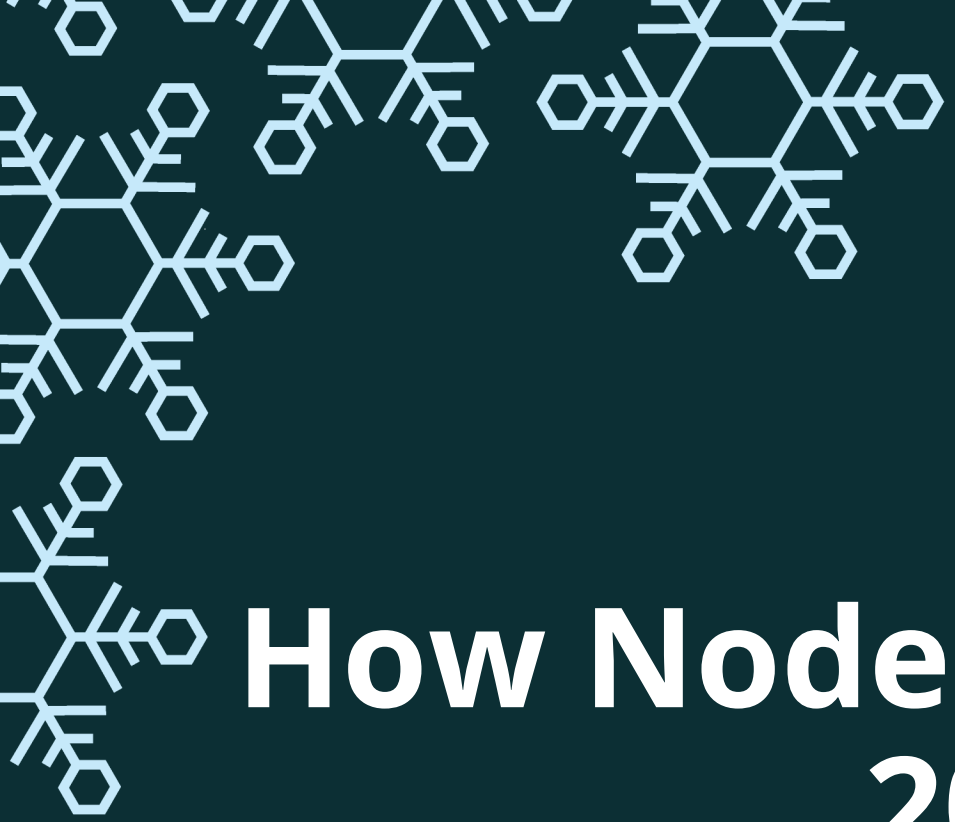
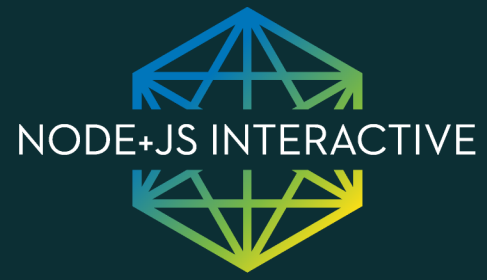




December 11–12, 2019
Montreal, Canada

How Node.js Bootstraps Itself: 2019 Edition

Joyee Cheung, Igalia

NODE+JS INTERACTIVE

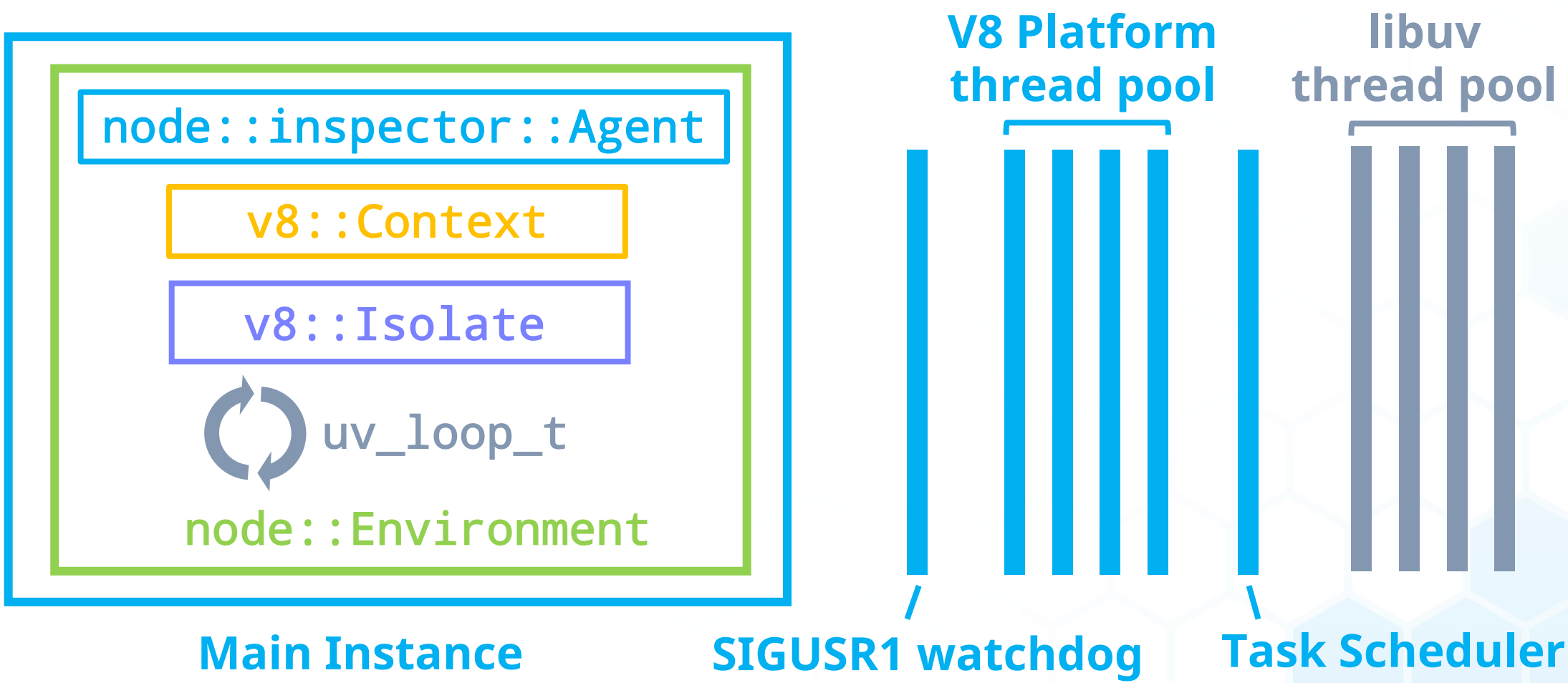
December 11–12, 2019
Montreal, Canada

About me

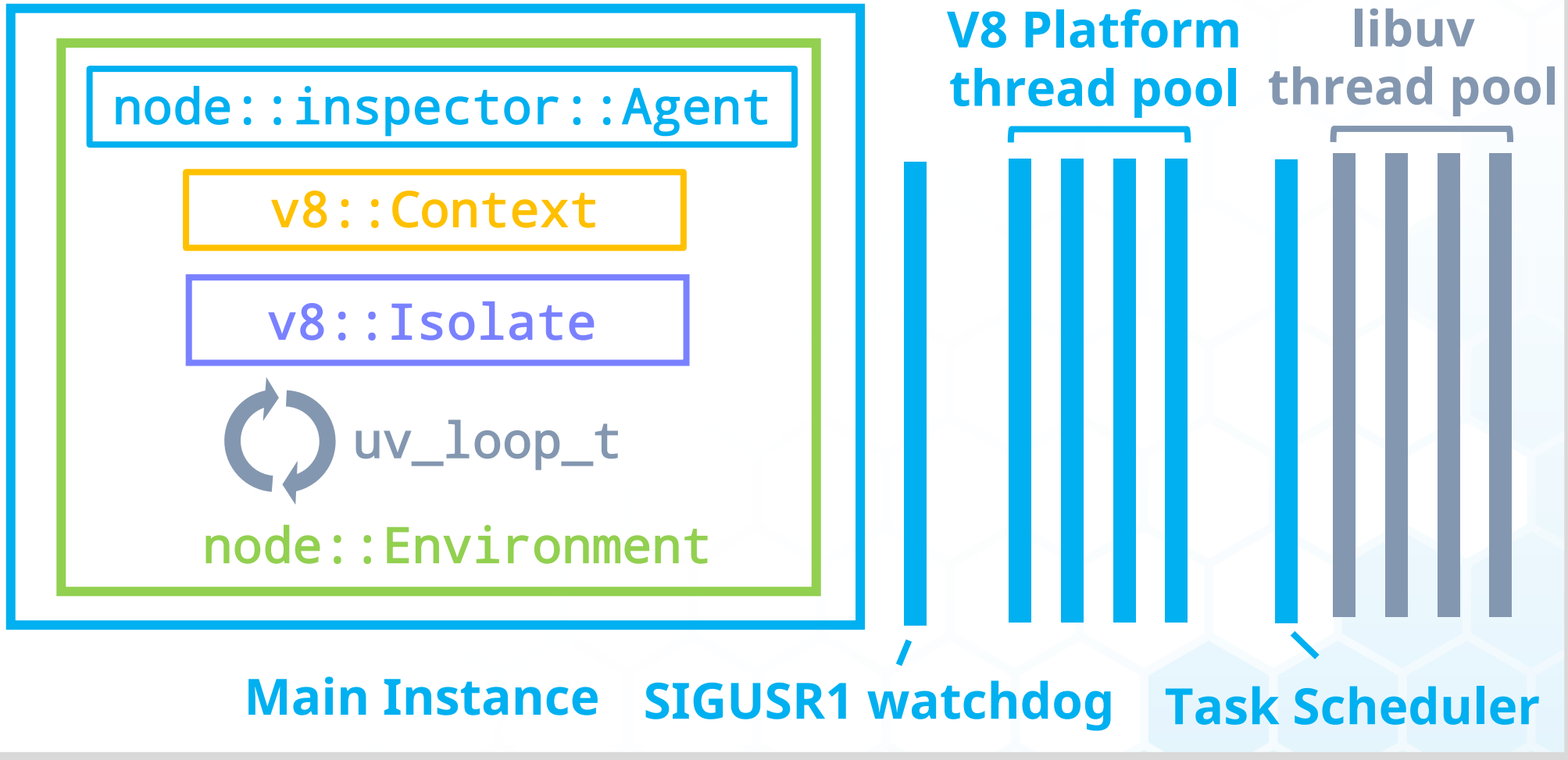
- Joyee Cheung (Cantonese) / Qiuyi Zhang (Mandarin)
- Compilers @ Igalia
- Node.js TSC & V8 Committer
- Champion of the startup performance initiative in Node.js
  - https://github.com/nodejs/TSC/blob/master/Strategic-Initiatives.md
- @joyeecheung on GitHub & Twitter

Slides of this talk:
https://github.com/joyeecheung/talks/blob/master/node_js_interactive_2019/how-node-js-bootstraps-itself-2019-edition.pdf

