

The HTTP Protocol

RES, Lecture 6

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

	Monday	Tuesday	Friday
UDP Docker Music (challenge)	10.4	11.4	14.4
Docker Docker Music (challenge)	24.4	25.4	28.4
	1.5 TE	2.5 challenge	5.5 challenge
HTTP Code HTTP client	8.5	9.5	12.5
HTTP HTTP infrastructure	15.5	16.5 challenge	19.5 challenge
No lecture HTTP infrastructure	22.5	23.5	26.5
HTTP HTTP infrastructure	29.5	30.5	2.6
	5.6	6.6	9.6
Présentations HTTP infra	12.6 challenge	13.6	16.6

Welcome to “the Web”



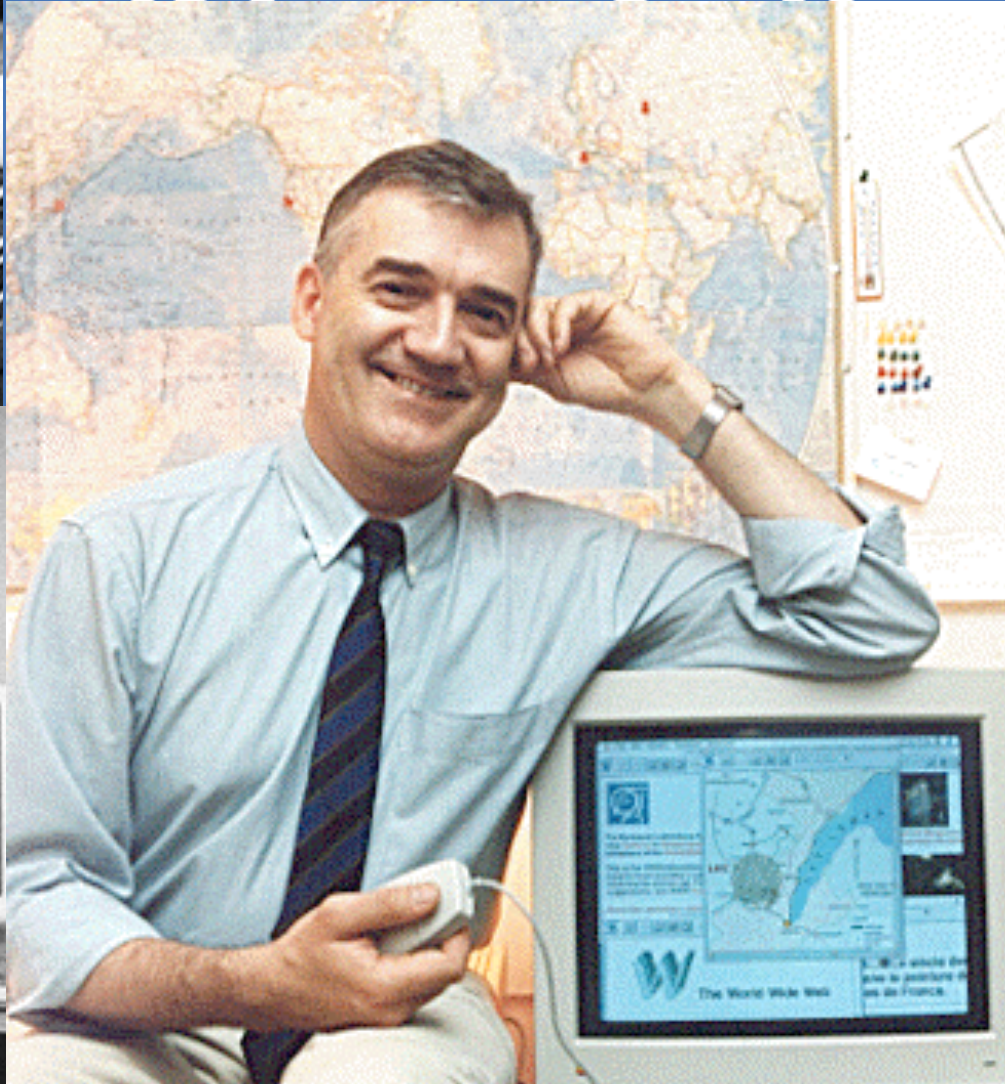
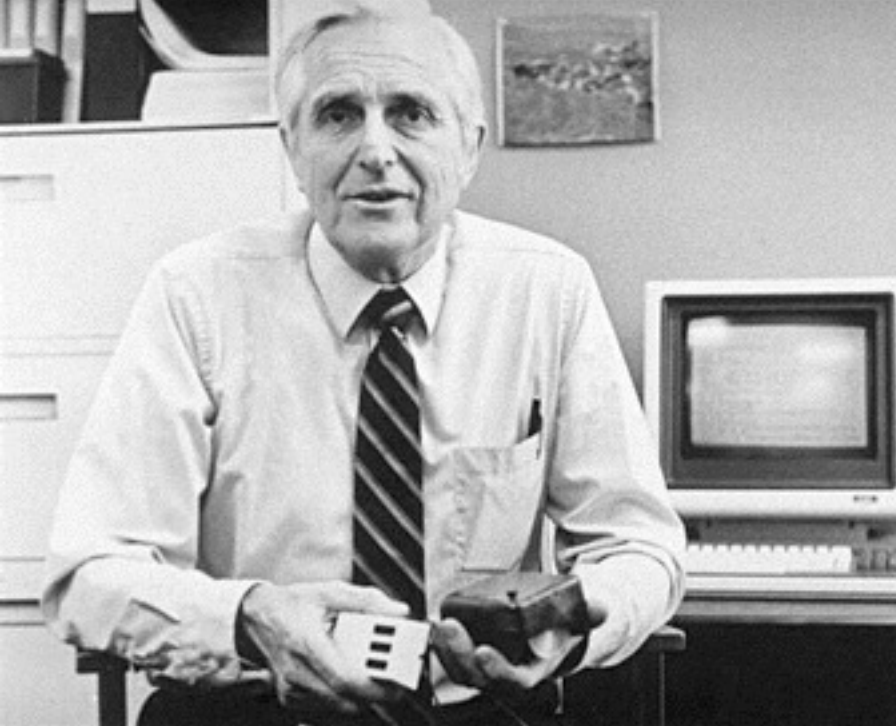
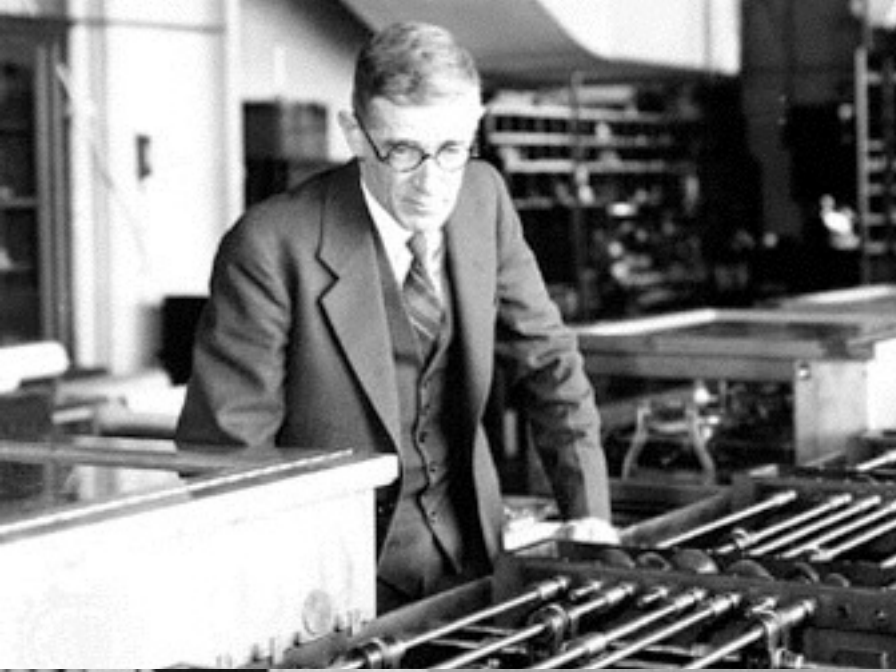
*It started as an (hypertext) **library**...*



