Asynchronous JavaScript

- Multi-tasking
- Single-threaded
- Event loop
- Callbacks
- Promises
- Async/await

Multi-tasking

- Multi-tasking is the ability to run multiple tasks at the same time
- Multi-tasking can be achieved in two ways:
  - Multi-threading
  - **Event loop**

```
ExecutorService executor = Executors.newFixedThreadPool(2);
executor.submit(() -> {
  System.out.println("Asynchronous task in Java");
});
executor.shutdown();
```

Threads vs. Processes

- A process is an instance of a computer program that is being executed
- A thread is a sequence of instructions within a program that can be executed independently of other code
- A process can have multiple threads

Broswer Multi-Process Architecture

- Chrome: tabs are processes
- main thread and IO thread

Rendering Process

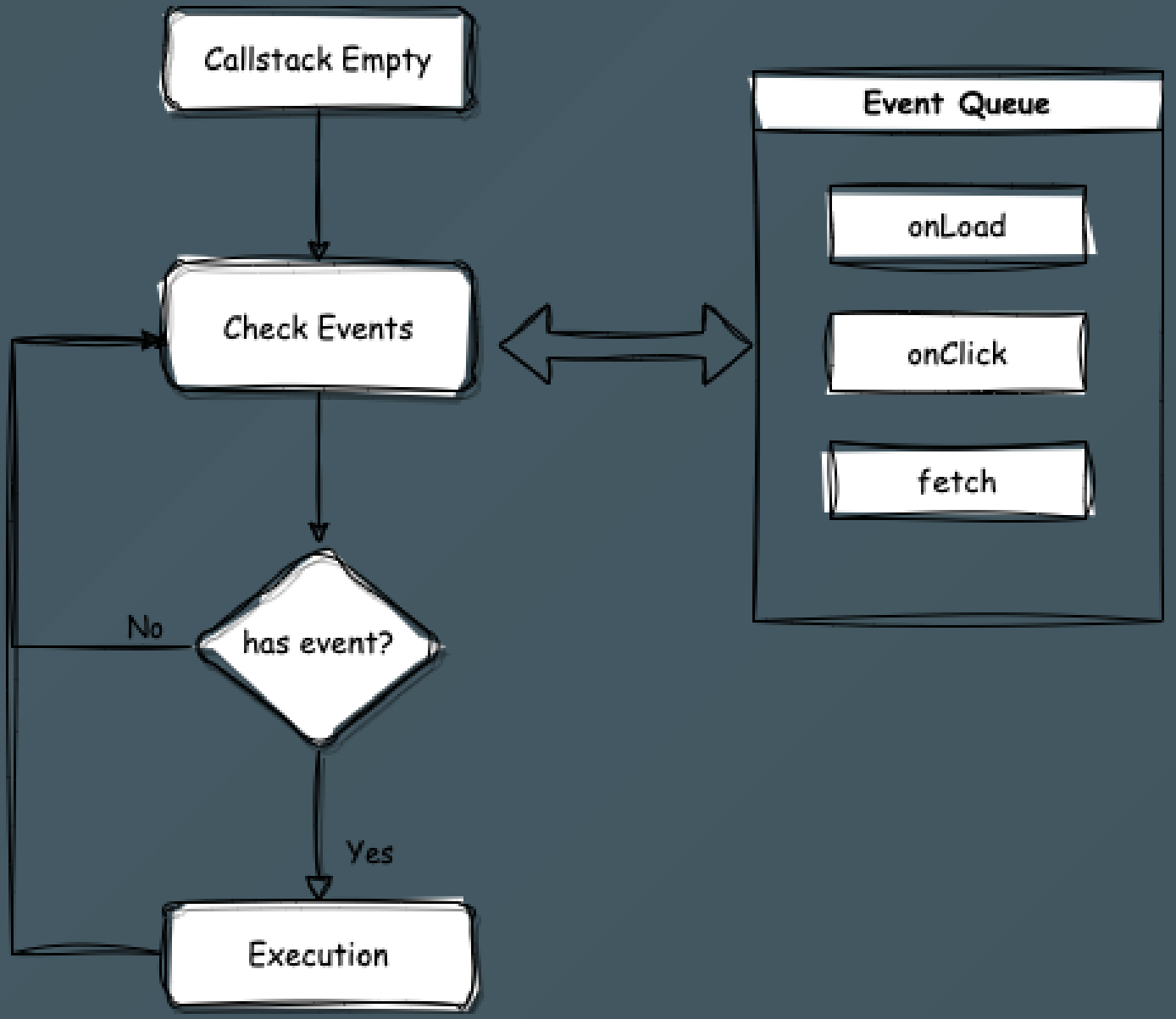
- The rendering process is responsible for everything that happens inside of a tab
  - Parsing HTML
  - Style calculations
  - Layout
  - Paint
  - ...etc

