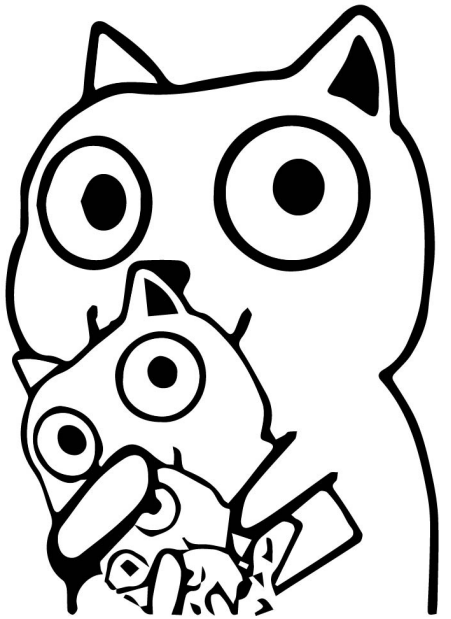
Bringing JavaScript Back to Life

Joyee Cheung, Igalia

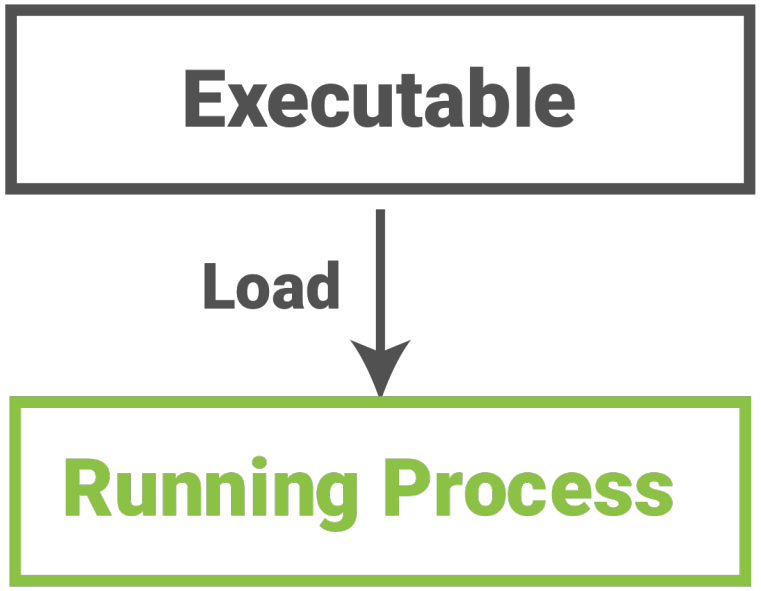
About Me

- Joyee Cheung / Qiuyi Zhang (张秋怡)
- Based in 杭州 Hangzhou, China
- Compilers team @ Igalia
- Node.js TSC & Diagnostics WG
- joyeecheung @ GitHub/Twitter

