Tackling Component State Reactively

“If you stick to the paradigms of OOP the design **patterns appear naturally**”

Gang Of Four

Table of content

Layers of State

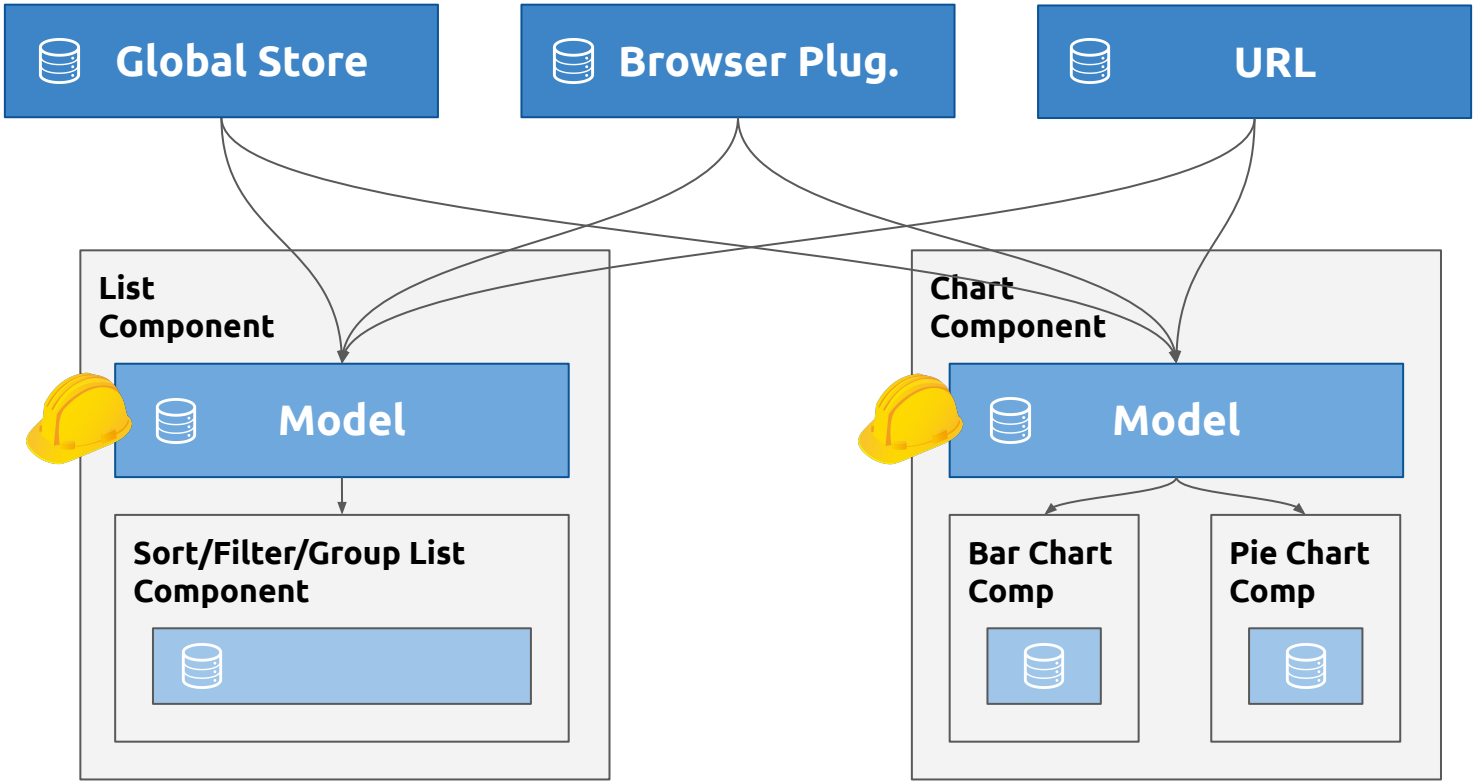
Server State

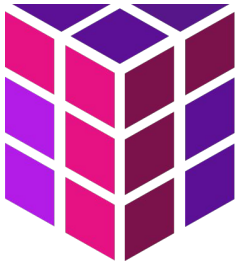
Client State

Persistent State

Ephemeral State

What is Ephemeral State?





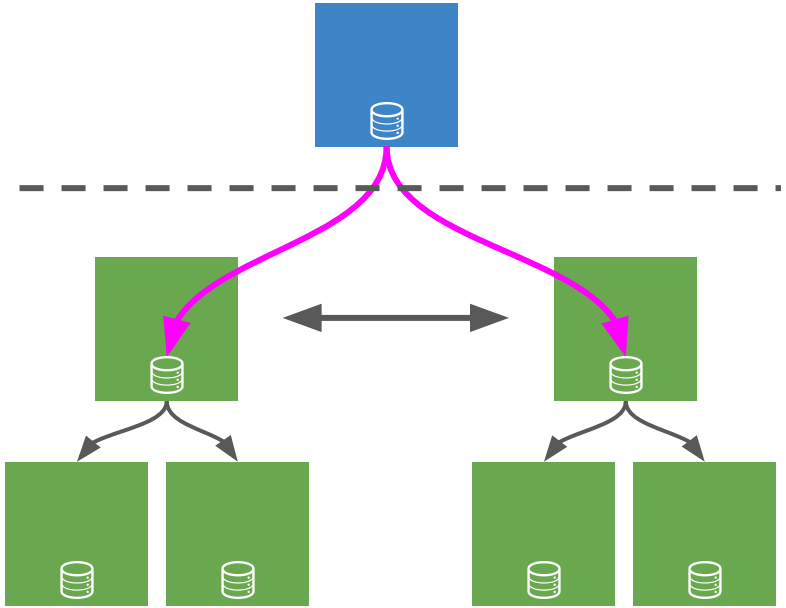
TRILON.

