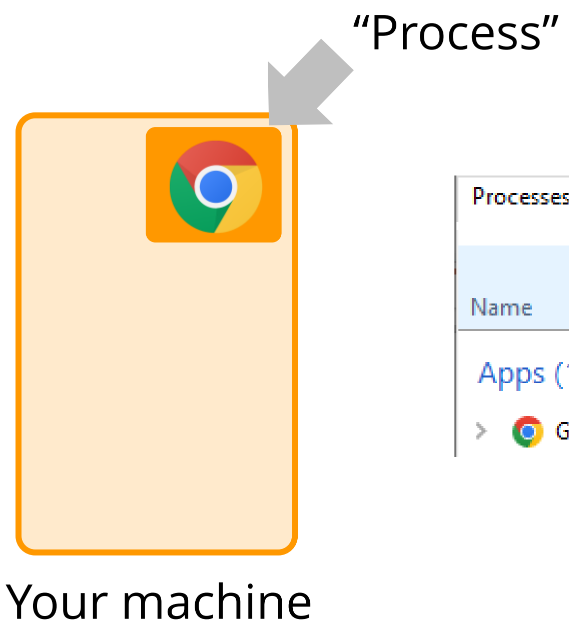
Open-Source Software Practice

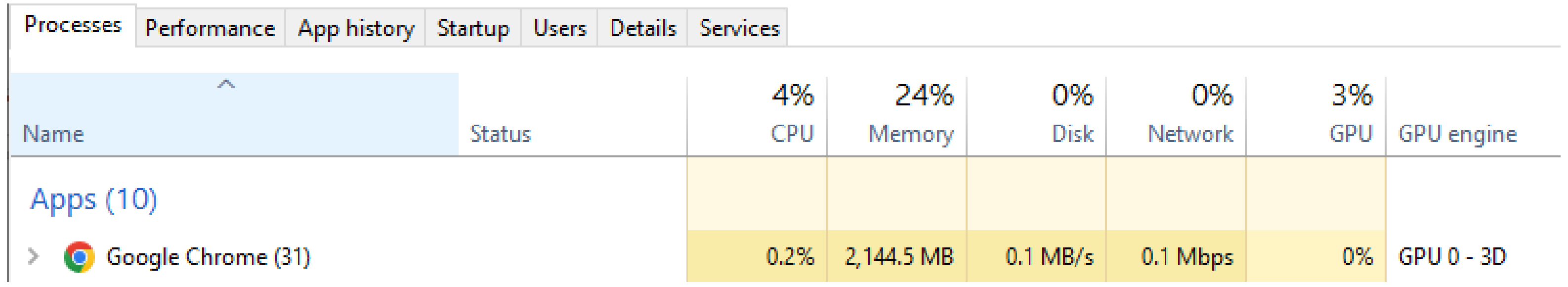
12. Client-Server Model

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

If You Open SKKU-TODO…

- You launch Chrome. The OS spawns a process.

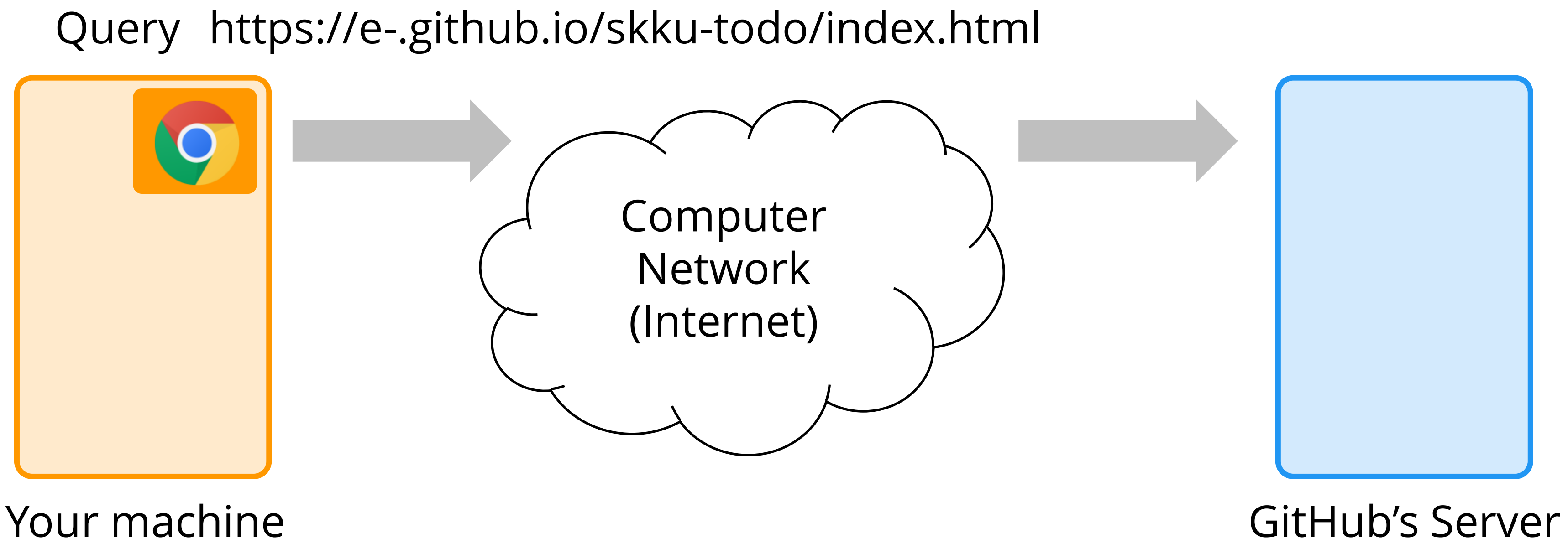




Name	Status	4% CPU	24% Memory	0% Disk	0% Network	3% GPU	GPU engine
Apps (10)							
Google Chrome (31)		0.2%	2,144.5 MB	0.1 MB/s	0.1 Mbps	0%	GPU 0 - 3D

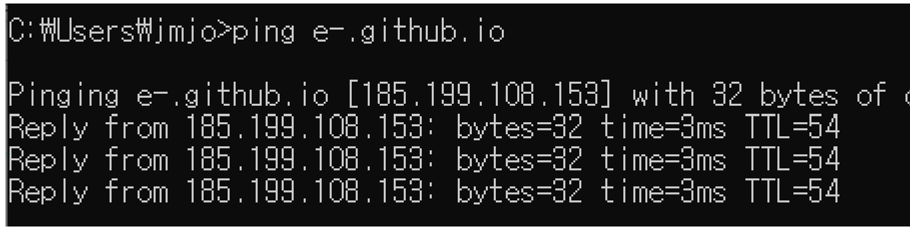
If You Open SKKU-TODO...

- Type “https://e-.github.io/skku-todo/index.html”.

