

Routing

Single Page Application

Routing

- 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

Routing

Comment rendre
les différentes pages ?

Single Page Application

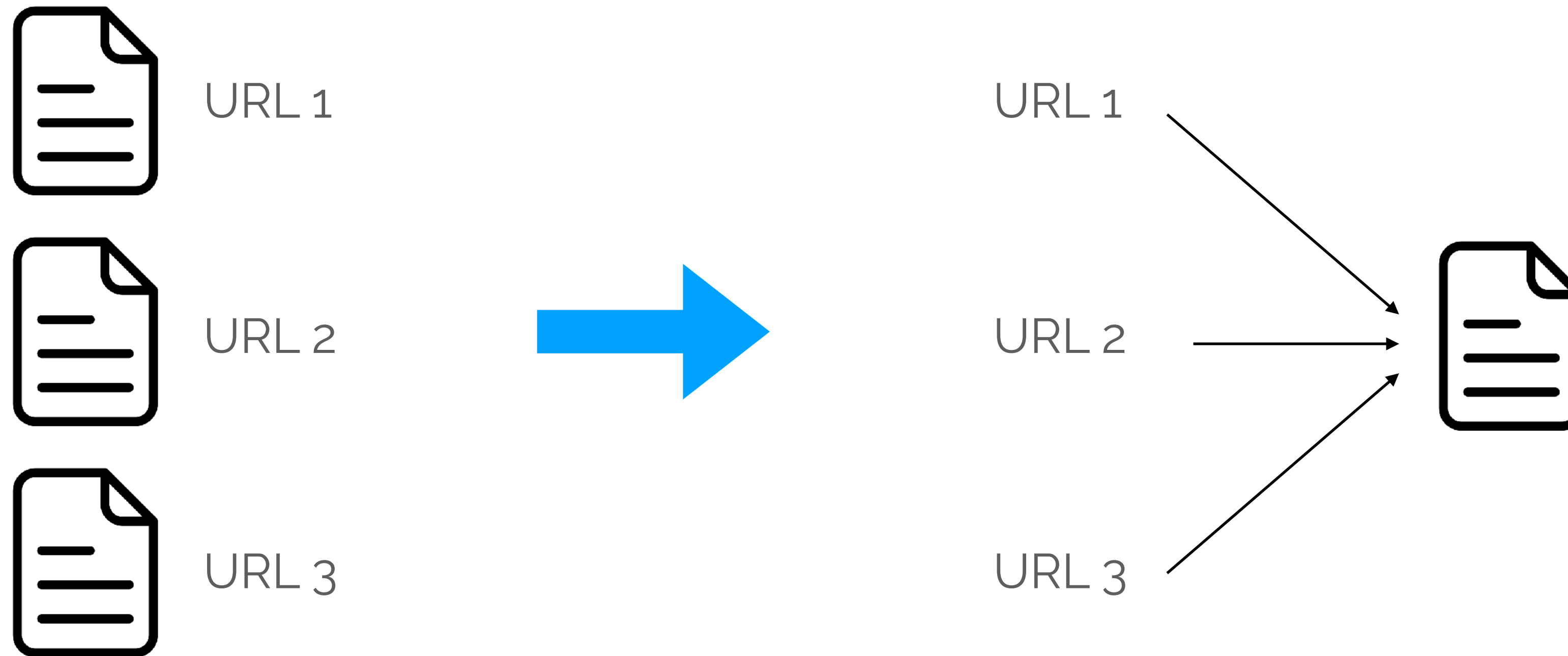
Routing

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

Routing



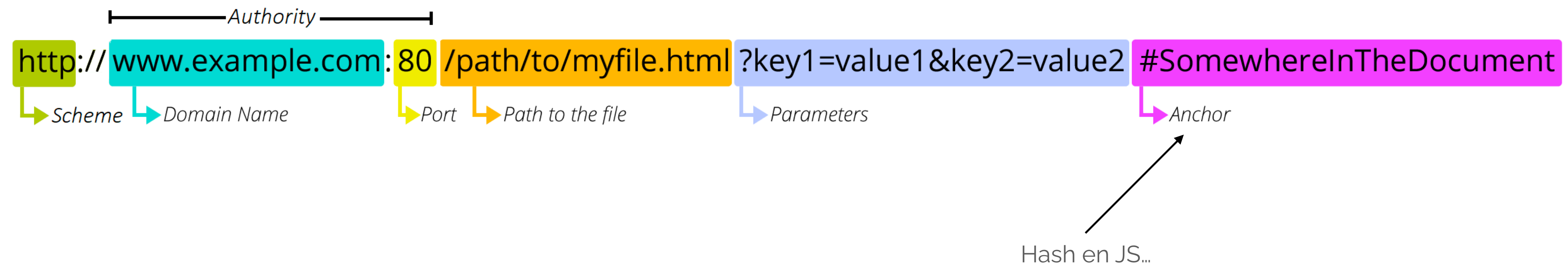
Single Page Application – Routing

Routing

Comment gérer l'état
de la page en cours?

Single Page Application – Think RESTful..

