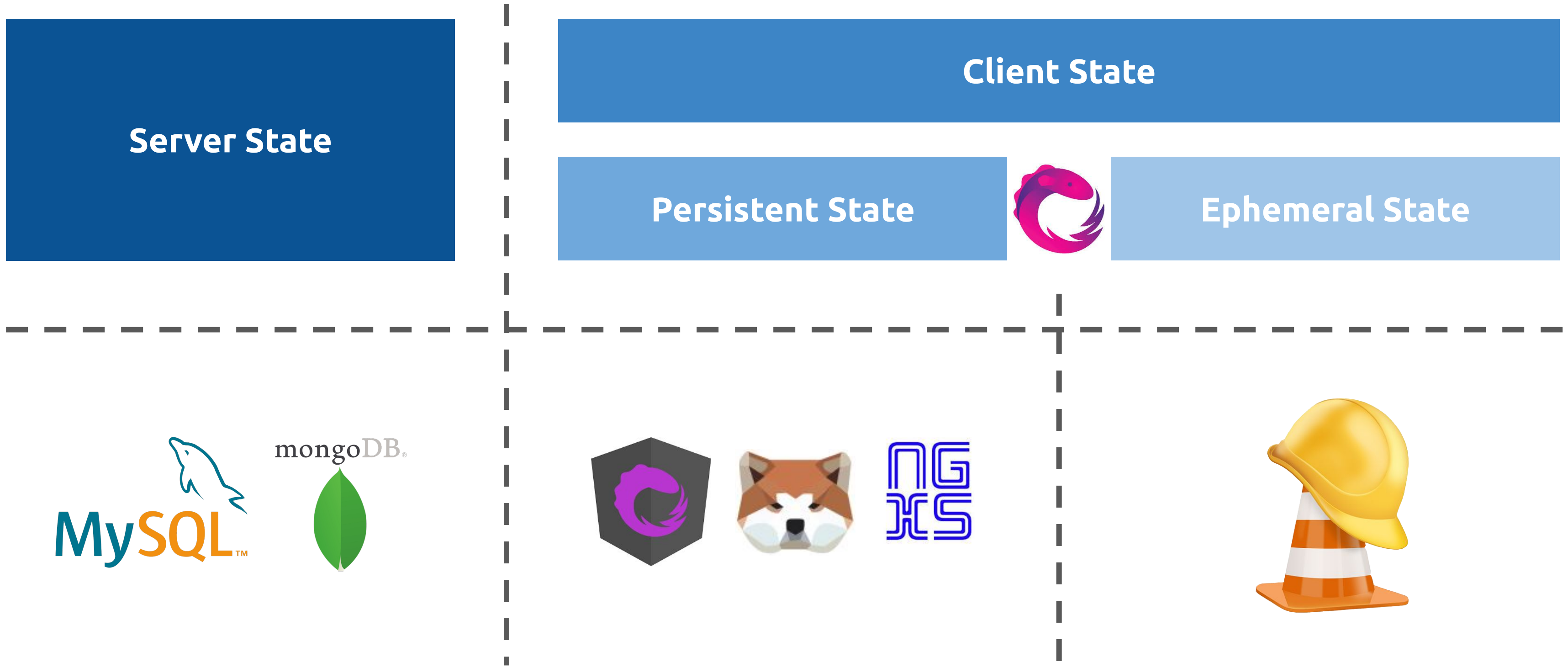
Tackling Component State Reactively

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

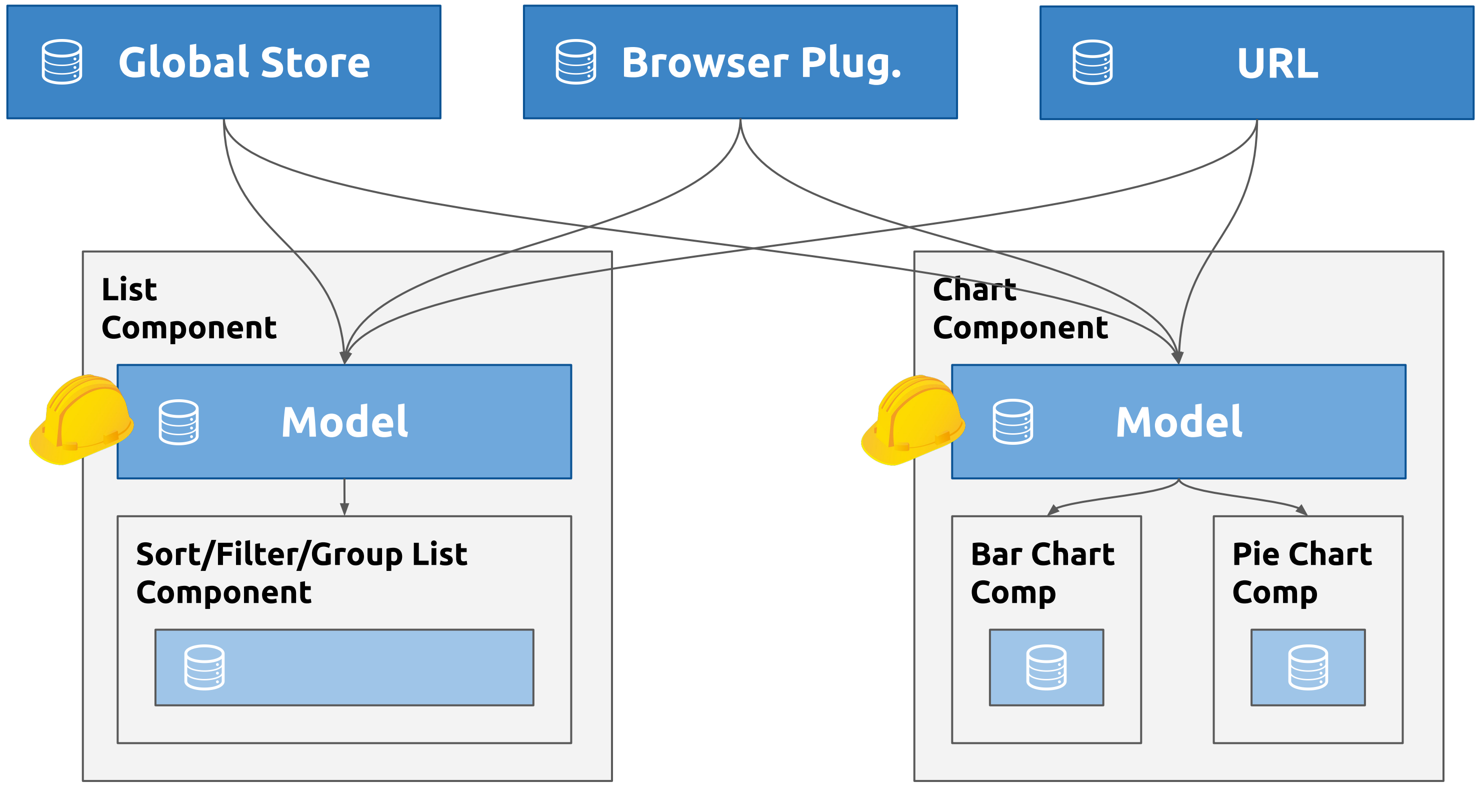
Gang Of Four

Table of content

Layers of State



What is Ephemeral State?



I’m Michael Hladky.
Book my consulting ;)

Trainer & Consultant

Angular
RxJS
Architecture

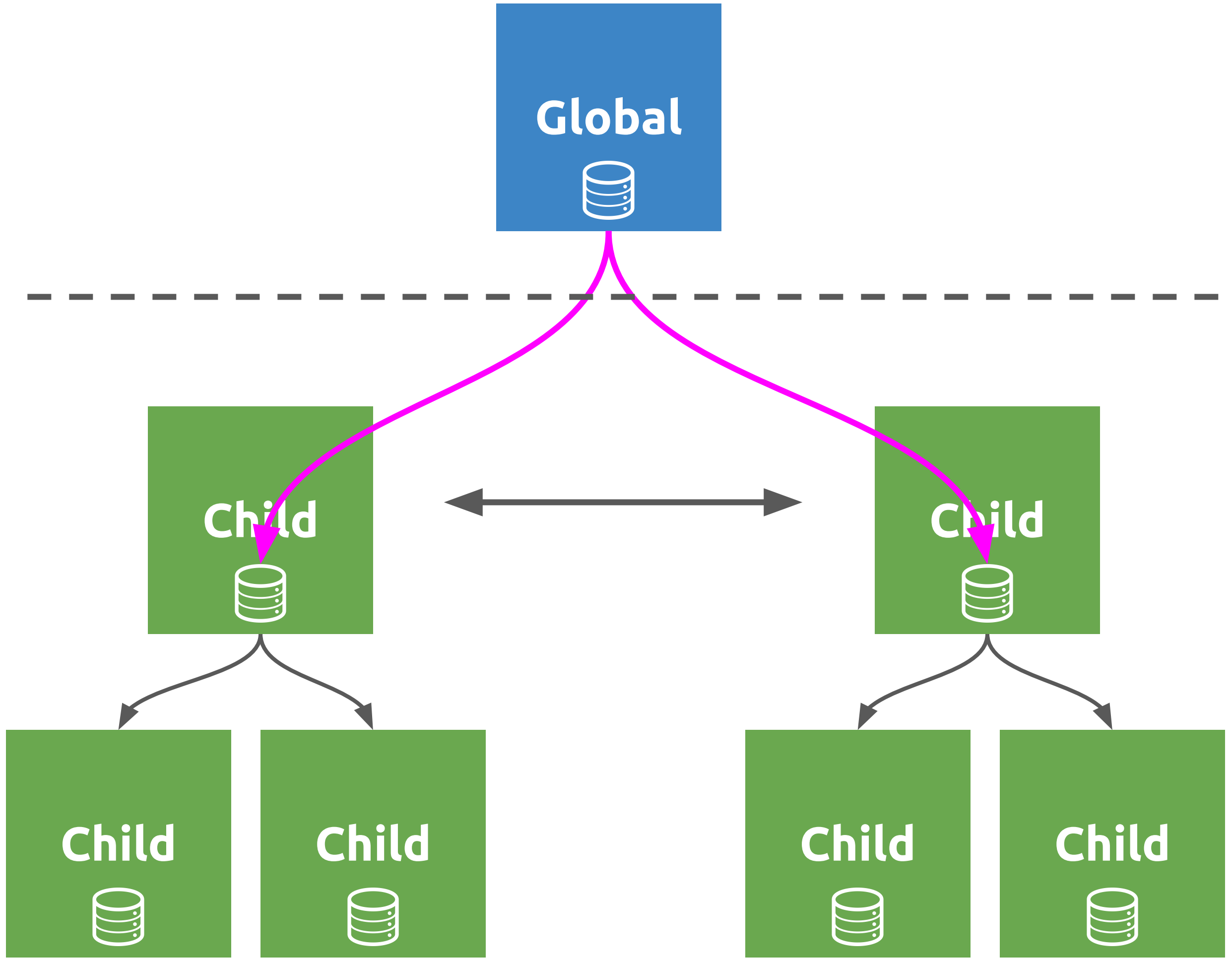
@Michael_Hladky
office@hladky.at

Terminology and Categorisation

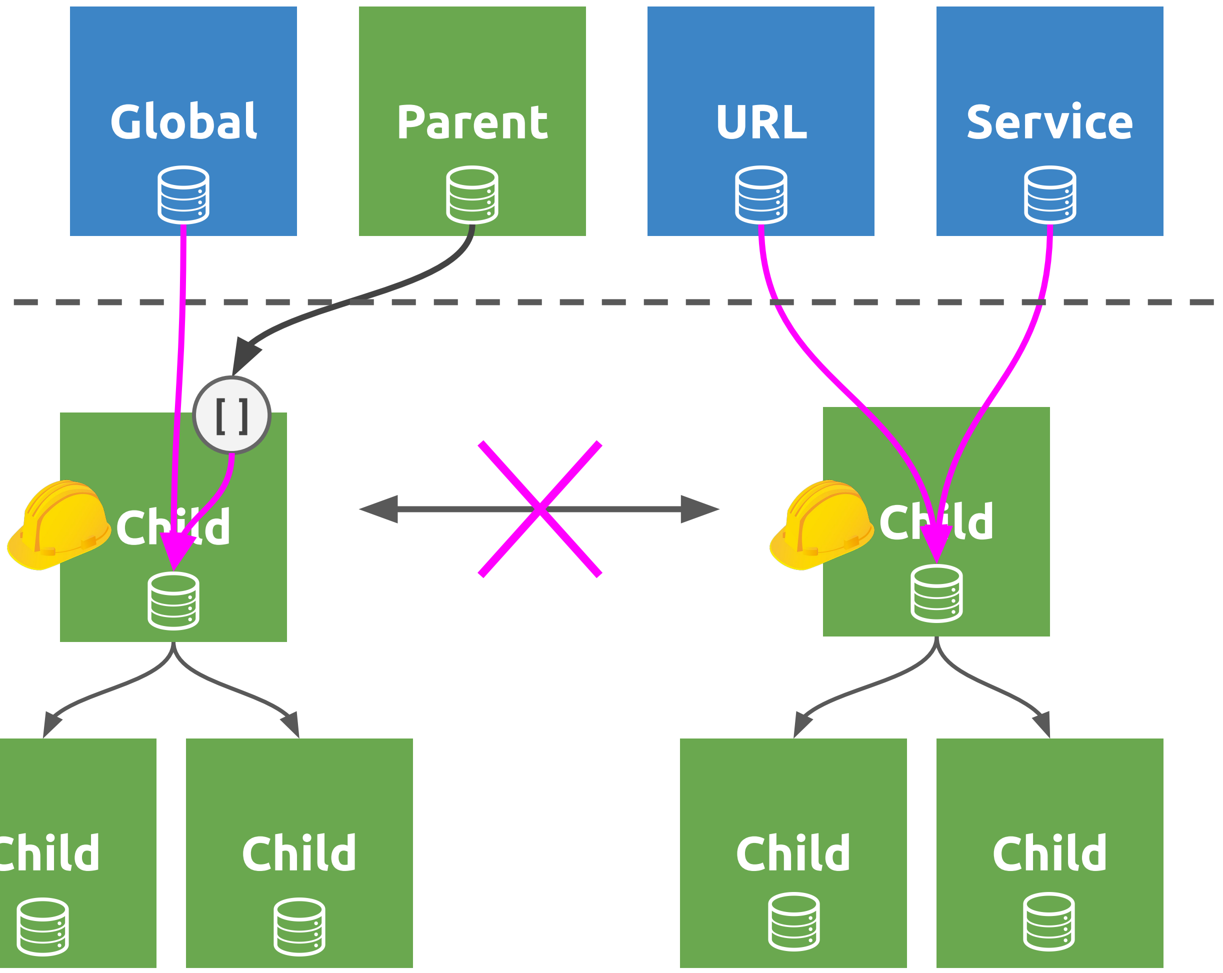
Persistent	State	Ephemeral
Globally	**Accessibility**	**Locally**
Static	**Lifetime**	**Dynamic**
Remote	**Processed Sources**	**Local**

