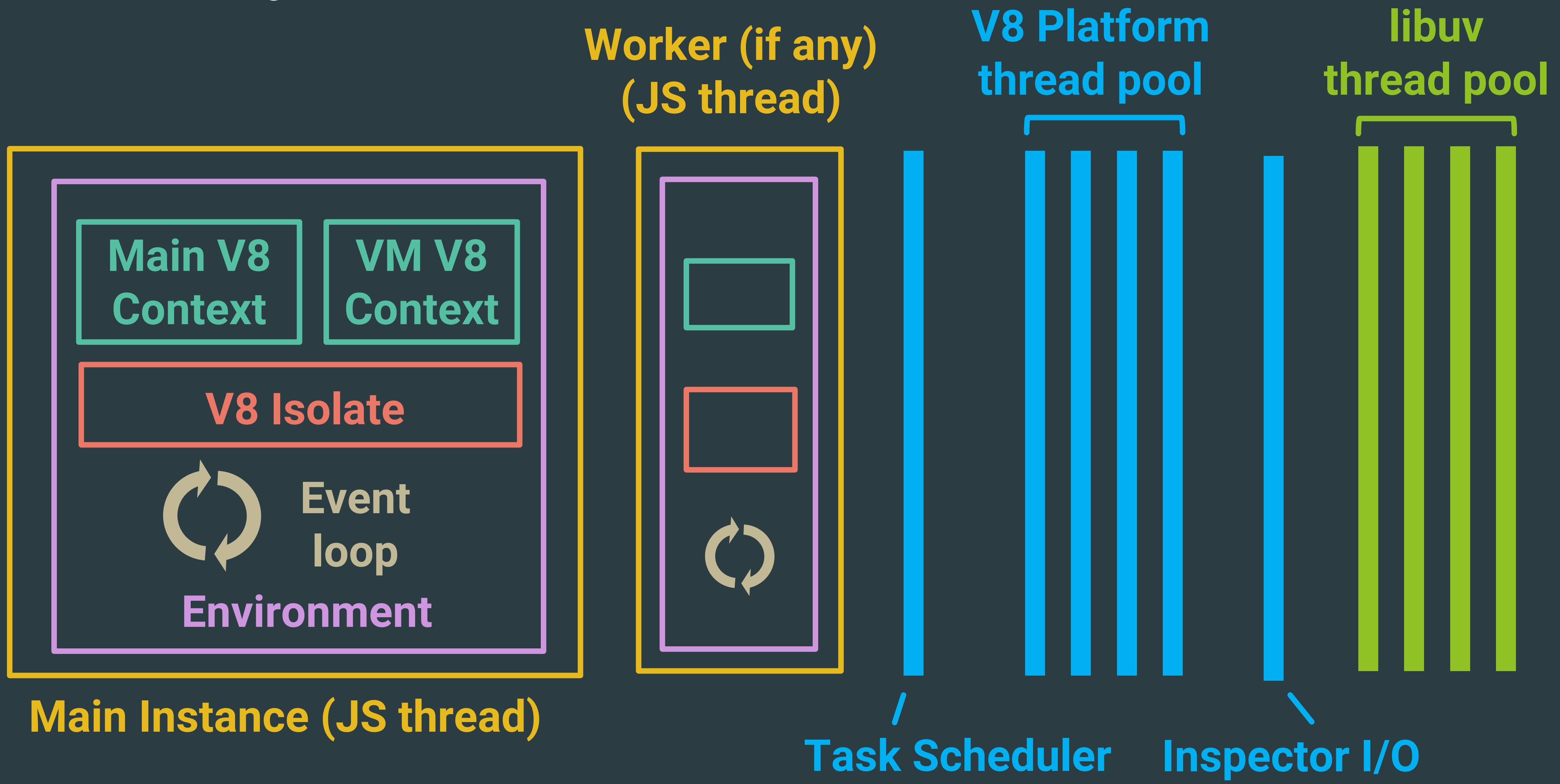
Bootstrap of Node.js Core

Joyee Cheung
OpenJS collaboration summit, May 2019

A Node.js Process



Bootstrap (2019.05)

```
node::Start()
```

Bootstrap (2019.05)

```
     node::Start()
           ↓
InitializeOncePerProcess()
```

Parse the CLI arguments, Initialize the V8 Platform， OpenSSL, ICU, signal handler…

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， OpenSSL, ICU, signal handler...

↓

`NodeMainInstance() / Worker()`

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， OpenSSL, ICU, signal handler...

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate`

JS heap, JS exceptions, Microtask queue...

Bootstrap (2019.05)

$$\begin{tikzcd}
\texttt{node::Start()} \arrow[d] & \\
\texttt{InitializeOncePerProcess()} \arrow[d] & \\
\texttt{NodeMainInstance() / Worker()} \arrow[r] & \boxed{\begin{array}{c}\texttt{v8::Isolate} \\ \downarrow \\ \texttt{v8::Context}\end{array}}
\end{tikzcd}$$

Parse the CLI arguments, Initialize the V8 Platform， OpenSSL, ICU, signal handler…

JS heap, JS exceptions, Microtask queue…

global proxy, JS builtins

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， OpenSSL, ICU, signal handler...

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate` — **JS heap, JS exceptions, Microtask queue...**

↓

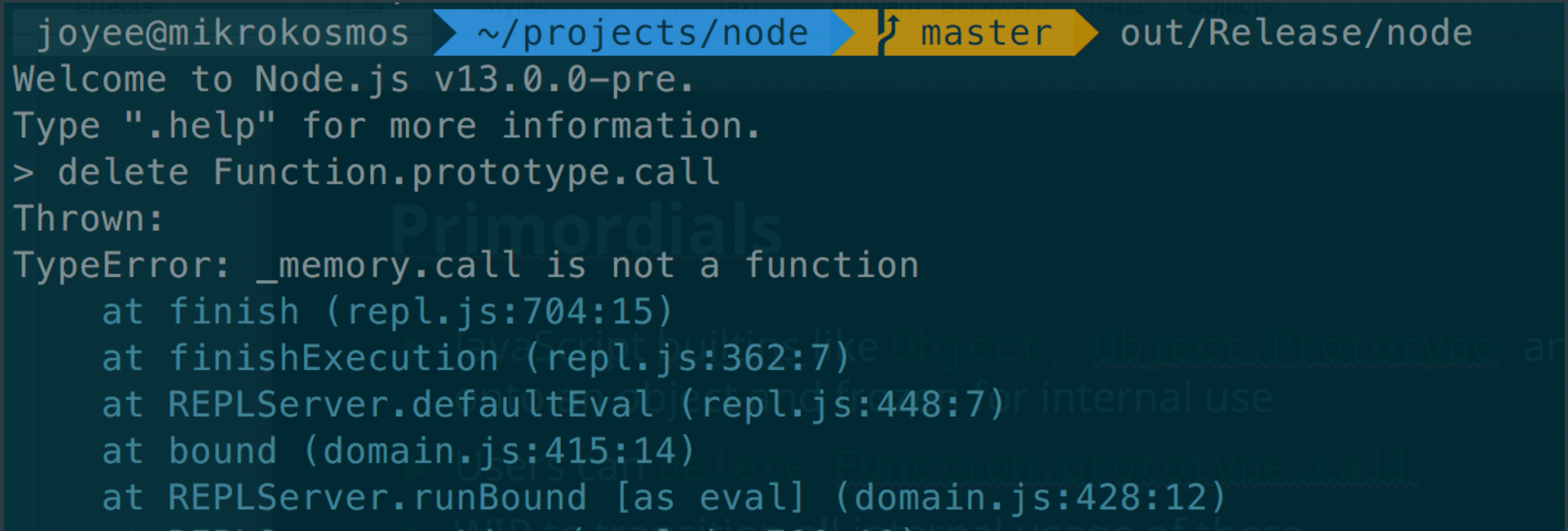
`v8::Context` — **global proxy, JS builtins**

