



# Promises

Das Beste, was JavaScript  
in den letzten fünf Jahren passiert ist



Johann H. Addicks - [http://de.wikipedia.org/wiki/Bild:Amstrad\\_PWC\\_-\\_Schneider\\_Joyce.jpg](http://de.wikipedia.org/wiki/Bild:Amstrad_PWC_-_Schneider_Joyce.jpg). CC BY-SA 3.0

# 1985

```
10 INPUT "Wie heisst Du?"; A$  
20 PRINT "Hallo "; A$
```

```
alert('Hallo '+prompt('Wie heit Du?'));
```

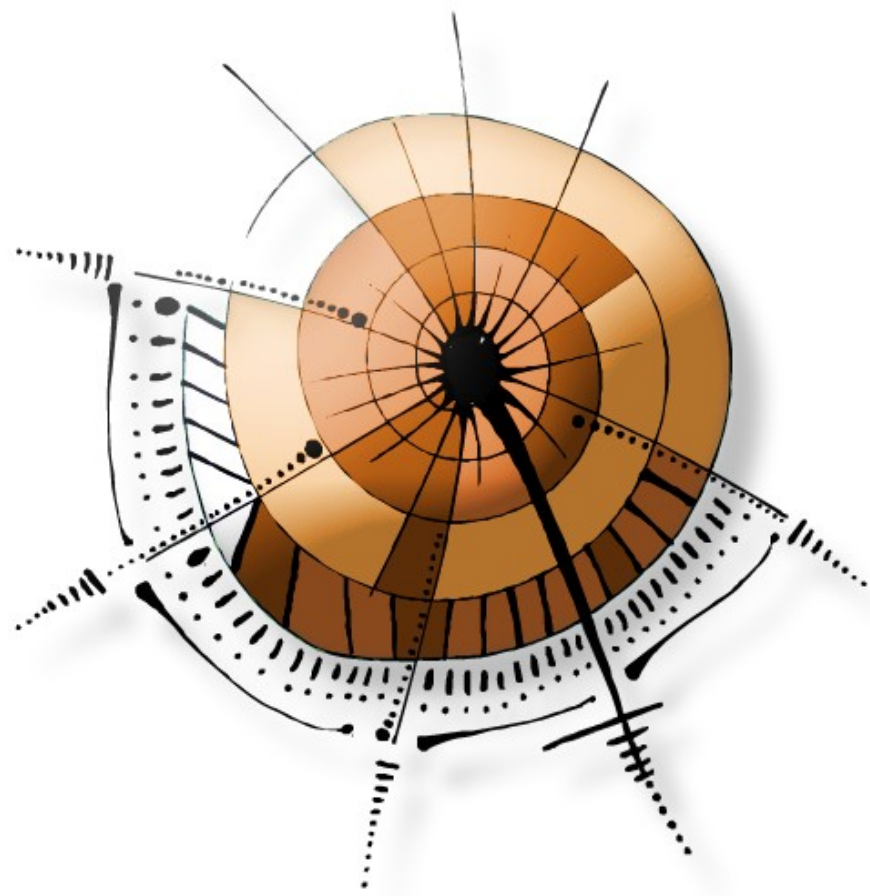
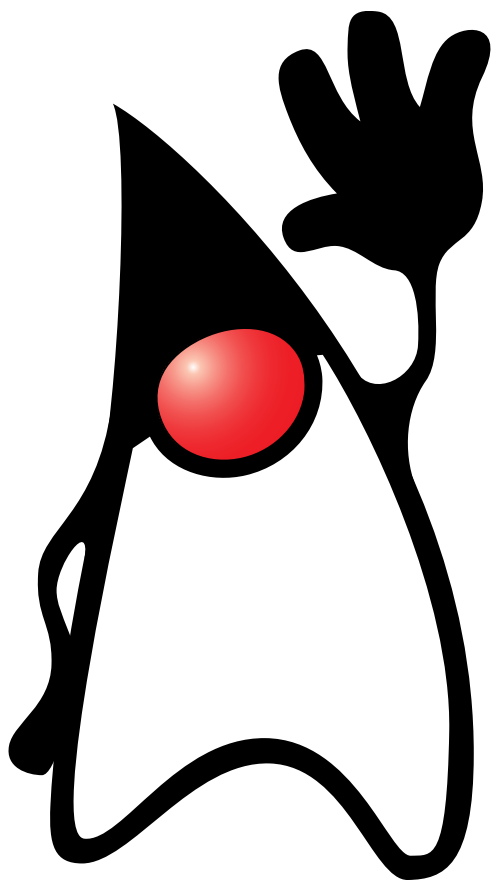


