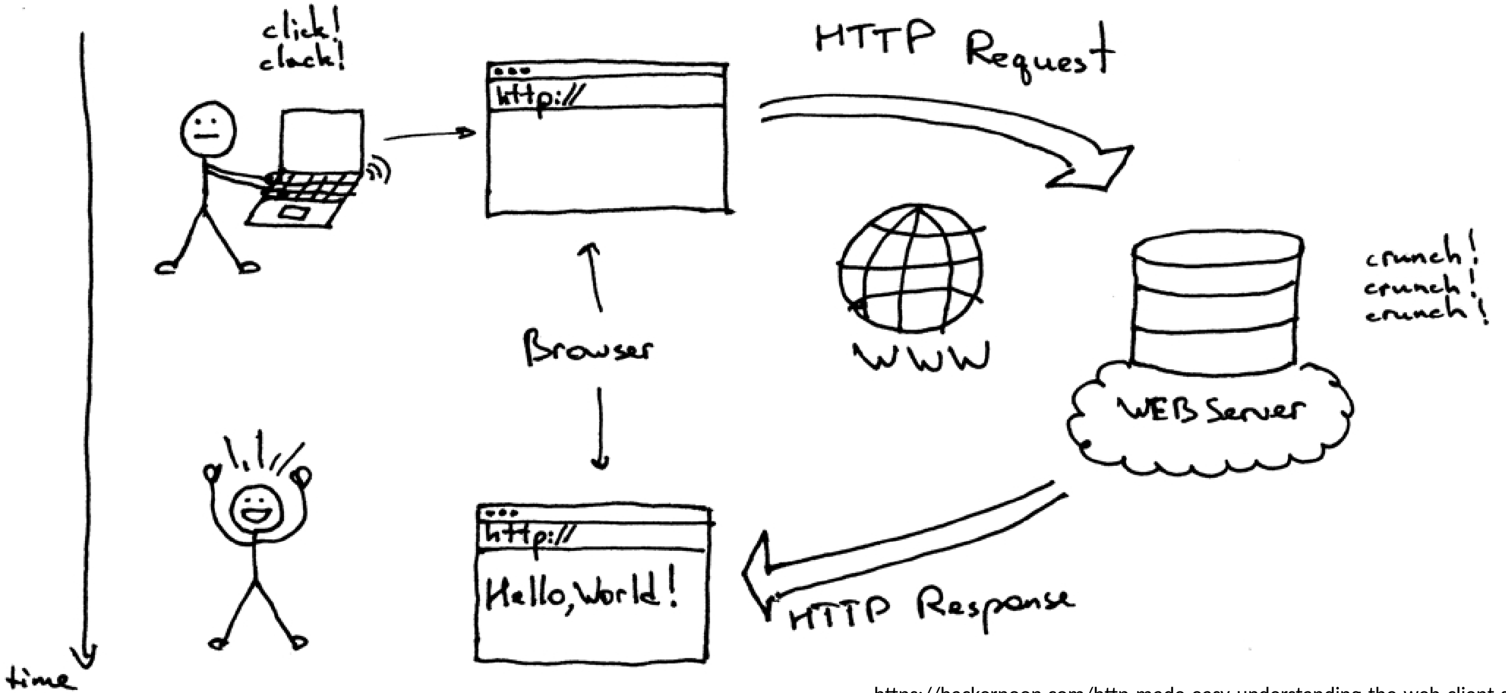
http://www.

The Web Services Protocol

- Application-layer protocol
- Client sends service requests using HTTP messages
- Server replies using HTTP messages

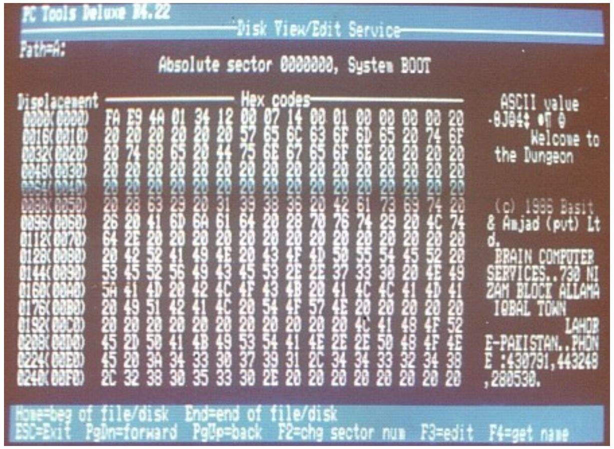


https://hackernoon.com/http-made-easy-understanding-the-web-client-server-communication-yz783vg3

When was HTTP first specified?

1981

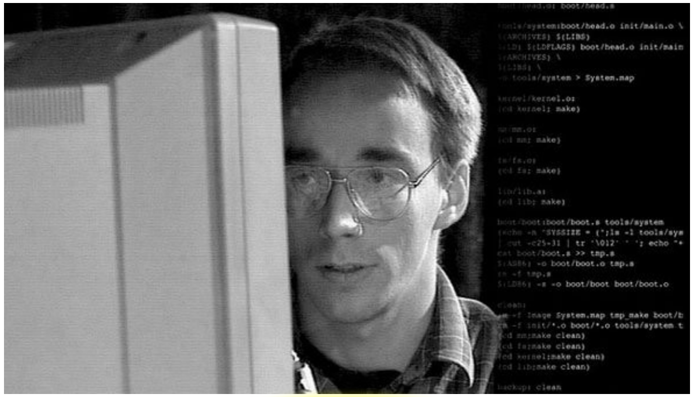
(IBM PC 5150)



```
PC Tools Deluxe R4.22
                         Disk View/Edit Service
Path=A:
              Absolute sector 0000000, System BOOT

Displacement ---------------- Hex codes ----------------      ASCII value
 0000(0000)  FA E9 4A 01 34 12 00 07 14 00 01 00 00 00 00 20  -ΘJ☺4↕ •¶ ☺
 0016(0010)  20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 6F        Welcome to
 0032(0020)  20 74 68 65 20 44 75 6E 67 65 6F 6E 20 20 20 20   the Dungeon
 0048(0030)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0064(0040)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0080(0050)  20 28 63 29 20 31 39 38 36 20 42 61 73 69 74 20   (c) 1986 Basit
 0096(0060)  26 20 41 6D 6A 61 64 20 28 70 76 74 29 20 4C 74  & Amjad (pvt) Lt
 0112(0070)  64 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20  d.
 0128(0080)  20 42 52 41 49 4E 20 43 4F 4D 50 55 54 45 52 20   BRAIN COMPUTER
 0144(0090)  53 45 52 56 49 43 45 53 2E 2E 37 33 30 20 4E 49  SERVICES..730 NI
 0160(00A0)  5A 41 4D 20 42 4C 4F 43 4B 20 41 4C 4C 41 4D 41  ZAM BLOCK ALLAMA
 0176(00B0)  20 49 51 42 41 4C 20 54 4F 57 4E 20 20 20 20 20   IQBAL TOWN
 0192(00C0)  20 20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52             LAHOR
 0208(00D0)  45 2D 50 41 4B 49 53 54 41 4E 2E 2E 50 48 4F 4E  E-PAKISTAN..PHON
 0224(00E0)  45 20 3A 34 33 30 37 39 31 2C 34 34 33 32 34 38  E :430791,443248
 0240(00F0)  2C 32 38 30 35 33 30 2E 20 20 20 20 20 20 20 20  ,280530.

Home=beg of file/disk  End=end of file/disk
ESC=Exit  PgDn=forward  PgUp=back  F2=chg sector num  F3=edit  F4=get name
```

1986

(Brain: first computer virus for MS-DOS)



1991

(Linus Torvalds introduces Linux)

1996

(Google search engine)

Hypertext Transport Protocol

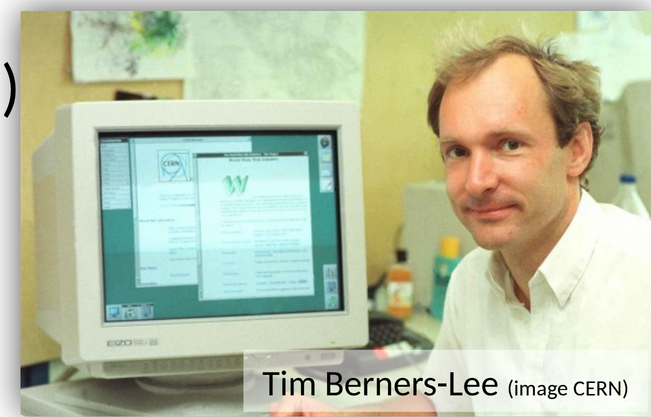
- 1989-90, Tim Berners-Lee’s problem at CERN:

  how to integrate and exchange information held on different computers in scattered places?

- Already exist:
  - TCP: reliable transport of information on the Internet
  - DNS: domain name (“www.centralesupelec.fr”) ↔ IP @ (“138.195.9.117”)

    Human friendly

    Computer friendly
  - object in a database that *references* others
- Put them all together: HTTP
  - Retrieve linked documents (resources)
  - Accessible via the Internet



Tim Berners-Lee (image CERN)

Client-Server Protocol request / response

Client
(web browser or other application)

1. user clicks on hyperlink

2. HTTP request message

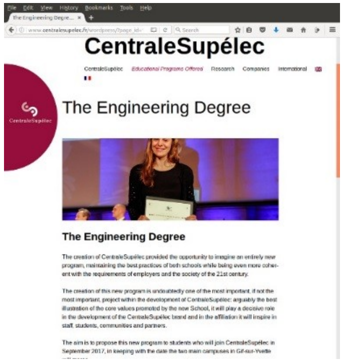
4. HTTP response message

web server

back-end
(database…
)

3. create or retrieve file

5. display file


File Edit View History Bookmarks Tools Help
The Engineering Degre... ×
www.centralesupelec.fr/wordpress/?page_id=
Search
CentraleSupélec

CentraleSupélec

CentraleSupélec Educational Programs Offered Research Companies International

The Engineering Degree

The Engineering Degree

The creation of CentraleSupélec provided the opportunity to imagine an entirely new program, maintaining the best practices of both schools while being even more coherent with the requirements of employers and the society of the 21st century.

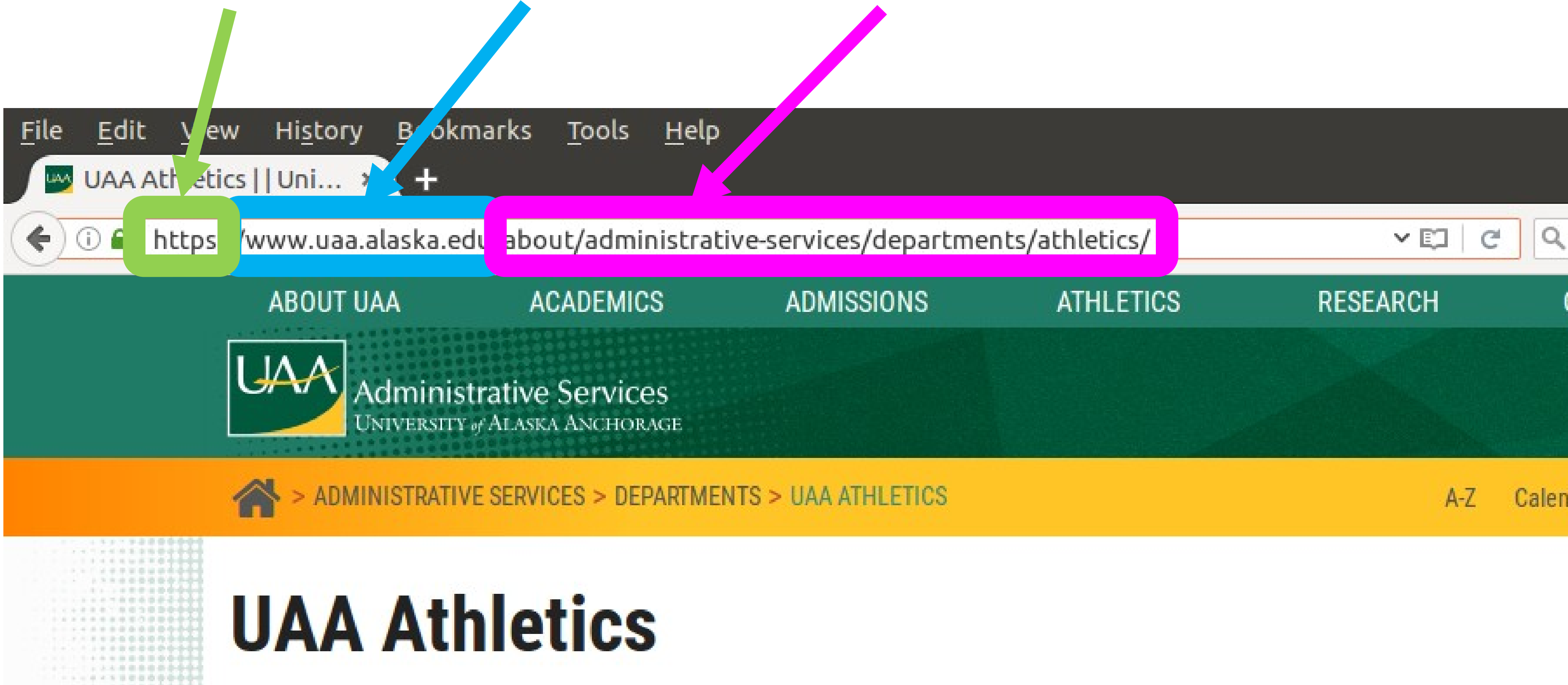
The creation of this new program is undoubtedly one of the most important, if not the most important, project within the development of CentraleSupélec: arguably the best illustration of the core values promoted by the new School, it will play a decisive role in the development of the CentraleSupélec brand and in the affiliation it will inspire in staff, students, communities and partners.

