The HTTP Protocol

RES, Lecture 4

Olivier Liechti

heig-vd

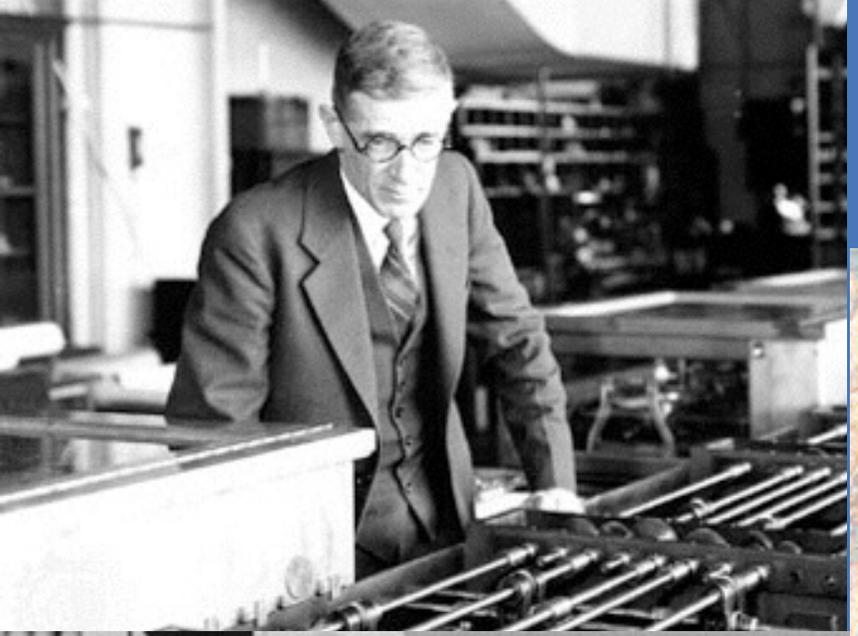
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

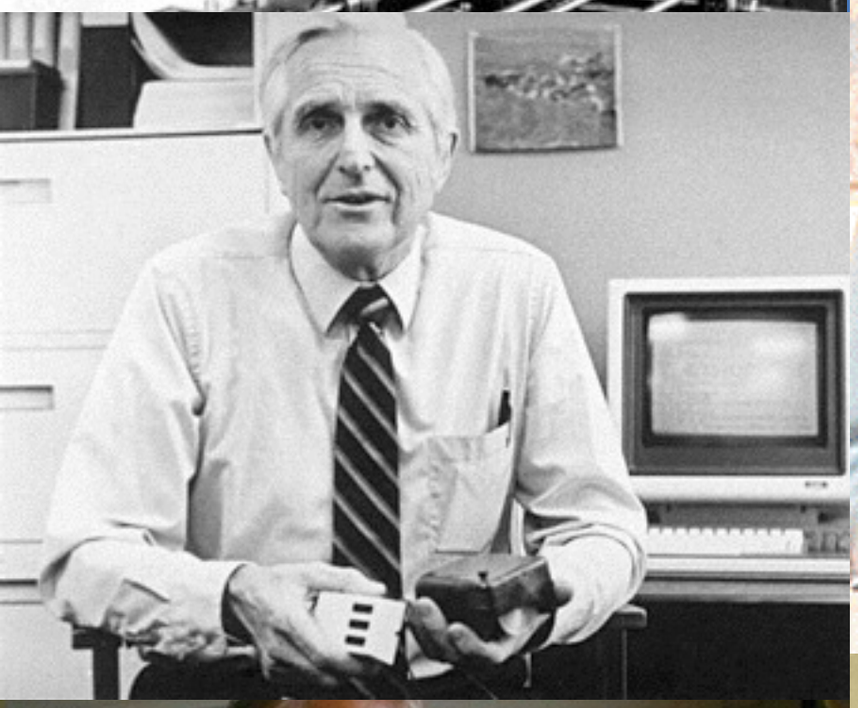
Welcome to “the Web”

