



Optimisez les performances

Animé par Mazen Gharbi



Optimisations

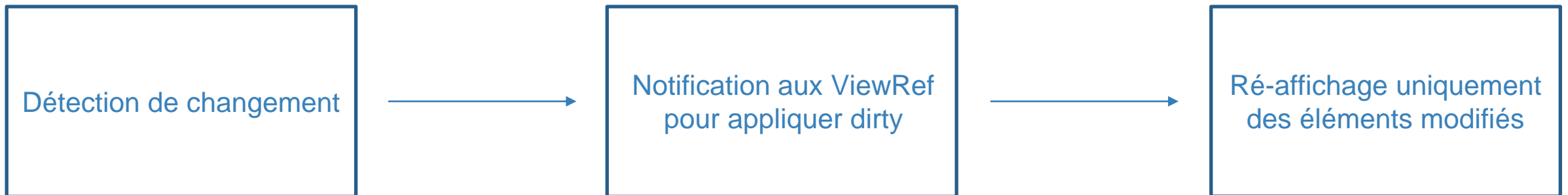
Introduction



- ▷ Angular est un framework très puissant
 - › Avec beaucoup d'inversion de contrôle
- ▷ Par défaut, non optimisé, c'est à nous de le rendre performant !
- ▷ Pour y arriver, l'équipe Google nous fournit tous les outils nécessaires

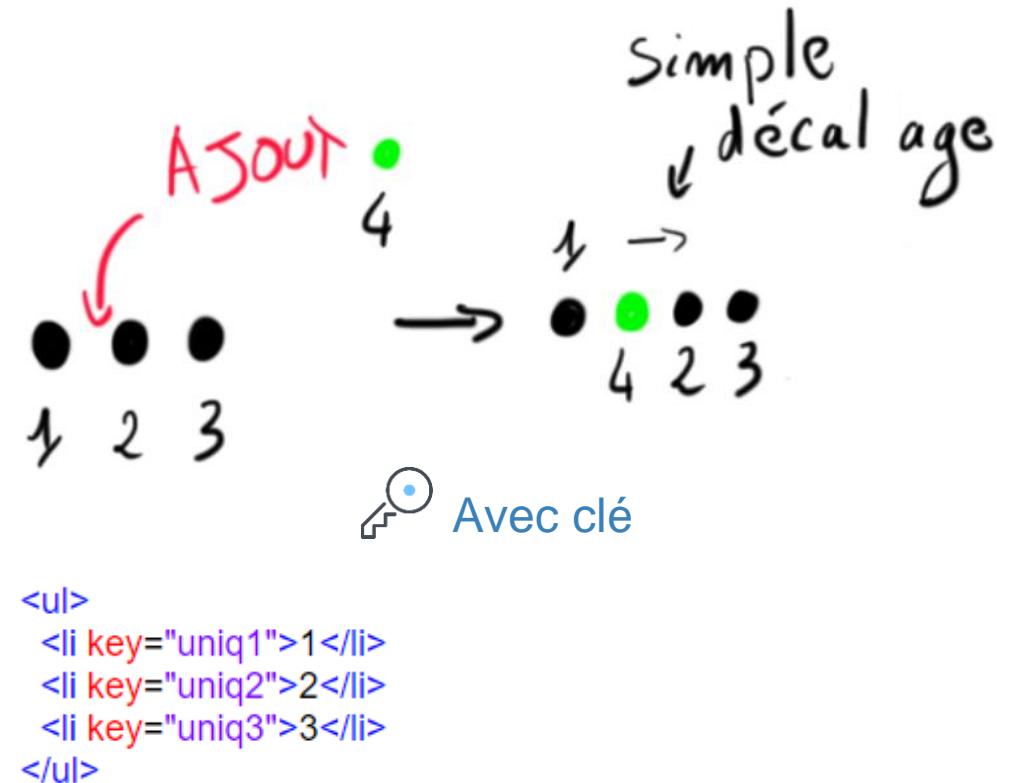
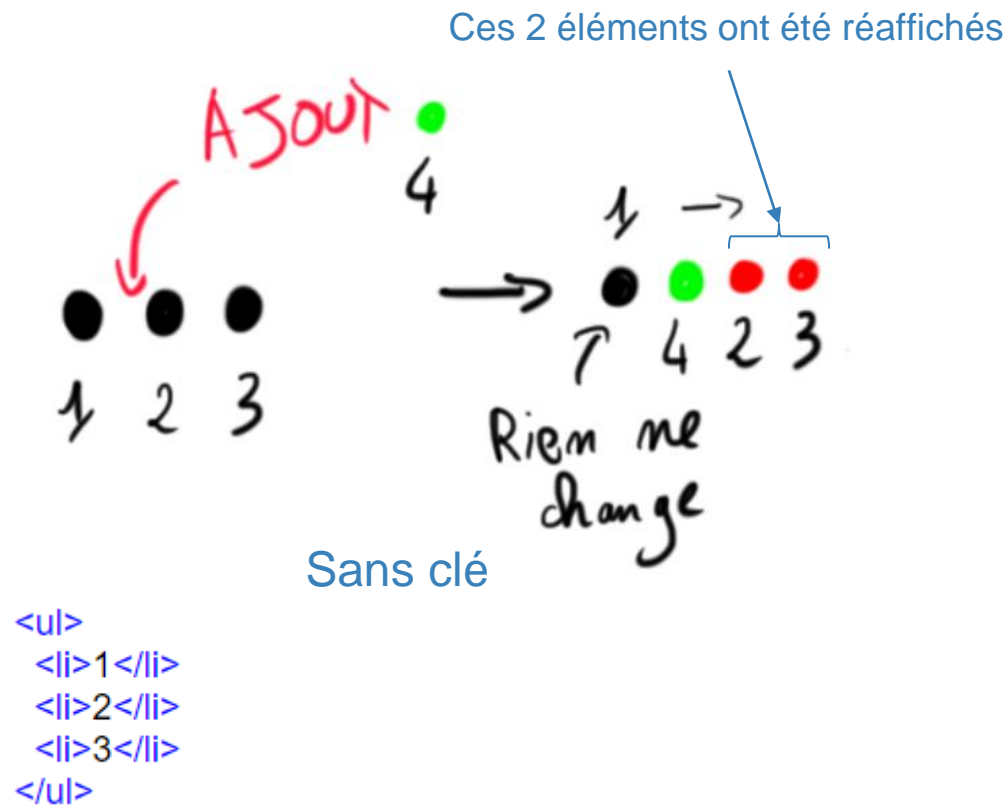
Optimisez le *ngFor

