Effective State Management without Redux

(I'M NOT READY FOR NGRX!)

HELLO!

Jesse Sanders

Founder & CEO, BrieBug

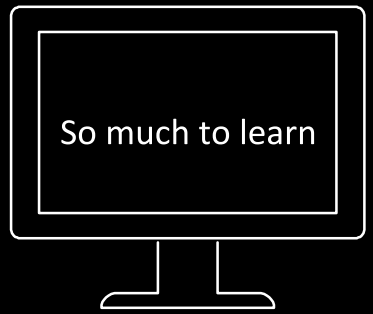
jesse.sanders@briebug.com

@JesseS_BrieBug

- NgRx
- NgRx Actions
- NgXs
- NgRx Data
- MobX
- ?

So many coding options

NgRx



- Actions
- Reducers
- Effects
- Selectors
- Architecture

Do I Need NgRx?

Do I need NgRx?

Is my application complex enough?

Do I need NgRx?

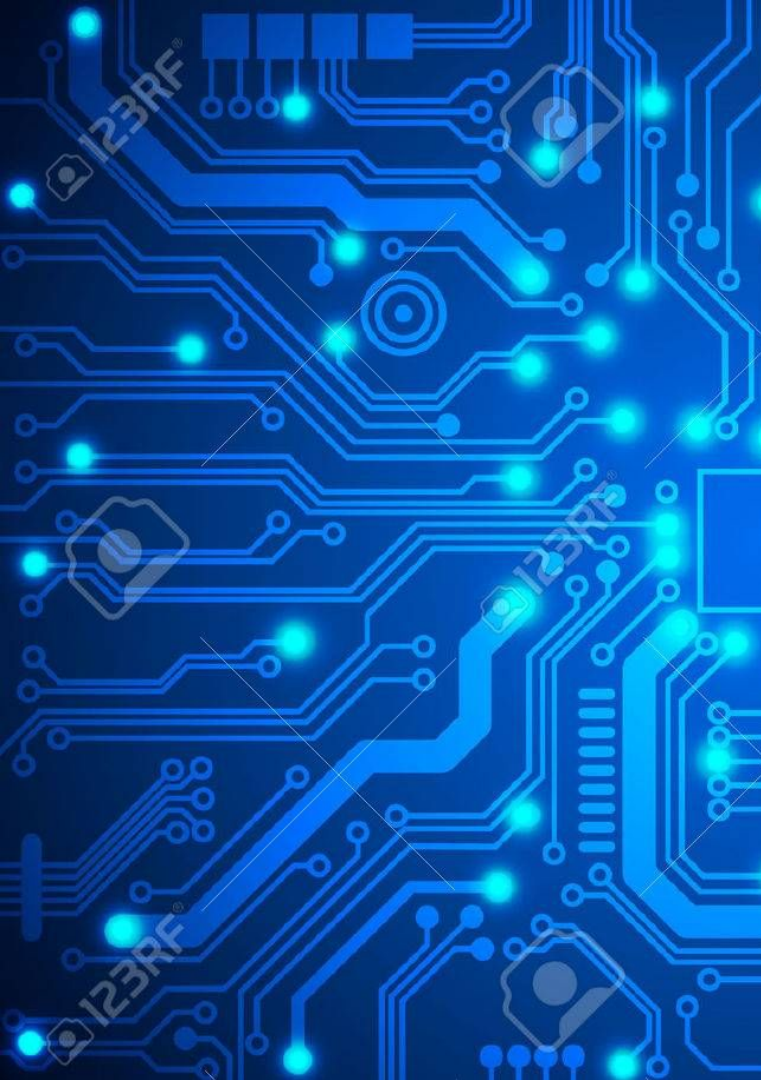
I want to start

SIMPLE

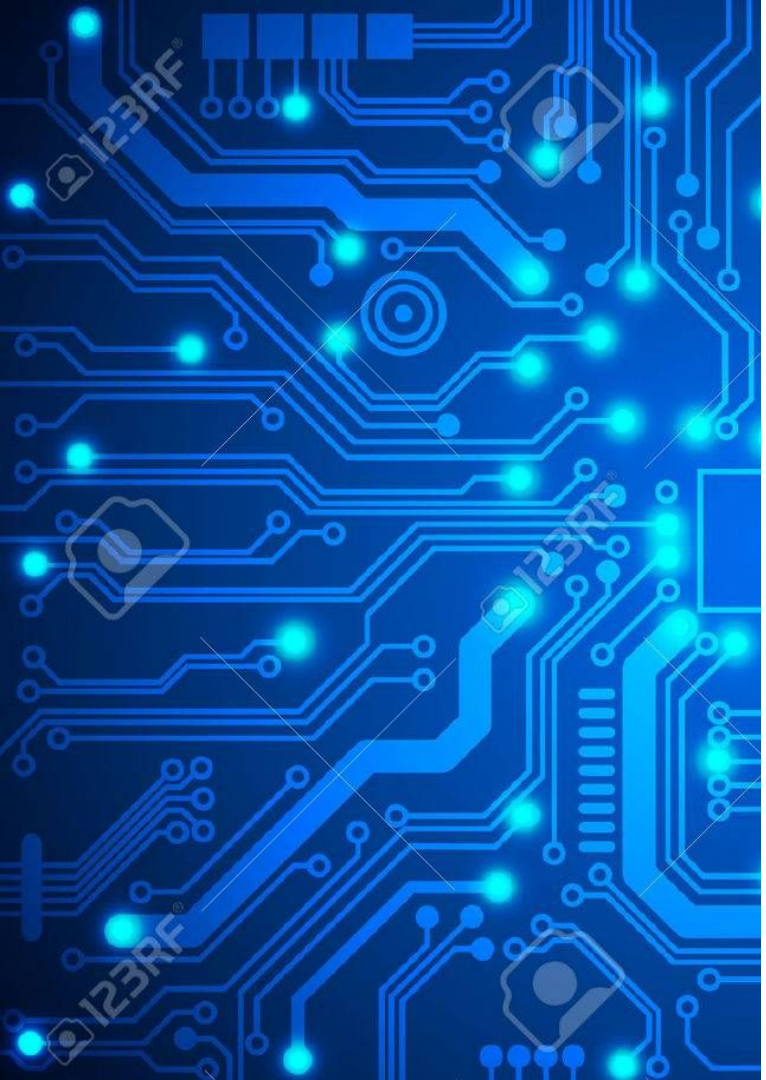
Do I need NgRx?

Upgrade path?