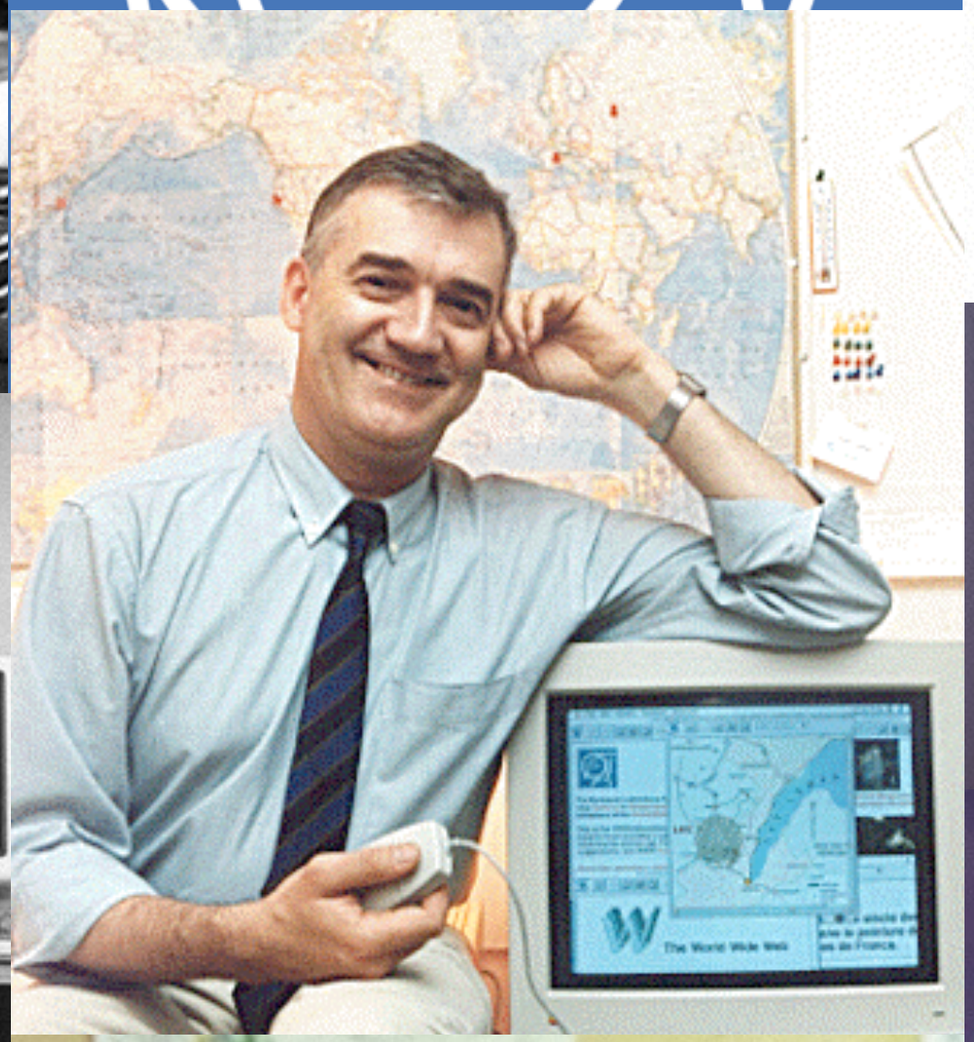
It started as an (hypertext) ***library****...*

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

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






The World Wide Web


amazon


facebook

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

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

What is HTTP?

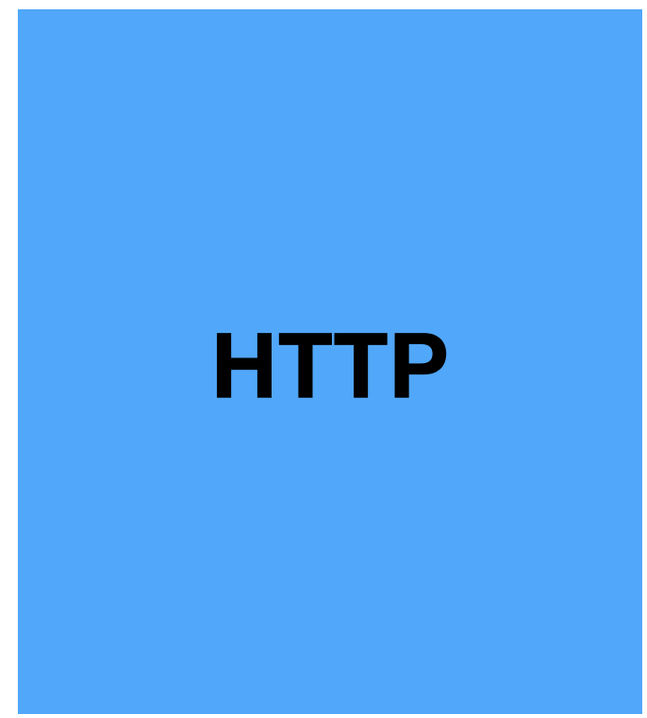
The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, **hypermedia information systems**.

