# Observable - An Introduction

- Observables are push based

- Observables are cold [by default]

- Observables can emit multiple values

- Observables can deliver both synchronous and asynchronous values

- Observables can be cancelled

Creational Operators

- Stand alone functions to create observables

- Sources can be :
    - **event** : fromEvent(document."click)
    - **request**: of("http://api.github.com/users/octocat")
    - **timer**: interval(1000)
    - **static data** : from ([1,2,3,4,5])
    - combination of other **observable** sources
    - + more...

Creational Operators

interval

range

fromEvent

of

Observable

timer

from

Pipeable Operators
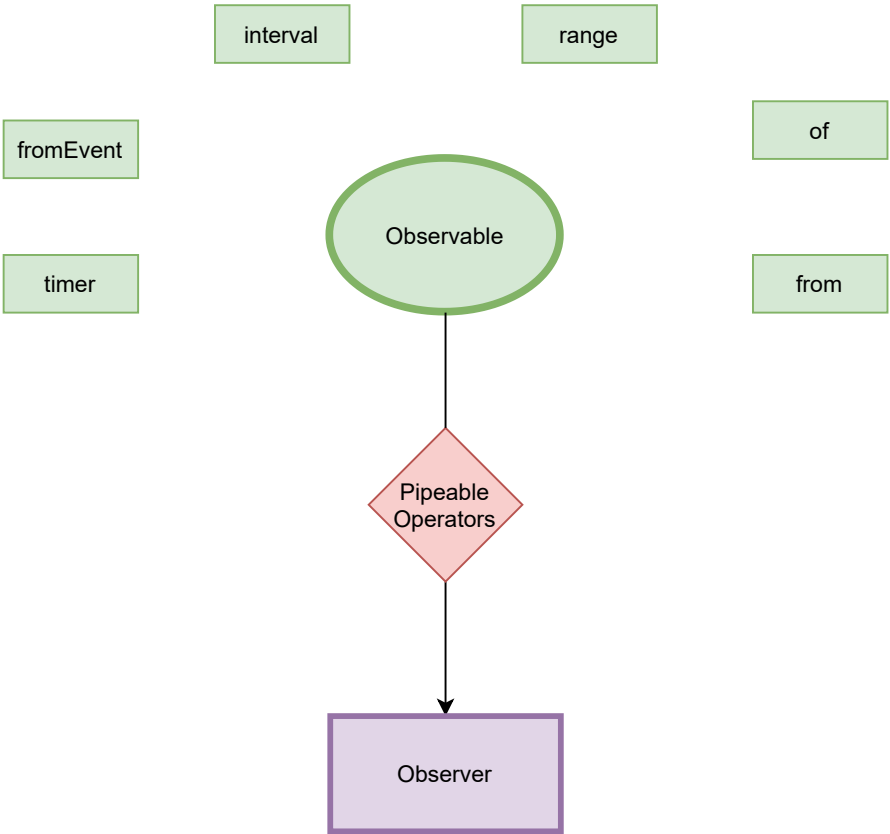
Observer

Creational Operators

fromEvent	creates observables from DOM events

of	creates observables from static values

from	creates observables from Array, iterators and Promises

interval / timer	Emits item based on a duration

Pipeable Operators

- Operators are the power behind RxJS, letting you more easily compose complex asynchronous code

- Operators can be applied by including them in the pipe() method.

- Operators return a new observable without modifying the input observable.

- A core set of Operators can solve the majority of use case, while others can be pickup up as the situation arises.

