

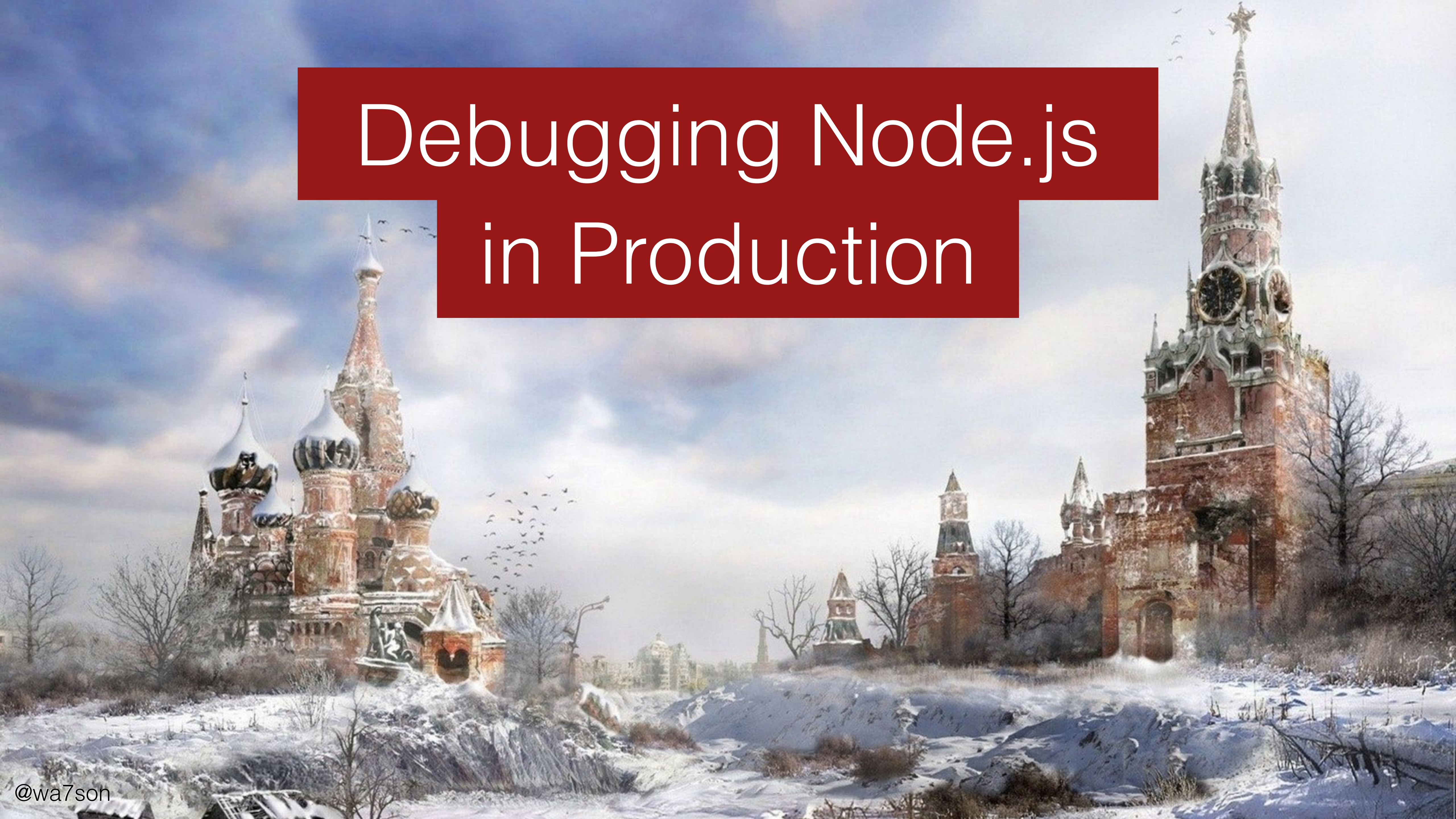


Thomas Watson

@wa7son

github.com/watson

Debugging Node.js in Production



Plan

1. Introduction
2. Debugging CPU Intensive Code
3. Debugging Crashes
4. Debugging Memory Leaks
5. Q&A

Who is this guy anyway?

