Asynchronous JS

SWE 432, Fall 2018

Web Application Development

Review: Asynchronous

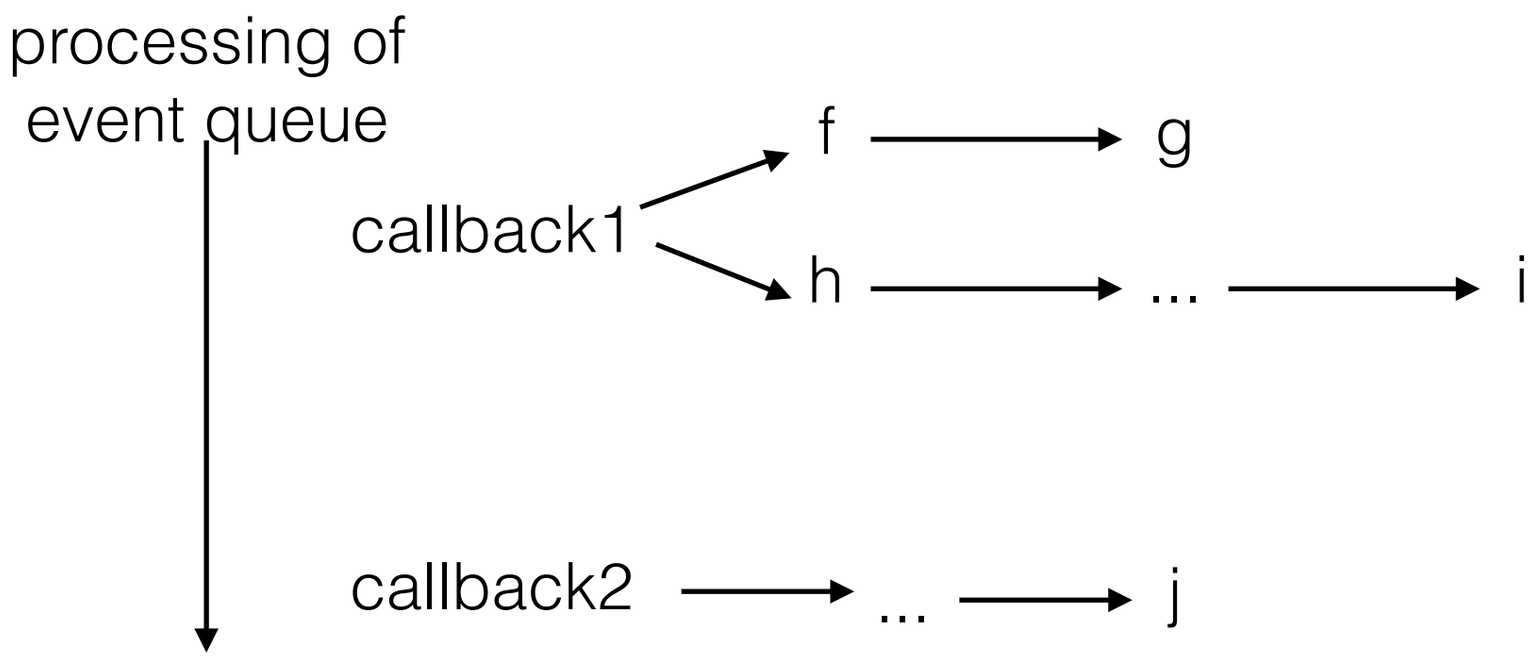
- Synchronous:
  - Make a function call
  - When function call returns, the work is done
- Asynchronous:
  - Make a function call
  - Function returns immediately, before completing work!

