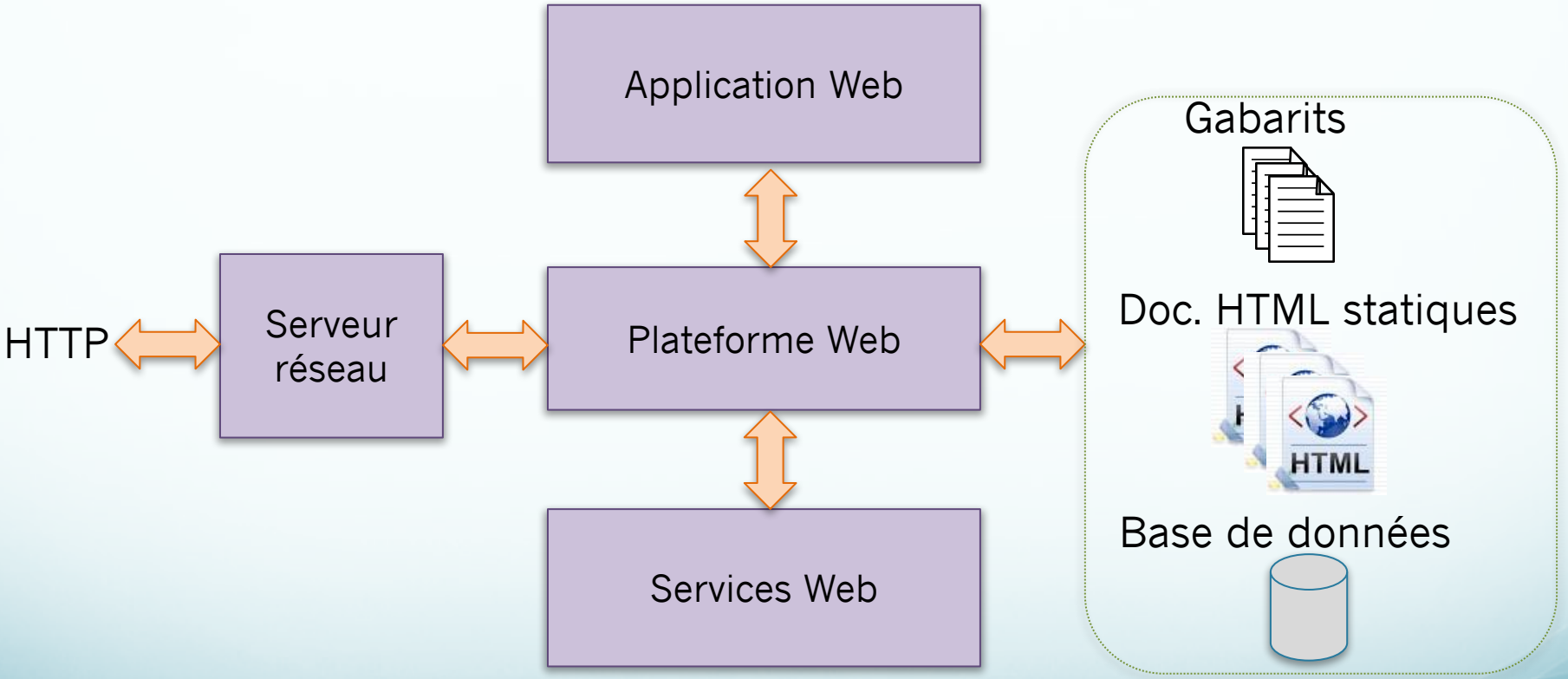