Accessibility

Global
Accessible Data

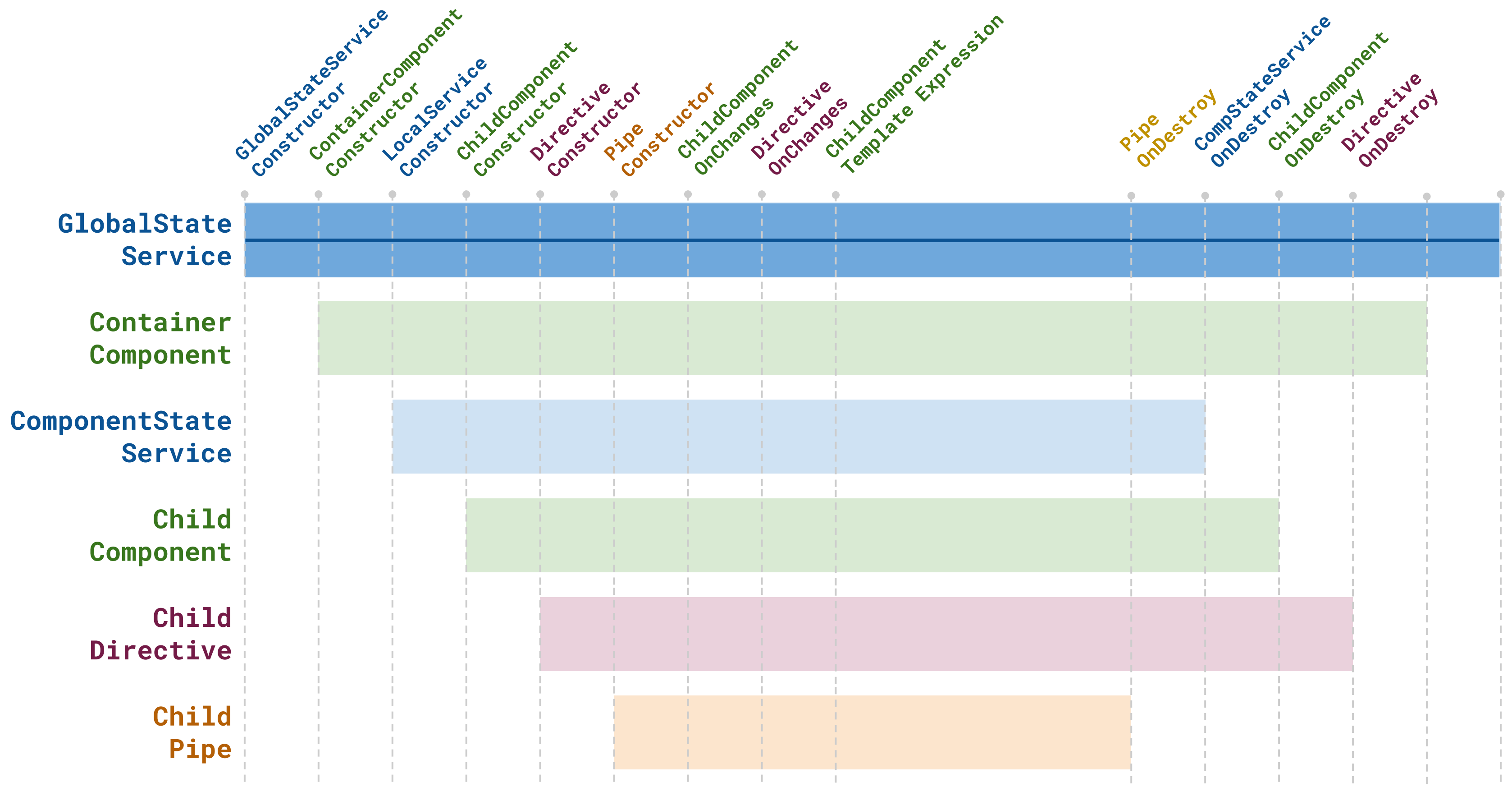


Local
Accessible Data

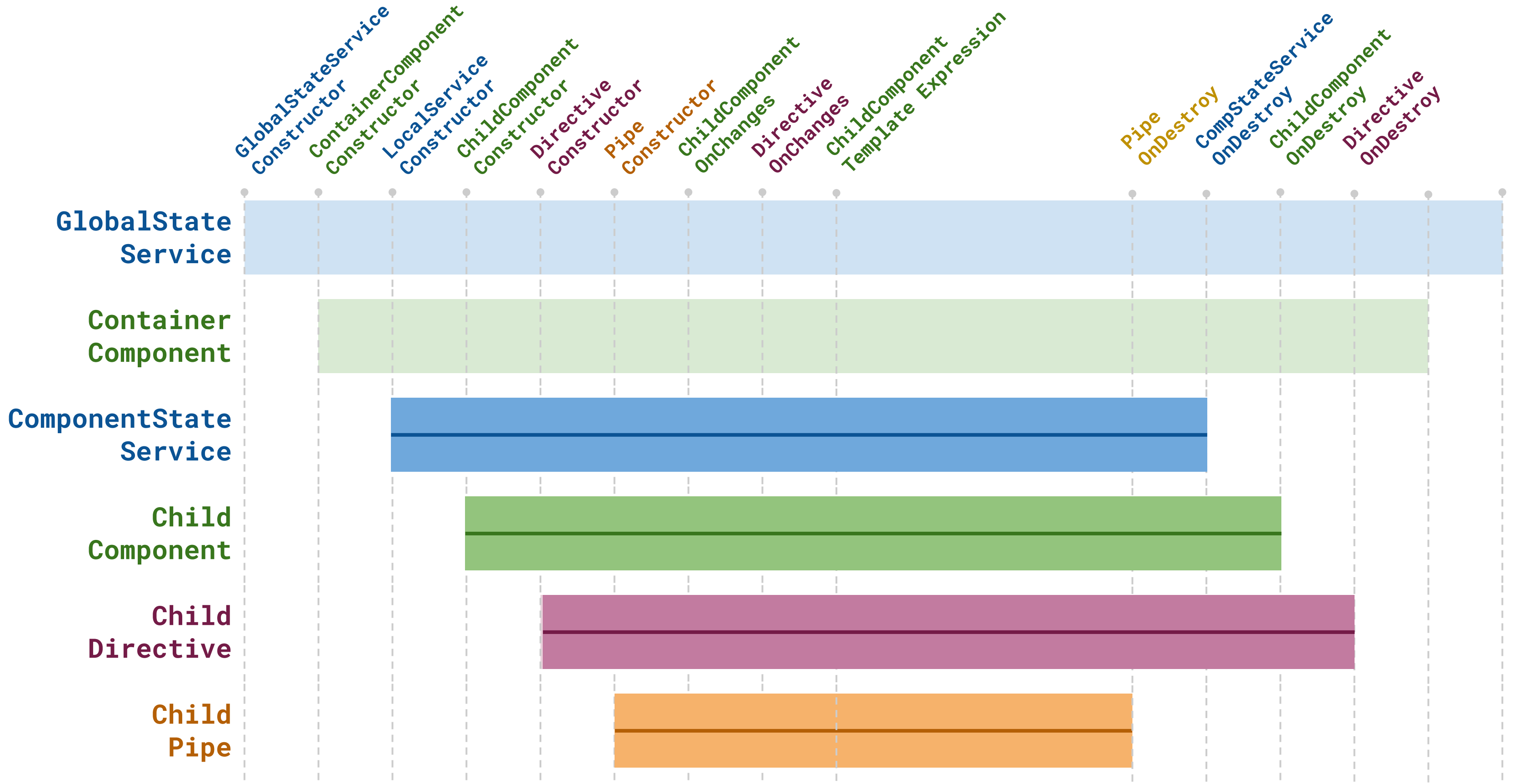


LifeTime

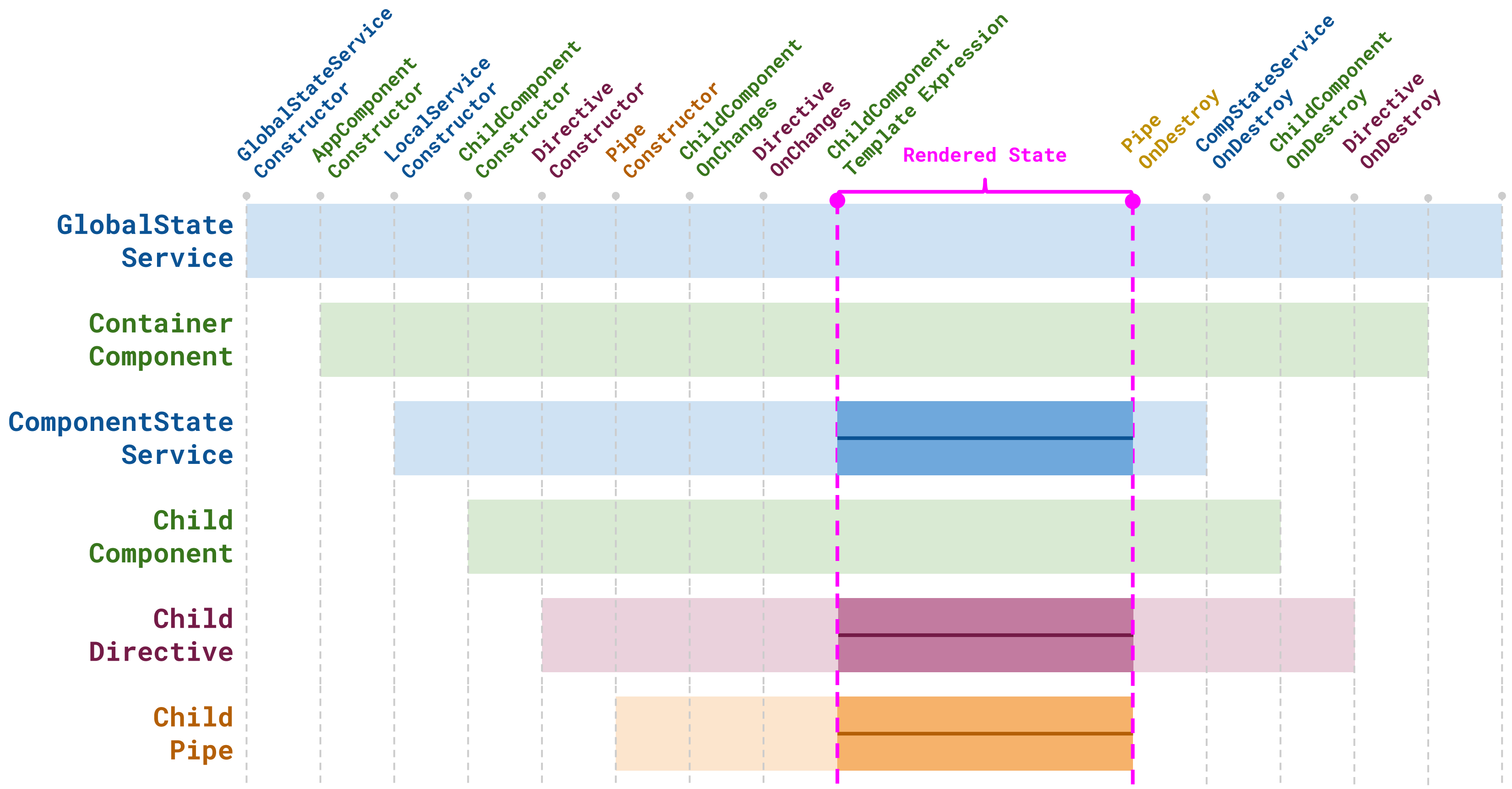
Static Lifetime - Global Singleton Service



