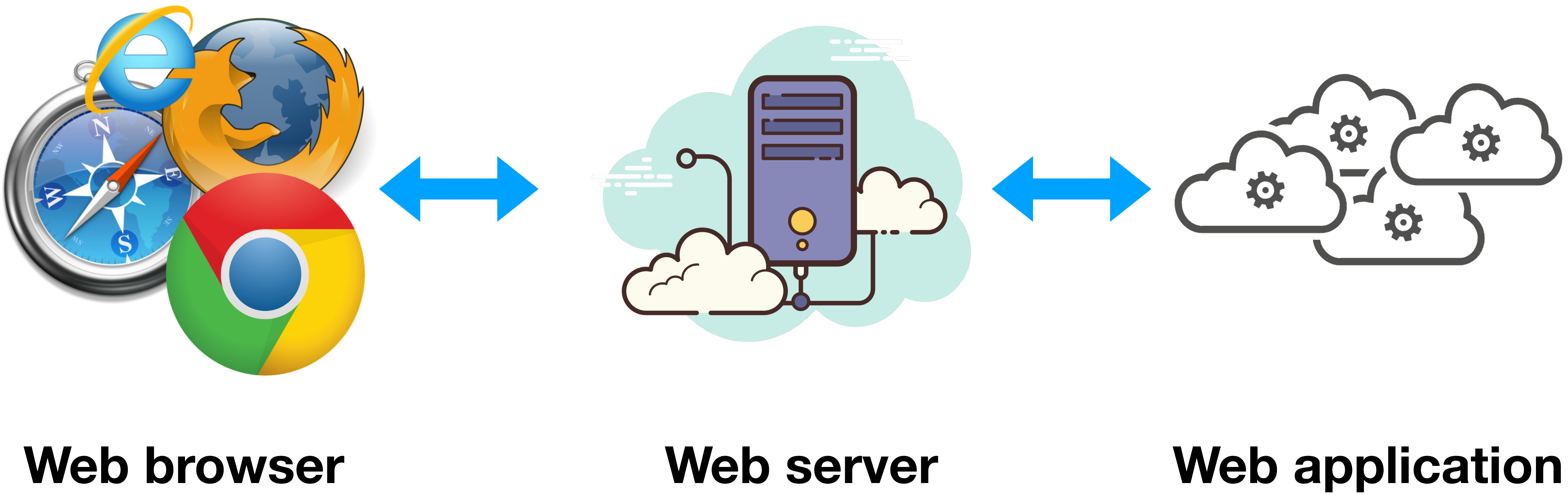
Web Application Development

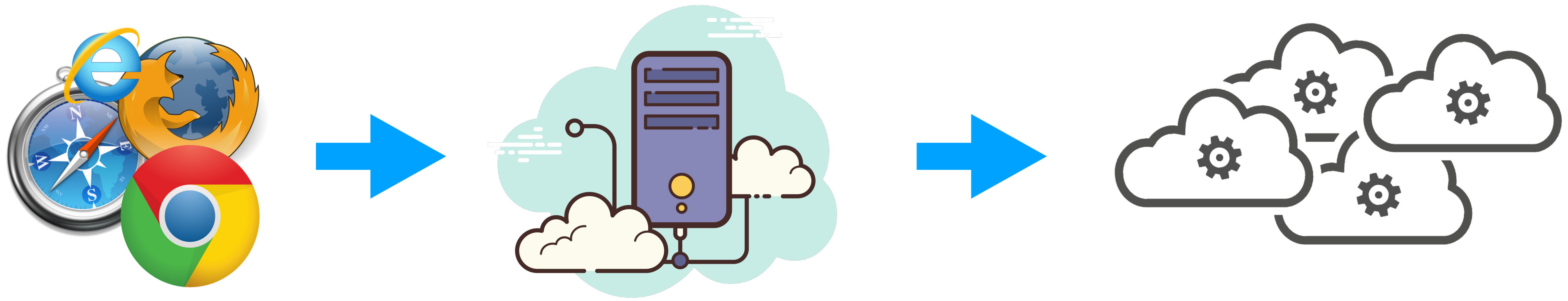
Alexander Menshchikov

Backend

Web Application Architecture



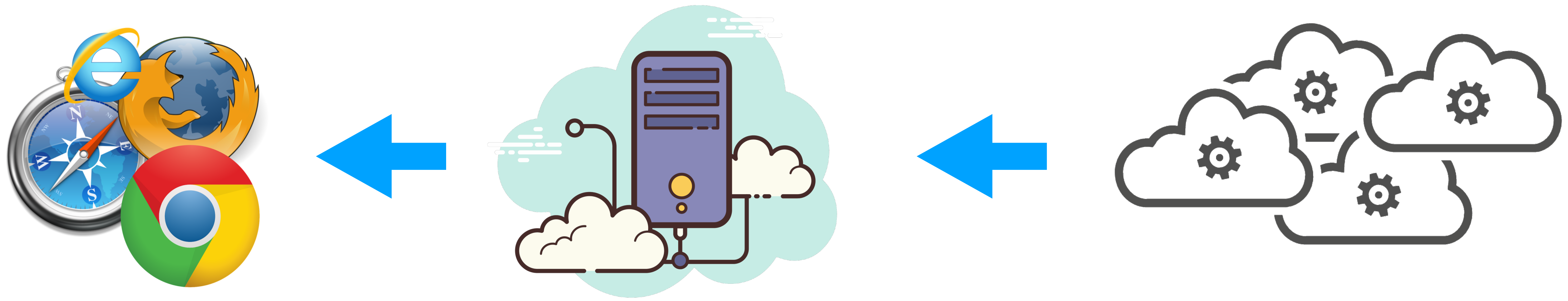
Architecture. Step 1



HTTP request
wad.itmo.xyz/

/ is mapped
to Application

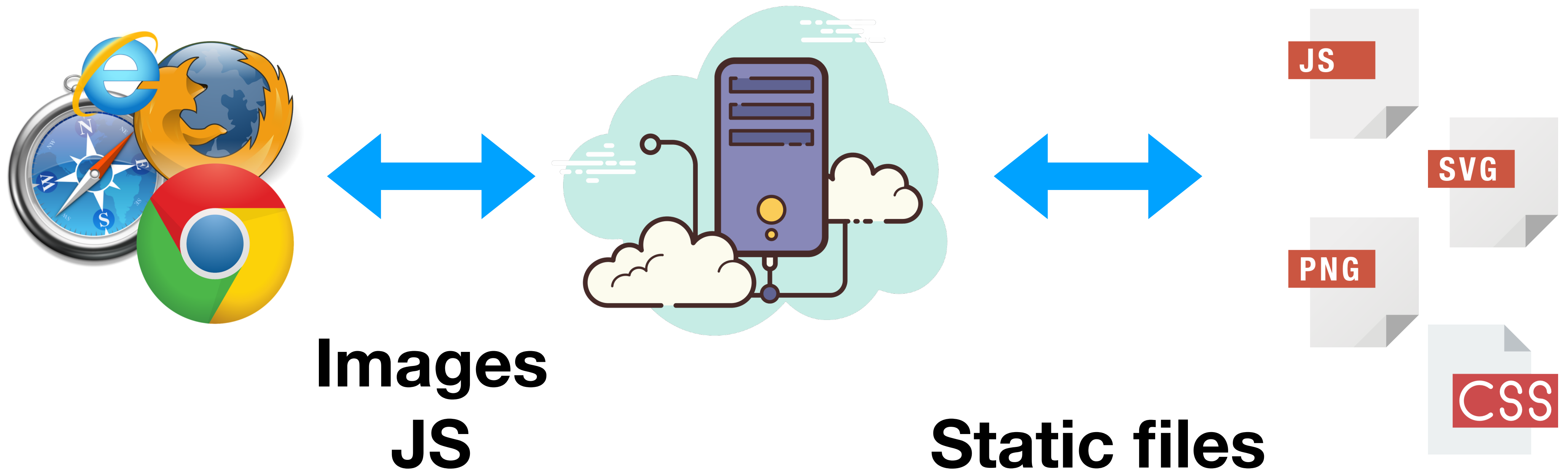
Architecture. Step 2



Send HTML
back to client

Render HTML

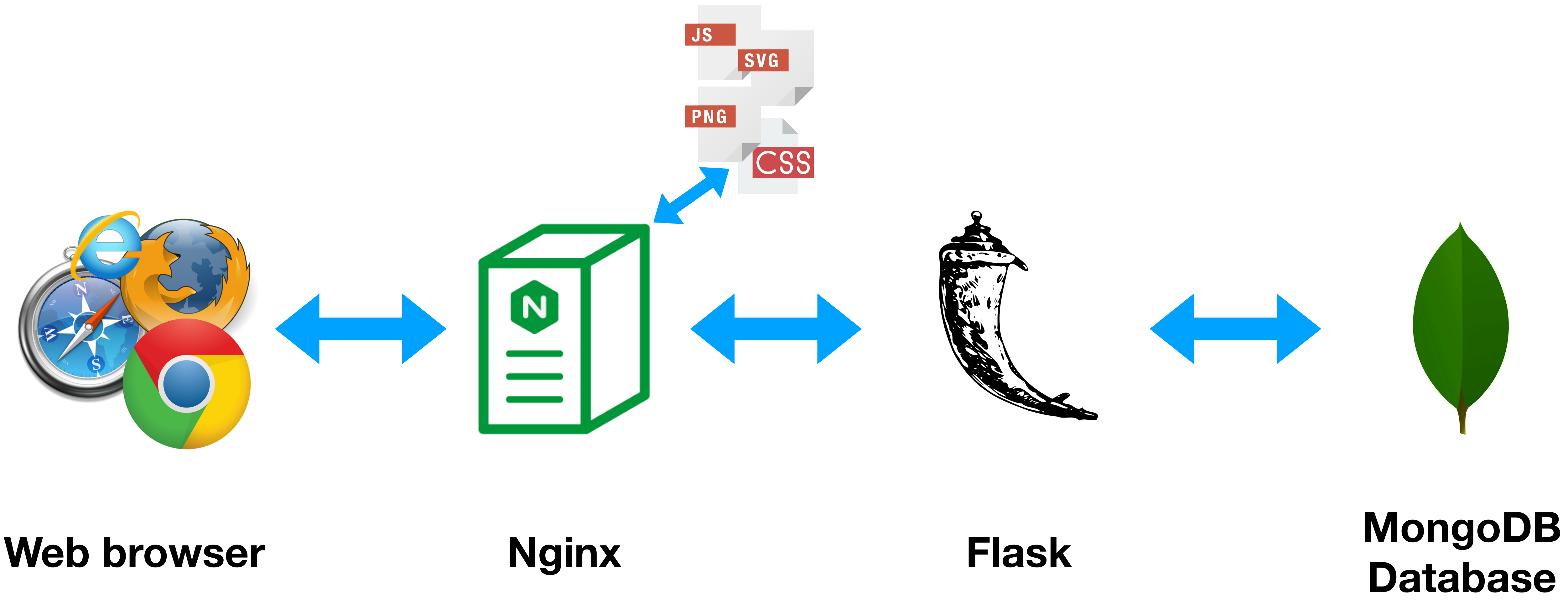
Architecture. Step 3



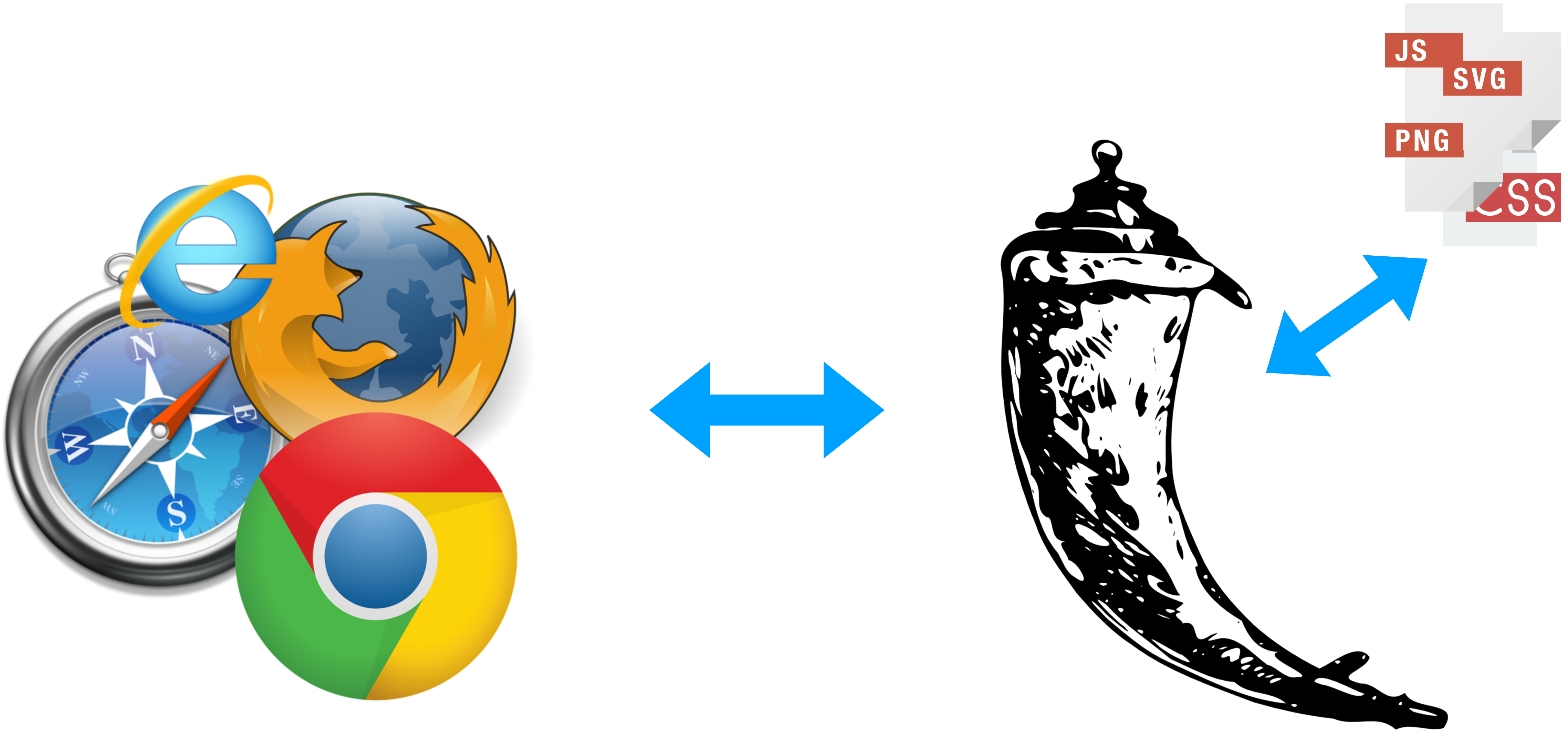
Images
JS
CSS

Static files

Web Application Architecture



Web Application Architecture



Web browser

Flask

HTTP

HTTP Request

curl http://wad.itmo.xyz -vvv

Method

```
*   Trying 185.199.108.153...
* TCP_NODELAY set
* Connected to wad.itmo.xyz (185.199.108.153) port 80 (#0)
> GET / HTTP/1.1
> Host: wad.itmo.xyz
> User-Agent: curl/7.64.1
> Accept: */*
>
```

```
97 db 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ..GET /  HTTP/1.1
0d 0a 48 6f 73 74 3a 20  77 61 64 2e 69 74 6d 6f   ..Host:  wad.itmo
