Introduction to Computer Science I
COMP 2406A – Winter 2020

Client-Server Architecture

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Understand the difference between client and server

Understand the need for resource organization, naming schemes, and protocols

Understand the request/response cycle

Understand some of the benefits of asynchronous processing

The Most Important Thing

The most important thing to realize – we will be developing software for two different entities:

The client – the software the ‘user’ interacts with

The server – ‘our’ software that provides resources

The Most Important Thing

In general, the client starts with nothing (think about Google, YouTube, Instagram, etc.)

All resources the client needs are provided by the server(s) upon request

In general, the user/client make requests for resources and the server sends resources in response to these requests

Client Software

Client software is anything we expect to be used by the browser of the user – HTML, CSS, Javascript

These resources, though, are still STORED on the server

When a user requests a resource (i.e., todo.html), the server sends that resource to them

Server Software

Server software is used to accept requests, process those requests, and provide the necessary resources

The resources the server provides are typically:

1. **The software the client requires**
2. **The data the client requires**

Resources can be HTML, CSS, Javascript, JSON or other text data, images, videos, etc.

Request/Response Model

Since the server contains all information the client needs, many requests and responses are made

There is an initial request for some resource (i.e., HTML page), which may trigger additional requests (i.e., Javascript, images, CSS, JSON)

Additionally, user action and/or client-side Javascript may initiate even more requests

Request/Response Model

A 'resource' in this case, refers to something on the server that the client wants to interact with.

We use URLs to specify WHAT resources, so the server knows what processing/response in required

Protocols and APIs

In order to facilitate all these requests/responses, there needs to be organization of data

We define communication protocols, naming schemes, APIs to formalize this (you have been doing this in other programming too)

The user needs to be able to specify what they want and the server needs to be able to understand their request

Examples From Real World

To try and make this clear, we will look at some offline systems and their web-based counterparts

We will see that the web-based systems are just implementations of the offline systems

The protocol is more formal, as computers require formality

Examples From Real World

We will look at a bank, library, and trivia game

In all of these offline systems, ‘you’ will be the client

In the online versions, we just have client software to support all the operations ‘you’ would have performed

The Bank

If you go to a bank, what happens?

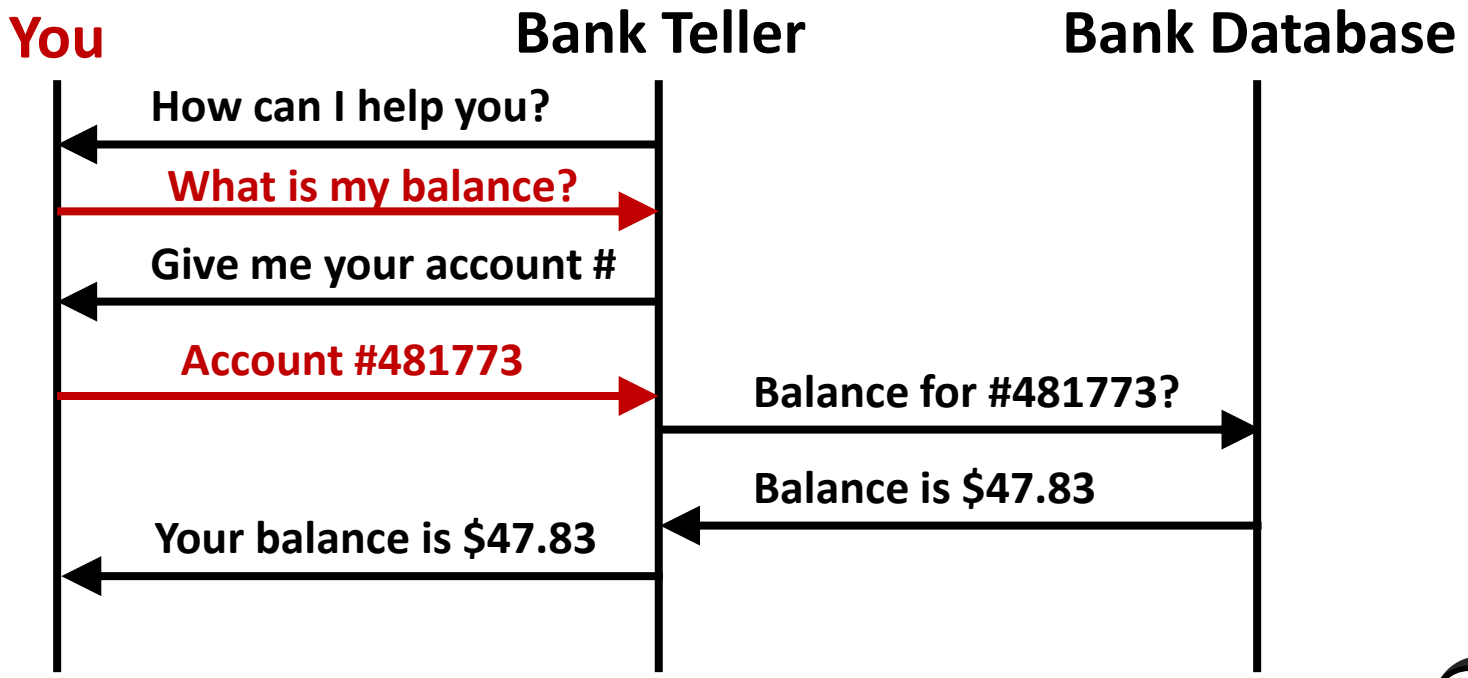
You need to specify what operations you want performed (check balance, deposit, withdraw, etc.)

The bank employee will need to get specific information from you (account numbers, amounts)

These transactions can be captured by interaction diagrams…

The Bank

In an interaction diagram, time is on the Y axis. We can show messages, requests, data, or other interactions with arrows between entities.



The Bank

A similar interaction happens if you use an ATM

1. **The machine asks for your card and PIN**
2. **The machine gives you a list of options (deposit, withdraw, balance check)**
3. **You pick an option (i.e., withdraw)**
4. **The machine asks you for information (the amount to withdraw)**
5. **The machine validates your request**
6. **The machine gives you money**

The Bank

In these cases, you are the client – you are specifying requests to be handled

The bank teller or ATM was acting as the server – they accept your requests, possibly take some actions on their underlying data, and give you responses

We can implement this same system on the web…

The Bank

Resources the bank may provide could be:

