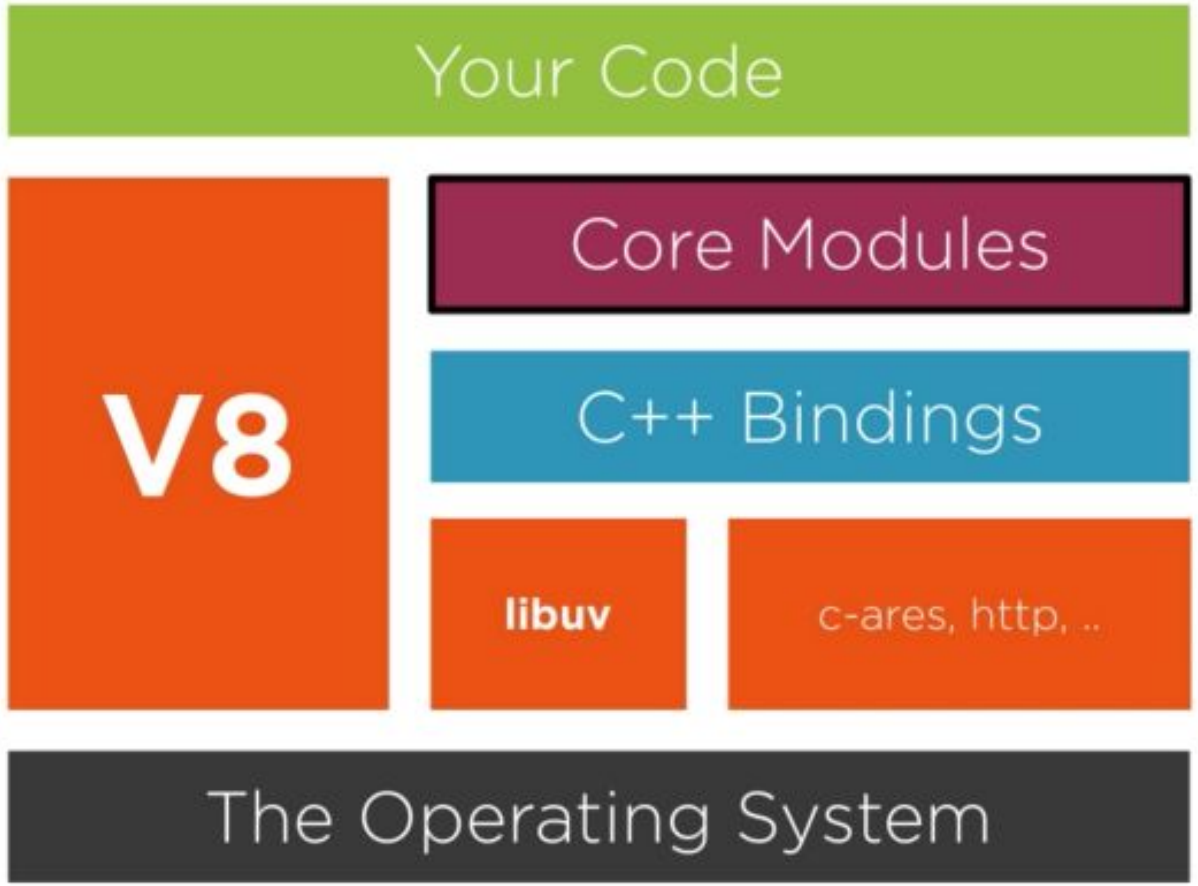
Асинхронность под капотом Node.js

Архитектура Node.js



libuv

Overview

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it's also used by Luvit, Julia, pyuv, and others.

Feature highlights

- Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
- Asynchronous TCP and UDP sockets
- Asynchronous DNS resolution
- Asynchronous file and file system operations
- File system events
- ANSI escape code controlled TTY
- IPC with socket sharing, using Unix domain sockets or named pipes (Windows)
- Child processes
- Thread pool
- Signal handling
- High resolution clock
- Threading and synchronization primitives

https://github.com/libuv/libuv

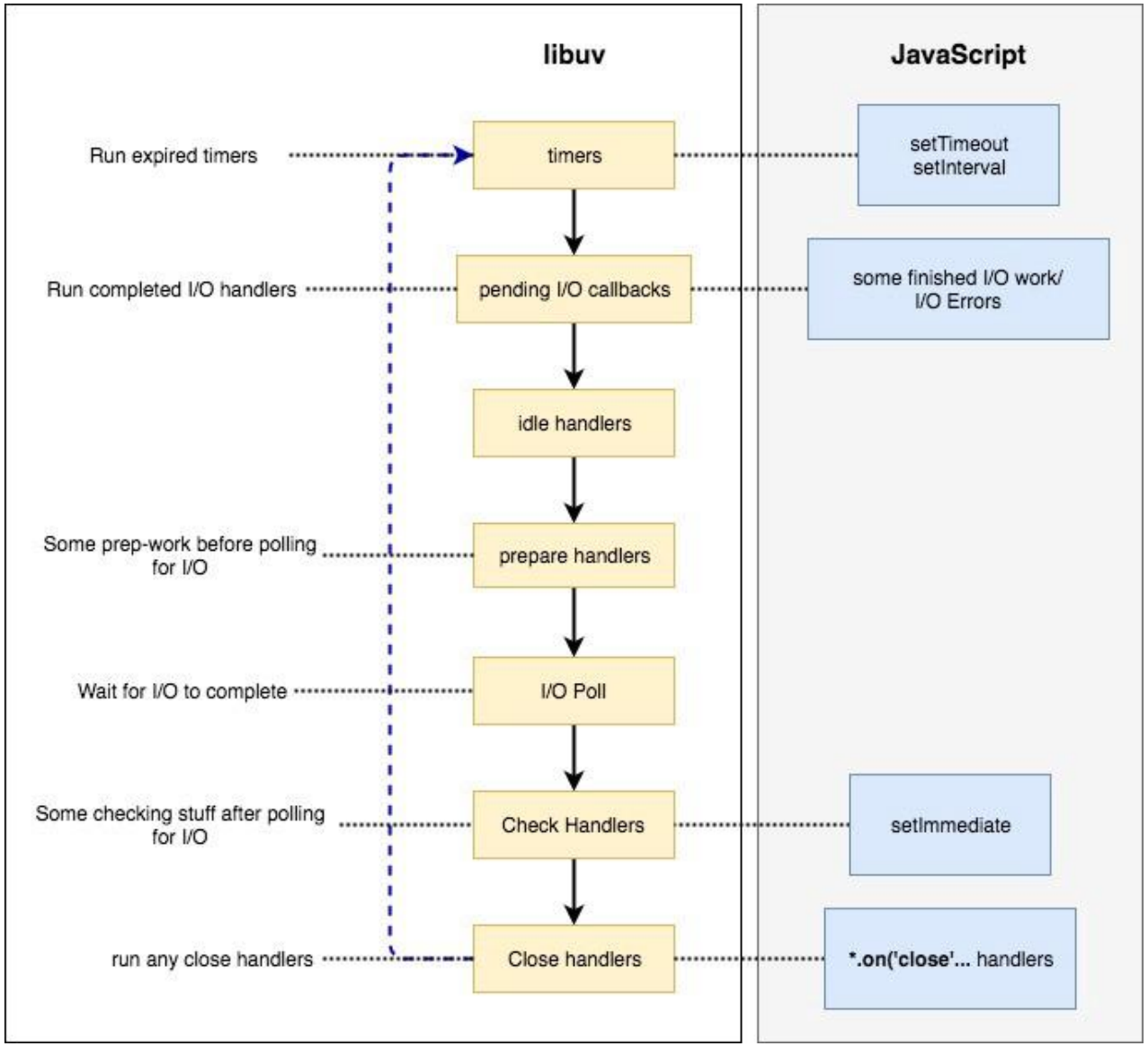
Event loop

What is the Event Loop?

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

Как устроен EventLoop

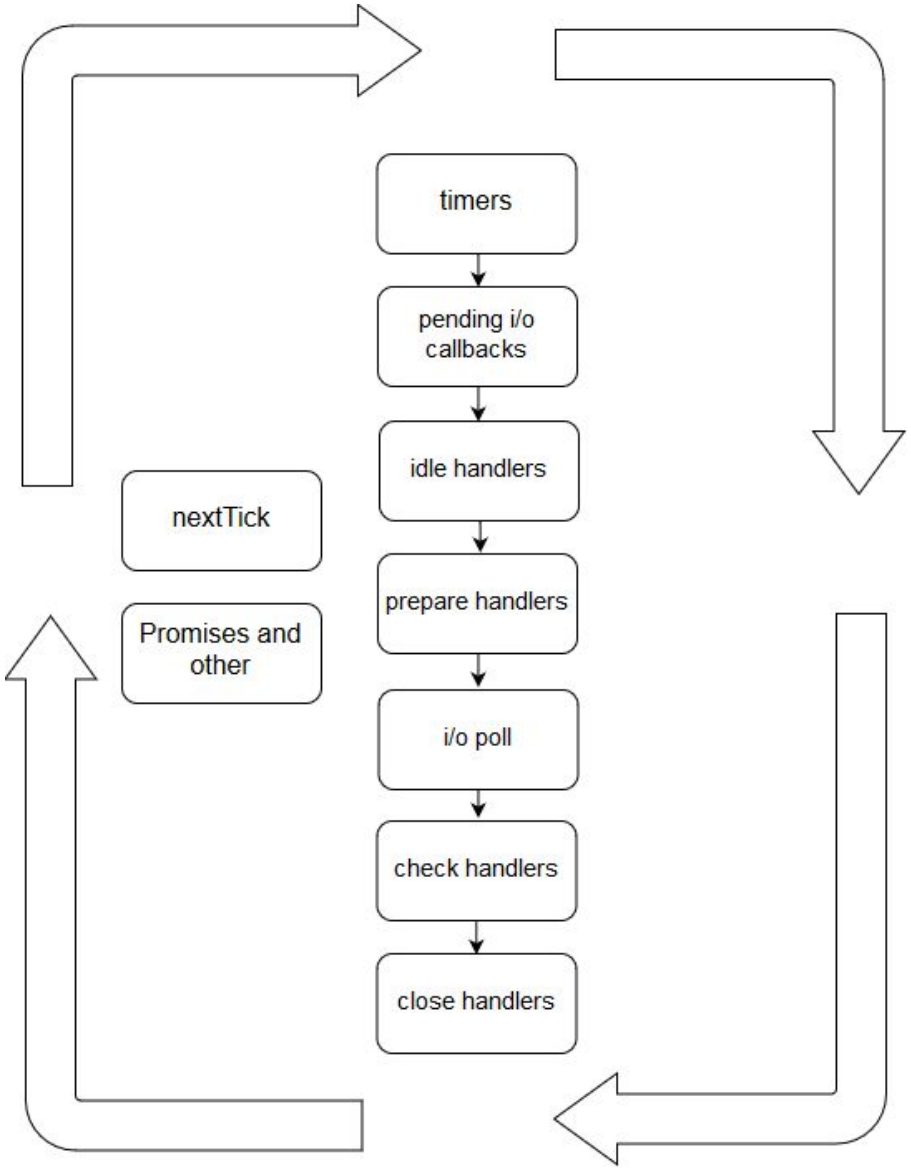


Несколько заблуждений об event loop

- Event loop работает в отдельном потоке
- Event loop работает как очередь или стек
- Все операции с ОС происходят через thread pool

