Lecture 3.

HTTP v1.0
application layer protocol

into details

HTTP 1.0: RFC 1945, T. Berners-Lee, R. Fielding, H. Frystyk, may 1996
HTTP 1.1: RFC 2068, 2616

G.Bianchi, G.Neglia, V.Mancuso

Università degli Studi di Palermo

Generalities

Ascii protocol

- ⇨ uses plain text

➔ case sensitive

- ⇨ GET is legal
- ⇨ get is not…

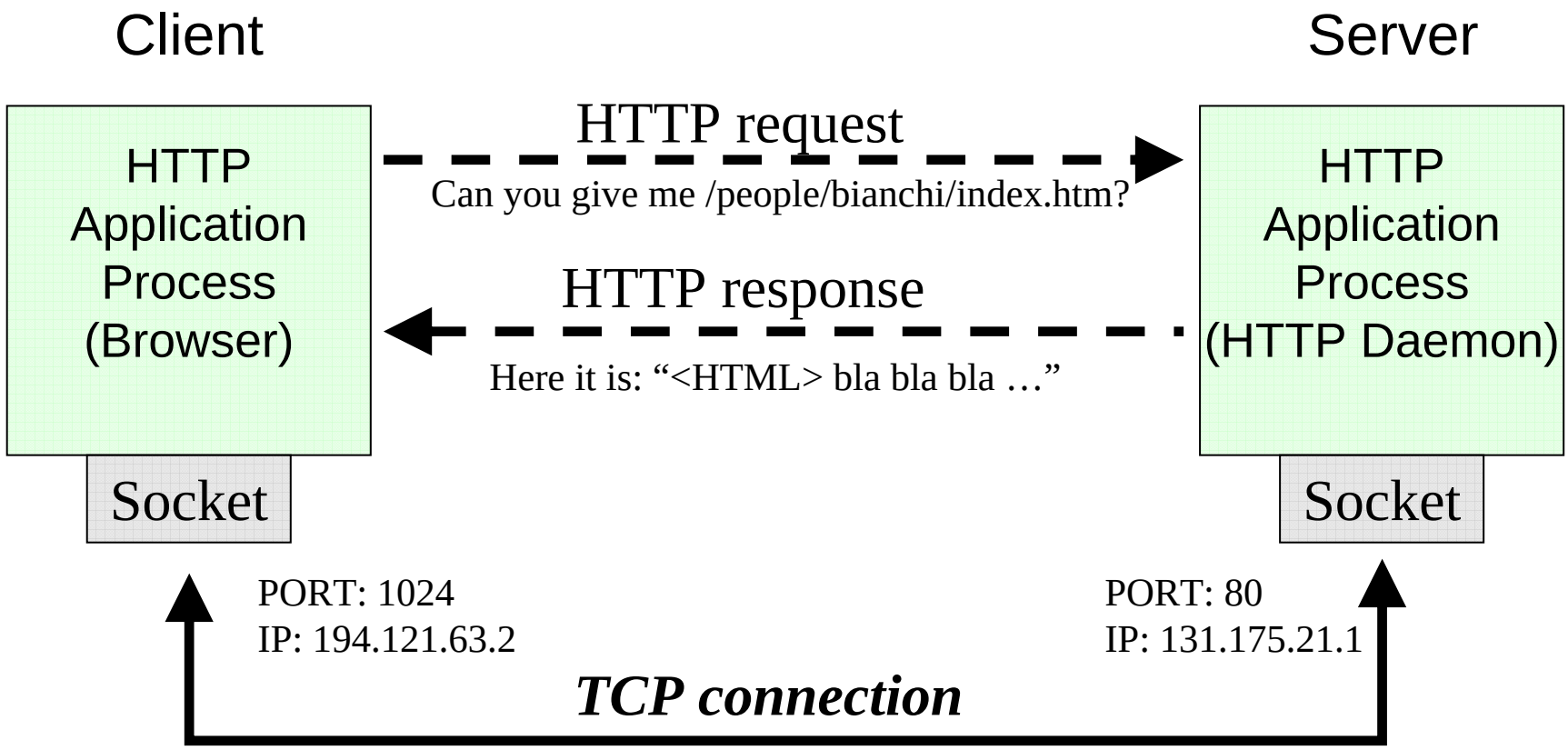
➔ Messages and delivery order:

