NODE+JS INTERACTIVE

December 11–12, 2019
Montreal, Canada

Next Generation N-API

A hands-on workshop

N-API Team

About Gabriel Schulhof

- Works at Intel
- Involved with the API working group
    - Promises
    - Exception handling
    - Environment propagation
    - Module loading
    - Wrap/Unwrap
- GitHub: @gabrielschulhof
- Email: gabriel.schulhof@intel.com

About Michael Dawson
IBM Community Lead for Node.js

- Active Node.js community member
  - Technical Steering Committee TSC member
  - Community Committee member
  - n-api, build, security, benchmarking, diagnostics, release, user-feedback, teams and WGs.

Twitter: @mhdawson1

GitHub: @mhdawson
Linkedin: https://www.linkedin.com/in/michael-dawson-6051282

About Jim Schlight

inspiredware

- Head of a consultancy based in Ashland, Oregon
- Member of the N-API Working Group
  - node-pre-gyp
  - prebuild
  - Documentation
- GitHub: @jschlight
- Twitter: @inspiredware

About Nicola Del Gobbo

- Developer at Packly
- Member of the N-API Working Group
- GitHub: @NickNaso
- Twitter: @NickNaso
- Linkedin: https://it.linkedin.com/in/ndelgobbo

Contributors

Anna Henningsen
@addaleax

Gabriel Schulhof
@gabrielschulhof

Hitesh Kanwathirtha
@digitalinfinity

Jim Schlight
@jschlight

Micheal Dawson
(@mhdawson)

Nicola Del Gobbo
@NickNaso

Kevin Eady
@KevinEady

Arunesh Chandra
@aruneshchandra

Taylor Wall
@boingoing

Anisha Rohra
@anisha-rohra

Kyle Farnung
@kfarnung

And many others ...

Objectives of the workshop

- Orientation to N-API
- Awareness of available tools and processes
- A good start on your own projects

Workshop schedule

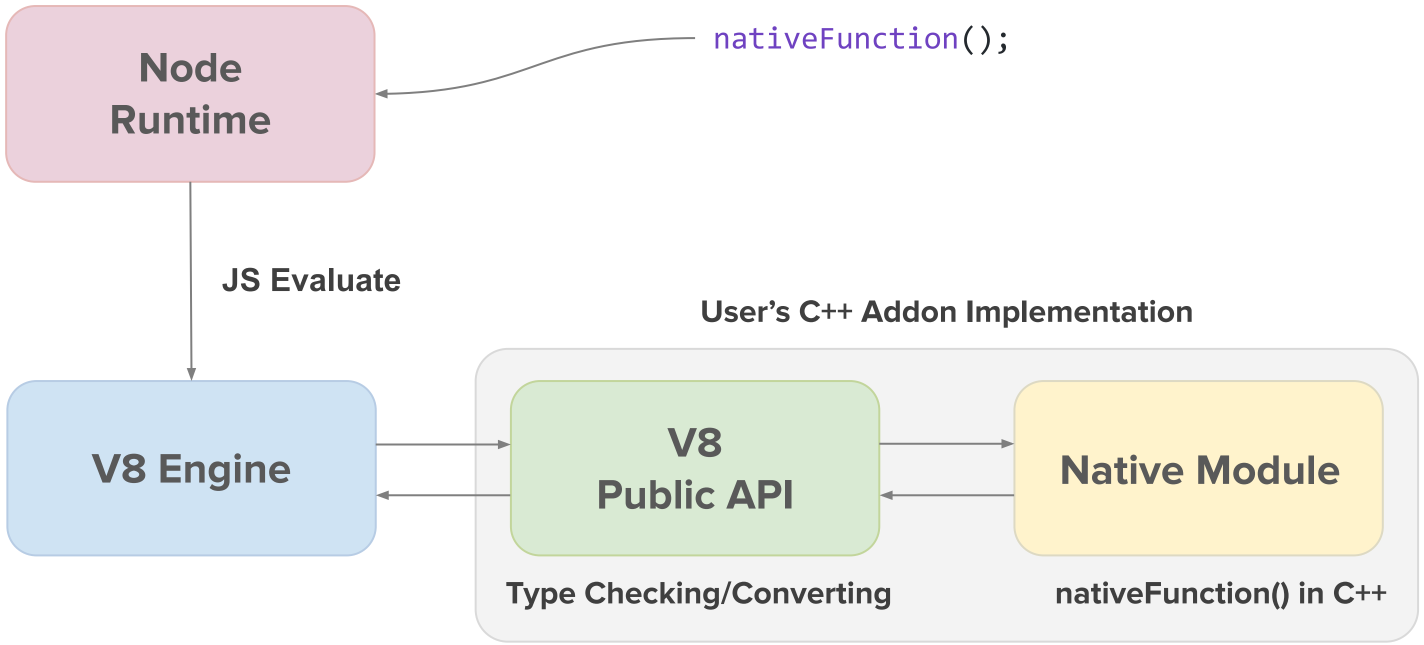
- Introduction to N-API and node-addon-api
- Online tutorials
- Let’s port some modules/individual projects
- Wrap-up and assessment

What is a native addon?

