JavaScript

JavaScript

What are you?

I am

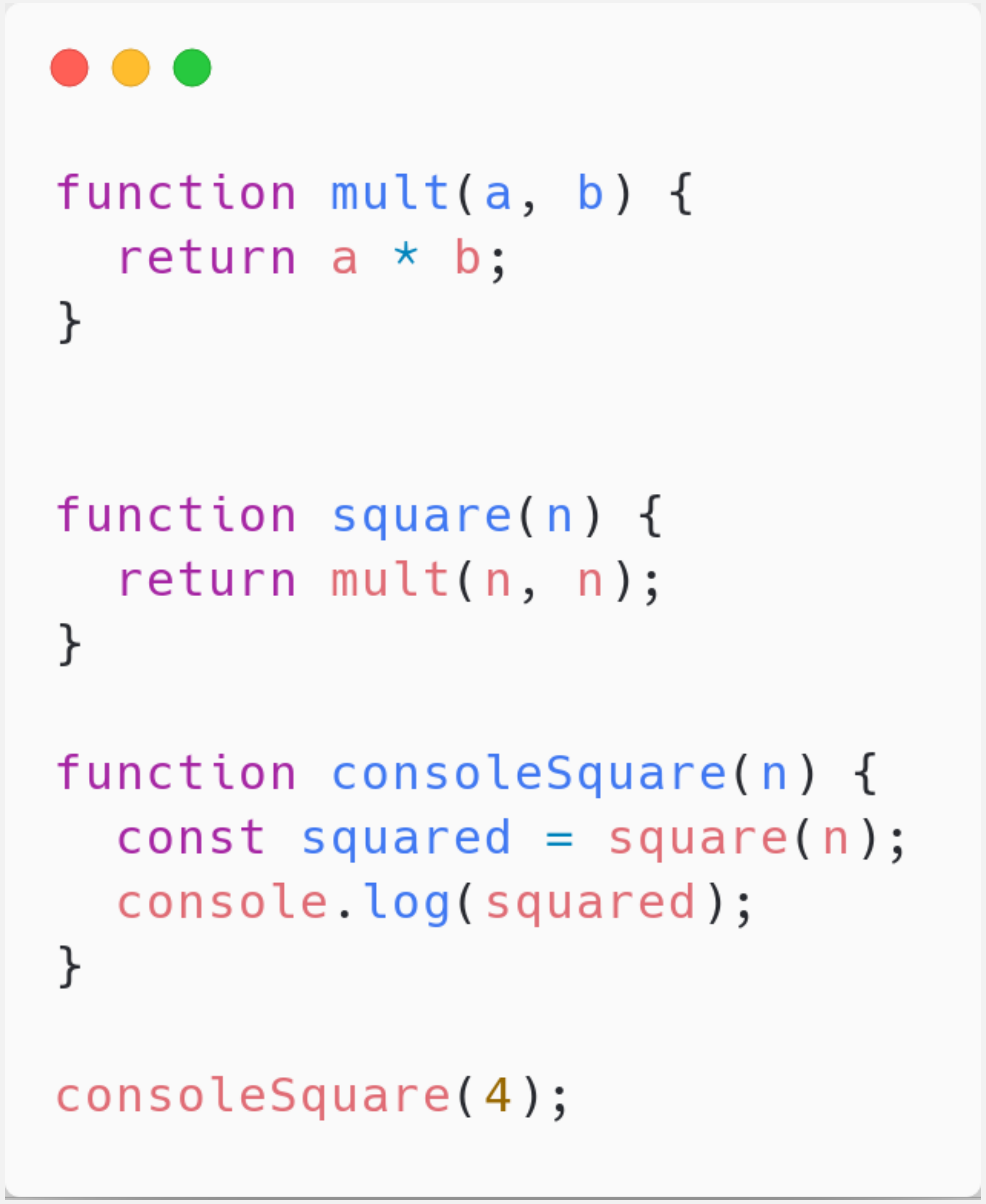
- Single threaded
- Non-blocking
- Asynchronous
- Concurrent

language

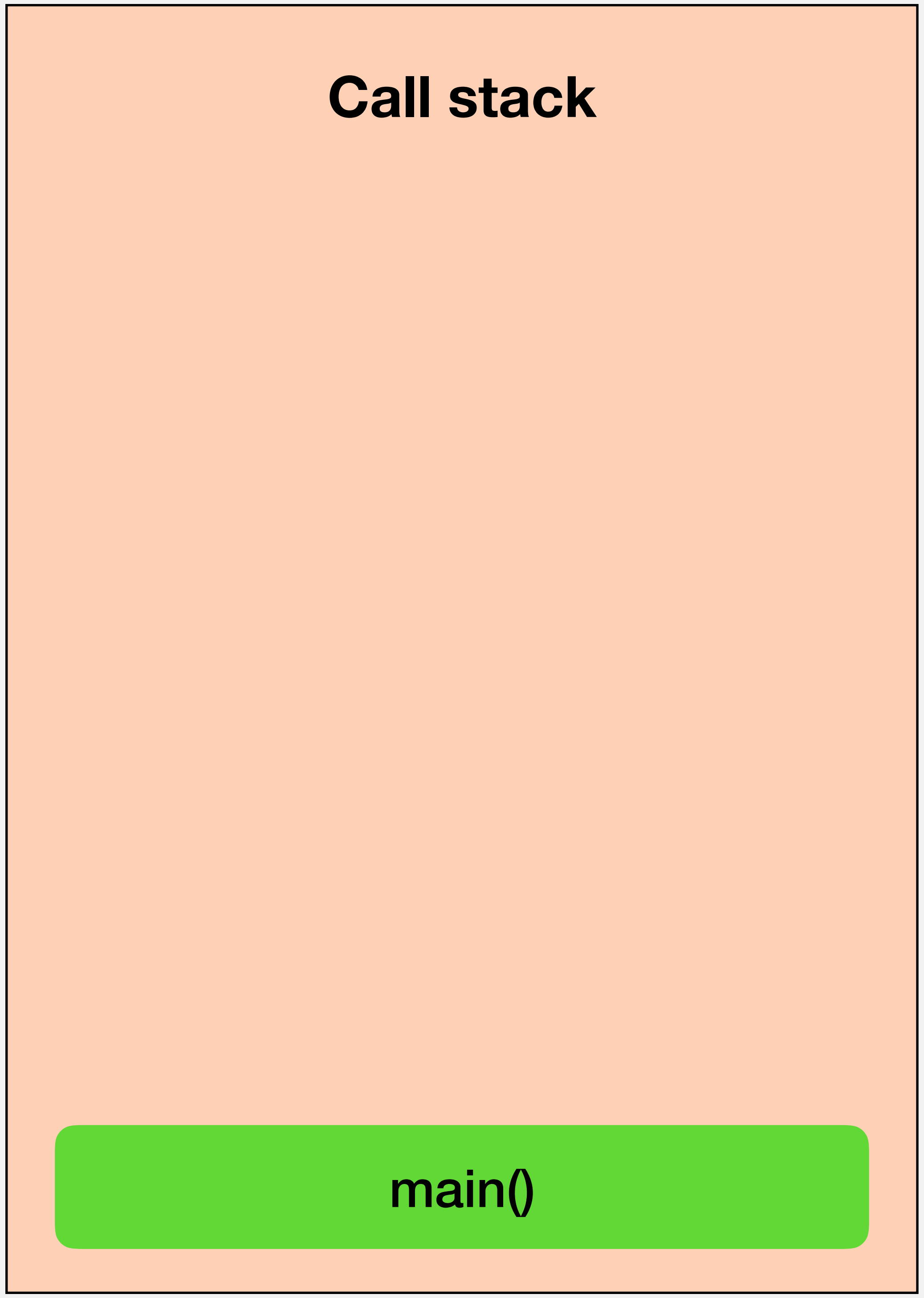
I have

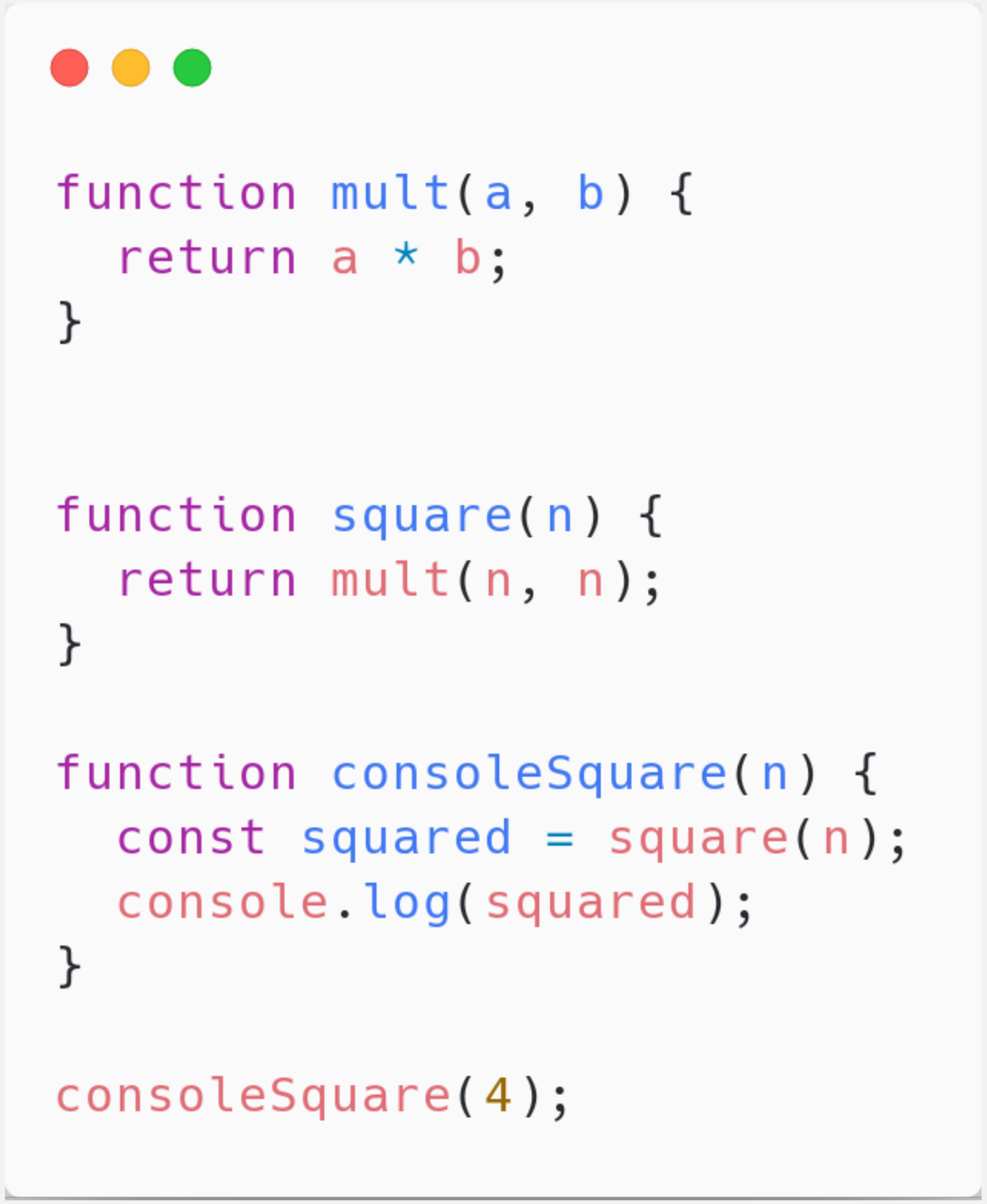
- a call stack
- an event loop
- a callback queue
- and other APIs

One thread ==
one call stack ==
one thing at time

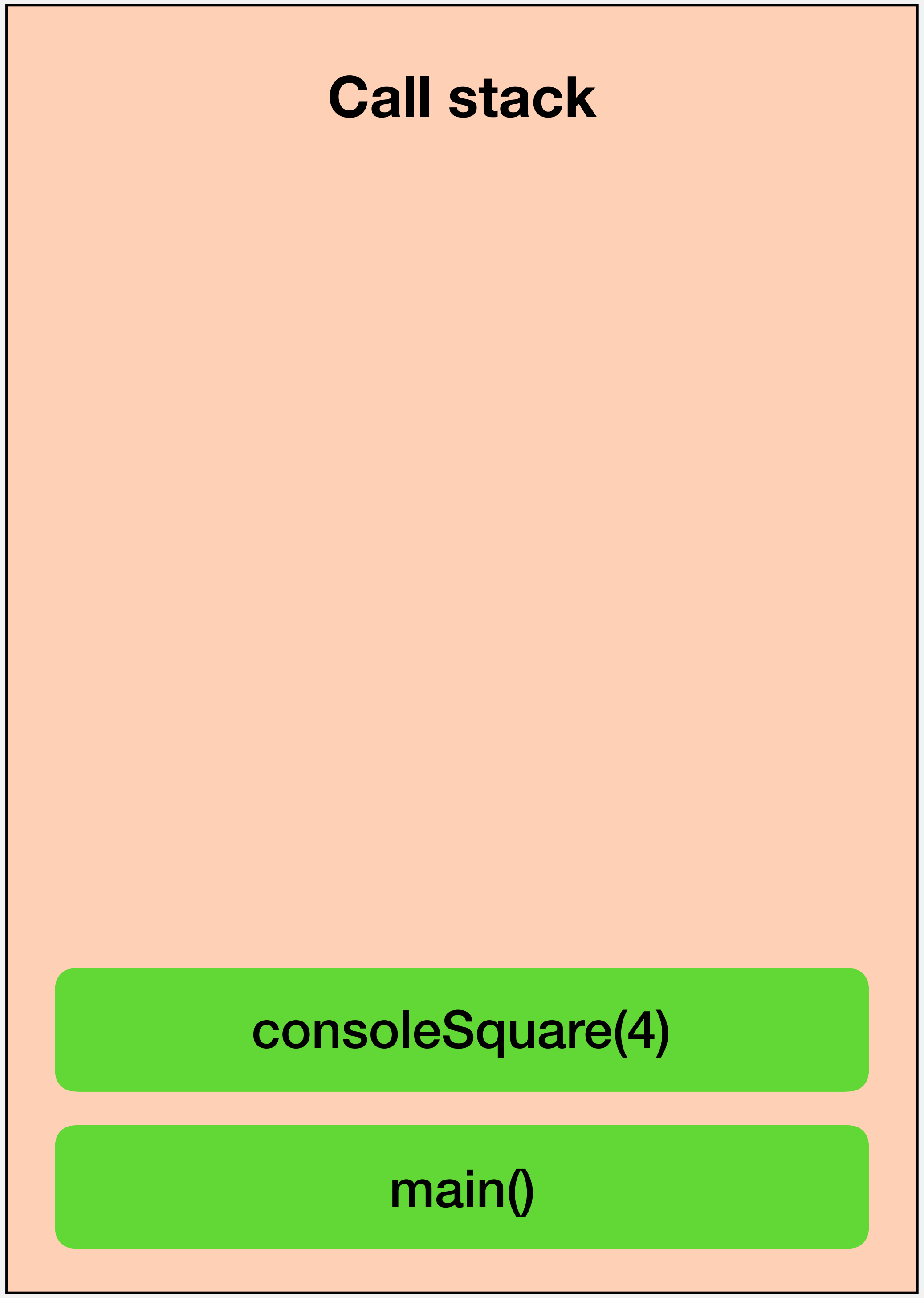


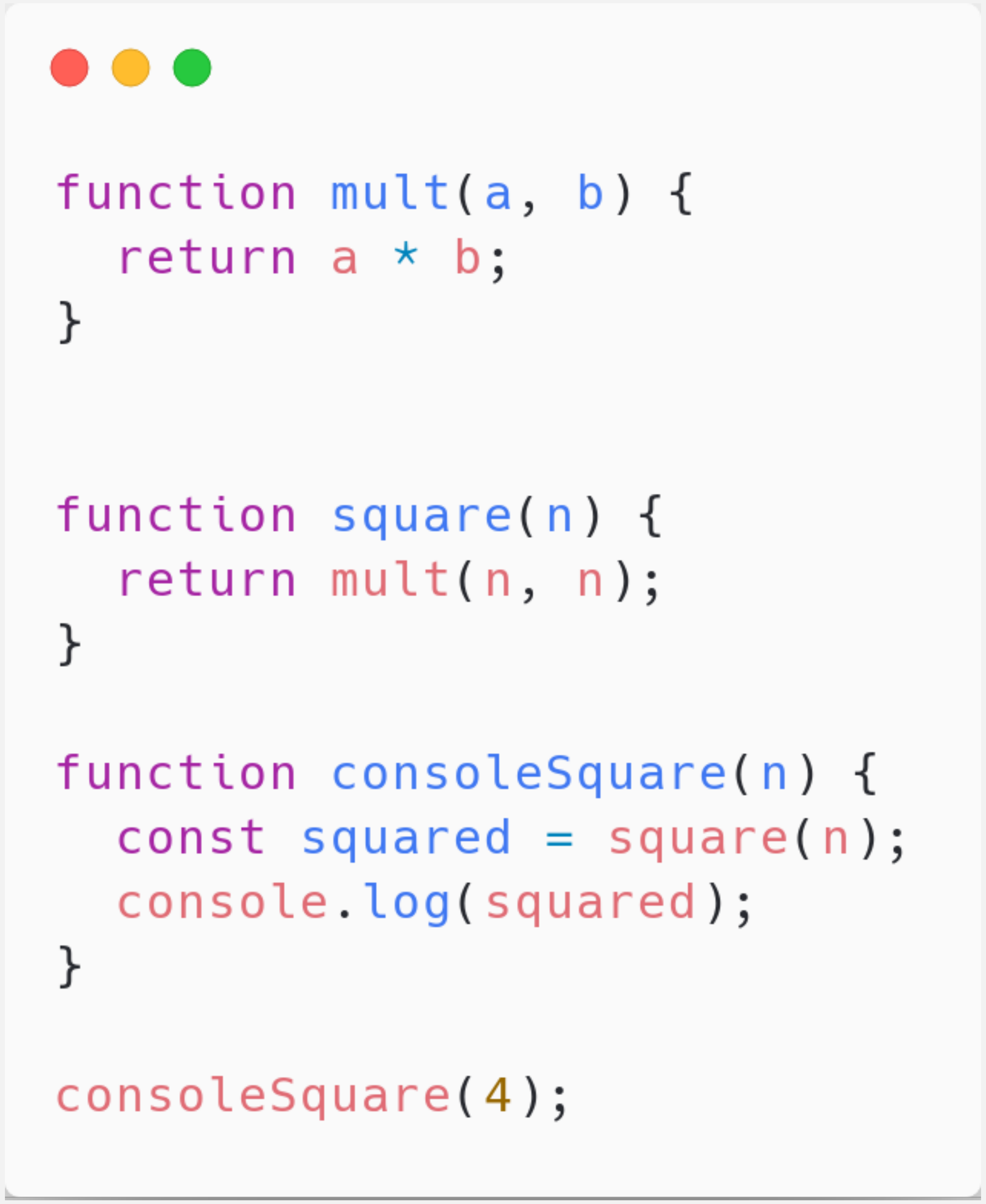
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



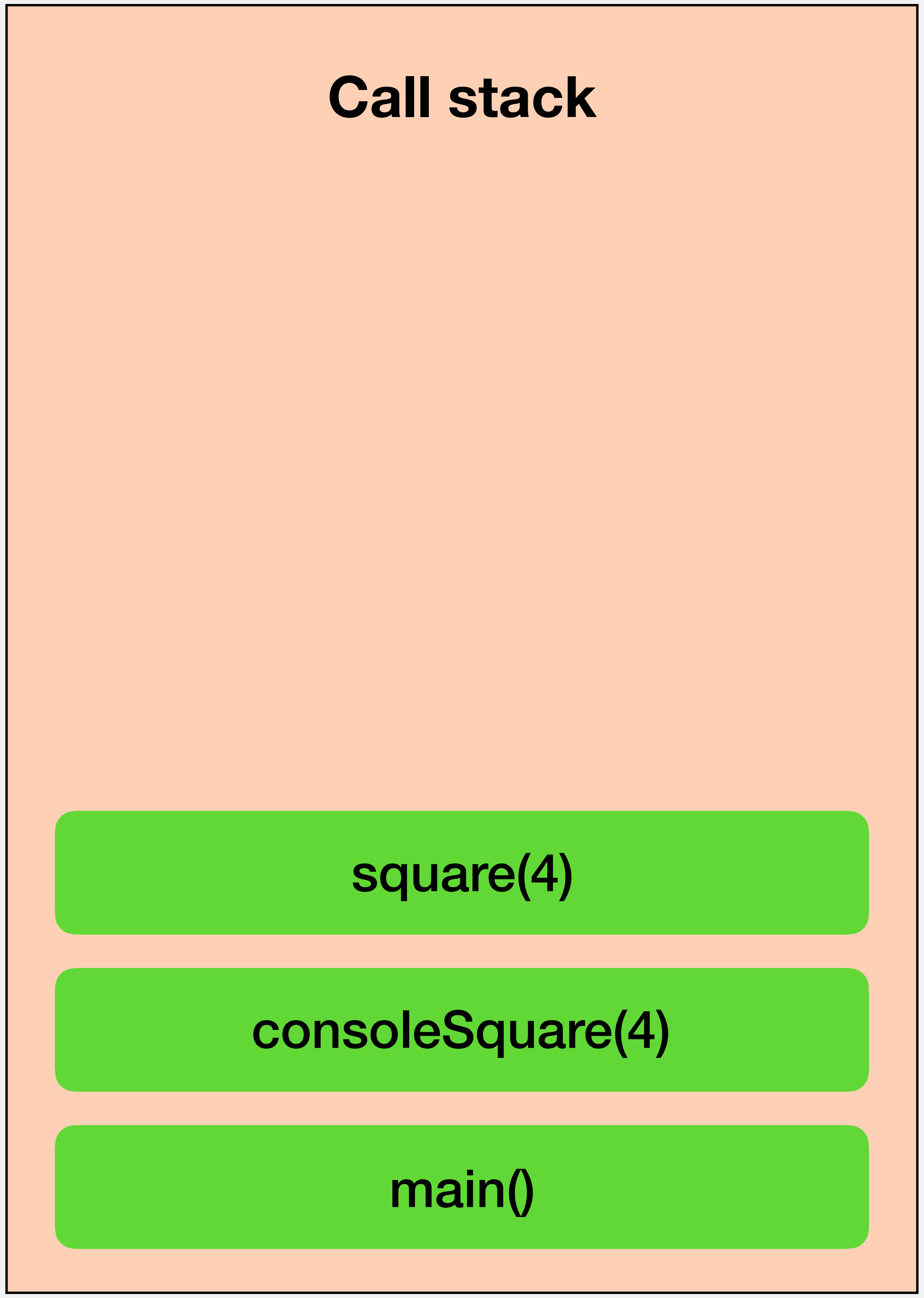


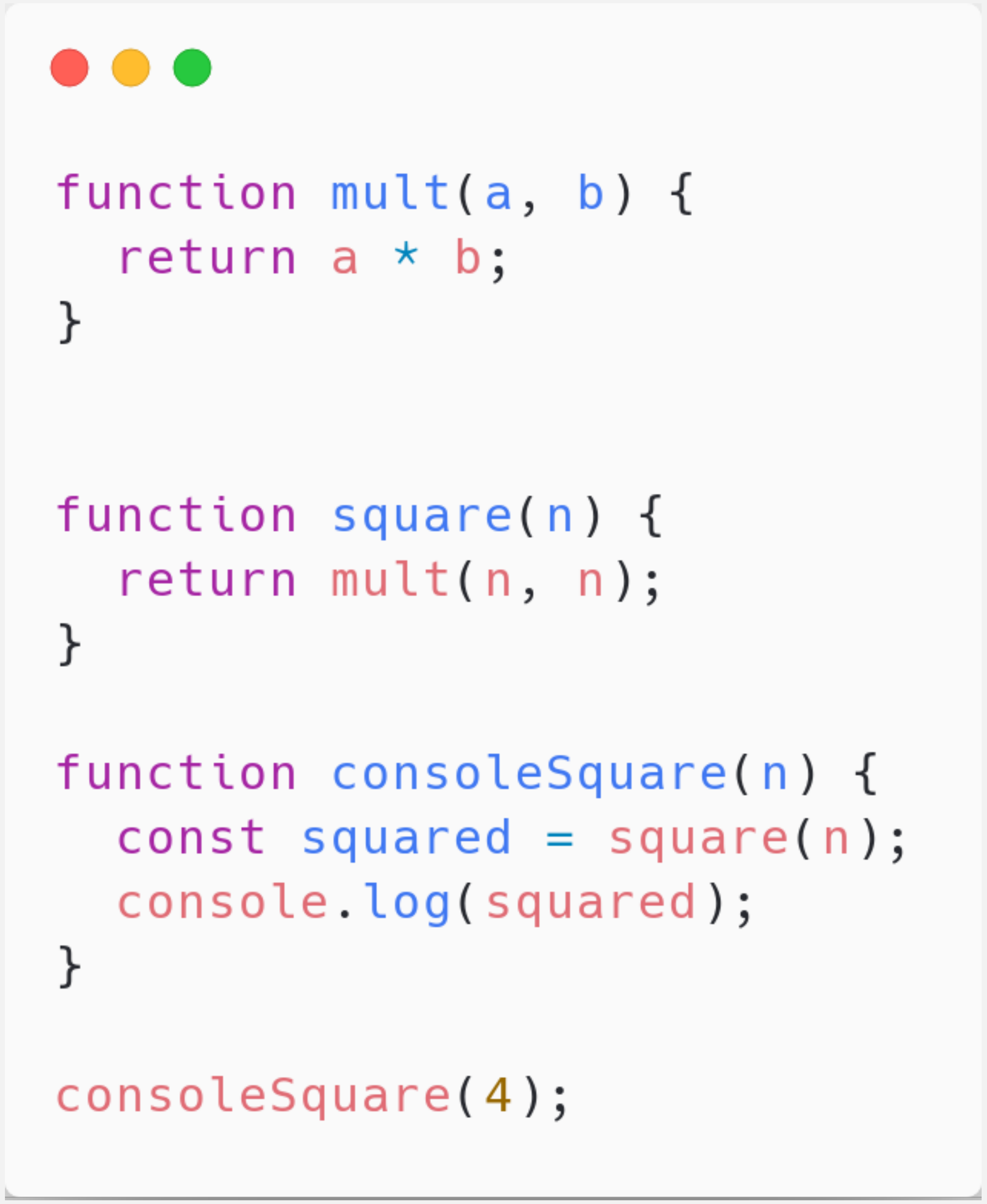
```javascript
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



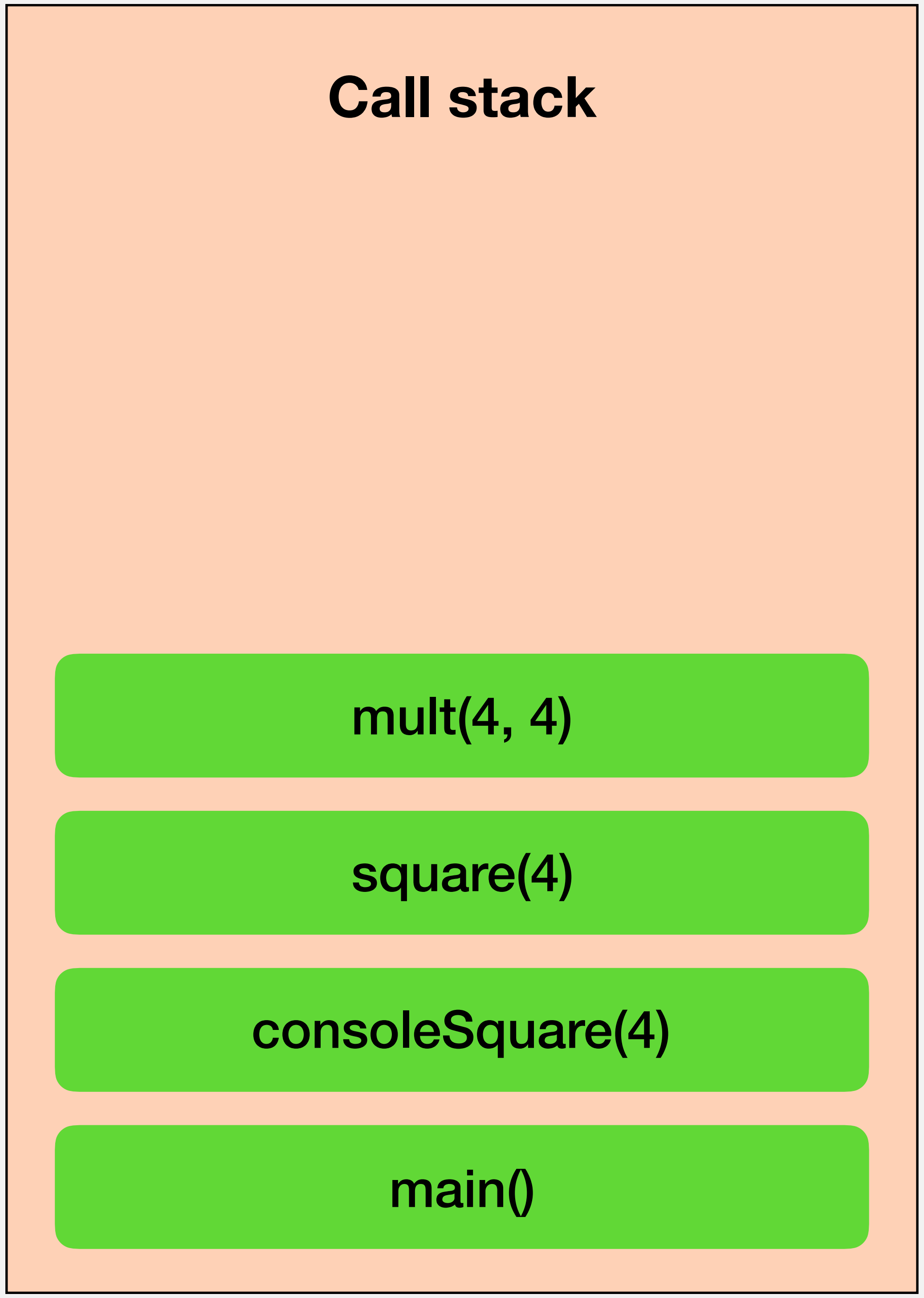


```javascript
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```

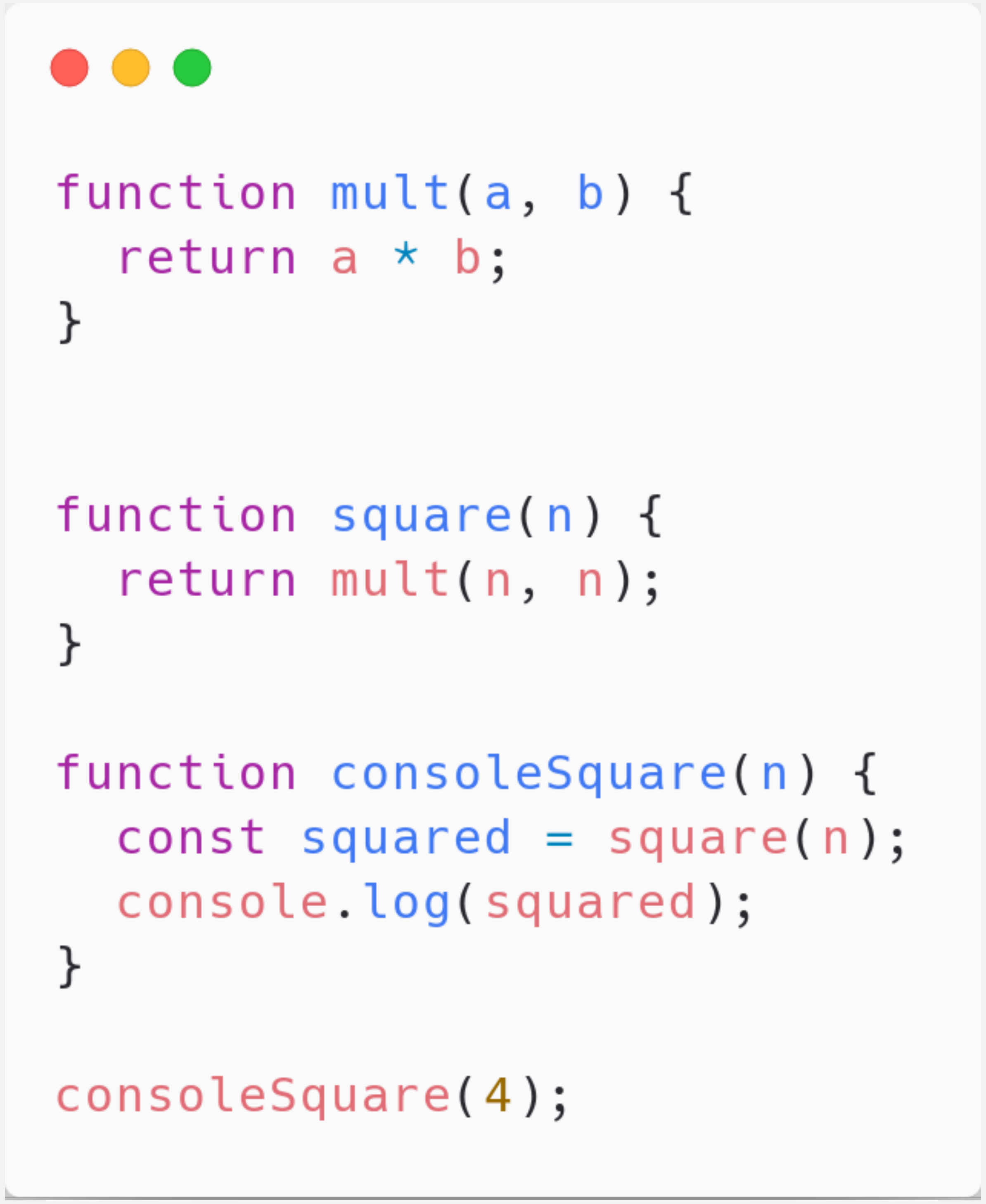




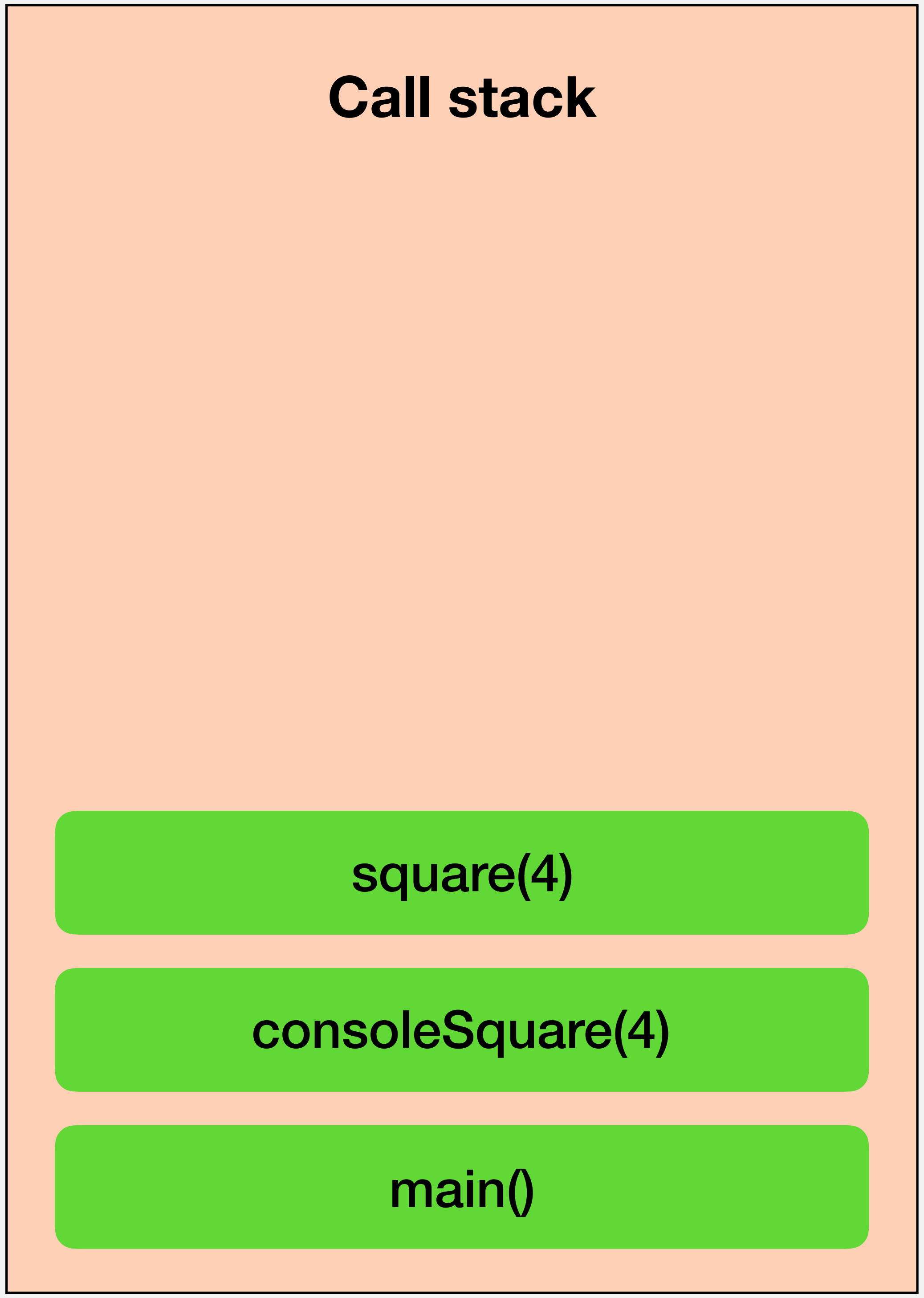
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```

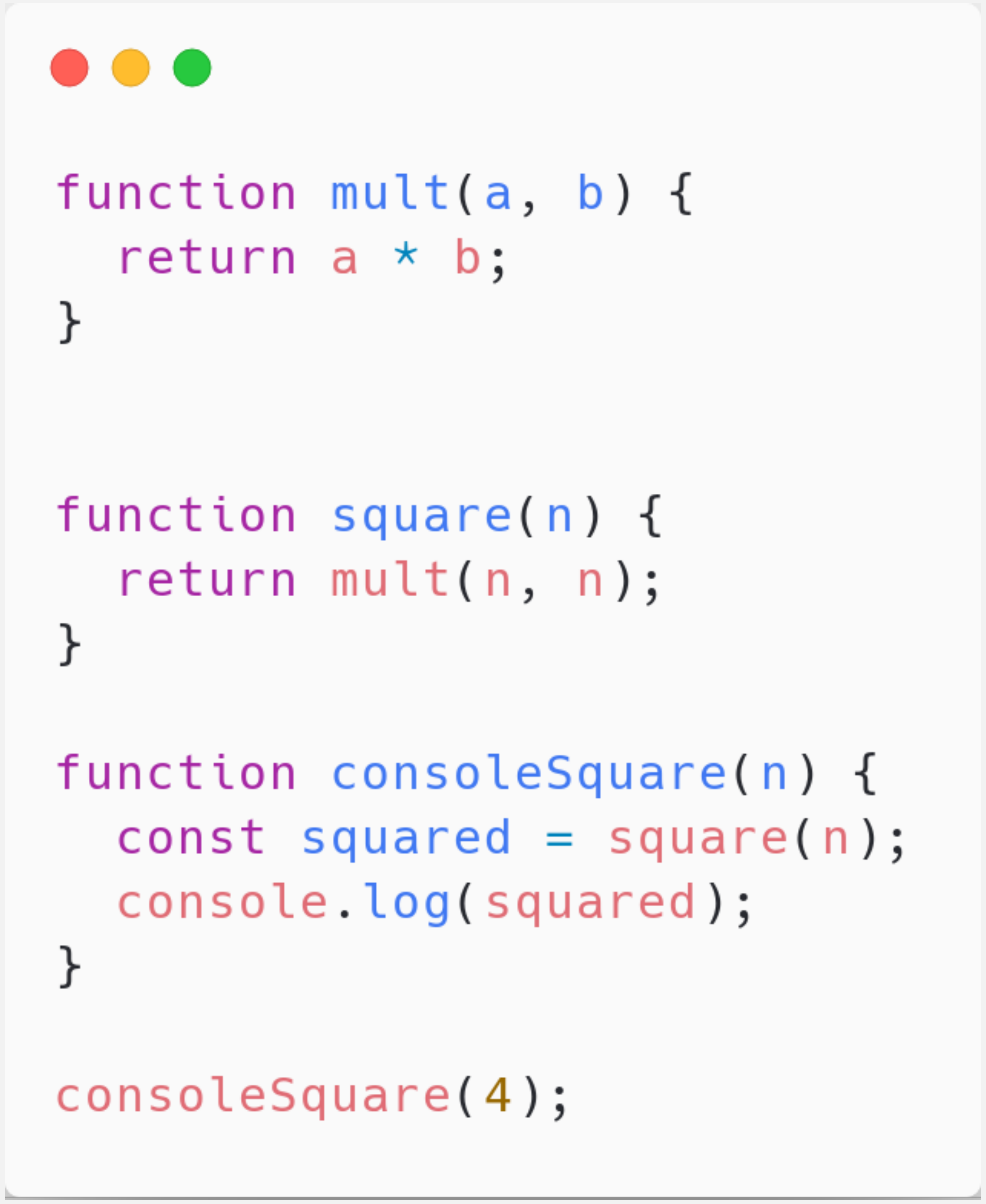


JS

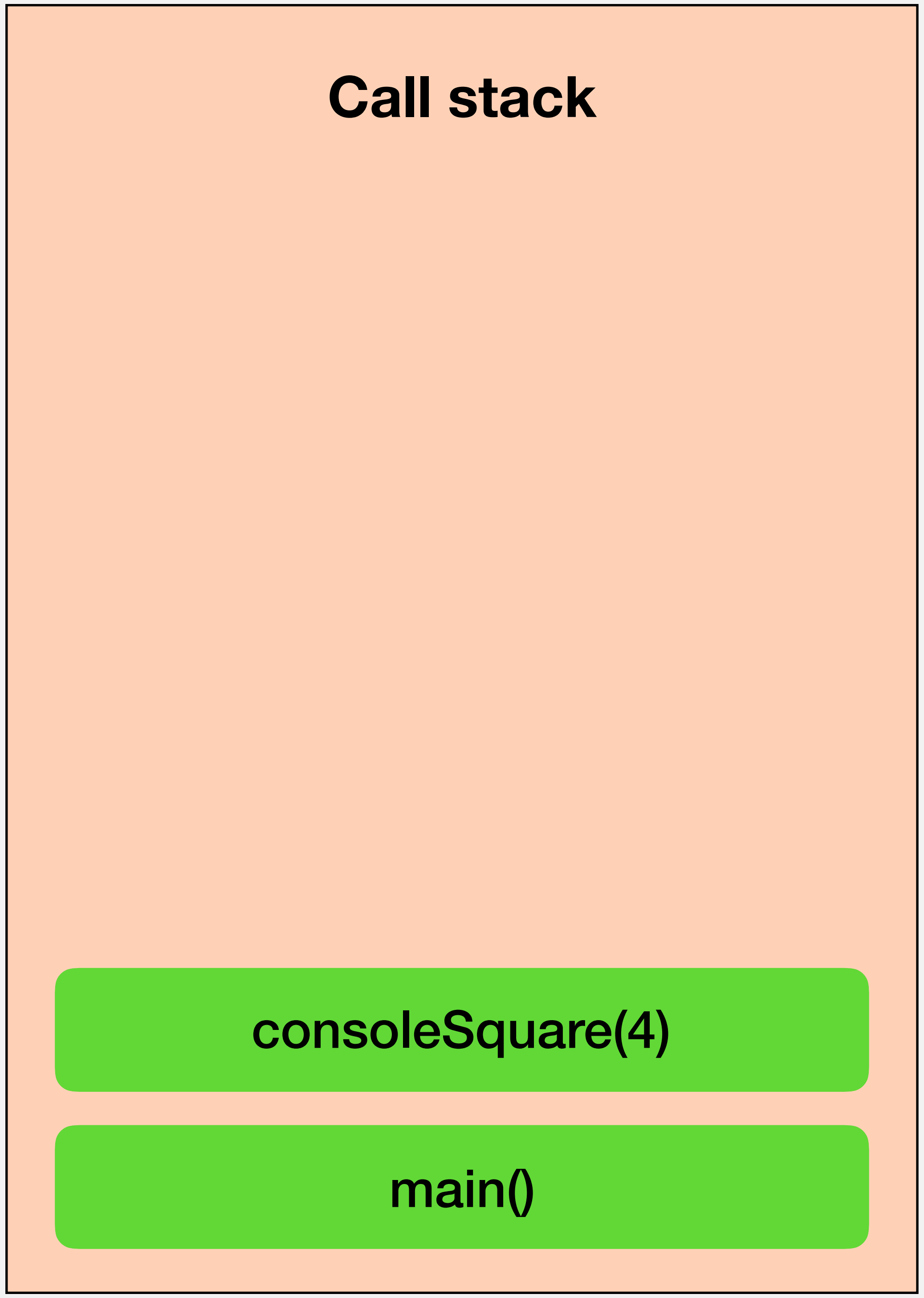


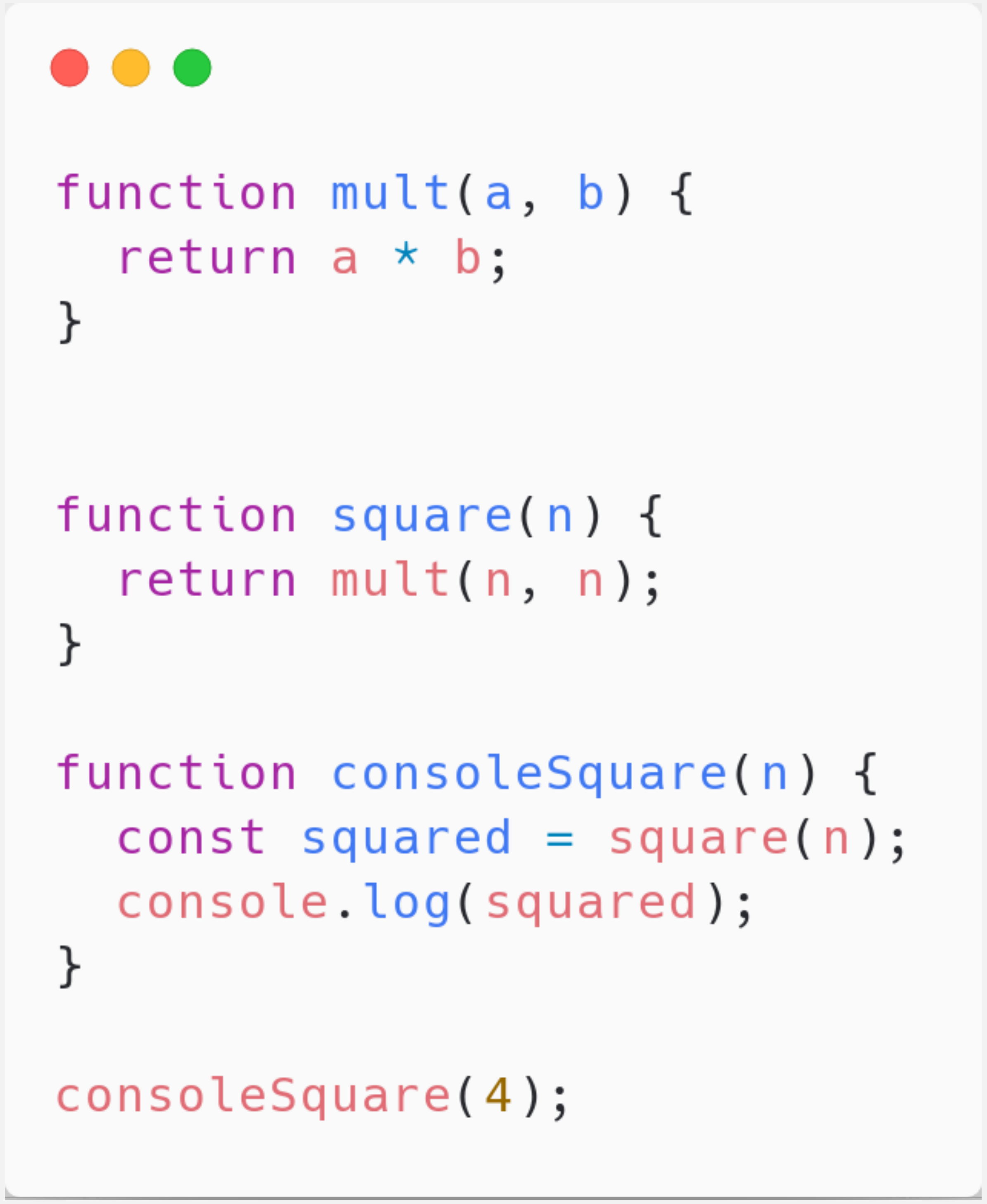
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



