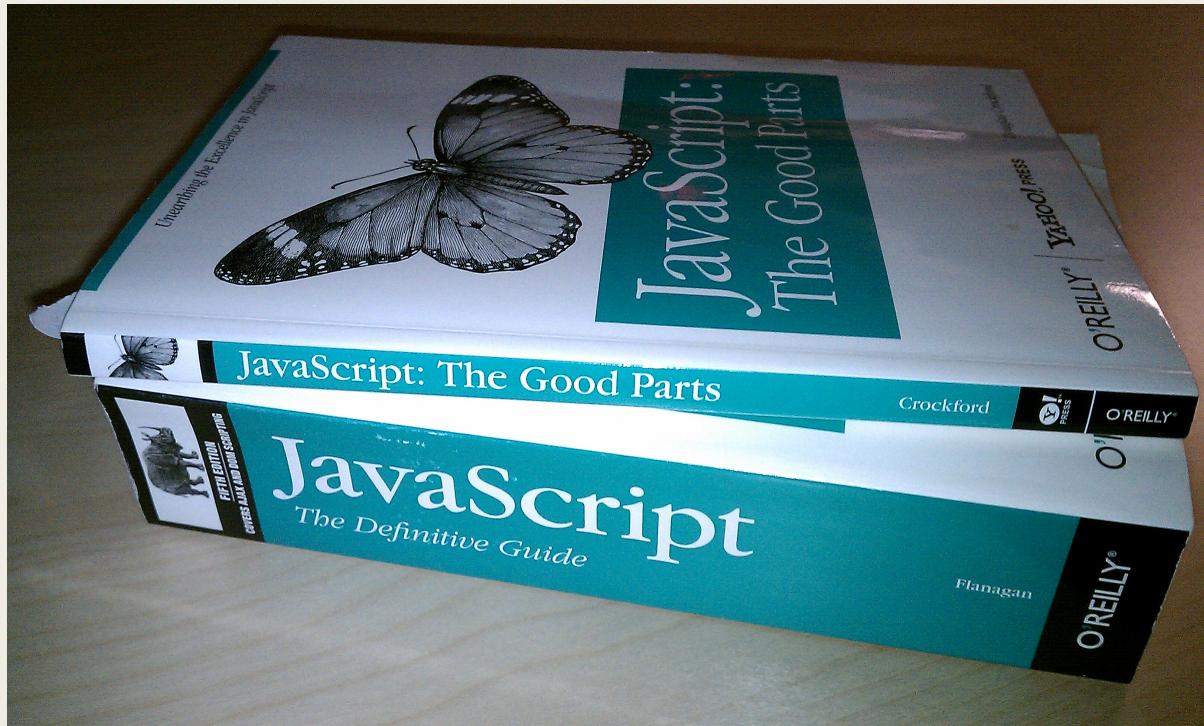


FALLING IN LOVE WITH ASYNC-AWAIT

Atishay Jain, Senior Computer Scientist, Adobe

Byetconf JS 2019

JavaScript



[Image](#) by Nathan on Flickr

Agenda

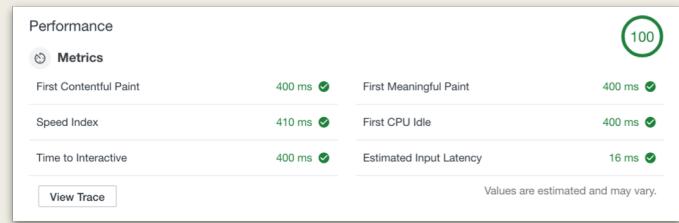
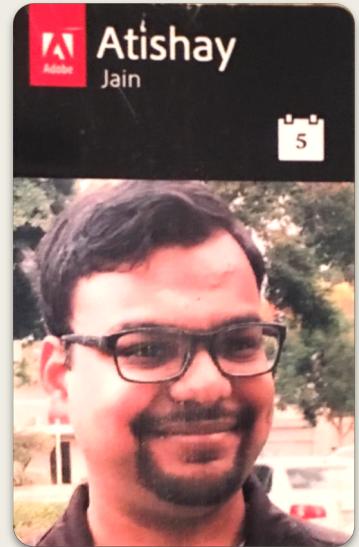
- About me
- The async-await paradigm
- A beginners walk-through of async await
 - *Callback vs promise vs async await*
 - *The program flow*
- Intermediate async-await
 - *A detour to promises*
 - *Async guarantees*
 - *The program flow*
 - *Wrapping asynchronous functions*
- Advanced async-await
 - *Exploiting the promises underneath*
 - *Exploiting callbacks deep down*
- Conclusion



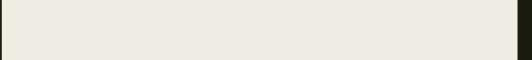
Image from [gvfcat](#)

About Me

- Work in San Francisco.
- Crazy about the web
 - *Working on web since the Flash & IE6 days.*
 - *Have worked on Desktop (Adobe InDesign), and on Mobile (Adobe Capture)*
 - *Author of VS Code's "All Autocomplete" plugin*
 - *Rewrote my website for the nth time last year) - <https://atishay.me> (Perf score 100)*
 - *Guest author at CSS-Tricks*
- Ship node based desktop software to millions of Creative Cloud users
- Use these techniques(& much more) in daily life
- Doing asynchronous node.js since Node 0.6.



