# Computer Network I

## Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

# Application Layer

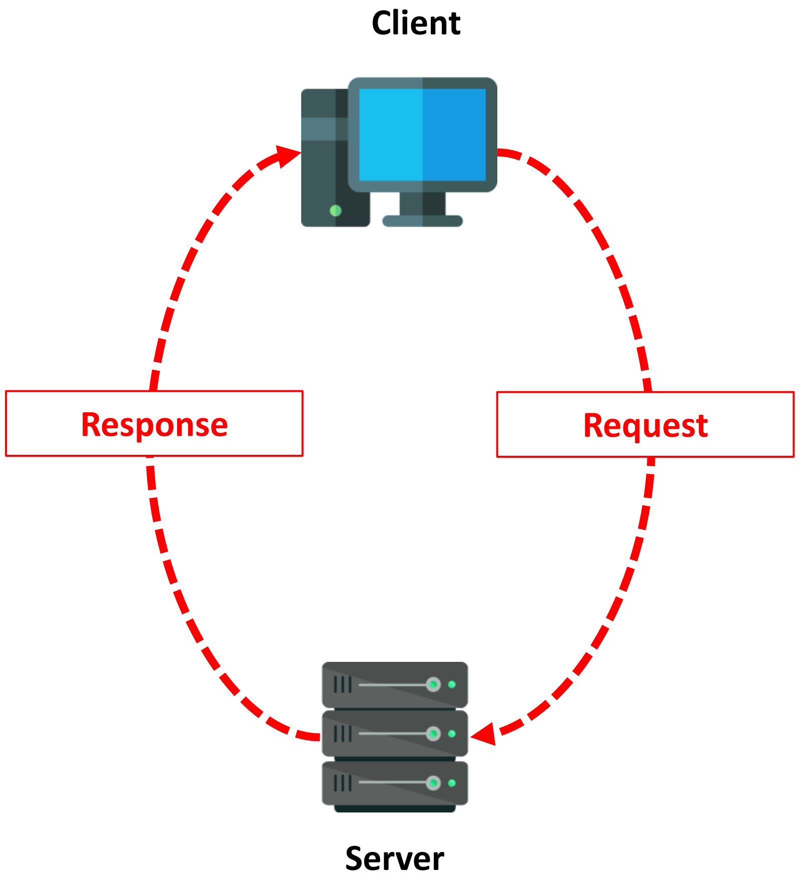
## HTTP: Message Format

- In HTTP protocol we have 2 different formats for the request and the response messages.

- **Request**: specifies the command (method) that the HTTP server has to perform (e.g., give me a web page, fill a specific form, etc.).

- **Response**: reports the outcome of the command (success, fail, etc.) and possible data (e.g., the requested file).

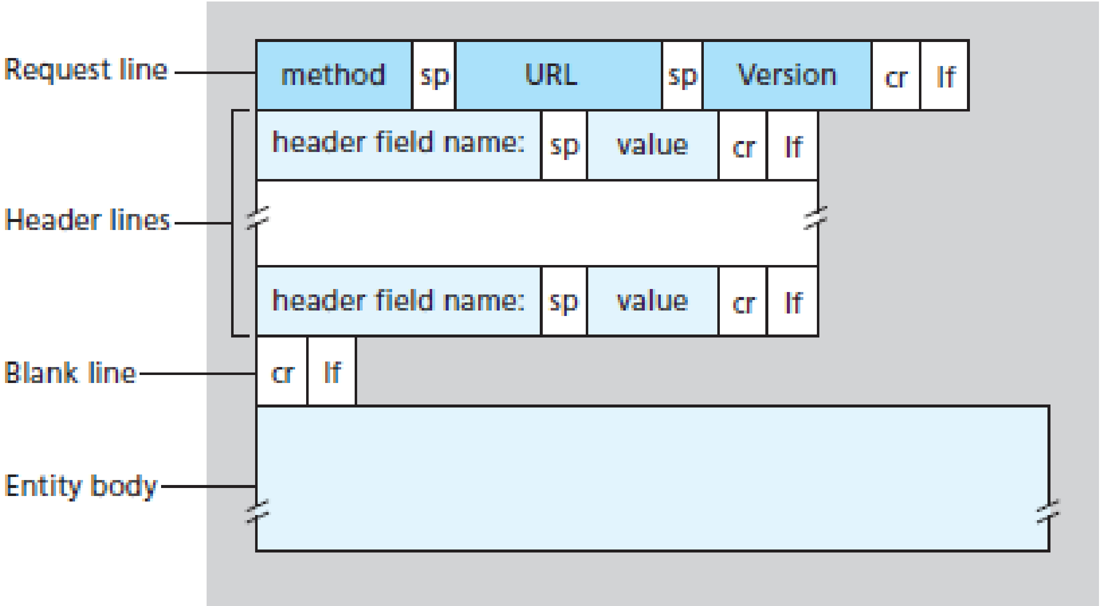
- Both messages are written in ordinary ASCII text, so they are easily readable by humans.



# Application Layer

## HTTP: Request Message Format

- The request message format include the following fields:
  - The **method**: specifies the requested command to be executed by the server.
  - The **URL**: is used to identify the object on which we want to operate.
  - The **version**: specifies the HTTP version (e.g., HTTP/1.1).
  - The **header lines**: contain the parameters of the request, the number and the type of these lines are not fixed. Each line include the **name** and the **value** of the parameter.
    - For instance, we can here specify if we want **persistent** or **non-persistent** connection.
    - Custom headers can also be used.
  - The **body**: is method-specific and contains data that are potentially associated with the command (e.g., text used to fill a form).



The fields are separated by special characters:

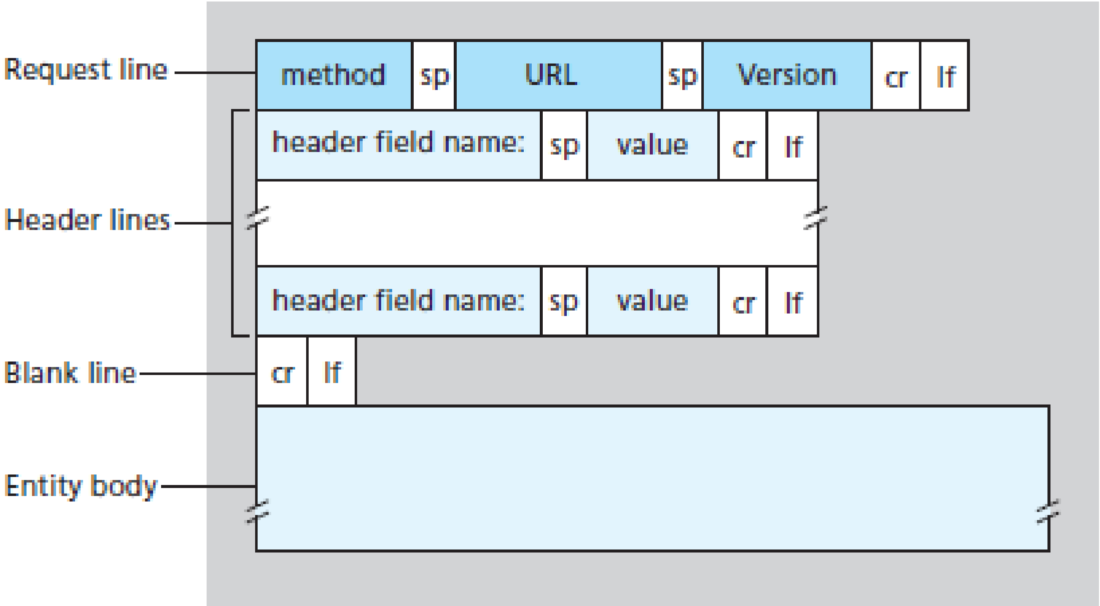
- The sp is space character.
- The cr is carriage return (\r).
- The lf is line feed (\n).