↓

`per_context/*.js` — **Node.js primordials**

Primordials

- JavaScript builtins like `Object, Object.prototype` are cloned onto an object and frozen for internal use
- Users can `delete Function.prototype.call`
- WIP to transition all internal usage of these



```
 joyee@mikrokosmos  ~/projects/node   master  out/Release/node
Welcome to Node.js v13.0.0-pre.
Type ".help" for more information.
> delete Function.prototype.call
Thrown:
TypeError: _memory.call is not a function
    at finish (repl.js:704:15)
    at finishExecution (repl.js:362:7)
    at REPLServer.defaultEval (repl.js:448:7)
    at bound (domain.js:415:14)
    at REPLServer.runBound [as eval] (domain.js:428:12)
```

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， signal handler...

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate` — **JS heap, JS exceptions, Microtask queue...**

↓

`v8::Context` — **global proxy, JS builtins**

↓

`per_context/*.js` — **Node.js primordials**

↓

`node::Environment` — **Stuff that does not have a better place to go**

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform, signal handler...

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate` — **JS heap, JS exceptions, Microtask queue...**

↓

`v8::Context` — **global proxy, JS builtins**

↓

`per_context/*.js` — **Node.js primordials**

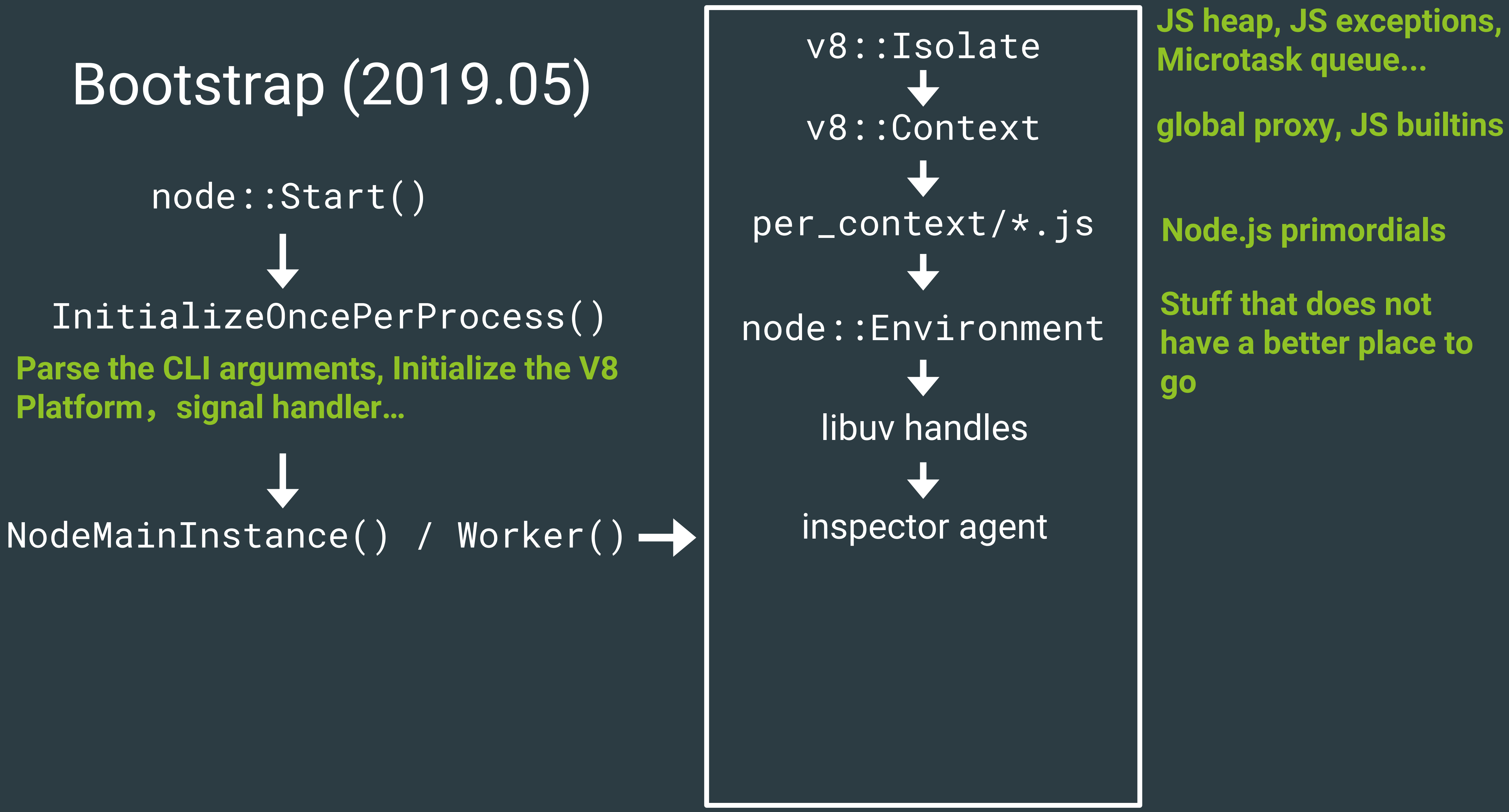
↓

`node::Environment` — **Stuff that does not have a better place to go**

↓

libuv handles

Bootstrap (2019.05)



Parse the CLI arguments, Initialize the V8 Platform， signal handler...

JS heap, JS exceptions, Microtask queue...

global proxy, JS builtins

Node.js primordials

Stuff that does not have a better place to go

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， signal handler...

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate` — JS heap, JS exceptions, Microtask queue...

