



Introduction cours avancé

Animé par Mazen Gharbi

Présentation



Introduction

```
public ngOnInit() {  
  let user = new User();  
  user.name = 'Mazen GHARBI';  
  user.addSkills(['Angular', 'React', 'Vue', 'NodeJS', 'PHP', 'Symfony', ...LIST_OTHERS]);  
  user.company = '';  
  user.email = 'mazenGharbi@gmail.com';  
}
```

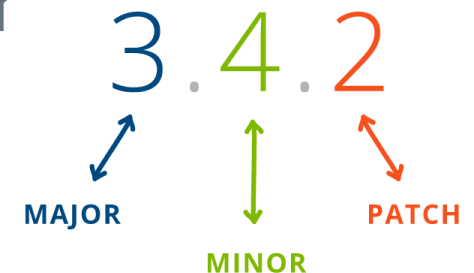
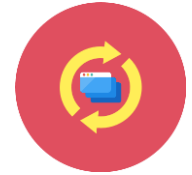


Présentation

Et vous?

Présentation des mises à jours

- ▶ Angular suit un cycle de mise à jour à rythme régulier
 - › 1 majeure tous les 6 mois
 - › 1 mineure par semaine
- ▶ Un outil existe pour nous accompagner lors des mises à jour
 - › Angular Update Guide
- ▶ Si un **break-change** survient, les éléments dépréciés restent tout de même disponibles durant 2 major versions



Angular 4 – pas d'Angular 3 !

▷ Compatibilité avec TypeScript 2.1

```
const obj = {a: 1, b: 2, c: 3}  
let maProp: keyof obj; // 'a' | 'b' | 'c'
```

```
let { a, b, ...array } = values;
```

async / await



▷ Modularisation du package @angular/animations › Et mise en place de Renderer2

▷ Amélioration des directives structurelles *ngIf et *ngFor

```
<div *ngIf="isTrue; else notTrue">Bonjour</div>  
<ng-template #notTrue>Aurevoir</ng-template>
```

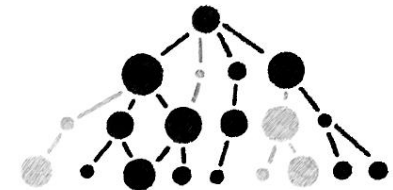
```
<div *ngIf="user$ | async as user">  
  Hi, {%raw%}{{ user.name }}!{%endraw%}  
</div>
```

▷ Réduction du poids du build (en AOT) - View Engine › Réduction de 60% environ du code JS généré

Angular 5



- ▷ Simplification de la création des [PWA](#)
- ▷ Abandon de **Http** au profit de **HttpClient**
- ▷ Nouveaux pipes Number / Dates et Currency
 - › Avec gestion du i18n
- ▷ Optimisation du build
 - › Décorateurs retirés du build / Implémentation du Tree Shaking
- ▷ Validation de formulaire améliorée



```
// updateOn -> blur / submit /  
this.newUserForm = this.fb.group({  
  userName: ['Bob', { updateOn: 'submit', validators: [Validators.required] } ]  
});
```

→ Ré-interprété uniquement à la validation

Angular 6



▷ Amélioration de la CLI !

› « ng **add** ... » / « ng **update** ... »

▷ Nouvelle déclaration des services

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

▷ Angular elements


› [Plus d'informations ici](#)



Angular 6 – Angular elements

- ▷ Les éléments Angular sont des composants configurés et préparés pour être utilisés en tant que **Web Components**
 - › Autrement appelés des Custom Elements
- ▷ Les custom elements sont des composants natifs à JavaScript
 - | `"document-register-element": "^1.7.2",`
- ▷ Mais.. Ce n'est pas ce qui est utilisé en Angular, nous manipulons un équivalent
 - › Problématique de **rétro-compatibilité**
- ▷ Angular nous propose de convertir ses composants en composants natifs

