

# The Hyper-Text Transfer Protocol (HTTP)

Antonio Carzaniga

Faculty of Informatics  
Università della Svizzera italiana

October 13, 2016

- HTTP message formats
- HTTP methods
- Status codes
- Headers
- Web caching

# Anatomy of a Request

GET /carzaniga/index.html HTTP/1.1	
Host: www.inf.usi.ch	
Connection: close	
User-agent: Mozilla/4.0	
Accept-Language: it	

# Anatomy of a Request

GET /carzaniga/index.html HTTP/1.1	request line
Host: www.inf.usi.ch	
Connection: close	
User-agent: Mozilla/4.0	
Accept-Language: it	

# Anatomy of a Request

GET /carzaniga/index.html HTTP/1.1	request line
Host: www.inf.usi.ch Connection: close User-agent: Mozilla/4.0 Accept-Language: it	zero or more header lines

# Anatomy of a Request

GET /carzaniga/index.html HTTP/1.1	request line
Host: www.inf.usi.ch Connection: close User-agent: Mozilla/4.0 Accept-Language: it	zero or more header lines
	empty line

# Anatomy of a Request

GET /carzaniga/index.html HTTP/1.1	request line
Host: www.inf.usi.ch Connection: close User-agent: Mozilla/4.0 Accept-Language: it	zero or more header lines
	empty line
	object body (possibly empty)

# Anatomy of a Request (2)

- Request line

```
GET /carzaniga/index.html HTTP/1.1
```



# Anatomy of a Request (2)

- Request line

GET /carzaniga/index.html HTTP/1.1  
↑  
*method*

# Anatomy of a Request (2)

- Request line

*space*  
↓  
GET /carzaniga/index.html HTTP/1.1  
↑  
*method*

# Anatomy of a Request (2)

- Request line

The diagram shows a request line: GET /carzaniga/index.html HTTP/1.1. Annotations include: a downward arrow from the word "space" pointing to the space between "GET" and the URL; an upward arrow from the word "method" pointing to "GET"; and an upward arrow from the word "URL" pointing to the path "/carzaniga/index.html".

space

GET /carzaniga/index.html HTTP/1.1

method URL









# Anatomy of a Request (2)

## ■ Request line

*space* ↓  
GET /carzaniga/index.html HTTP/1.1  
↑ ↓  
*method* *URL* *version*

## ■ Header line

*space* ↓  
Host: www.inf.usi.ch  
↑  
*name*



# Anatomy of a Request (2)

## ■ Request line

*space* ↓ *space*  
GET ↓ /carzaniga/index.html ↓ HTTP/1.1  
↑ ↑ ↑  
*method* *URL* *version*

## ■ Header line

*space*  
↓  
Host: ↓ www.inf.usi.ch  
↑ ↑  
*name* *value*

# Anatomy of a Request (2)

## ■ Request line

*space* ↓ *space*  
GET ↓ /carzaniga/index.html ↓ HTTP/1.1  
↑ ↑ ↑  
*method* *URL* *version*

## ■ Header line

*space*  
↓  
Host: ↓ www.inf.usi.ch  
↑ ↑  
*name* *value*

- Line terminator: *CRLF* (“carriage return” and “line feed”)
  - ▶ two bytes: numeric values 13 and 10

**GET** retrieve the object identified by the URL

**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**HEAD** like GET, but without the body

- useful for testing the validity of links

**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**HEAD** like GET, but without the body

- useful for testing the validity of links

**POST** allows one to submit data to the server

- e.g., a mail message in a web mail system, a form in an e-commerce site...
- the given URL is the object that *handles* the posting

**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**HEAD** like GET, but without the body

- useful for testing the validity of links

**POST** allows one to submit data to the server

- e.g., a mail message in a web mail system, a form in an e-commerce site...
- the given URL is the object that *handles* the posting

**PUT** requests that the enclosed object be stored under the given URL

**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**HEAD** like GET, but without the body

- useful for testing the validity of links

**POST** allows one to submit data to the server

- e.g., a mail message in a web mail system, a form in an e-commerce site...
- the given URL is the object that *handles* the posting

**PUT** requests that the enclosed object be stored under the given URL

**DELETE** deletes the given object



**GET** retrieve the object identified by the URL

**OPTIONS** requests the available communication options for the given object

**HEAD** like GET, but without the body

- useful for testing the validity of links

**POST** allows one to submit data to the server

- e.g., a mail message in a web mail system, a form in an e-commerce site...
- the given URL is the object that *handles* the posting

