ПРОГРАММИРОВАНИЕ В INTERNET

ГЛОБАЛЬНЫЕ ОБЪЕКТЫ

Глобальные объекты

- global
- buffer
- require
- console
- exports
- module
- process
- и др. (подробнее: https://nodejs.org/api/globals.html)

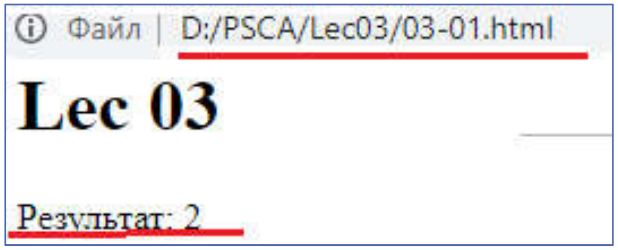
Объект global =

специальный объект, который предоставляет доступ к глобальным, то есть доступным из каждого модуля приложения, переменным и функциям. Примерным аналогом данного объекта в JavaScript для браузера является объект window.

global vs window

```html
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>02-03</title>
    <script src="./s03-01.js" ></script>
</head>
<body>
  <h1>Lec 03</h1>
<div id='result'></div>
<script>

    var x = 3;
    result.innerHTML  =  'Результат: ' + mod2(8);

  </script>
</body>
```

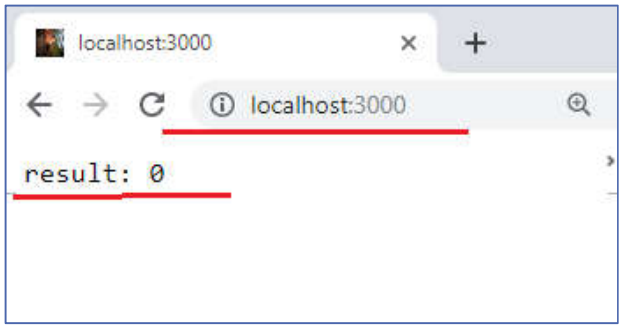
```
var x = 2;

mod2 = (inp)=>{return parseInt(inp)%x;}
```



global vs window

```
var http = require('http');
var fs   = require('fs');
var mod  = require('./m03-01');

http.createServer(function (request, response) {

    response.contentType='text/plain';

    var  x = 3;

    let result = mod.mod2(8);

    response.end(`result: ${result}`);

}).listen(3000);

console.log('Server running at http://localhost:3000/');
```

```
var x = 2;
exports.mod2 = (inp)=>{return parseInt(inp)%x;}
```



Объект global

```
console.log('m03-01', global);
```

```
D:\NodeJS\samples\cwp_03>node 03-20
m03-01 <ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  performance: Performance {
    nodeTiming: PerformanceNodeTiming {
      name: 'node',
      entryType: 'node',
      startTime: 0,
      duration: 49.31469999998808,
      nodeStart: 1.1279999995604157,
      v8Start: 6.340099999681115,
      bootstrapComplete: 37.592699999921024,
      environment: 21.660199999809265,
      loopStart: -1,
      loopExit: -1,
      idleTime: 0
    },
    timeOrigin: 1663340597424.766
  },
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  }
}
```

Объект process = глобальный объект, который предоставляет информацию о текущем процессе Node.js и контролирует его.

