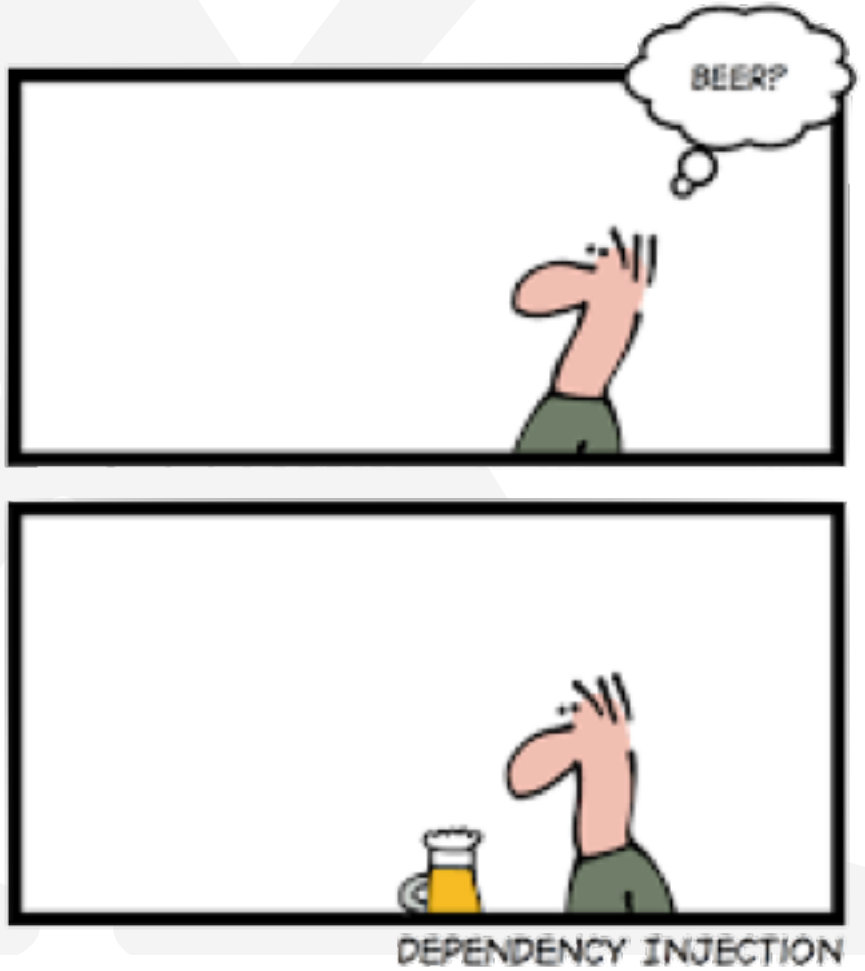
Dependency Injection & Http

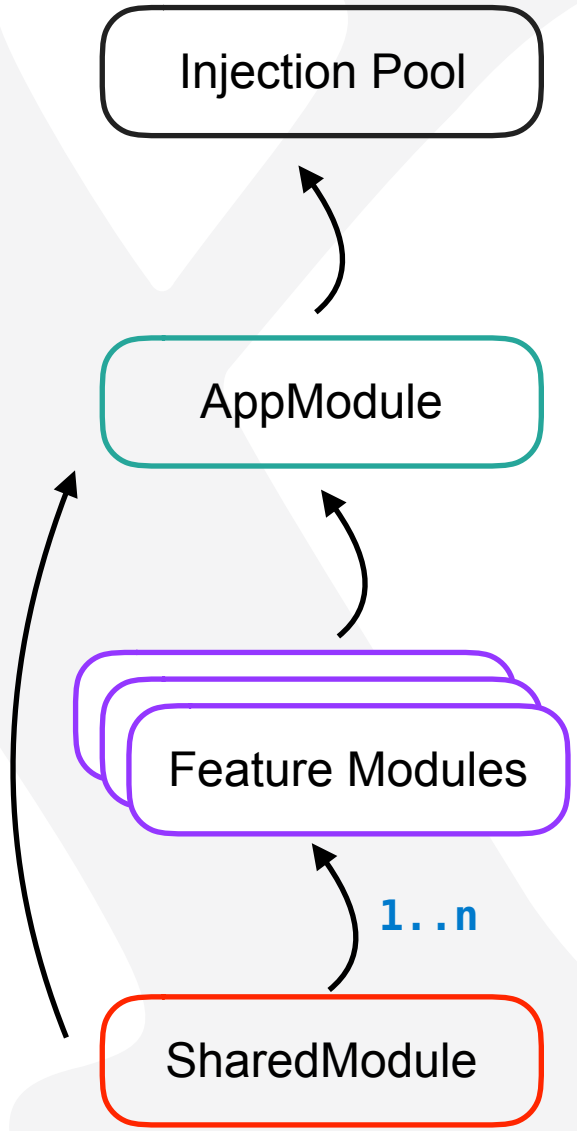
Dependency injection

- Manage dependencies
- No manual instantiation
- Injected by Angular



Dependency injection - Service

- Service "provide" themselves
- Available for all modules
- Providing to components also possible
- Share common tasks
- "Cache"



Creating services

A service is just a TypeScript class

```
export class ContactsService {

 constructor() { }

  fetchContacts(): Contact[] {
    return [ {…}, {…}, {…} ];
  }
}
```

Creating services

@Injectable decorator

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor() { }

  fetchContacts(): Contact[] {
    return [ {…}, {…}, {…} ];
  }
}
```

Using services

constructor injection

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];


  constructor() {

  }

  ngOnInit() {
    this.contacts = [ {…}, {…}, {…} ]
  }

}
```

Using services

constructor injection

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];
  private contactsService: ContactsService;

  constructor(_contactsService: ContactsService) {
    this.contactsService = _contactsService;
  }

  ngOnInit() {
    this.contacts = this.contactsService.fetchContacts()
  }

}
```

Using services

constructor injection

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];
  private contactsService: ContactsService;

  constructor(_contactsService: ContactsService) {
    this.contactsService = _contactsService;
  }

  ngOnInit() {
    this.contacts = this.ContactsService.fetchContacts()
  }

}
```

Using services

constructor injection using "private"

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
     this.contacts = this.ContactsService.fetchContacts()
  }

