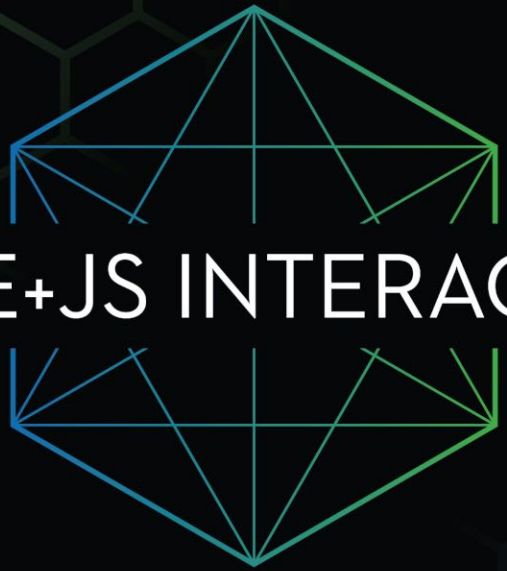


NODE+JS INTERACTIVE

OCTOBER 10-12, 2018 | VANCOUVER, CANADA

Bringing JavaScript Back to Life

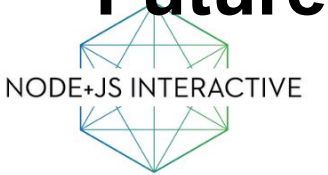
Joyee Cheung, *Igalia*

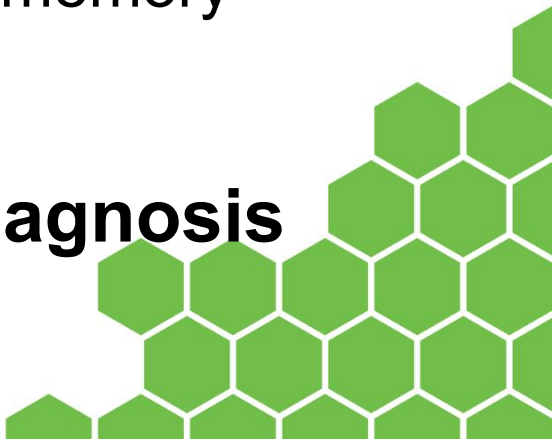
About Me

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

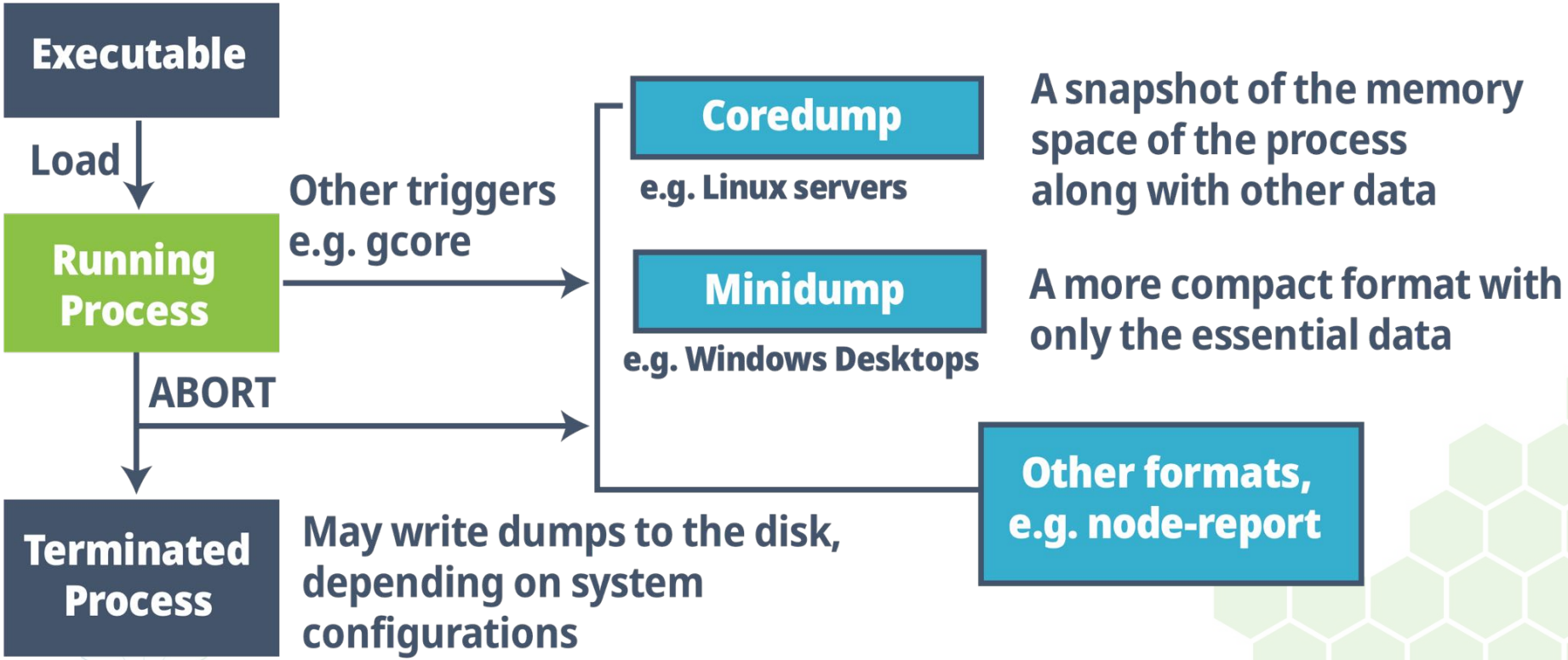
Agenda

- **Introduction to post-mortem diagnosis**
  - Analysis of a coredump
  - Recovering states from dynamic language VMs
- **A tour of llnode**
  - Reconstruct JavaScript values from raw memory
  - Unwinding the Stack
  - JavaScript API of llnode
- **Future of Node/JS Post-mortem Diagnosis**



