Università della Svizzera italiana

Faculty of Informatics

Server-side JavaScript

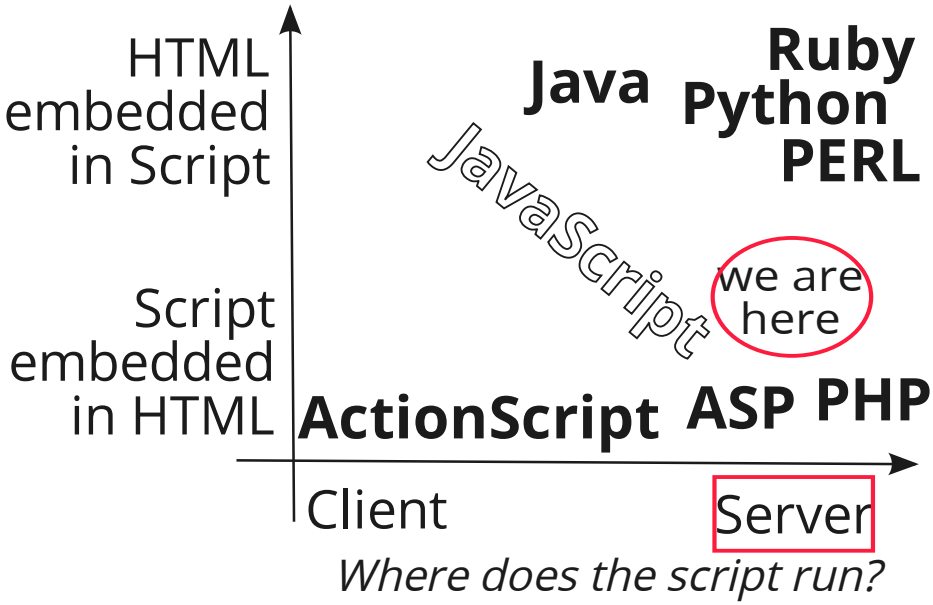
Prof. Cesare Pautasso

http://www.pautasso.info

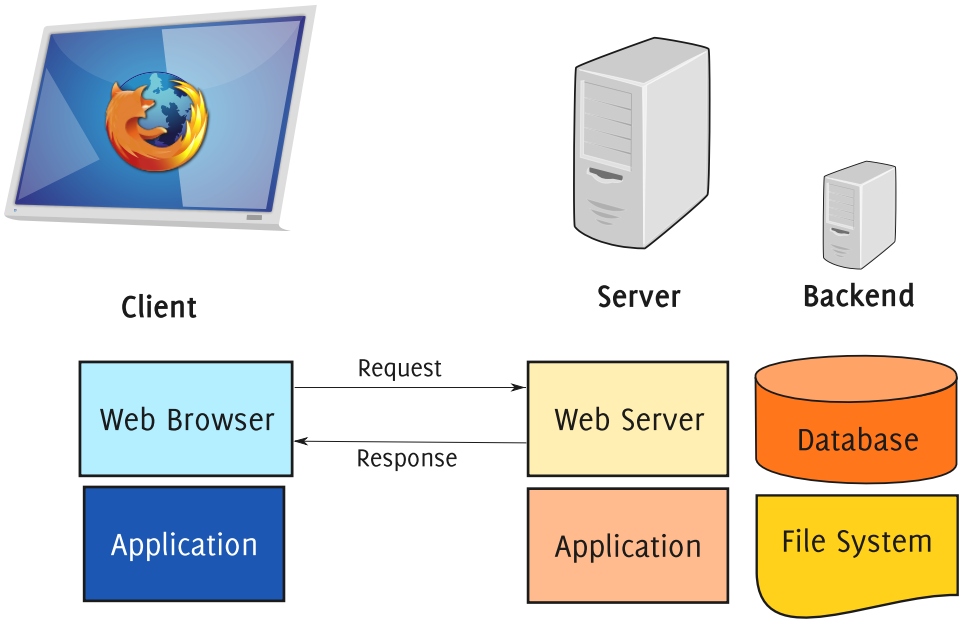
cesare.pautasso@usi.ch

@pautasso

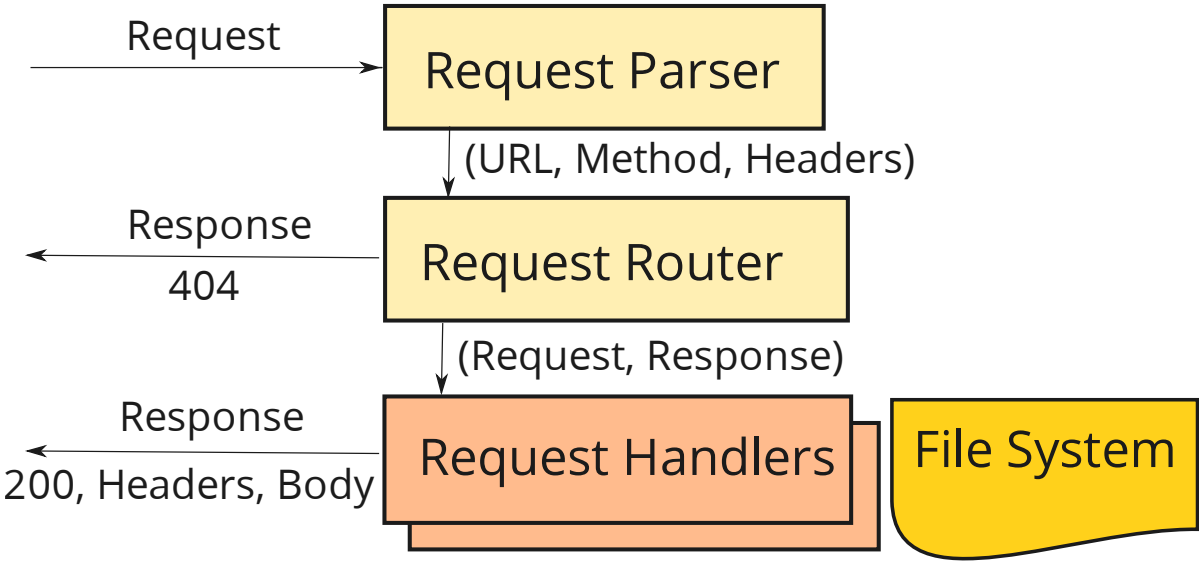
Where are we?