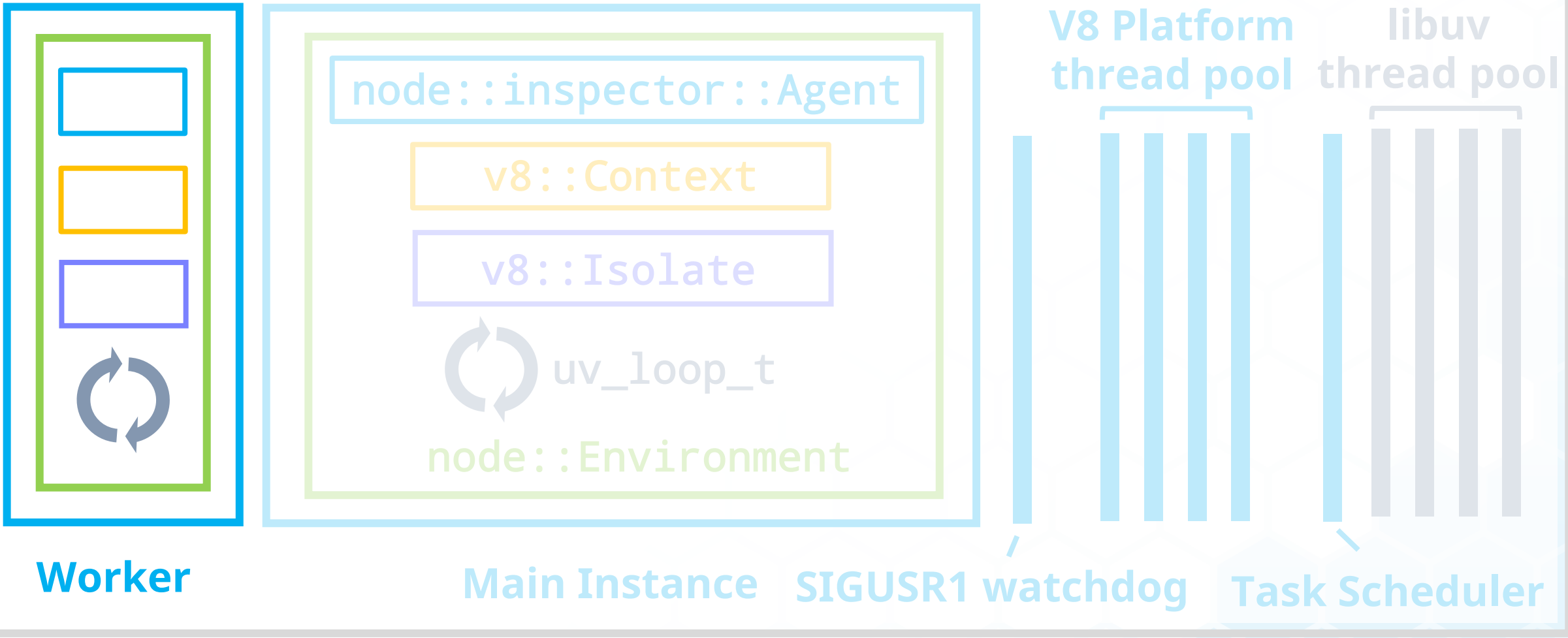
Overview of a Node.js process



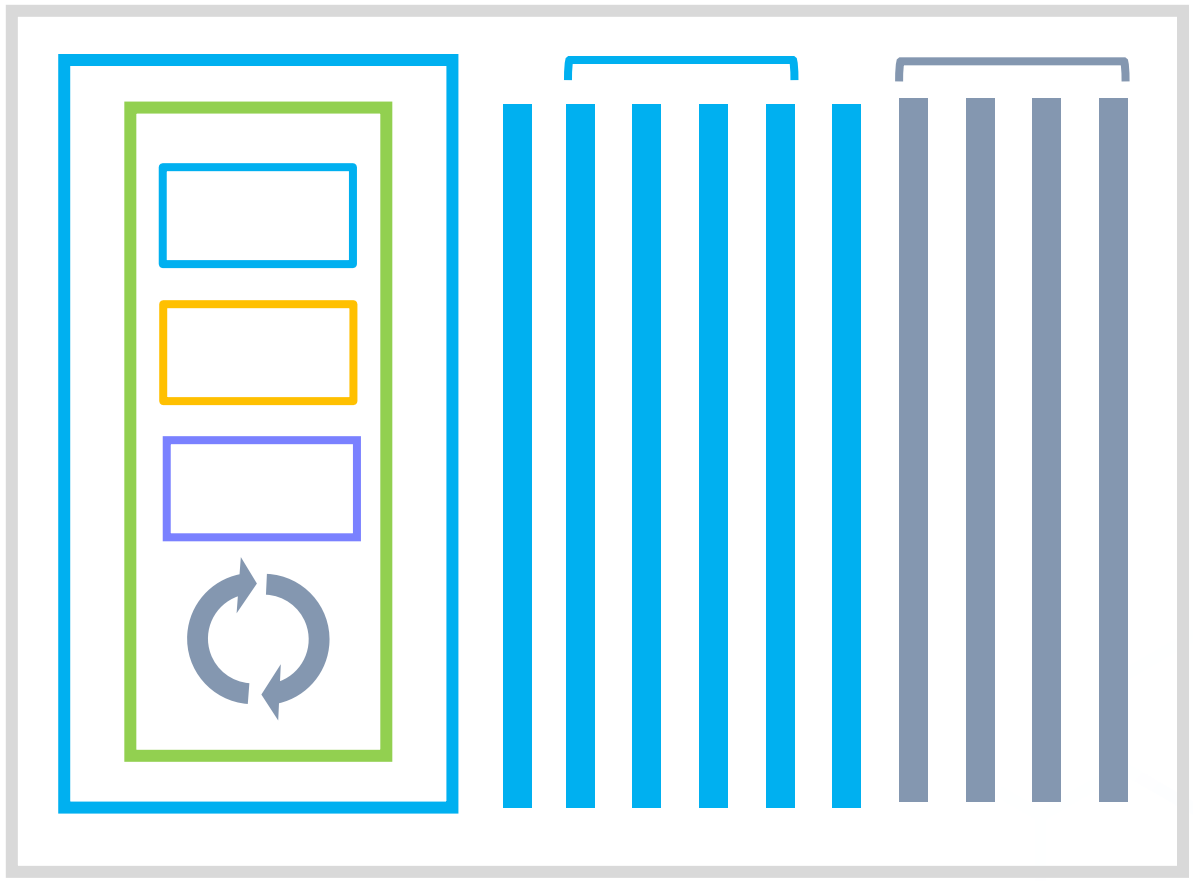
Creating a Worker



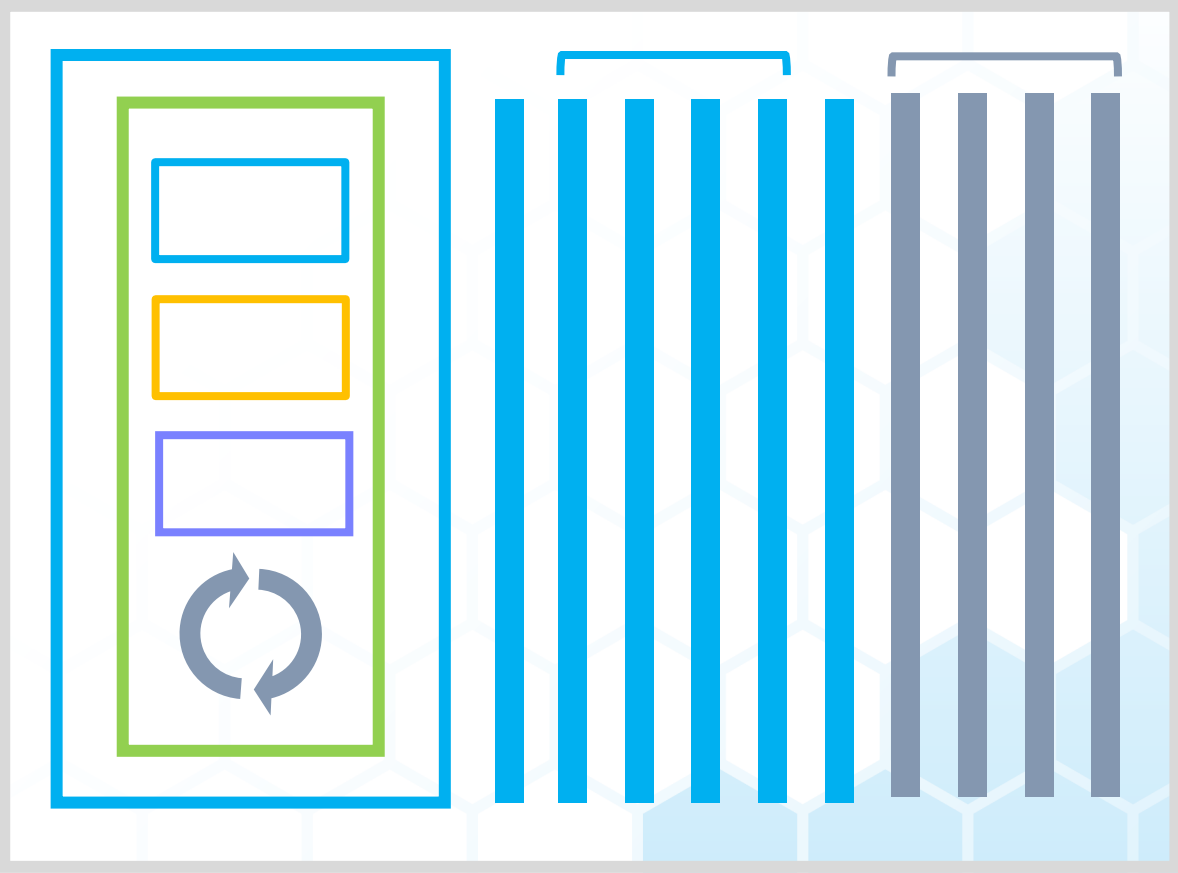
Creating a Worker



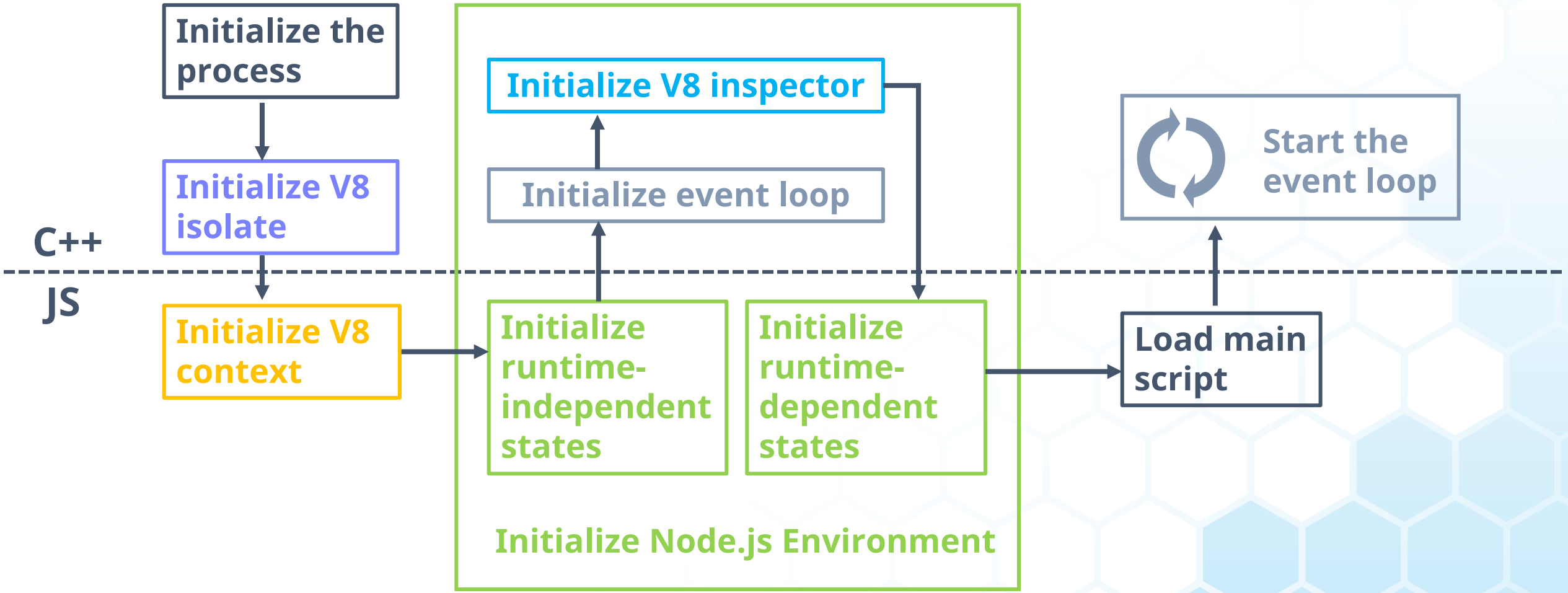
Creating a child process



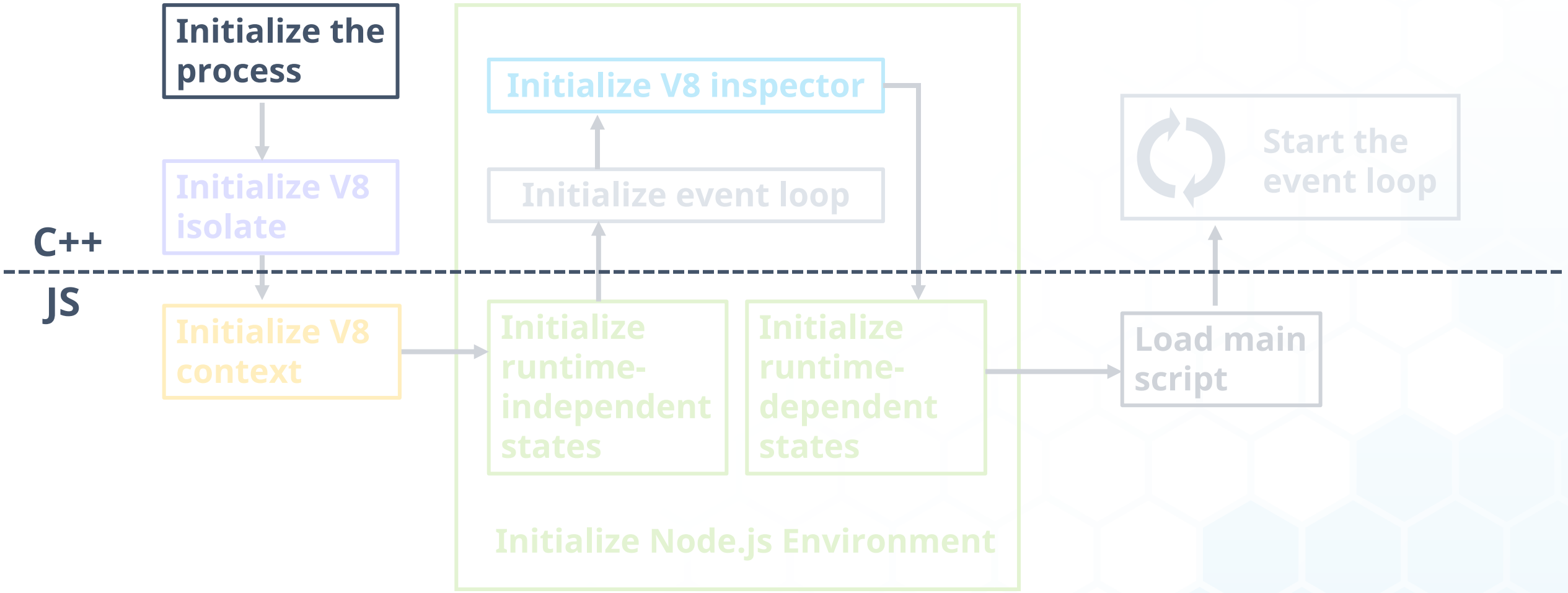
Creating a child process



Overview of the bootstrap



Setting up the process

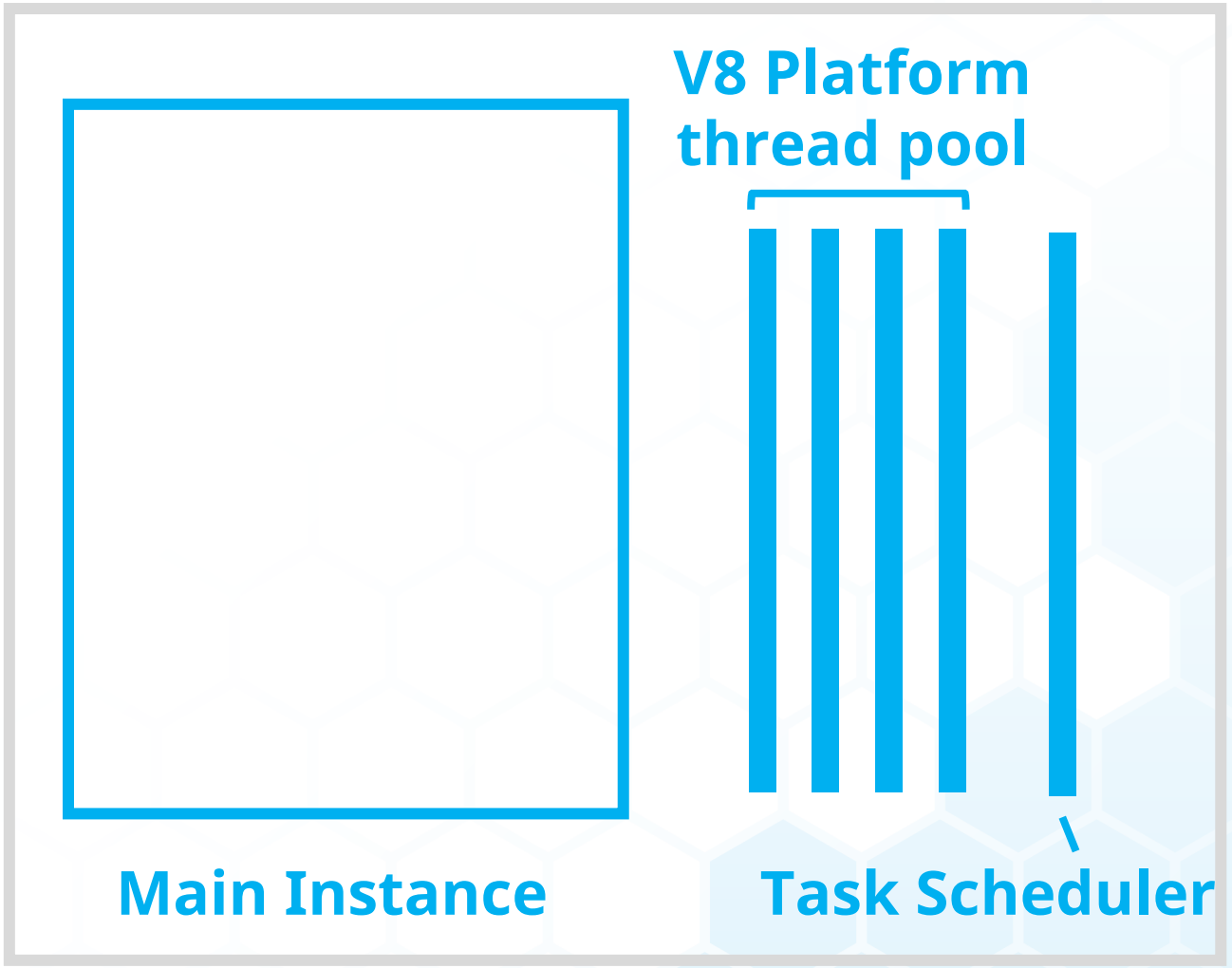


Setting up the process

- Setup signal handlers
- Parse CLI arguments (strings to C++ structures)
- Initialize ICU, OpenSSL

Setting up the process

- Setup signal handlers
- Parse CLI arguments (strings to C++ structures)
- Initialize ICU, OpenSSL
- **Initialize the V8 platform**

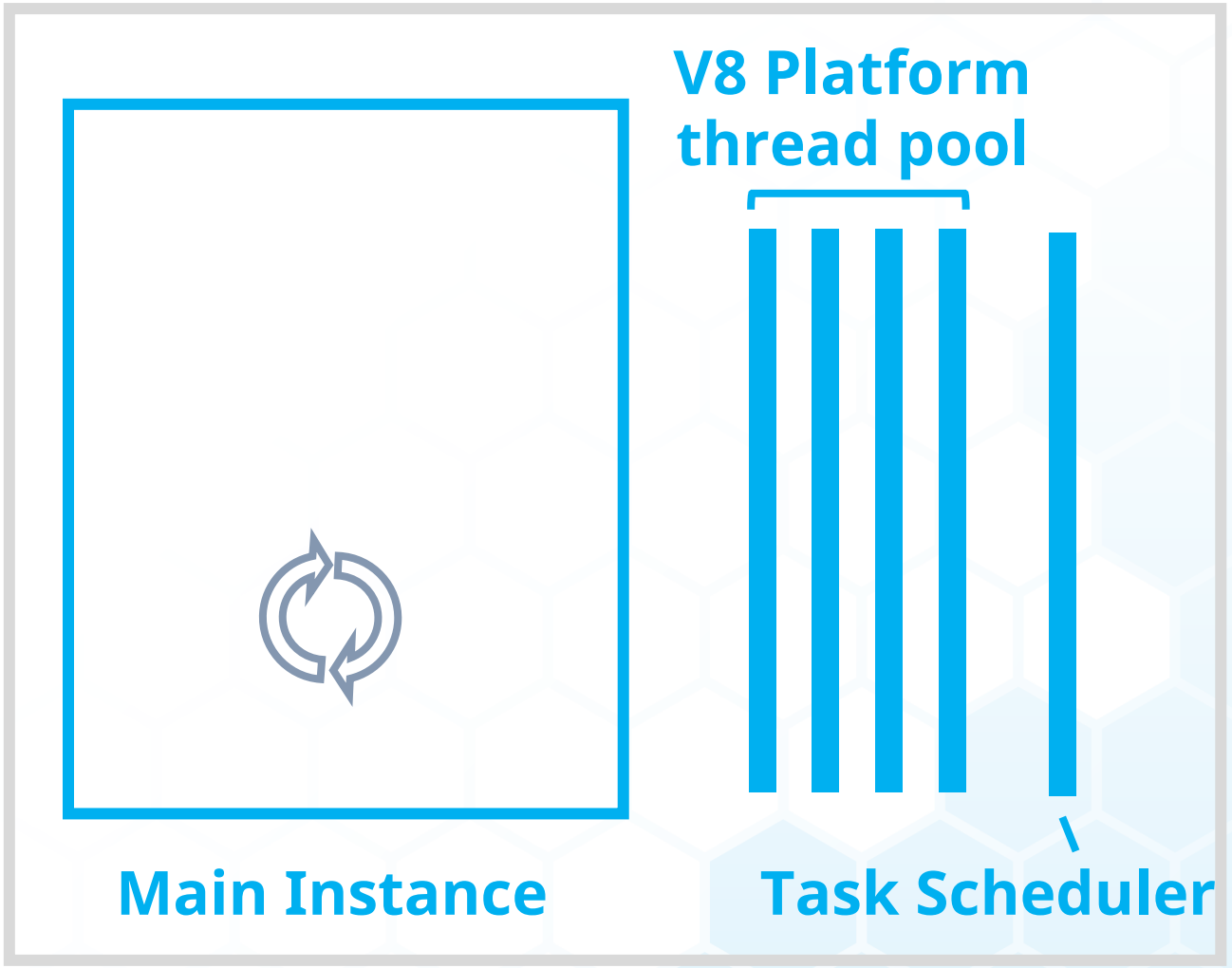


Setting up the process

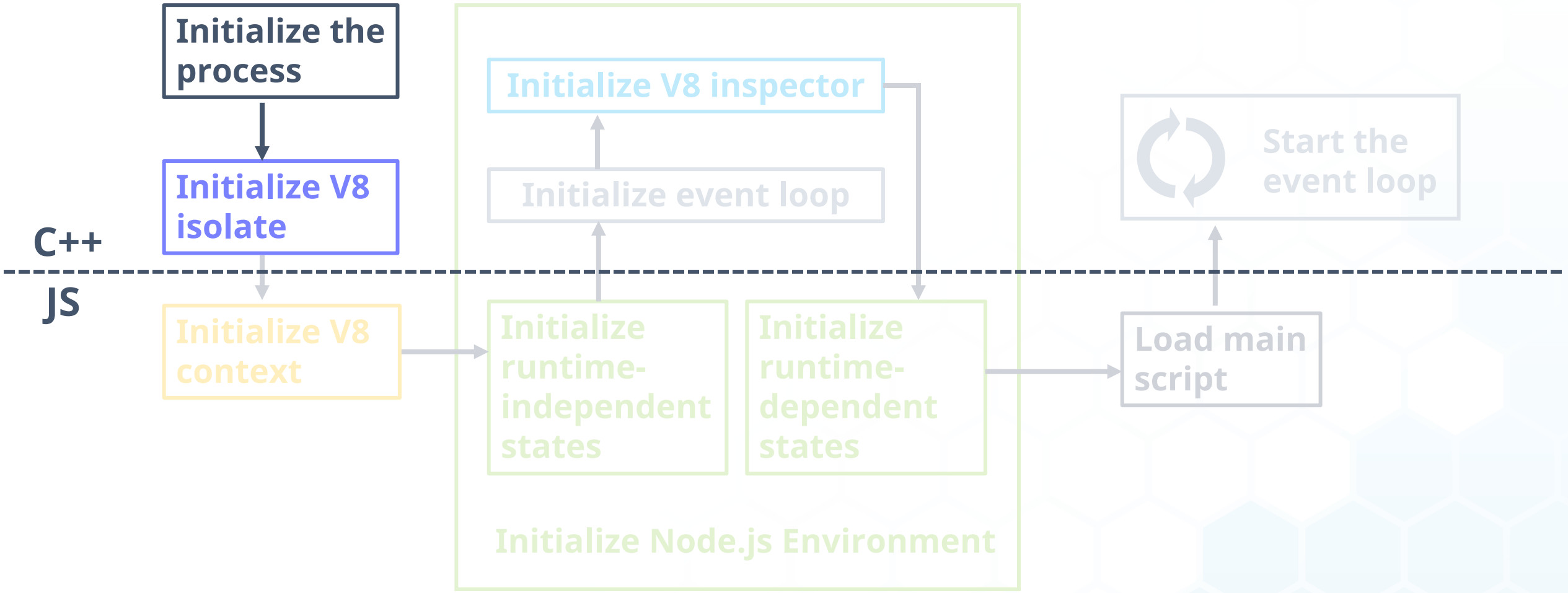
- Setup signal handlers
- Parse CLI arguments (string to C++ structures)
- Initialize ICU, OpenSSL
- Initialize the V8 platform

