Cécile HARDEBOLLE
cecile.hardebolle@supelec.fr

Crédit : Joyent, Inc.

Prérequis

- Pratique de JavaScript

- Compréhension de l'architecture client-serveur web
  - rôle du client vs. rôle du serveur
  - protocole HTTP

Pour les exercices :

- Utilisation d'une base de données
- Ecriture de requêtes SQL

Objectifs

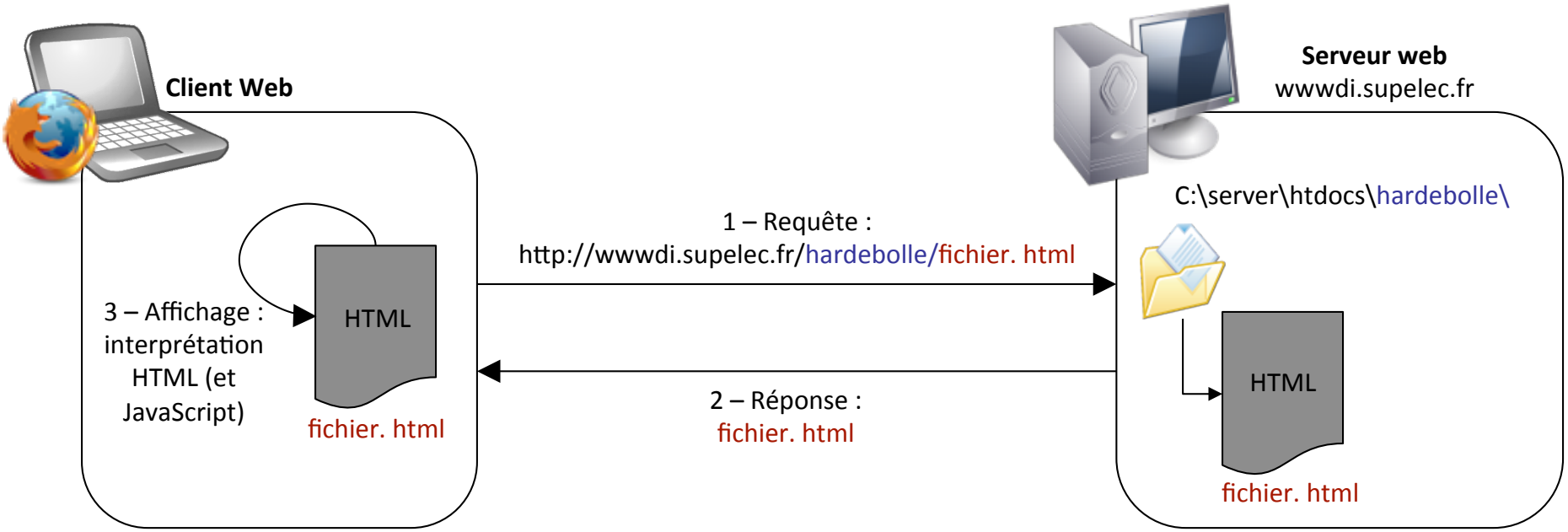
Etre capable de :

- Ecrire un serveur JavaScript répondant à des requêtes HTTP
- Ecrire une application Web (ou un web service) répondant avec des données formatées ou des pages web

Introduction

Rappel : site web « statique »