I’m Michael Hladky.
Lead Instructor at trilon.io

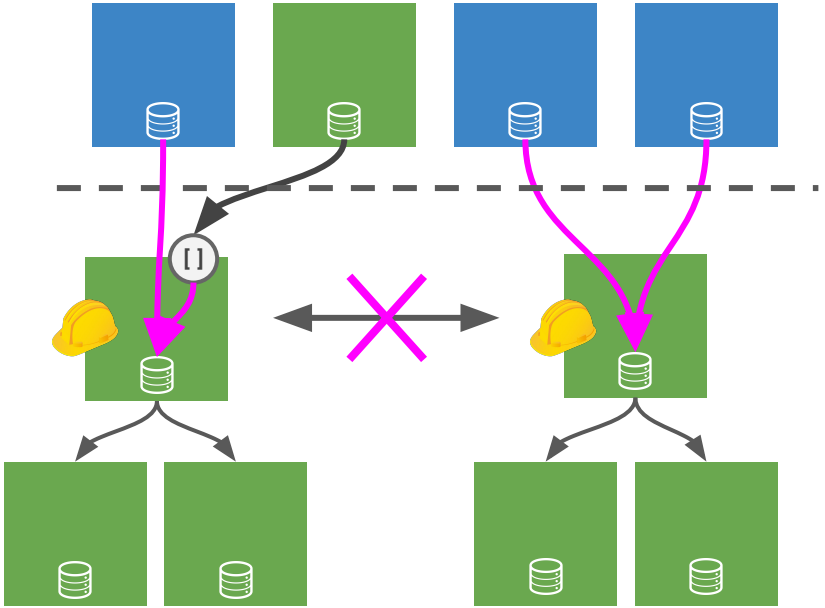
Terminology and Categorisation

Accessibility

Globally

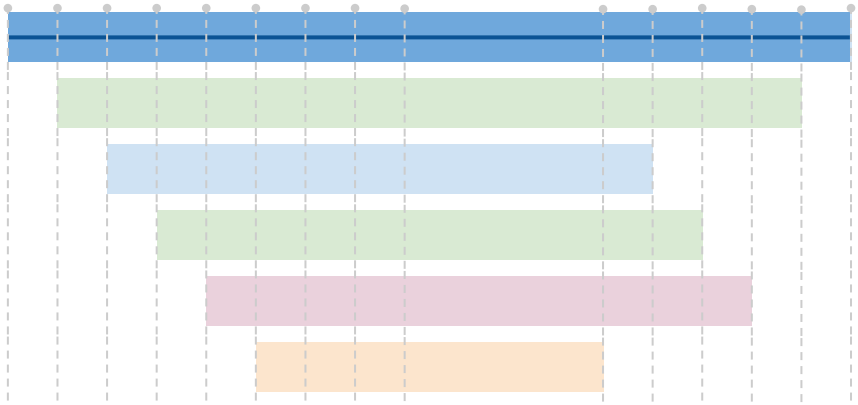


Locally

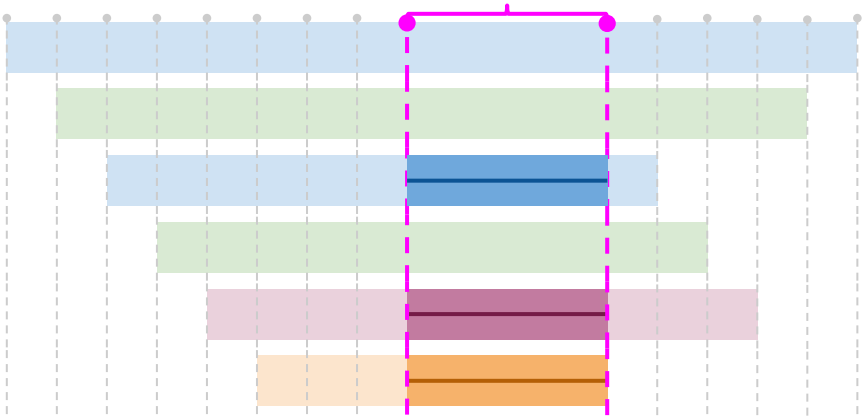


LifeTime

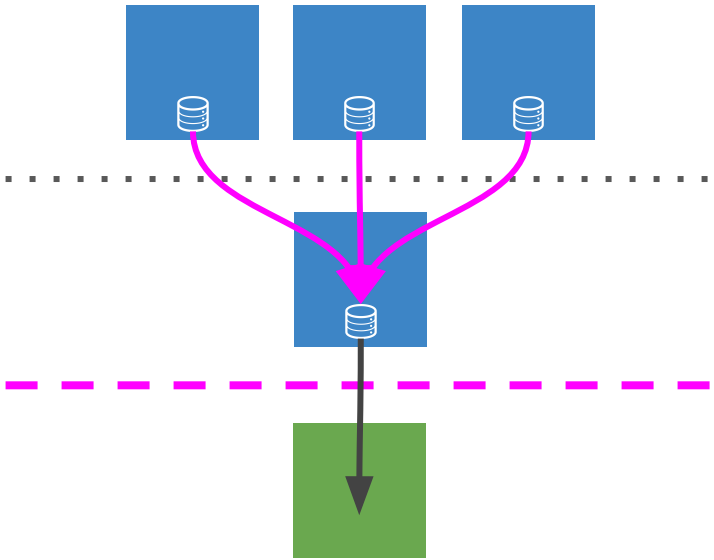
Static



Dynamic

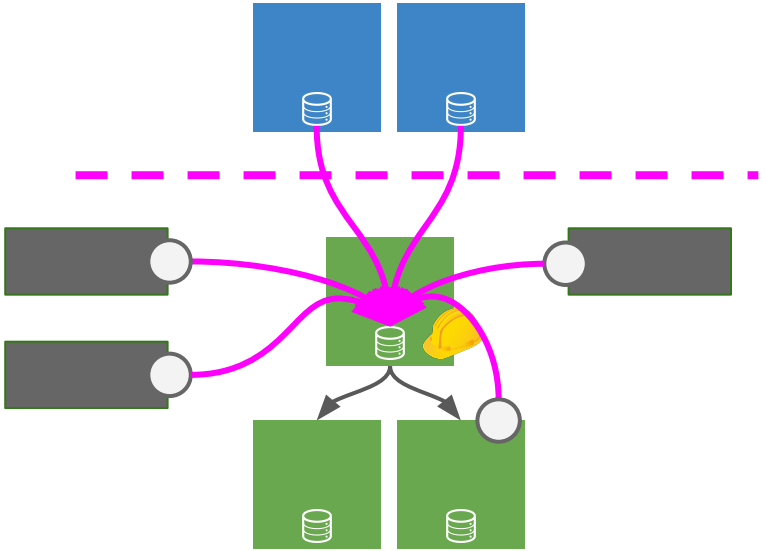


Processed Sources



Global

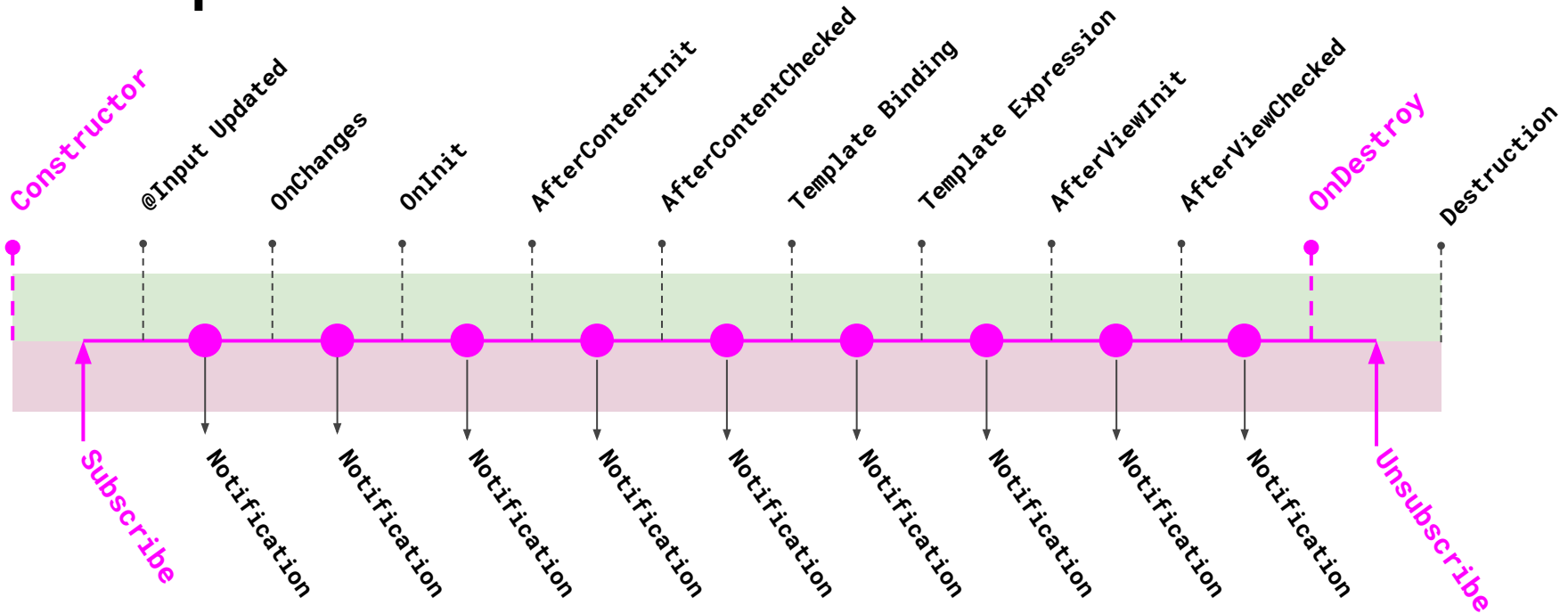