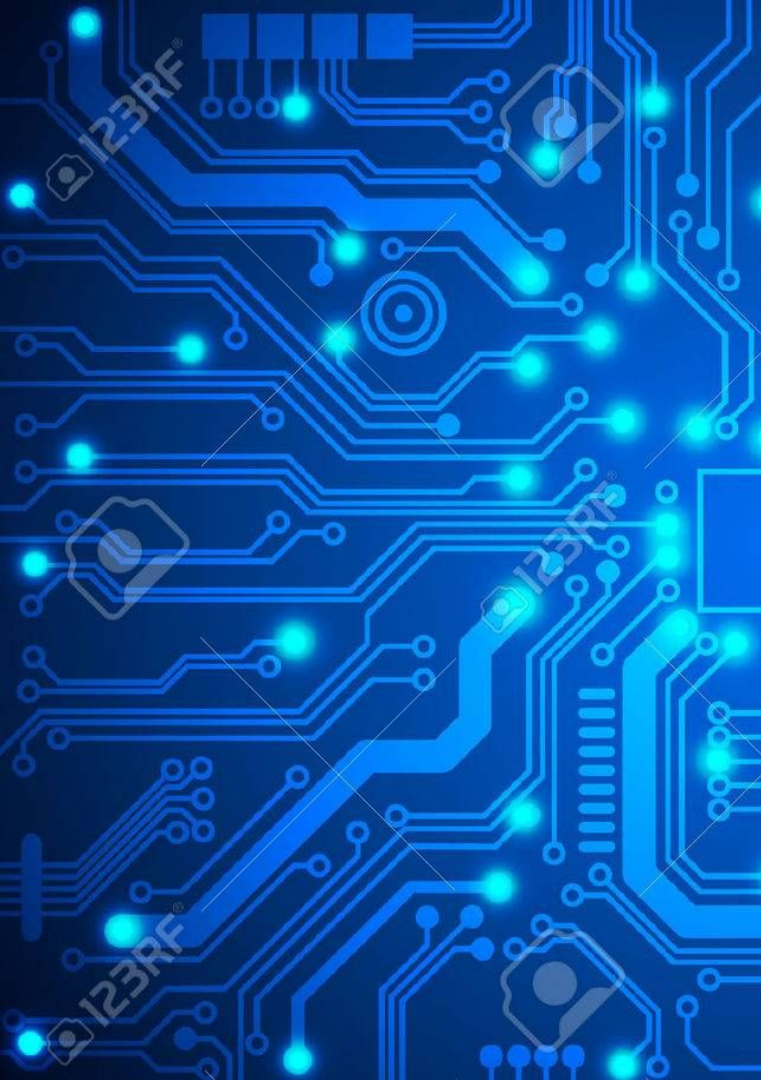
Arch Goals?