↓

`v8::Context` — global proxy, JS builtins

↓

`per_context/*.js` — Node.js primordials

↓

`node::Environment` — Stuff that does not have a better place to go

↓

libuv handles

↓

inspector agent

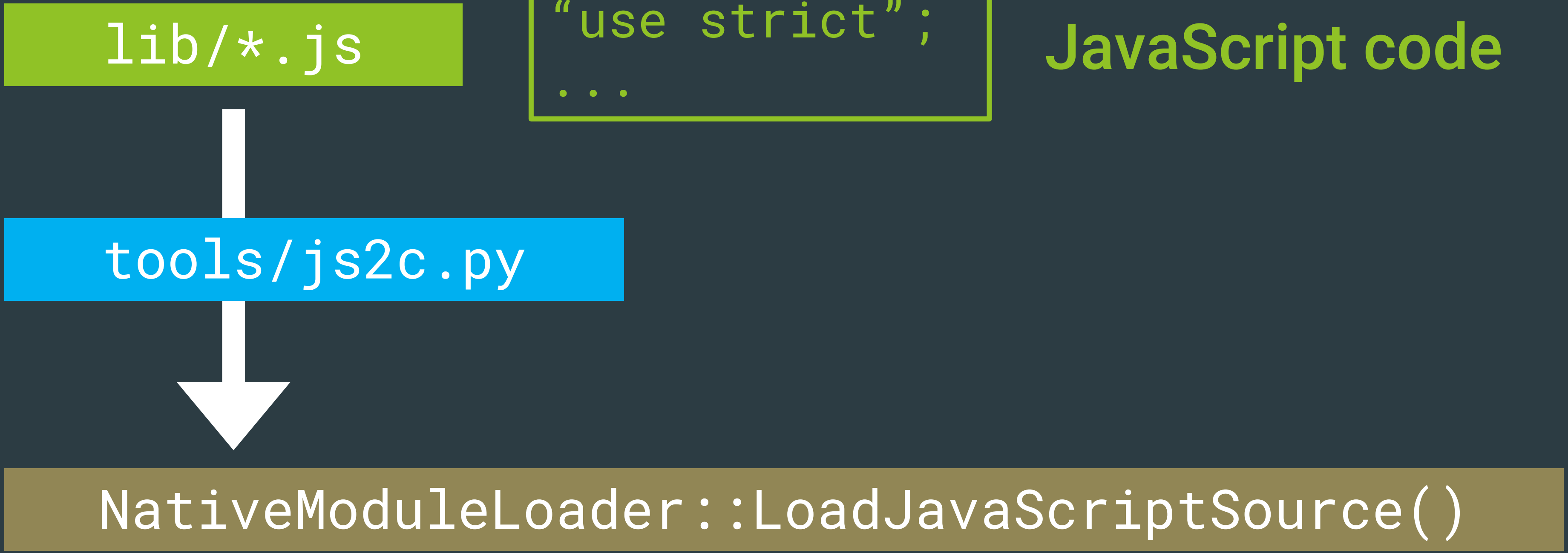
↓

`bootstrap/*.js` — global, process, task queues, ESM/CJS loaders ...

`lib/internal/bootstrap/loaders.js`

- Internal module loaders
- **C++** binding loaders
  - `process.binding()`
  - `process._linkedBinding()`
  - `internalBinding()`
- `require()` for loading other internal **JavaScript** modules

Built-in Modules (Native Modules)



```
"use strict";
...
```

JavaScript code

```
static const uint16_t assert_raw[] = {
    ...
};
```

static data array containing the source

Built-in Modules (Native Modules)

```
function (exports, require, module, process,
          internalBinding, primordials) {

  require(‘internal/fs/utils’);

  module.exports = {…};

}
```

Compiled with a special wrapper
that include access to more internals

Built-in Modules (Native Modules)

- V8 code cache shared among main thread and worker threads
  - ~60% startup time improvement for workers
- Pre-built code cache embedded in the binary
  - ~30% startup time improvement for the main thread

`lib/internal/bootstrap/node.js`

- Set up most stuff on `process` and `global`
- C++ passes `isMainThread`, `ownsProcessState` into the script
  - `false` for workers, `true` for the main thread

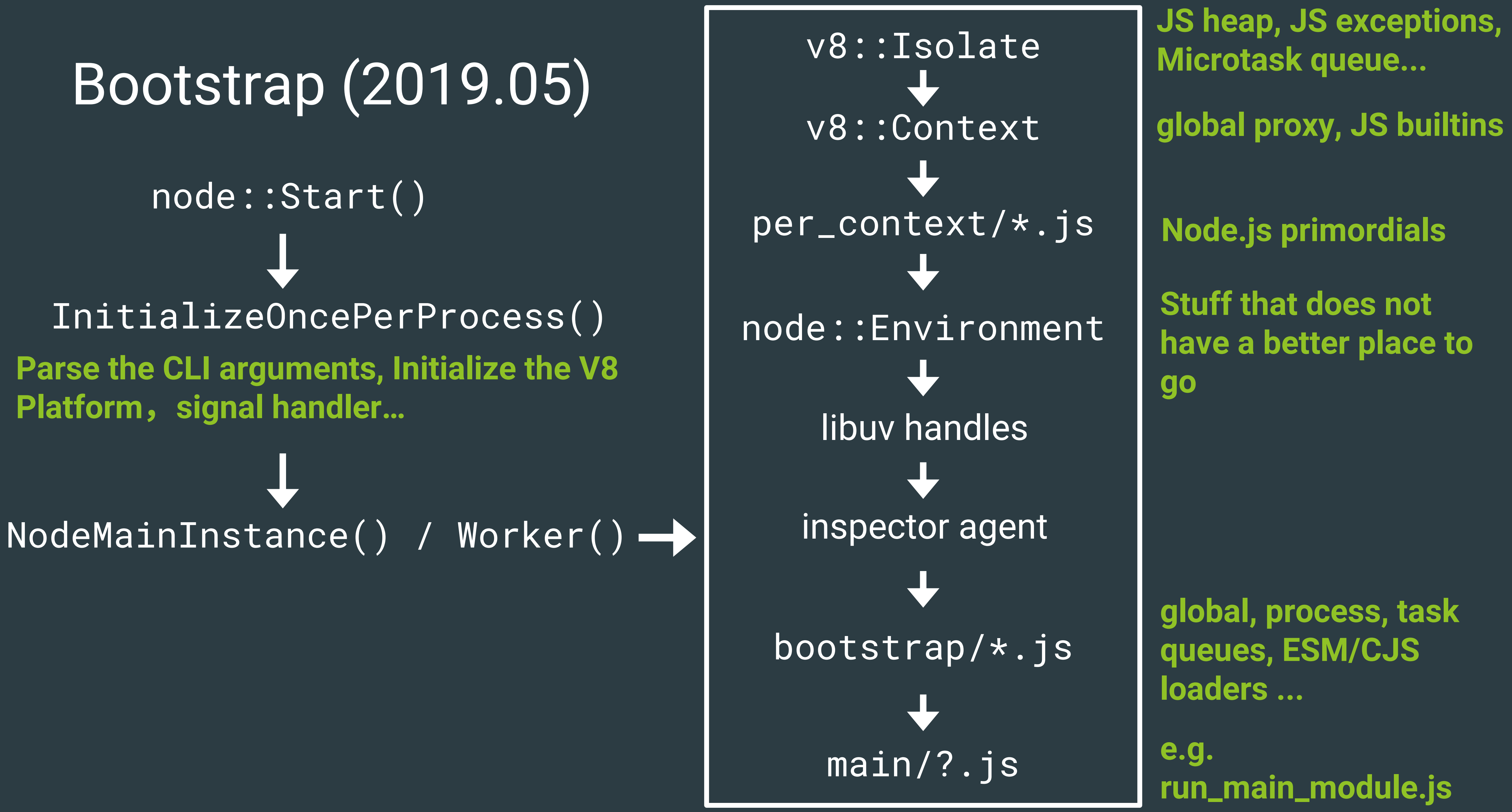
`lib/internal/bootstrap/node.js`

- Set up most stuff on `process` and `global`
- C++ passes `isMainThread`, `ownsProcessState` into the script
  - `false` for workers, `true` for the main thread
- Set up JavaScript callbacks that will be added as `v8::Persistent` to the `Environment`
  - Async hook callbacks
  - Timers & `process.nextTick()` schedulers
- Must not run async operations (not snapshottable)

`lib/internal/bootstrap/pre_execution.js`

- Not actively run. Required by main scripts (explained later)
- Not included in the snapshot

