Web Application Concept

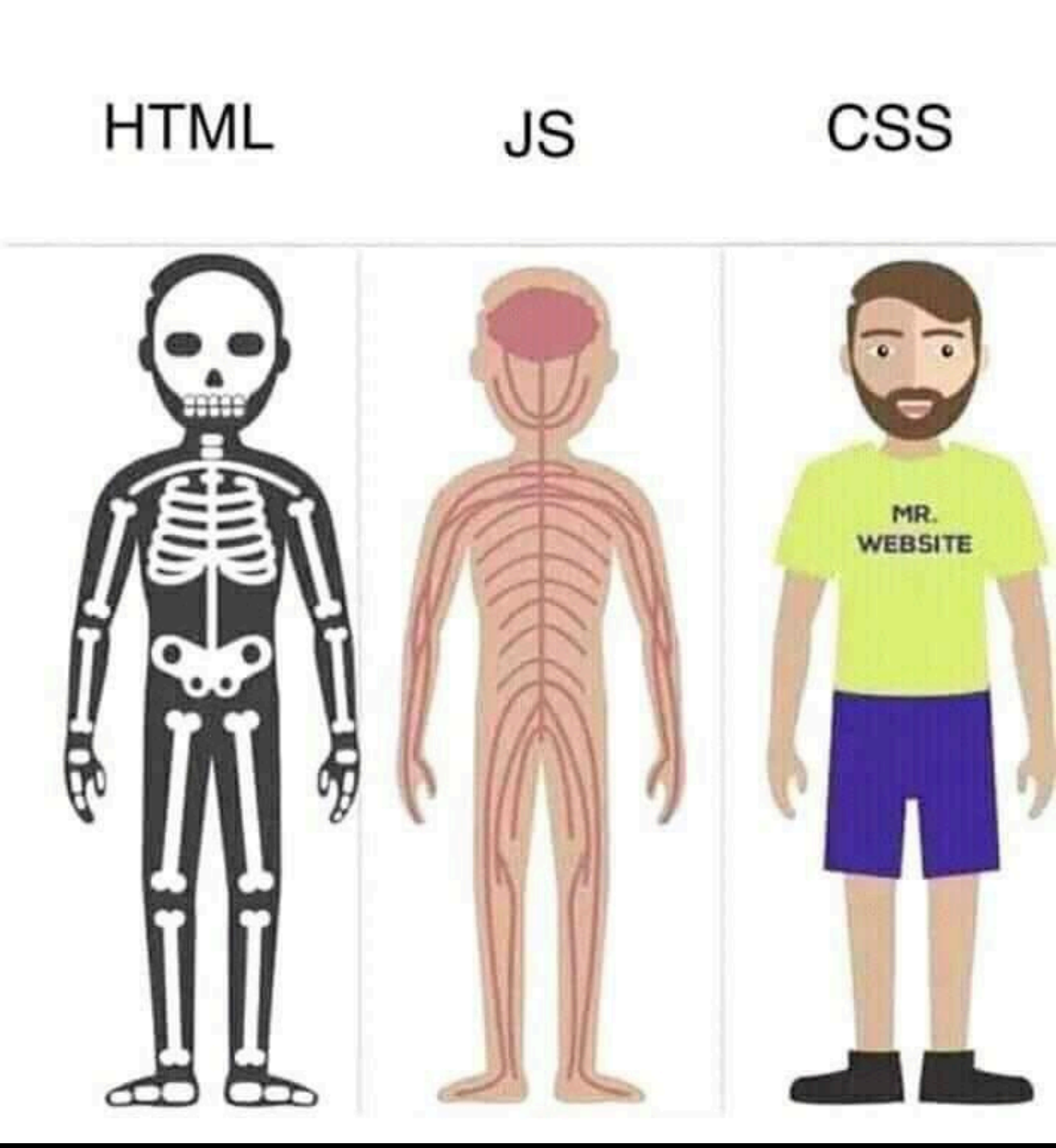
Cyrus Wong

"A web application or web app is any software that runs in a web browser. It is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a web browser to render the application."

- Wikipedia

Web applications

- Two main parts
  - Server
    - Python
  - Client (Browser)
    - HTML,
    - JavaScript and
    - CSS



Web applications

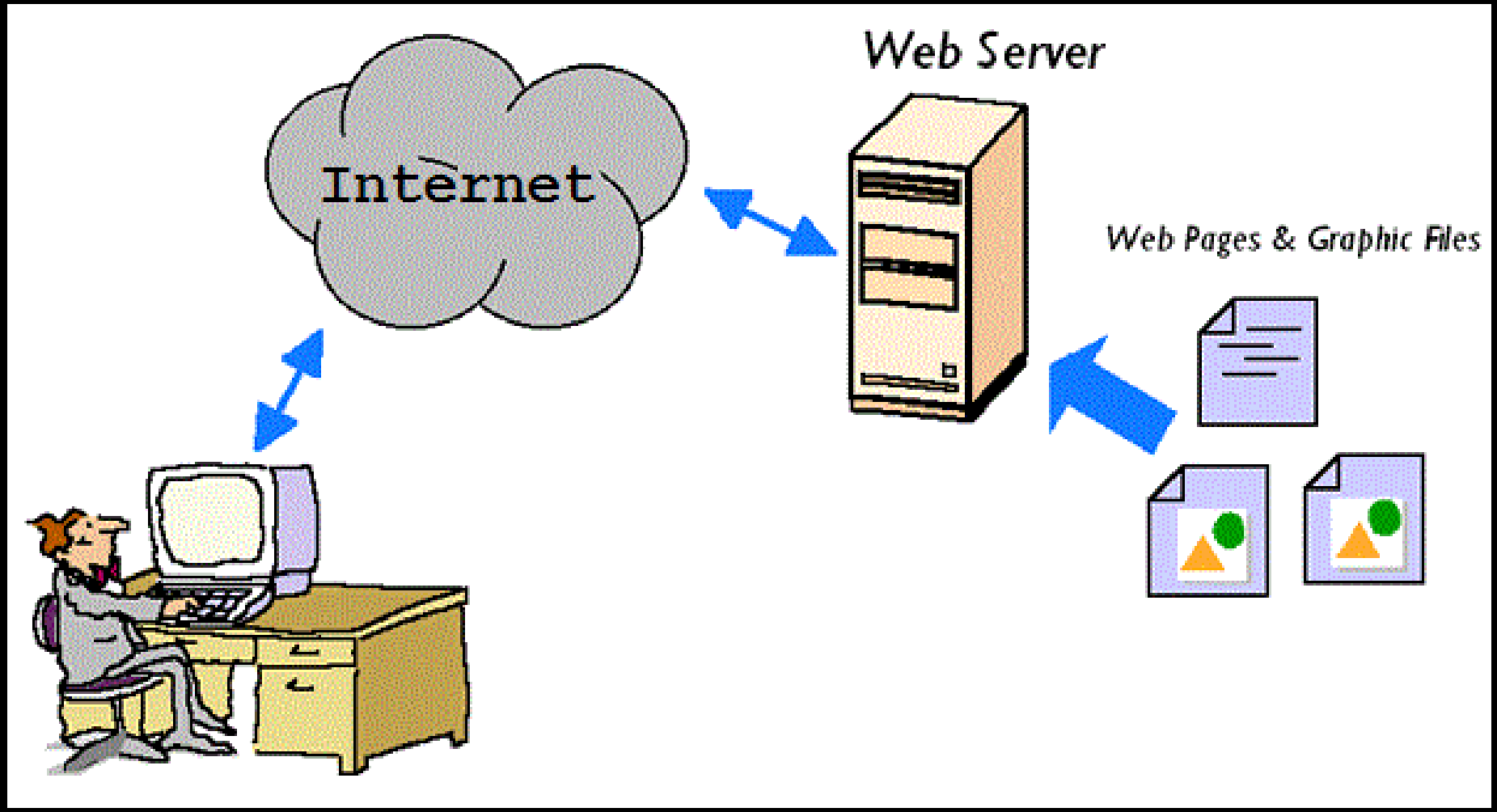
- So-called web app
- Is a Software running in web browser
- Created by browser-supported programming language (e.g. JavaScript, HTML and CSS)
- Behaviors depends on how web browsers render

HTTP Protocol

- HTTP (Hypertext Transfer Protocol) is the system Clients and Servers interact via internet

- Normal Flow:
  Type a URL into your client browser (e.g. http://www.vtc.edu.hk), your browser requests the home page from the web server and the web server sends the web page (in HTML) back

HTTP Protocol



Client-Side / Server-Side Programming

Client-Side Programming is good in:

- Checking and validation of user's information before sending to the server.
- Responding user's actions and event performed on the browser.

Server-Side Programming is good in:

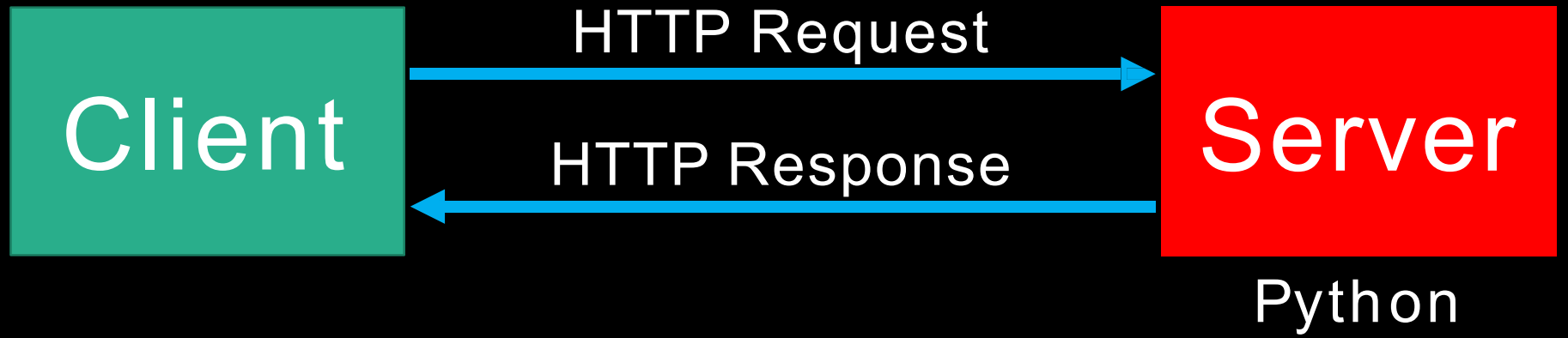
- Adding up-to-date information using database on the server.
- Controlling the content of a web page under different situations (states), e.g. login state, administrator state, etc.