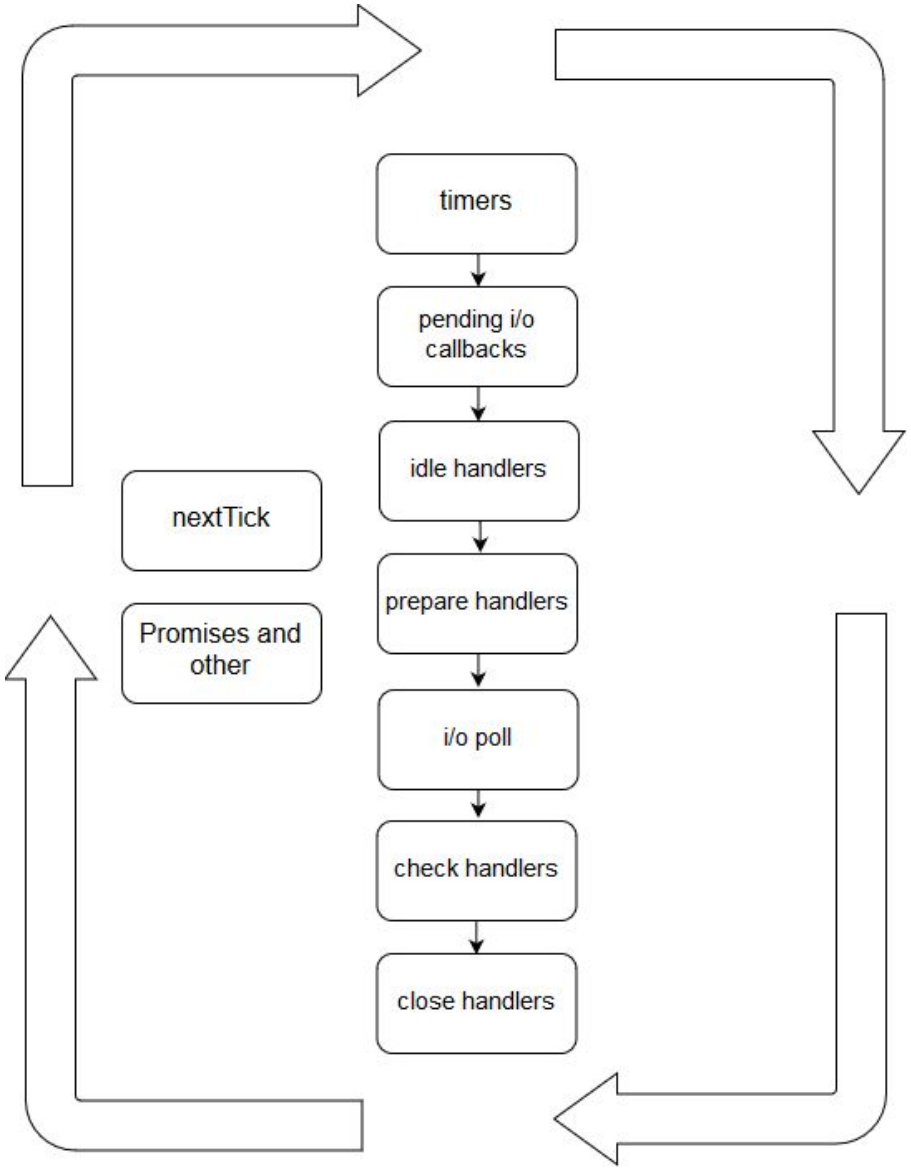
Пример кода

```
1 const fs = require('fs');
2
3 console.log("start app");
4
5 setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7 setImmediate(() => console.log("setImmediate 1")); //(2)
8
9 Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



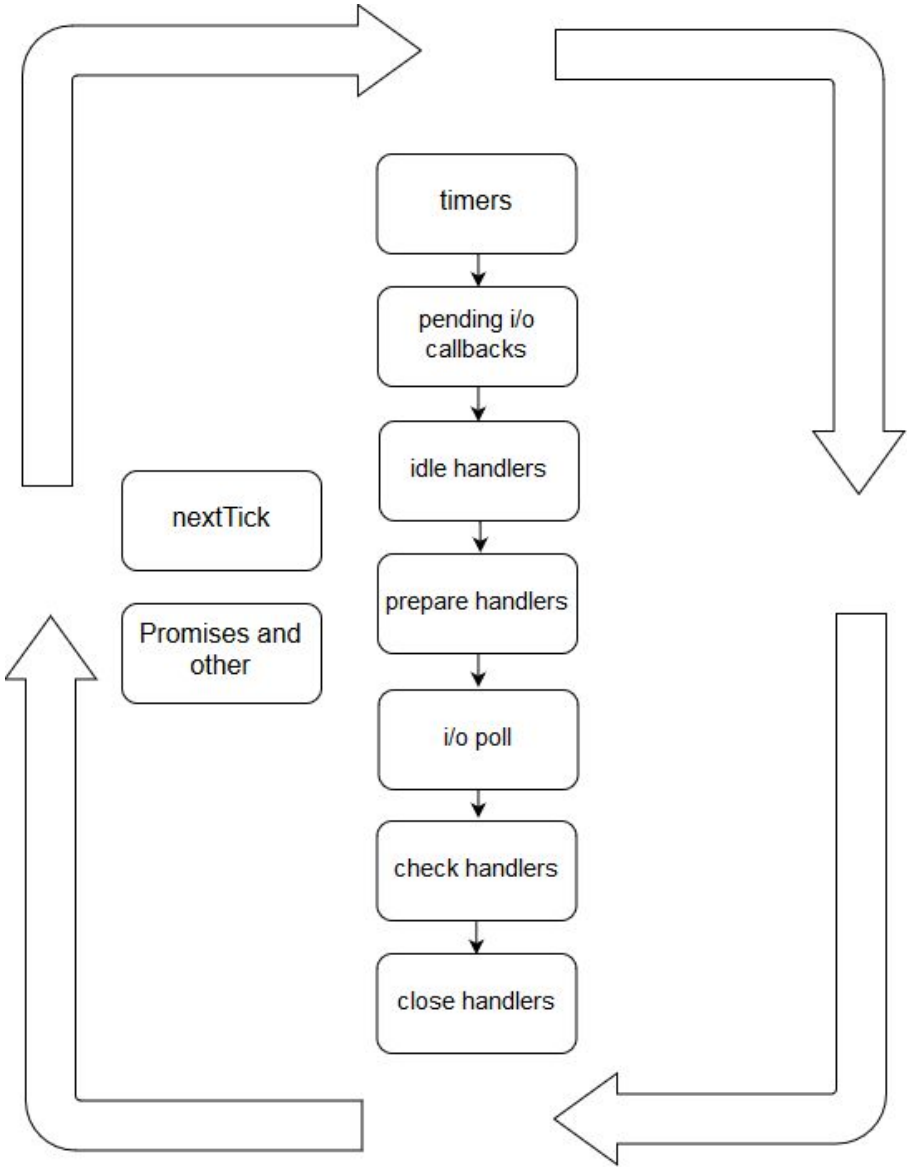
