2 The Hypertext Transfer Protocol

I.2

Application-layer Protocols

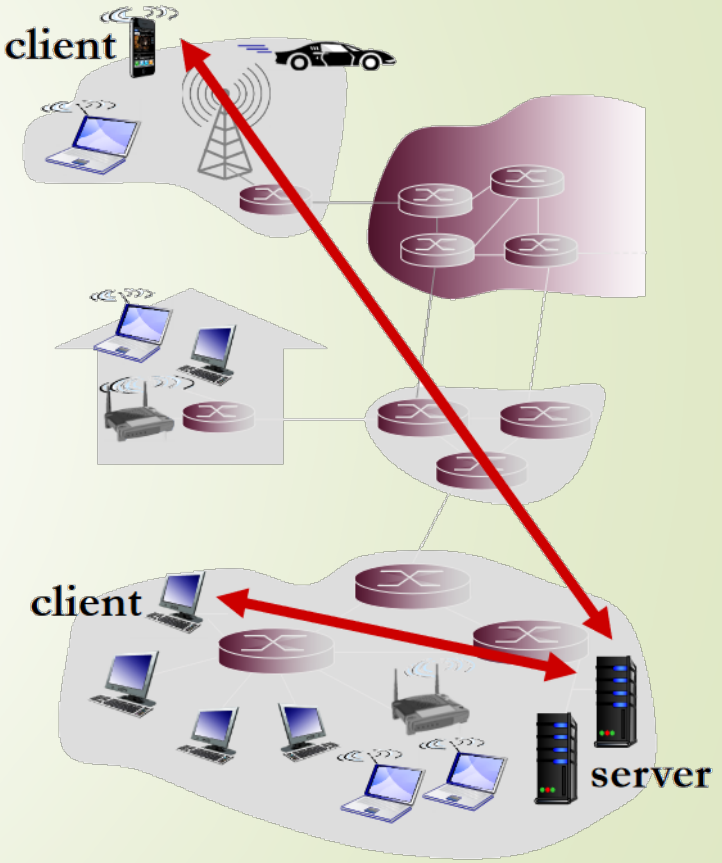
Requirement

- to allow apps running on different computers to communicate
  - system is called client-server or peer-to-peer depending on whether it relies on central control (at a server)
  - apps don't worry about low-level details of the network

Application-layer Protocols

Client/Server architecture

- Servers: handle requests
  - always powered on
  - usually reside in data centers, no peripherals
  - permanent IP addresses, usually have a DNS hostname
  - listen for requests from clients
- Clients: make requests
  - opposite of above, in every way
  - do not listen for unsolicited messages
  - only accept responses to their requests
  - do not communicate directly with other clients. Server must relay messages.



I.2

Application-layer Protocols

Differences

Server

- do not move
- location/address in a network is constant
- can listen for requests

Client

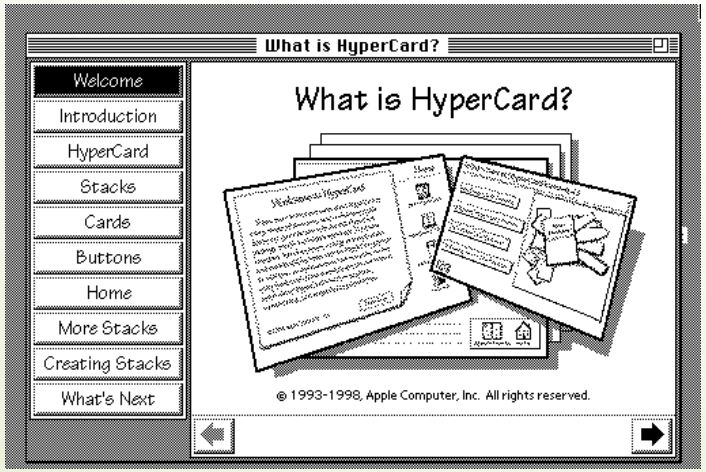
- can move around with users
- are difficult to find
  - thus, do not listen for requests
- send requests on behalf of user's apps, and listen briefly for a response from the one server that was contacted

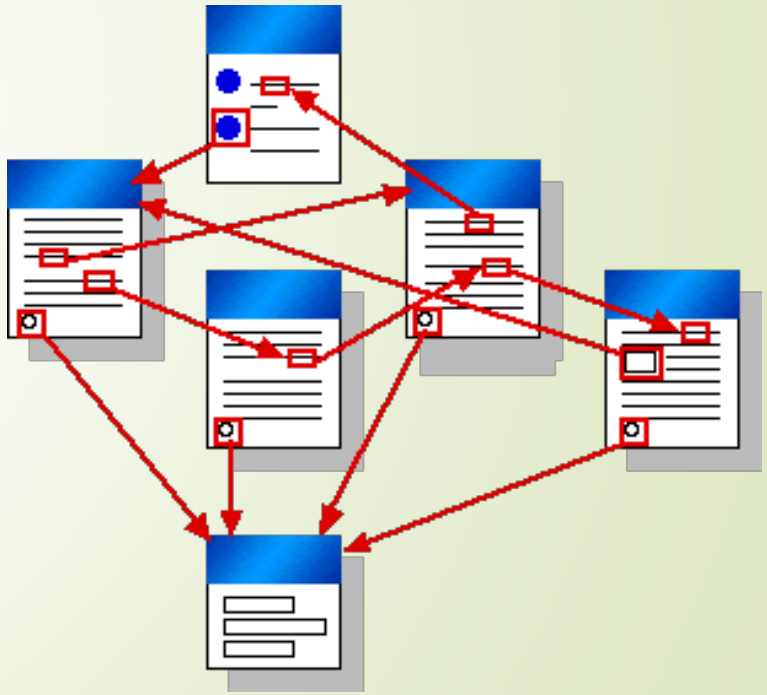
I.2

Hypertext

Definition

- text displayed on an electronic devices with references (hyperlinks) to other text that the reader can immediately access
- hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, keypress set, or screen touch





I.2

Hypertext Transfer Protocol

text based protocol

Characteristics of HTTP/1.1

- an application-level protocol for distributed, collaborative, hypermedia information systems
  - provide fundamental means of exchanging information and requesting services on the Web
  - on top of the TCP (Transmission Control Protocol)
- text-based protocol, human readable
- capabilities
  - contain meta-information about: data transferred , modifiers on the request/response semantics
  - stateless
    each request is self-contained – contains all info needed to give a response
  - support for hierarchical proxies, caching, persistent connections, virtual hosts

↳ http 1.1 / http 2.1

I.2

HTTP Communication

HTTP Communication Model

1. opening a TCP connection to a server
2. sending a request to the server
3. receiving a response from the server
   which includes the data of the requested page
4. closing the connection (optional)

I.2

Hypertext Transfer Protocol

Non-Persistent and Persistent Connections

- design decision
  should each request/response pair be sent over a separate TCP connection, or should all of the requests and their corresponding responses be sent over the same TCP connection?
  - non-persistent connections
    connections in which for each object a new connection for sending that object from source to destination has to be created
  - persistent connections
    connections in which for each object the same connection is used