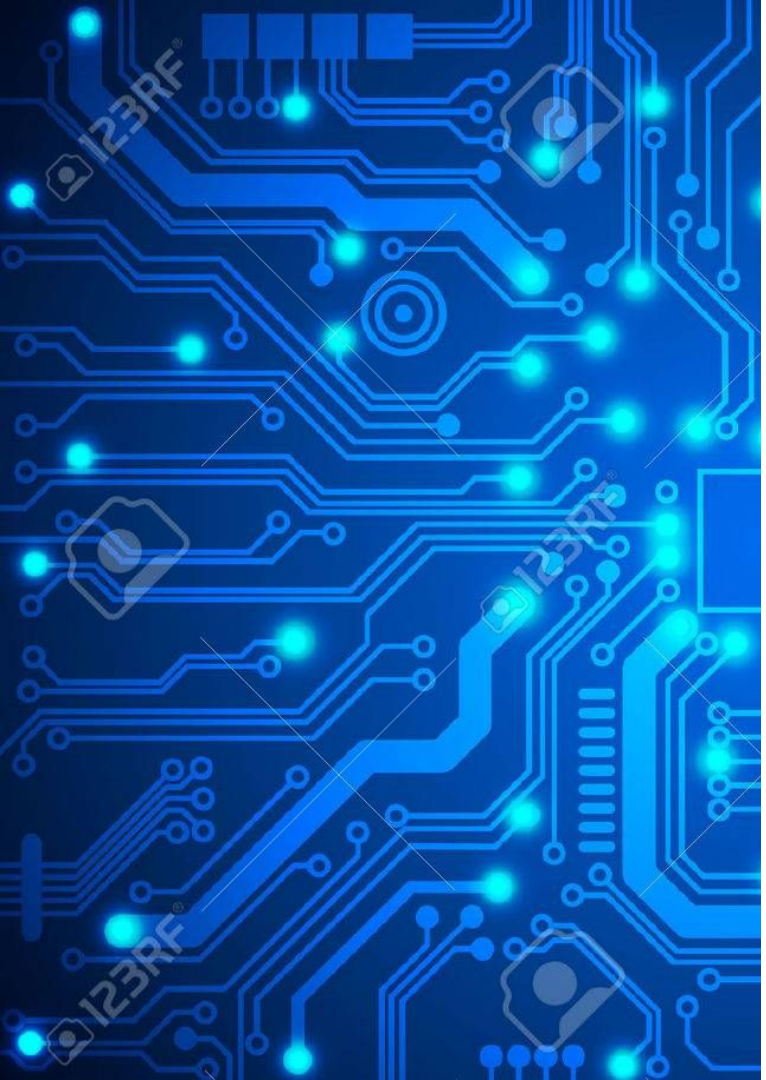
Maximize code reuse



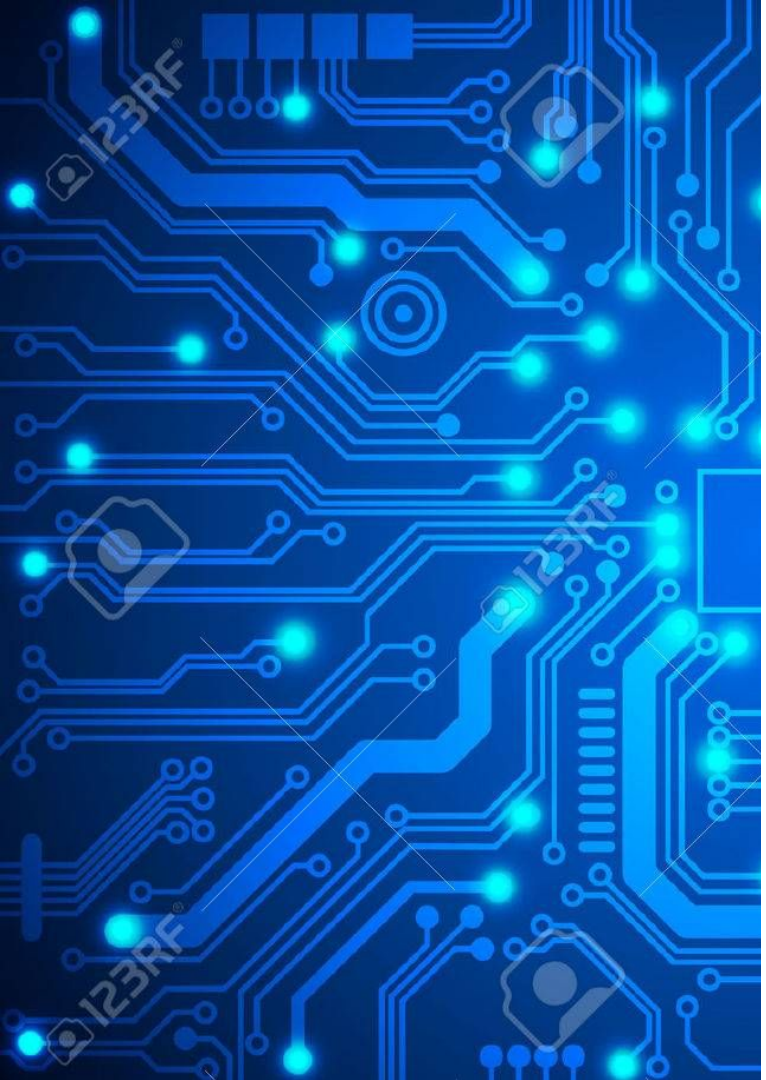
Share Data



Stream Data



Reduce Complexity



Scalability

Core Concepts

Container/ Presenter

Core Concepts

Service with a Subject

Container Component

- Top level feature/page
- Contain child components

What do containers do?

Control page flow
Load/Save data
Handle child events

```
export class UsersComponent implements OnInit {
 users$: Observable<User[]>;
 constructor(private userService: UserService) {}

 ngOnInit() {
   this.users$ = this.userService.loadAll();
 }

 onUserClick(user) {
   console.log("User Selected:" + user);
 }
}
```

Containers

- Loading data from service
- Handles events from child

```
<app-user-list
    [users]="users$ | async"
    (userClick)="onUserClick($event)">
</app-user-list>
```

Containers

- Passes data to child
- Handles events from child

Presentation Component

- Present data
- Make things pretty
- Know nothing about services or stores.