THE ASYNC-AWAIT PARADIGM

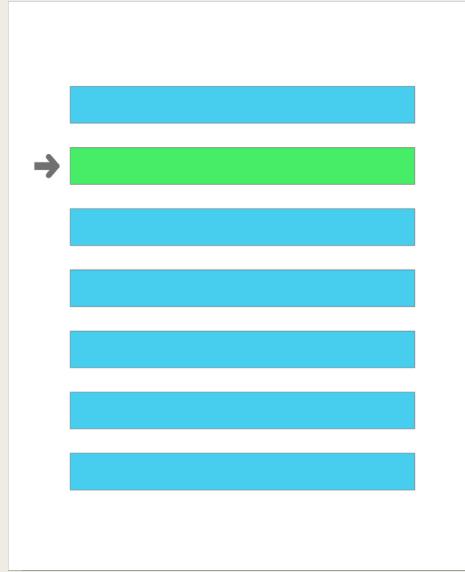


The concept of async-await



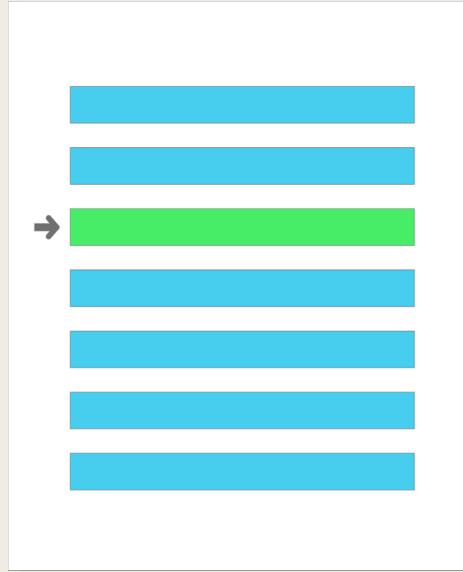
Serial Dispatch

The concept of async-await



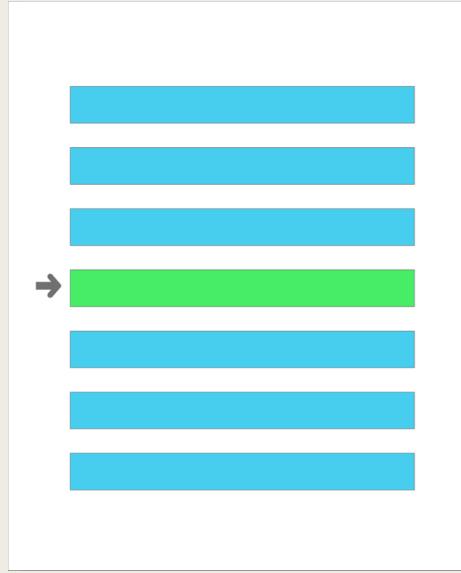
Serial Dispatch

The concept of async-await



Serial Dispatch

The concept of async-await



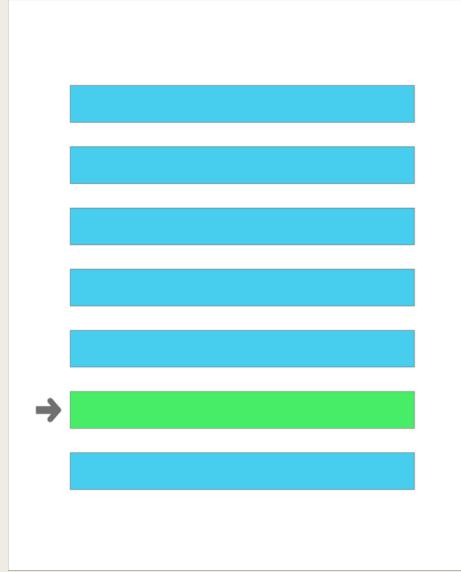
Serial Dispatch

The concept of async-await



Serial Dispatch

The concept of async-await



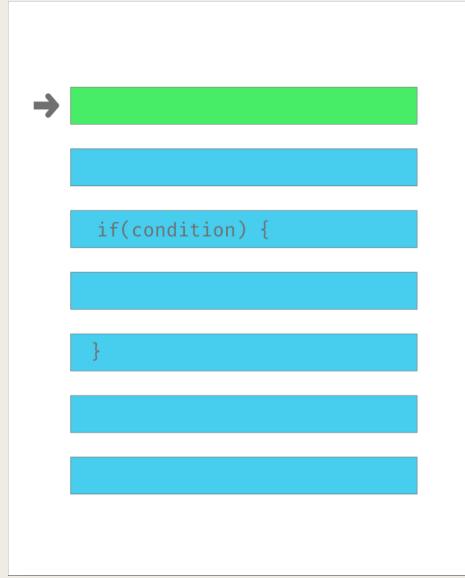
Serial Dispatch

The concept of async-await



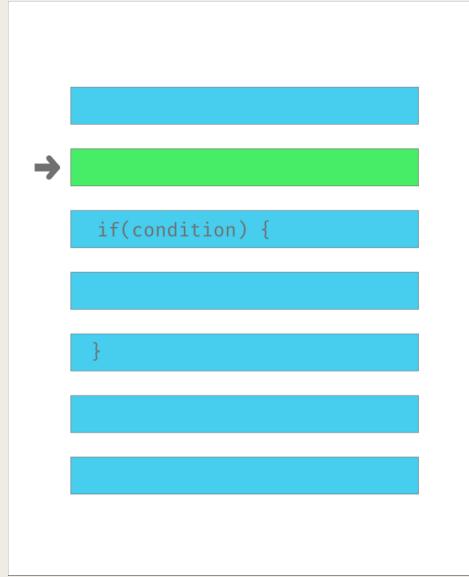
Serial Dispatch

The concept of async-await



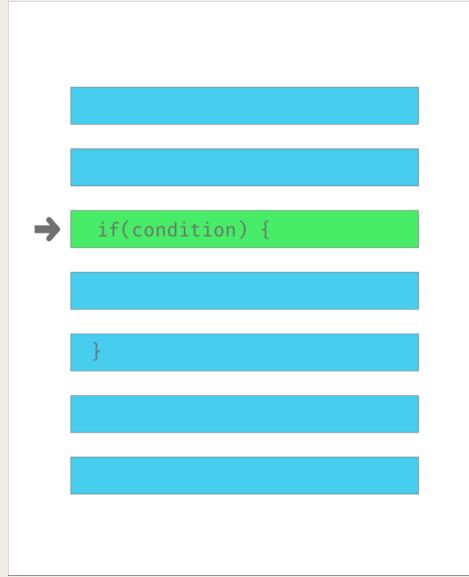
Conditionals Dispatch

The concept of async-await



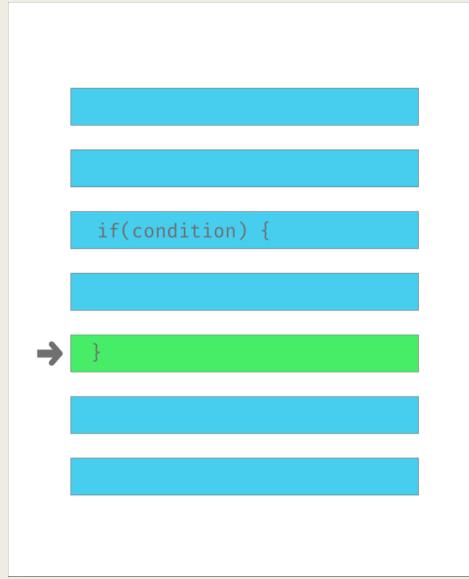
Conditionals Dispatch

The concept of async-await



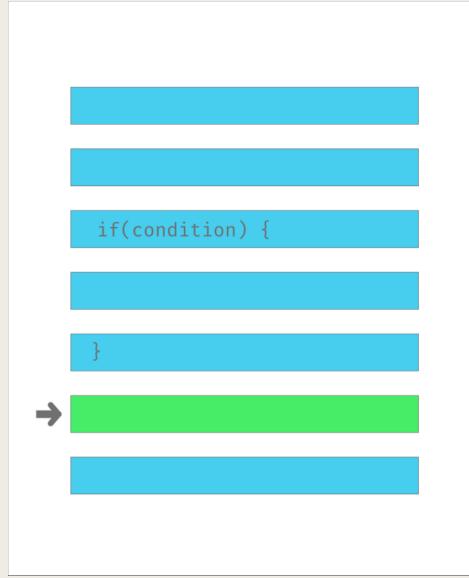
Conditionals Dispatch

The concept of async-await



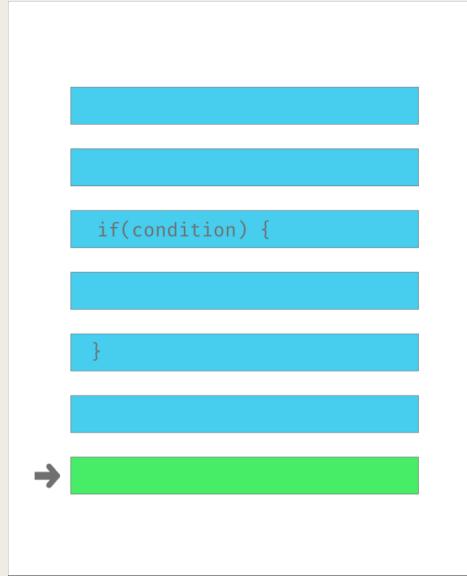
Conditionals Dispatch

The concept of async-await



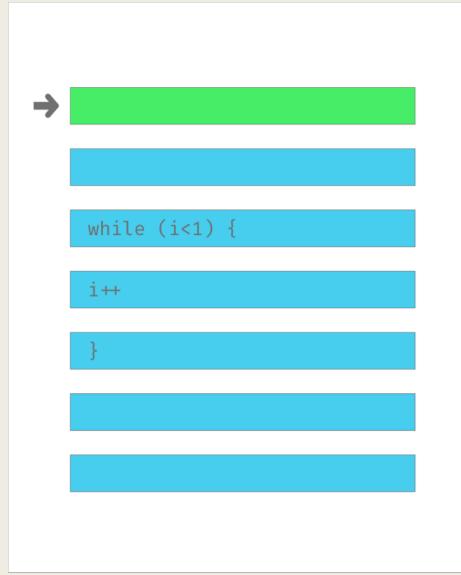
Conditionals Dispatch

The concept of async-await



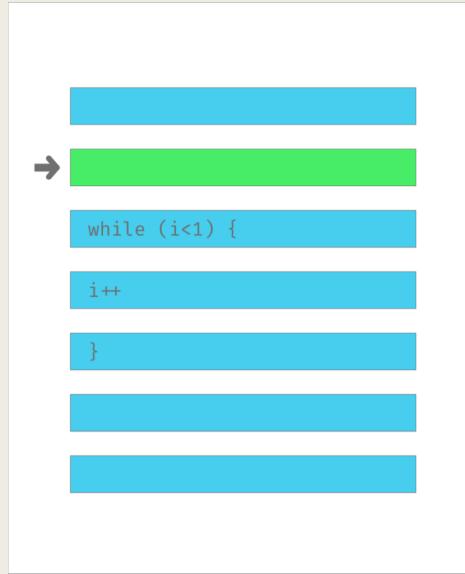
Conditionals Dispatch

The concept of async-await



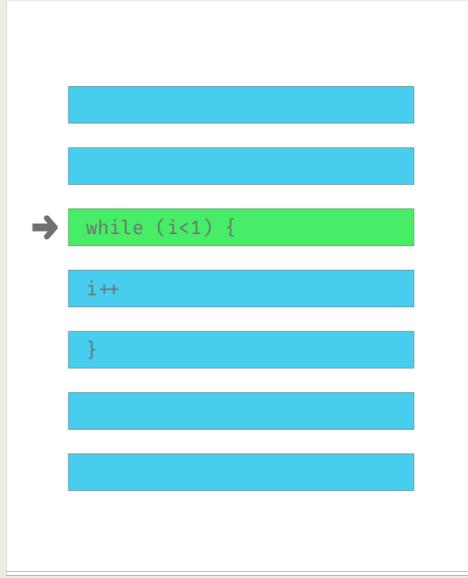
Loop

