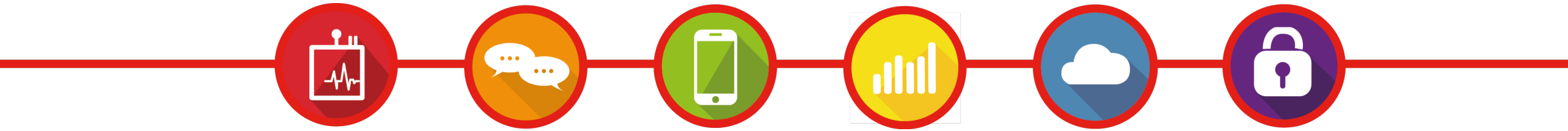
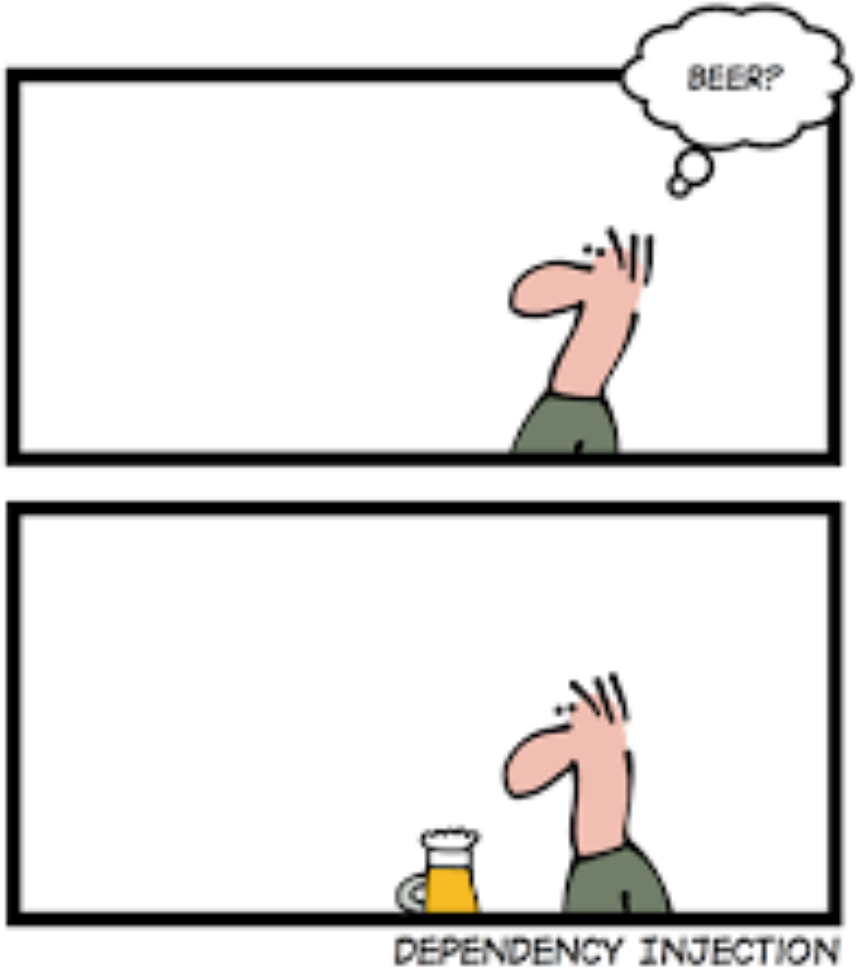
Dependency injection & Http



Dependency injection

- What
- Angular "Services"
- Http
  - observables



Dependency injection

- Manage dependencies
- No manual instantiation
- Injected by Angular

Services

- Usually ‘provided’ by NgModule
- Available for all modules
- Providing to components also possible
- Share common tasks
- "Cache"

Creating services

Services connect business components to outside data

```
@Injectable()
export class ContactsService {

 constructor() { }

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

Providing services

The providers array provides the service to all modules

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

}
```