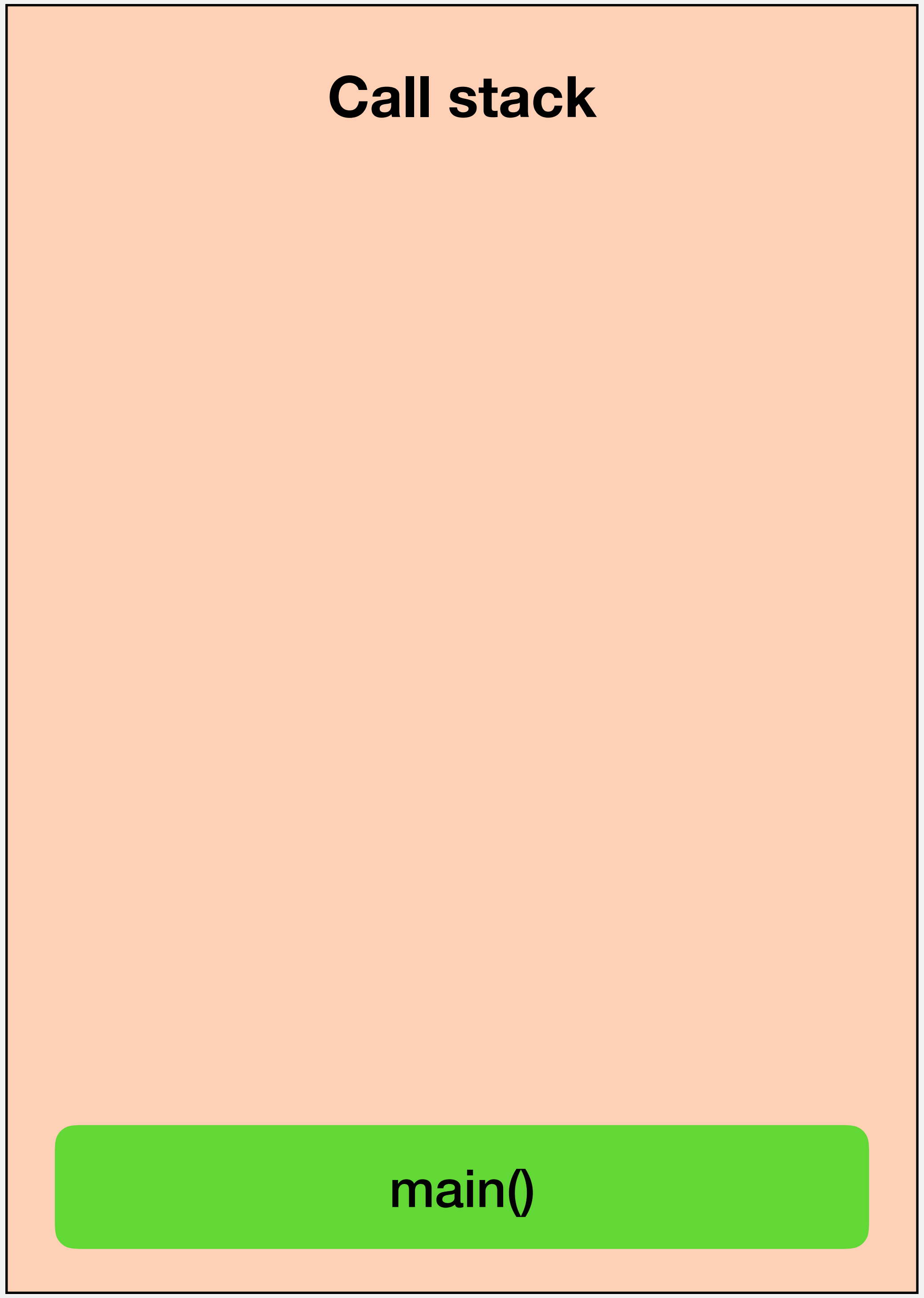


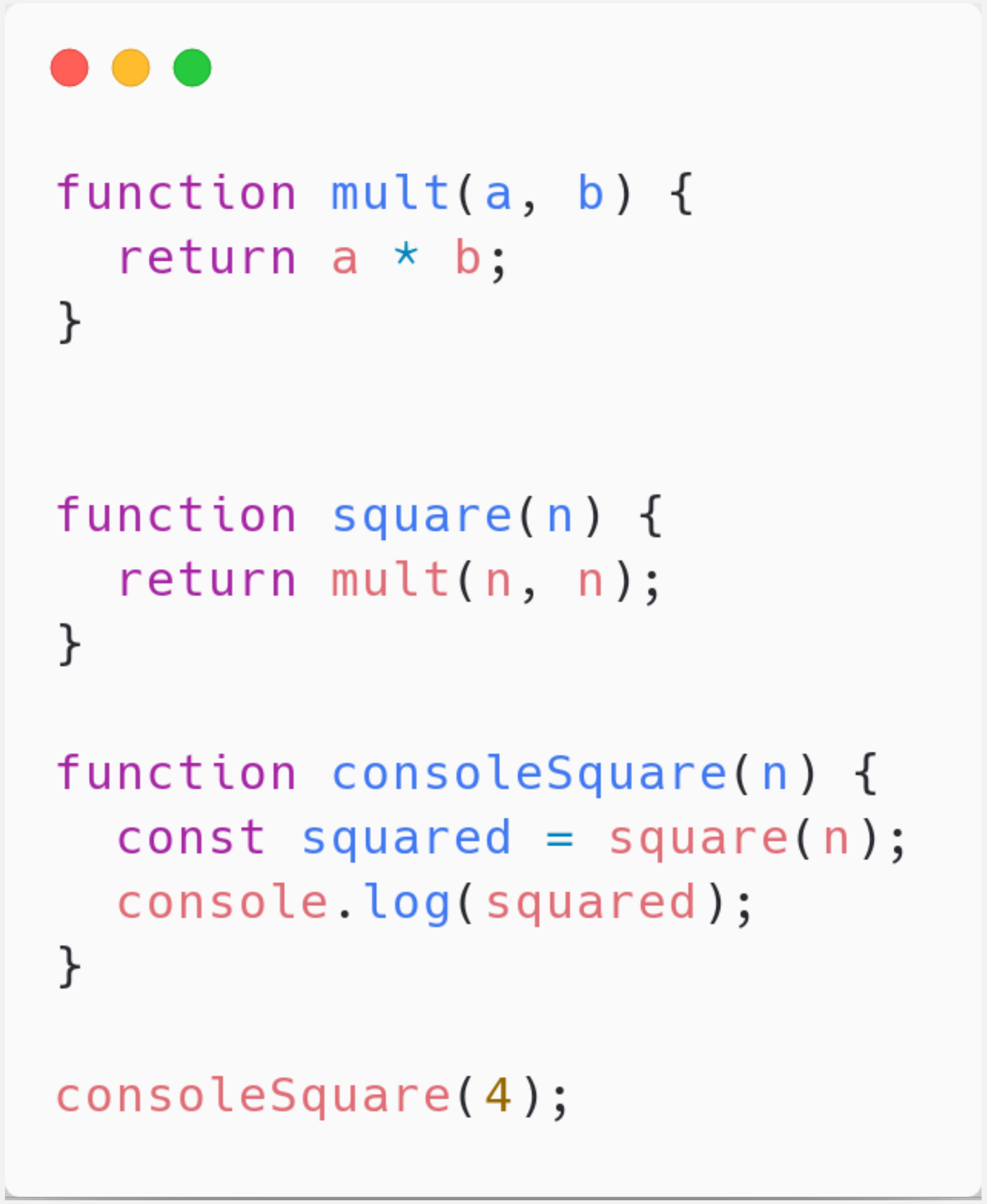
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



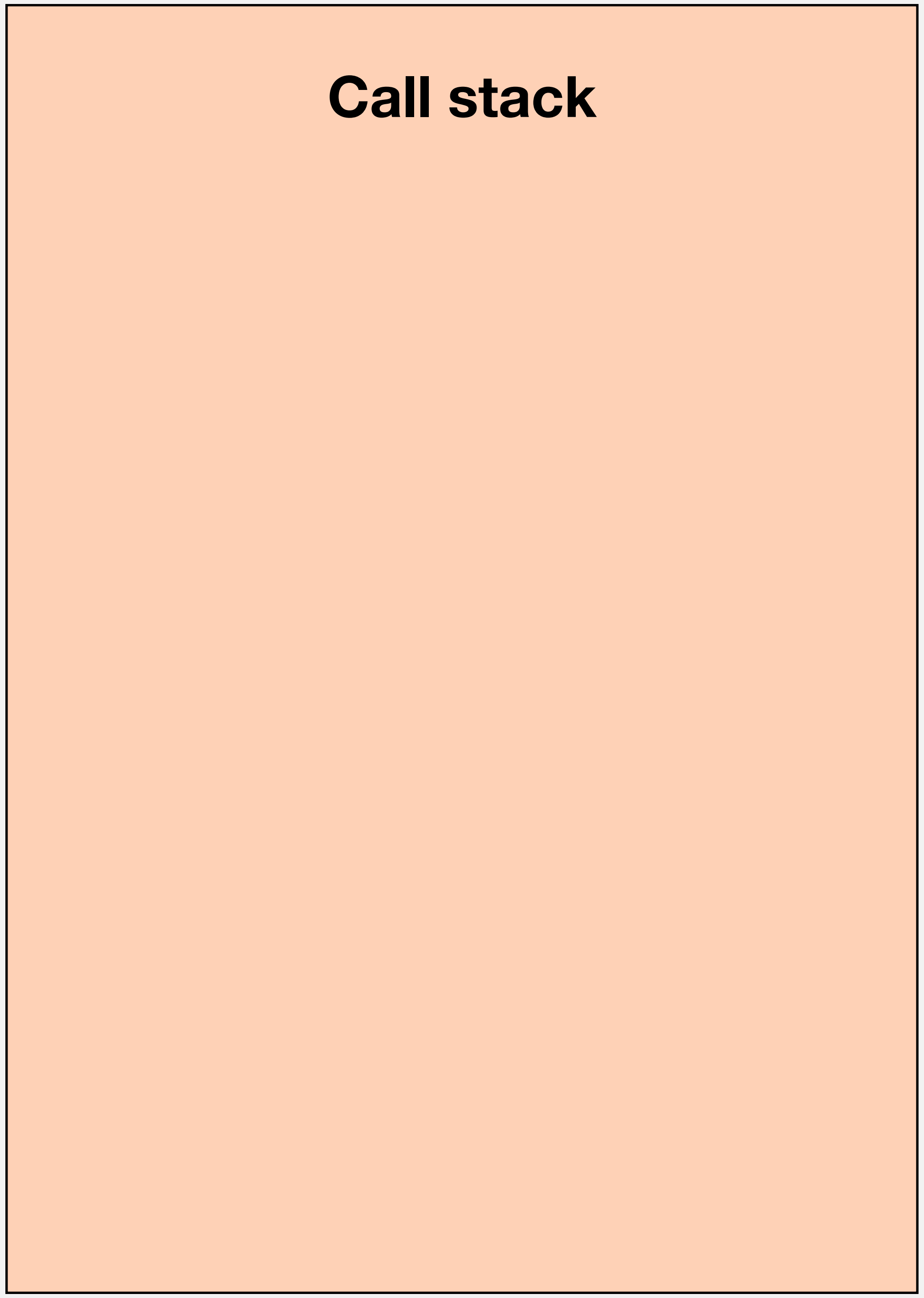


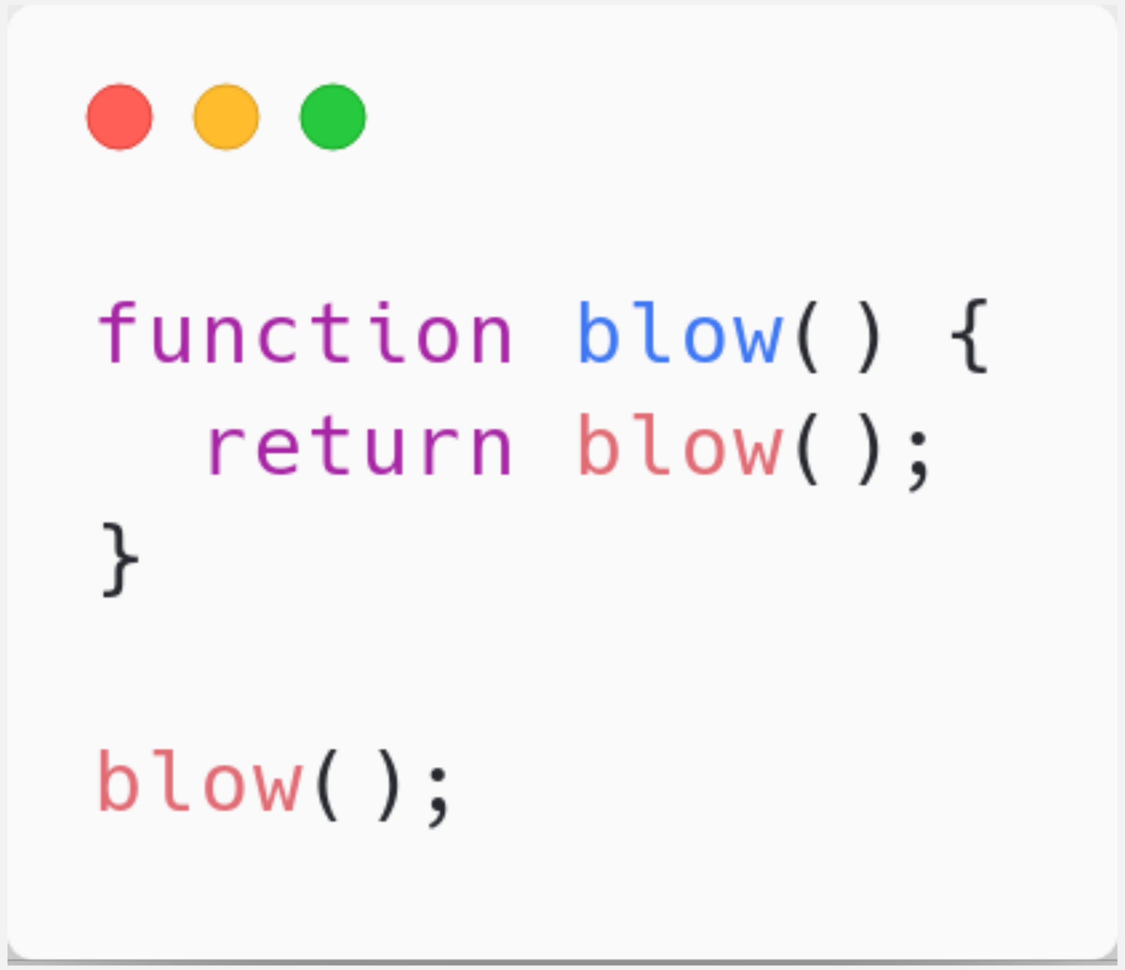
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



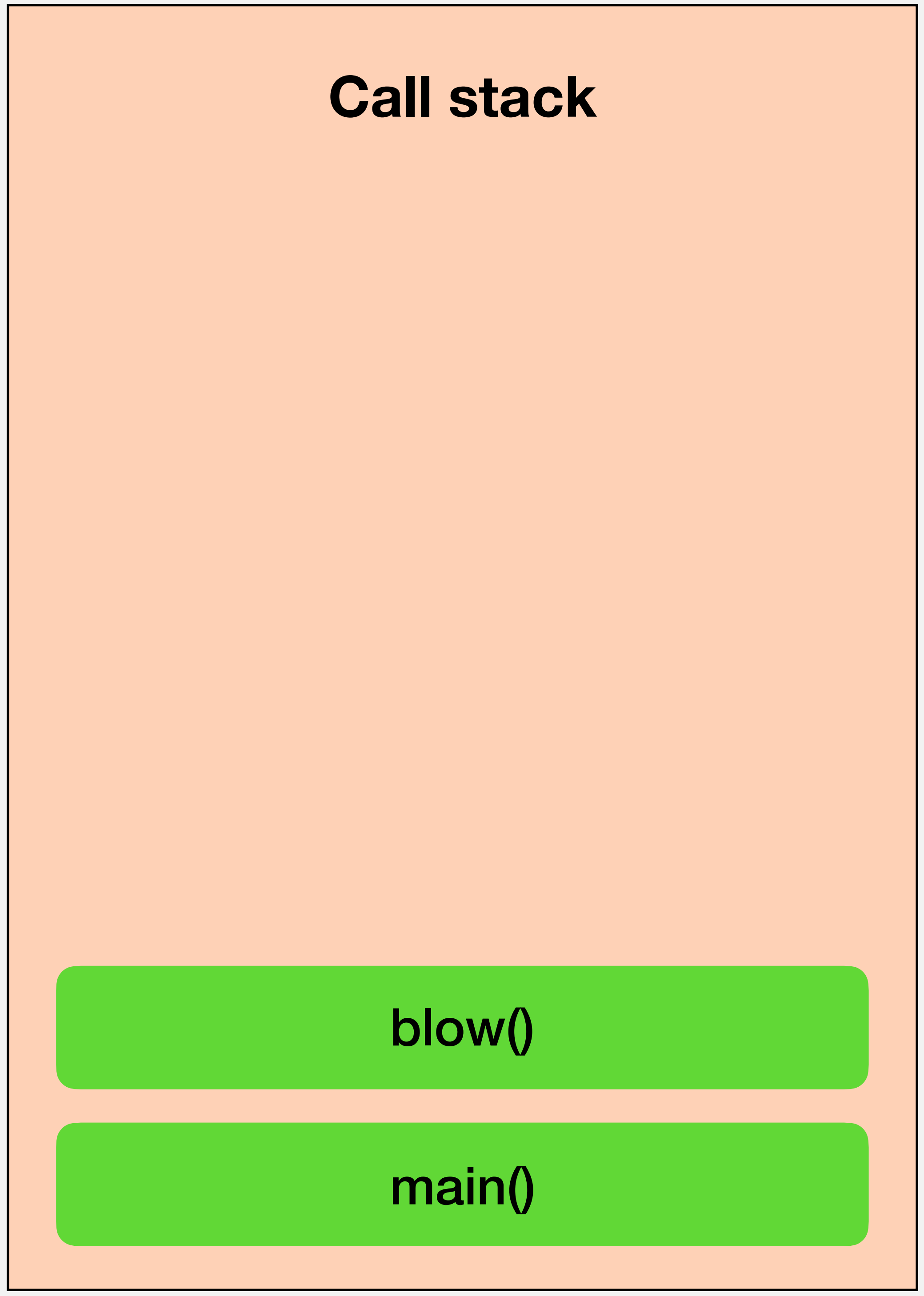


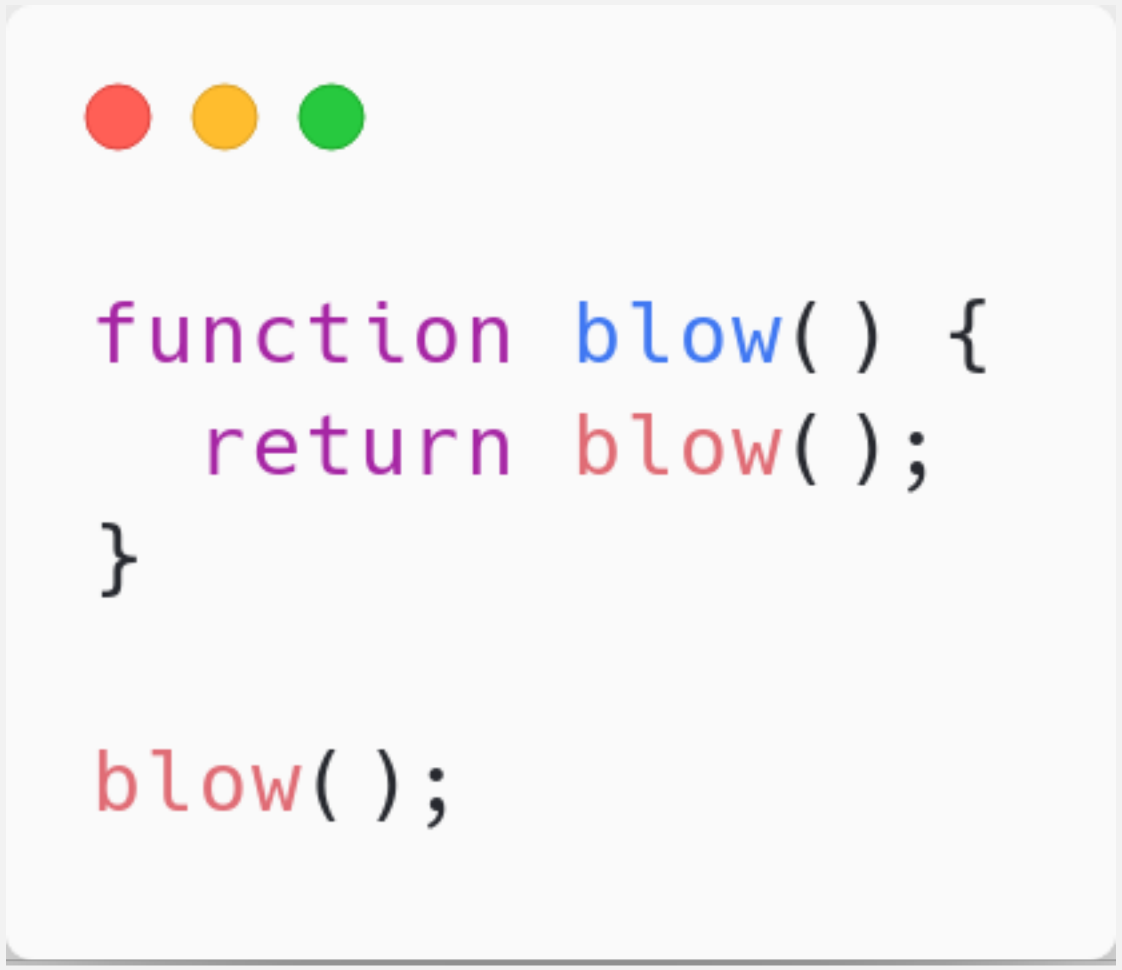
```
function mult(a, b) {
  return a * b;
}


function square(n) {
  return mult(n, n);
}

function consoleSquare(n) {
  const squared = square(n);
  console.log(squared);
}

consoleSquare(4);
```



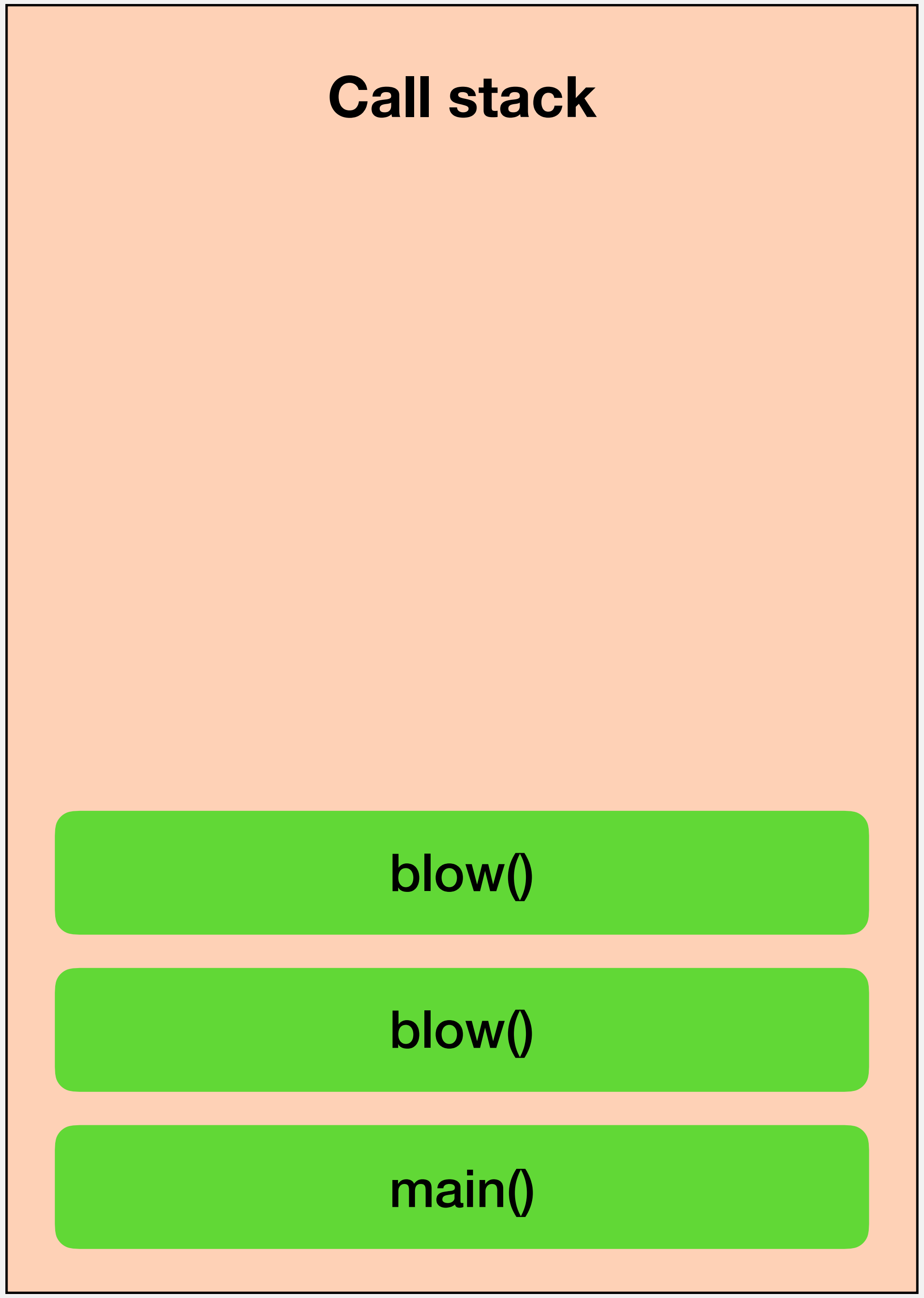


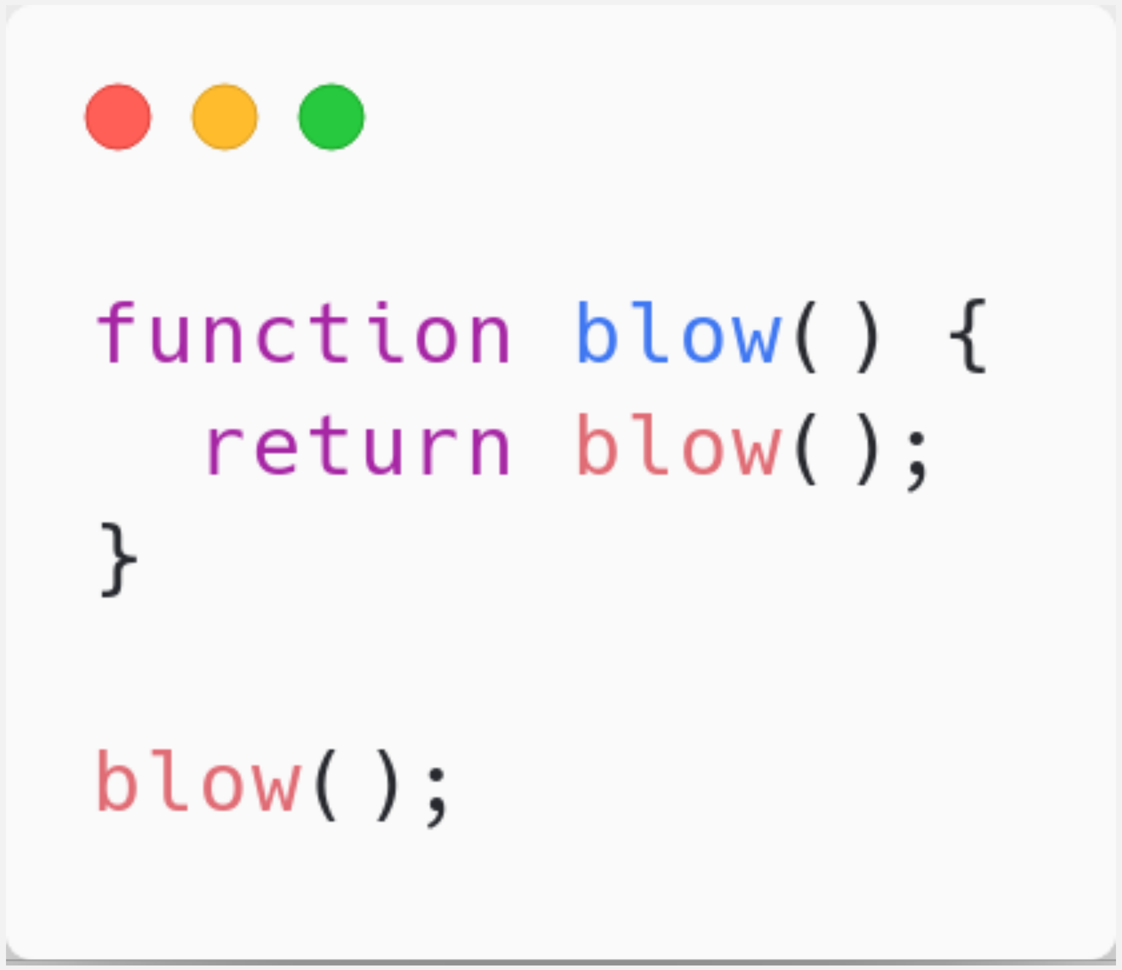
```
function blow() {
  return blow();
}

blow();
```



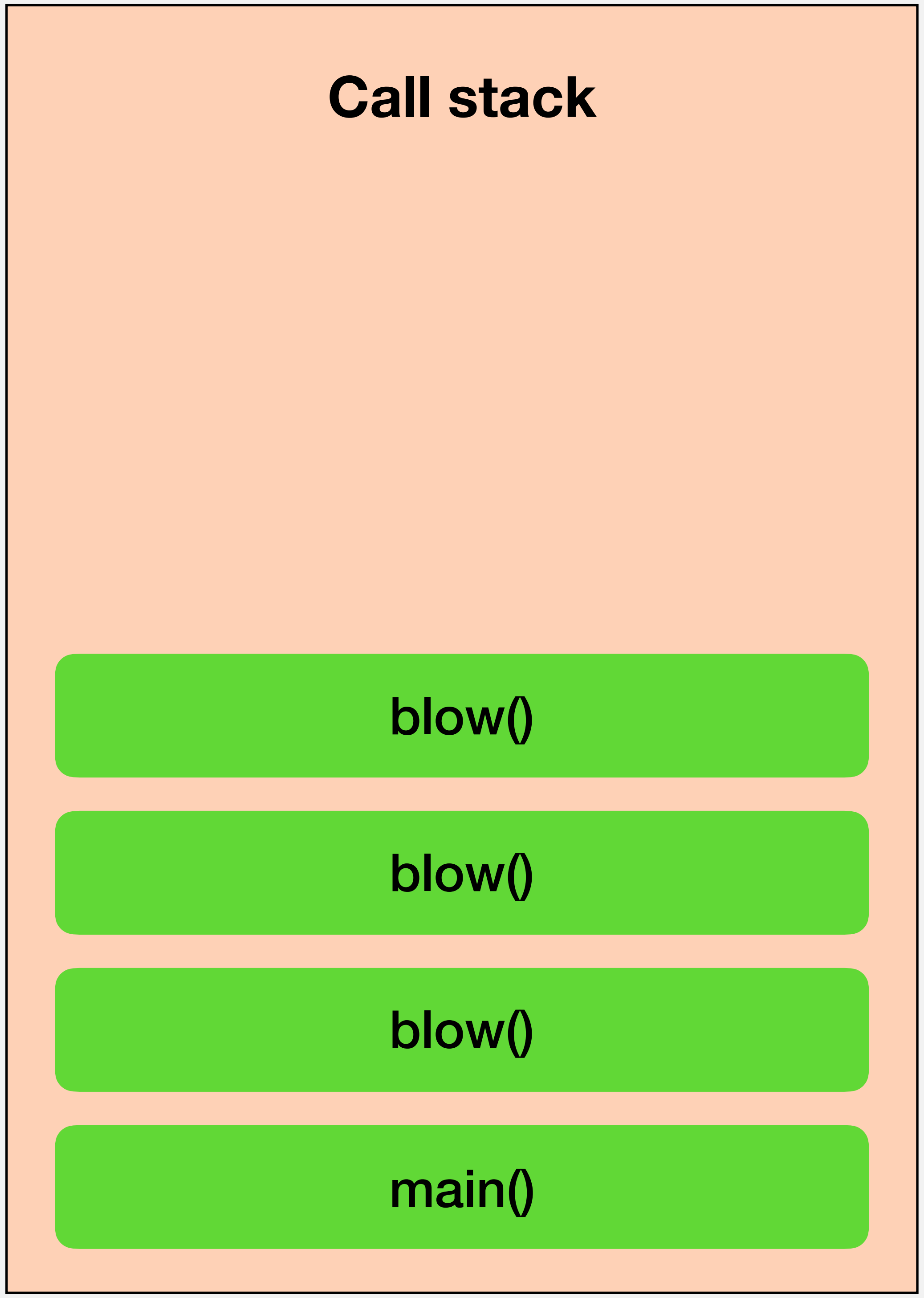


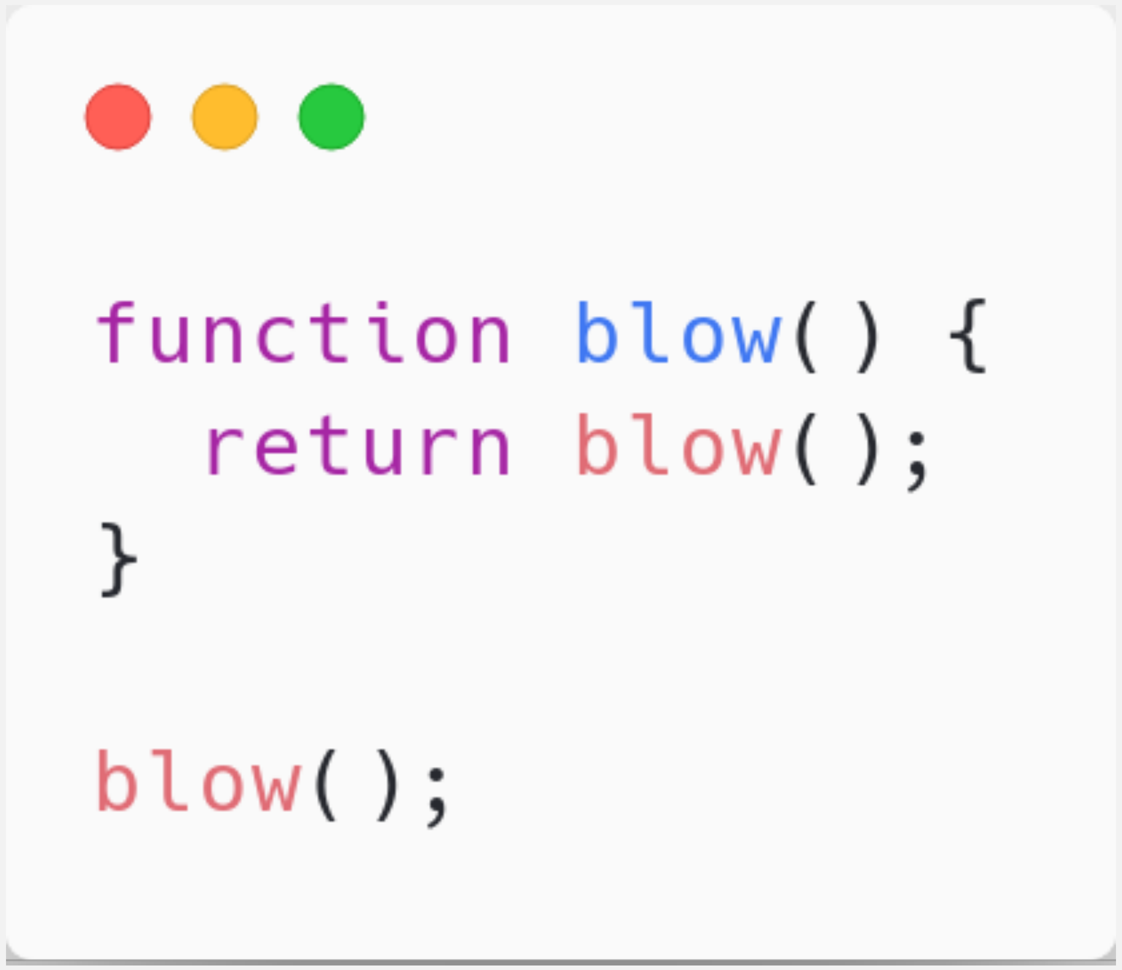
```
function blow() {
  return blow();
}

blow();
```



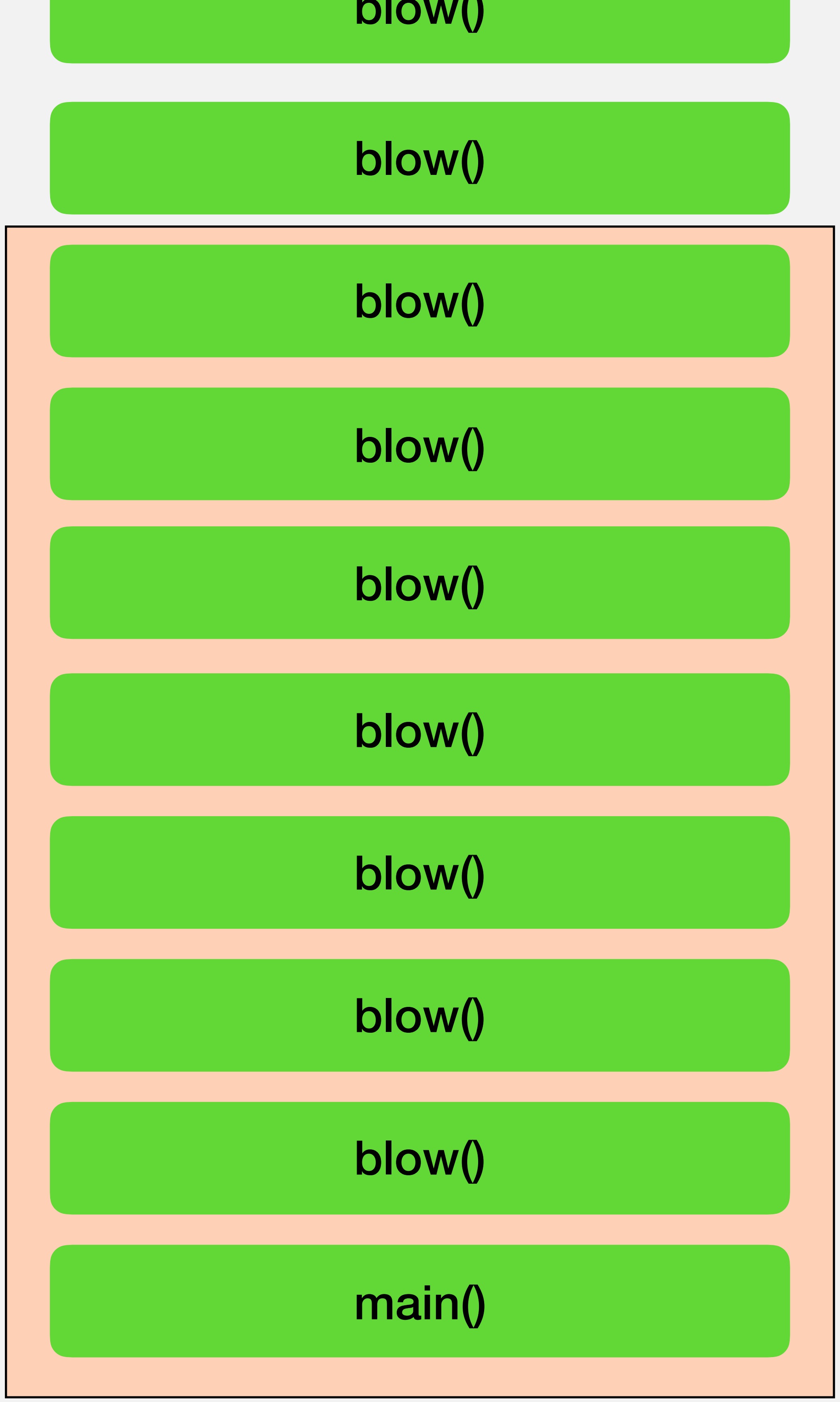


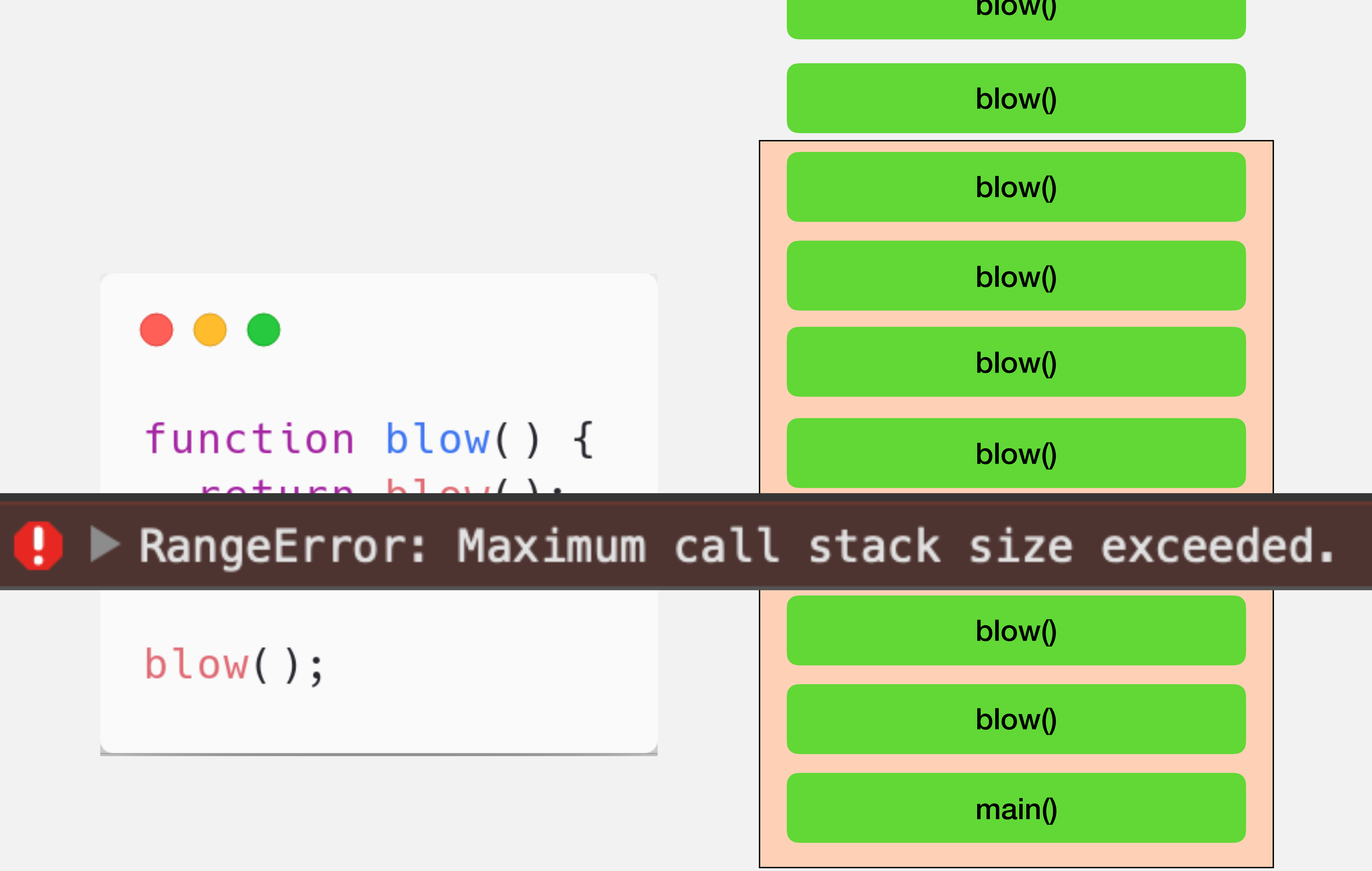
```
function blow() {
  return blow();
}

blow();
```





```
function blow() {
  return blow();
}

blow();
```





```javascript
function blow() {
  return blow();

blow();
```

```
RangeError: Maximum call stack size exceeded.
```

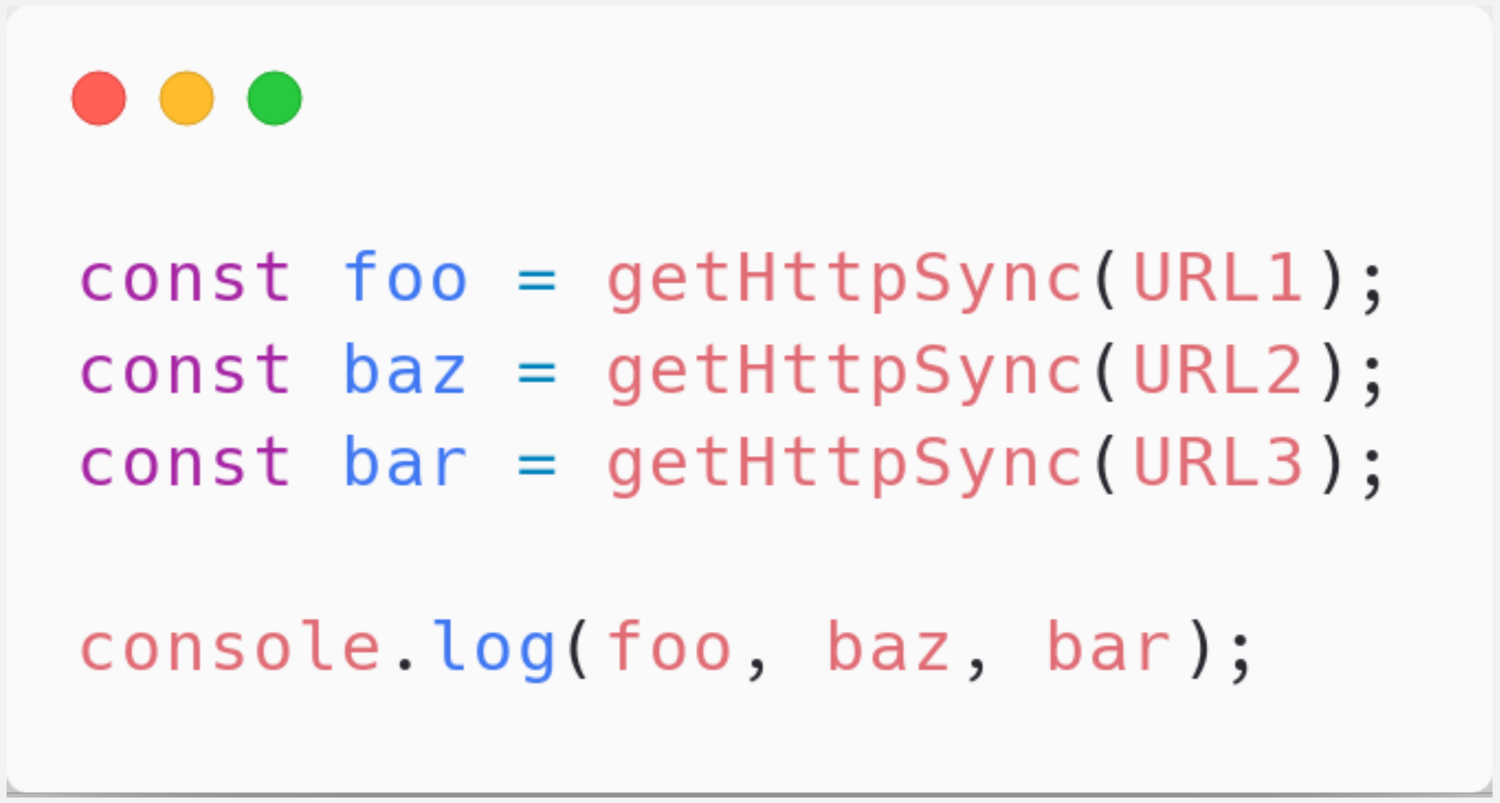
I am

- ~~Single threaded~~
- Non-blocking
- Asynchronous
- Concurrent

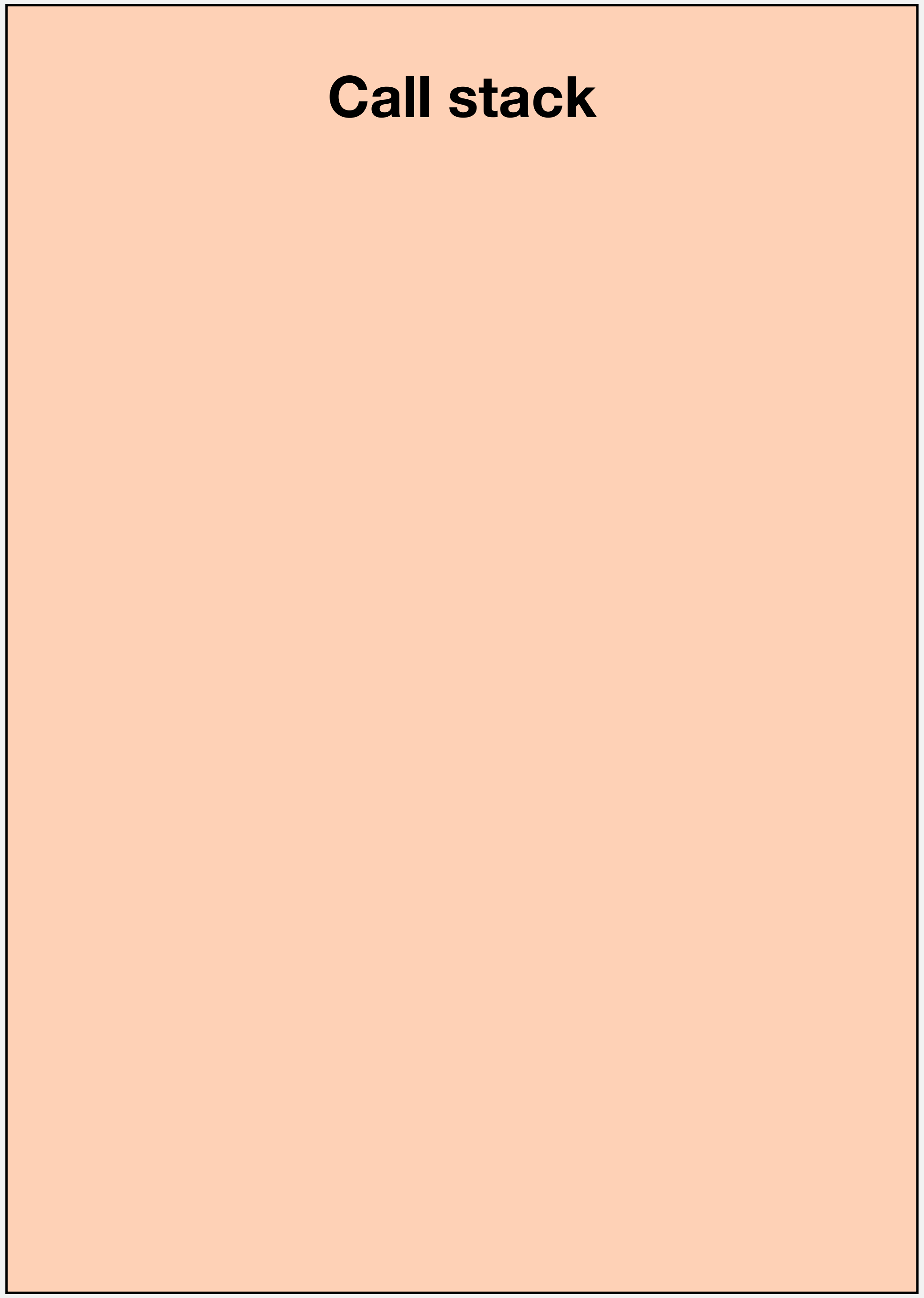
What is blocking?

