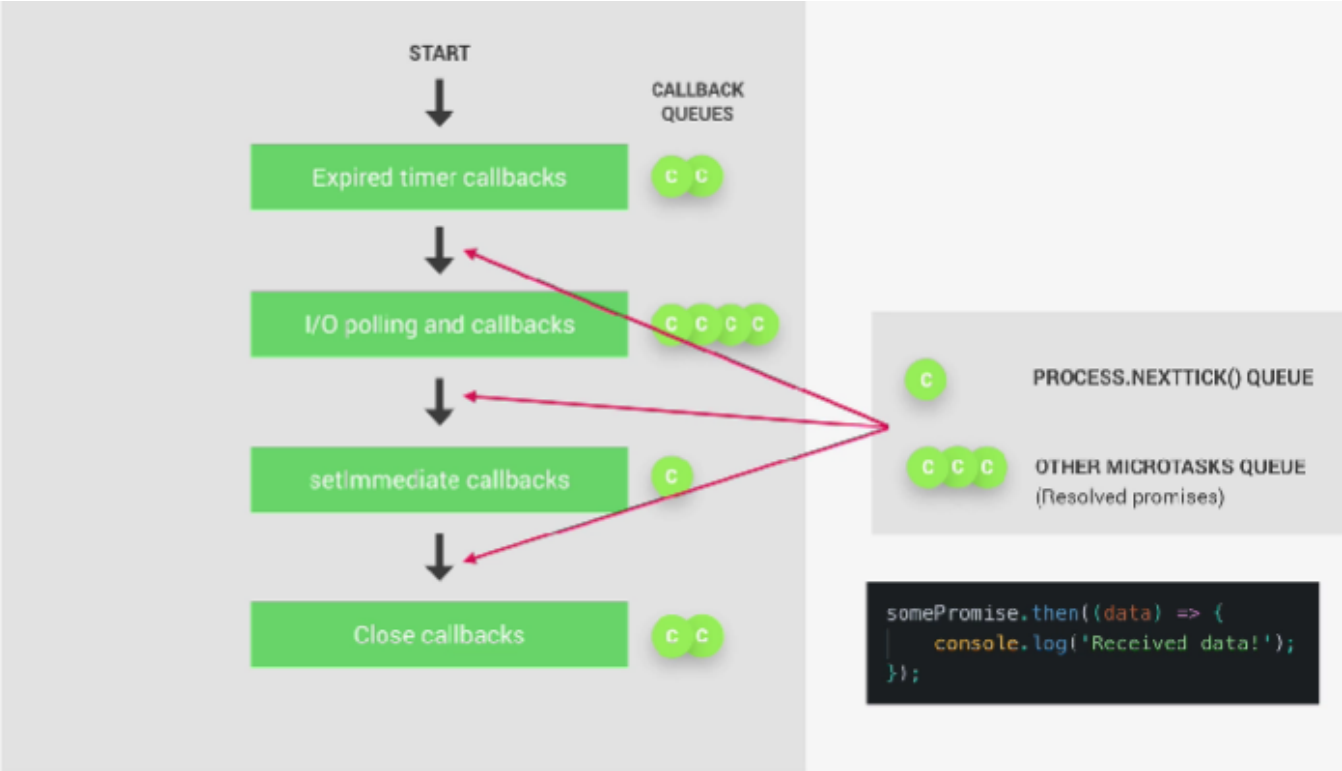
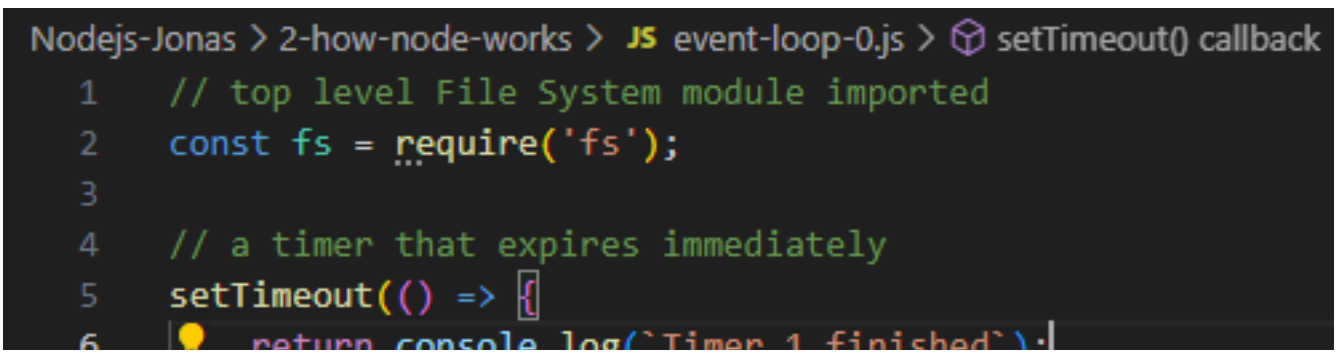
4. Revised - The Event Loop in Practice

Friday, May 24, 2024 9:12 PM




```
// import file system module
const fs = require('fs');
// 0. Start app
// 1. Execute 'top-level' code
// 2. Require module
// 3. Register Event Callbacks
// 4. Start Event Loop
setTimeout(() => console.log('Timer 1 finished'), 0);
setImmediate(() => console.log('Immediate 1 finished'));
fs.readFile('./test-file.txt', () => {
    console.log('I/O finished');
})
```



```
    }, 0);

    // setImmediate = no need any time out cuz Immediate
    setImmediate(() => {
        return console.log(`Immediate 1 finished`);
    });

    // async fs.readFile()
    fs.readFile('text-file.txt', () => {
        console.log(`I/O finished`);
    });
```

Nodejs-Jonas > 2-how-node-works > JS event-loop-0.js > ...

```
1   // top level File System module imported
2   con
3
4   //
5   set
6
7   },
8
9   //
10  set
11
12  });
13
14
15  fs.readFile('text-file.txt');
```