1995



1995





2005



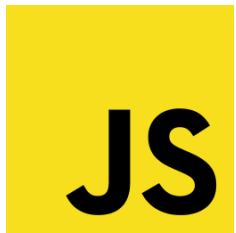
3.6



4-8



8



5

2010



2015 aka 6

2015



# Wozu?

Konzept für asynchrone Programmstruktur

# Asynchron?



Warten auf Nutzereingabe



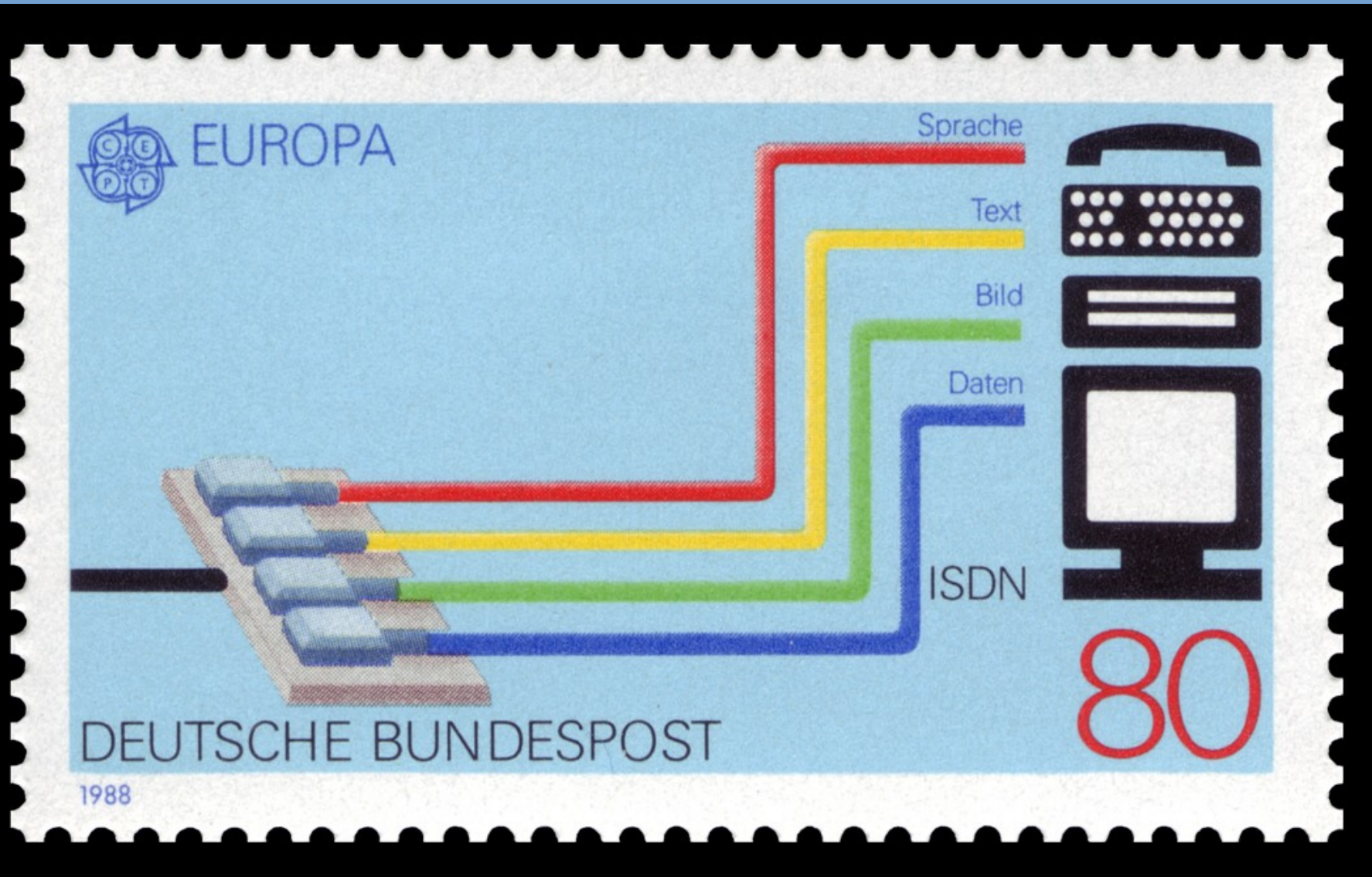
# Asynchron?