# Application Layer

## HTTP: Request Message Format (Methods)

- The **GET** method is used to retrieve objects (resources) form the server.
    - This is one of the most used commands as every time we browse a new webpage the associated file must be retrieved.
    - In the GET method the **body is empty**.

- The **POST** method is used to set info inside objects (resources) of the server.
    - A request generated with a form does not necessarily use the POST method, HTML forms often use the GET method and include the inputted data (in the form fields) in the requested URL.
    - The body contains the info to be posted.



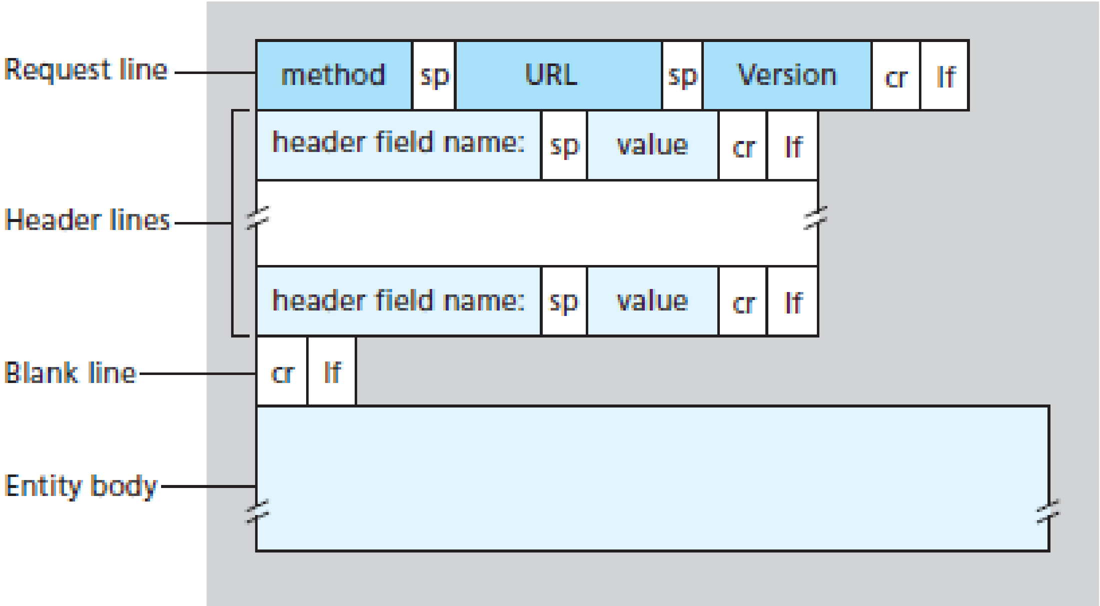
# Application Layer

## HTTP: Request Message Format (Methods)

- The **HEAD** method is similar to the GET method, when a server receives a request with the HEAD method, it responds with an HTTP message, but the object is not returned.
  - Application developers often use the HEAD method for debugging.

- The **PUT** method is often used in conjunction with Web publishing tools.
  - It allows a user to upload an object to a specific path (directory) on a specific Web server.
  - The PUT method is also used by applications that need to upload objects to Web servers.

- The **DELETE** method allows a user, or an application, to delete an object on a Web server.



# Application Layer

## HTTP: Request Message Format (Example)

- In this example we have a **browser** (Mozilla/5.0), implementing “HTTP/1.1”, that is **requesting** the object “/somedir/page.html” to the “www.someschool.edu” web server.

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

- We have **five lines**:
  - The **GET request line** (resource and version).
  - The **four header lines** (host, connection, user-agent, accept-language)…

# Application Layer

## HTTP: Request Message Format (Example)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

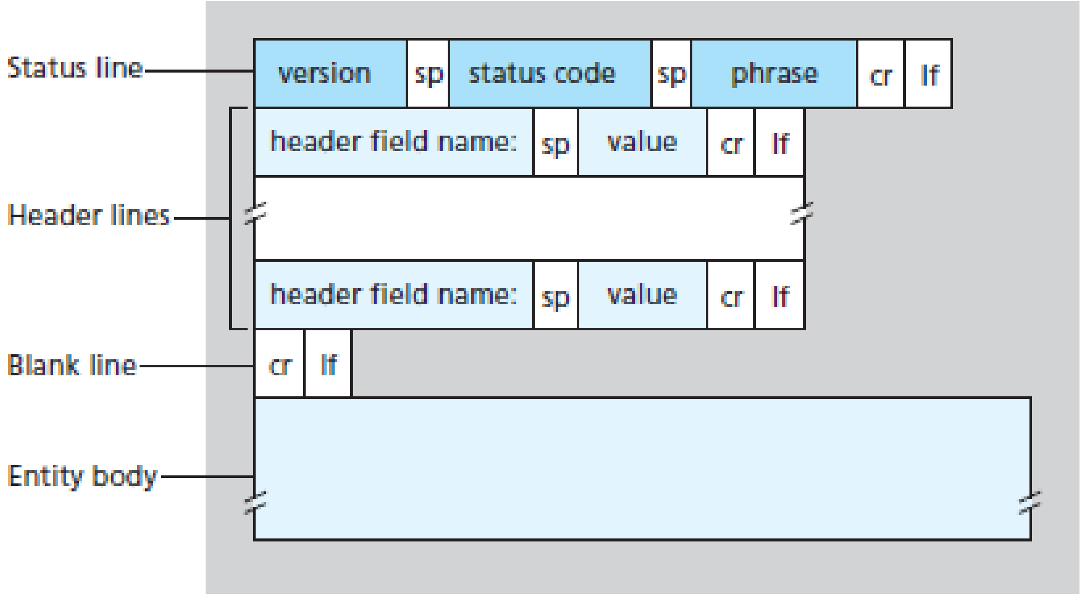
- The line **“Host: www.someschool.edu”** specifies the **host on which the object resides**.
  - **Destination is not always the next host**: a message could be **forwarded by another host** (e.g., proxy server), so the address in the TCP is different from the real target host.
- The line **“Connection: close”** specifies that the browser is asking for a non-persistent connection (i.e., close after finish).
- The line **“User-agent: Mozilla/5.0”** specifies the **browser’s type**.
  - This is sometimes useful because the **server can send different versions** of the same object (each version is addressed by the same URL) **depending on the type**.
- The line **“Accept-language: fr”** indicates that the **user prefers to receive a French version** of the object (if exists).

