HTTP

HTTP History

- 1991 – HTTP 0.9 (HTTP definition)
- 1996 – HTTP 1.0 (RFC 1945)
- 1999 – HTTP 1.1 (RFC 2608)
- 2007 – HTTPbis group was formed to revise and clarify HTTP 1.1

2014 - HTTP

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.

2014 – updated pack of HTTP 1.1 specifications

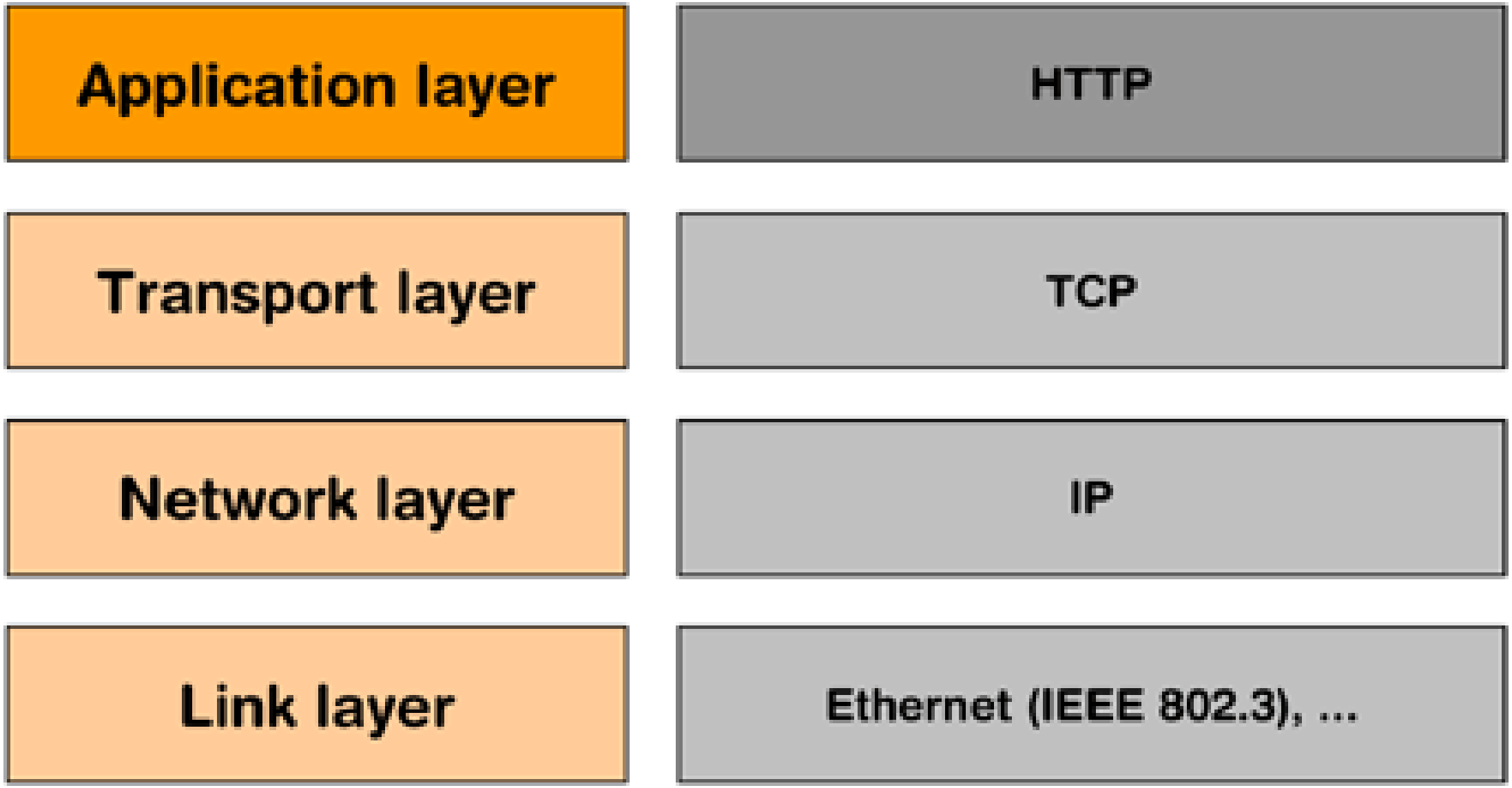
- RFC 7230 - HTTP/1.1: Message Syntax and Routing
- RFC 7231 - HTTP/1.1: Semantics and Content
- RFC 7232 - HTTP/1.1: Conditional Requests
- RFC 7233 - HTTP/1.1: Range Requests
- RFC 7234 - HTTP/1.1: Caching
- RFC 7235 - HTTP/1.1: Authentication