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



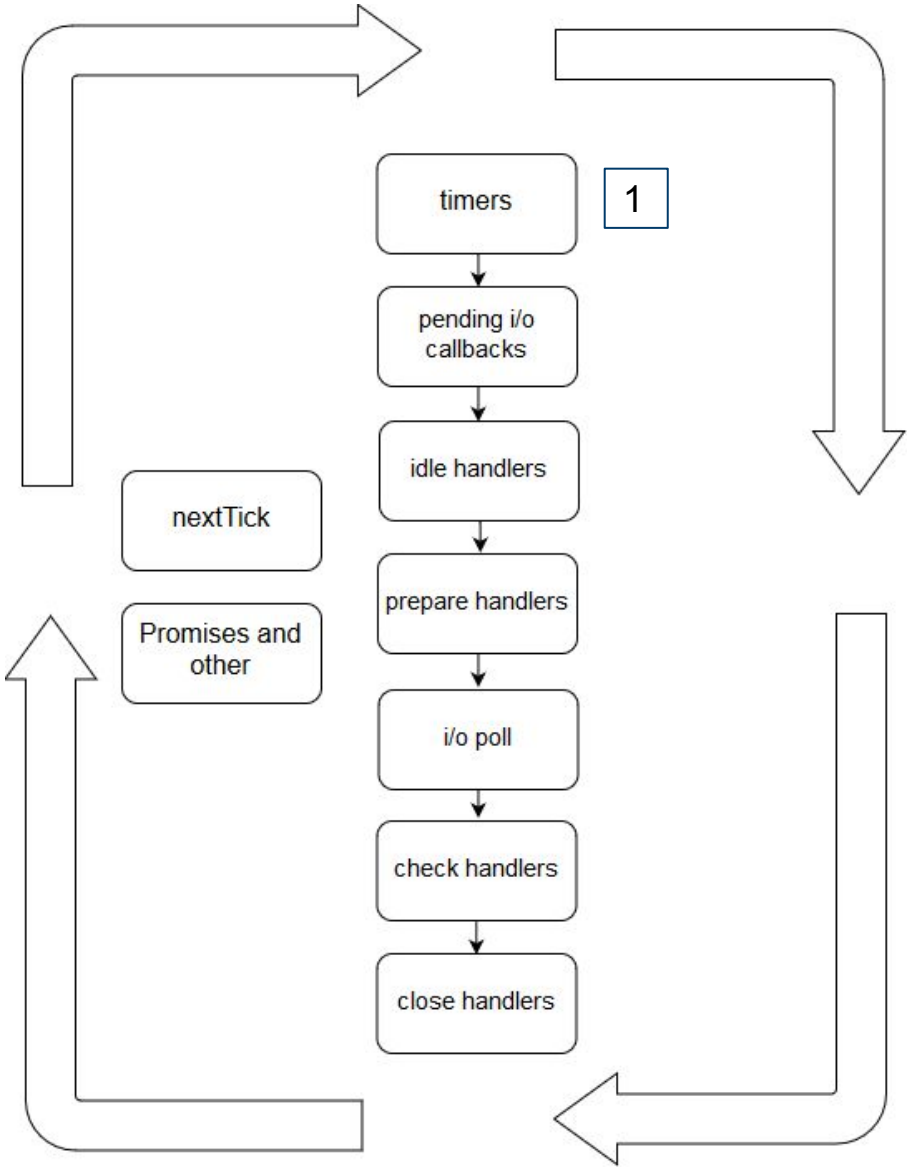
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



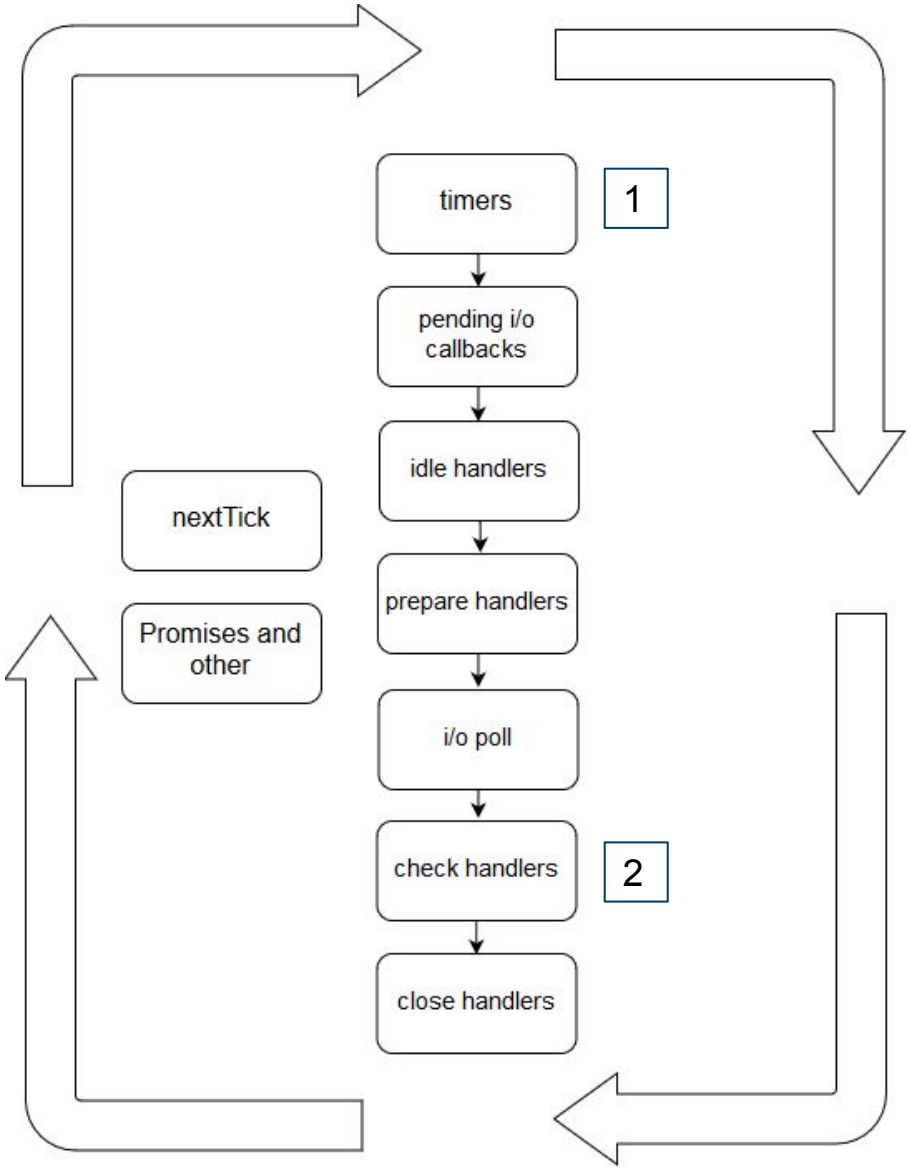
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



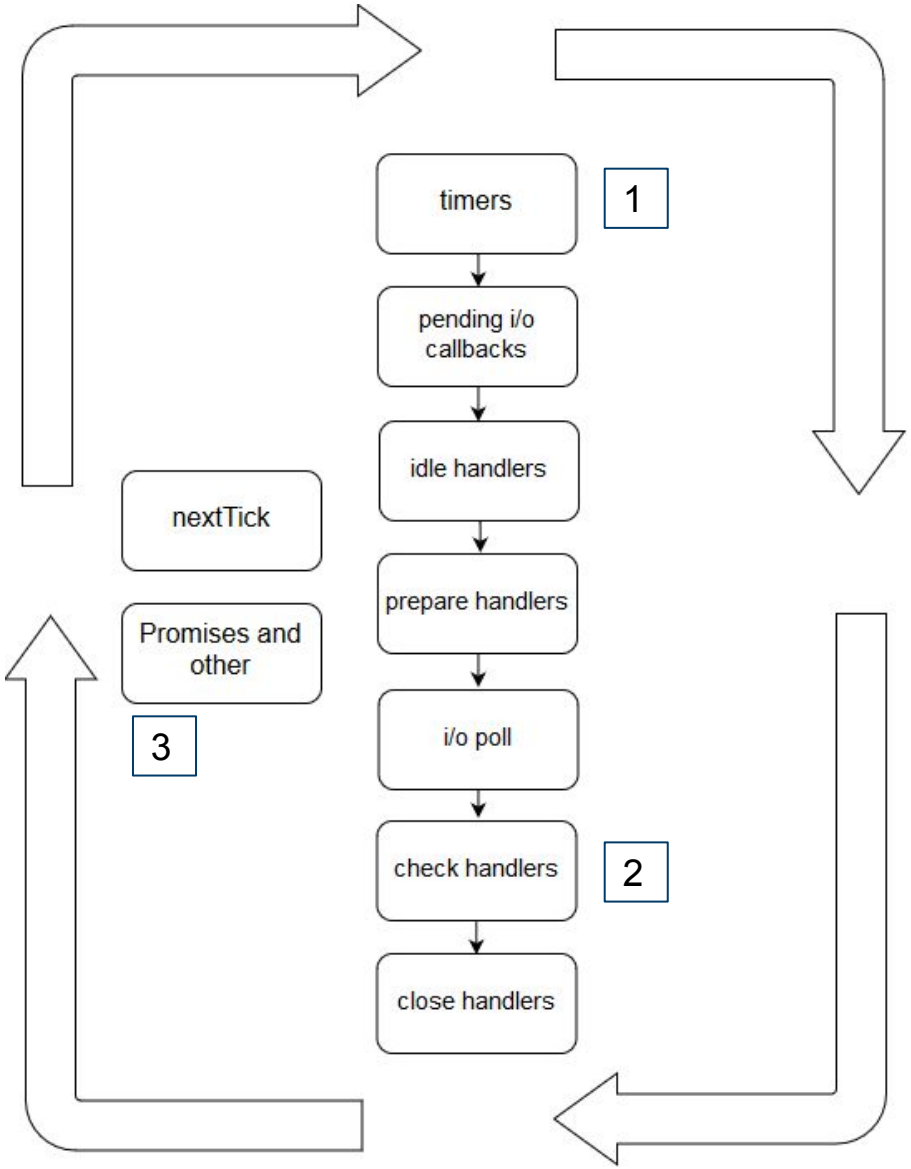
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