- ⇨ First: HTTP request
- ⇨ Follows: HTTP response

➔ Messages + entity bodies:

- ⇨ structured sequence of octets
- ⇨ Any content (web pages, images, resources, etc)
- ⇨ transmitted on TCP
  - → But TCP not mandatory: any reliable transport connection is ok

Request/Response



Of course HTTP ignores IP & PORT: These info belong to lower layers, and have already been used to address the web server and enable connection!

Request/Response syntax

➔ **Request-Line (mandatory)**

GET /docs/pippo.html HTTP/1.0

⇨ *Full "absolute" path required*

⇨ *Protocol version required*

➔ **Status-Line (mandatory)**

HTTP/1.0 200 OK

⇨ *Protocol version, status code, and reason phrase*

➔ **Headers (optional, one or more, any order)**

⇨ *general header*

→*General information (es: date, no-cache)*

⇨ *Request header*

→allows client to optionally pass additional information about the request, and about the client itself that could not be stored in the request line

⇨ *Response header*

→allows server to optionally pass additional information about the response, and about the server itself that could not be stored in the status line

⇨ *entity header (information about entity eventually transferred)*

➔ **null line**

➔ **entity body (one or more, separated by null lines)**

Examples

Request:

```
GET /test/index.html?foo=bar+baz&name=steve HTTP/1.0\r\n
Connection: Keep-Alive\r\n
User-Agent: Mozilla/4.07 [en] (X11; I; Linux 2.0.36 i686)\r\n
Host: ninja.cs.berkeley.edu:5556\r\n
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*\r\n
Accept-Encoding: gzip\r\n
Accept-Language: en\r\n
Accept-Charset: iso-8859-1,*,utf-8\r\n
\r\n
xxxxxxxxxxxxxxxxxxxxx
```

Response:

```
HTTP/1.0 200 OK
Server: Netscape-Enterprise/2.01
Date: Thu, 04 Feb 1999 00:28:19 GMT
Accept-ranges: bytes
Last-modified: Wed, 01 Jul 1998 17:07:38 GMT
Content-length: 1848
Content-type: text/html
\r\n
xxxxxxxxxxxxxxxxxxxxxx
```

HTTP methods

➔ **GET: retrieve a page**

⇨ GET+If-Modified-Since to refresh cache entities

➔ **HEAD: identical to GET, but with no body retrieve**

⇨ full header information retrieved, though

⇨ Usage: testing hyperlinks validity.

➔ **POST: append information to selected URL.**

⇨ used to send user data (collected through forms) ...

⇨ ...to a data-accepting process (or gateway to some other protocol).

In addition (not really used: big security issues if not careful):

➔ **PUT: overwrites a page with new content**

➔ **DELETE: removes a page**

➔ **LINK, UNLINK (never used: not included in HTTP/1.1)**

Status codes

- **2xx: success**
  - action successfully received,understood, and accepted
    - 200=OK, 204=no content, 201=created, 202=accepted, …
- **3xx: redirection**
  - further action must be taken to complete the request
    - 301=moved permanently, 302=moved temporarily, 304=not modified
- **4xx: client Error**
  - request contains bad syntax or cannot be fulfilled
    - 400=bad request, 404=not found, 401=unauthorized, 403=forbidden, ...
- **5xx: server error**
  - server failed to fulfill an apparently valid request
    - 500=internal server error, 501=not implemented, 502=bad gateway, 503=service unavailable, ...

Brilliant idea: unrecognized xnn codes treated as x00 codes!