/ - the home page

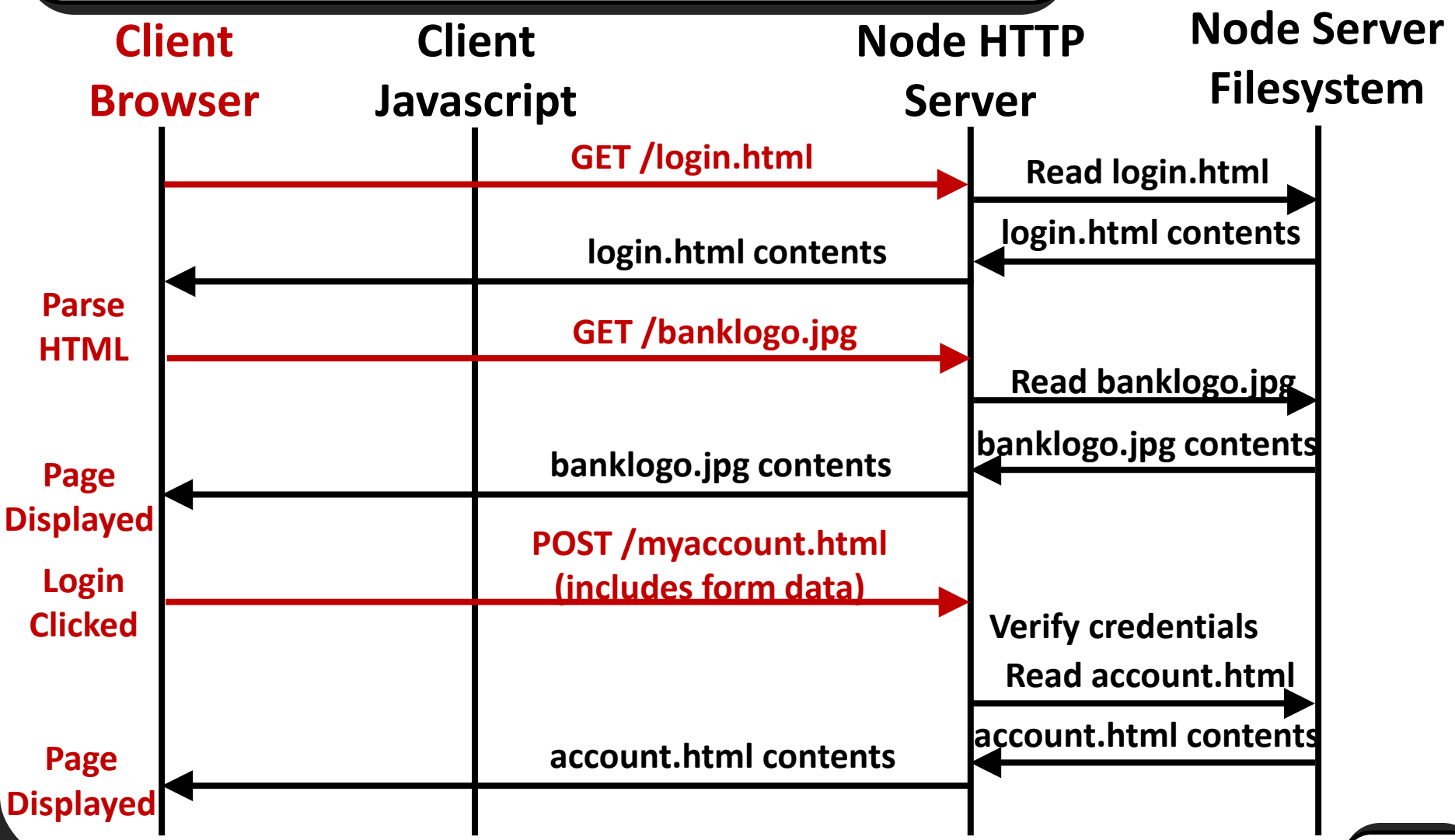
/login – the login page for clients

/accountinfo – a client's account information

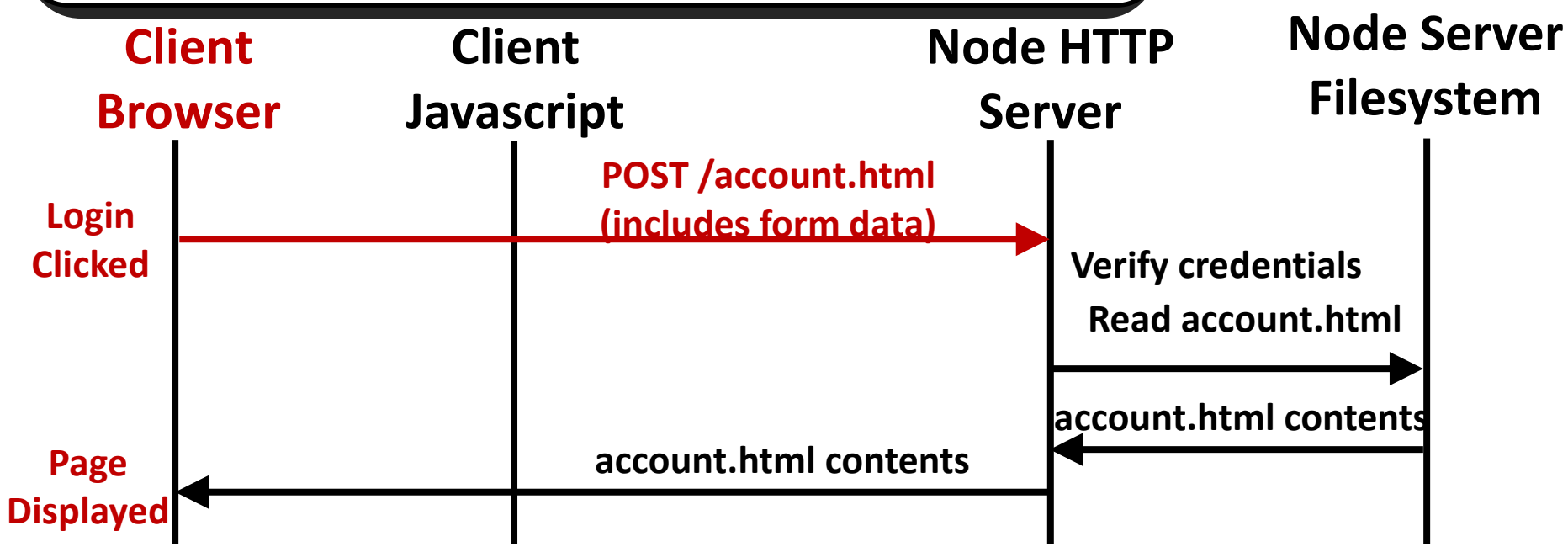
/deposit – depositing money to an account

/withdraw – withdrawing money from an account

The Bank Web App



The Bank Web App



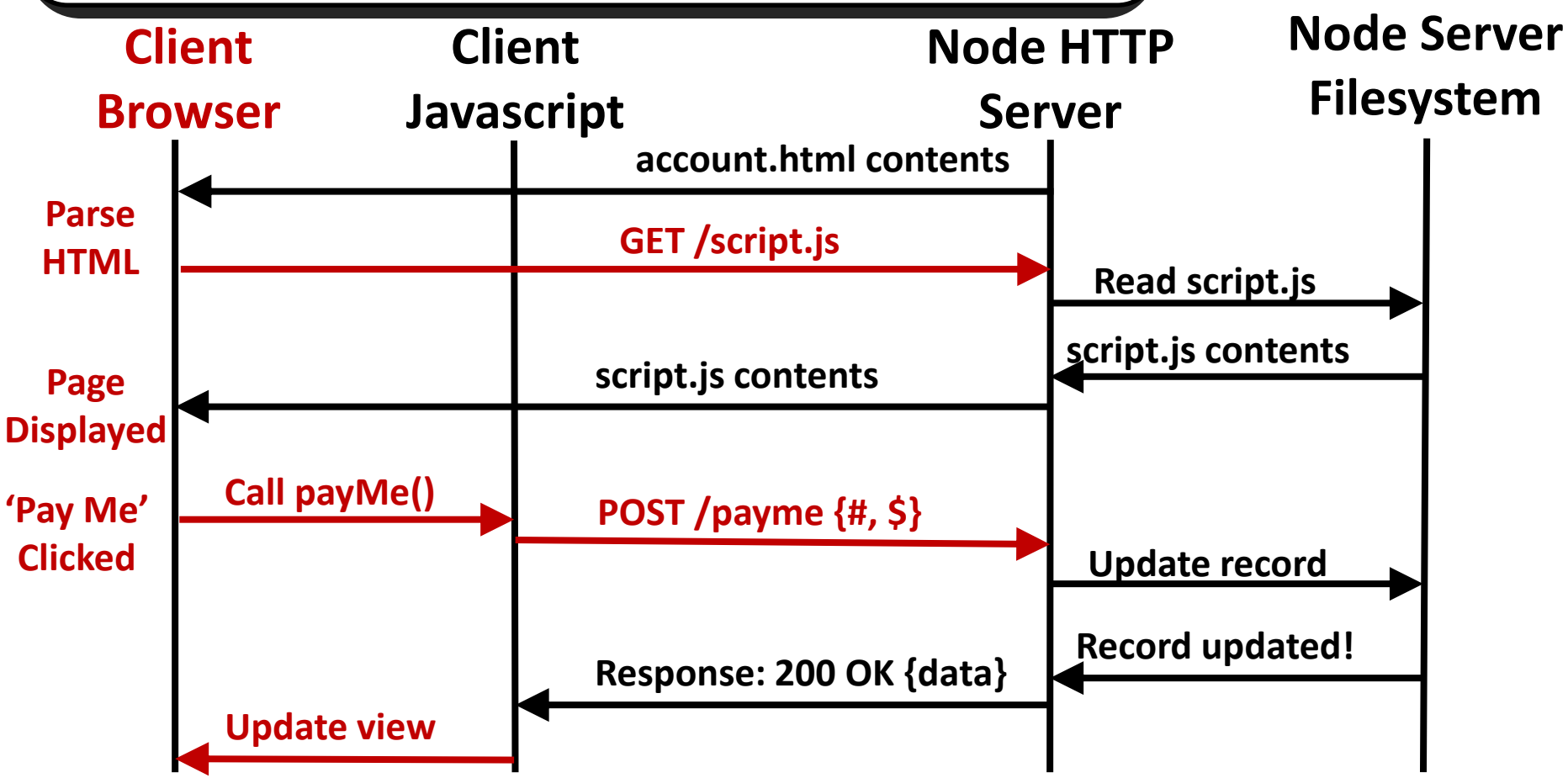
Note that "Read account.html" is just a way of loading the contents that will be sent as a response.
We could use a function, a template, a string, another HTTP request, etc., to generate this.

