Thinking Asynchronously

Paul O’Fallon

@paulofallon

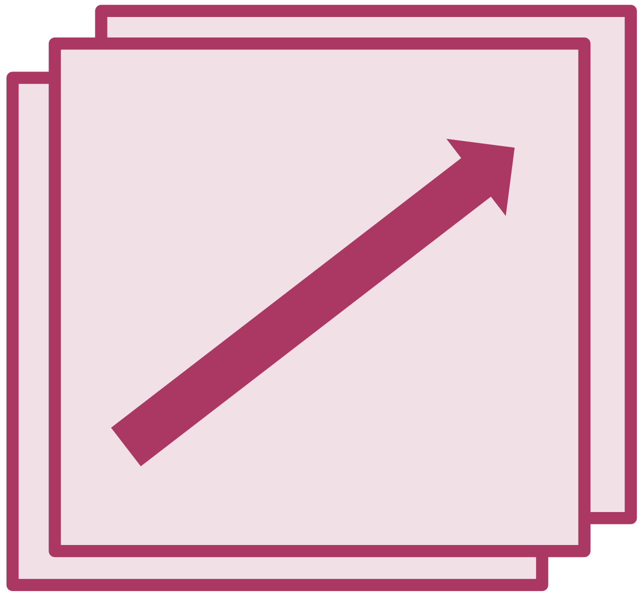
Overview

Node’s event loop

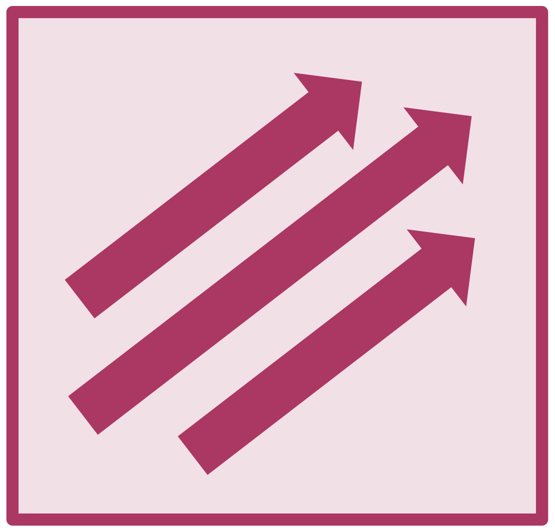
Asynchronous development model

How Node leverages this approach

How Is Node.js Different?



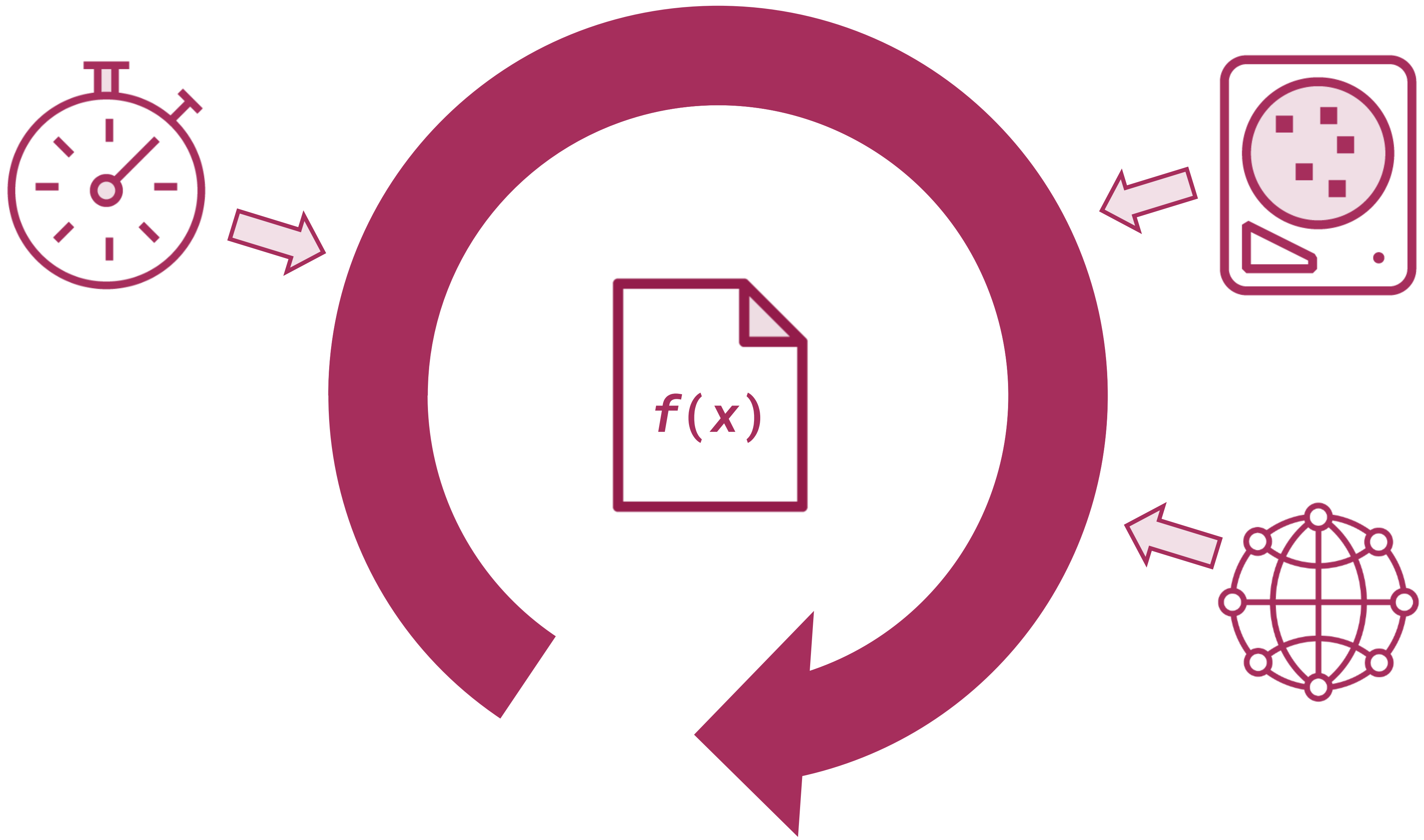
Process-Per-Client
(Multi-Process)



