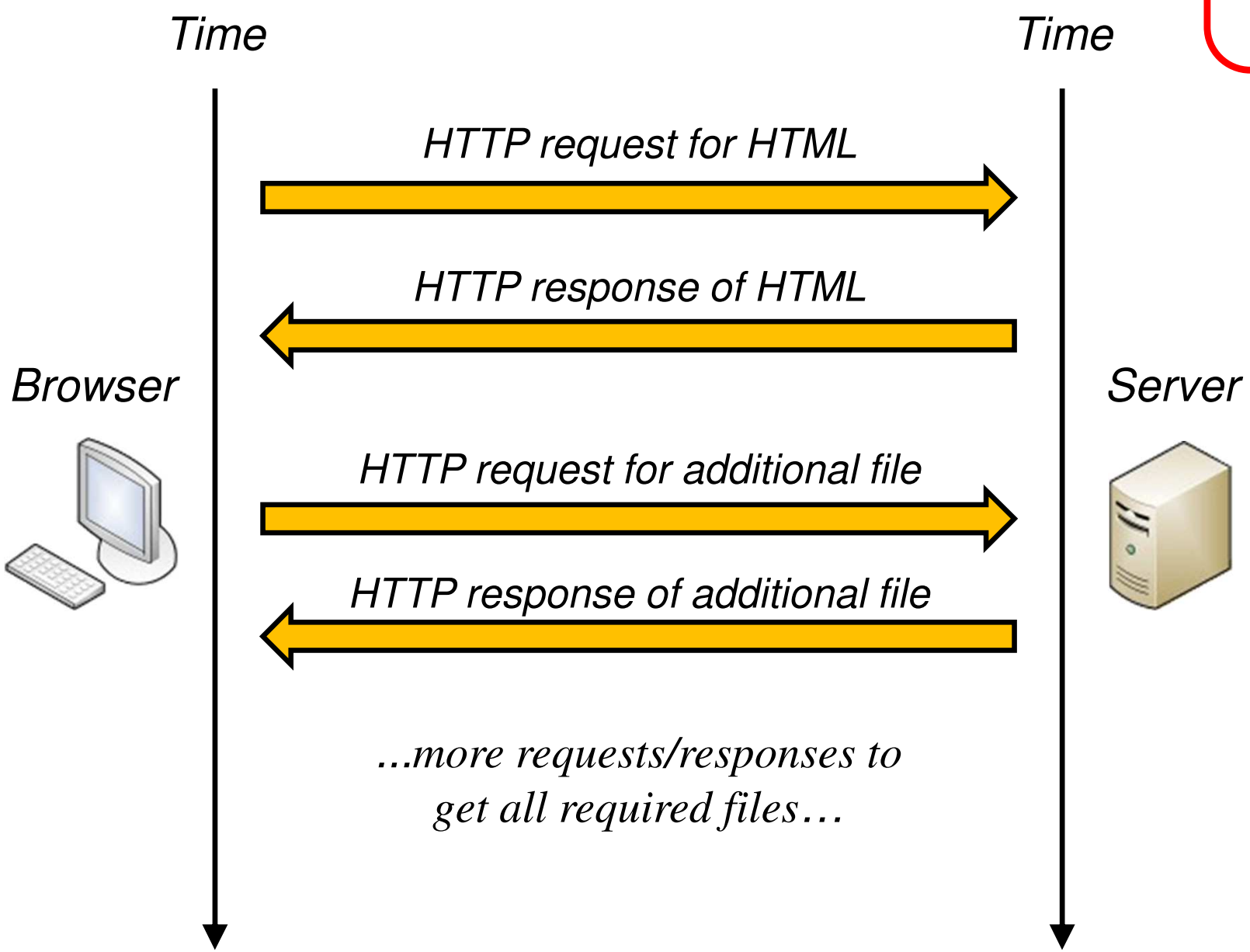
COMP4021
Internet Computing

Using AJAX

Gibson Lam

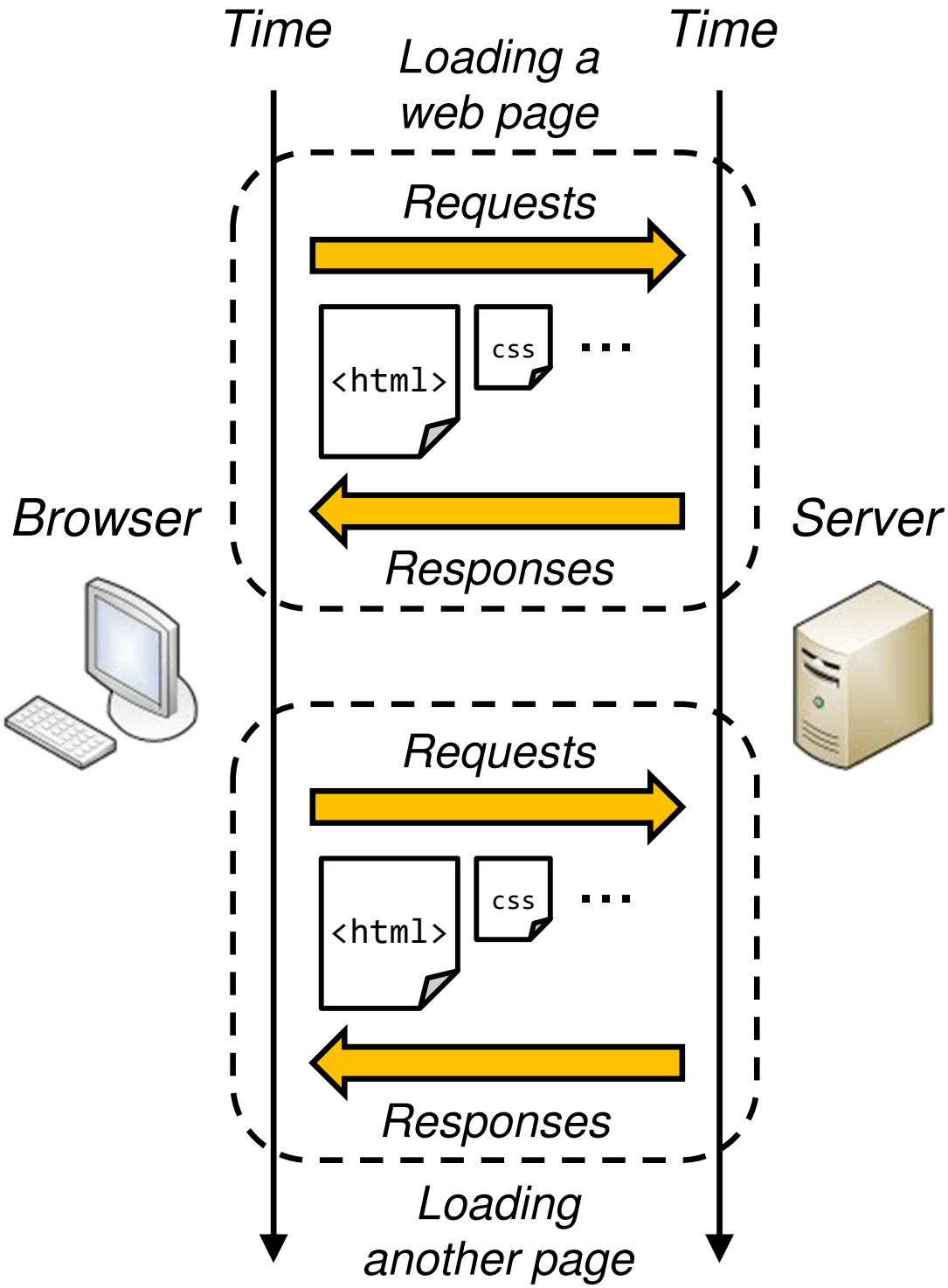
Basic HTTP Process

You have seen this before



Updating Page Content

- The basic approach loads a web page and its related files in a set of requests and responses
- If the page content is updated, a new set of requests and responses is used to load the page or a brand new page