See Real World Examples From GitHub

```
function readFile(path: fs.PathOrFileDescriptor, options: ({
    encoding?: null | undefined;
    flag?: string | undefined;
} & EventEmitter.Abortable) | null | undefined, callback:
(err: NodeJS.ErrnoException | null, data: Buffer) => void):
void (+7 overloads)
namespace readFile
```

Asynchronously reads the entire contents of a file.

```
import { readFile } from 'node:fs';
```

```
[nodemon] starting `node event-loop-0.js`
Timer 1 finished
I/O finished
Immediate 1 finished
[nodemon] clean exit - waiting for changes before restart
```

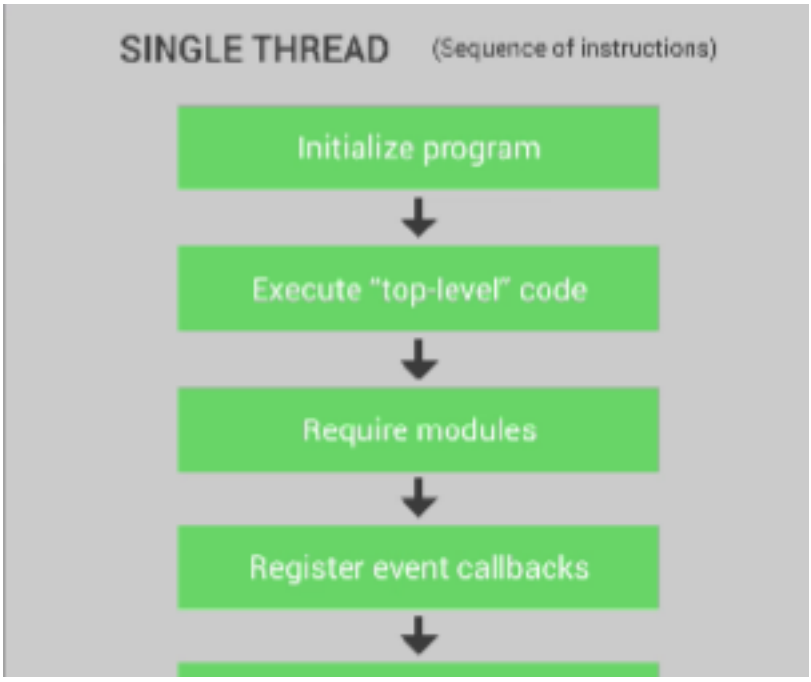
Nodejs-Jonas > 2-how-node-works > JS event-loop-0.js > ...

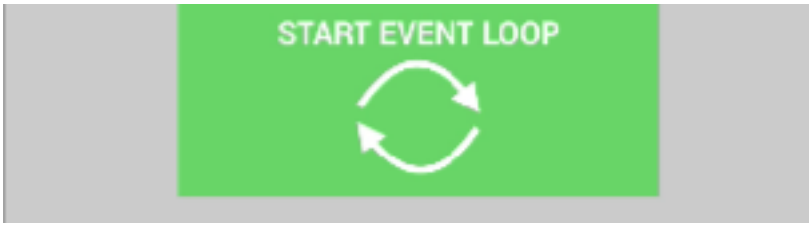
```
1   // top level File System module imported
2   const fs = require('fs');
3
4   // a timer that expires immediately
```

```
5   setTimeout(() => {
6       return console.log(`Timer 1 finished`);
7   }, 0);
8
9   // setImmediate = no need any time out cuz Immediate
10  setImmediate(() => {
11      return console.log(`Immediate 1 finished`);
12  });
13
14  // async fs.readFile()
15  fs.readFile('text-file.txt', () => {
16      console.log(`I/O finished`);
17  });
18
19  // Top level code executes first
20  // even before Event Loop starts
21  console.log(`I'm a Top level code`);
```

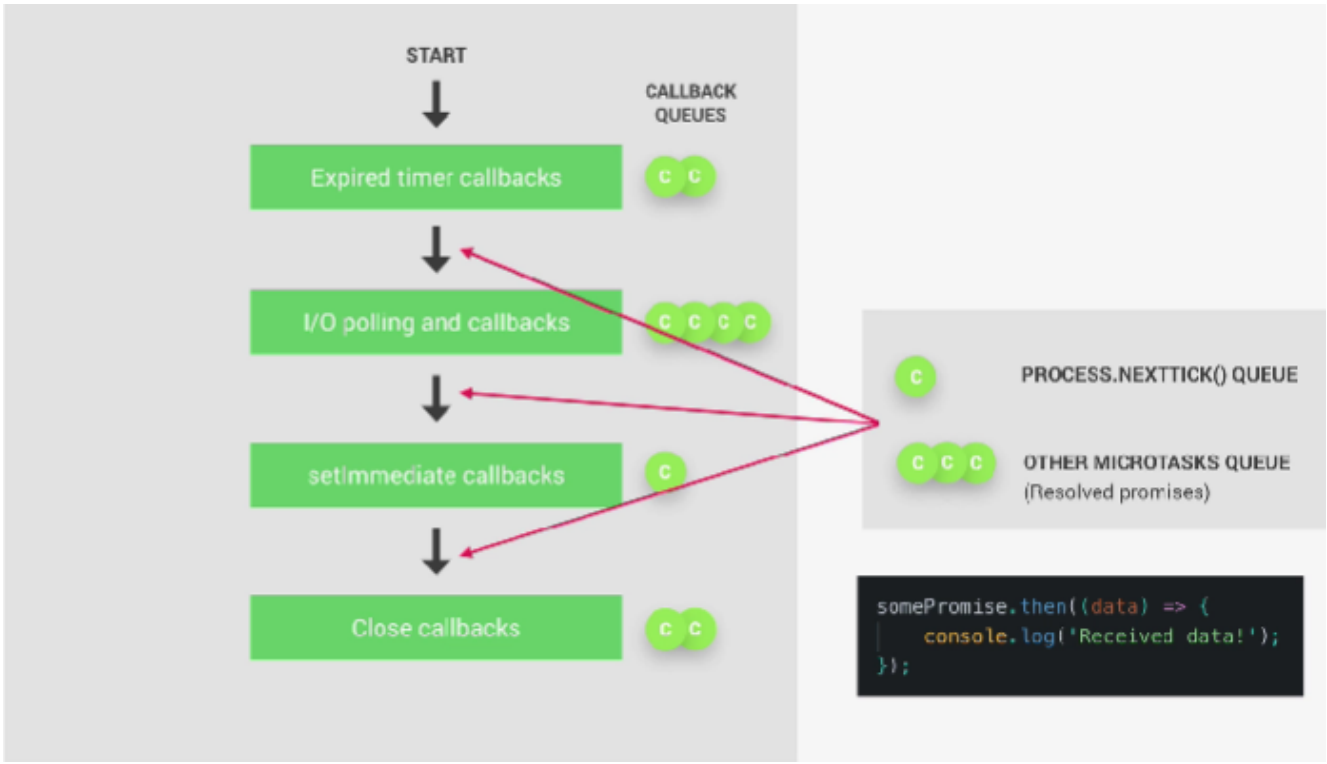
```
[nodemon] starting `node event-loop-0.js`
I'm a Top level code
Timer 1 finished
I/O finished
Immediate 1 finished
[nodemon] clean exit - waiting for changes before restart
```

