Node.js Core 的启动过程及 V8 Snapshot 集成

张秋怡
Slides: https://github.com/joyeecheung/talks

Node 地下铁，2019.09.08

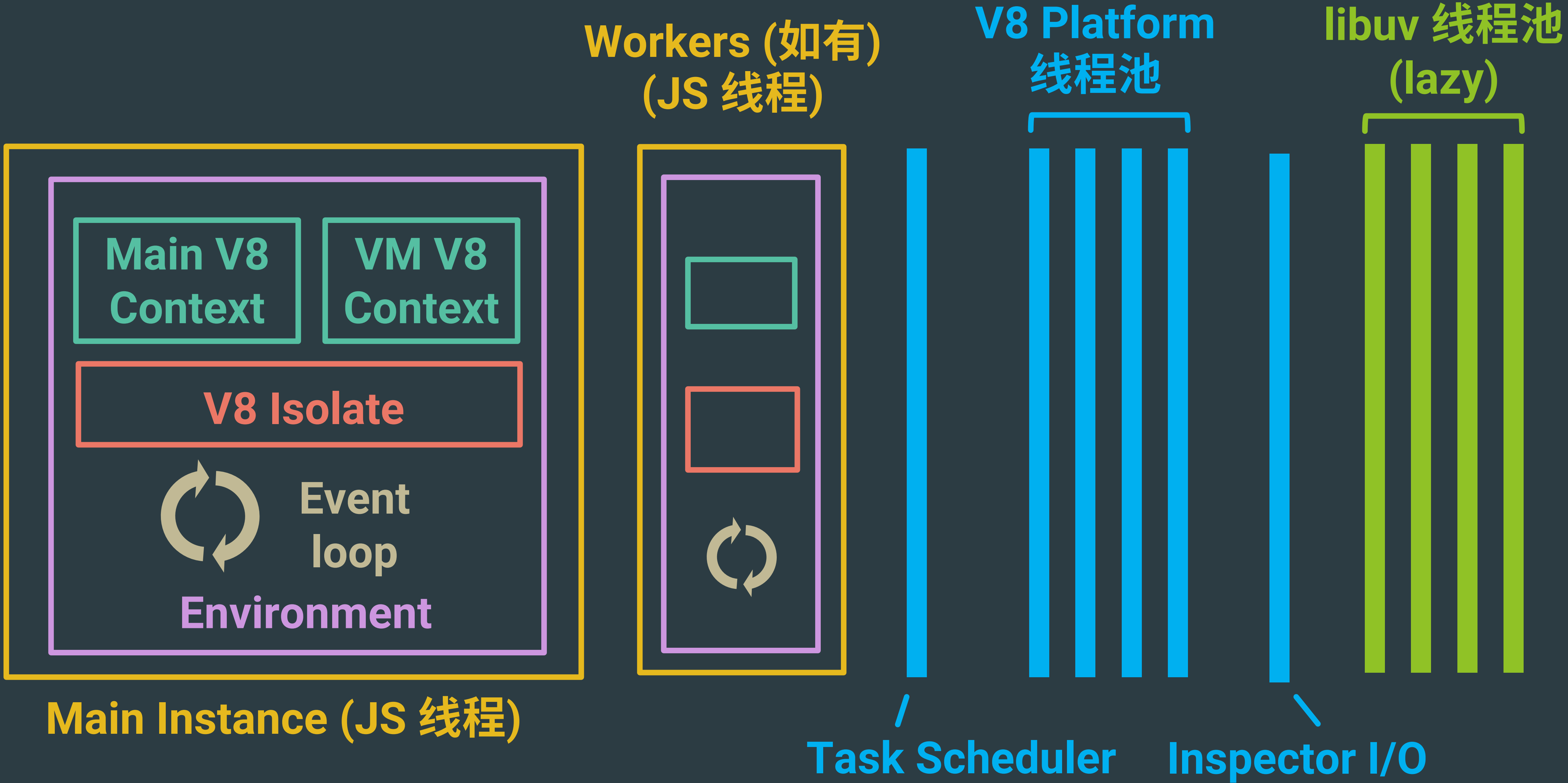
概览

- 从 2018 年底开始进行重构启动过程代码
- **尽可能懒加载**
  - 不初始化不一定用到的东西
  - 重构后初始化依然需要加载至少 60 + 个内部 module

概览

- **嵌入 code cache**
  - 包括编译出的 bytecode 等，预先编译好存储在 binary 内
  - 启动时省去 parse 源代码生成 bytecode，但依然需要执行 bytecode 来完成初始化
- **嵌入 V8 Snapshot**
  - 不是 heap snapshot，是 startup snapshot
  - 直接从预先编译生成的 snapshot 反序列化整个堆，不用执行代码来初始化到需要的状态

Node.js 进程模型


Workers (如有)
(JS 线程)
V8 Platform
线程池
libuv 线程池
(lazy)
Main V8
Context
VM V8
Context
V8 Isolate
Event
loop
Environment
Main Instance (JS 线程)
Task Scheduler
Inspector I/O

Bootstrap (2019.09)

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

Bootstrap (2019.09)

$$\begin{tikzcd}
\texttt{node::Start()} \arrow[d] \\
\texttt{InitializeOncePerProcess()}
\end{tikzcd}$$

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

Bootstrap (2019.09)

`node::Start()`

↓

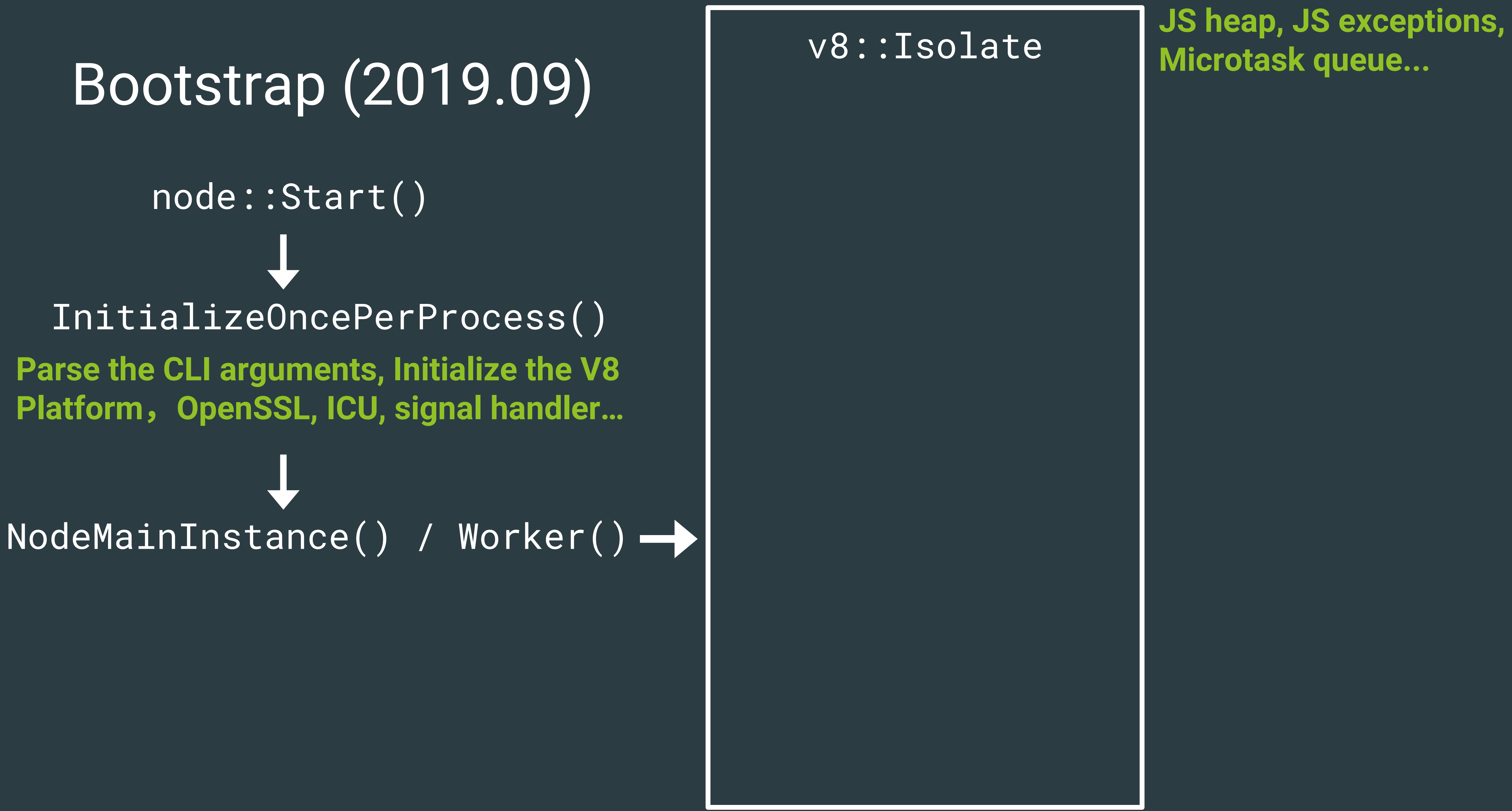
`InitializeOncePerProcess()`

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

↓

`NodeMainInstance() / Worker()`

Bootstrap (2019.09)


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 (2019.09)