# Application Layer

## HTTP: Response Message Format

- The general format of the response message is similar to the request one.

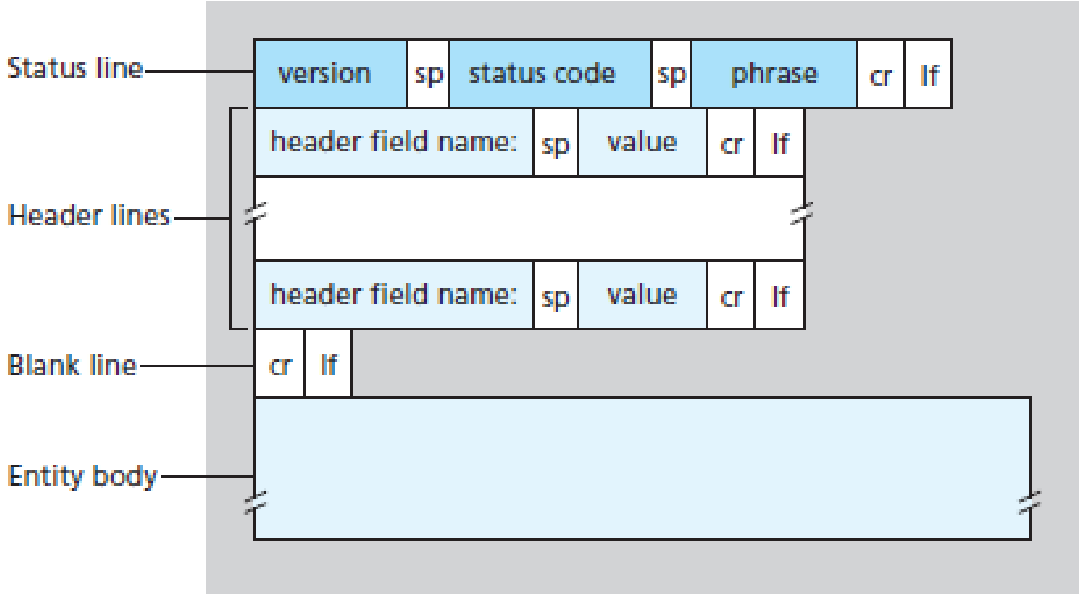
- In this case, instead of a request line, we have a **status line** that reports the outcome of the command that includes:
  - The **version**: reports the HTTP version of the server's response.
  - The **status code**: a code (number) that specifies the outcome of the command.
  - The **phrase**: contains the result of the request.



# Application Layer

## HTTP: Response Message Format (Status Codes)

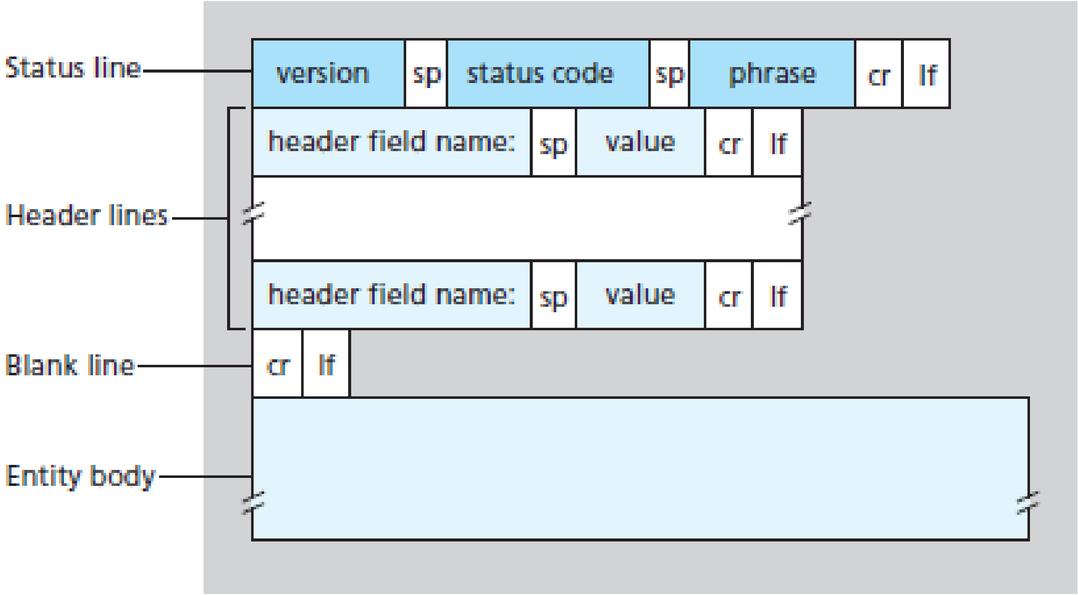
- **Status codes** are divided into classes:
  - 100-199 **informational**: info regarding the request.
  - 200-299 **successful**: the request has been executed successfully.
  - 300-399 **redirection**: there are additional actions needed from the client.
  - 400-499 **client-error**: the request cannot be executed because of a client issue.
  - 500-599 **server-error**: the request cannot be executed because of a server issue.



# Application Layer

## HTTP: Response Message Format (Status Codes)

- Some typical examples are:
  - **200 OK**: request succeeded, and the information is returned in the response.
  - **301 Moved Permanently**: requested object has been permanently moved; the new URL is specified in the “Location” header of the response message.
  - **400 Bad Request**: generic error code indicating that the request could not be understood by the server.
  - **404 Not Found**: The requested document does not exist on this server.
  - **505 HTTP Version Not Supported**: The requested HTTP protocol version is not supported by the server.



# Application Layer

## HTTP: Response Message Format (Example)

- In this example, the status line indicates that **the server is using HTTP/1.1 and that everything is OK** (the server has found and is sending the requested object).
- There are **five header lines** (connection, date, server, last-modified, content-length, content-type).
- The **body contains the requested object** (represented by data data data data data ...).