Dynamic Lifetime - Angular Building Blocks

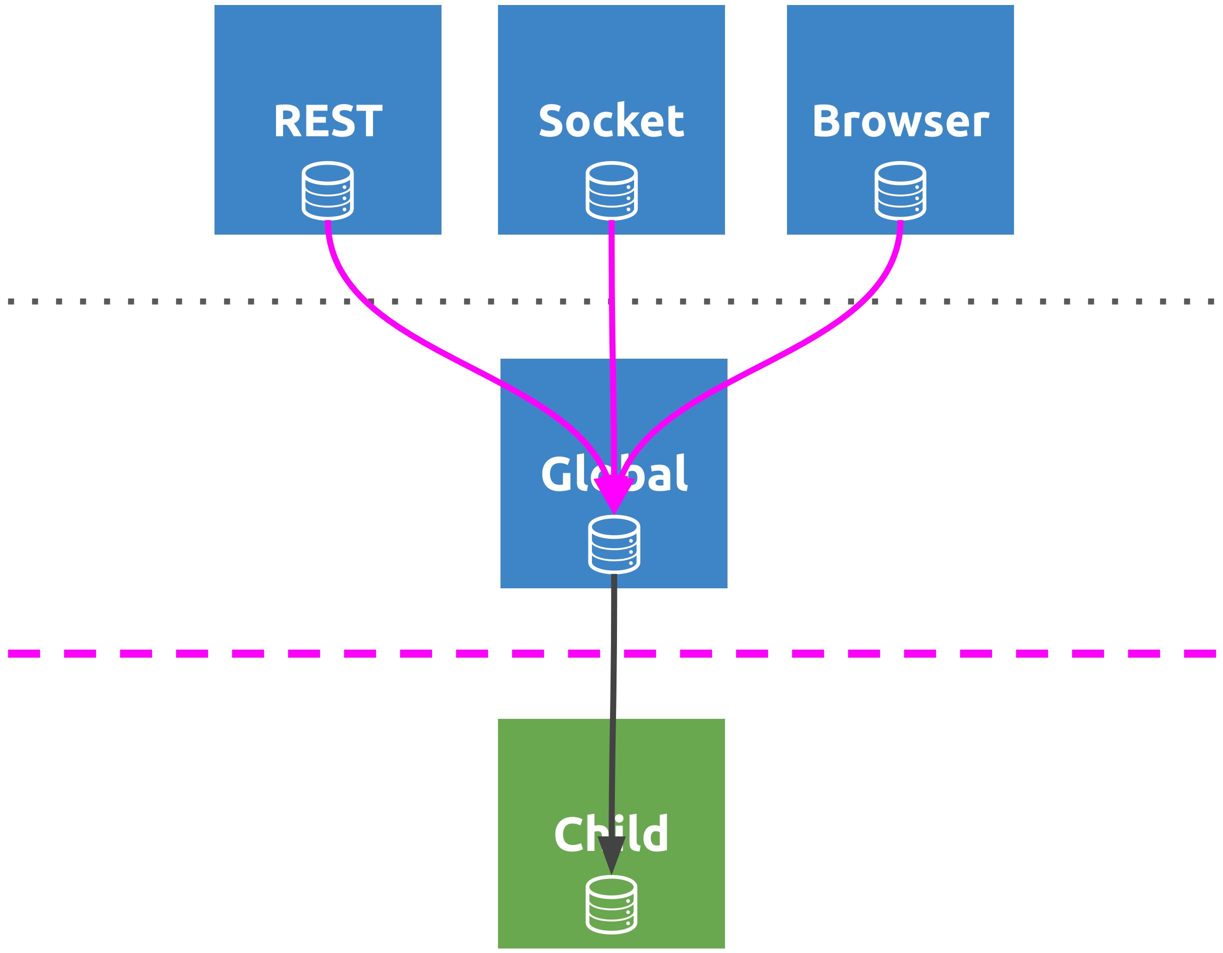


Dynamic Lifetime - Data over `async` Pipe

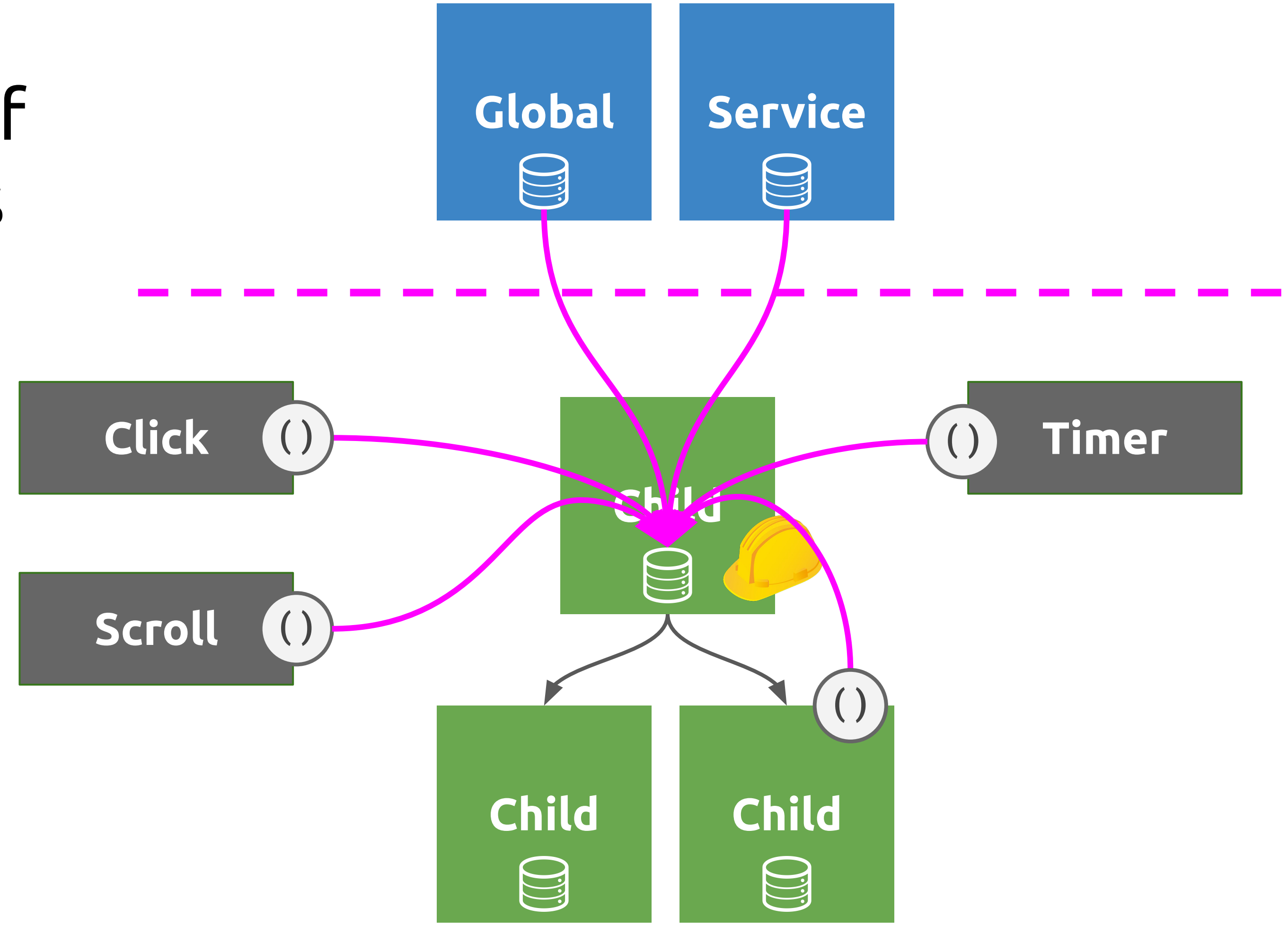


Processed Sources

Processing of global sources



Processing of local sources



Problems

Timing

Sharing Work or Instances

Subscription Handling

Late Subscriber

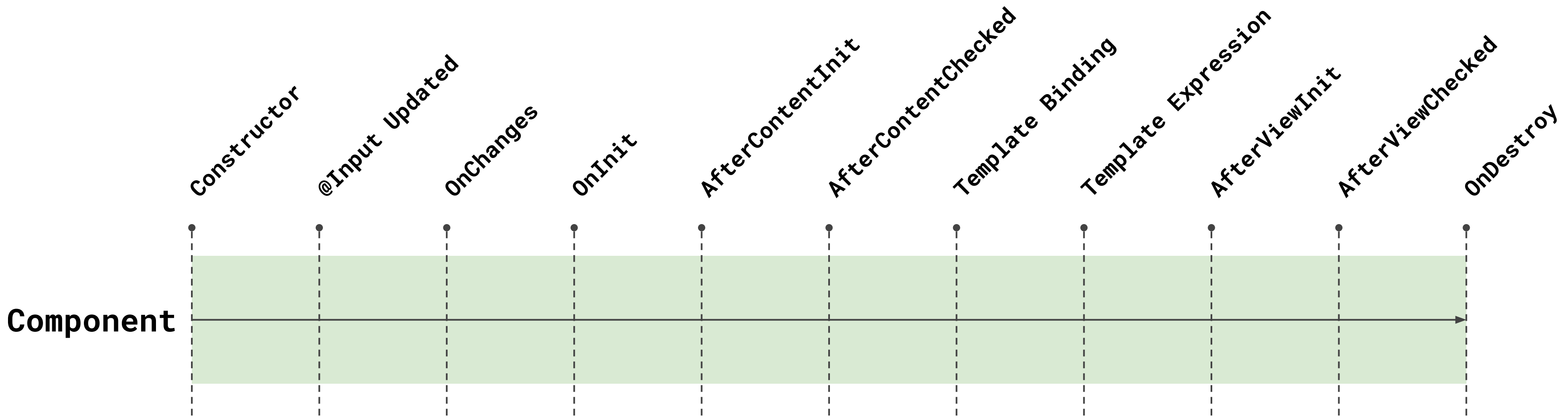
Subscription-Less Interaction

Protection against misusage

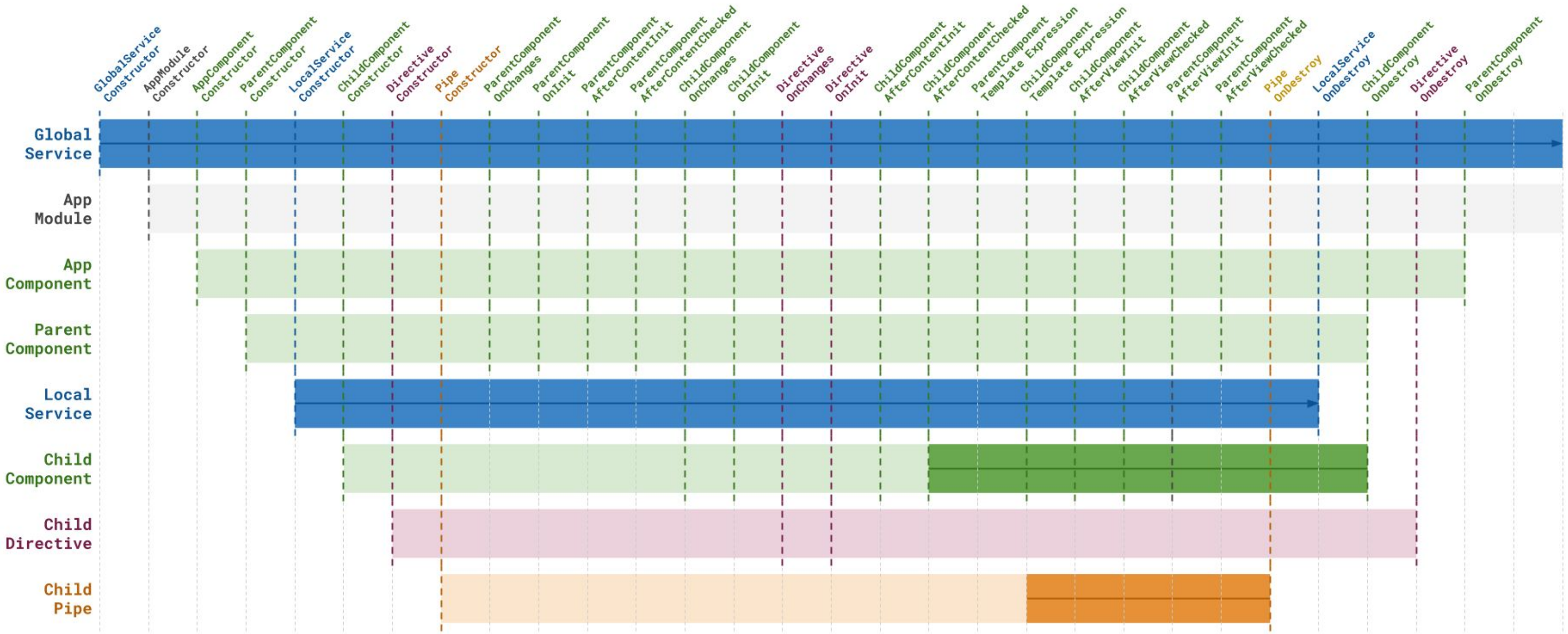
Cleanup of Dead State

Timing

Lifecycle Hooks - One Single Component



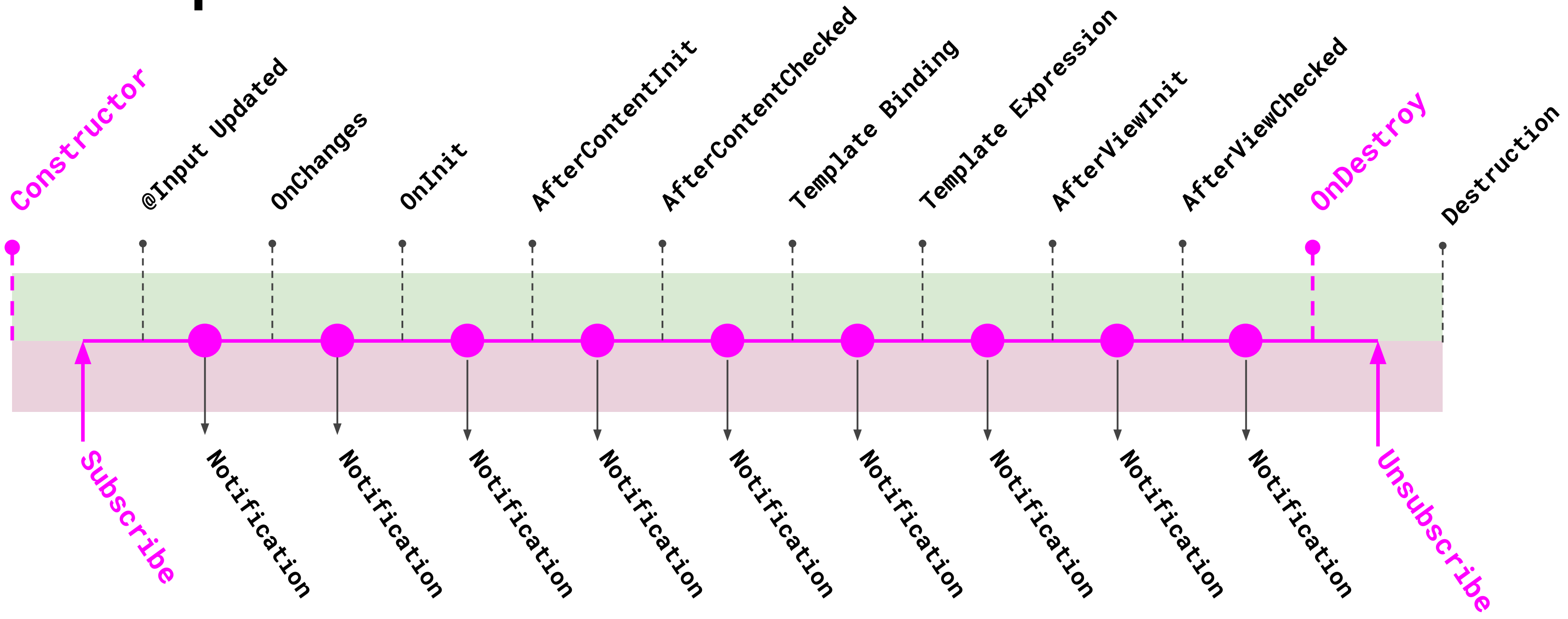
Lifecycle Hooks and Subscriptions - Hello World


GlobalService Constructor
AppModule Constructor
AppComponent Constructor
ParentComponent Constructor
LocalService Constructor
ChildComponent Constructor
Directive Constructor
Pipe Constructor
ParentComponent OnChanges
ParentComponent OnInit
ParentComponent AfterContentInit
ParentComponent AfterContentChecked
ChildComponent OnChanges
ChildComponent OnInit
Directive OnChanges
Directive OnInit
ChildComponent AfterContentInit
ChildComponent AfterContentChecked
ParentComponent Template Expression
ChildComponent Template Expression
ChildComponent AfterViewInit
ChildComponent AfterViewChecked
ParentComponent AfterViewInit
ParentComponent AfterViewChecked
Pipe OnDestroy
LocalService OnDestroy
ChildComponent OnDestroy
Directive OnDestroy
ParentComponent OnDestroy
Global Service
App Module
App Component
Parent Component
Local Service
Child Component
Child Directive
Child Pipe

Subscription Handling
By Lifetime

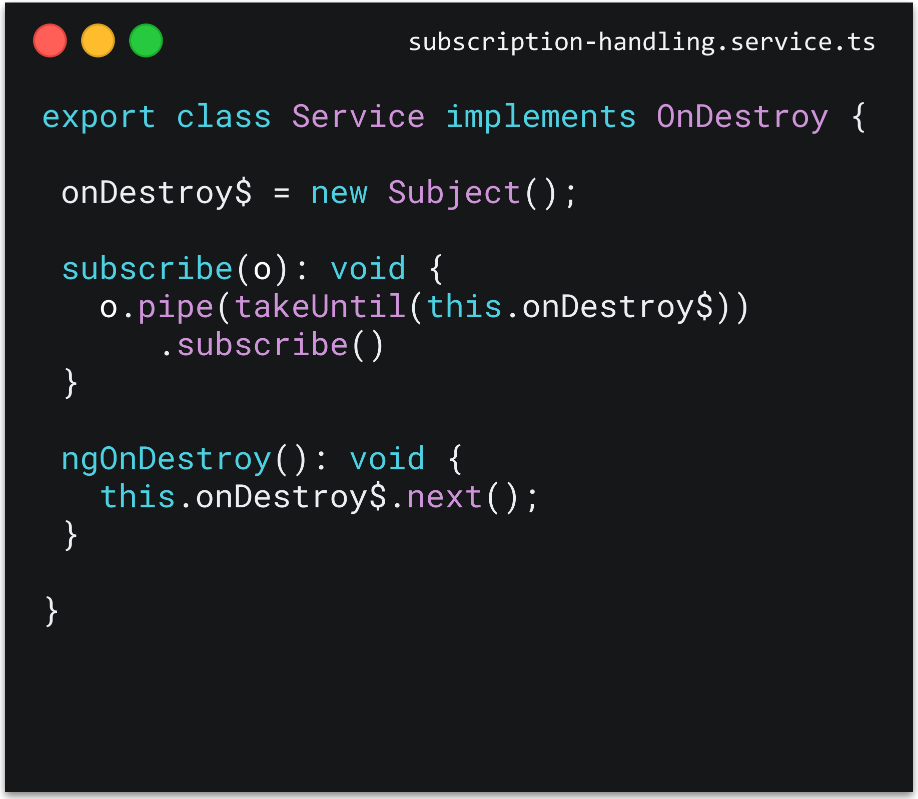
When to subscribe/unsubscribe?