*... which is now blending with the **physical** world*



*... which evolved to become an online application **platform**...*





http://en.wikipedia.org/wiki/Image:First_Web_Server.jpg



<http://www.w3.org/Consortium/technology>

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, **hypermedia information systems**.

It is a **generic, stateless**, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through **extension** of its request methods, error codes and headers [47].

A feature of HTTP is the typing and **negotiation of data representation**, allowing systems to be built independently of the data being transferred.

HTTP is one of the first two standards
of “the Web”



HTML

Markup Language to
create hypertext documents

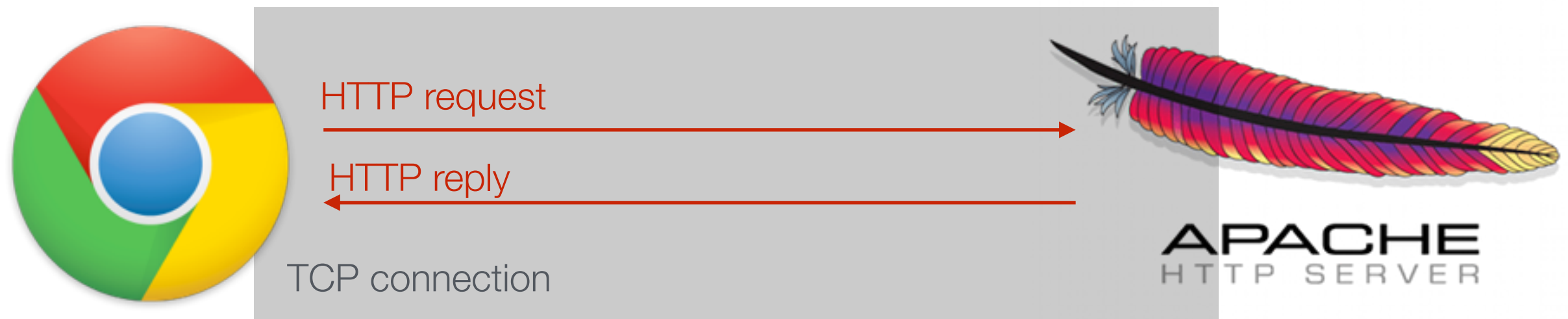


HTTP

Protocol to **transfer** hypertext
documents (and other content)

What is HTTP?

