# The World Wide Web

## COMPUTER NETWORKS A.A. 24/25

Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

## Sect. 1 WWW

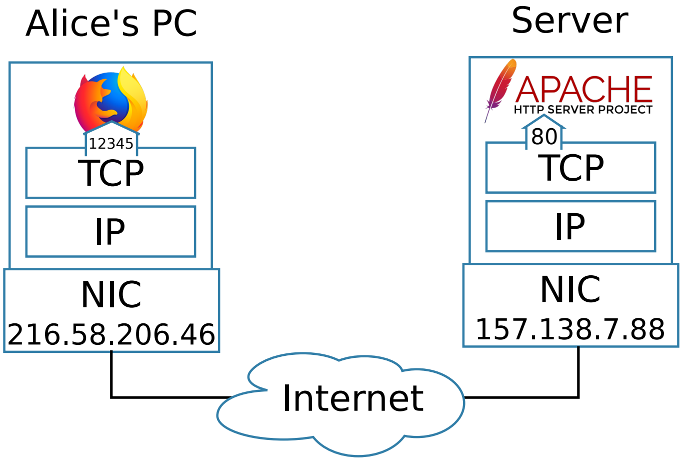
## The World Wide Web

What we know today as the WWW is a mix of various components

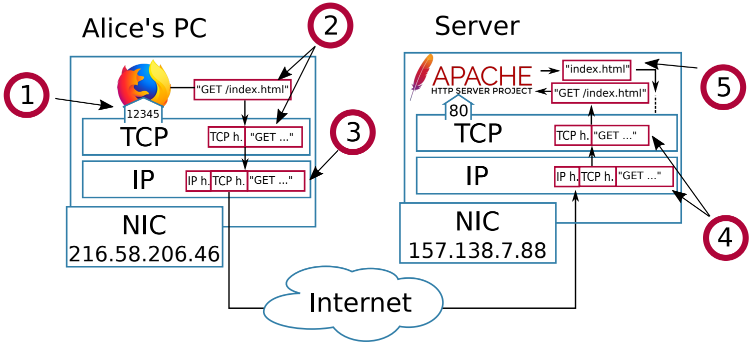
- A schema for creating references to locations
- A language used to build pages
- An application layer protocol to retrieve the pages

## Server Organization

- Before we go into the details of the protocols, let's see how the application functionally work

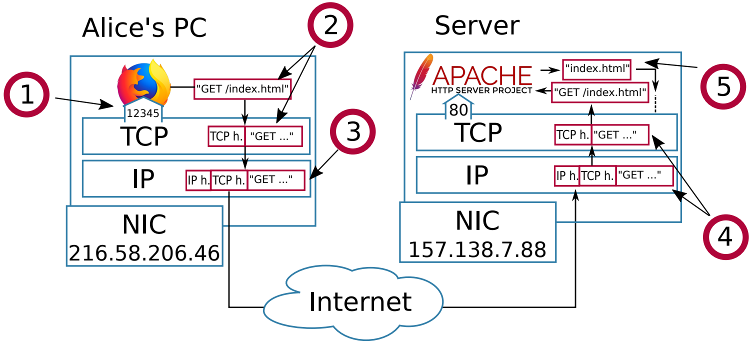


## Communication Flow



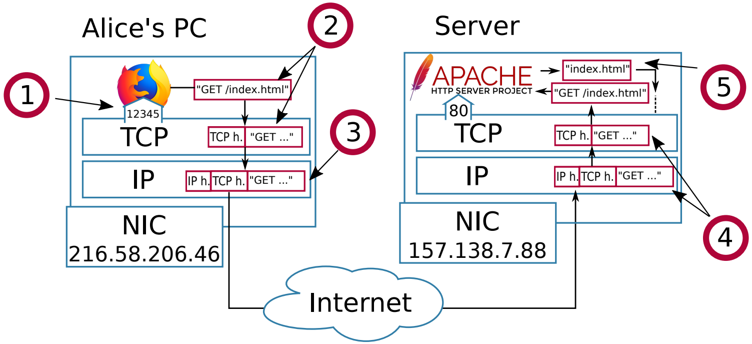
1. The browser is assignes a port from the TCP layer, the web server gets port 80