Component



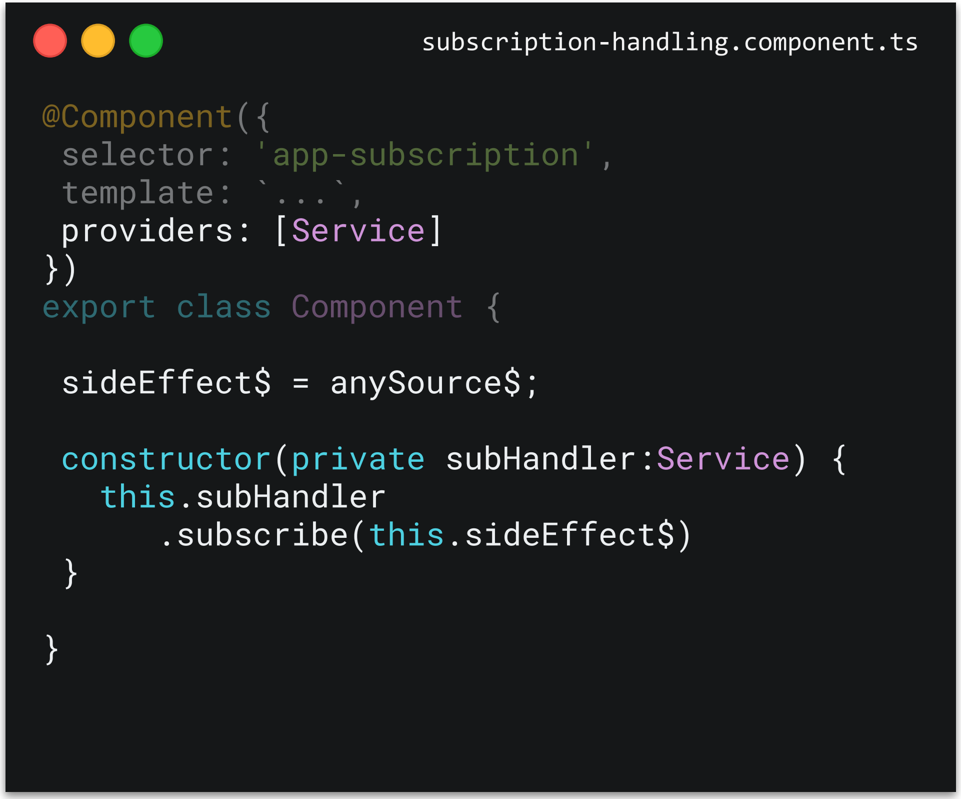
Observable

Subscription Handling via Component Providers



subscription-handling.service.ts

```
export class Service implements OnDestroy {

 onDestroy$ = new Subject();

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

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

}
```



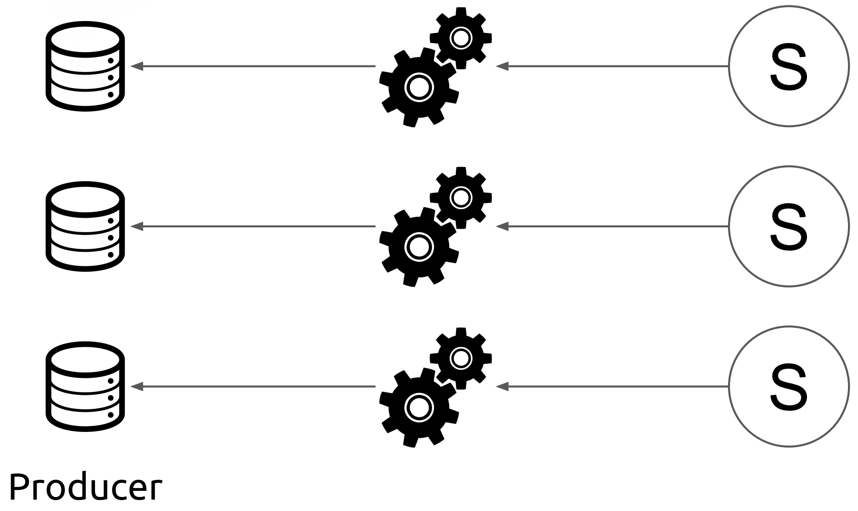
subscription-handling.component.ts

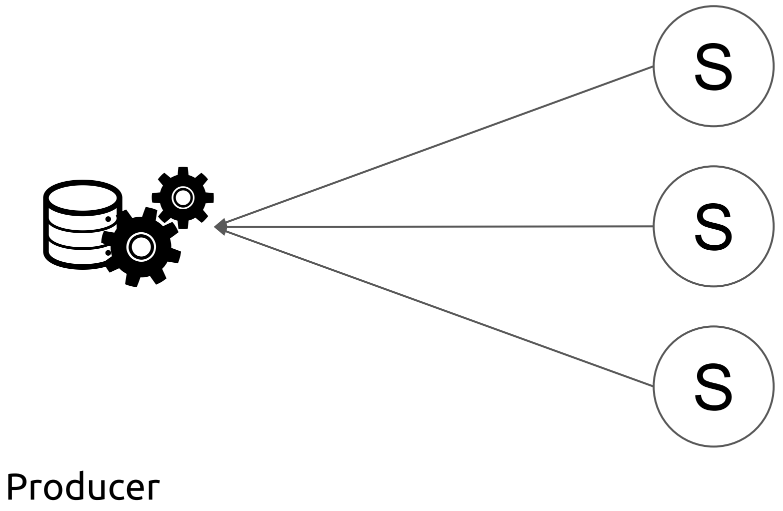
```
@Component({
 selector: 'app-subscription',
 template: `...`,
 providers: [Service]
})
export class Component {

 sideEffect$ = anySource$;

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

}
```

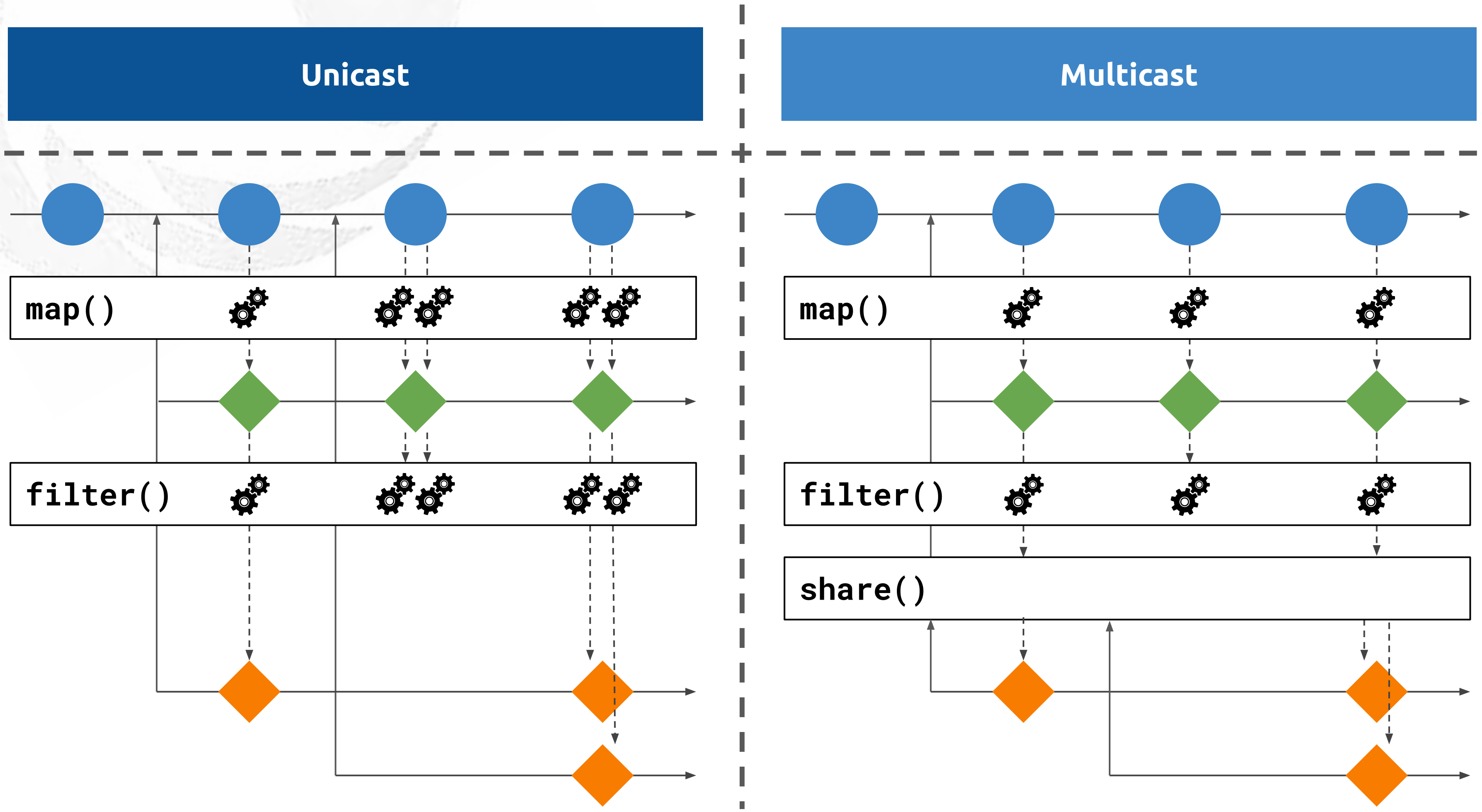
Share Work and Instance

Unicast

Multicast

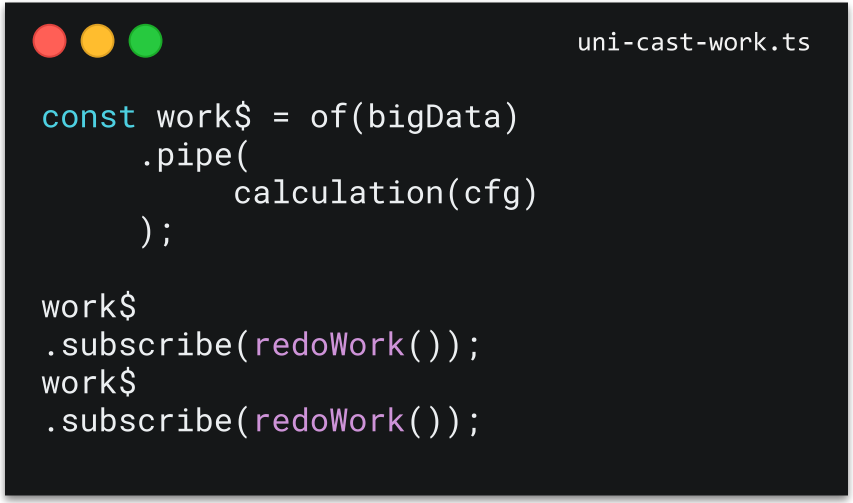




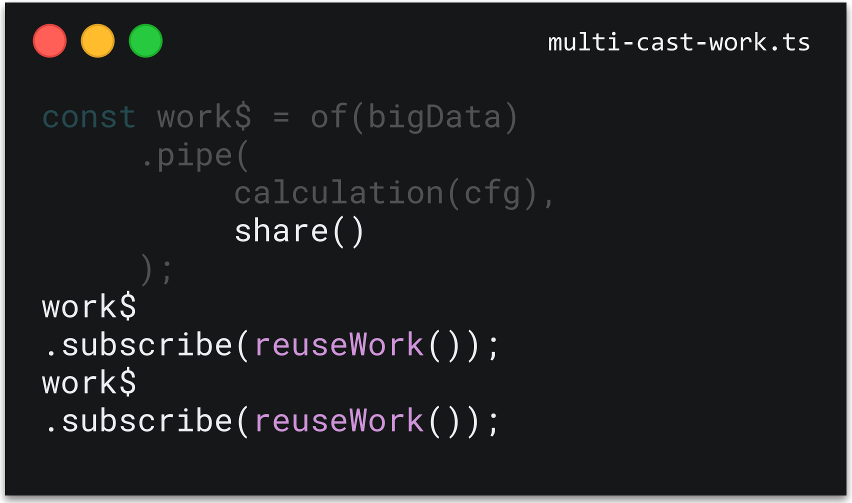

Unicast
Multicast
map()
filter()
map()
filter()
share()

Unicast

Multicast


uni-cast-work.ts

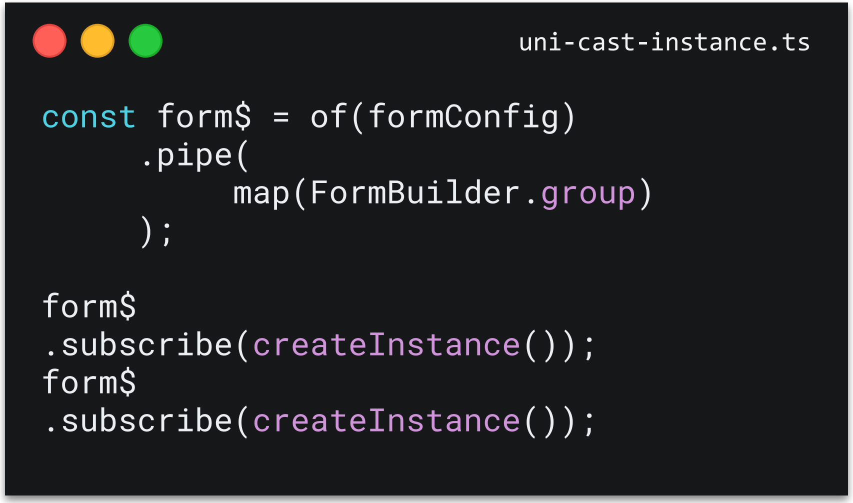
```
const work$ = of(bigData)
      .pipe(
            calculation(cfg)
      );

work$
.subscribe(redoWork());
work$
.subscribe(redoWork());
```


multi-cast-work.ts

```
const work$ = of(bigData)
      .pipe(
            calculation(cfg),
            share()
      );
work$
.subscribe(reuseWork());
work$
.subscribe(reuseWork());
```