Review: Asynchronous

- How we do multiple things at a time in JS
- NodeJS magically handles these asynchronous things in the background
- Really important when doing file/network input/output

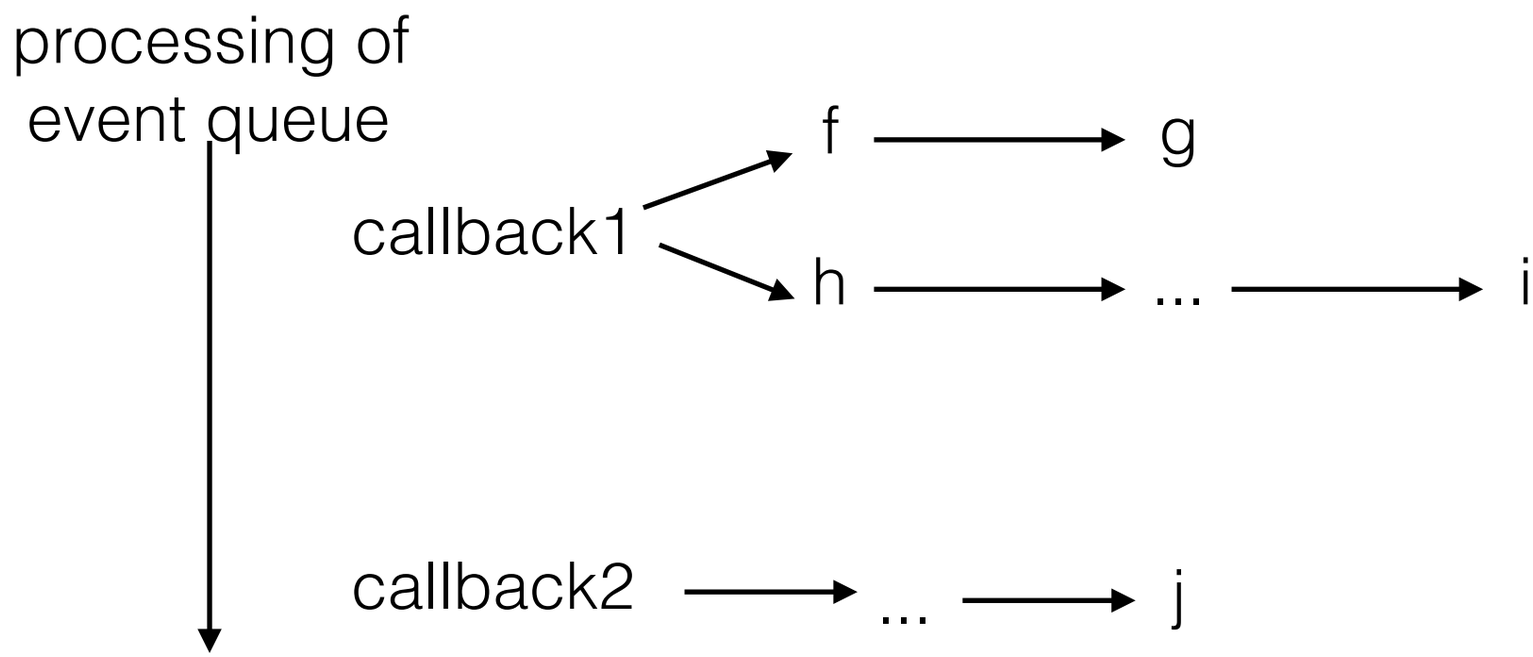
Review: Run-to-completion semantics

- Run-to-completion
  - The function handling an event and the functions that it (transitively) synchronously calls will keep executing until the function finishes.
  - The JS engine will not handle the next event until the event handler finishes.