What do presentation components do?

Receive data via inputs
Display data
Raise events to parents
Defer everything

```
export class UsersListComponent implements OnInit {
  @Input() users: User[];

  @Output() userClick = new EventEmitter<User>();

  onUserClick(user: User) {
    this.userClick.emit(user);
  }
}
```

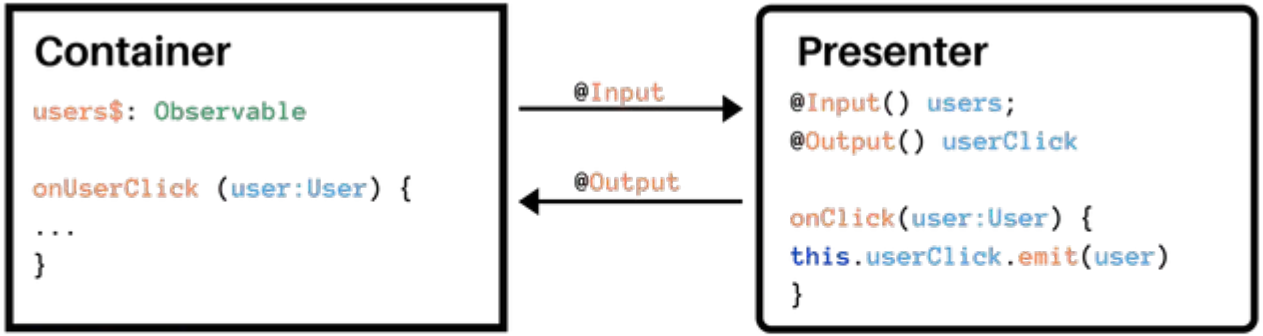
Presenters

- Receive data via Inputs
- Raises event to parent

```
<div *ngFor="let user of users">
    <span (click)="onUserClick(user)">
        {{user.name}} - {{user.role}}
    </span>
</div>
```

Presenters

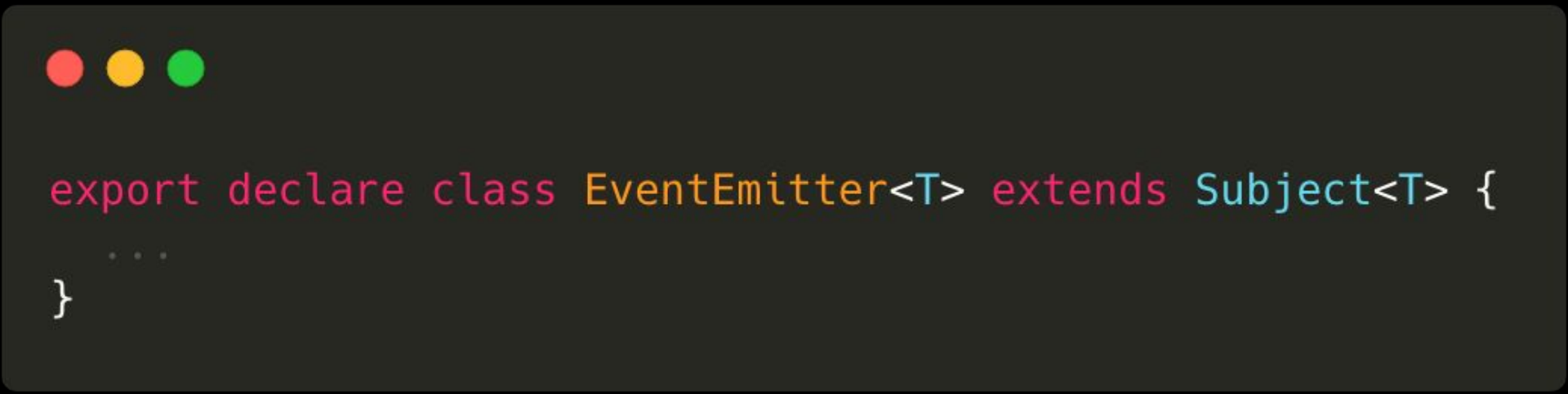
- Handles click event locally to pass up to parent




```
<app-users
    [users]="users$|async"
    (userClick)="onUserClick($event)">
</app-users>
```

Container / Presenter

What if I have deeply nested components?



