

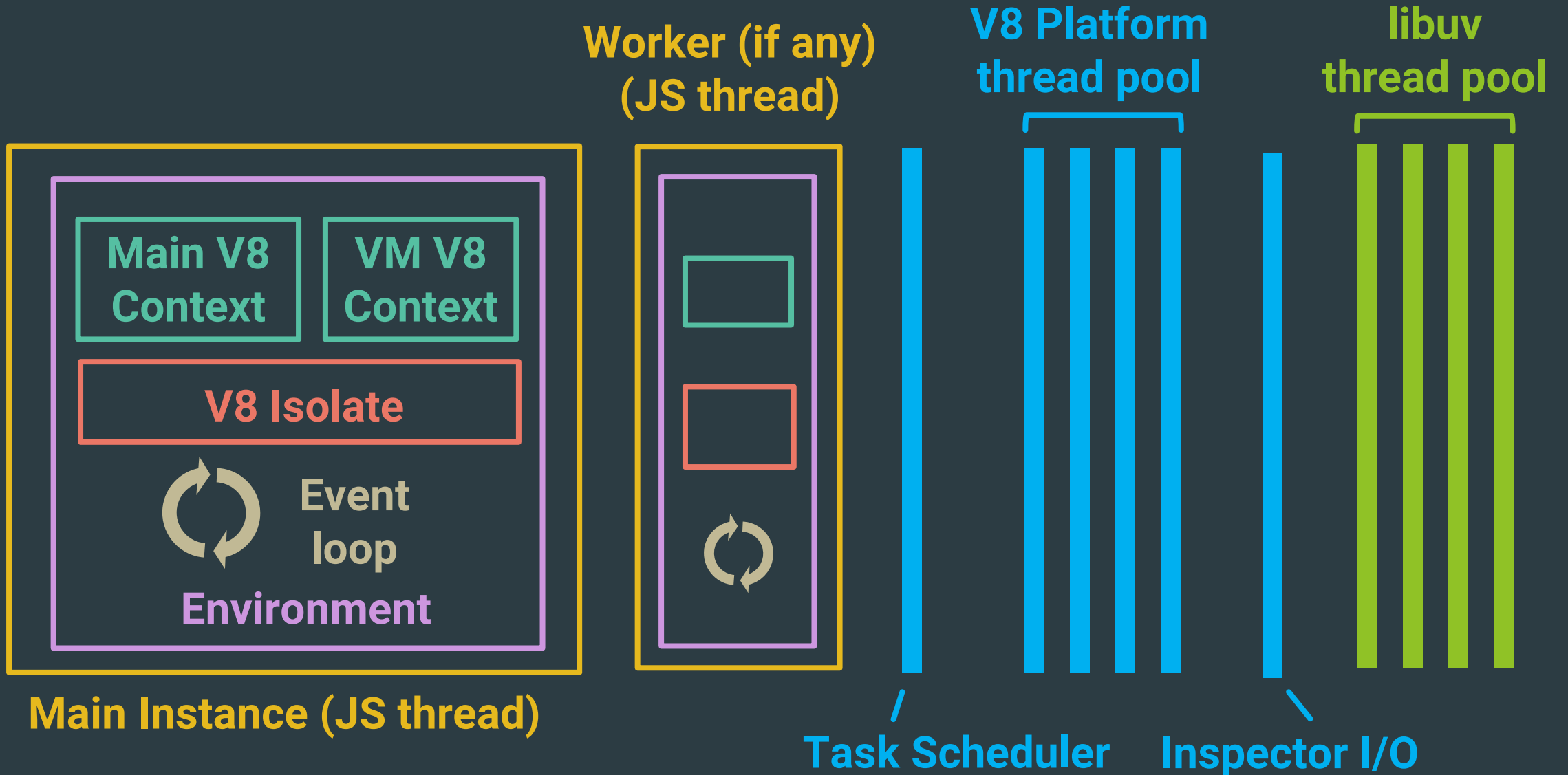
# Bootstrap of Node.js Core

**Joyee Cheung**

Slides: <https://github.com/nodejs/summit/issues/147>

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)

