

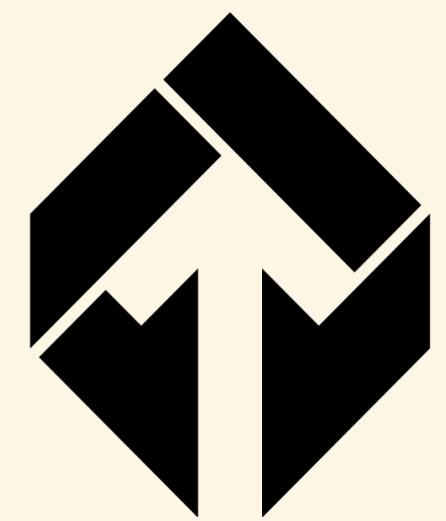
the RISE of async... JavaScript



Jeremy Fairbank

blog.jeremyfairbank.com

@elpapapollo | [gh/jfairbank](https://github.com/jfairbank)

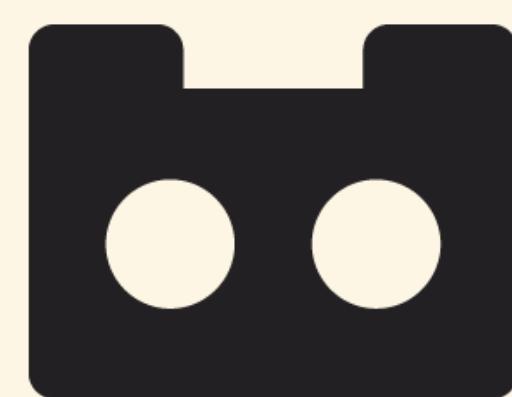


Push

We help brands excel.
pushagency.io

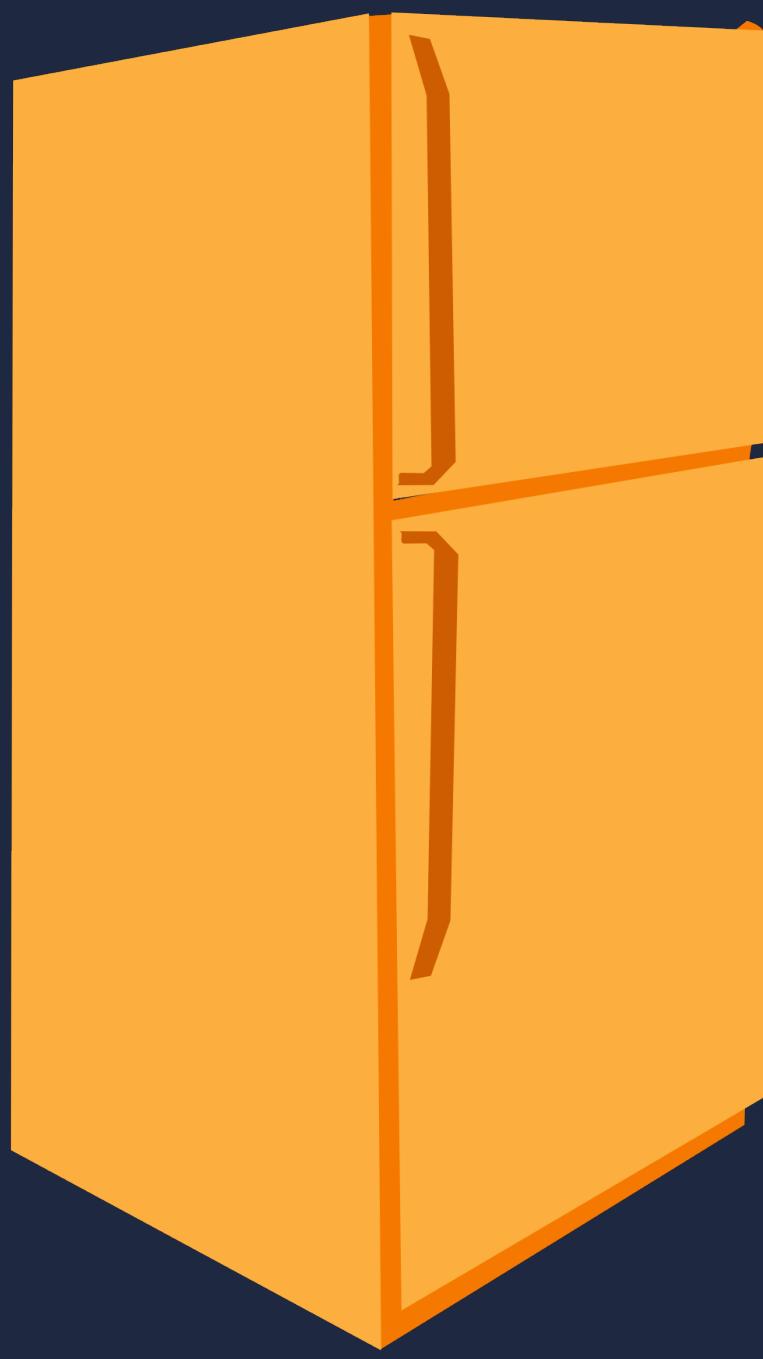
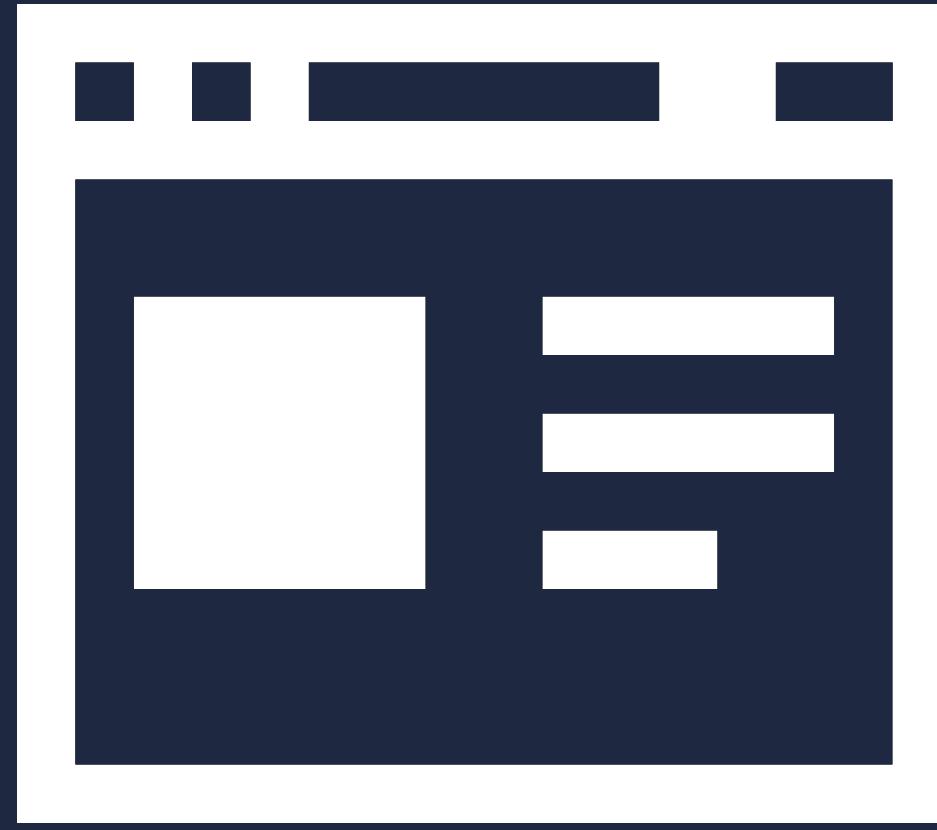
The Push website homepage features a dark header with the Push logo and navigation links for Services, About, Work, and Contact. Below the header is a large image of a hand holding a smartphone displaying a website. To the right of the phone, the text "SimplyBuilt.com" and "Get a professional website that looks great on all devices." is displayed, along with a "Learn More" button. At the bottom of the page are three circular icons with corresponding text: "Strategy" (a chess knight icon), "Design" (a pencil icon), and "Technology" (a server icon).

