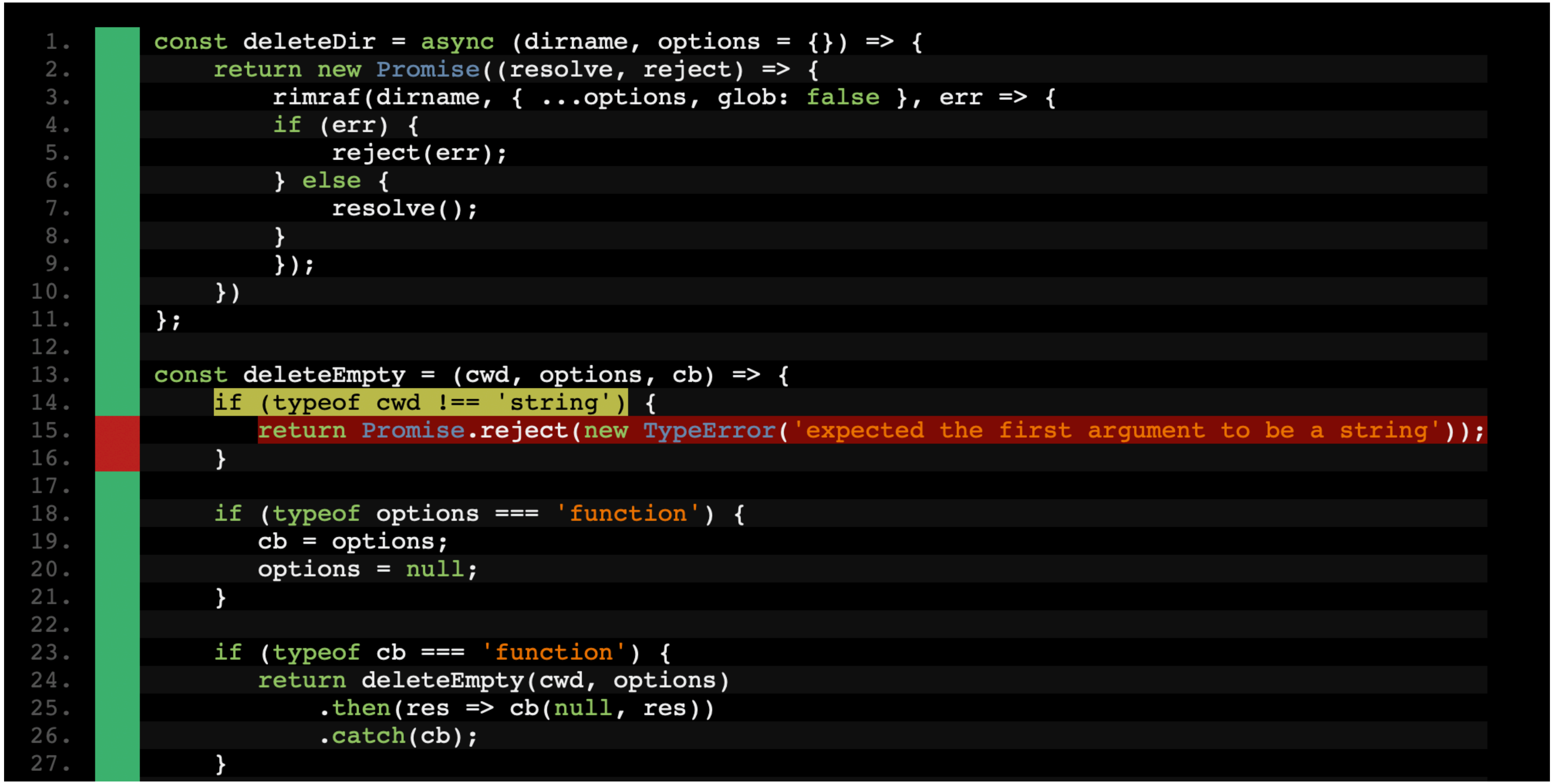
JStudy

A brief intro to code coverage, JS promises, and asynchrony

Code Coverage

- **Statement Coverage**
- **Branch Coverage**



```
const deleteDir = async (dirname, options = {}) => {
    return new Promise((resolve, reject) => {
        rimraf(dirname, { ...options, glob: false }, err => {
        if (err) {
            reject(err);
        } else {
            resolve();
        }
        });
    })
};

const deleteEmpty = (cwd, options, cb) => {
    if (typeof cwd !== 'string') {