2015

~~Google SPDY~~ ~~Microsoft Speed+Mobility~~

HTTP/2 (RFC 7540)

HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients.


Application layer
HTTP
Transport layer
TCP
Network layer
IP
Link layer
Ethernet (IEEE 802.3), ...

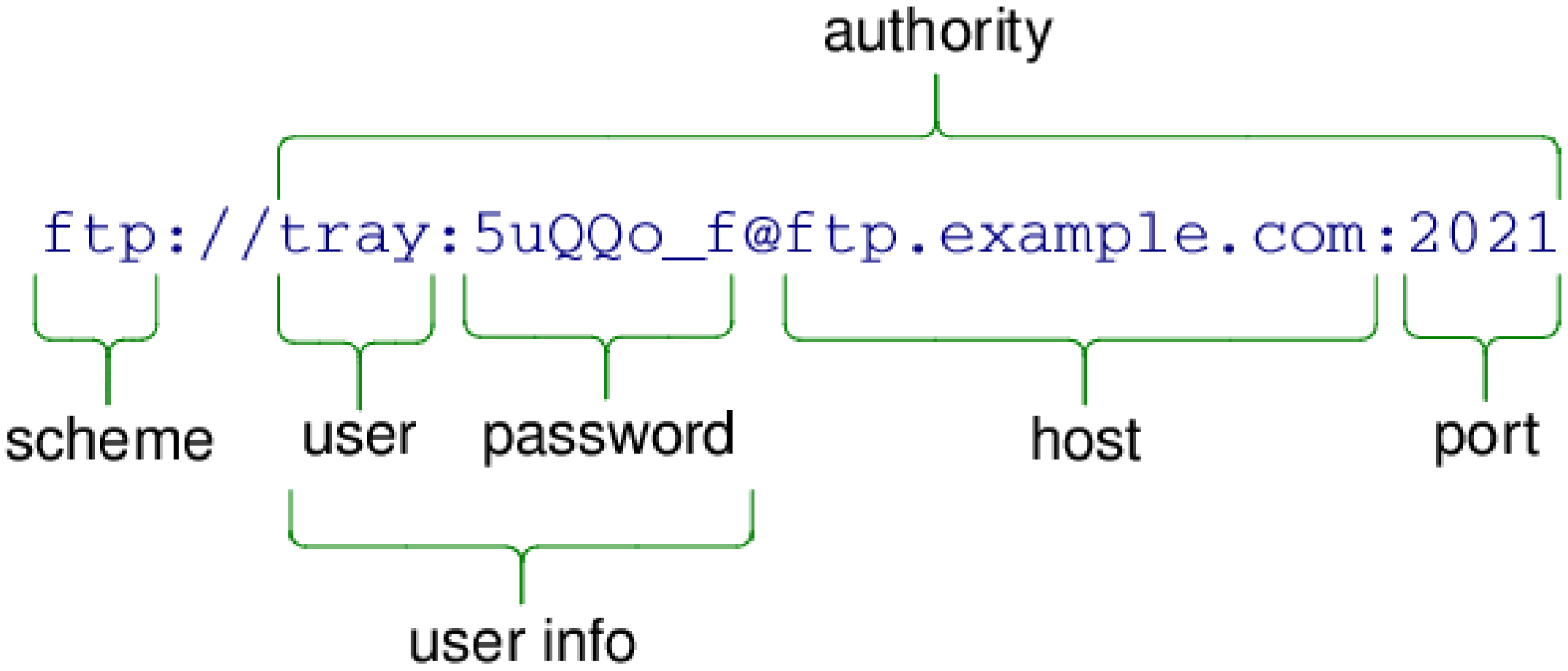
URI Scheme

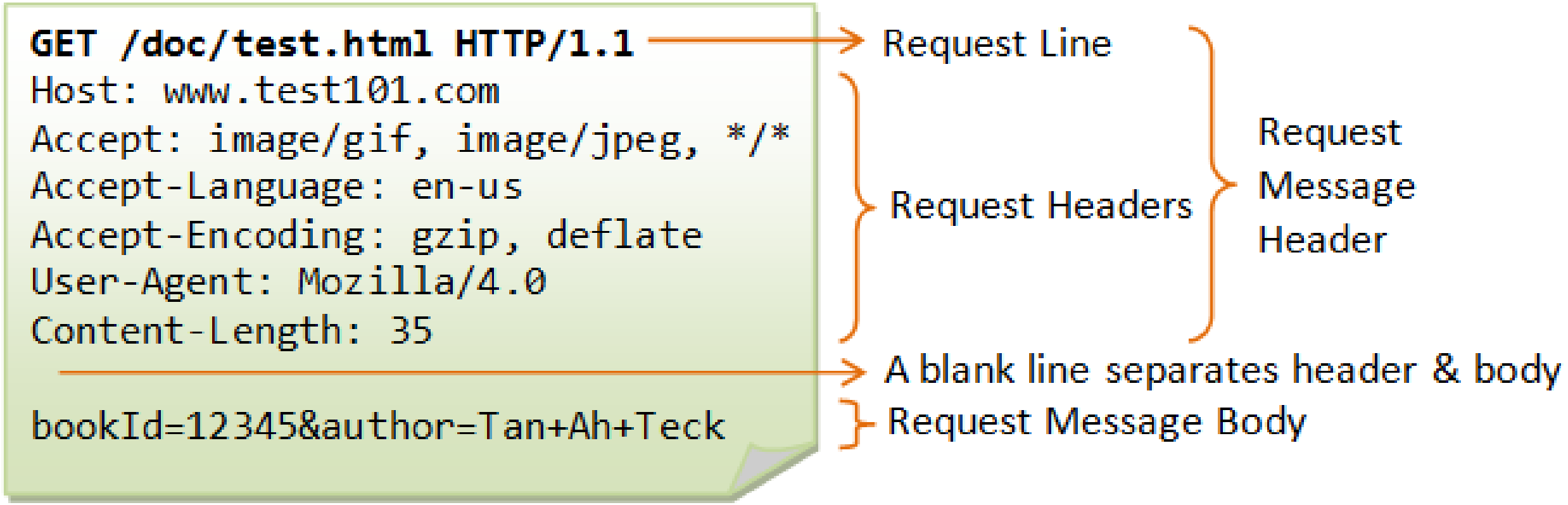

http://www.google.com/search?q=facebook#result
protocol
domain
path
parameters
fragment