EventEmitter

Observable
Trade off service reference/deep event emitters

What if my component is really simple?

Container Presenter?

Do it anyway-best practice
Reusable components
Clear upgrade path
Async Pipe
Reactive vs Imperative
KISS

Core Concepts

Service with a Subject

```
export class UsersComponent implements OnInit {
 users$: Observable<User[]>;
 ngOnInit() {
   this.users$ = this.userService.loadAll();
 }
}
// service
export class UserService {
  loadAll() {
    return this.httpClient.get<Widget[]>("/api/users")
  }
}
```

Containers and Regular Services

What is Service with a Subject?

Regular Service
Local behavior subject
Service API calls return void
Set local variable with value

Service with a Subject

- Declare BehaviorSubject
- Service calls set local variable

```
@Injectable({providedIn: "root"})
export class WidgetService {
 widgets$ = new BehaviorSubject<Widget[]>([]);

 constructor(private httpClient: HttpClient) {}

 loadAll() {
   this.httpClient
     .get<Widget[]>("/api/widgets")
     .subscribe(widgets => this.widgets$.next(widgets));
 }
}
```

Protect the Subject

```
@Injectable({providedIn: "root"})
export class WidgetService {
 private mwidgets$ = new BehaviorSubject<Widget[]>([]);

 get widgets$(): Observable<Widget[]> {
    return this.mwidgets$.asObservable();
 }

...
   this.httpClient
     .get<Widget[]>("/api/widgets")
     .subscribe(widgets => this.mwidgets$.next(widgets));
```

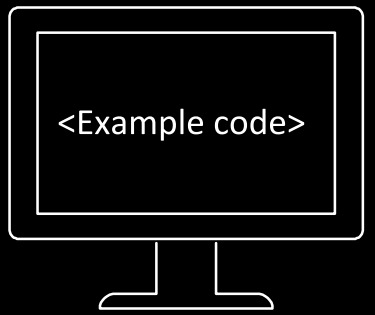
```
export class UsersComponent implements OnInit {
 users$: Observable<User[]>;
 constructor(private userService: UserService) {}

 ngOnInit() {
   this.users$ = this.userService.users$;
   this.userService.loadAll();
 }
}
```

Service with a Subject Containers

- User Model
- User Service
- Dashboard Container
- Navbar
- User List
- Permissions List