Hypertext Transfer Protocol

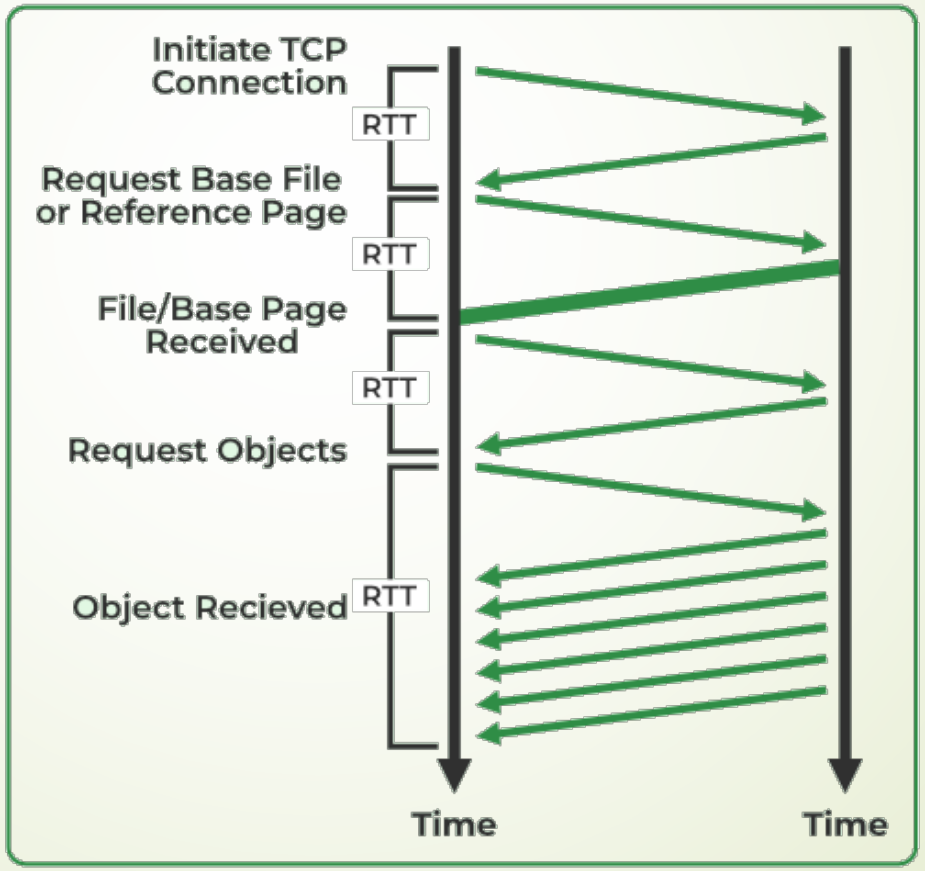
Non-Persistent and Persistent Connections

Persistent HTTP	Non-Persistent HTTP
The server leaves the connection open after sending a response.	Requires 2 RTTs per object.
Subsequent HTTP messages between the same client/server are sent over an open connection.	OS overhead for each TCP connection
The client sends requests as soon as it encounters a referenced object.	Browsers often open parallel TCP connections to fetch referenced objects.
As little as one RTT for all the referenced objects.	Here, at most one object can be sent over one TCP Connection.

single doc

Hypertext Transfer Protocol

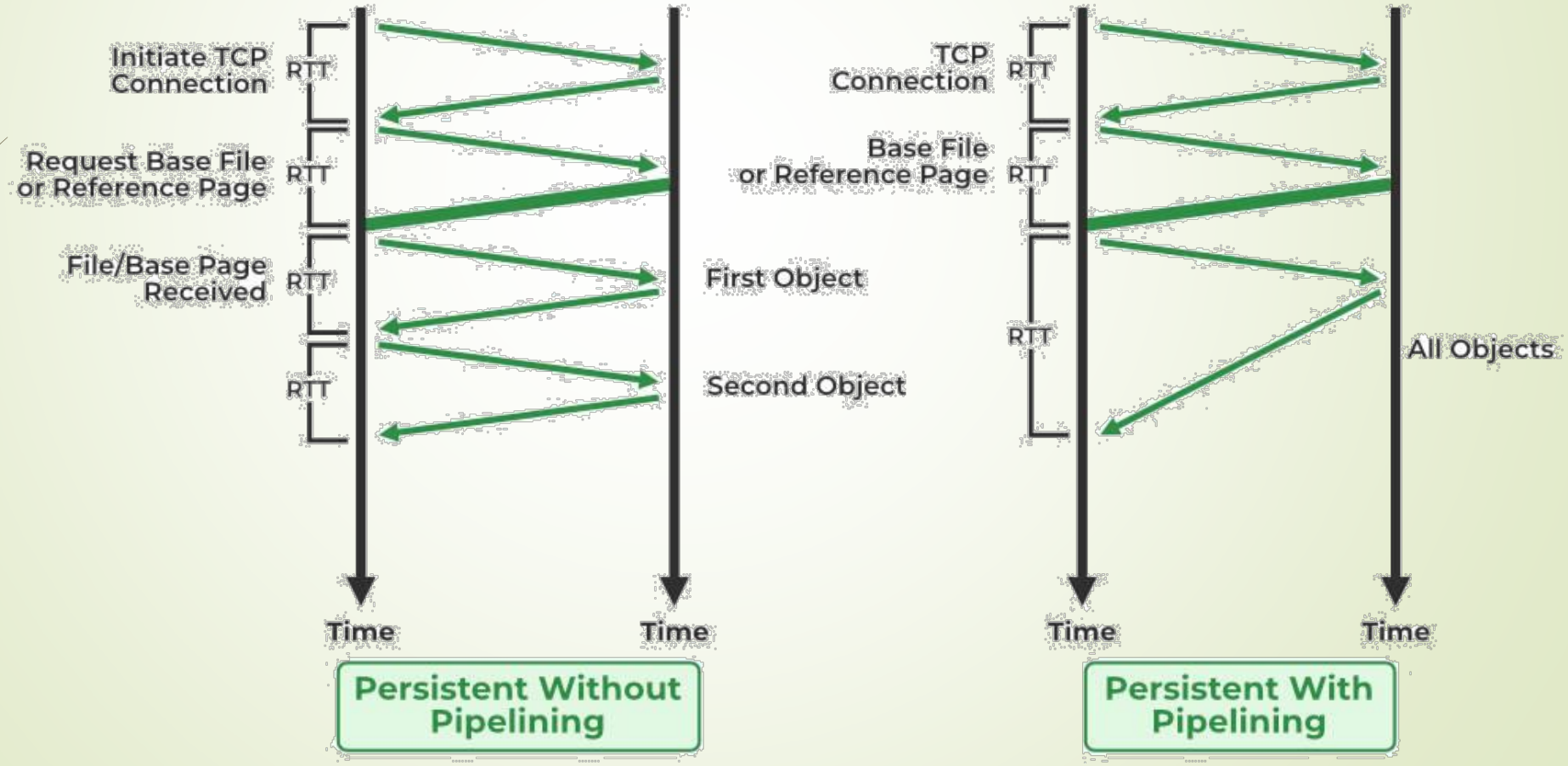
Non-Persistent and Parallel Connections



Hypertext Transfer Protocol

how to configure

Persistent Connections



I.2

HTTP Communication

HTTP Communication

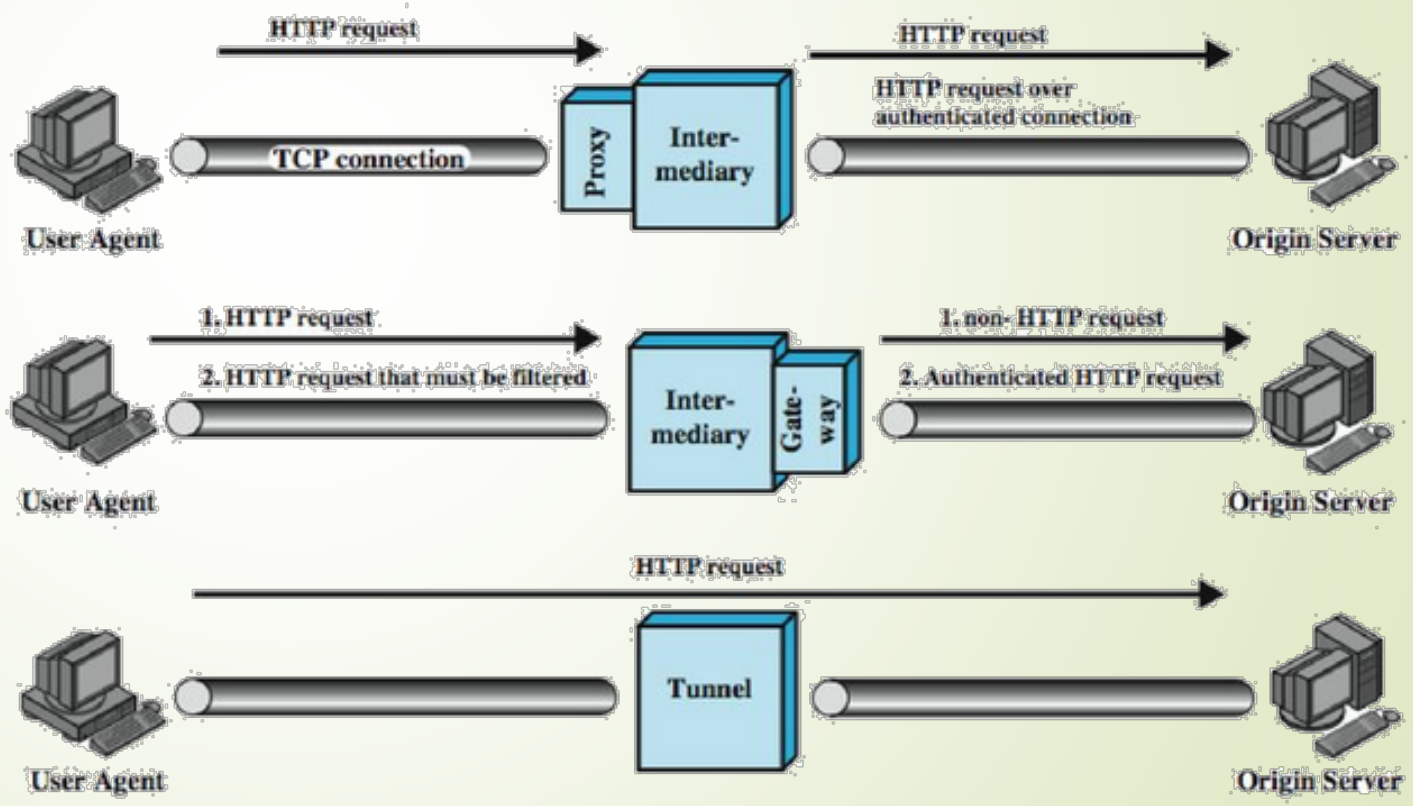
- opening a TCP-connection
  - the URL (web address) contains the name of the web server
  - the client (e.g. browser) asks a DNS server for the server's IP address
  - if IP address lookup fails: “unable to locate the server”
  - the client opens a TCP-connection to port 80 of this machine