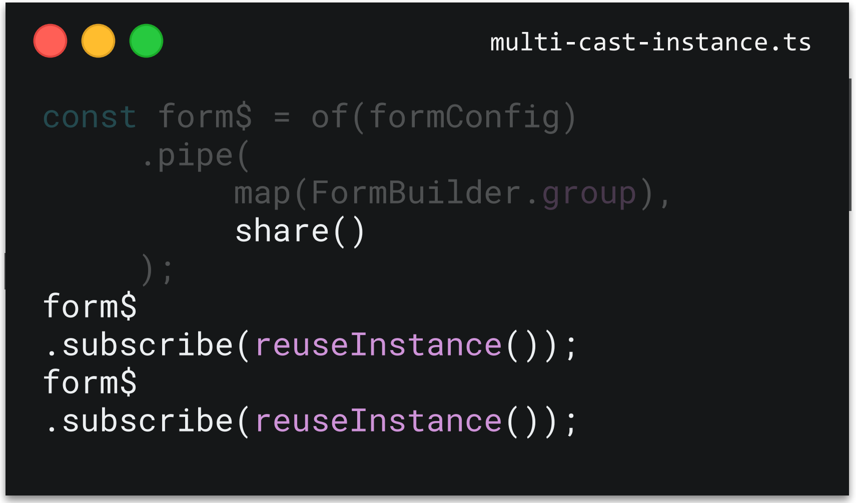
Unicast

Multicast



uni-cast-instance.ts

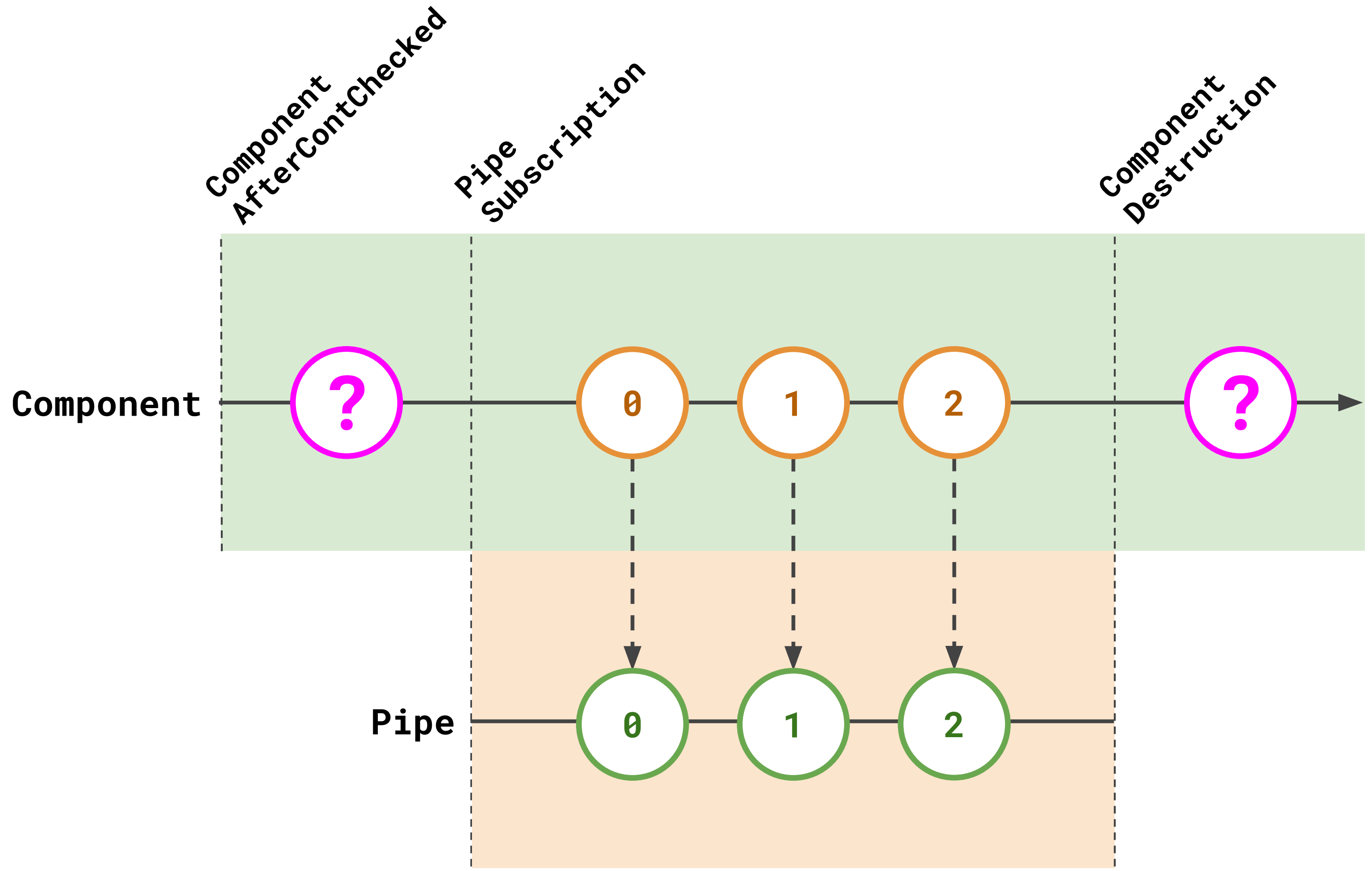
```
const form$ = of(formConfig)
     .pipe(
          map(FormBuilder.group)
     );

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



multi-cast-instance.ts

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

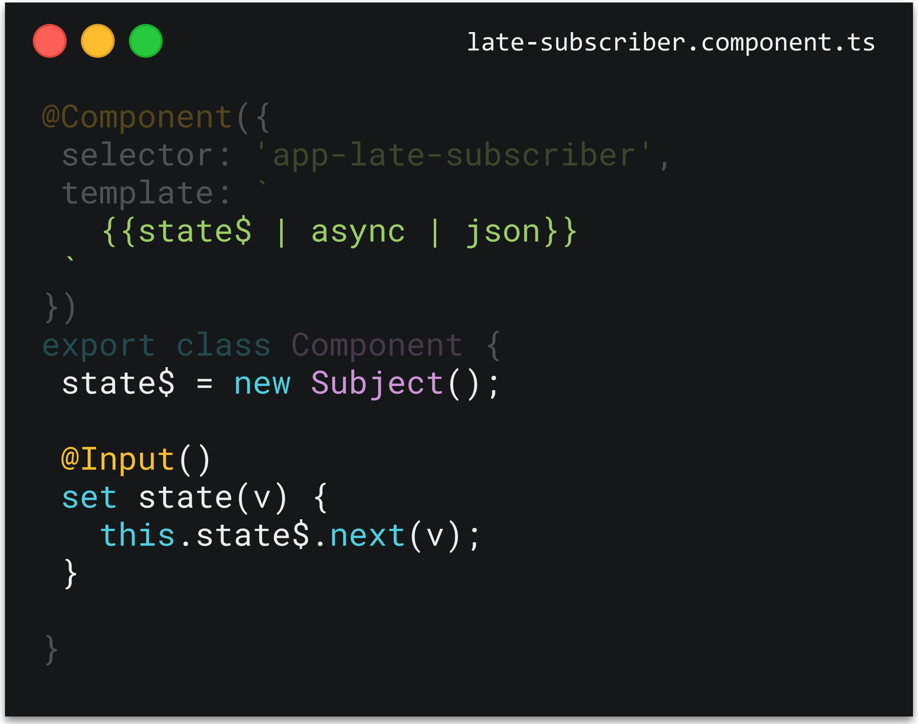
Late Subscriber


Component AfterContChecked
Pipe Subscription
Component Destruction
Component
?
0
1
2
?
Pipe
0
1
2

Problem

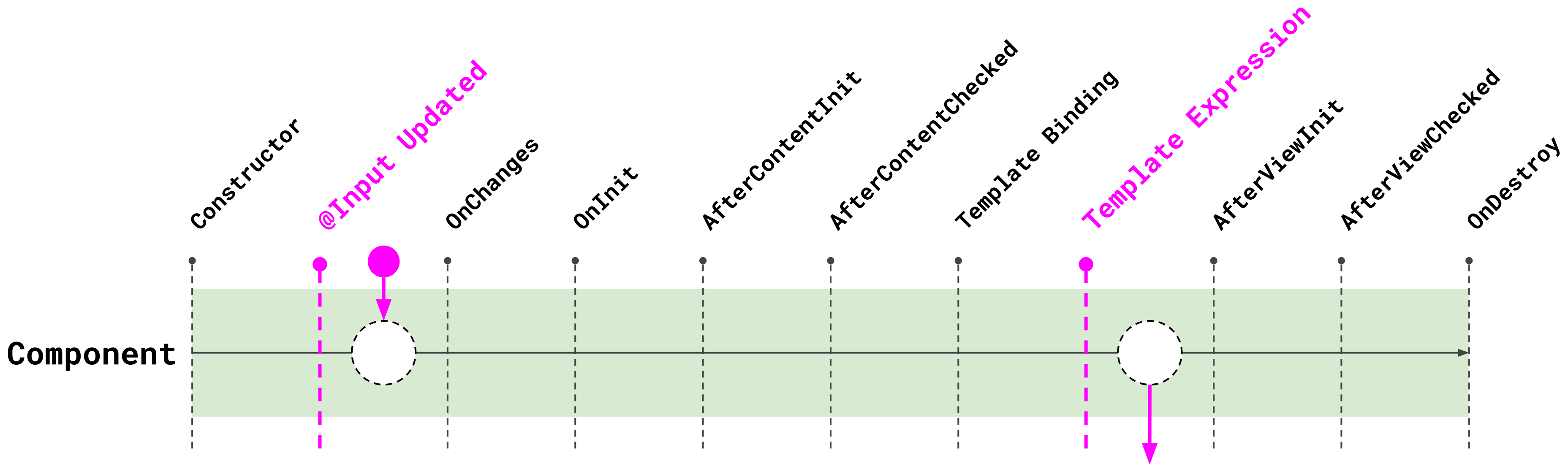
Late Subscriber - Lifecycle Hooks

Late Subscriber - Lifecycle Hooks - Problem



late-subscriber.component.ts

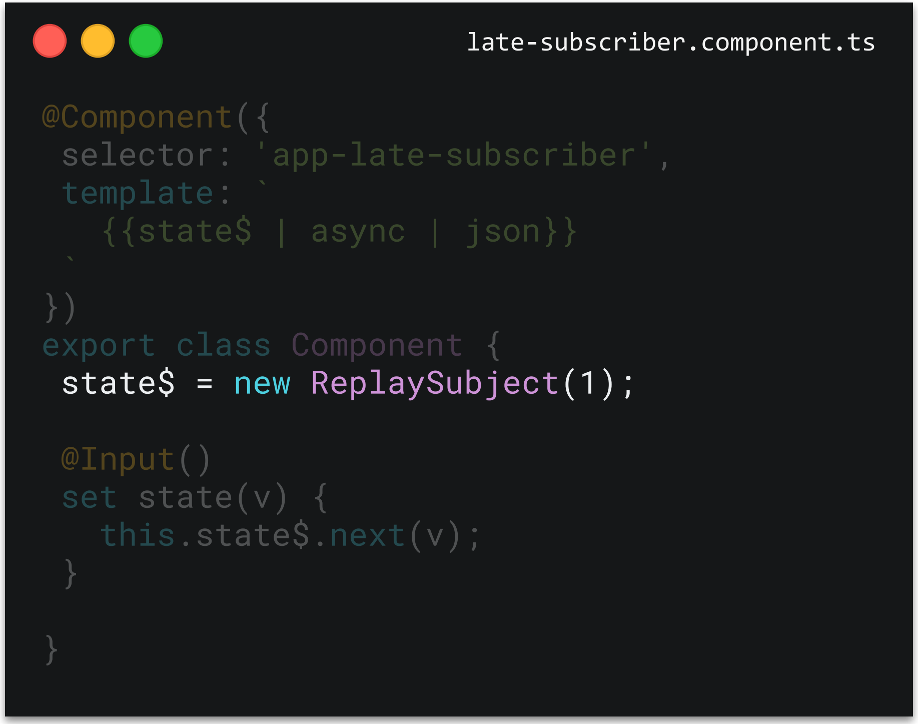
```
@Component({
 selector: 'app-late-subscriber',
 template: `
   {{state$ | async | json}}
 `
})
export class Component {
 state$ = new Subject();