`node::Start()`

↓

`InitializeOncePerProcess()`

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

↓

`NodeMainInstance() / Worker()` →

`v8::Isolate`

↓

`v8::Context`

JS heap, JS exceptions, Microtask queue...

global proxy, JS builtins

Bootstrap (2019.09)

`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...**

↓

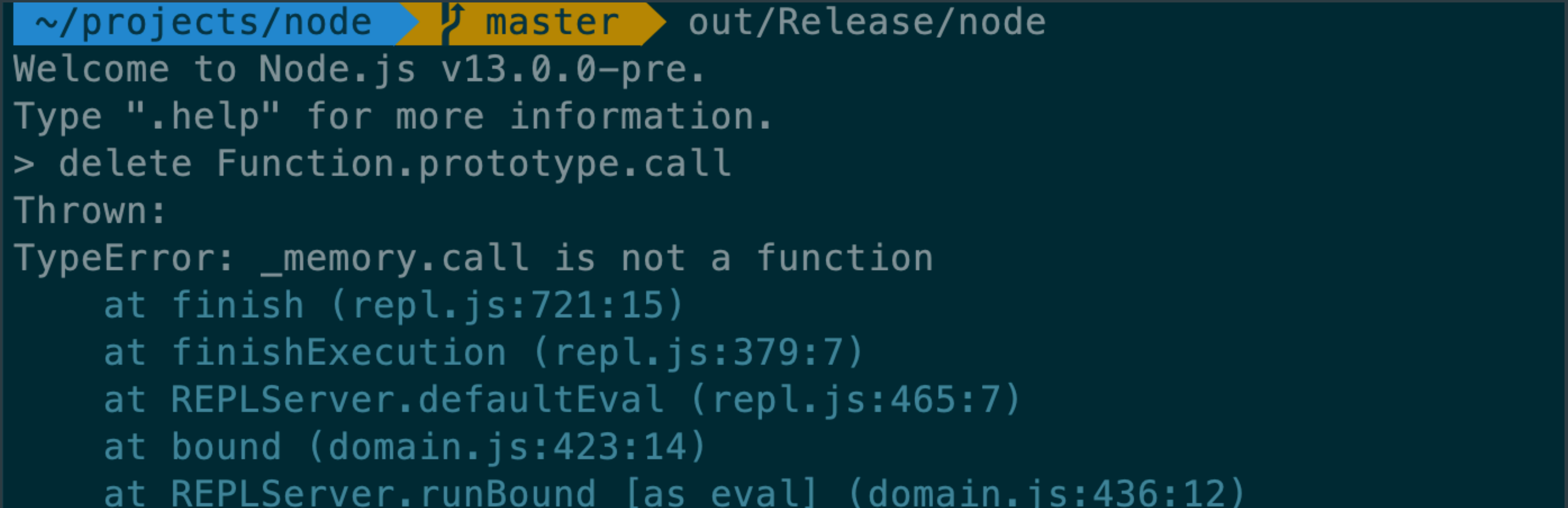
`v8::Context` — **global proxy, JS builtins**

↓

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

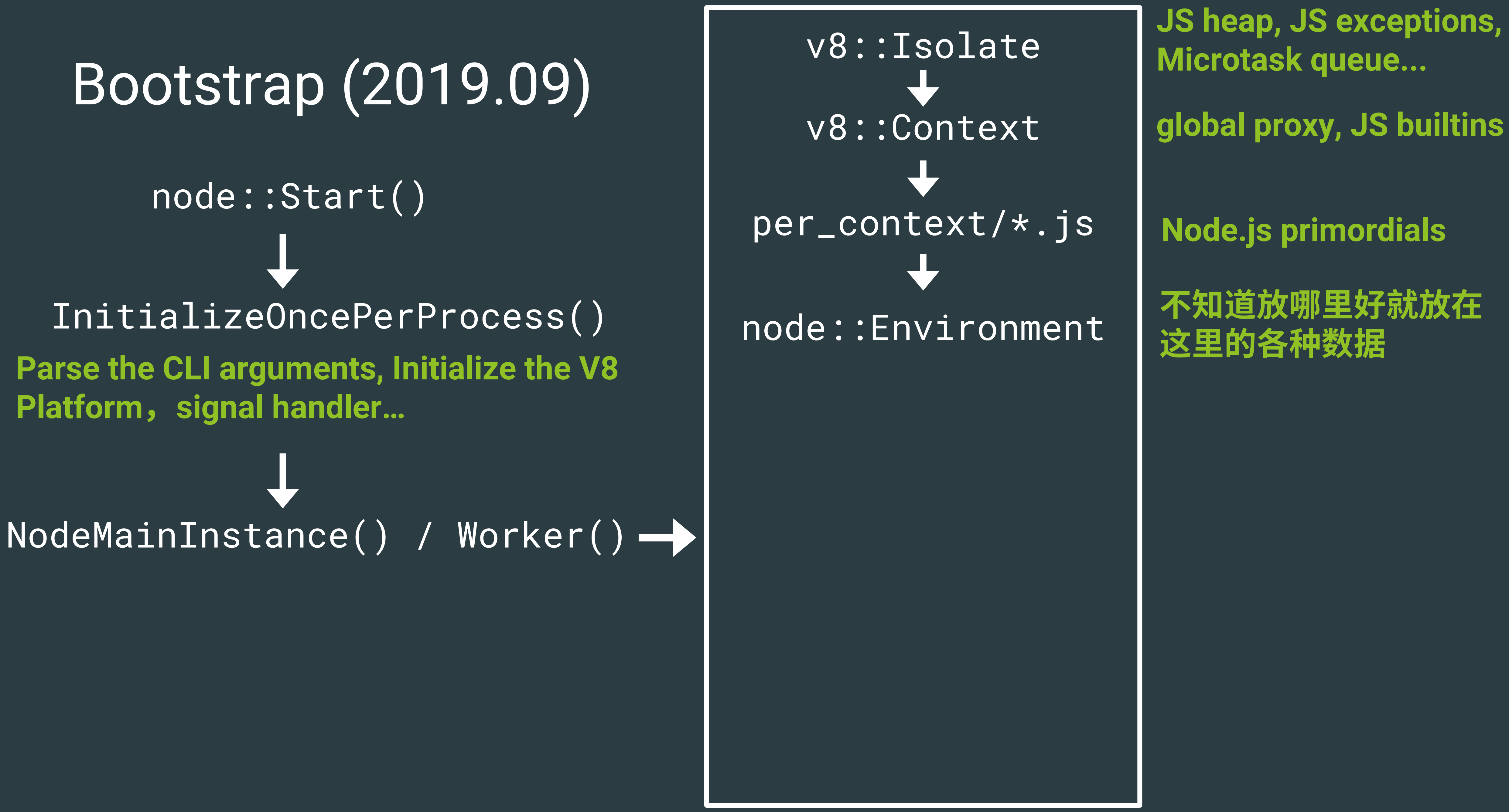
Primordials

- JavaScript builtins 如 `Object`, `Object.prototype` 在启动时会被 clone 进一个对象，并被 `Object.freeze()`，用于给内部代码使用
- 为什么？例：用户可以 `delete Function.prototype.call`
- 逐步迁移所有内部代码到使用这些 primoridials



```
 ~/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:721:15)
    at finishExecution (repl.js:379:7)
    at REPLServer.defaultEval (repl.js:465:7)
    at bound (domain.js:423:14)
    at REPLServer.runBound [as eval] (domain.js:436:12)
```

Bootstrap (2019.09)


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
不知道放哪里好就放在这里的各种数据

Bootstrap (2019.09)