console.log(`I'm a Top level code`); is NOT inside any callbacks






```
[nodemon] starting `node event-loop-0.js`
I'm a Top level code
Timer 1 finished
I/O finished
Immediate 1 finished
[nodemon] clean exit - waiting for changes before restart
```



Time completion depends heavily on

`Resources needed to for a Callback`

0. Start Node.js
1. Top-level code => console.log(`I'm Top level code`);
2. Require modules => const fs = require('fs');
3. Register callbacks

i. setTimeout(() => console.log(``));

ii. fs.readFile(filePath, () => {});

iii. setImmediate(() => console.log(``));

.

```
,
// ========================
// import file system module
const fs = require('fs');
// 0. Start app
// 1. Execute 'top-level' code
// 2. Require module
// 3. Register Event Callbacks
// 4. Start Event Loop
setTimeout(() => console.log('Timer 1 finished'), 0);
setImmediate(() => console.log('Immediate 1 finished'));
fs.readFile('./test-file.txt', () => {
    console.log('I/O finished');
    setTimeout(() => console.log('Timer 2 finished'), 0);
     // Event Loop finds Timer 3 as pending after I/O finished
    setTimeout(() => console.log('Timer 3 finished'), 3000);

     // When there's no I/O callbacks in the queue
     // Event Loop checks if there's any setImmediate()
     // And execute setImmediate() right away after I/O finished
     // Even before expired timers
    setImmediate(() => console.log('Immediate 2 finished'));
})
console.log('Hello from top-level code');
```

```
1   // top level File System module imported
2   const fs = require('fs');
3
4   // a timer that expires immediately
5   setTimeout(() => {
6       return console.log(`Timer 1 finished`);
7   }, 0);
8
9   // setImmediate = no need any time out cuz Immediate
10  setImmediate(() => {
11      return console.log(`Immediate 1 finished`);
12  });
13
14  // async fs.readFile()
15  fs.readFile('test-file.txt', () => {
16      console.log(`I/O finished`);
17
18      setTimeout(() => {
```

```
        return console.log(`Timer 2 finished`);
    }, 0);

    // setImmediate = no need any time out cuz Immediate
    setImmediate(() => {
        return console.log(`Immediate 2 finished`);
    });
});

// Top level code executes first
// even before Event Loop starts
console.log(`I'm a Top level code`);
```

```
$ node event-loop.js
Hello from top-level code
Timer 1 finished
Immediate 1 finished
I/O finished
Immediate 2 finished
Timer 2 finished
```

Program kept running until Timer3 finished

```
// top level File System module imported
const fs = require('fs');

// a timer that expires immediately
setTimeout(() => {
    return console.log(`Timer 1 finished`);
}, 0);

// setImmediate = no need any time out cuz Immediate
setImmediate(() => {
    return console.log(`Immediate 1 finished`);
});

// async fs.readFile()
fs.readFile('test-file.txt', () => {
    console.log(`I/O finished`);

    setTimeout(() => {
        return console.log(`Timer 2 finished`);
```

