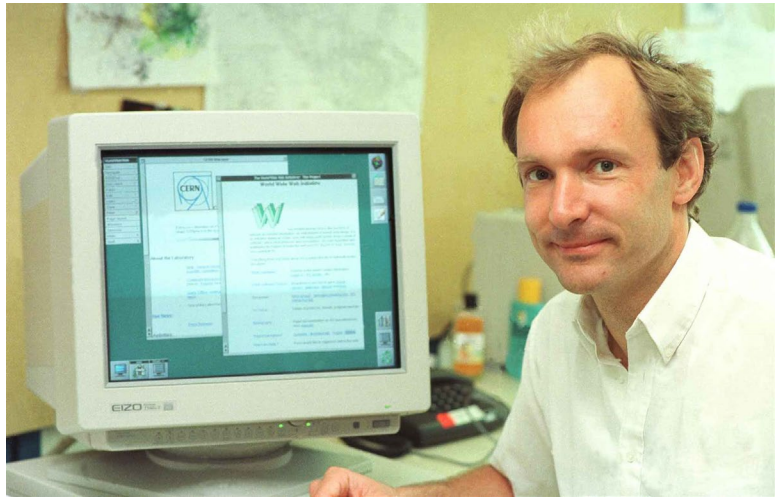
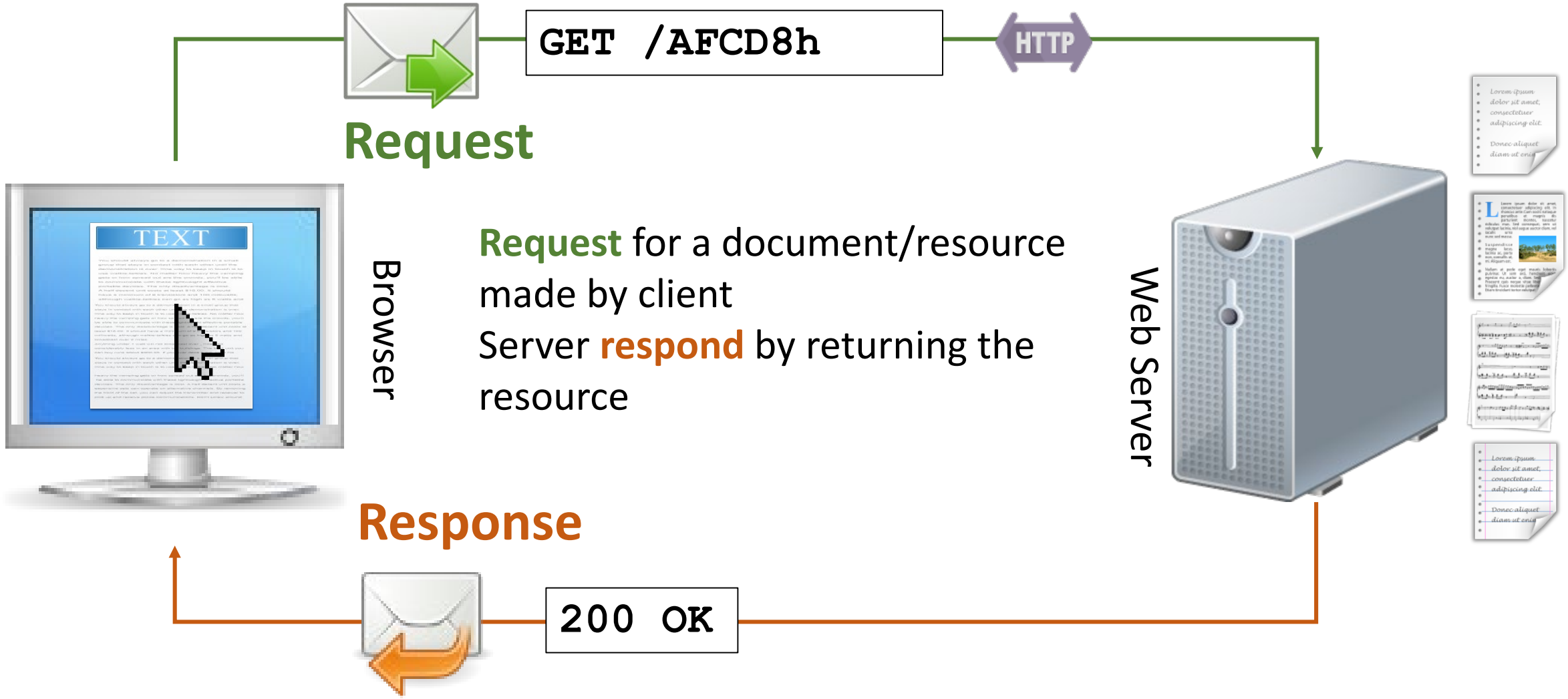
Hypertext Transfer Protocol

A Short History of HTTP

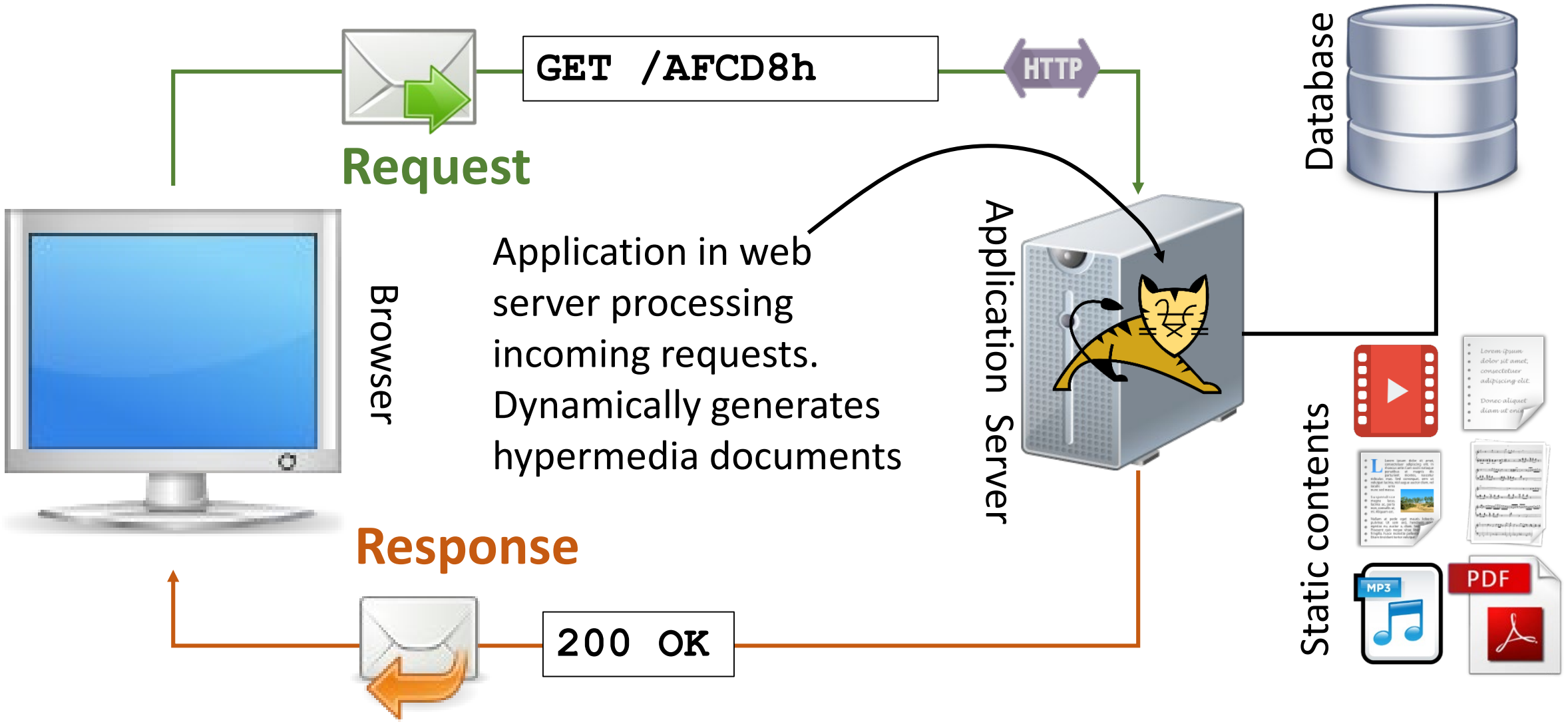


- Developed by Tim Berners-Lee in 1990 at CERN
  - To organize research documents available on the Internet
- Research documents are hypertext
  - Contains references to other research paper
  - Could go from paper to paper following a particular topic
- Research document (hypertext) would be authored in HTML
- References within the research document would linked with hyperlink
  - A hyperlink would reference a document on a server
  - When a link is activated (clicked), the document would be retrieved from the server and displayed
- Combining the idea of FTP/file server with hypertext/hyperlinks