Pourquoi utiliser les Angular elements

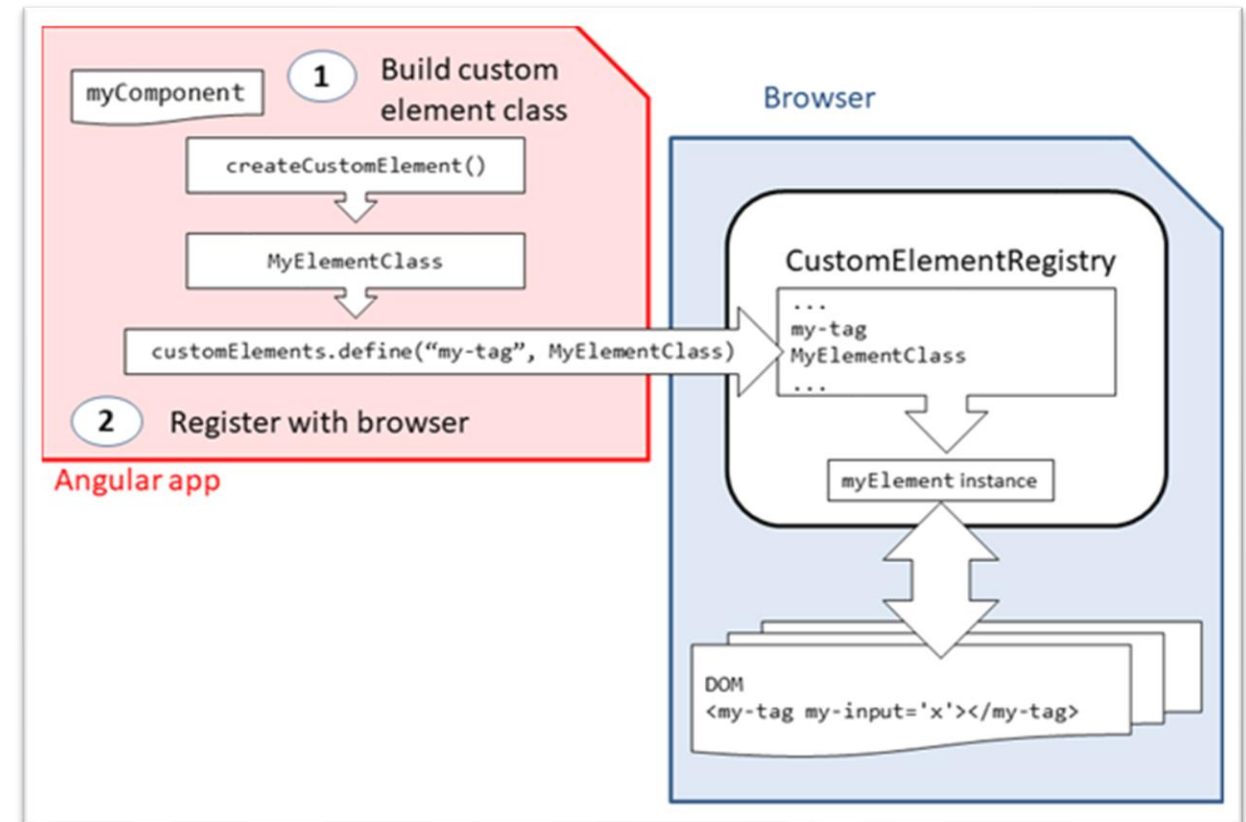
- ▷ Une ré-utilisabilité vrai de vrai ! 
- ▷ Optimiser le SSR
 - › Server Side Rendering
 - › Les web component n'attendent pas le bootstrap time
- ▷ Développement d'une même application en **sous-applications**
 - › Spotify Squad

Angular 6 – Angular elements

► Pour y arriver, il faut commencer par installer certaines dépendances :

```
> ng add @angular/elements
```

↓
"document-register-element": "^1.7.2",

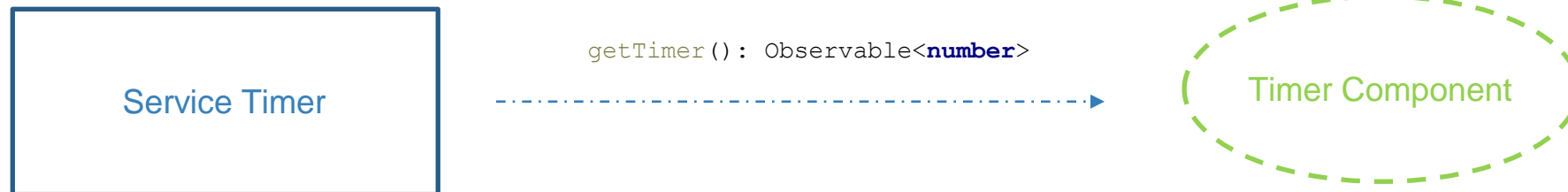


Angular 6 – Angular elements

- ▷ Créons une application très simple :

Compteur : 0

- ▷ L'architecture est classique :



Angular 6 – Angular elements

timer.service.ts

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

  }

  public getTimer(): Observable<number> {
    return interval(1000).pipe(startWith(0));
  }
}
```

timer.component

```
export class TimerComponent implements OnInit {
  public $timer: Observable<number>;

  constructor(private timerService: TimerService) {
  }

  ngOnInit(): void {
    this.$timer = this.timerService.getTimer();
  }
}
```

```
<p>
  Compteur : {{ $timer | async }}
</p>
```

Angular 6 – Angular elements

- ▷ Il est nécessaire d'ajouter notre composant dans la propriété **entryComponent** de votre module :

```
@NgModule({  
  declarations: [  
    ...  
  ],  
  imports: [  
    ...  
  ],  
  providers: [],  
  entryComponents: [TimerComponent]  
})
```

app.module.ts

- ▷ Cette propriété permet d'y déclarer les composants qui ne seront pas appelés dans un template
- ▷ Depuis Angular 9, ce n'est plus nécessaire

Angular 6 – Angular elements

▷ Toujours dans le module :

```
import { createCustomElement } from '@angular/elements';

export class AppModule {
  constructor(injectorModule: Injector) {
    const custom = createCustomElement(AppComponent, {injector: injectorModule});
  }
}
```

- ▷ Ainsi, nous enregistrons un custom elements côté Angular
 - › Angular se chargera de gérer automatiquement toutes les dépendances avec d'autres composants et nos services
- ▷ Il s'agit maintenant de le configurer pour JavaScript

```
customElements.define('app-hello-world', custom);
```

 Mot-clé Javascript vanilla, [voir ici](#)

Compiler tout ça

- ▷ Ensuite, il s'agira simplement de transpiler le projet et de récupérer le code JavaScript dans vos projets