}
```

Using services

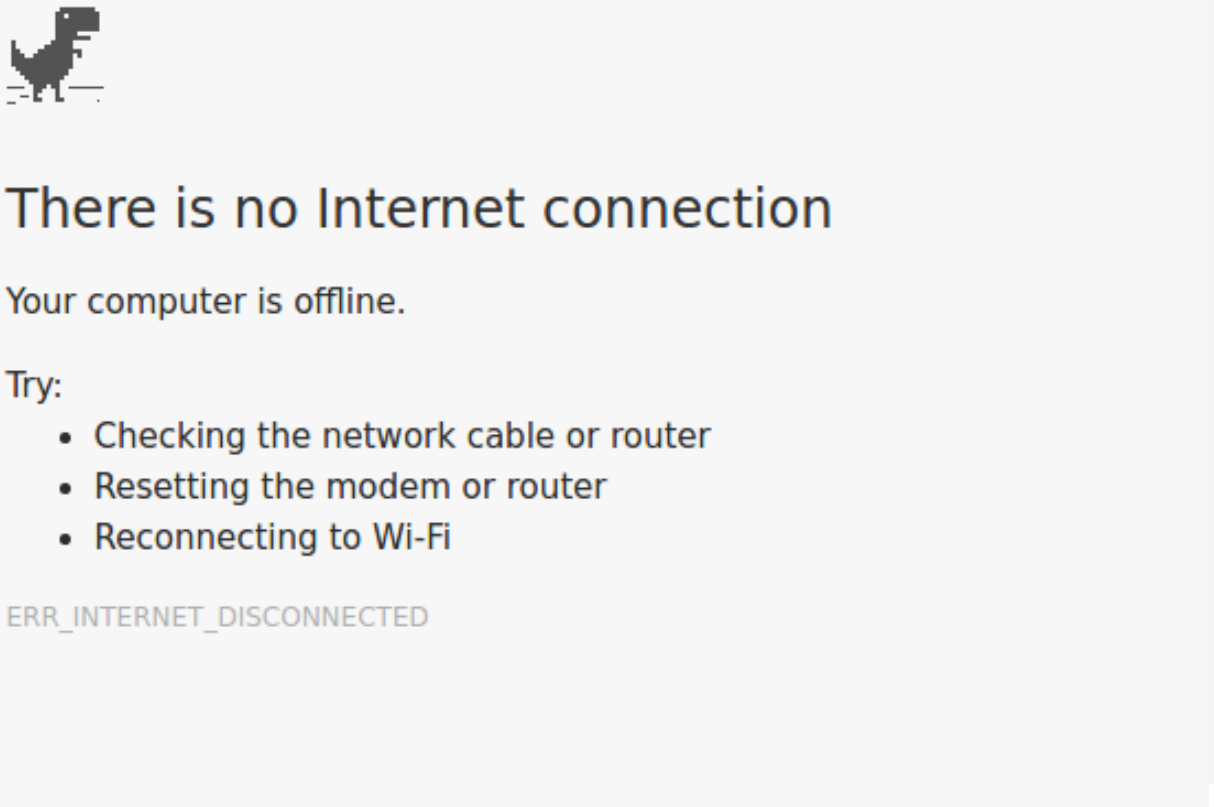
Call with "this"

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) { }

  ngOnInit() {
    this.contacts = this.contactsService.fetchContacts()
  }

}
```

Recap: Dependency injection

- `@Injectable({providedIn: 'root'})`
- Services injectable in all components
- Constructor injections
- private keyword