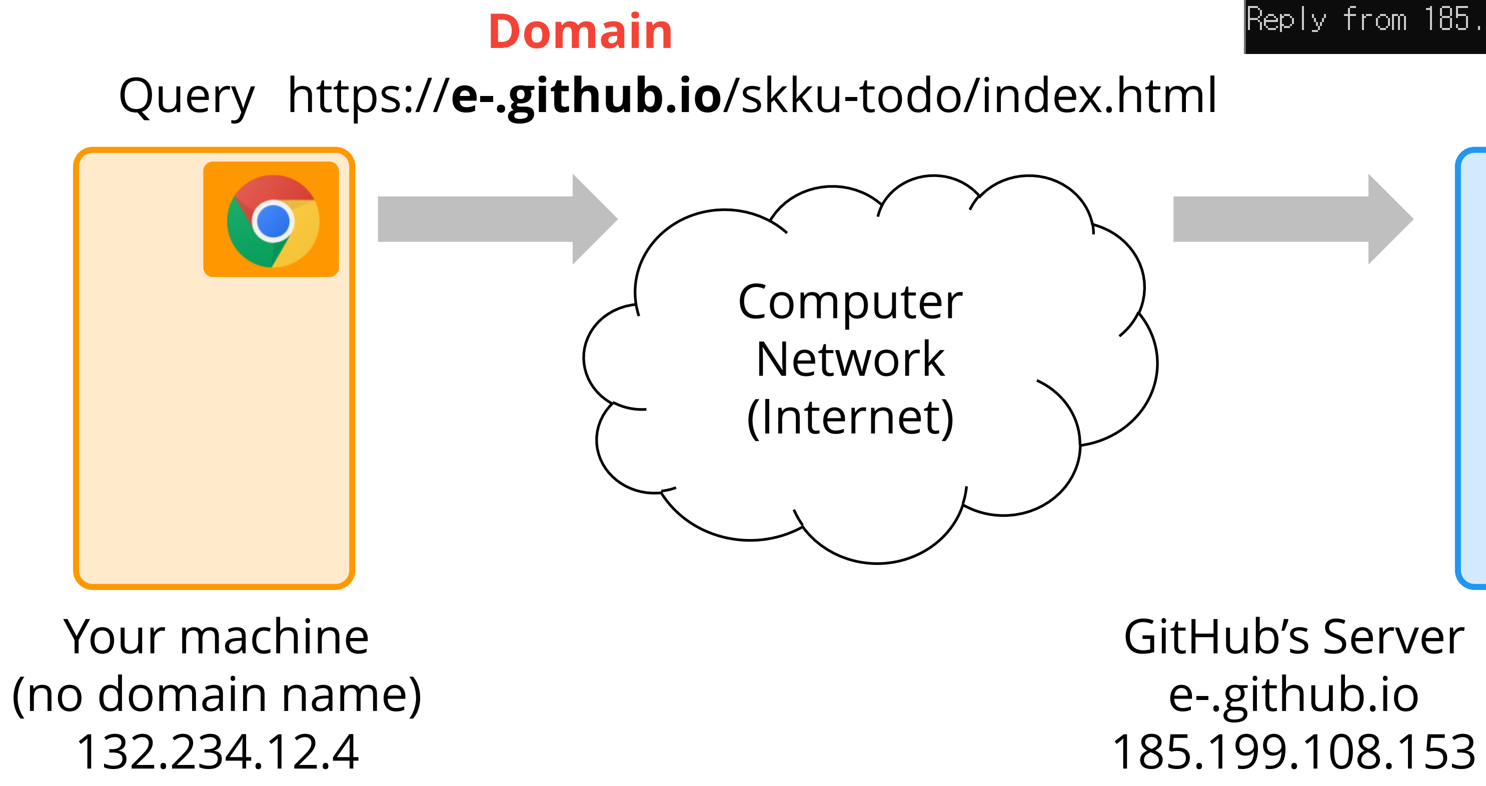

Query https://e-.github.io/skku-todo/index.html
Computer Network (Internet)
Your machine
GitHub’s Server

If You Open SKKU-TODO…

- Type "https://e-.github.io/skku-todo/index.html".

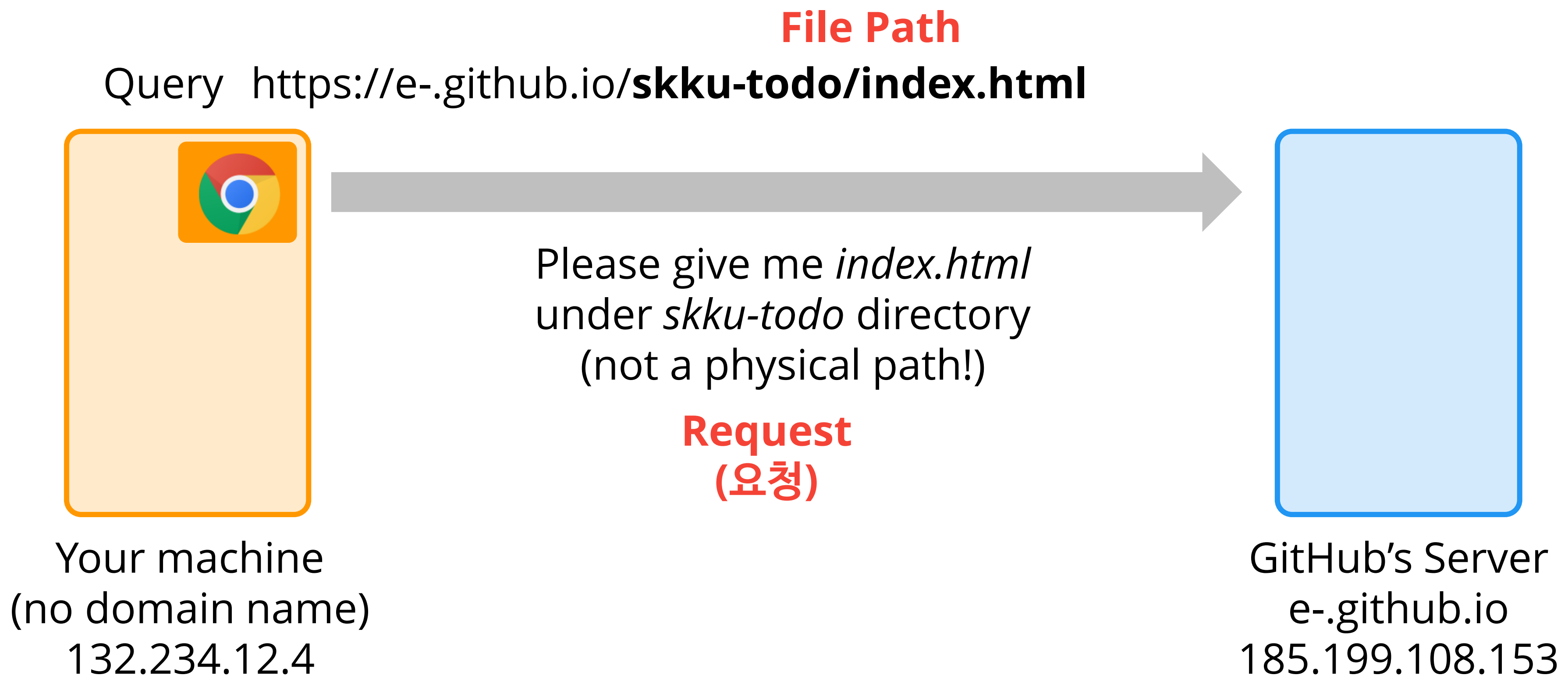


```
C:₩Users₩jmjo>ping e-.github.io

Pinging e-.github.io [185.199.108.153] with 32 bytes of
Reply from 185.199.108.153: bytes=32 time=3ms TTL=54
Reply from 185.199.108.153: bytes=32 time=3ms TTL=54
Reply from 185.199.108.153: bytes=32 time=3ms TTL=54
```


Domain
Query https://**e-.github.io**/skku-todo/index.html
Computer Network (Internet)
Your machine (no domain name) 132.234.12.4
GitHub's Server e-.github.io 185.199.108.153