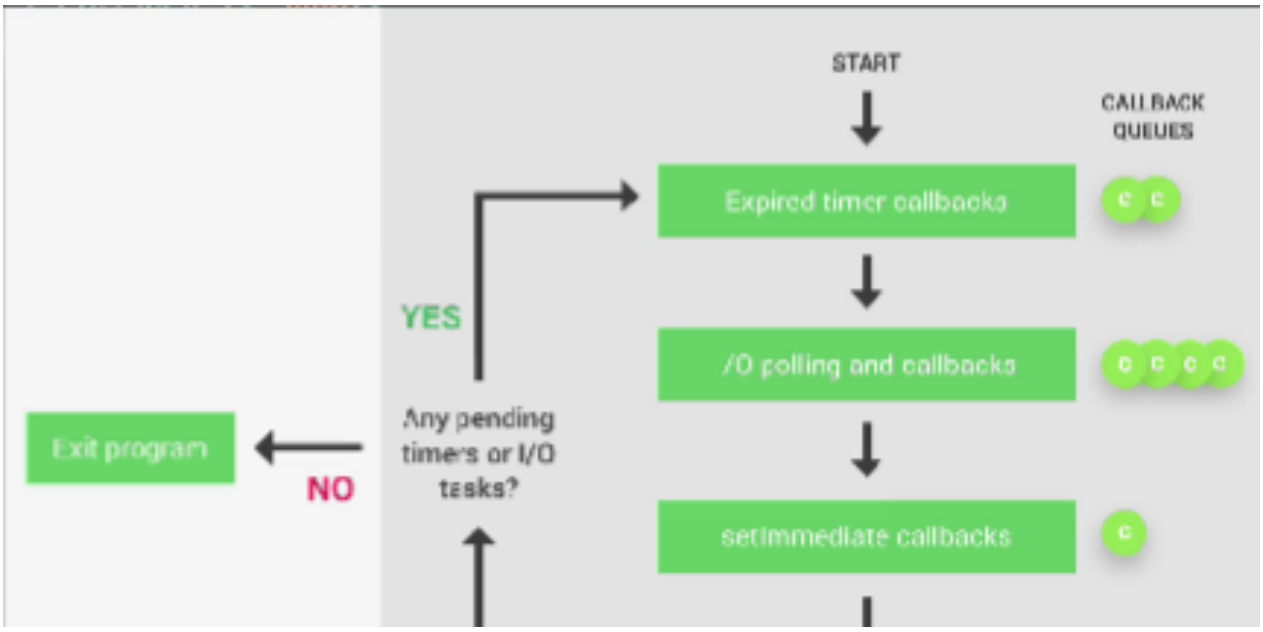
```
    }, 0);

    setTimeout(() => {
        return console.log(`Timer 3 finished`);
    }, 0);

    // setImmediate = no need any time out cuz Immediate
    setImmediate(() => {
        return console.log(`Immediate 2 finished`);
    });
});

// Top level code executes first
// even before Event Loop starts
console.log(`I'm a Top level code`);
```

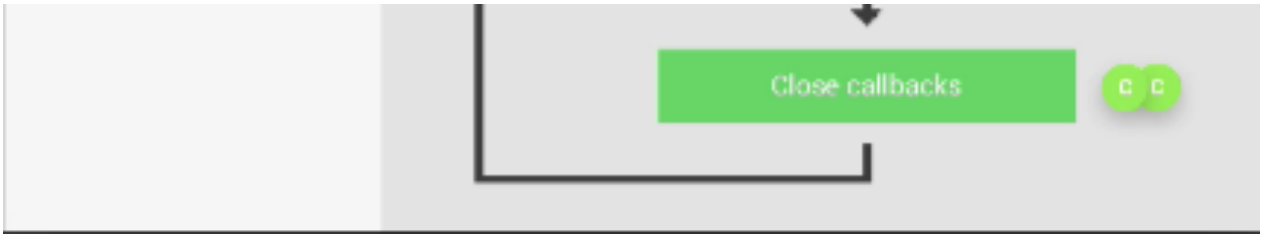
```
$ node event-loop.js
Hello from top-level code
Timer 1 finished
Immediate 1 finished
I/O finished
Immediate 2 finished
Timer 2 finished
Timer 3 finished
```

```
// top level File System module imported
const fs = require('fs');

// a timer that expires immediately
setTimeout(() => {
    return console.log(`Timer 1 finished`);
}, 0);

// setImmediate = no need any time out cuz Immediate
setImmediate(() => {
    return console.log(`Immediate 1 finished`);
});

// async fs.readFile()
fs.readFile('test-file.txt', () => {
    console.log(`I/O finished`);
```

```
17
18      setTimeout(() => {
19          return console.log(`Timer 2 finished`);
20      }, 0);
21
22      setTimeout(() => {
23          return console.log(`Timer 3 finished`);
24      }, 3000);
25
26      // setImmediate = no need any time out cuz Immediate
27      setImmediate(() => {
28          return console.log(`Immediate 2 finished`);
29      });
30  });
31
32  // Top level code executes first
33  // even before Event Loop starts
34  console.log(`I'm a Top level code`);
```

```
[nodemon] starting `node event-loop-0.js`
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Immediate 2 finished
Timer 2 finished
Timer 3 finished
[nodemon] clean exit - waiting for changes before restart
```





1. After all 1st level Expired timer Callbacks

2. I/O Polling Phase starts Inside the 1st I/O callback,

`since Event Loop is inside the Polling Phase, all setImmedaite callbacks are executed right away after Polling Phase`

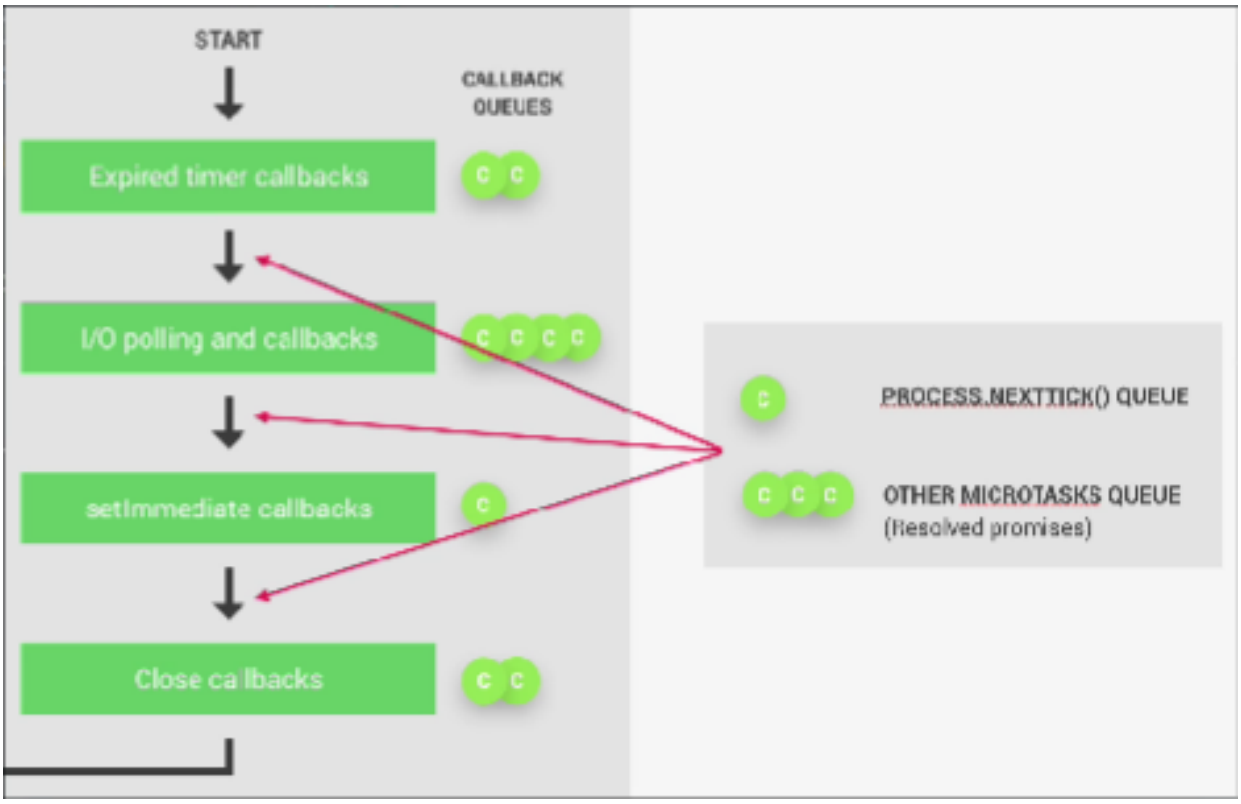
even before Expired Timers setTimeout callbacks

