<epam>

Async Programming in JavaScript World

Kamil Armatys && Fatih Erikli

Who are we?

JavaScript developer focused mainly on Node.JS, an Adobe Brackets extension contributor. Epamer from almost 2 years.

– Kamil Armatys

Python and Javascript developer focused on React.JS on front-end, and Django on back-end. Epamer for more than 2 years.

– Fatih Erikli

Introduction

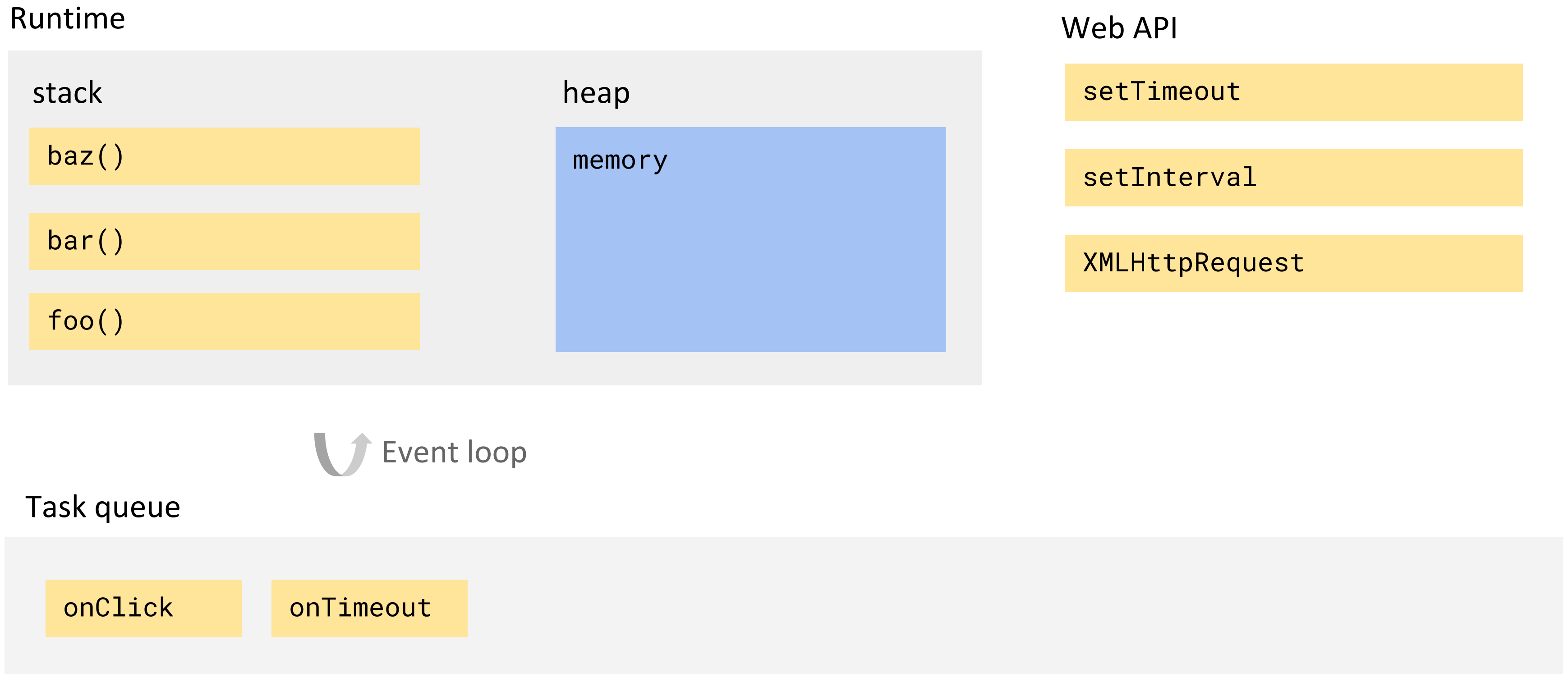
JavaScript is a **single-threaded** and **non-blocking asynchronous concurrent** programming language.

single thread === one thing at a time

Introduction

```
console.log('Hello');

setTimeout(() => console.log('timeout!'), 1000);

console.log('World!');
```