Local



Subscription Handling By Lifetime

Where to un/subscribe?

Component



Observable

Subscription Handling via Component Providers

subscription-handling.service.ts

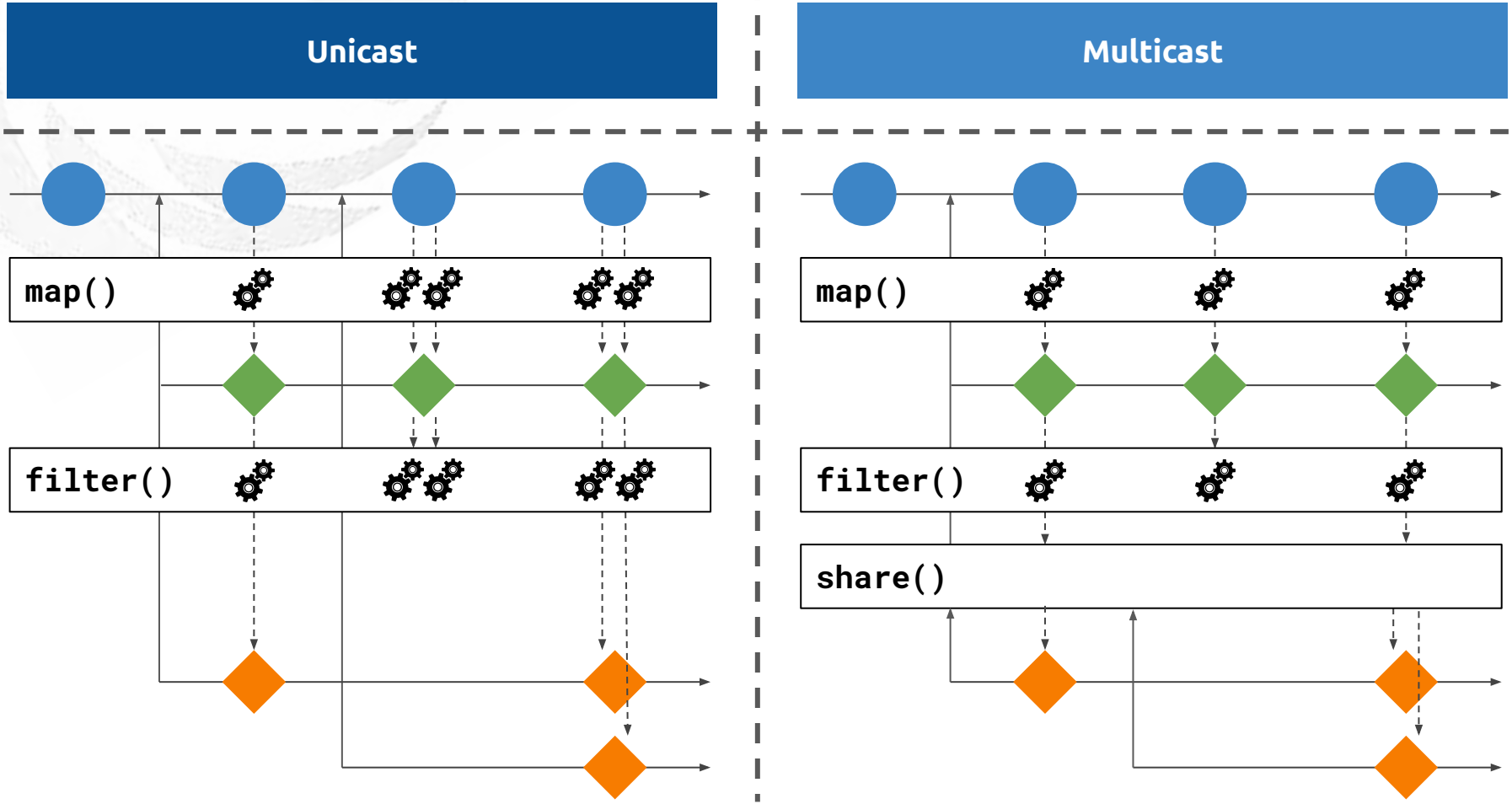
```
export class Service implements OnDestroy {

 onDestroy$ = new Subject();

 hold(o): void {
   o.pipe(takeUntil(this.onDestroy$))
      .subscribe()
 }

 ngOnDestroy(): void {
   this.onDestroy$.next();
 }

}
```

