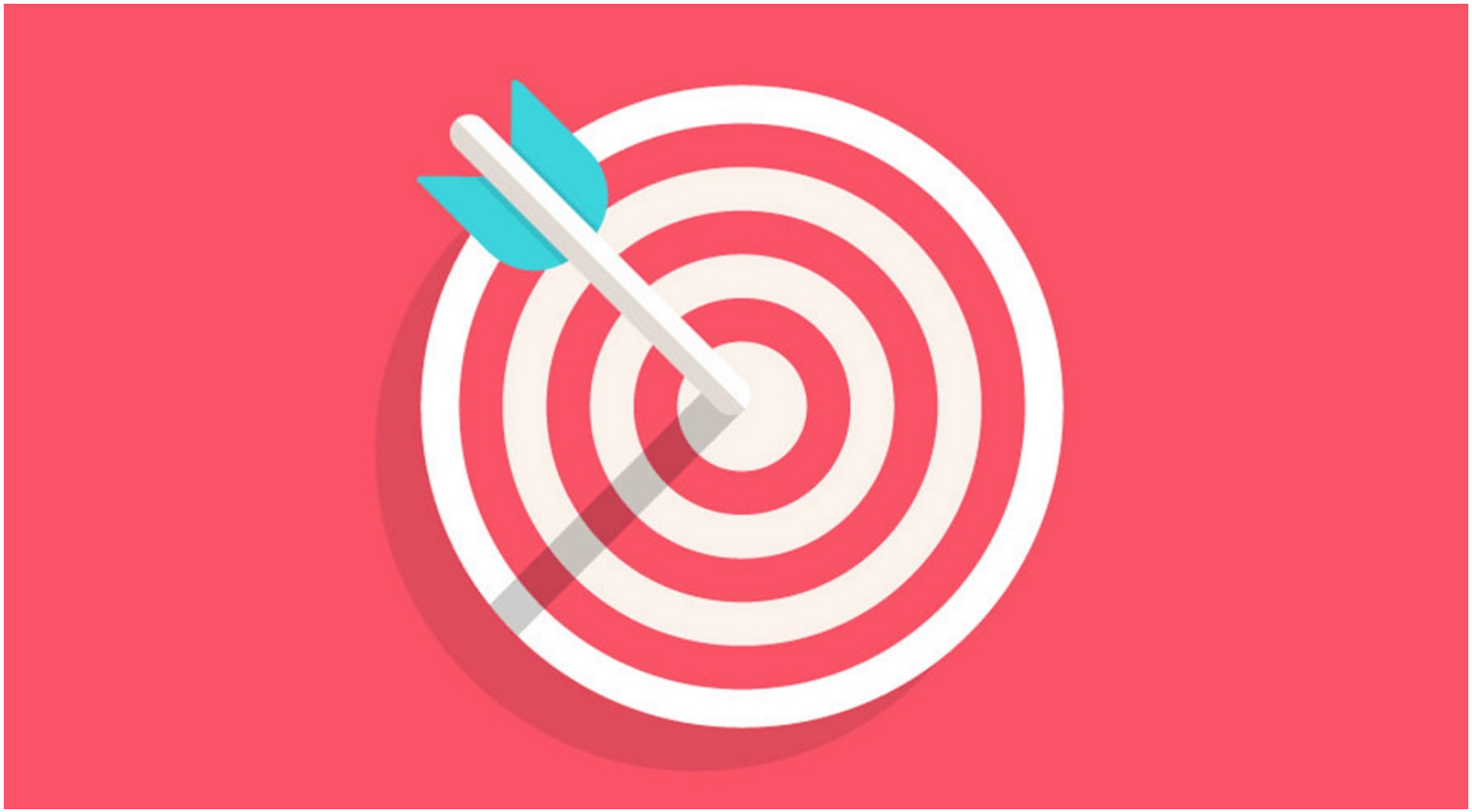
```
1
2 <h1>
3         Techno Web
4 </h1>
5
```