Event loop


Runtime
stack
baz()
bar()
foo()
heap
memory
Web API
setTimeout
setInterval
XMLHttpRequest
Event loop
Task queue
onClick
onTimeout

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script
Stack	script
Console	

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script
Stack	script
Console	Hello

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script
Stack	script
Console	Hello

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script setTimeout Callback
Stack	script
Console	Hello

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script setTimeout Callback
Stack	script
Console	Hello

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script	setTimeout Callback
Stack	script	
Console	Hello	World!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	run script setTimeout Callback
Stack	
Console	Hello World!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	setTimeout Callback
Stack	
Console	Hello World!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	setTimeout Callback
Stack	setTimeout Callback
Console	Hello World!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	setTimeout Callback
Stack	setTimeout Callback
Console	Hello World! timeout!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	setTimeout Callback
Stack	
Console	Hello World! timeout!

Event loop

```
console.log('Hello');
setTimeout(() => {
   console.log('timeout!');
}, 1000);
console.log('World!');
```

Task queue	
Stack	
Console	Hello World! timeout!

Event loop - sync and async callbacks

```
// Synchronous
[1,2,3,4].forEach((i) => {
   console.log(i);
});
```

```
// Asynchronous
setTimeout(() => {
   console.log('time is up');
}, 0);
```

Event loop - what will be the console output?

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

(Macro)Taks and Microtasks

Macro and Micro tasks - mutation observer

```
const node = qureySelector('#app');

// Listen for attribute changes on the
// outer element
new MutationObserver(function() {
   console.log('mutate');
}).observe(node, {
   attributes: true
});
```

Macro and Micro tasks

Task

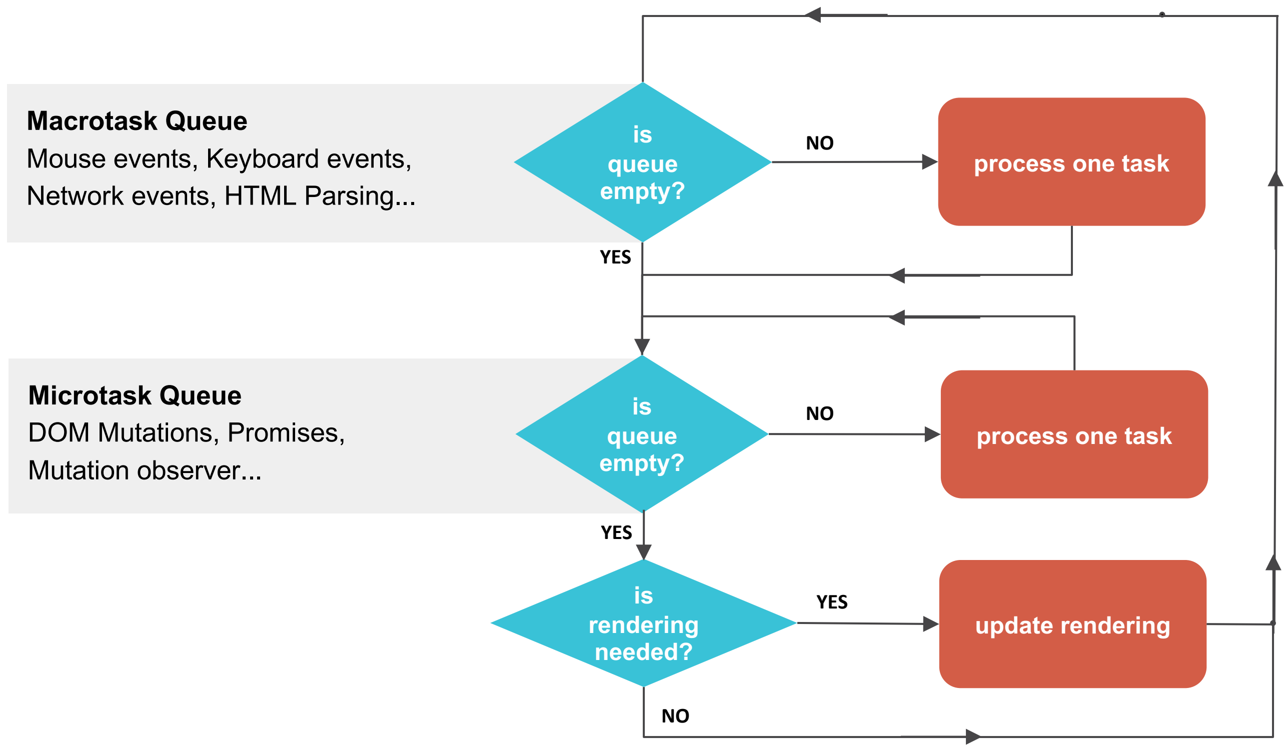
Macrotask

- setTimeout()
- setInterval()
- XMLHttpRequest.send()

