REST, NODE, Express

- Node.js is an event-driven, I/O model-based runtime environment and library for developing server-side web apps using JS.

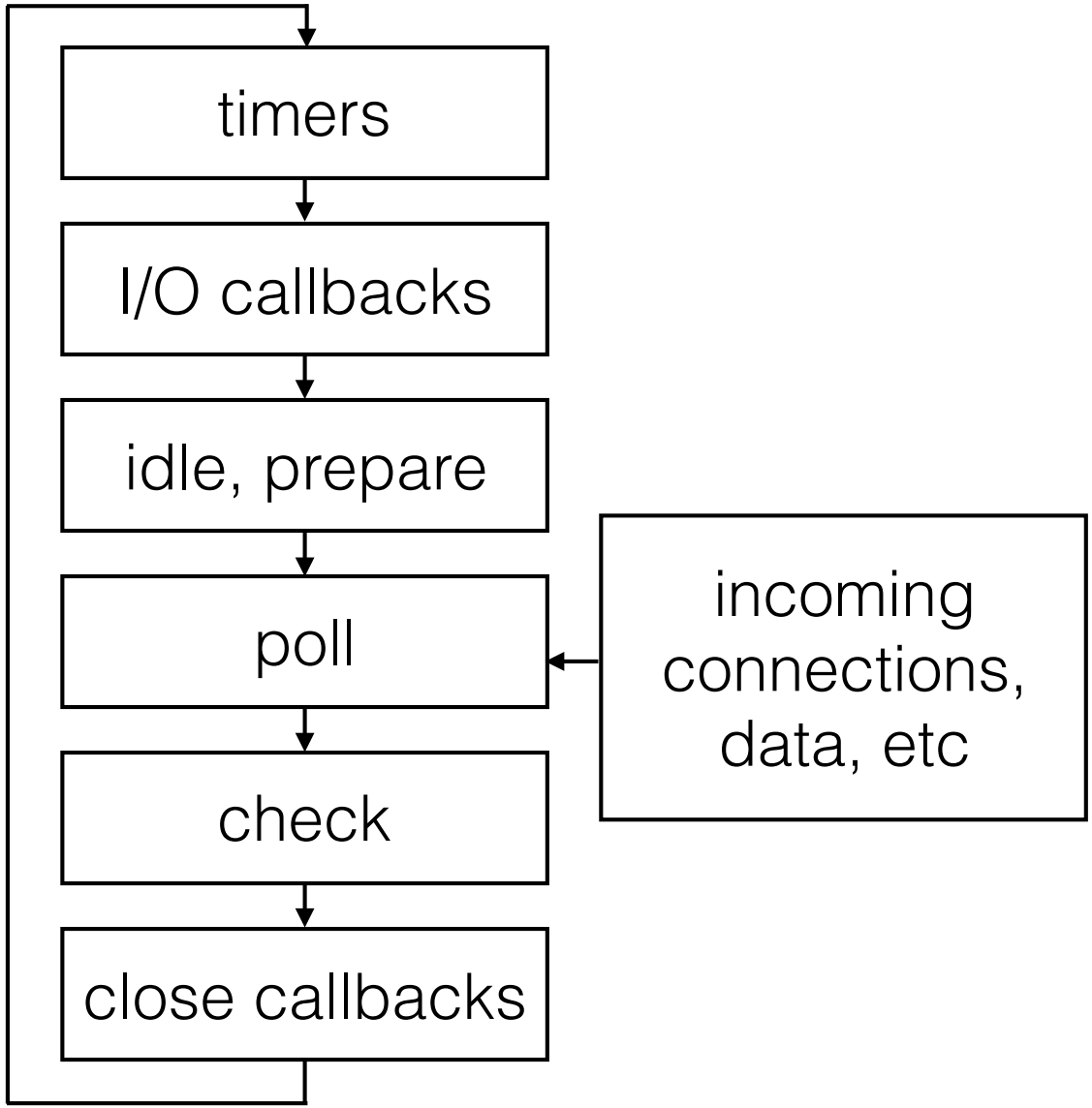
- Uses the V8 Javascript Engine

Node.js

- Use a JavaScript engine from a browser (Chrome's V8 engine)
  - Get the same JavaScript on browser and server
  - Don't need the DOM on the server
- Add events and an event queue
  - Everything runs as a call from event loop
- Make event interface to all OS operations
  - Wrap all OS blocking calls (file and network I/O)
  - Add some data handling support
- Add a proper module system
  - Each module gets its own scope

Event Loop

- Makes use of OS async operations
- 4 worker threads to handle other async operations



Node Modules

- Import using require()
  - System module: require(“fs”); // Looks in node_module
  - From a file: require(“./mod.js”); // reads specified file
  - From a directory: require(“./myModule”); // reads myModule/index.js
- Modules have a private scope
  - Require returns what is assigned to module.exportss

Node Modules

- Many standard Node modules
- Huge library of modules (npm)
- We will use:
  - Express - “Fast, unopinionated, minimalist framework”
  - Mongoose - mongoldb object modelling

npm

`npm init`

will create a `package.json` file

Then for any new modules that you want to install, use

`npm install module --save`

```
{
  "name": “books",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "dependencies": {
    "body-parser": "^1.15.2",
    "ejs": "^2.5.2",
    "express": "^4.14.0",
    "express-validator": "^2.20.10"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}
```

Git and installing modules

- You should not store generated files in git
  - wastes space, and can lead to confusion
- Add a **`.gitignore`** file to ignore files or directories that are generated.
- Example **`.gitignore`** file:

```
node_modules
npm-debug.log
```

Express.js

- Relatively thin layer on top of the base Node.js functionality
- What does a web sever implementor need?
  - Speak HTTP: Node's HTTP module does this
  - Routing: Map URLS to the web server function
  - Middleware support: Allow request processing layers to be added. Custom support for sessions, cookies, security, compression, etc.

```
var express = require('express');

var expressApp = express();
```

- expressApp object has methods for:
  - Routing HTTP requests
  - ○ Rendering HTML (e.g. run a preprocessor like Jade templating engine)
  - ○ Configuring middleware and preprocessors

```
expressApp.get('/', function (httpRequest, httpResponse)
{
      httpResponse.send('hello world');
});
expressApp.listen(3000);
```

Express routing

- By HTTP method:

```
expressApp.get(urlPath, requestProcessFunction);
expressApp.post(urlPath, requestProcessFunction);
expressApp.put(urlPath, requestProcessFunction);
expressApp.delete(urlPath,
                  requestProcessFunction);
expressApp.all(urlPath, requestProcessFunction);
```

- Many others less frequently used methods
- urlPath can contain parameters (e.g. '/user/:user_id')

httpRequest object

```
expressApp.get('/user/:user_id',
                function (httpRequest, httpResponse) ...
```

- Object with large number of properties
- Middleware (like JSON body parser, session manager, etc.) can add properties

`request.params` - Object containing url route params (e.g. user_id)

`request.query` - Object containing query params (e.g. &foo=9 ⇒ {foo: '9'})

`request.body` - Object containing the parsed body

`request.get(field)` - Return the value of the specified HTTP header field

httpResponse object

```
expressApp.get('/user/:user_id',

                    function (httpRequest, httpResponse) ...
```

- Object with a number of methods for setting HTTP response fields

`response.write(content)` - Build up the response body with content

`response.status(code)` - Set the HTTP status code of the reply

`response.set(prop, value)` - Set the response header property to value

`response.end()` - End the request by responding to it

`response.send(content)` - Do a write and end