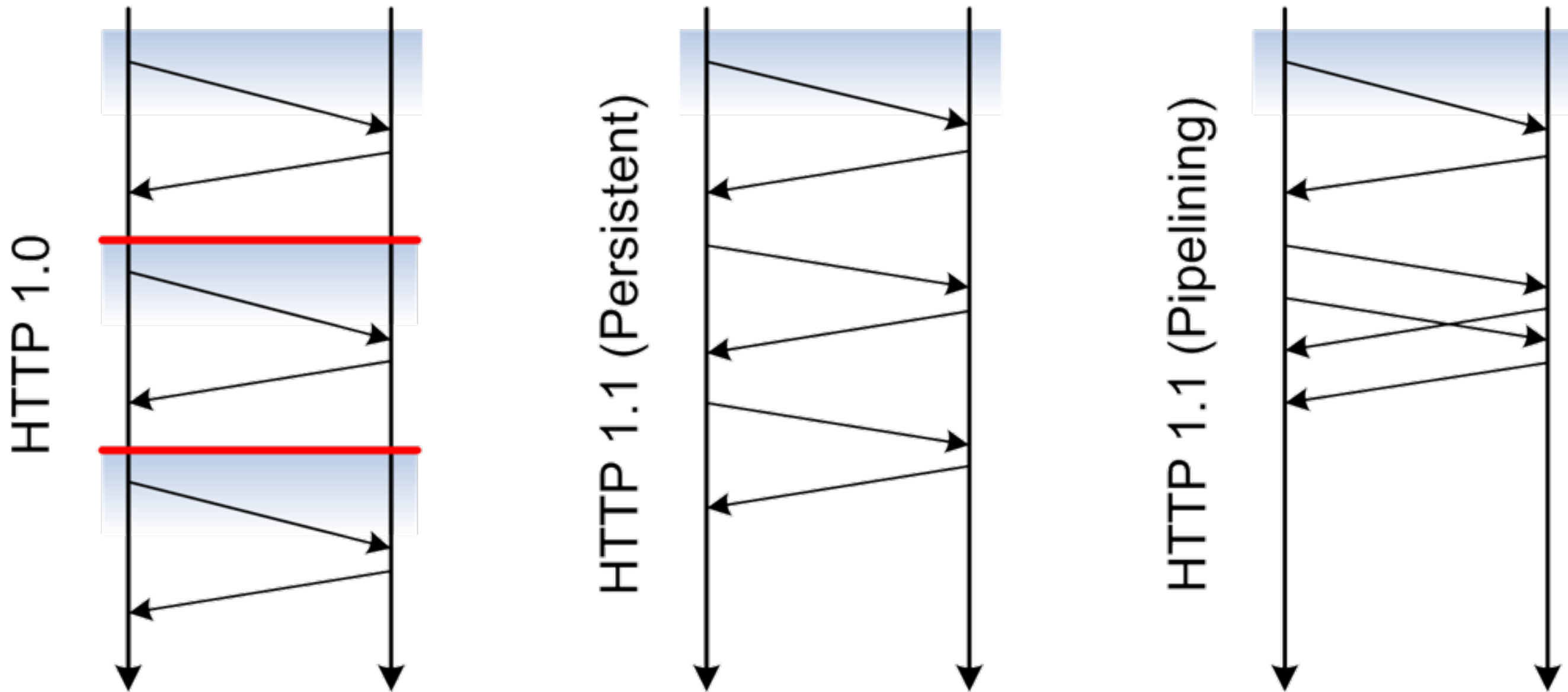
- **H**yper**T**ext **T**ransfer **P**rotocol.
- HTTP is an **application-level** protocol.
- HTTP is used to **transfer different types of payloads** (HTML, XML, JSON, PNG, MP4, WAV, etc.) between **clients** and **servers**
 - Sometimes, the client issues a request to GET (i.e. obtain, fetch, download) a payload from a server.
 - Sometimes, the client issues a request to POST (i.e. send, upload) a payload to a server.
- HTTP is built on top of **TCP** (the standard specifies that a server should accept requests on port 80).



The browser is an
HTTP client

The HTTP server accepts TCP connection
requests (by default on port 80)

HTTP & TCP Connections




[http://dret.net/lectures/web-fall07/foundations#\(20\)](http://dret.net/lectures/web-fall07/foundations#(20))


<http://www.apacheweek.com/features/http11>

Looking at a Conversation...





```
GET / HTTP/1.1 CRLF
Host: www.nodejs.org CRLF
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:28.0) Gecko/20100101 Firefox/28.0 CRLF
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 CRLF
Accept-Language: en-us,en;q=0.8,fr;q=0.5,fr-fr;q=0.3 CRLF
Accept-Encoding: gzip, deflate CRLF
Cookie: __utma=212211339.431073283.1392993818.1395308748.1395311696.27;
__utmz=212211339.1395311696.27.19.utmcsr=stackoverflow.com|utmccn=(referral)|utmcmd=referral
|utmcct=/questions/7776452/retrieving-a-list-of-network-interfaces-in-node-js- CRLF
Connection: keep-alive CRLF
CRLF
```



```
HTTP/1.1 200 OK CRLF
Server: nginx CRLF
Date: Sat, 05 Apr 2014 11:45:48 GMT CRLF
Content-Type: text/html CRLF
Content-Length: 6368 CRLF
Last-Modified: Tue, 18 Mar 2014 02:18:40 GMT CRLF
Connection: keep-alive CRLF
Accept-Ranges: bytes CRLF
CRLF
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link type="image/x-icon" rel="icon" href="favicon.ico">
    <link type="image/x-icon" rel="shortcut icon" href="favicon.ico">
    <link rel="stylesheet" href="pipe.css">
    ...
```

