

# Web & Mobile UI

## Conception d'applications web

Loris Gavillet, HEIG-VD, 02.2022, v1.3

# Enseignants

## Cours

### Phase 1

**Loris Gavillet**

[loris.gavillet@heig-vd.ch](mailto:loris.gavillet@heig-vd.ch)

Semaine 1 -> Semaine 6

### Phase 2

**Nicolas Chablotz**

[nicolas.chablotz@heig-vd.ch](mailto:nicolas.chablotz@heig-vd.ch)

Semaine 7 -> Semaine 12

# Objectifs

## Cours

- Être capable de réaliser des applications Web progressive (Progressive Web App) en intégrant un design préalablement conçu
- Manipuler différentes API's pour la réalisation d'interfaces web modernes
- Mettre en place des patrons de conception (design patterns)

# Contenu

## Cours

- De HTML5 vers un standard HTML vivant
- Responsive design, Media Queries
- Data-attributes, custom elements
- API HTML: LocalStorage, History, Service Workers, ...
- Offline mode
- Début templating

# Déroulement

## Cours

### Phase 1 (Semaine 1-6)

- Introduction au cours
- 1/3 Théorie - 2/3 Pratique
- Live coding - Corrections entre les cours
- Setup des outils de base
- Projet sur 9 “jours” de cours
- Examen sur 4 périodes

# Technologies utilisées

## Cours

### Phase 1

- Webpack
- HTML / CSS / JS
- API HTML
- Templating
- PWA / Service workers
- Intro aux frameworks

# Repository et informations générales

## Cours

**<https://github.com/lgavillet/webmobui-22>**

# Projet

# Concept Projet



# Spotlified

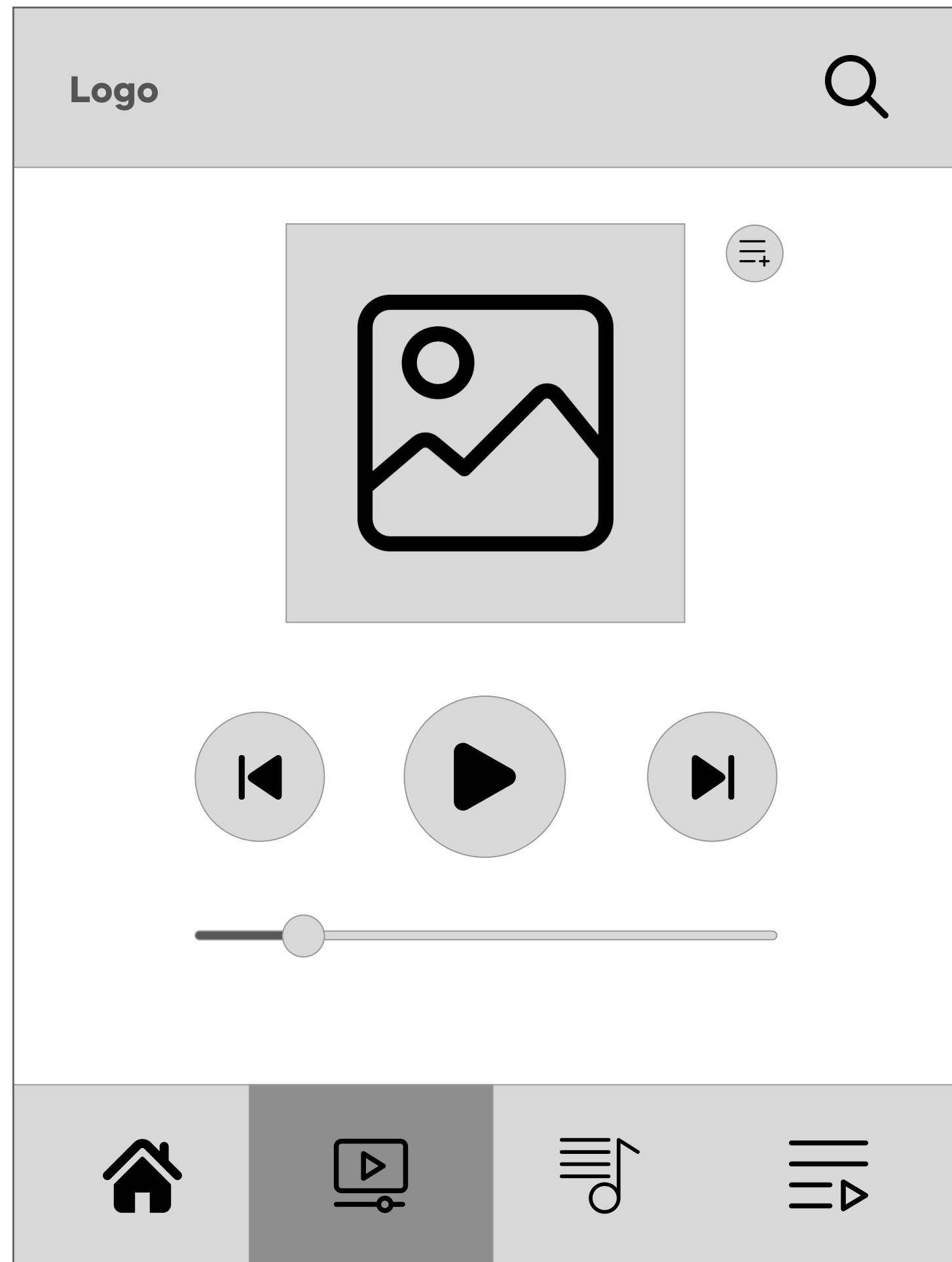
*Simplified spotify*

# Concept Projet

**Spotify version simplifiée**

3 modèles :

- Chansons
- Artistes
- Playlists



# Features

## Projet

- Homepage
- Liste des artistes
- Liste des chansons par artiste
- Lecteur audio
- Champ de recherche
- Gestion online/offline
- Gestion des playlists
- Caching
- Installation de l'application

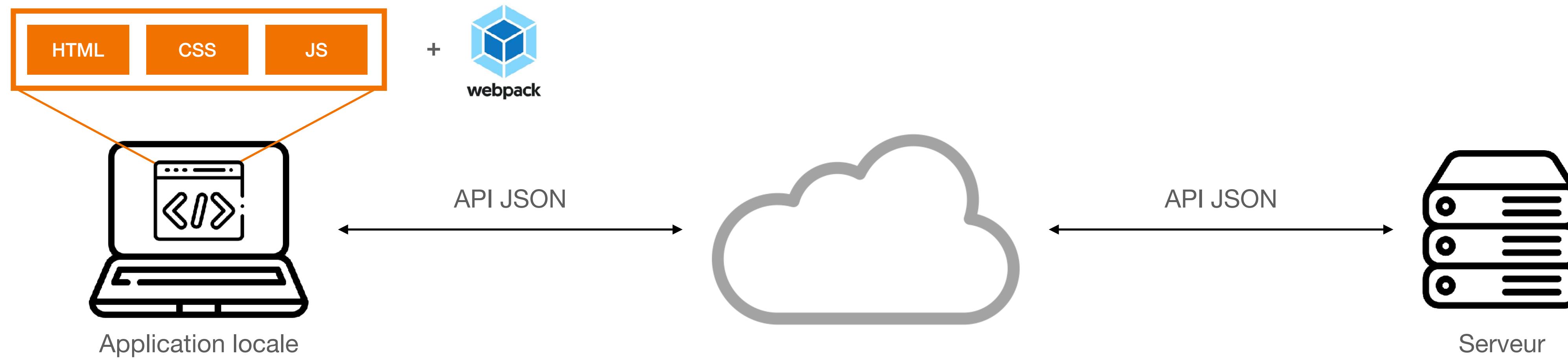
# Prototype

## Projet

<https://invis.io/DG12CQJQ9QPF#/463974083> Home

# Architecture

## Projet



# Composants ?

## Projet

- Projet webpack vide
- Squelette HTML
- Styles CSS structurels
- Icônes
- Routeur pour les pages web
- Client pour l'API JSON
- Lecteur audio
- Popover pour playlists
- Détection online/offline
- Local storage pour les playlists
- Manifest PWA
- Caching
- Service worker

# Packages

# What is it?

## Packages



# Définition

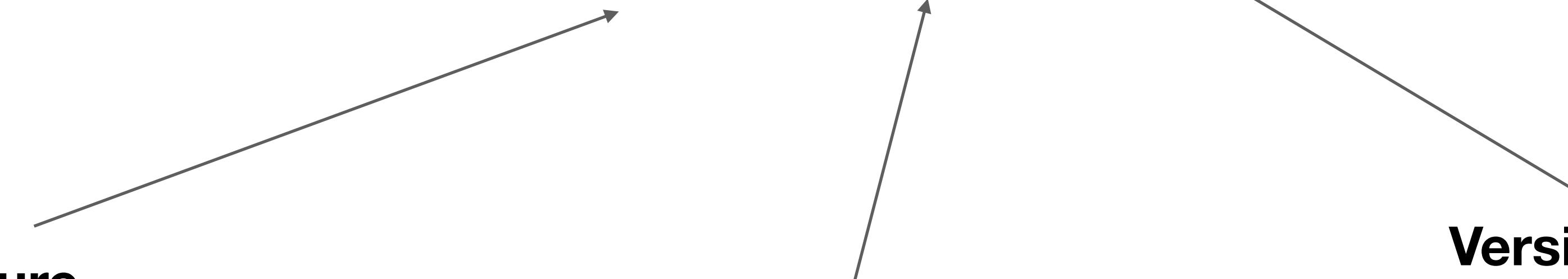
## Package

- Bout de code réutilisable
- Expose des fonctionnalités
- Intégrable dans une application
- Versionable

# Versioning - Semantic versioning

## Package

v16.2.6



### Version majeure

Le numéro de version MAJEUR quand il y a des changements non rétrocompatibles,

### Version mineure

Le numéro de version MINEUR quand il y a des ajouts de fonctionnalités rétrocompatibles

### Version patch

Le numéro de version de CORRECTIF quand il y a des corrections d'anomalies rétrocompatibles

# Choix d'un package

## Package

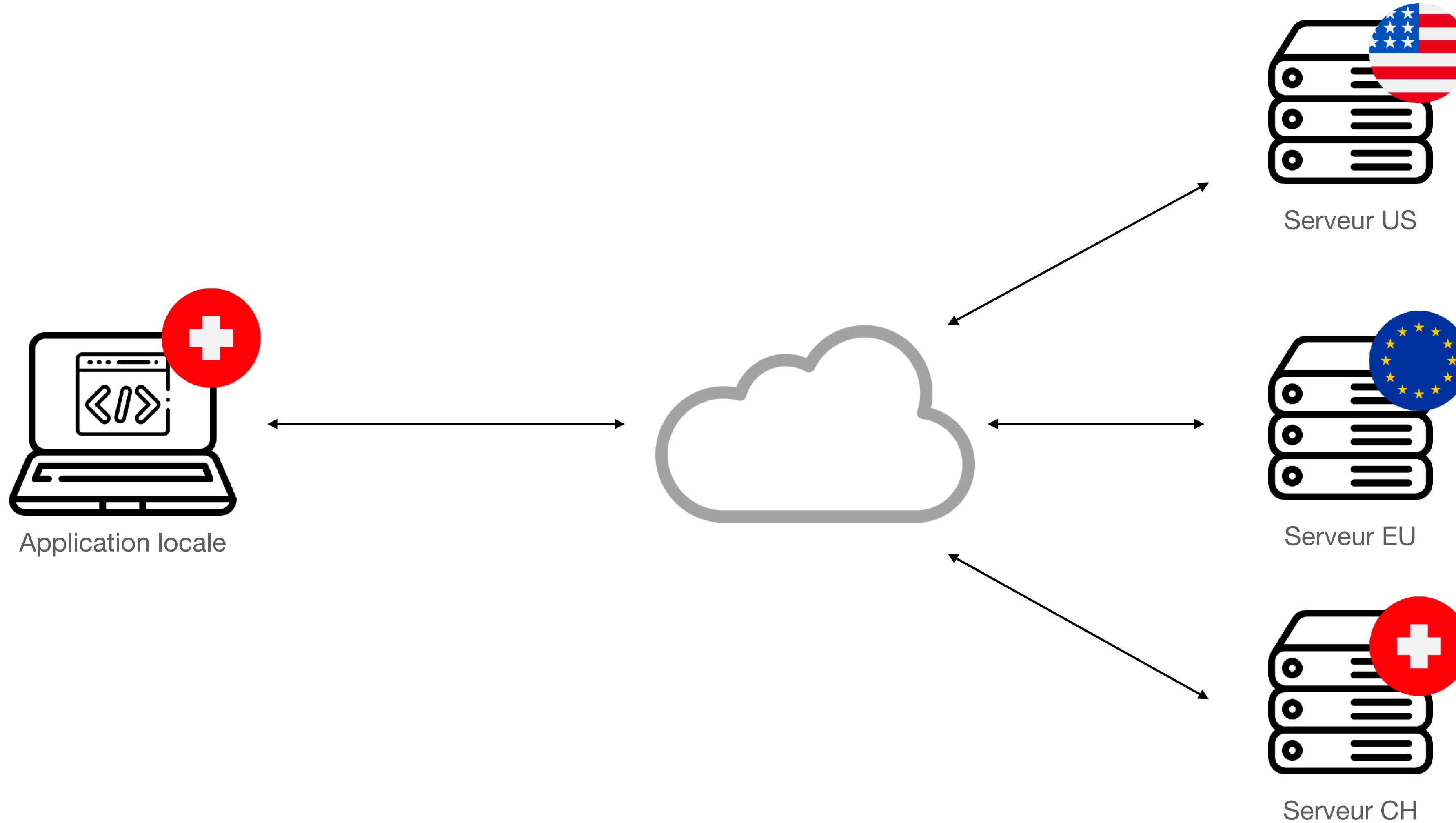
- Vérifier la version -> v0.x.x -> à bannir
- Nombre d'utilisateurs ?
- Maintenu ? Date dernier patch ?
- Dépendances ?

# Intégration Package

1. Copy-paste old school / Téléchargement
2. CDN - Content Delivery Network
3. Package manager

# CDN - Content Delivery Network

## Package



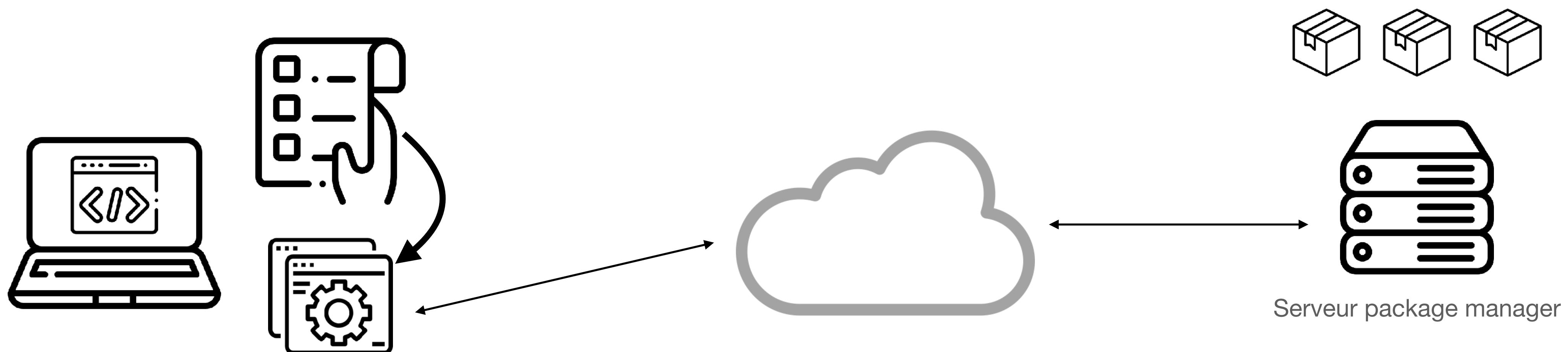
# CDN - Content Delivery Network

## Package

```
<link  
    rel="stylesheet"  
    href="https://fonts.googleapis.com/css2?family=Material+Icons"  
/>
```

# Package manager

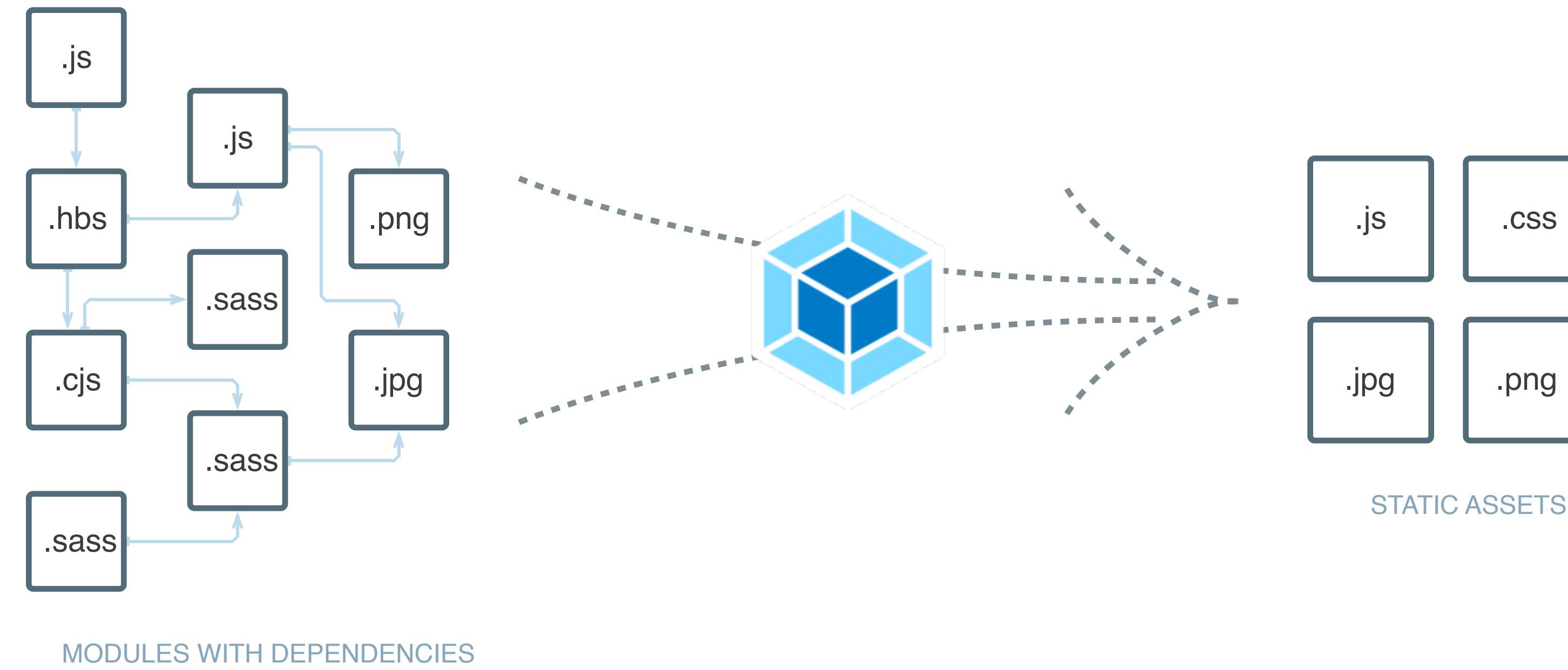
## Package



# Webpack

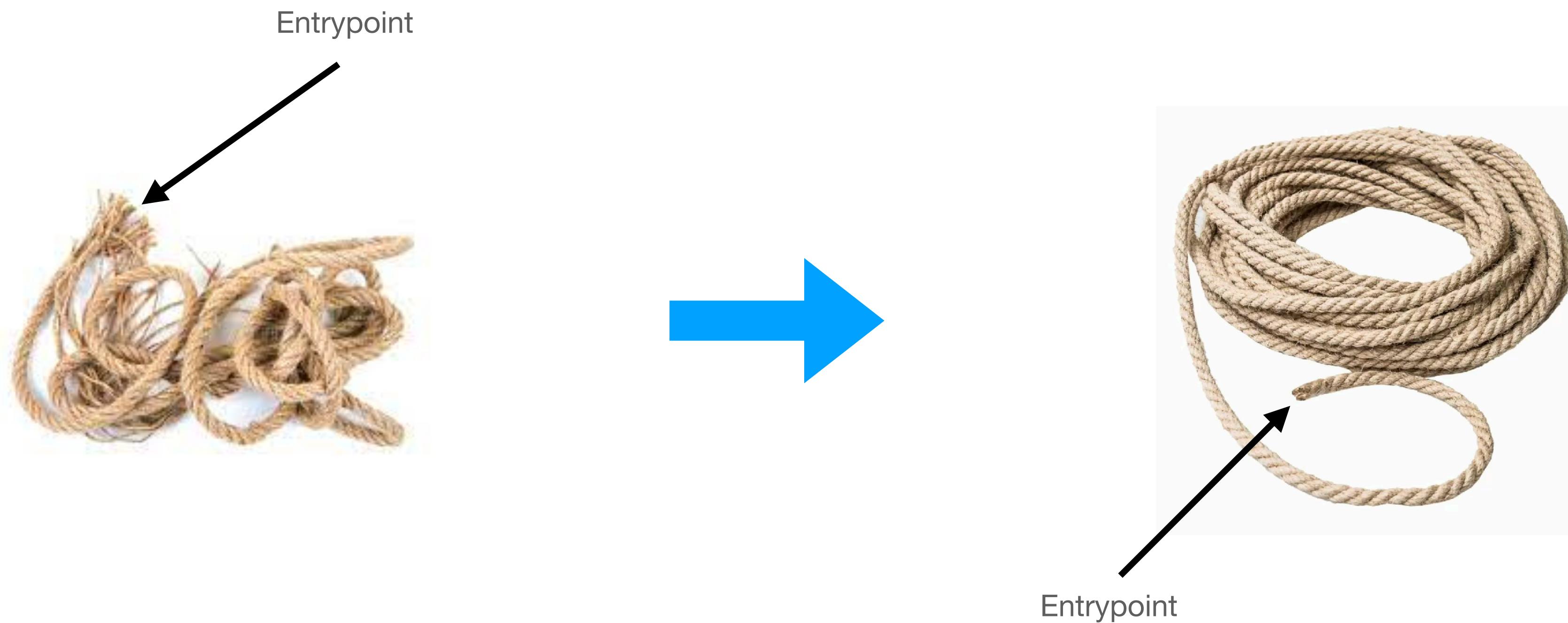
# Concept

## Webpack



# Concept

## Webpack



# Concept

## Webpack

*“webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or asset.”*

# Concept

## Webpack

Basé sur



- NodeJS est un langage de programmation backend, écrit en Javascript
- A ne pas confondre avec le Javascript frontend
- Node est livré avec un gestionnaire de package :  
**NPM (Node Package Manager)**

# Concept

## Webpack

Basé sur



- Webpack est un package de NodeJS
- S'installe comme tout autre package avec  
`> npm install webpack`

# Concept

## Webpack

- Webpack est un package de NodeJS
- S'installe comme tout autre package avec

```
> npm install webpack
```

# Code

# Pour le cours

## Webpack

- Uniquement packaging en javascript et css
- Serveur web avec Hot reloading
- Pré-configuré et prêt à l'emploi

<https://webpack.js.org/>

# Structure de base officielle

## Webpack

- ▶ /
  - ▶ dist/
    - bundle.js
  - ▶ src/
    - ▶ ...
    - index.js
    - index.html
    - package.json
    - package-lock.json
    - webpack.config.js

<https://webpack.js.org/>

# Structure pour le cours

## Webpack



# Installation

## Webpack

### Installation

- Installer Node (si pas déjà fait)  
<https://nodejs.org/en/download/> Version LTS
- Créer un dossier pour le code et copier le projet vide :  
[https://github.com/lgavillet/webmobui-22/tree/master/  
Projet vide](https://github.com/lgavillet/webmobui-22/tree/master/Projet%20vide)
- Ouvrir l'invite de commande dans votre dossier projet  
`> npm install`

<https://webpack.js.org/>

# Utilisation Webpack

Démarrer le serveur de dev

```
> npm run start
```

<https://webpack.js.org/>

# Node installé ?

## Webpack

```
node --version
```

**Go!**

# HTML / CSS / JS

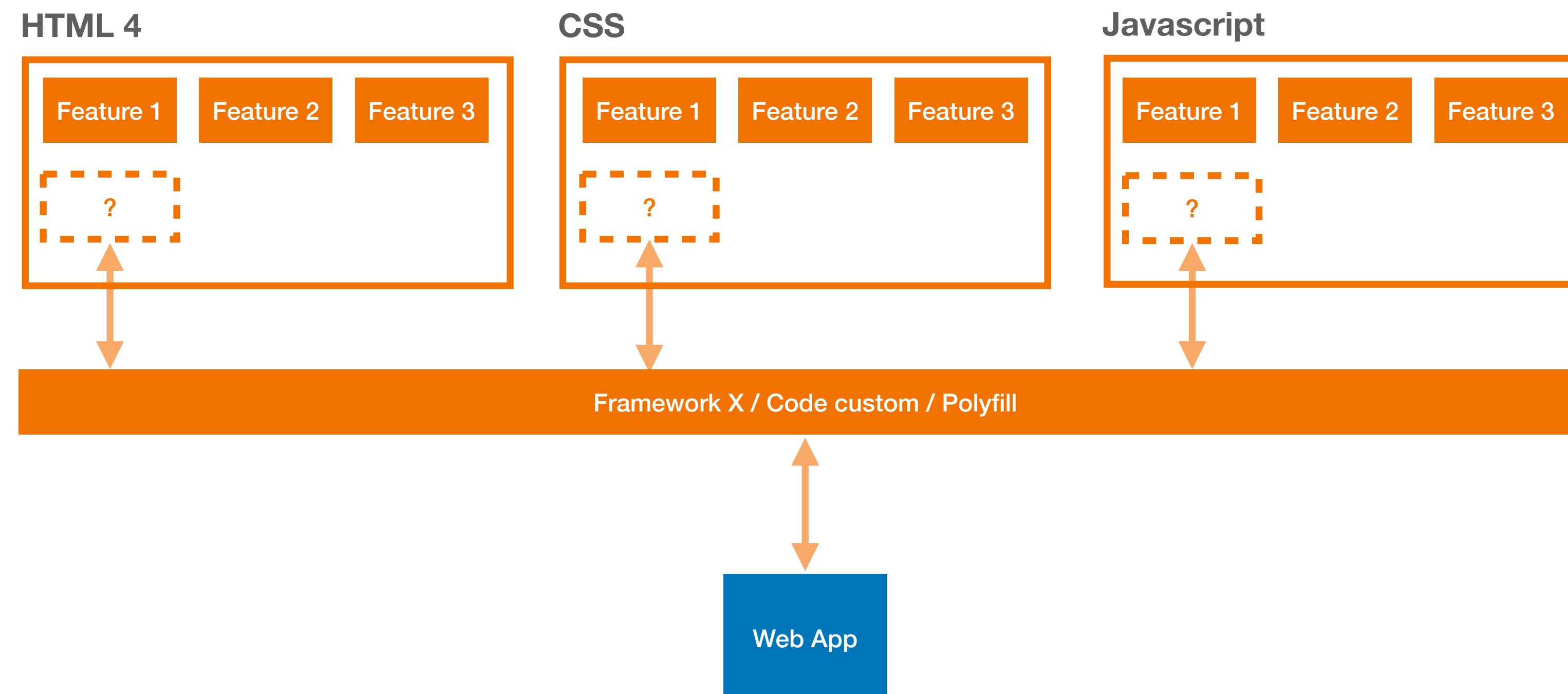
# Nouveau standard HTML

HTML / CSS / JS

*“Vers une fin des dépendances et une collaboration inter-langages”*

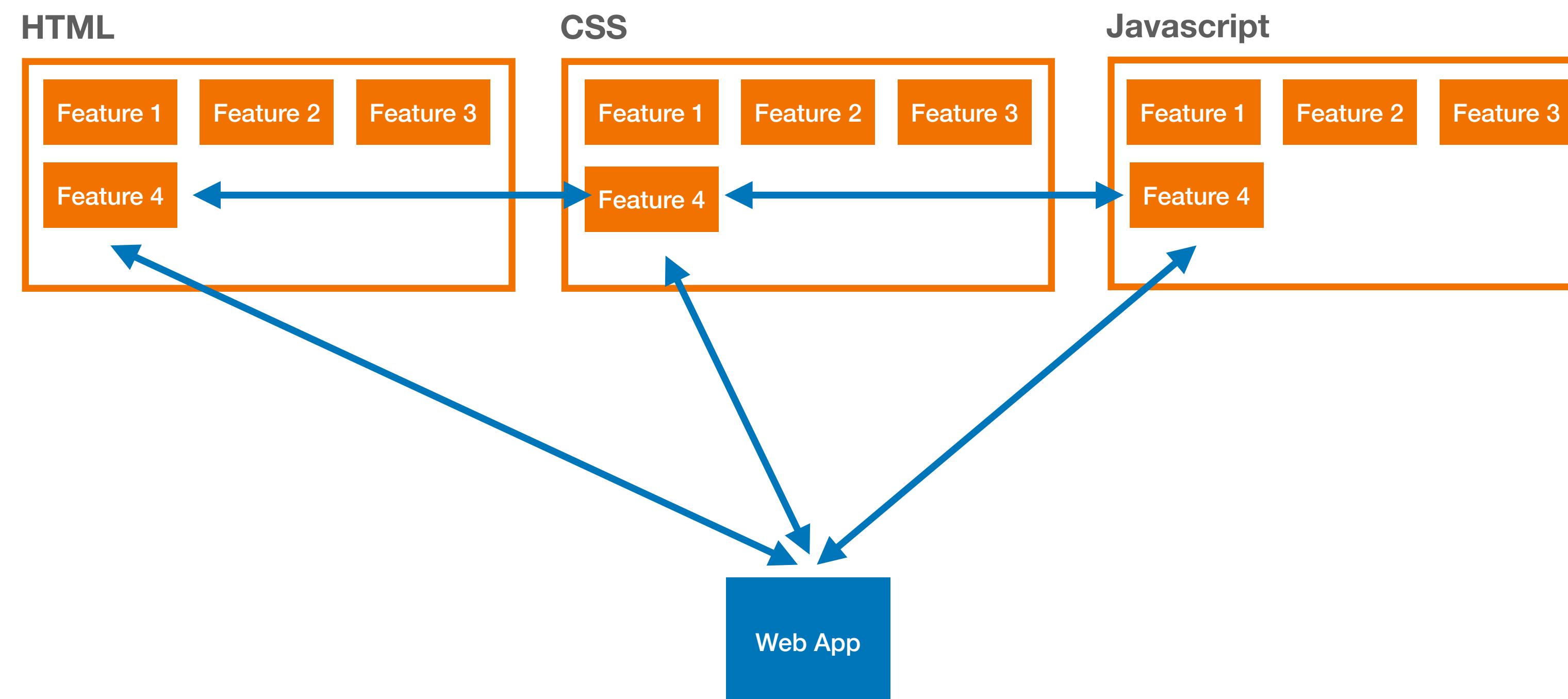
# Standardisations des use cases courants

## HTML / CSS / JS



# Standardisations des use cases courants

## HTML / CSS / JS



# Polyfill

## HTML / CSS / JS

- Un “Polyfill” est le terme générique donné à un bout de code permettant de combler une feature manquante dans un browser
- Permet la retrocompatibilité
- Le plus connu : Babel Polyfill

# Standardisations des use cases courants

## HTML / CSS / JS

- Validations de formulaires
- Lecteur audio/vidéo
- Eléments complexes (progress bar, ...) -> Shadow DOM !
- Sélectionner des éléments impaires
- Etc...