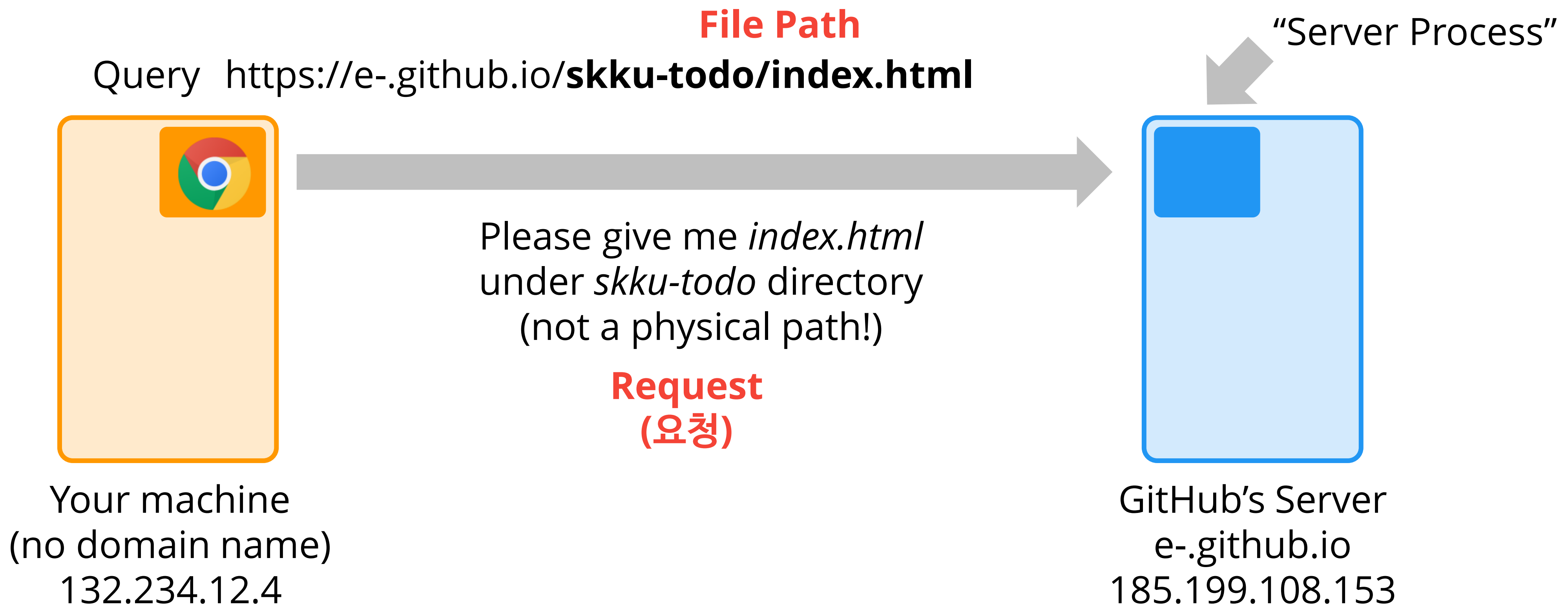
If You Open SKKU-TODO...

- Type “https://e-.github.io/skku-todo/index.html”.


File Path
Query https://e-.github.io/**skku-todo/index.html**
Please give me *index.html* under *skku-todo* directory (not a physical path!)
Request (요청)
Your machine (no domain name) 132.234.12.4
GitHub’s Server e-.github.io 185.199.108.153

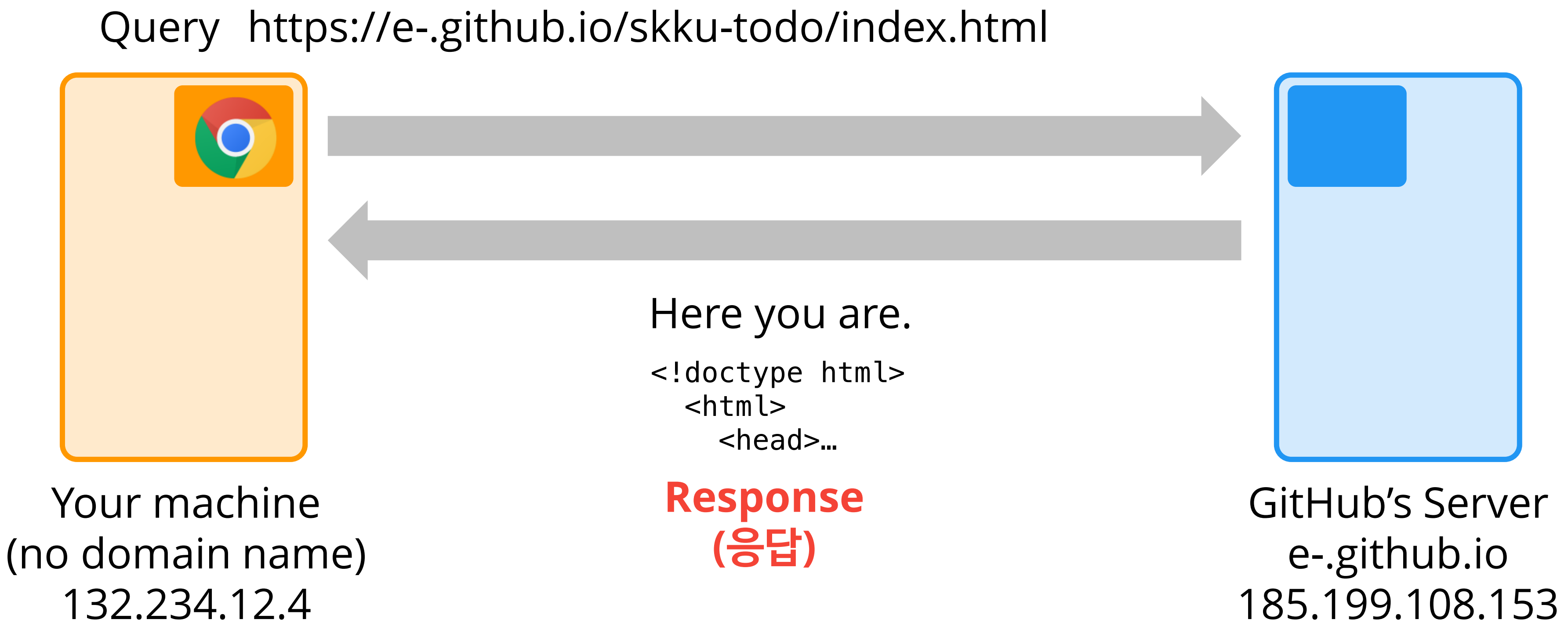
If You Open SKKU-TODO…

- Type “https://e-.github.io/skku-todo/index.html”.


File Path
Query https://e-.github.io/**skku-todo/index.html**
“Server Process”
Please give me *index.html* under *skku-todo* directory (not a physical path!)
Request (요청)
Your machine (no domain name) 132.234.12.4
GitHub’s Server e-.github.io 185.199.108.153

If You Open SKKU-TODO…

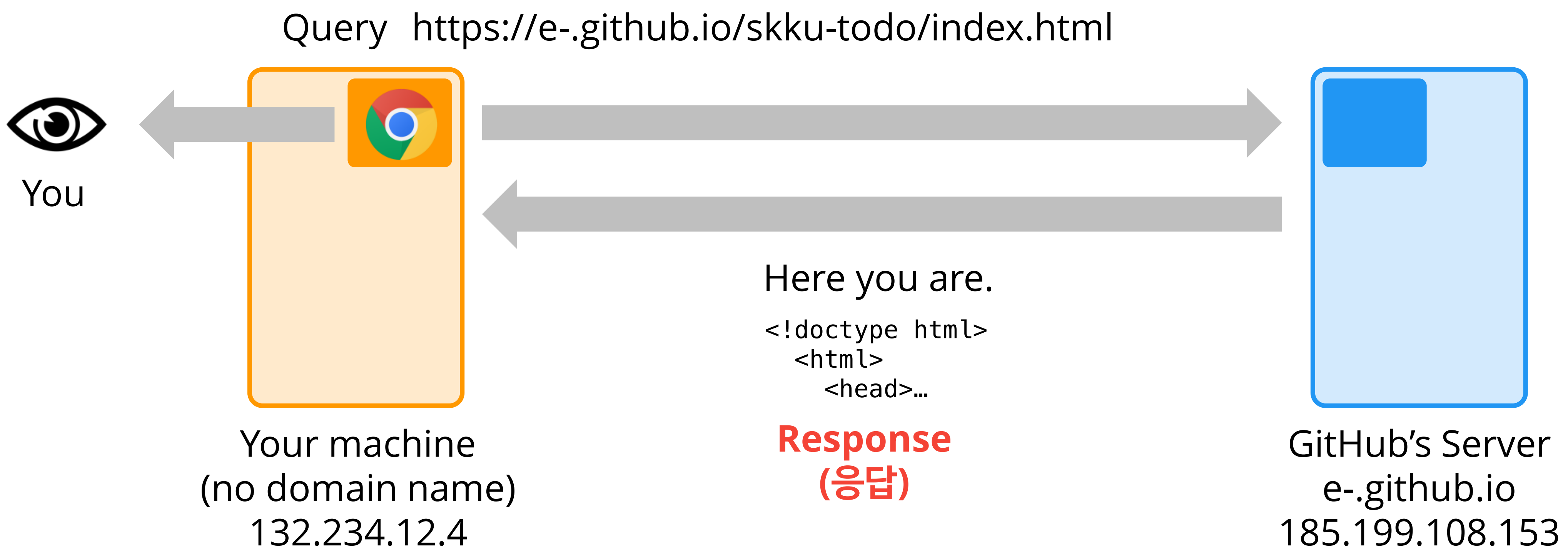
- Type “https://e-.github.io/skku-todo/index.html”.



```
<!doctype html>
  <html>
    <head>…
```

