

[illegible]

Generic Definition

- Software function (tool, resource, data...)
- Accessed via the network (remote, deployed, @)
- Offered to other software units (M2M)
- Platform- and language-independent
- Can be described and advertised
- 2 roles:
 - Service requester = client
 - Service provider = server

What is an API?

- a means of exposing business/enterprise resources via the Internet to external or internal software consumers
 - Well-defined interface: contract
 - Easily accessible by third parties
 - Use of standard protocol(s)

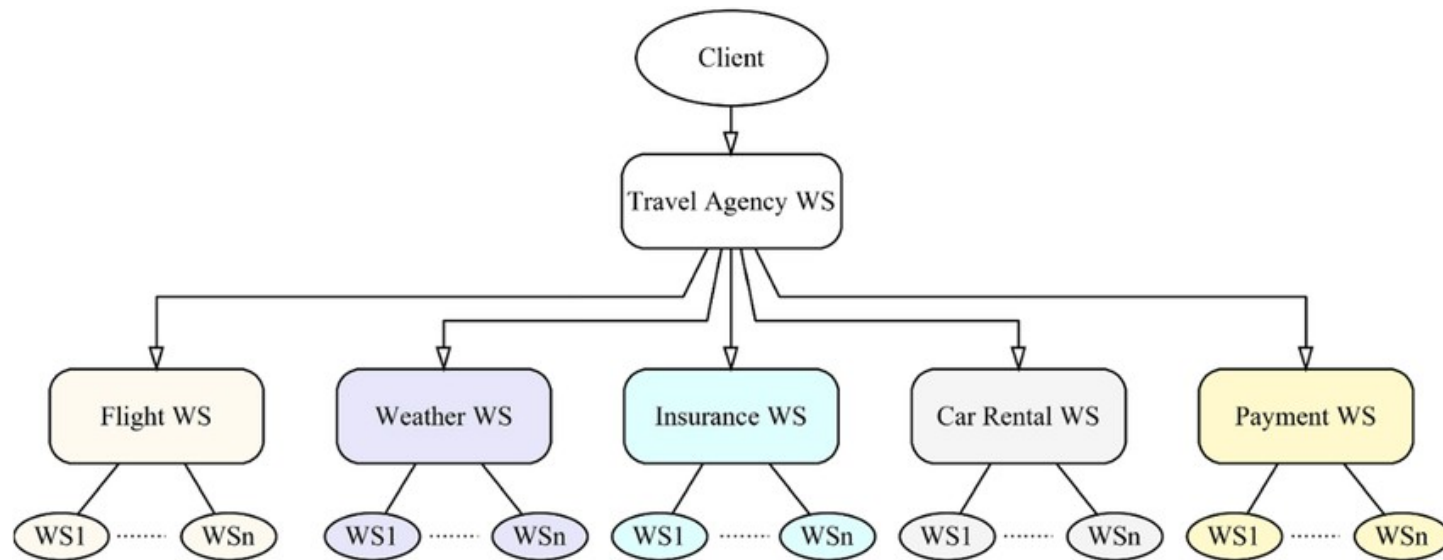
≈ Web Service

Usage

- Services are used as software libraries to build applications
- 2 contexts:
 - External services
 - Internal services

External Services (Partner or Public Services)

- open to the partners of the organization (B2B, B2C)

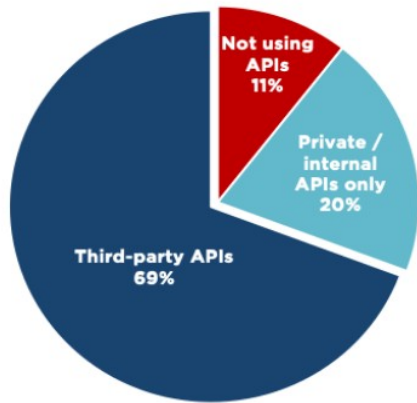


API becomes more of a priority than UI

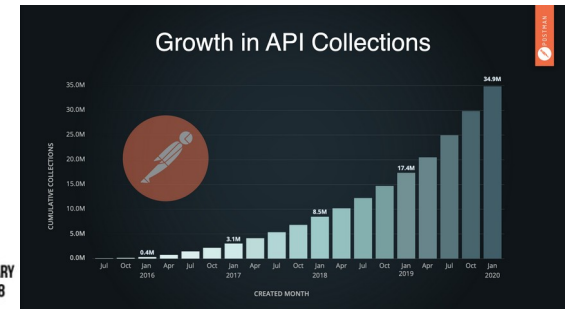
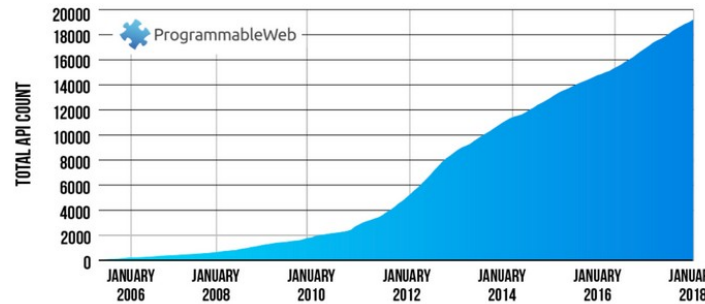
Nearly 90% of developers use APIs

% of developers (Q3 2020 n=15,299)

<https://nordicapis.com/apis-have-taken-over-software-development/>



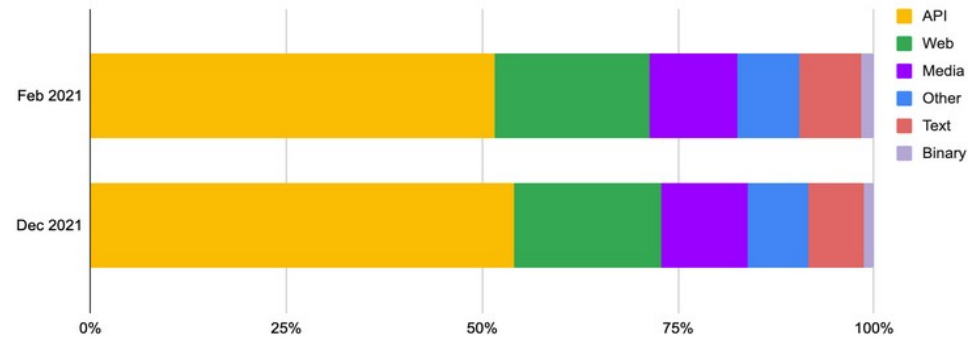
Source: SlashData Developer Economics survey 19th edition / DATA



Cloudflare traffic: API use in 2021

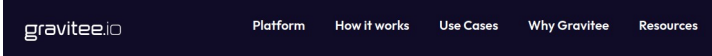
<https://blog.cloudflare.com/landscape-of-api-traffic/>

Traffic composition by content type



Programmatic access is considered at least as vital as human access, if not more so.

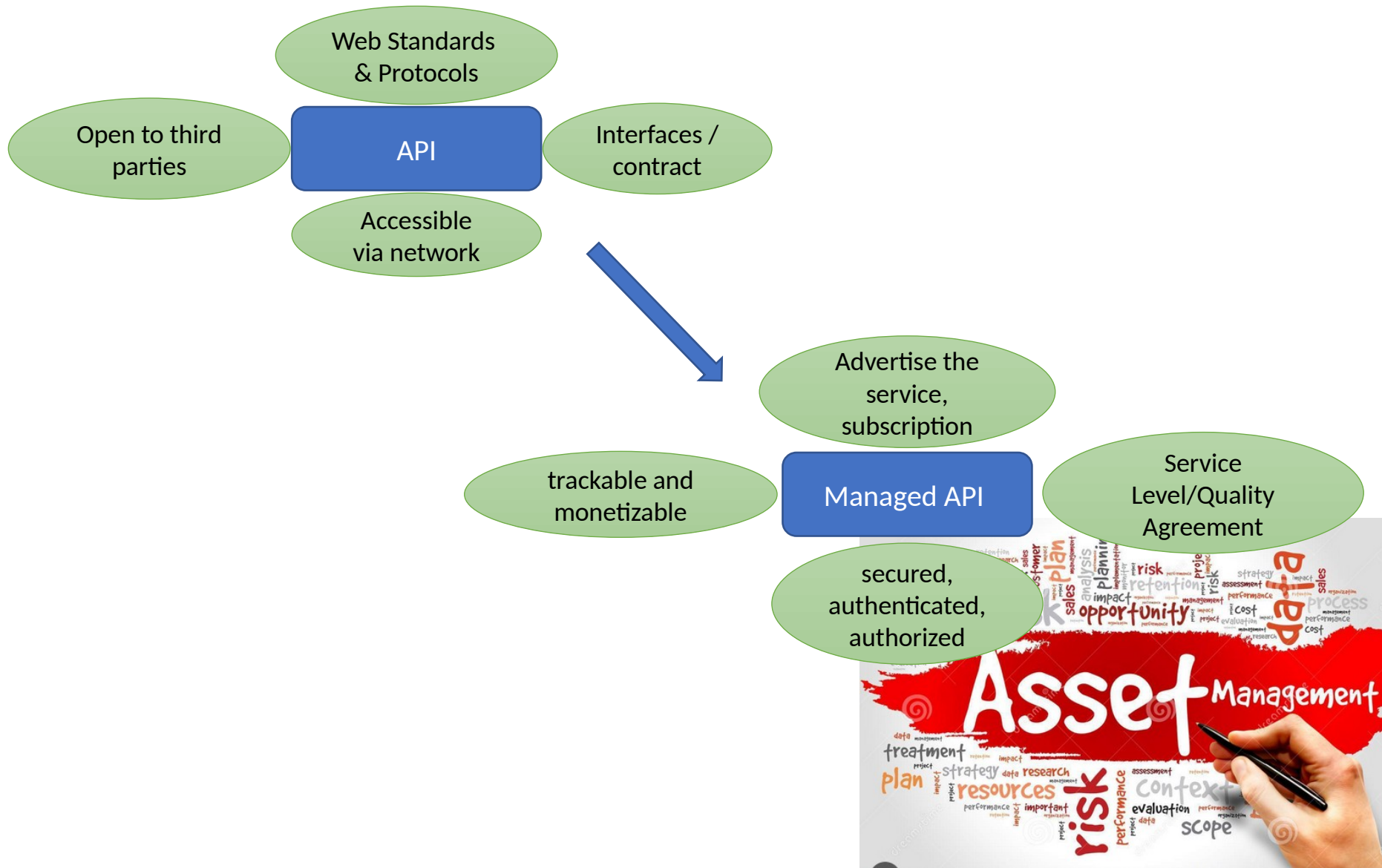
<https://www.postman.com/state-of-api/>



API-First As The Norm

Adopting an API-first strategy will be increasingly common in the future. In fact, Postman's 2021 found that 39.2% of teams have already designed and defined APIs and schema before they ever

From Basic WS to Managed API



Benefits of Web Services from the Client's Point of View

- Take advantage of third-party data or programs without having to:
 - develop, test, update and maintain code
 - acquire and maintain a hosting infrastructure
- Easily compose services and replace one component by an alternative

Trade-offs for the Client

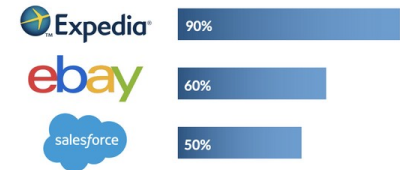
Developers lost control of the services and the services are remote →

- A service might be temporally unavailable
- Performances might become poor
- Data of the client can get lost, divulged, corrupted...
- A service might not longer be maintained
- The service fee might increase
- ...

From the Service Provider's Point of View

- Benefits
 - Increases revenue
 - Extends customer reach
 - New form of marketing : B2D “business to developer”
 - Stimulates innovation
- Risks
 - Decreases ad revenue
 - No more control on the final user's experience

Percentage of Revenue Generated Through APIs

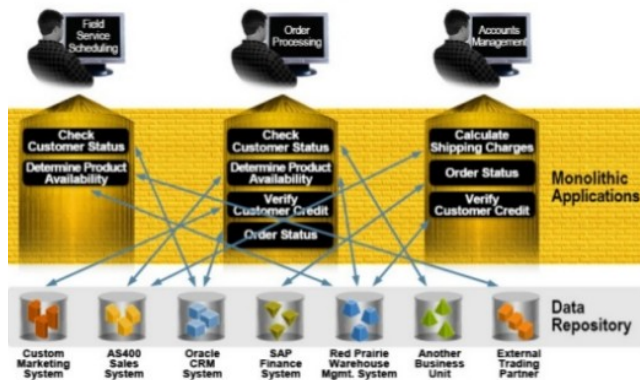


Source: Harvard Business Review, The Strategic Value of APIs, 2015.

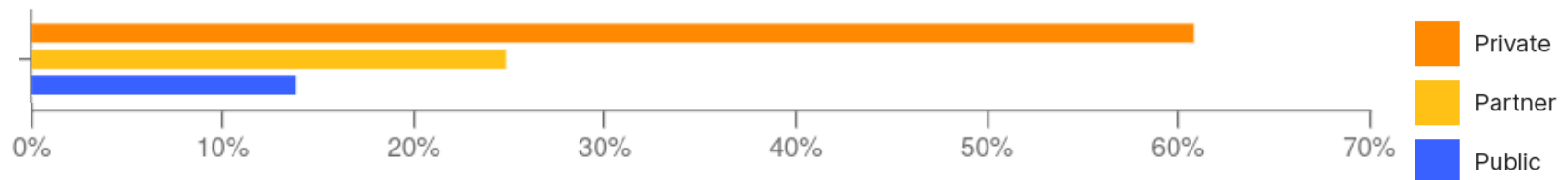
Internal Services (Private Services)

- access restricted to the organization

Monolithic Systems



Reuse Services via Re-composition



Internal Services: an injunction!