HTTP/1.0 General Headers

optionally sent by either client & server

- **Date**
  - **Date: Sun, 06 Nov 1994 08:49:37 GMT**
  - 3 accepted date formats (the first is the preferred one):
    - Sun, 06 Nov 1994 08:49:37 GMT
      - RFC 822, updated by RFC 1123
      - Fixed-length field
    - Sunday, 06-Nov-94 08:49:37 GMT
      - RFC 850, obsoleted by RFC 1036
    - Sun Nov 6 08:49:37 1994
      - ANSI C's asctime() format

- **Pragma**
  - **Pragma: no-cache**
  - implementation-specific directives
    - The word "pragma" taken from programming languages (directives to compiler)
  - No-cache is the only popularly used pragma

HTTP/1.0 Headers
for resource handling & caching

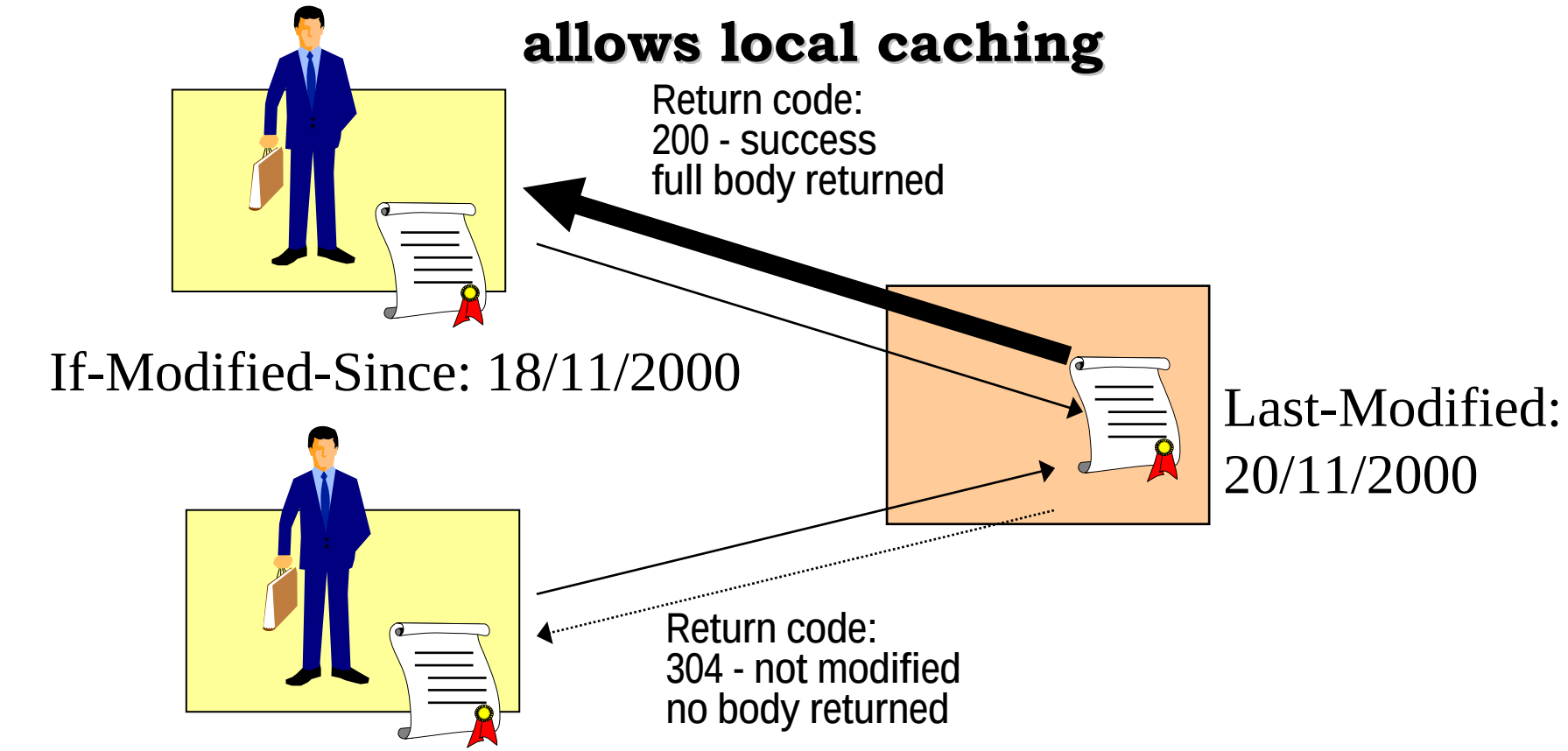
➔ **If-Modified-Since – sent by client**
- ⇨ If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
- ⇨ For conditional GET (see next slide)