- Thomas Watson
- Open Source developer at github.com/watson
- Node.js Lead at Opbeat
- Member of the Diagnostics Working Group under the Node.js Foundation
- Tweets as @wa7son



Prior Work

**Yes,
that Joyent!**



- **Brendan Gregg**, Netflix
- Previously: Sun Microsystems + Joyent
- LISA Outstanding Achievement Award "For contributions to the field of system administration, particularly groundbreaking work in systems performance analysis methodologies."

Debugging Node.js in Production

Why is that important?

Debugging Node.js in Production

Why is it harder?

A black cat is resting its head on the keyboard of a silver laptop, which is placed on a wooden table. The background shows a kitchen with a wooden countertop and various items. A semi-transparent red rectangle is overlaid on the image, containing the text "CPU Intensive Code" in white.

CPU Intensive Code

Causes of Slowness

- (Single-threaded)
- CPU intensive code
- Slow I/O
- Event Loop saturation
- Running out of memory
- Garbage Collection

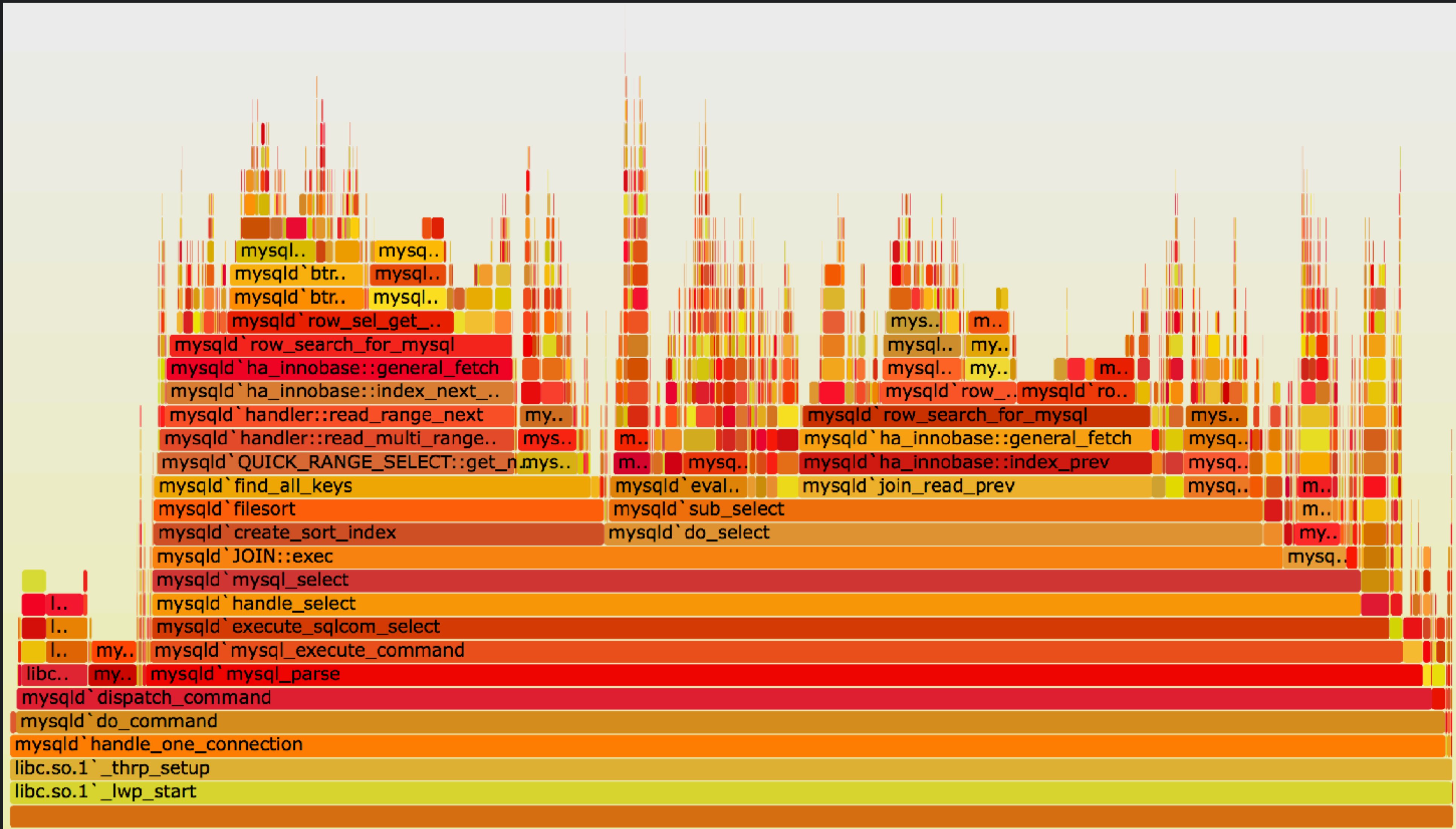
CPU Intensive Code

- Synchronous I/O
- JSON.parse
- Regular Expressions
- Crypto
- Templates
- ...