## Communication Flow



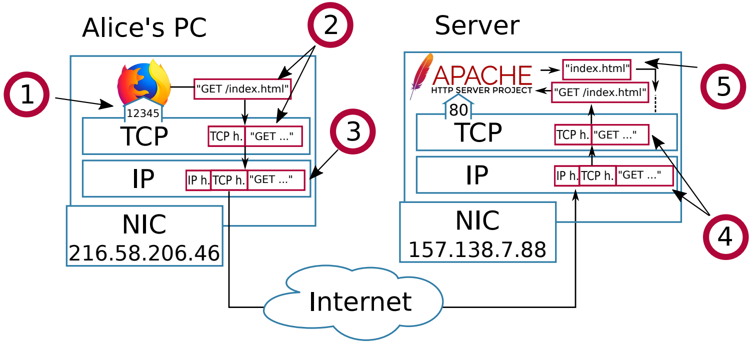
2. The browser issues an HTTP command (more on this later), it gest incapsulated in a TCP segment

## Communication Flow



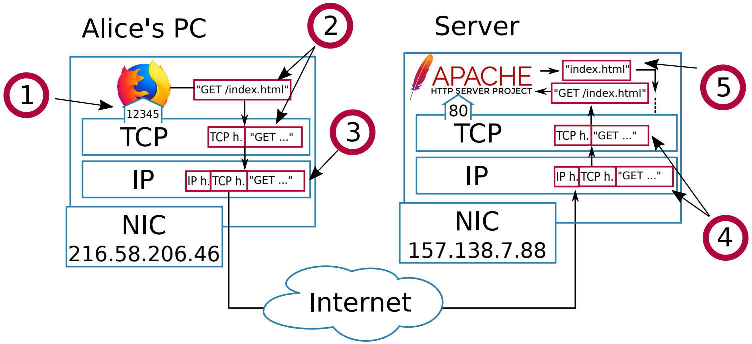
3. This gets incapsulated in an IP segment

# Communication Flow



4. The IP and TCP layer decapsulate it

## Communication Flow



5. The command is received by the web server, that elaborates and answers

## WWW

↳1.1 URI

## Universal Resource Identifiers: URI

- An URI is a character string that unambiguously identifies a resource on the world wide web
- An URL is an URI that provides you with the location to retrieve a resource, contrarily to an URN that is an URI that simply identifies its name, and not its location
- A (simplified) format of an URI/URL is the following:

```
URI = scheme ":" "//" authority path [ "?" query ] [ "#" fragment ]
```

- let's see it piece by piece

## URI: scheme

- A scheme can be seen as a selector, indicating the meaning of the fields after it.
- In practice, the scheme often identifies the application-layer protocol that must be used by the client to retrieve the document, but it is not always the case
- Some valid schemes are: `http`, `https`, `ftp`, `mailto`

## URI: authority

- This includes the domain name or the IP address of the server where the document can be retrieved using the protocol specified via the scheme.
- This name can be preceded by some information about the user (e.g. a user name and a password, even if passwords in URIs are deprecated) who is requesting the information.
- This is included before the @ character
- The host name can be followed by the colon character (:) and a port number.
- The port should be included only if the scheme does not implicitly sets one.

## URI: path

- The third part of the URI is the path to the document.
- This path is structured as filenames on a Unix host (but it does not imply that the files are indeed stored this way on the server).
- If the path is not specified, the server will return a default document.

## URI: query and fragment

- The last two optional parts of the URI are used to provide a query parameter, that will be parsed by the code on the receiving server (`?parameter=value`)
- Several parameters are separated using the ampersand `&`
- and to indicate a specific fragment in the document `#SubsectionTitle` to point to

## URI examples

`http://tools.ietf.org/html/rfc3986.html`

| | |
|---|---|
| schema: | `http` |
| authority: | `tools.ietf.org` |
| path: | `/html/rfc3986.html` |

## URI examples

`mailto:infobot@example.com?subject=current-issue`

| | |
|---|---|
| schema: | `mailto` |
| authority: | `infobot` (user) in the `example.com` domain |
| path: | none |
| query: | `subject=current-issue` |

This will be opened by an a MUA that will start editing an e-maiil for `infobot@example.com` with subject *current issue*.

## URI examples

`https://en.wikipedia.org/wiki/Italy#Name`

| | |
|---|---|
| schema: | `http` |
| authority: | `en.wikipedia.org` |
| path: | `/wiki/Italy` |
| query: | none |
| fragment: | `#Name`. The page will be displayed starting from this section |

## URI examples

`ftp://cnn.example.com&story=breaking_news@10.0.0.1/top_story.htm`

And what about this?