Bootstrap (2019.05)


node::Start()
InitializeOncePerProcess()
Parse the CLI arguments, Initialize the V8 Platform， signal handler...
NodeMainInstance() / Worker()
v8::Isolate
JS heap, JS exceptions, Microtask queue...
v8::Context
global proxy, JS builtins
per_context/*.js
Node.js primordials
node::Environment
Stuff that does not have a better place to go
libuv handles
inspector agent
bootstrap/*.js
global, process, task queues, ESM/CJS loaders ...
main/?.js
e.g. run_main_module.js

Main scripts

- `lib/internal/main/*.js`
- Main thread
  - `StartMainThreadExecution()`
  - Select a script based on CLI arguments, etc.
- Worker threads
  - `worker_thread.js`
- Runs `lib/internal/bootstrap/pre_execution.js` first to bootstrap the parts that depend on run time states

Main scripts

- `check_syntax.js: node –c test.js`
- `eval_stdin.js: cat test.js | node –e`
- `eval_string.js: node –e ‘1’`
- `inspect.js: node inspect ...`
- `print_bash_completion.js: node --completion-bash`
- `print_help.js: node --help`
- `prof_process.js: node --prof-process v8.log`
- `run_third_party_main.js:` for embedders

Main scripts

- `repl.js`: `node`
- `worker_thread.js`: for workers
- `run_main_module.js`
  - `node index.js`
  - `node --experimental-modules index.mjs`

Main scripts

- `repl.js`: `node`
- `run_main_module.js`
  - `node index.js`
  - `node --experimental-modules index.mjs`
- `worker_thread.js`: for workers

lib/internal/bootstrap/pre_execution.js

- Bootstrap that depend on run time states
  - e.g. CLI arguments, environment variables
  - Including CJS & ESM loader initilization

```
if (!getOptionValue('--no-warnings') &&
    process.env.NODE_NO_WARNINGS !== '1') {
  process.on('warning', onWarning);
}
```

User land CommonJS Modules

- Loader implemented in `lib/internal/modules/cjs/`

```
function (exports, require, module, __filename, __dirname) {
  require('fs');
}
```

Wrap user code with objects initialized by Node.js

User land ECMAScript Modules

- Loader implementation in `lib/internal/modules/esm/`
- Does not mess with the context except things added to the global proxy
  - `Buffer, process`, etc.

User land ECMAScript Modules

- An internal `WeakMap` holding `ModuleWrap -> Options`
  - `Options` includes dynamic `import()` callback and `import.meta` data
  - Per-isolate
    - `HostImportModuleDynamicallyCallback`
    - `HostInitializeImportMetaObjectCallback`

Bootstrap (2019.05)

`node::Start()`

↓

`InitializeOncePerProcess()`

Parse the CLI arguments, Initialize the V8 Platform， signal handler...

↓

`NodeMainInstance() / Worker()` →

↓

```
do {
  uv_run(...)
} while (...)
```

Event Loop

`v8::Isolate` — **JS heap, JS exceptions, Microtask queue...**

↓

`v8::Context` — **global proxy, JS builtins**

↓

`per_context/*.js` — **Node.js primordials**

↓

`node::Environment` — **Stuff that does not have a better place to go**

↓

libuv handles

↓

inspector agent

↓

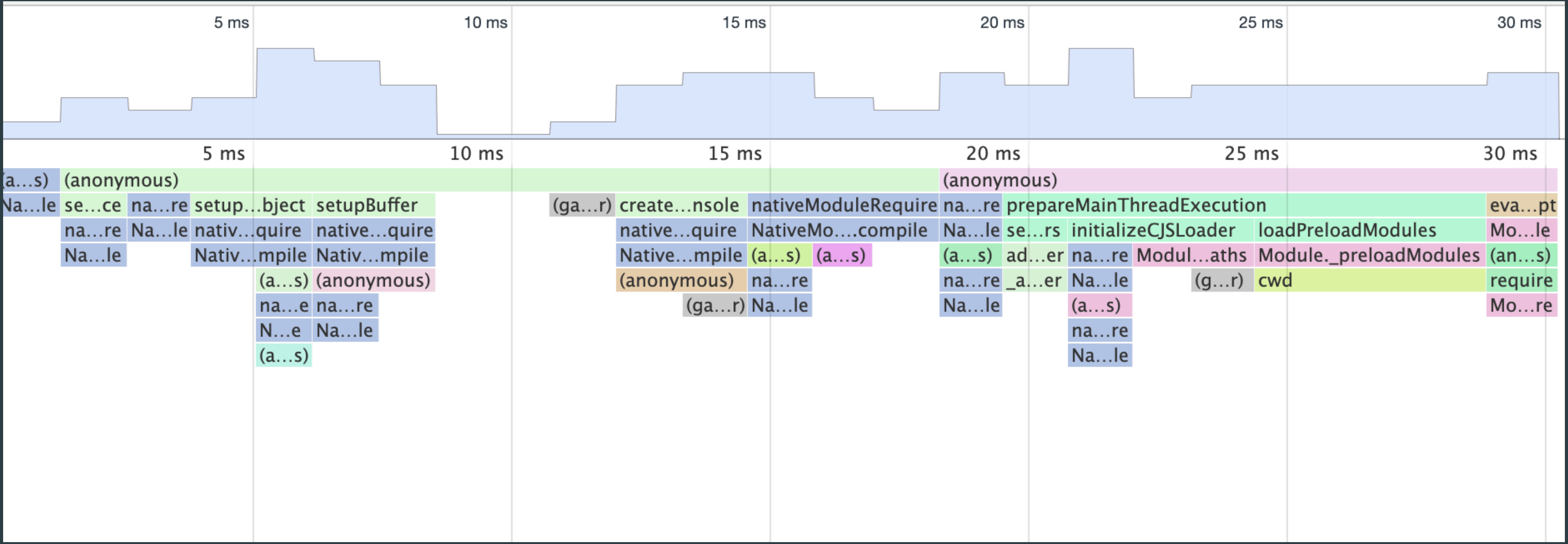
`bootstrap/*.js` — **global, process, task queues, ESM/CJS loaders ...**

