HTTP

URI

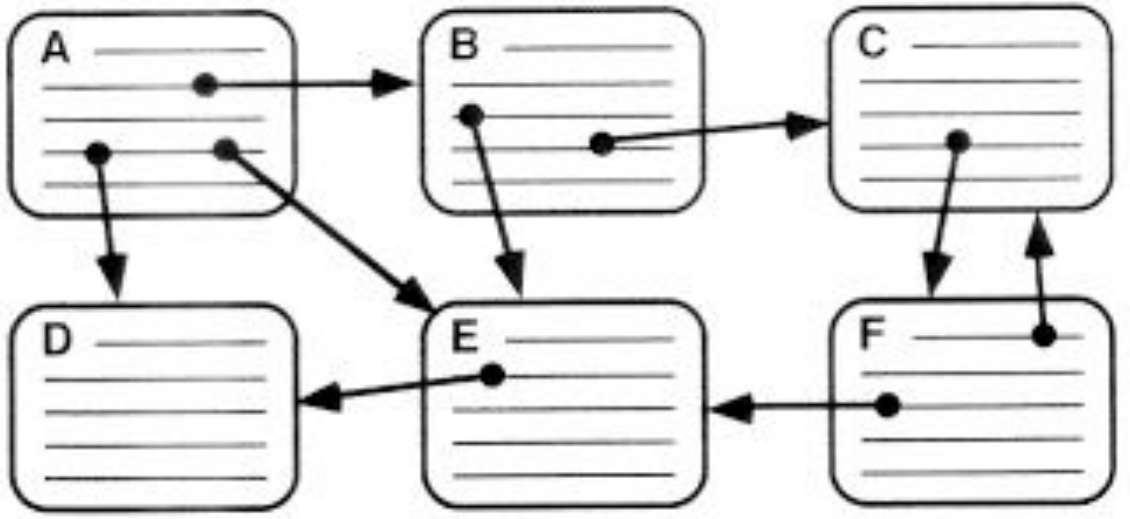
HTTP 1, 2

Proxies

Cookies

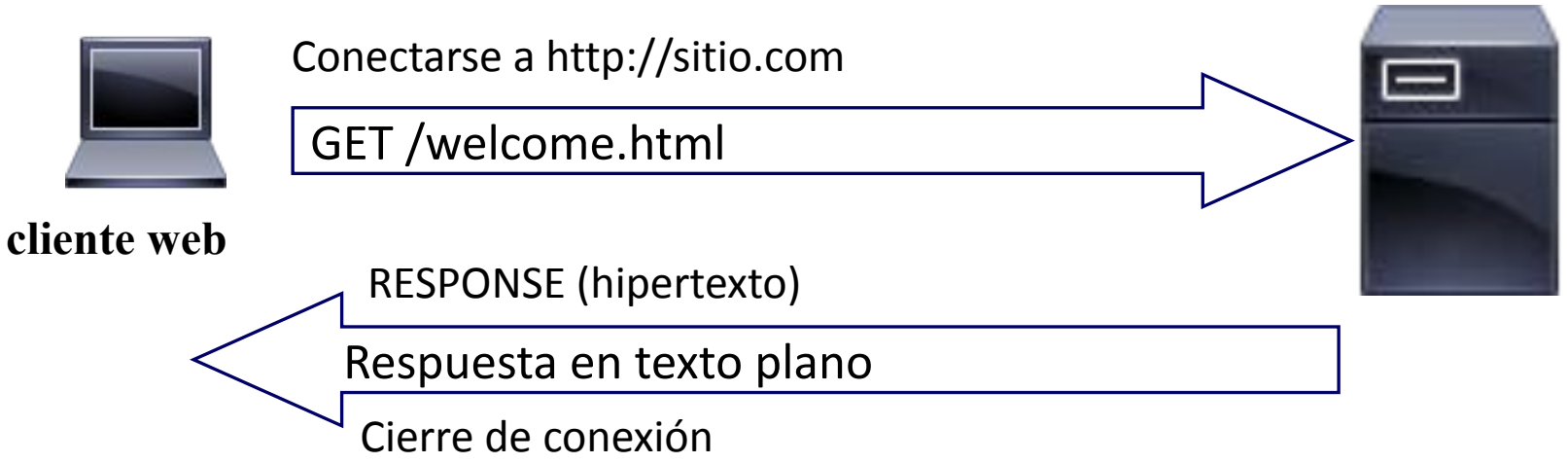
Previo a HTTP

- Vía FTP se obtenía un documento o texto
- En base a las referencias se debía acceder a otro sitio FTP y descargar el/los textos deseados
- Surge la necesidad de aplicaciones basadas en "hipertextos"