Initial Web

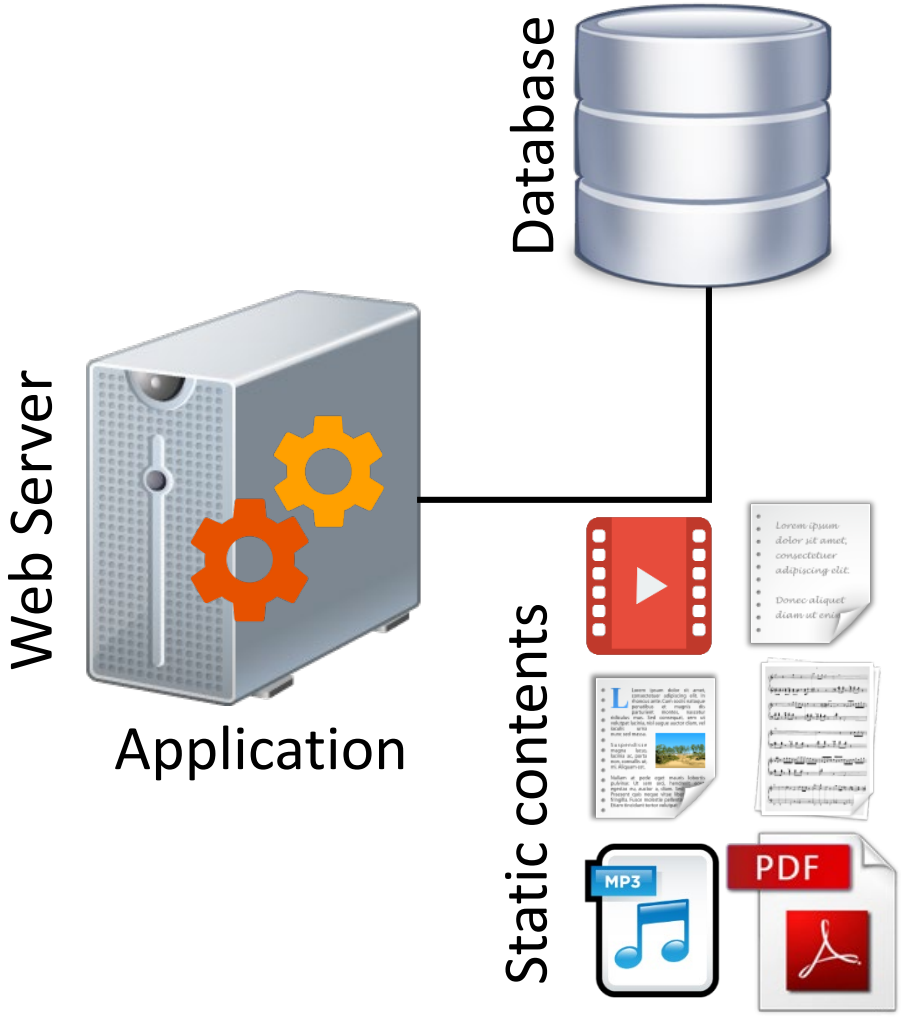

GET /AFCD8h
HTTP
Request
TEXT
Browser
Request for a document/resource made by client
Server respond by returning the resource
Web Server
Response
200 OK

Web Applications

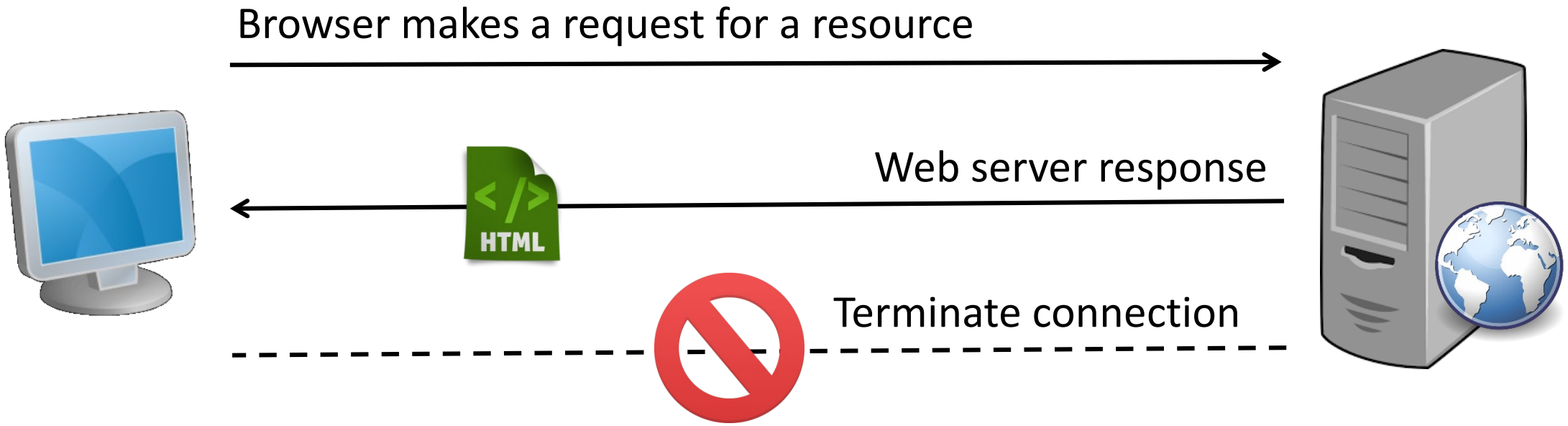


HTTP Based Service

- Application in a web server could produce any type of content, not just hypertext
  - Images, movies, database records
- Application in a web server could perform any service for you
  - Eg. buy 2 4PM movie ticket
- Web Server evolved from serving files to serving application services
- Application services are accessed over HTTP