↓

`main/?.js` — **e.g. run_main_module.js**

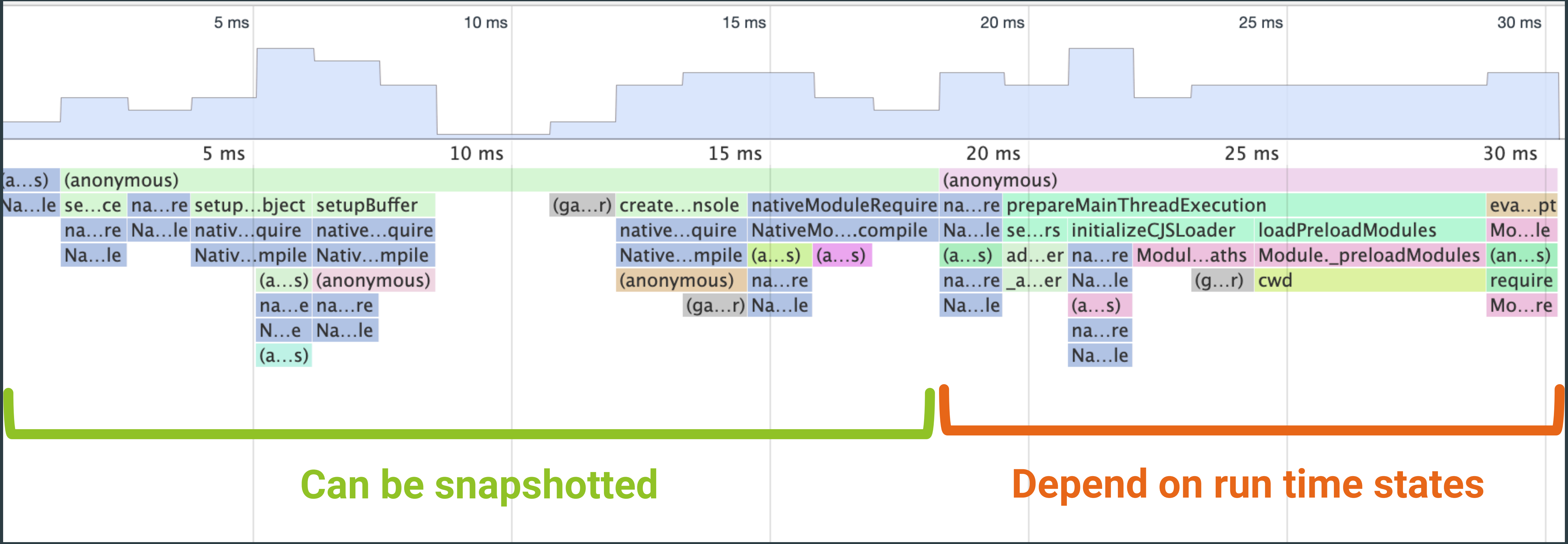
Snapshot Integration

```
out/Release/node --cpu-prof-interval=100 --cpu-prof -e "{}"
```



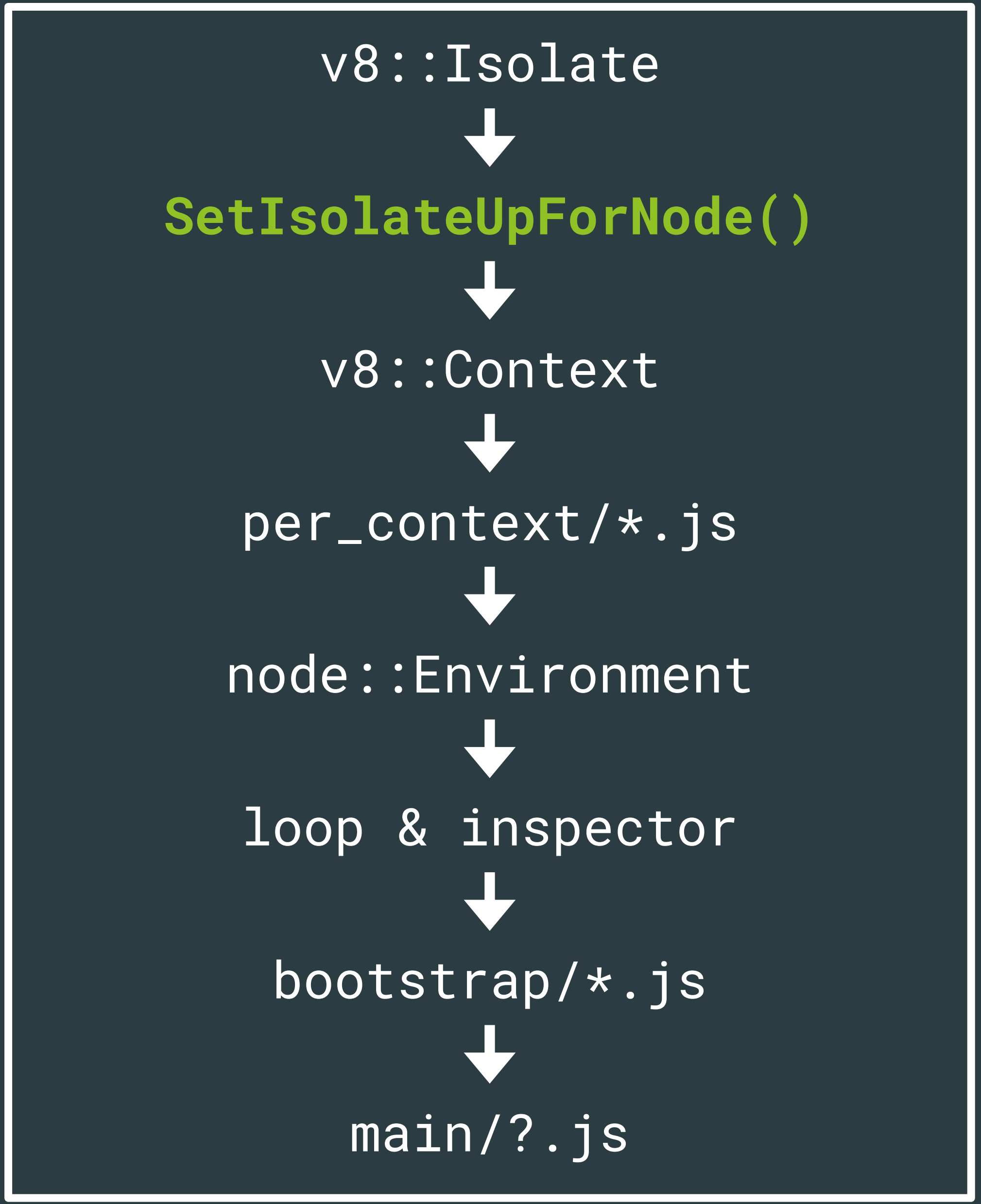
Snapshot Integration

```
out/Release/node --cpu-prof-interval=100 --cpu-prof -e "{}"
```