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”

Markup Language to **create** hypertext documents

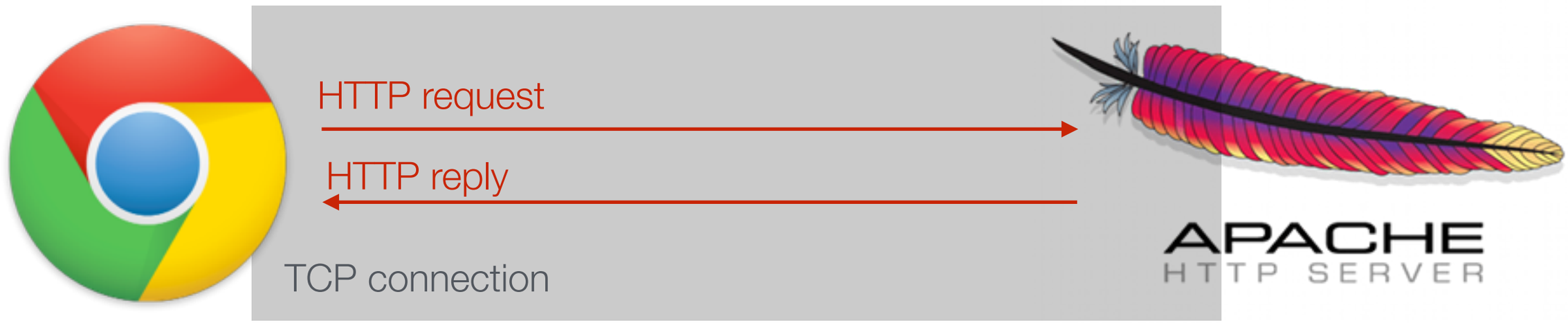


Protocol to **transfer** hypertext documents (and other content)

What is HTTP?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **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)

Looking at a Conversation...

chrome

WIRESHARK

Firebug

```
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

Docker Setup

Environment (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Let's create a **Docker image** to setup an HTTP server:
  - In this setup, we will **not use apache httpd** server (nor nginx).
  - Instead, we will **use the express.js** web framework (which runs on top of Node.js).
  - When we start the server, it will **accept HTTP requests on port 3000** (default port used by the express.js framework).

http://expressjs.com/

Environment (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
#
# This image is based on another image
#
FROM dockerfile/nodejs:latest

#
# For information: who maintains this Dockerfile?
#
MAINTAINER Olivier Liechti

#
# When we create the image, we copy files from the host into
# the image file system. This is NOT a shared folder!
#
COPY file_system /opt/res/

#
# With RUN, we can execute commands when we create the image. Here,
# we install the PM2 process manager
#
RUN npm install -g pm2@0.12.9

#
# Run npm install in the app folder, to install dependencies
#
WORKDIR /opt/res/
RUN npm install

#
# With CMD, we start the PM2 process manager. The process manager is responsible
# for starting the node.js script (the express.js app)
#
CMD pm2 start -x /opt/res/bin/www --no-daemon
```

1. Create a Docker image

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Remember that we have prepared an environment in
`git@github.com:SoftEng-HEIGVD/Teaching-HEIGVD-RES-2015-Vagrant.git`

```
host $ vagrant ssh

vagrant $ cd /vagrant/docker/image_expressjs/
```

This folder contains a **Dockerfile** and a **file_system** directory that will be copied into the image file system.

```
vagrant $ docker build -t heigvd/express-demo .
vagrant $ docker images
```

Let's give a **name** to our image.