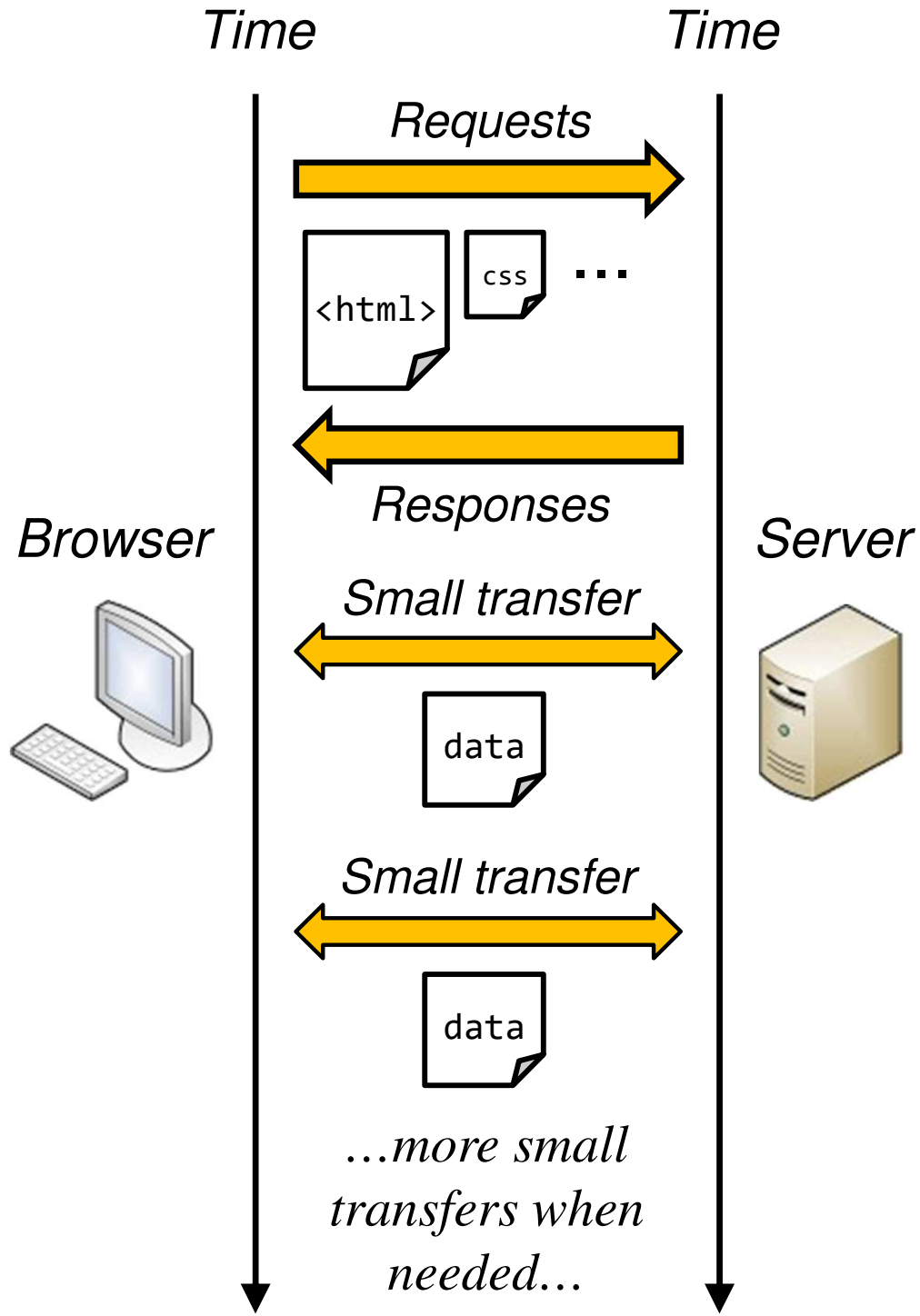


Issues With the Approach

- You probably see that there are two issues with this approach:
  - Duplicated content transfer
    - When content is repeated among web pages, which is likely in the same web site, time is wasted in transferring the duplicate content
  - Slow response time
    - The response time is slow when web pages frequently change and update their content