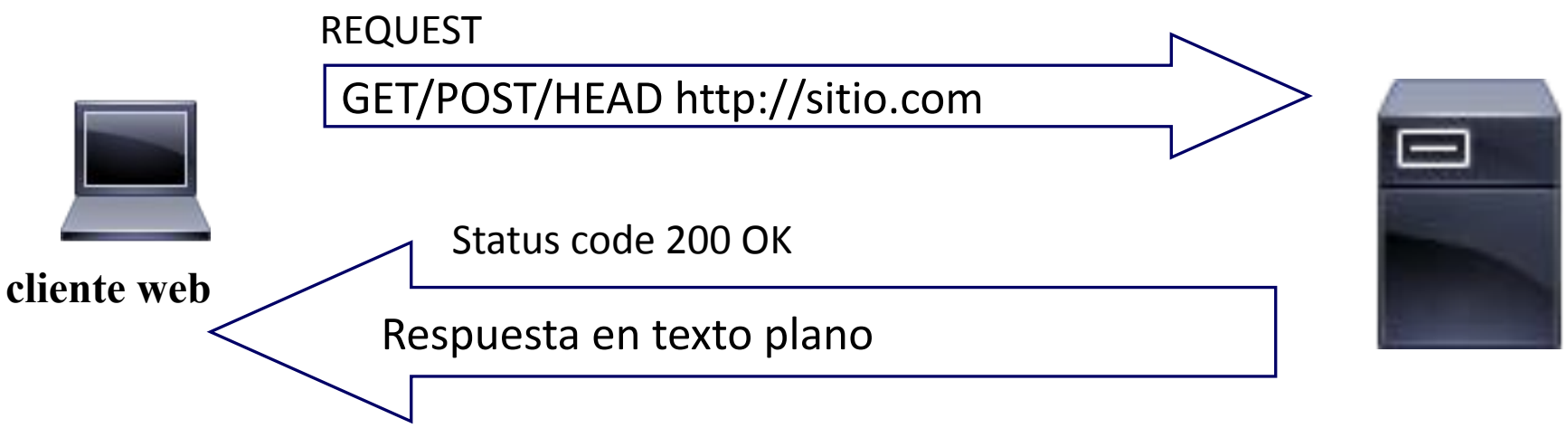
HTTP 0.9 (1991)

- Protocolo basado en texto de recuperación de información
- Mecanismo *request – response*
- Contenido estático
- No mantiene estado (no establece una sesión)
- Orientado a "línea de comando"



HTTP 1.0 (1996)

- "Browser friendly"
- Comandos como GET, POST, HEAD
- Códigos de estado
- No solo hipertexto



Un request por CADA recurso (no confundir con "página web")

HTTP 1.1 (1999)

- Cabeceras en las peticiones
- Métodos PUT, DELETE, TRACE, OPTIONS
- Negociación de contenido
- Soporte de cache
- Conexiones persistentes

Cada recurso se identifica con una URI (Uniform Resource Identifier)

cliente web

REQUEST

GET/POST/HEAD http://sitio.com

X-HEADER: X-value

Status code 200 OK

X-HEADER: X-value

respuesta en texto plano

HTTP/2

- Protocolo binario
- Compresión de headers
- Multiplexación de recursos
- Server push
- …

http://www.http2demo.io/

HTTP: recursos

- Un recurso es un bloque de información identificado por su URI
- Puede ser un archivo (físico) o generado por un programa (abstracto)
- URL: Uniform Resource Locator
- URN: Uniform Resource Name
- RFC 1630, 2396, 2718, 3305, 3986
- Internationalized Resource Identifiers (IRIs): RFC 3987

URI (ver RFC 3986 3.3)

<scheme>://<authority><path>?<query>

- El path termina con el primer "?" o "#" o si no hay más caracteres
- Puede ser relativo o absoluto
- Si representa una aplicación puede recibir parámetros

```
• http:www.example.org/path/name?param1;param2;param3
• /relative_path/name?user="..";pwd="..."
• ../../logo.png
```

URL:sintaxis

- Identifica un recurso por su ubicación (*location*)

Algunos sitios o recursos requieren autenticación

puerto (opcional si es 80)

<scheme>://<user>:<password>@<host>:<port>/<path>?<query>

Protocolo a utilizar: por ejemplo HTTP, FTP, FILE

http://soyyo:miclave@www.unsitio.com:90

http://192.168.0.100

http://localhost:8080

URL: sintaxis

Identificación del recurso dentro del servidor

Parámetros

<scheme>://<user>:<password>@<host>:<port>/<path>?<query>

http://soyyo:miclave@www.unsitio.com:90/index.html

http://soyyo:miclave@www.unsitio.com:90/pagZZ.html?width=1024&lang=es

URL: sintaxis

- Puede incluir al final un "fragmento"

```
http://www.unsitio.com/intro.html#chapter1
```