 @Input()
 set state(v) {
   this.state$.next(v);
 }

}
```


Constructor
@Input Updated
OnChanges
OnInit
AfterContentInit
AfterContentChecked
Template Binding
Template Expression
AfterViewInit
AfterViewChecked
OnDestroy
Component

Solution

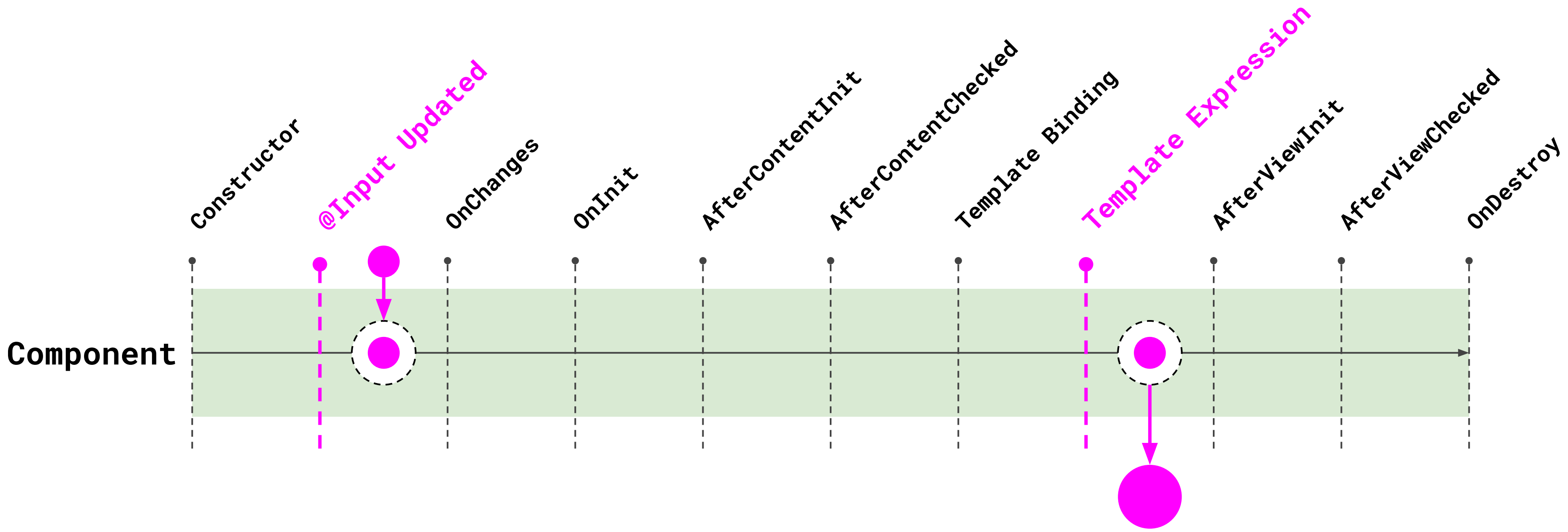
Late Subscriber - Lifecycle Hooks

Late Subscriber - Lifecycle Hooks - Solution



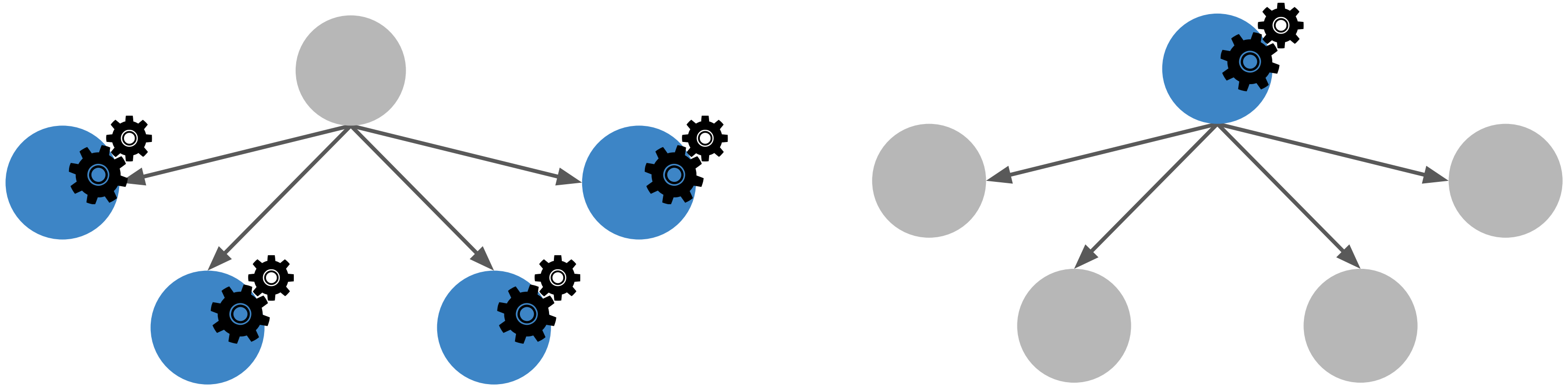
late-subscriber.component.ts

```
@Component({
 selector: 'app-late-subscriber',
 template: `
   {{state$ | async | json}}
 `
})
export class Component {
 state$ = new ReplaySubject(1);

 @Input()
 set state(v) {
   this.state$.next(v);
 }

}
```


Constructor
@Input Updated
OnChanges
OnInit
AfterContentInit
AfterContentChecked
Template Binding
Template Expression
AfterViewInit
AfterViewChecked
OnDestroy
Component

Caveat
Push workload to multiple others

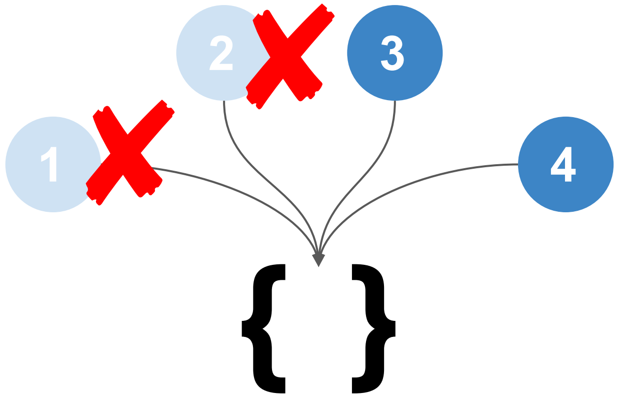
Is not always the best solution

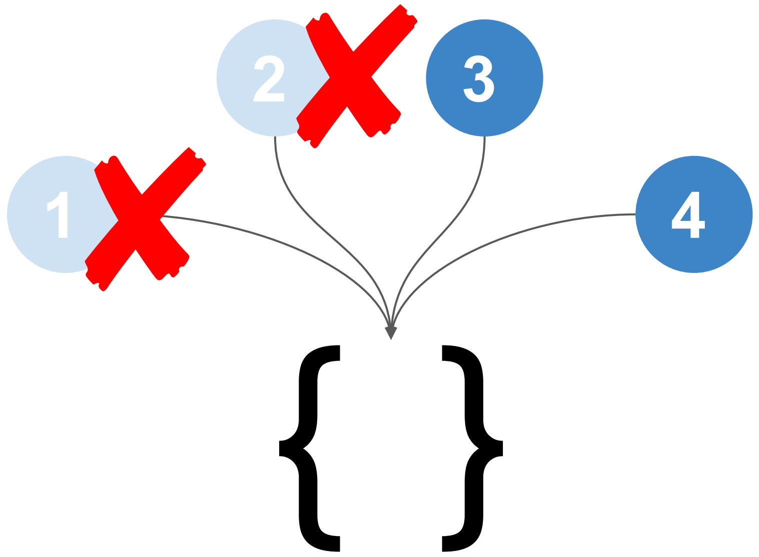


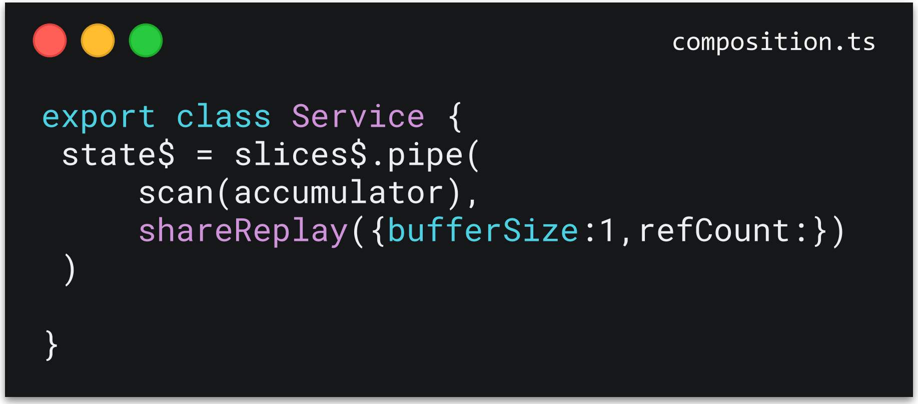
Caveat

State Composition is still cold!

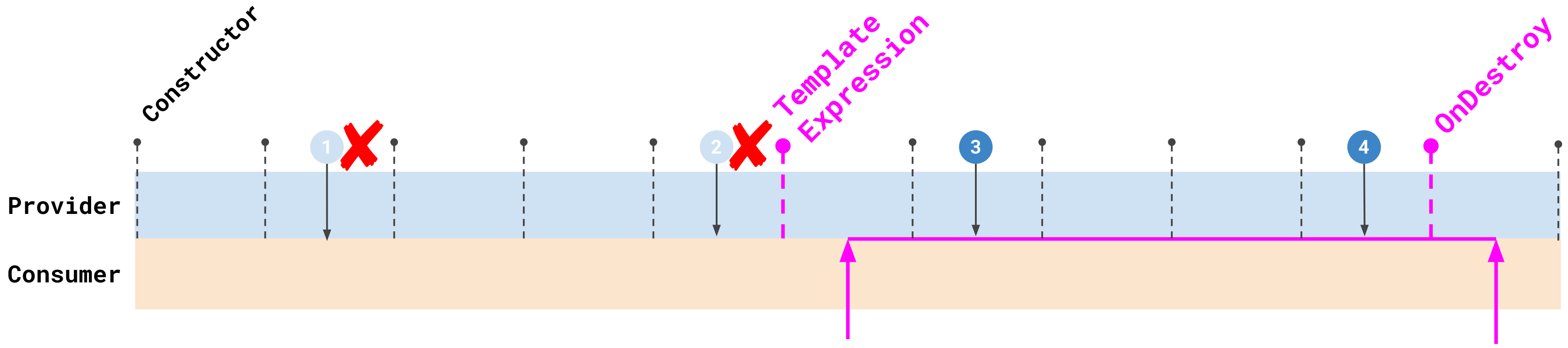
We rely on the consumer to start it!

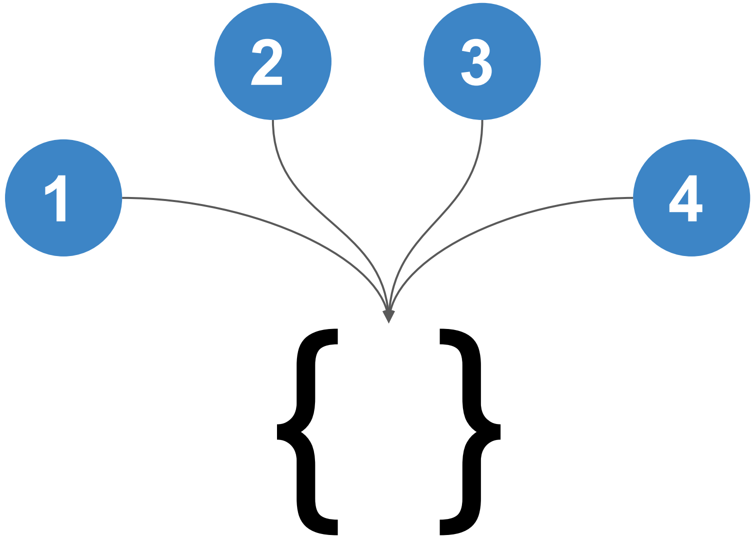


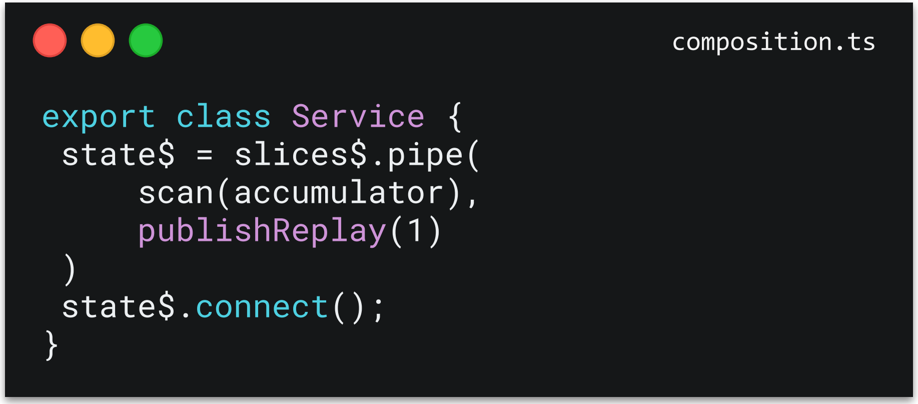



composition.ts

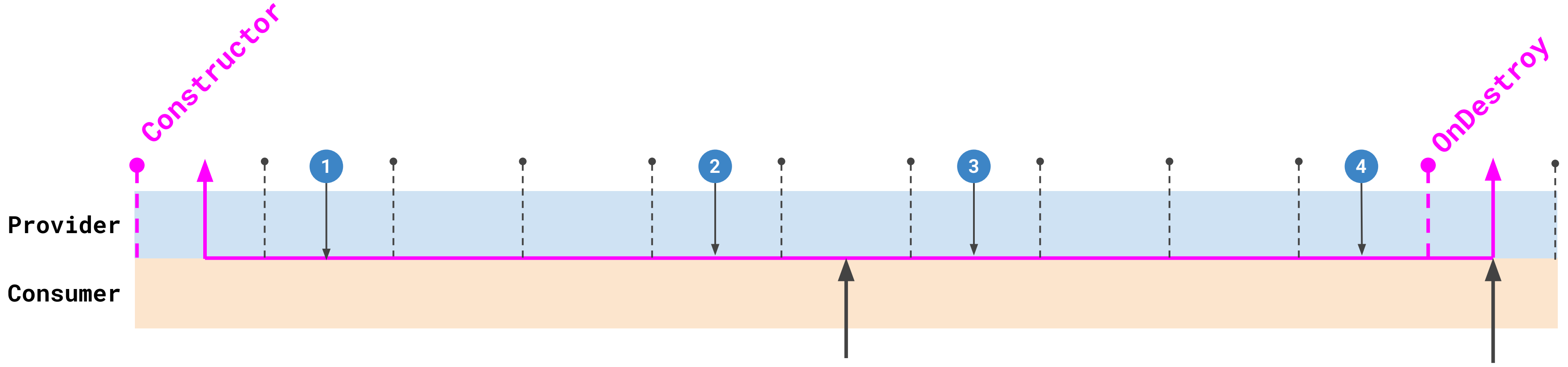
```
export class Service {
 state$ = slices$.pipe(
      scan(accumulator),
      shareReplay({bufferSize:1,refCount:})
 )