The SimplyBuilt website builder interface is shown against a dark background. It features a sidebar with icons for Account, Pages, Blocks, Styles, Settings, and Help. A preview window shows a website with a sunset background and the text "Welcome to your Website." Below the preview are several small thumbnail images. Various office-related icons are scattered around the interface, including paperclips, headphones, a ruler, and a potted plant.

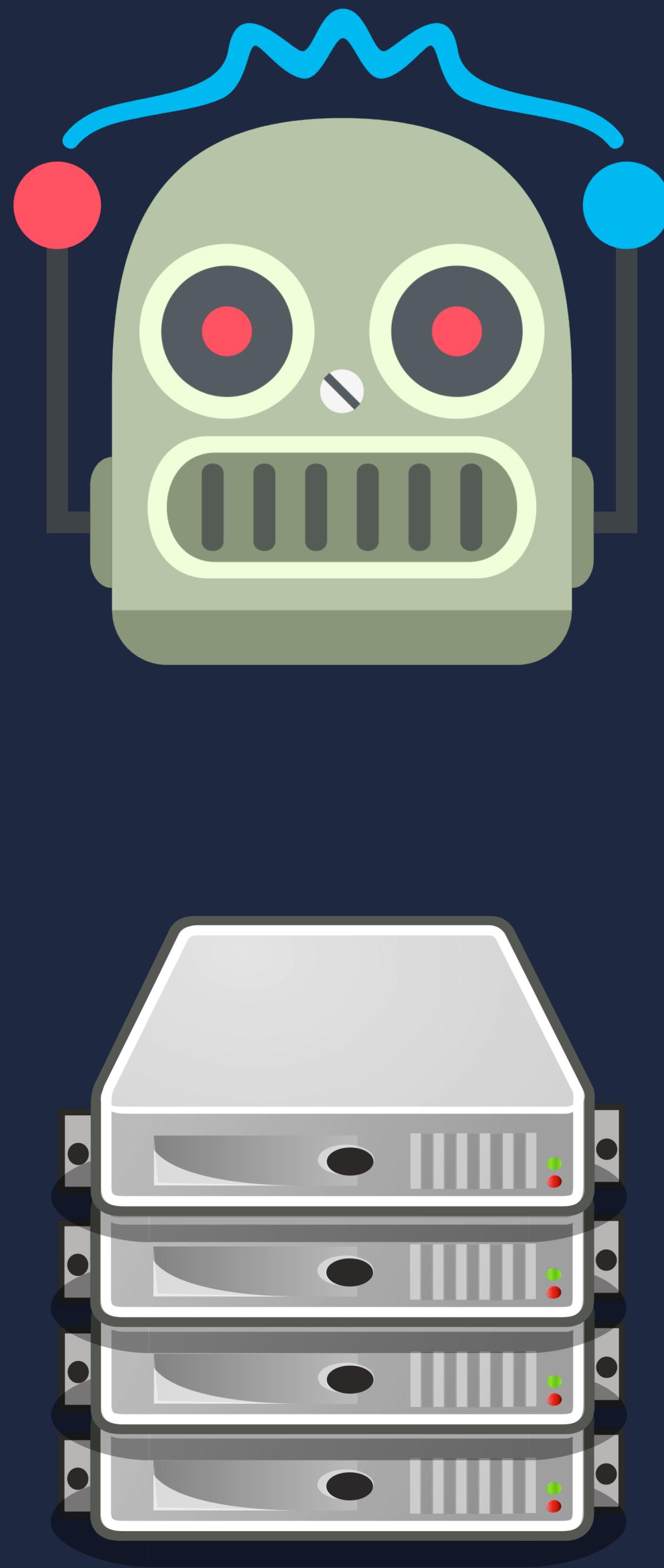


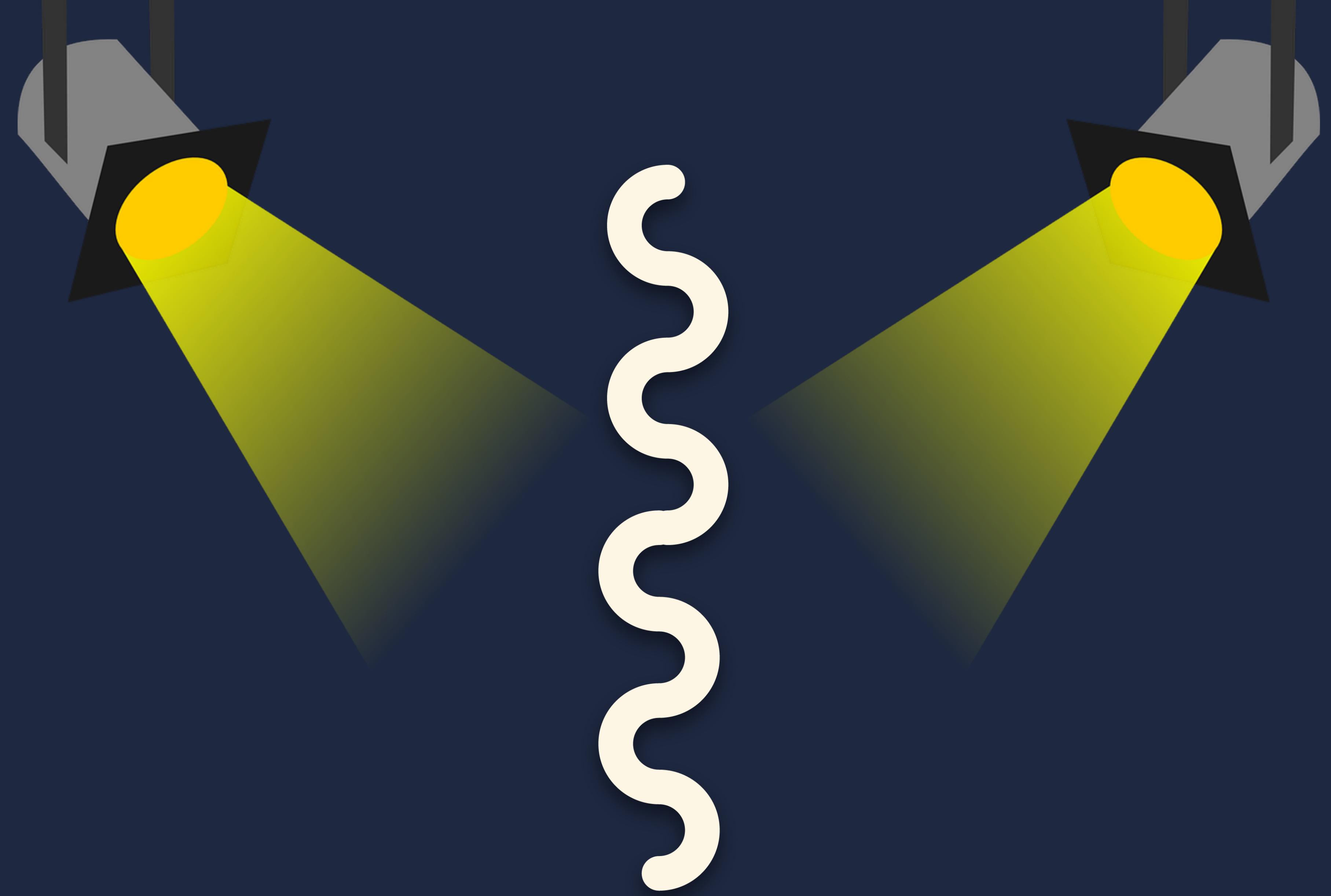
SimplyBuilt

Your website, SimplyBuilt.
simplybuilt.com



JS

A large, bold, black "JS" logo centered on a yellow background. The letters are stylized with rounded edges and a slight drop shadow.







“Call me maybe?”





Callback

```
fetchUserById(1, function(err, user) {  
  if (err) {  
    console.error('Could not retrieve user');  
  } else {  
    console.log(user);  
  }  
});
```



```
function fetchCustomerNameForOrder(orderId, done, fail) {
  fetchOrder(orderId, function(err, order) {
    if (err) {
      logError(err);
      fail(err);
    } else {
      fetchCustomer(
        order.customerId,
        function(err, customer) {
          if (err) {
            logError(err);
            fail(err);
          } else {
            done(customer.name);
          }
        }
      );
    }
  });
}
```

```
function fetchCustomerNameForOrder(orderId, done, fail) {  
  fetchOrder(orderId, function(err, order) {  
    if (err) {  
      logError(err);  
      fail(err);  
    } else {  
      fetchCustomer(  
        order.customerId,  
        function(err, customer) {  
          if (err) {  
            logError(err);  
            fail(err);  
          } else {  
            done(customer.name);  
          }  
        }  
    }  
  })  
}
```



```
function fetchCustomerNameForOrder(orderId, done, fail) {  
    fetchOrder(orderId, function(err, order) {  
        if (err) {  
            logError(err); ←  
            fail(err); ←  
        } else {  
            fetchCustomer(  
                order.customerId,  
                function(err, customer) {  
                    if (err) {  
                        logError(err); ←  
                        fail(err); ←  
                    } else {  
                        done(customer.name);  
                    }  
                }  
            );  
        }  
    });  
}
```

Async API



```
function fetchCustomerNameForOrder(orderId) {  
  return fetchOrder(orderId)  
    .then(order => fetchCustomer(order.customerId))  
    .then(customer => customer.name);  
}
```