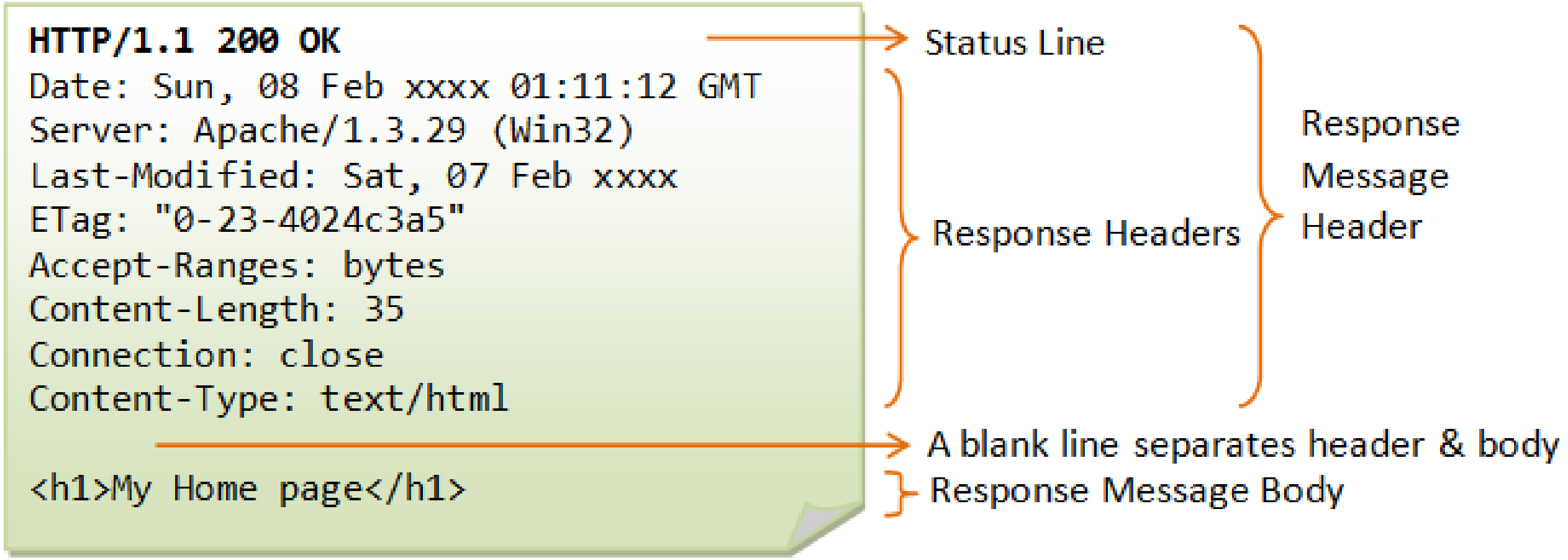
What is real domain?

http://facebook.com:mail=box@dangero.us

URI Authority




GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
bookId=12345&author=Tan+Ah+Teck
Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body


HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html
<h1>My Home page</h1>
Status Line
Response Headers
Response Message Header
A blank line separates header & body
Response Message Body

SAFE METHODS

- Request methods are considered **"safe"** - if their defined semantics are essentially read-only; i.e., the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource.

- The purpose of distinguishing between safe and unsafe methods is to allow **automated retrieval processes** (spiders) and **cache performance optimization** (pre-fetching) to work without fear of causing harm. In addition, it allows a user agent to apply appropriate constraints on the automated use of unsafe methods when processing potentially untrusted content.

Common Method Properties (RFC 7231)

IDEMPOTENT METHODS

- A request method is considered **"idempotent"** if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request.

- Idempotent methods are distinguished because the **request can be repeated automatically** if a communication failure occurs before the client is able to read the server's response.

Common Method Properties (RFC 7231)

CACHABLE METHODS

Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be stored for future reuse; for specific requirements see [RFC7234]. In general, safe methods that do not depend on a current or authoritative response are defined as cacheable; this specification defines GET, HEAD, and POST as cacheable, although the overwhelming majority of cache implementations only support GET and HEAD.

Common Method Properties (RFC 7231)

Method	Safe	Idempotent	Cacheable
GET	X	X	X
HEAD	X	X	X
POST			X (???)
PUT		X	
DELETE		X	
CONNECT			
OPTIONS	X	X	X
TRACE	X	X	X

GET vs POST

Use **GET** if:

- The interaction is more like a question (i.e., it is a safe operation such as a query, read operation, or lookup).