(Flame Graph)

Flame Graph



Q: How can we know what part of our program takes up most CPU time?

A: Ask the CPU periodically what it's working on

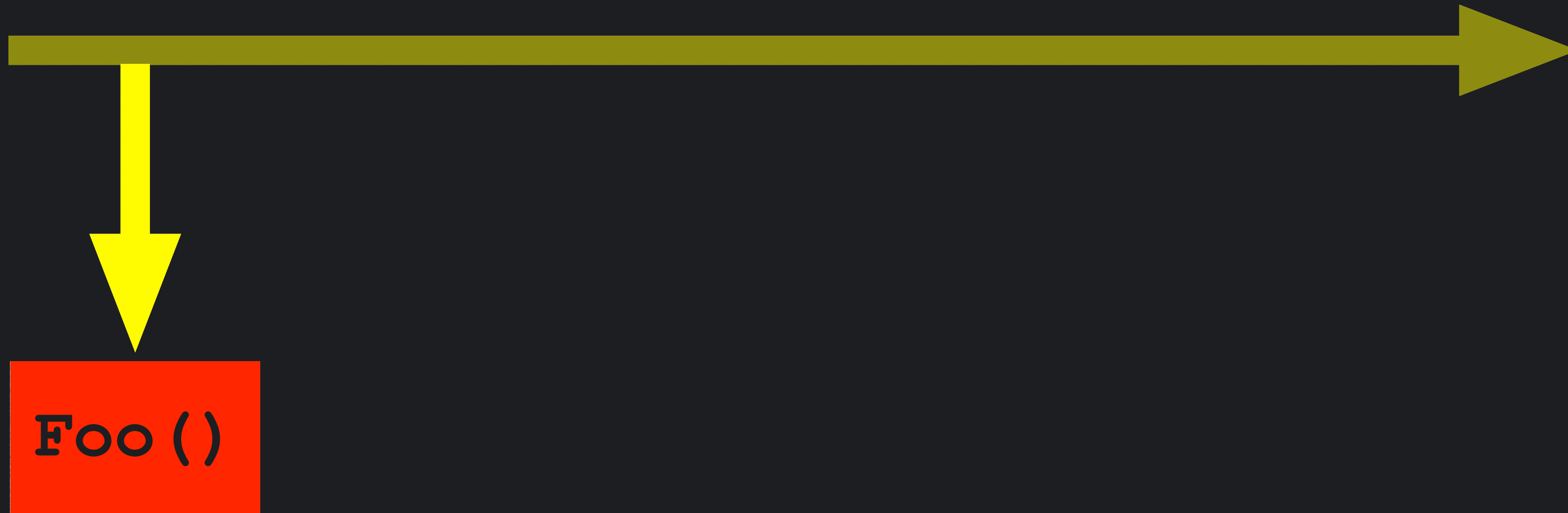
CPU Sampling



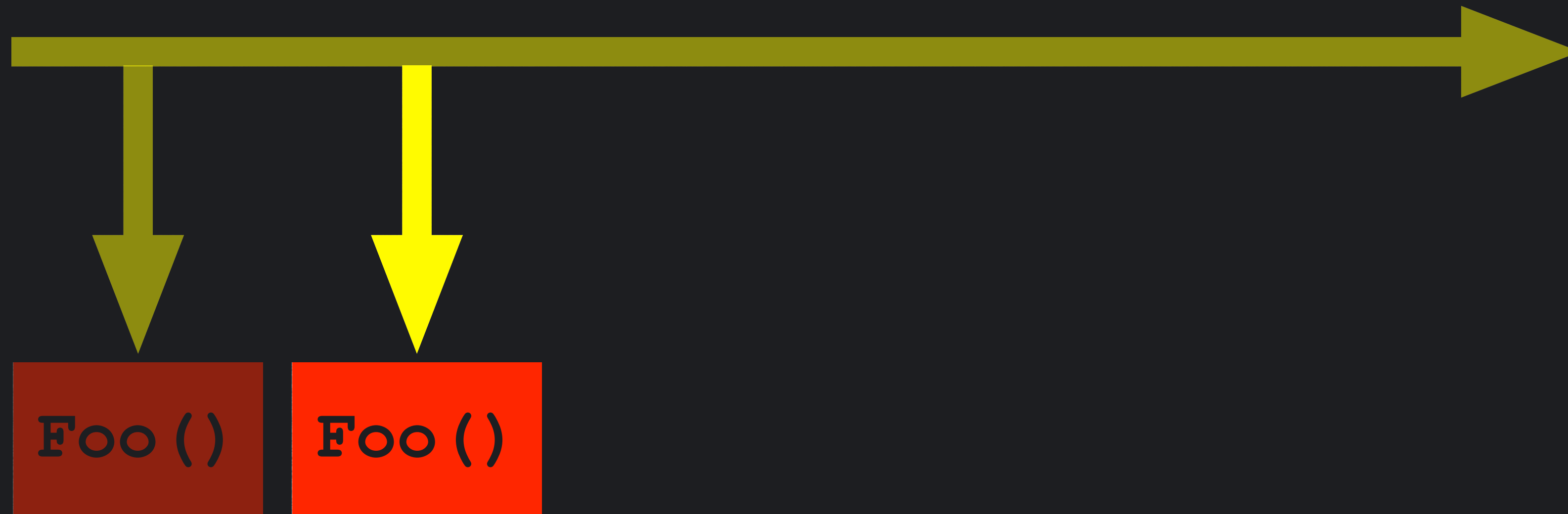
What is CPU Sampling?



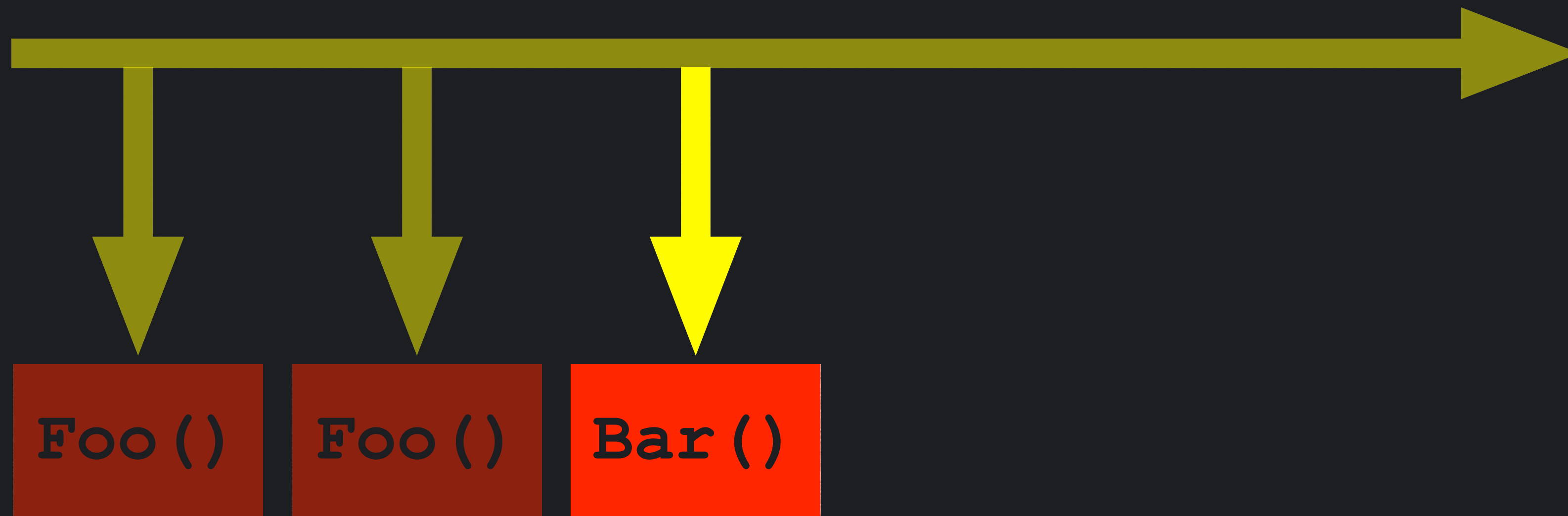
What is CPU Sampling?