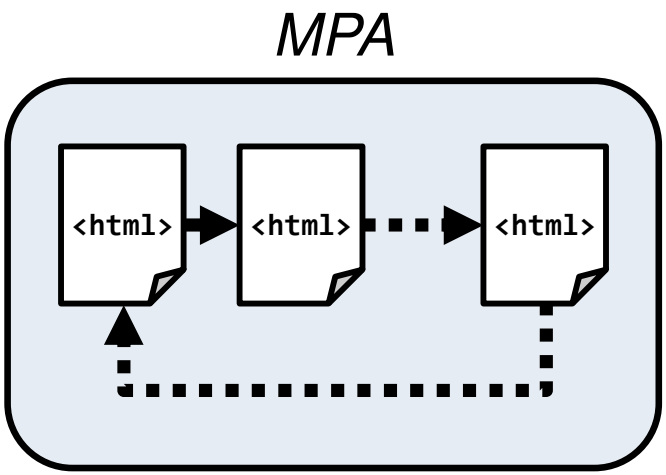
A Dynamic Approach

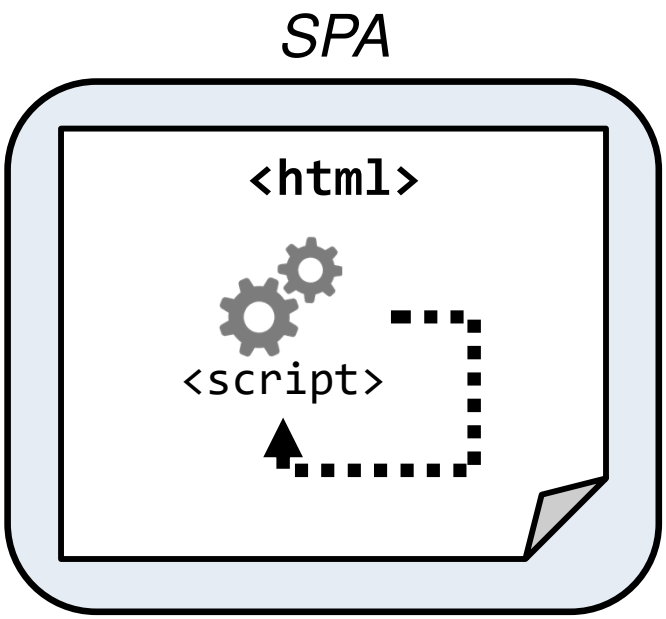
- A relatively big transfer happens at the start when the web page is first loaded
- Every update on the page is done by some small data transfers using JavaScript



Single and Multi Page Applications

- When building a web application using the basic approach, it is called a multi-page application (MPA)
- If the dynamic approach is used, it will become a single-page application (SPA)



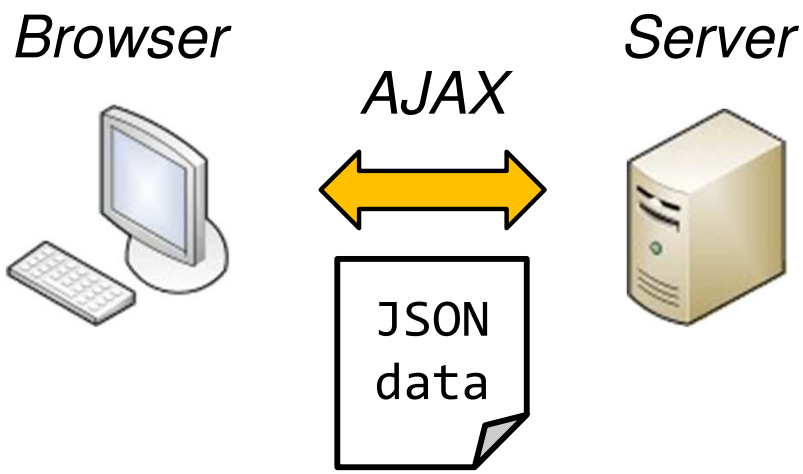


SPA vs MPA

- SPA:
  - Better user experience in dynamic applications
  - Can build rich applications with sophisticated UI
  - Bad for search engine indexing
  - Heavy workload on the front-end
- MPA:
  - Simpler development
  - Good for search engine indexing
  - Slower for dynamic applications
  - Better security as everything stays on the server

