



Client



JS



Prezi

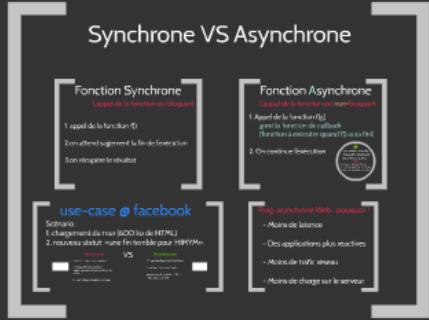
Client





AJAX

Asynchronous JAvascript and Xml



XMLHttpRequest

- Objet JavaScript permettant l'envoi asynchrone de requêtes HTTP sur le serveur
- On peut envoyer n'importe quoi

The diagram shows the structure of XMLHttpRequest, consisting of four main parts: 'Request', 'Response', 'Error handling', and 'Callback function'.

JSON
échanger des données avec du JavaScript

une simple structure de données JavaScript dans une chaîne de caractère

The diagram shows the structure of JSON, represented as a string of characters: [{ "name": "John", "age": 30, "city": "New York"}, { "name": "Peter", "age": 30, "city": "Paris"}]

Synchrone VS Asynchrone

Fonction Synchrone

L'appel de la fonction est bloquant

1. appel de la fonction f()
- 2.on attend sagement la fin de l'exécution
- 3.on récupère le résultat

Fonction Asynchrone

L'appel de la fonction est non-bloquant

1. Appel de la fonction f(g)
g est la fonction de callback
(fonction à exécuter quand f() aura fini)

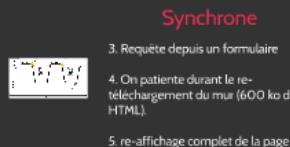
2. On continue l'exécution



use-case @ facebook

Scénario :

1. chargement du mur (600 ko de HTML)
2. nouveau statut: «une fin terrible pour HIMYM»



VS



Prog. asynchrone Web : pourquoi ?

- Moins de latence
- Des applications plus réactives
- Moins de trafic réseau
- Moins de charge sur le serveur

Fonction Synchrone

L'appel de la fonction est bloquant

1. appel de la fonction f()
- 2.on attend sagement la fin de l'exécution
- 3.on récupère le résultat

Fonction Asynchrone

L'appel de la fonction est non-bloquant

1. Appel de la fonction f(g)

g est la fonction de callback

(fonction à exécuter quand f() aura fini)

2. On continue l'exécution

- + pas d'attente inutile,
- + très performant avec des entrées/sorties
- flot d'exécution non linéaire
- difficile à programmer /debugger « asynchronous spaghetti code »



- pas d'attente inutile,
- très performant avec des entrées/sorties



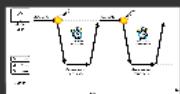
- flot d'exécution non linéaire
- difficile à programmer /debugger
« asynchronous spaghetti code »

use-case @ facebook

Scénario :

1. chargement du mur (600 ko de HTML)
2. nouveau statut: «une fin terrible pour HIMYM»

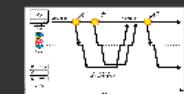
Synchrone



3. Requête depuis un formulaire
4. On patiente durant le re-téléchargement du mur (600 ko de HTML).
5. re-affichage complet de la page