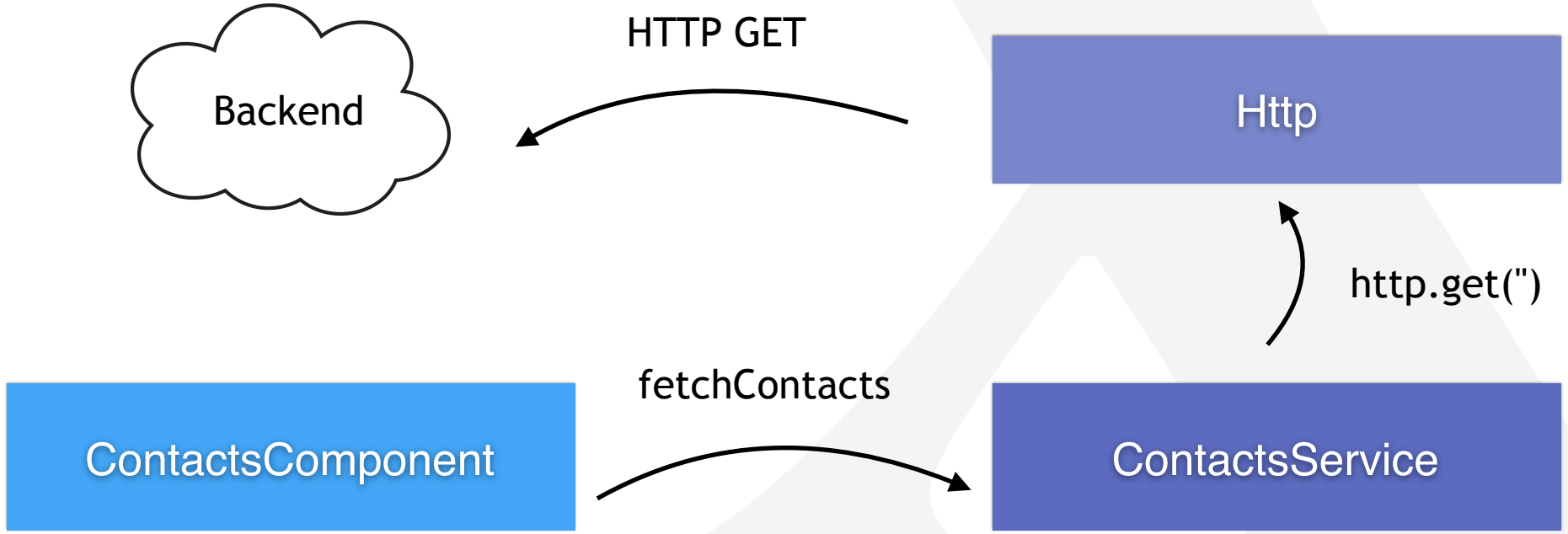
Http & Observables



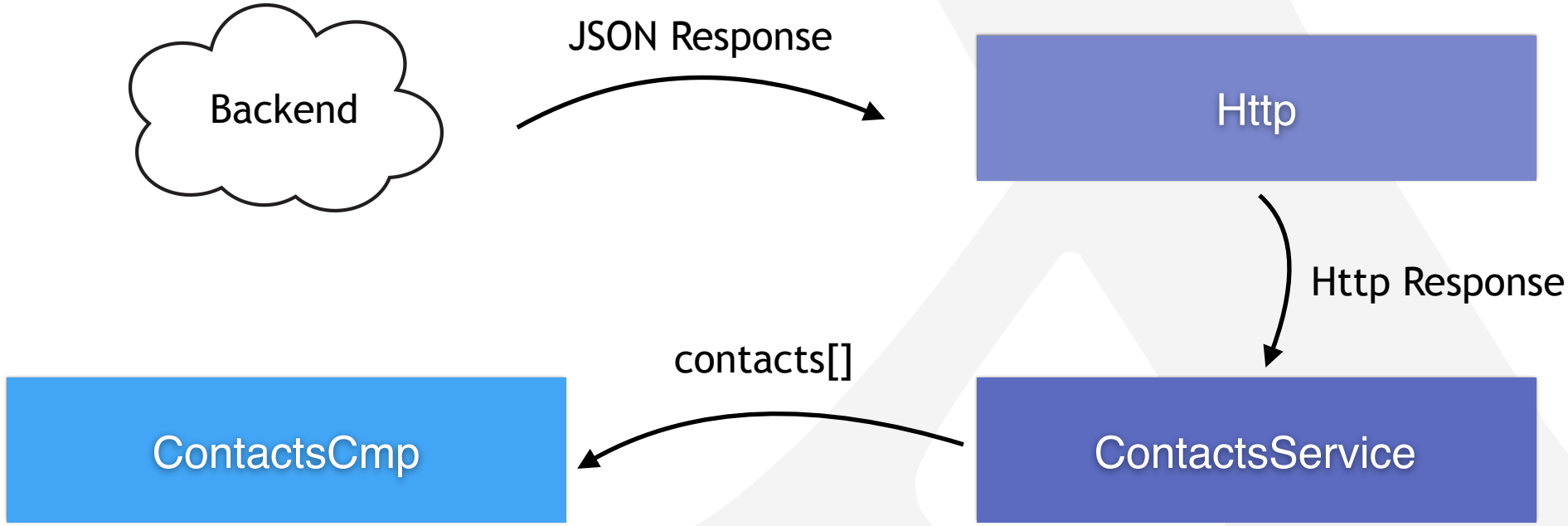
Http & Observables

- Angular provides Http functionality
  - HttpClientModule
  - ~~HttpModule~~
- HttpClient service
- RxJS Observables

Request for data



Response from backend



HttpClient service

Add Http functionality to application

```
@NgModule({
   declarations: [ … ],
   imports: [ HttpClientModule ],
   exports: []
})
export class AppModule {

}
```