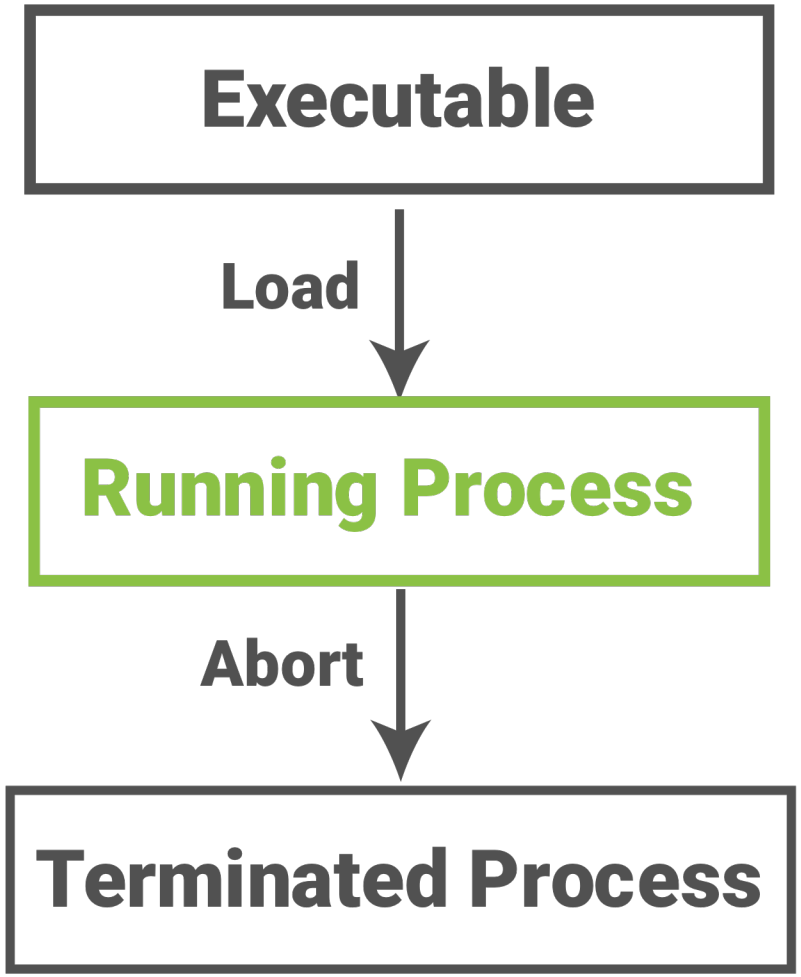


Introduction to Post-Mortem Diagnostics

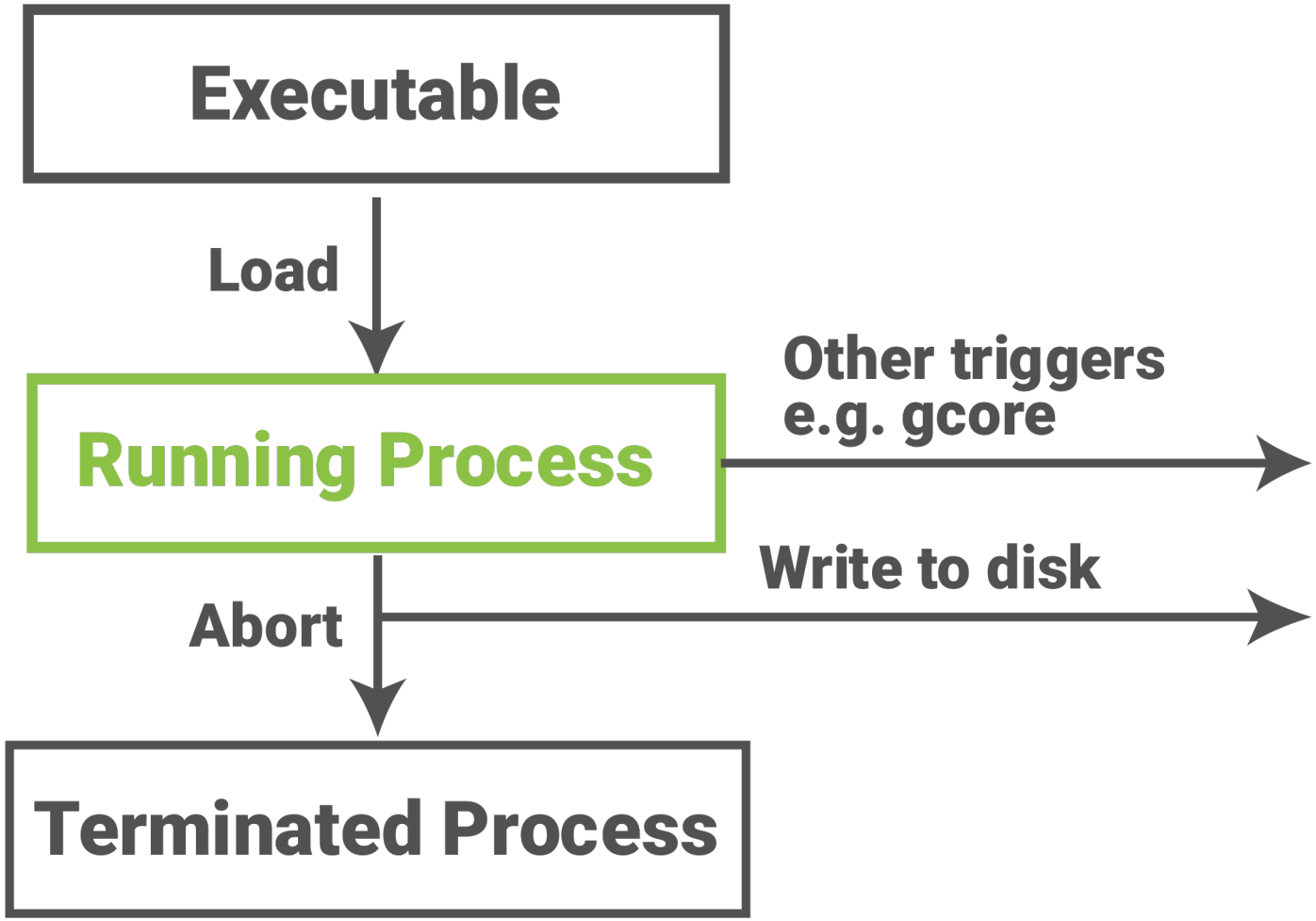
Introduction to Post-Mortem Diagnostics



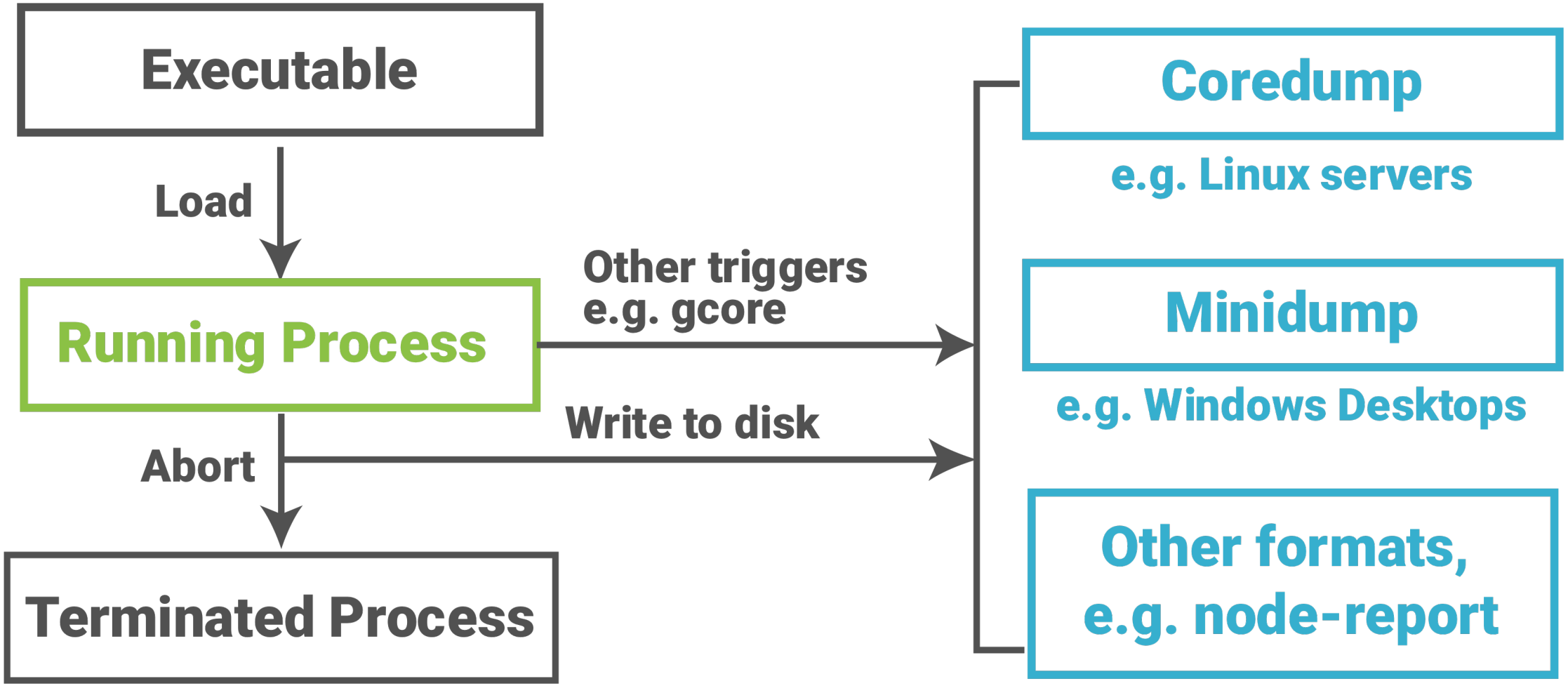
Introduction to Post-Mortem Diagnostics



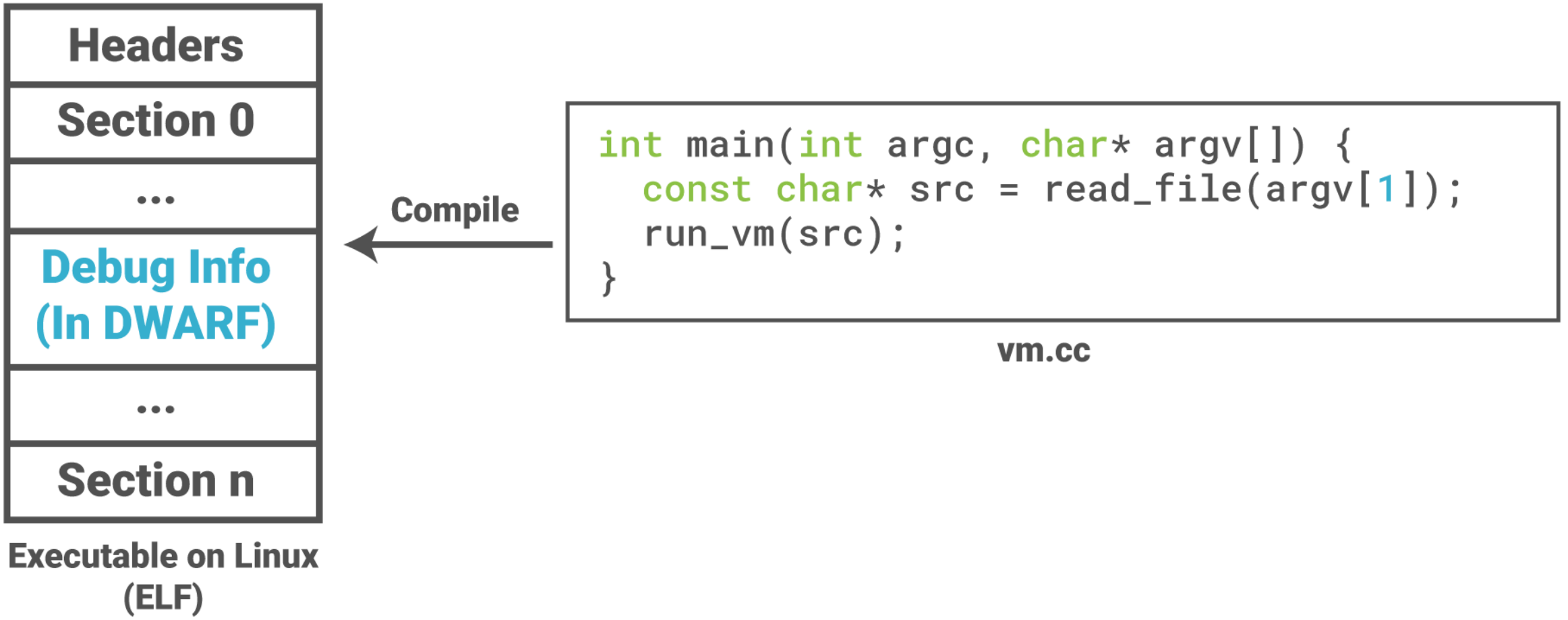
Introduction to Post-Mortem Diagnostics



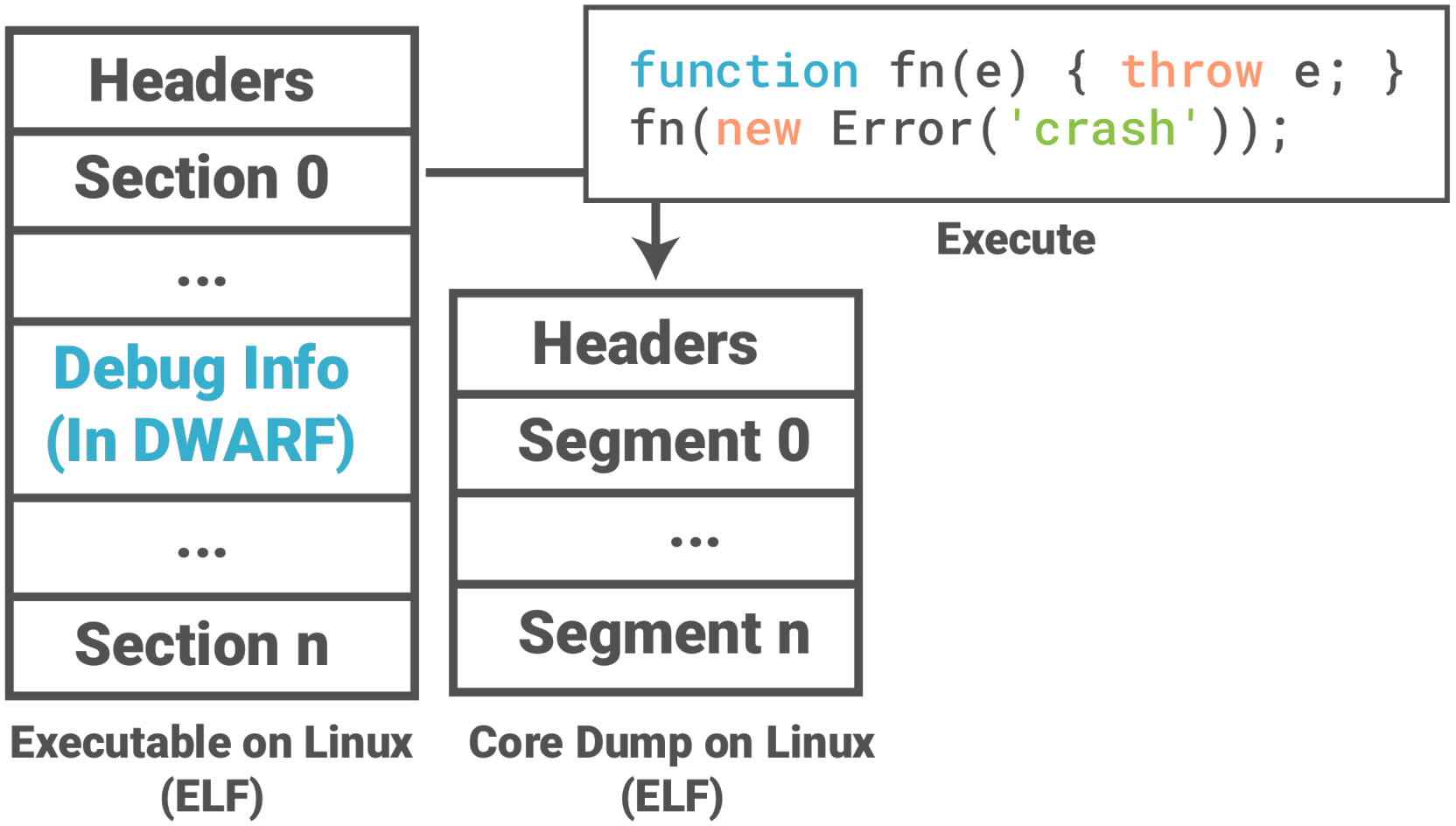
Introduction to Post-Mortem Diagnostics



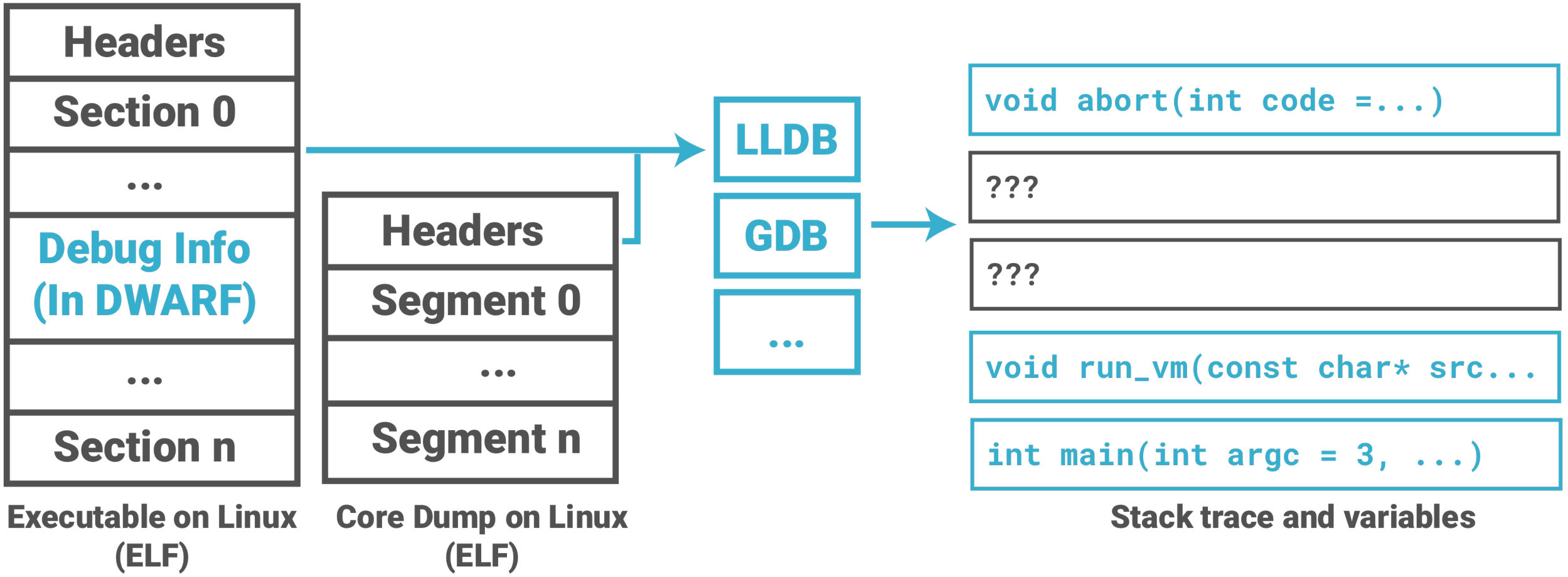
Analysis of a Core Dump



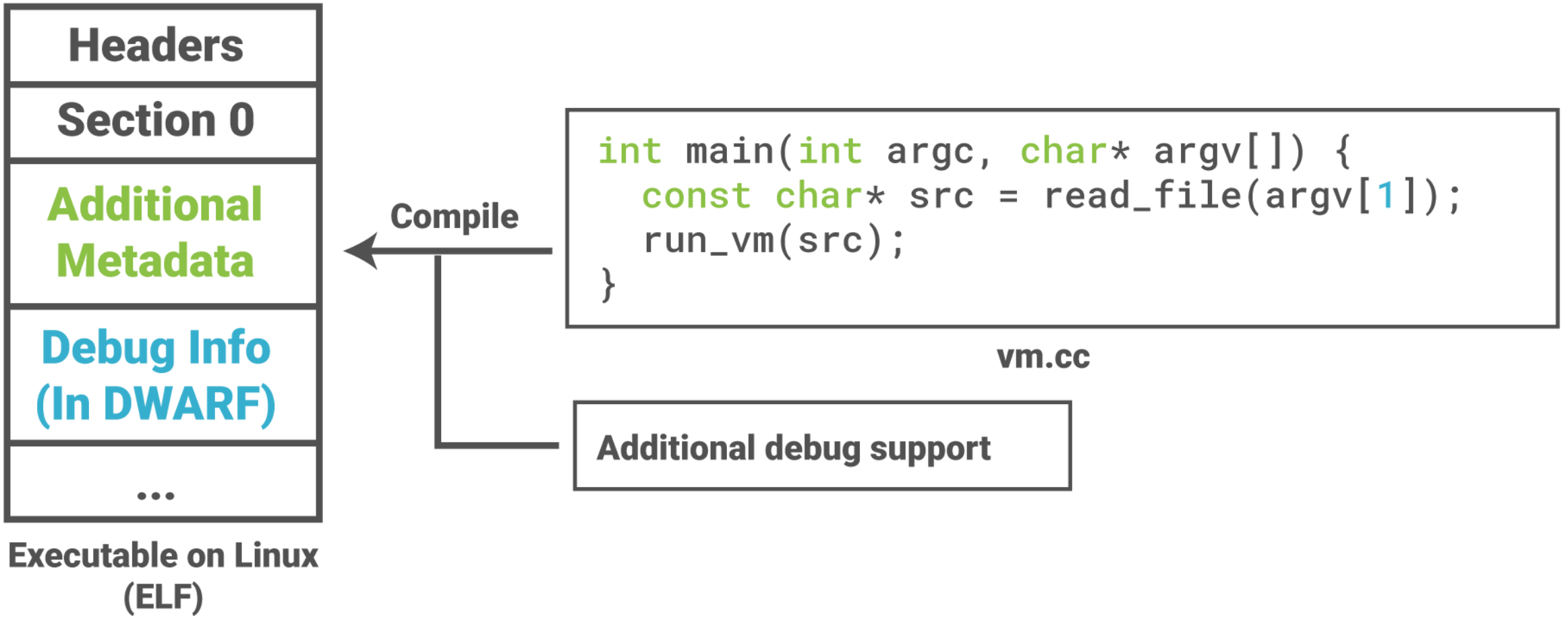
Analysis of a Core Dump



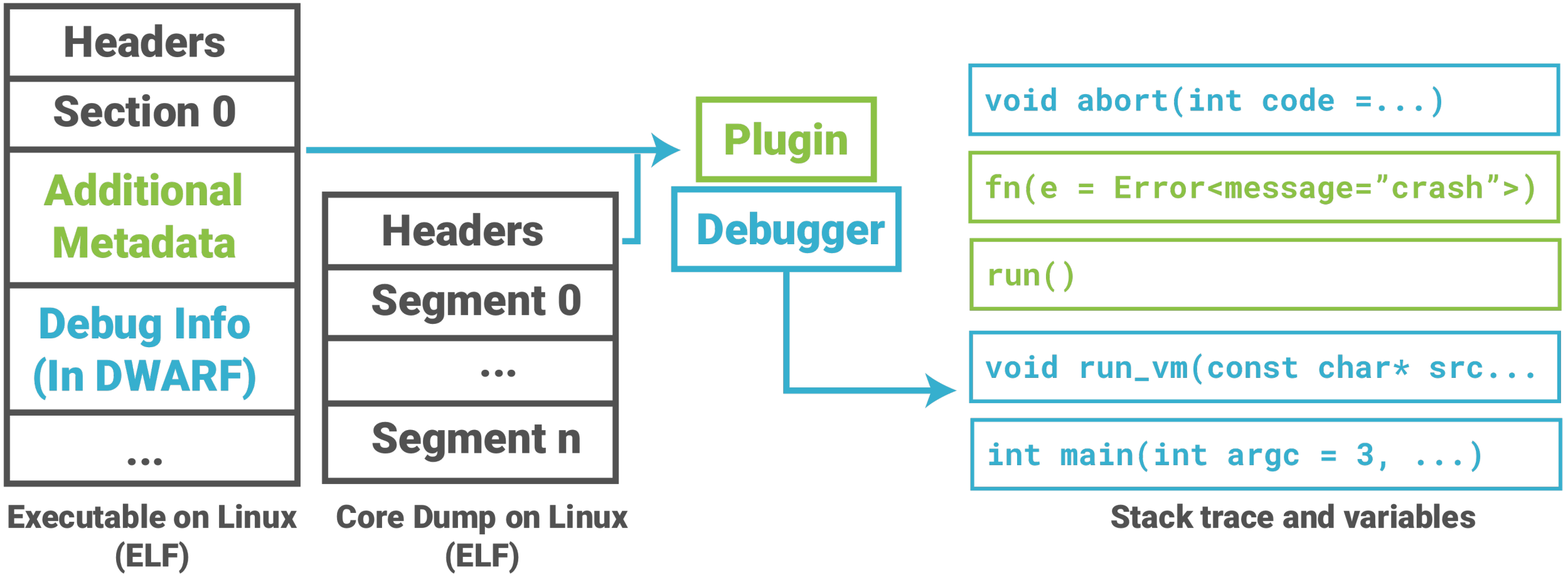
Analysis of a Core Dump



Recovering States from Dynamic Language VMs



Recovering States from Dynamic Language VMs



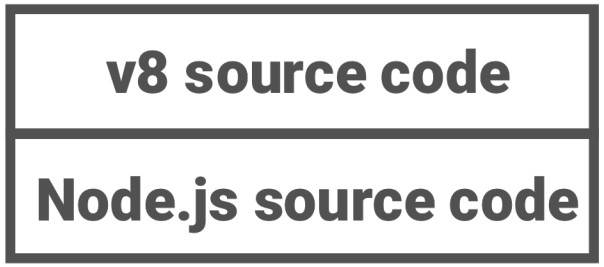
A Tour of llnode

- https://github.com/nodejs/llnode
- Project under the Node.js Diagnostics Working Group
- Plugin of the LLDB debugger

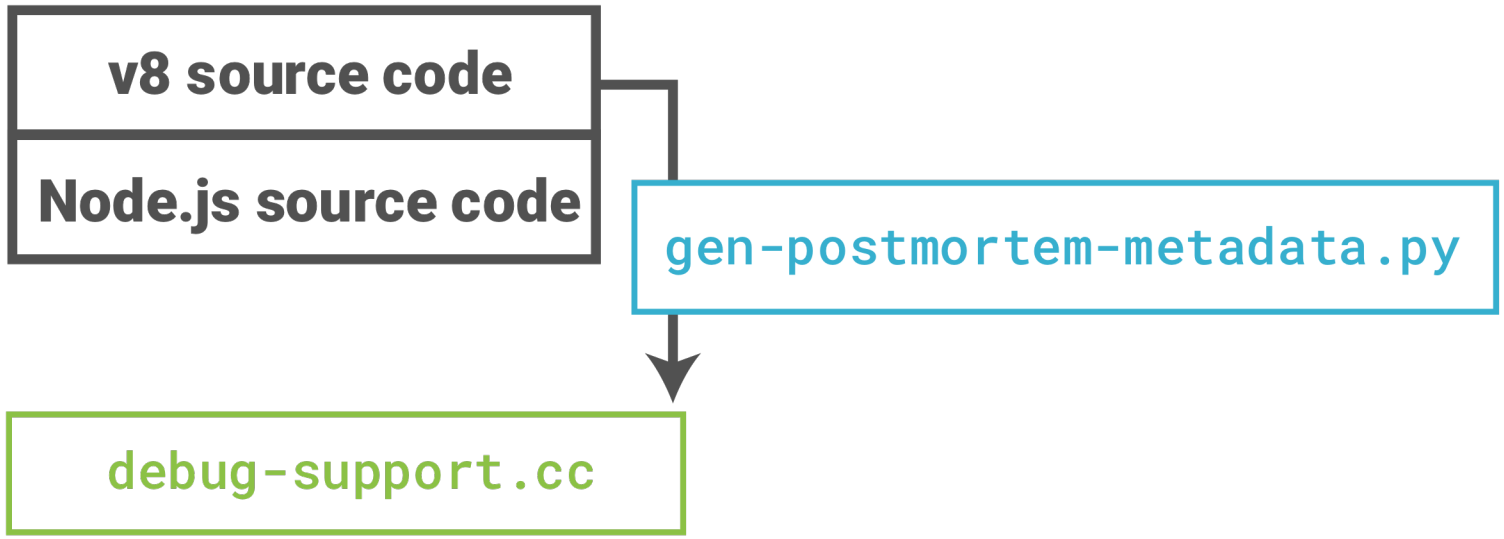
A Tour of llnode

- Can be used to debug
  - Node.js Core dumps
  - Live processes of Node.js applications
- Also works with
  - Other programs embedding v8 built with post-mortem debugging support
  - e.g. d8 shell

