Startup Snapshot in Node.js

Joyee Cheung, 14 May 2023

About me

- Compilers @ Igalia
- Node.js TSC member & V8 committer
- Been working on startup performance → startup snapshot strategic initiative

Challenge of Node.js core startup

- Node.js is growing: increasing number of
  - Globals (in particular Web APIs)
  - Built-in modules and new APIs in existing modules
  - …and generally “things to do to make new things work”: set up handlers, define accessors…
- In v18-v20:
  - Fetch
  - WebCrypto
  - `File` API, `Blob`
  - Web streams: `globalThis.ReadableStream`, `globalThis.CompressionStream`…
  - `util.parseArgs`, `util.MIMEType`…

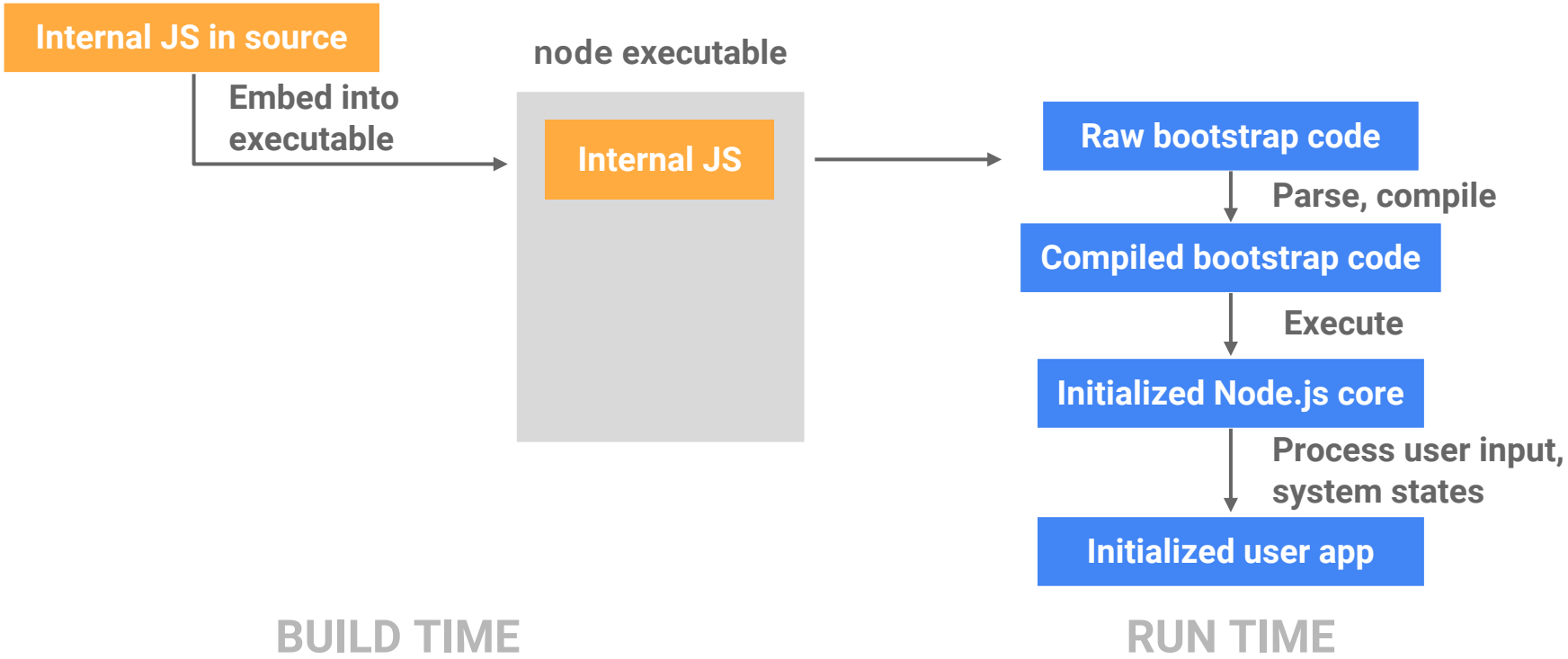
Challenge of Node.js core startup

- The Node.js core is roughly **half in JavaScript**, half in C++ (and some others)
- **Pros**
  - Lower the contribution barrier
  - Reduce some C++ → JavaScript callback costs
- **Cons**
  - Need to parse and compile JavaScript
  - Overall initialization speed is affected - JS code that only gets run once don't get optimized
  - Need to defend against prototype pollution and monkey patching: copying JS built-ins at startup to avoid blowups caused by e.g. `delete String.prototype.startsWith`

