

# W31

## Programmation web côté serveur

---

Pierre Kraemer

[kraemer@unistra.fr](mailto:kraemer@unistra.fr)

# Node.JS

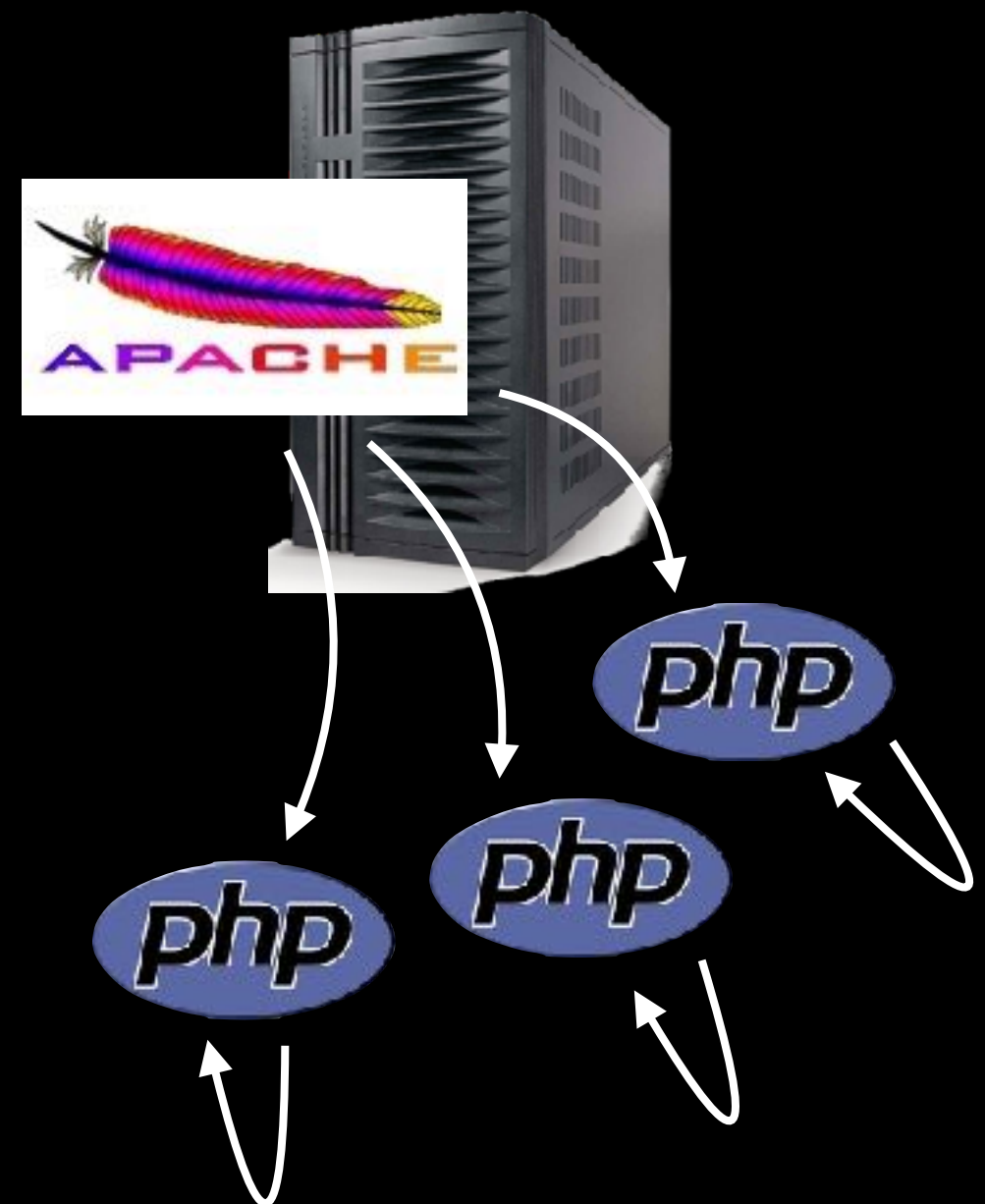
- JavaScript côté serveur
- basé sur le moteur V8 de Chrome
- exécution événementielle non-bloquante dans un seul thread