2e 78 79 7a 0d 0a 55 73  65 72 2d 41 67 65 6e 74   .xyz..Us er-Agent
3a 20 63 75 72 6c 2f 37  2e 36 34 2e 31 0d 0a 41   : curl/7 .64.1..A
63 63 65 70 74 3a 20 2a  2f 2a 0d 0a 0d 0a         ccept: * /*....
```

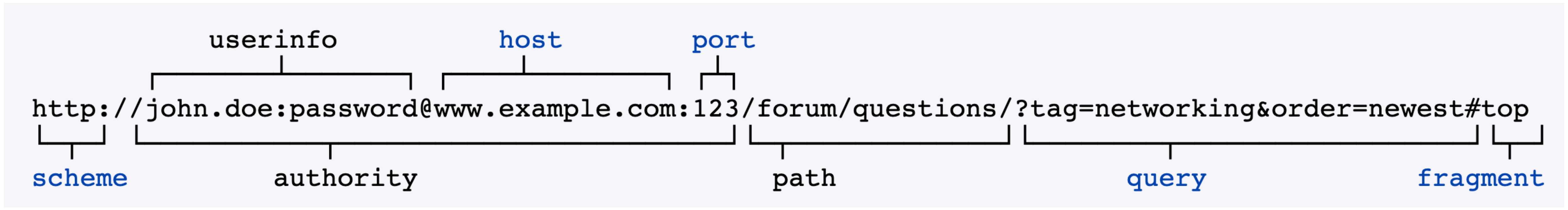
HTTP Response

`curl http://wad.itmo.xyz -vvv`

Status

```
< HTTP/1.1 301 Moved Permanently
< Server: GitHub.com
< Content-Type: text/html
< Location: https://wad.itmo.xyz/
< Content-Length: 162
< Date: Thu, 02 Apr 2020 11:30:02 GMT
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host wad.itmo.xyz left intact
* Closing connection 0
```

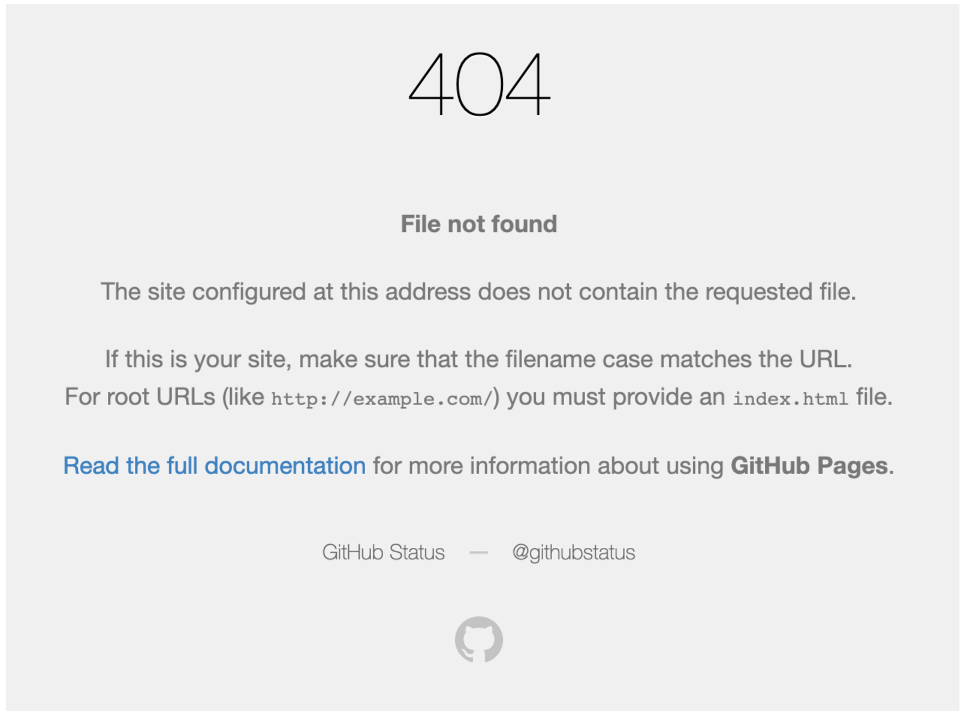
URI


userinfo
host
port
http://john.doe:password@www.example.com:123/forum/questions/?tag=networking&order=newest#top
scheme
authority
path
query
fragment

https://wad.itmo.xyz/index.html

https://wad.itmo.xyz/

https://wad.itmo.xyz/qwerty →



404

File not found

The site configured at this address does not contain the requested file.

