PRACTICAL PROMISES

a pithy promise primer

WHAT IS A CALLBACK?

WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors…

- **Blocking**
- **Non-blocking**

BLOCKING CALLBACKS

*think: **portable code***

predicates

```
e.g. arr.filter(function predicate (elem) {…});
```

comparators

```
e.g. arr.sort(function comparator (elemA, elemB) {…});
```

iterators

```
e.g. arr.map(function iterator (elem) {…});
```

NON-BLOCKING CALLBACKS

*think: **control flow***

WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors…

- **Blocking**
- **Non-blocking**

WHAT IS A CALLBACK?

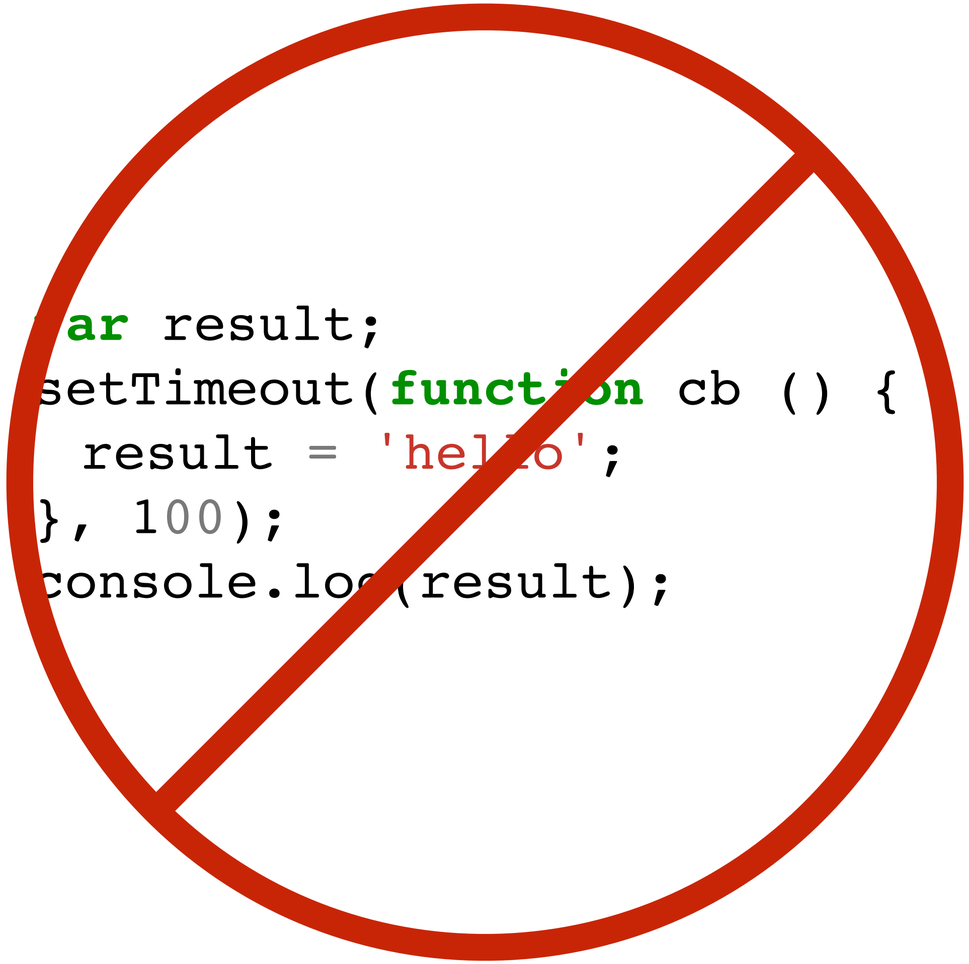
Technically: a function passed to another function

