Doing things in parallel (async.parallel)

One of the benefits of using node is the ability to run multiple I/O tasks concurrently. Lets see how we can achieve that using callbacks or promises.

Callbacks

The most popular node module to run callback-based functions in parallel is caolan's async

```
function readTwoFiles(file1, file2, callback) {
    async.parallel([
        fs.readFile.bind(fs, file1),
        fs.readFile.bind(fs, file2),
    ], callback);
}
readTwoFiles(file1, file2, function(err, files) {
    console.log(files[0], files[1]);
})
```

The callback is called with no error and an array of results after all operations are complete, or when the first error is encountered.

Promises

Bluebird and Q give you `.all()`, a method that creates a new promise from an array of promises:

```
function readTwoFiles(file1, file2) {
    return Bluebird.all([fs.readFileAsync(file1),
        fs.readFileAsync(file2)]);
```

```
}
readTwoFiles(file1, file2).then(function(files) {
    console.log(files[0], files[1]);
})
```

The resulting promise is fulfilled when all the promises in the array are fulfilled or is rejected with an error when the first error is encountered.

Notes

async.parallel expects functions that take a single callback argument. `Function.bind()` allows us to create such functions by binding some of the arguments with predefined values. Therefore

```
fs.readFile.bind(fs, file1)
```

returns a function that works like this:

```
function(callback) { fs.readFile(file1, callback); }
```