The concept of async-await



Loop

The concept of async-await

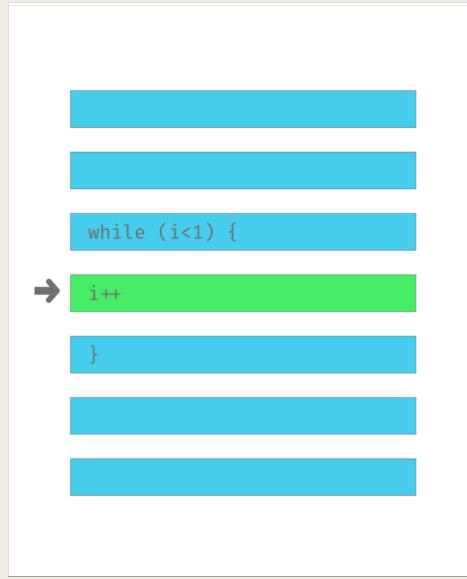


A diagram illustrating a loop structure. It consists of several horizontal bars of varying lengths. The first bar is long and light blue. Below it is a shorter light blue bar. To the left of these two bars is a green arrow pointing right, followed by the text "while (i<1) {". This is followed by a medium-length light blue bar containing the text "i++". Below that is another medium-length light blue bar containing the symbol "}".

```
while (i<1) {
    i++
}
```

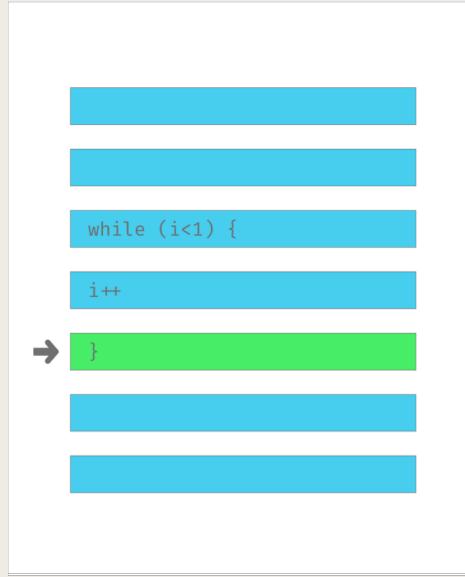
Loop

The concept of async-await



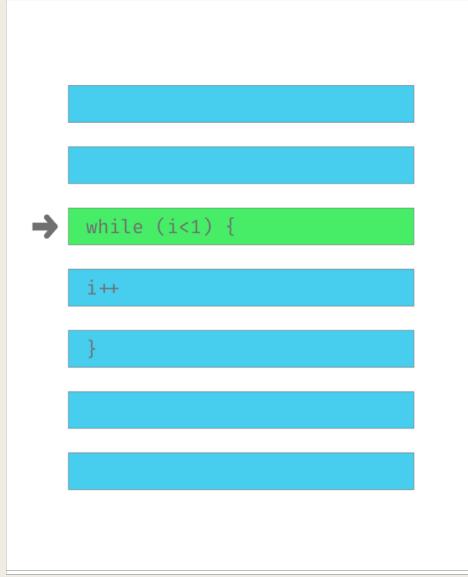
Loop

The concept of async-await



Loop

The concept of async-await

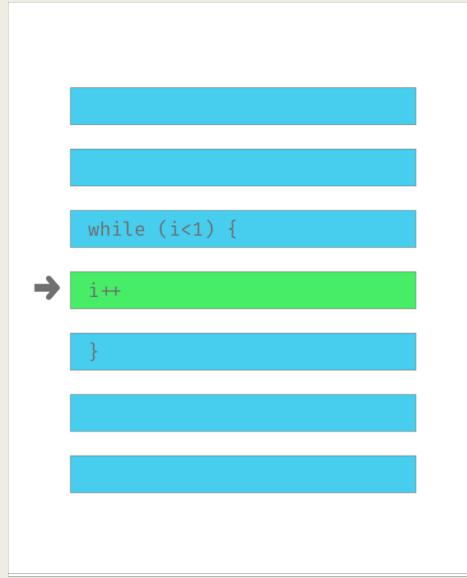


A diagram illustrating a loop structure. It consists of several horizontal bars of varying lengths. The first bar is long and light blue. Below it is a shorter light blue bar. To the left of these two bars is a green arrow pointing right, followed by the text "while (i<1) {". This is followed by a medium-length light blue bar containing the text "i++". Below that is another medium-length light blue bar containing the text "}".

```
while (i<1) {
    i++
}
```

