Express.js
and
Middleware Functions

Outline and Learning Objectives

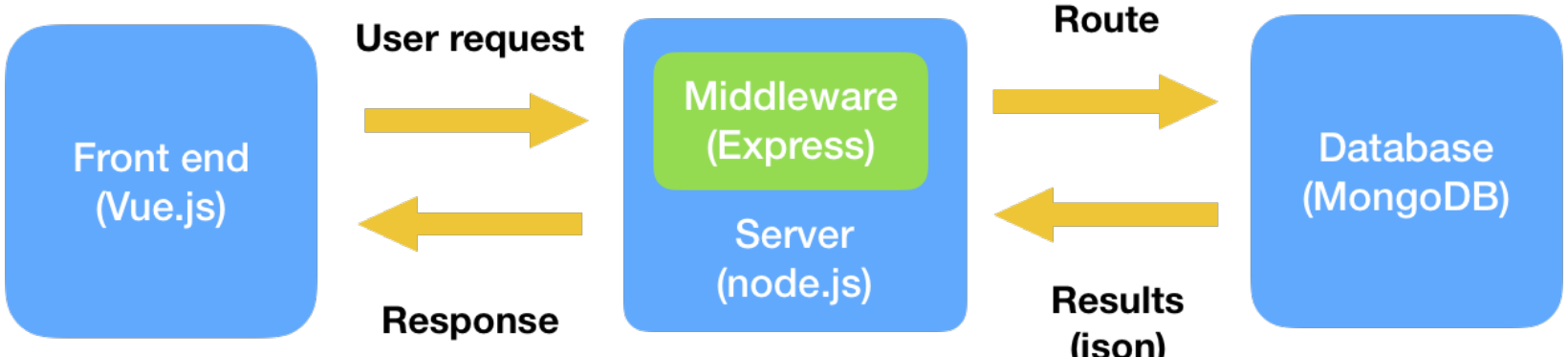
- **Express.js Overview:**
  - to understand how to use Express.js as a Middleware, and what are the main differences and advantages compared to Node.js
- **The Core Parts of Express.js:**
  - to understand the basics of Middleware Functions, Routing Functions, Subapplications, and Conveniences
- **Express.js Installation and First Application:**
  - to understand how to install Express.js, and how to structure a first “Hello World” App
- **Middleware Functions**
  - to understand the design and behaviour of Middleware Functions
  - to build a Static File Server
- **Suggestions for Reading**

Express.js Overview

Express.js and Node.js

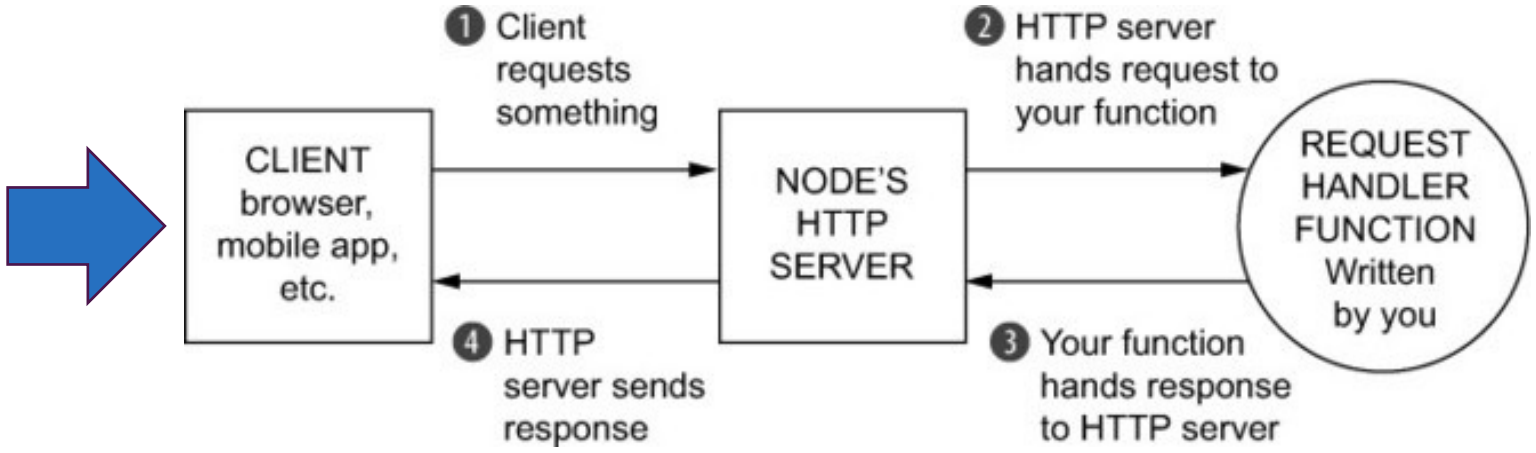
- **What is Express.js?**
- Express.js is a framework that **builds on top of Node.js**
- It makes it **easier to organize** your application's functionality with **Middleware** and **Routing**
- It adds helpful utilities (called **Conveniences**) to Node.js's **HTTP objects**
- It facilitates the rendering of **dynamic HTML views**
- It is **minimal**, **lightweight**, **more extensible** (3rd Party Modules), and **more flexible** compared to other **heavyweight, more rigid, frameworks -> (advantages: easiness, flexibility and extensibility; disadvantages: easier to make errors** when using it and **less rigid, and guided, structure to follow)**

- **What Express adds to Node.js:**
- it adds many helpful **conveniences** to Node.js's HTTP server, **abstracting its complexity**;
- It lets you **refactor one monolithic request handler function** into **many smaller request**;
- **handlers** that handle only **specific bits and pieces** -> This is **more maintainable** and **more modular**



