This cute Node.js

Alexander Lobashev, Raiffeisen Bank

t.me/alobashev

Architecture

Event Loop, I/O

Library

Architecture

Event loop, I/O

Library


node
JS

Node is a js runtime platform

Written in C, C++ and js

Built on Chrome V8 Engine

Uses libuv for I/O

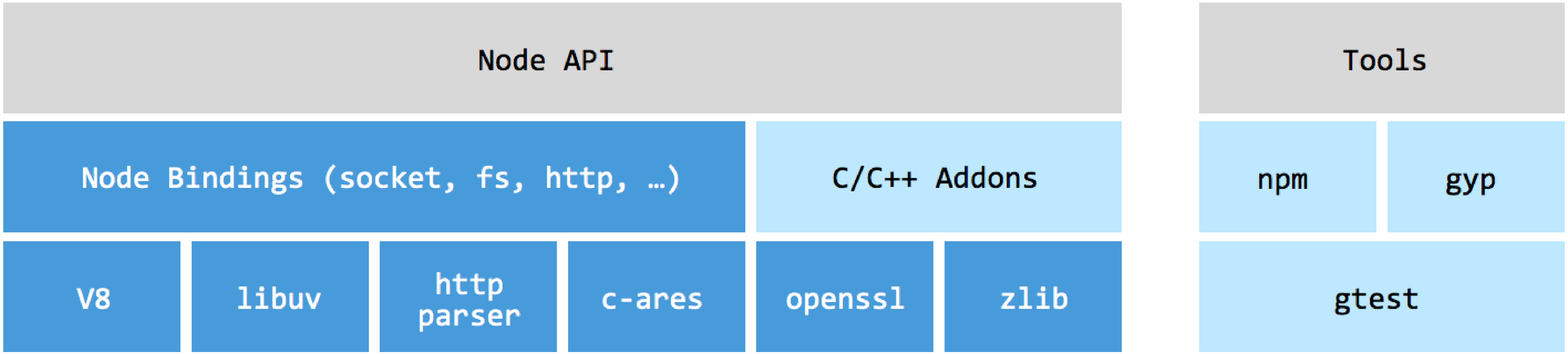
Used everywhere (web, mobile, robots, IoT, …)

Goals

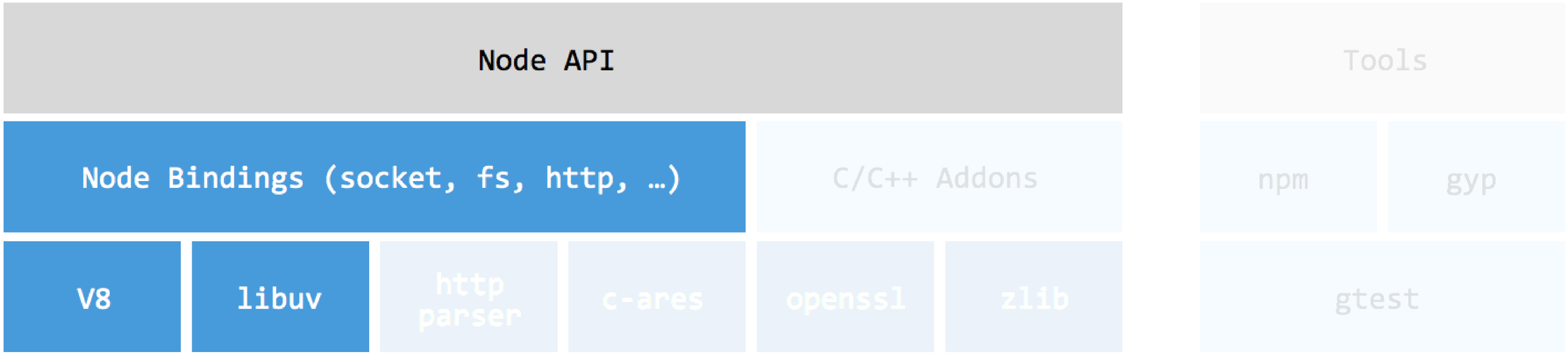
Single Threaded

Event-Driven Model

Non-Blocking I/O



https://nodejs.org/en/docs/meta/topics/dependencies *


Node API
Tools
Node Bindings (socket, fs, http, …)
C/C++ Addons
npm
gyp
V8
libuv
http parser
c-ares
openssl
zlib
gtest