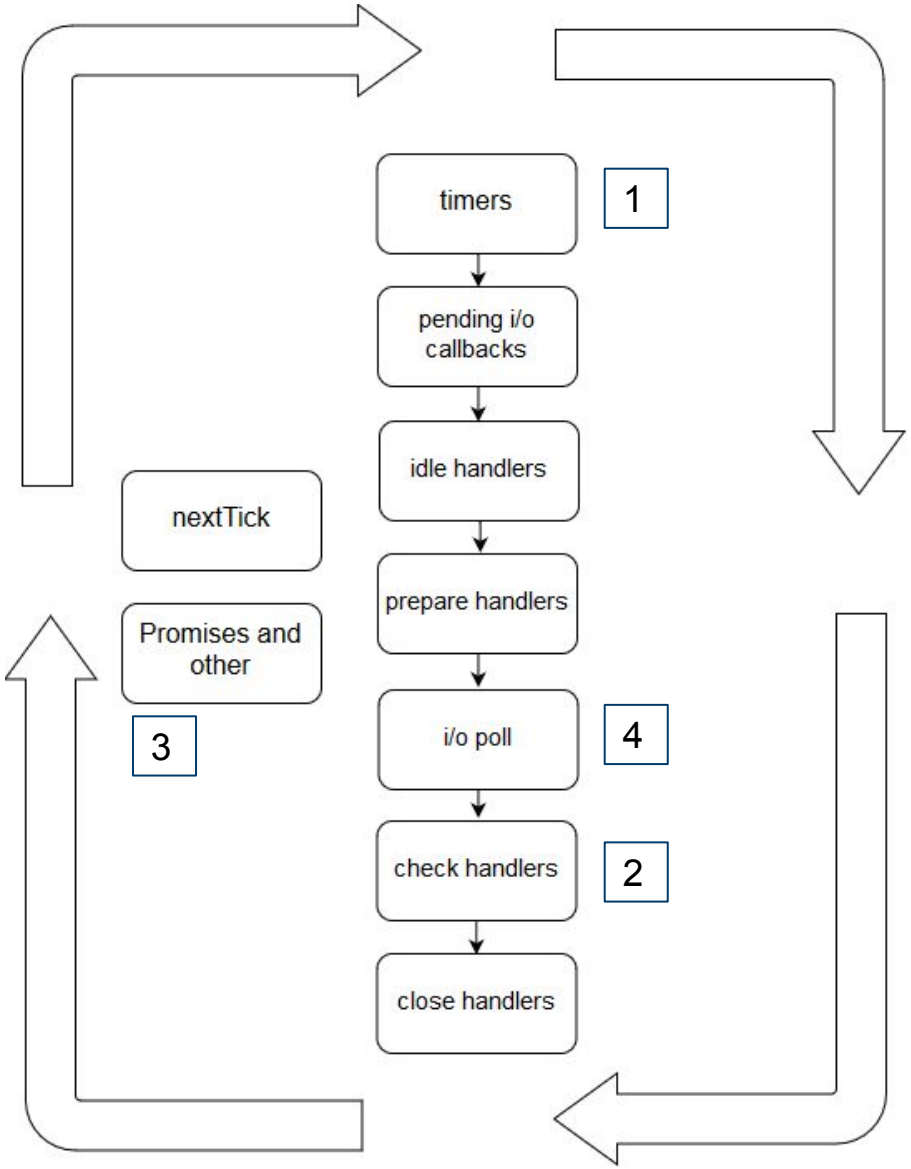
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



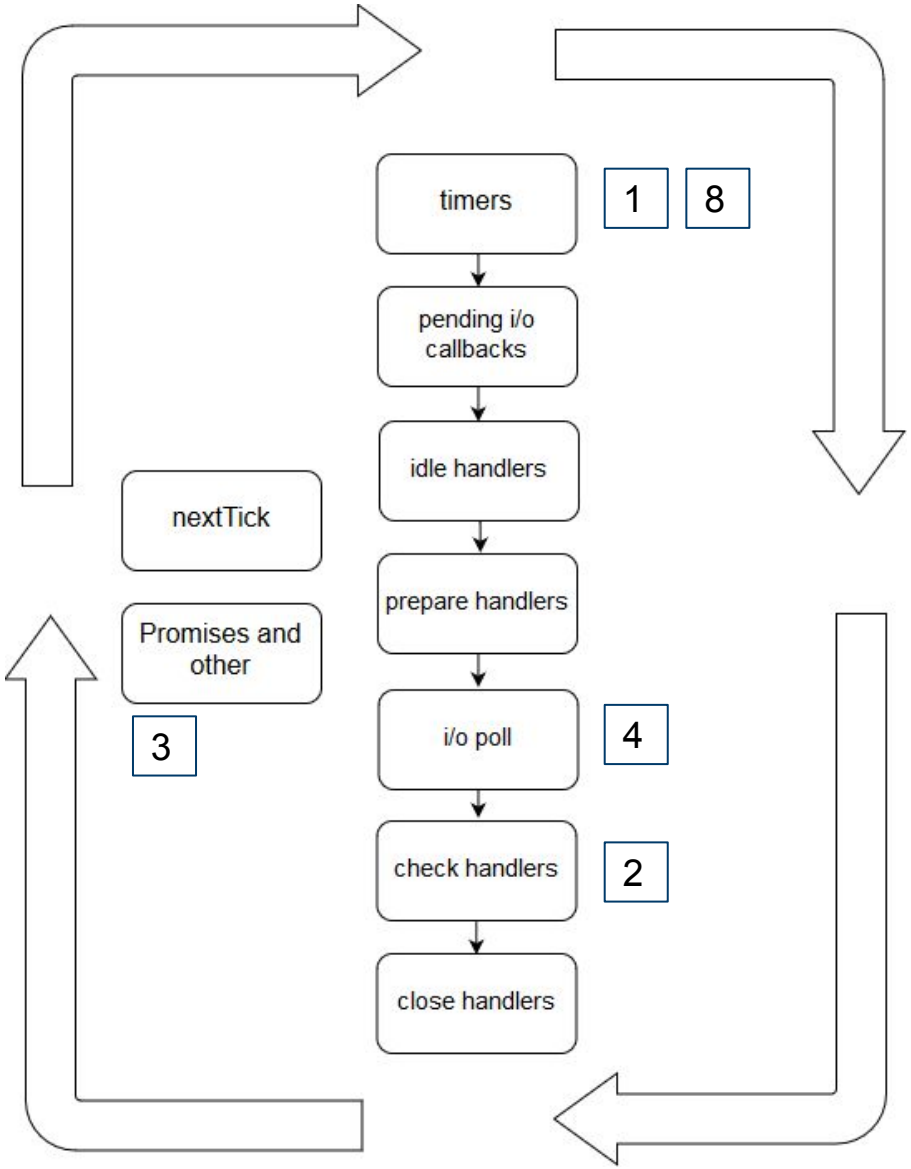
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



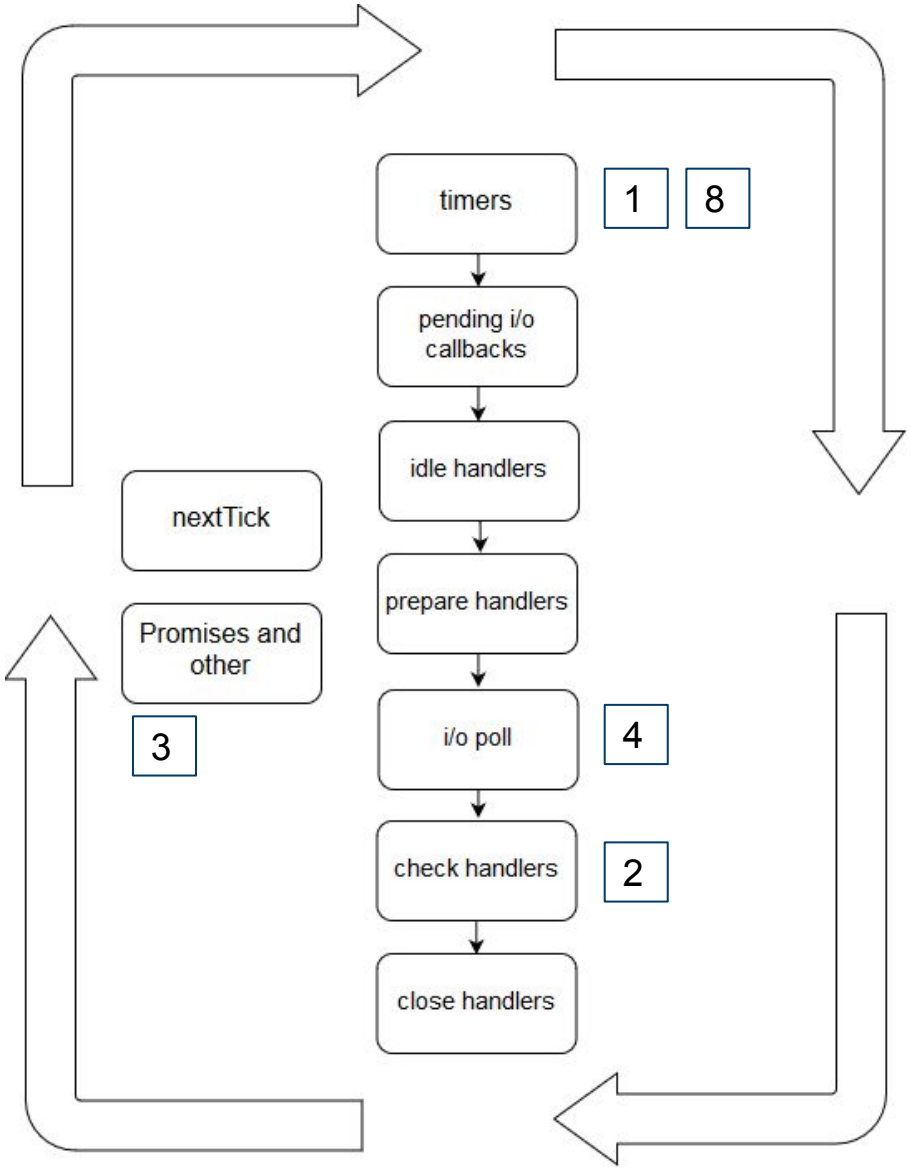
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



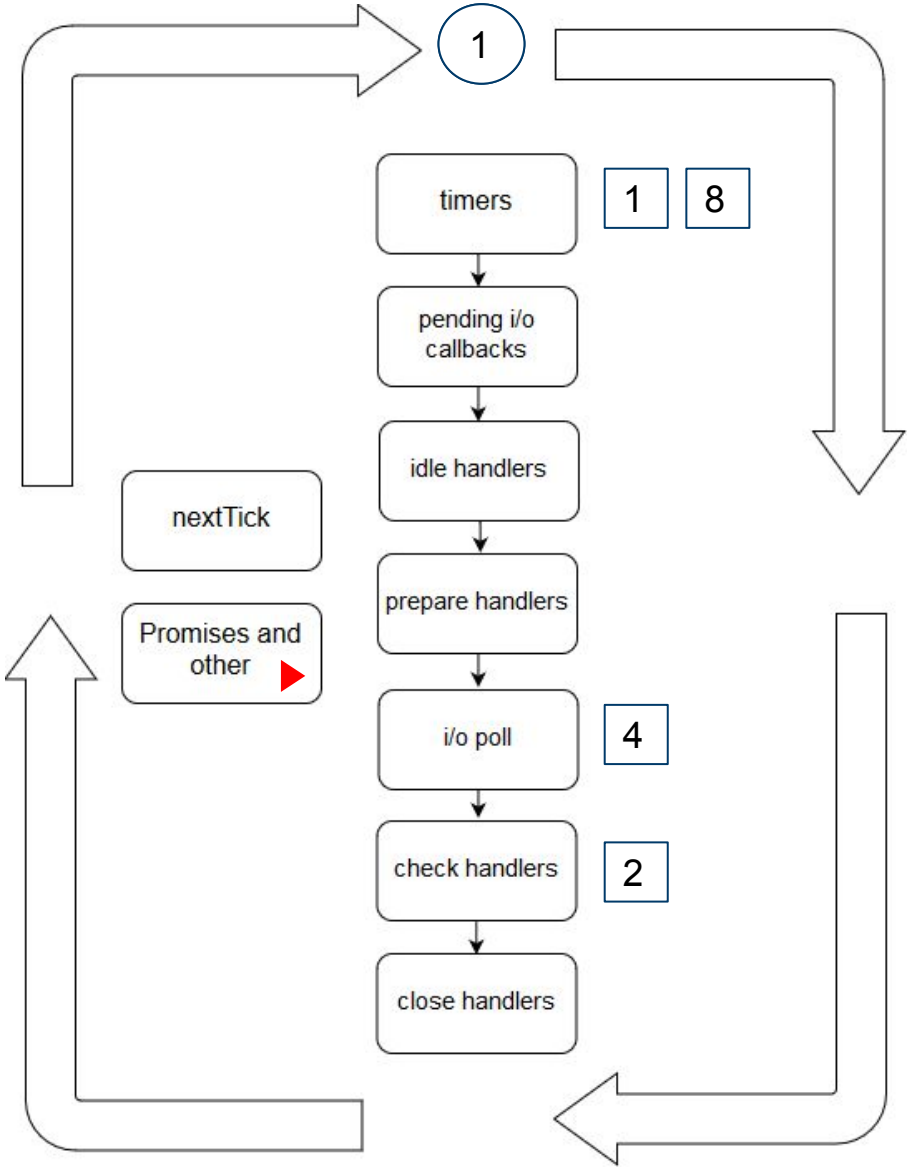