      return Promise.reject(new TypeError('expected the first argument to be a string'));
    }

    if (typeof options === 'function') {
      cb = options;
      options = null;
    }

    if (typeof cb === 'function') {
      return deleteEmpty(cwd, options)
          .then(res => cb(null, res))
          .catch(cb);
    }
```

Asynchronous Code

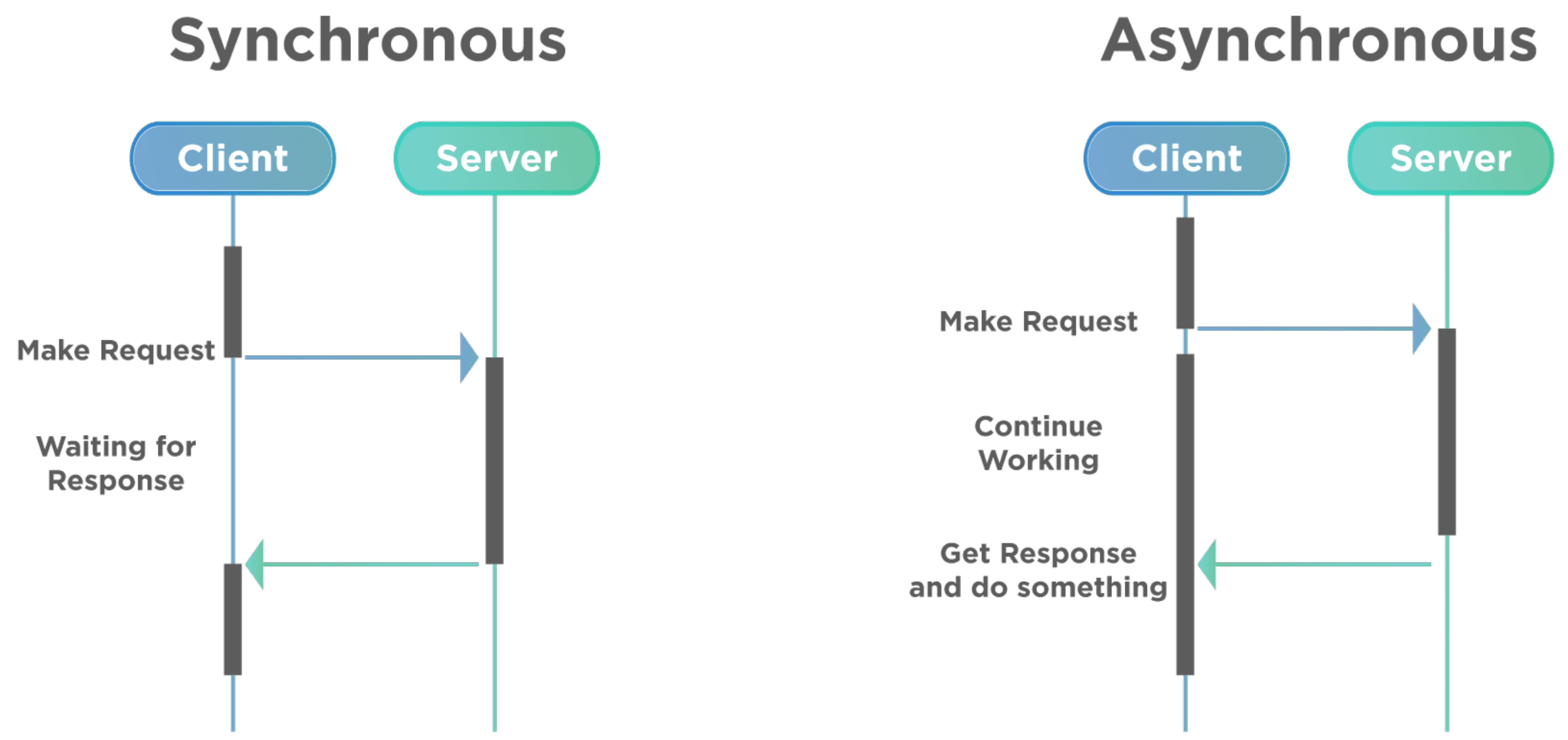
```
readFileAsync(path) {
  // read the file at “path”

  // mechanism to inform end of task

  // return
}

Main() {
  readFileAsync(path);
  doOtherStuff();
}
```

