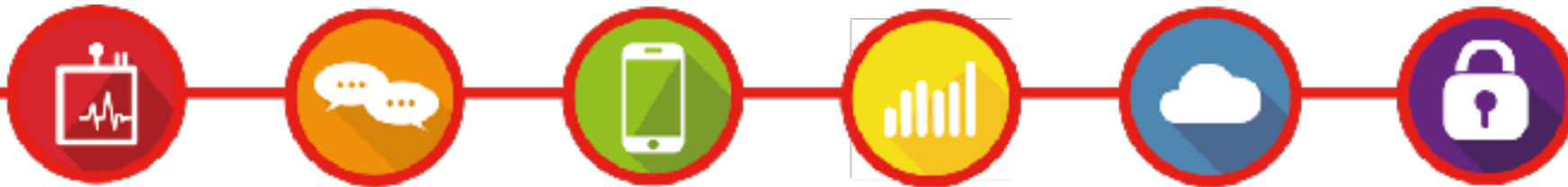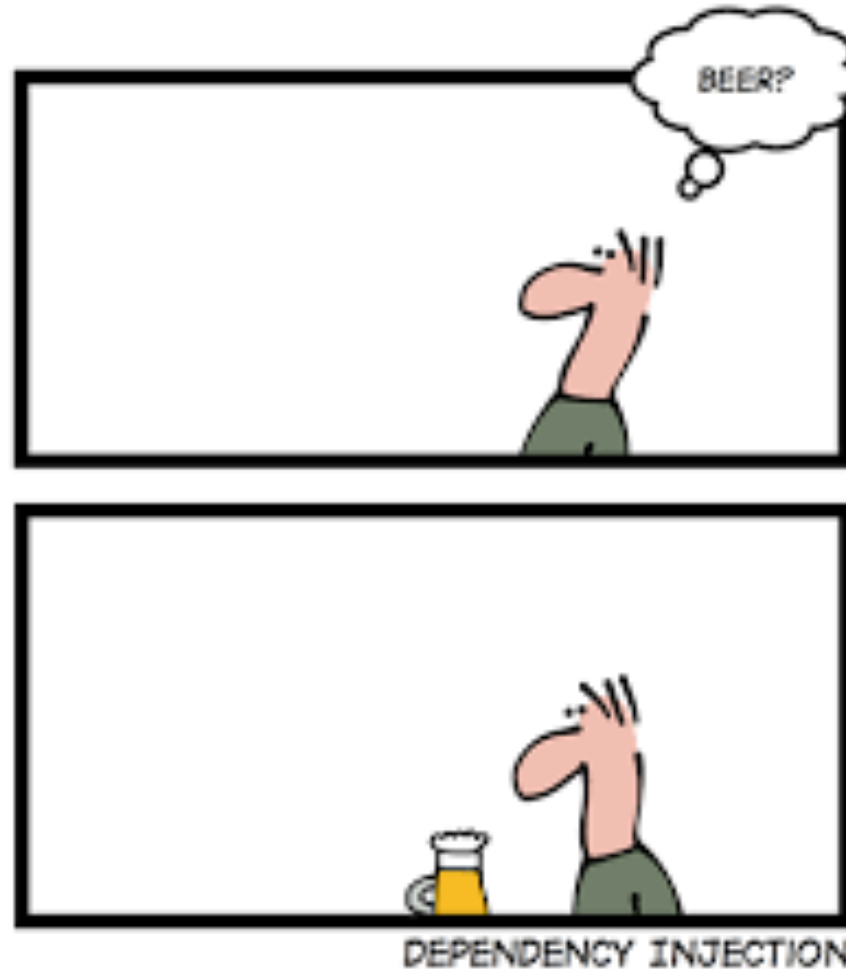# Dependency injection & Http

Dependency injection

Dependency injection

- Manage dependencies

- No manual instantiation

- Injected by Angular

Services

- Available for <u>all</u> modules

- Providing to components also possible

- Share common tasks

- "Cache"