Written C, C++

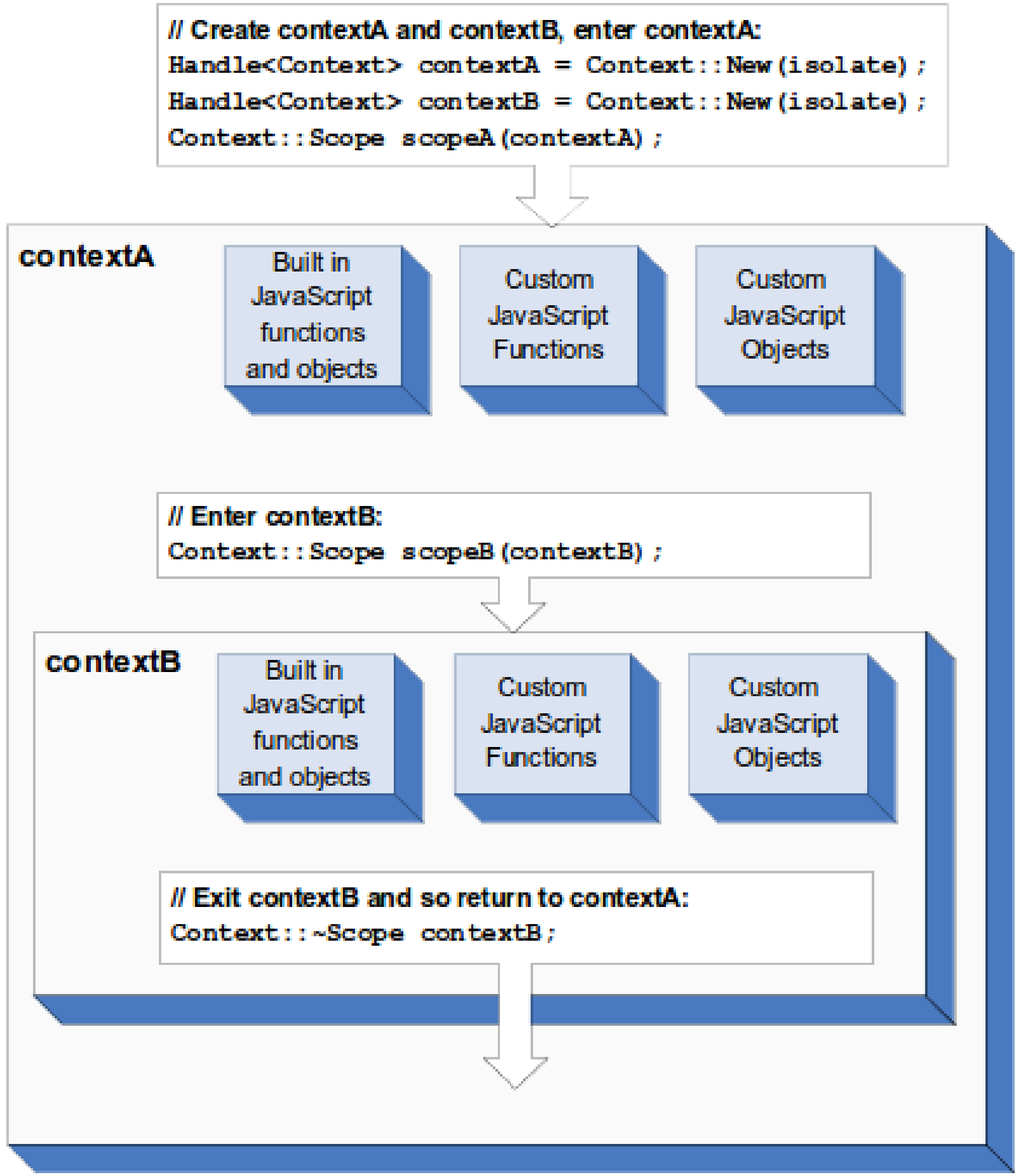
Bridge System C++ and JS

Drop custom EventLoop

…

C++ > V8 > Run JS

```
  Local<Context> context = Context::New(isolate);

  // Делаем этот контекст активным для компиляции и запуска кода JavaScript
  Context::Scope context_scope(context);

  // Создаем новую строку из исходного кода JavaScript
  // Он может быть жестко запрограммирован, считан из файла или любым другим способом.
  Local<String> source = String::NewFromUtf8(isolate, "'Hello' + ', World!'", NewStringType::kNormal).ToLo

  // Компилируем наш JavaScript
  Local<Script> script = Script::Compile(context, source).ToLocalChecked();

  // Запускаем скомпилированный код и ожидаем результатов
  Local<Value> result = script->Run(context).ToLocalChecked();

  // Преобразуем результат в utf8 и выводим
  String::Utf8Value utf8(result);
* https://chromium.googlesource.com/v8/v8/+/branch-heads/5.8/samples/hello-world.cc
  printf("%s\n", *utf8);
* https://gist.github.com/netpoetica/28ce31478cfc43edcaa7
```


// Create contextA and contextB, enter contextA:
Handle<Context> contextA = Context::New(isolate);
Handle<Context> contextB = Context::New(isolate);
Context::Scope scopeA(contextA);
contextA
Built in JavaScript functions and objects
Custom JavaScript Functions
Custom JavaScript Objects
// Enter contextB:
Context::Scope scopeB(contextB);
contextB
Built in JavaScript functions and objects
Custom JavaScript Functions
Custom JavaScript Objects
// Exit contextB and so return to contextA:
Context::~Scope contextB;

Templates

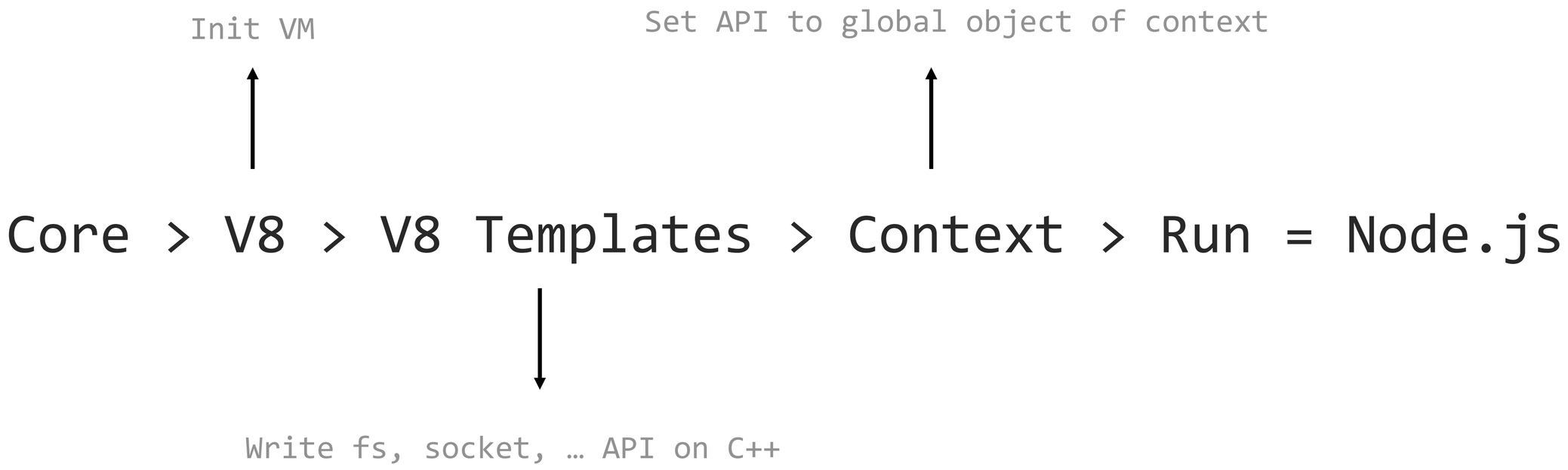
Bridge C++ code to JS (e.g. DOM)

Function Templates

Object Templates

```
// Функция LogCallback
static void LogCallback(const v8::FunctionCallbackInfo<v8::Value>& args) {
    // Некоторый код
}

// Создаем новый ObjectTemplate
Local<ObjectTemplate> global = ObjectTemplate::New(isolate);

// Создаем новый FunctionTemplate и связываем C++ функцию LogCallback с ним
// Добавляем в global наш FunctionTemplate
global->Set(
    String::NewFromUtf8(isolate, "log"),
    FunctionTemplate::New(isolate, LogCallback)
);

// Передаем переменную global в новый контекст js, и мы можем вызвать функцию из глобальной области
// В результате будет запущен метод C++ из js.
Persistent<Context> context = Context::New(isolate, NULL, global);
```

* https://github.com/nodejs/node/blob/master/deps/v8/samples/process.cc


Init VM
Set API to global object of context
Core > V8 > V8 Templates > Context > Run = Node.js
Write fs, socket, … API on C++

Init VM

Set API to global object of context

Core > V8 > V8 Templates > Context > Run ≠ Node.js

Write fs, socket, … API on C++

Event Loop

Module Loader

Architecture

Event loop, I/O

Library

Node API

Tools

Node Bindings (socket, fs, http, ...)