If this is your site, make sure that the filename case matches the URL.
For root URLs (like `http://example.com/`) you must provide an `index.html` file.

Read the full documentation for more information about using **GitHub Pages**.

GitHub Status — @githubstatus

HTTP Status codes

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

- 1xx: Informational
- 2xx: Success
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 502 Bad Gateway
- 504 Gateway Timeout.

HTTP Headers

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Request

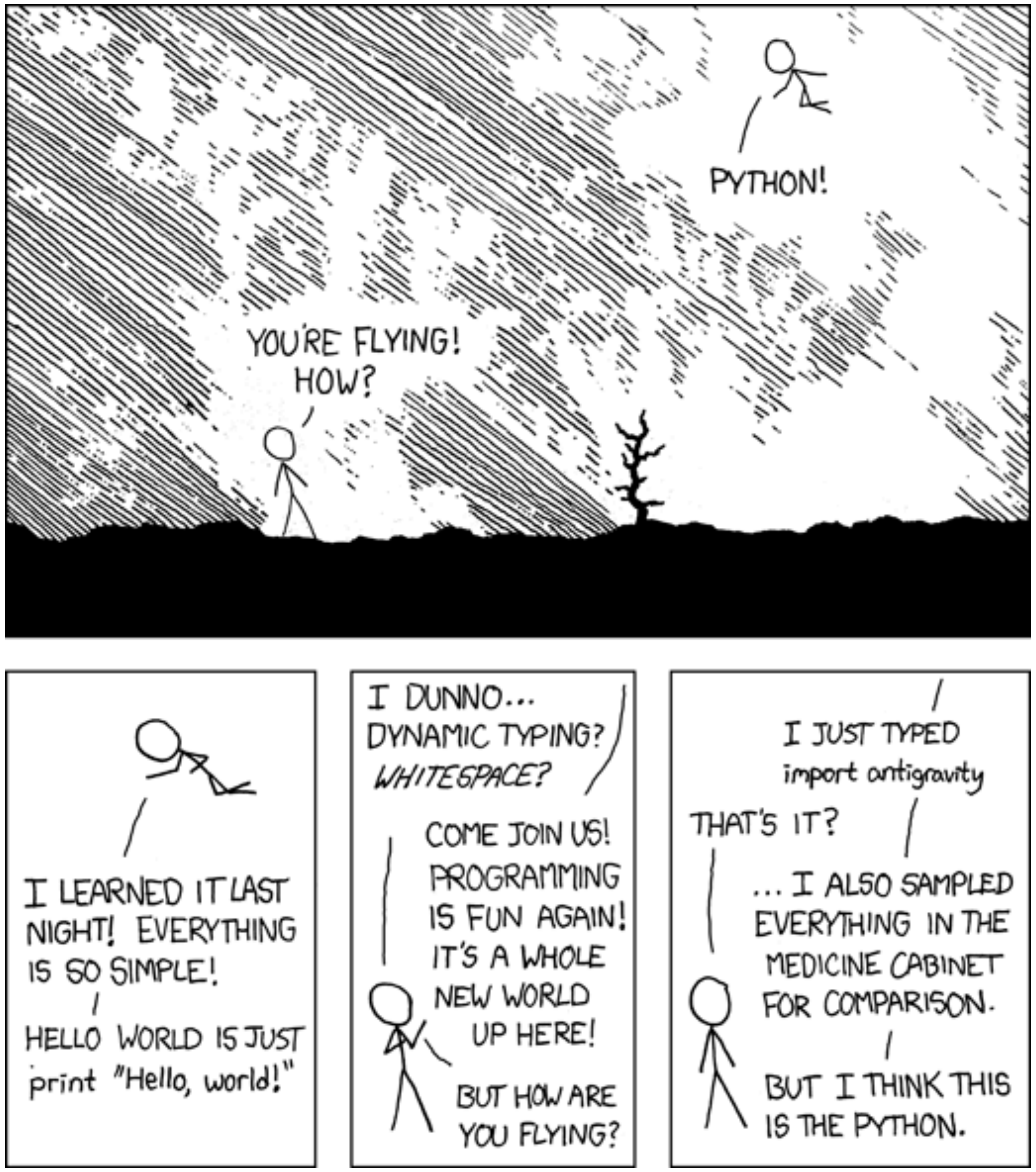
- Authorization
- Content-Type
- Cookie
- Host
- Referer
- User-Agent