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



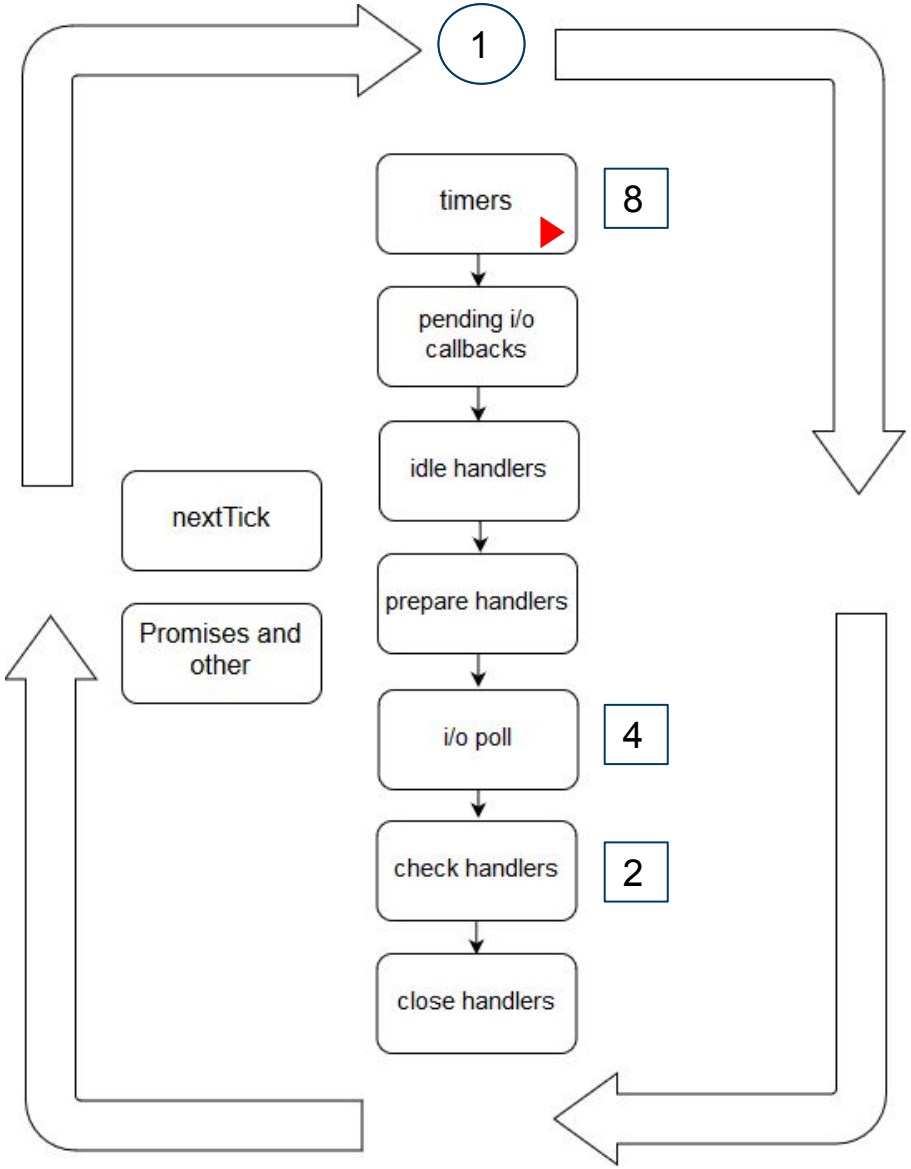
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



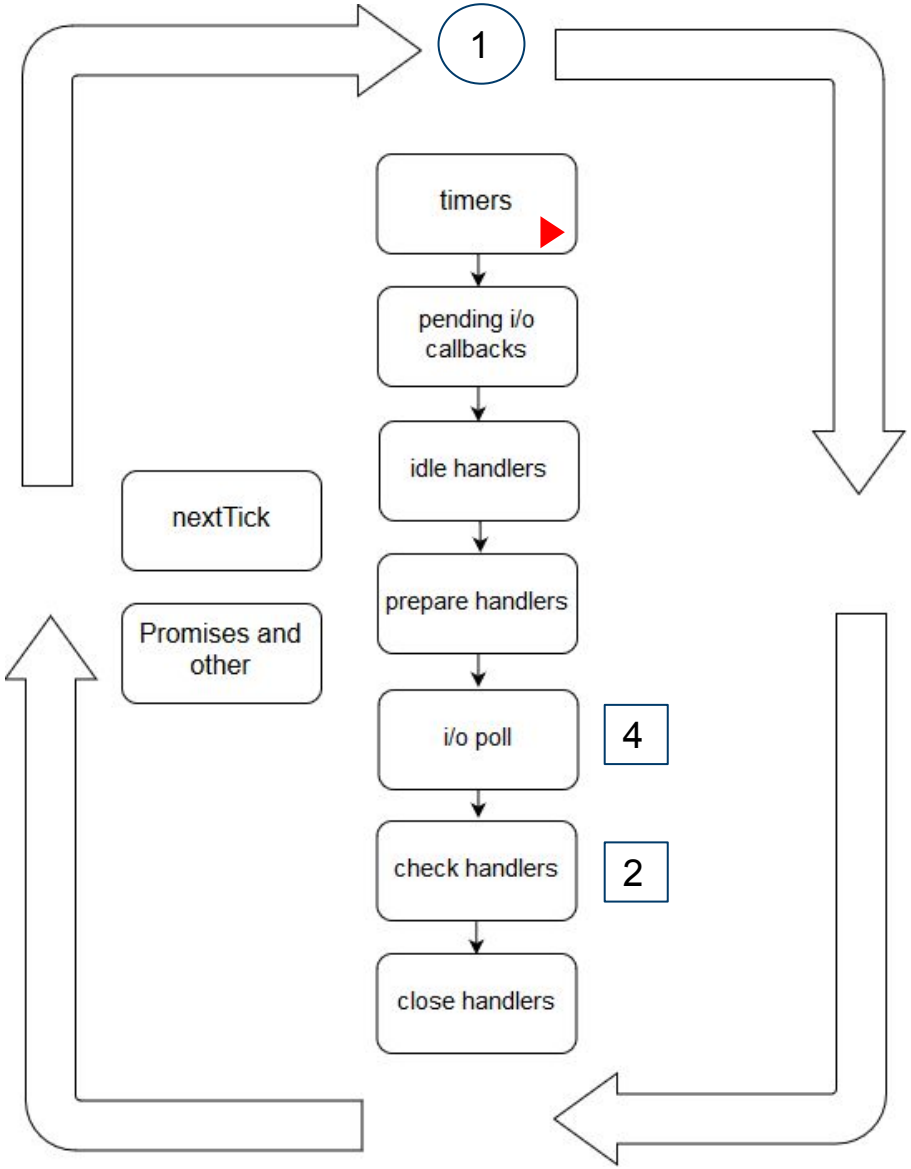
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



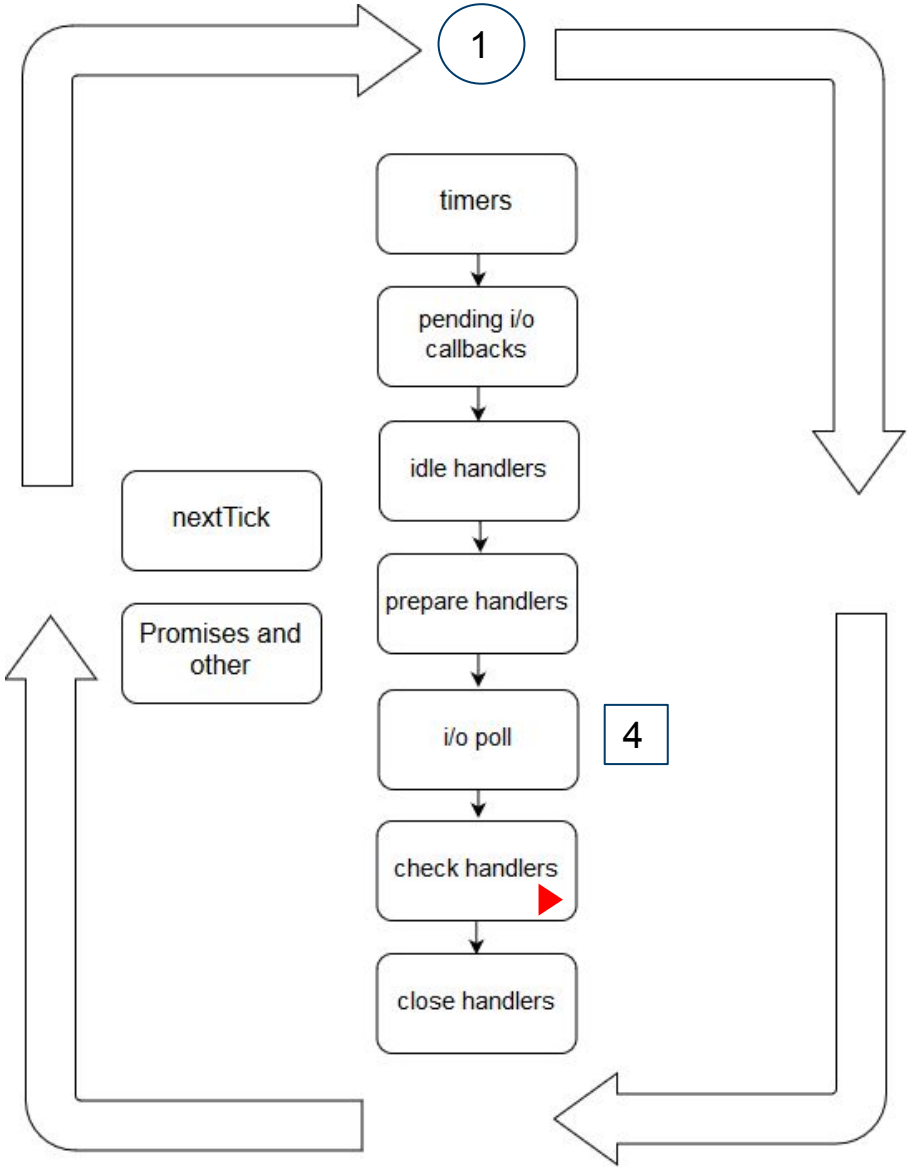
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