**PUT** requests that the enclosed object be stored under the given URL

**DELETE** deletes the given object

**TRACE** see RFC 2616, Section 9.8

**CONNECT** see RFC 2616, Section 9.8

# Anatomy of a Response

```
HTTP/1.1 405 Method Not Allowed
Date: Fri, 18 Mar 2005 01:18:04 GMT
Server: Apache/2.0.46 (Red Hat)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 329
Connection: close
Content-Type: text/html
```

```
<html><head>
<title>405 Method Not Allowed</title>
</head><body>
. . .
```

# Anatomy of a Response

HTTP/1.1 405 Method Not Allowed	status line
Date: Fri, 18 Mar 2005 01:18:04 GMT	
Server: Apache/2.0.46 (Red Hat)	
Allow: GET,HEAD,POST,OPTIONS,TRACE	
Content-Length: 329	
Connection: close	
Content-Type: text/html	
<html><head>	
<title>405 Method Not Allowed</title>	
</head><body>	
. . .	

# Anatomy of a Response

HTTP/1.1 405 Method Not Allowed	status line
Date: Fri, 18 Mar 2005 01:18:04 GMT Server: Apache/2.0.46 (Red Hat) Allow: GET,HEAD,POST,OPTIONS,TRACE Content-Length: 329 Connection: close Content-Type: text/html	zero or more header lines
<html><head> <title>405 Method Not Allowed</title> </head><body> . . .	

# Anatomy of a Response

HTTP/1.1 405 Method Not Allowed	status line
Date: Fri, 18 Mar 2005 01:18:04 GMT Server: Apache/2.0.46 (Red Hat) Allow: GET,HEAD,POST,OPTIONS,TRACE Content-Length: 329 Connection: close Content-Type: text/html	zero or more header lines
	empty line
<html><head> <title>405 Method Not Allowed</title> </head><body> . . .	

# Anatomy of a Response

HTTP/1.1 405 Method Not Allowed	status line
Date: Fri, 18 Mar 2005 01:18:04 GMT Server: Apache/2.0.46 (Red Hat) Allow: GET,HEAD,POST,OPTIONS,TRACE Content-Length: 329 Connection: close Content-Type: text/html	zero or more header lines
	empty line
<html><head> <title>405 Method Not Allowed</title> </head><body> . . .	object body (possibly empty)

# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

*version*





# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

*version*      *status code*

# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

*version*      *status code*      *brief description*

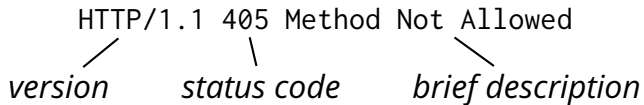
The diagram illustrates the structure of an HTTP status line. The text 'HTTP/1.1 405 Method Not Allowed' is shown at the top. Three lines of text below it, 'version', 'status code', and 'brief description', are connected to the corresponding parts of the status line by diagonal lines. 'version' points to 'HTTP/1.1', 'status code' points to '405', and 'brief description' points to 'Method Not Allowed'.

# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

*version*      *status code*      *brief description*



- The *status code* is a 3-digit value (e.g., 200 or 401)

# Anatomy of a Response

- Status line

HTTP/1.1 405 Method Not Allowed

*version*      *status code*      *brief description*

The diagram illustrates the structure of an HTTP status line. The text 'HTTP/1.1 405 Method Not Allowed' is shown above three labels: 'version', 'status code', and 'brief description'. Three lines connect the labels to the corresponding parts of the status line: a line from 'version' to 'HTTP/1.1', a line from 'status code' to '405', and a line from 'brief description' to 'Method Not Allowed'.

- The *status code* is a 3-digit value (e.g., 200 or 401)
- The rest has exactly the same structure as a request

# Status Codes

1xx “informational” (see Section 10.1 of RFC 2616)

1xx “informational” (see Section 10.1 of RFC 2616)

2xx successful operation (see Section 10.2 of RFC 2616)

1xx “informational” (see Section 10.1 of RFC 2616)

2xx successful operation (see Section 10.2 of RFC 2616)

3xx redirection. E.g., indicates that the object has moved, either temporarily or permanently



- 1xx “informational” (see Section 10.1 of RFC 2616)
- 2xx successful operation (see Section 10.2 of RFC 2616)
- 3xx redirection. E.g., indicates that the object has moved, either temporarily or permanently
- 4xx client error. E.g., malformed request (400), object not found (404), method not allowed (405), unauthorized (401).