Thread-Per-Client
(Multi-Threaded)

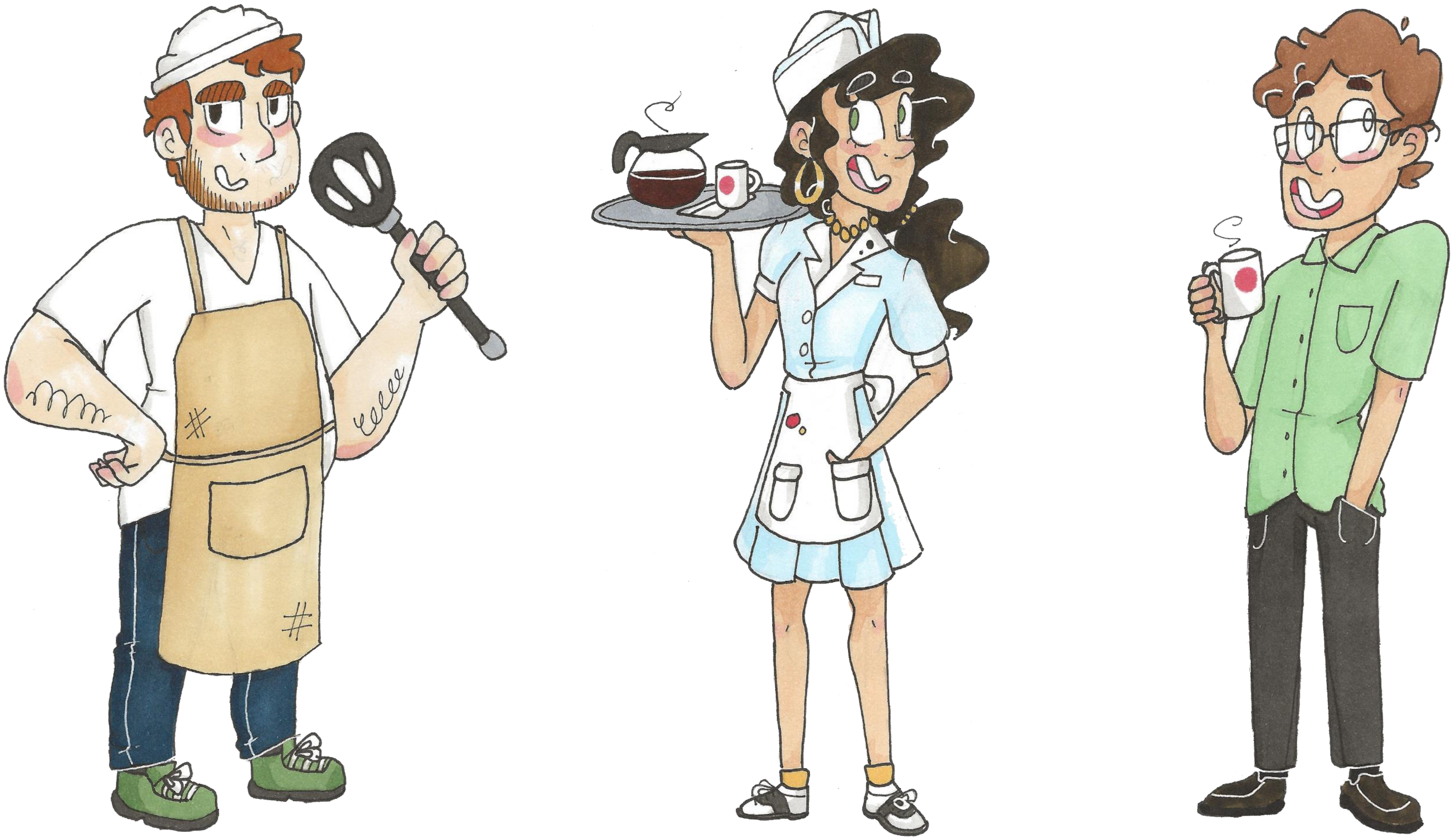
Event Loop
(“Single Threaded”)