Объект process

```
D:\NodeJS\samples\cwp_03>node -p "process.versions"
{
  node: '16.17.0',
  v8: '9.4.146.26-node.22',
  uv: '1.43.0',
  zlib: '1.2.11',
  brotli: '1.0.9',
  ares: '1.18.1',
  modules: '93',
  nghttp2: '1.47.0',
  napi: '8',
  llhttp: '6.0.7',
  openssl: '1.1.1q+quic',
  cldr: '41.0',
  icu: '71.1',
  tz: '2022a',
  unicode: '14.0',
  ngtcp2: '0.1.0-DEV',
  nghttp3: '0.1.0-DEV'
}
```

```
D:\PSCA\Lec03>node -p "process.env"
{ ALLUSERSPROFILE: 'C:\\ProgramData',
  APPDATA: 'C:\\Users\\Win10_ISiT_Server\\AppData\\Roaming',
  CLIENTNAME: 'USER',
  CommonProgramFiles: 'C:\\Program Files\\Common Files',
  'CommonProgramFiles(x86)': 'C:\\Program Files (x86)\\Common Files',
  CommonProgramW6432: 'C:\\Program Files\\Common Files',
  COMPUTERNAME: 'A306-2-SMW60',
  ComSpec: 'C:\\WINDOWS\\system32\\cmd.exe',
  DriverData: 'C:\\Windows\\System32\\Drivers\\DriverData',
  FSHARPINSTALLDIR:
   'C:\\Program Files (x86)\\Microsoft SDKs\\F#\\10.1\\Framework\\v4.0\\',
  HOMEDRIVE: 'C:',
  HOMEPATH: '\\Users\\Win10_ISiT_Server',
  JAVA_HOME: 'C:\\Program Files\\Java\\jdk1.8.0_151',
  LOCALAPPDATA: 'C:\\Users\\Win10_ISiT_Server\\AppData\\Local',
  LOGONSERVER: '\\\\A306-2-SMW60',
  MOZ_PLUGIN_PATH:
   'C:\\Program Files (x86)\\Foxit Software\\Foxit Reader\\plugins\\',
  NUMBER_OF_PROCESSORS: '8',
  OneDrive: 'C:\\Users\\Win10_ISiT_Server\\OneDrive',
  OS: 'Windows_NT',
  Path:
   'C:\\ProgramData\\DockerDesktop\\version-bin;C:\\Program Files\\Docker\\Docker\\R
C:\\ProgramData\\Oracle\\Java\\javapath;C:\\WINDOWS\\system32;C:\\WINDOWS;C:\\WINDOW
am Files\\7-Zip;C:\\Program Files\\Microsoft SQL Server\\110\\Tools\\Binn\\;C:\\Prog
icrosoft SQL Server\\110\\Tools\\Binn\\;C:\\Program Files (x86)\\Microsoft SQL Serve
```

```
D:\NodeJS\samples\cwp_03>node -p "process.release"
{
  name: 'node',
  lts: 'Gallium',
  sourceUrl: 'https://nodejs.org/download/release/v16.17.0/node-v16.17.0.tar.gz',
  headersUrl: 'https://nodejs.org/download/release/v16.17.0/node-v16.17.0-headers.tar.gz',
  libUrl: 'https://nodejs.org/download/release/v16.17.0/win-x64/node.lib'
}
```

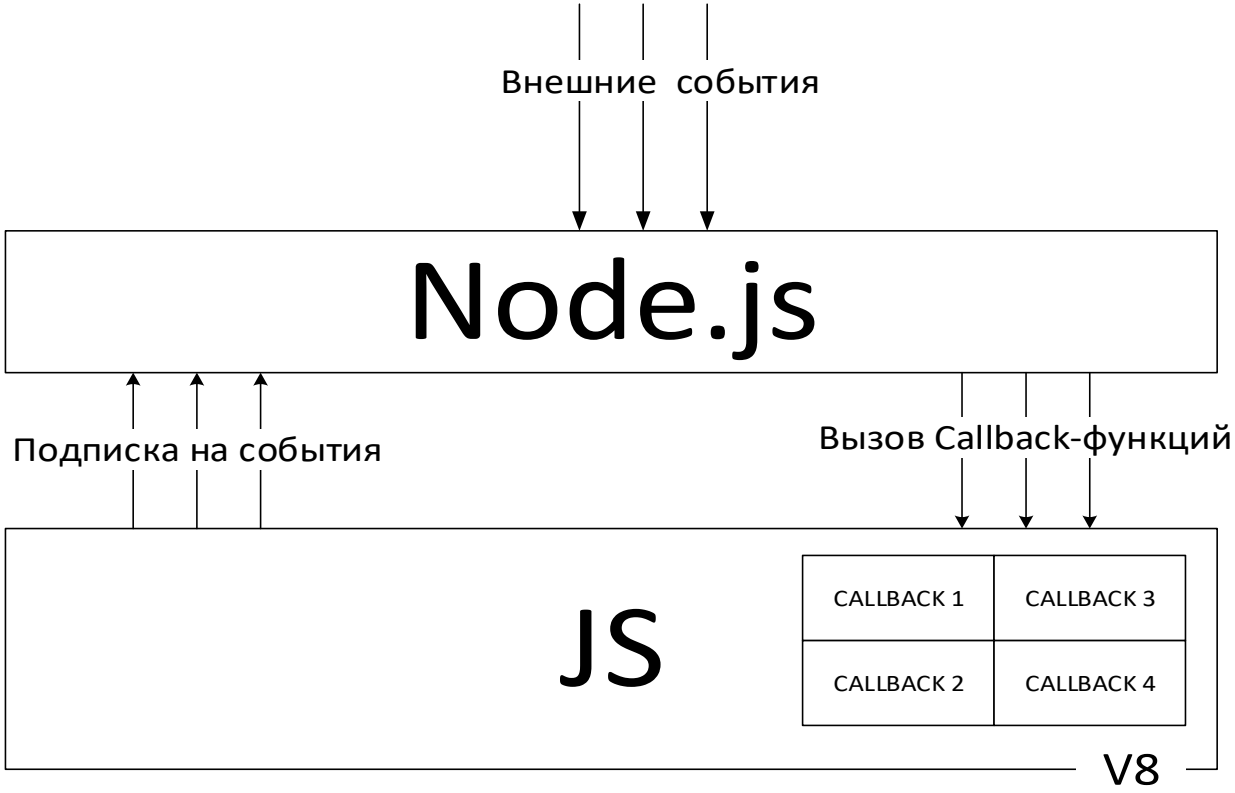
```
process.stdin.setEncoding('utf-8');
process.stdin.on('readable', ()=>{

    let chunk = null;
    while ((chunk = process.stdin.read()) != null){
        if      (chunk.trim() == 'exit')     process.exit(0);
        else if (chunk.trim() == 'uptime')   process.stdout.write('uptime =  '+ process.uptime().toString() +'\n');
        else if (chunk.trim() == 'version')  process.stdout.write('version = '+process.version +'\n');
        else if (chunk.trim() == 'title')    process.stdout.write('title = '  +process.title +'\n');
        else if (chunk.trim() == 'release')   {
            let obj = process.release;
            process.stdout.write('release:sourceUrl  = '    + obj.sourceUrl+'\n');
            process.stdout.write('release:headersUrl  = '   + obj.headersUrl+'\n');
            process.stdout.write('release:libUrl  = '       + obj.libUrl+'\n');
            process.stdout.write('release:lts  = '          + obj.lts+'\n');
        }
        else  process.stdout.write(chunk);
        // else if (chunk.trim() == 'report')   process.stdout.write('report = ' +process.report.filename +'\n'); v.11
        // else if (chunk.trim() == 'usage')   { // v.12
        //     let obj = process.resourceUsage();
        //     process.stdout.write('usage:userCPUTime  = '    + obj.userCPUTime.toString()+'\n');
        //     process.stdout.write('usage:systemCPUTime  = '  + obj.systemCPUTime.toString()+'\n');
        //     process.stdout.write('usage: maxRSS  = '        + obj. maxRSS
        // }

    }
});
```