subscription-handling.component.ts

```
@Component({
 selector: 'app-subscription',
 template: `...`,
 providers: [Service]
})
export class Component {

 sideEffect$ = anySource$;

 constructor(private subHandler:Service) {
   this.subHandler
      .hold(this.sideEffect$)
 }

}
```

Demo Time!

State Selections and Memoistion
What to share?


Unicast
Multicast
map()
filter()
map()
filter()
share()

Unicast

uni-cast-instance.ts

```
const form$ = of(formConfig)
     .pipe(
          map(FormBuilder.group)
     );

form$
.subscribe(createInstance());
form$
.subscribe(createInstance());
```

Multicast

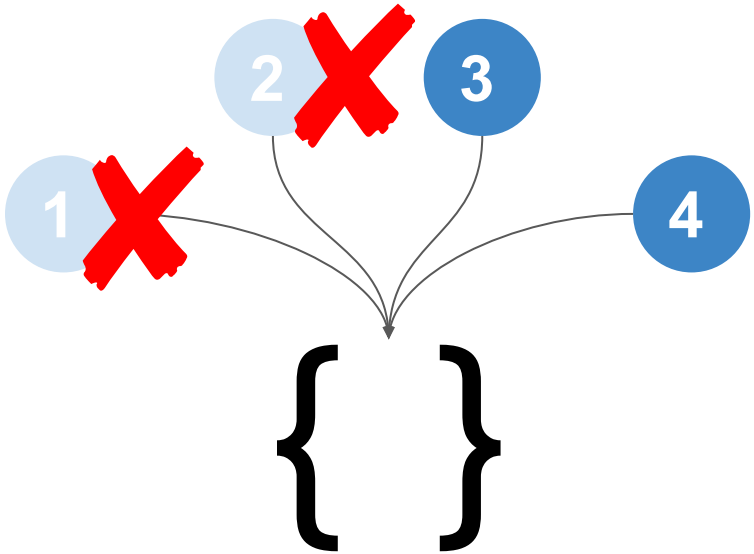
multi-cast-instance.ts

```
const form$ = of(formConfig)
     .pipe(
          map(FormBuilder.group),
          shareReplay(1)
     );
form$
.subscribe(reuseInstance());
form$
.subscribe(reuseInstance());
```

Demo Time!

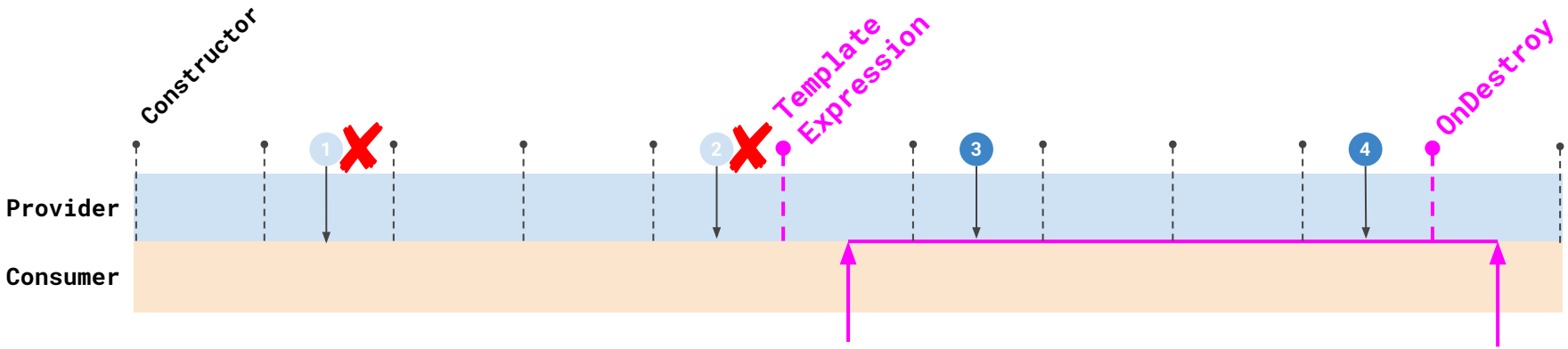
State Composition is cold by default!

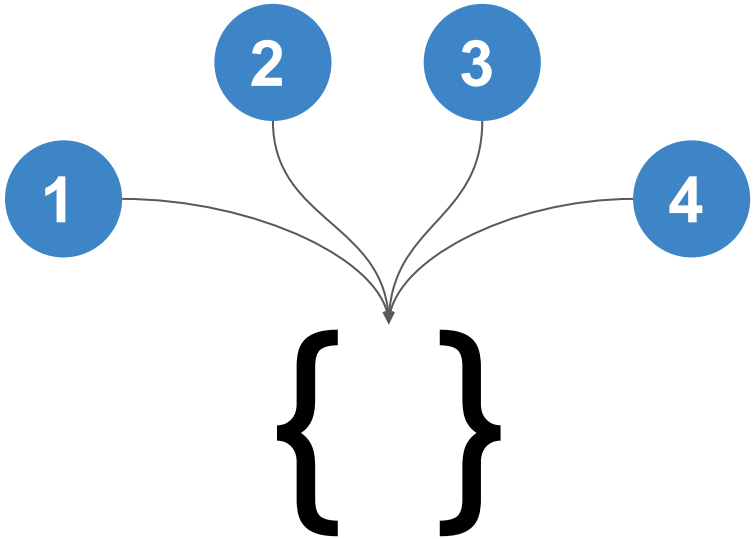
We rely on the consumer to start it!