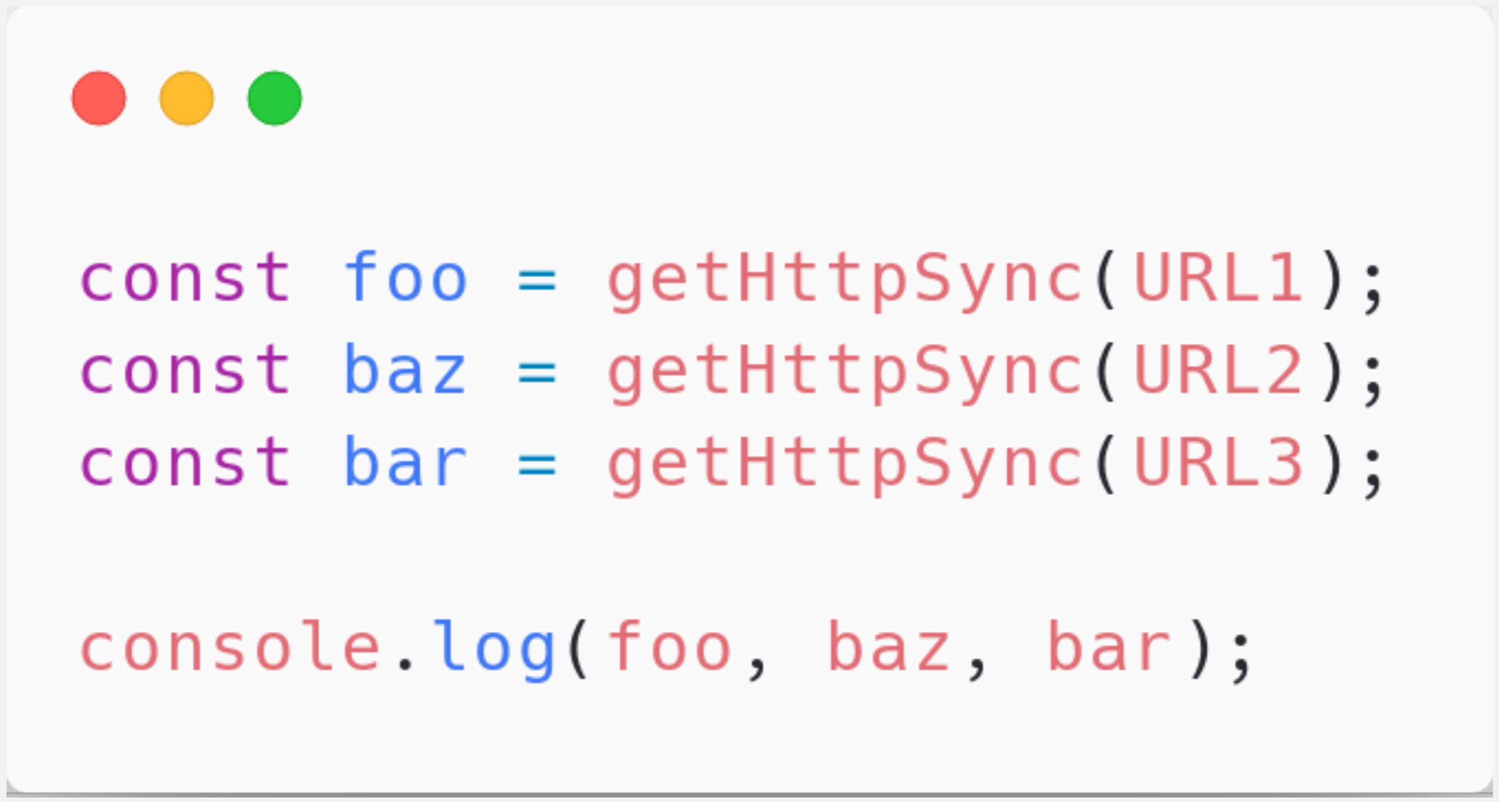
What is blocking?

Slow function calls (such as loop from 1 to 1 billion, image processing, networking etc.) on a call stack that block other function calls

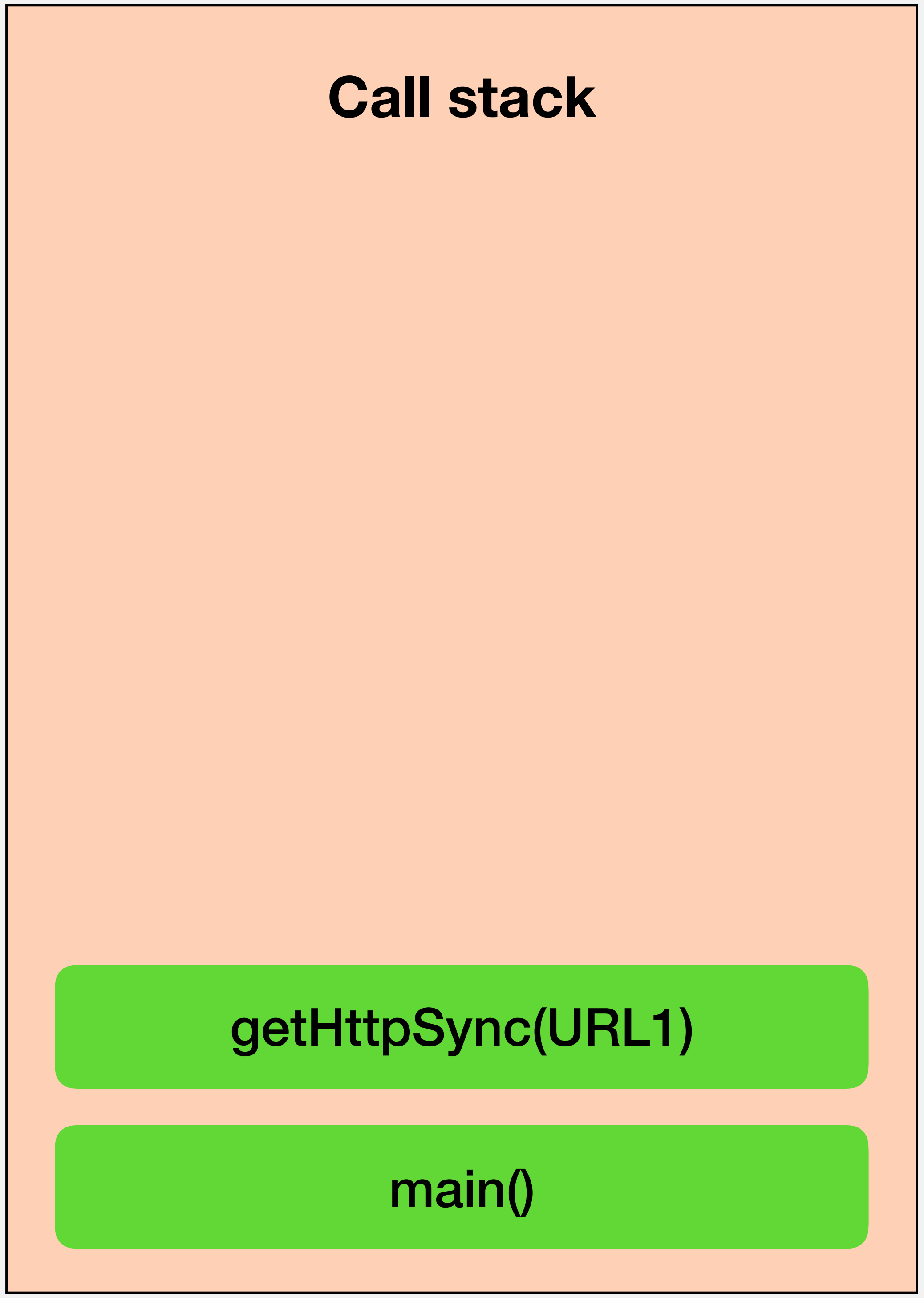


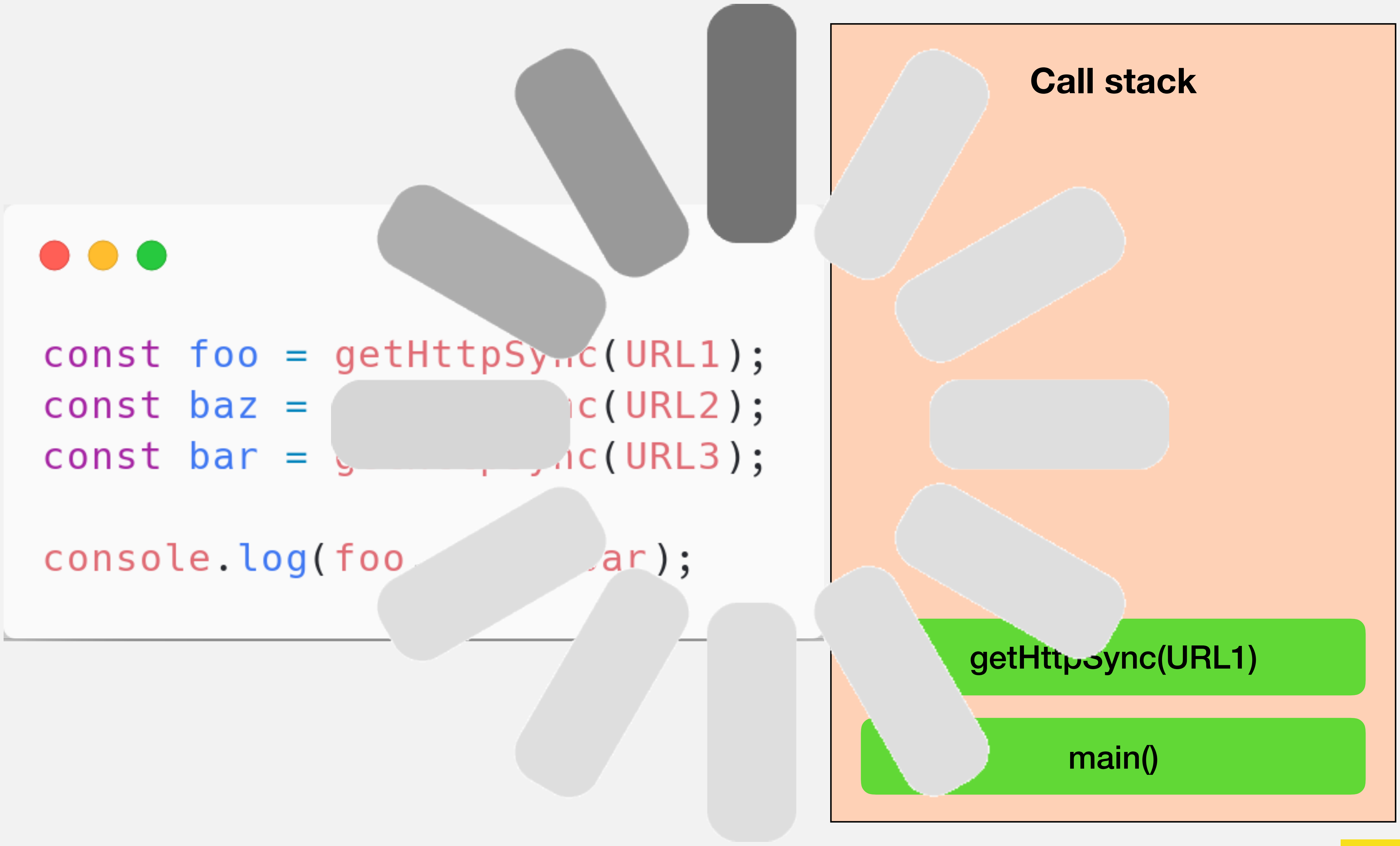
```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```





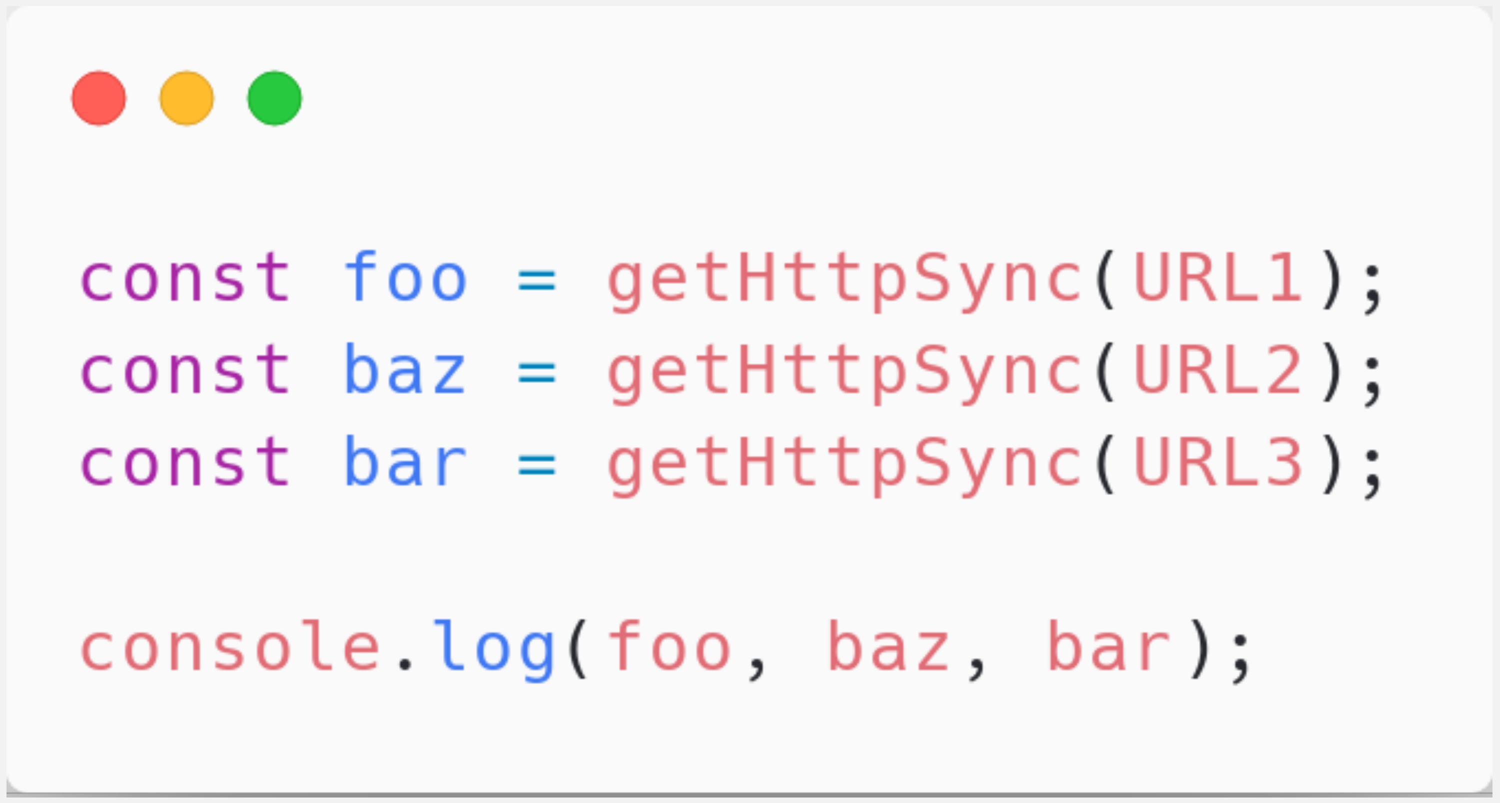
```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```



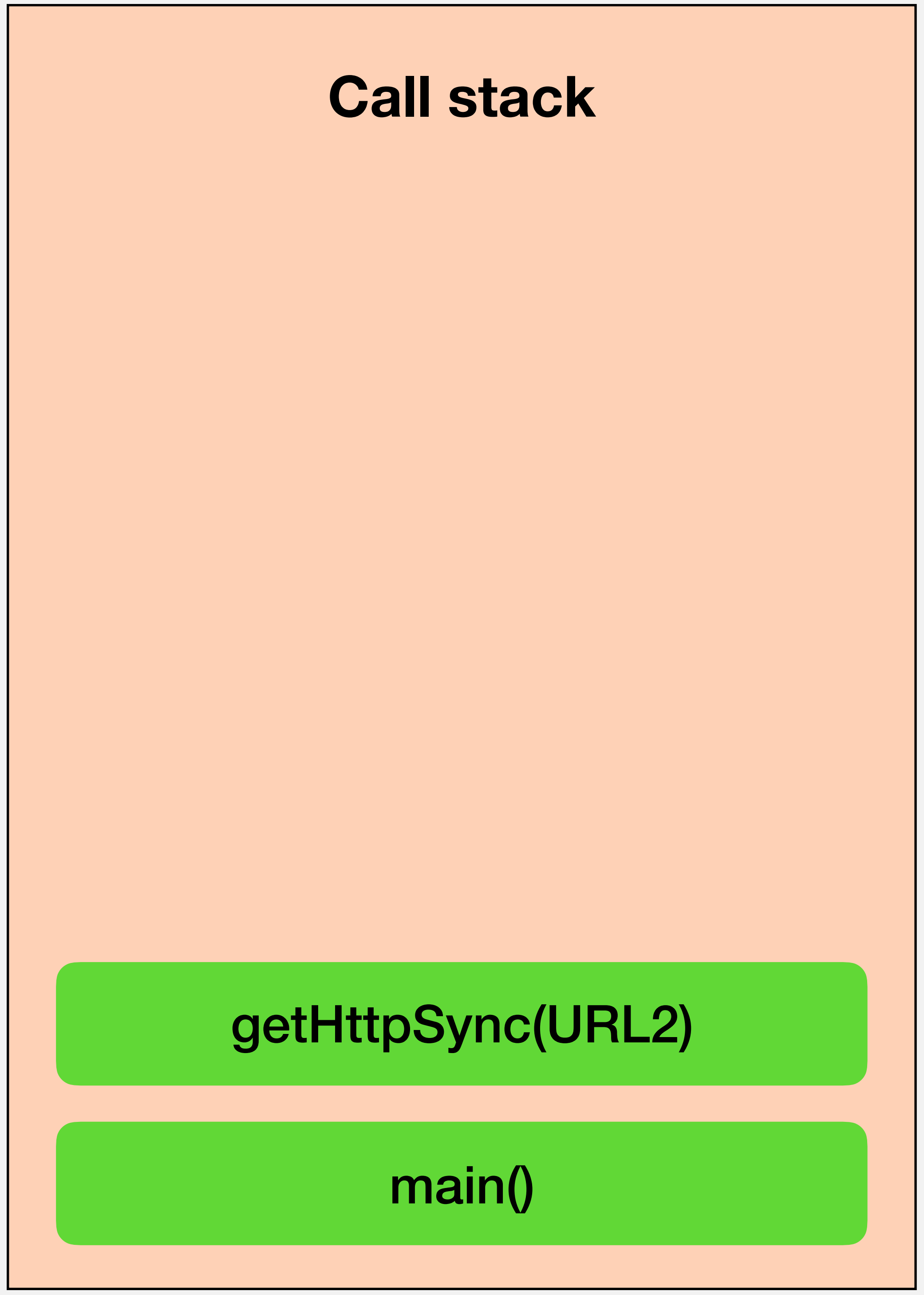


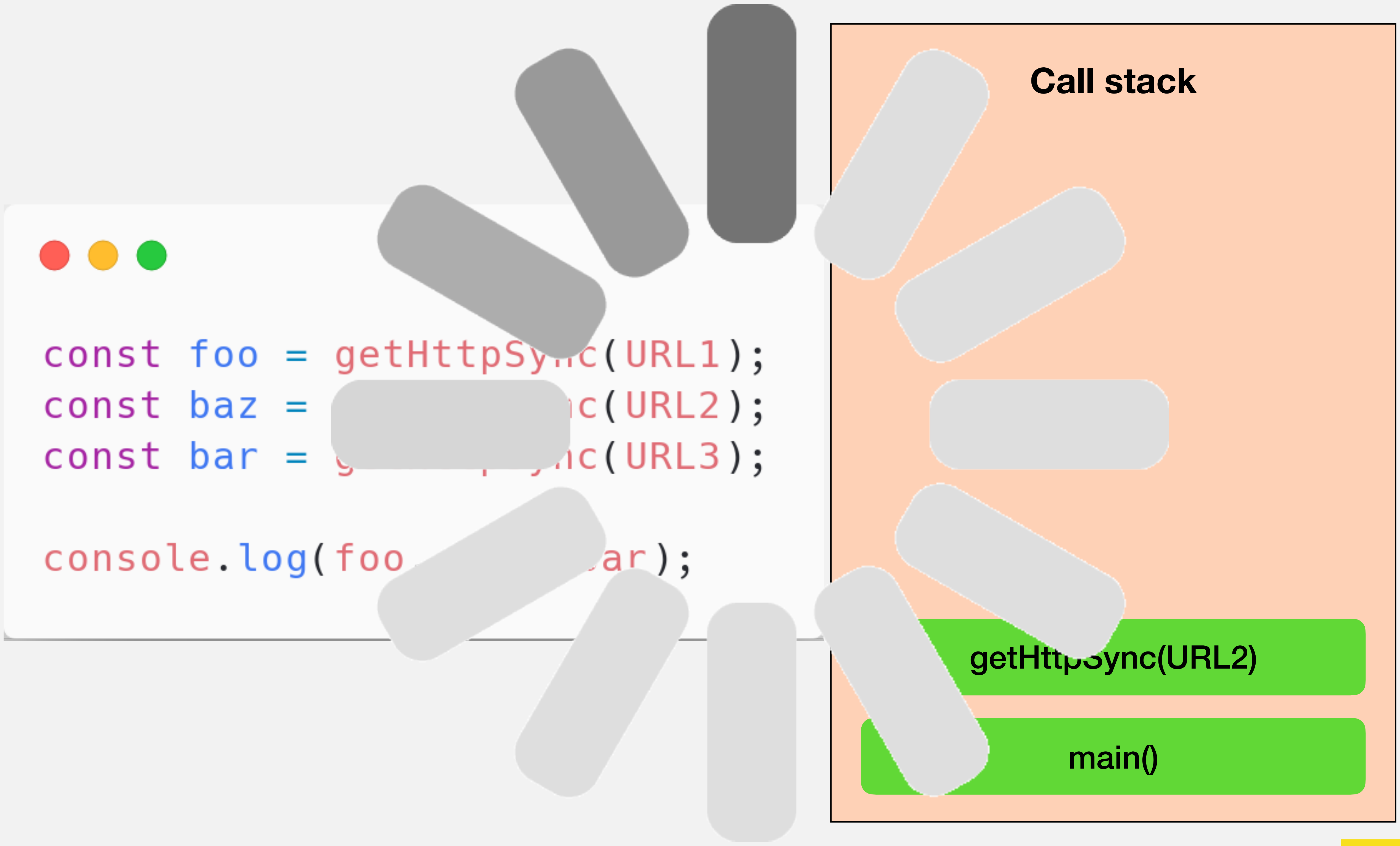
```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```

JS

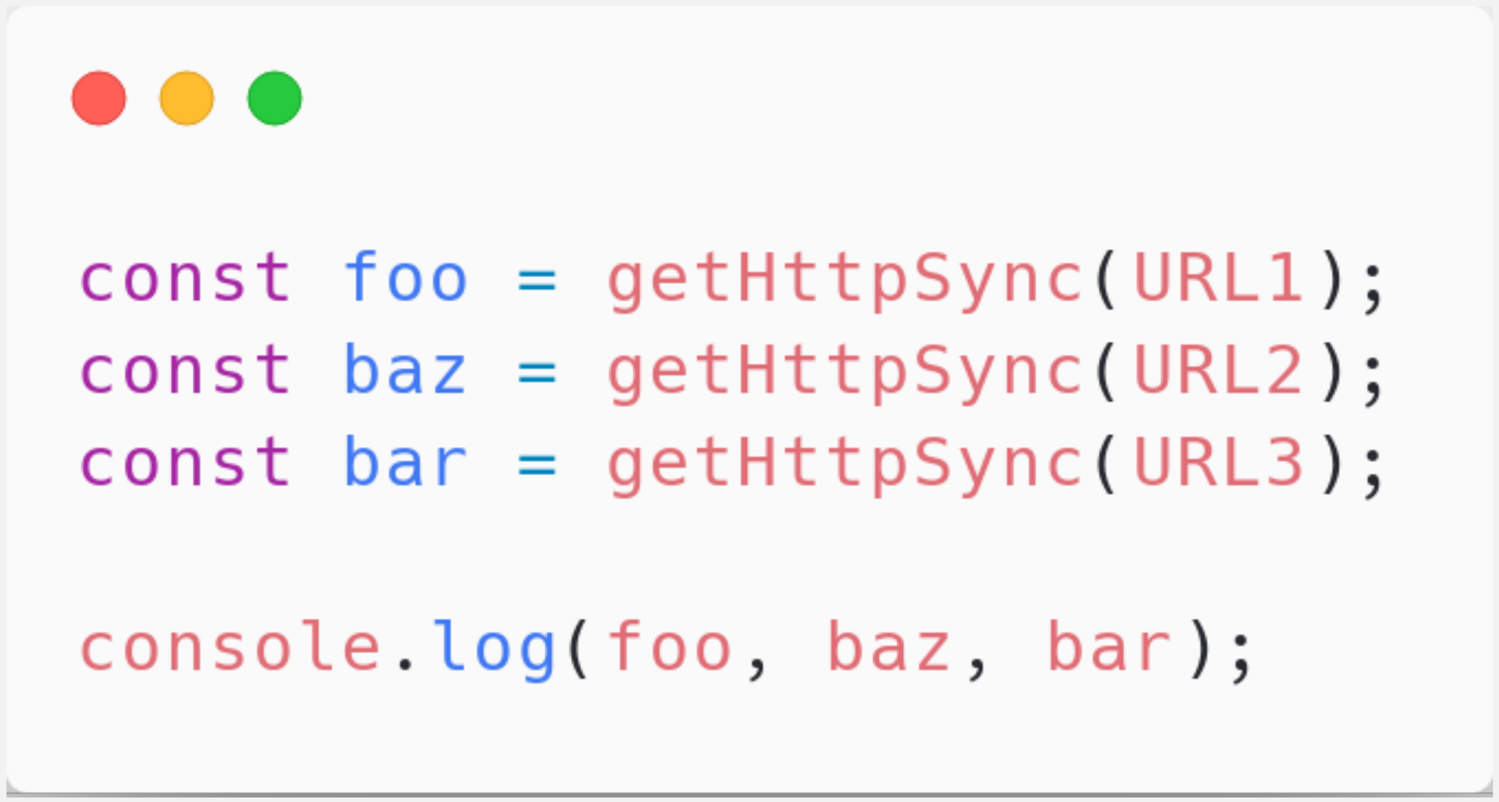


```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```

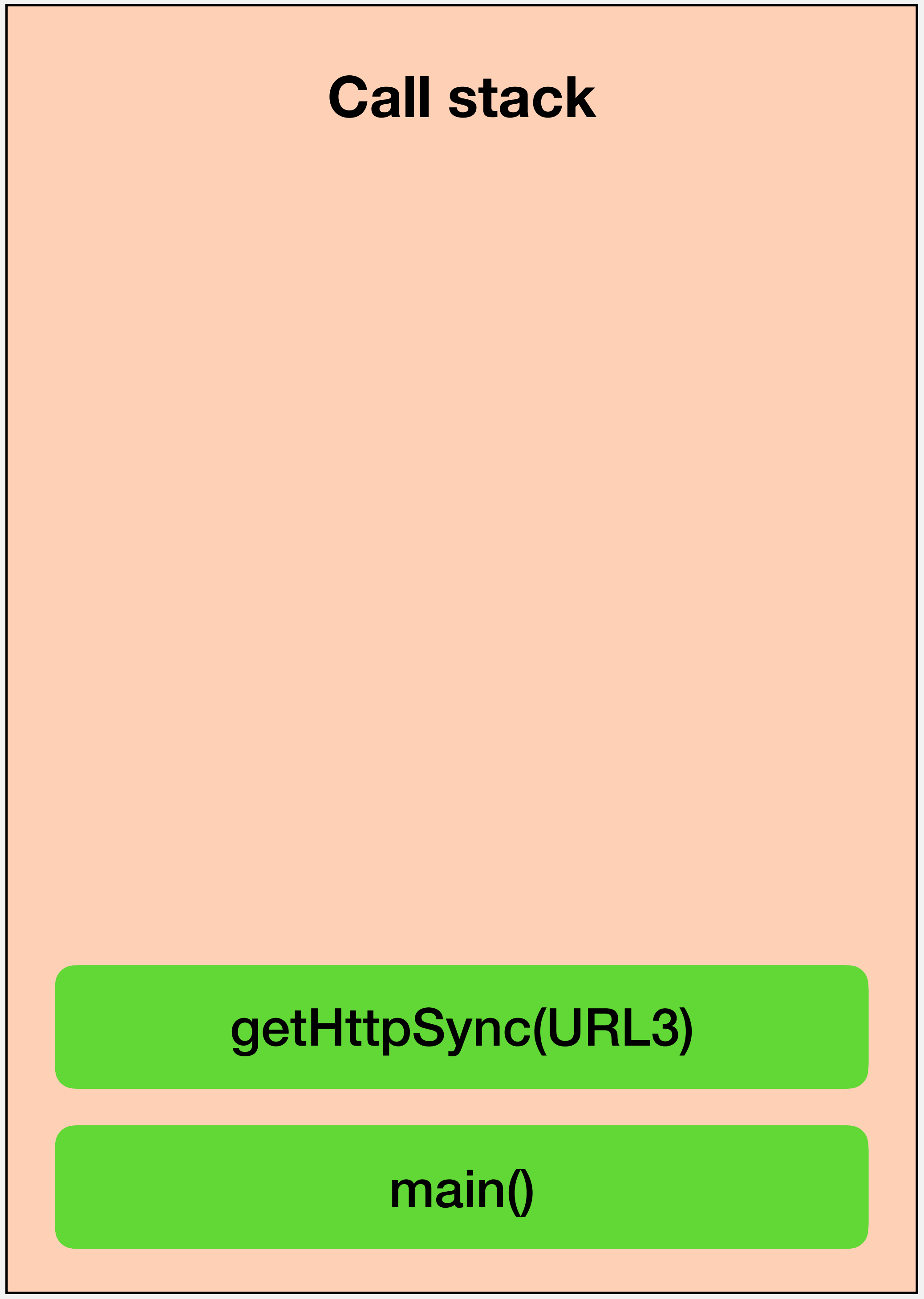


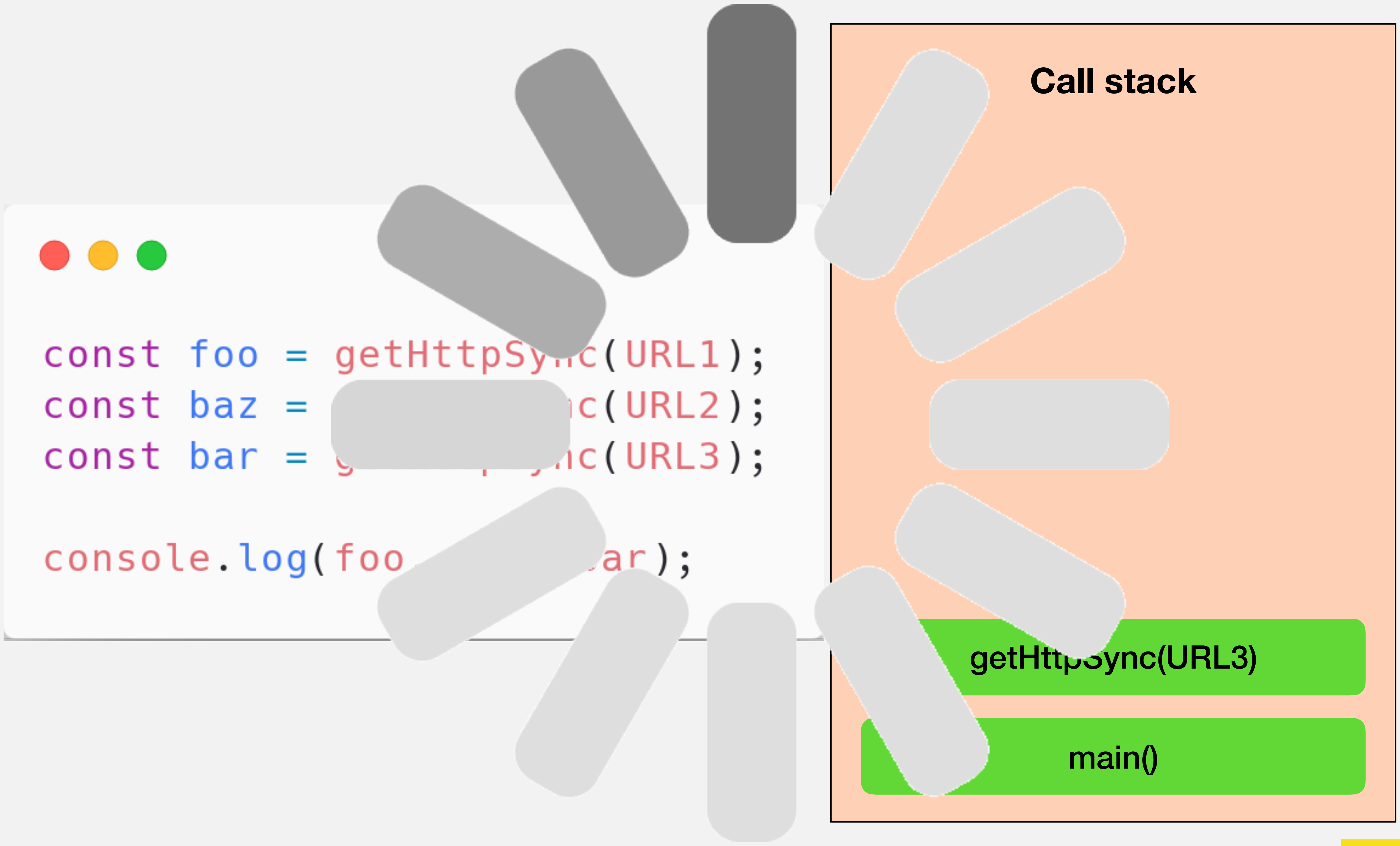


```
const foo = getHttpSync(URL1);
const baz = [illegible]c(URL2);
const bar = [illegible]c(URL3);

console.log(foo[illegible]ar);
```

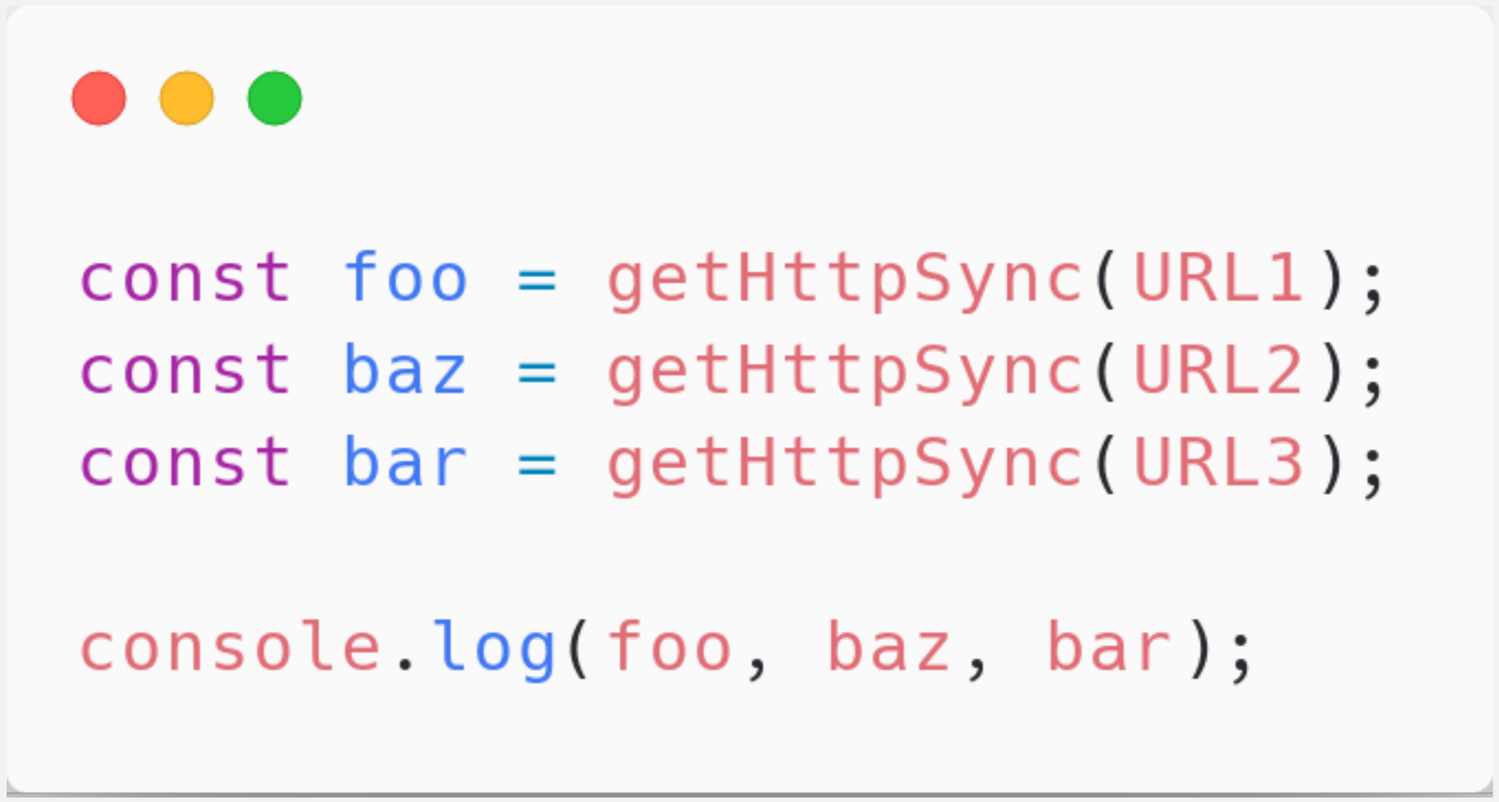


```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```

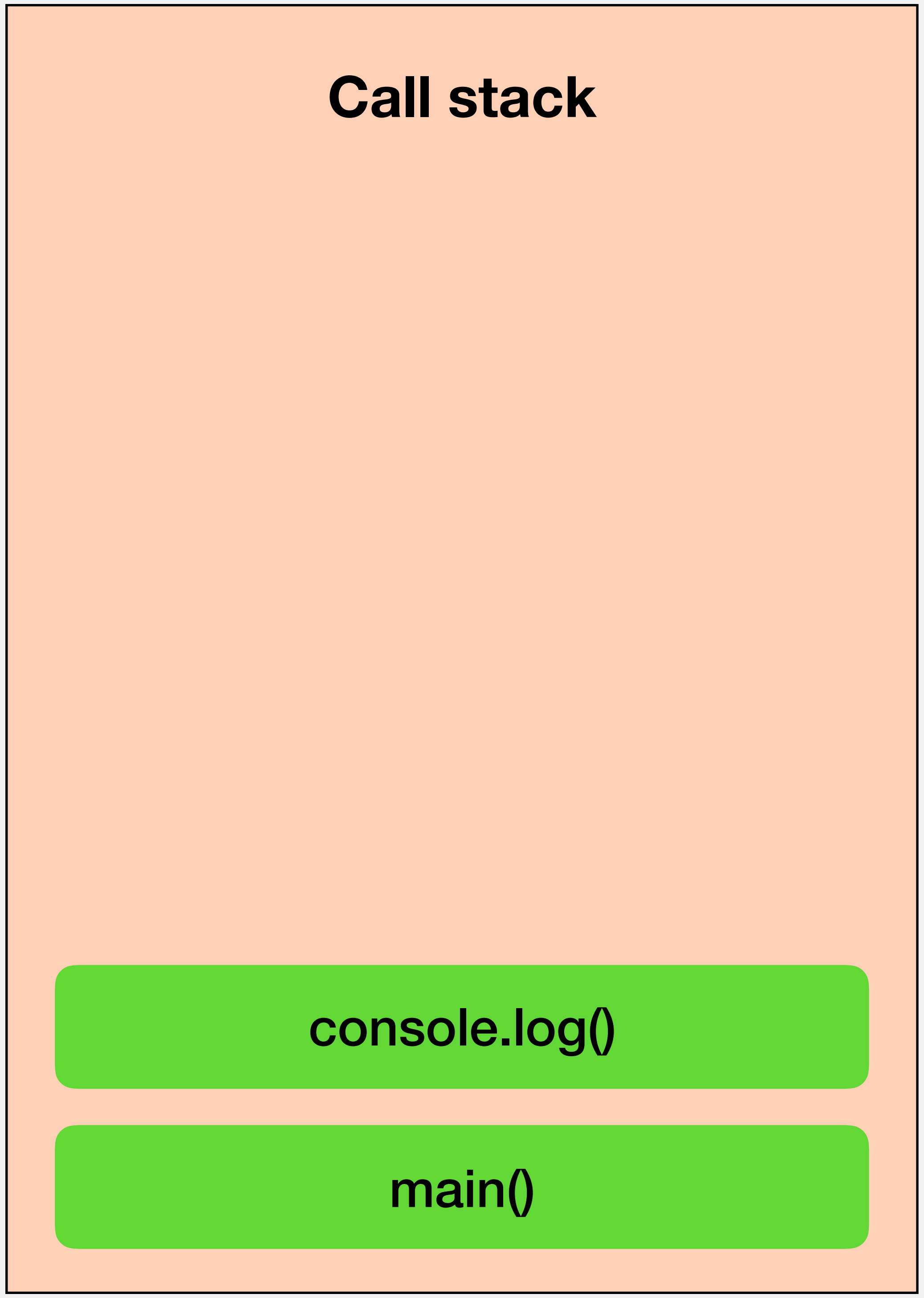




```
const foo = getHttpSy c(URL1);
const baz =          c(URL2);
const bar = g      p  nc(URL3);

console.log(foo       ar);
```



```
const foo = getHttpSync(URL1);
const baz = getHttpSync(URL2);
const bar = getHttpSync(URL3);

console.log(foo, baz, bar);
```



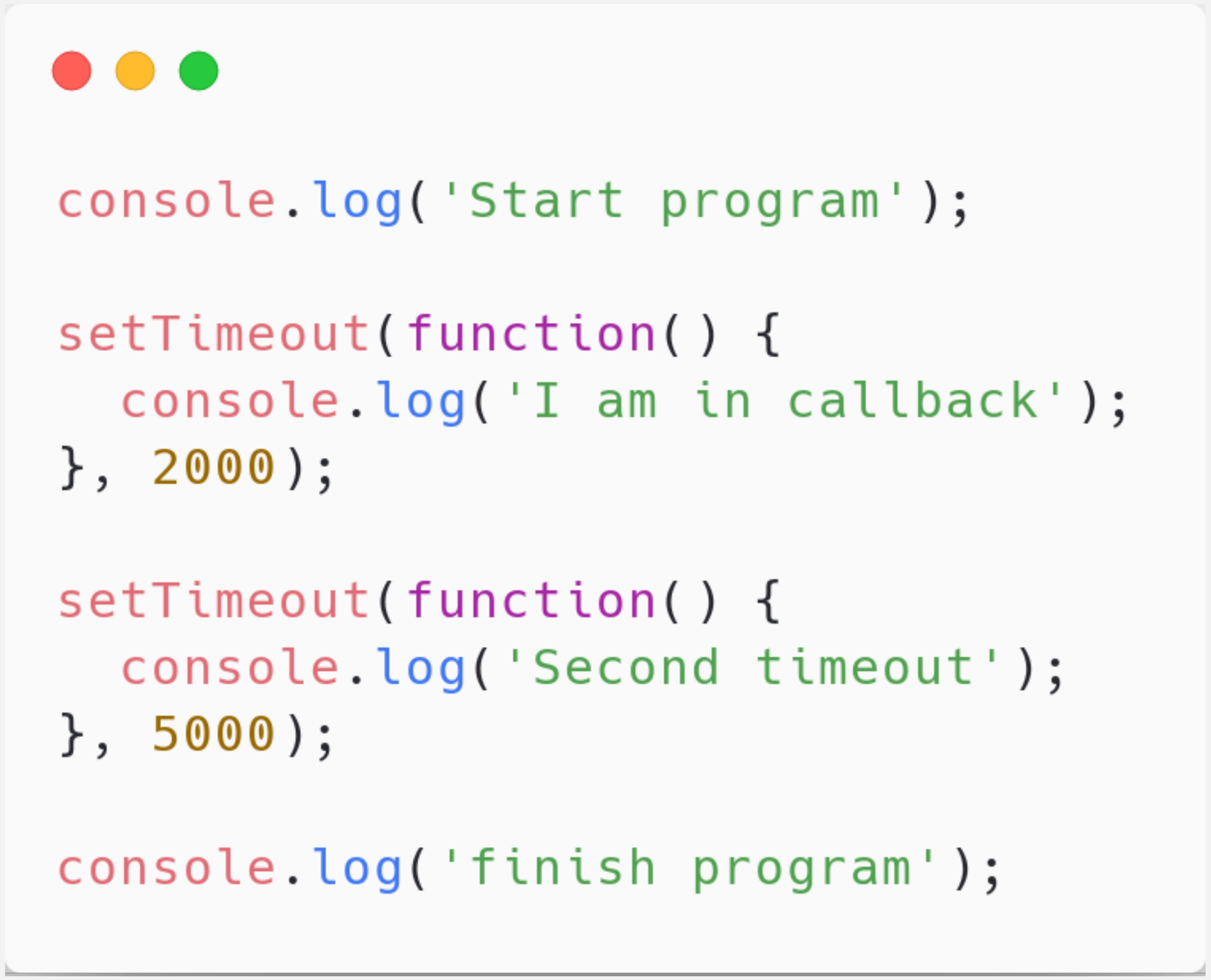
Solution?

Solution?

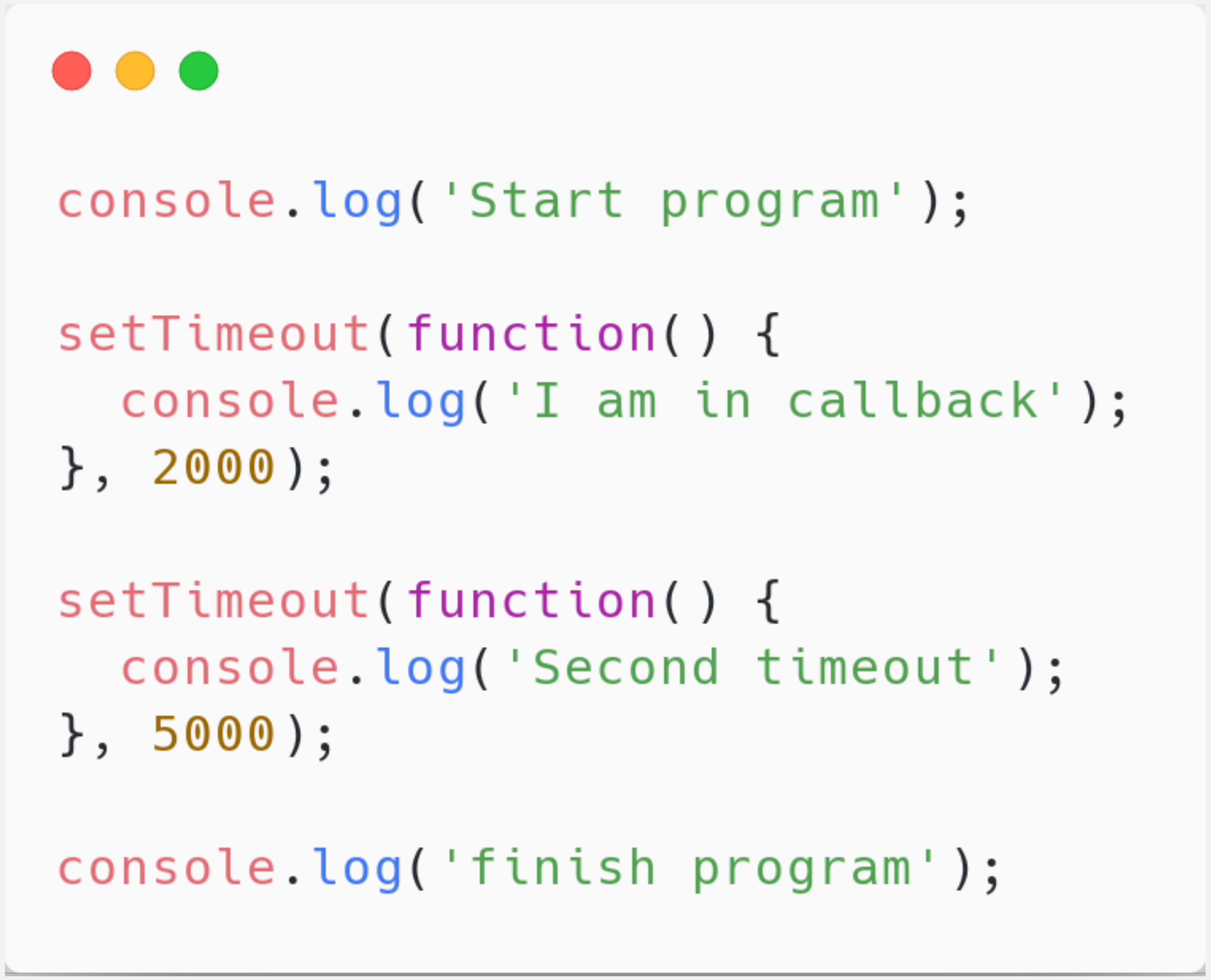
Asynchronous callbacks.

…

Call me maybe?



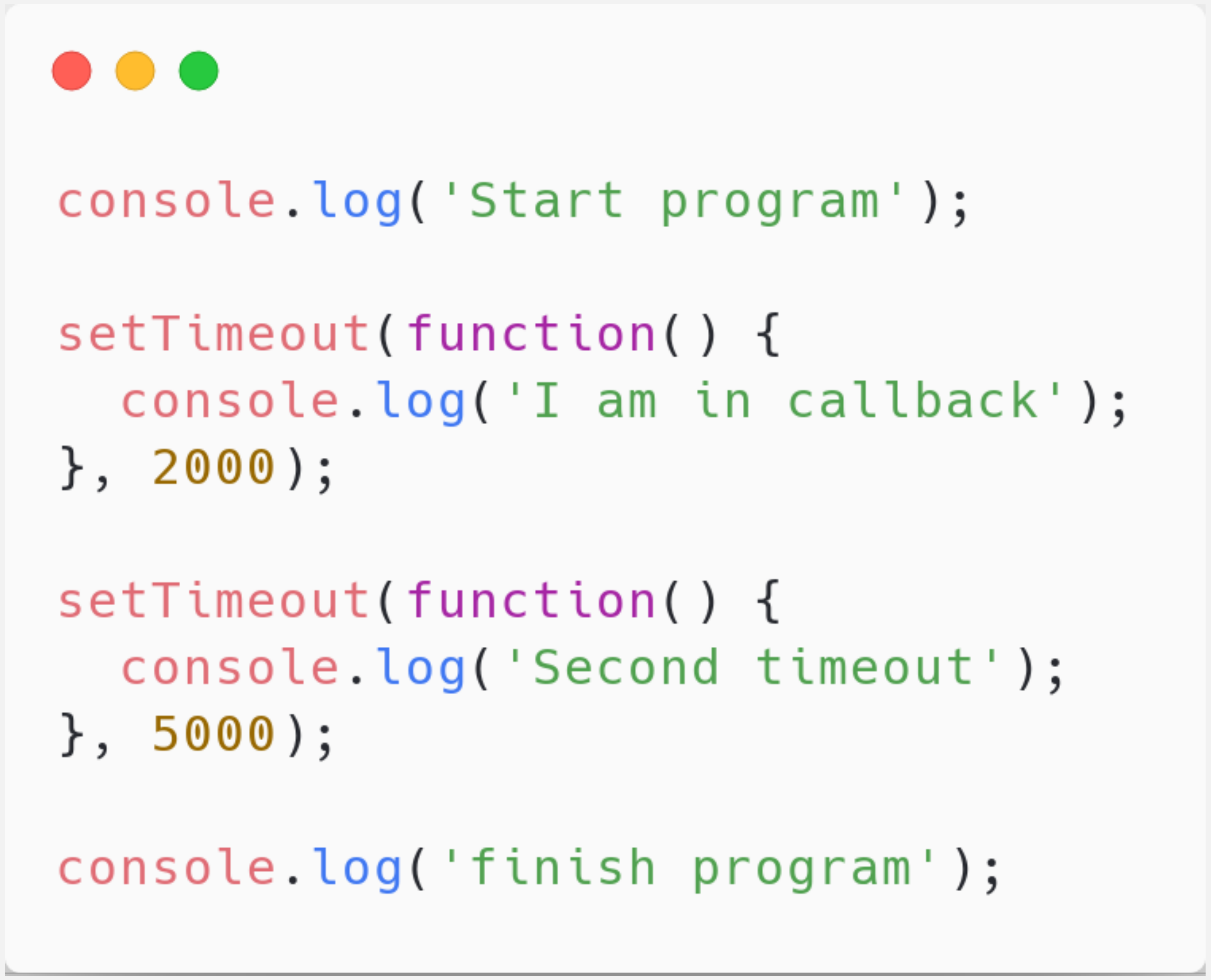
```javascript
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