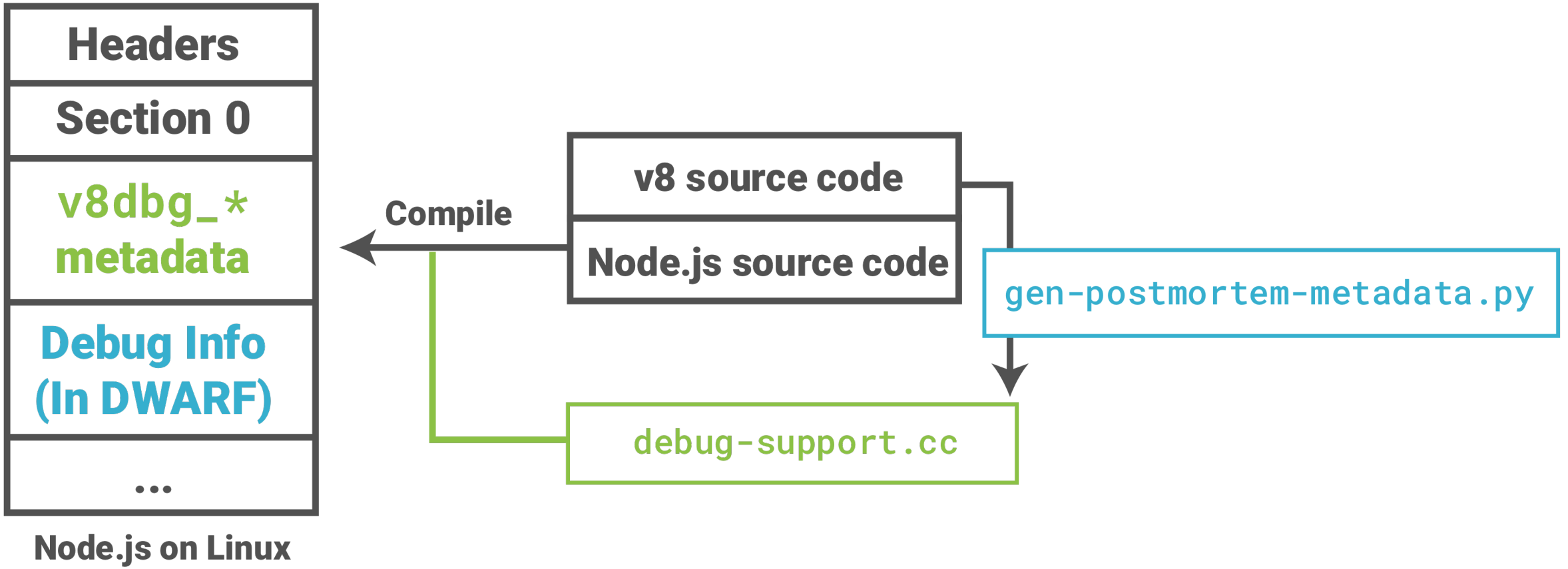
A Tour of llnode



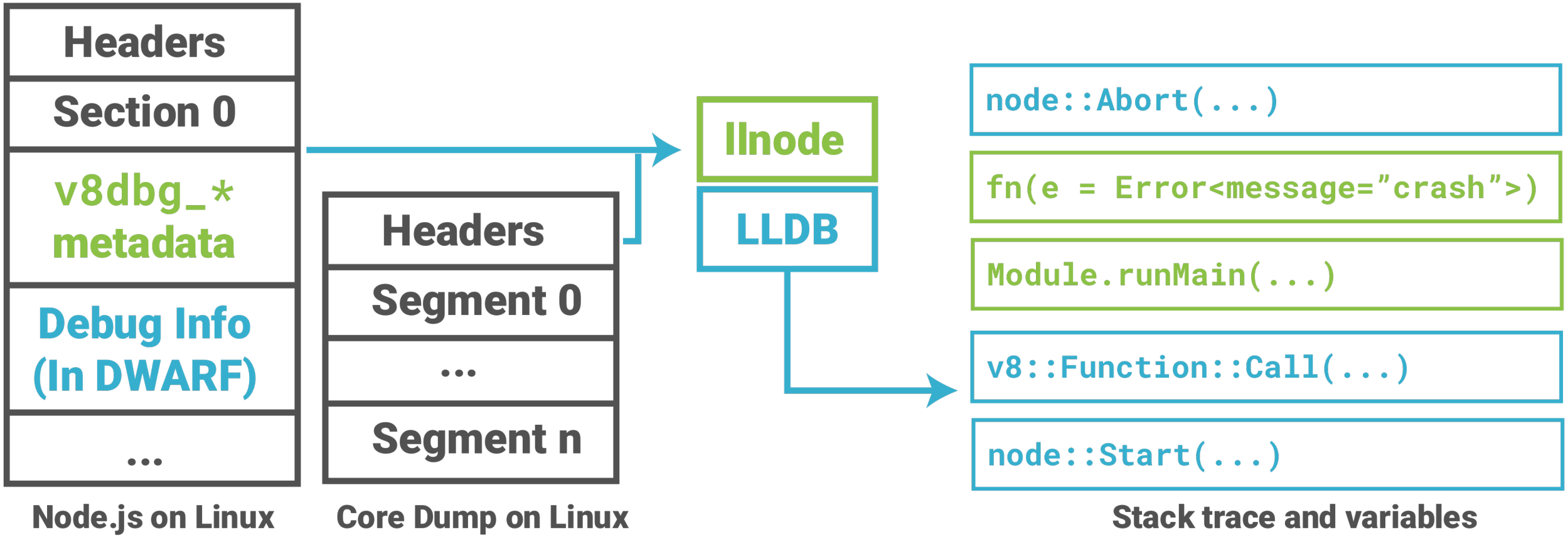
A Tour of llnode



A Tour of llnode



A Tour of llnode



A Tour of llnode

1. (Maintainers) read the V8 source code for reverse-engineering **algorithms**

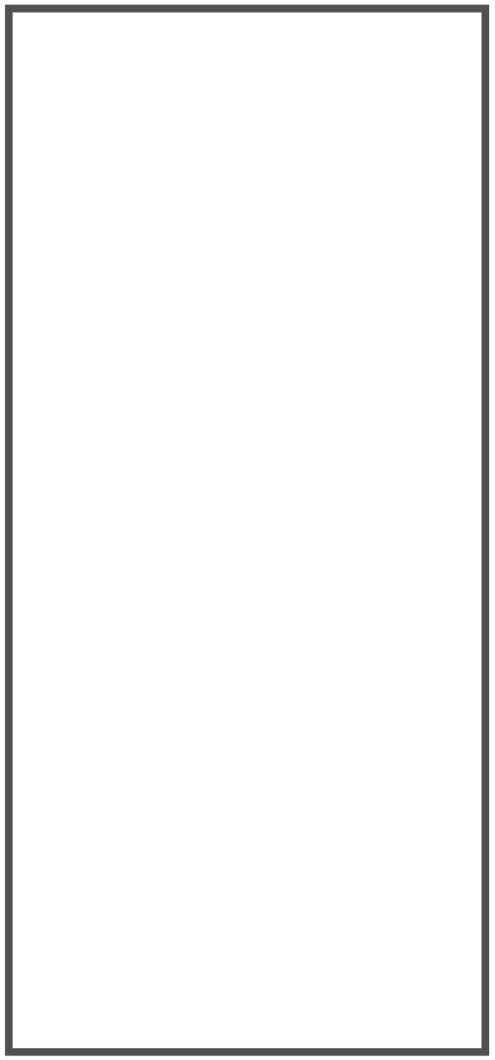
A Tour of llnode

1. (Maintainers) read the V8 source code for reverse-engineering **algorithms**
2. Load the exact values of **v8dbg_*** **offsets/constants** from an executable embedding V8 e.g. Node.js
   - If the necessary metadata is not exposed, submit a patch to the upstream

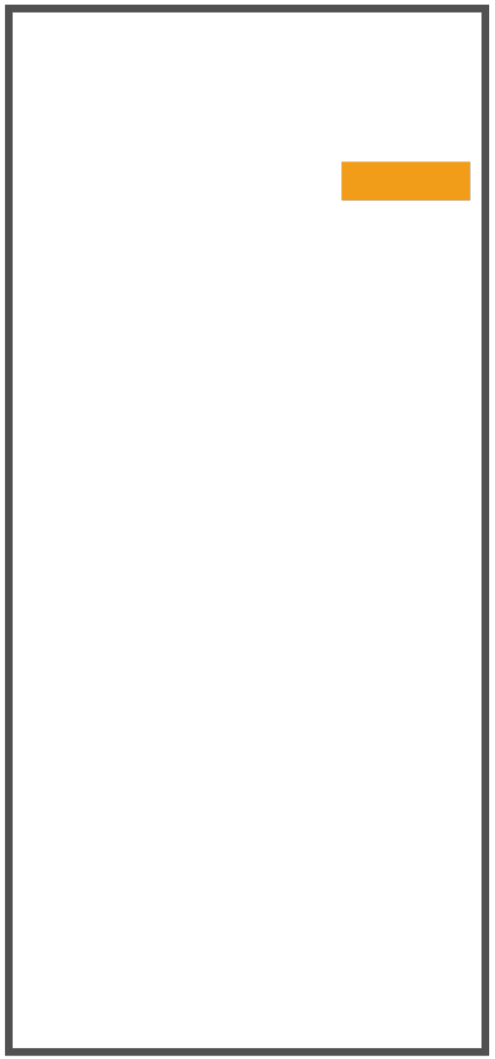
A Tour of llnode

1. (Maintainers) read the V8 source code for reverse-engineering **algorithms**
2. Load the exact values of **v8dbg_*** **offsets/constants** from an executable embedding V8 e.g. Node.js
   - If the necessary metadata is not exposed, submit a patch to the upstream
3. Use the algorithm and the metadata to **interpret memory blocks** in the core dump
   - Infrastructures and cross-platform support come from the LLDB API

Reconstruct JS Values from Raw Memory

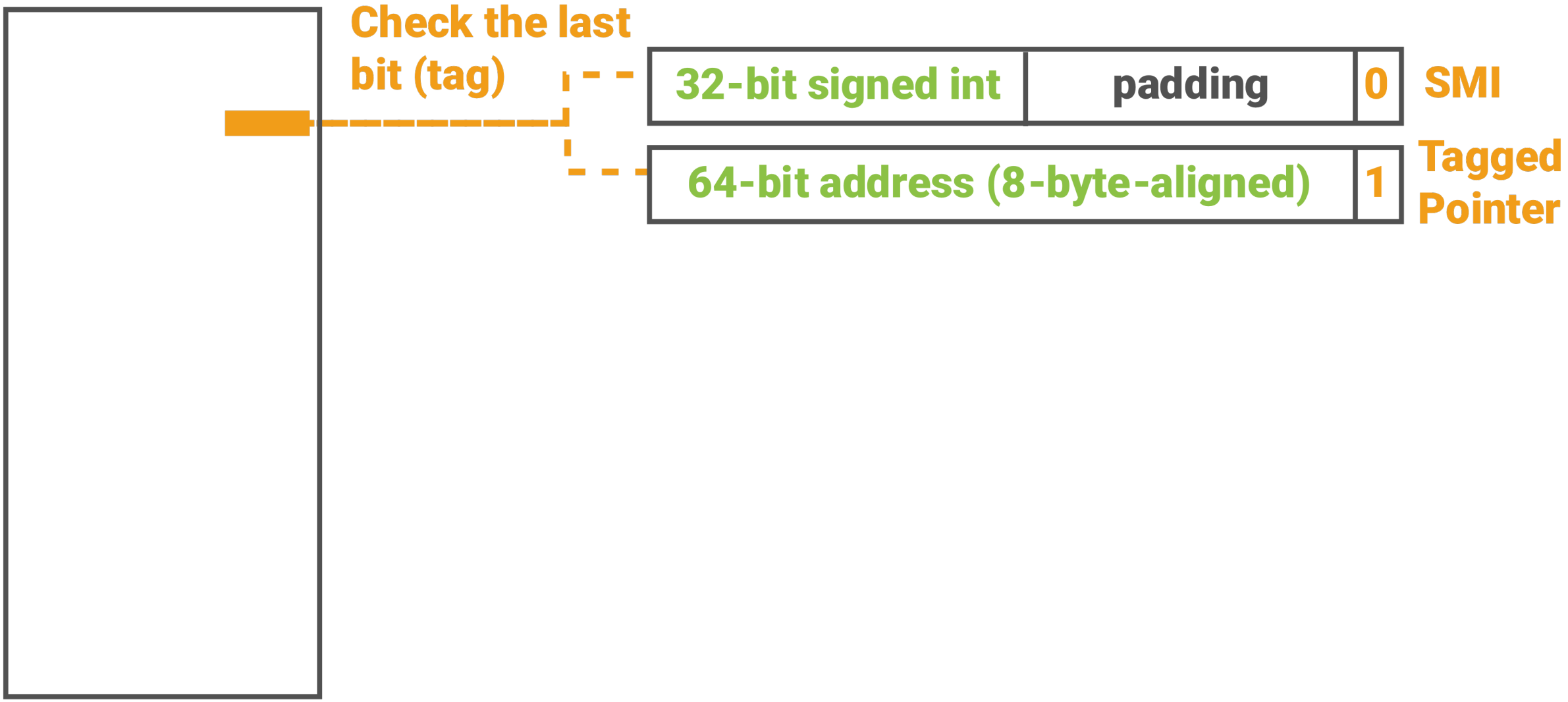


Reconstruct JS Values from Raw Memory

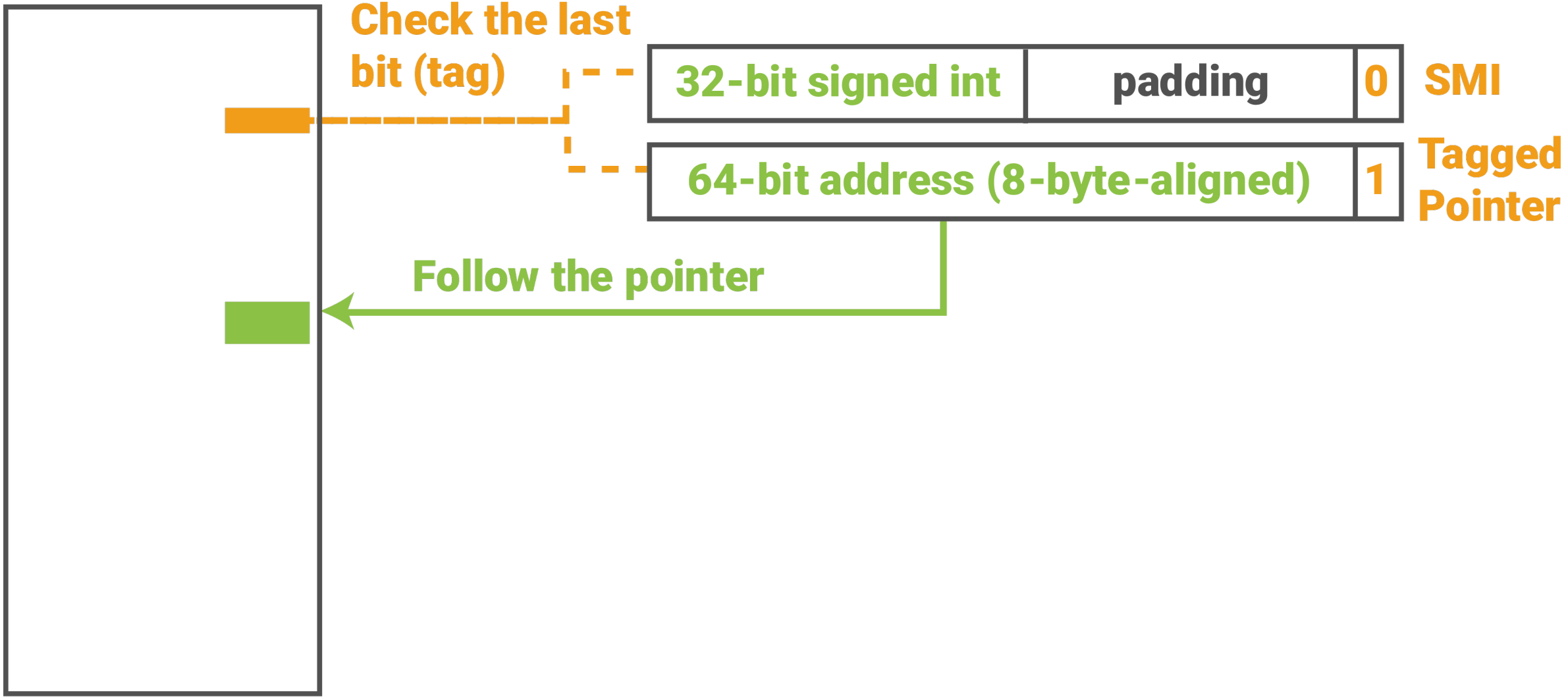


Check the last bit (tag)