Routing



Single Page Application – Think RESTful..

Routing

<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

Routing

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 ancres 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 Routing

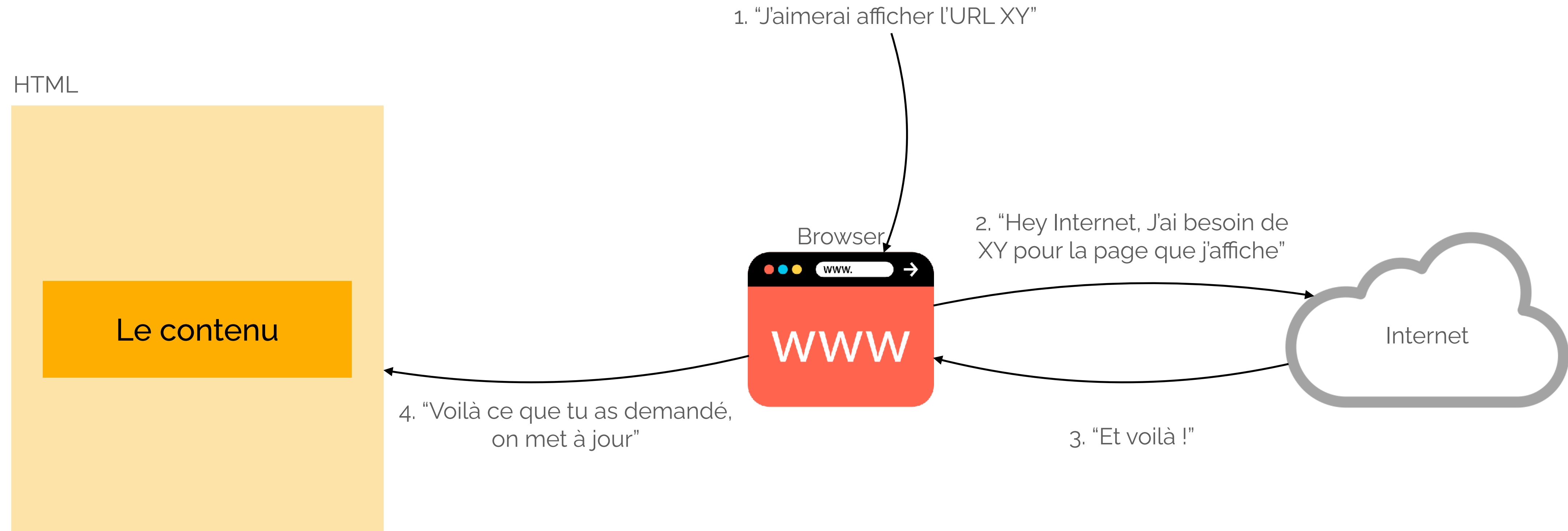
- Possible par exemple de prendre la structure d'URL suivante :
 - /#home
 - /#player
 - /#artists
 - /#artists-12



Avantages ? Inconvénients ?