`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**

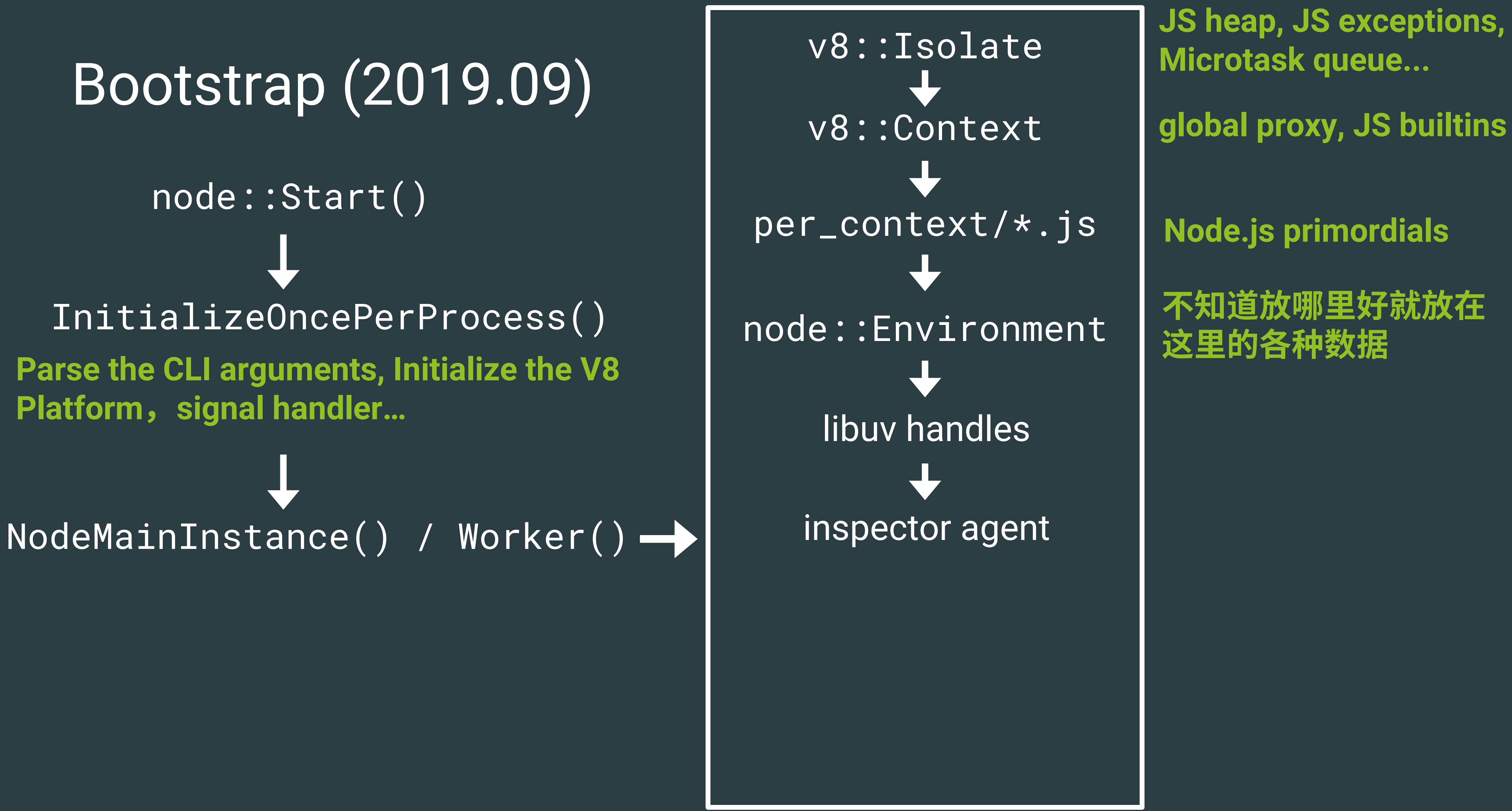
↓

`node::Environment` — **不知道放哪里好就放在这里的各种数据**

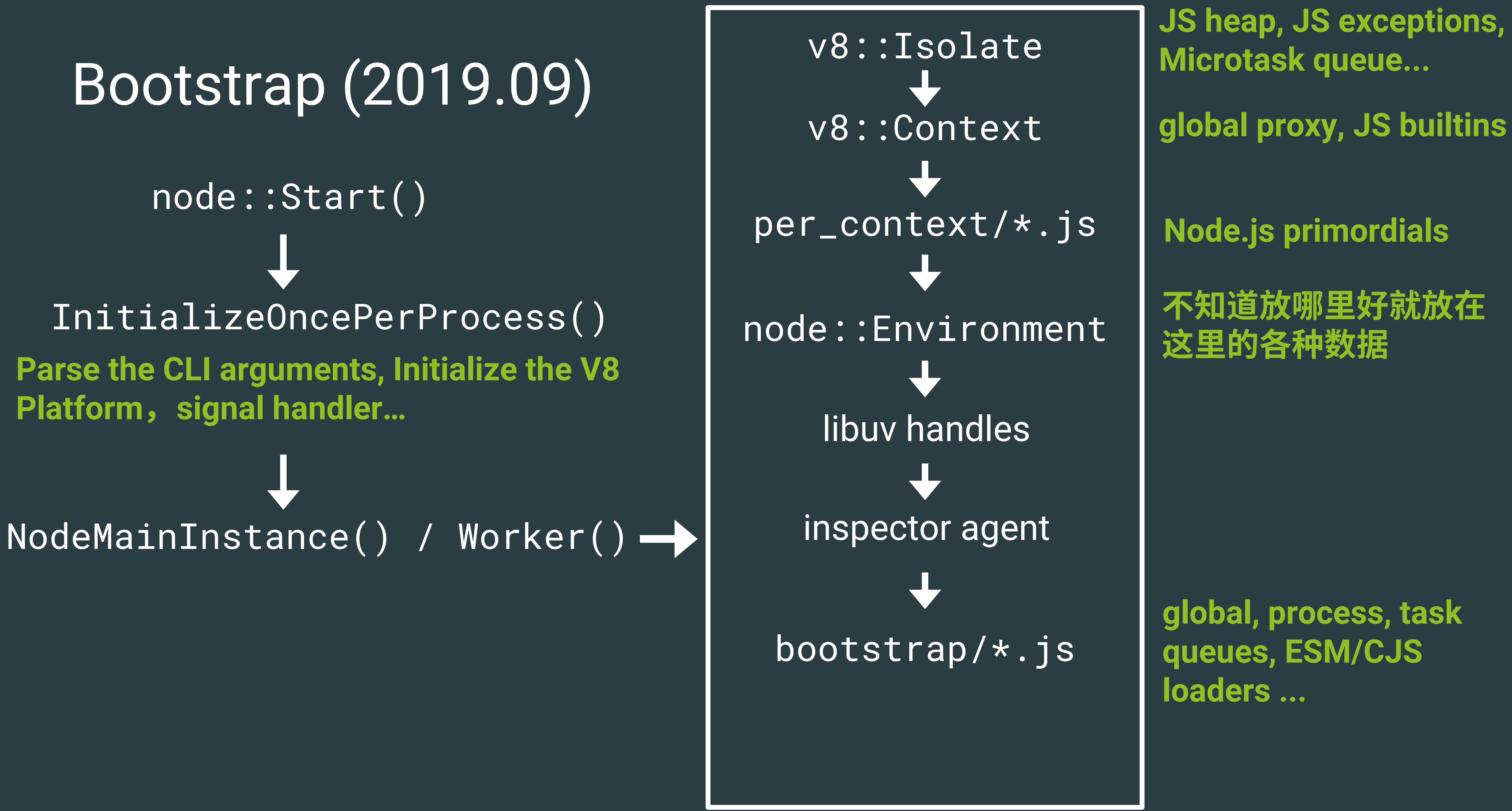
↓

libuv handles

Bootstrap (2019.09)


node::Start()
InitializeOncePerProcess()
Parse the CLI arguments, Initialize the V8 Platform， signal handler...
NodeMainInstance() / Worker()
v8::Isolate
v8::Context
per_context/*.js
node::Environment
libuv handles
inspector agent
JS heap, JS exceptions, Microtask queue...
global proxy, JS builtins
Node.js primordials
不知道放哪里好就放在这里的各种数据

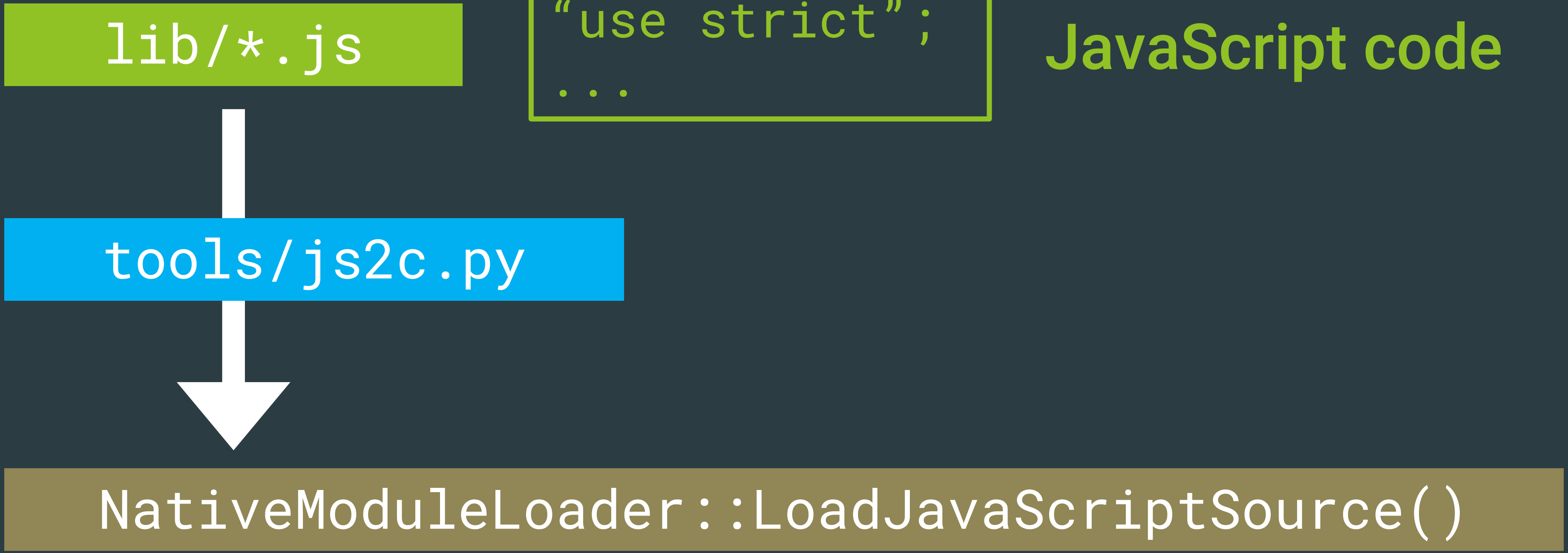
Bootstrap (2019.09)


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
不知道放哪里好就放在这里的各种数据
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)