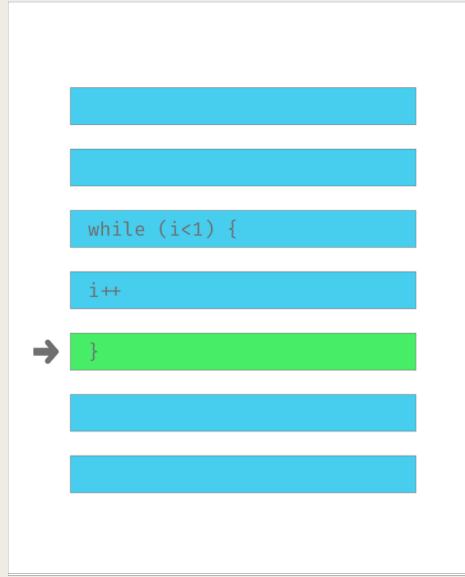
Loop

The concept of async-await



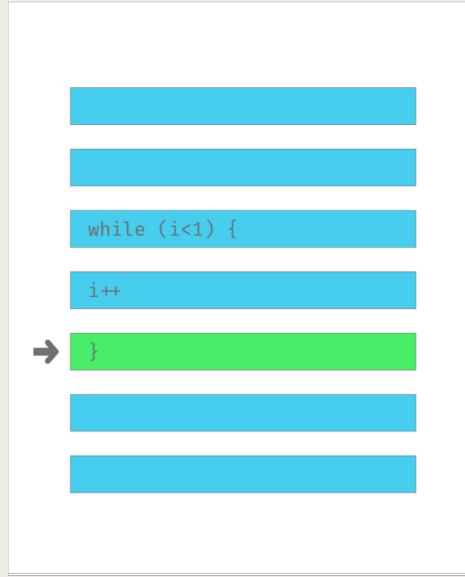
Loop

The concept of async-await



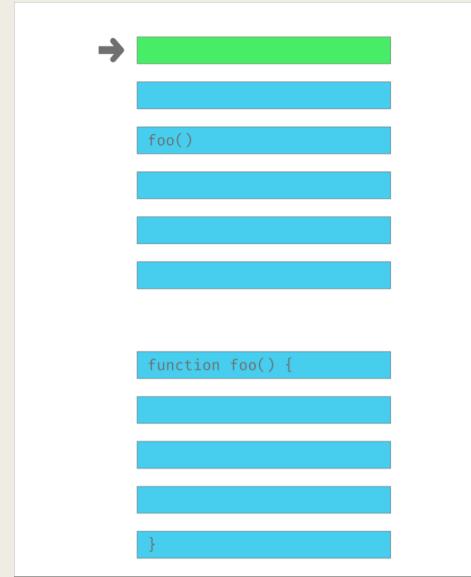
Loop

The concept of async-await



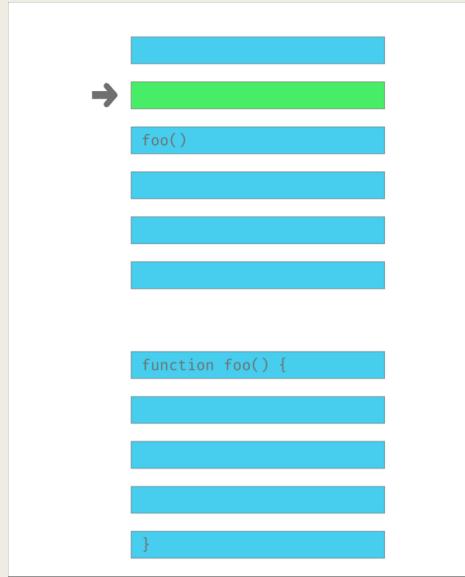
Loop

The concept of async-await



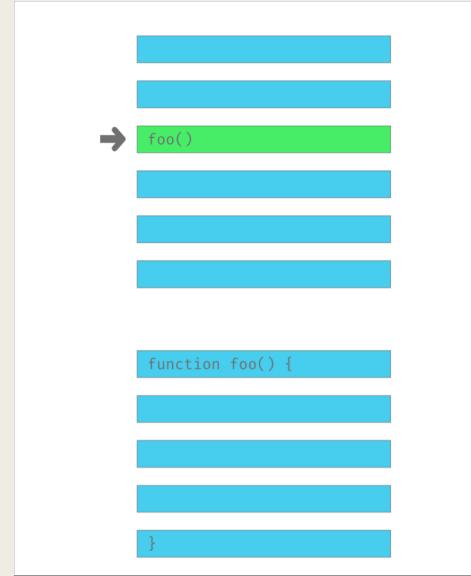
Function

The concept of async-await



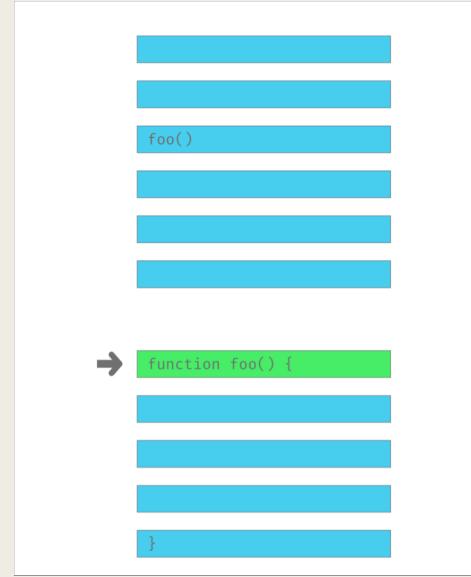
Function

The concept of async-await



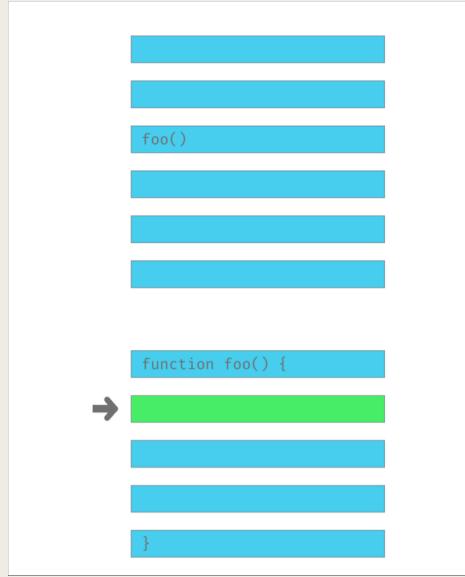
Function

The concept of async-await



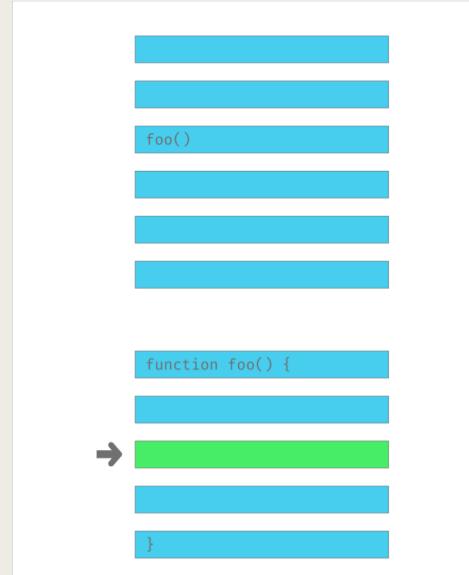
Function

The concept of async-await



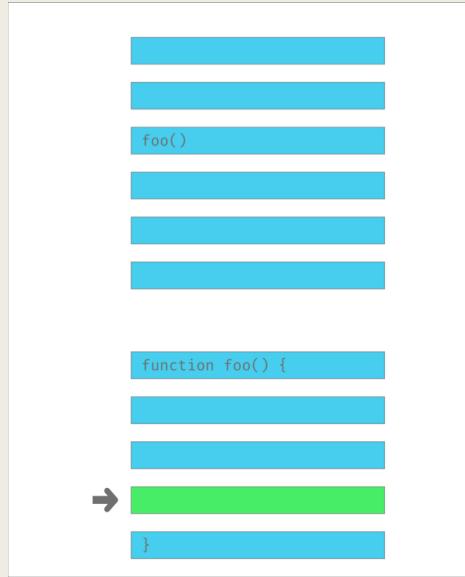
Function

The concept of async-await



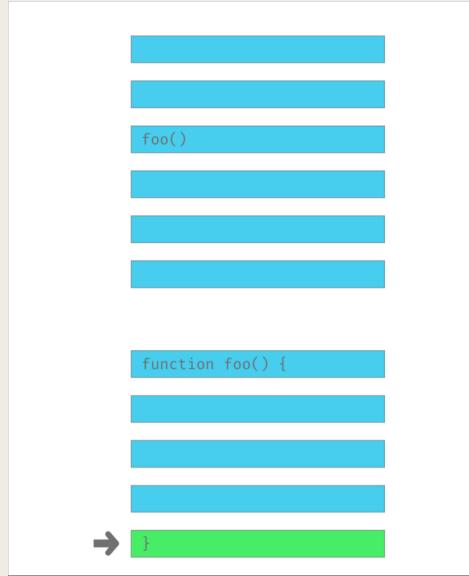
Function

The concept of async-await



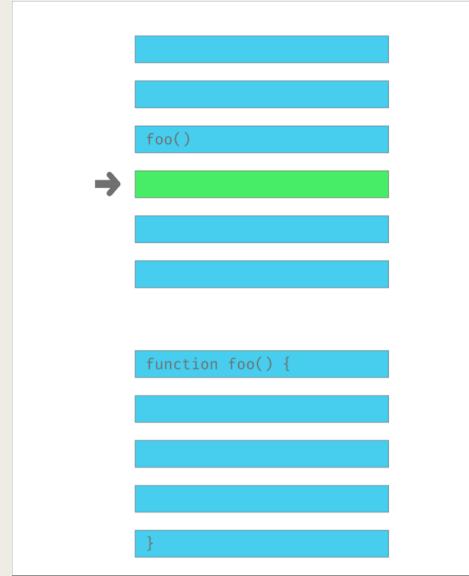
Function

The concept of async-await



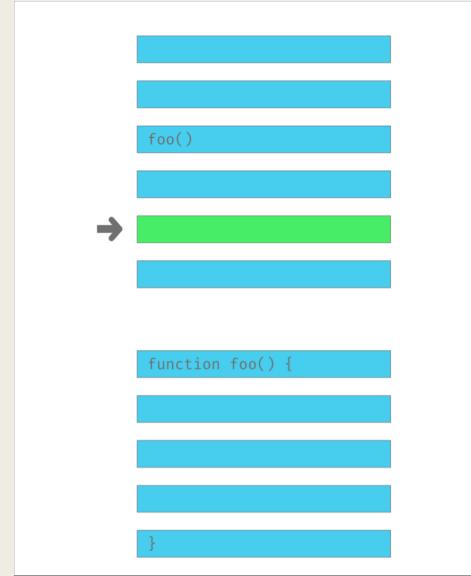
Function

The concept of async-await



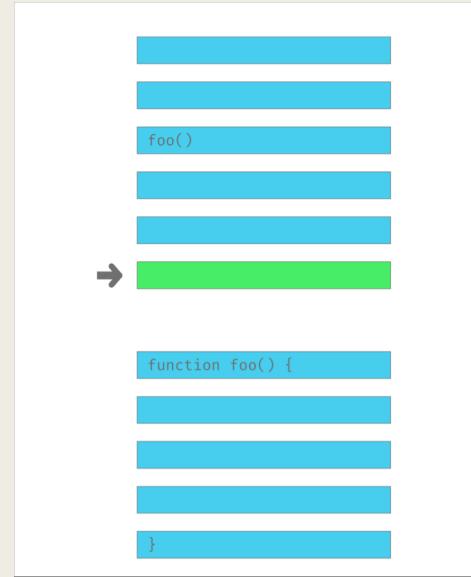
Function

The concept of async-await



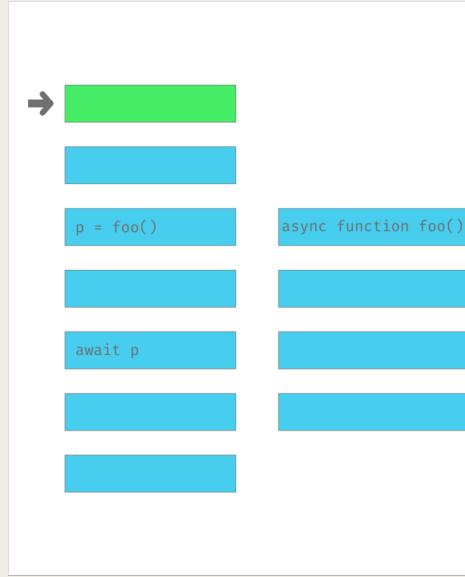
Function

The concept of async-await



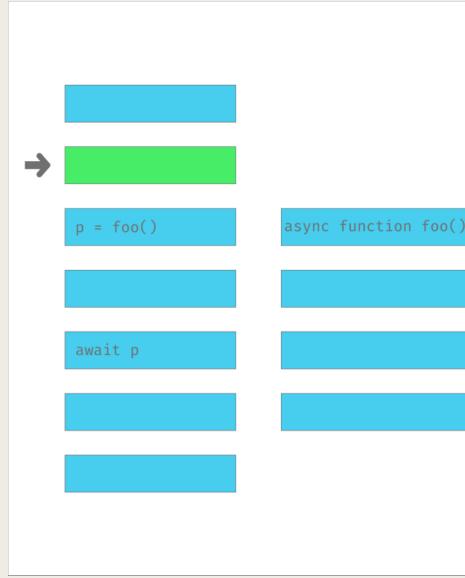
Function

The concept of async-await



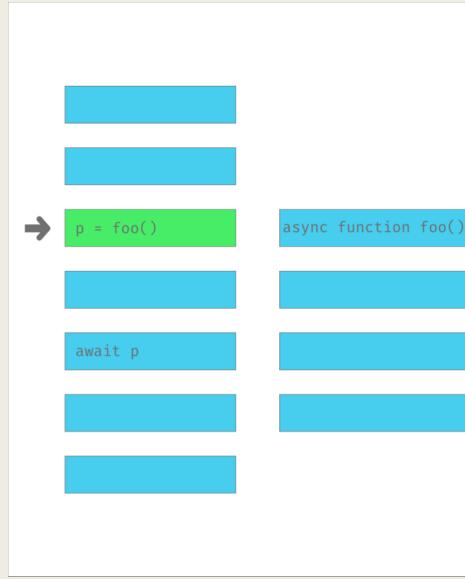
Async

The concept of async-await



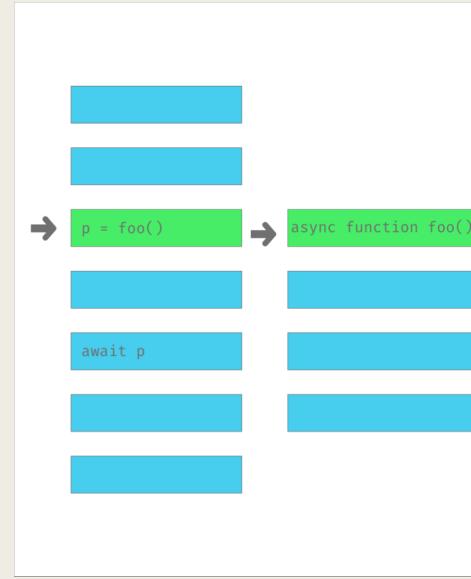
Async

The concept of async-await



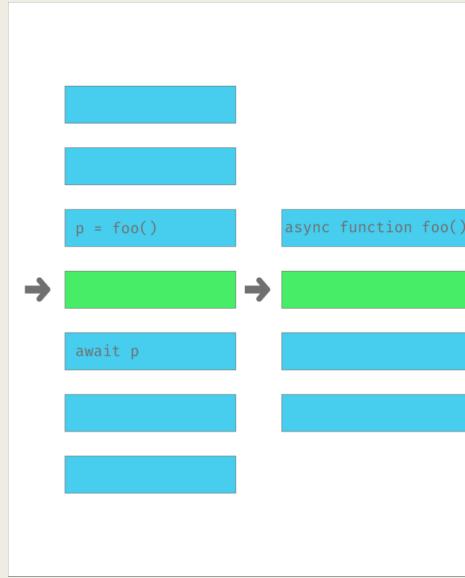
Async

The concept of async-await



