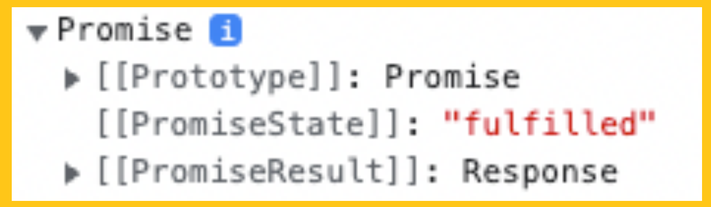
HEIG-VD / INGÉNIERIE DES MÉDIAS / PROGRAMMATION WEB

PROMISES

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

HE VD
IG

Avec les promesses, plus besoin d’événements et de callbacks imbriqués pour gérer l’asynchronicité.



Les Promises sont la façon “contemporaine” de gérer l’asynchronicité en JavaScript.

Une Promise est un espace réservé pour le résultat futur d’une opération asynchrone.

Autrement dit, il s’agit d’un container pour une valeur livrée de façon asynchrone.

En bref, c’est un container pour une valeur future.

HE VD IG

Une promesse que je pourrai voir un concert quand (et si) celui-ci a lieu.

🤞 J’achète le billet maintenant

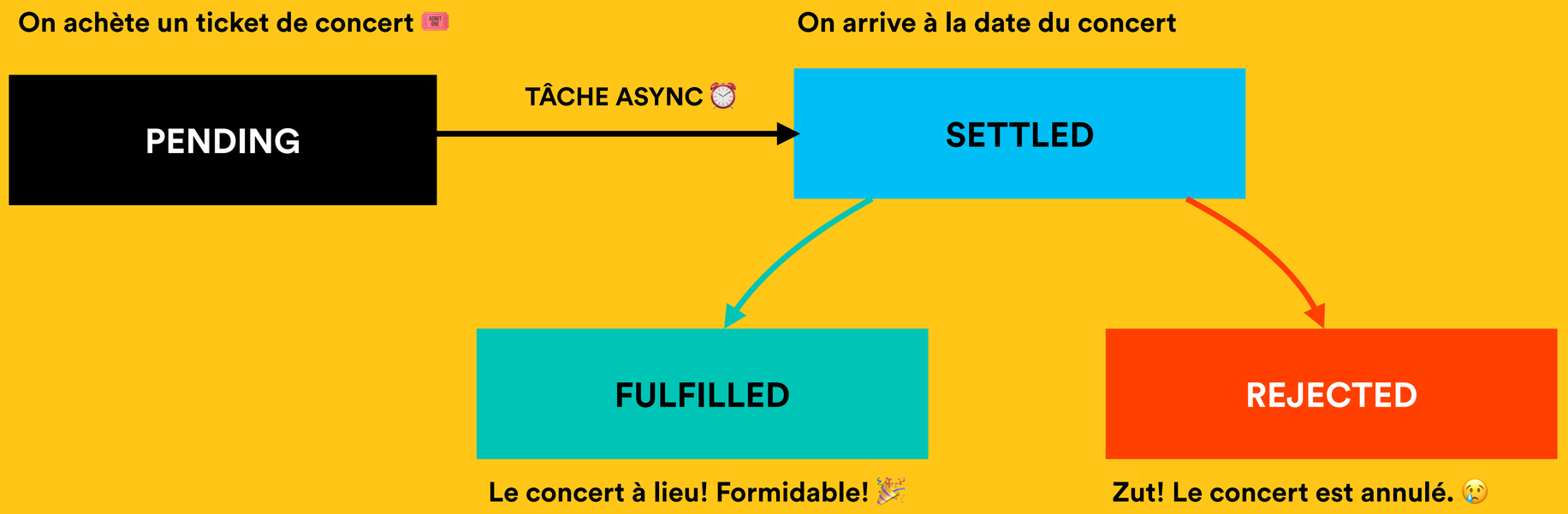
⏰ La tournée suit son court (ou se casse la figure) de façon asynchrone (je peux faire autre chose en attendant la date).