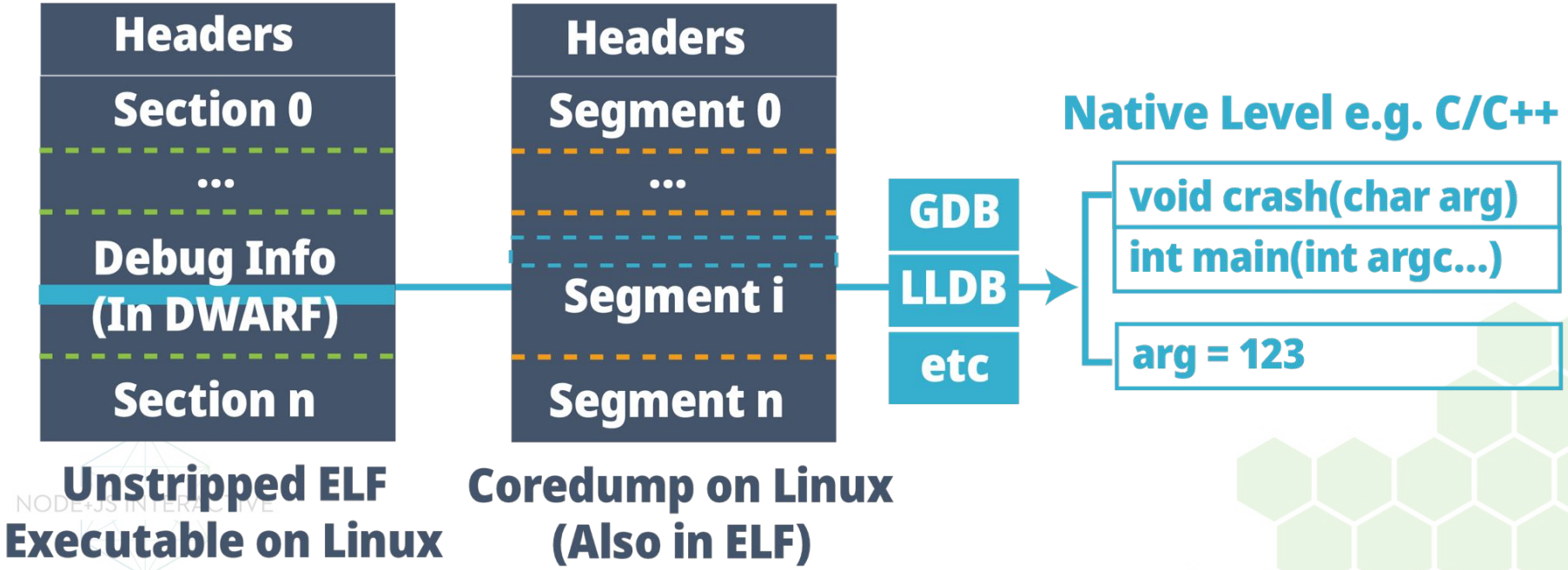


Introduction to Post-Mortem Diagnostics



Analysis of a Coredump

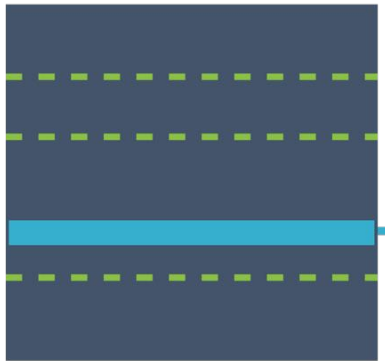
```
int main(int argc, char* argv[]) {
 crash(123);
}
```



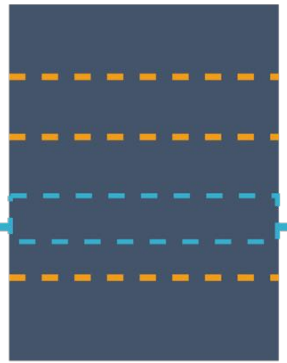
Analysis of a Coredump

```
int main(int argc, char* argv[]) {
 run_vm(argc, argv);
}
```

```
function fn(e) { throw e }
fn(new Error('crash'))
```



Executable embedding dynamic language VMs



Coredump with dynamic symbols/type info generated as data

GDB

LLDB

etc

Dynamic Language Level

Stack Frames ???

Variables ???

void run_vm(...)

int main(int argc...)

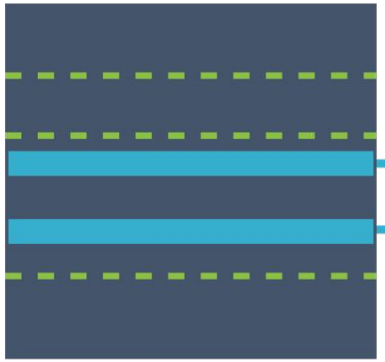
argc = 3

Native Level e.g. C/C++

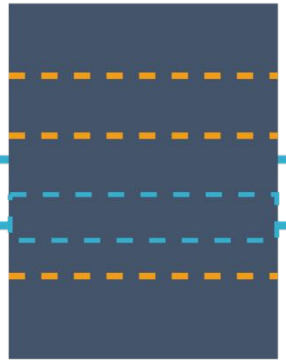
Recovering States from Dynamic Language VMs

```
int main(int argc, char* argv[]) {
 run_vm(argc, argv);
}
```

```
function fn(e) { throw e }
fn(new Error('crash'))
```



Executable embedding dynamic language VMs (+ additional debug info)



Coredump with dynamic symbols/type info generated as data

Plugin

Debugger

Dynamic Language Level

function fn(e) {...}

e = Error { ... }

void run_vm(...)

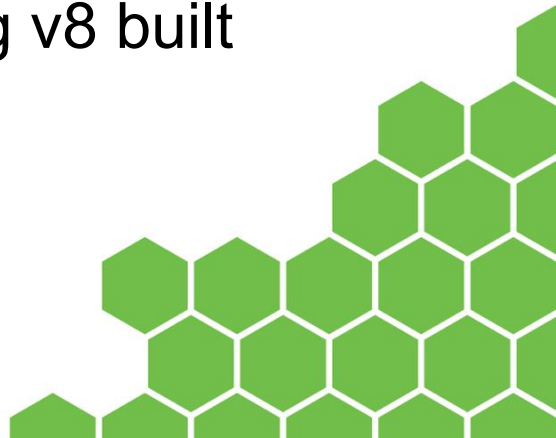
int main(int argc...)