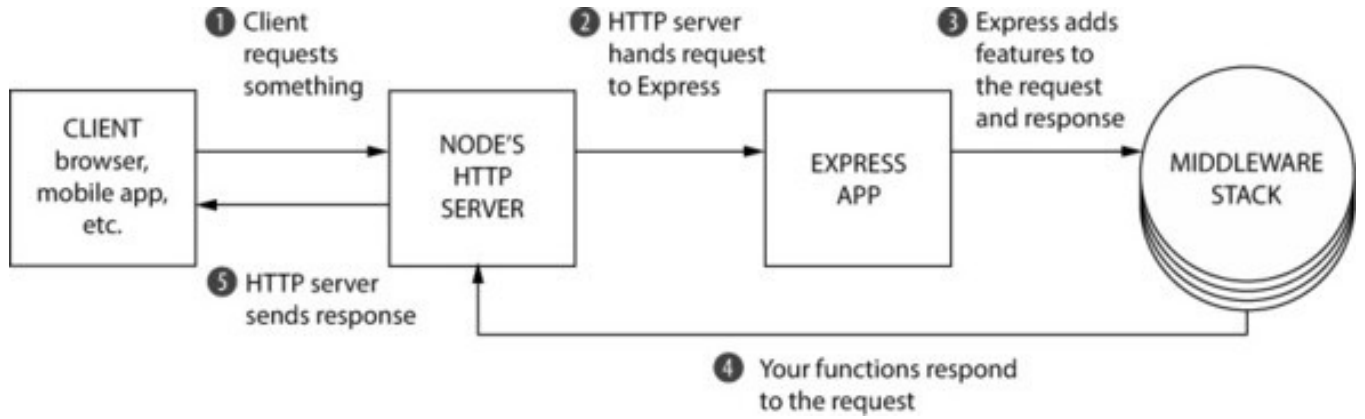
From Node.js to Express.js

- **Node.js:**
- **[Limitation]** 1 Single Javascript function for handling all the requests
- All the “Routing” logic in the same function
- High number of nested “if else”;
- Not very Flexible
- Not easily Maintainable



Node.js **+ Express.js:**

- “Routing” logic organized with **Express Middleware** and **Routing**
- More Flexible, Maintainable and Modular
- **Conveniences** (high-level ready-to-use functions)

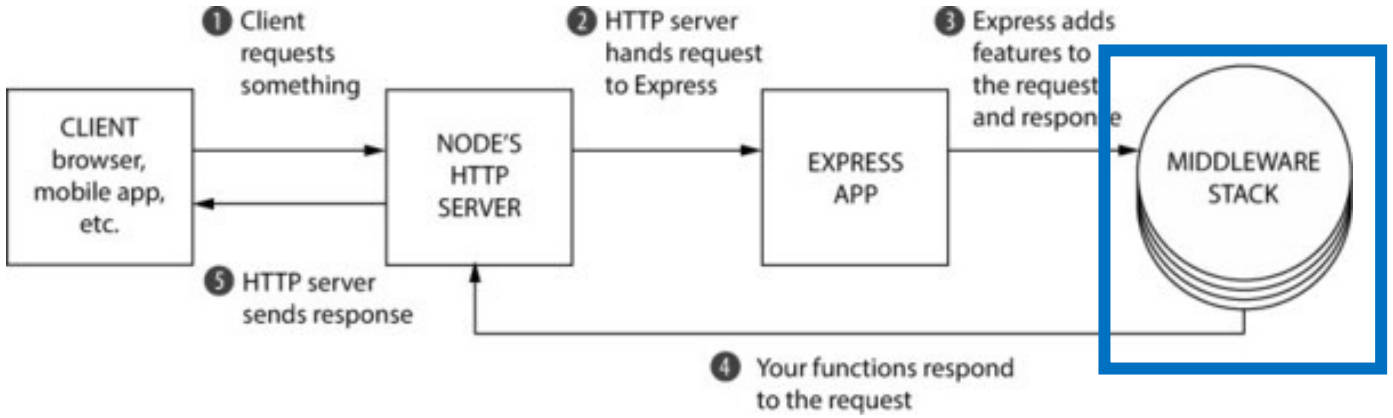


The Core Parts of Express.js (Part 1)

1. Middleware
2. Routing
3. Subapplications (Routers) and Conveniences

Middleware (Functions)

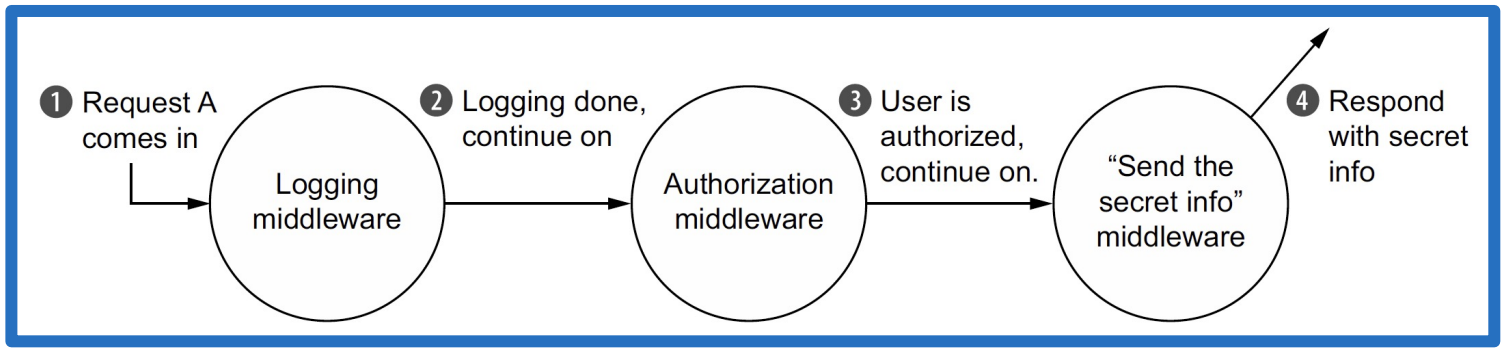
- Rather than one monolithic request handler function (node.js), you call **several request handler functions** that each deal with **a small chunk of the work**.
- These smaller request handlers are called **middleware functions**, or middleware.
- Middleware can handle tasks from **logging requests** to **sending static files**, and many more.
- **(When reached)** Middleware Functions are **always executed**
- Middleware functions are **executed in the order they are added to the stack**
- When one **Middleware finishes**, usually Express will continue onto **next**