➔ **Last-Modified - returned by server**
- ⇨ Last-Modified: Sat, 29 Oct 1994 19:43:31 GMT
- ⇨ Date and time the server “believes” the data was modified
- ⇨ semantically imprecise - file modification? Record timestamp? Date in case file dynamically generated?

➔ **Expires - sent by server**
- ⇨ Expires: Thu, 14 Dec 2000 16:00:00 GMT
- ⇨ Date after which a resource should be considered stale
  - →primitive caching expiration date functionality
  - →Allows to quantify how “volatile” a resource is
- ⇨ cannot force clients to update view, only on refresh

Conditional GET

If-Modified-Since header field allows local caching



HTTP/1.0 Headers
for redirection & back-tracking

➔ **Location - returned by server**
- ⇨ Location: http://www.unipa.it
- ⇨ indicates URL for automatic redirection to the resource
- ⇨ used in case of 3xx redirections

➔ **Referer - sent by client**
- ⇨ Referer: http://cerbero.elet.polimi.it
- ⇨ specifies address from which request was generated
  - → i.e. the page you come from
  - → none if request entered from keyboard
- ⇨ Applications: back button, caching optimization, logging statistics, etc
- ⇨ All sort of privacy issues! Must be careful with this…

HTTP/1.0 Headers
for information disclosure (1)

➔ **From - sent by client**
- ⇨ From: bianchi@elet.polimi.it
- ⇨ specify mailbox of human behind user agent
- ⇨ Not really used (privacy issues)

➔ **User-Agent - sent by client**
- ⇨ User-Agent: Mozilla/4.07 [en] (X11; I; Linux 2.0.36 i686)
- ⇨ identifies client software
- ⇨ why? Optimize layout, send based on capability of client
  - →Multi-channel portals build on this idea

HTTP/1.0 Headers
for information disclosure (2)

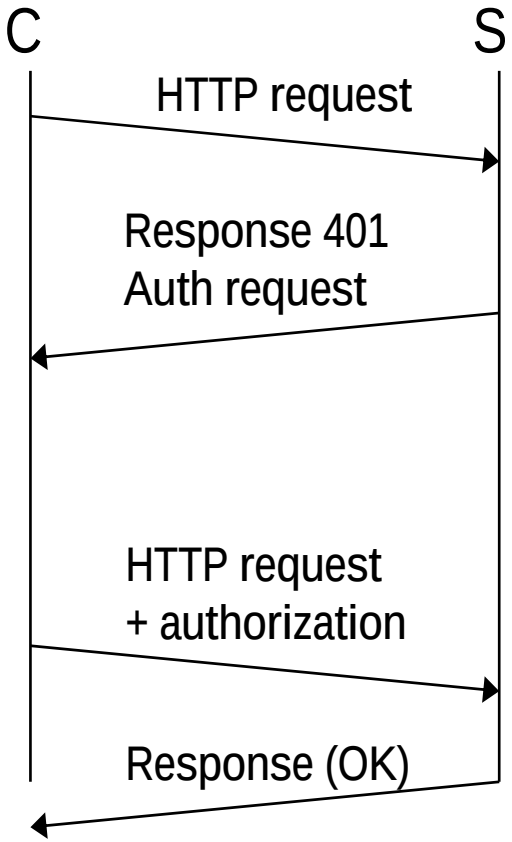
➔ **Server - returned by server**
- ⇨ Server: Netscape-Enterprise/2.01
- ⇨ identifies server software (origin server – no proxy info)
  - →Used for measurement & statistics
  - →Allows hackers to better prepare an attack :-)