The aim is to propose this new program to students who will join CentraleSupélec in September 2017, in keeping with the date the two main campuses in Gif-sur-Yvette

Resources = addressable files

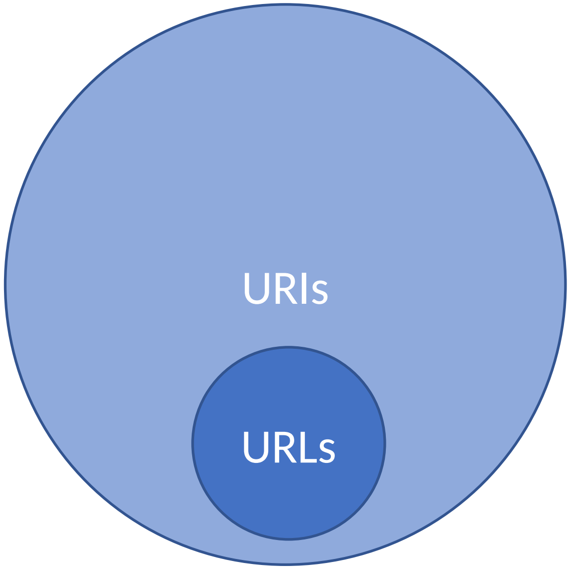
- Any kind of file: HTML file, JPG image file, binary file...
- URL (Uniform Resource Locator)

= protocol + server host name + path on server



Side note about URL and URI

- URI: identifier (name of a restaurant)
- URL: locator (GPS coordinates of the restaurant)



All URLs are URIs:

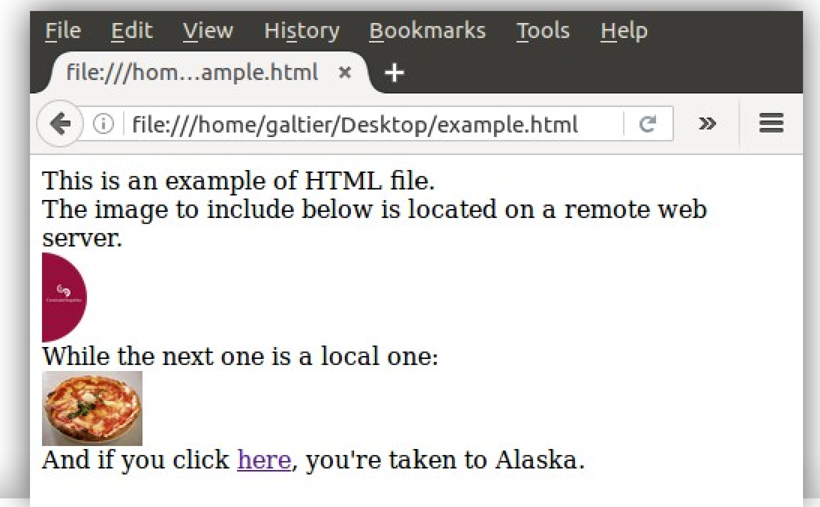
with the GPS coordinates I arrive to the right restaurant

Not all URIs are URLs:

the name of the restaurant gives no information on its location

HTML file may include references to others resources

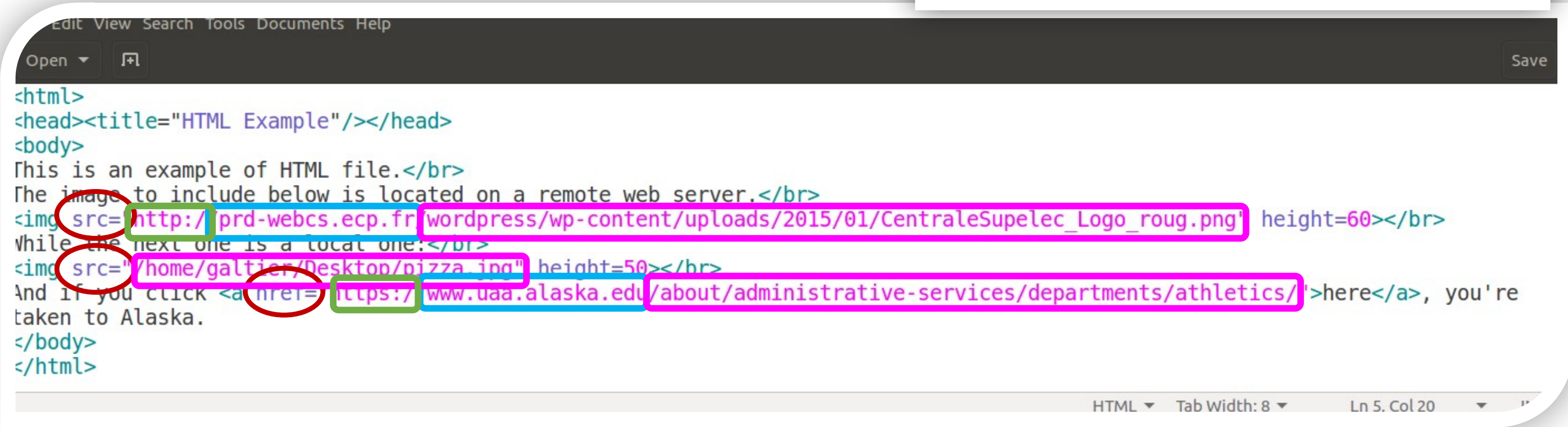
- 3 resources are required to display this web page:
  - HTML file
  - CentraleSupelec logo image
  - Pizza image



This is an example of HTML file.
The image to include below is located on a remote web server.

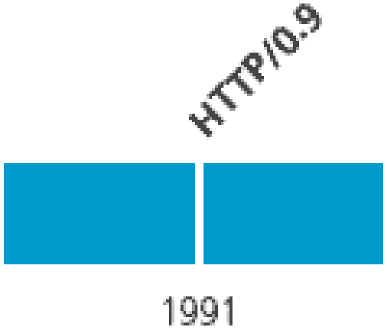
While the next one is a local one:

And if you click here, you're taken to Alaska.

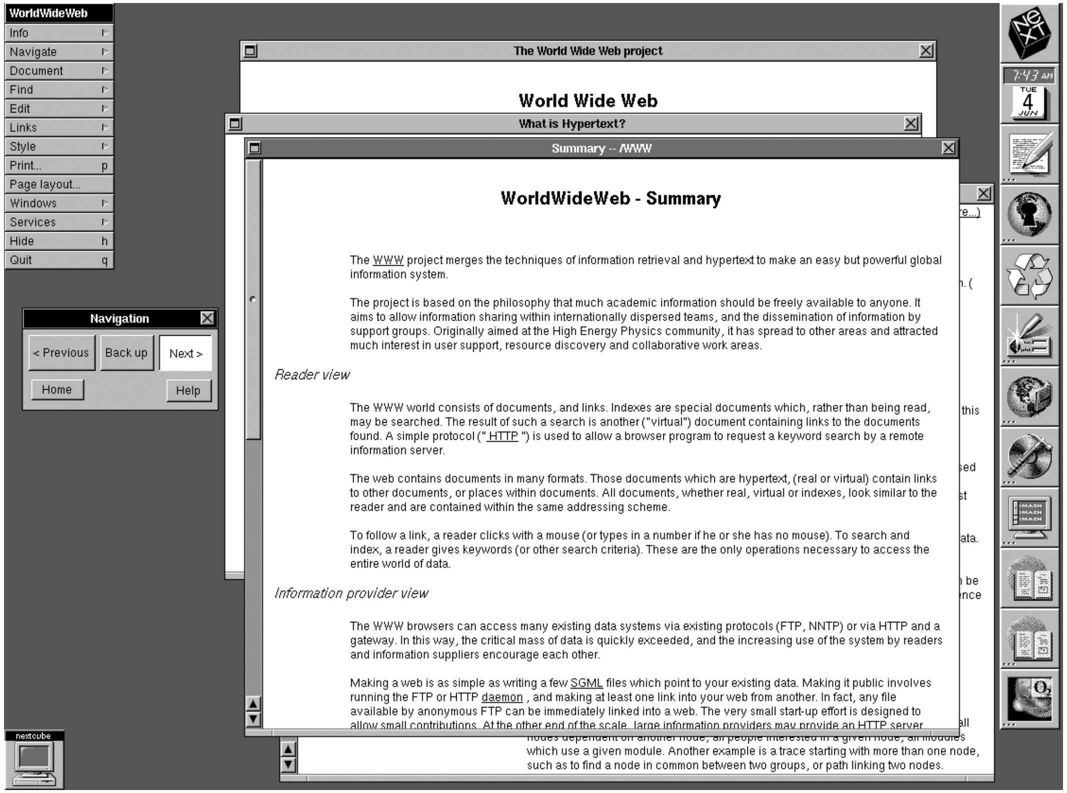


```
<html>
<head><title="HTML Example"/></head>
<body>
This is an example of HTML file.</br>
The image to include below is located on a remote web server.</br>
<img src='http://prd-webcs.ecp.fr/wordpress/wp-content/uploads/2015/01/CentraleSupelec_Logo_roug.png' height=60></br>
While the next one is a local one:</br>
<img src="/home/galtier/Desktop/pizza.jpg" height=50></br>
And if you click <a href='https://www.uaa.alaska.edu/about/administrative-services/departments/athletics/'>here</a>, you're
taken to Alaska.
</body>
</html>
```

HTTP Versions



- 1991 – v0.9
  - First documented version
  - First web browser



WorldWideWeb - Summary

The WWW project merges the techniques of information retrieval and hypertext to make an easy but powerful global information system.

The project is based on the philosophy that much academic information should be freely available to anyone. It aims to allow information sharing within internationally dispersed teams, and the dissemination of information by support groups. Originally aimed at the High Energy Physics community, it has spread to other areas and attracted much interest in user support, resource discovery and collaborative work areas.

Reader view

The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another ("virtual") document containing links to the documents found. A simple protocol (" HTTP ") is used to allow a browser program to request a keyword search by a remote information server.

The web contains documents in many formats. Those documents which are hypertext, (real or virtual) contain links to other documents, or places within documents. All documents, whether real, virtual or indexes, look similar to the reader and are contained within the same addressing scheme.

To follow a link, a reader clicks with a mouse (or types in a number if he or she has no mouse). To search and index, a reader gives keywords (or other search criteria). These are the only operations necessary to access the entire world of data.

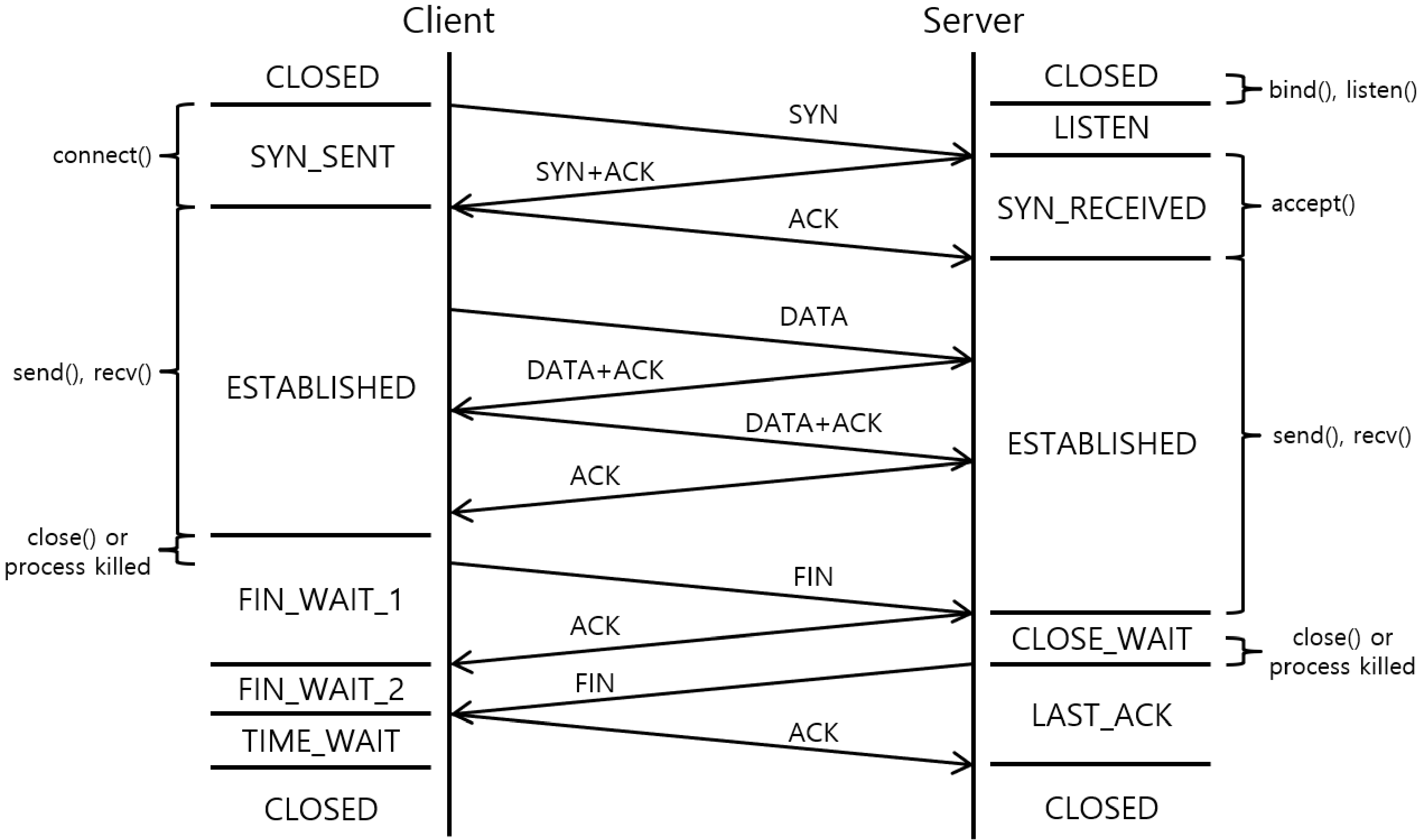
Information provider view

The WWW browsers can access many existing data systems via existing protocols (FTP, NNTP) or via HTTP and a gateway. In this way, the critical mass of data is quickly exceeded, and the increasing use of the system by readers and information suppliers encourage each other.