C/C++ Addons

npm

gyp

V8

libuv

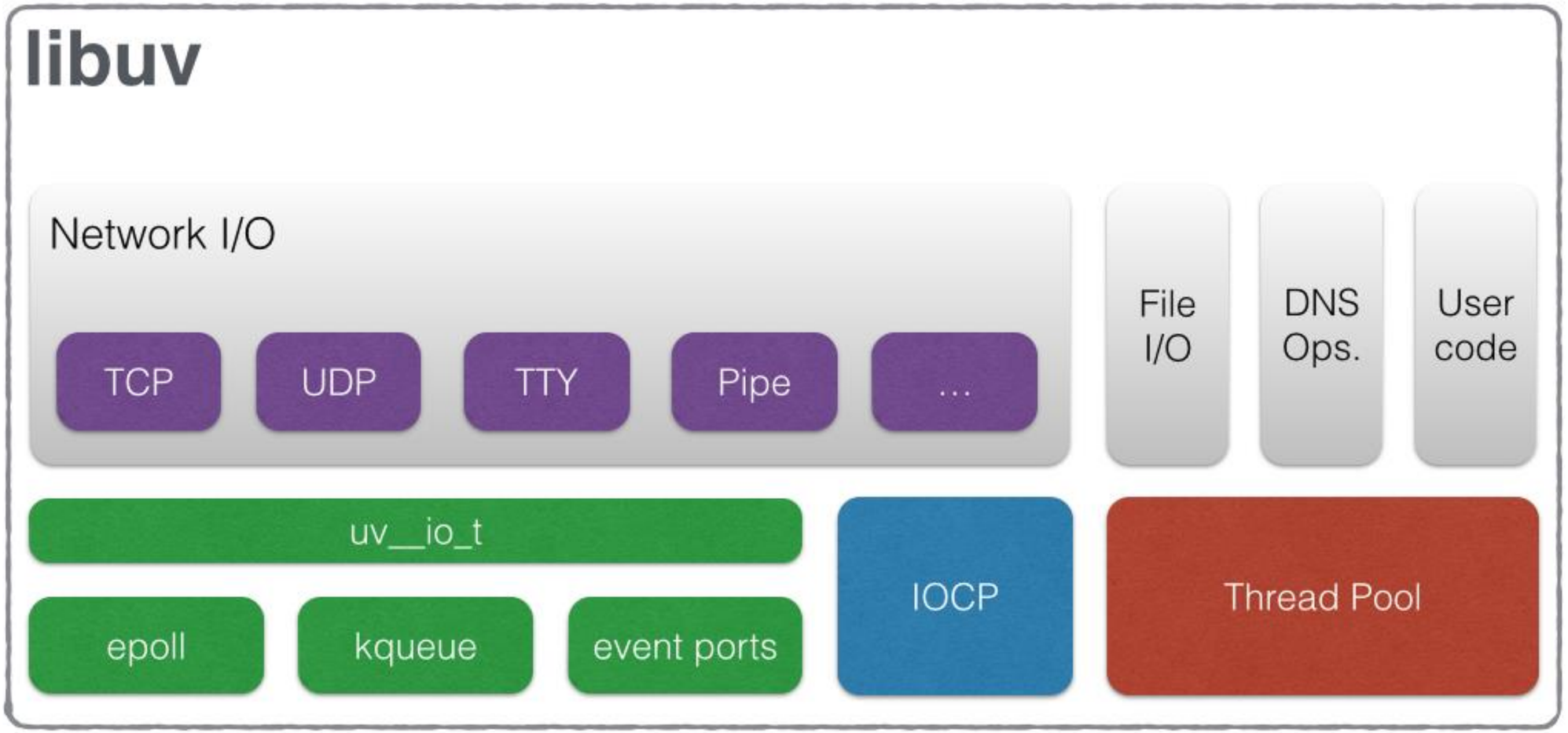
http parser

c-ares

openssl

zlib

gtest


libuv
Network I/O
TCP
UDP
TTY
Pipe
...
File I/O
DNS Ops.
User code
uv__io_t
epoll
kqueue
event ports
IOCP
Thread Pool

Event loop

Event loop

Init with start Node

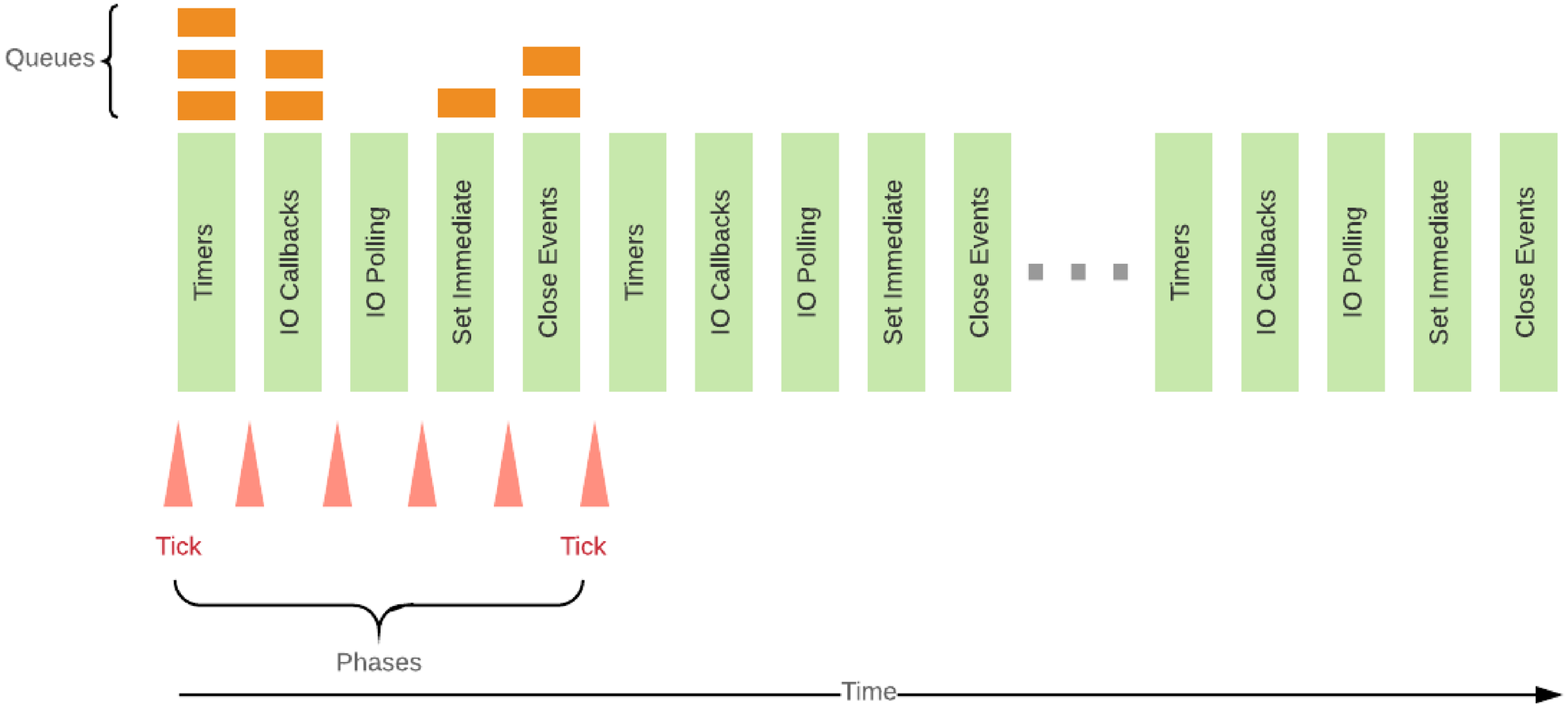
Use main thread

Handler multiple concurrent clients

The code execution mechanism

Starting run after exec main module

```
// Всё просто и сложно :)
while (queue.waitForMessage()) {
    queue.processNextMessage();
}
```


Queues
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Tick
Tick
Phases
Time