Let's look at some code



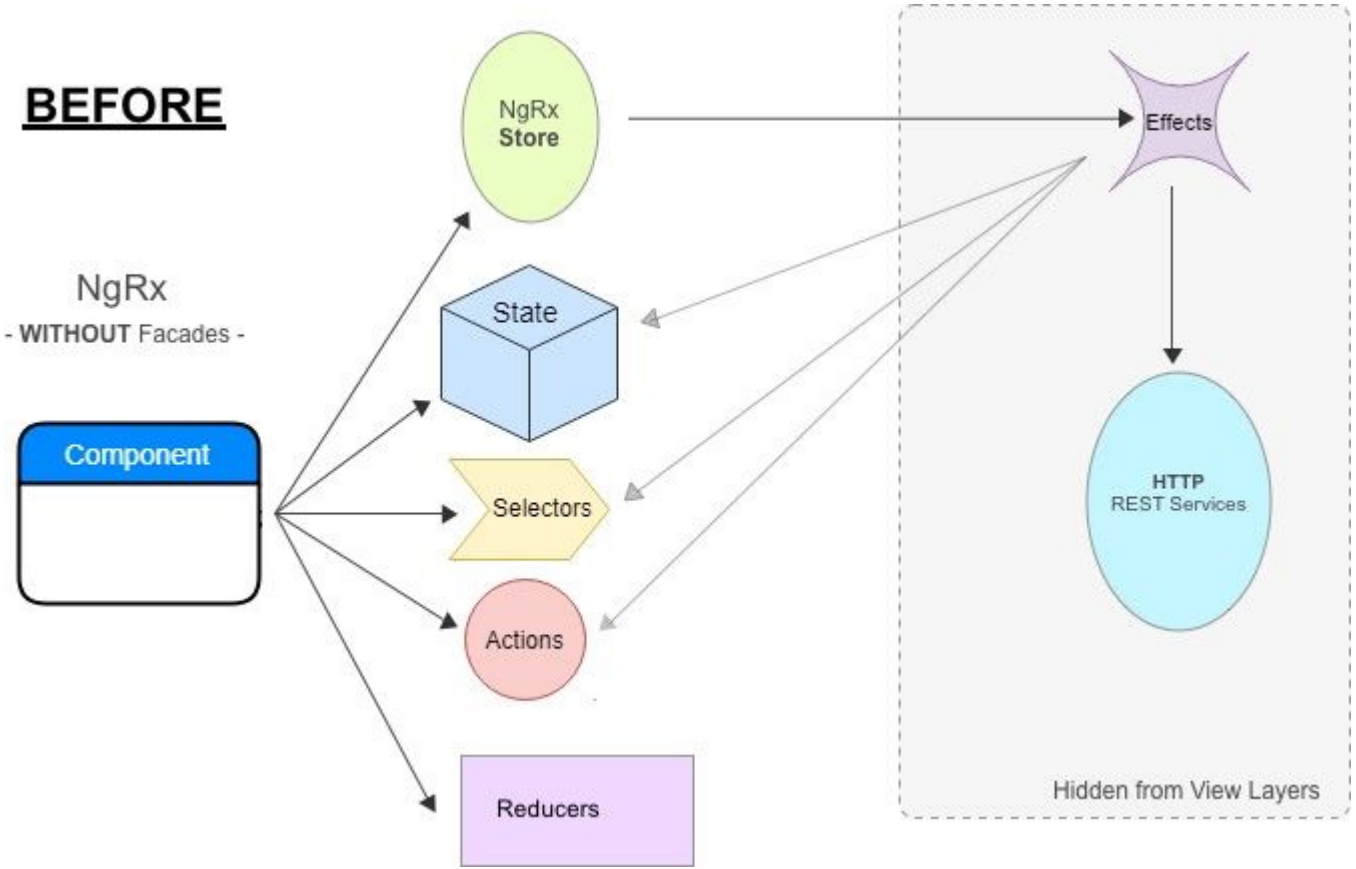
NgRx + Facade

How do I Upgrade?