HTTP Protocol

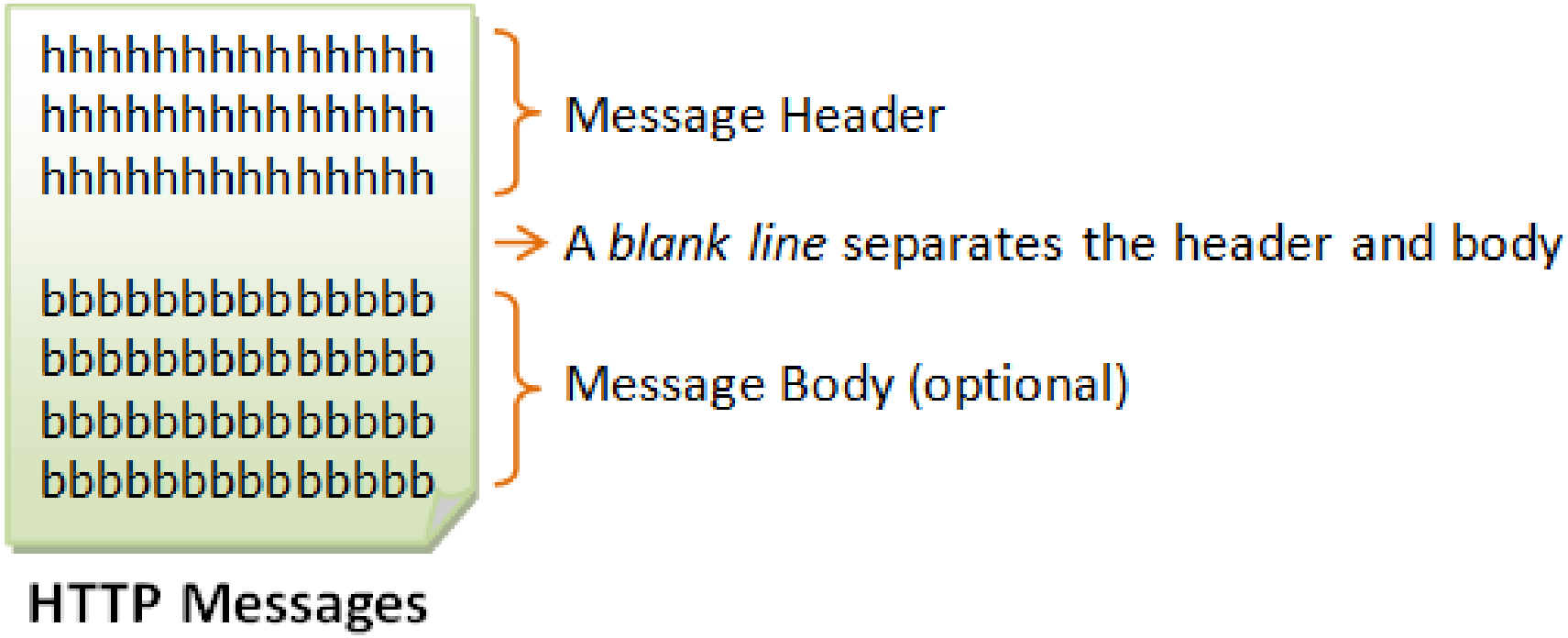


Client terminates the connection and open another connection to request the next resource

HTTP Protocol

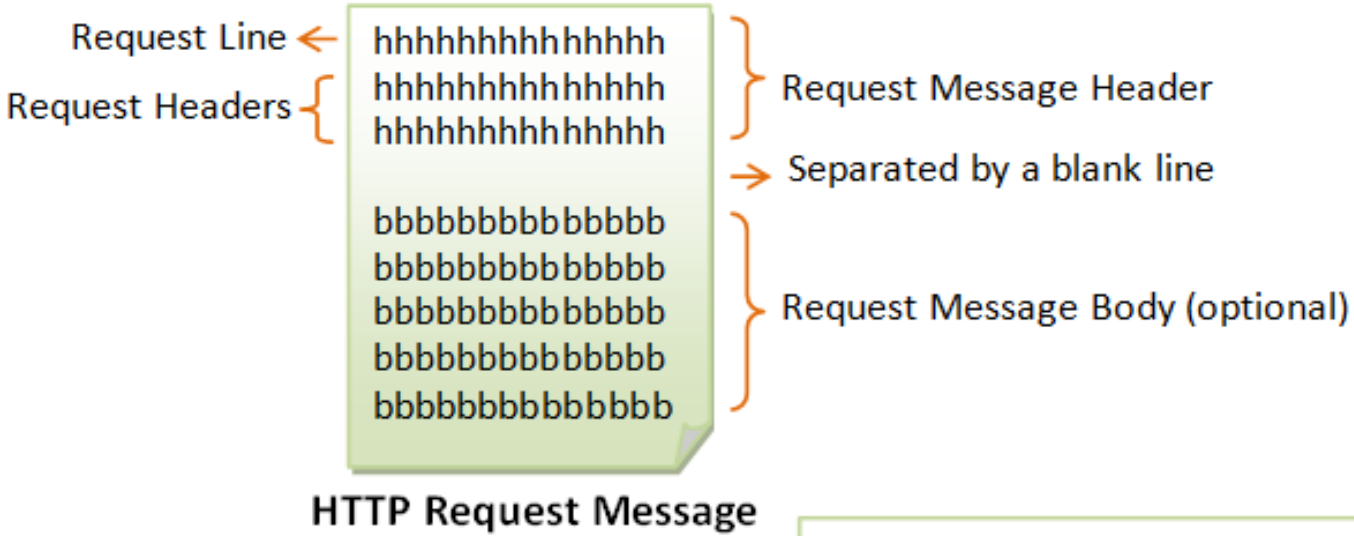
- A text protocol made by a client to a web server
- Request response protocol
  - Client has to initiate the request to the server
  - Server cannot send unsolicited data to a client
- Half duplex protocol
  - Data can be transmitted both ways
  - But on one side can transmit at any one time like walkie-talkie
- Connection is used once
  - Need to reconnect to the server for next request