For the main instance

- Initialize the **default libuv event loop**



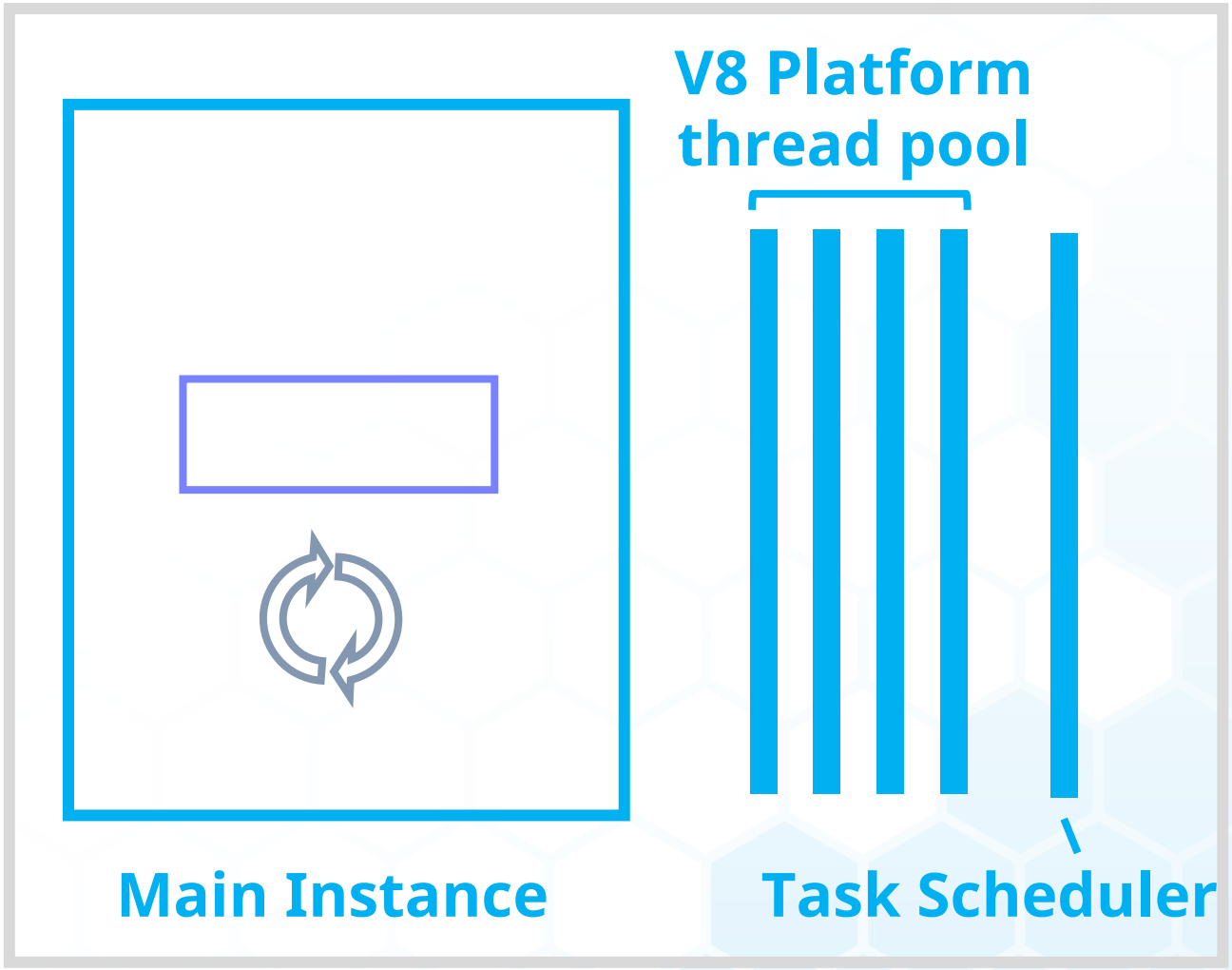
Setting up the V8 isolate



Setting up the V8 isolate

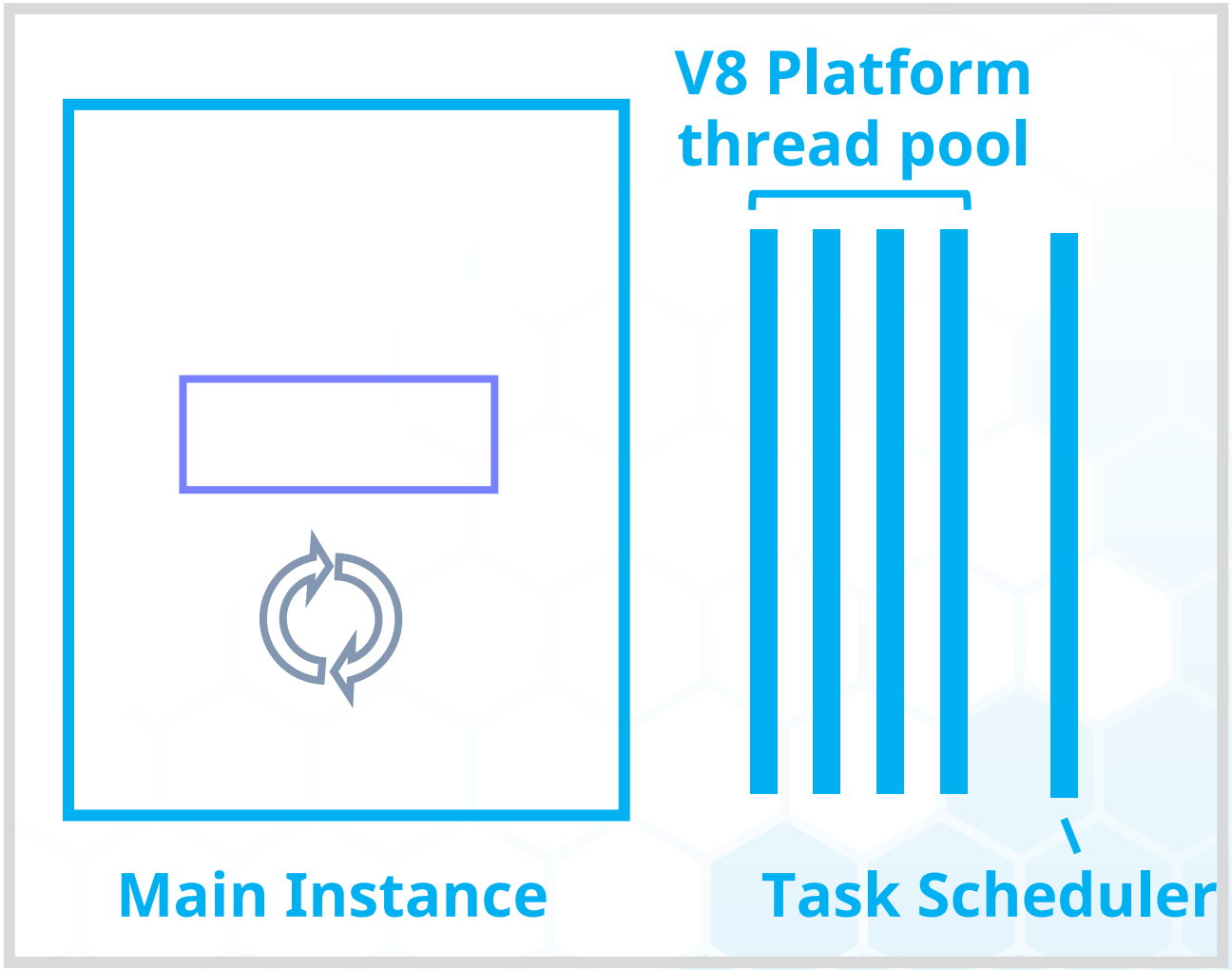
What's a V8 isolate?

- `v8::Isolate` is the instance of the v8 JavaScript engine
- Encapsulates the JS heap, microtask queue, pending exceptions…



Setting up the V8 isolate

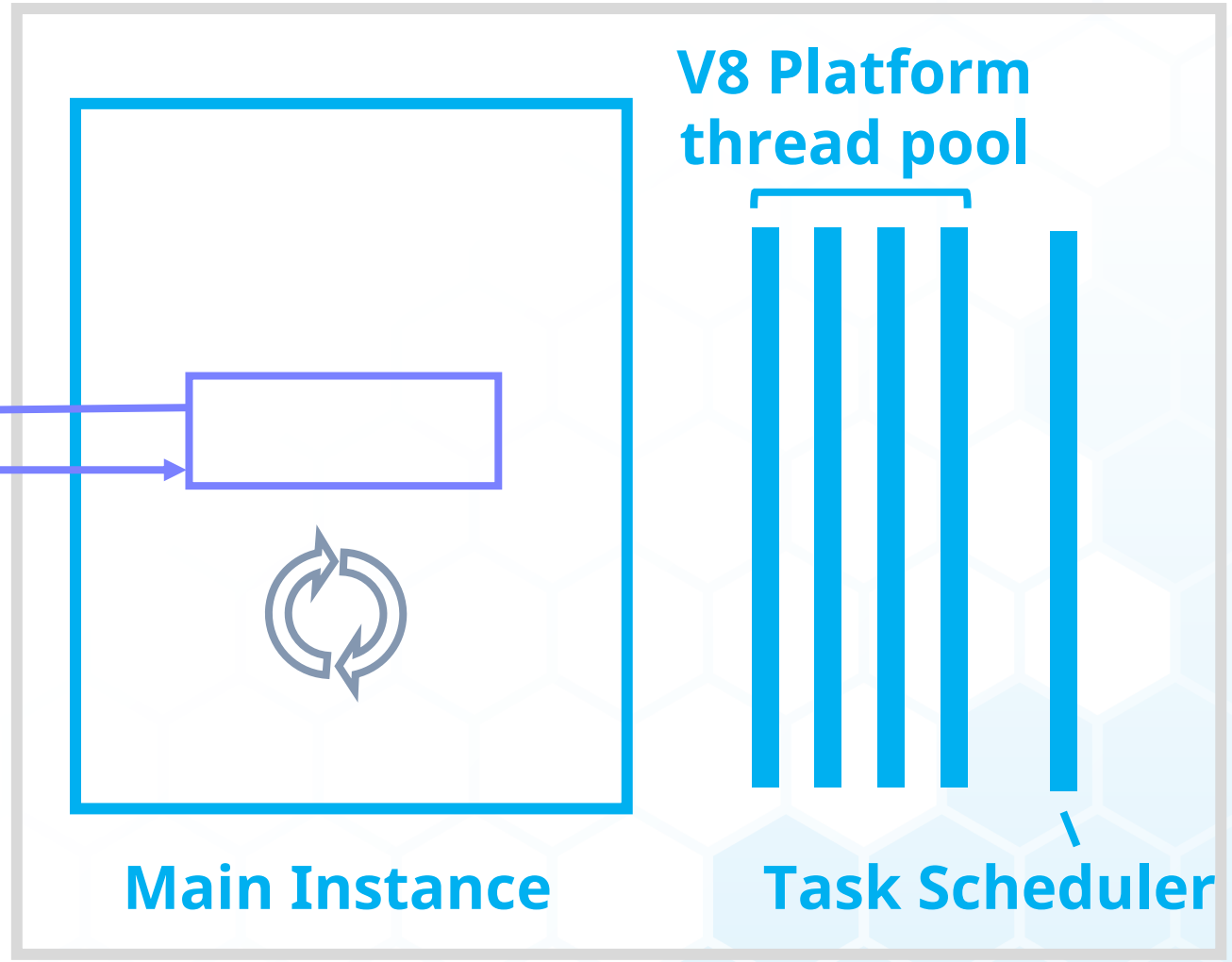
- Configure resource constraints (e.g. memory)
- Create array buffer allocator
- Deserialize from the V8 isolate snapshot



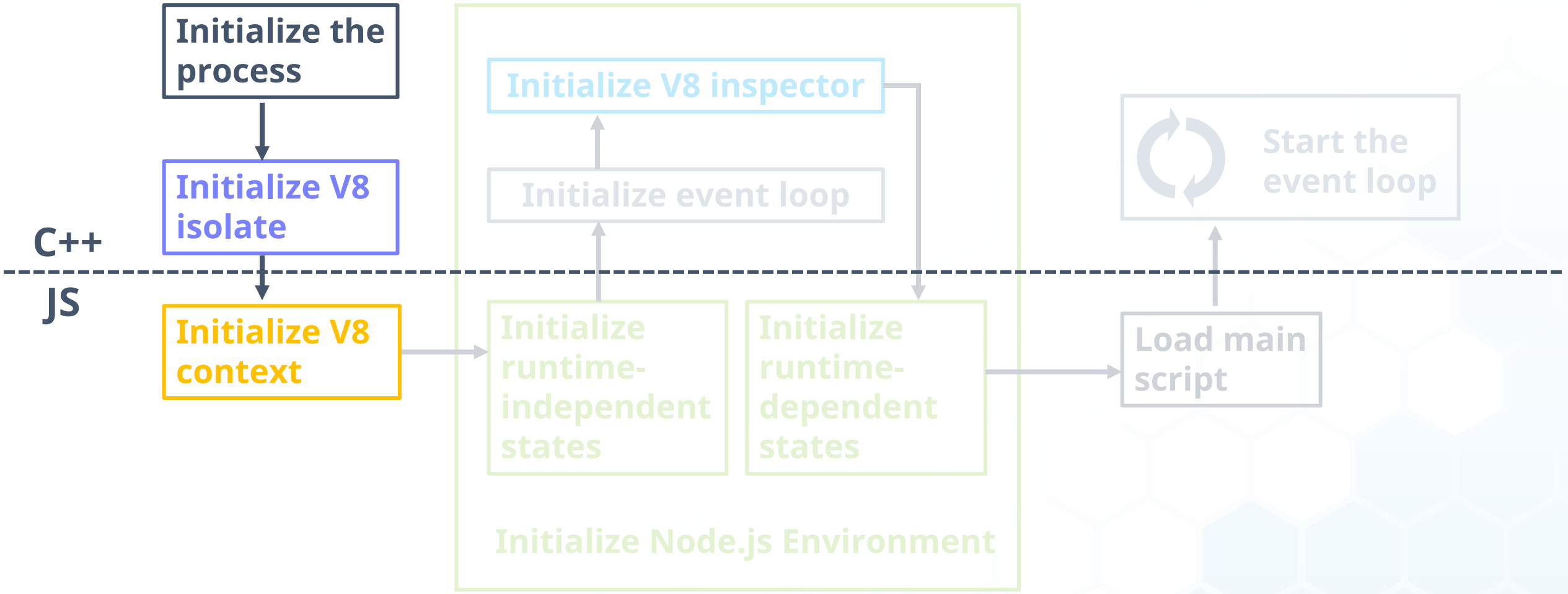
Setting up the V8 isolate

Setup various per-isolate callbacks in C++

- GC callbacks
- Uncaught exception listeners
- Promise rejection callbacks
- etc.



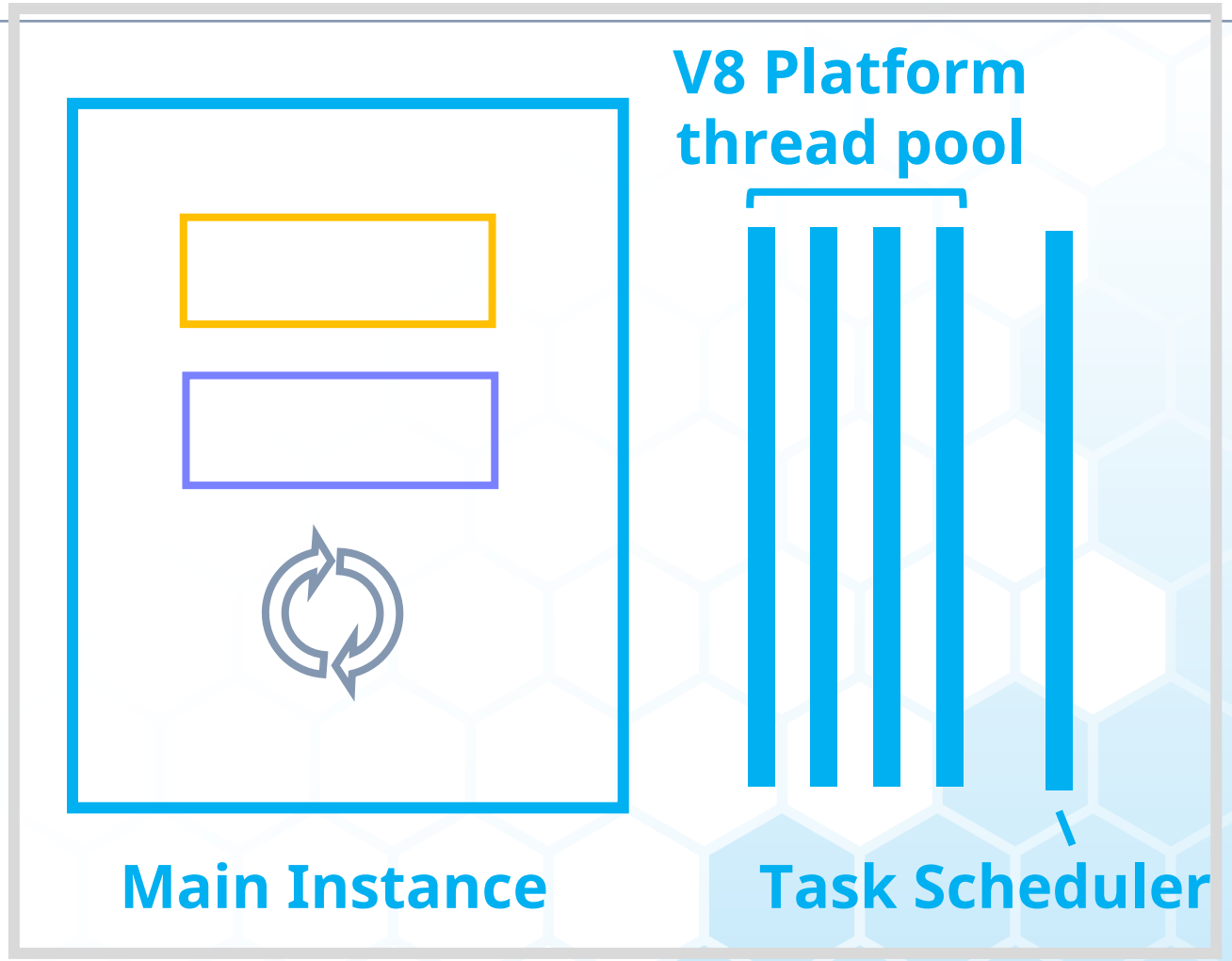
Setting up the V8 context



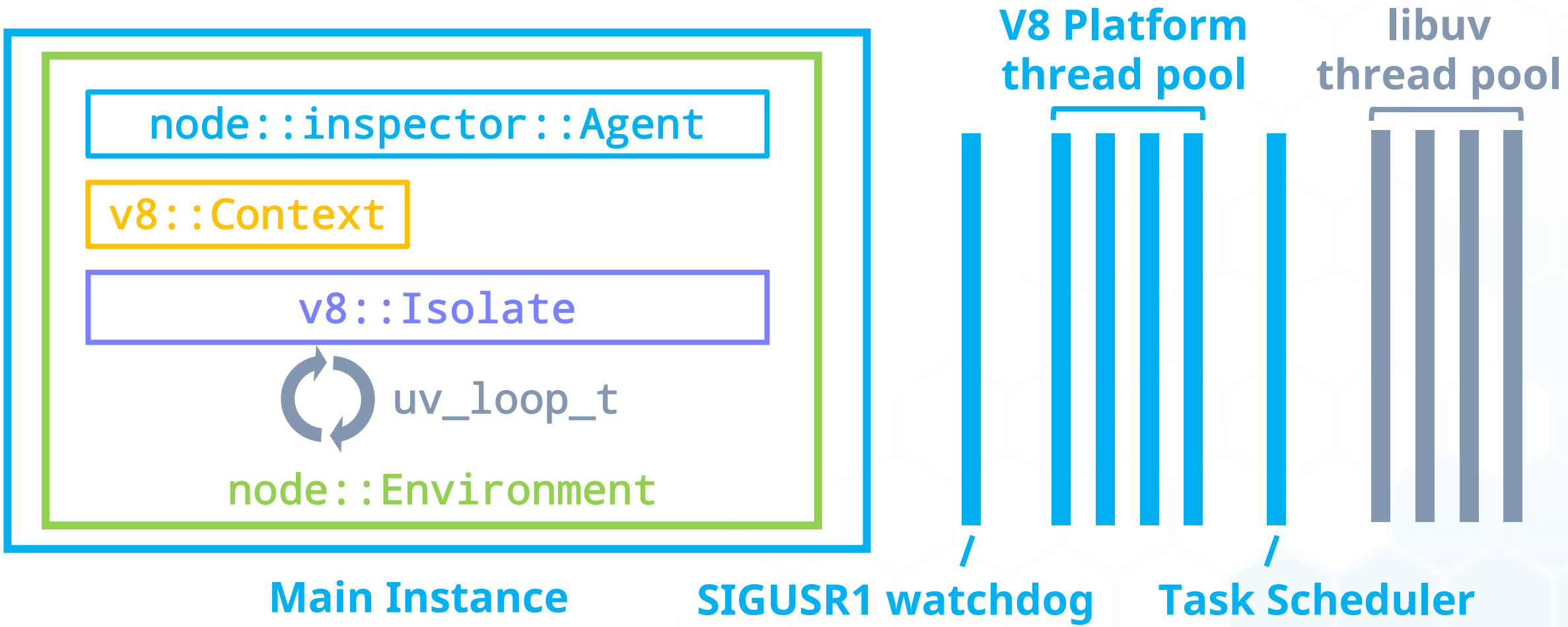
Setting up the V8 context

What's a V8 context?