Warten auf DOMReady



# Asynchron?



Warten auf Antwort eines Requests



# Asynchron?



Kommunikation mit anderen Seiten

# Bisherige Lösungen

- Events

```
button.addEventListener('click',  
    doSomething, false);
```

```
window.onload = doSomething;
```

- Callbacks

```
fs.readFile(name, callback);
```



# Zu spät



```
document.querySelector('img')  
    .addEventListener('load', doSomething);
```

# Unübersichtlich

```
fs.readFile(name, function (err, result) {  
    if (err) {  
        handleError(err);  
    } else {  
        handleSuccess(result);  
    }  
});
```

# Unübersichtlich

```
function readJson(name, callback) {  
  fs.readFile(name, function (err, res) {  
    if (err) {  
      return callback(err);  
    }  
  
    try {  
      var parsed = JSON.parse(res);  
    } catch (err) {  
      return callback(err);  
    }  
  
    callback(null, parsed);  
  }  
}
```



# Sequenzen

```
function updateUser(userId, email, password, callback) {  
  User.get(userId, function (err, user) {  
    if (err) return callback(err);  
  
    user.setPassword(password, function (err) {  
      if (err) return callback(err);  
  
      user.setEmail(email, function (err) {  
        if (err) return callback(err);  
  
        user.save(function (err) {  
          if (err) return callback(err);  
  
          callback();  
        });  
      });  
    });  
  });  
}
```

# Parallelität

```
twitter.getTweets(username, function (result) {  
    tweets = result;  
    somethingFinished();  
});
```

```
facebook.getTimeline(username, function (result) {  
    timeline = result;  
    somethingFinished();  
});
```

```
instagram.getPhotosOfFood(username, function (result) {  
    photos = result;  
    somethingFinished();  
});
```

```
function somethingFinished() {  
    if (tweets && timeline && photos) {  
        showNiceSocialMediaSummary(tweets, timeline, photos);  
    }  
}
```



# Promises





# Was ist ein Promise?

Ein Versprechen auf eine zukünftige Antwort.

Ein Objekt mit einer Methode `then()`, die gewissen Anforderungen genügt.