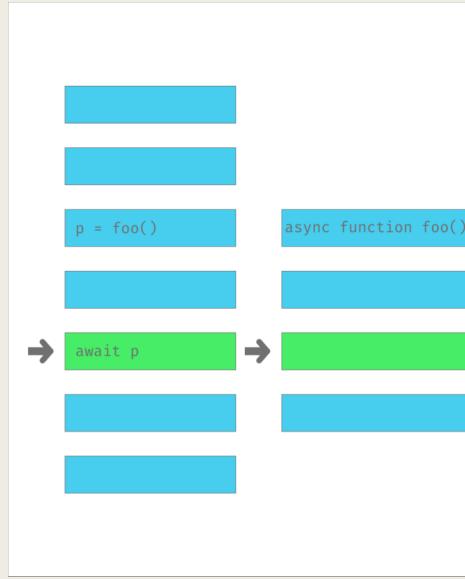
Async

The concept of async-await



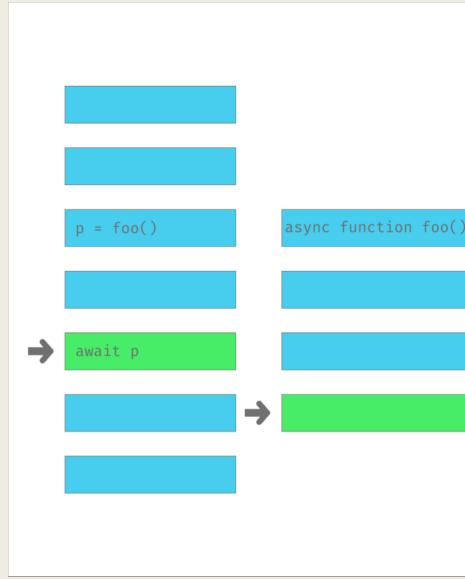
Async

The concept of async-await



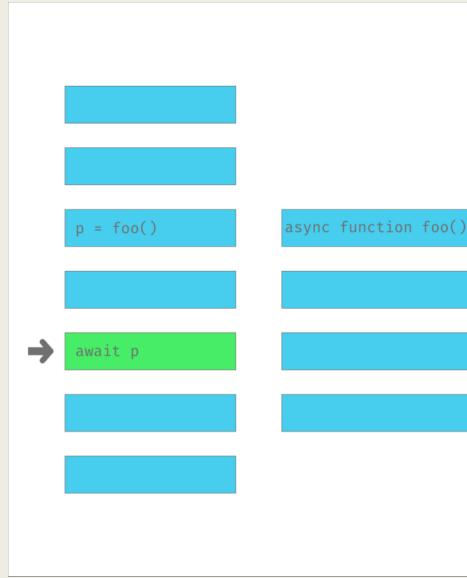
Async

The concept of async-await



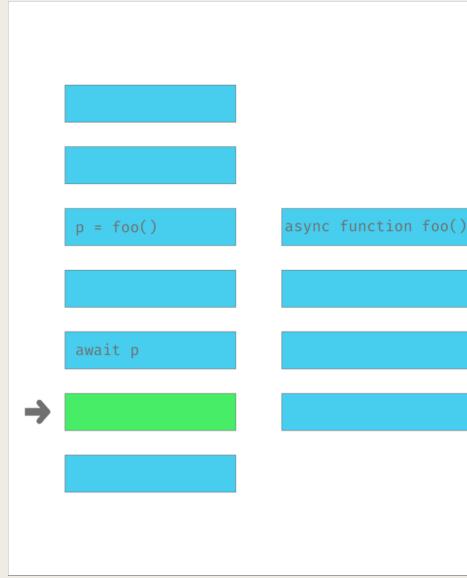
Async

The concept of async-await



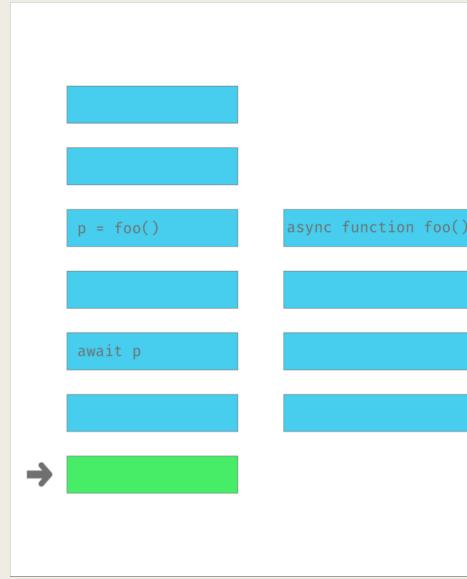
Async

The concept of async-await



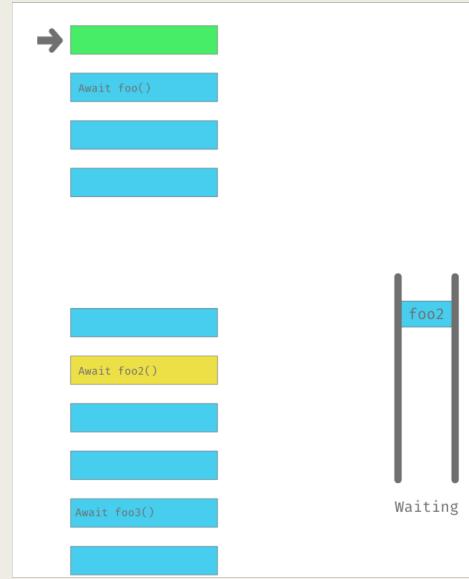
Async

The concept of async-await



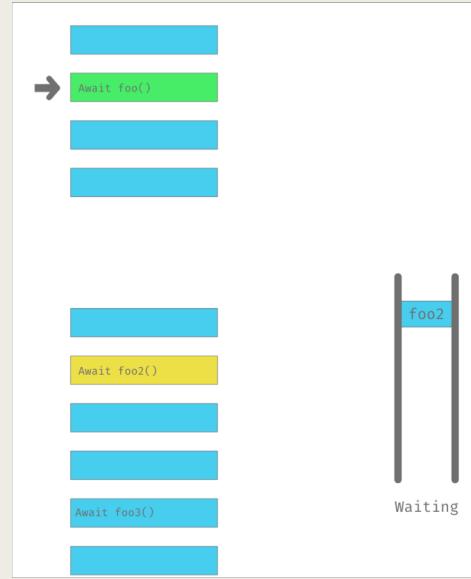
Async

The concept of async-await



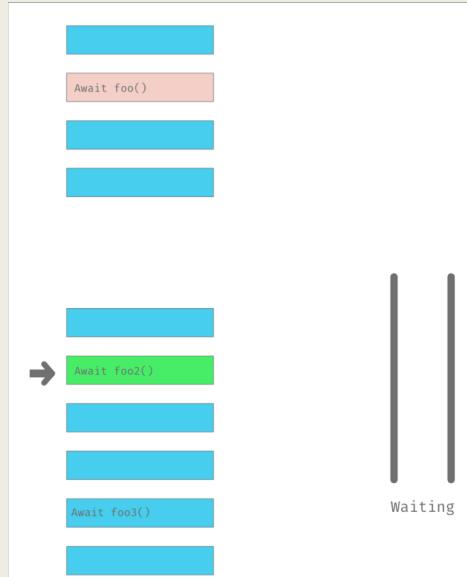
Event Dispatch

The concept of async-await



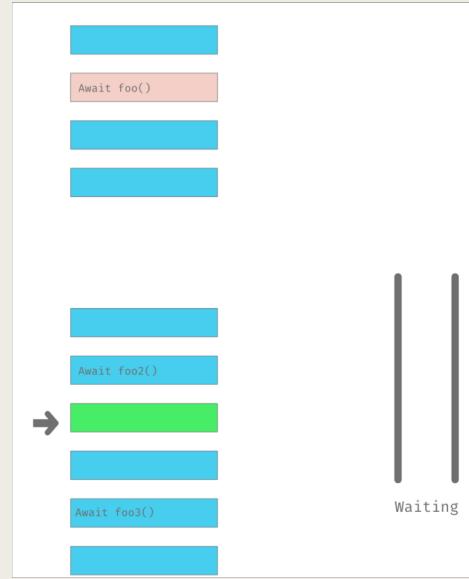
Event Dispatch

The concept of async-await



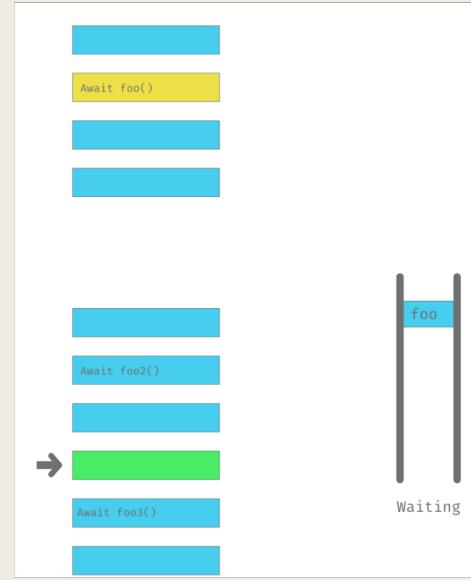
Event Dispatch

The concept of async-await



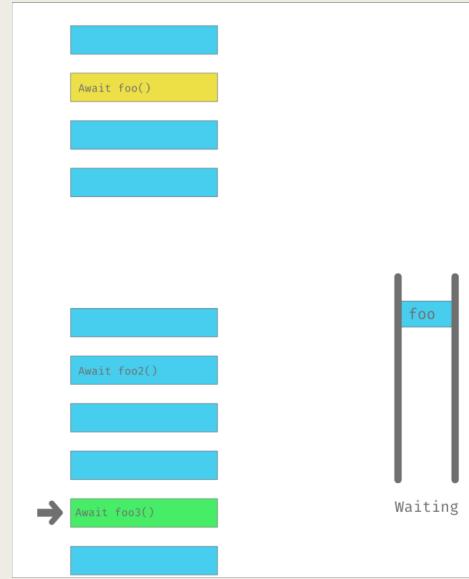
Event Dispatch

The concept of async-await



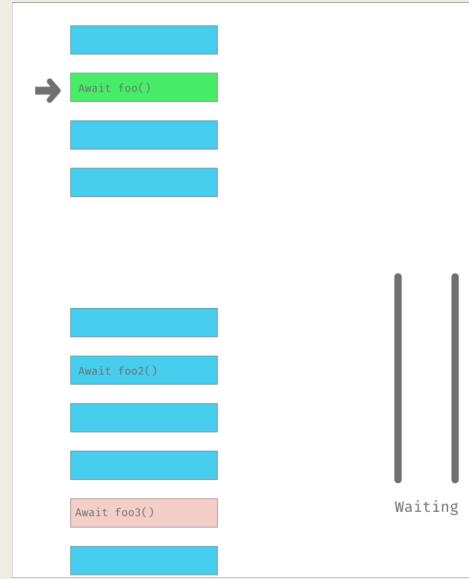
Event Dispatch

The concept of async-await



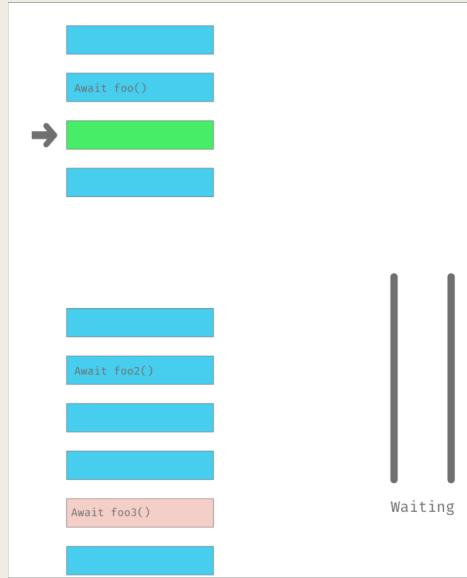
Event Dispatch

The concept of async-await



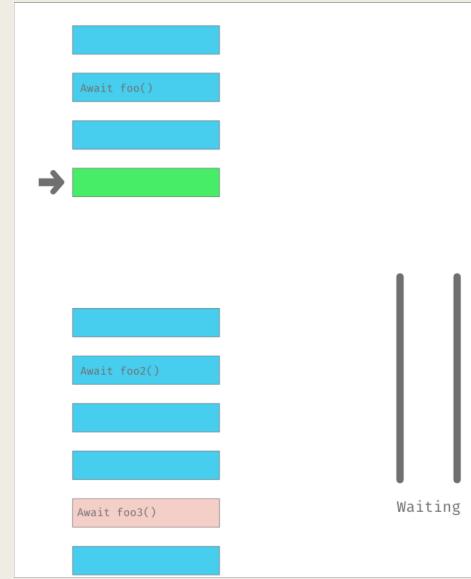
Event Dispatch

The concept of async-await



Event Dispatch

The concept of async-await



Event Dispatch

BEGINNING ASYNC-AWAIT

A callback vs promise vs async await

A callback vs promise vs async await

```
function work(data, cb) {
    db.connect((err) => {
        if(err) {return cb(err);}
