Università della Svizzera italiana

Faculty of Informatics

Server-side JavaScript

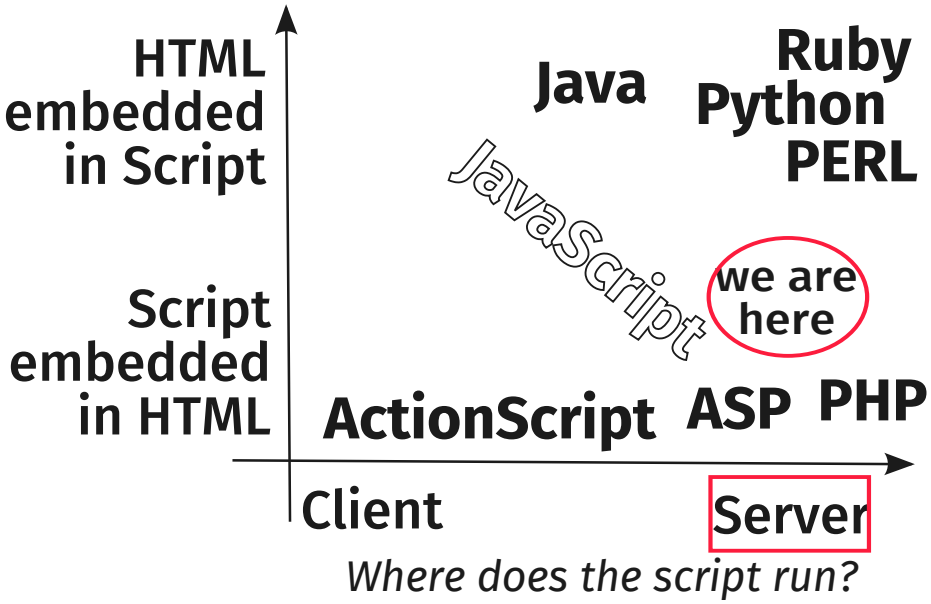
Prof. Cesare Pautasso

http://www.pautasso.info

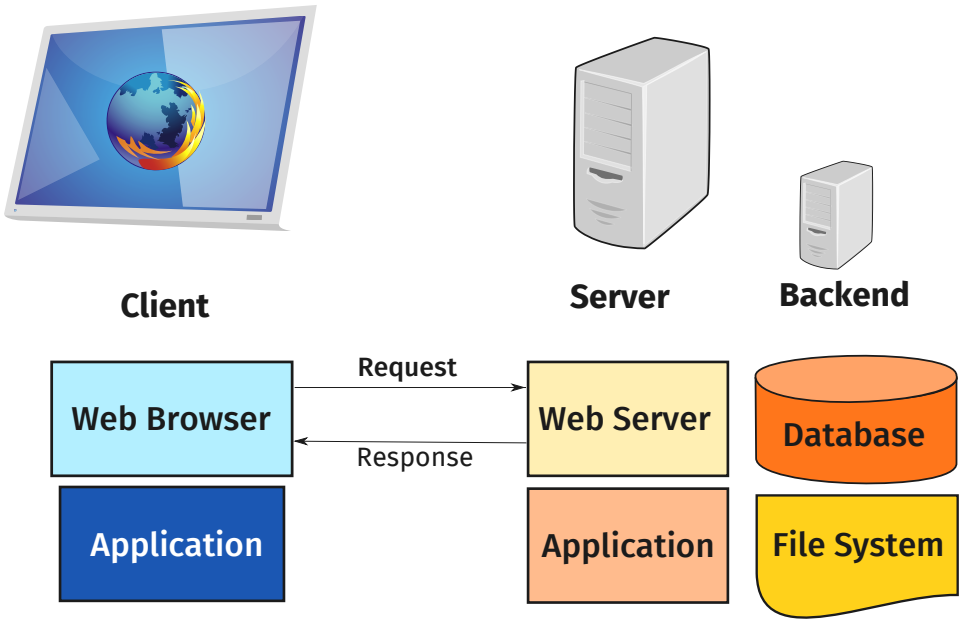
cesare.pautasso@usi.ch

@pautasso

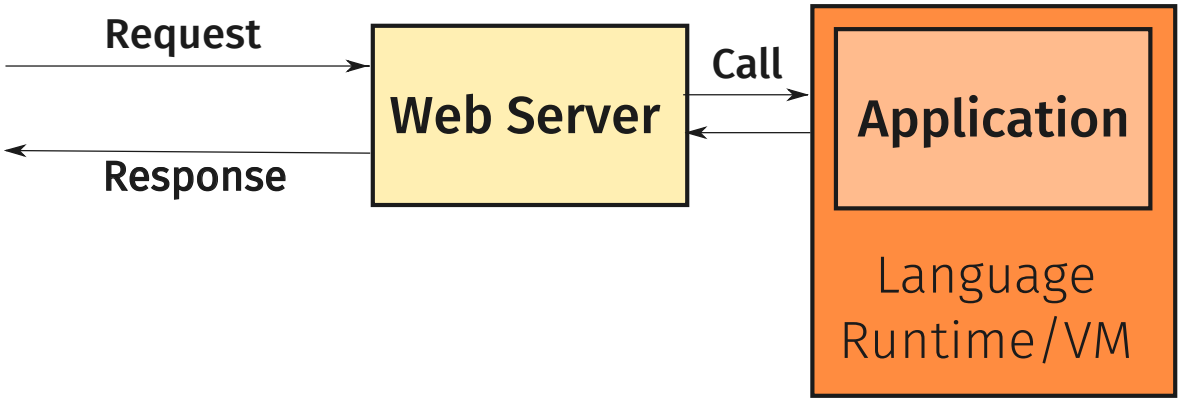
Where are we?

