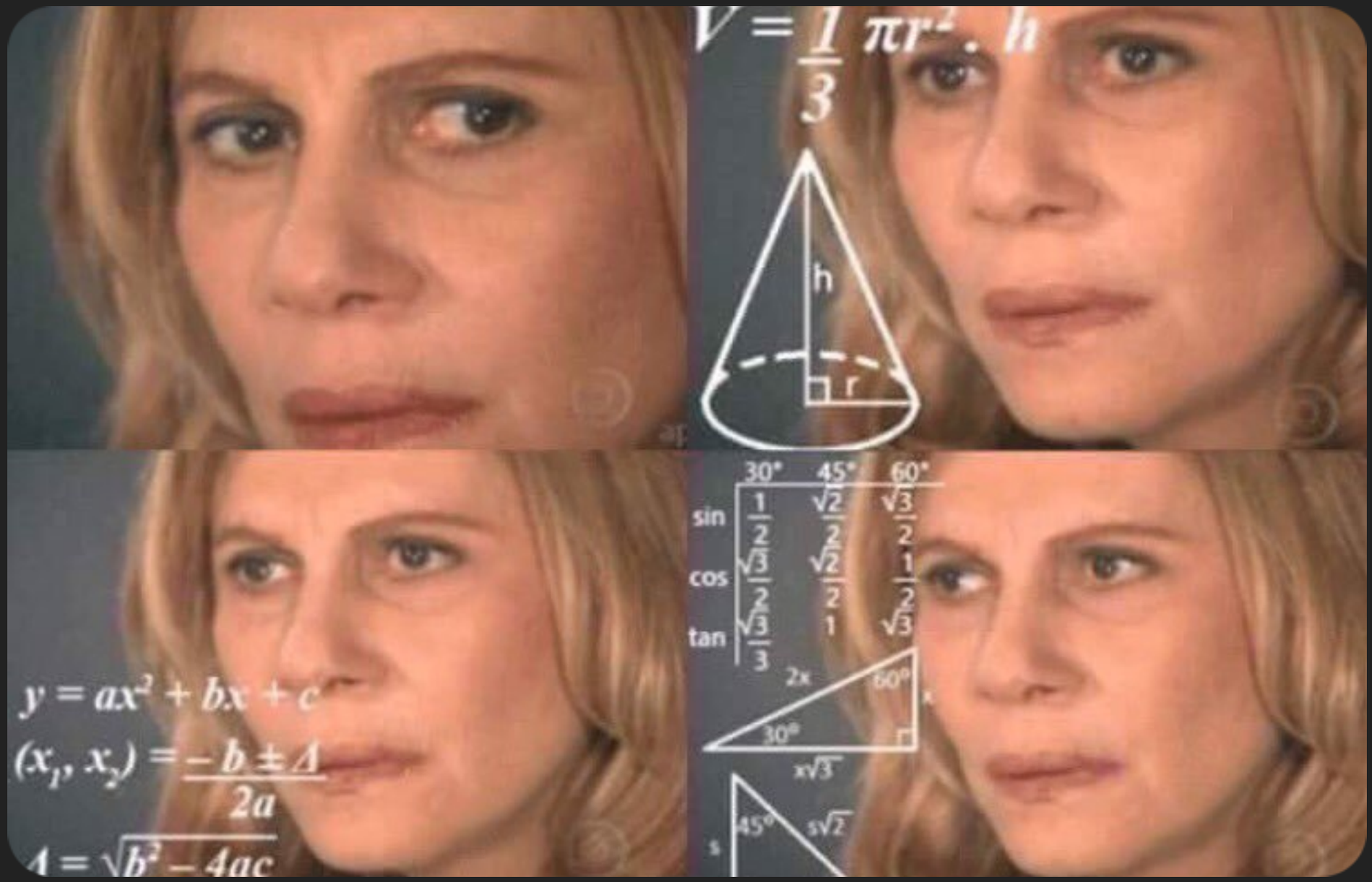
ASYNCHRONY

UNDER THE HOOD

Shelley Vohr (@codebytere)

BASIC ASYNC

@codebytere



RUN-TO-COMPLETION

```
setTimeout(() => {
  console.log('second')
}, 0)
console.log('first')
```

STACK

```
function baz(z) {
  console.log(new Error().stack)
}

Error
    at baz (stack_trace.js:2:17)
    at bar (stack_trace.js:6:5)
    at foo (stack_trace.js:9:5)
    at <global> (stack_trace.js:11:1)

foo(3)
return
```