# PHP

Client  
navigateur web



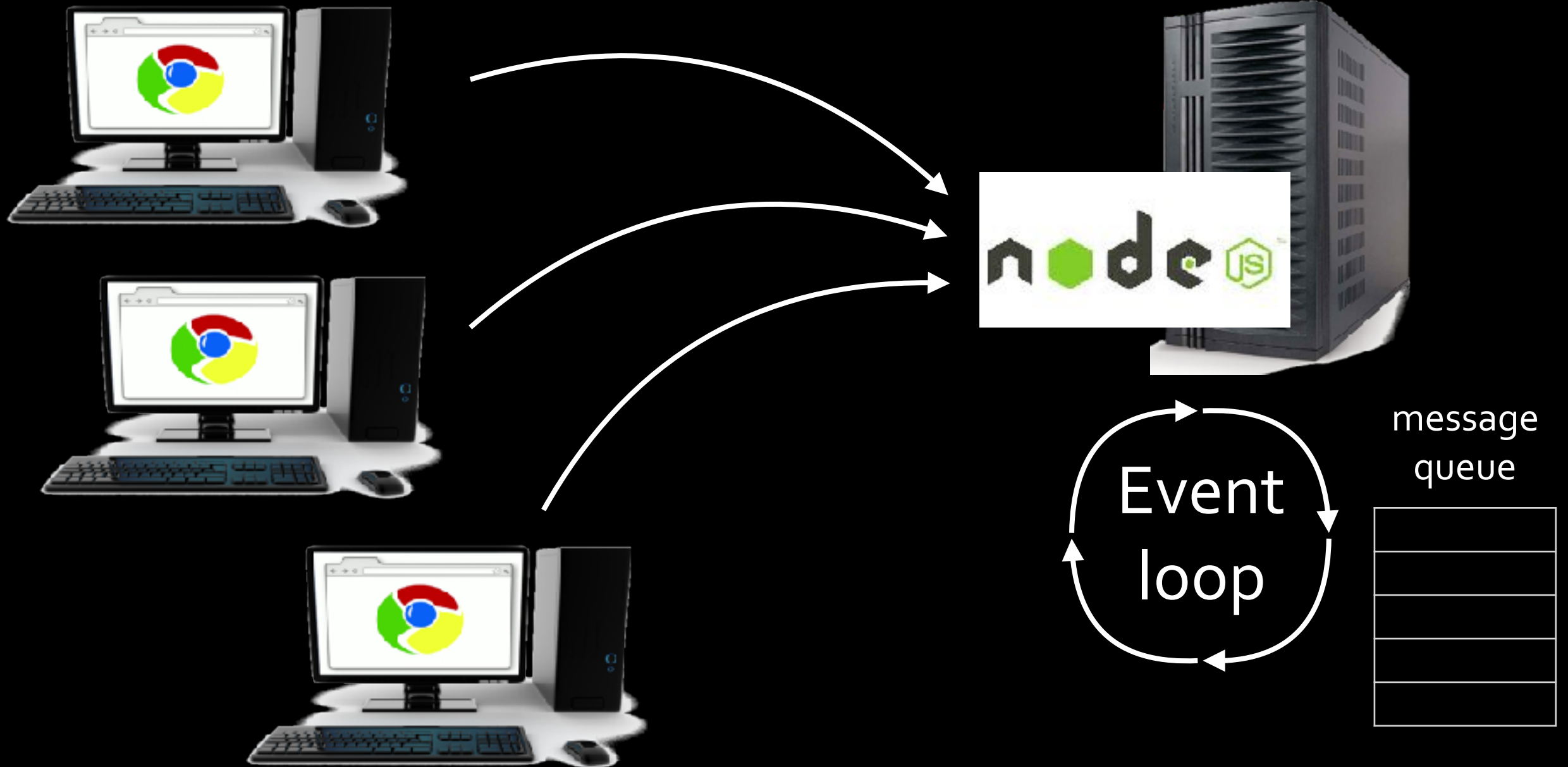
Serveur  
serveur HTTP



# Node.JS

Client  
navigateur web

Serveur  
serveur HTTP



# Premier exemple

quand le serveur  
émet l'événement  
'request'

charge le  
module 'http'

crée un objet  
serveur HTTP

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

```
const server = http.createServer();
```

```
server.on('request', function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.write('Hello Node.JS !');  
  res.end();  
});
```

```
server.listen(8008);
```

on fait appel à  
cette fonction  
(callback)

démarre l'écoute  
du serveur sur le  
port 8008

# Premier exemple

objet  
IncomingMessage

objet  
ServerResponse

```
function (req, res) {  
  res.writeHead(200, { 'Content-Type': 'text/plain' });  
  res.write('Hello Node.JS !');  
  res.end();  
}
```