Requests and Responses

gateway -> increases security

Intermediate HTTP Systems

- Origin server
- Proxy
- Gateway
- Tunnel



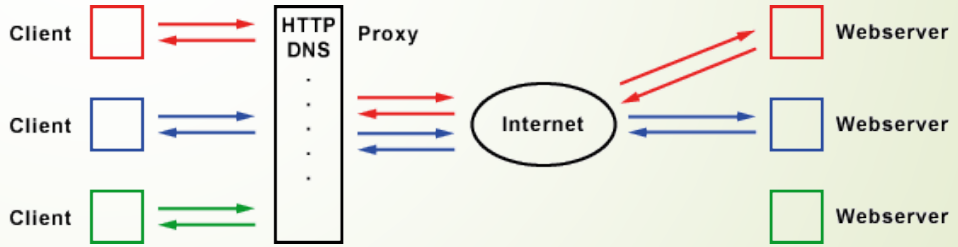
I.2

Requests and Responses

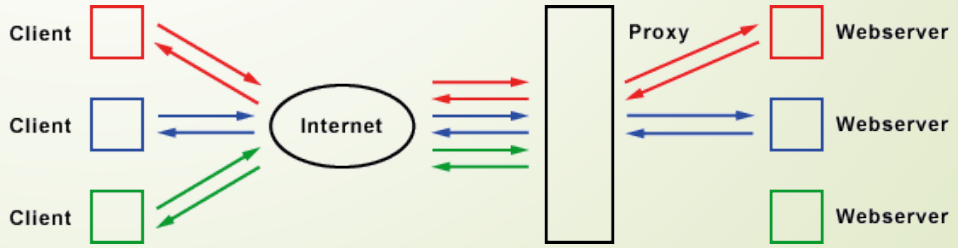
Proxy Server

- proxy refers to an instance authorized to carry out an action on behalf of someone else, and proxy servers deliver this in the online world
- proxy server acts as a gateway between users and the internet and prevents access to anyone outside the network

Standard Proxy



Reverse Proxy



I.2

Requests and Responses

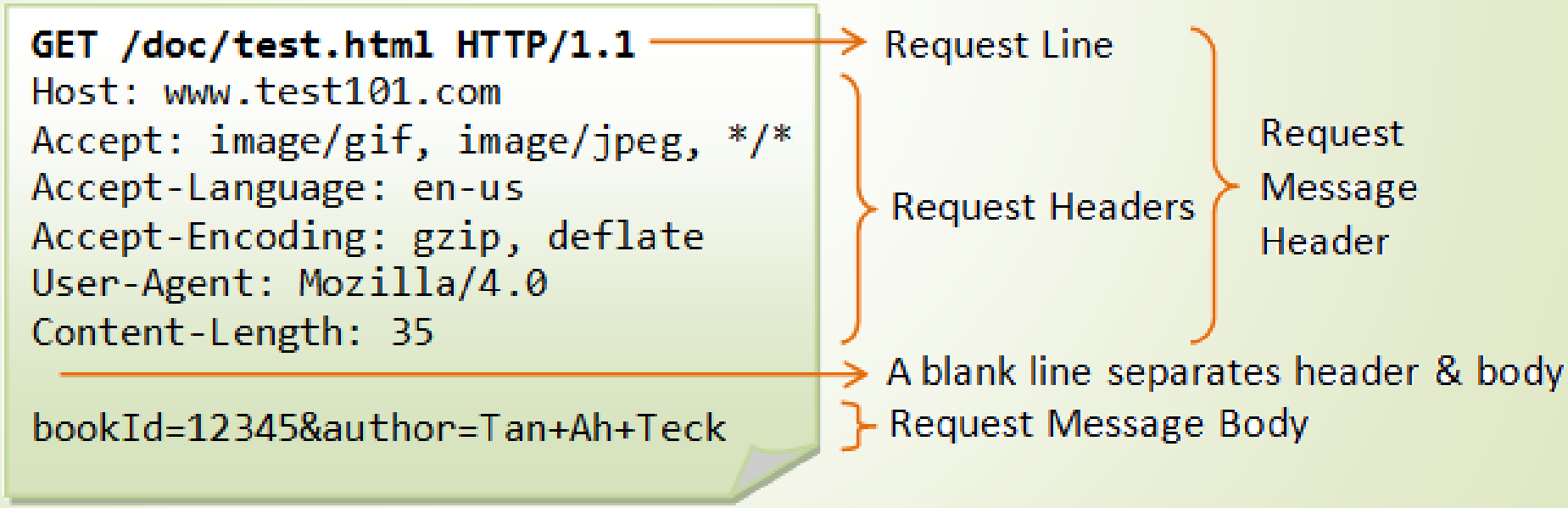
Proxy Server

- advantages
  - faster access to always the same data
  - cost savings on Internet data traffic
  - integration of virus and spam filters possible
  - enable users to hide their IP addresses to anonymize browsing activity
- disadvantages
  - cache coherence due to outdated content in the cache
  - data caching can lead to information misuse
  - not every application supports proxies
  - proxies are not available for every Internet protocol

HTTP Communication

HTTP Communication

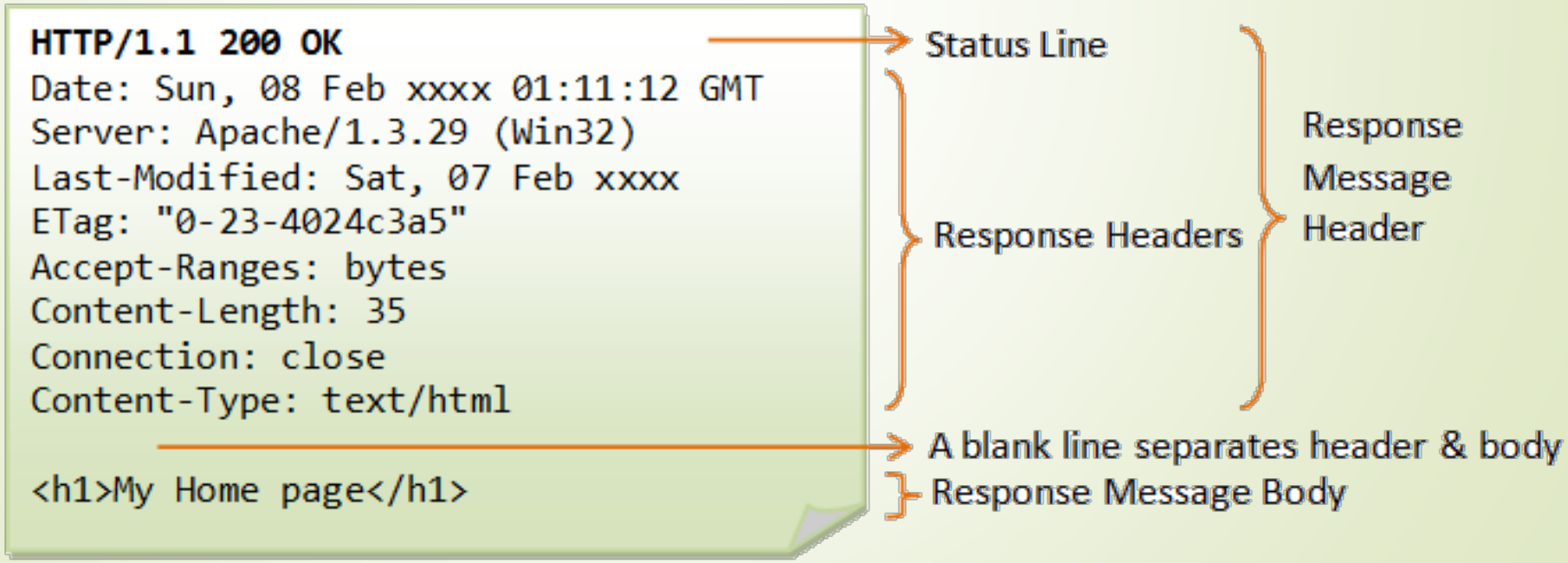
- request
  - done with a human-readable message