2 Cour n°2



3

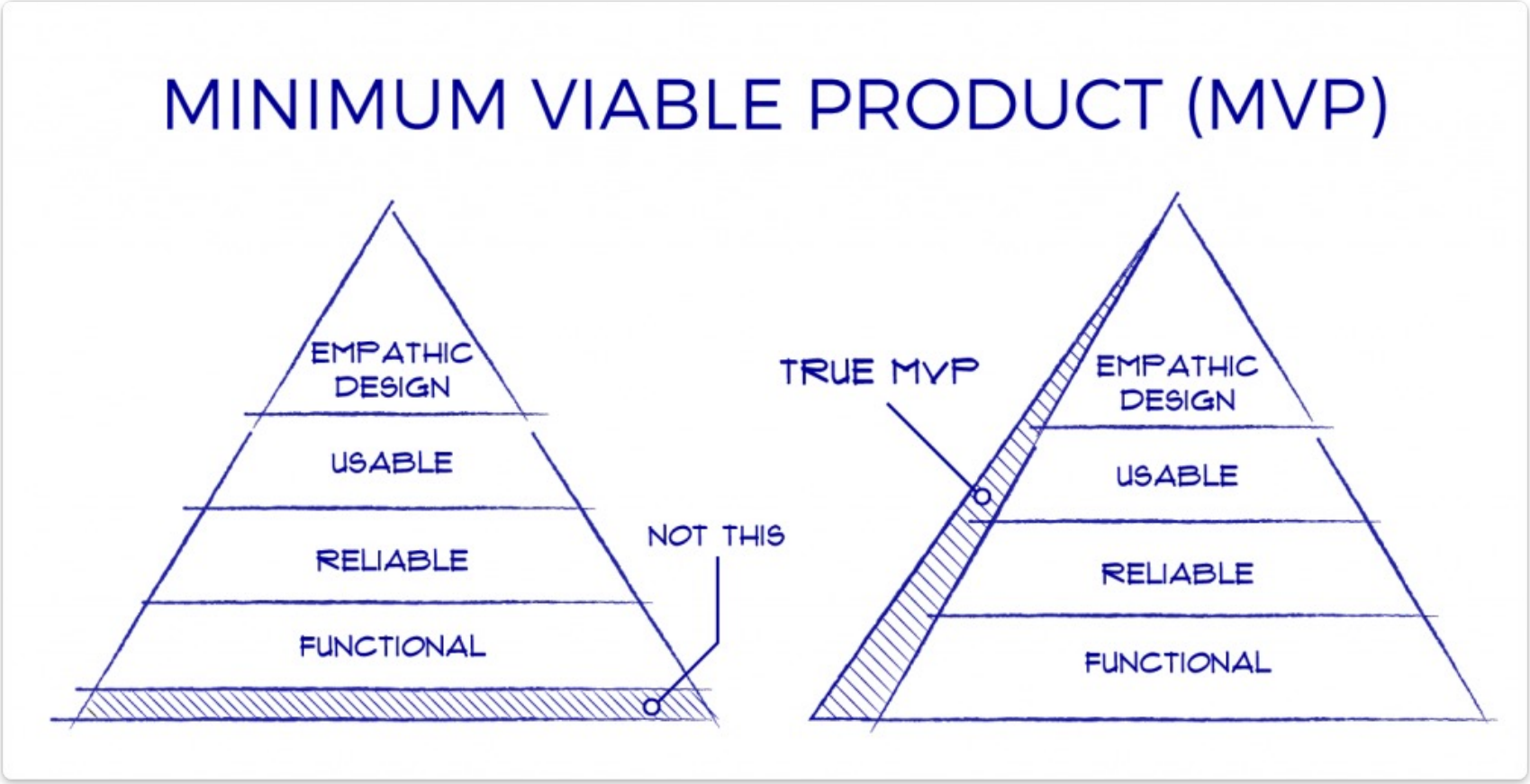
Youssef Mekouar

Doctorant en informatique

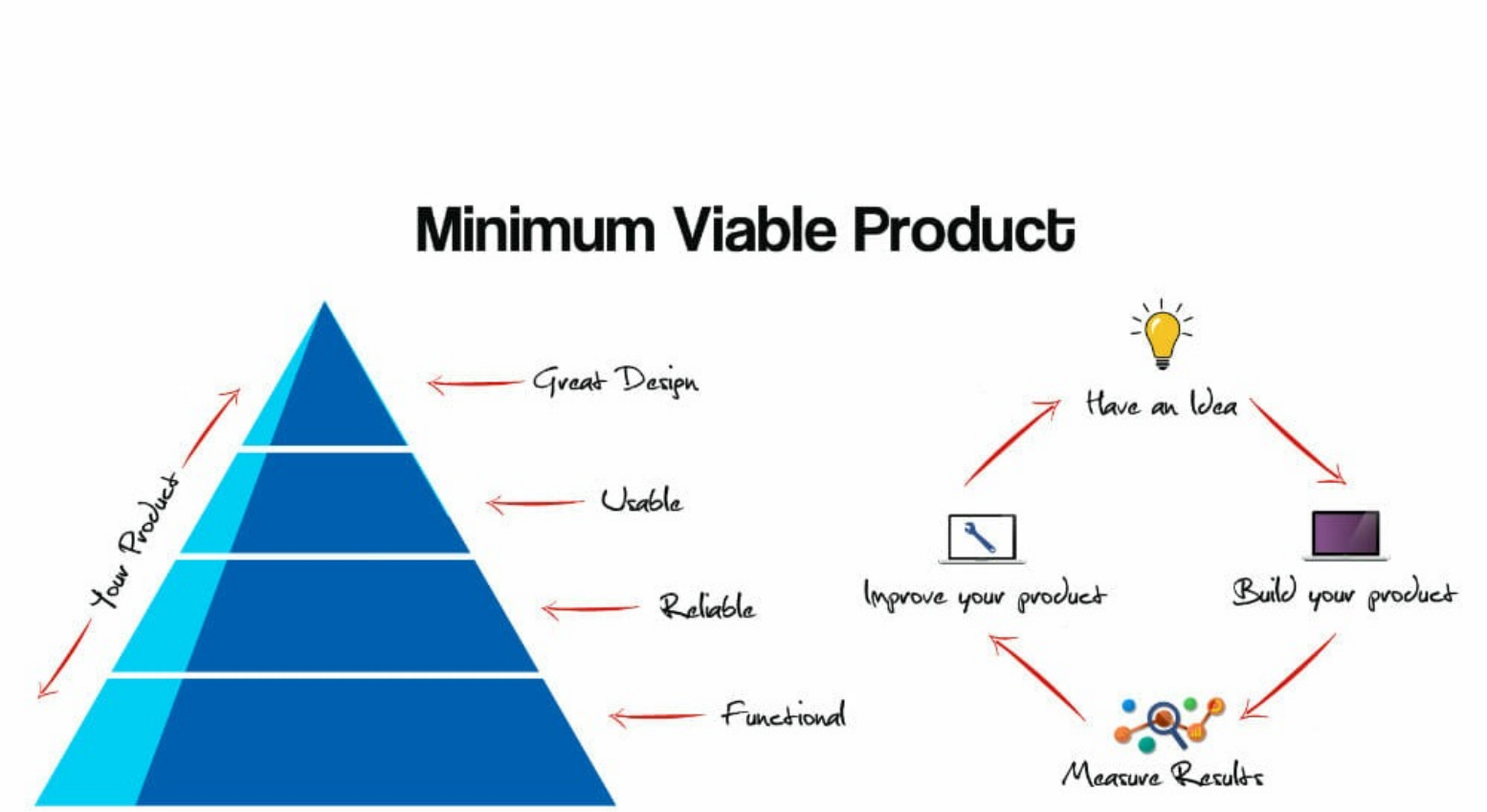
Ingénieur en informatique

Spécialité WEB IA

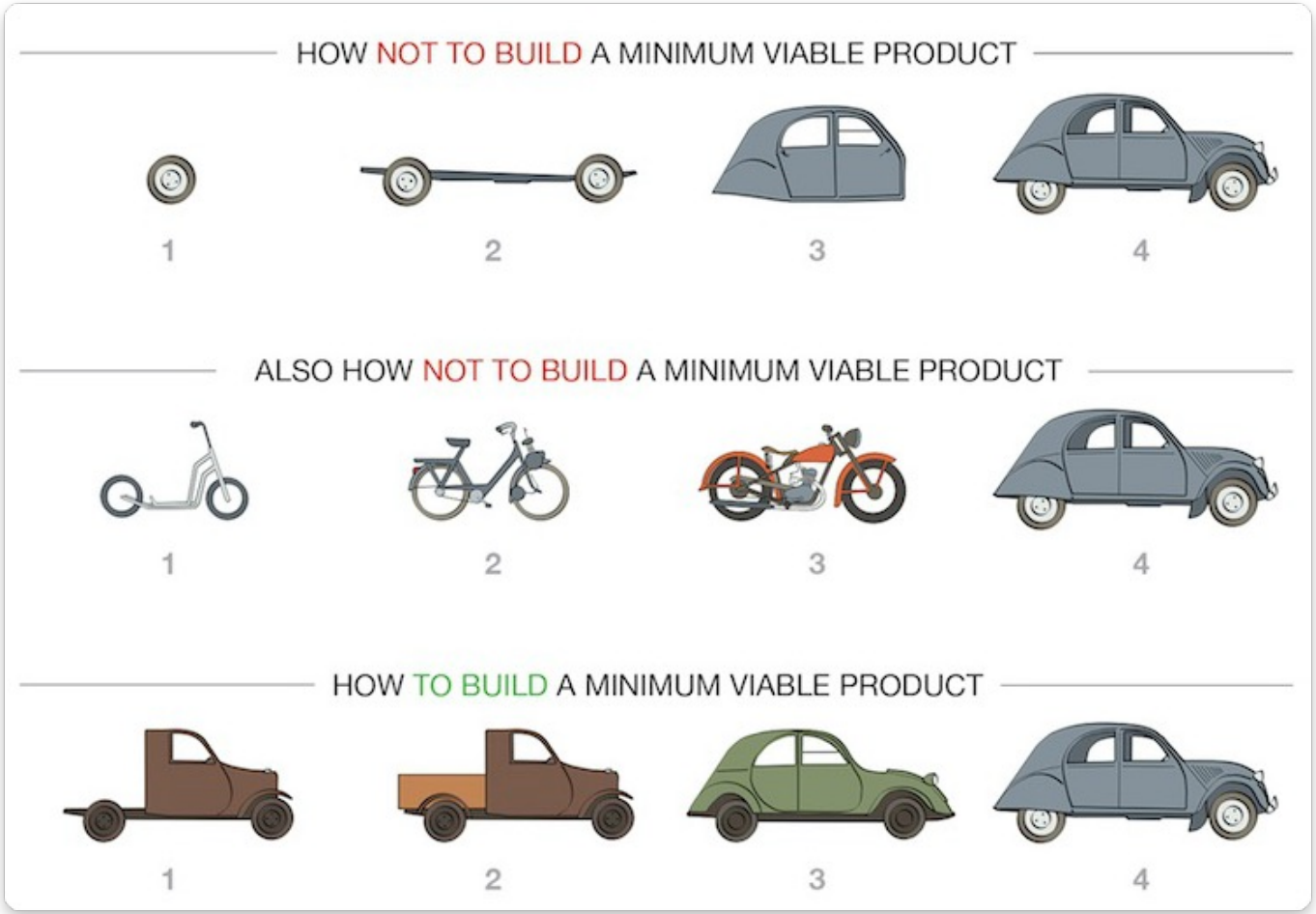
4


MINIMUM VIABLE PRODUCT (MVP)
EMPATHIC DESIGN
USABLE
RELIABLE
FUNCTIONAL
NOT THIS
TRUE MVP
EMPATHIC DESIGN
USABLE
RELIABLE
FUNCTIONAL

5


Minimum Viable Product
Great Design
Usable
Reliable
Functional
Your Product
