Fork me on GitHub

Concurrency level (async.mapLimit)

In the previous example we ran the same async task on multiple items in series. Each task waited for the previous task to complete, then downloaded an item then extracted its name.

Sometimes we might want to run a limited number of tasks in parallel. In that case we need to use something else.

Callbacks

With callbacks we need yet another utility function - async.mapLimit

```
async.mapLimit(ids, 3, function(id, callback) {
    getFromStorage(id, function (err, res) {
        if (err) return callback(err);
        callback(null, res.name);
    })
}, function(err, results) {
    // results is an array of names
});
```

Promises

With promises we finally need a new helper function - `Promise.some` which is provided to us by bluebird

```
var queued = [], parallel = 3;
var namePromises = ids.map(function(id) {
    // How many items must download before fetching the next?
    // The queued, minus those running in parallel, plus one of
    // the parallel slots.
    var mustComplete = Math.max(0, queued.length - parallel + 1);
    // when enough items are complete, queue another request for an item
    var download = Promise.some(queued, mustComplete)
        .then(function() { return getItem(id); });
    queued.push(download);
    return download.then(function(item) {
        // after that new download completes, get the item's name.
        return item.name;
    });

  });
Promise.all(namePromises).then(function(names) {
    // use all names here.
});
```

Even though the code is a bit longer, its quite clear: When enough of the queued items are complete, queue another download, wait for it to complete then extract the item's name.