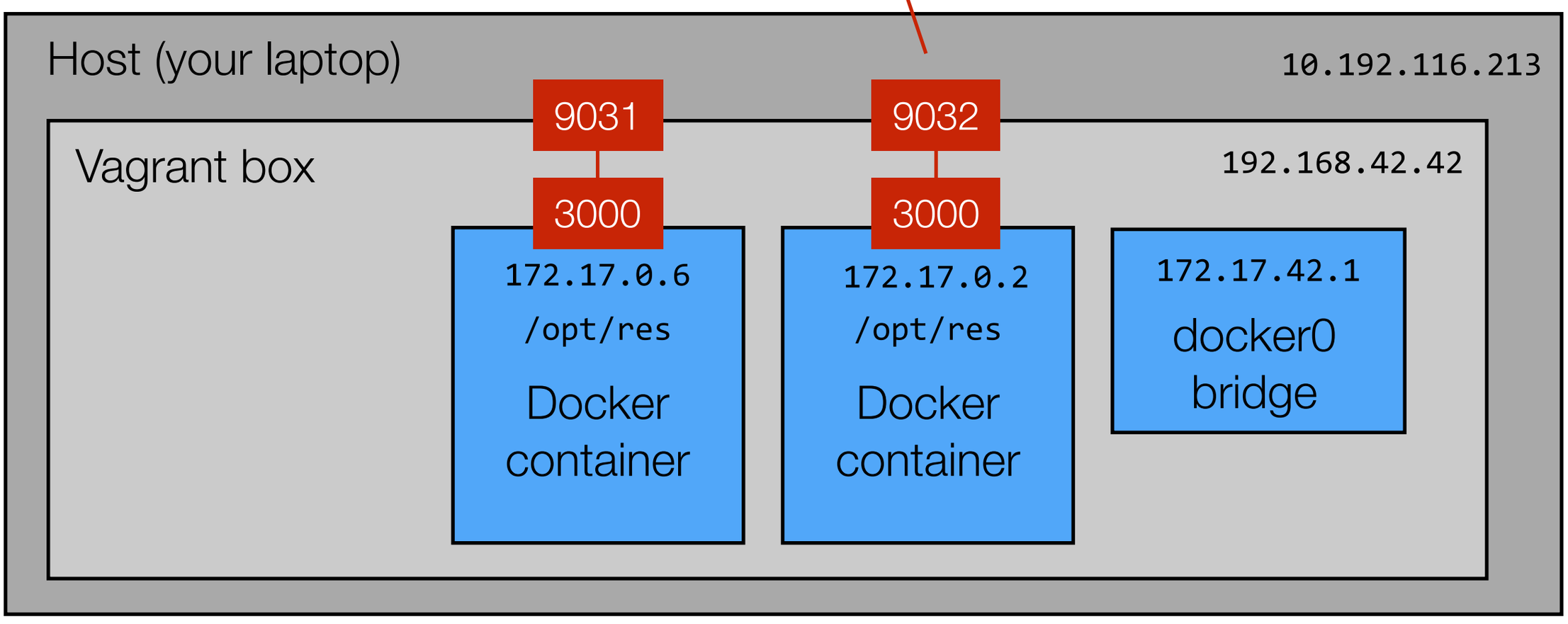
2. Run 2 Docker containers

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

We want to run containers based on this image

```
vagrant $ docker run -d -p 9031:3000 heigvd/express-demo
vagrant $ docker run -d -p 9032:3000 heigvd/express-demo
vagrant $ docker ps -a
```