```
> ng build
```

- ▷ Puis il s'agira de récupérer ces fichiers dans un nouveau projet afin d'utiliser notre composants :

```
<app-timer></app-timer>
```

```
<script type="text/javascript" src="runtime.js"></script>
```

```
<script type="text/javascript" src="polyfills.js"></script>
```

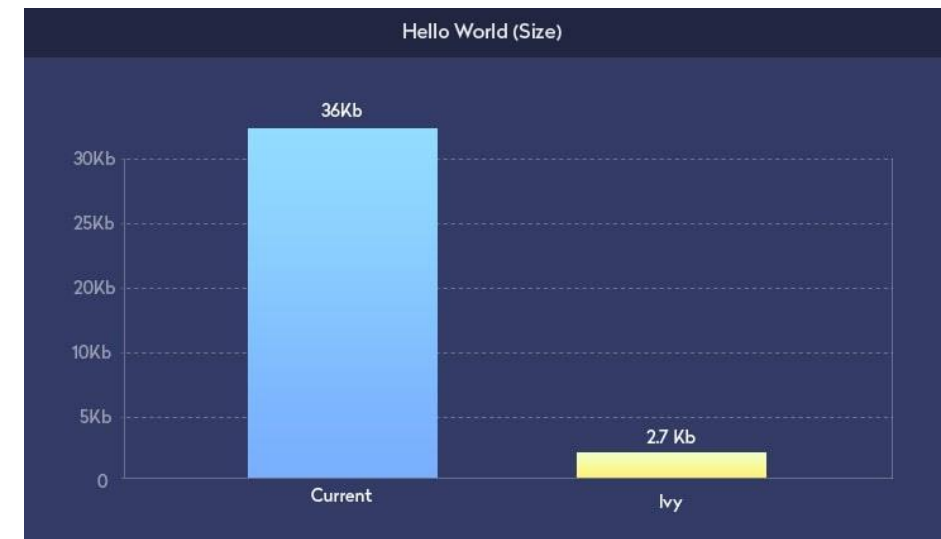
```
<script type="text/javascript" src="main.js"></script>
```

- ▷ Il existe des techniques pour concaténer ces 3 fichiers sous 1 seul qu'on peut appeler `timer.js`

Angular 7



- ▷ Amélioration du Component Dev Kit
 - › Virtual Scroll / Drag & Drop / etc.
- ▷ Mise en place de « **budgets de build** » dans le angular.json
- ▷ Et c'est le début d'une grande épopée...
- ▷ **IVY RENDER**



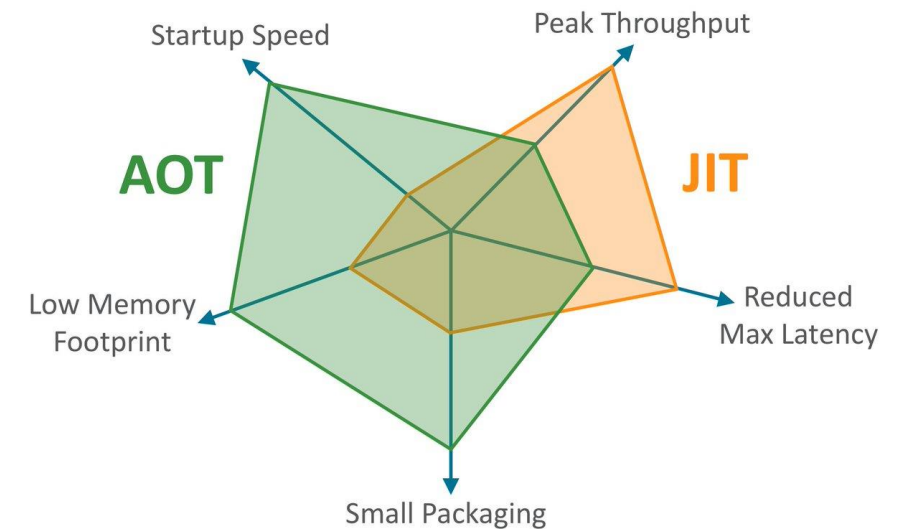
Angular 7 - Ivy Render

▷ Votre code peut être exécutés dans 2 contextes différents :

▷ 1. Ahead of time Compilation (**AOT**)

▷ 2. Just in time Compilation (**JIT**)

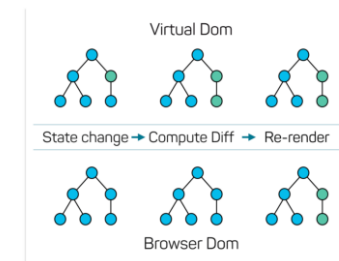