Request

GET / HTTP/1.1 CRLF

Host: www.nodejs.org CRLF

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:28.0) Gecko/20100101 Firefox/28.0 CRLF

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 CRLF

Accept-Language: en-us,en;q=0.8,fr;q=0.5,fr-fr;q=0.3 CRLF

Accept-Encoding: gzip, deflate CRLF

Cookie: __utma=212211339.431073283.1392993818.1395308748.1395311696.27;
__utmz=212211339.1395311696.27.19.utmcsr=stackoverflow.com|utmccn=(referral)|utmcmd=referral
|utmcct=/questions/7776452/retrieving-a-list-of-network-interfaces-in-node-js- CRLF

Connection: keep-alive CRLF

CRLF

n header lines

no content (because it is a GET)

1 request line

1 empty line

Response



HTTP/1.1 200 OK CRLF

Server: nginx CRLF

Date: Sat, 05 Apr 2014 11:45:48 GMT CRLF

Content-Type: text/html CRLF

Content-Length: 6368 CRLF

Last-Modified: Tue, 18 Mar 2014 02:18:40 GMT CRLF

Connection: keep-alive CRLF

Accept-Ranges: bytes CRLF

CRLF

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link type="image/x-icon" rel="icon" href="favicon.ico">
    <link type="image/x-icon" rel="shortcut icon" href="favicon.ico">
    <link rel="stylesheet" href="pipe.css">
```

n header lines

1 status line

1 empty line

body

Resources, Resource Representations & Content Negotiation



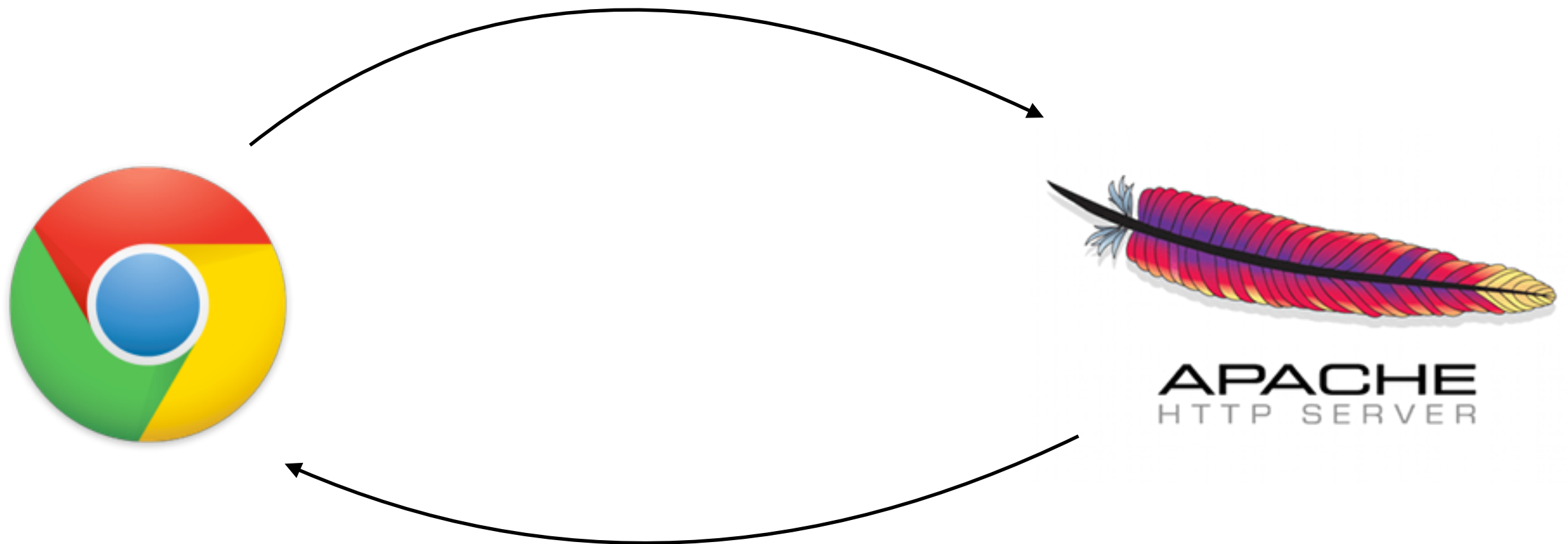
Resource vs Resource Representation

- **The notion of resource is very generic and can represent anything...**
 - An online document
 - A list of online documents
 - A stock quote updated in realtime
 - A vending machine
- **What is transferred is not the resource, but a representation of the resource**
 - HTML representation, JSON representation, PNG representation
 - french representation, english representation, japanese representation
 - etc.

Content Negotiation

- **When making a request, the client specifies its abilities and preferences**
 - media type: image, text, structured text?
 - media format: JSON, XML, etc.?
 - language: english, french, etc.
 - character encoding: UTF-8, ASCII, etc.
- **When answering the request, the server tries to do its best and indicates what it has been able to do**
- **Special headers are used to support this process**
 - **Request:** Accept, Accept-Charset, Accept-Language
 - **Response:** Content-Type, Content-Language

“This is what I am **able to** process, and these are my **preferences...**” (e.g. I am able to deal with plain text and XML, but I prefer JSON)”