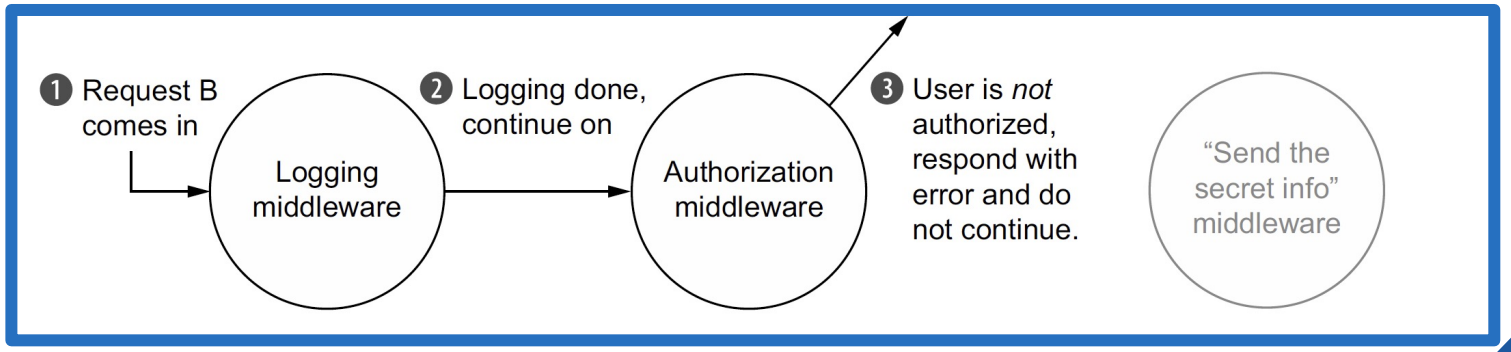
An Example

- The **'logging' middleware is first** in the chain and is always called.
- **Next**, it continues to the **'authorization' middleware**.
- **If the user is authorized, it continues** on to the **next 'secret info' middleware**;
- **Otherwise**, the middleware returns an **error message** and stops the chain.

- Scenario 1



- Scenario 2



The Core Parts of Express.js (Part 2)

1. Middleware
2. Routing
3. Subapplications (Routers) and Conveniences

Routing

- Like **Middleware**, **Routing** breaks the **one big request handler function into smaller pieces**.

- **(When reached)** Routing Functions are **executed on the base of conditions (Client Requests):**
  - **URL requested** (e.g., /homepage)
  - **HTTP Method** (e.g., GET, POST, PUT, DELETE, …)
- Routing functions are **executed in the order they are added to the stack (but conditionally, as above)**
- When one **Routing** Function **finishes**, usually **the response is returned**

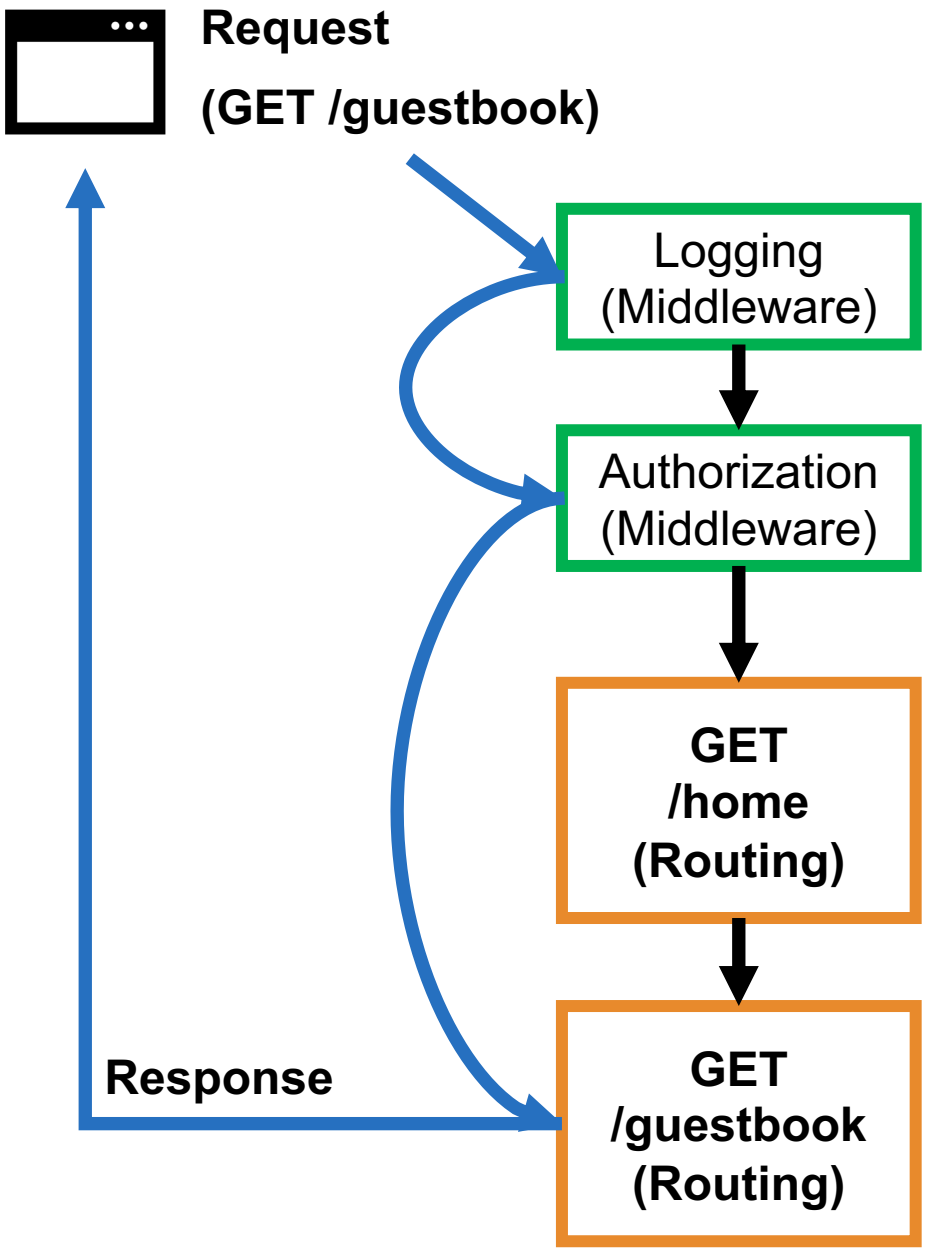
- **For example**, you might build a **web page with a homepage and a guestbook**:
  - when the browser sends an **"HTTP GET"** to the **homepage URL**, Express should **send the homepage**.
  - when the browsers asks for the **guestbook URL**, it should send them the **HTML for the guestbook**.