Note that we may also have some requests/responses happening asynchronously (i.e., in the background)

For example, if the account page loads a Javascript file, the user could trigger requests by interacting with the page

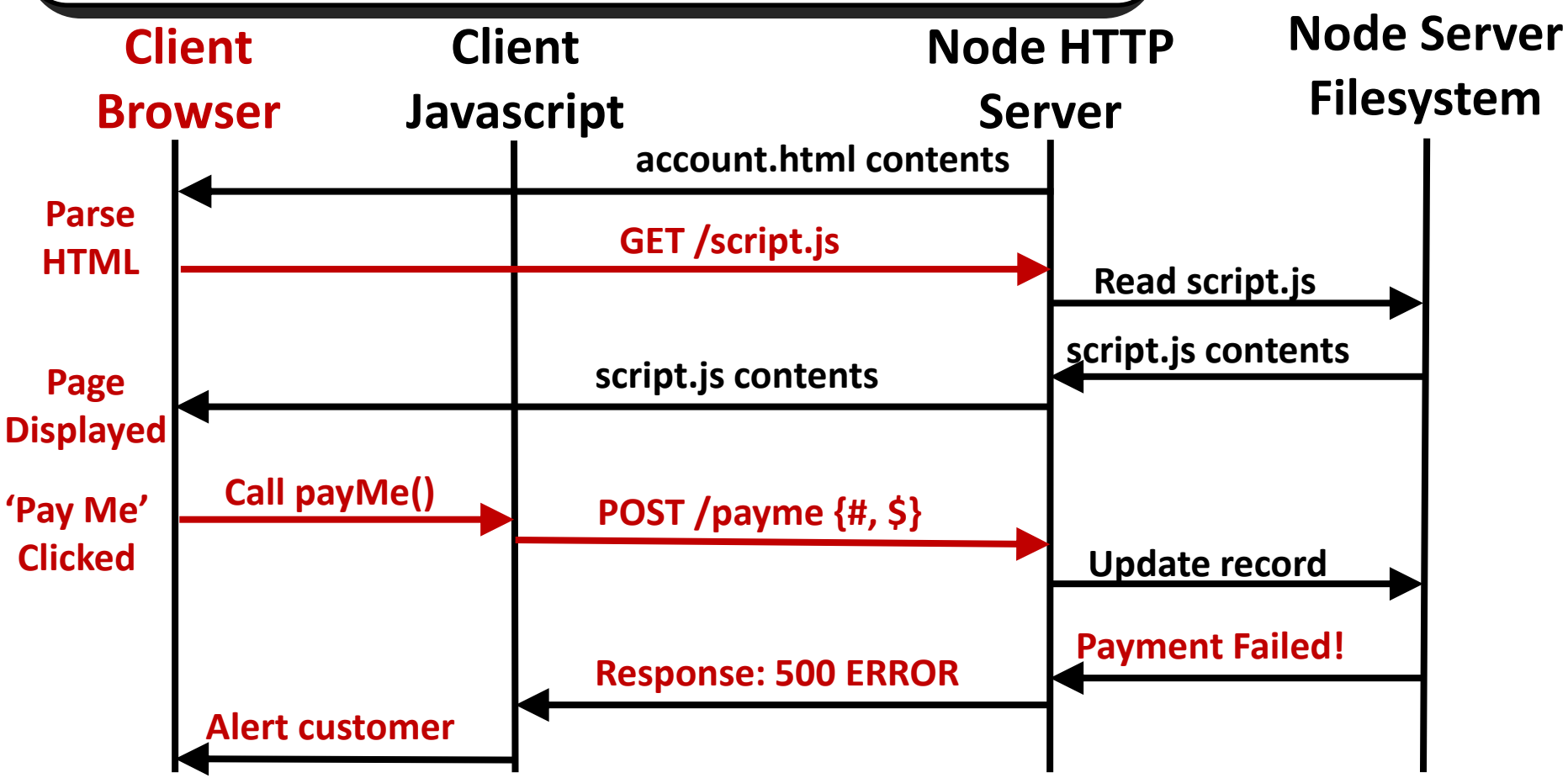
The same client Javascript can handle the responses to these requests and update the page state

The Bank Web App



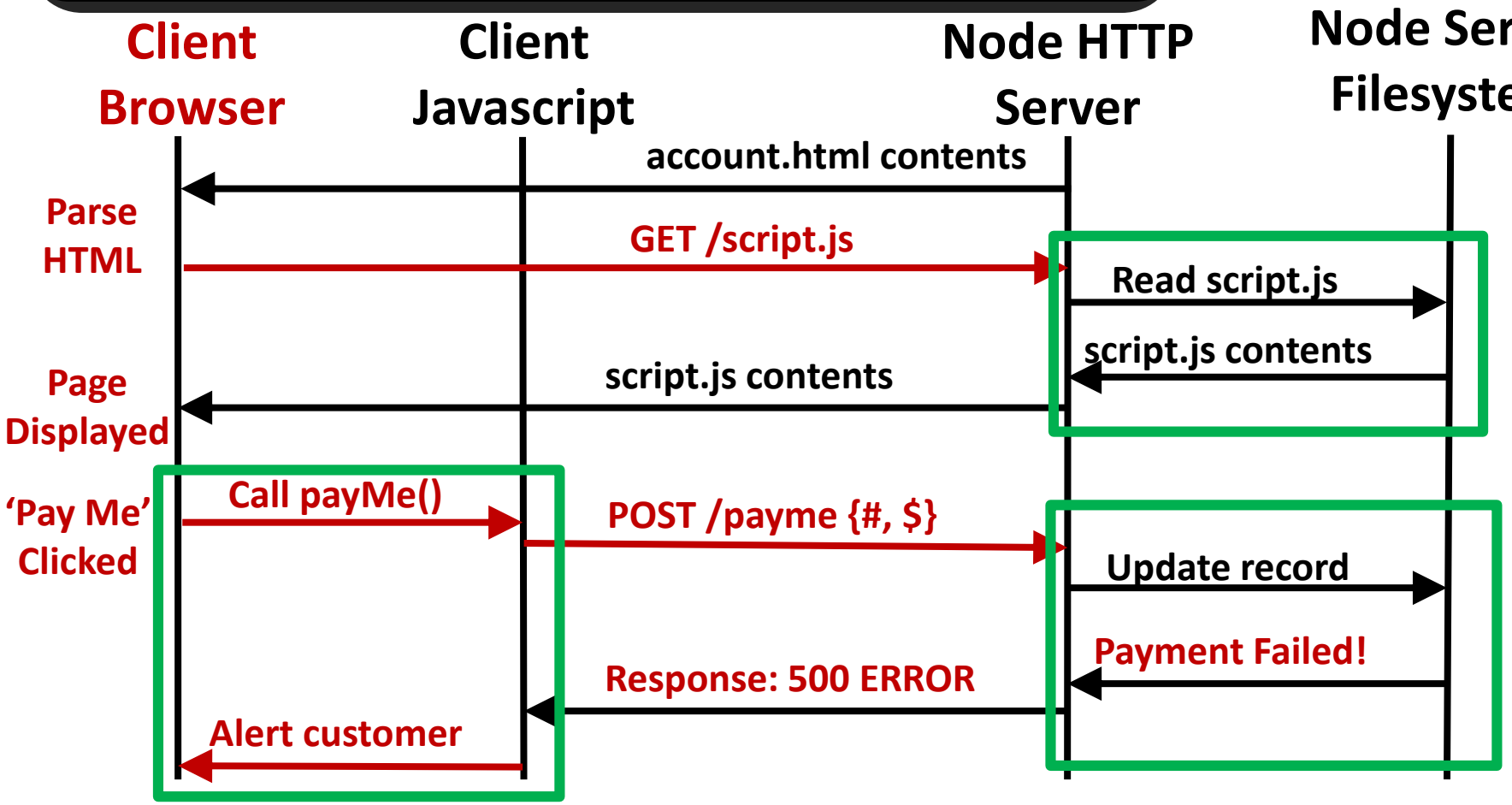
This is just one case where everything went well