- Jeff Bezos's mandate (2002)



1. All teams will expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

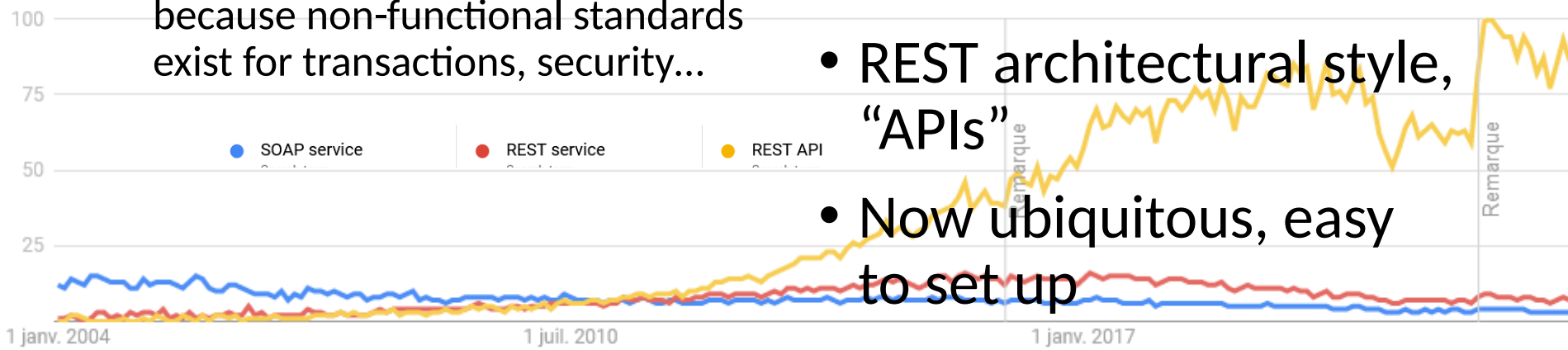
2 “Flavors” of Web Services

Process-Oriented Services

- Distributed Information Systems required middleware, RPC and Object Brokers, were poorly adapted to B2B → use web protocols for transport and XML as IDL and format
- SOAP + WSDL standards
- “first-generation” web services, still used for complex applications because non-functional standards exist for transactions, security...

Resources-Oriented Services

- From static web pages to dynamic web pages to web applications (UI = web browser, business processes are executed on the server) to web services
- REST architectural style, “APIs”
- Now ubiquitous, easy to set up



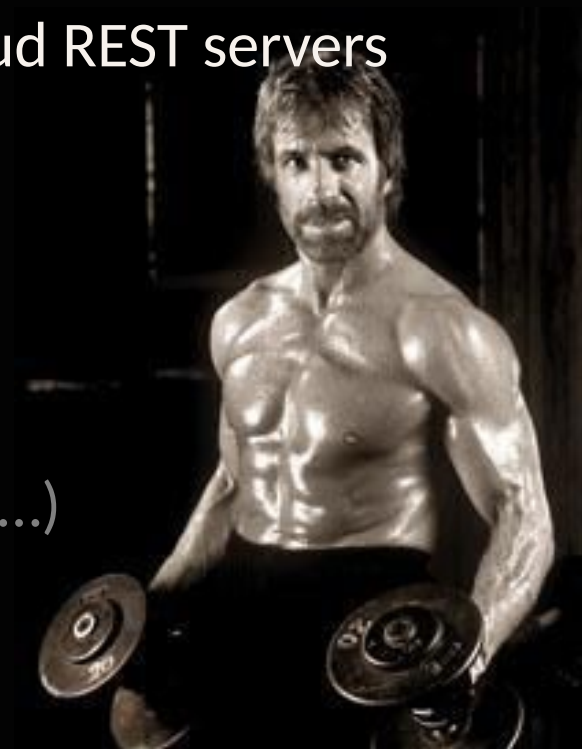
After this Course...

You will be able to design, set up and take advantage of a Service-Oriented Architecture

- find Web Services and understand their interfaces, including GraphQL
- write well-designed and documented APIs
- implement in Python and deploy on the cloud REST servers
- write Python clients
- cite several Chuck Norris's facts

Not covered:

- SOAP and WSDL
- DevOps (deployment, mock tests, load tests...)
- security
- scripted composition of services

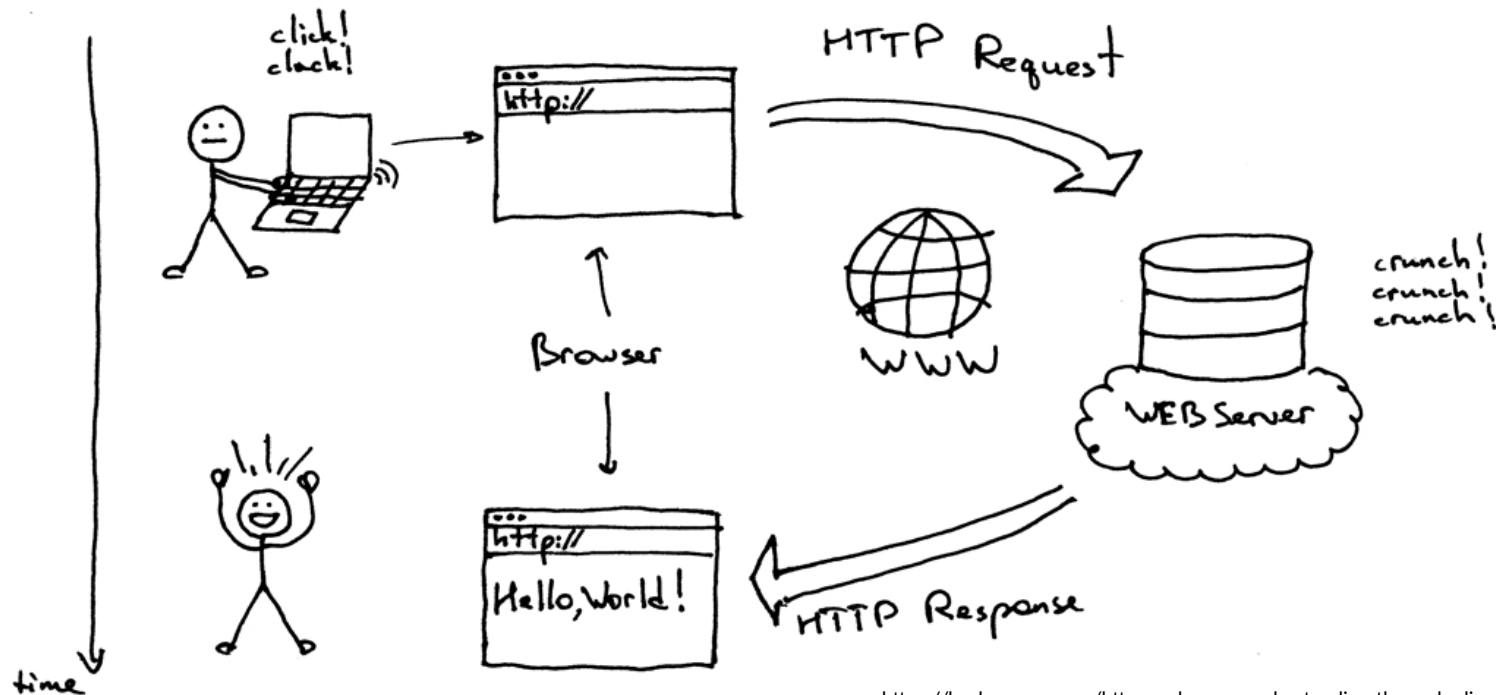


http://www.



The Web Services Protocol

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

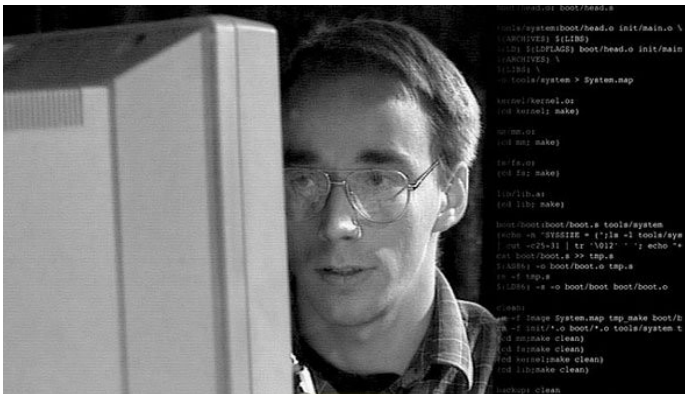


When was HTTP first specified?



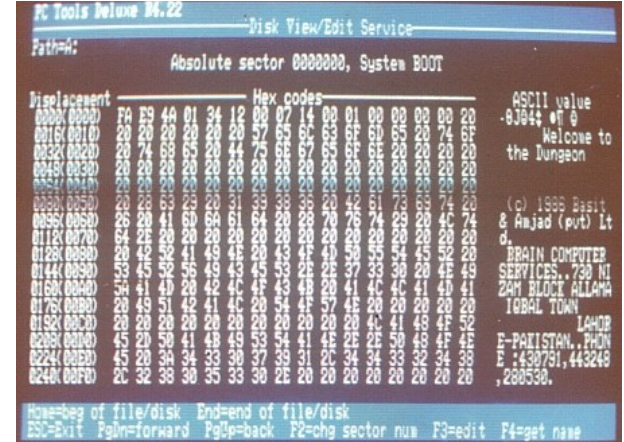
1981

(IBM PC 5150)



1991

(Linus Torvalds introduces Linux)



1986

(Brain: first computer virus for MS-DOS)

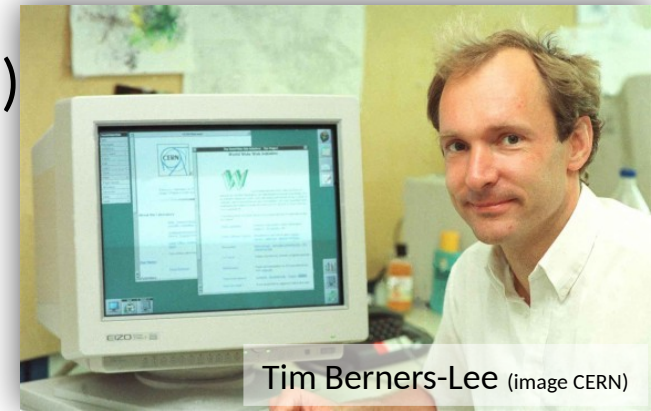


1996

(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)

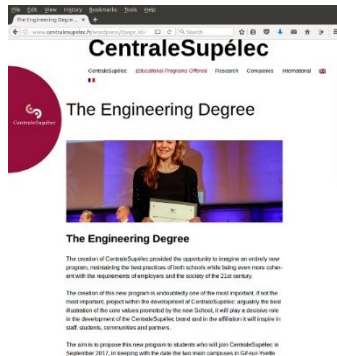


1. user clicks on hyperlink

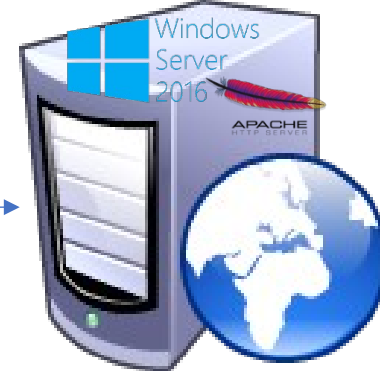
2. HTTP request message

4. HTTP response message

5. display file



web server



back-end
(database...)

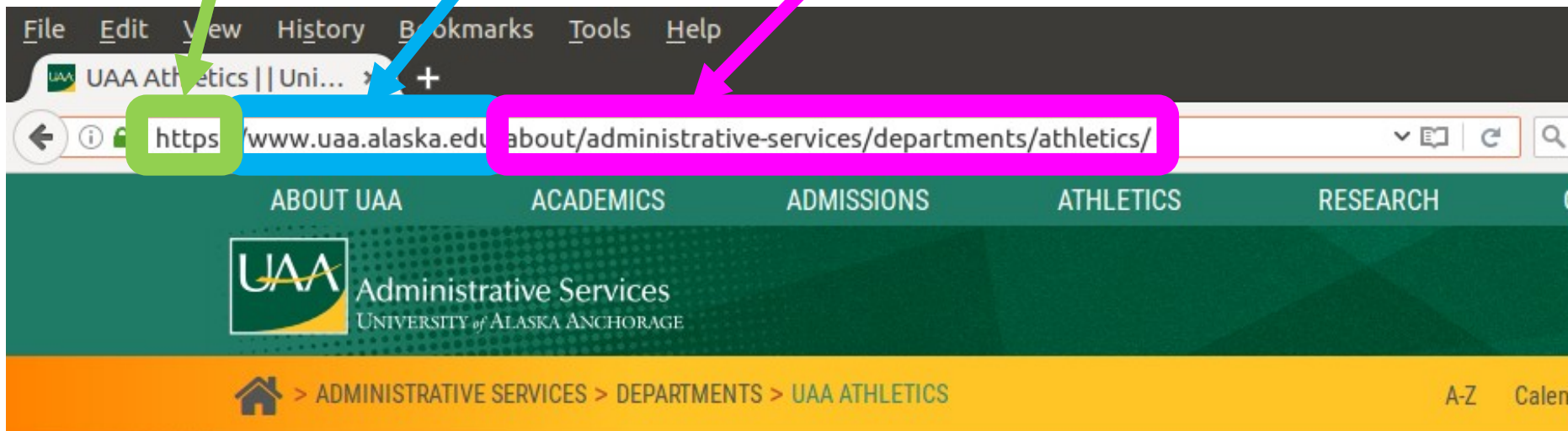


3. create or retrieve file



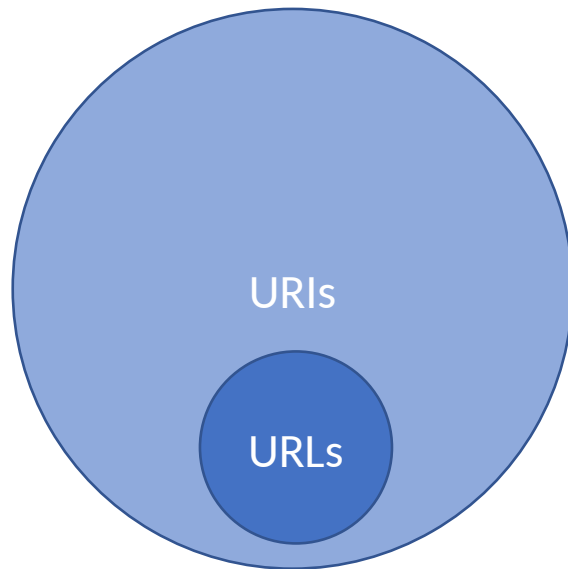
Resources = addressable files

- 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:

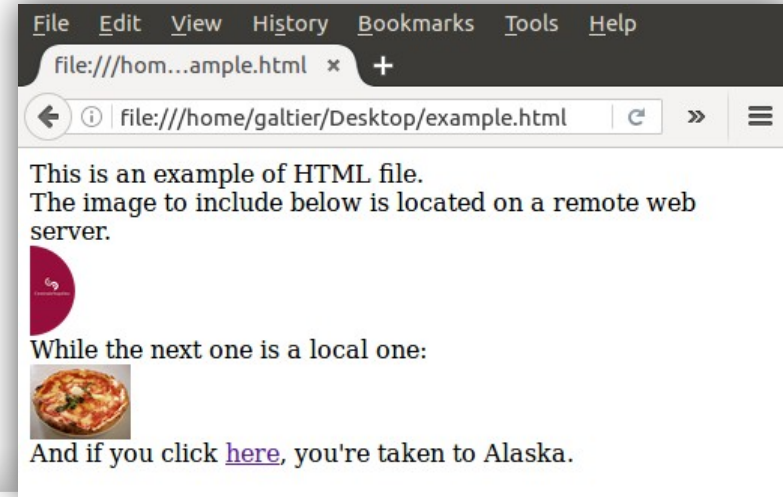
with the GPS coordinates I arrive to the right restaurant