If You Open SKKU-TODO...

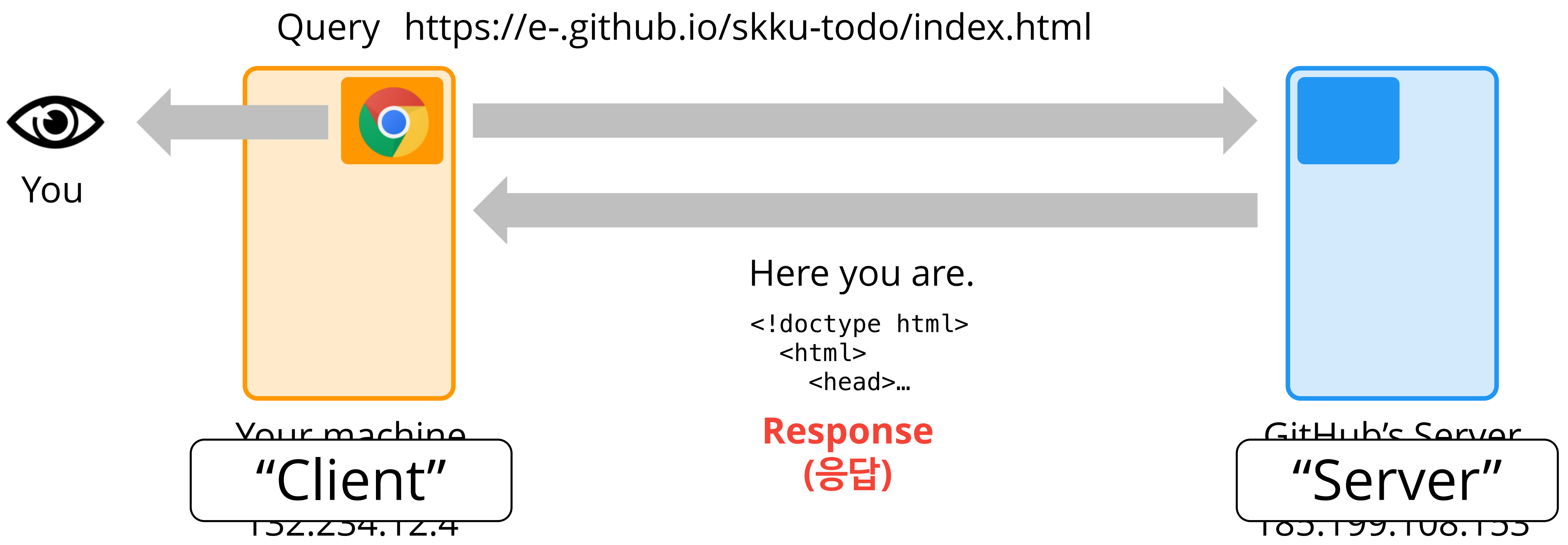
- We see the rendered result.



```
<!doctype html>
  <html>
    <head>…
```

The Client-Server Model

- Client-server model or client-server architecture



The Client-Server Model

- Client
  - Users
  - Make a request
  - Want to get some information or want something useful to happen.
  - Your laptop, smartphone, …
- Server
  - Given a request, make a response.
  - Have resources that a client want to retrieve.
  - Always power on
  - Big computers in a dedicated server room
- Not physically separated. A Web server process is a server for your browser but a client from DB’s point of view.

Protocol

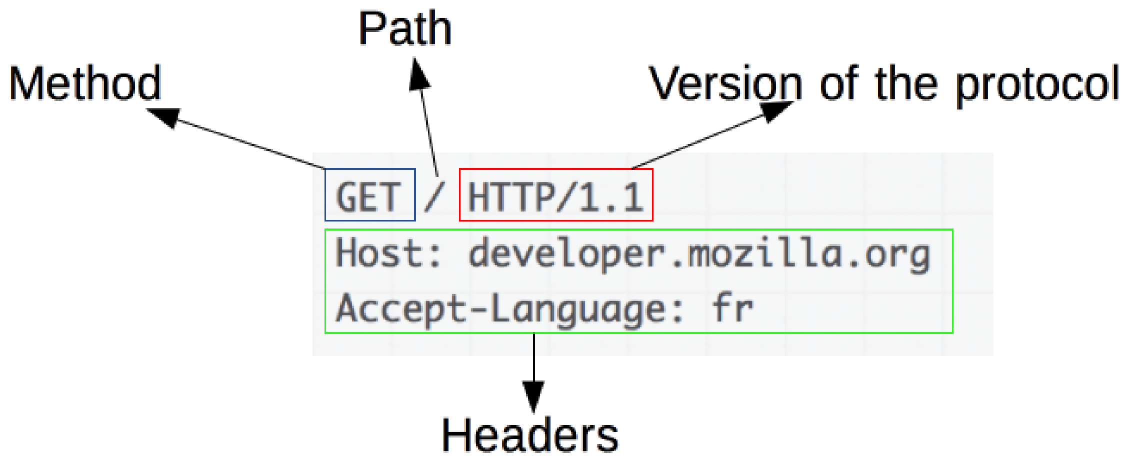
- A client and a server want to communicate with each other.
- They need to speak the same language.

- **Protocol**: the official procedure or system of rules governing affairs of state or diplomatic occasions

- HTTP (HyperText Transfer Protocol)
- HTTPS (Hypertext Transfer Protocol Secure)
- FTP, SSH, POP3, SMTP, …

HyperText Transfer Protocol

- A protocol for fetching resources such as HTML documents.
- A request:

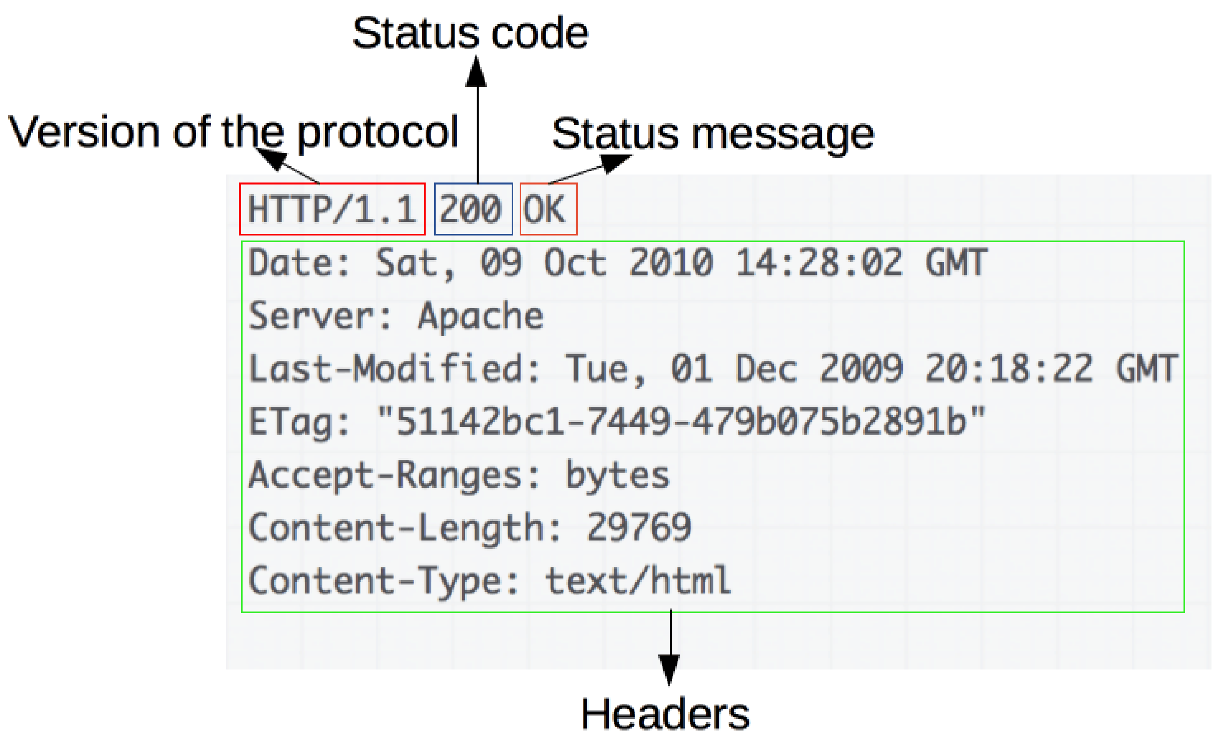


```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

- **Method**: GET, POST, OPTIONS, HEAD, …
- **Path**: the location of the resource
- **Headers**: optional to convey additional information for the servers, e.g., browser information, …

HyperText Transfer Protocol

- A response:



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

… actual content delivered

- **Status Code**: was the request successful?
- **Status Message**: simple text description of the code.
- **Headers**
- **The body**: the actual content of the fetched resource

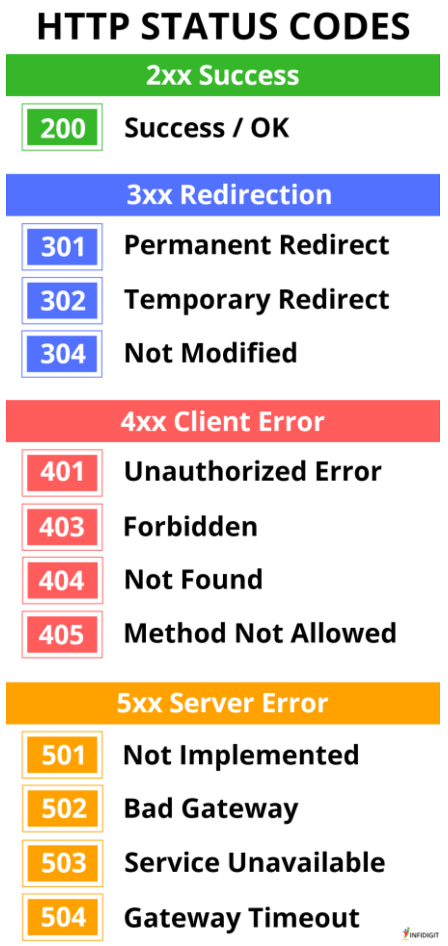
Method of a Request

- **GET**: simply retrieve a resource without changing it.

- **POST**: submit an entity to the server, hoping for changes.
  - Sign up forms: username/password are transferred.
  - Your state should change to “logged in”.
  - Not safe (it changes the server state) nor idempotent (multiple identical requests will not have the same outcome)

- HEAD (same as GET but without the body)
- PUT, DELETE, OPTIONS, …

Status Code of a Response



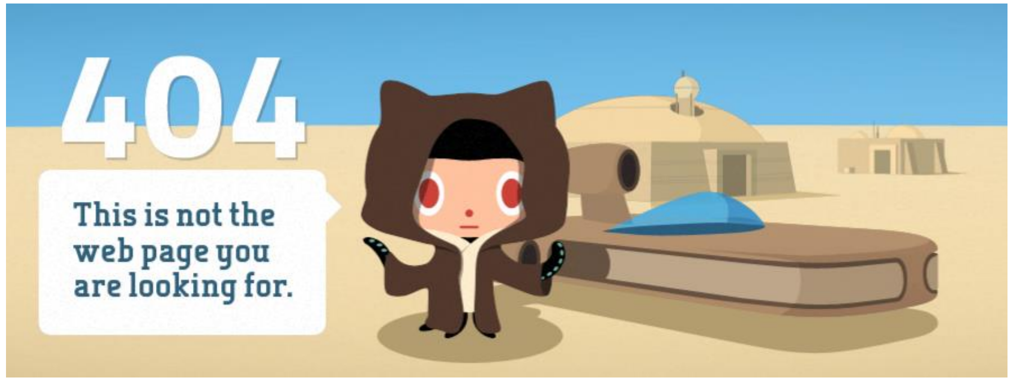
HTTP STATUS CODES

2xx Success	
200	Success / OK

3xx Redirection	
301	Permanent Redirect
302	Temporary Redirect
304	Not Modified

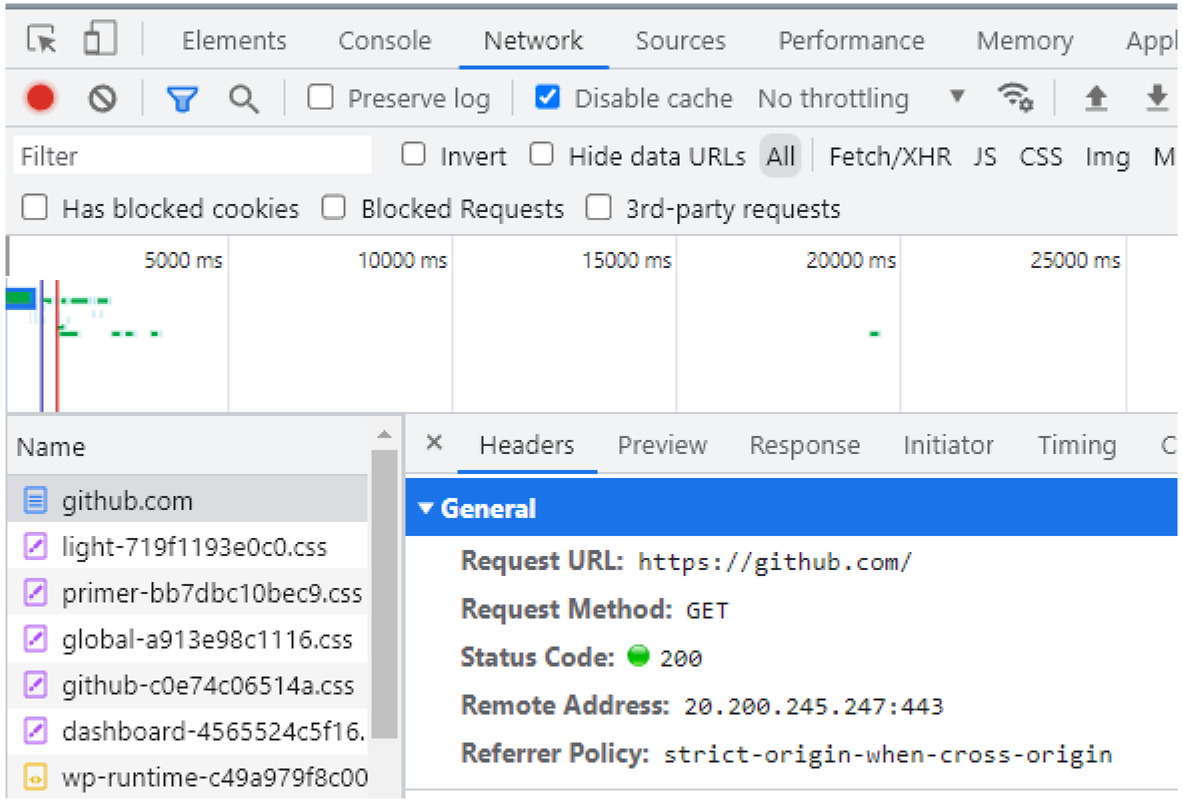
4xx Client Error	
401	Unauthorized Error
403	Forbidden
404	Not Found
405	Method Not Allowed

5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout



HTTP

- Use the inspector tool to see the actual request and response.



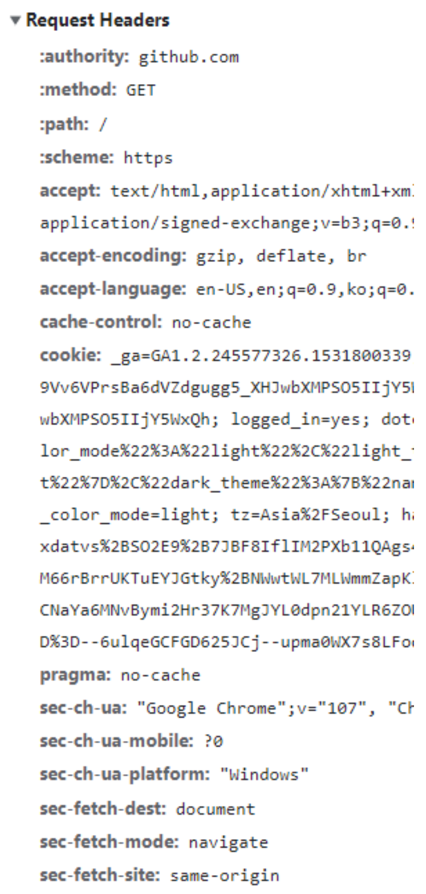
github.com
light-719f1193e0c0.css
primer-bb7dbc10bec9.css
global-a913e98c1116.css
github-c0e74c06514a.css
dashboard-4565524c5f16.
wp-runtime-c49a979f8c00

General

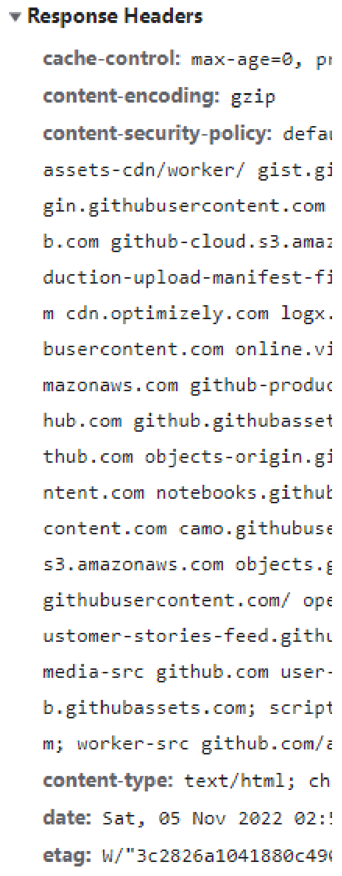
Request URL: `https://github.com/`
Request Method: `GET`
Status Code: ● `200`
Remote Address: `20.200.245.247:443`
Referrer Policy: `strict-origin-when-cross-origin`

HTTP

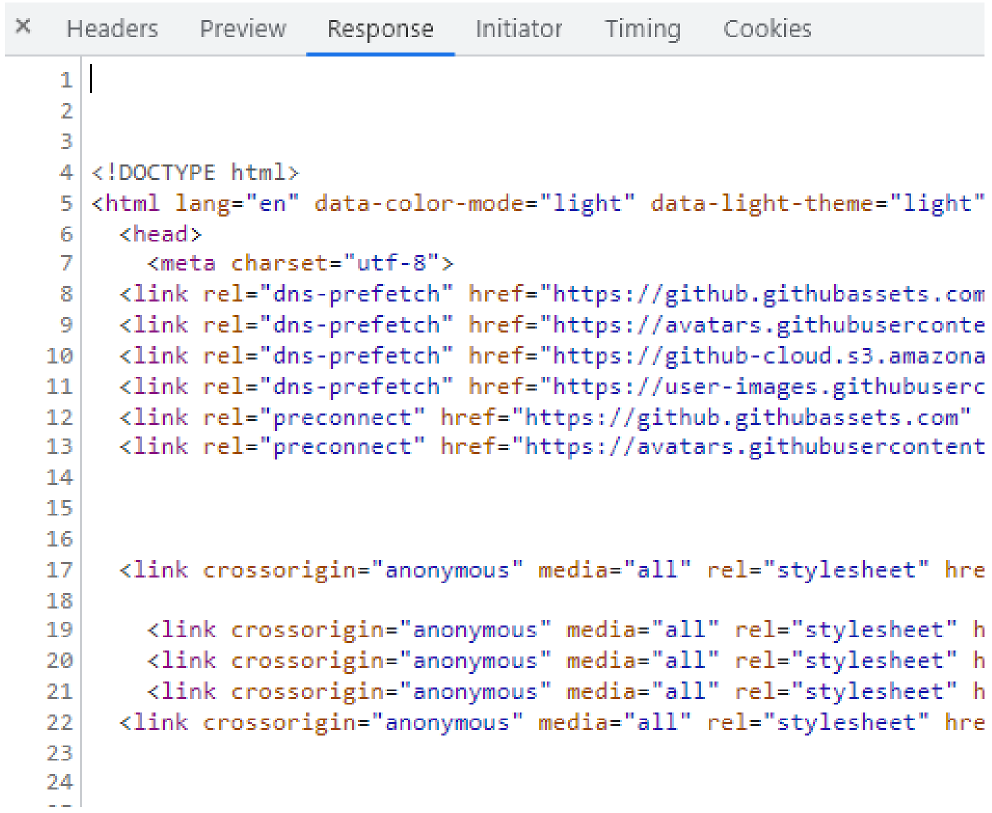
- Use the inspector tool to see the actual request and response.