Making a web is as simple as writing a few SGML files which point to your existing data. Making it public involves running the FTP or HTTP daemon , and making at least one link into your web from another. In fact, any file available by anonymous FTP can be immediately linked into a web. The very small start-up effort is designed to allow small contributions. At the other end of the scale, large information providers may provide an HTTP server

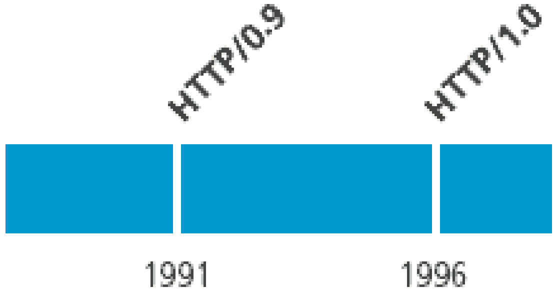
which use a given module. Another example is a trace starting with more than one node, such as to find a node in common between two groups, or path linking two nodes.

Side note on TCP 3-way handshake

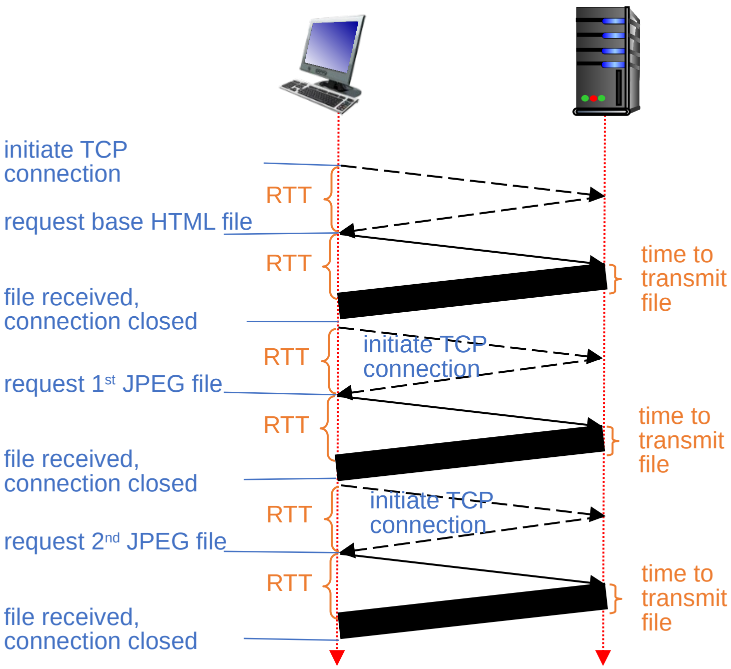


https://ssup2.github.io/theory_analysis/TCP_Handshake/

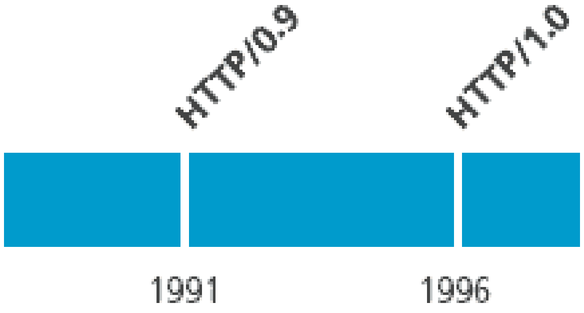
HTTP Versions



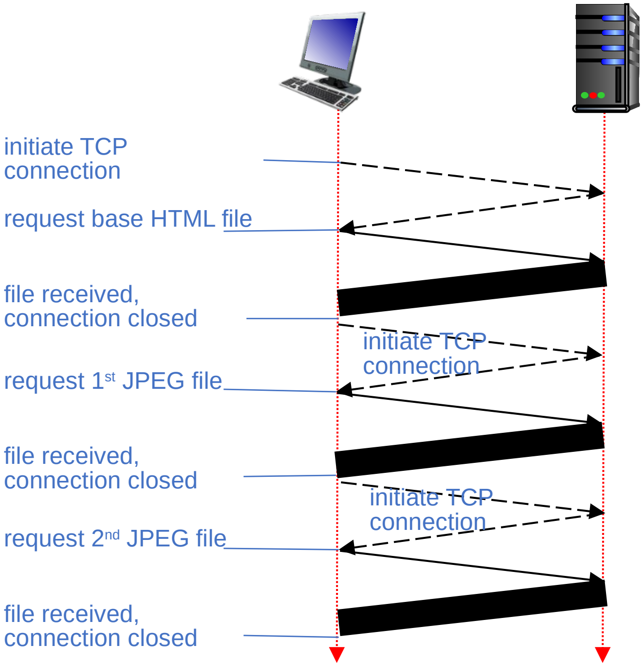
- 1996 – v1.0
  - One TCP connection per resource



HTTP Versions



- 1996 – v1.0
    - One TCP connection per resource


initiate TCP connection

request base HTML file

file received, connection closed

initiate TCP connection

request 1st JPEG file

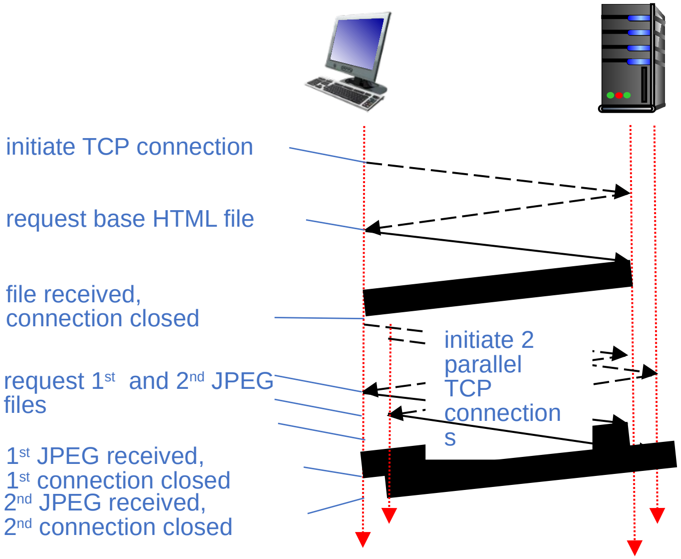
file received, connection closed

initiate TCP connection

request 2nd JPEG file

file received, connection closed

with parallel connections


initiate TCP connection

request base HTML file

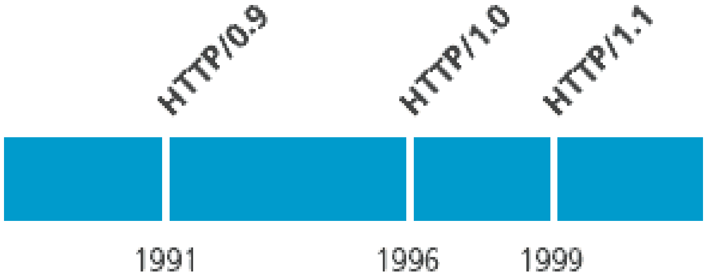
file received, connection closed

initiate 2 parallel TCP connection s

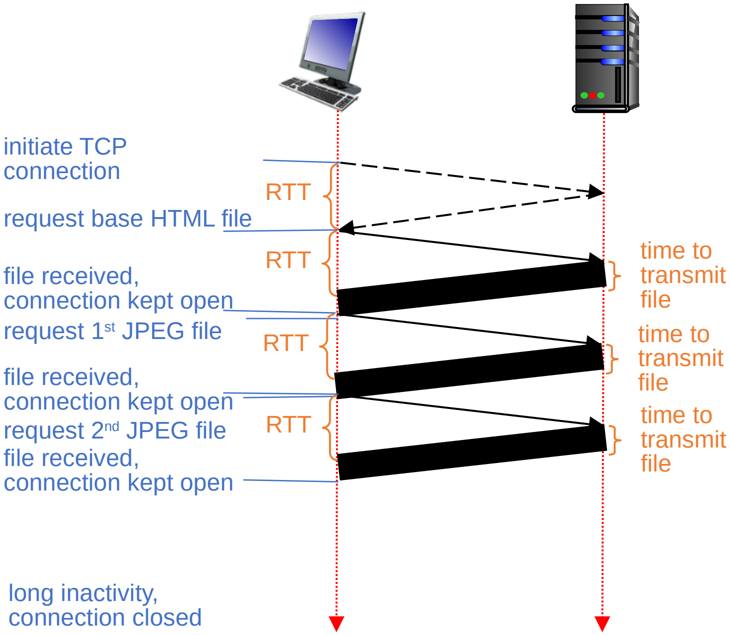
request 1st and 2nd JPEG files

1st JPEG received, 1st connection closed 2nd JPEG received, 2nd connection closed