Filtering Operators

take	Emits a set number of values from stream

takeWhile	Completes a stream when a condition is met

takeUntil	Completes a stream based on another stream

distinct	Ignores NON unique values

filter	Ignores NOT needed values

reduce	Accumulates data over time

scan	Managing state changed incrementally

Rate Limiting Operators

debounceTime	Takes the latest value after a pause

throttleTime	Ignores values between windows/gap

sampleTime	Sample a stream on a uniform duration

auditTime	Audit a stream after a duration once event occurs

Transforming Operators

mergeMap	Flattening inner observable as they occurs

switchMap	Switch to a new observable on emissions

concatMap	Subscribe to observables in order

exhaustMap	ignore emissions when an inner observable is active

Combination Operators

startWith / endWith	Add values to the stream at start / end

concat	Queue observable emissions

merge	Combines multiple active observables

combineLatest	Receives the latest values from multiple observables on emissions

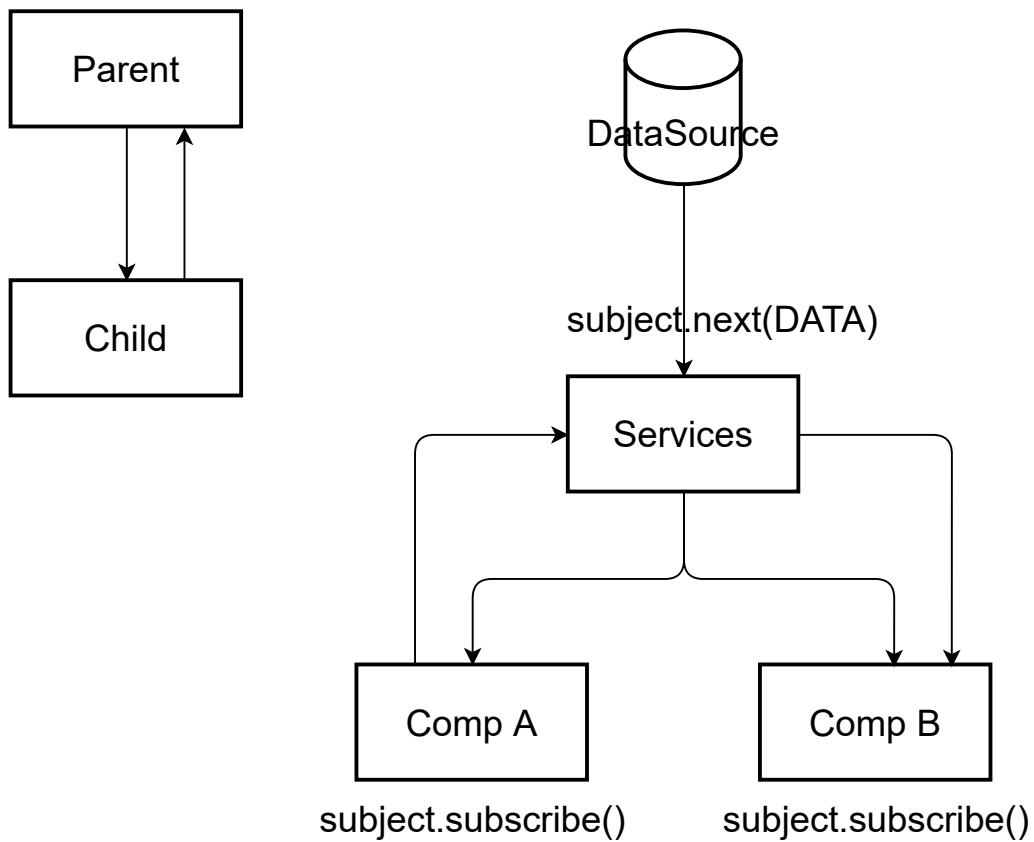|	
forkJoin	Receives the latest values from multiple observables on completion

Subjects are both - observable and observer

subscribe()

next()
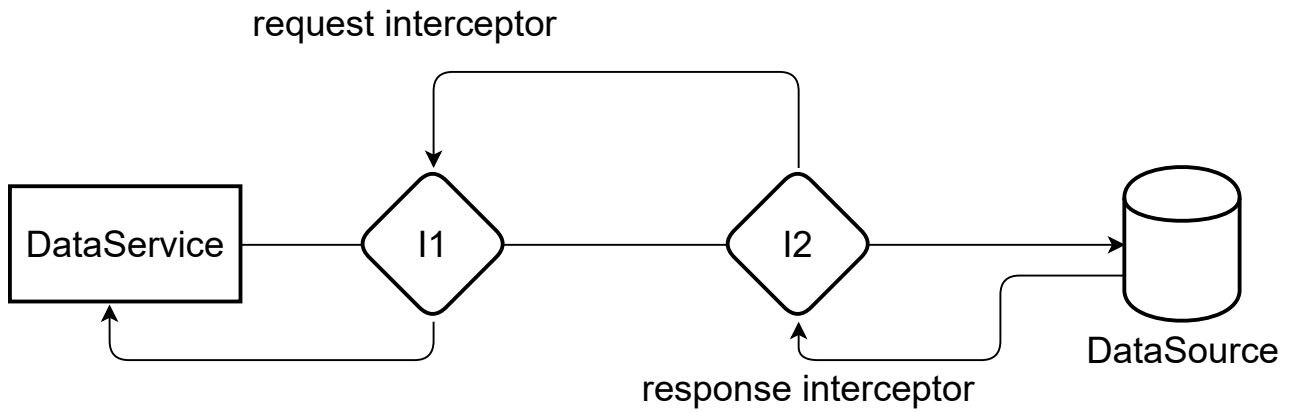
error()