composition.ts

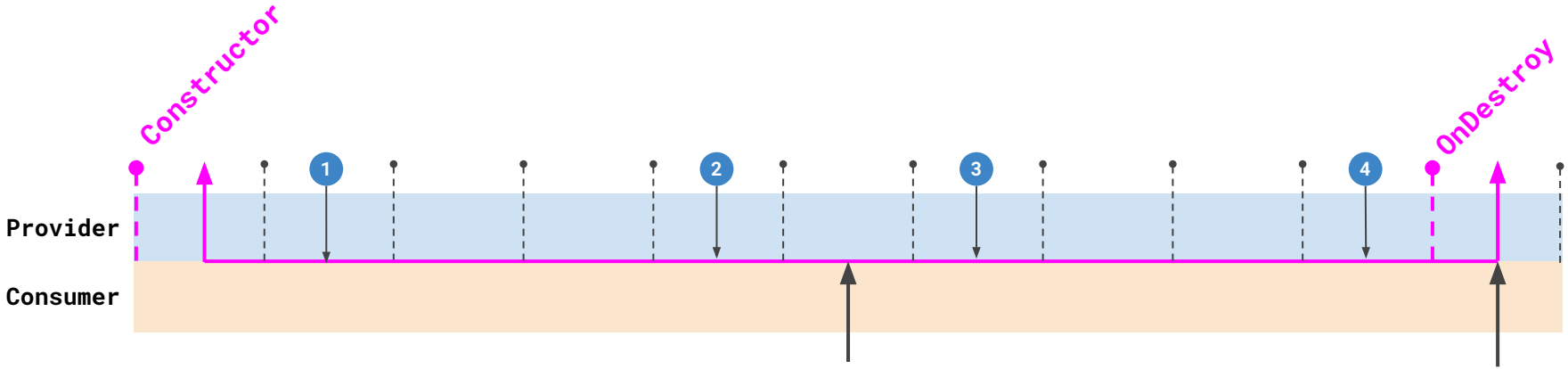
```ts
export class Service {
 state$ = slices$.pipe(
     scan(accumulator),
     shareReplay({bufferSize:1,refCount:})
 )

}
```





composition.ts

```ts
export class Service {
 state$ = slices$.pipe(
     scan(accumulator),
     publishReplay(1)
 )
 state$.connect();
}
```



Demo Time!

Subscription-Less Interaction
with Component-State

Problem

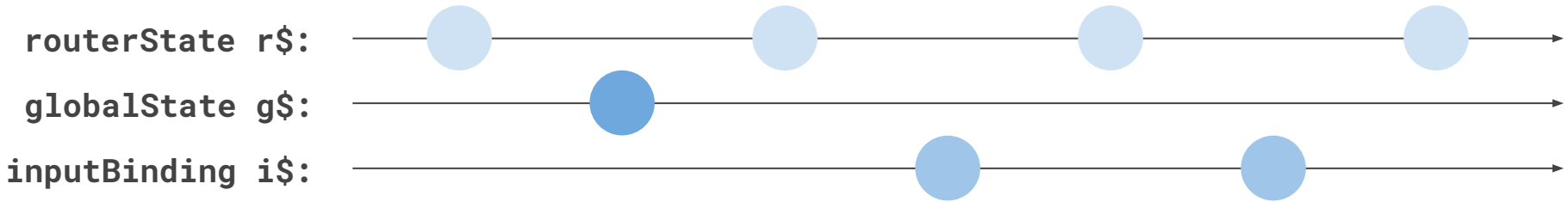
Setters are not composable

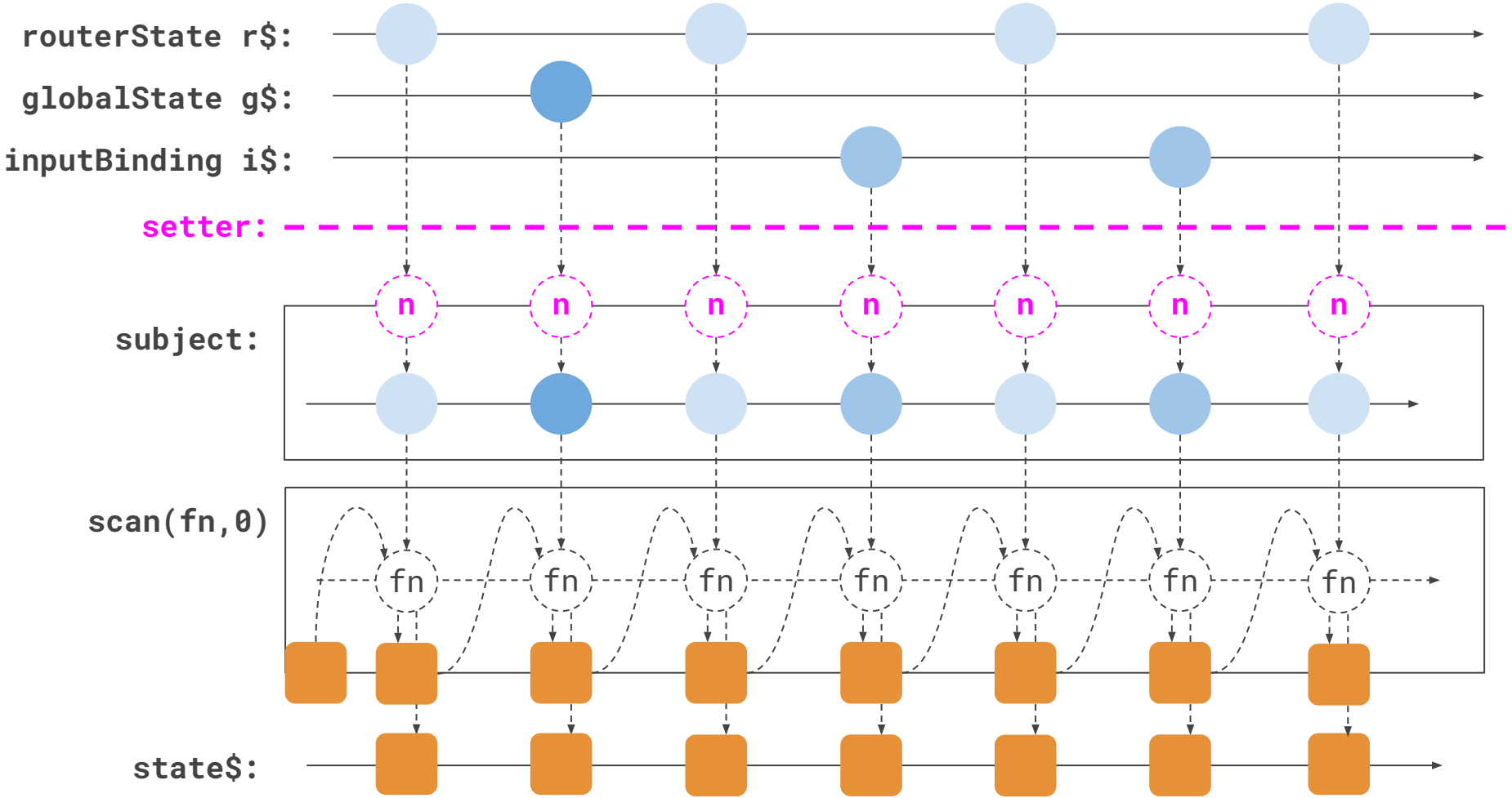
setState.service.ts

```ts
subscription:Subscription;

_state$ = new Subject();
state$ = _state$.pipe(scan(fn));

setState(slice) { _state$.next(slice) }
```