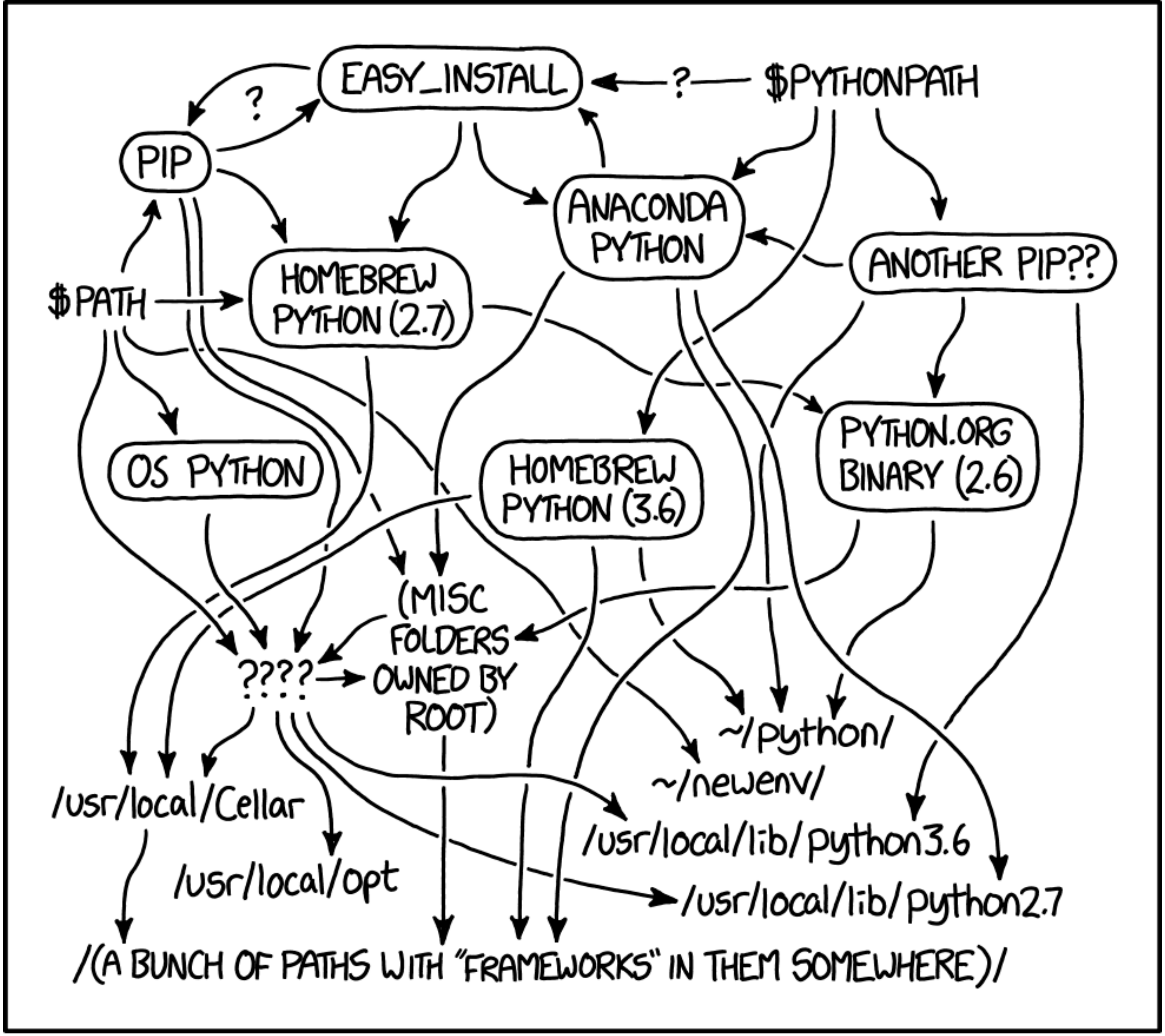
Response

- Location
- Server
- Set-Cookie

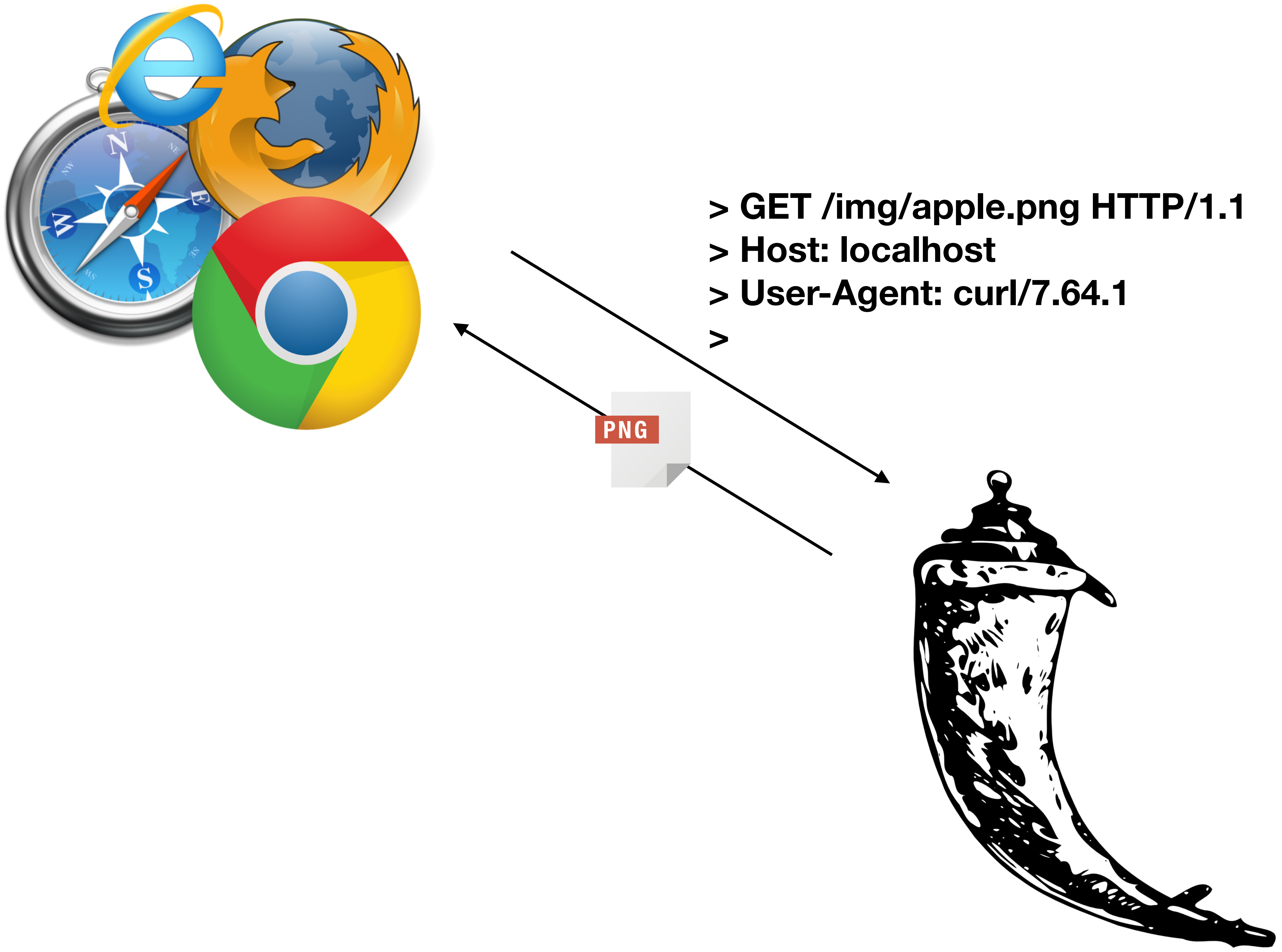
Demo

Python Flask

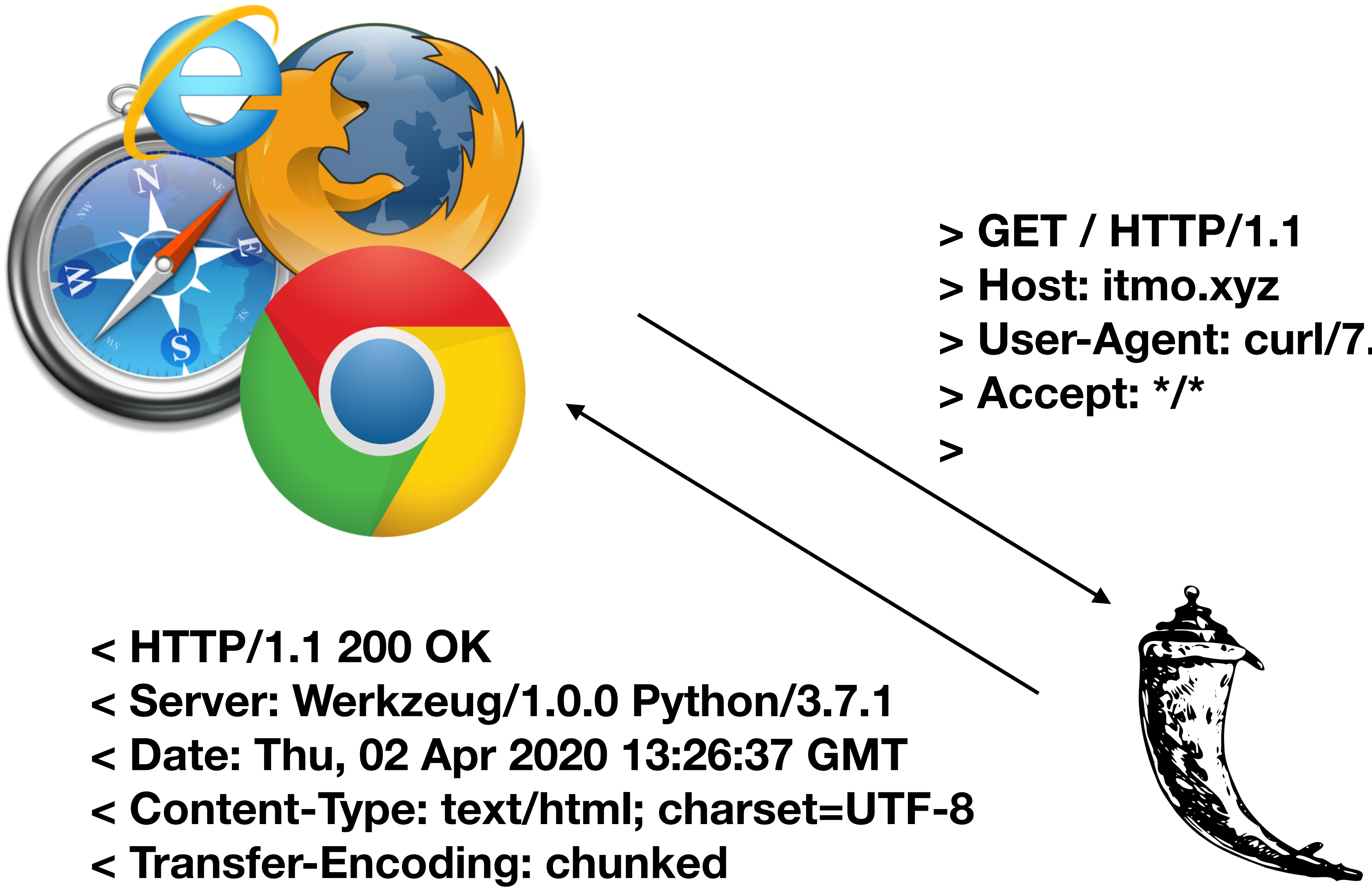

PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.



```
> GET /img/apple.png HTTP/1.1
> Host: localhost
> User-Agent: curl/7.64.1
>
```



```
> GET / HTTP/1.1
> Host: itmo.xyz
> User-Agent: curl/7.64.1
> Accept: */*
>
```

```
< HTTP/1.1 200 OK
< Server: Werkzeug/1.0.0 Python/3.7.1
< Date: Thu, 02 Apr 2020 13:26:37 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
<
Visit <a href='https://github.com/itmo-wad'>github.com/itmo-wad</a>
```

Literature

- Documentation: https://flask.palletsprojects.com/en/1.1.x/
- Step-by-step tutorial: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
- Python simple tutorial: https://pythontutor.ru/

Demo

HTTP data transfer

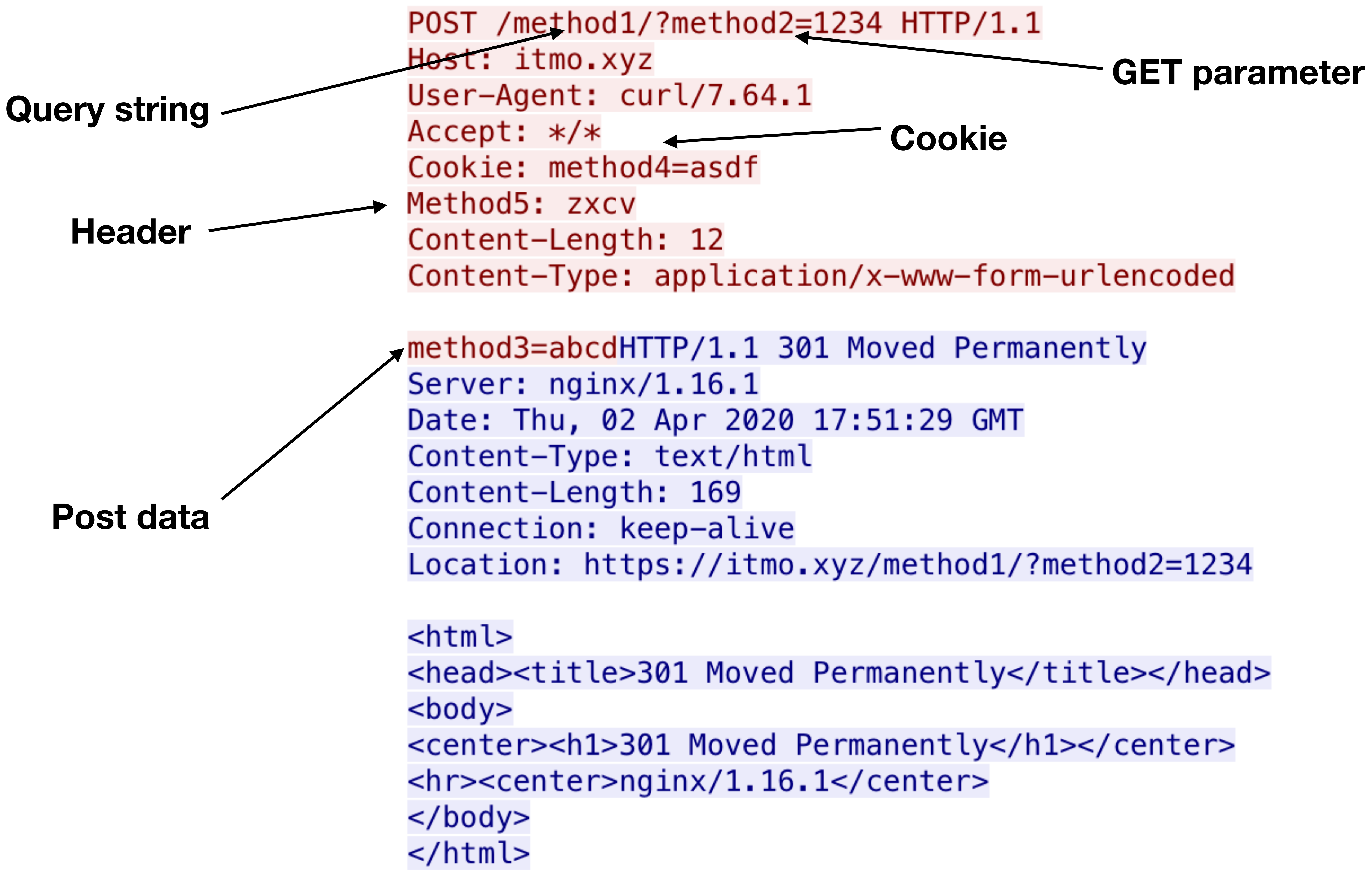
Input data

```
POST /method1/?method2=1234 HTTP/1.1
Host: itmo.xyz
User-Agent: curl/7.64.1
Accept: */*
Cookie: method4=asdf
Method5: zxcv
Content-Length: 12
Content-Type: application/x-www-form-urlencoded

method3=abcdHTTP/1.1 301 Moved Permanently
Server: nginx/1.16.1
Date: Thu, 02 Apr 2020 17:51:29 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://itmo.xyz/method1/?method2=1234

<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.16.1</center>
</body>
</html>
```