map the container TCP port to a box TCP port



3. Find the container IP address

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
vagrant $ docker ps
```

```
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS              PORTS                    NAMES
37b67054c0ad        heigvd/express-demo:latest  "/bin/sh -c 'pm2 sta   11 hours ago        Up 11 hours         0.0.0.0:9030->3000/tcp   goofy_perlman
```

We did not give a name to our container when starting it (unlike last week), so Docker chose a random name for it.

docker inspect gives a lot of information about a container; we are only interested by the IP address, so we **grep** that information.

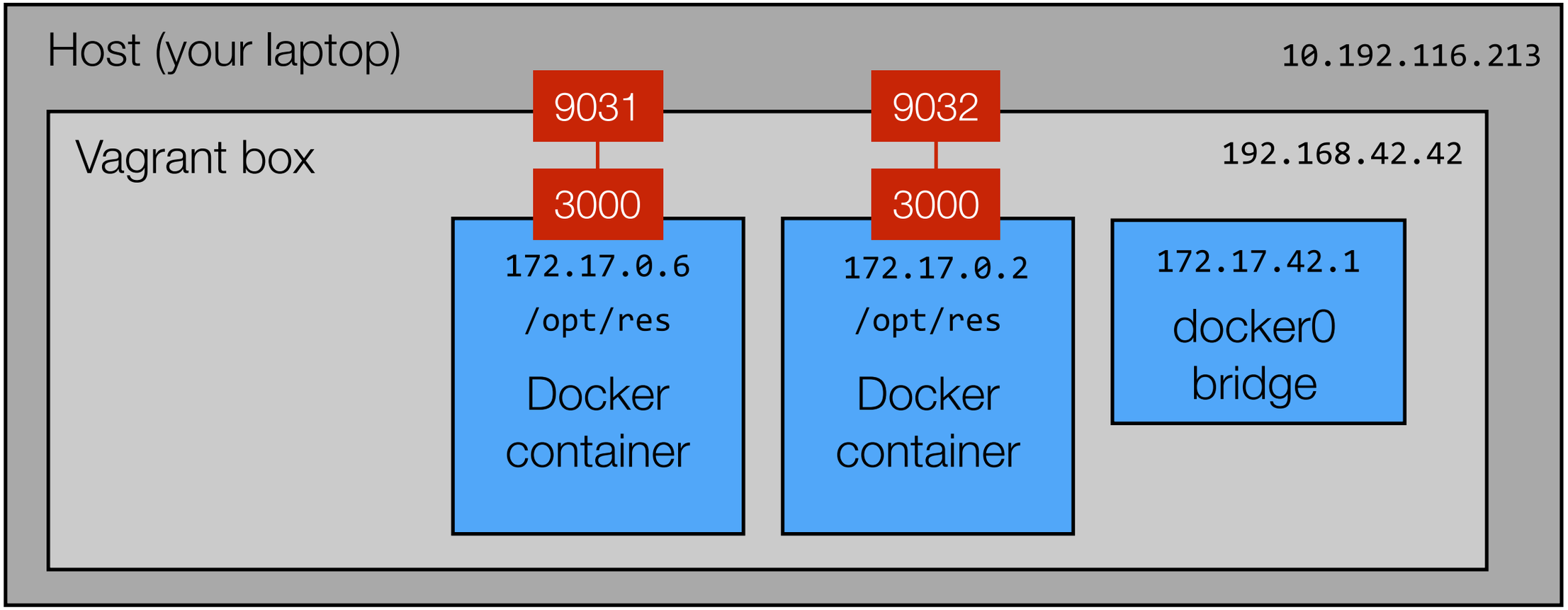
```
vagrant $ docker inspect goofy_perlman | grep -i ipaddress
        "IPAddress": "172.17.0.10",
```

4. Connect to the HTTP server

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

When connecting to the HTTP server, the IP address and TCP port depends on where you are (on your host, in your box, etc.)

```
   host $ telnet 192.168.42.42 9031
vagrant $ telnet localhost 9031
vagrant $ telnet 172.17.0.6 3000
```



5. Issue an HTTP request

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
vagrant $ telnet localhost 9030
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 355
ETag: W/"163-390ef0c8"
Date: Wed, 15 Apr 2015 03:57:17 GMT
Connection: close

<!DOCTYPE html><html><head><title>RES Web Server, powered by Express</
title><link rel="stylesheet" href="/stylesheets/style.css"></
head><body><h1>RES Web Server, powered by Express</h1><p>
Welcome to <b>RES Web Server, powered by Express</b>.</p><p>This is a
simple page to demonstrate how express.js can be used to create a web
server.</p></body></html>Connection closed by foreign host.
```

Resources, Resource Representations & Content Negotiation

Resource vs Resource Representation

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

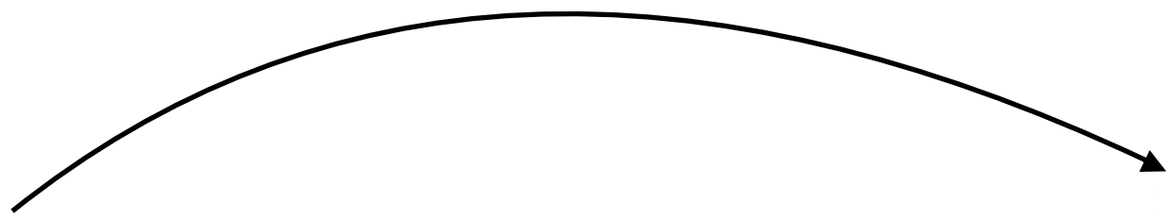
- **The notion of resource is very generic and can represent anything...**
  - An online document
  - A list of online documents
  - A stock quote updated in realtime
  - A vending machine
- **What is transferred is not the resource, but a representation of the resource**
  - HTML representation, JSON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