Providing services

Every reference means an instance of the service

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

}
```

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

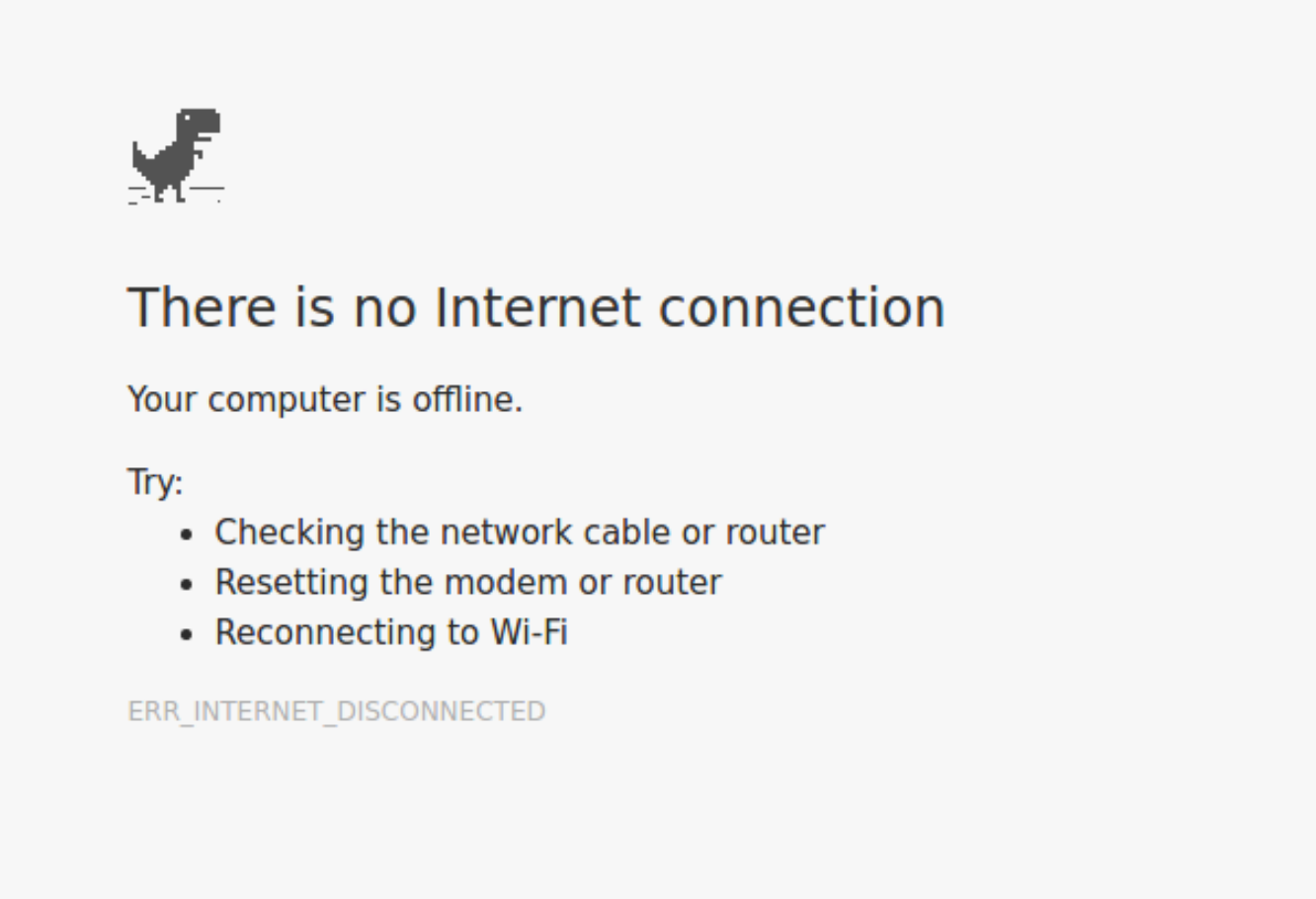
Call with "this"

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

  contacts: Contact[];

  constructor(private ContactsService: ContactsService) { }

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

}
```