two flavors…

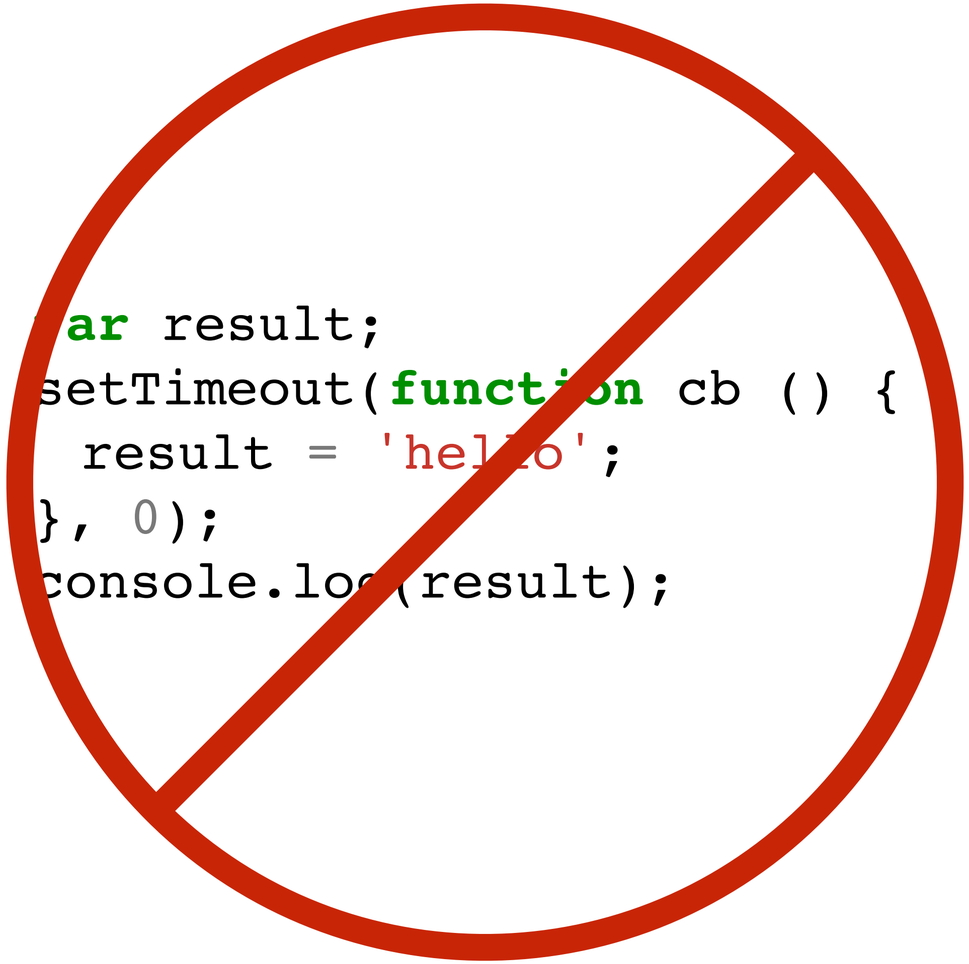
- Blocking
- Non-blocking

- event handler
- middleware
- vanilla async

LIKE THIS?


ar result;
setTimeout(function cb () {
result = 'hello';
}, 100);
console.log(result);

LIKE THIS?

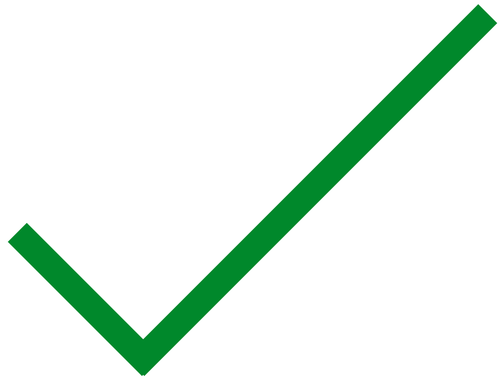

ar result;
setTimeout(function cb () {
result = 'hello';
}, 0);
console.log(result);

LIKE THIS?

```
var result = setTimeout(function () {
  return 'hello';
}, 0);
console.log(result);
```

LIKE THIS?

```
setTimeout(function cb () {
  var result = 'hello';
  console.log(result);
}, 0);
```



PROMISE

"A promise represents the eventual result of an asynchronous operation."