- 1xx “informational” (see Section 10.1 of RFC 2616)
- 2xx successful operation (see Section 10.2 of RFC 2616)
- 3xx redirection. E.g., indicates that the object has moved, either temporarily or permanently
- 4xx client error. E.g., malformed request (400), object not found (404), method not allowed (405), unauthorized (401).
- 5xx server error. E.g., internal server error (500), service overloaded (503)



- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding

- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - ▶ e.g., Accept-Charset, Accept-Encoding

- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - ▶ e.g., Accept-Charset, Accept-Encoding
- Object properties useful for cache management
  - ▶ e.g., Expires, Last-Modified, ETag

- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - ▶ e.g., Accept-Charset, Accept-Encoding
- Object properties useful for cache management
  - ▶ e.g., Expires, Last-Modified, ETag
- Explicit cache control
  - ▶ e.g., Cache-Control

- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - ▶ e.g., Accept-Charset, Accept-Encoding
- Object properties useful for cache management
  - ▶ e.g., Expires, Last-Modified, ETag
- Explicit cache control
  - ▶ e.g., Cache-Control
- Method-specific responses
  - ▶ e.g., Allow as a response to OPTIONS



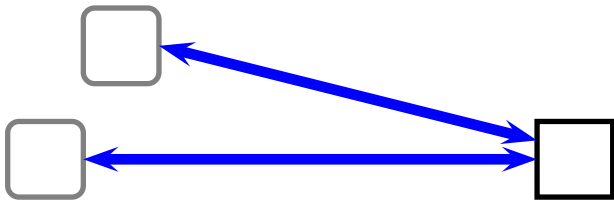
- Object characterization
  - ▶ e.g., Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - ▶ e.g., Accept-Charset, Accept-Encoding
- Object properties useful for cache management
  - ▶ e.g., Expires, Last-Modified, ETag
- Explicit cache control
  - ▶ e.g., Cache-Control
- Method-specific responses
  - ▶ e.g., Allow as a response to OPTIONS
- Authorization/identification
  - ▶ e.g., Authorization



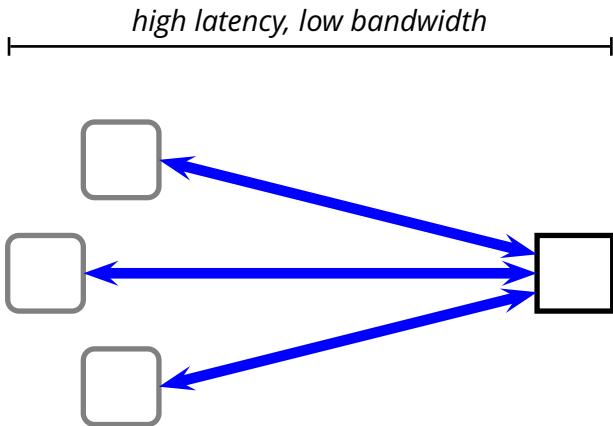
- Same idea as caching in a memory hierarchy



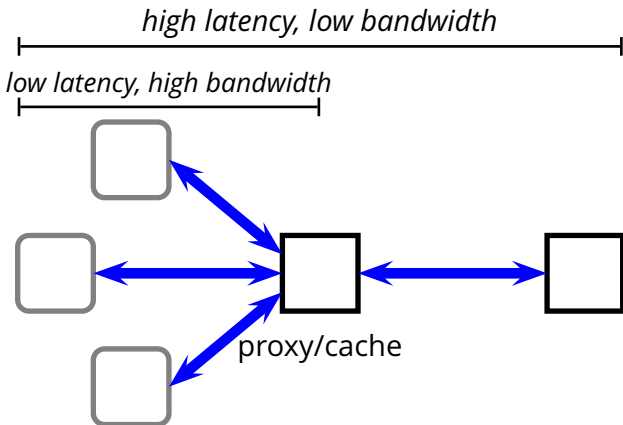
- Same idea as caching in a memory hierarchy