pipe()

Subject

complete()

Observer

Observer

Observer

Types of Subject

```
                        ┌─────────────┐
                        │   Subject   │
                        └──────┬──────┘
          ┌────────────────────┼────────────────────┐
          ▼                    ▼                     ▼
   ┌─────────────┐      ┌─────────────┐      ┌─────────────┐
   │  Behaviour  │      │   Replay    │      │    Async    │
   │   Subject   │      │   Subject   │      │   Subject   │
   └──────┬──────┘      └──────┬──────┘      └──────┬──────┘
          ▼                    │                    ▼
  ╱─────────────────╲          │          ╱─────────────────╲
 ╱ Deliver the       ╲         │         ╱ Emits the final   ╲
 ╲ starting value to ╱         │         ╲ value on          ╱
  ╲ subscriber      ╱          ▼          ╲ completion      ╱
   ╲───────────────╱   ╱─────────────────╲ ╲───────────────╱
                      ╱ Replay history to  ╲
                      ╲ new Subscribers    ╱
                       ╲─────────────────╱
```
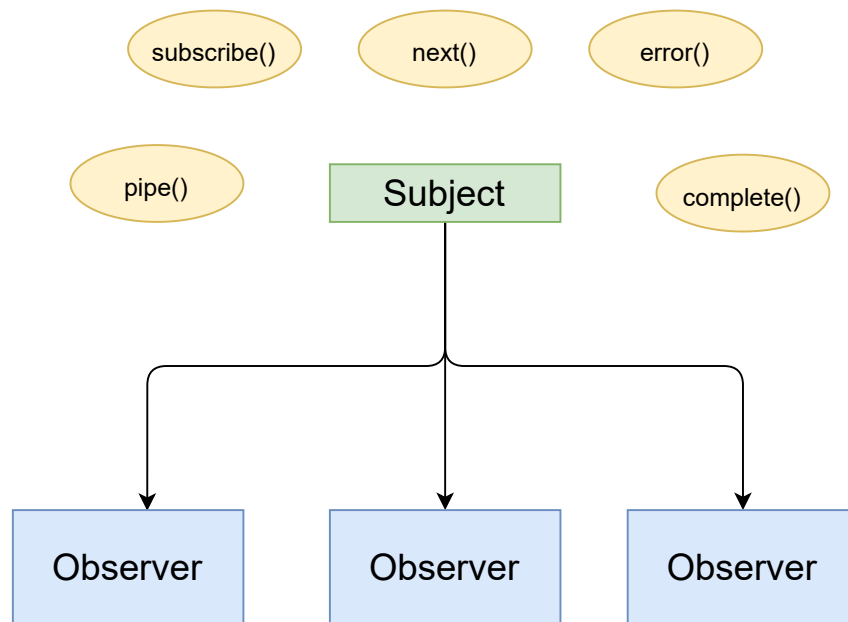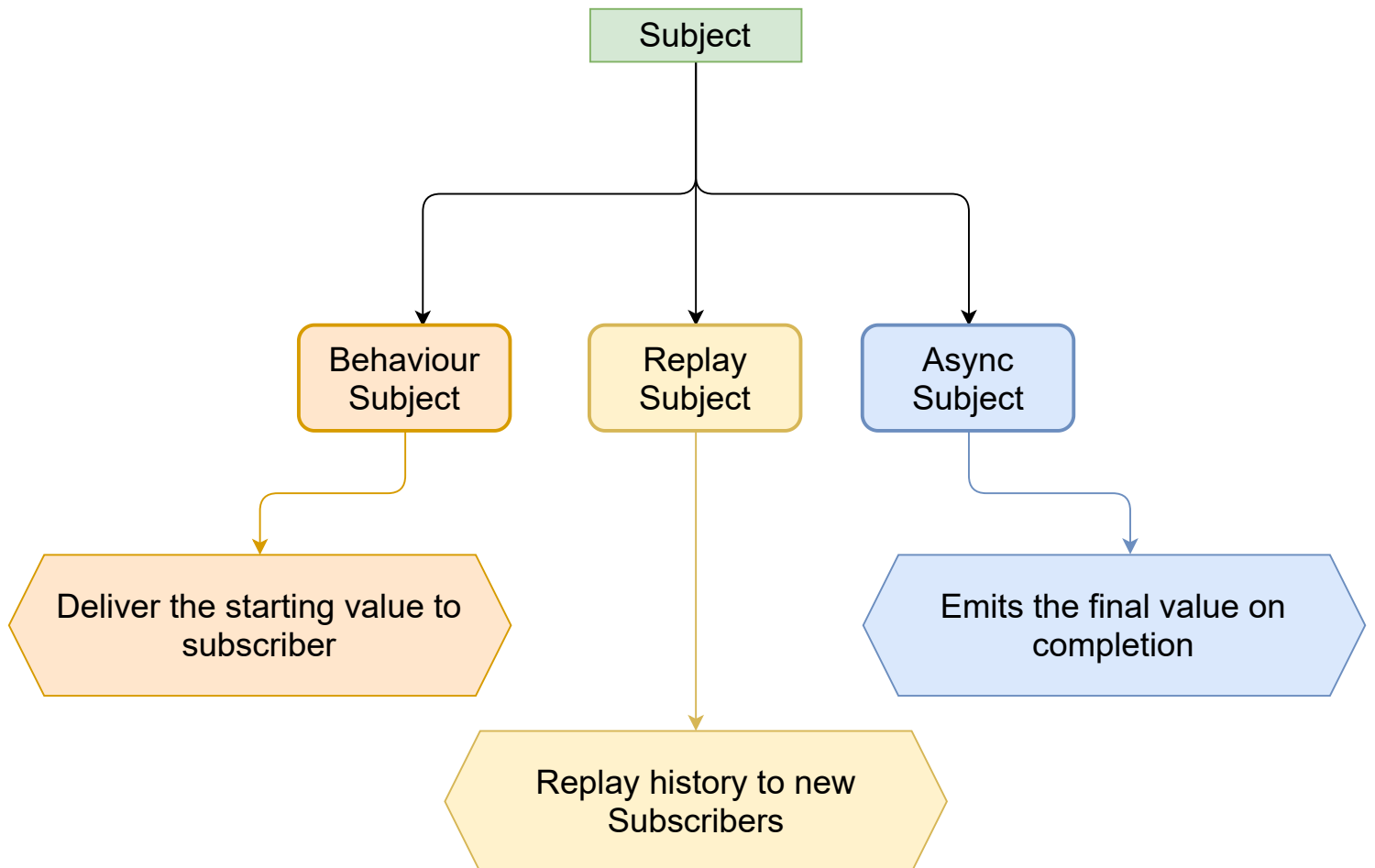
Understanding RxJS Subject

```
Parent
  │ ↑
  ↓ │
Child
```

```
DataSource
    │
    │ subject.next(DATA)
    ↓
 Services
```

Comp A

Comp B

subject.subscribe()   subject.subscribe()

Interceptors

request interceptor

DataService I1 I2 DataSource

response interceptor

What is Redux?

- for State Management
- Pattern for maintaining state
- Building blocks - Store, Reducers, Actions, Effects
- Predictable state container
- JavaScript Library,
- Use redux with any JS framework / library
- React : react-redux
- Vue - Vuex
- Angular - @ngrx/store, @ngrx/effects
- State - Data at that moment

Do I need Redux? Why Redux?