HTTP Communication

HTTP Communication

- response
  - server answers with the requested object

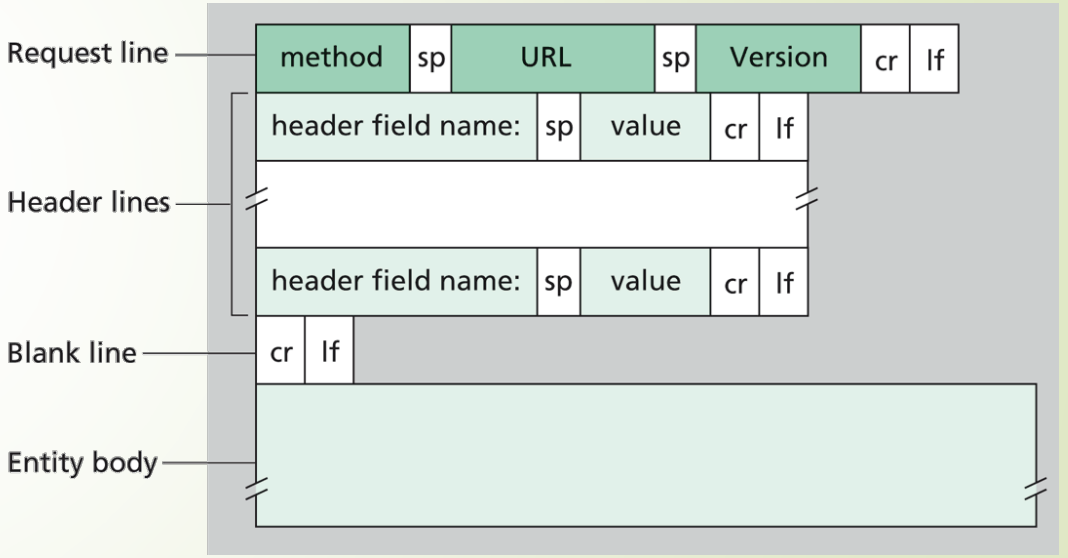


I.2

Requests and Responses

Request Format

- a command line
  - a request method
  - an identification of the resource to which the method should be applied
  - the HTTP-version of the request
- zero or more headers
- an empty line
- a body (entity, data) (optional)



I.2

Requests and Responses

Request

- resource identifications are:
  - an absolute path
  - an absolute URI (Uniform Resource Identifier)
    - a URL (“Uniform Resource Locator”, “web address”)
      specifies its location on a computer network and a mechanism for retrieving it
    - a URN (“Uniform Resource Name”)
      persistent, location-independent identifiers assigned within defined namespaces
      are globally unique and persistent over long periods of time, even after the resource
      which they identify ceases to exist or becomes unavailable

I.2

Requests and Responses

Request Methods

- **GET**: to request data
- **POST**: to post data to the server, and perhaps get data back, too
- **PUT**: to create a new document on the server
- **DELETE**: to delete a document
- **HEAD**: like GET, but just return headers

Response Codes

- 200 OK: success
- 301 Moved Permanently: redirects to another URL

Client errors (400–499):

- 403 Forbidden: lack permission
- 404 Not Found: URL is bad

Server errors (500-599):

- 500 Internal Server Error
- … and many more

I.2

Requests and Responses

Request Methods

- **TRACE**: to send back the request as data
- **OPTIONS**: sends back the methods that would be acceptable for the given URI
- **CONNECT**: to create an HTTP tunnel through a proxy server
  - by sending an HTTP CONNECT request, the client asks an HTTP Proxy server to tunnel the TCP connection to the desired destination
  - the server proceeds to make the connection on behalf of the client
  - once the connection has been established by the server, the proxy server continues to proxy the TCP stream to and from the client

I.2

Requests and Responses

Difference between POST and PUT

- **POST**
  the URL of the request identifies the resource that processes the entity body
- **PUT**
  the URL of the request identifies the resource that is contained in the entity body

Difference between GET and HEAD

- **GET**
  return the requested resource in the entity body of the response along with response headers
- **HEAD**
  return all the response headers in the GET response only

Requests and Responses

Uploading data: GET and POST

- Data are transferred from the client to the server which should be assigned to the given URI

- **GET**

  entity body is empty, input data – as (key, value)-pairs – is uploaded in URL field of request line

  http://site.com/form**?**first=jane**&**last=austen

- **POST**

  input is uploaded to server in entity body
  data not visible in the URL

I.2

Requests and Responses

General Headers

- date
  date and time when the request or response was constructed
- connection
  client/server can state whether they want to keep the TCP-connection after sending the request/response (keep-alive) or not (close)
- trailer
  used for chunked encoding to specify headers that will be sent after the body