Have an Idea
Build your product
Measure Results
Improve your product

6 Pourquoi un produit minimum viable est nécessaire ?



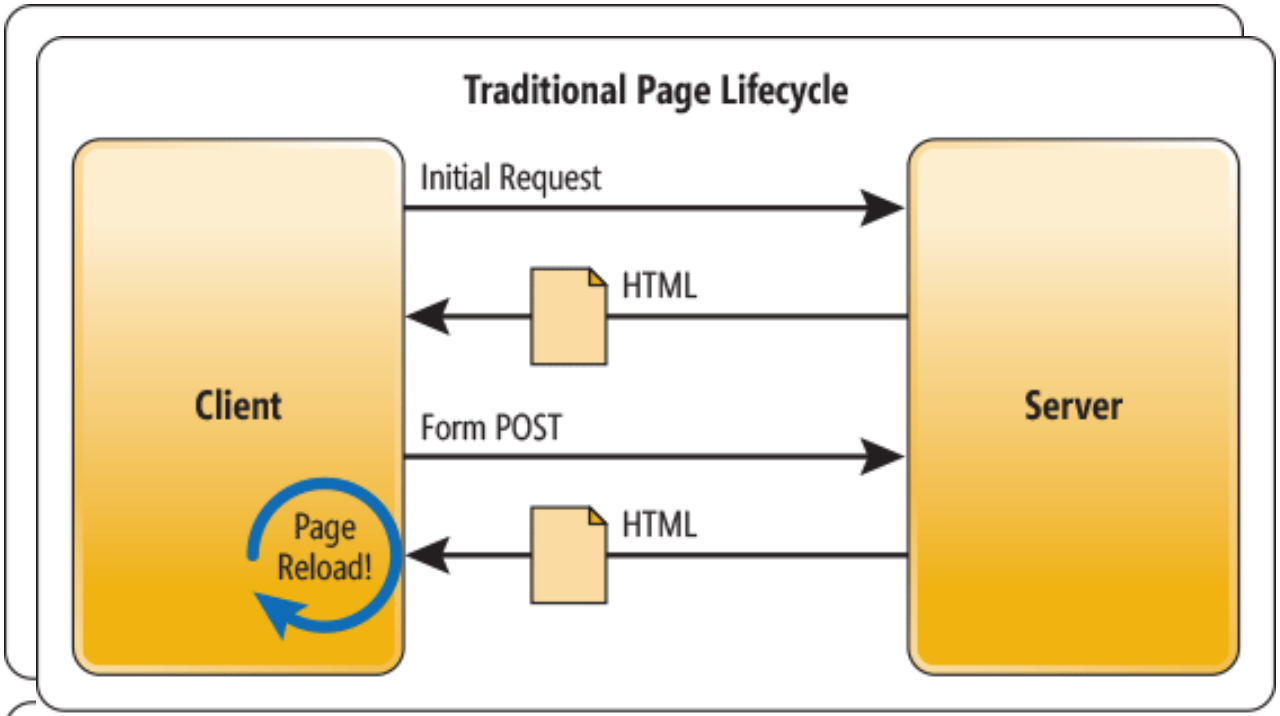
7 Qu'est-ce qu'une SPA ?

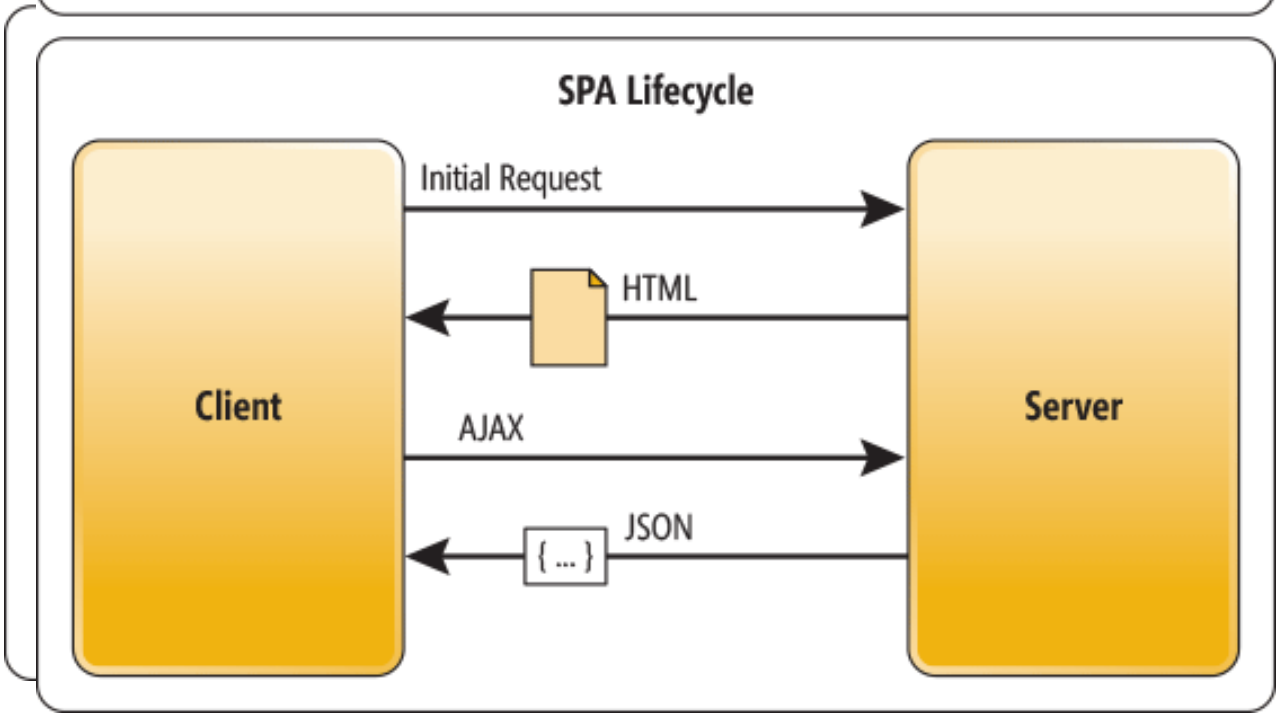
Une **SPA** (pour Single Page Application) est une **application monopage** qui exécute principalement l'interface utilisateur dans le navigateur et qui communique avec le serveur via des services web.