- ▷ ***ngFor** est une directive emblématique d'Angular
 - › Mais elle peut être optimisée en terme de performance !
- ▷ Souvenez-vous, dès lors qu'un changement est détecté, la machine se met en marche :



Problème au ré-affichage

▷ Mais pour les listes, il y a un problème...



Utiliser trackBy avec *ngFor

▷ Soit une liste d'utilisateur :

```
ngOnInit() {  
  this.users = [  
    { name: 'Toto', id: 0 },  
    { name: 'Titi', id: 1 },  
  ]  
}
```

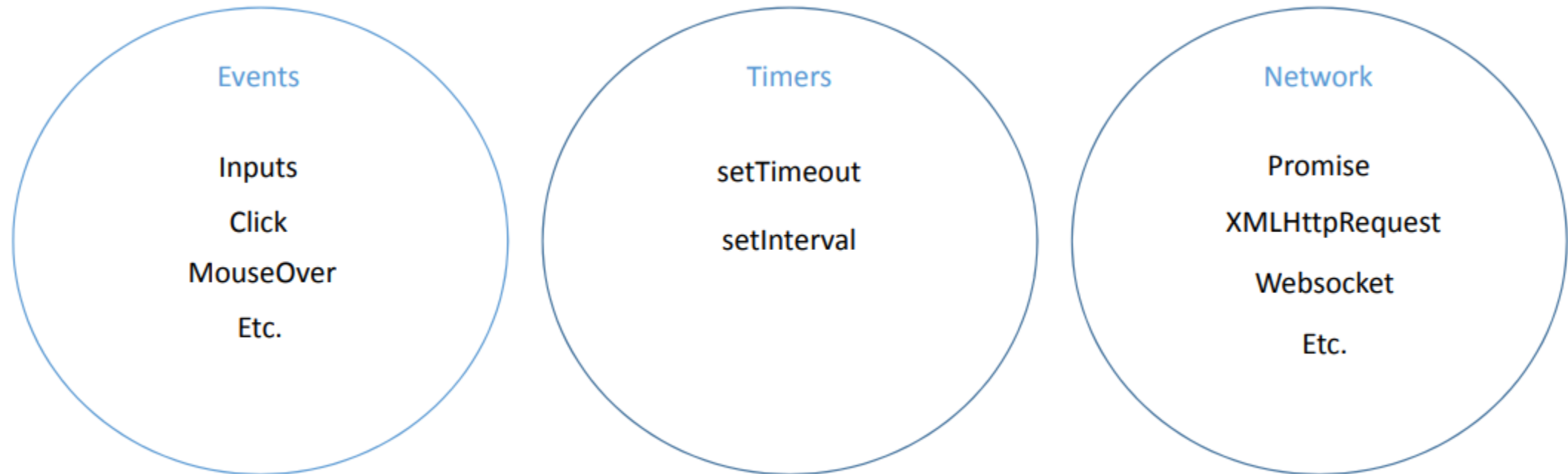
```
<p *ngFor="let user of users">  
  Salut {{user.name}}  
</p>
```