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

 (data data data data data ...)
```

# Application Layer

## HTTP: Response Message Format (Example)

- The line **“Connection: close”** informs the client that the **connection will be closed** after this message (non-persistent).
- The line **“Date: …”** indicates the time and **date when the HTTP response was created** and sent by the server.
- The line **“Server: …”** indicates that the message was **generated by an Apache Web server** (similar to user-agent).
- The line **“Last-Modified: …”** indicates the time and date when the **object was created or last modified**.
    - Useful in case of caching, as cached files can be outdated.
- The line **“Content-Length: …”** indicates the **number of bytes in the object**.
- The line **“Content-Type: …”** indicates that the **object is HTML text** (here, the object type is officially indicated by this header line and not by the file extension).

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

 (data data data data data ...)
```

# Application Layer

## HTTP: Cookies

- An HTTP **server is typically stateless**, this simplifies server design reduces the use of resources and allow servers to handle thousands of simultaneous TCP connections.

- Pure statelessness is also a strong limitation as **several web functions are client-specific** (for example, Amazon’s cart depends on the client, Netflix suggests contents based on client’s preferences, etc.).

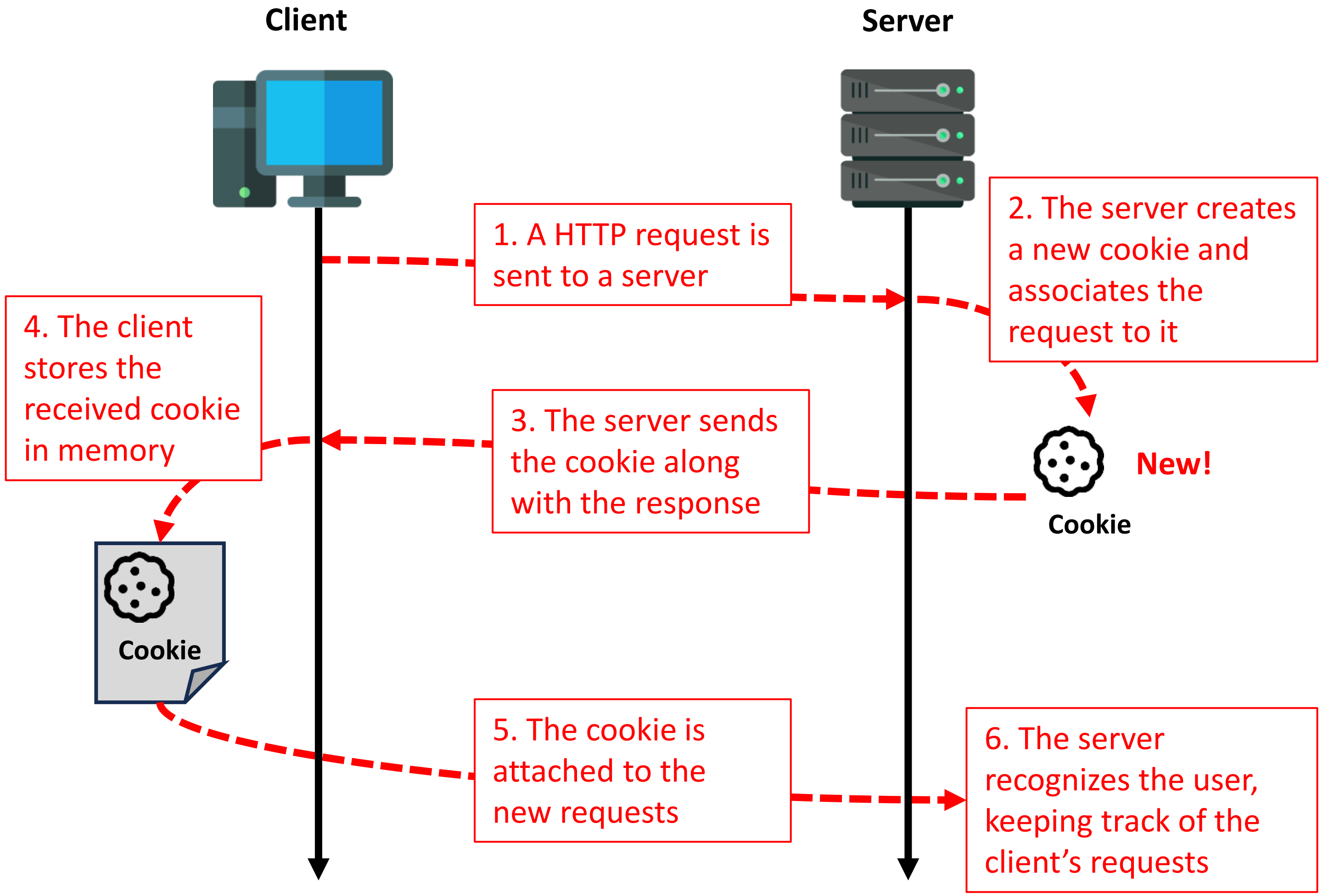
- For these purposes, HTTP uses cookies. A **cookie** is a digital token (alphanumeric ID) used by servers to identify a specific client.
  - Cookies allow web sites to keep track of users, most of the major commercial Web sites use cookies.
  - Cookies may also have attributes (e.g., expiration date).

# Application Layer

## HTTP: Cookies

- A **cookie is created by the server** and delivered to the client.

- Cookie technology involves four main components:
  1. A **“Set-cookie” header** line in the HTTP response message.
  2. A **“Cookie” header** line in the HTTP request message.
  3. A **file** on the client system (managed by the user’s browser).
  4. A **back-end database** on the server.