These don’t block execution



Asynchrony in JavaScript

- Promises
- async/await
- Callbacks

JavaScript Promises

Promise Definition:

The `Promise` object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Promise States

A `Promise` is in one of these states:

- pending: initial state, neither fulfilled nor rejected.
- fulfilled: meaning that the operation was completed successfully.
- rejected: meaning that the operation failed.

Example promise

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});
```

Asynchronous code with Promises

```
function readFileAsync(path) {
    return new Promise(function (resolve, reject) {
       readFile(path, function (err, result) {
            if (err) {
                reject(err);
            } else {
                resolve(result);
            }
        });
    });
}

function main() {
    var path = '/path/to/file'
    readFileAsync(path)
        .then(function onResolve(result) {
            // do something with result
        })
        .catch(function onError(error) {
            // handle error
        })
}
```

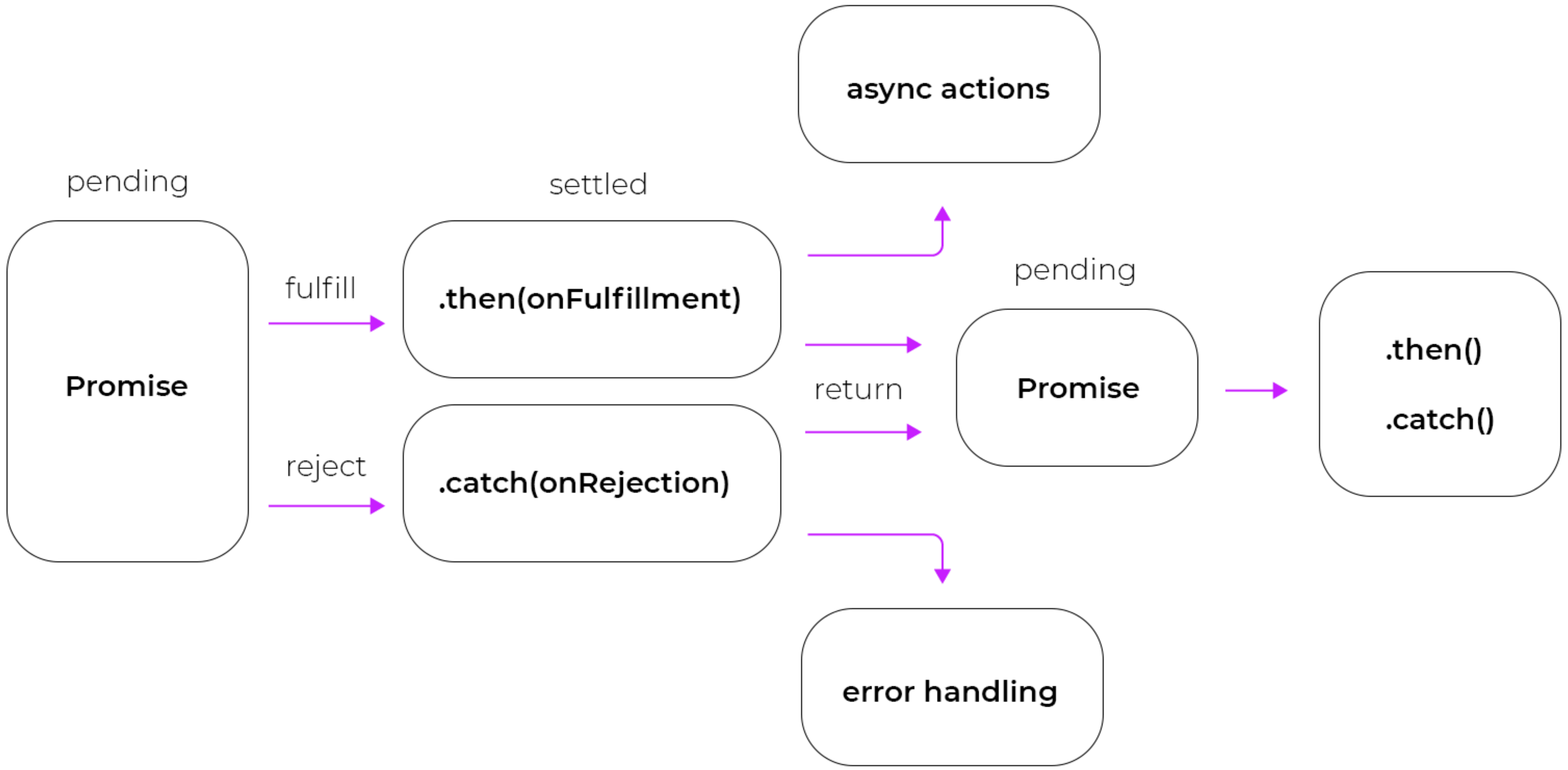
Asynchronous code with await

The await operator is used to wait for a Promise.
It can only be used inside an async function within regular JavaScript code

- pauses `async` function execution until promise settlement.
- resumes execution after fulfillment.
- When resumed, the value of fulfilled `Promise is returned`.
- If the `Promise` is rejected, the `await` expression throws the rejected value.