```
▼ Request Headers
    :authority: github.com
    :method: GET
    :path: /
    :scheme: https
    accept: text/html,application/xhtml+xm
    application/signed-exchange;v=b3;q=0.
    accept-encoding: gzip, deflate, br
    accept-language: en-US,en;q=0.9,ko;q=0.
    cache-control: no-cache
    cookie: _ga=GA1.2.245577326.1531800339
    9Vv6VPrsBa6dVZdgugg5_XHJwbXMPSO5IIjY5
    wbXMPSO5IIjY5WxQh; logged_in=yes; dot
    lor_mode%22%3A%22light%22%2C%22light_
    t%22%7D%2C%22dark_theme%22%3A%7B%22na
    _color_mode=light; tz=Asia%2FSeoul; h
    xdatvs%2BSO2E9%2B7JBF8IflIM2PXb11QAgs
    M66rBrrUKTuEYJGtky%2BNWwtWL7MLWmmZapK
    CNaYa6MNvBymi2Hr37K7MgJYL0dpn21YLR6ZO
    D%3D--6ulqeGCFGD625JCj--upma0WX7s8LFo
    pragma: no-cache
    sec-ch-ua: "Google Chrome";v="107", "C
    sec-ch-ua-mobile: ?0
    sec-ch-ua-platform: "Windows"
    sec-fetch-dest: document
    sec-fetch-mode: navigate
    sec-fetch-site: same-origin
```