Node.js Addons are **dynamically-linked shared objects**, written in **C++**, that can be loaded into Node.js using the **require()** function, and used just as if they were an ordinary Node.js module.

They are used primarily to provide an **interface** between **JavaScript** running in Node.js and **C/C++** libraries.

What is a native addon?



Motivations for N-API

The API to implement native add-ons has been changed across different version of Node.js

Most of the changes were on **V8 API** and **ObjectWrap API** and other node internals

About **30%** of modules depend on a native add-on. A breakage on a native addon could become very important e.g. **node-sass**

Motivations for N-API

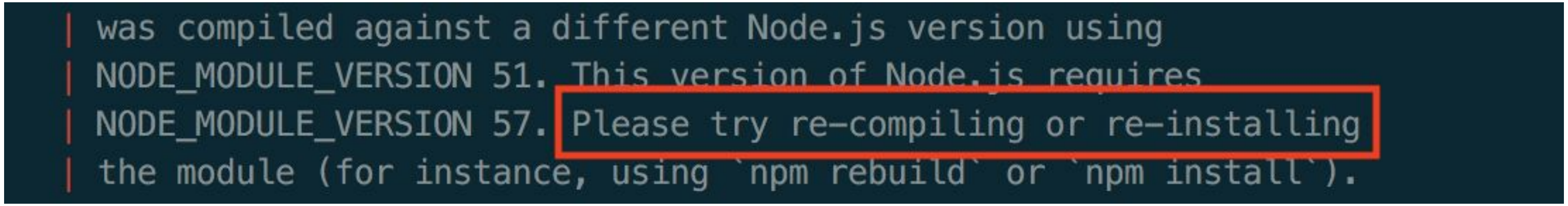
Need an adapter to stay compatible across different versions of Node.js

NAN - Native Abstraction for Node.js

- **API compatibility**
- Strongly bonded with **V8 API**
- You have to recompile your native add-ons when switching to a different version of Node.js

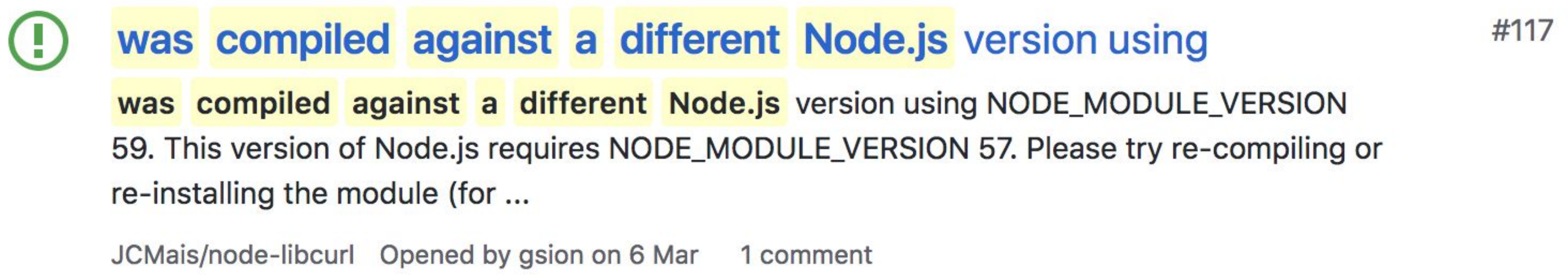
Motivations for N-API

End user



```
| was compiled against a different Node.js version using
| NODE_MODULE_VERSION 51. This version of Node.js requires
| NODE_MODULE_VERSION 57. Please try re-compiling or re-installing
| the module (for instance, using `npm rebuild` or `npm install`).
```