Routing Vs. Middleware and Example

- **[Common Objective]** rather than one monolithic request handler function, the aim is to have **several request handler functions** that each deal with **a small chunk of the work**.

- **[Middleware] (When reached)** Middleware Functions are **always executed**
- Middleware functions are **executed in the order they are added to the stack**
- When one **Middleware finishes**, usually Express will continue onto **next**

- **[Routing] (When reached)** Routing Functions are **executed on the base of conditions (Client Requests):**
  - **URL requested** (e.g., /homepage)
  - **HTTP Method** (e.g., GET, POST, PUT, DELETE, …)
- Routing functions are **executed in the order they are added to the stack (but conditionally, as above)**
- When one **Routing** Function **finishes**, usually **the response is returned**



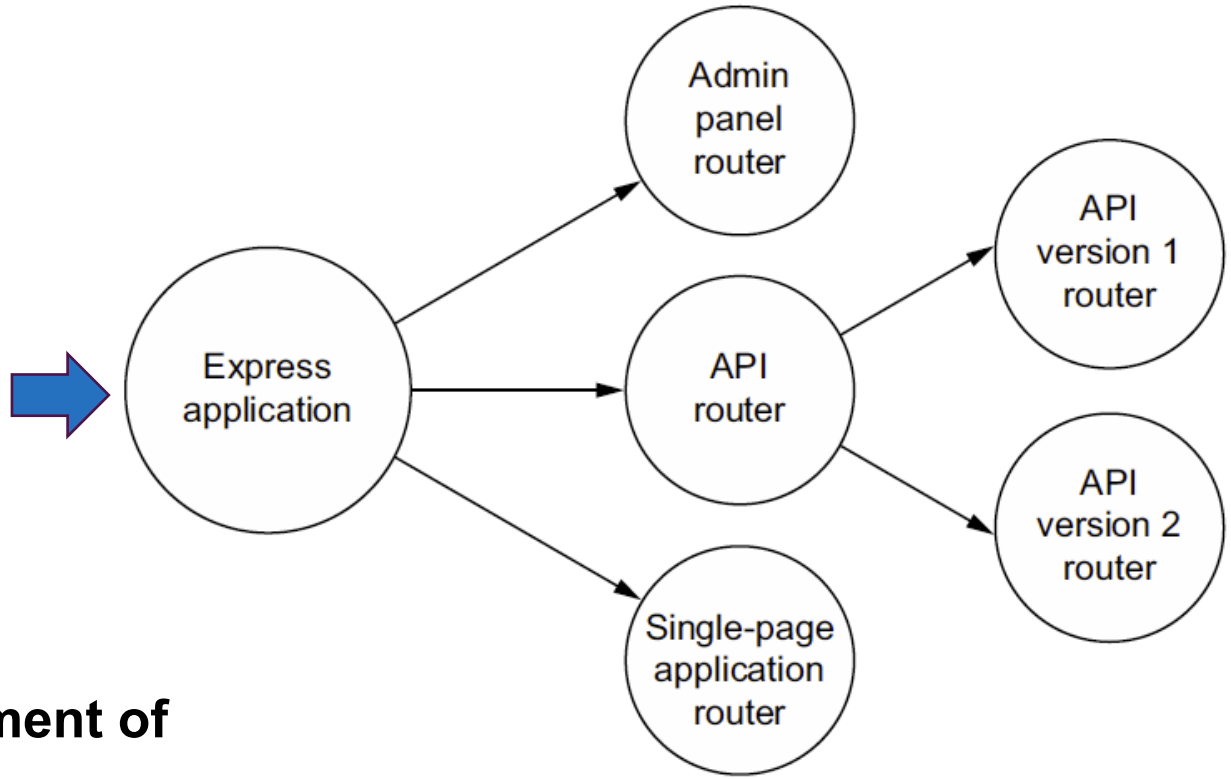
The Core Parts of Express.js (Part 3)

- [x] 1. Middleware
- [x] 2. Routing
- [ ] 3. Subapplications (Routers) and Conveniences

Subapplications (Routers) and Conveniences

Subapplications (Routers)

- Express allows you to define routers that can be used to **break up larger applications**.
  - It allows you to further **compartmentalize your app into smaller pieces**.
- You might have an administration panel in your app, and that can function differently from the rest of your app.



Conveniences

- **High-Level ready-to-use Functions** and **enrichment of existing objects**, **for example**:
  - Adding **more attributes** and **functions** to the **request and response** of your request handler to speed up your coding;
  - To **send a JPEG file from a folder** with Express is just a matter of calling the `sendFile` **function** (while in raw Node.js you should need to write many lines of code)

Express.js Installation and First Application

Install Express

- `npm install express --save`
- Running this command will download the latest version of Express.
- It will add Express in a folder called `node_modules` .
- Adding `--save` to the installation command will save it under the dependencies key of `package.json` .

```
{
    "name": "hello-world",
    "author": "your project name",
    "dependencies": {
        "express": "^5.0.0"
    }
}
```

(**`--save`** was required in previous versions of Node.js, while in the latest Node.js versions you can omit it)

“Hello World” with Express

```
var express = require("express"); // Requires the Express module
var http = require('http');

// Calls the express function to start a new Express application
var app = express();

app.use(function(request, response) { // middleware
    console.log("In comes a request to: " + request.url);
    response.end("Hello, world!");
});

http.createServer(app).listen(3000); // start the server
```