# Nouveau standard HTML

## HTML / CSS / JS

- Amélioration de HTML 4
- Fortement lié à Javascript
- Nouveaux éléments sémantiques (header, main, section, article, ...)
- Form inputs améliorés (types d'inputs, claviers mobiles, validations, transformations)
- APIs (géolocalisation, local storage, history, push, service workers, ...)
- Distribution automatique des éléments
- Custom elements

# **W3C vs WHATWG**

## **HTML / CSS / JS**

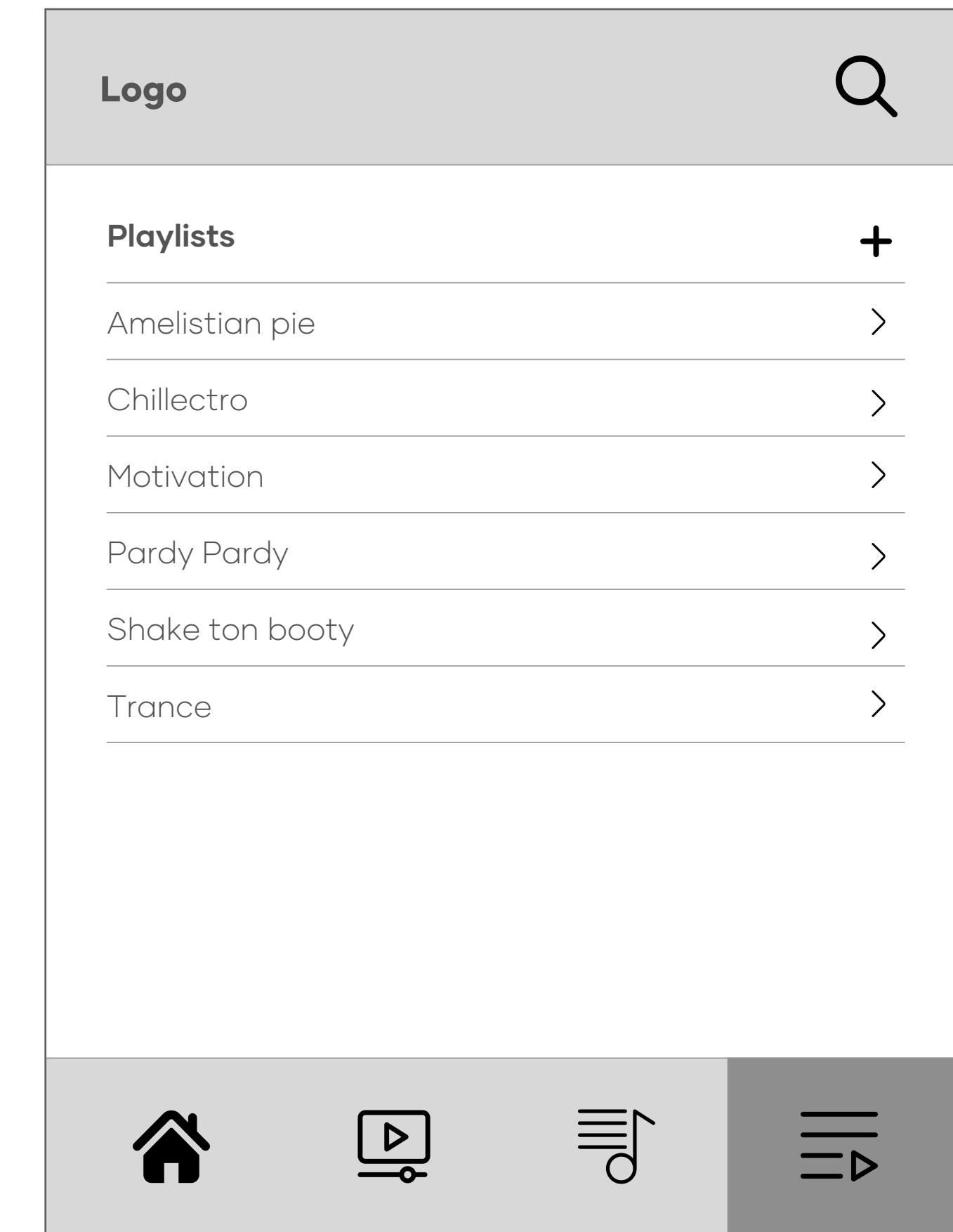
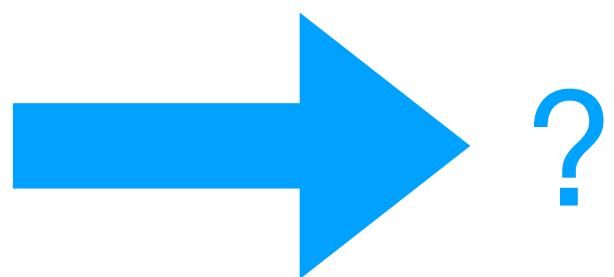
- W3C (**World Wide Web Consortium**)  
Acteur historique...
- WHATWG (**Web Hypertext Application Technology Working Group**)  
Collaboration non officielle des différents navigateurs web

# HTML

# Eléments sémantiques

## HTML

```
<header />  
<nav />  
<main />  
<section />  
<footer />  
<article />
```



# Custom elements

## HTML

*Here's the magic !*



```
<my-super-custom-element />
```

# Extended custom elements

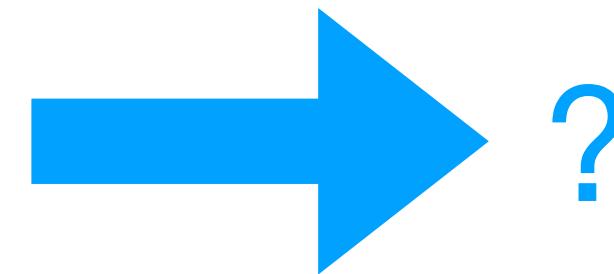
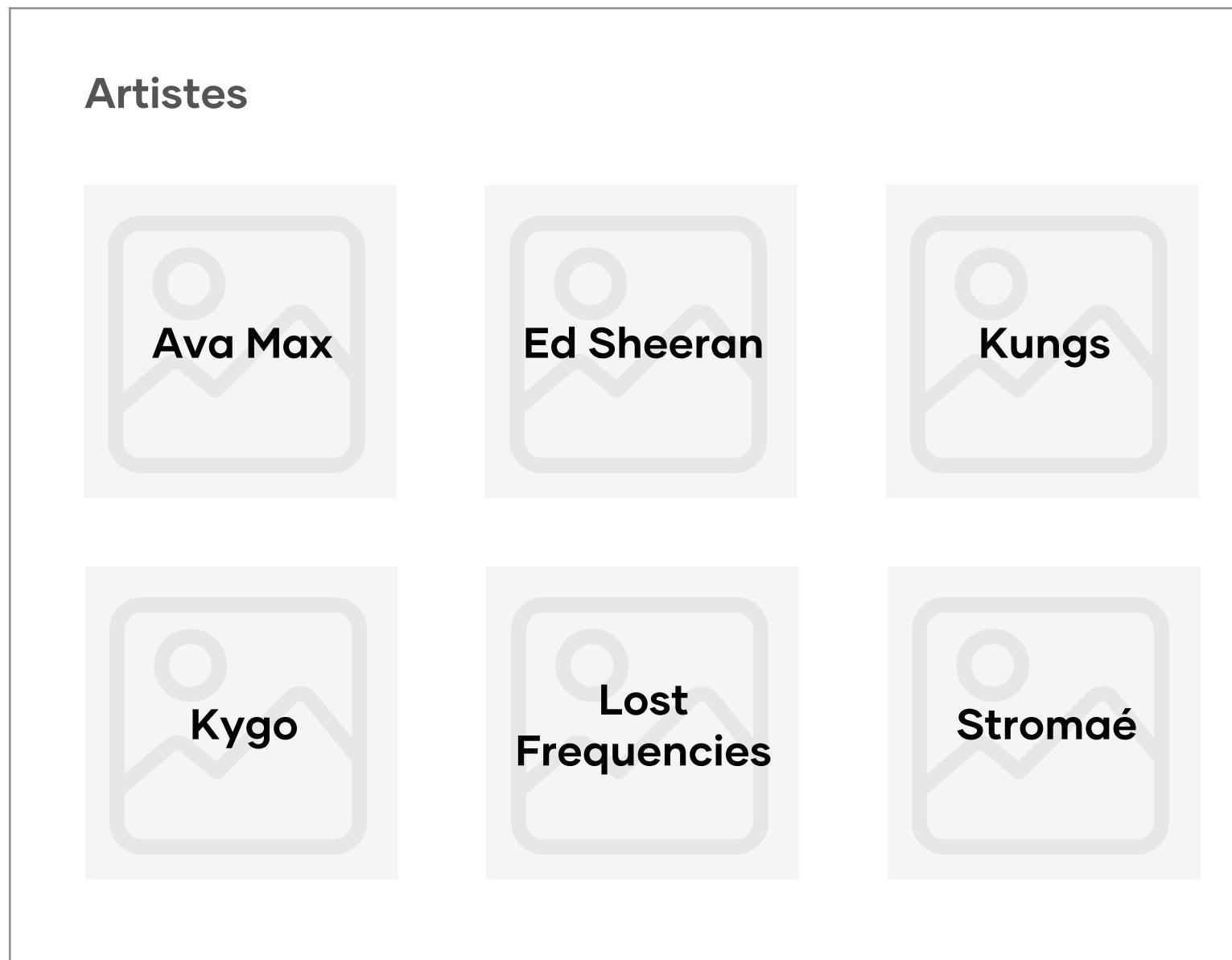
## HTML

```
<mega-button />
```

```
const MegaButton = document.registerElement('mega-button', {  
  prototype: Object.create(HTMLElement.prototype),  
  extends: 'button'  
});
```

# Custom elements

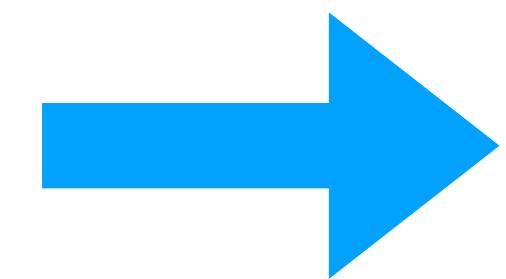
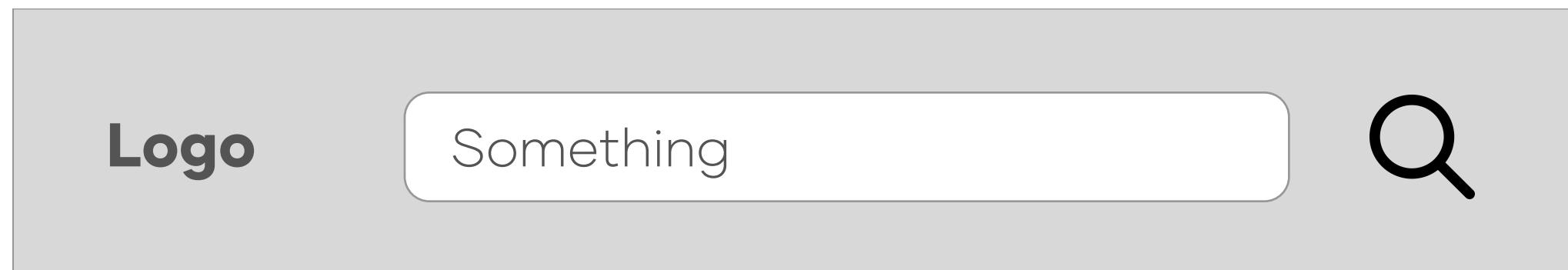
## HTML



<artist-cover>  
...  
</artist-cover>

# Inputs on steroids

HTML

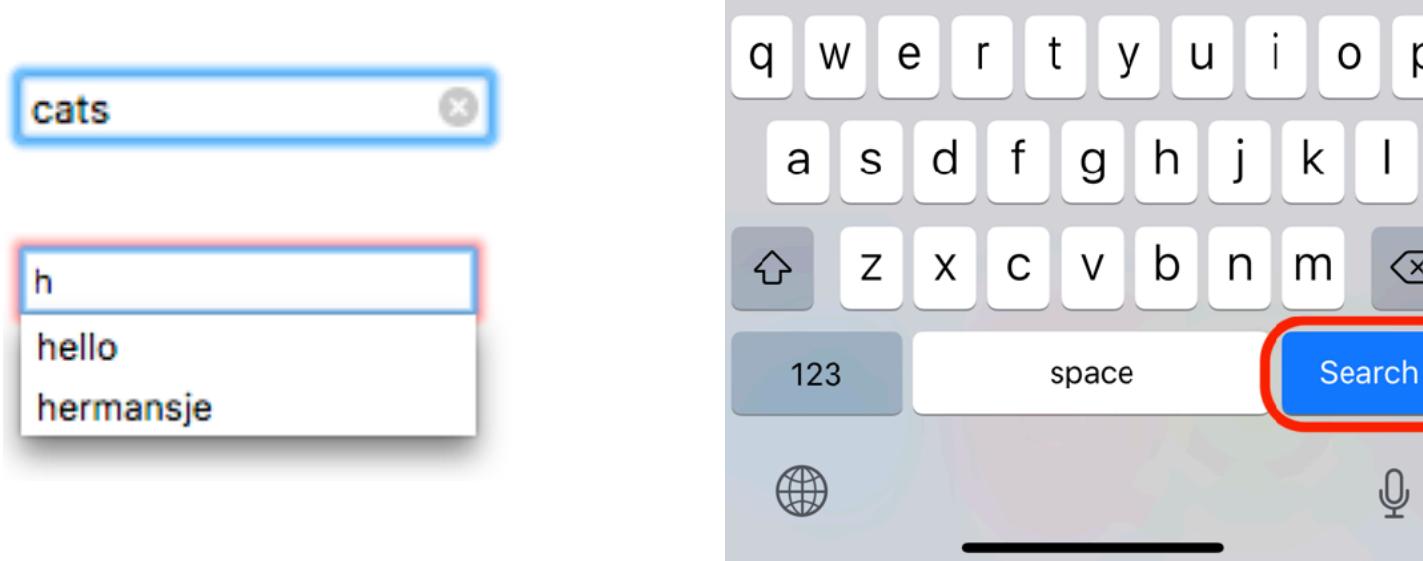


?

```
<input type="search" />
```

# Inputs on steroids

## HTML



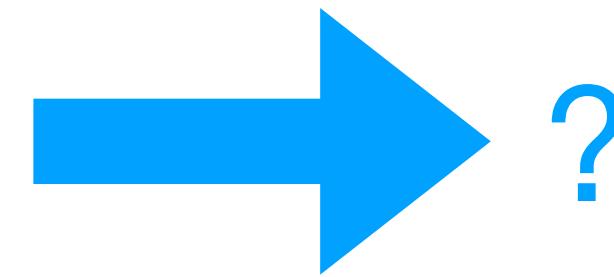
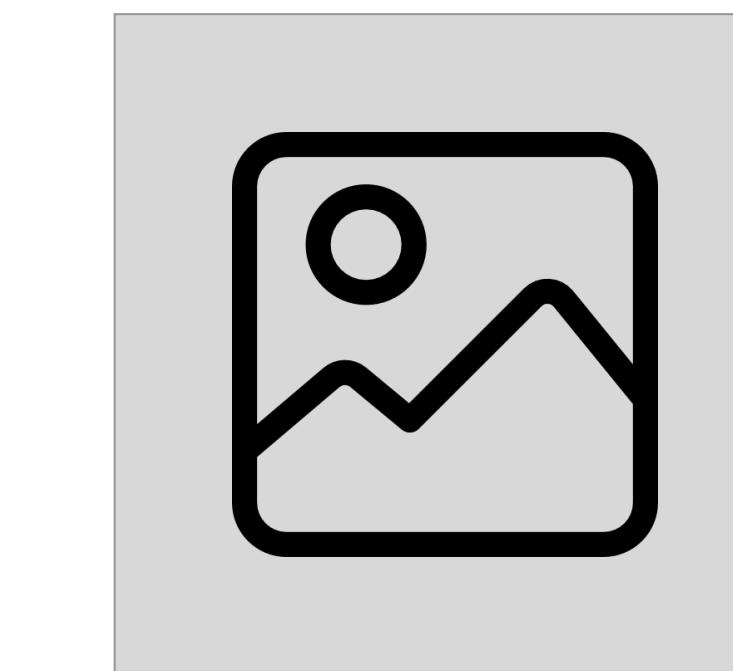
```
<input  
  type="search"  
  spellcheck="false"  
  autocapitalize="false"  
  autofocus  
/>
```



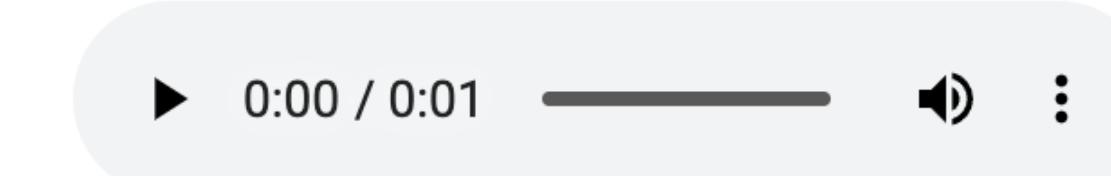
autofocus et autofocus="true" sont sémantiquement équivalents

# Audio element

HTML



? <**audio** **src**=“<https://...>” />



# Audio element

HTML

*Presque...*

```
<audio  
  id="audio-player"  
  src="https://..."  
/>
```

```
#audio-player {  
  display: none;  
}
```

# Shadow DOM

## HTML and a bit more...

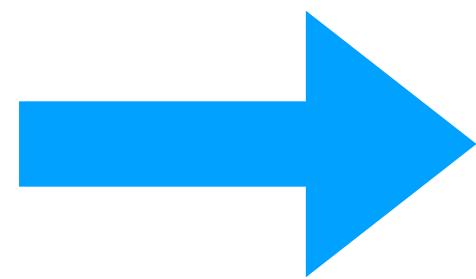
- “Black box”
- Encapsulation et abstraction du DOM contenu dans un élément
- Principalement utilisé par les browsers pour abstraire des éléments complexes
- Exemple: <**audio** />



# Shadow DOM

## HTML and a bit more...

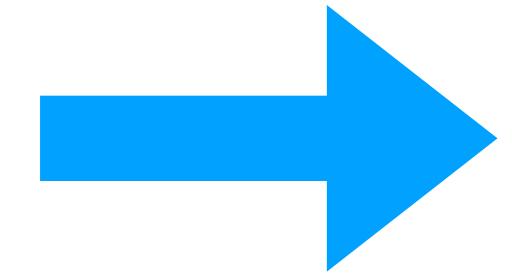
```
<!DOCTYPE html>
<html>
  ><head>...</head>
  ><body>
    ..  <audio src="http://...."></audio> == $0
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
  ><head>...</head>
  ><body>
    ..  <audio src="http://...."> == $0
      >#shadow-root (user-agent)
        ><div pseudo="-webkit-media-controls" class="phase-pre-ready state-no-source">(flex)
          ><div pseudo="-webkit-media-controls-overlay-enclosure">
            ><input pseudo="--internal-media-controls-overlay-cast-button" type="button" aria-label="play on remote device" style="display: none;">
              >#shadow-root (user-agent)
            </input>
          </div>
          ><div pseudo="-webkit-media-controls-enclosure">(flex)
            ><div pseudo="-webkit-media-controls-panel" style="display: none;">
              ><input type="button" pseudo="--webkit-media-controls-play-button" aria-label="play" class="pause" disabled style="display: none;">...</input>
              ><div aria-label="elapsed time: 0:00" pseudo="--webkit-media-controls-current-time-display" style="display: none;">0:00</div>
              ><div aria-label="total time: / 0:00" pseudo="--webkit-media-controls-time-remaining-display" style="display: none;">/ 0:00</div>
              ><input type="range" step="any" pseudo="--webkit-media-controls-timeline" max="NaN" min="0" aria-label="audio time scrubber 0:00 / 0:00" aria-valuetext="elapsed time: 0:00" disabled>...</input>
              ><div pseudo="--webkit-media-controls-volume-control-container" class="closed" style="display: none;">...</div>
              ><input type="button" pseudo="--webkit-media-controls-fullscreen-button" aria-label="enter full screen" style="display: none;">...</input>
              ><input type="button" aria-label="show more media controls" title="more options" pseudo="--internal-media-controls-overflow-button" style="display: none;">...</input>
            </div>
            ><div role="menu" aria-label="Options" pseudo="--internal-media-controls-text-track-list" style="display: none;">...</div>
            ><div role="menu" aria-label="Options" pseudo="--internal-media-controls-playback-speed-list" style="display: none;">...</div>
            ><div pseudo="--internal-media-controls-overflow-menu-list" role="menu" class="closed" style="display: none;">
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="Play " class="animated-1" style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="enter full screen Full screen " style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="download media Download " style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="Mute " style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="play on remote device Cast " style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="show closed captions menu Captions " style="display: none;">...</label>
              ><label pseudo="--internal-media-controls-overflow-menu-list-item" role="menuitem" tabindex="0" aria-label="show playback speed menu Playback speed " class="animated-0" style="display: none;">...</label>
            </div>
          </div>
        </audio>
      </body>
    </html>
```

# Progress v1

## HTML

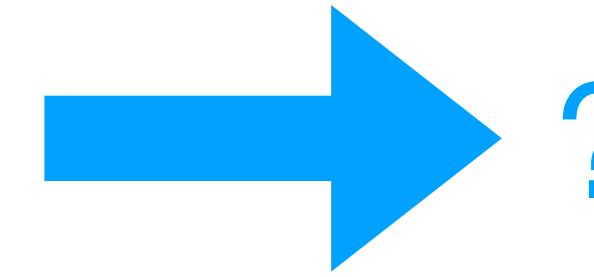


?

```
<progress value="20" max="100" />
```

# Progress v2

## HTML



```
<input  
    type="range"  
    value="20"  
    min="0"  
    max="100"  
/>
```

# Data attributes

## HTML

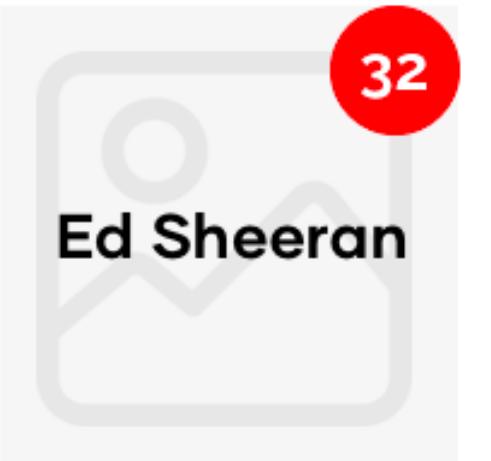
```
<artist-cover  
  data-id="1"  
  data-song-count="32"  
  data-thumbnail-url="https://..."  
>  
  Ed Sheeran  
</artist-cover>
```

# Data attributes

HTML and a bit more...

```
<artist-cover  
  data-id="1"  
  data-song-count="32"  
  data-thumbnail-url="https://...">  
  Ed Sheeran  
</artist-cover>
```

```
artist-cover {  
  /*_/_!\ Bientôt..._*/  
  background-image: attr(data-image url);  
}  
  
artist-cover:before {  
  /*_/_Possible! Content only _*/  
  content: attr(data-song-count);  
  position: absolute;  
  top: 0;  
  right: 0;  
  border-radius: 50%;  
  background-color: #ff0000;  
  color: #fff;  
  text-align: center;  
}
```



# Limitations...

## HTML and a bit more...

```
<input type="date" />
```

# Code

# css

A dramatic scene of a volcanic eruption at night. A large, dark mountain peak is erupting, with bright orange and yellow lava flows cascading down its sides and filling the foreground. The sky is filled with dark, billowing smoke and ash. In the foreground, several human skeletons are scattered across the lava field, their arms raised as if they are communicating or performing a ritual. The overall atmosphere is one of chaos, destruction, and otherworldly beauty.

*Here be dragons...*

# Browser defaults

## CSS

- Chaque browser dispose d'une feuille de style interne par défaut
- Sensé être normé et cross-browser compatible
- En réalité... Pas vraiment.
- Utilisation de feuilles de style CSS reset

# CSS reset/reboot/normalize/younameit css

- Permet de charger des styles par défaut consistants
- Assurance d'une compatibilité cross-browser à 100%
- Pléthore de versions online
- La plus répandue <https://necolas.github.io/normalize.css/>  
(utilisée par Bootstrap, Twitter, GitHub, ...)

# CSS icons

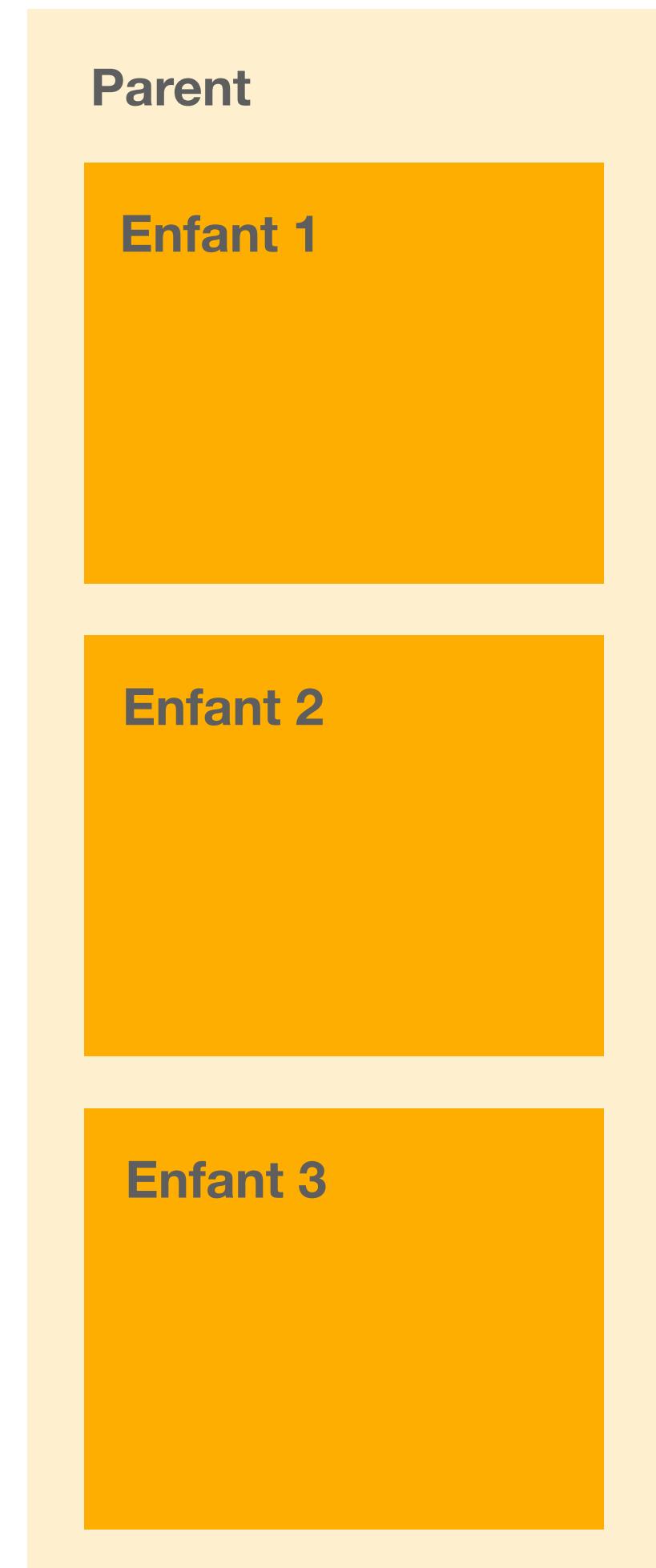
## css

- Actuellement Google Material icons
- Intégré en mode CDN
- <https://fonts.google.com/icons>
- Exemple : <face>
- Libre à vous !

# Flexboxes

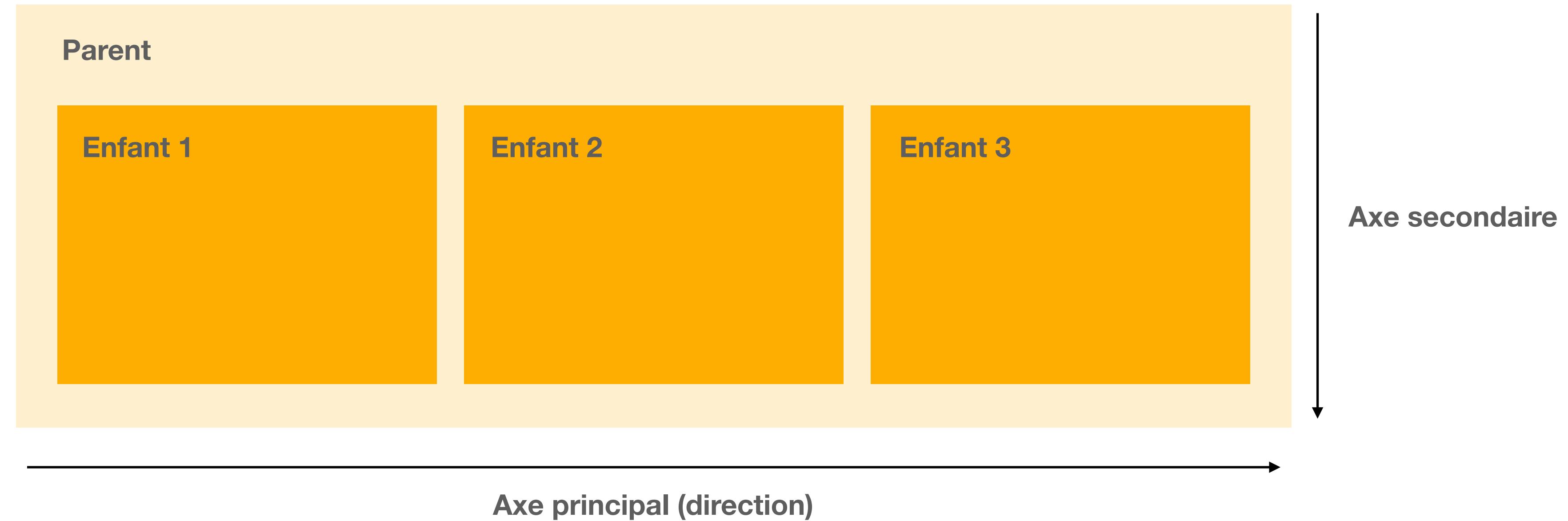
## CSS

- Permet de distribuer l'espace entre des éléments, au sein d'un parent
- Unidimensionnel - Ligne ou colonne
- Gère également l'alignement et le dimensionnement
- Comparable à des “float” gérées par le parent
- Très fortement lié au concept de “responsive design”