“I am **able** to generate XML (not JSON), so you should be able to process this payload”

Protocol Syntax

HTTP Methods

GET

POST

PUT

DELETE

(PATCH)

URI

<http://www.heig-vd.ch>

Protocol Version

HTTP/1.0

HTTP/1.1

HTTP/2.0

HTTP Requests

Full-Request = Request-Line ; Section 5.1
 *(General-Header ; Section 4.3
 | Request-Header ; Section 5.2
 | Entity-Header) ; Section 7.1
 CRLF
 [Entity-Body] ; Section 7.2

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Request-Header = Authorization ; Section 10.2
 | From ; Section 10.8
 | If-Modified-Since ; Section 10.9
 | Referer ; Section 10.13
 | User-Agent ; Section 10.15

Entity-Header = Allow ; Section 10.1
 | Content-Encoding ; Section 10.3
 | Content-Length ; Section 10.4
 | Content-Type ; Section 10.5
 | Expires ; Section 10.7
 | Last-Modified ; Section 10.10
 | extension-header

HTTP Responses

Full-Response = Status-Line ; Section 6.1
 *(General-Header ; Section 4.3
 | Response-Header ; Section 6.2
 | Entity-Header) ; Section 7.1
 CRLF
 [Entity-Body] ; Section 7.2

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Response-Header = Location ; Section 10.11
 | Server ; Section 10.14
 | WWW-Authenticate ; Section 10.16

Entity-Header = Allow ; Section 10.1
 | Content-Encoding ; Section 10.3
 | Content-Length ; Section 10.4
 | Content-Type ; Section 10.5
 | Expires ; Section 10.7
 | Last-Modified ; Section 10.10
 | extension-header

Status Codes

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- o 1xx: Informational - Not used, but reserved for future use
- o 2xx: Success - The action was successfully received, understood, and accepted.
- o 3xx: Redirection - Further action must be taken in order to complete the request
- o 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- o 5xx: Server Error - The server failed to fulfill an apparently valid request

Status Codes

Status-Code	=	"200"	; OK
		"201"	; Created
		"202"	; Accepted
		"204"	; No Content
		"301"	; Moved Permanently
		"302"	; Moved Temporarily
		"304"	; Not Modified
		"400"	; Bad Request
		"401"	; Unauthorized
		"403"	; Forbidden
		"404"	; Not Found
		"500"	; Internal Server Error
		"501"	; Not Implemented
		"502"	; Bad Gateway
		"503"	; Service Unavailable
		extension-code	

extension-code = 3DIGIT

Reason-Phrase = *<TEXT, excluding CR, LF>

Parsing HTTP Messages

