NODE JS

Revathi G P
Department of
Computer Science and Engineering

NODE JS

NodeJS Introduction

Revathi G P
Department of Computer Science and Engineering

NodeJS Introduction

- Node.js is an open source, cross-platform runtime environment built on Google Chrome's JavaScript Engine (V8 Engine).
- Node.js was developed by Ryan Dahl in 2009 and its latest version is v20.17.0
- Node.js is a platform built on chrome’s JavaScript runtime for easily building fast and scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- Node.js is open source, completely free, and used by thousands of developers around the world.

NodeJS Introduction

- Node.js applications are written in JavaScript, and can be run within the Node.js runtime on mac OS X, Microsoft Windows, and Linux.
- Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.
- **Node.js = Runtime Environment + JavaScript Library**

Why Node?

When you create websites with PHP for example, you associate the language with an HTTP web server such as Apache or Nginx. Each of them has its own role in the process:

- Apache manages HTTP requests to connect to the server. Its role is more or less to manage the in/out traffic.
- PHP runs the .php file code and sends the result to Apache, which then sends it to the visitor.

As several visitors can request a page from the server at the same time, Apache is responsible for spreading them out and running different *threads* at the same time.

Each thread uses a different processor on the server (or a processor core)

Why Node?

- A Node.js app runs in a single process, without creating a new thread for every request.
- Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.
- When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

NODE JS

Why Node?

PES UNIVERSITY

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

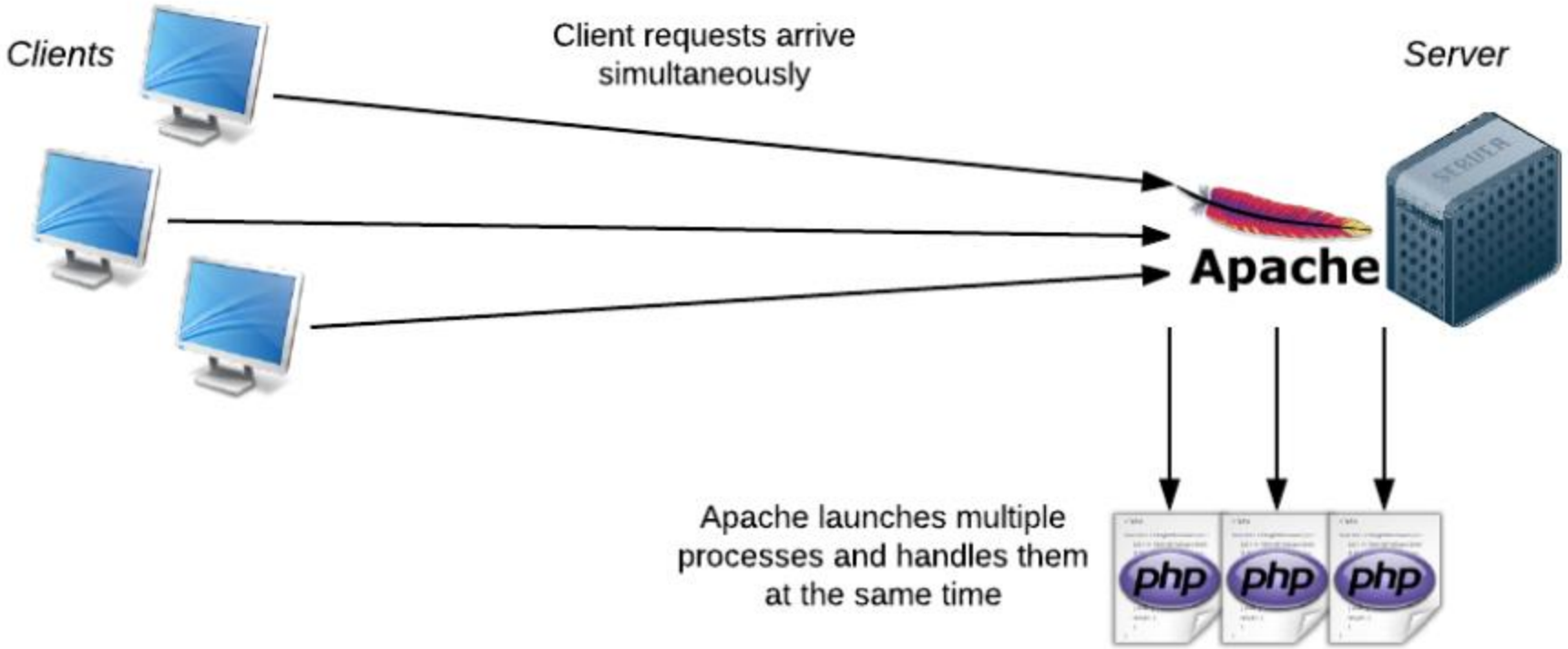
Here is how Node.js handles a file request:

- Sends the task to the computer's file system.
- Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

NODE JS

Why Node?