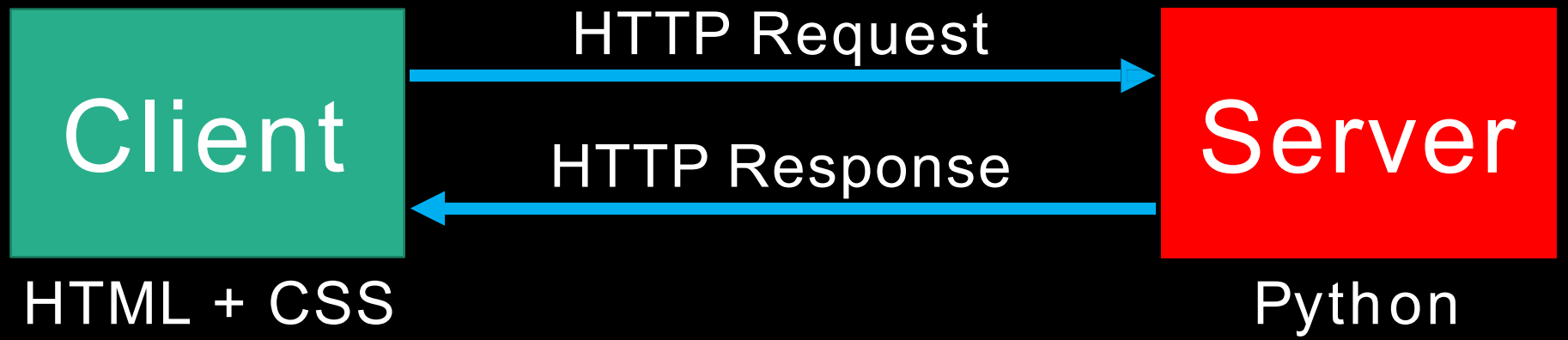
Client

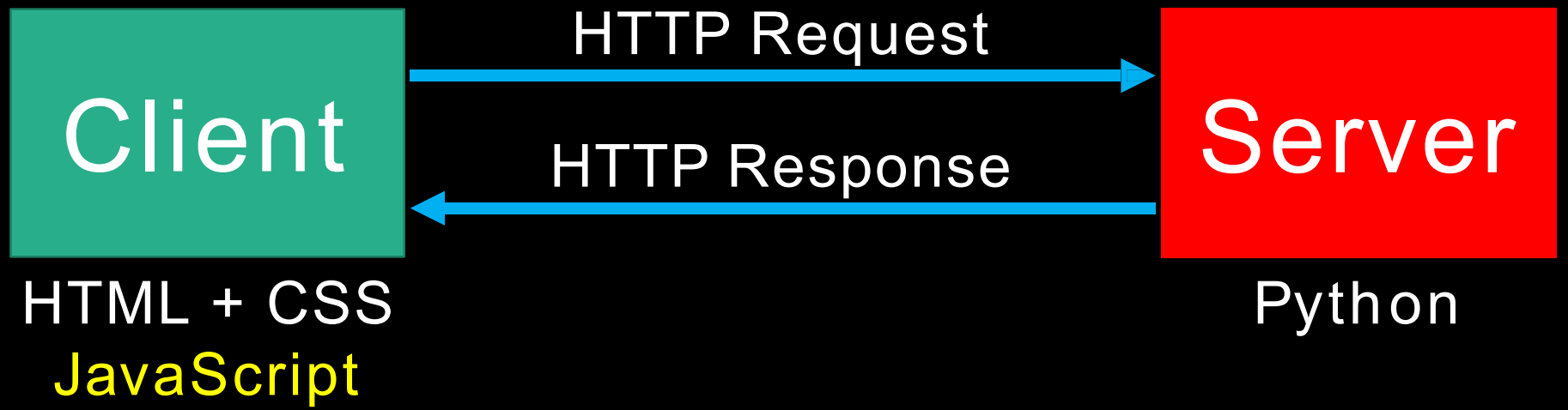
Server


Client
HTTP Request
Server


Client
HTTP Request
Server
Python


Client
HTTP Request
HTTP Response
Server
Python


Client
HTTP Request
HTTP Response
Server
HTML + CSS
Python


Client
HTTP Request
HTTP Response
Server
HTML + CSS
JavaScript
Python

HTML

HTML Tags

- <h1>, <h2>, ..., <h6>
- , <ol>,
- <img>
- <a>
- <table>
- <form>
- <b>, <i>
- <p>

DOM

DOM

- The *Document Object Model*, a way of considering more visually the natural nesting structure that HTML seems to take on.

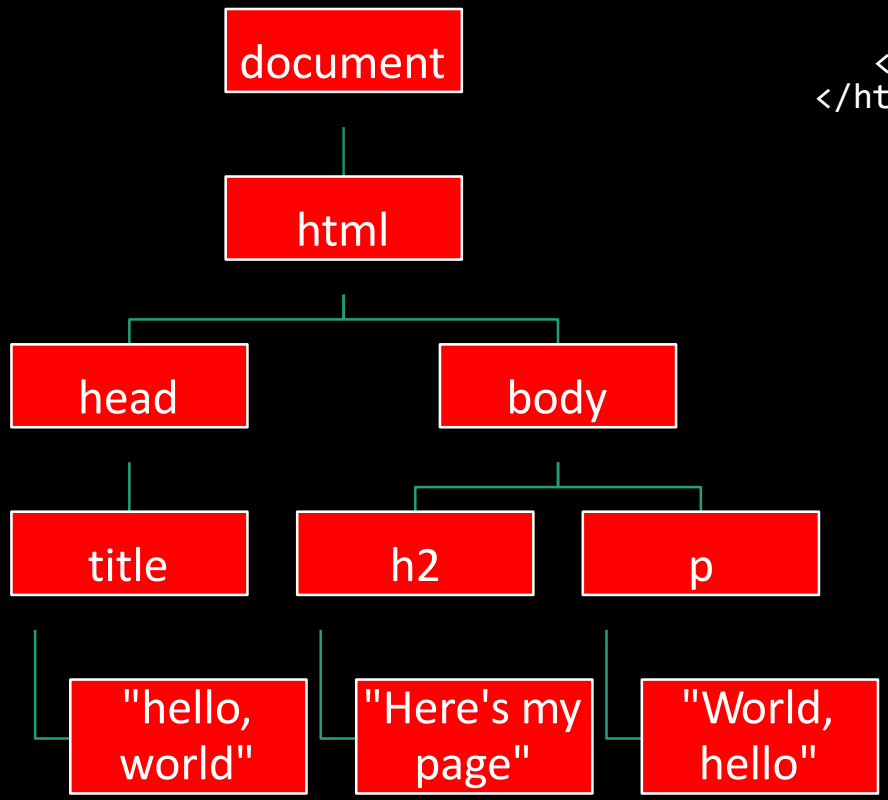
DOM

- The *Document Object Model*, a way of considering more visually the natural nesting structure that HTML seems to take on.

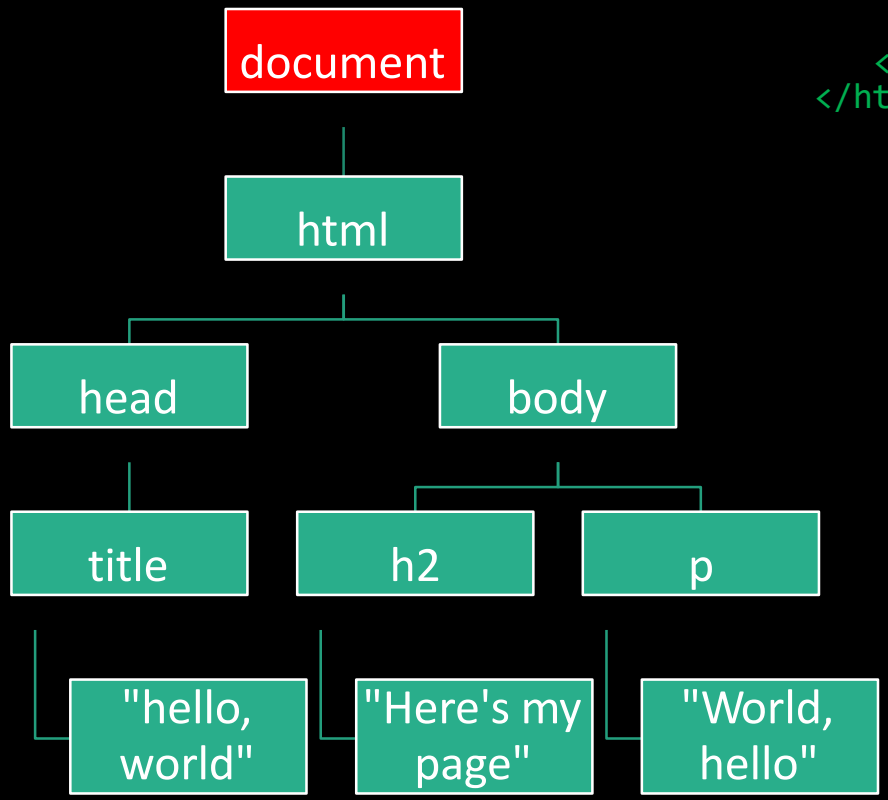
- Later, we'll be able to leverage the power of this document object to style and manipulate our sites programmatically.

```
<!DOCTYPE
html>
<html>
   <head>
      <title>Hello,
      world</title>
   </head>
   <body>
      <h2>Here's my
      page</h2>
      <p>World,
      hello</p>
   </body>
</html>
```

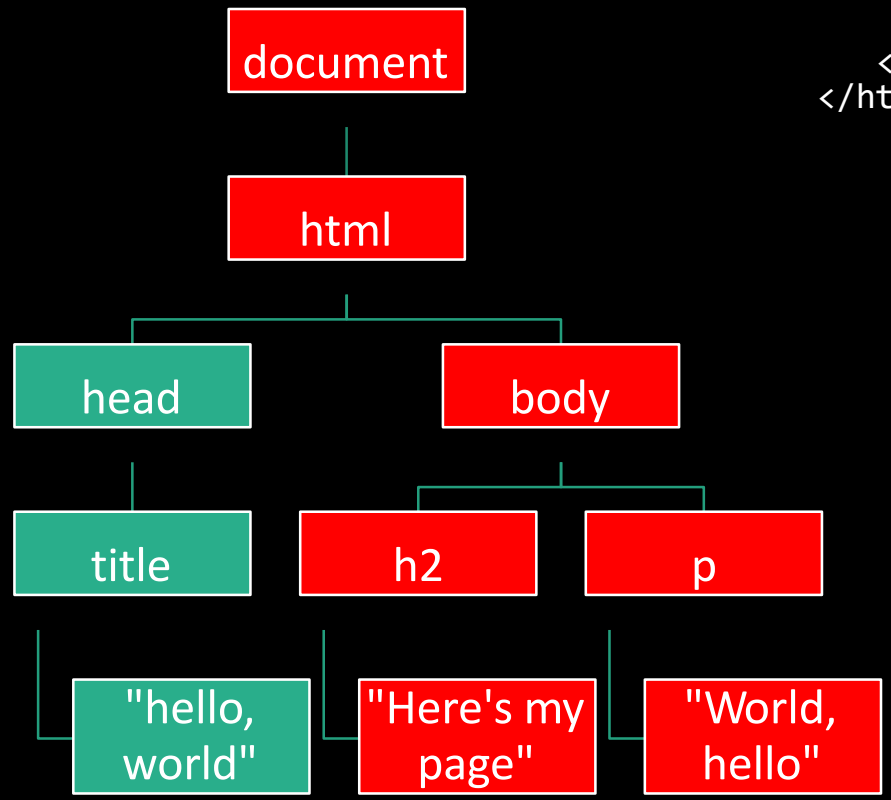
```
<!DOCTYPE
html>
<html lang="en">
    <head>
        <title>Hello,
        world</title>
    </head>
    <body>
        <h2>Here's my
        page</h2>
        <p>World,
        hello</p>
    </body>
</html>
```