HTTP Versions

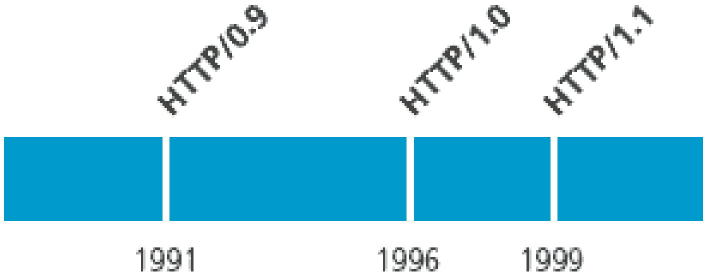


- 1999 – v1.1
  - Persistent connection

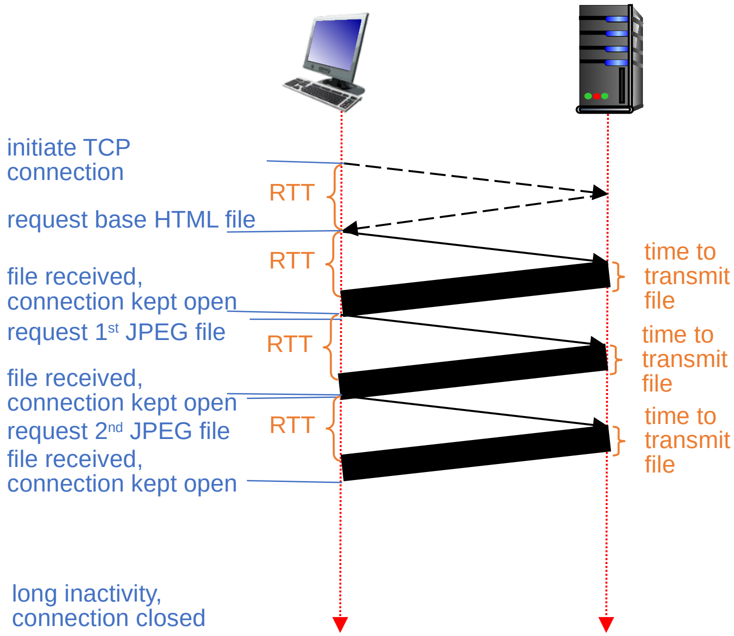


```
GET https://www.centralesupelec.fr/ HTTP/1.1
Host: www.centralesupelec.fr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86
Accept: text/html,application/xhtml+xml,applica
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

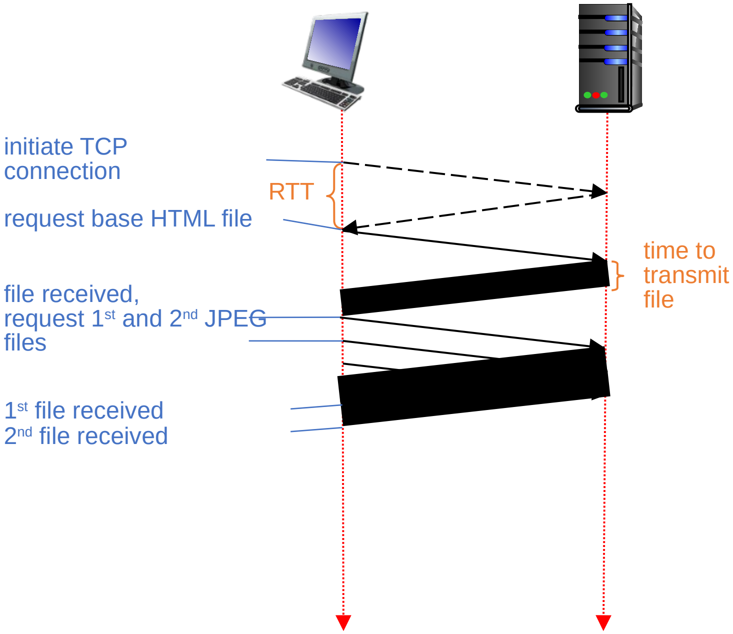
HTTP Versions



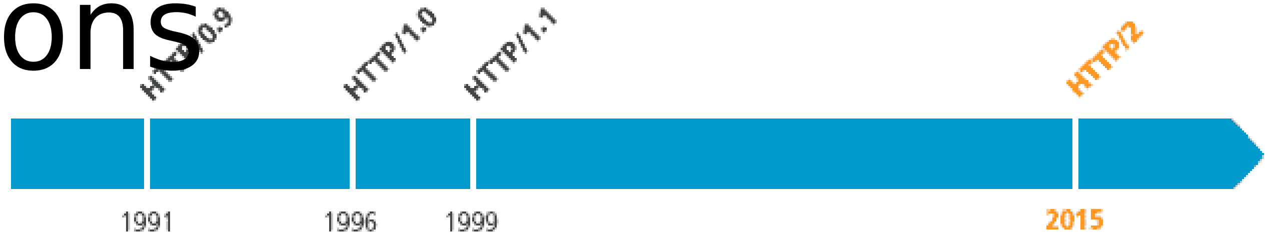
- 1999 – v1.1
  - Persistent connection



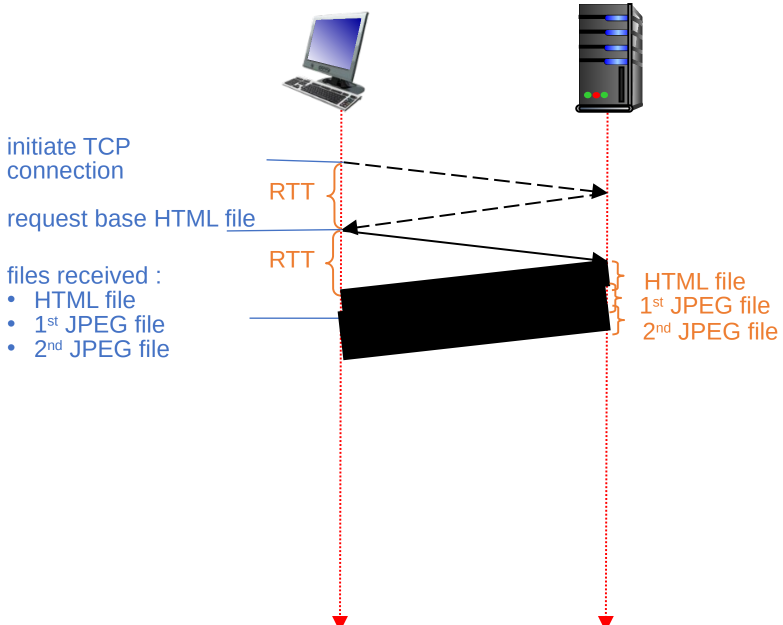
with pipelining



HTTP Versions



- 2015 – v2
  - Server “pushes” content
  - [and other optimizations]



initiate TCP connection

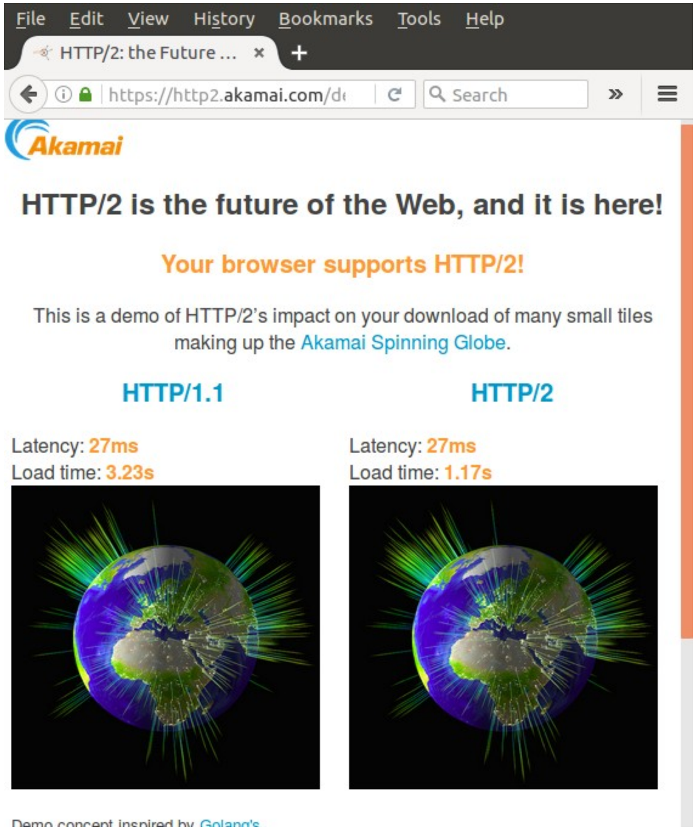
RTT

request base HTML file

RTT

files received :
- HTML file
- 1st JPEG file
- 2nd JPEG file

HTML file
1st JPEG file
2nd JPEG file


File Edit View History Bookmarks Tools Help
HTTP/2: the Future ...
https://http2.akamai.com/d
Search
Akamai

HTTP/2 is the future of the Web, and it is here!

Your browser supports HTTP/2!

This is a demo of HTTP/2's impact on your download of many small tiles making up the Akamai Spinning Globe.

HTTP/1.1	HTTP/2
Latency: 27ms	Latency: 27ms
Load time: 3.23s	Load time: 1.17s

Demo concept inspired by Golang's

Optional Reading Exercise

- Find the document which describes HTTP/2.
- What is the “head-of-line blocking” (HOL blocking) problem observed in HTTP/1.1?
- Read the beginning of the FAQ at https://http2.github.io/faq/

Reading: Results

- HTTP/2 is defined in RFC 7540.
- HOL blocking:
    - Imagine a HTTP client that sends to a server 2 requests over the same TCP connection, and that the first response is "large" in content length while the second response is "small" in content length.
    - Due to the nature of the HTTP 1.x protocol, the second response must wait for the first response to complete: the second response is *head-of-line blocked* by the first response.
    - HTTP/2 is fully multiplexed (instead of ordered and blocking), allowing multiple request and response messages to be in flight at the same time (it's even possible to intermingle parts of one message with another on the wire).

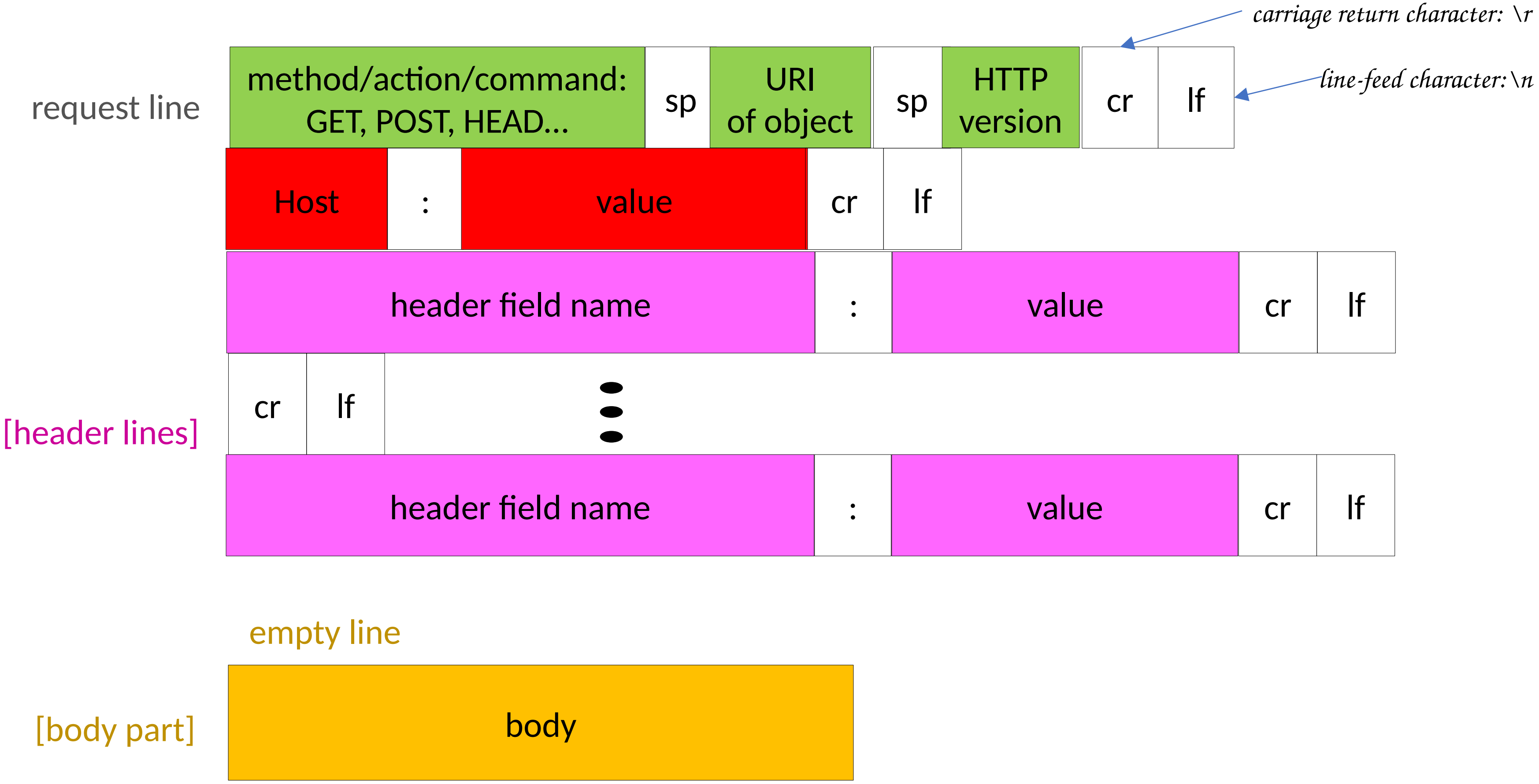
HTTP Messages

- 2 kinds of messages
  - Request
  - Response
- In ASCII (HTTP 1.x)

HTTP Requests Commands

- GET
  - retrieves an object
  - no request body
- HEAD
  - same response as GET but empty response body (used to test the access to or the "freshness" of the object without actually downloading it)
- POST
  - results in the creation of a new resource on the server
  - usual request: contains data
  - usual response: URL of the created resource
- PUT
  - updates an existing resource
  - request usually contains data
- DELETE
  - deletes a resource

HTTP Request Format



carriage return character: \r

line-feed character:\n

request line

method/action/command: GET, POST, HEAD... | sp | URI of object | sp | HTTP version | cr | lf

Host | : | value | cr | lf

header field name | : | value | cr | lf

[header lines]

cr | lf

header field name | : | value | cr | lf

empty line

[body part]

body

HTTP GET Request Example

```
GET /node/44 HTTP/1.1\r\n
Host: mapi.centralesupelec.fr\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP POST Request Example

```
POST /post.php HTTP/1.1\r\n
Host: posttestserver.com\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Content-Type: text/xml\r\n
Content-Length: 27\r\n
Connection: keep-alive\r\n
\r\n
firstname=John\nlastname=Doe
```