Объект process предоставляет доступ к стандартным системным потокам: stdin, stdout, stderr.

```
Администратор: C:\Windows\System32\cmd.exe - node  03-02

D:\PSCA\Lec03>node 03-02
Server running at http://localhost:3000/
uptime
uptime =  15.901
version
version = v10.15.0
title
title = Администратор: C:\Windows\System32\cmd.exe - node  03-02
release
release:sourceUrl  = https://nodejs.org/download/release/v10.15.0/node-v10.15.0.tar.gz
release:headersUrl  = https://nodejs.org/download/release/v10.15.0/node-v10.15.0-headers.tar.gz
release:libUrl  = https://nodejs.org/download/release/v10.15.0/win-x64/node.lib
release:lts  = Dubnium
```

Объект buffer = глобальный объект, предназначенный для работы с двоичными данными: набором октетов.

Принцип издатель-подписчик



```
var http = require('http');

var server = http.createServer();
console.log('-- after createServer');

server.on('request', (request, response)=>{

    console.log('request');
    response.contentType='text/html';
    response.end('request came');

});
console.log('-- after sever.on  request');

server.on('connection', ()=>{
    console.log('connection');

});
console.log('-- after sever.on  connection');

server.listen(3000,()=>{
    console.log('listen');
});

console.log('Server running at http://localhost:3000/');
```

Node.js событийно ориентирован. То есть в коде мы подписываемся на какие-то события и когда они происходят, то вызывается коллбэк.

```
D:\PSCA\Lec03>node 03-03
-- after createServer
-- after sever.on  request
-- after sever.on  connection
Server running at http://localhost:3000/
listen
connection
request
connection
request
request
request
request
request
```

```
var http = require('http');
var url = require('url');
var fs = require('fs');

var fib = (n) => { return (n < 2 ? n : fib(n - 1) + fib(n - 2)); } // Фиббоначи

http.createServer(function (request, response) {
    let rc = JSON.stringify({ k: 0 });
    let path = url.parse(request.url).pathname;

    if (path === '/fib') {
        let param = url.parse(request.url, true).query.k;
        if (typeof param != 'undefined') {
            let k = parseInt(param);
            if (Number.isInteger(k)) {
                response.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
                response.end(JSON.stringify({ k: k, fib: fib(k) }));
            }
        }
    }
    else if (path === '/') {
        let html = fs.readFileSync('fib.html');
        response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
        response.end(html);
    }
    else {
        response.end(rc);
    }
}).listen(5000, () => console.log('Server running at http://localhost:5000/'));
```

