

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

Décomposer le problème

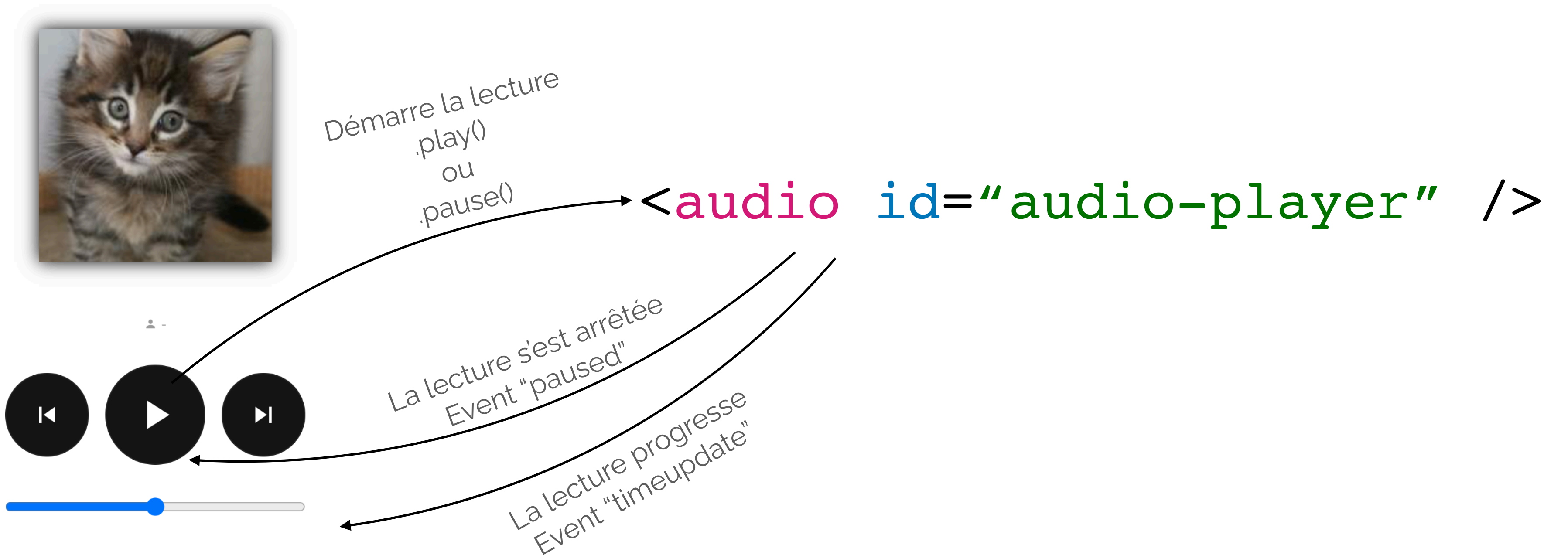
Player

Le problème peut se diviser en deux sous-problèmes :

- La logique de lecture (play/previous, next, previous) avec le tag audio
- La mise à jour de la UI avec les meta data (artiste, titre, durée, avance, ...)

Tag audio

Player



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' )

...

const 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' )
```

```
...
```

```
const avancerPlayer = (event) => {
```

```
    player.currentTime = event.currentTarget.value
```

```
}
```

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


Tag audio – Événements

Player

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

// Appelé quand la valeur de player.paused à changé (true/false)
player.addEventListener( 'play', changerIcône )

// 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', mettreAJourValeurSlider )
```

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

Formattage 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)

```
import formatTimestamp from './lib/formatTimestamp'
```

```
...
```

```
console.log(formatTimestamp(duréeEnSecondes))
```