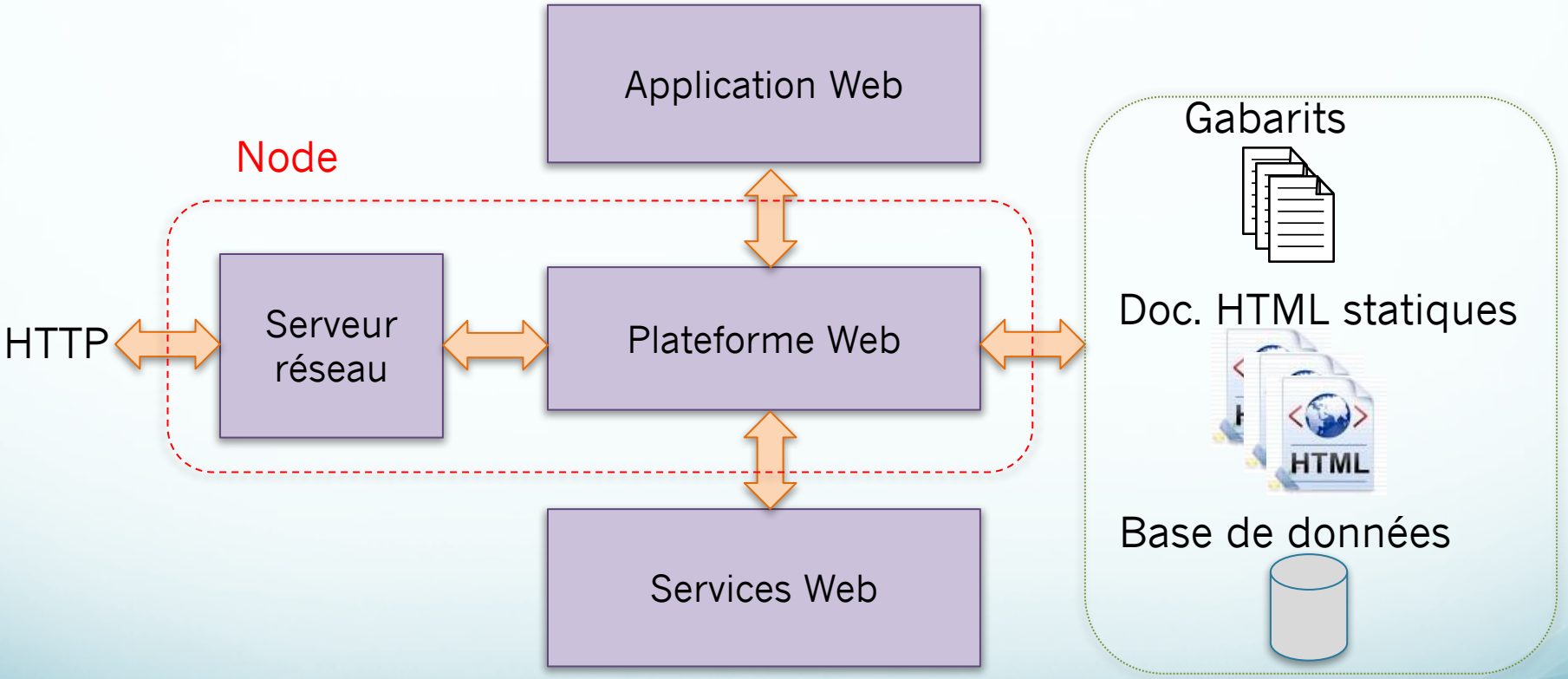
Serveur