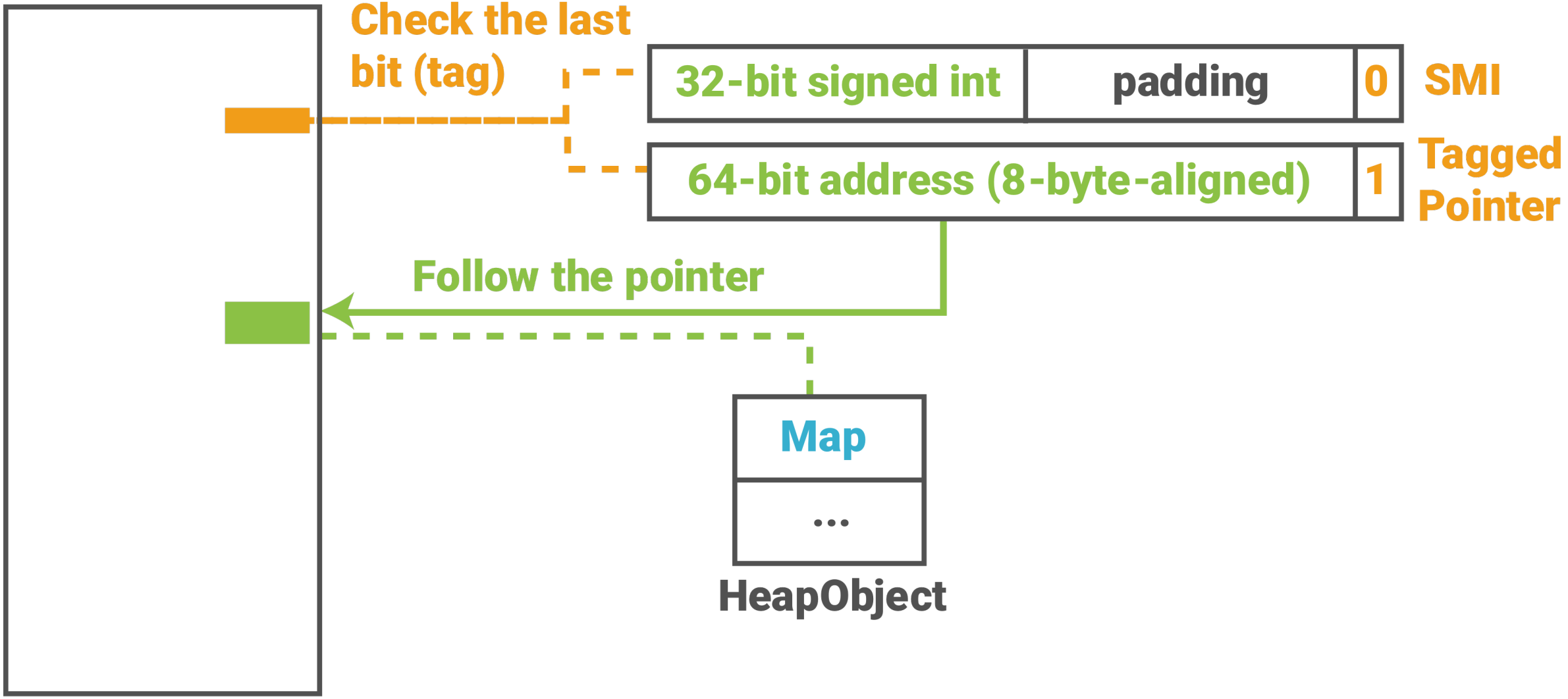
Reconstruct JS Values from Raw Memory



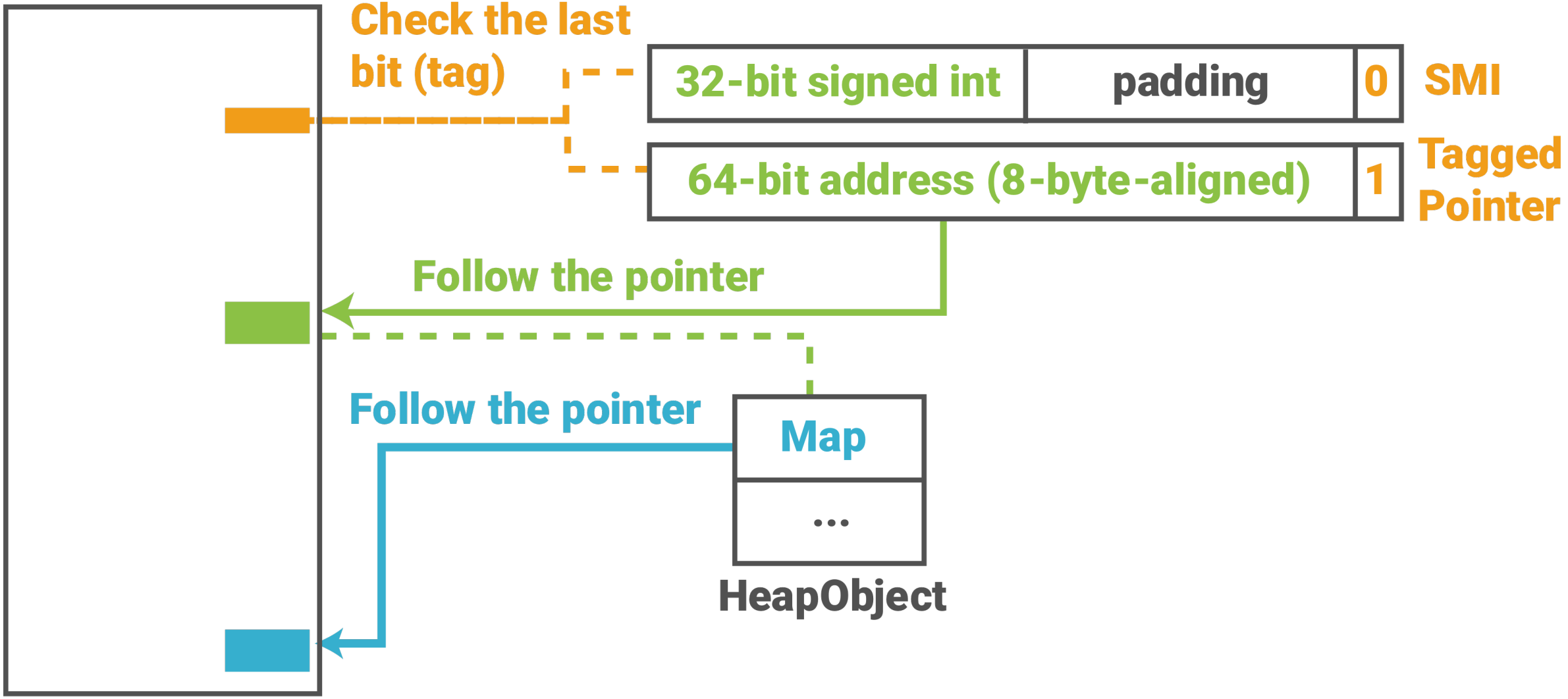
Reconstruct JS Values from Raw Memory



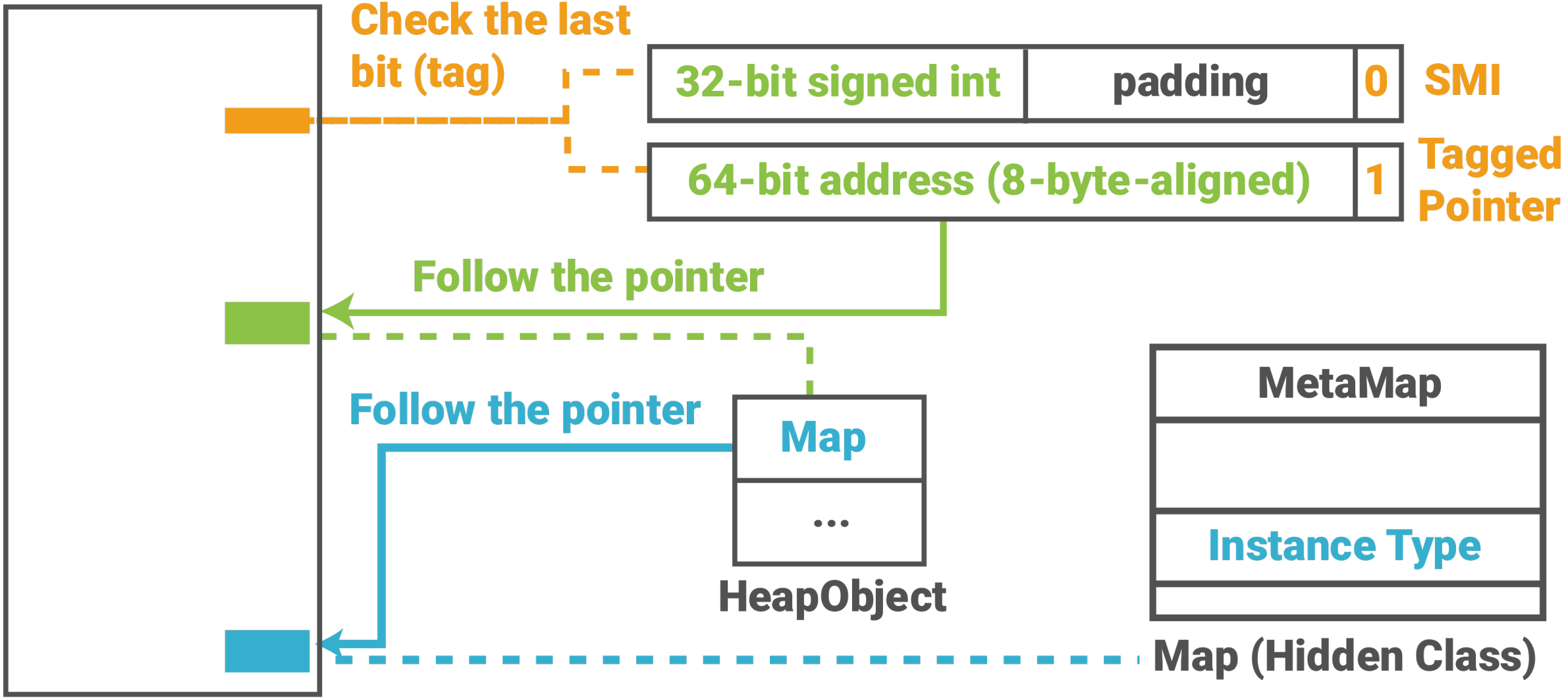
Reconstruct JS Values from Raw Memory



Reconstruct JS Values from Raw Memory



Reconstruct JS Values from Raw Memory



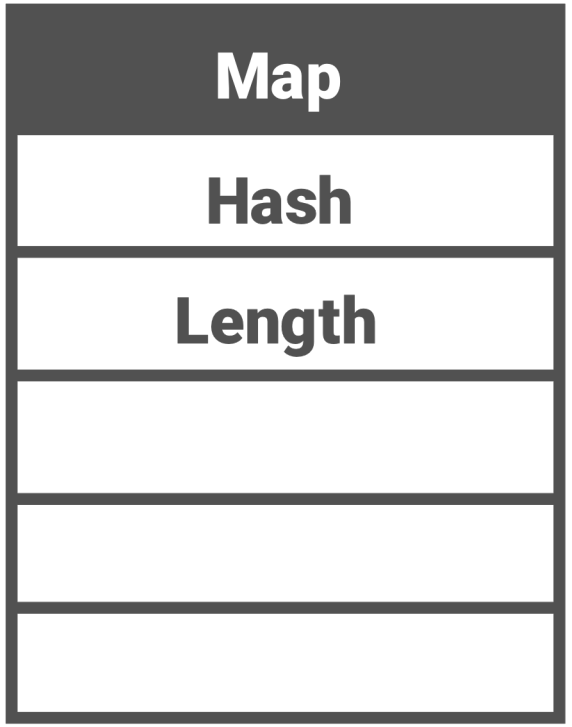
Reconstruct JS Values from Raw Memory

Known Layout

SeqOneByteString

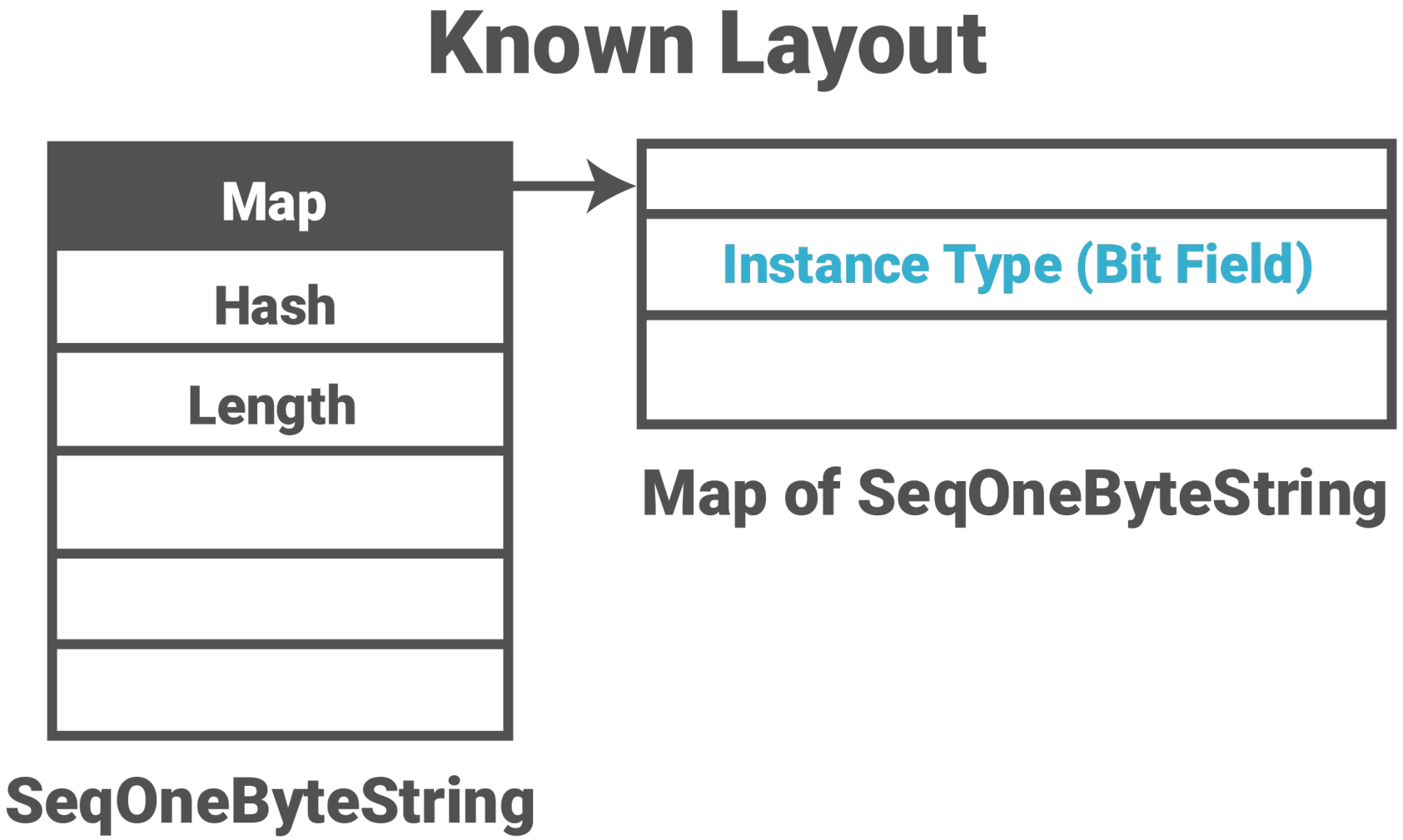
