Node.js Startup Performance

Joyee Cheung, Igalia

About me

- Joyee Cheung (Cantonese) / Qiuyi Zhang (Mandarin)
- Compilers @ Igalia
- Node.js TSC & V8 Committer
- Champion of the startup performance initiative in Node.js
  - https://github.com/nodejs/TSC/blob/master/Strategic-Initiatives.md
- @joyeecheung on GitHub & Twitter

Slides of this talk:

https://github.com/joyeecheung/talks/blob/master/nodeconf_remote_202011/node-startup-performance.pdf

The journey of Node.js startup performance

Node Startup Improvement #27196

Closed **suresh-srinivas** opened this issue on Apr 12, 2019 · 18 comments

suresh-srinivas commented on Apr 12, 2019 Contributor

Is your feature request related to a problem? Please describe.
Node takes around 60ms to start on a modern Skylake Server, processing ~230M instructions to execute a mostly empty function (see below). From looking at the startup timeline view (using trace-events). During this time, it loads 56 JavaScript core library files and compiles them. Much of the startup time is due to that.

```
"events.js"
"internal/trace_events_async_hooks.js"
"async_hooks.js"
"internal/errors.js"
"internal/validators.js"
"internal/async_hooks.js"
"internal/safe_globals.js"
```

The journey of Node.js startup performance

suresh-srinivas commented on Apr 18, 2019 Contributor Author

Not everyday you get to see a 2X improvement for `node -e {}` Nice work **@joyeecheung** and other collaborators.

	nodejs v10.13.0	nodejs master	nodejs v10.13.0/master
cycles	149,778,245	77,046,107	1.9
instructions	210,776,025	91,695,996	2.3

We will analyze this some more. Let us know where we can help.

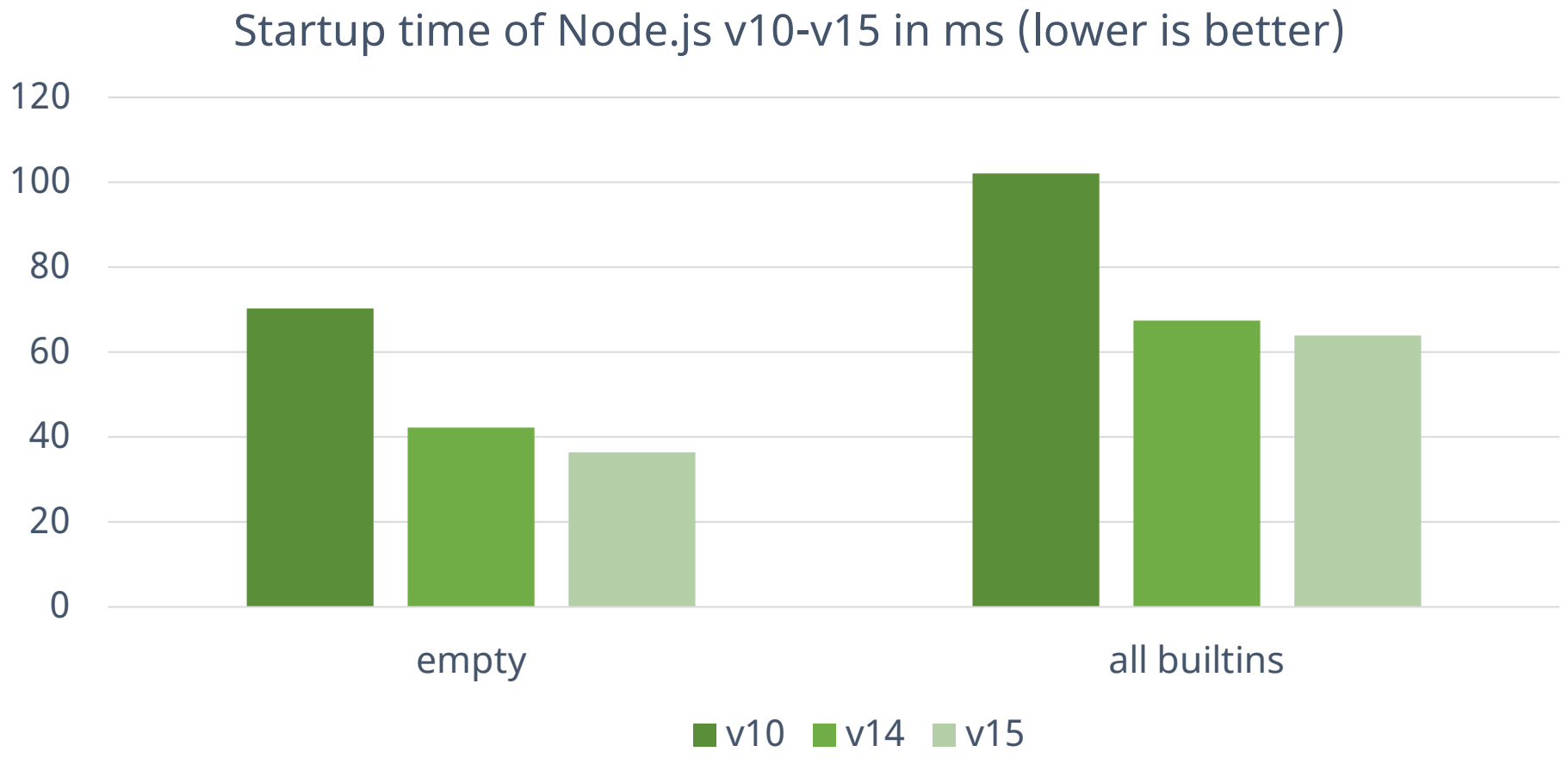
❤️ 1

suresh-srinivas commented on Apr 18, 2019 Contributor Author

The new trace for empty shows it is down to 21ms. Very impressive. **@hashseed** there is a 1ms of GC Scavenge around 15 ms. Any suggestions?

There is also a bit of sparse script, parse and compile bits of green. Is that loading the cached code? Would be nice to batch it

The journey of Node.js startup performance

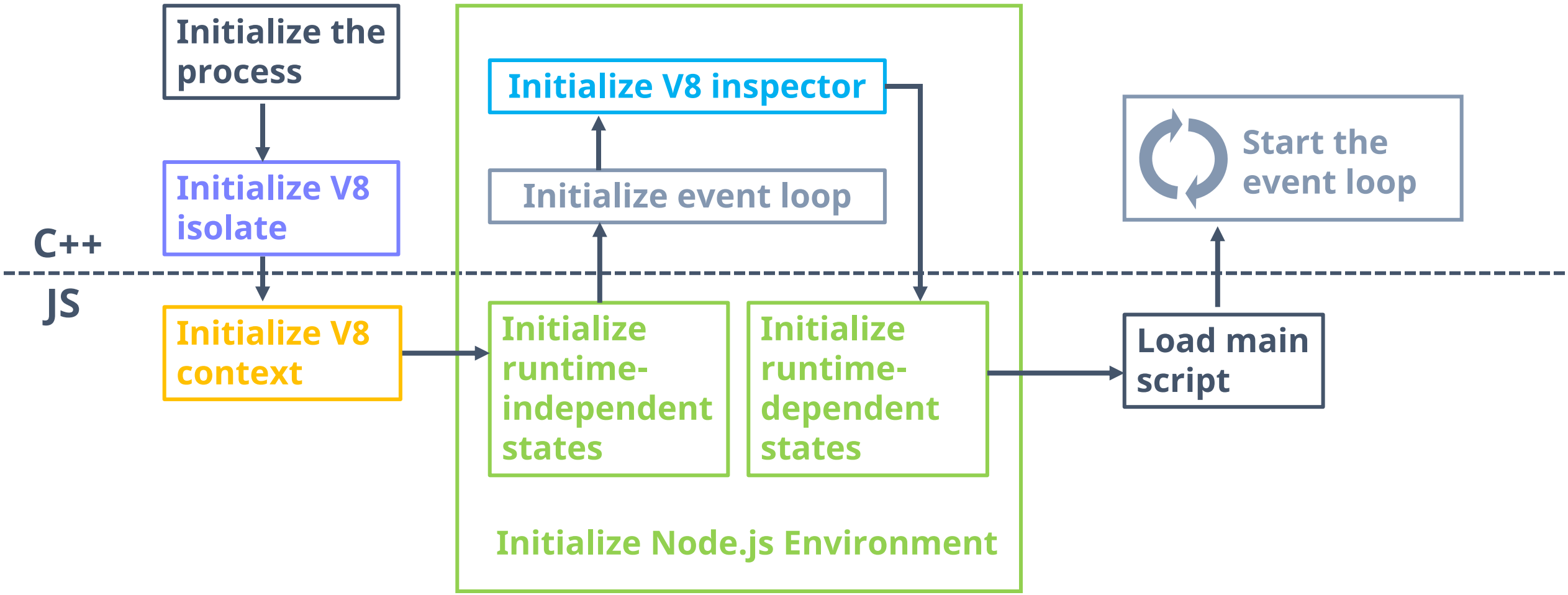


* Unlike in the previous issue, these benchmarks were run on a MacBook with Coffee Lake 2.9 GHz Core-i9 CPUs