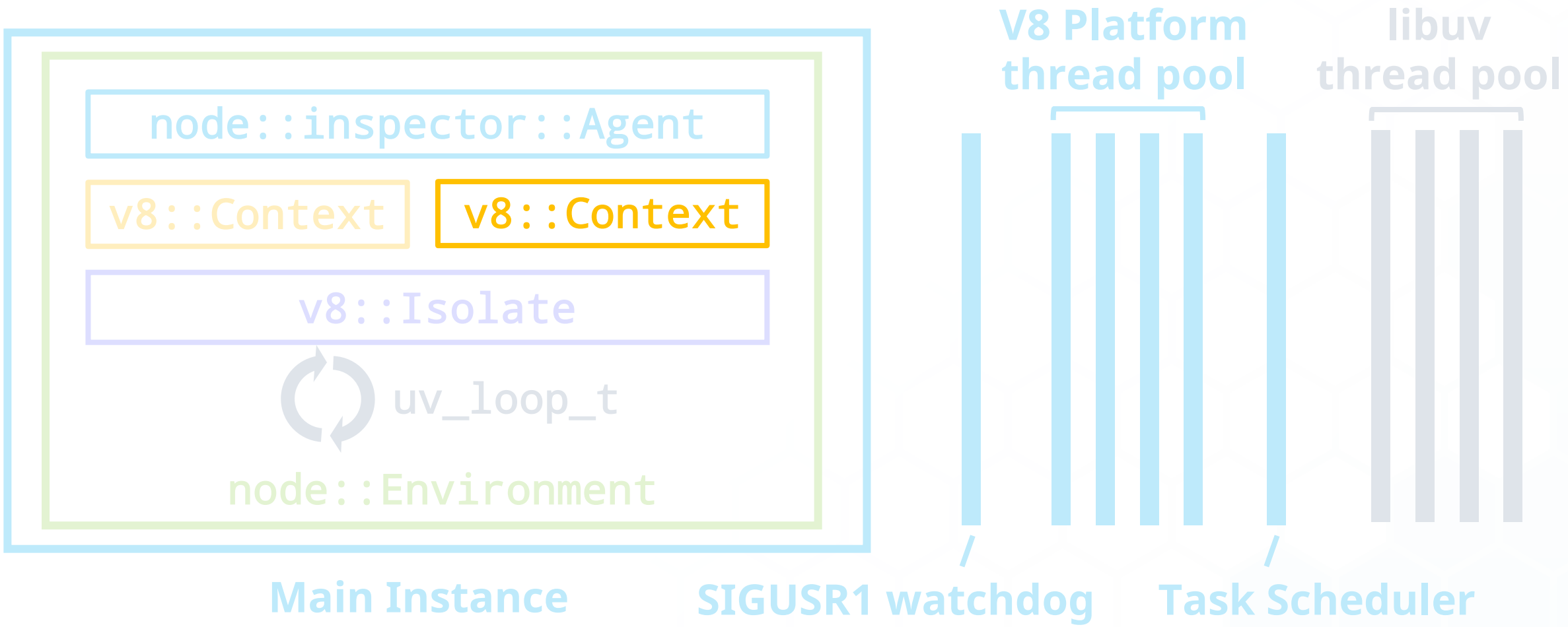
- A sandboxed execution context
- Encapsulates JavaScript builtins (primordials) e.g. `globalThis`, `Array`, `Object`…
- What's inside the returned result of `vm.createContext()`



Creating a vm Context



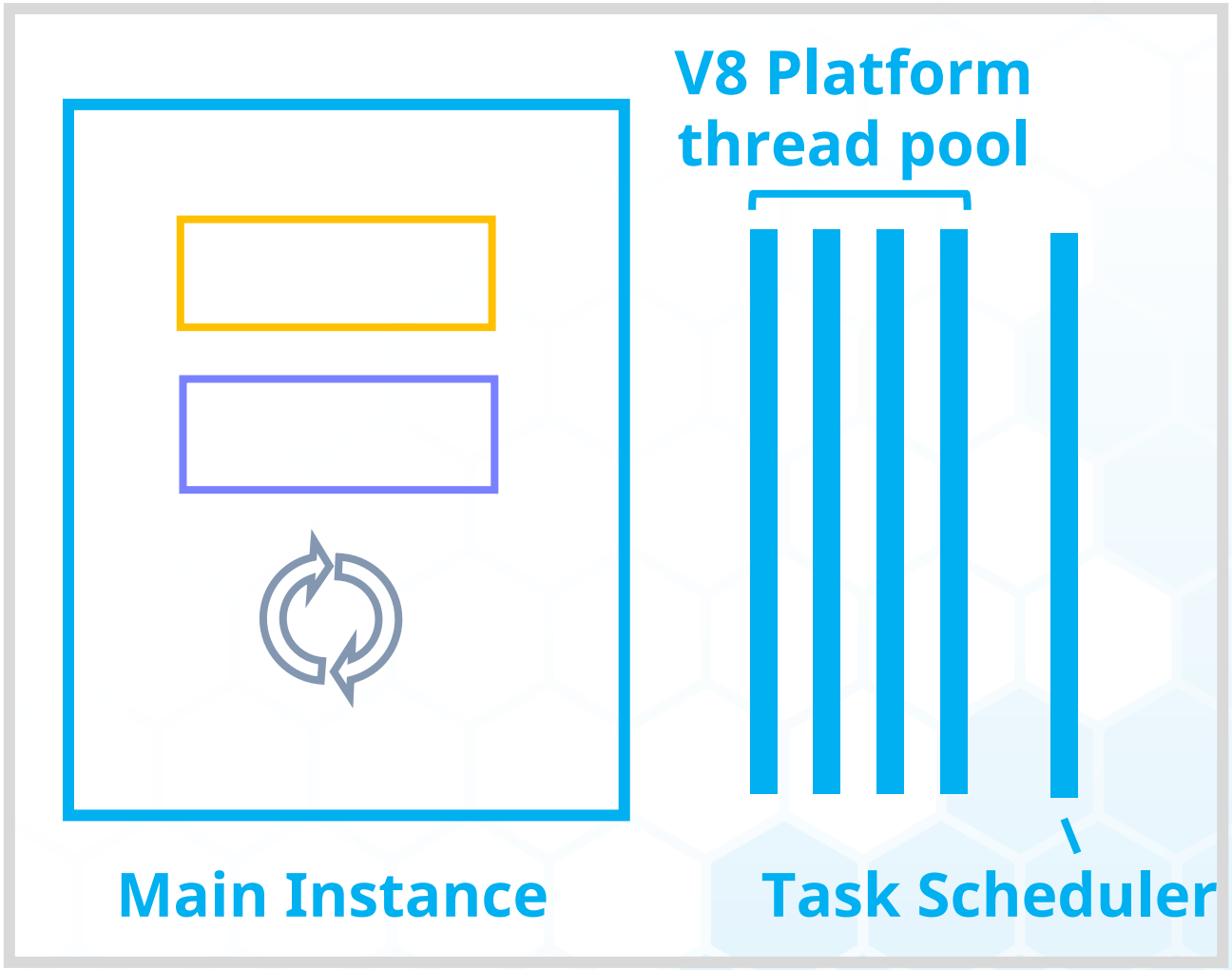
Creating a vm Context



Setting up the V8 context

What's inside a Node.js's V8 context?

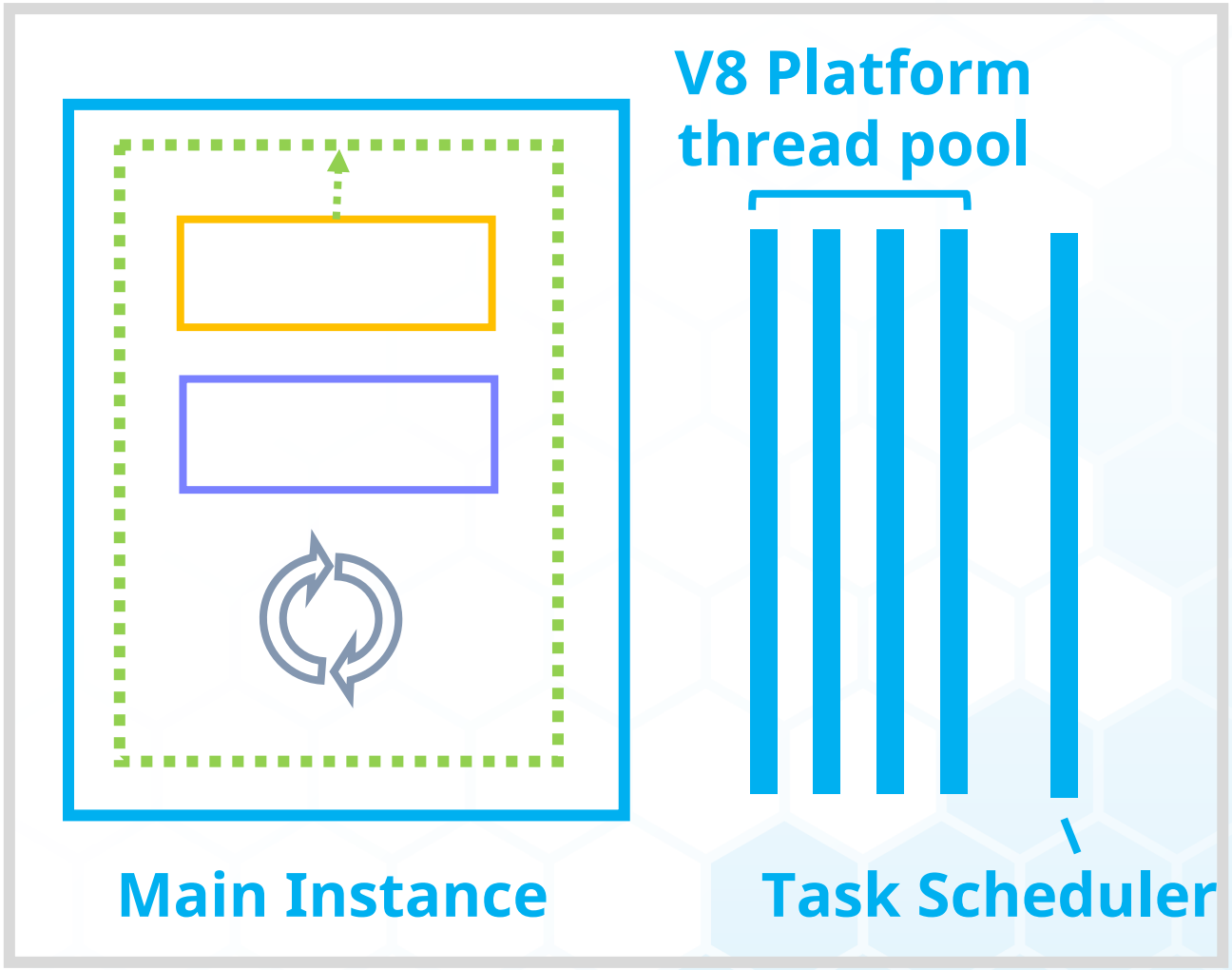
- Immutable copy of `primordials`
- `DOMException` for Web APIs (surprise!)



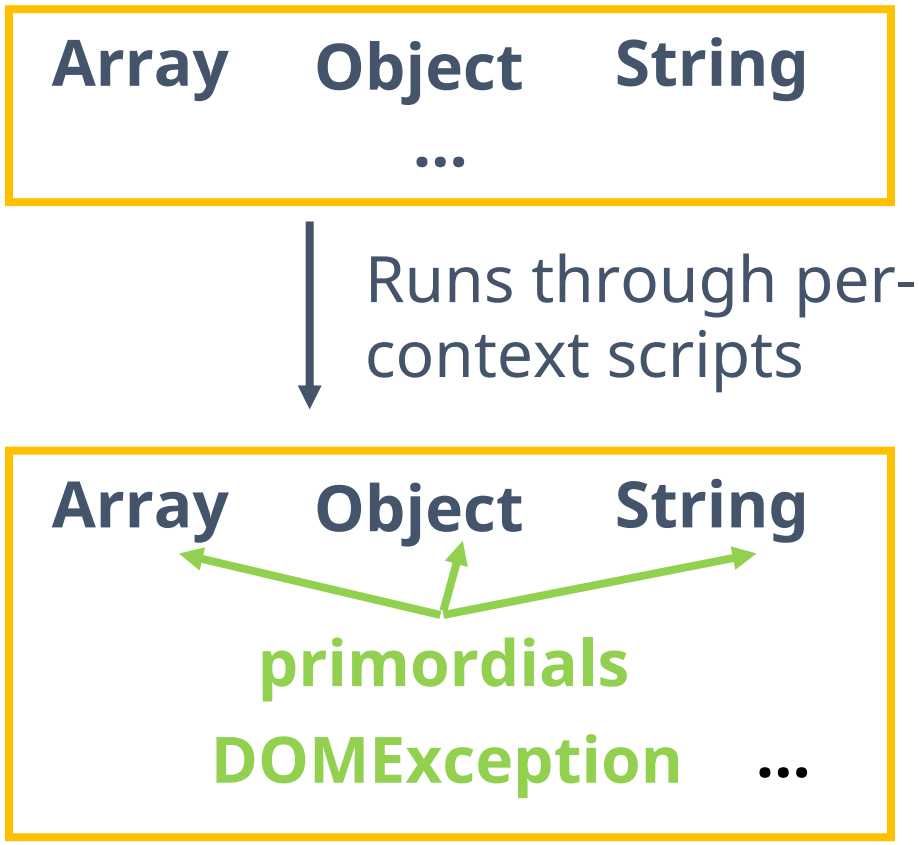
Setting up the V8 context

Main contexts & vm contexts

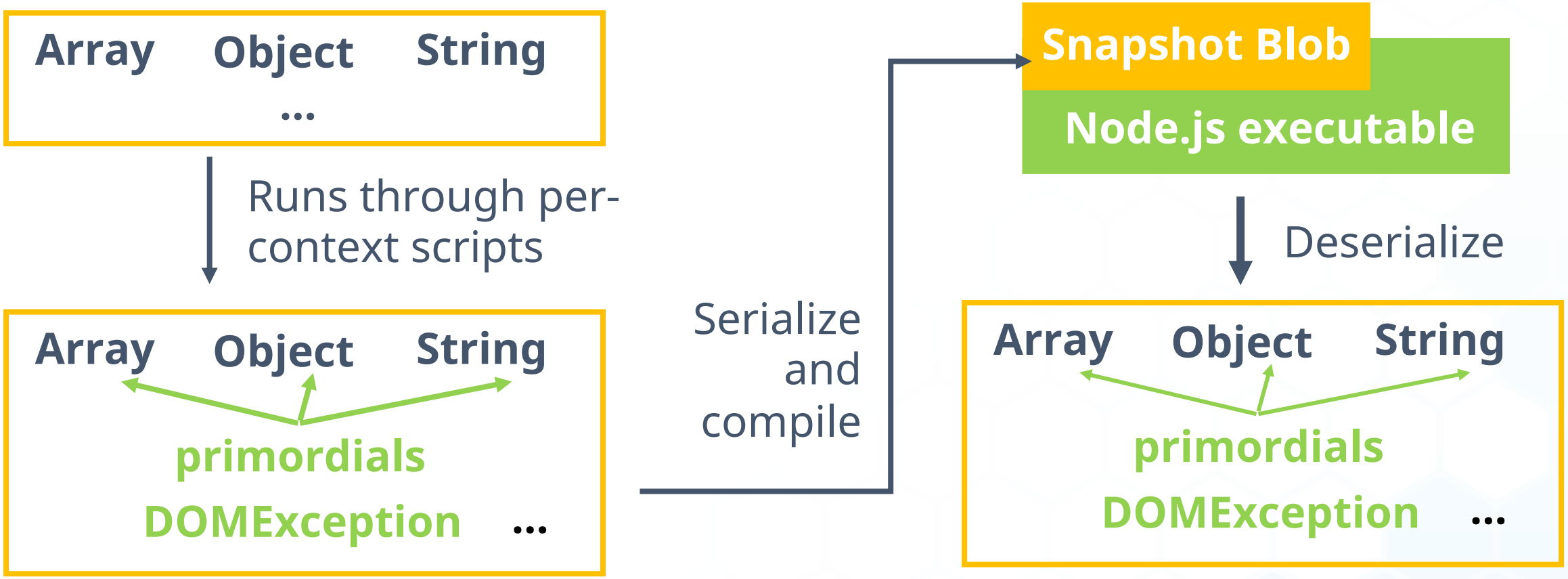
- Each Node.js instance has a main context (and potentially contexts created with vm).
- Main contexts contain a pointer to its associated Node.js `Environment` (not yet created)
- VM contexts do not have that pointer and are not bootstrapped further