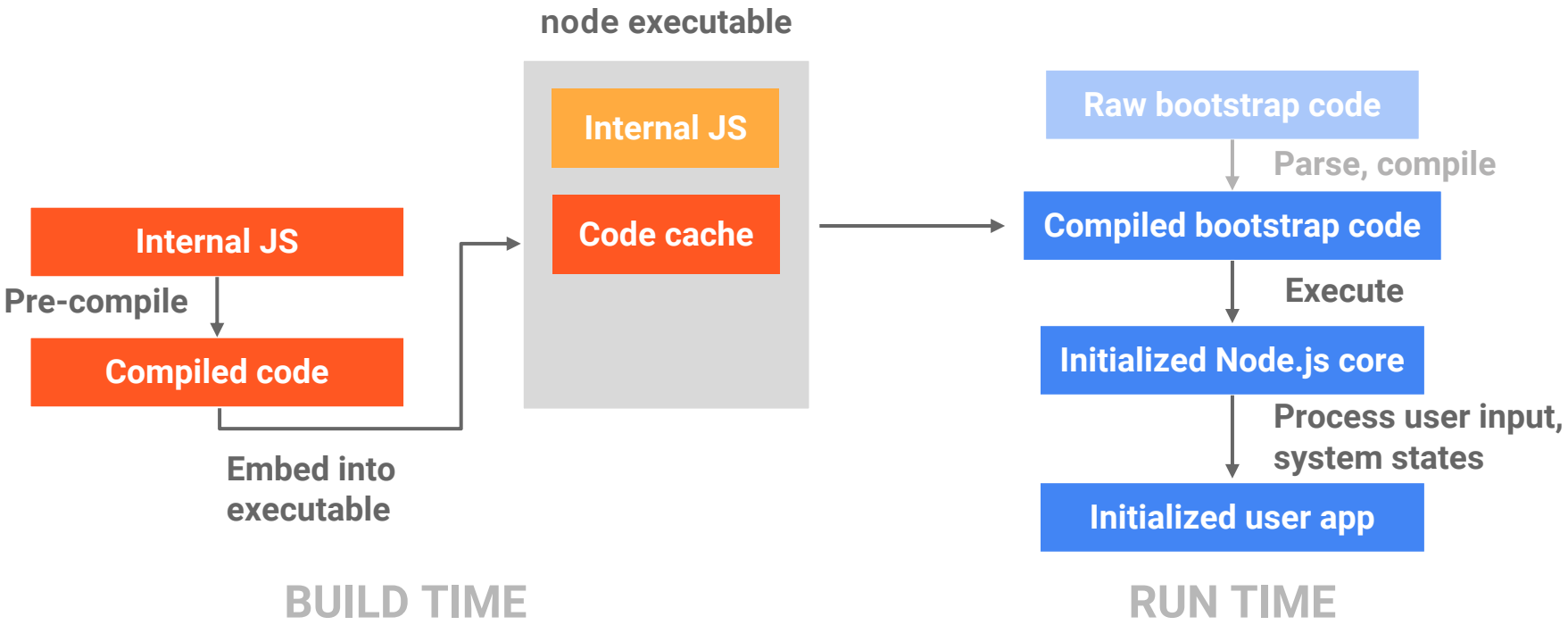
Trying to keep the startup under control

1. **Lazy loading:** do not load non-essential features until it’s actually used
   - e.g. crypto, performance timing, messaging
2. **Code cache:** when loading additional features (built-in modules)
   - Bytecode and metadata etc.
   - Compiled and embedded into the executable at build time.
   - Skips compilation when validation passes.
3. **V8 startup snapshot:** pre-initialize features that are almost always loaded / essential initializations
   - URL, fs, etc…used by other internals
   - Buffer, setTimeout, etc.: widely used
   - Skips execution of the initialization code

