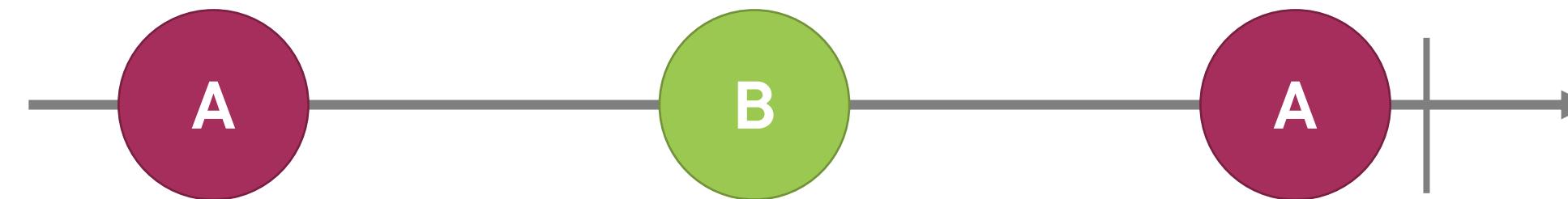
Static creation function, not a pipeable operator

Used for

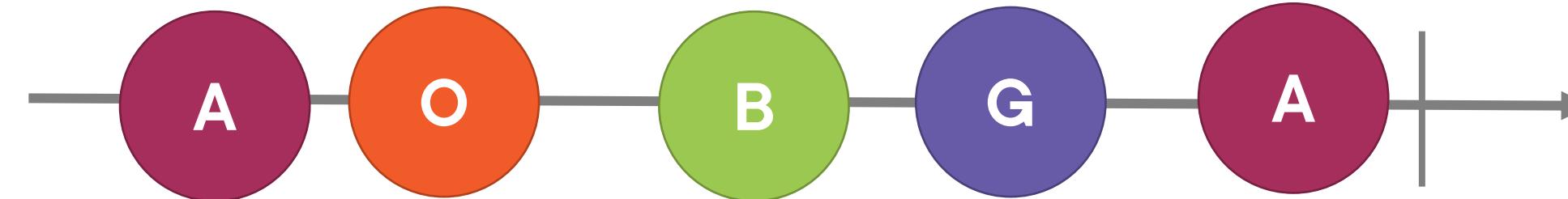
- Combining sequences of similar types to blend their emitted values