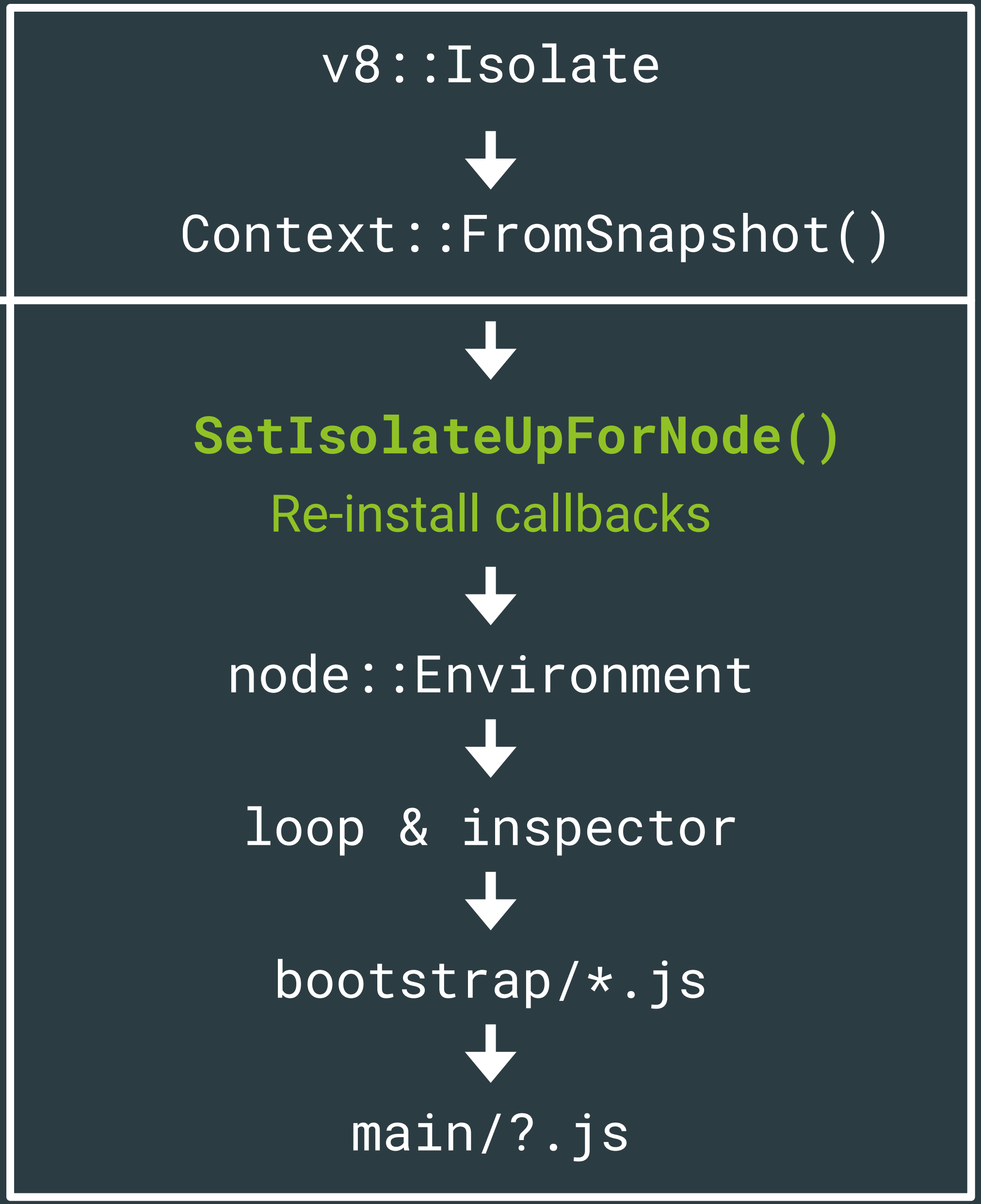
Snapshot Integration

Original



Snapshot Integration

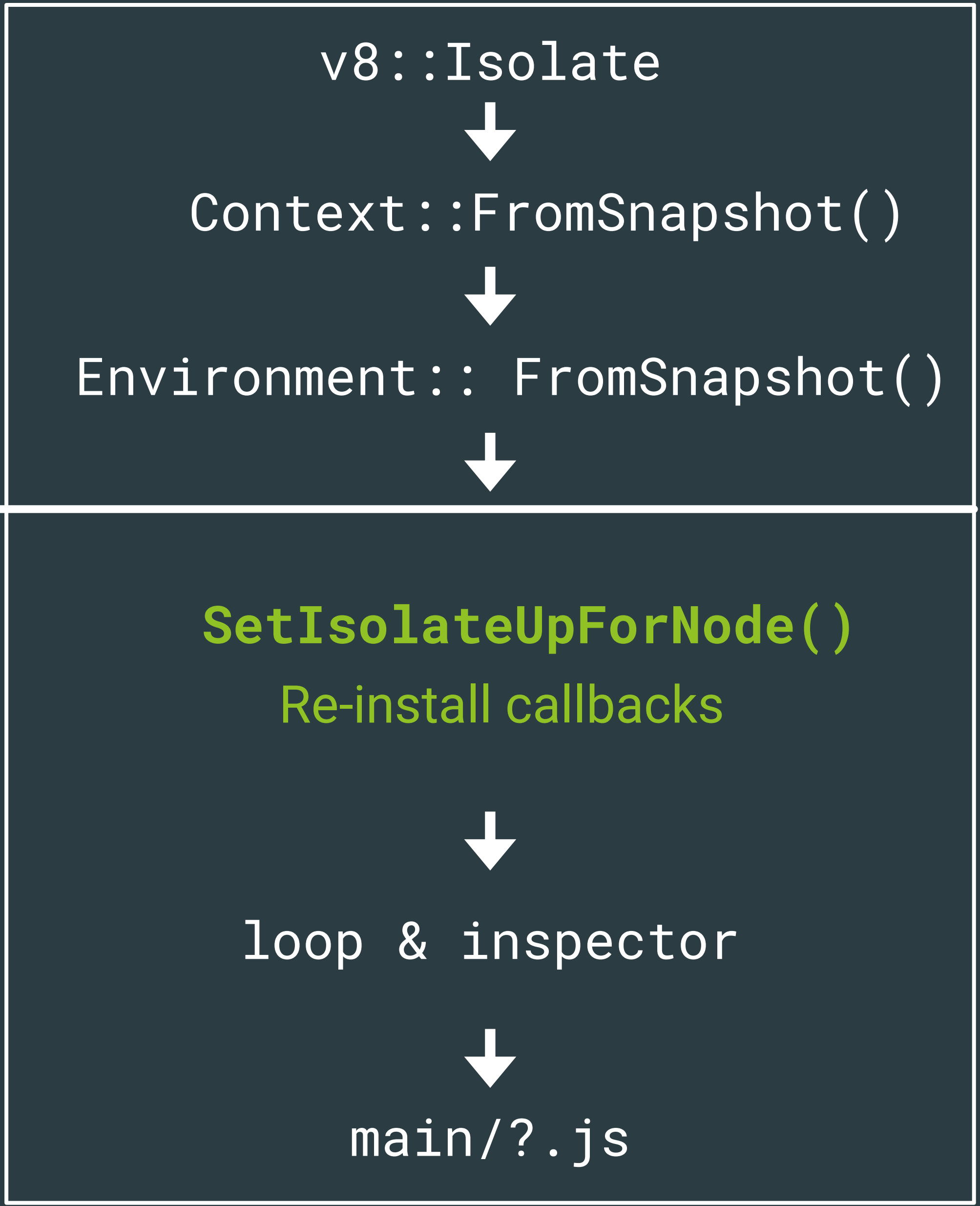
Snapshotted (2019.05)