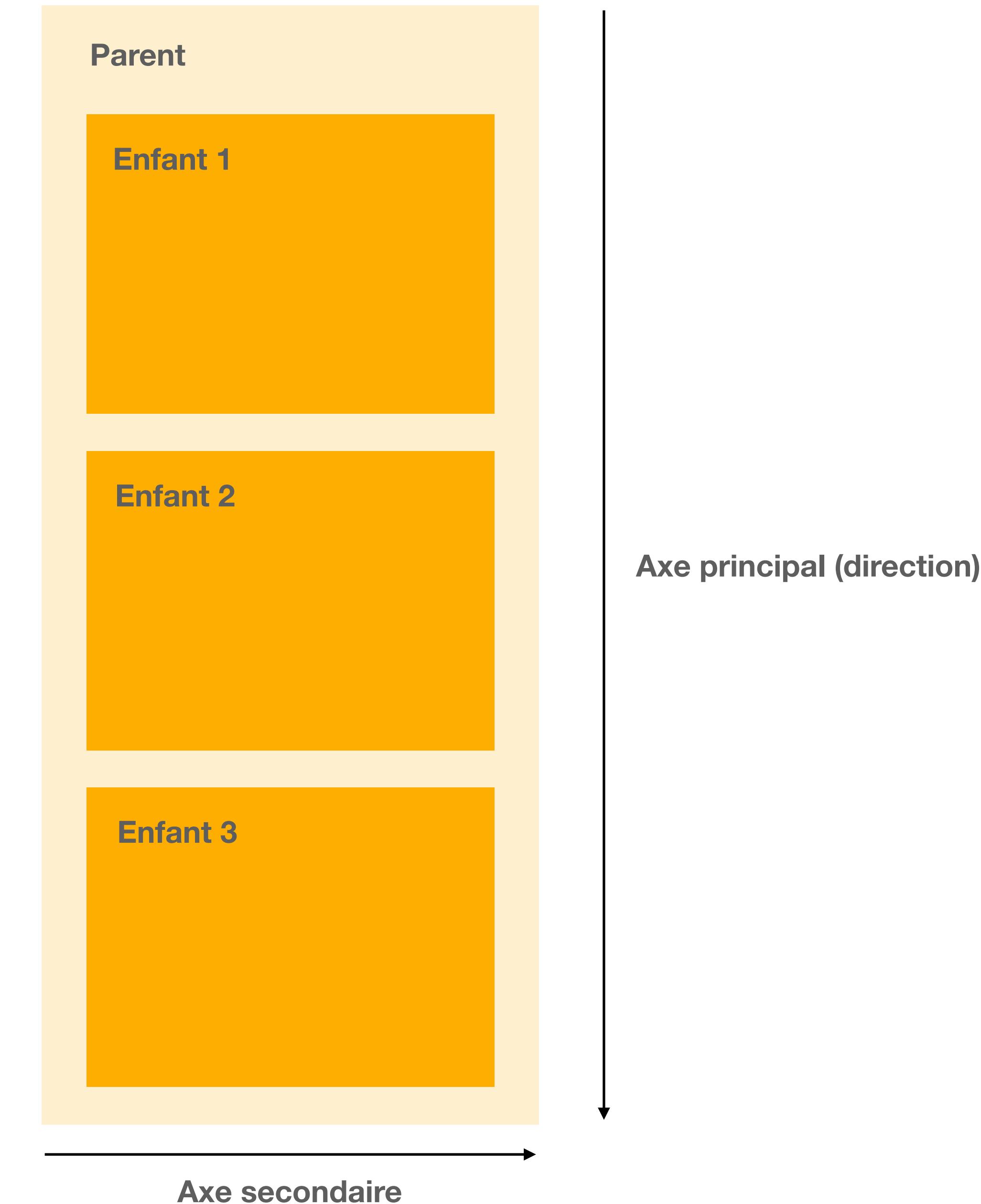
# Flexboxes

## CSS



# Flexboxes

## CSS



# Flexboxes - Propriétés CSS

## Propriétés du parent

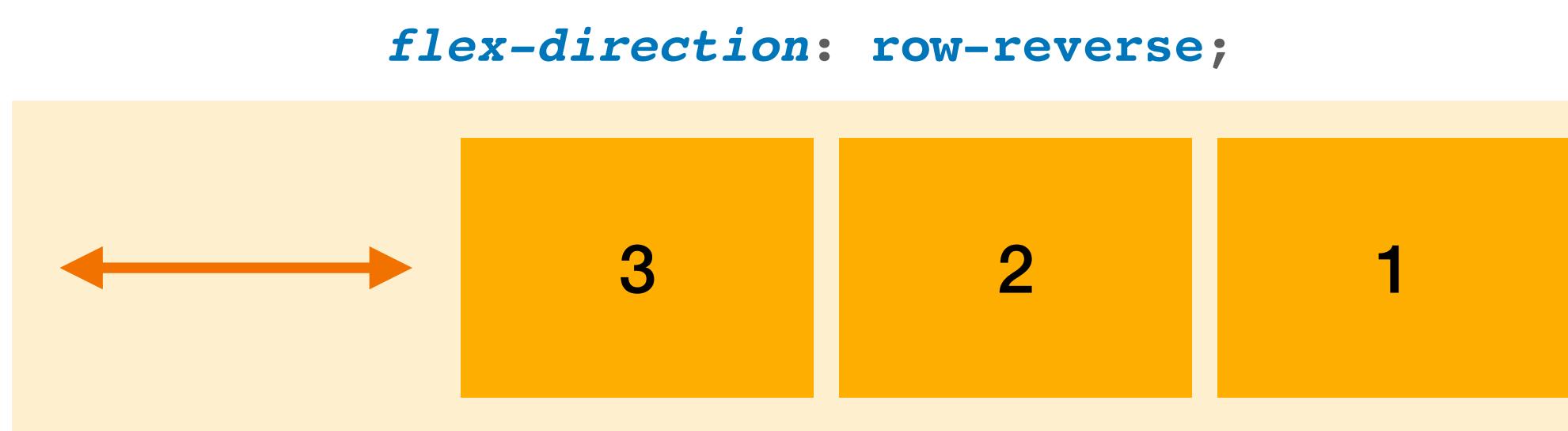
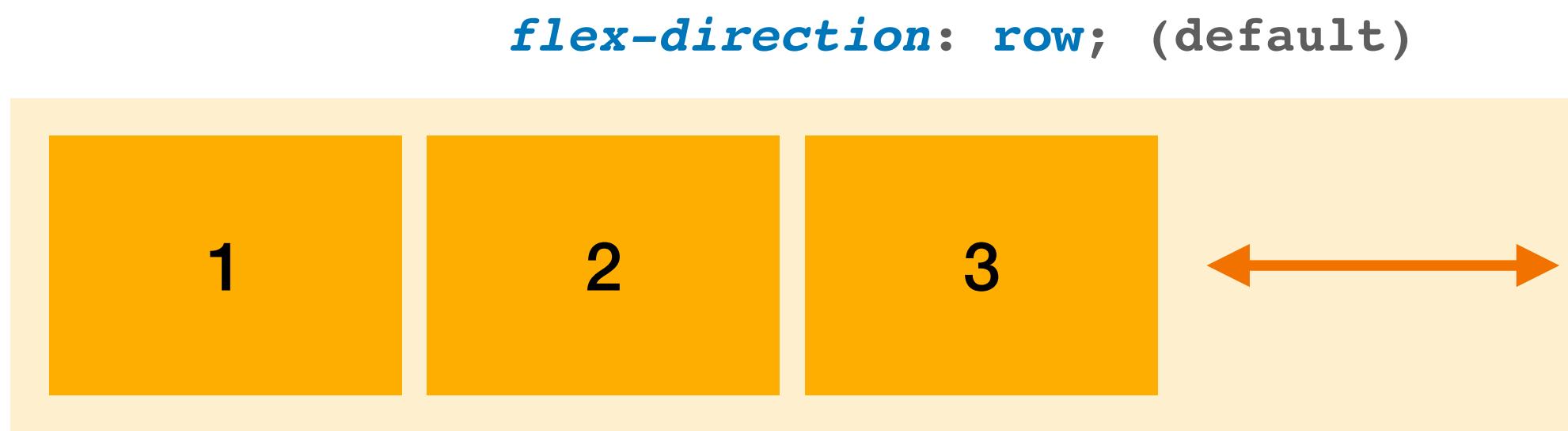
- Direction (`flex-direction`)
- Alignement axe principal (`justify-content`)
- Alignement axe secondaire (`align-items`)
- Multi-lignes (`flex-wrap`)
- Alignement multi-lignes (`align-content`)
- Ecartement (`gap`)

## Propriétés des enfants

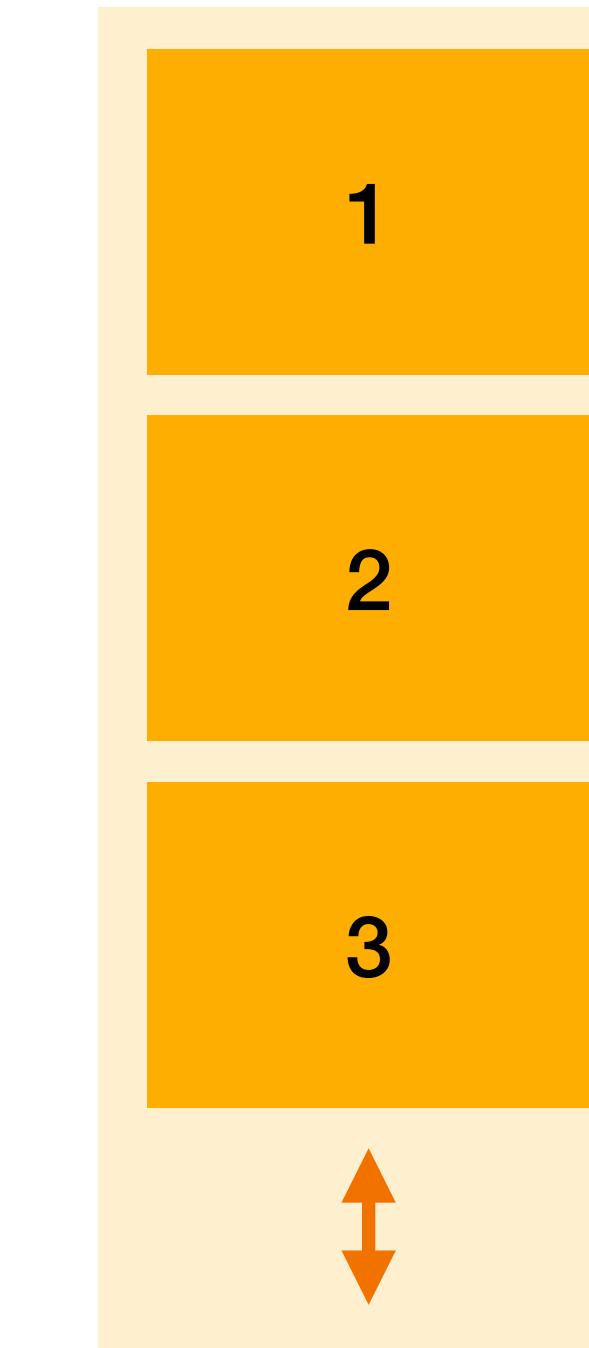
- Taille de base (`flex-basis`)
- Taille du rétrécissement (`flex-shrink`)
- Taille de l'agrandissement (`flex-grow`)
- Overrides des propriétés du parent (`...-self`)

# Flexboxes - Direction (flex-direction)

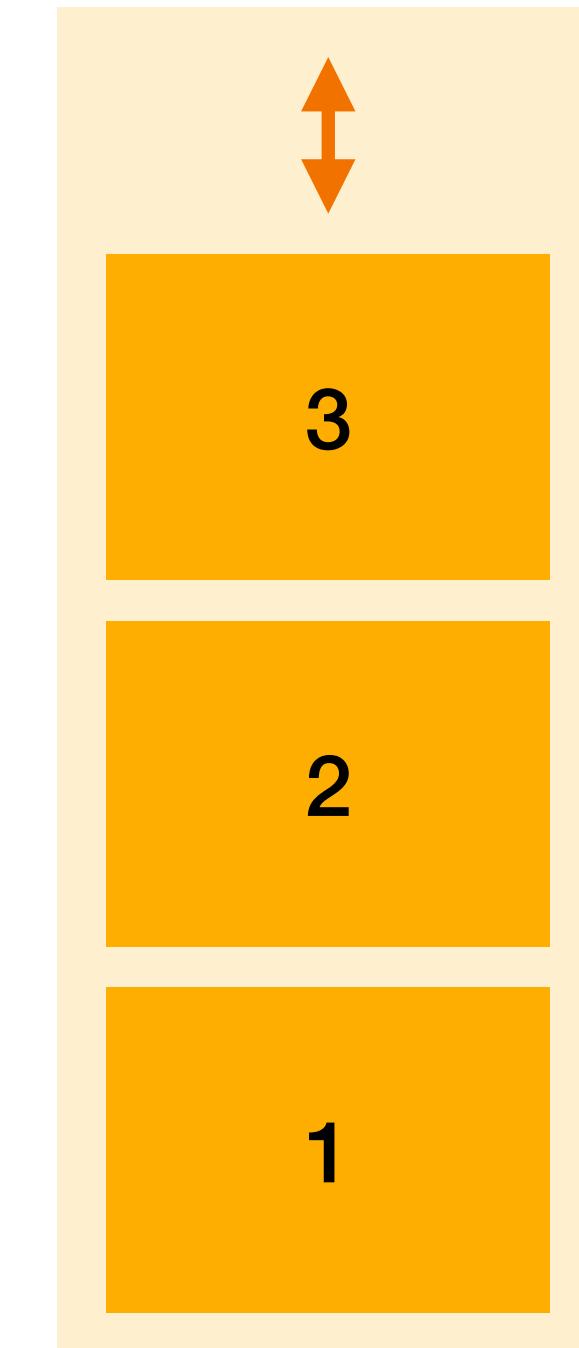
CSS



*flex-direction: column;*



*flex-direction: column-reverse;*

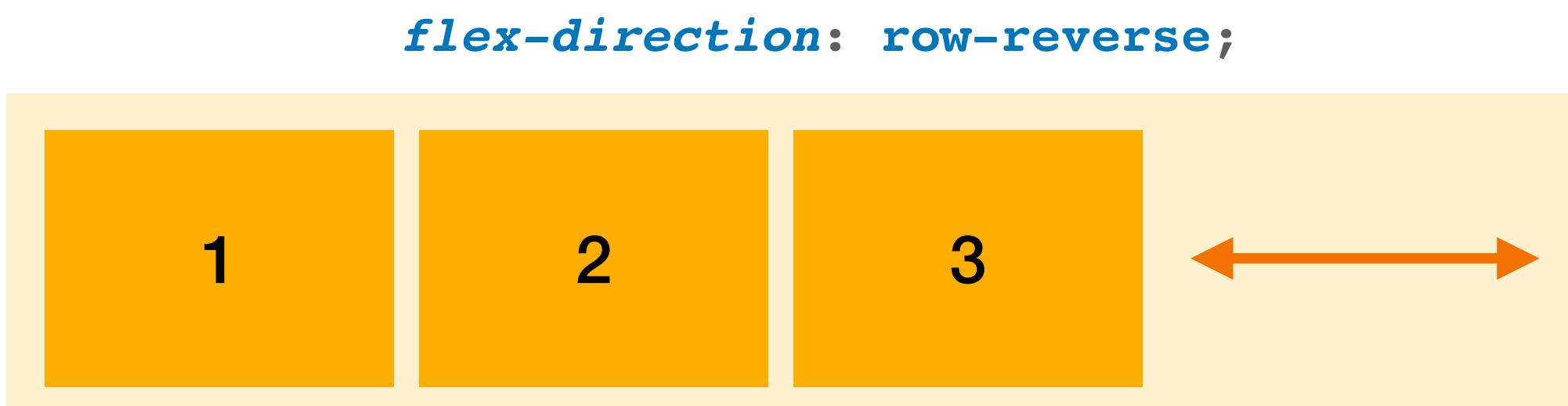
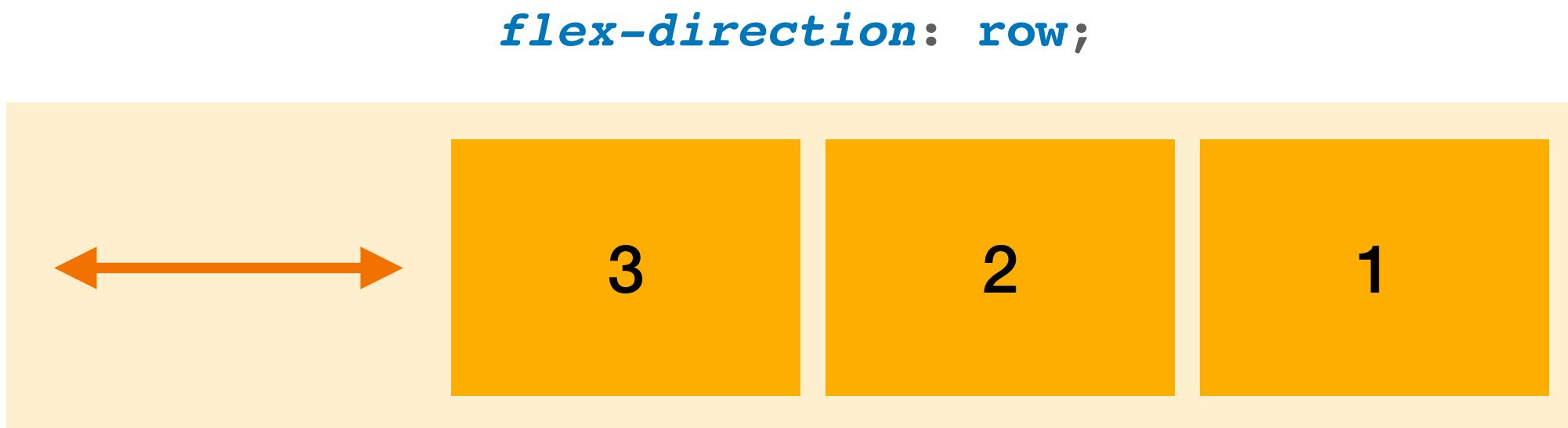


# Flexboxes - Direction rtl

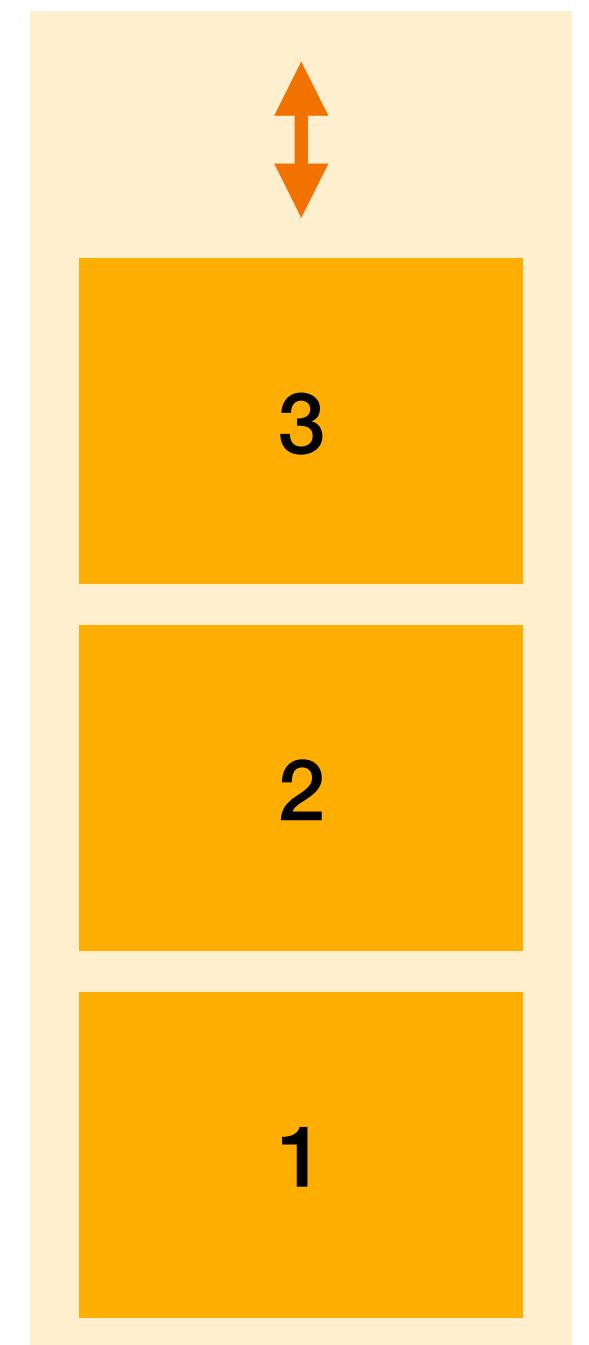
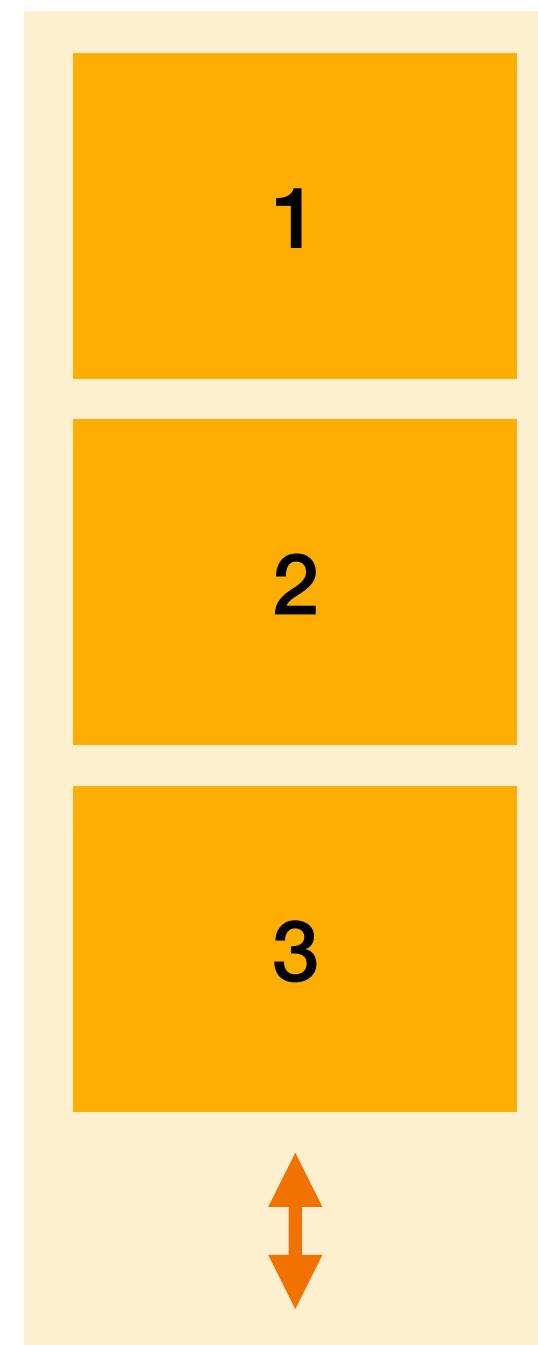
## CSS



En écriture droite-gauche (arabe) -> `direction: rtl;`



`flex-direction: column;`   `flex-direction: column-reverse;`



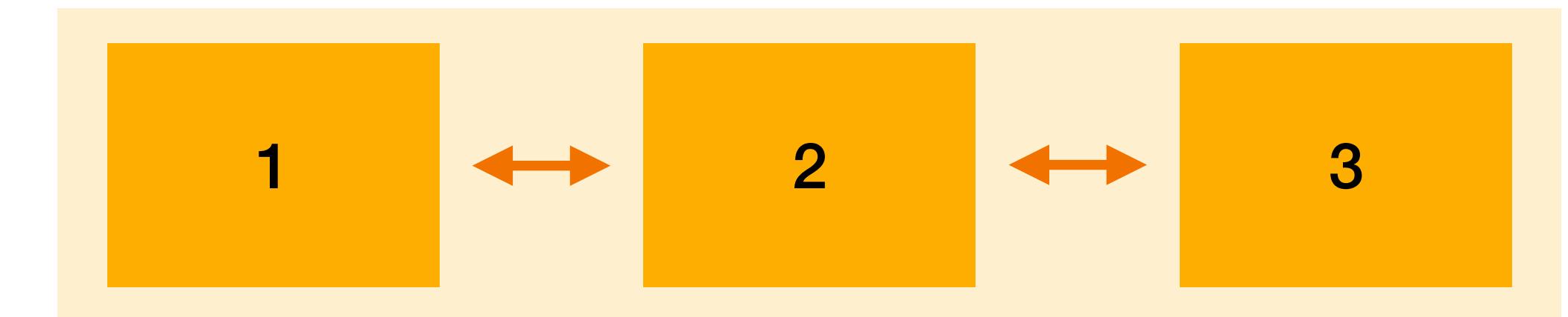
# Flexboxes - Alignement axe principal (justify-content)

## CSS

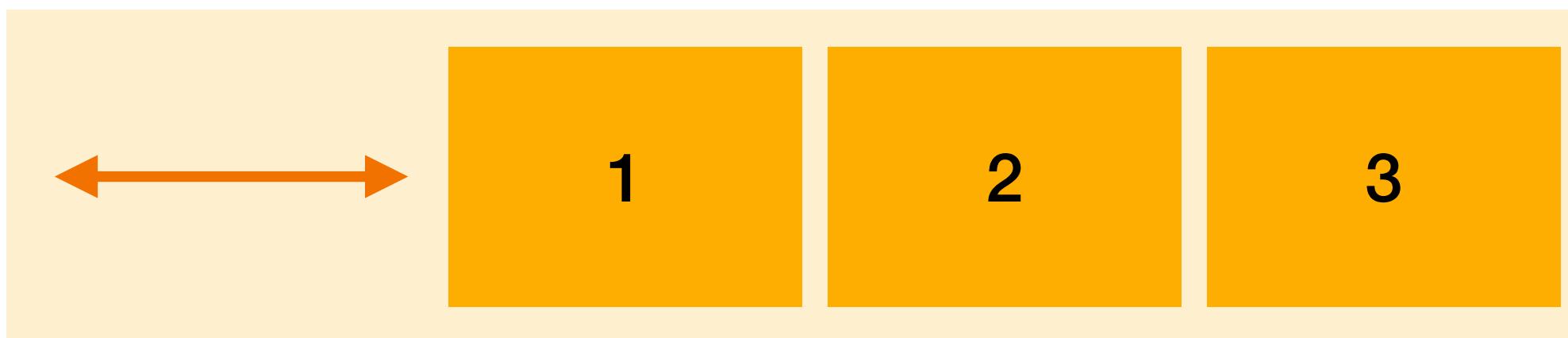
*justify-content: flex-start; (default)*



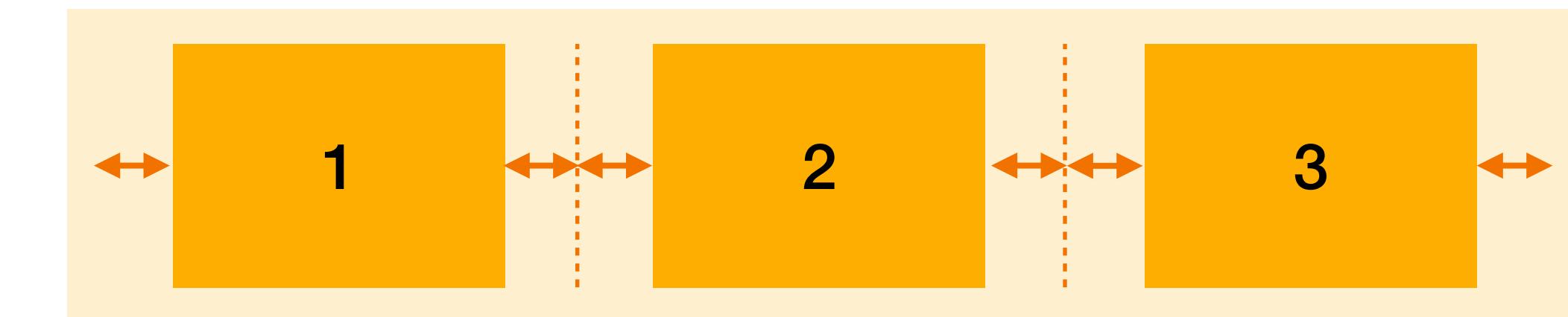
*justify-content: space-between;*



*justify-content: flex-end;*



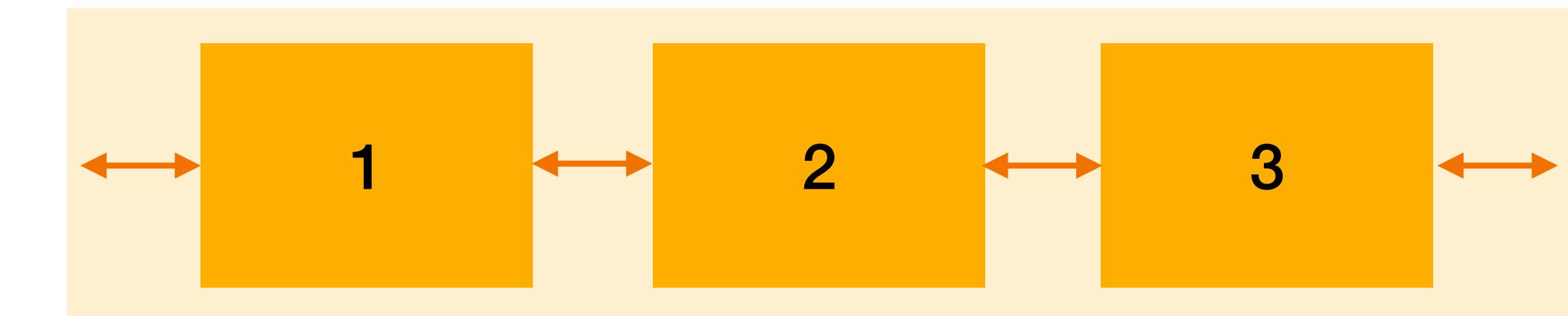
*justify-content: space-around;*



*justify-content: center;*



*justify-content: space-evenly;*



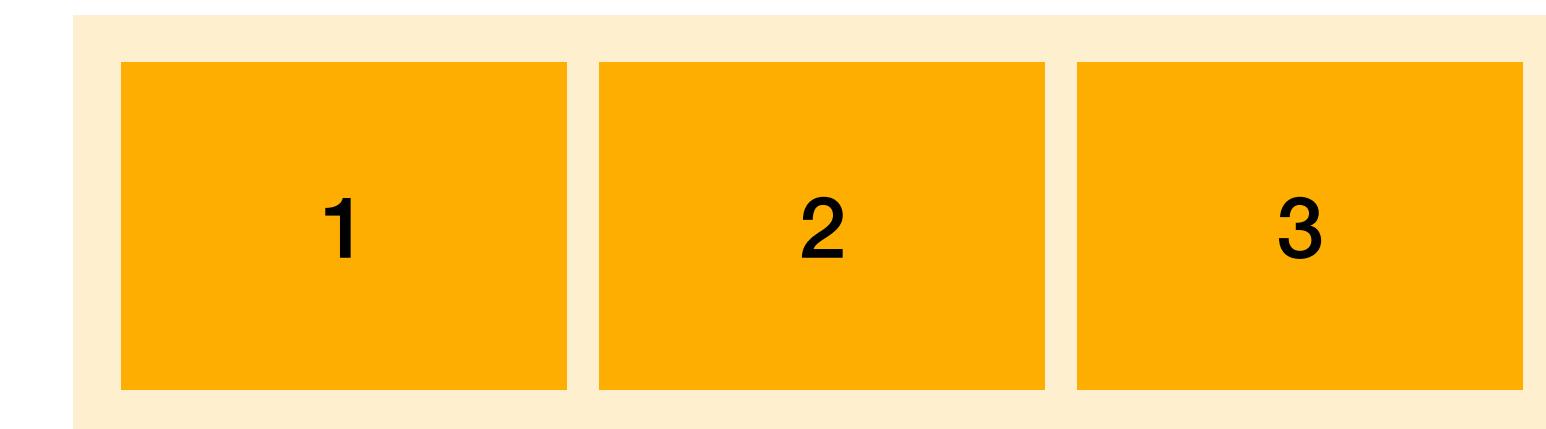
# Flexboxes - Alignement axe secondaire (align-items)

CSS

`align-items: flex-start;`



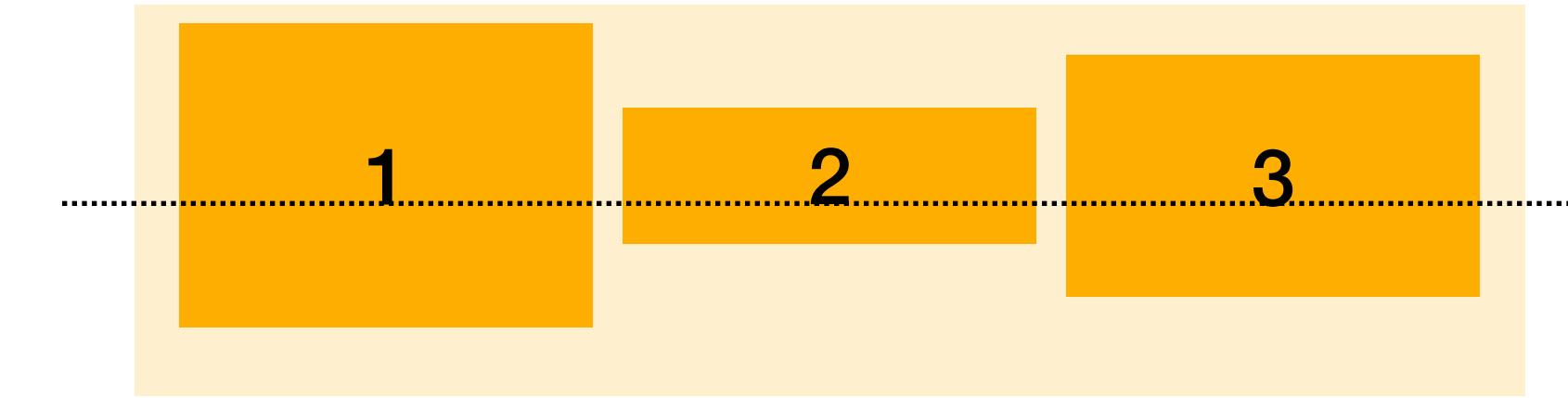
`align-items: stretch; (default)`



`align-items: flex-end;`



`align-items: baseline;`



`align-items: center;`



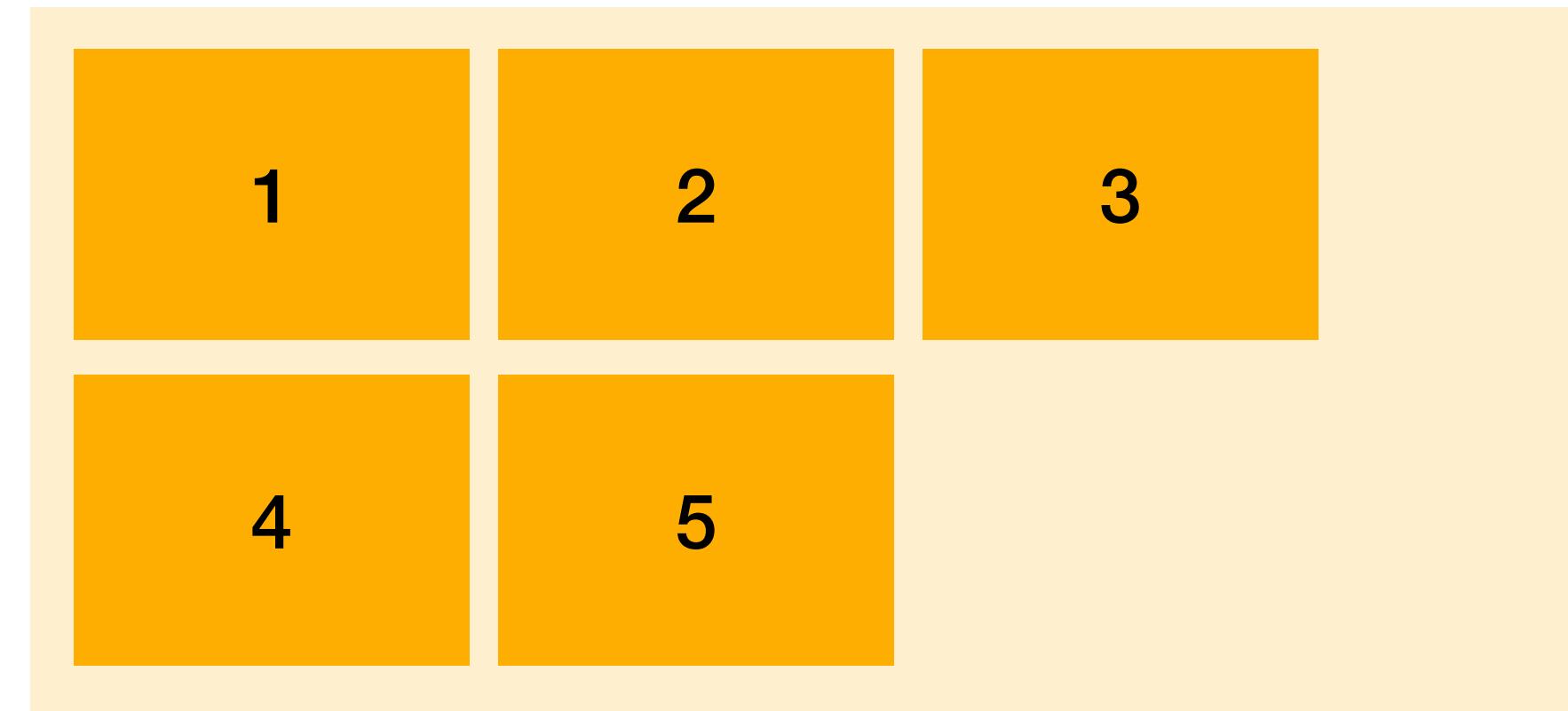
# Flexboxes - Multi-lignes (wrap)