Services in services: HttpClient

This is what we had...

```
@Injectable({providedIn: ‘root'})
export class ContactsService {

 constructor() { }

  fetchContacts(): Contact[] {
    return [ {…}, {…}, {…} ];
  }
}
```

Services in services: HttpClient

… this is the result

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Services in services: HttpClient

private keyword to connect service to "this"

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Services in services: HttpClient

Angular HttpClient uses Observables

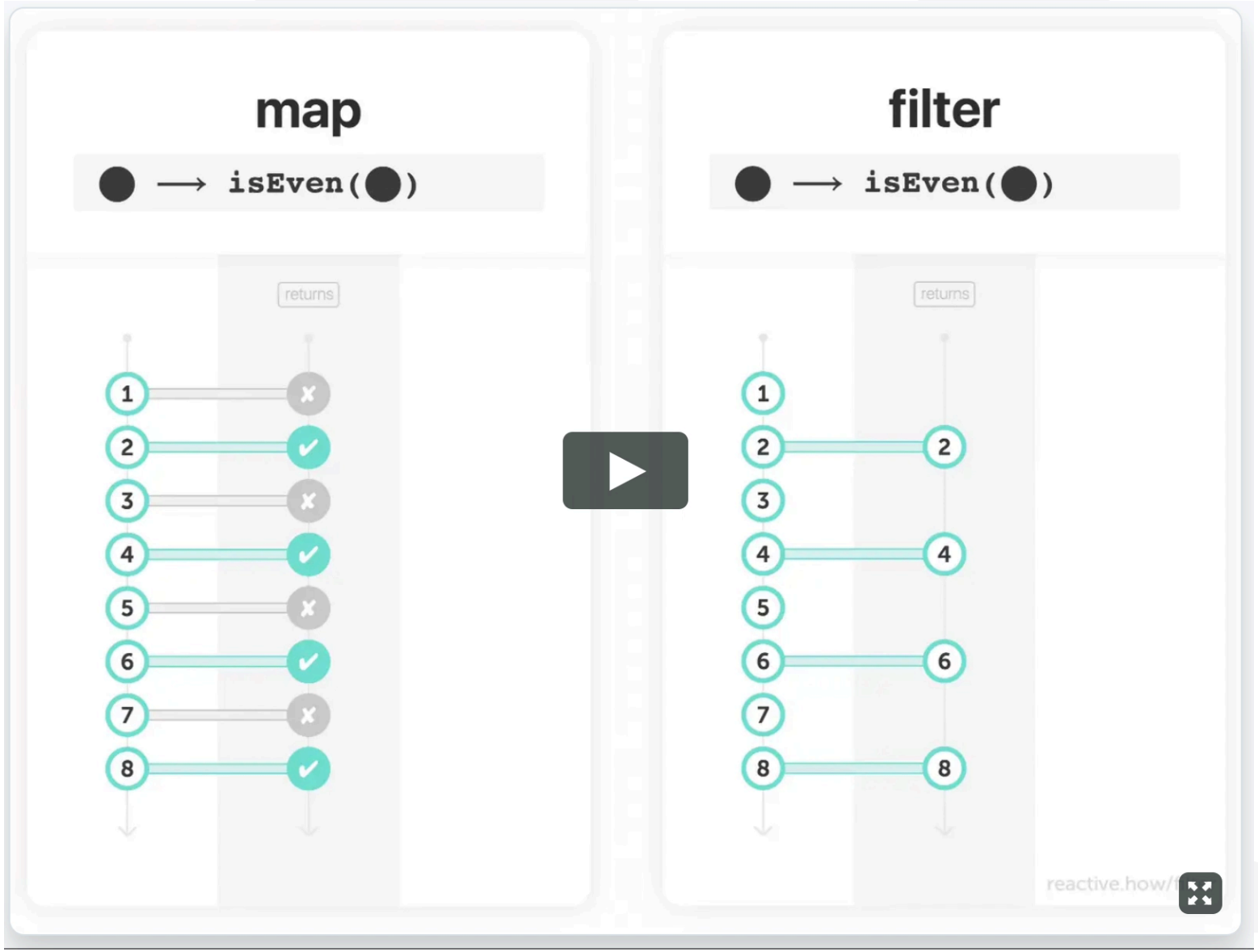
```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Observables

- "Sequence of events" or "Stream of data"
- React to events
- Extend with Operators
- subscribe() / unsubscribe()

Operators

- Transform data
- Filter stream



Service: Observables from HttpClient

Angular HttpClient uses Observables

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Service: Observables from HttpClient

Angular HttpClient uses Observables

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Service: Observables from HttpClient

Catch any errors