➔ **Allow - returned by server**
- ⇨ lists set of supported methods
- ⇨ Allow: GET, HEAD
- ⇨ never used in practice - clients know what they can do

HTTP/1.0 Headers
for authentication

- **WWW-Authenticate - sent by server**
  - WWW-Authenticate: <challenge>
  - Es: WWW-Authenticate: basic realm="WallyWorld"
    - Basic=scheme used (may specify enhanced schemes)
    - Challenge string: assigned by server to identify protected space
  - included in 401 (unauthorized) response messages
  - tells client to resend request with Authorization: header
    - Authorization must be valid for the current "challenge"

- **Authorization - sent by client**
  - Authorization: <credentials>
  - Es: Authorization: basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
    - <credentials> = Base64(username:password)
    - Base64: coding done on 64 characters only.
      - A…Z a…z 0…9 + /
      - = used as special 65th symbol
      - See RFC 1521



Authentication does not mean encryption!!

Università degli Studi di Palermo

Incrementally added hacks

not really "standard" and consistently implemented
but extensively used

- **Accept: image/gif, image/jpeg, text/*, */***
  - ⇨ Used in a request, to specify which type of media can be accepted as response
- **Accept-Encoding: gzip**
  - ⇨ Allows to specify the encoding format acceptable for the client
- **Accept-Language: en**
  - ⇨ Allows to specify the desided language for the response
- **Retry-After: (date) or (seconds)**
  - ⇨ Frequently associated to a 503 (service unavailable) response
- **[Set-]Cookie: Part_Number="Rocket_Launcher_0001"; Version="1"; Path="/acme"**
- **… (many more) …**

Cookies

➔ **HTTP is stateless**
- ⇨ Need for cookies

➔ **Cookie: small txt strings**
- ⇨ Store information necessary to retrieve user state
  - → Preference & personalization
  - → Save passwords for further visits
  - → And a lot more

Example (set by finance.yahoo.com):
PRF
s=8388608&t=IONA+GSPN+CNXT+ISIL+ALVR+INTC
finance.yahoo.com/
1024
3400107776
30338494
644956128
29604307
*

➔ **Temporary/permanent**
- ⇨ Whether the cookie lasts for a single browsing session or beyond

➔ **Set by HTTP response; later on send by HTTP requests:**
- ⇨ [Set-]Cookie: Part_Number="Rocket_Launcher_0001"; Version="1"; Path="/acme“

➔ **A LOT of privacy issues!**
- ⇨ WinXP: See your cookies in \C:\Documents and Settings\yourname\Cookies
  - → Your cookie page SHOWS UP your navigation preferences!
- ⇨ Malicious cookie settings from some sites
  - → Goal: gain access to your personal information

Cookie Overview

- HTTP cookies are a mechanism for creating and using session-persistent state.
- Cookies are simple string values that are associated with a set of URL's.
- Servers set cookies using an HTTP header.
- Client transmits the cookie as part of HTTP request whenever an associated URL is visited in the future.