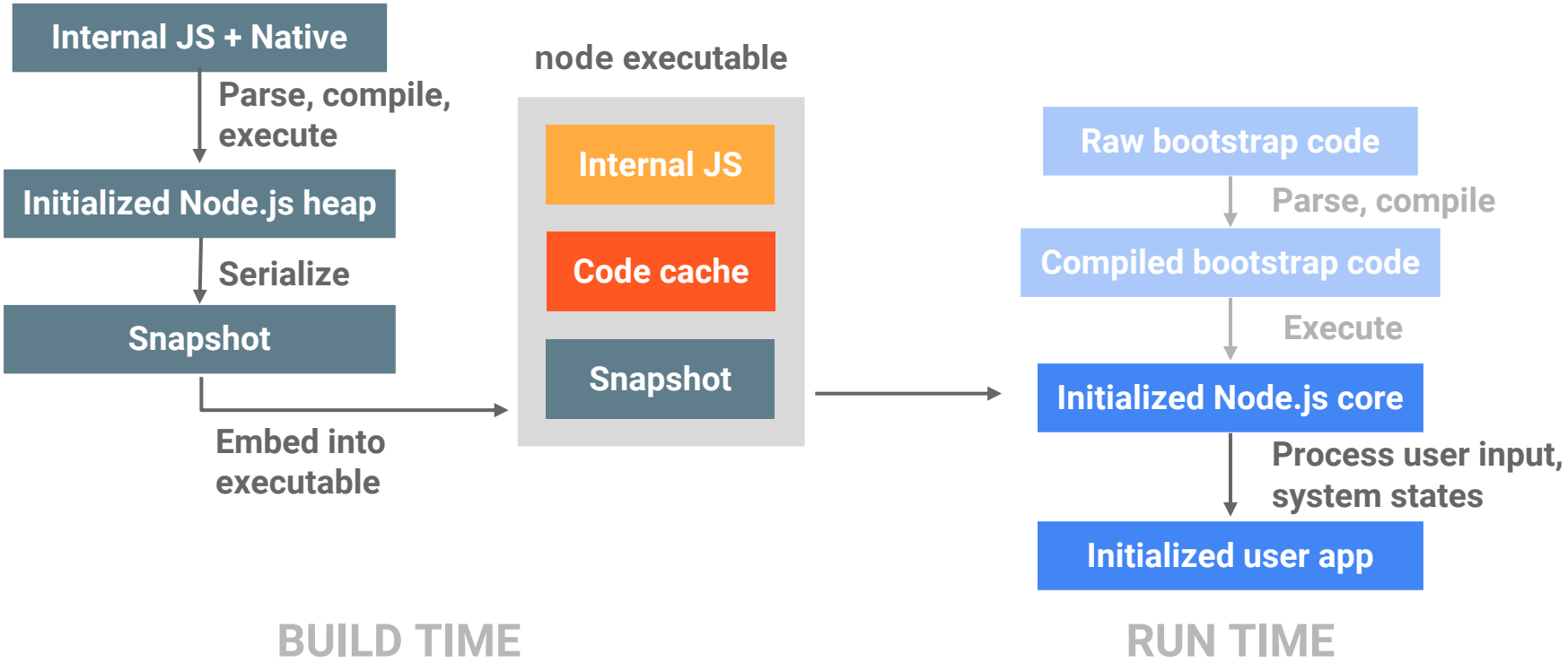
Startup snapshots within Node.js core



Startup snapshots within Node.js core



Startup snapshots within Node.js core

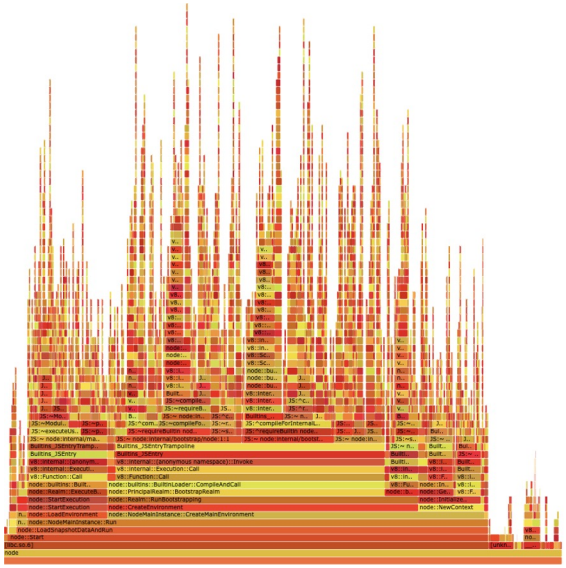


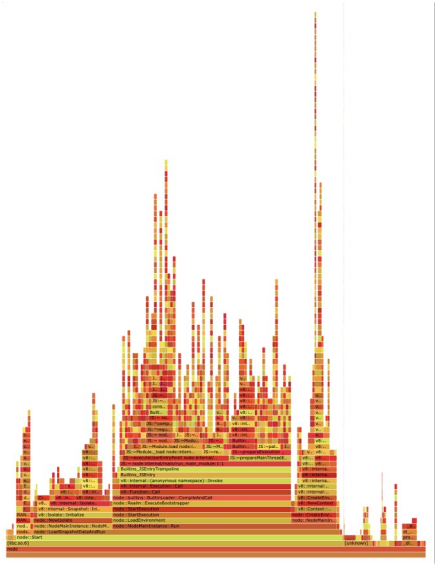
What are V8 startup snapshots?

- V8 heap states serialized into a binary blob
- **Isolate snapshot**: shared by main instance and workers
  - Primitives: strings, symbols, etc.
  - Native bindings
- **Context snapshot**: main context, vm contexts, worker context (minimal)
  - Execution context
  - Globals
  - Objects
  - Functions

Startup snapshots within Node.js core

- The default startup (with snapshot) is generally 2x faster than startup with `--no-node-snapshot`: ~40ms -> ~20ms
- A sustainable path for growing core & keeping startup under control