Microtask

- Promise.then()
- MutationObserver()
- nextTick()

Macro and Micro tasks - the execution order


Macrotask Queue
Mouse events, Keyboard events,
Network events, HTML Parsing...

Microtask Queue
DOM Mutations, Promises,
Mutation observer...

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script
Microtask	
Stack	script
Console	

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
    console.log('setTimeout');
}, 0);

Promise.resolve()
    .then(() => {
        console.log('promise1');
    })
    .then(() => {
        console.log('promise2');
    });

console.log('end');
```

Task	run script
Microtask	
Stack	script
Console	start

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script
Microtask	
Stack	script
Console	start

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
    console.log('setTimeout');
}, 0);

Promise.resolve()
    .then(() => {
        console.log('promise1');
    })
    .then(() => {
        console.log('promise2');
    });

console.log('end');
```

Task	run script	setTimeout callback
Microtask		
Stack	script	
Console	start	

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script	setTimeout callback
Microtask		
Stack	script	
Console	start	

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
    console.log('setTimeout');
}, 0);

Promise.resolve()
    .then(() => {
        console.log('promise1');
    })
    .then(() => {
        console.log('promise2');
    });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	
Stack	script
Console	start

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script setTimeout callback
Microtask	Promise then
Stack	script
Console	start

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script	setTimeout callback
Microtask	Promise then	
Stack	script	
Console	start	

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	Promise then
Stack	script
Console	start · end

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	Promise then
Stack	
Console	start · end

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	Promise then
Stack	
Console	start · end

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script setTimeout callback
Microtask	Promise then
Stack	Promise callback
Console	start end

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script	setTimeout callback	
Microtask	Promise then		
Stack	Promise callback		
Console	start	end	promise1

Macro and Micro tasks - example

```javascript
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script	setTimeout callback	
Microtask	Promise then	Promise then	
Stack	Promise callback		
Console	start	end	promise1

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script	setTimeout callback	
Microtask	Promise then		
Stack			
Console	start	end	promise1

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	Promise then
Stack	Promise callback
Console	start · end · promise1

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script / setTimeout callback
Microtask	Promise then
Stack	Promise callback
Console	start / end / promise1 / promise2

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	run script · setTimeout callback
Microtask	
Stack	
Console	start · end · promise1 · promise2

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	setTimeout callback
Microtask	
Stack	
Console	start end promise1 promise2

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
    console.log('setTimeout');
}, 0);

Promise.resolve()
    .then(() => {
        console.log('promise1');
    })
    .then(() => {
        console.log('promise2');
    });

console.log('end');
```

Task	setTimeout callback			
Microtask				
Stack	setTimeoutCallback			
Console	start	end	promise1	promise2

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
    console.log('setTimeout');
}, 0);

Promise.resolve()
    .then(() => {
        console.log('promise1');
    })
    .then(() => {
        console.log('promise2');
    });

console.log('end');
```

Task	setTimeout callback
Microtask	
Stack	setTimeoutCallback
Console	start end promise1 promise2 setTimeout

Macro and Micro tasks - example

```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	setTimeout callback
Microtask	
Stack	
Console	start end promise1 promise2 setTimeout

Macro and Micro tasks - example

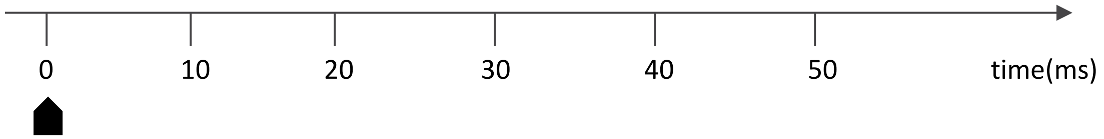
```
console.log('start');

setTimeout(() => {
   console.log('setTimeout');
}, 0);

Promise.resolve()
   .then(() => {
      console.log('promise1');
   })
   .then(() => {
      console.log('promise2');
   });

console.log('end');
```