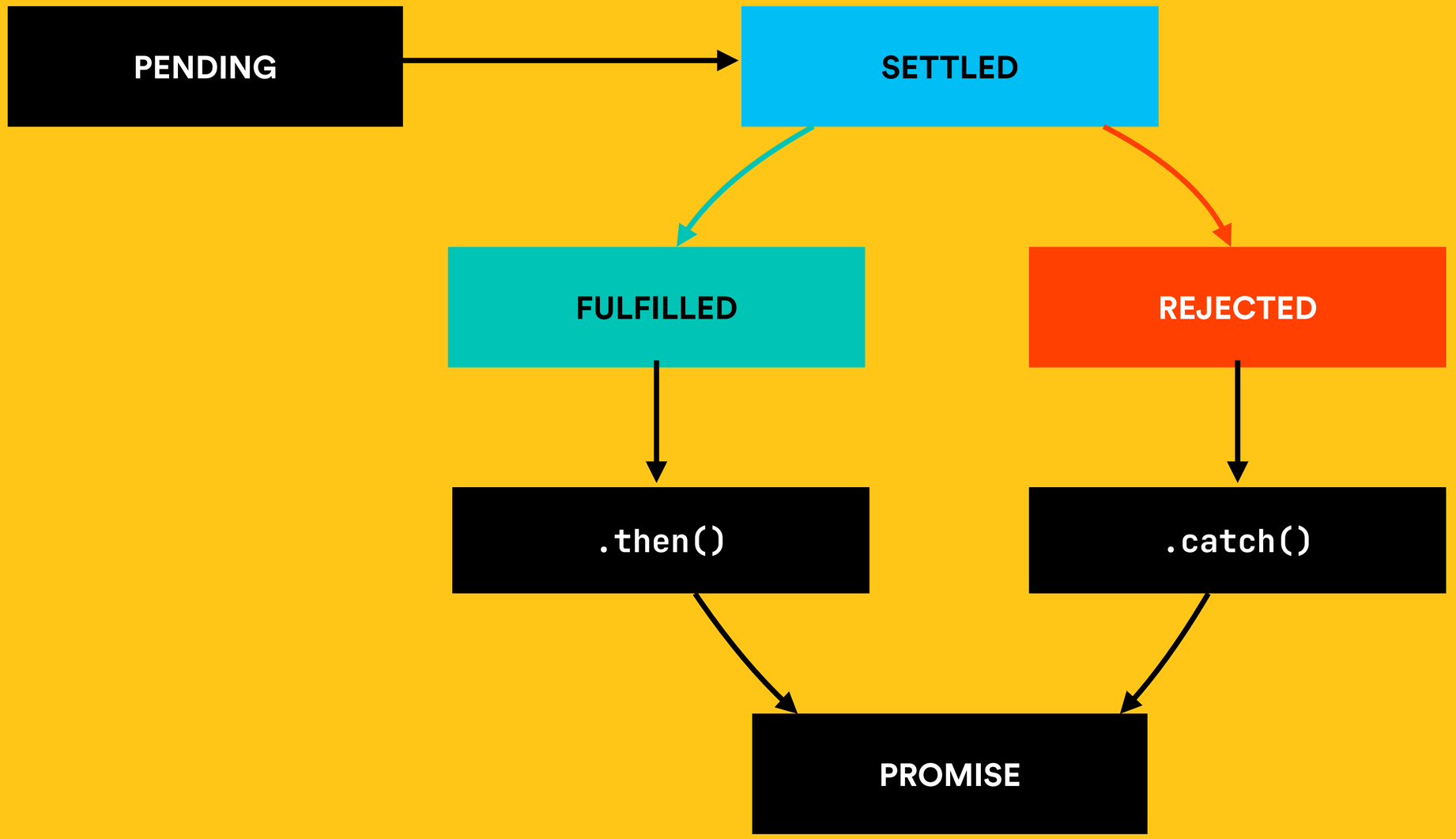
🎉 Si la date programmée a lieu, je peux entrer dans la salle et assister au concert.

On achète un ticket de concert 🎟️

PENDING


On achète un ticket de concert 🎟️
On arrive à la date du concert
PENDING
TÂCHE ASYNC ⏰
SETTLED


On achète un ticket de concert 🎟️
On arrive à la date du concert
PENDING
TÂCHE ASYNC ⏰
SETTLED
FULFILLED
REJECTED
Le concert à lieu! Formidable! 🎉
Zut! Le concert est annulé. 😢


PENDING
SETTLED
FULFILLED
REJECTED
.then()
.catch()
PROMISE

FETCH

```
fetch(url)
  .then((res) ⇒ res.json())
  .then((data) ⇒ console.log(data))
  .catch((err) ⇒ console.err(err))
  .finally(() ⇒ console.log("Done!"))
```

L’API Fetch est une alternative moderne à XMLHttpRequest. Il s’agit d’une simple fonction prenant en paramètre une url.

La fonction `fetch()` retourne systématiquement une Promise. Le seul motif de rejection est une erreur de réseau (même un status 404 est considéré comme fulfilled).

```
Promise.all([fetch(url), fetch(url)])
      .then((res) ⇒ console.log(res));
```

Si vous avez plusieurs requêtes `fetch()` à effectuer, le seul moyen de les envoyer en parallèle est d'utiliser les méthodes suivantes:

`Promise.all()`

Fulfilled lorsque toutes les promesses sont tenues. Rejected si une seule promesse est rompue. La valeur promise est un tableau des toutes les résolutions.

HEIG VD