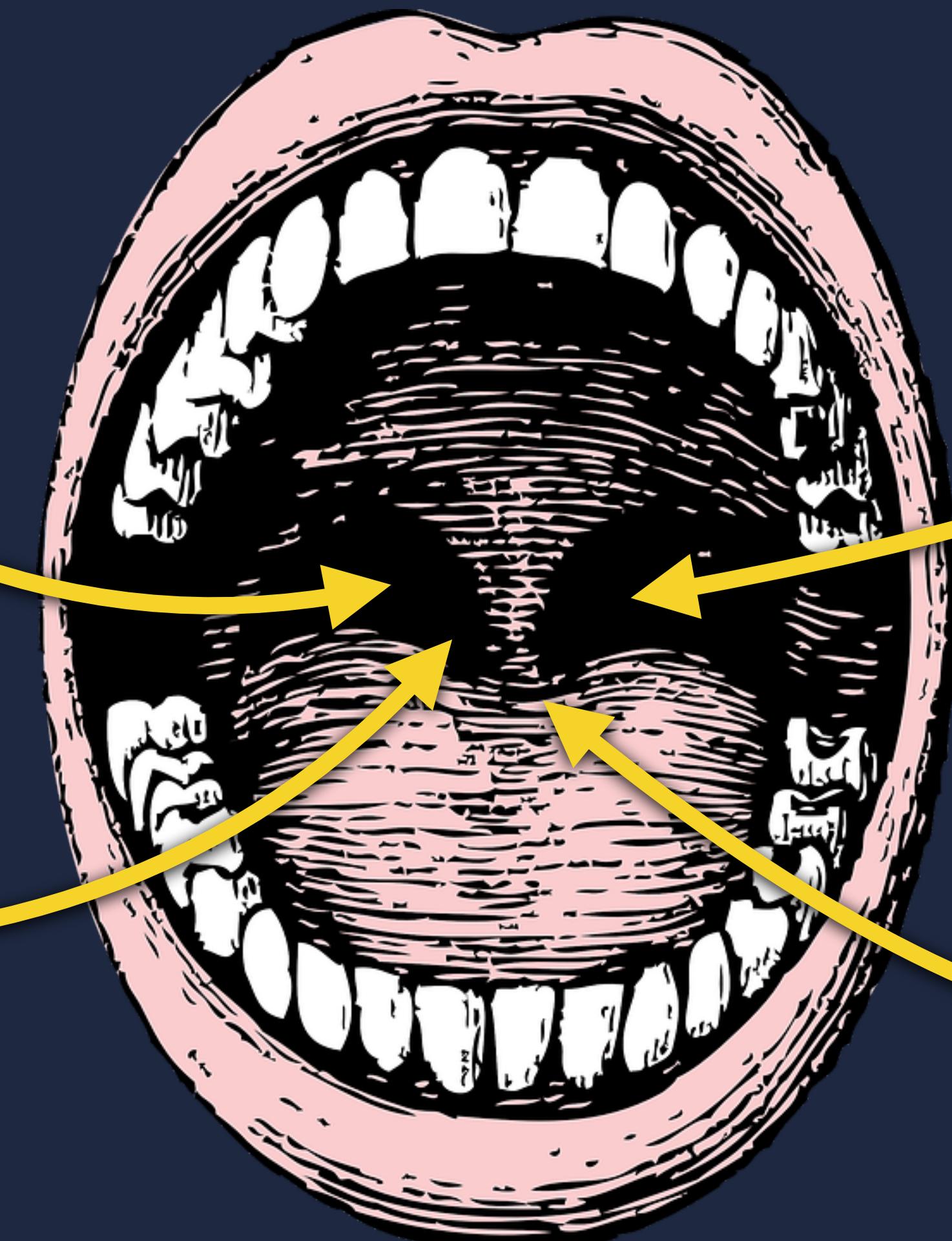


Error

Error

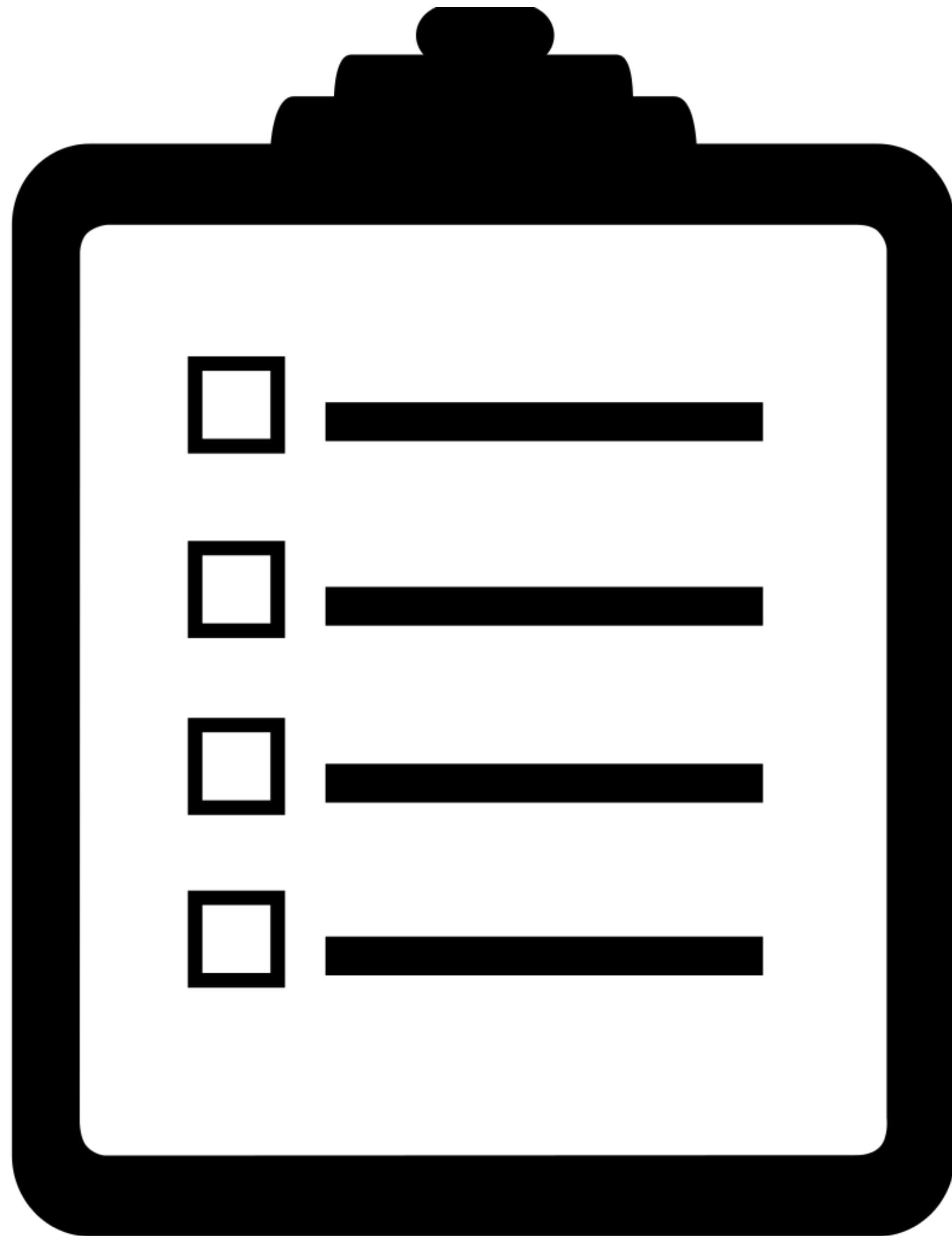
Error

Error



```
function fetchCustomerNameForOrder(orderId) {  
  return fetchOrder(orderId)  
    .then(order => fetchCustomer(order.customerId))  
    .then(customer => customer.name)  
    .catch(err => {  
      logError(err); ←  
      throw err;  
    });  
}
```

Getting there...