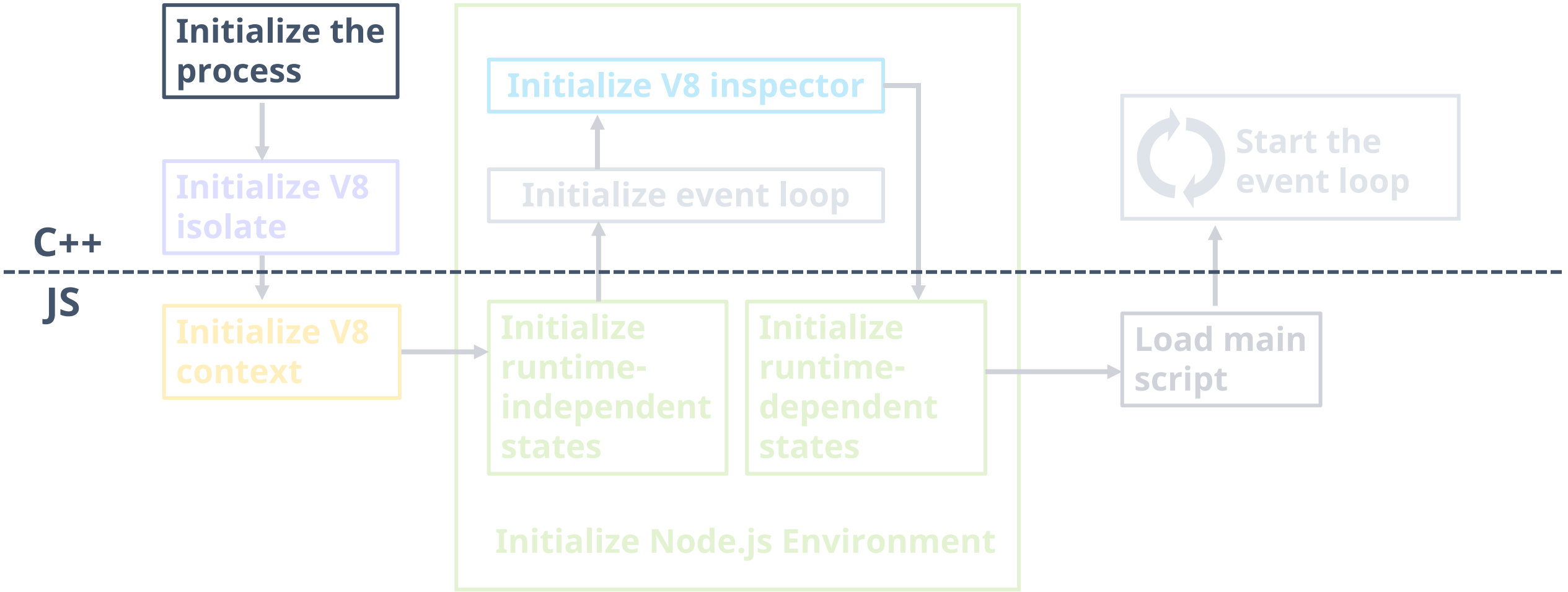
The journey of Node.js startup performance

1. Refactoring to avoid unnecessary work
2. Implement code caching
3. Integrating V8 startup snapshot

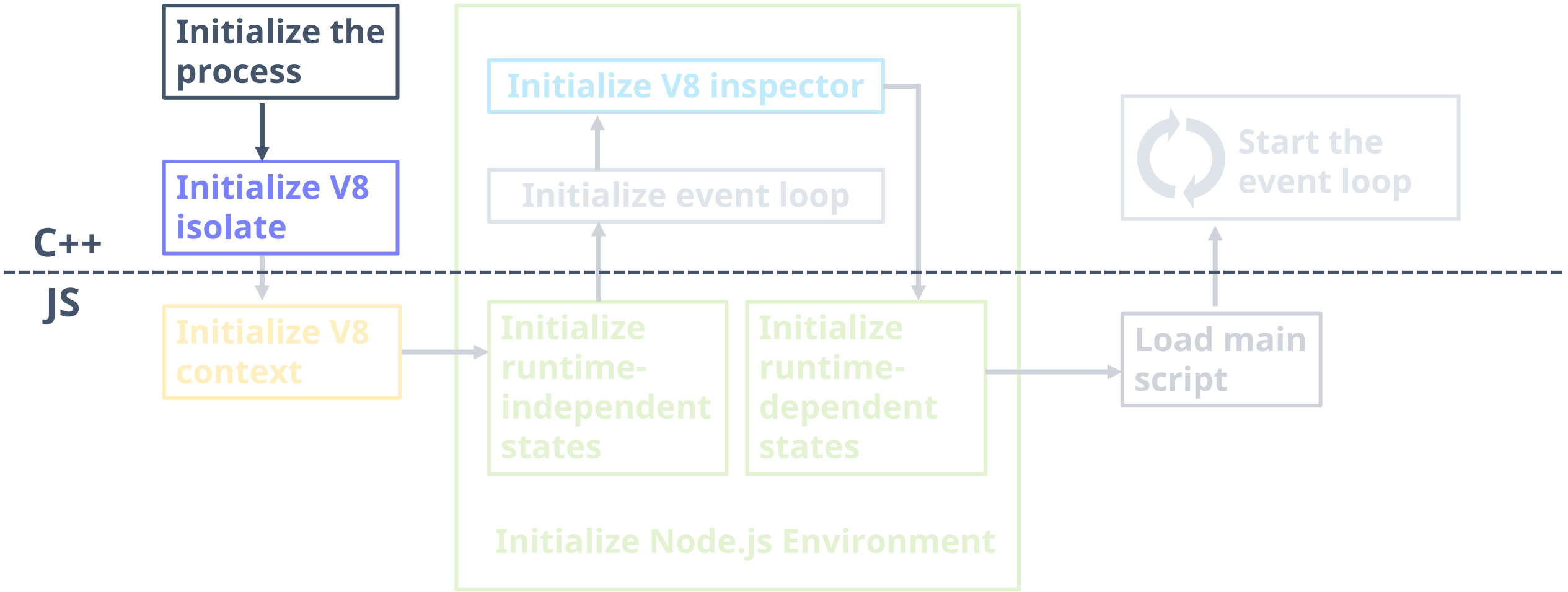
Overview of the Node.js bootstrap



Setting up the process



Setting up the V8 isolate

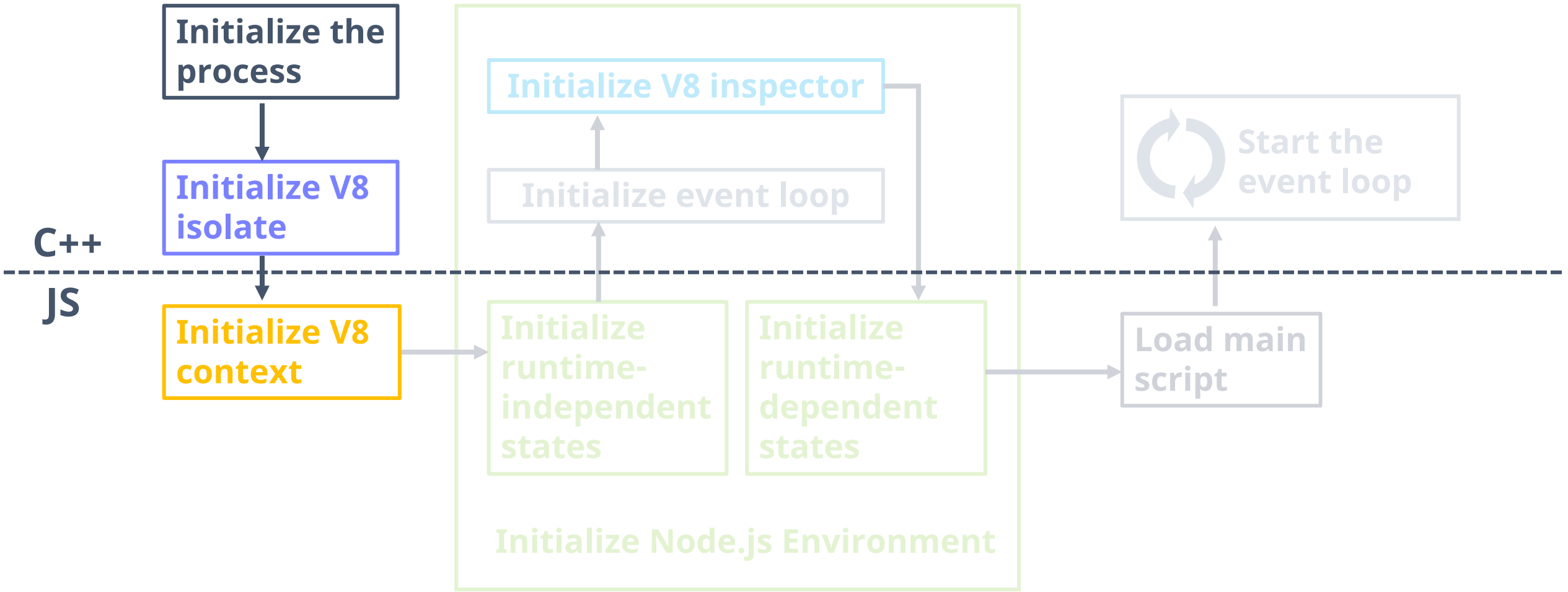


Setting up the V8 isolate

What's a V8 isolate?

- `v8::Isolate` is the instance of the v8 JavaScript engine
- Encapsulates the JS heap, microtask queue, pending exceptions…
- In Node.js, the main instance and each worker gets their own V8 isolates

Setting up the V8 context



Setting up the V8 context

What's a V8 context?

