PROMISES, PROMISES

code class, 14 Oct 2016

@jbmoelker

Asynchrony

refers to the occurrence of events independently of the main program flow and ways to deal with such events.

@jbmoelker

Asynchronous JavaScript

- ☐ Callback functions
- ☐ Promises
- ☐ Generator functions
- ☐ Async / Await
- ☐ …

CALLBACK FUNCTIONS

@jbmoelker

A callback function is executed after an asynchronous operation has finished.

@jbmoelker

example: set timeout

```
window.setTimeout(callback, 1000, 'hello');

function callback (word) {
  console.log(word);
}
console.log('world');

// outputs 'world', 'hello'
```

developer.mozilla.org/en-US/docs/Web/API/WindowTimers/setTimeout

example: geolocation position

```
navigator.geolocation.getCurrentPosition(callback);

function callback (position) {
  console.log(
    position.coords.latitude,
    position.coords.longitude
  );
}
```

developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation

example: request animation frame

```
window.requestAnimationFrame(callback);

function callback (timestamp) {
  console.log(timestamp);
  window.requestAnimationFrame(callback);
}
```

developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

example: read file

```
const fs = require('fs'); // from NodeJS core

fs.readFile('path/to/file.txt', 'utf8', callback);

function callback (err, contents) {
  if (err) { console.error(err); }
  console.log(contents);
}
```

nodejs.org/api/fs.html#fs_fs_readfile_file_options_callback

example: http request

```
const request = require('request');

request('https://www.voorhoede.nl', callback);

function callback (err, response, body) {
  if (err) { console.error(err); }
  console.log(response.statusCode, body);
}
```

github.com/request/request

PROMISES

@jbmoelker

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

@jbmoelker

example: read file

```
const fs = require('fs'); // from NodeJS core

fs.readFile('path/to/file.txt', 'utf8', callback);

function callback (err, contents) {
  if (err) { console.error(err); }
  console.log(contents);
}
```

nodejs.org/api/fs.html#fs_fs_readfile_file_options_callback

example: read file promisified

```
const readFile = require('./my-lib/read-file');

readFile('path/to/file.txt', 'utf8')
  .then(contents => console.log(contents))
  .catch(err => console.error(err));
```

example: read file promisified

```
const readIntro = readFile('path/to/intro.txt', 'utf8');

readIntro
  .then(contents => console.log(contents))
  .catch(err => console.error(err));

// later
readIntro.then(contents => console.log(contents))
```

Promise constructor

```
function readIntro (filename, options) {
  return new Promise((resolve, reject) => {
    fs.readFile(filename, options, (err, contents) => {
      if (err) { reject(err); }
      resolve(contents);
    });
  }
}
```

developers.google.com/web/fundamentals/getting-started/primers/promises#promises_arrive_in_javascript

