sourcemind

JAVASCRIPT BASICS

ASYNCHRONOUS JAVASCRIPT

PROMISES, SCHEDULING

Scheduling

- `setTimeout` allows us to run a function once after the interval of time.
- `setInterval` allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

```
function sayHi() {
  alert('Hello');
}

setTimeout(sayHi, 1000);
```

Pass a function, but don't run it

Novice developers sometimes make a mistake by adding brackets `()` after the function:

```
// wrong!
setTimeout(sayHi(), 1000);
```

```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

setInterval

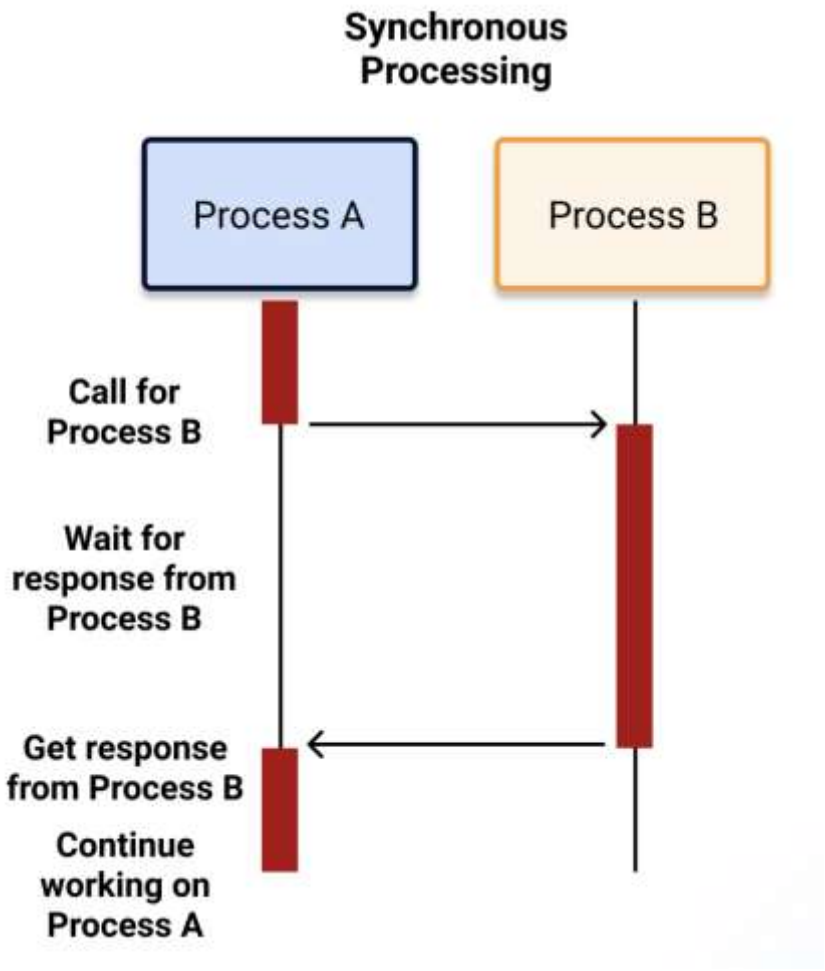
The `setInterval` method has the same syntax as `setTimeout`:

```
let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

To stop further calls, we should call `clearInterval(timerId)`.