argc = 3

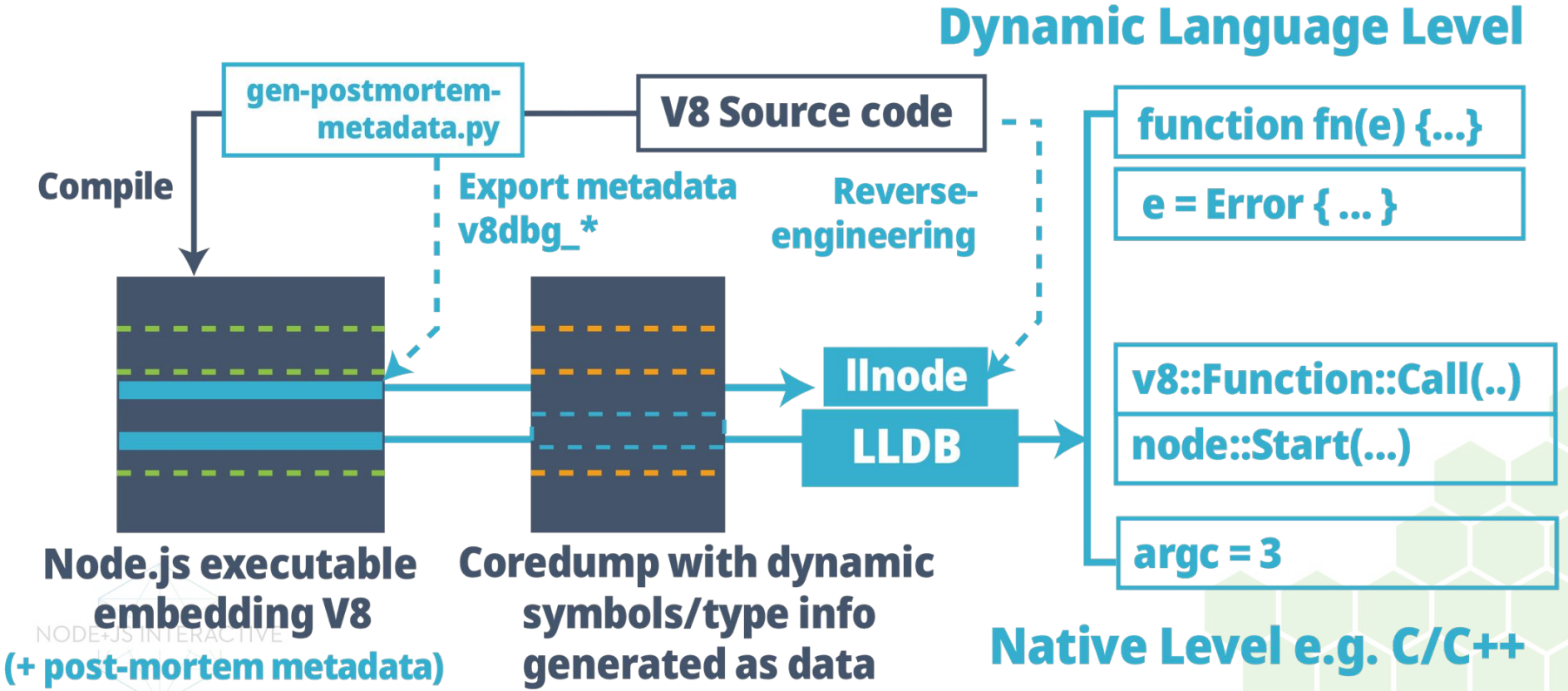
Native Level e.g. C/C++

A Tour of llnode

- Project under the Node.js Diagnostics Working Group
- https://github.com/nodejs/llnode
- Plugin of the LLDB debugger
- Can be used to debug both coredumps and live processes of Node.js applications
- Also works with other programs embedding v8 built with post-mortem support
  - e.g. d8 shell

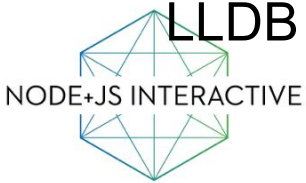


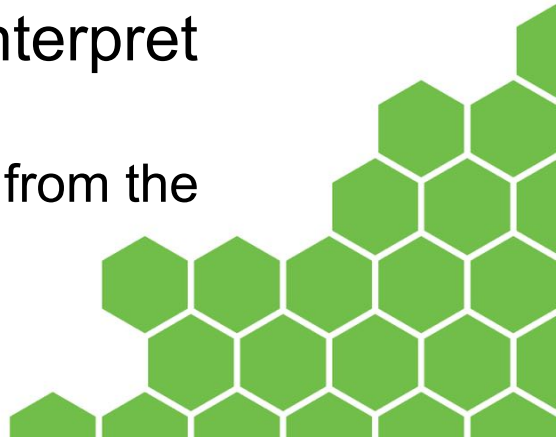
A Tour of llnode



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 coredump
   - Infrastructures and cross-platform support come from the LLDB API