Recap: Dependency injection

- providers: [ ]
- Services injectable in all components
- Constructor injections
- **private** keyword

Http & Observables

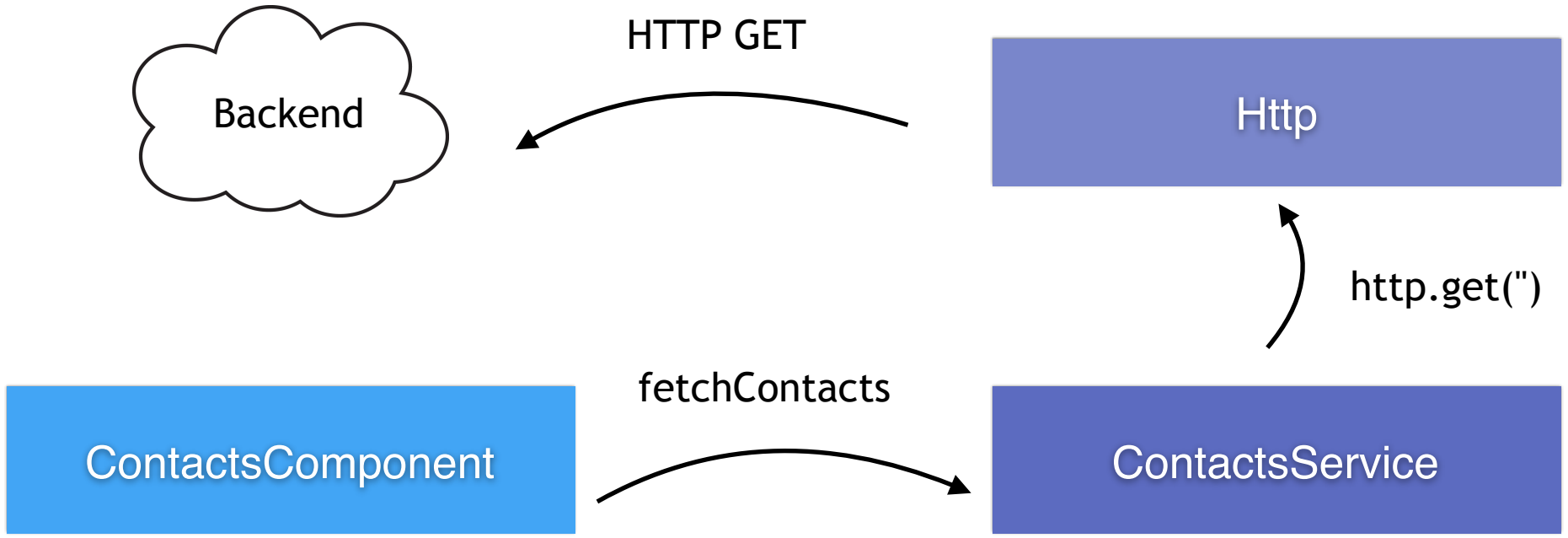


Http & Observables

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

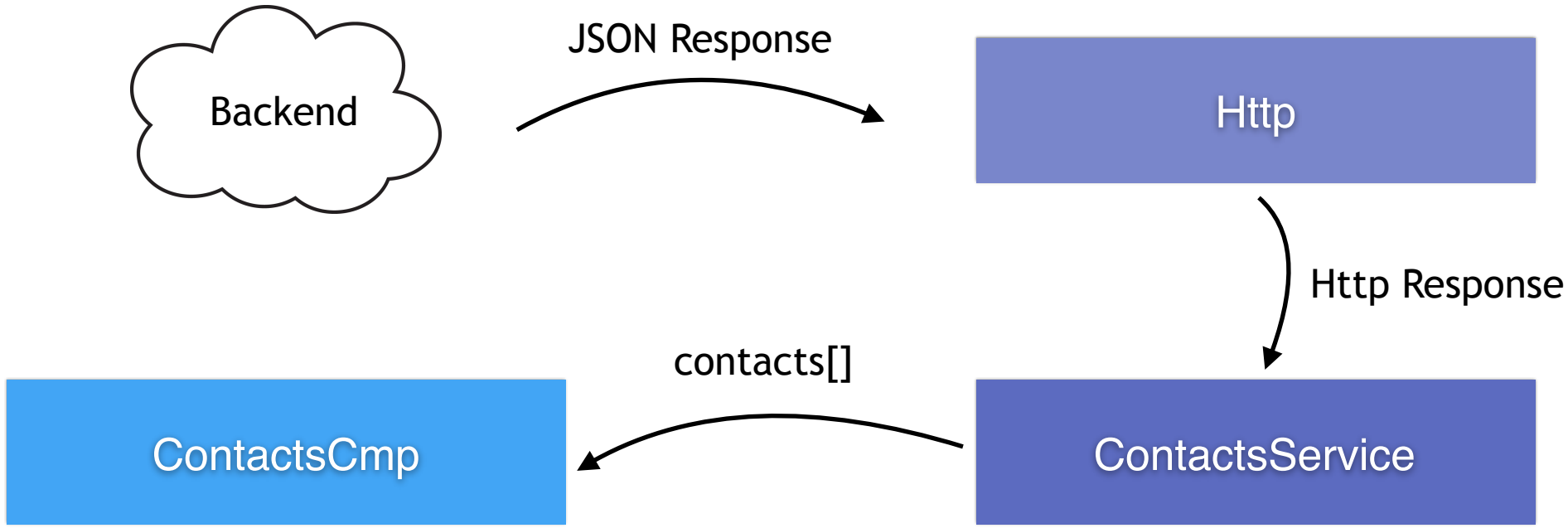
Data via Http

Request



Data via Http

Response



HttpClient service

Add Http functionality to application

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

}
```

HttpClient service

Add Http functionality to application

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

}
```

Services in services: HttpClient