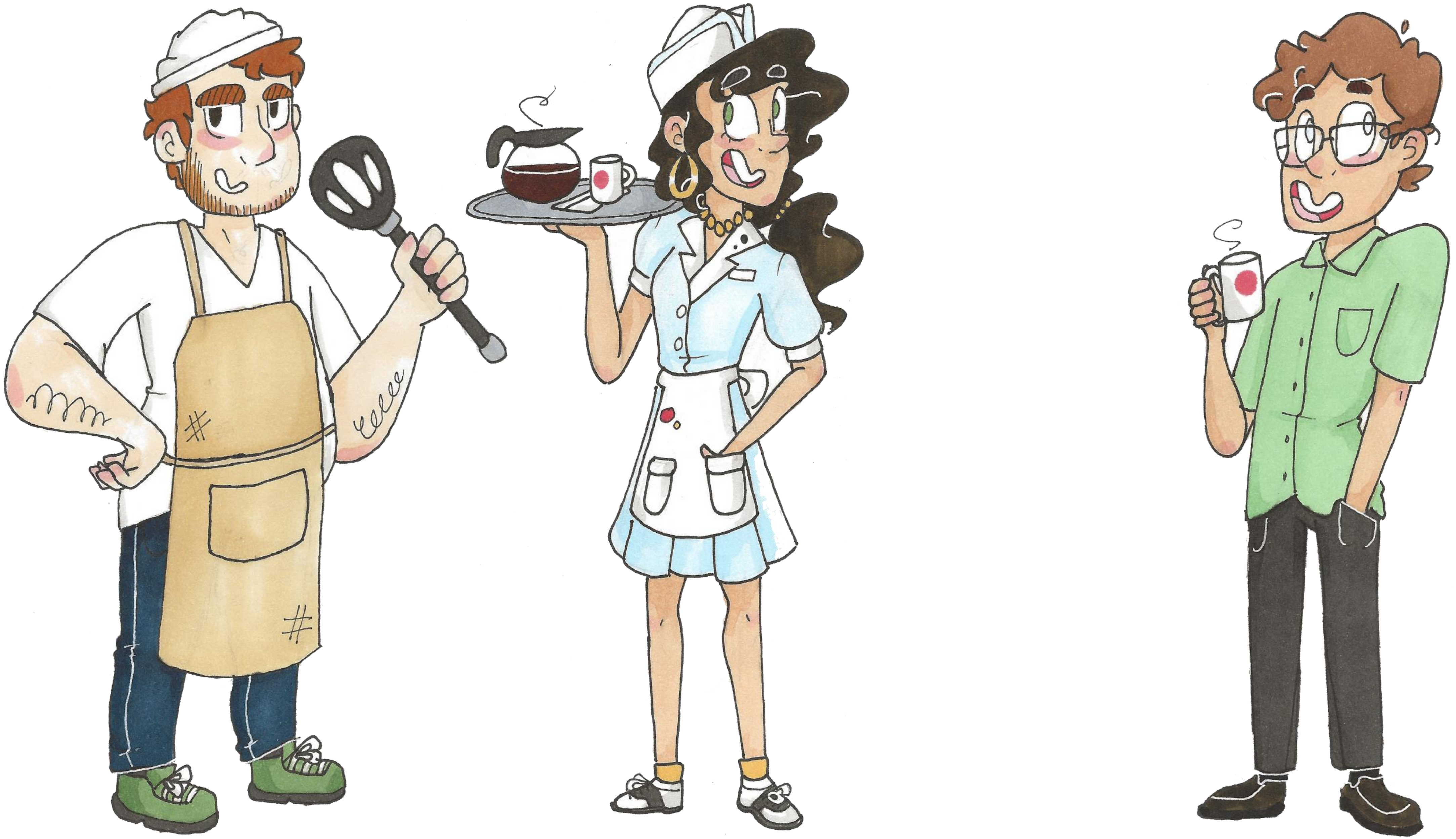
Node's Event Loop

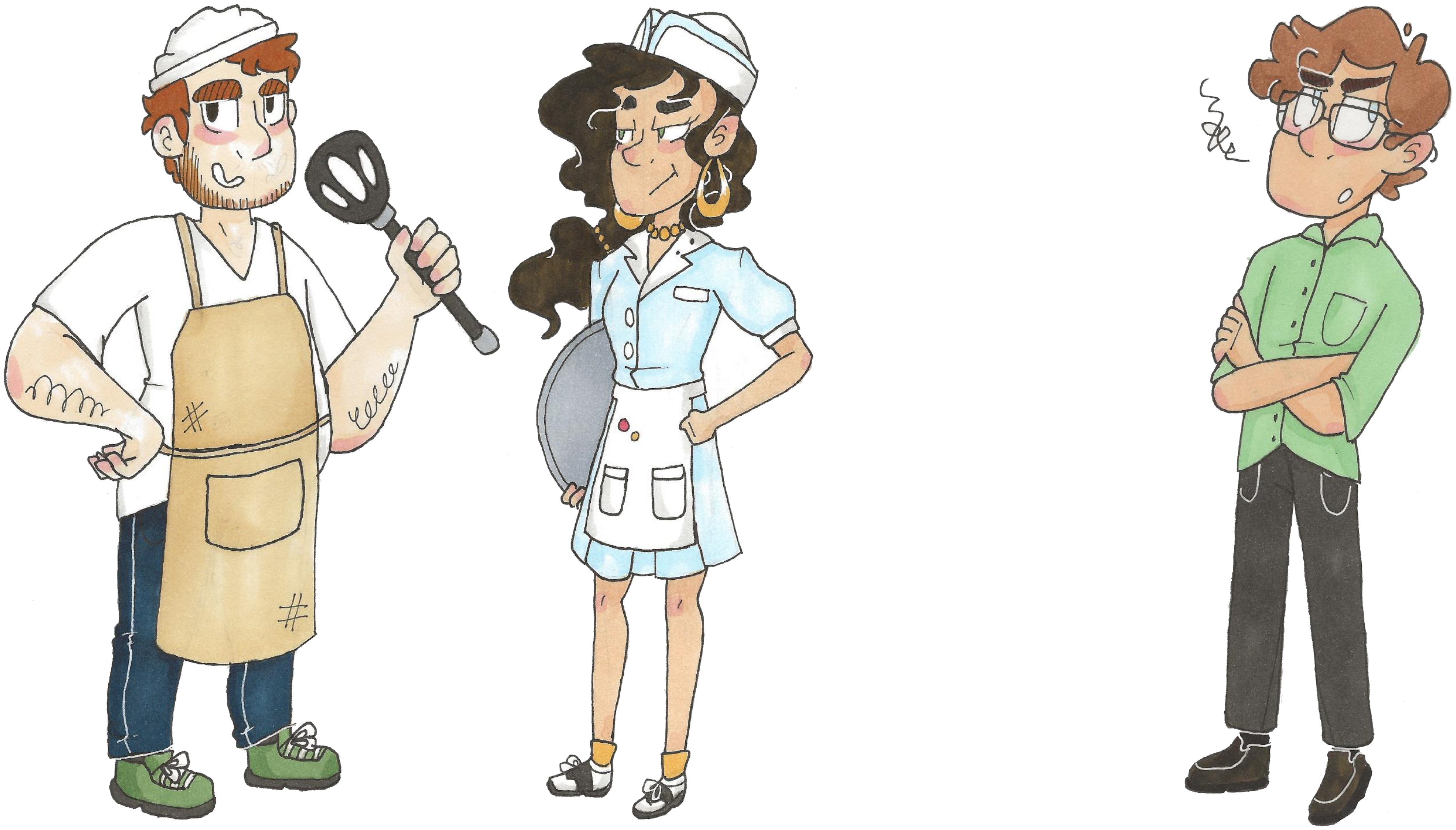


"The fair treatment of clients is thus the responsibility of your application."

https://nodejs.org/en/docs/guides/dont-block-the-event-loop/







Traditional “Synchronous” Programming

```
function serveCustomer(customer) {
  let order = customer.placeOrder(menu)
  let food = cook.prepareFood(order)
  let tip = customer.eatAndPay(food)
  return tip
}
```

Asynchronous Programming

```