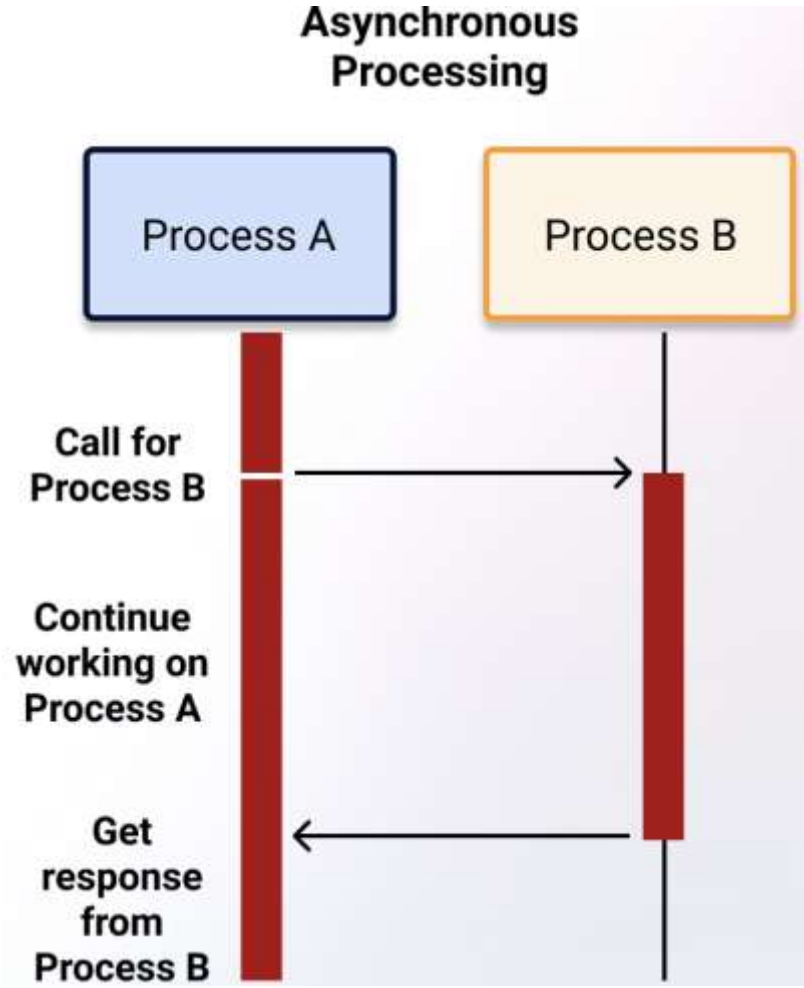
Tasks

1. Write a function printNumbers(from, to) that outputs a number every second, starting from from and ending with to.
2. Create StopWatch (with start, stop, pause buttons);