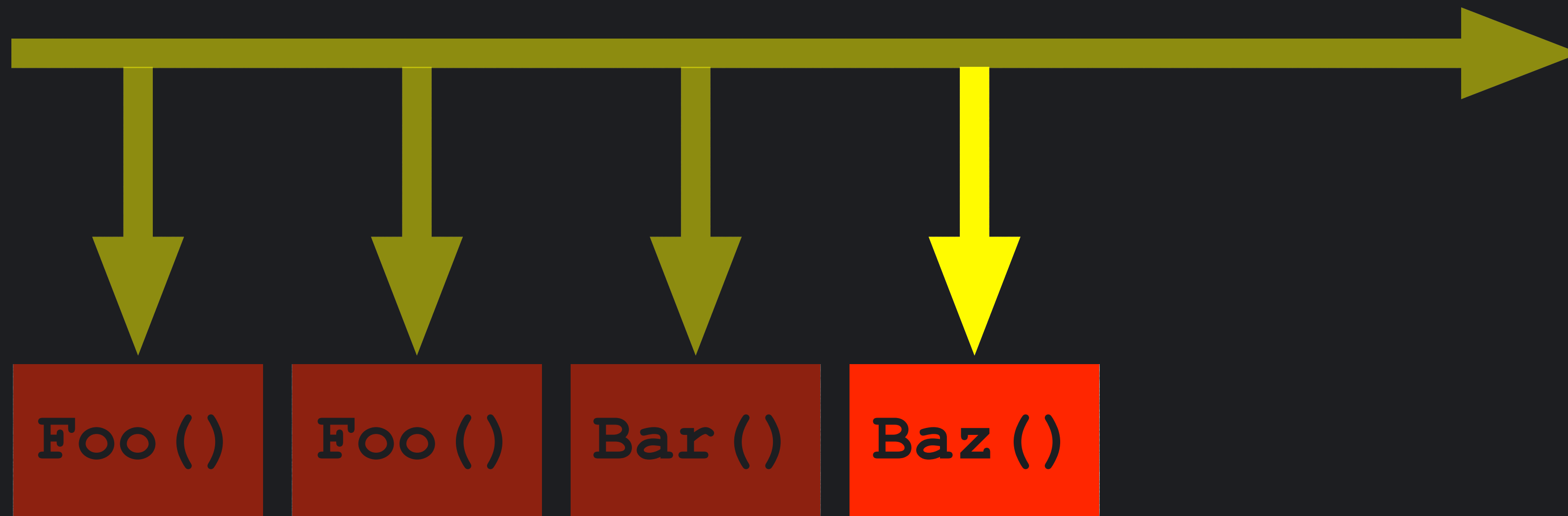
What is CPU Sampling?



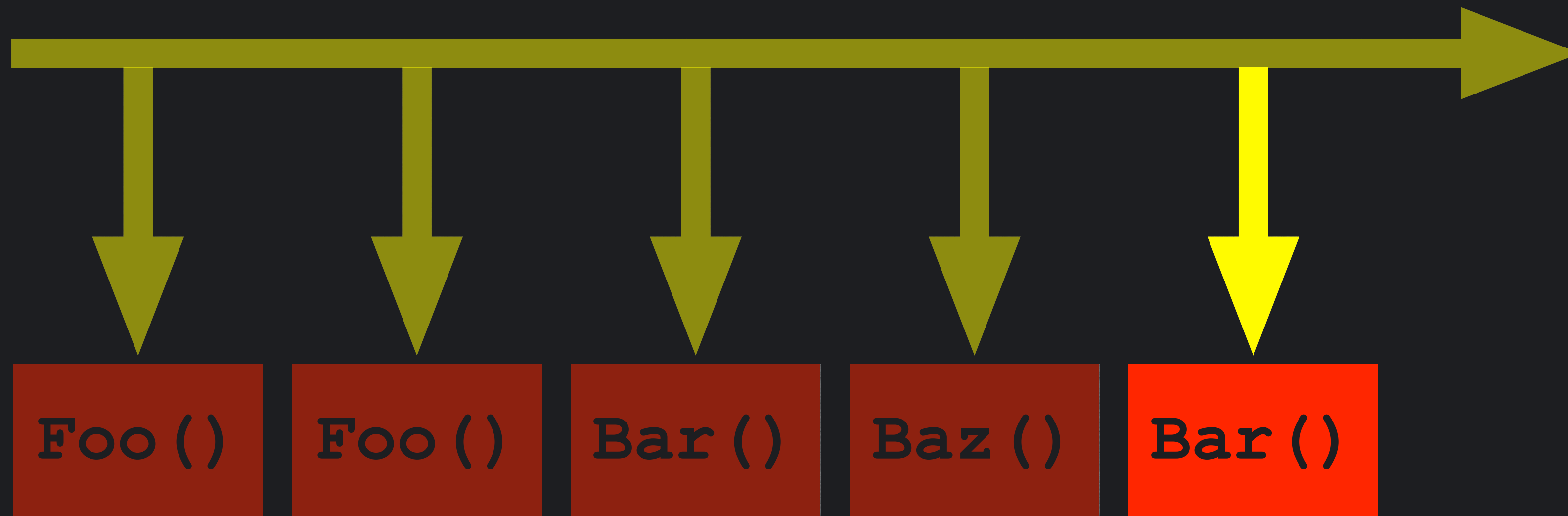
What is CPU Sampling?