Creating services

Services connect business components to outside data

```
export class ContactsService {

  constructor() { }

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

Creating services

Services connect business components to outside data

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

  constructor() { }

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

ilionx
Your partner in digital business

Using services

constructor injection

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

  contacts: Contact[];


  constructor() {

  }


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

}
```

Using services

constructor injection

```typescript
@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()
  }

}
```

onx
Your partner in digital business

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()
  }

}
```

ilionx
Your partner in digital business

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

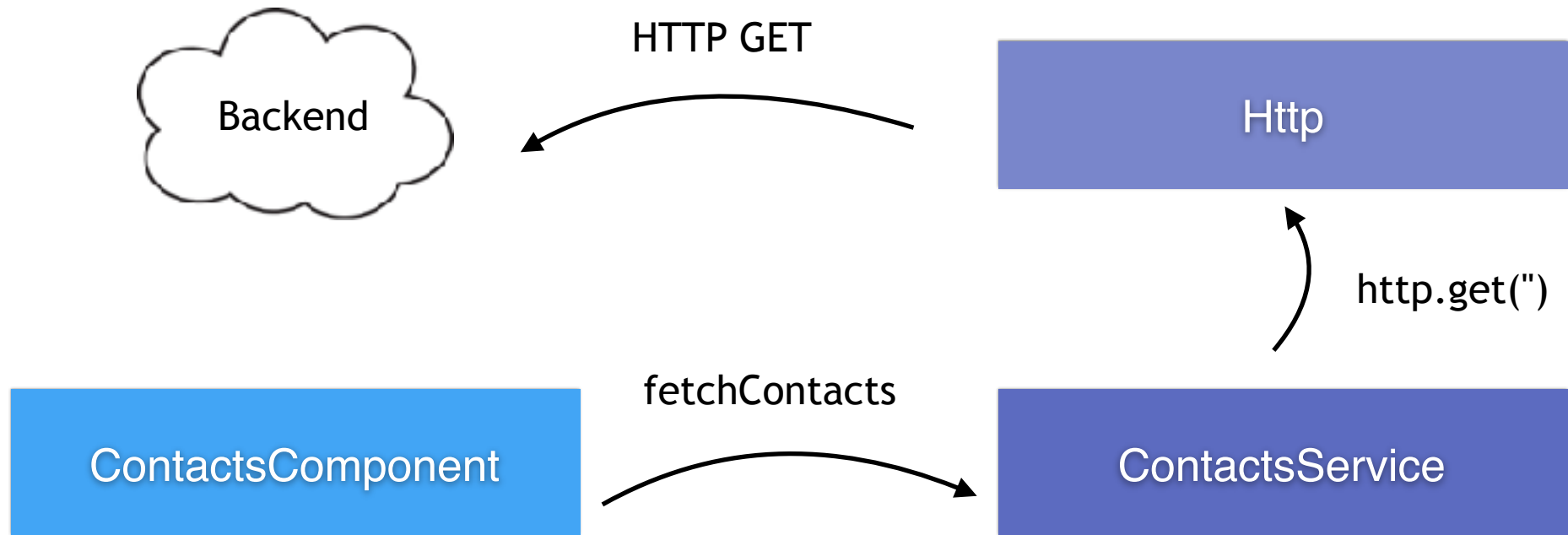- Constructor injections

- **private** keyword

Http & Observables

Http & Observables

- Angular provides Http functionality

    - HttpClientModule

    - ~~HttpModule~~

- HttpClient service

- RxJS Observables
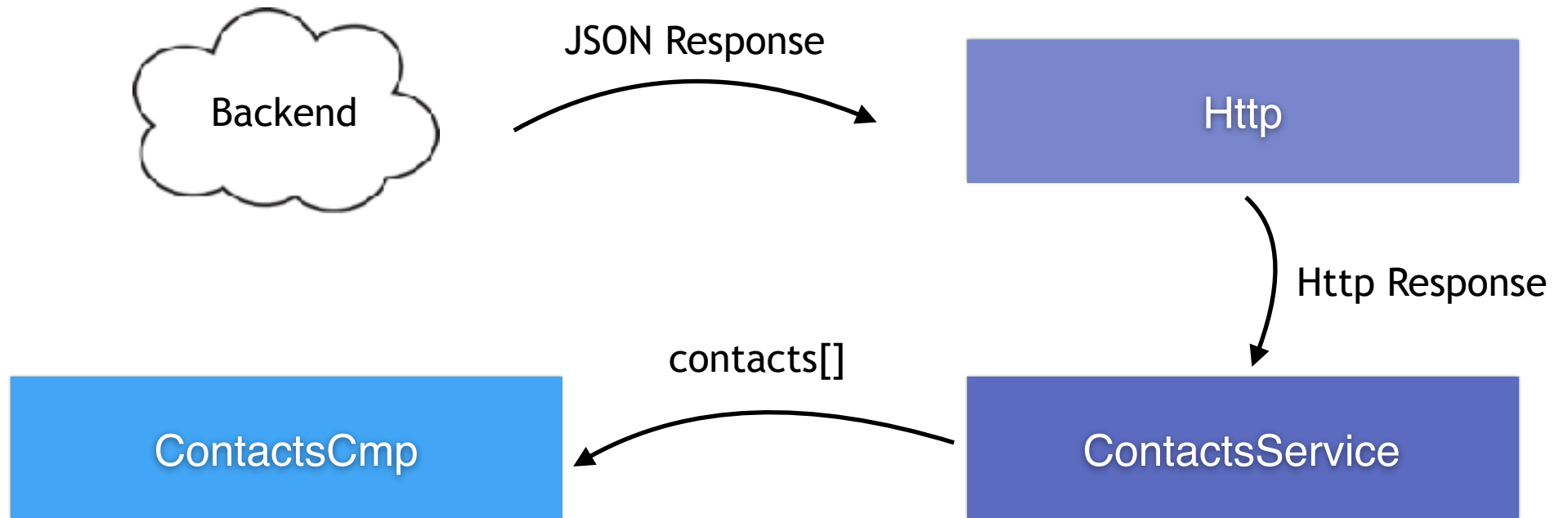
Data via Http

Request

Data via Http

Response

HttpClient service

Add Http functionality to application

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

}
```