HTTP Message Structure



Images from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Request

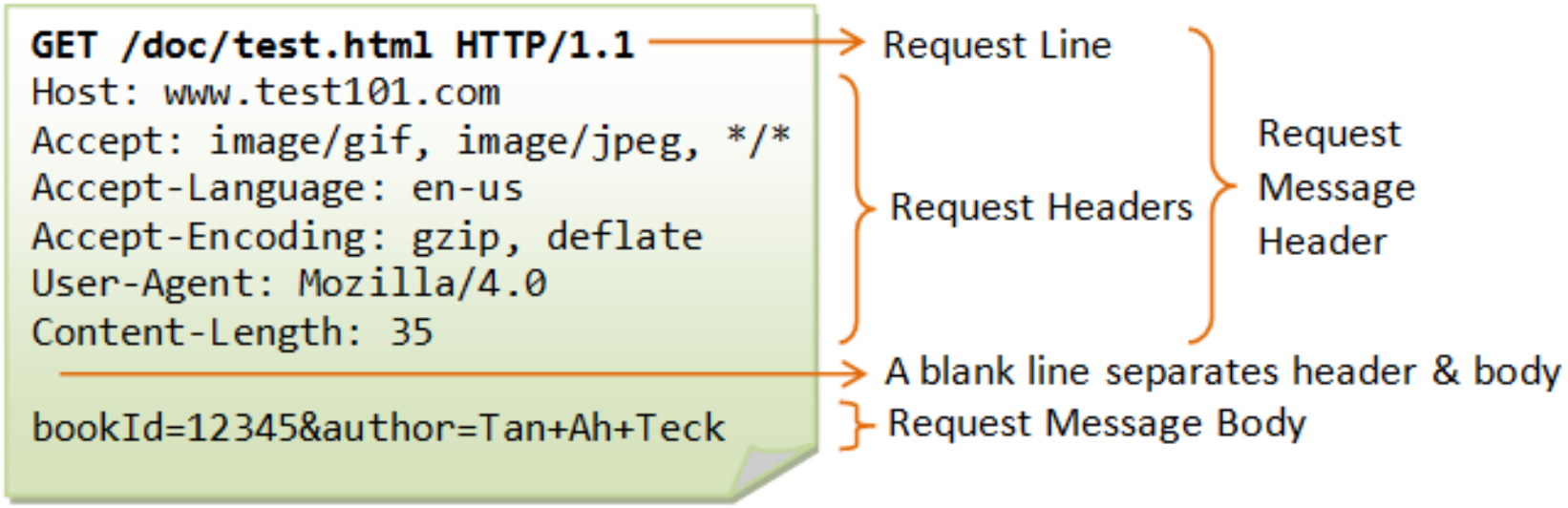


HTTP Request Message

HTTP method Resource name

```
GET /doc/test.html
```

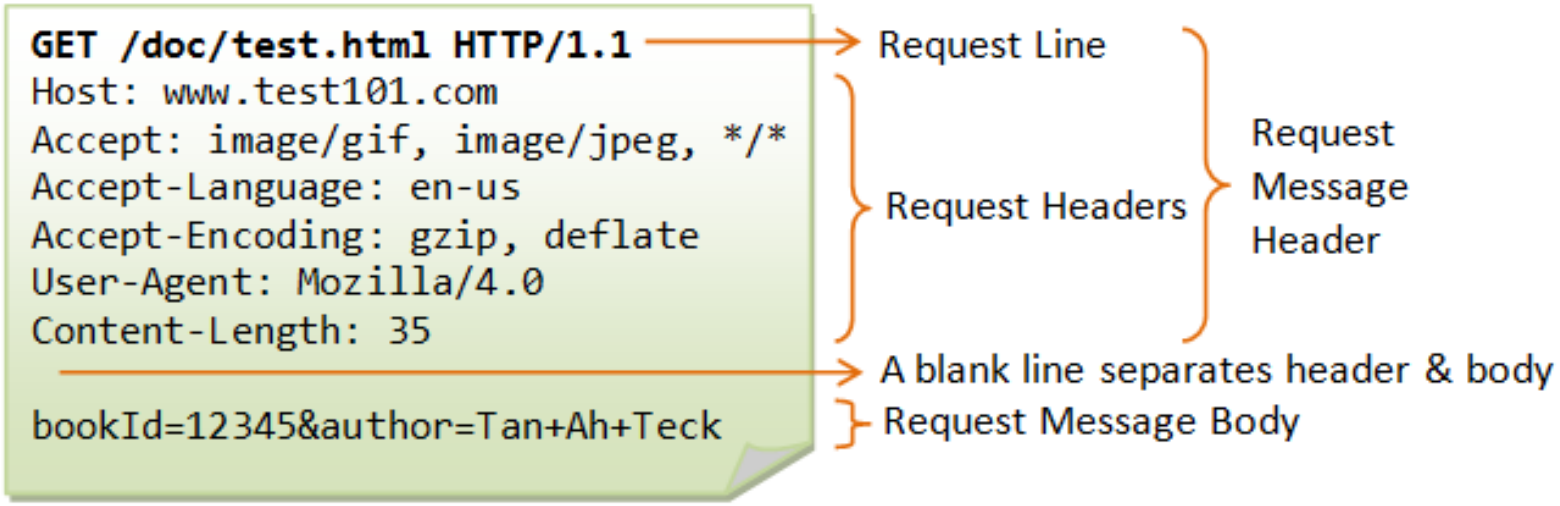
GET
POST
PUT
DELETE
HEAD
OPTION
TRACE



Images from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Request

- Request line
  - VERB
    - One of HTTP method
    - GET, POST, PUT, DELETE, etc.
  - NOUN
    - Resource nameHTTP headers
- HTTP headers
  - Key value pairs
- Entity body/payload
  - Holds the request's content, if any



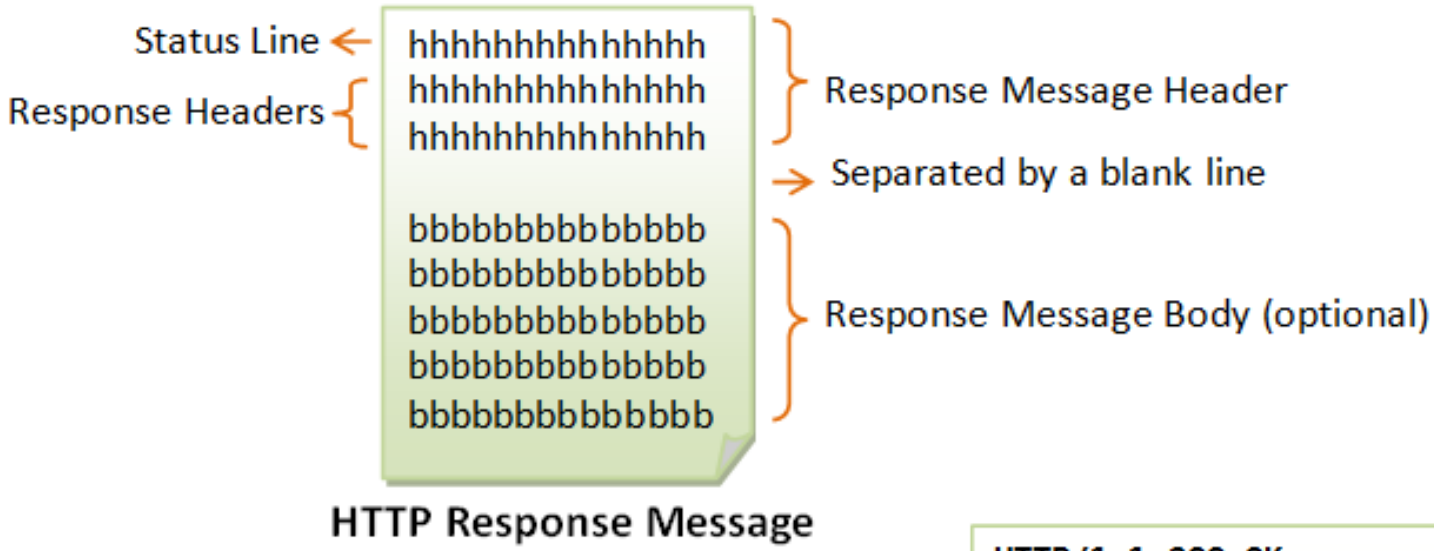
HTTP Method

- GET - Request a representation/data from the specified resource
- POST - Used to submit data to the specified resource
- PUT - Replaces the data of the specified resource
- PATCH - Partially modify the specified resource
- DELETE - Deletes the resource
- HEAD - Like GET but without the response payload (if any)

HTTP Method Example

HTTP Method	Action	Example
GET	Request data for a given resource Read only	`GET /employees` - collection `GET /employee/34` - item
POST	Create a new employee with the data provided	`POST /employee` - new employee data is in the request's body
PUT	Perform a full update on the employee with the provided data	`PUT /employee/34` - update data is in the request's body
PATCH	Perform a partial update on the employee with the provided data	`PATCH /employee/34` - update data is in the request's body
DELETE	Delete a resource	`DELETE /employee/34` - deletes the provided resource