Initially ⟶ Empty

After foo(3) ⟶

Location in global scope

After bar(x + 1) ⟶

Location in foo()

Location in global scope

After baz(y + 1) ⟶

Location in bar()

Location in foo()

Location in global scope

After console.log() ⟶

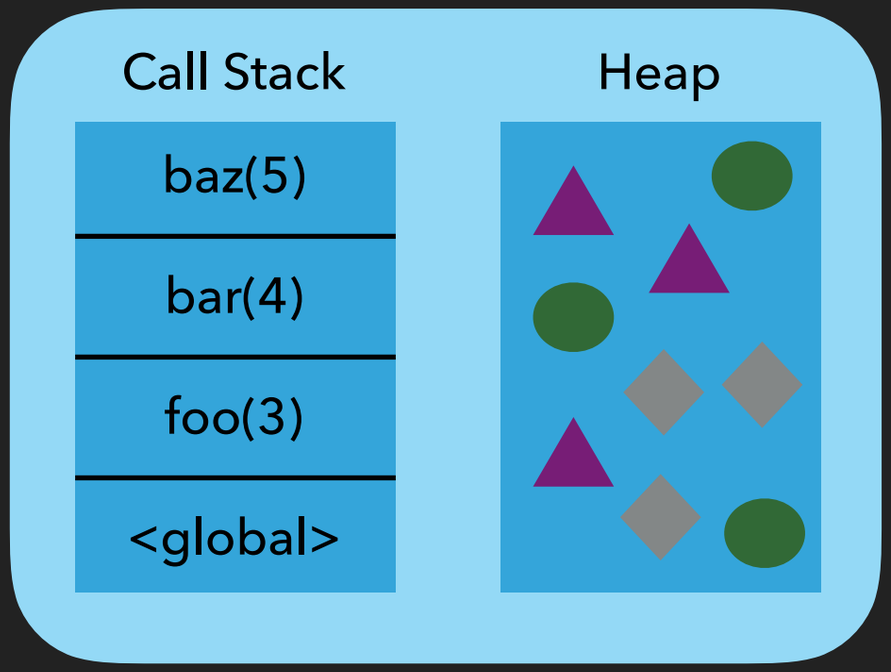
Location in baz()

Location in bar()

Location in foo()

Location in global scope

@codebytere



Task Sources

Current Task

Task Queue

Task 1

Task 2

Task 3

Event Loop

CALLBACKS

@codebytere

```
doA(() => {
  doB()
  doC(() => {
    doD()
  })
  doE()
})
doF()
```

```
first(() => {
  third()
  fourth(() => {
    sixth()
  })
  fifth()
})
second()
```

ERRORS

@codebytere

```
function doSomething((err, file) ⇒ {
  if (err) {
    console.error(`oh no! ${err}`)
    return
  }
  // continue with other things
})
```

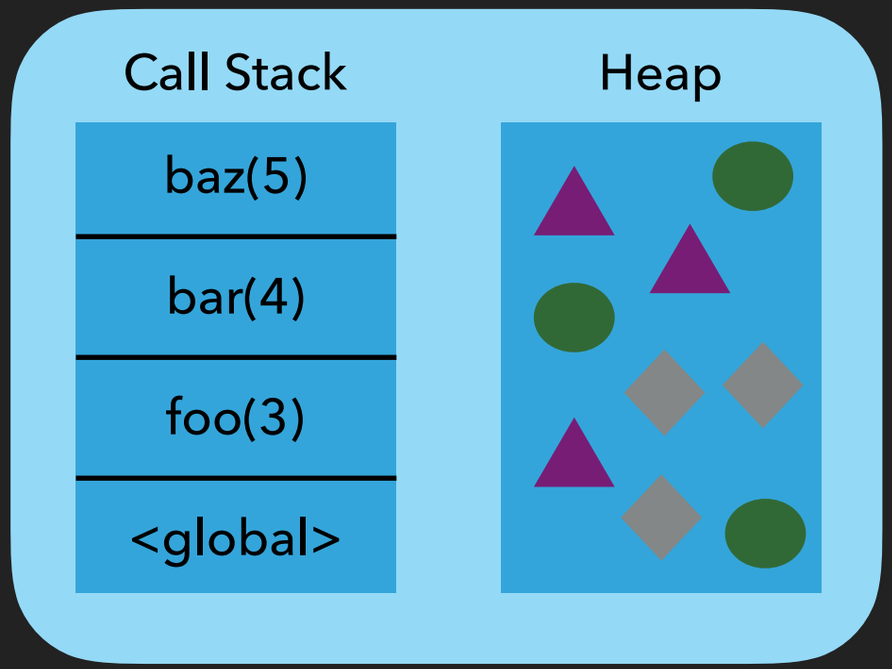
```
function doSomething((err, file) ⇒ {
  if (err) {
    throw new Error(`oh no! ${err}`)
  }
  // continue with other things
})
```