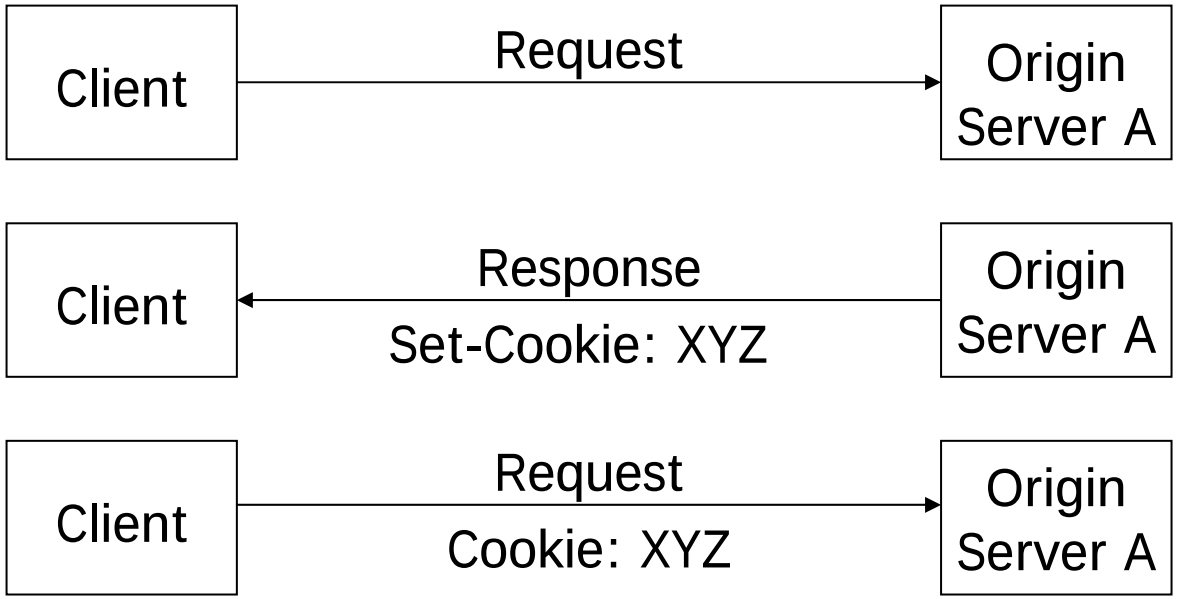
Terminology and Usefulness

➔Where are cookies used?

- ✓Shopping applications
- ✓Storing login information
- ✓Tracking pages visited by a user

Terminology and Usefulness

➔How do cookies work?



Anatomy of a cookie.

- **Cookie has 6 parts:**
  - Name
  - Value
  - Domain
  - Path
  - Expiration
  - Security flag
- **Name and Value are required, others have default value.**

Cookie details

- **Domain**
  - Indicates server name associated with cookie
  - Can be partial
    - Ex: Cookie associated with ".unc.edu" will be returned to any server with that ending
- **Path**
  - Indicates URL path name associated with cookie
  - Can be partial
- **Expire: Indicates when cookie will expire**
- **Secure: Indicates only send when secure**

Cookie header syntax

- **Header name is “Set-cookie”**
- **Header value is attribute/value pairs**

```
Set-cookie: name=cname; value=cvalue;
  domain=.cs.unc.edu; path=/~kmp
```

Setting a cookie.

- **A cookie is set using the "Set-cookie" header in an HTTP response.**
- **String value of the Set-cookie header is parsed into semi-colon separated fields that define the different parts of the cookie.**
- **Java servlet API has support for cookies**
  - Cookie class
  - addCookie method in HttpServletResponse
- **Cookie is stored by the client.**

Sending cookies

- **Every time a client makes an HTTP request, it tests every cookie for a match.**
- **Cookies match if…**
  - ⇨Cookie domain is suffix of URL server.
  - ⇨Cookie expiration has not passed.
  - ⇨Cookie path is prefix of URL path.
  - ⇨Cookie security flag is on and connection is secure.
- **If a match is made, then name/value pair of cookie is sent as “Cookie” header in request.**

Cookie Matching

- **Biggest misunderstanding:**
  - Servers do not RETRIEVE cookies!
  - Servers RECEIVE cookies previously planted.
- **Step 1:**
  - Some response by server installs cookie with "Set-cookie" header.
  - Client saves cookie to disk.

Cookie Matching

➔Step 2:

- ⇨Browser goes to some page which *matches* specification of previously received cookie.
- ⇨Cookie name and value sent in request as "Cookie" HTTP header.

➔Step 3:

- ⇨Servlet detects presence of cookie uses cookie value as part of content generation.

An Example

- **We can avoid explicit registration of user id by using cookies.**
  - If cookie is present, use that to look up state.
  - If not, generate and set new cookie.
- **Advantages?**
  - Anonymous and transparent.
- **Disadvantages?**
  - If user moves to different machine, can't get to previously stored cart.