Umkehrung der Verantwortlichkeit

# Was ist ein Promise?

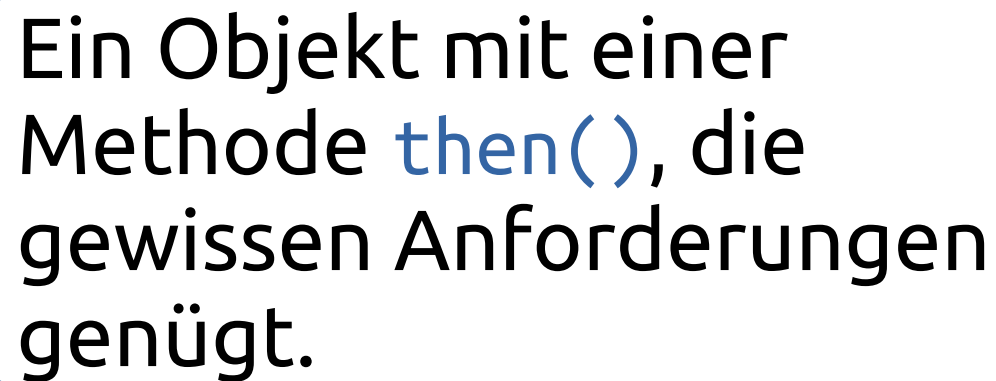


Umkehrung der  
Verantwortlichkeit

# Verantwortlichkeit

- Verwaltung der Callbacks
- Fehlerbehandlung
- Verletzt Prinzip der Single Responsibility
- Auslagern an Promise

# Was ist ein Promise?



Ein Objekt mit einer Methode `then()`, die gewissen Anforderungen genügt.

# Gewisse Anforderungen

- `successHandler`
- `errorHandler`

# Callback...

```
heyDoSomethingForMe(param, function(err, result) {  
  if (err) {  
    handleError(err);  
  } else {  
    handleSuccess(result);  
  }  
});
```



# ...wird zu Promise

```
heyDoSomethingForMe(param)  
  .then(handleSuccess, handleError);
```

# Promise

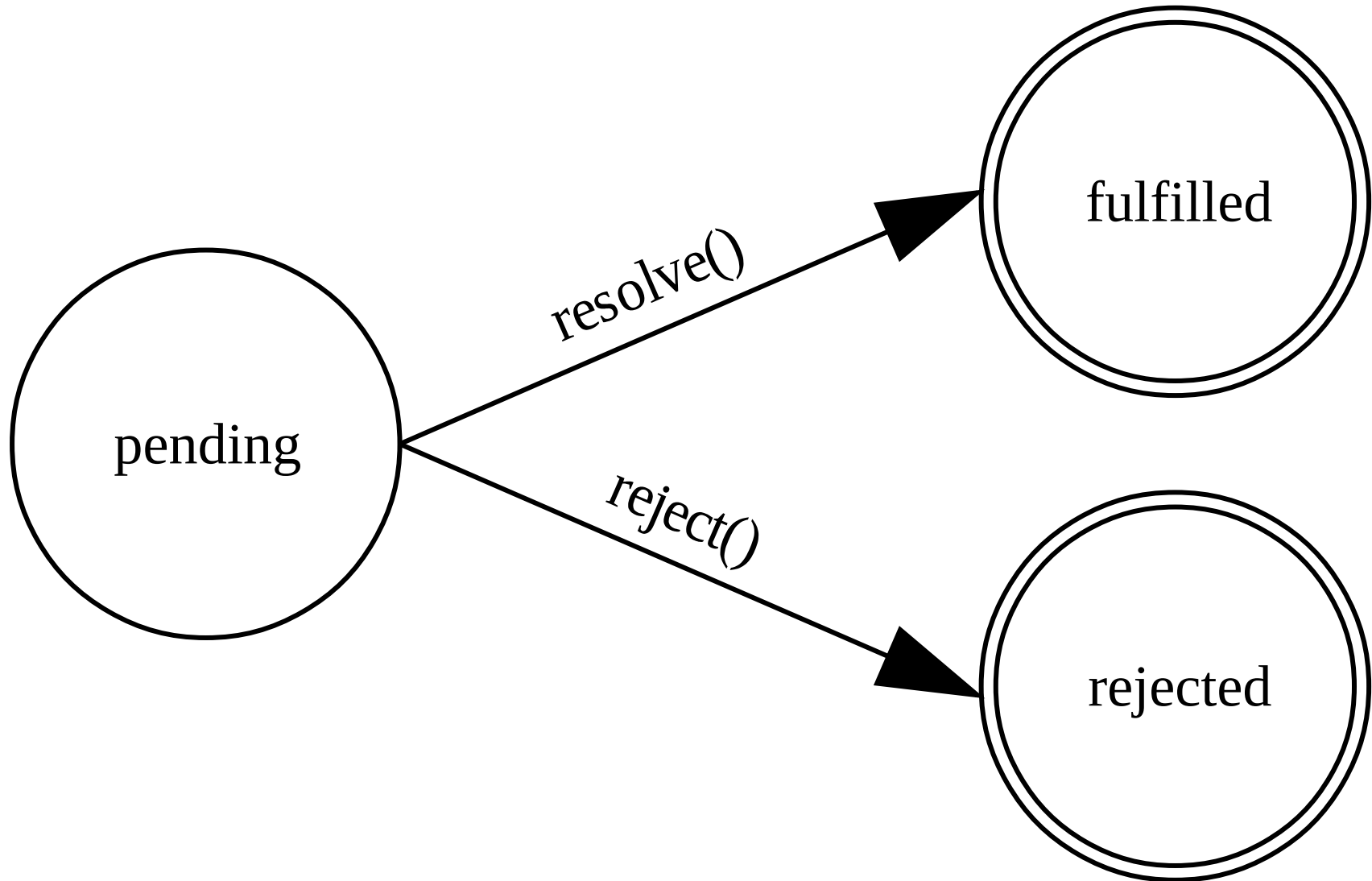
Drei Zustände:

- pending
- fulfilled
- rejected

Zwei Zustandsübergänge:

- resolve
- reject

# Promise



# Was ist ein Promise?



Ein Versprechen auf eine zukünftige Antwort.

# Promise erzeugen

```
function heyDoSomethingForMe(param) {  
    return new Promise(function (resolve, reject) {  
        setTimeout(resolve, 2000);  
    });  
}
```

# Promise erzeugen

```
function readFile(name) {  
    return new Promise(function (resolve, reject) {  
        fs.readFile(name, function (err, result) {  
            if (err) {  
                reject(err);  
            } else {  
                resolve(result);  
            }  
        });  
    });  
}
```



# Callback...

```
fs.readFile(name, function (err, result) {  
  if (err) {  
    handleError(err);  
  } else {  
    handleSuccess(result);  
  }  
});
```

# ...wird zu Promise

```
readFile(name)  
  .then(handleSuccess, handleError);
```

# Promise erzeugen

```
function delay(time) {  
    return new Promise(function (resolve) {  
        setTimeout(resolve, time);  
    });  
};
```

# Promise-basierter HTTP-Request

```
function request(url) {  
    return new Promise(function (resolve, reject) {  
        var req = new XMLHttpRequest();  
        req.open('GET', url);  
        req.onreadystatechange = function () {  
            if (req.readyState === 4) {  
                var response = {  
                    status: req.status,  
                    ok: req.status >= 200 && req.status < 300,  
                    data: req.responseText  
                };  
  
                if (response.ok) {  
                    resolve(response);  
                } else {  
                    reject(response);  
                }  
            }  
        };  
        req.send(null);  
    });  
};
```

# Promise-basierter HTTP-Request

```
request('/users/mreuter')  
  .then(function (response) {  
    header.setAvatar(response.avatarUrl);  
  });  
  
request('/users/mreuter')  
  .then(function (response) {  
    loginBox.setUsername(response.username);  
  });
```

# Gewisse Anforderungen

- `successHandler`
- `errorHandler`
- Status wird gehalten

# Status wird gehalten

```
userPromise = request('/users/mreuter');  
userPromise.then(setAvatar);
```

```
// später dann  
userPromise.then(setUsername);
```