Reconstruct JS Values from Raw Memory

Known Layout

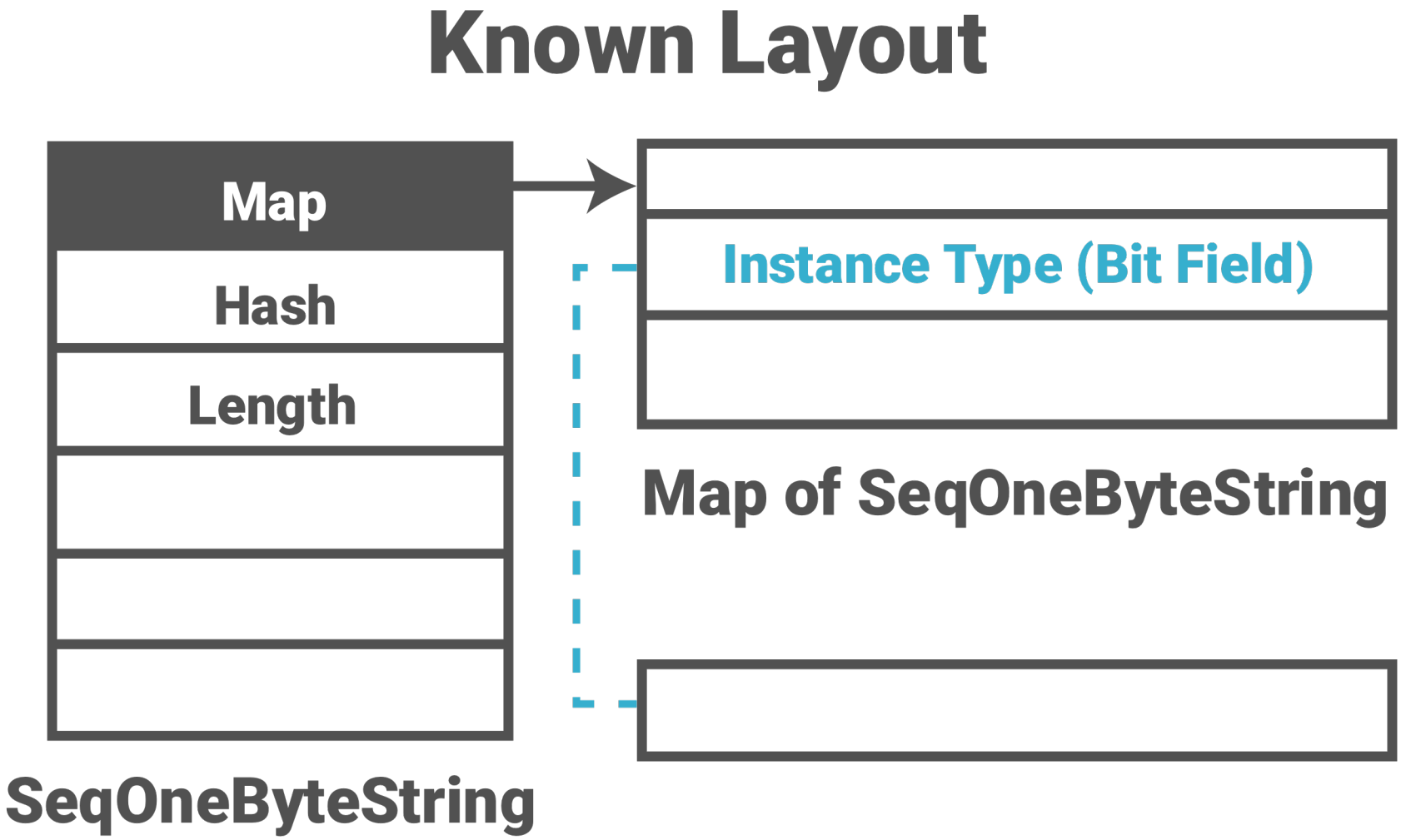


SeqOneByteString

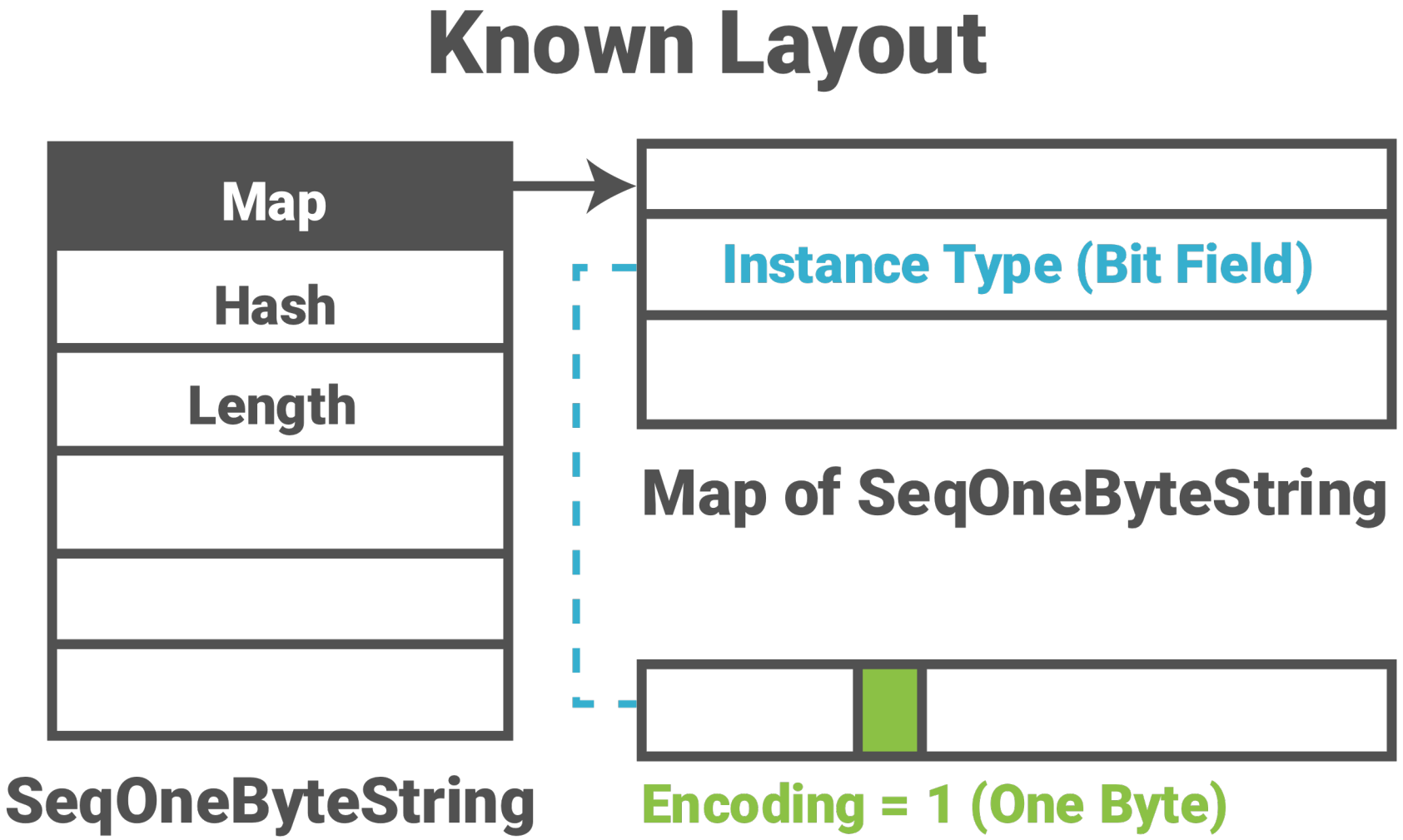
Reconstruct JS Values from Raw Memory



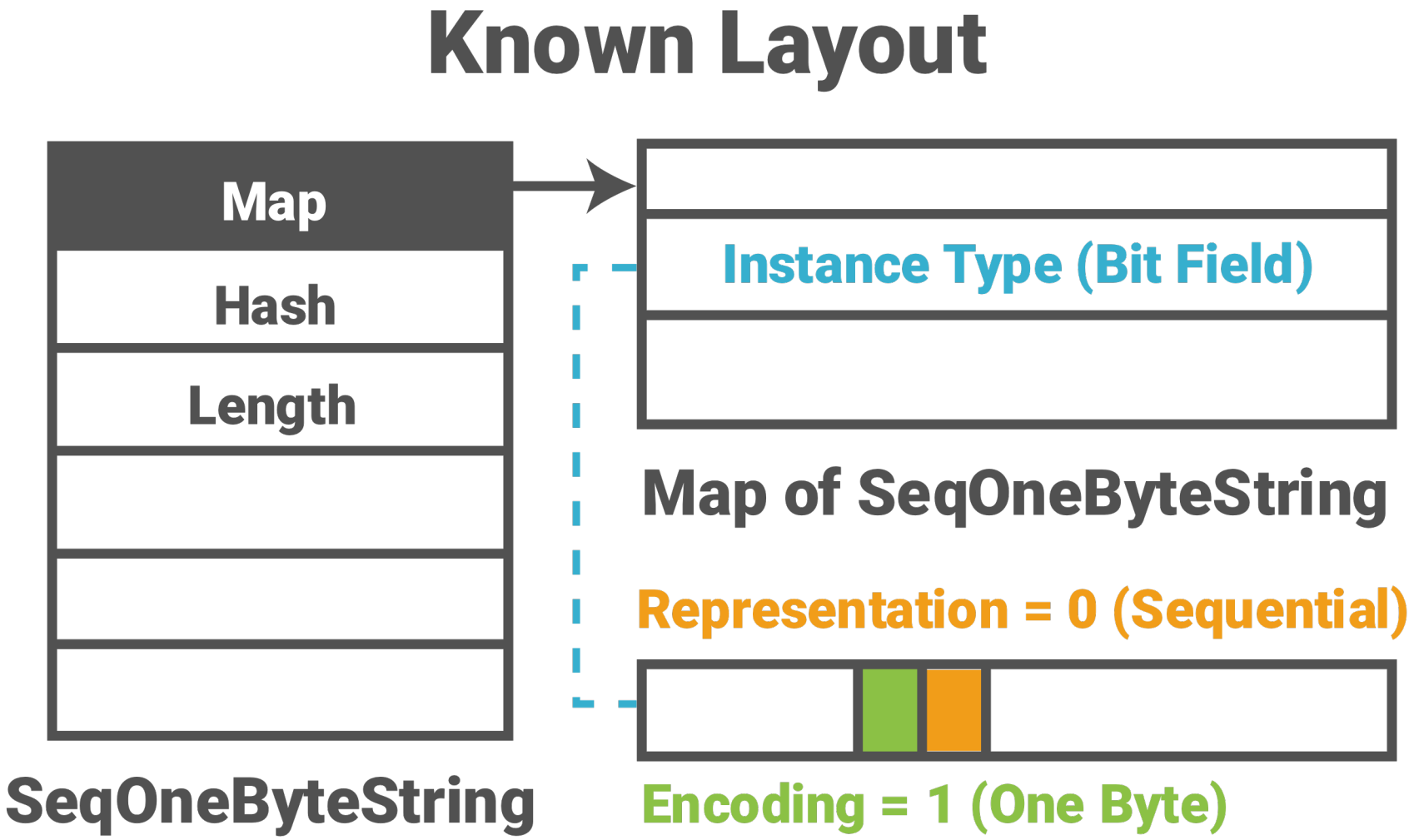
Reconstruct JS Values from Raw Memory



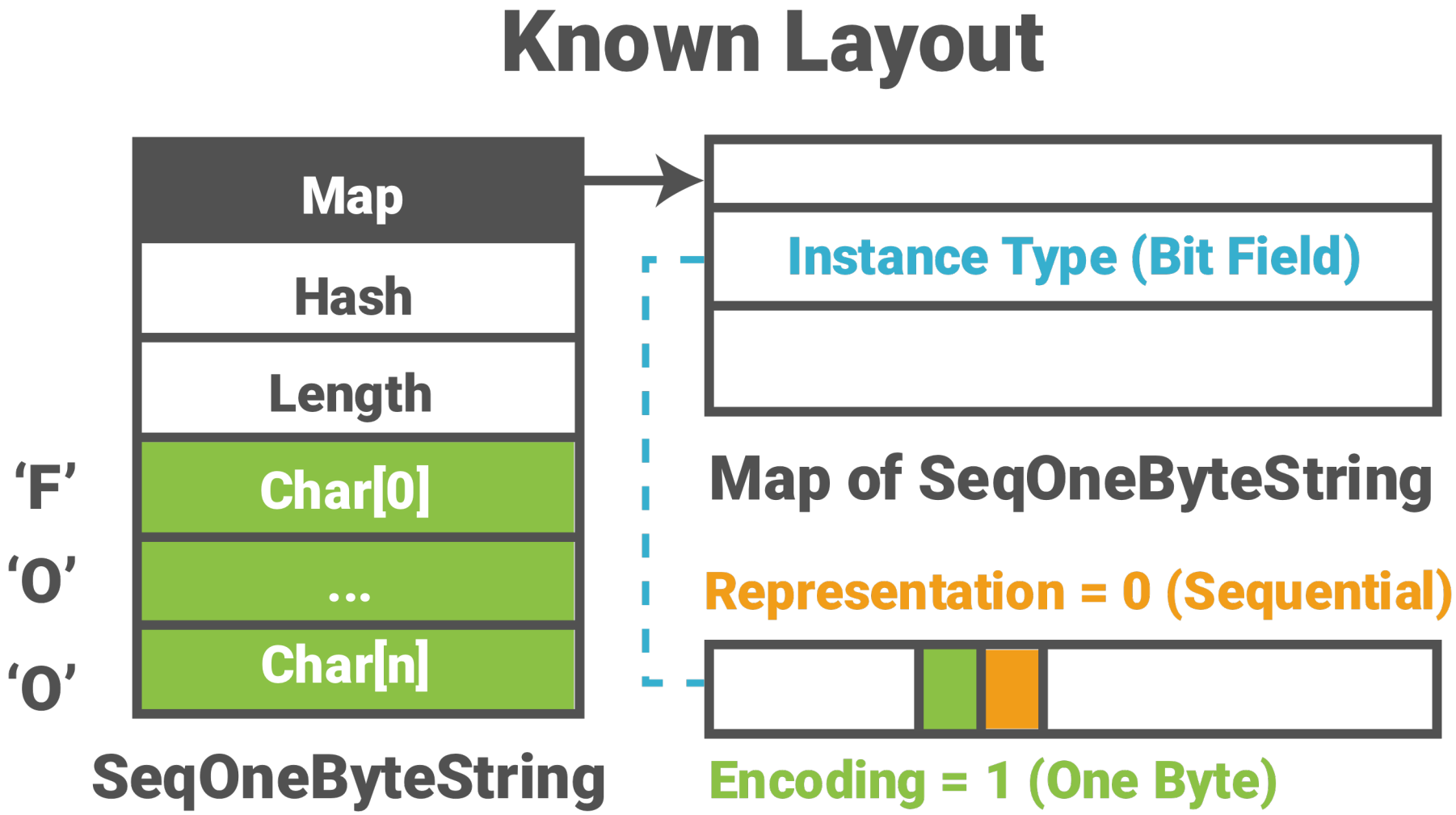
Reconstruct JS Values from Raw Memory



Reconstruct JS Values from Raw Memory

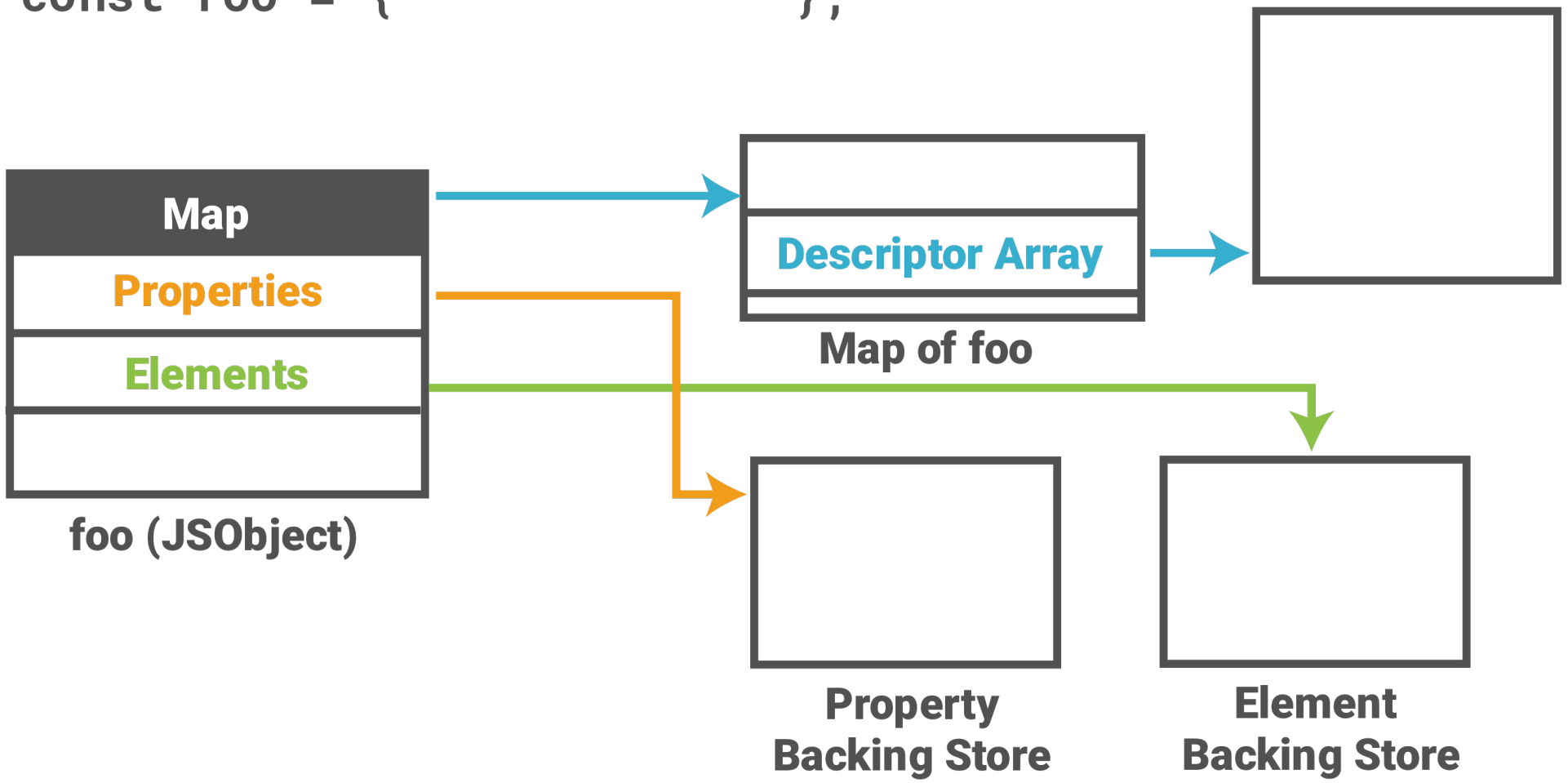