This is what we had...

```
export class ContactsService {

 constructor() { }

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

Services in services: HttpClient

... this is the result

```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

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

Services in services: HttpClient

@Injectable() adds inject ability to any class

```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetchContacts(): 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"

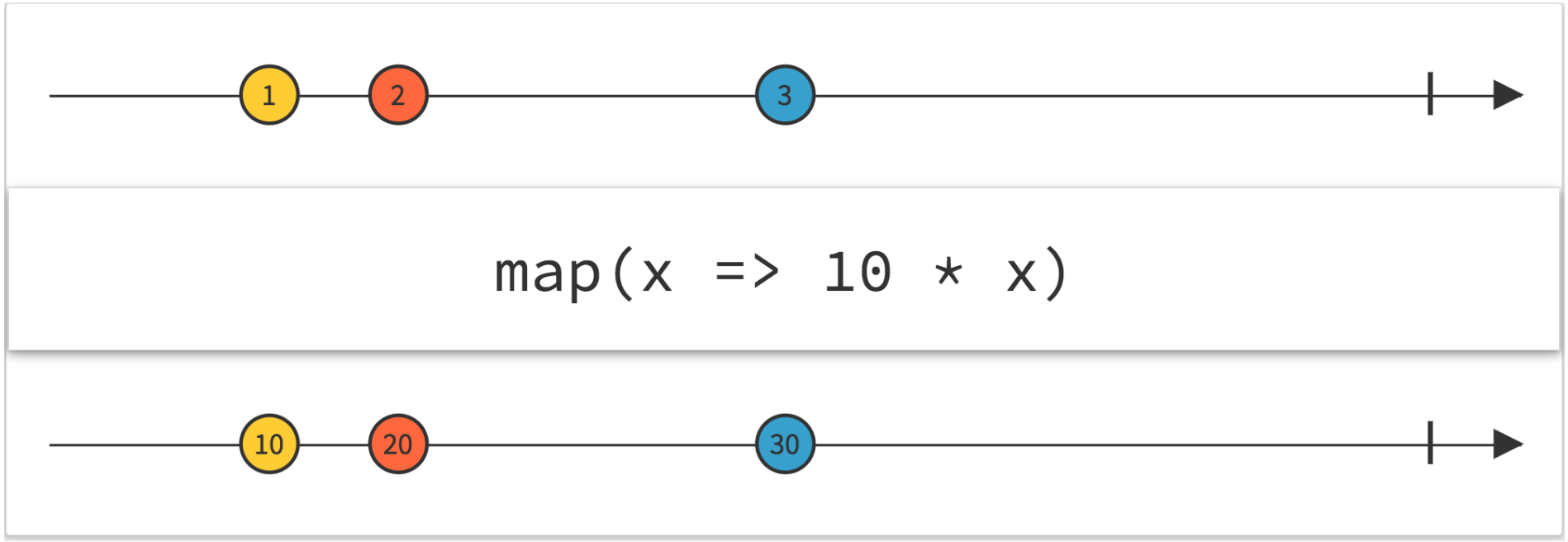
```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