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



```
File Edit View Search Tools Documents Help
Open [icon] Save

<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>
</br>
While the next one is a local one:</br>
</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>
```

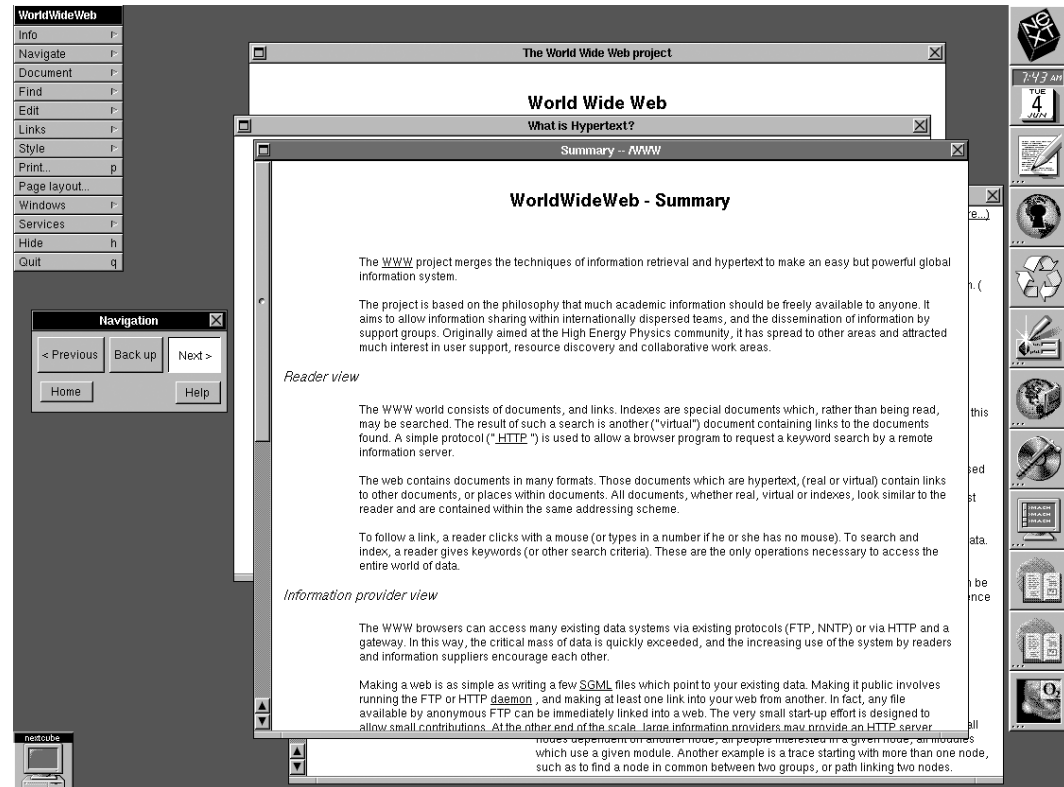
HTTP Versions

HTTP/0.9

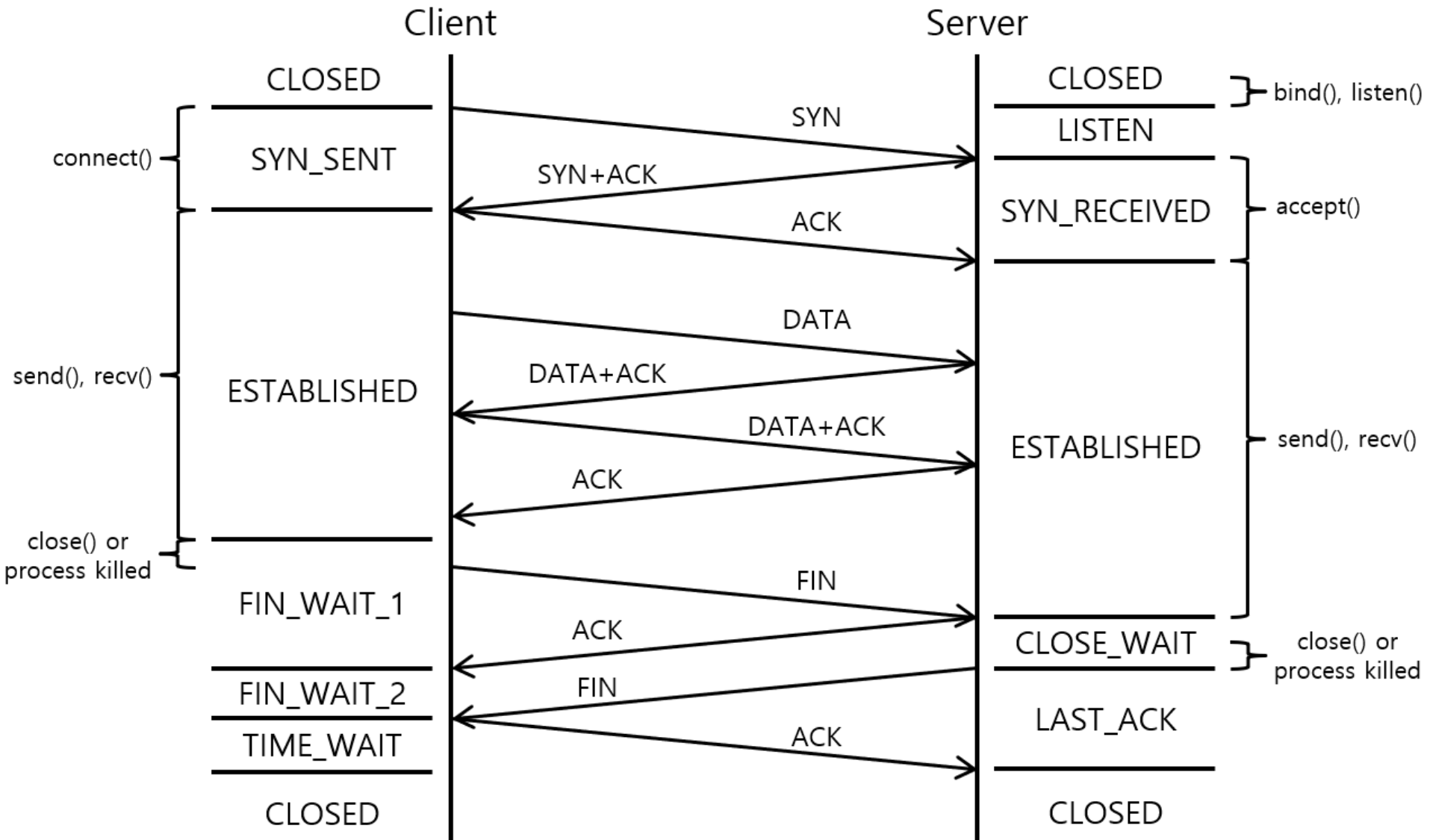


1991

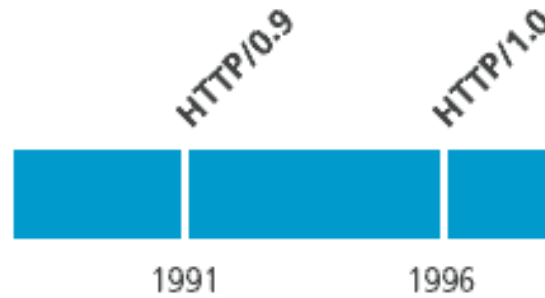
- 1991 – v0.9
 - First documented version
 - First web browser



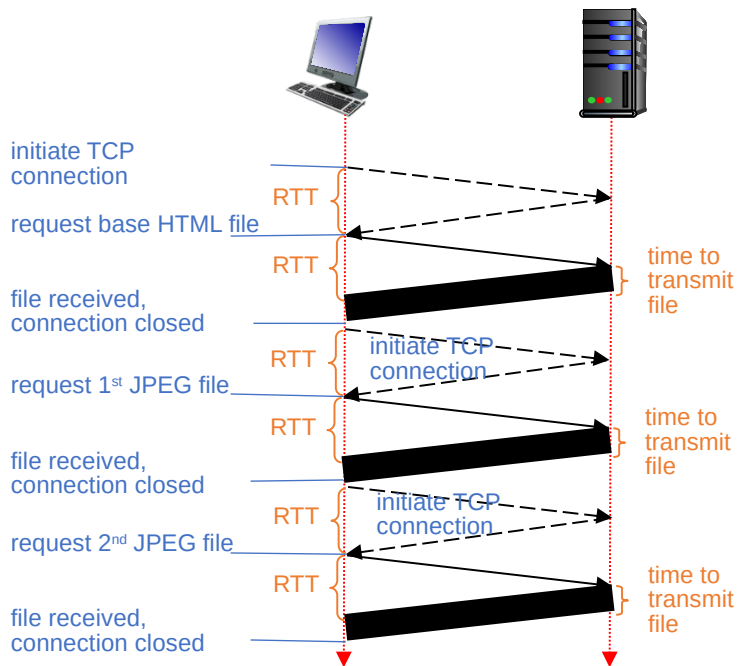
Side note on TCP 3-way handshake



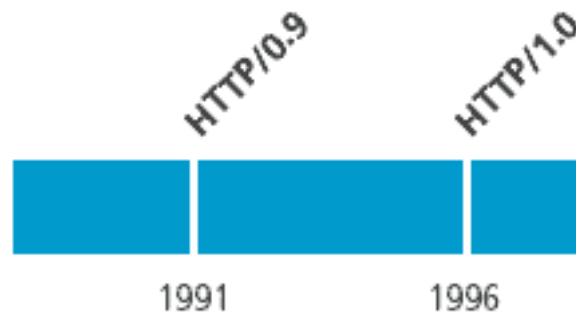
HTTP Versions



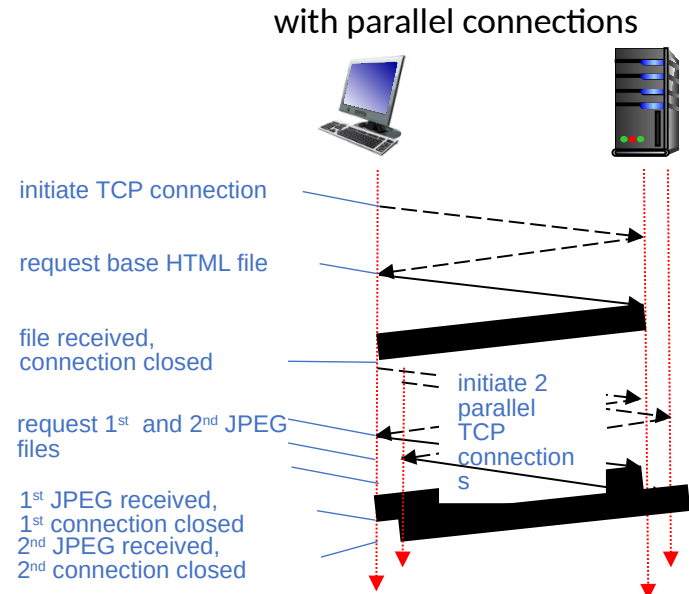
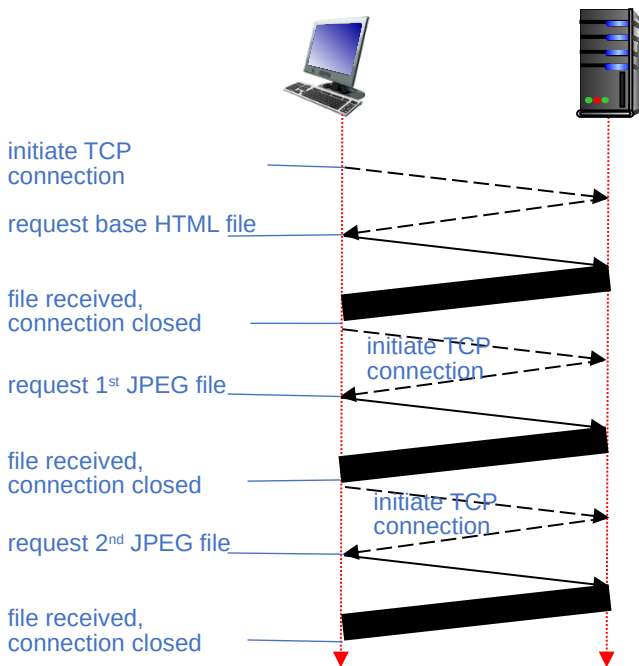
- 1996 – v1.0
 - One TCP connection per resource



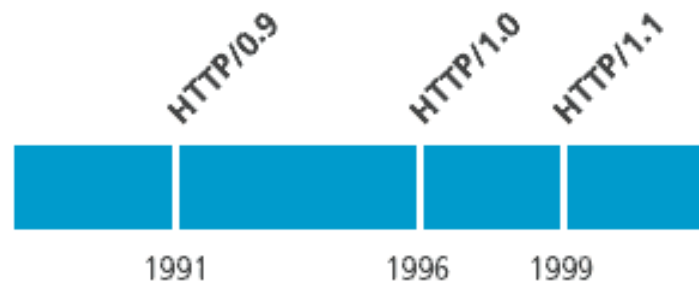
HTTP Versions



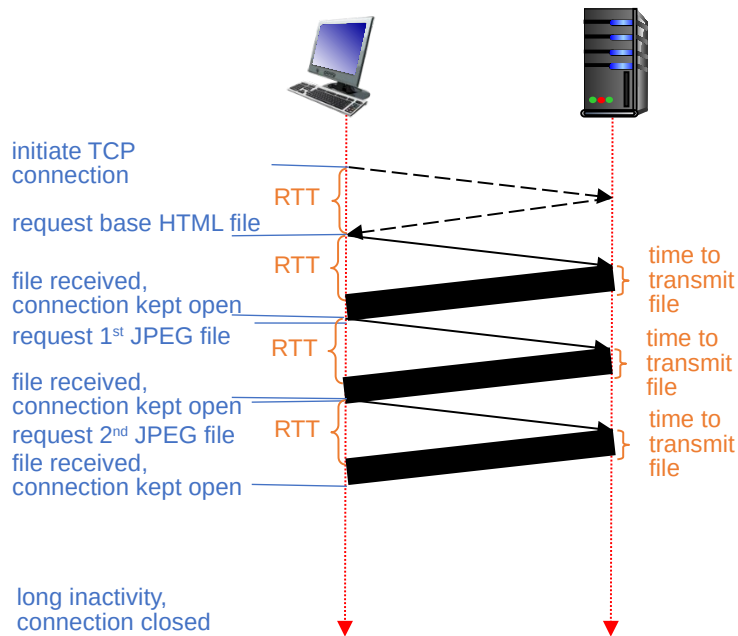
- 1996 – v1.0
 - One TCP connection per resource