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 (2019.05)

node::Start()



InitializeOncePerProcess()

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



NodeMainInstance() / Worker() →

v8::Isolate



v8::Context



per\_context/\*.js

**JS heap, JS exceptions,  
Microtask queue...**

**global proxy, JS builtins**

**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`



`v8::Context`



`per_context/*.js`



`node::Environment`

**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



v8::Context



per\_context/\*.js



node::Environment



libuv handles

**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



v8::Context



per\_context/\*.js



node::Environment



libuv handles



inspector agent

**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`



`v8::Context`



`per_context/*.js`



`node::Environment`



`libuv handles`



`inspector agent`



`bootstrap/*.js`

**JS heap, JS exceptions,  
Microtask queue...**

**global proxy, JS builtins**

**Node.js primordials**

**Stuff that does not  
have a better place to  
go**

**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)

lib/\*.js

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

JavaScript code

tools/js2c.py

NativeModuleLoader::LoadJavaScriptSource()

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

static data array  
containing the source

# Built-in Modules (Native Modules)

lib/\*.js

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

JavaScript code

tools/mkcodecache

NativeModuleEnv::InitializeCodeCache()

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

static data array  
containing the 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

- ▶ 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)
- ▶ Must not depend on any CLI arguments or environment variables

# lib/internal/bootstrap/pre\_execution.js

- ▶ Not actively run. Required by main scripts (explained later)
- ▶ Bootstrap things that depend on CLI arguments and environment variables
- ▶ 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`  
↓  
`v8::Context`  
↓  
`per_context/*.js`  
↓  
`node::Environment`  
↓  
`libuv handles`  
↓  
`inspector agent`  
↓  
`bootstrap/*.js`  
↓  
`main/? .js`

**JS heap, JS exceptions,  
Microtask queue...**

**global proxy, JS builtins**

**Node.js primordials**

**Stuff that does not  
have a better place to  
go**

**global, process, task  
queues, ESM/CJS  
loaders ...**

**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`

# Main scripts

- ▶ `run_third_party_main.js`
  - ▶ Run `lib/_third_party_main.js` embedders
- ▶ `environment.js`
  - ▶ For C++ test fixtures