Review: Implications of run-to-completion

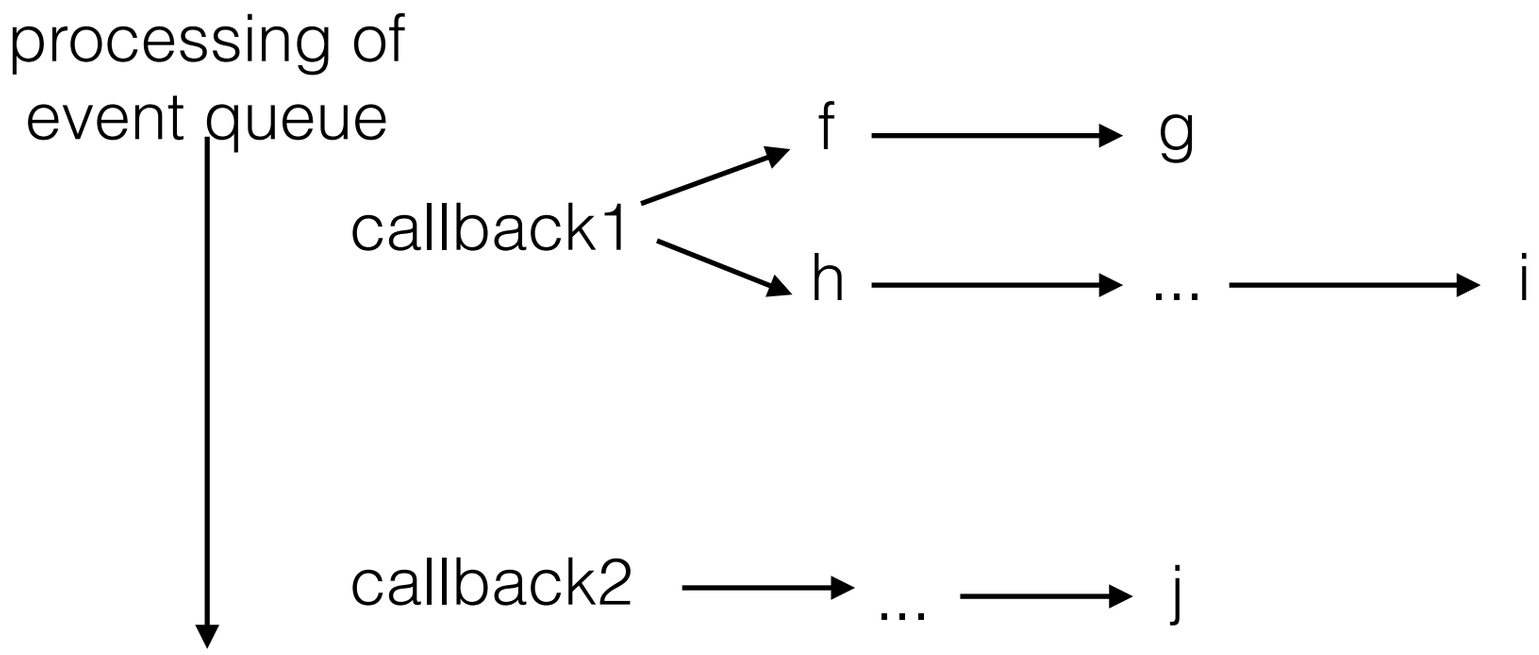
- Good news: no other code will run until you finish (no worries about other threads overwriting your data)



j will not execute until after i

Review: Implications of run-to-completion

- Bad/OK news: Nothing else will happen until event handler returns
  - Event handlers should never block (e.g., wait for input) --> all callbacks waiting for network response or user input are **always** asynchronous
  - Event handlers shouldn't take a long time either



j will not execute until i finishes

Review: Promising many things

- Can also specify that *many* things should be done, and then something else

- Example: load a whole bunch of images at once:

```
Promise
  .all([loadImage("GMURGB.jpg"), loadImage(“JonBell.jpg")])
.then(function (imgArray) {
        imgArray.forEach(img => {document.body.appendChild(img)})
    })
.catch(function (e) {
  console.log("Oops");
  console.log(e);
});
```

Review: Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
.then(function(resultOfStep1){
    //Do something, maybe asynchronously
    return theResultOfThisStep
})
.then(function(resultOfStep2){
    //Do something, maybe asynchronously
    return theResultOfThisStep
})
.then(function(resultOfStep3){
    //Do something, maybe asynchronously
    return theResultOfThisStep
})
.catch(function(error){