11

URN

- Identifica un recurso por su nombre
- No implica que el recurso exista o cómo acceder a él

```
urn:isbn:0132856204
urn:isan:0000-0002-3C36-0000-Y-0000-0000-9
urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66
urn:www.apache.org:
```

http://www.iana.org/assignments/urn-namespaces/urn-namespaces.xhtml

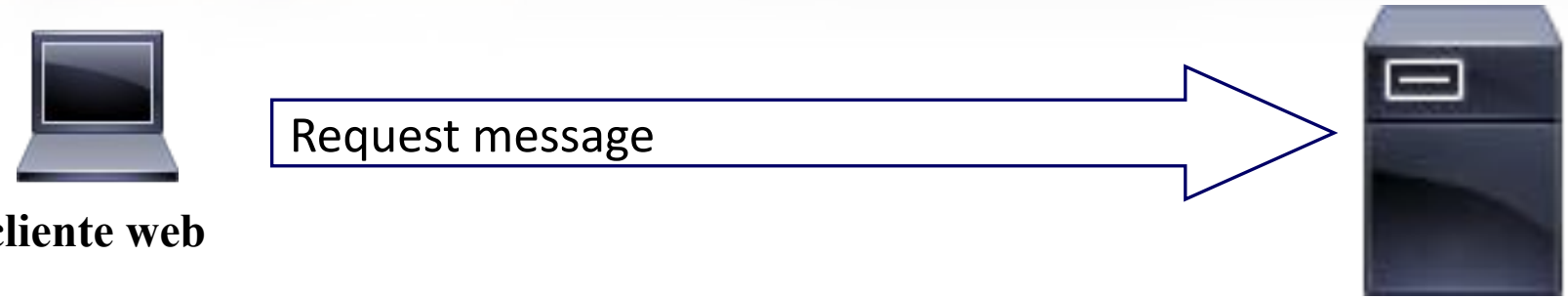
URI Scheme

- URIs pueden ser usadas para acceder a recursos, ya sea por medio de URL o URN
- Ejemplo en html

```
<a href="/img/logo.png">
<a href="urn:isbn:0453457513">
```

Mensajes HTTP

- El cliente y el servidor intercambian mensajes HTTP
  - *Request message* (cliente → servidor)
  - *Response message* (servidor → cliente)



- Solicitudes (*Request*)
  - **GET**: Solicita un recurso al servidor
  - **HEAD**: Solicita solo los headers del recurso.
  - **POST**: Envía información al servidor para ser procesada

Ejemplo: HTTP - GET

cliente web

```
GET / HTTP/1.1
Host: www.clarin.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml
Accept-Language: es-ar,es
Accept-Encoding: gzip,deflate
Keep-Alive: 300
Connection: keep-alive
```

Start line

Headers

Sólo US-ASCII

Start line

Headers

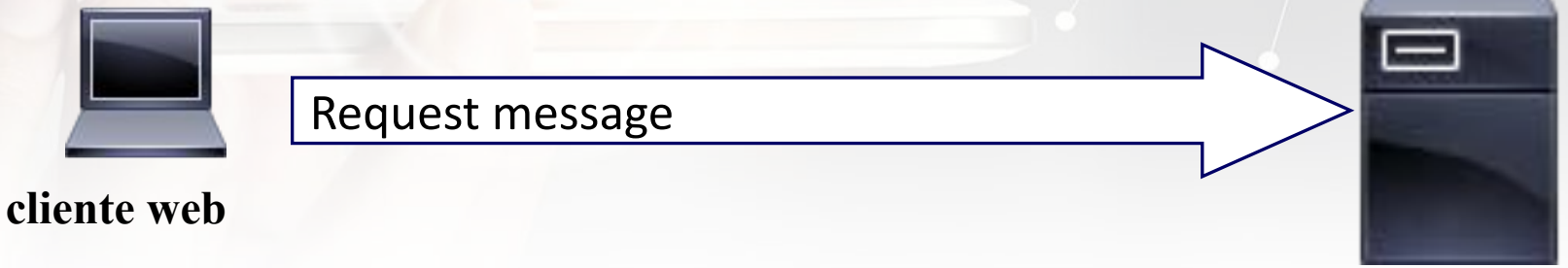
Body

```
HTTP/1.1 200 OK
Date: Mon, 15 Feb 2010 12:47:26 GMT
Server: Apache/2
Content-Length: 24495
Content-Type: text/html; charset=ISO-8859-1

<html><head>
<title>Alerta meteorológico en Capital</title>
.....
```