HTTP Versions



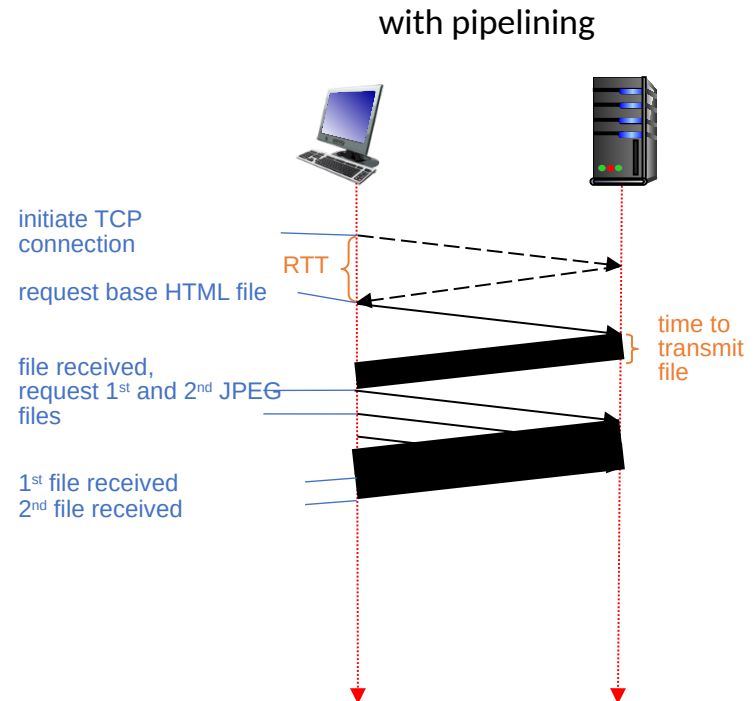
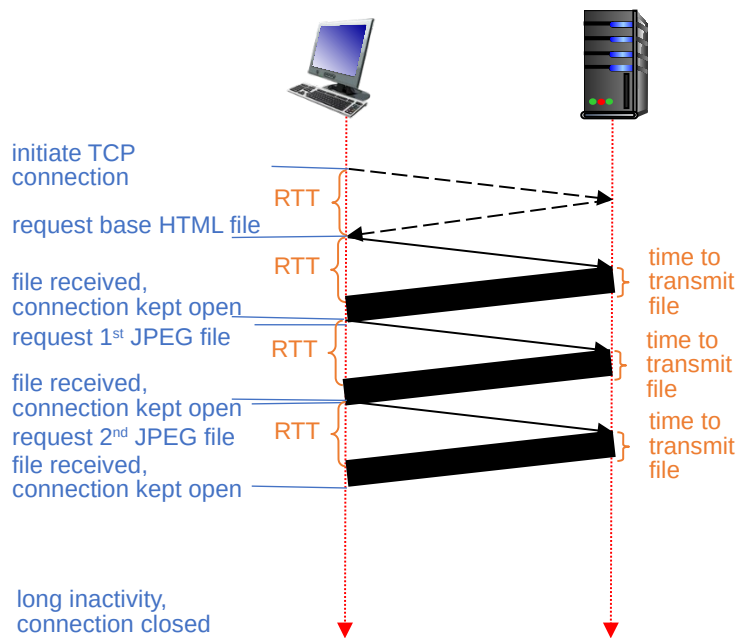
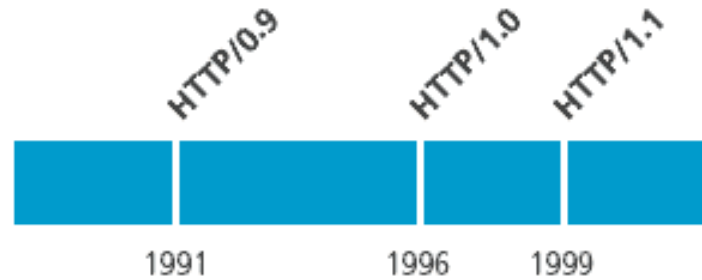
- 1999 – v1.1
 - Persistent connection



```
GET https://www.centralesupelec.fr/ HTTP/1.1
Host: www.centralesupelec.fr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

HTTP Versions

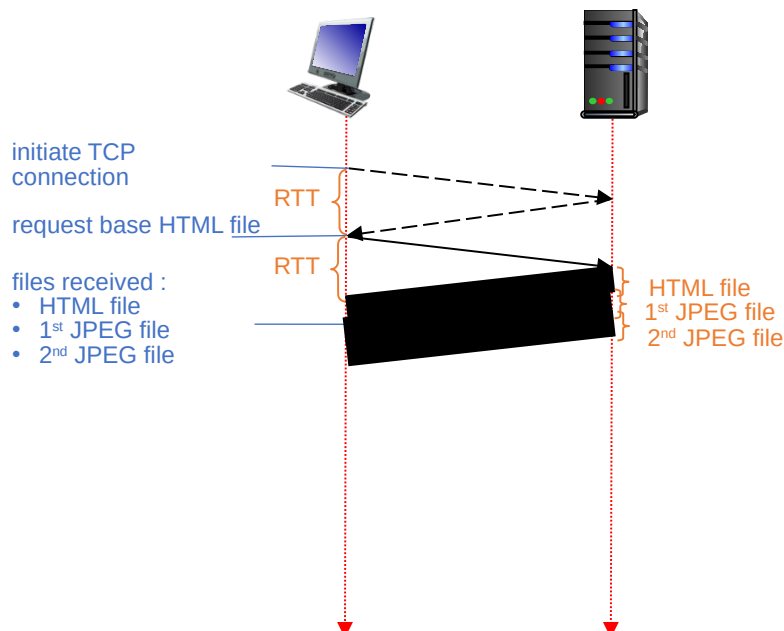
- 1999 – v1.1
 - Persistent connection



HTTP Versions



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



A screenshot of a web browser window displaying a page titled "HTTP/2: the Future ...". The browser's address bar shows the URL "https://http2.akamai.com/di". The page content includes the Akamai logo and the heading "HTTP/2 is the future of the Web, and it is here!". Below this, a subheading reads "Your browser supports HTTP/2!". A paragraph states: "This is a demo of HTTP/2's impact on your download of many small tiles making up the Akamai Spinning Globe." The page then compares HTTP/1.1 and HTTP/2 performance. For HTTP/1.1, the latency is 27ms and the load time is 3.23s. For HTTP/2, the latency is 27ms and the load time is 1.17s. Below the performance metrics are two images of the Akamai Spinning Globe, which is a globe composed of many small tiles. The globe is shown in two states: one with many tiles missing (HTTP/1.1) and one with all tiles present (HTTP/2). The page footer mentions "Demo content inspired by Galaxie".

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).

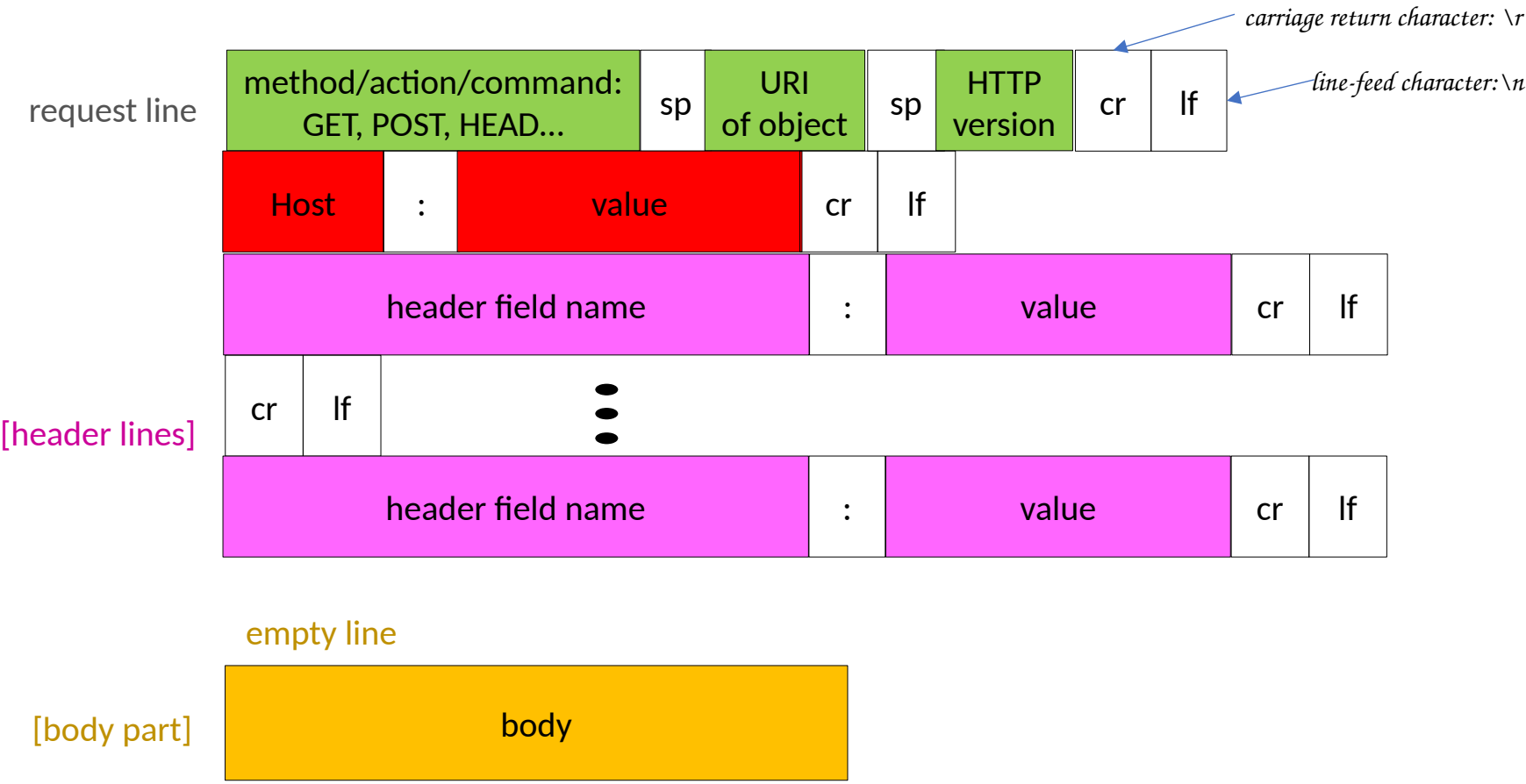
HTTP Messages

- 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



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\r\nlastname=Doe

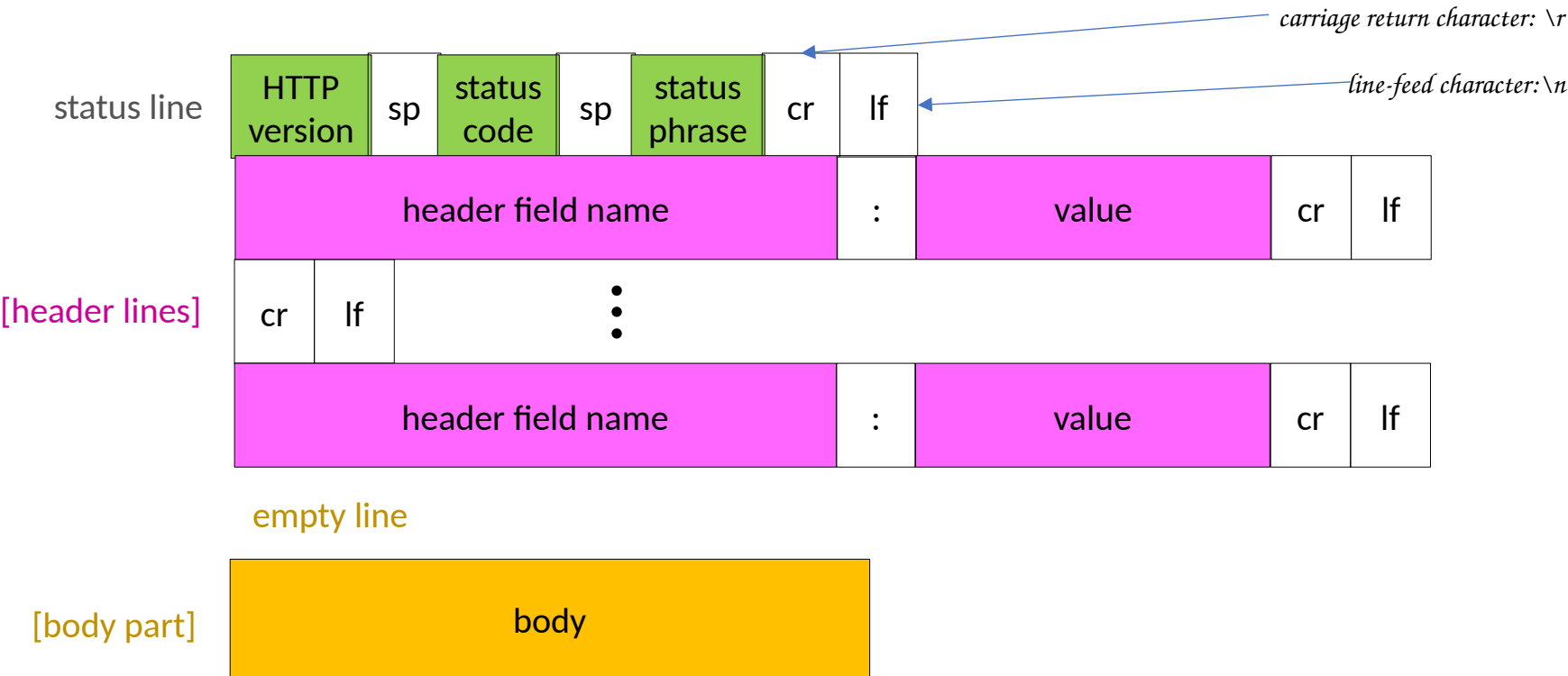
Request Parameters

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

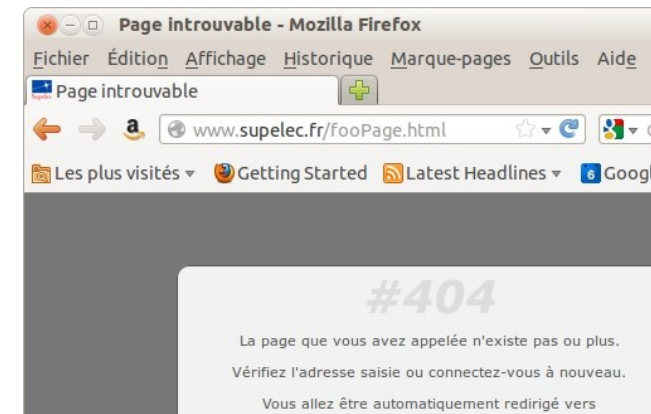
GET /products?priceMin=10&priceMax=40

HTTP Response Format



Status 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



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.....(l....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...l.;.T.X}....N.r .<d...#.....S..
..#Oa. ...V..EPj..G...A..D.K...Z1..c.h,b.4..b.3...l.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?



Hyper Text Coffee Pot Control Protocol

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.

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]

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](#):

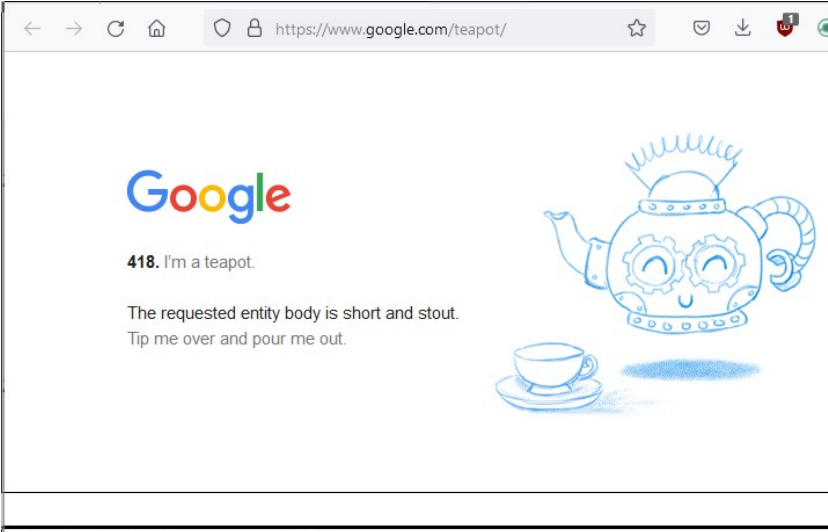
406 Not Acceptable	The HTCPCP server is unable to provide the requested addition for some reason; the response should indicate a list of available additions. The RFC observes that "In practice, most automated coffee pots cannot currently provide additions."
418 I'm a teapot	The HTCPCP server is a teapot ; 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]}

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



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
```