- Keeping the data in top level component is good enough? - No

- Maintaining State in Angular Services being complex? - Yes

- Is your data changing very frequently? - Yes

....Go For Redux

Redux Building Blocks -

- Action : can be triggered by any Event (click, XHR, user interaction etc), carries the payload/data, defines what happened in your app, "type" Property. Object { type : "", payload : "" }

- Store : stores the data/State to maintain, container for state.

- State : Simple JS Object, Single source of Truth,

- Reducers : pure functions, no side-effects

(state, action) => state

Redux 3 Principles-

- Single Source of Truth

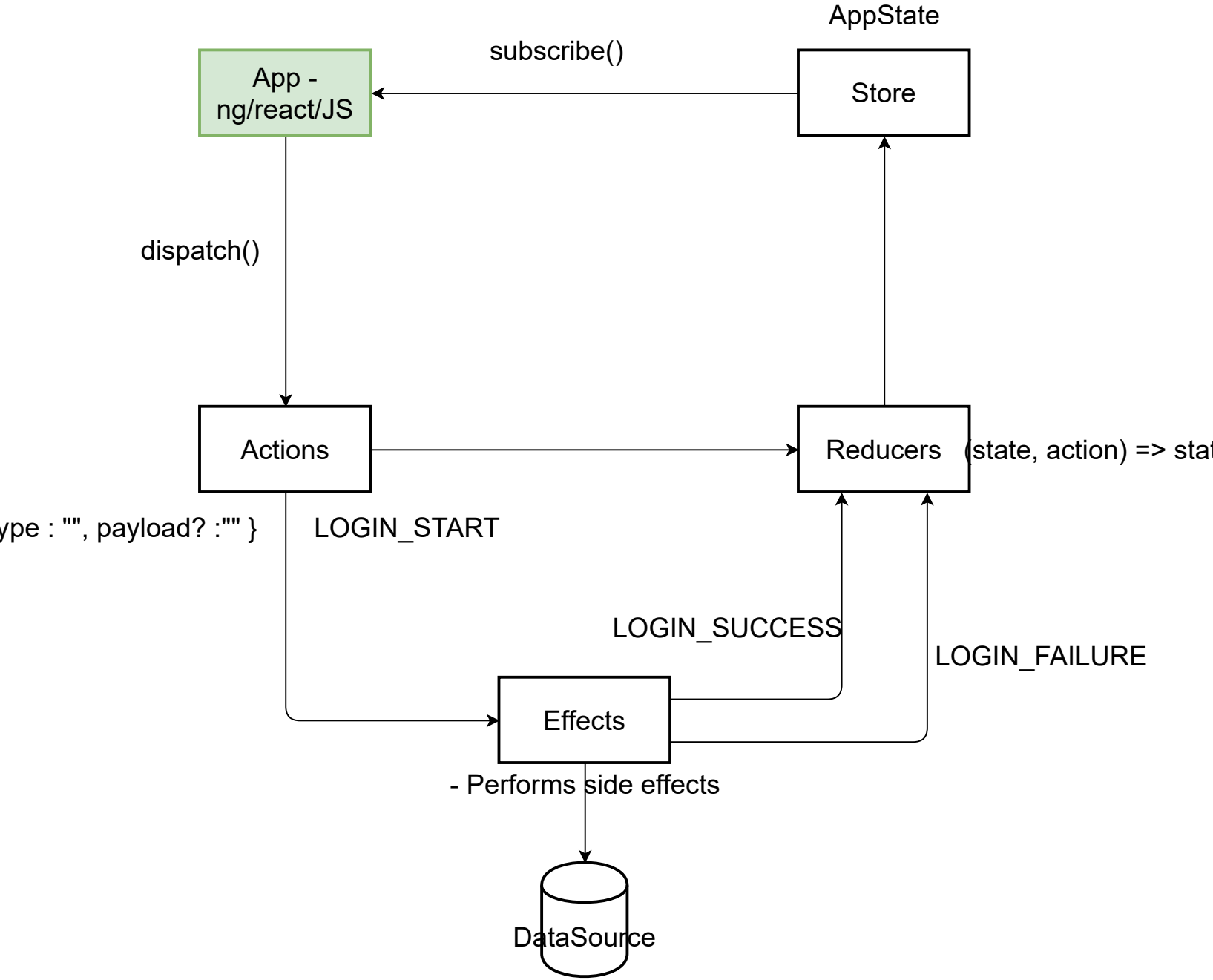- State is immutable : not changeable, Never change the existing state