V8 snapshot: before



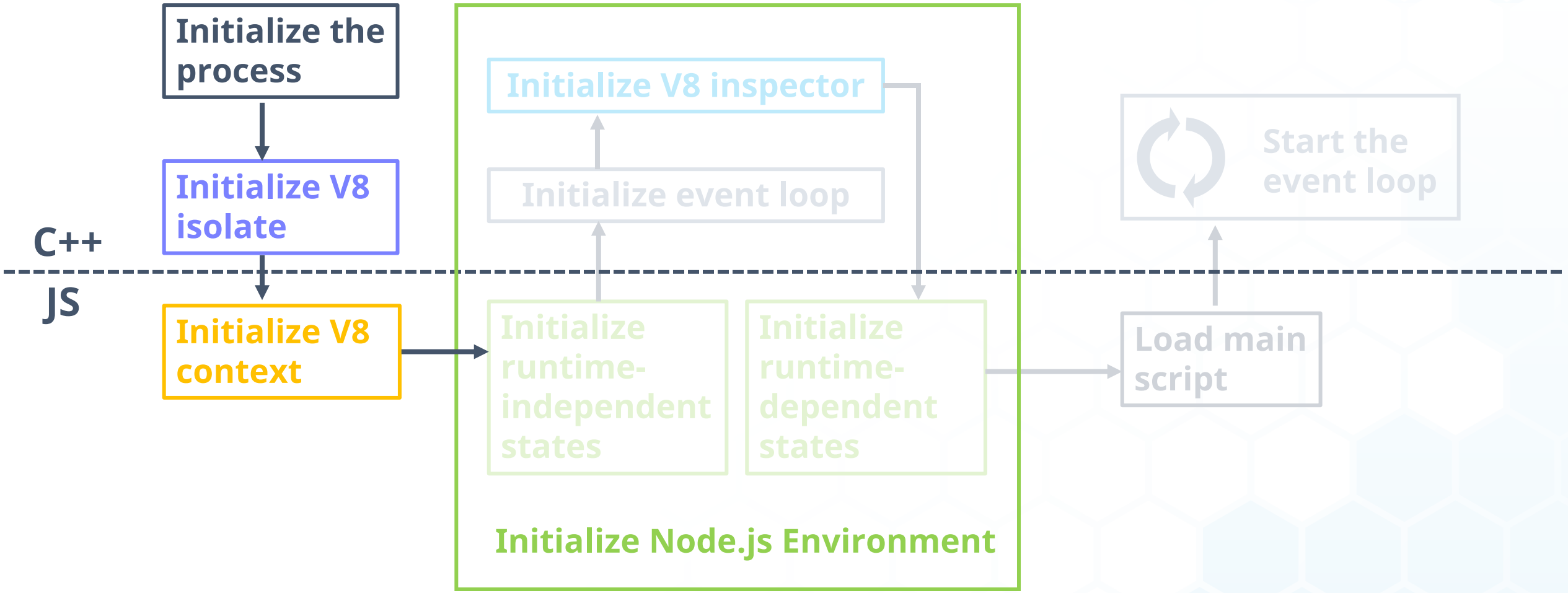
V8 snapshot: after



node_mksnapshot process (build time)

Node.js process (run time)

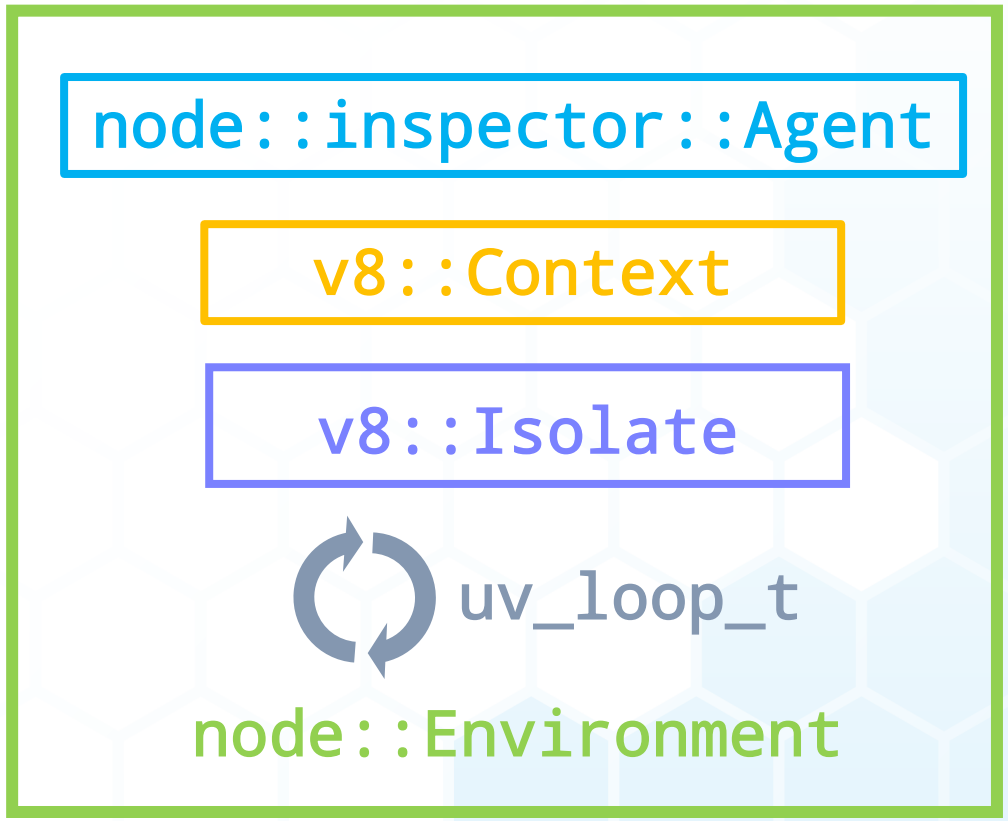
Setting up the Environment



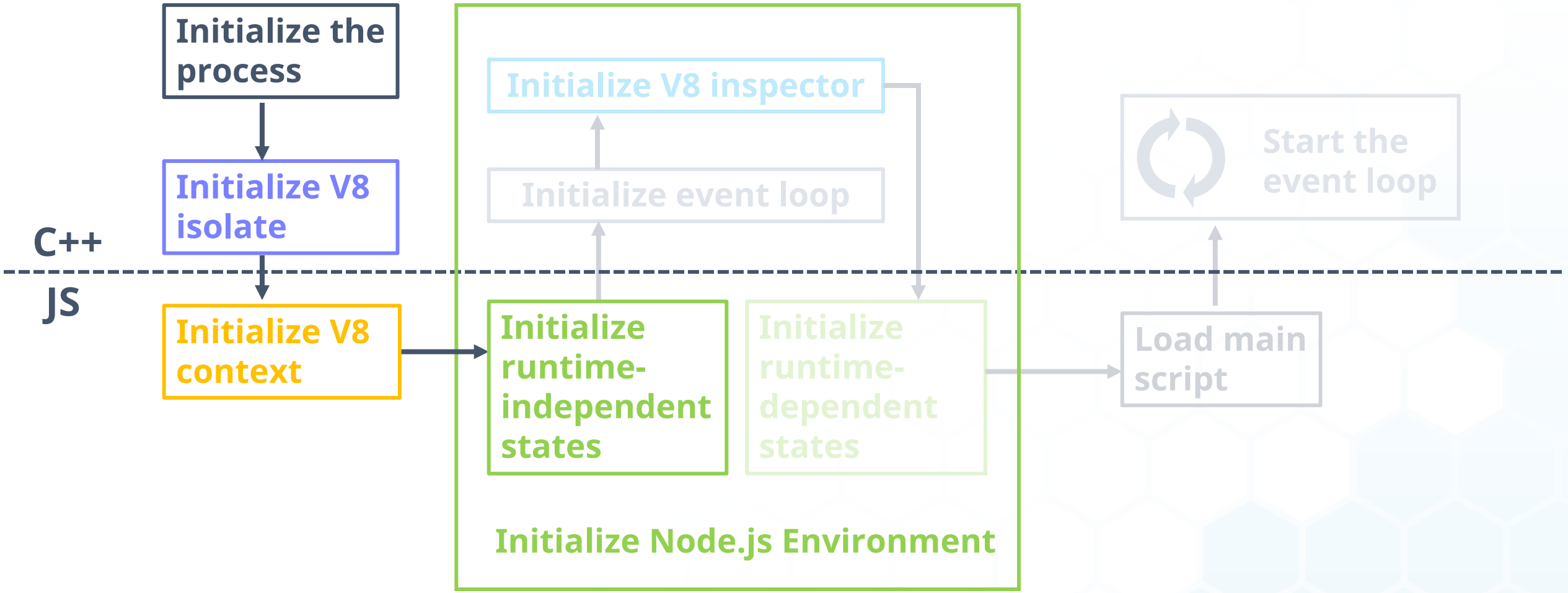
Setting up the Environment

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

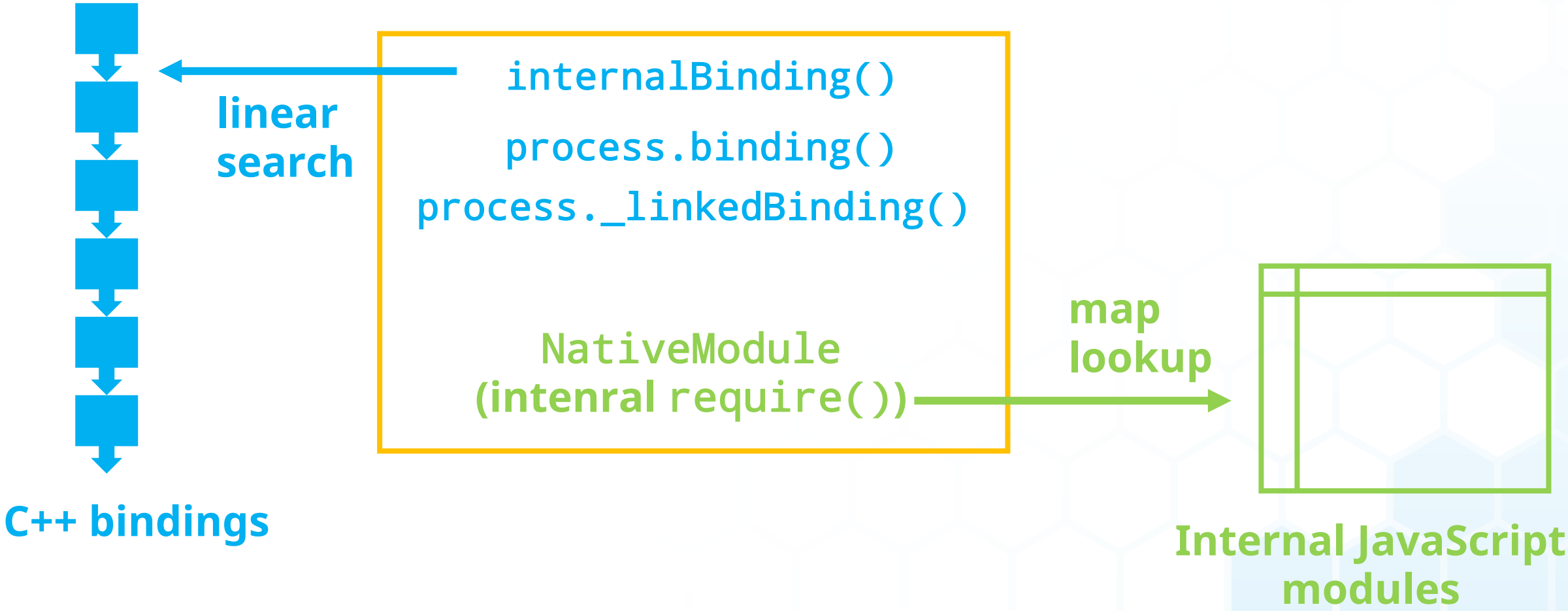


Setting up the Environment

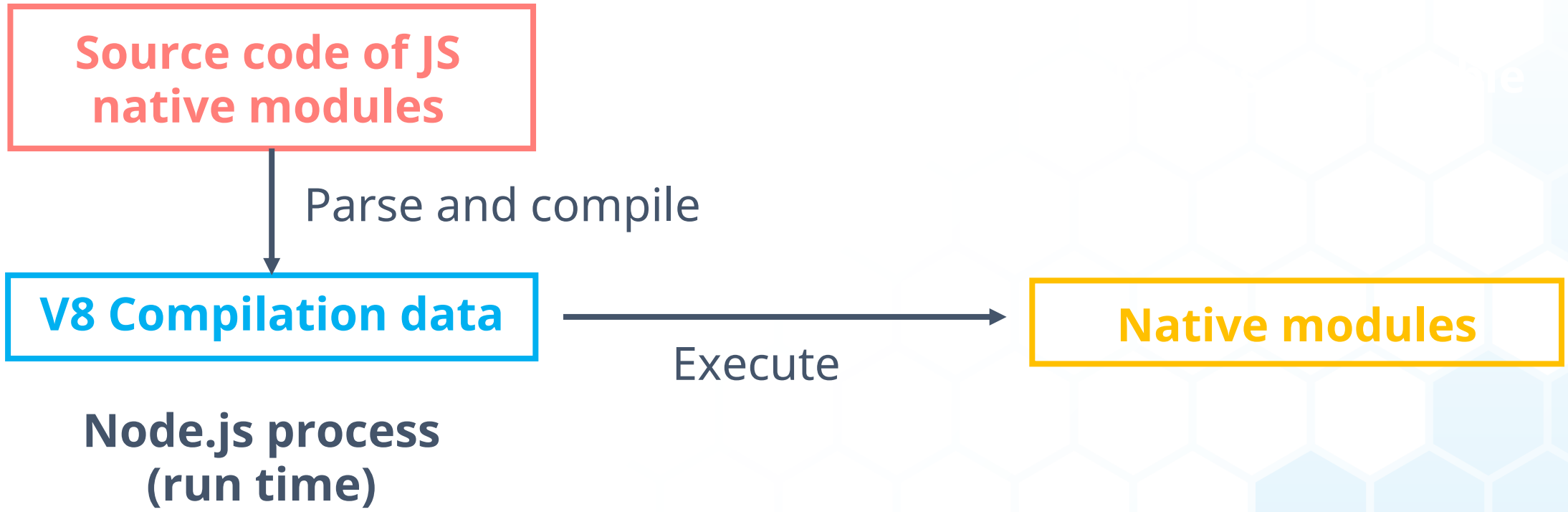
What needs to be initialized?

- **Components independent of runtime states**
  - Internal JavaScript module and C++ binding loaders
  - The process object and other globals
  - JavaScript callbacks that C++ hooks invoke

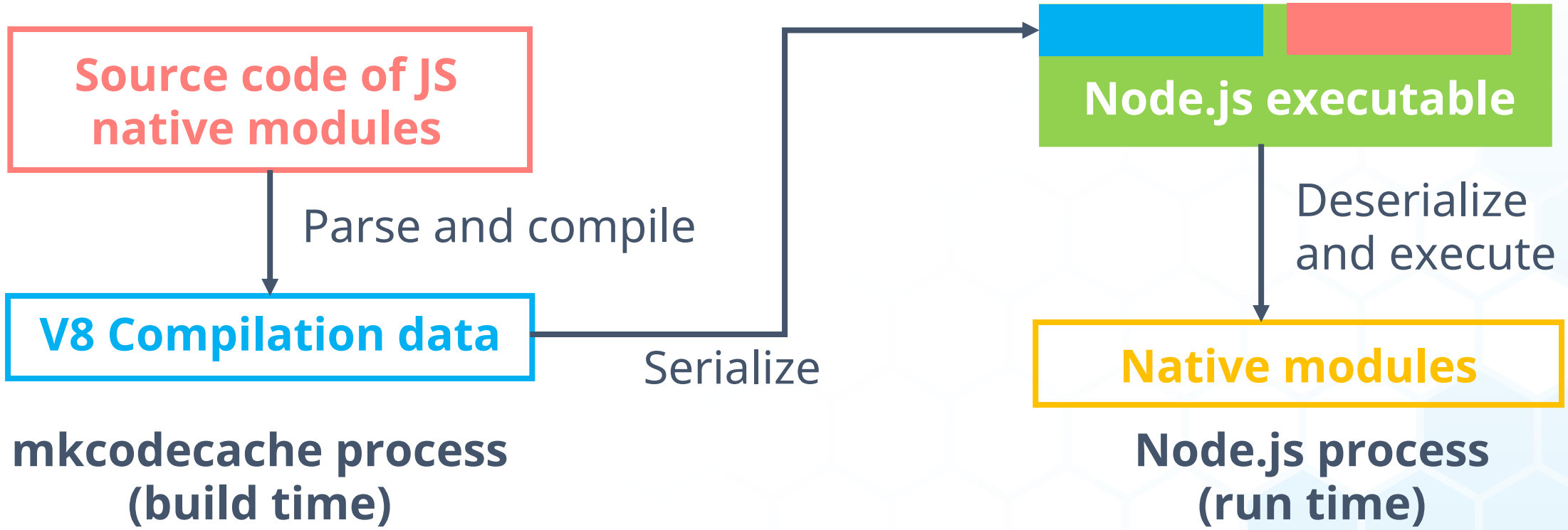
Environment: internal loaders



V8 code cache: before



V8 code cache: after



Environment: globals

- Most globals are implemented in internal JavaScript with access to internal C++ bindings (glue to dependencies)
- Created and attached to `global` or `process` during bootstrap
- `global` is a legacy alias to the ECMAScript stage 4 `globalThis`

Environment: globals

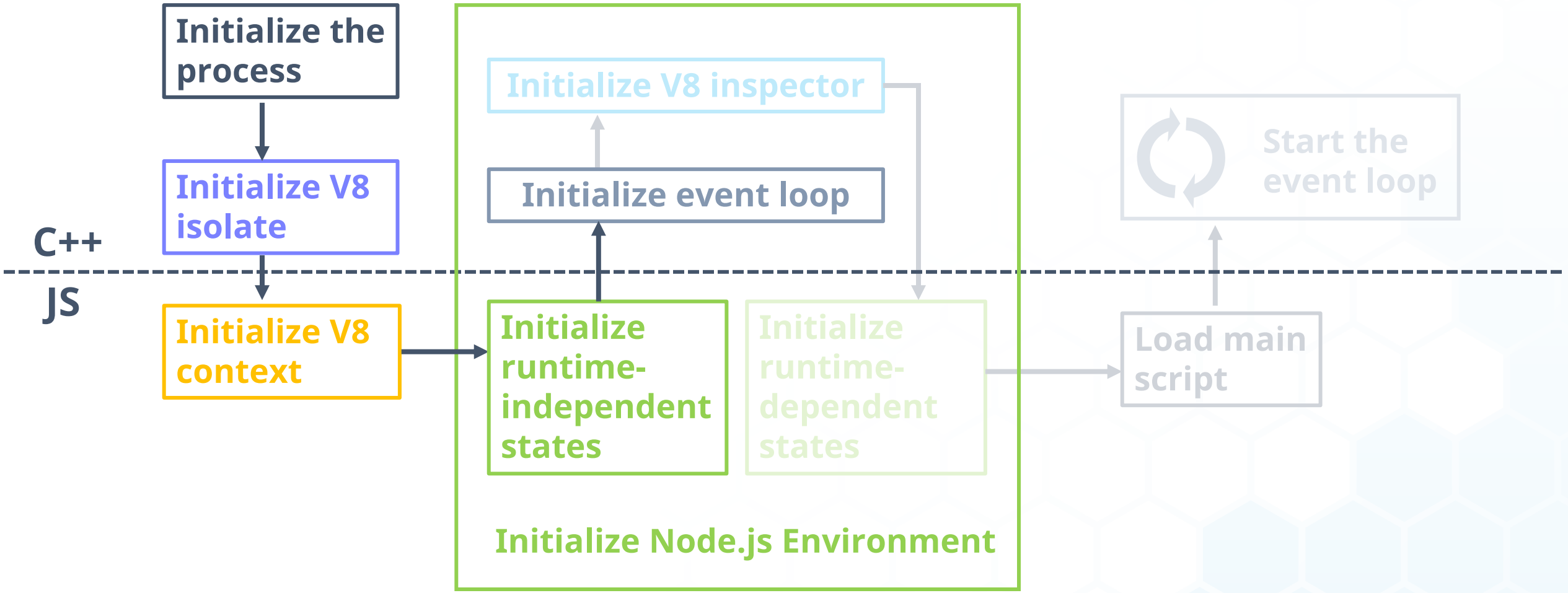
```
process.nextTick =
 require('internal/process/task_queues').setupTaskQueue().nextTick;
...

Object.defineProperty(global, 'process', { value: process, ...});
Object.defineProperty(global, 'global', { value: global, ...});
Object.defineProperty(global, 'setTimeout', {
  value: require('timers').setTimeout),
  ...
};
```