```
async function main() {
    var path = '/path/to/file'
    try {
        var result = await readFileAsync(path)
        // do something with result
    } catch (error) {
        // handle error
    }
}
```

Promise Behavior

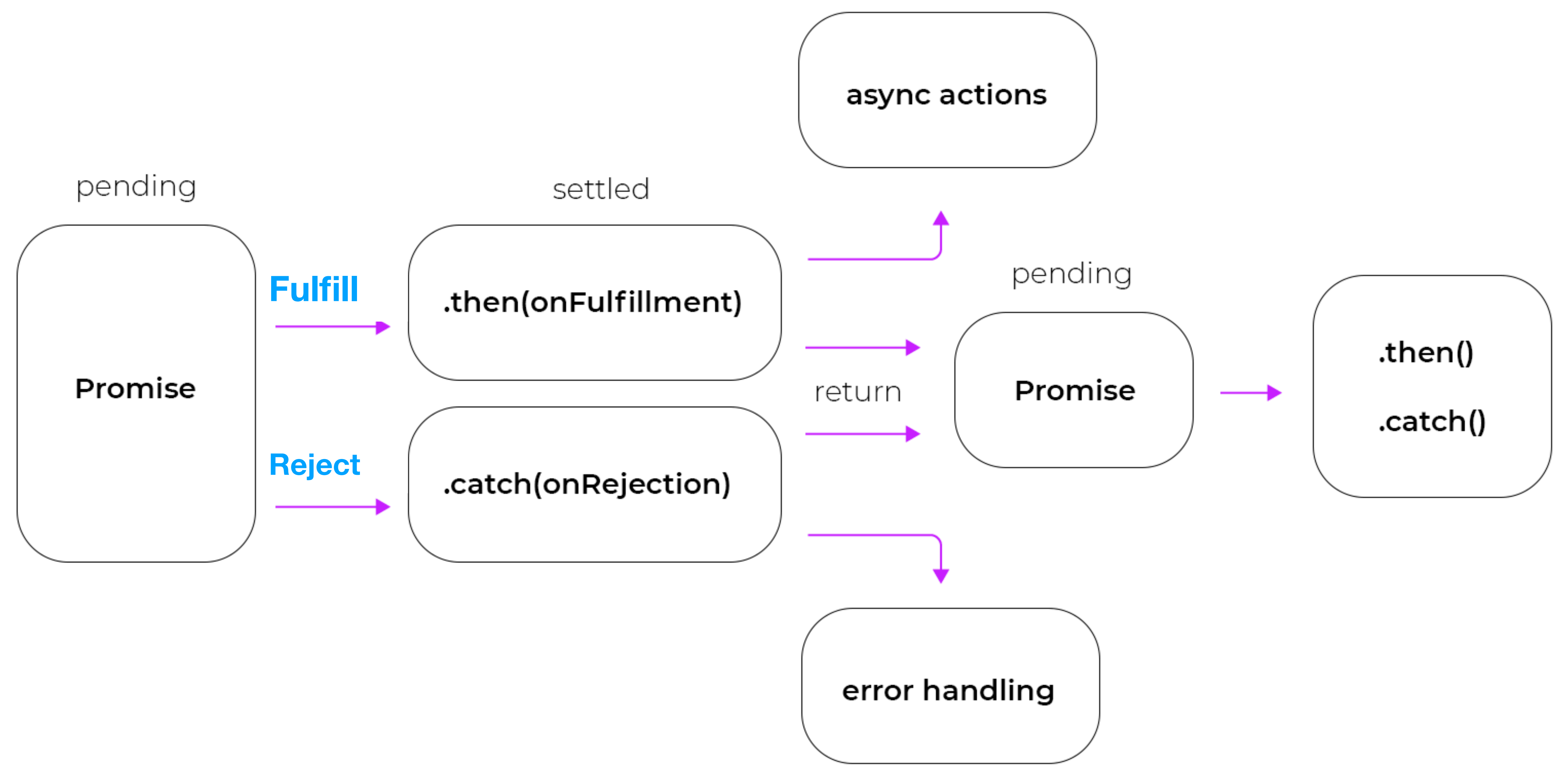


Promise Settlement

Promises can be settled by calls to “resolve” and “reject” arguments

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});
```