The Apache server is multithread

NODE JS

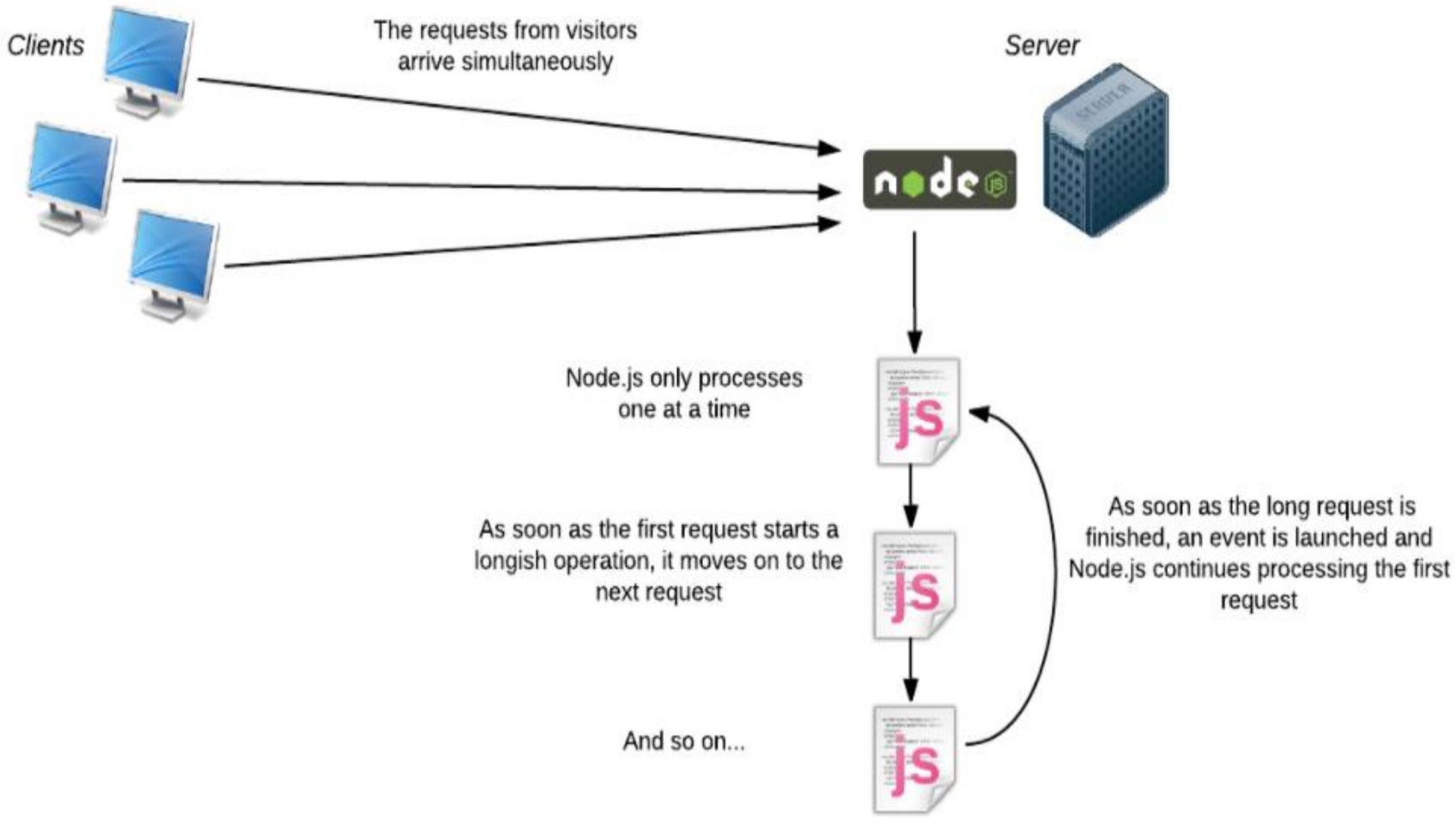
Mono-Thread – Handling multiple Requests?

- Node.js doesn't use an HTTP server like Apache. In fact, it's up to us to create the server! Isn't that great?
- Unlike Apache, Node.js is **monothread**. This means that there is only one process and one version of the program that can be used at any one time in its memory.

But I thought that Node.js was really fast because it could manage loads of requests simultaneously. If it's monothread, can it only perform one action at a time?

NODE JS

Mono-Thread Style



NODE JS

V8 Engine and C++ Bindings

PES UNIVERSITY

- Chrome needs to run some Javascript for a particular Web page, it doesn't run the JavaScript itself. It uses the V8 engine to get that done so it passes the code to V8 and it gets the results back. Same case with Node also to run a Javascript code.
- V8 is written in C++ . Chrome and Node are largely written in C++ because they both provide bindings when they're instantiating the V8 engine.
- This facilitates to create their own JavaScript runtime with interesting and novel features.