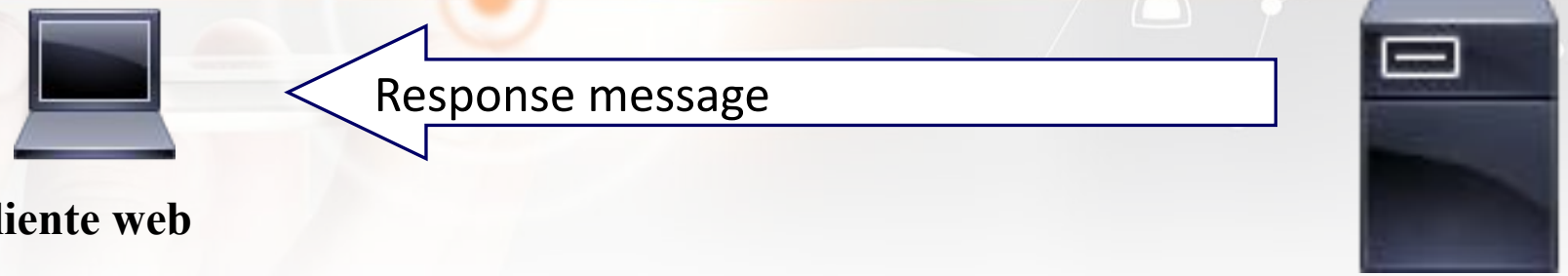
Sólo US-ASCII

Mensajes HTTP



- Solicitudes (*Request*) menos utilizados
  - **PUT**: Envía un recurso al servidor
  - **TRACE**: Analiza el recorrido de la solicitud.
  - **OPTIONS**: Consulta los métodos disponibles en el servidor.
  - **DELETE**: Elimina un recurso del servidor.

Mensajes HTTP



- Respuestas (*Response*)
  - **Start Line:** Versión, código de respuesta y mensaje.

1XX	Información – El proceso continúa
2XX	Éxito - Acción recibida, comprendida y aceptada
3XX	Redirección – Se requiere nueva acción
4XX	Error en cliente – Sintaxis y/o Semántica inválida
5XX	Error en servidor – Falló pedido correcto

Tipos de información

Si HTTP sólo transmite texto, ¿cómo puedo transferir imágenes, video, etc.?

- Utiliza MIME para describir contenido multimedia
- Cada objeto es etiquetado por el web server
  - text/html
  - text/plain
  - video/mp4
  - image/jpeg
  - application/pdf
  - …

Ejemplo: HTTP - POST

cliente web

```
POST /itbaV/mynav.asp HTTP/1.1
Host: iol.itba.edu.ar
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml
Referer: http://iol.itba.edu.ar/itbaV/welcome.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 76

secretNr=378924198&txtdni=12345678&txtpwd=mipass&Sub
mit=Conectar&cmd=login
```

Body

```
HTTP/1.1 200 OK
Connection: close
Date: Mon, 15 Feb 2019 17:05:22 GMT
Server: Microsoft-IIS/6.0
Content-Length: 11276
Content-Type: text/html

<html><head>
<title>ITBA OnLine</title>
.....
```

Get vs Post

- GET requests
  - pueden ser "cacheados"
  - el browser los mantiene en el historial
  - pueden ser "bookmarked"
  - tienen longitud acotada
  - sólo para pedir datos
- POST requests
  - nunca son "cacheados"
  - no se mantienen en el historial del browser
  - no pueden ser "bookmarked"
  - no tienen restricción de longitud de datos