- Readable, synchronous program flow
- Nonblocking
- Use native flow control constructs

```
async function fetchCustomerNameForOrder(orderId) {  
  try {  
    const order = await fetchOrder(orderId);  
    const customer = await fetchCustomer(order.customerId);  
  
    return customer.name;  
  } catch (err) {  
    logError(err);  
    throw err;  
  }  
}
```

```
async function fetchCustomerNameForOrder(orderId) {  
  try {  
    const order = await fetchOrder(orderId);  
    const customer = await fetchCustomer(order.customerId);  
  
    return customer.name;  
  } catch (err) {  
    logError(err);  
    throw err;  
  }  
}
```



```
async function fetchCustomerNameForOrder(orderId) {  
  try {  
    const order = await fetchOrder(orderId);  
    const customer = await fetchCustomer(order.customerId);  
  
    return customer.name; ←—————  
  } catch (err) {  
    logError(err);  
    throw err;  
  }  
}
```

```
async function fetchCustomerNameForOrder(orderId) {  
  try { ←  
    const order = await fetchOrder(orderId);  
    const customer = await fetchCustomer(order.customerId);  
  
    return customer.name;  
  } catch (err) { ←  
    logError(err);  
    throw err;  
  }  
}
```



Await

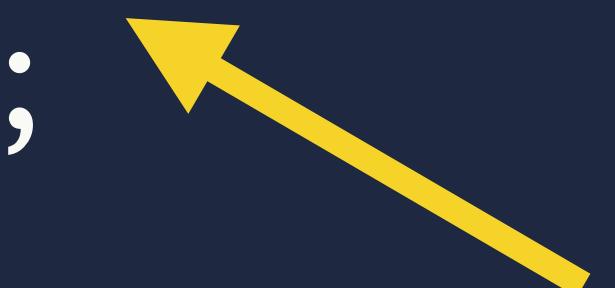
```
function fetchOrder(orderId) {  
  return fetch(`/orders/${orderId}`)  
    .then(response => response.json());  
}
```

```
function printOrder(orderId) {  
  fetchOrder(orderId)  
    .then(order => console.log(order));  
}
```



```
function fetchOrder(orderId) {  
  return fetch(`/orders/${orderId}`)  
    .then(response => response.json());  
}
```

```
async function printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);
```


}

`fetchOrder(orderId)` →



```
async printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

await



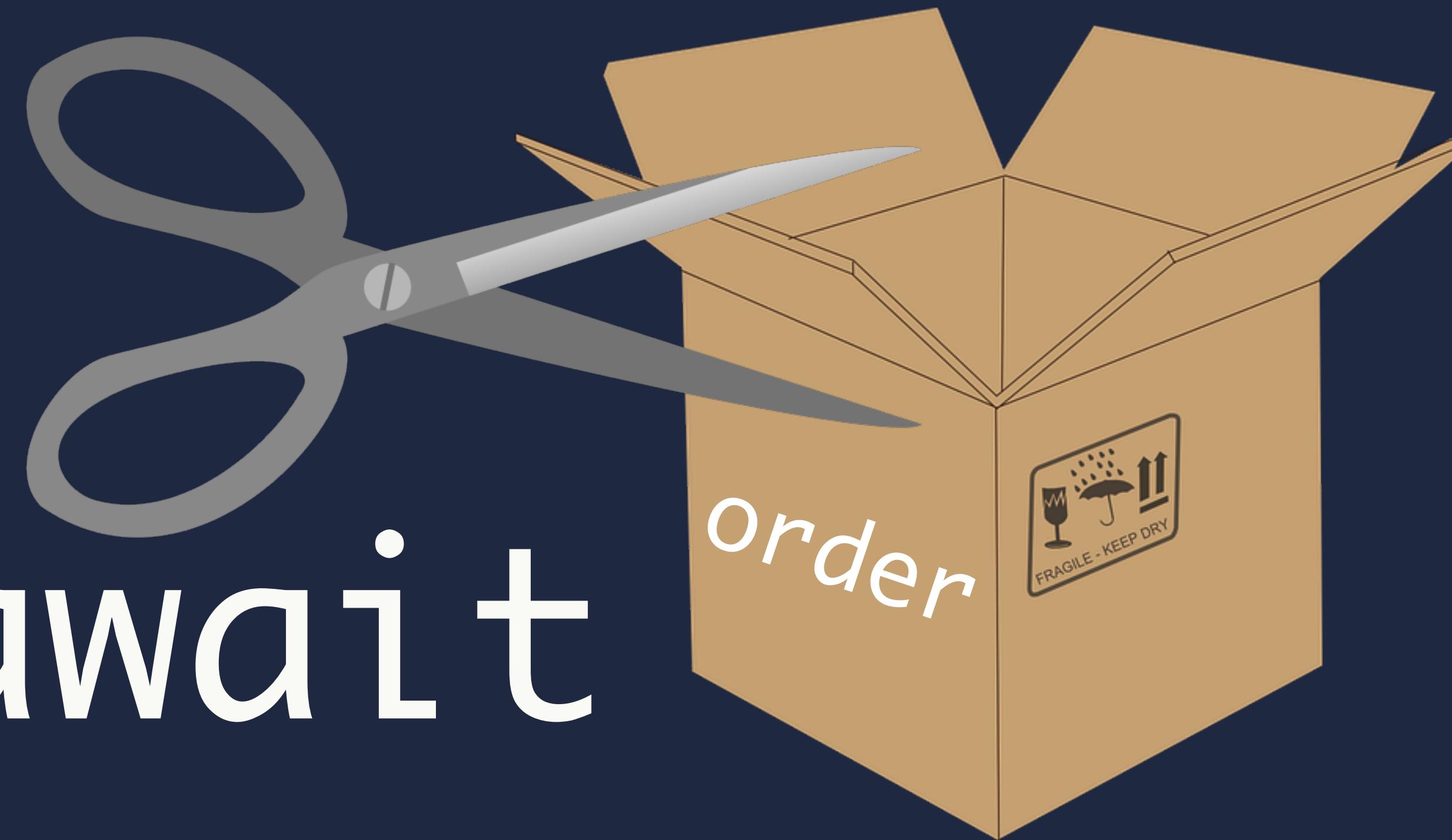
```
async printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