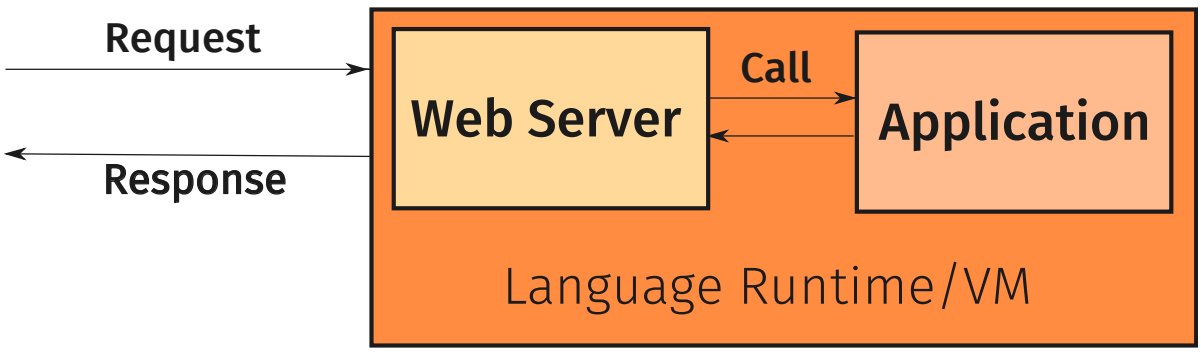


Web Architecture



Apps and Web Servers





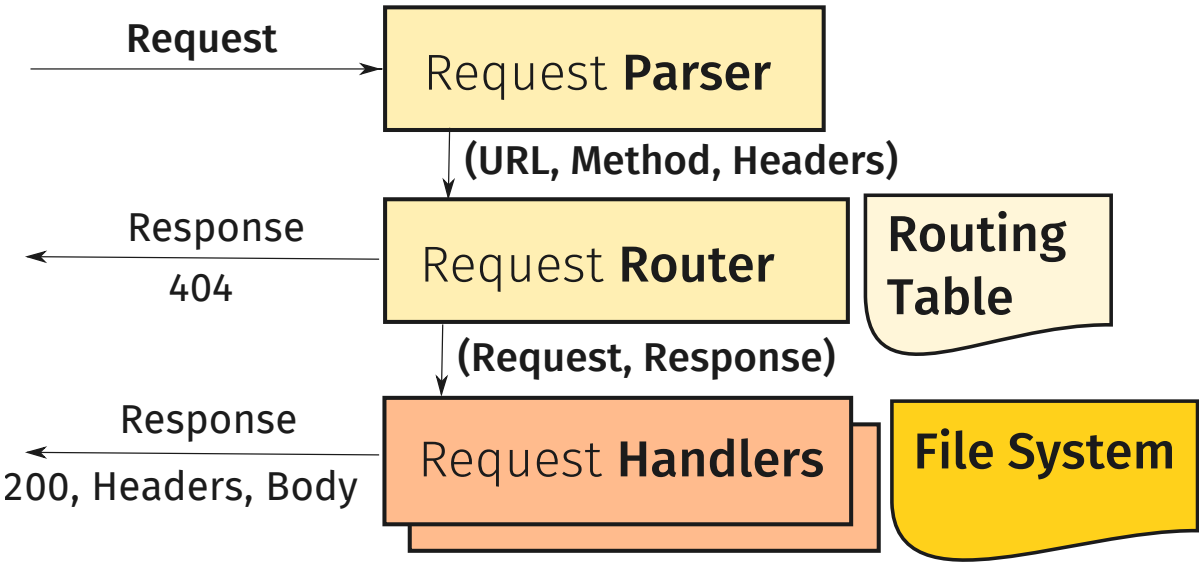
Hello World on the Server

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

function onrequest(request, response) {
  response.writeHead(200);
  response.end("Hello World\n");
}