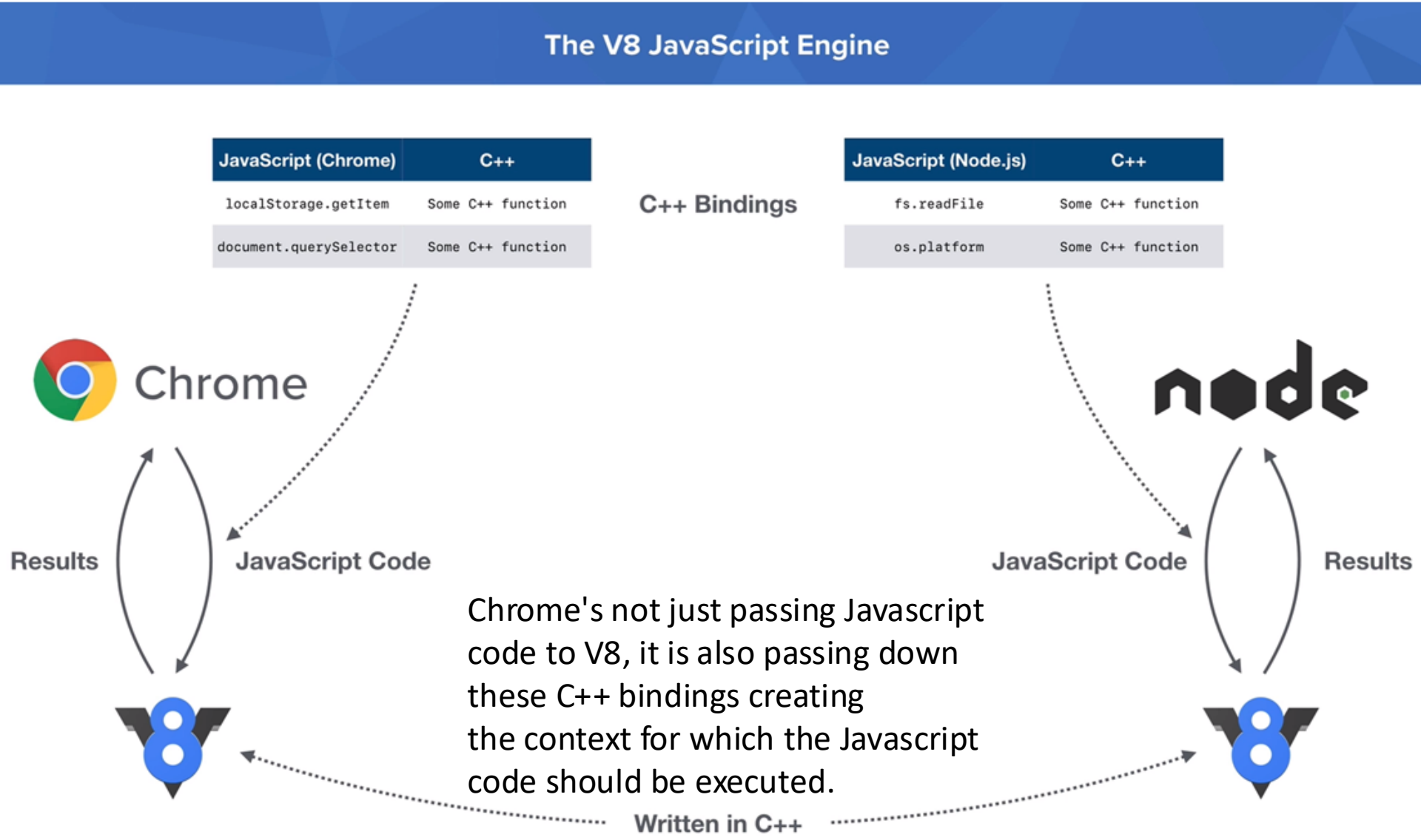
Example Instance:

Chrome to interact with the DOM when the DOM isn't part of JavaScript.

Node to interact with file system when the file system isn't part of JavaScript

NODE JS

V8 JavaScript Engine



Chrome's not just passing Javascript code to V8, it is also passing down these C++ bindings creating the context for which the Javascript code should be executed.

PES UNIVERSITY

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

Features of Node.js

- Asynchronous and Event Driven
- Non Blocking I/O
- Very Fast
- Single Threaded but Highly Scalable
- No Buffering
- License

NODE JS

Features of Node.js

Very Fast

Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

Single Threaded but Highly Scalable

Uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers.