Changement de section – Chargement de la page

Routing



Changement de section – Lien classique

Routing

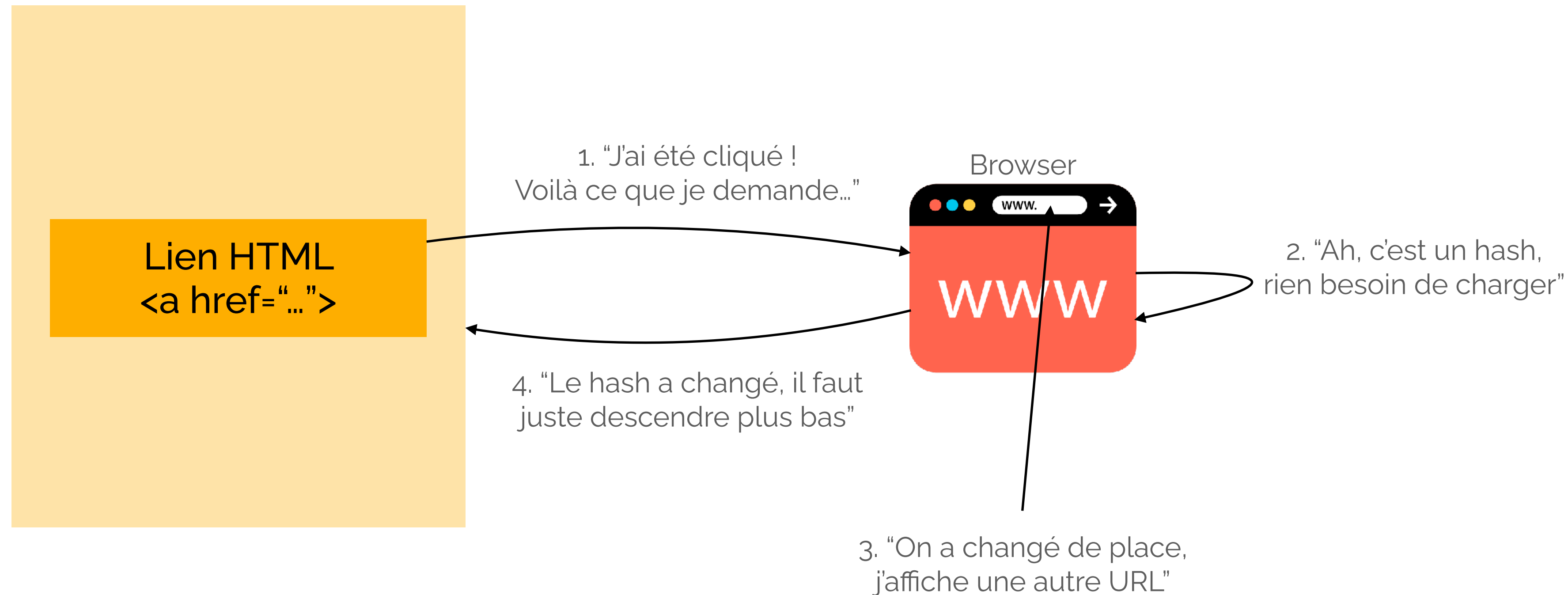
HTML



Changement de section – Lien avec hash

Routing

HTML



Changement de section – Comment ? V1

Routing

o. On intercepte avec un listener 'click' et on remplace l'action par défaut, par l'affichage de la section