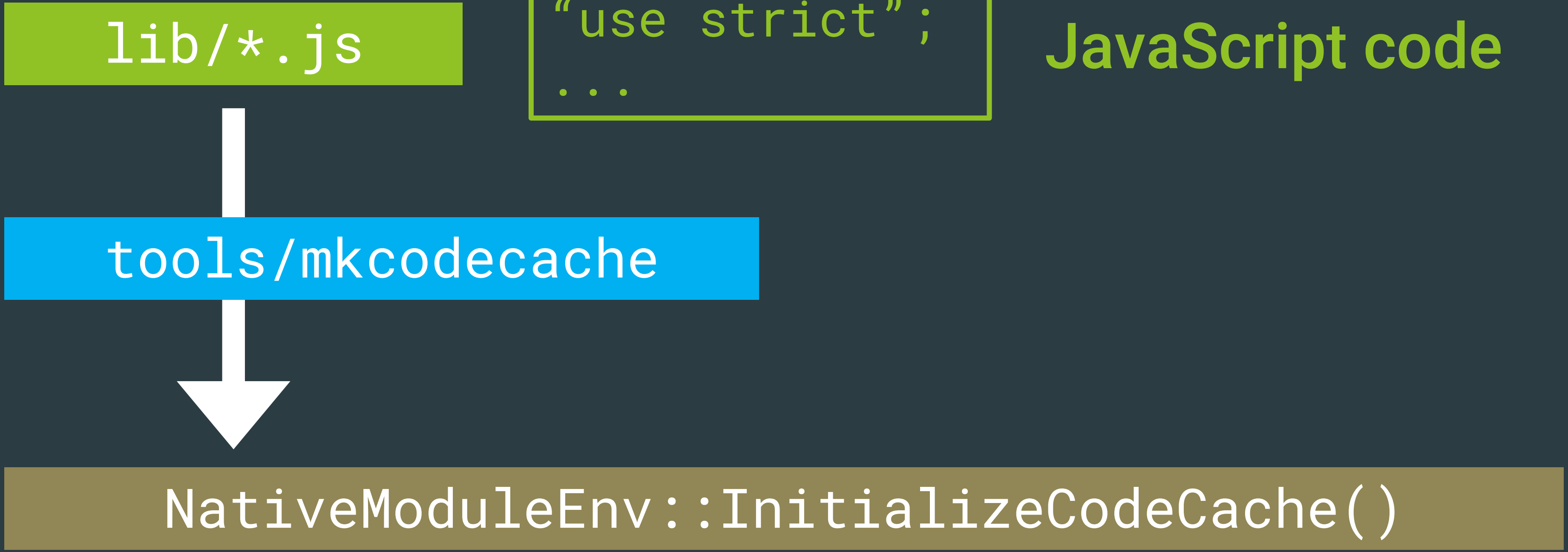
```
“use strict”;
...
```

JavaScript code

```
static const uint16_t timers_raw[] = {
    39,117,115,101...
};
```

static data array 包含 builtins 的源代码

Built-in Modules (Native Modules)



```
“use strict”;
...
```

JavaScript code

```
static const uint8_t assert[] = {
   165,3,222,192,132, ...
};
```

static data array 包含 code cache

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

`lib/internal/bootstrap/node.js`

- 初始化 `process` 和 `global` 上面的大部分成员
- C++ 将 `isMainThread, ownsProcessState` 传入，用于针对不同的场景进行不同的初始化
  - worker 里是 `false`, 主线程里是 `true`

`lib/internal/bootstrap/node.js`

- 初始化将会作为 `v8::Persistent` 存储在 `Environment` 的 JavaScript callbacks
  - Async hook callbacks
  - Timers & `process.nextTick()` schedulers
- 不能进行任何异步操作 (无法从 snapshot 还原)
- 不能依赖任何命令行参数和环境变量。如果打包进 snapshot，编译环境（Node.js 发布用的集群）和运行环境（用户机器）不一致会导致大部分参数无法生效。

lib/internal/bootstrap/pre_execution.js

- 被 main scripts 加载 （见后），不主动运行
- 主要根据命令行参数和环境变量进行进一步的初始化
    - 包括 CJS & ESM loaders
- 不包含在 snapshot

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

User land CommonJS Modules

- Loader 实现在 `lib/internal/modules/cjs/`

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

Wrap user code with objects initialized by Node.js

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

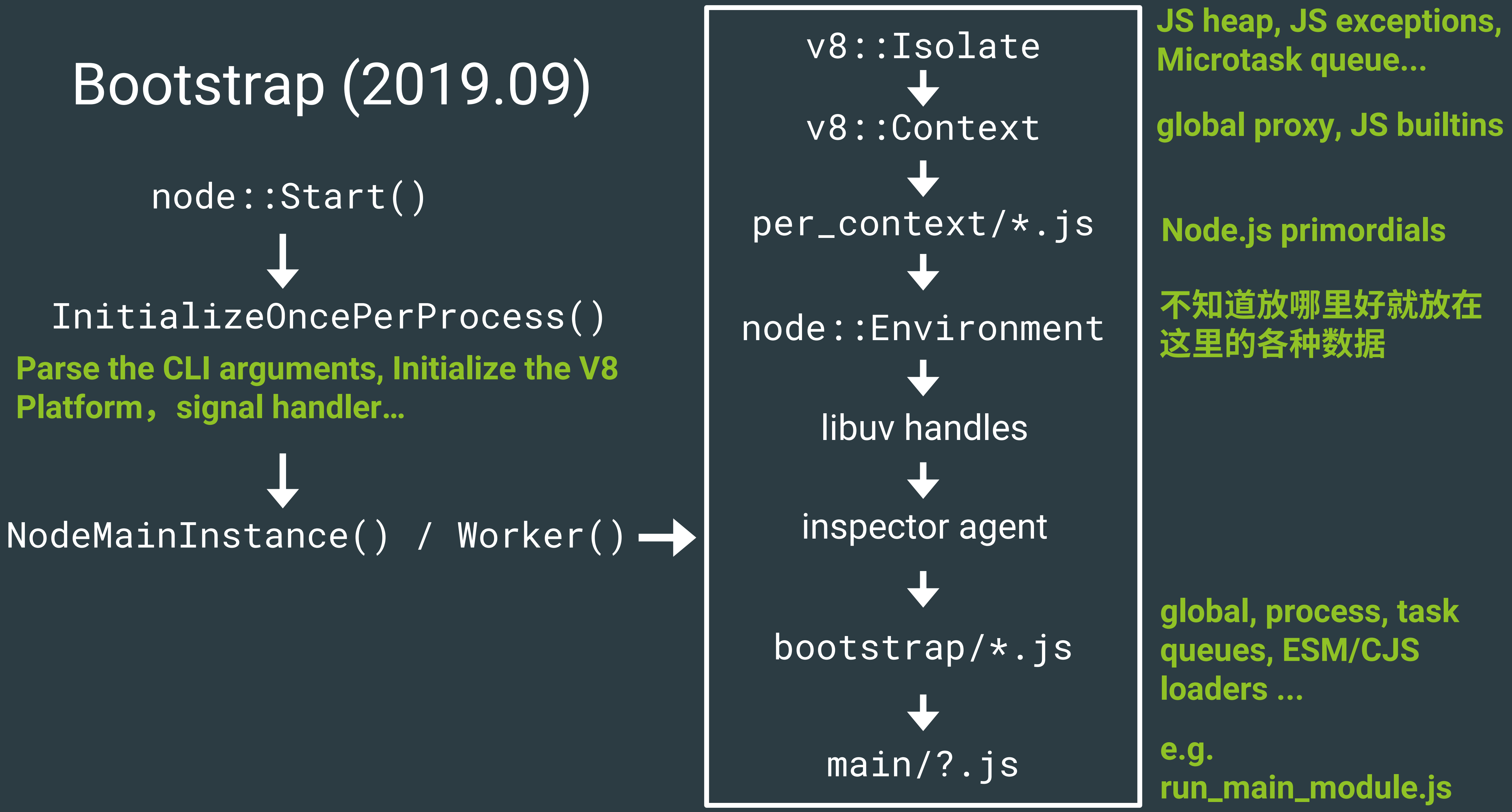
User land ECMAScript Modules

- Loader 实现在 `lib/internal/modules/esm/`
- 不改变 context，只有 bootstrap scripts 往 global proxy 注入的全局变量（这部分不分 ESM 和 CJS）
  - `Buffer`，`process`, etc.