Task	
Microtask	
Stack	
Console	start end promise1 promise2 setTimeout

(Macro)tasks

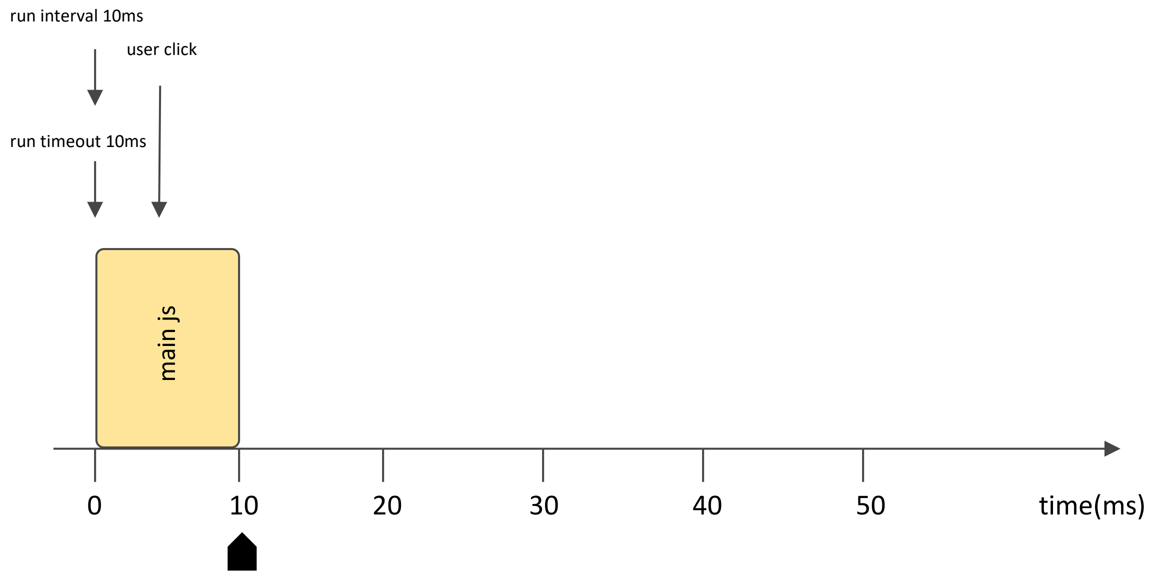
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



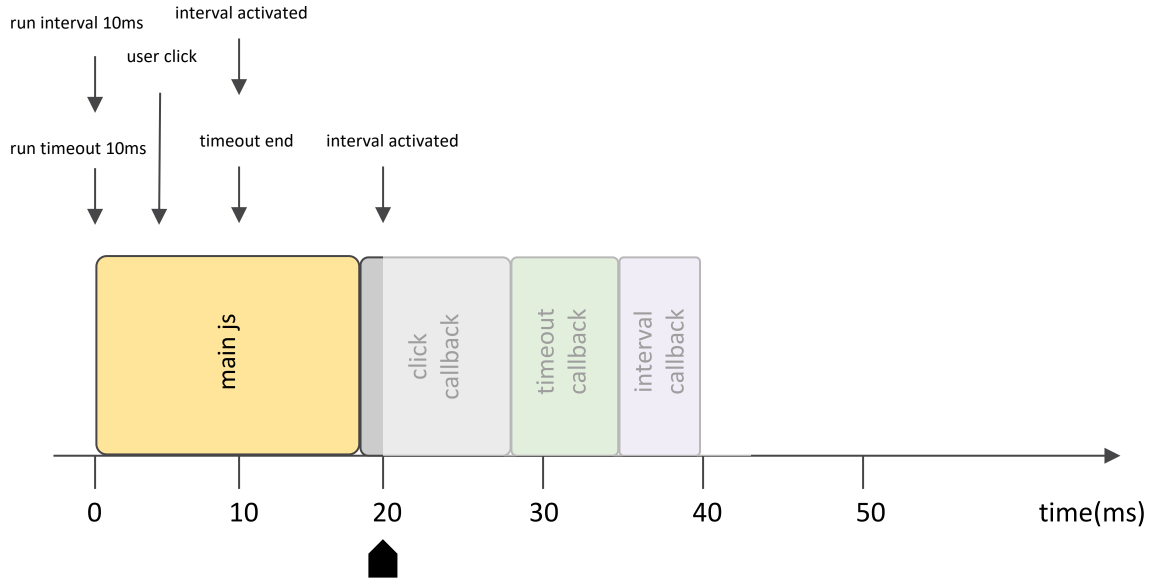
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