HTTP Response

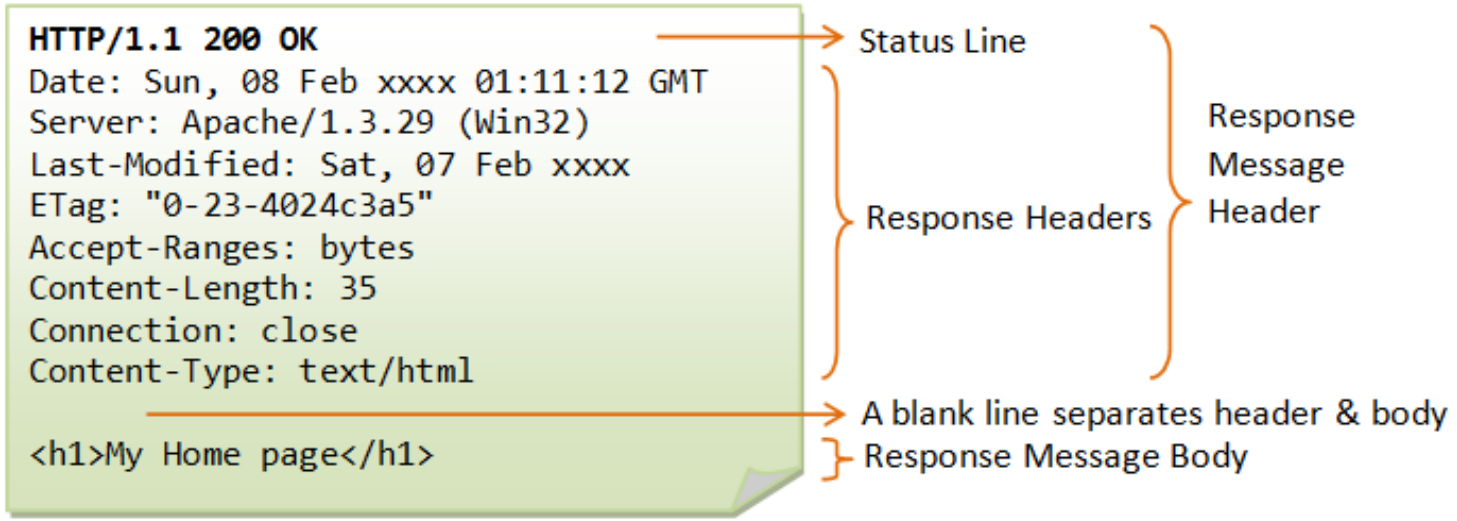


HTTP Response Message

Status code

200 OK

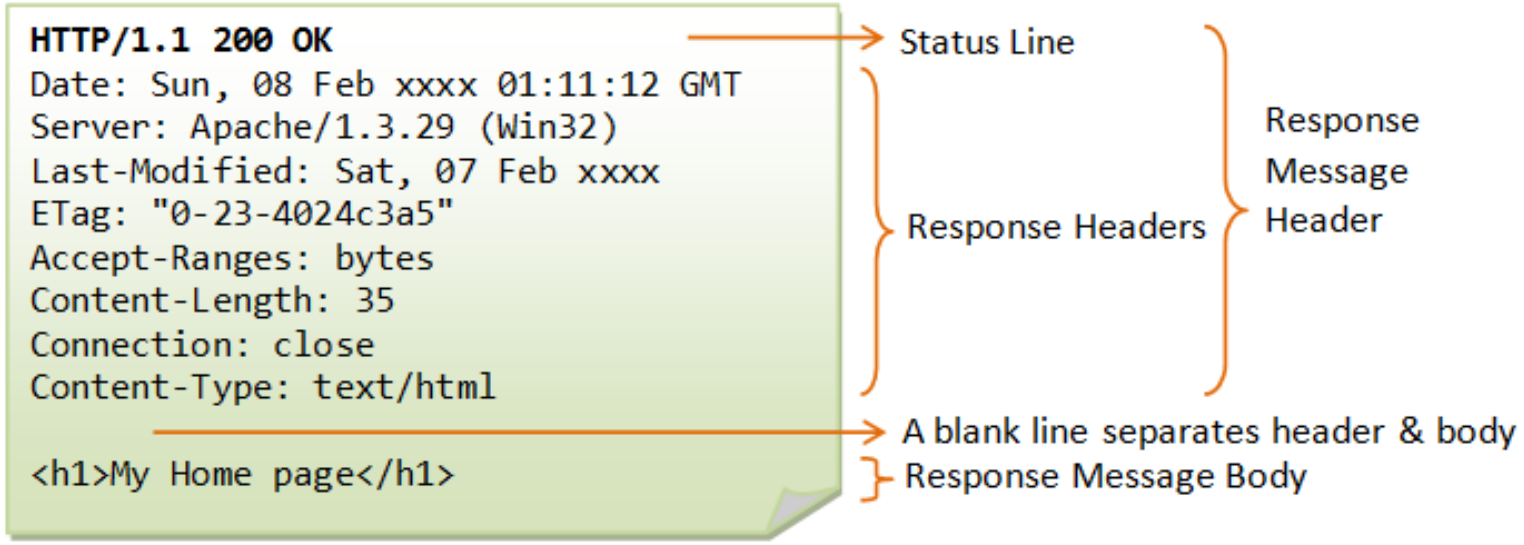
100
200
300
400
500



Images from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Response

- Response line
  - A status code, indicating if the request has been successful, or not, and why.
  - A status message,
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.



HTTP Status Code

- Information - 100 range
  - Provisional response
  - Eg. 100 Continue
- Successful - 200 range
  - Request has been successfully received, accepted and processed
  - Eg. 201 Created
- Redirect - 300 range
  - Client has to take further actions in order for the request to be fulfilled
  - Eg. 301 Moved Permanently
- Client error - 400 range
  - Request error or error in the request
  - Eg. 400 Bad Request
- Server error - 500 range
  - The server has encounter a catastrophic failure and is unable to process the request
  - Eg. 503 Service Unavailable

Method, Resource and Status

Operation	Verb	Noun	Outcome
Read	GET	/customer/1	200 OK
Create	POST	/customer	201 Created
Update	PUT	/customer/1	200 OK
Delete	DELETE	/customer/1	200 OK

Idempotent and Safe

Safe

`100`

The value 100 never changes no matter how many times you look at it

Idempotent

`100 x 1 x 1 ...`

The operation will produce the same result no matter how many time you apply the operation

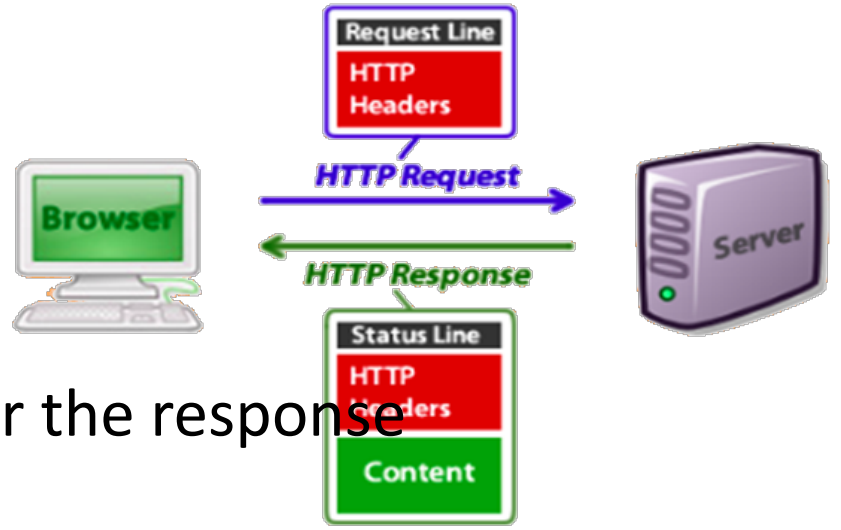
Safe and Idempotent Methods

- Safe methods are methods that do not modify any resources on the server
  - Eg. `GET /employees` - will fetch a list of employees but does not modify the resource
  - Results can be cached or pre-fetched without any repercussion to the server
- Idempotent methods are methods that will produce the same result when called multiple times
  - Idempotent modifies the resource
  - Eg. `PUT /employee/34` - updates the resource with the provided new data; if the data remains the same calling PUT multiple times on `/employee/34` will have the same effect
  - Eg. `POST /employee` is not idempotent because an employee may be created multiple times especially if the employee id is allocated based on a number generator

Safe and Idempotent Methods

Method	Idempotent	Safe
`GET`	✓	✓
`POST`	✗	✗
`PUT`	✓	✗
`PATCH`	✗	✗
`DELETE`	✓	✗

HTTP Headers



- Allows additional information in either the request or the response
- HTTP headers are key value pairs separated by :
  - Eg `Host: acme.corp`
- Types of HTTP headers
  - Request headers - contains information about the client or the requested resource
    - Eg. `Accept: text/plain`
  - Response headers - contains information about the response
    - Eg. `Server: nginx/1.15.12`
  - Entity headers - containers information about the payload
    - Eg. `Transfer-Encoding: chunked`
- Custom headers are prefixed with a `X-`
  - Eg. `X-Powered-By: Express`

Web Application

- HTTP connections are initiated by the client
- A web server/application waits for incoming request
- Process request based on
  - HTTP method
  - Resources
  - Other information - HTTP headers, payload, query string
- Returns a response
  - Payload/data
  - An appropriate status code