Maintainers



was compiled against a different Node.js version using

#117

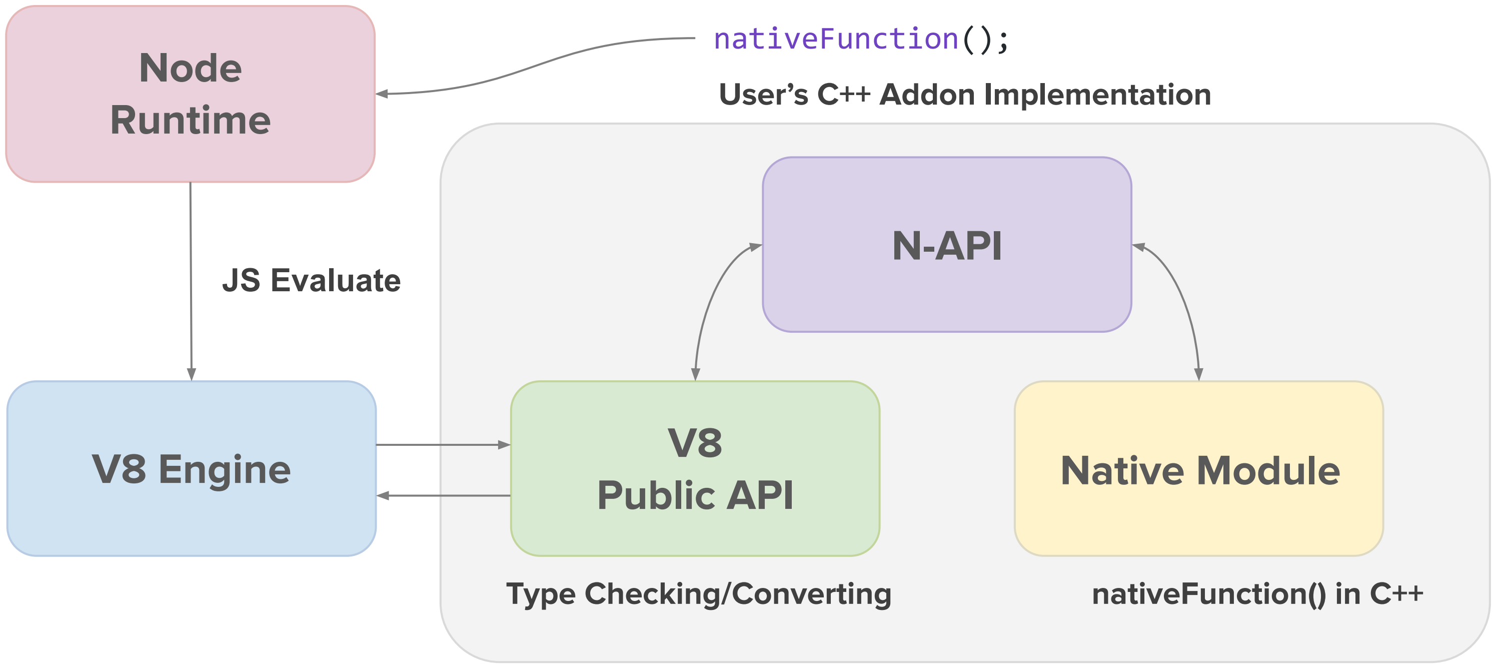
was compiled against a different Node.js version using NODE_MODULE_VERSION 59. This version of Node.js requires NODE_MODULE_VERSION 57. Please try re-compiling or re-installing the module (for ...

JCMais/node-libcurl Opened by gsion on 6 Mar 1 comment

What’s N-API

- Abstraction of the underlying JavaScript engine
- Defines and exports C types and functions that are independent from the JavaScript engine
- A binary-stable ABI

What's N-API


nativeFunction();
User's C++ Addon Implementation
Node Runtime
JS Evaluate
V8 Engine
N-API
V8 Public API
Native Module
Type Checking/Converting
nativeFunction() in C++

Without N-API

Addon

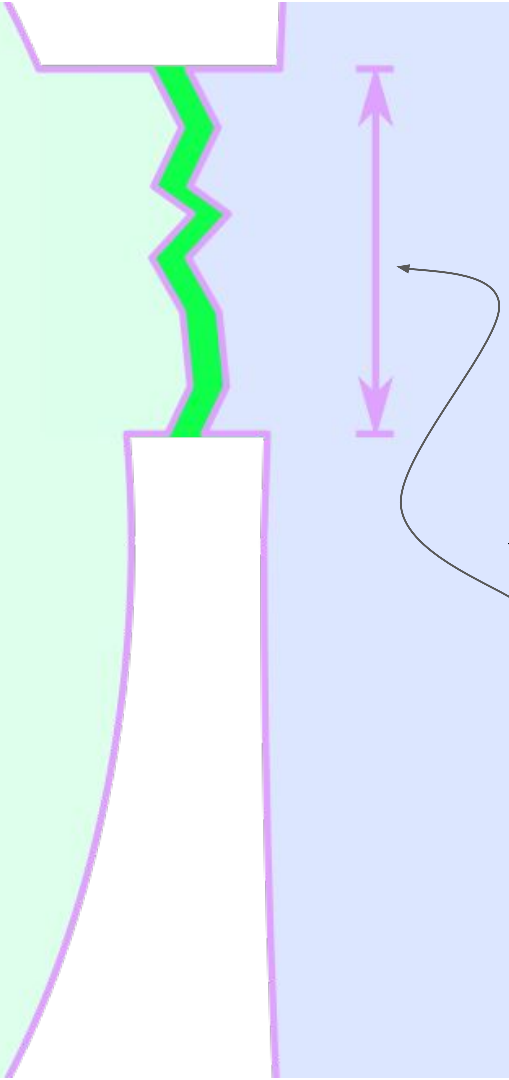
```
     Node.js: 6.12.2
          V8: 5.1.281
NODE_MODULE_VERSION: 48
```

Without N-API

- ABI break
  - At worst, mysterious segfaults
  - At best, addon fails to load
  - `NODE_MODULE_VERSION` mismatch

```
           Node.js: 8.9.3
                V8: 6.1.534
NODE_MODULE_VERSION: 57
```