## CSS

`flex-wrap: nowrap; (default)`



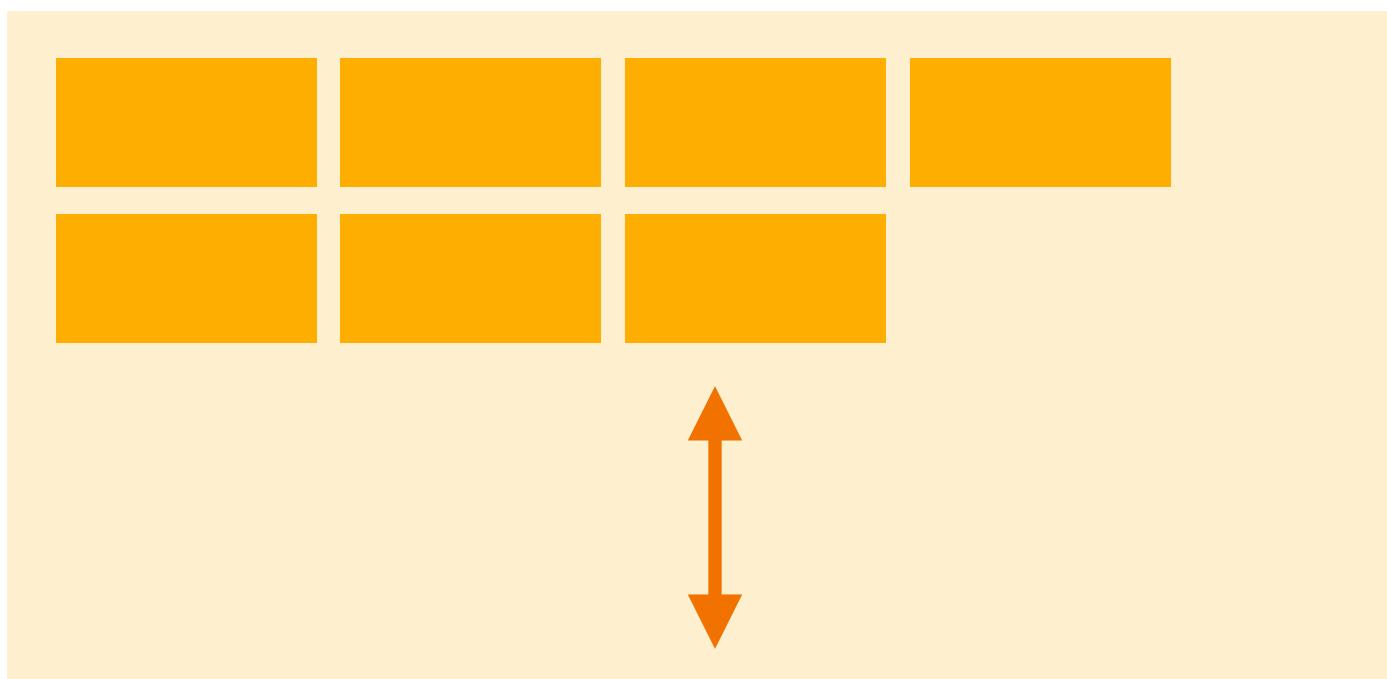
`flex-wrap: wrap;`



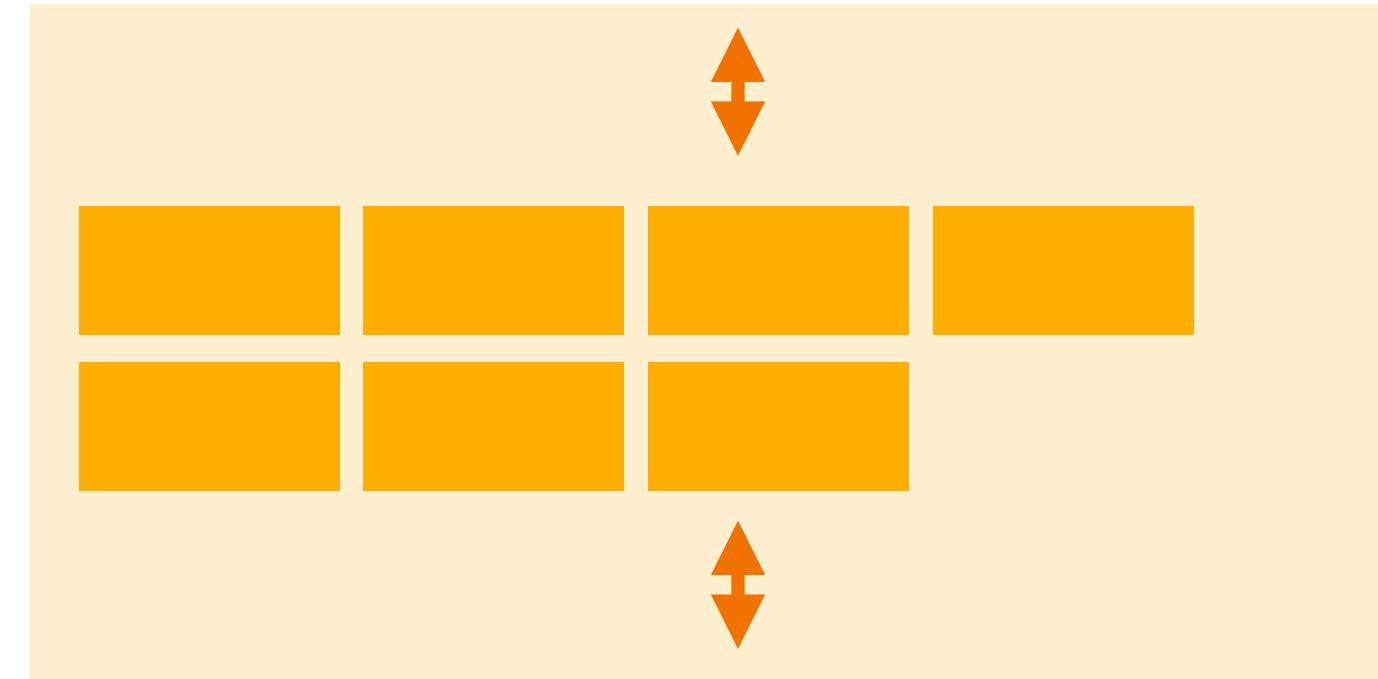
# Flexboxes - Alignement multi-lignes (align-content)

CSS

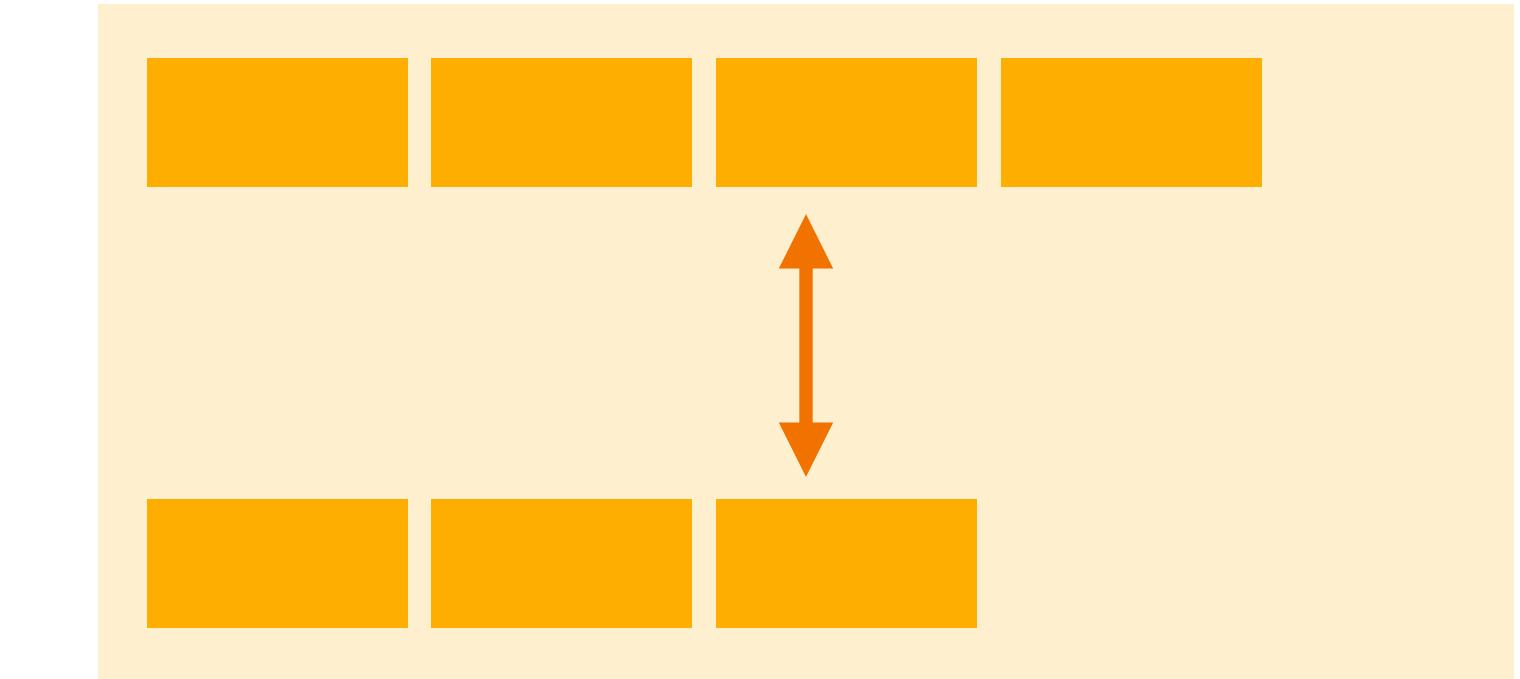
*align-content: flex-start;*



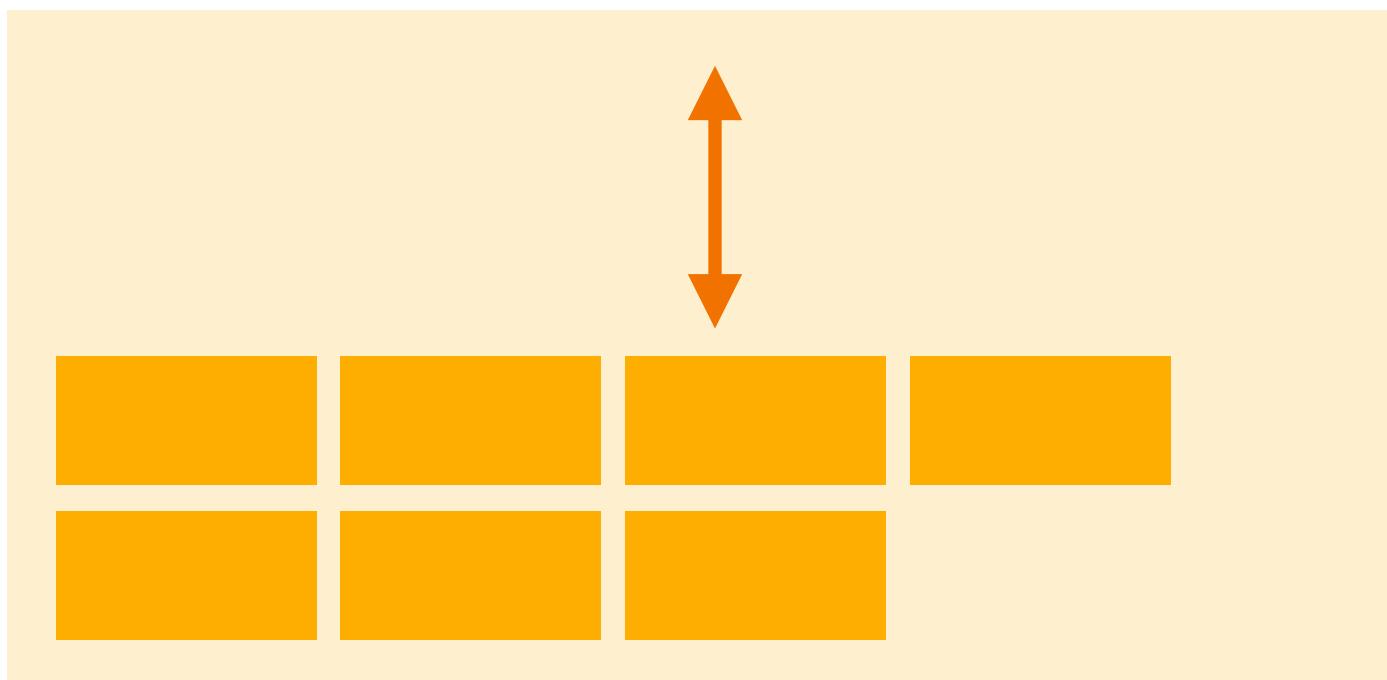
*align-content: center;*



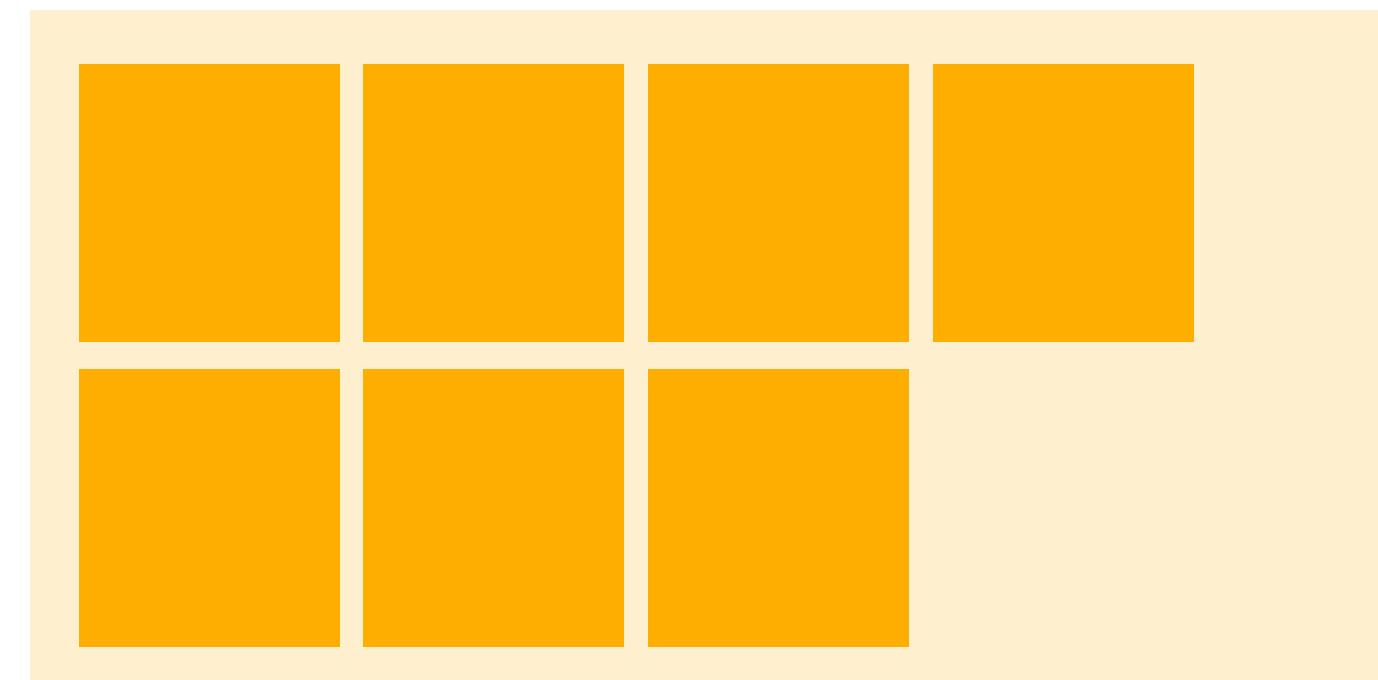
*align-content: space-between;*



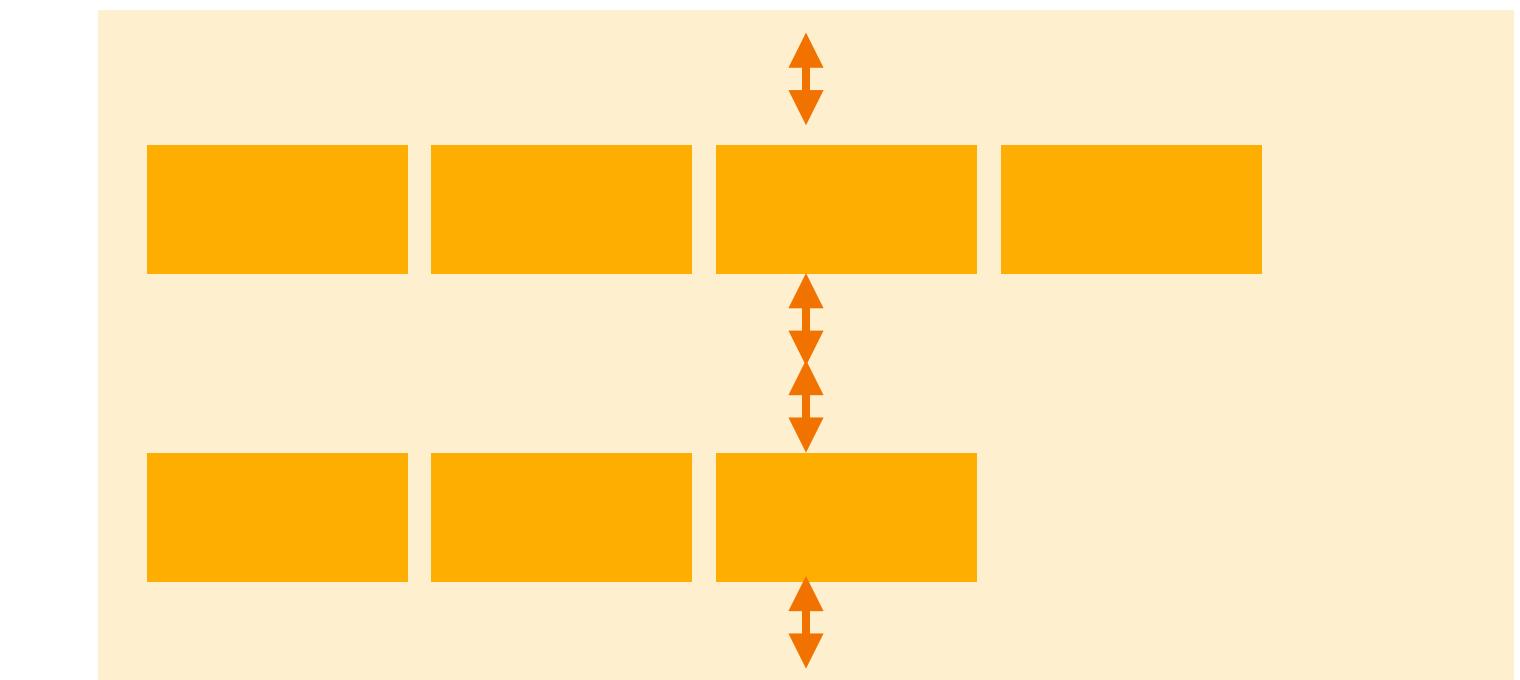
*align-content: flex-end;*



*align-content: stretch;*

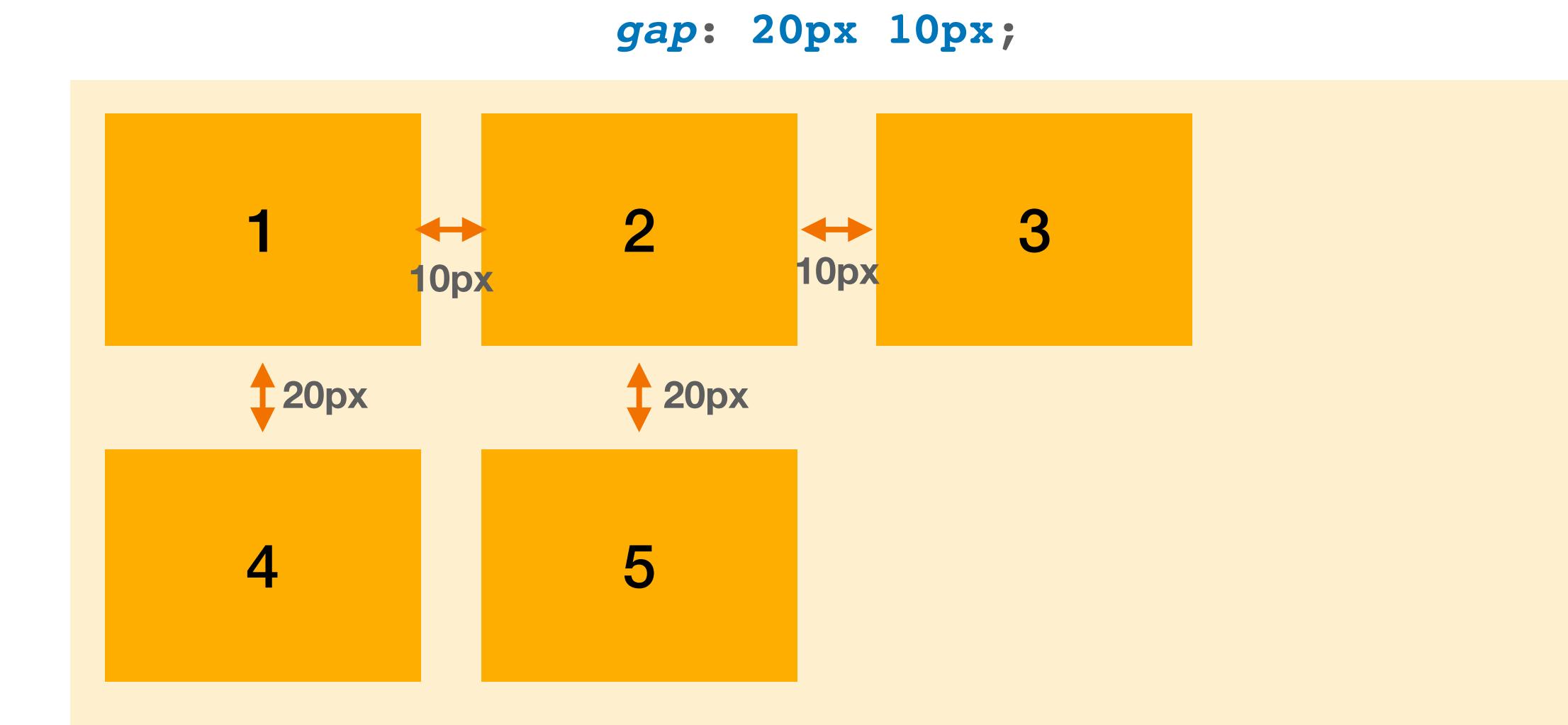
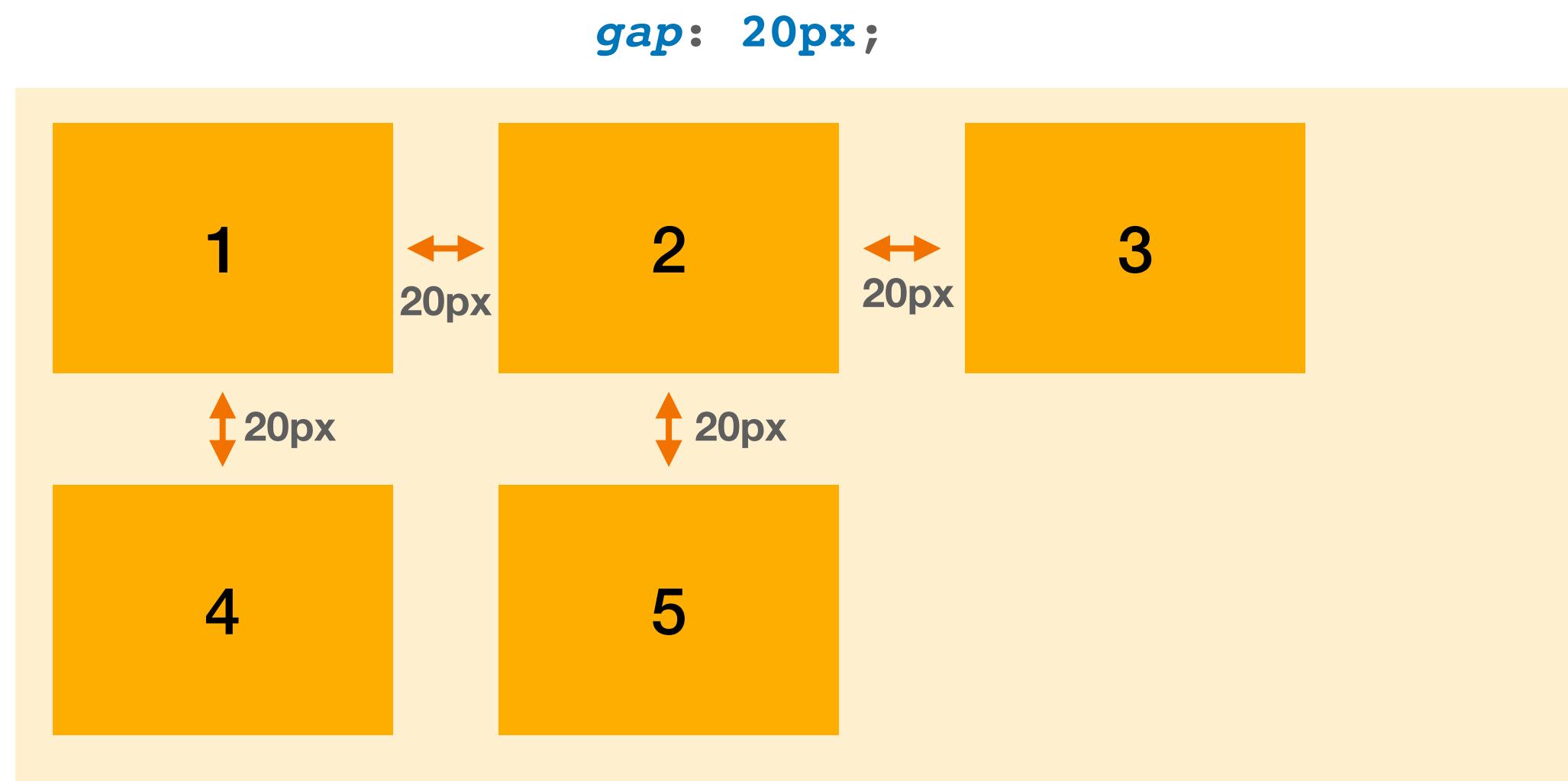