Process for Parsing HTTP Messages

- **Do NOT read characters, read bytes**
 - At the beginning, you want to parse line by line
 - When consuming the body, you may be dealing with binary content
- **HTTP 1.0**
 - On the client side, read until the connection is closed (end of stream reached).
 - On the server side, use the **Content-Length** header (for POST requests)
- **HTTP 1.1**
 - Static content: use the **Content-Length** header
 - Dynamic content: use the **chunked** transfer encoding

Recommendations

- **Implement your own `LineByLineInputStream`**
 - Remember the lecture about IOs & decorators?
 - You would like to have a `readLine()` method... but this one is available only in `Reader` classes
 - Implement your subclass of `FilterInputStream` and detect `\r\n` sequences
- **Add functionality incrementally, starting with a client**
 - Start with HTTP 1.0 (read until close of connection)
 - Deal with `Content-Length` header
 - Deal with `chunked` transfer encoding

HTTP is a Stateless
Request-Reply Transfer Protocol

Stateless Protocol... but Stateful Applications!



Managing State on Top of HTTP

- **Approach 1: move the state back-and-forth**
 - One way to do it is to use **hidden fields** in HTML forms
- **Approach 2 (better): maintain state on the backend, transfer session IDs**
 - One way to do it is to use parameters in the **query string** (security...)
 - One way is to use **cookies**



Passing State Back and Forth

C: Hello, I am new here. My name is Bob.

S: Welcome, let's have a chat [You told me that "My name is Bob"].

C: [My name is Bob]. What's the time?

S: Hi again Bob. It's 10:45 AM. [You told me that "My name is Bob". You asked me what is the time]

The image shows a user interface with a progress bar at the top and a table below it. The progress bar has four steps: 1 Account Data (light blue), 2 Profile (dark blue), 3 Confirm (light grey), and 4 Finish (light grey). Step 2 is currently active. Below the progress bar is a table with three columns: #, Name, and Description. The table contains two rows of data. At the bottom of the interface are two green buttons: 'previous' on the left and 'next' on the right.

#	Name	Description
1	Mark	Otto
2	Jacob	Thornton

Passing Session ID Back and Forth

C: Hello, I am new here. My name is Bob.

S: Welcome Bob, let's have a chat. Your session id is 42.