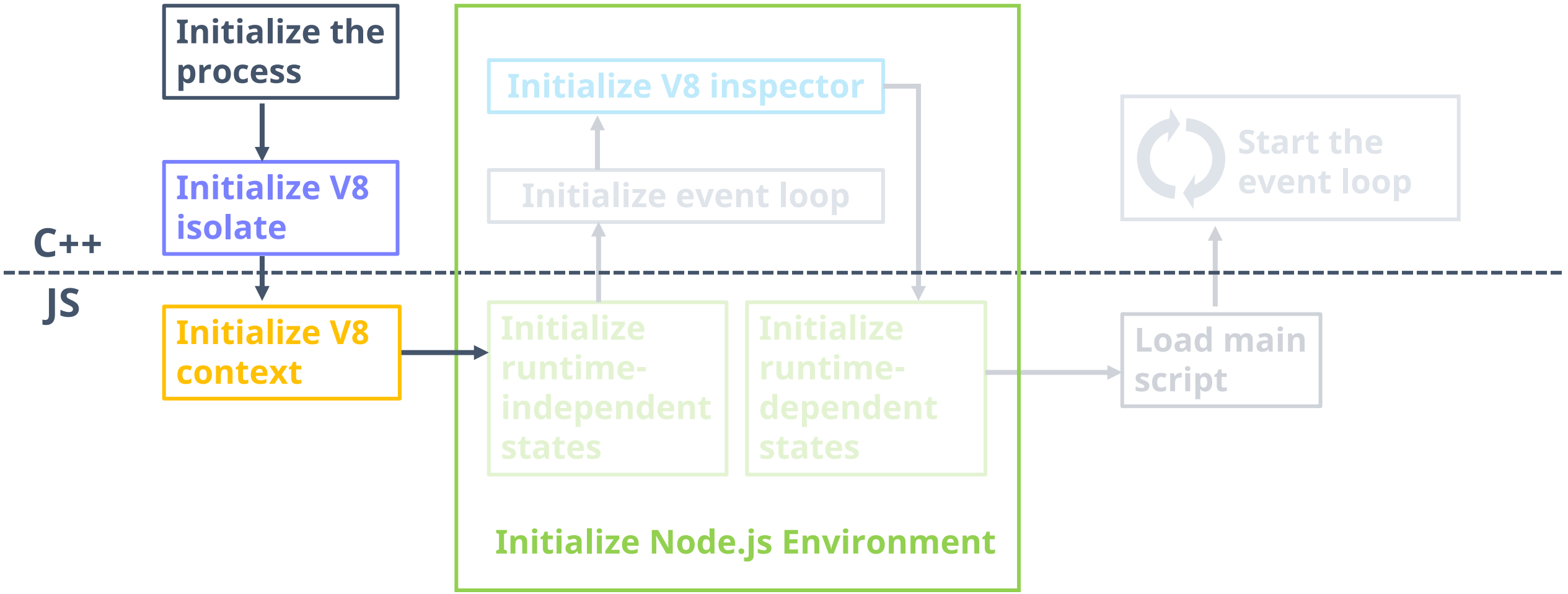
- A sandboxed execution context
- Encapsulates JavaScript builtins (primordials) e.g. `globalThis`, `Array`, `Object`...
- What's inside the returned result of `vm.createContext()`

Setting up the V8 context

What's a V8 context?

- In Node.js, userland JavaScript is executed in the main V8 context by default, sharing the same context as the built-ins of Node.js.
- Node.js copies the original JS built-ins at the beginning of the bootstrap for the built-in modules to use.

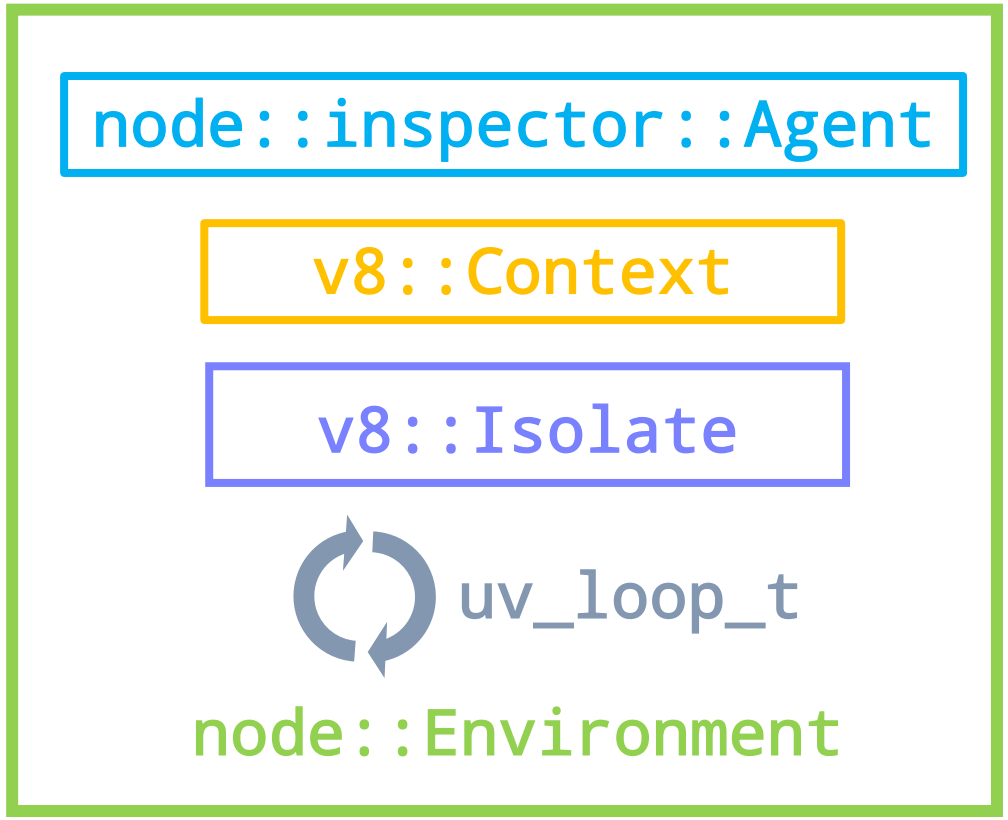
Setting up the Environment



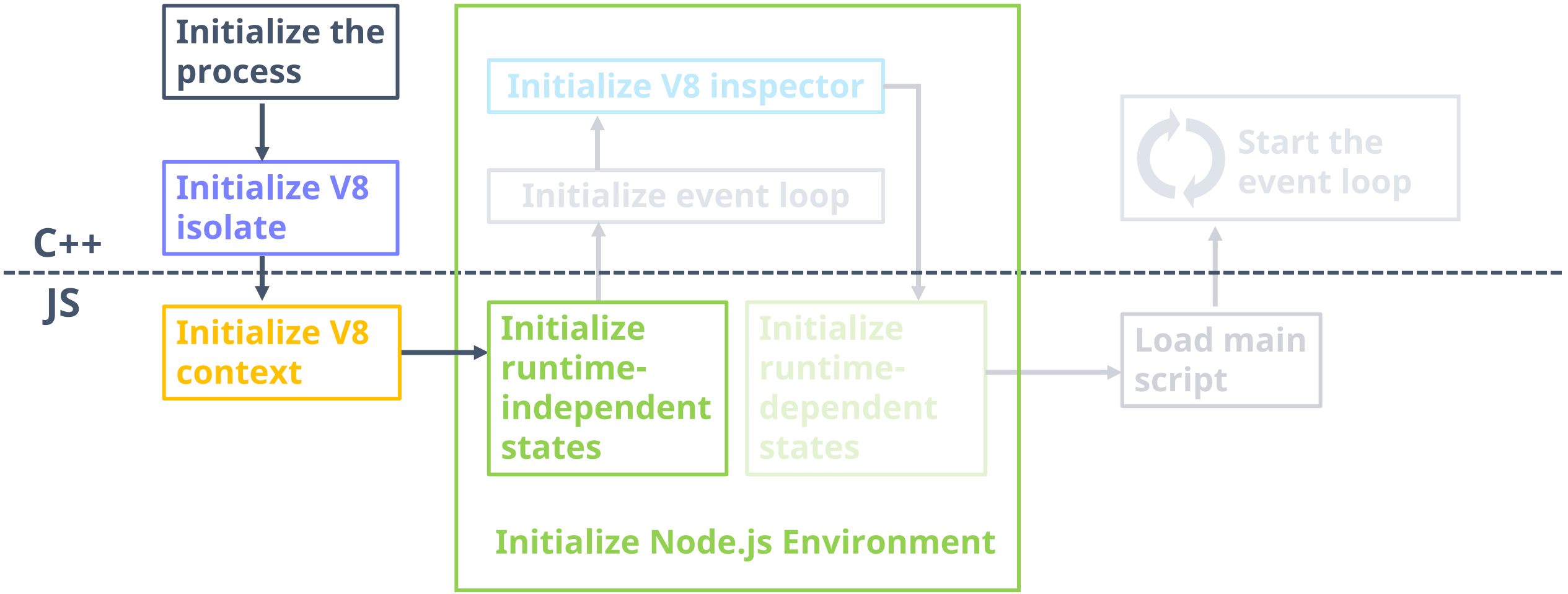
Setting up the Environment

What's a Node.js Environment?