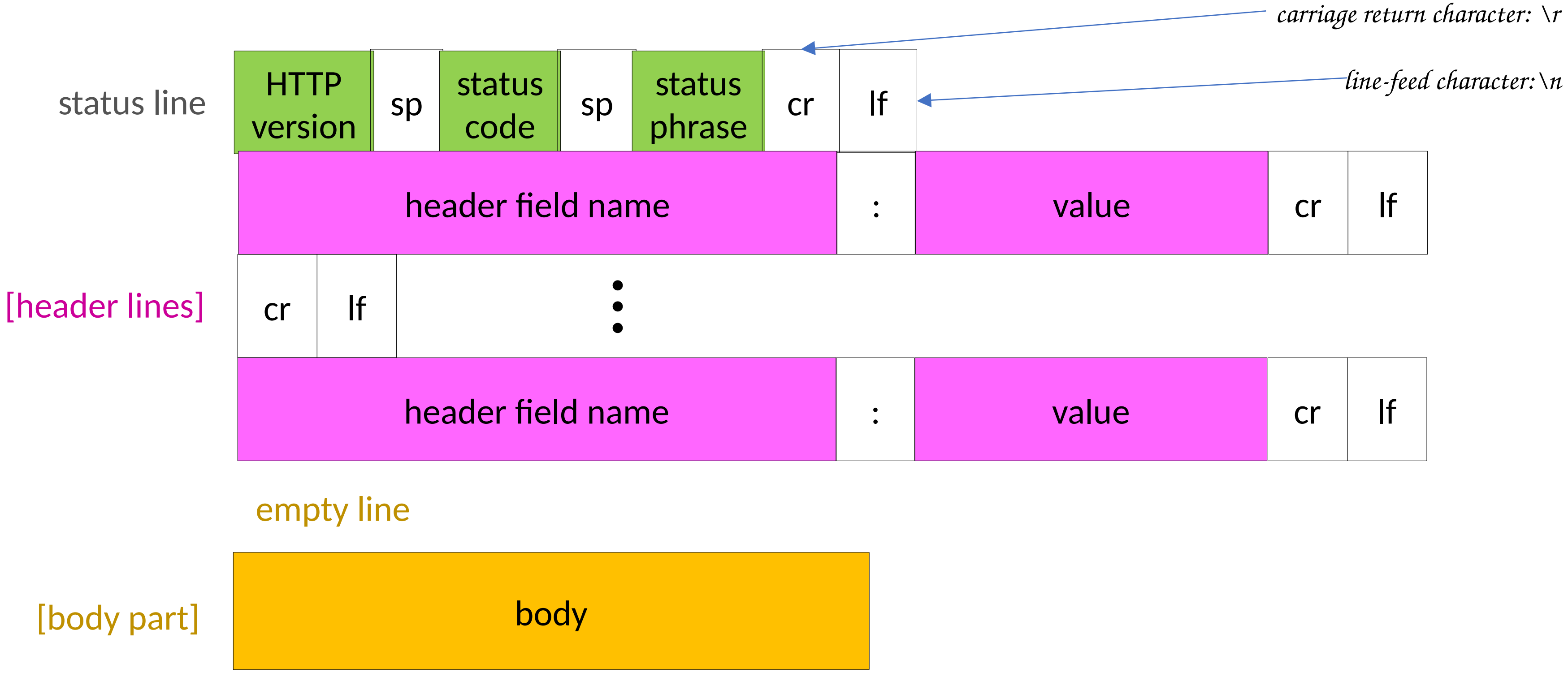
Request Parameters

3 symbols to add parameters to an URL:

- ? concatenates the URL and the string of parameters
- & separates multiple parameters
- = assigns a value to a parameter

```
GET /products?priceMin=10&priceMax=40
```

HTTP Response Format



status line

HTTP version | sp | status code | sp | status phrase | cr | lf

carriage return character: \r

line-feed character:\n

[header lines]

header field name | : | value | cr | lf

⋮

header field name | : | value | cr | lf

empty line

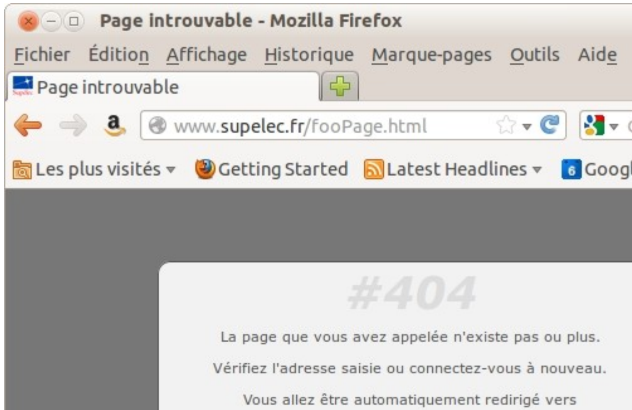
cr | lf

[body part]

body

Status Codes

- 2xx: success
  - 200 OK
- 3xx: further action required
  - 301 Moved Permanently: the new URL is specified in a header field
- 4xx: client error
  - 400 Bad Request: badly formulated query
  - 404 Not Found: object does not exist on the server
- 5xx: server-side error
  - 505 HTTP Version Not Supported



#404

La page que vous avez appelée n'existe pas ou plus.

Vérifiez l'adresse saisie ou connectez-vous à nouveau.

Vous allez être automatiquement redirigé vers

HTTP Response Example

```
HTTP/1.1 200 OK
Date: Wed, 01 Feb 2017 12:48:22 GMT
Server: Apache/2.4.10 (Debian)
[...]
Content-language: fr
Content-Encoding: gzip
Content-Length: 4740
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

..........;.r.8...W...-{.(.KO..!K..-.$..q;.(...D.4..T.v.\.....q.{...Z.2_....b.....(I....Df"..Hn...|....}.WQ..m....
WD.sR)..J.....L:9.C..MC...X.I...
J..(...'"....J. D....d%bN,. $..Y..........z.............y(.MS....#.qV.....>.9.j.0
s&...v.M...')......m8..<=.i..%B........S.x}.J.:V..{.".HM..4b..!.YJ......X{i...I.;.T.X}....N.r .<d...#.........S..
..#Oa.       ...V..EPj..G...A..D.K...Z1..c.h,b.4..b.3...I.6..La..>.L8#l.U.\.......2..y!...S.,.....%.....>..ID...
N..^
```

HTTP Response Example

```
HTTP/1.1 404 Not Found
Date: Wed, 01 Feb 2017 13:14:55 GMT
Server: Apache/2.4.10 (Debian)
[…]
Content-language: fr
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

305d
<!DOCTYPE html>
<html lang="fr" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/  dc: ht
<div id="block-zircon-content" class="block block-system block-system-main-block">
    La page demand..e n'a pas pu ..tre trouv..e.
 </div>
```

Optional Reading

- What is HTTP error code 418?

https://www.neper.fr/2023/01/16/les-easter-eggs-de-google-418-teapot/

Hyper Text Coffee Pot Control Protocol

14 languages

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

The **Hyper Text Coffee Pot Control Protocol** (**HTCPCP**) is a facetious communication protocol for controlling, monitoring, and diagnosing coffee pots. It is specified in RFC 2324, published on 1 April 1998 as an April Fools' Day RFC,[2] as part of an April Fools prank.[3] An extension, HTCPCP-TEA, was published as RFC 7168 on 1 April 2014[4] to support brewing teas, which is also an April Fools' Day RFC.

Hyper Text Coffee Pot Control Protocol

Back-end infrastructure of error418.net, which implements HTCPCP

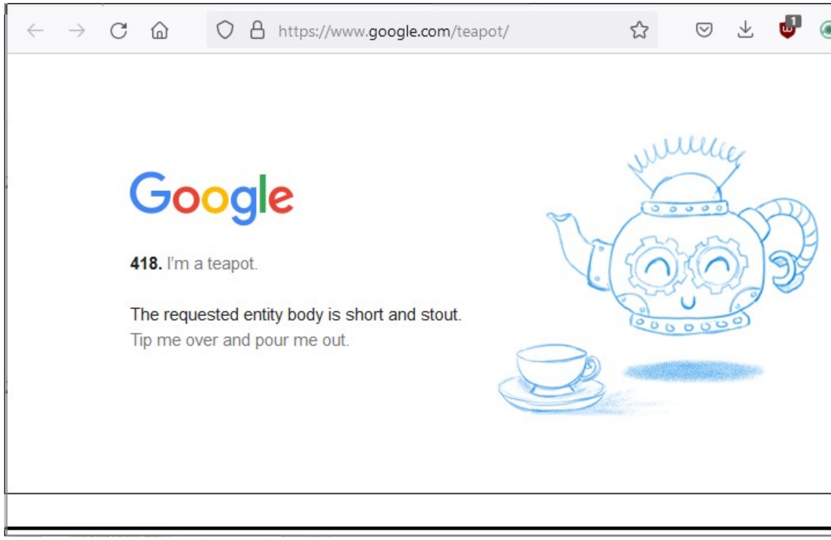
International standard	Internet Engineering Task Force
Developed by	Larry Masinter
Introduced	April 1, 1998
Website	rfc2324

Protocol [edit]

RFC 2324 was written by Larry Masinter, who describes it as a satire, saying "This has a serious purpose – it identifies many of the ways in which HTTP has been extended inappropriately."[5] The wording of the protocol made it clear that it was not entirely serious; for example, it notes that "there is a strong, dark, rich requirement for a protocol designed espressoly [*sic*] for the brewing of coffee".

Despite the joking nature of its origins, or perhaps because of it, the protocol has remained as a minor presence online. The editor Emacs includes a fully functional client side implementation of it,[6] and a number of bug reports exist complaining about Mozilla's lack of support for the protocol.[7] Ten years after the publication of HTCPCP, the *Web-Controlled Coffee Consortium* (*WC3*) published a first draft of "HTCPCP Vocabulary in RDF"[8] in parody of the World Wide Web Consortium's (W3C) "HTTP Vocabulary in RDF".[9]

On April 1, 2014, RFC 7168 extended HTCPCP to fully handle teapots.[4]



https://www.google.com/teapot/

Google

418. I'm a teapot.

The requested entity body is short and stout.
Tip me over and pour me out.

Commands and replies [edit]

HTCPCP is an extension of HTTP. HTCPCP requests are identified with the Uniform Resource Identifier (URI) scheme `coffee` (or the corresponding word in any other of the 29 listed languages) and contain several additions to the HTTP methods:

`BREW` or `POST`	Causes the HTCPCP server to brew coffee. Using POST for this purpose is deprecated. A new HTTP request header field "Accept-Additions" is proposed, supporting optional additions including Cream, Whole-milk, Vanilla, Raspberry, Whisky, Aquavit, etc.
`GET`	"Retrieves" coffee from the HTCPCP server.
`PROPFIND`	Returns metadata about the coffee.
`WHEN`	Says "when", causing the HTCPCP server to stop pouring milk into the coffee (if applicable).

It also defines two error responses:

`406 Not Acceptable`	The HTCPCP server is unable to provide the requested addition for some reason; the response should indicate a list of available additions. The RFC observes that "In practice, most automated coffee pots cannot currently provide additions."
`418 I'm a teapot`	The HTCPCP server is a teapot; the resulting entity body "may be short and stout" (a reference to the song "I'm a Little Teapot"). Demonstrations of this behaviour exist.[1][10]

Optional Lab Exercise

- Use putty or telnet to connect to port 80 of a web server (http://www.columbia.edu for instance) and issue HTTP/1.x requests (get /~fdc/sample.html). Observe the responses.

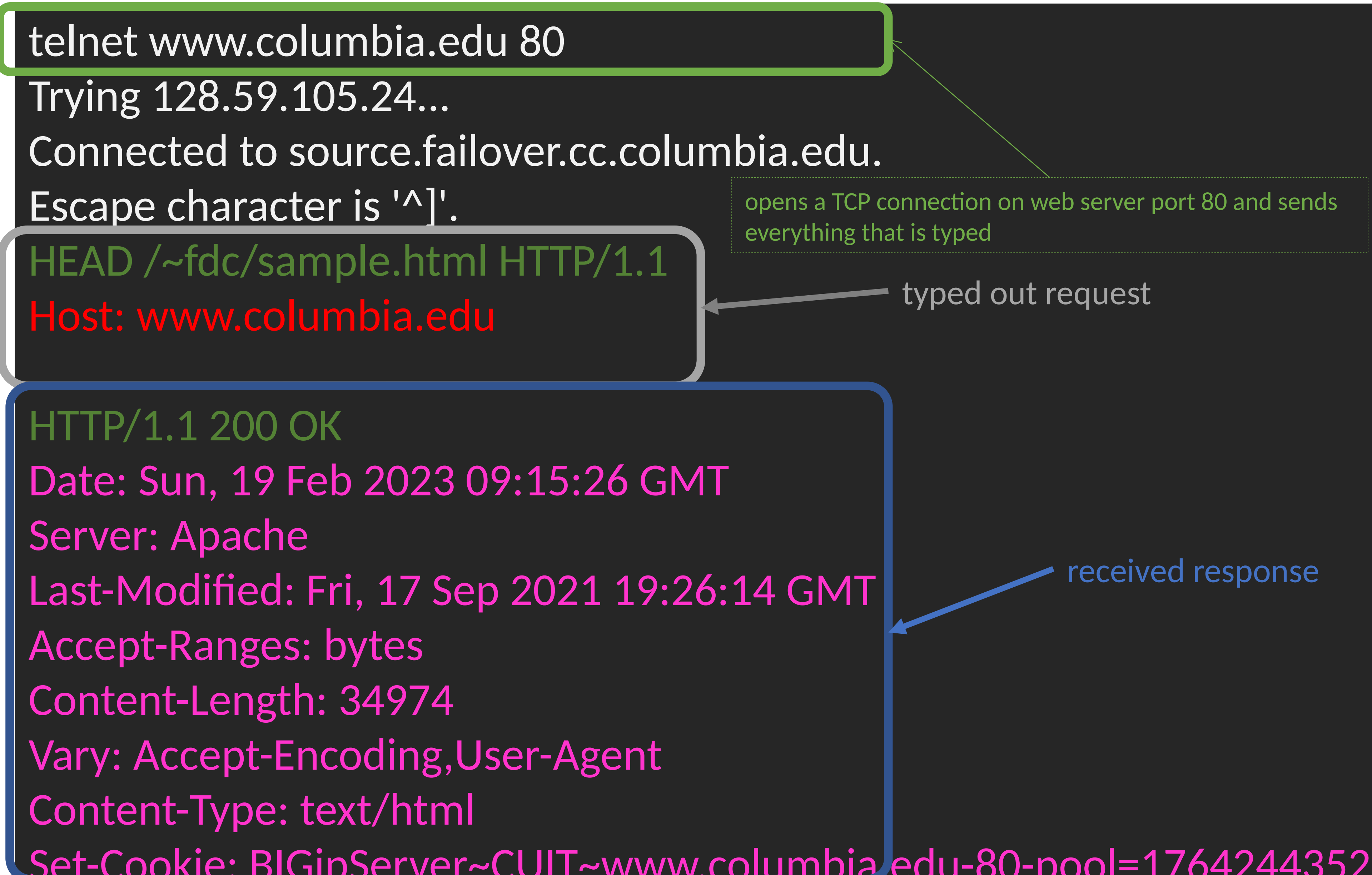
  ```
  telnet serverName 80
  ```

- For HTTPS, use:

  ```
  openssl s_client -connect
  serverName:443
  ```

(note: this exercise is limited to HTTP/1.x because HTTP/2 is no longer textual but uses binary format commands)