Snapshot Integration

Goal

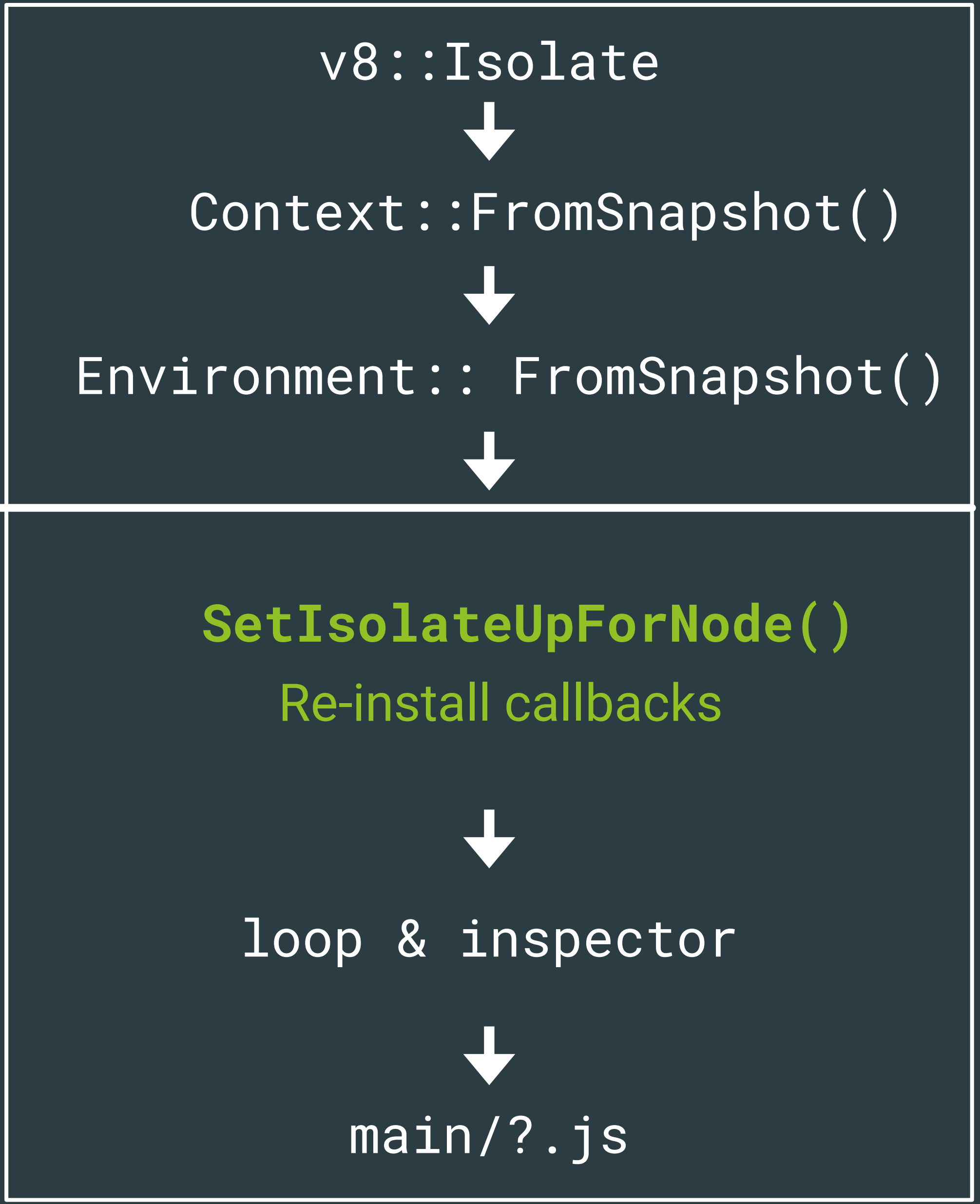
Deserialize from snapshot instead of executing `per_context/*.js` & `bootstrap/*.js`



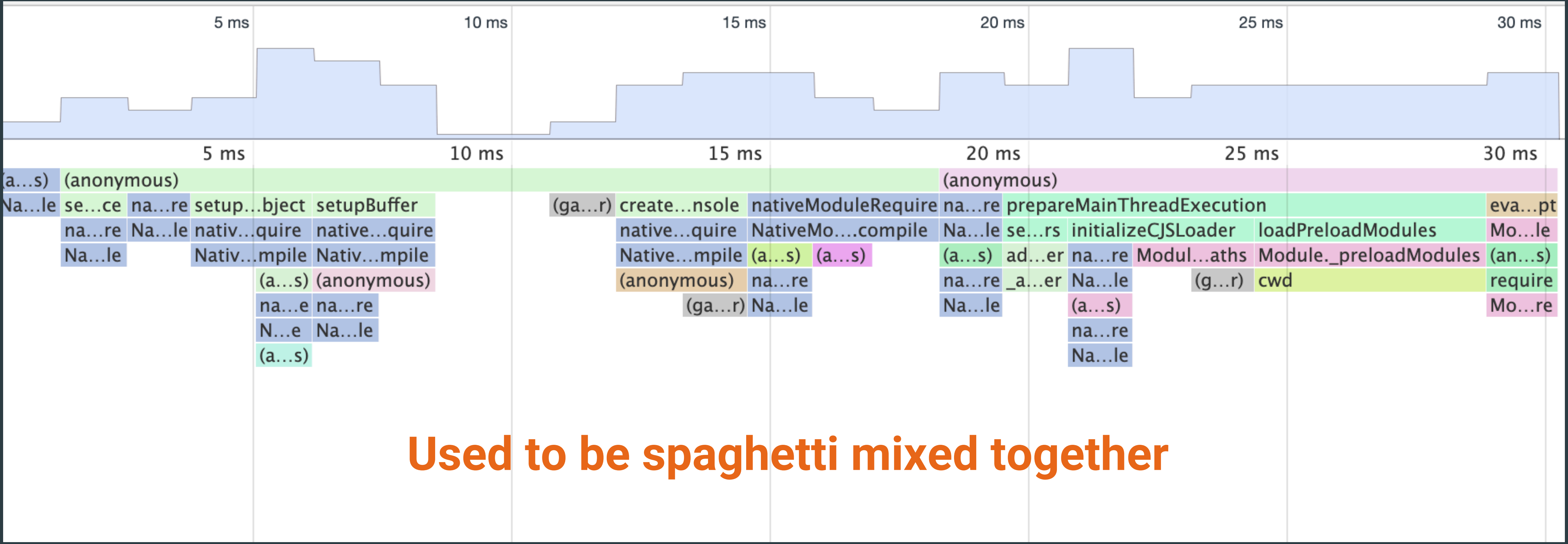
Snapshot Integration

Refactoring

The bootstrap process must be independent of run time states before the snapshot is captured.