Синхронное вычисление числа Фибоначчи

```
var http = require('http');
var url = require('url');
var fs = require('fs');

var fib = (n) => { return (n < 2 ? n : fib(n - 1) + fib(n - 2)); }

http.createServer(function (request, response) {
    let rc = JSON.stringify({ k: 0 });
    let path = url.parse(request.url).pathname;

    if (path === '/fib') {
        let param = url.parse(request.url, true).query.k;
        if (typeof param != 'undefined') {
            let k = parseInt(param);
            if (Number.isInteger(k)) {
                response.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
                process.nextTick(() => response.end(JSON.stringify({ k: k, fib: fib(k) })))
            }
        }
    }
    else if (path === '/') {
        let html = fs.readFileSync('fib.html');
        response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
        response.end(html);
    }
    else {
        response.end(rc);
    }
}).listen(5000, () => console.log('Server running at http://localhost:5000/'));
```

process.nextTick()

```
var http = require('http');
var url = require('url');
var fs = require('fs');

var fib = (n) => { return (n < 2 ? n : fib(n - 1) + fib(n - 2)); }

http.createServer(function (request, response) {
    let rc = JSON.stringify({ k: 0 });
    let path = url.parse(request.url).pathname;

    if (path === '/fib') {
        let param = url.parse(request.url, true).query.k;
        if (typeof param != 'undefined') {
            let k = parseInt(param);
            if (Number.isInteger(k)) {
                response.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
                setImmediate(() => response.end(JSON.stringify({ k: k, fib: fib(k) })))
            }
        }
    }
    else if (path === '/') {
        let html = fs.readFileSync('fib.html');
        response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
        response.end(html);
    }
    else {
        response.end(rc);
    }
}).listen(5000, () => console.log('Server running at http://localhost:5000/'));
```

setImmediate()

HTML-страница для измерения времени, затраченного на вычисление числа Фибоначчи

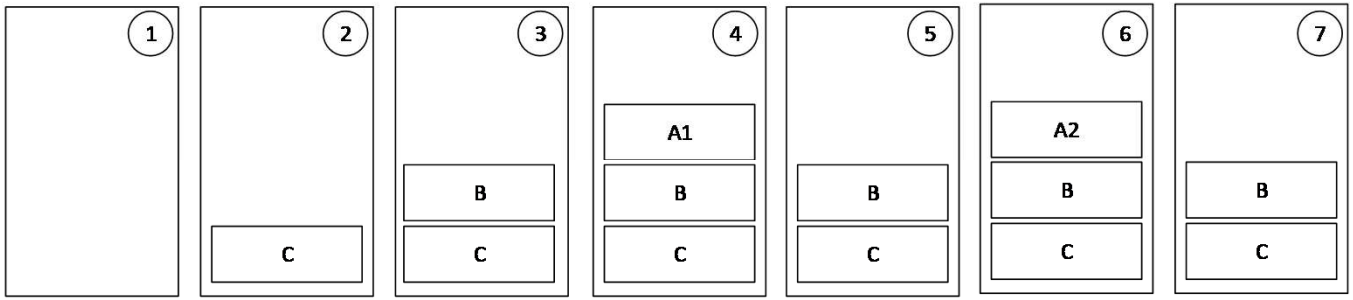
```
<!DOCTYPE html>
<html>

<head>
    <meta name="viewpoint" content="width=device-width" />
    <title>fibonacci</title>
</head>

<body>
    <div id='result'></div>
    <script>
        result.innerHTML = '';
        let n = 0;
        const d = Date.now();
        for (var k = 0; k < 20; k++) {
            fetch(`http://localhost:5000/fib?k=${k}`, {
                method: 'GET',
                mode: 'no-cors',
                headers: { 'Accept': 'application/json' }
            })
                .then(response => { return response.json(); })
                .then((pdata) => {
                    result.innerHTML += (n++) + '.Результат: ' + (Date.now() - d) + '-' + pdata.k + '/' + pdata.fib + '<br/>';
                });
        }
    </script>
</body>

</html>
```