AJAX

- You can make those small transfers in the dynamic approach using AJAX, which stands for **A**synchronous **J**avaScript **a**nd **X**ML
- Although AJAX has XML in its name, data transfers are usually done using JSON nowadays



The AJAX Process

- AJAX is basically an HTTP request made inside JavaScript
- You can do almost everything that a browser does using AJAX, for example:
  - Making GET or POST request
  - Sending and getting information to and from the server
- It runs asynchronously so you do not need to wait for it to finish before doing something else

Making AJAX Calls

- You use the `fetch()` function to make AJAX calls in JavaScript
- For example, you can run this code to get some JSON content from an URL:

```
fetch("https://static.data.gov.hk/
        covid-vaccine/summary.json")
```

The JSON content


```
{"firstDoseTotal": 6187826, "firstDosePercent": "91.9%",
"secondDoseTotal": 5650469, "secondDosePercent": "83.9%",
"thirdDoseTotal": 2701887, "latestDaily": 39678, "sevenDayAvg": 59060,
"firstDoseDaily": 2893, "secondDoseDaily": 12535, "thirdDoseDaily": 24250,
"totalDosesAdministered": 14944166, "age3to11FirstDose": 314911,
"age3to11FirstDosePercent": "59.7%", "age3to11SecondDose": 89073,
"age3to11SecondDosePercent": "16.9%"}
```

Using Promises

- Instead of returning the URL content, `fetch()` only returns a promise
- That means you need to write the code using `.then()` and `.catch()`, i.e.:

```
fetch("https://static.data.gov.hk/
        covid-vaccine/summary.json")
    .then((response) => {
        ... Request is successful...
    })
    .catch((error) => {
        ...An error occurs...
    });
```

The Response Object

- When you write the code in `fetch().then()`, you can get the HTTP response as a `Response` object, i.e.:

```
fetch(...URL...)
    .then((response) => {
        ...
    })...
```

The Response object

- This object provides some functions to help you convert the response to different formats

Using the Response Object

- You can use the `Response` object to get its content in plain text form or in JSON form respectively using these two functions:
  - `response.text()`
  - `response.json()`
- Just like `fetch()`, these functions return a promise so you can chain a `.then()` after the `fetch()`

The Complete Code

- Here is the complete code to get some JSON data from a URL and then show it nicely:

```
fetch("https://static.data.gov.hk/
      covid-vaccine/summary.json")
  .then((response) => response.json())
  .then((data) => {
    $("#result").text(
      JSON.stringify(data, null, "  "));
  })
  .catch((error) => {
    $("#result").text(error);
  });
```

Format the JSON

The arrow function returns the promise `response.json()` *to the next* `.then()`

Sending Data to the Server

- The previous example is a simple GET request
- It will be useful if the AJAX call can send some data to the server
  - Although you can do that by putting a query string at the end of the URL, it is commonly done using a POST request
- You can configure the `fetch()` function to do that, as shown on the next slide

Making a POST Request

- Here is the code to make a POST request to send some JSON data to the server

```
fetch("/getuser", {
    method: "POST",
    headers: { "Content-Type":
               "application/json" }
    body: JSON.stringify(data)
})
    ...
```

Use the POST method

The data sent as JSON

Need to use the correct MIME type

Getting JSON Data Using Express

- The Express app does not recognise incoming JSON by itself
- If you want to read JSON data, you will need to set up a JSON ‘middleware’ (= a processor) using this code:

```
app.use(express.json());
```

- This middleware converts any incoming JSON into a JavaScript object automatically

Reading JSON Example

- Here is an example reading the JSON data in the Express app:

You need to use `post()` *here*

```
app.post("/getuser", (req, res) => {
    const data = req.body;

    ...
});
```

This contains the incoming JSON data

- Notice that `req.body` is a **JavaScript object**, rather than the JSON text

Returning Data

- After receiving the request, the Express app can then return some data
- Just like what you did before, you can return some JSON data using this code:

```
app.post("/getuser", (req, res) => {
    ...

    res.json(...some JavaScript data...);
});
```

- Once the browser receives the response, it will complete one cycle of the AJAX call