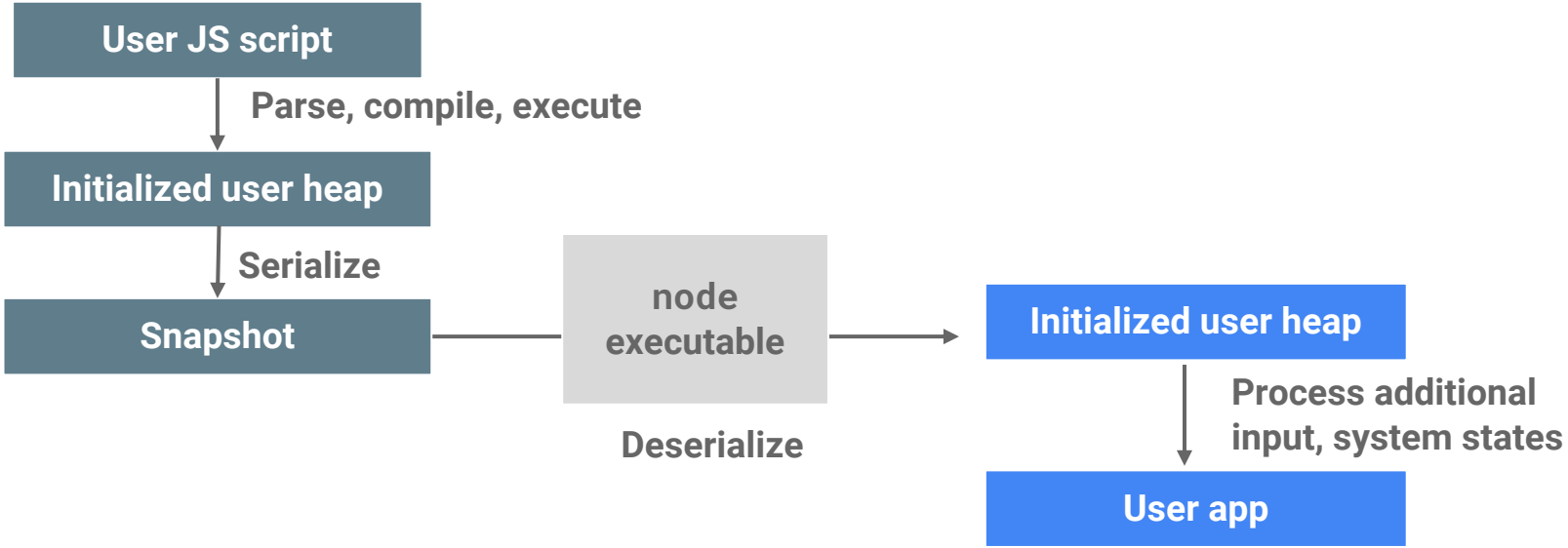
User-land startup snapshots

Creating snapshots from user application code, useful if

- Startup performance matters e.g. in CLI tools
- A lot of code needs to be compiled/run during startup
- A lot of system-independent data needs to be loaded during startup

User-land startup snapshots

- Currently requires the snapshot script to be a one-file bundle
  - User-land module support is WIP
- Run-to-completion: until async operations are finished, promises are resolved



User-land startup snapshots

Build-time generation, embedded binary: from v17.9.0

- Building Node.js from source and embedding the snapshot into the binary

```
$ echo 'globalThis.data = "hello"' > snapshot.js
$ cd /path/to/node
$ ./configure --node-snapshot-main=snapshot.js && make node
$ out/Release/node # globalThis.data contains "hello"
```

User-land startup snapshots

Run-time generation, separate binary: from v18.8.0

- Use the default Node.js binary to write snapshot to a separate blob for loading later

```
$ node --snapshot-blob snapshot.blob --build-snapshot snapshot.js
$ node --snapshot-blob snapshot.blob # deserialize snapshot
```

User-land startup snapshots

Runtime generation, embedded binary: in v20.?

- Layer on top of single-executable application (work in progress)
- Use the default Node.js binary to generate a blob which includes the snapshot (and more), and inject it into one single executable
- No need to compile Node.js from source
- A single-line utility command to come

```
$ echo '{"snapshot_main":"sea.js","output":"sea.blob"}' > sea.json
$ node --experimental-sea-config sea.json
$ cp node sea
$ npx postject sea NODE_SEA_BLOB sea.blob --sentinel-fuse \
  $NODE_SEA_FUSE
$ ./sea  # contains snapshot
```

JS API: Synchronizing run-time states