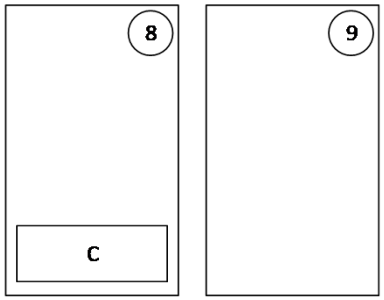
sync	nextTick	setImmediate
0.Результат: 32-0/0	0.Результат: 73-0/0	0.Результат: 161-0/0
1.Результат: 61-1/1	1.Результат: 73-1/1	1.Результат: 162-1/1
2.Результат: 61-2/1	2.Результат: 73-2/1	2.Результат: 163-2/1
3.Результат: 61-3/2	3.Результат: 73-3/2	3.Результат: 163-3/2
4.Результат: 80-4/3	4.Результат: 74-4/3	4.Результат: 163-4/3
5.Результат: 81-5/5	5.Результат: 74-5/5	5.Результат: 163-5/5
6.Результат: 81-6/8	6.Результат: 74-6/8	6.Результат: 163-6/8
7.Результат: 81-7/13	7.Результат: 100-7/13	7.Результат: 185-7/13
8.Результат: 82-8/21	8.Результат: 100-8/21	8.Результат: 185-8/21
9.Результат: 82-9/34	9.Результат: 100-9/34	9.Результат: 185-9/34
10.Результат: 83-10/55	10.Результат: 101-10/55	10.Результат: 185-10/55
11.Результат: 85-11/89	11.Результат: 101-11/89	11.Результат: 186-11/89
12.Результат: 86-12/144	12.Результат: 102-12/144	12.Результат: 186-12/144
13.Результат: 87-13/233	13.Результат: 105-13/233	13.Результат: 187-13/233
14.Результат: 87-14/377	14.Результат: 106-14/377	14.Результат: 187-14/377
15.Результат: 87-15/610	15.Результат: 106-15/610	15.Результат: 187-15/610
16.Результат: 87-16/987	16.Результат: 106-16/987	16.Результат: 187-16/987
17.Результат: 89-17/1597	17.Результат: 108-17/1597	17.Результат: 189-17/1597
18.Результат: 89-18/2584	18.Результат: 108-18/2584	18.Результат: 189-18/2584
19.Результат: 89-19/4181	19.Результат: 108-19/4181	19.Результат: 189-19/4181

Вывод: nextTick() срабатывает быстрее, чем setImmediate().

Принцип функционирования стека-вызовов

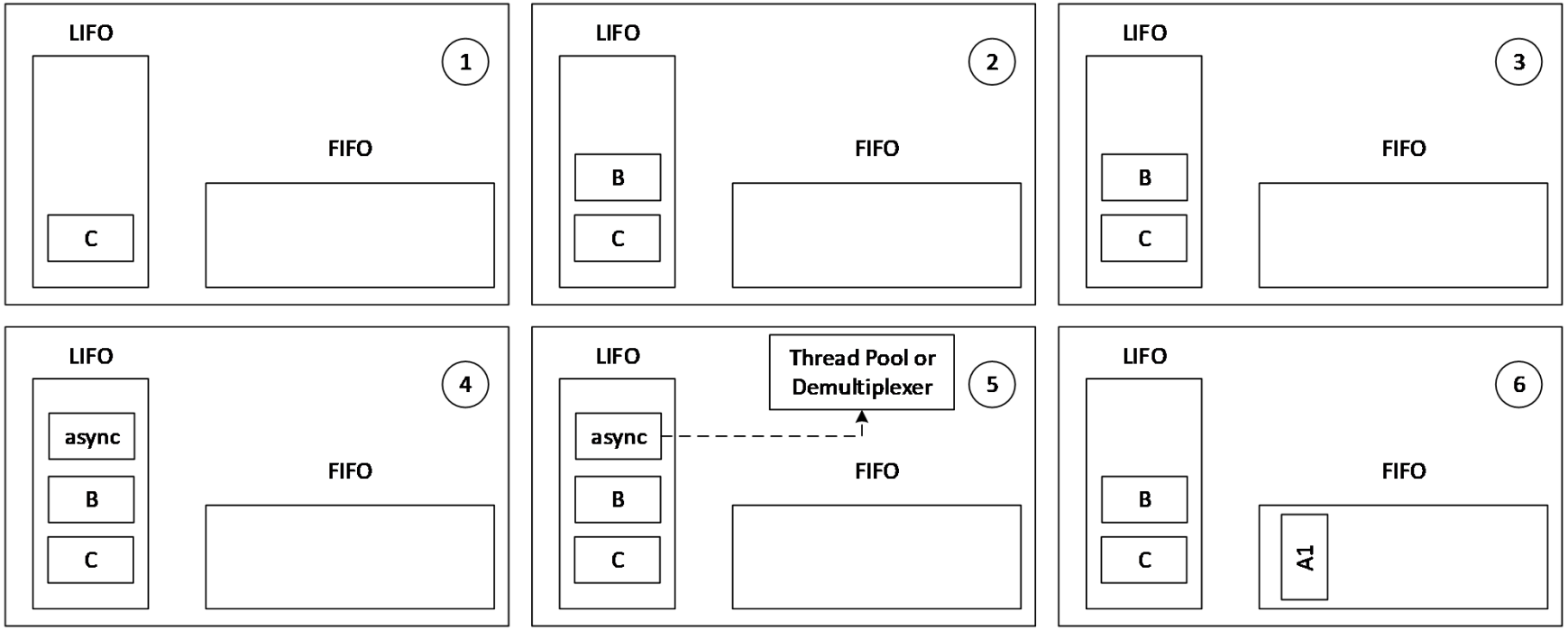
```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function B() { A1(); A2(); /* выполнение B */ }
function C() { B();        /* выполнение C */ }

C();
```