## Requested

- ▶ bundled cli tool entry point?
- ▶ better entry point 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 initialization

```
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()` →

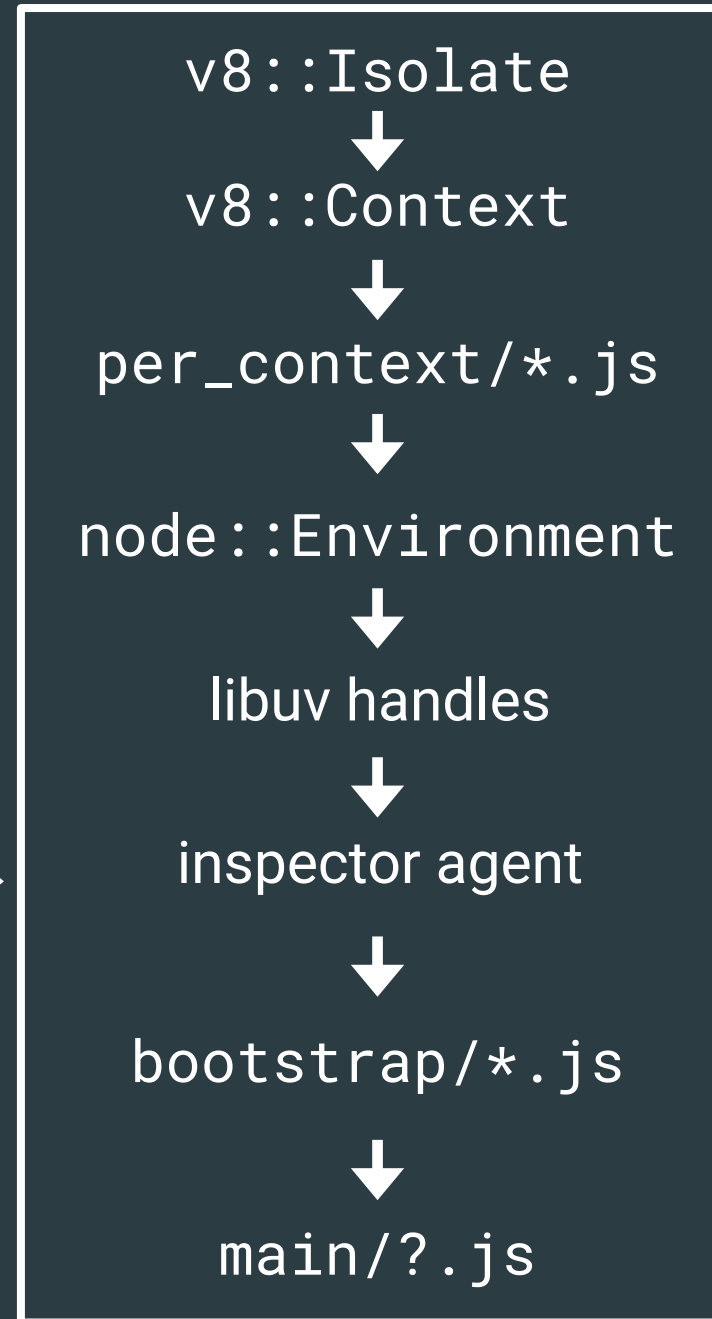


`do {`

`uv_run(...)`

`} while (...)`

**Event Loop**



**JS heap, JS exceptions, Microtask queue...**

**global proxy, JS builtins**

**Node.js primordials**

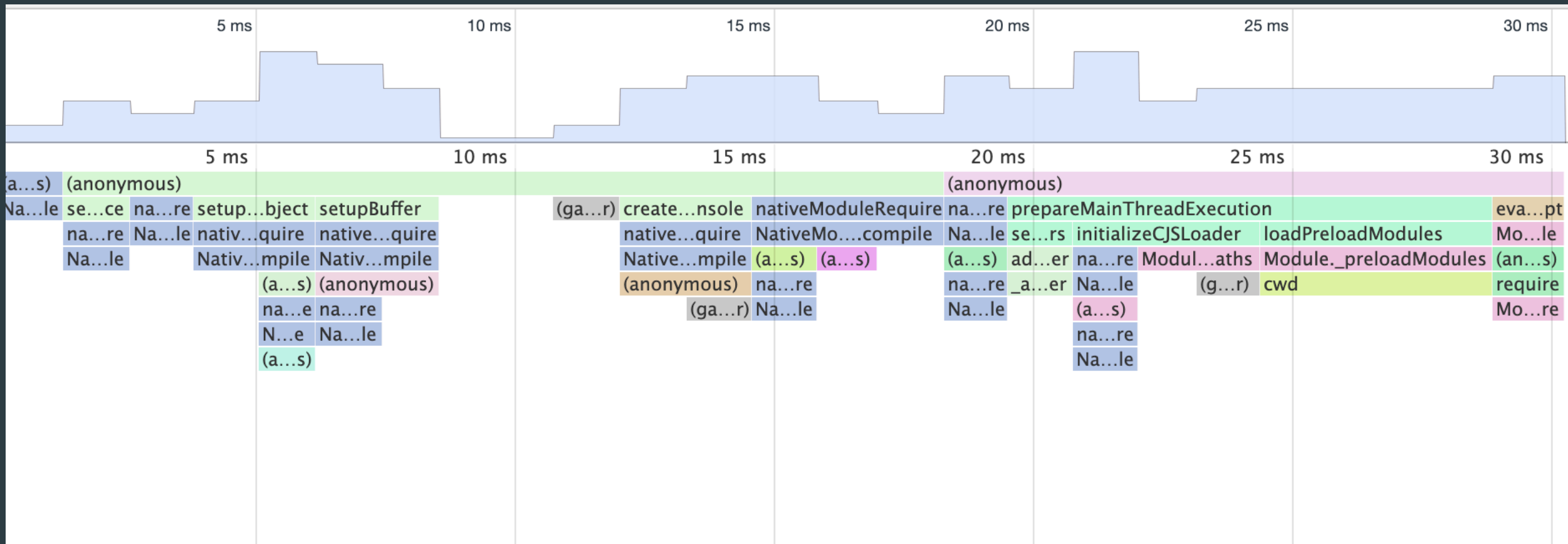
**Stuff that does not have a better place to go**

**global, process, task queues, ESM/CJS loaders ...**

**e.g. `run_main_module.js`**

# Snapshot Integration

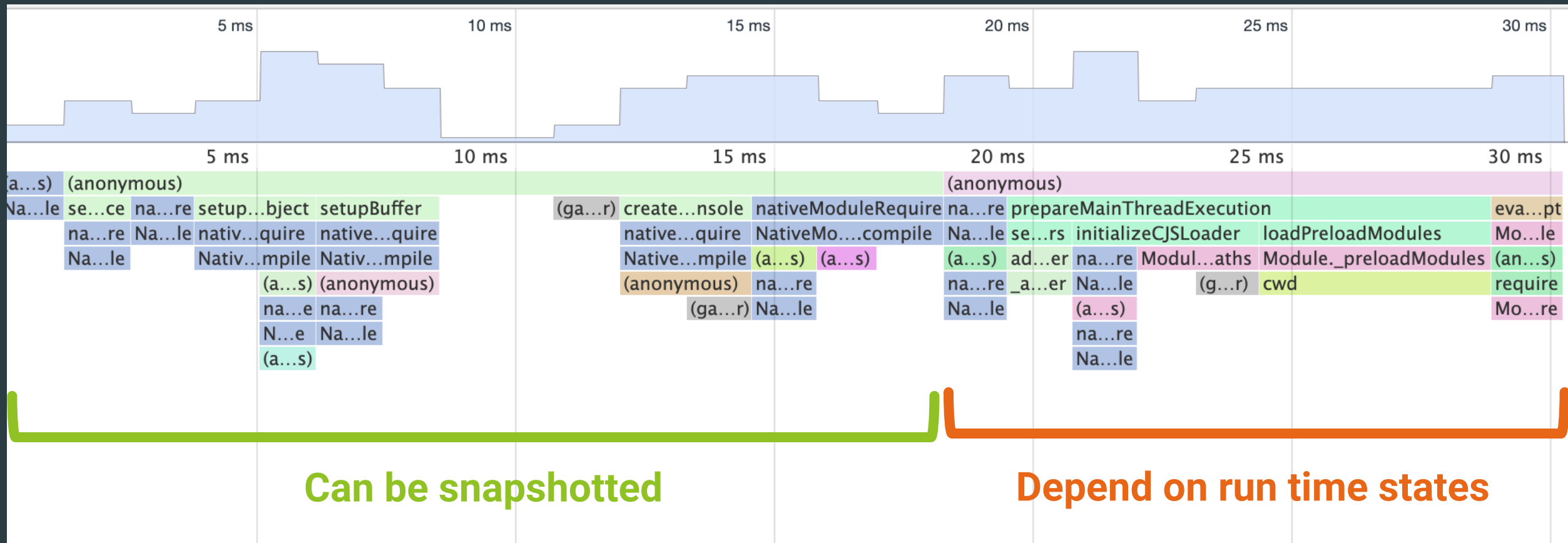
```