## Sect. 2 HTML

## Markup Languages: HTML

- Now that we know how to refer to objects, we need a way to add links to documents
- To do this we enrich the text inside documents with attributes, named tags
- The way this is done is with a Markup language, The Hypertext Markup Language (HTML)

```
<tag>Some text to be displayed</tag>

<tag attribute1="value1" attribute2="value2">some text to be displayed</tag>
```

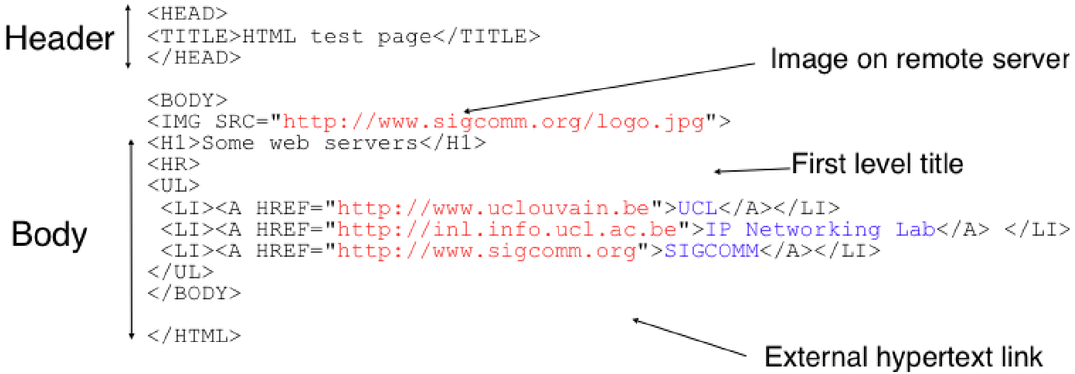
## HTML

- HTML was first standardized by IETF, then its following versions were taken over by the W3C consortium
- Right now the latest stable version is HTML 5.0

## An HTML page

- An HTML document is made of a header and a body, included in the respective tags
- Inside the page the text can be formatted using tags, including the presence of external objects
- These external objects will be referenced by their URL and will be loaded separately by the Browser, when it renders the page

# Example Page



```
<HTML>

<HEAD>
<TITLE>HTML test page</TITLE>
</HEAD>

<BODY>
<IMG SRC="http://www.sigcomm.org/logo.jpg">
<H1>Some web servers</H1>
<HR>
<UL>
 <LI><A HREF="http://www.uclouvain.be">UCL</A></LI>
 <LI><A HREF="http://inl.info.ucl.ac.be">IP Networking Lab</A> </LI>
 <LI><A HREF="http://www.sigcomm.org">SIGCOMM</A></LI>
</UL>
</BODY>

</HTML>
```

## HTML tags

In the previous example some common tags are shown, like:

`IMG`: This provides the URL for a remote image. The browser will load the image and compose it with the rest of the page. Images can be stored anywhere on the Internet

`HR`: an horizontal rule

`UL,LI`: Unordered list and a list item

`A`: a link, where `HREF` is the URI

## HTML

### ↳2.1 Dynamic Pages

## Dynamic Pages in HTML

- An HTML page is just a file, so if you want to have a page with dynamic contents, there are two ways.
- The first is to add server-side code, that makes the page change when you visit it, based on what you have done before, or based on your query.
- This means that the URL does not point to an HTML file, but to a file in some language that gets executed on the server, and will return an HTML file to the browser.
- This is referred to as Common Gateway Interface: CGI

## PHP Example

PHP is a widely used server-side programming language, that is embedded inside the `HTML` page itself, or it is directly referenced by the URL, when you see it points to a `.php` file.

```
<html>
 <body>
  <h1>Example PHP page</h1>
  <?php echo "This is the output of PHP code"; ?>
 </body>
</html>
```

This code will produce something like:

# Example PHP page

This is the output of PHP code

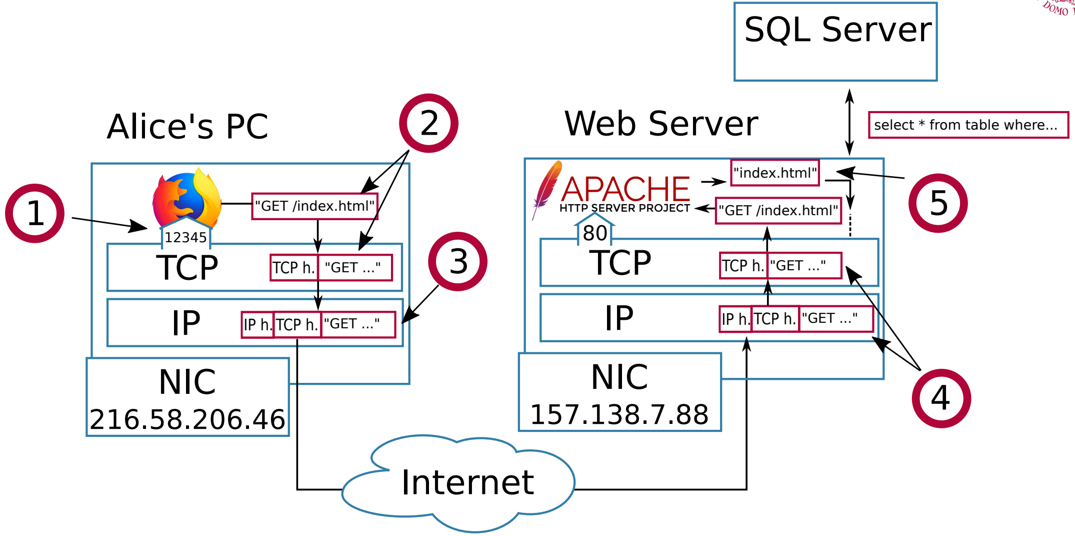
## PHP query variables

- Server-side code can use the variables you pass as queries in the URL
- For instance using the following code, if the user browses: `http://example.com/?name=Hannes`

```
<html>
 <body>
 <?php
    echo "Hello " . htmlspecialchars($_GET["name"]) . "!";
 ?> </body>
</html>
```

This code will produce something like:

Hello Hannes!


SQL Server
select * from table where...
Alice's PC
Web Server
1
2
3
4
5
"GET /index.html"
12345
TCP
TCP h.
"GET ..."
IP
IP h.
NIC
216.58.206.46
APACHE
HTTP SERVER PROJECT
"index.html"
80
NIC
157.138.7.88
Internet

## Client Side Code

- With server-side programs the user should not access the source code.
- The code is ran by the server, the `HTML` is generated and delivered to the browser
- The other way of making dynamic pages is with client-side programming
- This means that the code is embedded in the `HTML` page, sent to the browser and the browser runs it.
- The most popular client-side language is Javascript

## Javascript Example

```html
<html>
<body>
  <p>Hello World!</p>
  <script>
    alert("Hello, world! in JS");
  </script>
</body>
</html>
```

Try it yourself. Save the code in an `HTML` file and open it with your browser.

## Client Vs Server side code

- The two models are used for similar but different things
- The server-side code is used for instance, to query a database and select the right information. The user should not have access to the database itself, only to the information it contains
- The client-side is used to make dynamic actions, that in some cases may even not require the use of the connection
- You can use both things togeter

## Example

- An e-commerce has a database of thousands of items you can purchase
- When you search for a specific keyword, the back-end will filter all the objects that are relevant for you and send them to the browser
- The browser then can order them based on the user desire (by price, age. . . ). This will be done client-side by JS in the user browser, without the need to query the website again.

## HTML

↳ 2.2 Style Sheets

## Formatting the Page content: CSS

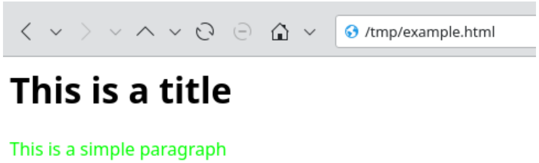
- The Cascade Style Sheet CSS is another standard that allows to graphically adjust the contents of your web page
- It provides decoupling between data and style.
- For instance, you want to write some text in red.
- You can do it inside the HTML page, but then. . .