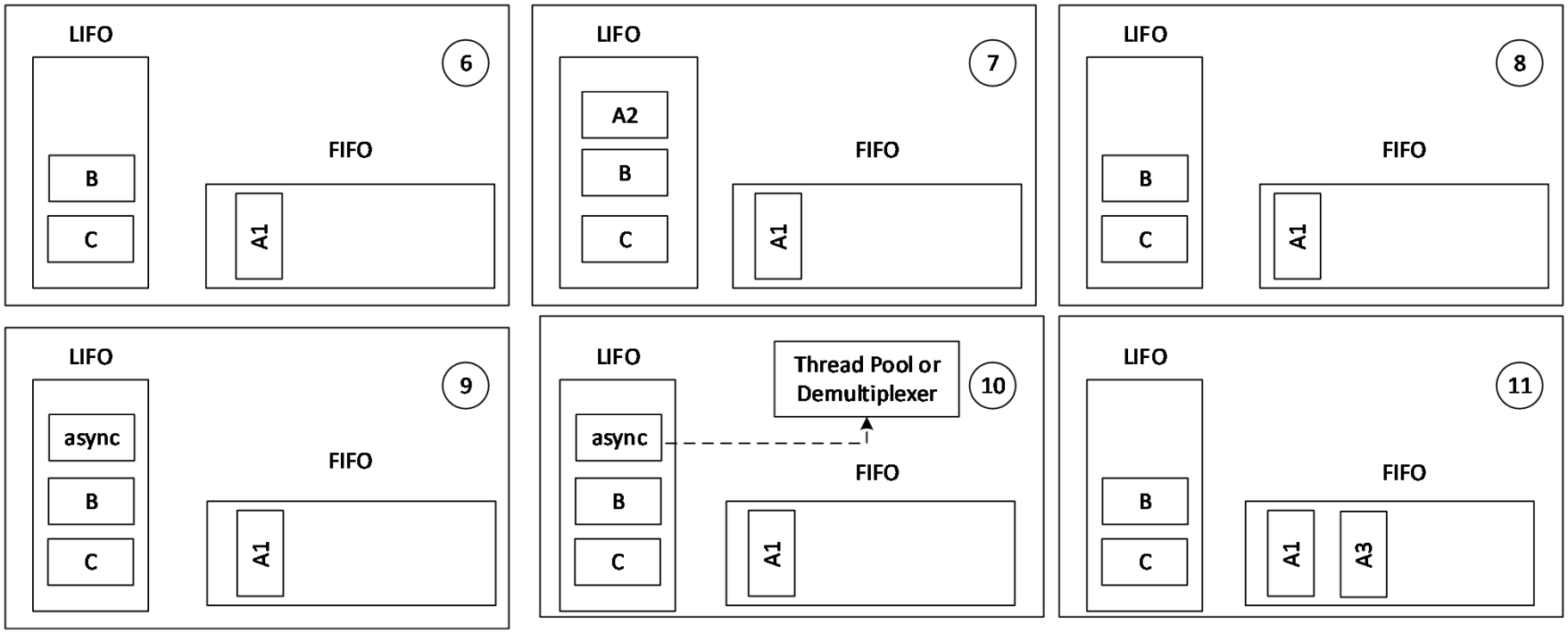
Принцип функционирования стека-вызовов и очереди коллбэков

```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { async(()=>{A1()}); A2(); async(()=>{A3()});  /* выполнение B */}
function C() { B(); /* выполнение C */}

C();
```



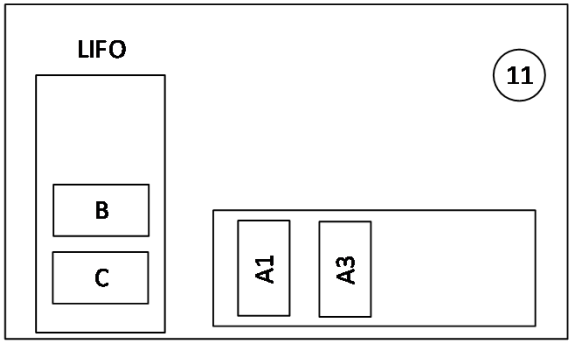
Принцип функционирования стека-вызовов и очереди коллбэков