C: My session id is 42. What's the time?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: Hi again Bob. It's 10:45 AM.

C: My session id is 42. How do you do?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: I am fine, Bob, thank you.

C: My session id is 42. How do you do?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: I told you I am fine... are you stupid or what?

C: My session id is 42. If you take it like that, I am gone.
Forever.

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: -- putting 42 file into trash...

S: Bye Bob.

Passing Session ID in URL

GET /login HTTP/1.1
Host: intra.heig-vd.ch

HTTP/1.1 302 Found
Location: http://intra.heig-vd.ch/home?sessionId=83939

GET /home?sessionId=83939 HTTP/1.1
Host: intra.heig-vd.ch

GET /page2?sessionId=83939 HTTP/1.1
Host: intra.heig-vd.ch

GET /logout?sessionId=83939 HTTP/1.1
Host: intra.heig-vd.ch

Passing Session ID in Cookies

- The client **sends a request to the server for the first time**. It cannot send any cookie (it does not have any!).
- The server **understands that it is a new client**. It creates a session and generates a unique session ID.
- In the HTTP response, the server sends the unique session ID in a **Set-Cookie header** (“Here is a token... next time you come to see me, show me the token and I will recognize you!”).
- The **client keeps track of the cookie** either in memory or on disk (it has a cookie store, i.e. a list of cookies sent by servers).
- Before the client sends a second request to the server, it lookups the cookie store: “I am about to visit a server... did I receive any cookie from it?”. If yes, it sends the cookie in a **Cookie header**.

Passing Session ID in Cookies

- The server sends a cookie (token):

Set-Cookie: GAPSSESSID=4rm?????na; path=/; domain=gaps.heig-vd.ch

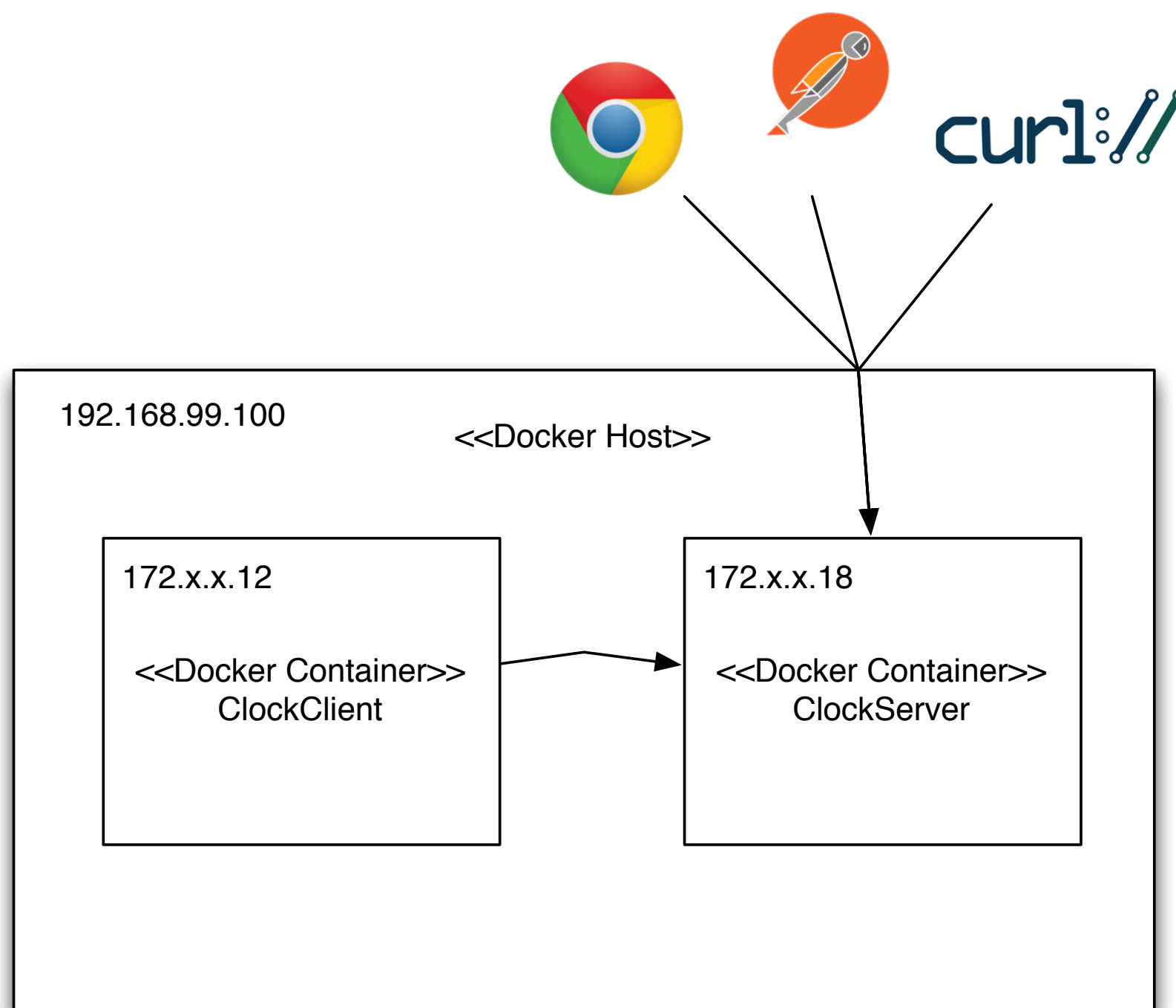
- The client sends it each time it returns to the server:

Cookie: GAPSSESSID=4rm?????na

Where does the server store the state?

- What is transferred in the cookie is **only a session ID**, not session data (and BTW not user credentials!).
- The **data associated to the session** typically includes the user identity (and roles), the previous actions performed by the user, the preferences, etc.
- **This data can be stored in various places.** Where you store it can have a big impact on performance, scalability and fault-tolerance. There is no universal “best solution” - the choice depends on many factors.