No Buffering

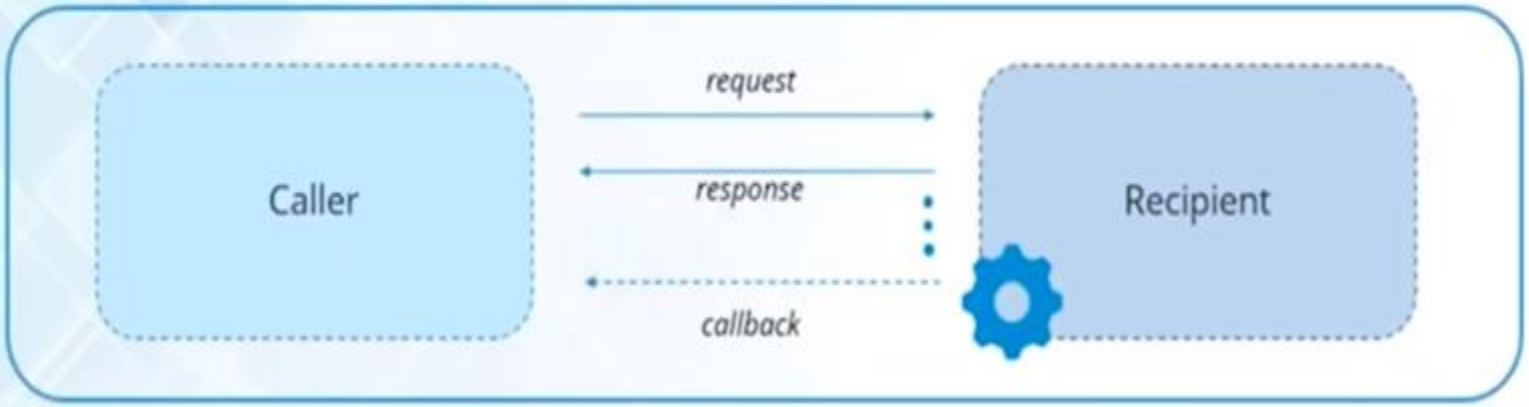
Node.js applications never buffer any data. These applications simply output the data in chunks.

NODE JS

Asynchronous and Event Driven

PES UNIVERSITY

Asynchronous and Event Driven :



- When request is made to server, instead of waiting for the request to complete, server continues to process other requests
- When request processing completes, the response is sent to caller using callback mechanism