1. A HTTP request is sent to a server

2. The server creates a new cookie and associates the request to it

3. The server sends the cookie along with the response

4. The client stores the received cookie in memory

5. The cookie is attached to the new requests

6. The server recognizes the user, keeping track of the client’s requests

# Application Layer

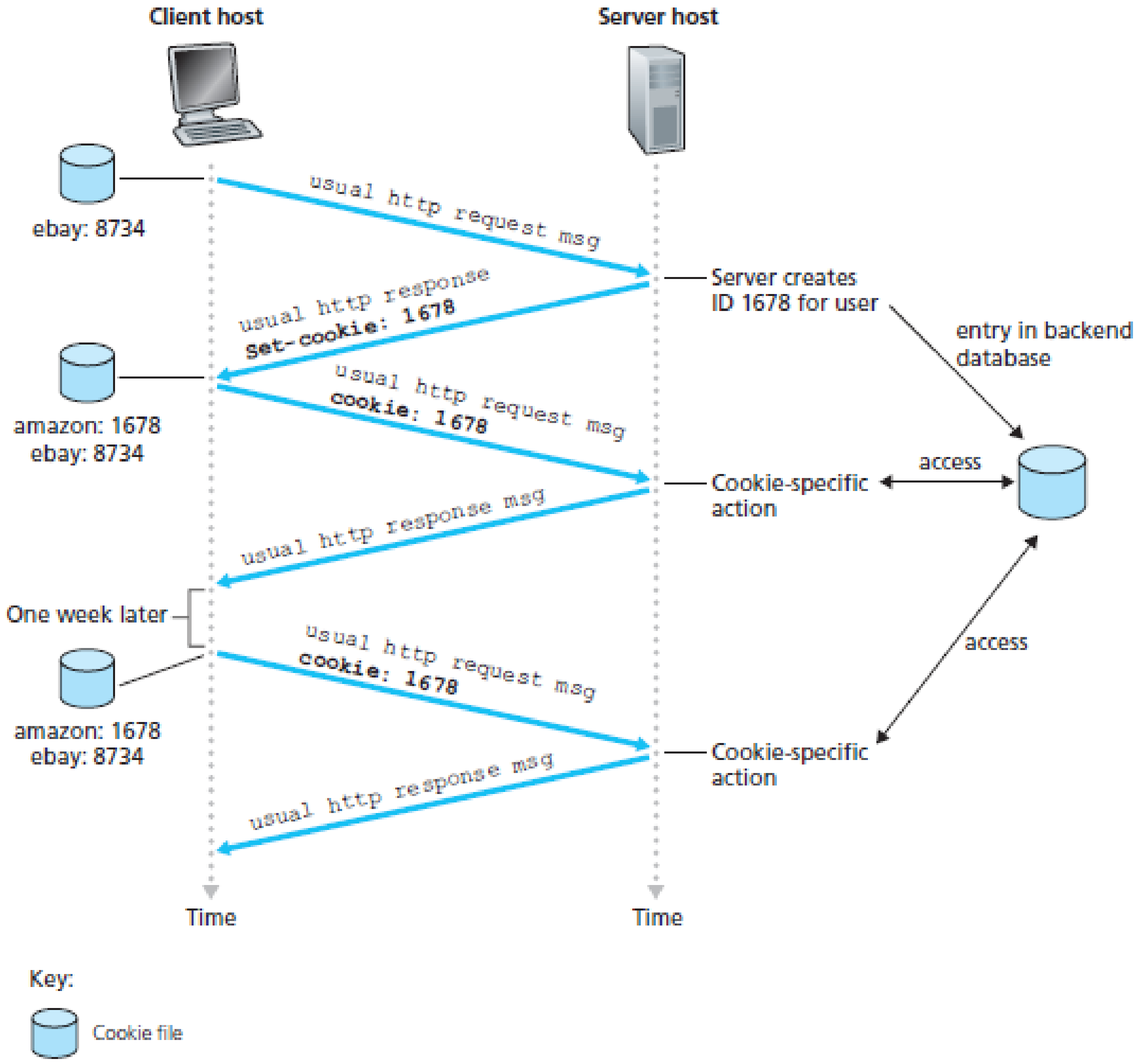
## HTTP: Cookies (Example)

- Let’s assume a client host, which uses eBay regularly, to **contact Amazon.com for the first time**.

- The client already has a cookie for eBay, but **it has no cookie for Amazon**.

- When the request comes into the Amazon Web server, **the server creates a cookie** (ID number) and an associated entry in its back-end database.

- The Amazon Web server then responds to client’s browser and **includes in the HTTP response a “Set-cookie” header** line, which contains the ID (`Set-cookie: 1678`).



# Application Layer

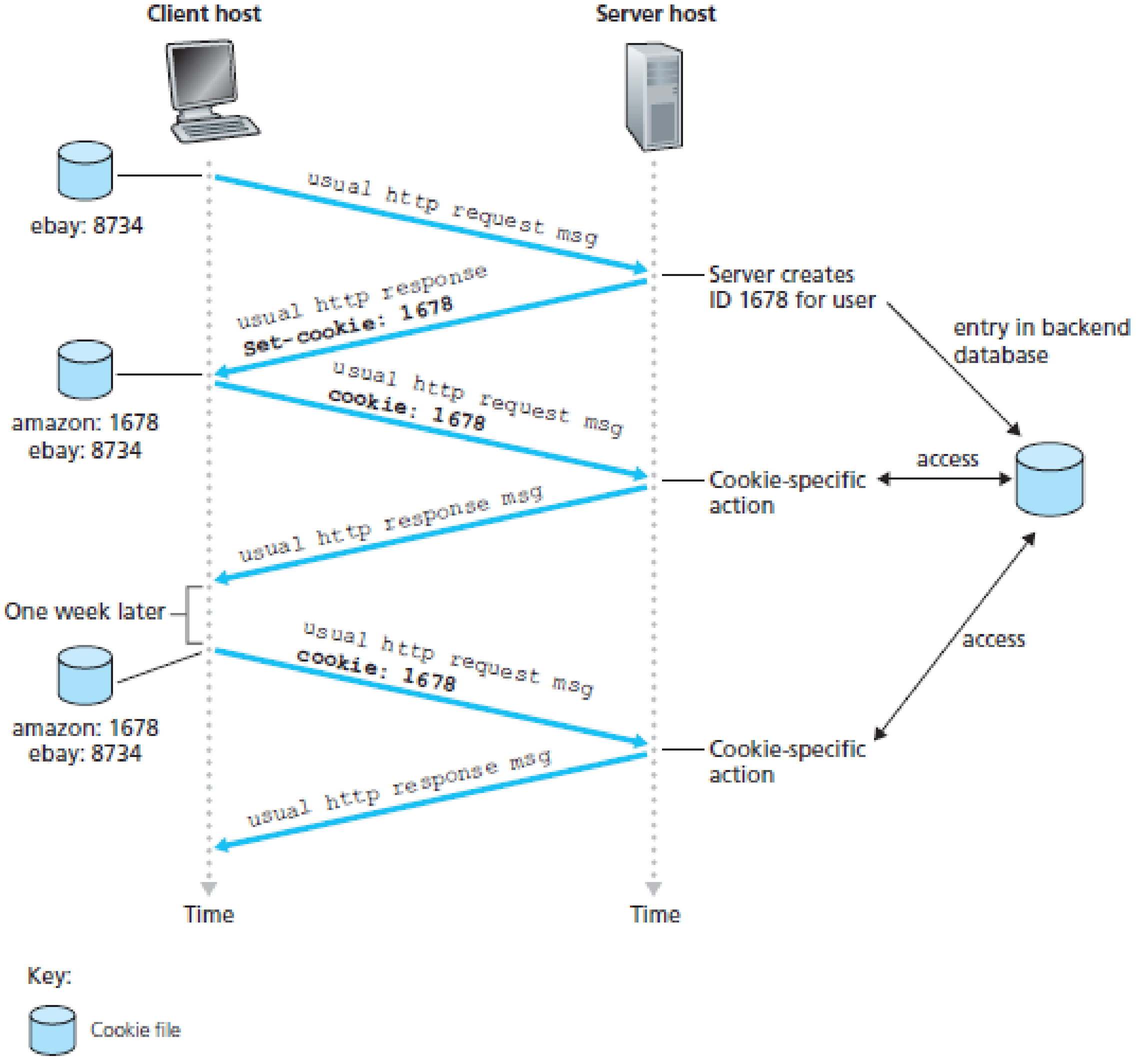
## HTTP: Cookies (Example)

- When the response is received, **the browser appends a line to a special cookie file** including:
  - the hostname of the server.
  - the ID number of the cookie.