});
```

HW1 Survey

Go to:
b.socrative.com, Click student login
Room name: SWE432
Student ID: Your G-number (Including the G)

Reminder: Survey can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

Today

- Asynchronous activity
- Async/await
- Programming activity

Async Programming Example

1 second each

Go get a candy bar | Go get a candy bar | Go get a candy bar | Go get a candy bar | Go get a candy bar

Go get a candy bar | Go get a candy bar | Go get a candy bar | Go get a candy bar | Go get a candy bar

thenCombine

2 seconds each

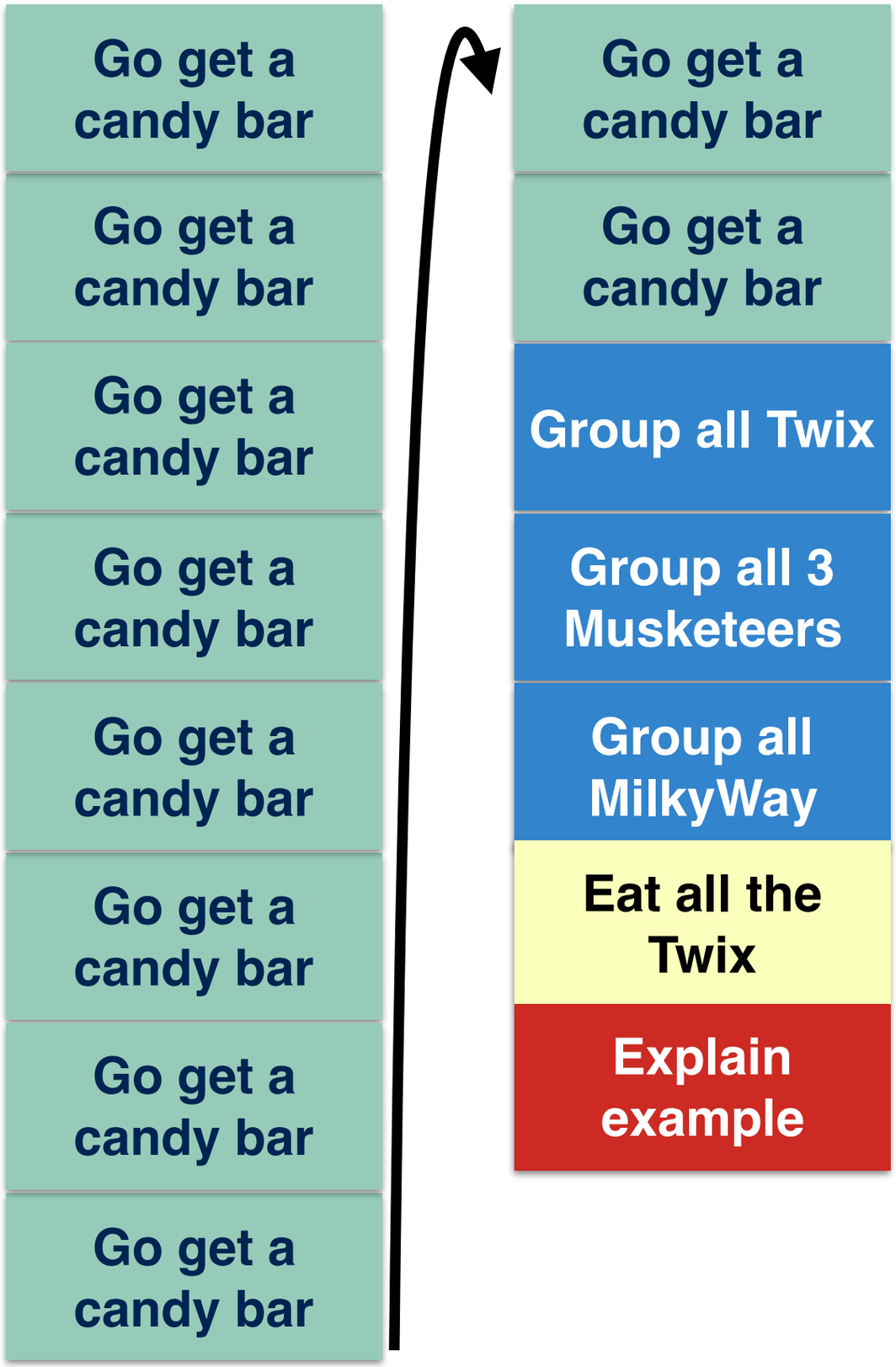
Group all Twix | Group all 3 Musketeers | Group all MilkyWay | Group all MilkyWay Dark | Group all Snickers

when done

Eat all the Twix

Explain example

Synchronous Version



Asynchronous Version

Go get a candy bar	Go get a candy bar	Go get a candy bar	Go get a candy bar	Go get a candy bar
Go get a candy bar	Go get a candy bar	Go get a candy bar	Go get a candy bar	Go get a candy bar

...

Group all Twix	Group all 3 Musketeers	Group all MilkyWay	Group all MilkyWay Dark	Group all Snickers

Note ordering constraint: *Explain example* happens after we start getting candy bars, independent of what is happening with candy bars

Eat all the Twix

Explain example

Async Programming Example (Sync)