- Node.js refreshes `process.env` and `process.argv` etc. when the snapshot is deserialized
- States computed from system states can be synchronized during deserialization.

```
let debug_level = 0;
function computeDebugLevel() {
   switch (process.env.DEBUG_LEVEL) {
       case 'none': debug_level = 0; break;
       case 'debug': debug_level = 1; break;
   }
}
```

JS API: Synchronizing run-time states

```
const {
 addSerializeCallback, addDeserializeCallback, isBuildingSnapshot
} = require('v8').startupSnapshot;

// Usual startup
computeDebugLevel();

// Snapshot synchronization
if (isBuildingSnapshot()) {
 addSerializeCallback(() => { debug_level = 0;  /* reset */ })
 addDeserializeCallback(computeDebugLevel);  /* re-compute */
}

// Or, defer the computation until deserialization if building snapshot
if (!isBuildingSnapshot()) { computeDebugLevel(); }
else { addDeserializeCallback(computeDebugLevel); }
```

JS API: configure main function

```
// In the snapshot:
const greetings = { en_US: 'hello', zh_CN: '你好', es_ES: 'hola' };
```

1. Pass a separate main script that does the logging

```
$ echo "console.log(greetings[process.env.LANGUAGE])" > hello.js
$ LANGUAGE=en_US node --snapshot-blob snapshot.blob hello.js # logs "hello"
```

2. Configure the main function in the same snapshot script

```
require('v8').startupSnapshot.setDeserializeMainFunction(() => {
  console.log(greetings[process.env.LANGUAGE]);
});
```

```
$ LANGUAGE=en_US node --snapshot-blob snapshot.blob  # logs "hello"
```

Summary

- Startup snapshot has been integrated into Node.js core to speed up core startup
- Experimental user-land snapshot support is now available, with JS APIs in `v8.startupSnapshot`
- Support for single executable applications and more features is WIP

Thanks

- @addaleax, @cjihrig, @jasnel, @legendecas, @RaisinTen, et al.
- Bloomberg & Igalia