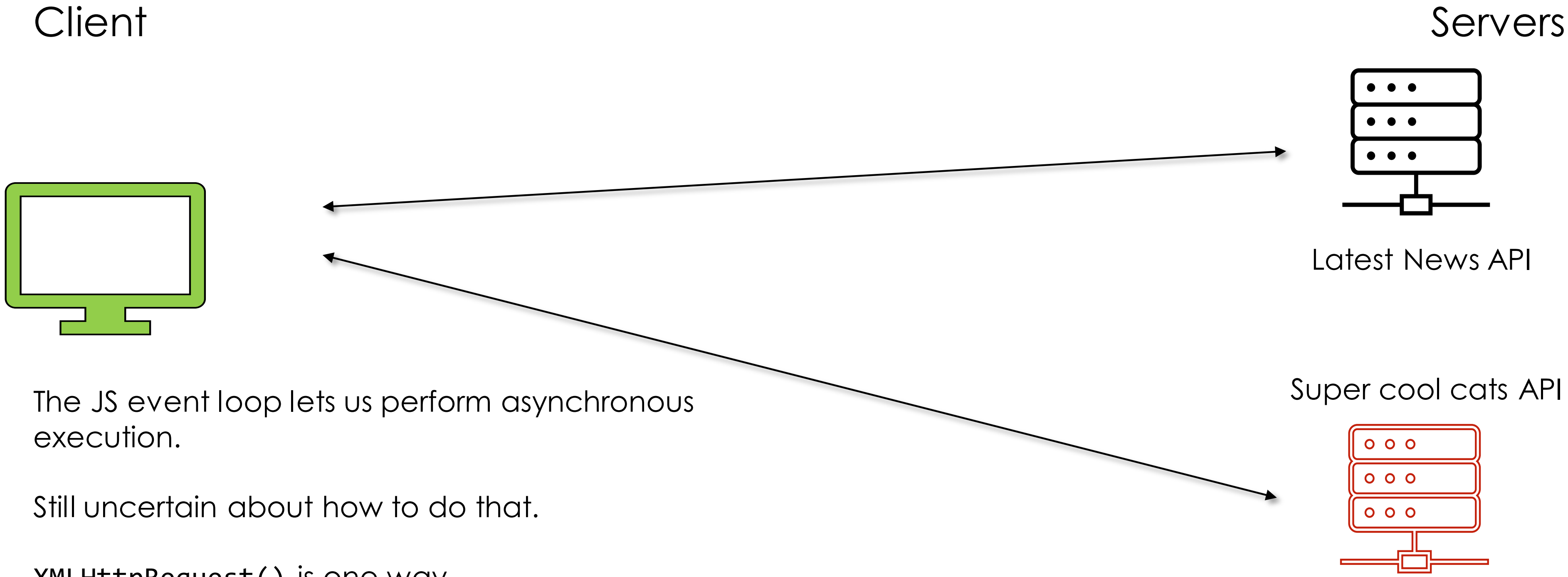
ASYNCHRONOUS NETWORKING

Networking with `XMLHttpRequest()`

OVERVIEW

- Client-Server Model + AJAX
- Concurrency & JS
- Networking with `XMLHttpRequest()`
- Networking with Promises & `fetch()`
- Networking with `async/await & fetch()`