VS

Asynchrone

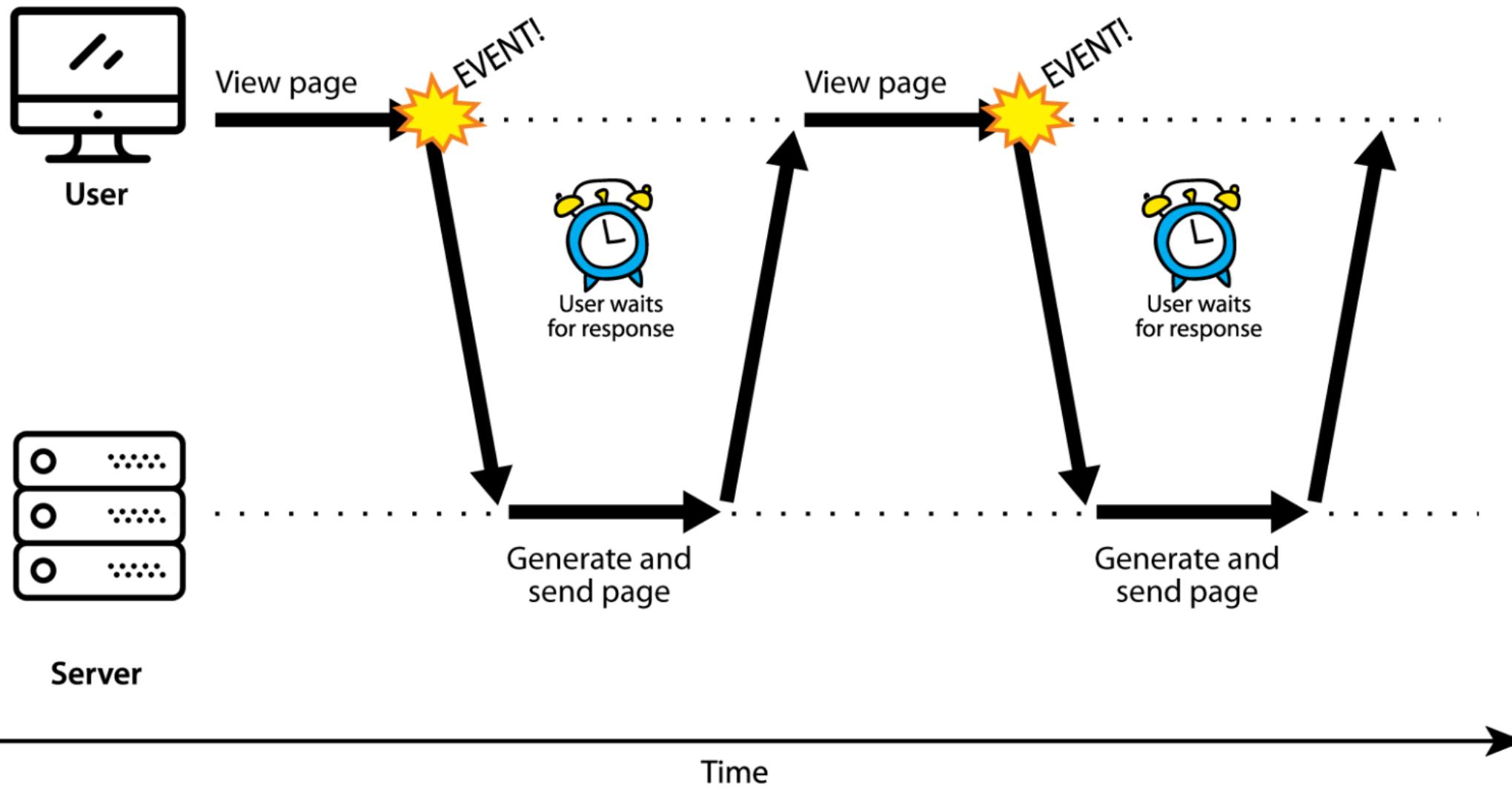


3. requête depuis du JavaScript
 4. on continue de glandeur
- @réponse du serveur (< 1ko) ->
mise à jour du mur

Synchrone

3. Requête depuis un formulaire
4. On patiente durant le re-téléchargement du mur (600 ko de HTML).
5. re-affichage complet de la page





Asynchrone

3. requête depuis du JavaScript

4. on continue de glande

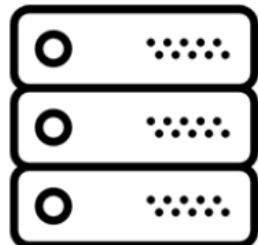
@réponse du serveur (< 1ko) ->
mise à jour du mur