Reconstruct JS Values from Raw Memory



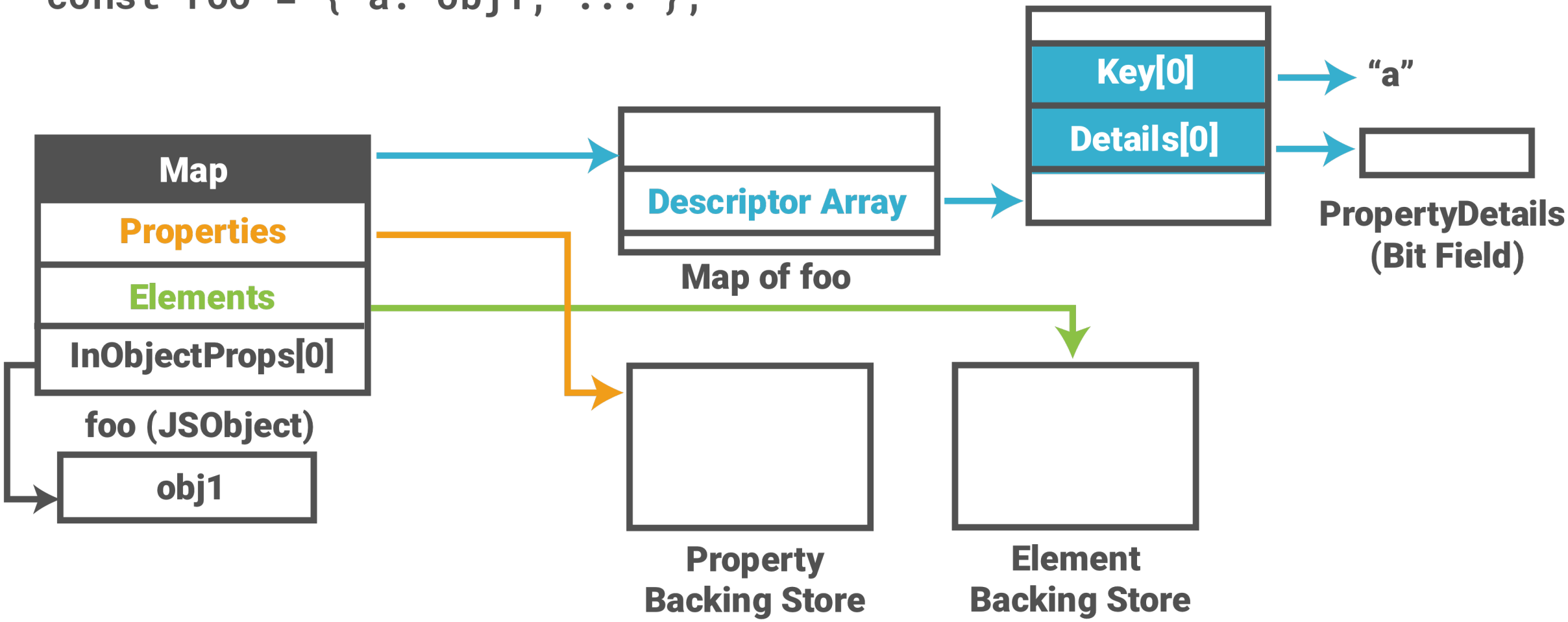
Reconstruct JS Values from Raw Memory

```
const foo = {                    };
```



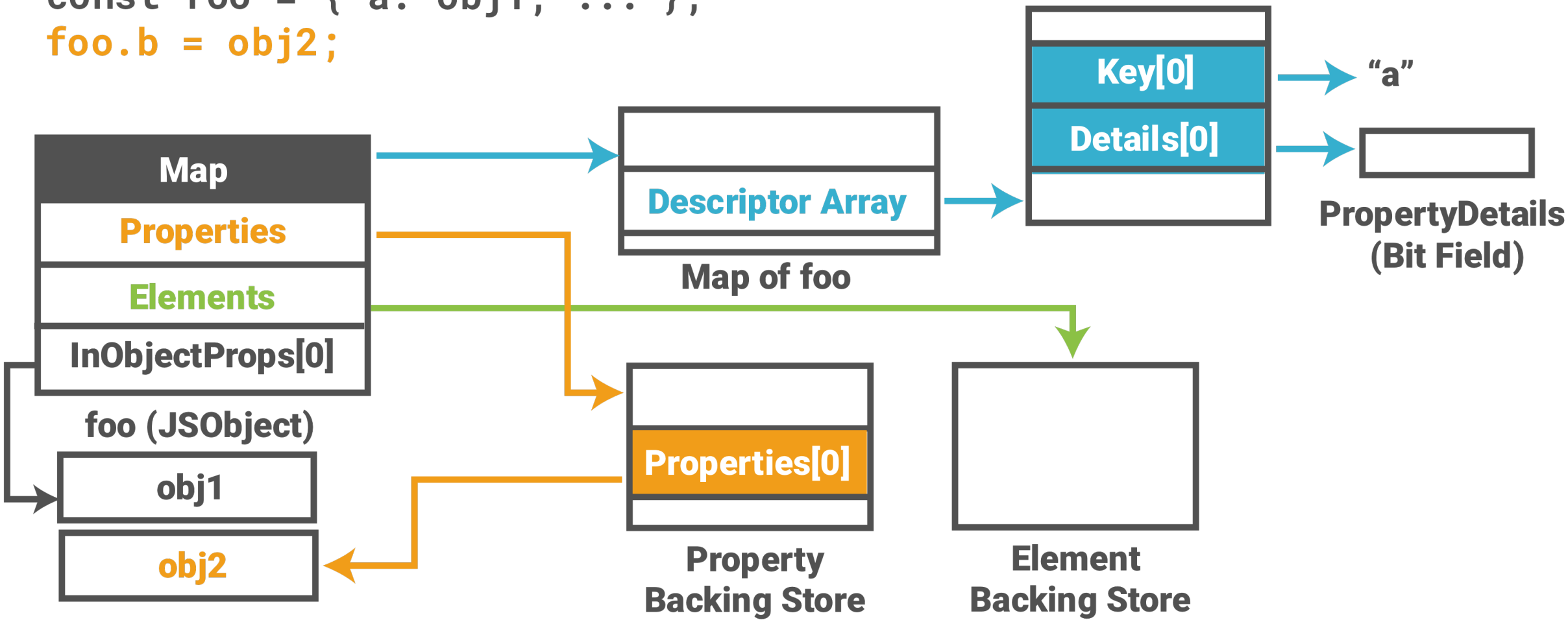
Reconstruct JS Values from Raw Memory

```
const foo = { a: obj1, ... };
```

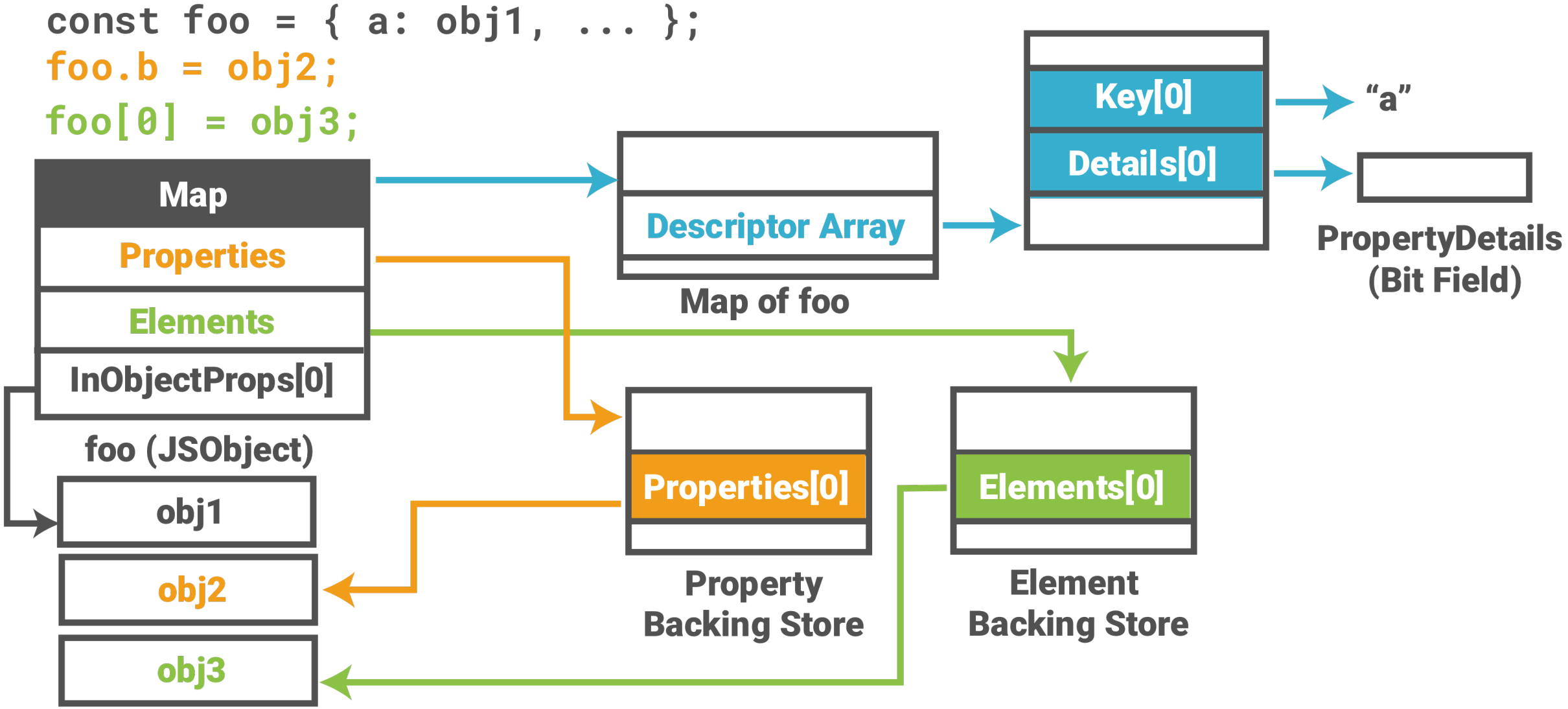


Reconstruct JS Values from Raw Memory

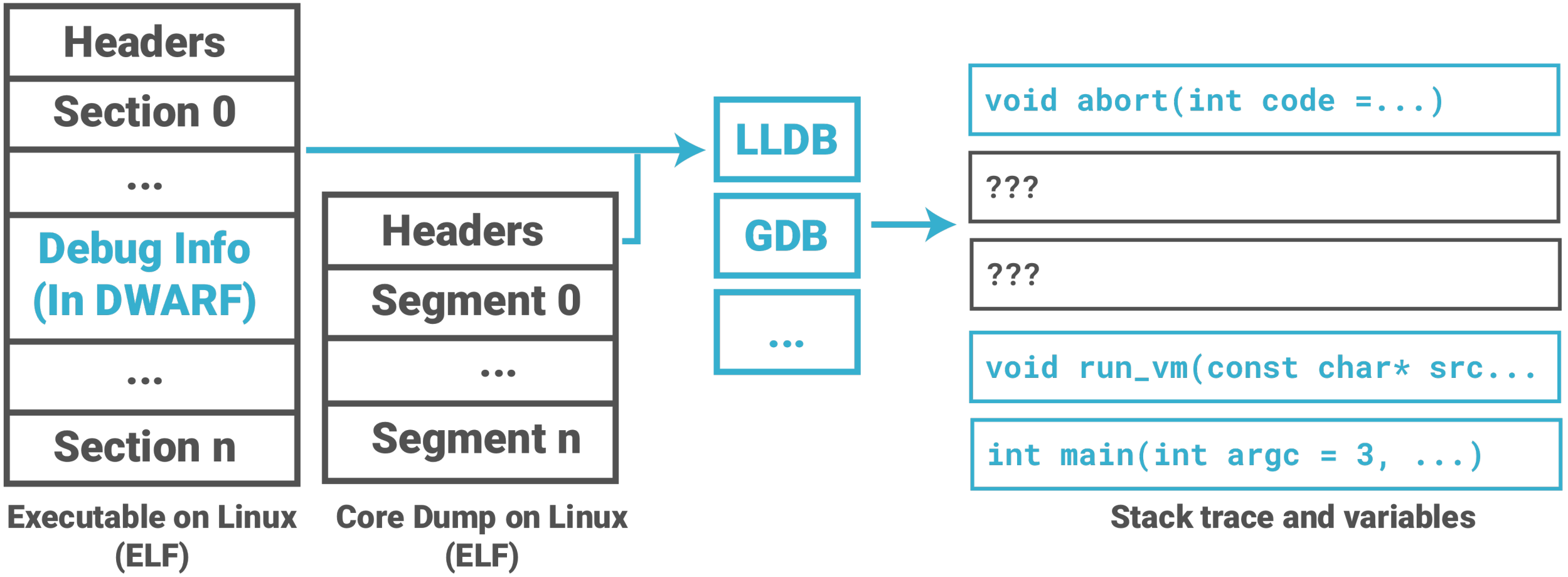
```
const foo = { a: obj1, ... };
foo.b = obj2;
```