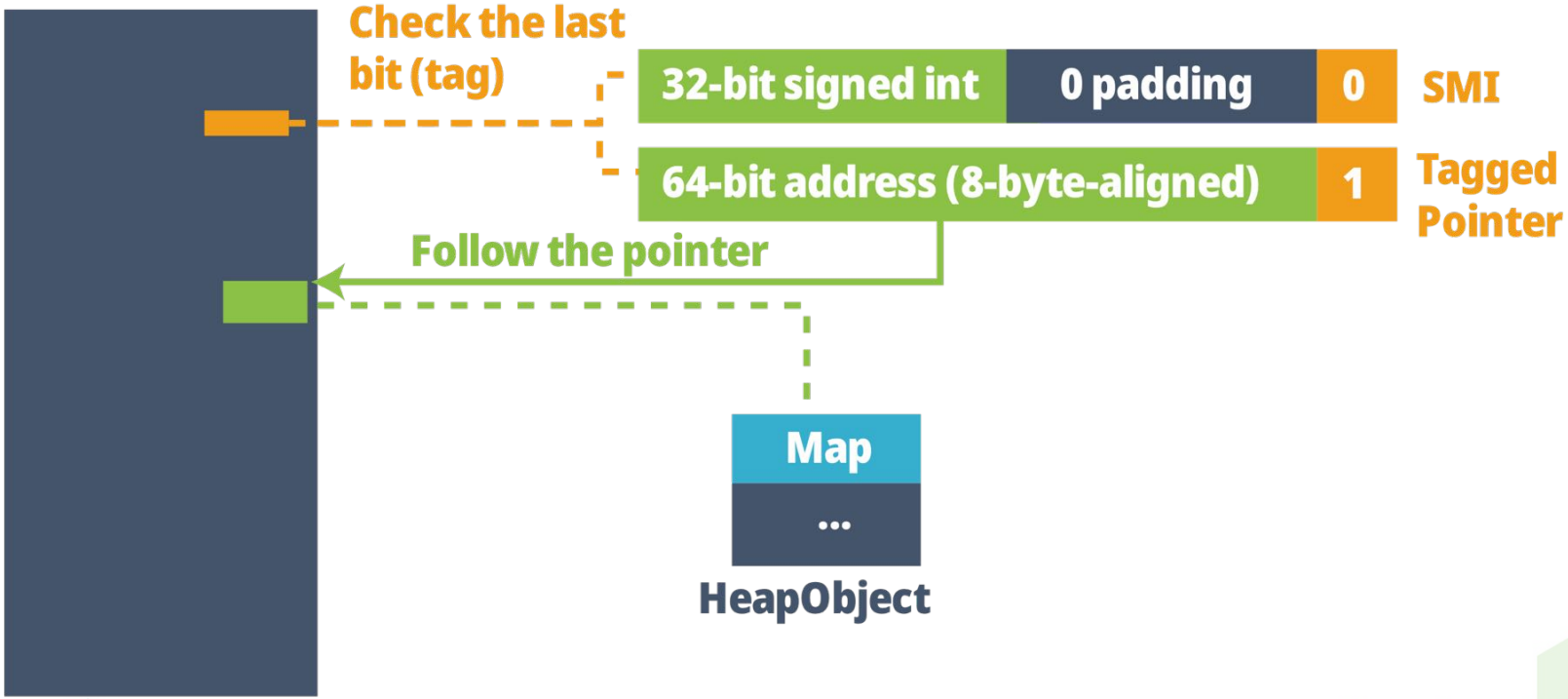




Reconstruct JS Values from Raw Memory

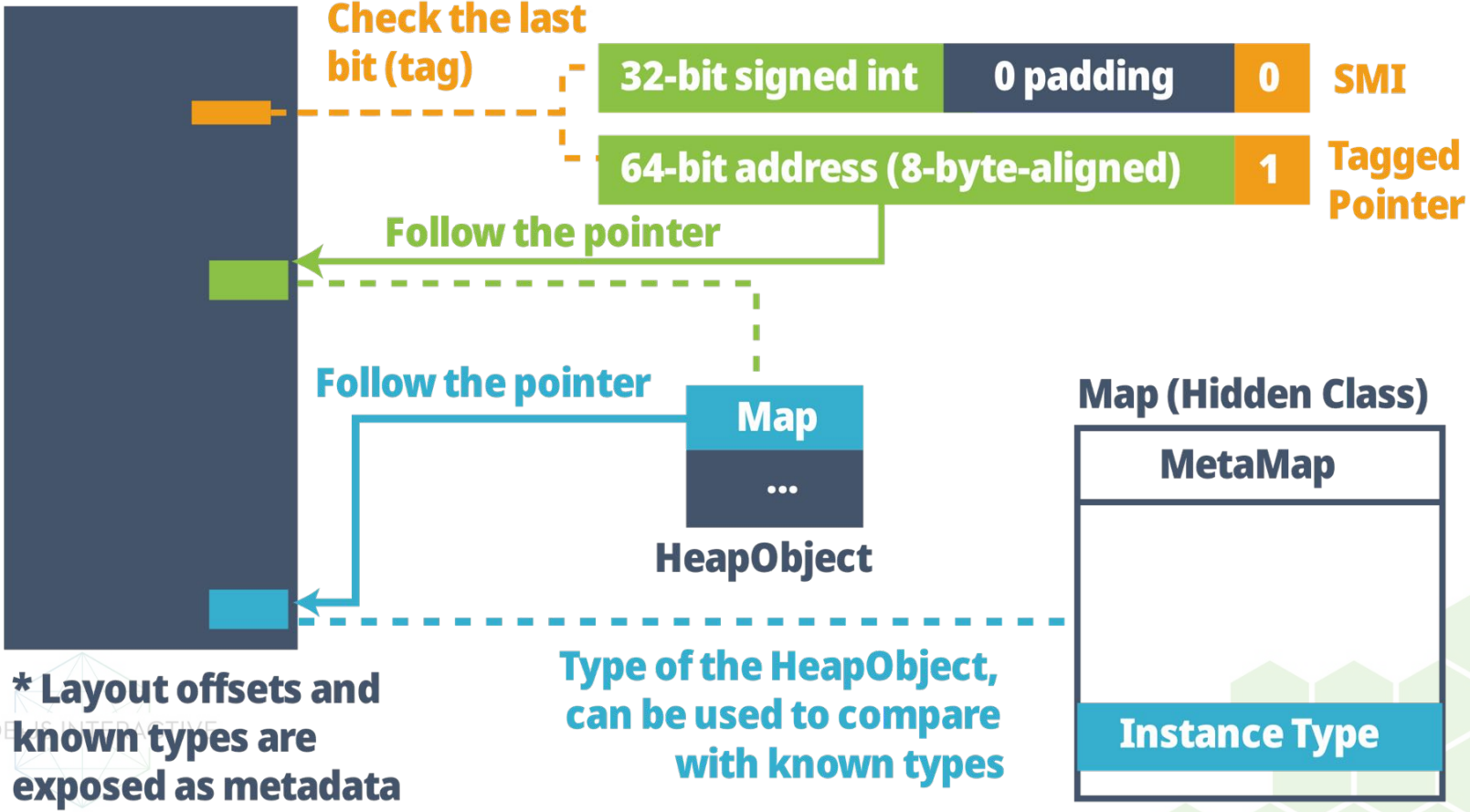
* Layout offsets and known types are exposed as metadata