        db.query(query1, (err, result1) =>
        {
            if(err) { return cb(err) }
            db.query(query2, cb);
        });
    });
}
```

Callback

```
function work() {
    return db.connect()
        .then(() => {
            return db.query(query1);
        })
        .then((result1)=> {
            return db.query(query2);
        });
}
```

Promises

```
async function work() {
    await db.connect();
    result1 = await db.query(query1);
    return await db.query(query2);
}
```

async await

BEGINNING ASYNC-AWAIT

The Program Flow

The program flow

```
async function foo() {  
    ...  
    await foo1();  
    ...  
    if(await foo2()) {  
        ...  
        await foo3()  
        ...  
    }  
    for(...) {  
        await foo4(...);  
    }  
    ...  
}
```

The natural waterfall model

The program flow

```
x = [...]
x.forEach(async y => {
    await foo(y);
});
```

The program flow

```
// Wrong will return promise to
// forEach and not wait
x = [...]
x.forEach(async y => {
    await foo(y);
});
```



The program flow

```
// Wrong will return promise to
// forEach and not wait
x = [...]
x.forEach(async y => {
    await foo(y);
});
```



```
// Right – will wait for each
// await in sequential order
x= [...];
for(y of x) {
    await foo(y)
}
```



Using for-of

Note: for..in does not make it to the JavaScript Good parts and should be avoided in the favor of for..of where possible.