Lab: Results



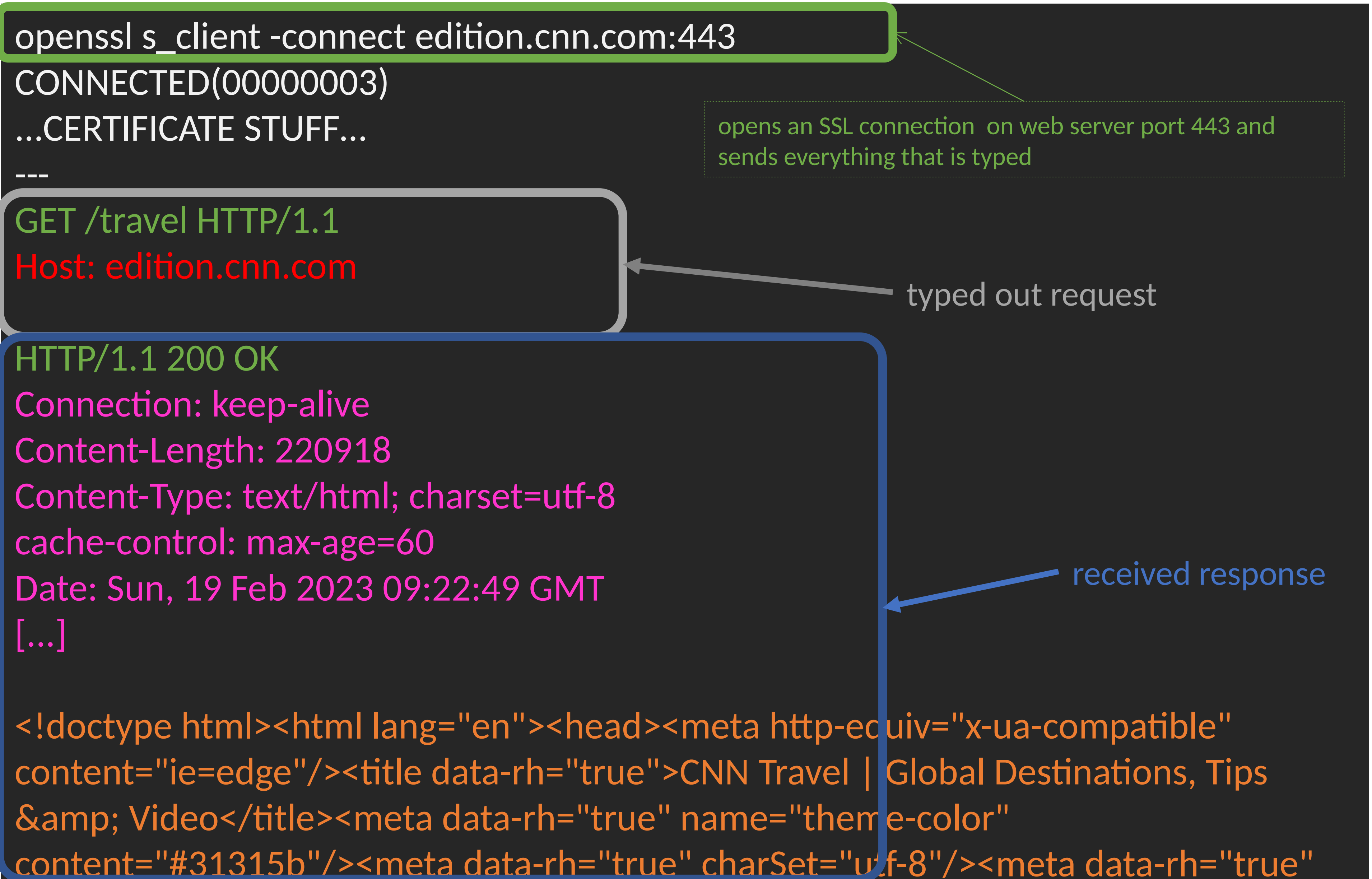
```
telnet www.columbia.edu 80
Trying 128.59.105.24...
Connected to source.failover.cc.columbia.edu.
Escape character is '^]'.
HEAD /~fdc/sample.html HTTP/1.1
Host: www.columbia.edu

HTTP/1.1 200 OK
Date: Sun, 19 Feb 2023 09:15:26 GMT
Server: Apache
Last-Modified: Fri, 17 Sep 2021 19:26:14 GMT
Accept-Ranges: bytes
Content-Length: 34974
Vary: Accept-Encoding,User-Agent
Content-Type: text/html
Set-Cookie: BIGipServer~CUIT~www.columbia.edu-80-pool=1764244352
```

opens a TCP connection on web server port 80 and sends everything that is typed

typed out request

received response

Lab: Results



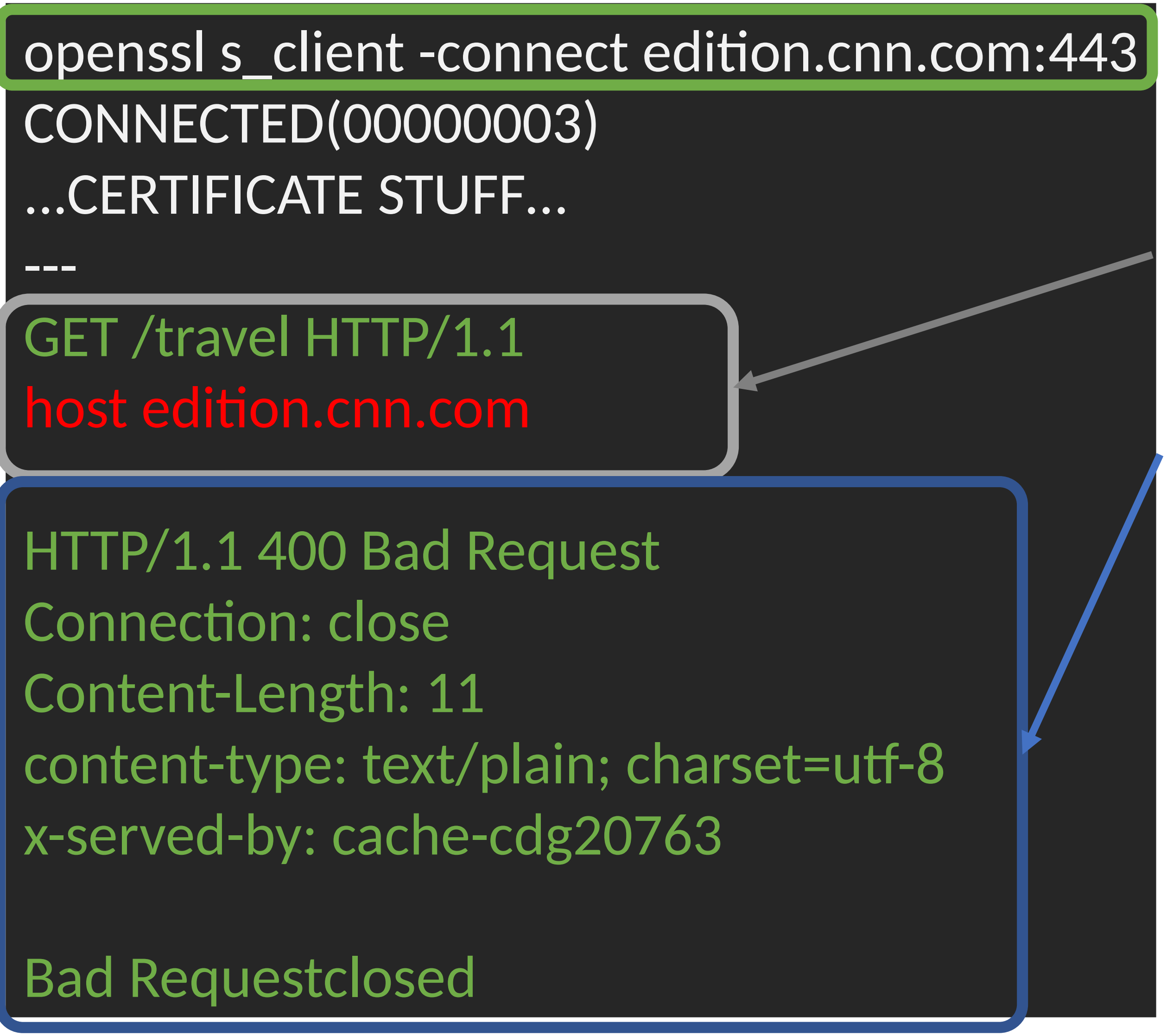
```
openssl s_client -connect edition.cnn.com:443
CONNECTED(00000003)
...CERTIFICATE STUFF...
---
GET /travel HTTP/1.1
Host: edition.cnn.com

HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 220918
Content-Type: text/html; charset=utf-8
cache-control: max-age=60
Date: Sun, 19 Feb 2023 09:22:49 GMT
[...]

<!doctype html><html lang="en"><head><meta http-equiv="x-ua-compatible"
content="ie=edge"/><title data-rh="true">CNN Travel | Global Destinations, Tips
&amp; Video</title><meta data-rh="true" name="theme-color"
content="#31315b"/><meta data-rh="true" charSet="utf-8"/><meta data-rh="true"
```

opens an SSL connection on web server port 443 and sends everything that is typed

typed out request

received response

Lab: Results



```
openssl s_client -connect edition.cnn.com:443
CONNECTED(00000003)
...CERTIFICATE STUFF...
---
GET /travel HTTP/1.1
host edition.cnn.com

HTTP/1.1 400 Bad Request
Connection: close
Content-Length: 11
content-type: text/plain; charset=utf-8
x-served-by: cache-cdg20763

Bad Requestclosed
```