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

typed out request

```
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
```

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
& 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)

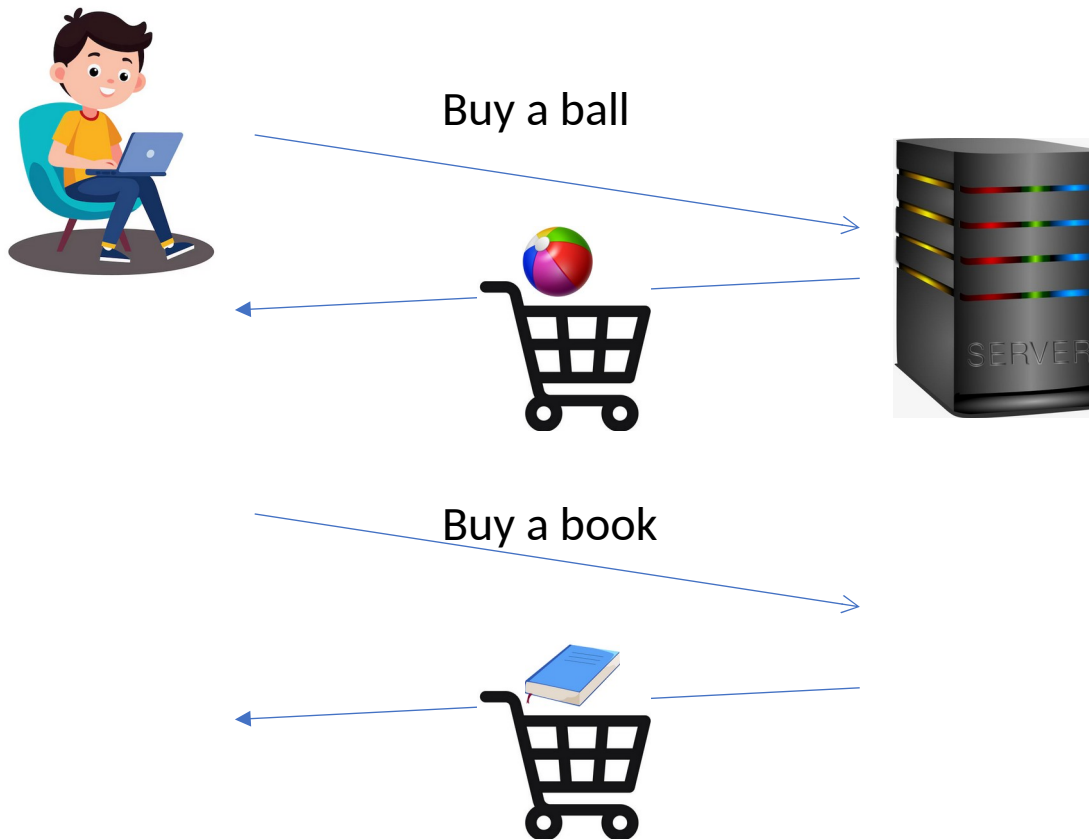
received response

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.

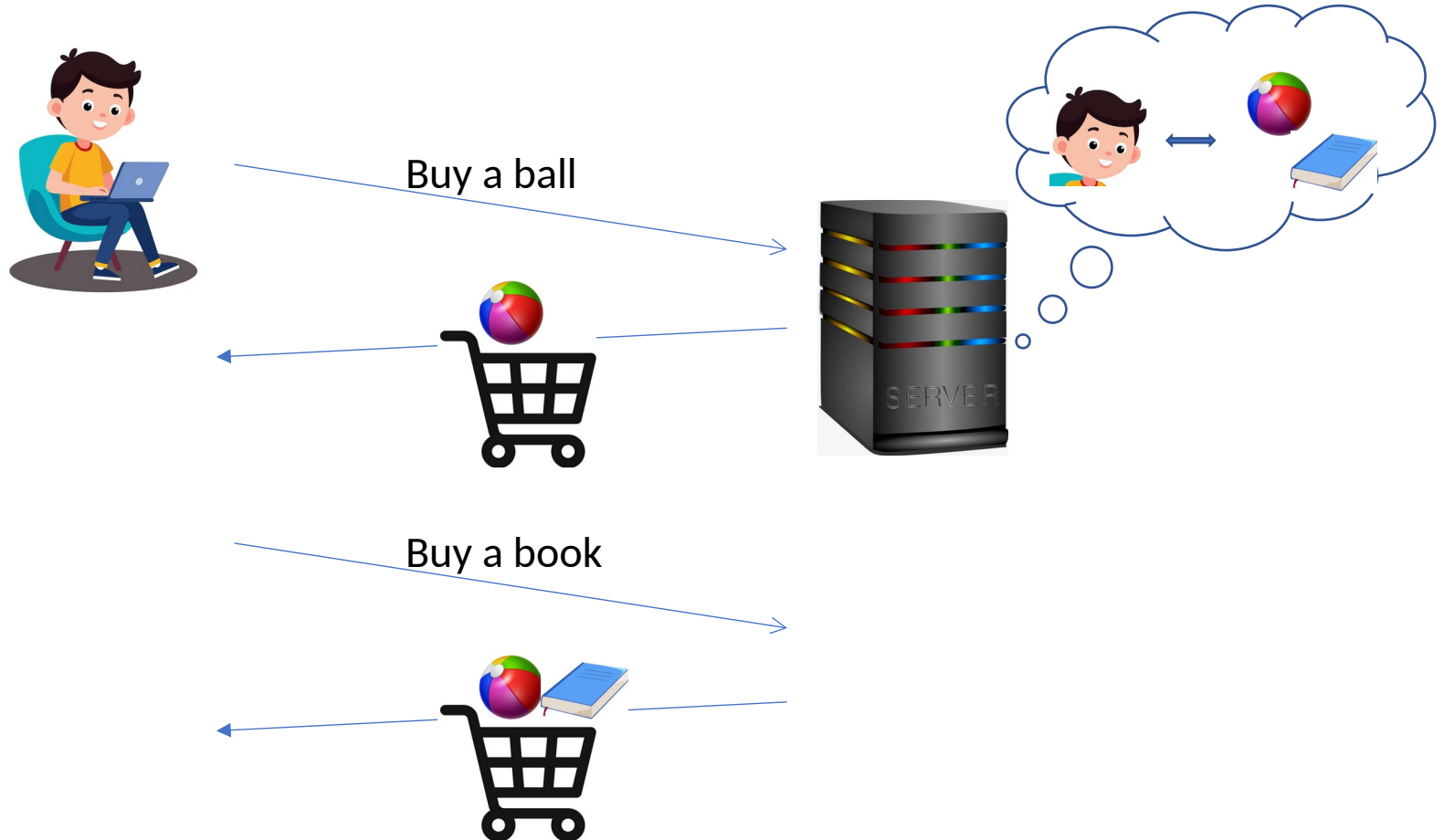
Stateless HTTP Server

- Server doesn't keep track of interaction with the client



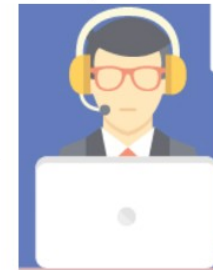
Stateful Non-HTTP Server

- Server keeps track of interaction with the client



Stateful Service is not Fault Tolerant

Hi,
My name is Bob.
I have XYZ problem. This was a long conversation with lot of details, but I hope now you got all the details you need from me to process my request.



Hi,
My name is Bob.
The previous call got disconnected abruptly due to some issue. But since I passed on all the details already to the previous agent, I hope you can retrieve all the details to process my request.



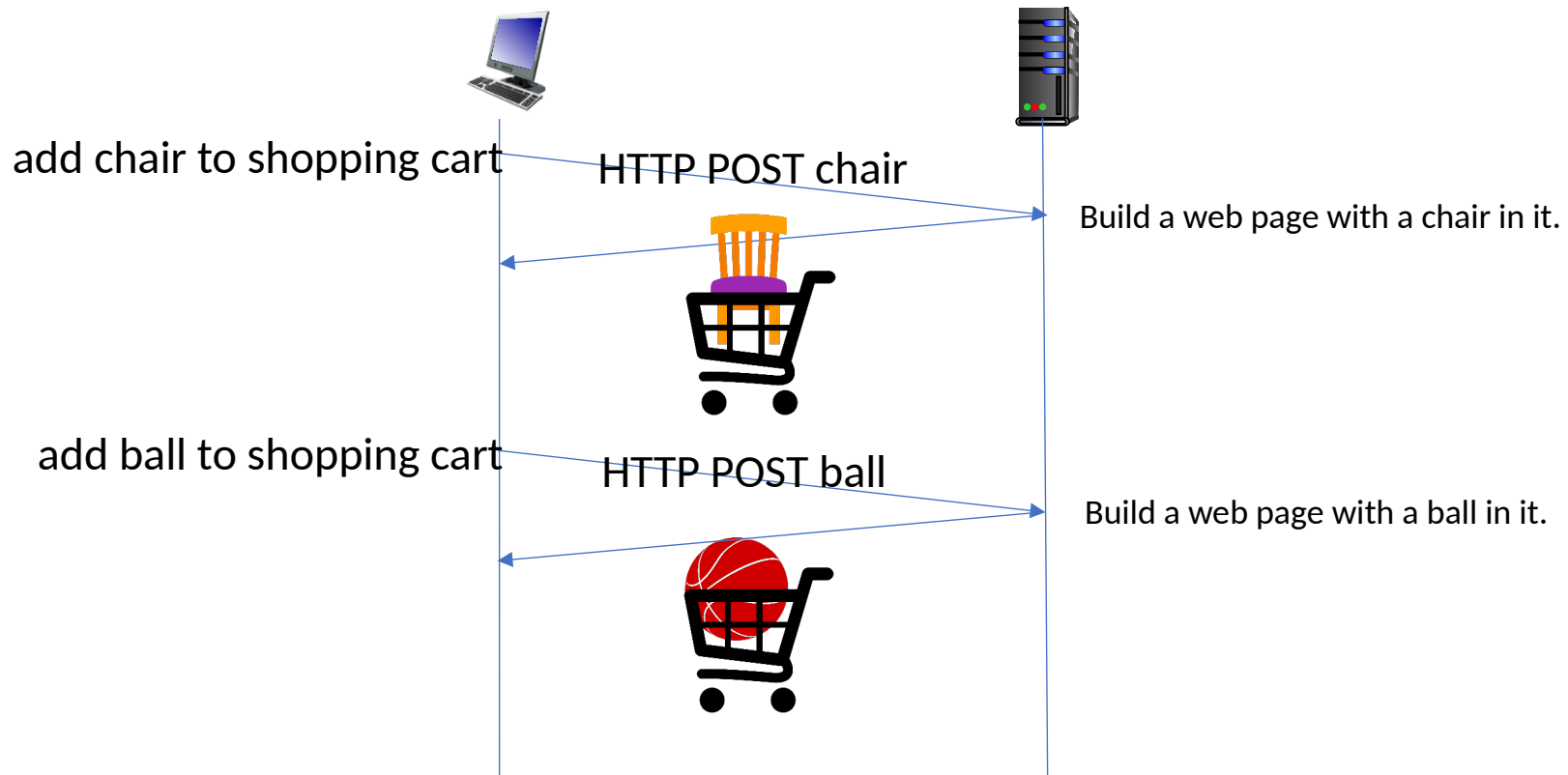
Sorry, sir. Your information was lost due to the disruption in the previous call so I do not have access to any of it and hence cannot process your request unless we go through all of it again from scratch. Sorry again for the inconvenience.