— THE PROMISES/A+ SPEC

CALLBACK V PROMISE

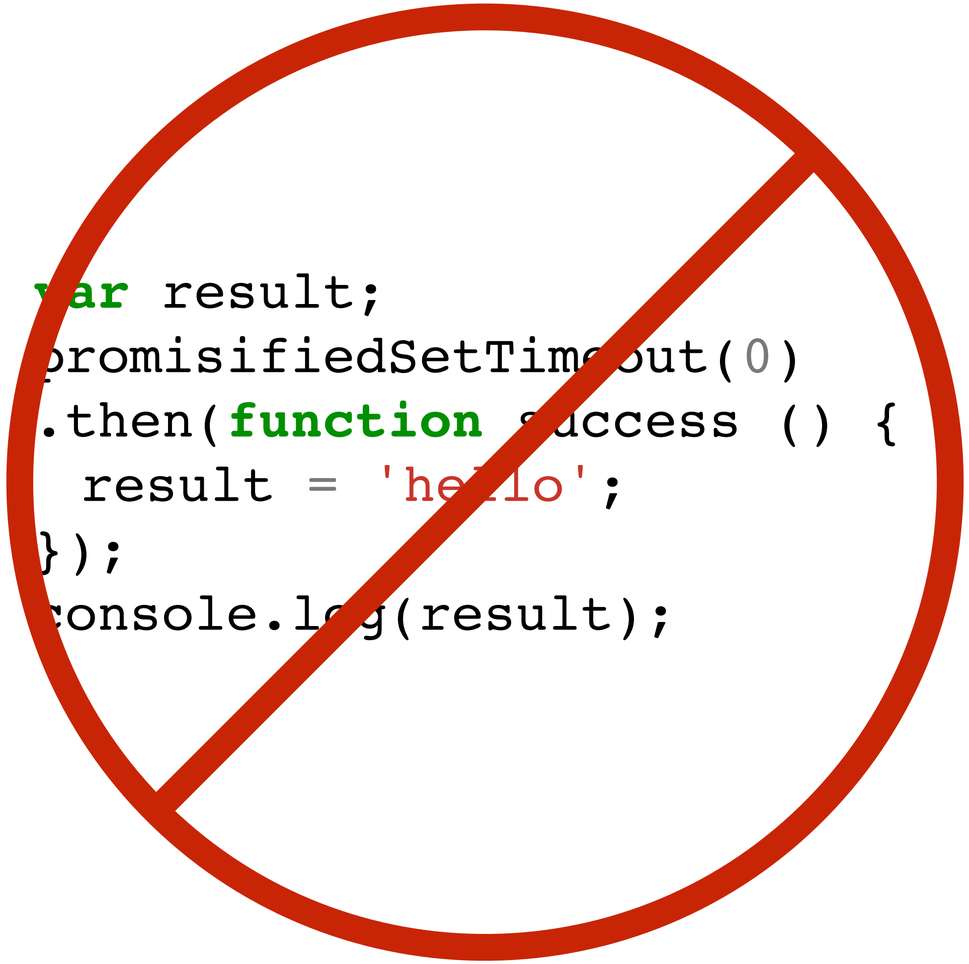
vanilla async callback

```
fs.readFile('file.txt',
  function callback (err, data) {…}
);
```

async promise

```
fs.readFileAsync('file.txt')
.then(
  function onSuccess (data) {…},
  function onError (err) {…}
);
```

LIKE THIS?

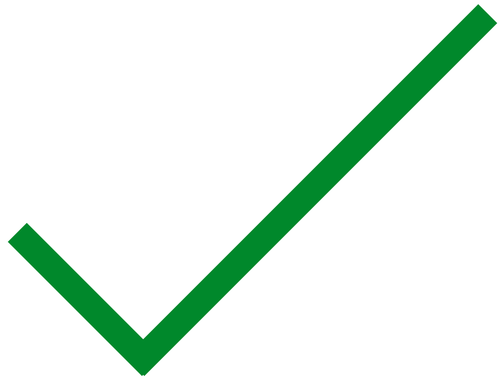

var result;
promisifiedSetTimeout(0)
.then(function success () {
result = 'hello';
});
console.log(result);

LIKE THIS?

```
var result = promisifiedSetTimeout(0)
.then(function success() {
  return 'hello';
});
console.log(result);
```

LIKE THIS?

```
promisifiedSetTimeout(0)
.then(function success () {
  var result = 'hello';
  console.log(result);
});
```



PROMISE

"A promise represents the eventual result of an asynchronous operation."

— THE PROMISES/A+ SPEC

READING A FILE

SYNCHRONOUS

```
var path = 'demo-poem.txt';
console.log('- I am first -');
try {
  var buff = fs.readFileSync(path);
  console.log(buff.toString());
} catch (err) {
  console.error(err);
}
console.log('- I am last -');
```

ASYNC (CALLBACKS)

```
var path = 'demo-poem.txt';
fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});
console.log('- I am first -');
```

ASYNC (PROMISES)

```
var path = 'demo-poem.txt';
promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString())
})
.then(function () {
  console.log('- I am last -');
});
console.log('- I am first -');
```

PORTABLE

```
const promise = fs.readFileAsync('file.txt');

// call some other function on it
doSomething(promise);

// export the promise, use it elsewhere!
module.exports = promise;
```