```
let lib = require("./lib.js");

let thingsToFetch = ['t1','t2','t3','s1','s2','s3','m1','m2','m3','t4'];
let stuff = [];
for(let thingToGet of thingsToFetch)
{
    stuff.push(lib.getSync(thingToGet));
    console.log("Got a thing");
}
//Got all my stuff
let ts = lib.groupSync(stuff,"t");
console.log("Grouped");
let ms = lib.groupSync(stuff,"m");
console.log("Grouped");
let ss = lib.groupSync(stuff,"s");
console.log("Grouped");

console.log("Done");
```

Async Programming Example (Callbacks, no parallelism)

```
let lib = require("./lib.js");

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;

lib.getASync(thingsToFetch[0],(v)=>{
    stuff.push(v);
    console.log("Got a thing")
    lib.getASync(thingsToFetch[1],(v)=>{
        stuff.push(v);
        console.log("Got a thing")
lib.getASync(thingsToFetch[2],(v)=>{
    stuff.push(v);
    console.log("Got a thing")
    lib.getASync(thingsToFetch[3],(v)=>{
        stuff.push(v);
        console.log("Got a thing")
        lib.getASync(thingsToFetch[4],(v)=>{
            stuff.push(v);
            console.log("Got a thing")
            lib.getASync(thingsToFetch[5],(v)=>{
                stuff.push(v);
                console.log("Got a thing")
                lib.getASync(thingsToFetch[6],(v)=>{
                    stuff.push(v);
                    console.log("Got a thing")
                    lib.getASync(thingsToFetch[7],(v)=>{
                        stuff.push(v);
                        console.log("Got a thing")
                        lib.getASync(thingsToFetch[8],(v)=>{
                            stuff.push(v);
                            console.log("Got a thing")
                            lib.getASync(thingsToFetch[9],(v)=>{
                                stuff.push(v);
                                console.log("Got a thing")
                                lib.groupAsync(stuff, "t", (t) => {
                                    ts = t;
                                    console.log("Grouped");
                                    lib.groupAsync(stuff, "m", (m) => {
```

Async Programming Example (Callbacks)

```
let lib = require("./lib.js");

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;
for (let thingToGet of thingsToFetch) {
    lib.getASync(thingToGet, (v) => {
        stuff.push(v);
        console.log("Got a thing")
        outstandingStuffToGet--;
        if (outstandingStuffToGet == 0) {
            let groupsOfStuffTogetStill = 3;
            lib.groupAsync(stuff, "t", (t) => {
                ts = t;
                console.log("Grouped");
                groupsOfStuffTogetStill--;
                if (groupsOfStuffTogetStill == 0)
                    console.log("Done");

            });
            lib.groupAsync(stuff, "m", (m) => {
                ms = m;
                console.log("Grouped");
                groupsOfStuffTogetStill--;
                if (groupsOfStuffTogetStill == 0)
                    console.log("Done");
            });
            lib.groupAsync(stuff, "s", (s) => {
                ss = s;
                console.log("Grouped");
                groupsOfStuffTogetStill--;
                if (groupsOfStuffTogetStill == 0)
                    console.log("Done");
            })
        }
    });
}
```

Async Programming Example (Promises, no parallelism)

```
let lib = require("./lib.js");

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;
lib.getPromise(thingsToFetch[0]).then(
    (v)=>{
        stuff.push(v);
        console.log("Got a thing");
        return lib.getPromise(thingsToFetch[1]);
    }
).then(
     (v)=>{
        stuff.push(v);
        console.log("Got a thing");
        return lib.getPromise(thingsToFetch[1]);
    }
).then(
    (v)=>{
        stuff.push(v);
        console.log("Got a thing");
        return lib.getPromise(thingsToFetch[1]);
    }
).then(
    (v)=>{
        stuff.push(v);
        console.log("Got a thing");
        return lib.getPromise(thingsToFetch[2]);
    }
).then(
    (v)=>{
        stuff.push(v);
        console.log("Got a thing");
        return lib.getPromise(thingsToFetch[3]);
    }
).then(
    (v)=>{
        stuff.push(v);
```

Async Programming Example (Promises)