Michel Gagnon
École polytechnique de Montréal

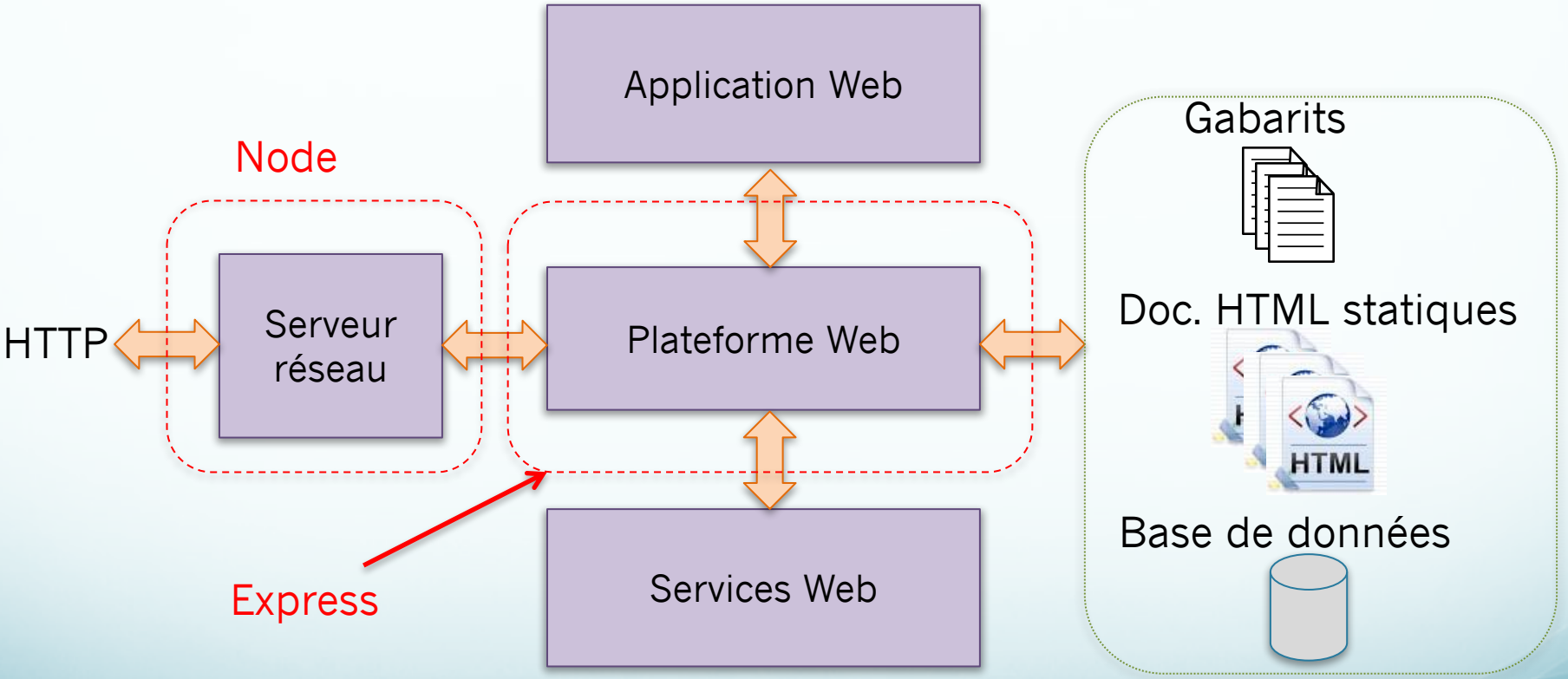
Architecture d'un serveur



Architecture d'un serveur



Architecture d'un serveur



Node

- Pas vraiment de distinction entre serveur et application web
- Programmation par événements
- S'exécute sur un seul thread
- On programme en Javascript, compilé à la volée (just-in-time)
- Indépendant du système d'exploitation utilisé
- Routage pour faire le lien entre une requête et le module qui produit le contenu désiré

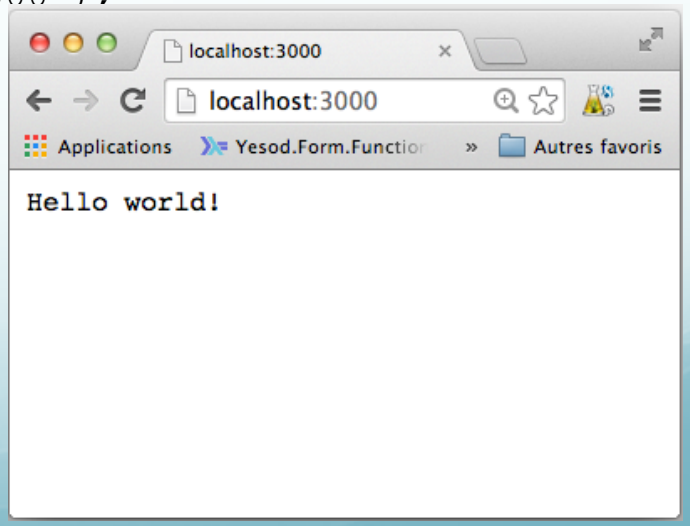
Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur localhost:3000');
```



Node – exemple simple

Les fonctionnalités de Node sont regroupées dans des modules (chaque module est un objet)

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur localhost:3000 ');
```

Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur localhost:3000 ');
```

On crée un serveur.
On lui passe une fonction qui sera exécutée pour chaque requête reçue.

Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur loc
```

Le paramètre **res** est un objet de type http.ServerResponse.

Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur loc
```

Cette méthode est appelée pour spécifier le code de la réponse et ses en-têtes.

Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur loc
```

Cette méthode est appelée pour envoyer la requête, tout en spécifiant le contenu qui doit y être intégré.