Ejemplo: HTTP - OPTIONS

cliente web

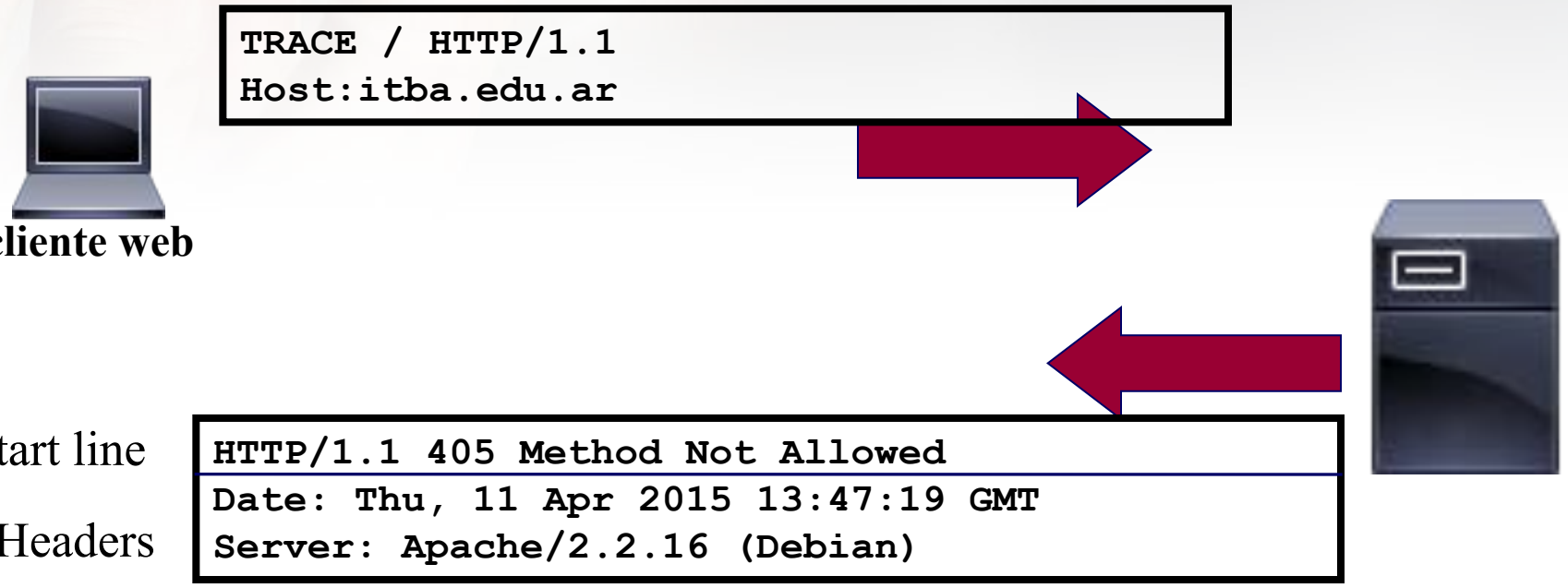
```
OPTIONS * HTTP/1.1
User-Agent: Mozilla/4.0 (compatible;
MSIE5.01; Windows NT)
```

Start line

Headers

```
HTTP/1.1 200 OK
Date: Mon, 01 Aug 2019 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: httpd/unix-directory
```

Ejemplo: HTTP - TRACE



https://tools.ietf.org/html/rfc7231#section-4.3.8

Headers HTTP

- Estructura: `<nombre>: <valor asociado>`
- Finalizan con una línea en blanco.
- Tipos:
  - Generales
  - De solicitudes
  - De respuestas
  - De contenido

Usualmente se envían en este orden

Headers generales

- Cache-control: Directivas para cache
- Connection: Se definen opciones de conexión
- Date: Fecha de creación del mensaje
- Transfer-Encoding: Indica el encoding de transferencia
- Via: Muestra la lista de intermediarios por los que pasó el mensaje.

Headers de solicitud

- Accept : Tipo de contenido aceptado por el cliente
- Accept-Charset: Charset (ISO-xxxx, UTF-8) aceptado por el cliente
- Accept-Encoding: Encoding (gzip, compress) aceptado por el cliente
- Expect: Comportamiento esperado del server frente al request.
- From: Email del usuario de la aplicación que generó el request.
- Host: Servidor y puerto destino del request.
- If-Modified-Since: Condiciona al request a la fecha indicada.
- Referer: URL del documento que generó el request
- User-Agent: Aplicación que generó el request
- Upgrade: solicita que use otro protocolo
- Range: solicita un rango (en bytes) del recurso

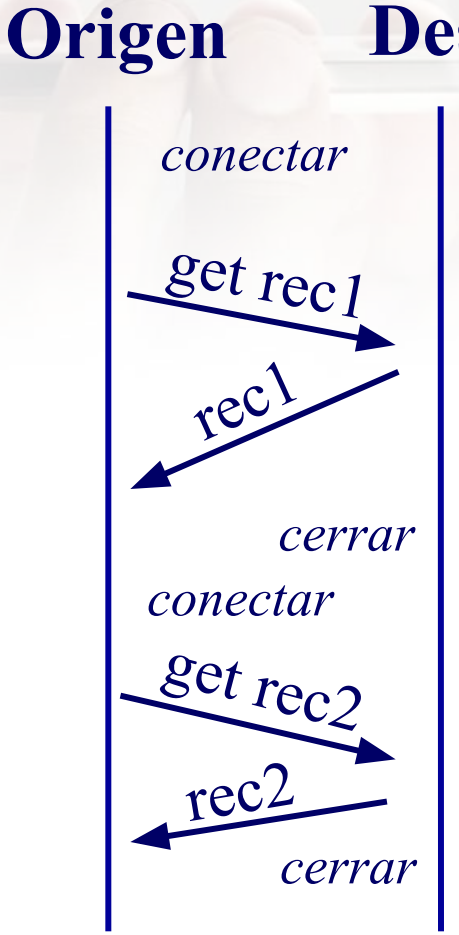
Headers de respuesta

- Age: Estimación en segundos del tiempo que fue generada la respuesta en el server.
- Connection: close, keep-alive
- Location: URI a redireccionar
- Retry-After: Tiempo de delay para reintento
- Server: Descripción del software del server.
- Authorization: indica que el recurso necesita autorización