Web Application Example


```
const express = require('express');
const count = 0;
const app = express()

app.get('/', (req, resp) => {
   resp.set('X-Count', '' + ++count);
   resp.type('text/html');
   resp.status(200);
   resp.send('<h1><code>hello, world</code></h1>');
});

app.use((req, resp) => {
   resp.status(404);
   resp.end();
});
```

Any **GET** / message will be processed by this Express middleware

Set custom header.
Value must be string

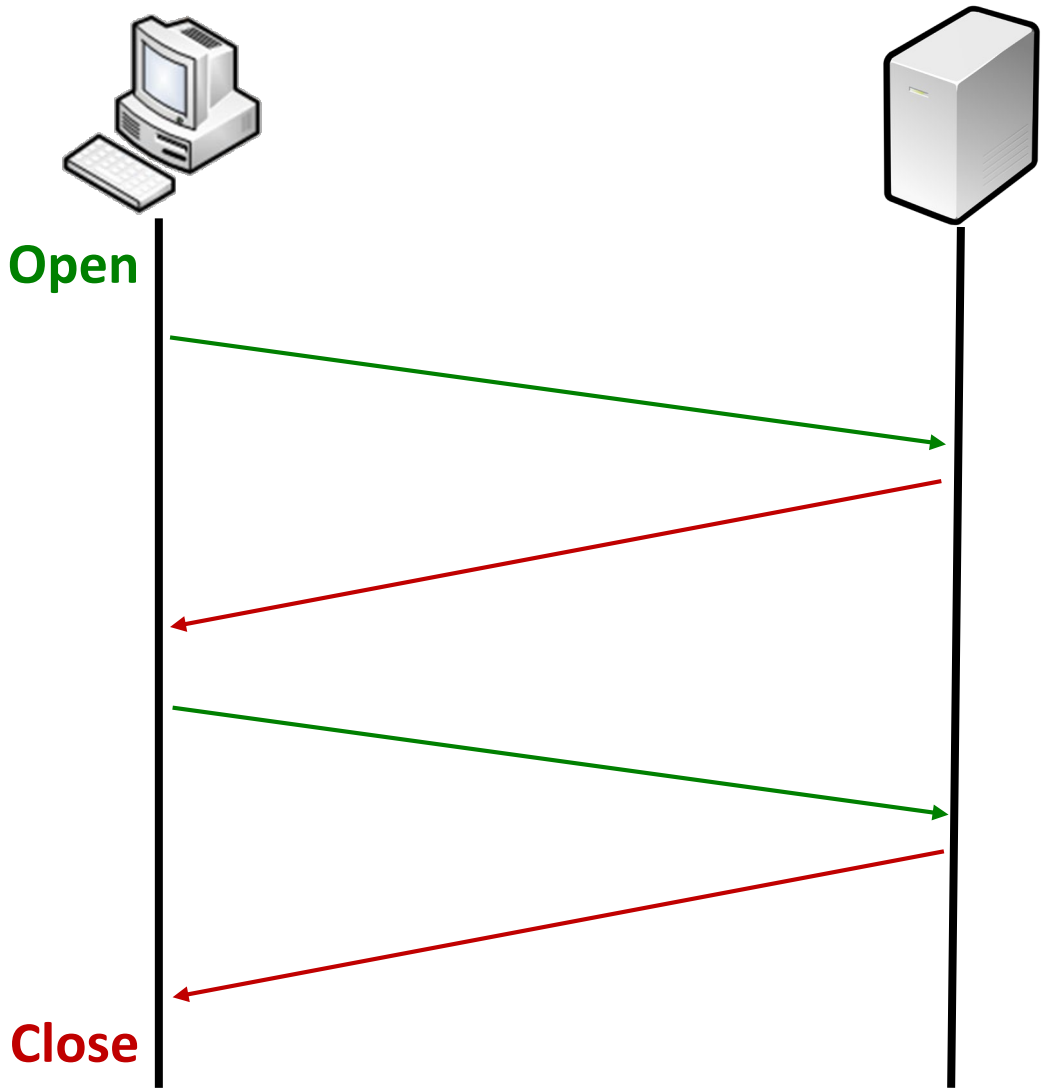
Standard header

Response with a status code of 200 and the payload

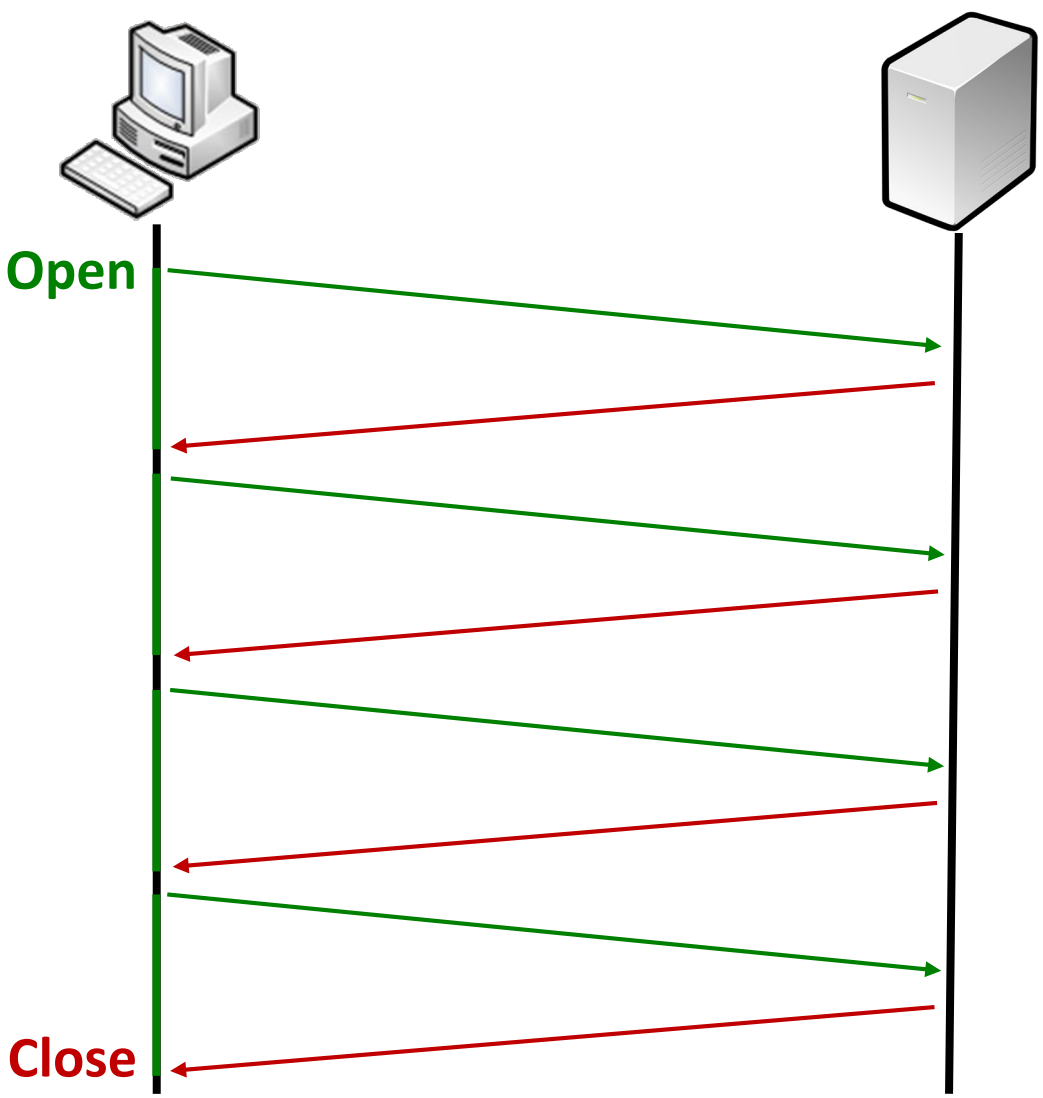
Any other type of request will be not be processed