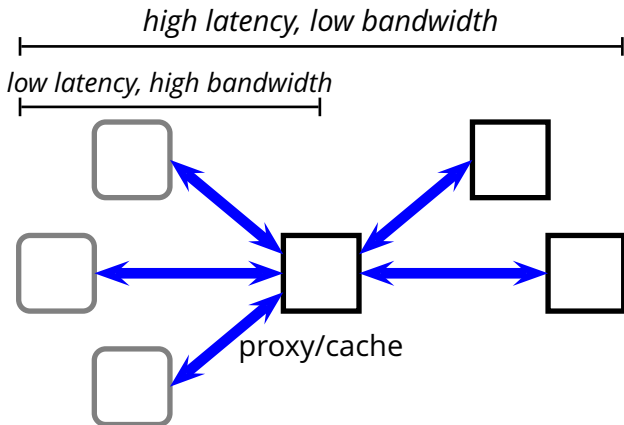
- Same idea as caching in a memory hierarchy



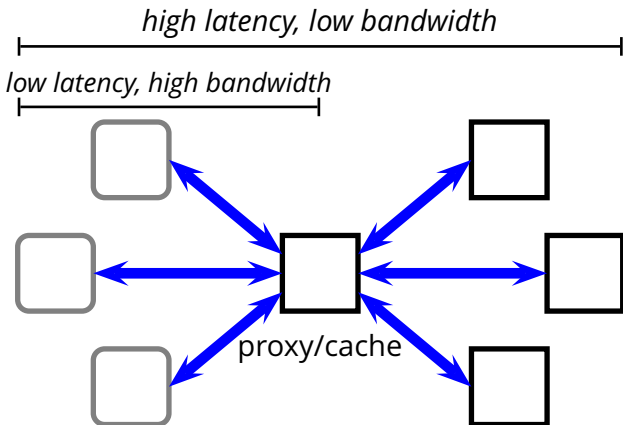
- Same idea as caching in a memory hierarchy