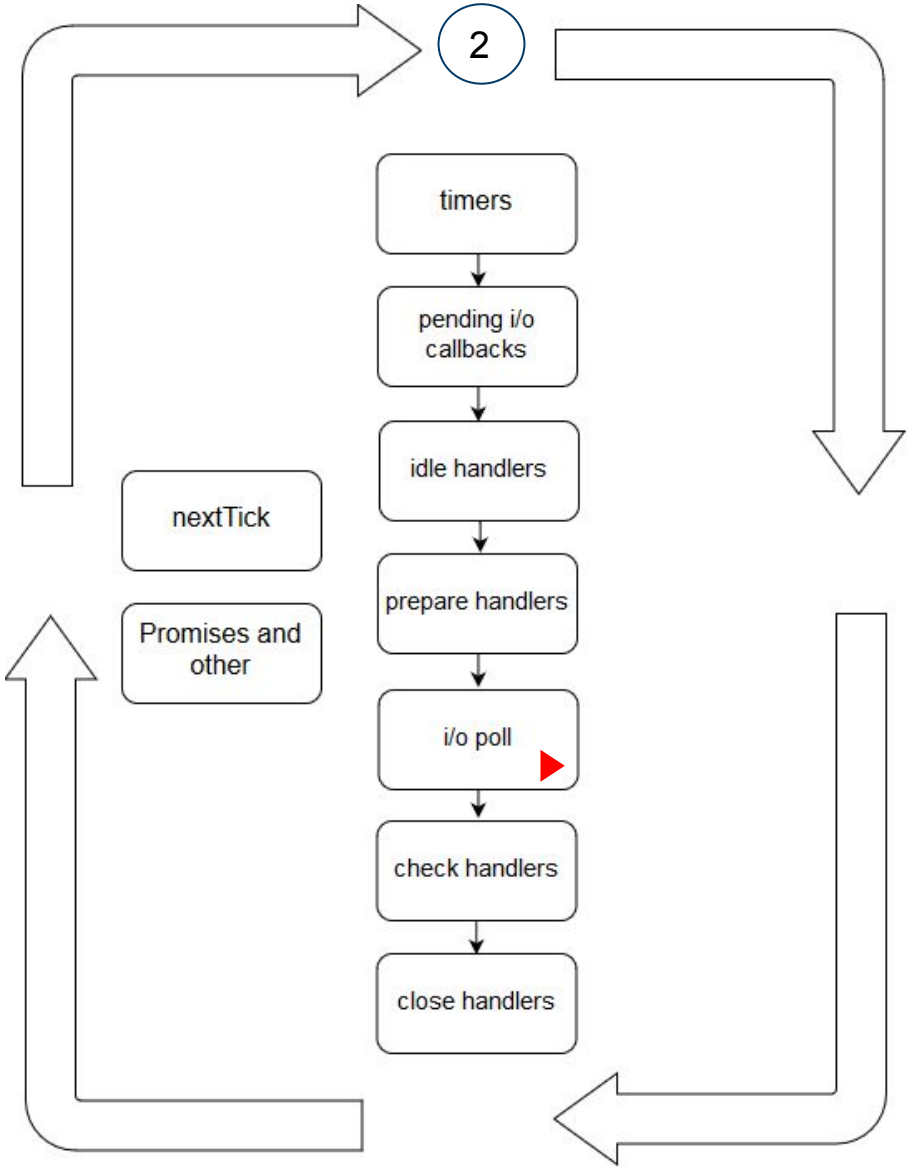
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



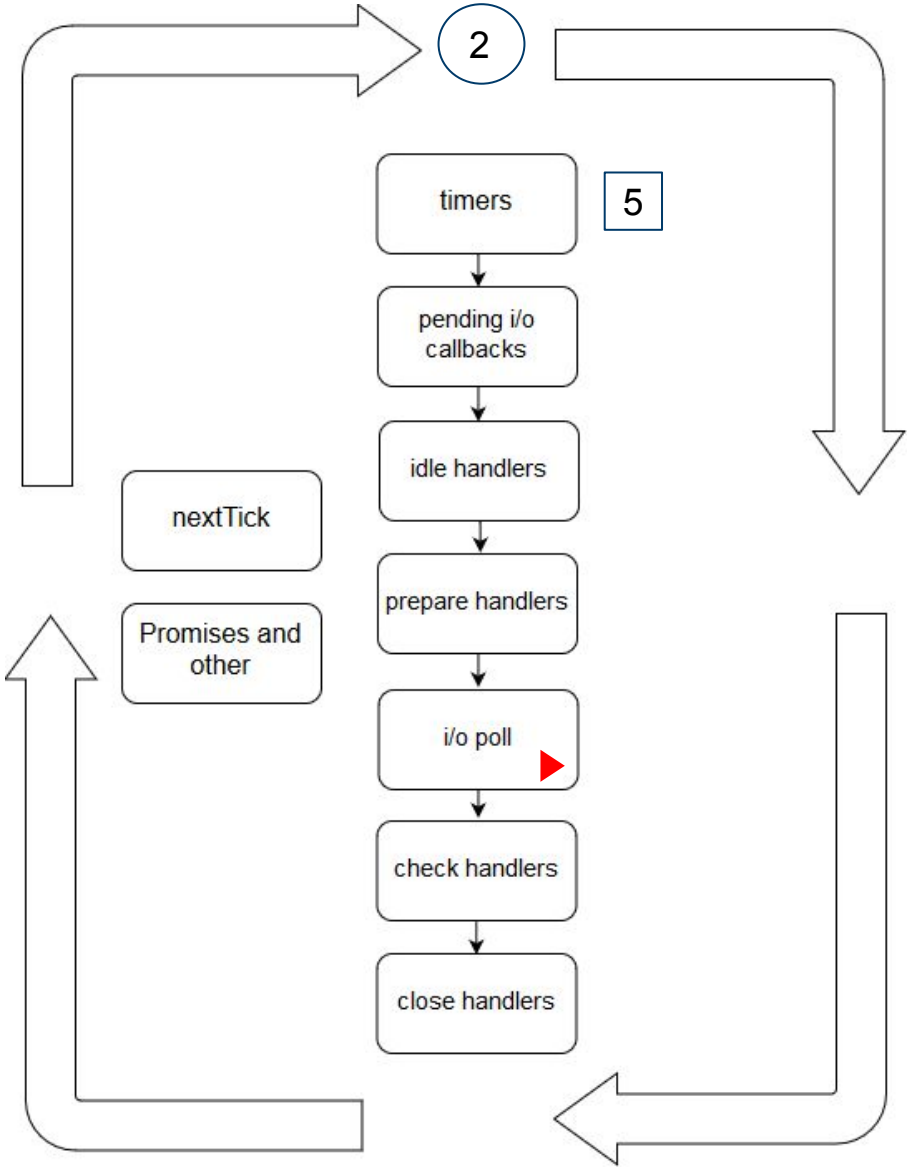
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



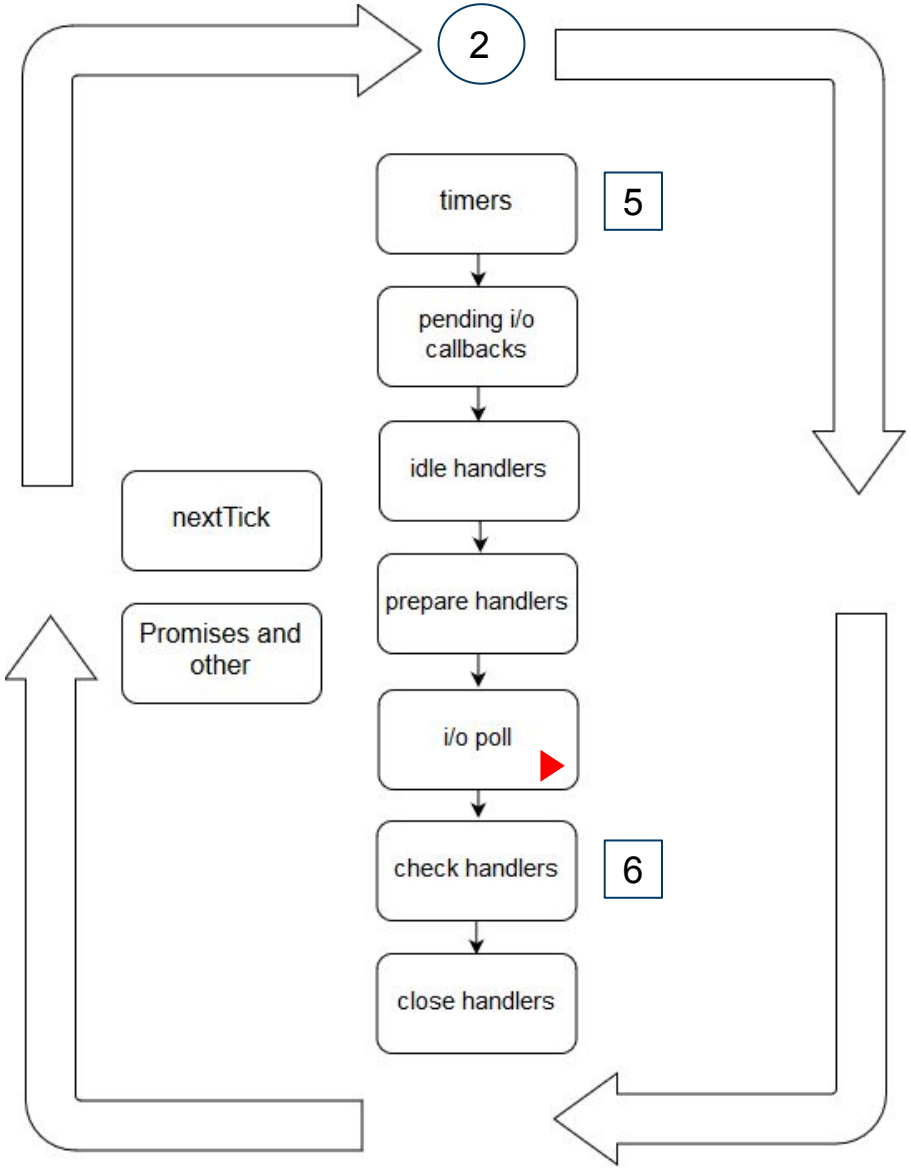
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



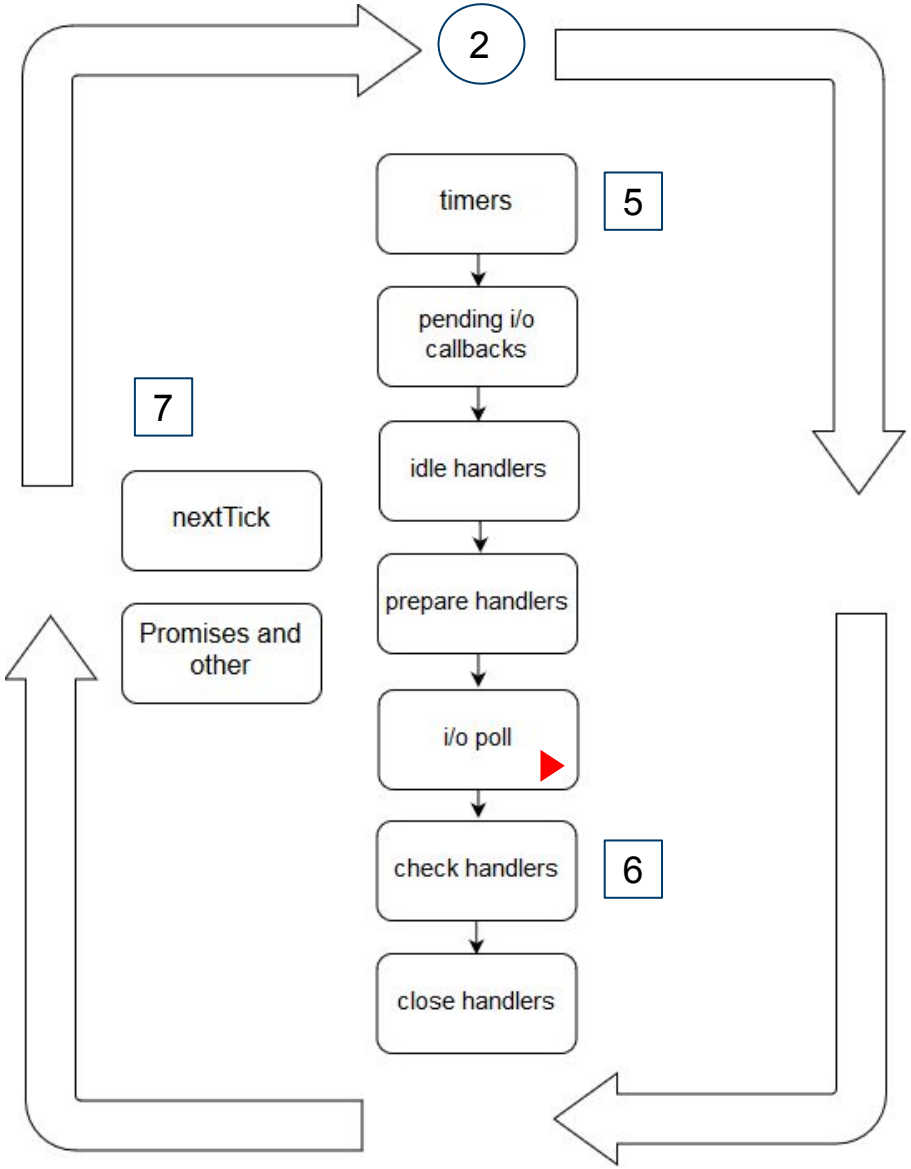