▷ Pour optimiser, il suffit d'ajouter la propriété trackBy :

```
public trackByUserId(index: number, user: User) {  
  return user.id;  
}
```

```
<p *ngFor="let user of users; trackBy: trackByUserId">  
  Salut {{user.name}}  
</p>
```

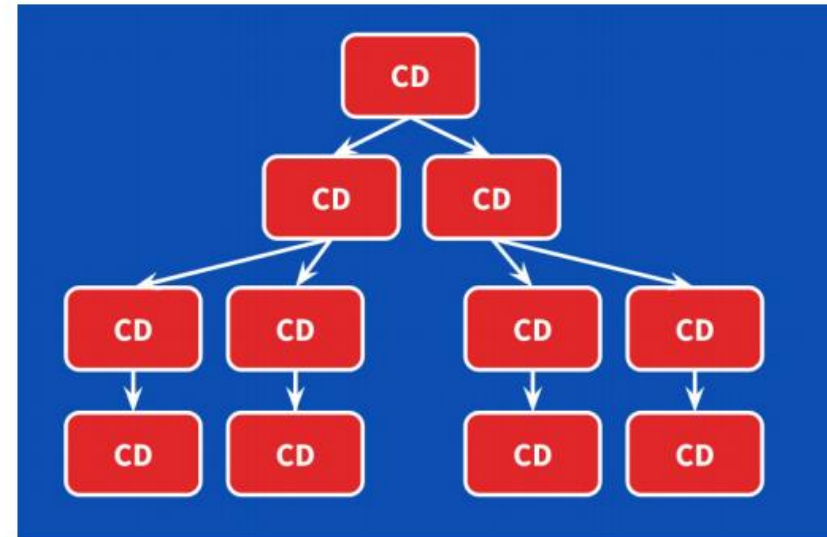
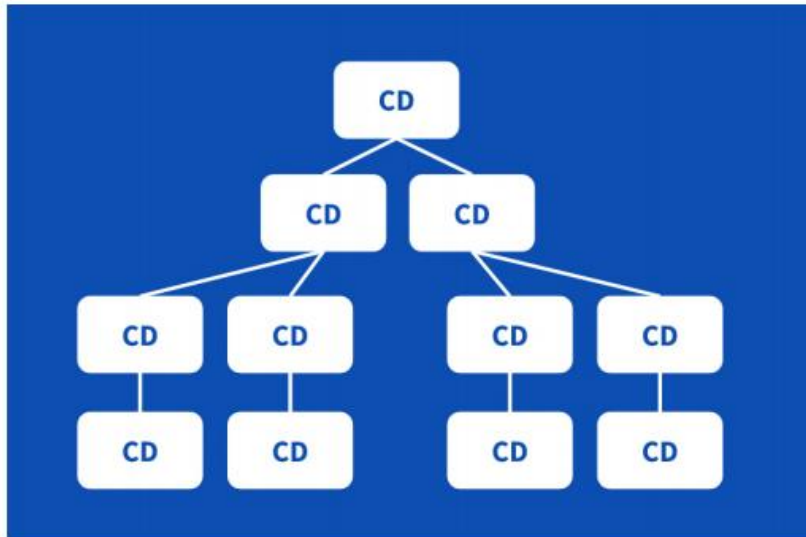
Tous vos composants doivent être OnPush



Tous vos composants doivent être OnPush

► Pour activer le mode OnPush, on le fait dans le composant :