- State should be updated by Pure Functions (Reducers)

const add = (a, b) => a + b;

XHR Call, Date.now(), Math.random() - side effects

{ ty

Redux Data Flow - Uni-directional Data Flow

AppState

App - ng/react/JS

subscribe()

Store

dispatch()

Actions

Reducers   (state, action) => state

{ type : "", payload? :"" }   LOGIN_START

LOGIN_SUCCESS

LOGIN_FAILURE

Effects

- Performs side effects

DataSource

- Services should be reactive and stateless

Stateful Services

te

Angular Security

The "Angular way" safeguards you from XSS

Use innerHTML with caution

Never use templates generated by concatenating user input

Never use native DOM APIs to interact with HTML elements

Avoid template engines on server-side templates

- XSS
- XSUF/XSRF - token
- XSSI
- Content-Security-Policy - Maintained on Server

-ng eject -

Angular App Optimization Techniques

OnPush Change Detection Strategy

Detach Change Detection, if required

*ngFor Trackby Function

Using Pure Pipes

Lazy Loading

Preloading Modules - Custom Strategy

Using Web-Workers for complex calculations

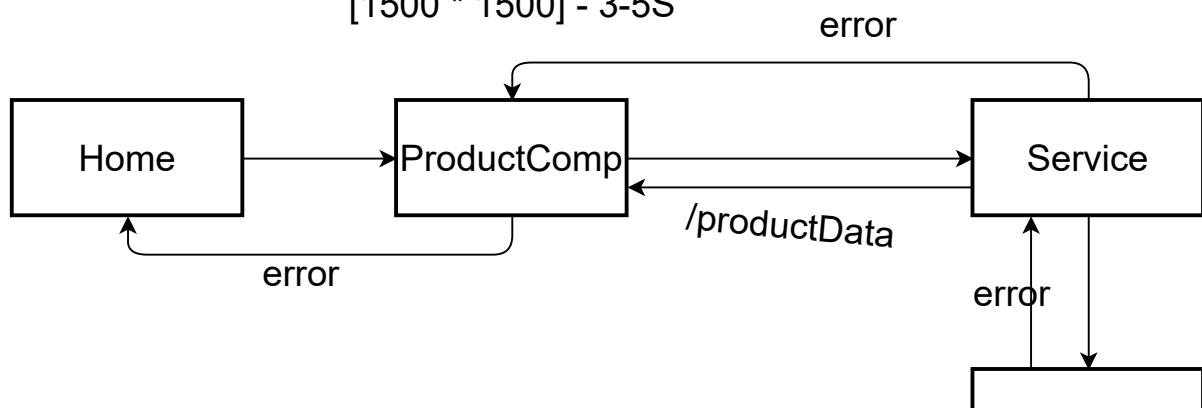Using Resolve Guards on Component loading
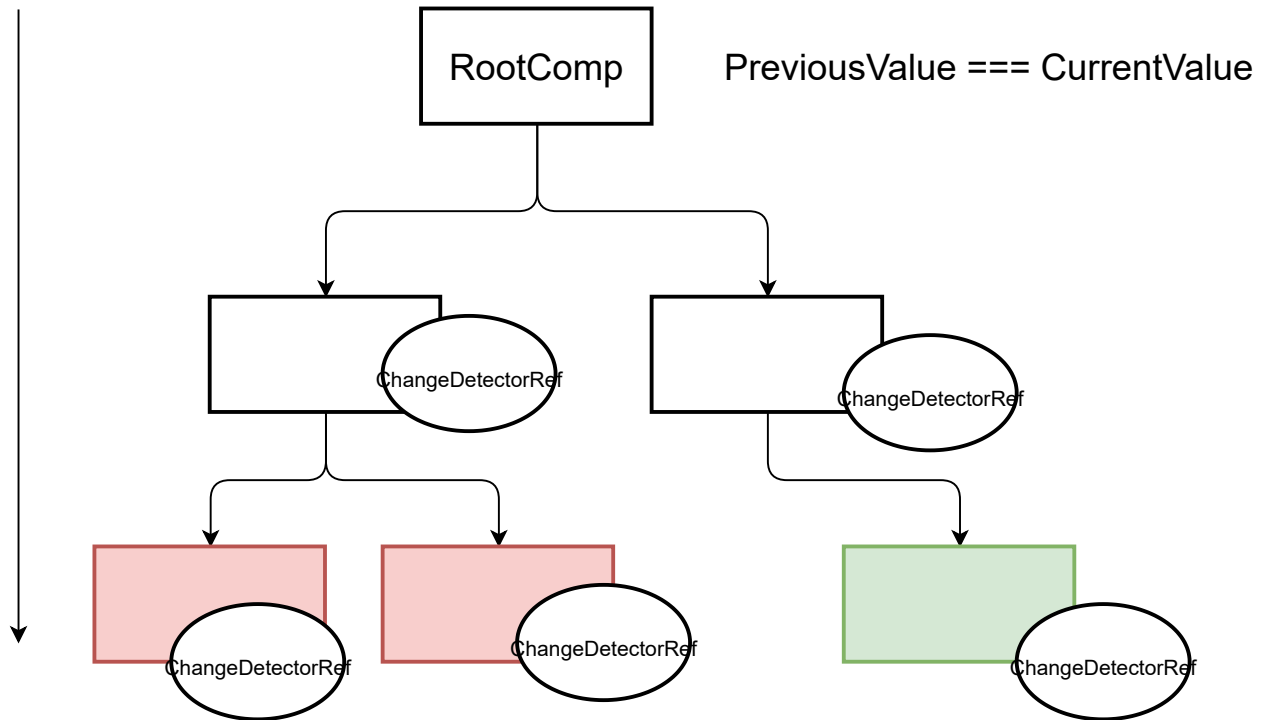
Unsubscribing the Observable
ngOnDestroy()