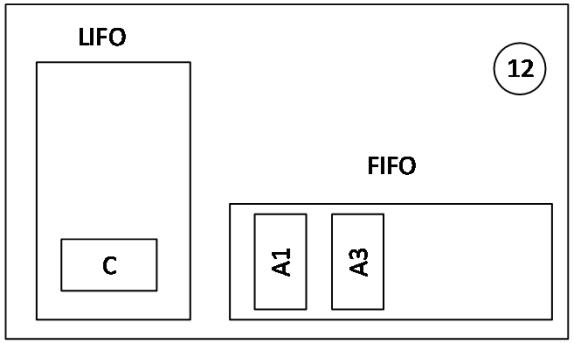
```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { async(()=>{A1()}); A2(); async(()=>{A3()});  /* выполнение B */}
function C() { B(); /* выполнение C */}

C();
```

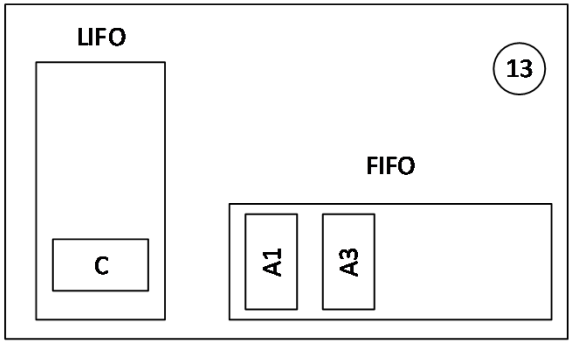


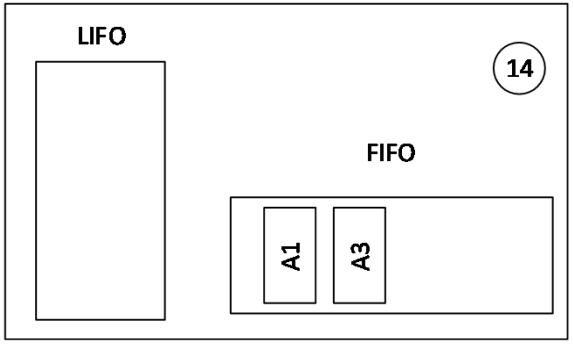
Принцип функционирования стека-вызовов и очереди коллбэков

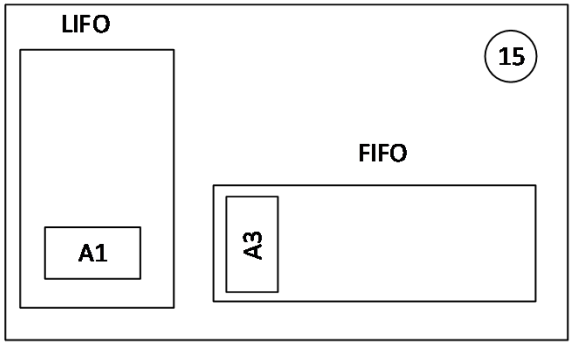
```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { async(()=>{A1()}); A2(); async(()=>{A3()});  /* выполнение B */}
function C() { B(); /* выполнение C */}

C();
```