Reconstruct JS Values from Raw Memory

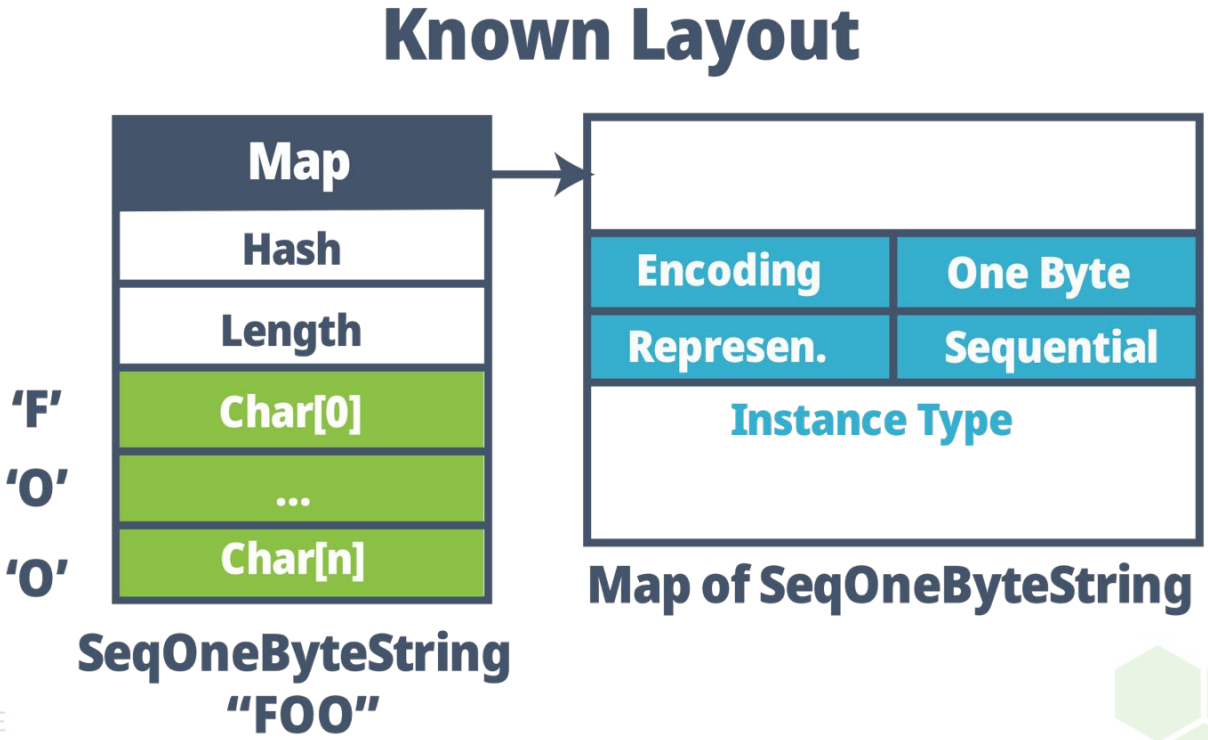


* Layout offsets and known types are exposed as metadata

Reconstruct JS Values from Raw Memory

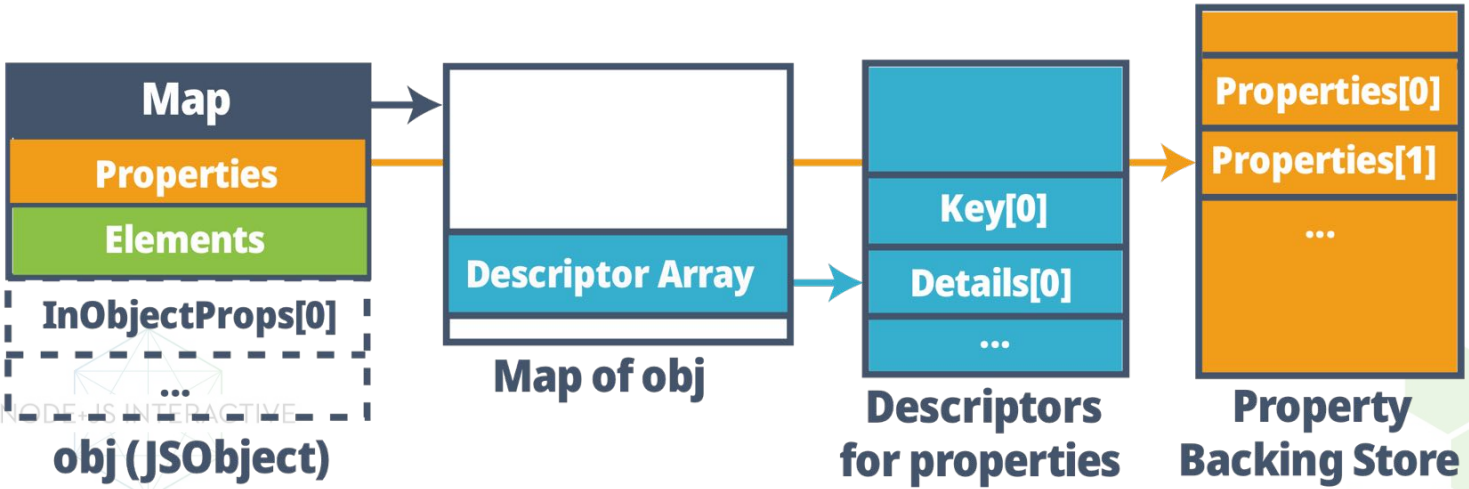


Reconstruct JS Values from Raw Memory



Reconstruct JS Values from Raw Memory

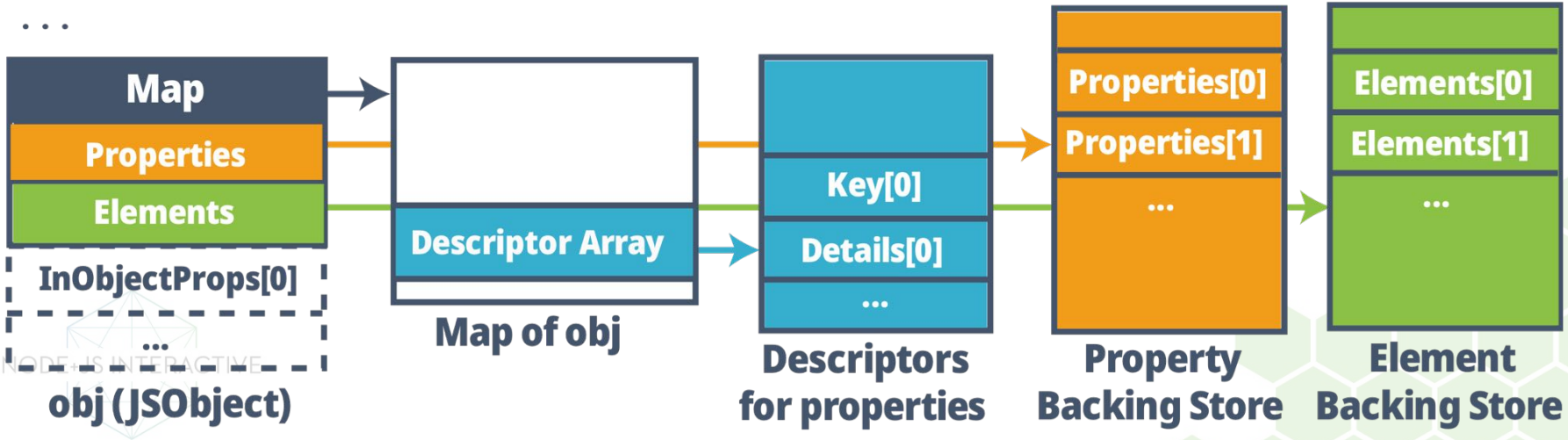
```