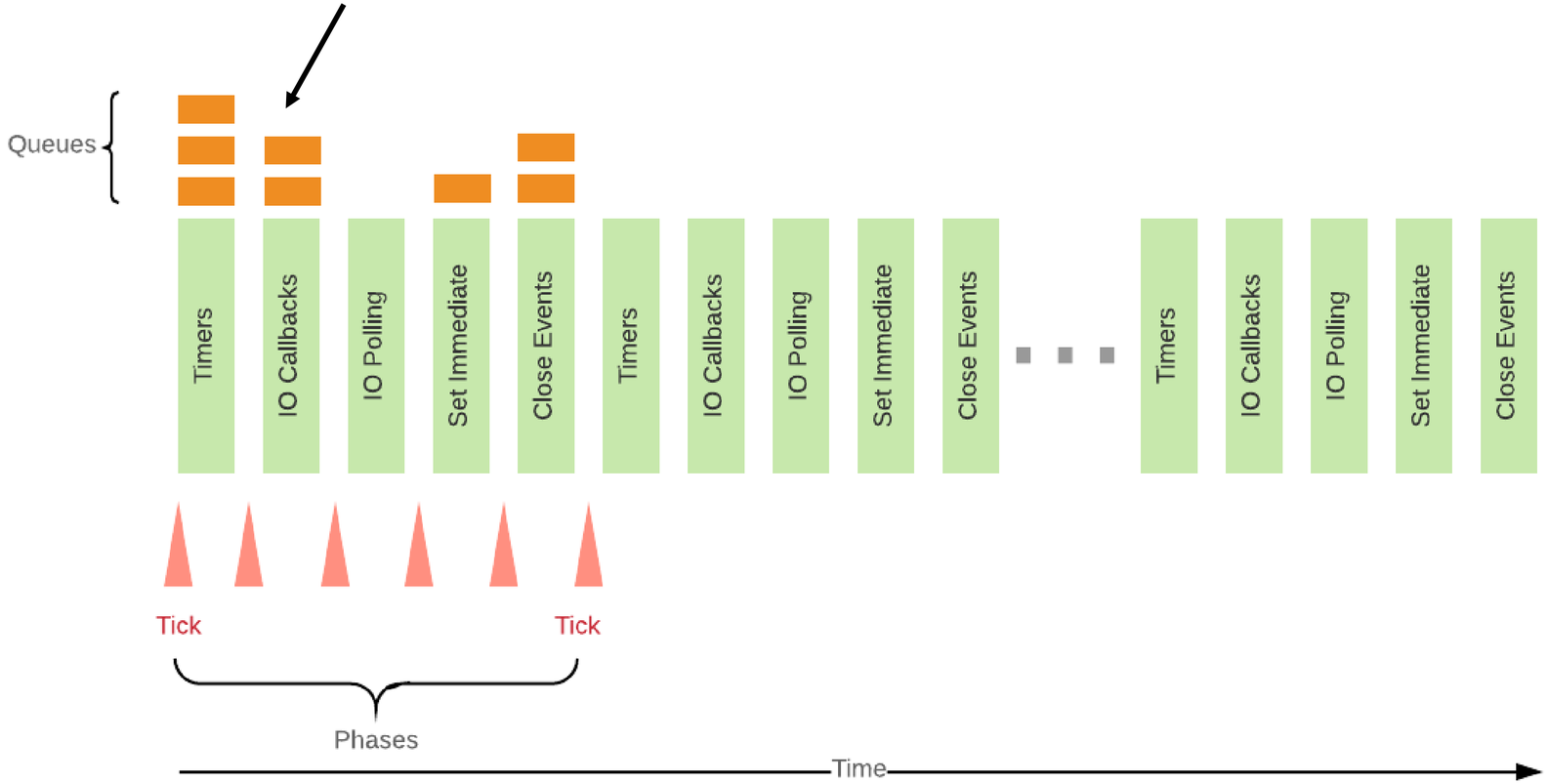
```
// a.js
module.exports = () => {
    console.log('module a');

    require('fs').readFile(__filename, () => {
        setTimeout(() => {
            console.log('timeout');
        }, 0);

        setImmediate(() => {
            console.log('immediate');
        });
    });
};

// main.js
const a = require('./a');

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

