# FLOW CONTROL WITH ASYNC.JS

# HOW IS FLOW CONTROL DIFFERENT IN NODE?

# BLOCKING/THREAD BASED

```
Database db = new Database();

try {
  db.connect('user@host:1234/mydb');
} catch (Error e) {
  System.err.println("Could not connect to db");
  e.printStackTrace();
}

try {
  Result results = db.query('SELECT COUNT(*) FROM users');
} catch (Error e) {
  System.err.println("Query failed to run");
  e.printStackTrace();
}

// Operate on results
```

# FLOW CONTROL IN NODE

```
var db = new Database();

db.connect('user@host:1234/mydb', function(err, result) {
  if (err) {
    // handle error
  }

  // no error, handle result
});
```

# THE WRONG WAY

```
var db = new Database();

var a = db.connect('user@host:1234/mydb', function(err, result) {
  if (err) {
    // handle error
    return err;
  }

  // no error, handle result
  return result;
});

console.log(a); // undefined
```

# A BETTER WAY

```
var db = new Database();

db.connect('user@host:1234/mydb', function(err, conn) {
  if (err) {
    // handle error
    return errorHandler(err);
  }

  // no error, handle result
  connectionHandler(conn);
});
```

# TYPES OF FLOW

- serial
- parallel

# SERIAL EXECUTION

- execute each function in series
- the results of the current function may be the inputs to the next
- callback when complete

# AN ASYNCHRONOUS FUNCTION

```
countUsers('user@host.tld:1234', function(err, result) {
  if (err) {
    console.log('Could not count users');
    console.log(err);
    return;
  }

  console.log(result);
});
```

# SERIAL EXECUTION: CALLBACKS

```
var db = require('database-manager');

function countUsers(dsn, fn) {

  db.connect(dsn, function(err, conn) {
    if (err) {
      return fn(err);
    }

    conn.use('mydb', function(err, mydb) {
      if (err) {
        return fn(err);
      }

      mydb.query('SELECT COUNT(*) FROM user', fn);
    });
  });
}
```

# SERIAL EXECUTION: ASYNC.JS

```
var db = require('database-manager');
var async = require('async');


function countUsers(dsn, fn) {

  async.waterfall([

    function connect(callback) {
      db.connect(dsn, callback);
    },

    function use(conn, callback) {
      conn.use('mydb', callback);
    },

    function query(mydb, callback) {
      mydb.query('SELECT COUNT(*) FROM user', callback)
    }

  ], fn);
}
```

# PARALLEL EXECUTION

- perform many async operations at once
- wait for all to complete
- callback

# PARALLEL EXECUTION: CALLBACKS

**01_parallel_execution.js:**

```
var fs = require('fs');
var path = require('path');
var files = ['a.json', 'b.json', 'c.json'];
var complete = 0;
var obj = {};

var done = function(err) {
  if (err) {
    throw err;
  }
  console.log(obj);
};

files.forEach(function(filename) {
  console.log('begin read of', filename);
  fs.readFile(path.join(__dirname, 'support', filename), function(err, j
    if (err) {
      return done(err);
    }

    obj[filename] = JSON.parse(json);

    complete++;
    console.log(Math.floor((complete / files.length)*100) + '% complete'
    if (complete >= files.length) {
      done();
    }
  });
});
```

# PARALLEL EXECUTION WITH ASYNC.JS

**02_parallel_execution_async.js:**

```
var fs = require('fs');
var async = require('async');
var path = require('path');
var files = ['a.json', 'b.json', 'c.json'];

var obj = {};

async.each(files, function(filename, callback) {
  console.log('begin read of', filename);
  fs.readFile(path.join(__dirname, 'support', filename), function(err, j
    if (err) {
      return callback(err);
    }
    obj[filename] = JSON.parse(json);

    console.log(Math.floor((Object.keys(obj).length / files.length)*100)

    callback();
  });

}, function done(err) {
  if (err) {
    throw err;
  }
  console.log(obj);
});
```

# WHEN TO USE ASYNC VS NESTED CALLBACKS

# BE NICE TO YOUR TEAMATES

and future you

```
perform(function(err, result1) {
  perform2(result1, function(err, result2) {
    perform3(result2, function(err, result3) {
      perform4(result3, function(err, result4) {
        perform5(result4, function(err, result5) {
          perform6(result5, function(err, result6) {
            perform7(result6, function(err, result7) {
              // Handle results and call callback
            });
          });
        });
      });
    });
  });
});
```

# CALLBACKS VS EVENT EMITTERS VS STREAMS

- `callback`s should be called once
- `EventEmitter`s are for state transfer
- `stream`s are for state or data transer

# ASYNC.JS FEATURES

- collection iterators
- flow control
- some basic utility functions

# ASYNC.JS COLLECTION ITERATORS

Peform parallel async operations on arrays

- `each` - iterate over an array; no result
- `map` - iterate over an array; array result
- `filter` - filter an array
- `reduce` - reduces an array into a single value
- `detect` - finds the first occurrance
- and more: https://github.com/caolan/async#collections

# ASYNC.JS FLOW CONTROL

Control the execution timing of functions

- `waterfall` - run functions in series, final result is the result of the last function
- `series` - run functions in series, final result is an object or array
- `parallel` - run functions concurrently, final result is an object or array
- `queue` - creates a worker queue with the specified number of concurrent workers
- and more: https://github.com/caolan/async#control-flow]

# USING ASYNC AND THE GITHUB API

# GENERATE A GITHUB API TOKEN

```
curl https://api.github.com/authorizations --user "<your username>"
```

```
$ curl https://api.github.com/authorizations --user "<your username>" --
Enter host password for user '<your username>':
{
  ...

  "token": "21cb501a85e2fd3faxxxxxxxxxxxxxxxxxxxxxxx",
  ...
}
```

# COLLECT A REPO CONTRIBUTORS

## 03_github_contributors.js

```
var async = require('async'),
    request = require('request');

if (process.argv.length < 5) {
  return console.log('Usage: node 03_github_watchers.js <username> <toke
}

var url = 'https://api.github.com/';
var auth = {
  headers: {'User-Agent' : 'nodefirm' },
  auth: { user: process.argv[2],  pass: process.argv[3] },
  json: true
};

request.get(url + 'repos/' + process.argv[4] + '/contributors', auth, fu
  if (err) throw err;
  async.map(array, function(contributor, callback) {
    request.get(url + 'users/' + contributor.login + '/keys', auth, func
      if (err) throw err;

      callback(null, [contributor.login, (array.length) ? array[0].key :
    });
  }, function(err, results) {
    console.log(results);
  });
});
```

# SUMMARY

- choose the right tool for the job
- Async.js
  - manages callbacks
  - decreases boilerplate
  - improves maintenance and readability
  - unifies your team's approach