## CSS Example

Try to save a file like this, and browse it

```
<html>
<body>
  <h1>This is a title</h1>
  <p style="color:#00FF00";>This is a simple paragraph</p>
</body>
</html>
```

This is what you should see



This is a title

This is a simple paragraph

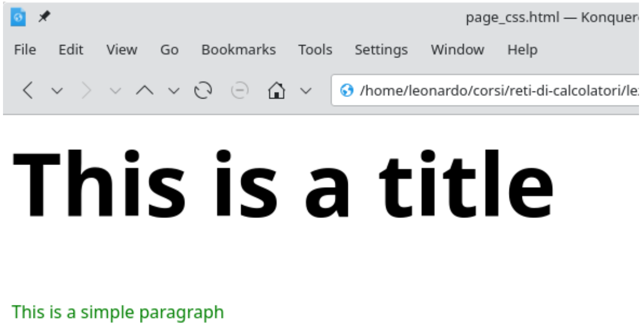
## Style issues

- If you want to modify the style of all paragraphs, you need to modify all the <p> tags
- CSS instead allow to specify the style in a different file, that is loaded by the browser with a `<link>` tag when the page is served
- On the left you have the `example.html` file. It includes the `style.css` file, that is on the right.

```html
<html>
<link href="style.css" rel="stylesheet" type="text/css">
<body>
  <h1>This is a title</h1>
  <p>This is a simple paragraph</p>
</body>
</html>
```

```css
h1 {
  color: black;
  font-size: 5em;
}
p {
  color: green;
}
```

## The Styled HTML


page_css.html — Konquer
File Edit View Go Bookmarks Tools Settings Window Help
/home/leonardo/corsi/reti-di-calcolatori/le

This is a title

This is a simple paragraph

## A melting Pot of *includes*

- Nowadays many sites load their components from a multitude of other sources, such as:
  - They load JS libraries from third parties, to exploit the newest features
  - They load CSS from known sites like Twitter Bootstrap or FontAwesome
  - They load images from external sites and caches
- Try it yourself. Open the Chrome browser, then "more tools"→"Developer tools"→"sources" (and "network") and load some popular web page like `repubblica.it`

## Sect. 3 HyperText Transfer Protocol HTTP

## HTTP

- The third component of the world wide web is the HyperText Transfer Protocol (HTTP), a text-based protocol like SMTP.
- The client sends a request and the server returns a response.
- HTTP runs above the bytestream service (TCP) and HTTP servers listen by default on port 80.

## HTTP

- Each HTTP request contains three parts :
  - a method, that indicates the type of request, a URI, and the version of the HTTP protocol used by the client
  - a header, that is used by the client to specify optional parameters for the request. An empty line is used to mark the end of the header
  - an optional MIME document attached to the request

## HTTP

- The response sent by the server also contains three parts :
  - a status line, that indicates whether the request was successful or not
  - a header, that contains additional information about the response. The response header ends with an empty line.
  - a MIME document

## HTTP Methods

The most popular methods are

- `GET`: used to retrieve a document from a server. It is followed by the path of the URI of the requested document and the version of HTTP used by the client.
- `HEAD`: allows the retrieval of the header lines for a given URI without retrieving the entire document. It can be used by a client to verify if a document exists, for instance.
- `POST`: used by a client to send a document to a server. The document is attached to the HTTP request as a MIME document.

## GET example

For example, to retrieve the `http://www.w3.org/MarkUp/` URI, a client must:

- open a TCP connection on port 80 with host www.w3.org
- send a HTTP request containing the following line:

```
GET /MarkUp/ HTTP/1.0
```

## HTTP headers

MIME headers are partly the same as in the SMTP protocol, for instance the ones that express the encoding. Some new ones are:

User-Agent: some info the client tells about the software running on the client itself. Used by servers to provide different versions of the website to different devices[1]

Referrer: the previous website that was browsed by this client. Very useful to understand what are the pages that link to your page and send you the more traffic.

Host: the domain the client requests, in case on the server there is more than one domain, with the same internal path (like `/index.html`)

Server: some info about the software running on the server. Not really used anymore, in order to avoid security scanners.

[1] Not a very good idea `https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent`

## HTTP Status Code

- The status line of the HTTP response includes the HTTP version followed by a three digit status code and additional information in English.
- HTTP status codes have a similar structure as the reply codes used by SMTP:

| | |
|---|---|
| 2** | Valid response. 200 Ok indicates that the HTTP request was OK. |
| 3** | Document is no longer available on the server. |
| 4** | the server has detected an error in the HTTP request sent by the client. 400 Bad Request indicates a syntax error in the HTTP request. 404 Not Found indicates that the requested document does not exist on the server. |
| 5** | All status codes starting with digit 5 indicate an error on the server. |