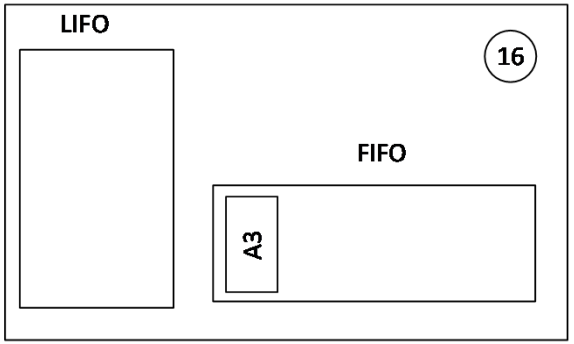






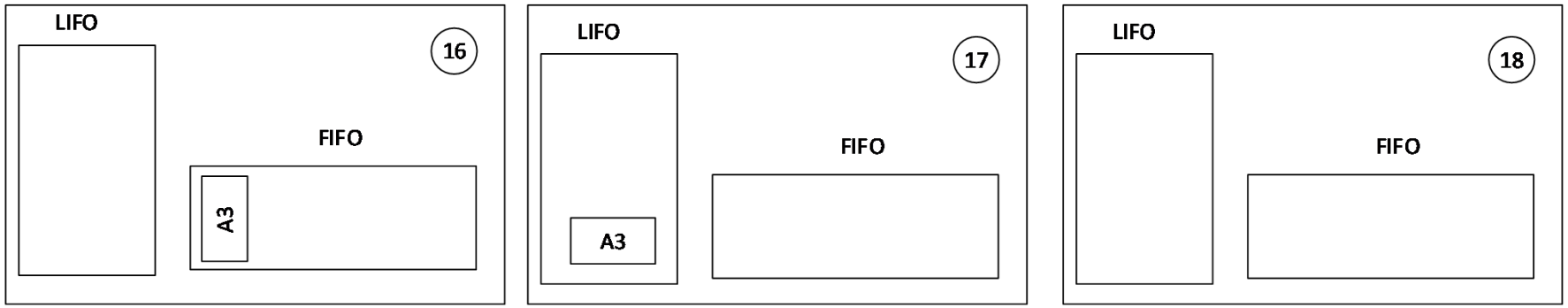




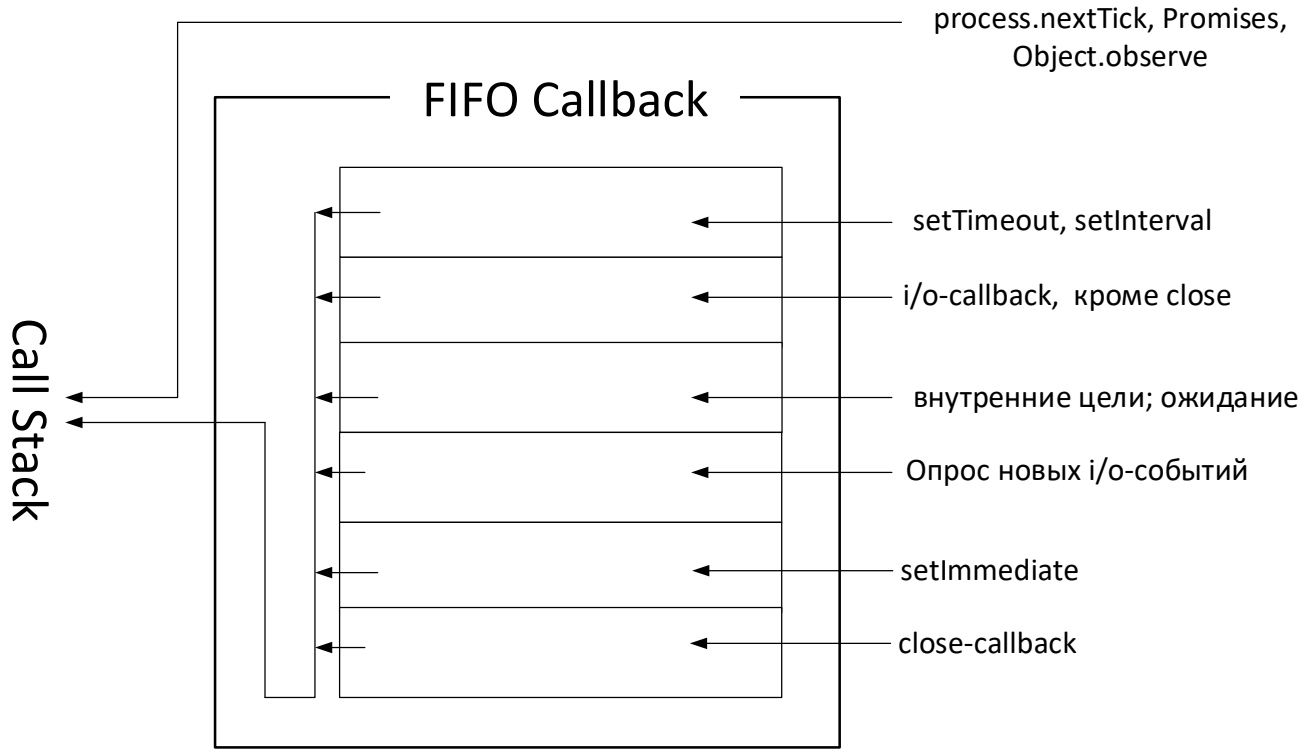


Принцип функционирования стека-вызовов и очереди коллбэков

```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { sync(()=>{A1()}); A2(); sync(()=>{A3)}  /* выполнение B */ }
function C() { B();  /* выполнение C */ }

C();
```



Обзор фаз event loop’a



- макрозадачи – выполняются по одной за один проход цикла (*setTimeout, setInterval, setImmediate, requestAnimationFrame, I/O, UI rendering*);
- микрозадачи – на каждом проходе цикл выполняет все накопившееся (*process.nextTick, Object.observe, Promises*).