- Same idea as caching in a memory hierarchy



- Same idea as caching in a memory hierarchy

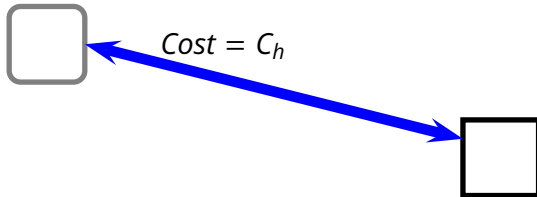




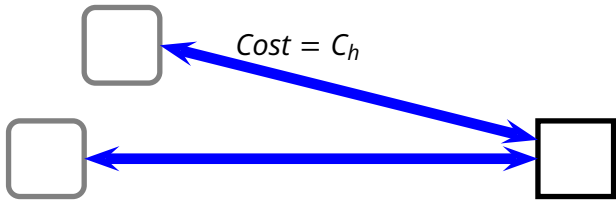
**Example**



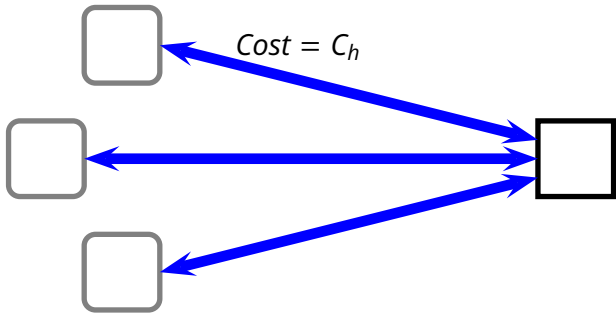
# Example



# Example

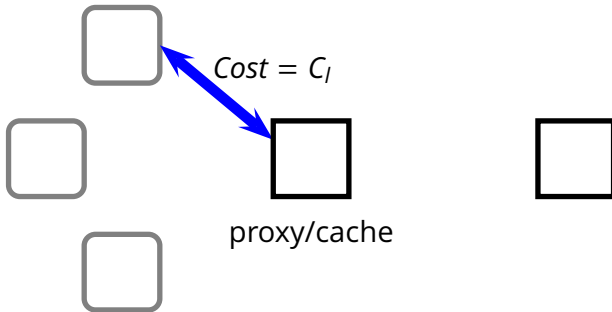


# Example

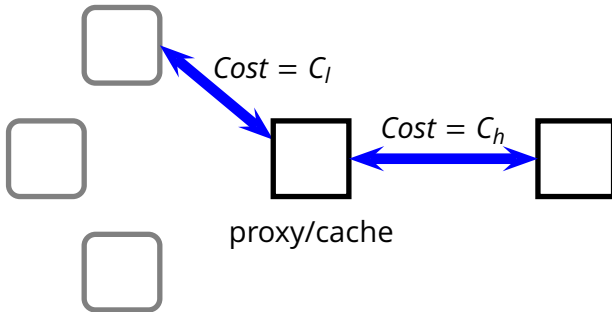




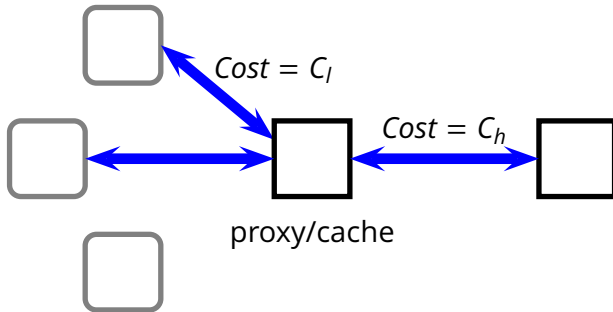
- Without proxy/cache: *total cost* =  $3C_h$



- Without proxy/cache: *total cost* =  $3C_h$

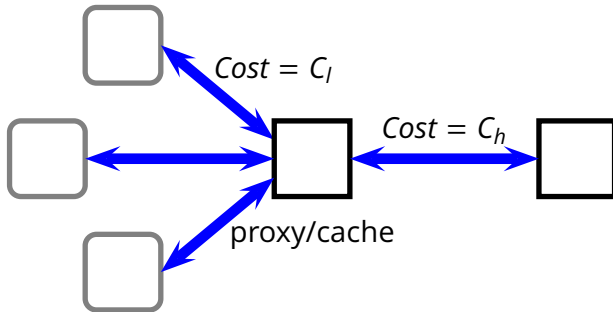


- Without proxy/cache: *total cost* =  $3C_h$

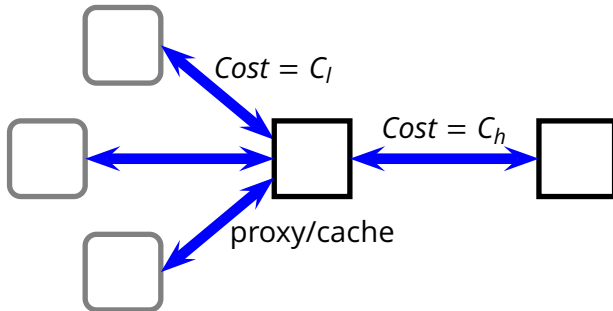


- Without proxy/cache: *total cost* =  $3C_h$





- Without proxy/cache: *total cost* =  $3C_h$



- Without proxy/cache: *total cost* =  $3C_h$
- With proxy/cache: *total cost* =  $C_h + 3C_l$

- A client request goes to a *proxy* (cache) server

- A client request goes to a *proxy* (cache) server
- The proxy may
  1. forward the request to the *origin* server, thereby acting as a client
  2. get the response from the origin server
  3. possibly store (cache) the object
  4. forward the response back to the client

- A client request goes to a *proxy* (cache) server
- The proxy may
  1. forward the request to the *origin* server, thereby acting as a client
  2. get the response from the origin server
  3. possibly store (cache) the object
  4. forward the response back to the client
- The proxy may
  1. respond immediately to the client, possibly using a cached object

- Benefits of the proxy/cache architecture

- Benefits of the proxy/cache architecture
  - ▶ performance: reduced latency

- Benefits of the proxy/cache architecture
  - ▶ performance: reduced latency
  - ▶ performance: reduced network traffic



- Benefits of the proxy/cache architecture
  - ▶ performance: reduced latency
  - ▶ performance: reduced network traffic
  - ▶ security: privacy, the server sees the proxy as a client

- Benefits of the proxy/cache architecture
  - ▶ performance: reduced latency
  - ▶ performance: reduced network traffic
  - ▶ security: privacy, the server sees the proxy as a client
  - ▶ security: protection from intrusions, in combination with a firewall

- Benefits of the proxy/cache architecture
  - ▶ performance: reduced latency
  - ▶ performance: reduced network traffic
  - ▶ security: privacy, the server sees the proxy as a client
  - ▶ security: protection from intrusions, in combination with a firewall
  