PROMISES

@codebytere

```
const p = Promise.resolve('hello')

p.then(val ⇒ {
  console.log(val)
  return `${val} world`
}).then(newVal ⇒ {
  console.log(newVal)
})
```

@codebytere


Call Stack
Heap
baz(5)
bar(4)
foo(3)
<global>

Task Sources
Task Queue
Current Task
Task 1
Task 2
Task 3
Microtask Queue
Microtask 1
Microtask 2
Microtask 3

ERRORS

@codebytere

```
const prom = new Promise((resolve, reject) => {
  if (true) {
    // throw new Error('rejected!')
    reject(new Error('rejected!'))
  } else {
    resolve('success!')
  }
})
```

```
prom.then(val => `${val} we did it!`)
  .then(val => console.log( `got ${val}`))
  .catch(err => {
    console.log(`error: ${error.message}`)
    console.log(`error stack: ${error.stack}`)
  })
```

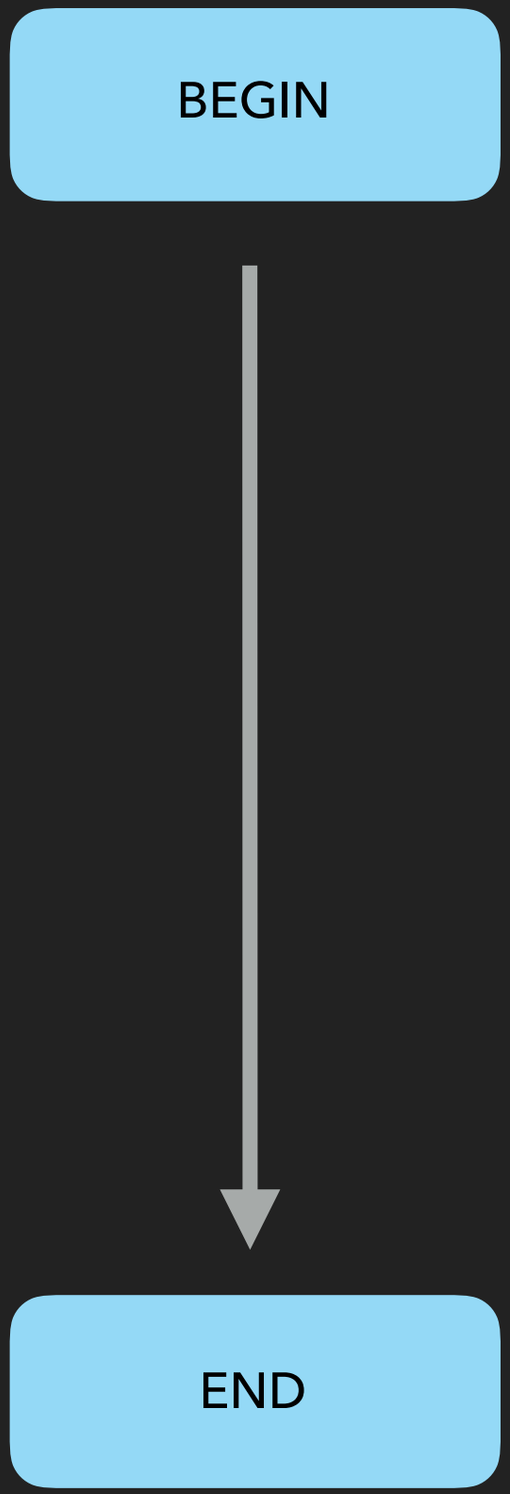
```
prom.then(val => `${val} we did it!`)
  .then(val => console.log( `got ${val}`))
```