NODE JS

Callback Functions

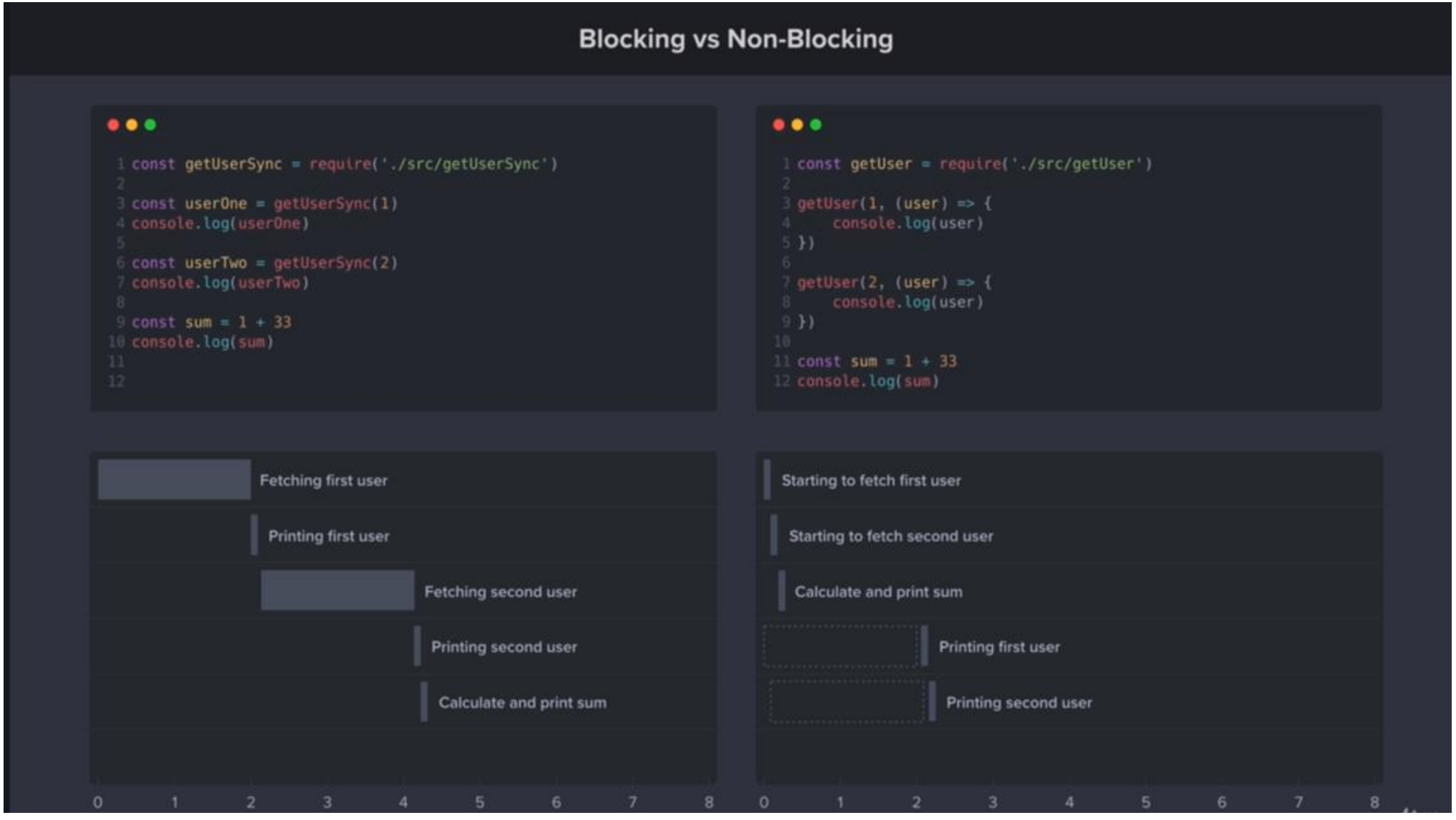
PES UNIVERSITY

- Callback is an asynchronous equivalent for a function.
- A callback function is called at the completion of a given task. Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

NODE JS

Non- Blocking I/O



NODE JS

Blocking and Non- Blocking I/O

PES UNIVERSITY

BLOCKING I/O

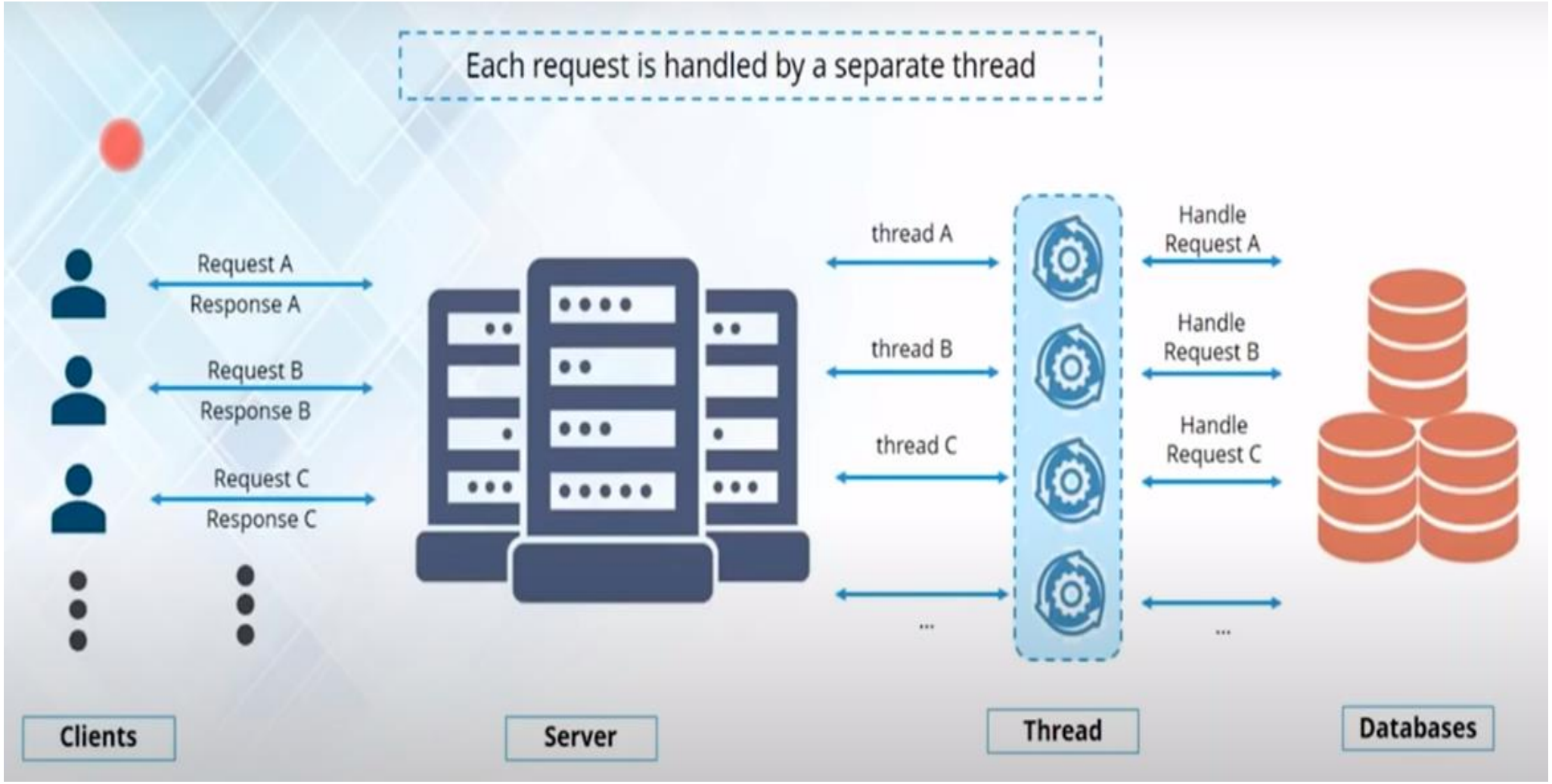
```
var fs = require('fs');
var data = fs.readFileSync('test.txt');
console.log(data.toString());
console.log('End here');
```

```
var fs = require('fs');
fs.readFile('test.txt',function(err,data){
    if(err)
    {
        console.log(err);
    }
    setTimeout(()=>{
        console.log("PES University. Display after 2 seconds")
    },2000);
});
console.log('start here');
```