Use **POST** if:

- The interaction is more like an order, or
- The interaction changes the state of the resource in a way that the user would perceive (e.g., a subscription to a service), or
- The user be held accountable for the results of the interaction.

URIs, Addressability, and the use of HTTP GET and POST

Task: Mailing List Subscription

1. The user sends a subscribe message to an administrative mailbox (mylist-request@example.org).
2. The list processing software sends an email response to the user, requesting that the user confirm the subscription request, and including a link to a confirmation page.
3. The user follows the link to the confirmation page and is informed "your subscription is confirmed".

Task: Mailing List Subscription

1. The user sends a subscribe message to an administrative mailbox (mylist-request@example.org).
2. The list processing software sends an email response to the user, requesting that the user confirm the subscription request, and including a link to a confirmation page.
3. ~~The user follows the link to the confirmation page and is informed "your subscription is confirmed".~~

POST vs PUT

The fundamental difference between the **POST** and **PUT** methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle **the enclosed representation** according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the **state of the target resource**. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

http://tools.ietf.org/html/rfc7231#page-28

HTTP Status Codes

- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

- Cacheable by default: 200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501

http://tools.ietf.org/html/rfc7231#section-6

MIME Types

- application/*
- audio/*
- example/*
- image/*
- message/*
- model/*
- multipart/*
- text/*
- video/*

http://www.iana.org/assignments/media-types/media-types.xhtml

Content Negotiation

- **Proactive Negotiation** (a.k.a., server-driven negotiation).

Preferences are sent by the user agent in a request to encourage an algorithm located at the server to select the preferred representation, it is called proactive negotiation

- **Reactive negotiation** (a.k.a., agent-driven negotiation),

Selection of the best response representation (regardless of the status code) is performed by the user agent after receiving an initial response from the origin server that contains a list of resources for alternative representations (300 Multiply Choices).

http://tools.ietf.org/html/rfc7231#section-3.4

Proactive content negotiation

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Accept

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, */*;q=0.8

1) text/html, application/xhtml+xml, image/webp
2) application/xml; q=0.9
3) */*;q=0.8

Accept-Language

- **Accept-Language:** en-US,en;q=0.8

- 1) en-US
- 2) en; q=0.8

Accept-Encoding

Accept-Encoding: gzip, deflate, sdch

Accept-Encoding: identity = no encoding

http://www.iana.org/assignments/http-parameters/http-parameters.xhtml#content-coding

http://bitsup.blogspot.com.by/2015/09/brotli-content-encoding-for-firefox-44.html

http://www.theverge.com/2016/1/20/10797268/googles-new-algorithm-chrome-run-faster

Vary (Response header)

- Vary: accept-encoding, accept-language
- Vary: *

Cookie

Server -> User Agent

Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly

Set-Cookie: lang=en-US; Path=/; Domain=example.com;

User Agent -> Server

Cookie: SID=31d4d96e407aad42; lang=en-US

RFC 6265 HTTP State Management Mechanism

Cookie attributes

Name	Definition
Expires	The maximum lifetime of the cookie, represented as the date and time at which the cookie expires.
Max-Age	The maximum lifetime of the cookie, represented as the number of seconds until the cookie expires. (has precedence over Expires)
Domain	Specifies those hosts to which the cookie will be sent.
Path	The user agent will include the cookie in an HTTP request only if the path portion of the request-uri matches (or is a subdirectory of) the cookie's Path attribute
Secure	Limits the scope of the cookie to "secure" channels (where "secure" is defined by the user agent)
HttpOnly	Limits the scope of the cookie to HTTP requests (not browser API).

Cookie Limits

Practical user agent implementations have limits on the number and size of cookies that they can store. General-use user agents SHOULD provide each of the following minimum capabilities:

- At least 4096 bytes per cookie (as measured by the sum of the length of the cookie's name, value, and attributes).

- At least 50 cookies per domain.

- At least 3000 cookies total.

Q & A