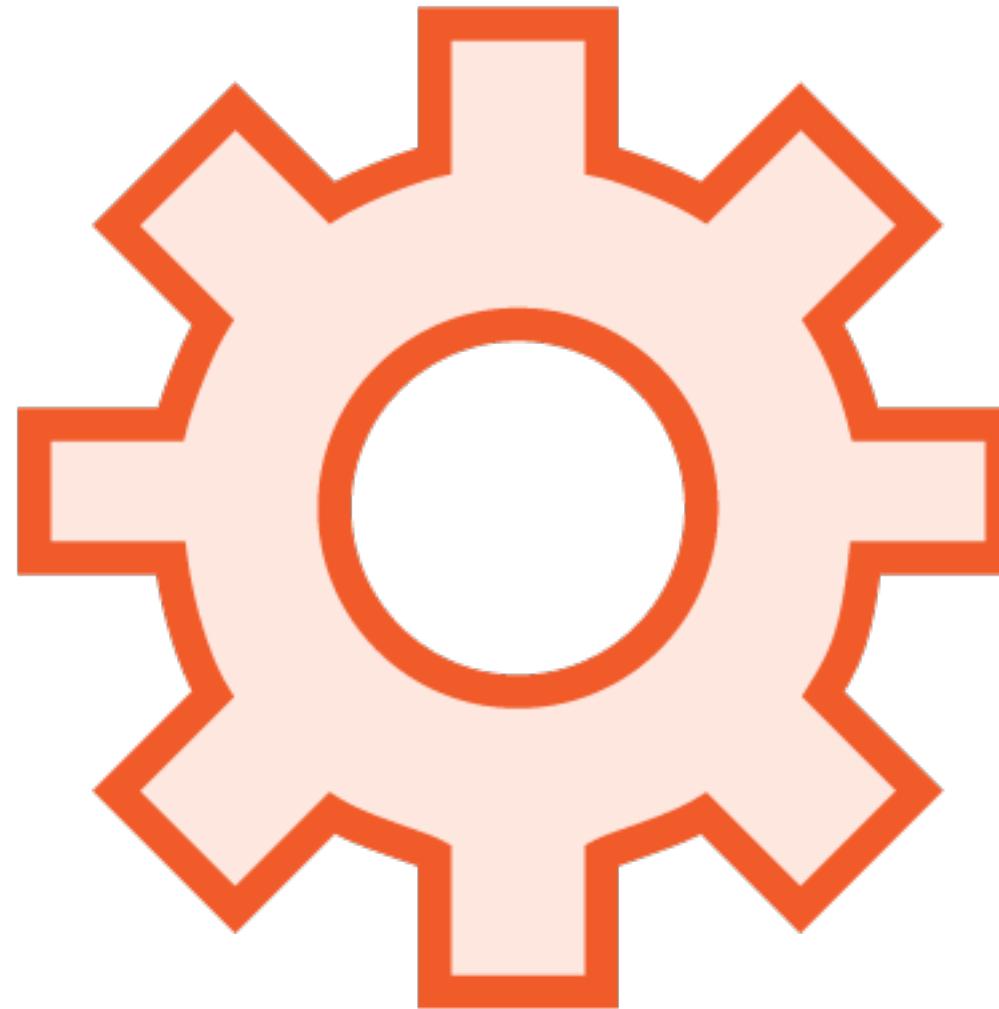
Marble Diagram: merge



merge(...)



RxJS Creation Function: merge



merge is a combination function

- Takes in a set of Observables, subscribes
- Creates an output Observable

When an item is emitted from any Observable

- Item is emitted to the output Observable

Completes when all input Observables complete



Reacting to an Add Operation

Acme Product Management [Home](#) [Product List](#) [Add Product](#)

Add Product

Product Name

Product Code

Star Rating (1-5)

Tag Delete Tag

Add Tag

Description

Save Cancel Delete



Reacting to an Add Operation

Acme Product Management [Home](#) [Product List](#) [Product List \(Alternate UI\)](#)

Product List

- Display All - ▾ Add Product

Product	Code	Category	Price	In Stock
Leaf Rake	GDN-0011	Garden	\$29.92	15
Garden Cart	GDN-0023	Garden	\$49.49	2
Hammer	TBX-0048	Toolbox	\$13.35	8
Saw	TBX-0022	Toolbox	\$17.33	6
Video Game Controller	GMG-0042	Gaming	\$53.93	12



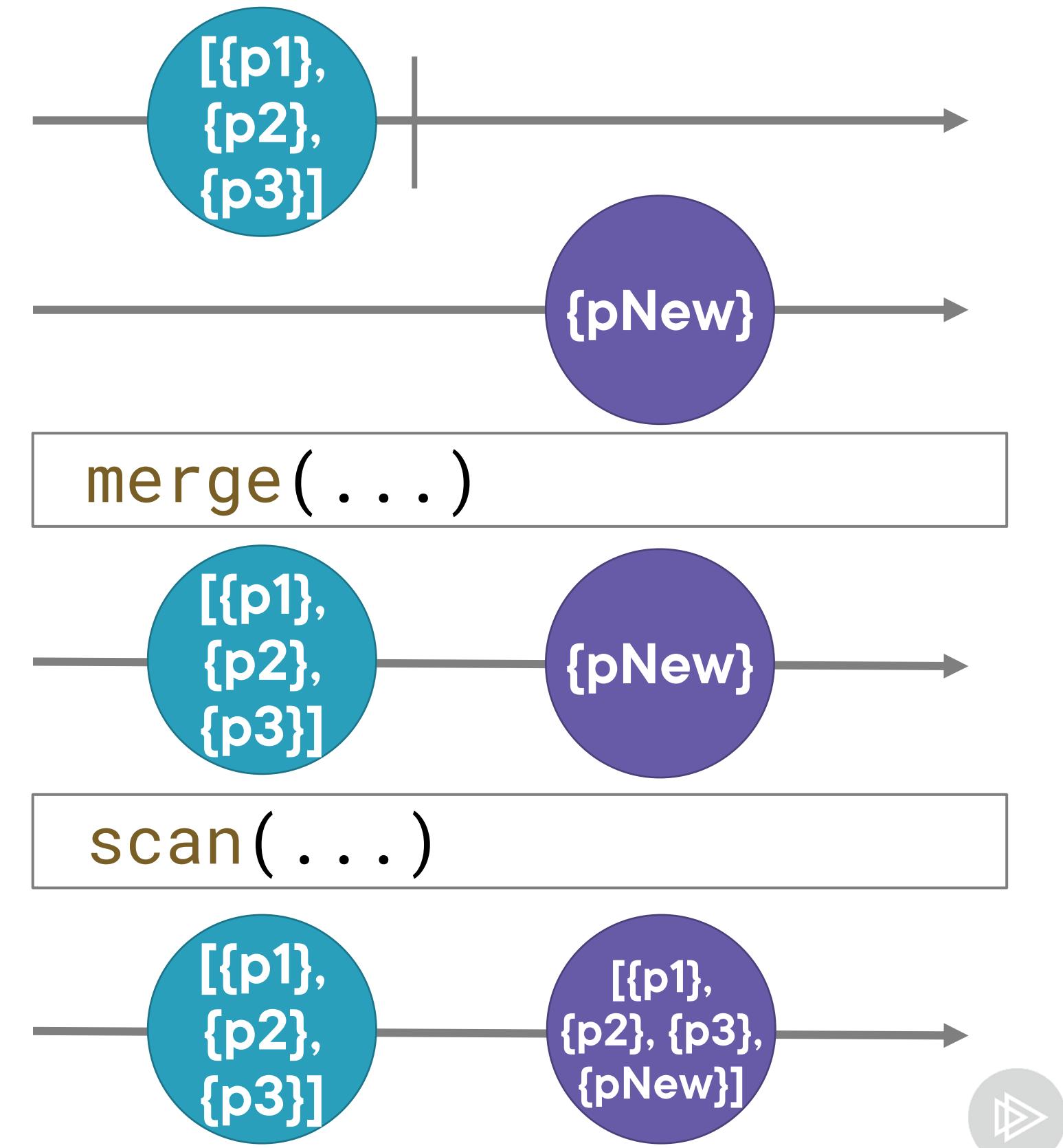
Reacting to an Add Operation

```
merge(  
  this.products$,  
  this.insertAction$  
)  
.pipe(  
  scan((acc, value) =>  
    (value instanceof Array) ?  
      [...value] : [...acc, value],  
      [] as Product[])  
)
```