await

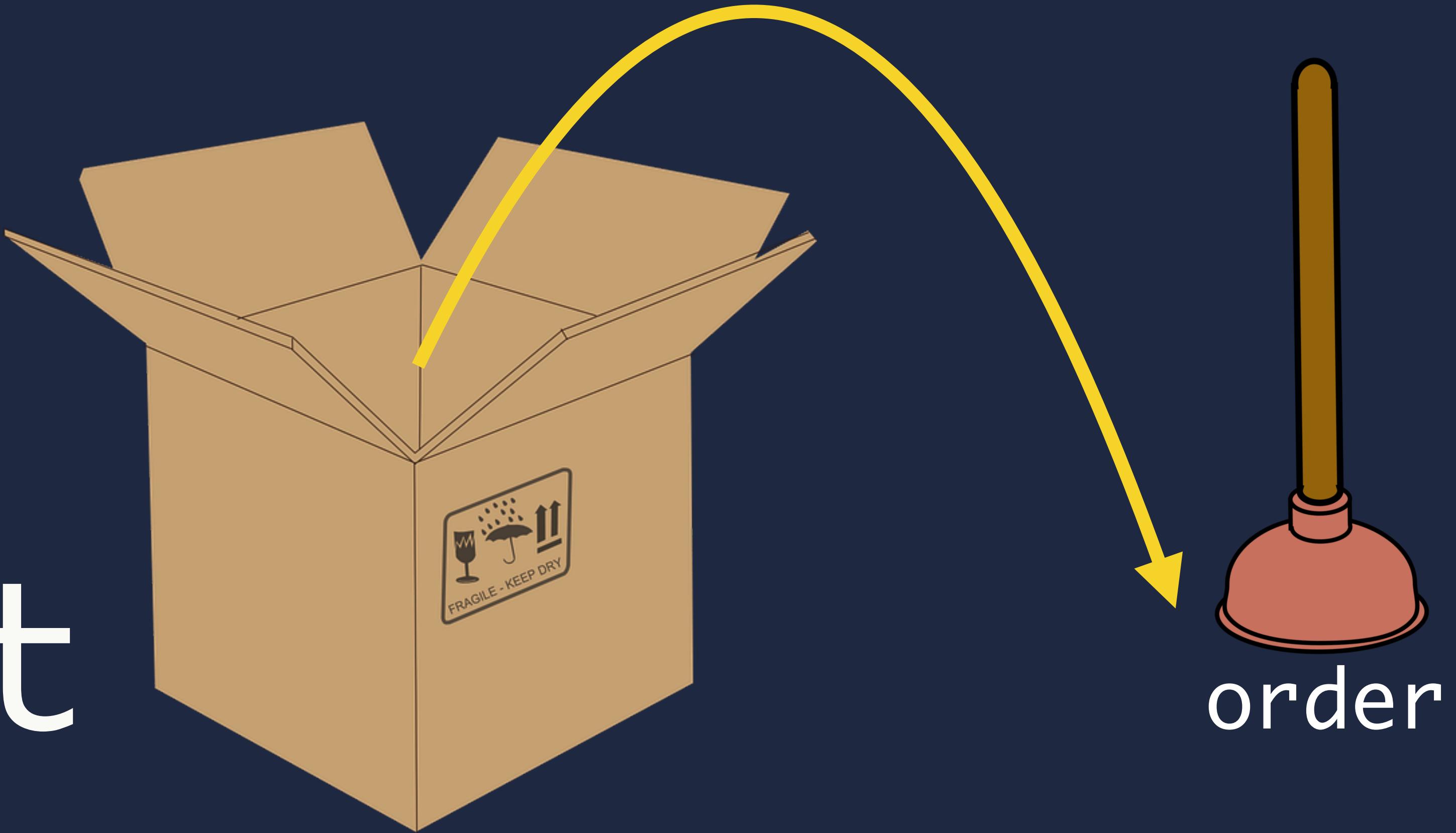


```
async printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

await



await



```
async printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```



```
async function printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

→ printOrder(1);

Invoke

```
→ async function printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

```
printOrder(1);
```

Encounter await

```
→ async function printOrder(orderId) {  
  const order = await fetchOrder(orderId);  
  console.log(order);  
}
```

```
printOrder(1);
```

Wrap in `Promise.resolve`

```
async function printOrder(orderId) {  
  → const promise = Promise.resolve(fetchOrder(orderId));  
  
  console.log(order);  
}  
  
printOrder(1);
```

Wrap in `Promise.resolve`

```
async function printOrder(orderId) {  
  const promise = Promise.resolve(fetchOrder(orderId));  
  
  → console.log(order);  
}  
  
printOrder(1);
```

Wrap rest with then callback

```
async function printOrder(orderId) {  
  const promise = Promise.resolve(fetchOrder(orderId));  
  
  → return promise.then(order => {  
    console.log(order);  
    return Promise.resolve(undefined);  
  });  
}  
  
printOrder(1);
```

Wrap rest with then callback

```
async function printOrder(orderId) {  
  const promise = Promise.resolve(fetchOrder(orderId));  
  
  return promise.then(order => {  
    console.log(order);  
    return Promise.resolve(undefined);  
  });  
}  
  
printOrder(1);
```

→
Wrap implicit return with
Promise.resolve

```
function fetchCustomerNameForOrder(orderId) {  
  return Promise.resolve(fetchOrder(orderId))  
    .then(order => {  
      return Promise.resolve(fetchCustomer(order.customerId))  
        .then(customer => {  
          return Promise.resolve(customer.name);  
        });  
    });  
});  
}
```



```
async function meaningOfLife() {  
    return 42;  
}
```

meaningOfLife() === 42; ???

```
async function meaningOfLife() {  
    return 42;  
}
```



```
function meaningOfLife() {  
    return Promise.resolve(42);  
}
```



```
meaningOfLife()  
.then(answer => answer === 42);
```

Exceptional Situations

```
async function printOrder(orderId) {  
  try {  
    const order = await fetchOrder(orderId);  
    console.log(order);  
  } catch (e) {  
    console.log('Error retrieving order', e);  
  }  
}
```

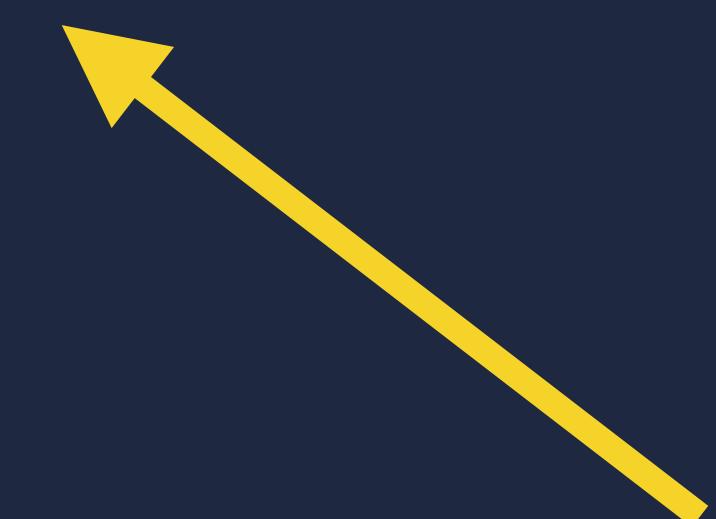
```
async function printOrder(orderId) {  
  try {  
    const order = await fetchOrder(orderId);  
    console.log(order);  
  } catch (e) {  
    console.log('Error retrieving order', e);  
  }  
}
```



```
function fetchOrder(orderId) {  
  return fetch(`/orders/${orderId}`)  
    .then(resp => {  
      if (resp.status === 404) {  
        throw new Error('Order not found');  
      }  
      return resp.json();  
    });  
}
```



```
function fetchOrder(orderId) {  
  return new Promise((resolve, reject) => {  
    $.get(`/orders/${orderId}`)  
      .done(order => resolve(order))  
      .fail(resp => {  
        if (resp.status === 404) {  
          reject(new Error('Order not found'));  
        }  
      });  
  });  
}
```