Content management issues

➔Early days of the Internet (<1990)

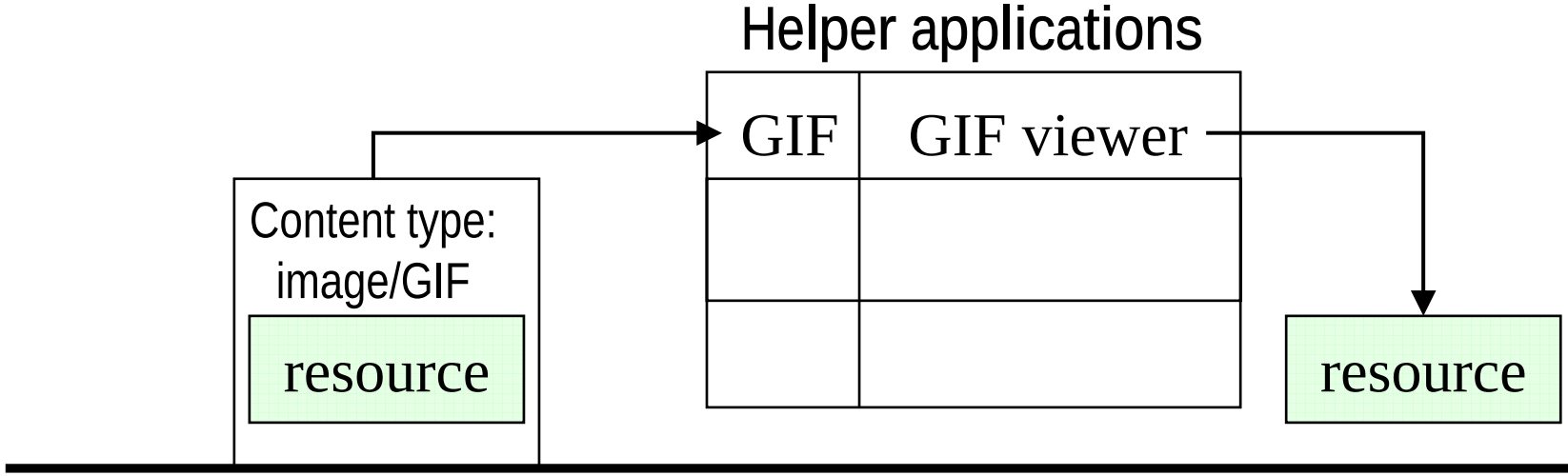
- ⇨messages in english text
- ⇨No other media

➔Resources today:

- ⇨text
  - →in languages with accents (italian, french, german,…)
  - →Non latin alphabets (Russian, Hebrew)
  - →languages wihout alphabet (Chinese, Japanese)
- ⇨other resources (audio, video, images)
  - →each media with various coding schemes

Entity header

- **Meta-information about the entity body**
  - Content-Type
  - Content-Encoding
- **MIME-like approach**
  - Problem of content management originally appeared in email.
  - Solution: Multipurpose Internet Mail Extension (RFC 1521)
- **Key idea: associate a content descriptor to each content**



HTTP content management

➔Content-Type - sent by server

- ⇨ MIME-like field, specifying the media-type.
- ⇨ Format: type/subtype
- ⇨ media type values registered in IANA (Internet Assigned Numbers Authority).
- ⇨ Content-Type: text/html
  - ➔with optional charset parameter: default ISO-8859-1;
- ⇨ Content-Type: image/jpeg
- ⇨ nasty one: multipart/mixed

➔Content-Encoding - sent by either

- ⇨ selects an encoding (data compression scheme) for the transport, not the content
- ⇨ Content-Encoding: x-gzip (x-compress)
- ⇨ resource typically stored with this coding, and is decoded before rendering
- ⇨ sadly, no common support for encodings (Windows)

Even a man can do it!

- telnet www.tti.unipa.it
- correct: telnet www.tti.unipa.it 80
- GET /index.html HTTP/1.0
- (blank line)