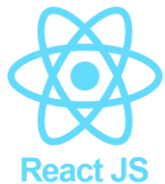
▷ Dans tous les cas, votre code est **transpilé** !



[Thomas Wuerthinger](#)

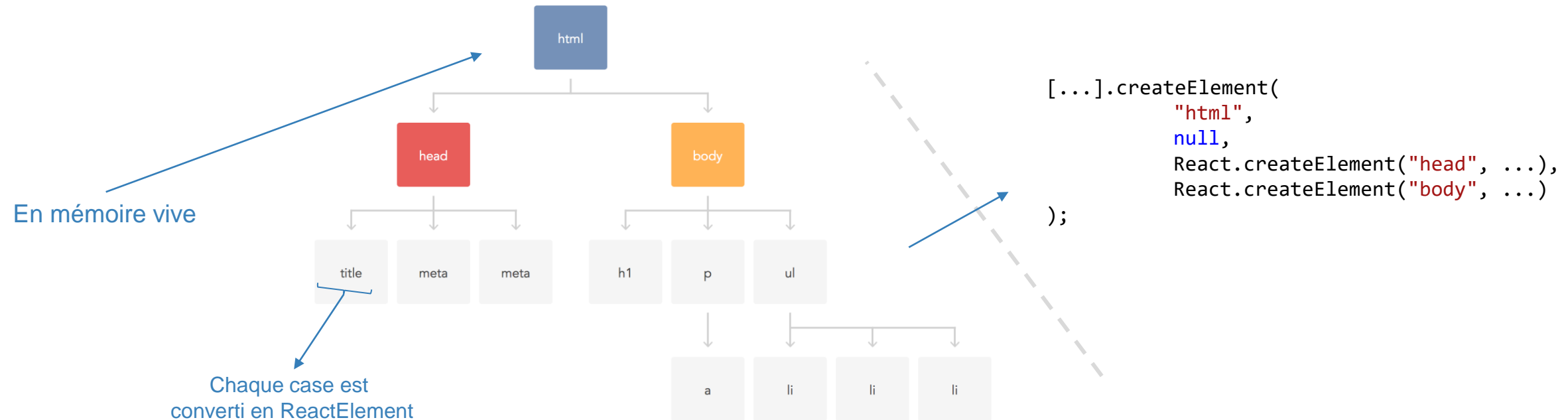
Angular 7 - Ivy Render

- ▷ Avant Ivy Render, il y avait « **View Engine** »
- ▷ Ce qui est long pour les frameworks JS, c'est le **ré-affichage de la page**
 - › Et non pas, comme on pourrait croire, la manipulation des nœuds;
 - › [Discussion reddit](#) qui en parle très bien ;
- ▷ Pour régler ça, React et Vue implémentent le **Virtual DOM**

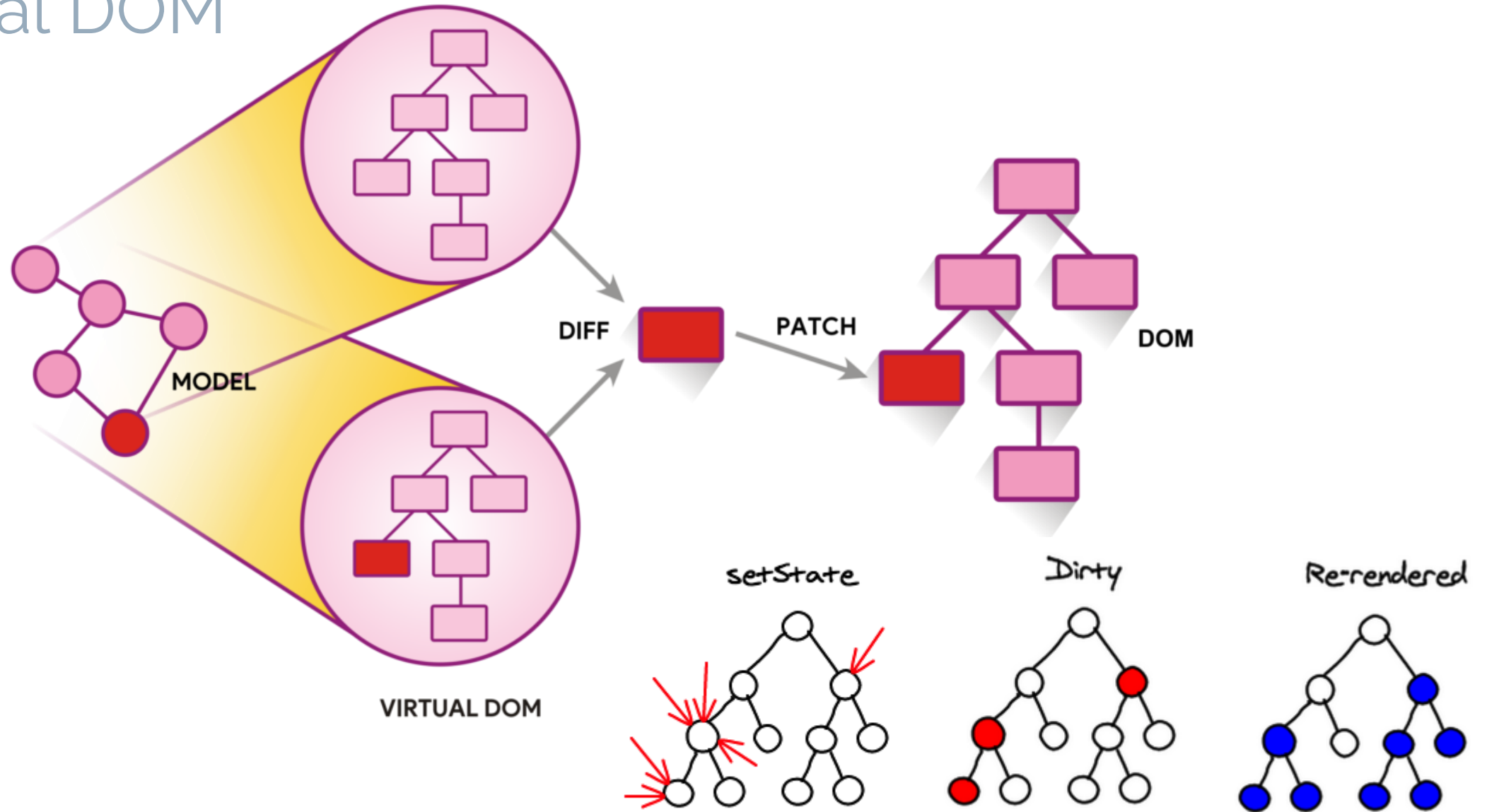


Virtual DOM

- ▷ React et Vue construisent une représentation **virtuel** du DOM actuel ;
 - › *Ne pas confondre avec le Shadow DOM*
- ▷ Chaque nœud de l'arbre est représenté par un **objet Javascript** ;



Virtual DOM



Et Angular dans tout ça ?

- ▷ Faisons un focus sur **View Engine** pour commencer ;
- ▷ **ViewEngine = Renderer2**, un service que vous pouvez injecter
 - › Permet de représenter votre vue avec du code JavaScript