```
> Example
> module a
> immediate
> timeout
```

Run module a.js (console.log + readFile)

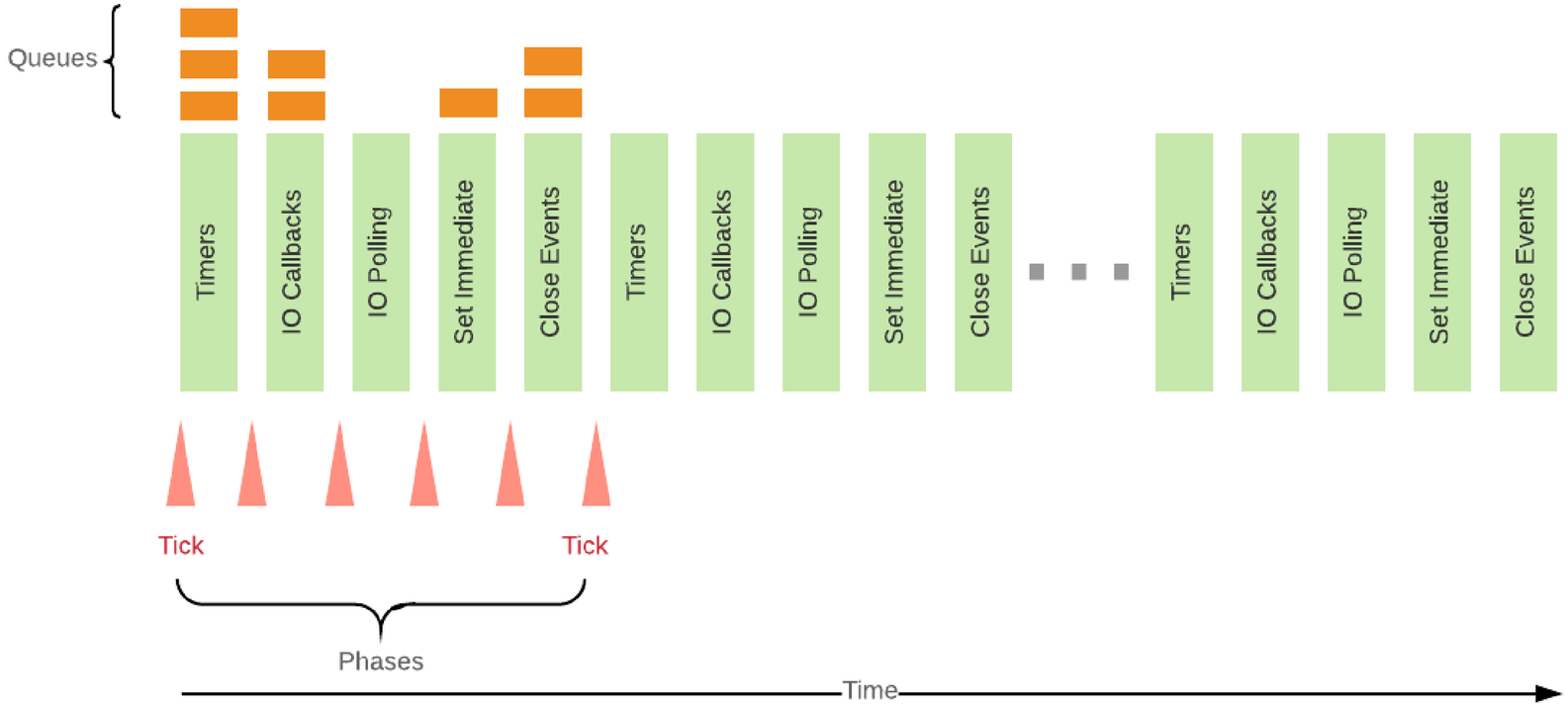
1 Iteration

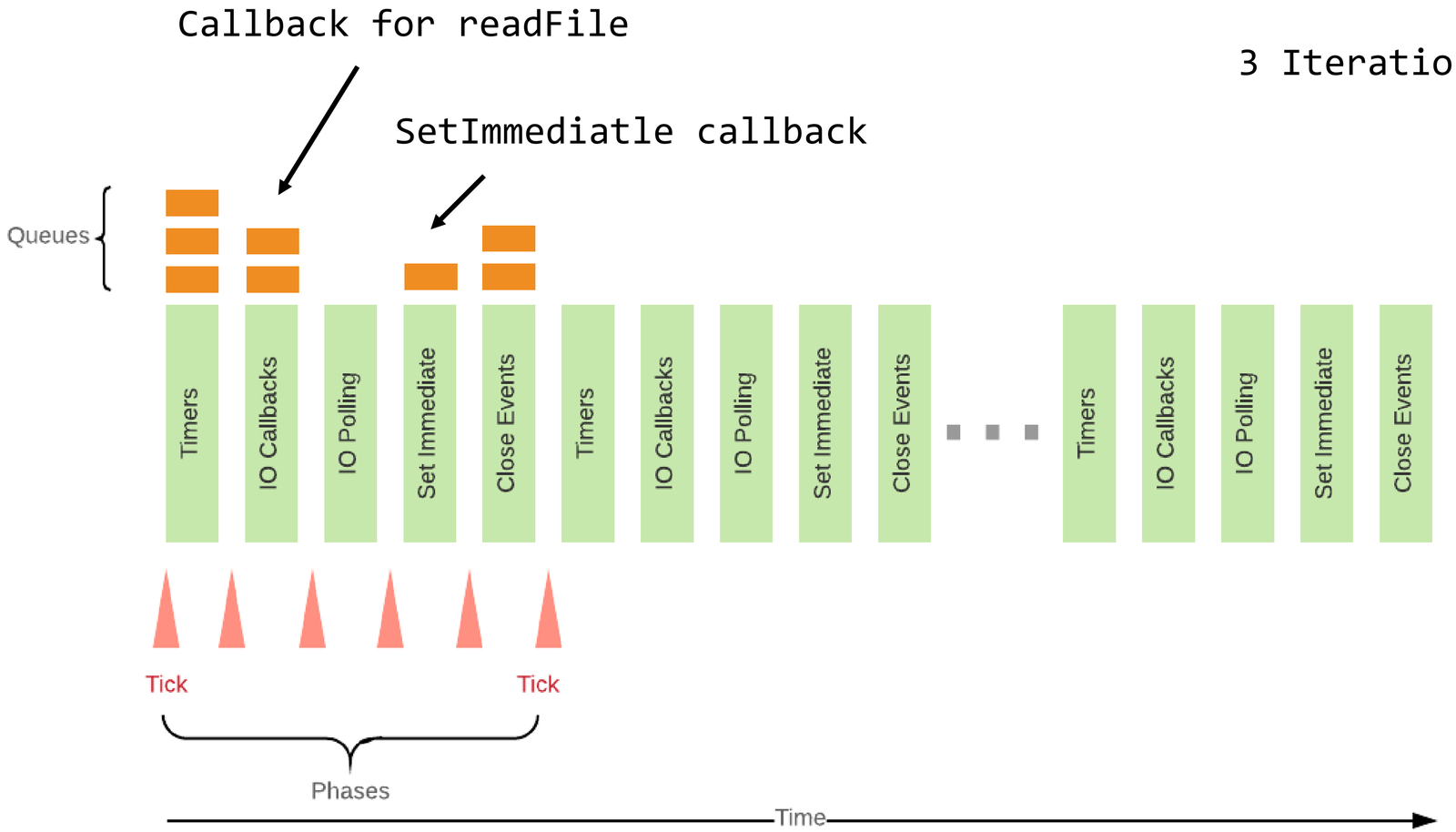
Main file
> log: example



2 Iteration

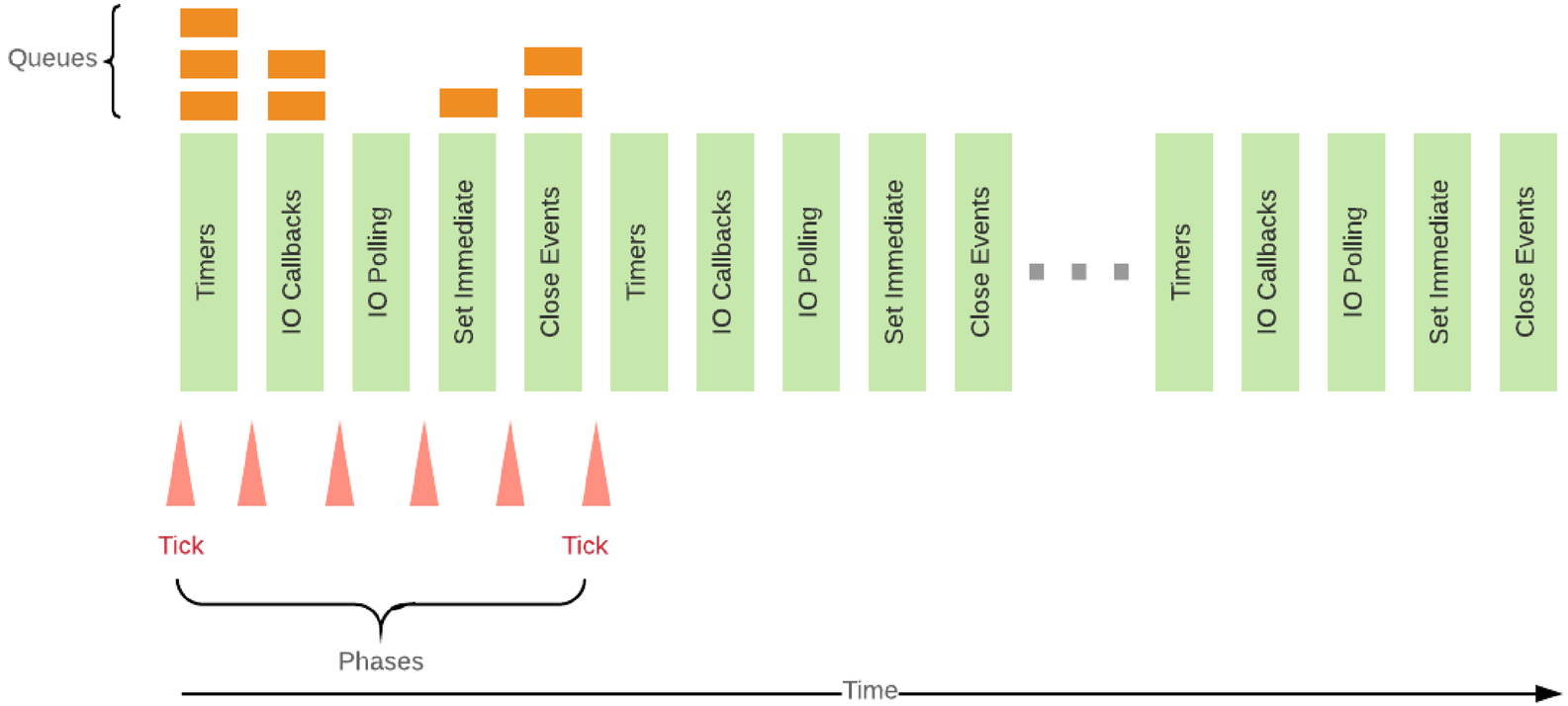
Waiting readFile




Callback for readFile
3 Iteration
SetImmediatle callback
Queues
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Timers
IO Callbacks
IO Polling
Set Immediate
Close Events
Tick
Tick
Phases
Time

Callback for setTimeout

4 Iteration



Phases in Detail

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

```
require('fs').readFile(__filename, (err, data) => {
    console.log(data);
});
```

Non-blocking I/O