Input data

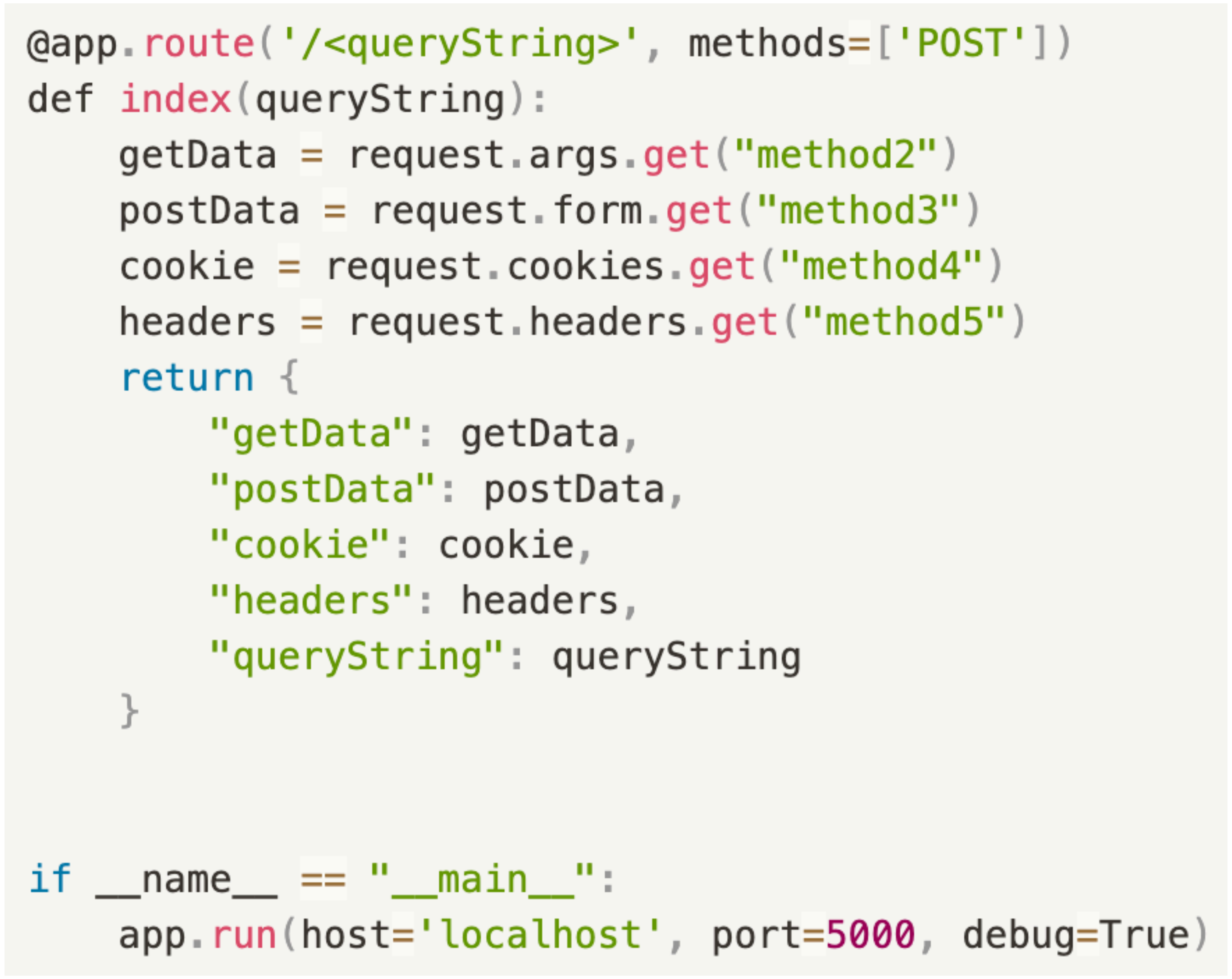


```
POST /method1/?method2=1234 HTTP/1.1
Host: itmo.xyz
User-Agent: curl/7.64.1
Accept: */*
Cookie: method4=asdf
Method5: zxcv
Content-Length: 12
Content-Type: application/x-www-form-urlencoded

method3=abcdHTTP/1.1 301 Moved Permanently
Server: nginx/1.16.1
Date: Thu, 02 Apr 2020 17:51:29 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://itmo.xyz/method1/?method2=1234

<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.16.1</center>
</body>
</html>
```

Get data in Flask



```python
@app.route('/<queryString>', methods=['POST'])
def index(queryString):
    getData = request.args.get("method2")
    postData = request.form.get("method3")
    cookie = request.cookies.get("method4")
    headers = request.headers.get("method5")
    return {
        "getData": getData,
        "postData": postData,
        "cookie": cookie,
        "headers": headers,
        "queryString": queryString
    }


if __name__ == "__main__":
    app.run(host='localhost', port=5000, debug=True)
```

```
curl -X POST -H "Cookie: method4=444" -H "method5: 555" --data
"method3=333" http://localhost:5000/111?method2=222
```

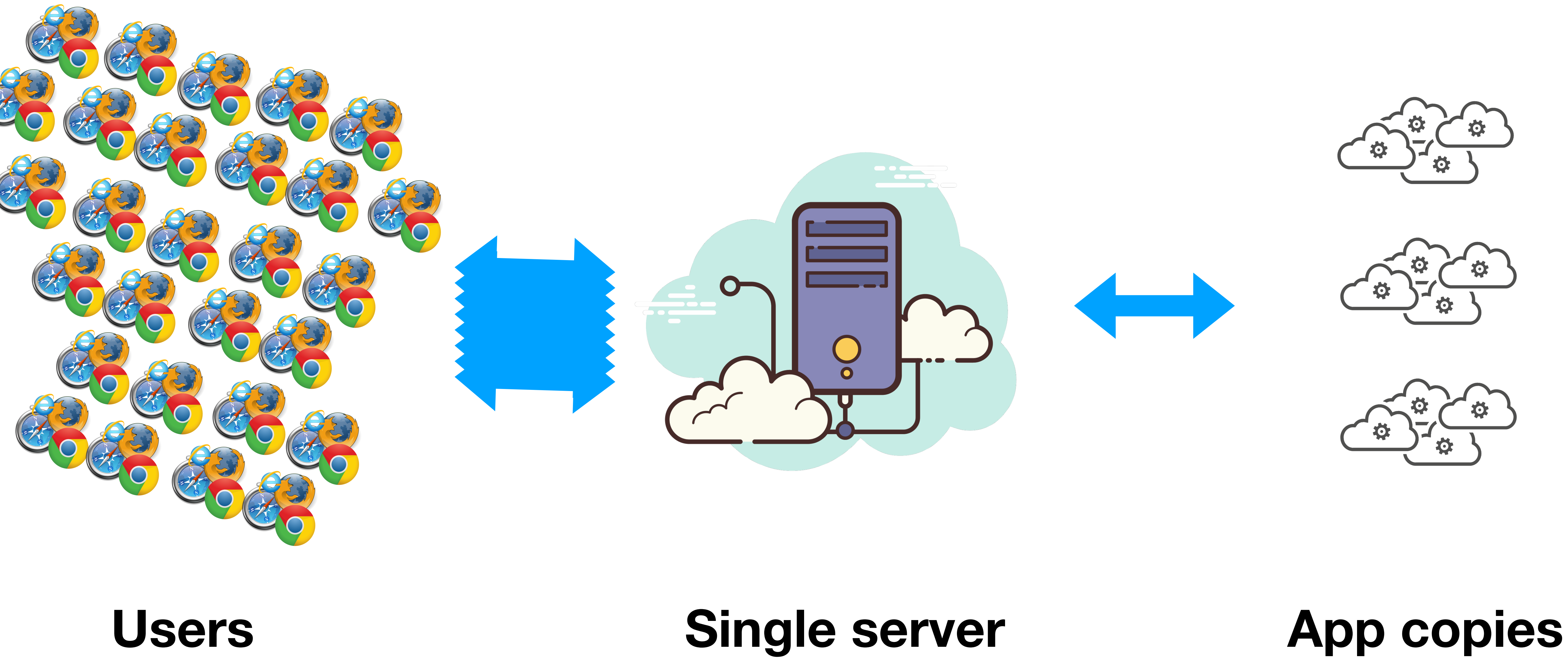
Literature

- GET and POST: https://www.w3schools.com/tags/ref_httpmethods.asp
- Cookie: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- URL Encode: https://www.w3schools.com/tags/ref_urlencode.ASP
- POST Encode: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