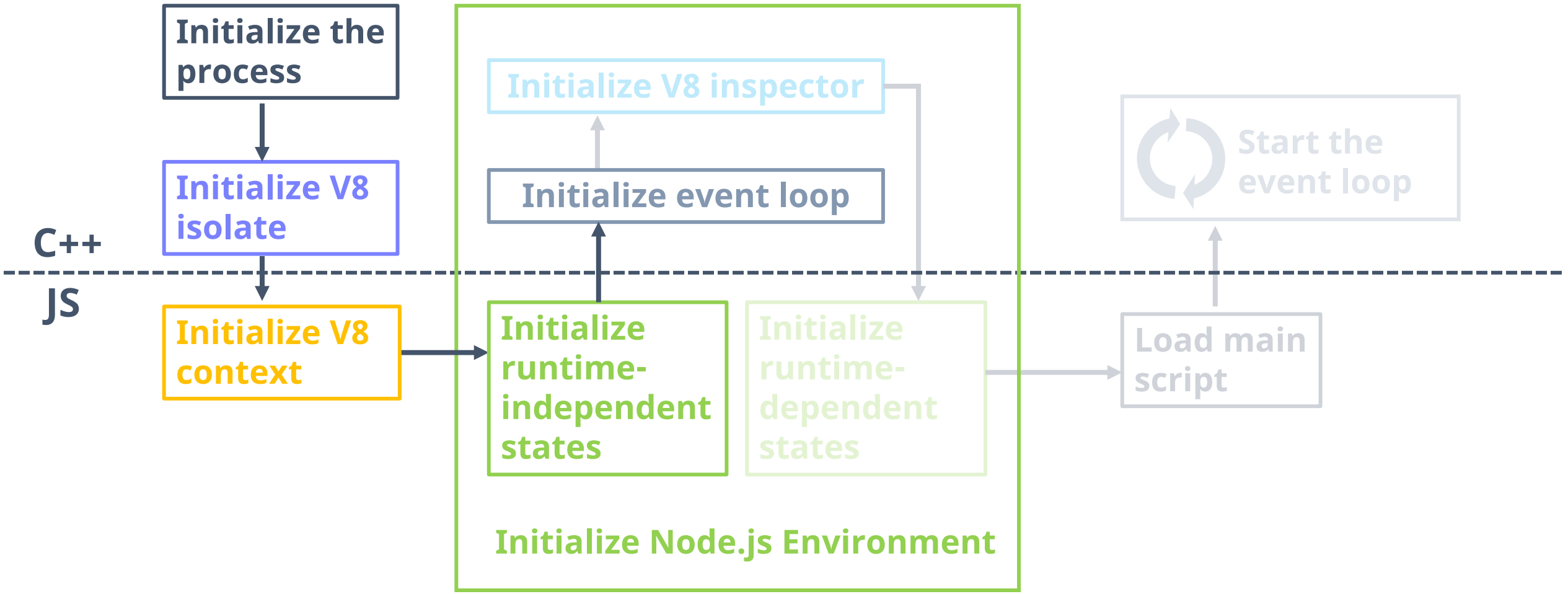
- Encapsulation of the Node.js instance
- Associated with
  - One V8 inspector agent (for JS debugging)
  - One main V8 context
  - One V8 isolate
  - One libuv event loop