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(new Error(‘error’));
  }, 300);
});
```

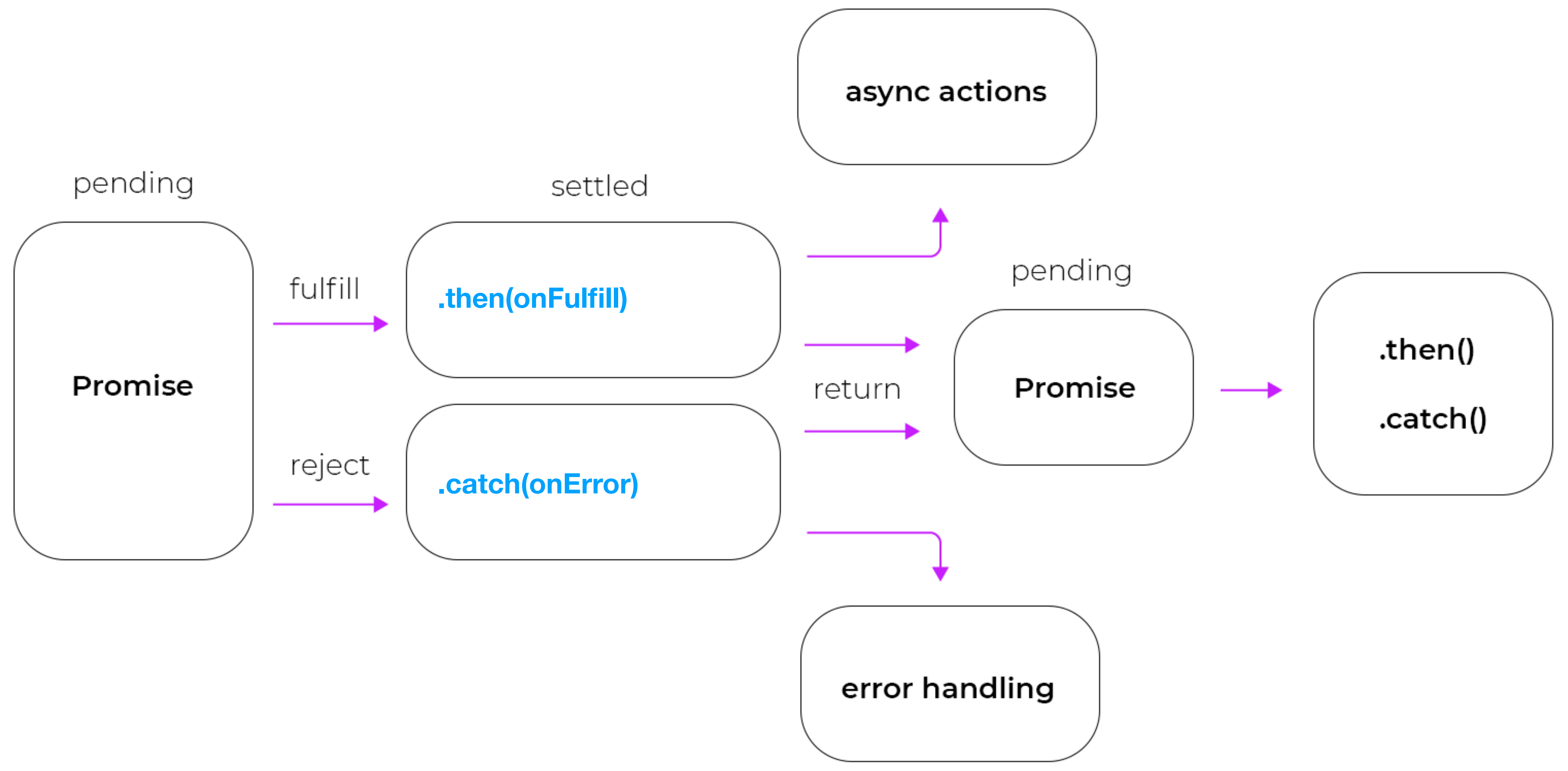


Promise Reactions

We can register reaction handlers on promises using .then and .catch