```
let lib = require("./lib.js");

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;

let promises = [];
for (let thingToGet of thingsToFetch) {
    promises.push(lib.getPromise(thingToGet));
}
Promise.all(promises).then((data) => {
    console.log("Got all things");
    stuff = data;
    return Promise.all([
            lib.groupPromise(stuff, "t"),
            lib.groupPromise(stuff, "m"),
            lib.groupPromise(stuff, "s")
        ]
    )
}).then((groups) => {
    console.log("Got all groups");
    ts = groups[0];
    ms = groups[1];
    ss = groups[2];
    console.log("Done");
});
```

Problems with Promises

```
const makeRequest = () => {
    return promise1()
        .then(value1 => {
            // do something
            return promise2(value1)
                .then(value2 => {
                    // do something
                    return promise3(value1, value2)
                })
        })
}
```

Promise 3 requires the results of Promise 1, 2 -> need this nested “callback hell”

Problems with Promises

```
const makeRequest = () => {
    try {
        return promise1()
            .then(value1 => {
                // do something
            }).catch(err => {
                //This is the only way to catch async errors
                console.log(err);
            })
    }catch(ex){
        //Will never catch async errors!!
    }
}
```

Async/Await

- The latest and greatest way to work with async functions
- A programming pattern that tries to make async code look more synchronous
- Just “await” something to happen before proceeding
- https://javascript.info/async-await

Async/Await -> Synchronous

```
let lib = require("./lib.js");

async function getAndGroupStuff() {
    let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
    let stuff = [];
    let ts, ms, ss;

    let promises = [];
    for (let thingToGet of thingsToFetch) {
        stuff.push(await lib.getPromise(thingToGet));
        console.log("Got a thing");
    }
    ts = await lib.groupPromise(stuff,"t");
    console.log("Made a group");
    ms = await lib.groupPromise(stuff,"m");
    console.log("Made a group");
    ss = await lib.groupPromise(stuff,"s");
    console.log("Made a group");
    console.log("Done");
}

getAndGroupStuff();
```

Async/Await

- Rules of the road:
  - You can only call **await** from a function that is **async**
  - You can only **await** on functions that return a **Promise**

- Beware: await makes your code synchronous!

```
async function getAndGroupStuff() {
...
    ts = await lib.groupPromise(stuff,"t");
...
}
```

Async/Await Activity

Rewrite this code so that all of the things are fetched (in parallel) and then all of the groups are collected

```
let lib = require("./lib.js");

async function getAndGroupStuff() {
    let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
    let stuff = [];
    let ts, ms, ss;

    let promises = [];
    for (let thingToGet of thingsToFetch) {
        stuff.push(await lib.getPromise(thingToGet));
        console.log("Got a thing");
    }
    ts = await lib.groupPromise(stuff,"t");
    console.log("Made a group");
    ms = await lib.groupPromise(stuff,"m");
    console.log("Made a group");
    ss = await lib.groupPromise(stuff,"s");
    console.log("Made a group");
    console.log("Done");
}

getAndGroupStuff();
```

download lib.js: https://bit.ly/2QvyrOu

download this code: https://bit.ly/2OvsWhq

Async/Await

```
async function getAndGroupStuff() {
    let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
    let stuff = [];
    let ts, ms, ss;

    let promises = [];
    for (let thingToGet of thingsToFetch) {
        promises.push(lib.getPromise(thingToGet));
    }
    stuff = await Promise.all(promises);
    console.log("Got all things");
    [ts, ms, ss] = await Promise.all([lib.groupPromise(stuff, "t"),
lib.groupPromise(stuff, "m"), lib.groupPromise(stuff, "s")]);
    console.log("Got all groups");
    console.log("Done");
}

getAndGroupStuff();
```