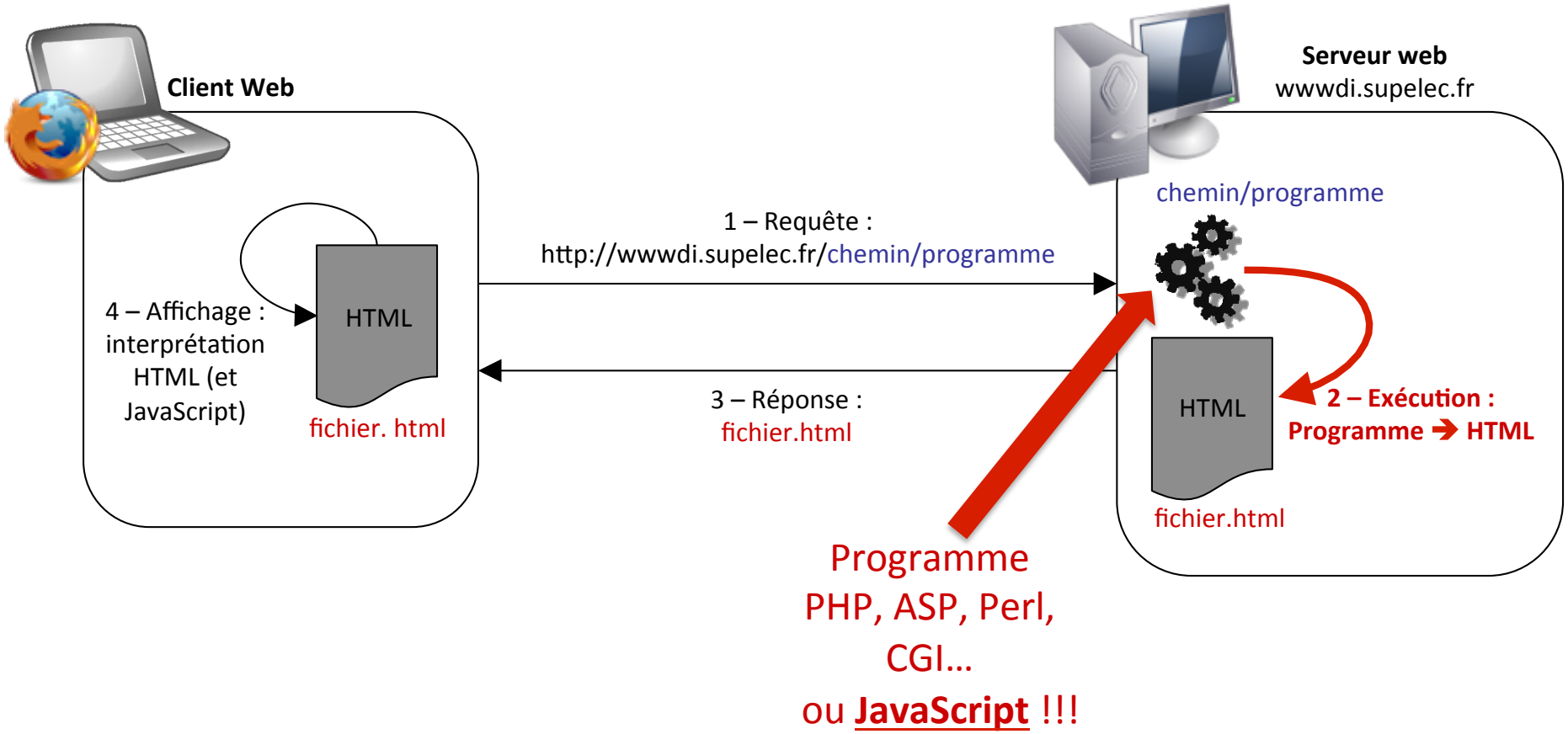
- Requête/réponse HTTP pour *récupérer une page HTML*



- Pour tester, les navigateurs savent aussi ouvrir des pages sur disque sans passer par HTTP

Application web / site web « dynamique »

- Requête/réponse HTTP pour *appeler un programme*
- Le programme génère dynamiquement la page HTML



Qu'est-ce que Node.js ?