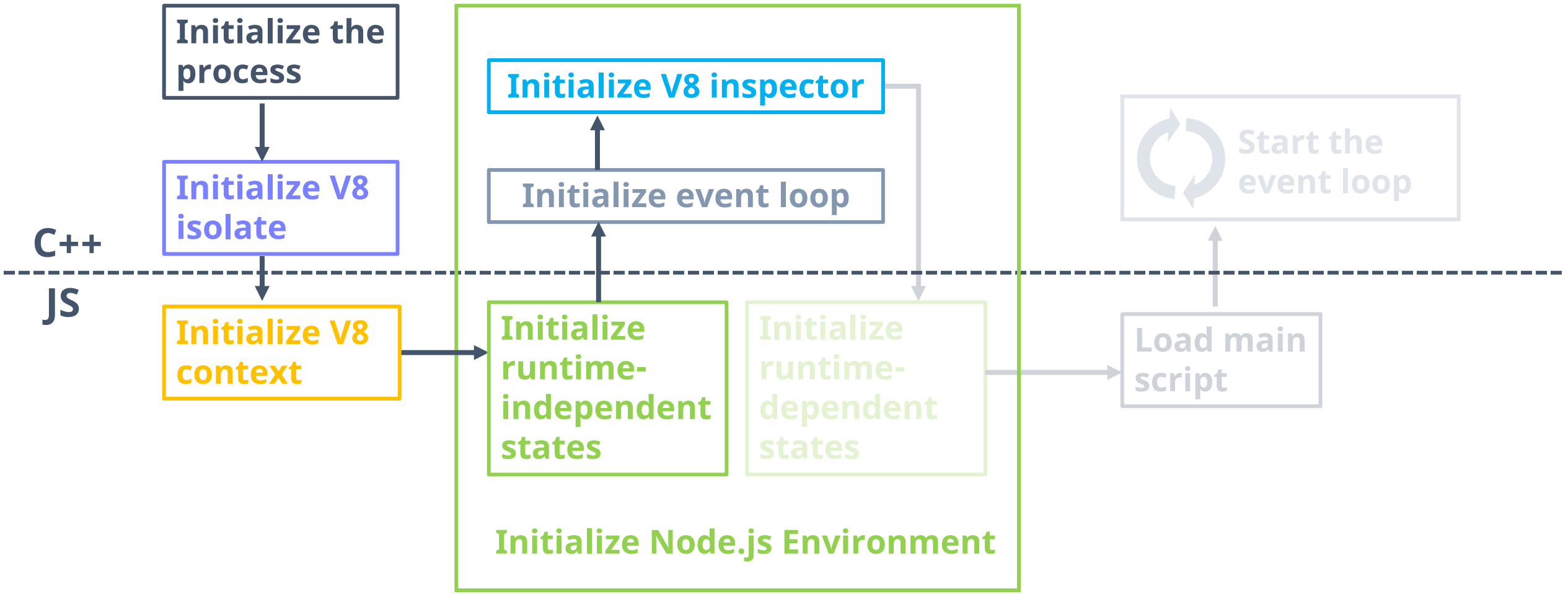
Setting up the Environment



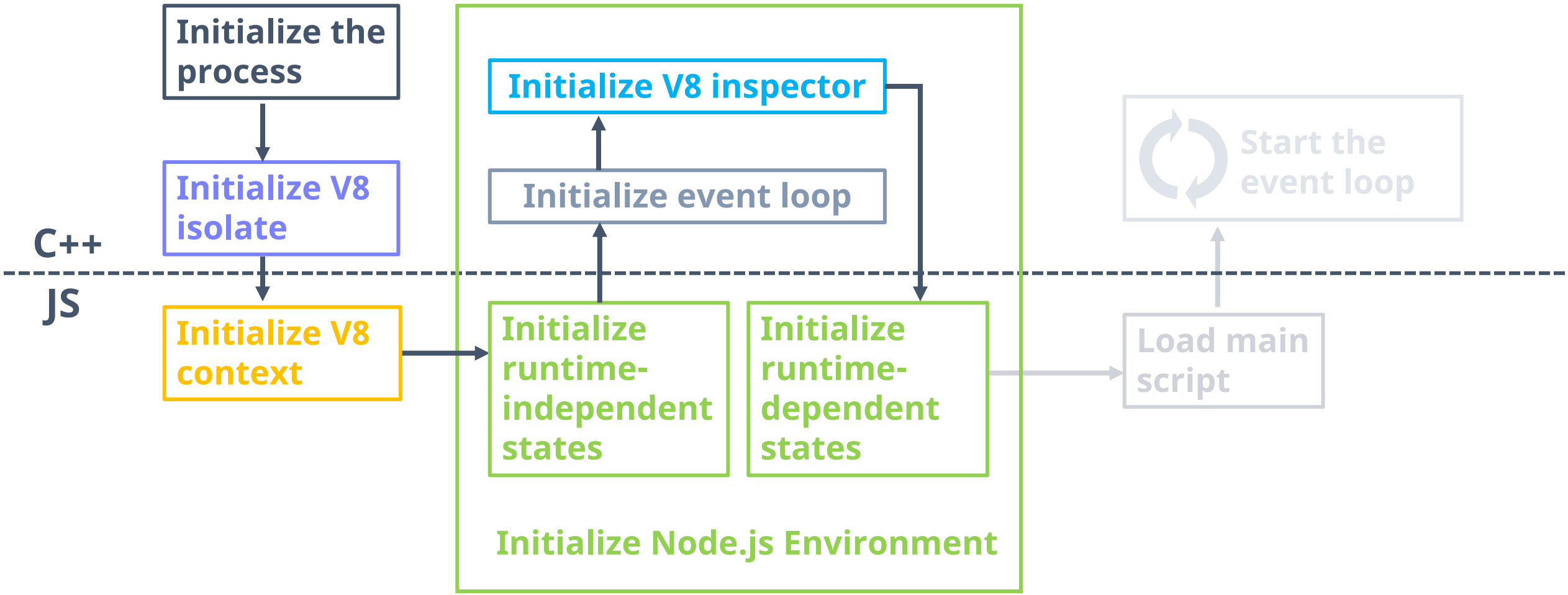
Setting up the Environment



Setting up the Environment



Setting up the Environment

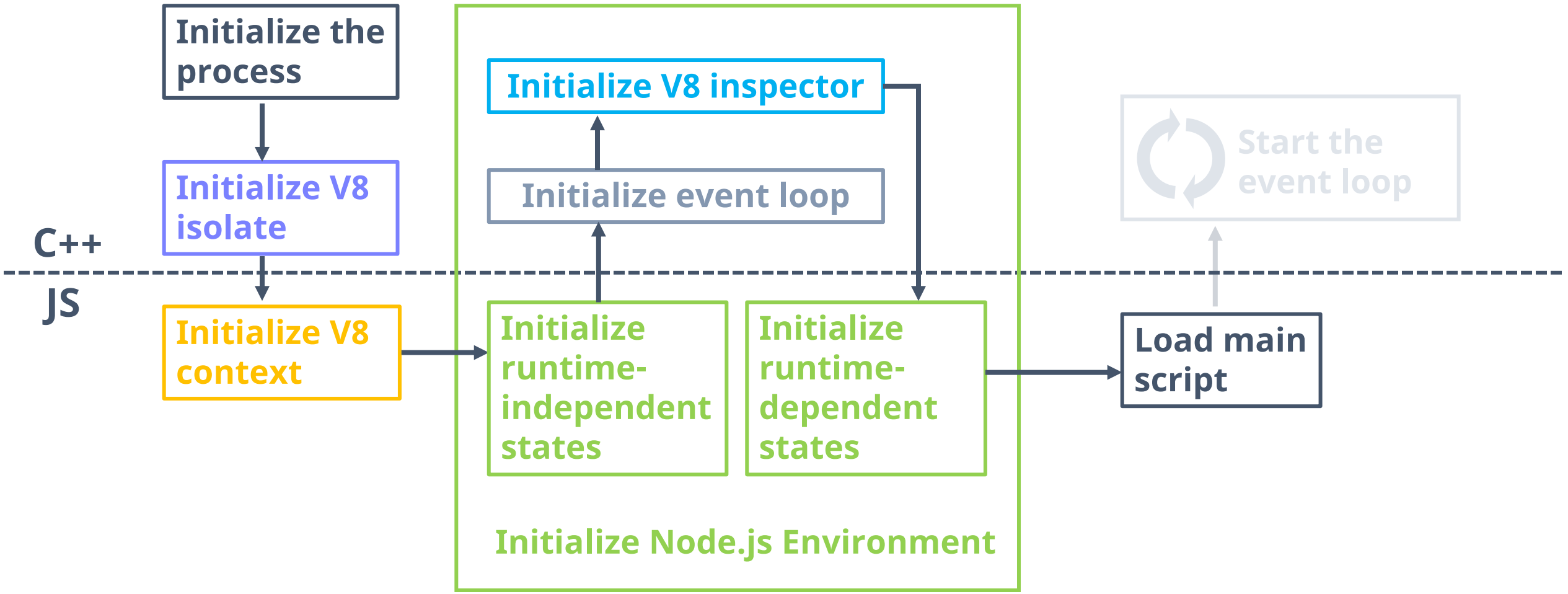


Refactoring for snapshot integration

- Introduce a new step *pre-execution* to handle runtime configurations
  - **CLI flags**: e.g. `--no-warnings` , `--experimental-policy`, `--experimental-report`
  - **Environment variables**: e.g. `NODE_DEBUG`, `NODE_V8_COVERAGE`

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

Start execution



Start execution: from CLI

Create and initialize Environment

↓ Select a main script

Load run_main_module.js

↓ Detect module type

Read and compile ${cwd}/index.js as CJS

↓

Start event loop

```
$ node index.js
```

Start execution: Worker

Create and initialize Environment

↓ Select a main script

Load worker_thread.js

↓ Setup message port and start listening

Start event loop

↓

Compile and run the script sent from the port

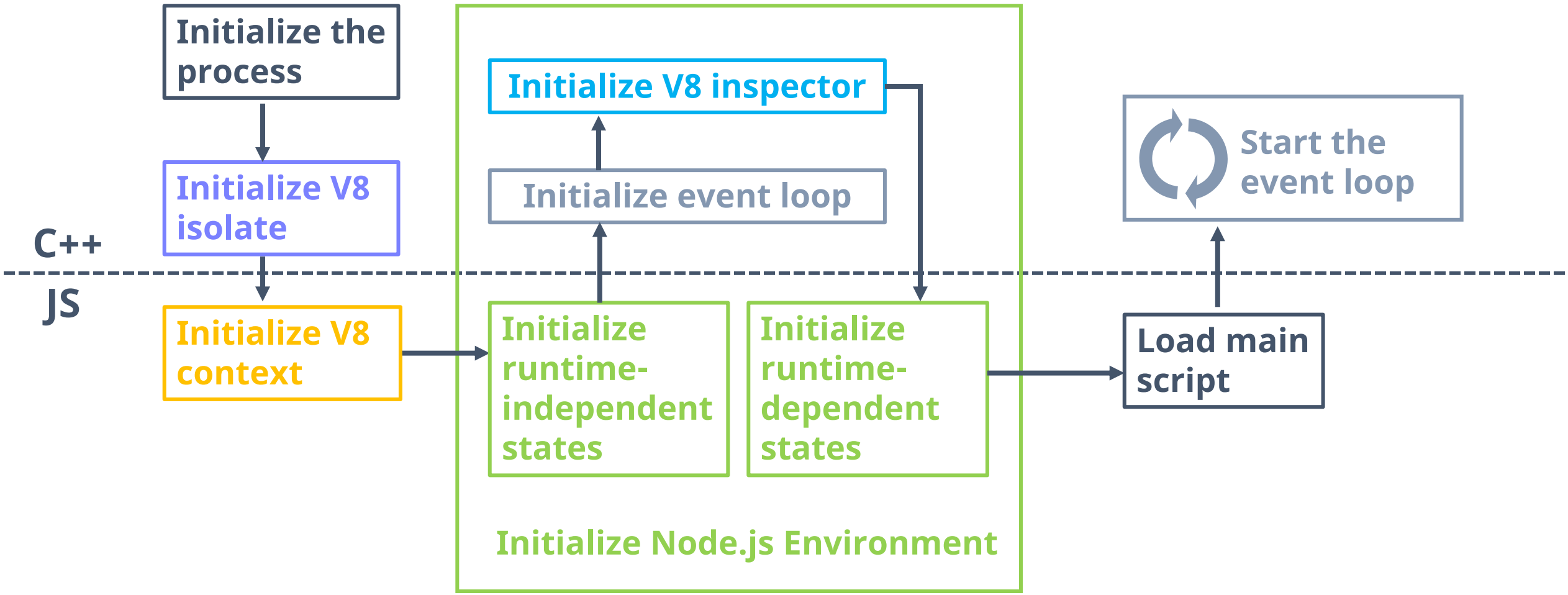
From user code on the main thread

```
const { Worker } =
  require('worker_threads');
const script =
`console.log('hello')`;
new worker_threads
  .Worker(script, { eval: true });
```

From the worker_thread.js on the worker thread

```
evalScript('[worker eval]', script);
```

Start execution



The journey of Node.js startup performance

1. Refactoring to avoid unnecessary work
2. Implement code caching
3. Integrating V8 startup snapshot

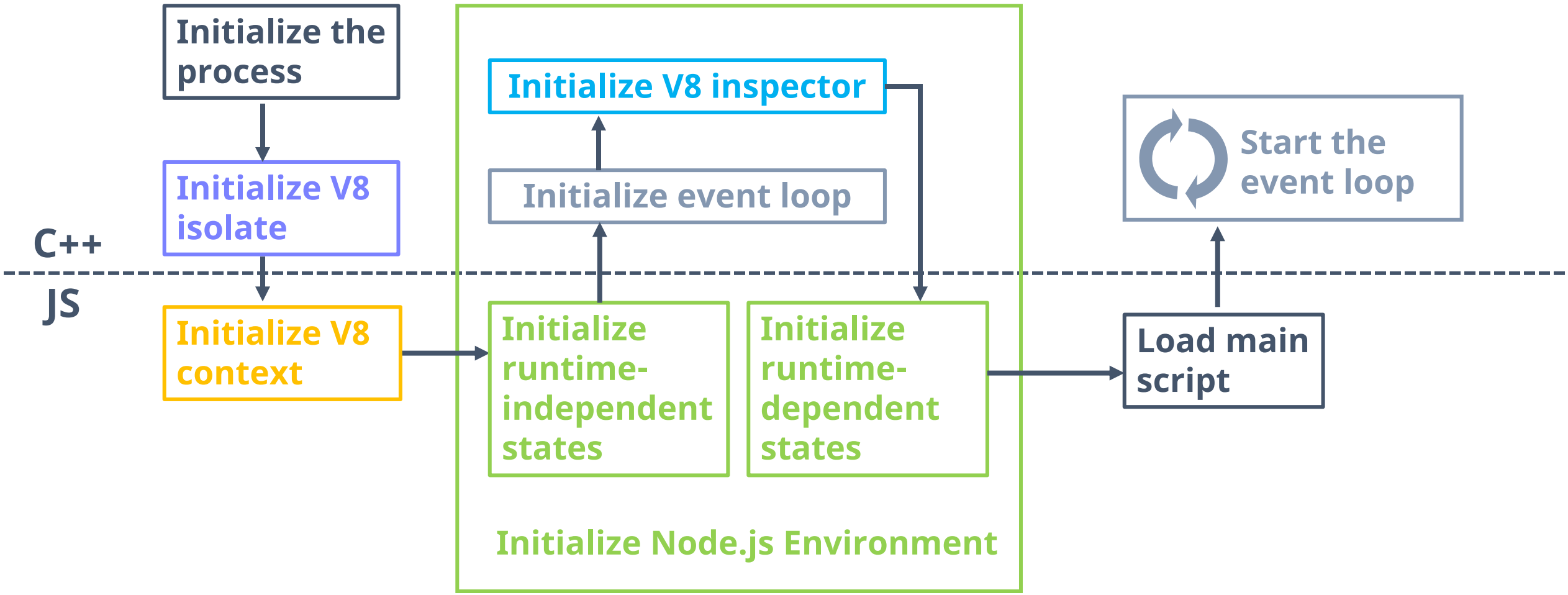
Refactoring

- Lazy-load builtins that are not always used
  - A lot of builtin modules depend on each other
  - Caveat: we’d spend more time loading them on demand later
  - Can be reverted when startup snapshot covers these modules

