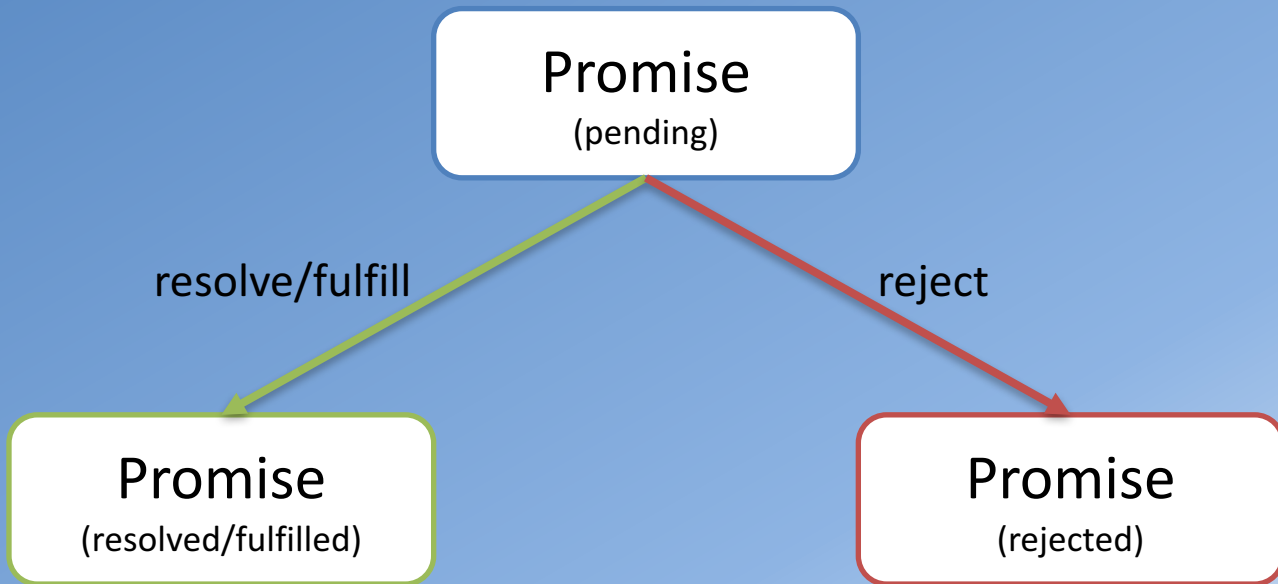


Promise

- Promise is a mechanism that supports asynchronous computation
- Proxy for a value not necessarily known when the promise is created:
 - It represents a value that may be available now, or in the future, or never

Promise



`new Promise (function (resolve, reject) { . . . });`

Promise Example

```
...
getDishes()
    .then ( function(result) {
    })
    .catch ( function(error) {
    });

getDishes(): Promise<Dish[]> {
    return new Promise (
        function(resolve, reject) {
            // do something
            if (successful) {
                resolve(result);
            }
            else {
                reject(error);
            }
        }
    );
}
```

The diagram illustrates the internal structure of a Promise. On the left, a code snippet shows a function `getDishes()` returning a Promise, which is then chained with `.then()` and `.catch()` methods. On the right, the implementation of `getDishes()` is shown as a function that returns a new `Promise` object. The `Promise` constructor takes a function with `resolve` and `reject` arguments. A blue arrow points from the `return new Promise` line in the implementation to the `getDishes()` call in the snippet. A yellow arrow points from the `if (successful) { resolve(result); }` block in the implementation to the `.then` method in the snippet. A red arrow points from the `else { reject(error); }` block in the implementation to the `.catch` method in the snippet.

Why Promises?

- Solves the callback hell (heavily nested callback code) problem
- Promises can be chained
- Can immediately return:
 - `Promise.resolve(result)`
 - `Promise.reject(error)`