setState.component.ts

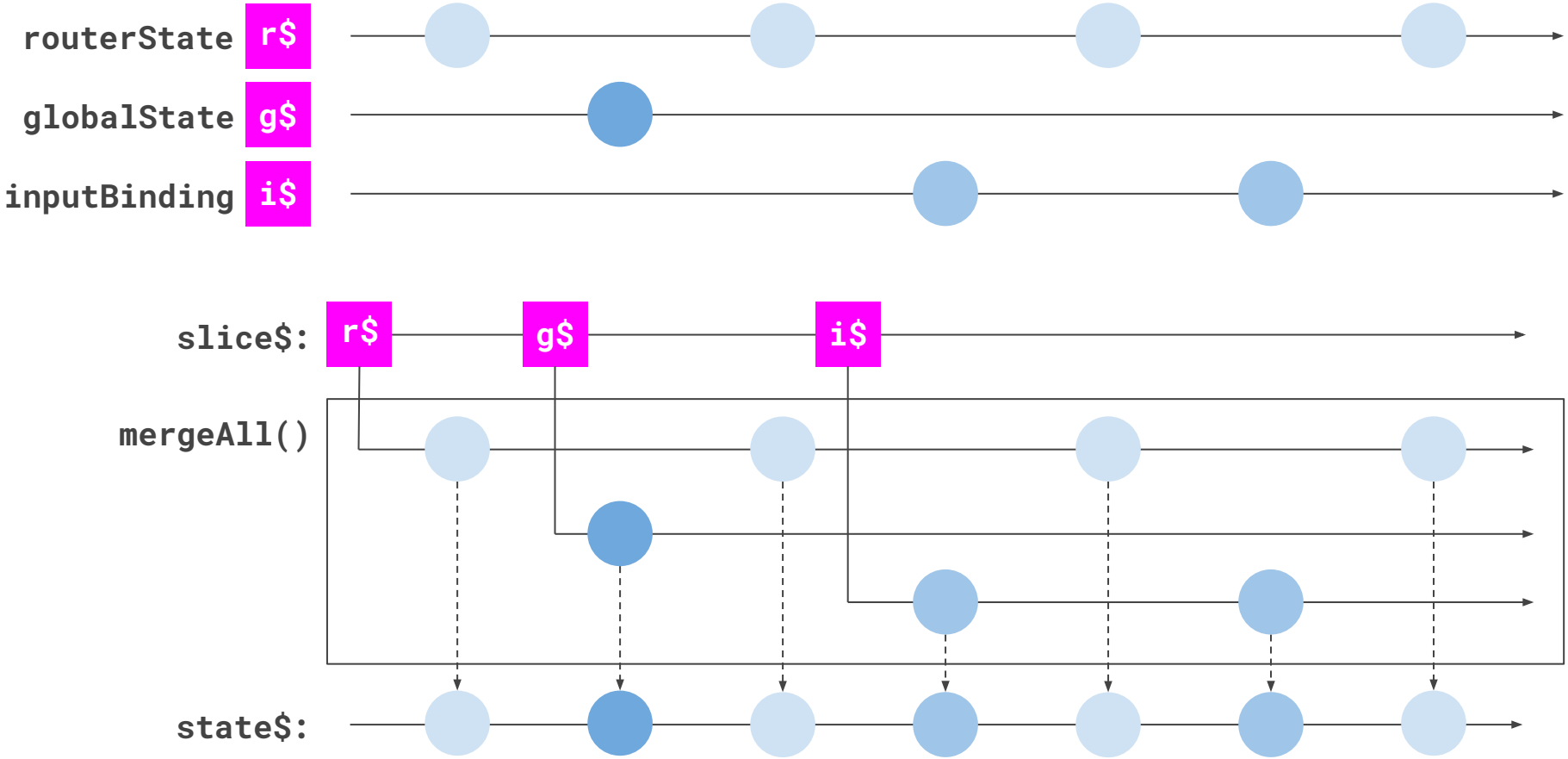
```ts
routerState$
  .pipe(takeUntil(destroy$))
  .subscribe(slice => setState(slice));

globalState$
  .pipe(takeUntil(destroy$))
  .subscribe(slice => setState(slice));

inputBinding$
  .pipe(takeUntil(destroy$))
  .subscribe(slice => setState(slice));
```