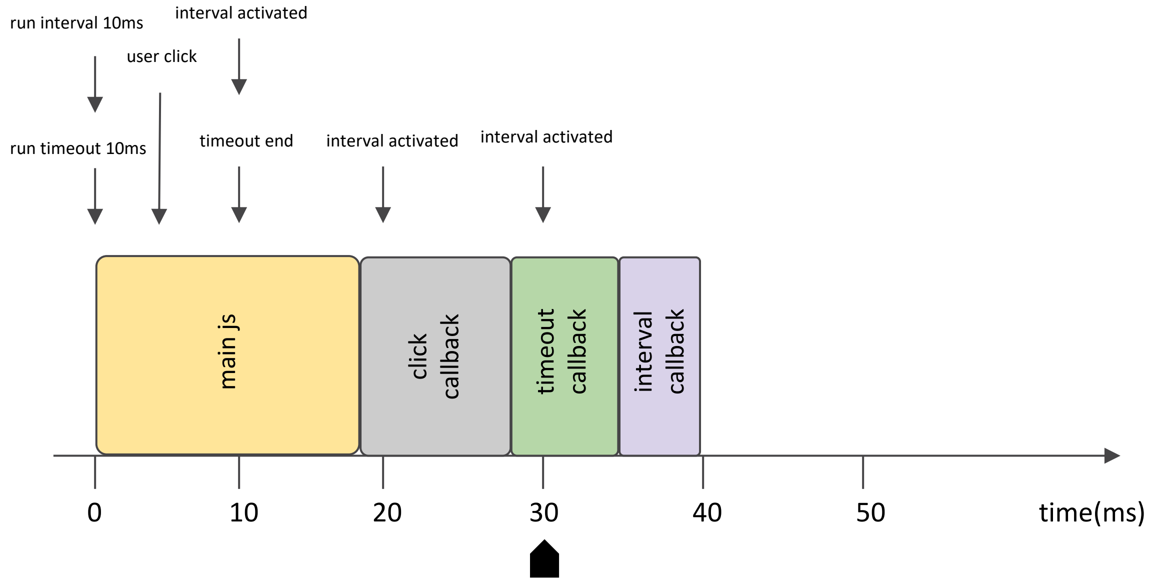
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



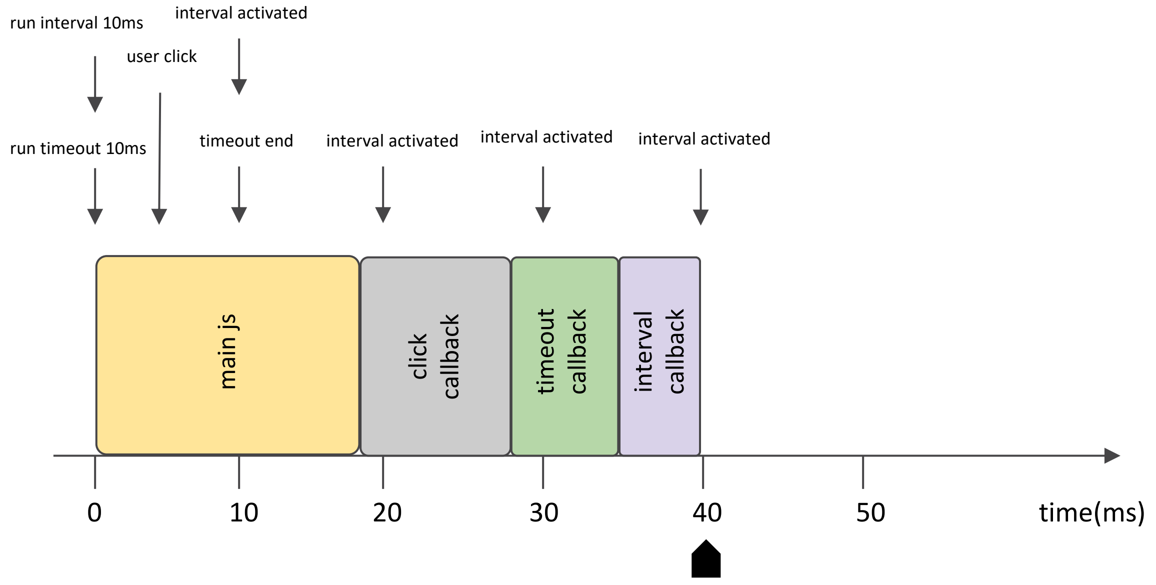
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