Snapshot Integration



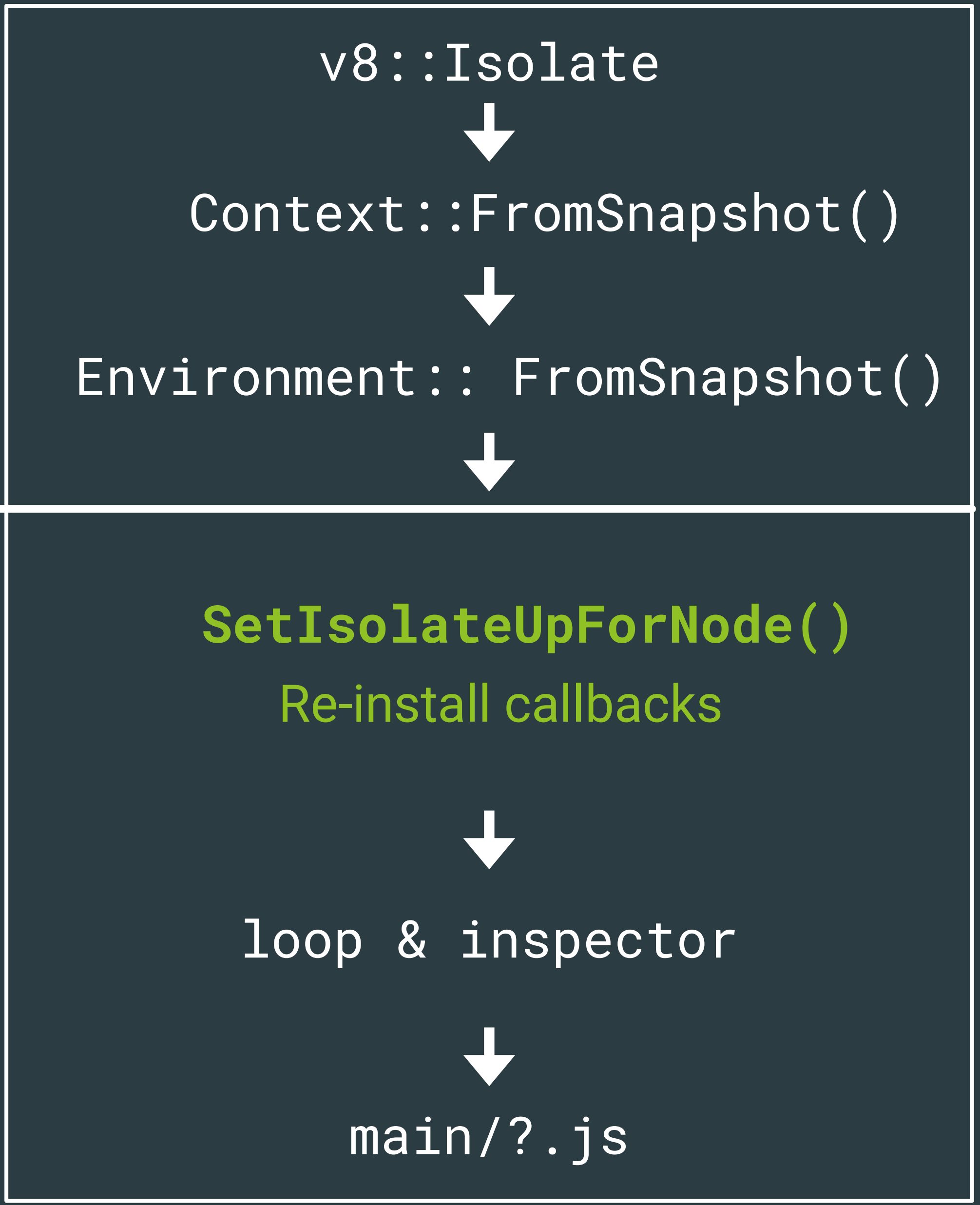
Used to be spaghetti mixed together

Snapshot Integration

```
if (!getOptionValue('--no-warnings') &&
    process.env.NODE_NO_WARNINGS !== '1') {
  process.on('warning', onWarning);
}
```

Refactoring

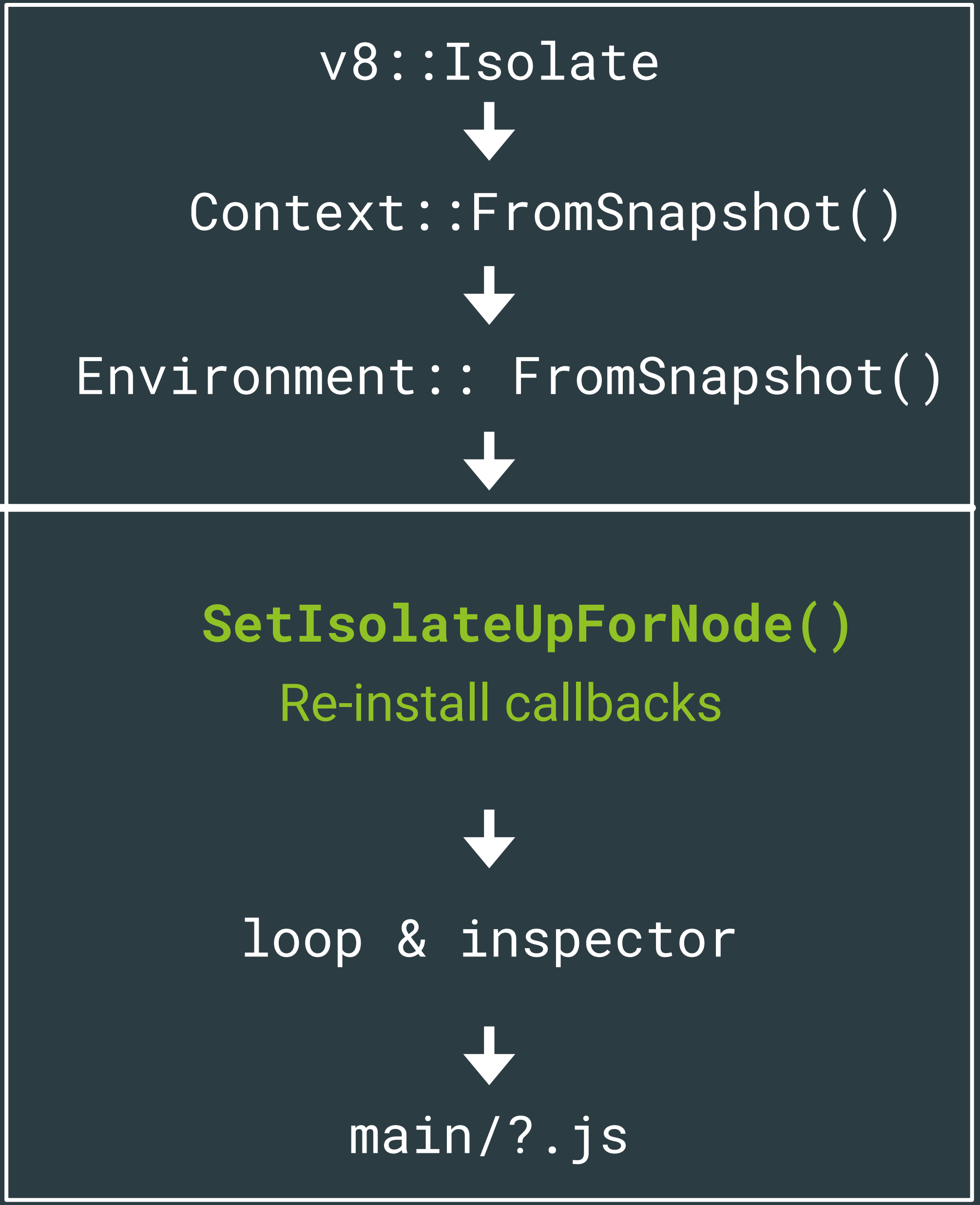
`lib/internal/bootstrap/pre_execution.js`



Snapshot Integration

Refactoring

Reorganize so that we can reinstall C++ states



Current state

- **v12.3.1** v.s. **v10.16.0**
  - ~60% faster child process startup
  - ~120% faster worker startup
  - Some refactoring has also been backported to v10 so the actual speed up is higher

Current state

- **v12.3.1** v.s. **v10.16.0**
  - ~60% faster child process startup
  - ~120% faster worker startup
  - Some refactoring has also been backported to v10 so the actual speed up is higher
- Mostly from lazy-loading and embedded code cache
- Further speedup anticipated from snapshot integration
  - 4x in https://github.com/nodejs/node/issues/17058

Challenges

- Lack of reviews
  - https://github.com/nodejs/node/pull/27539 27 days without reviews
  - Incrementally refactoring the spaghetti code + 7-day wait = slow

Challenges

- Lack of reviews
  - https://github.com/nodejs/node/pull/27539 27 days without reviews
  - Incrementally refactoring the spaghetti code + 7-day wait = slow
- Fixed hash seed
  - Rehashing maps & sets
  - https://bugs.chromium.org/p/v8/issues/detail?id=9187
  - Snapshot is currently still disabled on master behind a build time flag

Future plans

- Finish integration before v12 LTS
- Explore user-land snapshot builder & loader

Thank you