- Problems

## ■ Benefits of the proxy/cache architecture

- ▶ performance: reduced latency
- ▶ performance: reduced network traffic
- ▶ security: privacy, the server sees the proxy as a client
- ▶ security: protection from intrusions, in combination with a firewall

## ■ Problems

- ▶ latency (just like any other caching system)
- ▶ complexity

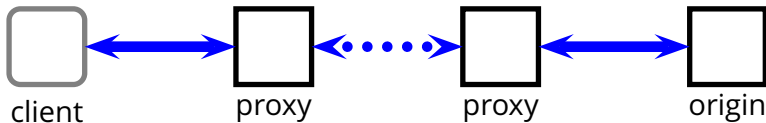
- The proxy/cache architecture is central to several features of HTTP—in fact, *it affects its overall design*

# HTTP and Caching

- The proxy/cache architecture is central to several features of HTTP—in fact, *it affects its overall design*
- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a ***request chain***

# HTTP and Caching

- The proxy/cache architecture is central to several features of HTTP—in fact, *it affects its overall design*
- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a ***request chain***



## HTTP and Caching (2)

- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a *request chain*



- HTTP defines



## HTTP and Caching (2)

- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a *request chain*



- HTTP defines
  - ▶ how protocol versions are handled on the request chain

## HTTP and Caching (2)

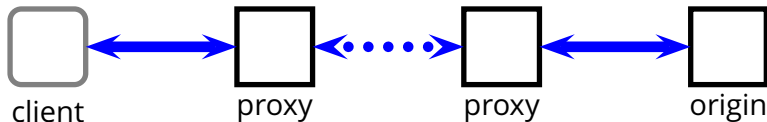
- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a ***request chain***



- HTTP defines
  - ▶ how protocol versions are handled on the request chain
  - ▶ how each method must be handled w.r.t. the request chain
    - ▶ e.g., responses to OPTIONS requests are not cacheable; 302 responses are only cacheable if indicated by a Cache-Control or Expires header field

## HTTP and Caching (2)

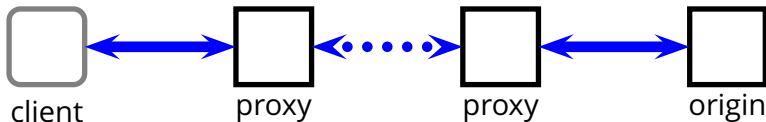
- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a *request chain*



- HTTP defines
  - ▶ how protocol versions are handled on the request chain
  - ▶ how each method must be handled w.r.t. the request chain
    - ▶ e.g., responses to OPTIONS requests are not cacheable; 302 responses are only cacheable if indicated by a Cache-Control or Expires header field
  - ▶ specific authentication mechanisms for proxies

## HTTP and Caching (2)

- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through a *request chain*



- HTTP defines
  - ▶ how protocol versions are handled on the request chain
  - ▶ how each method must be handled w.r.t. the request chain
    - ▶ e.g., responses to OPTIONS requests are not cacheable; 302 responses are only cacheable if indicated by a Cache-Control or Expires header field
  - ▶ specific authentication mechanisms for proxies
  - ▶ a lot of headers to control caching along the request chain

# HTTP and Caching (3)

## HTTP and Caching (3)

- Cached pages may become stale

## HTTP and Caching (3)

- Cached pages may become stale
- A HEAD request could be used to see if an object has been updated, in which case the cache can be invalidated
  - ▶ but how does a proxy decide that it is okay to respond to a client with a cached object?

## HTTP and Caching (3)

- Cached pages may become stale
- A HEAD request could be used to see if an object has been updated, in which case the cache can be invalidated
  - ▶ but how does a proxy decide that it is okay to respond to a client with a cached object?
- Servers specify explicit expiration times using either the Expires header, or the max-age directive of the Cache-Control header



## HTTP and Caching (3)

- Cached pages may become stale
- A HEAD request could be used to see if an object has been updated, in which case the cache can be invalidated
  - ▶ but how does a proxy decide that it is okay to respond to a client with a cached object?
- Servers specify explicit expiration times using either the Expires header, or the max-age directive of the Cache-Control header
- A client or proxy can use a *conditional* GET by including a If-Modified-Since header

# Cache-Control in Requests

# Cache-Control in Requests

- ```
GET /carzaniga/index.html HTTP/1.1  
Host: www.inf.usi.ch  
Cache-Control: no-cache
```

# Cache-Control in Requests

- ```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Cache-Control: no-cache
```

“Please, do not use cached objects!”