```
constructor(private renderer: Renderer2) {
  this.renderer.setProperty(this.element, 'innerText', newValue);
}
```

- ▷ Un simple composant comme celui de gauche va être convertit comme cela :

```
@Component({
  selector: 'app-root',
  template: `Hello {{name}}`
})
export class AppComponent {
  public name = 'Médor';
}
```

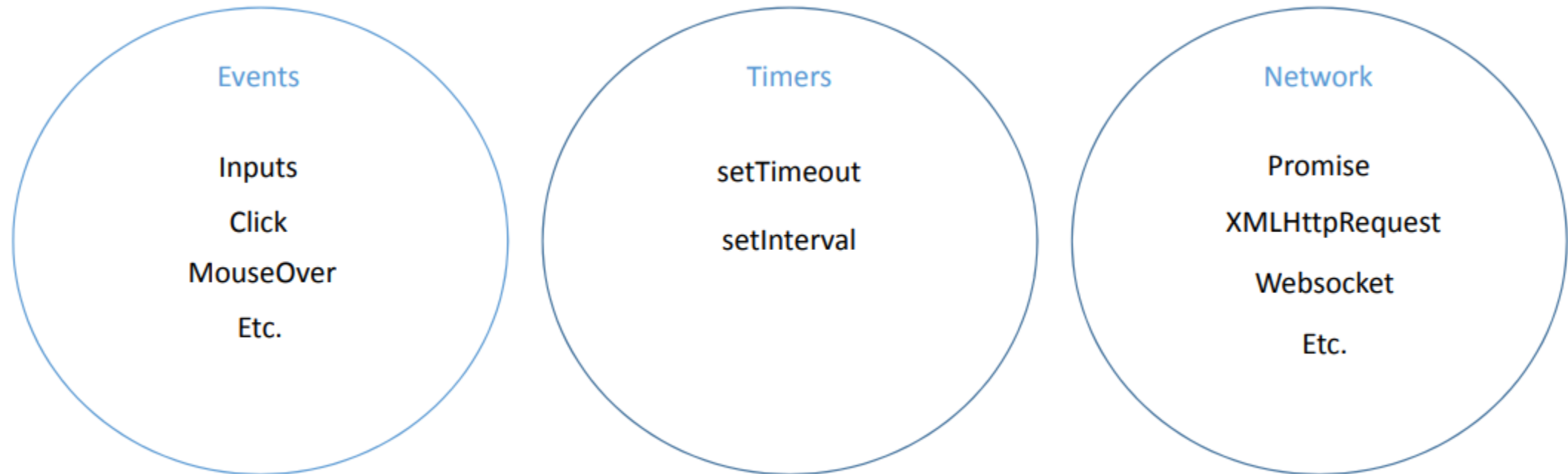
i0 représente le binding

app.component.ngfactory.js

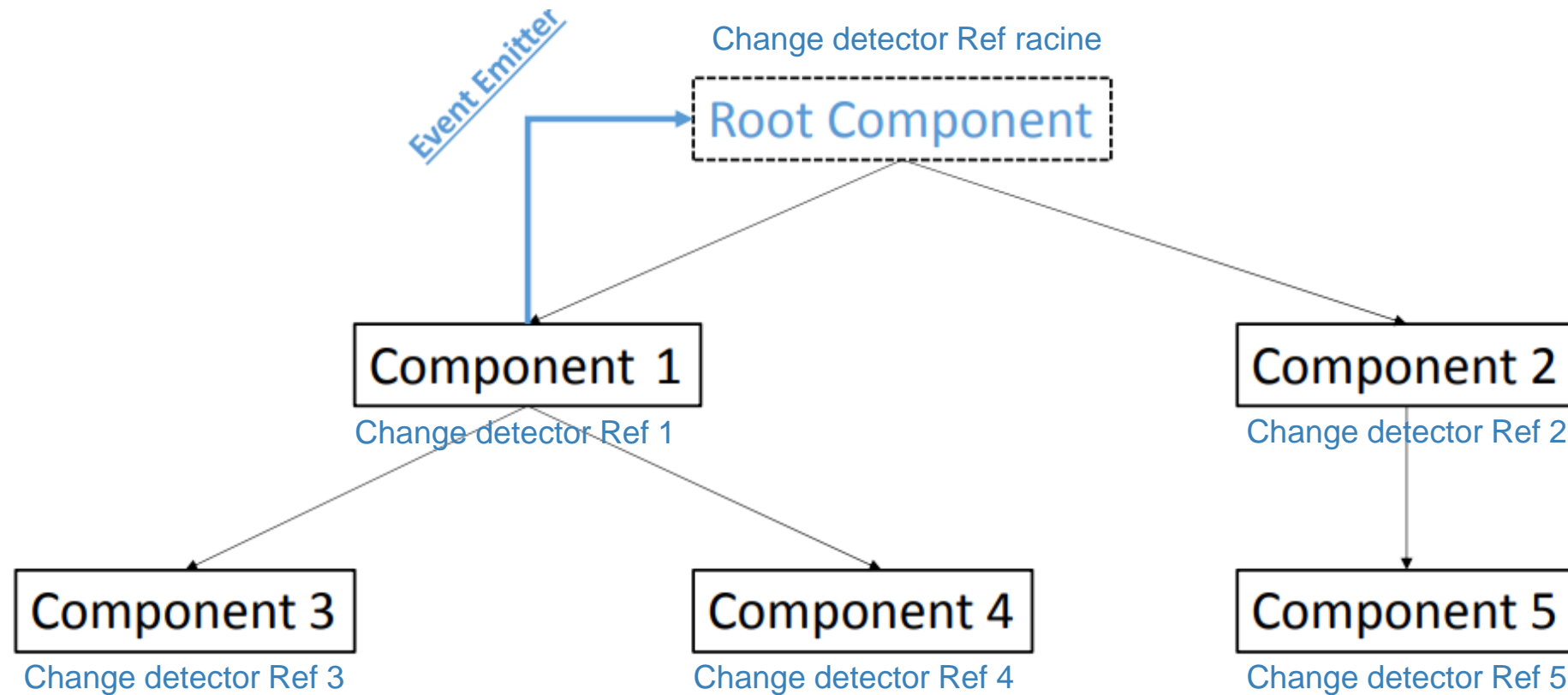
Généré par ViewEngine