The Bank Web App



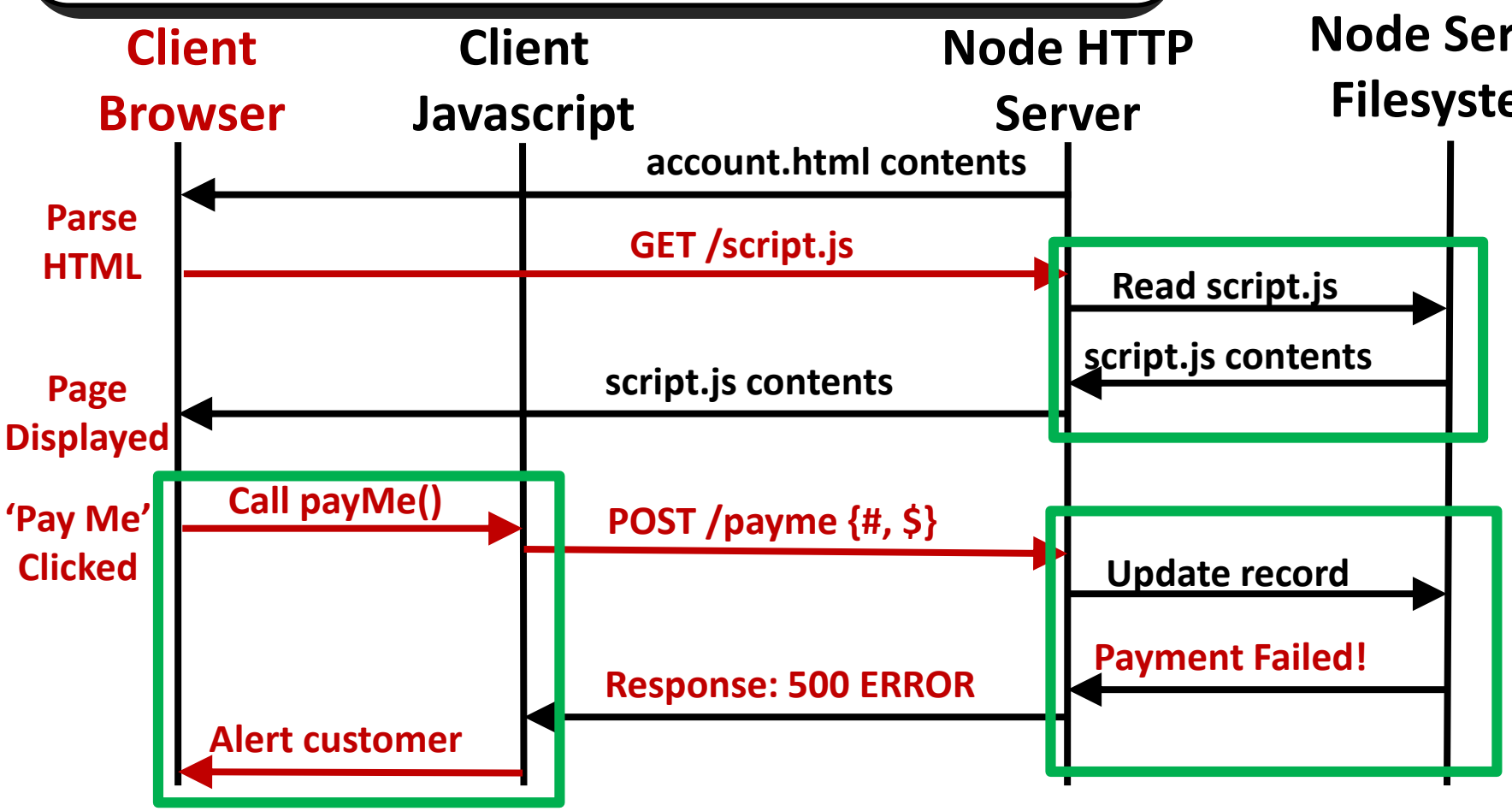
There could also be errors…

The Bank Web App



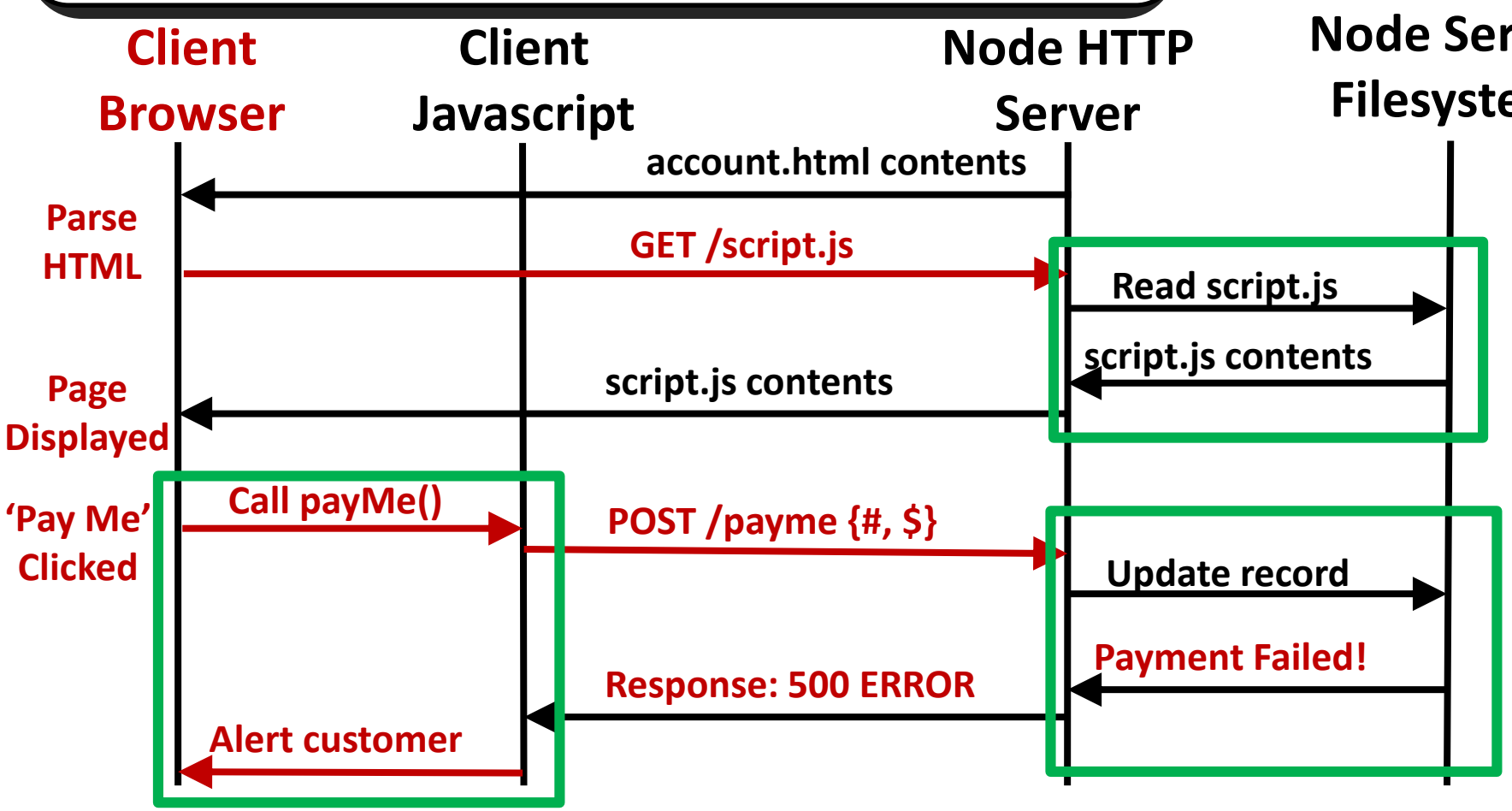
Many operations are asynchronous on the client or server