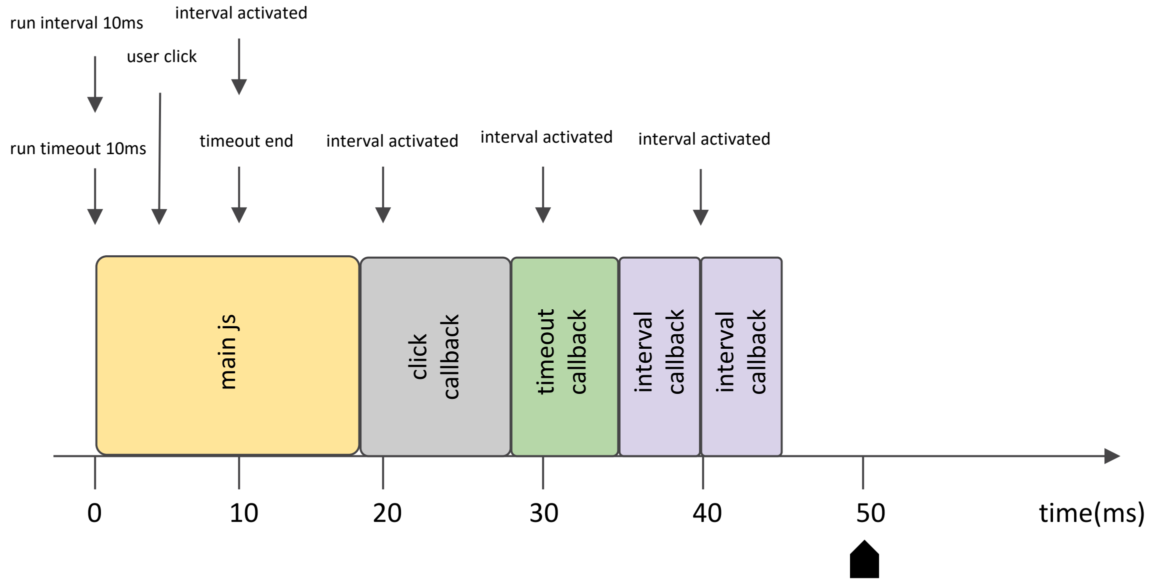
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