Reacting to an Add Operation

```
merge(  
  this.products$,  
  
  this.insertAction$  
)  
  
.pipe(  
  scan((acc, value) =>  
    (value instanceof Array) ?  
      [...value] : [...acc, value],  
    [] as Product[])  
)
```



Demo



Reacting to an add operation



RxJS Checklist: Reacting to Actions



Create an action stream (Subject/BehaviorSubject)

```
private actionSubject = new Subject<number>();  
action$ = this.actionSubject.asObservable();
```

Combine the action and data streams

```
products$ = combineLatest([  
  this.productService.products$,  
  this.action$  
]).pipe(...);
```

Emit a value to the action stream when an action occurs

```
onSelected(categoryId: string): void {  
  this.actionSubject.next(+categoryId);  
}
```



RxJS Checklist: Reacting to a Selection



```
private pSelSubject = new BehaviorSubject<number>(0);
pSelAction$ = this.pSelSubject.asObservable();

selectedProduct$ = combineLatest([
  this.products$,
  this.pSelAction$
]).pipe(
  map(([products, selectedProductId]) =>
    products.find(product => product.id === selectedProductId)
  )
);
```

```
selectedProductChanged(selectedProductId: number): void {
  this.pSelSubject.next(selectedProductId);
}
```



RxJS Checklist: Reacting to an Error



```
private errorSubject = new Subject<string>();
error$ = this.errorSubject.asObservable();

product$ = this.productService.selectedProduct$
  .pipe(
    catchError(err => {
      this.errorSubject.next(err);
      return EMPTY;
    })
  );
```

```
<div *ngIf="error$ | async as errorMessage">
  {{ errorMessage }}
</div>
```



RxJS Checklist: Features



merge: Merges the emissions of multiple Observables

```
merge(a$, b$, c$)
```

scan: Applies an accumulator function

```
scan((acc, curr) => acc + curr)
```



RxJS Checklist: Reacting to an Add Operation



```
merge(  
  this.products$,  
  this.insertAction$  
)  
.pipe(  
  scan((acc, value) =>  
    (value instanceof Array) ? [...value] : [...acc, value],  
    [] as Product[])  
)
```



RxJS Checklist: Reacting to an Add Operation



```
merge(
  this.products$,
  this.insertAction$
    .pipe(
      concatMap(newProd => {
        return this.http.post<Product>(this.url, newProd)
      }),
    )));
  .pipe(
    scan((acc, value) =>
      (value instanceof Array) ? [...value] : [...acc, value],
      [ ] as Product[])
  );
}
```





Coming up next...

Caching Observables