- Choice 1: **store everything in the DB.** One advantage is that if a web server crashes, it does not bring down the state. One disadvantage is that performance may
- Choice 2: **keep in the RAM of the web server.** One advantage is the performance. One disadvantage is that you have to use sticky sessions. Another disadvantage is that RAM consumption could be high.
- Choice 3: **use a distributed caching layer** (e.g. redis, memcached). One advantage is fault tolerance and scalability. One disadvantage is higher complexity and cost.

Training



Useful tools for your
detective arsenal



heig-vd

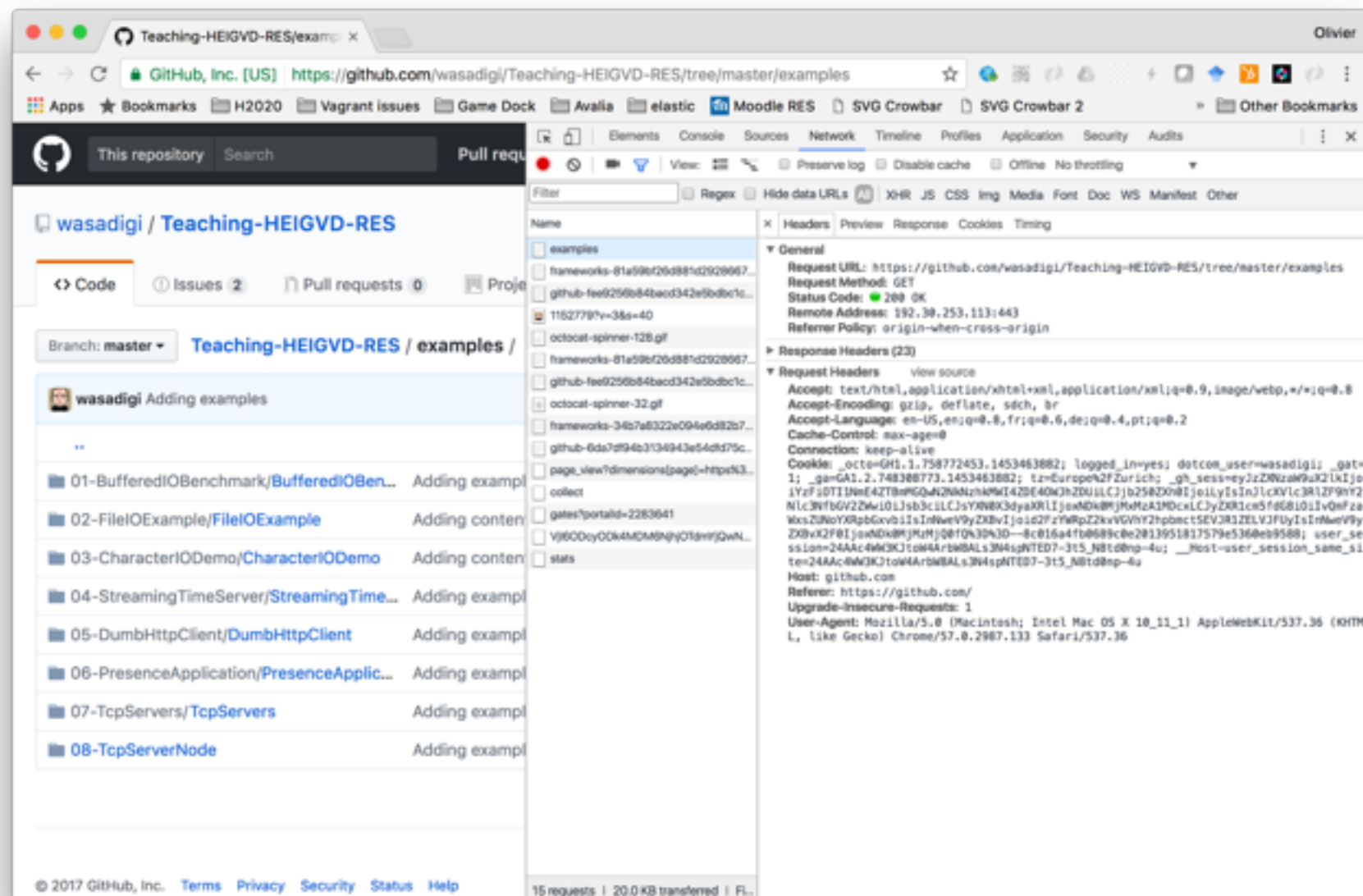
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What can I do with it?

inspect request and response headers and body

Where do I find help?

<https://developer.chrome.com/devtools>





heig-vd

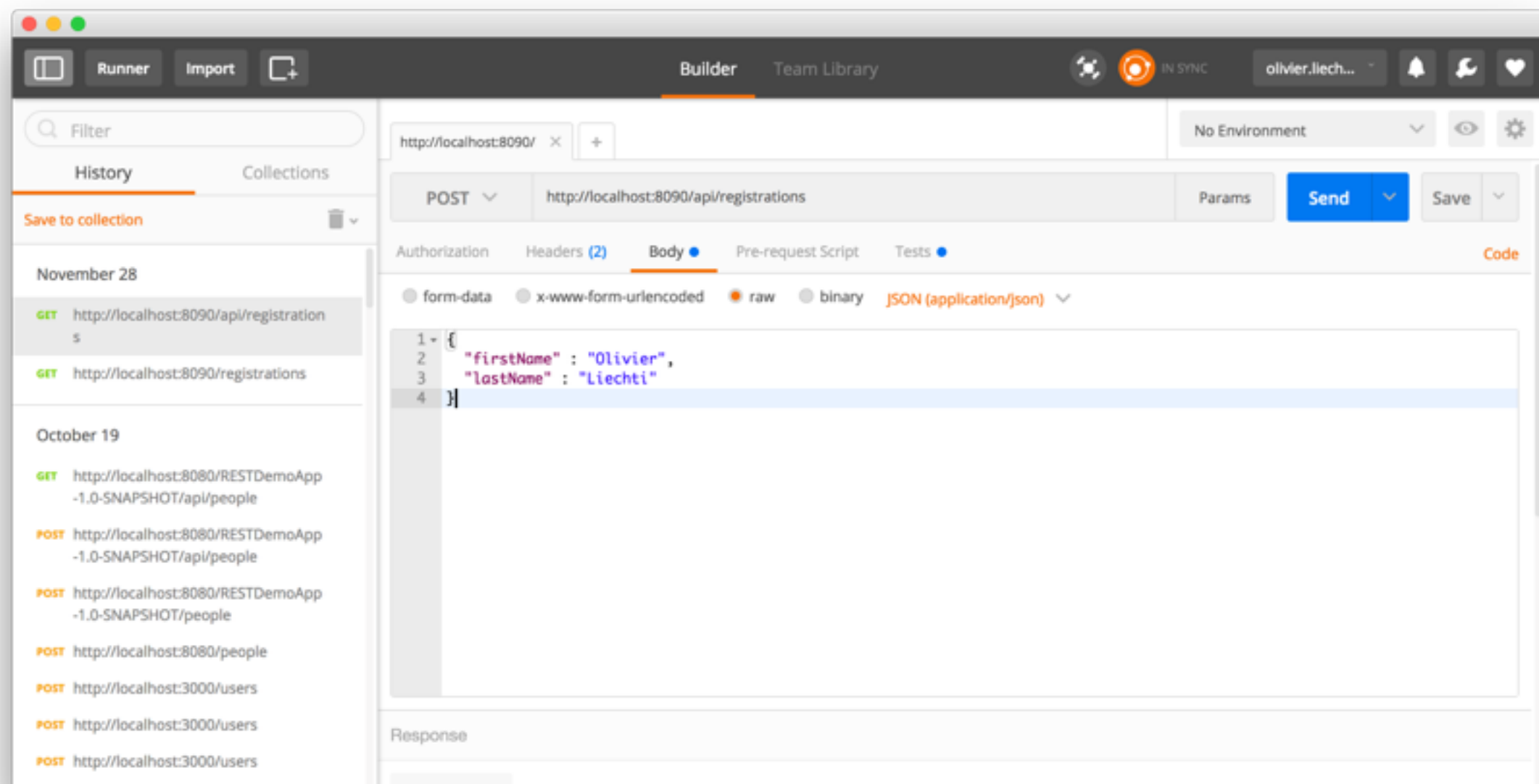
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What can I do with it?

Send HTTP requests (also POST, also the body...)

Where do I find help?

<https://www.getpostman.com/>





What can I do with it?	send HTTP requests from terminal, inspect headers
Where do I find help?	https://curl.haxx.se/
How do I install it?	Package manager. You can use it in Docker.



<p>Low level</p> <p>You know the drill, just implement the protocol syntax!</p>	<pre>Socket s = new Socket("www.heig-vd.ch", 80); ...</pre>
<p>Intermediate level</p> <p>(see Java tutorial)</p>	<pre>URL url = new URL("www.heig-vd.ch"); URLConnection con = url.openConnection();</pre>
<p>High level</p> <p>(see http://hc.apache.org/ and don't forget to add the dependency in your pom.xml)</p>	<pre>Request.Get("http://somehost/") .connectTimeout(1000) .socketTimeout(1000) .execute().returnContent().asString();</pre>



Low level	<pre>var client = new net.Socket(); client.connect(80, 'www.heig-vd.ch', function() {...}); client.on('data', function(data) {...});</pre>
Intermediate level (see Java tutorial)	<pre>var http = require('http'); var options = {...}; var request = http.request(options, function(response) {...}); request.end();</pre>
High level (many npm modules: request, request-promise, superagent)	<pre>request .post('/api/pet') .send({ name: 'Manny', species: 'cat' }) .set('Accept', 'application/json') .end(function(err, res){ ... });</pre>