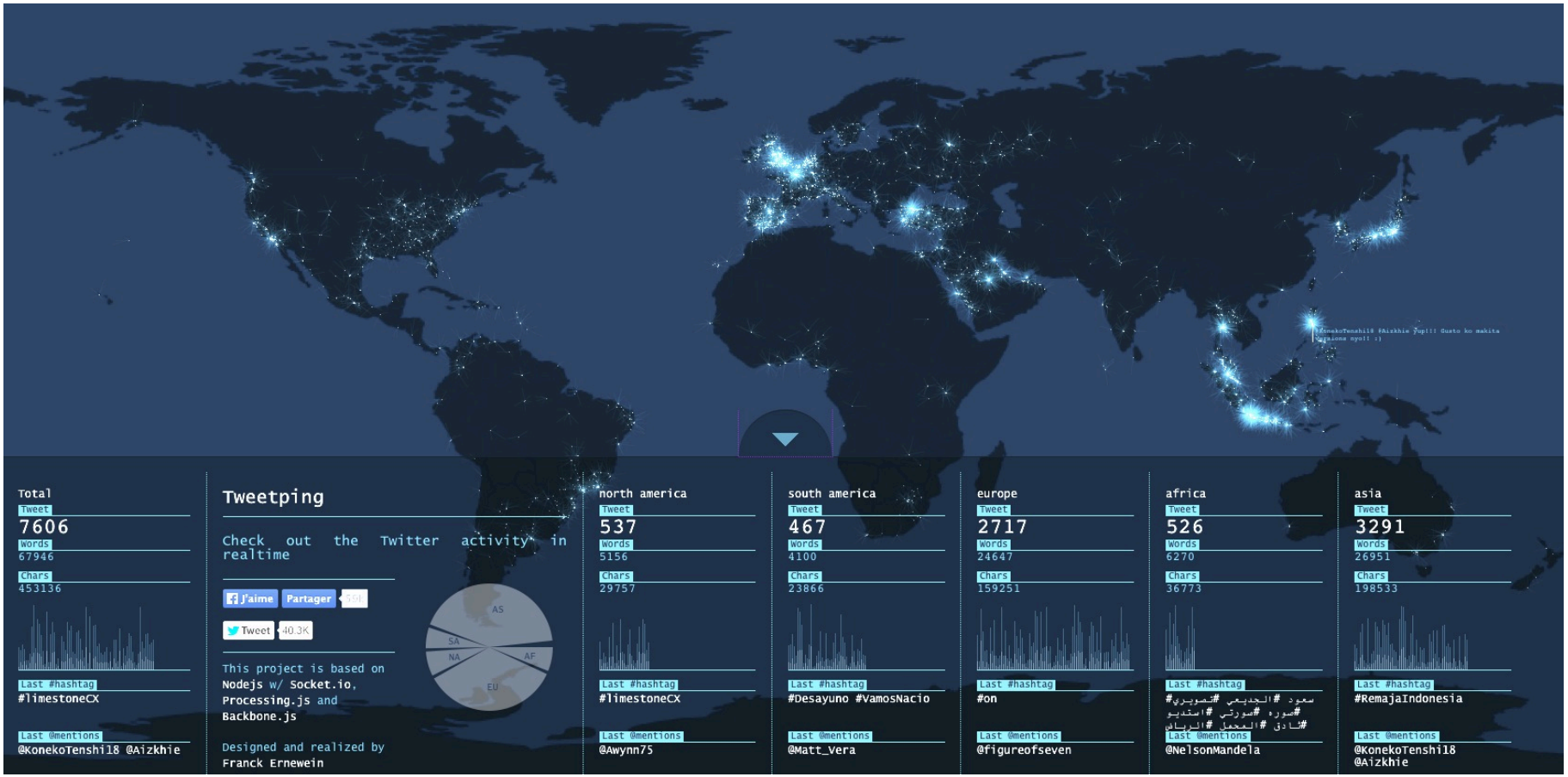
- Node.js = une plateforme d'exécution JavaScript basée sur le moteur d'exécution JavaScript V8 de Google
  + une API JavaScript

    ☛ pour créer des applications JavaScript côté serveur répondant sur le réseau (HTTP, sockets…)

- Créé en 2009 par Ryan Dahl (Joyent Inc.) et écrit en C++

- Grands principes :
  - exécution pilotée par les événements
  - appels de fonctions asynchrones (utilisation de callbacks)
  - entrées/sorties non bloquantes

Un exemple d'utilisation : tweetping.net



Dans le vif du sujet !