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



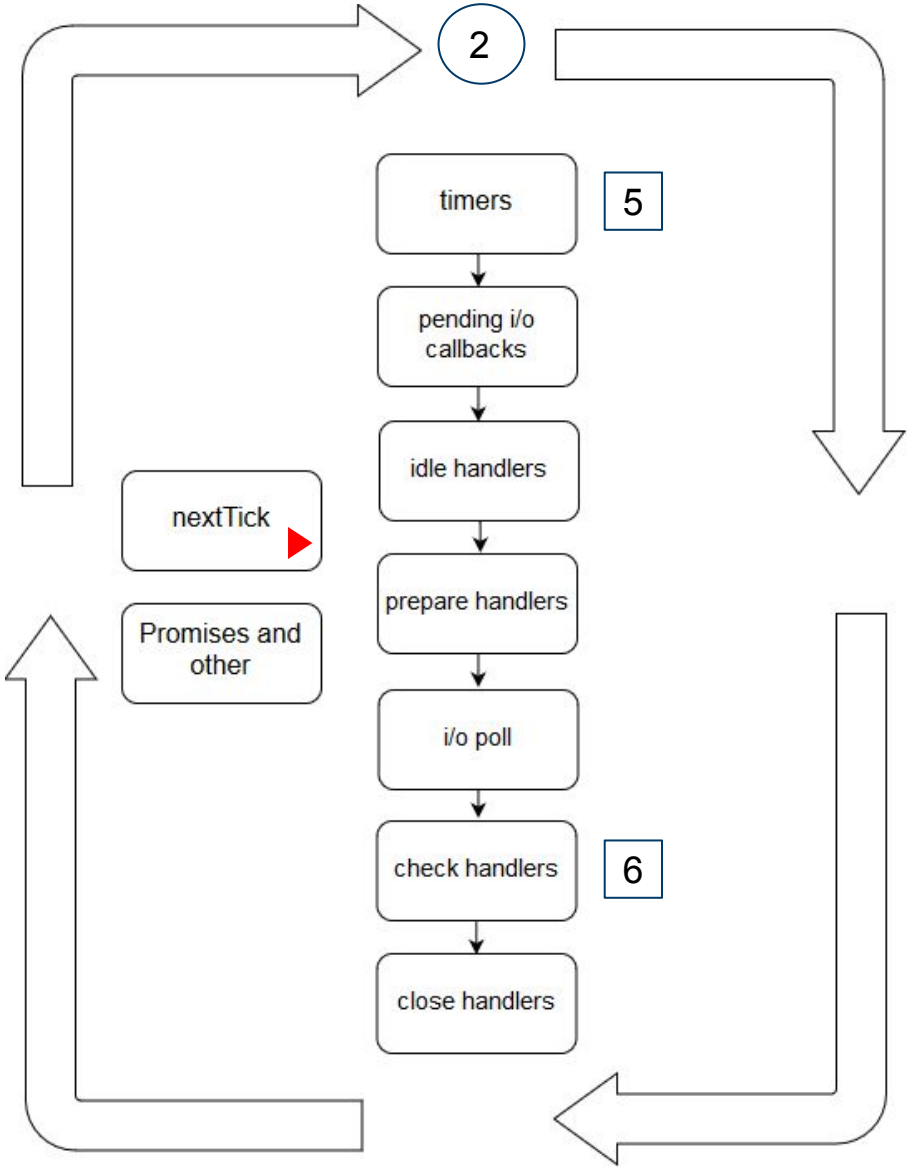
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



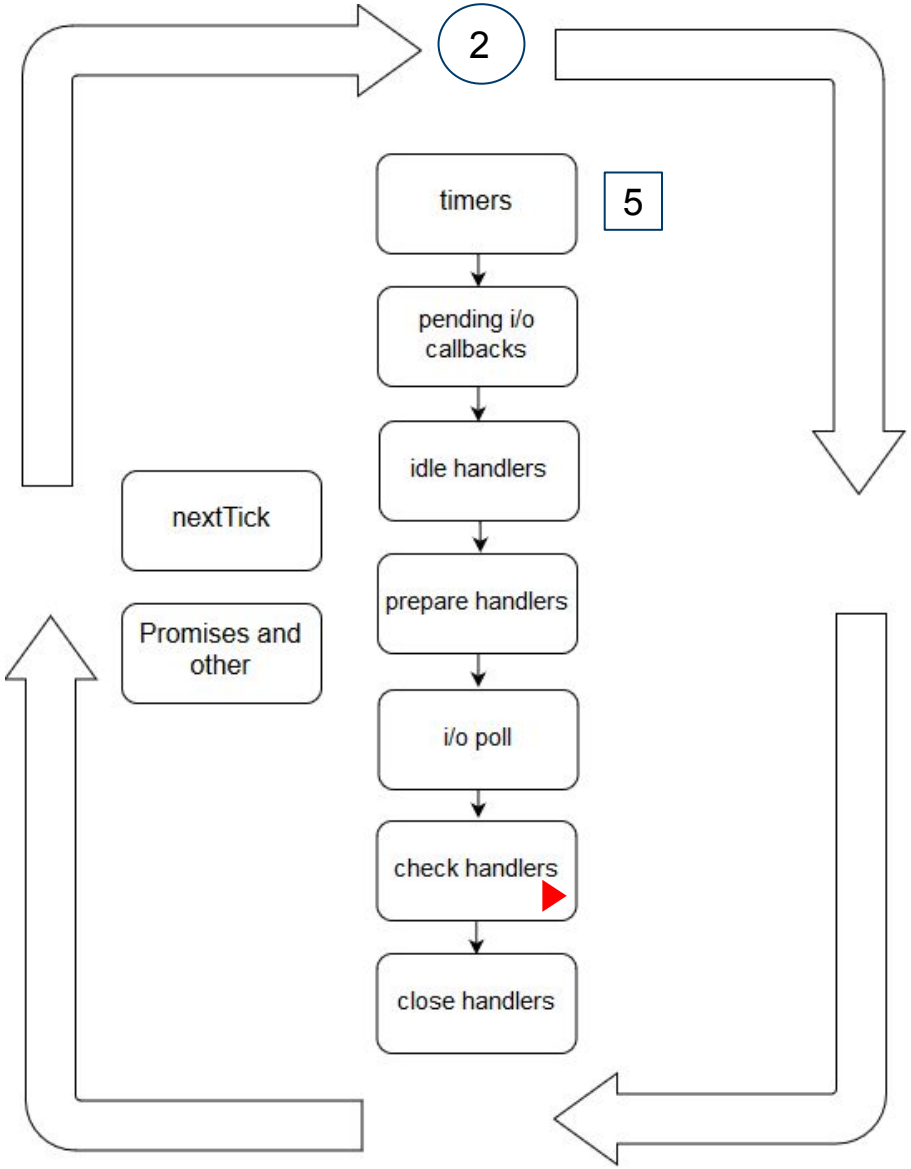
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



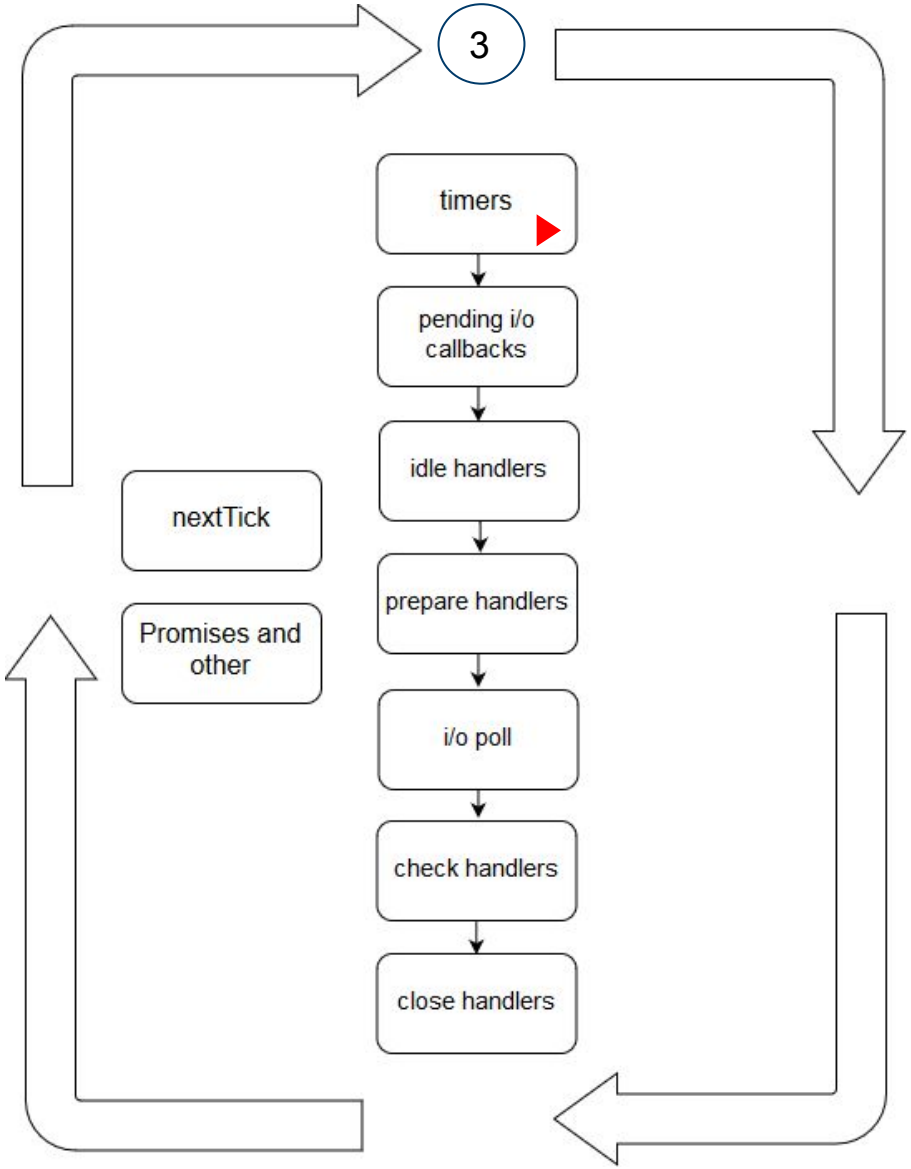
Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



Пример кода

