TypeScript

Promises

... asynchronous programming in JavaScript

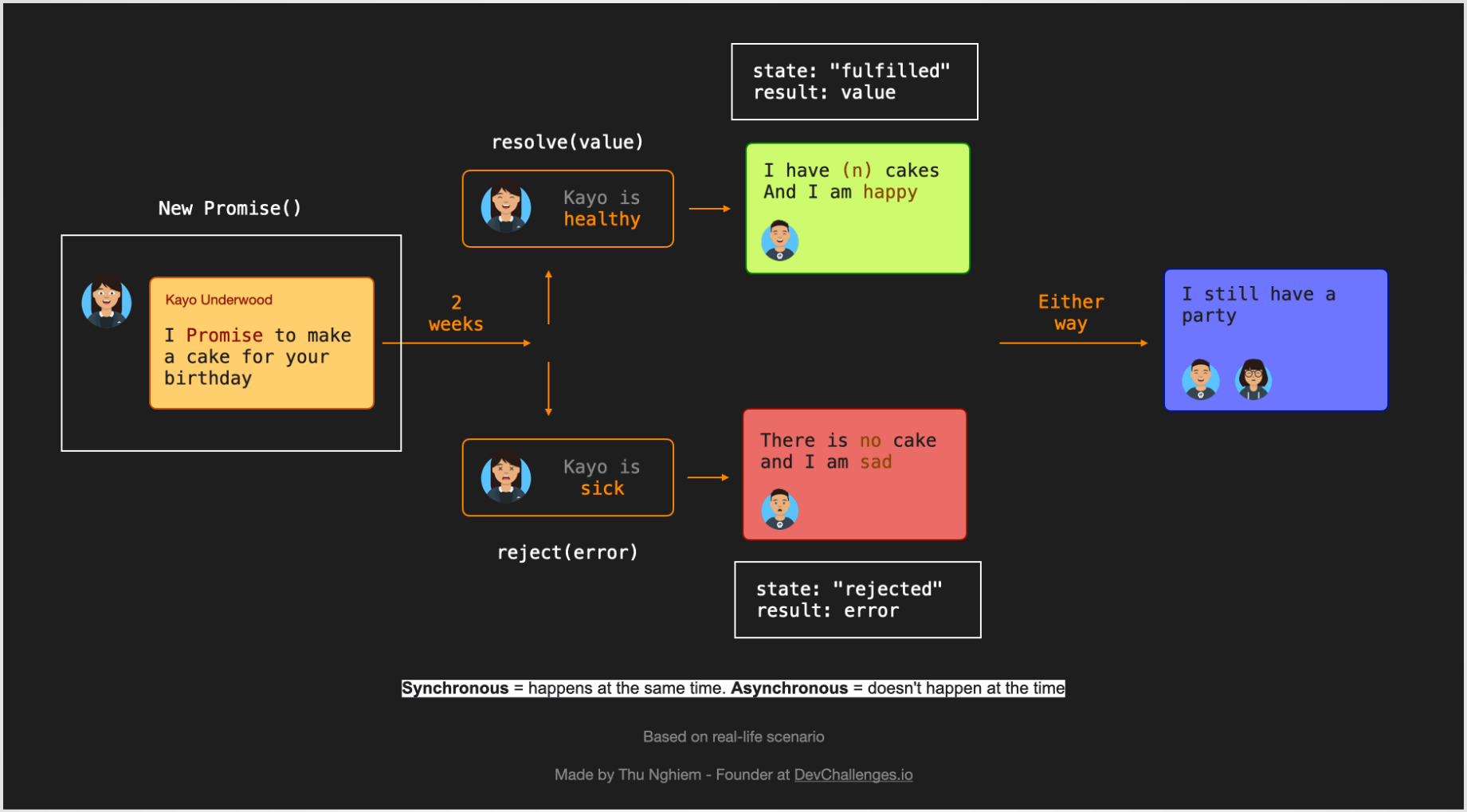
Promise

Erklärung

Promises

- Asynchronous programming pattern
  - *A proxy for a value not yet known*
- Leichter lesbar als mit Threads zu jonglieren
- In JS gibt es ja sowieso keine Threads :)