I.2

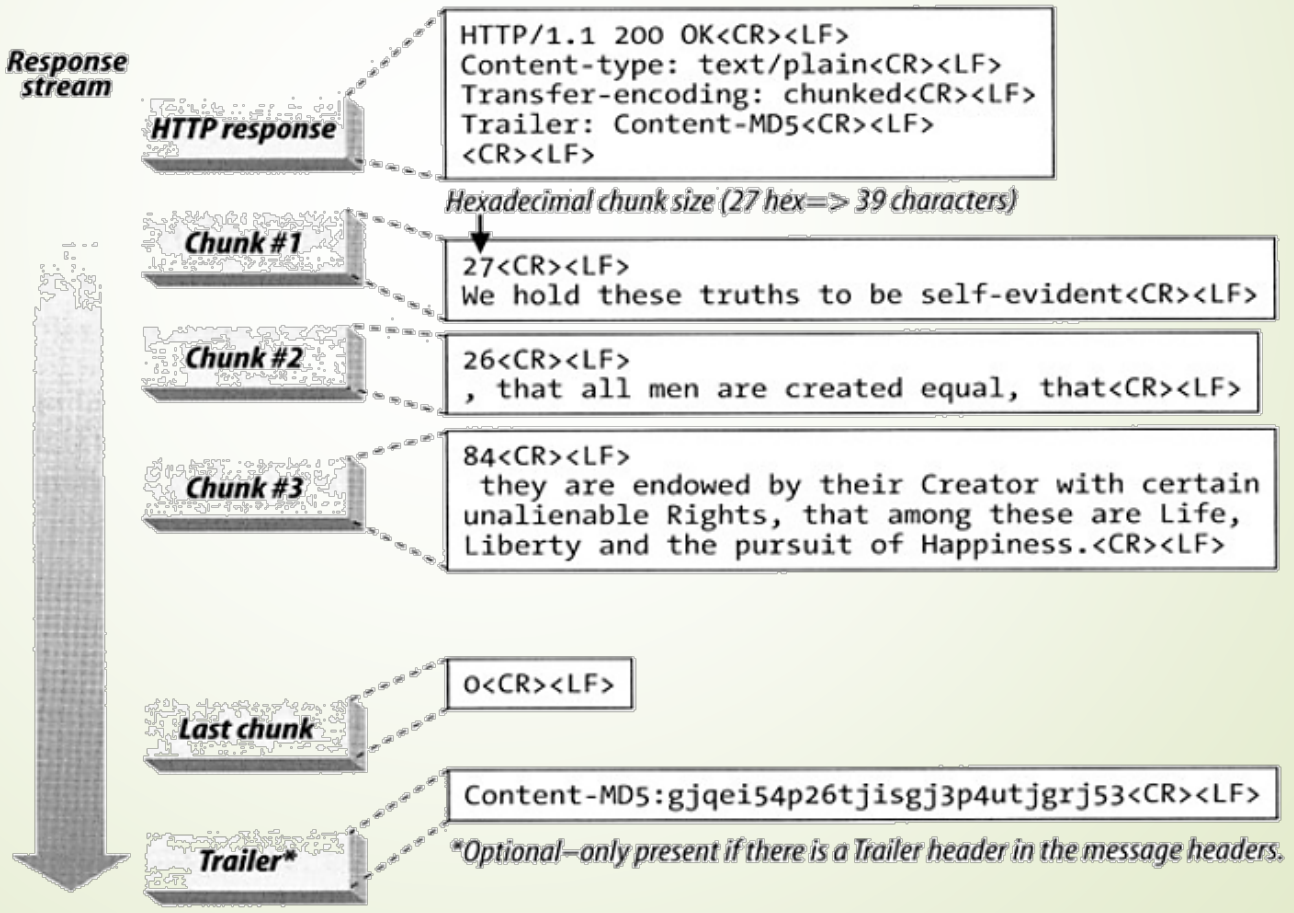
Requests and Responses

General Headers

- transfer-Encoding
  encoding of the body in order to safely transfer it
  - chunked — used to send the series of data in a chunk format
    - dividing (cutting) data into smaller "blocks."
    - are sent independently of one another, usually through a single persistent connection
    - have to mentioned the length of each chunk before sending the chunk
  - compress — compressed using the Lempel-Ziv-Welch (LZW) algorithm
  - deflate — compressed using the deflate compression algorithm
  - gzip — compressed using the Lempel-Ziv coding

Requests and Responses

General Headers



I.2

Requests and Responses

General Headers

- upgrade
  for changing to a different protocol
- via
  proxies between client and server add this header with their address to the request
- warning - deprecated
  contains information about potential problems with the contents of the message and can be applied to messages of any type

I.2

Requests and Responses

General Headers

- cache-control
  used to specify browser caching policies in both client requests and server responses
  policies include how a resource is cached, where it's cached and its maximum age before expiring

Requests and Responses

Request Headers

- headers for content negotiation
  - accept: acceptable media types
  - accept-encoding: acceptable encodings
  - te: acceptable transfer encodings
  - accept-charset: acceptable character sets
  - accept-language: acceptable languages
  - client can indicate preferred encodings by attaching Q values
    - accept-encoding: compress, gzip
    - accept-encoding: compress;q=0.5, gzip;q=1.0

I.2

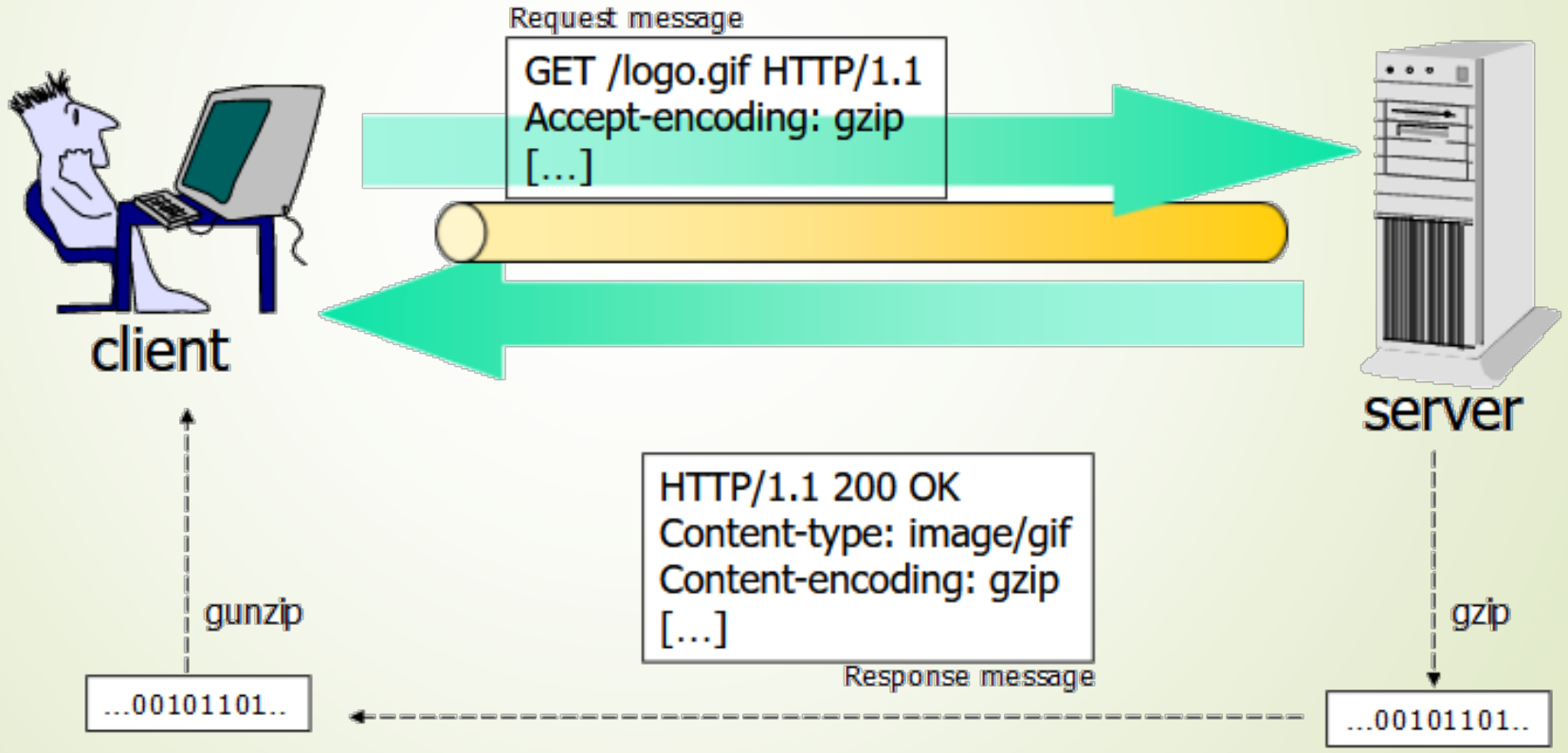
Requests and Responses

Request Headers

- accept:
  MIME types (multipurpose internet mail extensions)
  - contain multiple messages stuck together and sent as a single, complex message
  - each component is self-contained, with its own headers describing its contents; the different components are concatenated together and delimited by a string
- important MIME types
  - application/octet-stream
  - text/plain, text/css, text/html, text/javascript
  - image/jpeg, image/png, ...
  - audio/webm, video/webm

Requests and Responses

Request Headers



I.2

Requests and Responses

Request Headers

- expect

  the client uses the value “100-continue” to state that after sending the headers it waits for an acknowledgement before it will send the data

  - send along with POST requests to warn the server that they’re about to send a large payload
  - server can
    - decline, by sending back 401/405 and successively closing the connection
    - accept, by sending back 100 Continue, after which the client will send over the payload
    - ask the client to re-send the original request in an unaltered way (original headers + payload), by replying with a 417 Expectation Failed