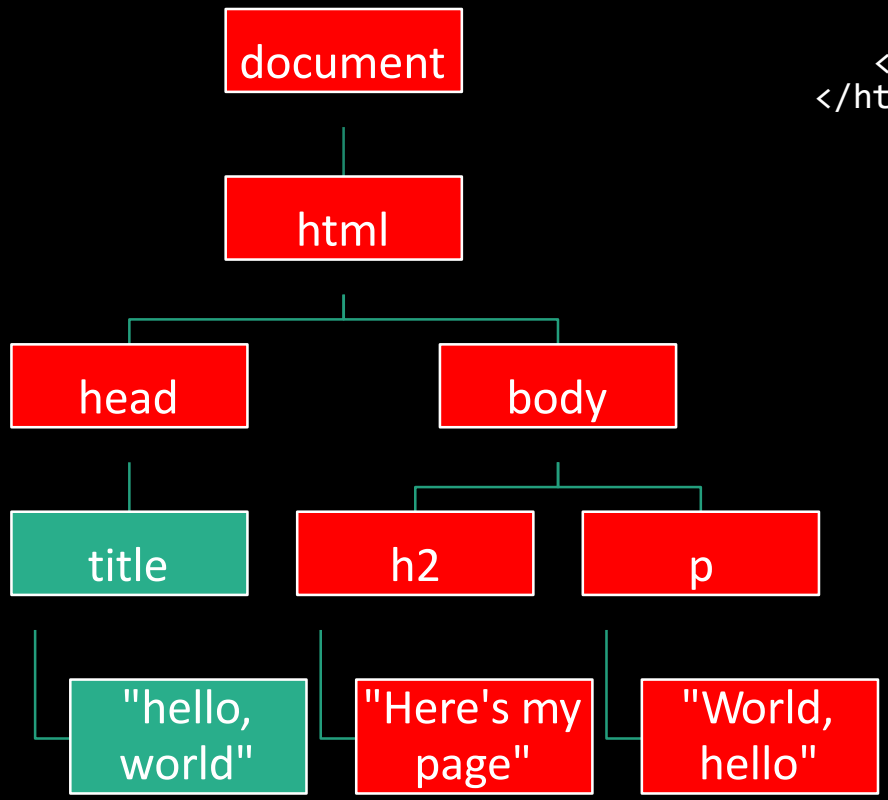

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```



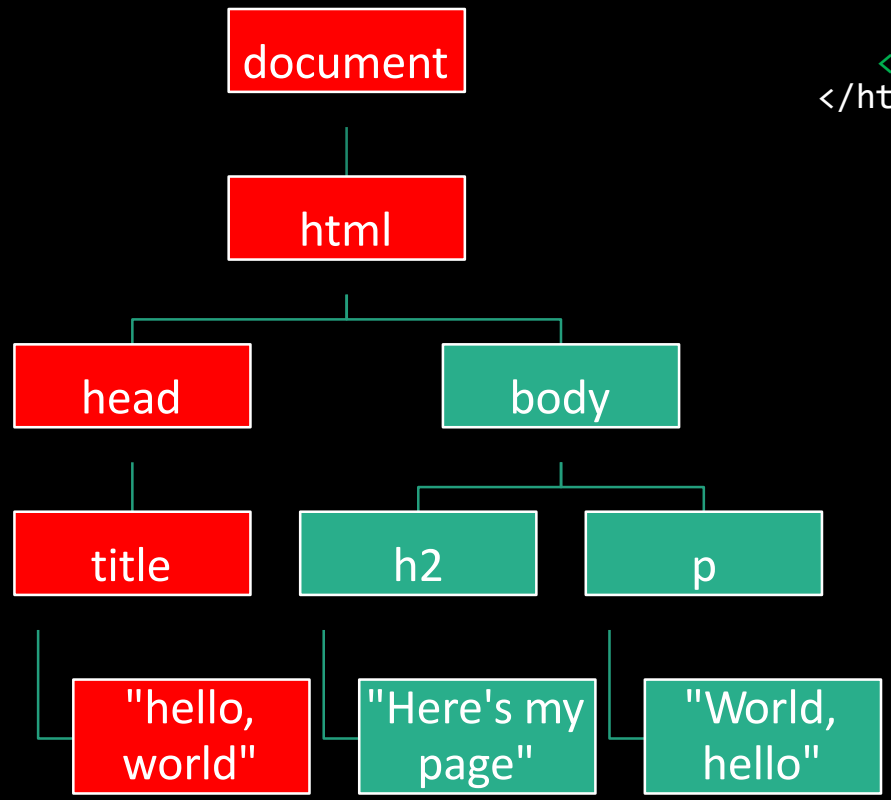

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```



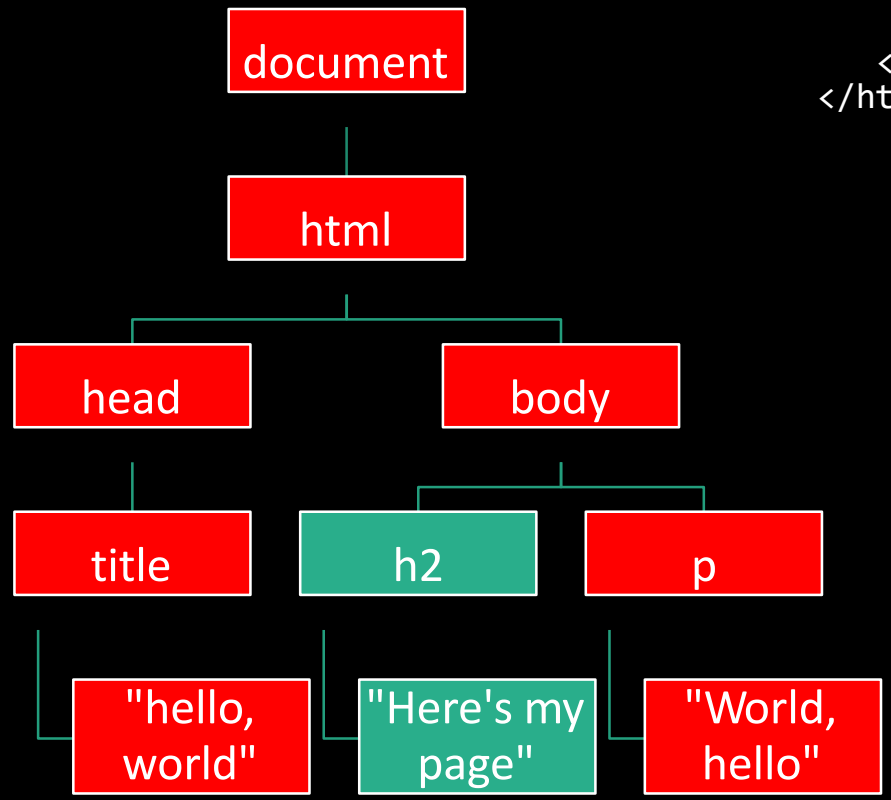

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```



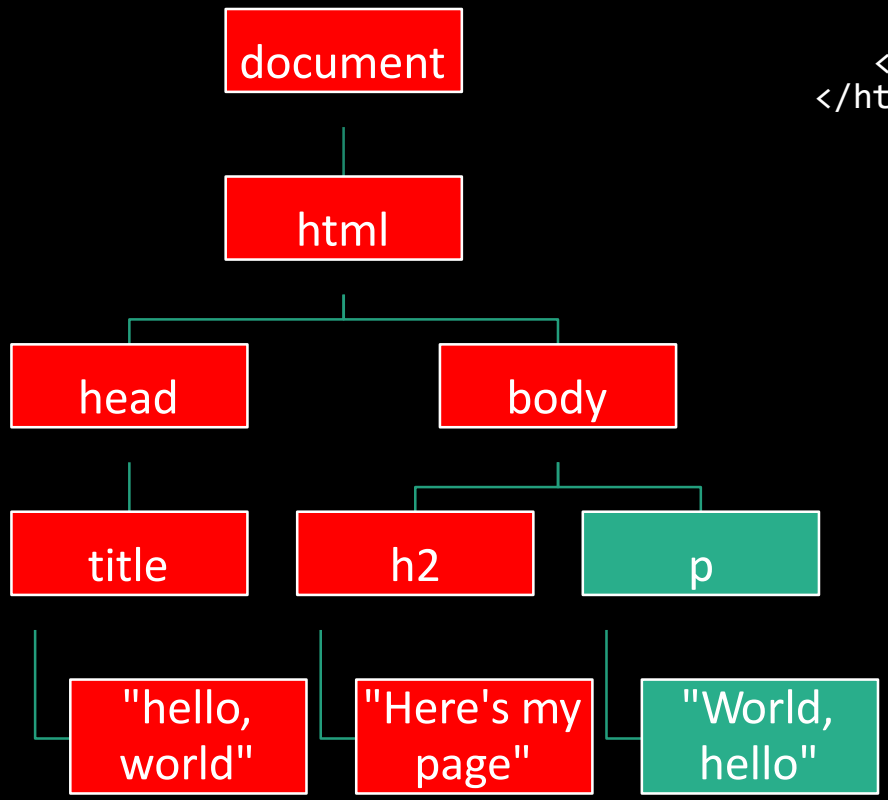

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```




```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```




```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```




```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
   </body>
</html>
```

CSS

CSS

- If HTML is the content of our web pages, then CSS (*Cascading Style Sheets*) are how we will style our sites to make them more aesthetically pleasing.

CSS Properties

- color
- text-align
- width
- height
- margin
- padding
- font-family, font-size, font-weight
- border

Abstract Elements and Attributes

- `div`
- `span`

- `id`
- `class`

JavaScript

JavaScript ES6

```
<script>
   alert('Hello, world!');
</script>
```

Functions

```
function hello() {
   alert('Hello, world!');
}
```

Functions

```
function hello() {
    alert('Hello, world!');
}
```

Events

- onclick
- onmouseover
- onkeydown
- onkeyup
- onload
- onblur
- ...

Query Selector

- `document.querySelector('tag')`
- `document.querySelector('#id')`
- `document.querySelector('.class')`

Ways to Define Variables

- `const`
- `let`
- `var`

Arrow Functions

```
() => {
    alert('Hello, world!');
}
```

Arrow Functions

```
x => {
    alert(x);
}
```

Arrow Functions

```
x => x * 2;
```

Higher-Order Functions

(aka passing functions as arguments to other functions)

```
// Callback function, passed as a parameter in the higher order function
function callbackFunction(){
    console.log('I am a callback function');
}

// higher order function
function higherOrderFunction(func){
    console.log('I am higher order function')
    func()
}

higherOrderFunction(callbackFunction);
```

```
const users = [
    {firstName: 'John', lastName: 'Doe', age: 25},
    {firstName: 'Jane', lastName: 'Doe', age: 30},
    {firstName: 'Jack', lastName: 'Doe', age: 35},
    {firstName: 'Jill', lastName: 'Doe', age: 40},
    {firstName: 'Joe', lastName: 'Doe', age: 45},
]

// Find the users with age greater than 30
const output = users.filter(({age}) => age > 30)
console.log(output);

/* [{firstName: 'Jack', lastName: 'Doe', age: 35}, {firstName: 'Jill', lastName:
'Doe', age: 40}, {firstName: 'Joe', lastName: 'Doe', age: 45}]
*/
```

Local Storage

Using Local Storage

- `localStorage.getItem(key);`
- `localStorage.setItem(key, value);`

Ajax

Ajax Example

```
const request = new XMLHttpRequest();
request.open(method, url);

request.onload = function() {
    // do something
};

const data = new FormData();
data.append(key, value);
request.send(data);
```