The Bank Web App



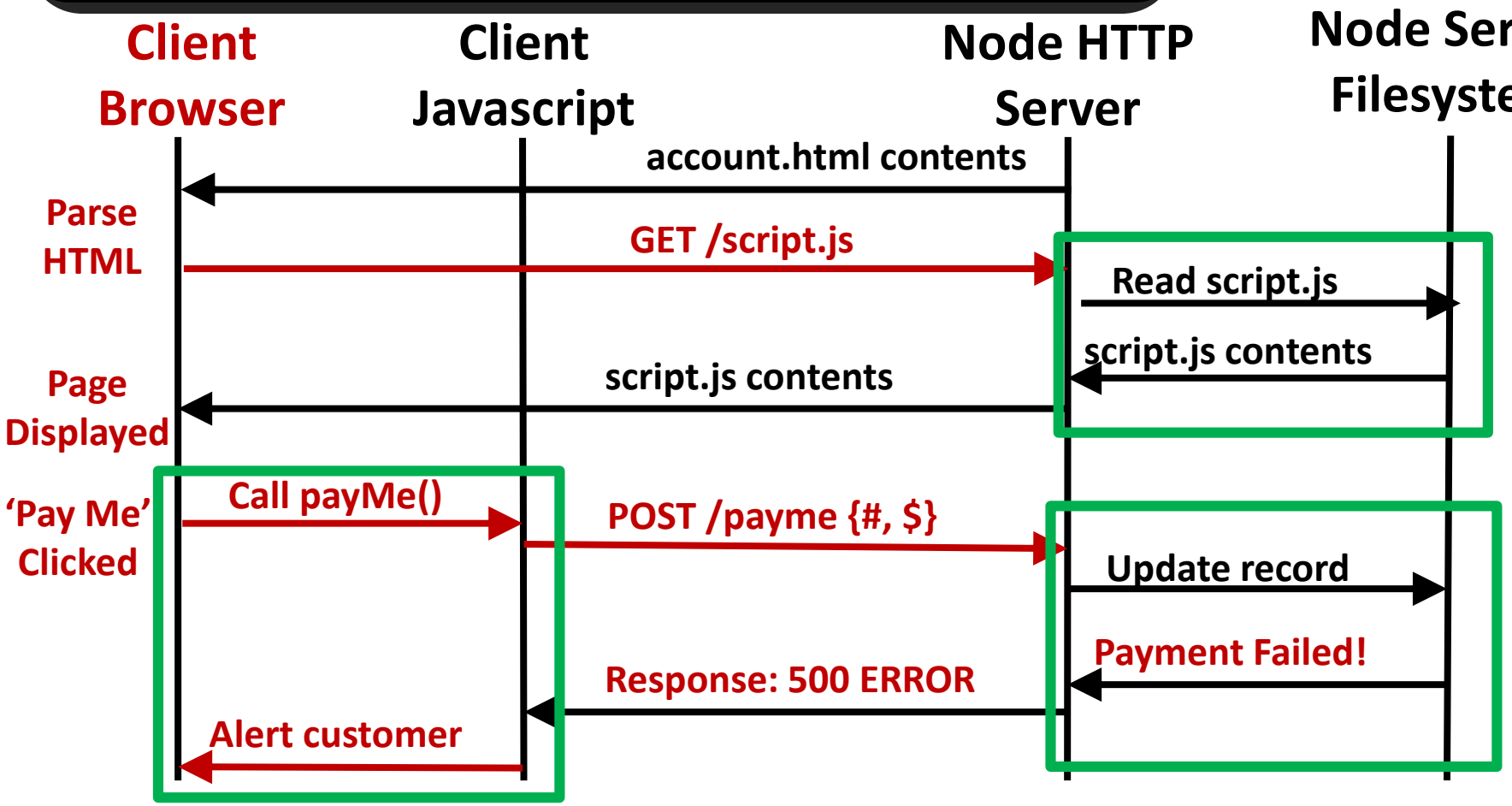
This allows other processing to be done while they wait

The Bank Web App



The client can continue to read/interact with the page

The Bank Web App



The server can continue to process requests, send responses

The Hard Part

The important part is deciding:

What resources are necessary

What the naming scheme is

What the data requirements are

How to handle requests and responses

This process is similar to non-web programming

The Library

Another real-world system we can consider is the library

A library has many books, some of which may have content you are interested in

The library provides a way for you to find the content you want…

The Dewey Decimal System

Books in the library are divided into categories, sub-categories, etc.

For example[1]:

500 Natural sciences and mathematics
510 Mathematics
516 Geometry
516.3 Analytic geometries
516.37 Metric differential geometries
516.375 Finsler geometry

1. https://en.wikipedia.org/wiki/Dewey_Decimal_Classification#Design

The Library

If you go into a library, you can look at progressively more specific categories that match what you are looking for:

Technology → Agriculture and Animal Husbandry → Domestic Animals → Cats

As with many things, there are multiple ways we can implement a web-based library

1. https://en.wikipedia.org/wiki/Dewey_Decimal_Classification#Design

The Library

We can build a single-page web app

The user requests the route “/”

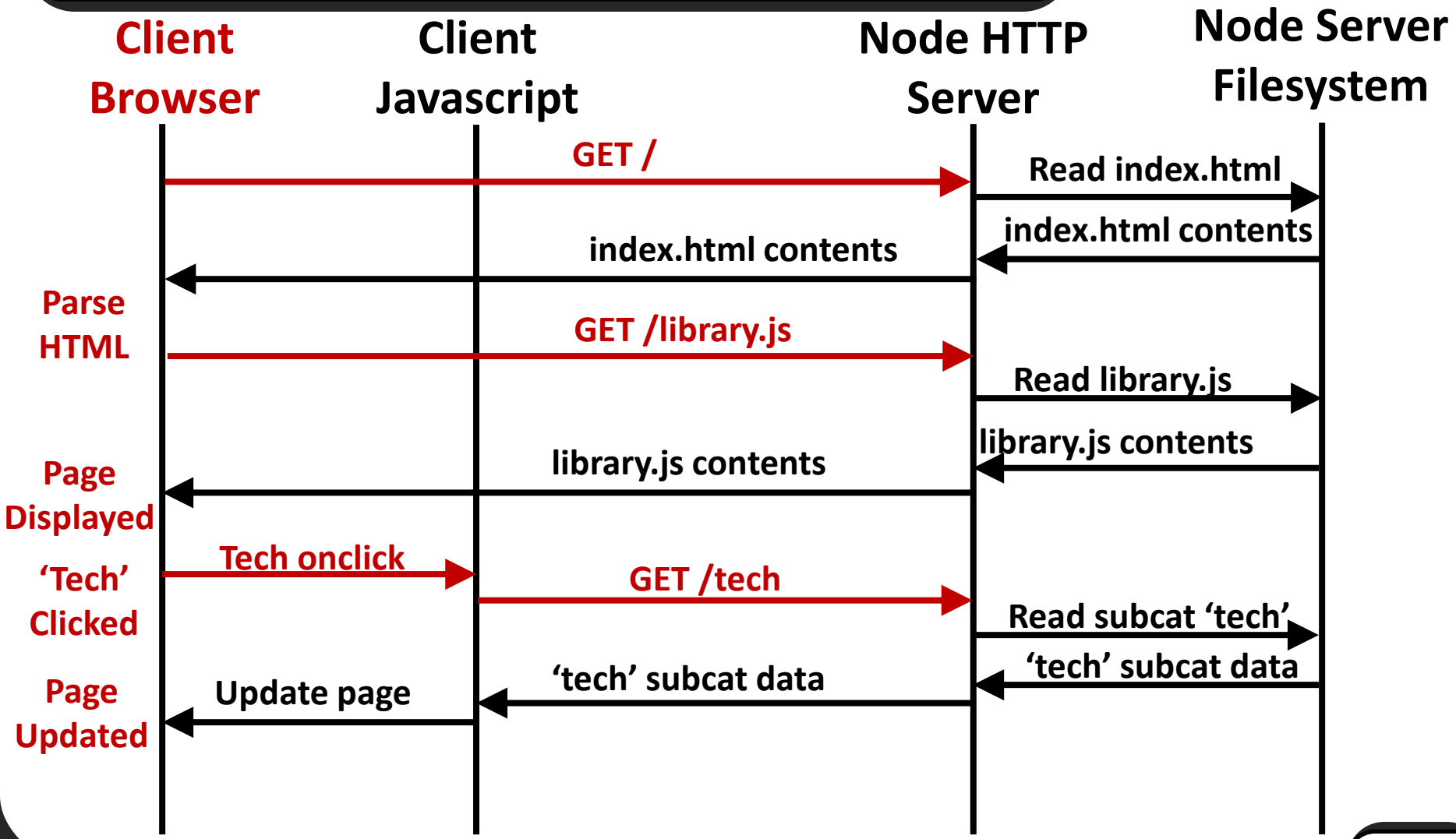
We send HTML with the most general categories and a Javascript file