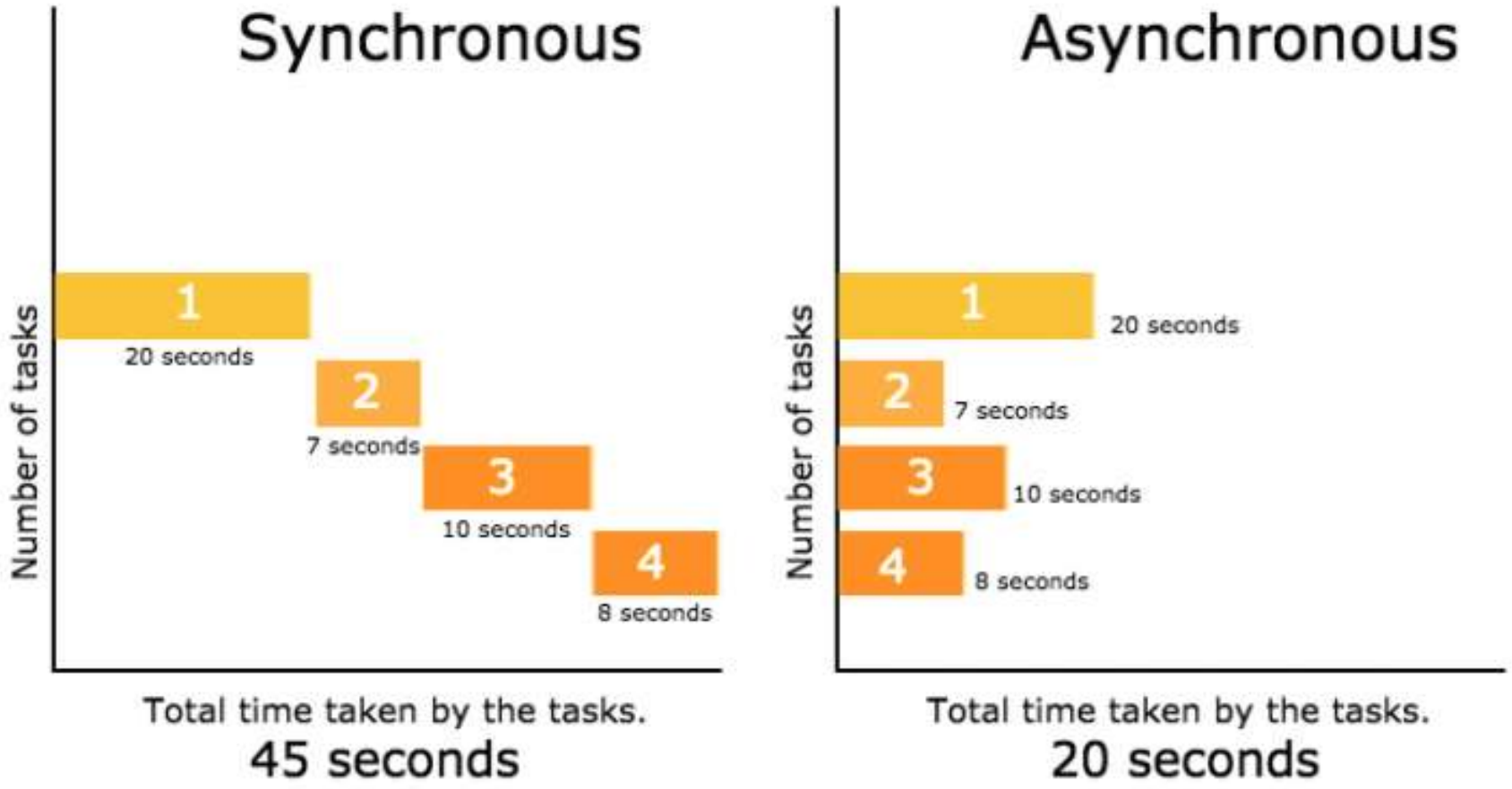


JavaScript Thread is **single-threaded by default used to run a script in the web application, perform layout and garbage collection**.

Being single-threaded is to execute only a single set of instructions at any time in the process.