Socket.IO

Socket.IO

- on
- emit

JSON

- JavaScript Object Notation.
- Lightweight data-interchange format.
- Easy to read and write than XML.
- Language independent.
- Supports array, object, string, number and values.

```
{
    "origin": "Tokyo",
    "destination": "Shanghai",
    "duration": 185
}
```

```
{
    "origin": "Tokyo",
    "destination": "Shanghai",
    "duration": 185,
    "passengers": ["Alice", "Bob"]
}
```

```
{
    "origin": {
        "city": "Tokyo",
        "code": "HND"
    },
    "destination": {
        "city": "Shanghai",
        "code": "PVG"
    },
    "duration": 185,
    "passengers": ["Alice", "Bob"]
}
```

APIs

/flights

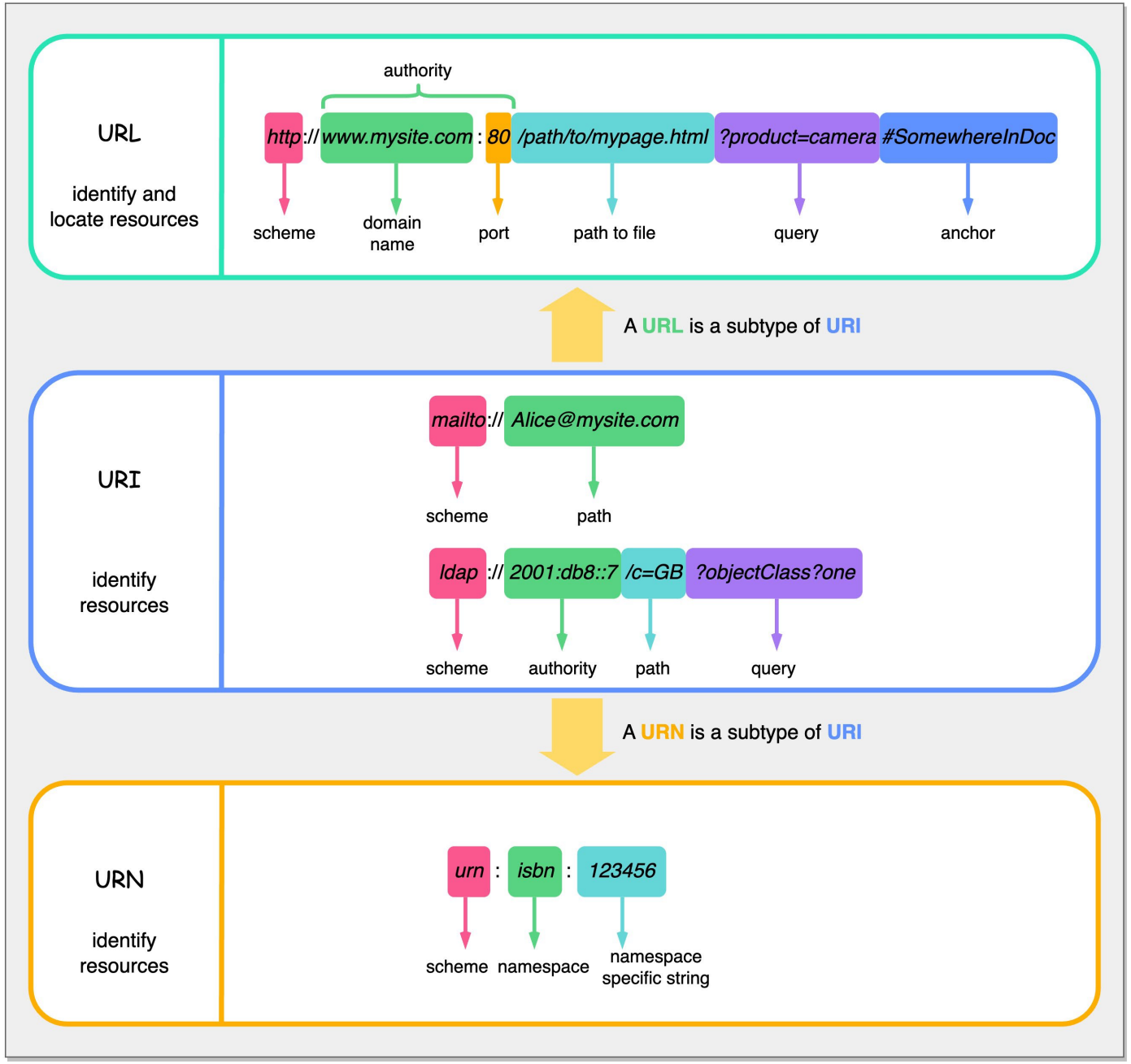
/flights/44

/flights/44/passengers

/flights/44/passengers/6

URL vs URI vs URN

blog.bytebytego.com



HTTP Methods

- GET
- POST
- PUT
- PATCH
- DELETE

HTTP Methods

- **GET**: retrieve a resource (SELECT)
- **POST**: create a new resource (INSERT)
- **PUT**: replace a resource
- **PATCH**: update a resource (UPDATE)
- **DELETE**: delete a resource (DELETE)

Design Effective & Safe APIs

blog.bytebytego.com

Use resource names (nouns)	GET /querycarts/123	GET /carts/123
Use plurals	GET /cart/123	GET /carts/123
Idempotency	POST /carts	POST /carts {requestId: 4321}
Use versioning	GET /carts/v1/123	GET /v1/carts/123
Query after soft deletion	GET /carts	GET /carts? includeDeleted=true
Pagination	GET /carts	GET /carts? pageSize=xx&pageToken=xx
Sorting	GET /items	GET /items? sort_by=time
Filtering	GET /items	GET /items? filter=color:red
Secure Access	X-API-KEY=xxx	X-API-KEY = xxx X-EXPIRY = xxx X-REQUEST-SIGNATURE = xxx hmac(URL + QueryString + Expiry + Body)
Resource cross reference	GET /carts/123? item=321	GET /carts/123/items/321
Add an item to a cart	POST /carts/123? addItem=321	POST /carts/123/items:add { itemId: "items/321" }
Rate limit	No rate limit - DDos	Design rate limiting rules based on IP, user, action group etc

requests module

Using the `requests` module

- `requests.get(url)`
- `requests.post(url)`
- `requests.put(url)`
- `requests.patch(url)`
- `requests.delete(url)`

HTTP Status Codes

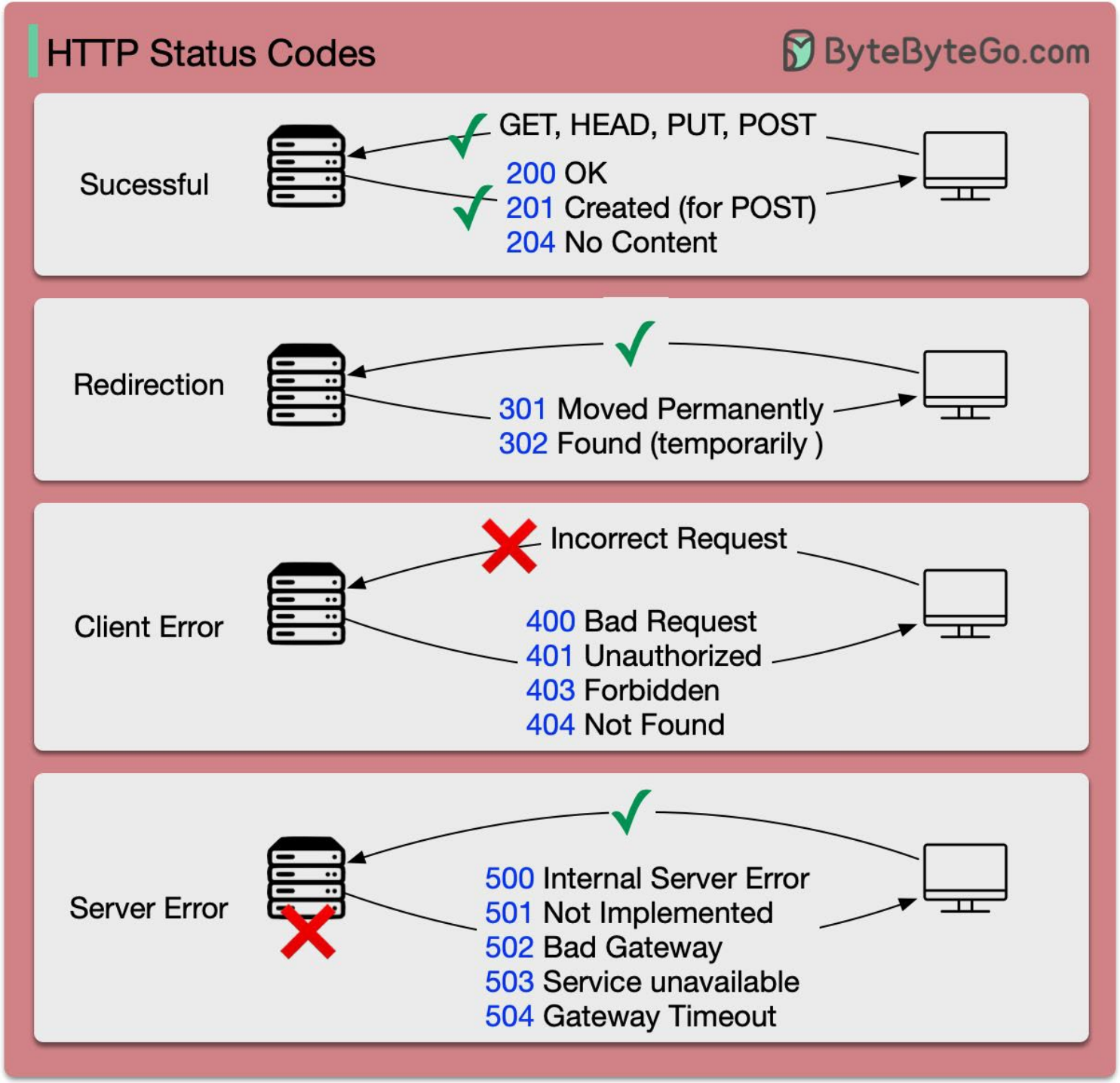
- 200 OK
- 201 Created
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 422 Unprocessable Entity
- ...

HTTP Status Codes

- 200 OK
- 201 Created
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 422 Unprocessable Entity
- ...

HTTP Status Codes

- 200 OK
- 201 Created
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 422 Unprocessable Entity
- ...