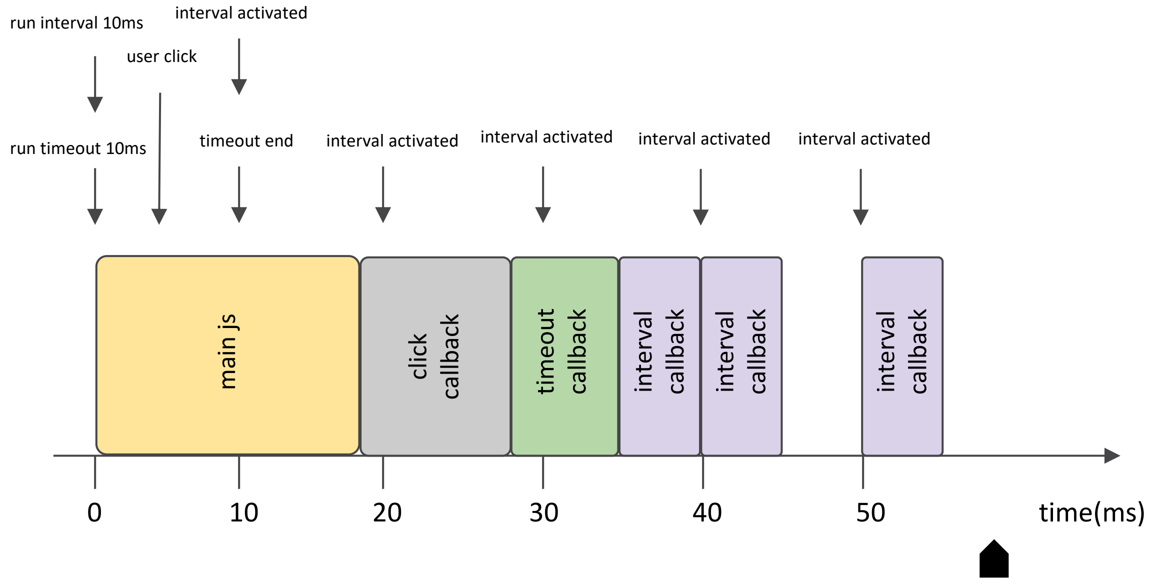
Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



Macrotasks - Timers

```
setTimeout(timeoutCallback, 10);
setInterval(intervalCallback, 10);
doImageProcessing(img); // blocking and long-running process
```



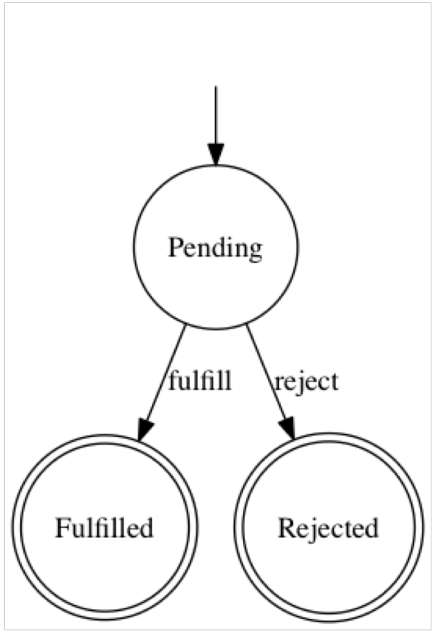
Microtasks

Promise (a.k.a Future) — a brief history

- It’s not a new concept. The term promise was proposed in 1976.
- In javascript world, in the beginning of 2011, jQuery deferred objects which is very close to the concept of Promise became very popular.
- In 2012, Promises were proposed as a spec to standardize the concept.
- Eventually, the promises proposal accepted into the ES 2015 spec and was implemented by major browsers and Node.js.

How does it look like?

```
var promise = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});

promise.then(function(value) {
  console.log(value);
  // expected output: "foo"
});

console.log(promise);
// expected output: [object Promise]
```



https://blog.codecentric.de/en/2015/03/cancelable-async-operations-promises-javascript/

Methods to control the async flow: Promise.all

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

Promise.all([promise1, promise2, promise3]).then(function(values) {
    // values:
    // [3, 42, "foo"]
});
```

Methods to control the async flow: Promise.race

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

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

Promise.race([promise1, promise2]).then(function(value) {
  // the value is ‘two’
  // Both resolve, but promise2 is faster
});
```

Language features to maintain the async flow

Generators and Async/Await

Generators

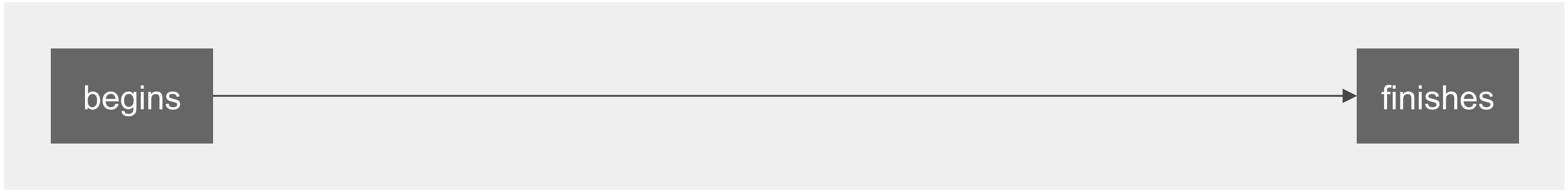
It’s an ES6 feature that allows us to create **stoppable** functions.

```
function regularFunction() {
    console.log('I');
    console.log('am');
    console.log('unstoppable');
}
```