# Status wird gehalten

```
var loadUserPromise;
```

```
function loadUser() {  
  if (!loadUserPromise) {  
    loadUserPromise = request('/users/mreuter');  
  }  
  return loadUserPromise;  
}
```

```
loadUser().then(setAvatar);
```

```
loadUser().then(setUsername);
```



# Gewisse Anforderungen

- `successHandler`
- `errorHandler`
- Status wird gehalten
- `then()` gibt wieder ein Promise zurück
- `successHandler` und `errorHandler` sind optional

# Chaining

heyDoSomethingForMe()

Erfolg



.then(handlerA)

Fehler



.then(handlerB, errorHandler)



Erfolg



.then(handlerC);

# Callback...

```
function updateUser(userId, email, password, callback) {  
  User.get(userId, function (err, user) {  
    if (err) return callback(err);  
  
    user.setPassword(password, function (err) {  
      if (err) return callback(err);  
  
      user.setEmail(email, function (err) {  
        if (err) return callback(err);  
  
        user.save(function (err) {  
          if (err) return callback(err);  
  
          callback();  
        });  
      });  
    });  
  });  
}
```

# ...wird zu Promise

```
function updateUser(userId, email, password) {  
  function setEmail(user) {  
    return user.setEmail(email);  
  }  
  
  function setPassword(user) {  
    return user.setPassword(password);  
  }  
  
  function saveUser(user) {  
    return user.save();  
  }  
  
  return User.get(userId)  
    .then(setPassword)  
    .then(setEmail)  
    .then(saveUser);  
}
```

# Chaining

```
request('/user/mreuter')  
  .then(function (response) {  
    return JSON.parse(response);  
  })  
  .then(setAvatar);
```

# Chaining

```
request( '/user/mreuter' )  
  .then(JSON.parse)  
  .then(setAvatar);
```

# Gewisse Anforderungen

- `successHandler`
- `errorHandler`
- Status wird gehalten
- `then()` gibt wieder ein `Promise` zurück
- `successHandler` und `errorHandler` sind optional


# Fehlerbehandlung

```
readFile(filename)  
  .then(JSON.parse)  
  .then(successHandler, errorHandler);
```



# Erfolg als Fehler behandeln

```
register()  
  .then(function (response) {  
    if (response.resultAddressCheck !== 0) {  
      return Promise.reject(response);  
    }  
  
    return response;  
  })  
  .then(successHandler, errorHandler);
```



Erzeugt ein Promise im  
Zustand rejected

# Fehler als Erfolg behandeln

```
checkCard()  
  .catch(function (error) {  
    if (error.message === "ACCOUNT_DISABLED") {  
      return { registered: true, disabled: true };  
    }  
  
    return Promise.reject(error);  
  })  
  .then(successHandler, errorHandler);
```

*kurz für .then(null, handler)*

*Wird umgewandelt in Promise  
im Zustand fulfilled*

# Zwischenschritt einbauen

```
loadUser()  
  .then(showUser);
```

loadUser resolt mit einem Objekt

```
{ name, street, zip, city }
```

API-Change zu

```
{ name, address: { street, zip, city } }
```

# Zwischenschritt einbauen

```
loadUser()  
  .then(function (user) {  
    if (typeof user.address !== 'object') {  
      user.address = {  
        street: user.street,  
        zip: user.zip,  
        city: user.city  
      };  
    }  
    return user;  
  })  
  .then(showUser);
```

# Gefahren der Fehlerbehandlung

```
loadUser()  
  .then(handleSuccess);
```

An orange curved arrow originates from the `loadUser()` call and points to a question mark, indicating that the `then` method only handles success and ignores potential errors.

?

```
loadUser()  
  .then(handleSuccess, handleError);
```

An orange straight arrow points downwards from the `handleError` parameter in the `then` method call to a question mark, indicating that this approach properly handles both success and error cases.

?

# Gefahren der Fehlerbehandlung

```
loadUser()  
  .then(handleSuccess, handleError)  
  .catch(function (error) {  
    setTimeout(function () {  
      throw Error;  
    });  
  });
```

Geht nur, wenn am Ende kein Promise zurückgegeben wird.