```
"use strict";
/**
 * @fileoverview This file was generated by the Angular template compiler. Do not edit.
 *
 * @suppress {suspiciousCode,uselessCode,missingProperties,missingOverride,checkTypes}
 * tslint:disable
 */
Object.defineProperty(exports, "__esModule", { value: true });
var i0 = require("@angular/core");
var i1 = require("../app.component");
var styles_AppComponent = [];
var RenderType_AppComponent = i0.ɵɵrt({ encapsulation: 2, styles: styles_AppComponent, data: {} });
exports.RenderType_AppComponent = RenderType_AppComponent;
function View_AppComponent_0_1() { return i0.ɵɵvid(0, [i0.ɵɵl()(), i0.ɵɵtd(0, null, ["Hello ", ""])]),
  null, function (_ck, _v) { var _co = _v.component; var currVal_0 = _co.name; _ck(_v, 0, 0, currVal_0); }); }
exports.View_AppComponent_0 = View_AppComponent_0;
function View_AppComponent_Host_0_1() { return i0.ɵɵvid(0, [i0.ɵɵl()(), i0.ɵɵeld(0, 0, null, null, 1, "app-root",
  [], null, null, null, View_AppComponent_0, RenderType_AppComponent)], i0.ɵɵdid(1, 49152, null, 0, i1.AppComponent, [], null, null)],
  null, null); }
exports.View_AppComponent_Host_0 = View_AppComponent_Host_0;
var AppComponentNgFactory = i0.ɵɵccf("app-root", i1.AppComponent, View_AppComponent_Host_0, {}, {}, []);
exports.AppComponentNgFactory = AppComponentNgFactory;
//# sourceMappingURL=app.component.ngfactory.js.map
```

Les zones Angular



Change detectors



► Dès qu'un CD repère un changement, on enclenche le cycle de Dirty Checking

Code permettant à Angular de chercher quelle propriété a changé