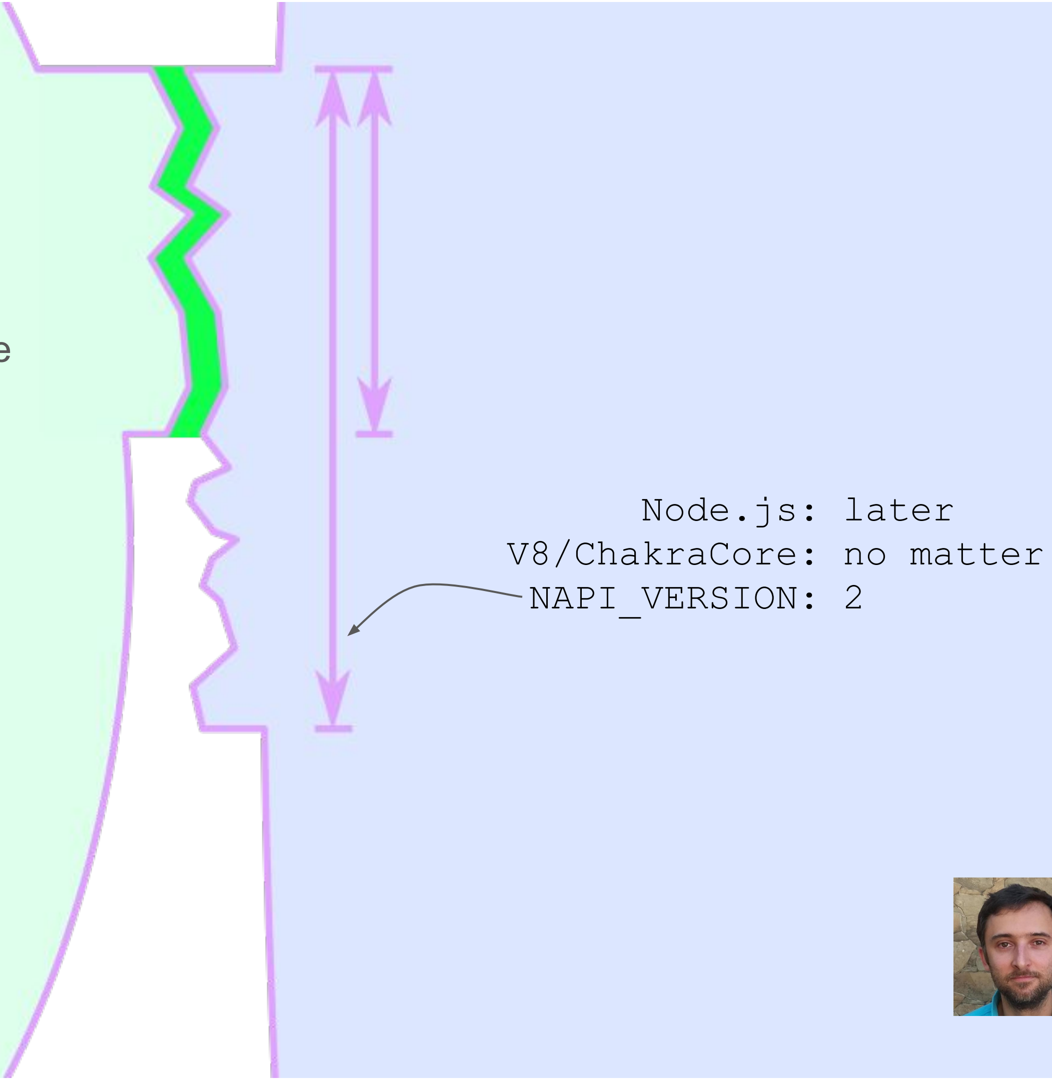
With N-API



```
      Node.js: no matter
V8/ChakraCore: no matter
 NAPI_VERSION: 1
```