Stateful Service doesn't Scale

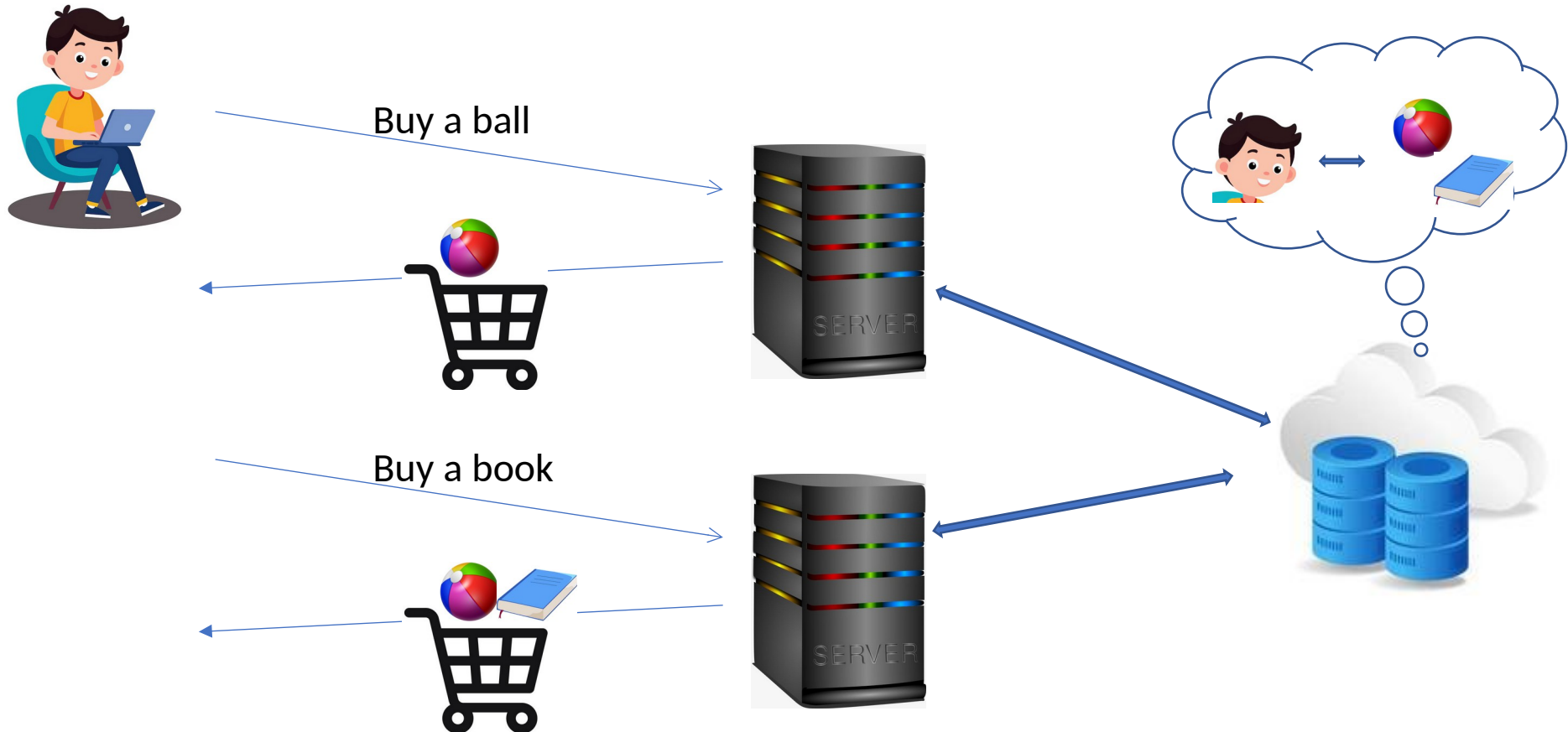
- All clients previously handed by a tele assistant need to be handled by the same assistant when returning to the web server even if this assistant is busy while other are idle

But sometimes Stateless is not what we want



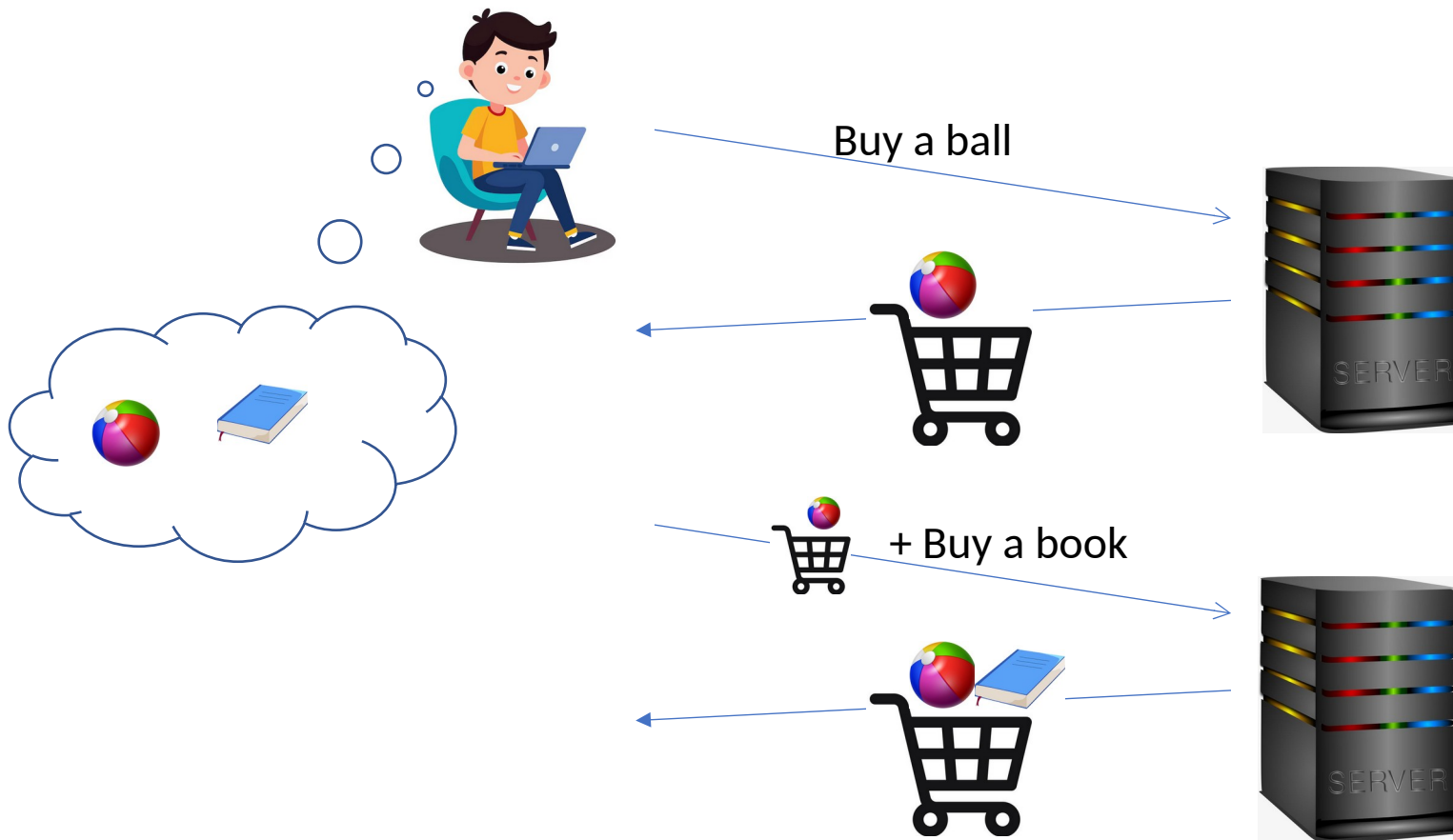
Stateful Service with Stateless Server

- Solution 1: State saved on external storage



Stateful Service with Stateless Server

- Solution 2: State provided by client



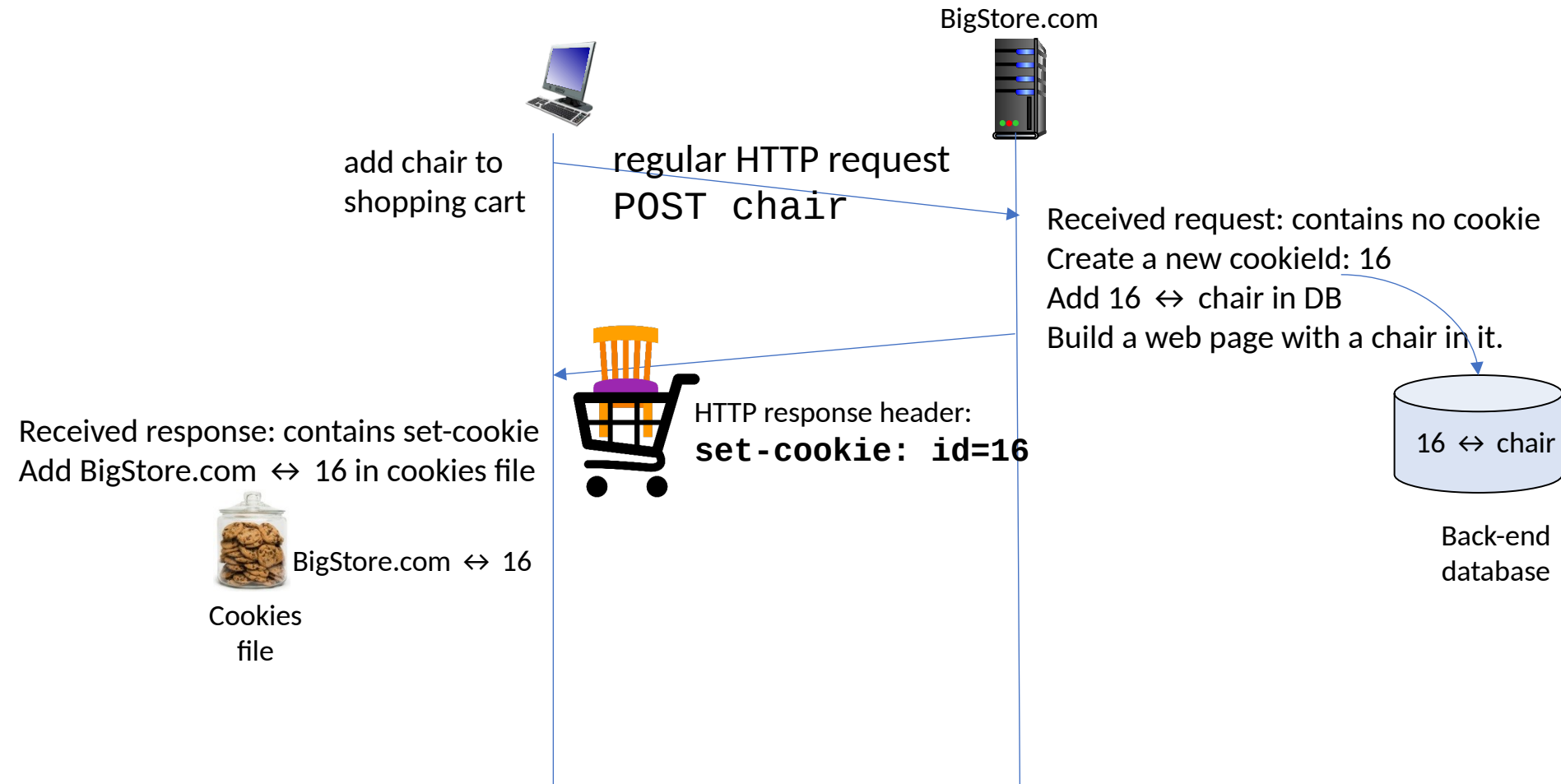
Cookies

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

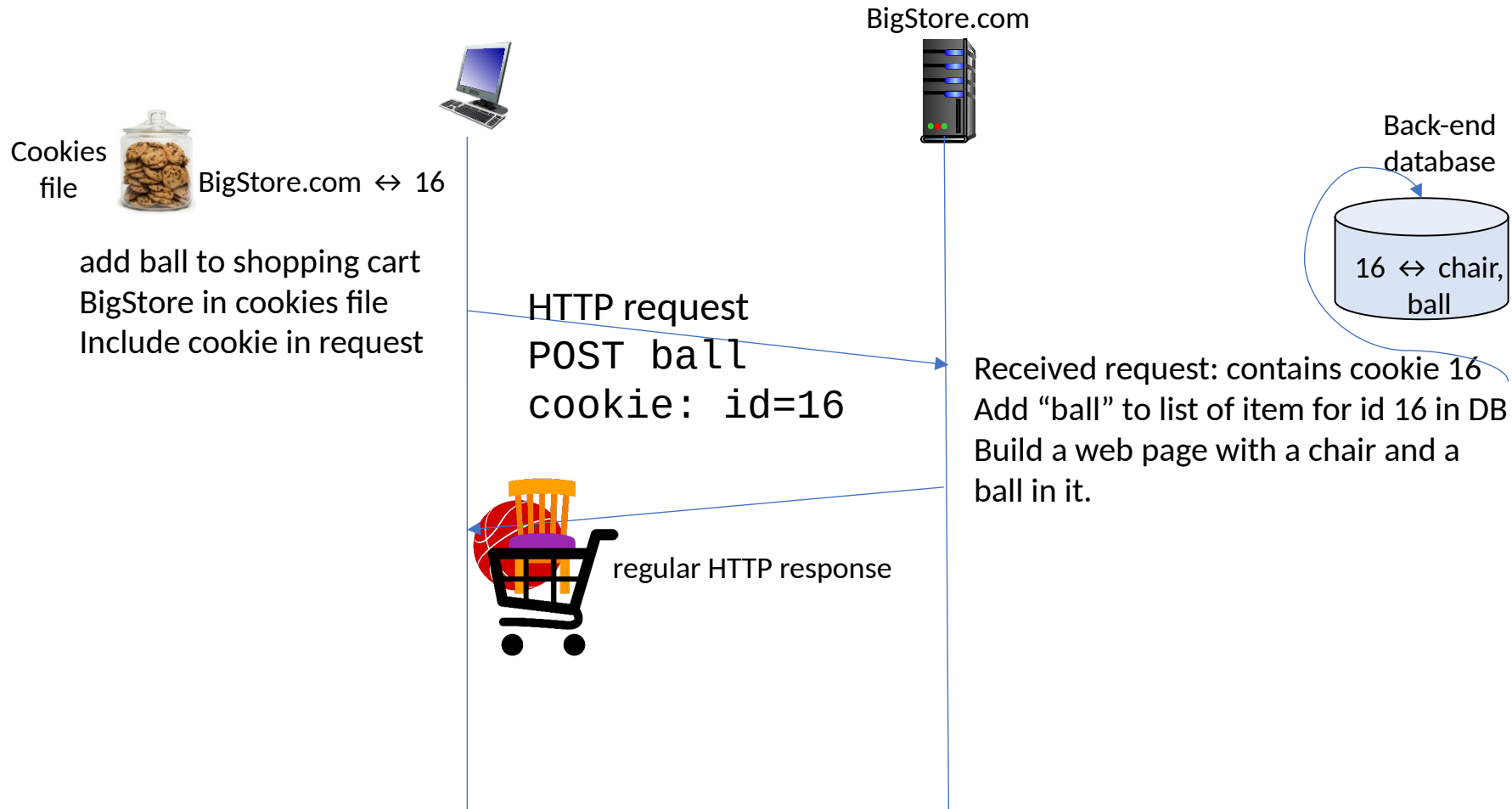
*Shhhhh... they're in here
somewhere. Dad's always
talking about the cookies
in his computer.*