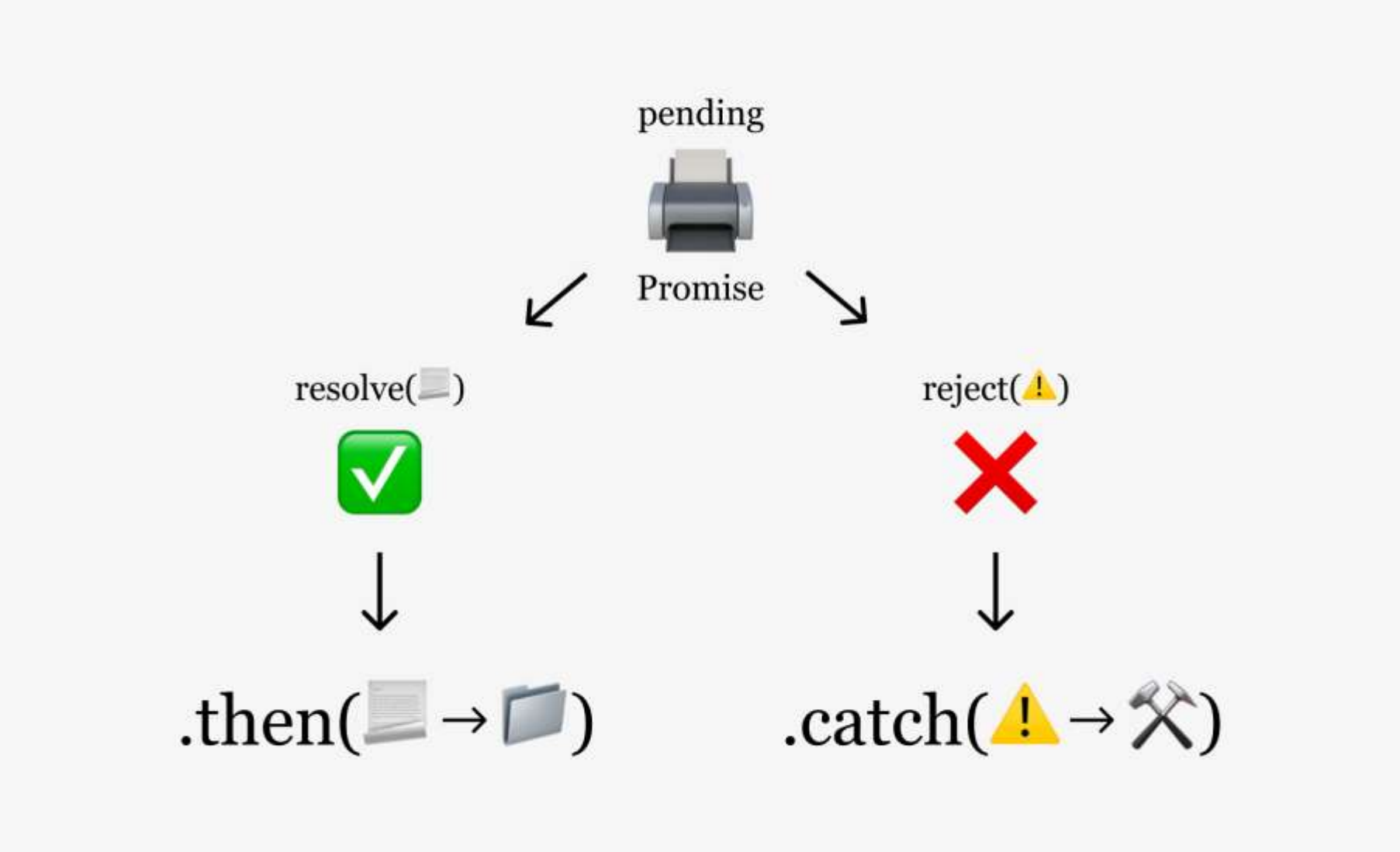
The picture can't be displayed.



JavaScript is a synchronous, blocking, single-threaded language.
That just means that only one operation can be in progress at a time.

The picture can't be displayed.

Promise



The picture can't be displayed.

Promise Syntax

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

myPromise.state	myPromise.result
"pending"	undefined
"fulfilled"	a result value
"rejected"	an error object

The picture can't be displayed.

Example

```
const getData = new Promise(function(resolve, reject) {
        if (Math.random() > 0.5) {
            setTimeout(() => resolve("Data loaded"), 1000);
        } else {
            setTimeout(() => reject("Data loading failed"), 1000);
        }
});

getData
    .then((data) => console.log(data))
    .catch(err => console.log(err));
```

Promise Chaining

```
let p = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve(10);
    }, 3 * 100);
});

p.then((result) => {
    console.log(result); // 10
    return result * 2;
}).then((result) => {
    console.log(result); // 20
    return result * 3;
}).then((result) => {
    console.log(result); // 60
    return result * 4;
});
```

```
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then((values) => {
  console.log(values);
});
// expected output: Array [3, 42, "foo"]
```

Promise.all()

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(resolve, 500, 'one');
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'two');
});

Promise.race([promise1, promise2]).then((value) => {
  console.log(value);
  // Both resolve, but promise2 is faster
});
// expected output: "two"
```

Promise.race()

Async/Await

```
let value = await promise;
```

The `await` keyword can only be used inside an `async` function.

```
async function myAsyncFunction() {
    try {
        const data  = await getData;
        console.log(data);
    } catch(error) {
        console.error(error)
    }
}

myAsyncFunction();
```

Tasks

1. The function job must return a promise object .The promise must resolve itself 2 seconds after the call to job and must provide hello world in the data
2. Create function which receive a data and returns a Promise
   2.1. If data is not a number, return a promise rejected instantly and give the d ata "error" (in a string).
   2.2. If data is an odd number, return a promise resolved 1 second later and give the data "odd" (in a string).
   2.3. If data is an even number, return a promise rejected 2 seconds later and give the data "even" (in a string).
3. Create async `order()` for CoffeShop class