I.2

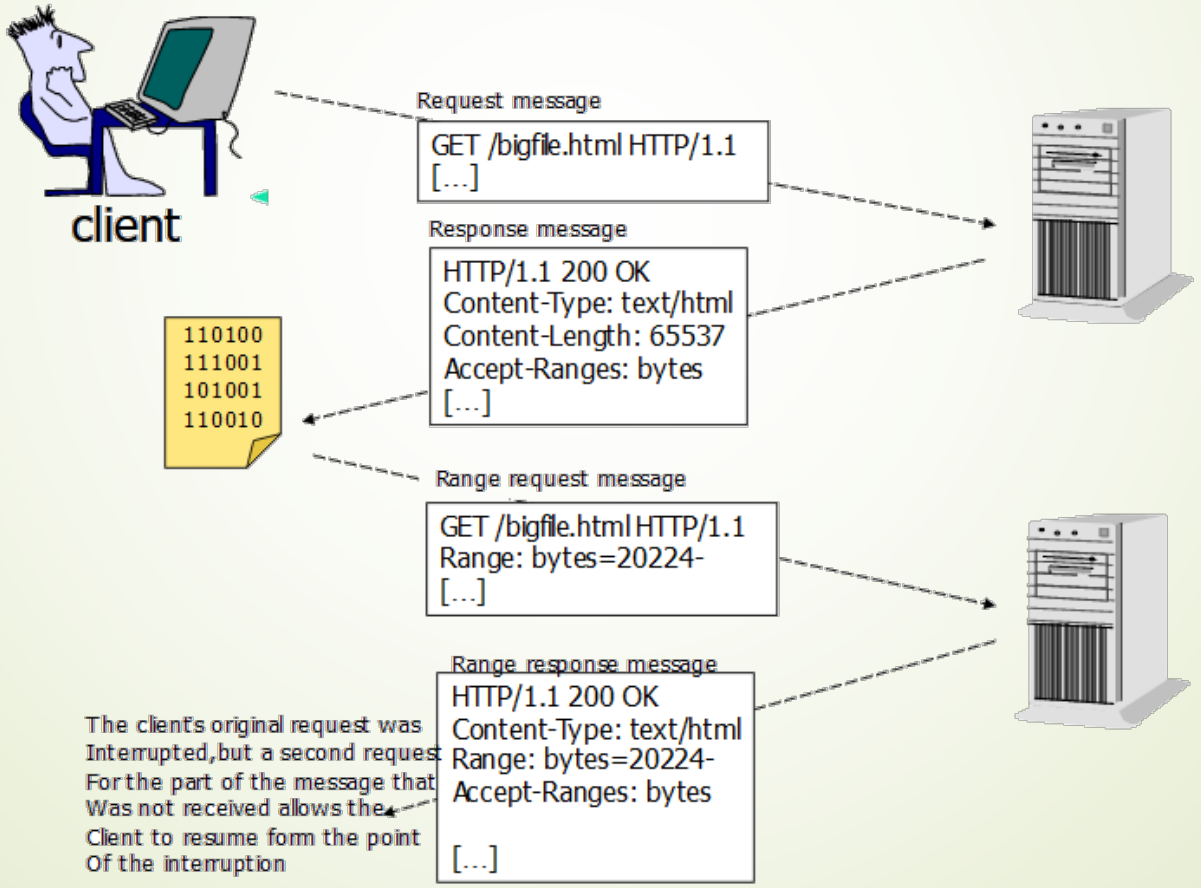
Requests and Responses

Request Headers

- host
  name of the web server
- from
  email-address of the user who is responsible for the request
- max-forwards
  number of proxies that can still relay the request on its way to the server
- user-agent
  information about the browser
- range
  only a part of the entity is requested

I.2 Requests and Responses

Request Headers



I.2

Requests and Responses

Request Headers

- headers for conditional requests
  - server will return the requested resource only if it matches one of the listed ETags for the GET and HEAD methods
  - ETag
    qualifier for determining version of a web page resource from a website's server
  - use of the If-Match HTTP header is for the GET and HEAD methods to combine with a range header

Requests and Responses

Request Headers

- headers for conditional requests
  - conditions
    - If-Match
    - If-Modified-Since
    - If-None-Match
    - If-Range
    - If-Unmodified-Since

Requests and Responses

Response Headers

- accept-ranges
  the server may use this in order to show whether it can process requests for partial entities
- age
  age of the response in seconds
- etag
  unique identifier of this version
- location
  URI of the entity (e.g. in case of a redirection)

Requests and Responses

Response Headers

- proxy-authenticate
  the proxy requires a password
  the header contains information about the method of authentication
- retry-after
  information when this request should be tried again
- server
  information about the server software
- vary
  criteria used in the content negotiation
  if there are several variants for the same URI, a proxy must know which Accept-headers were used in the selection

I.2

Requests and Responses

Response Headers

- www-authenticate
  authentication method for protected page
  - needs one or more challenge-response authentications to provide the required restrictions on the authentication method for the Request-URI
  - is generally used to provide responses after credentials are applied and for further restrictions to secure the webserver
  - when generated
    - the credentials shall be supplied by the user, i.e. an authorization header is used to supply these credentials to the web server, then the resource will be re-requested
    - the client will be providing the most protected challenge-response authentication required

I.2

Requests and Responses

Response Headers

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
  realm="http-auth@eg.com",
  qop="auth, auth-int",
  algorithm=RLA-123,
nonce="5xyz/qrs7YYtfNTCoB2SNkv/Ipf52BxDBzFZF4ViTo1b",
opaque="QTse/qeU824ImnopCqrsciny9TTkSqMAFRtzQOTo5tdN
```

- digest WWW-authenticate: method where the header contains a digest of the required browser
- realm: defines the username and password
- QOP: quality of protection is used on all HTTP Headers to provide a secured web server
- algorithm: a group of procedures required to generate a digest
- nonce: provides an original type of string that develops a direct response
- opaque: this is also considered as an asymmetric password authenticated key exchange (aPAKE) restriction

Requests and Responses

Entity Headers

- allow
  methods that are applicable to this entity
- content specification
  - content-encoding: (compression)
  - content-language
  - content-type
  - content-length
  - content-location
  - content-range
  - content-md5
- expires
- last-modified

I.2

Access Control

Access Control/Password-Protected Pages

- restriction by IP-address by web-server
  - knows IP-address of client
  - can be configured such that they permit or reject access to certain directories depending on the client's IP-address
    - example

```
order deny,allow
deny from all
allow from 134.176.28
```

    - in central configuration file
    - in file .htaccess in the resource's directory

I.2

Access Control

Access Control/Password-Protected Pages

- restricted users
  - grant access only to users that can identify themselves
  - required authentication method specified in the header WWW-Authenticate
    - browser opens a dialog box for entering username and password
    - browser sends a second request for the same URL/URI, which now includes an Authorization header
    - string is encoding of username and password

I.2

Access Control

API Keys

- objective: rate limiting
- to restrict access, users must first obtain an API key (a long string) which must be provided with any API request
- passed to server as key/value-pair

I.2

Cache Control

Cache Control

- goals:
  - reduction of the server load
  - reduction of the network load (used bandwidth)
  - faster answers on the client
- observation
  - relatively few pages get a relatively large percentage of all requests (uneven distribution)
  - after a page was visited, it is often visited again after a short time

I.2

Cache Control

Cache Control

- mechanisms of HTTP to ensure semantical transparency
  - server should define a minimum lifespan (expiration time) for the response
  - the possibility to validate an earlier response with the origin server
- normally, only responses to GET and HEAD requests may be buffered
- POST requests may only be buffered if the server has explicitly stated that it is cacheable

I.2

Cache Control

Cache Control

- exclusion of caching
  - response header **cache-control: no-cache**
    - forbid to use a cached response without revalidation
  - response header **cache-control: no-store**
    - forbid to store the response on any persistent media
  - response header **cache-control: private**
    - state that the response may only be cached for this single user
- server can specify an expiration date in the response

  expires: Sun, 24 Dec 2022 18:00:00 GMT

  server guarantees that until this date, the resource will not significantly change

I.2

Cache Control

Cache Control

- header **cache-control** can alternatively specify a maximal age in seconds
- caches can be configured such that they use expired responses and clients can request them
- the server can explicitly specify that the cache must respect the expiration time

  cache-control: must-revalidate

- if the server does not specify an expiration time, the cache can heuristically estimate it

I.2

Cache Control

Client Cache Control

- client can declare that it accepts responses that have already expired (e.g. up to one day ago):

  cache-control: max-stale=86400

- vice versa, the client can request that the response will not expire for a certain time (e.g. 1 hour):

  cache-control: min-fresh=86400

- cache-control instructions no-cache, no-store, max-age=... can also be used by the client

- client can request that the origin server is asked, and it does not get a cached copy or that it is interested only in responses from the cache

  cache-control: max-age=0 **cache-control: only-if-cached**

I.2

Cache Control

Validating Cache Entries

- what happens if the response has expired
  - check validity of response by a conditional GET-request
  - example

    last-modified: Sun, 23 Oct 2022 18:00:00 GMT

    cache can include the following header in the request

    if-modified-since: Sun, 23 Oct 2022 18:00:00 GMT

    responses
    - server sends the status code 304 "Not Modified", but does not include the contents of the file
    - the server sends the complete response

I.2

Cache Control

Validating Cache Entries

- ETags - Entity Tag Validators – replaces date of last modification
- tag identifies a unique version of the resource
- if the resource changes, its ETag must change
- again, there are conditional requests

If-None-Match: "60304-46b-39168772"

I.2

State Management

Keeping State Information

- done with "Cookies", which are pieces of data that
  - the server sends to the client, and
  - the client then basically includes with all future requests to the same server
- a cookie can e.g. contain a user number or session number
- although it is possible to implement something that looks like a user session with cookies (or unique numbers in URIs), one must be aware of certain differences

I.2

Python Implementations

HTTP module

- a standard library package that collects several modules for working with HTTP
- http.client is a low-level HTTP protocol client
  - for high-level URL opening use urllib.request
- http.server contains basic HTTP server classes based on socketserver
- http.cookies has utilities for implementing state management with cookies
- http.cookiejar provides persistence of cookies

I.2

Python Implementations

Requests