out/Release/node --cpu-prof-interval=100 --cpu-prof -e "{}"
```





# Snapshot Integration

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



# Snapshot Integration

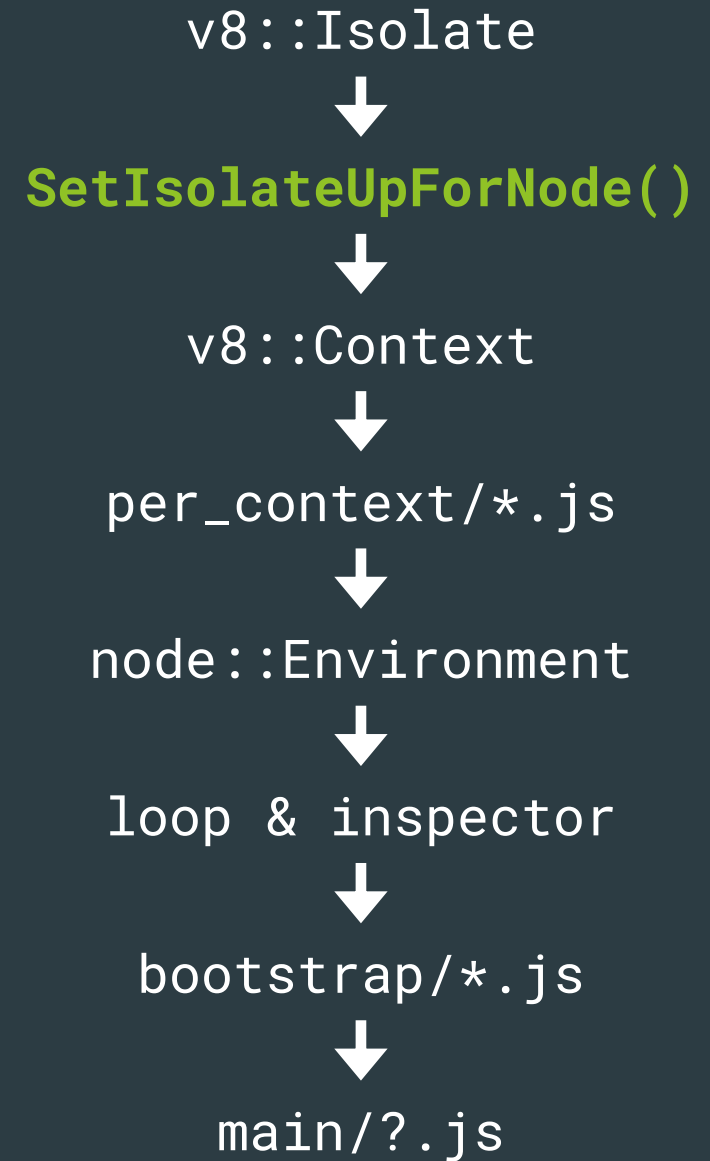
```
bash-5.0$ time luajit -e "local x = 1"
real    0m0.005s
user    0m0.002s
sys     0m0.002s
bash-5.0$ time perl -e 1
real    0m0.007s
user    0m0.003s
sys     0m0.003s
bash-5.0$ time /Users/joyee/.jsvu/v8 -e 1
real    0m0.029s
user    0m0.008s
sys     0m0.017s
bash-5.0$ time ./node -e 1
real    0m0.050s
user    0m0.032s
sys     0m0.012s
```

d8 with default snapshot

node master without snapshot

# Snapshot Integration

Original



# Snapshot Integration

**Snapshotted (2019.05)**

---