# Promise API

`new Promise(executor)`

`Promise.resolve()`

`Promise.reject()`

`Promise.all()`

`Promise.race()`

`Promise.prototype.then()`

`Promise.prototype.catch()`

# Ohne Promise.all

```
twitter.getTweets(username, function (result) {  
  tweets = result;  
  somethingFinished();  
});
```

```
facebook.getTimeline(username, function (result) {  
  timeline = result;  
  somethingFinished();  
});
```

```
instagram.getPhotosOfFood(username, function (result) {  
  photos = result;  
  somethingFinished();  
});
```

```
function somethingFinished() {  
  if (tweets && timeline && photos) {  
    showNiceSocialMediaSummary(tweets, timeline, photos);  
  }  
}
```



# Mit Promise.all

```
Promise.all([
    twitter.getTweets(username),
    facebook.getTimeline(username),
    instagram.getPhotosOfFood(username)
]).then(function (results) {
    var tweets = results[0],
        timeline = results[1],
        photos = results[2];

    showNiceSocialMediaSummary(
        tweets, timeline, photos);
});
```

Fulfilled, wenn alle Promises fulfilled sind

# Schöner mit ES2015

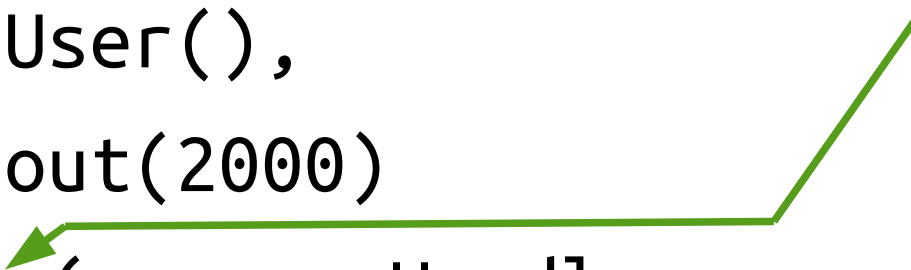
```
Promise.all([
  twitter.getTweets(username),
  facebook.getTimeline(username),
  instagram.getPhotosOfFood(username)
]).then(function (results) {
  showNiceSocialMediaSummary(...results);
});
```

# Promise.race

```
function timeout(time) {  
  return new Promise(resolve, reject) {  
    setTimeout(reject, time);  
  });  
}
```

```
Promise.race([  
  loadUser(),  
  timeout(2000)  
]).then(successHandler, errorHandler);
```

Resolved, wenn das erste  
Promise resolved



# Promises in freier Wildbahn

Can I use

Promises



Settings

1 result found

Promises - OTHER

Global

66.33% + 0.12% = 66.45%

A promise represents the eventual result of an asynchronous operation.

Current aligned

Usage relative

Show all

IE

Edge

\*

Firefox

Chrome

Safari

Opera

iOS Safari

\*

Opera Mini

\*

Android Browser

\*

Chrome for Android

8

31

9

38

43

10

39

44

11

12

40

45

8

32

9

8

44

44

13

41

46

9

33

42

47

34

43

48

# Polyfill

<https://github.com/jakearchibald/es6-promise>

# Promise-Bibliotheken

- Q  
<https://github.com/krisKowal/q>
- When.js  
<https://github.com/cujojs/when>
- RSVP.js  
<https://github.com/tildeio/rsvp.js/>
- \$q in Angular  
[https://docs.angularjs.org/api/ng/service/\\$q](https://docs.angularjs.org/api/ng/service/$q)

# Q

- `Q.when()`
- `Q.spread()`
- `Q.allSettled()`
- `Q.nfcall()`
- `Q.denodeify()`
- `Q.prototype.done()`
- `Q.prototype.finally()`





# DEVFEST

Matthias Reuter

matt@gweax.de