Nodejs-Jonas > 2-how-node-works > JS event-loop-0.js > ...

```
1   // top level File System module imported
2   const fs = require('fs');
3
4   // a timer that expires immediately
5   setTimeout(() => {
6       return console.log(`Timer 1 finished`);
7   }, 0);
8
9   // setImmediate = no need any time out cuz Immediate
10  setImmediate(() => {
11      return console.log(`Immediate 1 finished`);
12  });
13
14  // async fs.readFile()
15  fs.readFile('test-file.txt', () => {
16      console.log(`I/O finished`);
17
18      // Inside I/O Polling Phase, the execution sequence:
19      // 1. process.nextTick()
20      // 2. setImmediate()
21      // 3. setTimeout()
22      setTimeout(() => {
23          return console.log(`Inside fs Polling. Timer 2 finished`);
24      }, 0);
25
26      setTimeout(() => {
27          return console.log(`Inside fs Polling. Timer 3 finished`);
28      }, 3000);
29
30      // setImmediate() goes after process.nextTick()
31      setImmediate(() => {
32          return console.log(`Inside fs Polling. Immediate 2 finished`);
```

```
34
35      // This goes first
36      process.nextTick(() => console.log(`Inside fs Polling. Process.
        nextTick`));
37  });
38
39  // Top level code executes first
40  // even before Event Loop starts
41  console.log(`I'm a Top level code`);
```

```
[nodemon] starting `node event-loop-0.js`
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
[nodemon] clean exit - waiting for changes before restart
```

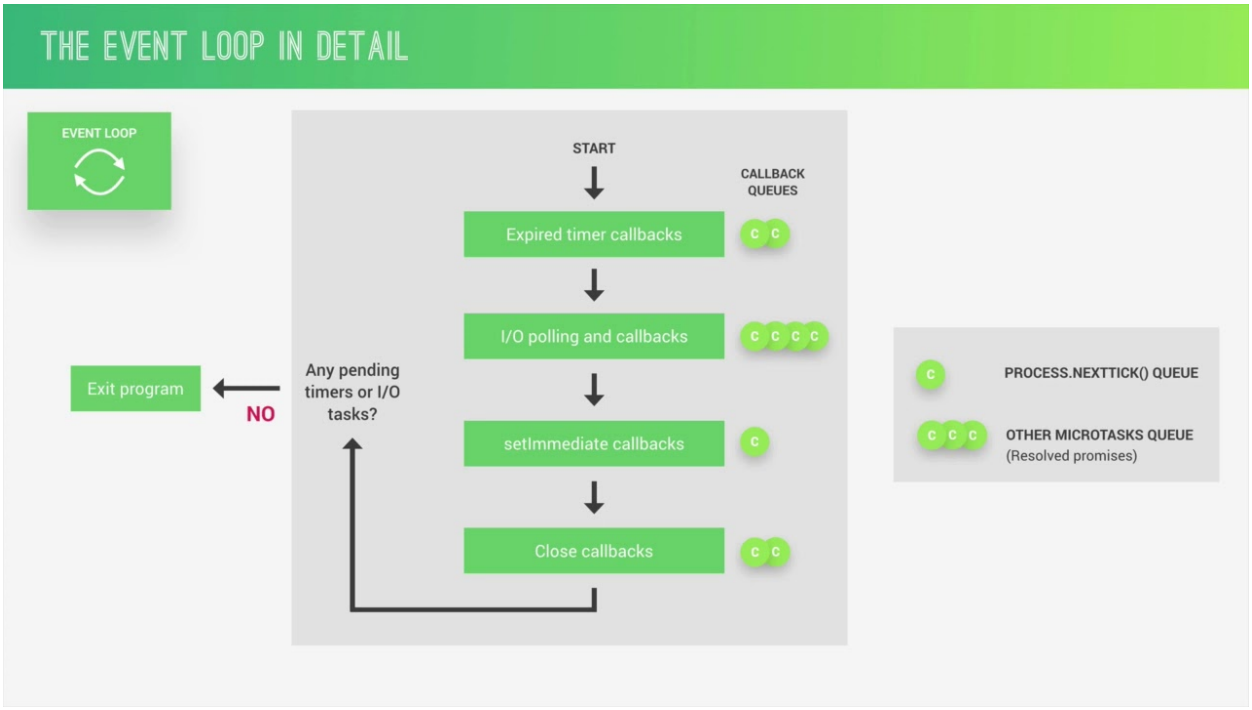


process.nextTick() is 1 of Micro-task queue
that is executed after each phase of:

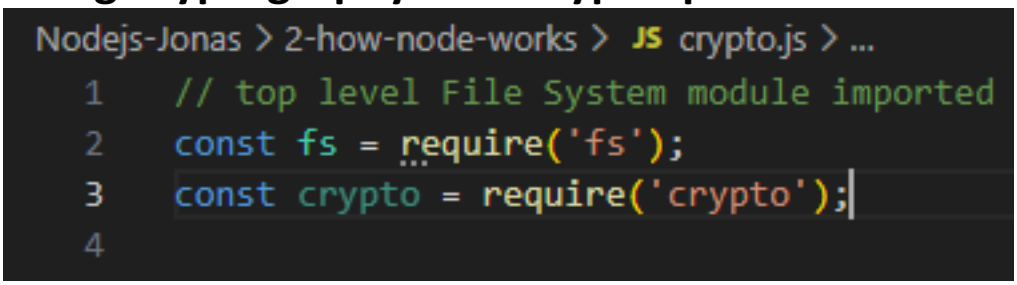
1. Expired timer callbacks
2. I/O Polling and callbacks

3. setImmediate callbacks
4. Close callbacks

**** process.nextTick() happens before the next Loop phase ****



Using Cryptography to encrypt a password



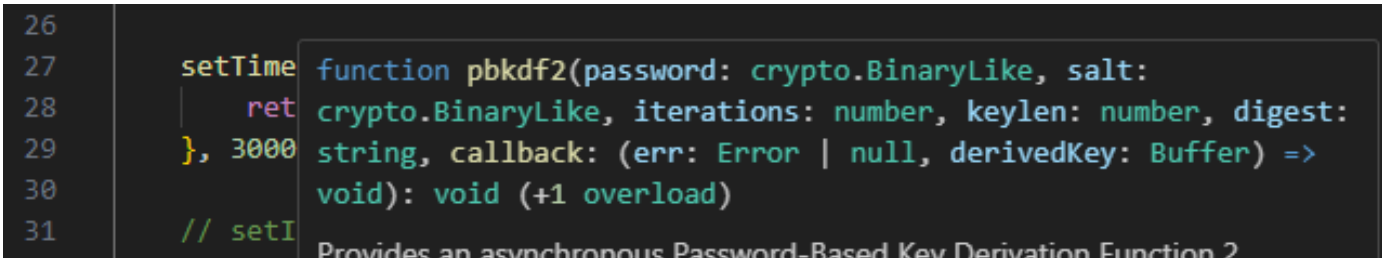
crypto.js

```
// Using Libuv to set Thread Pool Size = 1
// Terminal: export UV_THREADPOOL_SIZE=1
// we'll only have 1 Thread in our Thread Pool
require('dotenv').config();
console.log(`process.env.UV_THREADPOOL_SIZE:\n
${process.env.UV_THREADPOOL_SIZE}`);
// top level File System module imported
const fs = require('fs');
const crypto = require('crypto');
const startTime = Date.now();
```

```
// a timer that expires immediately
setTimeout(() => {
    return console.log(`Timer 1 finished`);
}, 0);
// setImmediate = no need any time out cuz Immediate
setImmediate(() => {
    return console.log(`Immediate 1 finished`);
});
// async fs.readFile()
fs.readFile('test-file.txt', () => {
    console.log(`I/O finished`);
    // Inside I/O Polling Phase, the execution sequence:
    // 1. process.nextTick()
    // 2. setImmediate()
    // 3. setTimeout()
    setTimeout(() => {
        return console.log(`Inside fs Polling. Timer 2
finished`);
    }, 0);
    setTimeout(() => {
        return console.log(`Inside fs Polling. Timer 3
finished`);
    }, 3000);

    // setImmediate() goes after process.nextTick()
    setImmediate(() => {
        return console.log(`Inside fs Polling. Immediate 2
finished`);
    });
    // This goes first
    process.nextTick(() => console.log(`Inside fs Polling.
Process.nextTick`));
    // crypto.pbkdf2('password', 'salt', iterations, keyLength,
digest='typeOfAlgorithm', callback)
    // const password = process.env.PASSWORD;
    const password = 'password';
    crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512',
(err, derivedKey) => {
        if (err) console.log(`Error: ${err}`);
        // Convert Buffer to string
        const encryptedPassword = derivedKey.toString('hex');
        console.log(Date.now() - startTime, `ms taken to encrypt
this Password`);
        //console.log(`Encrypted Password:
${encryptedPassword}`);
```

```
    });
    crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512',
(err, derivedKey) => {
        if (err) console.log(`Error: ${err}`);
        // Convert Buffer to string
        const encryptedPassword = derivedKey.toString('hex');
        console.log(Date.now() - startTime, `ms taken to encrypt
this Password`);
        //console.log(`Encrypted Password:
${encryptedPassword}`);
    });
    crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512',
(err, derivedKey) => {
        if (err) console.log(`Error: ${err}`);
        // Convert Buffer to string
        const encryptedPassword = derivedKey.toString('hex');
        console.log(Date.now() - startTime, `ms taken to encrypt
this Password`);
        //console.log(`Encrypted Password:
${encryptedPassword}`);
    });
    crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512',
(err, derivedKey) => {
        if (err) console.log(`Error: ${err}`);
        // Convert Buffer to string
        const encryptedPassword = derivedKey.toString('hex');
        console.log(Date.now() - startTime, `ms taken to encrypt
this Password`);
        //console.log(`Encrypted Password:
${encryptedPassword}`);
    });
});
// Top level code executes first
// even before Event Loop starts
console.log(`I'm a Top level code`);
```

crypto.pbkdf2 method implementation



```
32      setImme
33          ret
34      });
35
36      // This
37      process
        nextTic
38
39      crypto.pbkdf2('password', 'salt')
40  });
```

(PBKDF2) implementation. A selected HMAC digest algorithm specified by `digest` is applied to derive a key of the requested byte length (`keylen`) from the `password` , `salt` and `iterations` .