Les **avantages d'une SPA** par rapport à une application web standard sont :

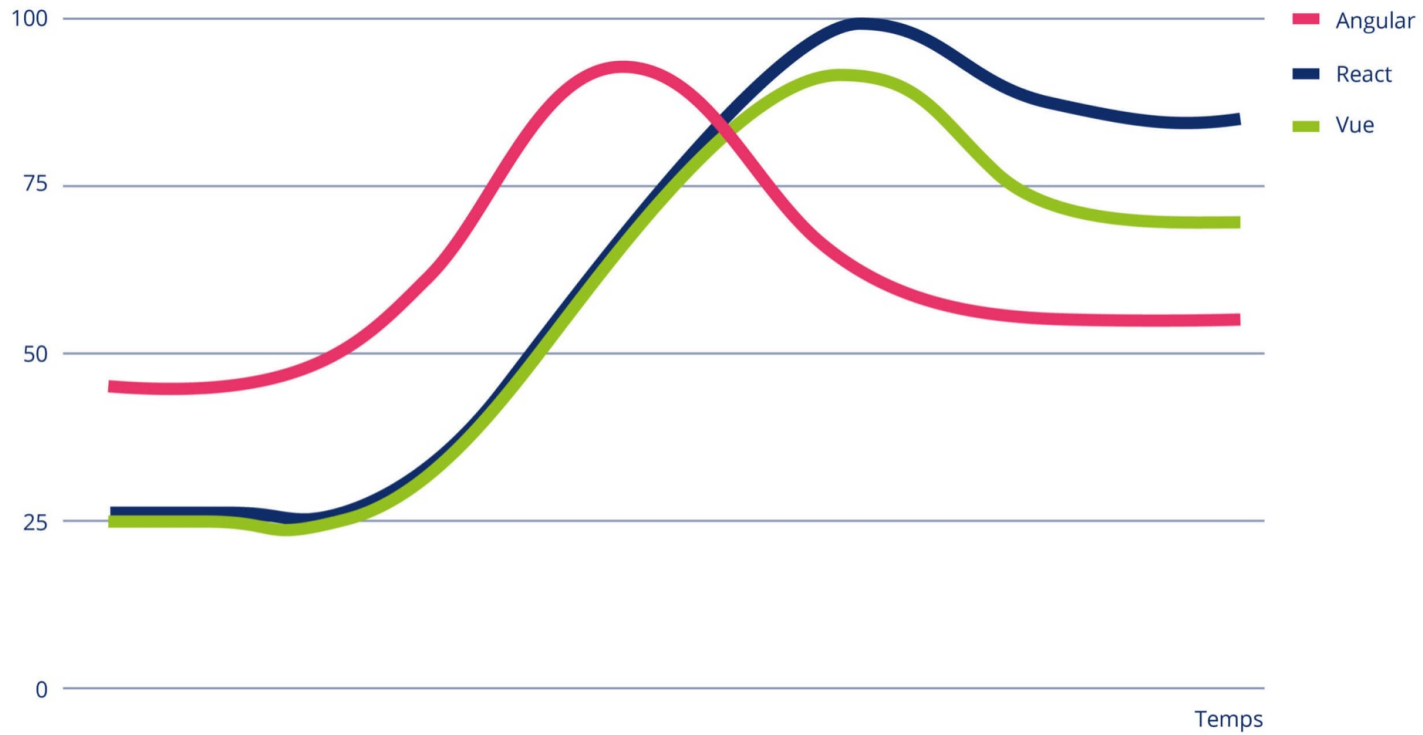
- de **fluidifier l'expérience utilisateur** en évitant de charger une nouvelle page à chaque action de l'utilisateur ;
- de **permettre de s'exécuter localement** sur le navigateur de l'utilisateur.

Les frameworks servant de base à **une application SPA** sont généralement **développés** avec les technologies web classiques : **HTML5, CSS3, Javascript ou Typescript.**


Traditional Page Lifecycle
Client
Server
Initial Request
HTML
Form POST
HTML
Page Reload!


SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

9 Apprentissage d'Angular



Courbes d'apprentissage des 3 frameworks

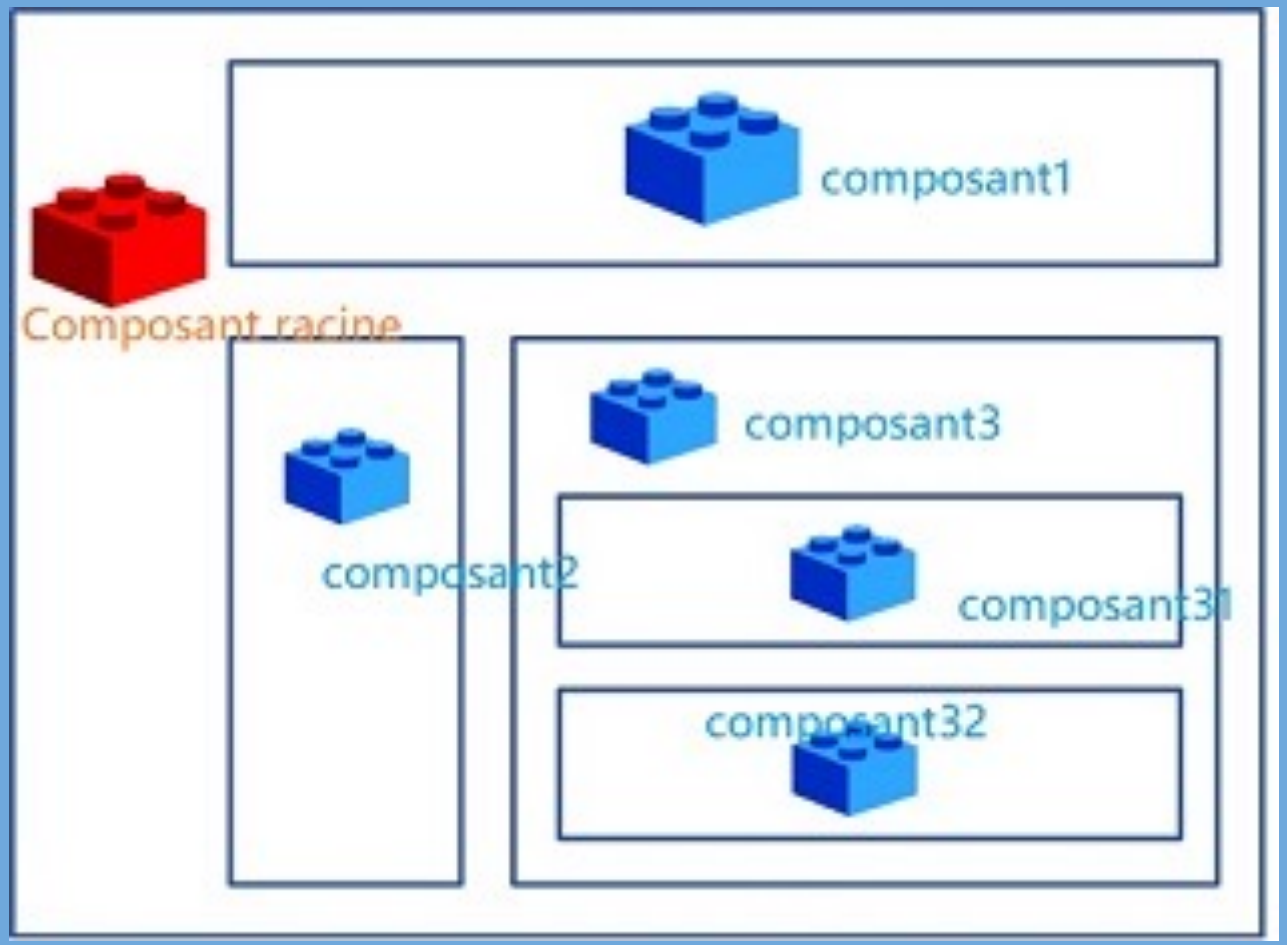
10 ANGULAR

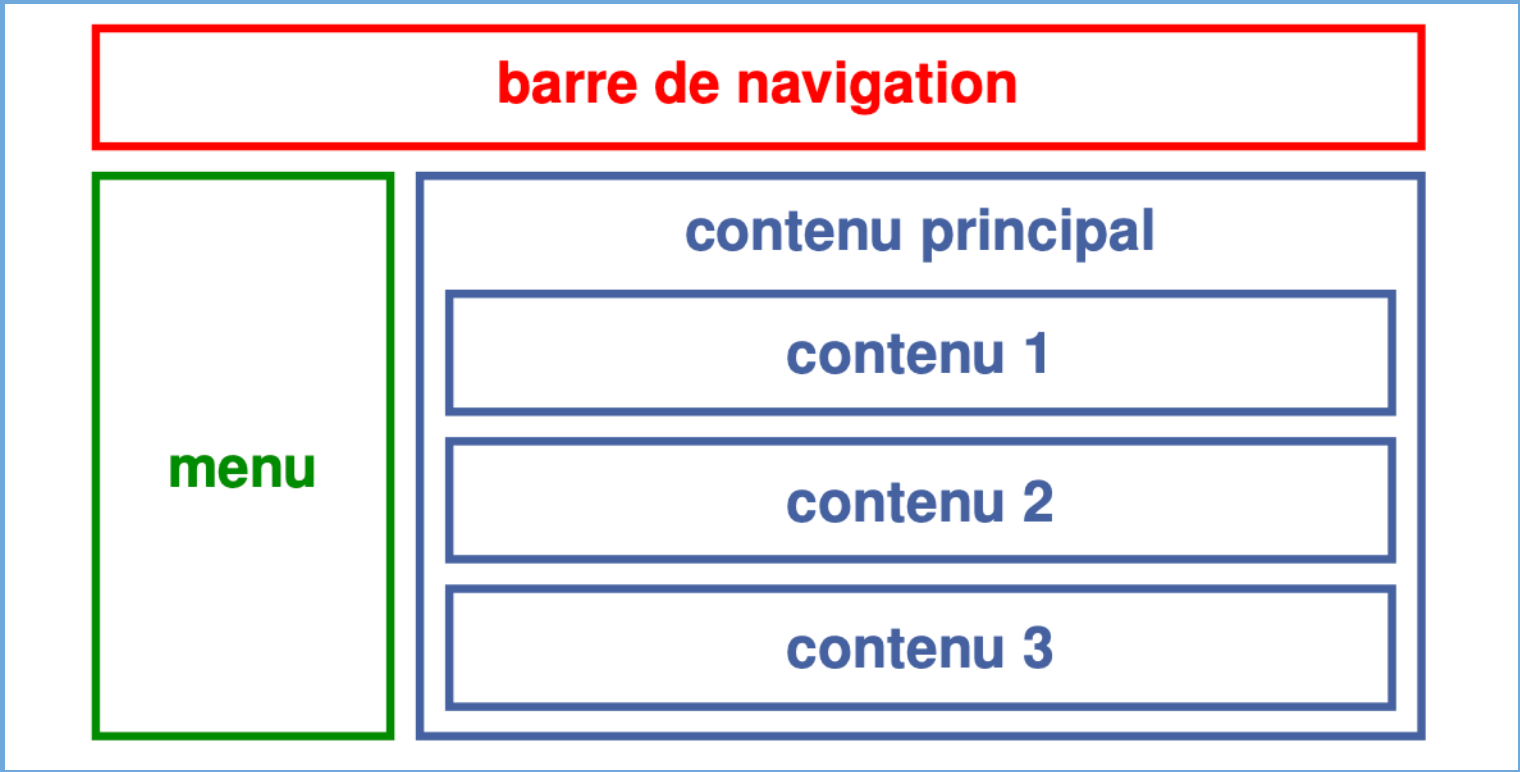
11 WEB COMPONENTS

Approche modulaire

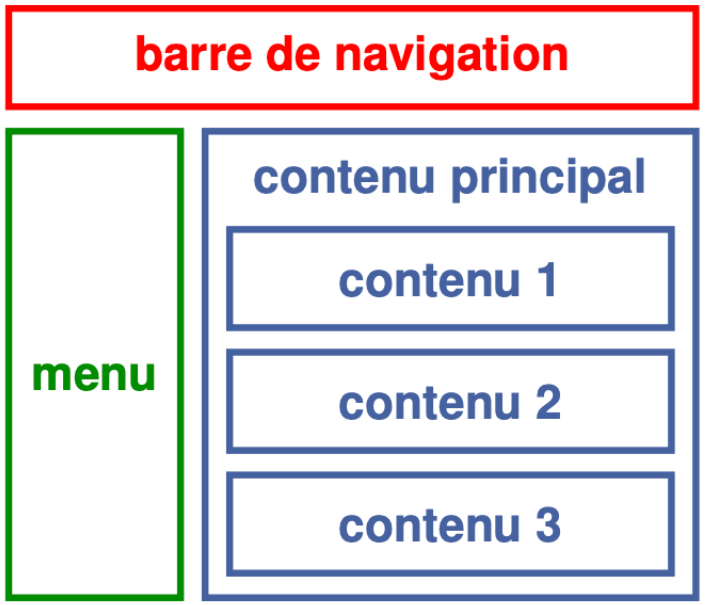
Comment découper cette page ?