```
1  const fs = require('fs');
2
3  console.log("start app");
4
5  setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7  setImmediate(() => console.log("setImmediate 1")); //(2)
8
9  Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```



Пример кода

```
1 const fs = require('fs');
2
3 console.log("start app");
4
5 setTimeout(() => console.log("set timeout 1"), 0); //(1)
6
7 setImmediate(() => console.log("setImmediate 1")); //(2)
8
9 Promise.resolve().then(()=> {
10   console.log("promise"); // 3
11 });
12
13 fs.readFile(__filename, () => {
14   console.log("read file"); // (4)
15   setTimeout(() => console.log("set timeout 2"), 0); // (5)
16   setImmediate(() => console.log("setImmediate 2")); // (6)
17   process.nextTick(() => console.log("next tick")); // (7)
18 });
19
20 setTimeout(() => console.log("set timeout 3"), 0); // (8)
21
22 console.log("end app");
```

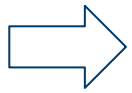
```
Anvar@DESKTOP-2QRO73R MINGW64 /d
$ node event_loop.js
start app
end app
promise
set timeout 1
set timeout 3
setImmediate 1
read file
next tick
setImmediate 2
set timeout 2
```

Коротко про cluster module

- Позволяет использовать процессор на полную
- Запускает новые процессы Node.js с помощью cluster.fork()
- Родительский процесс может общаться с детьми, но у детей нет общей памяти
- Каждый процесс запускает заново V8 и libuv
- Есть два режима для передачи данных в дочерний процесс - SCHED_NONE и SCHED_RR

Пример использования cluster module

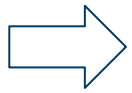
```
1 const cluster = require('cluster');
2 const http = require('http');
3 const numCPUs = require('os').cpus().length;
4
5 if (cluster.isMaster) {
6   console.log(`Master ${process.pid} is running`);
7
8   // Fork workers.)
9   for (let i = 0; i < numCPUs; i++) {
10    cluster.schedulingPolicy = cluster.SCHED_NONE;
11    cluster.fork();
12  }
13  console.log(`created ${numCPUs} workers`);
14
15  cluster.on('exit', (worker, code, signal) => {
16    console.log(`worker ${worker.process.pid} died`);
17  });
18 } else {
19  // Workers can share any TCP connection
20  // In this case it is an HTTP server
21  http.createServer((req, res) => {
22    res.writeHead(200);
23    let workerName = `worker ${process.pid}`;
24    res.end(`hello world from worker ${workerName}`);
25  }).listen(8000);
26
27  console.log(`Worker ${process.pid} started`);
28 }
```



```
» await Promise
  .all(Array.from({length: 100}, _ => f()))
  .then(array => Promise.all
  (array.map(async e => await e.text())))

← (100) […]
    0: "hello world from worker worker 14904"
    1: "hello world from worker worker 14904"
    2: "hello world from worker worker 14904"
    3: "hello world from worker worker 14904"
    4: "hello world from worker worker 14904"
    5: "hello world from worker worker 14904"
    6: "hello world from worker worker 14904"
    7: "hello world from worker worker 14904"
    8: "hello world from worker worker 14904"
    9: "hello world from worker worker 14904"
   10: "hello world from worker worker 14904"
   11: "hello world from worker worker 14904"
   12: "hello world from worker worker 14904"
   13: "hello world from worker worker 14904"
   14: "hello world from worker worker 14904"
   15: "hello world from worker worker 14904"
   16: "hello world from worker worker 14904"
   17: "hello world from worker worker 14904"
   18: "hello world from worker worker 14904"
   19: "hello world from worker worker 14904"
   20: "hello world from worker worker 14904"
   21: "hello world from worker worker 14904"
```

Пример использования cluster module

```
1 const cluster = require('cluster');
2 const http = require('http');
3 const numCPUs = require('os').cpus().length;
4
5 if (cluster.isMaster) {
6   console.log(`Master ${process.pid} is running`);
7
8   // Fork workers.)
9   for (let i = 0; i < numCPUs; i++) {
10    cluster.schedulingPolicy = cluster.SCHED_RR;
11    cluster.fork();
12  }
13  console.log(`created ${numCPUs} workers`);
14
15  cluster.on('exit', (worker, code, signal) => {
16    console.log(`worker ${worker.process.pid} died`);
17  });
18 } else {
19  // Workers can share any TCP connection
20  // In this case it is an HTTP server
21  http.createServer((req, res) => {
22    res.writeHead(200);
23    let workerName = `worker ${process.pid}`;
24    res.end(`hello world from worker ${workerName}`);
25  }).listen(8000);
26
27  console.log(`Worker ${process.pid} started`);
28 }
```



```
» await Promise
  .all(Array.from({length: 100}, _ => f()))
  .then(array => Promise.all
  (array.map(async e => await e.text())))

← (100) […]
    0: "hello world from worker worker 11000"
    1: "hello world from worker worker 11000"
    2: "hello world from worker worker 11000"
    3: "hello world from worker worker 10272"
    4: "hello world from worker worker 15144"
    5: "hello world from worker worker 11000"
    6: "hello world from worker worker 10272"
    7: "hello world from worker worker 15144"
    8: "hello world from worker worker 11000"
    9: "hello world from worker worker 10272"
   10: "hello world from worker worker 15144"
   11: "hello world from worker worker 11000"
   12: "hello world from worker worker 10272"
   13: "hello world from worker worker 15144"
   14: "hello world from worker worker 5464"
   15: "hello world from worker worker 10692"
   16: "hello world from worker worker 11000"
   17: "hello world from worker worker 19112"
   18: "hello world from worker worker 10272"
   19: "hello world from worker worker 15144"
   20: "hello world from worker worker 11000"
   21: "hello world from worker worker 5464"
```

И совсем чуть-чуть про workers

- Доступны с версии Node.js 10.5.0
- Потоки, а не процессы
- Можно шарить память между воркерами (ArrayBuffer и SharedArrayBuffer)
- Один V8, один event loop

И совсем немного про workers

workers.js

```
1 const { Worker } = require('worker_threads')
2
3 function runService(workerData) {
4   return new Promise((resolve, reject) => {
5     const worker = new Worker('./service.js', { workerData });
6     worker.on('message', resolve);
7     worker.on('error', reject);
8     worker.on('exit', (code) => {
9       if (code !== 0)
10        reject(new Error(`Worker stopped with exit code ${code}`));
11    })
12  })
13 }
14
15 async function run() {
16   const result = await runService('world')
17   console.log(result);
18 }
19
20 run().catch(err => console.error(err))
```

И совсем немного про workers

service.js

```
1 const { workerData, parentPort } = require('worker_threads')
2
3 // You can do any heavy stuff here, in a synchronous way
4 // without blocking the "main thread"
5 let message = `${workerData} from worker`;
6 parentPort.postMessage({ hello: message });
```

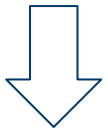
И совсем немного про workers

workers.js

```
const { Worker } = require('worker_threads')

function runService(workerData) {
  return new Promise((resolve, reject) => {
    const worker = new Worker('./service.js', { workerData });
    worker.on('message', resolve);
    worker.on('error', reject);
    worker.on('exit', (code) => {
      if (code !== 0)
        reject(new Error(`Worker stopped with exit code ${code}`));
    })
  })
}

async function run() {
  const result = await runService('world')
  console.log(result);
}

run().catch(err => console.error(err))
```

service.js

```
const { workerData, parentPort } = require('worker_threads')

// You can do any heavy stuff here, in a synchronous way
// without blocking the "main thread"
let message = `${workerData} from worker`;
parentPort.postMessage({ hello: message });
```



```
Anvar@DESKTOP-2QRO73R MINGW64 /d
$ node workers.js
{ hello: 'world from worker' }
```

Спасибо за внимание

tg: @RamazanovAnvar

Примеры здесь: https://github.com/GSTQ/node_meetup