HTML

Lien HTML

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

Browser



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

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

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

Changement de section – Comment ? V1

Routing

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

Routing

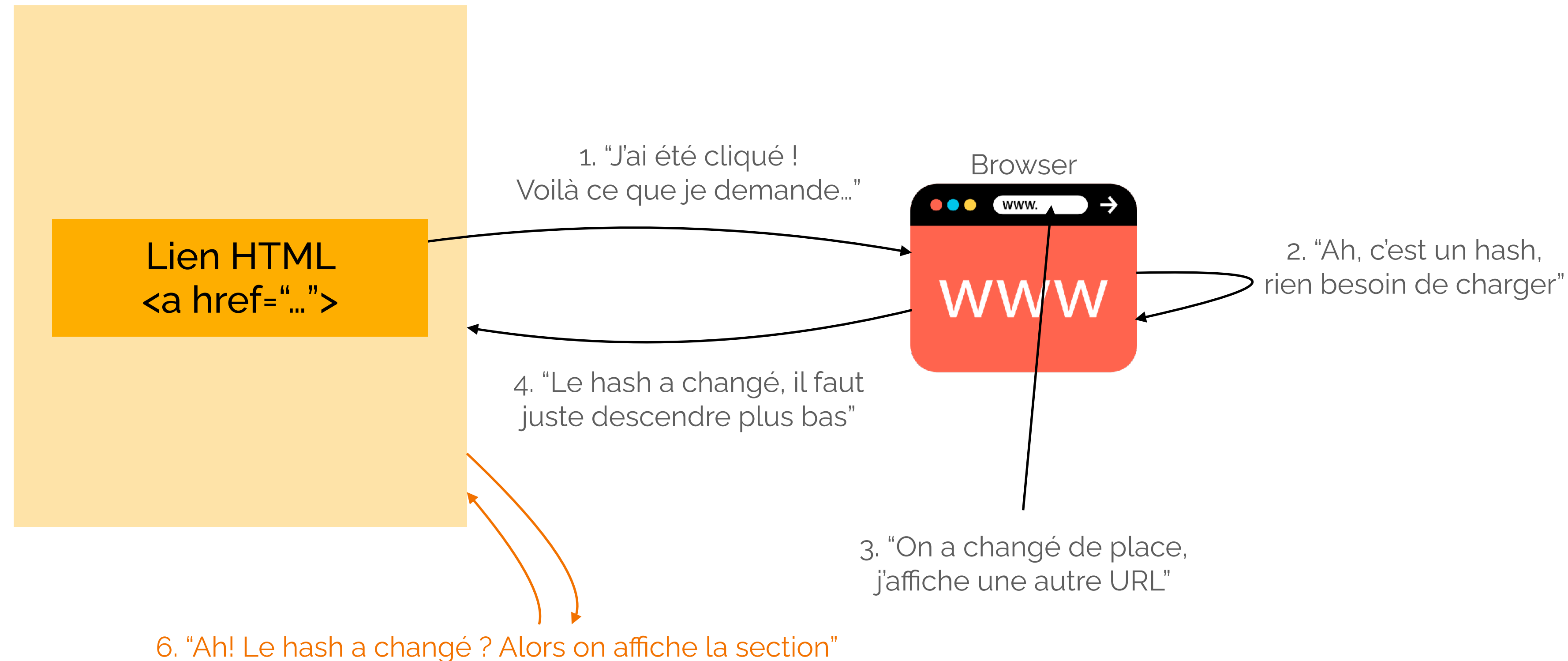
L'API History

- `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

Routing

HTML



V1

https://developer.mozilla.org/fr/docs/Web/API/History_API

`history...`

V2

<https://developer.mozilla.org/fr/docs/Web/API/WindowEventHandlers/onhashchange>

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

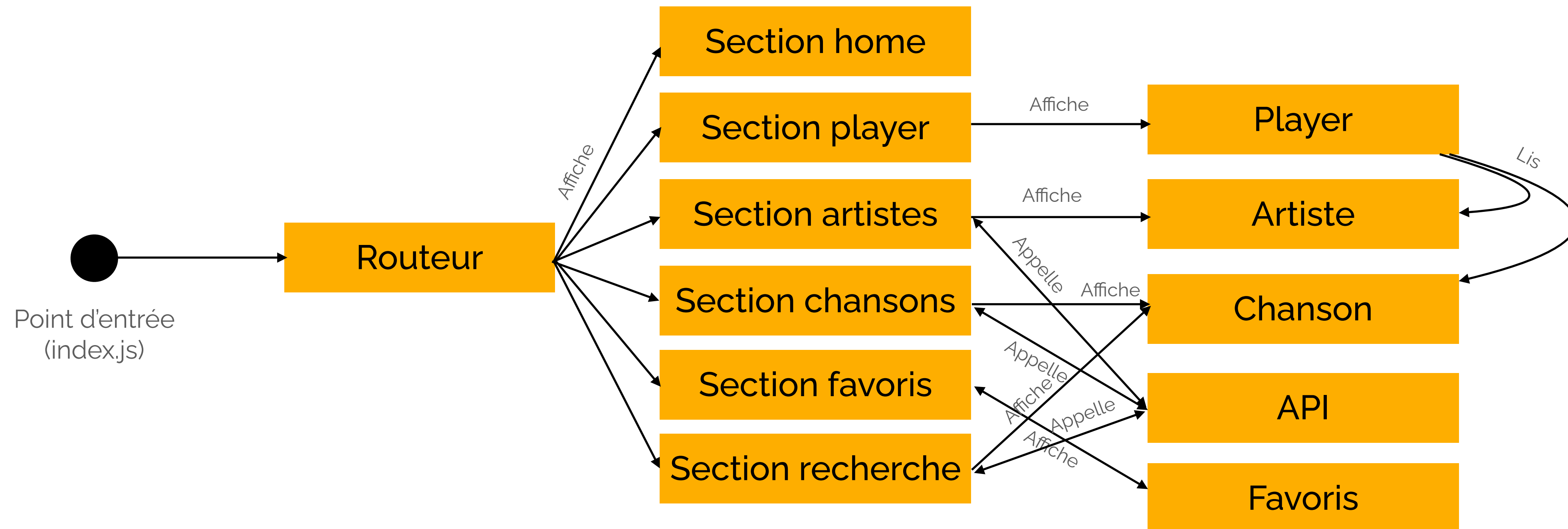
Changement de section – Comment ?

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