User land ECMAScript Modules

- 一个内部的 `WeakMap` 包含 `ModuleWrap -> Options`
  - `Options` 包括 dynamic `import()`的 callback 和 `import.meta` data
  - Per-isolate
    - `HostImportModuleDynamicallyCallback`
    - `HostInitializeImportMetaObjectCallback`

Bootstrap (2019.09)


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
不知道放哪里好就放在这里的各种数据
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`
- 主线程
  - `StartMainThreadExecution()`
  - 根据命令行参数等条件，选择一个 main script 运行
- Workers
  - `worker_thread.js`
- 先加载 `lib/internal/bootstrap/pre_execution.js` 根据运行环境进行初始化

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`

Main scripts

- `run_third_party_main.js`
  - 运行 embedders 嵌入的 `lib/_third_party_main.js`
- `environment.js`
  - For C++ test fixtures

Requested features

- Customize entry point for bundled CLI tools
- Better entry points 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

Bootstrap (2019.09)

`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` — 不知道放哪里好就放在这里的各种数据

↓

libuv handles

↓

inspector agent

↓

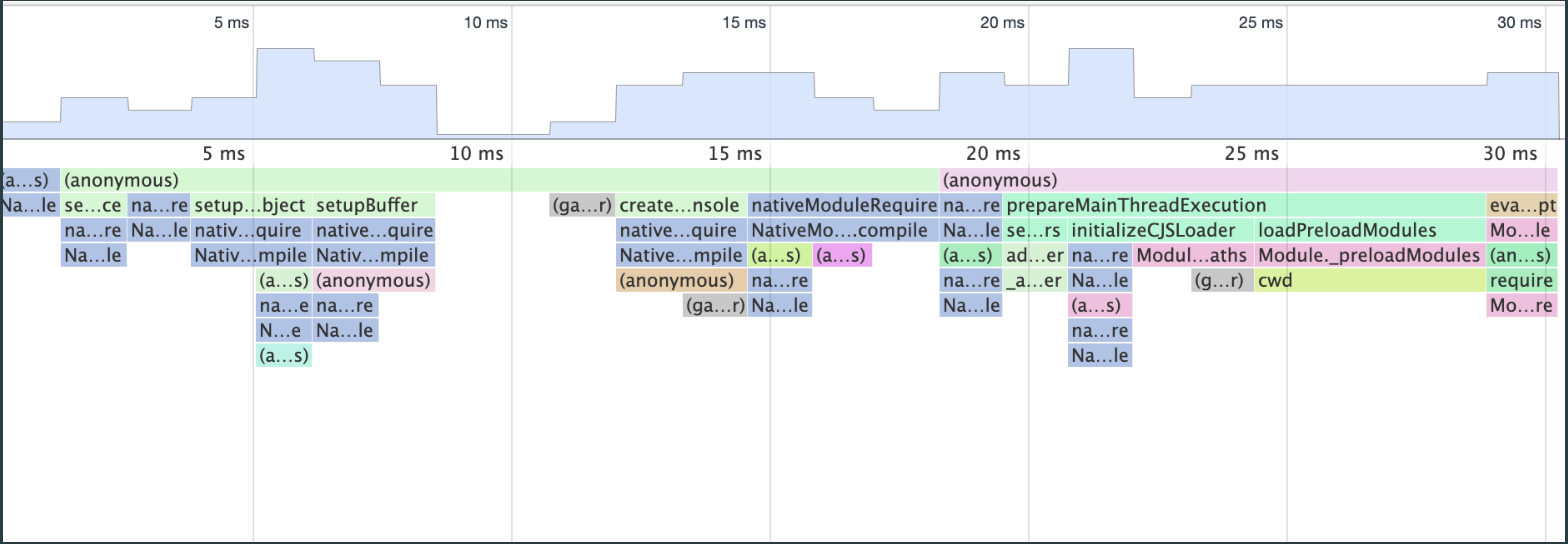
`bootstrap/*.js` — global, process, task queues, ESM/CJS loaders ...