Reconstruct JS Values from Raw Memory

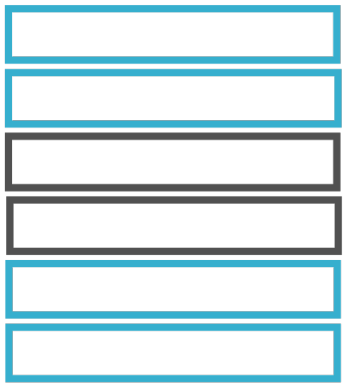


Unwinding the Stack: Native

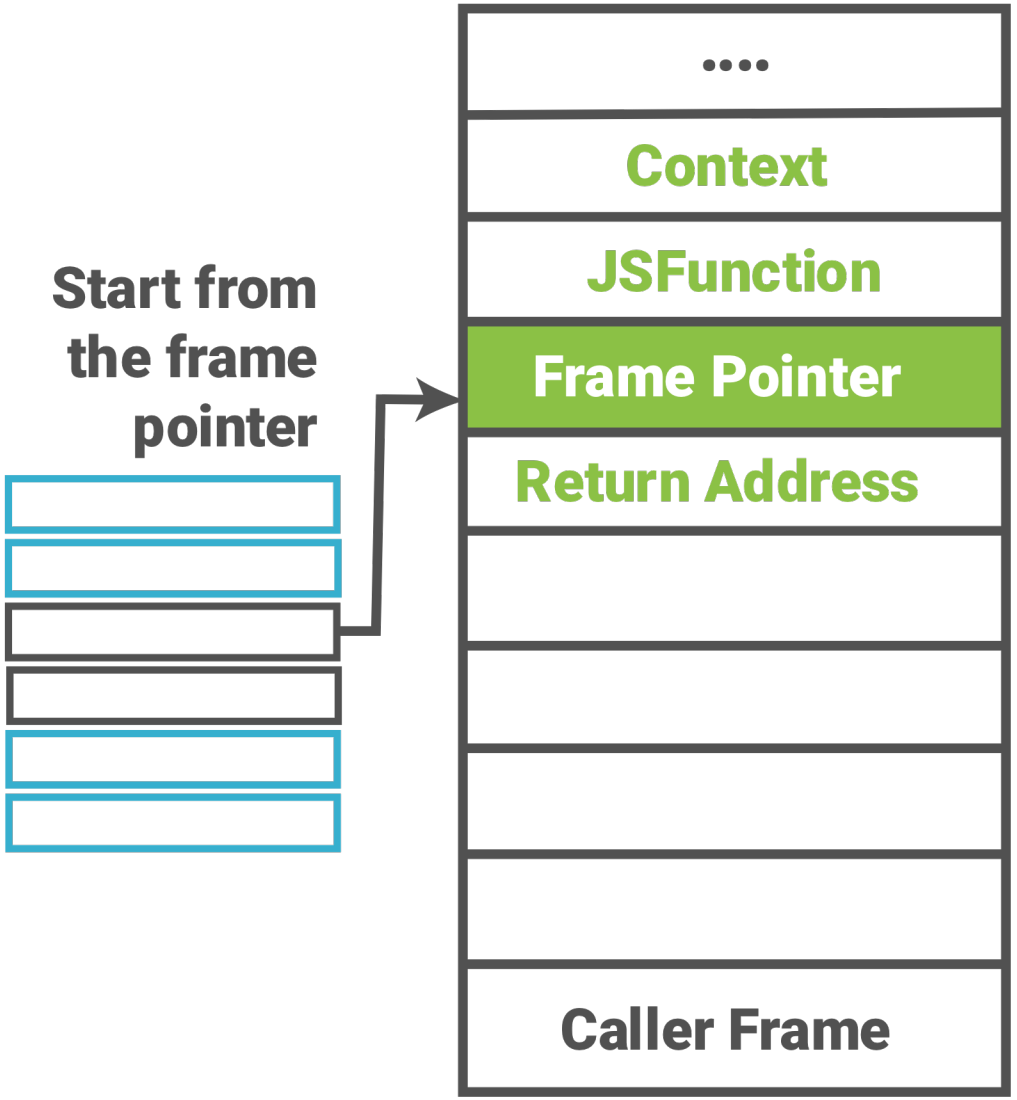


Unwinding the Stack: JS Symbols

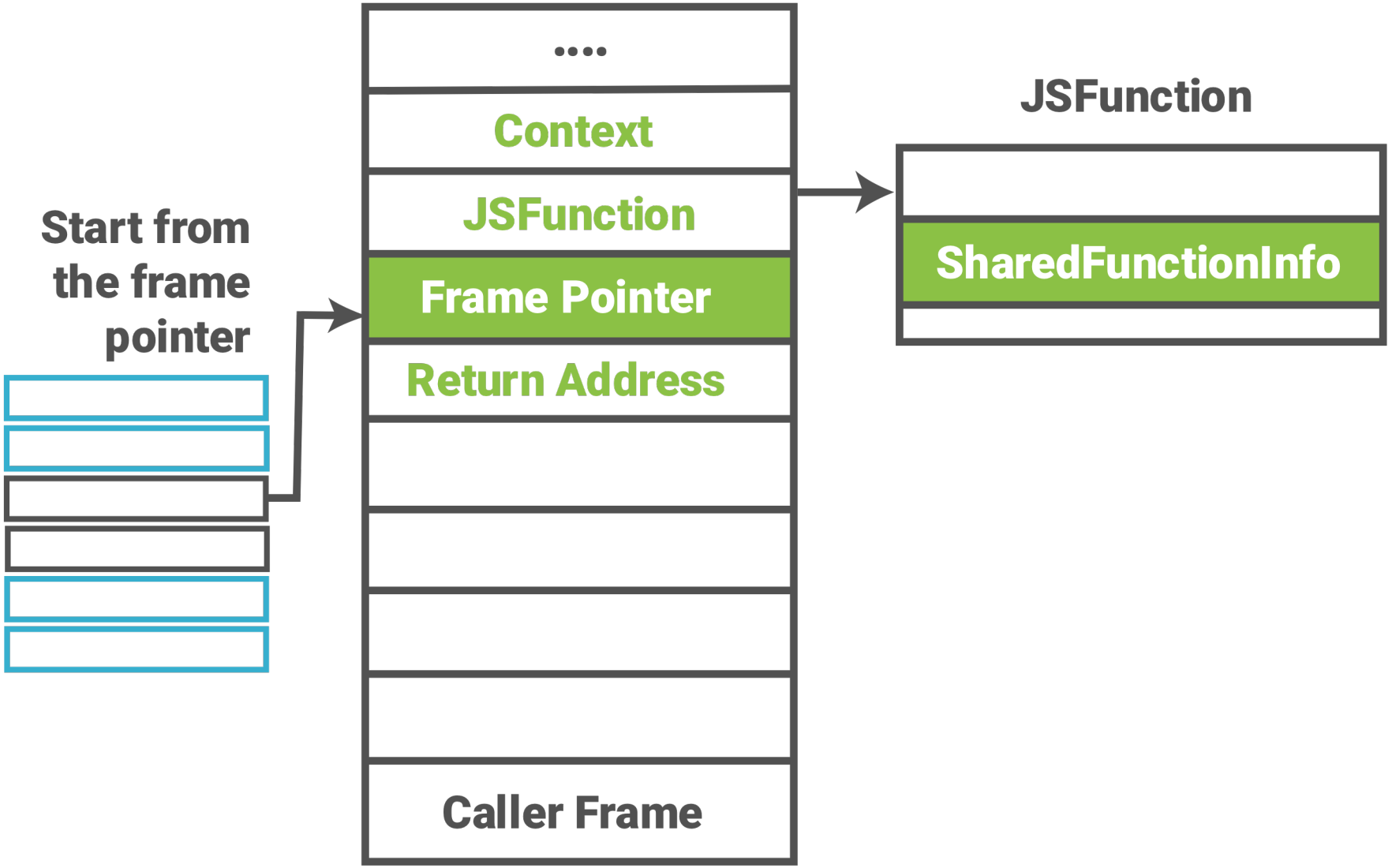
Start from the frame pointer



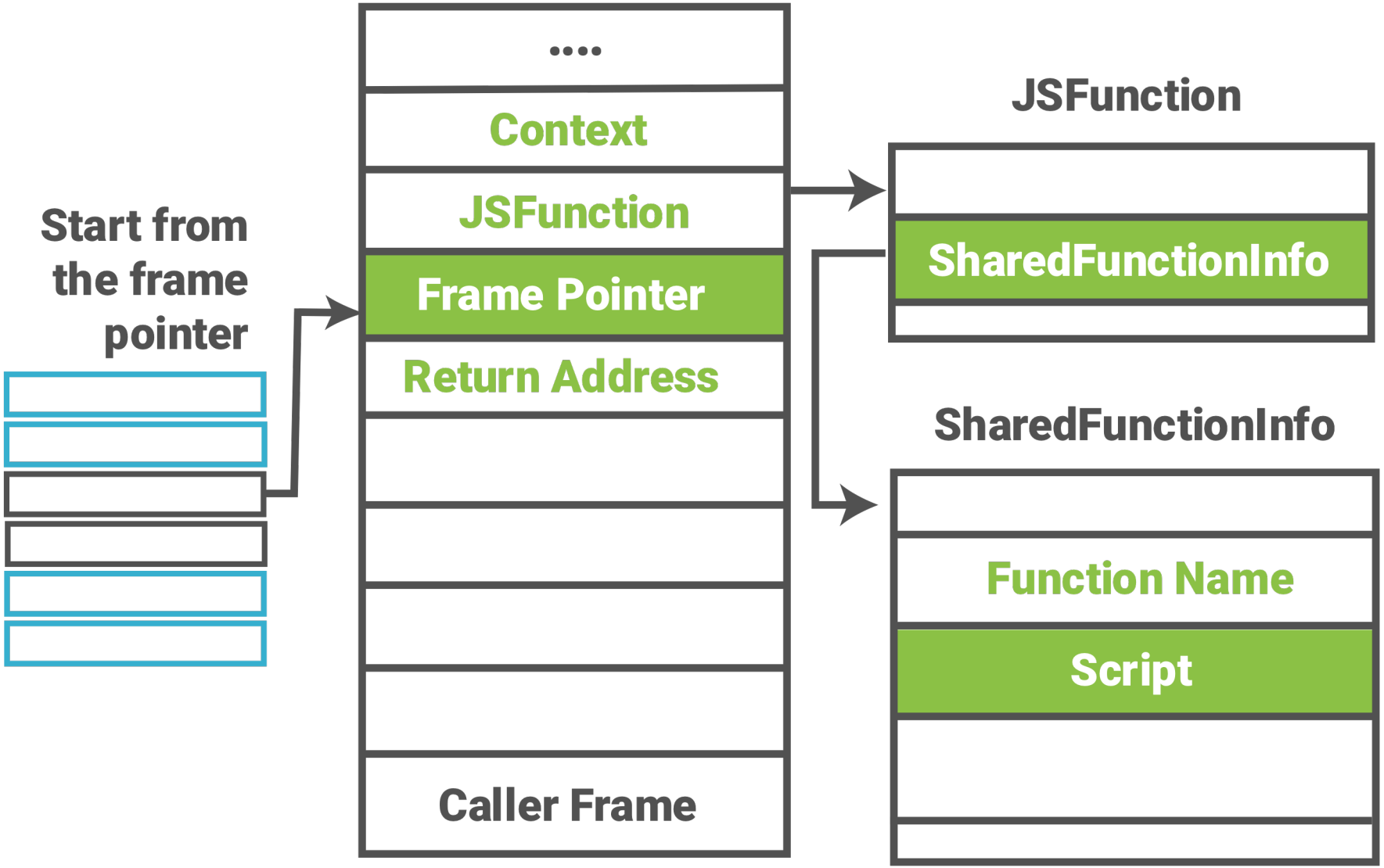
Unwinding the Stack: JS Symbols



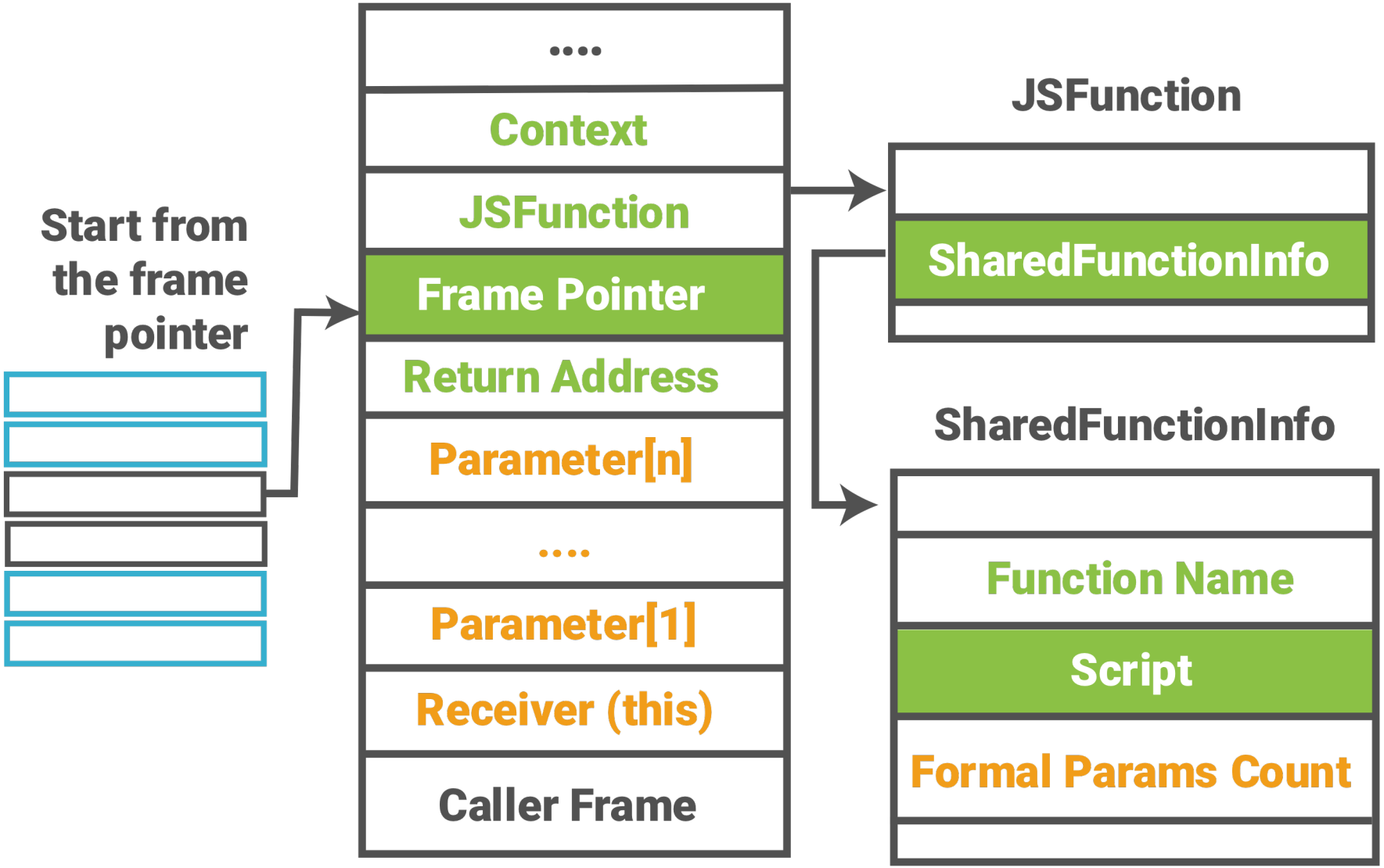
Unwinding the Stack: JS Symbols



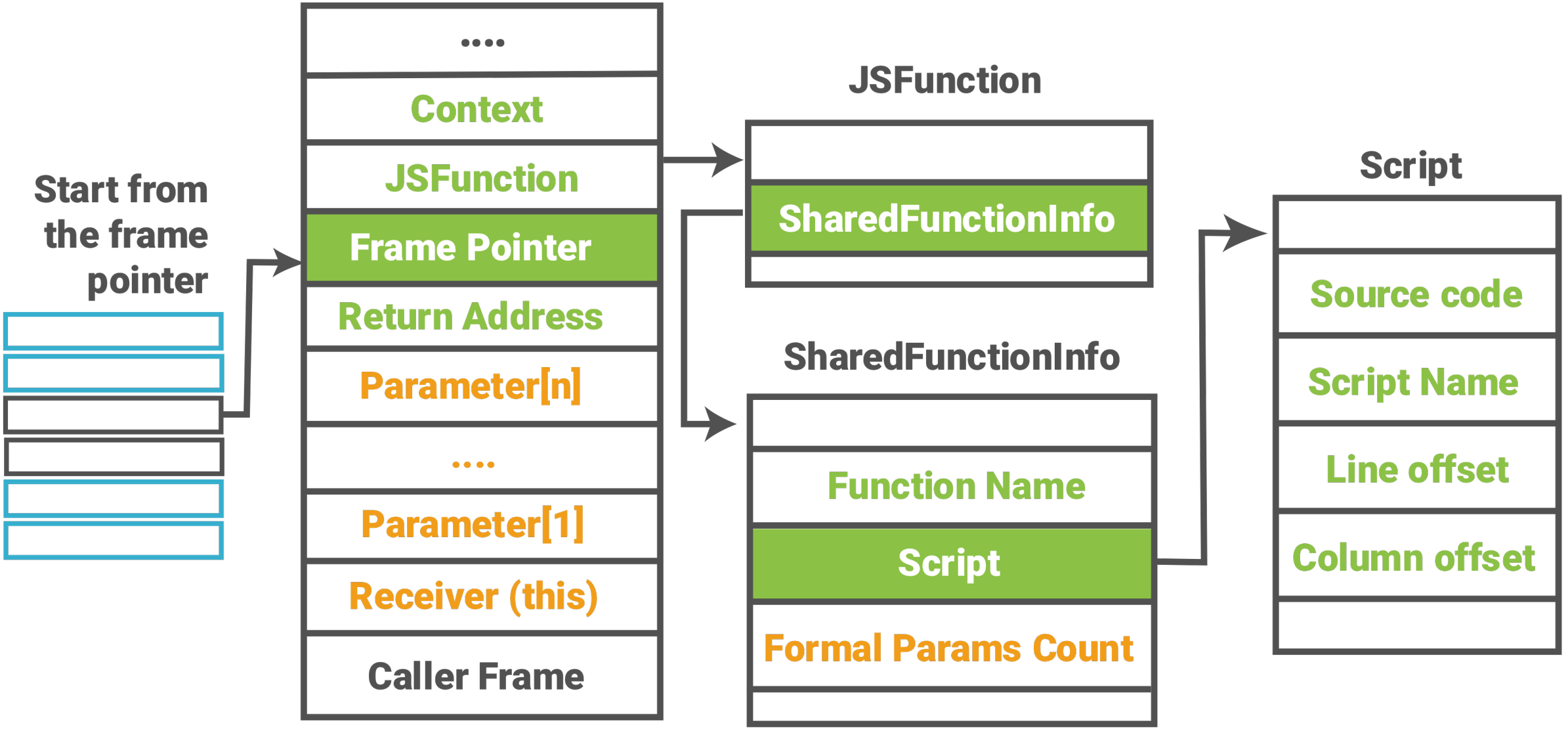
Unwinding the Stack: JS Symbols



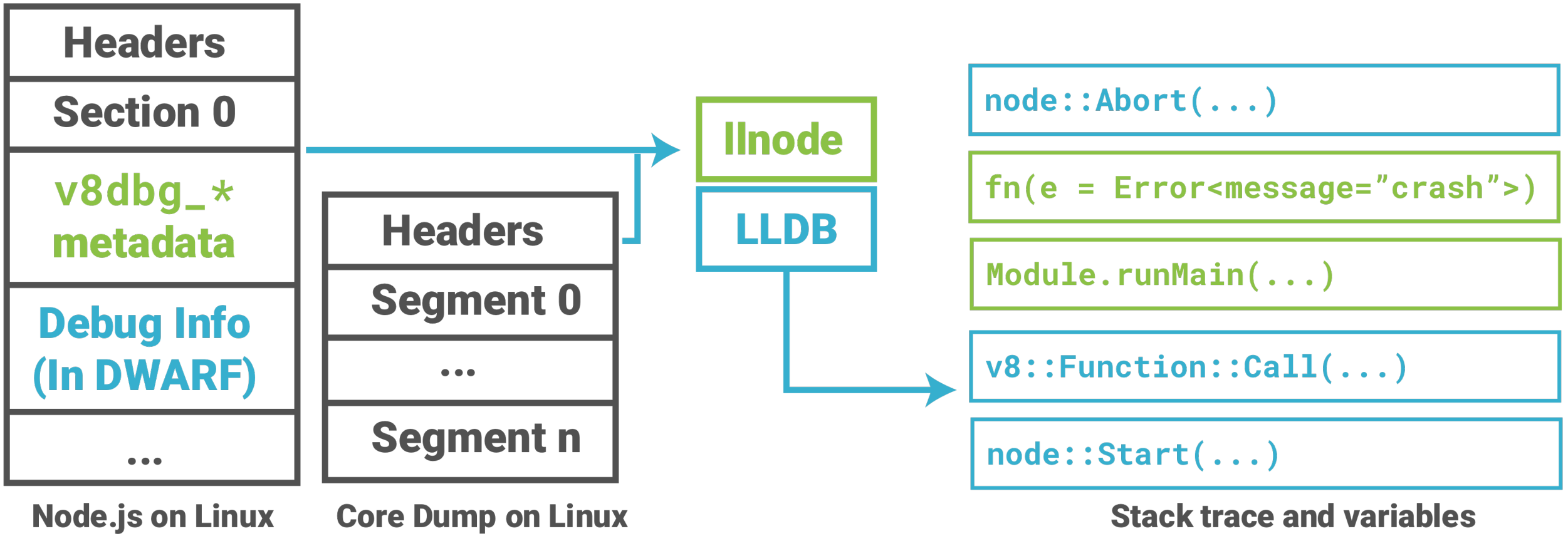
Unwinding the Stack: JS Symbols



Unwinding the Stack: JS Symbols



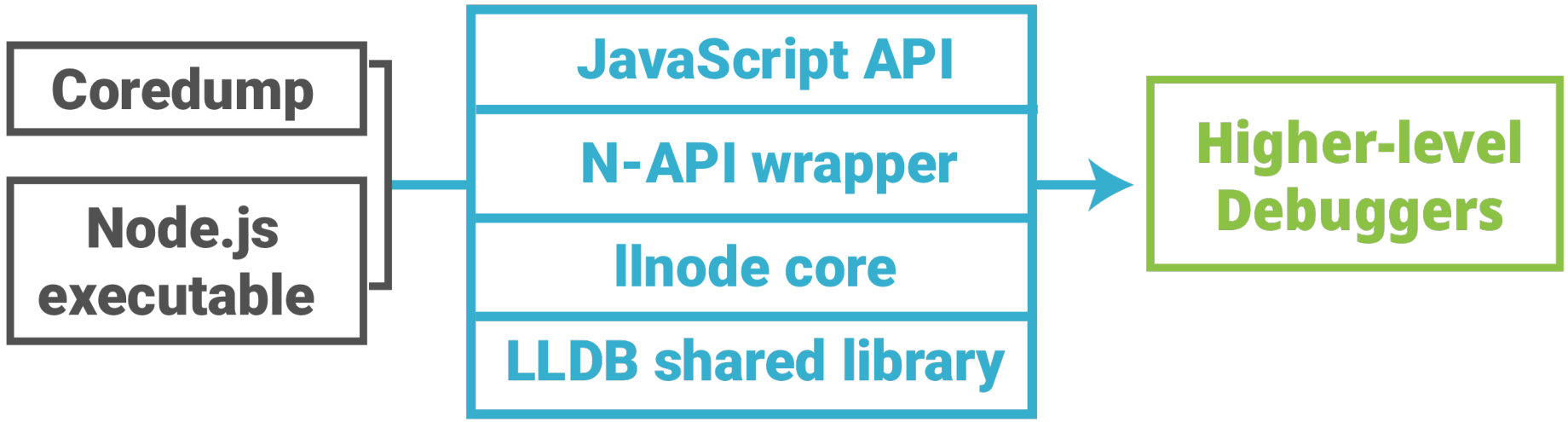
Unwinding the Stack: JS Symbols



JavaScript API of llnode

- As an ordinary Node.js C++ addon (macOS, Linux, FreeBSD)
- Restore JavaScript states back into JavaScript
- https://github.com/nodejs/llnode/blob/master/JSAPI.md
- https://zhuanlan.zhihu.com/p/41178823

```
npm install llnode --llnode_build_addon=true
```



JavaScript API of llnode

```
const llnode = require('llnode')
  .fromCoredump('/path/to/core', '/path/to/node');
const process = llnode.getProcessObject();
console.log(`Process ${process.pid}: ${process.state}}`);
process.threads.forEach((thread) => {
        console.log(`Thread ${thread.threadId}`);
        thread.frames.forEach((frame, index) => {
                console.log(`#${index} ${frame.function}`);
        });
});
```

JavaScript API of llnode