Cookie Example



Cookie Example



Uses

create a user session layer on top of stateless HTTP

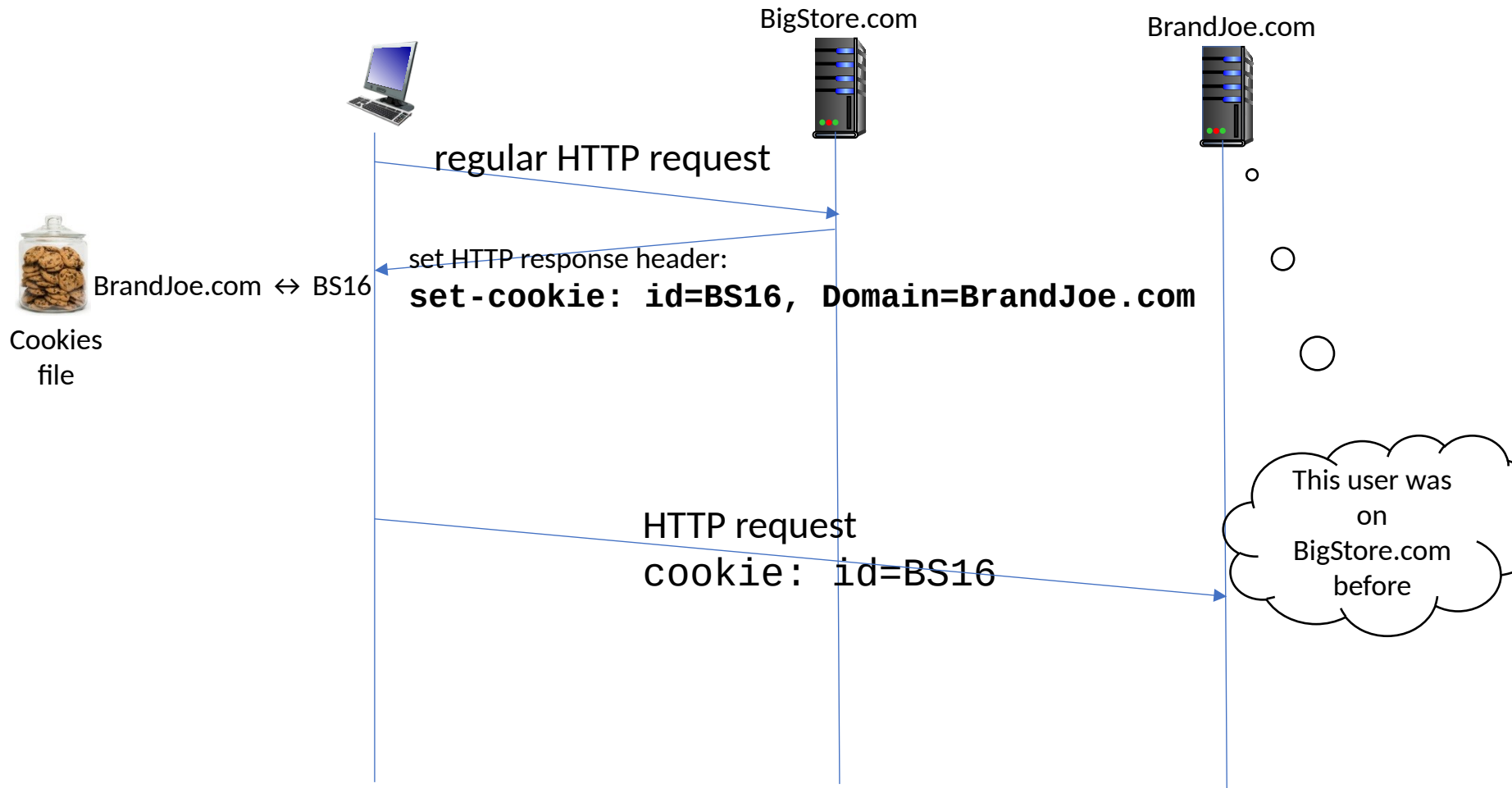
- content adaptation (recommendation based on previous visits etc.).
- shopping carts (e-business)
- session definition at application layer (Web mail)
- authorization
- ...