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css'],  
  changeDetection: ChangeDetectionStrategy.OnPush  
})
```



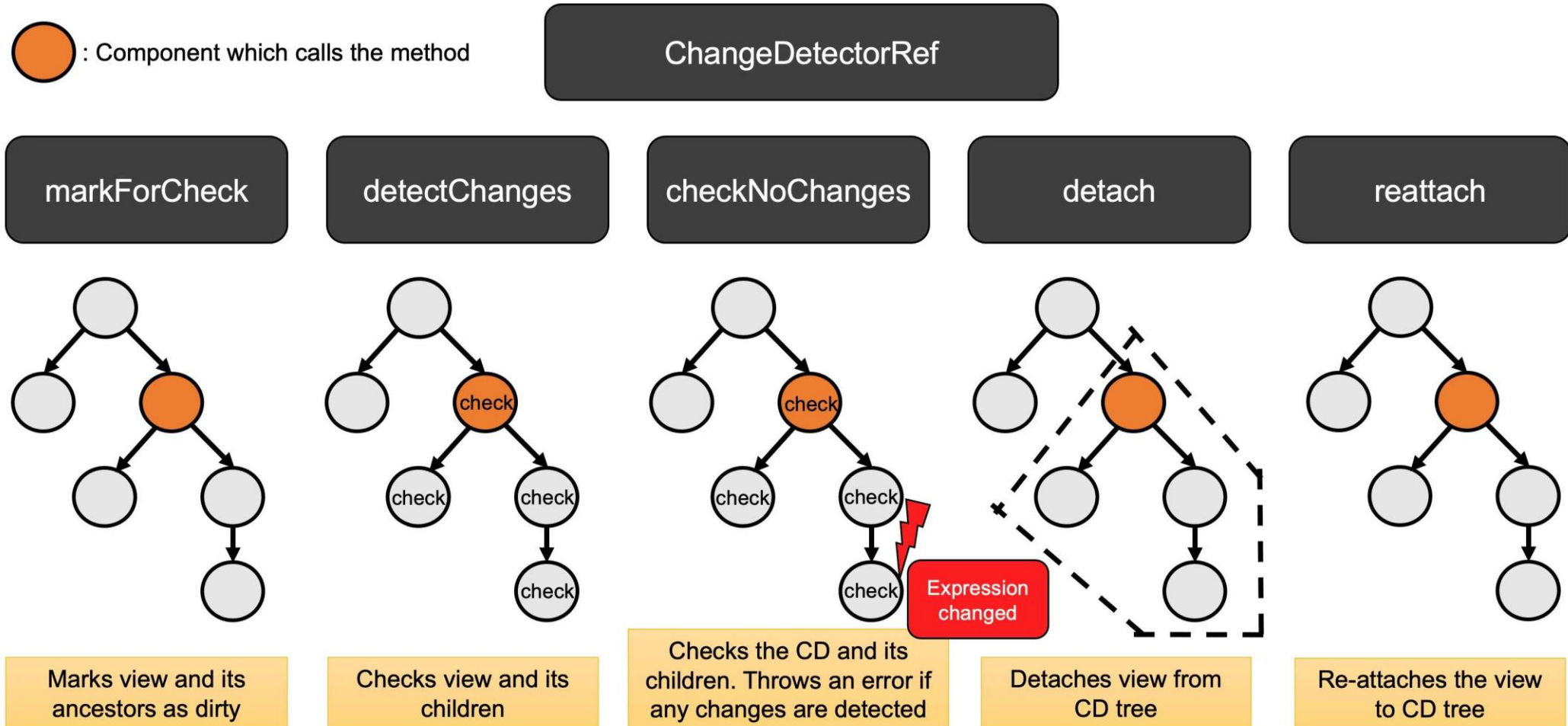
Exécuter le code complexe en dehors des zones

Service NgZone à injecter



```
this.ngZone.runOutsideAngular(() => {  
  this.interval = window.setInterval(() => {  
    this.maFunction();  
  }, 10)  
});
```

Détacher vos composants complexes



Rendez vos objets immutables

- ▷ Les objets immuables sont ceux qui renvoient une nouvelle référence à la variable lorsque leurs valeurs sont mises à jour.

Immutable