const obj = {
 foo: 1, // in-object properties
 ...
};
obj.x = ‘test’; // Named properties with descriptors
```



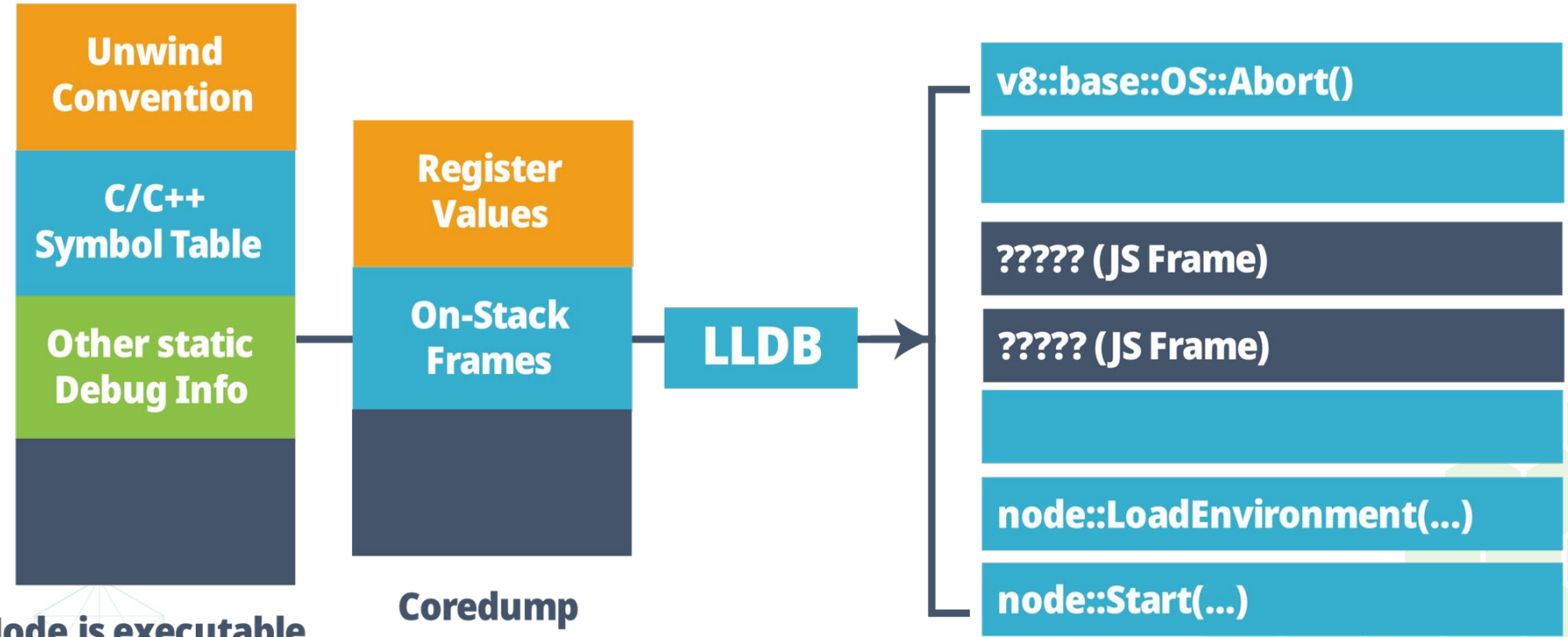
Reconstruct JS Values from Raw Memory

```
const obj = {
foo: 1, // in-object properties
...
};
obj.x = 'test'; // Named properties with descriptors
...
obj[0] = 1; // Indexed elements
...
```