var server = http.createServer(onrequest);
server.listen(8000);

console.log("Listening on http://127.0.0.1:8000/");
```


Server Programming

1. The server should always be ready for handling client requests
2. Requests should be answered as quickly as possible
3. One server can handle many types of requests

Web Server Architecture



Web Server in JavaScript

```
var http = require("http");
var url = require("url");
function onrequest(request, response) {
var pathname = url.parse(request.url).pathname;
 if (typeof rh[pathname] === 'function') {
   rh[pathname](request, response);
 } else {
   response.writeHead(404);
   response.end();
 }
}
http.createServer(onrequest).listen(8888);
```


Routing Request Handlers

```
//Request Handlers
var root = function(request, response) { … }
var hello = function(request, response) {
  response.writeHead(200,
              {"Content-Type": "text/html"});
  response.write("…"); //body
  response.end();
}
```

```
//Routing Table (Map URL -> Request Handler)
rh["/"] = root;
rh["/hello"] = hello;
```

Working with requests

```
request.url
request.method
request.headers
```


Working with URLs

```
var url = require(‘url’);
var p_url = url.parse('/status?name=cp', true);
```

```
p_url = {
  href: '/status?name=cp',
  search: '?name=cp',
  query: { name: ‘cp' },
  pathname: '/status'
}
```


Working with responses

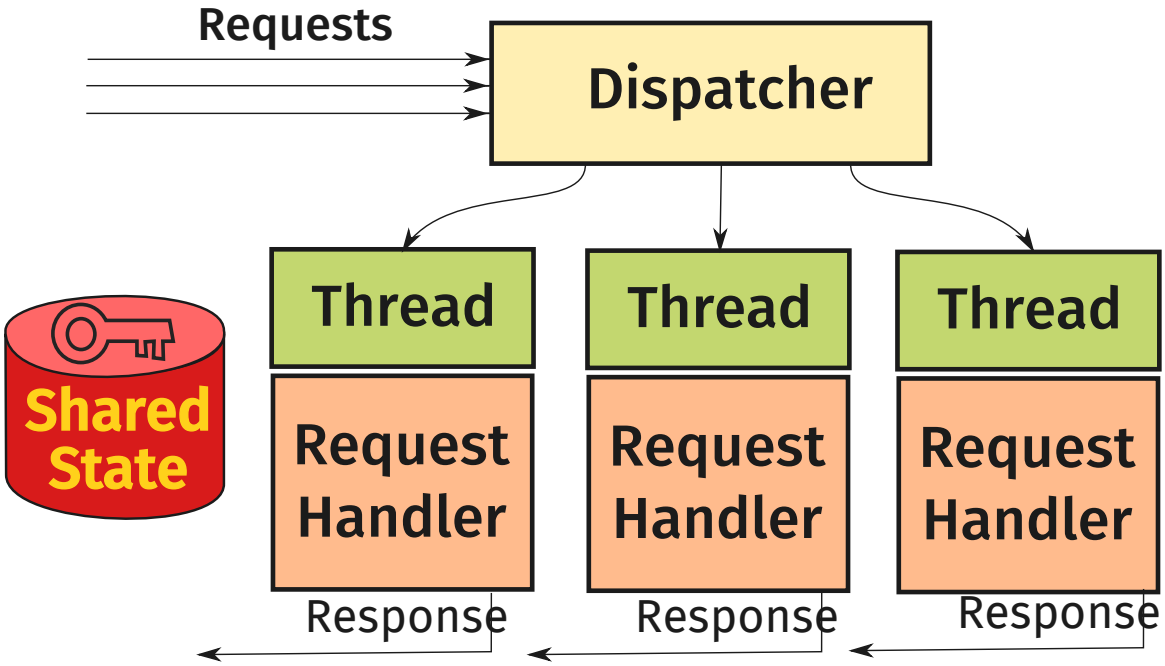
```
response.writeHead(statusCode, [msg], [headers]);
response.write(body);
response.end();
```

Explicit Headers

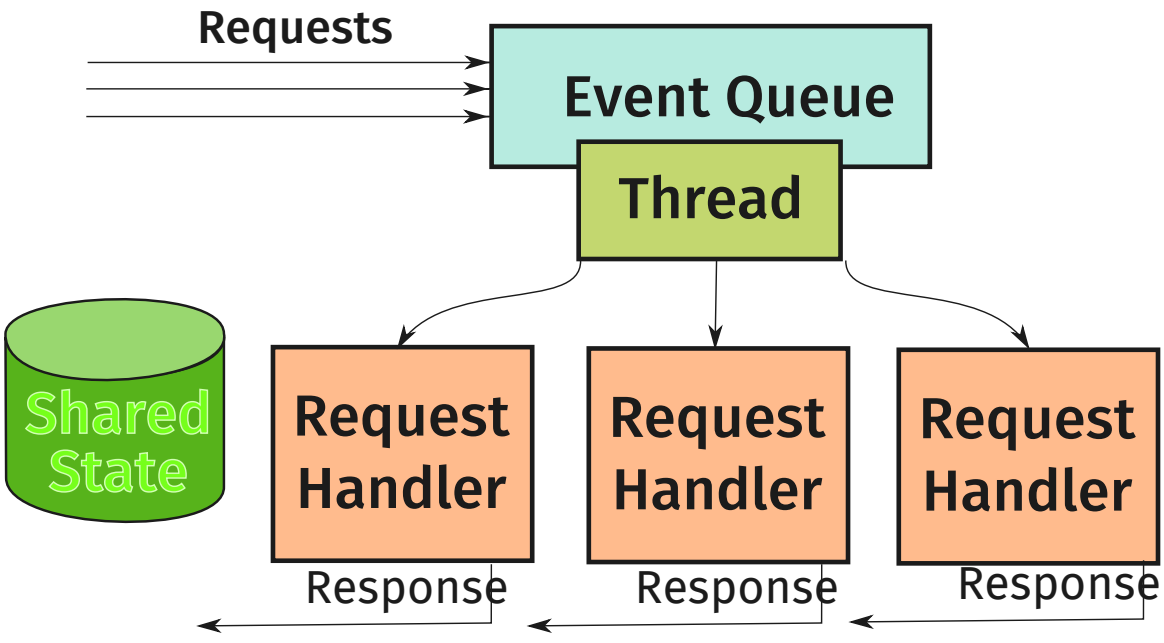
```
response.setHeader(key, value);
response.getHeader(key);
response.statusCode
response.write(body);
response.end();
```

Implicit Headers

Concurrency - Threads



Concurrency - Events