function serveCustomer(customer, done) {
  customer.placeOrder(menu, (error, order) => {
    cook.prepareFood(order, (error, food) => {
      customer.eatAndPay(food, done)
    }
  }
}
```

https://github.com/pofallon/nodejs-big-picture/tree/master/callbacks

Callbacks: the Christmas Tree Problem

```
function serveCustomer(customer, done) {
  customer.placeOrder(menu, (error, order) => {
    cook.prepareFood(order, (error, food) => {
      customer.eatAndPay(food, done)
    }
  }
}
```

Callbacks: the Christmas Tree Problem

```
function serveCustomer(customer, done) {
  customer.placeOrder(menu, (error, order) => {
    cook.prepareFood(order, (error, food) => {
      customer.eatAndPay(food, done)
    }
  }
}
```

Promises & Async/Await to the Rescue

```
function serveCustomer(customer) {
  return customer.placeOrder(menu)
    .then(order => cook.prepareFood(order))
    .then(food => customer.eatAndPay(food))
}
```

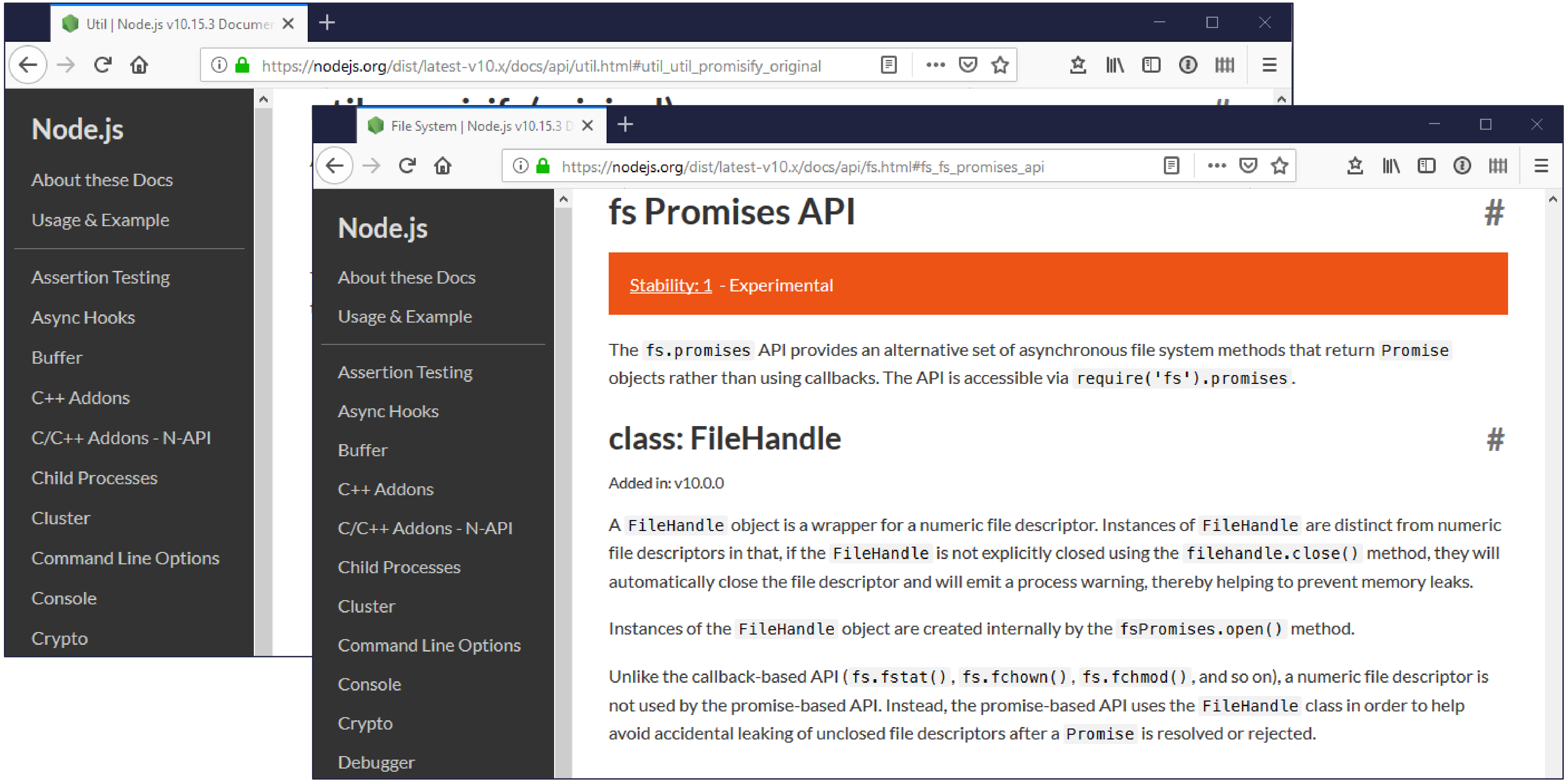
https://github.com/pofallon/nodejs-big-picture/tree/master/promises