```
▼ Response Headers
    cache-control: max-age=0, p
    content-encoding: gzip
    content-security-policy: defa
    assets-cdn/worker/ gist.g
    gin.githubusercontent.com
    b.com github-cloud.s3.ama
    duction-upload-manifest-f
    m cdn.optimizely.com logx.
    busercontent.com online.v
    mazonaws.com github-produ
    hub.com github.githubasse
    thub.com objects-origin.g
    ntent.com notebooks.githu
    content.com camo.githubus
    s3.amazonaws.com objects.
    githubusercontent.com/ op
    ustomer-stories-feed.gith
    media-src github.com user-
    b.githubassets.com; scrip
    m; worker-src github.com/
    content-type: text/html; ch
    date: Sat, 05 Nov 2022 02:
    etag: W/"3c2826a1041880c49
```



× Headers Preview Response Initiator Timing Cookies

```html



<!DOCTYPE html>
<html lang="en" data-color-mode="light" data-light-theme="light"
  <head>
    <meta charset="utf-8">
  <link rel="dns-prefetch" href="https://github.githubassets.com
  <link rel="dns-prefetch" href="https://avatars.githubuserconte
  <link rel="dns-prefetch" href="https://github-cloud.s3.amazona
  <link rel="dns-prefetch" href="https://user-images.githuberc
  <link rel="preconnect" href="https://github.githubassets.com"
  <link rel="preconnect" href="https://avatars.githubusercontent



  <link crossorigin="anonymous" media="all" rel="stylesheet" hre

    <link crossorigin="anonymous" media="all" rel="stylesheet" h
    <link crossorigin="anonymous" media="all" rel="stylesheet" h
    <link crossorigin="anonymous" media="all" rel="stylesheet" h
  <link crossorigin="anonymous" media="all" rel="stylesheet" hre


```

Port

- Wait.. I sometimes see a number in url after the domain. What is that for?

http://www.my-domain.com:8080/page/I/want/to/see

Protocol I will use
http://
https:// (default)
…

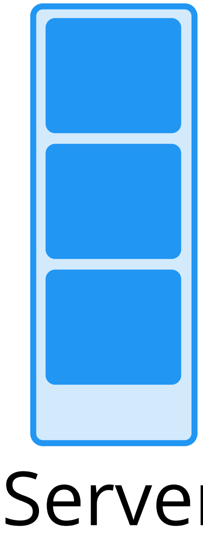
(Optional) the port
of the host that I
want to connect.

Port

- Wait.. I sometimes see a number in url after the domain. What is that for?

http://www.my-domain.com:8080/page/I/want/to/see

A host computer can have multiple server processes at the same time.



Process 1: HTTP, listening to 80 (default)

Process 2: HTTPS, listening to 443 (default)

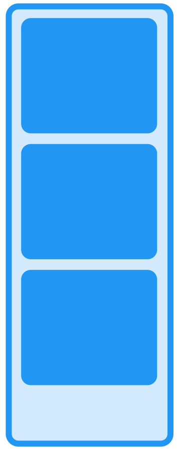
Process 3: RDP, listening to 3389 (default)

Port

- Wait.. I sometimes see a number in url after the domain. What is that for?

http://www.my-domain.com:8080/page/I/want/to/see

A host computer can have multiple server processes at the same time.



Process 1: HTTP, listening to 80 (default)

Process 2: HTTPS, listening to 443 (default)

Process 3: HTTP, listening to 8080

Let’s Make a Server

- Frameworks for the server-side development:

- Express.js (JS)
- Django (Python)
- Flask (Python)
- Spring (Java)
- Ruby on Rails (Ruby)
- ~~PHP~~
- …

Express.js

- **Express.js**: a very minimal yet powerful web framework for node.js

- `npm install express`
  - We will use 4.x.

- Let’s write a “Hello, World!” server.

- https://github.com/e-/skku-chat

Hello, World!

- Type the following code in *main.js*.
- `node main.js`
  - Ctrl + C to exit

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello, World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Hello, World!

- Navigate to http://localhost:3000 or http://127.0.0.1:3000
- You will see the "Hello, World!" message.

```
const express = require('express')

const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello, World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

- Import express.js

- Create a new app

- An event-listener-like logic specification
- "If you get a GET request on "/", send "Hello, World" to the client"

- Run the app on port 3000.
- The process will go into an infinite loop until Ctrl+C.

Serving a File

- We could send a text message to the client as a response.
- What about sending an HTML code?

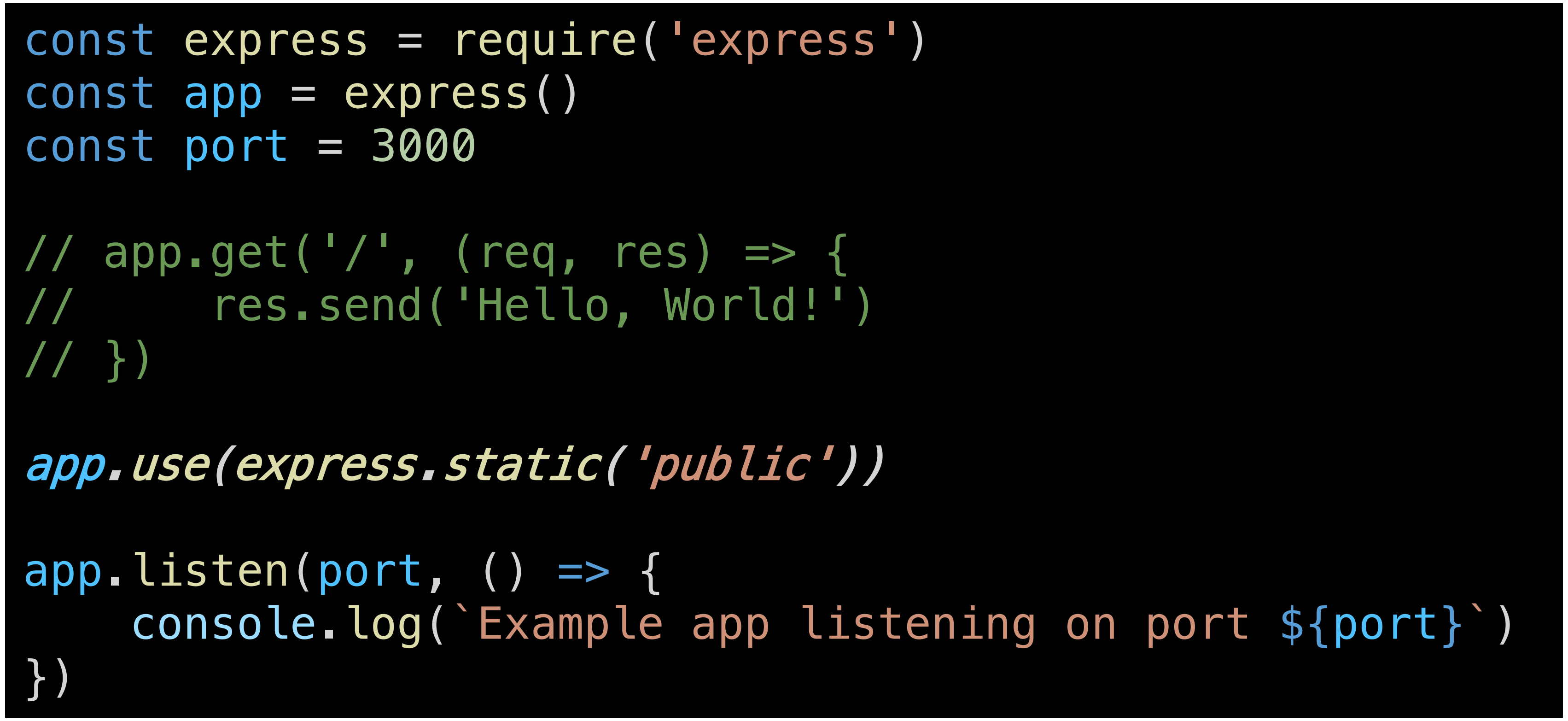
- Create a directory “public”.
  - The files under this directory will be directly served to clients.
  - Don’t put any private files., e.g., *main.js* (source code of the server itself)

- Create *index.html* under *public*, and type the code on the next page.