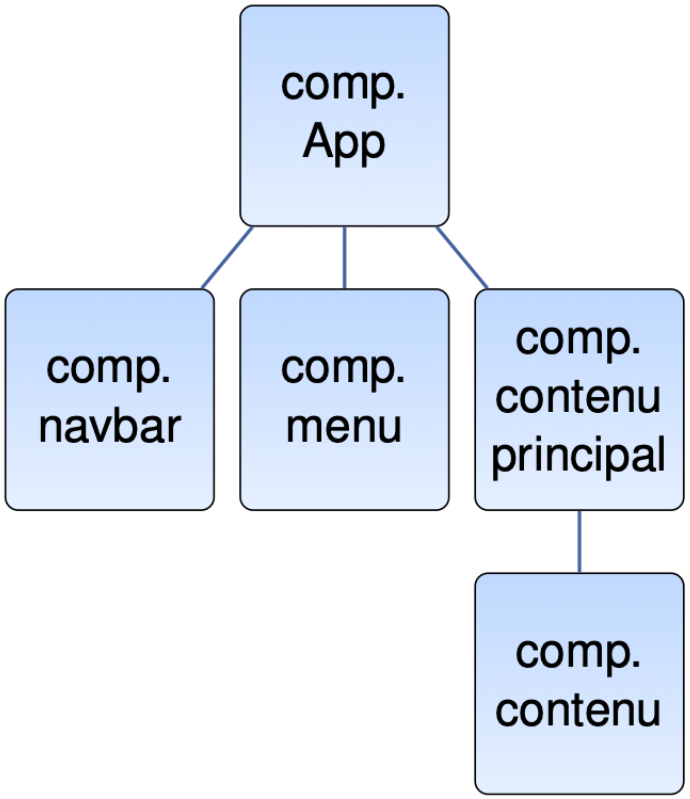
12


composant1
Composant racine
composant3
composant2
composant31
composant32

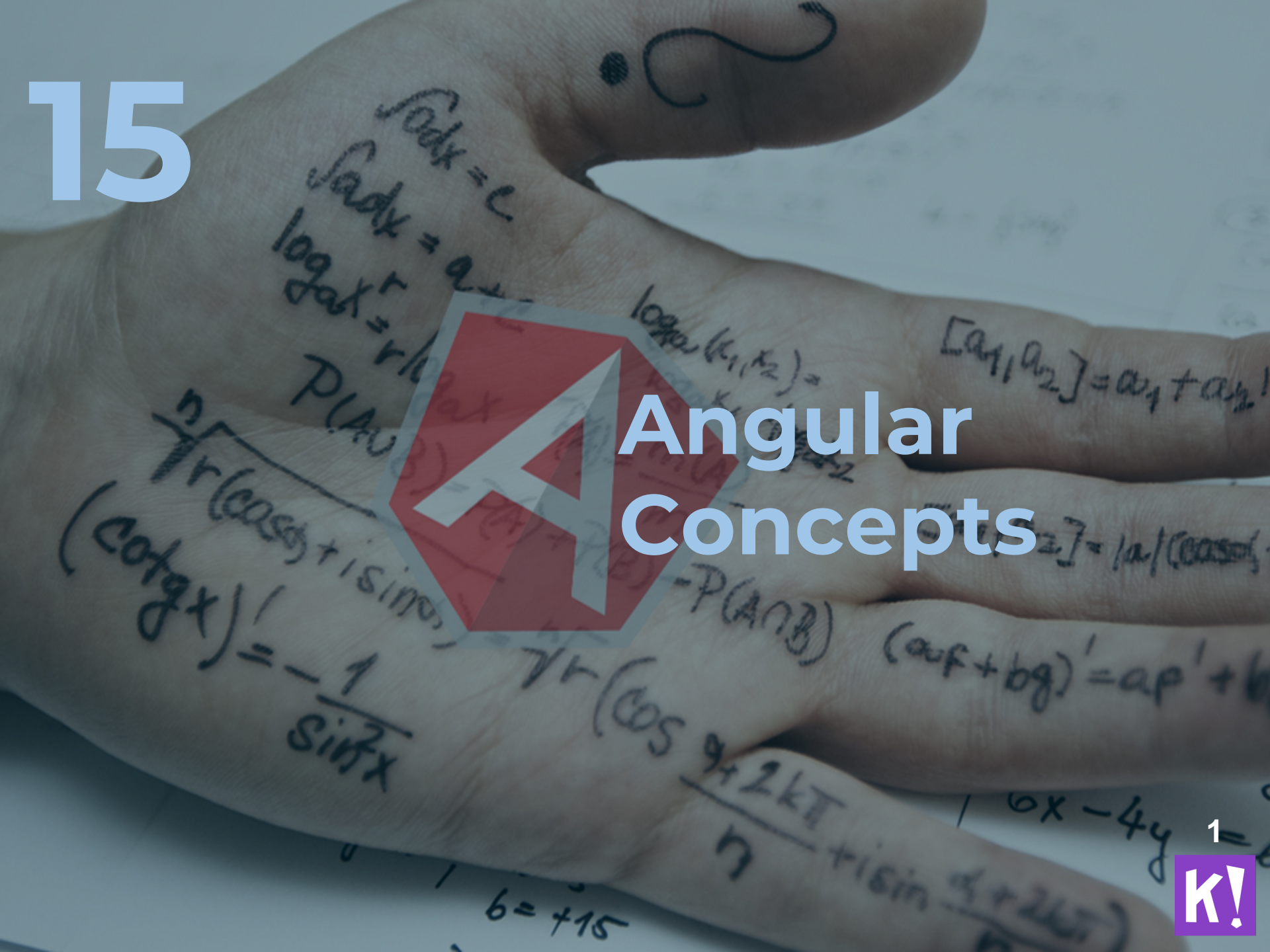


- Affichage ==⇒ structure
- Chaque rectangle = composant Angular
- Intérêt des composants : réutilisables plusieurs fois
- Un composant peut en inclure d'autres





permet de structurer facilement le code !!!


15
Angular
Concepts
1

16 ANGULAR CONCEPTS

COMPONENT

Template

Component Class

Component Test

Component style

17 ANGULAR CONCEPTS

COMPONENT

```ts
import { Component, OnInit } from "@angular/core";

@Component({
  selector: "app-home",
  templateUrl: "./home.component.html",
  styleUrls: ["./home.component.css"]
})
export class HomeComponent implements OnInit {

  public title: string;
  public description: string;

  constructor() {
    this.title = "My app";
    this.description = "My first app";
  }

  ngOnInit() { }

}
```