Environment: initialize hooks

Example of hooks

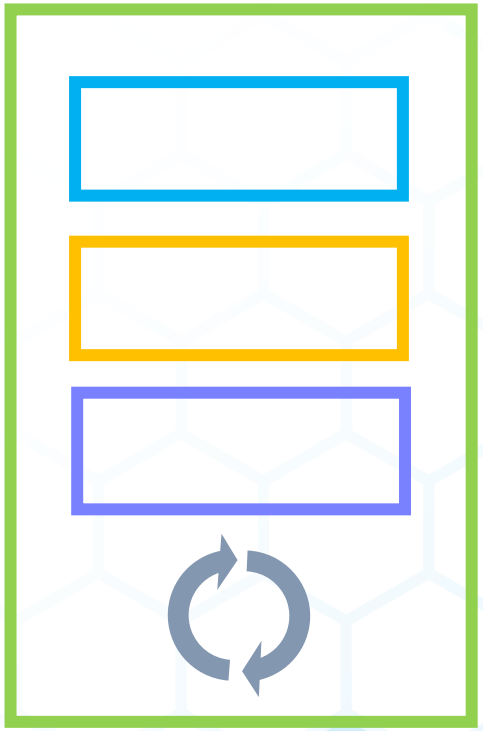
- **Async hooks (per-Environment)**
  - User callbacks need to be invoked at different stages of async operations
- **`process.nextTick` (per-Environment)**
  - Queued user callbacks need to be invoked when async operations are done
- **`Error.prepareStackTrace` (per-Isolate)**
  - User hook needs to be invoked when `error.stack` is accessed
- **`process.on('uncaughtException')` / `process.on('unhandledRejection')` (per-Isolate)**
  - Need to be invoked when there are uncaught exceptions/unhandled rejections

Setting up the Environment

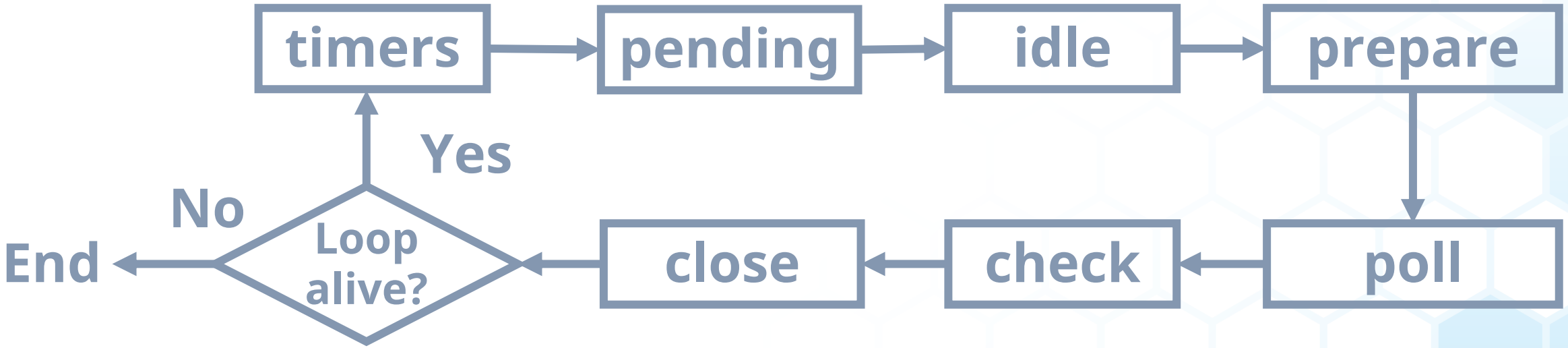


Environment: initialize the event loop

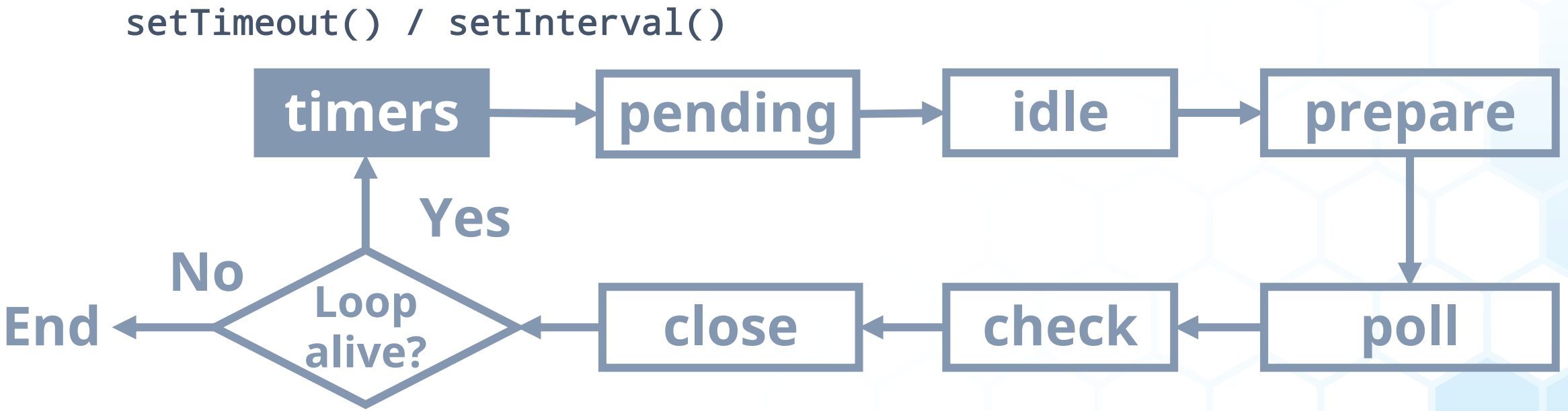
- Some handles are initialized at bootstrap
- Some are activated immediately, some are activated on demand
- More handles (e.g. poll handles) can be added on-demand (e.g. for I/O)