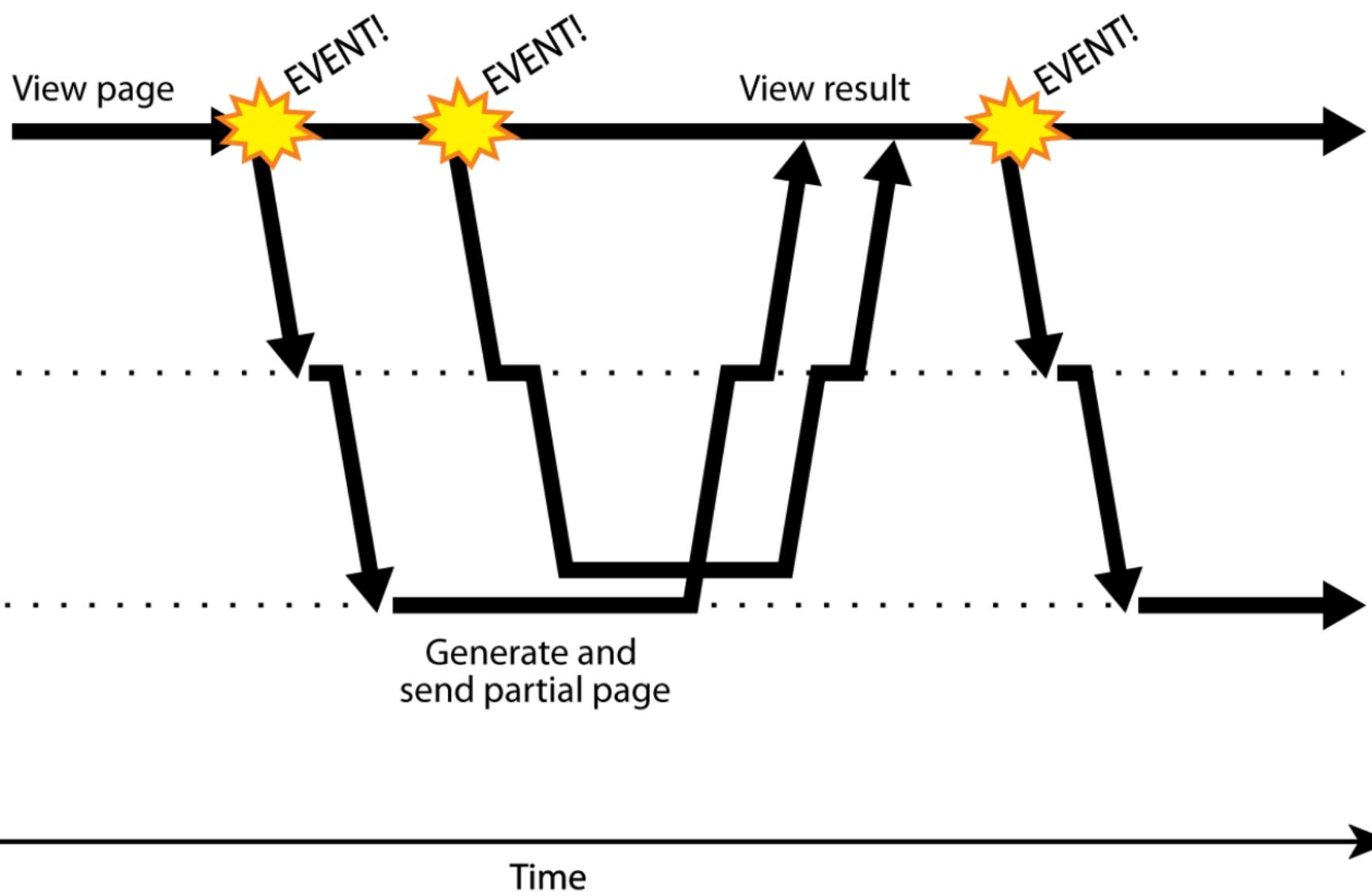
User



Ajax



Server



Prog. asynchrone Web : pourquoi ?

- Moins de latence
- Des applications plus réactives
- Moins de trafic réseau
- Moins de charge sur le serveur



XMLHttpRequest

- Objet JavaScript permettant l'envoi asynchrone de requêtes HTTP sur le serveur

- On peut envoyer n'importe quoi

```
var xhr = new XMLHttpRequest();
//La méthode d'envergure // method: POST/GET...
//et un URL de serveur destination
xhr.open(method,url);
//La méthode à contrôler à chaque fois
//qu'un état de la requête change
xhr.onreadystatechange = function() {
    //On vérifie si la réponse a été reçue
    if(xhr.readyState == 4) {
        //Si le status = 200,
        //on affiche le contenu
        console.log(xhr.responseText);
    }
}
xhr.send(); //envoie la requête (ici, sans contenu)
```

Exemple de requête GET
(récupérer le sujet du projet 'eswap' sur le site du cours)

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://users.polytech.unice.fr/~hermine/adw/test.php?query=subject&id=démineur');
xhr.onreadystatechange = function() {
    if(xhr.readyState == 4) {
        if(xhr.status == 200) {
            console.log(xhr.responseText);
        } else {
            console.log('Erreur: code=' + xhr.status);
        }
    }
}
xhr.send()
```

Exemple de requête POST
(authentifier sur le site du cours.)