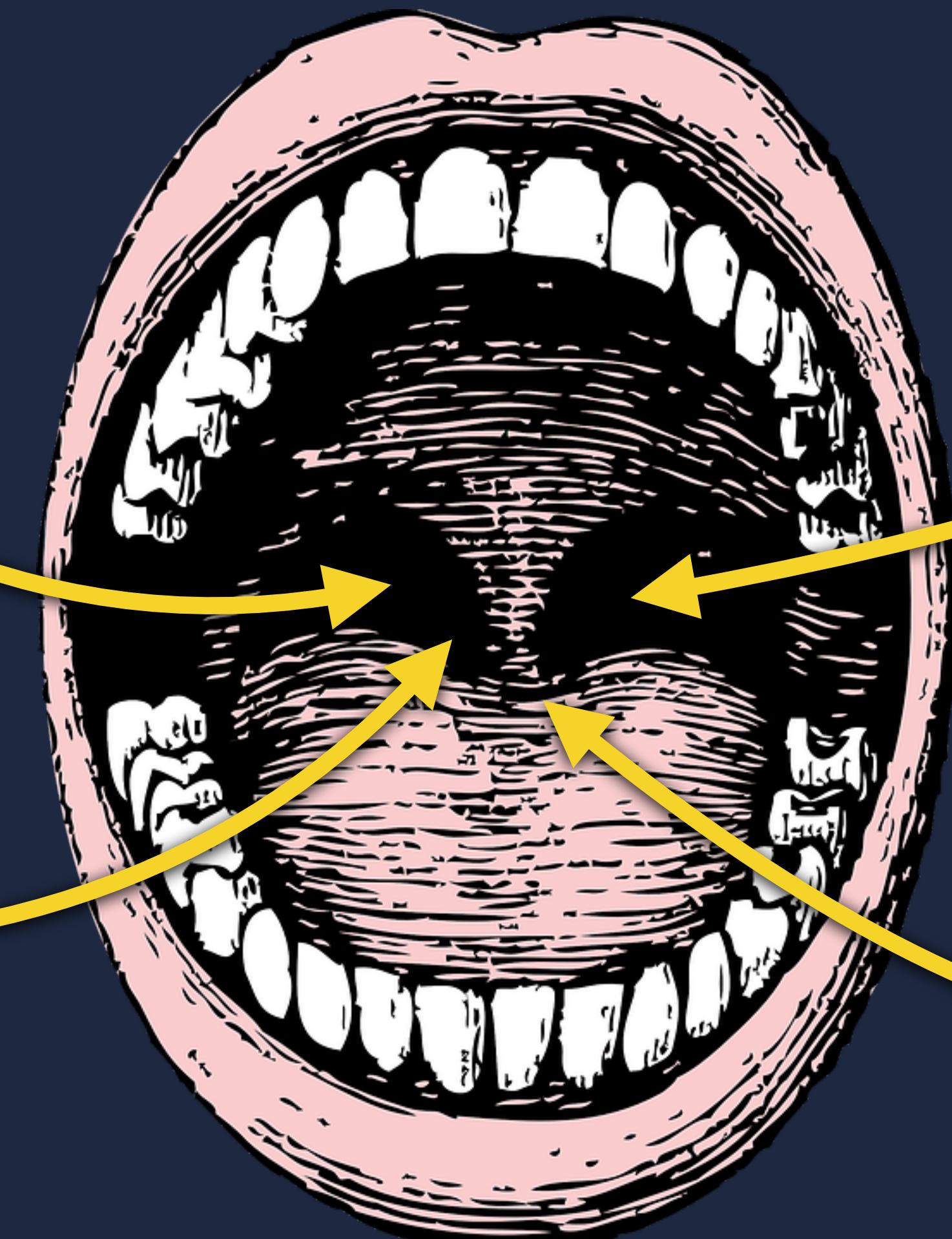


Error

Error

Error

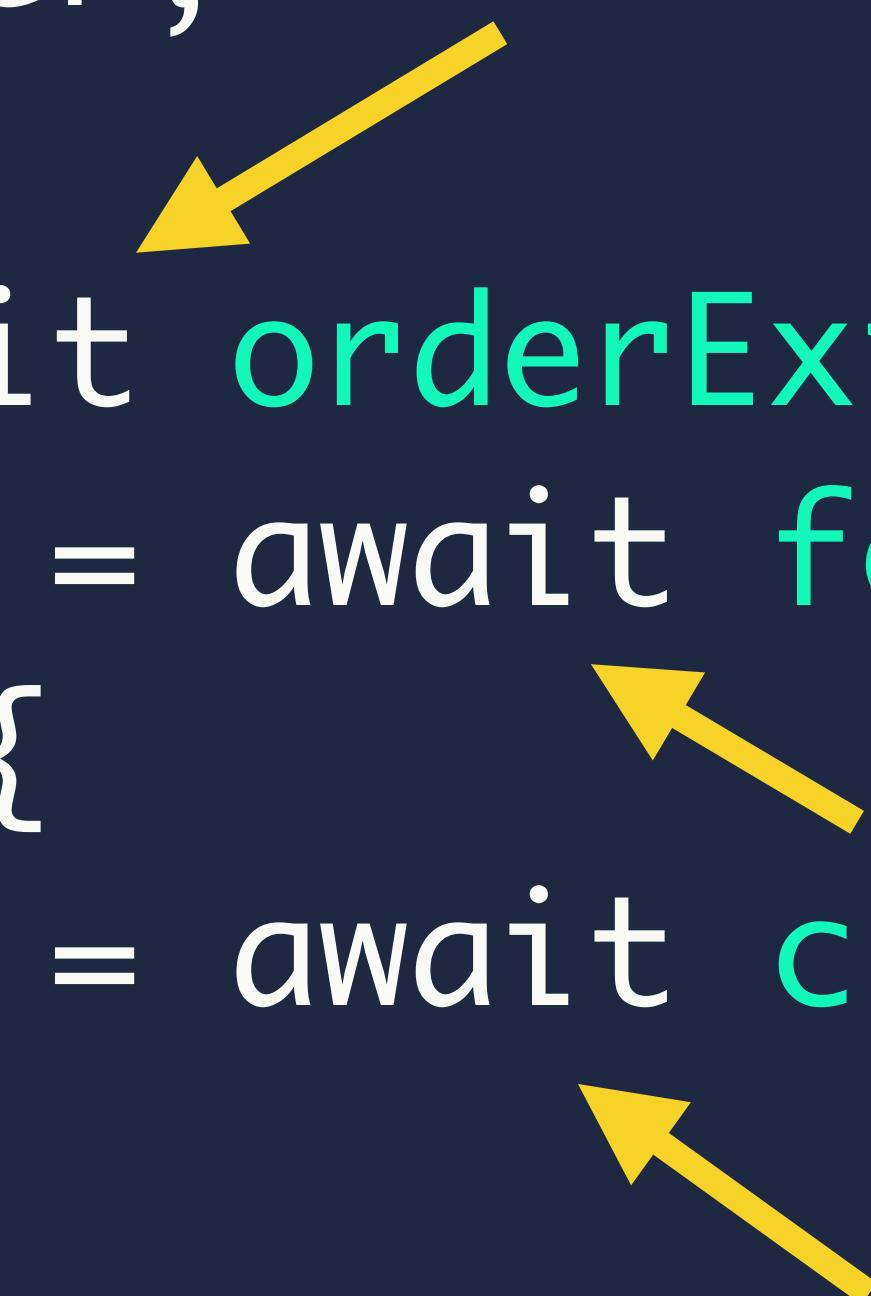
Error



Regain Control

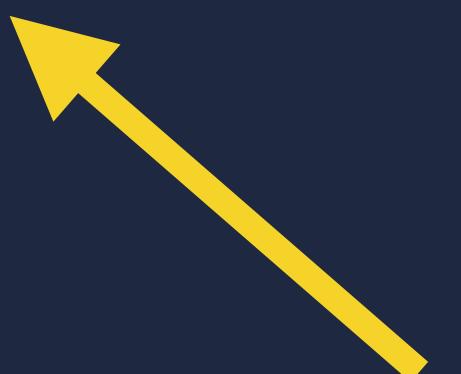
```
async function findOrCreateOrder(orderId) {  
  let order;  
  
  if (await orderExists(orderId)) {  
    order = await fetchOrder(orderId);  
  } else {  
    order = await createOrder();  
  }  
  
  return order;  
}
```

```
async function findOrCreateOrder(orderId) {  
  let order;  
  
  if (await orderExists(orderId)) {  
    order = await fetchOrder(orderId);  
  } else {  
    order = await createOrder();  
  }  
  
  return order;  
}
```

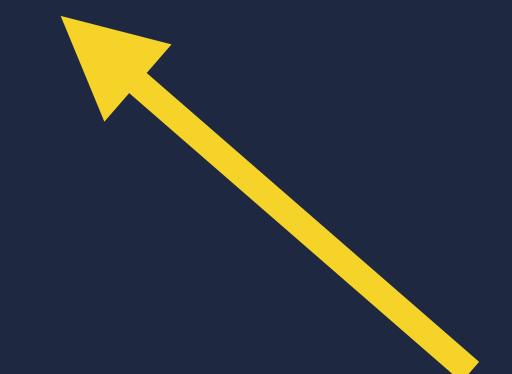


```
async function printOrders(orderIds) {  
  for (const id of orderIds) {  
    const order = await fetchOrder(id);  
    console.log(order);  
  }  
}
```

```
async function printOrders(orderIds) {  
  for (const id of orderIds) {  
    const order = await fetchOrder(id);  
    console.log(order);  
  }  
}  
}
```



The variable `order` is the value returned by the `fetchOrder` function.

```
async function printOrders(orderIds) {  
  for (const id of orderIds) {  
    const order = await fetchOrder(id);  
    console.log(order);   
  }  
}  
}
```

Problem?

```
async function printOrders(orderIds) {  
  for (const id of orderIds) {  
    const order = await fetchOrder(id);  
    console.log(order);  
  }  
}  
printOrders([1, 2, 3]);
```

```
printOrders([1, 2, 3]);
```



```
→ await fetchOrder(1);  
await fetchOrder(2);   
await fetchOrder(3); 
```