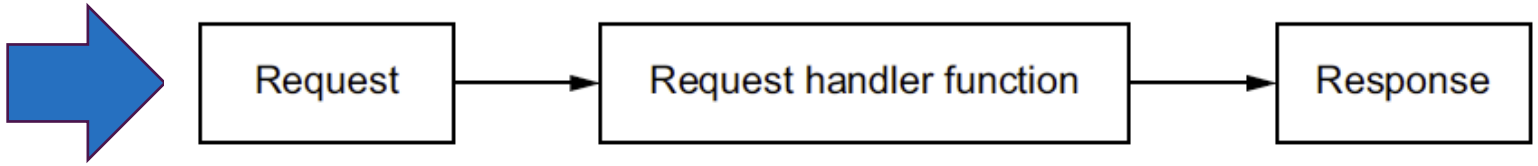
- **Not that different from the Node http server**
- The `app.use` **Middleware** is similar to the **Node.js** `requestHandler` **function**

Middleware Functions (Part 1)

1. Behaviour and Design
2. Example: a Static File Server

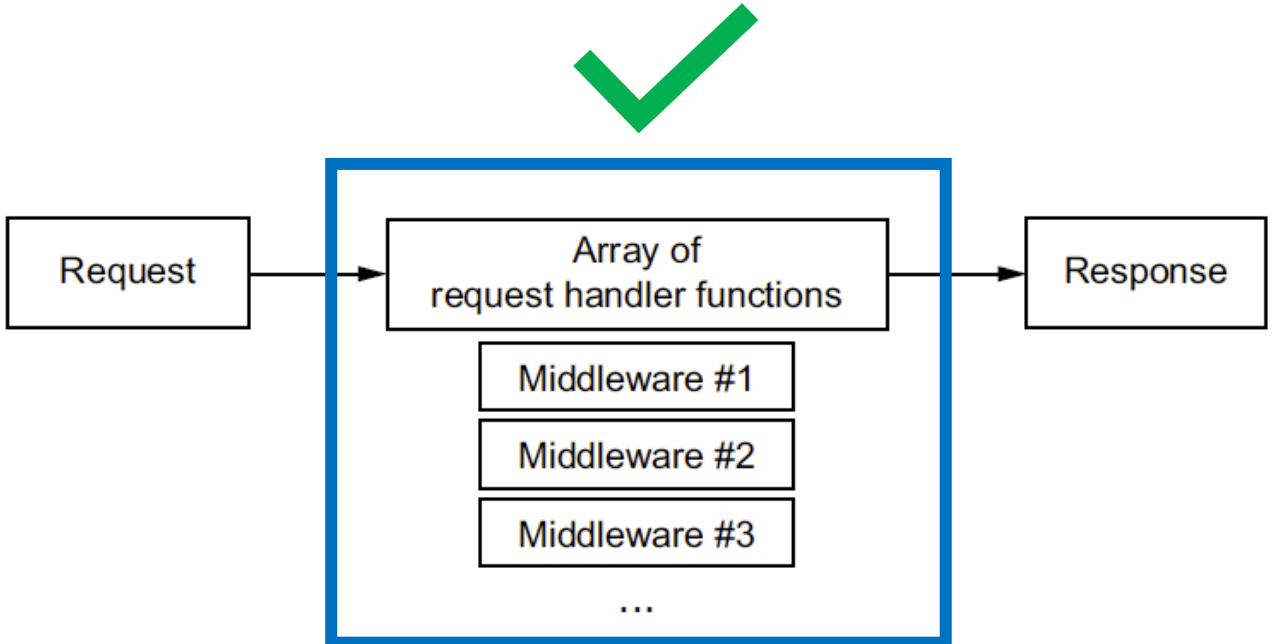
Middleware Design

- **Node.js:**
- **[Limitation]** without Middleware there is only one master request function that handles everything
- All the “Routing” logic in the same function
- High number of nested “if else”;
- Not very Flexible
- Not easily Maintainable



Node.js **+ Express.js:**

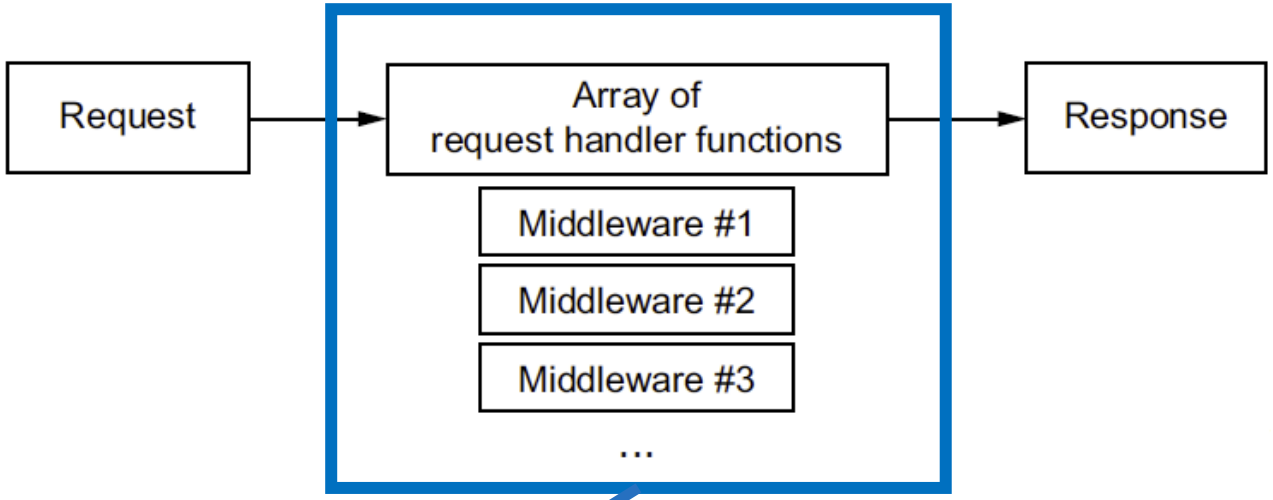
- “Routing” logic organized with **Express Middleware** and **Routing**
- More Flexible, Maintainable and Modular
- **Conveniences** (high-level ready-to-use functions)