Promises & Async/Await to the Rescue

```
const serveCustomer = async (customer) => {
  let order = await customer.placeOrder(menu)
  let food = await cook.prepareFood(order)
  let tip = await customer.eatAndPay(food)
  return tip
}
```

https://github.com/pofallon/nodejs-big-picture/tree/master/async

The Node.js Core APIs Are Evolving Too



fs Promises API

Stability: 1 - Experimental

The `fs.promises` API provides an alternative set of asynchronous file system methods that return `Promise` objects rather than using callbacks. The API is accessible via `require('fs').promises`.

class: FileHandle

Added in: v10.0.0

A `FileHandle` object is a wrapper for a numeric file descriptor. Instances of `FileHandle` are distinct from numeric file descriptors in that, if the `FileHandle` is not explicitly closed using the `filehandle.close()` method, they will automatically close the file descriptor and will emit a process warning, thereby helping to prevent memory leaks.

Instances of the `FileHandle` object are created internally by the `fsPromises.open()` method.

Unlike the callback-based API (`fs.fstat()`, `fs.fchown()`, `fs.fchmod()`, and so on), a numeric file descriptor is not used by the promise-based API. Instead, the promise-based API uses the `FileHandle` class in order to help avoid accidental leaking of unclosed file descriptors after a `Promise` is resolved or rejected.

EventEmitter

```
emitter.emit()

emitter.on()
```

https://nodejs.org/dist/latest-v10.x/docs/api/events.html

EventEmitter

```
emitter.emit(‘data’, ‘Hello World!’)
```

```
emitter.emit()

 emitter.on()
```

```
emitter.on(‘data’, (msg) => {
  console.log(msg)
})
```

https://nodejs.org/dist/latest-v10.x/docs/api/events.html

Serving Customers with Events

```
const serveCustomer = (customer, done) => {
  customer.on('decided', order => {
    order.on('prepared', food => customer.eatAndPay(food))
    cook.prepareFood(order)
  })
  customer.on('leaving', tip => done(null, tip))
  customer.placeOrder(menu)
}
```