Formattage du temps

Player

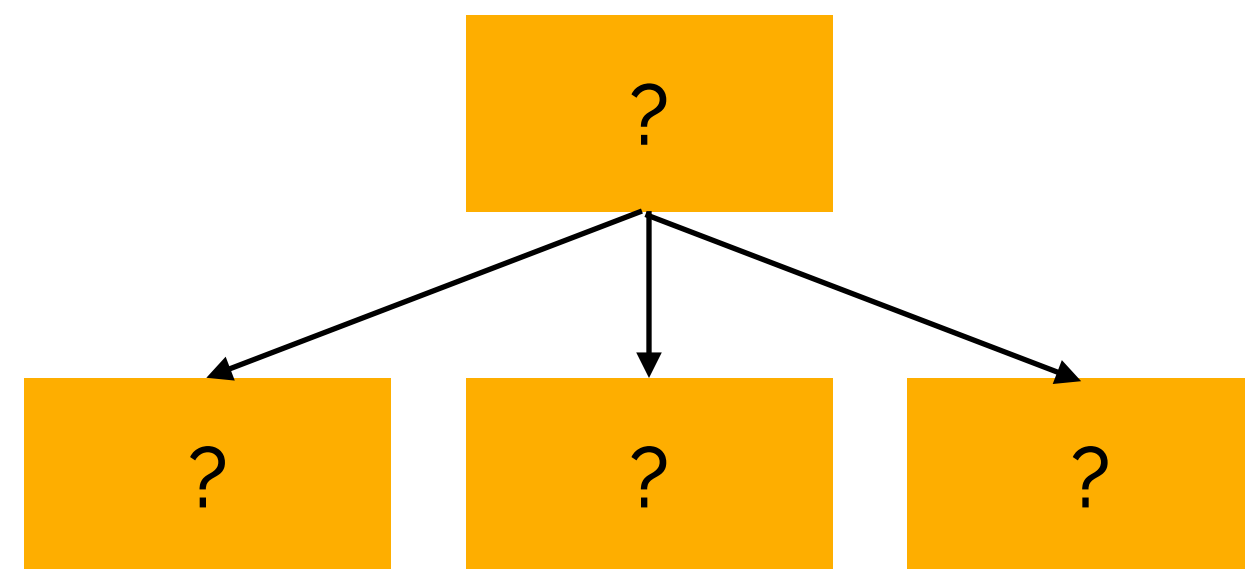
```
import formatTimestamp from './lib/formatTimestamp' // ou autre chemin que vous aurez choisi  
  
...  
  
console.log(formatTimestamp(duréeEnSecondes))
```

Structure

Player



Comment structurer le code ?



Events, listeners & custom elements

Concept

Events, listeners & custom elements

```
<song-item title="Oh My My ft. Séb Mont" />
```



Proposition

Events, listeners & custom elements

```
<song-item title="Oh My My ft. Séb Mont" />
```



```
const newSongItem = document.createElement('song-item')

// ...

newSongItem.addEventListener('play_click', () => console.log('ça play'))

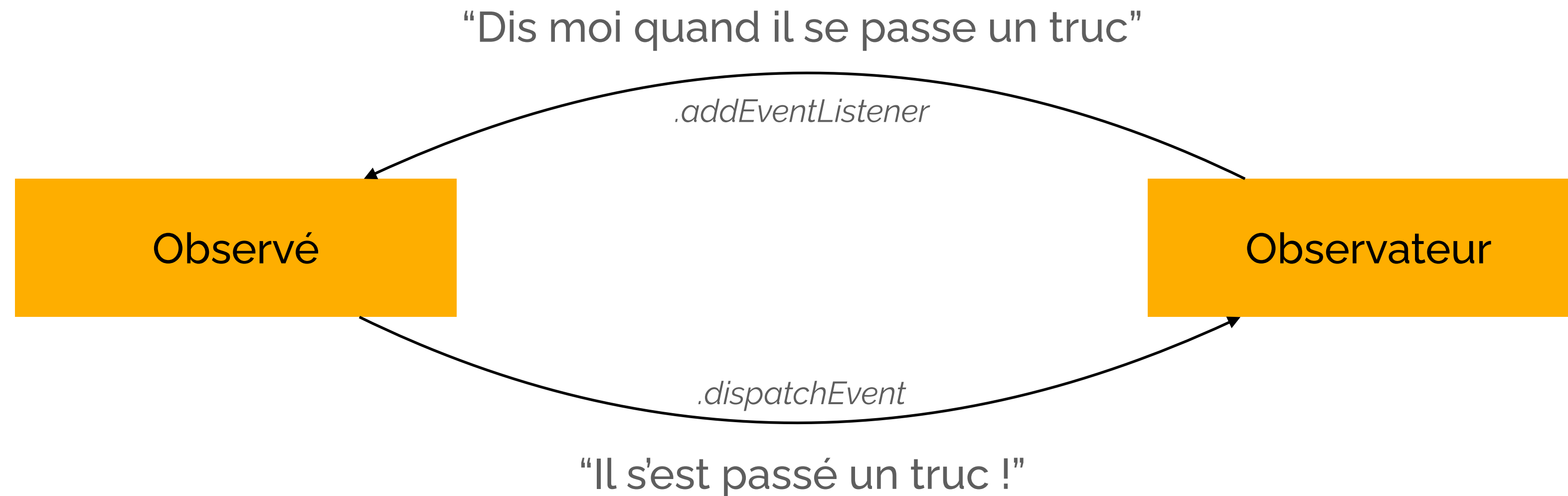
newSongItem.addEventListener('favorite_click', () => console.log('ça favorise'))

// ...

songList.append(newSongItem)
```

