### **Special Topics**



#### **Motivation**



Asynchronous execution =>

• Callback Hell

```
a(function(resA) {
  b(resA, function(resB) {
    c(resB, function(err, resC) {
      if (err) fail();
      console.log(resC);
    });
  });
});
```

- Proper error handling
- Consistent state

#### API

```
KEEP
CALM
AND
KEEP YOUR
PROMISE
```

- 1. new Promise(function(resolve, reject) {
   ...
  })
- 2. p.then(onFulfilled[, onRejected])
- 3. p.catch(onRejected)
- 4. Promise.all(iterable)
- 5. Promise.race(iterable)
- 6. Promise.resolve(value)
- 7. Promise.reject(value)

# Pitfalls / Best Practices



since mid 2014 (FF 29, Chrome 33, Edge)

- 99% no new Promise
- Promise chaining
- Promise keeps its value
- p.catch resets promise chain
- use Promise.resolve and Promise.reject
- work well together with functional programming
- async / await (ES6)

#### Exercise



```
Promise.race = function(promises) {
   return new Promise((resolve, reject) => {
     promises.forEach(function(prom) {
        prom.then(resolve, reject);
     });
   });
}
```

#### Exercise



```
Promise.all = function(promises) {
  var accumulator = [];
  var ready = Promise.resolve(null);
  promises.forEach(function(promise, idx) {
    ready = ready.then(function() {
      return promise;
    }).then(function(value) {
      accumulator[idx] = value;
   });
  });
  return ready.then(function() {
    return accumulator;
  });
```



## **DEMO**

#### References



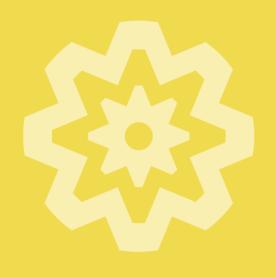
- JavaScript Promises: an Introduction <u>https://developers.google.com/web/fundamentals/primers/promises</u>
- Promise definition
   https://developer.mozilla.org/en/docs/We
   b/JavaScript/Reference/Global\_Objects/Promise
- Promises.js Patterns https://www.promisejs.org/patterns/
- Bluebird (Promise library)
   http://bluebirdjs.com/docs/why-promises.
   html

#### Motivation



- Single threaded environment -> computation slows render thread -> multi threading
- Multimedia + Games
- Types
  - O (Dedicated) Workers
    (IE 10, FF 3.5, Chrome 5, Safari 4)
    -> mid 2009
  - o Shared Workers
    (FF 29, Chrome 4) -> mid 2014
  - Service Workers
    (FF 44, Chrome 45) -> end 2015

#### API



```
// main.js
var worker = new Worker('task.js');
worker.postMessage(obj);
worker.onmessage = function(msg) {
worker.onerror = function(err) { ... }
worker.terminate();
// task.js
// self.onmessage = function(e) {
addEventListener('message', function(e) {
  self.postMessage(e.data);
}, false);
```

#### Restrictions



- 1300 kB max. message size
- working
  - o navigator
  - o location (read-only)
  - XMLHttpRequest
  - setTimeout() / clearTimeout()
  - setInterval() / clearInterval()
  - o importScripts() / Subworkers
- forbidden
  - o the DOM
  - o window
  - document
  - o parent

#### **Extras**



- Transferable objects
   worker.postMessage(buf, [buf]);
- Inline workers
  var blob = new Blob(['...']);
  var blobURL = URL.createObjectURL(blob);
  var worker = new Worker(blobURL);
- navigator.hardwareConcurrency
- Benchmark
  http://pmav.eu/stuff/javascript-webworke
  rs/



## **DEMO**

#### Service Workers



- programmable network proxy
- in the background
- replace AppCache
- heavy use of promises
- enable
  - push notificationsPush API + Notifications API
  - background sync
  - o pre-fetching

#### References



- The Basics of Web Workers
   https://www.html5rocks.com/en/tutorials/workers/basics/
- Service Workers: an Introduction <u>https://developers.google.com/web/fundam</u> <u>entals/primers/service-workers/</u>



Software to Solve Complex Video Problems







Encoding

**Analytics** 

Bitmovin API based products help developers around the world solve the most complex video problems with cloud native software that runs anywhere

### We are Hiring!



bitmovin.com/careers



twitter.com/bitmovin



facebook.com/bitmovin



linkedin.com/company/bitmovin