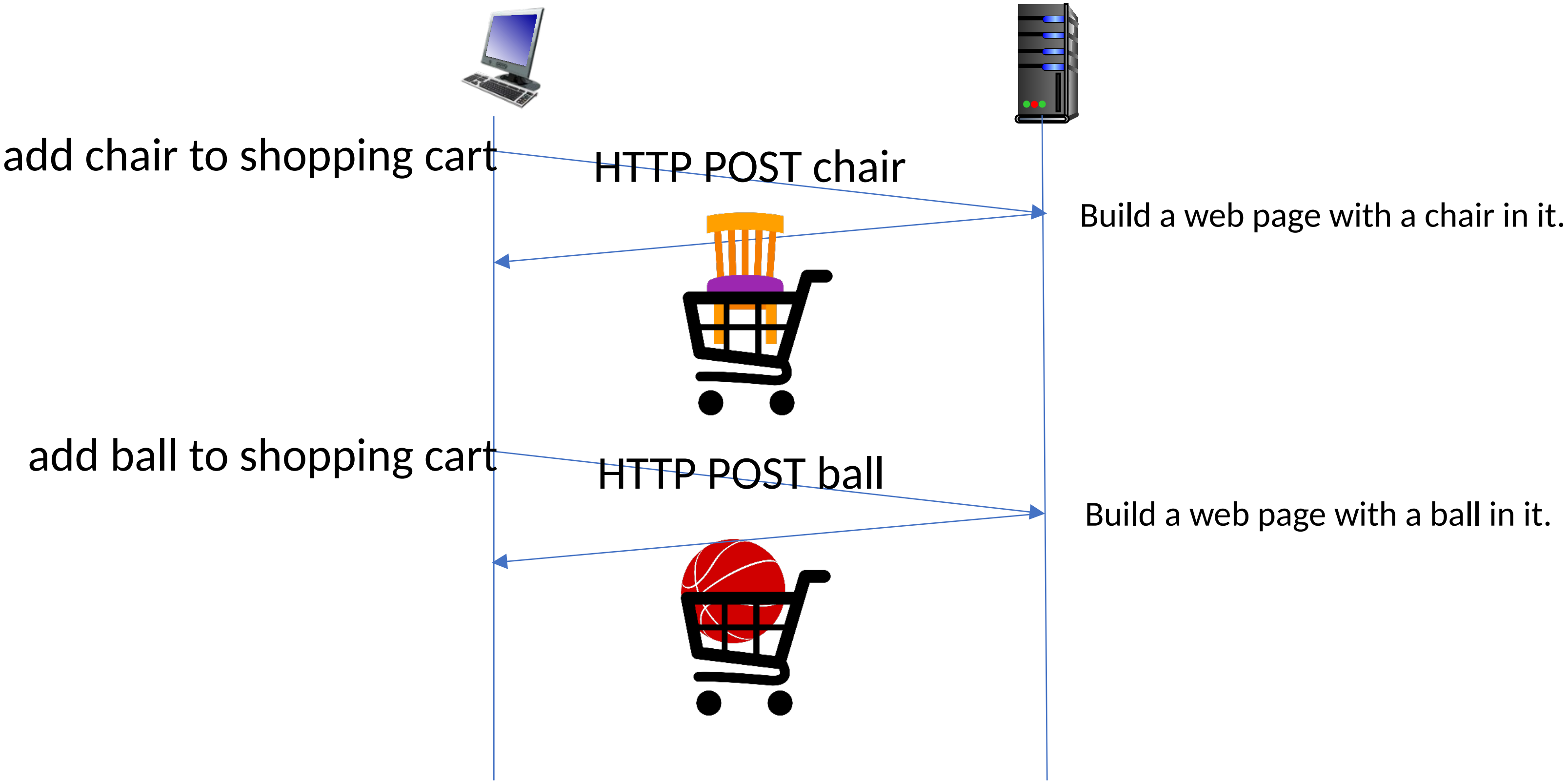
typed out request
(correct syntax is
Host: edition.cnn.com)

received response

HTTP Server is Stateless

- A stateless protocol does not require the server to retain information or status about each user for the duration of multiple requests.
- Successive requests from a given client to a server are not treated as a chain but rather as separate requests, independent from the previous ones.

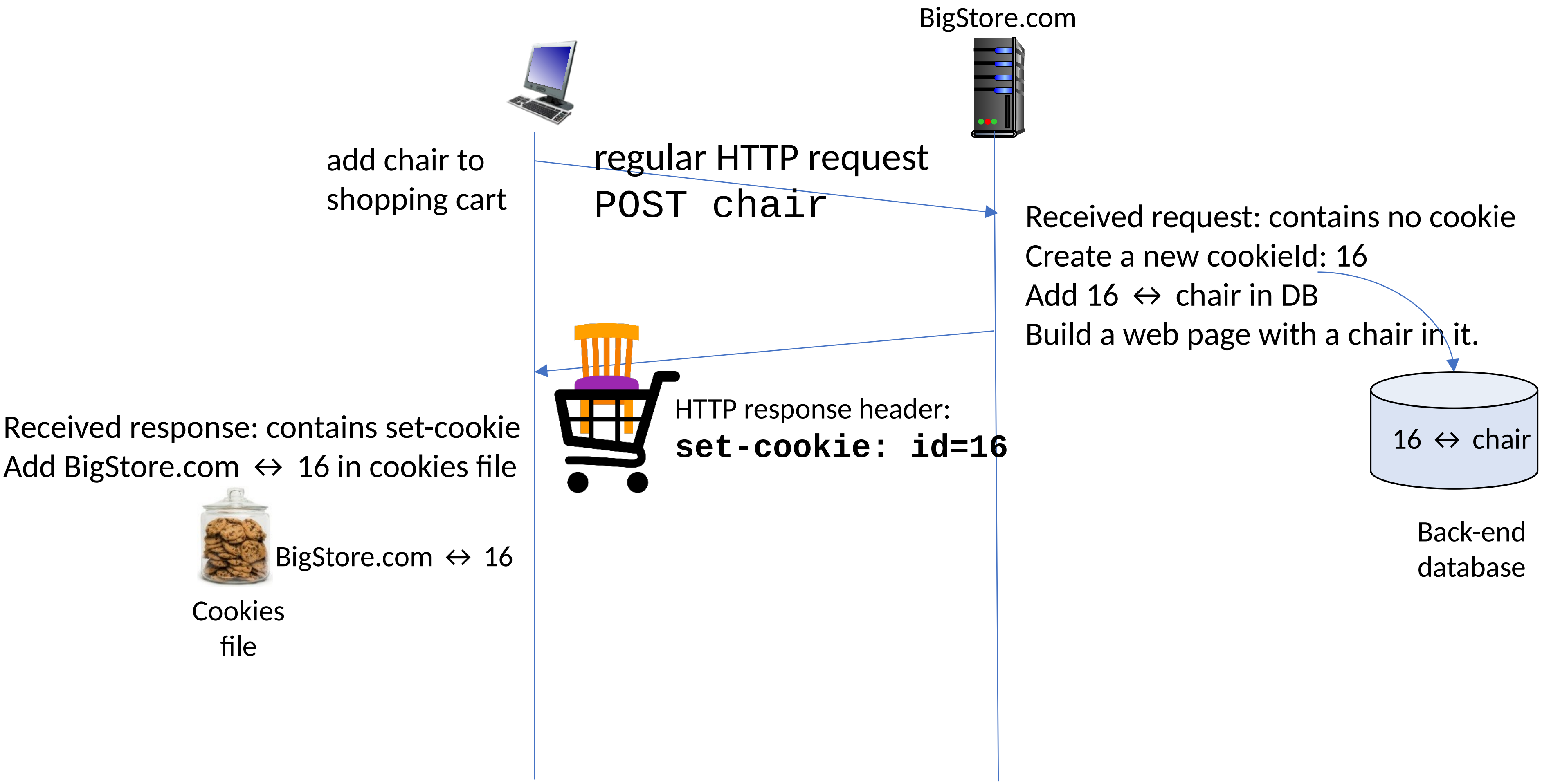
What we get is not what we want.


add chair to shopping cart
HTTP POST chair
Build a web page with a chair in it.
add ball to shopping cart
HTTP POST ball
Build a web page with a ball in it.

Cookies

Shhhhh… they're in here somewhere. Dad's always talking about the cookies in his computer.

Cookie Example



BigStore.com

add chair to shopping cart

regular HTTP request

`POST chair`

Received request: contains no cookie
Create a new cookieId: 16
Add 16 ↔ chair in DB
Build a web page with a chair in it.

HTTP response header:

`set-cookie: id=16`

16 ↔ chair

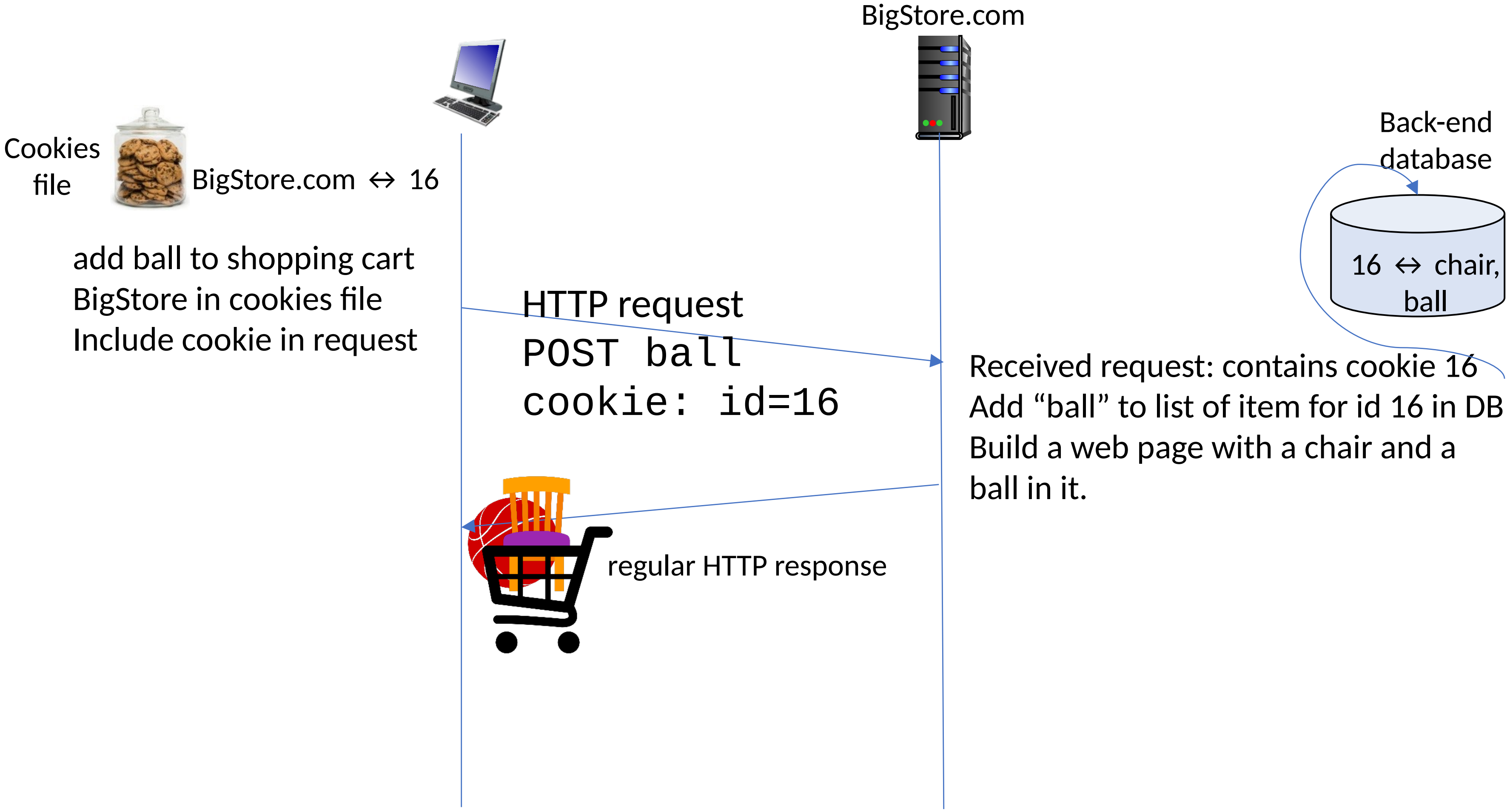
Back-end database

Received response: contains set-cookie
Add BigStore.com ↔ 16 in cookies file

BigStore.com ↔ 16

Cookies file

Cookie Example



BigStore.com

Cookies file

BigStore.com ↔ 16

add ball to shopping cart
BigStore in cookies file
Include cookie in request

HTTP request

```
POST ball
cookie: id=16
```

Back-end database

16 ↔ chair, ball

Received request: contains cookie 16
Add “ball” to list of item for id 16 in DB
Build a web page with a chair and a ball in it.