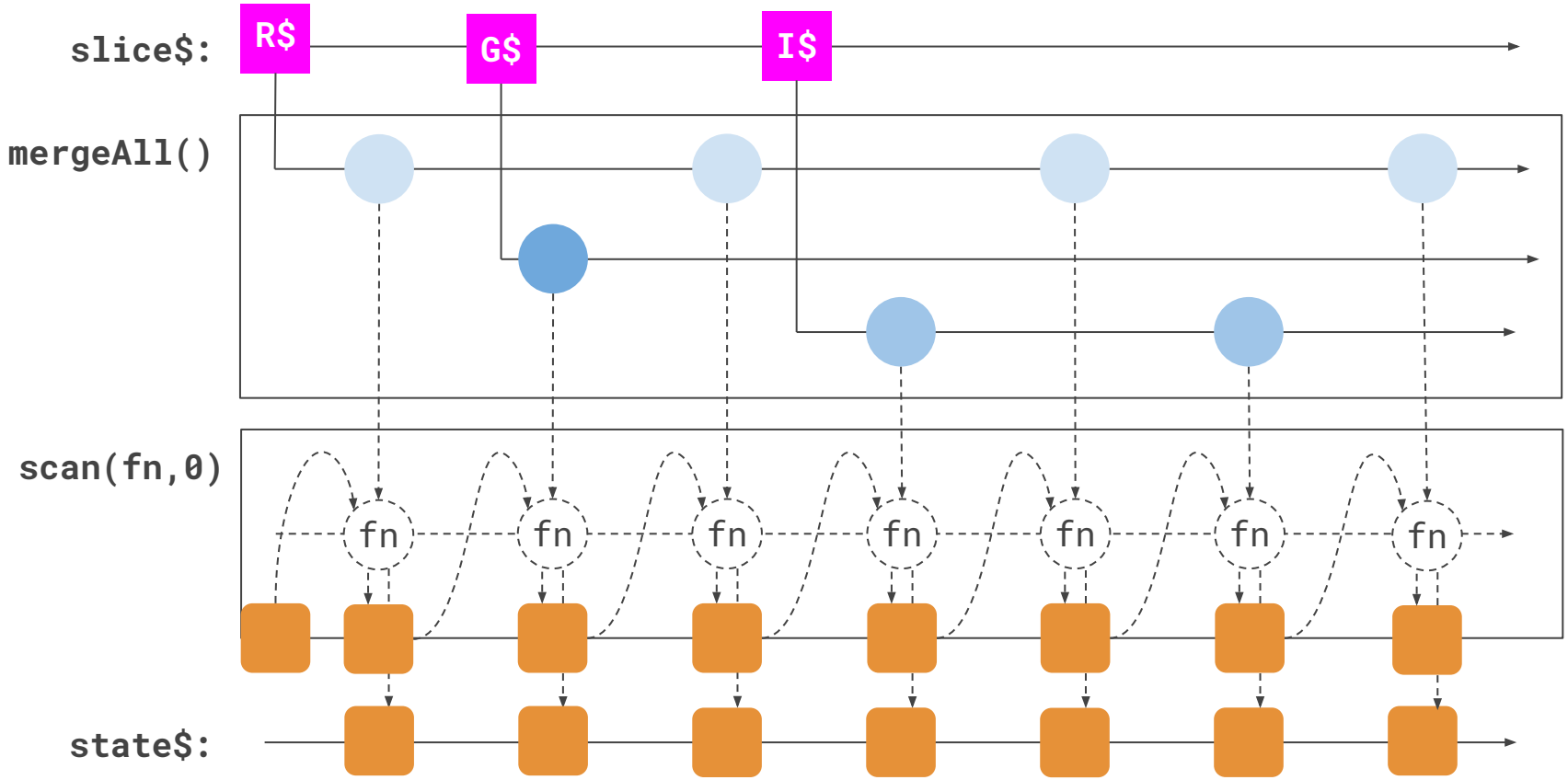



routerState r$:
globalState g$:
inputBinding i$:
setter:
subject:
n
scan(fn,0)
fn
state$:

Solution

Use Higher Order Operators


routerState
r$
globalState
g$
inputBinding
i$
slice$:
r$
g$
i$
mergeAll()
state$:

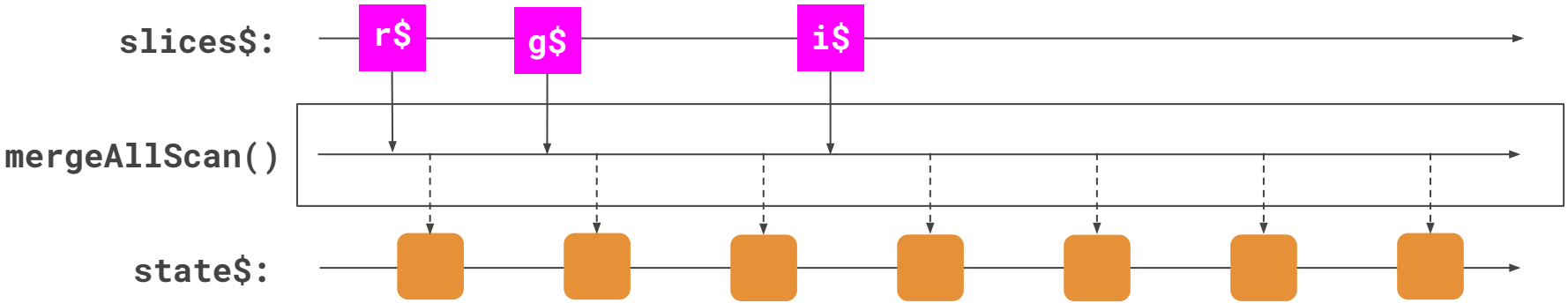

slice$:
R$
G$
I$
mergeAll()
scan(fn,0)
fn
fn
fn
fn
fn
fn
fn
state$:

connectState.service.ts

```
subscription:Subscription;

_state$ = new Subject();
state$ = _state$.pipe(
mergeAll(),scan(fn));

connectState(slice$){
  _state$.next(slice$)
}
```

connectState.component.ts

```
connectState(routerState$);

connectState(globalState$);

connectState(inputBinding$);
```



Demo Time!