RECAP

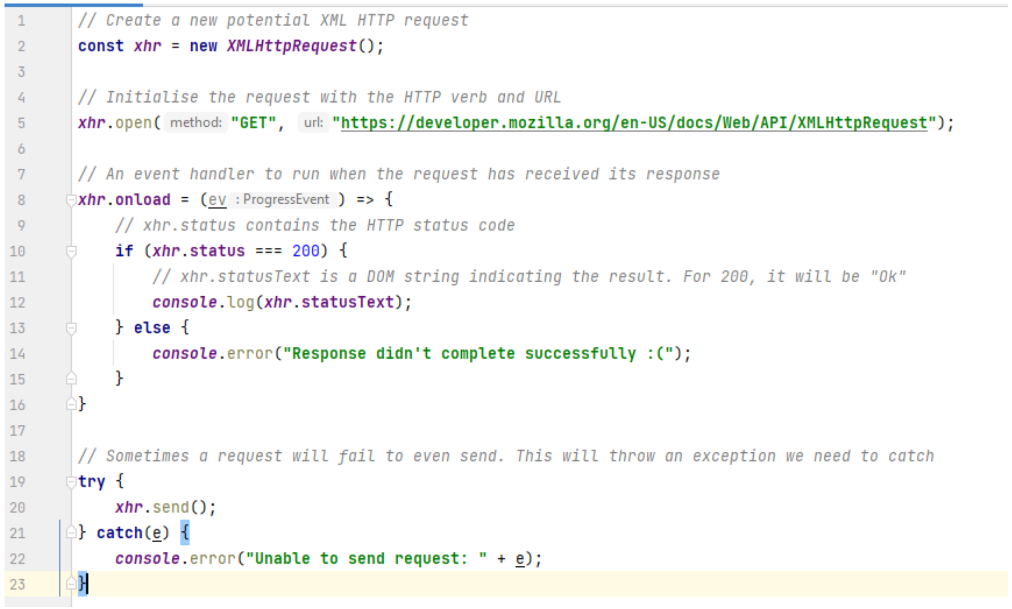


The JS event loop lets us perform asynchronous execution.

Still uncertain about how to do that.

`XMLHttpRequest()` is one way

XML HTTP



```
// Create a new potential XML HTTP request
const xhr = new XMLHttpRequest();

// Initialise the request with the HTTP verb and URL
xhr.open( method: "GET",  url: "https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest");

// An event handler to run when the request has received its response
xhr.onload = (ev :ProgressEvent ) => {
    // xhr.status contains the HTTP status code
    if (xhr.status === 200) {
        // xhr.statusText is a DOM string indicating the result. For 200, it will be "Ok"
        console.log(xhr.statusText);
    } else {
        console.error("Response didn't complete successfully :(");
    }
}

// Sometimes a request will fail to even send. This will throw an exception we need to catch
try {
    xhr.send();
} catch(e) {
    console.error("Unable to send request: " + e);
}
```

A minimal example

- Built-in API
  - Allows creation of a handle representing a request
  - Asynchronous by default
  - Many knobs and dials (see the docs for a full discussion)

- Error-Handling through exceptions and status codes
  - Network errors, failure-to-send errors are exceptions
  - Normal HTTP errors communicated through `xhr.status`

- Customisation via callbacks
  - Main callbacks: xhr`.onload, xhr.onerror`

- Fallen out of favour
  - Need to know for legacy codebases

BASIC XMLHTTPREQUEST DEMO

See examples/basic-xmlhttprequest

HANDLING PROBLEMS

Expected Errors

- Authentication errors
- Bad parameter errors
- Non-existent domain errors
- All checked through HTTP status codes
  - 4xx for not using an API correctly
  - 5xx for internal server errors
- HTTP 200 means no error

Unexpected Errors

- Network connectivity issues
- Remote connection suddenly dropped
- Timeouts
- Need to be handled by `try/catch` blocks
- Often the best strategy is to retry (with exponential back-off)

XMLHTTP ERROR-HANDLING DEMO

See examples/xmlhttp-error-handling

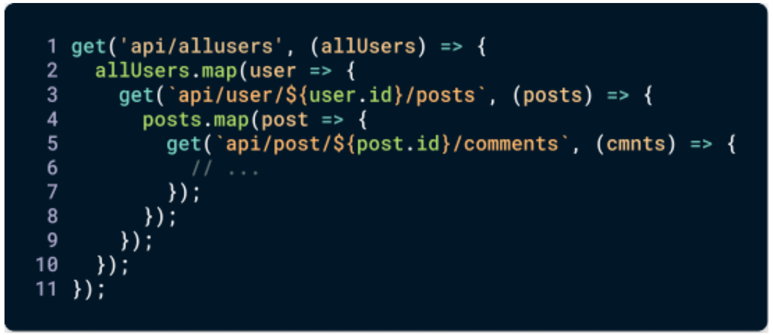
AVOIDING CALLBACK HELL

- Callback Hell: deep nesting of callbacks

- Often a result of:
  - Many different people working on code
  - Ad-hoc decision making

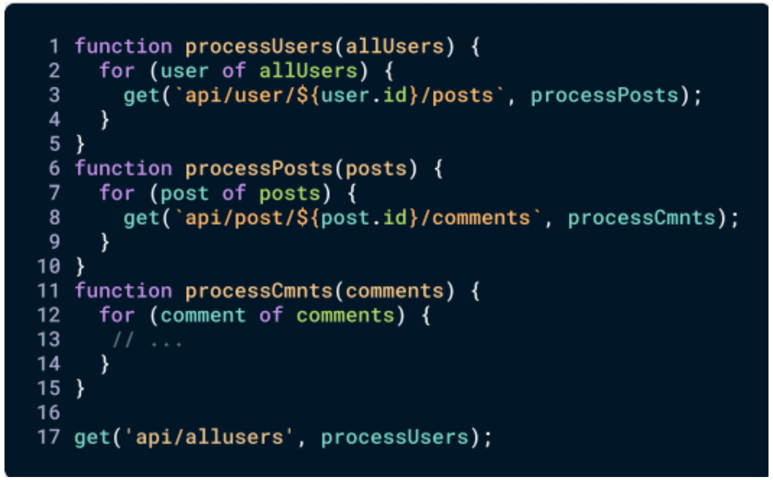
- Software Engineering Principles still apply:
  - SRP - functions do a single thing only
  - DRY – repetitive actions placed into a function
  - Let bundlers minimise your code for you