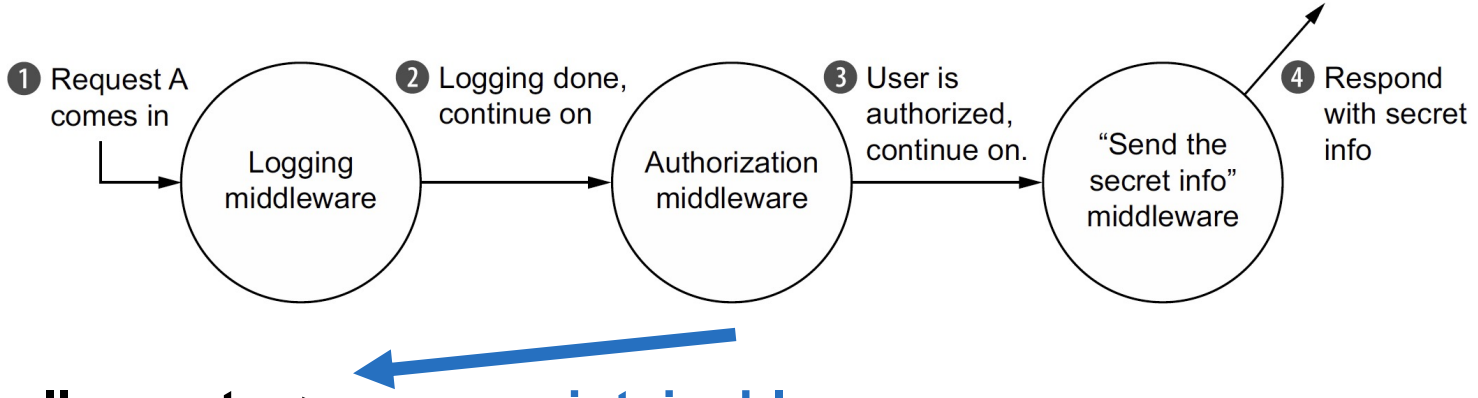
Middleware Design, Example and Types of Middleware Functions

Node.js **+ Express.js:**

- "Routing" logic organized with **Express Middleware** and **Routing**
- More Flexible, Maintainable and Modular
- **Conveniences** (high-level ready-to-use functions)



Example

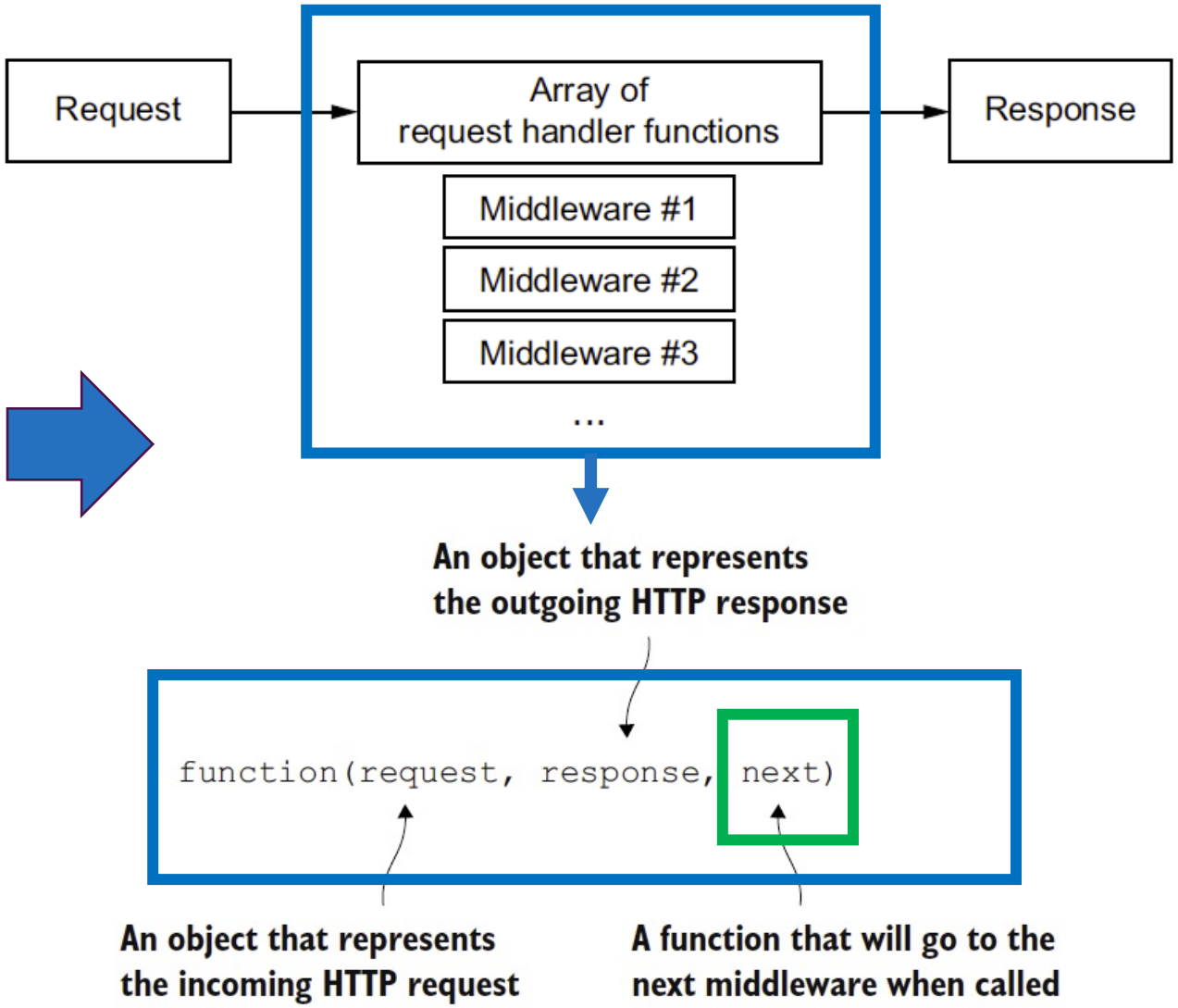


- **Split** your (handler) function into **smaller parts** -> **more maintainable**
- Each middleware can be reused independently elsewhere -> more **modular** and **reusable code** (e.g., **3rd Party Middleware functions** available)
- **[Typologies]** 2 types of Middleware Functions: **Passive** and **Active**