- **Async-Await** sind nur syntactic sugar um Promises in einer besser lesbaren Form darzustellen



Synchronous = happens at the same time. **Asynchronous** = doesn't happen at the time

Based on real-life scenario

Made by Thu Nghiem - Founder at DevChallenges.io

Promises

Neues Promise erzeugen

new Promise(...

```
1  const onMyBirthday = (isKayoSick) => {
2    return new Promise((resolve, reject) => {
3      setTimeout(() => {
4        if (!isKayoSick) {
5          resolve(2)
6        } else {
7          reject(new Error("I am sad"))
8        }
9      }, 2000)
10   })
11 }
```

- **setTimeout()** wird nur einmal ausgeführt

Benutzung

```
onMyBirthday(<true/false>)
  .then((result) => {
    console.log(`I have ${result} cakes`); // In the console: I have 2 cakes
  })
  .catch((error) => {
    console.log(error); // Does not run
  })
  .finally(() => {
    console.log("Party"); // Shows in the console no matter what: Party
  });
```

Warten auf mehrere Promises

```
1  Promise.allSettled([
2    Promise.resolve(33),
3    new Promise(resolve => setTimeout(() => resolve(66), 0)),
4    99,
5    Promise.reject(new Error('an error'))
6  ])
7  .then(values => console.log(values));
8
9  // [
10 //   {status: "fulfilled", value: 33},
11 //   {status: "fulfilled", value: 66},
12 //   {status: "fulfilled", value: 99},
13 //   {status: "rejected",  reason: Error: an error}
14 // ]
```

- Wartet bis alle 4 Promises aufgelöst wurden
- Deren Ergebnis ist egal (error, success)
- **allSettled([])** liefert wieder ein Promise
- Auf dieses Promise kann man **await** ausführen, wenn man mag

Promises

async-await

Promise vs. Async Await

Promise Chains	Async/Await
Scope	
• Only the promise chain itself is asynchronous	• The entire wrapper function is asynchronous
Logic	
• Synchronous work can be handled in the same callback • Multiple promises use Promise.all()	• Synchronous work needs to be moved out of the callback • Multiple promises can be handled with simple variables
Error Handling	
• Then • Catch • Finally	• Try • Catch • Finally

Vergleich: new Promise() vs. Async

```
1 // p(number) is an async function...
2
3 methodReturningPromise (number) {
4   return p(number).then((result) => {
5     console.log(result)
6   })
7 }
```

```
1 // p(number) is an async function...
2
3 async methodReturningPromise (number) {
4   const result = await p(number)
5   console.log(result)
6 }
```

Man kann auch ohne Probleme mischen
(die Funktion der linken Variante kann zum
Beispiel auch **async** sein)

Vergleich: Await vs. then

```
1  // p1(), p2() and p3() are async functions...
2
3  async myThen () {
4    p1().then((r1) => {
5      p2().then((r2) => {
6        p3().then((r3) => {
7          console.log(r1 + r2 + r3)
8        })
9      })
10   }).catch((e) => {
11     console.log('An error ocurred.')
12   })
13 }
```

```
1  // p1(), p2() and p3() are async functions...
2
3  async myAwait () {
4    try {
5      const r1 = await p1;
6      const r2 = await p2;
7      const r3 = await p3;
8      console.log(r1 + r2 + r3)
9    } catch (e) {
10     console.log('An error ocurred.')
11   }
12 }
```

Unterschiede

- Eine **async** function returned immer ein pending Promise
- Bei **await** wird die aktuelle Funktion unterbrochen bis die damit verknüpfte Funktion aufgelöst ist
  - **await** generiert eigentlich einen **then** Block
  - Alles unterhalb eines **await** wird suspended
- Mit **then** kann man allerdings mehrere Funktionen parallel starten

- **then** kann ich auch in nicht-Async-Methoden verwenden

Referenzen

- https://levelup.gitconnected.com/async-await-vs-promises-4fe98d11038f
- https://www.freecodecamp.org/news/learn-promise-async-await-in-20-minutes/
-