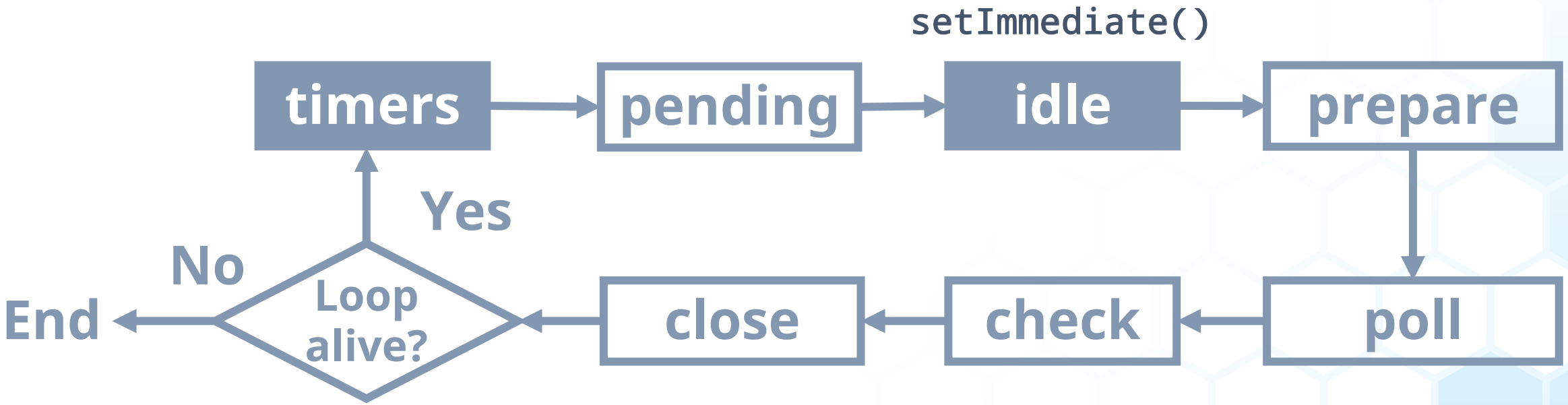
Environment: initialize the event loop



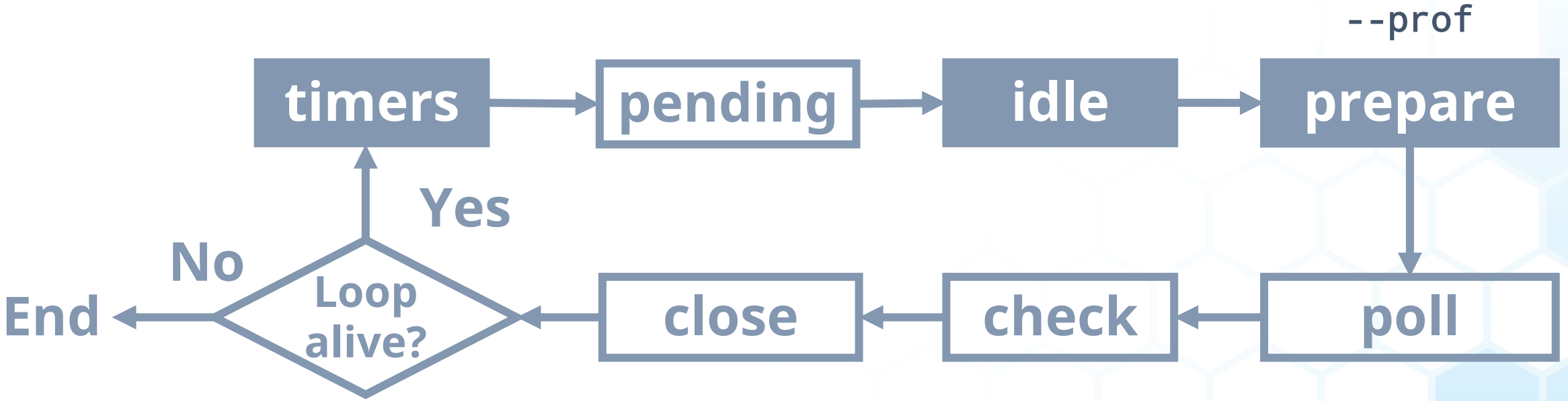
Environment: initialize the event loop



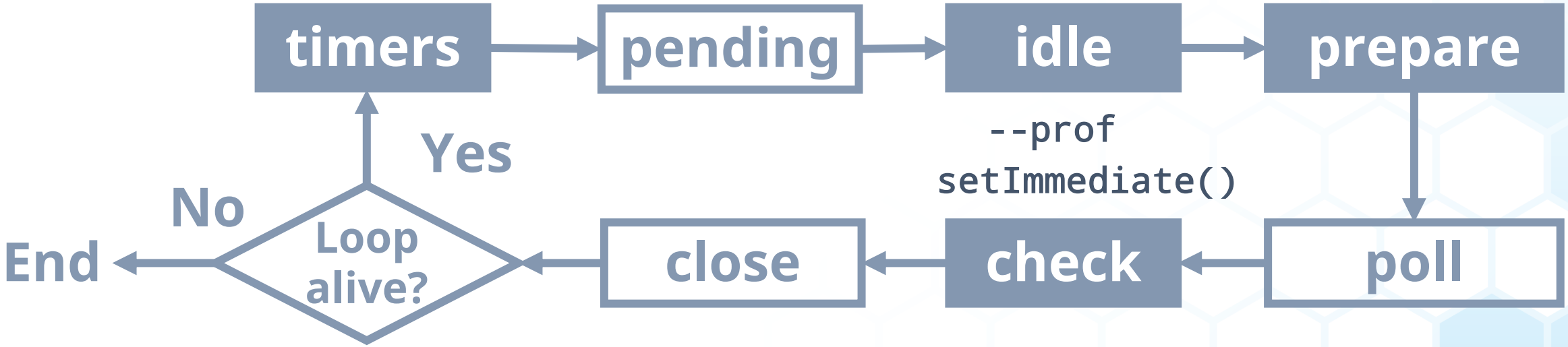
Environment: initialize the event loop



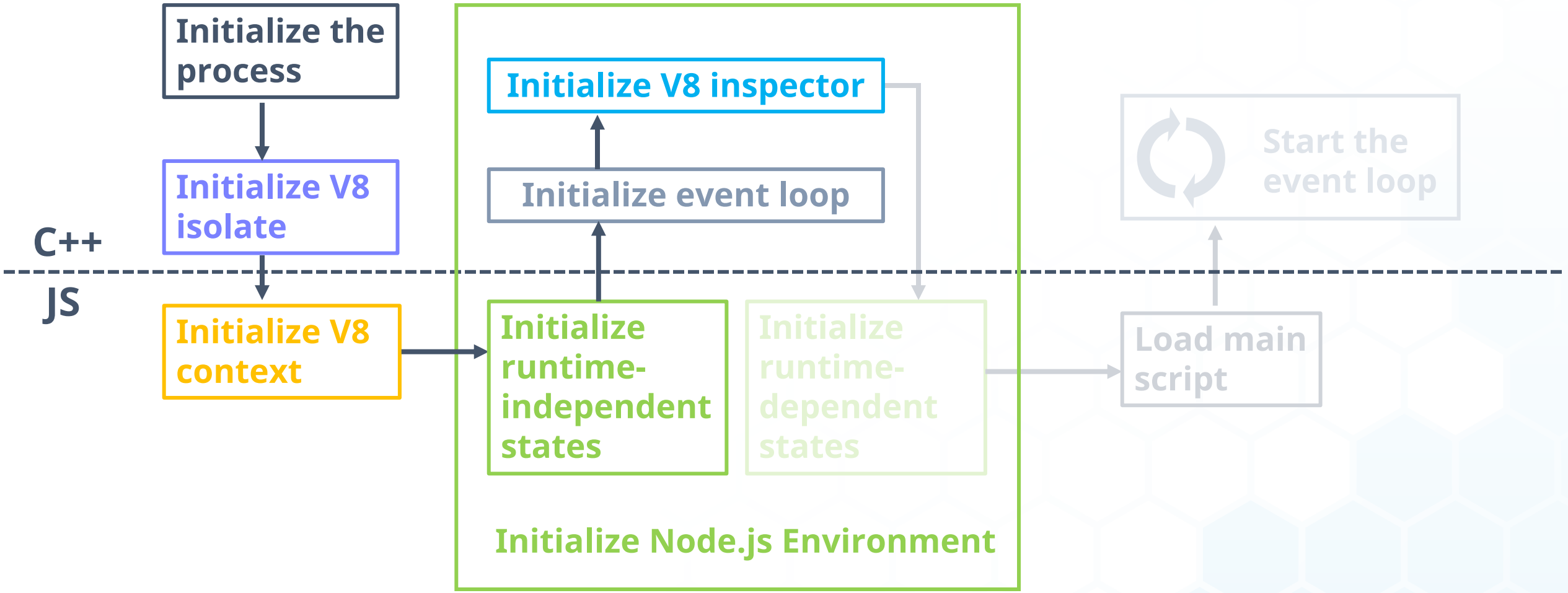
Environment: initialize the event loop



Environment: initialize the event loop

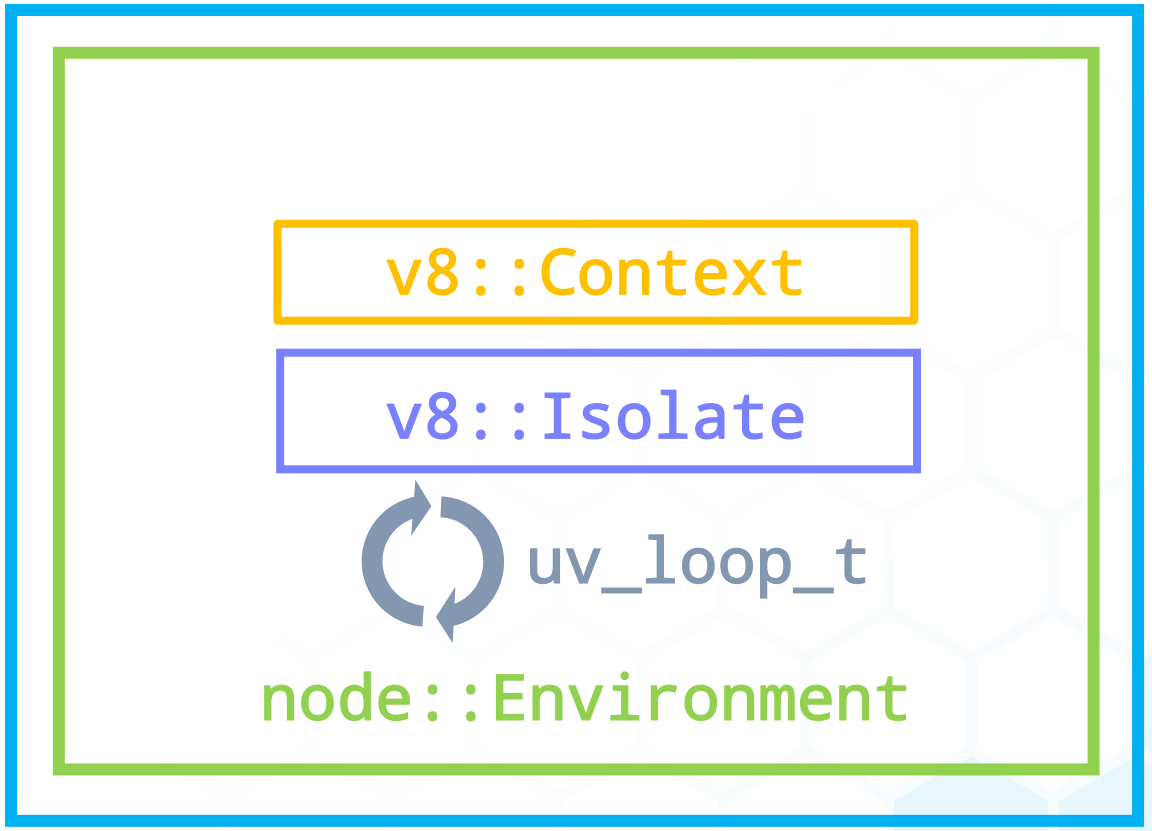


Setting up the Environment



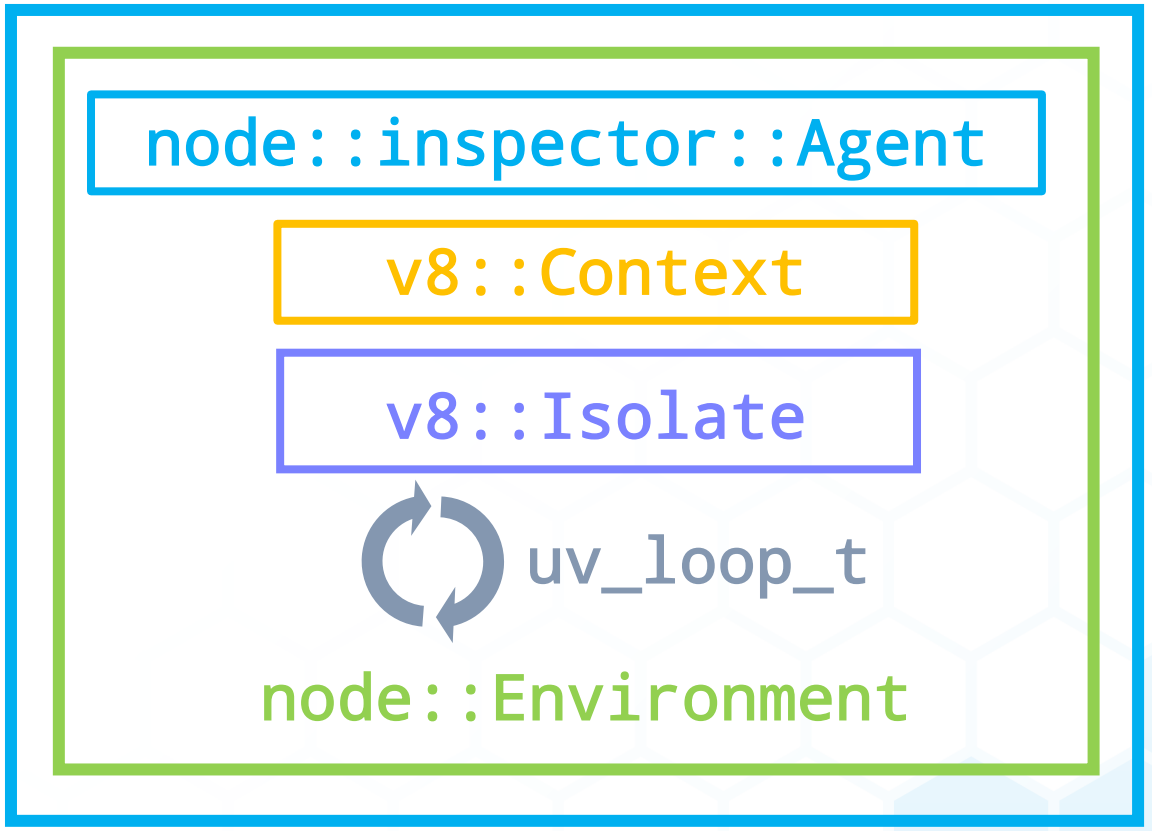
Environment: inspector

- Initialize inspector agent



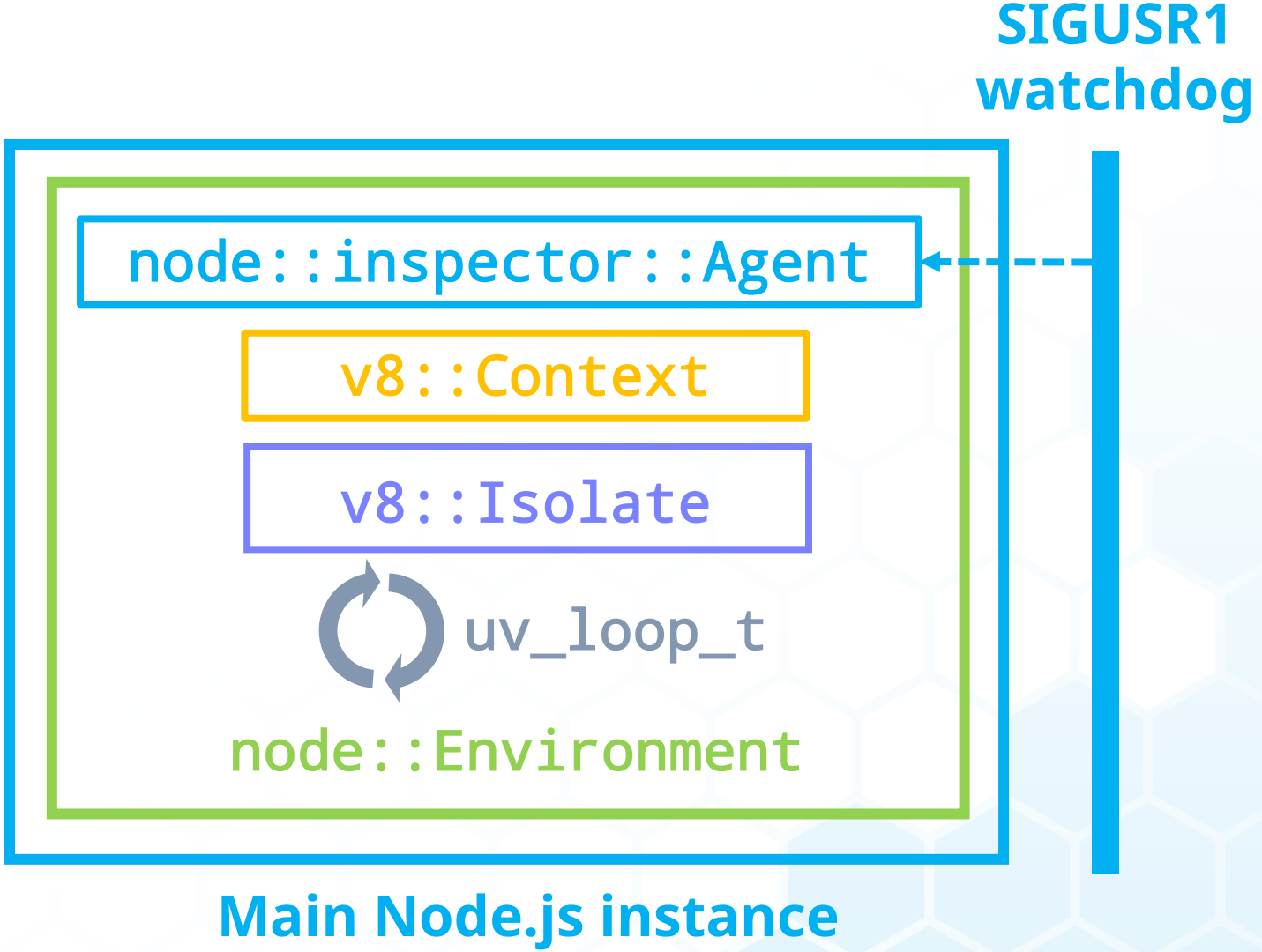
Environment: inspector

- Initialize inspector agent



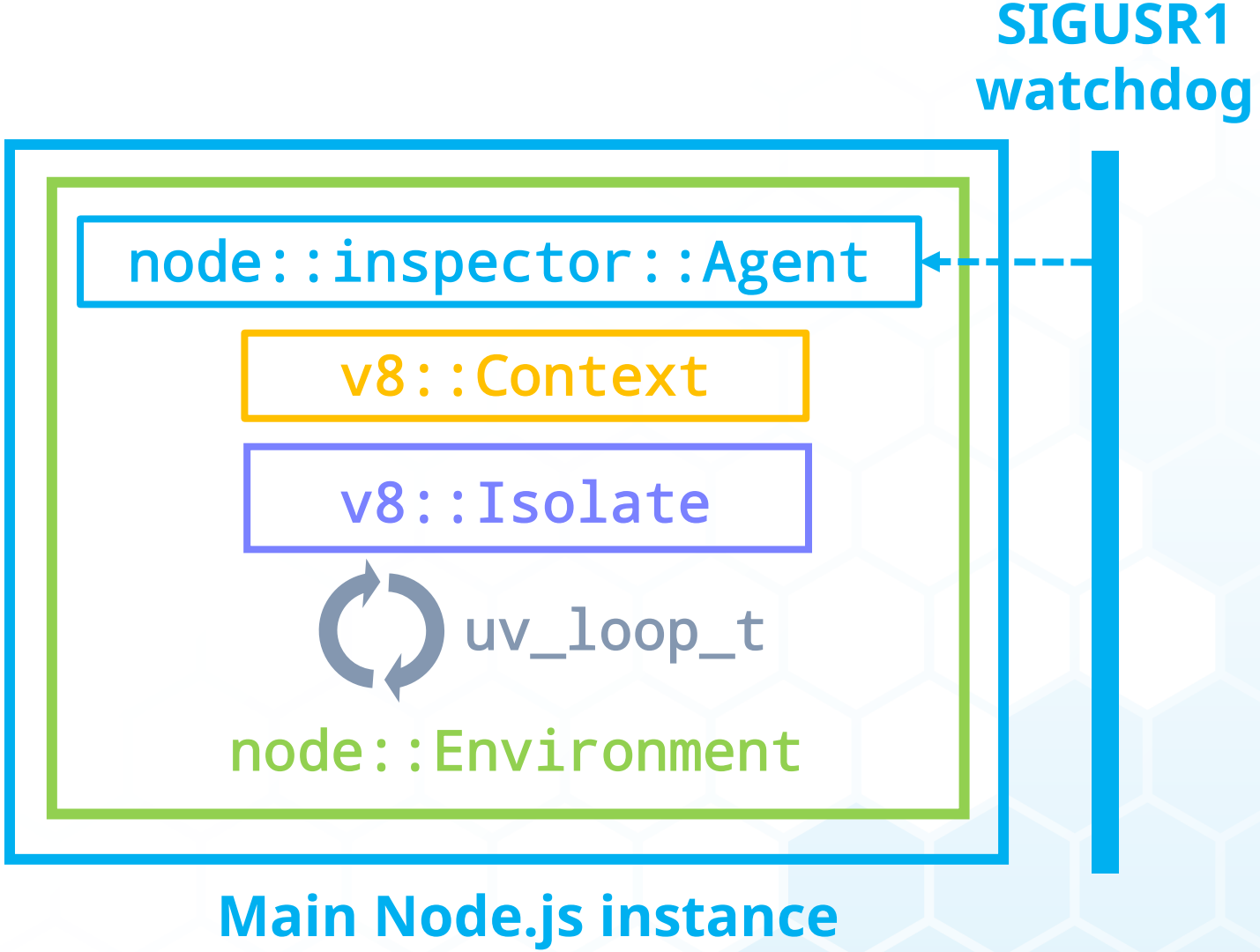
Environment: inspector

- Initialize inspector agent
- Start the SIGUSR1 watchdog thread (main instance only)

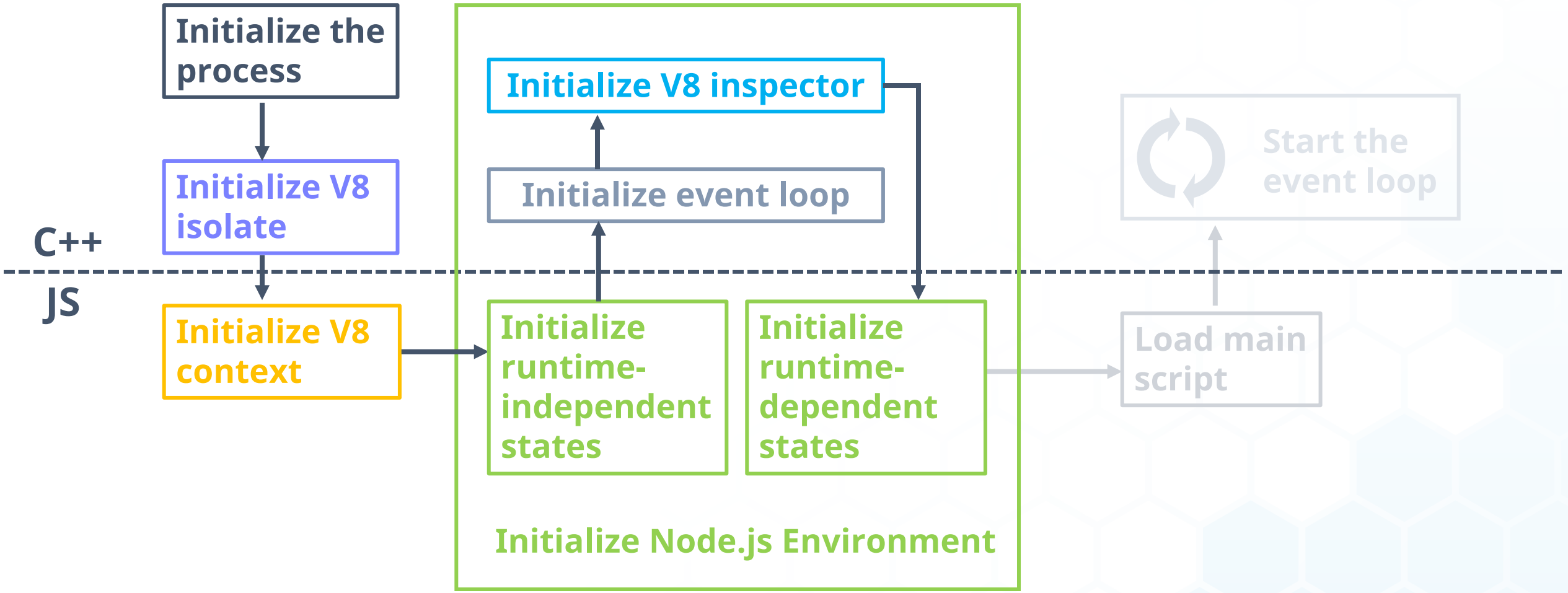


Environment: inspector

- Initialize inspector agent
- Start the SIGUSR1 watchdog thread
- Create more threads for listening on the inspector port and/or profiling, depending on CLI flags
  - `--inspect-brk, --cpu-prof, --heap-prof`



Setting up the Environment



Pre-execution

- Handle various runtime configurations
  - **CLI flags**: e.g. `--no-warnings` , `--experimental-policy`, `--experimental-report`
  - **Environment variables**: e.g. `NODE_DEBUG`, `NODE_V8_COVERAGE`

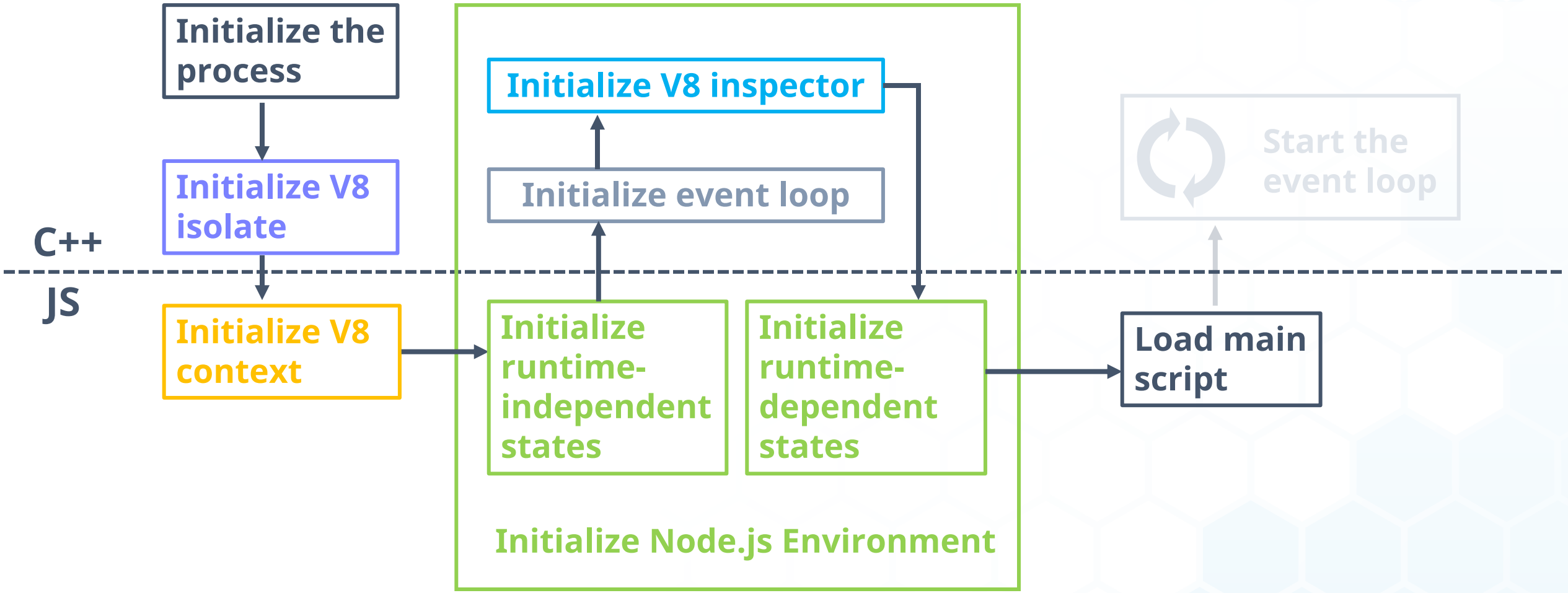
```
const { onWarning } = require('internal/process/warning');
if (!getOptionValue('--no-warnings') &&
  process.env.NODE_NO_WARNINGS !== '1') {
  process.on('warning', onWarning);
}
```

December 11–12, 2019
Montreal, Canada

Pre-execution

- Handle various runtime configurations
- Initialize IPC channel for clusters and child_process
- Initialize user-land module loaders: CJS (`require()`) and ESM (`import`)
- Load `--require` modules

Start execution



Start execution

- Choose a main script according to the CLI args, etc.
  - `lib/internal/main/*.js`
  - Compiled into the binary at build time (similar to internal JS modules)