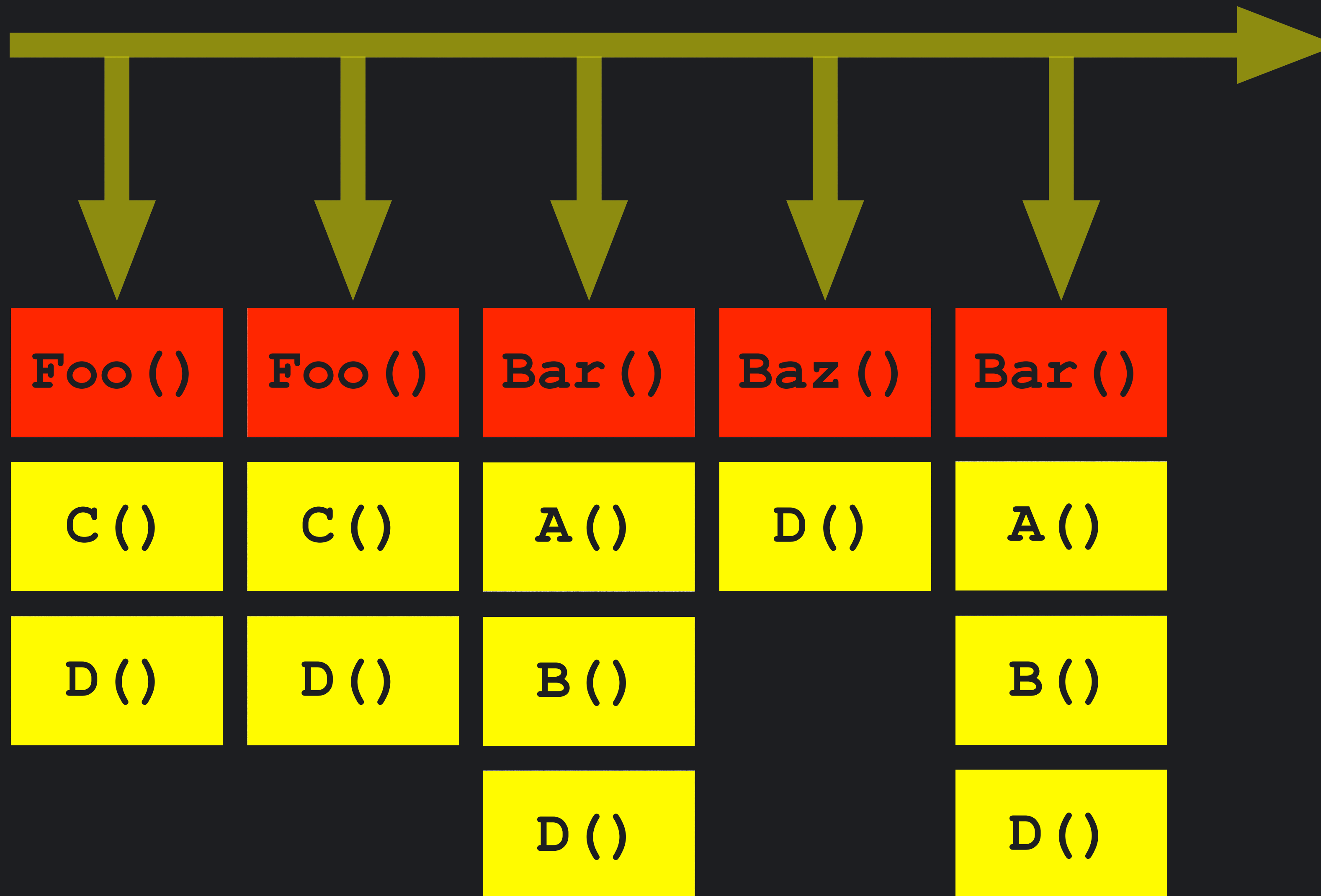
What is CPU Sampling?