```
var xhr = new XMLHttpRequest();
xhr.open('POST', 'http://users.polytech.unice.fr/~hermine/adw/test.php?query=auth');
//On envoie les données d'un formulaire
xhr.setRequestHeader('Content-type','application/x-www-form-urlencoded');
xhr.onreadystatechange = function() {
    if(xhr.readyState == 4) {
        //Le contenu du formulaire est dans le corps de la
        //requete (principe de fonctionnement de POST)
        xhr.responseText;
    }
}
```

```
var xhr = new XMLHttpRequest()

// La requête a envoyer // method: 'POST",GET',...
// url: l'URL du serveur destination
xhr.open(method,url)

// la méthode à exécuter à chaque fois
// que le statut de la requête change
xhr.onreadystatechange = function() {

    if (xhr.readyState == 4) { //La réponse a été reçu
        if (xhr.status == 200) {
            /* le serveur a répondu par un code 200,
               on affiche le contenu */
            console.log(xhr.responseText)
        } else {
            /*...*/
        }
    }
    xhr.send() //envoie la requête (ici, sans contenu)
```

Exemple de requête GET

(récupérer le sujet de projet ‘eswap’ sur le site du cours)

```
var xhr = new XMLHttpRequest()
xhr.open('GET', 'http://users.polytech.unice.fr/
~hermenie/adw/rest.php?query=subject&id=démineur')
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
      console.log(xhr.responseText)
    } else {
      console.log('Erreur. code=' + xhr.status)
    }
  }
}
xhr.send()
```

Exemple de requête POST

(s'authentifier sur le site du cours)

```
var xhr = new XMLHttpRequest()
```

```
xhr.open('POST', 'http://users.polytech.unice.fr/  
~hermenie/adw/rest.php?query=auth')
```

```
//On envoie les données d'un formulaire  
xhr.setRequestHeader("Content-type", "application/x-  
www-form-urlencoded");  
xhr.onreadystatechange = function() {/*...*/}
```

```
//Le contenu du formulaire est dans le corps de la  
// requête (principe de fonctionnement de POST)  
xhr.send('username=' + l + '&password=' + p)
```

JSON

échanger des données avec du JavaScript

une simple structure de données
JavaScript dans une chaîne de caractère

Exemple

```
[{"nom": "philip",  
 "prenom": "julien",  
 "mails": [  
     "julien.philip@unice.fr",  
     "julien.philip@inria.fr"  
 ]}
```

JSON

JSON.parse() pour décoder
JSON.stringify() pour encoder

```
var message = "[{"type": "foo", "values": [1,2,3]}]"  
var x = JSON.parse(message)  
  
console.log(x.type + " " + x.values[0]);  
//affiche le foo 1  
  
var msg2 = JSON.stringify(x)  
//msg2 == message
```

Debugger l'envoi de requête

```
console.log()  
  
onglet 'network' dans la console du navigateur pour  
afficher les requêtes envoyées  
  
postman: un des plugins pour chrome pour exécuter  
des requêtes HTTP
```

Ca vaut encore le coup de générer du contenu
affichable côté serveur ? Non parce que
JavaScript ça tabasse quand même et en plus
on allège la charge du serveur !

Une tendance pour les (très très) gros sites

côté serveur

- envoie de pages statique
- interaction par une approche Ajax

côté client

- génération de toute l'interface
- interaction avec le serveur

Exemple

```
{  
    'nom':'philip',  
    'prenom':'julien',  
    mails:[  
        'julien.philip@unice.fr',  
        'julien.philip@inria.fr'  
    ]  
}
```

JSON

JSON.parse() pour décoder
JSON.stringify() pour encoder

```
var message = "{‘type’ : ‘foo”,values’: [1,2,3]}"  
var x = JSON.parse(message)
```

```
console.log(x.type + “ + x.values[0]);  
//affiche « foo 1 »
```

```
var msg2 = JSON.stringify(x)  
//msg2 == message
```

Debugger l'envoi de requête

console.log()

onglet '*network*' dans la console du navigateur pour afficher les requêtes envoyées

postman: un des plugins pour chrome pour executer des requêtes HTTP

Ca vaut encore le coup de générer du contenu affichable côté serveur ? Non parce que JavaScript ça tabasse quand même et en plus on allège la charge du serveur !

Une tendance pour les (très très) gros sites

côté serveur

- envoie de pages statique
- interaction par une approche Ajax

côté client

- génération de toute l'interface
- interaction avec le serveur