*align-content: space-around;*



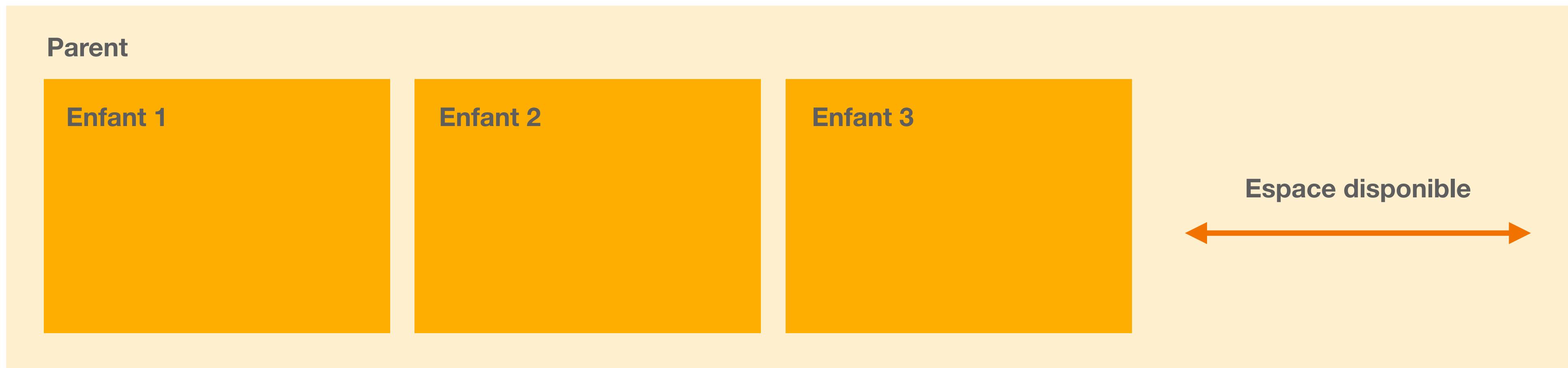
# Flexboxes - Ecartement (gap)

css



# Flexboxes - Sizing

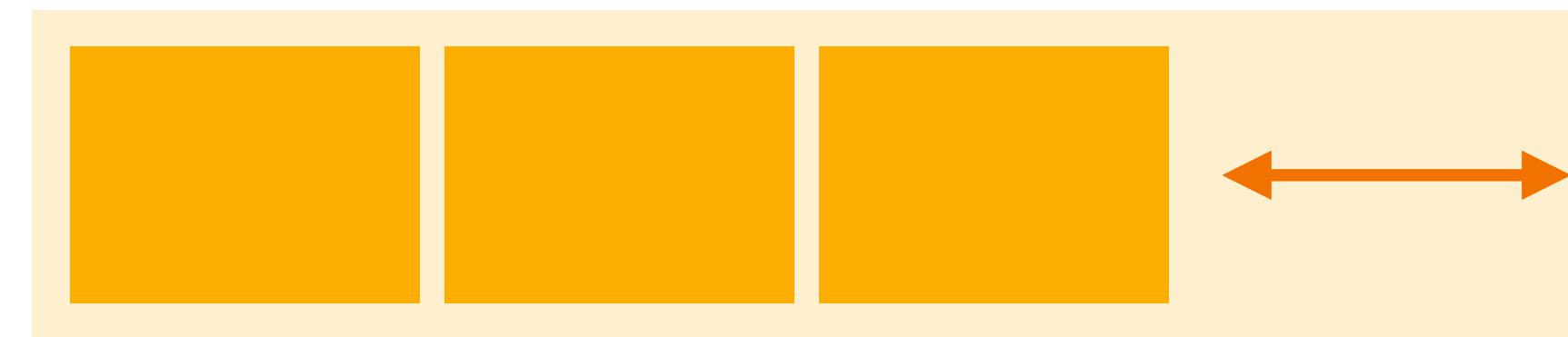
## CSS



# Flexboxes - Sizing & responsiveness

## CSS

***flex-basis: [px|%|auto|...];***



Rétrécissement

***flex-shrink: entier;***



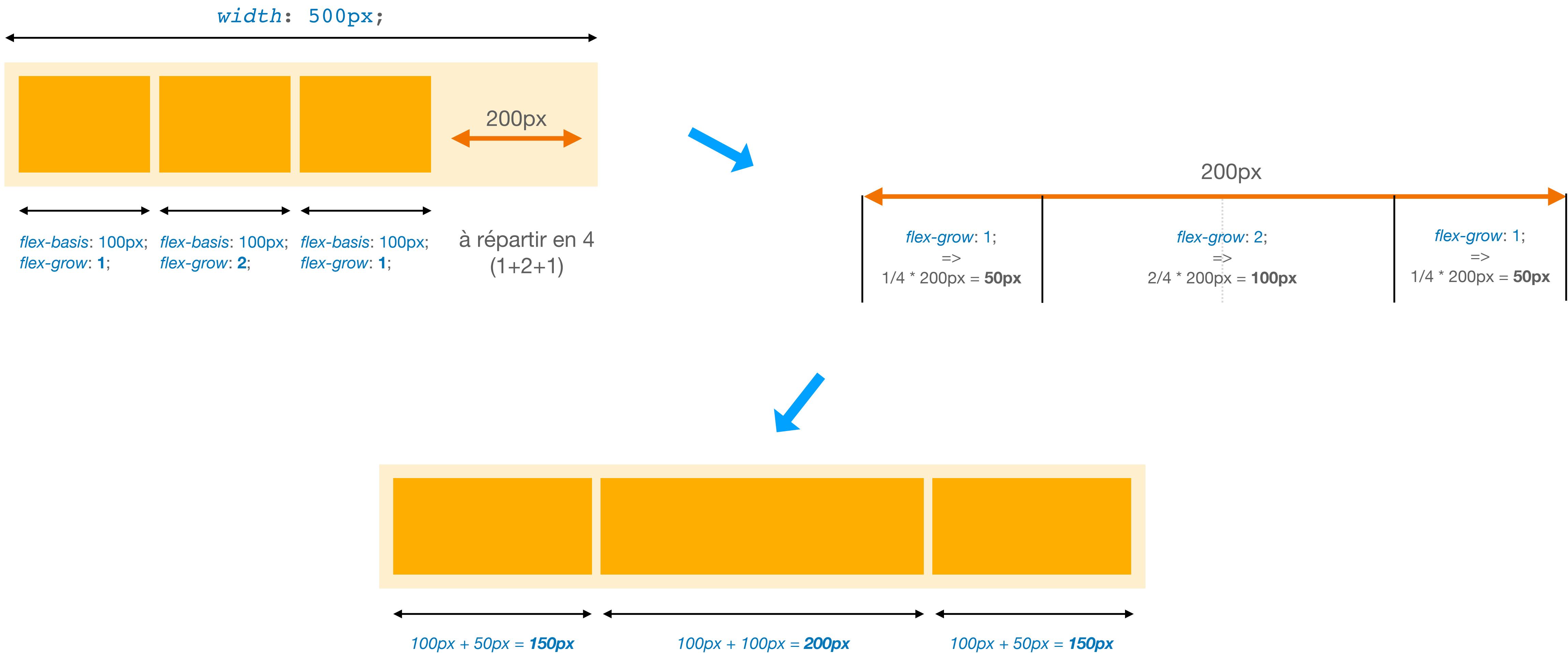
Agrandissement

***flex-grow: entier;***



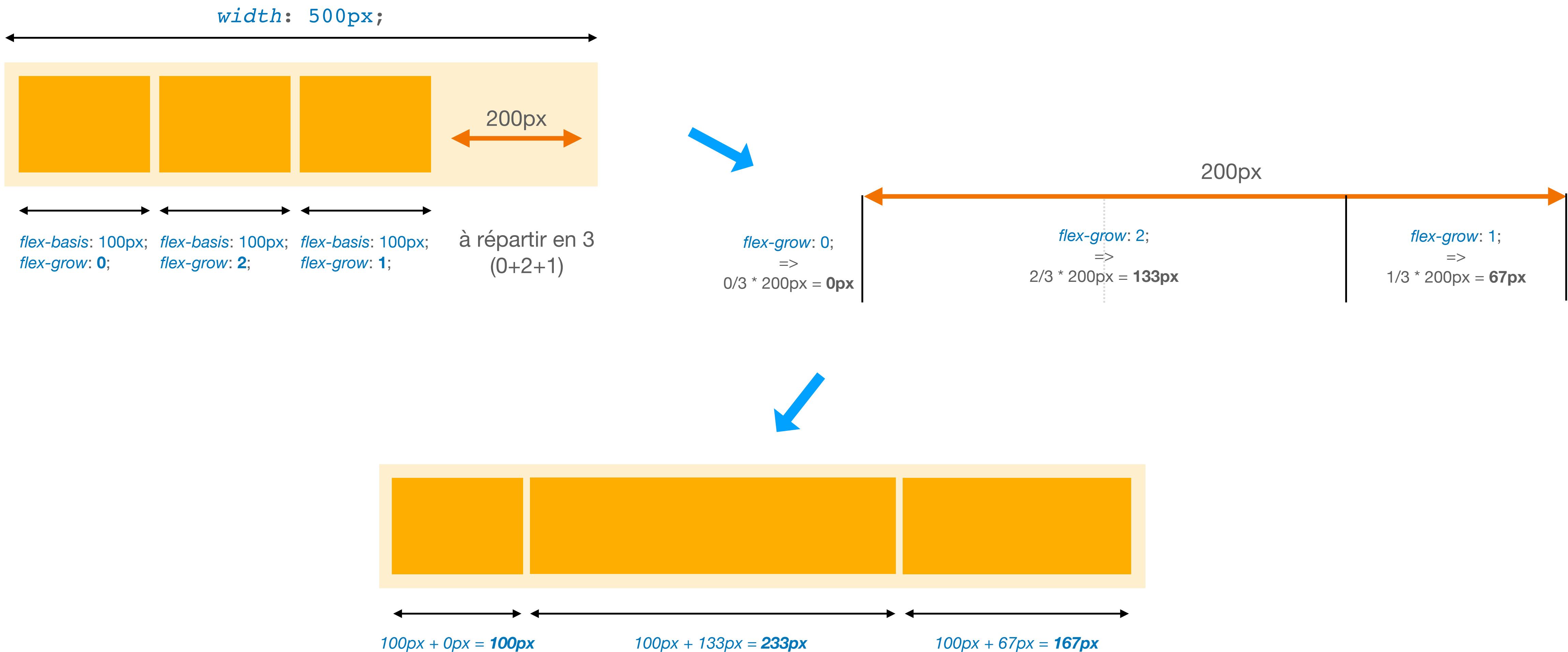
# Flexboxes - Sizing flex-grow

## CSS

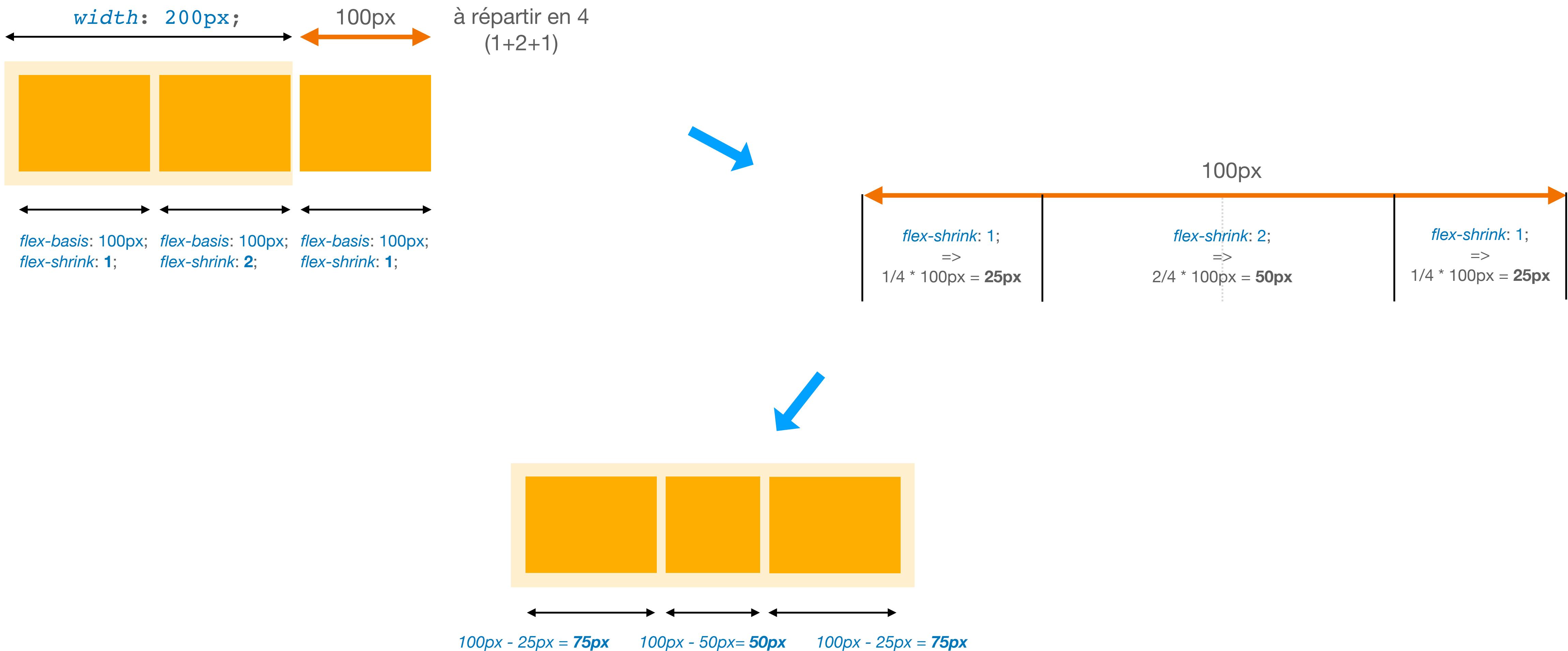


# Flexboxes - Sizing flex-grow

## CSS



# Flexboxes - Sizing flex-shrink CSS



# Flexboxes - Sizing flex-basis & sizing

## CSS



Flex-basis hérite du width (si défini) ou “auto” par défaut



Par défaut, un élément flex enfant a un min-width défini à 0 et non “auto” comme les autres éléments (resp. height selon direction)



Flex-grow vaut 0 par défaut

Cela implique que, par défaut, il ne peut être plus petit que son contenu et overflow son parent



Flex-shrink vaut 0 par défaut

Pour autoriser l’overflow, définir “min-width: auto” sur l’enfant

# Flexboxes - Overrides parents

## CSS

- `align-self` (overrides `align-items`)
- `justify-self` (overrides `justify-content`)

# Flexboxes - Order

## CSS

```
<div style="display: flex">  
  <div style="order: 2">2</div>  
  <div style="order: 3">3</div>  
  <div style="order: 1">1</div>  
</div>
```

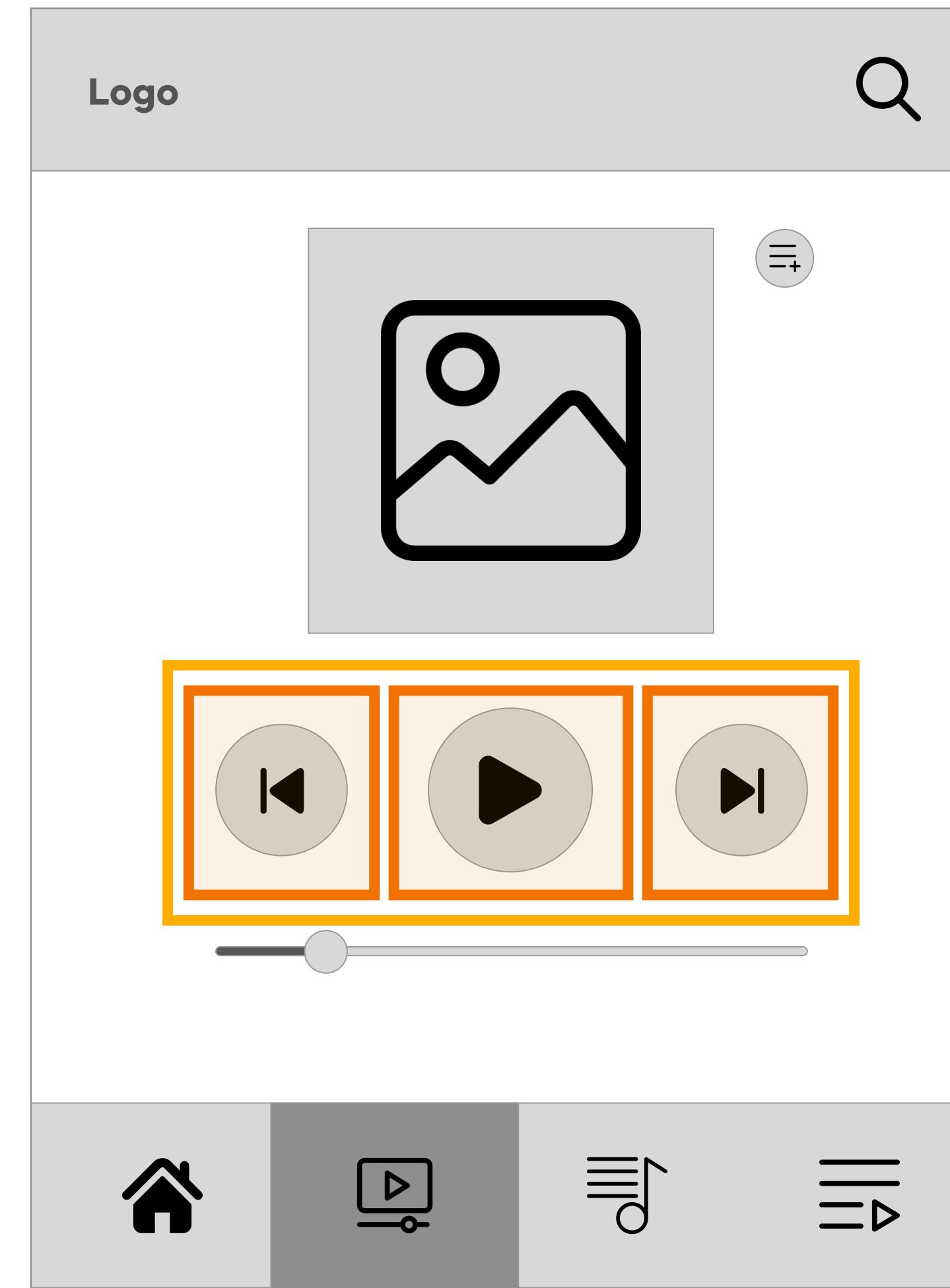


# Flexboxes dans le projet

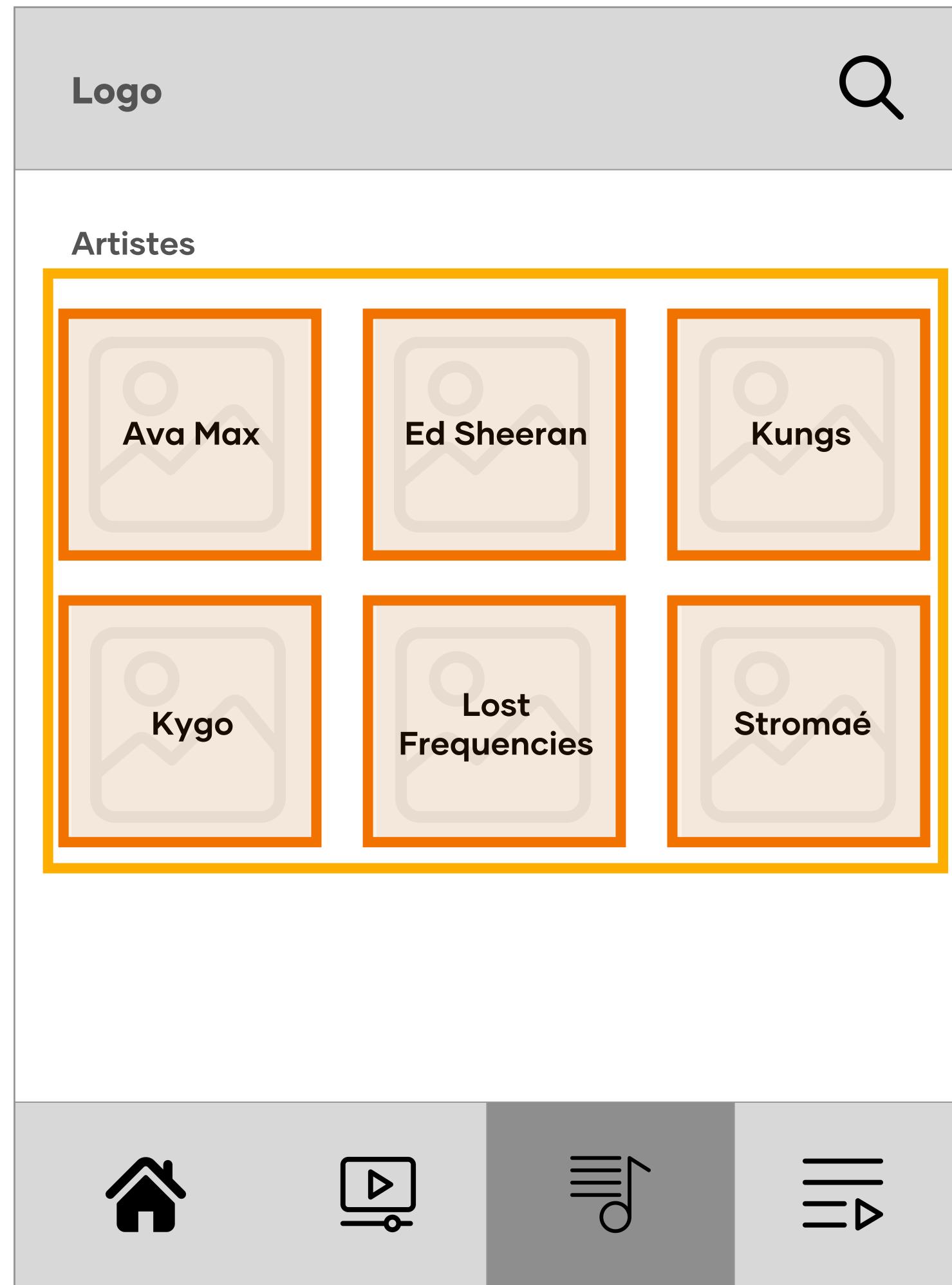
## CSS



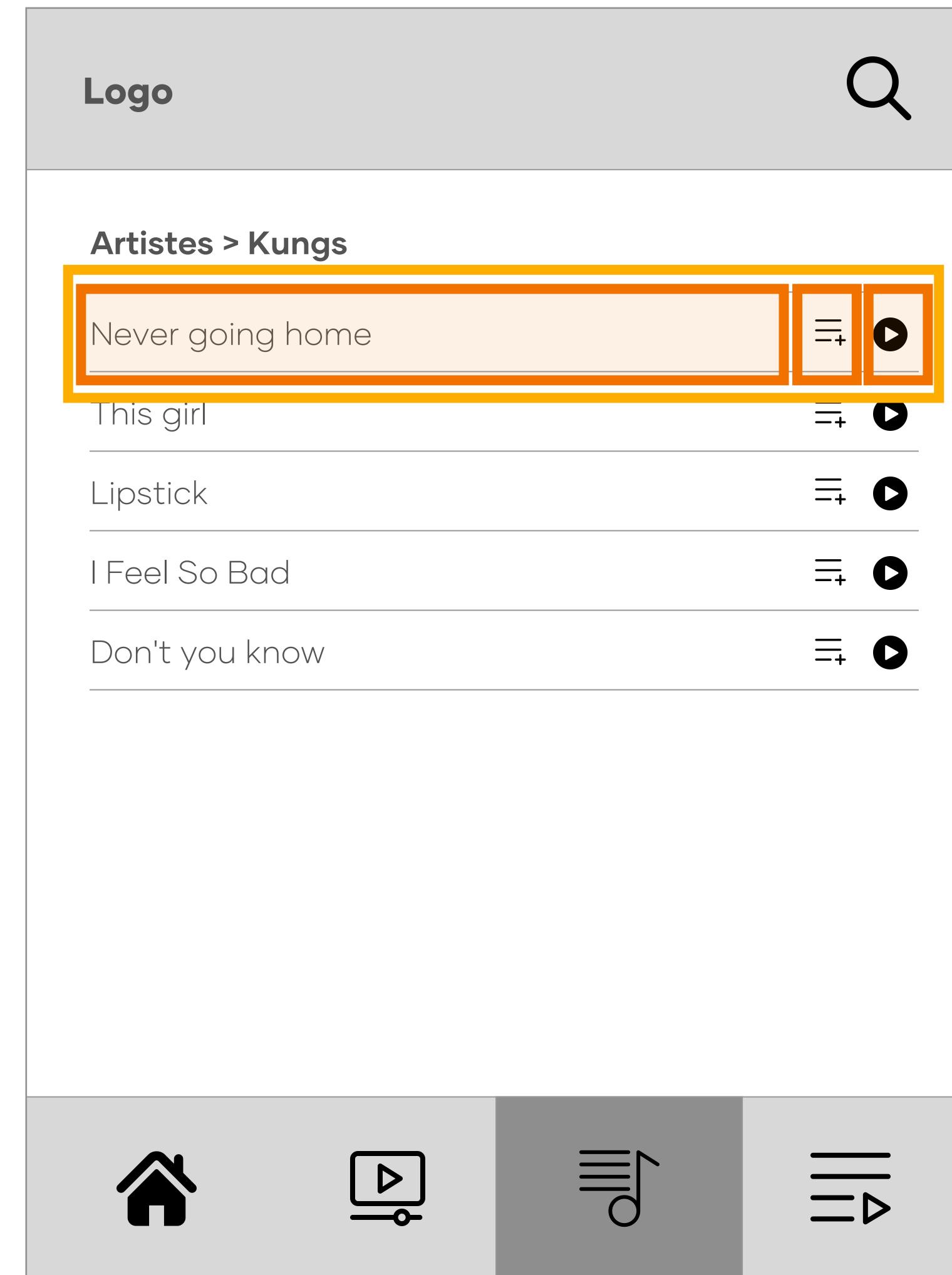
# Flexboxes dans le projet css



# Flexboxes dans le projet CSS



# Flexboxes dans le projet CSS



# Code

# En parlant de CDN...

## Tips & tricks

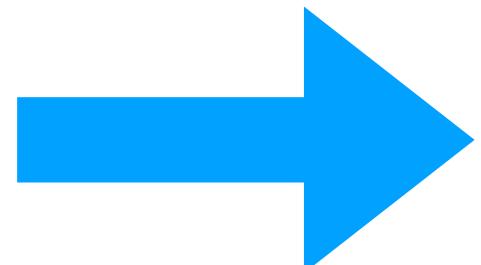
Super CDN pour des images **placeholder**, peu importe votre projet :

**placekitten.com**

**Usage: [http://placekitten.com/\[width\]/\[height\]](http://placekitten.com/[width]/[height])**

```

```



# Unités CSS

## CSS

Deux types d'unités :

- Unités absolues - px, cm, pt, in, ...
- Unités relatives - %, em, rem, vw, vh, ...

# Unités CSS - em css

- 1 em = Taille de police à 100% de l'élément parent
- Si pas d'élément parent, 1 em est égal à la hauteur d'une lettre en taille standard, selon la résolution d'écran

# Unités CSS - em

## CSS

Pas de parent ? Résolution par défaut. Exemple: 16px

```
<body>  
<div>  
  <p>Hello</p>  
</div>  
</body>
```

```
body {  
  font-size: 1em;  
}  
  
div {  
  font-size: 0.75em;      3/4 du parent -> 9 pixels  
}  
  
p {  
  font-size: 0.75em;  
}
```

# Unités CSS - rem

## css

- 1 rem = Taille de police à 100% de l'élément root (**Root EM**)
- Si pas d'élément parent, 1 rem est égal à la hauteur d'une lettre en taille standard, selon la résolution d'écran

# Unités CSS - rem

## css

Pas de parent ? Résolution par défaut. Exemple: 16px

```
<body>  
<div>  
  <p>Hello</p>  
</div>  
</body>
```

```
body {  
  font-size: 1rem;  
}  
  
div {  
  font-size: 0.75rem; 3/4 du root -> 12 pixels  
}  
  
p {  
  font-size: 0.75rem;  
}
```

# Unités CSS - rem

## css

- Le REM est beaucoup plus utilisé, typiquement pour des design responsive ou dans des frameworks CSS
- En définissant toutes les tailles de polices et marges en rem, d'après l'élément root, une seule adaptation du CSS met à jour l'entier du document

# Unités CSS - rem

## CSS

```
body {  
    font-size: 16px;  
}  
  
h1 {  
    font-size: 2.5rem;  
    margin-bottom: 1rem;  
}  
  
h2 {  
    font-size: 2rem;  
    margin-bottom: 0.75rem;  
}  
...
```

# Unités CSS - vw, vh

## CSS

- vw = viewport width
- vh = viewport height
- Très utile pour donner la largeur ou la hauteur de l'écran à un élément, sans devoir chaîner des `height: 100%` sur tous les parents

# Variables CSS

## CSS

- CSS dispose aussi de variables, comme Javascript par exemple
- Une variable commence toujours par "--"
- Ce sont en fait des custom properties... mais utilisées comme variables !
- Une variable peut être déclarée sur n'importe quel déclaration, mais ne sera accessible que par l'élément et ses enfants

# Variables CSS

## css

### Déclaration

```
body {  
  --text-color: #00ff00;  
}
```

### Utilisation

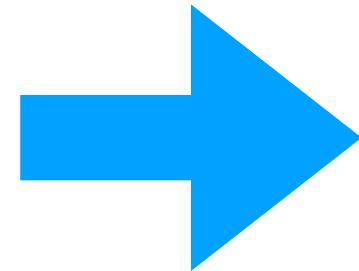
```
h1 {  
  color: var(--text-color);  
}
```

# Variables CSS - Portée css

## Exemple

```
header {  
  --text-color: #00ff00;  
}
```

```
footer {  
  /* rien. */  
}
```



```
header h1 {  
  color: var(--text-color); /* Yes! */  
}
```

```
footer h1 {  
  color: var(--text-color); /* Nope. Invalide */  
}
```

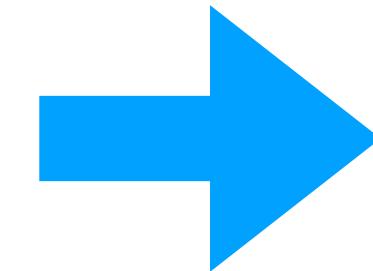
# Variables CSS - Déclaration idéale

## css

```
/* Utiliser le pseudo element :root */  
  
:root {  
  --text-color: #00ff00;  
}
```

# Variables CSS - Variable d'une variable CSS

```
:root {  
  --primary-color: #00ff00;  
  
  --link-color: var(--primary-color)  
}
```



```
--primary-color => #00ff00  
  
--link-color => #00ff00  
}
```

# Viewport css

- Le viewport est la zone de la fenêtre dans laquelle le contenu web peut être vu
- Le viewport est souvent plus grand que la zone affichée par le navigateur ->  
La view

# Viewport CSS



[https://developer.mozilla.org/fr/docs/Web/HTML/Viewport\\_meta\\_tag](https://developer.mozilla.org/fr/docs/Web/HTML/Viewport_meta_tag)

# Viewport - Déjà vu ? Vue compactée ?

## CSS



[https://developer.mozilla.org/fr/docs/Web/HTML/Viewport\\_meta\\_tag](https://developer.mozilla.org/fr/docs/Web/HTML/Viewport_meta_tag)

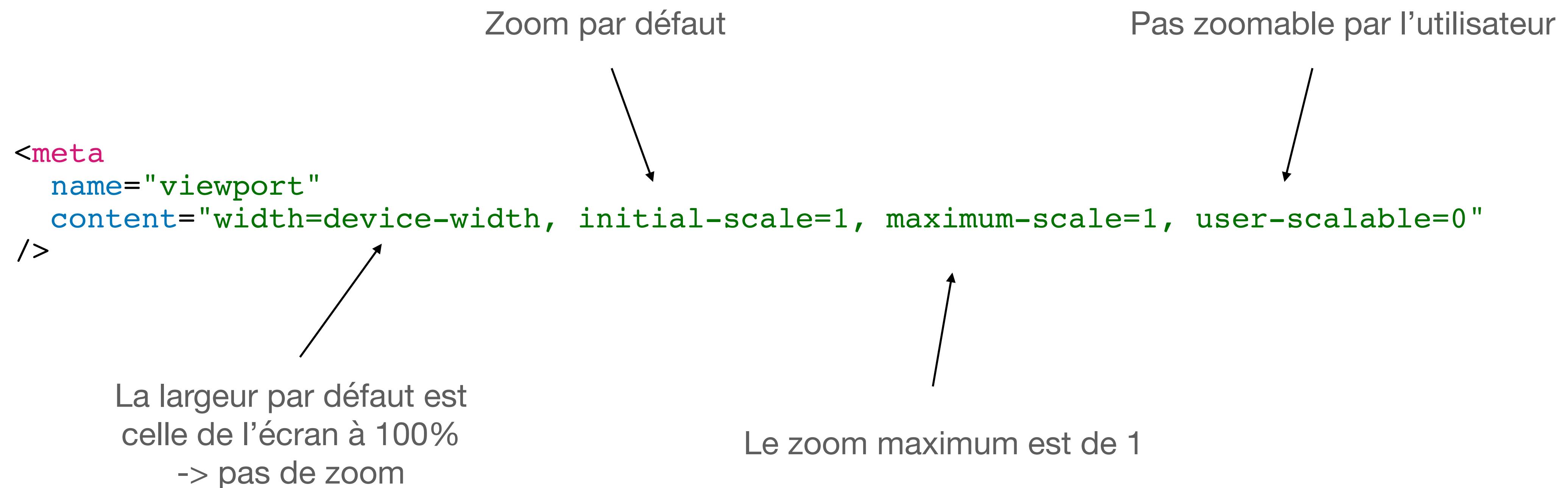
# Viewport - Autozoom

## CSS

- Les résolutions des versions mobiles sont de plus en plus précises et on ne distingue plus les pixels
- Un mécanisme d'autozoom est utilisé par les navigateurs pour renderer une version plus large de la page et la redimensionner à une résolution plus petite
- Par exemple, si l'écran d'un téléphone mobile a une largeur de 640 pixels, les pages peuvent être affichées dans une fenêtre virtuelle de 980 pixels, puis réduites pour tenir dans l'espace de 640 pixels.