```
ngOnInit() {  
  const myUsers = [  
    {  
      id: 1,  
      name: 'Joe';  
    }  
  ];  
  
  myUsers[0] = { ...myUsers[0], name: 'Dupond' };  
}
```

Mutable

```
ngOnInit() {  
  const myUsers = [  
    {  
      id: 1,  
      name: 'Joe';  
    }  
  ];  
  
  myUsers[0].name = 'Dupond';  
}
```

Immutable et OnPush

- ▷ En mode OnPush, la vue du composant enfant se met à jour **si la référence de l'élément change**

Compteur: 0

Incrementer

Mutable

Compteur: 0

Incrementer

Immutable

Utilisez les pipes !

- ▷ Les pipes permettent de mettre en cache la valeur
- ▷ Tant que la valeur d'entrée ne change pas, Angular récupère la valeur en cache



Webpack

Chef d'orchestre

- ▷ Permet de compiler des modules JavaScript
 - › On parle de « Module bundler »
- ▷ En Angular, il permet :
 - › De regrouper nos ressources ;
 - › Surveille les changements et réexécute les tâches ;
 - › Exécuter la transpilation Typescript/Babel vers ES5
 - › Permet d'utiliser `require()` pour les fichiers CSS ;
 - › Exécute un serveur Web de développement ;
 - › Gère le remplacement de nos module à chaud (HMR) ;
 - › Divise les fichiers de sortie en plusieurs fichiers ;
 - › Réalise le tree-shaking ;
 - › etc..

Entrées & Bundles

- ▷ Webpack prend des **points d'entrée** ;
- ▷ Réalise des traitements
 - › Avec les loader ;
 - › Et les plugins ;
- ▷ Puis **génère des bundles**
- ▷ Le fichier de configuration webpack est défini par un fichier .js
 - › On peut construire différents fichiers de configuration (dev / prod / test)

Configuration Webpack

- ▷ Créons un fichier « webpack.config.dev.js » à la racine de notre projet
 - › Représentera la configuration de l'environnement de dev

```
"use strict";

const { CustomPlugin } = require("custom");

module.exports = {
  mode: "development",
  entry: ["./src/main.js"], // Porte(s) d'entrée(s)
  module: {
    rules: [
      // Définition de nos loaders
      {
        // Appliquer le loaders custom-loader aux fichiers avec l'extension .js
        test: /\.js$/,
        use: "custom-loader"
      }
    ]
  },
  // Chargement de nos plugins
  plugins: [new CustomPlugin()]
};
```

webpack.config.dev.js

Script build

- ▷ Avec npm, vous avez la possibilité de définir des scripts
 - › Dans la propriété scripts du fichier « package.json »

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "build:dev": "webpack --config webpack.config.dev.js"  
},
```

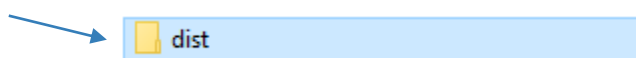
package.json

- ▷ Pour lancer le script :

```
> npm run build:dev
```

- ▷ Vous constaterez qu'un dossier « dist » a été créé :

Contient le résultat du build



Brancher les fils

- ▷ Enfin, il va être nécessaire de constituer le contenu du fichier « index.html » !

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Webpack configuration ☺</title>
</head>
<body>
  <div id="app"></div>
  <script src="dist/main.js"></script>
</body>
</html>
```

index.html

- ▷ Et voilà !

Webpack et Angular

- ▷ Evidemment, Angular utilise Webpack
- ▷ Avec la Angular CLI, c'est l'équipe Angular qui se charge de maintenir la configuration WebPack pour nous
 - › Cela peut parfois être très compliqué comme en témoigne les [issues](#) GIT
- ▷ On peut aisément remplacer la configuration WebPack actuel
 - › Commençons par installer une nouvelle librairie :

```
> npm i -D @angular-builders/custom-webpack
```

- ▷ *Attention, il vous faudra maintenir votre configuration vous-même !*

Notre configuration