INTERMEDIATE ASYNC-AWAIT

A detour to promises

A detour to promises



Image from [Pexels](#)

A detour to promises

- Promises have a then/catch/finally.

```
async function f() { ... }
await f()
  .then((resp) => doSomething(resp))
  .catch((e) => handleError(e))
  .finally(() => cleanup());
```

A detour to promises

- There is a concept of unhandled rejection.

```
process.on("unhandledRejection", ...);  
  
window.addEventListener("unhandledrejection", ...);
```

A detour to promises

- Use Promises to convert a callback based method to async

```
function x(data, callback) {
  y.doSomething(data, (err, z) => {
    if (err) { return callback(err); }
    processed = somePostProcessing(z);
    callback(null, processed)
  });
}
```

```
x(data, (err, processed) => { });
```

Conversion guide: <https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/>

A detour to promises

- Use Promises to convert a callback based method to async

```
function x(data, callback) {
  y.doSomething(data, (err, z) => {
    if (err) { return callback(err); }
    processed = somePostProcessing(z);
    callback(null, processed)
  });
}
```

```
x(data, (err, processed) => { });
```

```
function x(data) {
  return new Promise((resolve, reject) => {
    y.doSomething(data, (err, z) => {
      if (err) {reject(err);}
      processed = somePostProcessing(z);
      resolve(processed);
    });
});
```

```
const processed = await x(data);
```

Conversion guide: <https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/>

A detour to promises

- Use Promises to convert a callback based method to async

```
function x(data, callback) {
  y.doSomething(data, (err, z) => {
    if (err) { return callback(err); }
    processed = somePostProcessing(z);
    callback(null, processed)
  });
}
```

```
x(data, (err, processed) => { });
```

```
async function x(data) {
  return new Promise((resolve, reject) => {
    y.doSomething(data, (err, z) => {
      if (err) {reject(err);}
      processed = somePostProcessing(z);
      resolve(processed);
    });
});
```

```
const processed = await x(data);
```

Conversion guide: <https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/>

INTERMEDIATE ASYNC-AWAIT

The Program Flow

The program flow

```
async foo() {  
    ...  
    await fetch()  
    ...  
}  
  
const p = foo();  
// Do some parallel tasks  
...  
await p;
```

Delayed await

The program flow

```
const tasks = [
  fetch(p1),
  fetch(p2),
  fetch(p3)
];

responses = await Promise.all(tasks);
```

Running stuff in parallel

Note: This will return quickly in case of exceptions. There are ways to workaround this, while the standards committee is working on [Promise.allSettled](#) to provide the easy solution.

The program flow

```
await Promise.all([...].map(async x => {
  ...
  await foo(x);
  ...
}));
```

Using array.map

The program flow

```
// setTimeout async version
const timeout = async (time) => new Promise(resolve => setTimeout(resolve, time));

// Adding a timeout to fetch
await Promise.race([fetch(...), timeout(2000)]);
```

Racing functions

The program flow

```
function handleError(error) {
  console.log(error);
  return fallbackFoo2Response;
}

await foo1(...);
await foo2(...).catch(handleError);
await foo3(...)
```

```
await Promise.all([
  .map(async () => {
    ...
  })
  .map(x => x.catch(e => e))
]);
```

Errors without try...catch

ADVANCED ASYNC-AWAIT

Async guarantees

Async guarantees

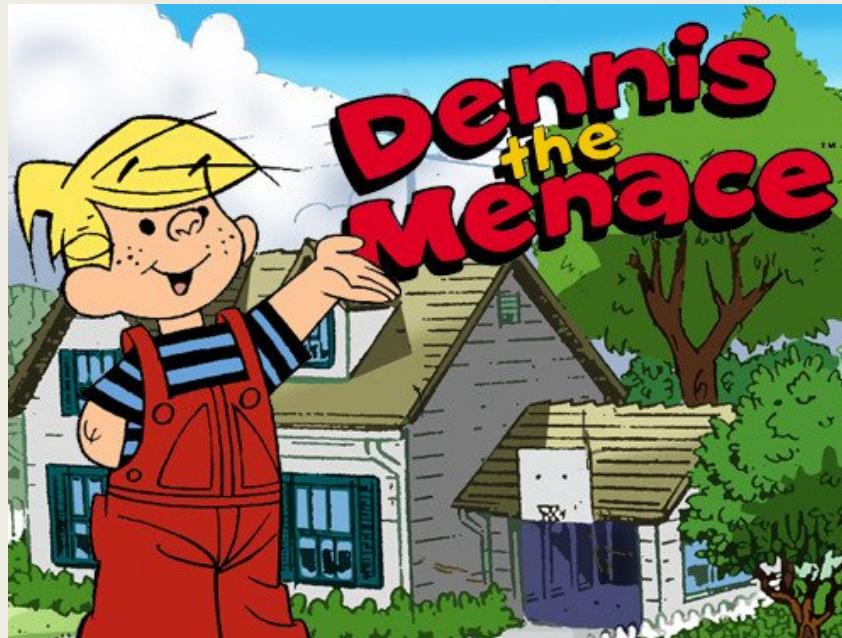


Image from
[IMDB](#)

Async guarantees

```
undefined = true;
```

```
foo.apply(window, arguments)
```

```
function(cb) {  
  x((err) => {  
    if(err) { callback(err); }  
    callback(true);  
  })  
}
```

Async guarantees

- It always returns once
- There is only one return value
- It is always a promise

```
undefined = true;
```

```
foo.apply(window, arguments)
```

```
function(cb) {  
  x((err) => {  
    if(err) { callback(err); }  
    callback(true);  
  })  
}
```

ADVANCED ASYNC-AWAIT

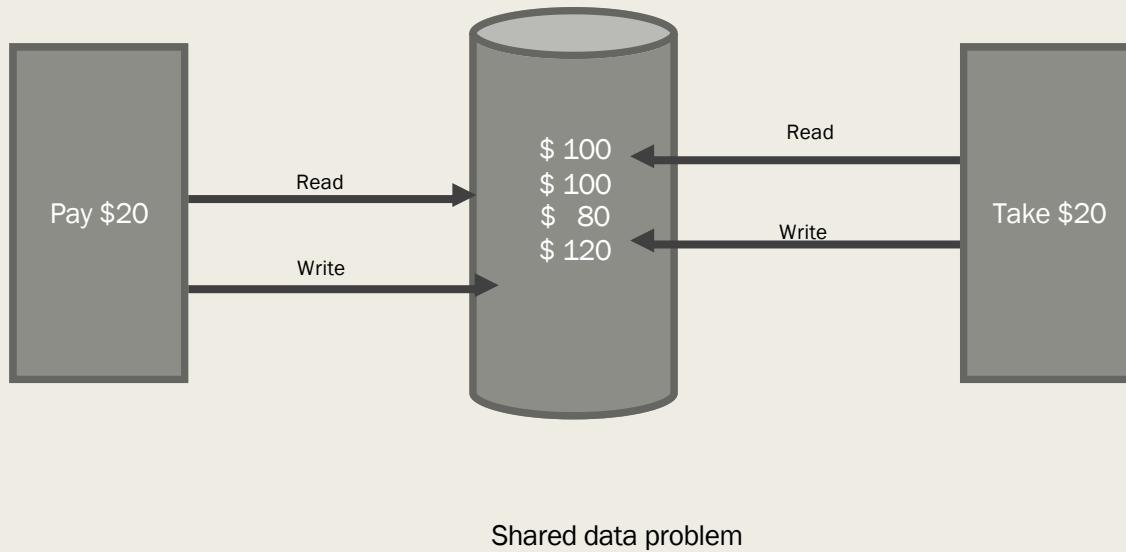
Wrapping asynchronous functions

Wrapping asynchronous functions



Image from [Pixabay](#)

Wrapping asynchronous functions



Wrapping asynchronous functions

```
async a() {  
    await lock();  
    await fs.readFile(file);  
    await fs.writeFile(file);  
    unlock()  
}  
  
async b() {  
    await lock()  
    await fs.readFile(file);  
    await fs.writeFile(file);  
    unlock();  
}
```

```
let mutex = false;  
  
async lock() {  
    while(mutex) { await timeout(100); }  
    mutex = true;  
}  
  
unlock() {  
    mutex = false;  
}
```