↓

`main/?.js` — e.g. run_main_module.js

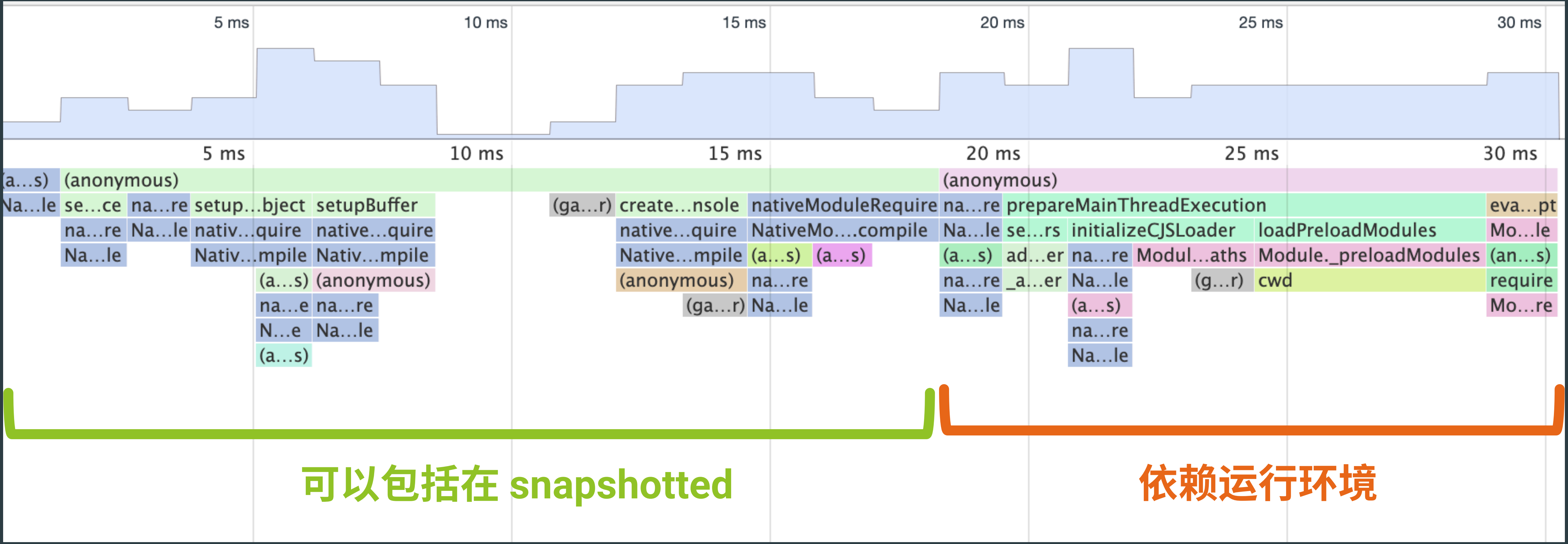
Snapshot 集成

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

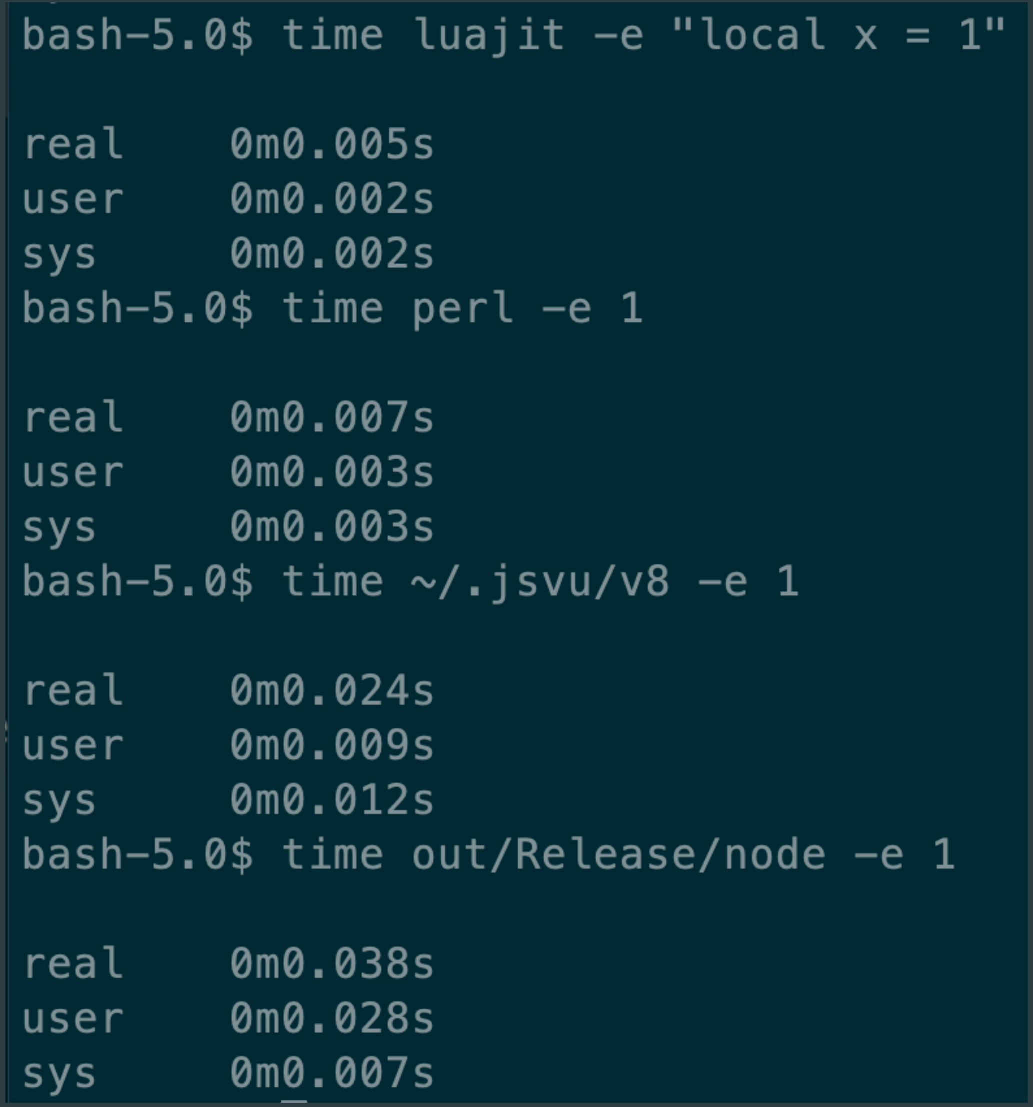


Snapshot 集成

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



Snapshot 集成



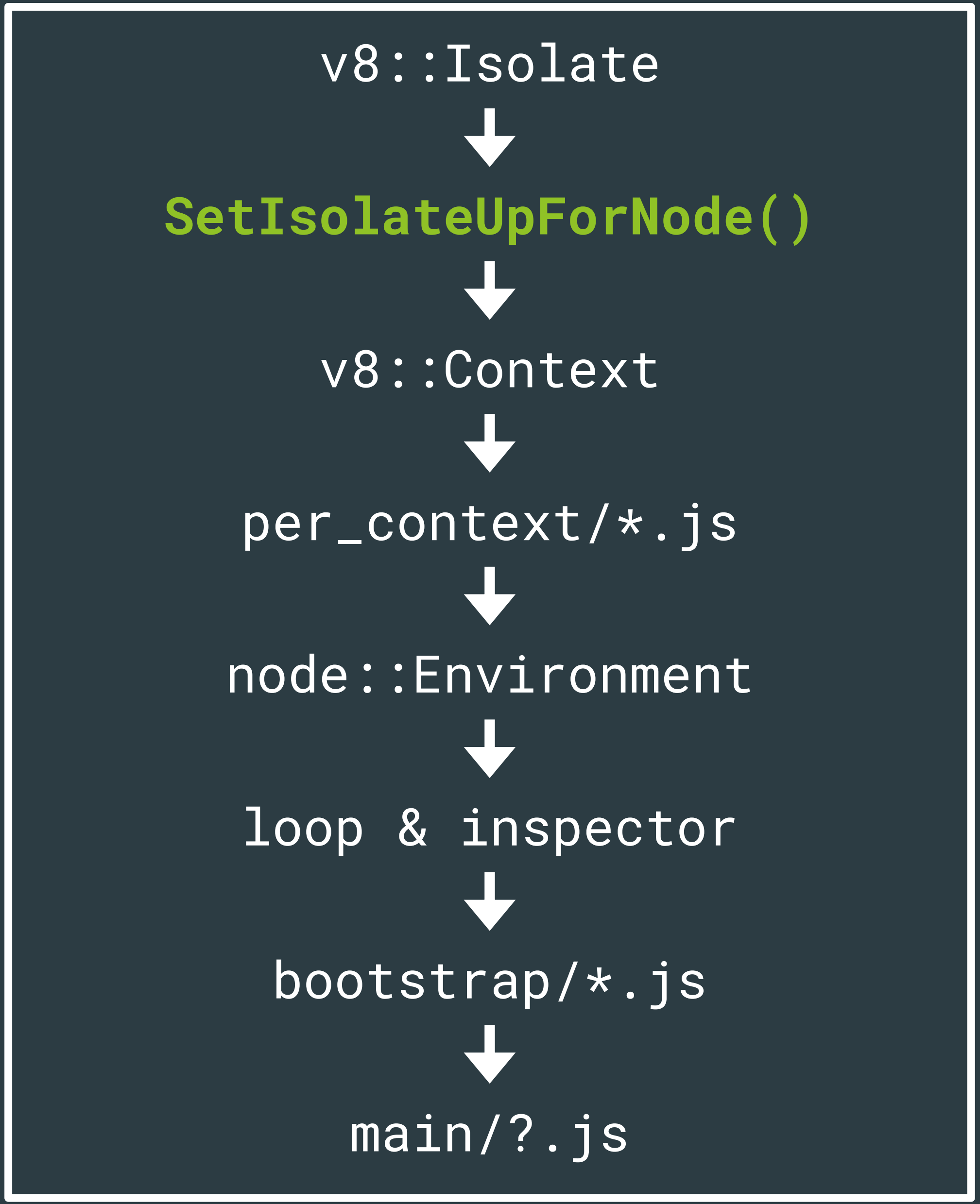
```
bash-5.0$ time luajit -e "local x = 1"

real	0m0.005s
user	0m0.002s
sys	0m0.002s
bash-5.0$ time perl -e 1

real	0m0.007s
user	0m0.003s
sys	0m0.003s
bash-5.0$ time ~/.jsvu/v8 -e 1

real	0m0.024s
user	0m0.009s
sys	0m0.012s
bash-5.0$ time out/Release/node -e 1

real	0m0.038s
user	0m0.028s
sys	0m0.007s
```