```python
from requests import get


response = get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:php'},
    headers={'Accept': 'application/vnd.github.v3.text-match+json'},
)


# View the new `text-matches` array which provides information
# about your search term within the results
json_response = response.json()
repository = json_response['items'][0]
print(f'Text matches: {repository["text_matches"]}')
```

I.2

Python Implementations

HTTPX

- fully featured HTTP client for Python 3, which provides sync and async APIs, and support for both HTTP/1.1 and HTTP/2
  - broadly requests-compatible API
  - Keep-Alive & Connection Pooling
  - sessions with Cookie persistence
  - browser-style SSL verification
  - Basic/Digest Authentication
  - Elegant Key/Value Cookies
  - Automatic Decompression
  - Automatic Content Decoding
  - Unicode Response Bodies
  - Multipart File Uploads
  - HTTP(S) Proxy Support
  - Connection Timeouts
  - Streaming Downloads
  - .netrc Support
  - Chunked Requests

I.2

Python Implementations

urllib

- urllib is a package that collects several modules for working with URLs:
  - urllib.request for opening and reading URLs
  - urllib.error containing the exceptions raised by urllib.request
  - urllib.parse for parsing URLs
  - urllib.robotparser for parsing robots.txt files

I.2

HTTP/2

Expectations

- substantially and measurably improve end-user perceived latency in most cases, over HTTP/1.1 using TCP
- address the "head of line blocking" problem in HTTP
- not require multiple connections to a server to enable parallelism, thus improving its use of TCP, especially regarding congestion control
- retain the semantics of HTTP/1.1, leveraging existing documentation, including (but not limited to) HTTP methods, status codes, URIs, and header fields
- clearly identify any new extensibility points and policy for their appropriate use

I.2

HTTP/2

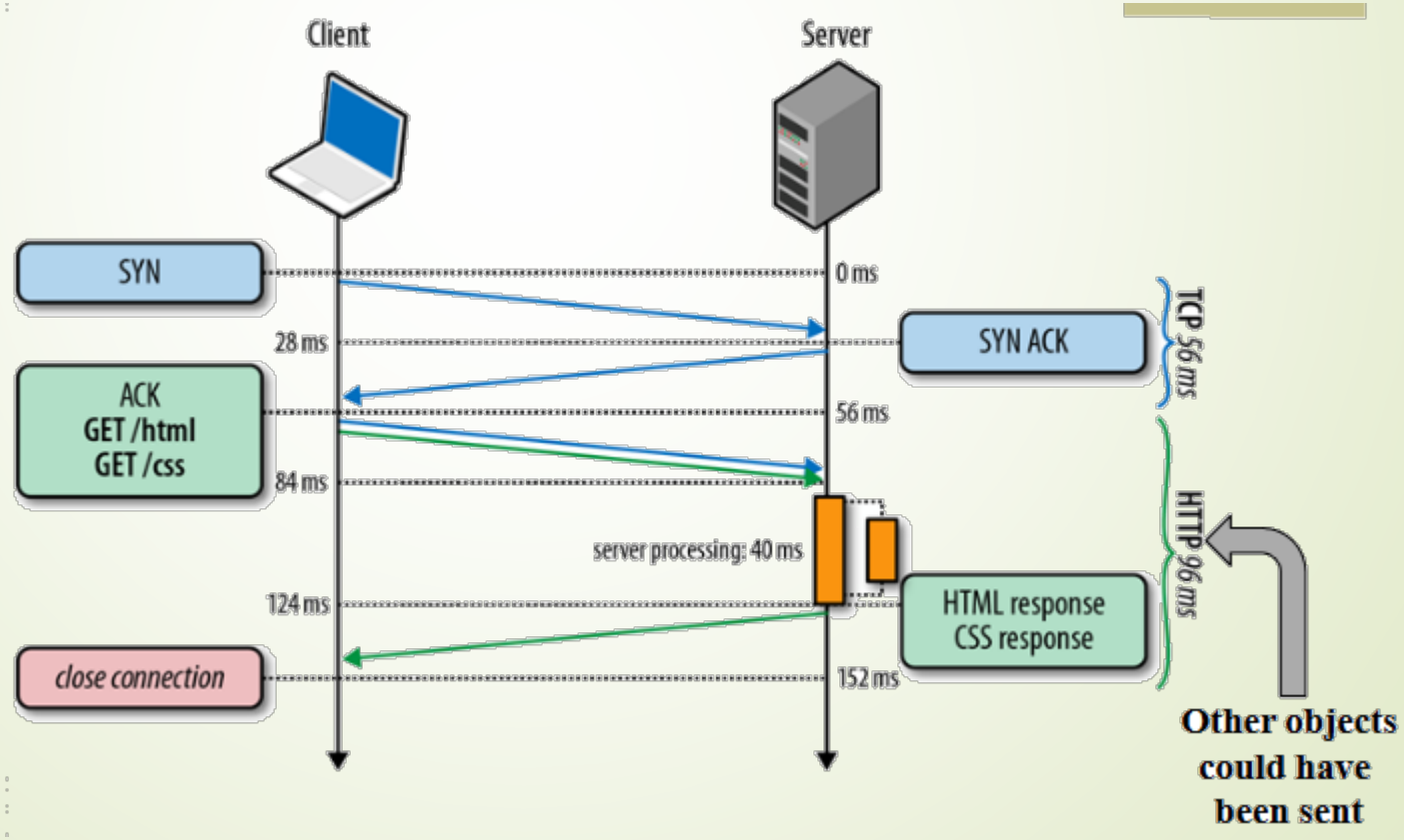
Expectations