The Library

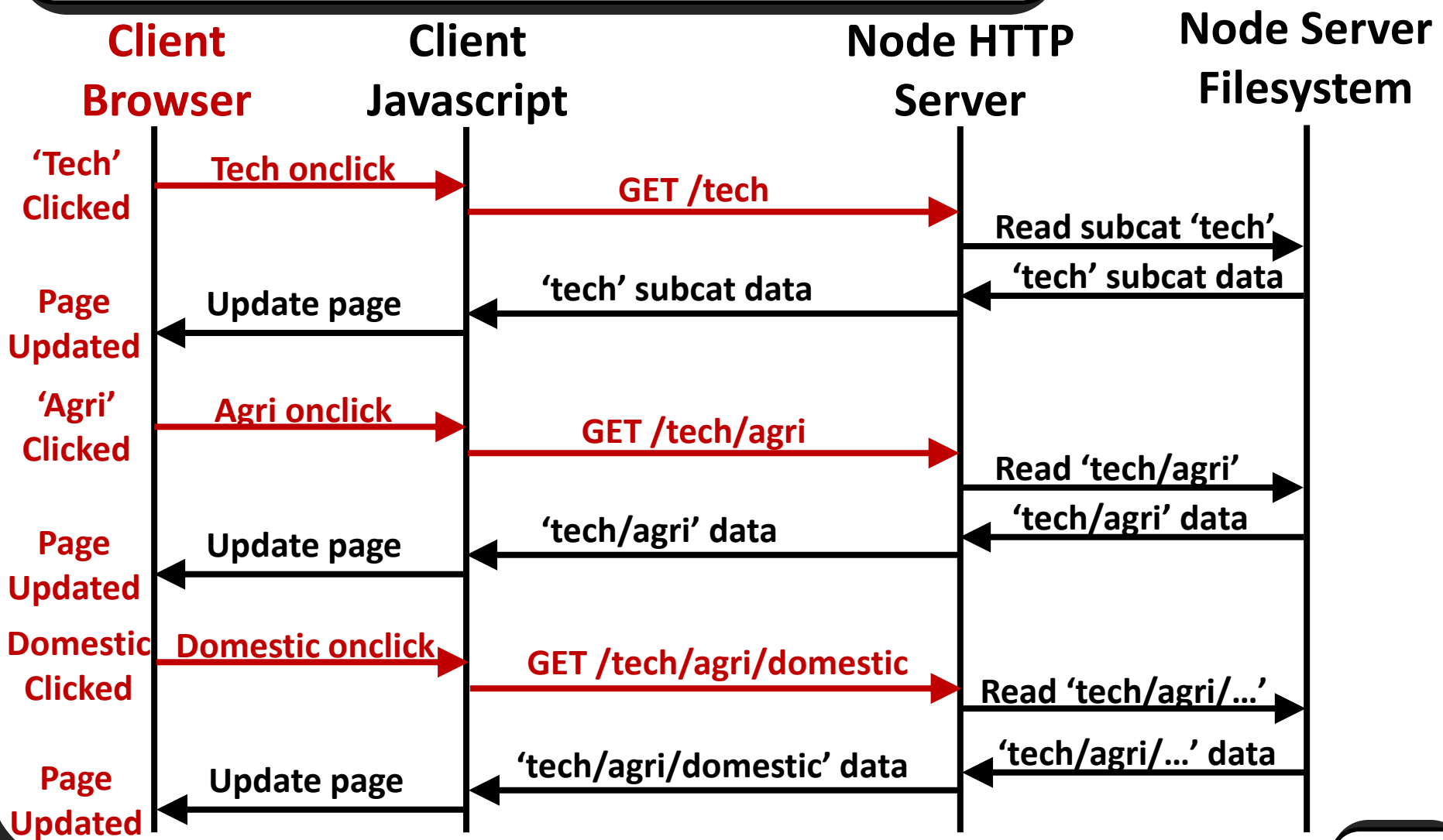
User can then select a category, triggering the client Javascript to make a request for new information from the server

The server can respond with the list of sub-categories and the client Javascript can update the page

The Bank Web App



The Bank Web App



Again, there is some abstraction here

How we ‘read’ the data for ‘tech/agriculture/domestic’ can vary

We can use a file system, database, nested objects, etc. – this is the concern of the server, not the client

The client is only interested in the response

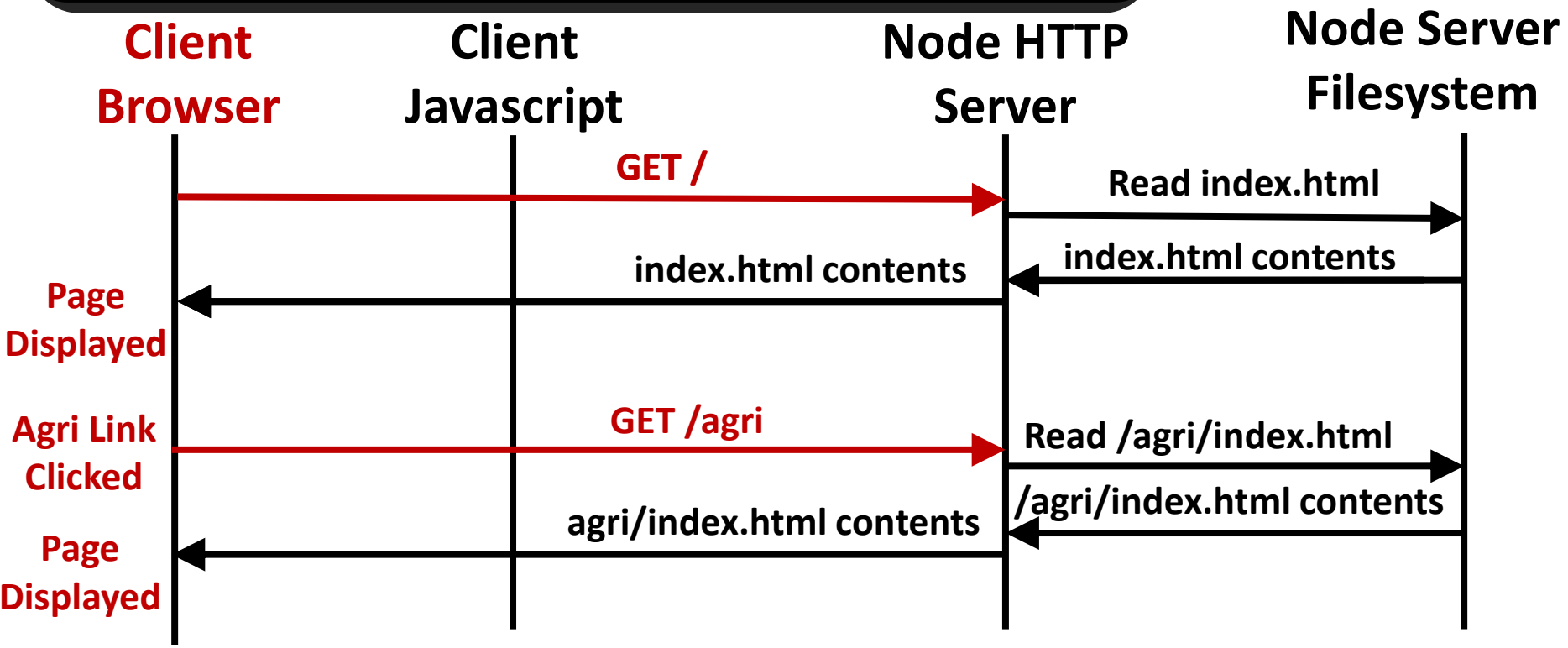
The Library

An alternative design is to use links within the HTML to navigate between pages

The server can process the URLs in the requests to determine the data from the overall hierarchy to be sent in response