d8 with default snapshot

node master without snapshot

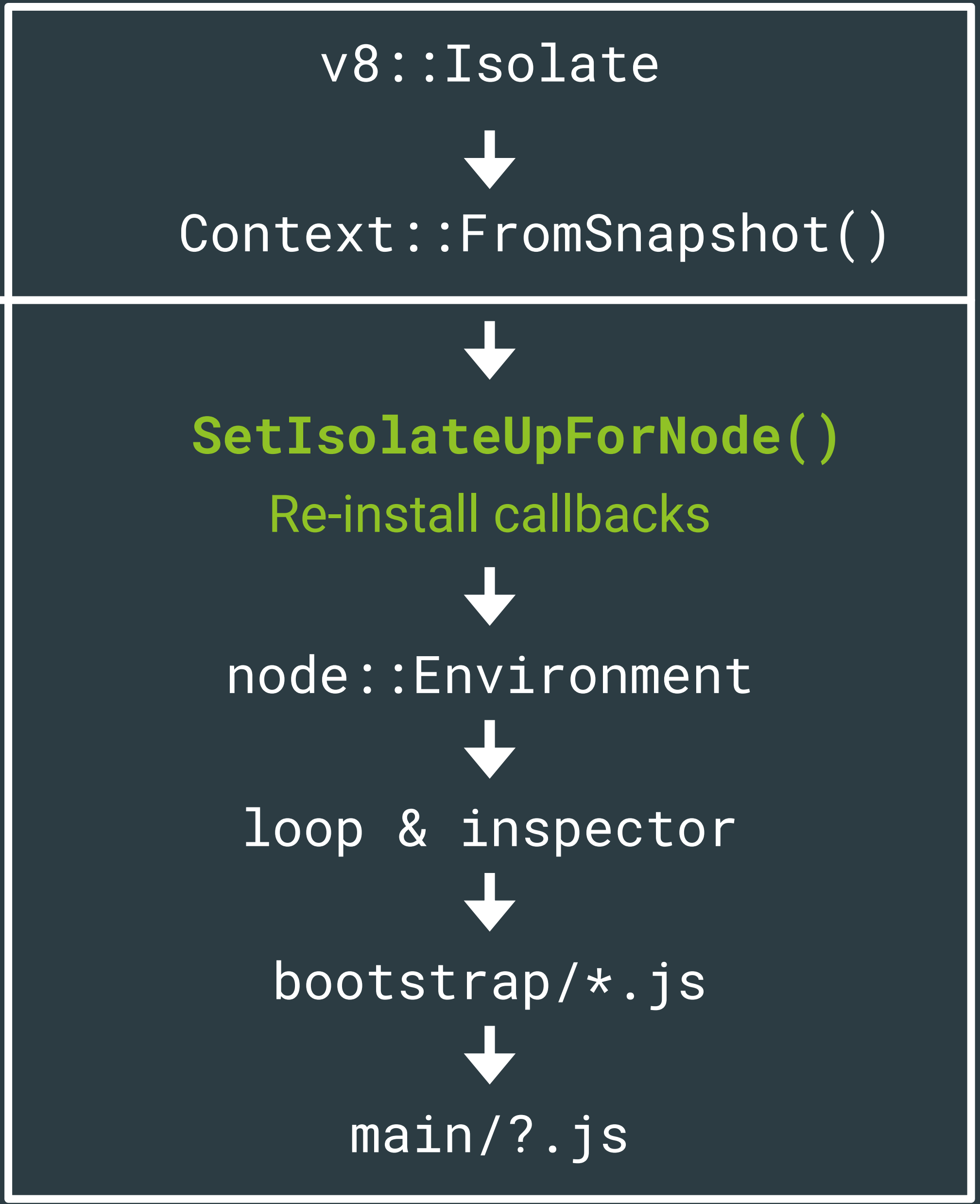
Snapshot 集成

Original



Snapshot 集成

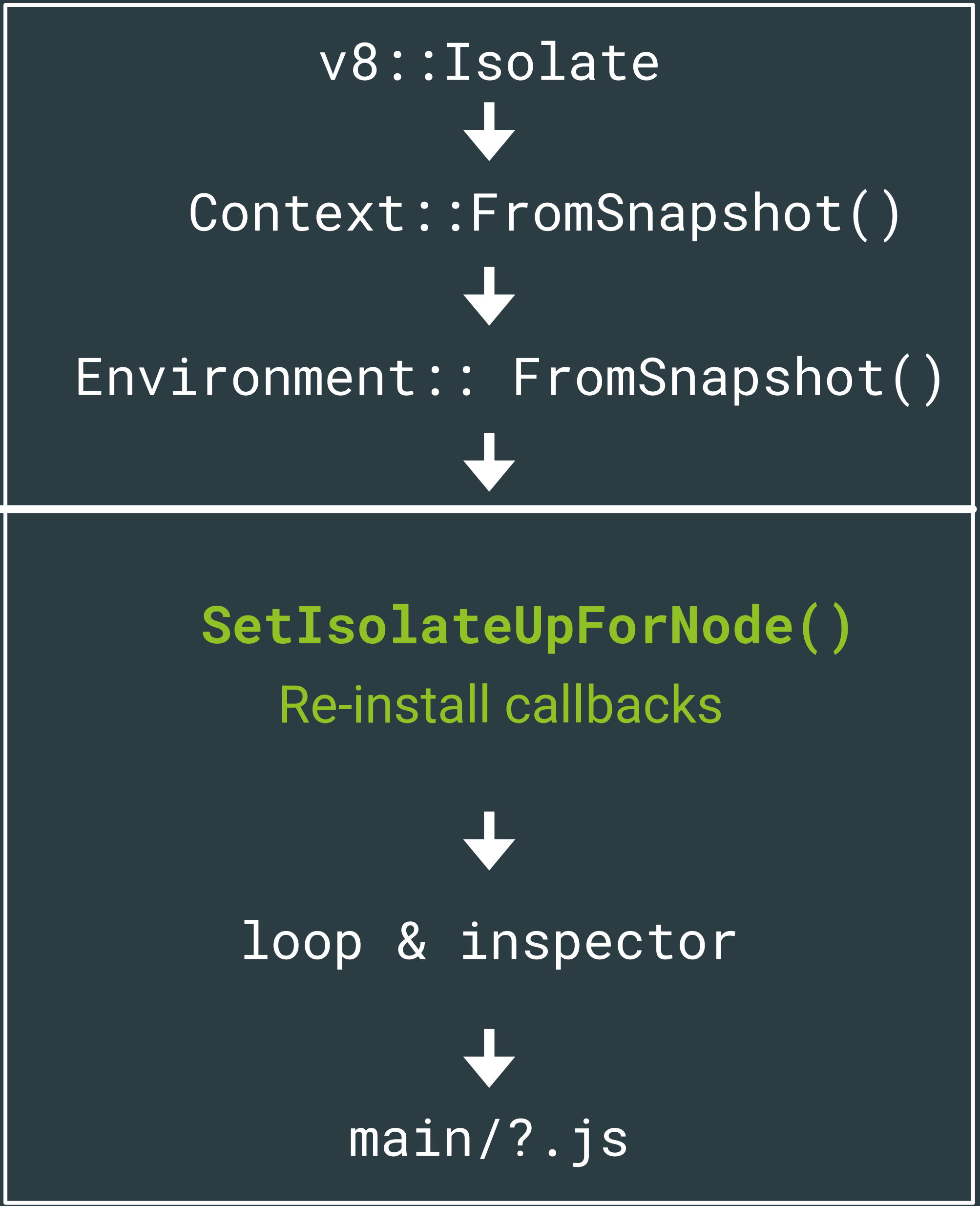
Snapshotted (2019.05)



Snapshot 集成

目标

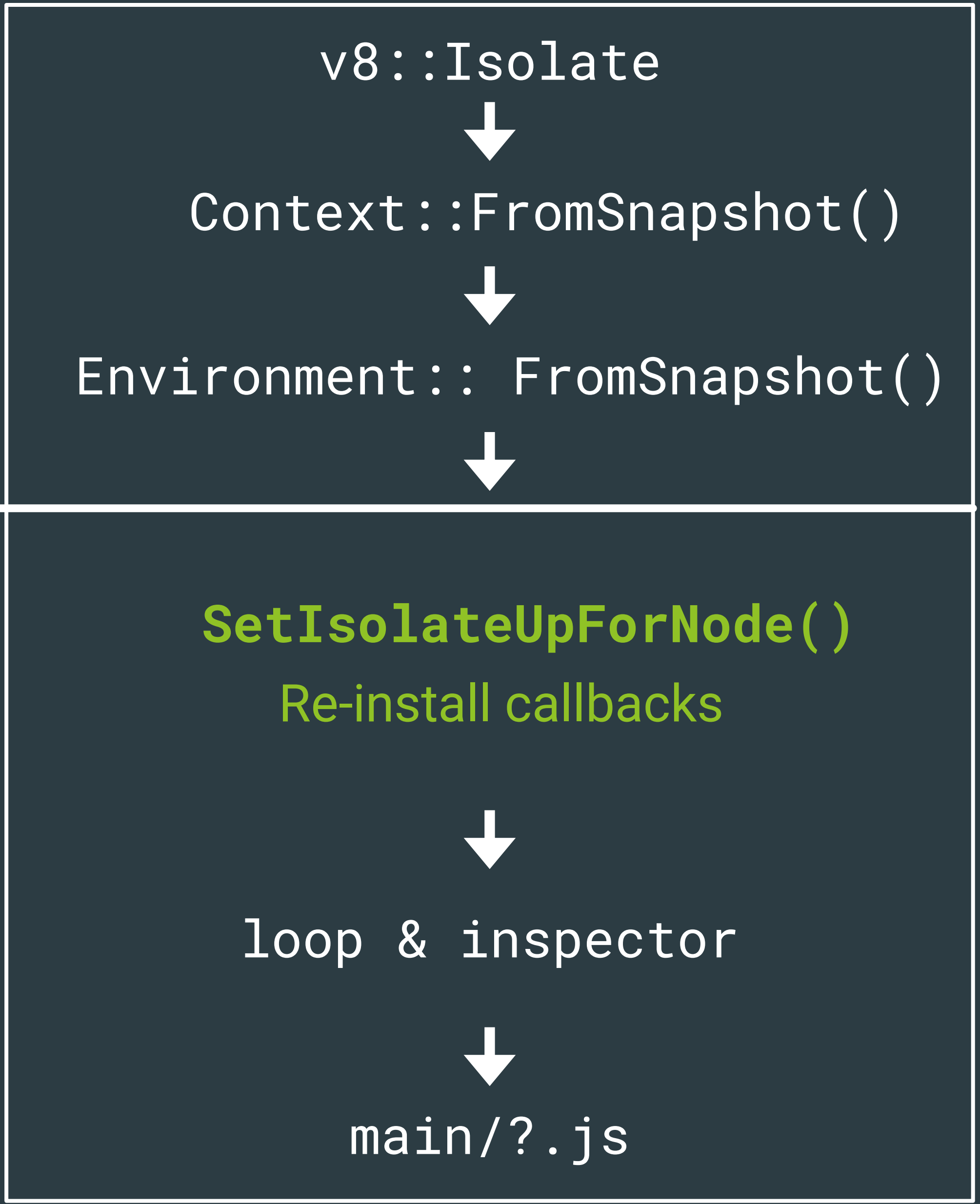
直接从 snapshot 加载部分初始化好的环境，
而不是从头执行 `per_context/*.js` & `bootstrap/*.js` 来初始化