# Viewport - La balise meta viewport

## CSS



# Media queries

## CSS

- Design responsive
- Blocs CSS conditionnels liés au type de display ou à sa taille
- Applicable à certains tags HTML via l'attribut `media=`
- Utilisable via Javascript pour tester et surveiller

# Media queries - Blocs CSS

```
@media [not|only] mediatype and (mediafeature [and|or|not] mediafeature) {  
    ...  
    CSS-Code;  
    ...  
}
```

*mediatype* = all | print | screen | speech

*mediafeature* = [max-width | min-height | orientation | ...]: value

# Media queries - Blocs CSS

## CSS

### Exemple 1

```
.container {  
  display: flex;  
  flex-direction: row;  
}  
...  
  
@media (max-width: 767px) {  
  .container {  
    flex-direction: column;  
  }  
}
```

### Exemple 2

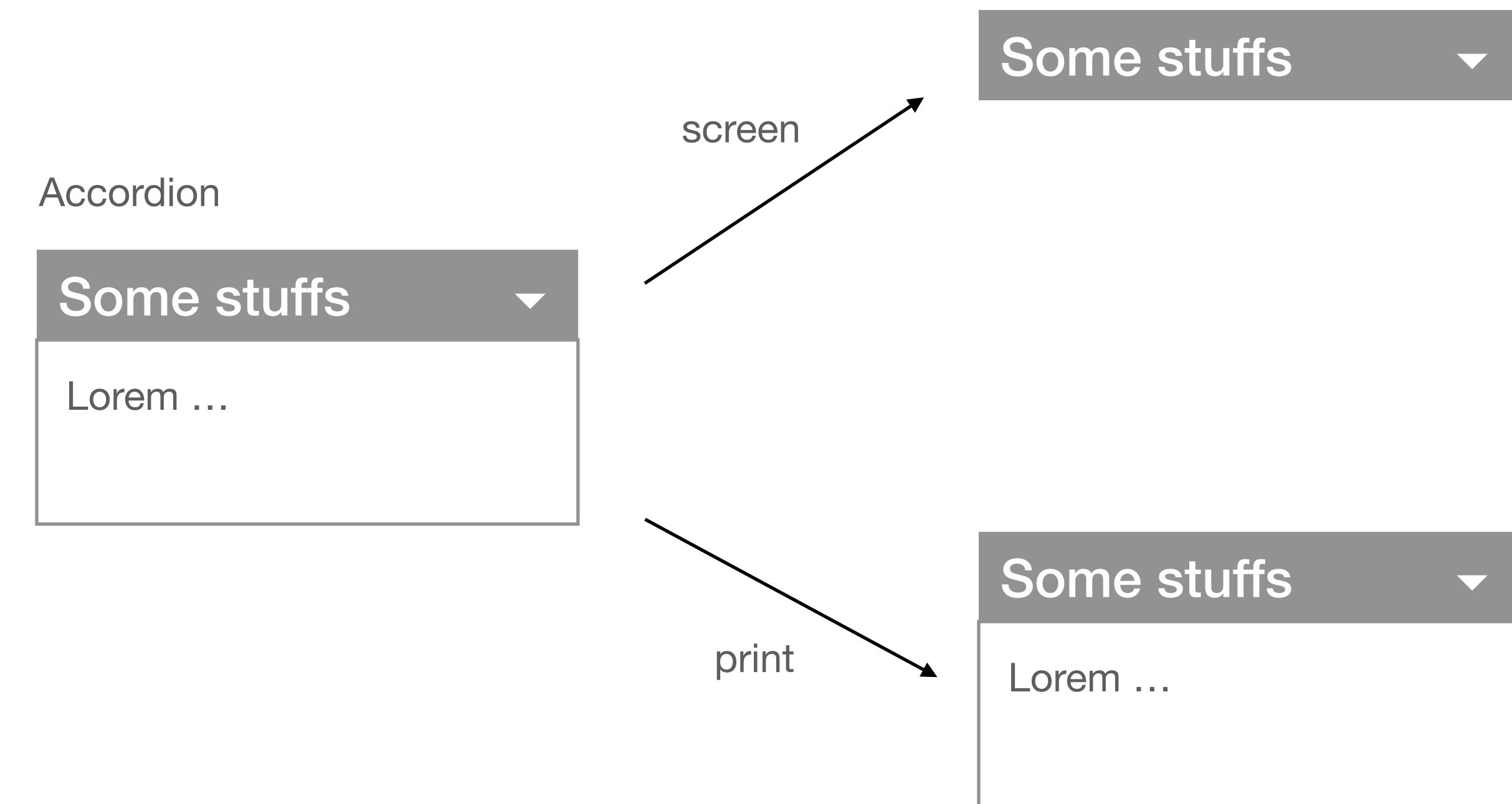
```
body {  
  background-color: white;  
}  
...  
  
@media screen and (prefers-color-scheme: dark) {  
  body {  
    background-color: black;  
  }  
}
```

# Media queries - Blocs CSS

## CSS

### Exemple 3

```
.accordion .accordion-content {  
  display: none;  
}  
...  
  
@media print {  
  .accordion .accordion-content {  
    display: block;  
}  
}
```



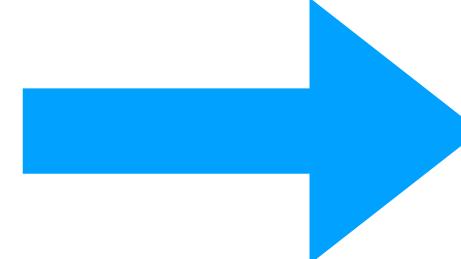
# Media queries - Blocs CSS



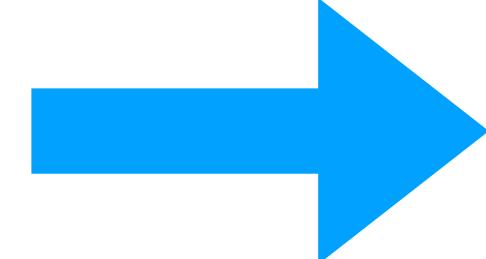
Cumul des règles par qualification des sélecteurs

```
.container {  
  display: flex;  
  flex-direction: row;  
}  
...  
  
@media (max-width: 767px) {  
  .container {  
    flex-direction: column;  
  }  
}
```

Sélecteur interprété  
comme



```
.container {  
  display: flex;  
  flex-direction: row;  
}  
...  
  
@media (max-width: 767px) .container {  
  flex-direction: column;  
}
```



# Media queries - Blocs CSS

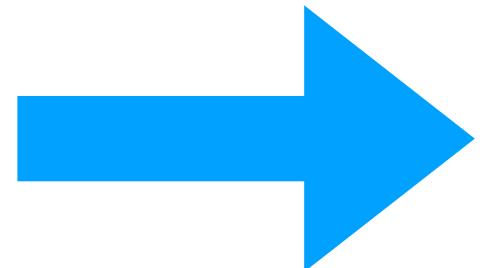
## CSS



Cumul des règles par qualification des sélecteurs

```
.container {  
  display: flex;  
  flex-direction: row;  
}  
...  
  
@media (max-width: 767px) .container {  
  flex-direction: column;  
}
```

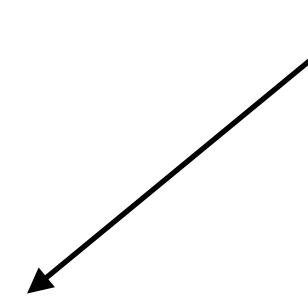
Comparable à une sur-qualification  
du type



```
.container {  
  display: flex;  
  flex-direction: row;  
}  
...  
  
body .container {  
  flex-direction: column;  
}
```

# Media queries - Tags HTML CSS

Accepte des media queries



```
<link rel="stylesheet" media="screen and (min-width: 1201px)" href="widescreen.css">
<link rel="stylesheet" media="screen and (max-width: 600px)" href="smallscreen.css">
```



**Avantages ? Inconvénients ?**

# Media queries - Breakpoints

## CSS

- **320px – 480px** : Mobile devices
- **481px – 768px** : iPads, Tablets
- **769px – 1024px** : Small screens, laptops
- **1025px – 1200px** : Desktops, large screens
- **1201px and more** : Extra large screens, TV

# Media queries - Javascript

## CSS

### Tester une media query

```
const mql = window.matchMedia("(orientation: portrait)");

if (mql.matches) {
  /* La zone d'affichage/viewport est en portrait */
} else {
  /* La zone d'affichage/viewport est en paysage */
}
```

# Media queries - Javascript

## CSS

### Ajouter un listener sur une media query

```
const mql = window.matchMedia("(orientation: portrait)");
mql.addListener(handleOrientationChange);

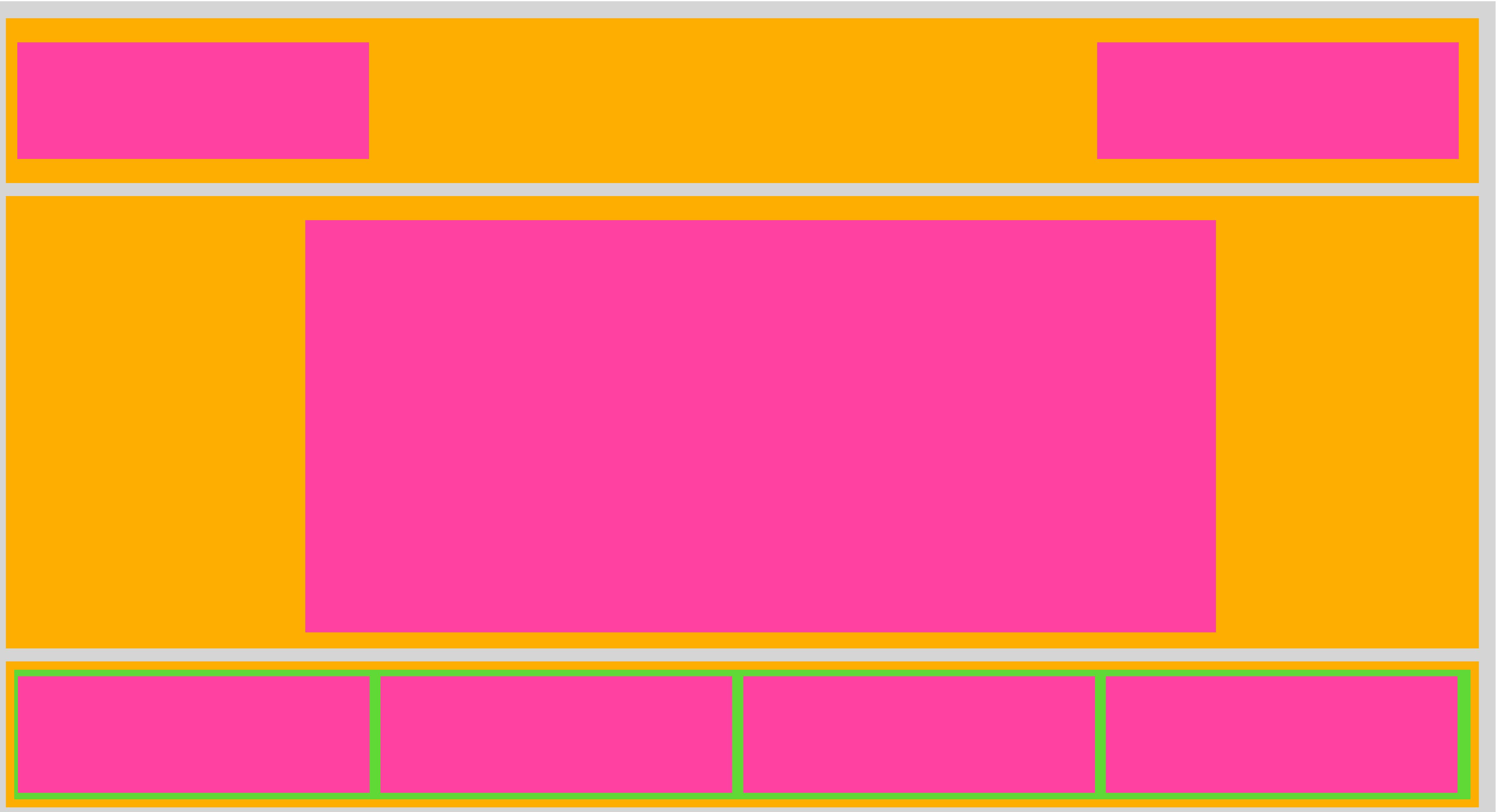
function handleOrientationChange(mql){
    if (mql.matches) {
        /* La zone d'affichage/viewport est en portrait */
    } else {
        /* La zone d'affichage/viewport est en paysage */
    }
}
```

# Composants ?

## Projet

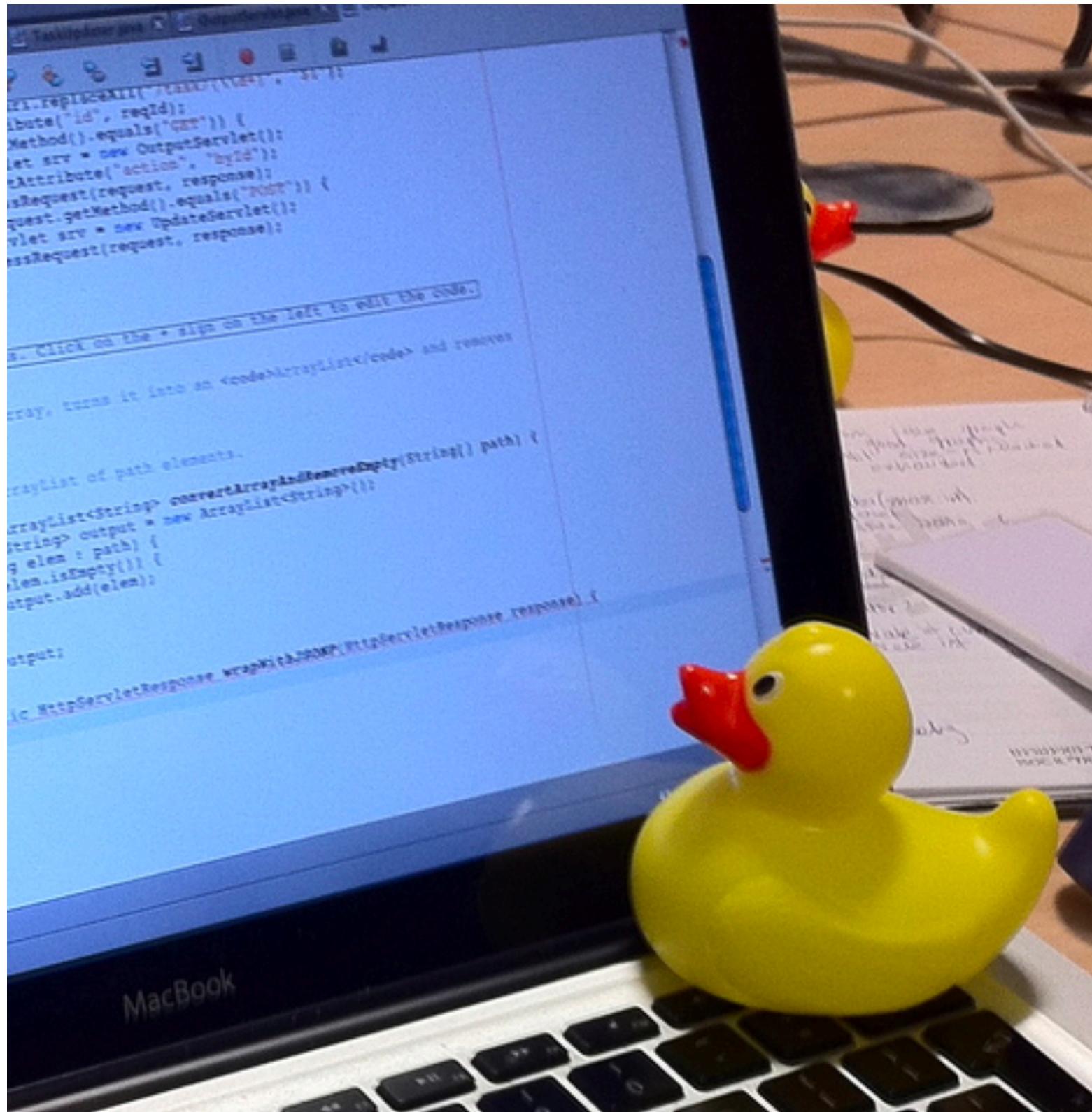
- ✓ Projet webpack vide
  - Popover pour playlists
  - Détection online/offline
  - Local storage pour les playlists
  - Manifest PWA
  - Caching
  - Service worker
- ✓ Squelette HTML
- ✓ Styles CSS structurels
- ✓ Icônes
- Routeur pour les pages web
- Client pour l'API JSON
- Lecteur audio

# Code



# Rubber duck debugging

## Tips & tricks



[https://fr.wikipedia.org/wiki/M%C3%A9thode\\_du\\_canard\\_en\\_plastique](https://fr.wikipedia.org/wiki/M%C3%A9thode_du_canard_en_plastique)

# JS

# Single Page Application

## JS

- Une Single Page Application est une application web qui réside sur une seule page HTML
- Le javascript va gérer l'affichage des différents éléments et non les pages HTML
- Typiquement utilisé pour des applications PWA

# Single Page Application

## JS

Comment renderer  
les différentes pages ?

# Single Page Application

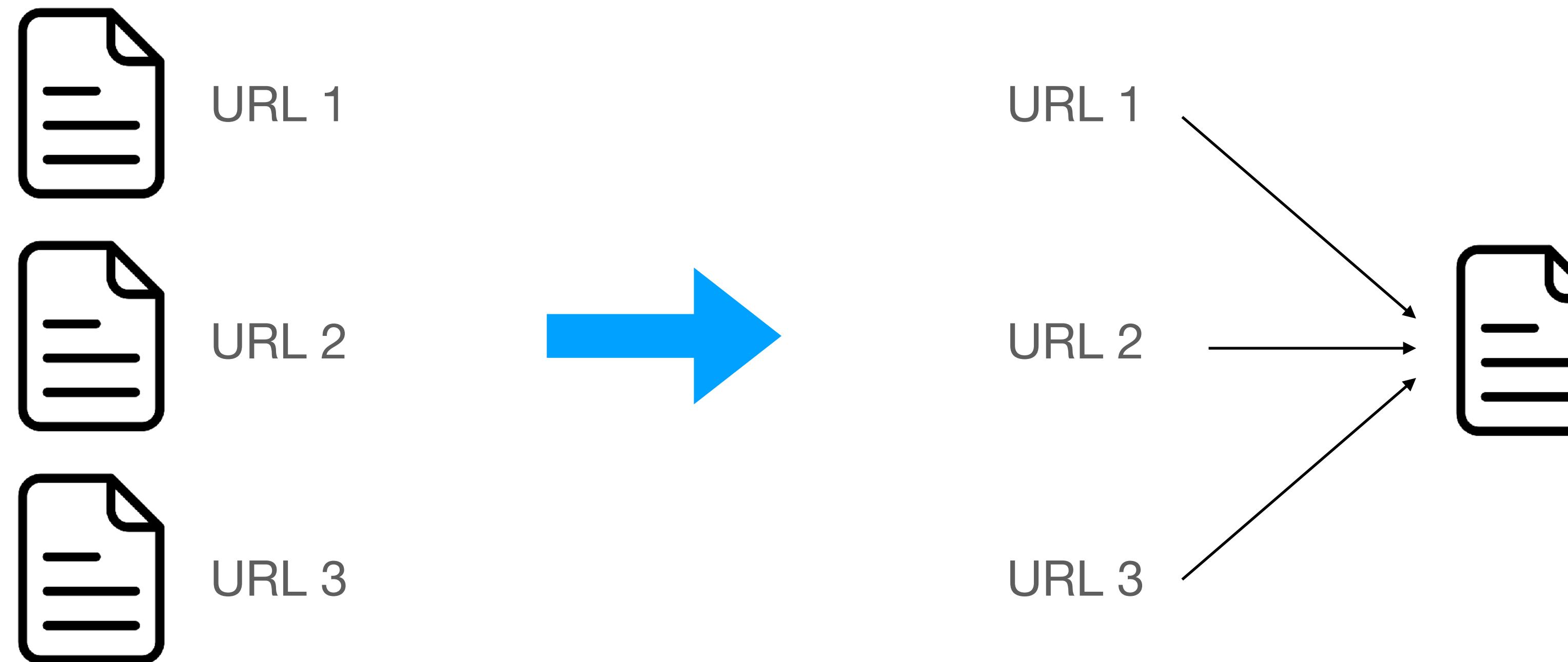
## JS

Deux options principales :

- Les vues ou pages sont existantes dans le DOM et sont affichées/cachées par CSS (`display: none`)
- Les vues ou pages sont à chaque fois renderée au complet par le javascript. Typiquement un custom element ou une classe javascript avec une méthode `render`

# Single Page Application - Routing

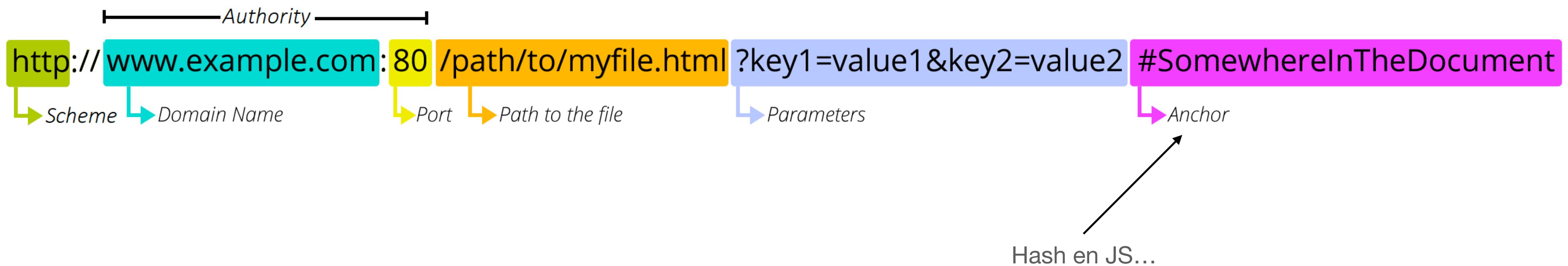
## JS



# Single Page Application - Routing JS

Comment géré l'état  
de la page en cours?

# Single Page Application - Think RESTful... JS



[https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/What\\_is\\_a\\_URL](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL)

# Single Page Application - Think RESTful... JS

http://localhost:8080/hello?something=bonjour#quelquechose

> window.location

=>

```
hash => "#quelquechose"
host => "localhost:8080"
hostname => "localhost"
href => "http://localhost:8080/hello?something=bonjour#quelquechose"
origin => "http://localhost:8080"
pathname => "/hello"
port => "8080"
protocol => "http:"
search => "?something=bonjour"
```

# Single Page Application - Routing JS

Deux options principales :

- Utiliser les Anchors ou Hash
- Utiliser l'API history et la réécriture d'URL sur le path

# Single Page Application - Anchors/Hash JS

- Historiquement, les anchors (<a />) sont des ancrés dans la page
- Une manière de mettre des liens internes à la page pour avancer dans le contenu
- Single page by design -> ne refresh pas la page lorsqu'on clique dessus  
-> parfait pour notre app !

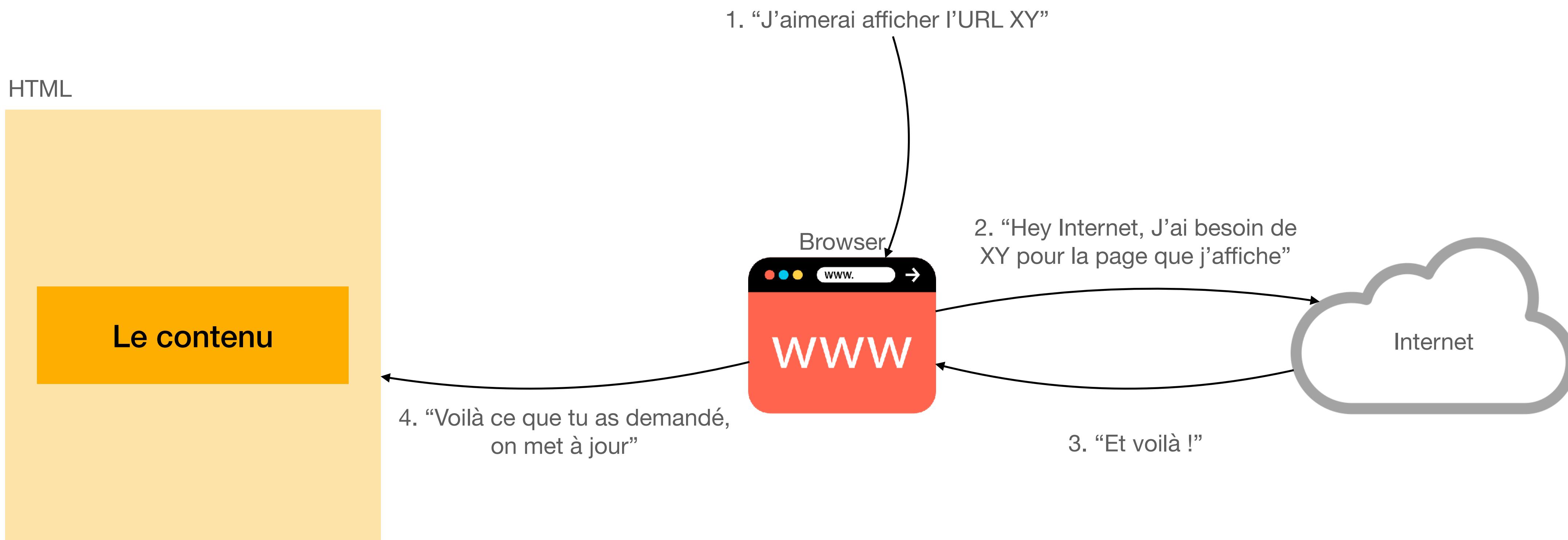
# Single Page Application - Anchors/Hash JS

- Possible par exemple de prendre la structure d'URL suivante :
  - /#home
  - /#player
  - /#artists
  - /#playlists-12



Avantages ? Inconvénients ?

# Changement de section - Chargement de la page JS



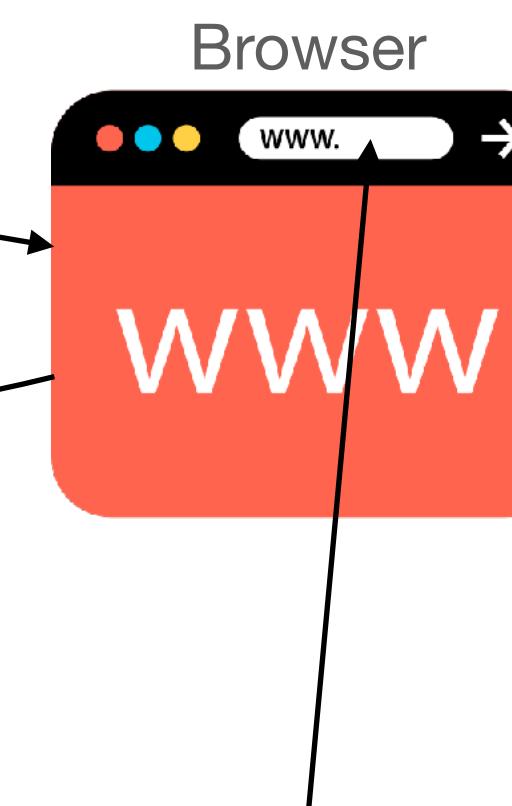
# Changement de section - Lien classique

## JS

HTML

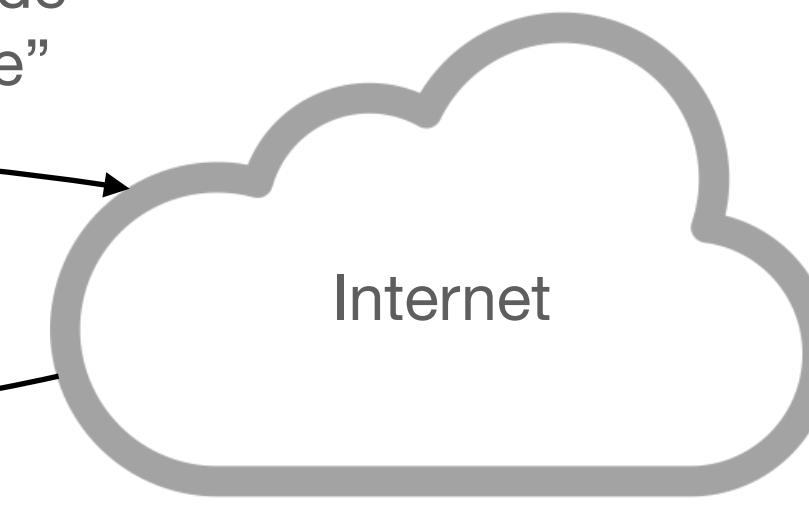


1. "J'ai été cliqué !  
Voilà ce que je demande..."



4. "On a changé de place,  
j'affiche une autre URL"

2. "Hey Internet, J'ai besoin de  
XY pour la page que j'affiche"



3. "Et voilà !"

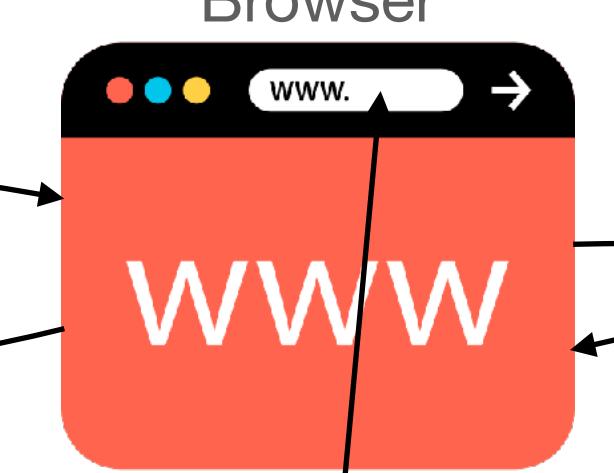
5. "Voilà ce que tu as demandé,  
on met à jour"

# Changement de section - Lien avec hash JS

HTML



1. "J'ai été cliqué !  
Voilà ce que je demande..."



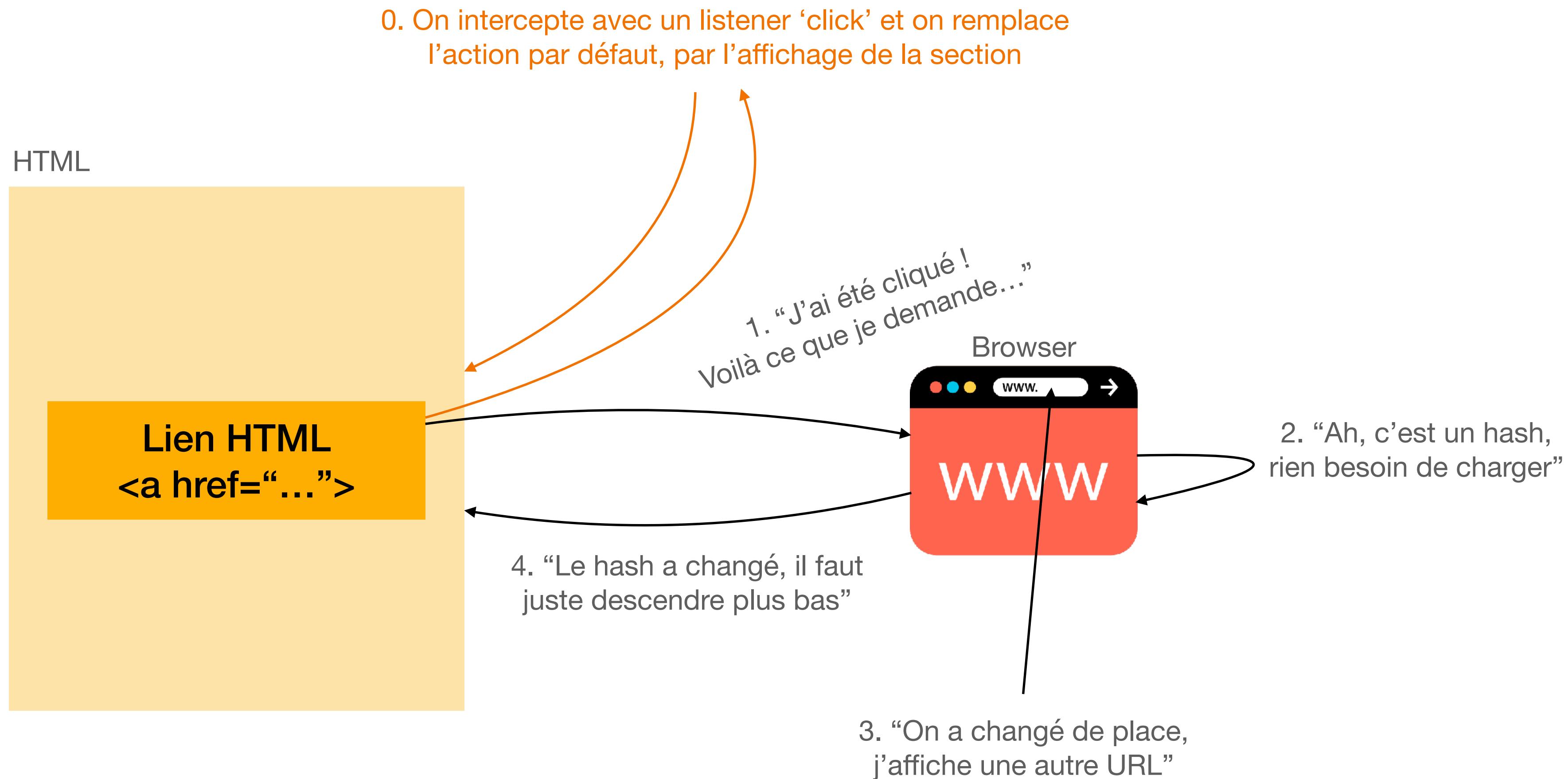
2. "Ah, c'est un hash,  
rien besoin de charger"

3. "On a changé de place,  
j'affiche une autre URL"

4. "Le hash a changé, il faut  
juste descendre plus bas"

# Changement de section - Comment ? V1

## JS



# Changement de section - Comment ? V1

## JS

### Manipulation des URLs

- Le browser nous permet de modifier l'historique de navigation manuellement
- Possible de le contrôler (précédent/suivant)
- Ajouter une entrée dans l'historique
- Remplacer une entrée
- Être informé d'un changement d'état

# Changement de section - Comment ? V1

## JS

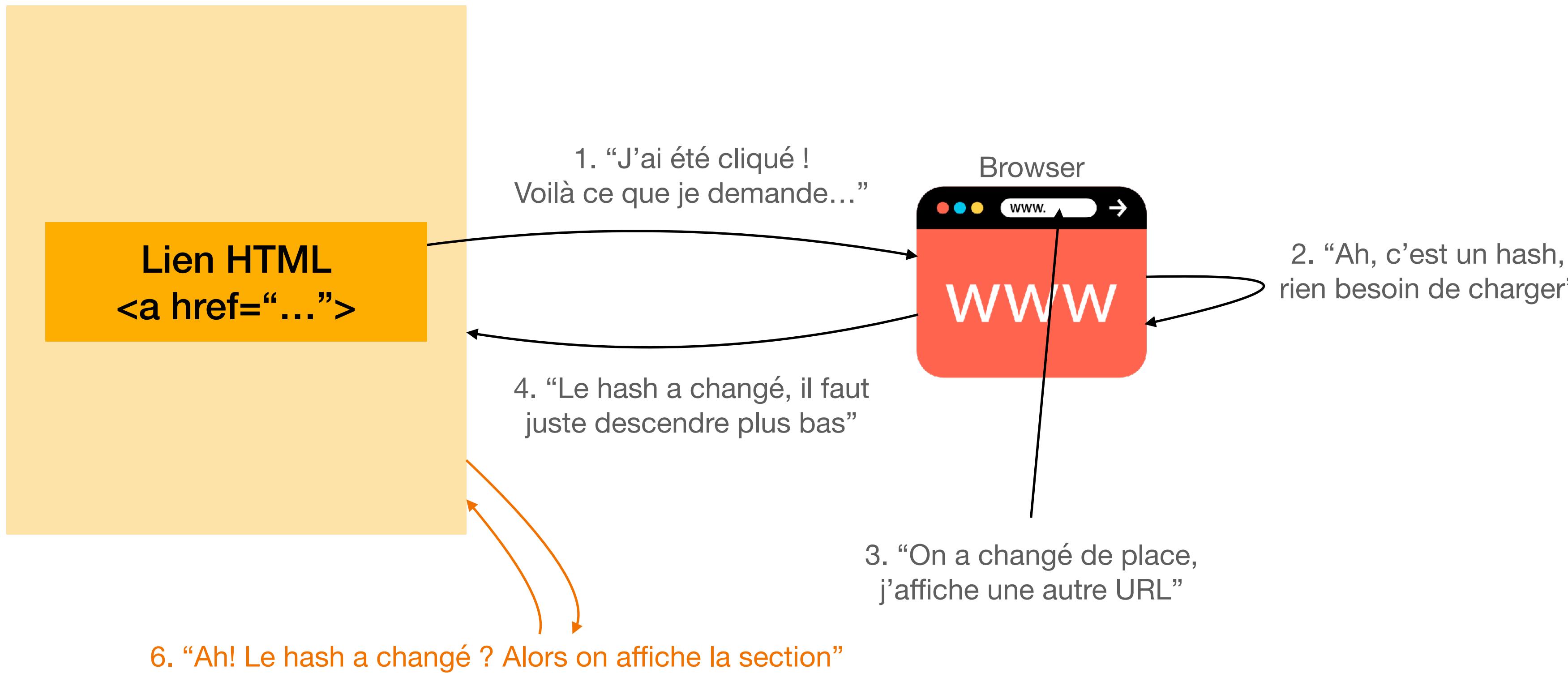
### L'API History

- `history.go(entier)` ou `history.forward()`/`back()`
- `history.pushState(état, titre, url)`
- `history.replaceState(état, titre, url)`
- L'événement `popstate` sur `window`

# Changement de section - Comment ? V2

## JS

HTML



# V1

[https://developer.mozilla.org/fr/docs/Web/API/History API](https://developer.mozilla.org/fr/docs/Web/API/History_API)

history...