```

1  (function(exports,styles,resolvedMetadataCache,ToDoList,viewFactory_TodoList0,AbstractChangeDetector,Change
2  /**/) {
3
4      var ChangeDetector_App_0 = function ChangeDetector_App_0() {
5          AbstractChangeDetector.call(
6              this, "App_0", 2,
7              ChangeDetector_App_0.gen_propertyBindingTargets, ChangeDetector_App_0.gen_directiveIndices,
8              5);
9          this.dehydrateDirectives(false);
10     }
11
12     ChangeDetector_App_0.prototype = Object.create(AbstractChangeDetector.prototype);
13
14     ChangeDetector_App_0.prototype.detectChangesInRecordsInternal = function(throwOnChange) { throwOnCha
15         var _l_context = this.context,_l_todos0,_l_DoCheck1; _l_context = App {todos: Array[3]}, _l_todos0 = [
16         var isChanged = false; isChanged = false
17         var changes = null; changes = null
18
19
20         this.propertyBindingIndex = 0;
21
22         _l_todos0 = _l_context.todos; _l_todos0 = [Object, Object, Object], _l_context = App {todos: Array[3]}
23
24
25         if (throwOnChange && !ChangeDetectionUtil.devModeEqual(this.todos0, _l_todos0)) { throwOnChange = 1
26             this.throwOnChangeError(this.todos0, _l_todos0);
27         }
28
29         if (ChangeDetectionUtil.looseNotIdentical(this.todos0, _l_todos0)) { _l_todos0 = [Object, Object, Obj
30
31         this.directive_0_0.todos = _l_todos0; _l_todos0 = [Object, Object, Object]
32
33         isChanged = true; isChanged = false
34
35         this.todos0 = _l_todos0; _l_todos0 = [Object, Object, Object]
36     }
37
38
39
40
41
42
43
44     if (!throwOnChange) this.directive_0_0.ngDoCheck();
45
46     changes = null;
47
48     isChanged = false;
49
50
51
52
53
54

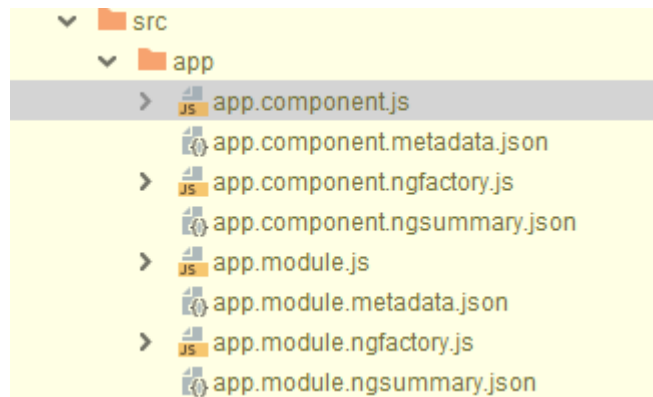
```

Enclenché par le CD

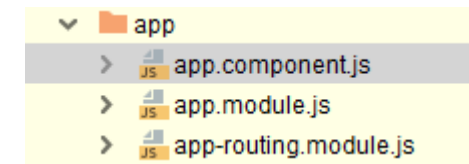
Si la propriété est différente,
on tag le composant dirty

Revenons à Ivy

- ▷ Comme énoncé, Ivy permet de fortement réduire la taille du build



Angular 4

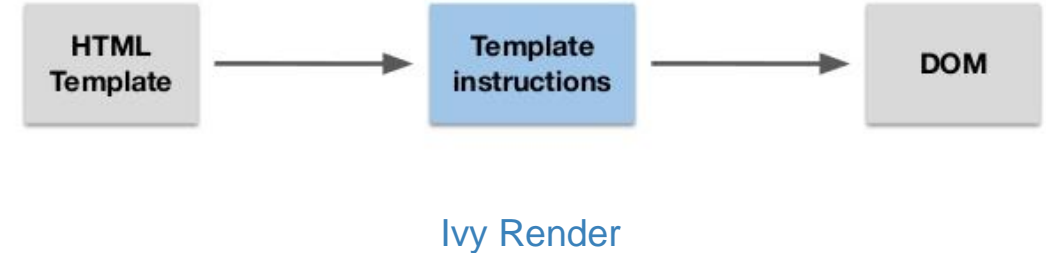
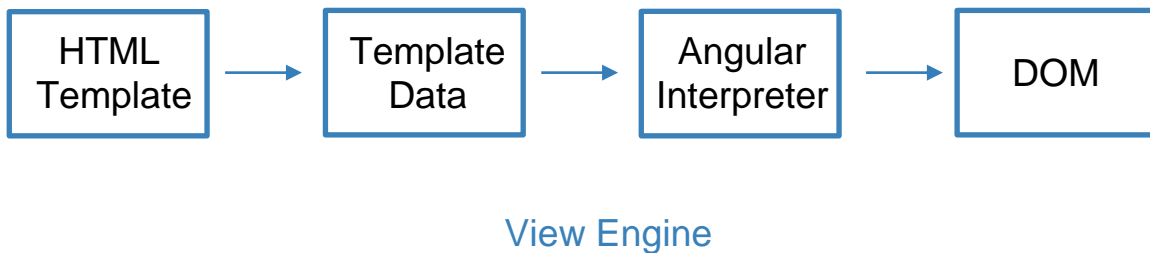


Angular 11

- ▷ On a le choix d'utiliser Ivy ou View Engine
 - › Depuis Angular 9, Ivy est le choix par défaut

Ivy render, un couteau suisse

- ▷ Ivy, en plus de traiter l'affichage des composant, agit comme un **interpréteur** !
 - › Réalise des **optimisations** sur le code et son poids
 - › Là ou ViewEngine déléguait ce traitement
- ▷ Crée les nœuds DOM et exécute la détection des changements directement !



Angular 8

- ▷ Mise à jour vers TypeScript 3.4
- ▷ On en apprend plus sur Ivy – 3 objectifs :
 1. Améliorer le temps de build ;
 2. Diminuer la taille du build ;
 3. Agrémenter Angular de nouvelles fonctionnalités ! (Meta programming etc.)
- ▷ Nouvelle écriture du lazy loading