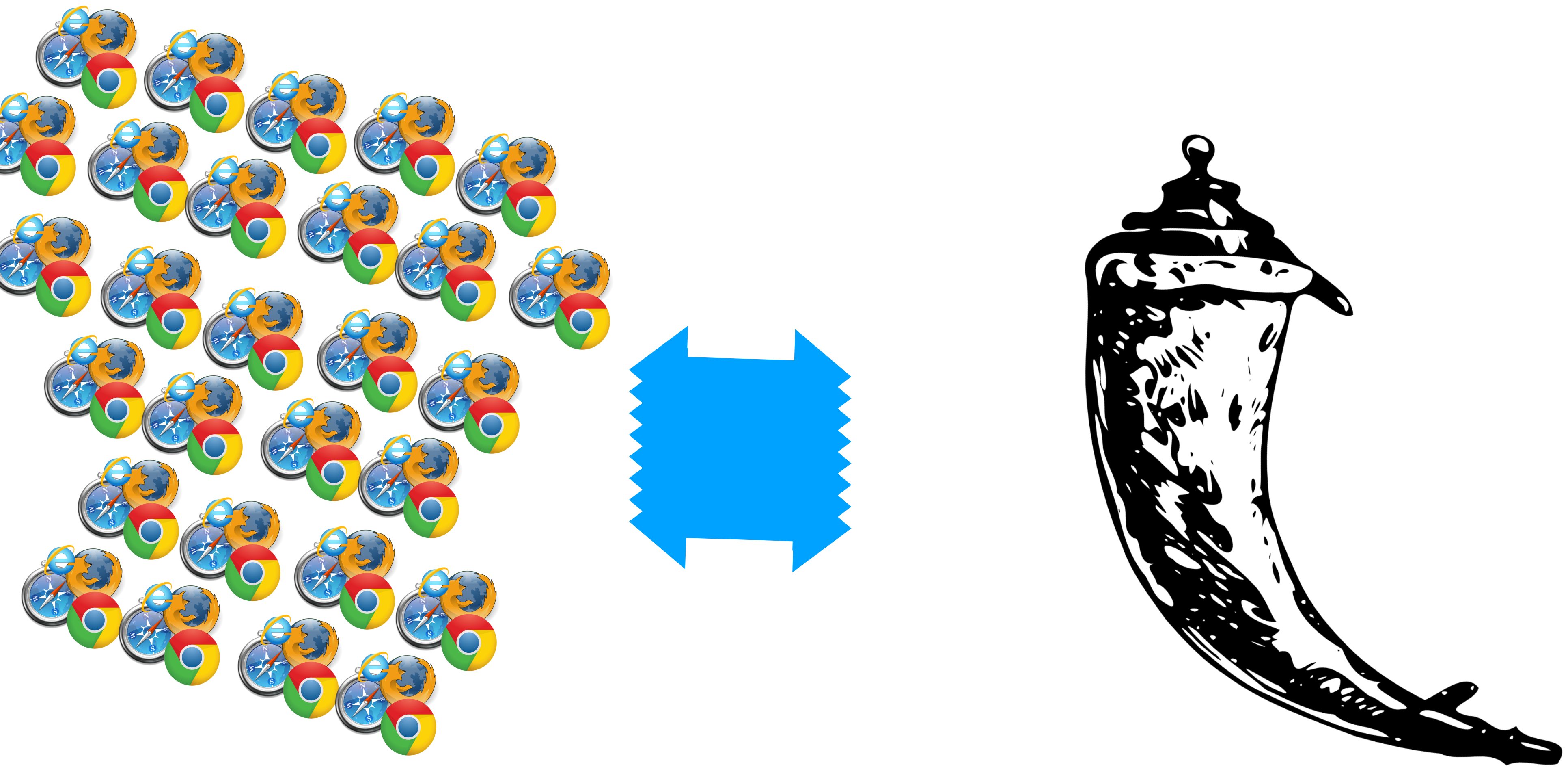
Demo

Flask parallelism

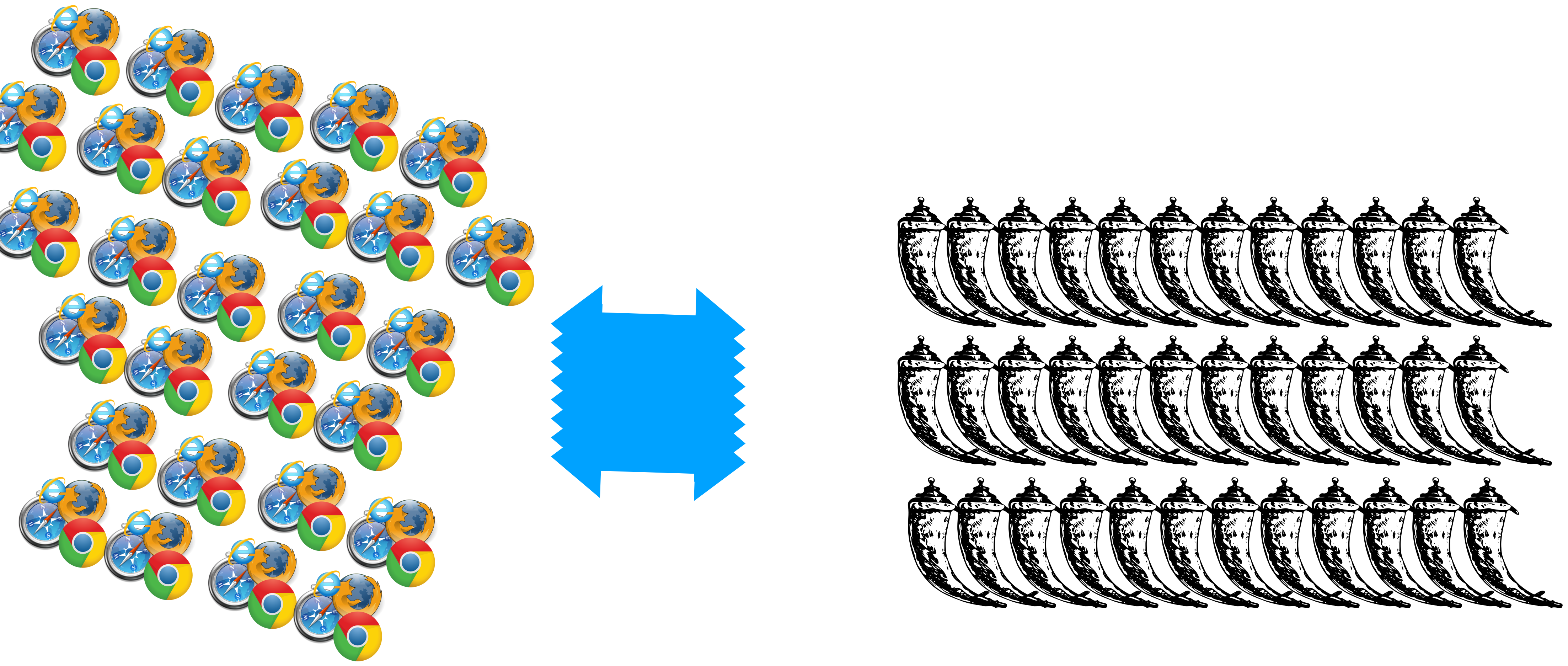
Parallel requests



Flask



Flask



🚀 threaded=**True**

Literature

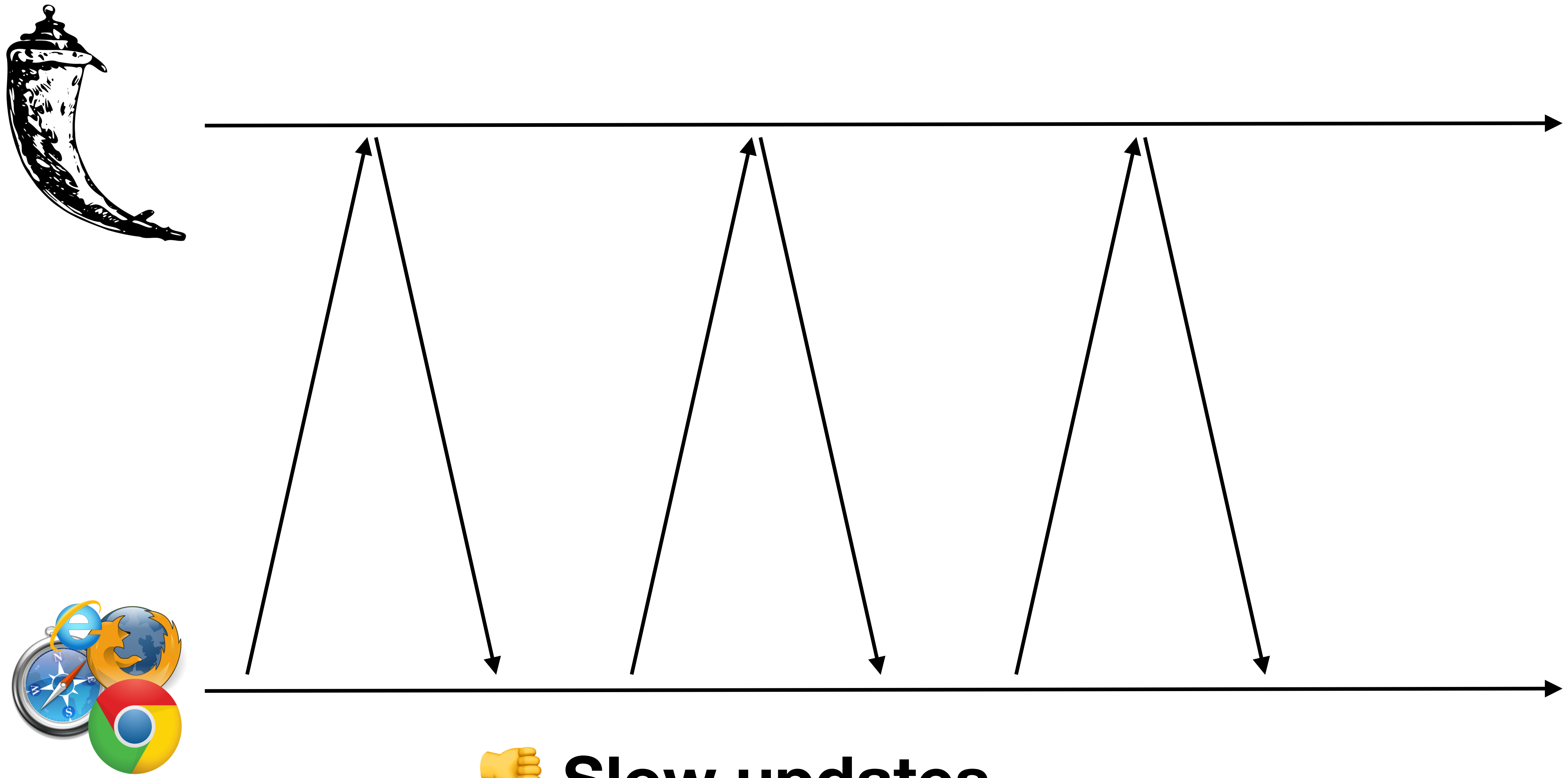
- Python Multithreading and Multiprocessing Tutorial: https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python

Demo

Pub/Sub

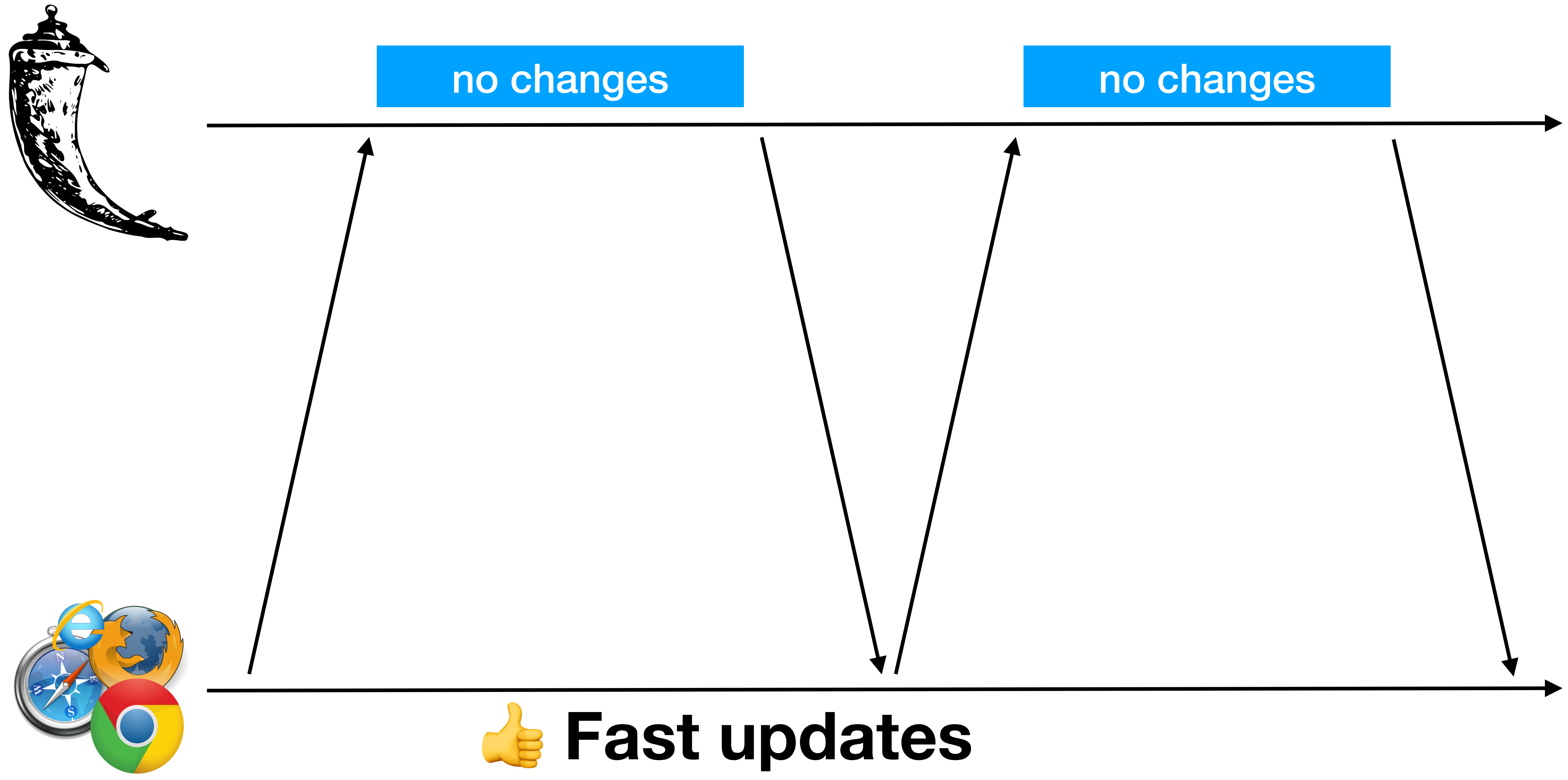
Short polling, Long polling, WebSocket

Short polling



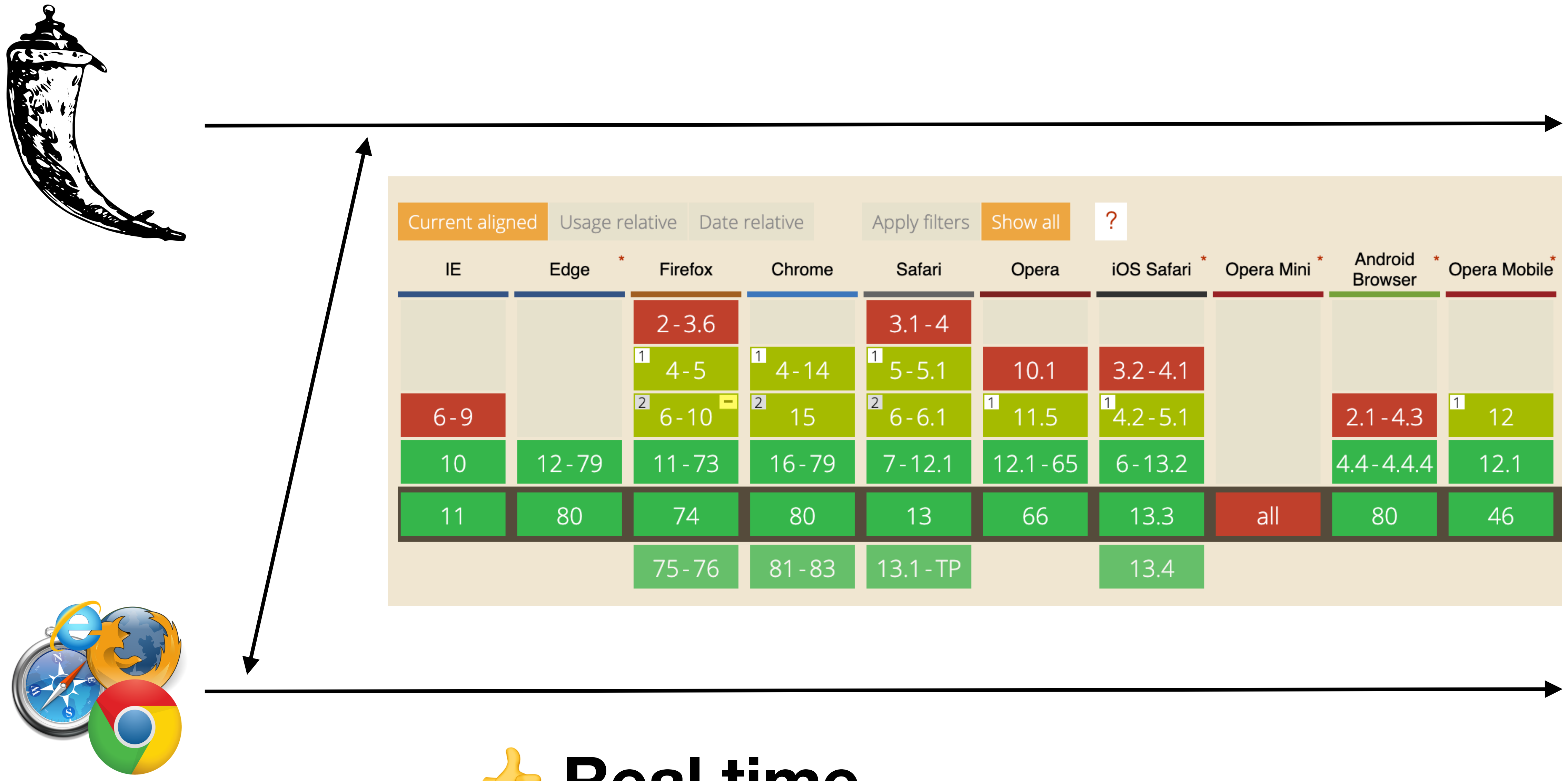
- 👎 **Slow updates**
- 👎 **Resource intensive**
- 👍 **Simple**