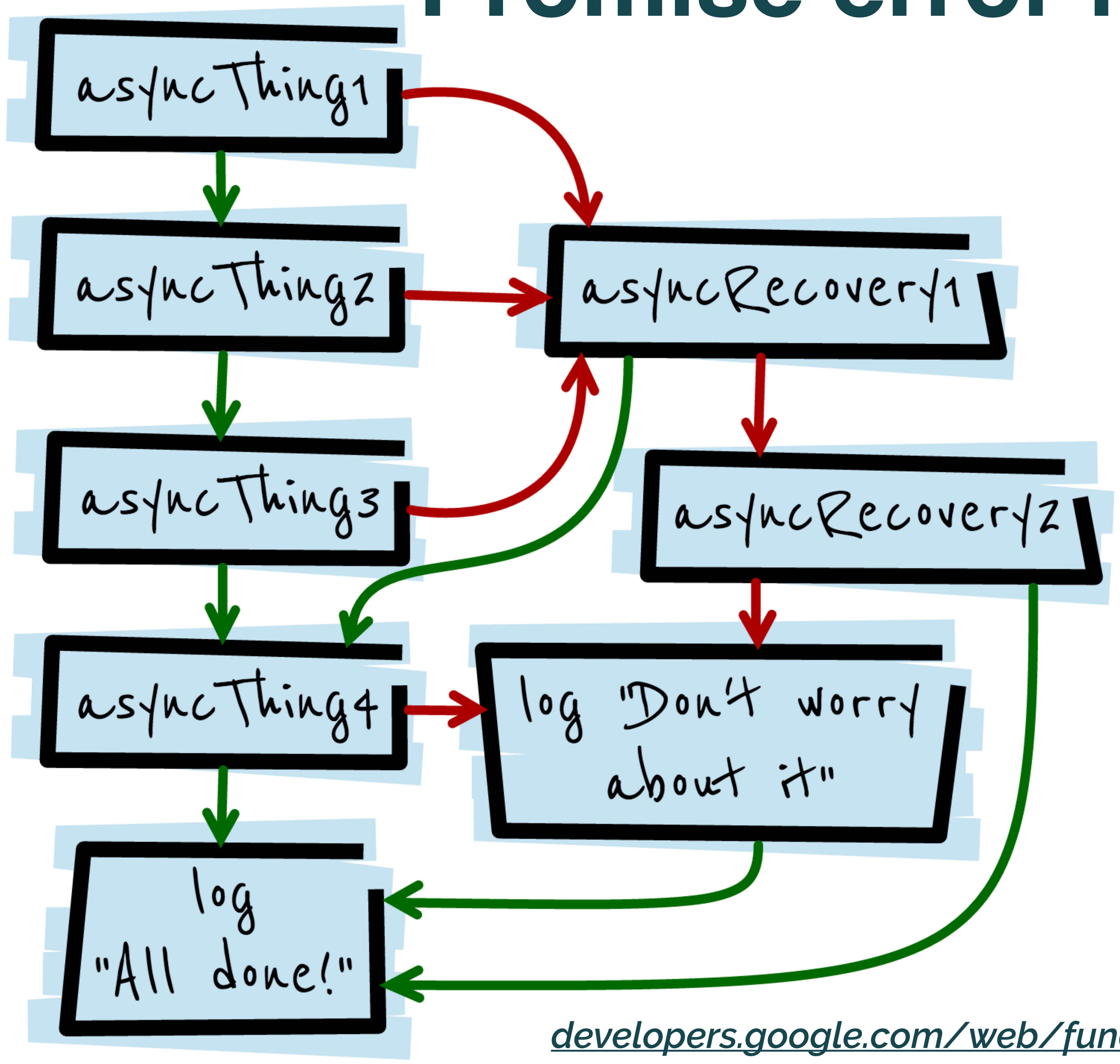
Promise chaining

```
const readFile = require('./my-lib/read-file');

readFile('path/to/file.txt', 'utf8')
  .then(contents => findAuthor(contents))
  .then(author => getBio(author))
  .then(bio => console.log(bio));
```

Promise error handling

```
const readFile = require('./my-lib/read-file');

readFile('path/to/file.txt', 'utf8')
  .then(contents => console.log(contents))
  .then(() => console.log('we succeeded'))
  .catch(err => console.error(err))
  .then(() => console.log('we are done');
```

Promise error handling



green lines are promises that fulfill, red lines are promises that reject.

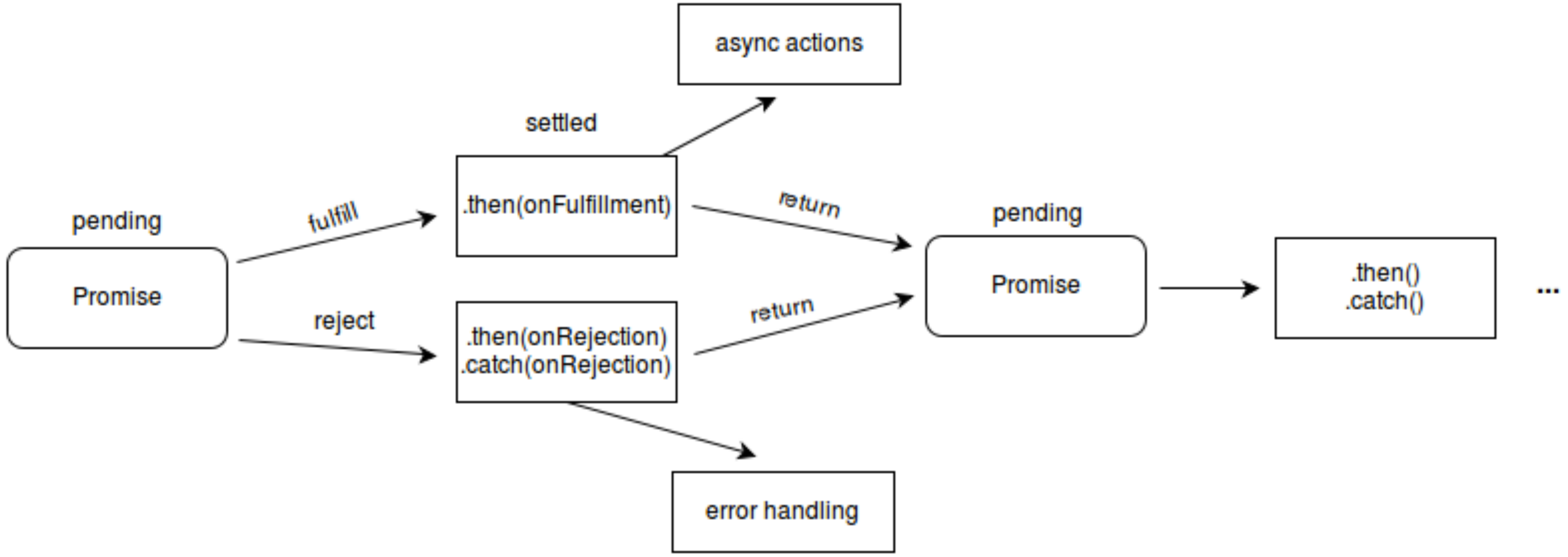
developers.google.com/web/fundamentals/getting-started/primers/promises