HttpClient service

Add Http functionality to application

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

}
```

ilionx
Your partner in digital business

Services in services: HttpClient

This is what we had...

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

  constructor() { }

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

ilionx
Your partner in digital business

Services in services: HttpClient

... this is the result

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

  constructor(private http: HttpClient) { }

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

ilionx
Your partner in digital business

Services in services: HttpClient

private keyword to connect service to "this"

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

  constructor(private http: HttpClient) { }

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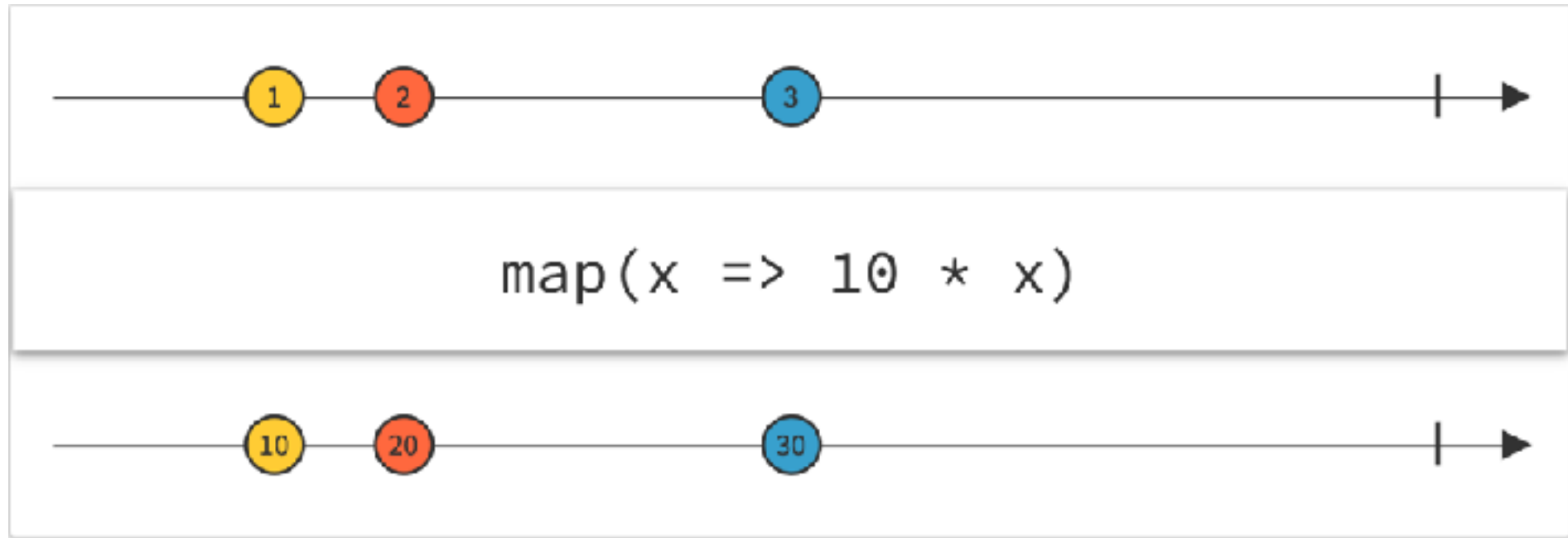
Services in services: HttpClient

Angular HttpClient uses Observables

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

  constructor(private http: HttpClient) { }

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

ilionx

Your partner in digital business

Observables