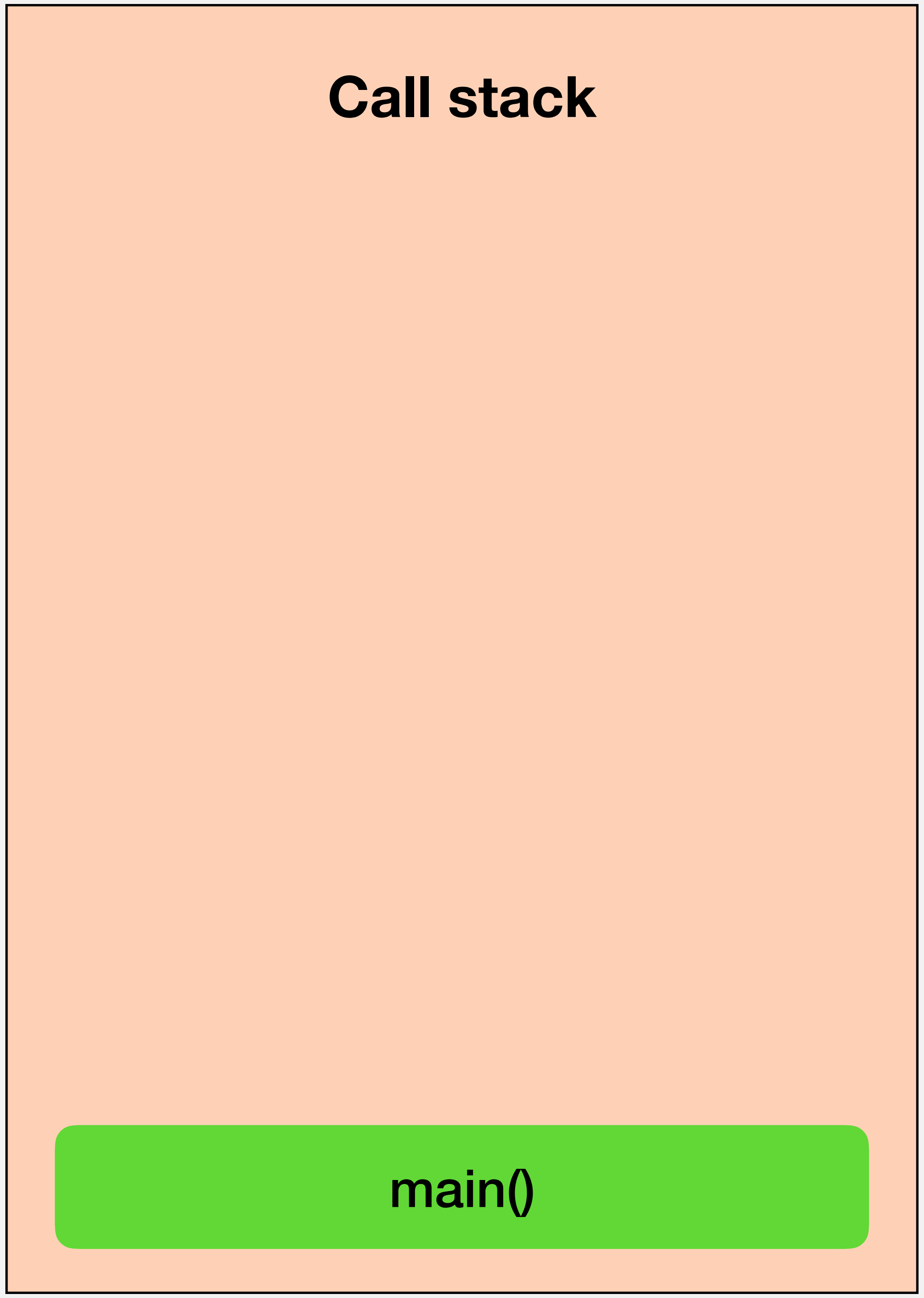
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

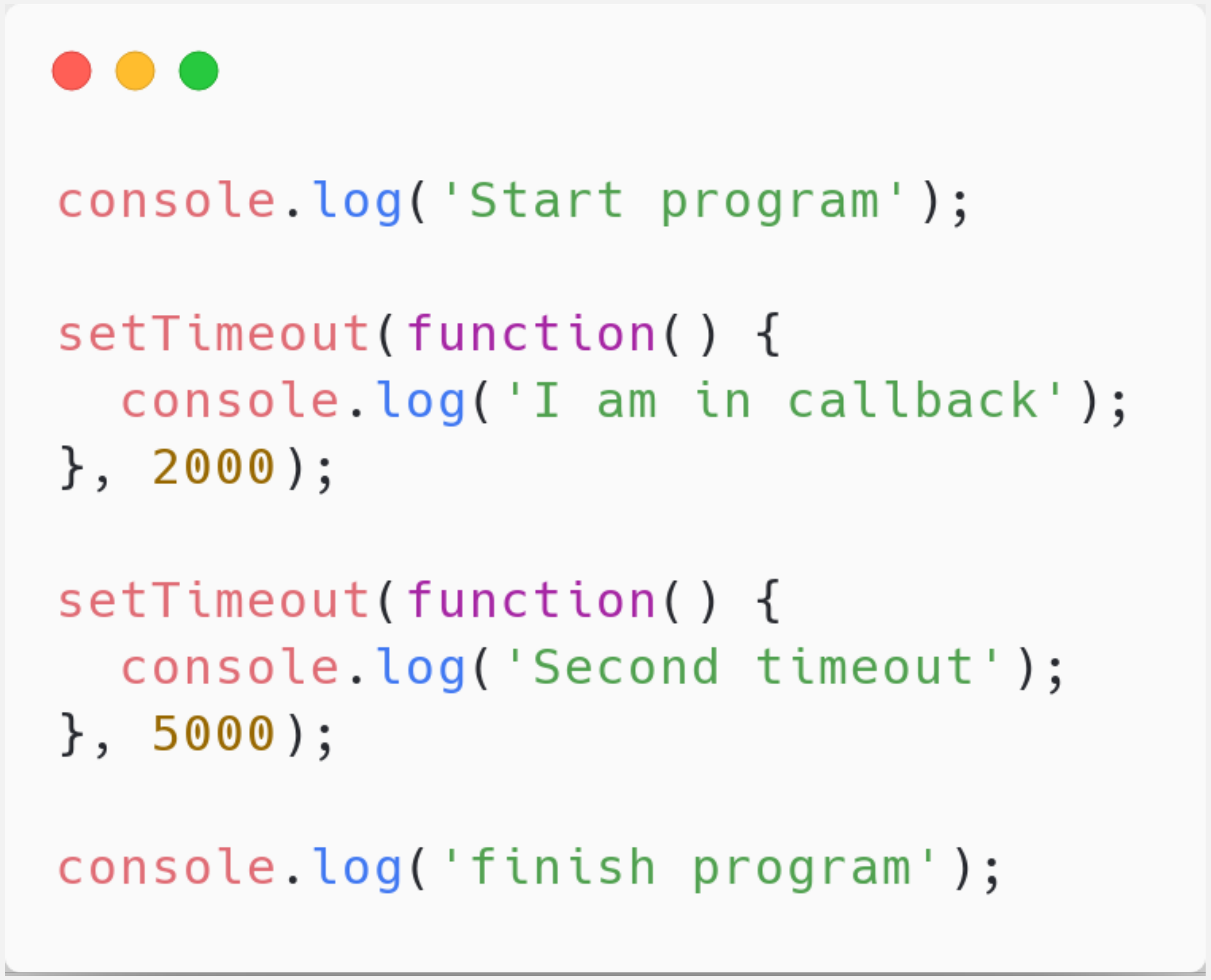
Start program
finish program
I am in callback
Second timeout

JS

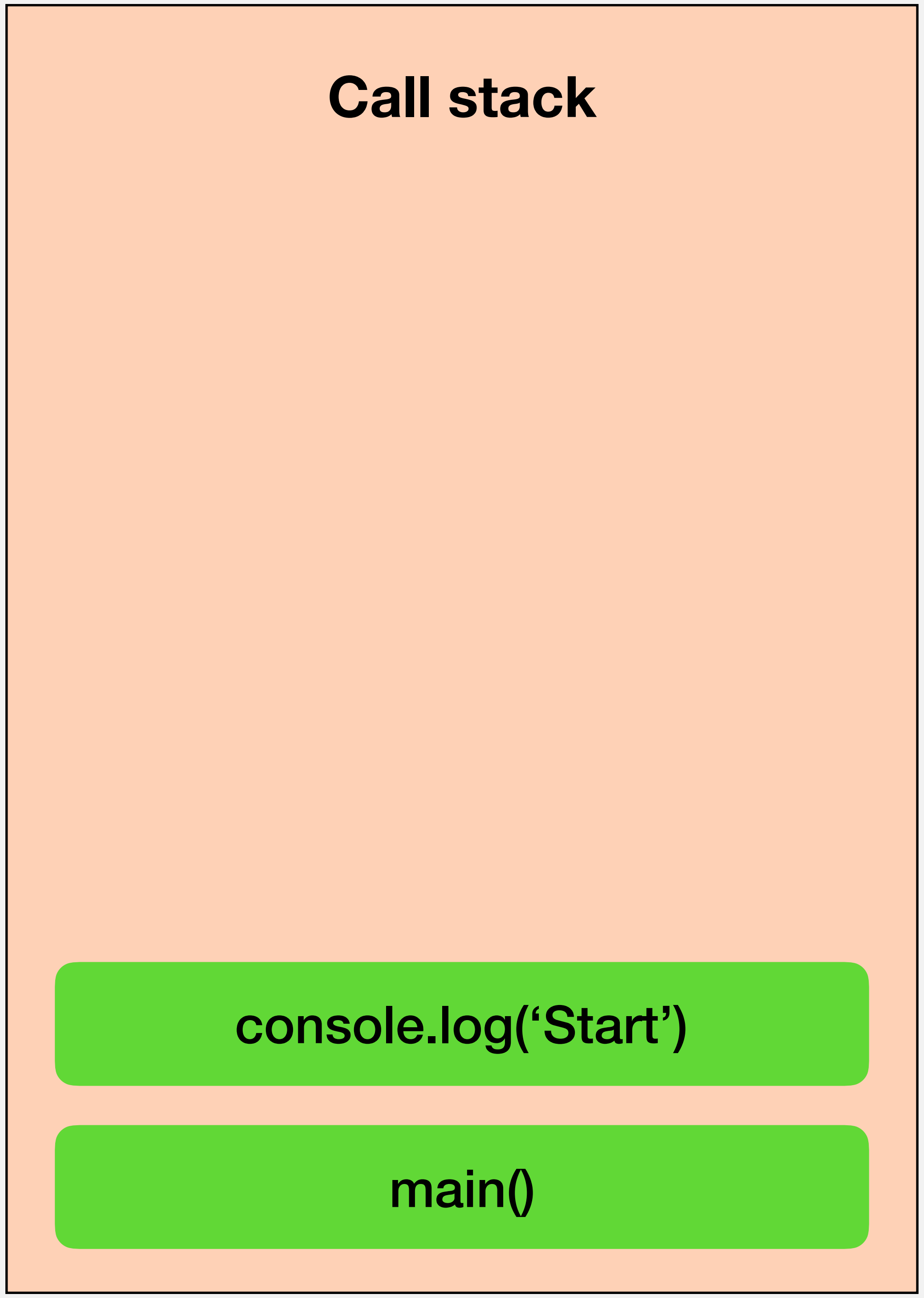


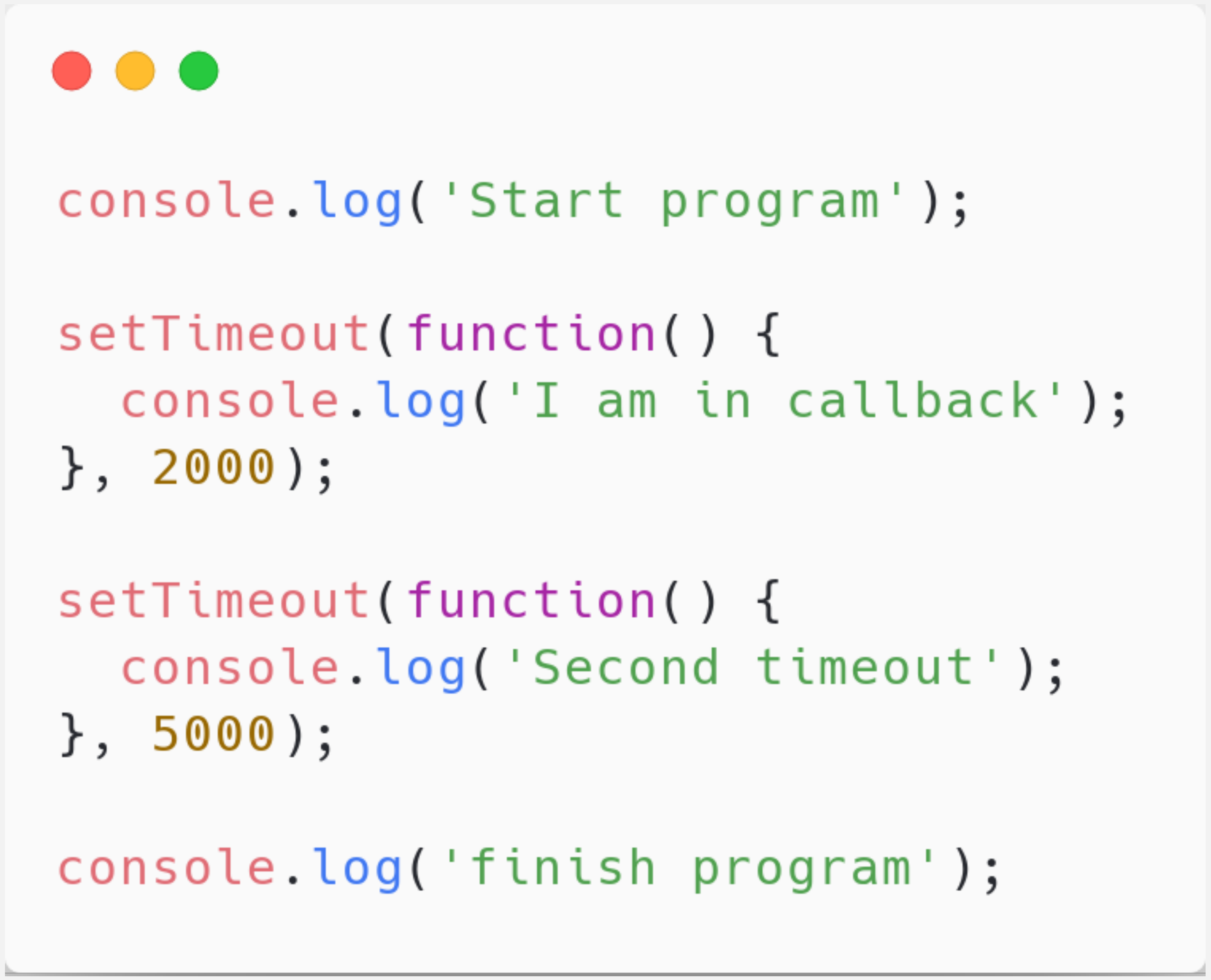
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



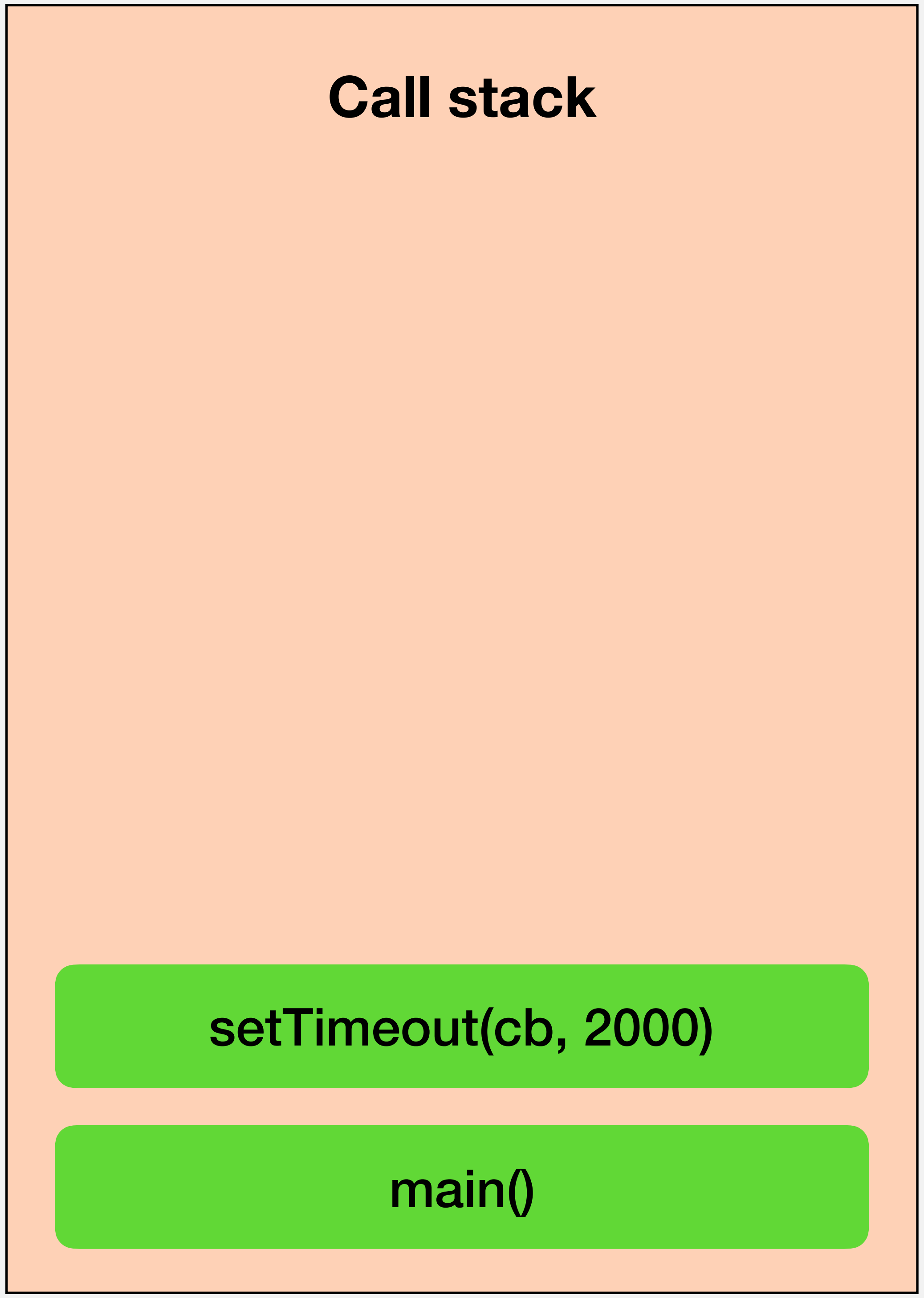


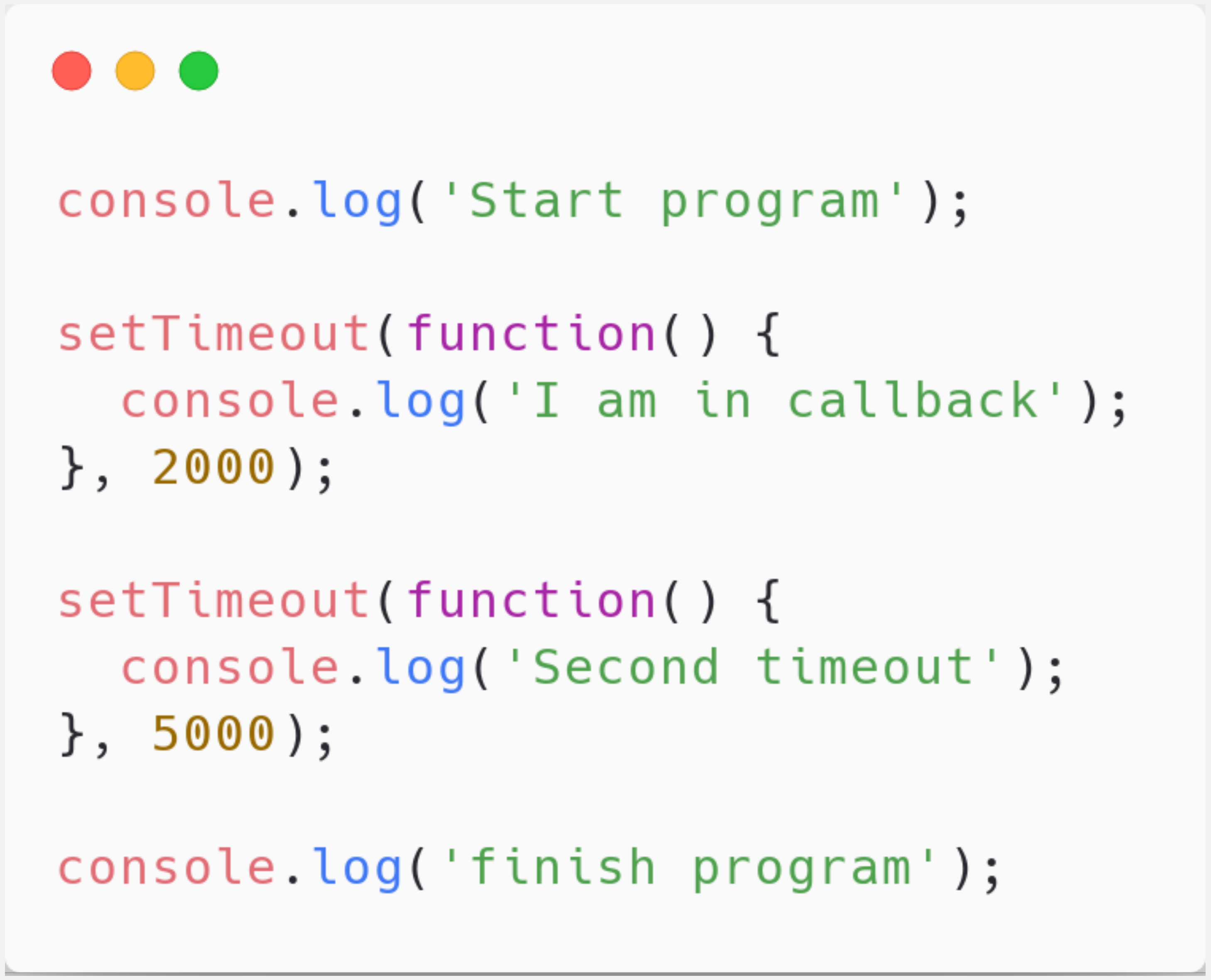
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