- From now on, the requests from the client to Amazon **will be associated to the new cookie** by including the header line `Cookie: 1678` to the HTTP messages.

- Amazon server is then able to **track the client's activity through the database**. It knows exactly which pages user 1678 visited, in which order, and at what times.

- If the client returns to Amazon's site **one week later**, the browser will continue to put the header line Cookie: 1678 in the request messages.



# Application Layer

## HTTP: Cookies

- Amazon (and other web sites) uses cookies to provide different services:
  - **Shopping cart service**, the server can maintain a list of intended purchases during browsing.
  - **Products recommendations**, based on the visited web pages.
  - **User's registration**, by associating user's info to the cookie in the database (credit-card, name, e-mail, address) so you don't have to re-insert them every time.
- Although cookies often simplify the Internet shopping experience for the user, they are **controversial** because they can also be considered as an invasion of privacy.
- Using a combination of **cookies** and user-supplied **account information**, a Web site can learn a lot about a user and potentially sell this information to a third party.

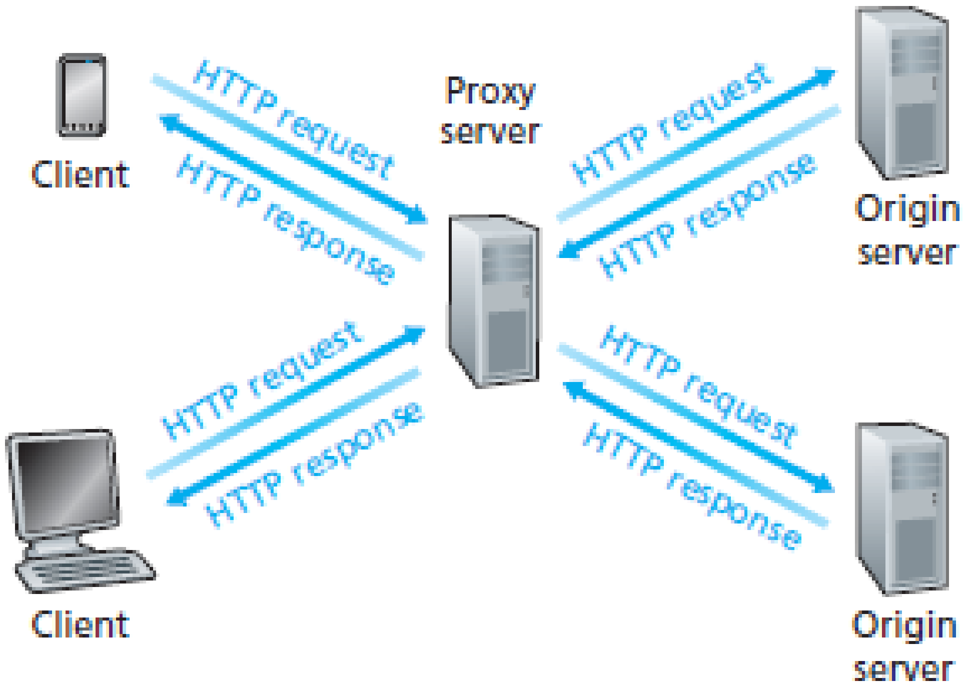
# Application Layer

## HTTP: Web Caching

- A **Web cache** (also called a **proxy** server) is a network entity that satisfies HTTP requests on the behalf of an origin Web server.

- The Web cache has its own disk storage and **keeps copies of recently requested objects** in this storage.

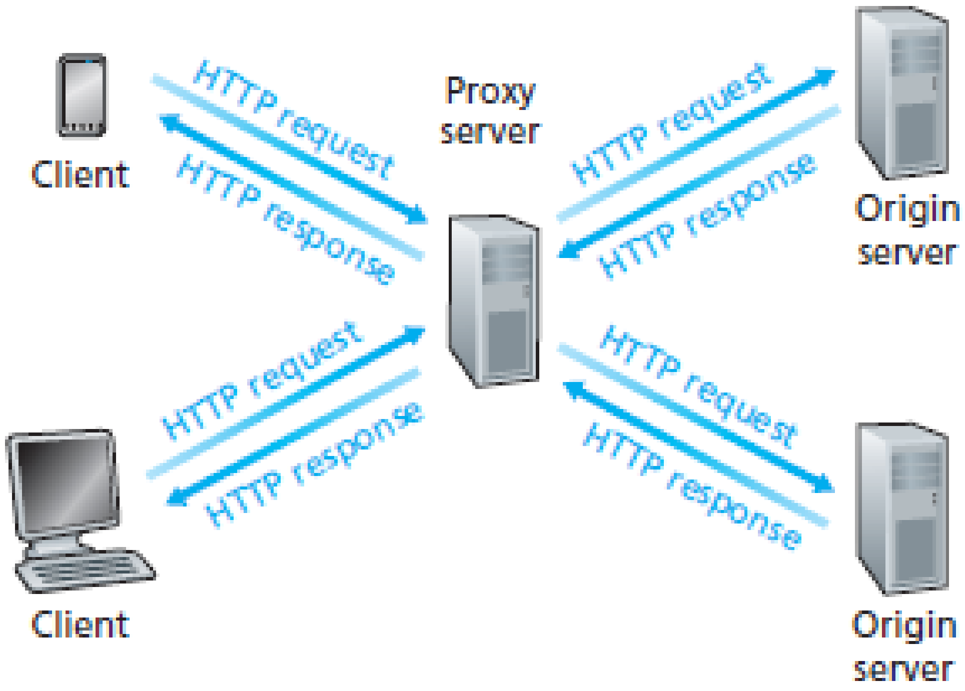
- A browser can be configured so that all the HTTP **requests are first directed to the Web cache** to check if a copy of the requested object is available.



# Application Layer

## HTTP: Web Caching

- Let’s assume a browser to request the object http://www.someschool.edu/campus.gif passing through a web cache:
  1. The **browser establishes a TCP connection** to the Web cache and sends an HTTP request for the object to the Web cache.
  2. The Web cache **checks if it has a copy of the object** stored locally. If so, the Web cache returns the object within an HTTP response message to the client browser.
  3. If the **Web cache does not have the object**, the Web cache opens a TCP connection to the origin server (www.someschool.edu) and sends an HTTP request for the object.
  4. When the **Web cache receives the object, it stores a copy in its local storage** and sends a copy, within an HTTP response message, to the client browser.