Next Slides

Middleware Function and Input Objects

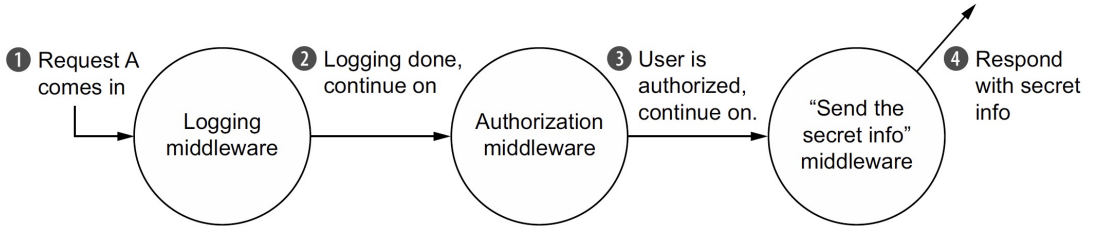
- Every Express middleware takes three arguments.
- The first two are `req` and `res` .
- The third argument is a function called `next` (`req` and `res` are objects).
  - When `next` is called, Express will go on to the next function in the middleware stack.



- **[Typologies]** 2 types of Middleware Functions: **Passive** and **Active**

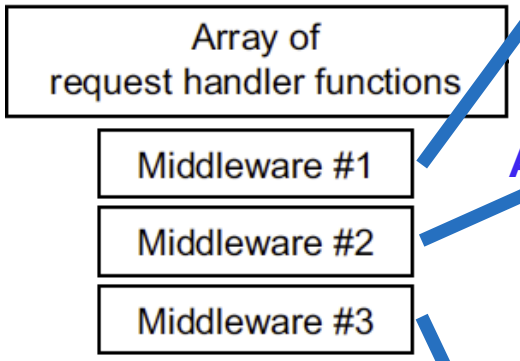
Next Slides

Passive and Active Middleware



[Passive Middleware]: does not change request or response

[Active Middleware]: changes request or response



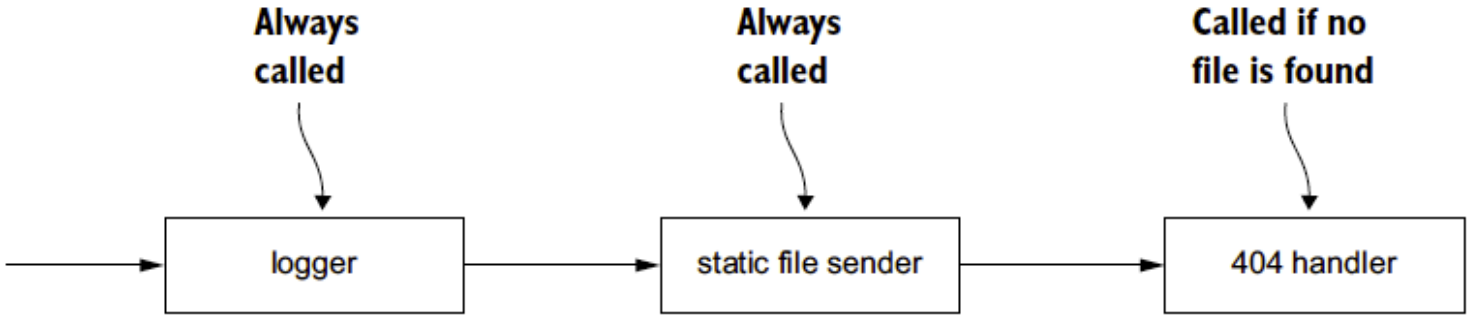

```
var express = require("express");
var http = require("http");
var app = express();

app.use(function(request, response, next) {
    console.log("In comes a " + request.method + " to " + request.url);
    next();
});

app.use(function(request, response, next) {
    var minute = (new Date()).getMinutes();
    if ((minute % 2) === 0) { // continue if it is on a even minute
        next();
    } else { // otherwise responds with an error code and stops
        response.statusCode = 403;
        response.end("Not authorized.");
    }
});

app.use(function(request, response) { // only run if authorised
    response.end('Secret info: the password is "swordfish"!');
});

http.createServer(app).listen(3000);
```

Middleware Functions (Part 2)

1. Behaviour and Design
2. Example: a Static File Server

Example App: a Static File Server

- An app that serves files from a folder (HTML files, images, or an MP3):
  - **this folder** will be called `static`
  - if there is a file called `celine.mp3` and a user visits `/celine.mp3` , your server should send that MP3 over the internet
  - if the user requests `/burrito.html` and **no such file exists** in the folder, your server should send a **404 error**
- The server also **logs every request**, whether or not it is successful.
  - **It logs the URL** that the user requested with **the time that they requested it**.



- This **Express application** will have **three middleware functions**:
  - **The 'logger'** that will output the requested URL and time to the console.
  - **The 'static file sender'** that will check if the file exists in the folder.
    - If it does, it will send that file over the internet (and end the request).
    - If the requested file does not exist, it will continue on to the final middleware.
  - **The '404 handler'** that will return a 404 message and finish up the request.

First Steps and npm start

Let's create the `package.json`

```
{
    "name": "static-file-fun",
    // Tells Node not to publish in the public module registry
    "private": true,
    "scripts": {
        // When you run npm start, it will run node app.js.
        "start": "node app.js"
    }
}
```

Why use `npm start` and not just `node app.js` ?

1. **It is a convention**. Most Node web servers can be started with `npm start`.
    - The main file can be called as you prefer (e.g., `application.js` instead of `app.js`)
2. It allows you **to run a more complex command** or **set of commands**
    - Commands with **arguments**
    - **Multiple commands**
3. Allow you **to run locally installed command** (not globally installed)
    - for example **version different** from that of the global package.