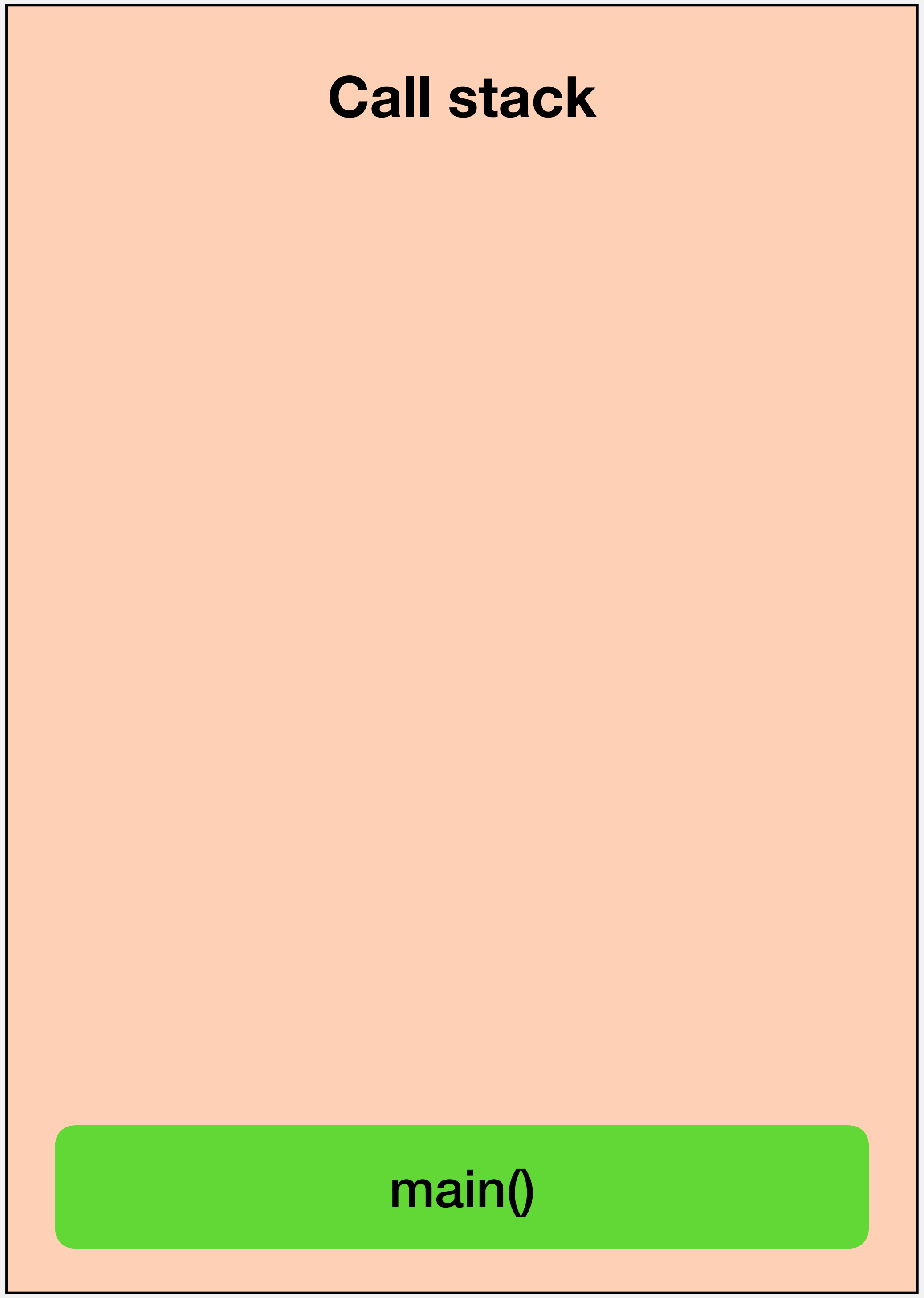


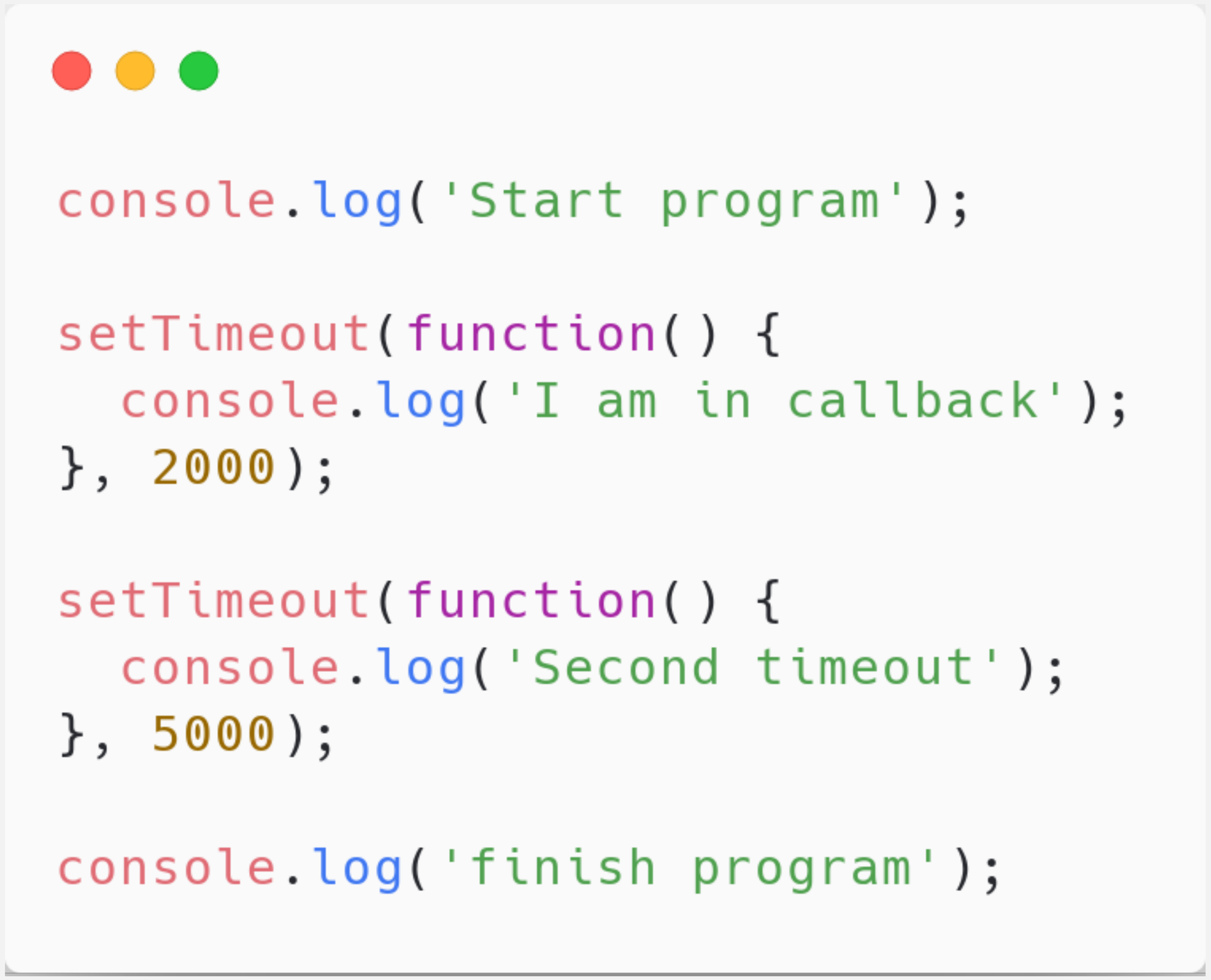


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

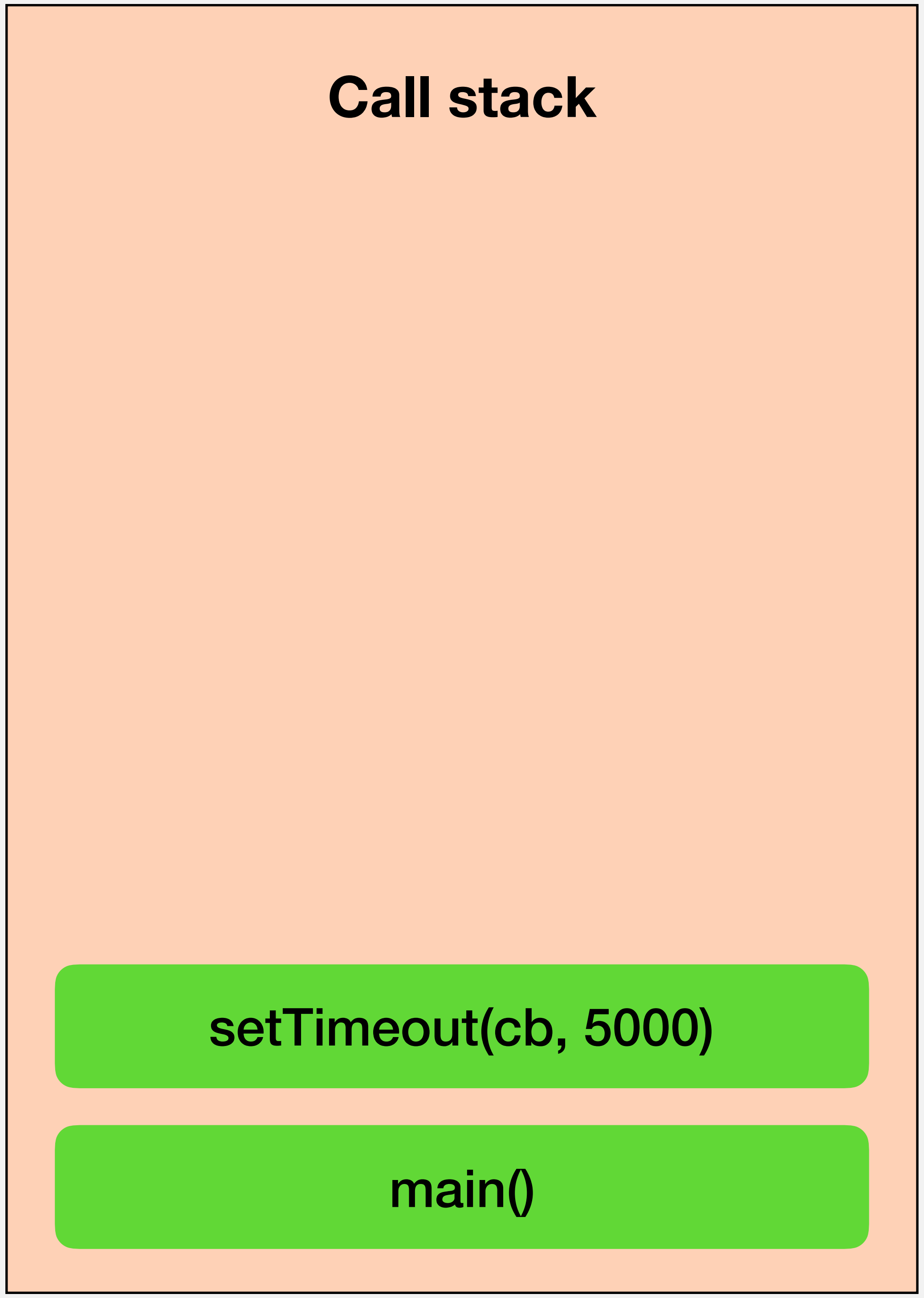


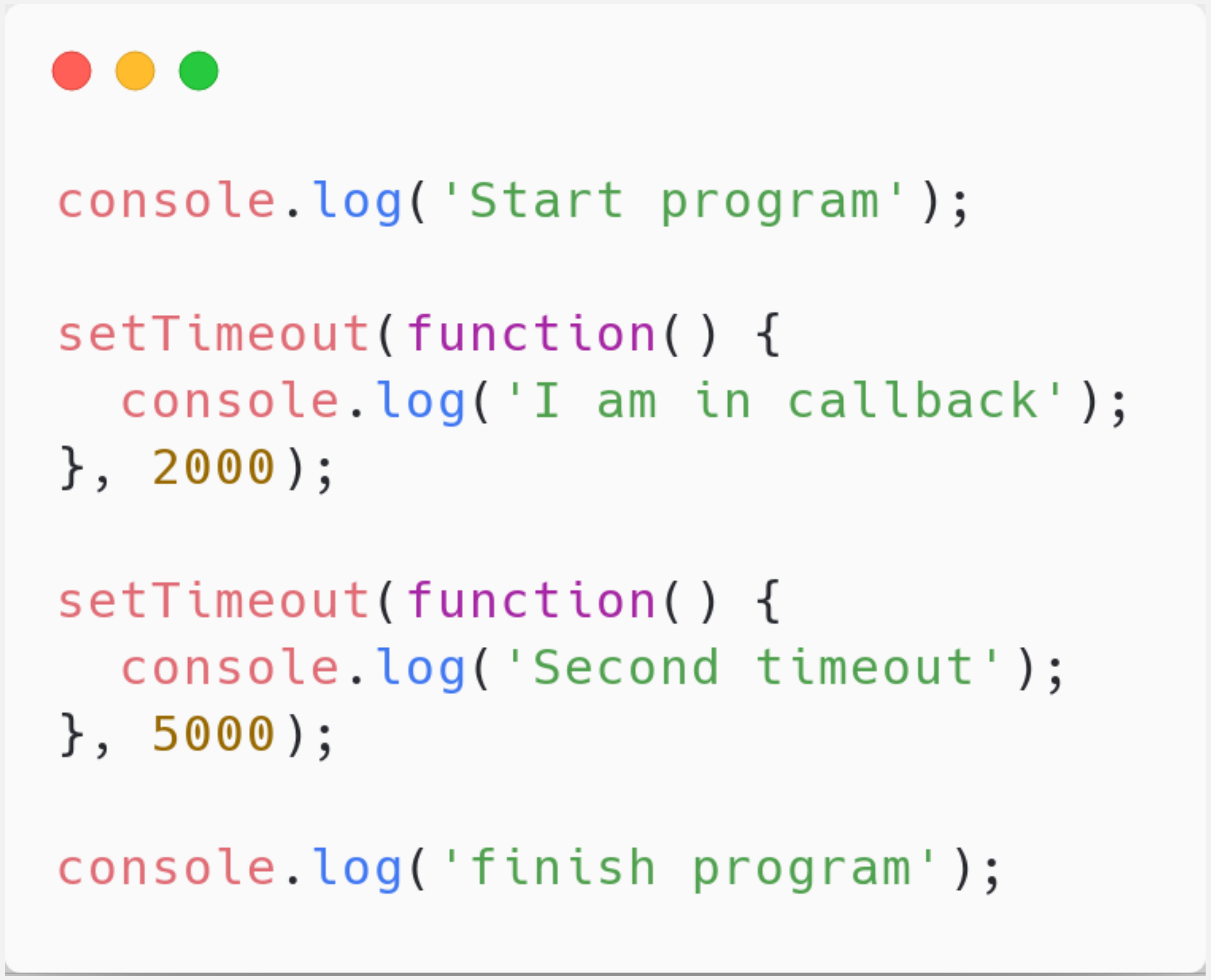

```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



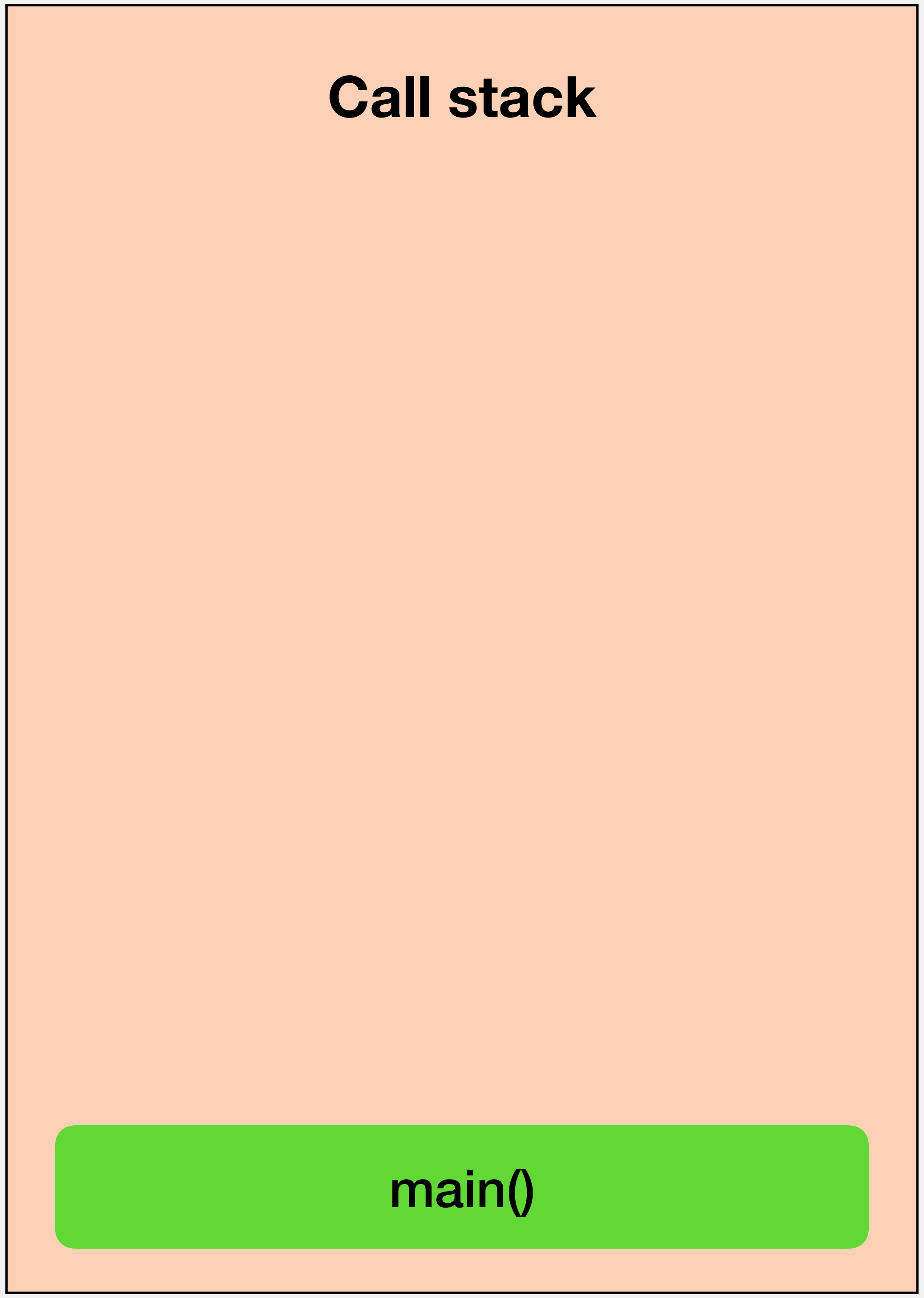


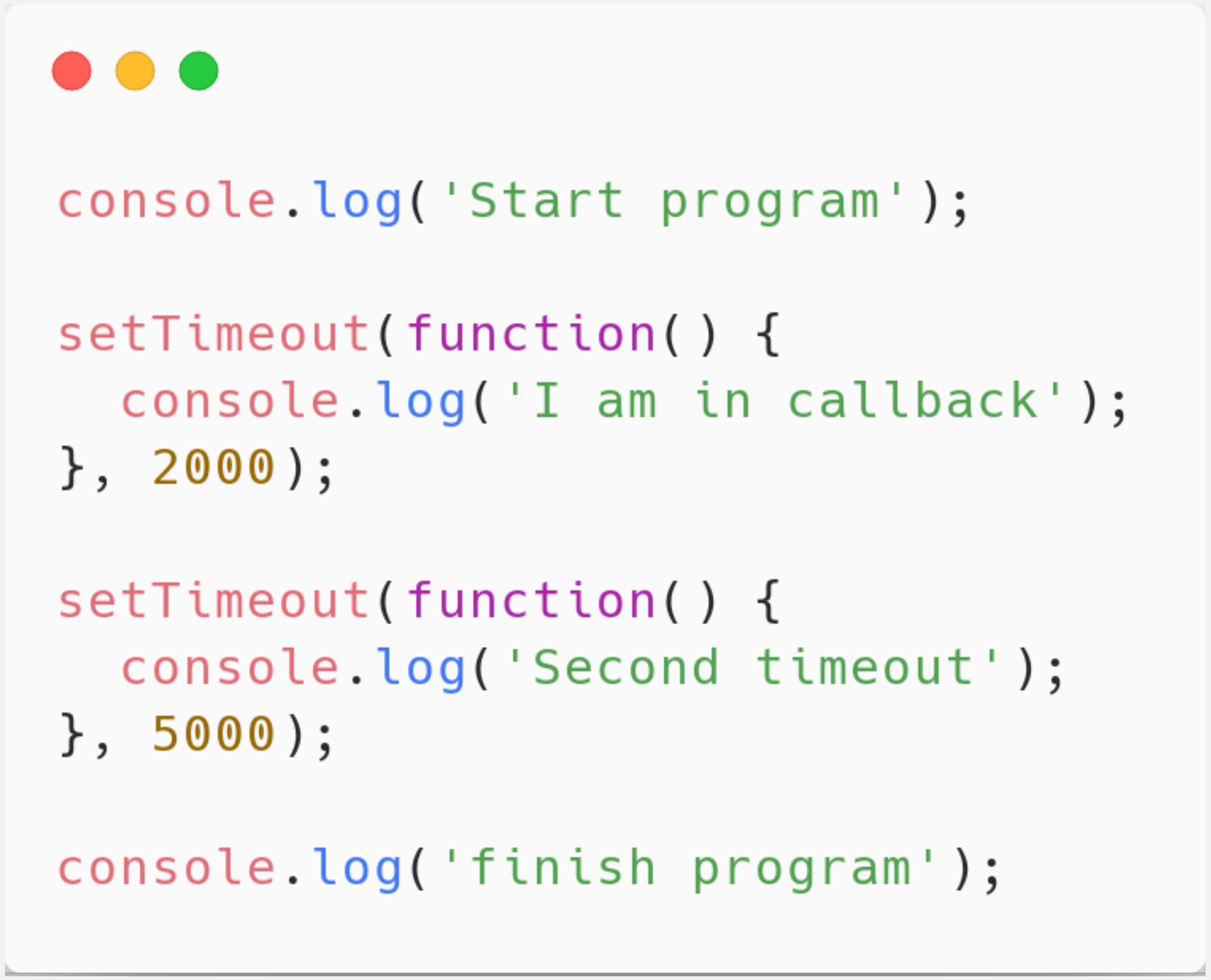
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



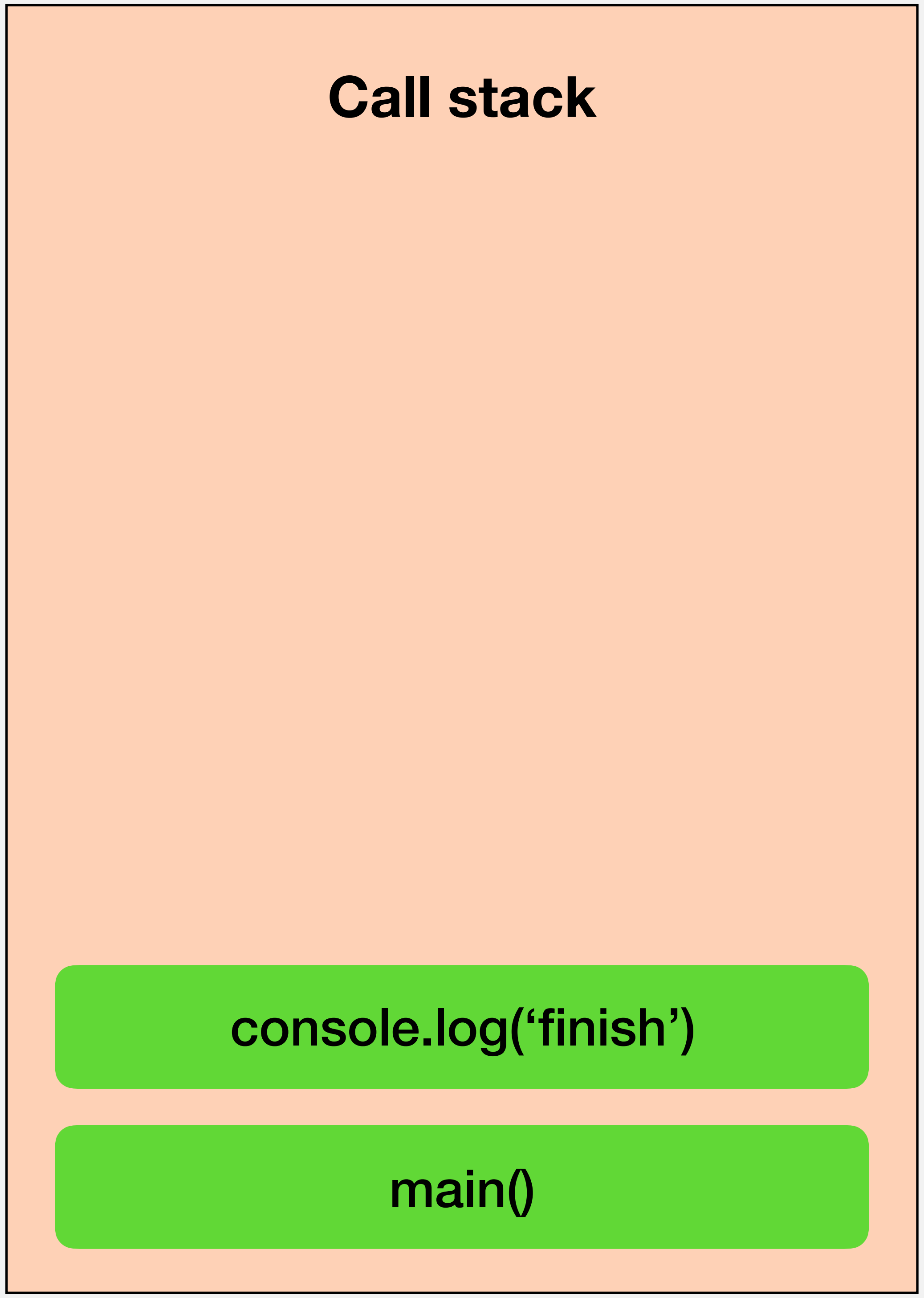


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

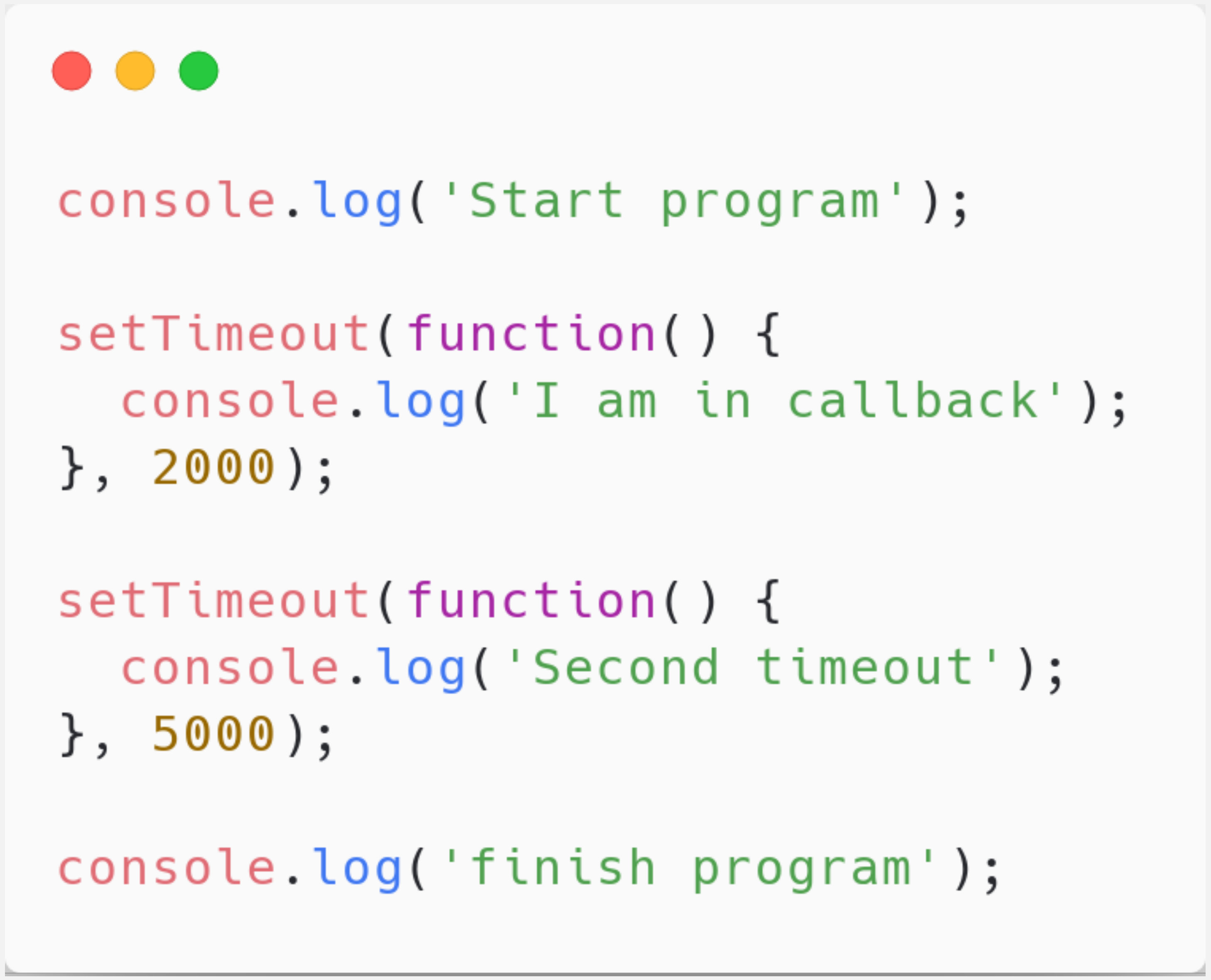


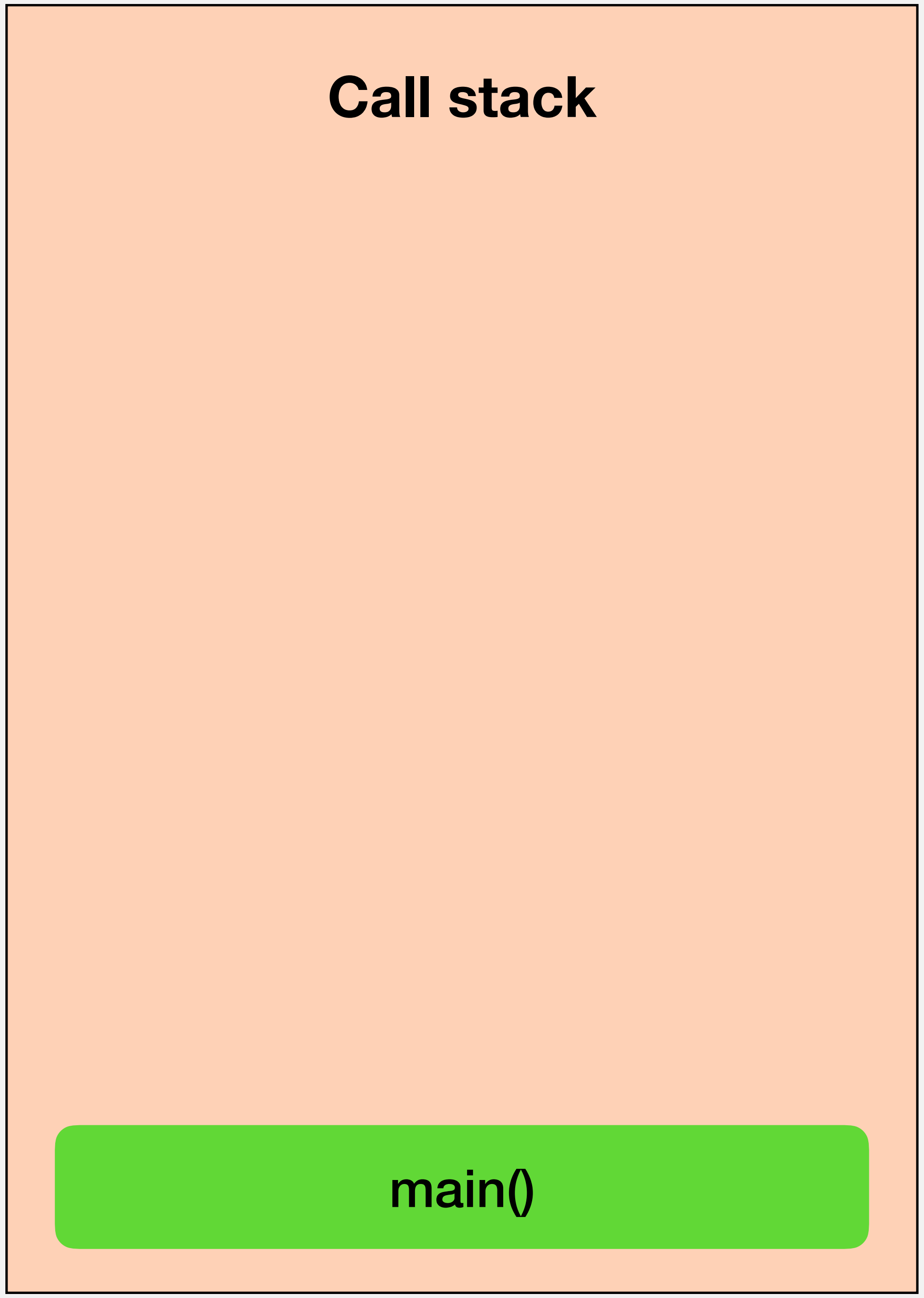


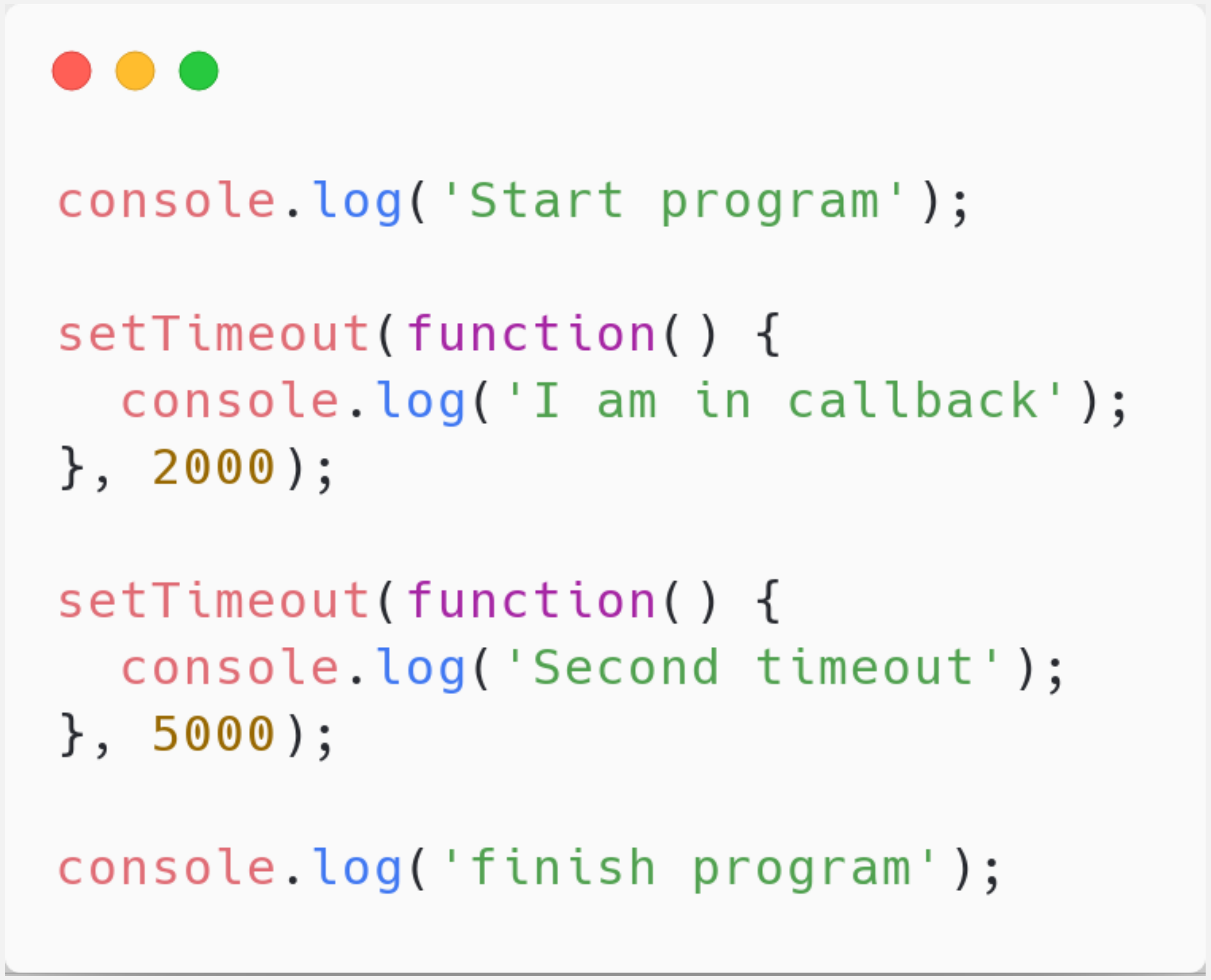
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



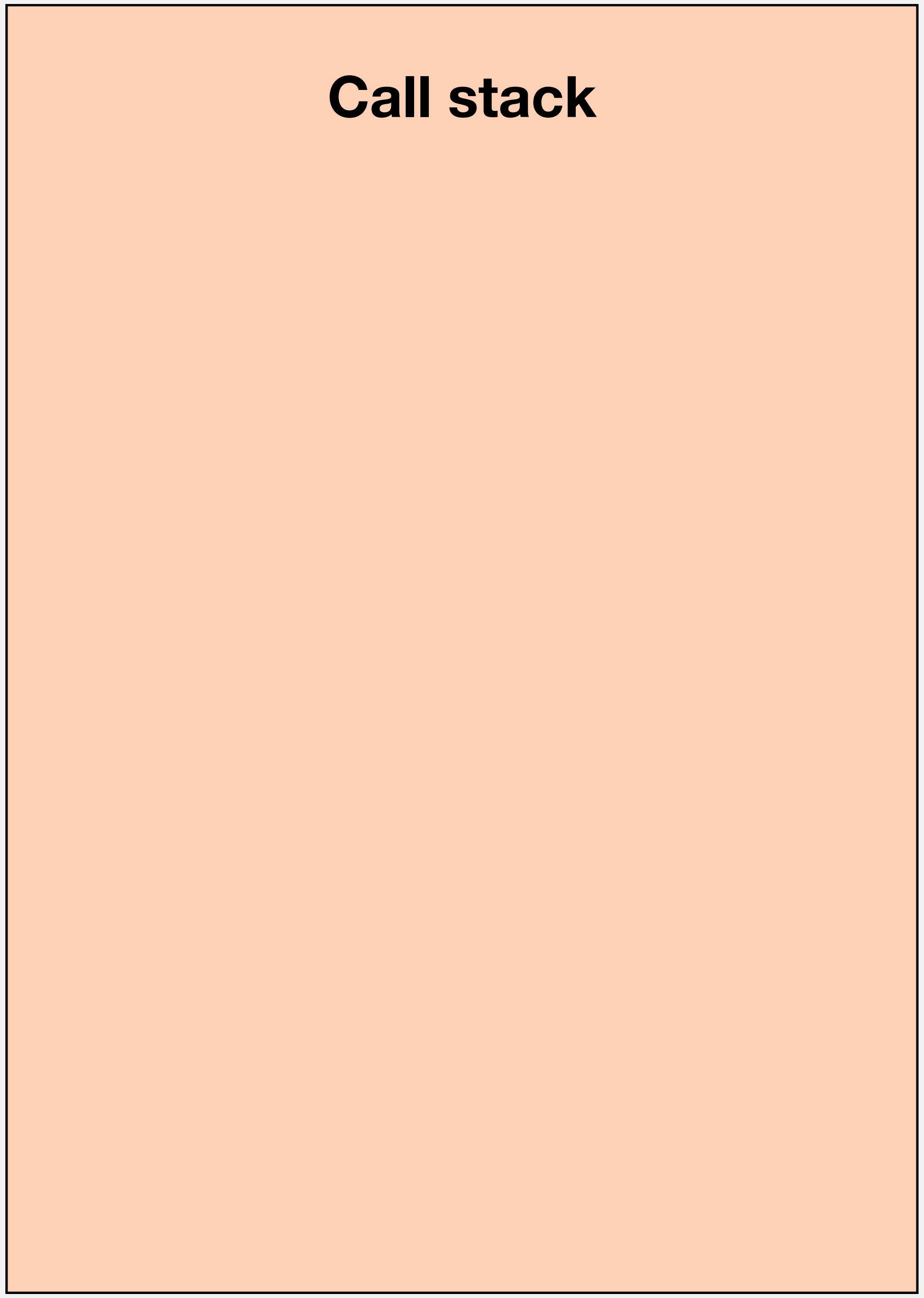
JS

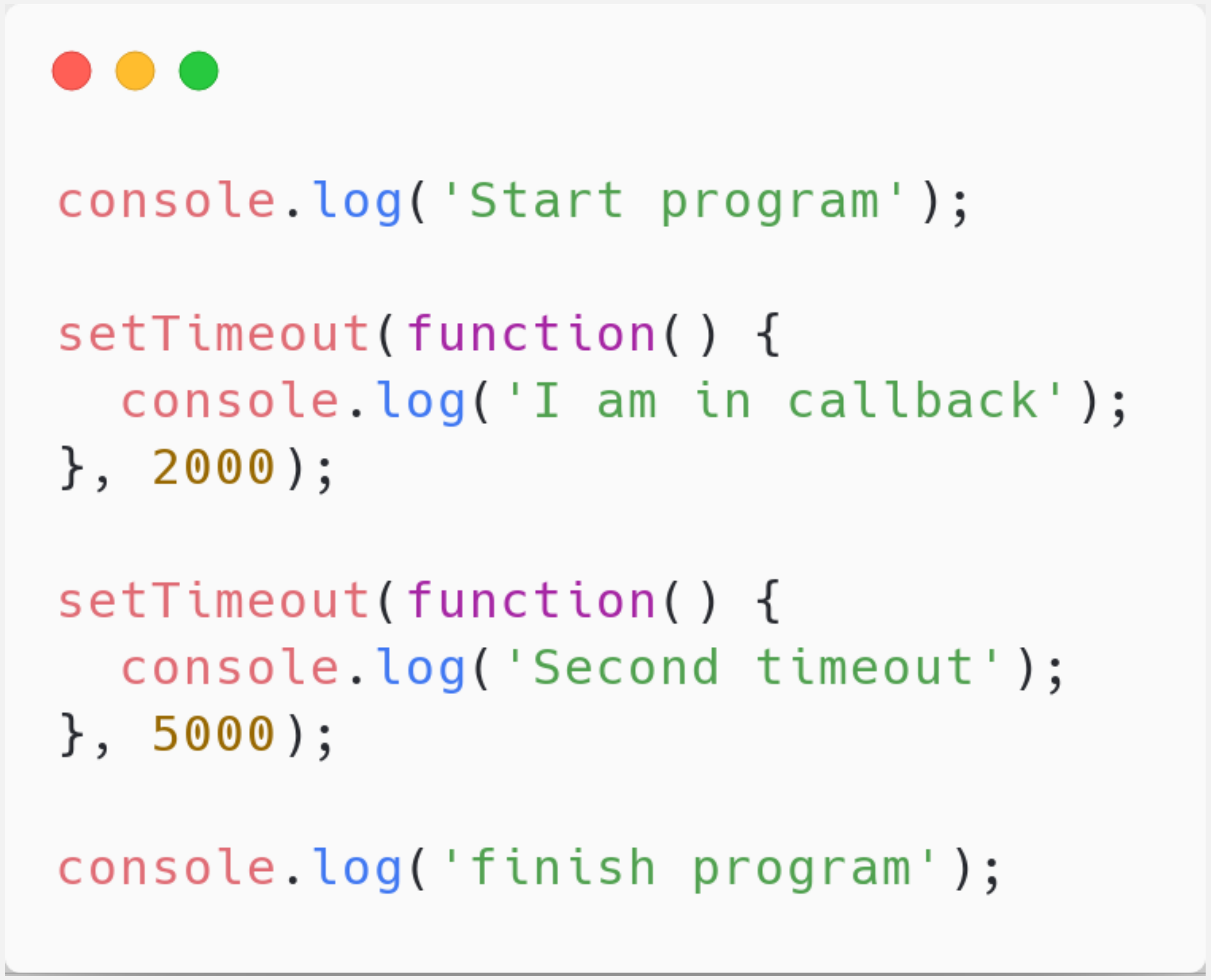

```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



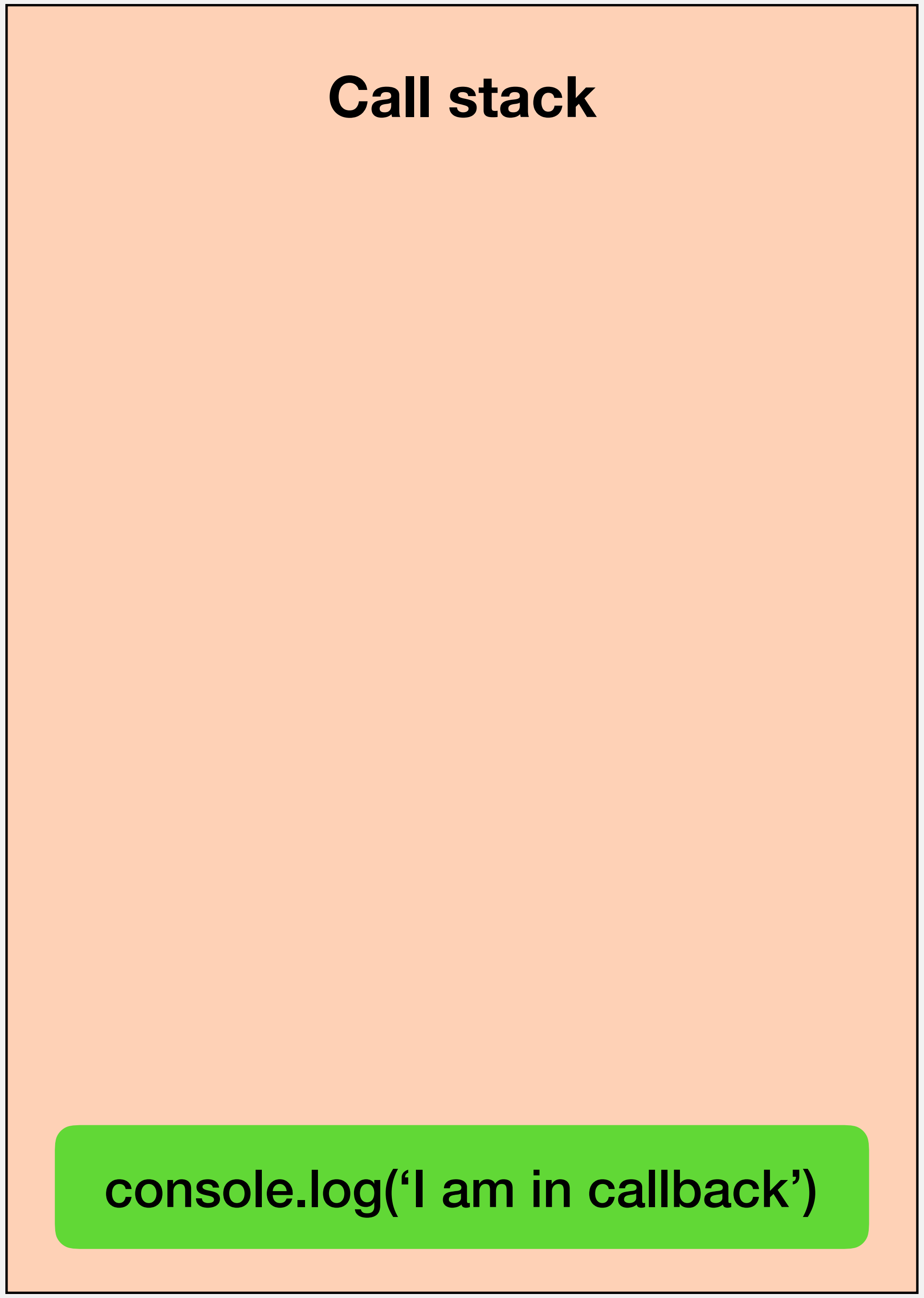


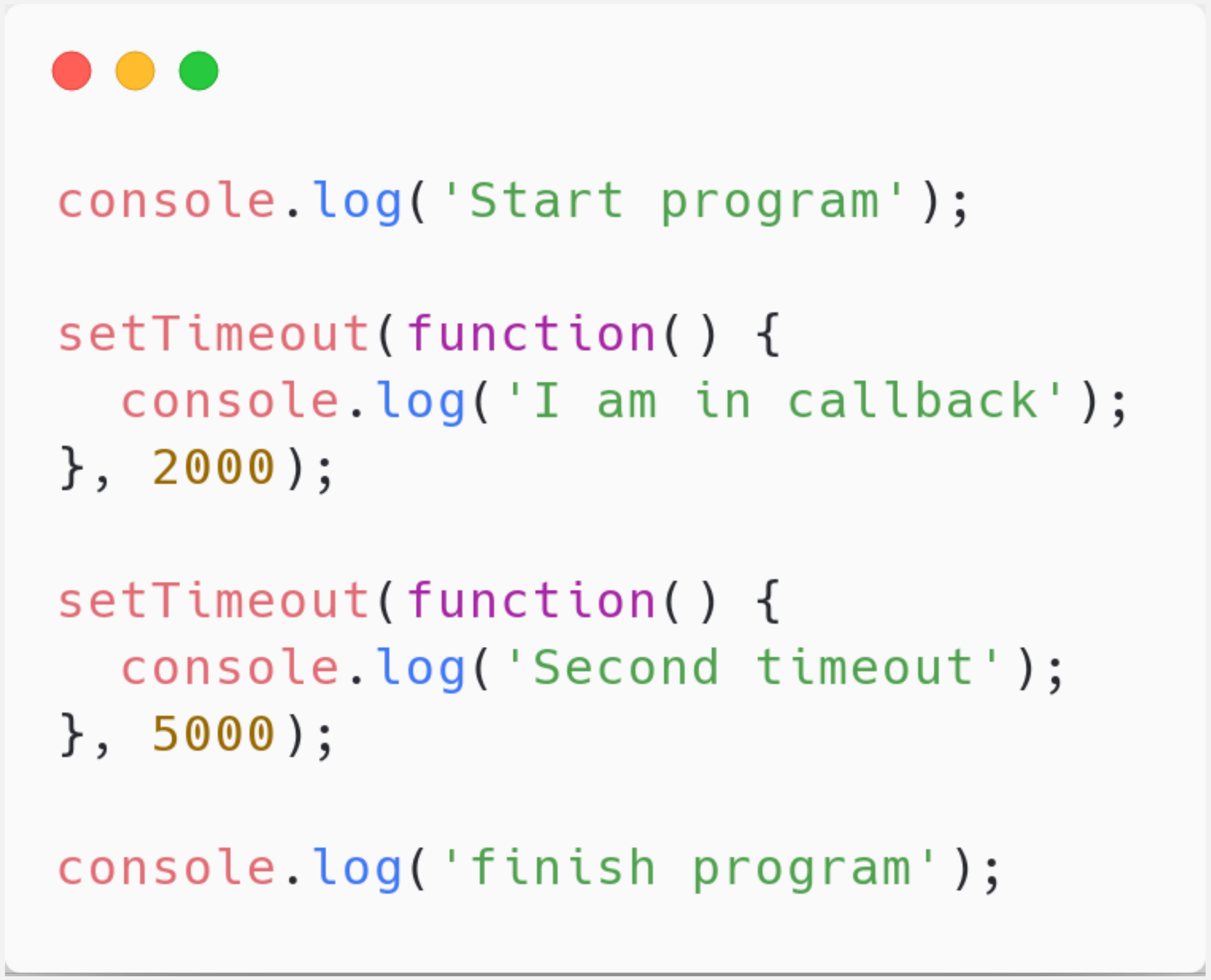
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



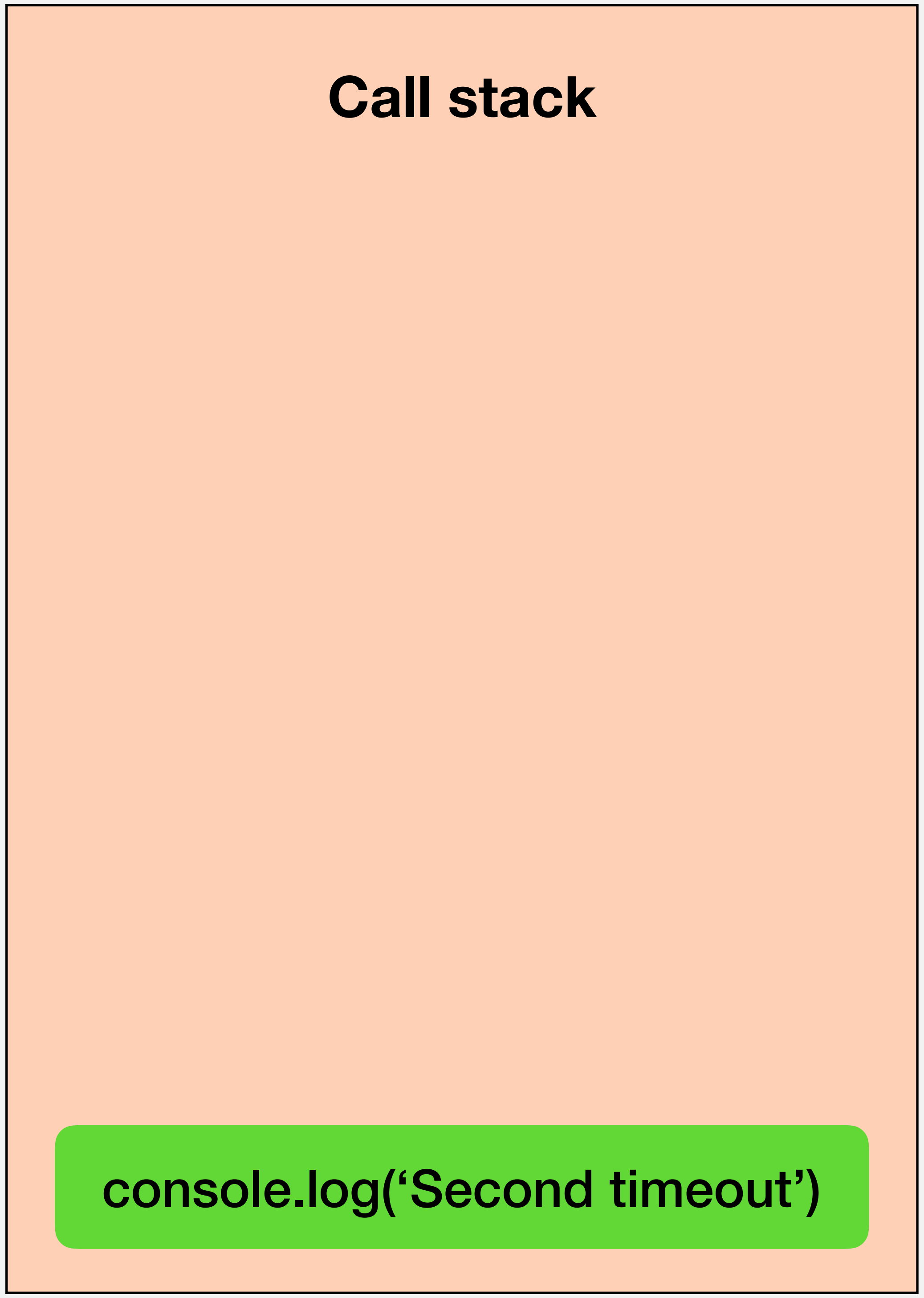


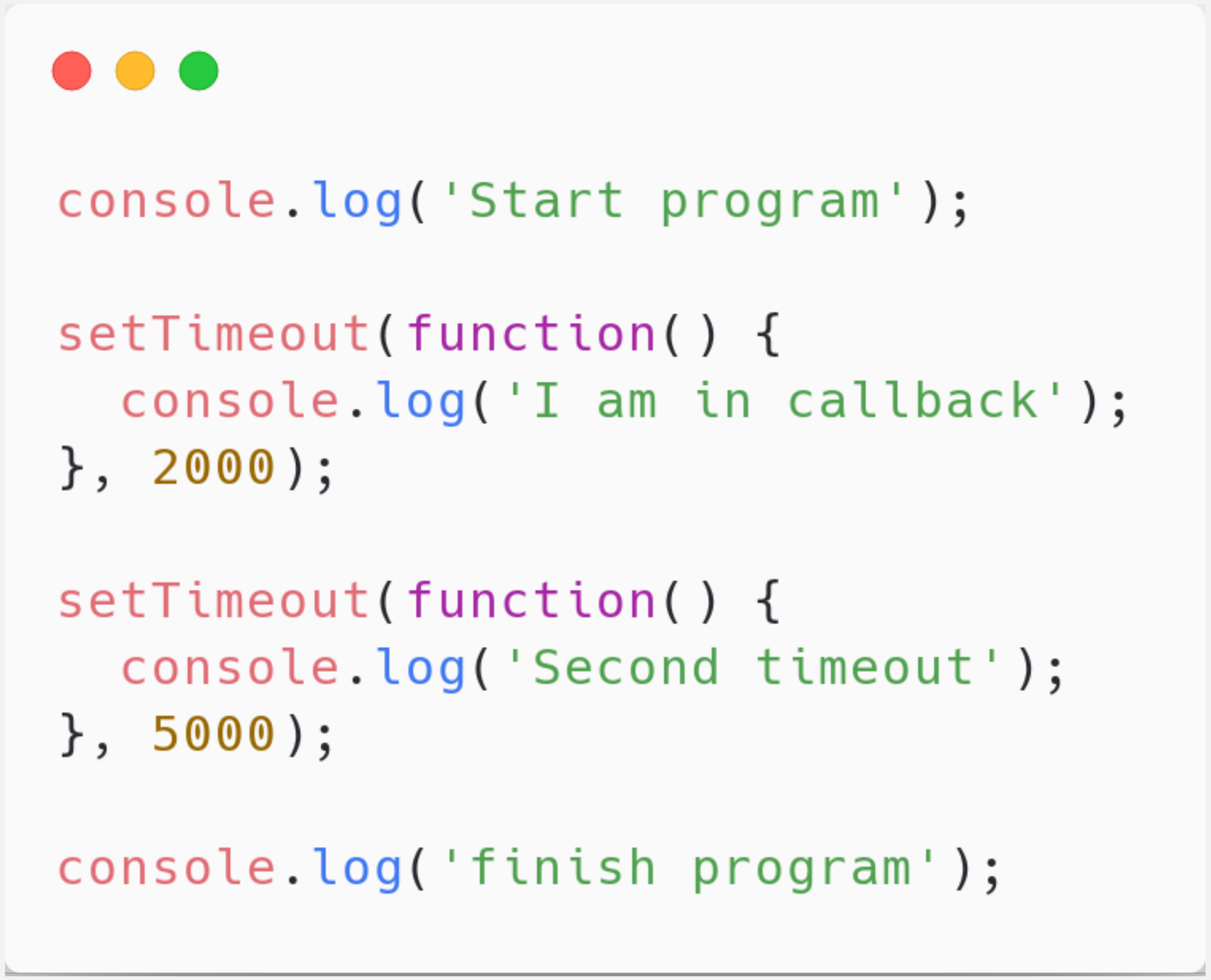
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



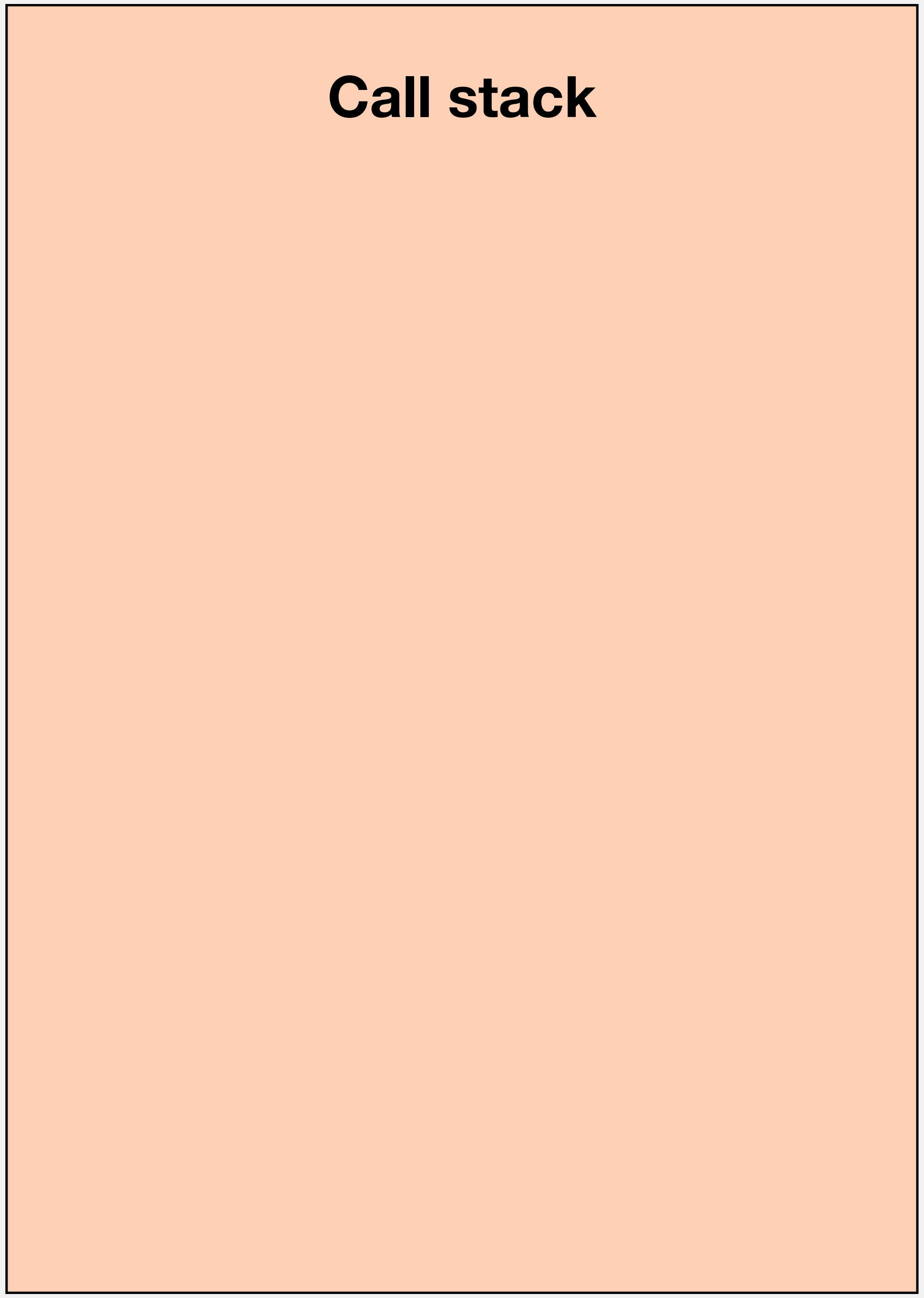


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```





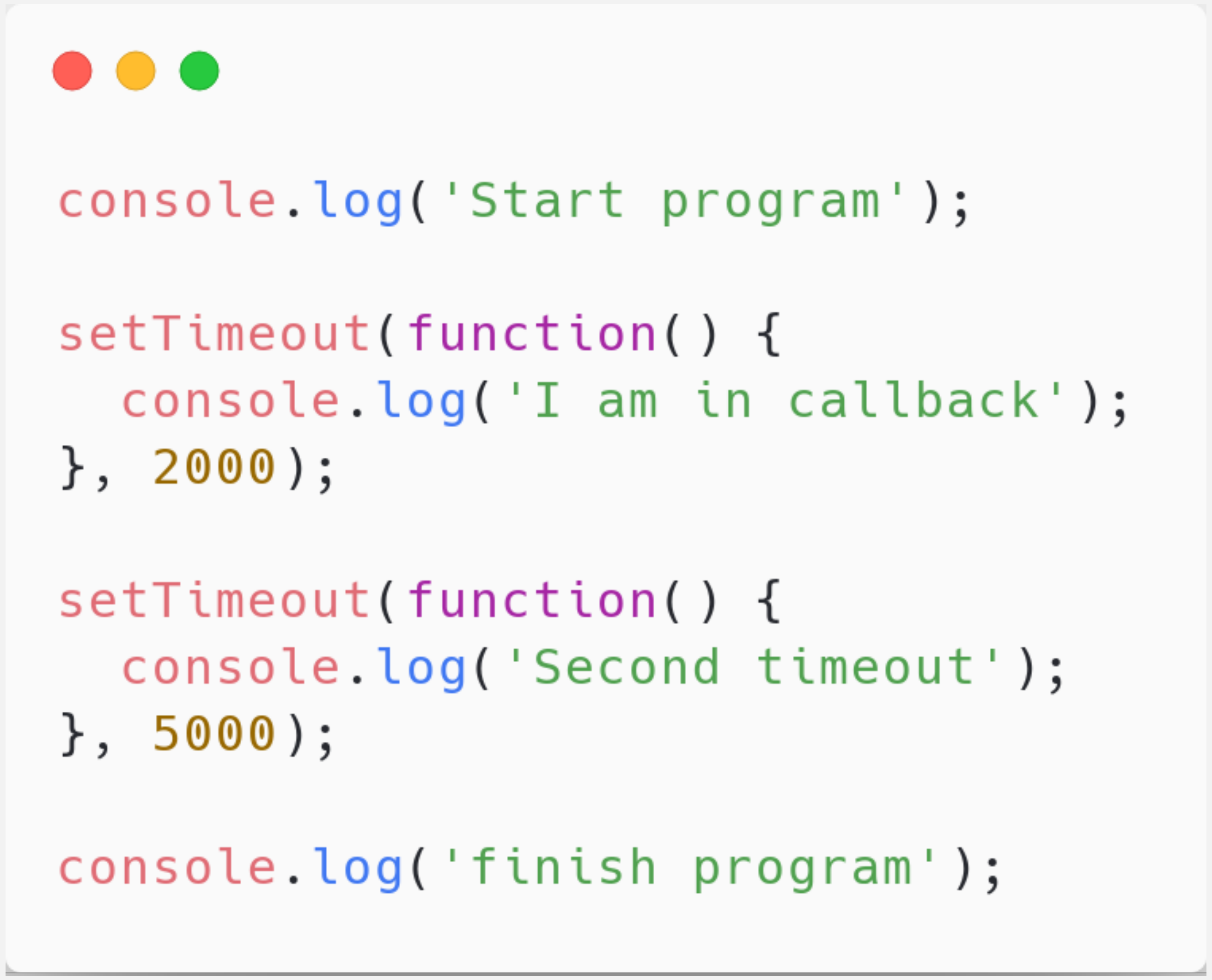
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



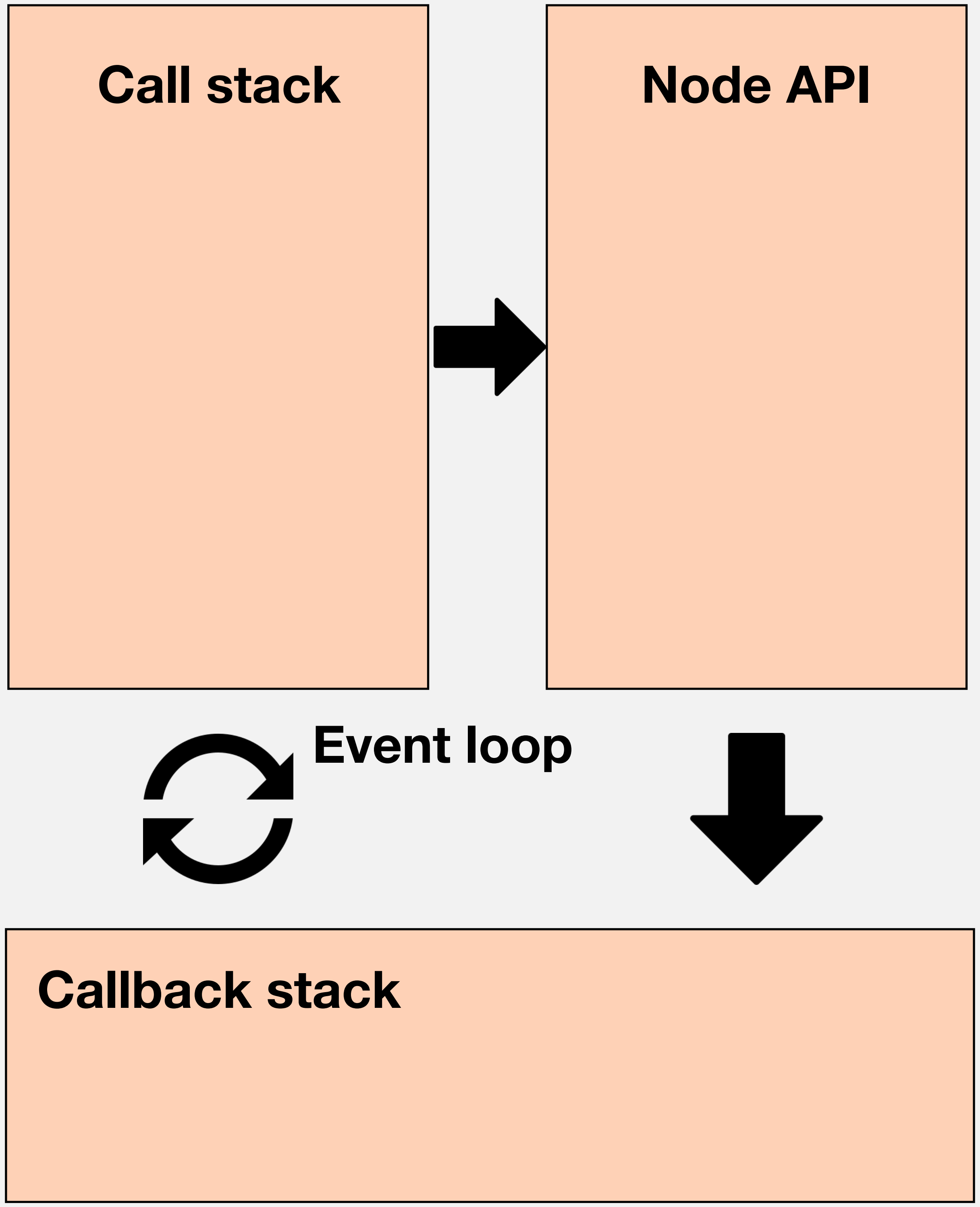
I am

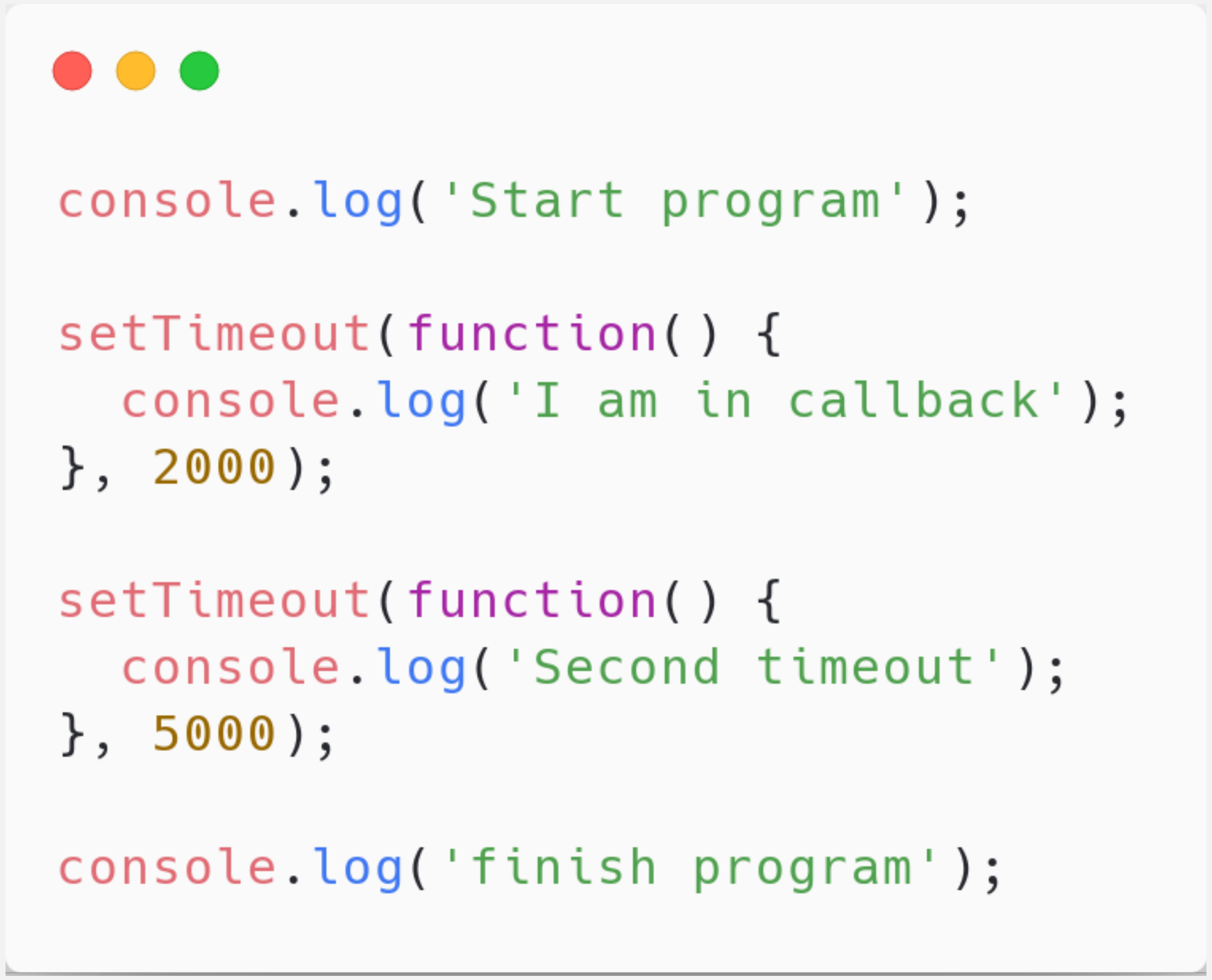
- ~~Single threaded~~
- ~~Non-blocking~~
- ~~Asynchronous~~
- Concurrent