Promise states

- **fulfilled** - async task succeeded
- **rejected** - async task failed
- **pending** - async task not (yet) complete
- **settled** - async task completed

developers.google.com/web/fundamentals/getting-started/primers/promises#promise_terminology

Promise states



developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise#Description

“The nature of promises is that they remain **immune to changing** *circumstances.”*

— Frank Underwood, House of Cards

@jbmoelker

Promise methods

- Promise.**resolve**(value)
- Promise.**reject**(reason)
- Promise.**all**(array)
- Promise.**race**(array)

developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise#Methods

library: Bluebird

- **Promisify** callbacks:
  .promisify, .promisifyAll

- **Handle collections** of promises:
  .map, .filter, .each, .reduce, …

- **Utility** functions:
  .spread, .join, …

bluebirdjs.com

promisify with Bluebird

```
const promisify = require('bluebird').promisify;
const readFile = promisify(require('fs').readFile);

readFile('path/to/file.txt', 'utf8')
  .then(contents => console.log(contents))
  .catch(err => console.error(err));
```

bluebirdjs.com/docs/api/promise.promisify.html

EXERCISES

@jbmoelker

Exercise 1: Promisify get groceries

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))

// todo: promisify code so snippet above works
// plnkr.co/edit/lWajJm
```

Exercise 1: Promisify get groceries

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))

// solution: plnkr.co/edit/P3CUqA
```

Exercise 2: Clean up after

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))

// todo: ‘cleanUp(mess)’ when mean is consumed
// plnkr.co/edit/mgDjSA
```

Exercise 2: Clean up after

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))
  .then(mess => cleanUp(mess))

// solution: plnkr.co/edit/g7TE0D
```

Exercise 3: Tell mom if something’s wrong

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))
  .then(mess => cleanUp(mess))

// todo: use `.catch` to `tellMom(problem)`
// plnkr.co/edit/psGwVj
```

Exercise 3: Tell mom if something’s wrong

```
getGroceries(list)
  .then(ingredients => makeDinner(ingredients))
  .then(meal => haveDinner(meal))
  .catch(problem => tellMom(problem))
  .then(mess => cleanUp(mess))

// solution: plnkr.co/edit/frMkd7
```

Exercise 4: Set table while making dinner

```
getGroceries(list)
  .then(ingredients => /*
    todo: makeDinner and setTable at same time
  */)
  .then(meal => haveDinner(meal))
  .catch(problem => tellMom(problem))
  .then(mess => cleanUp(mess))

// plnkr.co/edit/IYLEsu
```

Exercise 4: Set table while making dinner

```
getGroceries(list)
  .then(ingredients => Promise.all([
      makeDinner(ingredients),
      setTable()
   ]))
  .then(meal => haveDinner(meal))
  // …

// solution: plnkr.co/edit/XBpjyH
```

Exercise 5: From marketplace or storage

```
getGroceries(/*
   todo: getGroceriesFromMarketPlace()
     or getGroceriesFromStorage(),
     and use whatever is fastest
 */)
 // ...

// plnkr.co/edit/yxE5hK
```

Exercise 5: From marketplace or storage

```
getGroceries(list => Promise.race([
    getGroceriesFromMarketPlace(list),
    getGroceriesFromStorage(list)
 ]))
 // ...

//solution: plnkr.co/edit/gbmCQg
```

DE VOORHOEDE

front-end developers