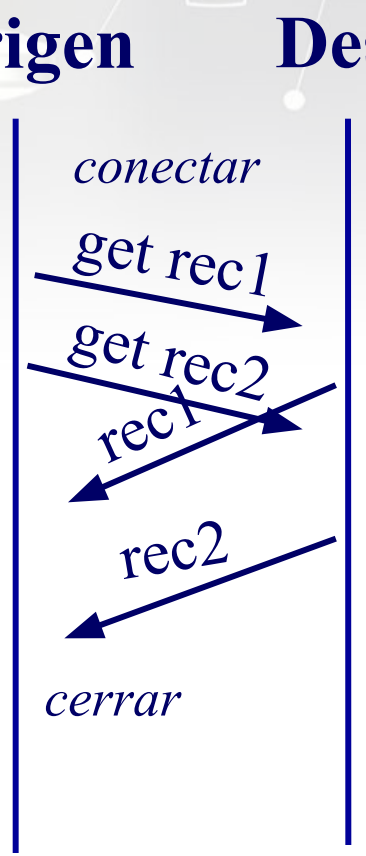
Headers de contenido

- Allow: Métodos aplicables al recurso.
- Content-Encoding
- Content-Length
- Content-Location
- Content-MD5
- Content-Type
- Expires: Fecha de expiración del recurso.
- Last-Modified: Fecha de modificación del recurso

Conexiones HTPP



No persistente

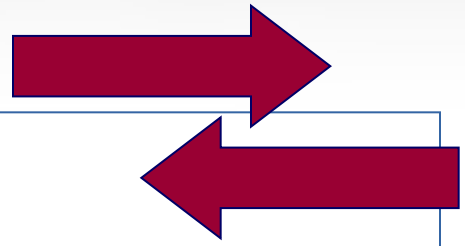


Persistente

Negociación de contenido

cliente web

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```



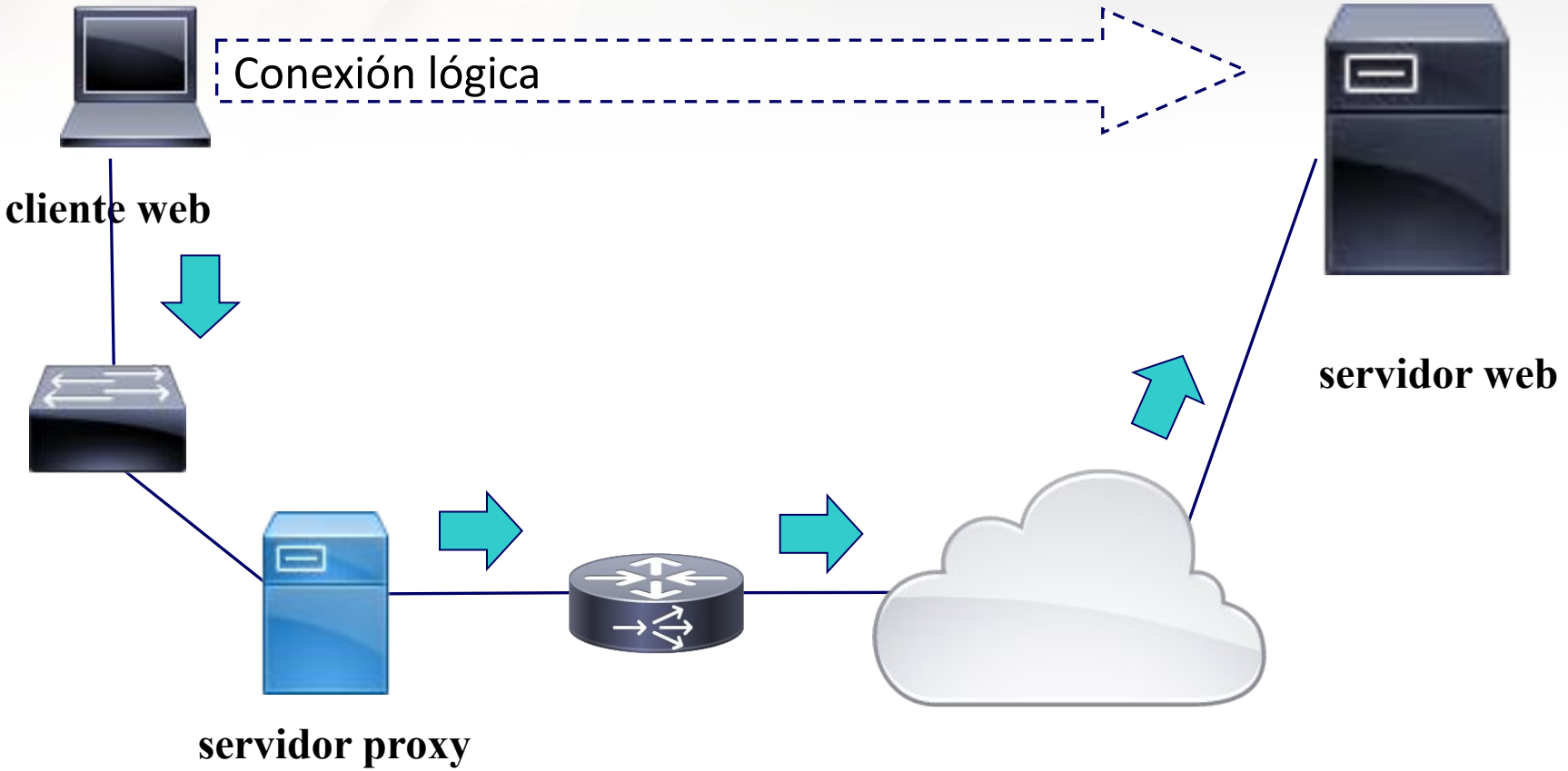
```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Content-Location: http://www.example.com/alice.en.html
```