- ▶ proxies must go to the origin server

# Cache-Control in Requests

- ```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Cache-Control: no-cache
```

“Please, do not use cached objects!”

- ▶ proxies must go to the origin server

- ```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Cache-Control: max-age=20
```

# Cache-Control in Requests

- ```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Cache-Control: no-cache
```

“Please, do not use cached objects!”

- ▶ proxies must go to the origin server

- ```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Cache-Control: max-age=20
```

“Please, give me a cached object only if it is less than 20 seconds old”

# Cache-Control in Replies

# Cache-Control in Replies



```
HTTP/1.1 200 Ok  
Cache-Control: no-cache  
  
. . .
```



## Cache-Control in Replies



```
HTTP/1.1 200 Ok  
Cache-Control: no-cache  
  
. . .
```

“Please, do not cache this objects!”

# Cache-Control in Replies

- HTTP/1.1 200 Ok  
Cache-Control: no-cache  
  
. . .

“Please, do not cache this objects!”

- HTTP/1.1 200 Ok  
Cache-Control: max-age=100; must-revalidate  
  
. . .

## Cache-Control in Replies

- HTTP/1.1 200 Ok  
Cache-Control: no-cache  
...

“Please, do not cache this objects!”

- HTTP/1.1 200 Ok  
Cache-Control: max-age=100; must-revalidate  
...

“You *may* use this object up to 100 seconds from now. After that, you *must* revalidate the object.”

- ▶ without the must-revalidate directive, a client may use a stale object

# Sessions in HTTP

- HTTP is a stateless protocol

- HTTP is a stateless protocol
  - ▶ so how do you implement a “shopping cart”?

- HTTP is a stateless protocol
  - ▶ so how do you implement a “shopping cart”?
- HTTP provides the means for higher-level applications to maintain *stateful sessions* (see RFC 2109)

- HTTP is a stateless protocol
  - ▶ so how do you implement a “shopping cart”?
- HTTP provides the means for higher-level applications to maintain *stateful sessions* (see RFC 2109)
- Set-Cookie header
  - ▶ sent within an HTTP response, from the server to the client
  - ▶ tells the client to store the given “cookie” as a session identifier for that site



- HTTP is a stateless protocol
  - ▶ so how do you implement a “shopping cart”?
- HTTP provides the means for higher-level applications to maintain *stateful sessions* (see RFC 2109)
- Set-Cookie header
  - ▶ sent within an HTTP response, from the server to the client
  - ▶ tells the client to store the given “cookie” as a session identifier for that site
- Cookie header
  - ▶ sent within an HTTP request, from the client to the server
  - ▶ tells the server that the request belongs to the given session