HTTP Status Codes
ByteByteGo.com
Sucessful
GET, HEAD, PUT, POST
200 OK
201 Created (for POST)
204 No Content
Redirection
301 Moved Permanently
302 Found (temporarily)
Client Error
Incorrect Request
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
Server Error
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service unavailable
504 Gateway Timeout

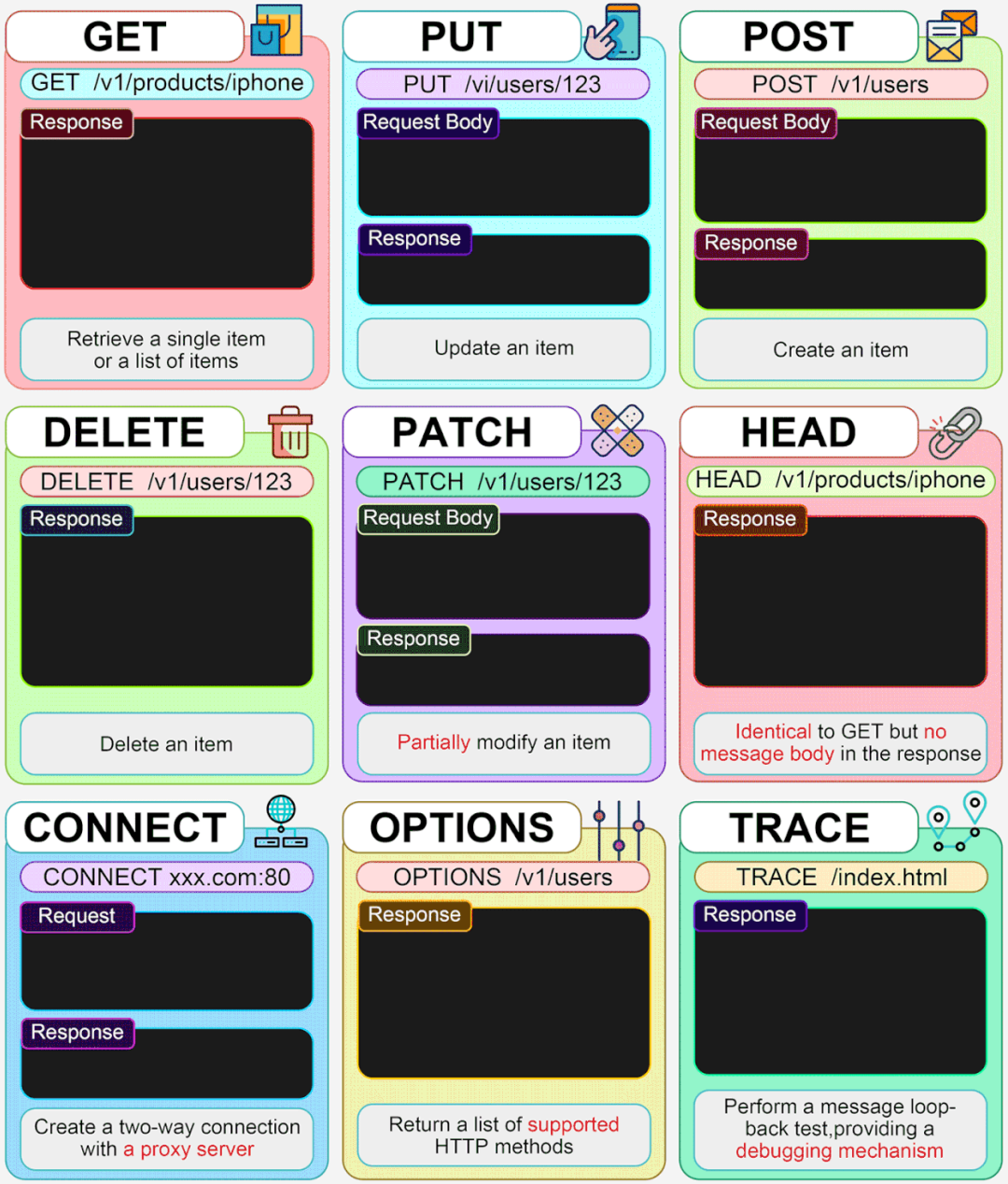
GET vs POST

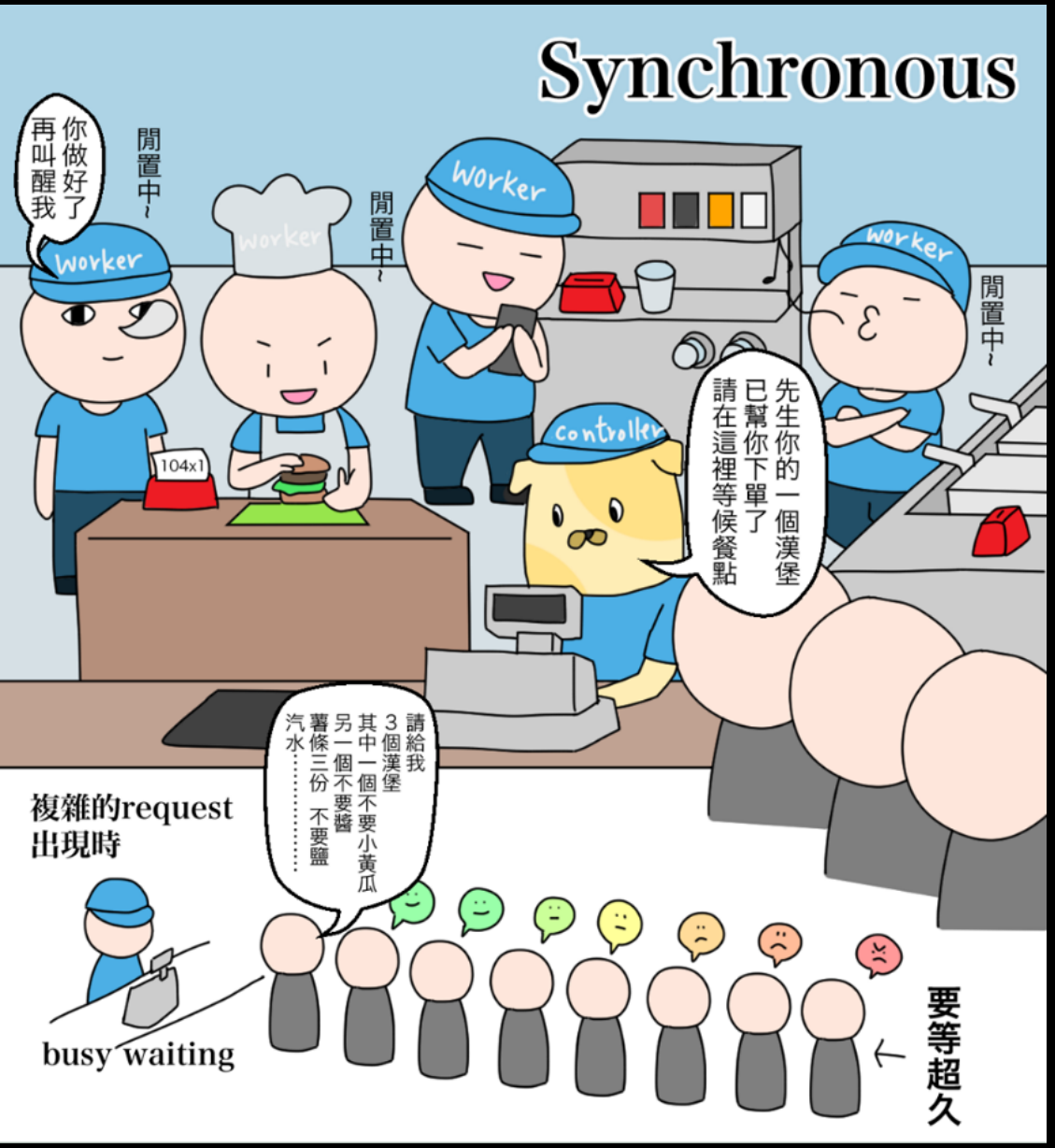
	Get	Post
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed

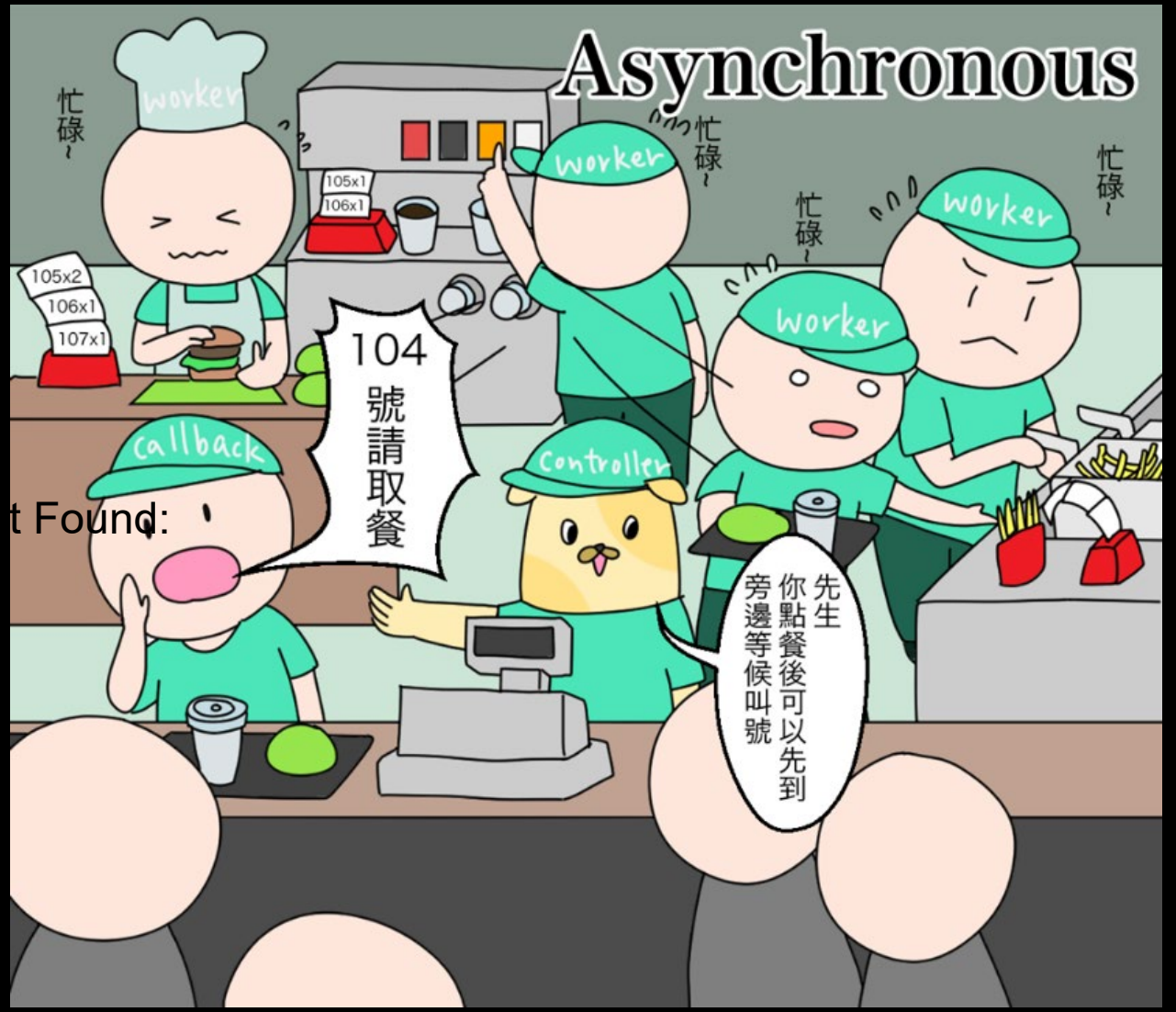
	Get	Post
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Top 9 HTTP Request Methods

ByteByteGo




Synchronous
你做好了再叫醒我
worker
閒置中~
worker
閒置中~
Worker
worker
閒置中~
104x1
controller
先生你的一個漢堡已幫你下單了請在這裡等候餐點
請給我3個漢堡其中一個不要小黃瓜另一個不要醬薯條三份 不要鹽汽水……
複雜的request出現時
busy waiting
要等超久


Asynchronous
忙碌~
worker
105x1
106x1
忙碌~
worker
忙碌~
worker
忙碌~
105x2
106x1
107x1
104號請取餐
callback
controller
worker
先生你點餐後可以先到旁邊等候叫號

t Found:

Why HTTP/2

Performance matters

HTTP/2 reduces the impact of latency on web applications

TLS is becoming the default

HTTP/2 amortizes TLS costs for the entire application

Enabling new web development

HTTP/2 Goals

User perceivable improvement in web site performance

Work with today's internet

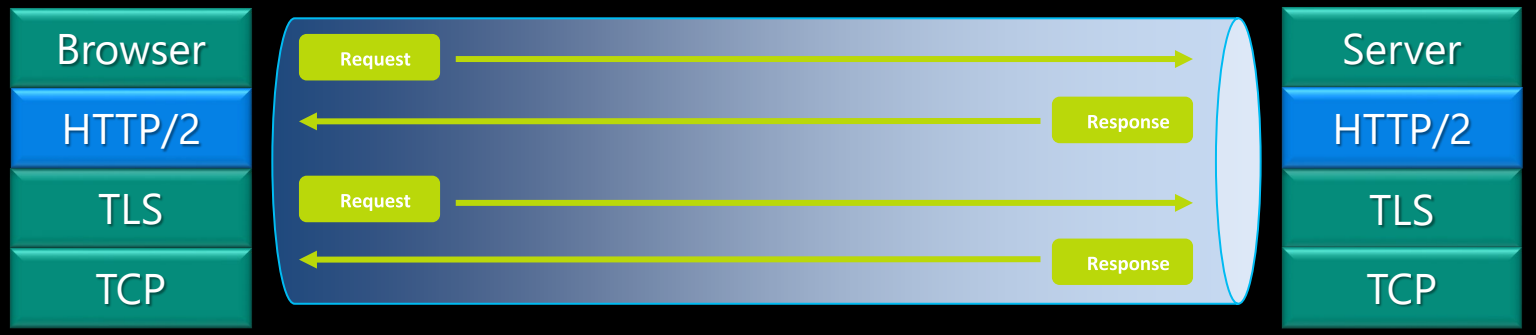
Remain compatible with existing content

HTTP/2 Overview

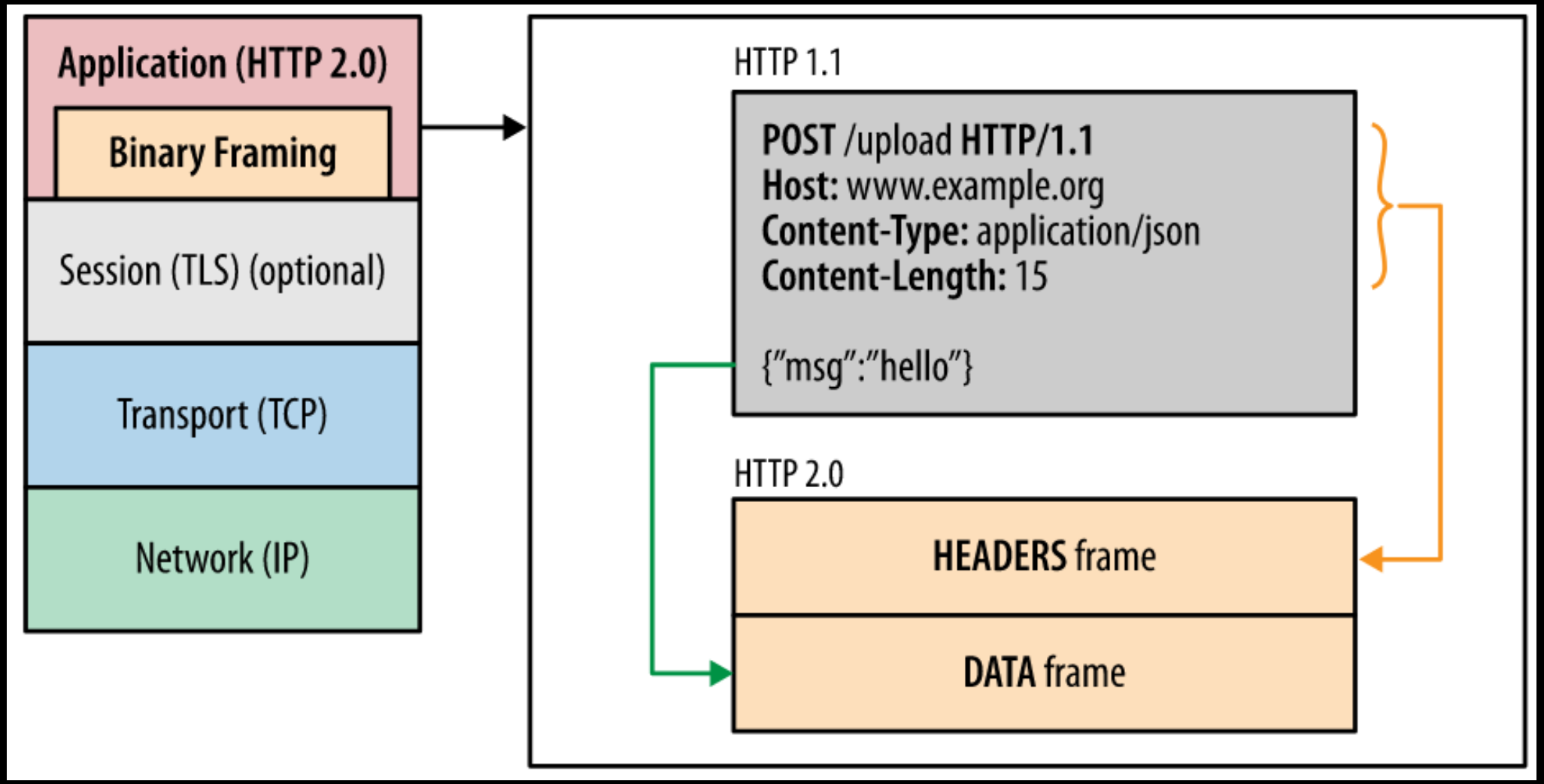
Multiplexing

Header Compression

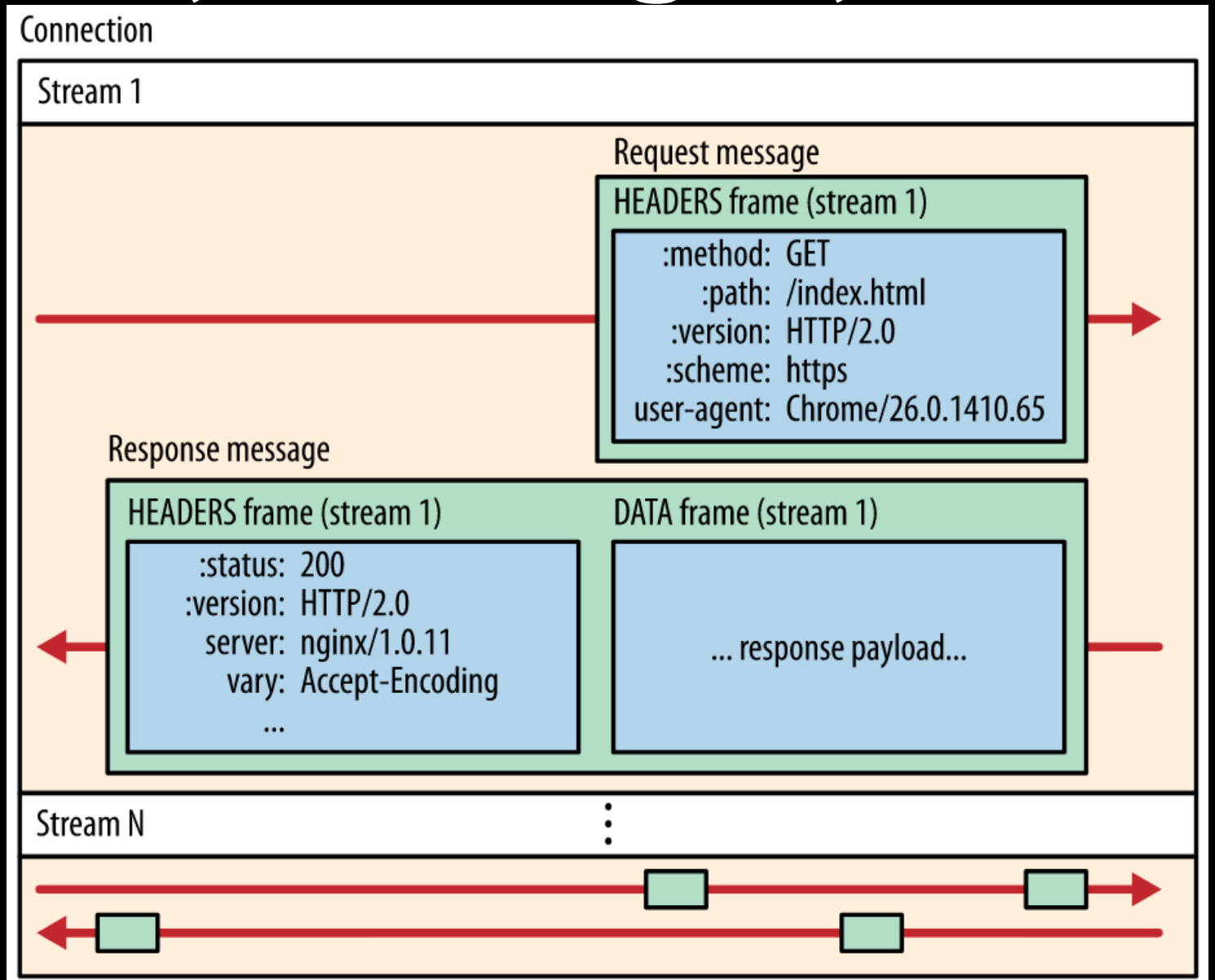
Server Push



Binary framing layer

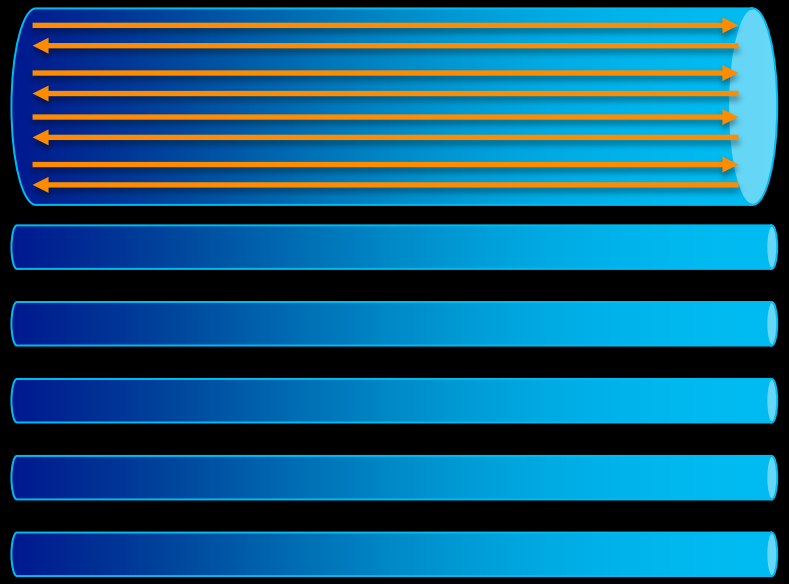


Streams, messages, and frames