Client Code

```html
<!doctype HTML>

<html>

<head>
    <title>SKKU-CHAT</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
</head>

<body>
    <div class="container">
        <p>
            Visits so far: <span id="counter"></span>
        </p>
    </div>

    <script>
    </script>
</body>

</html>
```

Server Code



```
const express = require('express')
const app = express()
const port = 3000

// app.get('/', (req, res) => {
//     res.send('Hello, World!')
// })

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Visits so far:

Counting the Visits

- Let’s make something useful: visit counter.
- Once the page is loaded, send a request to the server ***again*** (without a refresh!) to load the # of visitors.

- You can issue a request using JS without refreshing the page.
- Long ago, this was new and specially called AJAX (Asynchronous JavaScript and XML).
- But these days, we don’t consider it special since everyone uses it. It became de facto standard of the Web.

The fetch() API

- How can we make a request using JS? Use the fetch() API.

- Example: `fetch(`*`url`*`, `*`options`*`).then(`*`handler`*`)`

```
fetch(
  “localhost:3000/counter”,
  {method: “GET”})
.then((count) => { })
```

Client Code

```
<body>
    <div class="container">
        <p>
            Visits so far: <span id="counter"></span>
        </p>
    </div>

    <script>
        window.addEventListener("load", () => {

            fetch("/counter", { method: "GET" })

                .then((res) => res.text())

                .then((count) => {
                    let counter = document.getElementById("counter");
                    counter.textContent = count;
                })
        })
    </script>
</body>
```

- When the page is loaded,
- fetch the “/counter” resource,
- get the result as text,
- and put the text into the #counter element.

Server Code

```
const express = require('express')
const app = express()
const port = 3000

let counter = 0;

app.get('/counter', (req, res) => {

    counter++;

    res.send(counter.toString())
})

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

- When someone requests “/counter”,
- increment the global counter variable,
- and returns the counter as text.

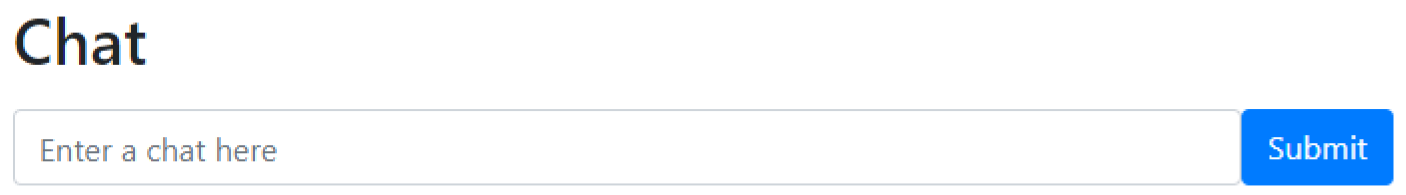
Visits so far: 8

Simple Chat App

- In the previous example, we just requested a resource to the server without attaching extra data.
- Can we send data to the server?

- Let's make a very simple chat app.

Client Code - HTML



```html
<h2>Chat</h2>
<ul id="chats">

</ul>
<div class="d-flex">
    <input type="text" id="chat" class="form-control"
placeholder="Enter a chat here">
    <button type="submit" id="submit" class="btn btn-
primary">Submit</button>
</div>
```

Client Code - JS

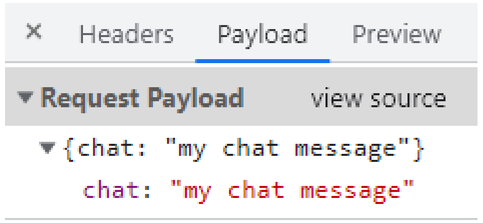
- Getting the messages from the server.
- Clear the list and append the <li> elements for the messages.

```
function loadChats() {
    fetch("/chats", { method: "GET" })
        .then((res) => res.json())
        .then((chats) => {
            let list = document.getElementById("chats");
            list.innerHTML = "";

            chats.forEach(chat => {
                let li = document.createElement("li")
                li.textContent = chat;
                list.appendChild(li);
            })
        })
}
```

Client Code - JS

- When the user clicks on the submit button, read the text in the <input> tag and send it to the server as a JSON object.

```
document.getElementById("submit").addEventListener("click", () => {
    let input = document.getElementById("chat");
    let chat = input.value;

    fetch("/chats", {
        method: "POST",
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ chat })
    })
        .then(() => {
            loadChats();
        })
})
```



```
▾{chat: "my chat message"}
   chat: "my chat message"
```

Simple Chat Server

```
const express = require('express')
const app = express()
const port = 3000

let counter = 0;

app.get('/counter', (req, res) => {
    counter++;
    res.send(counter.toString())
})

app.use(express.json()) // for parsing application/json

let chats = [];

app.get('/chats', (req, res) => {
    res.send(chats);
})

app.post('/chats', (req, res) => {
    chats.push(req.body.chat);
    res.send(200);
})

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Server Push

- Open “localhost:3000” on Tab 1.
- Submit a chat message “Hello”
- Open “localhost:3000” again on Tab 2.
- The message should be visible on Tab 2.

- If we enter another message in Tab 1, is Tab 2 updated? No.
- Tab 2 does not know whether there is a new message on the server.

Server Push

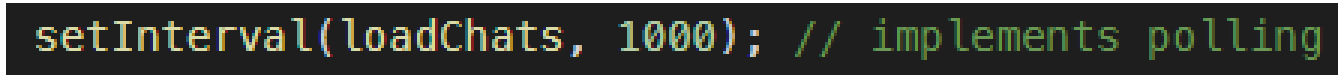
- **Limitation of HTTP 1.1**: the client only initiates a request.
- There is no way for the server to notify the client of something.
  - e.g., chat application

- However, there ARE chat applications on the Web. How do they work?
- They are using the “server-push” technologies, e.g., WebSocket.

- But, we are not going further about this.
- Anyway, “real-time” Web applications are everywhere these days.

Server Push

- One workaround to this problem is to use “polling”.
- Load the messages from the server every one second.
- Simple but inefficient if there is no update.



```
setInterval(loadChats, 1000); // implements polling
```

SKKU-Chat

- If you exit a server process, all messages are gone. You need to store the messages if you want them to be persistent. Use a database.
- Others cannot connect your server if your IP is not public. If you want to host a server, you need to have a public and static IP address.
- You cannot fetch pages under a different domain (e.g., accessing google.com:80 from localhost:3000).
- The chat app we developed is unsafe. Never use it in the real world.
- GitHub Pages will not work for SKKU-Chat since it simply delivers the files statically without actually executing the *main.js* script.

Python Open Source?

- Different programming languages have different ecosystems.
- However, they share important concepts!
  - A package manager, package index, metadata files…

	JavaScript (on Node.js)	Python
Command	node	python python3
Package Index	NPM	PyPI
Package manager command	npm	pip
Metadata	package.json	setup.py
Publish	npm publish	twine upload

Summary: OSSP

- Git and GitHub
- VSCode, Plugins, editorconfig, ...
- Node, JavaScript, HTML, and CSS
- Unit Testing
- Automation using GitHub Actions
- Publishing using GitHub Pages
- Cross-platform apps using Electron
- Collaboration (pull requests, issues, ...)
- Client-server model

+ a command line app, a Web app, a desktop app, a Web page, and a simple chat app that uses the client-server architecture

Summary: OSSP

If you have any further questions about Git, GitHub, open-source projects, web development (HTML, CSS, JS, Bootstrap, TypeScript, ...), server dev., being a successful undergrad, Data Science, Human-AI Interaction, etc., please feel free to hit us up.

- Jaemin Jo@*IDCLab* (jmjo@g.skku.edu)
- Jiwon Choi@*IDCLab* (jasonchoi3@g.skku.edu)