Events – rappel

Events, listeners & custom elements



Custom events

Events, listeners & custom elements

- Comme des custom elements, le browser nous permet de générer des custom events
- Il suffit d'en créer une instance d'un custom event et de définir une chaîne de caractères pour l'identifier
- Il est ensuite possible de dispatcher un événement depuis n'importe quel objet

Custom events

Events, listeners & custom elements

```
// Déclaration d'un custom event
```

```
const playEvent = new CustomEvent('play_click')
```

```
class SongItem extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = ...  
  
    const playButton = this.querySelector('.play-button')  
  
    playButton.addEventListener('click', (e) => {  
      e.preventDefault()  
      this.dispatchEvent(playEvent)  
    })  
  }  
}
```

```
customElements.define("song-item", SongItem)
```

```
const newSongItem = document.createElement('song-item')
```

```
// ...
```

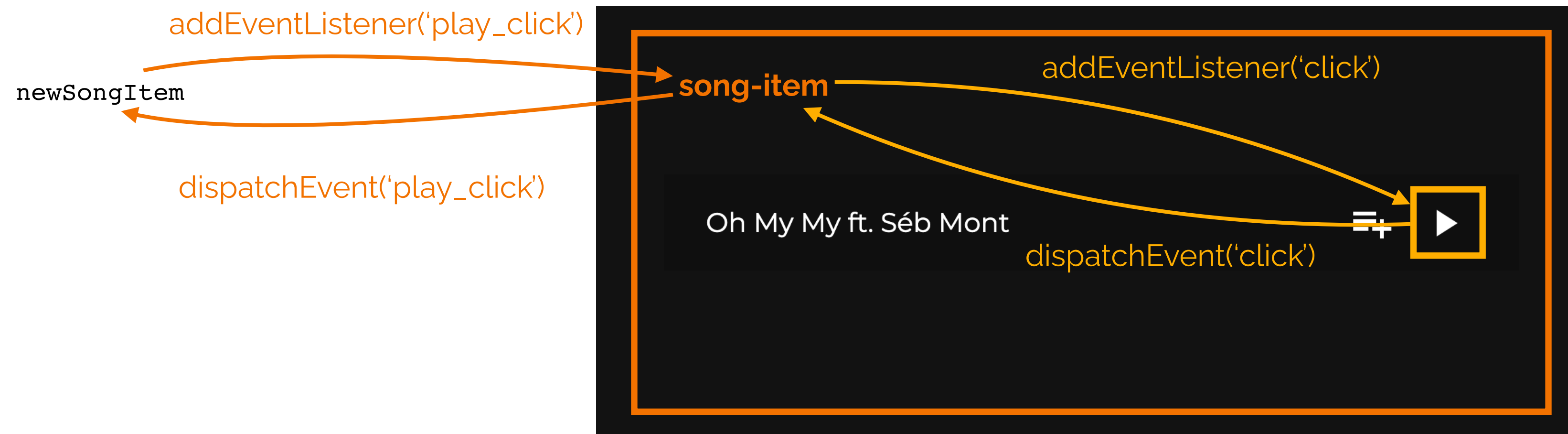
```
newSongItem.addEventListener('play_click', () => console.log('ça play'))
```

```
// ...
```

```
songList.append(newSongItem)
```

Concept

Events, listeners & custom elements



Goal

- Version “basique” du player -> il doit être possible de :
 - Lire une chanson, via une méthode `playSong(song)`
 - Mettre en play/pause grâce aux boutons du player
- Lire une chanson en cliquant sur le bouton “play” d'une chanson
- Avancer sur les autres features du player