- substantially and measurably improve end-user perceived latency in most cases, over HTTP/1.1 using TCP
- address the "head of line blocking" problem in HTTP
- not require multiple connections to a server to enable parallelism, thus improving its use of TCP, especially regarding congestion control
- retain the semantics of HTTP/1.1, leveraging existing documentation, including (but not limited to) HTTP methods, status codes, URIs, and header fields
- clearly identify any new extensibility points and policy for their appropriate use

HTTP/2

Head-of-line Blocking

- "When a single (slow) object prevents following objects from making progress"



I.2

HTTP/2

Features

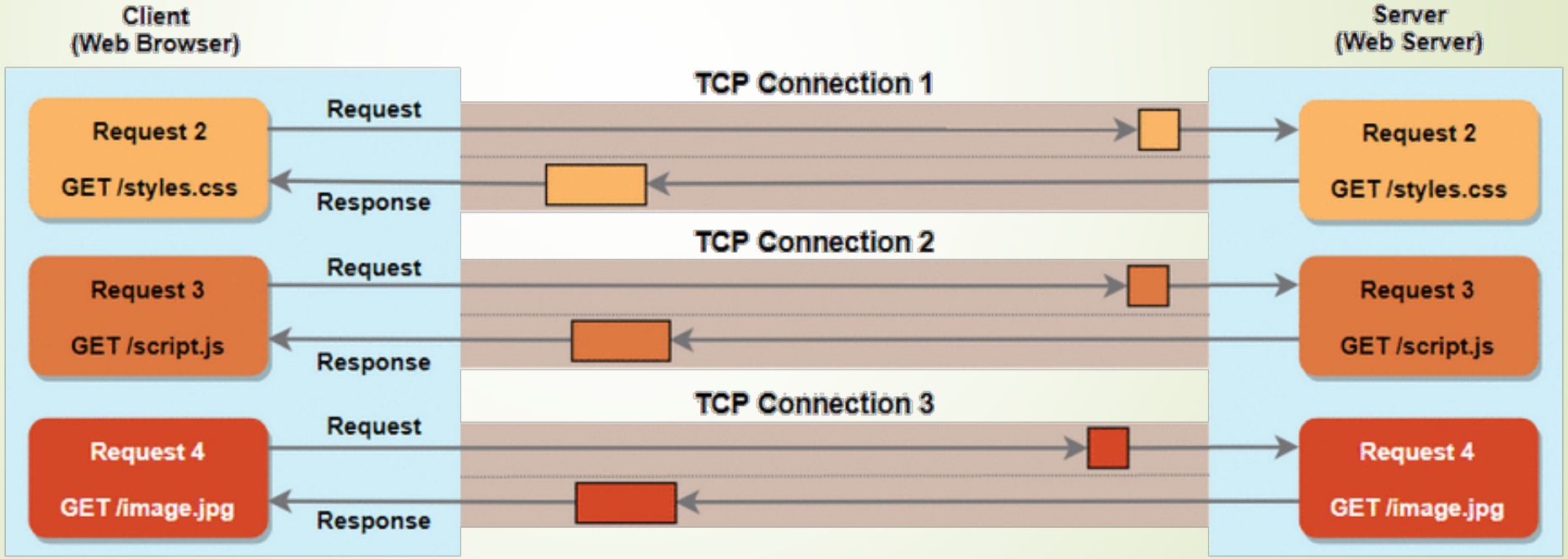
- HTTP headers are compressed
- responses are multiplexed over single TCP connection
  - server can send response data whenever it is ready
  - "Fast" objects can bypass slow objects – avoids HOL blocking
  - fewer handshakes, more traffic (help cong. ctl., e.g., drop tail)
- multiplexing uses prioritized flow controlled streams
  - urgent responses can bypasses non-critical responses
  - multiple parallel prioritized TCP connections

I.2

HTTP/2

Feature "Multiplexing"

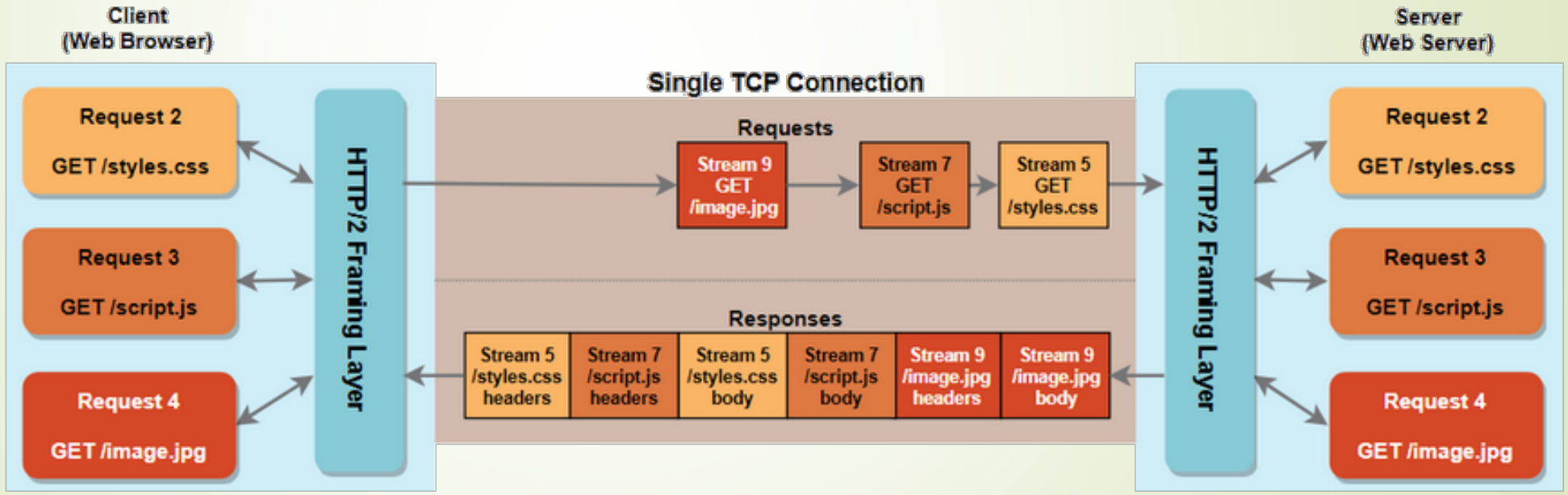
- multiple HTTP/1 requests in parallel required multiple TCP connections



HTTP/2

Feature "Multiplexing"

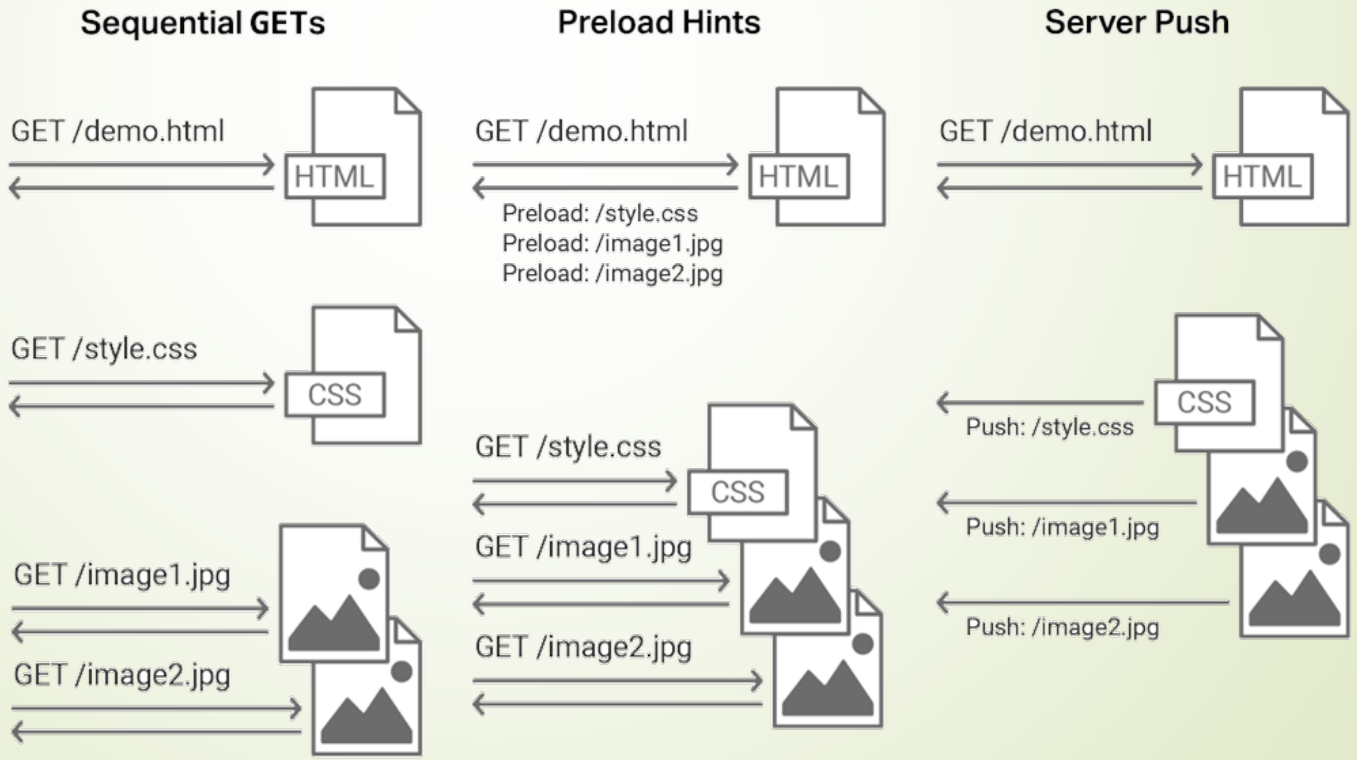
- requesting 3 resources across a multiplexed HTTP/2 connection



HTTP/2

Feature "PUSH"

- a PUSH features allows server to push embedded objects to the client without waiting for a client request



I.2

HTTP/2

Feature "PUSH"

- Server can “push” objects that it knows (or thinks) the client will need
  - avoids delay of having client parse the page and requesting the objects (> RTT)
- but what happens if object is in the client cache
  - server sends PUSH_PROMISE before the PUSH
  - client can cancel/abort the PUSH
- how does server know what to PUSH?
  - very difficult problem with dynamic content
  - Javascripts can rewrite web page – changes URLs