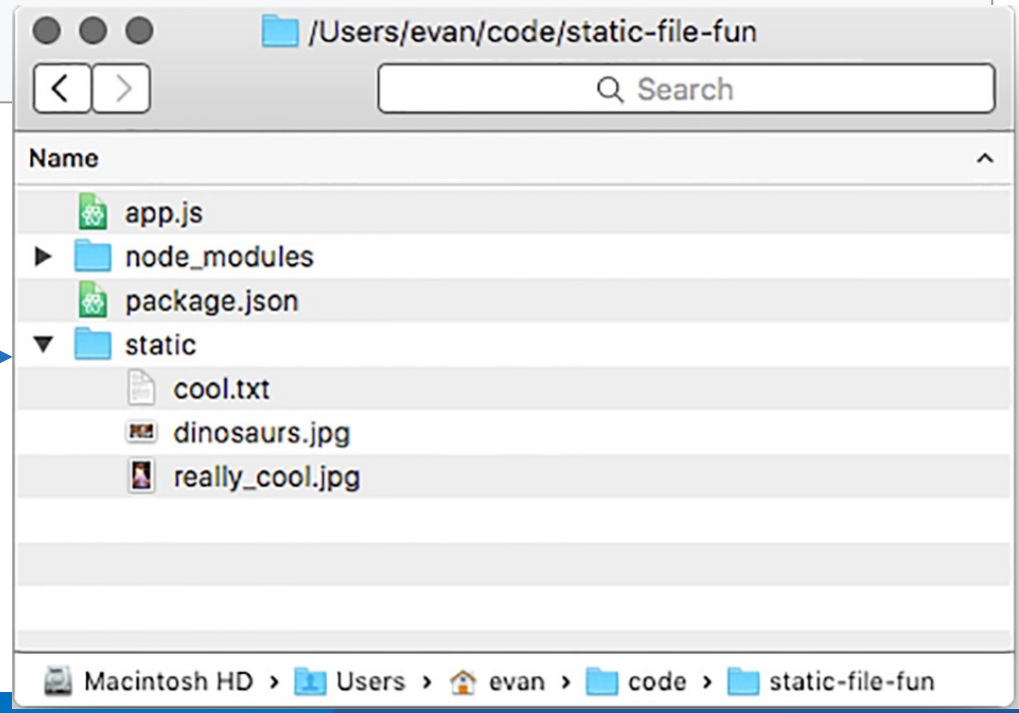
Next Steps

Let's install Express:

```
npm install express --save
```

```
{
    "name": "static-file-fun",
    // Tells Node not to publish in the public module registry
    "private": true,
    "scripts": {
        // When you run npm start, it will run node app.js.
        "start": "node app.js"
    },
    "dependencies": {
        "express": "^5.0.0"
    }
}
```

- Create a **folder** called `static`
- **Put a few files inside**: maybe an HTML file or an image or two
- **Create** `app.js` in the root of your project

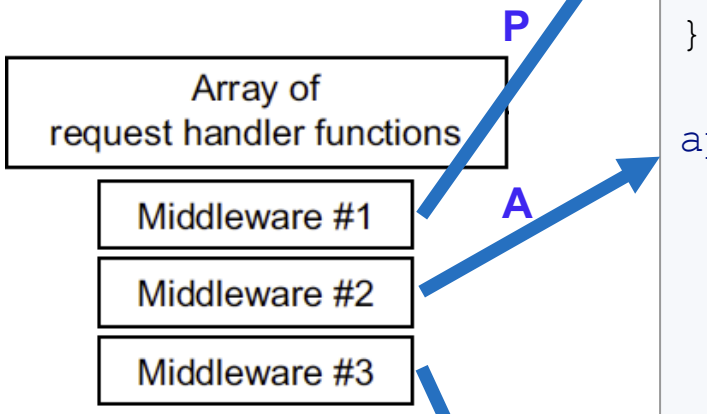


The 3 Middleware Functions (Basic Part)

```
var express = require("express");
var path = require("path");
var fs = require("fs");

var app = express();

...

app.listen(3000, function() {
    console.log("App started on port 3000");
});
```

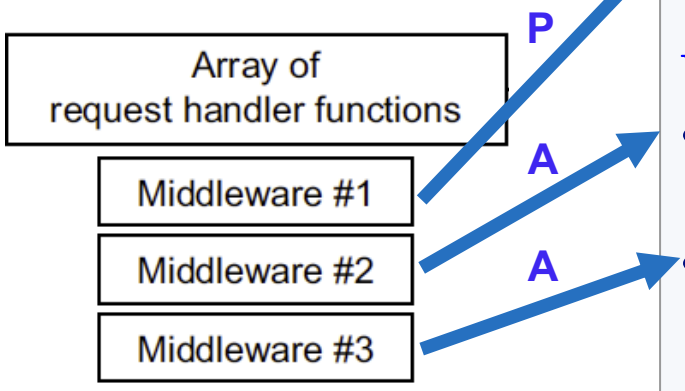
Next Slide

The 3 Middleware Functions (Core Part)




```
...
app.use(function(req, res, next) {
    console.log("Request IP: " + req.url);
    console.log("Request date: " + new Date());
    next();
});

app.use(function(req, res, next) {
    var filePath = path.join(__dirname, "static", req.url);
    fs.stat(filePath, function(err, fileInfo) {
        if (err) {
            next();
            return;
        }

        if (fileInfo.isFile()) {
            res.sendFile(filePath);
        } else {
            next();
        }
    });
});

app.use(function(req, res) {
    res.status(404);
    res.send("File not found!");
});
...
```

The 3 Middleware Functions (Reusing 3rd Party Middleware Functions)




```
var express = require("express");
var morgan = require("morgan");
var path = require("path");

var app = express();

app.use(morgan("short"));

var staticPath = path.join(__dirname, "static");
app.use(express.static(staticPath));

app.use(function(req, res) {
    res.status(404);
    res.send("File not found!");
});

app.listen(3000, function() {
    console.log("App started on port 3000");
});
```

Suggestions for Reading

Reading

Chapter 1 of the **“Express in Action”** textbook

Chapter 3 of the **“Express in Action”** textbook

Chapter 4 of the **“Express in Action”** textbook

Questions?