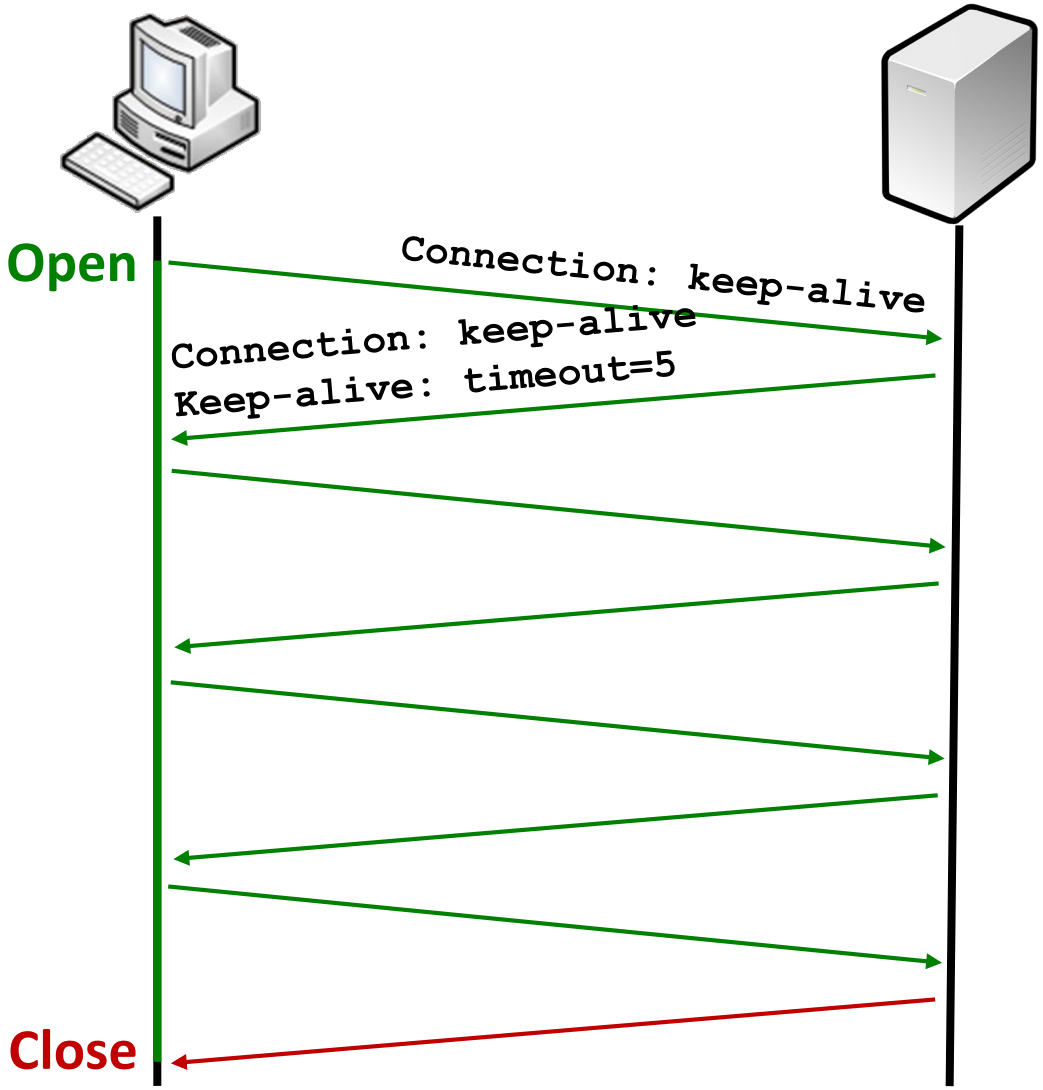
HTTP 1.1

- In efficient use of TCP connection
- Head of line blocking
  - HTTP connection can only handle one request
  - If current request block on a resources, then next resource will have to wait even though it does not block
- Solution - reduce the number of request to the server
  - Concatenation
  - Inline resources
  - Sprite sheets
  - `Connection` and `Keep-Alive` headers