Content Negotiation

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **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

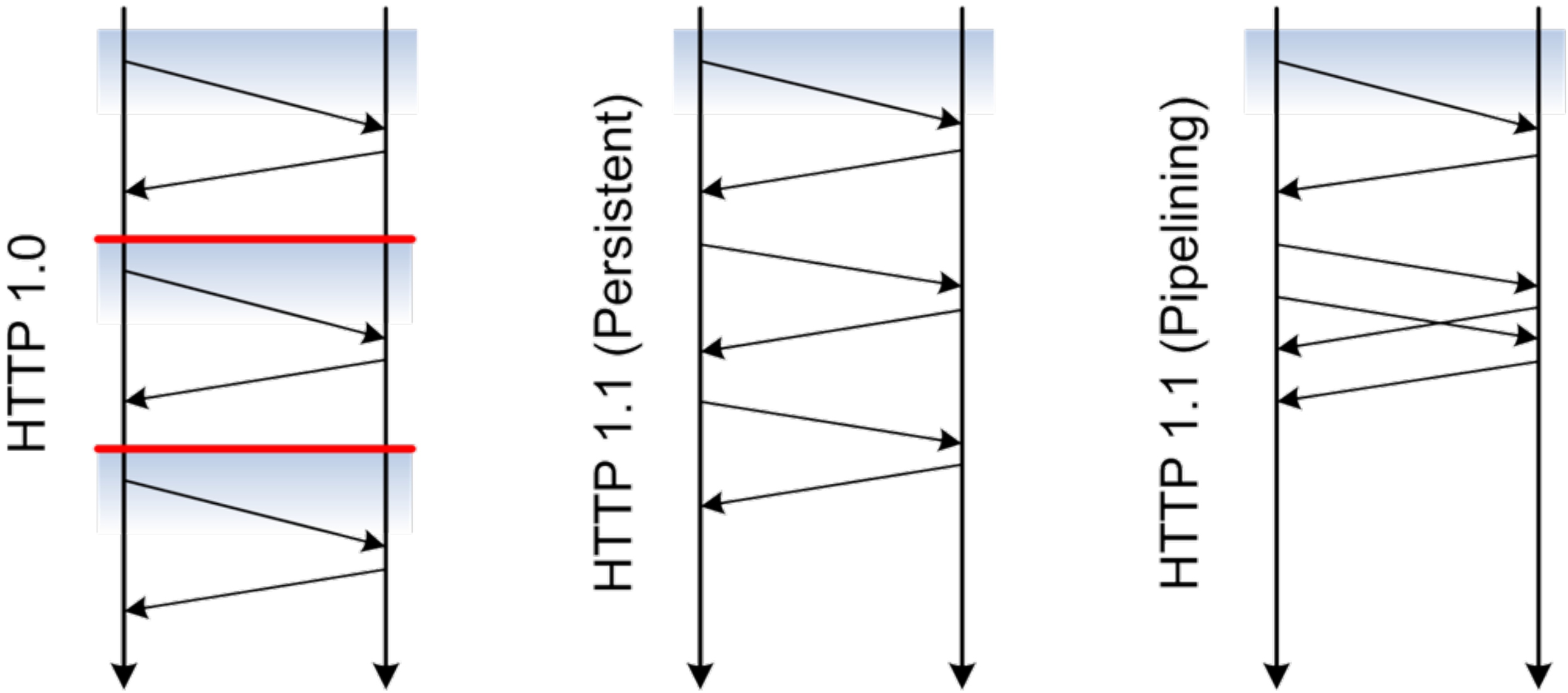
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Do NOT read characters, read <u>bytes</u>**
  - 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 & TCP Connections



http://dret.net/lectures/web-fall07/foundations#(20)
http://www.apacheweek.com/features/http11

HTTP is a Stateless
Request-Reply Transfer Protocol

Stateless Protocol... but Stateful Applications!

Managing State on Top of HTTP

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Approach 1: moving the state back-and-forth**
  - One way to do it is to used **hidden fields** in HTML forms
- **Approach 2: maintaing 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

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

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]

Passing Session ID Back and Forth

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
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.
```