  fetchContacts(): 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()
export class ContactsService {

 constructor(private http: HttpClient) { }

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

Observables



Observables

- Promises on steroids
- React to events
- Operators
- subscribe() / unsubscribe()
- AsyncPipe

Http & Observables

Angular Http uses Observables

```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

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

Http & Observables

Do a GET call

```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

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

Http & Observables

Catch any errors

```
@Injectable()
export class ContactsService {

 constructor(private http: HttpClient) { }

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

Http & Observables

This is what we had...

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Http & Observables

... this is the result

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Observables

fetchContacts returns an Observable

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Http & Observables

Subscribe starts the Observable sequence

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Http & Observables

Handle 'success' scenario

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Http & Observables

Handle error scenario

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

  contacts: Contact[];

  constructor(private contactsService: ContactsService) {}

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

}
```

Http & Observables

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.fetchContacts()
        .pipe(
          catchError(
            (error: MyAppError) => this.handleError(error)
          )
        )
  }
}
```

Http & Observables

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.fetchContacts()
        .pipe(
          catchError(
            (error: MyAppError) => this.handleError(error)
          )
        )
  }
}
```

Http & Observables

Async Pipe in *ngFor

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

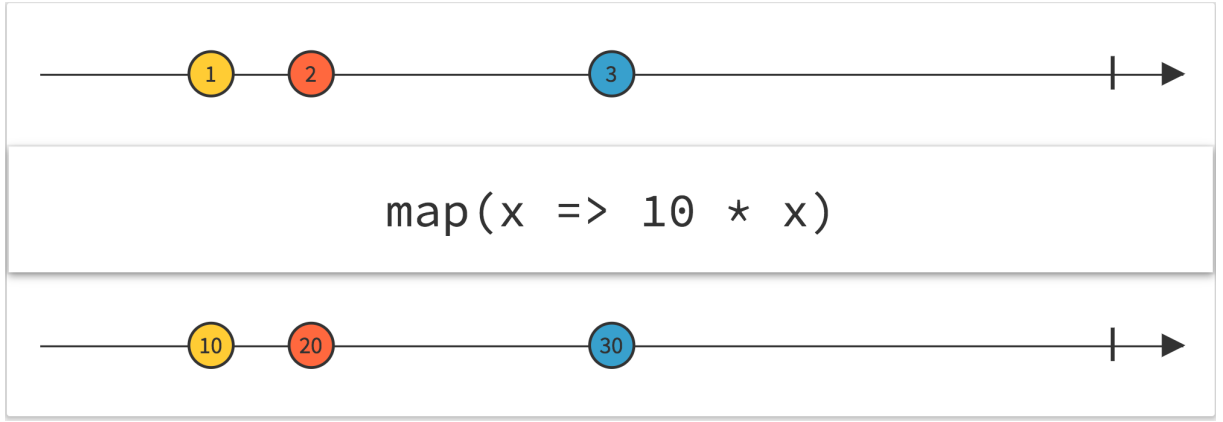
Http & Observables

Async Pipe in *ngIf

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

Recap: Http & Observable

- HttpClient
- Observables
- subscribe()
- AsyncPipe



Dependency injection: Demo