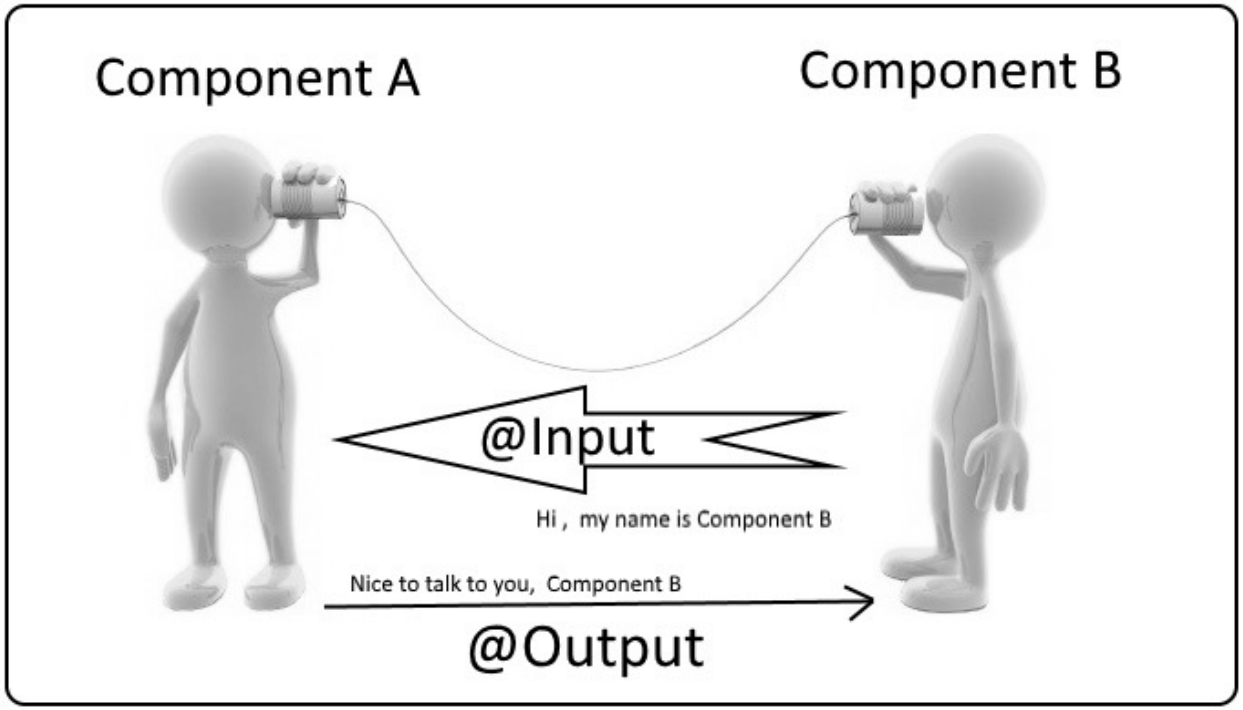
```html
<h2>{{title}}</h2>
<div class="description">
  {{description}}
</div>
```

```css
:host {
  background-color: blue;
}

h2 {
  color: white;
  font-weight: bold;
}

.description {
  color: lightgray;
}
```

18 Communication entre composants

Les méthodes dites ***view*** et ***content*** sont 2 approches différentes pour faire communiquer des composants.

La principale différence se trouve dans le partage du *scope* (la portée des variables) entre les composants parent et enfant.



19 La méthode view

- **Scope isolé** entre les composants
- **Couplage** composant parent/enfant **lourd**

Chaque **composant** gardant **son scope isolé** de l'autre, les **directives @Inputs et @Outputs** leur permettent de **s'échanger des valeurs.**

20 @Input

Input, permet de spécifier une valeur en entrée du composant.
Pour l'utiliser, vous allez devoir **modifier** le code à **2 endroits** :

- Déclarez-le avec **l'annotation @Input()** dans le composant (fichier ts)
- Liez-le en **“one-way”** binding avec [myInputVarNameHere] dans le template html **du parent**

```ts
@Component({
    selector: 'app-todo',
})
export class TodoComponent {
    @Input()
    todo: any;
    ...
}
```

todo.component.ts (enfant)

```html
<app-todo [todo]="todo"></app-todo>
```

todo-list.component.html (parent)

21

@Output

@Output, permet de spécifier un événement en sortie du composant. Pour l'utiliser, vous devez **modifier le code à 3 endroits** :

- **Instanciez** un attribut de type **EventEmitter** dans le composant (ts) qu'on décore avec la directive @Output
- **Liez-le** en **event-binding** avec (myOutputVariableName) dans le template HTML du parent
- **Faîtes** appel à la **fonction emit()** de **l'EventEmitter** au moment ou l'output doit se produire

```
@Component({
    selector: 'app-todo'
})
export class TodoComponent {
    @Output()
    notify: EventEmitter<any> = new EventEmitter();
    ...
}
```

todo.component.ts (enfant)

```
<app-todo (notify)="displayChecked($event)"></app-todo>
```

todo-list.component.html (parent)

22 La méthode content

- **pas d'input/output**
- **scope partagé entre les composants**
- **plus souple, plus facilement réutilisable**

La balise ng-content

Pour mettre en place la projection de contenu de la méthode content, vous allez devoir modifier le code à 2 endroits :

- Ajoutez du HTML à l'intérieur des balises de l'enfant, dans le fichier HTML du parent
- Récupérez-le dans le composant enfant, en projetant son contenu dans une balise <ng-content>

23

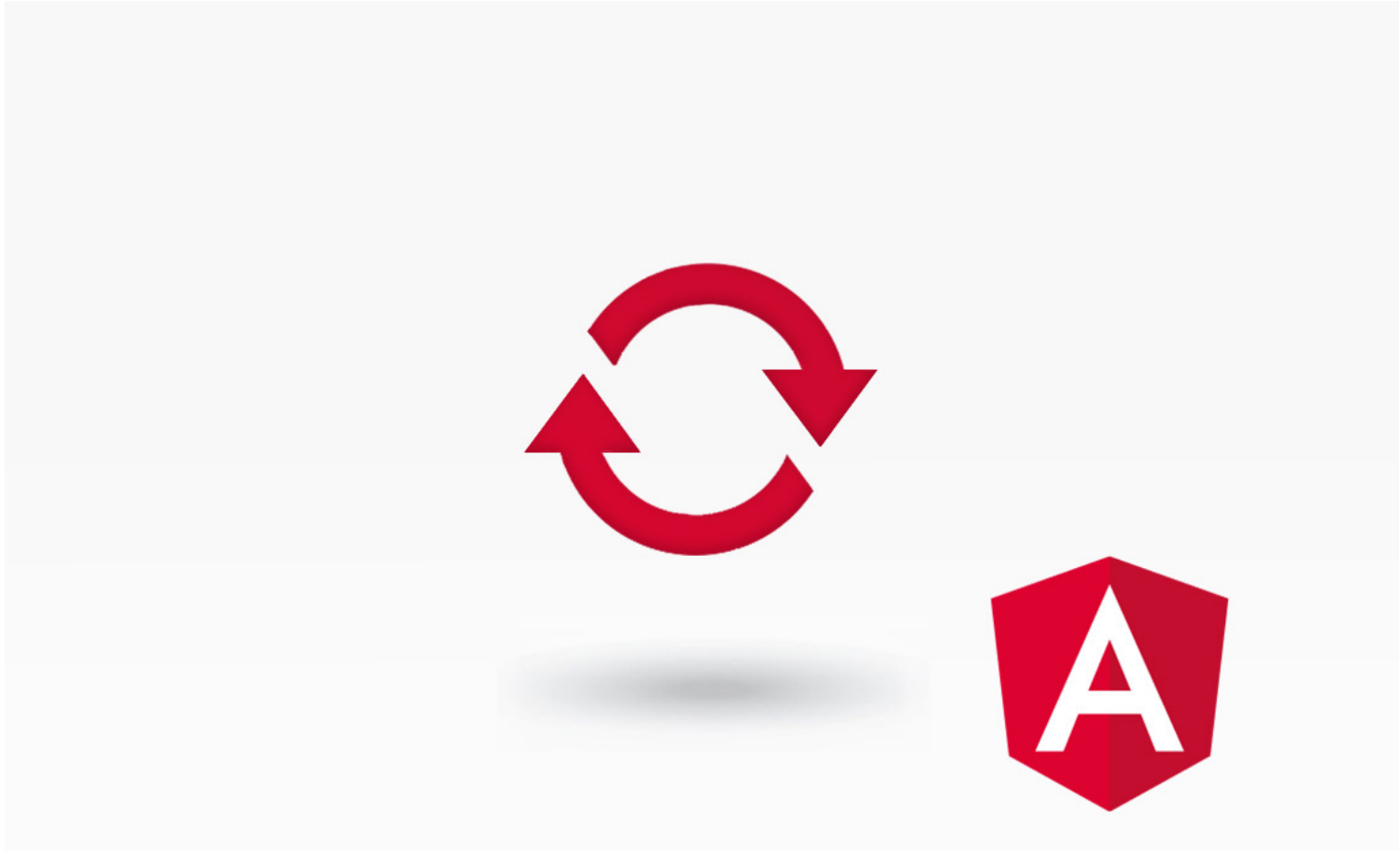
```
@Component({
    selector: 'fa-input',
    template: `
            <i class="fa" [ngClass]="classes"></i>
            <ng-content></ng-content>
`
})
export class FaInputComponent {
}
```