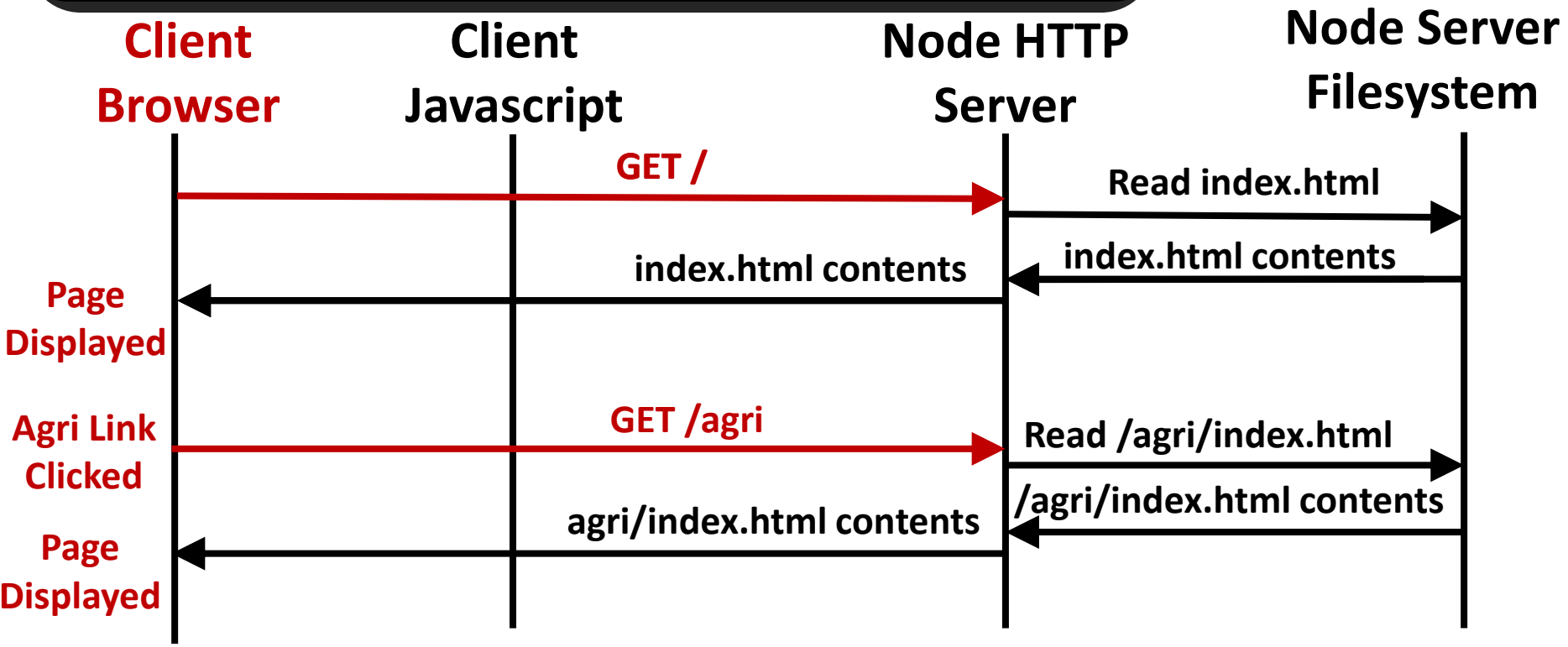
For example: /index.html, /tech/index.html, /tech/agri/index.html, /tech/agri/domestic/index.html

The Bank Web App



The main difference here is that a new page is loaded for each request, instead of having a partial page update. With GET requests, this allows bookmarking, history, etc.

The Bank Web App



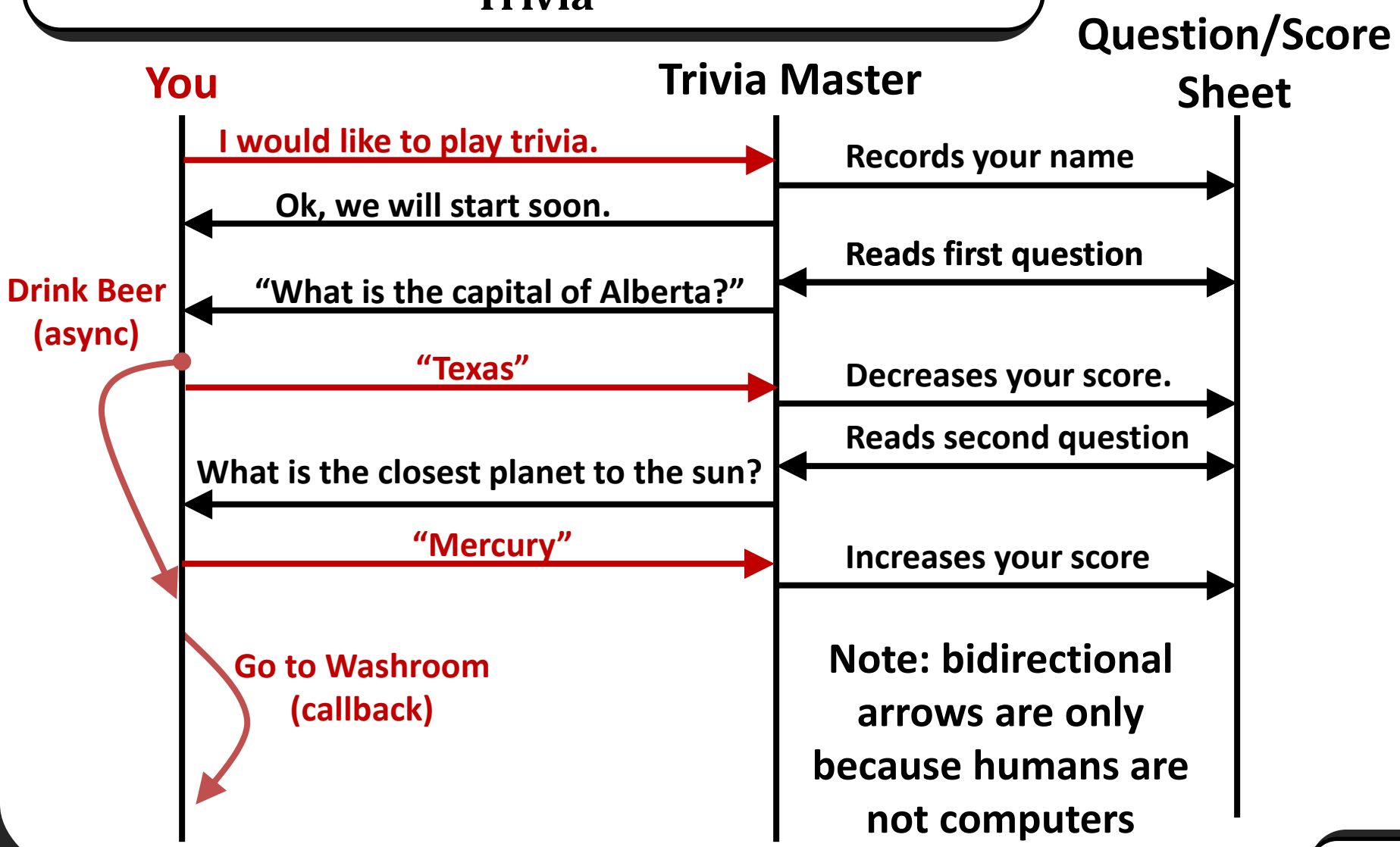
Again, it doesn't have to be read from a file. The path /agri/index.html just represents a resource on the server (we'll see a lot of this once we talk about Express and REST)

Trivia

Without trivia web apps, people are forced to go outside to pubs to play trivia

The process, however, is still very similar

Trivia


You
Trivia Master
Question/Score Sheet
I would like to play trivia.
Records your name
Ok, we will start soon.
Reads first question
Drink Beer (async)
“What is the capital of Alberta?”
“Texas”
Decreases your score.
Reads second question
What is the closest planet to the sun?
“Mercury”
Increases your score
Go to Washroom (callback)
Note: bidirectional arrows are only because humans are not computers