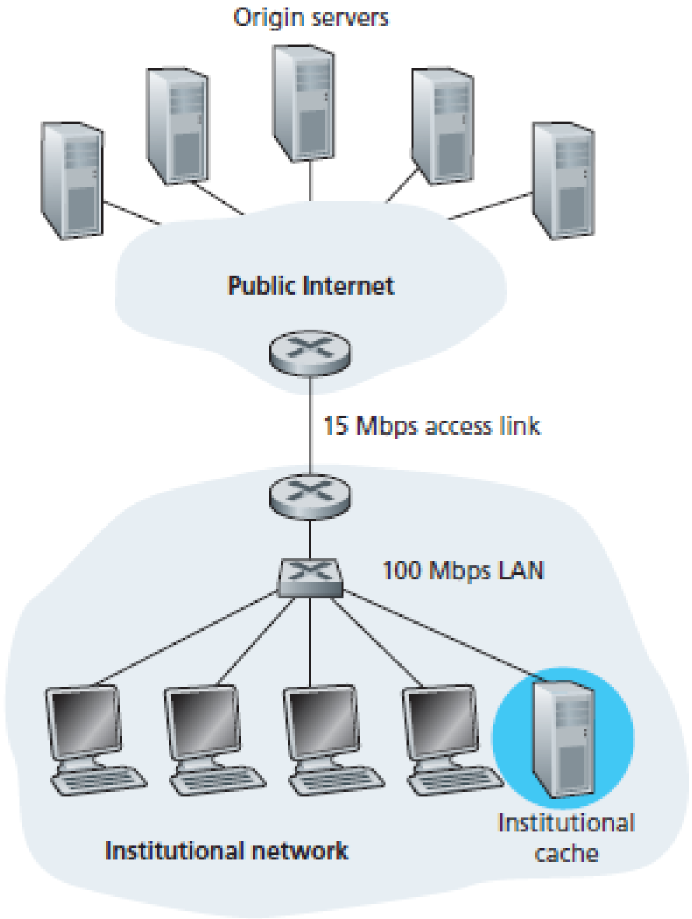


# Application Layer

## HTTP: Web Caching

- Note that a cache is both a server (when providing objects) and a client (when requesting objects) at the same time.

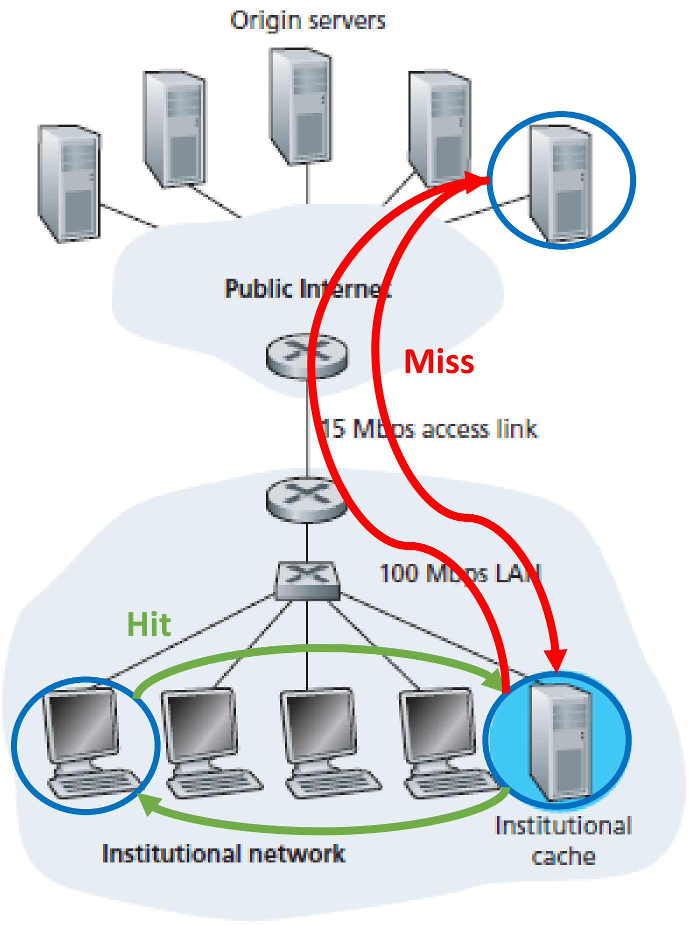
- Web caching has seen deployment in Internet for two reasons:
  1. A Web cache can substantially **reduce the response time for a client request**, particularly if a high-speed connection stands between the client and the cache.
  2. Web caches can substantially **reduce traffic** of a company or institution toward Internet, in so reducing costs due to bandwidth.



# Application Layer

## HTTP: Web Caching

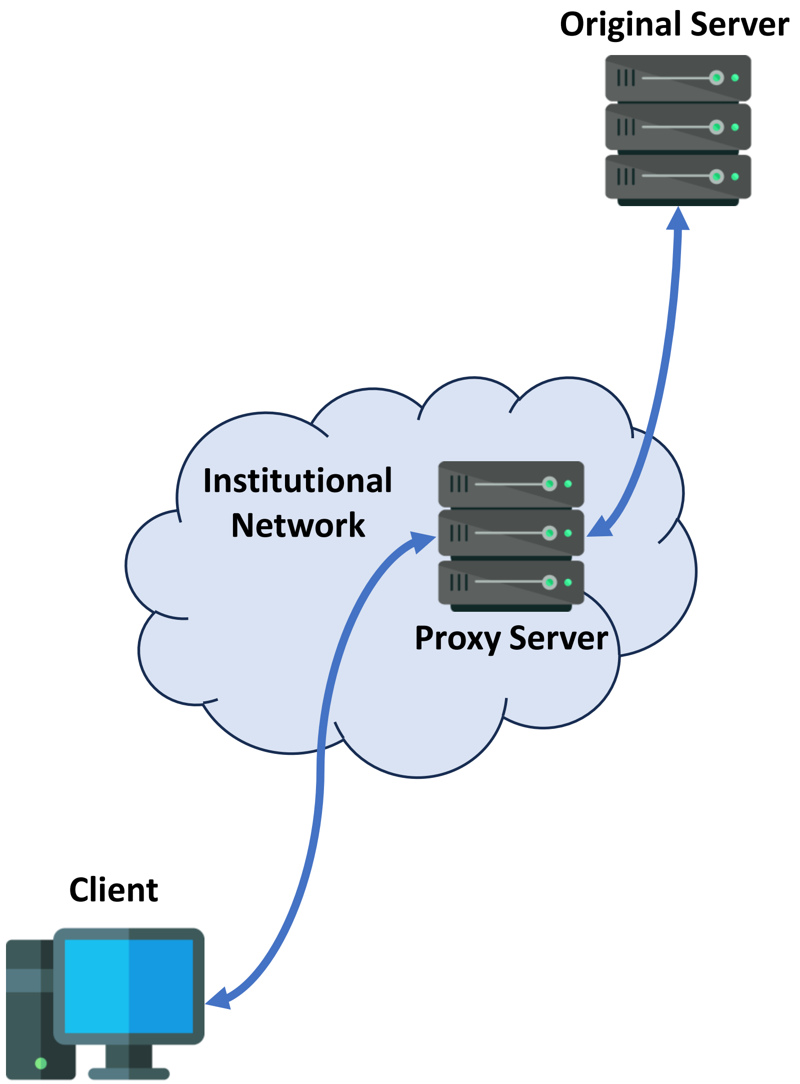
- Web caches are **often installed into a company/institution local network** to speed-up and reduce traffic.
- A **hit** happens when a cache successfully provide an object without contacting the original server.
- The hit rate, i.e., the fraction of requests that are satisfied by a cache, typically **ranges from 0.2 to 0.7**.
  - Hit rate increases when more clients use the cache.
- This means that up to 70% of requests can be served locally.