fa-input.component.ts (composant enfant)

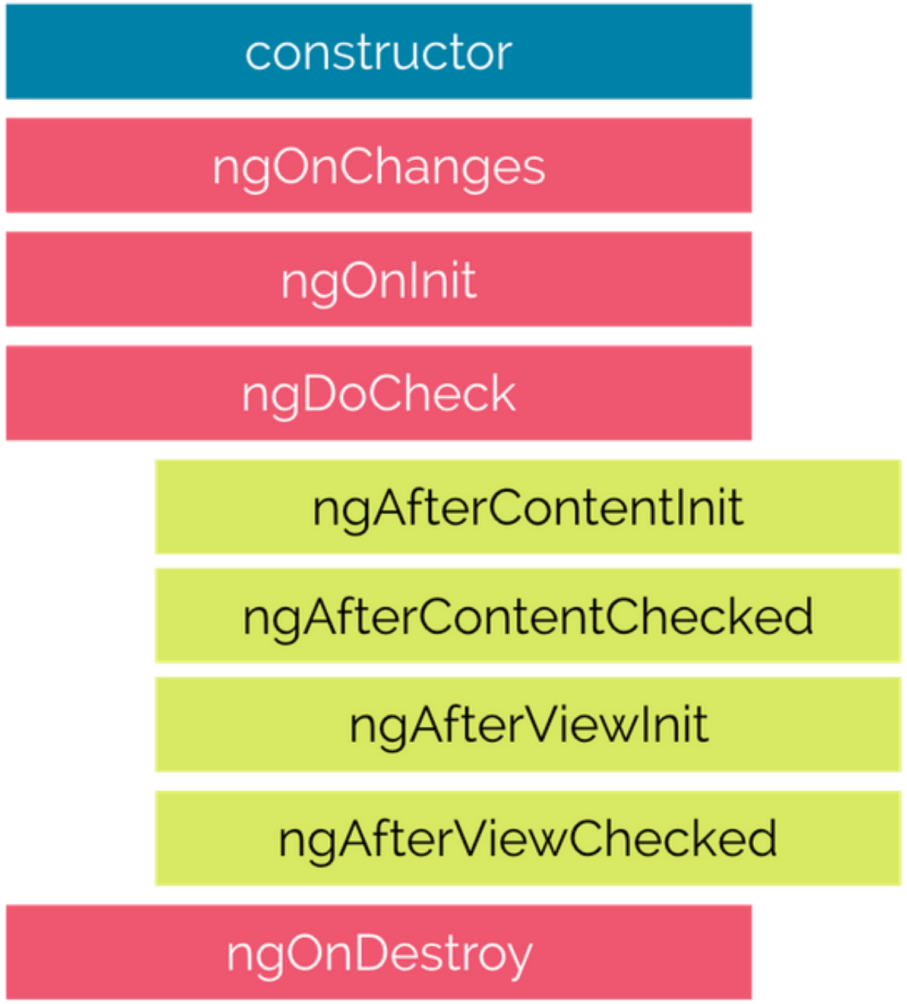
```
<h1>FA Input</h1>
<fa-input>
    <input type=”email” placeholder=”Email”/>
</fa-input>
```

app.component.html (parent)

24 Cycles de vie d’un composant



25


constructor
ngOnChanges
ngOnInit
ngDoCheck
ngAfterContentInit
ngAfterContentChecked
ngAfterViewInit
ngAfterViewChecked
ngOnDestroy

26 Constructor

```
constructor() {
    this.title = "Constructor";
    this.details = "Life cycle component";
}
```

Attention : Ainsi, ne placez **aucune initialisation complexe dans le constructeur.** Vous voulez des composant qui se construisent sans consommer beaucoup de ressources et de manière sûre.

27 NgOnChanges : - Détecter les changements d'Input

```
1  @Input() lastname: string;
2
3  constructor() {}
4
5  ngOnChanges() {
6    this.uppercase(this.lastname);
7  }
8
9  private uppercase(input: string) {
10   input = input.toUpperCase();
11 }
```

Immédiatement après le constructeur, Angular appelle le hook ngOnChanges().
Il est également appelé après chaque changement d'une des propriétés passées en Input.

28 NgOnInit - Initialiser le composant

```
1       studentList: Array<any>;
2
3       constructor(private studentService: StudentService){}
4
5       ngOnInit(){
6           this.getStudent();
7       }
8
9       getStudent(){
10          this.studentService
11              .getStudentsFromService()
12              .subscribe(students =>  this.studentList = students);
13      }
```

Le hook NgOnInit est déclenché dès que le DOM du composant a fini de charger. Utilisez la méthode ngOnInit() dans les 2 cas suivants :

- **Vous voulez mettre en place une initialisation complexe** (donc hors du constructeur), typiquement appeler une API qui charge les données de votre page.

- **Vous devez modifier l'état de votre component après le chargement des attributs passés en @Input**

29 NgDoCheck - Réagir à tous les changements du DOM

Vous souhaitez réagir à un changement du DOM qui est indétectable par vos autres hooks ? Dans ce cas vous devrez utiliser la méthode ngDoCheck()

Cette méthode vous permet de réagir à tout changement dans la page qui ne serait pas détecté par le framework.

Attention ! Ne l'utilisez pas plus que nécessaire : cette méthode coûte énormément de ressources et peut donc diminuer les performances globales de votre application.

De ce fait, vous ne verrez pas beaucoup ce hook !


QUESTIONS?