## Example

```
GET / HTTP/1.0
User-Agent: curl/7.19.4 (universal-apple-darwin10.0) libcurl/7.19.4 OpenSSL/0.9.8l zlib/1.2.3
Host: www.ietf.org
```

```
HTTP/1.1 200 OK
Date: Mon, 15 Mar 2010 13:40:38 GMT
Server: Apache/2.2.4 (Linux/SUSE) mod\_ssl/2.2.4 OpenSSL/0.9.8e (truncated)
Last-Modified: Tue, 09 Mar 2010 21:26:53 GMT
Content-Length: 17019
Content-Type: text/html
<!DOCTYPE HTML PUBLIC .../HTML>
```

## Persistent Connections

- Initially, one `TCP` connection was available for only one `HTTP` request, which was OK for simple pages
- Today we have `HTML` pages mades of hundreds of parts (images, icons, CSS...) and every one requires a different `HTTP` request.
- So to download a web page with 10 images, we need one for the page and one for each image.
- The `Connection: Keep Alive` header was added in order to allow to send more than one request in one connection only.
- The client sends the header, and the server answers with the same header specifying how many requests can be used in the rest of the connection

## Example

```
GET / HTTP/1.1
Host: www.kame.net
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_2; en-us)
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Fri, 19 Mar 2010 09:23:37 GMT
Server: Apache/2.0.63 (FreeBSD) PHP/5.2.12 with Suhosin-Patch
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Length: 3462
Content-Type: text/html
<html>...
</html>
```

## Stateless operation

- It is important to know that every HTTP request, being or not in the same connection is uncorrelated from the previous ones.
- The HTTP protocol does not maintain a state, or some link from one request to another
- This is different than saying that the backend (the software running on the server that creates and provides the pages to the HTTP server) does not maintain a state. In fact it does.

## Typical Example: Authentication

- After you perform an authentication, the website offers you different content
- The backend needs to remember that one `HTTP` request was made by the same client that made the authentication
- The most common way of doing it is with so-called `cookies`

## HTTP cookies

- A cookie is some information generated by the server and sent to the client
- The client will have to return the cookie at every request, in order to show that it is the same one that received the cookie initially.
- `HTTP` does not care about cookies, but offers two headers `Set-Cookie` and `Cookie` that can be used to move cookie back and forward between the server and the client.
- Whenever the client goes back to the same website, it may provide the cookie to be recognized as the same user

## Cookies. . .

- A cookie itself is just a seemingly random number, however,
- . . . entire books and international laws have been written on cookies, but their details mostly pertain to web programming and not to networking topics
- Some terminology and some key concepts follow

## Types of Cookies

Cookies can be distinguished on the way they are used:

- Session management: user log-in
- Personalization: client language or geographical location
- Tracking: profiling user activities

## A Typical Attack

- If Alice has authenticated to Bob's website and received a cookie, if Eve can access the cookie, it can impersonate Alice
- This is the case in which Eve is able, for instance, to poison the cache of a DNS server, and convince Alice to connect to Eve site instead of Bob's even only for some time
- Alice will connect to Eve site, provide the cookie, and Eve can impersonate Alice with Bob.