Persistent Connection - HTTP 1.1

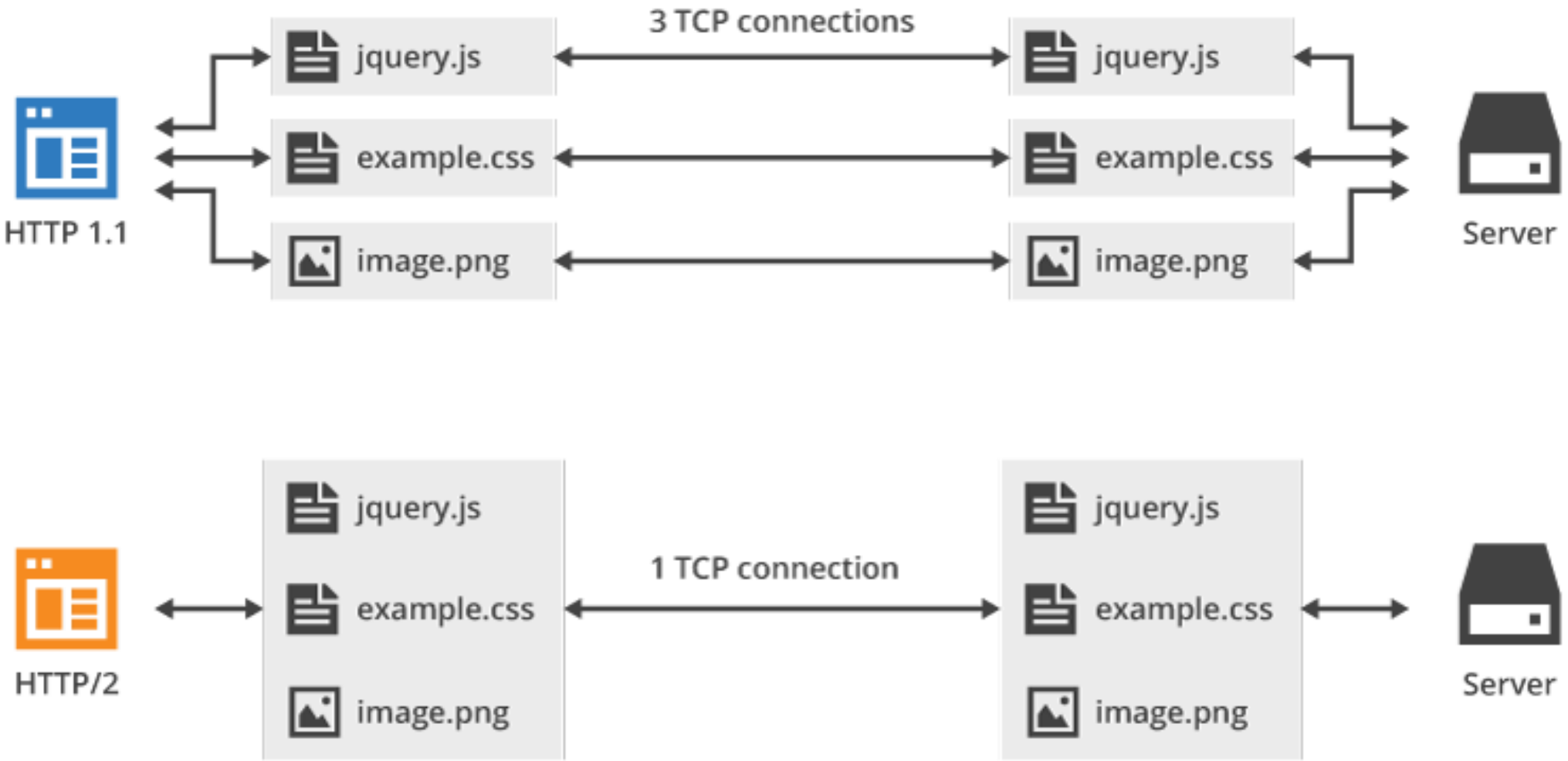




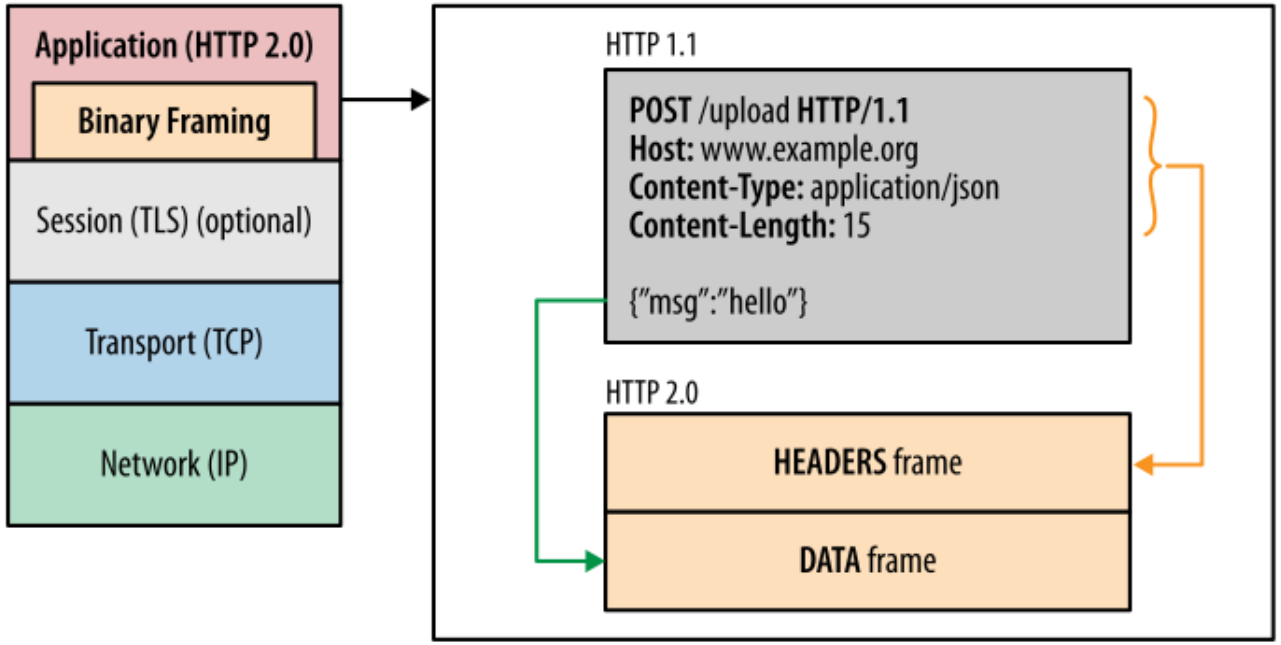
HTTP/2

- Major revision of the HTTP 1.1 protocol
- Fixed many of HTTP 1.1 issues
  - Arises because of the growth and adoption of HTTP to different uses
- Backward compatible with HTTP 1.1
- Web applications using HTTP/2 will not notice any differences
- Changes is mostly at the network and transport layer

HTTP/2 Request Multiplexing

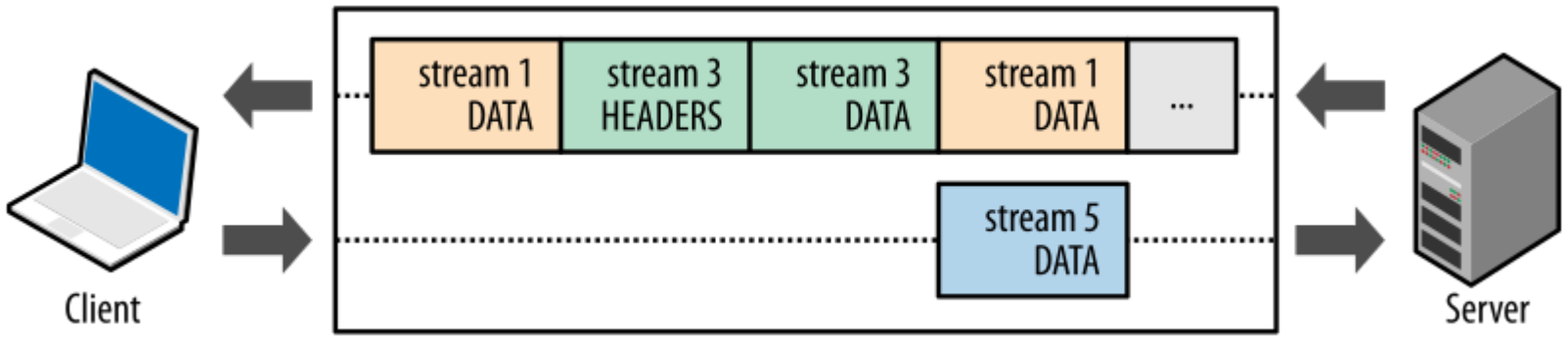


Multiplexing with Binary Frames



Messages from request and responses are broken into frames. These are sent to the client which will then reassemble the message.

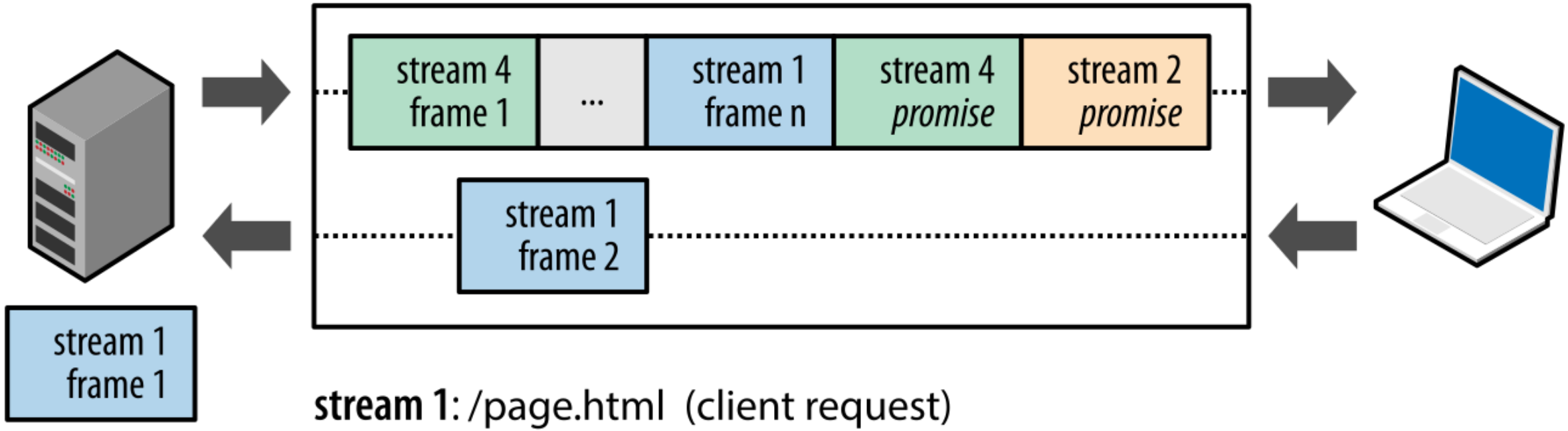
Multiple messages can be multiplex in this manner



Images from https://developers.google.com/web/fundamentals/performance/http2

Server Push

Server pushes resources to the client even before the client ask for it.
These resources are transported in PUSH_PROMISE frames



stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

Server Push - HTTP/2