https://github.com/pofallon/nodejs-big-picture/tree/master/events

Streams

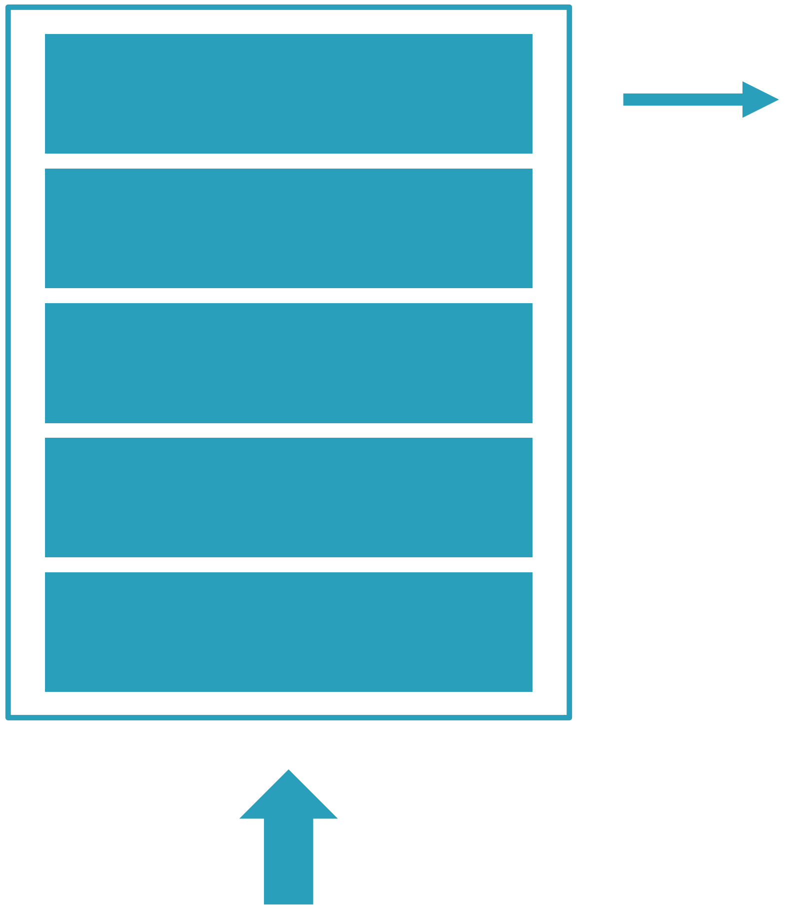
Readable Streams

Read/Write Streams
(Duplex, Transform)

Writable Streams

https://nodejs.org/dist/latest-v10.x/docs/api/stream.html

Readable Streams



Events: readable, data, end, error

Methods: read, pause, resume, destroy

Properties: readable, readableLength

(these are just a sample … see the docs for more)