```
const data = require('fs').readFileSync(__filename);
console.log(data);
```

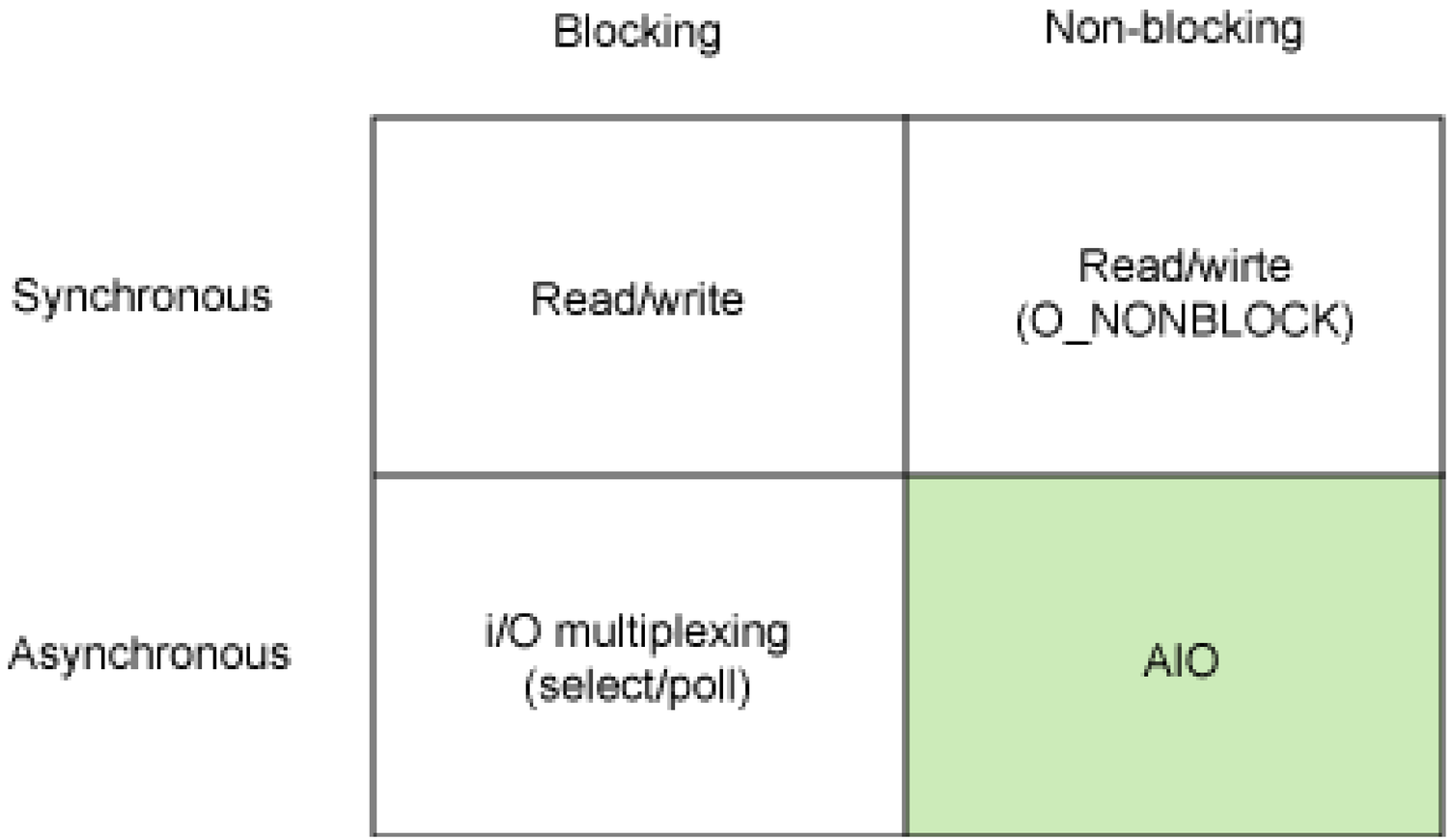
Blocking I/O

Async* I/O

I/O

Blocking I/O

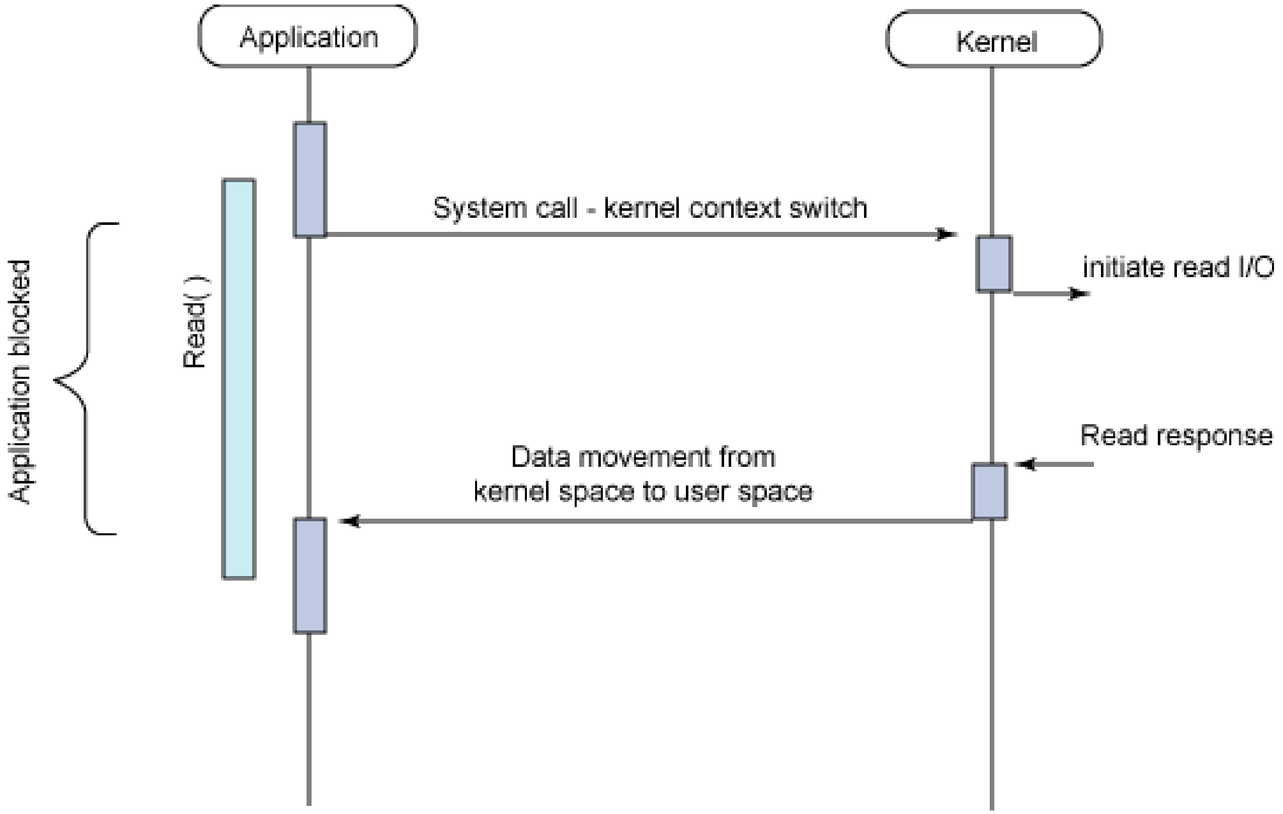
Non-Blocking I/O


Blocking
Non-blocking
Synchronous
Read/write
Read/wirte
(O_NONBLOCK)
Asynchronous
i/O multiplexing
(select/poll)
AIO

Blocking I/O