Concurrency and event loop

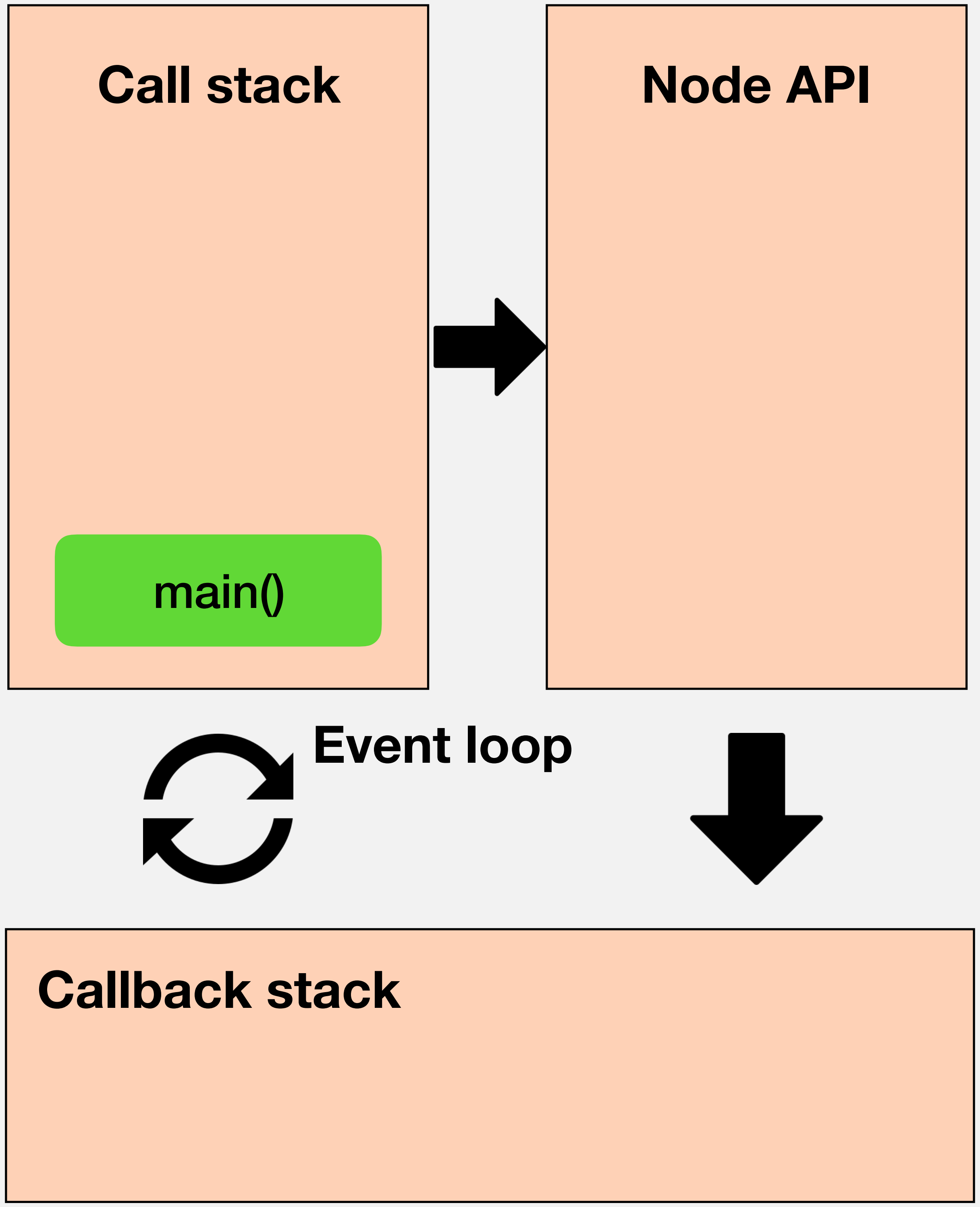


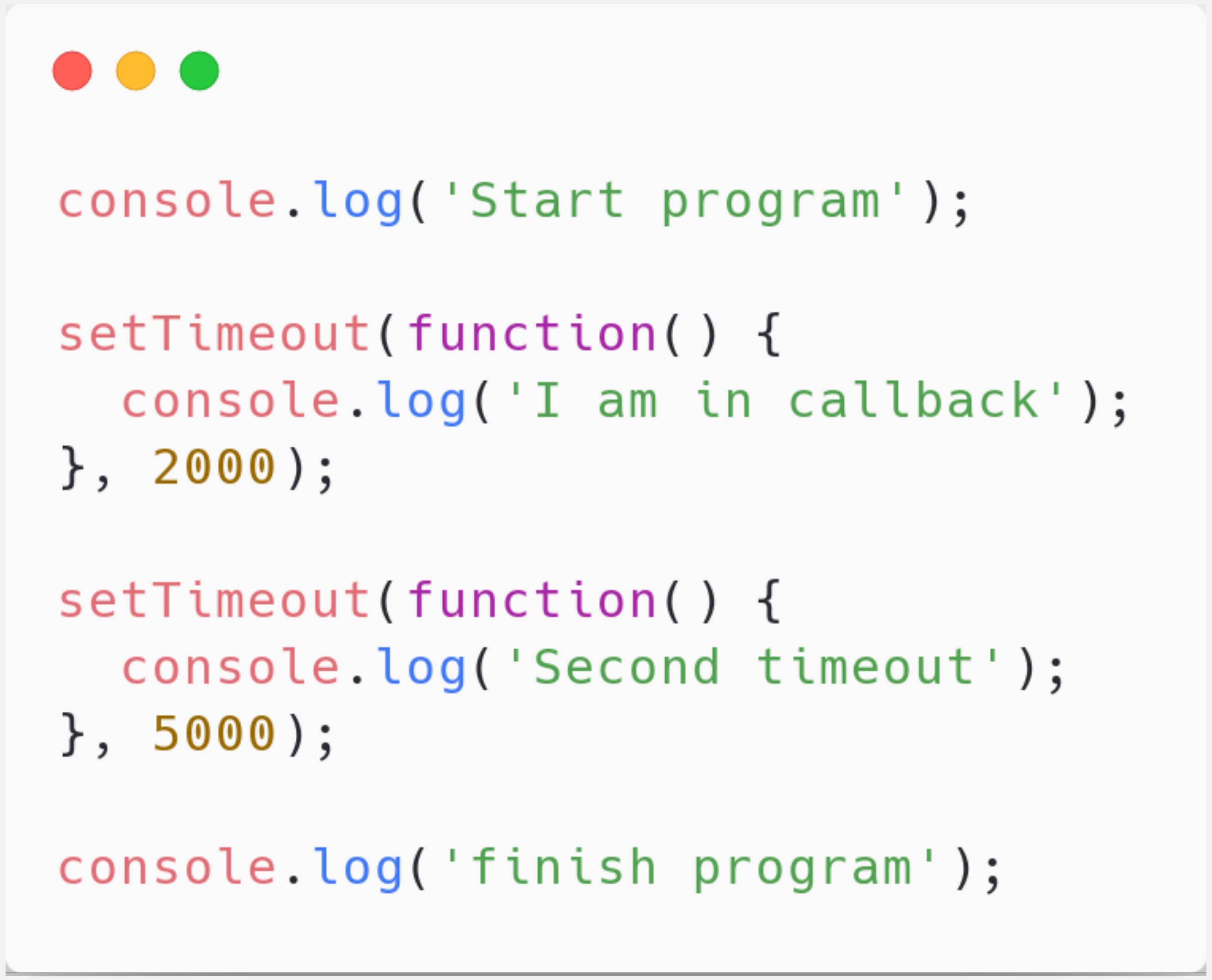
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



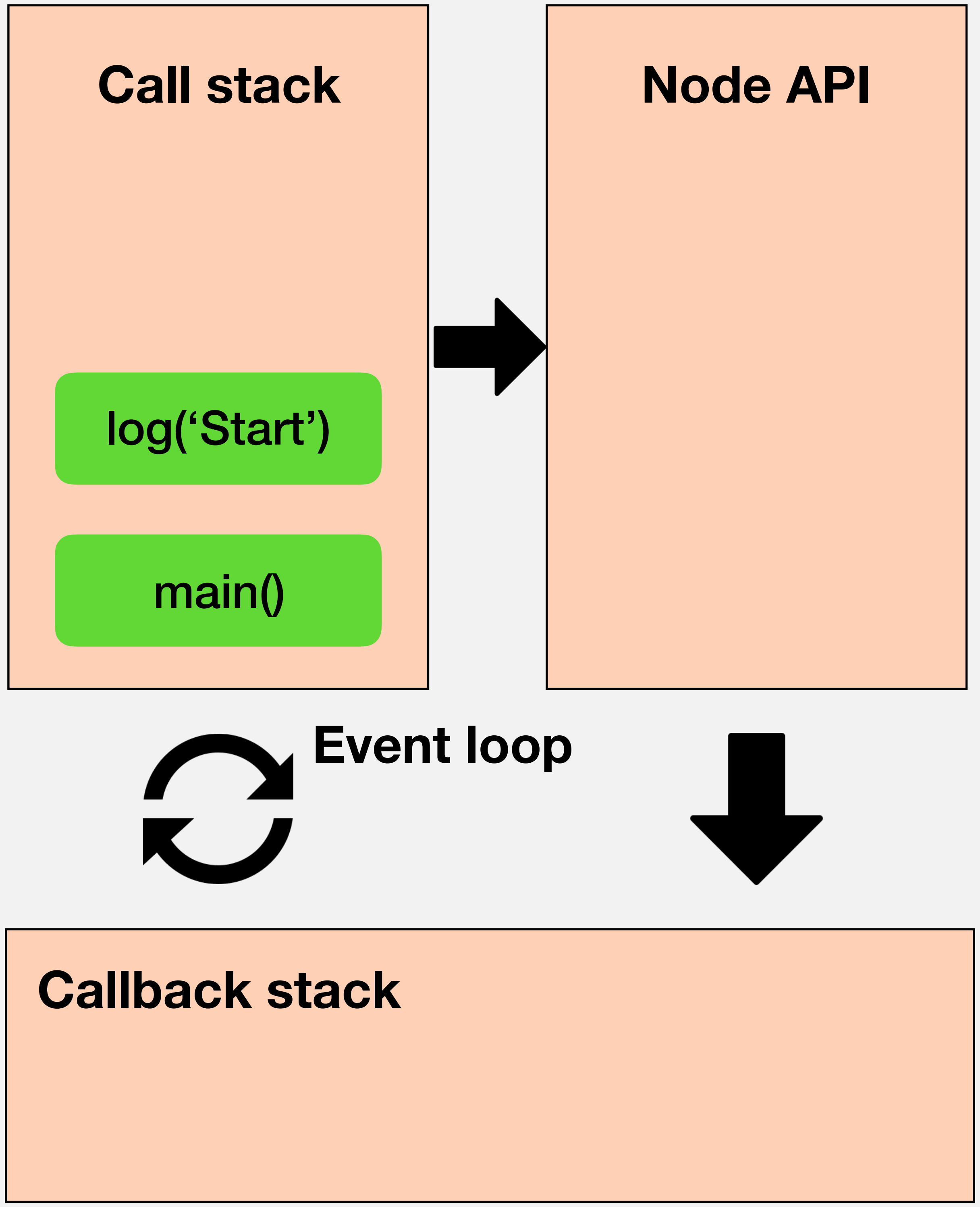


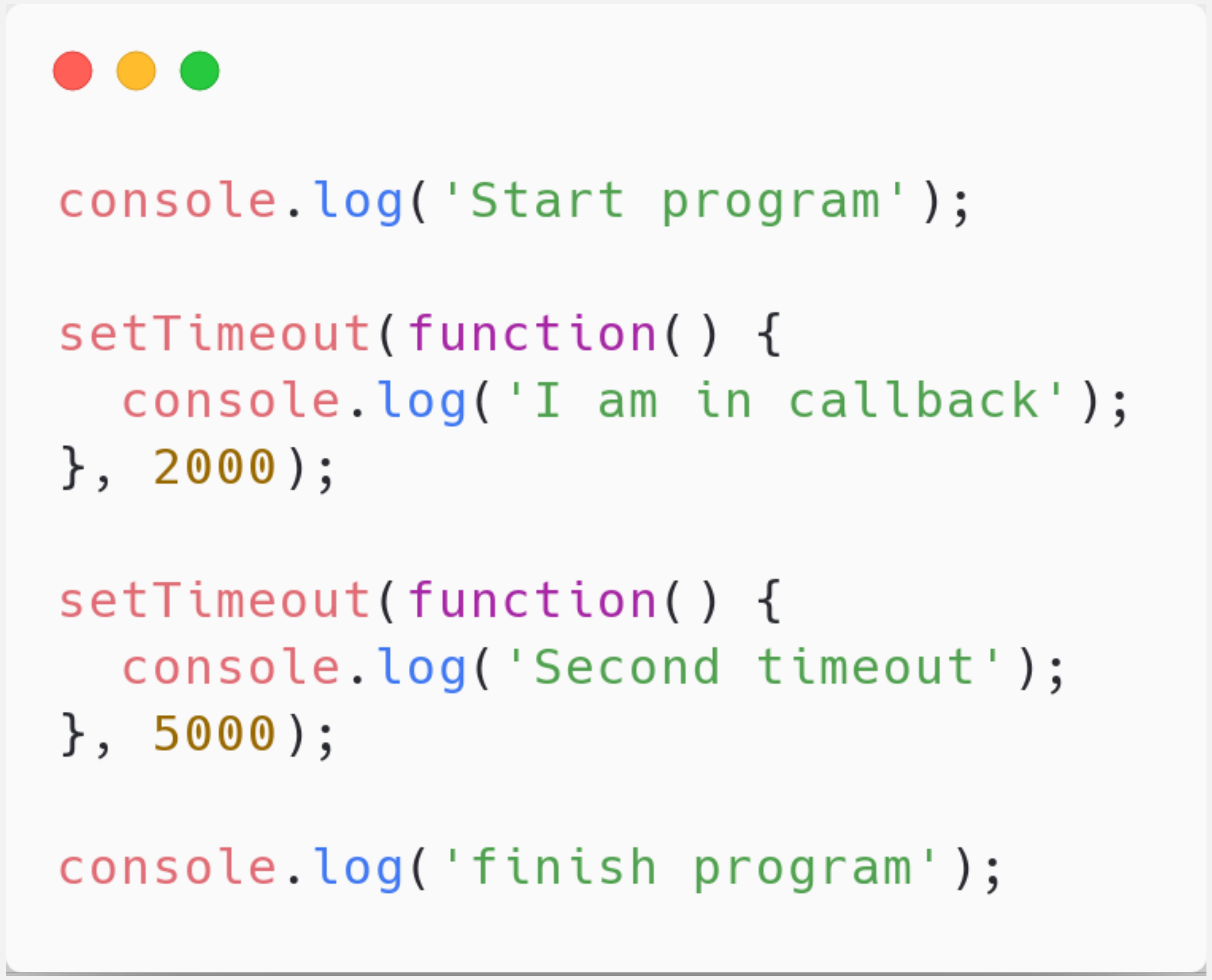
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



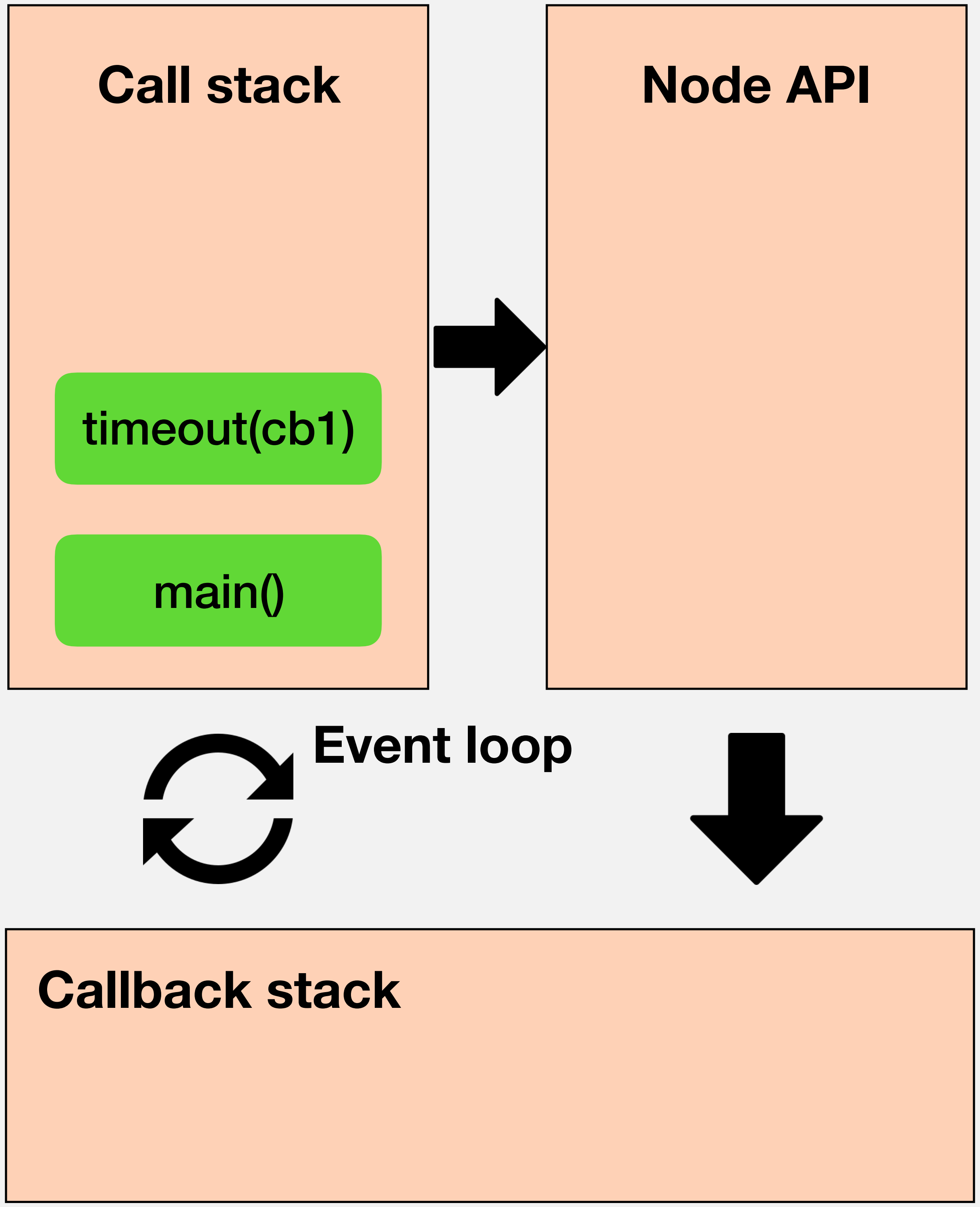


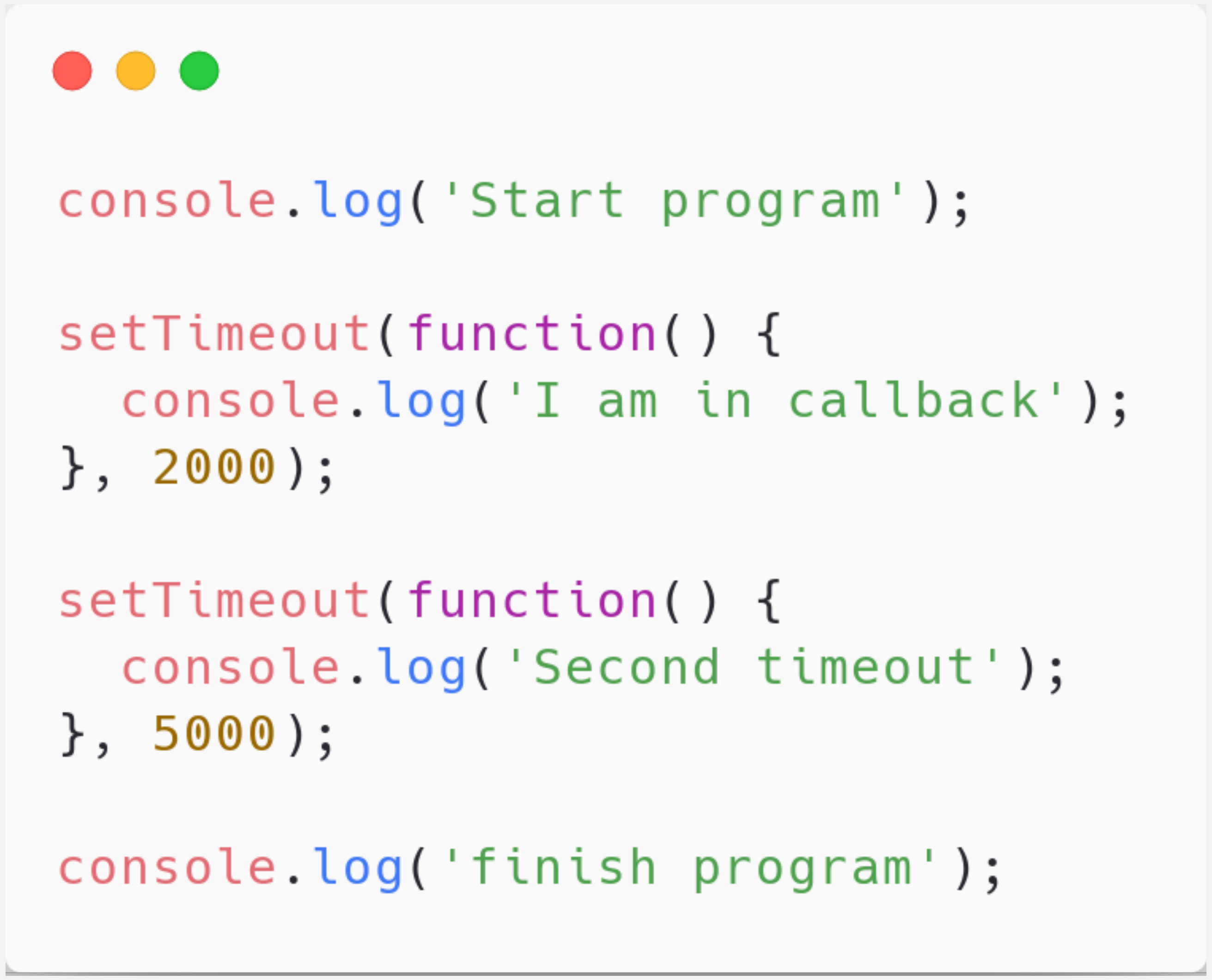
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



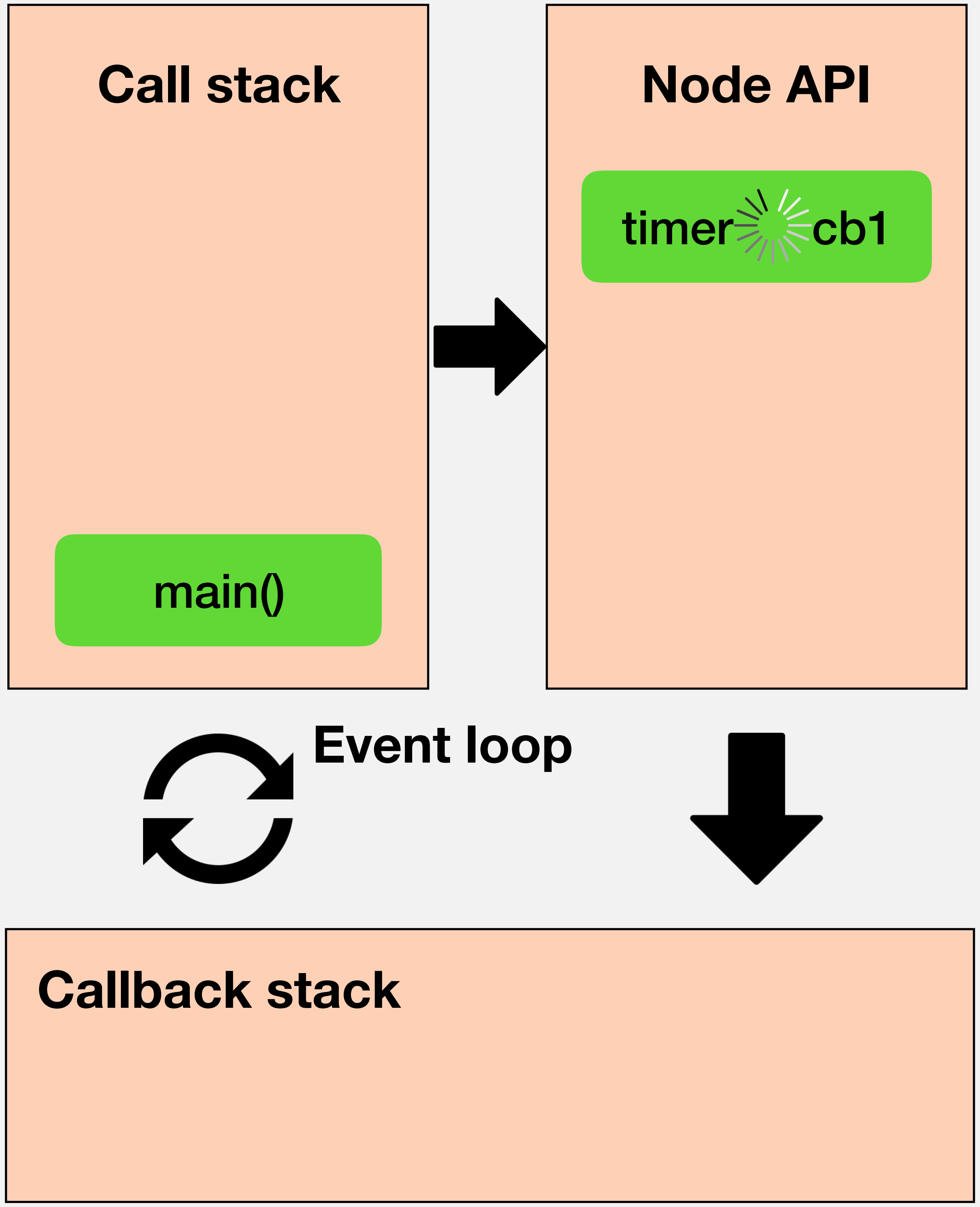


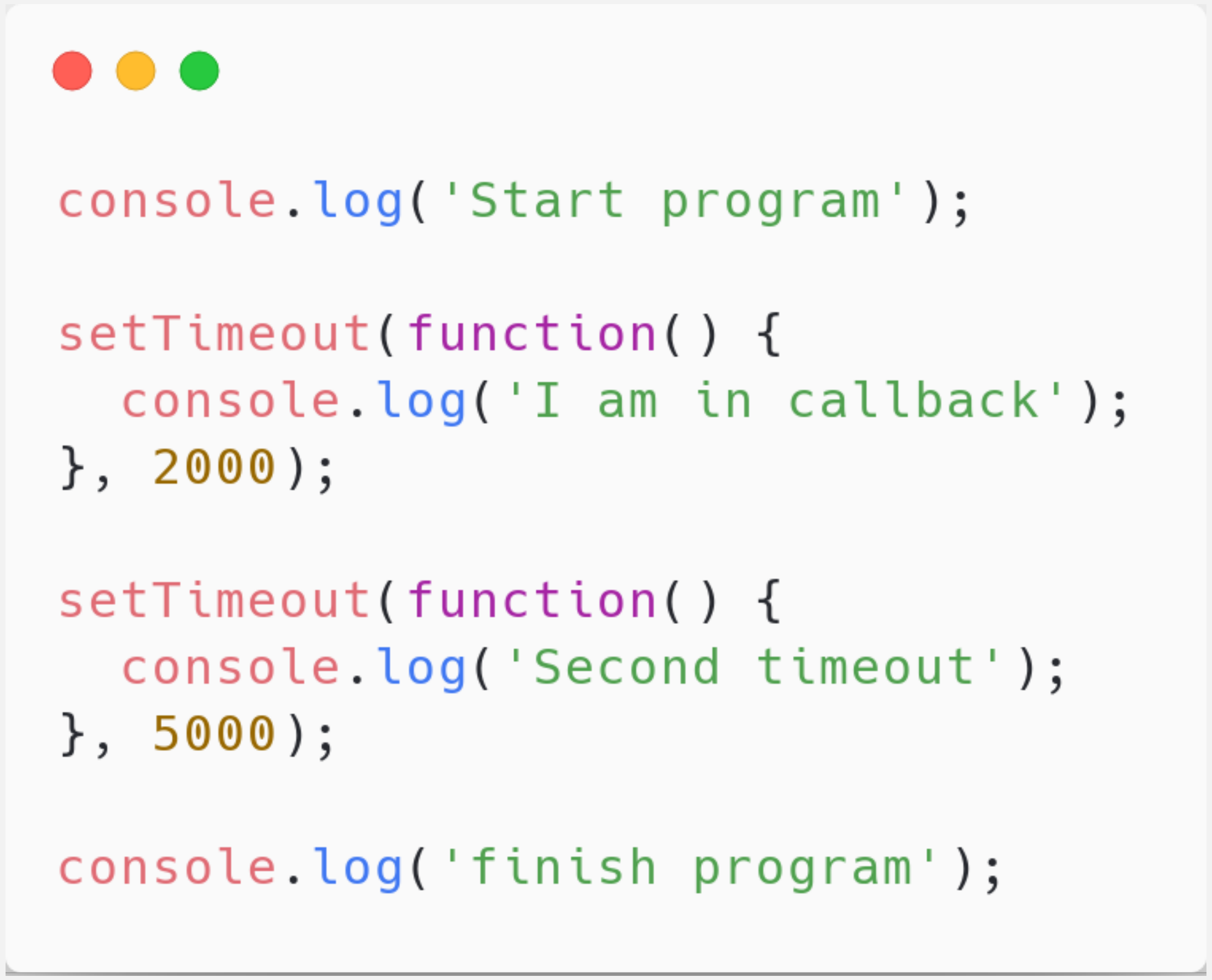
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



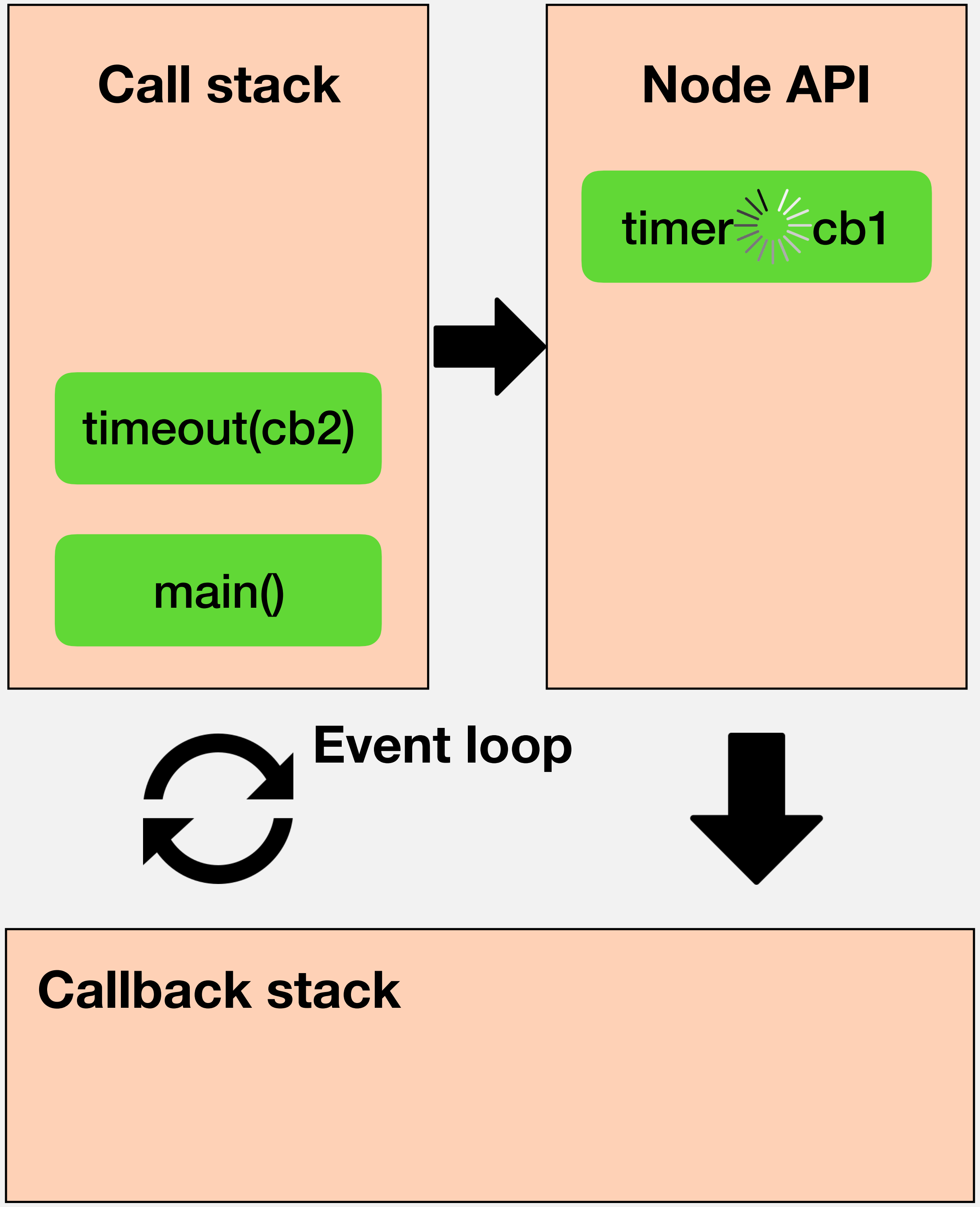


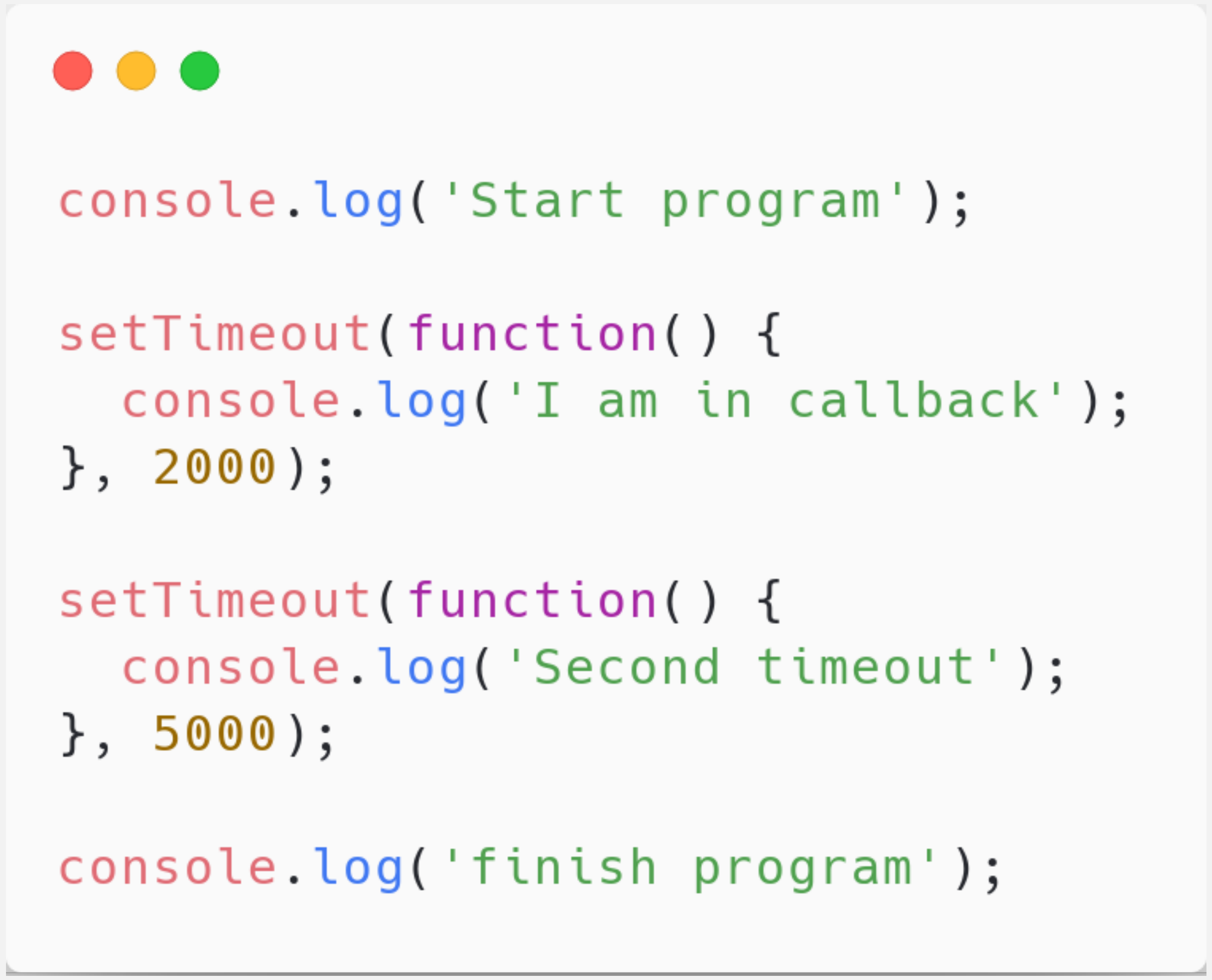
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



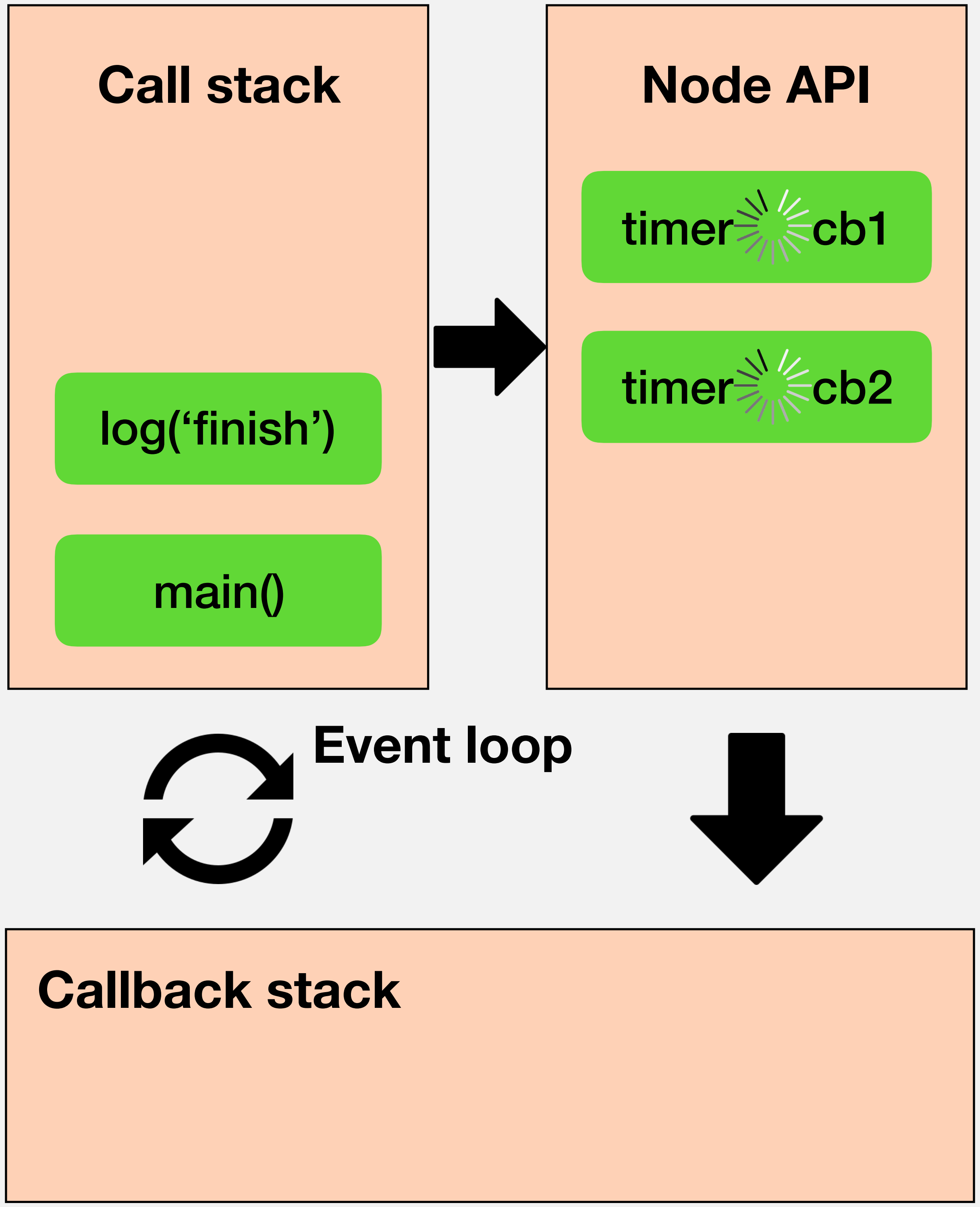


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

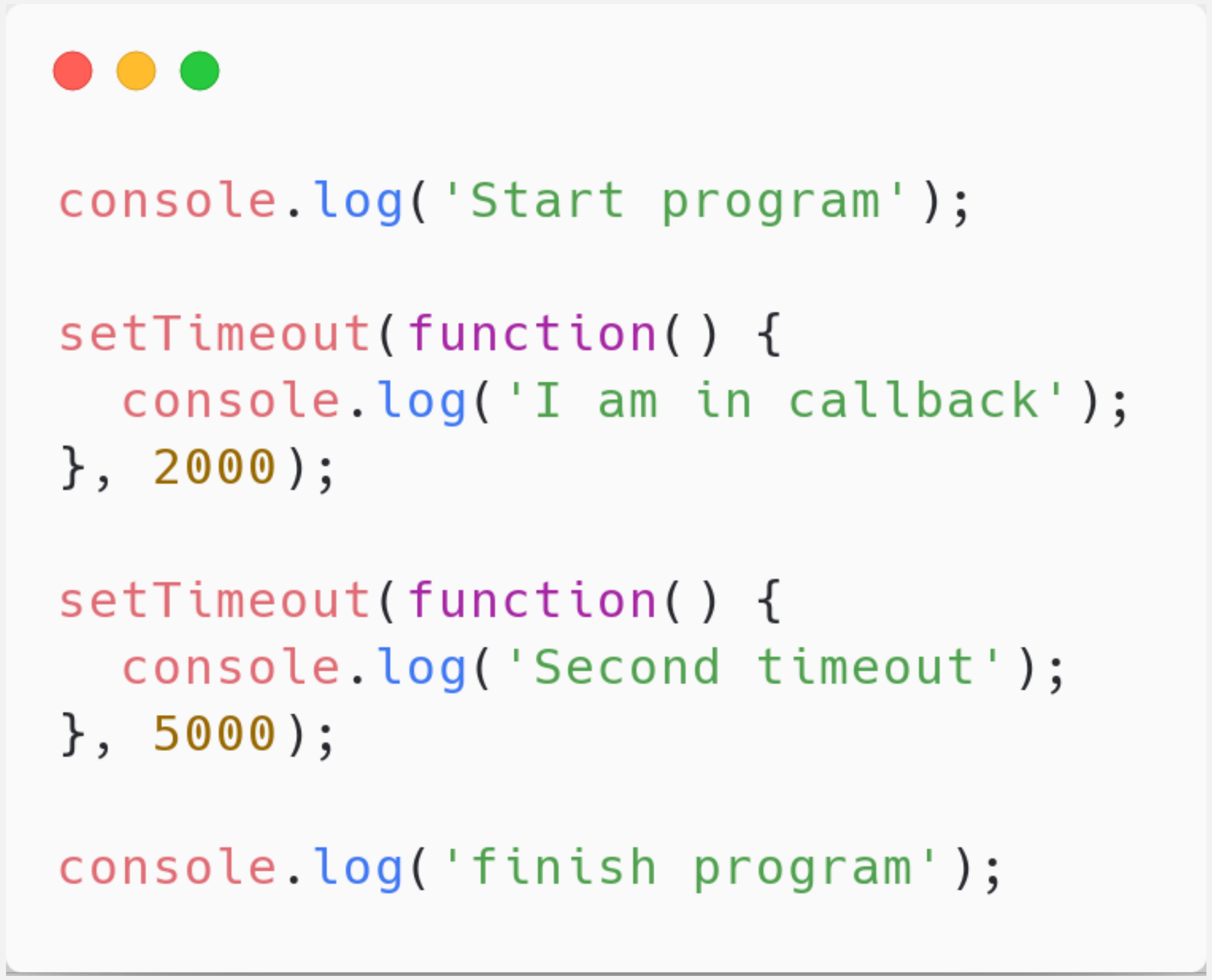




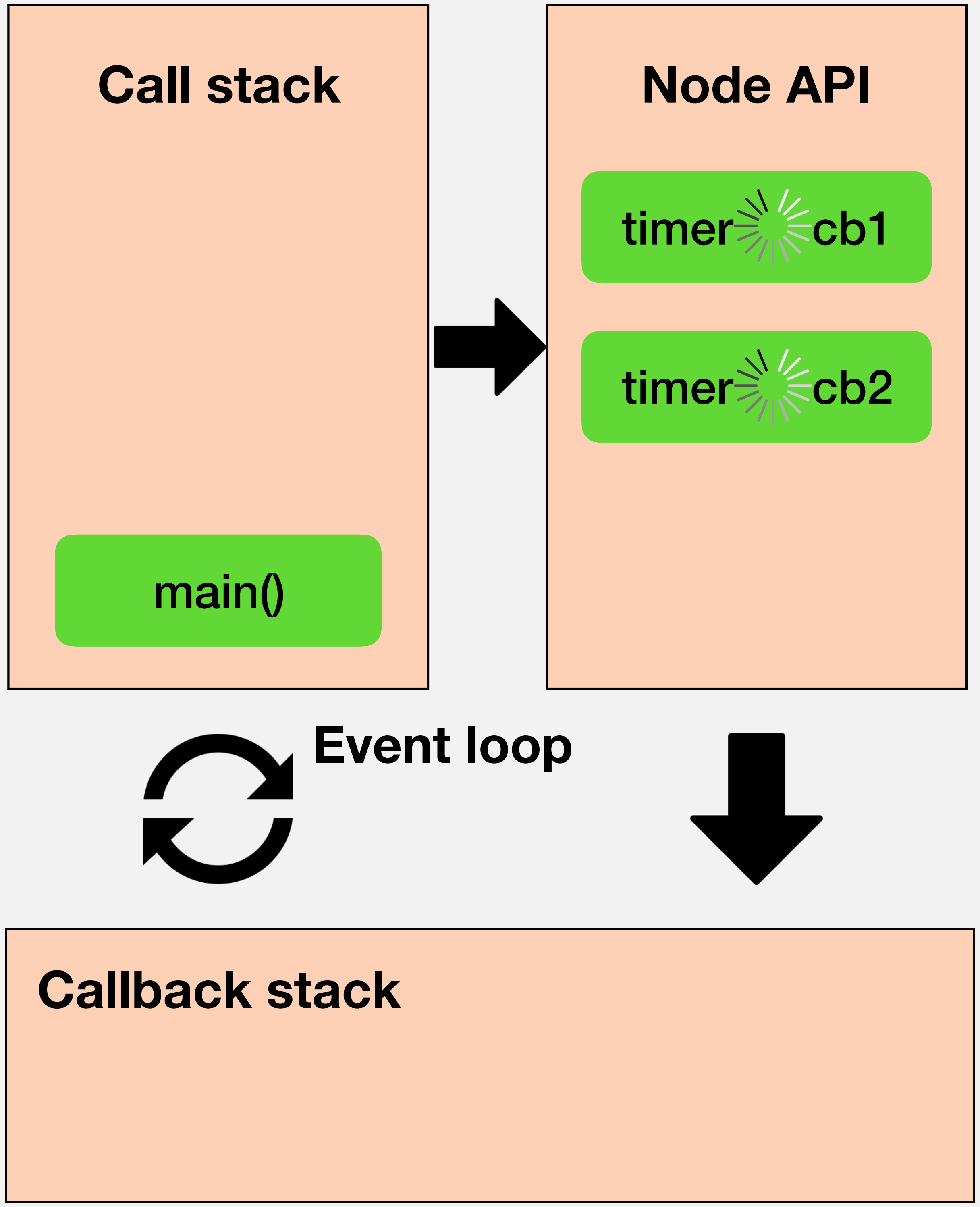
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