informe le serveur  
de la fin de la  
réponse

ajoute des  
données à la  
réponse

écrit l'en-tête de la  
réponse

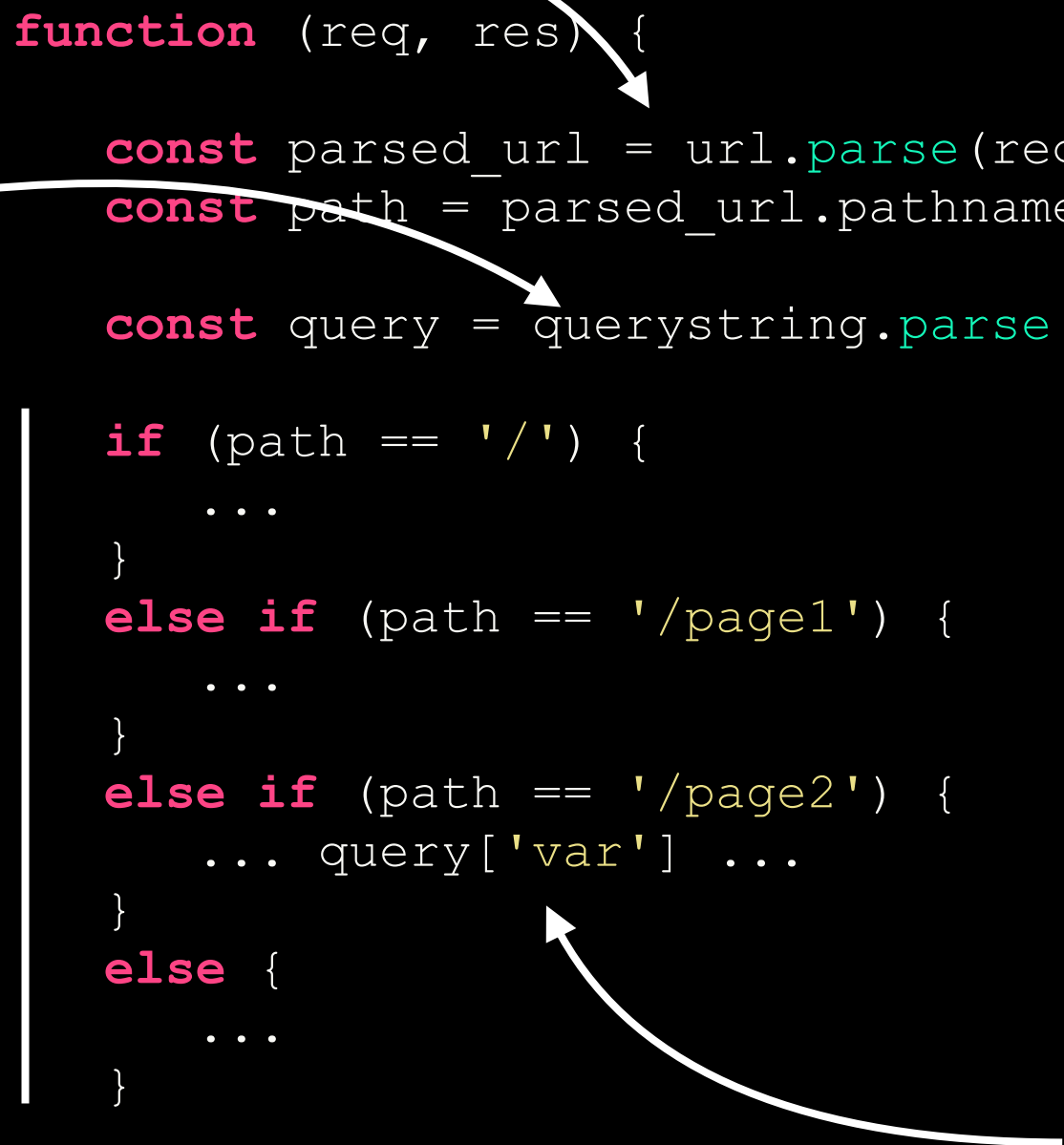
# Router les requêtes

module qui analyse  
l'url de la requête

module qui analyse  
la query string

on fait des  
réponses  
différentes  
selon l'url  
demandée

```
function (req, res) {  
  const parsed_url = url.parse(req.url);  
  const path = parsed_url.pathname;  
  
  const query = querystring.parse(parsed_url.query);  
  
  if (path == '/') {  
    ...  
  }  
  else if (path == '/page1') {  
    ...  
  }  
  else if (path == '/page2') {  
    ... query['var'] ...  
  }  
  else {  
    ...  
  }  
  
  res.end();  
}
```



accès aux  
variables de la  
query string

# Express

crée un objet  
express

charge le module  
'express'

```
const express = require('express');
```

```
const app = express();
```

appelle le  
middleware  
suivant

```
app.use(function (req, res, next) {  
  ...  
  next();  
});
```

définit des  
middlewares  
qui seront  
exécutés les  
uns après les  
autres

```
app.use('/mount/point', function (req, res, next) {  
  ...  
  next();  
});
```

middleware appelé  
uniquement pour les  
url correspondant à  
cette route

```
app.use(function (req, res, next) {  
  ...  
  next();  
});
```

```
app.listen(8008);
```