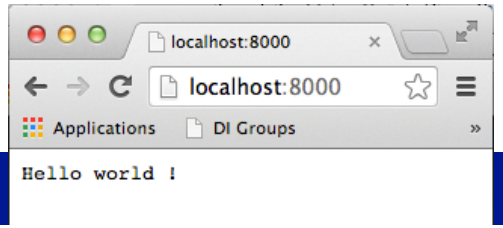
Premiers pas : un serveur web avec Node.js

Fichier « mywebserver-v1.js »

```
var http = require('http');

var serv = http.createServer( //création d'un serveur web

  function (req, res) { //callback sur les requêtes HTTP
      //construction d'une réponse HTTP
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.write('Hello world !');

      res.end(); //envoi de la réponse
  }

);

serv.listen(8000); //commence à accepter les requêtes

console.log('Server running at http://localhost:8000');
```

- Démarrer le serveur puis envoyer une requête HTTP :

  node mywebserver-v1.js



HTTP (HyperText Transfer Protocol)

- HTTP = protocole de communication de type
  requête/réponse synchrone

- Requête = méthode HTTP + URL + en-tête + corps
  - Principales méthodes HTTP :
    GET (*lecture*), POST (*création*),
    PUT (*modification*), DELETE (*suppression*)…

- Réponse = code de statut HTTP + en-tête + corps
  - Principaux statuts HTTP :
    200 (*OK*), 400 (*Bad request*), 404 (*Not f*ound)
    401 (*Unauthorized*), 403 (*Forbidden*),
    301 (*Moved permanently*), 308 (*Permanent redirect*),
    500 (*Internal server error*)…

Exemple de requête/réponse HTTP

Client Web

```
GET /index.html HTTP/1.1
Host: www.supelec.fr
```

Serveur web
www.supelec.fr

```
HTTP/1.1 200 OK
Date                Mon, 16 Dec 2013 12:40:37 GMT
Server              Apache
Set-Cookie          PHPSESSID=14446acd6ff800328bf3170…
Expires             Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control       no-store, no-cache, must-revalidate, …
Pragma              no-cache
Vary                Accept-Encoding
Keep-Alive          timeout=15, max=100
Connection          Keep-Alive
Transfer-Encoding   chunked
Content-Type        text/html; charset=ISO-8859-1
```

Type MIME

Comment « voir » le contenu des échanges HTTP ?

- http://web-sniffer.net/
- plugin REST client Chrome ou Firefox
- application java RESTClient de WizTools.org

MIME (Multipurpose Internet Mail Extension)

- MIME = standard définissant le type et l'encodage de contenus échangés sur internet
  - Utilisé pour les emails avec le protocole SMTP
  - Utilisé pour le web avec le protocole HTTP

- Indication du type de contenu :
  `Content-Type: type/sous-type ; charset=jeuDeCaractères`
  - Exemple de types de contenu :
    `text/plain`, `text/html`, `text/xml`, `application/json`, `application/sql` `audio/mpeg`, `image/jpeg`, `application/pdf`
  - Exemples de jeux de caractères :
    `UTF-8`, `ISO-8859-1`, `ASCII`

Fonction de callback

- Fonction de callback associée à un traitement = fonction appelée lors d'un événement lié au traitement

- Exemple avec `createServer` : callback appelé lorsque et à chaque fois que, une requête HTTP est reçue