@codebytere

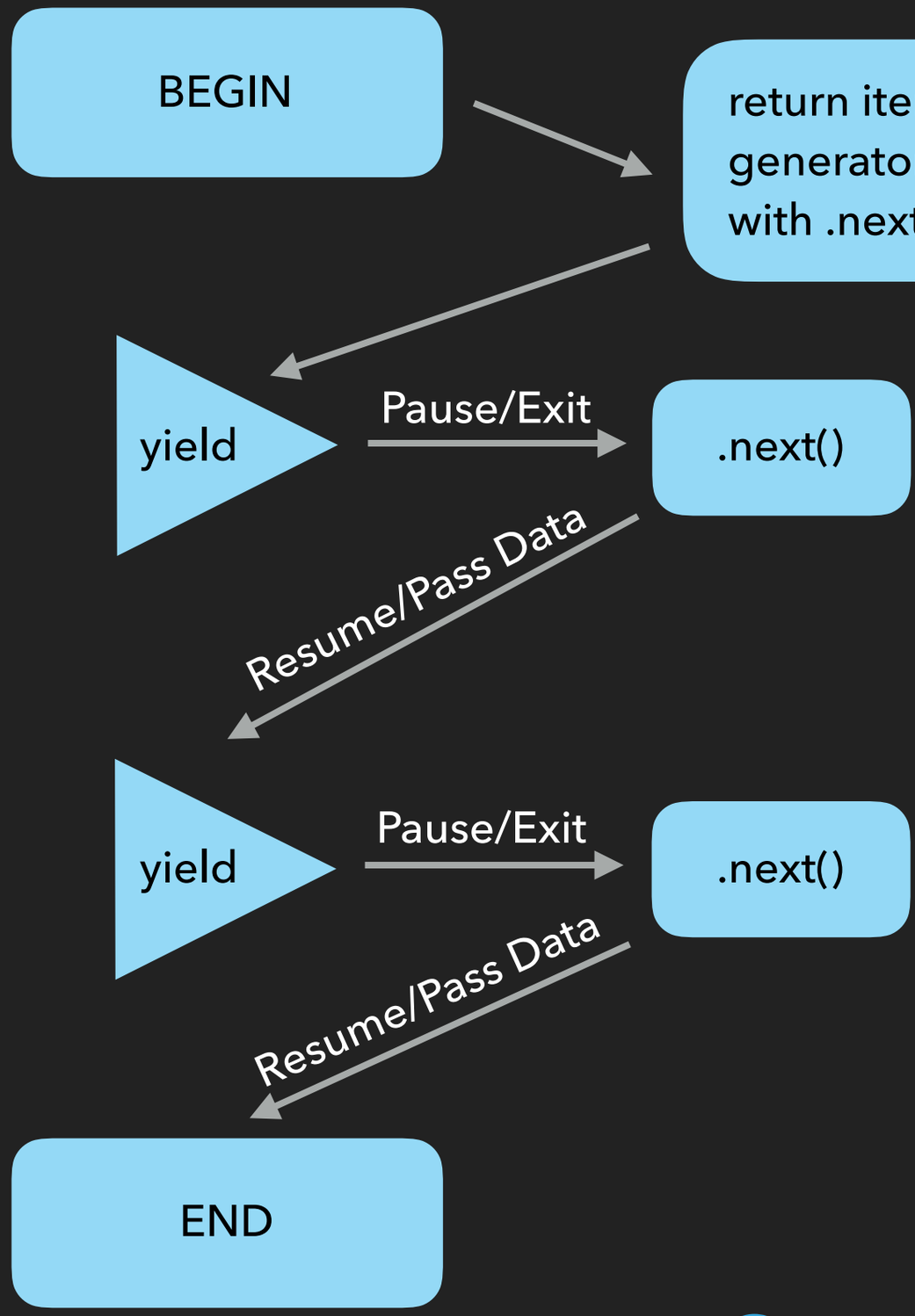
GENERATORS

```
function* counter() {
  let index = 0
  while(true) {
    yield index++
  }
}

const gen = counter()

console.log(gen.next().value) // 0
console.log(gen.next().value) // 1
console.log(gen.next().value) // 2
```

FUNCTION



GENERATOR



@codebytere

ERRORS

```
const it = foo()
const res = it.next()
it.throw("ERROR!")
```

```
function *foo() {
  try {
    const x = yield 3
    console.log(`x: ${x}`)
  }
  catch (err) {
    console.log(`Error: ${err}`)
  }
}
```

ASYNC/AWAIT

@codebytere

```
async function getAddress () {
  let [
    streetAddress,
```

```
async function getAddress() {
  const streetAddress = await getStreetAddress()
  const city = await getCity()
  const state = await getState()
  const zip = await getZipCode()

  return `${streetAddress}, ${city} ${state} ${zip}`
}
```

```
  return `${streetAddress}, ${city}, ${state}, ${zip}`
}
```