1. Landing Page Modules - Eagerly loaded
2. Lazy Modules - Frequently used by user - Preloaded
3. Modules on demand

> ng add web-worker app

[1500 * 1500] - 3-5S

```
                              error
            ┌─────────────────────────────────────┐
            │                                      │
   ┌────────┴──┐        ┌──────────────┐       ┌───┴──────┐
   │  Home     │───────▶│  ProductComp │──────▶│  Service │
   │           │        │              │◀──────│          │
   └───────────┘        └──────────────┘  /productData └──┬──┘
        ▲                      │                          │
        │        error         │                        error
        └──────────────────────┘                          │
```

Cloud/Server

RootComp

PreviousValue === CurrentValue

ChangeDetectorRef

ChangeDetectorRef

ChangeDetectorRef

ChangeDetectorRef

ChangeDetectorRef
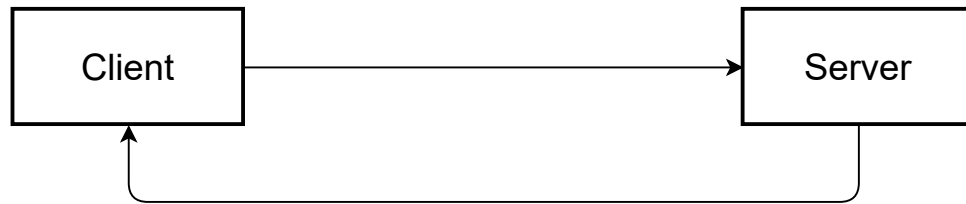
ChangDetectionStrategy.onPush

- Principle of Immutability -

let user = { email : "test@test.com"}
user.email = "john@test.com"
user = {email : "jenny@test.com"}

ChangeDetectorRef - detech, reattach, detectChanges

Socket / Network Programming

Client ———————→ Server

Client ←——————— Server

- Disconnected Connection Architecture

Angular App   NodeSocketServer

Client ———————→ Server

Client ←——————— Server

Connected Architecture
Duplex Communication
socket.io-client

- Express
- Socket.io

Client ←——————— Server

handle event on client - on()   emit the event on Server - emit()