NON-BLOCKING I/O

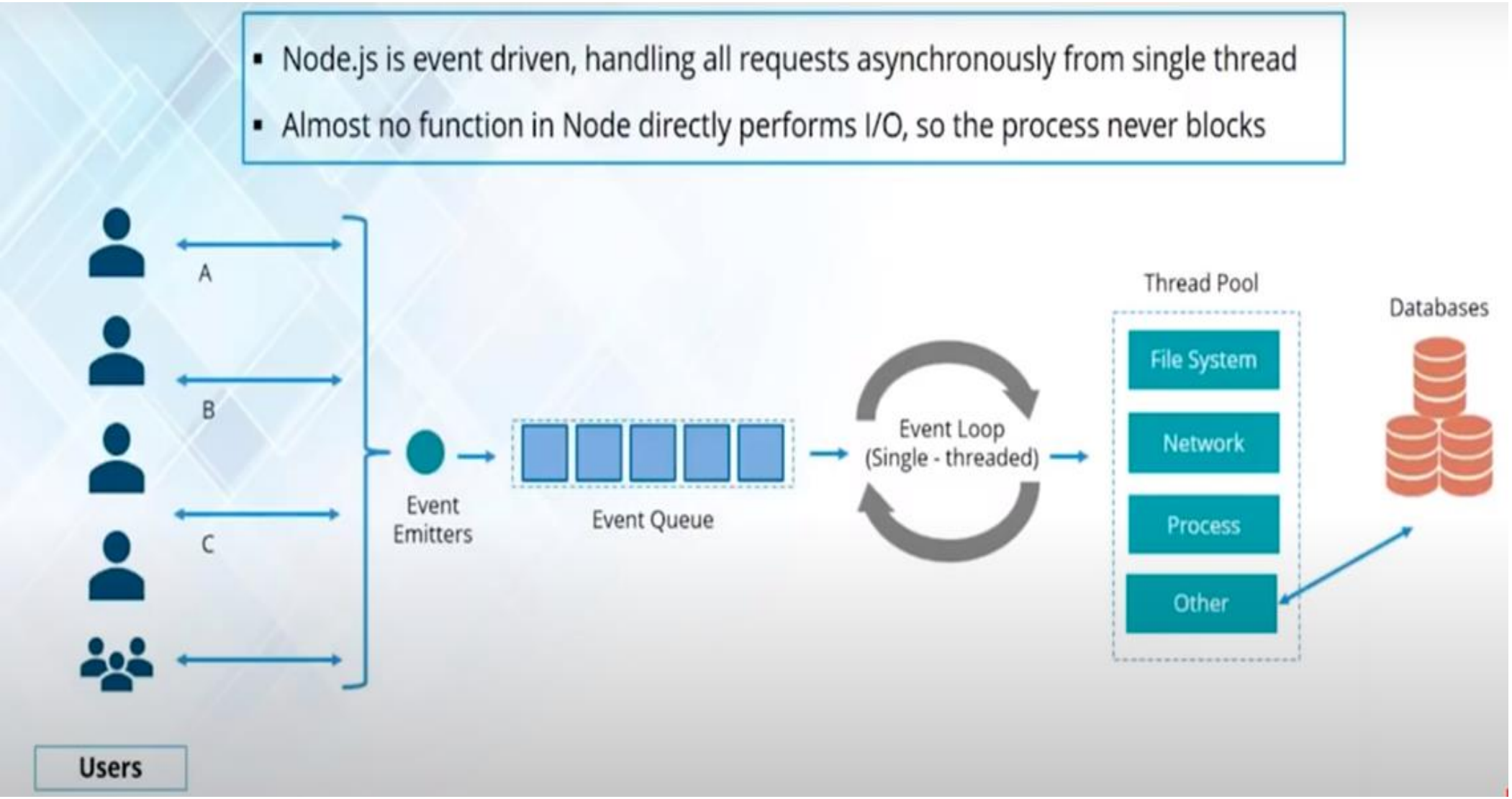
NODE JS

Multi Threaded Model



NODE JS

Single Threaded Model



NODE JS

Multi Threaded vs Asynchronous Event Driven Model

PES UNIVERSITY

Multi-Threaded	Asynchronous Event-driven
Lock application / request with listener-workers threads	Only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
Multithreaded server might block the request which might involve multiple events	Manually saves state and then goes on to process the next event
Using context switching	No contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

Applications of Node.js

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications
- Not for CPU intensive applications.