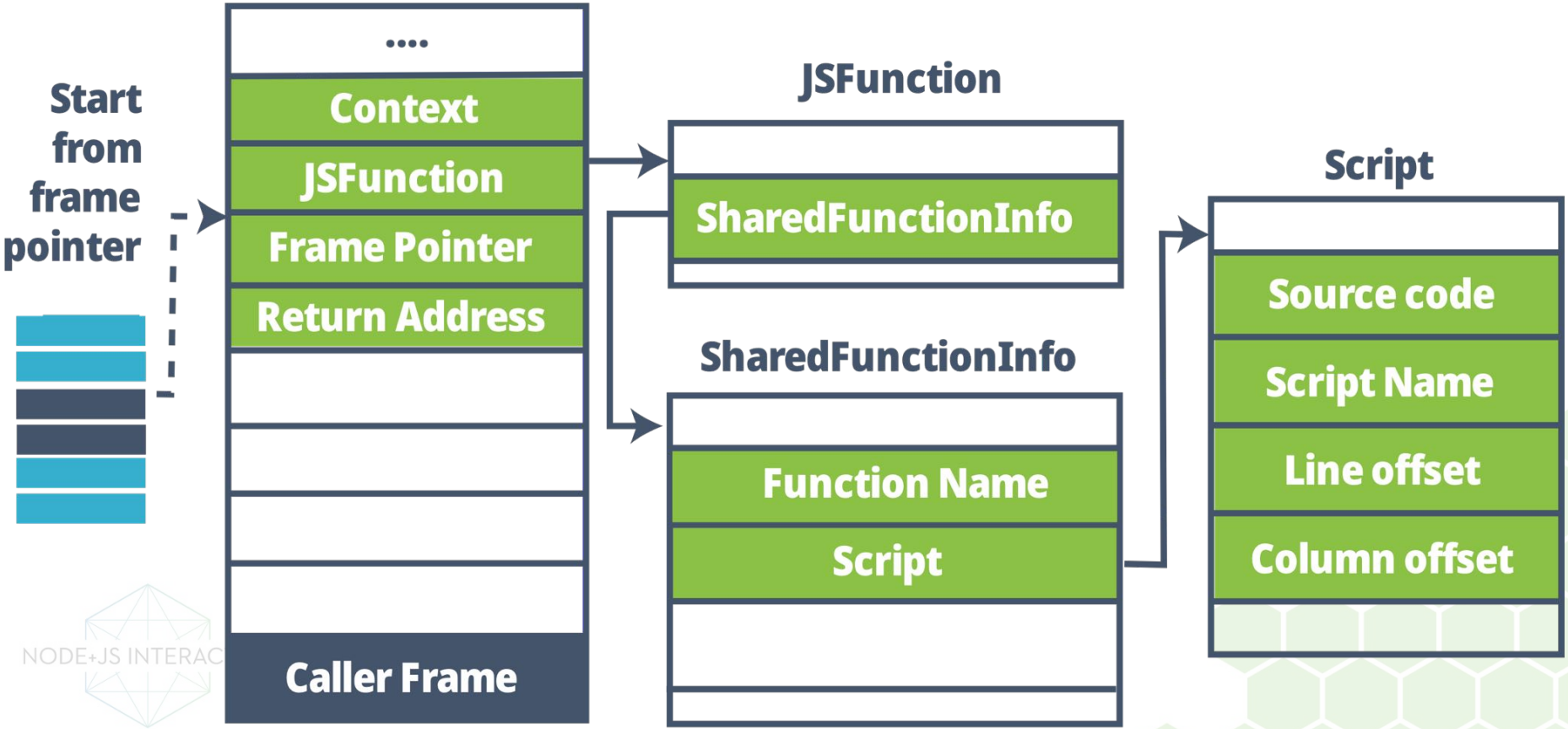
* Objects in dictionary mode have flexible property dictionaries instead of fixed descriptors