How JavaScript Works

- JavaScript is single-threaded
  - created V8 instance under main thread
  - simplifies interactions
- Most of the time, JavaScript runs in the main thread
  - exception: Web Workers
- Concurrency vs Parallelism

Event Loop

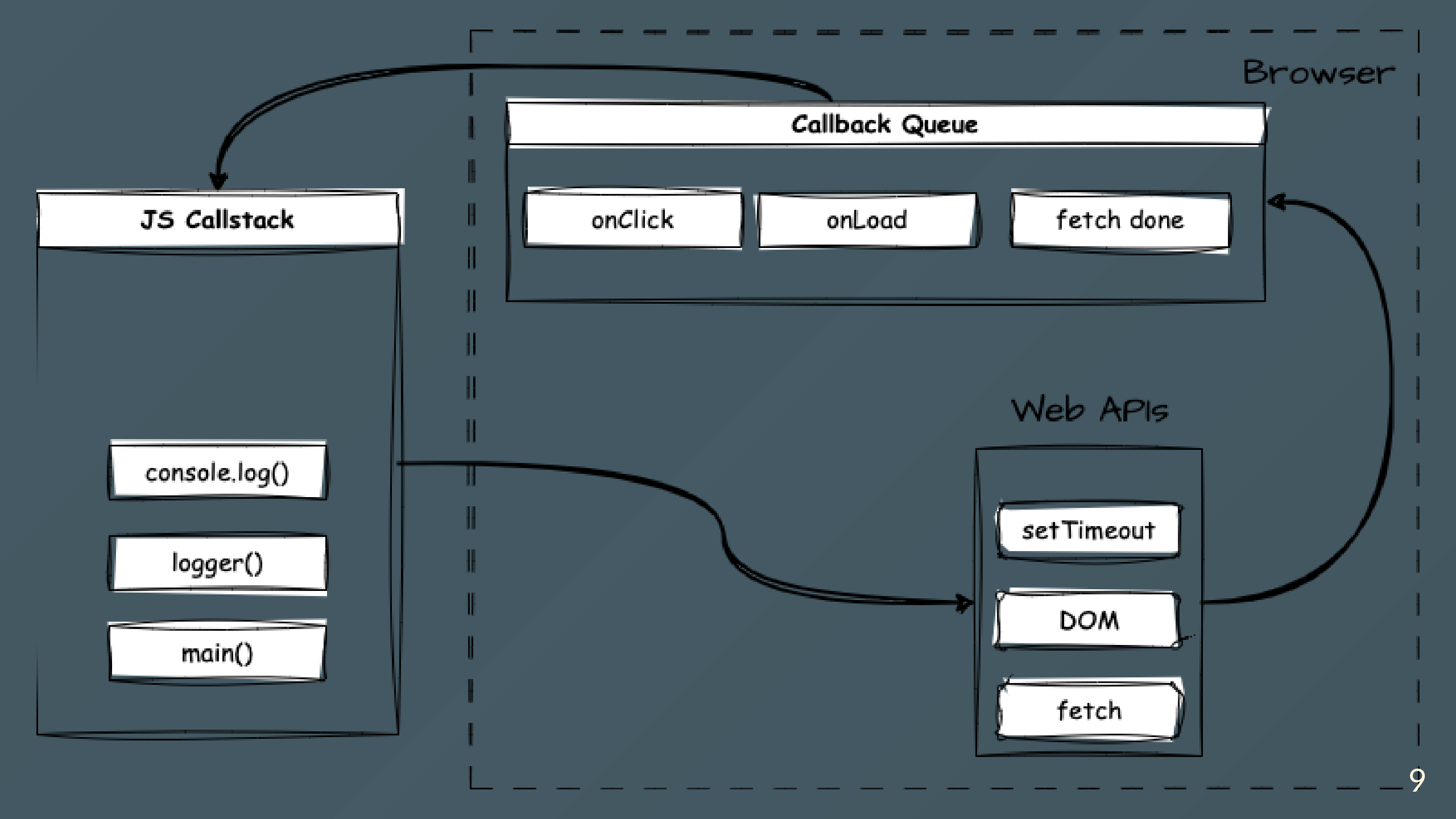
- Event loop is a mechanism that handles asynchronous tasks



```
function logger() {
  console.log('log');
}

button.addEventListener('click', logger);

logger();
```