alternativa:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

Proxy servers

Un *proxy server* actúa como intermediario entre una aplicación cliente y un web server. Puede ser **explícito** o **transparente**.

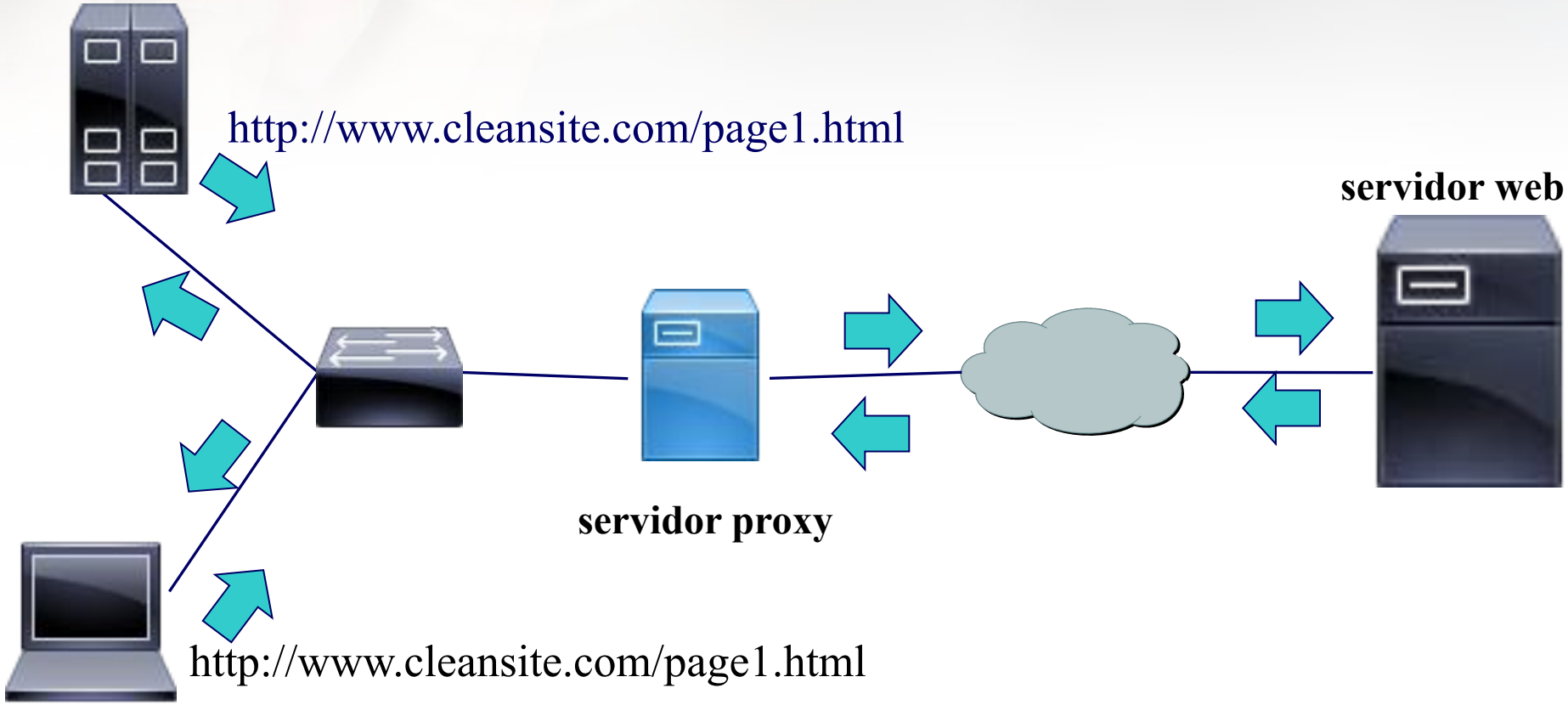


Proxy servers

Un *proxy server* permite controlar el acceso a determinados sitios o páginas, y almacenar en caché recientes consultas.

Proxy servers

Un *proxy server* permite controlar el acceso a determinados sitios o páginas, y almacenar en caché recientes consultas.



Cache: sin directivas

proxy

servidor web

GET /schwifty

200 OK
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Oh, yeah! You gotta get schwifty

HTTP

Cache: max-age

GET /schwifty

200 OK
Cache-control: max-age:600
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Age: 50
Cache-control: max-age:600
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Cache-control: max-age:600
Oh, yeah! You gotta get schwifty

Cache: last-modified

proxy

servidor web

GET /schwifty

200 OK
Last-Modified: Thu, 1 August
Etag: a1b5c78xgf
Oh, yeah! You gotta get schwifty

GET /schwifty

200 OK
Last-Modified: Thu, 1 August…
Etag: a1b5c78xgf
Oh, yeah! You gotta get schwifty

GET /schwifty
If-modified since: Thu, 1 August…
If-non-match: a1b5c78xgf

304 Not modified

Cache: max-age y last-modified

GET /schwifty

200 OK
Cache-control: maxage=60
Last-Modified: Thu, 1 August
Etag: a1b5c78xgf
Oh, yeah! You gotta get schwifty

GET /schwifty

proxy

GET /schwifty

200 OK
Cache-control: maxage=60
Last-Modified: Thu, 1 August…
Etag: a1b5c78xgf
Oh, yeah! You gotta get schwifty

servidor web

Cookies

- Pequeño texto de información enviada por el servidor y almacenada por el browser
- Usadas para mantener un estado entre el cliente y el servidor