Suspicious

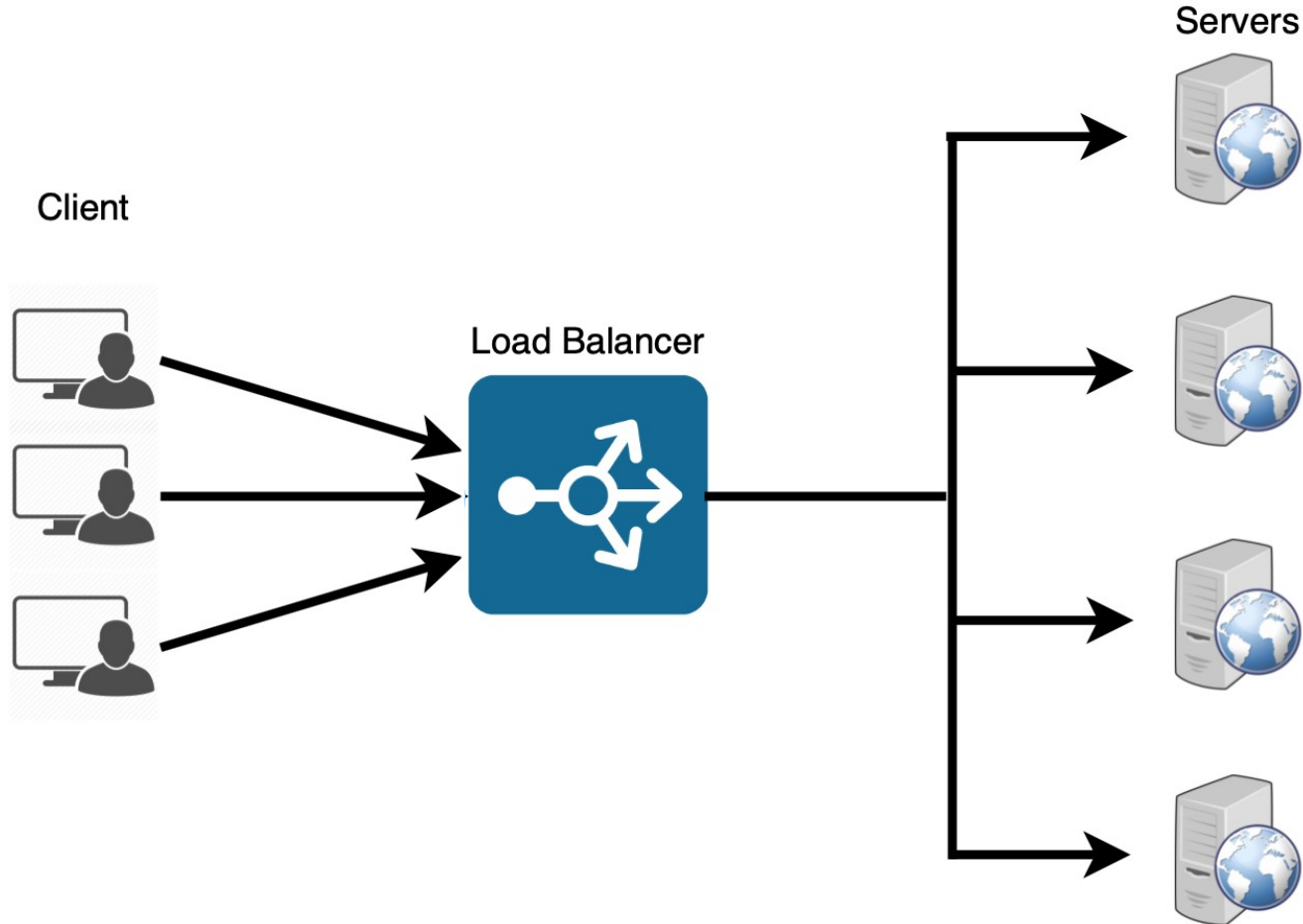
- Invasion of privacy



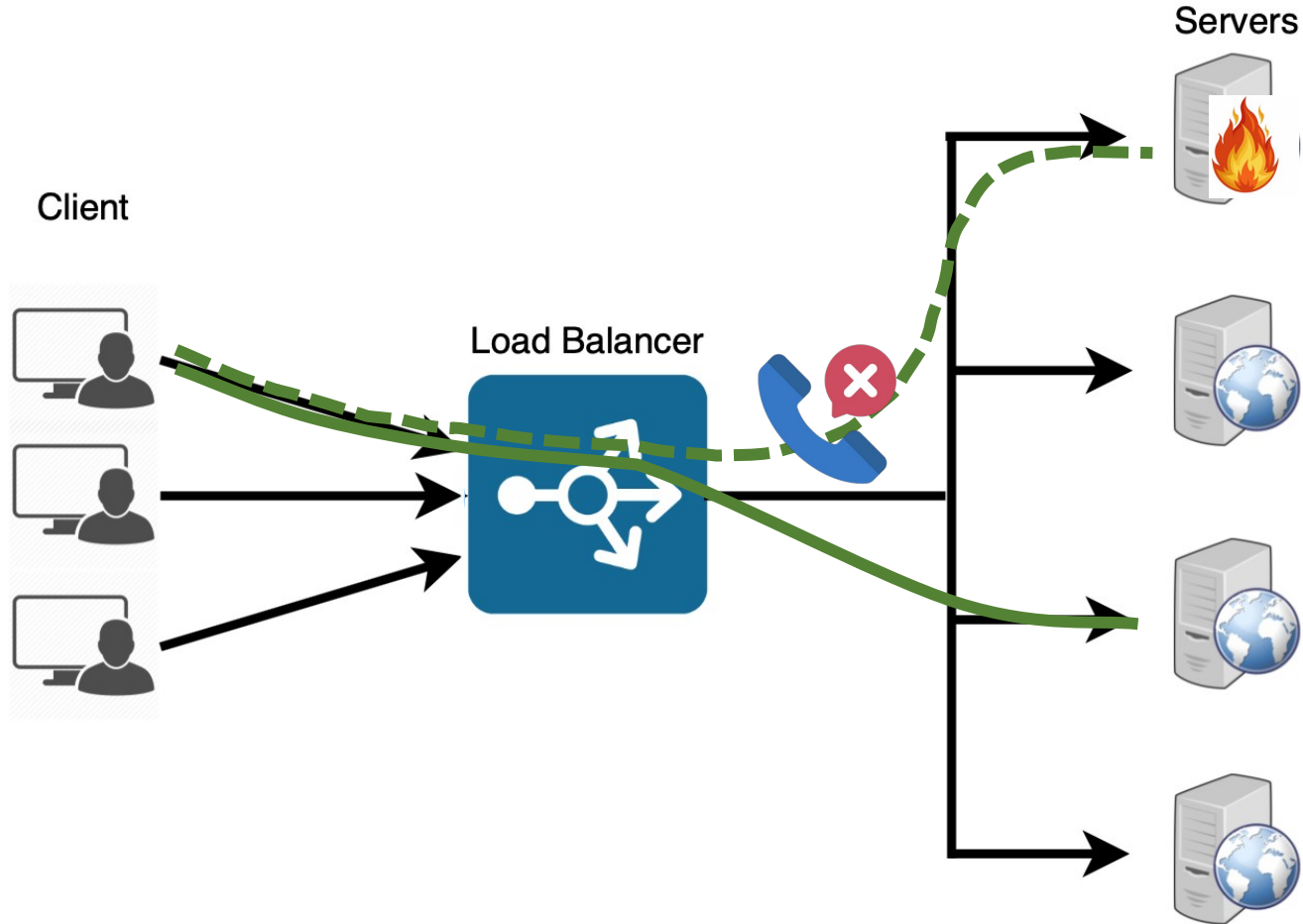
Third-party advertising cookies



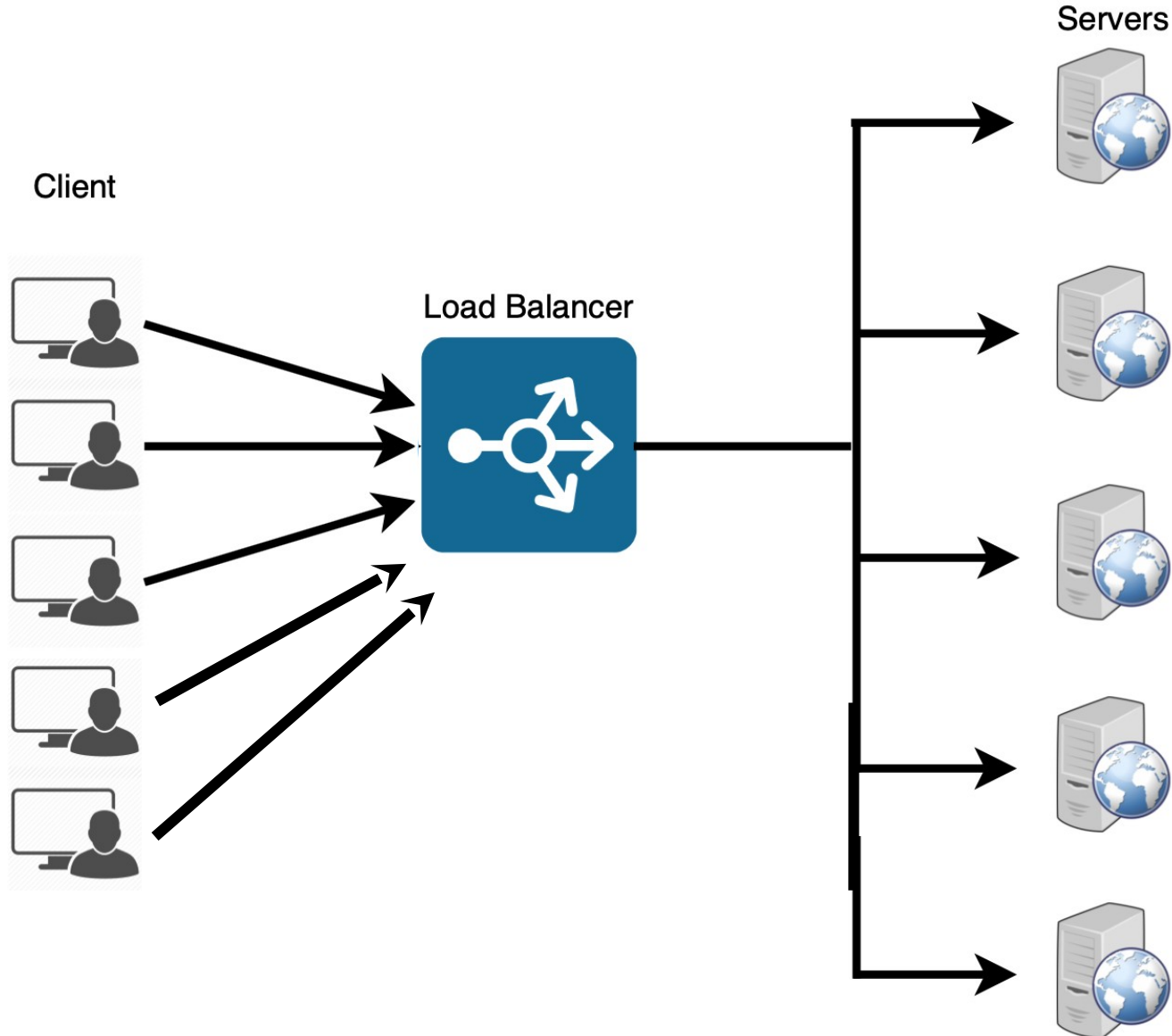
Stateless Service can be duplicated



Provides FT

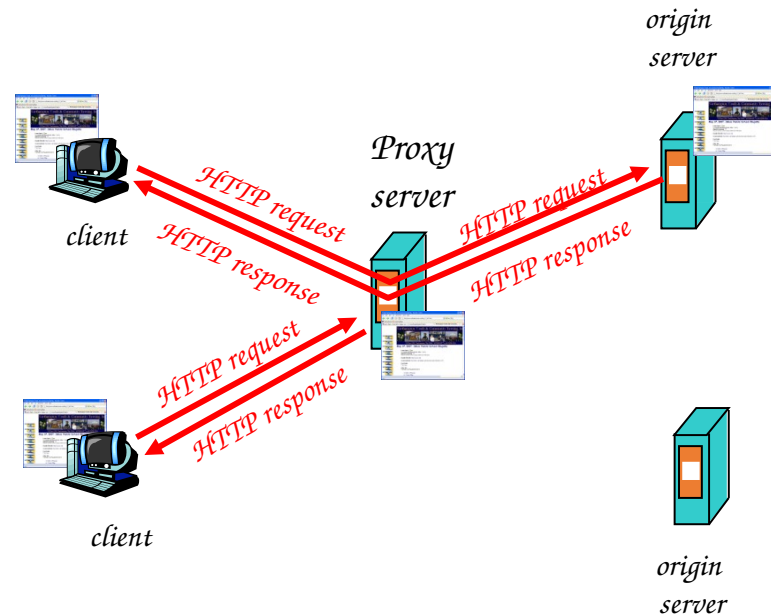


Provides Scalability



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



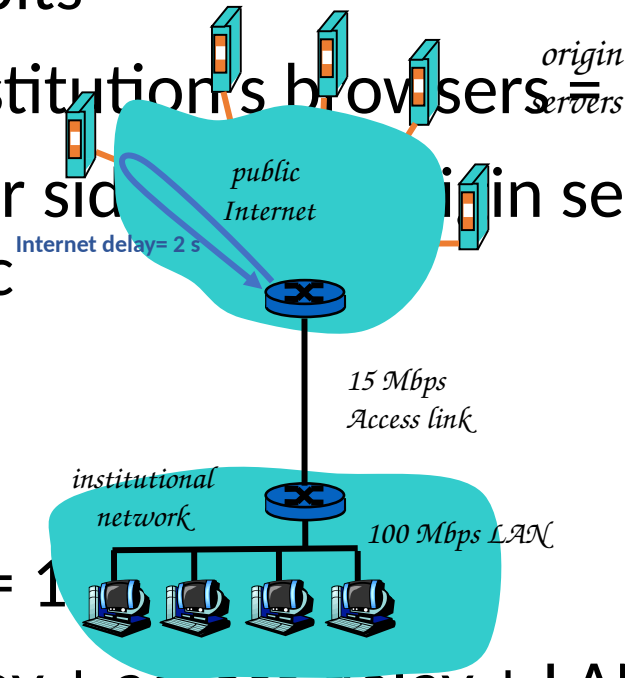
Caching Example

assumptions

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

consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + milliseconds



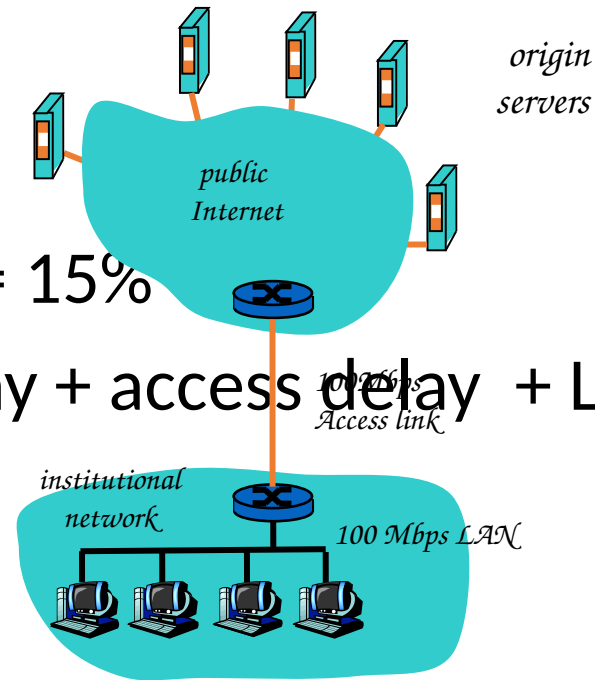
Caching Example (cont)

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 + msec + msec
often a costly upgrade



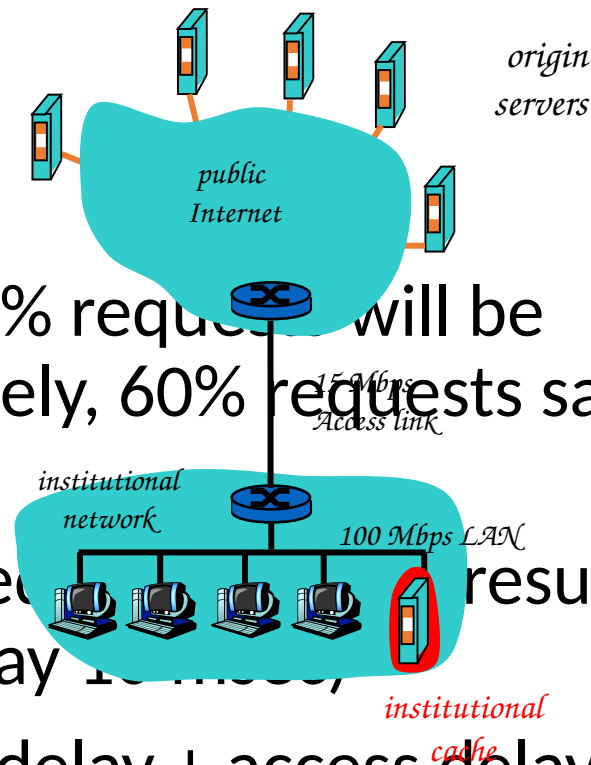
Caching Example (cont)

possible solution:

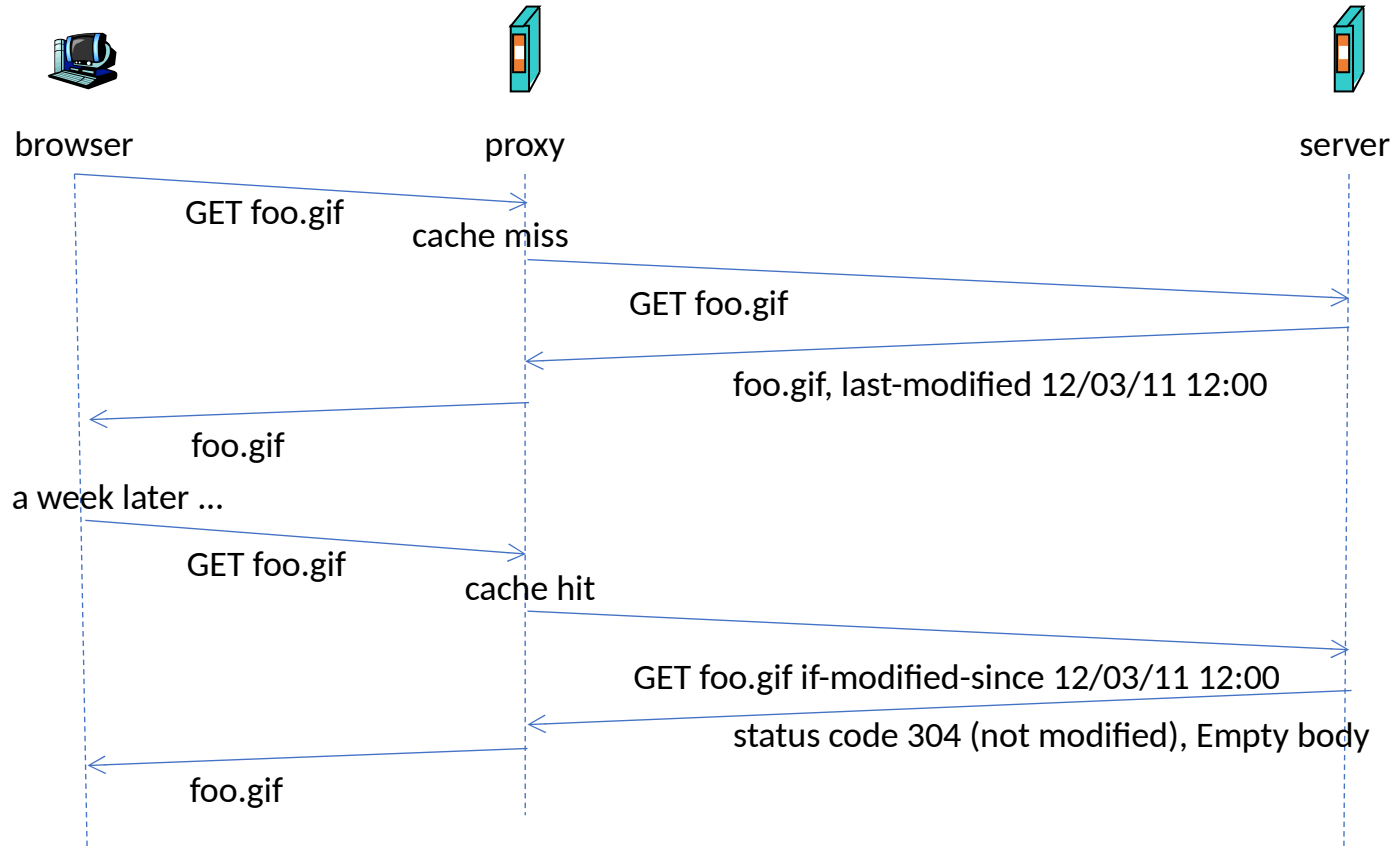
- install cache

consequence

- suppose hit rate is 0.4 (40% requests will be satisfied almost immediately, 60% requests satisfied by origin server)
- utilization of access link reduced, resulting in negligible delays (say 10 msec)
- avg total delay = Internet delay + access delay + LAN delay
$$= 0.6 * 2.01 \text{ secs} + 0.4 * 10 \text{ millisecs} < 1.3 \text{ secs}$$



Conditional GET



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.



Origins

- *Representational State Transfer* – REST: defined in 2000 Roy Fielding's PhD dissertation (after he worked on HTTP 1.1 and URI RFCs)
- Web application =
 - network of Web resources (a virtual state-machine)
 - where the user progresses through the application by selecting resource identifiers and resource operations (application state transitions), resulting in the next resource's representation (the next application state) being transferred to the end user for their use.
- An architectural style, not a standard nor a protocol

Principles of RESTful Architecture (1/2)

- A resource
 - is identified using an URI,
 - references
 - one entity (eg. user Paul) or
 - a set of entities (eg. all male users)
 - URI doesn't change (but the referenced entity might)
 - and can have multiple representations (JSON, XML...).
- The representation of a resource contains enough information for the client to request a change to its state.
 - Messages include enough information to describe how to process them (eg. Content type)
 - HATEOS (*Hypermedia as the Engine of Application State*)

HATEOAS Example

response if balance > 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
  <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
  <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
  <link rel="close" href="https://bank.example.com/accounts/12345/status" />
</account>
```

request

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

response if balance < 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
</account>
```

Principles of RESTful Architecture (2/2)

- Separation of concerns between the client (user interface concerns) and the server (data storage and processing concerns)
- Stateless communication: the server only stores resources states while the client is in charge of providing the application state.
- Responses should define the extent to which they can be cached.
- A client may not be directly connected to the end-server: there can be proxies, an additional security layer, and the server might call other servers to complete the service.

Semantics of HTTP methods

HTTP method	Operation on the resource	URIs: examples	HTTP response status	<i>location</i> header	safe	idem potent
GET	read	GET /serv/users GET /serv/users/34	200 OK	no	yes	Yes
POST	create	POST /serv/users # body { name: "Toto" }	201 Created	Yes	no	no
PUT	update	PUT /serv/users/34 # body { name: "Jacques" }	200, 204 No Content	no	no	Yes
PATCH	partial update					
DELETE	delete	DELETE /serv/users/34	200, 204, 202 Accepted	no	no	yes

Example of scenario

- Book a room:

POST [http://myhotel.com/reservations?date="12/03/2021"&nights=2&persons=4](http://myhotel.com/reservations?date=12/03/2021&nights=2&persons=4)

Server replies with reservation number 123

- Display reservation:

GET <http://myhotel.com/reservations/123>

- Update the reservation:

PATCH <http://myhotel.com/reservations/123?persons=3>

- Cancel the reservation:

DELETE <http://myhotel.com/reservations/123>

Best Practices for well-designed RESTful APIs

- Use only nouns for a URI:

~~/getAllReservations~~ GET /reservations

- Use plural nouns:

GET /reservations for all reservations

GET /reservations/123 for a specific reservation

- GET method should not alter the state of a resource
- Use sub-resources for relationships between resources

GET /reservations/123/persons/1: first occupant of the reservation #123

- Use “content-type” and “accept” HTTP headers to specify input/output format

Best Practices for well-designed RESTful APIs

- Provide proper HTTP status codes
 - 400 Bad Request (requête mal formée)
 - 401 Unauthorized (authentification requise)
 - 403 Forbidden (accès interdit)
 - 404 Not Found (ressource inexistante)
 - 500 Internal Server Error (erreur serveur)
- Explicit error messages but no sensitive information leak!

Best Practices for well-designed RESTful APIs

- Offer filtering and paging capabilities for large data sets

`GET /reservations?date=28/02/2021`

`GET /reservations?from=5&to=25`

- Version the API

2 strategies:

- In the URI: `GET /api/v2/reservations/123`



Easy to use with a web browser



Non-compliant with REST principle “one resource = one URI”

- In the accept header:

`GET /api/reservations/123 accept: application/v2`



More complex for the client



More REST-compliant