# Example

web  
browser  
*hispeed.ch*

web  
server  
*blah.com*

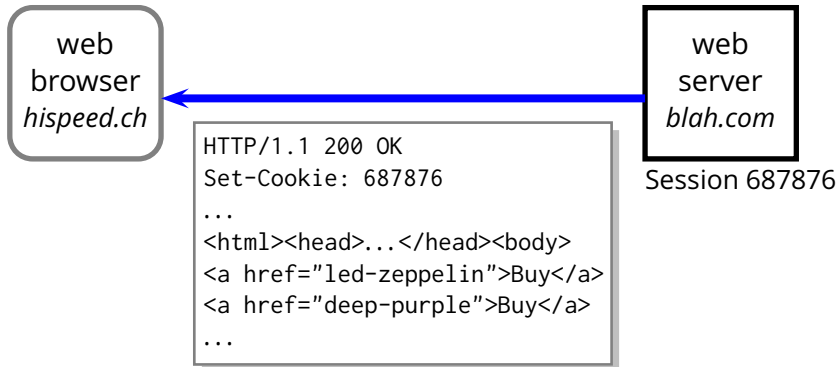
# Example



# Example



# Example



# Example



# Example



blah.com: 687876



Session 687876

# Example







blah.com: 687876



Session 687876  
Led Zeppelin

# Example



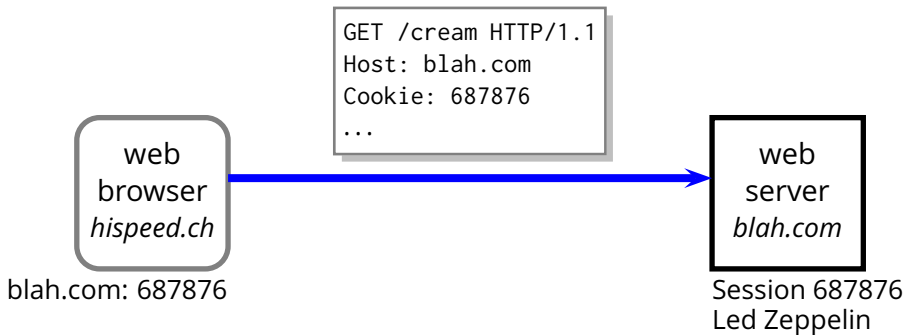


blah.com: 687876



Session 687876  
Led Zeppelin

# Example



# Example



blah.com: 687876



Session 687876  
Led Zeppelin  
Cream

# Example



# Example

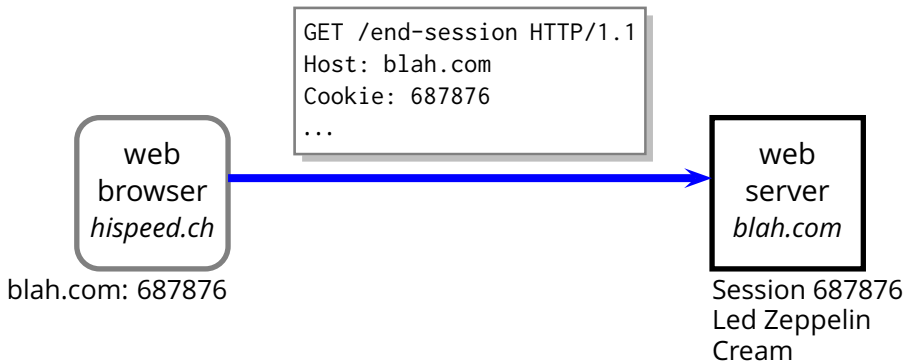


blah.com: 687876



Session 687876  
Led Zeppelin  
Cream

# Example





# Example

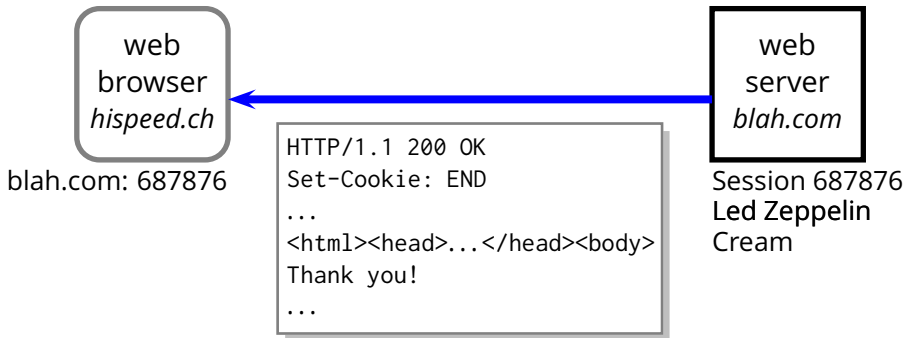


blah.com: 687876



Session 687876  
Led Zeppelin  
Cream

# Example





blah.com: END



Session 687876  
Led Zeppelin  
Cream

# Cookies and User Privacy

- A “session” identifies the actions of a user

# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!

# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!
- In our example, we can infer that user n. 687876...

# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!
- In our example, we can infer that user n. 687876...
  - ▶ likes rock-blues music from the sixties and seventies

# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!
- In our example, we can infer that user n. 687876...
  - ▶ likes rock-blues music from the sixties and seventies
  - ▶ lives in Switzerland



# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!
- In our example, we can infer that user n. 687876...
  - ▶ likes rock-blues music from the sixties and seventies
  - ▶ lives in Switzerland
  - ▶ ...

# Cookies and User Privacy

- A “session” identifies the actions of a user
- Web sites may use cookies to compile and collect user profiles
  - ▶ and obviously they do exactly that!
- In our example, we can infer that user n. 687876...
  - ▶ likes rock-blues music from the sixties and seventies
  - ▶ lives in Switzerland
  - ▶ ...
- If user n. 687876 buys something on line with a credit card, then he or she would also be immediately indentified