```
@Injectable({providedIn: 'root'})
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetch(): Observable<Contact[]> {
    return this.http.get<Contact[]>('/api/contacts')
      .pipe(
        catchError((error:any) => this.handleError(error))
      )
  }
}
```

Component: Observables from service

This is what we had...

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contacts = this.contactsService.fetch()
  }

}
```

Component: Observables from service

… this is the result

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contactsService.fetch()
        .subscribe(
          (contacts: Contact[]) => this.contacts = contacts,
          (error: MyAppError) => this.handleError(error)
        )
  }

}
```

Component: Observables from service

Subscribe starts the Observable sequence

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contactsService.fetch()
        .subscribe(
          (contacts: Contact[]) => this.contacts = contacts,
          (error: MyAppError) => this.handleError(error)
        )
  }

}
```

Component: Observables from service

Handle 'success' scenario

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contactsService.fetch()
        .subscribe(
          (contacts: Contact[]) => this.contacts = contacts,
          (error: MyAppError) => this.handleError(error)
        )
  }

}
```

Component: Observables from service

Handle 'error' scenario

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contactsService.fetch()
        .subscribe(
          (contacts: Contact[]) => this.contacts = contacts,
          (error: MyAppError) => this.handleError(error)
        )
  }

}
```

Component: AsyncPipe

Type contacts as an Observable

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Observable<Contact[]>;

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contacts = this.contactsService.fetch()
        .pipe(
          catchError(
            (error: MyAppError) => this.handleError(error)
          )
        )
  }
}
```

Component: AsyncPipe

Type contacts as an Observable

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Observable<Contact[]>;

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contacts = this.contactsService.fetch()
        .pipe(
          catchError(
            (error: MyAppError) => this.handleError(error)
          )
        )
  }
}
```

Component: AsyncPipe

Handle error scenario

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Observable<Contact[]>;

  constructor(private contactsService: ContactsService) {}

  ngOnInit() {
      this.contacts = this.contactsService.fetch()
        .pipe(
          catchError(
            (error: MyAppError) => this.handleError(error)
          )
        )
  }
}
```

Component: AsyncPipe

Async Pipe in *ngFor

```
<li
  *ngFor="let contact of
          (contacts | async)">
    {{ contact | json }}
</li>
```

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contacts: Observable<Contact[]>;

  constructor(private contactsService:
  ContactsService) {}

  ngOnInit() {
    this.contacts = this.contactsService.fetch()
        .pipe(
          catchError(
            (error: MyAppError) =>
            this.handleError(error)
          )
        )
  }
}
```

Component: AsyncPipe

Async Pipe in *ngIf

```
<main>
  <section *ngIf="contact | async;
                  let contact">
      {{ contact | json }}
  </section>
</main>
```

```
@Component({ selector: 'contacts' })
export class ContactsComponent implements OnInit {

  contact: Observable<Contact>;

  constructor(private contactsService:
  ContactsService) {}

  ngOnInit() {
    this.contact = this.contactsService.fetch()
        .pipe(
          catchError(
            (error: MyAppError) =>
            this.handleError(error)
          )
        )
  }
}
```

Recap Http

- Inject HttpClient in Service
- Return an Observable
- subscribe() / | async
- catchError

Code