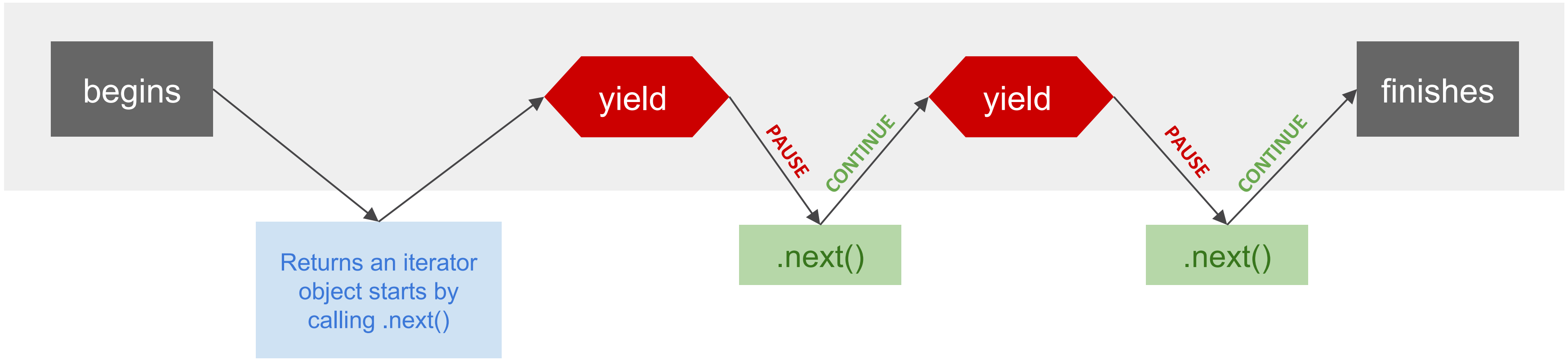
```
function *generatorFunction() {
    yield 'I';
    yield 'am';
    yield 'a';
    yield 'generator';
}
```

Generators

Regular Function



Generator Function



Generators

```
const f = generatorFunction()
f.next()
// {value: "I", done: false}
f.next()
// {value: "am", done: false}
f.next()
// {value: "a", done: false}
f.next()
// {value: "generator", done: false}
f.next()
// {value: undefined, done: true}
```

What if we yield a promise?

A coroutine can be implemented by using generators. This might be very useful with dealing with **callback hell.**

```
function delay(ms) {
  return new Promise(
       resolve => setTimeout(resolve, ms)
  );
}

function *app() {
    console.log('ping');
    yield delay(100);
    console.log('pong');
    yield delay(100);
    console.log('ping');
    yield delay(100);
    console.log('pong');
}
```

What if we yield promise?

We need a runner function that waits until the **yield**ed promise resolved and calls the **next** function again.

This technique is called **“coroutines”** in many programming languages.

```
runner(app)
ping
pong
ping
pong
```

```
function runner(generator) {
    const iterator = generator();

    function run(arg) {
        const result = iterator.next();

        if (result.done) {
            return result.value;
        } else {
            return Promise.resolve(result.value).then(run);
        }
    }

    return run();
}
```

Async/Await = Generators + Promises

What if we yield a promise?

```
function delay(ms) {
  return new Promise(
       resolve => setTimeout(resolve, ms)
  );
}

function *app() {
    console.log('ping');
    yield delay(100);
    console.log('pong');
    yield delay(100);
    console.log('ping');
    yield delay(100);
    console.log('pong');
}
```

```
function delay(ms) {
  return new Promise(
       resolve => setTimeout(resolve, ms)
  );
}

async function app() {
    console.log('ping');
    await delay(100);
    console.log('pong');
    await delay(100);
    console.log('ping');
    await delay(100);
    console.log('pong');
}
```

QUESTIONS?

JOIN US

epam.com/careers