regular HTTP response

Uses

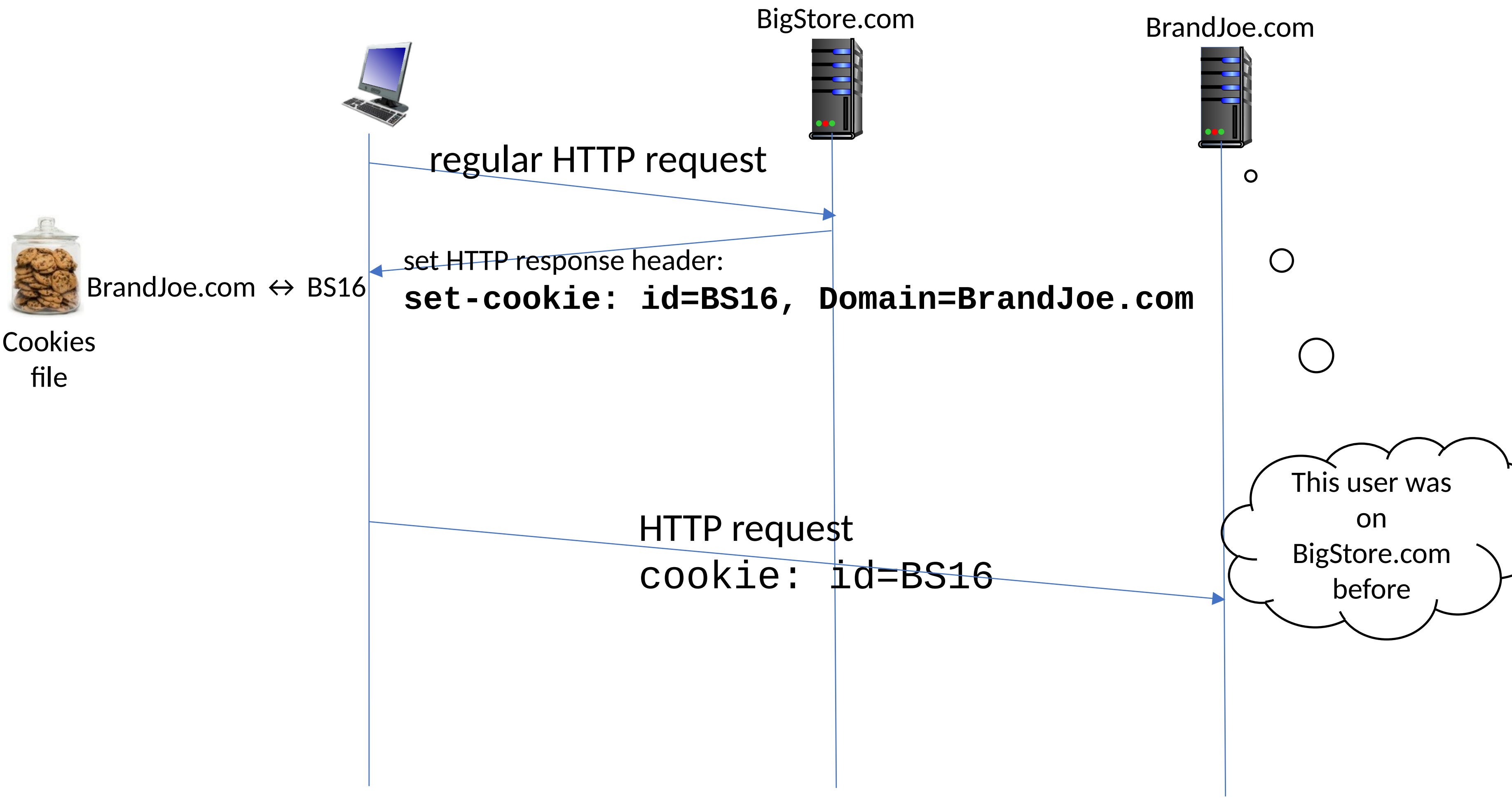
create a user session layer on top of stateless HTTP

- content adaptation (recommendation based on previous visits etc.).
- shopping carts (e-business)
- session definition at application layer (Web mail)
- authorization
- ...

Suspicions

- Invasion of privacy

Third-party advertising cookies



BigStore.com

BrandJoe.com

regular HTTP request

set HTTP response header:

set-cookie: id=BS16, Domain=BrandJoe.com

BrandJoe.com ↔ BS16

Cookies
file

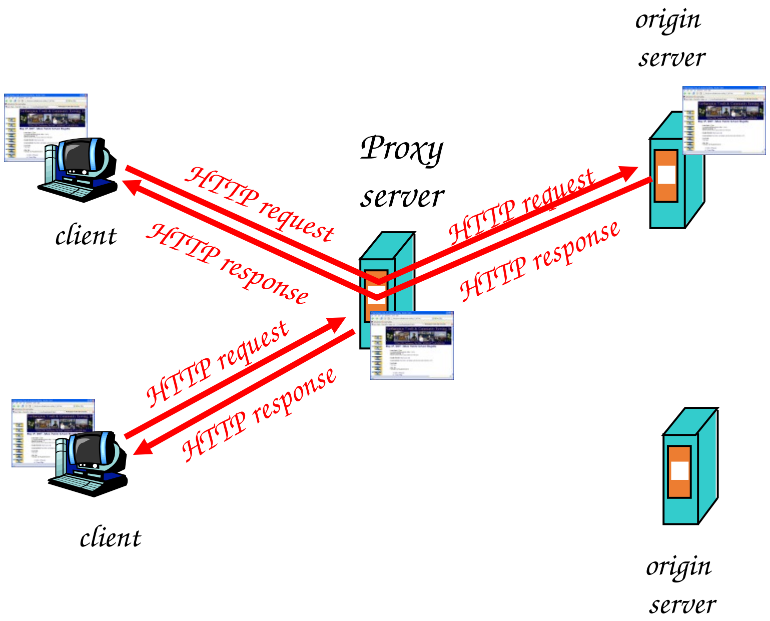
HTTP request

cookie: id=BS16

This user was
on
BigStore.com
before

Web Cache (proxy server)

- to satisfy the requests without involving the real server
- browser must be configured to send all HTTP requests to cache
- reduced traffic on Internet, improved response time



Caching Example

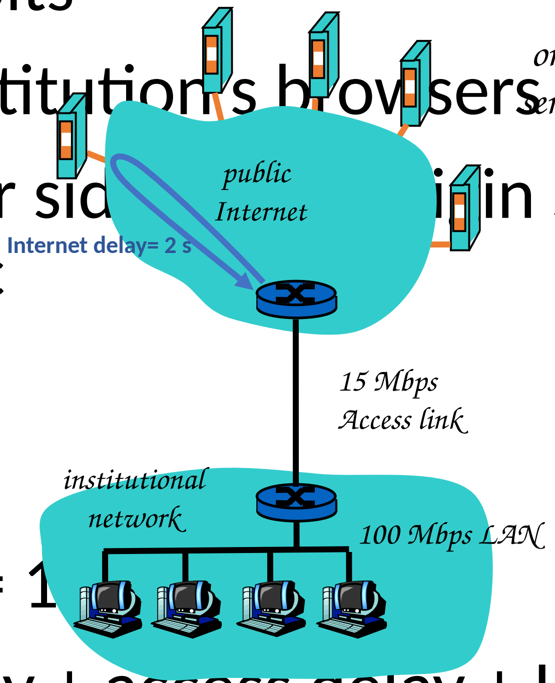
assumptions

- average object size = 1Mbits
- avg. request rate from institution's browsers = 15/sec
- delay from Internet router sid igin server and back to router = 2 sec

consequences

- utilization on LAN = 15%
- utilization on access link = 1
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching Example (cont)

possible solution

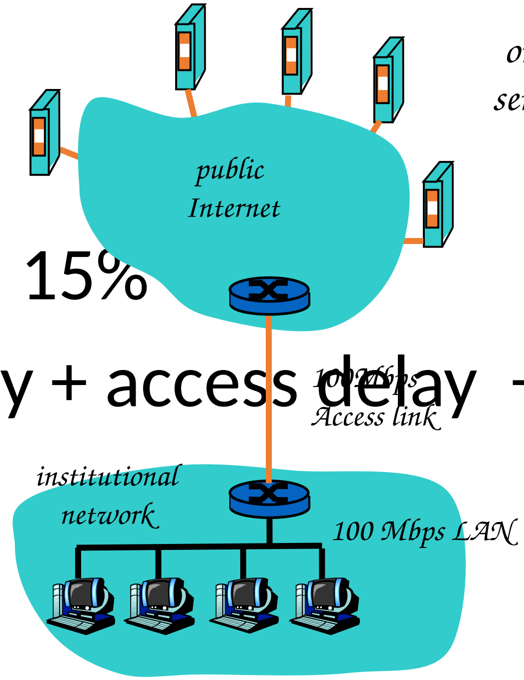
- increase bandwidth of access link to 100 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs

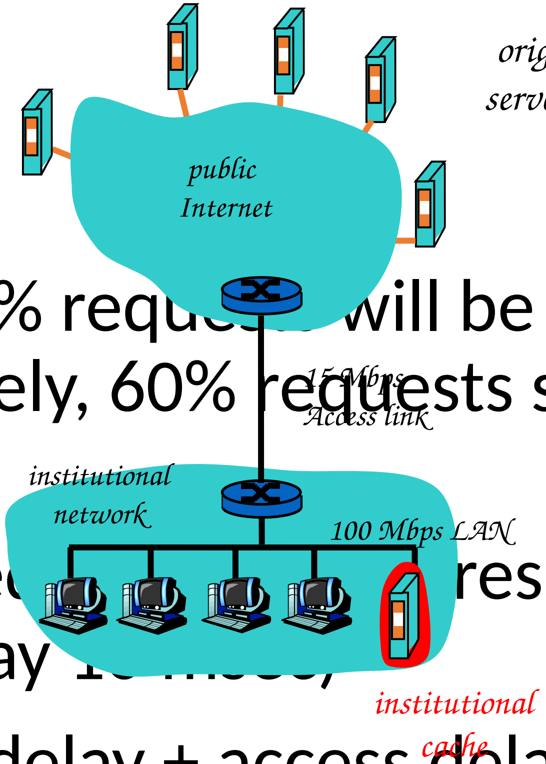
often a costly upgrade



Caching Example (cont)

possible solution:

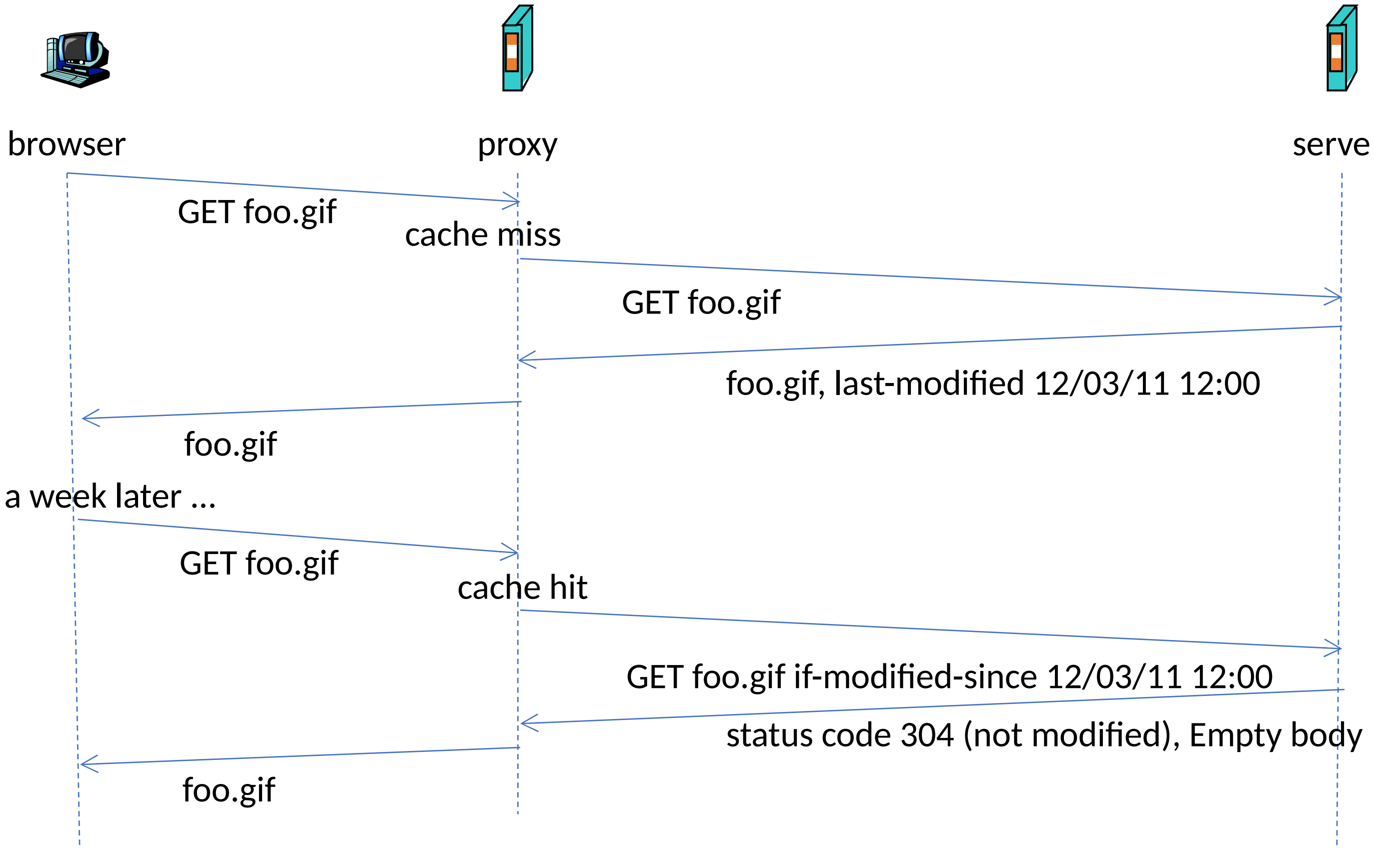
- install cache



consequence

- suppose hit rate is 0.4 (40% requests will be satisfied almost immediately, 60% requests satisfied by origin server)
- utilization of access link re[illegible] resulting and in negligible delays (say 10 msecs)
- avg total delay = Internet delay + access delay + LAN delay =0.6 * 2.01 secs * + 0.4 * 10 millisecs <1.3 secs

Conditional GET



browser

proxy

server

GET foo.gif

cache miss

GET foo.gif

foo.gif, last-modified 12/03/11 12:00

foo.gif

a week later …

GET foo.gif

cache hit

GET foo.gif if-modified-since 12/03/11 12:00

status code 304 (not modified), Empty body

foo.gif

Other Uses

- Allow multiple users to get a resource which access is limited to the proxy.
- Track and log web accesses.
- Deny access to a list of web sites.