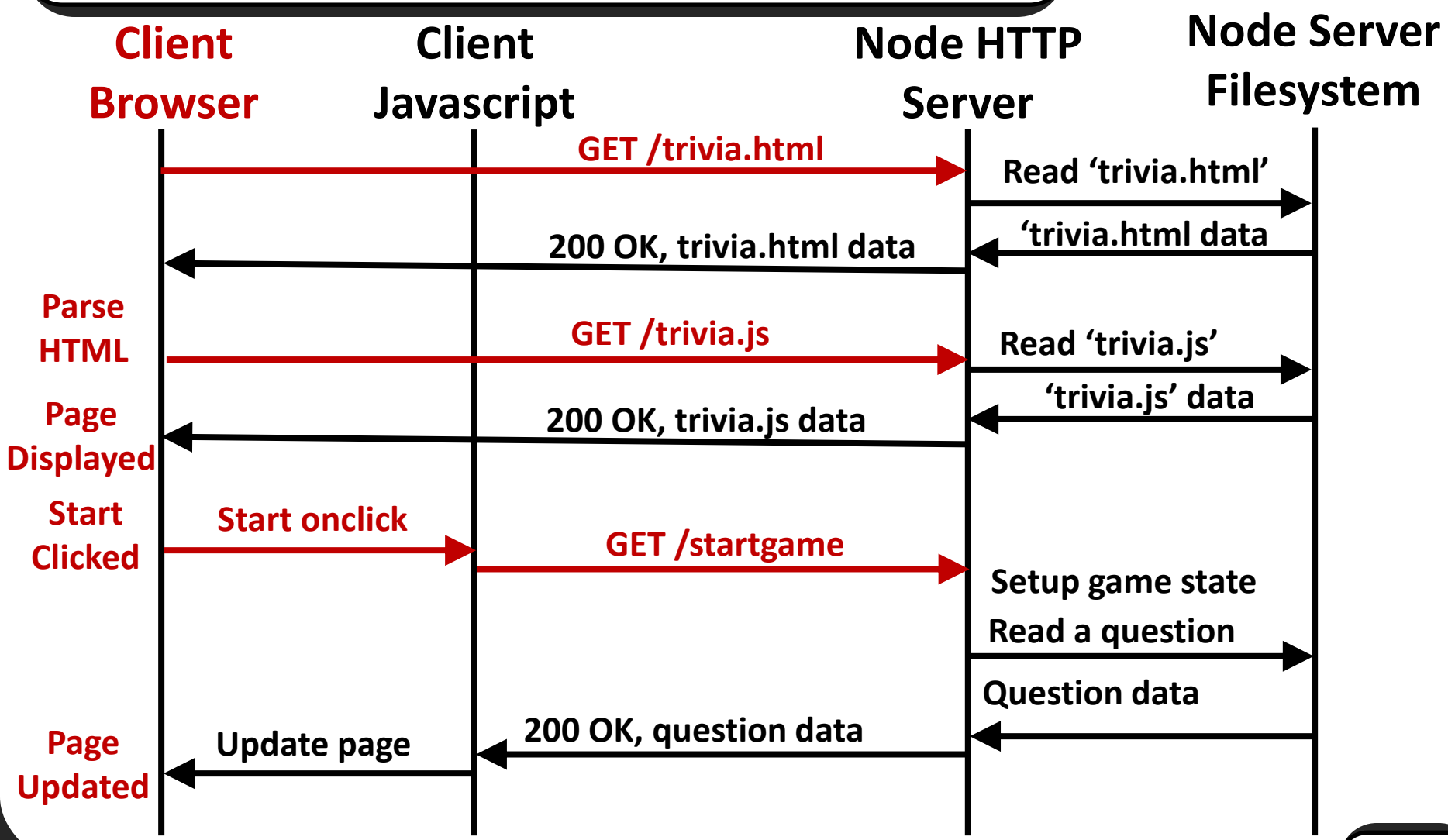
Trivia Web App

A trivia web app works similarly

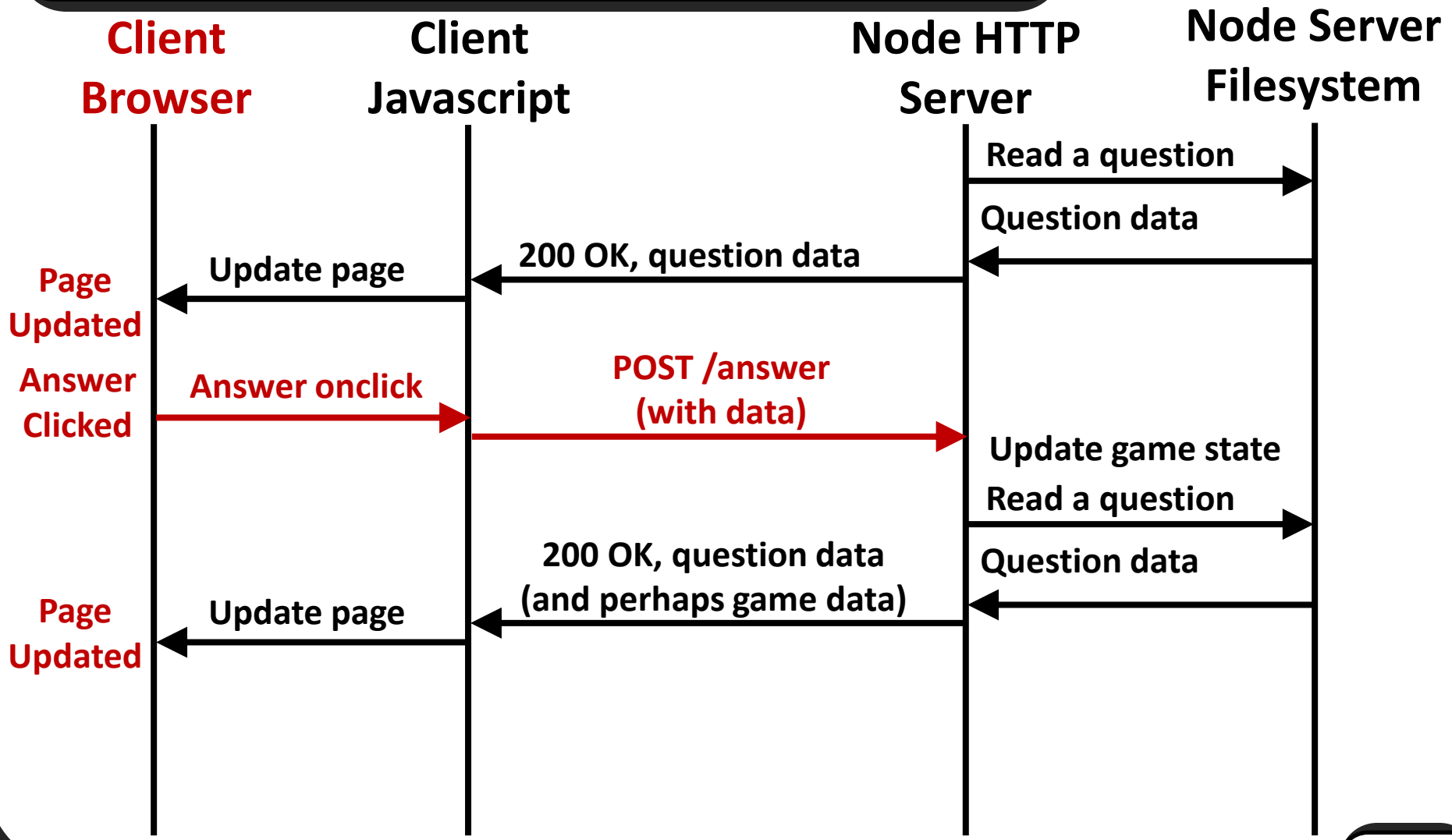
You need a way to request:

Joining, questions, answer submission, score updates, game progress updates, etc.

The Trivia Web App



The Trivia Web App



Trivia Web App

The server can store information about many games

Give the games unique IDs

Have the client Javascript include the game ID when making requests

Again all you need is a way to organize the data on the server and a way for clients to specify requests

Important Things to Understand

HTTP Protocol – GET, POST, DELETE, PUT

Server/Client architecture

Asynchronous programming and callback functions

Questions?

So are there any questions about client/server interactions, architecture, asynchronous functions, etc.?