Node – exemple simple

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

var serveur =
   http.createServer(function(req,res){
     res.writeHead(200, { 'Content-Type': 'text/plain' });
     res.end('Hello world!');
   })

serveur.listen(3000);

console.log('Serveur démarré sur localhost:3000 ');
```

Démarrage du serveur

Express

- Offre un outil d'« échafaudage »
- Ajoute une couche sur Node pour simplifier notre tâche de développeur
- Basé sur une pile de middlewares
- Un middleware est une fonction prenant trois arguments: la requête (**req**), la réponse (**res**), et un objet (**next**) représentant le prochaine middleware à être exécuté dans la pile
- On peut aussi définir un middleware qui, en plus des trois arguments cités précédemment prend comme premier argument un objet, qui sera défini si on a une situation d'erreur
- Les routeurs sont des cas particuliers de middlewares

Express – exemple simple

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

var app = express();
app.set('port', 3000);

app.get('/', function(req, res){
    res.type('text/plain');
    res.send('Les amateurs de jazz');
});
app.get('/about', function(req, res){
    res.type('text/plain');
    res.send('À propos de nous');
});

// Page 404
app.use(function(req, res){
    res.type('text/plain');
    res.status(404);
    res.send('404 - Not Found');
});
```

Pile de middlewares

- Chaque middleware peut, à la fin de son traitement, demander qu'on passe au suivant en appelant **next()**
- Un appel à **res.end()**, **res.send()** ou **res.render()** termine la cascade d'exécution de middlewares
  - **send()** envoie directement au client le contenu qui lui est passé en paramètre
  - **render()** répond en utilisant les paramètres suivants:
    - le gabarit qui doit être utilisé
    - un objet qui pourra contenir des informations qui seront extraites par le gabarit

Pile de middlewares

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

// Pile de middlewares associée à une route
app.use('/about', function (req, res, next){
     console.log('About 1');
     next();
});
app.use('/about', function (req, res, next){
     console.log('About 2');
     res.send('About');
});

// Cette forme est équivalente à la première,
// sauf qu'on spécifie la méthode HTTP
app.get('/about2', function (req, res, next){
     console.log('About GET 1');
     res.send('About GET');
});

// Ce middleware ne sera jamais appelé
app.get('/about2', function (req, res, next){
     console.log('About GET 2');
     res.send('About GET');
});
```

```
// Pile de middlewares pouvant être exécutée
// pour toute requête
app.use(function (req, res, next){
    console.log('Premier middleware exécuté');
    next();
});
app.use(function (req, res, next){
    console.log('Deuxième middleware exécuté');
    res.send('Terminé');
});

app.listen(3000, function(){
   console.log(
    'Express started on http://localhost:3000');
})
```

Route

- Est indiquée par un chemin: **/home/users**
- On peut utiliser les symboles spéciaux ?, + et *:
  - **/home/*** (n'importe quoi peut suivre **/home/**)
  - **/home/*/users** (n'importe quoi entre les deux)
  - **/home/b?elle** (indique que le **b** est facultatif)
  - **/home/(non)?** (**non** est facultatif)
  - **/users/noo+n** (le 2e **o** peut apparaître 1 ou plusieurs fois)
- On peut utiliser une forme **:id** pourra être reprise dans le code (ce sera un attribut de **req.params**)
  - **/home/user/:id**
- On peut utiliser une expression régulière:
  - **(/data)|/users)/about/**
- On peut utiliser un tableau:
  - **['/info', '/allo(/about)?', '/h(i|o)p']**

Formulaires

- Il faut utiliser le middleware **body-parser**
- En fait, il s'agit de plusieurs middlewares, parmi lesquels ont doit choisir le parseur qui nous intéresse:
  - **app.use(bodyparser.urlencoded({extended:true}));**
  - **app.use(bodyparser.json());**
- Ce middleware ajoute un attribut **body** à la requête **req**, qui contiendra toutes les paires attribut/valeur envoyées par la soumission du formulaire (que ce soit par le protocole standard HTML ou Ajax)

Routeur

- Un routeur permet de combiner plusieurs middlewares dans un module
- C'est un peu comme si on définissait une mini-application
- On l'utilise en l'associant à une route:
  - app.use('/user/', monRouteur);
- Il est normalement défini dans un fichier à part

Routeur

app.js