The supplied `callback` function is called with two arguments: `err` and `derivedKey` . If an error occurs while deriving the key, `err` will be set; otherwise `err` will be `null` . By default, the successfully

```
17  // async fs.readFile()
18  fs.readFile('test-file.txt', () => {
19      console.log(`I/O finished`);
20
21      // Inside I/O Polling Phase, the execution sequence:
22      // 1. process.nextTick()
23      // 2. setImmediate()
24      // 3. setTimeout()
25      setTimeout(() => {
26          return console.log(`Inside fs Polling. Timer 2 finished`);
27      }, 0);
28
29      setTimeout(() => {
30          return console.log(`Inside fs Polling. Timer 3 finished`);
31      }, 3000);
32
33      // setImmediate() goes after process.nextTick()
34      setImmediate(() => {
35          return console.log(`Inside fs Polling. Immediate 2 finished`);
36      });
37
38      // This goes first
39      process.nextTick(() => console.log(`Inside fs Polling. Process.
        nextTick`));
40
41      // crypto.pbkdf2('password', 'salt', iterations, keyLength,
        digest='typeOfAlgorithm', callback)
42      // const password = process.env.PASSWORD;
43      const password = 'password';
44      crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512', (err,
        derivedKey) => {
45          if (err) console.log(`Error: ${err}`);
46          // Convert Buffer to string
47          const encryptedPassword = derivedKey.toString('hex');
```

```
48          console.log(Date.now() - startTime, `ms taken to encrypt this
            Password`);
49          console.log(`Encrypted Password: ${encryptedPassword}`);
50      });
51  });
52
53  // Top level code executes first
54  // even before Event Loop starts
55  console.log(`I'm a Top level code`);
```

Async nature for crypto.pbkdf2()

Try encrypting a password with SHA-512 =>

```
[nodemon] restarting due to changes...
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
[nodemon] starting `node crypto.js`
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
3164 ms taken to encrypt this Password
Encrypted Password: f5d17022c96af46c0a1dc49a58bbe654a28e98104883e4af4de974cd
a2c74122dd082f4105a93fc80692ca4eb1a784cfeda81bfaa33f5192cc9143d818bd758104bb
2fd0dcfcfe53c1e717bed7069e29fd9cac1f0a483eb7481ca2b76395a4004b8784975561803c
9958e0979ee6deac2beba00983640adc5ccebe6c8ebdf20c66808fc9a394042282083c8f3758
1ae3290f3bad90f4a3888343dac38c7eff4793bfb251c3180750329700da5e9f4d4d5caf9c46
674b4d659ac5cd82e0767189cafdc2ec41684dc60af93e36ba95250f8223e64908bbadc2856a
f0280edbc8893f2d0db41c29b1d31e059ce921c32bcbc33067db9b43ecffdd31e6c6b2f3362b
476914755b1c4349cade2bcbbd0afe971f1cf6a62274ce3741a149cc0d92f9c607d1fa17c555
cab360b51b66293afb4b07ca0d41df47cd5f6596c27a1aafc96053f534e9ef9ffd08e95e5de5
eb4acde76ac3855134b3954e6e2df3808714a71bbb3290e185f115391e6e616c74cda0ea68a7
53ec51cf3452c0965229b944959e2dce7240cf8bc5c201c5409963d56e32e6dbfa00b62a2fd9
0d417ed515d6f0944573b3db12c014d3a113c2f3f82f0fe10f88c7aad119c2502dd99cf896af
1517d20bf047259c423116baf4b5fa0b8c4bc8aa52ada9eb2daf3283badafb5862e26011719c
a7323540e28eb58f24f851fc9b68a582f1308d961cab4b33f92ebf3e705c7fea76c763444363
e15fca3fca8ea580a4a99baf3df6a6872d8ae5cebcab751b3d3c2ee5bd4177ec90c8f8cfd9b9
1900083dc7d821a2abf721144ef12ad40dc44bb2e8d638a7ac03853eb2a24b14024a4bb37cf6
d710f1e6efcf2329ff3765e4fe72454fc66654c11660f6b55085e1fb6f61e3846ed180be3b9f
```

```
3eab18280d7691606e398b4d332878f2a0c013ed659670544bdeeabb4afe3288f9b7392343c5
e64bcbd1f671b9adb53f2f08dcd9be2978db8367b533ed00b7735643fe24274d4c3ce464ce2e
f7a28f9c3a93cb0e8c8ad39f49ceeb5ae4004c8921a981793e84ea9c266ac6e2d2b43fc29bee
b60c35cc903909b9e6ab9f451906c34875c413e9f738f8104906d3db34b7d13473cda6ccfe95
778c90595d177a2de8851300fb176052f2985777d8539ab15028929f8820e1991be1e10154fa
5265d2498c002b2b7662a27e42b13d27d56685a9f1fc538f5f0192fb7fee176f24d6cd58bc24
40f25f9e78e0d13531029d96f3f0fc2a0b3d9ee793f547aa125157445aca28c9dfd6cdb9cbc5
92ff0728bf3dbbcad6cf68576bed50fdc24053fe95ca834fedaaf3836e7a9dda6bc043ee45ac
03cd14eb4ba15168f4b26e4124f77b4b02c8e51837ef81b334d264e1645745df346bc4afb321
0b3c705565f6cbca7b1a7dff5a9dbfc4dc4af6bcb5a49933ca83e771276018d7e33578c95320
57f744a59a3ae1c9
[nodemon] clean exit - waiting for changes before restart
```