  - Servidor envía cookie a cliente HTTP (*«set-cookie» response header*)
  - Cliente HTTP retorna cookie al servidor (*«cookie» request header*)
- Persistencia
  - Session cookie
  - Persistent cookie
- Third-party cookie
- Secure cookie

Cookies

cliente web

```
POST /login.html HTTP/1.1
username=jPerez
pwd=123456
```

Valida usuario y clave.
Crea un sessionId

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionId=123xyz; Expires=Wed, 04 ...
Set-Cookie: lastSession=2019/06/08
...
```

```
GET /menu.html HTTP/1.1
Host: www.example.com
Cookie: sessionId=123xyz; lastSession=2019/08/08
...
```

¿La sesión es válida?

```
HTTP/1.1 200 OK
Content-Type: text/html
```

cURL

- Herramienta de línea de comandos para transferir recursos en base a URLs
  - HTTP (POST, PUT)
  - HTTPs
  - FTP (rfc 959)
  - DICT (rfc 2229)
  - POP3 (rfc 1996)
  - SMTP (rfc 5321)
  - etc.

cURL: ejemplos

- Incorrectos
  - curl https:infobae.com
  - curl https:infobae.com –head
- Correctos
  - curl http://google.com/humans.txt
  - curl https://www.google.com/humans.txt -i
  - curl -X POST -F 'locale=en' *url*

wget

- Pensado para descargar archivos a disco
  - wget https://wordpress.org/latest.zip
  - wget -i files.txt
  - wget --limit-rate=500k ...

files.txt es un archivo de texto con una URL por línea

netcat

Como HTTP es un protocolo de texto, podemos "armar" los datos a enviar manualmente. Para ello necesitamos una aplicación que simplemente se conecte al servidor, nos pida los datos, los envíe y nos muestre la respuesta

netcat permite esto y mucho más

- `netcat www.google.com 80`
- `netcat -l -p 8080 -v`

Material de lectura

- Capítulo 2.2 de la bibliografía
- Códigos de status HTTP
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- GET vs POST:
  https://www.w3schools.com/tags/ref_httpmethods.asp