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 });
```

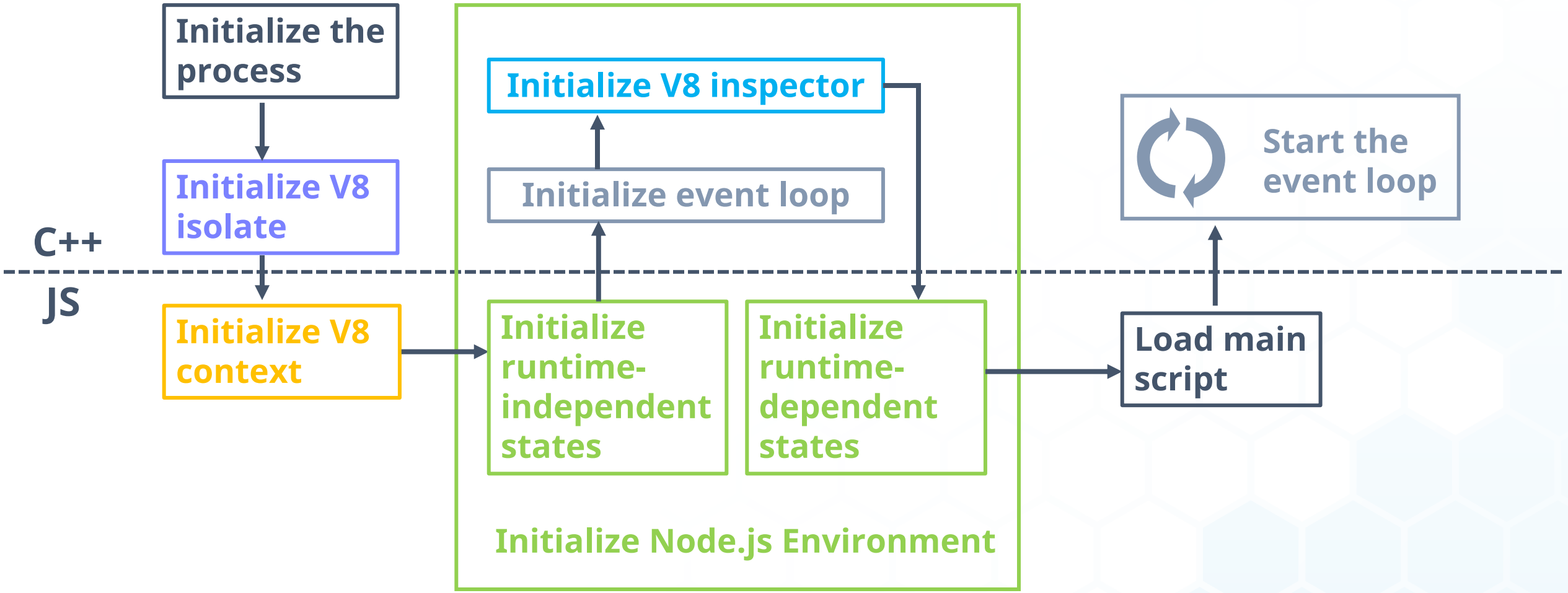
From the worker_thread.js on the worker thread

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

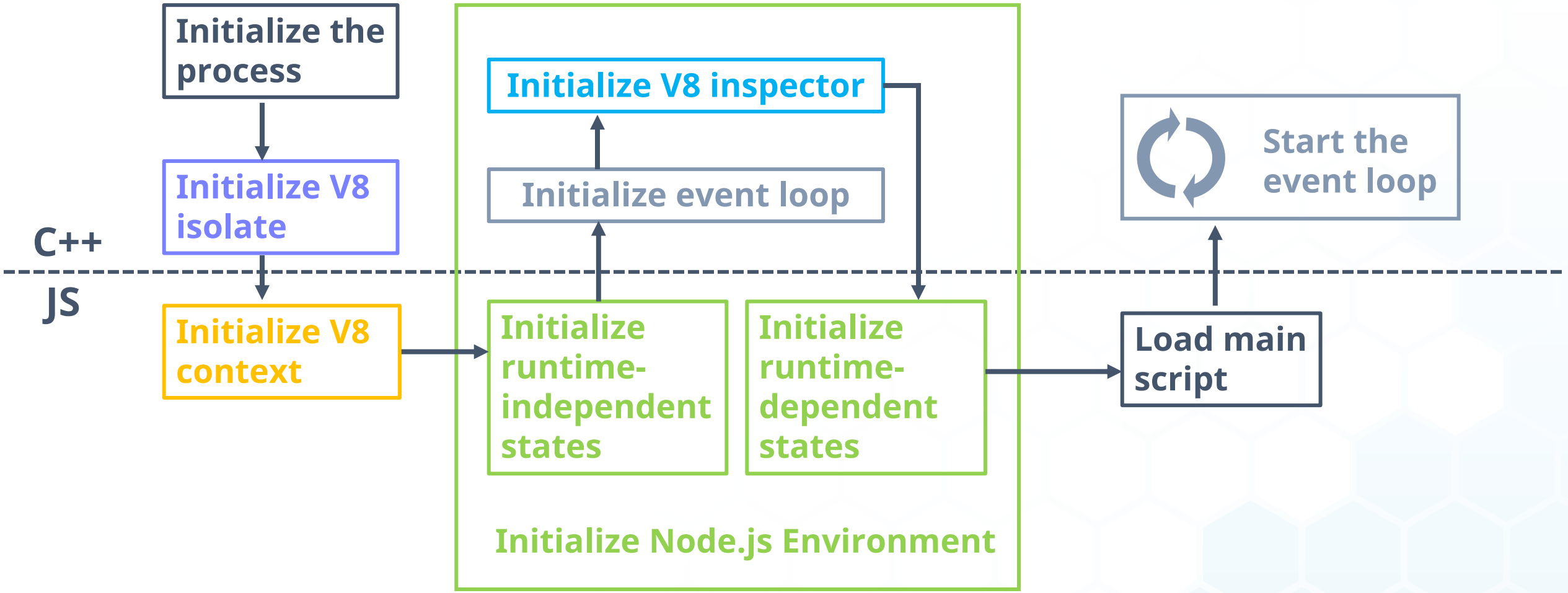
Start execution

- Kick off the event loop and run until nothing keeps it open
  - The libuv thread pool will be created if any asynchronous file system operation is used

Start execution

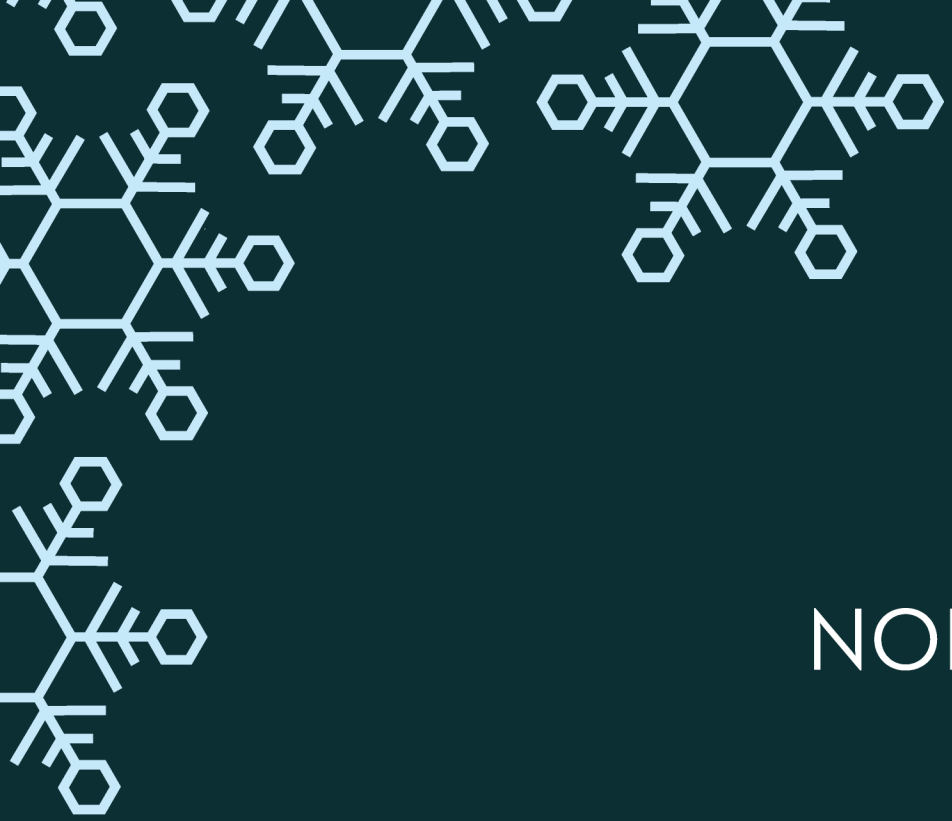


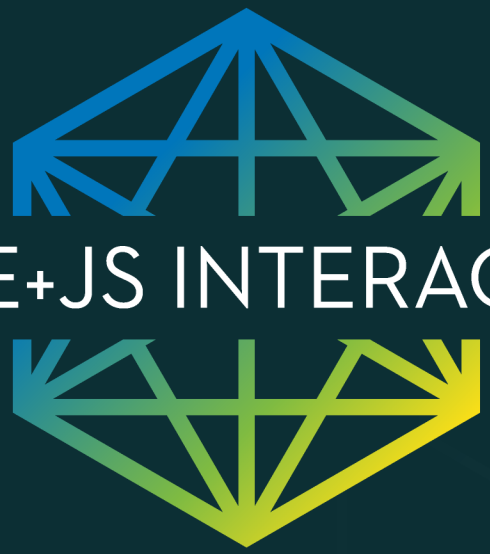
Summary


Initialize the process
Initialize V8 isolate
Initialize V8 context
C++
JS
Initialize V8 inspector
Initialize event loop
Initialize runtime-independent states
Initialize runtime-dependent states
Initialize Node.js Environment
Load main script
Start the event loop

Ongoing work

- V8 startup snapshot integration:
  - Including the runtime-independent part of Environment bootstrap into Nodejs's snapshot
- Startup performance initiative
  - https://github.com/nodejs/TSC/blob/master/Strategic-Initiatives.md





NODE+JS INTERACTIVE

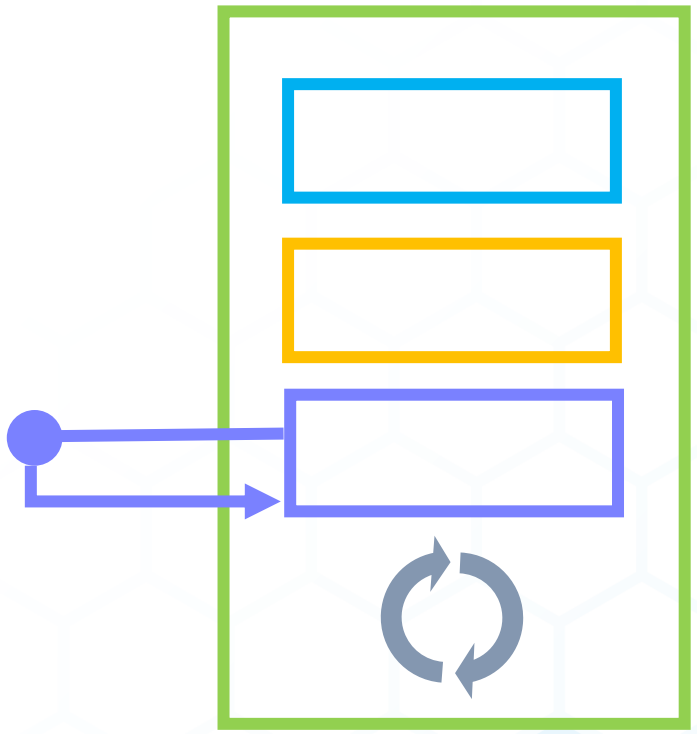
December 11–12, 2019

Montreal, Canada

How do hooks typically work?

If it's a per-Isolate hook (set by V8)...

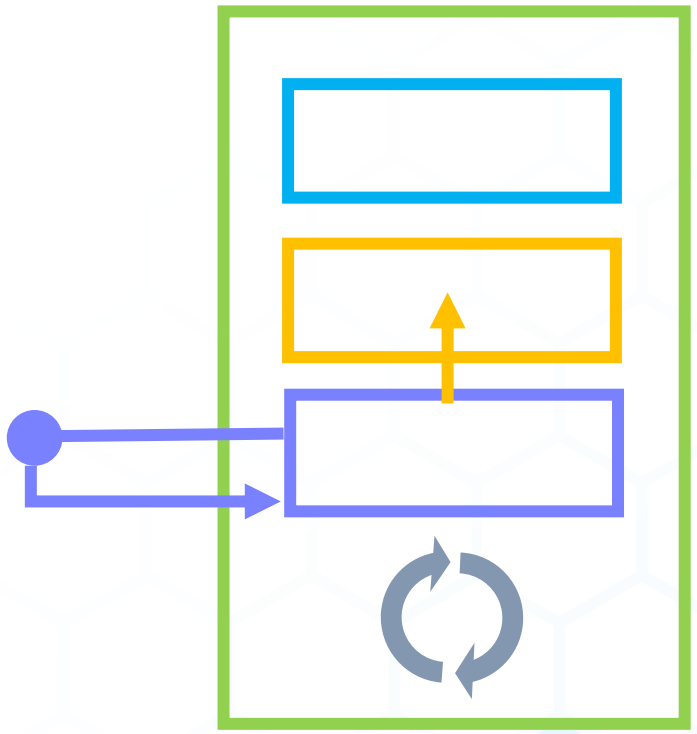
1. Inside the callback, get a pointer to the `v8::Isolate` via e.g. callback arguments



How do hooks typically work?

If it's a per-Isolate hook…

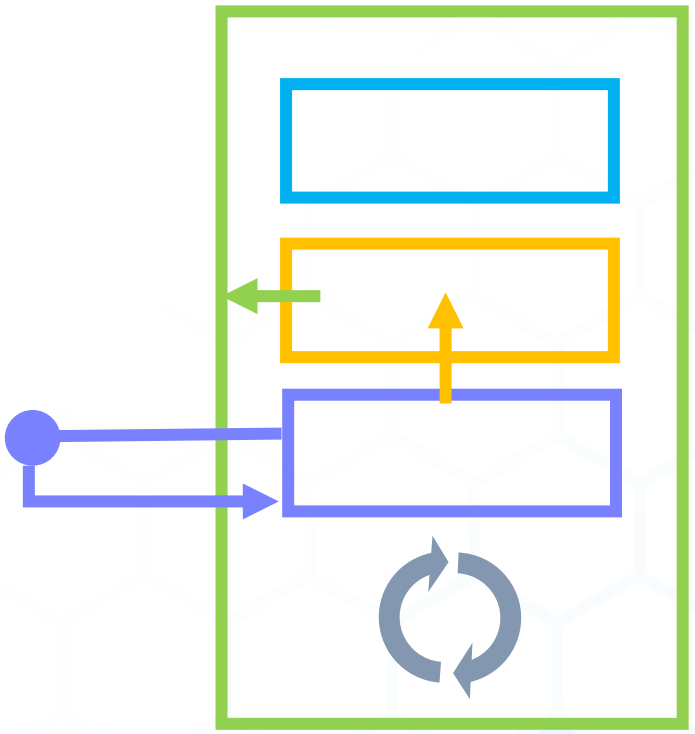
1. Inside the callback, get a pointer to the `v8::Isolate` via e.g. callback arguments
2. Get the `v8::Context` that the isolate enters into



How do hooks typically work?

If it's a per-Isolate hook...

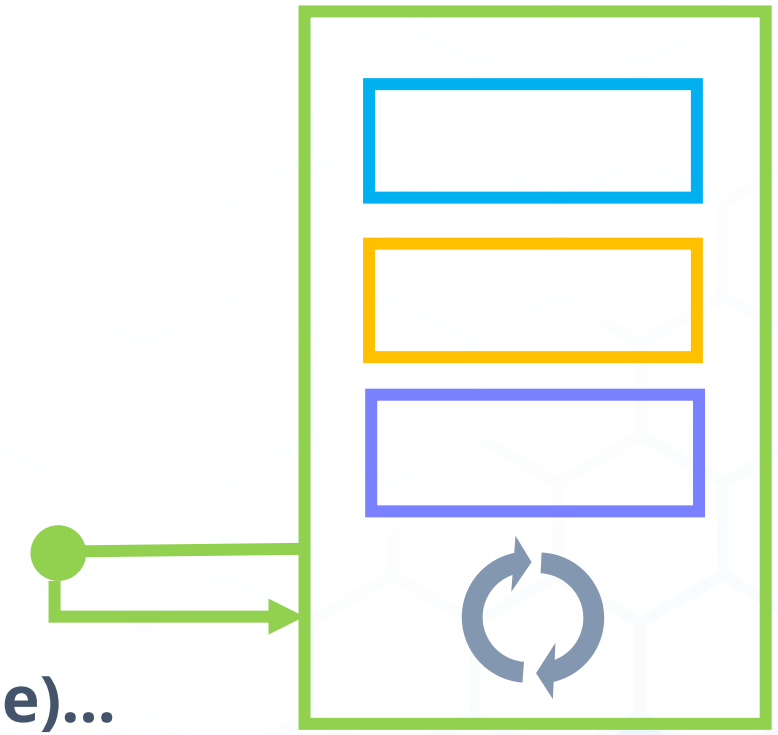
1. Inside the callback, get a pointer to the `v8::Isolate` via e.g. callback arguments
2. Get the `v8::Context` that the isolate enters into
3. Get the pointer to the `node::Environment` embedded in a slot of the context



How do hooks typically work?

If it's a per-Isolate hook...

1. Inside the callback, get a pointer to the `v8::Isolate` via e.g. callback arguments
2. Get the `v8::Context` that the isolate enters into
3. Get the pointer to the `node::Environment` embedded in a slot of the context

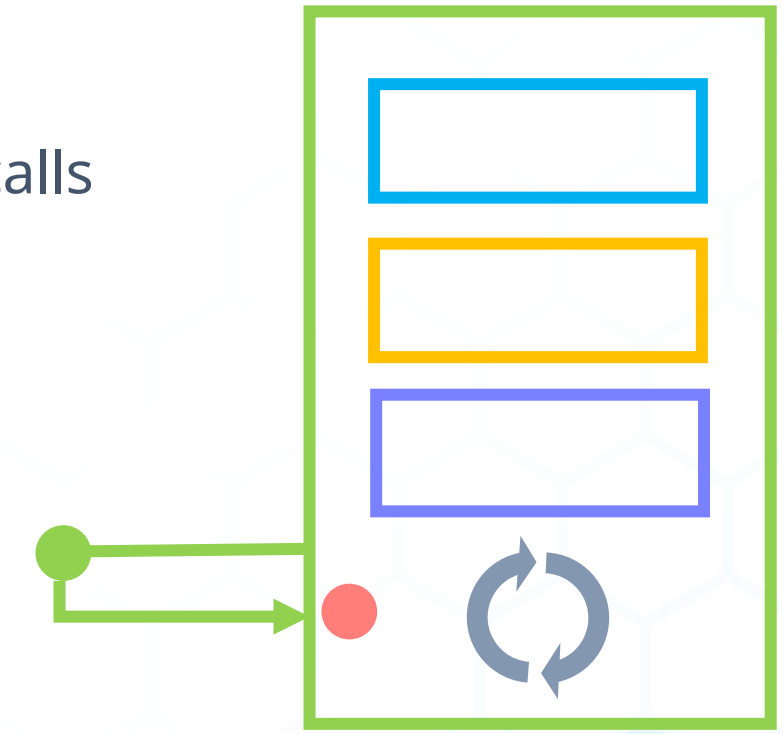


If it's a per-Environment hook (set by node)...
Get access to the `node::Environment` directly via the callback arguments.

How do hooks typically work?

After gaining access to Environment

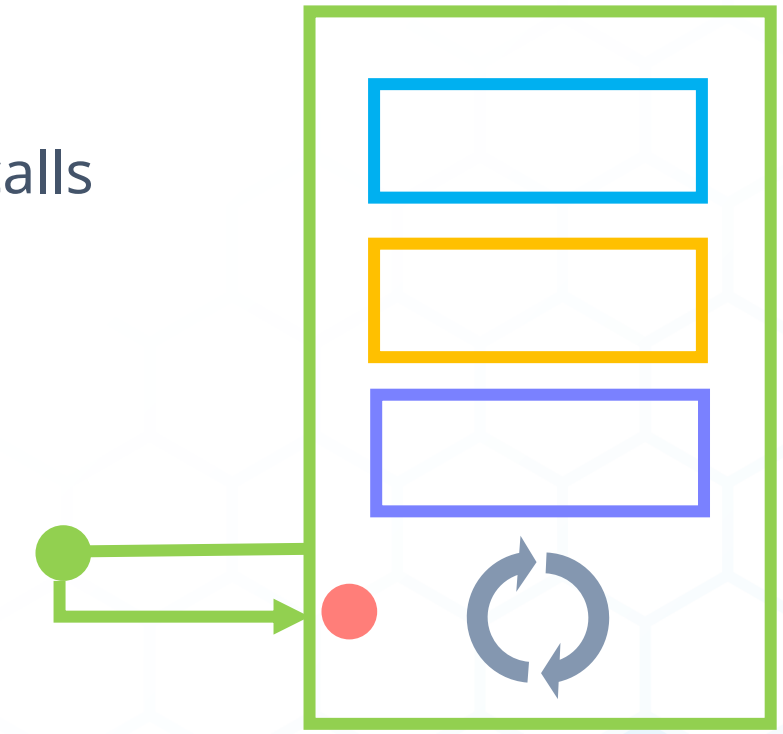
1. Get the internal JavaScript function stored in the `node::Environment` during bootstrap
2. Invoke the internal JavaScript function that calls user-provided callbacks



How do hooks typically work?

After gaining access to Environment

1. Get the internal JavaScript function stored in the `node::Environment` during bootstrap
2. Invoke the internal JavaScript function that calls user-provided callbacks



How are JS functions stored?

```
internalBinding('errors')
  .setPrepareStackTraceCallback(
    require('internal/errors')
      .prepareStackTrace
  );
```