```
Start I/O

Wait for I/O finish

Do something else

Do more
```

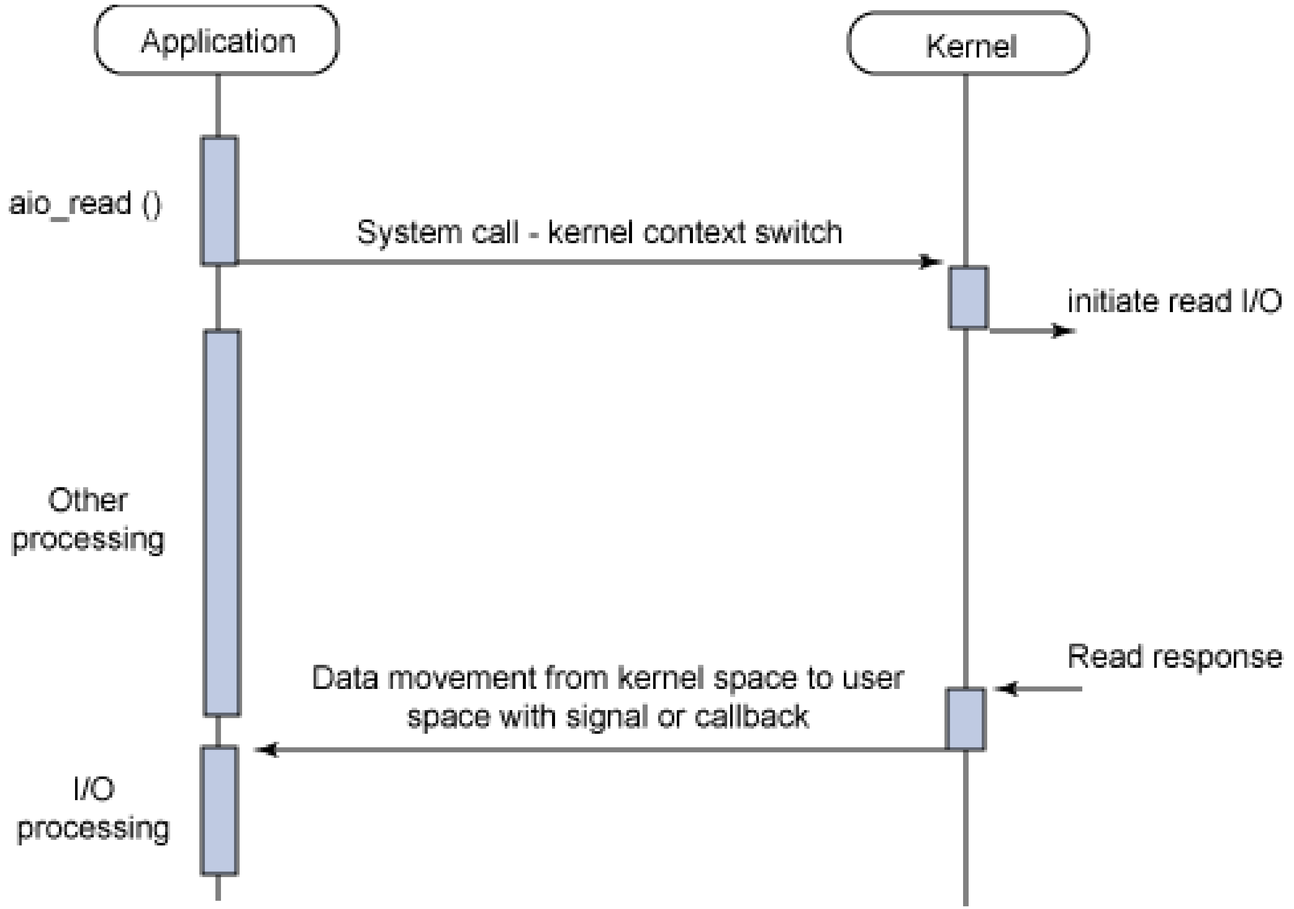
Blocking I/O


Application
Kernel
System call - kernel context switch
initiate read I/O
Application blocked
Read( )
Read response
Data movement from
kernel space to user space

Non-Blocking I/O

```
Start I/O

Do something else

Wait for I/O finish

Do more
```

Non-Blocking I/O



```cpp
void Environment::Start(int argc, const char* const* argv, int exec_argc, const char* const* exec_argv,
  HandleScope handle_scope(isolate());
  Context::Scope context_scope(context());

  uv_check_init(event_loop(), immediate_check_handle());
  uv_unref(reinterpret_cast<uv_handle_t*>(immediate_check_handle()));
  uv_idle_init(event_loop(), immediate_idle_handle());
  uv_check_start(immediate_check_handle(), CheckImmediate);
  uv_prepare_init(event_loop(), &idle_prepare_handle_);
  uv_check_init(event_loop(), &idle_check_handle_);
  uv_unref(reinterpret_cast<uv_handle_t*>(&idle_prepare_handle_));
  uv_unref(reinterpret_cast<uv_handle_t*>(&idle_check_handle_));

  // Register handle cleanups
  env->RegisterHandleCleanup(reinterpret_cast<uv_handle_t*>(env->immediate_check_handle()), HandleClear
  env->RegisterHandleCleanup(reinterpret_cast<uv_handle_t*>(env->immediate_idle_handle()), HandleCleanu
  env->RegisterHandleCleanup(reinterpret_cast<uv_handle_t*>(env->idle_prepare_handle()), HandleCleanup,
```

* https://github.com/nodejs/node/blob/master/src/env.cc

Problems

Memory leaks

Crash by single error

Callback hall

Architecture

Event Loop, I/O

Library

Library

Modules

Runtime Library

Module Loader

```
struct node_module {
  int nm_version;
  unsigned int nm_flags;
  void* nm_dso_handle;
  const char* nm_filename; // Имя файла
  node::addon_register_func nm_register_func; // Функция модуля, для ObjectTemplate
  node::addon_context_register_func nm_context_register_func;
  const char* nm_modname; // Имя модуля (fs, os, http)
  void* nm_priv;
  struct node_module* nm_link;
};
```

* https://github.com/nodejs/node/blob/master/src/node.h

```
  struct node_module* mp = reinterpret_cast<struct node_module*>(m);

  if (mp->nm_flags & NM_F_BUILTIN) {
    mp->nm_link = modlist_builtin;
    modlist_builtin = mp;
  } else if (mp->nm_flags & NM_F_INTERNAL) {
    mp->nm_link = modlist_internal;
    modlist_internal = mp;
  } else if (!node_is_initialized) {
    // "Linked" modules are included as part of the node project.
    // Like builtins they are registered *before* node::Init runs.
    mp->nm_flags = NM_F_LINKED;
    mp->nm_link = modlist_linked;
    modlist_linked = mp;
  } else {
    modpending = mp;
  }
```

* https://github.com/nodejs/node/blob/master/src/node.cc

Binding

```
  // Имя модуля
  Local<String> module = args[0].As<String>();
  node::Utf8Value module_v(env->isolate(), module);

  // Ищем модуль в нашей структуры модулей
  node_module* mod = get_builtin_module(*module_v);
  // Определяем объект exports
  Local<Object> exports;
  // Инициализируем модуль
  if (mod != nullptr) {
    exports = InitModule(env, mod, module);
  } else if (!strcmp(*module_v, "constants")) {
    exports = Object::New(env->isolate());
    CHECK(exports->SetPrototype(env->context(),
                                Null(env->isolate())).FromJust());
    DefineConstants(env->isolate(), exports);
  } else if (!strcmp(*module_v, "natives")) {
    exports = Object::New(env->isolate());
    DefineJavaScript(env, exports);
  } else {
* https://github.com/nodejs/node/blob/master/src/node.cc
    return ThrowIfNoSuchModule(env, *module_v);
```