## HyperText Transfer Protocol HTTP

↳3.1 Proxies and HTTP 2.0

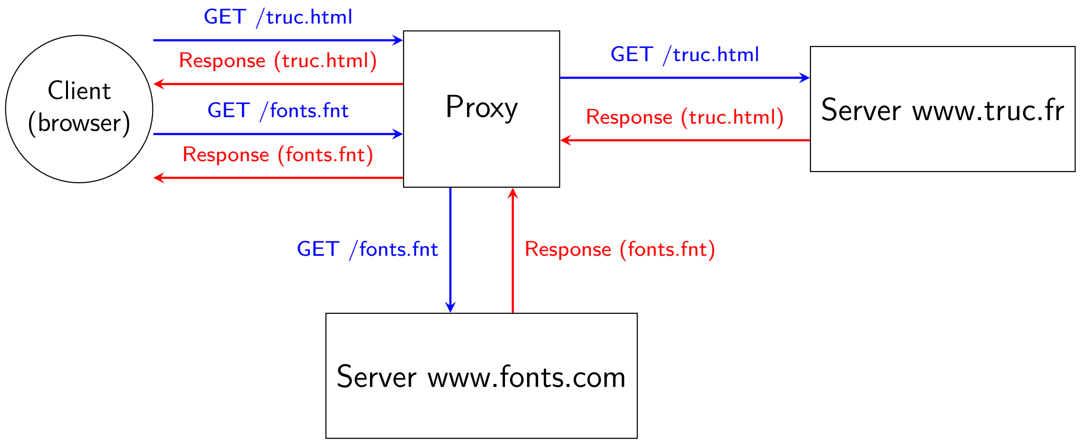
## Performance issues

- With time passing, the initial design of HTTP proved to be too slow for modern requirements
- In particular, a sequence of iterative calls to the server is too much depending on the Round-Trip-Time of the connection and so loading a page can take many seconds
- Plus, HTTP server now use dynamic pages and create pages from content included in databases, and the elaboration time it takes to retrieve all components is another source of delay
- We addressed the second issue with proxies, and the first with a totally new version of HTTP

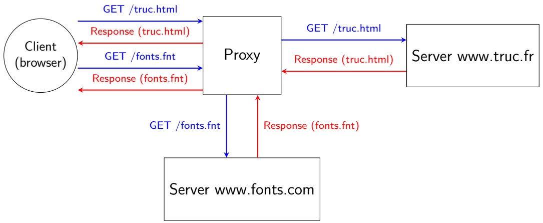
## Proxy Servers

- A proxy server is a cache that saves browsed pages for a certain amount of time
- It resides in the user network
- Browsers need to be configured to use the proxy and so the browser, instead of connecting to the destination `HTTP` server, will connect to the proxy
- The proxy will then connect to the original `HTTP` server, retrieve the content and save it for a while
- If the browser navigates the website again the proxy will not start a fresh connection but will simply serve the same page again

## Proxy Servers



## Proxy Servers



- In the example the web page is made of some HTML content and some fonts, coming from two different servers
- Once the proxy cached the whole page, it will serve it to further requests
- This has two benefits: speeds up the access to the page (no iterative requests) and saving in downlink bandwidth

## Proxy Server

- The proxy is set-up by the network administrator and can be accessed by the terminals in the local network
- It is a real MiTM so it must be carefully managed

## Interaction with HTTP

Some HTTP headers are used to know if the page has changed, and needs to be reloaded.

`Cache-Control`: This is used by the server to notify if the page can not be cached (`no-store`), can be cached up to a certain number of seconds (`max-age=10`), or it can be cached, but follow up requests must still ask to the server if some change happened using the `If-Modified-Since:` client header (`no-cache`)

`If-Modified-Since:` if client requests a page using this header, it is returned only if the page was modified since a certain time. Else it is not returned by the server, the cache is still valid.

## Reverse Proxies

- A reverse proxy is a proxy that is placed on the server premises instead that on the client ones
- The reverse proxy *hides* the real server, so that the client does not know the proxy exists
- The reverse proxy will query the servers and return the pages, exactly as the other kind of server.
- The goal of the reverse proxy is not to save bandiwdth, but to reduce the queries to the databases and the CPU load on the servers.

## HyperText Transfer Protocol HTTP

↳3.2 HTTP 2.0

## HTTP 2.0

- The most recent version of HTTP changed some of the internal ways HTTP works, with the goal of speeding up and changing some of the logic of HTTP
- The key features of HTTP 2.0 are the following ones. . .

## A binary protocol

- `HTTP 2.0` is not a text protocol anymore
- Now requests and replies are encoded in binary format
- Textual protocols are easy to debug and understand, but slow in practice, text can be largely compressed

## Parallel Streams

- The content of the page is split in *frames*
- This has nothing to deal with the (now obsolete) frames in `HTML`
- Frames can be sent in parallel data streams from the server to the client, so there is not anymore a strict sequence of requests that the client will make
- This allows the server to send smaller objects before larger ones (and not in the sequence they were asked by the client)

## HoL Blocking

- *head-of-line* blocking is a generic term to indicate a situation in which a large job slows down many small ones
- when the browser loads an `HTML` page it shows the content as soon as it gets downloaded
- So if the first content to be requested is a big image that takes time to be downloaded, this will block all the other elements and the web page will stop loading
- With `HTML 2.0` it's the server to decide which, among all the elements that the client requested, is the one to prioritize, so it can give priority to small textual ones.

## Push mechanism

- `HTTP 1.1` relied on a fully request-reply architecture
- in `HTTP 2.0` the server is allowed to *push* content to the client, even if the server did not ask for it
- This allows to save time, when the server knows that some content is required even before the client requests it.