NgRx + Facade

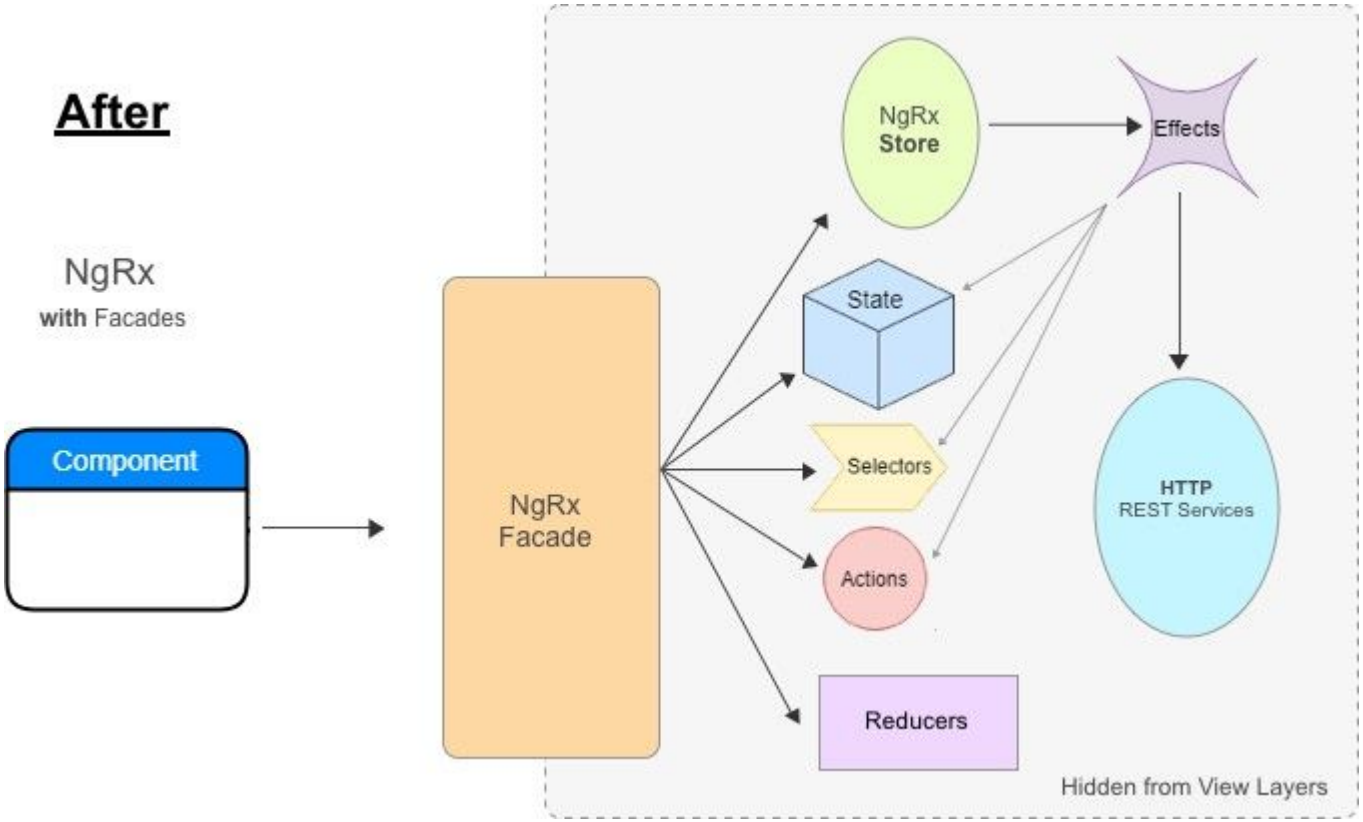
Thomas Burleson
@ThomasBurleson

How do I Upgrade?


BEFORE
NgRx
- WITHOUT Facades -
Component
NgRx
Store
State
Selectors
Actions
Reducers
Effects
HTTP
REST Services
Hidden from View Layers

After

NgRx
with Facades



```
@Injectable()
export class UsersService {
 users$ = this.store.select(usersSelect.getAllUsers);

 constructor(private store: Store<UsersState>) {}

 loadAllUsers() {
   this.store.dispatch(new LoadUsers());
 }
}
```

Facade Pattern

- Replace http calls with dispatch
- Replace behavior subject with store selectors

```
export class UsersComponent implements OnInit {
 users$: Observable<User[]>;
 constructor(private userService: UserService) {}

 ngOnInit() {
   this.users$ = this.userService.users$;
   this.userService.loadAllUsers();
 }
}
```

NgRx Facade Container

(Same exact code as Service w/ Subject Container)

Facade

Service w/ Subject
NgRx Data
NgRx
Component code stays same

Summary

- Simple pattern
- Easy to Test
- Component Reuse
- Upgrade Path to NgRx or NgRx Data

Google Developers Experts

Angular Training, Consulting, Experts

THANKS!

Questions?

You can find me at:

@JesseS_BrieBug

Jesse.Sanders@briebug.com