Browser
Callback Queue
onClick
onLoad
fetch done
JS Callstack
console.log()
logger()
main()
Web APIs
setTimeout
DOM
fetch

Asynchronous Programming with Callbacks

- Callbacks are functions that are passed as arguments to other functions
- Callbacks are executed after the function they are passed to completes

```
button.addEventListener('click', () => {
  console.log('button clicked');
});
```

Timers

- Timers are used to schedule tasks to be executed at a later time

```
setTimeout(() => {
  console.log('Hello');
}, 1000);
```

```
let intervalId = setInterval(() => {
  console.log('Hello');
}, 1000);

function stopInterval() {
  clearInterval(intervalId);
}
```

Network Events

- Network events are used to handle network requests

```
fetch('https://api.github.com/users')
  .then(response => response.json())
  .then(data => console.log(data));
```

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.github.com/users');
xhr.onload = () => {
  console.log(xhr.response);
};
xhr.send();
```

How Tasks are Executed

- Tasks are executed in the order they are added to the task queue

```
console.log('start');

setTimeout(() => {
  console.log('setTimeout');
}, 0);

// Promise.resolve().then(() => {
//   console.log('Promise');
// });

console.log('end');
```

http://latentflip.com/loupe/?code=Y29uc29sZS5sb2coInN0YXJ0Iik7CgokLm9uKCcjYnRuJywgJ2NsaWNrJywgZnVuY3Rpb24gKCkgewogICAgY29uc29sZS5sb2coJ2NsaWNrZWQgdGhlIGJ1dHRvbicpOwp9KQoKc2V0VGltZW91dCgnbW5jdGIvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coInNldFRpbWVvdXQgMjAwMCIpOwp9LCAyMDAwKTsKCnNldFRpbWVvdXQoZnVuY3Rpb24gdGltZW91dCgpIHsKICAgIGNvbnNvbGUubG9nKCJzZXRUaW1lb3V0IDMwMDAiKTsKfSwgMzAwMCk7CgpzZXRUaW1lb3V0KGZ1bmN0aW9uIHRpbWVvdXQoKSB7CiAgICBjb25zb2xlLmxvZyhzZXRUaW1lb3V0IDA0MDAwIik7CnsgfSwgIDQwMDApOwoKY29uc29sZS5sb2coImVuZCIpOw%3D%3D!!!PGJ1dHRvbiBpZD0iYnRuIj5jbGljayBtZSE8L2J1dHRvbj4%3D

Callback Hell

callback-hell.js

- Callback hell is a phenomenon that occurs when there are too many nested callbacks
- Callback hell makes code hard to read and maintain
- Hard to handle errors

Promises

promises.js

- What is a promise?
- How to create a promise?
- **How to use a promise?**
- How to handle errors?

Terminology

- Promise
- Pending, Fulfilled, Rejected, Settled, Resolved
- Resolve, Reject
- Then, Catch, Finally

Promises (cont'd)

- Promises are objects that represent the *result* of an asynchronous operation
- Promises have three states:
  - Pending
  - Fulfilled
  - Rejected

Chaining Promises

- Promises can be chained
- The result of a promise is passed to the next promise in the chain

```
fetch('https://api.github.com/users')
  .then(response => response.json())
  .then(data => console.log(data));
```

Resolving Promises

- when a promise is resolved with a value that is not itself a promise, the promise is fulfilled with that value
- if the value is a promise, the promise is resolved but not yet fulfilled. The fulfillment of the promise is handled by the promise that was passed in

Create a Promise

promise-creation.js

```
const promise = new Promise((resolve, reject) => {
  // do something
  if (/* success */) {
    resolve(/* value */);
  } else {
    reject(/* reason */);
  }
});
```

Promises in Parallel

promise-parallel.js

- Promises can be executed in parallel
- `Promise.all()`, `Promise.race()`, `Promise.allSettled()`

`async` - `await`

async-await.js

- `async` - `await` is a syntactic sugar for promises
- `async` - `await` makes asynchronous code look like synchronous code
- `async` - `await` is built on top of promises

Error Handling

try-catch.js

- coder always make mistakes, called bugs
- some bugs can be handled by try-catch
- error object
  - name
  - message
  - stack

How Tasks are Executed (cont'd)

- Tasks are executed in the order they are added to the task queue

```
console.log('start');

setTimeout(() => {
  console.log('setTimeout');
}, 0);

Promise.resolve().then(() => {
  console.log('Promise');
});

console.log('end');
```

Microtasks (jobs) vs. Macrotasks (tasks)

- Microtasks are executed before macrotasks
- Microtasks : JS
- Macrotasks : DOM, Network, Timer, Node.js
- https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/