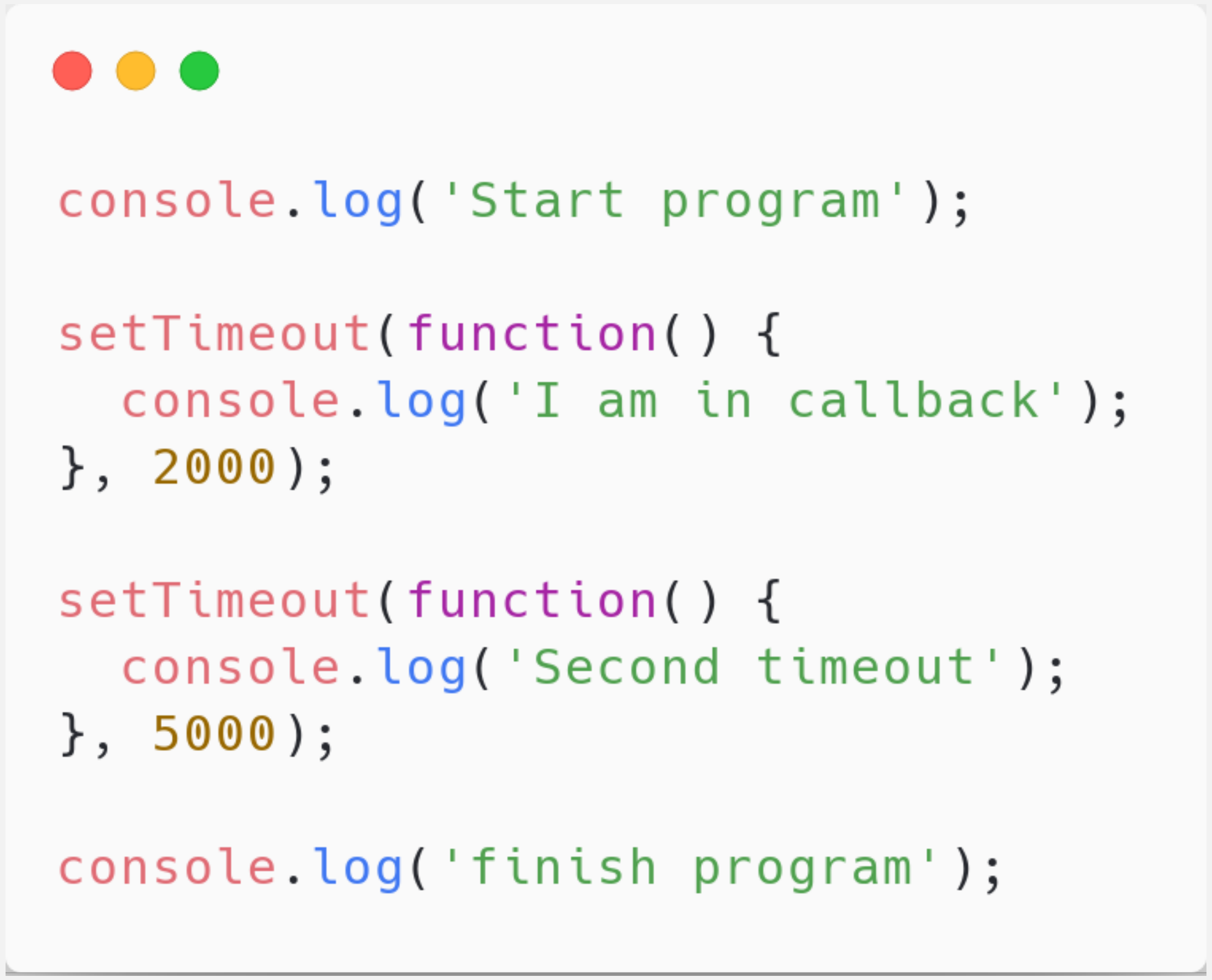


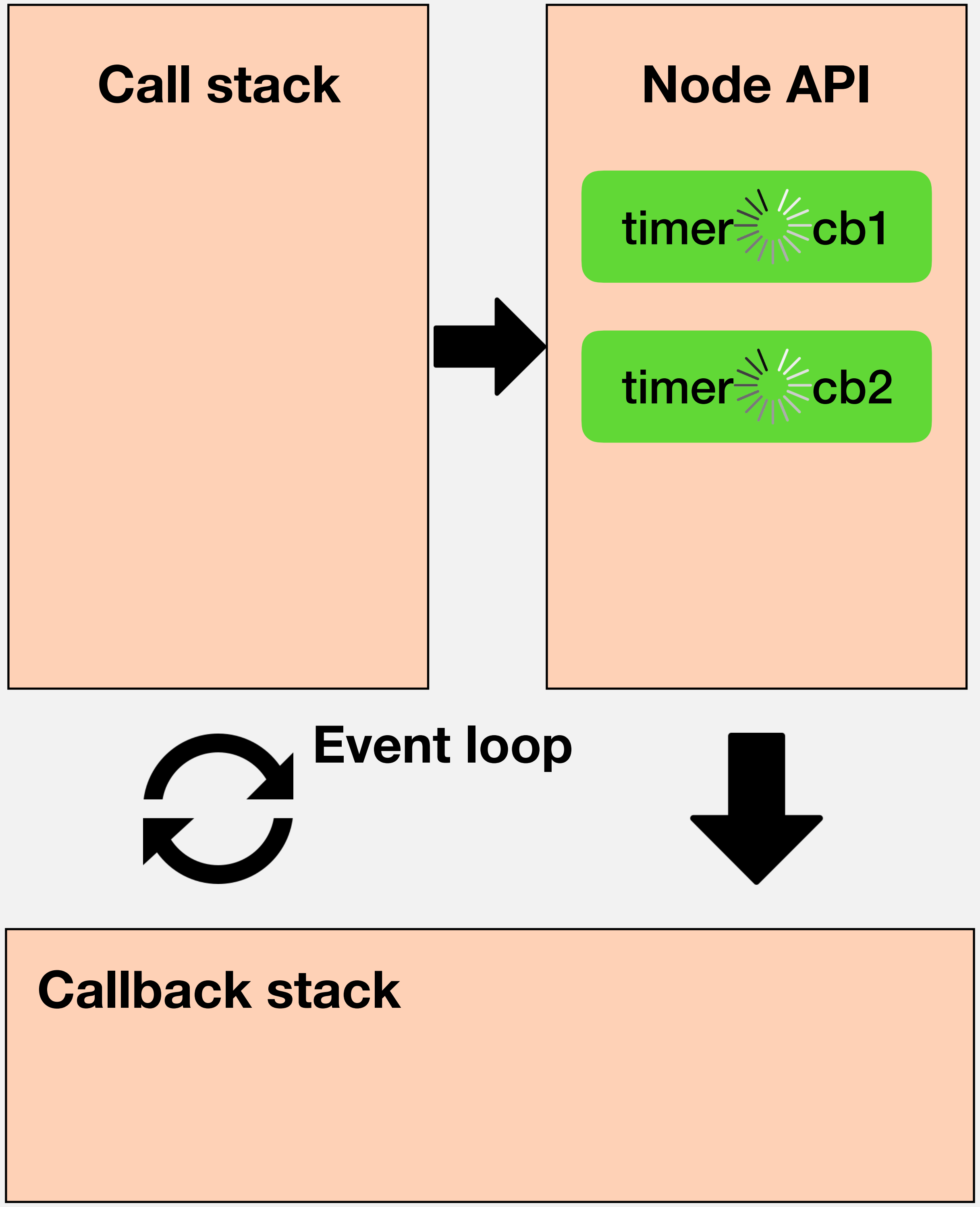
JS

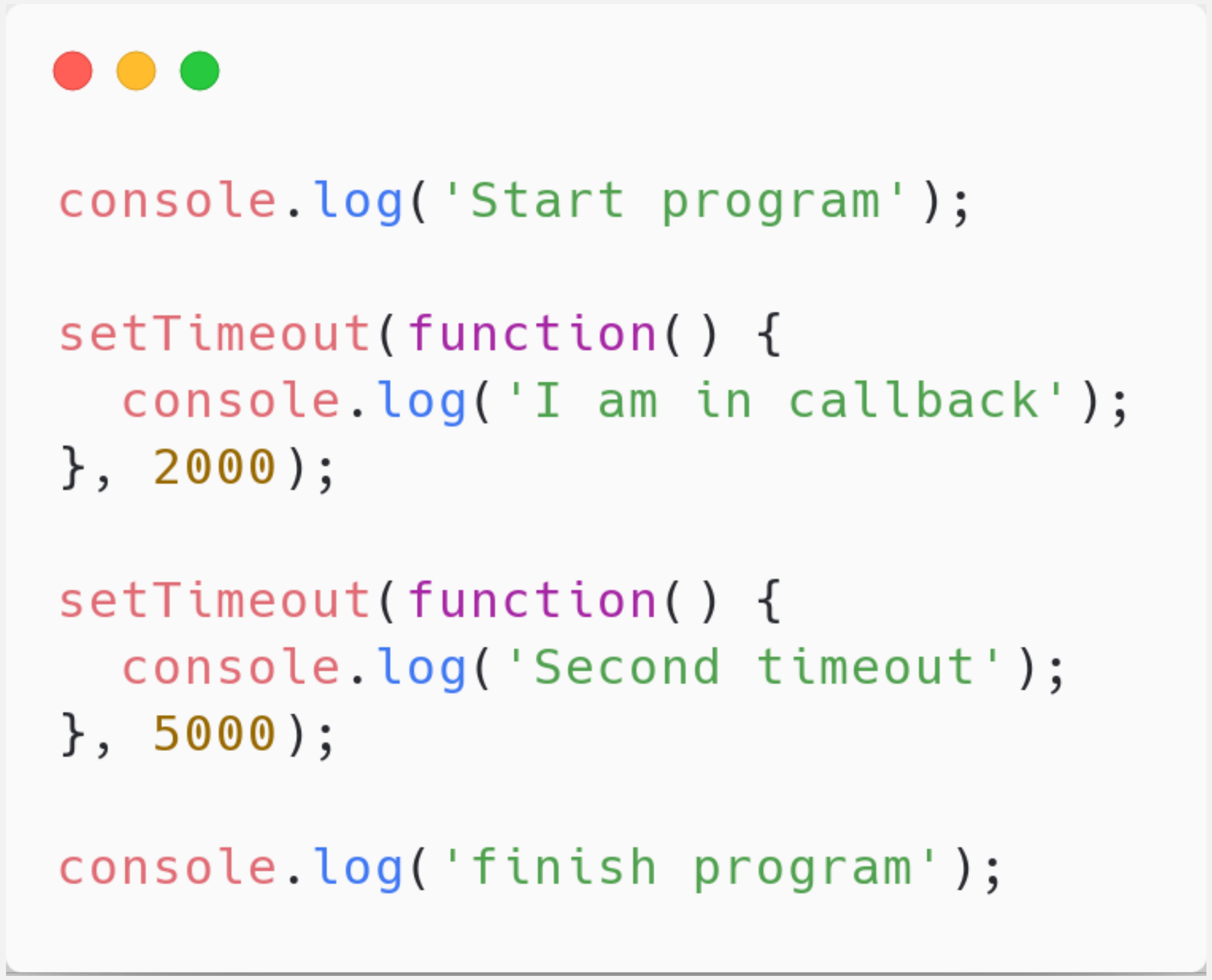


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

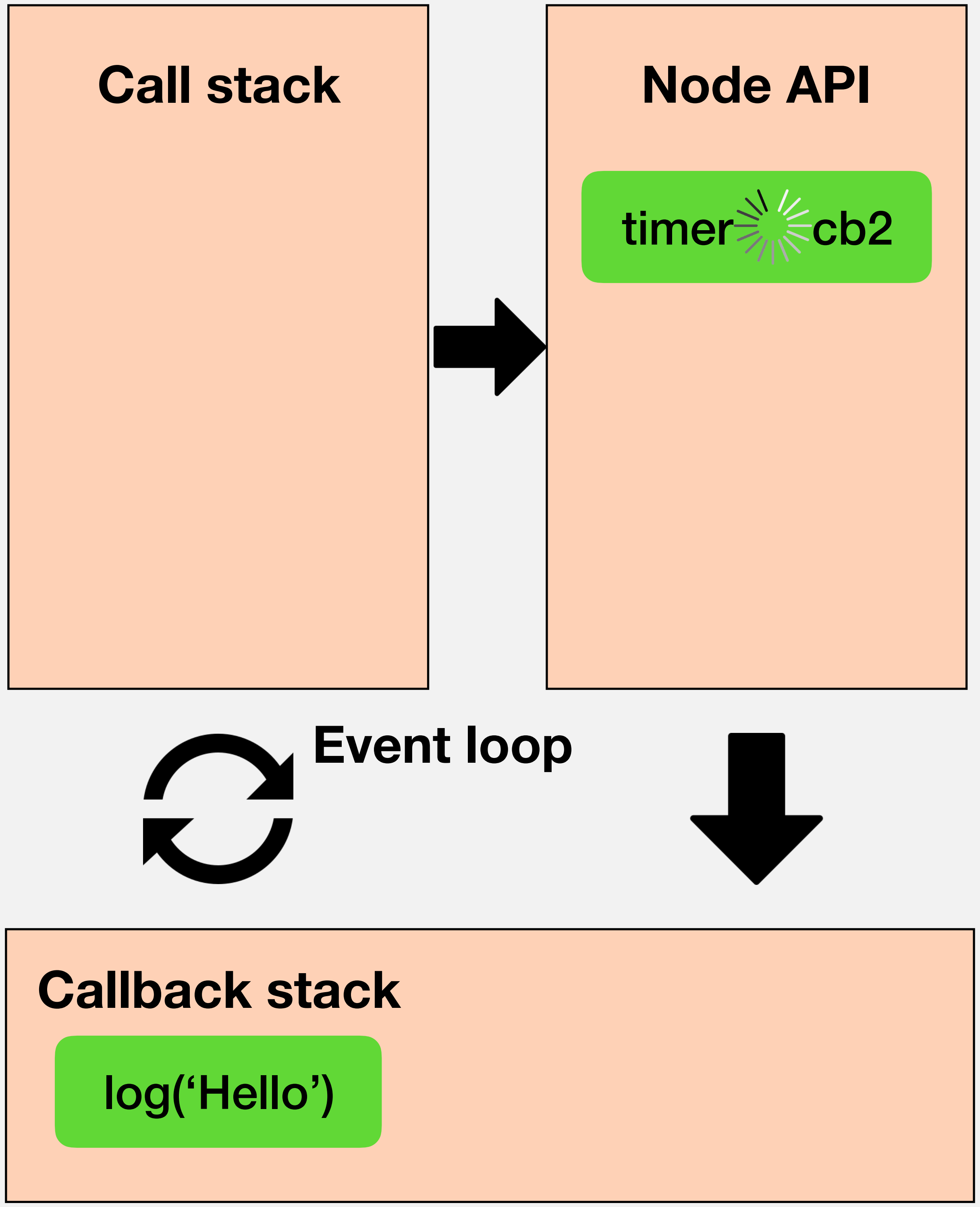



```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

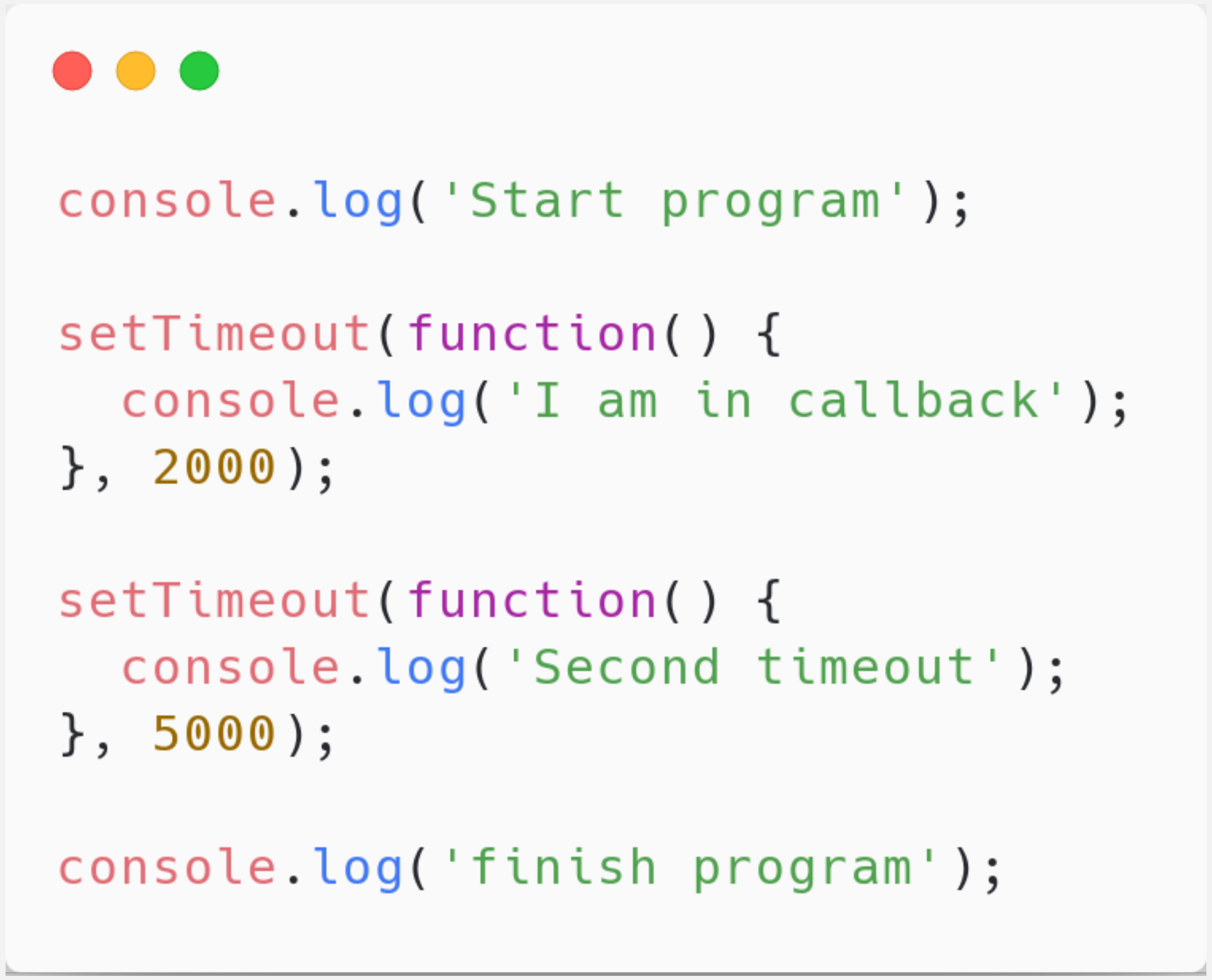




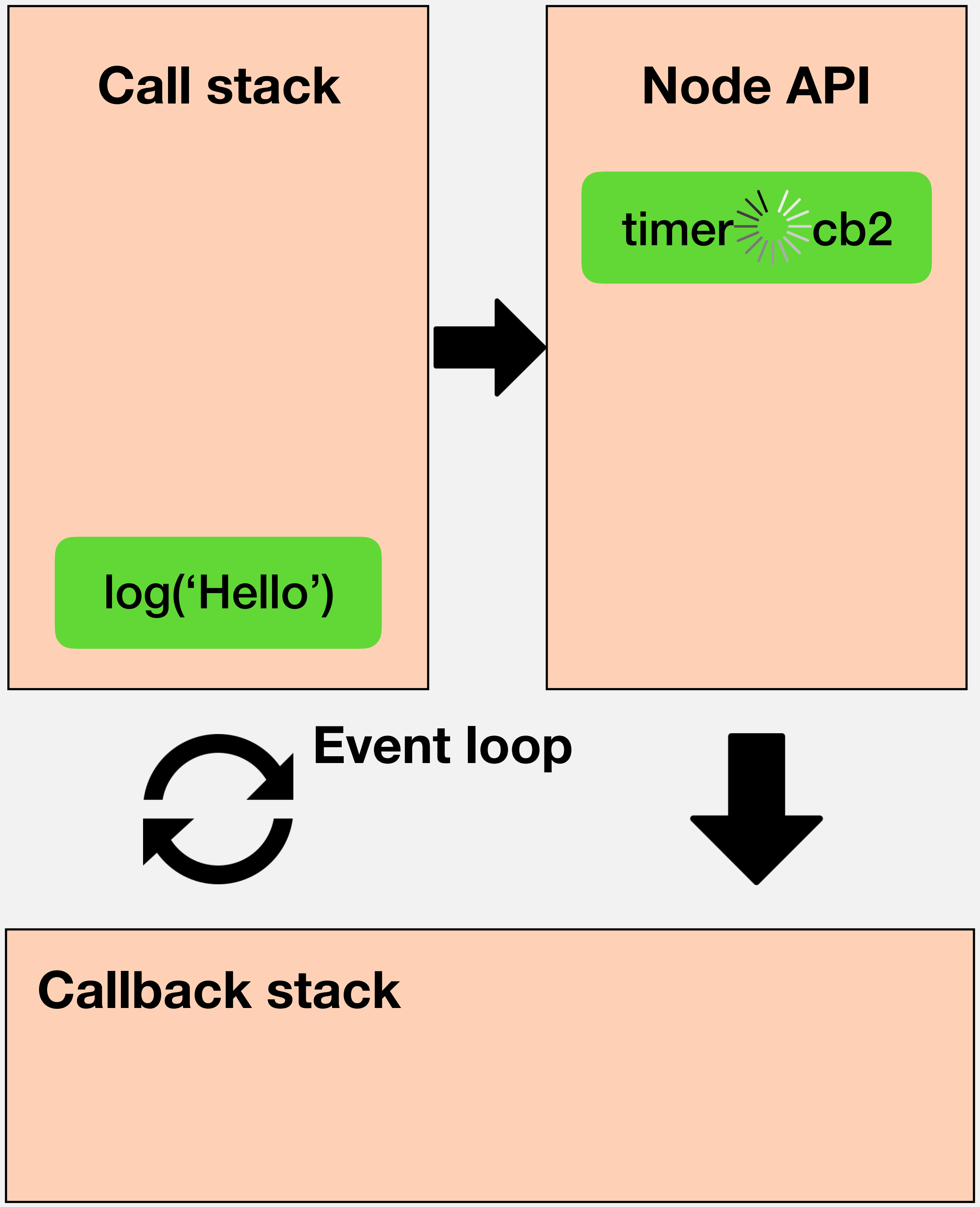
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

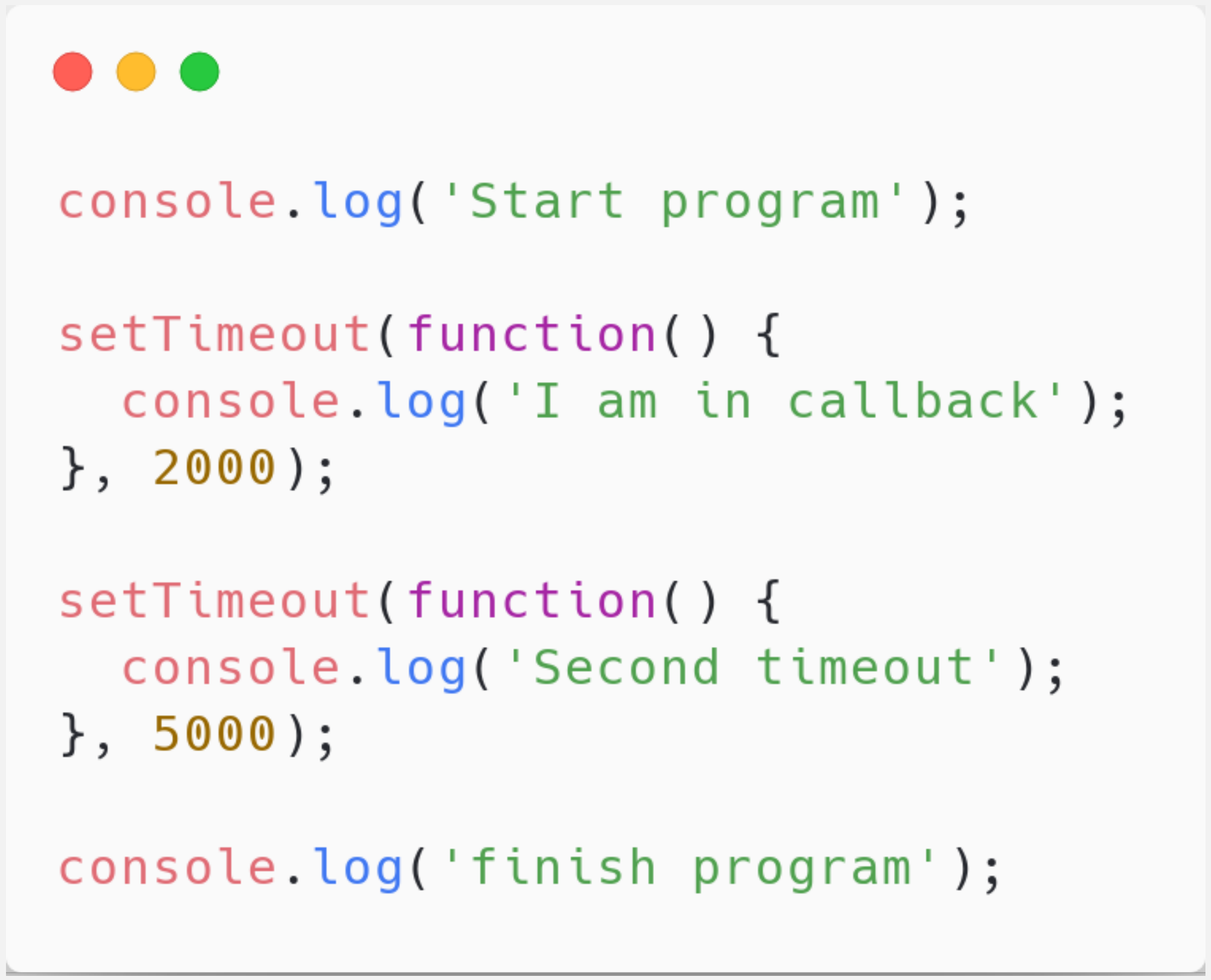


JS

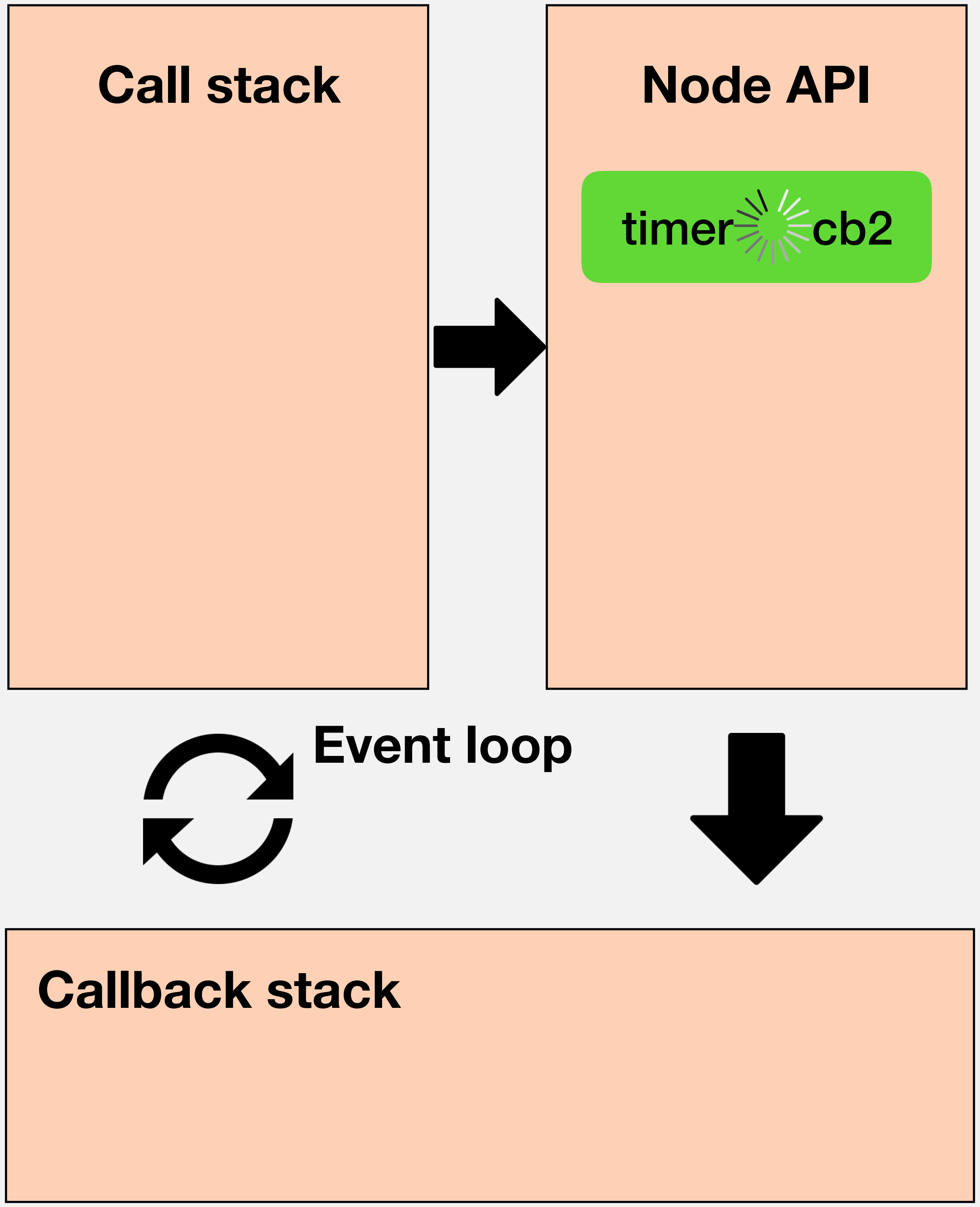


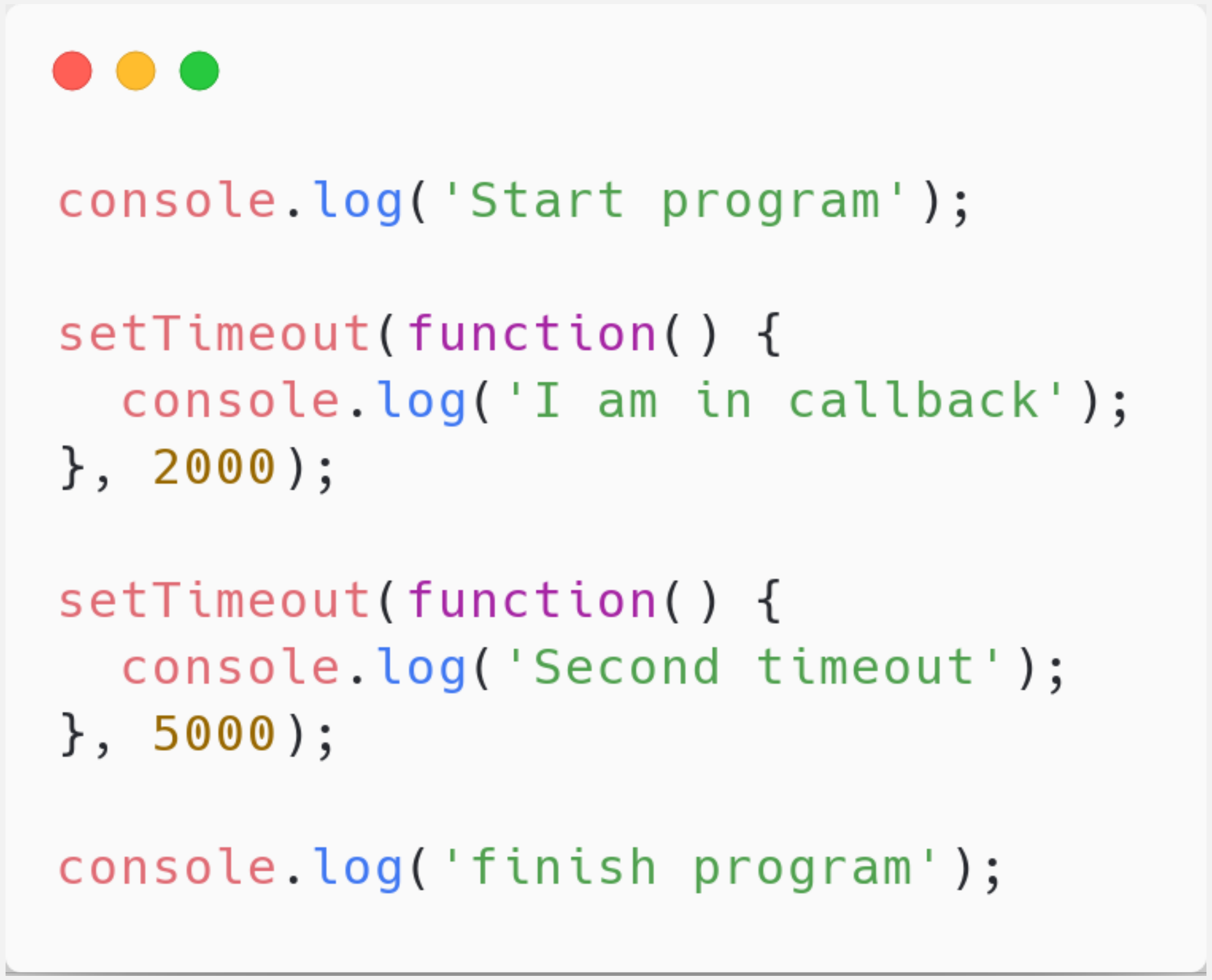
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



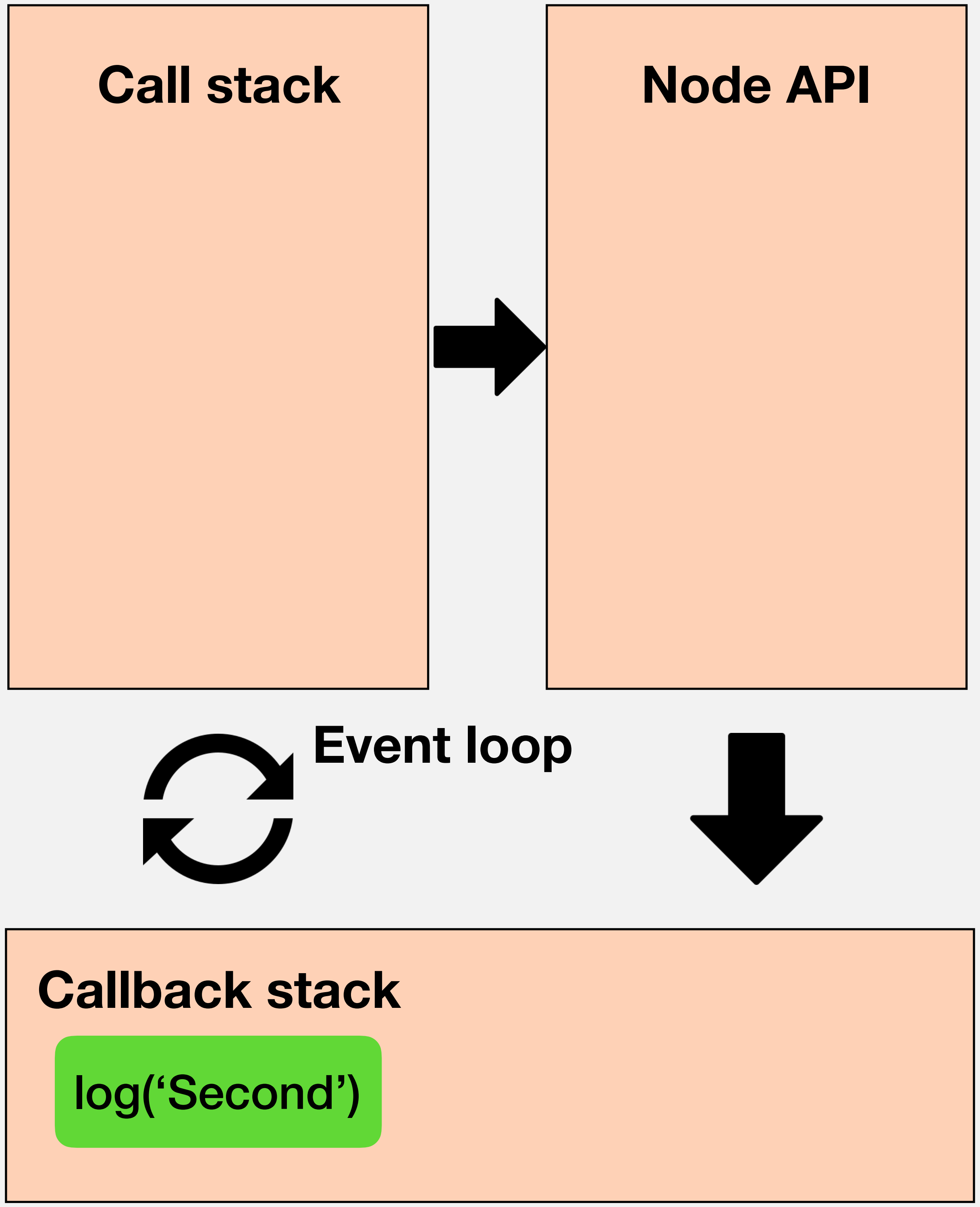


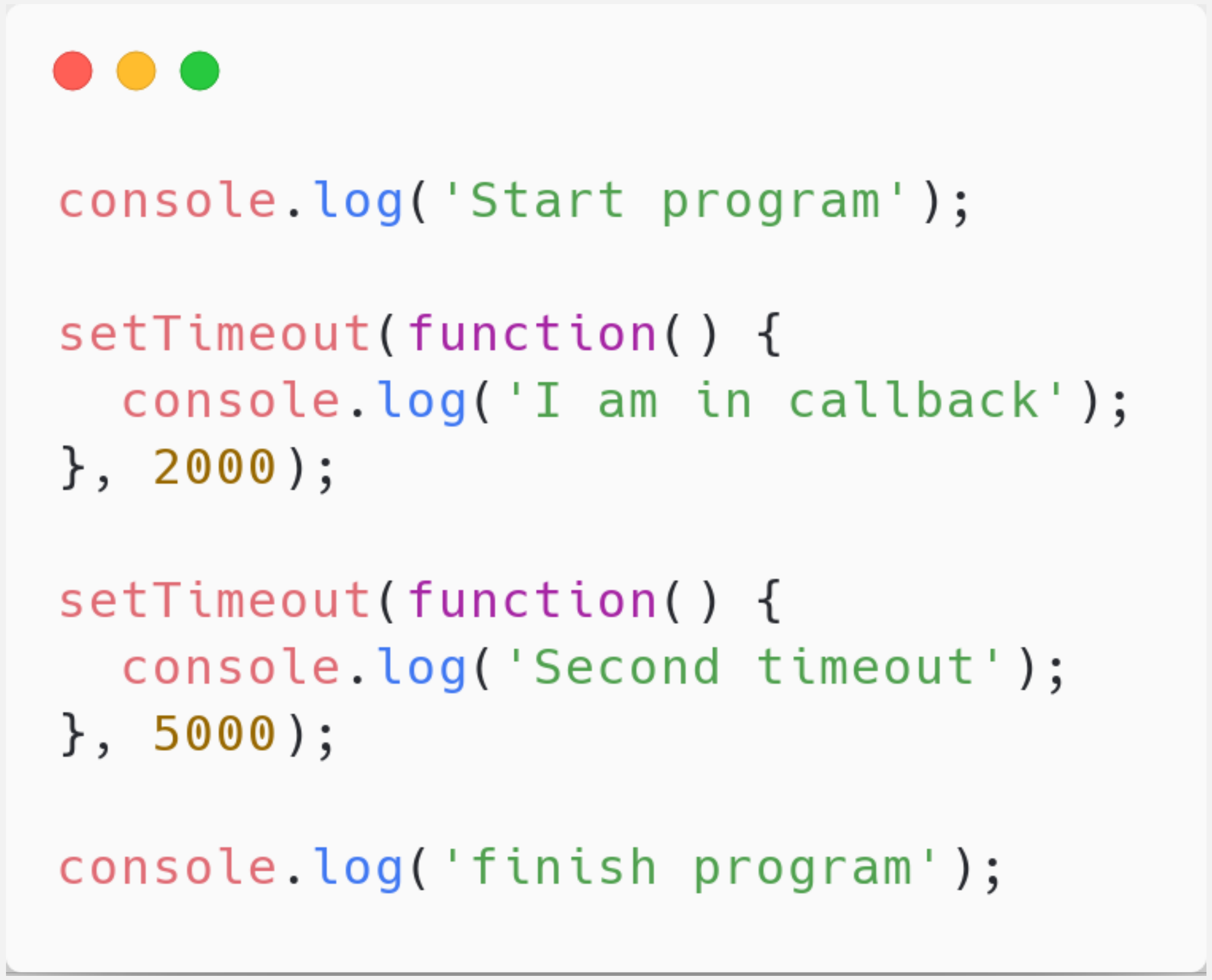
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

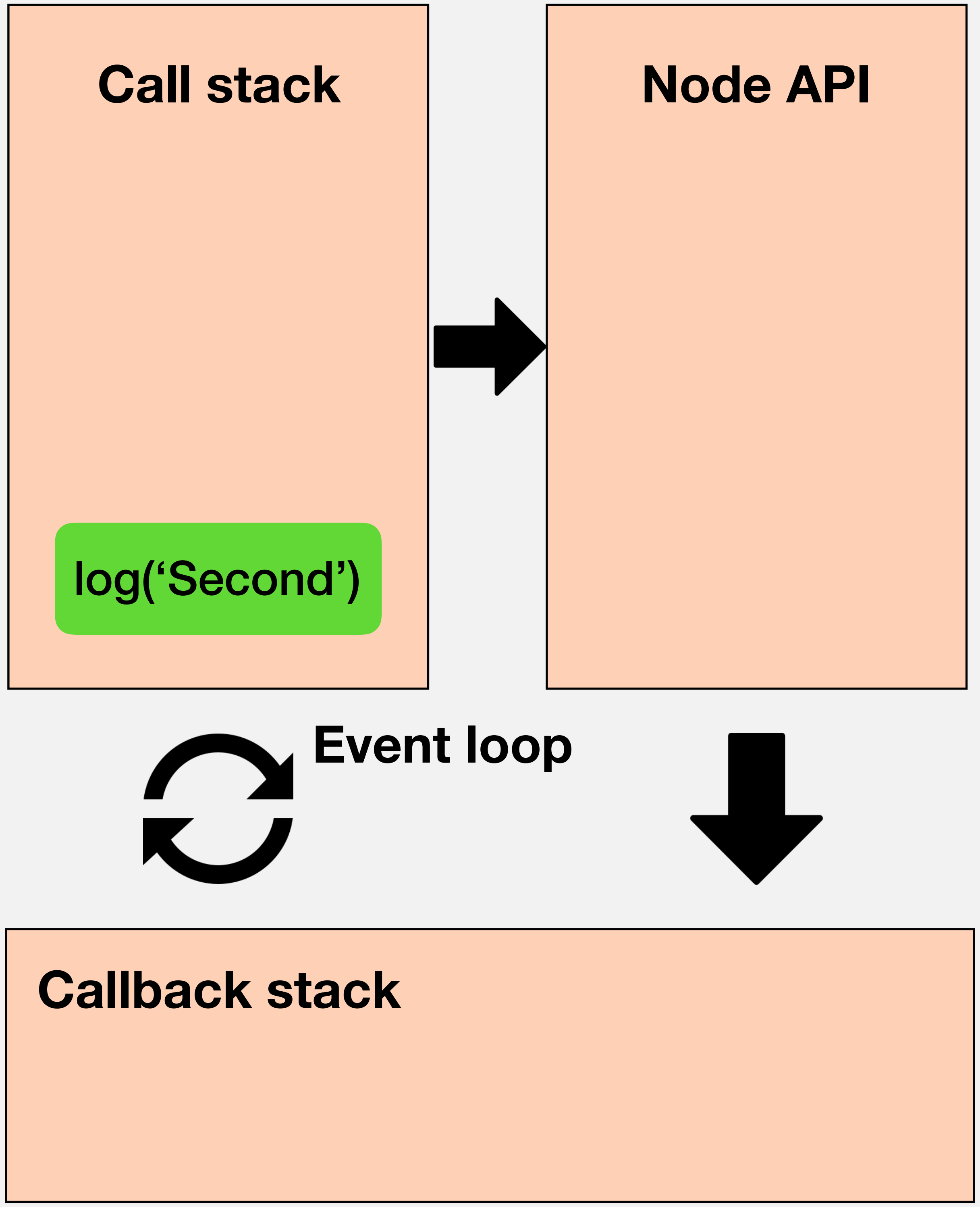


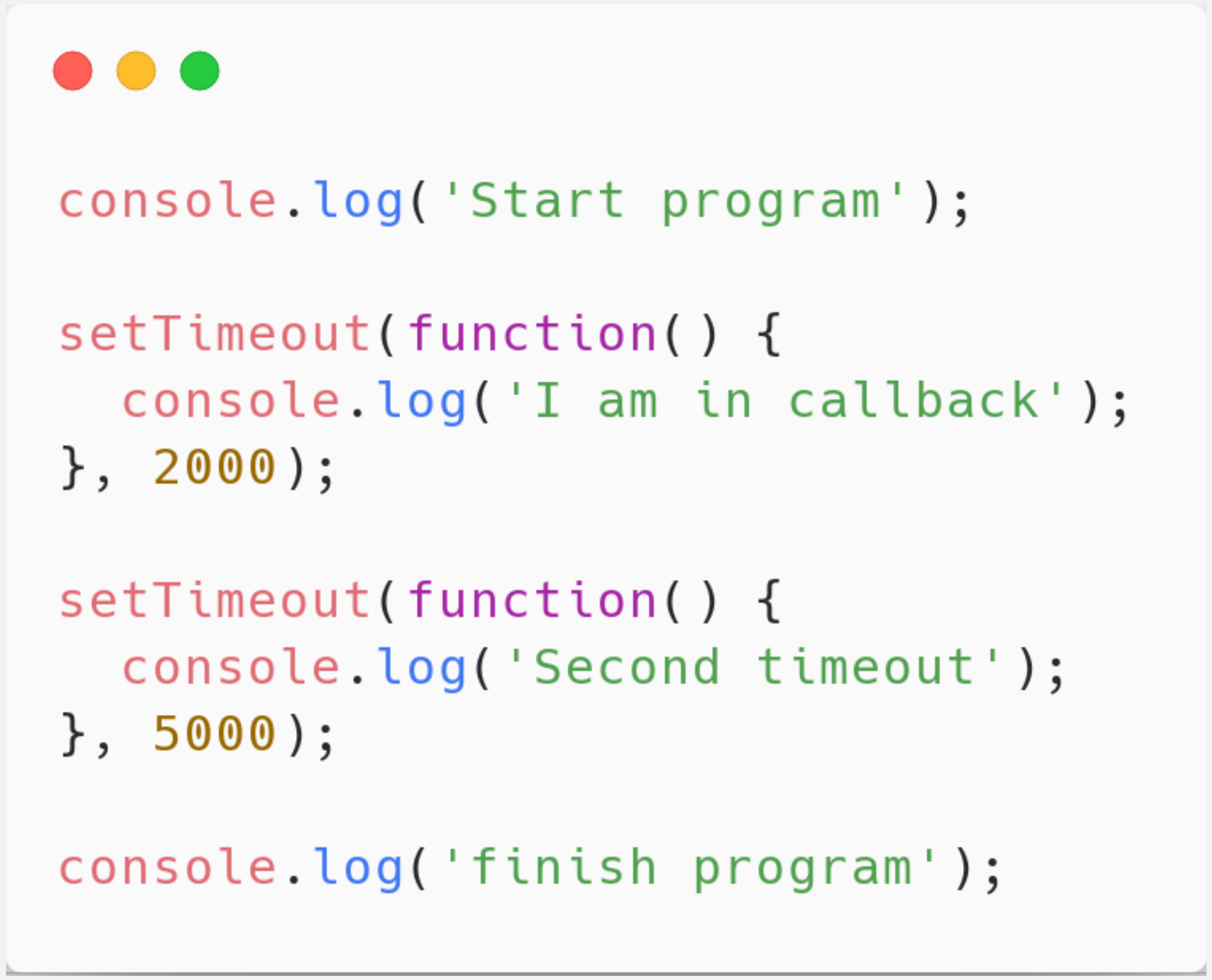