}
```





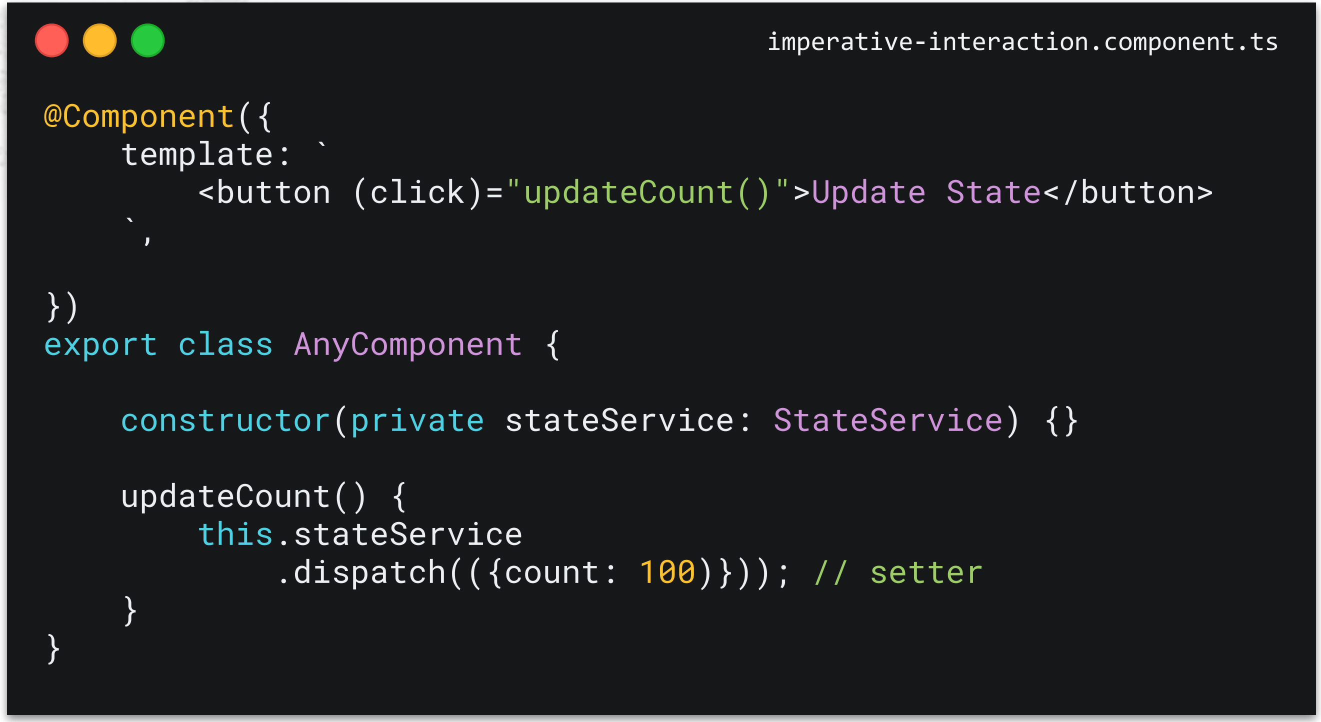

composition.ts

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



Subscription-Less Interaction
with Component-State

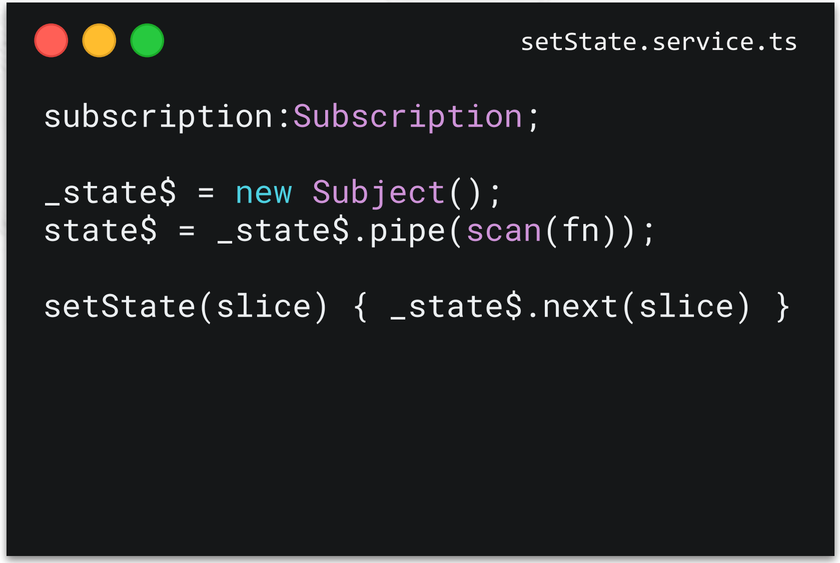
Setters are not Composable



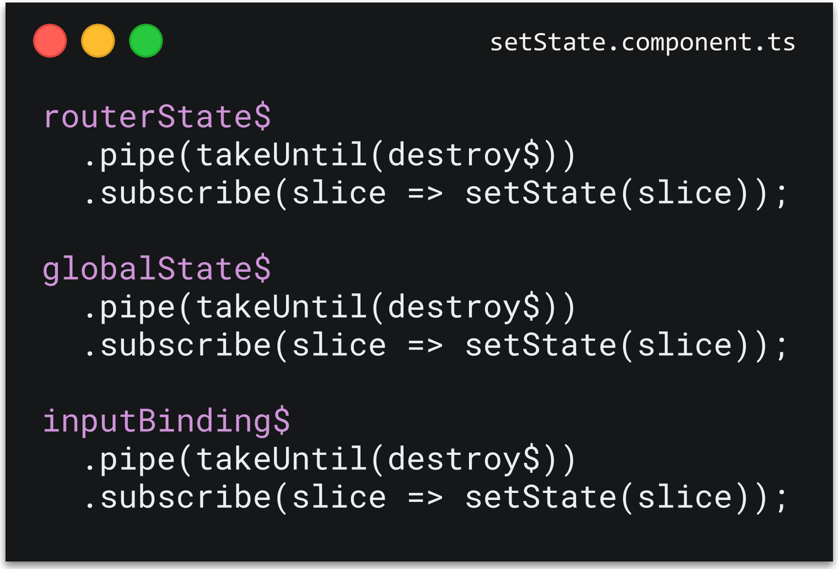
```
@Component({
    template: `
        <button (click)="updateCount()">Update State</button>
    `,

})
export class AnyComponent {

    constructor(private stateService: StateService) {}

    updateCount() {
        this.stateService
            .dispatch(({count: 100})); // setter
    }
}
```

Problem

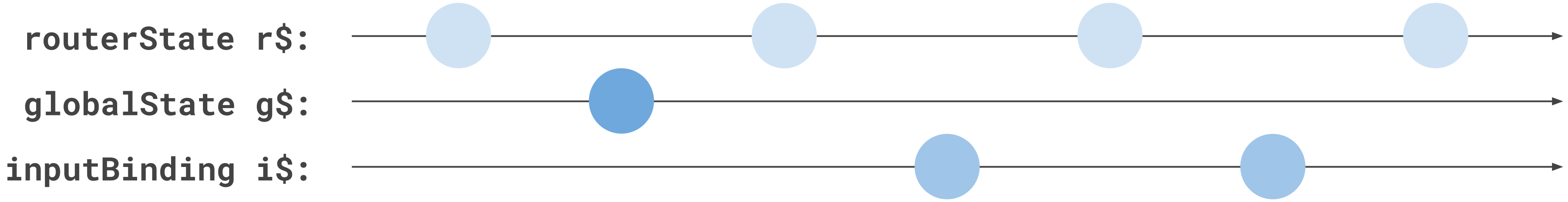
Setters are not composable

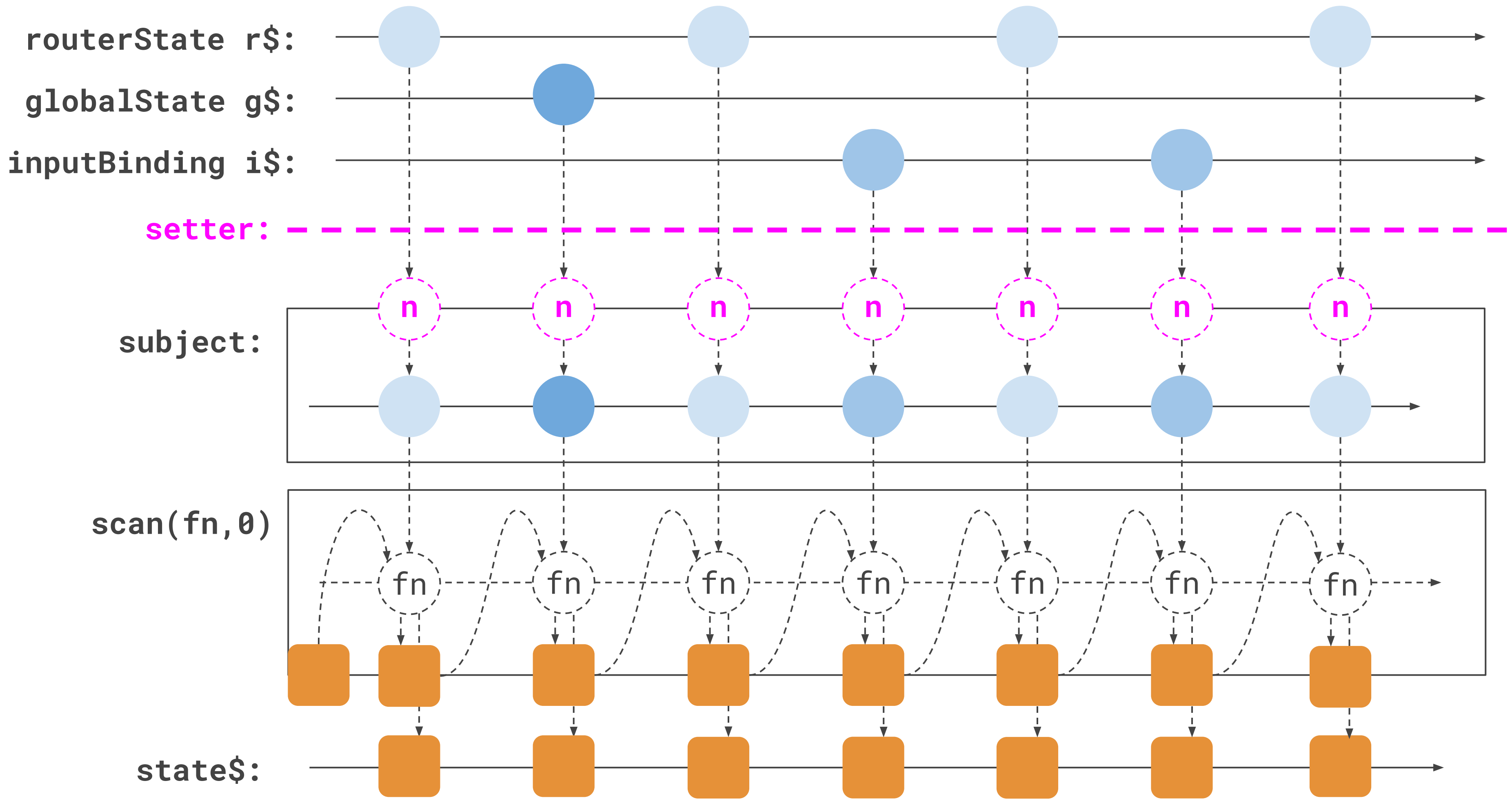

setState.service.ts

```
subscription:Subscription;