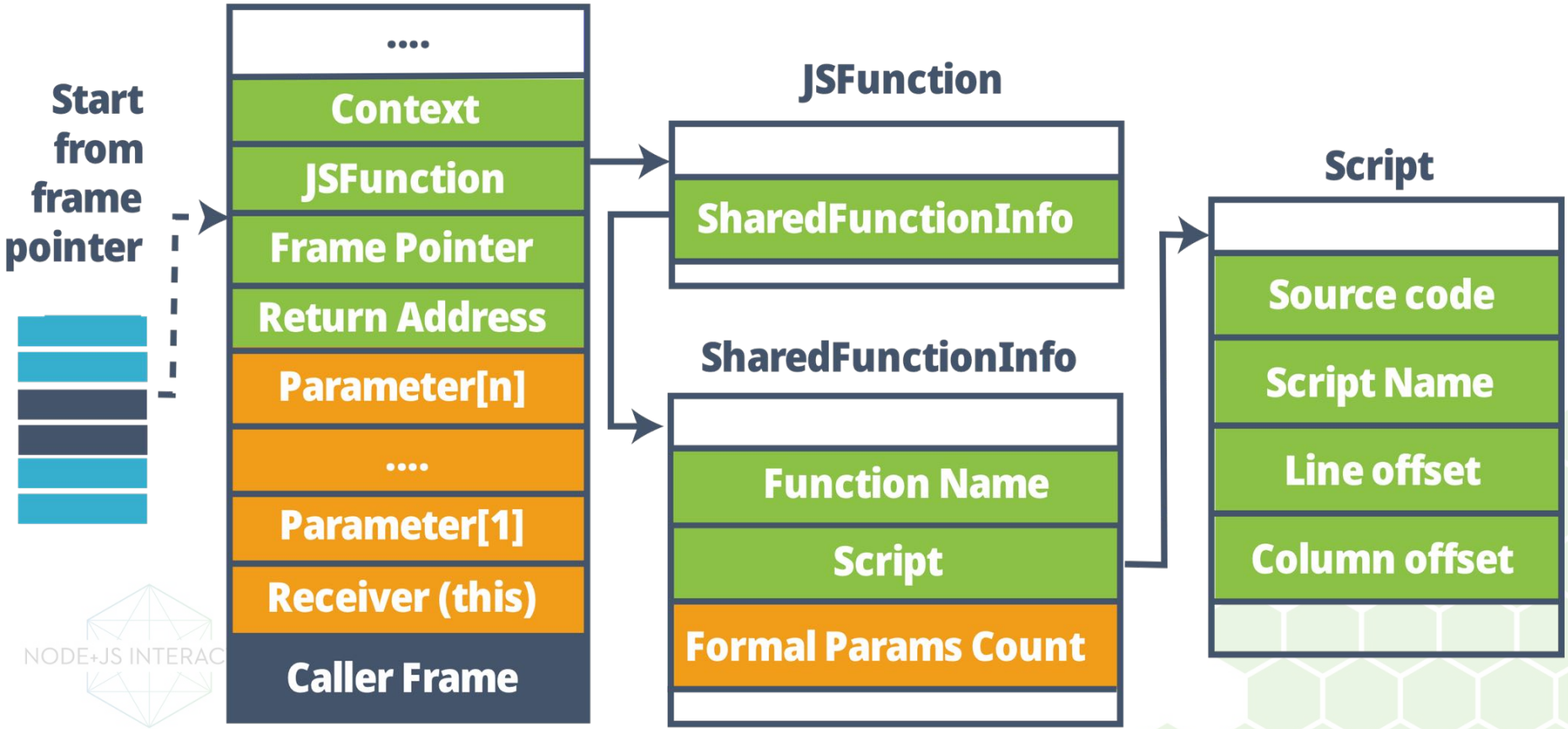
Unwinding the Stack: Native



Unwinding the Stack: JS Symbols

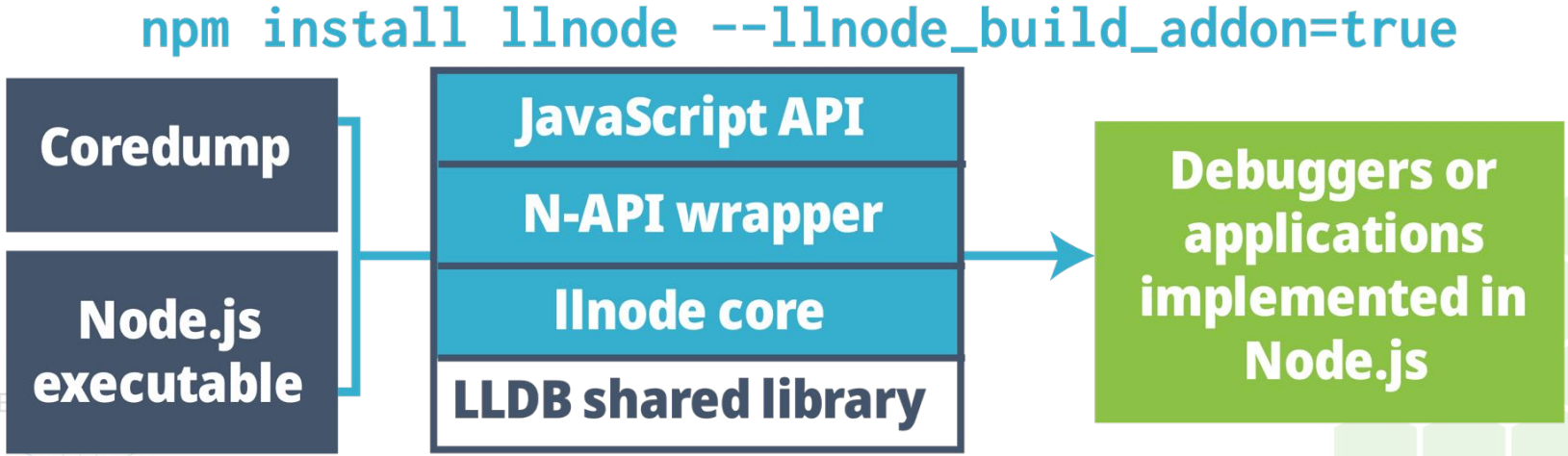


Unwinding the Stack: JS Symbols



JavaScript API of llnode

- As an ordinary Node.js C++ addon (macOS, Linux, FreeBSD, Windows?)
- Primitive interface for now
  - Return strings that are formatted for the LLDB console
  - https://github.com/nodejs/llnode/blob/master/JSAPI.md



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

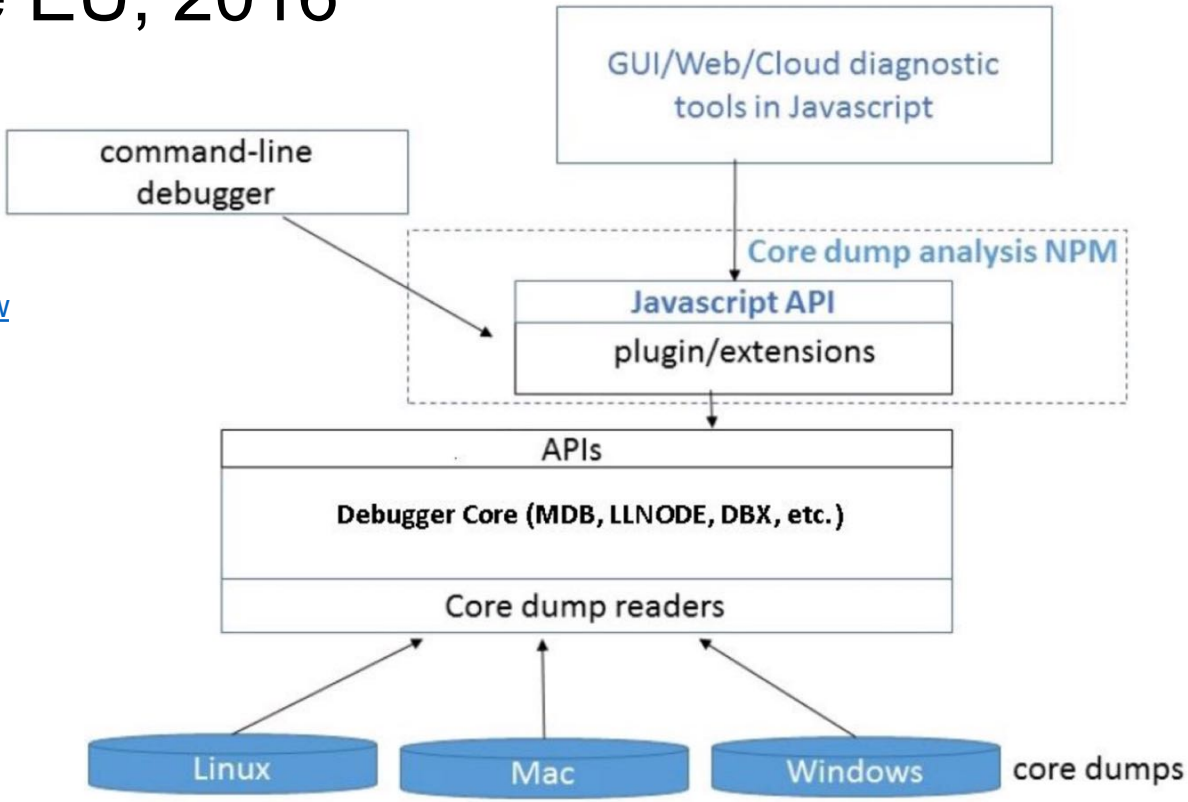
- Highly dependent on the implementation: easy to break whenever V8 changes its **internal** object layout
- Works well enough with the Node.js LTS schedule: only need to support very few versions of V8
- Minimum effort from the VM's side
  - It works!
  - No overhead during runtime
- Fragmentation among llnode, mdb_v8, .etc
  - Limited to the platform supported by the native debugger

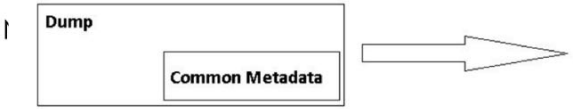
Future of Node/JS Post-mortem Diagnosis

Node Interactive EU, 2016

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

https://github.com/nodejs/diagnostics





Thanks