```
printOrders([1, 2, 3]);
```



```
→ await fetchOrder(1);  
      await fetchOrder(2);  
      await fetchOrder(3); 
```

```
printOrders([1, 2, 3]);
```



```
await fetchOrder(1);  
await fetchOrder(2);  
→ await fetchOrder(3);
```

```
async function printOrders(orderIds) {  
  const orders = await Promise.all(  
    orderIds.map(fetchOrder)  
  );  
  
  orders.forEach(order => console.log(order));  
}
```

```
async function printOrders(orderIds) {  
  const orders = await Promise.all(  
    orderIds.map(fetchOrder)  
);
```

```
  orders.forEach(order => console.log(order));  
}
```



```
printOrders([1, 2, 3]);
```



```
await Promise.all([
    fetchOrder(1),
    fetchOrder(2),
    fetchOrder(3)
]);
```



```
printOrders([1, 2, 3]);
```

Demo

rise.of.async.jeremyfairbank.com

BABE

```
$ npm install --save-dev \
  babel-core \
  babel-cli \
  babel-preset-es2015 \
  babel-plugin-syntax-async-functions \
  babel-plugin-transform-regenerator \
  babel-plugin-transform-runtime
```

.babelrc

```
{  
  "presets": ["es2015"],  
  "plugins": [  
    "syntax-async-functions",  
    "transform-regenerator",  
    "transform-runtime"  
  ]  
}
```

```
$ node_modules/.bin/babel-node myAsyncFile.js
```

Resources

- tc39.github.io/ecmascript-asyncawait/
- github.com/tc39/ecmascript-asyncawait
- github.com/tc39/ecma262

THANKS!

CODE:

github.com/jfairbank/rise-of-async-js-talk

SLIDES:

speakerdeck.com/jfairbank/fluent-conf-rise-of-async-javascript

Jeremy Fairbank

blog.jeremyfairbank.com

@elpapapollo | gh/jfairbank