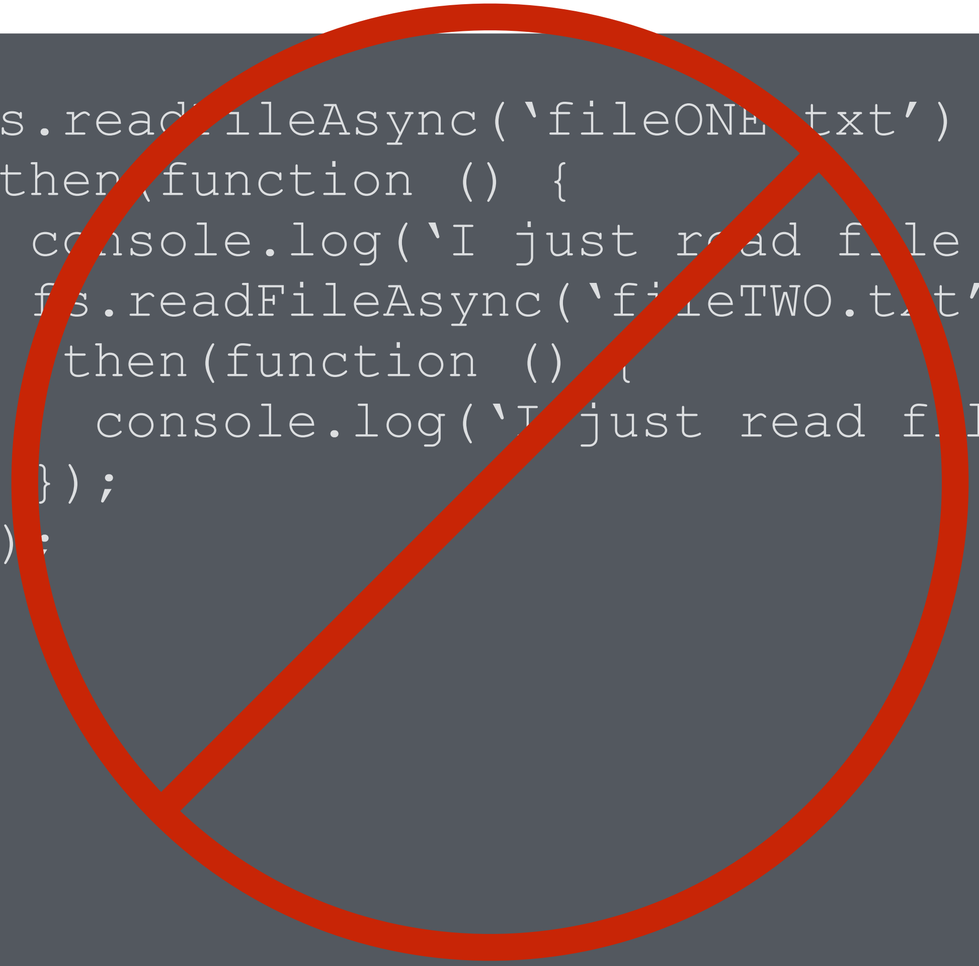
MULTIPLE HANDLERS

```
const promise = fs.readFileAsync('file.txt');

// do one thing when it finishes
promise
.then(function (fileContents) {
  console.log(fileContents);
});

// do another thing when it finishes
promise
.then(function () {
  // do something else
});
```

LINEAR/FLAT



```
fs.readFileAsync('fileONE.txt')
.then(function () {
  console.log('I just read file one');
  fs.readFileAsync('fileTWO.txt')
  .then(function () {
    console.log('I just read file two');
  });
});
```

LINEAR/FLAT

```
fs.readFileAsync('fileONE.txt')
.then(function (contents) {
  console.log('I just read file one:', contents);
  return fs.readFileAsync('fileTWO.txt');
})
.then(function (contents) {
  console.log('I just read file two:', contents);
});
```

UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
.then(function (contents) {
  console.log('I just read file one:', contents);
  return fs.readFileAsync('fileTWO.txt');
})
.then(function (contents) {
  console.log('I just read file two:', contents);
  return fs.readFileAsync('fileTHREE.txt')
})
.then(null, function (err) {
  console.log('An error occurred at some point');
  console.log(err);
});
```

UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
.then(function (contents) {
  console.log('I just read file one:', contents);
  return fs.readFileAsync('fileTWO.txt');
})
.then(function (contents) {
  console.log('I just read file two:', contents);
  return fs.readFileAsync('fileTHREE.txt')
})
.catch(function(err){
  console.log('An error occurred at some point');
  console.log(err);
});
```

PROMISE ADVANTAGES

- Portable
- Multiple handlers
- “Linear” or “flat” chains
- Unified error handling

IMPLEMENTATIONS

- Adehun
- avow
- ayepromise
- bloodhound
- bluebird
- broody-promises
- CodeCatalyst
- Covenant
- D
- Deferred
- fate
- ff
- FidPromise
- ipromise
- Legendary
- Lie
- microPromise
- mpromise
- Naive Promesse
- Octane
- ondras
- potch
- P
- Pacta
- Pinky
- PinkySwear
- Potential
- promeso
- promiscuous
- Promis
- Promix
- Promiz
- Q
- rsvp
- Shvua
- Ten.Promise
- then
- ThenFail
- typescript-deferred
- vow
- when
- yapa
- yapi
- Zousan