ERRORS

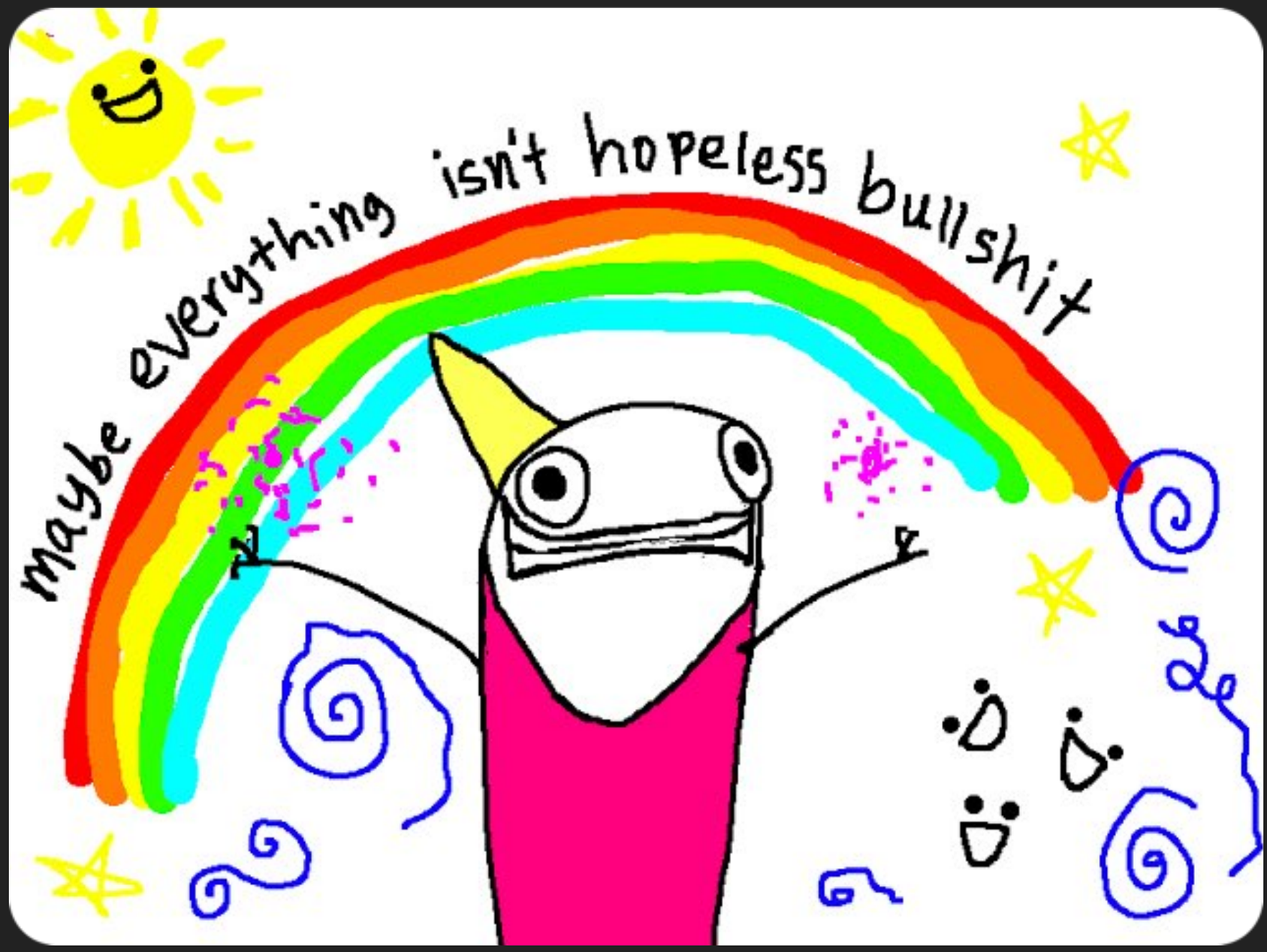
@codebytere

```
const asyncFunction = async () => {
  try {
   doSynchronousThings()
   const data = await getSomeData()
   return data.map(item => item.doSomething())
  } catch(err){
    console.error(err)
  }
}
```

```
const asyncFunction = () => {
  try {
   doSynchronousThings()
   return getSomeData()
    .then(data => data.map(item => item.doSomething()))
    .catch(e => console.error(e))
  } catch(err) {
    console.error(err)
  }
}
```

WRAPPING UP

@codebytere



CALLBACK → PROMISE → ASYNC/AWAIT

```
getData(a ⇒ {
    getMoreData(a, b ⇒ {
        getMoreData(b, c ⇒ {
            getMoreData(c, d ⇒ {
                getMoreData(d, e ⇒ {
                    console.log(e);
                });
            });
        });
    });
});
```

GO FORTH AND ASYNC!

@codebytere