v8::Isolate



Context::FromSnapshot()



**SetIsolateUpForNode()**

Re-install callbacks



node::Environment



loop & inspector



bootstrap/\*.js



main/? .js

# Snapshot Integration

## Goal

---

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

`v8::Isolate`  
↓  
`Context::FromSnapshot()`  
↓  
`Environment:: FromSnapshot()`  
↓

**`SetIsolateUpForNode()`**  
Re-install callbacks

↓  
`loop & inspector`

↓  
`main/? .js`

# Snapshot Integration

## Refactoring

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

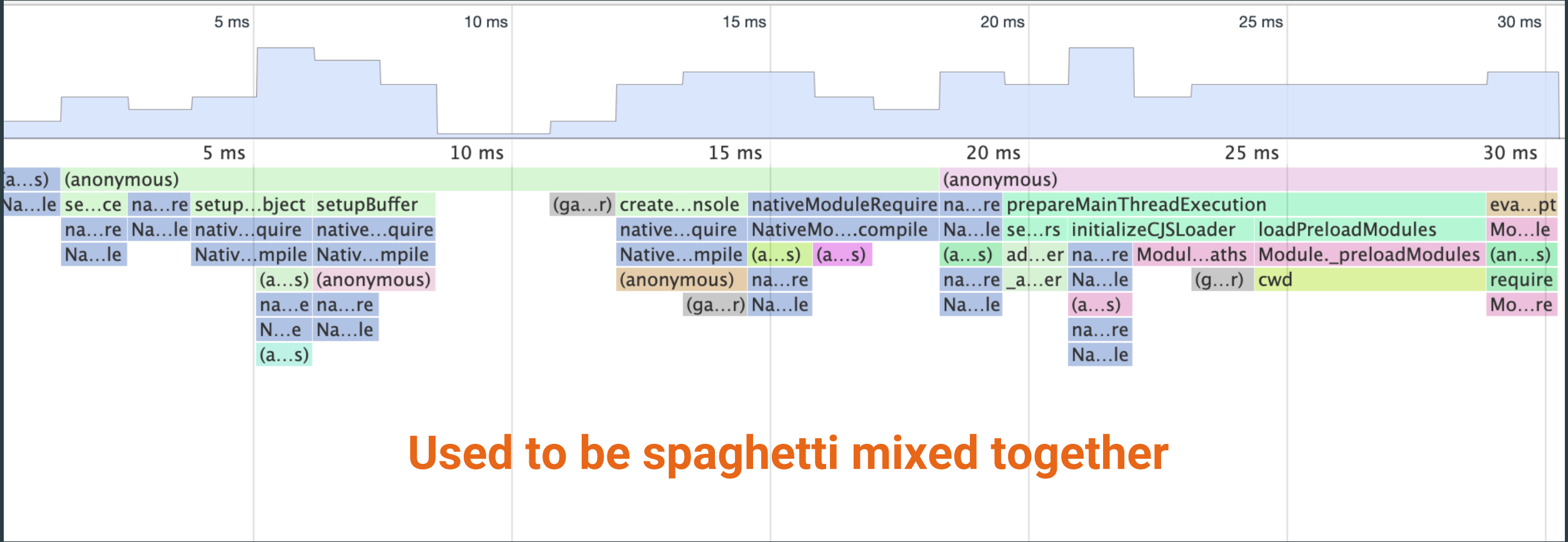
```
v8::Isolate  
↓  
Context::FromSnapshot()  
↓  
Environment:: FromSnapshot()  
↓
```

```
SetIsolateUpForNode()  
Re-install callbacks
```

```
↓  
loop & inspector
```

```
↓  
main/? .js
```

# 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

v8::Isolate



Context::FromSnapshot()



Environment::FromSnapshot()



**SetIsolateUpForNode()**

Re-install callbacks



loop & inspector



main/? .js



# Snapshot Integration

## Refactoring

---

Reorganize so that we can reinstall C++ states

`v8::Isolate`  
↓  
`Context::FromSnapshot()`  
↓  
`Environment:: FromSnapshot()`  
↓

**`SetIsolateUpForNode()`**  
Re-install callbacks

↓  
`loop & inspector`

↓  
`main/? .js`

# 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**