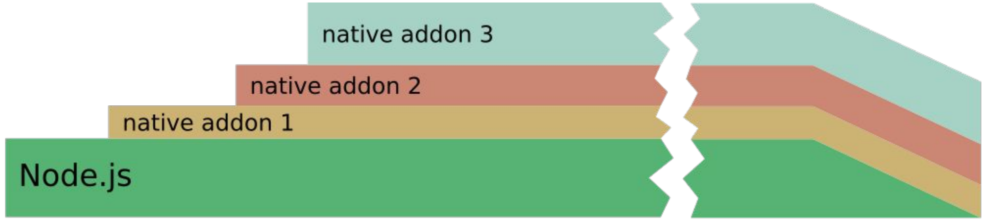
With N-API

- No ABI break
  - `NAPI_VERSION` is cumulative
  - Addon is forwards compatible

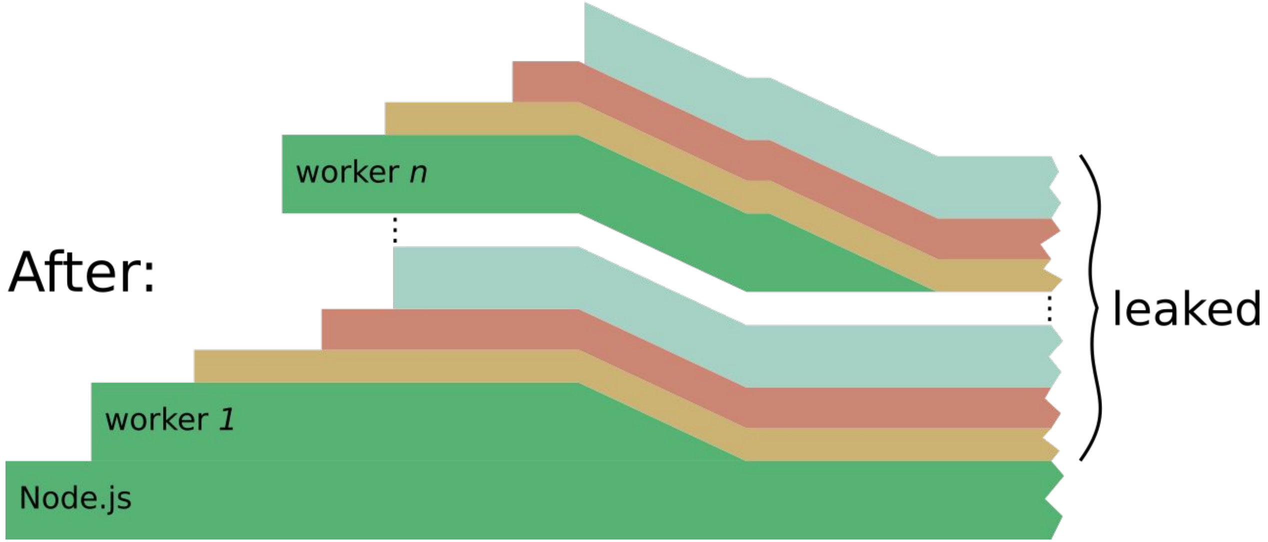


Context Awareness

Before:



After:



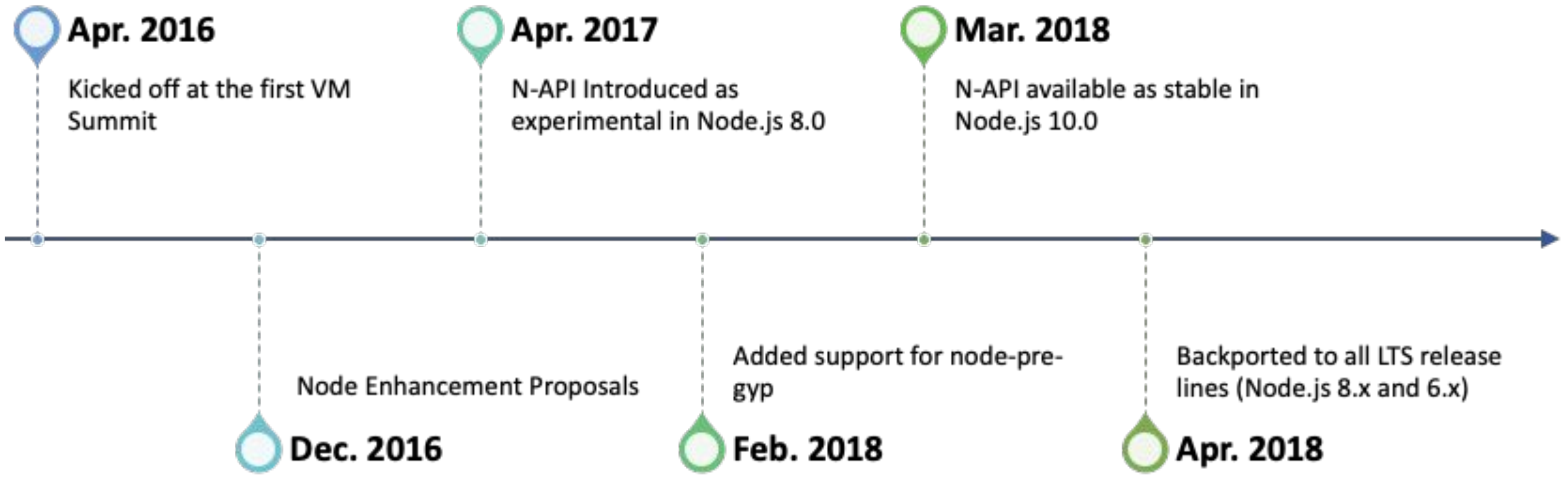
Context Awareness

Cleanup

- Unload DSO (one per process)
- Unload addon instance data
  - `napi_add_env_cleanup_hook() – N-API 5`
  - `napi_set_instance_data() – experimental`
- Clean up leftover references created with
  - `napi_wrap()`
  - `napi_add_finalizer() – N-API 5`
  - `napi_buffer_create_external()`
  - `et. al.`

GC will not free them! Need to track them and explicitly free them!