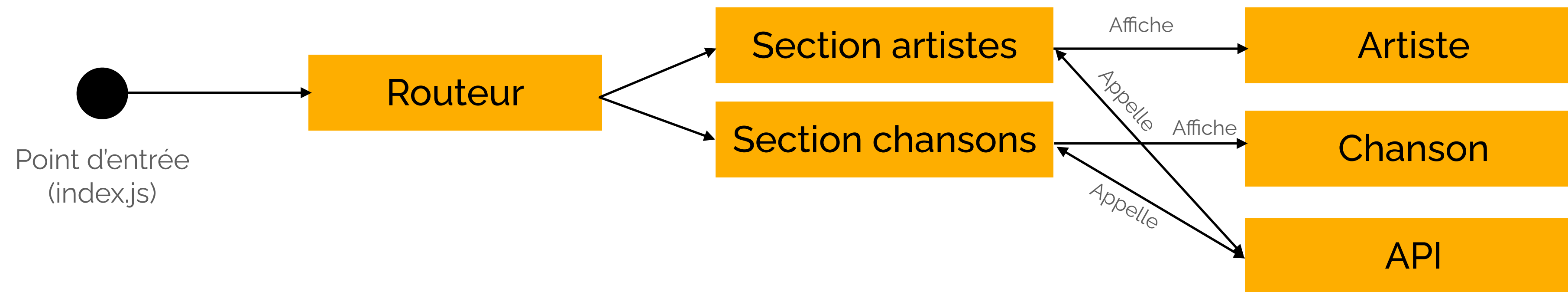
Put it together

Put it together



Vue globale

Put it together



Routeur

Put it together

```
// S'il n'y a pas de hash (par ex, on est sur "localhost:1234/"), le défaut devient '#home'
const hash = window.location.hash || '#home'

if(hash == '#home') {
  mettreLaClasseActiveSurLaSection('home')
} else if (hash == '#artists') {
  mettreLaClasseActiveSurLaSection('list')
  afficherArtistes()
}
```


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 rescousse !
- Elle permet de découper une chaîne en un tableau de sous-chaînes de caractères, selon un caractère donné
- Exemple:
`'une-chaîne-de-tirets'.split('-') ==> ['une', 'chaîne', '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 chaine 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

    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;

}
```