```
loadChildren: () => import('./calendar/calendar.module').then(m => m.CalendarModule)
```



Angular 9

ANGULAR 9



▷ Ca y est, **Ivy** est là.

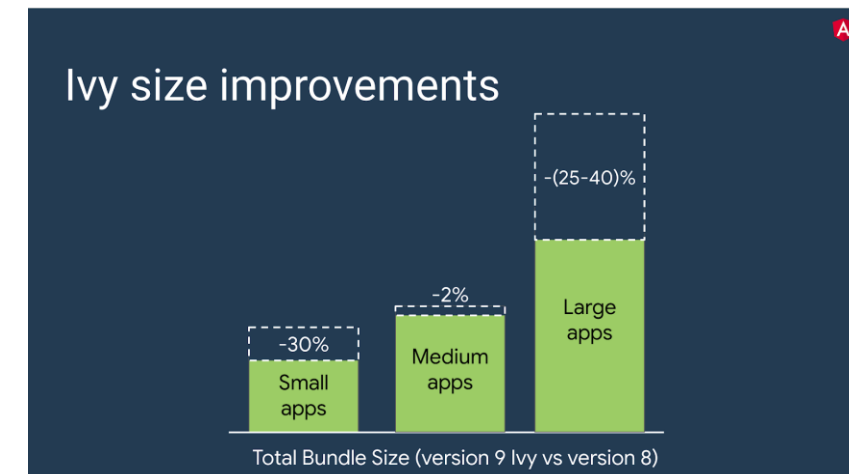
1. Les petites applications qui n'utilisent pas de nombreuses fonctionnalités profitent du **Tree Shaking** ;
2. Les grandes applications avec de nombreux composants profitent du build léger grâce au **factory** ;
3. Les applications de taille moyenne devraient voir des tailles de bundle équivalentes ou légèrement plus petites, car elles bénéficient moins du Tree Shaking et ne disposent pas de suffisamment de composants pour vraiment tirer parti du factory

▷ **TestBed** a été adapté pour Ivy

- › Les tests sont donc plus rapides en moyenne

▷ Meilleure gestion de logs et d'erreur

▷ Support de [TypeScript 3.7](#)



Angular 10

▷ Le compilateur est plus strict !

```
> ng new --strict
```

1. Active le mode strict dans TypeScript
2. Active le mode strict du « [template type checking](#) »
3. Les budgets des lots par défaut ont été réduits d'environ 75%
4. Configure les règles du « linter » pour empêcher les déclarations de type any

▷ Angular n'est plus compatible avec EcmaScript 5

▷ Enfin, durant cette version, l'équipe s'est occupé des issues Github !



We've dramatically increased our investment in working with the community. In the last three weeks our open issue count has decreased by over 700 issues across [framework](#), [tooling](#), and [components](#). We've touched over 2,000 issues, and we plan to make large investments over the next few months, working with the community to do even more.

Angular 11

▷ Optimisation du temps d'affichage

- › Les fonts vont être téléchargé directement lors du build et non pas à l'affichage

▷ Amélioration des tests unitaires avec les « harness »

- › Permet de débbugger aisément les éléments Material

```
beforeEach(() => {  
  fixture = TestBed.createComponent(MyDialogButton);  
  loader = TestBedHarnessEnvironment.loader(fixture);  
  rootLoader = TestBedHarnessEnvironment.documentRootLoader(fixture);  
});  
  
it('loads harnesses', async () => {  
  dialogButtonHarness = await TestBedHarnessEnvironment.harnessForFixture(fixture, MyDialogButtonHarness);  
});
```

▷ Amélioration des logs



Angular 12

- ▷ Abandon de Protractor probable !

- › L'équipe Angular réfléchit à utiliser une autre librairie (WebdriverIO etc..)

- ▷ Opérateur ??

```
{{ time ?? getTime() }}
```

- ▷ [Nouveau guide](#) pour apprendre à manipuler la projection de composants

- ▷ La commande « ng build » réalise désormais un build de production directement

- ▷ Plus de support pour IE 11 à venir !

Angular 13


- ▷ View Engine n'est plus utilisable, Ivy devient la norme !
- ▷ Simplification de la création dynamique des composants

Avant

```
@Directive({ ... })
export class MyDirective {
  constructor(private viewContainerRef: ViewContainerRef,
               private componentFactoryResolver:
                 ComponentFactoryResolver) {}
  createMyComponent() {
    const componentFactory = this.componentFactoryResolver.
      resolveComponentFactory(MyComponent);

    this.viewContainerRef.createComponent(componentFactory);
  }
}
```

Après



```
@Directive({ ... })
export class MyDirective {
  constructor(private viewContainerRef: ViewContainerRef) {}
  createMyComponent() {
    this.viewContainerRef.createComponent(MyComponent);
  }
}
```


Angular 13

- ▷ Ca y est.. C'est la **fin pour IE11** !
- ▷ Amélioration de la AngularCLI, des performances pour les T.U. etc.
- ▷ Une mise à jour très accès sur le système interne en somme

Sommaire

1. TypeScript en détail
2. Optimisez les performances de vos applications Angular
3. Les websockets et Angular
4. Promesses et Observables
5. Adaptation selon vos besoins