lance un serveur qui  
écoute sur le port spécifié



# Express

un premier  
middleware pour  
loguer les  
requêtes

```
app.use(function (req, res, next) {  
  console.log(req.method + ':' + req.url);  
  next();  
});
```

n'exécute le  
middleware que  
pour le verbe  
HTTP donné

```
app.get('/page1', function (req, res) {  
  ...  
});
```

```
app.post('/page1', function (req, res) {  
  ...  
});
```

```
app.get('/page2', function (req, res) {  
  ... req.query.val ...  
});
```

accès aux  
variables de la  
query string

ajout de  
paramètres dans  
les routes

```
app.get('/user/:id', function (req, res) {  
  ... req.params.id ...  
});
```

accès aux valeurs  
des paramètres

# Template engine

Syntaxe qui décrit la manière d'intégrer des données dans les vues

(jusqu'à présent on utilisait PHP pour cela)

Mustache.js

Swig

EmbeddedJS

JSRender

Markup.js

Jade

dom.js

# Mustache.js

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="utf-8">
  <title> {{ title }} </title>
</head>

<body>
  {{ #notes }}

    <article>
      <h1> {{ title }} </h1>
      <p> {{ text }} </p>
      <p> {{ author }} </p>
    </article>

  {{ /notes }}
</body>

</html>
```

insertion de la  
valeur d'une  
variable

sections  
(ici, itération sur un  
tableau d'objets)

accès aux champs  
d'objets

# Mustache.js depuis Express

configure  
Express pour  
utiliser  
Mustache.js

```
const mustache = require('mustache-express');
```

```
...
```

```
app.engine('mustache', mustache());  
app.set('view engine', 'mustache');  
app.set('views', __dirname + '/views');
```

```
...
```

```
app.get('/', function (req, res) {  
  res.render('home', {  
    title: 'Home'  
  });  
});
```

```
app.get('/notes', function (req, res) {  
  res.render('notes', {  
    notes: [ ... ]  
  })  
});
```

charge le module  
'mustache-express'

charge une vue en  
passant des  
données

# Modules

## my-module.js

```
const tab = [];  
  
function addElement(el) {  
  tab.push(el);  
}  
  
function getElement(idx) {  
  return tab[idx];  
}  
  
module.exports = {  
  addElement,  
  getElement  
};
```

interface  
publique du  
module



singleton



```
const myModule = require('./my-module');  
  
myModule.addElement(12);  
myModule.addElement(23);  
  
const i = myModule.getElement(0);
```

# Socket.IO

- module Node.JS
- liaison bi-directionnelle temps réel entre clients et serveur
- utilise des technologies différentes selon les capacités de la plateforme d'exécution

# Socket.IO (serveur)

initialise le  
serveur

```
...  
const app = express();  
  
const server = http.createServer(app);  
const io = require('socket.io').listen(server);
```

quand un  
client se  
connecte

```
...  
io.on('connection', function (socket) {  
    let name = '';
```

cette socket  
permet de  
contacter ce client

quand on  
reçoit un  
message

```
    socket.on('hello', function (n) {  
        name = n;  
        io.emit('client connected', name);  
    });  
  
    socket.on('disconnect', function () {  
        io.emit('client disconnected', name);  
    });
```

émet un message à  
tous les autres clients

```
});
```

```
server.listen(8008);
```

# Socket.IO (client)

```
...  
const app = express();  
  
const server = http.createServer(app).listen(8008);  
const io = require('socket.io').listen(server);  
...  
  
...  
    const name = 'pierre';  
io.on('connection', function (socket) {  
    socket.on('connect', function () {  
        let name = '';  
        socket.emit('hello', name);  
    });  
    socket.on('hello', function (n) {  
        name = n;  
        socket.on('client connected', function (name) {  
            io.emit('console.log', 'new connection -> ' + name);  
        });  
    });  
    socket.on('disconnect', function (name) {  
        io.emit('console.log', 'disconnected -> ' + name);  
    });  
});  
  
server.listen(8008);
```

établit la connexion

une fois la connexion établie

émet un message au serveur

quand on reçoit un message




# npm

*node package manager*

## package.json

```
{
  "name": "MyApp",
  "version": "0.0.1",
  "description": "My application is great",
  "author": {
    "name": "Pierre Kraemer"
  },
  "private": true,
  "dependencies": {
    "express": "^4.15.4",
    "mustache-express": "^1.2.5",
    "body-parser": "^1.17.2"
  }
}
```

nom et version des  
modules dont  
dépend l'application



# npm

```
$> npm install
```

- app.js
- package.json
- node\_modules/
  - | - express/
  - | - mustache-express/
  - | - body-parser/

[www.npmjs.org](http://www.npmjs.org)