A brief history of N-API


Apr. 2016
Kicked off at the first VM Summit
Dec. 2016
Node Enhancement Proposals
Apr. 2017
N-API Introduced as experimental in Node.js 8.0
Feb. 2018
Added support for node-pre-gyp
Mar. 2018
N-API available as stable in Node.js 10.0
Apr. 2018
Backported to all LTS release lines (Node.js 8.x and 6.x)

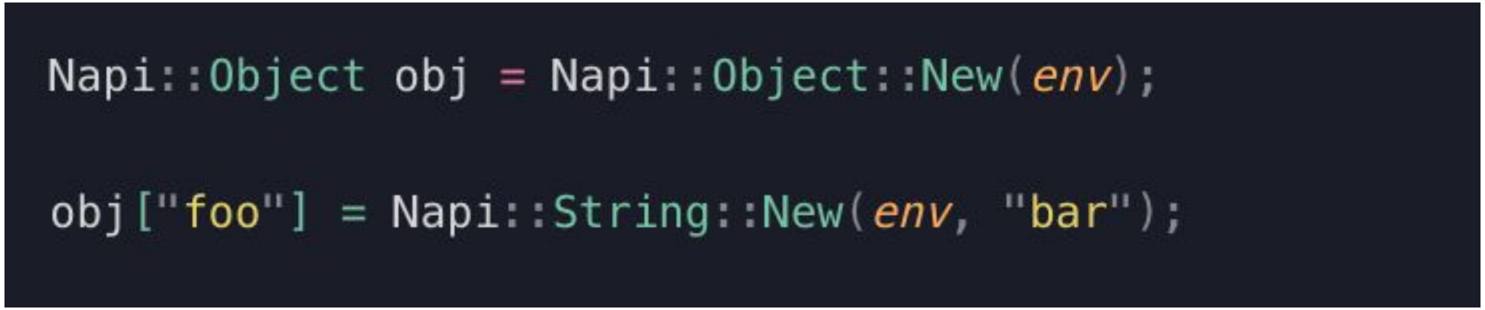
N-API and node-addon-api

- N-API is C whereas node-addon-api is C++
- node-addon-api not technically part of ABI stable API
    - Helper to simplify code using C++
    - All inline
    - Only depends on export N-API C functions

npm install **node-addon-api**

About **300k** downloads / week

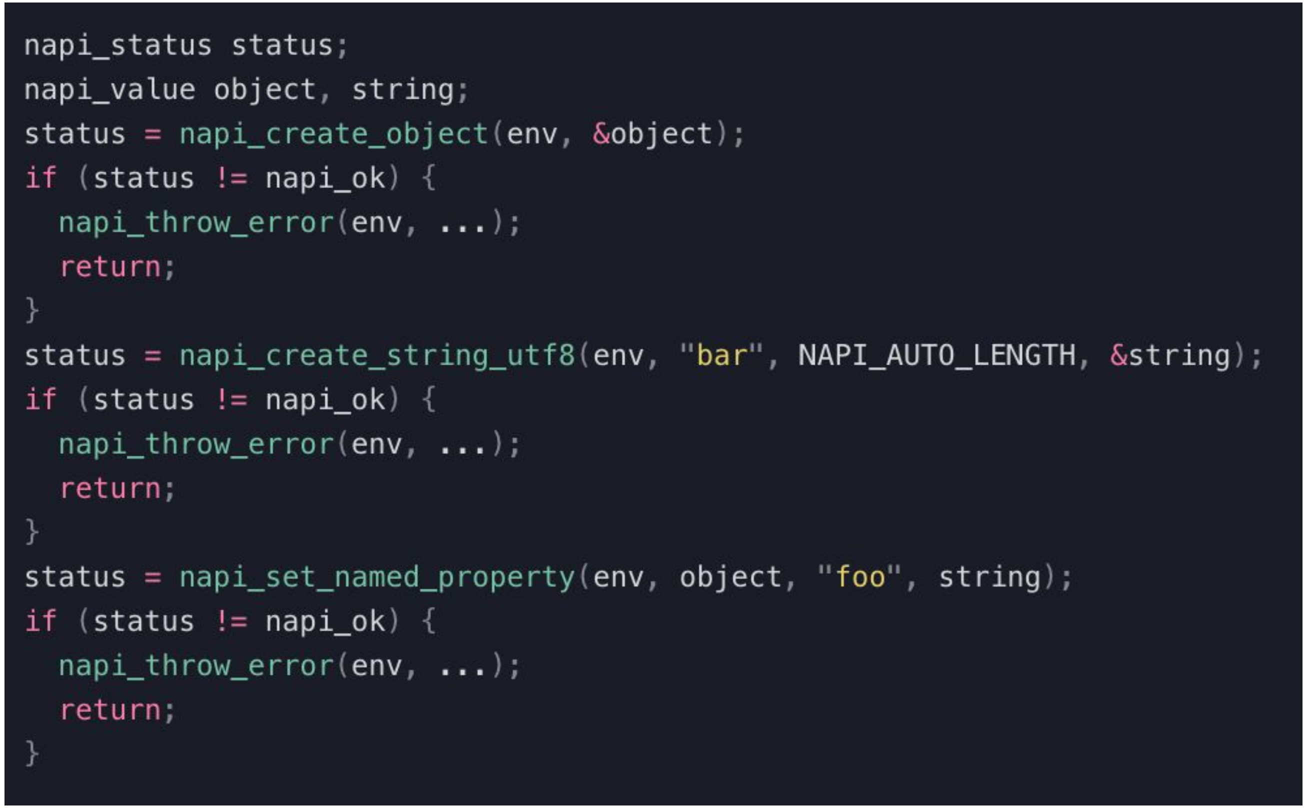
N-API and node-addon-api



```
Napi::Object obj = Napi::Object::New(env);

obj["foo"] = Napi::String::New(env, "bar");
```