```
const myPromise = new Promise((resolve, reject) => {
   resolve(777);
});

// At this point, "myPromise" is already settled.
myPromise.then((val) => {
   // Do something with val
});
```

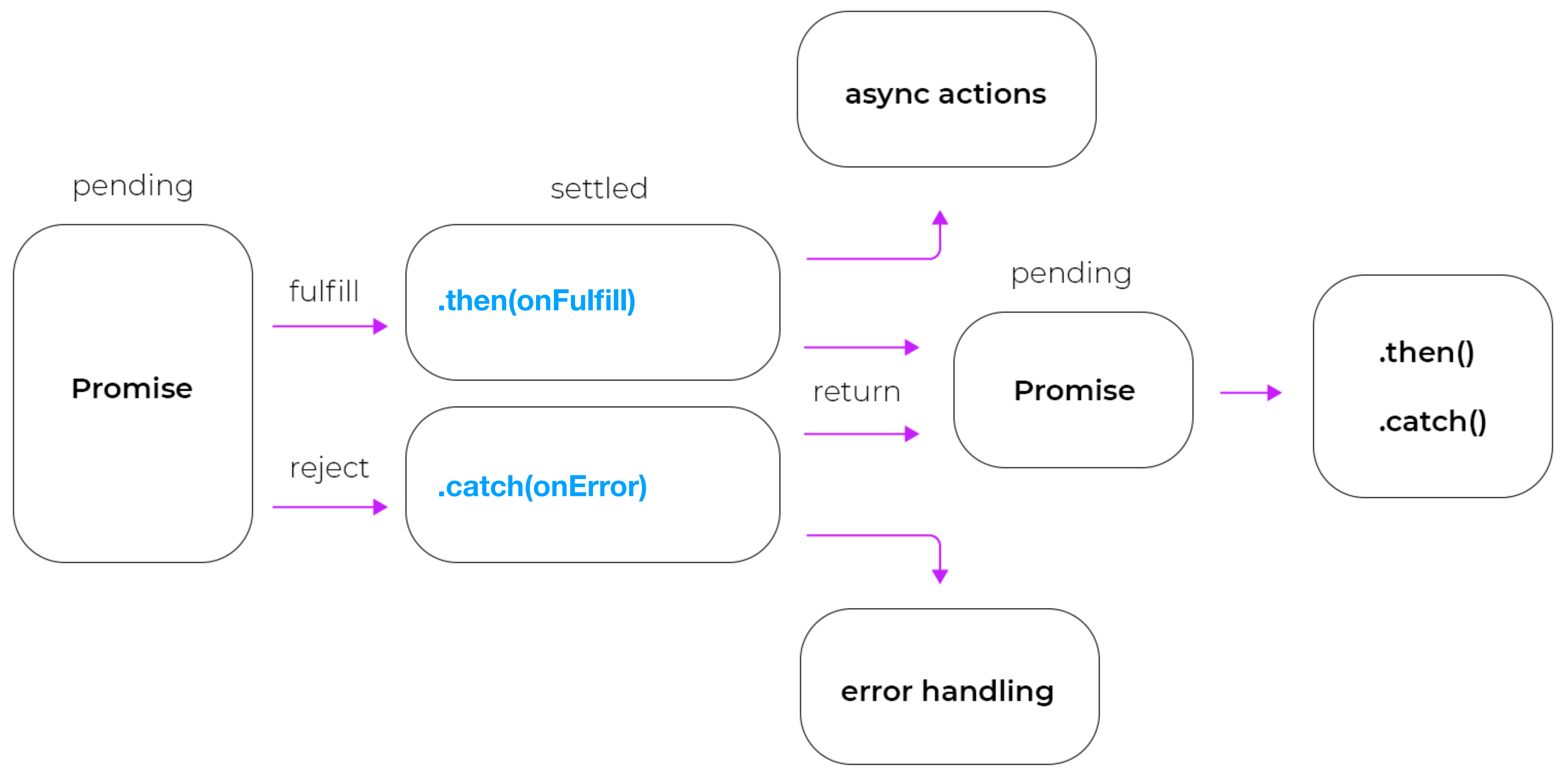


Promise Reactions (Chaining)

.then and .catch return promises themselves, so we can chain reactions:

```
const myPromise = new Promise((resolve, reject) => {
   resolve(777);
});