Try running 4 instances of crypto.pbkdf2()
inside fs.readFile('test-file.txt', () => {...});

```
18  fs.readFile('test-file.txt', () => {
44      crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512', (err,
        derivedKey) => {
45          if (err) console.log(`Error: ${err}`);
46          // Convert Buffer to string
47          const encryptedPassword = derivedKey.toString('hex');
48          console.log(Date.now() - startTime, `ms taken to encrypt this
            Password`);
49          //console.log(`Encrypted Password: ${encryptedPassword}`);
50      });
51
52      crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512', (err,
        derivedKey) => {
53          if (err) console.log(`Error: ${err}`);
54          // Convert Buffer to string
55          const encryptedPassword = derivedKey.toString('hex');
56          console.log(Date.now() - startTime, `ms taken to encrypt this
            Password`);
57          //console.log(`Encrypted Password: ${encryptedPassword}`);
58      });
59
60      crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512', (err,
        derivedKey) => {
61          if (err) console.log(`Error: ${err}`);
62          // Convert Buffer to string
63          const encryptedPassword = derivedKey.toString('hex');
64          console.log(Date.now() - startTime, `ms taken to encrypt this
            Password`);
65          //console.log(`Encrypted Password: ${encryptedPassword}`);
66      });
67
68      crypto.pbkdf2(password, 'salt', 100000, 1024, 'sha512', (err,
```

```
    derivedKey) => {
69      if (err) console.log(`Error: ${err}`);
70      // Convert Buffer to string
71      const encryptedPassword = derivedKey.toString('hex');
72      console.log(Date.now() - startTime, `ms taken to encrypt this
        Password`);
73      //console.log(`Encrypted Password: ${encryptedPassword}`);
74  });
75 });
```

```
[nodemon] starting `node crypto.js`
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
5212 ms taken to encrypt this Password
5329 ms taken to encrypt this Password
5352 ms taken to encrypt this Password
5558 ms taken to encrypt this Password
[nodemon] clean exit - waiting for changes before restart
```

All 4 password encryption callbacks take
almost the same time to complete
due to async nature & **default UV_THREADPOOL_SIZE=4;**

Assigning Number of Threads in our Thread Pool

Nodejs-Jonas > 2-how-node-works > JS crypto.js > ...

```
1  // Using Libuv to set Thread Pool Size = 1
2  // Terminal: export UV_THREADPOOL_SIZE=1
3  // we'll only have 1 Thread in our Thread Pool
4  require('dotenv').config();
5  console.log(`process.env.UV_THREADPOOL_SIZE:\n${process.env.
   UV_THREADPOOL_SIZE}`);
6
```

Setting Number of Node.js ThreadPool using Terminal

export UV_THREADPOOL_SIZE=4 && npm start;

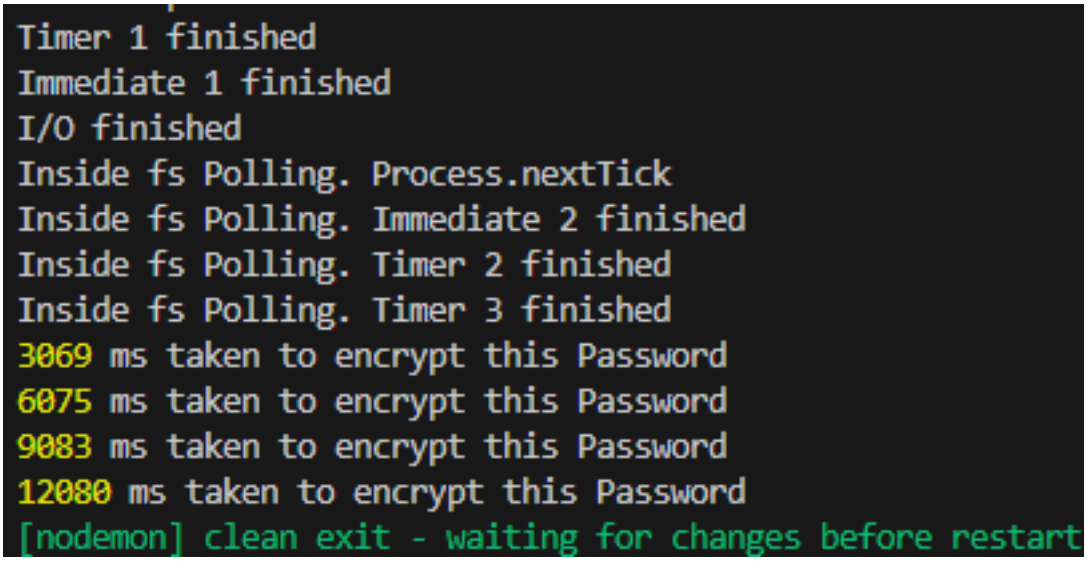
```
[nodemon] starting `node crypto.js`
process.env.UV_THREADPOOL_SIZE:
4
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
4373 ms taken to encrypt this Password
4471 ms taken to encrypt this Password
4601 ms taken to encrypt this Password
4637 ms taken to encrypt this Password
[nodemon] clean exit - waiting for changes before restart
```

export UV_THREADPOOL_SIZE=2 && npm start;

```
[nodemon] starting `node crypto.js`
process.env.UV_THREADPOOL_SIZE:
2
I'm a Top level code
Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
3625 ms taken to encrypt this Password
3656 ms taken to encrypt this Password
7446 ms taken to encrypt this Password
7457 ms taken to encrypt this Password
[nodemon] clean exit - waiting for changes before restart
```

export UV_THREADPOOL_SIZE=1 && npm start;

```
[nodemon] starting `node crypto.js`
process.env.UV_THREADPOOL_SIZE:
1
I'm a Top level code
```


Timer 1 finished
Immediate 1 finished
I/O finished
Inside fs Polling. Process.nextTick
Inside fs Polling. Immediate 2 finished
Inside fs Polling. Timer 2 finished
Inside fs Polling. Timer 3 finished
3069 ms taken to encrypt this Password
6075 ms taken to encrypt this Password
9083 ms taken to encrypt this Password
12080 ms taken to encrypt this Password
[nodemon] clean exit - waiting for changes before restart