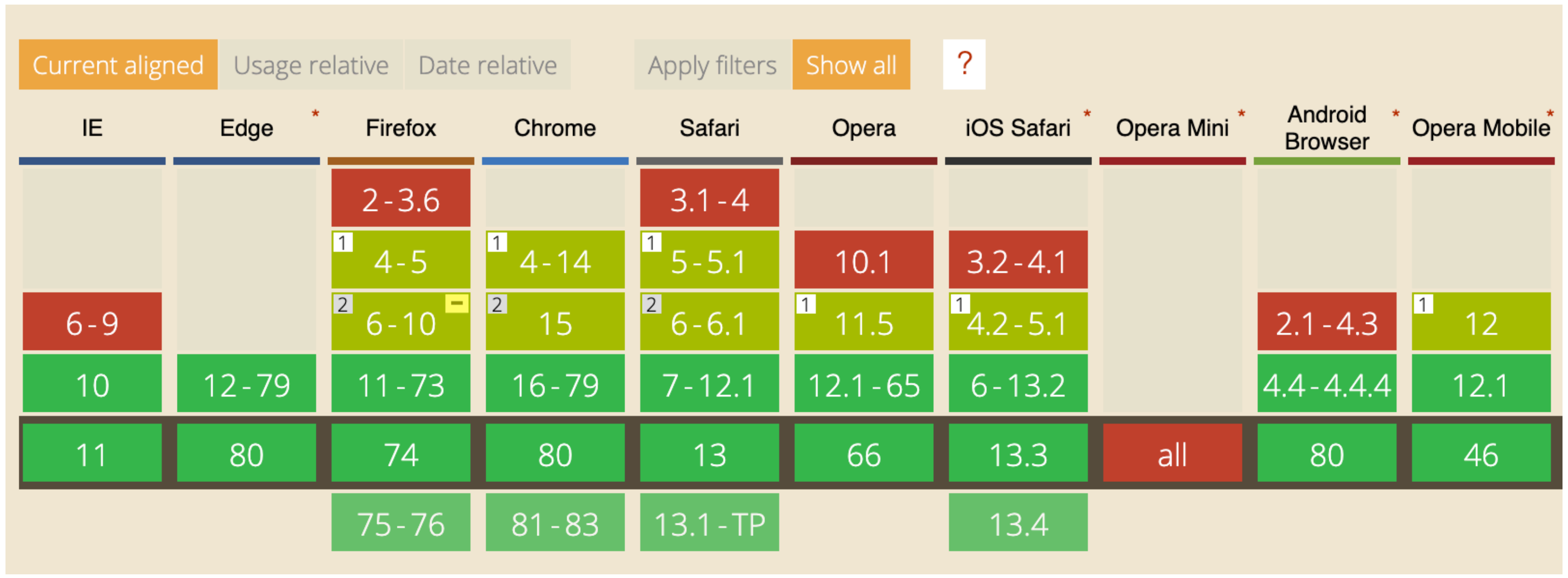
Long polling



- 👍 Fast updates
- 👍 Less resource intensive
- 👎 Kludge
- 👎 Takes 1 worker

Web Sockets





IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *
		2 - 3.6		3.1 - 4					
		[1] 4 - 5	[1] 4 - 14	[1] 5 - 5.1	10.1	3.2 - 4.1			
6 - 9		[2] 6 - 10	[2] 15	[2] 6 - 6.1	[1] 11.5	[1] 4.2 - 5.1		2.1 - 4.3	[1] 12
10	12 - 79	11 - 73	16 - 79	7 - 12.1	12.1 - 65	6 - 13.2		4.4 - 4.4.4	12.1
11	80	74	80	13	66	13.3	all	80	46
		75 - 76	81 - 83	13.1 - TP		13.4			

- 👍 **Real time**
- 👍 **Bidirectional**
- 👍 **Efficient**

Literature

- https://javascript.info/websocket
- https://javascript.info/long-polling
- https://github.com/heroku-python/flask-sockets
- https://www.ably.io/blog/websockets-vs-long-polling/

Demo

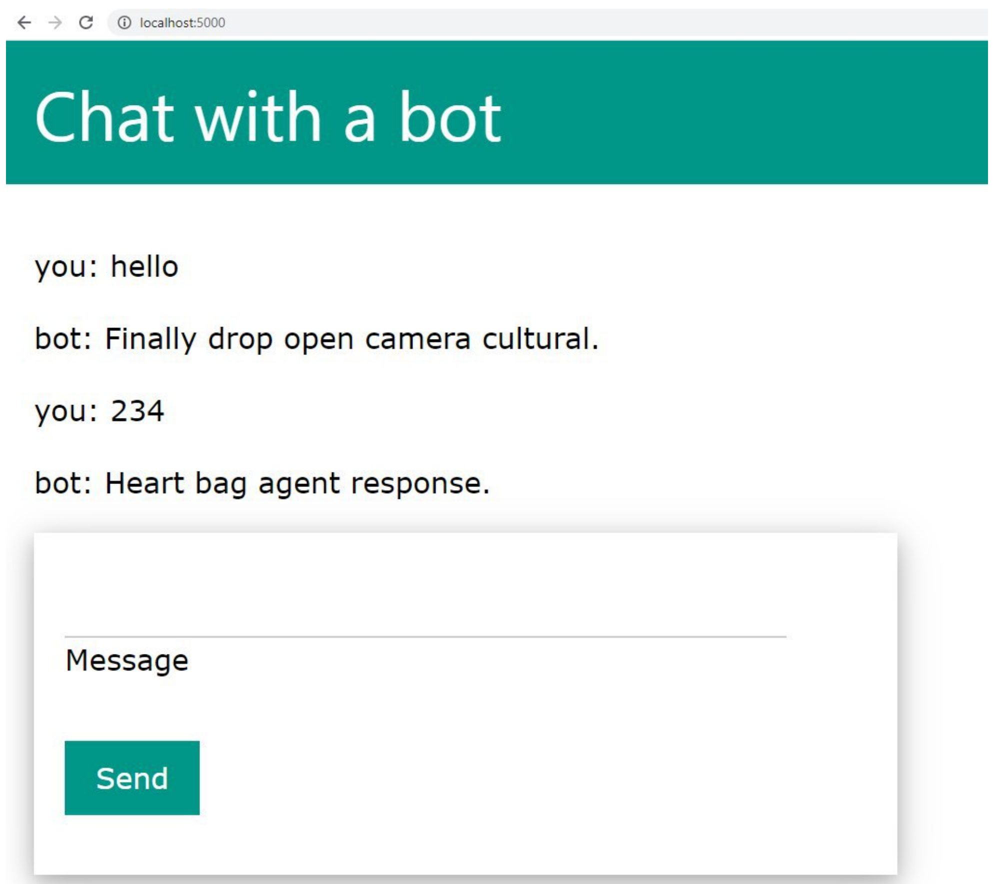
Assignment #2

Preparations

1. Install Python programming language **v3.8.2**: https://www.python.org/downloads/
2. Install Flask framework with **pip**: https://docs.python.org/3/installing/index.html#basic-usage

Basic part

1. Create web application, which can host you image gallery (from the previous week):
   - Listen on `localhost:5000`
   - Render HTML document on `http://localhost:5000/`
   - Return static images on `http://localhost:5000/img/<image_name>`
   - If you use external CSS and JS files, they should be returned on `http://localhost:5000/static/<js/css filename>`
2. You are allowed to use any JS or CSS frameworks
3. You are allowed to use only Python programming language and Flask framework


localhost:5000

Chat with a bot

you: hello

bot: Finally drop open camera cultural.

you: 234

bot: Heart bag agent response.

Optimal part

1. Create web application, which emulates a chat with a human:
   - Web page with messages log
   - Input for writing new message
   - Button for sending message to the server
2. After the button click message should be sent to the server with `HTTP POST` to `http://localhost:5000/`:
   2.1. It is okay to send also all messages if you don't know how to keep them on the server.
   2.2. It is also okay to keep them in the global variable
3. Robot should answer messages based on pre-defined set of rules
   3.1. Rules can be hardcoded as bases on the occurencies of different words
   3.2. There should be at least 10 rules describing typical conversation topics:
   - current weather
   - hello/greetings
   - ...

Challenging part

(Part for those, who already knows all that stuff)

1. Dialog should be kept on the server (global variable or text file)
2. Robot should add new messages independently from user (every second new message)
    - It can be done with another python script
    - Or it can be done with separate Thread
3. Message updates should be delivered to the user's page with the help of three methods (you can create three endpoints for this)
    - polling new messages every 1 second
    - long polling new messages
    - (optionally) through websockets

Deploy

1. Register on GitHub: https://github.com/
2. Join our organization: https://github.com/itmo-wad/
3. Create **new** personal repository for the second task
4. Commit and push your sources to GitHub. And don't forget to describe shortly what have you done in `README.md` file. Use Markdown format: https://guides.github.com/features/mastering-markdown/
5. *(optional)* Deploy you sources on Heroku or somewhere else and add link to the server to `README.md`

Code-review

1. Communicate in Telegram chat
2. Help others to complete the assignment

Literature

- Python Flask big tutorial:
  - https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
  - https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates
- HTML forms: https://www.w3schools.com/html/html_forms.asp
- Markdown: https://guides.github.com/features/mastering-markdown/

Practice time