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

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


setState.component.ts

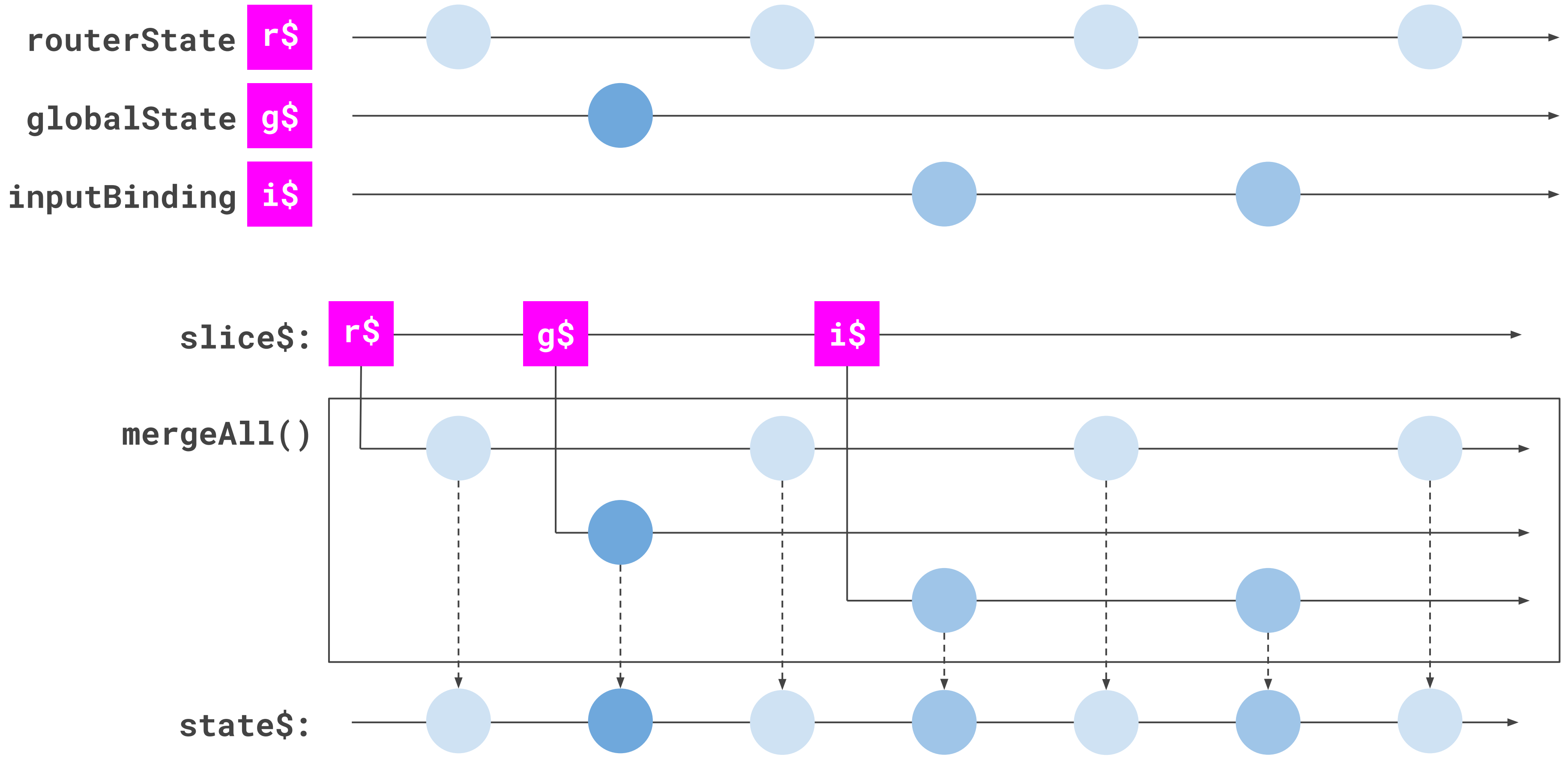
```
routerState$
   .pipe(takeUntil(destroy$))
   .subscribe(slice => setState(slice));

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

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

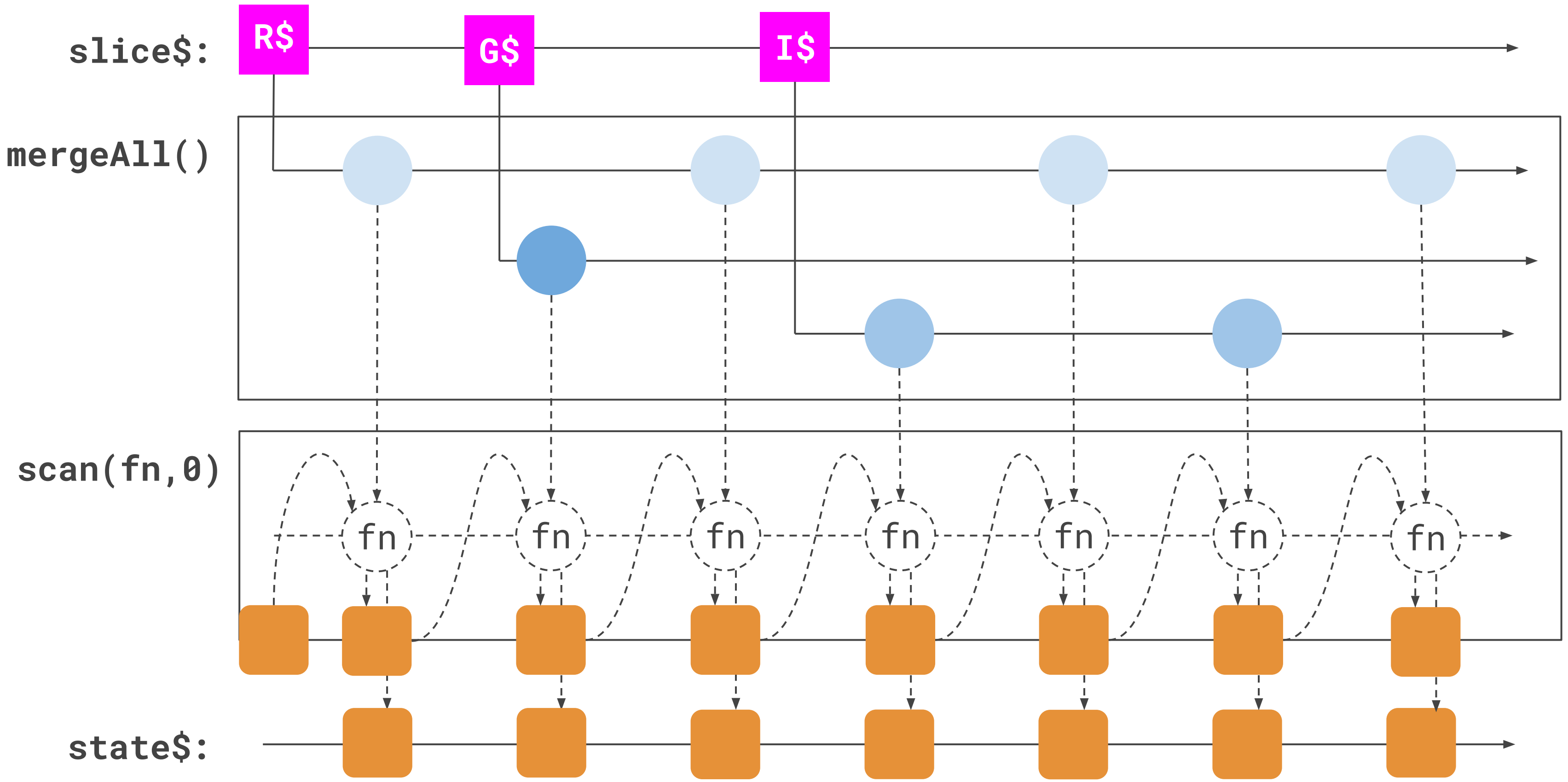


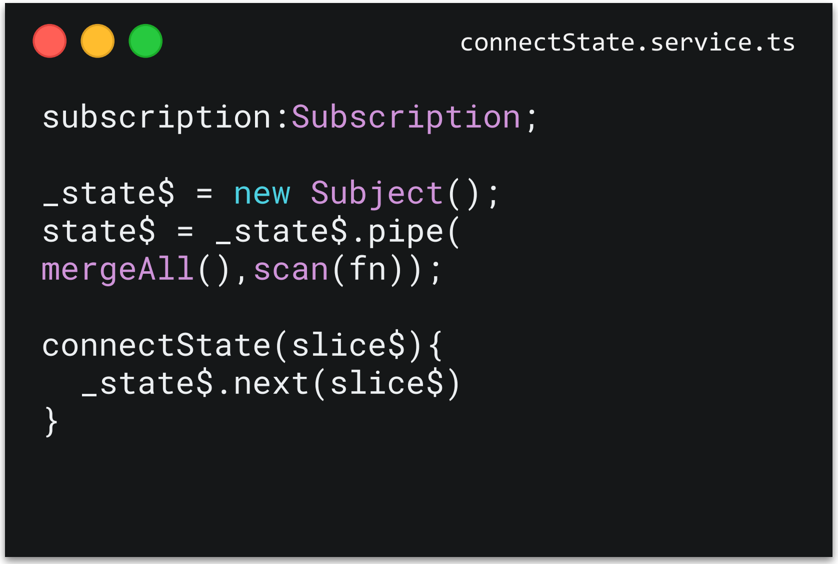

routerState r$:
globalState g$:
inputBinding i$:
setter:
n
subject:
scan(fn,0)
fn
state$:

Solution

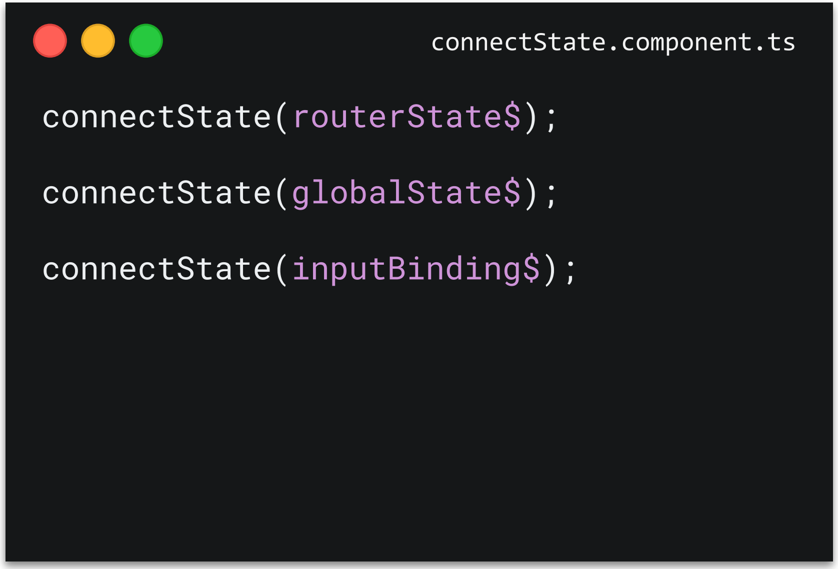
Use Higher Order Operators


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

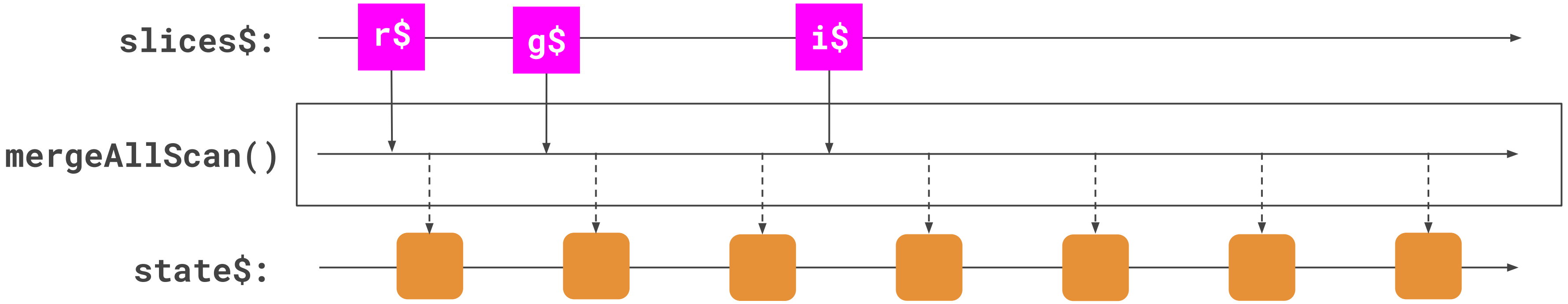

slice$:
R$
G$
I$
mergeAll()
scan(fn,0)
fn
state$:


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$);
```



Ephemeral

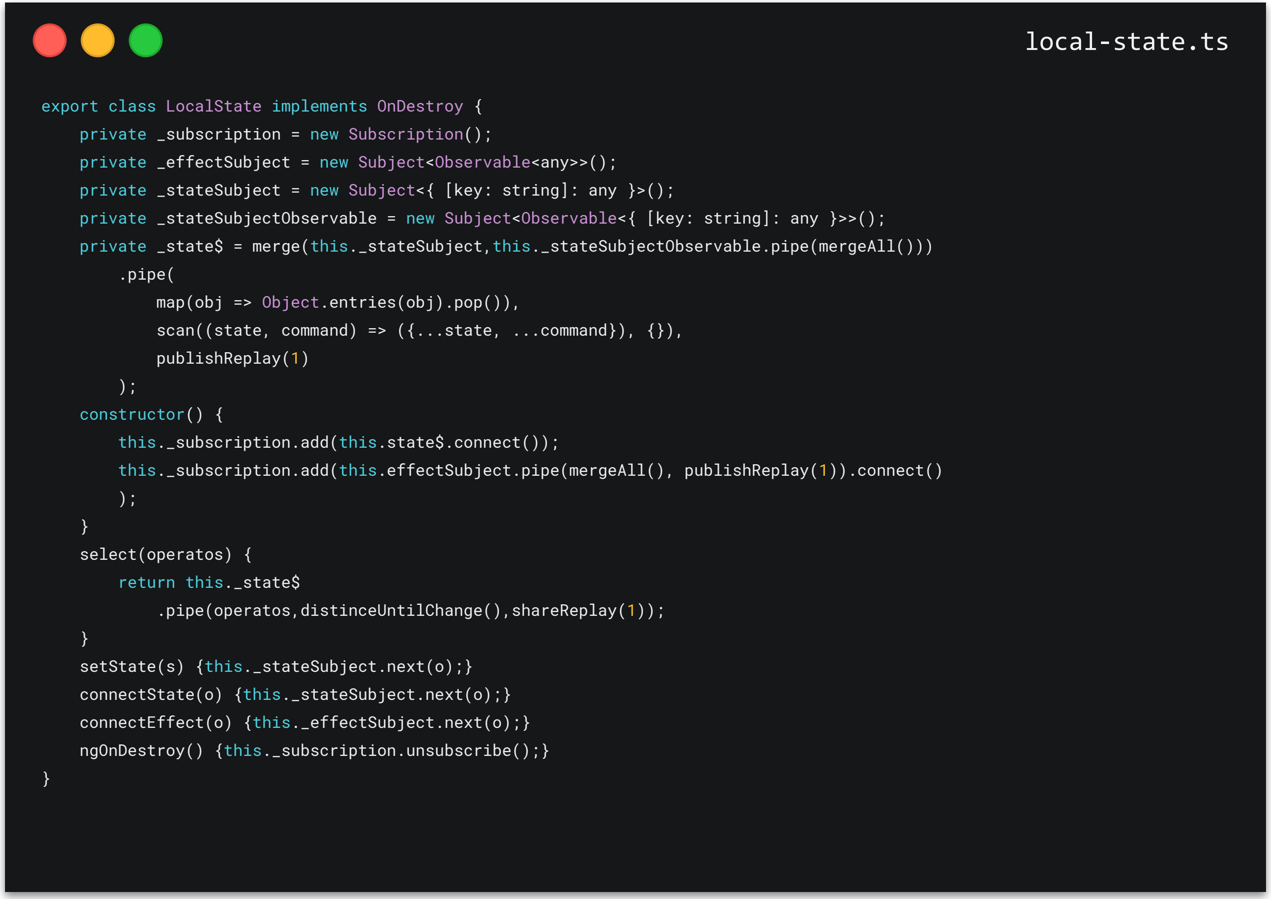
State Management

First Draft

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

Gang Of Four

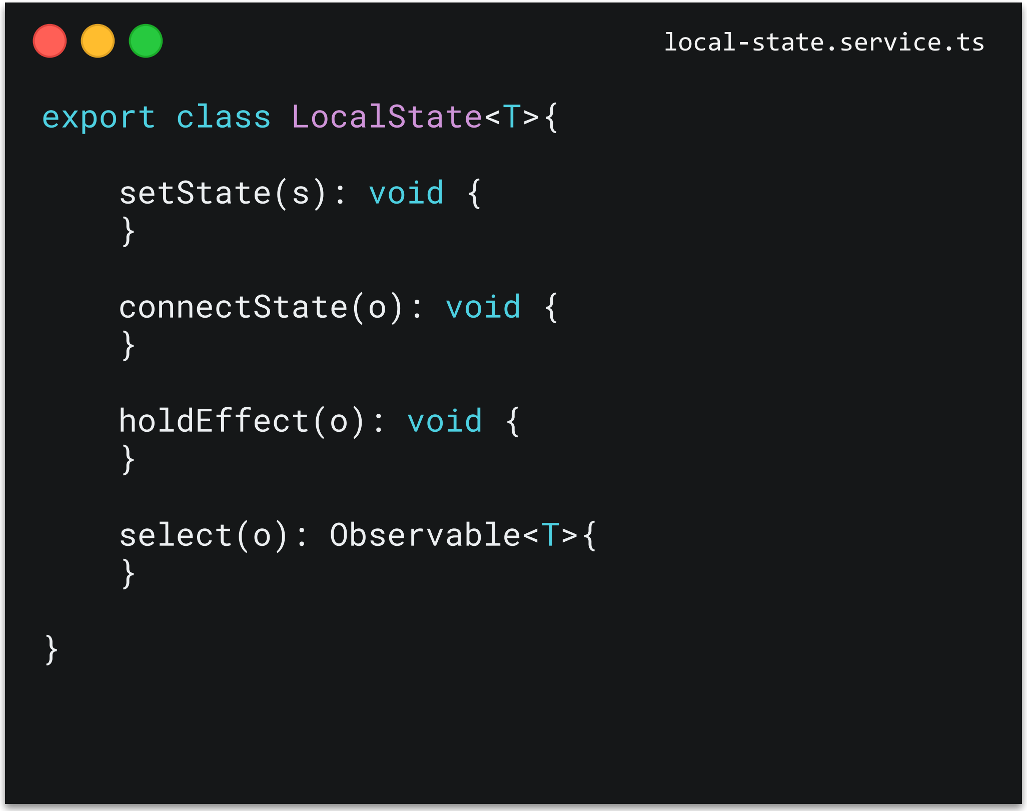
Service source



local-state.ts

```
export class LocalState implements OnDestroy {
    private _subscription = new Subscription();
    private _effectSubject = new Subject<Observable<any>>();
    private _stateSubject = new Subject<{ [key: string]: any }>();
    private _stateSubjectObservable = new Subject<Observable<{ [key: string]: any }>>();
    private _state$ = merge(this._stateSubject,this._stateSubjectObservable.pipe(mergeAll()))
        .pipe(
            map(obj => Object.entries(obj).pop()),
            scan((state, command) => ({...state, ...command}), {}),
            publishReplay(1)
        );
    constructor() {
        this._subscription.add(this.state$.connect());
        this._subscription.add(this.effectSubject.pipe(mergeAll(), publishReplay(1)).connect()
        );
    }
    select(operatos) {
        return this._state$
            .pipe(operatos,distinceUntilChange(),shareReplay(1));
    }
    setState(s) {this._stateSubject.next(o);}
    connectState(o) {this._stateSubject.next(o);}
    connectEffect(o) {this._effectSubject.next(o);}
    ngOnDestroy() {this._subscription.unsubscribe();}
}
```

Local State Interface



local-state.service.ts

```
export class LocalState<T>{

    setState(s): void {
    }

    connectState(o): void {
    }

    holdEffect(o): void {
    }

    select(o): Observable<T>{
    }

}
```

Demo Time!

Thanks for your time!

If you have any questions just ping me!

And book my consulting! ;)

Lib: github.com/BioPhoton/rxjs-state
Demo: research-reactive-ephemeral-state-in-component-oriented-frontend-frameworks
Research: dev.to/rxjs/research-on-reactive-ephemeral-state-in-component-oriented-frameworks-38lk

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