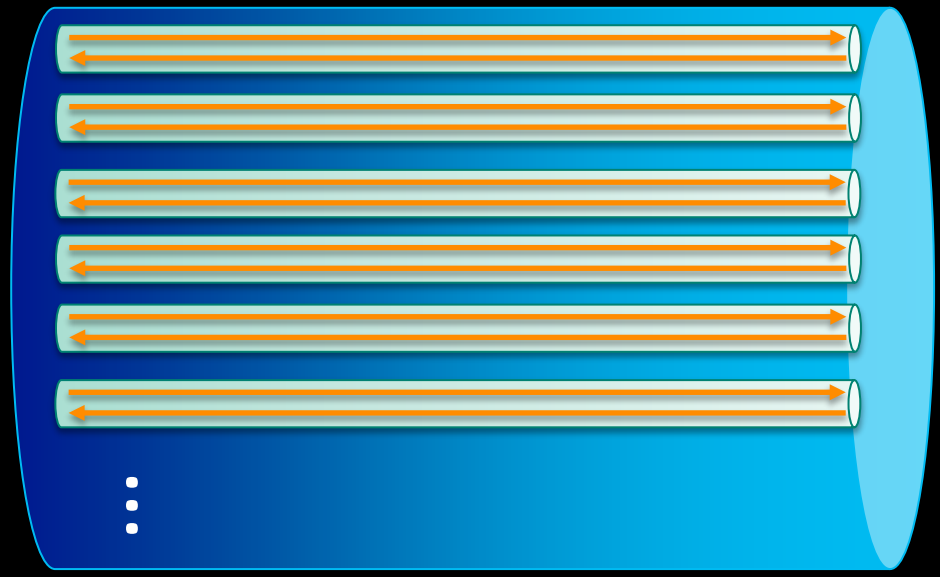


HTTP/2 Connections and Streams

HTTP/1.1 – Request = Connection

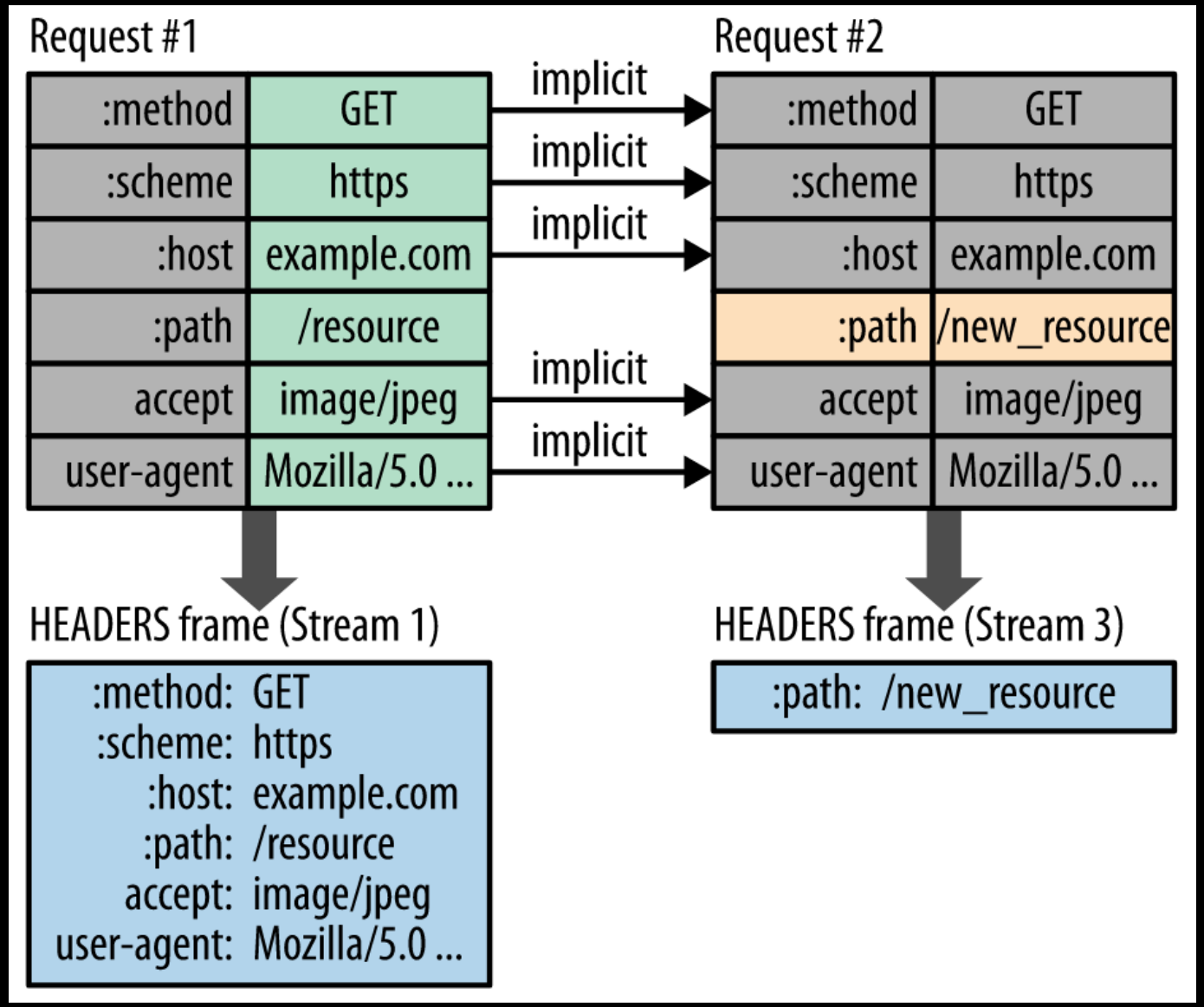


HTTP/2 – Request = Stream

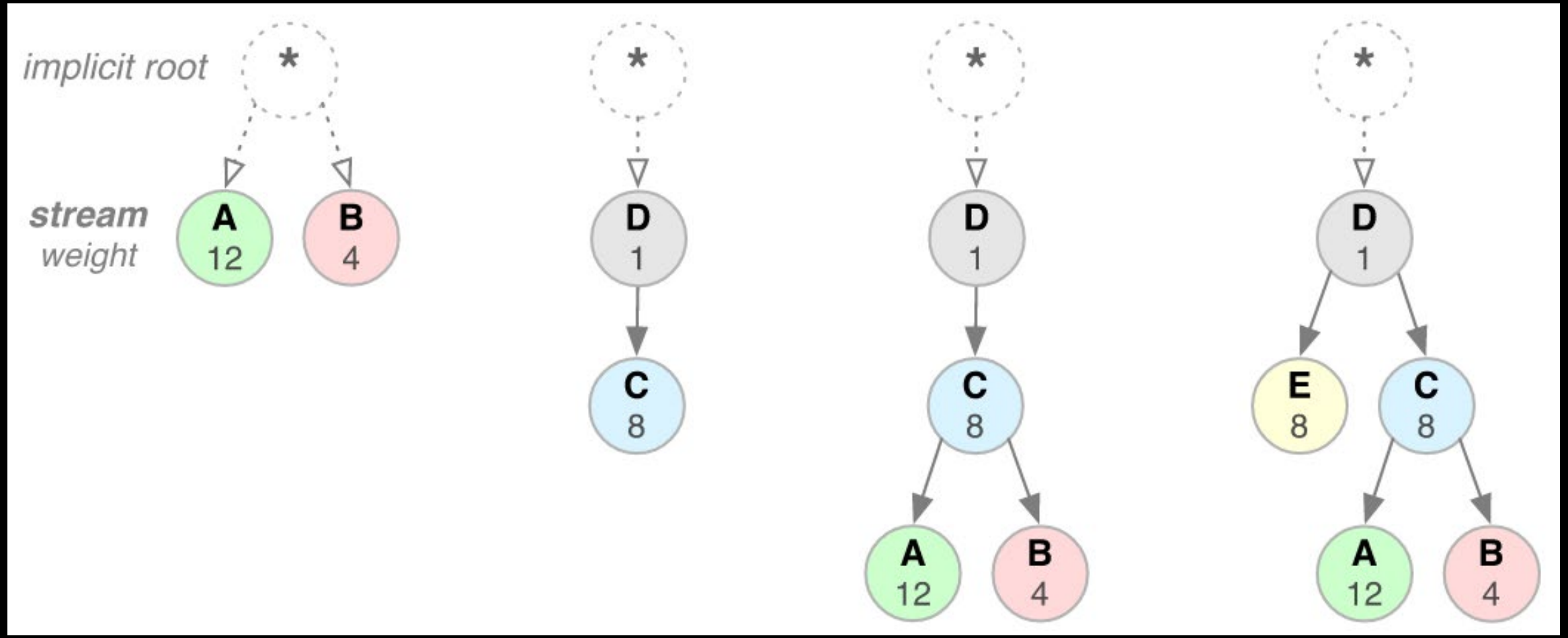


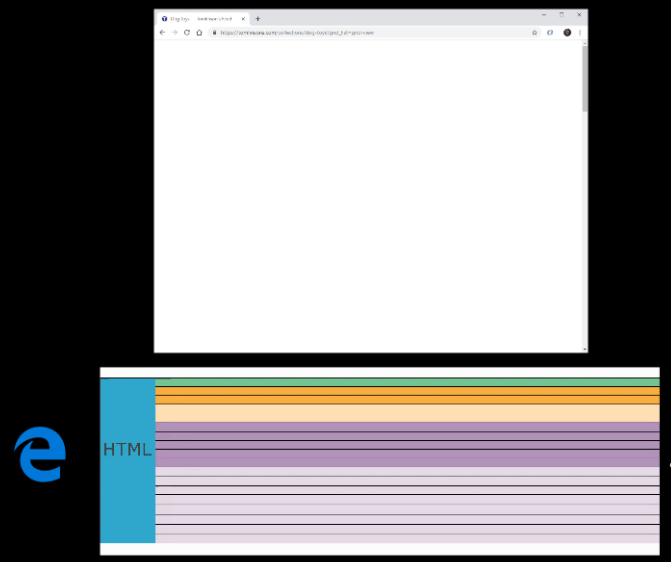
Each request required dedicated TCP connection	TCP connection can have multiple streams (requests)
Responses come in order per connection	Responses can come out of order, server can optimize
Each connection requires setup + slow start	No connection setup for new streams, no slow start
Application sees “connections”	Streams are represented as “connections” to apps