```
// Привязываем функцию GetBinging, чтобы была доступна в JS
v8::Local<v8::Function> get_binding_fn =
      env->NewFunctionTemplate(GetBinding)->GetFunction(env->context())
          .ToLocalChecked();

// Добавляем «связывание» в глобальных объект process
process.binding = function binding(module) {
    module = String(module);
    let mod = bindingObj[module];
    if (typeof mod !== ‘object') {
    mod = bindingObj[module] = getBinding(module);
      moduleLoadList.push(`Binding ${module}`);
    }
    return mod;
};
```

* https://github.com/nodejs/node/blob/master/src/node.cc
* https://github.com/nodejs/node/blob/master/lib/internal/bootstrap/loaders.js

Bindings

process.binding

process._linkedBinding

internalBinding

```
// Загружаем нативный модуль fs на C++
const binding = process.binding('fs');

// Функция fs.access
fs.access = function(path, mode, callback) {
    if (typeof mode === 'function') {
        callback = mode;
        mode = fs.F_OK;
    }

    path = getPathFromURL(path);
    validatePath(path);

    mode = mode | 0;
    var req = new FSReqWrap();
    req.oncomplete = makeCallback(callback);
    // Вызов нативного модуля
    binding.access(pathModule.toNamespacedPath(path), mode, req);
* https://github.com/nodejs/node/blob/master/src/node.cc
* }; https://github.com/nodejs/node/blob/master/lib/internal/bootstrap/loaders.js
```

require()

NativeModule

```
    this.loading = false;
  }

  NativeModule.require = function(id) {
    if (id === loaderId) {
      return loaderExports;
    }

    const cached = NativeModule.getCached(id);
    if (cached && (cached.loaded || cached.loading)) {
      return cached.exports;
    }

    if (!NativeModule.exists(id)) {
      const err = new Error(`No such built-in module: ${id}`);
      err.code = 'ERR_UNKNOWN_BUILTIN_MODULE';
      err.name = 'Error [ERR_UNKNOWN_BUILTIN_MODULE]';
      throw err;
    }

    moduleLoadList.push(`NativeModule ${id}`);
    const nativeModule = new NativeModule(id);
```

* https://github.com/nodejs/node/blob/master/lib/internal/bootstrap/loaders.js

```
    try {
        const fn = runInThisContext(source, {
            filename: this.filename,
            lineOffset: 0,
            displayErrors: true
        });

        const requireFn = this.id.startsWith('internal/deps/') ?
            NativeModule.requireForDeps :
            NativeModule.require;

        fn(this.exports, requireFn, this, process);
        this.loaded = true;
    } finally {
        this.loading = false;
    }
};

// Minimal sandbox helper
* https://github.com/nodejs/node/blob/master/lib/internal/bootstrap/loaders.js
const ContextifyScript = process.binding('contextify').ContextifyScript;
```

```
NativeModule.wrap = function(script) {
    return NativeModule.wrapper[0] + script + NativeModule.wrapper[1];
};

NativeModule.wrapper = [
    '(function (exports, require, module, process) {',
    '\n});'
];
```

* https://github.com/nodejs/node/blob/master/lib/internal/bootstrap/loaders.js

Module

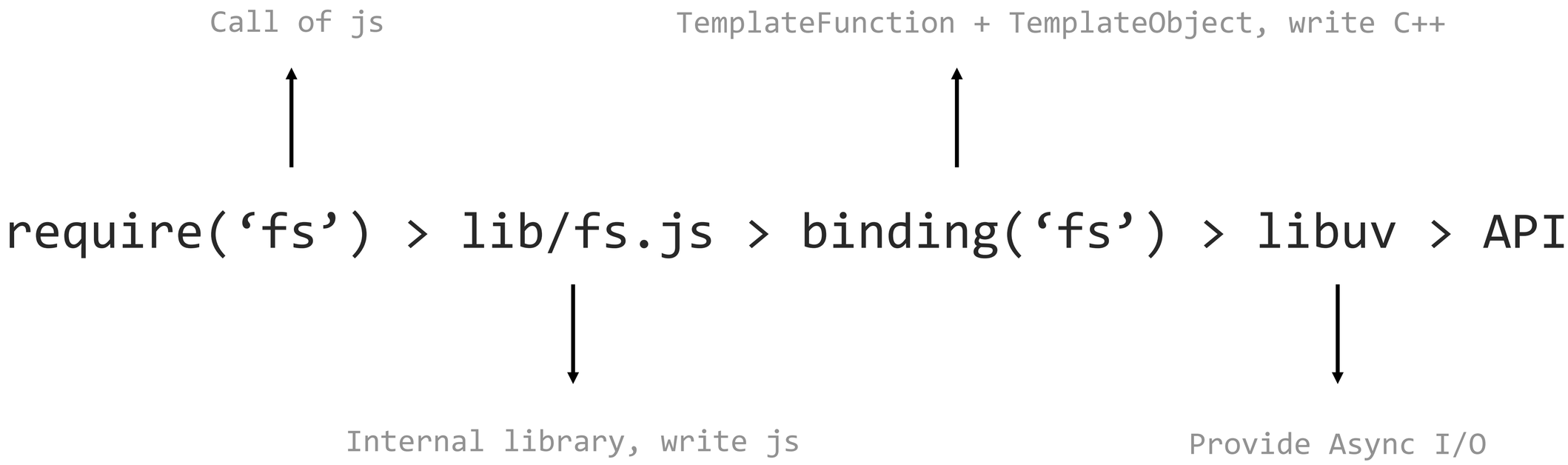
```
};
// Native extension for .js
Module._extensions['.js'] = function(module, filename) {
    var content = fs.readFileSync(filename, 'utf8');
    module._compile(stripBOM(content), filename);
};
// Native extension for .json
Module._extensions['.json'] = function(module, filename) {
    var content = fs.readFileSync(filename, 'utf8');
    try {
        module.exports = JSON.parse(stripBOM(content));
    } catch (err) {
        err.message = filename + ': ' + err.message;
        throw err;
    }
};
//Native extension for .node
Module._extensions['.node'] = function(module, filename) {
    return process.dlopen(module, path.toNamespacedPath(filename));
};
```

*https://github.com/nodejs/node/blob/master/lib/internal/modules/cjs/loader.js

```
// Returns exception, if any.
Module.prototype._compile = function(content, filename) {
    content = stripShebang(content);
    // create wrapper function
    var wrapper = Module.wrap(content);

    var compiledWrapper = vm.runInThisContext(wrapper, {
        filename: filename,
        lineOffset: 0,
        displayErrors: true
    });

    var dirname = path.dirname(filename);
    var require = makeRequireFunction(this);
    var depth = requireDepth;
    if (depth === 0) stat.cache = new Map();
    var result = compiledWrapper.call(this.exports, this.exports, require, this, filename, dirname);
```

* https://github.com/nodejs/node/blob/master/lib/internal/modules/cjs/loader.js

```
    if (depth === 0) stat.cache = null;
```


Call of js
TemplateFunction + TemplateObject, write C++
require(‘fs’) > lib/fs.js > binding(‘fs’) > libuv > API
Internal library, write js
Provide Async I/O

Runtime Library

Runtime Library

```
$ node -e "console.log(Object.keys(global));"
```

process (exit, kill, cwd, hrtime, …)

global

console

…

Summary

Learned how works nodejs

Blocking, Non-Blocking I/O

Write own modules loader

How works require()

Thanks

Alexander Lobashev, Raiffeisen Bank

t.me/alobashev