Link: https://www.youtube.com/watch?v=8aGhZQkoFbQ

NODE JS

Node JS Success Stories

UBER Story

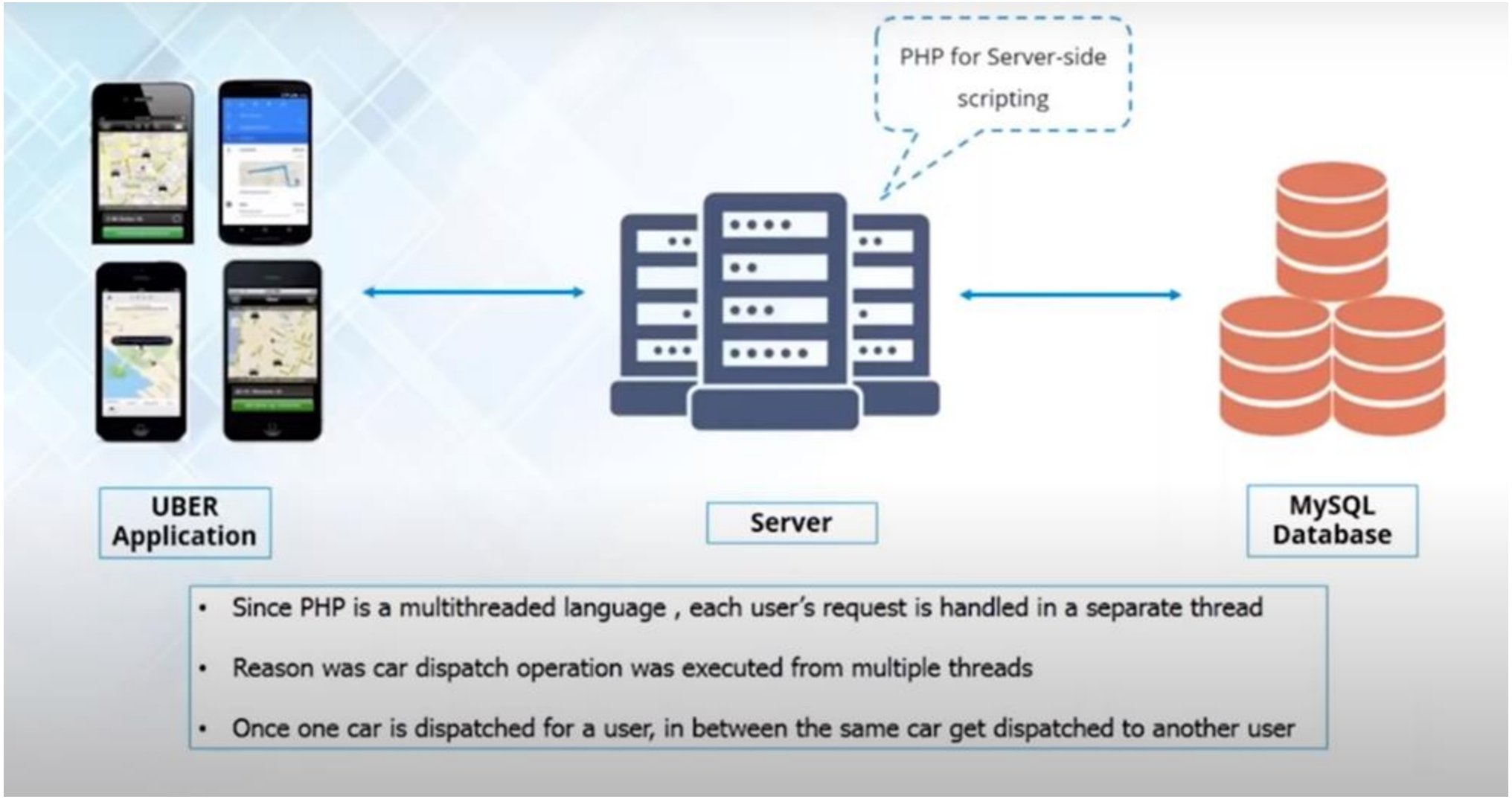
- Customers interact with Uber applications and generating request to book a cab.
- Requests are sent to the Uber server to check in the geospatial databases for the cab.
- Server send back the card details or driver details back to the customer in form of response.

Disadvantage of Multi Threaded model in this case:

- Every request is assigned a new thread from thread pool and it gets exhausted. Scalability is very poor
- Whenever a thread is working on a shared resource. It acquires a lock on that resource.