// At this point, "myPromise" is already settled.
myPromise.then((val) => {
   // Do something with val
   return val * 2;
}).then((val) => {
   // Do something else with val
}).catch((err) => {
   // handle any errors occurred in this chain.
})
```



Promise reaction handlers

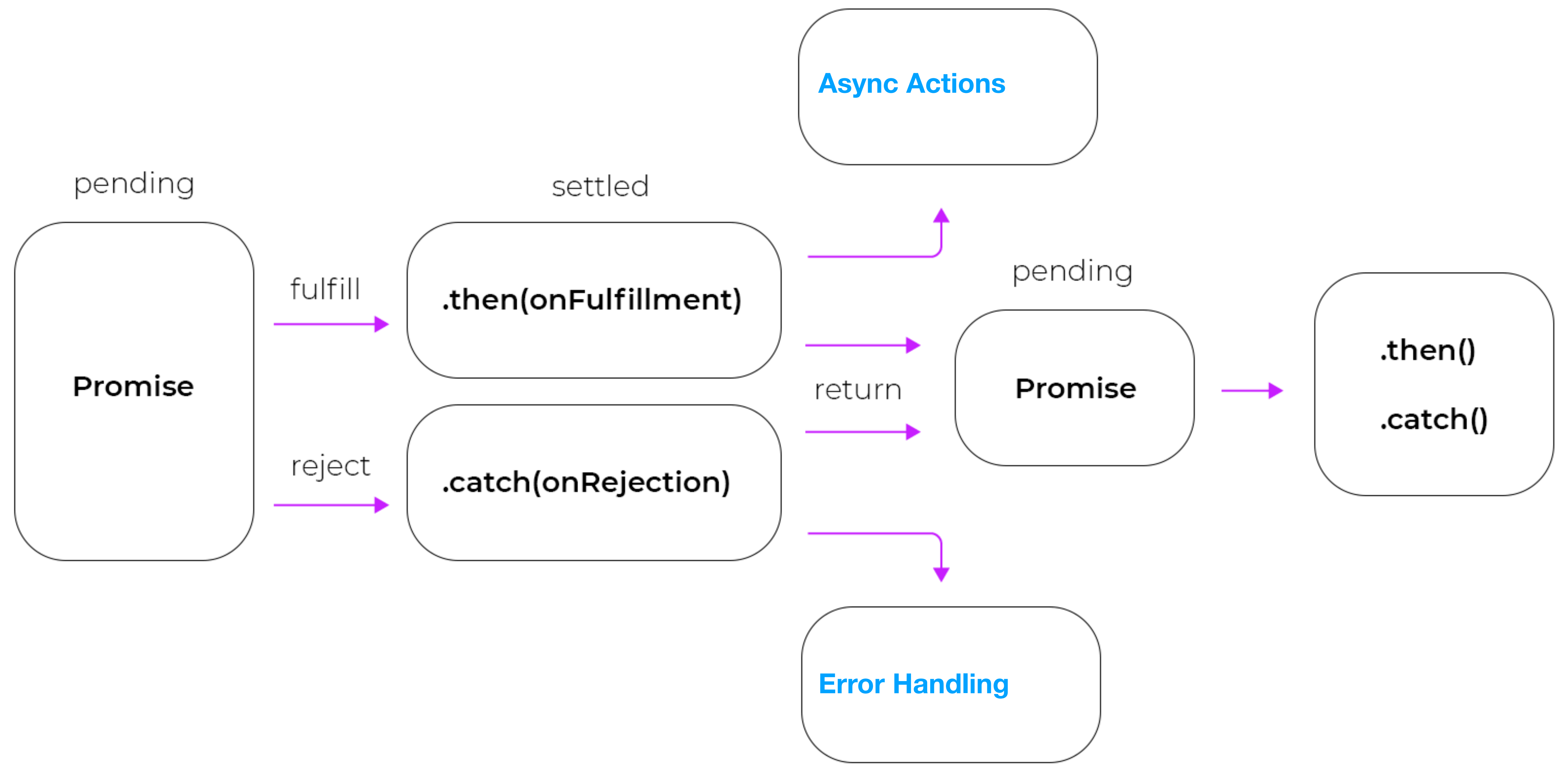
```
const myPromise = new Promise((resolve, reject) => {
   resolve(777);
});

myPromise.then((val) => {
   Return val * 2;
});

myPromise.catch((err) => {
   console.error(err);
})
```

Async Action

Error handling



Wrap up

- Code Coverage
- Javascript Promises
- async/await
- Asynchronous code coverage

?