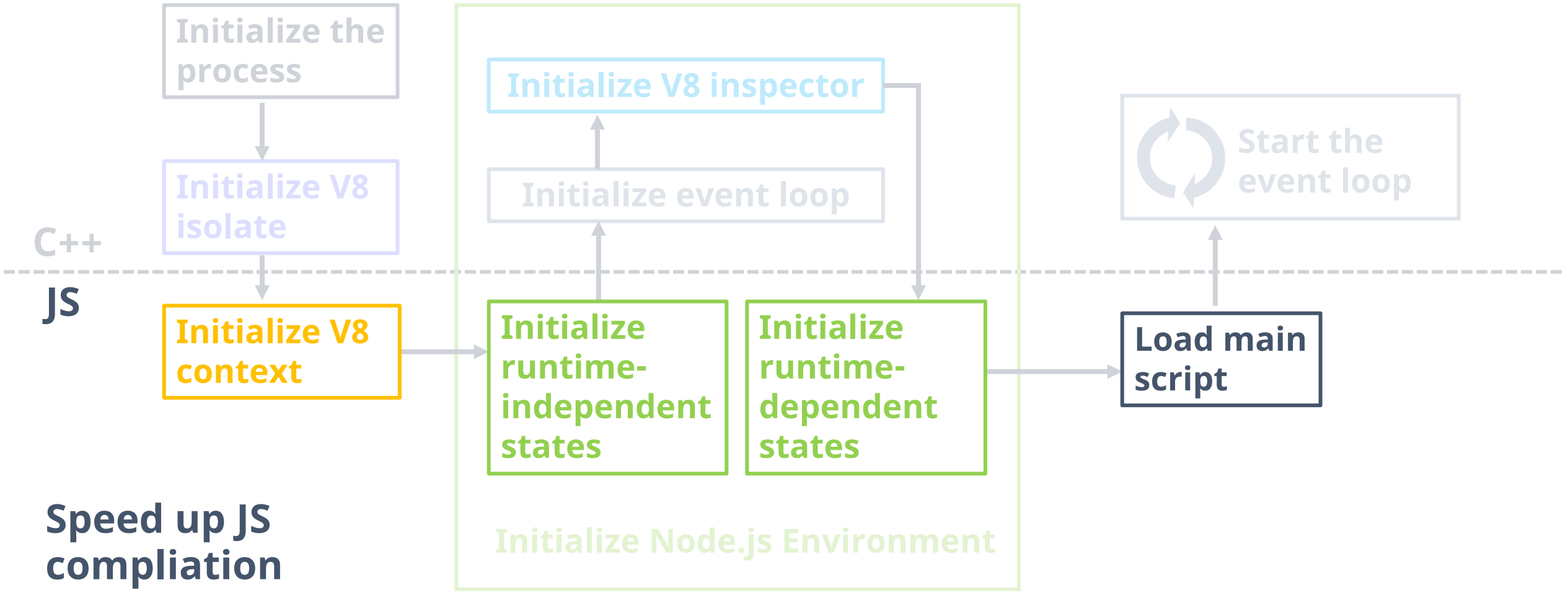
Refactoring

- Lazy-load builtins that are not always used
  - A lot of builtin modules depend on each other
  - Caveat: we'd spend more time loading them on demand later
  - Can be reverted when startup snapshot covers these modules
- Avoid unnecessary work
  - e.g. console creation
  - Startup snapshot doesn't help since it depends on runtime states