```
    "serve": {  
      "builder": "@angular-builders/custom-webpack:dev-server",  
      "options": {  
        "customWebpackConfig": {  
          "path": "./my-custom-webpack.config.js"  
        },  
        "browserTarget": "test:build"  
      }  
    },
```

On surcharge le builder webpack initial

Appel de notre fichier de configuration

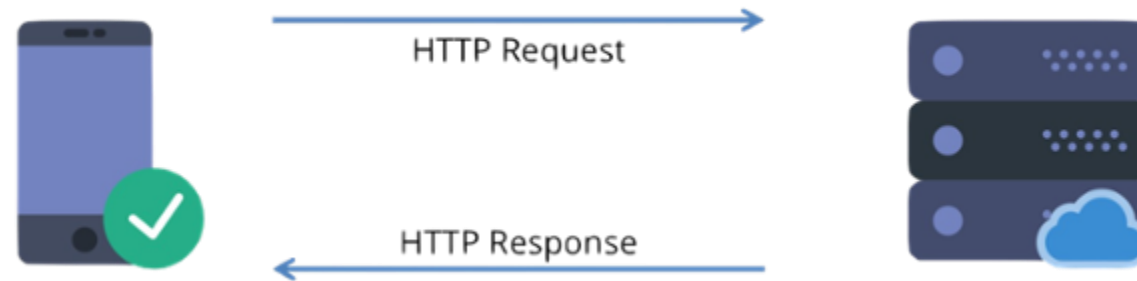
```
const webpack = require("webpack");
```

my-custom-config.config.js

```
module.exports = {  
  mode: 'development',  
  module: {  
    rules: [  
      // Nos règles de loader ici  
    ],  
  },  
  plugins: [  
    // Et les plugins là  
  ],  
};
```

Il était une fois ... Http

- Les requêtes HTTP sont basées sur un protocole de communication bien définie
- Une requête est envoyée du client vers le serveur, et le serveur répond :



- La réponse est constituée d'un code informant sur le statut de la requête (réussie / échouée / etc.)

Codes de retour

Code	Signification
200	Opération réussie
201	Retourné après une requête POST, le document a bien été traité
202	La requête a été enregistrée, elle sera exécutée plus tard
204	Le serveur a traité la demande, mais aucun résultat à renvoyer
301	La ressource demandée a été déplacée, c'est une redirection
304	Réponse à une requête GET, le serveur ne renvoie pas la ressource car aucune mise à jour sur celle-ci
305	La ressource doit obligatoirement être accédée à travers un proxy
400	Erreur de syntaxe, le serveur n'a pas compris la requête
401	Le client doit s'authentifier pour accéder à cette ressource
403	Accès interdit à la ressource ! Même si connecté
404	La ressource est introuvable
405	Méthode non autorisée pour l'accès à cette ressource
406	Document inexistant dans le format voulu par le client
500	Une erreur interne est survenue

Comprendre une requête HTTP

- ▷ HTTP est un protocole
 - › Comme tout protocole, les échanges sont codifiés
 - › Dans cet exemple, on soumet un formulaire au serveur

The diagram shows an HTTP request with the following components and annotations:

- Méthode**: Points to the `POST` method on line 1.
- Ressource voulue**: Points to the `/validation.php` path on line 1.
- Protocole utilisé**: Points to the `HTTP/1.1` version on line 1.
- DNS / adresse ip requêté**: Points to the `localhost:8000` host on line 2.
- Corps de la requête, Contient toutes les données envoyées au serveur**: A bracket on the right side groups lines 6 through 14, indicating the body of the request.
- Boundary = Séparateur**: Points to the `-----WebKitFormBoundary7MA4YWxkTrZu0gW--` separator on line 14.