Web Architecture



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'
}
```

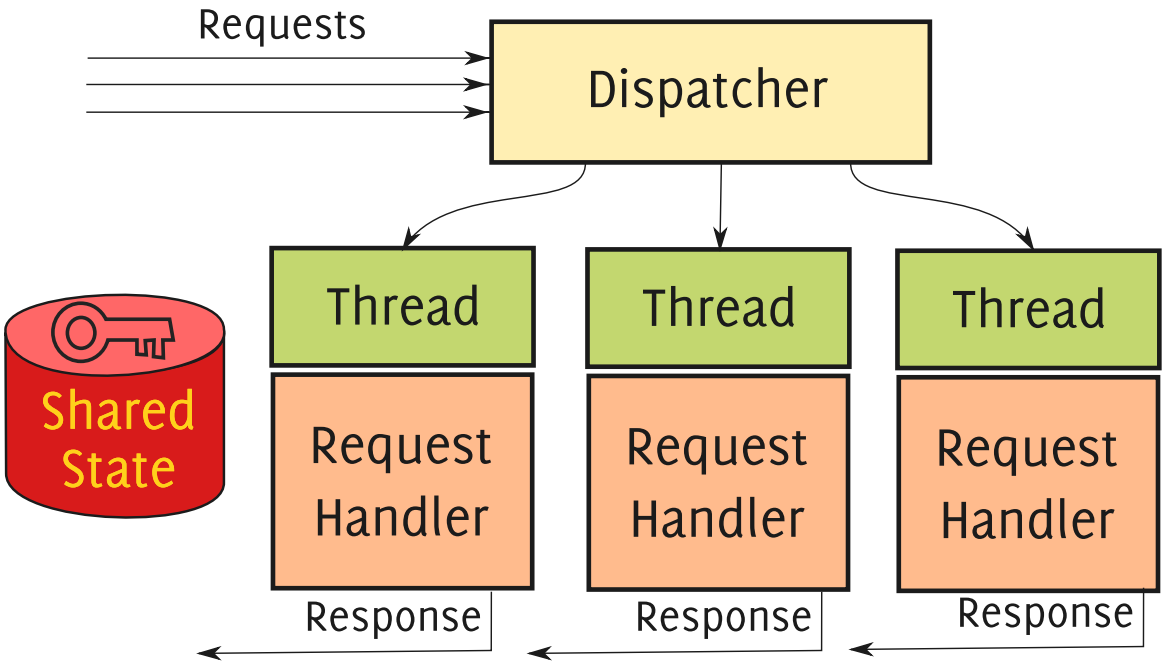

Working with responses

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

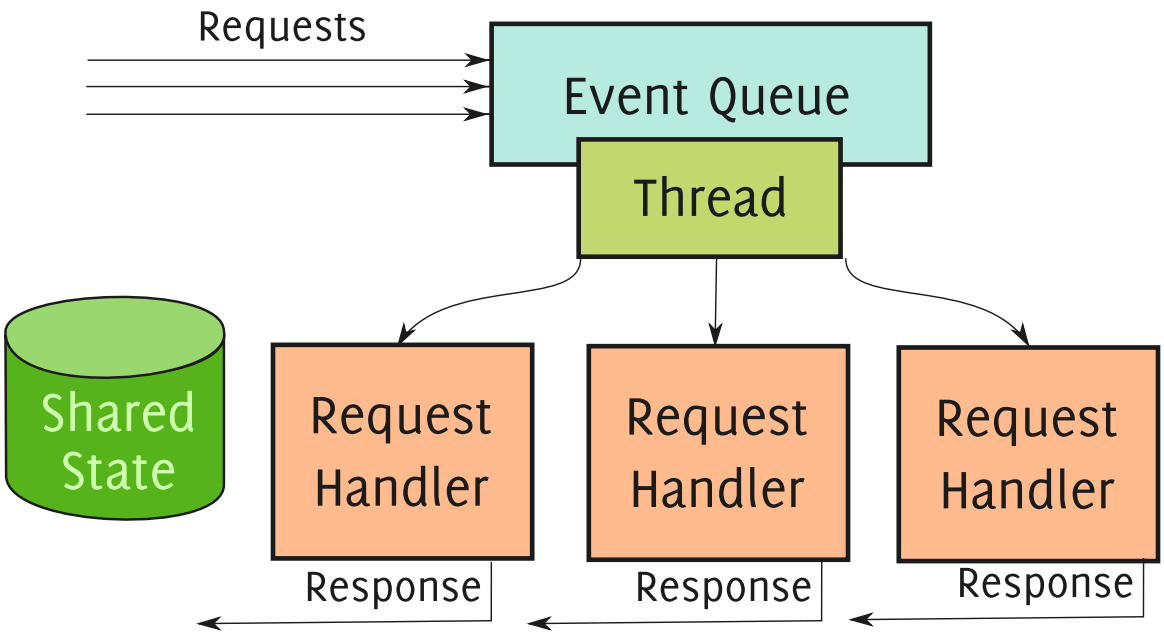
Explicit Headers

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

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

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);
  });
```

Tools

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

References

- 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