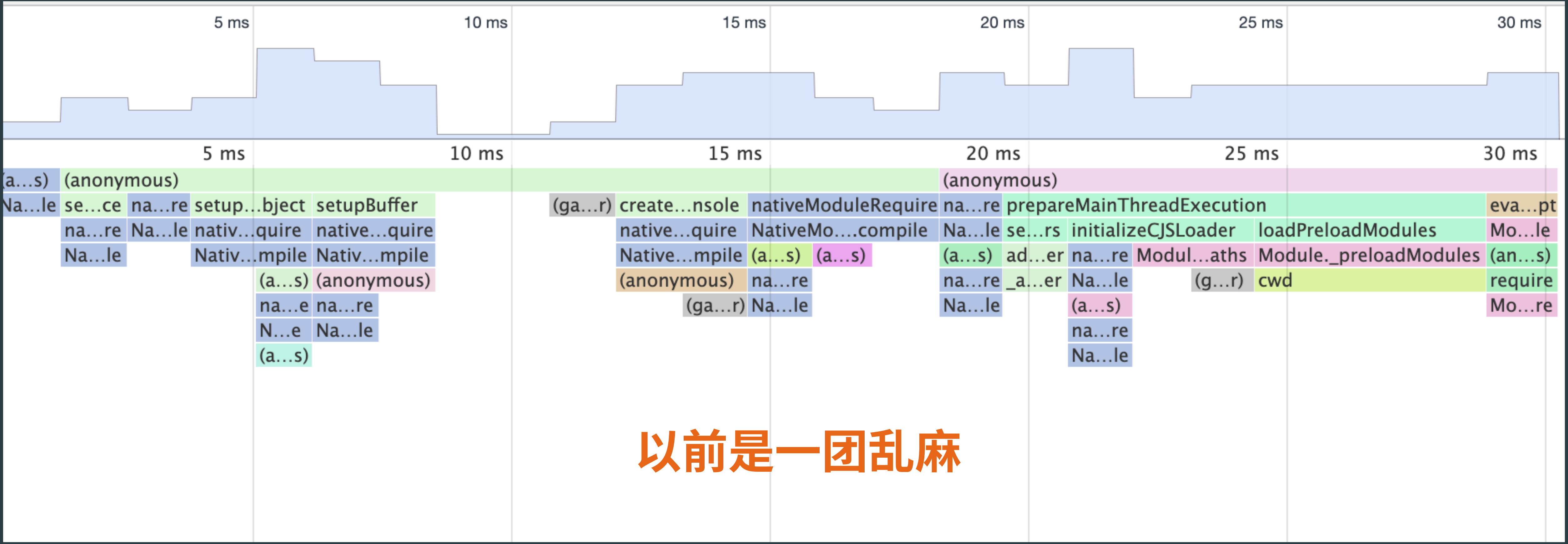
Snapshot 集成

重构

在截取 snapshot 前的启动流程必须不能依赖运行环境



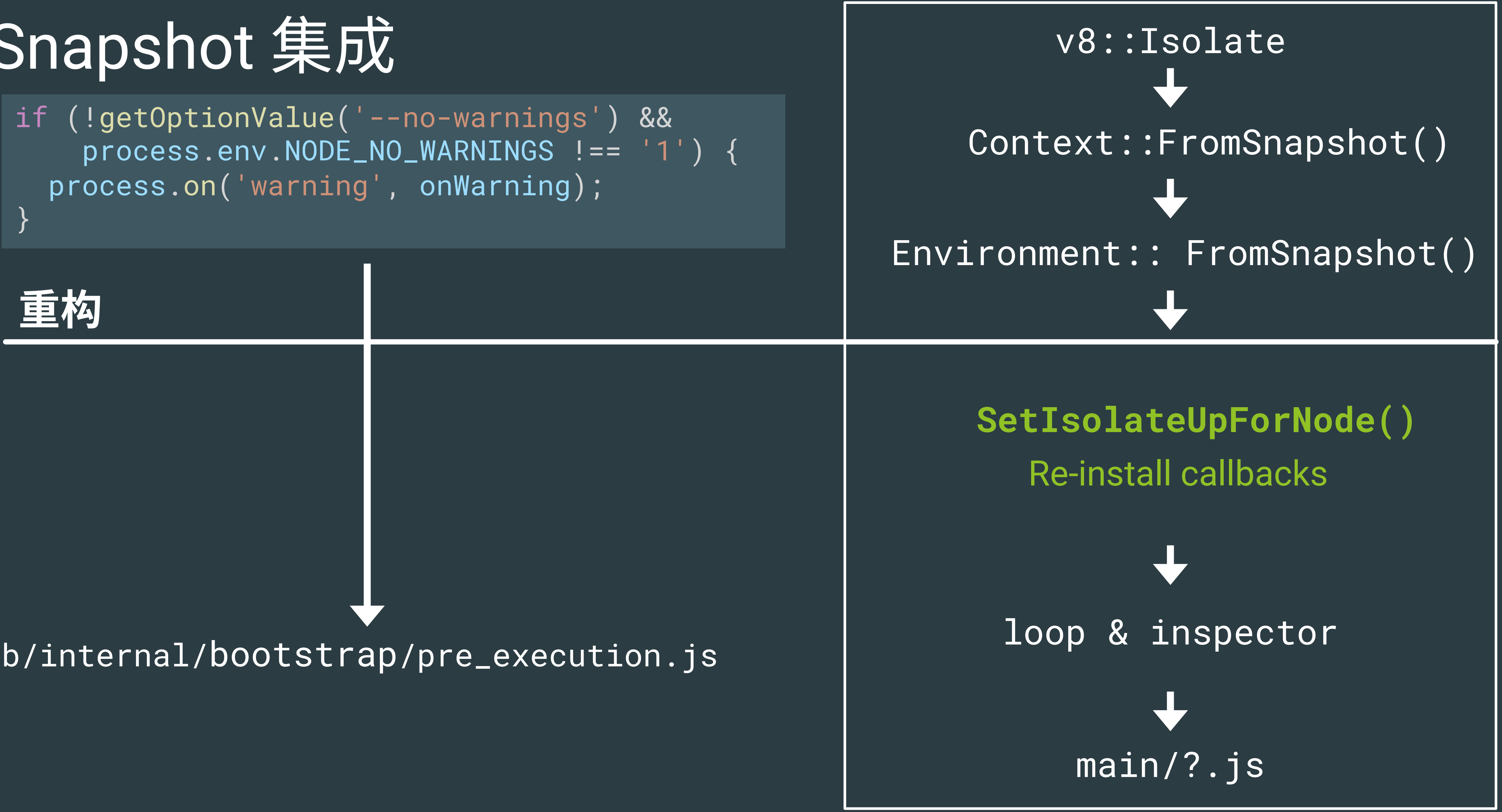
Snapshot 集成



以前是一团乱麻

Snapshot 集成

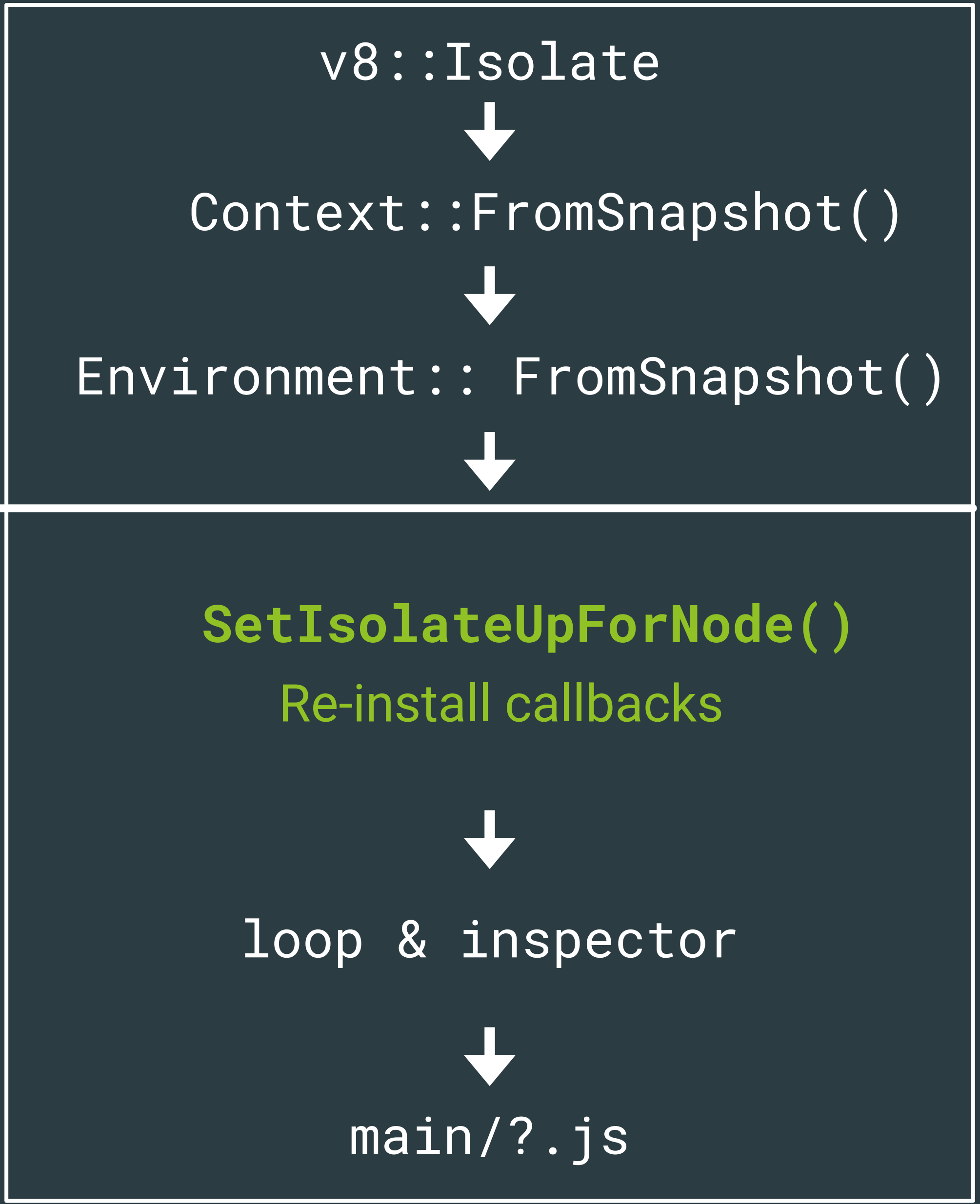
```
if (!getOptionValue('--no-warnings') &&
    process.env.NODE_NO_WARNINGS !== '1') {
  process.on('warning', onWarning);
}
```



Snapshot 集成

重构

调整步骤，需要在 snapshot 加载后重新同步 C++ 部分的状态（snapshot 里只保存了 JavaScript 的状态)



目前进展

- Per-context scripts are snapshotted and shipped by default since v12.5.0
- **v12.5.0** v.s. **v11.2.0**
  - ~60% faster child process startup
  - ~120% faster worker startup
- 大部分提升来自重构（更多懒加载）和 embedded code cache

进行中的工作

- V8 issues
  - Rehashing Map & Set
  - 优化 `v8::External`
  - Lazy initialization of ICU
- V8 部分解决后，`bootstrap/loaders.js` 和 `bootstrap/node.js` 的运行结果可以包含入embedded snapshot
  - https://github.com/nodejs/node/issues/17058 中的初步实现提升为 4x
  - 实际可能低一些，因为这个原型包含了一些不应该被截进去的状态

Future plans

- User-land snapshot builder & loader
  - 改进开发过程中的错误提示
    - 遍历到 context-dependent 的对象无法序列化
    - 遇到未知外部引用
  - 如何整合 C++ addons?
  - Warm up
  - 对包含在 snapshot 内的运行状态（e.g. 环境变量）进行提示
    - 用户自己打包的应用可以自行保证编译环境和运行环境一致

Thank you