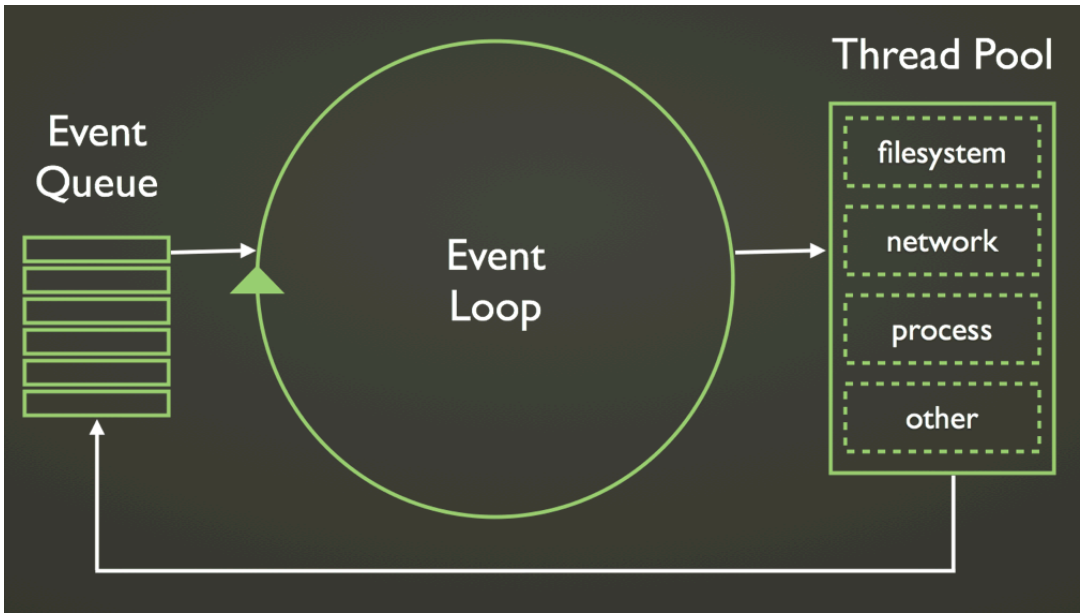
```
var serv = http.createServer(
  function(req, res) { //callback appelé sur requête HTTP
    …
  }
);
```

- Un autre exemple avec `setTimeout` : callback appelé lorsque le temps s'est écoulé

```
setTimeout(
  function() { //callback appelé à la fin du délai de 2s
    …
  }, 2000);
```

Boucle de traitement des événements

- Un seul programme (un seul thread) traite les événements dans l'ordre de leur occurrence
- Les traitements qui concernent les entrées/sorties sont exécutées de manière asynchrone par d'autres threads
  = « non blocking I/O »



Crédit : J. Kunkle, Node.js explained

Exemple avec deux instructions

Instruction **n°1**

```
setTimeout(
  function() { //callback sur le délai de 2s
    console.log("fin de l'attente");
  }, 2000);
```

Fonction appelée **à t_0+2s**

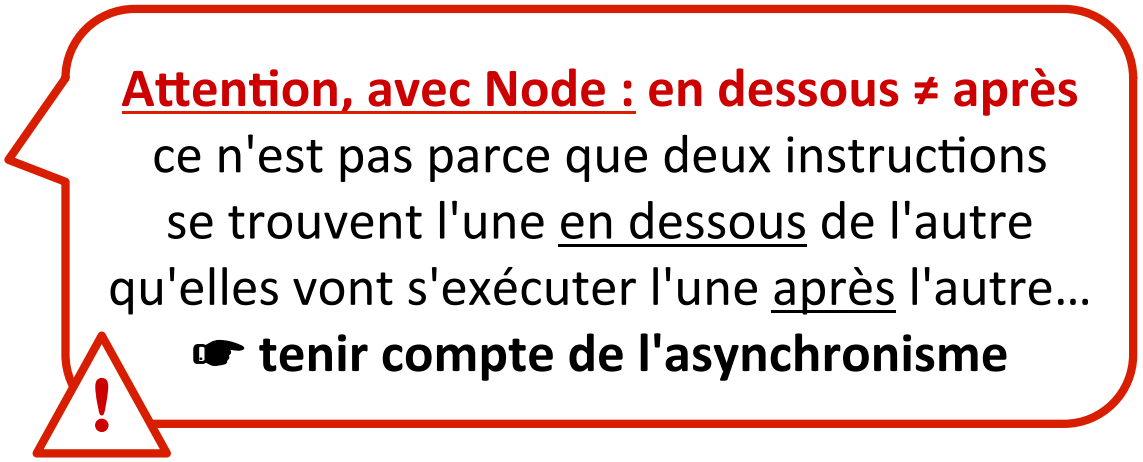
Instruction **n°2**

```
console.log("bonjour !");
```