# V2

[https://developer.mozilla.org/fr/docs/Web/API/WindowEventHandlers/  
onhashchange](https://developer.mozilla.org/fr/docs/Web/API/WindowEventHandlers/onhashchange)

window.addEventListener("hashchange", () => { ... })

ou

window.addEventListener("popstate", () => { ... })

# Changement de section - Comment ?

JS

- Garder en tête que les boutons précédent/suivant du navigateur doivent fonctionner
- Ouvrir le lien dans un nouvel onglet doit fonctionner également
- `window.location` vous donne toutes les infos sur l'URL en cours

**GO!**

# Architecture de code

# Modularité et responsabilité

## Architecture de code

- Une architecture est dite “modulaire” quand elle est séparée en plusieurs modules, avec des responsabilités précises
- Chaque module a une implémentation privée qui lui est propre et une implémentation publique pour interagir avec les autres modules
- Chaque module a des voisins directs, avec qui il a une forte interaction, et d'autres voisins indirects avec lesquels il échange par le biais d'autres modules
- Cela s'applique aussi bien à une vue globale qu'à une vue détaillée

# Modularité et responsabilité - Exemple

## Architecture de code



- Le CSS est responsable de la mise en page
- Il offre le langage CSS comme implémentation publique (on lui dit quoi faire)
- Il a comme voisin direct le HTML, car il met en page des éléments du langage HTML
- Le HTML est responsable de la structure de la page
- Il offre le langage HTML comme implémentation publique
- Il a comme voisin direct le CSS, car il lui fournit les éléments à mettre en page
- Le JS comme autre voisin direct, car il interagit avec les éléments HTML
- Le CSS est responsable de la dynamique de la page
- Il offre le langage JS comme implémentation publique
- Il a comme voisin direct le HTML, car il utilise les éléments HTML pour les rendre dynamique
- Le CSS n'est pas un voisin direct, car le JS va d'abord utiliser un élément HTML pour y ajouter des styles

# **Loi de Déméter - Principe de connaissance minimale**

## **Architecture de code**

*« Ne parlez qu'à vos amis immédiats ».*

# **Loi de Déméter - Principe de connaissance minimale**

## **Architecture de code**

- La loi de Déméter vise à limiter les connaissances de chaque module
- Le but est de diminuer les dépendances et donc la complexité
- Cela permet une plus grande flexibilité et une notion d'agilité

# Loi de Déméter - Principe de connaissance minimale

## Architecture de code



- Un client qui aurait besoin d'un conseiller au sein d'une entreprise, appelle la réception qui va l'aiguiller vers la personne adaptée à sa demande
- Le client ne se soucie pas de la liste des employés, leur présence, changements, etc...

# Loi de Déméter - Dans la pratique

## Architecture de code

On souhaite afficher une <section>



```
section {  
  /* display: flex */  
  display: none;  
}
```

```
<section id="ma-section">  
  ...  
</section>
```

```
const section = document.querySelector('#ma-section')  
section.style.display = 'flex'
```

Pas très “Déméter”... Le JS va directement modifier le CSS.  
Que se passe-t-il si cette section devient un block et plus un flex ?

# Loi de Déméter - Dans la pratique

## Architecture de code

On souhaite afficher une <section>



```
section {  
  display: none;  
}  
  
section.active {  
  display: flex;  
}
```

```
<section id="ma-section">  
  ...  
</section>
```

```
const section = document.querySelector('#ma-section')  
section.classList.add('active')
```



Mieux ! Le JS ne fait qu'ajouter un attribut HTML (une classe) et il ne s'occupe pas du mode d'affichage.  
C'est le CSS qui va gérer le changement -> Les responsabilités sont respectées !

# Loi de Déméter - Dans la pratique 2

## Architecture de code

Taille du texte en CSS



- S'occupe d'afficher le player
- La taille exacte du texte ne l'intéresse pas !
- S'occupe de gérer la taille du texte pour le monde
- Le player ne l'intéresse pas !

# Loi de Déméter - Changement de section

## Architecture de code



Plutôt **V1** ou **V2** ?

# Structure de l'application

## Architecture de code



Ou d'autres dans l'application ?

# Schémas UML

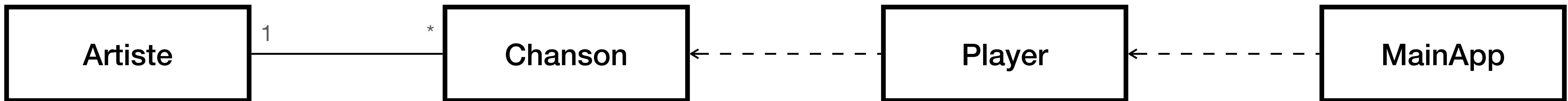
## Architecture de code

- Les schémas UML (**Unified Modeling Language**) sont des schémas standardisés pour représenter une application
- Il en existe plus d'une vingtaine...
- Nous allons utiliser les diagrammes de classe

# Schémas UML - Diagramme de classe

## Architecture de code

- Chaque classe (ou modèle) est représenté par un rectangle avec le nom de la classe
- Optionnellement, la liste de ses attributs et de ses fonctions
- Les éléments sont connectés ensemble par plusieurs types de flèches, représentant le type d'interaction (héritage, composition, ...)



# Spotlified

## Architecture de code



Quels sont les éléments importants de l'application ?

# JSON

# What is it?

## JSON

*“Le JavaScript Object Notation (JSON) est un format standard utilisé pour représenter des données structurées de façon semblable aux objets Javascript.”*

# What is it?

## JSON

- JSON est un format de données, fortement inspiré du JS
- Il dispose de sa propre syntaxe et est indépendant du Javascript
- La plupart des langages de programmation fournissent des librairies pour permettre de le convertir en des objets utilisables ou d'en générer pour l'exporter au sein d'un autre langage
- JSON se stock dans des chaînes de caractères

# Quand l'utiliser ?

## JSON

- Idéalement, lorsque deux systèmes séparés doivent s'échanger des informations (par ex. une API entre Javascript et PHP)
- Quand il faut stocker des données structurées en dehors du langage lui-même (par ex. enregistrer un tableau dans un fichier texte)

# Usecase

## JSON



# Exemple JSON

```
{  
  "squadName": "Super hero squad",  
  "homeTown": "Metro City",  
  "formed": 2016,  
  "secretBase": "Super tower",  
  "active": true,  
  "members": [  
    "Molecule Man",  
    "Madame Uppercut"  
  ]  
}
```

# Comment l'utiliser

## JSON

- JSON supporte les types de données standards, comme:
  - Chaîne de caractères
  - Nombres
  - Null
  - Booléen
- Il y a également deux types de données structurées :
  - Tableau
  - Objet (clé/valeur, comme en js)

# Comment l'utiliser

## JSON

Deux méthodes principales :

- `JSON.parse("...")`  
Converti une chaîne de caractères JSON en un objet javascript
- `JSON.stringify(unevariable)`  
Converti une variable Javascript en un objet JSON

# Comment l'utiliser - JSON.parse

## JSON

```
const monJson = '{ "cours": "WebmobUI", "lieu": "HEIG-VD"}'
```

```
const monJsonparsed = JSON.parse(monJson)
```

```
console.log(monJsonparsed.cours) => "WebmobUI"
```

```
console.log(monJsonparsed.lieu) => "HEIG-VD"
```

# Comment l'utiliser - JSON.stringify

## JSON

```
const monObjet = { cours: 'WebmobUI' , lieu: 'HEIG-VD' }
```

```
console.log(monObjet.cours) => "WebmobUI"
```

```
console.log(monObjet.lieu) => "HEIG-VD"
```

```
const monJson = JSON.stringify(monObjet)
```

```
console.log(monJson) => '{ "cours": "WebmobUI" , "lieu": "HEIG-VD" } '
```

# L'API Spotlified

# Concept

## L'API Spotlified

- L'API Spotlified tourne sur un serveur distant:  
<https://webmob-ui-22-spotlified.herokuapp.com/>
- Tous les endpoints retournent du JSON qui sera à “parser” par vos soins
- Les URLs sont dites “REST” pour plus de clarté

# **Endpoints**

## **L'API Spotlified**

3 endpoints principaux :

- Lister les artistes
- Listes les chansons d'un artiste
- Rechercher une chanson par texte libre

# Endpoints - Lister les artistes

## L'API Spotlified

URL: <https://webmob-ui-22-spotlified.herokuapp.com/api/artists>

Response: Tableau d'artistes

Exemple:

[

{ "id": 2, "name": "Alan Walker", image\_url: "https://...." },

{ "id": 3, "name": "Dynoro", image\_url: "https://...." }

]

# Endpoints - Lister les chansons d'un artiste

## L'API Spotlified

URL: <https://webmob-ui-22-spotlified.herokuapp.com/api/artists/:id/songs>

Response: Tableau de chanson, avec l'artiste dans la chanson

Exemple:

[

```
{ "id": 2, "title": "Faded", "audio_url": "https://....", "artist": { ... }},
```

```
{ "id": 3, "title": "Spectre", "audio_url": "https://....", "artist": { ... }}
```

]

# **Endpoints - Rechercher une chanson par texte libre**

## **L'API Spotlified**

URL: <https://webmob-ui-22-spotlified.herokuapp.com/api/songs/search/:query>

Response: Tableau de chanson, avec l'artiste dans la chanson

Exemple:

[

```
{ "id": 2, "title": "Faded", "audio_url": "https://....", "artist": { ... }},
```

```
{ "id": 3, "title": "Spectre", "audio_url": "https://....", "artist": { ... }}
```

]

# API Client

## L'API Spotlified



Comment structurer l'utilisation de l'API en mode Déméter ?

# API fetch

# API Fetch

## API Fetch

- L'API fetch permet de charger du contenu depuis une URL
- Elle est dite asynchrone
- Asynchrone signifie que le code suivant l'appel à fetch continuera son exécution sans n'avoir encore de résultat
- L'API fetch est basé sur les Promises
- Il est possible de synchroniser partiellement des promises en utilisant les mots-clés `async / await`

# API Fetch

## API Fetch

- L'API fetch retourne une promise
- Par défaut, la fonction passée à then prend un argument : Response
- Response est la réponse reçue par le navigateur, sous forme d'objet, avec différentes méthodes et attributs
- Par ex:  
`response.ok => true|false`  
`response.json() => retourne le body de la réponse, déjà parsé par JSON`

# Promises

## API Fetch

- Les Promises sont des promesses. Leurs appels vous promettent d'obtenir une réponse en retour, dans un avenir proche
- Une promesse peut être soit réalisée (resolved) ou rejetée (rejected)
- Vous avez la possibilité de définir les actions à effectuer, selon sa réalisation ou son rejet
- Vous utilisez les promesses dans la vie de tous les jours...

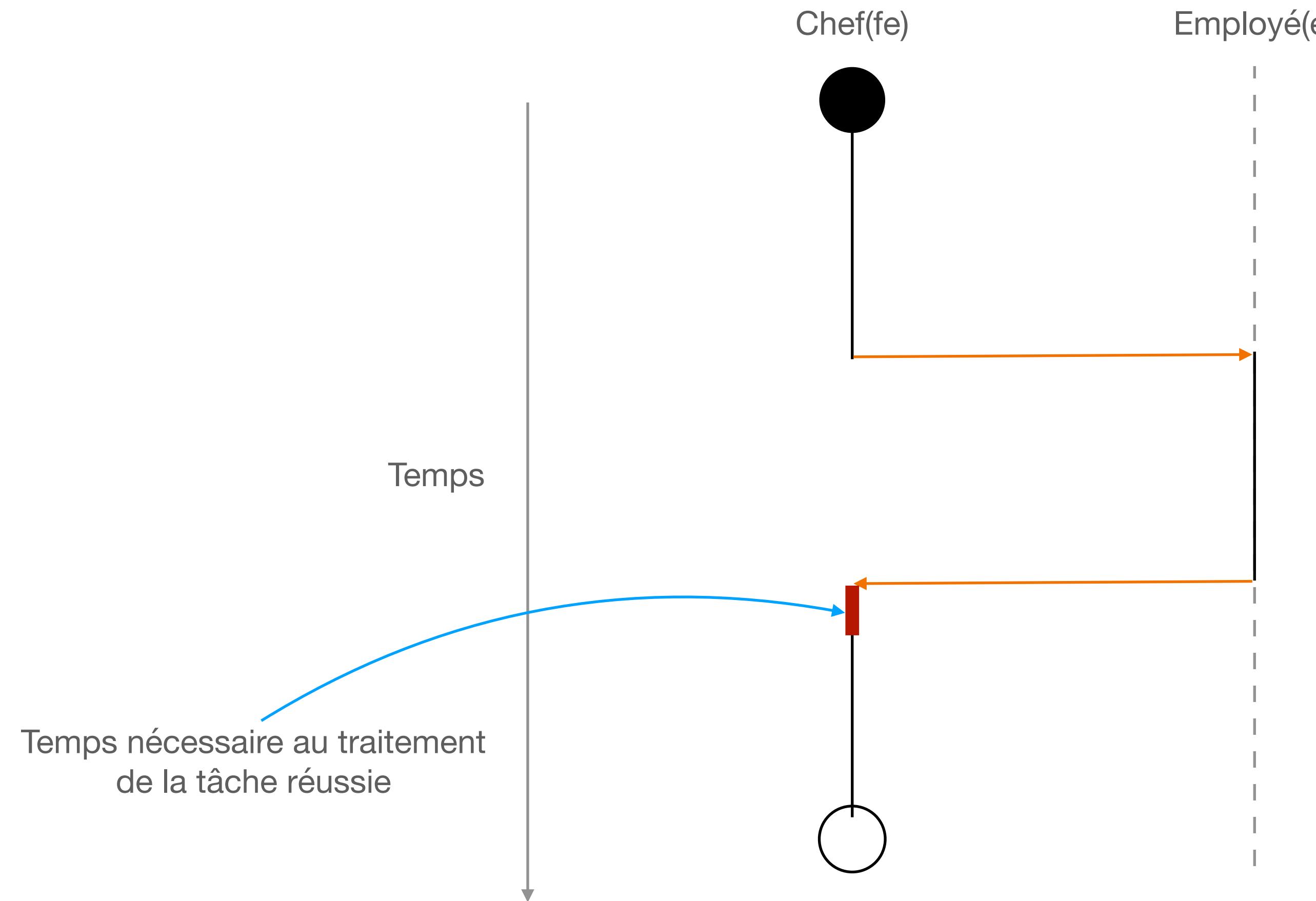
# Promises - Exemple

## API Fetch

- Supposons que vous êtes chef(fe) de service et que vous donnez une tâche à l'un(e) de vos employé(e)s
- Vous retournez ensuite à votre bureau, continuer votre travail jusqu'à ce que la personne vienne toquer à votre porte, dossier en main, pour vous signaler que la tâche est terminée
- Même constat lorsque vous commandez un colis, vous ne restez pas devant votre boite-aux-lettres, jusqu'à avoir reçu celui-ci. Il arrive quand il arrive.

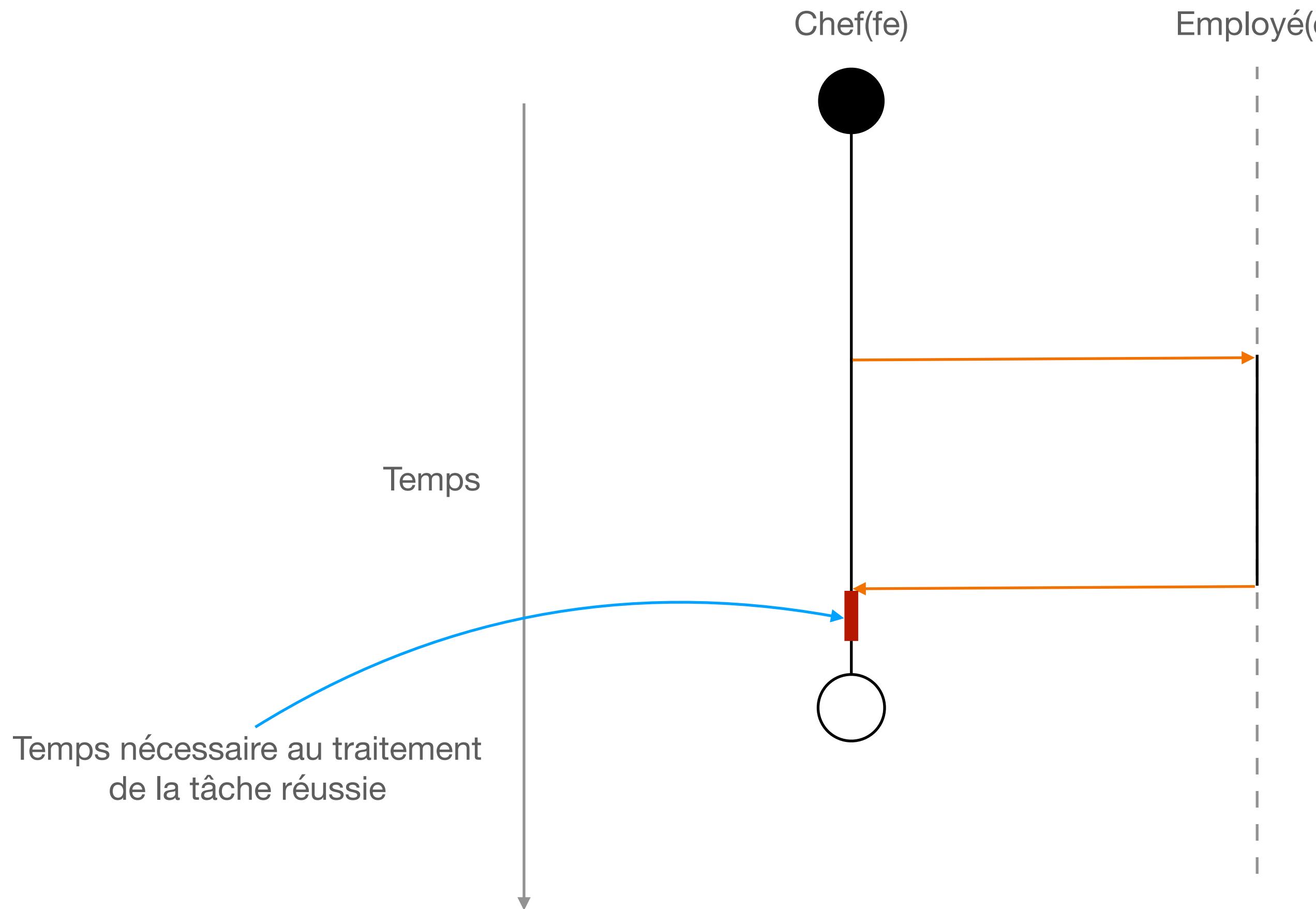
# Promises - Code synchrone

## API Fetch



# Promises - Code asynchrone

## API Fetch

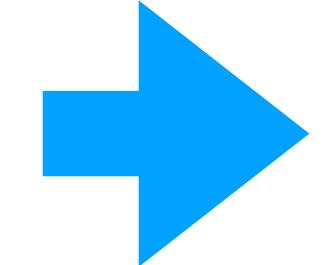


# Promises - Synchrone

## API Fetch

Prenons l'exemple de code suivant...

```
console.log('Hello 1')  
uneFonctionSynchroneClassiqueQuiAfficheHello2()  
console.log('Hello 3')
```



Console

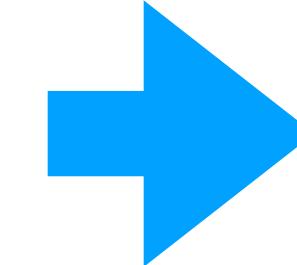
```
Hello 1  
Hello 2  
Hello 3
```

# Promises - Asynchrone

## API Fetch

Que se passe-t-il avec une méthode asynchrone ?

```
console.log('Hello 1')  
unePromiseAsynchroneQuiAfficheHello2()  
console.log('Hello 3')
```



Console

Hello 1

Hello 2

Hello 3

Console

Hello 1

OU

Hello 3

Un des deux... On ne sait pas...



Comment s'assurer que "Hello 3" ne sera affiché qu'après "Hello 2" ?

# Promises - then/catch/finally

## API Fetch

- Il existe trois types de méthodes utilisables sur une Promise et prennent toute une fonction en paramètre:
  - then - “Ensuite” - La fonction passée sera appelé lorsque tout se passe bien
  - catch - La fonction passée sera appelé lorsqu'il y a une erreur
  - finally - La fonction passée sera appelé dans les deux cas

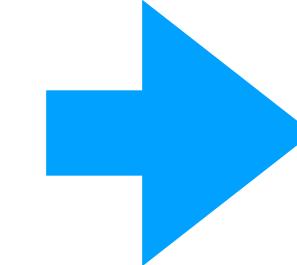
# Promises - then/catch/finally

## API Fetch

```
console.log('Hello 1')  
unePromiseAsynchroneQuiAfficheHello2()  
console.log('Hello 3')
```



Extraire le code devant s'exécuter après, dans une fonction, au sein d'un "then"



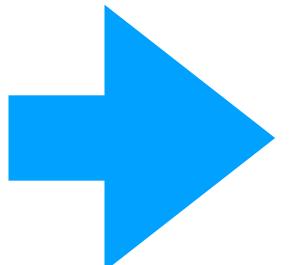
```
console.log('Hello 1')  
unePromiseAsynchroneQuiAfficheHello2()  
.then(() => console.log('Hello 3'))
```

Console

Hello 1

Hello 2

Hello 3

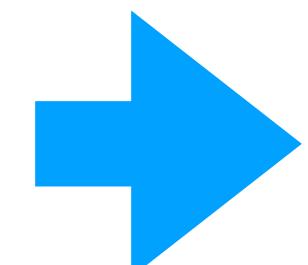


# Promises - then/catch/finally

## API Fetch

- Les promises permettent de structurer le code de manière claire, en donnant des instructions précises, selon le déroulement des événements
- Elles permettent surtout de ne pas bloquer l'exécution de la page (par exemple pendant le chargement de la liste des artistes) et d'avoir une expérience plus fluide

```
console.log('Hello 1')  
  
unePromiseAsynchroneQuiAfficheHello2()  
  
.then(() => console.log('Hello 3'))
```



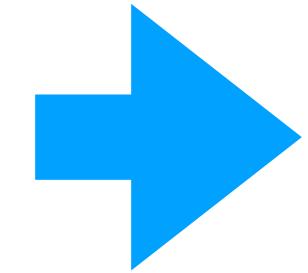
```
AfficheHello1()  
  
AfficheHello2()  
  
.ensuite(() => afficheHello3())
```

# Promises - Chaînage

## API Fetch

- Les promises sont construites sur le concept du chaînage
- Il est possible de mettre plusieurs then/catch/finally à la suite

```
console.log('Hello 1')  
unePromiseAsynchroneQuiAfficheHello2()  
.then(() => console.log('Hello 3'))  
.then(() => console.log('Hello 4'))  
.catch(() => console.log('Erreur !'))  
.finally(() => console.log('Terminé'))
```



```
AfficheHello1()  
AfficheHello2Asynchrone()  
.ensuite(() => afficheHello3())  
.ensuite(() => afficheHello4())  
.siErreur(() => afficheErreur())  
.quoiQuIlArrive(() => afficheTerminé())
```

# Promises - Chaînage

## API Fetch

- Le chaînage permet également de transformer l'information au fur et à mesure et la passer à l'étape suivante
- Chaque étape prend en argument la valeur de retour de la fonction précédente

# Promises - Chaînage

## API Fetch

- Supposons que l'on souhaite charger les artistes et afficher le premier artiste...

```
fetch('http.../api/artists') // Va charger les artistes sur le serveur et retourne la réponse par défaut
  .then((response) => {
    const artistes = response.json() // On prend la réponse de base et on la converti en JSON.
                                    // Cela retournera un tableau d'artistes
    const artist = artistes[0] // On prend le premier élément du tableau artistes
    console.log(artist) // On affiche le premier artiste
  })
```

# Promises - Chaînage

## API Fetch

- Le code suivant serait équivalent !

```
fetch('http.../api/artists') // Va charger les artistes sur le serveur et retourne la réponse par défaut
  .then((response) => response.json()) // On prend la réponse de base et on la converti en JSON.
    // Cela retournera un tableau d'artistes
  .then((artists) => artists[0]) // On prend le tableau retourné par la méthode précédente et
    // on retourne le premier artiste
  .then((artist) => console.log(artist)) // On affiche le premier artiste retourné par la méthode précédente
```

# Promises - Chaînage

## API Fetch

- Et avec les autres méthodes ?

```
afficherRondDeChargement()

fetch('http.../api/artists')

.then((response) => response.json())

.then((artists) => artists[0])

.then((artist) => console.log(artist))

.catch(() => alert('Il y a eu un problème avec le serveur !')) // On attrape l'erreur du fetch et on affiche un message

.finally(() => cacherRondDeChargement()) // Succès ou erreur, on cache le rond de chargement, car la promise est terminée
```

# Promises - Langage fonctionnel

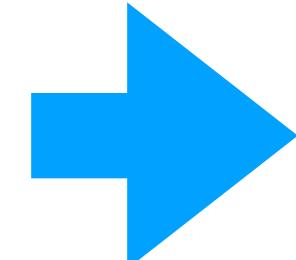
## API Fetch

- Quel intérêt d'utiliser autant de fonctions ?
- Javascript est un langage “Fonctionnel” -> Basé sur les fonctions et est hautement performant dans leur gestion
- Il n'est pas obligatoire d'utiliser des fonctions fléchées dans les then. N'importe quelle référence vers une fonction est acceptée
- Plus l'on sépare le code en fonctions, plus celui-ci sera clair et concis... et plus notre ami Déméter sera content.

# Promises - Langage fonctionnel

## API Fetch

```
AfficheHello1()  
  
AfficheHello2()  
  
.ensuite(() => afficheHello3())  
  
.ensuite(() => afficheHello4())  
  
.siErreur(() => afficheErreur())  
  
.quoiQuIlArrive(() => afficheTerminé())
```



```
AfficheHello1()  
  
AfficheHello2()  
  
.ensuite(afficheHello3)  
  
.ensuite(afficheHello4)  
  
.siErreur(afficheErreur)  
  
.quoiQuIlArrive(afficheTerminé)
```

# Promises - Langage fonctionnel

## API Fetch

```
// section_artistes.js

import { chargerArtistes } from 'api.js'

// api.js

function chargerArtistes() {
    return fetch('http.../api/artists')
        .then((response) => response.json())
}

function afficherArtistes(artistes) {
    for(const artiste of artistes){
        ...
    }
}

function afficherSectionArtistes() {
    chargerArtistes().then(afficherArtistes)
}
```

# Promises - Langage fonctionnel

## API Fetch

Faisons plaisir à Déméter...

```
// api.js

function fetchJson(url) {
  return fetch(url)
    .then((response) => response.json())
}

function chargerArtistes() {
  return fetchJson('http.../api/artists')
}

// section_artistes.js

import { chargerArtistes } from 'api.js'

function afficherArtiste(artiste) {
  ...
}

function afficherArtistes(artistes) {
  artistes.forEach(afficherArtiste)
  // ou
  for(const artiste of artistes){
    afficherArtiste(artiste)
  }
}

function afficherSectionArtistes() {
  chargerArtistes().then(afficherArtistes)
}
```

# Promises - Rendre synchrone

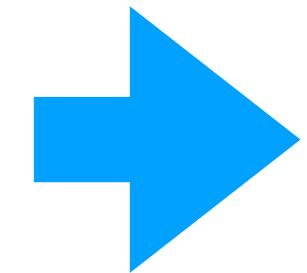
## API Fetch

- Il est possible de rendre des promises (presque) synchrones en utilisant les mots-clés async/await
- Le mot clé await mis devant l'appel à une promise la rend synchrone et permet de l'utiliser comme une fonction classique  
`const résultat = await mapromise()`
- “Presque”, car await ne peut être utilisé seul. Il doit obligatoirement être utilisé au sein d'une fonction async (donc au sein d'une promise)
- Cette action consiste simplement à remonter la promise d'un niveau

# Promises - Rendre synchrone

## API Fetch

```
// fichier.js  
  
const résultat = await fetch('...')
```



Erreur ! Pas possible au root d'un fichier

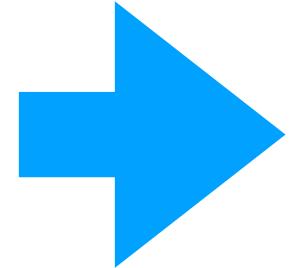
```
// fichier.js  
  
async function chargerSyncrone() {  
  
    const résultat = await fetch('...')  
  
}  
  
chargerSyncrone()
```

Possible, car englobé dans une fonction async

# Promises - Rendre synchrone

## API Fetch

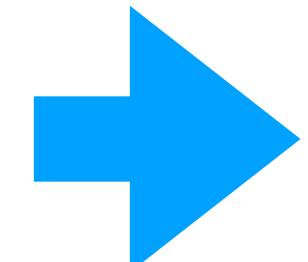
```
console.log('Hello 1')  
unePromiseAsynchroneQuiAfficheHello2()  
console.log('Hello 3')
```



```
async function afficherHellos() {  
    console.log('Hello 1')  
    await unePromiseAsynchroneQuiAfficheHello2()  
    console.log('Hello 3')  
}  
  
afficherHellos()
```

Console

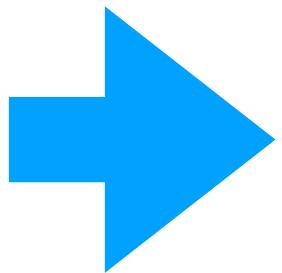
```
Hello 1  
Hello 2  
Hello 3
```



# Promises - Rendre synchrone

## API Fetch

```
console.log('Hello 1')
unePromiseAsynchroneQuiAfficheHello2()
  .then(() => console.log('Hello 3'))
  .then(() => console.log('Hello 4'))
  .catch((e) => console.log('Erreur !', e))
  .finally(() => console.log('Terminé'))
```



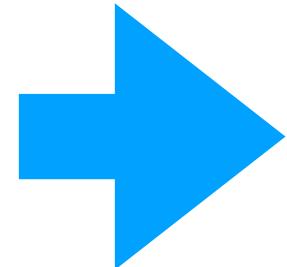
```
async function afficherHellos() {
  try {
    console.log('Hello 1')
    await unePromiseAsynchroneQuiAfficheHello2()
    console.log('Hello 3')
    console.log('Hello 4')
  } catch (e) {
    console.log('Erreur !', e)
  } finally{
    console.log('Terminé')
  }
}
```

```
afficherHellos()
```

# Promises - Rendre synchrone

## API Fetch

```
fetch('http.../api/artists')
  .then((response) => {
    const artistes = response.json()
    const artist = artistes[0]
    console.log(artist)
  })
}
```



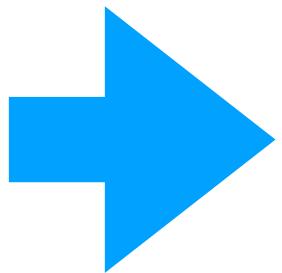
```
async function afficherArtiste() {
  const response = await fetch('http.../api/artists')
  const artistes = await response.json()
  const artist = artistes[0]
  console.log(artist)
}

afficherArtiste()
```

# Promises - Rendre synchrone

## API Fetch

```
afficherRondDeChargement()
fetch('http.../api/artists')
  .then((response) => response.json())
  .then((artists) => artists[0])
  .then((artist) => console.log(artist))
  .catch((e) => alert('Il y a eu un problème!'))
  .finally(() => cacherRondDeChargement())
```



```
async function afficherArtiste() {
  afficherRondDeChargement()

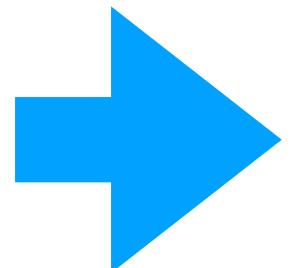
  try {
    const response = await fetch('http.../api/artists')
    const artistes = response.json()
    const artist = artistes[0]
    console.log(artist)
  } catch (e) {
    alert('Il y a eu un problème!')
  } finally {
    cacherRondDeChargement()
  }
  afficherArtiste()
```

# Promises - Async

## API Fetch

- Au fond, à quoi sert le mot clé async ?
- Il converti une fonction en promise... Promiseception...