```
map(x => 10 * x)
```

Observables

- Promises on steroids

- React to events

- Operators

- subscribe() / unsubscribe()

- AsyncPipe

Http & Observables

Angular Http uses Observables

```typescript
@Injectable({providedIn: 'root'})
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

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

  constructor(private http: HttpClient) { }

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

ilionx
Your partner in digital business

Http & Observables

Catch any errors

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

  constructor(private http: HttpClient) { }

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

}
```

ilionx
Your partner in digital business

Http & Observables

This is what we had...

```typescript
@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

```typescript
@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

```typescript
@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

```typescript
@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

```typescript
@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)
        )
      )
  }
}
```

onx
Your partner in digital business

Http & Observables

Type contacts as an Observable

```typescript
@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

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

Http & Observables

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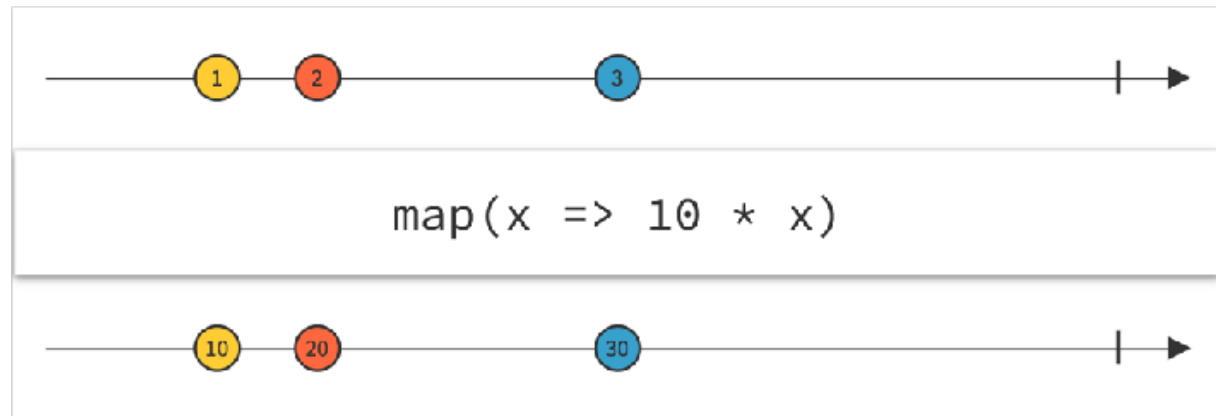
```html
<main>
  <section *ngIf="contact | async; let contact">
      {{ contact | json }}
  </section>
</main>
```

Recap: Http & Observable

- HttpClient

- Observables

- subscribe()

- AsyncPipe

Dependency injection: Demo