```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

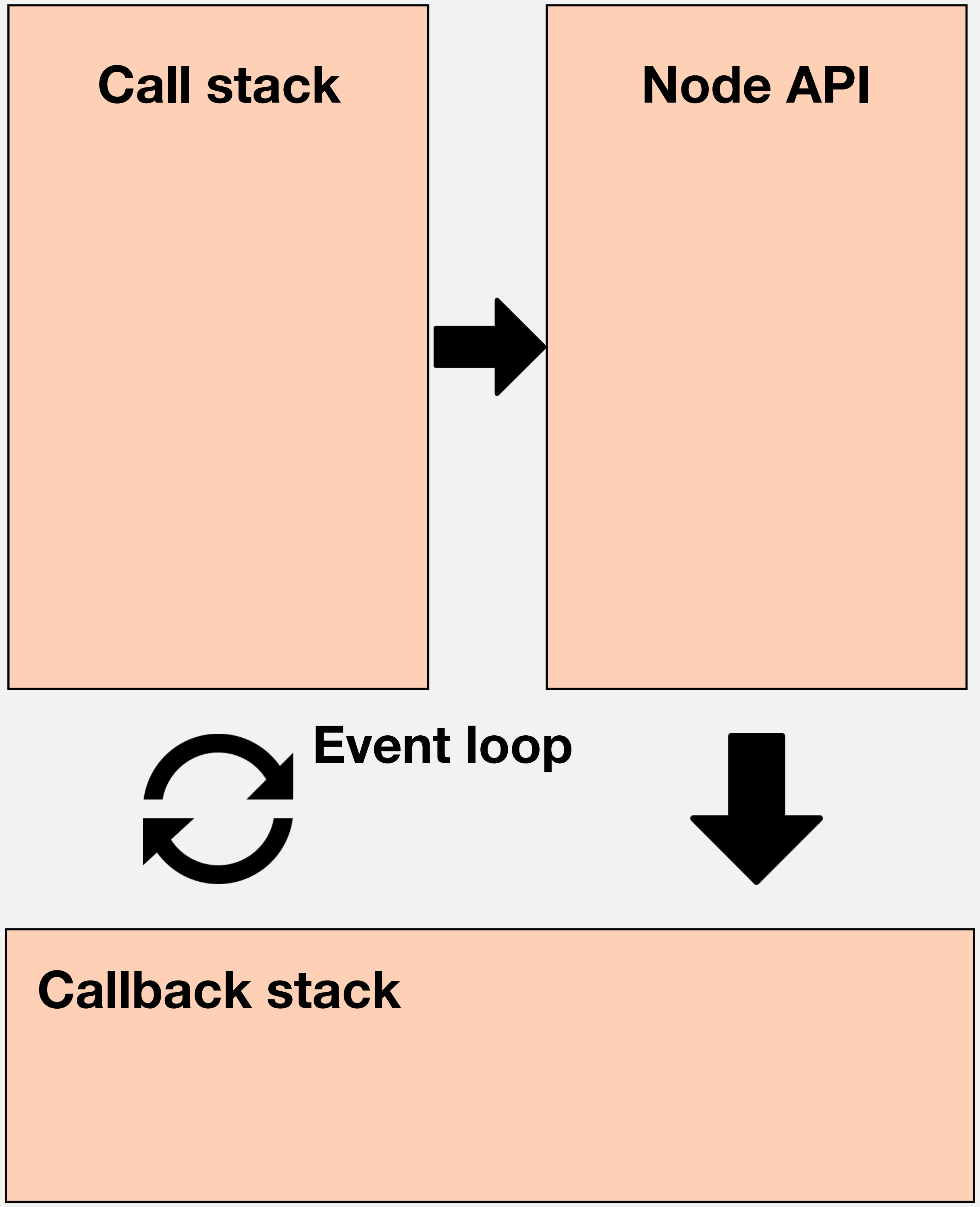



```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```

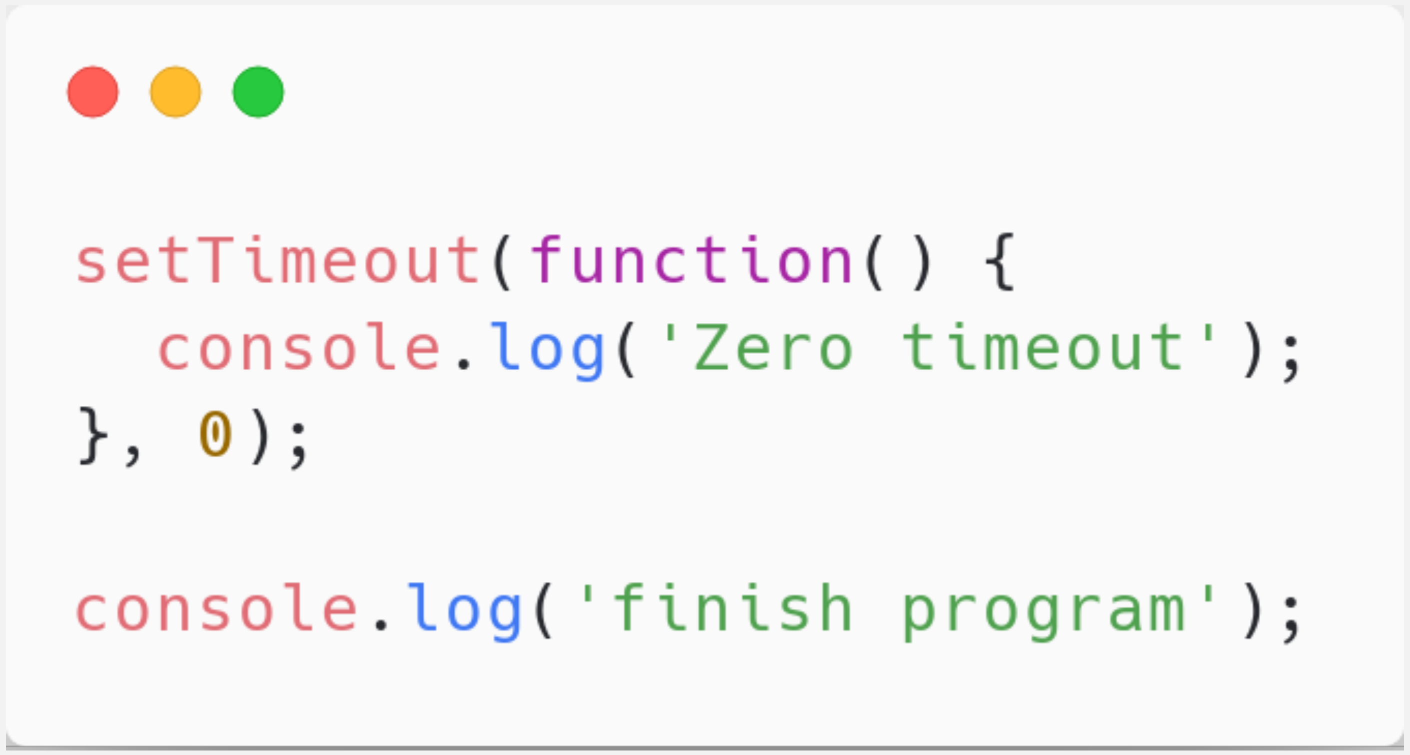




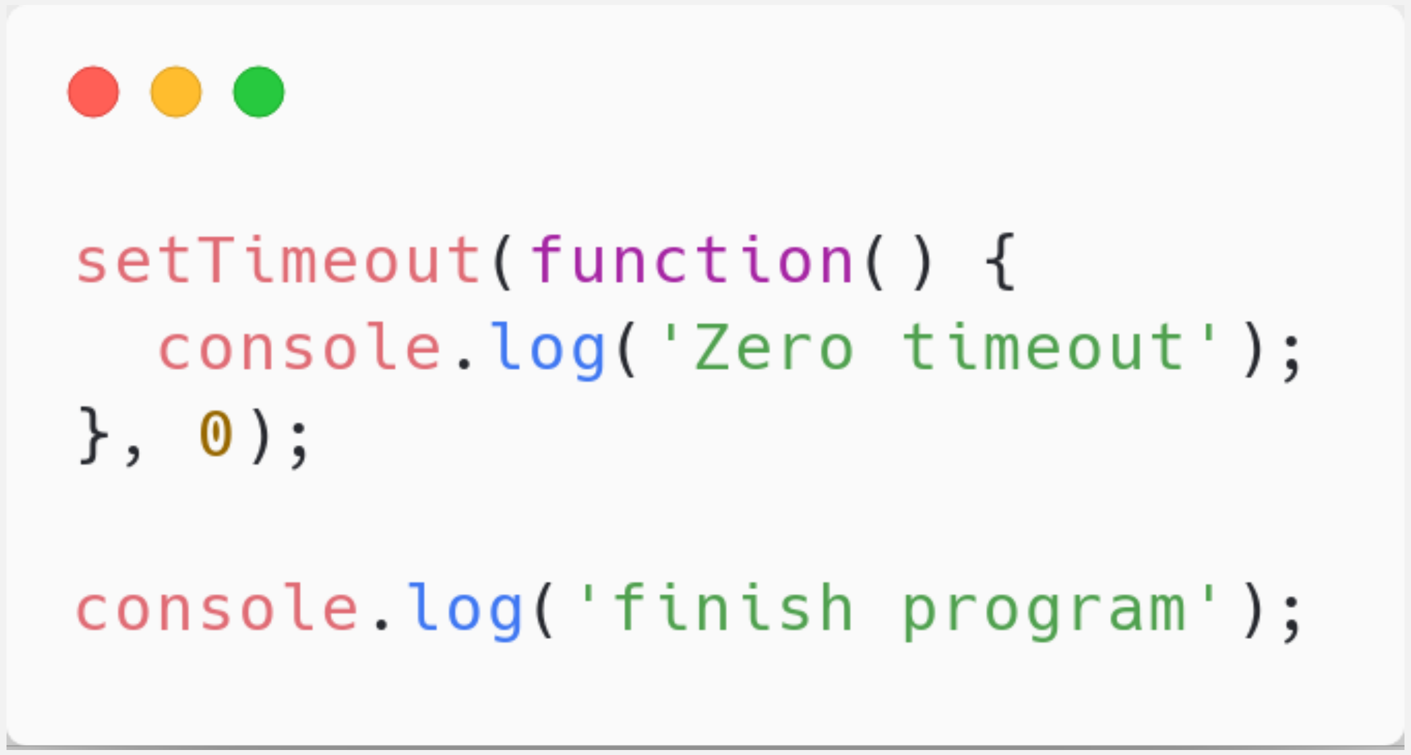
```
console.log('Start program');

setTimeout(function() {
  console.log('I am in callback');
}, 2000);

setTimeout(function() {
  console.log('Second timeout');
}, 5000);

console.log('finish program');
```



```
setTimeout(callback, 0)
```



```
setTimeout(function() {
  console.log('Zero timeout');
}, 0);

console.log('finish program');
```



```
setTimeout(function() {
  console.log('Zero timeout');
}, 0);

console.log('finish program');
```

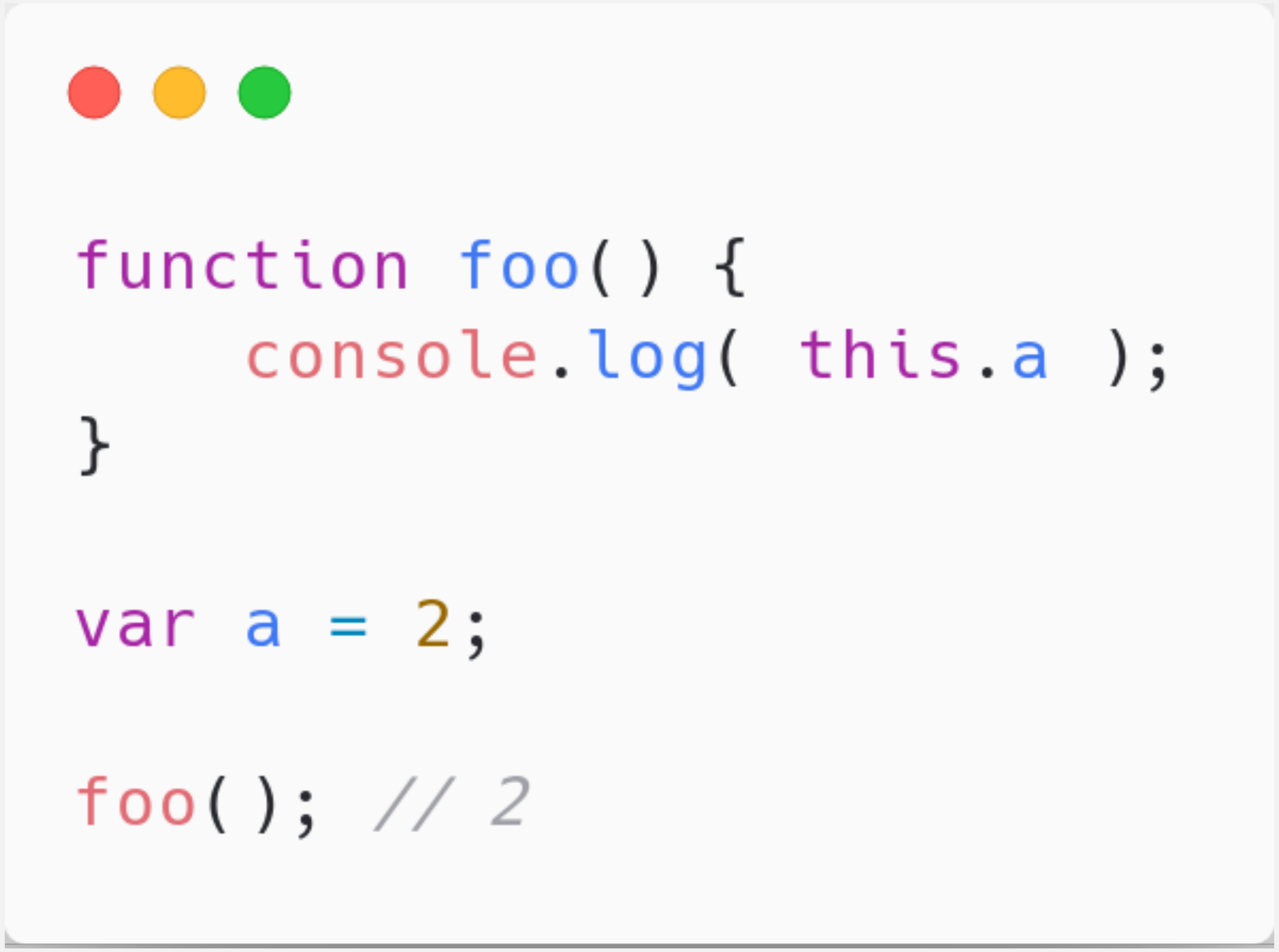
finish program
Zero timeout

I am

- ~~Single threaded~~
- ~~Non-blocking~~
- ~~Asynchronous~~
- ~~Concurrent~~

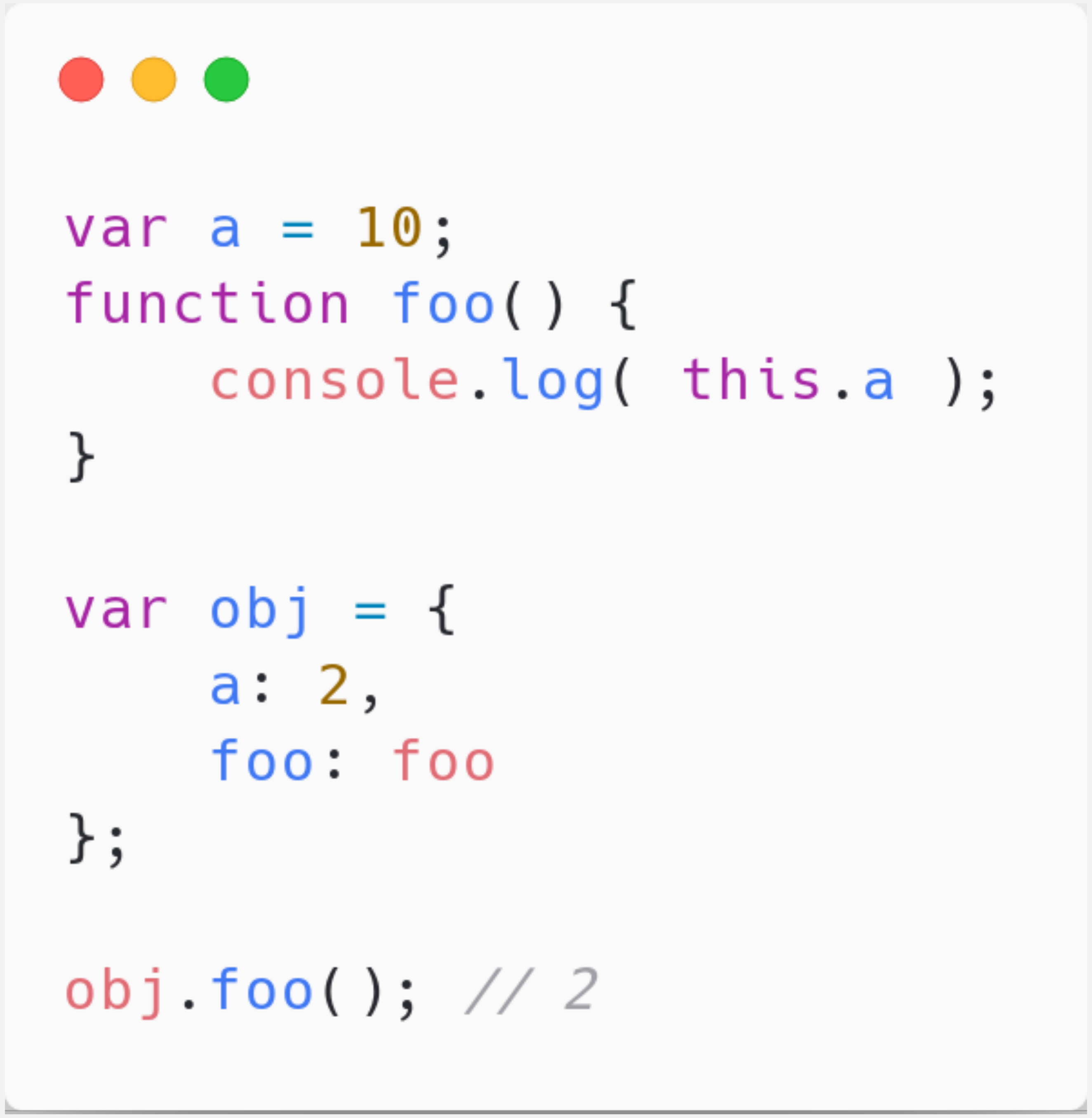
This

Привязка по умолчанию



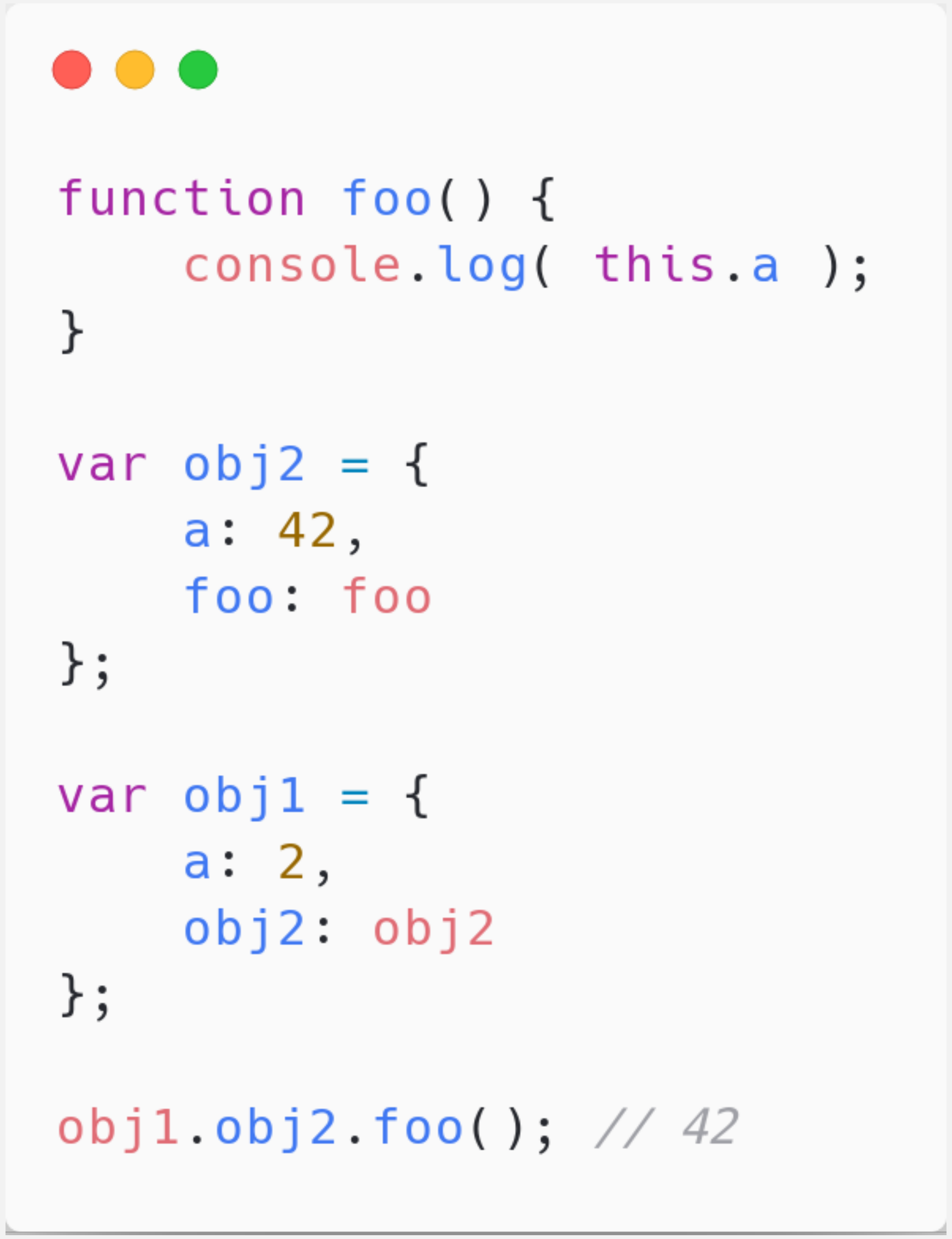
```
function foo() {
    console.log( this.a );
}

var a = 2;

foo(); // 2
```

Неявная привязка



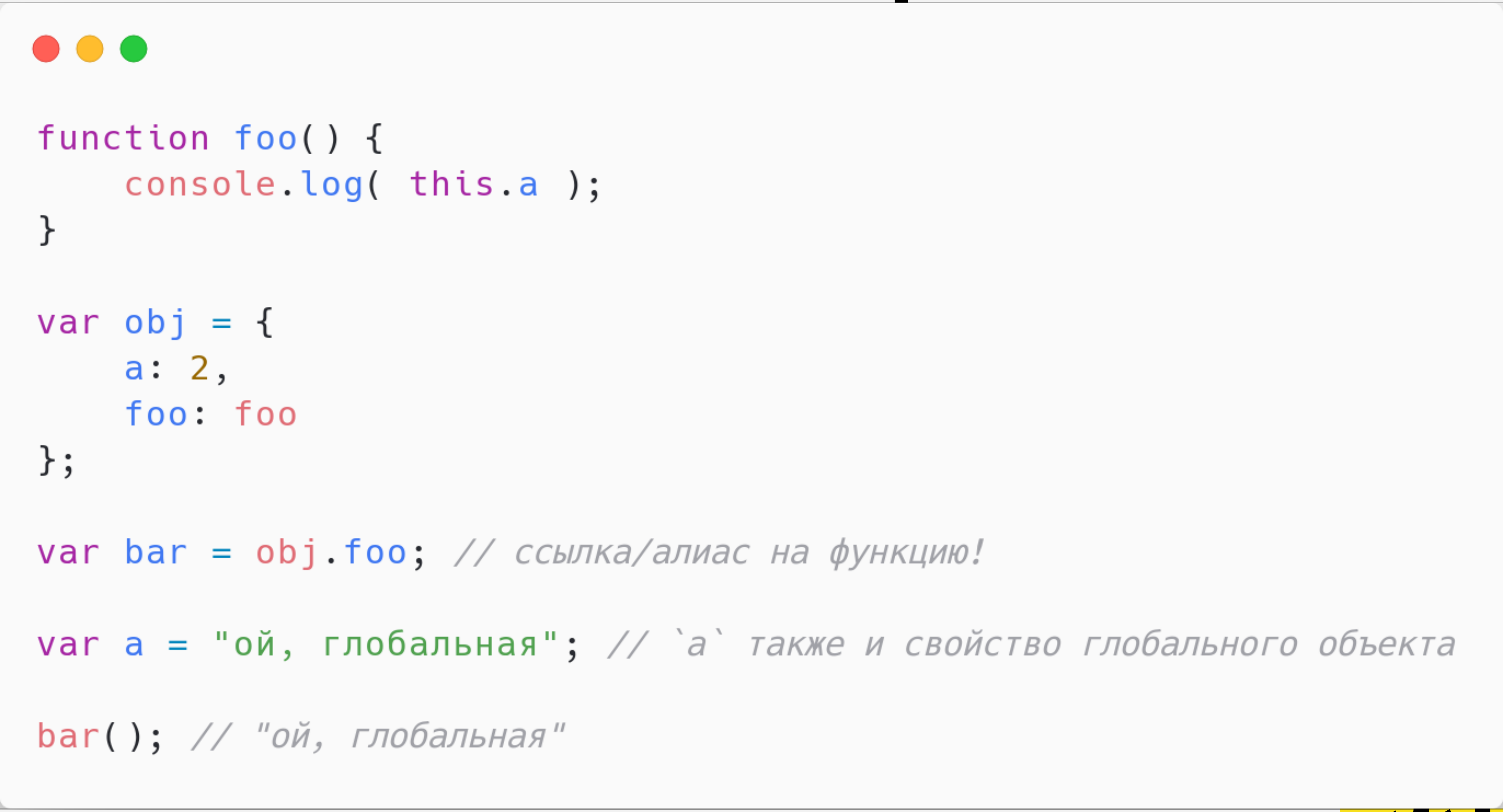
```
var a = 10;
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2,
    foo: foo
};

obj.foo(); // 2
```

Неявная привязка



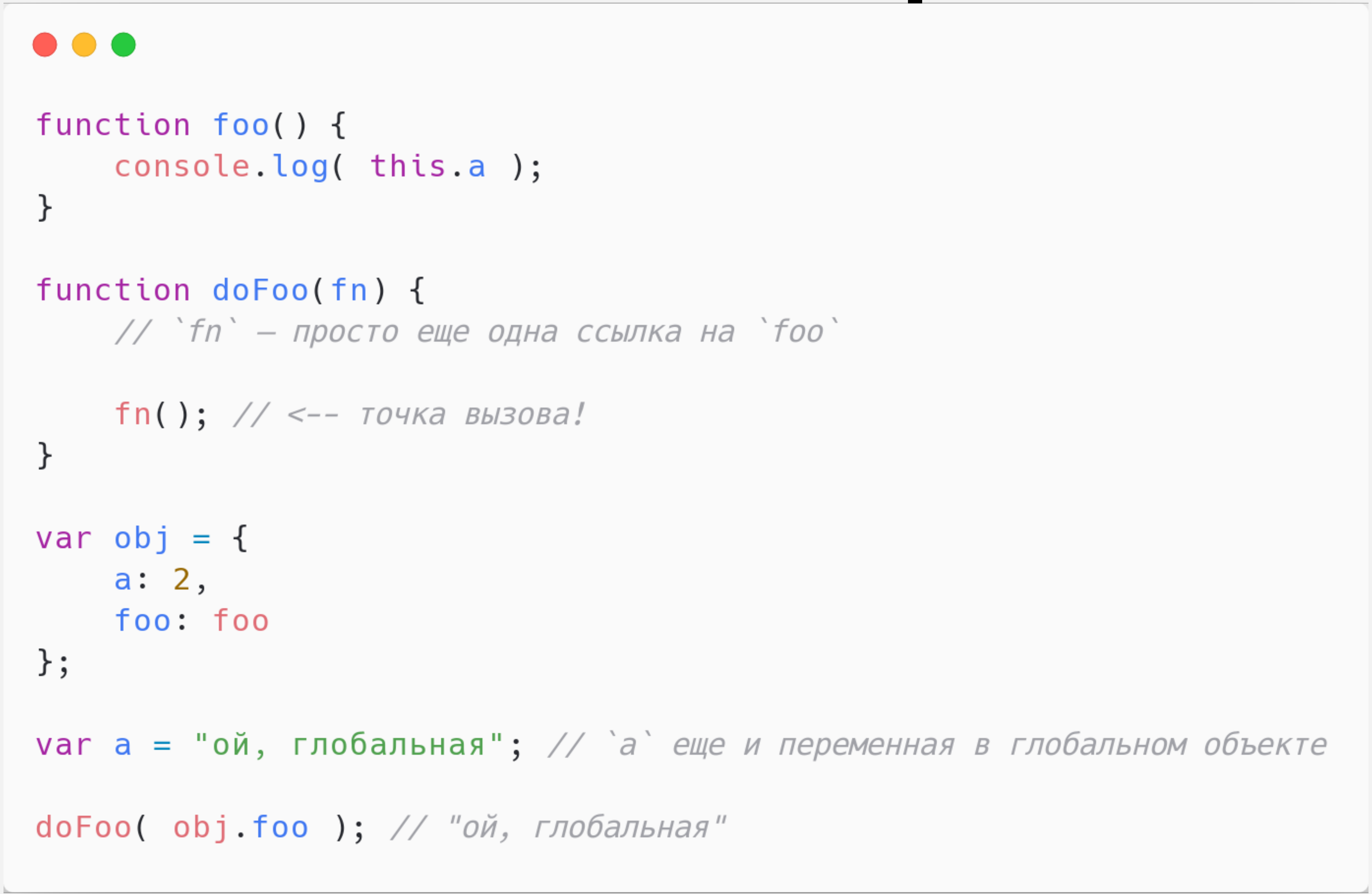
```
function foo() {
    console.log( this.a );
}

var obj2 = {
    a: 42,
    foo: foo
};

var obj1 = {
    a: 2,
    obj2: obj2
};

obj1.obj2.foo(); // 42
```

Неявно потерянный



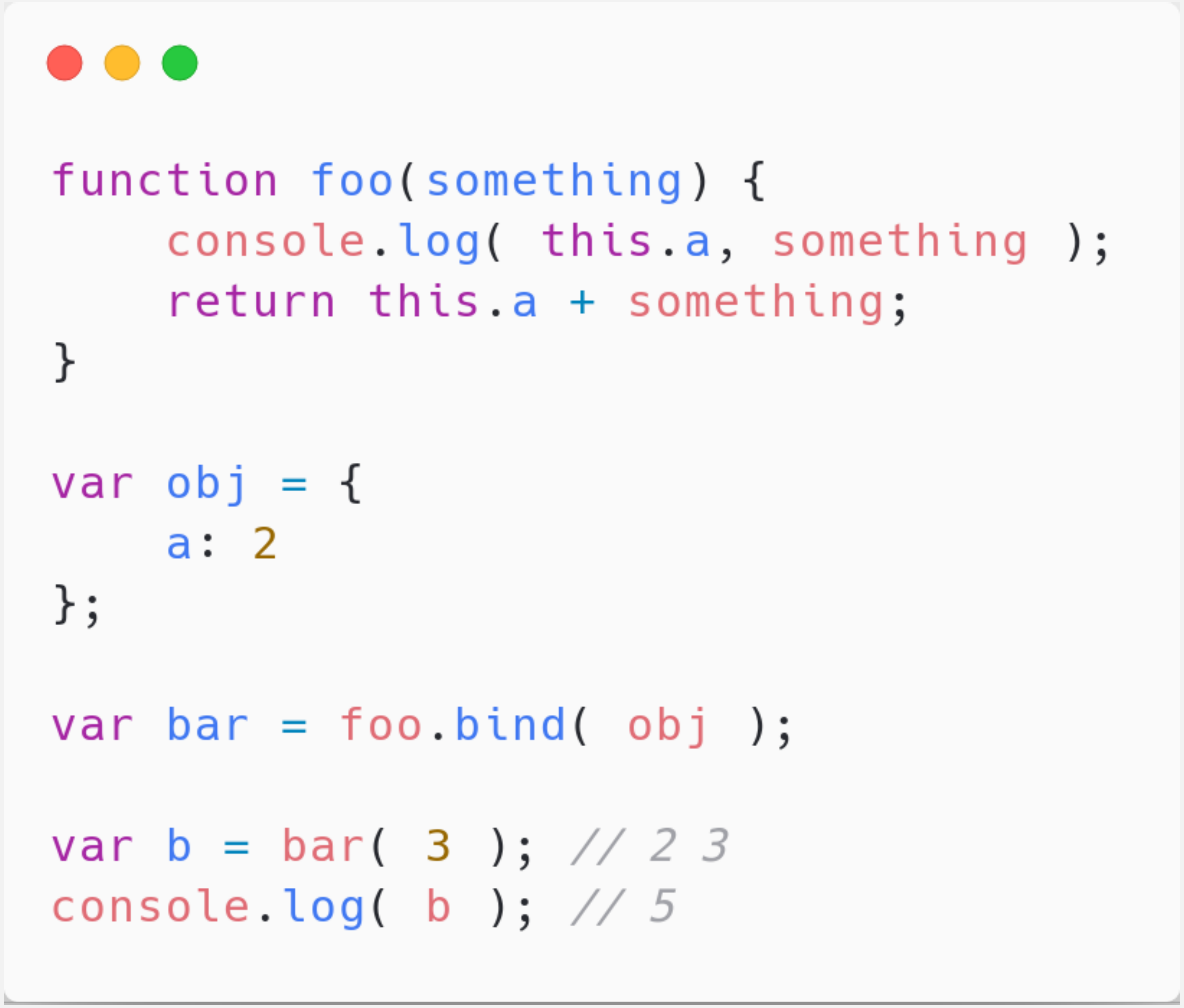
```javascript
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2,
    foo: foo
};

var bar = obj.foo; // ссылка/алиас на функцию!

var a = "ой, глобальная"; // `a` также и свойство глобального объекта

bar(); // "ой, глобальная"
```

Неявно потерянный



```javascript
function foo() {
    console.log( this.a );
}

function doFoo(fn) {
    // `fn` — просто еще одна ссылка на `foo`

    fn(); // <-- точка вызова!
}

var obj = {
    a: 2,
    foo: foo
};

var a = "ой, глобальная"; // `a` еще и переменная в глобальном объекте

doFoo( obj.foo ); // "ой, глобальная"
```

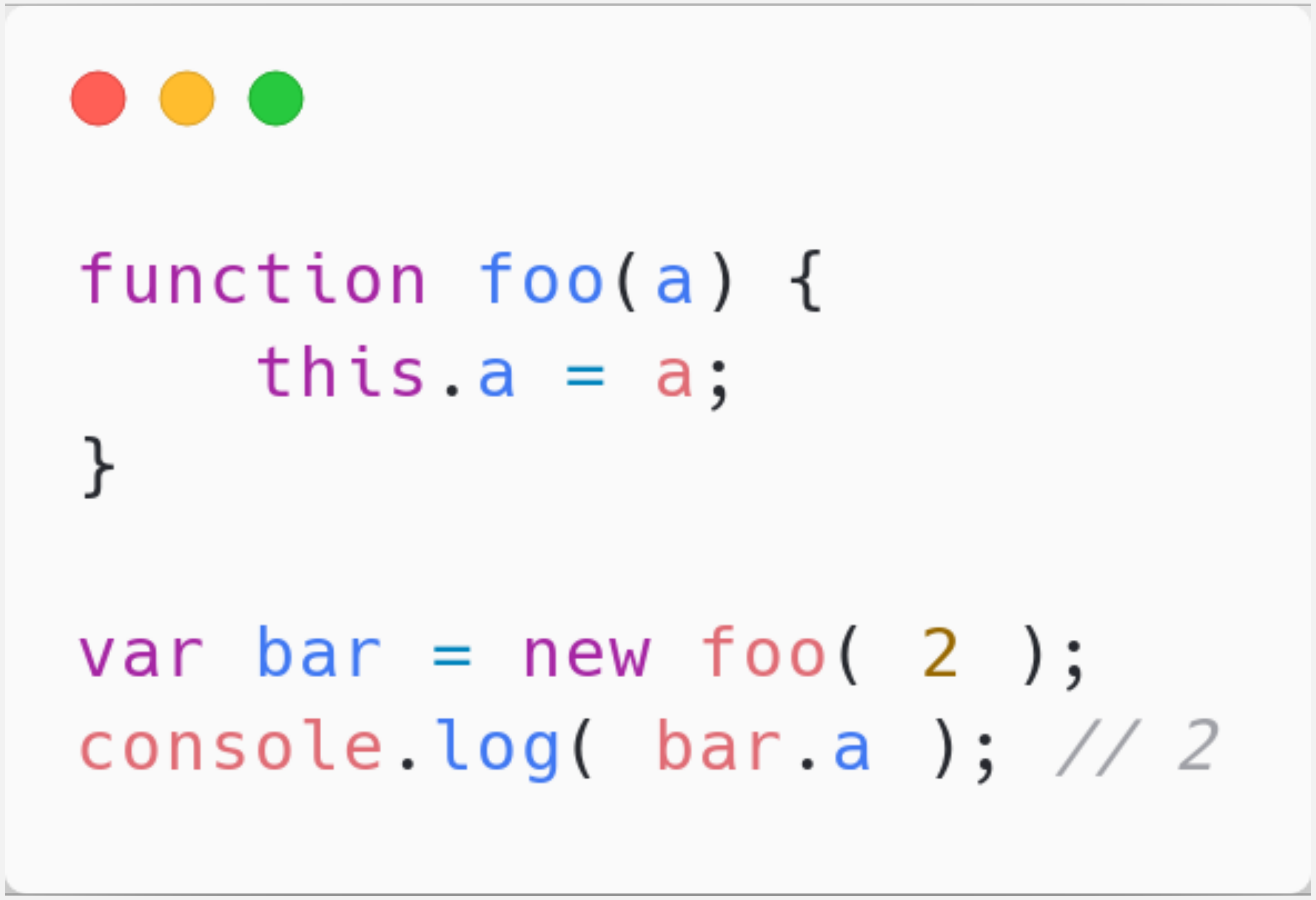
Явная привязка



```
function foo(something) {
    console.log( this.a, something );
    return this.a + something;
}

var obj = {
    a: 2
};

var bar = foo.bind( obj );

var b = bar( 3 ); // 2 3
console.log( b ); // 5
```

JS

Привязка new



```
function foo(a) {
    this.a = a;
}

var bar = new foo( 2 );
console.log( bar.a ); // 2
```

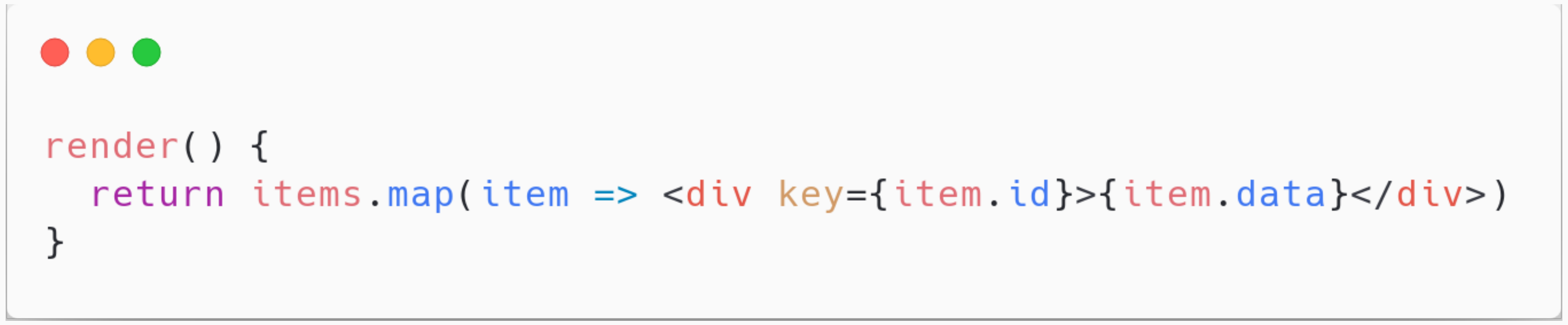
VirtualDOM

Virtual DOM

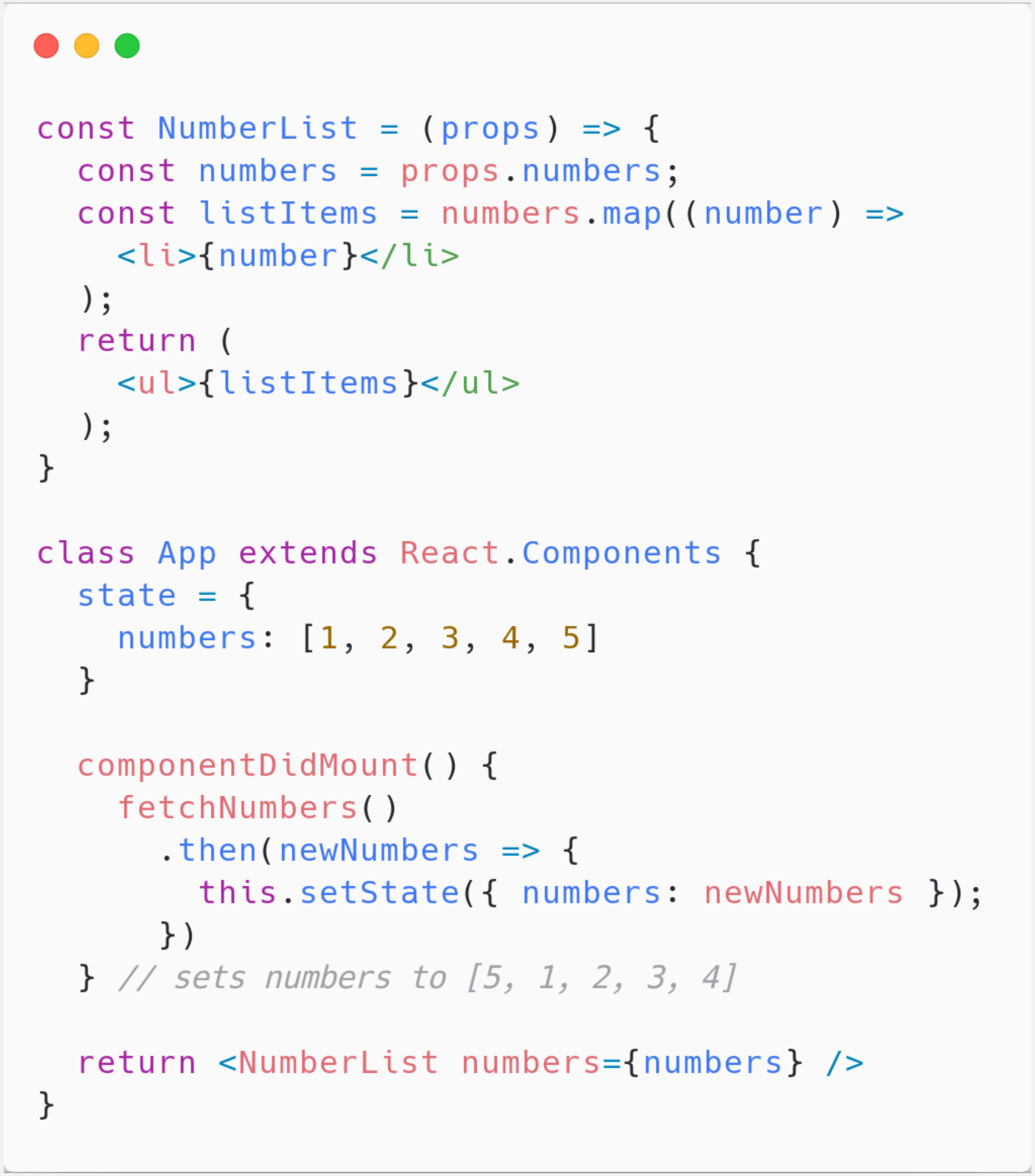
- JS object представляющий двойника browser DOM
- Очень быстрый, по сравнению с browser DOM
- Может создавать более 200.000 узлов в секунду
- Создается ПОЛНОСТЬЮ С НУЛЯ при каждом изменении состояния приложения

Трансформация одного дерева в другое занимает O(n^3). Реакт делает это за O(n) основываясь на двух предположениях.

- Два элемента с разными типами произведут разные поддеревья
- Разработчик может указать, какие элементы остаются стабильными между рендерами с помощью key

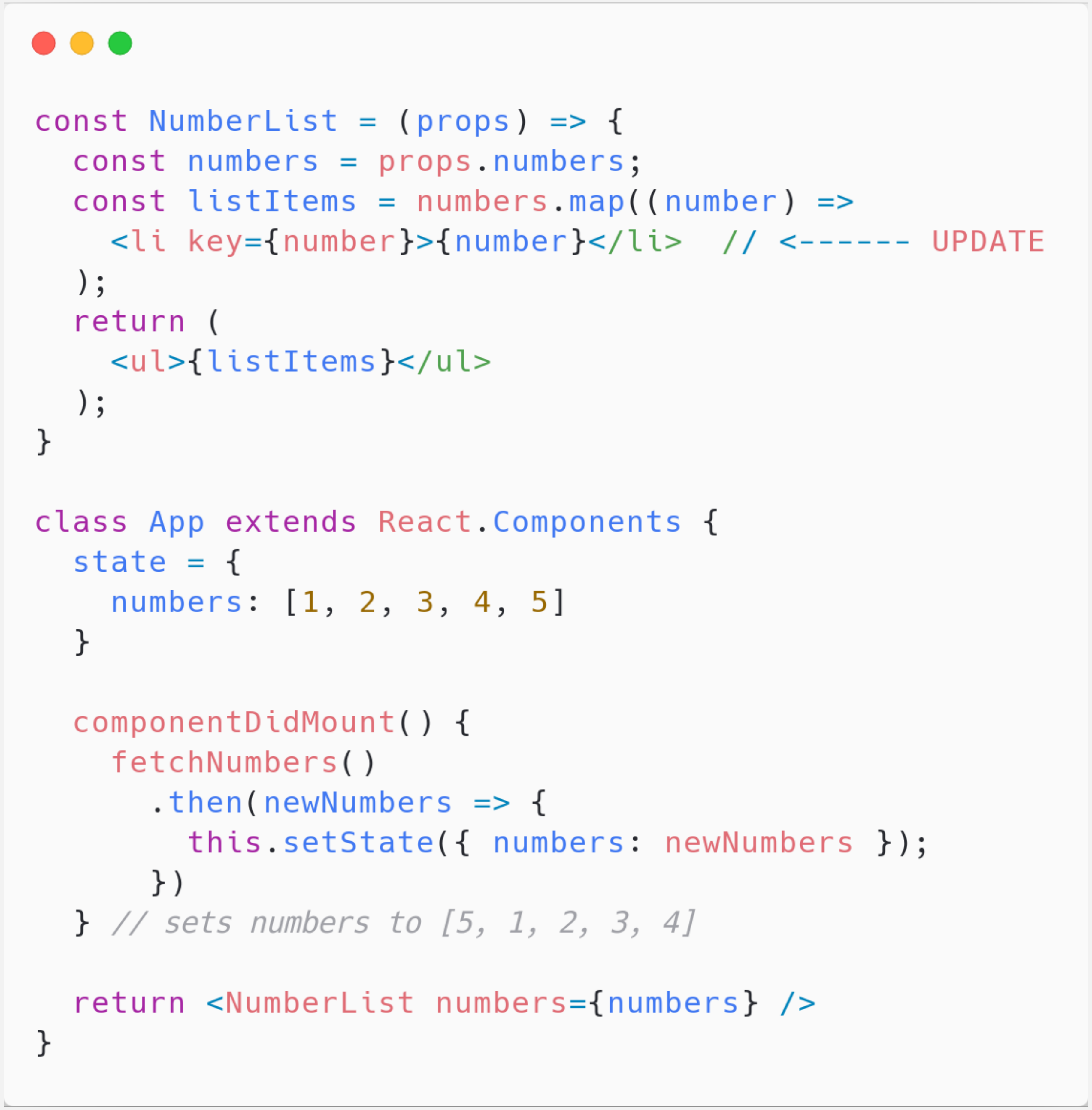


```
render() {
  return items.map(item => <div key={item.id}>{item.data}</div>)
}
```



```
const NumberList = (props) => {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

class App extends React.Components {
  state = {
    numbers: [1, 2, 3, 4, 5]
  }

  componentDidMount() {
    fetchNumbers()
      .then(newNumbers => {
        this.setState({ numbers: newNumbers });
      })
  } // sets numbers to [5, 1, 2, 3, 4]

  return <NumberList numbers={numbers} />
}
```

Every <li>{number}</li> will be destroyed and created new one



```
const NumberList = (props) => {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number}>{number}</li>  // <------- UPDATE
  );
  return (
    <ul>{listItems}</ul>
  );
}

class App extends React.Components {
  state = {
    numbers: [1, 2, 3, 4, 5]
  }

  componentDidMount() {
    fetchNumbers()
      .then(newNumbers => {
        this.setState({ numbers: newNumbers });
      })
  } // sets numbers to [5, 1, 2, 3, 4]

  return <NumberList numbers={numbers} />
}
```

Every <li>{number}</li> will be reused