NODE JS

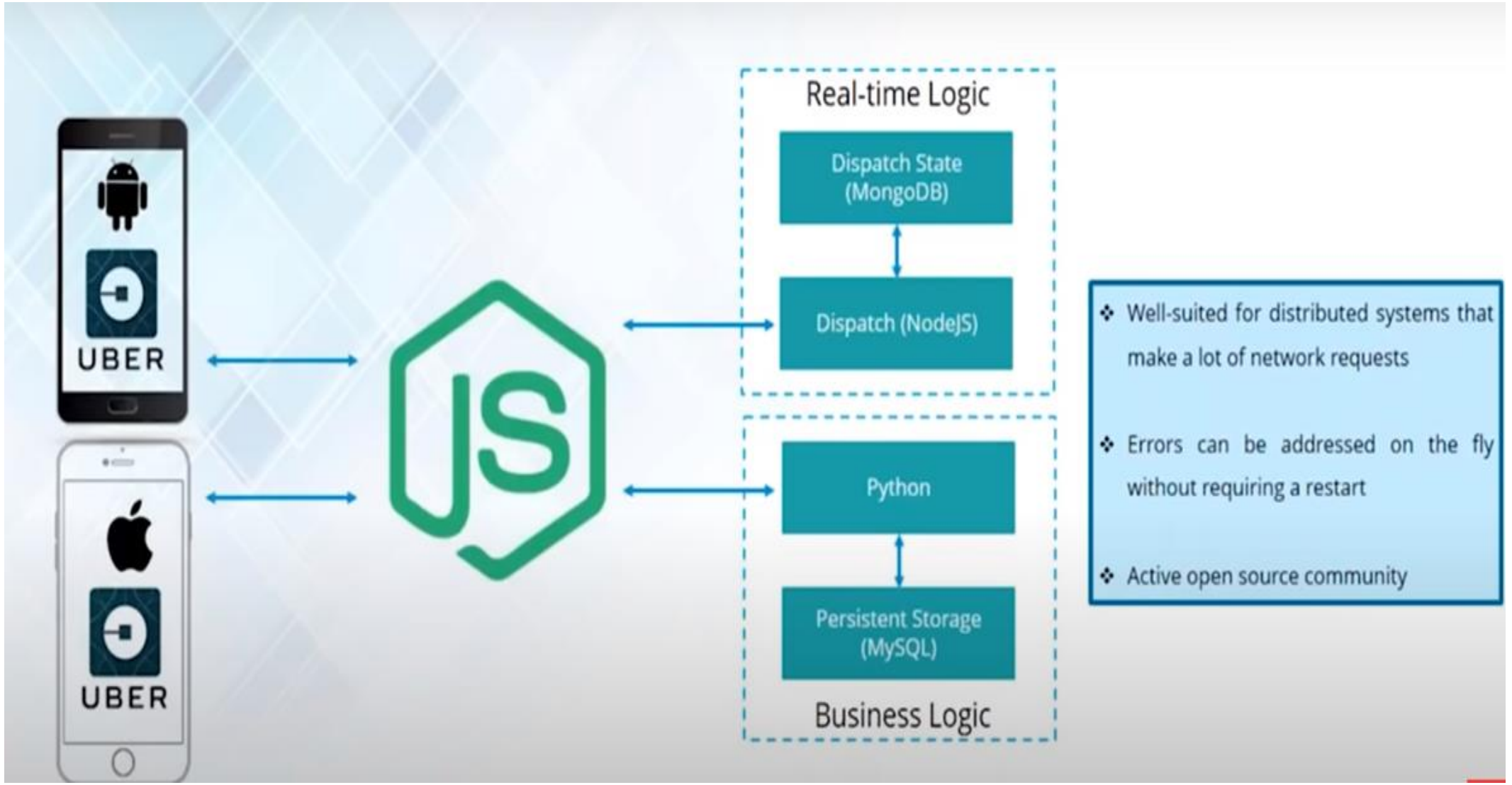
UBER Story – Old Architecture



- Since PHP is a multithreaded language , each user's request is handled in a separate thread
- Reason was car dispatch operation was executed from multiple threads
- Once one car is dispatched for a user, in between the same car get dispatched to another user

NODE JS

UBER Story – New Architecture



NODE JS

UBER Story – Single threaded Model

PES UNIVERSITY

- User performs an activity or event is generated, each new request is taken as an event.
- Event Emitter emit those events and then those events reside inside the event queue in the server.
- Events are executed using event Loops,which is a single thread mechanism
- A worker thread present inside the thread pool is assigned for each request.
- Only one thread in this event Loop will be handling the events directly and process will never get Blocked.

NODE JS

Node JS installation

- Go to https://nodejs.org/en/
- Look for the Latest Stable Version and download it
- After Installing, Verify the installation by giving the command as

```
C:\Users\DELL>node -v
v12.18.3
```

- Install a editor like Visual Code, Sublime Text or any suitable editors for running Node JS Applications

THANK YOU

Revathi G P
Department of
Computer Science and Engineering
revathigp@pes.edu