- Flat > Nested:
  - Callbacks should be named functions
  - Lambdas for easy one-liners



```
get('api/allusers', (allUsers) => {
  allUsers.map(user => {
    get(`api/user/${user.id}/posts`, (posts) => {
      posts.map(post => {
        get(`api/post/${post.id}/comments`, (cmnts) => {
          // ...
        });
      });
    });
  });
});
```

✗



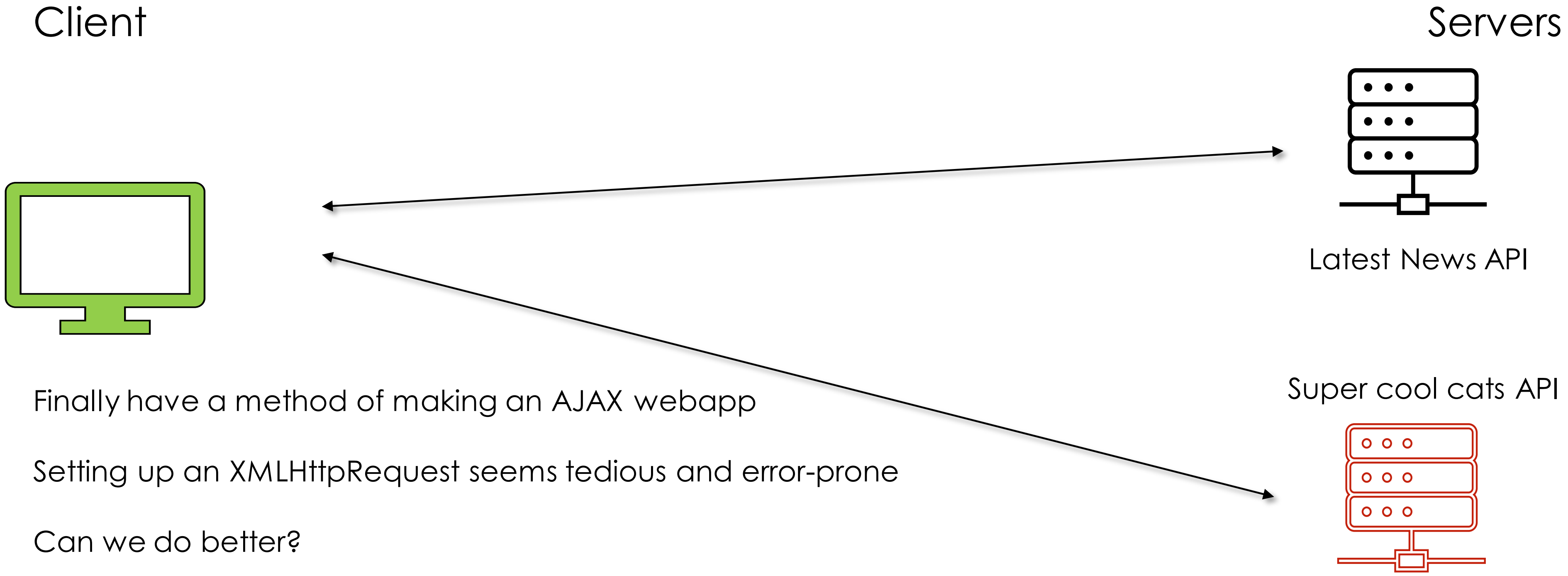
```
function processUsers(allUsers) {
  for (user of allUsers) {
    get(`api/user/${user.id}/posts`, processPosts);
  }
}
function processPosts(posts) {
  for (post of posts) {
    get(`api/post/${post.id}/comments`, processCmnts);
  }
}
function processCmnts(comments) {
  for (comment of comments) {
   // ...
  }
}

get('api/allusers', processUsers);
```

✓

Image Credits: Zain Afzal

IMPROVING



Finally have a method of making an AJAX webapp

Setting up an XMLHttpRequest seems tedious and error-prone

Can we do better?

SUMMARY

- Today:
  - Making asynchronous network requests with `XMLHttpRequest()`
  - Error-handling strategies
  - Avoiding callback hell
- Coming Up Next:
  - Promises
  - Networking with `fetch()`