Events

- **Pro:** No synchronization needed, there is only one thread that reads/write variables shared between different request handlers
- **Con:** A slow request handler will block the entire server. All I/O operations need to be non-blocking

Threads

- **Con:** Shared variables between thread need to be protected from concurrent access
- **Pro:** Threads run in parallel, if one is slow/blocked, it will not delay all other pending requests (unless there is a shared lock)

Non-blocking callbacks

```
//blocking function
function(arguments)
var y = f(x);
...
```

Blocking functions return their result when they finish

```
//non-blocking function
function(arguments, callback)
f(x, function(y) { ... });
```

Non-blocking functions return immediately and use callbacks to pass the results later when it's ready

Composing callbacks

```
var y = f(x);
var z = g(y);
```

How to write this code using non-blocking functions?

Request Events

```
var postData = "";
request.setEncoding("utf8");
request.on("data",
  function(chunk) { postData += chunk; });
request.on("end",
  function() {
    console.log ("received" + postData);
  });
```

Read the body of a post request

Tools

- Node.JS (http://nodejs.org/)
- Node Toolbox (http://toolbox.no.de/) (Package/Extension directory)
- Heroku (http://www.heroku.com/) (Node.js cloud deployment)

References

- Node Beginner (http://www.nodebeginner.org/) (Node.js tutorial)
- Node School (http://nodeschool.io/#learn-you-node)
- Ryan Dahl, Node.JS: JavaScript on the Server (http://www.youtube.com/watch?v=F6k8lTrAE2g) , Google Tech Talk, July 2010,
- Douglas Crockford, JavaScript: The Good Parts, O'Reilly, May 2008
- Mike Amundsen, Building Hypermedia APIs with HTML5 and Node, O'Reilly, 2011