“

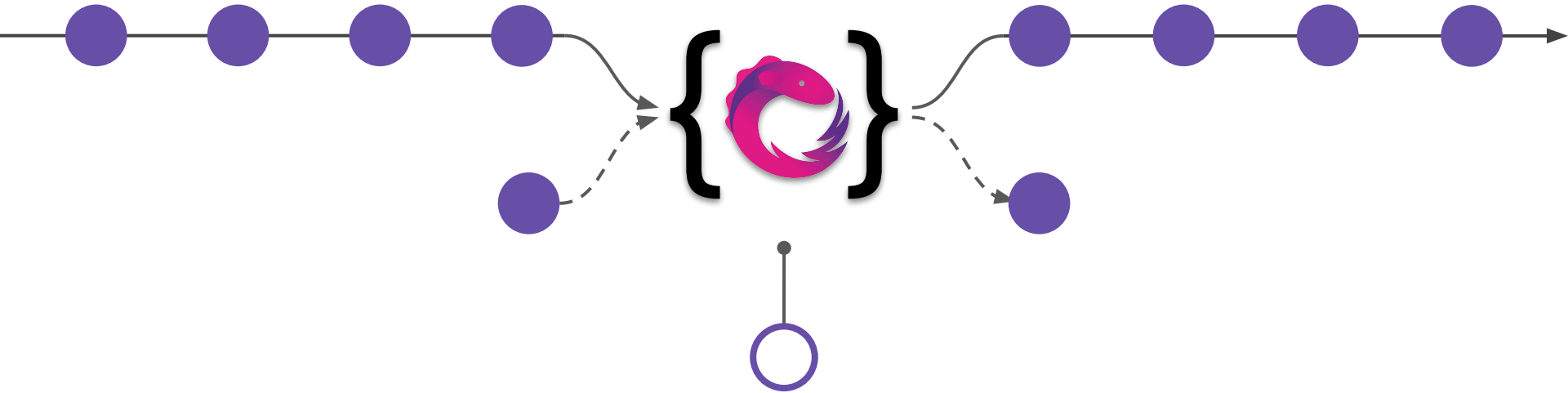
If you stick to the paradigms the design **patterns appear naturally**

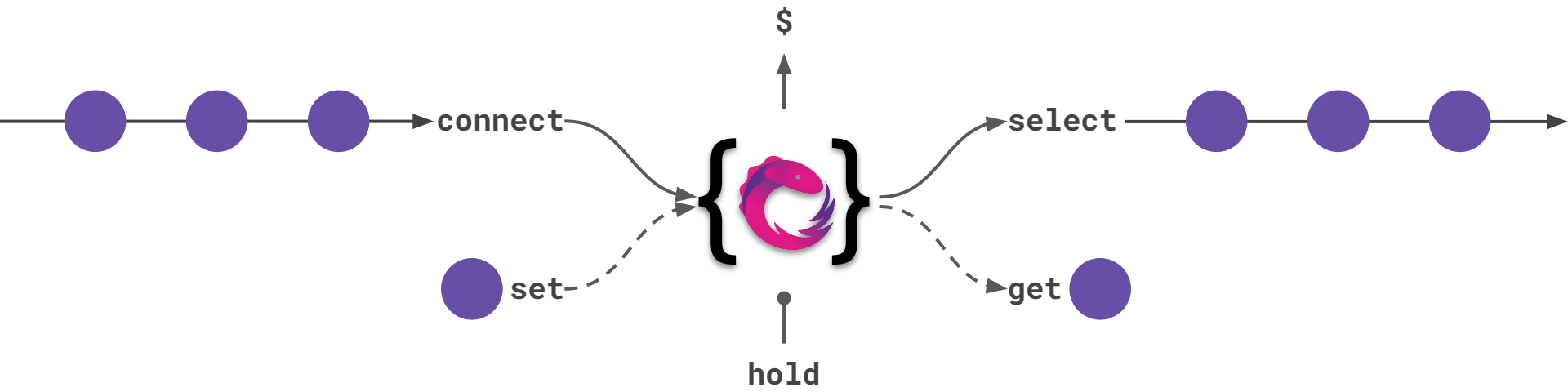
”

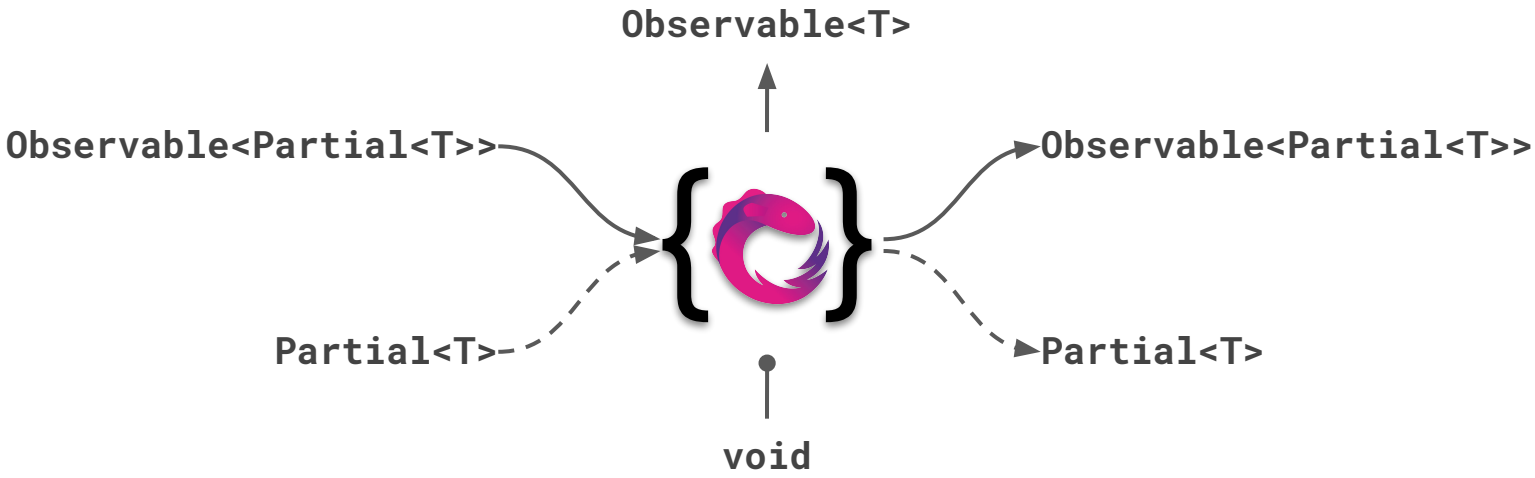
Gang Of Four

Reactive Component State

@rx-angular/state




$
connect
select
set
get
hold


Observable<T>
Observable<Partial<T>>
Observable<Partial<T>>
Partial<T>
Partial<T>
void

Thanks for your time!

If you have any questions just ping me!

And book my consulting! ;)

Lib:
https://www.npmjs.com/package/@rx-angular/state

Research:
dev.to/rxjs/research-on-reactive-ephemeral-state-in-component-oriented-frameworks-38lk

github.com/BioPhoton
michel@hladky.at
@Michael_Hladky