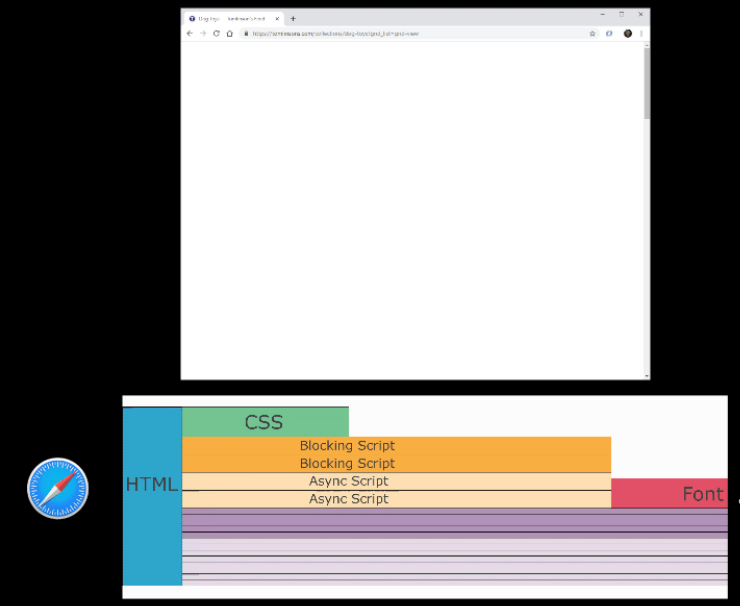
Header Compression

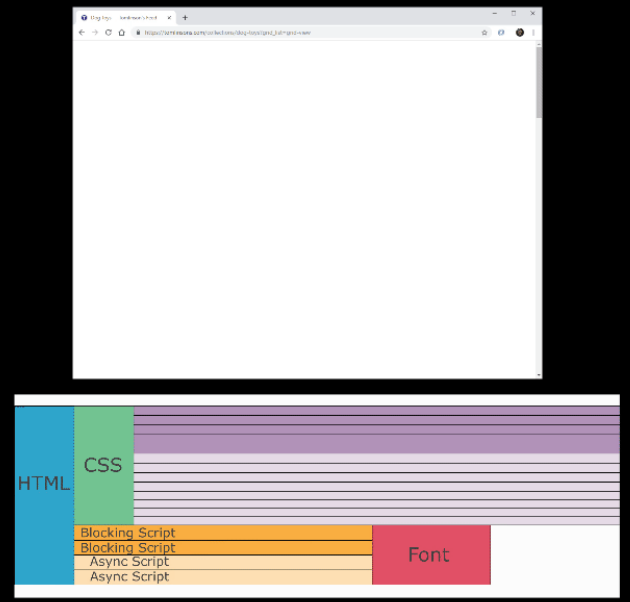


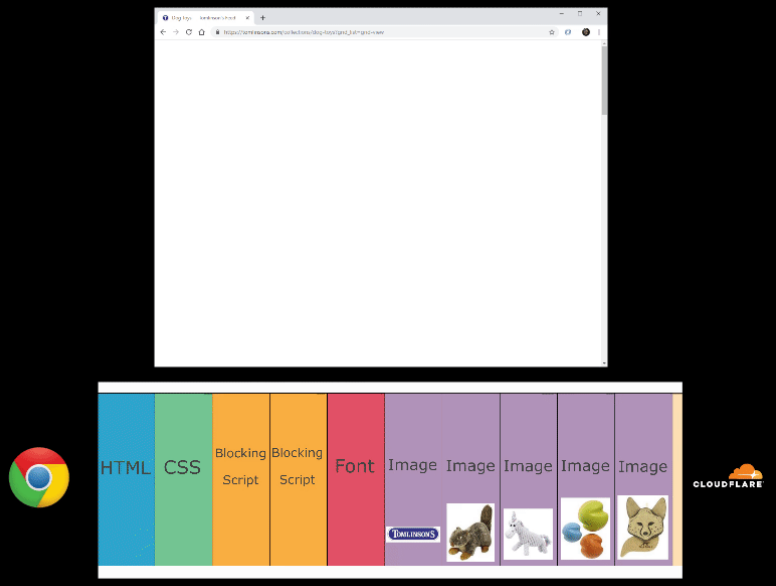
Stream prioritization



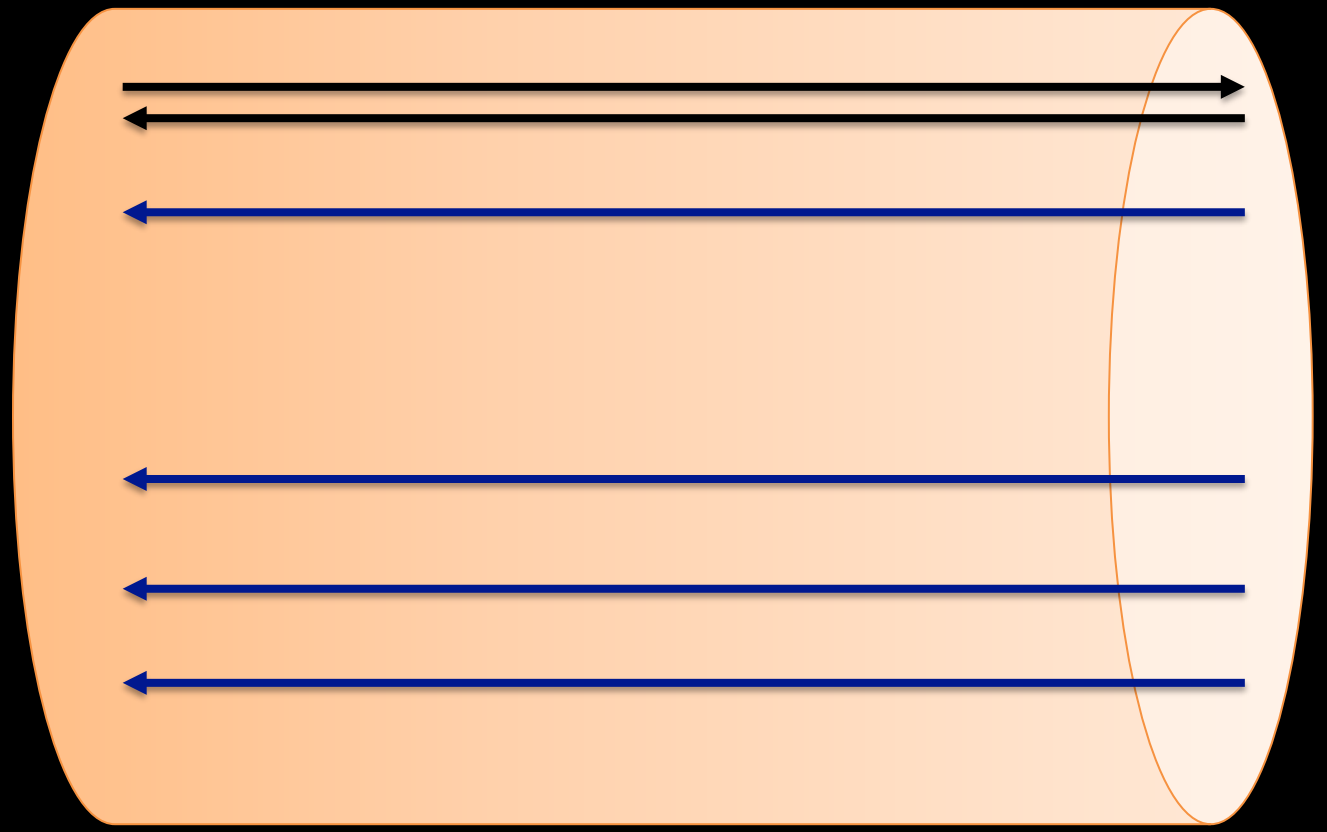

HTML
CLOUDFLARE


CSS
Blocking Script
Blocking Script
Async Script
Async Script
HTML
Font
CLOUDFLARE


HTML
CSS
Blocking Script
Blocking Script
Async Script
Async Script
Font
CLOUDFLARE


HTML
CSS
Blocking Script
Blocking Script
Font
Image
Image
Image
Image
Image
CLOUDFLARE

Server Push

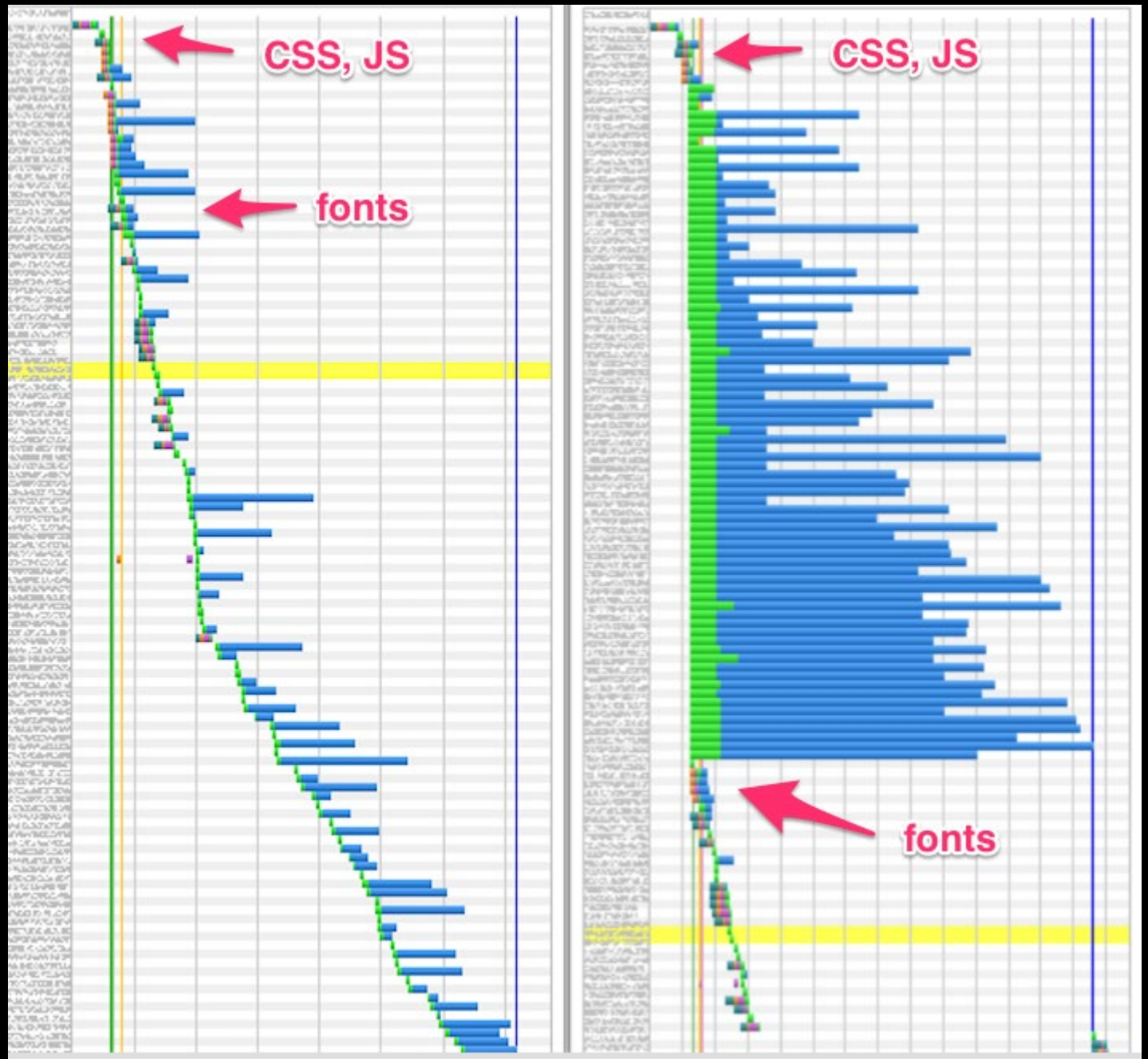


- App calls new IIS/ASP API with desired request headers
- Creates new request in http.sys pipeline, delivered to app as if client made request

HTTP/1.1

Latency: **98ms**

Load time: **2.20s**


CSS, JS
fonts
CSS, JS
fonts
http/1.1
http/2

HTTP1.x	HTTP2
SSL not required but recommended.	SSL not required but recommended.
Slow encryption.	Even faster encryption.
One client-server request per TCP connection.	Multi-host multiplexing. Occurs on multiple hosts at a single instant.
No header compression.	Header compression using improved algorithms that improve performance as well as security.
No stream prioritization.	Improved stream prioritization mechanisms used.

How did we get to HTTP 3.0?

ByteByteGo