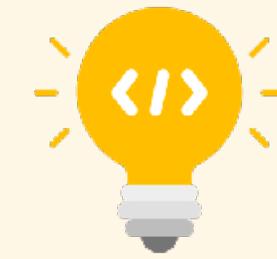
```
function afficherArtiste() {  
    ...  
}  
  
afficherArtiste().then(...)  
  
Erreur ! C'est une fonction classique
```



```
async function afficherArtiste() {  
    ...  
}  
  
afficherArtiste().then(...)  
  
Possible ! afficherArtiste est devenu une promise...
```

# Promises - Async/await

## API Fetch



then ou async/await ?

# Templating

# Concept

## Templating

- Le templating permet de définir un squelette de base à utiliser pour des éléments dynamiques
- Exemple: Un élément `<li>` dans la liste des artistes
- Il suffit ensuite de dupliquer cet élément vide autant de fois que nécessaire pour afficher la liste complète

# Deux écoles

## Templating

- Utiliser le templating manuel - Garder un élément vide, le cloner, modifier son DOM et l'insérer dans l'élément parent
- Utiliser un moteur de template - Un markup spécifique, interprété par une librairie qui gère le remplacement des zones à éditer et va l'intégrer dans l'élément parent  
Exemple: Handlebars, JSX, ...

# **Templating manuel**

## **Templating**

- Nous allons utiliser le templating manuel pour la première partie du cours (Spotlified)
- Moteur de template en partie 2, avec les frameworks javascript

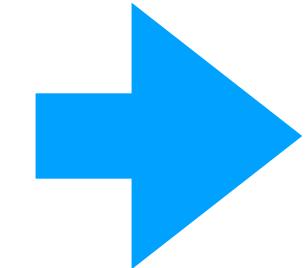
# Templating manuel

## Templating

- Où stocker les éléments vides ?
- HTML nous offre un tag prêt à l'emploi : <**template**></**template**>
- Il s'agit en gros d'une div cachée...

# Exemple Templating

```
<div class="artist-list">  
  
  <a href="#">  
      
    <div class="artist-list-item-title">Ava Max</div>  
  </a>  
  
  <a href="#">  
      
    <div class="artist-list-item-title">Ed Sheeran</div>  
  </a>  
  
  ...  
  
</div>
```



```
<div class="artist-list">  
  </div>  
  
<template id="artist-list-item-template">  
  <a href="#">  
    <img src="" />  
    <div class="artist-list-item-title"></div>  
  </a>  
</template>
```

# Exemple

## Templating

```
const artistList = document.querySelector('.artist-list')

const artistListItemTemplate = document.querySelector('#artist-list-item-template')

function afficherArtiste(artiste) {

    const newArtist = artistListItemTemplate.content.cloneNode(true) // true pour cloner également les enfants du node

    newArtist.querySelector('a').href = '#artists-' + artiste.id

    newArtist.querySelector('img').src = artiste.image_url

    newArtist.querySelector('.artist-list-item-title').innerText = artiste.name

    artistList.append(newArtist)

}

function afficherArtistes(artistes) {

    artistList.replaceChildren() // Remplace les enfants par rien, donc supprime tous les enfants

    for(const artiste of artistes){

        afficherArtiste(artiste)

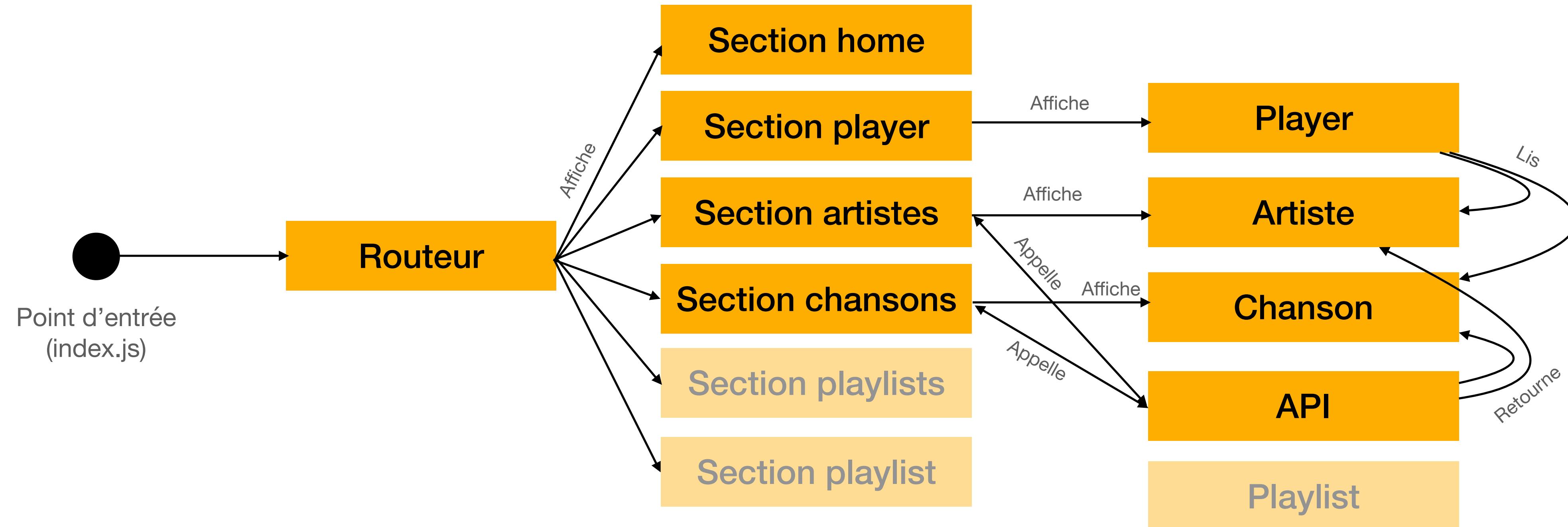
    }

}
```

# Put it together

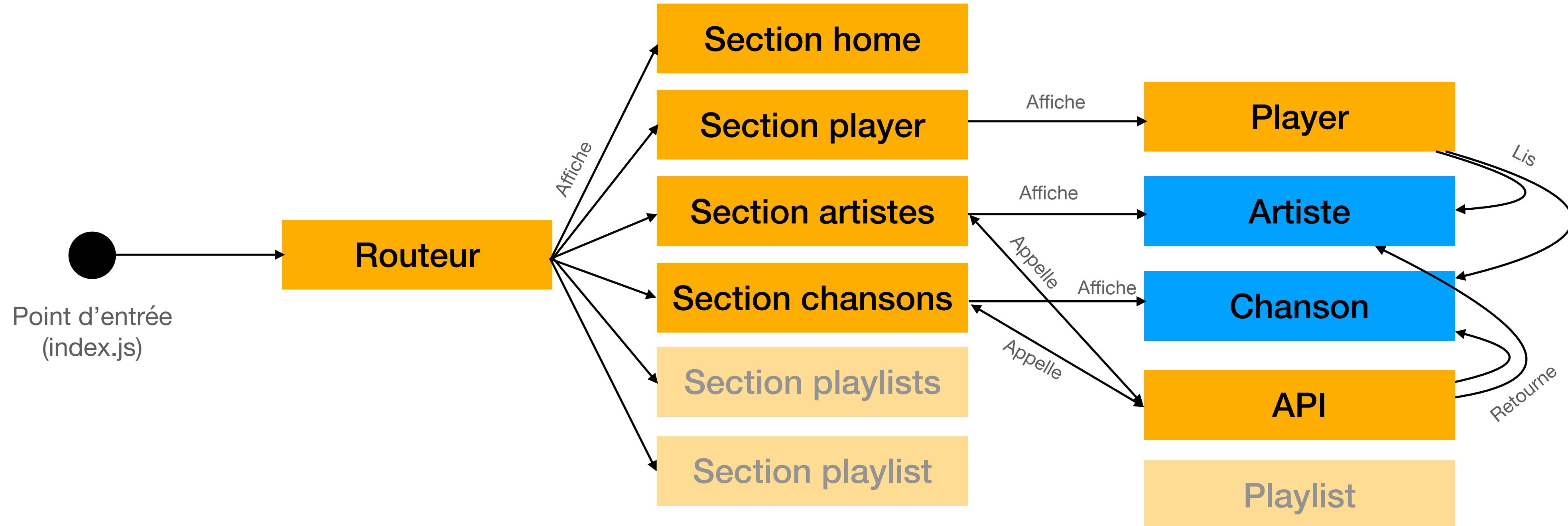
# Vue globale

## Put it together



# Vue globale

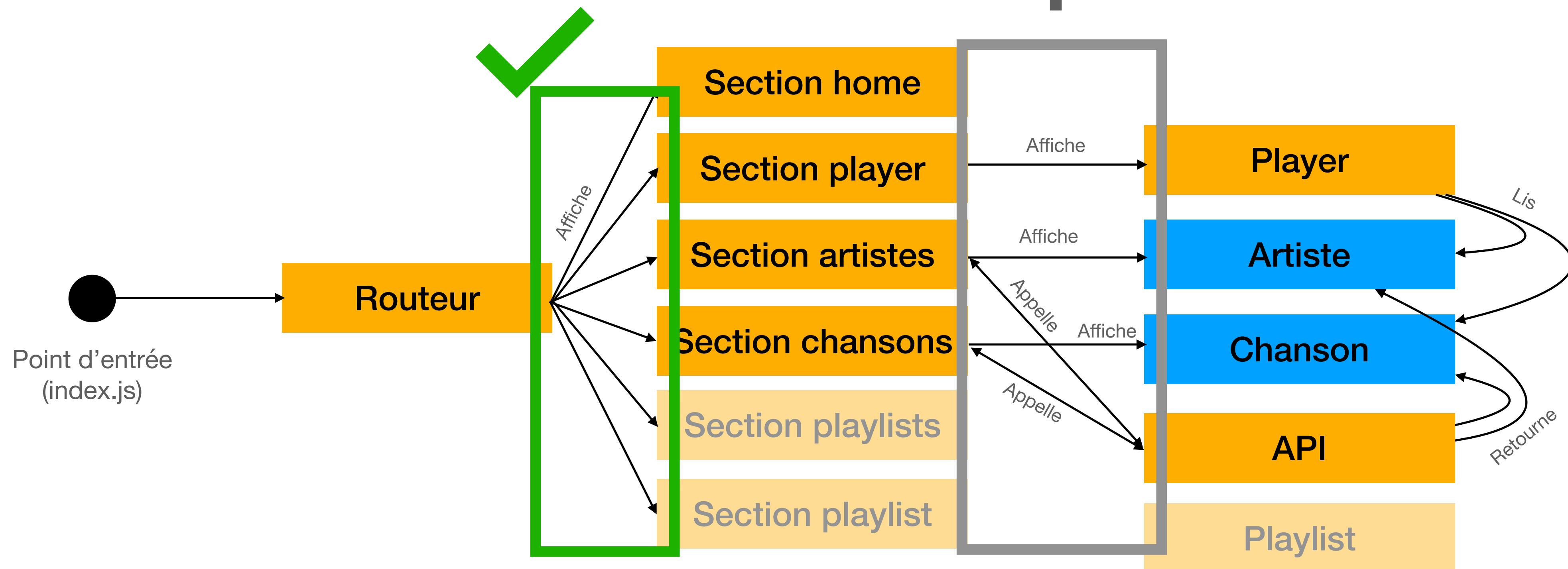
## Put it together



# Vue globale

## Put it together

?



# Routeur

## Put it together

```
// Affichage d'une section
function displaySection() {
    // Comme nos hash et nos ids de section sont les mêmes, hash = sectionid.
    // S'il n'y a pas de hash (par ex, on est sur "localhost:8080/"), le défaut devient '#home-section'
    const sectionId = window.location.hash || '#home-section'

    // Supprime/Ajoute la classe active sur la section
    document.querySelector('section.active')?.classList.remove('active')
    document.querySelector(sectionId)?.classList.add('active')

    // Supprime/Ajoute la classe active sur le lien
    document.querySelector('nav a.active')?.classList.remove('active')
    document.querySelector('nav a[href="' + sectionId + '"]')?.classList.add('active')

    // ... ???
    if(sectionId == '#artistes-section')
        loadArtistesSection() // Par exemple ????
}
```

# Routeur - Liens enfants (vue artiste)

## Put it together

- Comment gérer les liens enfants, type “#artists-12” ?
- Il nous faut différencier :
  - La liste des artistes “#artists”
  - La vue d'un artiste, selon son id “#artists-12”

# Routeur - Liens enfants (vue artiste)

Put it together

“#artists-12”



Est-ce qu'il y a un tiret ?

# Routeur - Liens enfants (vue artiste)

## Put it together

- Pléthore de manières de faire... Les deux principales seraient:
  - Chercher le tiret dans la chaîne de caractère et la découper avec des expressions régulières ou des fonctions de recherche...
  - Se dire que la chaîne “#artists-12” n'est en fait qu'une liste d'éléments séparés par des tirets... et les convertir en tableau

# Routeur - Liens enfants (vue artiste)

## Put it together

- La v2 semble plus simple...
- La fonction “split” vient à la rescouisse !
- Elle permet de découper une chaîne en un tableau de sous-chaînes de caractères, selon un caractère donné
- Exemple:  
`'une-chaine-de-tirets'.split('-') ==> ['une', 'chaine', 'de', 'tirets']`

# Routeur - Fonction split

## Put it together

- Exemple précédent

```
'une-chaine-de-tirets'.split('-') ==> [ 'une', 'chaine', 'de', 'tirets' ]
```

- Que se passe-t-il si pas de tirets ?

```
'uneChaineSansTirets'.split('-') ==> [ 'uneChaineSansTirets' ]
```

- Que se passe-t-il avec une chaîne vide ?

```
'.split('-') ==> [ '' ]
```

# Routeur - Fonction split

## Put it together

- Avec nos artistes :

- Avec tirets

```
const hashSplité = '#artists-12'.split('-') ==> ['#artists', '12']
```

- Que se passe-t-il si pas de tirets ?

```
'#artists'.split('-') ==> ['#artists']
```

# Routeur - Fonction split

## Put it together

- Exemple d'implémentation :

```
const hashSplité = window.location.hash.split('-')
```

```
// avec '#artists-12' ==> ['#artists', '12']
```

```
// avec '#artists-section' ==> ['#artists', 'section']
```

# Routeur - Fonction split

## Put it together

- Que se passe-t-il quand on essaie d'accéder à un élément d'un tableau qui n'existe pas ?

Exemple: La cellule 397 du tableau ['a', 'b', 'c'] ?

```
const tableau = ['a', 'b', 'c']
```

```
console.log(tableau[397]) // ==> undefined
```

# Routeur - Fonction split

## Put it together

Par exemple...

```
const hashSplité = window.location.hash.split('-')

// si le premier élément est artiste, on est dans la gestion des artistes...

if(hashSplité[0] == '#artists') {

    // est-ce que le deuxième élément retourne quelque chose ? Et donc n'est pas undefined ? Oui?
    // Alors il y a un id et on affiche cet artiste
    alors un peu comme vous préférez !

    if(hashSplité[1]) {

        afficherChansonsArtiste(hashSplité[1]) // la fonction prend en argument l'id de l'artiste à afficher
    }

    else {

        afficherArtistes()
    }
}
```

# Routeur - Fonction split

## Put it together

Par exemple... avec un switch

```
const hashSplité = window.location.hash.split('-')

// si le premier élément est artiste, on est dans la gestion des artistes...

switch(hashSplité[0]) {

    case '#artists':

        // est-ce que le deuxième élément retourne quelque chose ? Et donc n'est pas undefined ? Oui?
        // Alors il y a un id et on affiche cet artiste

        if(hashSplité[1]) {

            afficherChansonsArtiste(hashSplité[1])

        }

    else {

        afficherArtistes()

    }

    break;

    case '#player':

        ...

    break;

}
```

**GO!**

# Player

# Concept

## Player

- Le player doit être capable de lire une chanson
- Avancer dans la chanson
- Mettre à jour le bouton play/pause, selon son état
- Afficher son titre + les détails de l'artiste
- Utiliser les boutons précédent/suivant, selon le contexte dans lequel la chanson en cours a été lue

# Tag audio

## Player

- Le tag audio prend sa source grâce à l'attribut “src”
- Il dispose de plusieurs méthodes de contrôles
- Il émet plusieurs événements pour gérer son état

# Tag audio - Play/pause

## Player

```
const player = document.querySelector('#audio-player')
```

```
player.src = 'http://...hello.mp3'
```

```
player.paused // retourne true ou false
```

```
player.play()
```

```
player.pause()
```

```
// avancer dans la chanson, quand on déplace le slider, par ex  
player.currentTime = 1234
```

# Tag audio - Play/pause

## Player

```
const player = document.querySelector('#audio-player')

...

function togglePlayPause() {
  if(player.paused)
    player.play()
  else
    player.pause()
}

document.querySelector('#player-control-play').addEventListener('click', togglePlayPause)
```

# Tag audio - Avancer Player

```
const player = document.querySelector('#audio-player')

...

function avancerPlayer(event) {
    player.currentTime = event.currentTarget.value
}

document.querySelector('#player-progress-bar').addEventListener('change', avancerPlayer)
```

# Tag audio - Evénements

## Player

```
const player = document.querySelector('#audio-player')
```

```
// Appelé quand la valeur de player.paused a changé (true/false)  
player.addEventListener('play', changerIcone)
```

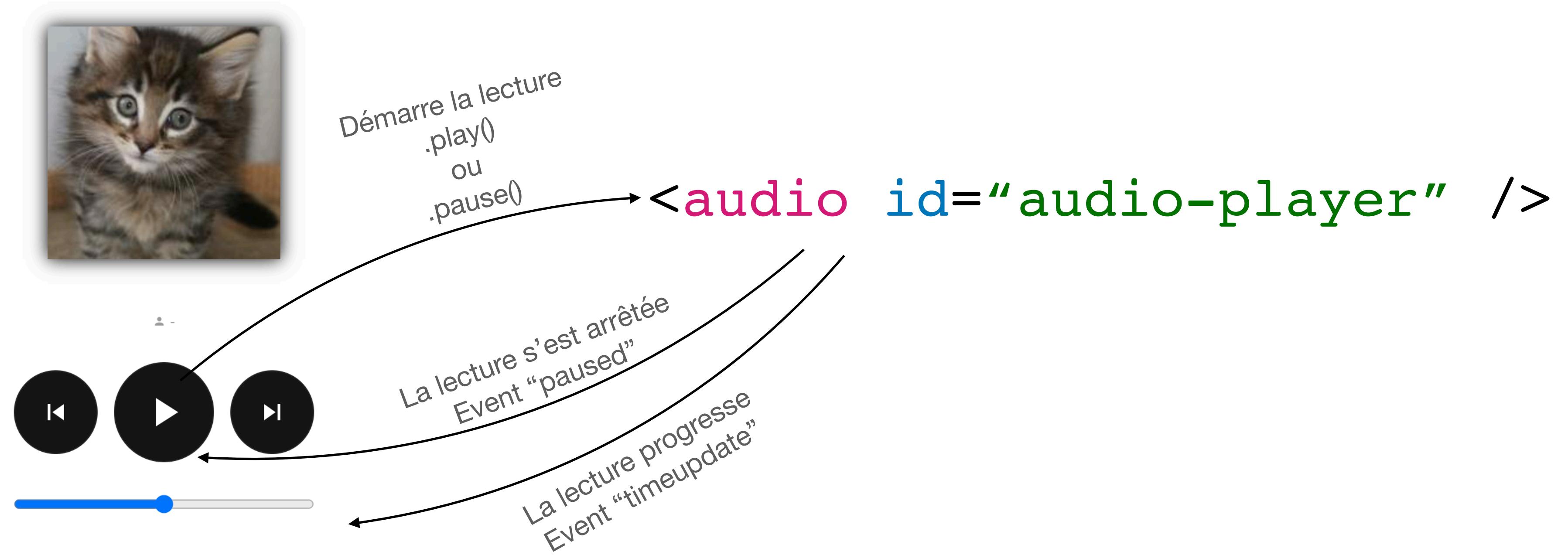
```
// Appelé quand la chanson est à la fin  
player.addEventListener('ended', chansonSuivante)
```

```
// Appelé quand la valeur de player.duration a changé (ex: une nouvelle chanson  
// est chargée, on met à jour la durée)  
player.addEventListener('durationchange', mettreAJourValeurMaxSlider)
```

```
// Appelé lorsque la chanson progresse (mettre à jour le slider)  
player.addEventListener('timeupdate', mettreAJourValeursSlider)
```

# Tag audio

## Player



# Tag audio - Next/prev

## Player

- Le tag audio ne gère pas la notion de précédent/suivant
- Ce sera à vous de garder une copie du tableau de chansons dans lequel se trouve la chanson en cours et passer à la précédente/suivante, selon sa position dans le tableau

# Fonctions du player

## Player

- Nous allons évidemment cacher toutes ces méthodes dans un seul fichier et offrir une interface au reste de l'application pour discuter avec le player
- Cela évite de copier/coller des `querySelector('#audio-player')` partout dans le code
- Cela permet aussi de gérer la liste de lecture en cours pour les boutons précédent/suivant

# Fonctions du player

## Player

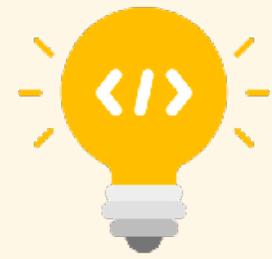
- Une section autre que celle du player doit pouvoir appeler une fonction de lecture du style:

```
lireChanson(laChanson, leTableauDeChansons)
```

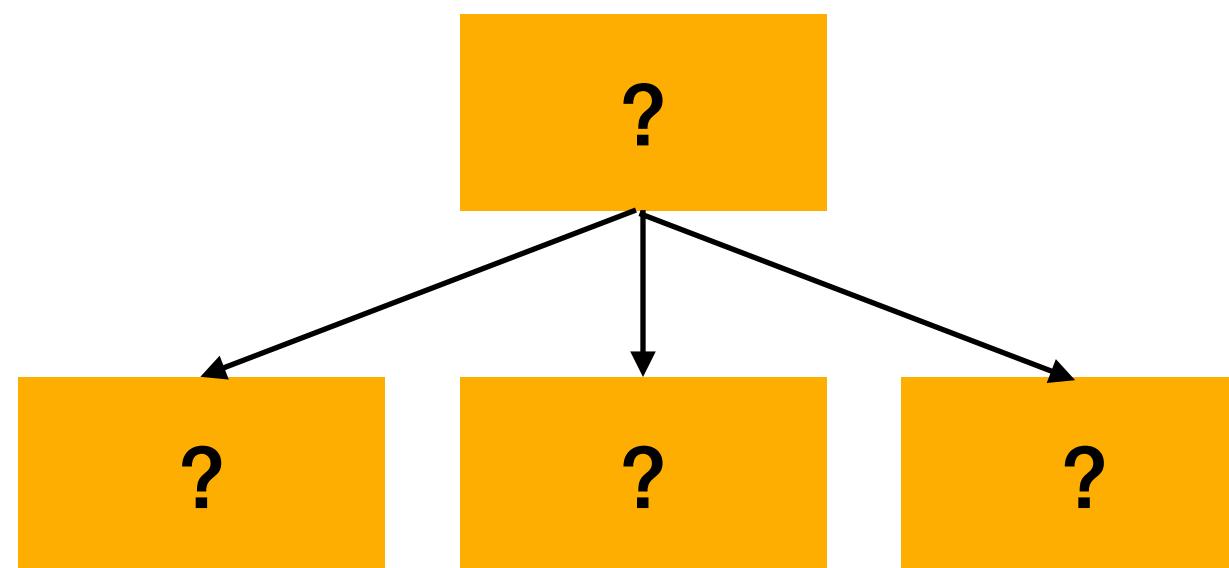
- Le reste doit être encapsulé dans un fichier player.js, par exemple
- Au chargement de l'application, tous les événements doivent être liés au tag audio pour mettre à jour l'interface

# Structure

## Player



### Comment structurer le code ?



**GO!**

# Formatage du temps

## Player

- Par défaut, les valeurs transmises par le tag audio sont en secondes
- Le but est de les afficher proprement, sous forme “MM:SS” (ex: 03:24)
- Vous pouvez utiliser le code suivant:

[https://github.com/lgavillet/webmobui-22/blob/master/Ressources/  
formatTimestamp.js](https://github.com/lgavillet/webmobui-22/blob/master/Ressources/formatTimestamp.js)

# Formatage du temps

## Player

```
import formatTimestamp from './lib/formatTimestamp' // ou autre chemin que vous aurez choisi  
...  
console.log(formatTimestamp(duréeEnSecondes))
```

# Recherche

# Structure

## Recherche



Comment intégrer la fonction de recherche ?

# Structure

## Recherche

- Créer un listener ‘change’ sur le champ de recherche
- Quand il change, il faut prendre sa valeur et la passer à l’endpoint de recherche
- La valeur se passe dans l’URL: /api/songs/search/:query
- Exemple: /api/songs/search/dynoro

# Structure

## Recherche

- Cela fonctionne bien pour de petites chaînes, mais comment passer des caractères complexes ?
- `encodeURIComponent(laChaine)`
- Il faudrait donc faire:
- `fetch("https:.../api/songs/search/" + encodeURIComponent(laChaine))`

# Structure

## Recherche

- L'API de recherche vous renvoie la même structure que l'endpoint des chansons d'un artiste
- Vous pouvez donc réutilisez une bonne partie de votre code pour afficher résultat!
- Une fois le résultat obtenu, il vous suffira d'afficher la section correspondante pour la liste des résultats (de l'intérêt de bouger la logique d'affichage dans des petites méthodes d'aide : ) )

# Playlists

# Concept

## Playlists

- Le but est de gérer les playlists localement
- En opposition à l'API qui gère les chansons de manière centralisée sur le serveur, nous utiliserons une des API Web pour les stocker dans le browser
- Plusieurs API disponibles...

# Les API de stockage

## Playlists

APIs de stockage :

- IndexedDB - Une vrai database, disponible dans le browser
- Web Storage - Stockage de clés/valeurs (genre de cookies sous stéroïdes)

Nous allons nous focuser sur la deuxième...

# L'API Web Storage

## Playlists

Web Storage est séparée en 2 sous-API :

- **sessionStorage**

Les données ne sont stockées que pendant la durée de la session. Lorsque le browser se ferme, les data sont effacées

- **localStorage**

Les données sont persistées entre les sessions et sont présentent après un restart

Ces deux APIs s'utilisent de la même manière, c'est uniquement la persistance des données qui change

# L'API Web Storage

## Playlists

Méthodes principales :

- `setItem(clé, valeur)` - Défini un élément “clé” avec comme valeur “valeur”
- `getItem(clé)` - Lis l’élément “clé”
- `removeItem(clé)` - Supprime l’élément “clé”
- `clear()` - Vide la totalité du localStorage

# L'API Web Storage

## Playlists

Exemple :

```
const uneValeur = 12345
```

```
localStorage.setItem('uneCléAChoix', uneValeur)
```

```
console.log(localStorage.getItem('uneCléAChoix'))
```

```
==> "12345"
```

# L'API Web Storage

## Playlists

Exemple :

```
const uneValeur = 12345
```

```
localStorage.setItem('uneCléAChoix', uneValeur)
```

```
console.log(localStorage.getItem('uneCléAChoix'))
```

==> “12345”  
?!?!?!?!

localStorage ne stock que des chaînes de caractères...

# L'API Web Storage

## Playlists

- Est-ce un problème ?
- Oui et non... nous connaissons... JSON !
- Il est donc possible de créer deux petites fonctions d'aide

```
function setJsonItem(clé, valeur) {  
    return localStorage.setItem(clé, JSON.stringify(valeur))  
}  
  
function getJsonItem(clé) {  
    return JSON.parse(localStorage.getItem(clé))  
}
```

# L'API Web Storage

## Playlists

- Cela fonctionne pour les structures standards, mais dès qu'il y a de grosses structures, cela devient complexe à gérer
- Il faudrait donc améliorer notre API custom pour gérer des cas plus complexe
- Exemple: Plusieurs playlists...  
`setJsonItem( 'playlist_1', [...] )`  
`setJsonItem( 'playlist_2', [...] )`

# L'API Custom JsonStorage

## Playlists

- M. Chablotz vous a préparé une API plus complexe qui permet de gérer facilement ces grosses structures

<https://github.com/Igavillet/webmobui-22/blob/master/Ressources/JsonStorage.js>

<https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>

# L'API Custom JsonStorage

## Playlists

- Comment importer et instancier l'API JsonStorage :

```
import JsonStorage from './lib/jsonStorage' // ou autre chemin que vous aurez choisi
```

...

```
const playlistStorage = new JsonStorage({ name: 'playlists' })
```

# L'API Custom JsonStorage

## Playlists

Les méthodes importantes :

- `addItem(valeur)` - Pour ajouter un élément, comme un tableau (cela va aussi générer un id aléatoire comme clé)
- `setItem(id, valeur)` - Modifier un élément
- `removeItem(id)` - Supprimer un élément
- `forEach(fonction(valeur, id))` - Itérer sur les éléments

# L'API Custom JsonStorage

## Playlists

### Utilisation de base et ajout d'une playlist

```
import JsonStorage from './lib/jsonStorage' // ou autre chemin que vous aurez choisi  
...  
  
const playlistStorage = new JsonStorage({ name: 'playlists' })  
...  
  
playlistStorage.addItem({ name: 'Ma playlist', songs: [...] })
```

# L'API Custom JsonStorage

## Playlists

### Modifier une playlist

```
const maPlaylist = playlistStorage.getItem(idDeLaPlaylist)

maPlaylist.name = 'Un nouveau nom'

maPlaylist.songs = maPlaylist.songs.slice(1) // on enlève la première, par exemple..

playlistStorage.setItem(idDeLaPlaylist, maPlaylist)
```

# L'API Custom JsonStorage

## Playlists

### Supprimer une playlist

...

```
playlistStorage.removeItem(idDeLaPlaylist)
```

# L'API Custom JsonStorage

## Playlists

### Afficher les playlists

...

```
playlistStorage.forEach((valeur, id) => {  
  
    console.log("l'id est :", id)  
  
    console.log("le nom est :", valeur.name)  
  
    console.log("les chansons:", valeur.songs)  
  
    afficherPlaylist(valeur)  
  
})
```

# Favoris

# Mise à jour du prototype

## Favoris

<https://invis.io/DG12CQJQ9QPF#/463974083> Home

<https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>

# L'API Custom JsonStorage

## Favoris

### Utilisation de base et ajout d'un favori

```
import JsonStorage from './lib/jsonStorage' // ou autre chemin que vous aurez choisi  
...  
  
const favoriteStorage = new JsonStorage({ name: 'favorites' })  
...  
favoriteStorage.addItem(uneSong)
```

# Trouver un favori

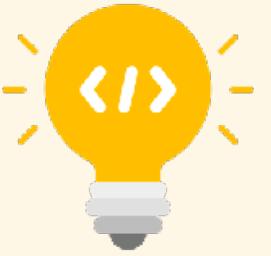
## Favoris

Vérifier si une chanson est présente dans les favoris

```
if(favoriteStorage.toArray().find((entry) => entry[1].id == idRecherché)) {  
    console.log("C'est dedans!")  
}  
else {  
    console.log("ça n'y est pas...")  
}
```

# Structure

## Favoris



### Comment structurer le code ?