The first relief - when sync becomes async

I hear you. Promise code doesn't look too different from regular node callbacks or event emitters, except that you use a second callback for the error. And you're still not convinced that there is any benefit to using `.then()` to create new promises.

Okay then. Lets take things to the next level.

Using `.then()`, you can attach the callbacks later if you want.

For example `fs.readFileAsync(file)` returns a promise. That promise is a value, so you can put that in a var, or return it from a function:

```
var filePromise = fs.readFile(file);
filePromise.done(function(res) { ... }, function(err) {});
```

Okay, that's still not much of an improvement. How about this then? You can attach more than one callback to a promise if you like:

```
filePromise.done(function(res) { uploadData(url, res); });
filePromise.done(function(res) { saveLocal(url, res); },
    function(err), {});
// etc
```

Hey, this is beginning to look more and more like streams - they too can be piped to multiple destinations. But unlike streams, you can attach more callbacks and get the value even *after* the file reading operation completes.

The promise will cache the value, and call your callback right after the next tick. The file reading operation is always done only once - no need to repeat it.

All of this makes it easier when you need to switch code that was previously sync but had to become async for some reason.

For example: imagine that you need to get some stats about two file versions

- like number of lines in both versions and the number of lines in their diff:

```
var first = files[0], other = files[1];
var diffFirstOther = diff(first, other);

var firstLines = first.split('\n').length,
    otherLines = other.split('\n').length;

var diffLines = diffFirstOther.split('\n');

var result = {
    first: firstLines,
    second: secondLines,
    diff: diffLines
};
```

Later on, you figure out that the `diff()` function is kind of expensive in terms of CPU power, so you want it to run in another process. Lets see what are your options:

Callbacks

You need to change the function `diff` to take a callback, and you move the diffLines calculation inside. So far so good:

```
var first = files[0], other = files[1];
diff(first, other, function(err, diffFirstOther) {
```

```
    var diffLines = diffFirstOther.split('\n');
});
```

But wait, now the result needs to be inside that callback too.

```
var first = files[0], other = files[1];
diff(first, other, function(err, diffFirstOther) {
    var diffLines = diffFirstOther.split('\n');
    var result = {
        first: firstLines,
        second: secondLines,
        diff: diffLines
    };
});
var firstLines = first.split('\n').length,
    otherLines = other.split('\n').length;
```

This works but its horribly unintuitive because it looks like you're using the variables `firstLines` and `secondLines` before they are available. So you decide its best to move those lines at the top:

```
var first = files[0], other = files[1];
var firstLines = first.split('\n').length,
    otherLines = other.split('\n').length;

diff(first, other, function(err, diffFirstOther) {
    var diffLines = diffFirstOther.split('\n');
    var result = {
        first: firstLines,
        second: secondLines,
        diff: diffLines
    };
});
```

Promises

With promises our code can stay almost exactly the same. No need to move things around to make them more intuitive. No need to figure out whether we

need to stuff other lines inside our callback because they have a dependency on the result of that callback. Instead, we combine:

- the ability to attach callbacks whenever we want,
- the power to create new promises by returning from `.then()`, and
- the use of promise values to replace the original values

```
var first = files[0], other = files[1];
var pDiffFirstOther = diff(first, other);

var firstLines = first.split('\n').length,
    otherLines = other.split('\n').length;

var pDiffLines = pDiffFirstOther.then(function(diff) {
    return diff.split('\n');
});

var pResult = pDiffLines.then(function(diffLines) {
    return {
        first: firstLines,
        second: secondLines,
        diff: diffLines
    };
});
```

Everything is exactly the same, except that we now have promises instead of values.

Whenever we want to do some extra sync (or async) processing on these promises, we can unpack them with `.then()` and apply our transformation inside the callback, then return a new value which will be packed in a new promise. No need to move code around to make it fit under the same callback.

That's pretty great, isn't it? But wait 'til you see what happens with errors