```
1 POST /validation.php HTTP/1.1
2 Host: localhost:8000
3 Cache-Control: no-cache
4 Content-Type: multipart/form-data; boundary=-----WebKitFormBoundary7MA4YWxkTrZu0gW
5
6 -----WebKitFormBoundary7MA4YWxkTrZu0gW
7 Content-Disposition: form-data; name="prenom"
8
9 Macademia
10 -----WebKitFormBoundary7MA4YWxkTrZu0gW
11 Content-Disposition: form-data; name="email"
12
13 hey@macademia.fr
14 -----WebKitFormBoundary7MA4YWxkTrZu0gW--
```


Principe générale

▷ Une requête possède une **entête** et un corps

Requête du client

Entête {
GET /dossier/page-voulue.html HTTP/1.0
Accept: www/source
Accept: image/gif
User-Agent: Mozilla/5.0
From: toto@macademia.fr
* une ligne blanche *
// Corps de la requête, ici il n'y en a pas car requête GET

Réponse du serveur

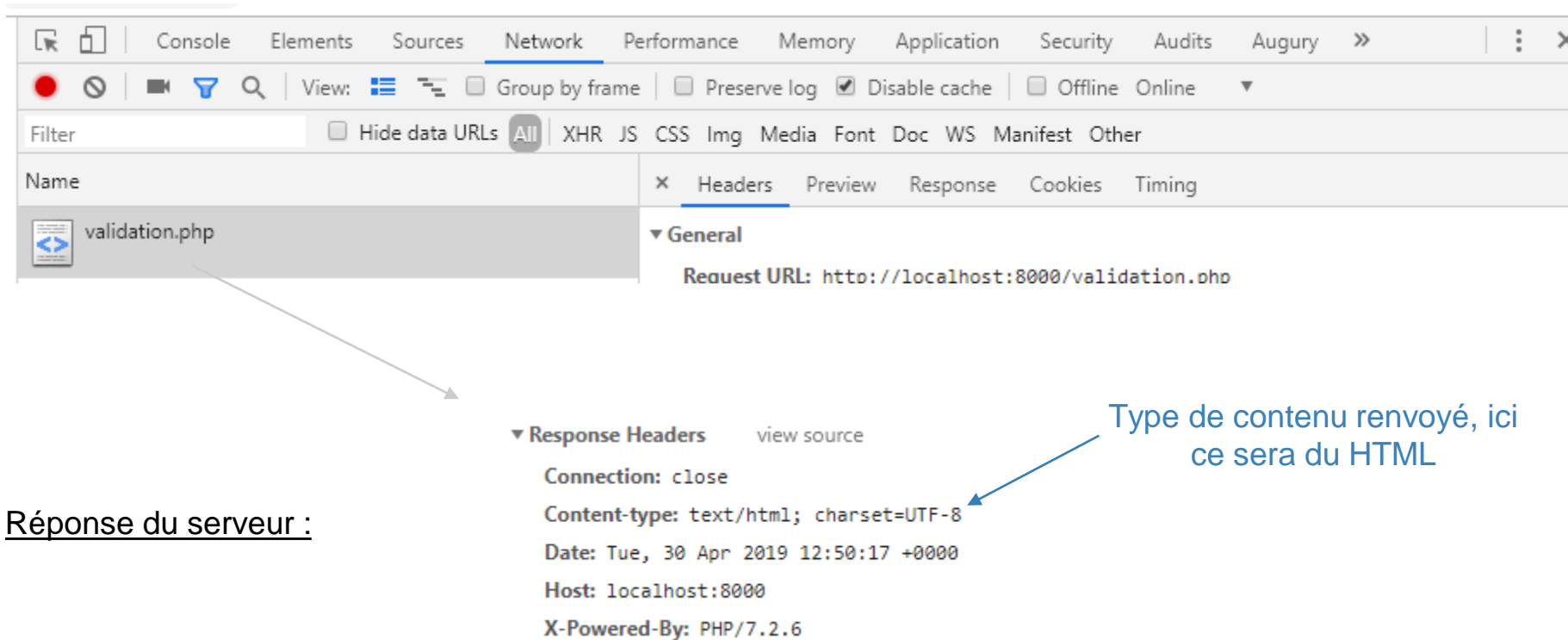
HTTP/1.0 200 OK
Date: Sat, 01 Jun 2019 16:30:24 GMT
Server: Apache/2.2.15 (CentOS)
MIME-version: 1.0
Last-modified: Mon, 15 Nov 96 23:33:16 GMT
Content-type: text/html // type du document retourné
Content-length: 2345 // sa taille
* une ligne blanche *
<HTML><HEAD><TITLE>...

Réponse du serveur

- ▷ Une fois la requête effectuée, le serveur réalise un traitement et met en attente le client
- ▷ Dès le traitement terminé, le serveur renvoie une réponse
 - › Cette réponse est évidemment formaté selon le protocole HTTP
- ▷ Les informations renvoyées par le serveur servent principalement au navigateur afin qu'il puisse formater les données correctement

Exemple de réponse

▷ Avec chrome, il est possible de visualiser les requêtes échangées :



Réponse du serveur :

HTTP 1.1

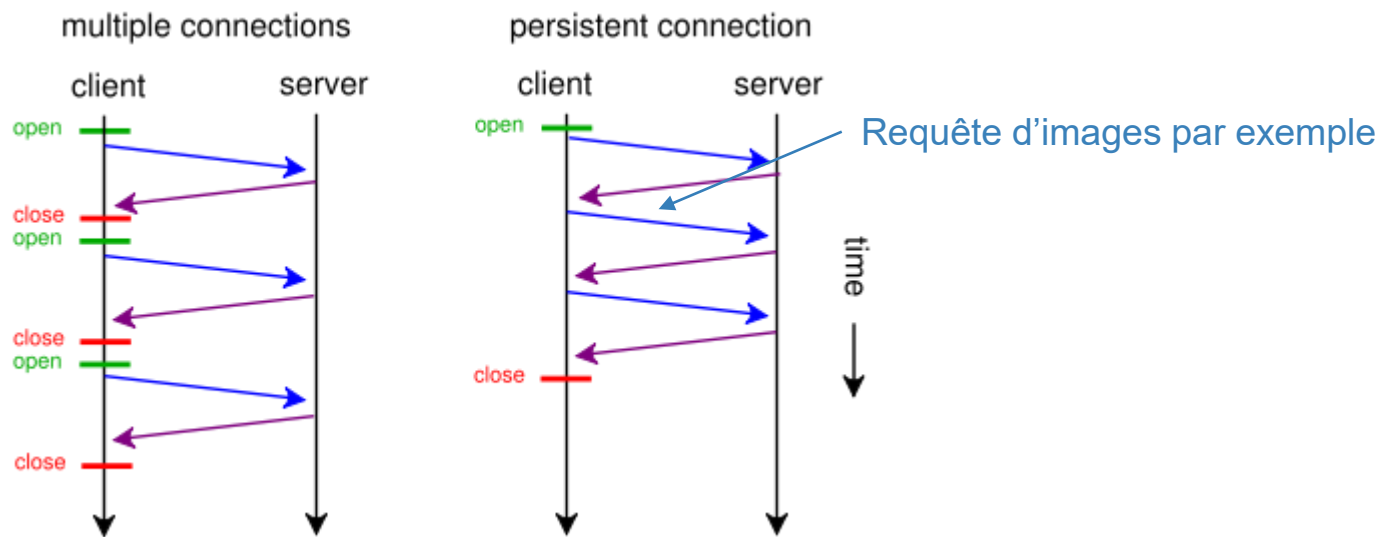
- ▷ C'est LA version utilisée dans 99% des cas
 - › HTTP 2 est apparu mais trop peu utilisé pour le moment
- ▷ Meilleure gestion du cache
 - › Entête Host désormais obligatoire
- ▷ Cette version a amené 3 nouvelles fonctionnalités:
 - › Mode **keep-alive** ;
 - › Les requêtes **Pipelined** ;
 - › Requetes et réponses **Chunked**.
- ▷ Des connexions persistantes, la connexion n'est pas fermée après une requête et reste disponible pour une nouvelle requête

Problème du HTTP1

- ▷ Quand le serveur renvoie les informations au client, il renvoie d'abord le HTML
- ▷ Puis le HTML va indiquer au serveur ce dont il a besoin
 - › Images
 - › Feuilles de style
 - › Vidéos
 - › Etc.
- ▷ Ce délai provoque de nombreux allers-retours et ralentit l'affichage
- ▷ **Webpack réunit nos fichiers pour faire le moins d'aller retour possible**

HTTP 1.1 – Keep Alive

- ▷ Keep Alive permet de maintenir la connexion actuelle « ouverte »
 - › Si on demande 3 pages d'affilées, cela évite d'ouvrir 2 connexions en +
 - › Ouvrir un socket et le refermer coute cher, on préfère éviter




- ▷ Nous aurons bientôt une révolution avec « Server Push » en HTTP 2

Le code splitting

- ▷ Parfois, il peut être plus intéressant de profiter du système du **Keep-Alive** ainsi que de la **parallélisation des requêtes** par le navigateur
 - › Au lieu de renvoyer 1 et unique fichier conséquent
- ▷ On pourra également gérer un ordre de priorité pour optimiser la vitesse d'affichage
- ▷ Par défaut, Angular applique déjà le code Splitting :
 - › Pour chaque point d'entrée (défini dans la config webpack) ;
 - › Pour tous les modules chargés dynamiquement (**loadChildren**)

Lazy loading

```
{  
  path: 'projects',  
  loadChildren: './+printing-projects/printing-  
projects.module#PrintingProjectsModule'  
},  
{  
  path: 'my-projects',  
  loadChildren: './+my-projects/my-  
projects.module#MyProjectsModule'  
},  
{  
  path: 'opportunities',  
  loadChildren: () => import('./lazy/lazy.module').then(m =>  
m.NomModule)  
},
```



ANGULAR 8

Questions?