Fonction appelée **à t_0**

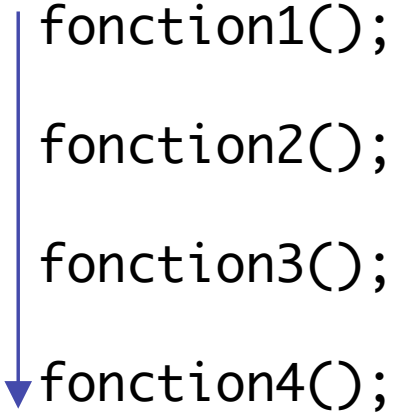
- Résultat :

```
bonjour !
fin de l'attente
```



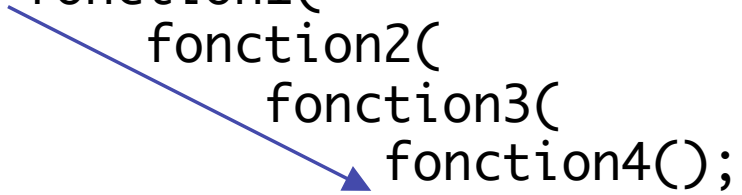
Un programme séquentiel ?

- Avec JavaScript :
  ☛ *synchrone*



```
fonction1();

fonction2();

fonction3();

fonction4();
```

- Avec JavaScript + Node.js :
  ☛ *asynchrone*



```
fonction1(
    fonction2(
        fonction3(
            fonction4();
        );
    );
);
```

La fonction passée en callback est appelée **une fois que la tâche de la fonction précédente est terminée**

Un serveur web plus évolué

Fichier « mywebserver-v2.js »

```
var http = require('http');
var fs = require('fs');

http.createServer(function (req, res) {

  fs.readFile(__dirname+'/page.html', function(err, data) {
    if (err) {
      res.writeHead(500, err.message);
      res.end();
    } else {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
    }
  });

}).listen(8000);

console.log('Server running at http://localhost:8000');
```

Des bibliothèques qui facilitent la vie…

- En standard dans Node :
  - http : pour communiquer via le protocole HTTP
  - fs : pour interagir avec le système de fichiers
  - timer : pour schéduler des traitements
  - crypto : pour chiffrer/déchiffrer des données

- A installer en plus avec le Node Package Manager (NPM) :
  - async : helpers pour faciliter la gestion d'appels asynchrones
  - express : pour construire des applications web
  - sqlite3 : pour connecter à une base de données SQLite
  - mongoose : pour connecter à une base de données MongoDB
  - persist : pour réaliser un mapping relationnel/objet

  …

Quelques exemples avec Express

Notre serveur web avec Express

Fichier « myexpresswebserver-v1.js »

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.sendfile(__dirname+'page.html');
});

app.listen(8000);
console.log('App listening on port 8000...');
```

- Répond à un GET sur http://localhost:8000/

Avec plusieurs « routes »

Fichier « myexpresswebserver-v2.js »

```
var express = require('express');
var app = express();

app.get('/accueil', function(req, res) {
  res.sendfile(__dirname+'page.html');
});

app.get('/autre', function(req, res) {
  res.sendfile(__dirname+'autrepage.html');
});

app.listen(8000);
console.log('App listening on port 8000...');
```

- Répond à un GET sur http://localhost:8000/accueil
- Répond à un GET sur http://localhost:8000/autre

D'autres méthodes HTTP sont possibles !

Avec des données dans une requête

Fichier « myexpresswebserver-v3.js »

```
var express = require('express');
var app = express();

app.get('/autre', function(req, res) {
  if("donnee" in req.query) {
    var txt = "Donnee reçue :" + req.query.donnee;
    res.send(txt);
  } else {
    res.sendfile(__dirname+'/autrepage.html');
  }
});

app.listen(8000);
console.log('App listening on port 8000...');
```

- Répond à un GET sur http://localhost:8000/autre?donnee=truc

Conclusion

En résumé

- Node.js permettant de créer des applications JavaScript côté serveur répondant sur le réseau (HTTP, sockets…)

- Node.js est :
  - multi-plateformes
  - léger
  - rapide
  - sans deadlocks

- Node.js à des particularités importantes :
  - exécution pilotée par les événements
  - appels de fonctions asynchrones (utilisation de callbacks)
  - entrées/sorties non bloquantes