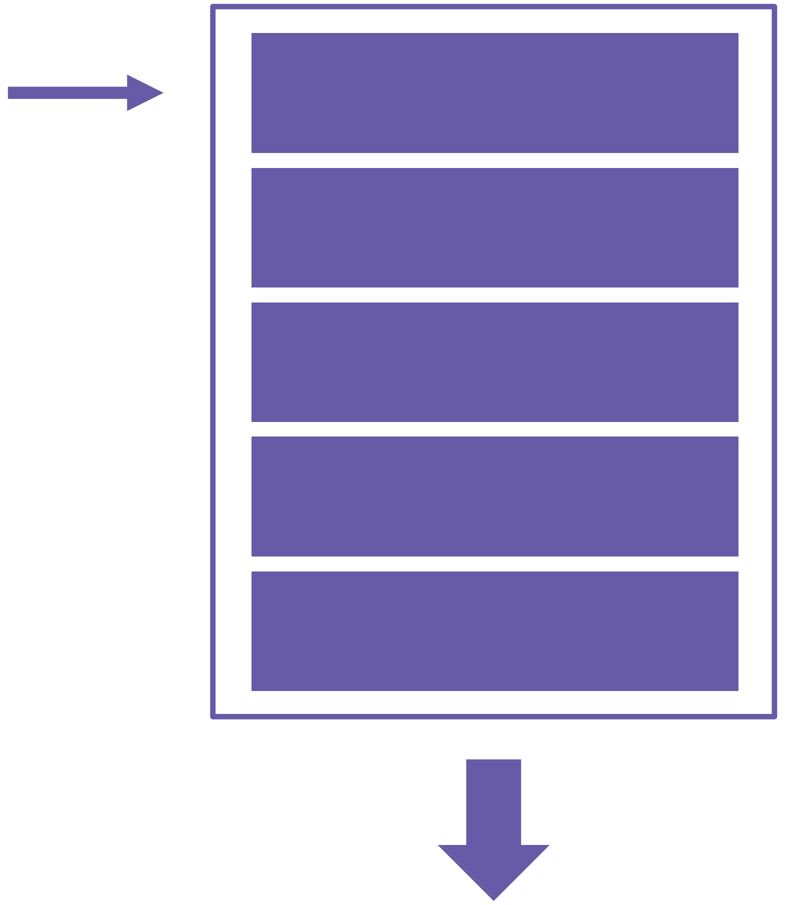
Writable Streams

Events: drain, close, finish, error

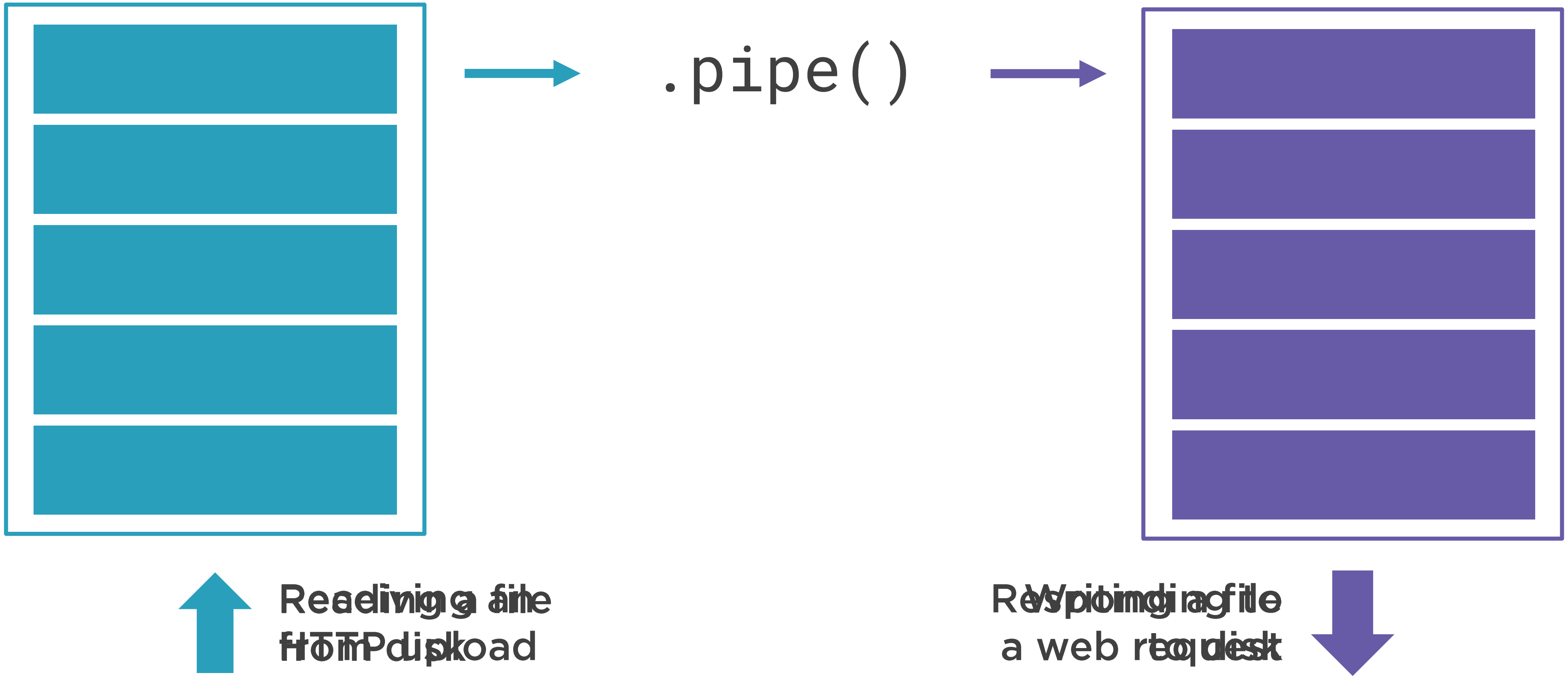
Methods: write, destroy, end

Properties: writable, writableLength

(these are just a sample … see the docs for more)



Piping Streams



Streams in the Node.js API

fs.ReadStream

fs.WriteStream

http.ClientRequest

http.IncomingMessage

http.ServerResponse

zlib.createGzip()

Example Streams Use Case

```
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/plain’);
  res.setHeader('Content-Encoding', 'gzip’)
  fs.createReadStream(path.join(__dirname, 'lorem.txt’))
    .pipe(zlib.createGzip())
    .pipe(res)
})
```

https://github.com/pofallon/nodejs-big-picture/tree/master/streams