Shared data and locks

Wrapping asynchronous functions

```
async a() {  
    await lock();  
    await fs.readFile(file);  
    await fs.writeFile(file);  
}
```



Fixing forgotten lock call problem

Wrapping asynchronous functions

```
async a() {  
    await lock();  
    await fs.readFile(file);  
    await fs.writeFile(file);  
}
```



```
async a() {  
    return await transaction(async () =>  
    {  
        await fs.readFile(file);  
        await fs.writeFile(file);  
    });  
}
```



Fixing forgotten lock call problem

Wrapping asynchronous functions

```
async transaction(method) {
    await lock();
    const value = await method().catch(e => {
        unlock();
        throw e;
    });
    unlock();
    return value;
}
```

The transaction wrapper

Wrapping asynchronous functions

```
async time(key, method) {
    console.time(key);
    return await method().finally(() => {
        console.timeEnd(key);
    });
}

// Usage
await time('myMethod', async ()=> {
    //do your work
})
```

A better way - the time wrapper

Wrapping asynchronous functions

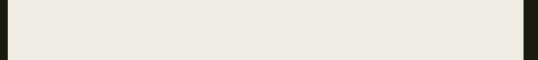
```
async time(key, method) {
  const context = {};
  console.time(key);
  return await method(context).finally(() => {
    console.timeEnd(key);
    fetch(url, {body: context});
  });
}

// Usage
await time(async (context)=> {
  //do your work
  context.x = "some result;
});
```

Include context

ADVANCED ASYNC-AWAIT

Exploiting the promises underneath



Exploiting promises in async

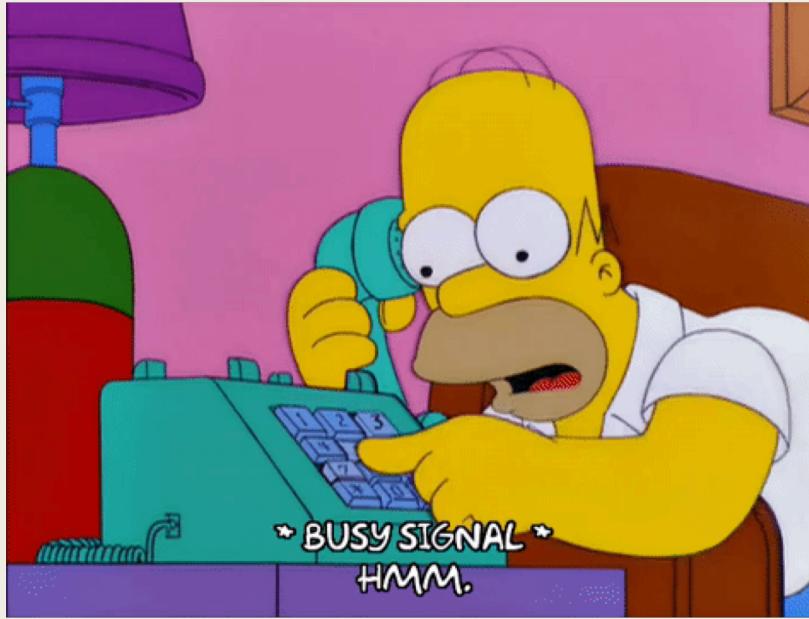


Image from [gifer](#)

The Parallel calls problem

Exploiting promises in async

```
function get(cb) {
  if (data) {
    cb(data);
  } else {
    callbacks.push(cb);
  }
  if (callbacks.length === 1) {
    ...
    callbacks.forEach(cb => cb(data));
  }
}
```

Doing it once

Exploiting promises in async

```
let get = memoize(async () => {
  ...
  return data;
});
```

Memoize method

Exploiting promises in async

```
1  function memoize(func, resolver) {
2      const memoized = function(...args) {
3          const key = resolver ? resolver.apply(this, args) :
4              args[0]
5          const cache = memoized.cache
6
7          if (cache.has(key)) {
8              return cache.get(key)
9          }
10         const result = func.apply(this, args)
11         memoized.cache = cache.set(key, result) || cache
12         return result
13     }
14     memoized.cache = new Map();
15     return memoized
16 }
```

Lodash - Memoize method

<https://github.com/lodash/lodash/blob/master/memoize.js>

Exploiting promises in async

```
9.1      if (result instanceof Promise) {  
9.2          result.state = 'pending';  
9.3          result.then(() => result.state = 'resolved')  
9.4              .catch(() => result.state = 'rejected');  
9.5      }
```

```
6      if (cache.has(key)) {  
6.1          let x = cache.get(key);  
6.2          if (x instanceof Promise) {  
6.3              if (x.state === 'pending') {  
7                  return x;  
7.1              }  
7.2          }  
8      }
```

Reuse the promise only if pending

Exploiting promises in async

```
// Convert to ES6 Decorator
function Memoize() {
    return function (target, functionName) {
        target[functionName] = memoize(target[functionName]);
    };
}

@Memoize
async get() {
    ...
    return data;
}
```

Creating decorator for memoize

ADVANCED ASYNC-AWAIT

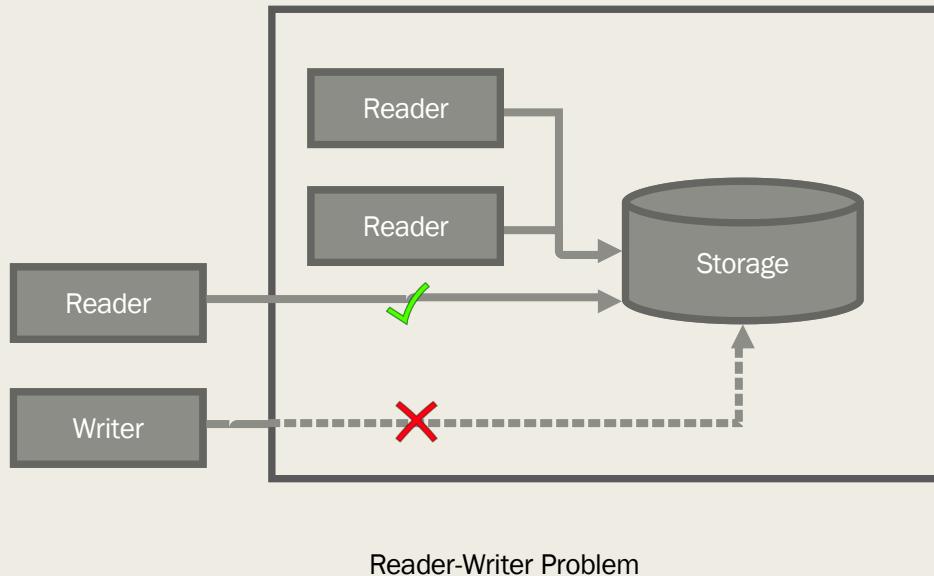
Exploiting callbacks deep down

Exploiting callbacks deep down



Reader-Writer Problem

Exploiting callbacks deep down



Exploiting callbacks deep down

```
await lock.read(async ()=> {  
    // Do reading  
});  
  
await lock.write(async () => {  
    // Do writing  
});
```

ReadWrite Lock API

Exploiting callbacks deep down

```
class Lock {
    pendingReads = []
    pendingWrites = []
    async read(func) { // Cannot use await here
        return new Promise(resolve => {
            pendingReads.push({func, cb: resolve});
            this.perform();
        });
    }
    async write(func) { // Cannot use await here
        return new Promise(resolve => {
            pendingWrites.push({func, cb: resolve});
            this.perform();
        });
    }
    perform() {
        ...
    }
}
```

Structure of the Lock class

Exploiting callbacks deep down

```
perform() {
    if (this.state === 'None' && this.pendingWrites.length > 0) {
        this.state = 'Write';
        const {func, cb} = this.pendingWrites.shift();
        func().finally(() => {
            cb(); this.state = 'None'; this.perform();
        })
    }

    if (this.state !== 'Write' && this.pendingReads.length > 0) {
        this.pendingReads.forEach(({func, cb})=> {
            this.state = 'Read';
            this.readInProgess++;
            func().finally(() => {
                cb();
                this.readInProgess--;
                if (this.readInProgess === 0) { this.state === 'None'; }
                this.perform();
            })
        });
    }

    readInProgess = 0
    state = 'None'
```

Structure of the Lock class

CONCLUSION

Conclusion

- Where callbacks rule:
 - *addEventListener*
 - *Method Wrappers (can be @decorators as well).*
- Where Promises rule
 - *Short hands with async*
- Where async await rule
 - *Everywhere else*



Image from [giphy](#)

THANK YOU

Atishay Jain

<https://atishay.me>

contact@atishay.me