Refactoring

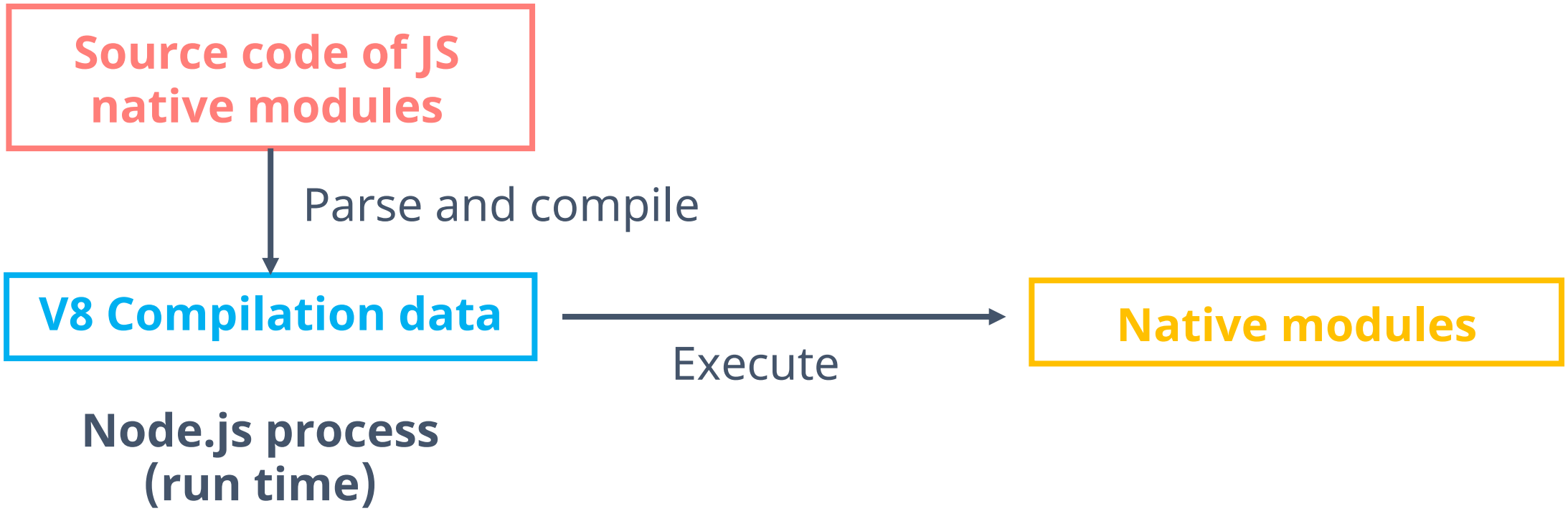


Code caching

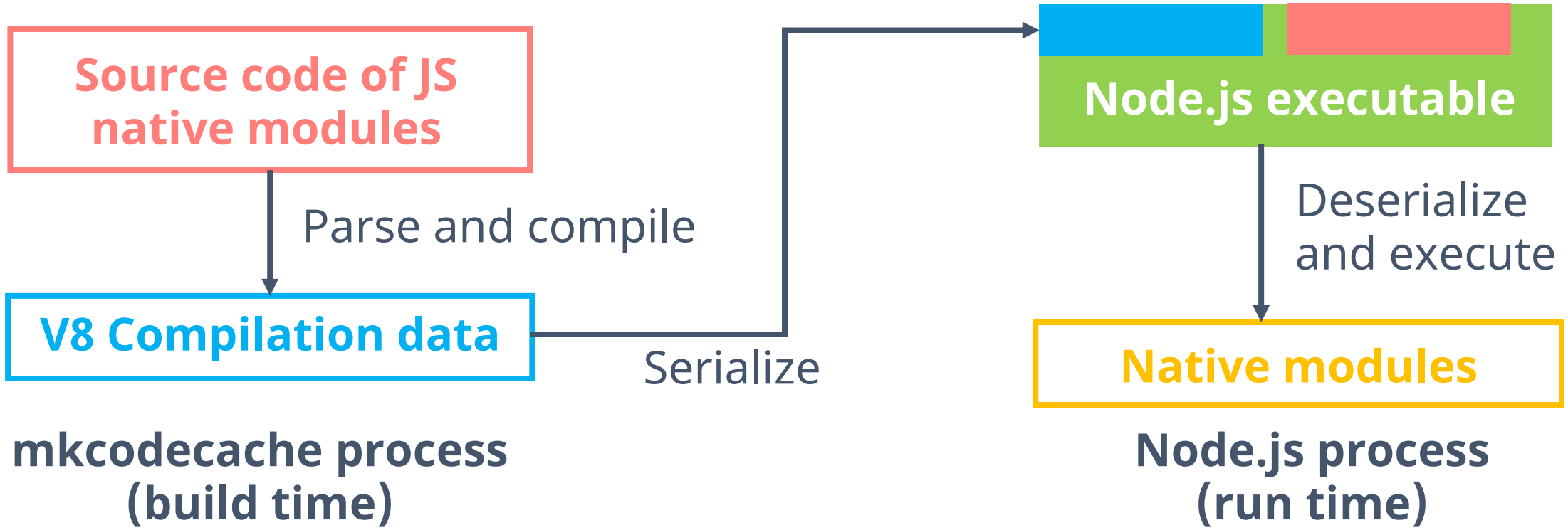


Speed up JS compliation

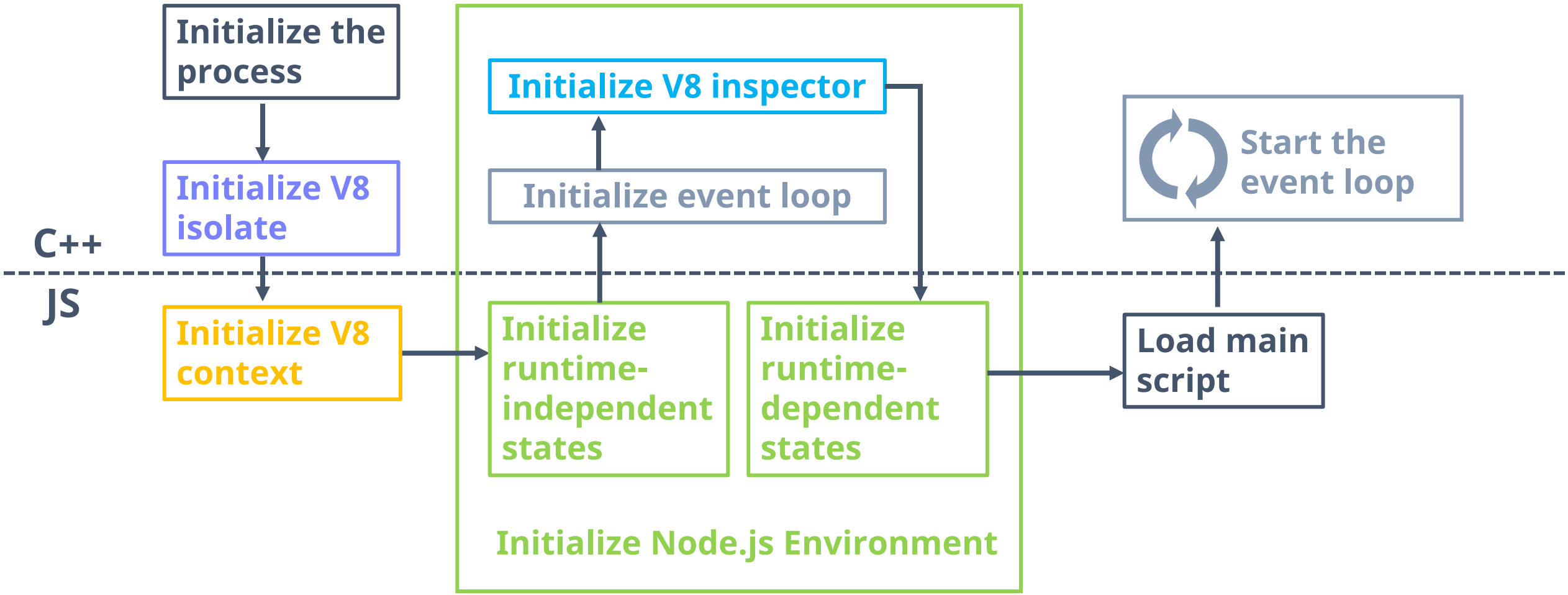
V8 code cache: before



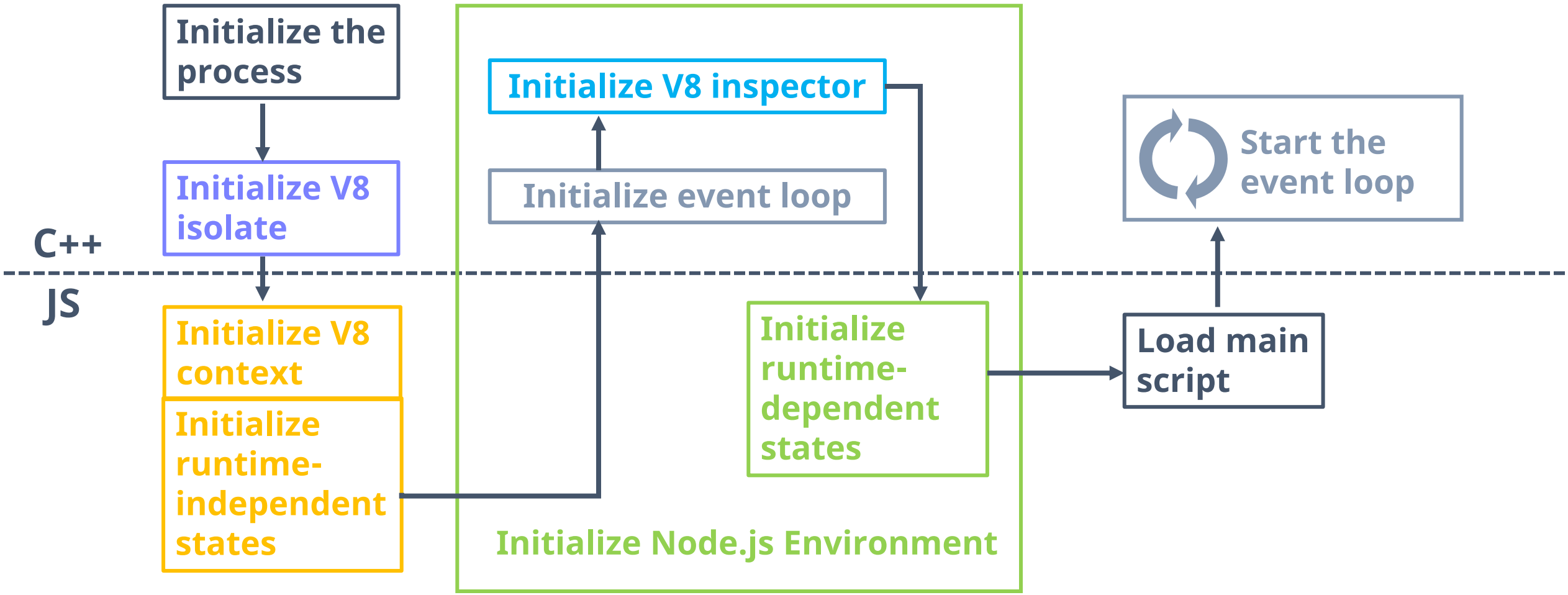
V8 code cache: after



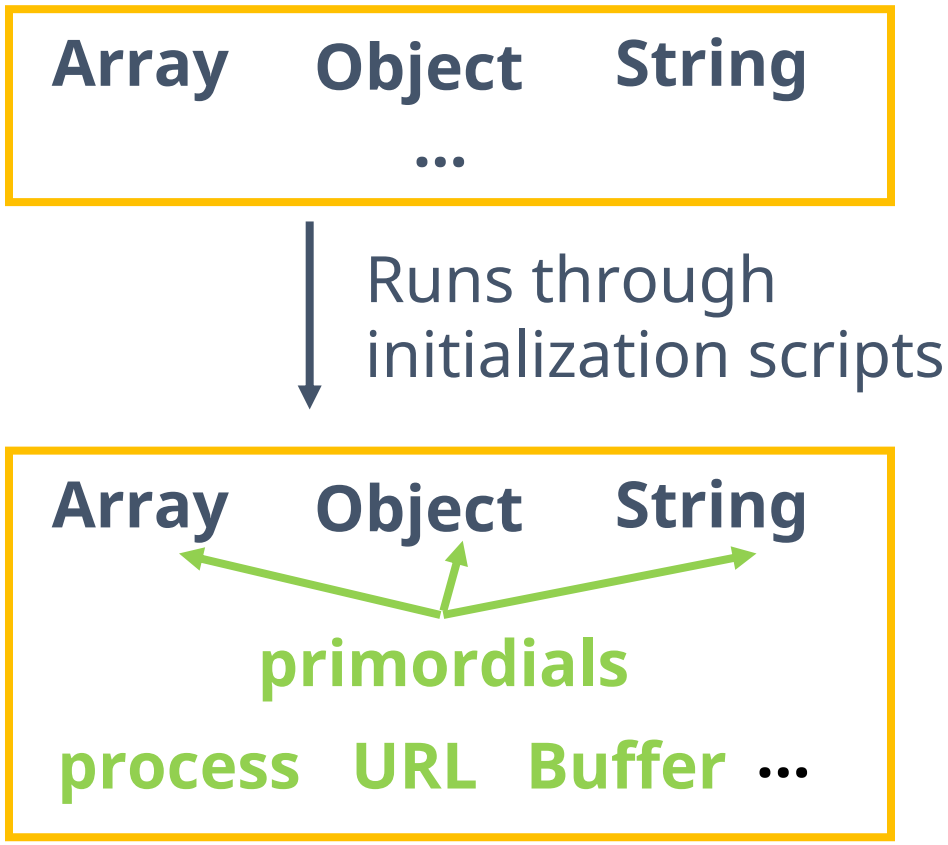
Refactoring for snapshot integration



Refactoring for snapshot integration

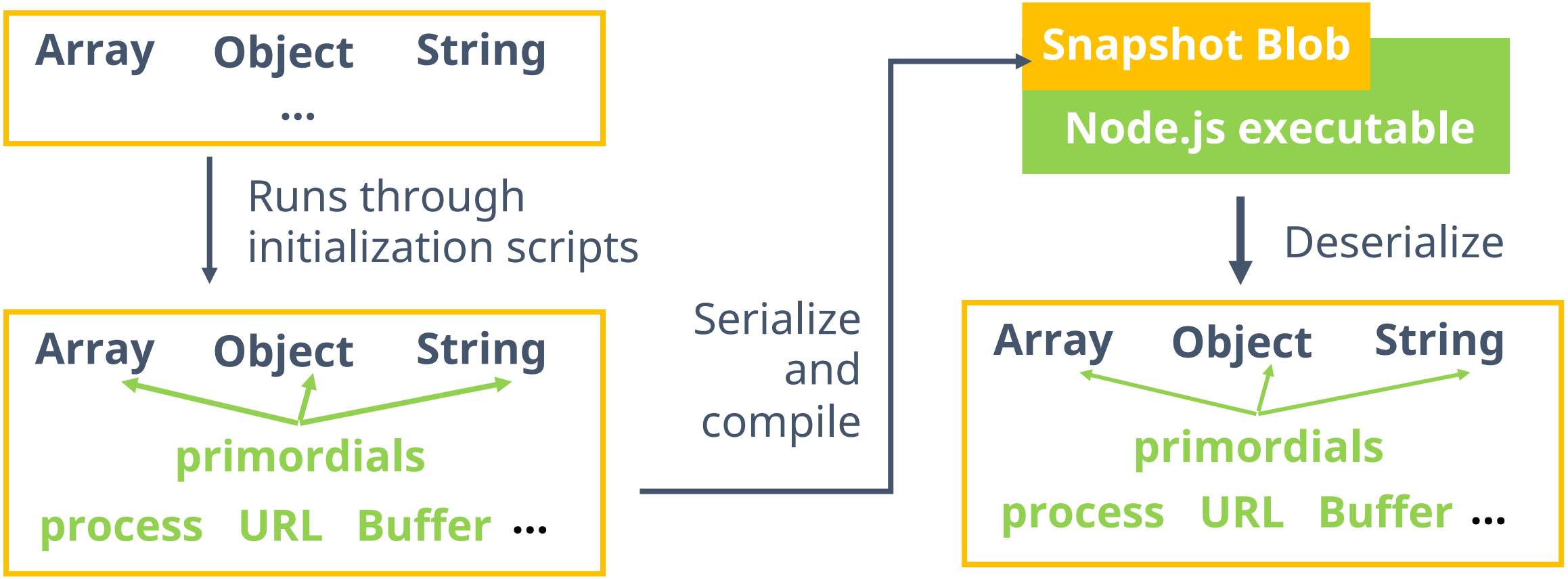


V8 startup snapshot: before



Node.js process

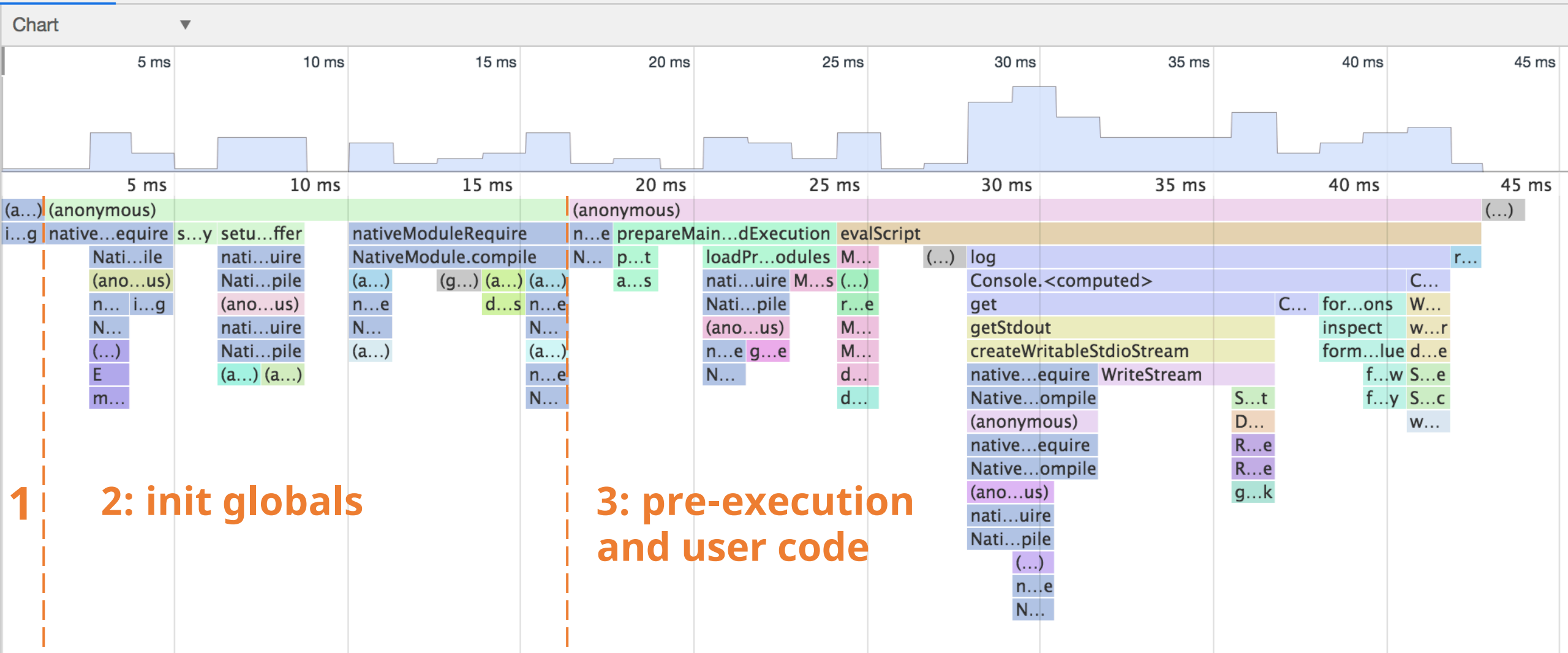
V8 startup snapshot: after



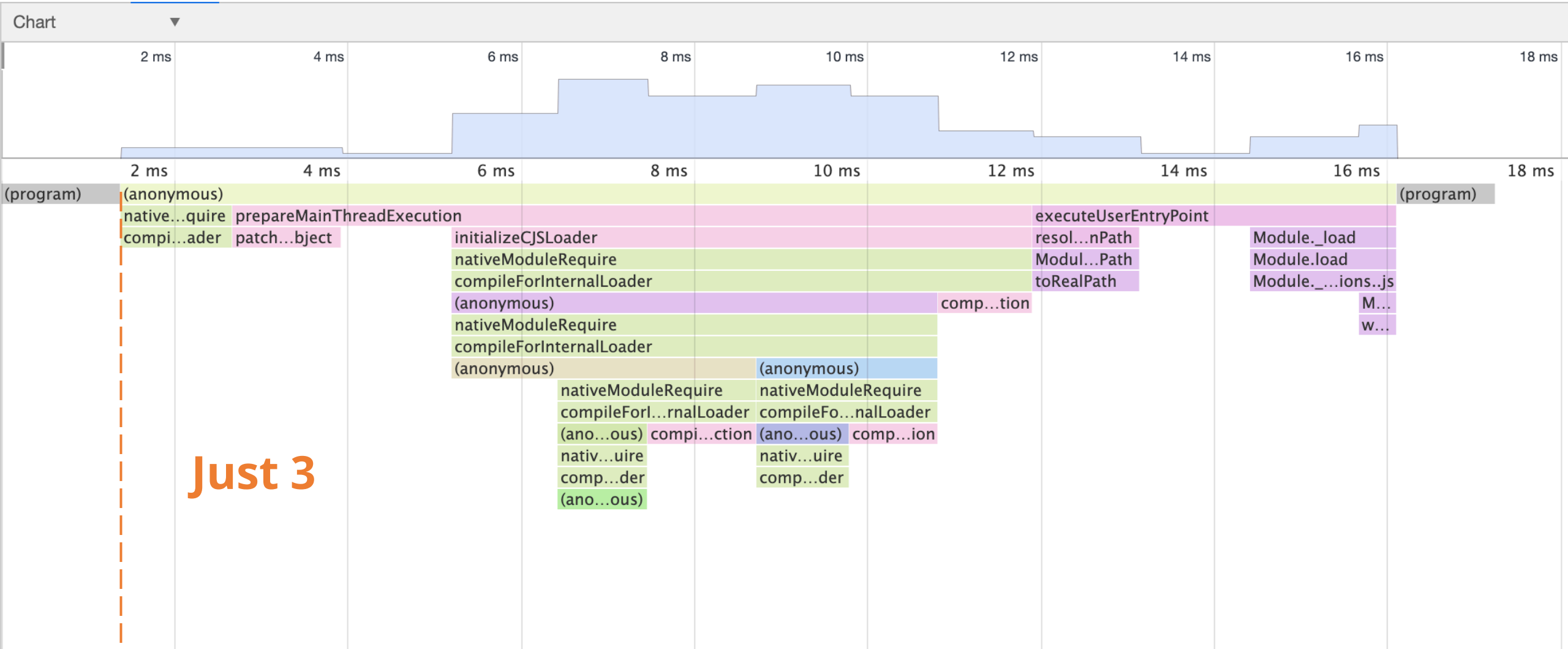
node_mksnapshot process (build time)

Node.js process (run time)

V8 startup snapshot: before



V8 startup snapshot: after



Ongoing work

Supporting more language features in the V8 snapshot

- JSMap and JSSet rehashing
  - Hash flooding vulnerability caused by fixed hash seed from the snapshot: https://v8.dev/blog/hash-flooding
  - Solution: generate new hash seed and rehash all the objects

Ongoing work

Supporting more language features in the V8 snapshot

- JSMap and JSSet rehashing
  - Hash flooding vulnerability caused by fixed hash seed from the snapshot: https://v8.dev/blog/hash-flooding
  - Solution: generate new hash seed and rehash all the objects
  - Rehashing was not implemented for Map and Sets
  - Implemented Map and Set rehashing in V8 so that they can be used in the startup snapshot of Node.js

Ongoing work

Supporting more language features in the V8 snapshot

- Class field initializers
  - Once used by the EventTarget: https://www.nearform.com/blog/node-js-and-the-struggles-of-being-an-eventtarget/

Ongoing work

Supporting more language features in the V8 snapshot

- Class field initializers
  - Once used by the EventTarget: https://www.nearform.com/blog/node-js-and-the-struggles-of-being-an-eventtarget/
  - Work in progress: reparse the initializers after deserializing them from the V8 snapshot

Future work

Userland snapshotting

- Take a snapshot of an application and write it to disk
- Load it from the command line or build it into an executable
- CLI or APIs under `worker_thread`/`child_process`

Tracking issue: https://github.com/nodejs/node/issues/35711

Thank you!

💗 to Igalia & Bloomberg for supporting my work

Special thanks to @addaleax