N-API and node-addon-api



```
napi_status status;
napi_value object, string;
status = napi_create_object(env, &object);
if (status != napi_ok) {
  napi_throw_error(env, ...);
  return;
}
status = napi_create_string_utf8(env, "bar", NAPI_AUTO_LENGTH, &string);
if (status != napi_ok) {
  napi_throw_error(env, ...);
  return;
}
status = napi_set_named_property(env, object, "foo", string);
if (status != napi_ok) {
  napi_throw_error(env, ...);
  return;
}
```

Node-addon-api Versus NAN

- Same
  - Includes a C++ wrapper
  - Provides common helper functionality
  - Reduces likelihood of needing to change code for new Node.js versions
- Different
  - Does not use **V8 Types**
  - **Not tied** to a specific JavaScript engine
  - Preserves **compile once/run multiple versions** from N-API
  - Reduces even further likelihood of having to change code

What happened in the last year (**core**)

N-API 4

- Added **thread-safe** function

N-API 5

- Added API to manage **date** object
- Finalizer callback (marked as stable)
- Optional callback in **thread-safe** function

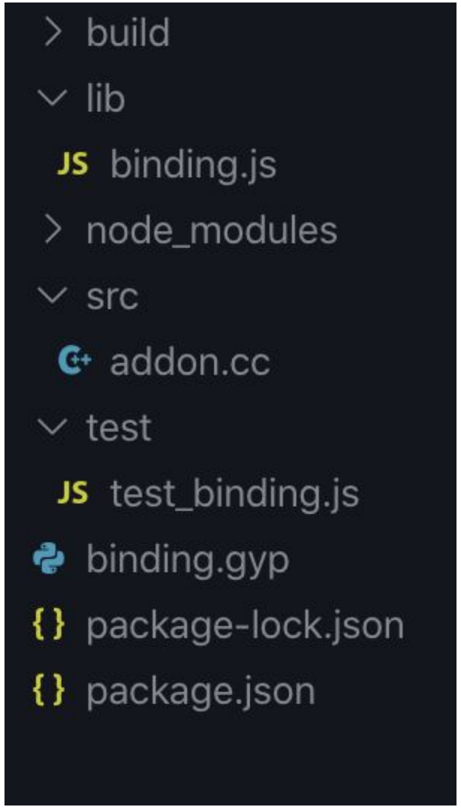
What happened in the last year (**node-addon-api**)

- Improved **documentation**
- Added **new asynchronous API**
  - Napi::AsyncContext
- Added **thread-safe function**
  - Napi::ThreadsafeFunction
- Improved **AsyncWorker** API
  - See: https://github.com/nodejs/node-addon-api/issues/231
- Added **AsyncProgressWorker** API
  - Napi::AsyncProgressWorker
- Added API to manage **date** object
  - Napi::Date

What happened in the last year (**tools**)

- Added support for **prebuild**
  - A command line tool for easily making prebuilt binaries
  - https://www.npmjs.com/package/prebuild#n-api-considerations
- Added support for **cmake-js**
  - Build tool that uses **CMake** to build the native add-on instead of **GYP**
  - https://www.npmjs.com/package/cmake-js#n-api-and-node-addon-api

How to organise your project



```
> build
v lib
  binding.js
> node_modules
v src
  addon.cc
v test
  test_binding.js
binding.gyp
package-lock.json
package.json
```

build folder contains the intermediary and final build products.

lib folder contains the the JavaScript code that will use the native code to export some features