```
var express = require('express');
var routeur1 = require('./routeur1');
var app = express();

app.use('/about',routeur1);

app.get('/about', function (req, res, next) {
    console.log('About 2');
    res.send('About');}
);
```

routeur1.js

```
var express = require('express');
router = express.Router();
router.get('/', function (req, res, next) {
    console.log('About 1');
    next();
});

module.exports = router;
```

Express - cookies

- Avec Express, on peut créer des cookies normaux et des cookies signés
- Cookie signé:
  - On lui ajoute une signature, qui est un encodage de son contenu utilisant une clé secrète
  - Lorsque le cookie est envoyé, on décode cette signature et on vérifie si le résultat obtenu correspond au contenu envoyé
- Pour créer des cookies, il faut utiliser le middleware **cookie-parser**
- On crée un cookie en appelant la méthode **cookie()** de l'objet **res** (on met {signed:true} comme 2e argument si on veut qu'il soit signé)
- On extrait les cookies par le biais de l'attribut **res.cookies** ou **res.signedCookies**

Cookies - Exemple

```
app.use(require('cookie-parser')('mon secret'));

app.get('/', function(req, res) {
        res.cookie('contenuNonSigne','John Lewis');
        res.cookie('contenuSigne','John Coltrane',{signed: true});
        res.render('main',{'body':'Do bi dou bi dou wap!'});
});

app.get('/about', function(req, res) {
    res.render('about',
               {'contenuCookieNonSigne':req.cookies.contenuNonSigne,
                'contenuCookieSigne':req.signedCookies.contenuSigne});
});
```

Express - Sessions

- Implémenté avec les cookies
- Il faut charger le middleware express-session
- Un attribut **session** est ajouté à l'objet **req**
- Initialement, cet attribut est associé à un objet vide
- Il suffit alors d'ajouter des infos à cet objet
- On peut accéder à cet objet dans n'importe quel middleware
- Par défaut, la session est gardée en mémoire vive

Session - Exemple

```
app.use(require('cookie-parser')('mon secret'));
app.use(require('express-session')());

app.get('/', function(req, res) {
    req.session.message = 'Bonjour, comment allez-vous';
    res.render('main',{'body':'Do bi dou bi dou wap!'});
});

app.get('/about', function(req, res) {
    res.render('about',{'message': req.session.message});
});
```

Modèle MVC

- La vue est un gabarit HTML:
  - Approche habituelle: basé sur un squelette HTML
  - Pug utilise une autre approche plus compacte et moins verbeuse
- Ce que permet normalement un modèle de gabarit:
  - interpolation
  - énoncés conditionnels
  - Itérations
  - combinaisons de plusieurs sources

Pug – Combinaison de vues

- On utilise **extends** pour réutiliser une autre vue
- La réutilisation est basée sur le concept de bloc:
  - La vue qu'on étend contient des blocs à des endroits définis
  - On a alors quatre possibilités:
    - On redéfinit un bloc, qui sera alors remplacé par le nouveau code Pug qu'on spécifie
    - On ajoute du code Pug au début du bloc (**prepend**)
    - On ajoute du code Pug à la fin du bloc (**append**)
    - On laisse le bloc tel quel

Pug - Exemple

```
doctype html
html
   head
      block hd
        title Les amateurs de jazz
   body
      block content
```

```
extends layout.pug

block content
   h1 Les amateurs de jazz
   p Ce site vous fera voir...
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Les amateurs de jazz</title>
  </head>
  <body>
     <h1>Les amateurs de jazz</h1>
    <p>Ce site vous fera voir le jazz comme
          vous ne l'avez jamais vu.</p>
  </body>
</html>
```

Pug - Exemple

```
doctype html
html
  head
    block hd
      title Les amateurs de jazz
  body
    block content
```

```
<!DOCTYPE html>
<html>
 <head>
  <title>Les amateurs de jazz</title>
  <link rel="stylesheet" href="style.css">
 </head>
 <body>
  <h1>Les amateurs de jazz</h1>
  <p>Ce site vous fera voir le jazz comme
      vous ne l'avez jamais vu.</p>
 </body>
</html>
```

```
extends layout.pug

block append hd
  link(rel='stylesheet',href='style.css')
block content
   h1 Les amateurs de jazz
   p Ce site vous fera voir...
```