In this document, we want show the difference of response time of the different methods for calling a function using the event loop.

There is 4 ways to call a function, 3 queuing this function on the event loop, and directly calling the function. From the documentation, here is a brief explanation of the differences, illustrated in Figure 1.

direct call straightforward.

nextTick The function `process.nextTick` pushes the function passed as an argument to the very beginning of the next loop. That means, the current execution will continue until it ends, then the queued function will be executed before anything else, including *IO Operations*.

setImmediate The function `setImmediate` pushes the function passed as an argument to the next event loop, just after *IO Operations* finished.

setTimeout The function `setTimeout` pushes the function passed as an argument to the next event loop to be executed after a certain delay, specified as an argument. This delay can't be respected, thus even when it's zero, the queued function will be executed after averything else.

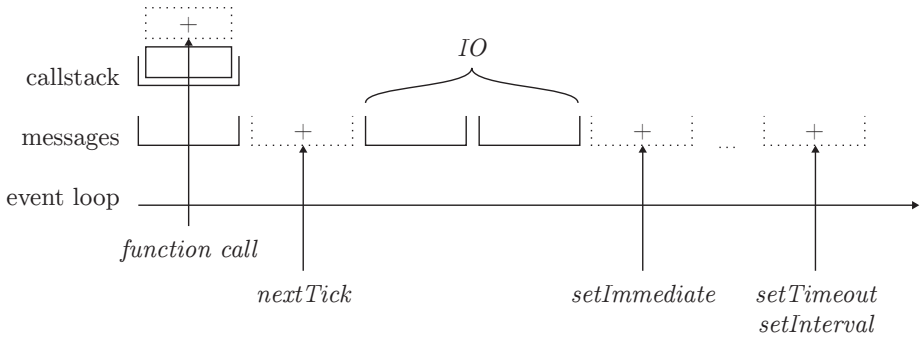


FIGURE 1 – Javascript event loop details

We tested the order of execution of these different queueing method with the listing 1. Even if it's in complete contradiction [1] with the node manual, the order appears to be the following : Results from the execution of listing 1. `0 directCall 1 nextTick 2 setTimeout 3 setImmediate`

```
1  var index = 0;
2
3  function test(name) {
4    console.log((index++) + " " + name);
5  }
6
7  setImmediate(function() {
8    test("setImmediate");
9  })
10
```

1. setImmediate(callback, [arg], [...]) : To schedule the "immediate" execution of callback after I/O events callbacks and before setTimeout ... However, this difference might come from a difference in the implementation, the last documentation version is 0.11.26, while my version of node is 0.10.22

```
11 setTimeout(function() {
12   test("setTimeout");
13 }, 0);
14
15 process.nextTick(function() {
16   test("nextTick");
17 })
18
19 test("directCall");
```

Listing 1 – test order of execution

We tested the difference of run time between these different methods. On Figure 1 we can see the difference between the time spent inside a function, and the time spent between the call and the end of this function.

We base our test on the `wait` function in listing 2. This function was called by different calling function in listings 3, 4, 5 and 6. We probed the time just before the callings functions, and just after exiting the `wait` function.

```
1 function wait(duration, start, _cb) {
2   var iterations = 0;
3   var end = time.now() + duration * 1000;
4   while( time.now() < end ) {
5     iterations++;
6   }
7   return _cb(start, time.now());
8 }
```

Listing 2 – wait function

```
1 function(fn, args) {
2   args[1] = time.now();
3   fn.apply(null, args);
4 }
```

Listing 3 – Direct call

```
1 function(fn, args) {
2   process.nextTick((function(fn, args) {
3     args[1] = time.now();
4     return function() {
5       fn.apply(null, args);
6     }
7   })(fn, args));
8 },
```

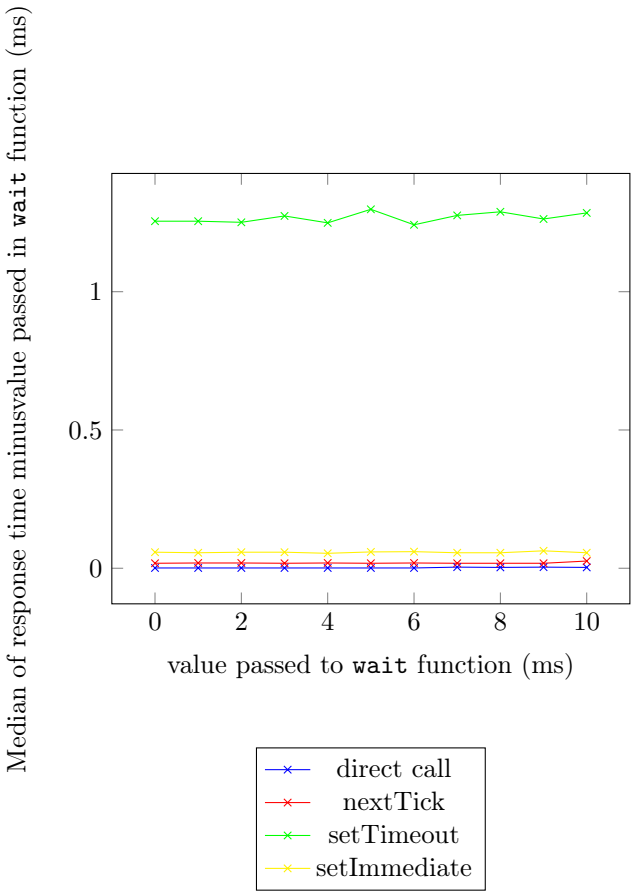
Listing 4 – *nextTick*

```
1 function(fn, args) {
2   args.unshift(0)
3   args.unshift(fn);
4   args[3] = time.now();
5   setTimeout.apply(null, args);
6 }
```

Listing 5 – *setTimeout*

```
1 function(fn, args) {
2   args.unshift(fn);
3   args[2] = time.now();
4   setImmediate.apply(null, args);
5 },
```

Listing 6 – *setImmediate*



On the figure , we can see that while *nextTick* and *setImmediate* are one order of magnitude slower than a direct call, *setTimeout* is about 3 order of magnitude slower than a direct call.

On the , we can see the resulting data of this experiment

directcall	*nextTick*	*setTimeout*	*setImmediate*
0.001	0.018	1.255	0.058
0.0009999999999998899	0.018999999999999906	1.255	0.05600000000000005
0.0009999999999998899	0.019000000000000128	1.251	0.05799999999999983
0.0009999999999998899	0.017999999999999794	1.274	0.05799999999999983
0.001000000000000334	0.019000000000000128	1.2489999999999997	0.05400000000000027
0.001000000000000334	0.017999999999999794	1.298	0.05900000000000016
0.001000000000000334	0.019000000000000128	1.242	0.05999999999999961
0.0039999999999995595	0.017999999999999794	1.2759999999999998	0.05600000000000005
0.0030000000000001137	0.018000000000000682	1.2889999999999997	0.05599999999999916
0.0039999999999995595	0.018000000000000682	1.263	0.06300000000000061
0.0030000000000001137	0.0259999999999998	1.2850000000000001	0.05599999999999916

FIGURE 2 – Raw data of the experiment