src folder contains the native **C / C++** code

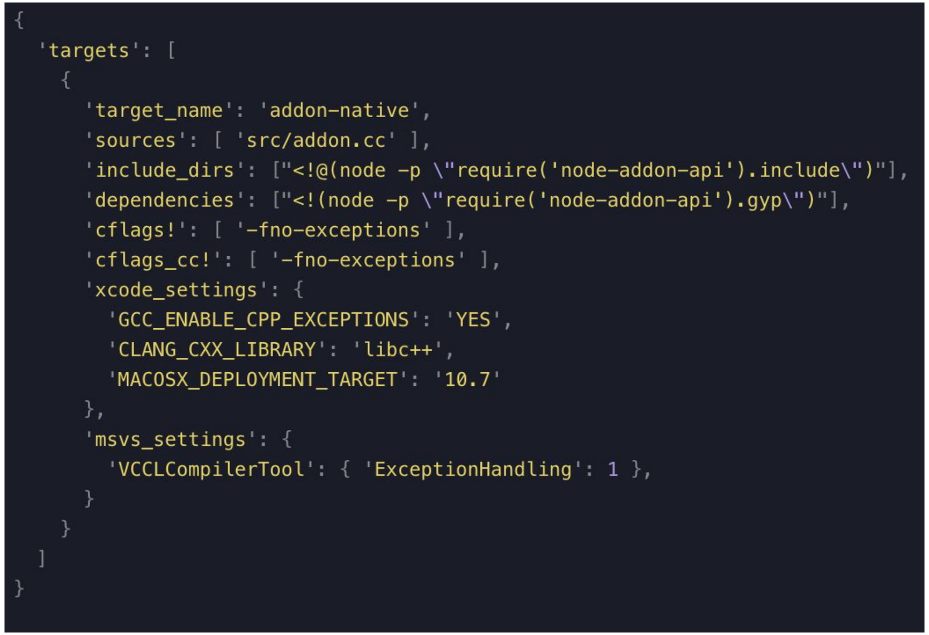
test folder contains the testing code

binding.gyp file that contains all settings to build the nativa add-on

package.json npm description of your module

package-lock.json used by npm to ensure deployment consistency

What’s binding.gyp



```
{
  'targets': [
    {
      'target_name': 'addon-native',
      'sources': [ 'src/addon.cc' ],
      'include_dirs': ["<!@(node -p \"require('node-addon-api').include\")"],
      'dependencies': ["<!(node -p \"require('node-addon-api').gyp\")"],
      'cflags!': [ '-fno-exceptions' ],
      'cflags_cc!': [ '-fno-exceptions' ],
      'xcode_settings': {
        'GCC_ENABLE_CPP_EXCEPTIONS': 'YES',
        'CLANG_CXX_LIBRARY': 'libc++',
        'MACOSX_DEPLOYMENT_TARGET': '10.7'
      },
      'msvs_settings': {
        'VCCLCompilerTool': { 'ExceptionHandling': 1 },
      }
    }
  ]
}
```

By default Node.js use **GYP** to build native add-on.

Native add-ons are builded using node-gyp a cross platform CLI tool written in Node.js that contains a fork of GYP.

GYP https://gyp.gsrc.io

node-gyp https://github.com/nodejs/node-gyp

There are other build tools like **cmake-js**

Add a badge to your project

- Simply add a URL to the top of your README file
- Indicates the minimum N-API version your addon supports
- Benefits
  - Identifies your addon as supporting N-API
  - Makes your addon easier to find
- N-API v3 is a good starting point because this is the first version considered as stable API

N-API v1 N-API v2 N-API v3 N-API v4 N-API v5 N-API experimental

Add a badge to your project

N-API v3

https://img.shields.io/badge/N--API-v3-green.svg

https://img.shields.io/badge/N--API-v4-green.svg

https://img.shields.io/badge/N--API-v5-green.svg

https://img.shields.io/badge/N--API-experimental-orange.svg

Online Tutorials

- For starting a new add-on module from scratch
  - napi.inspiredware.com/getting-started/first.html
- For starting a new add-on module from scratch with ObjectWrap
  - napi.inspiredware.com/getting-started/objectwrap.html
- For migrating an existing NAN module
  - napi.inspiredware.com/getting-started/migration.html
- Async Worker
  - napi.inspiredware.com/special-topics/asyncworker.html
- Node pre-gyp
  - napi.inspiredware.com/special-topics/node-pre-gyp.html
- List of candidate modules to port available at **goo.gl/9bksyw**

Online Tutorials

- //TODO insert the link for new tutorials
- List of candidate modules to port available at **goo.gl/9bksyw**

Opportunity to work on individual projects

- Workshop presenters available for help and questions
- List of candidate modules to port available at **tiny.cc/n-api** or **goo.gl/9bksyw**

Supporting resources

- N-API Documentation
  - https://nodejs.org/api/n-api.html — The C API
  - https://github.com/nodejs/node-addon-api — The C++ wrapper
- N-API migration assistant
  - https://github.com/nodejs/node-addon-api/blob/master/doc/conversion-tool.md
- Generator
  - https://www.npmjs.com/package/generator-napi-module
- Examples
  - https://github.com/nodejs/node-addon-examples/
- Node-pre-gyp Documentation
  - https://github.com/mapbox/node-pre-gyp

Supporting resources

- Prebuild
  - https://www.npmjs.com/package/prebuild
- Prebuildify
  - https://www.npmjs.com/package/prebuildify
- CMake.js
  - https://www.npmjs.com/package/cmake-js
- Other N-API bindings:
  - **Neon** https://github.com/neon-bindings/neon (Rust)
- Genepi (Automatic generation of N-API wrapper from a C++ library)
  - https://github.com/Geode-solutions/genepi

Wrap-up and assessment

What did you like/not like

- Workshop
- N-API

Get Involved:

- https://github.com/nodejs/abi-stable-node
- https://github.com/nodejs/node
- https://github.com/nodejs/node-addon-api
- Weekly N-API meeting - Monday 10:00 EST

NODE+JS INTERACTIVE

December 11–12, 2019
Montreal, Canada