```
llnode.getHeapTypes().forEach((type) => {
        console.log(`${type.typeName}: ${type.totalSize}`);
        console.log(`${type.instanceCount} instances`);
        for (const instance of type.instances) {
                console.log(`0x${instance.address}`,
                            instance.value);
        }
})
```

The llnode approach

- Highly dependent on the implementation
  - Easy to break whenever V8 changes its **internal** object layout
- Works well enough with the Node.js LTS schedule
  - Only need to support very few versions of V8

The llnode approach

- Minimum effort from the VM's side
  - It works!
  - No overhead during runtime
- Fragmentation among llnode, mdb_v8, .etc
  - Limited to the platform supported by the native debugger

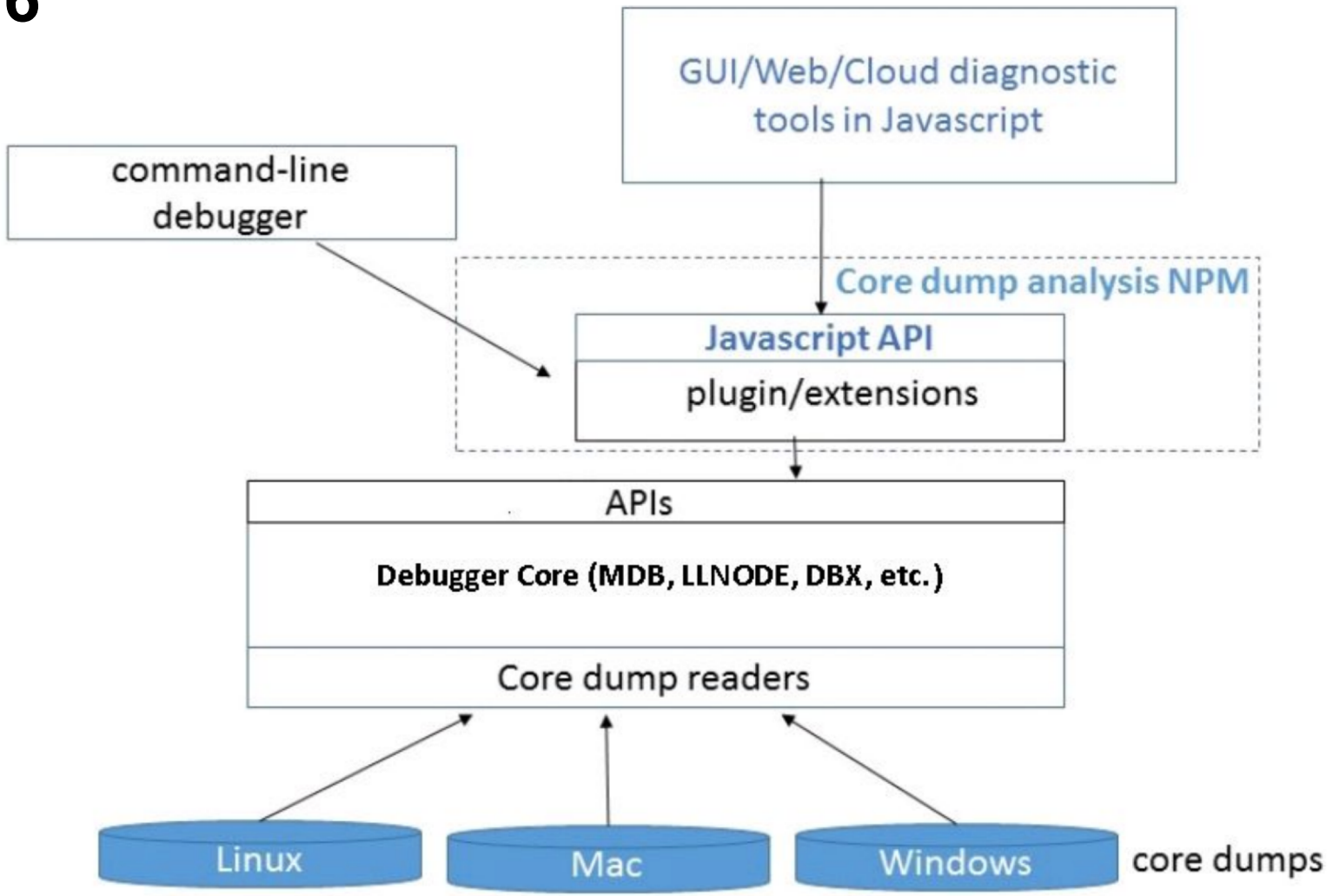
Future of Node/JS Post-mortem Diagnostics

Node Interactive EU, 2016

https://www.slideshare.net/michaeldawson3572846/post-mortem-talk-node-interactive-eu

Get Involved!

https://github.com/nodejs/diagnostics



Summary

- **Introduction to post-mortem diagnostics**
  - Analysis of a core dump
  - Recovering states from dynamic language VMs

Summary

- **Introduction to post-mortem diagnostics**
  - Analysis of a core dump
  - Recovering states from dynamic language VMs
- **A tour of llnode**
  - Reconstruct JavaScript values from raw memory
  - Unwinding the Stack
  - JavaScript API of llnode

Thank you!