# Application Layer

## HTTP: Web Caching

- The UNINA network also provides an institutional proxy (proxy.unina.it).
- Besides performance, **proxy servers can also be used to access to institutional services**.
- As requests are forwarded by the institutional proxy, from the original server's standpoint all **requests are coming from the institution**.
- There are also several commercial/free **proxy servers available on Internet**.



# Application Layer

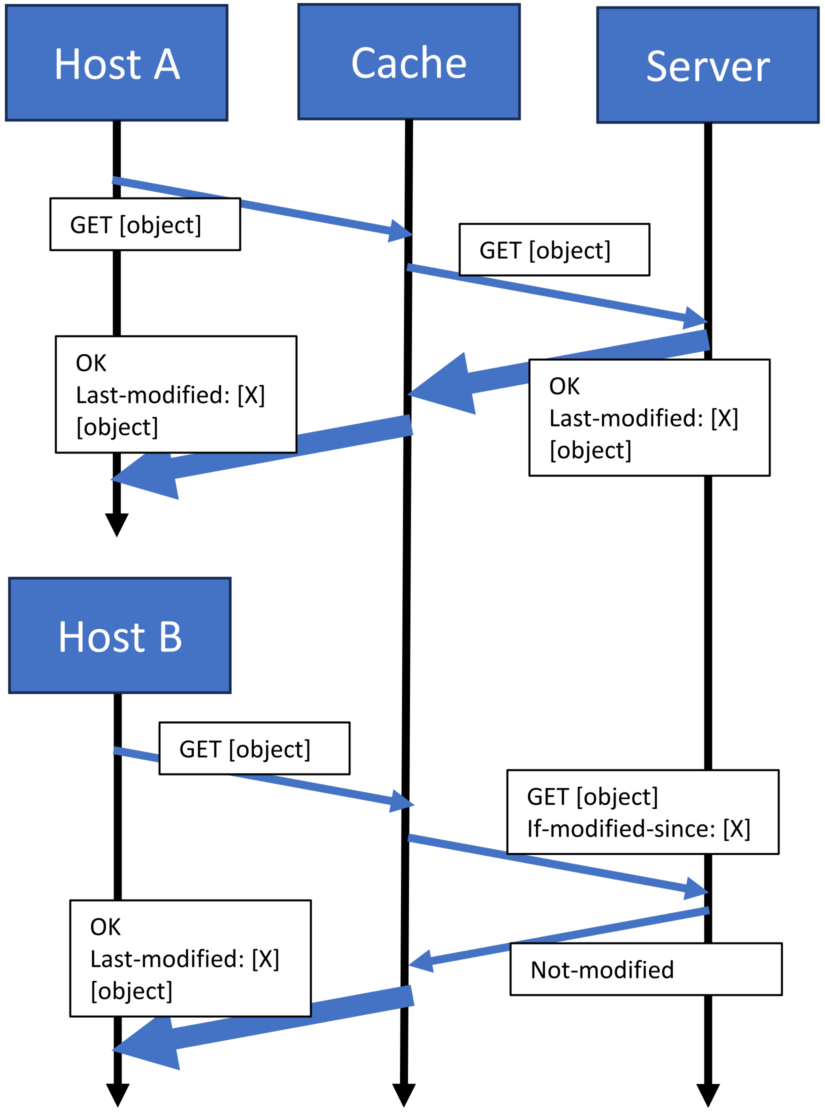
## HTTP: Web Caching

- Web caching introduces a new problem: the copy of an object residing in the cache **may be outdated**.

- To avoid this issue, HTTP has a mechanism that allows a cache to **verify the stored object**.

- The **conditional GET** is an HTTP request message including a GET method and “If-Modified-Since:” header.

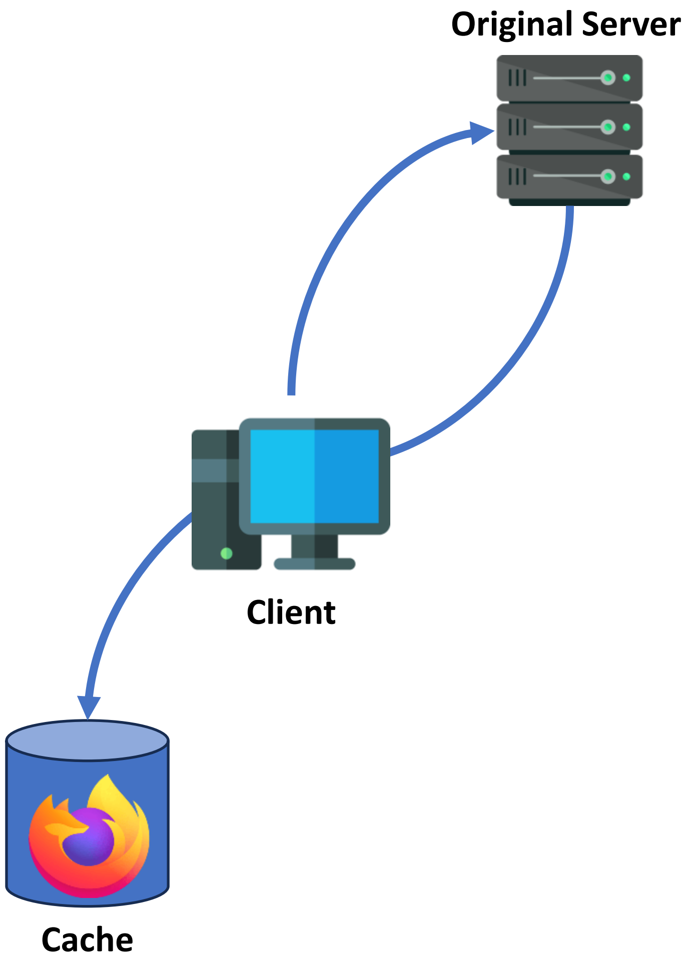
- The cache **checks if the object is up to date** and, if so, the stored version is sent back to the host (no further communication needed).



# Application Layer

## HTTP: Web Caching

- Caching **is also performed locally** by browsers.
- The principle is the same as servers, the browser stores objects locally so they no need to be retrieved from the server.
- This is a common techniques in modern browsers as **it drastically improves performance**.
- The local version of **the object may be not updated**, generating errors (quite frequent).


Original Server
Client
Cache