What is CPU Sampling?

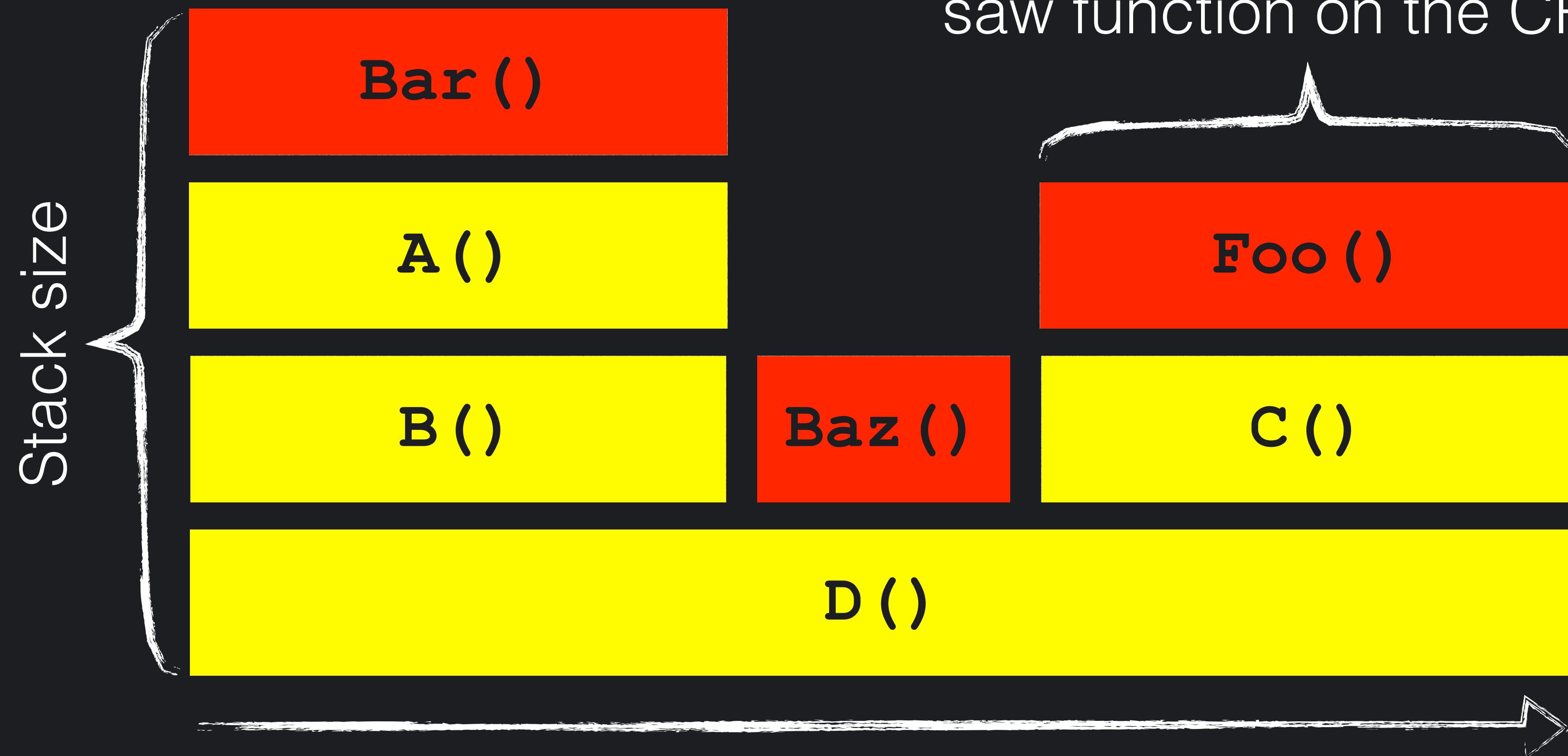


What is CPU Sampling?

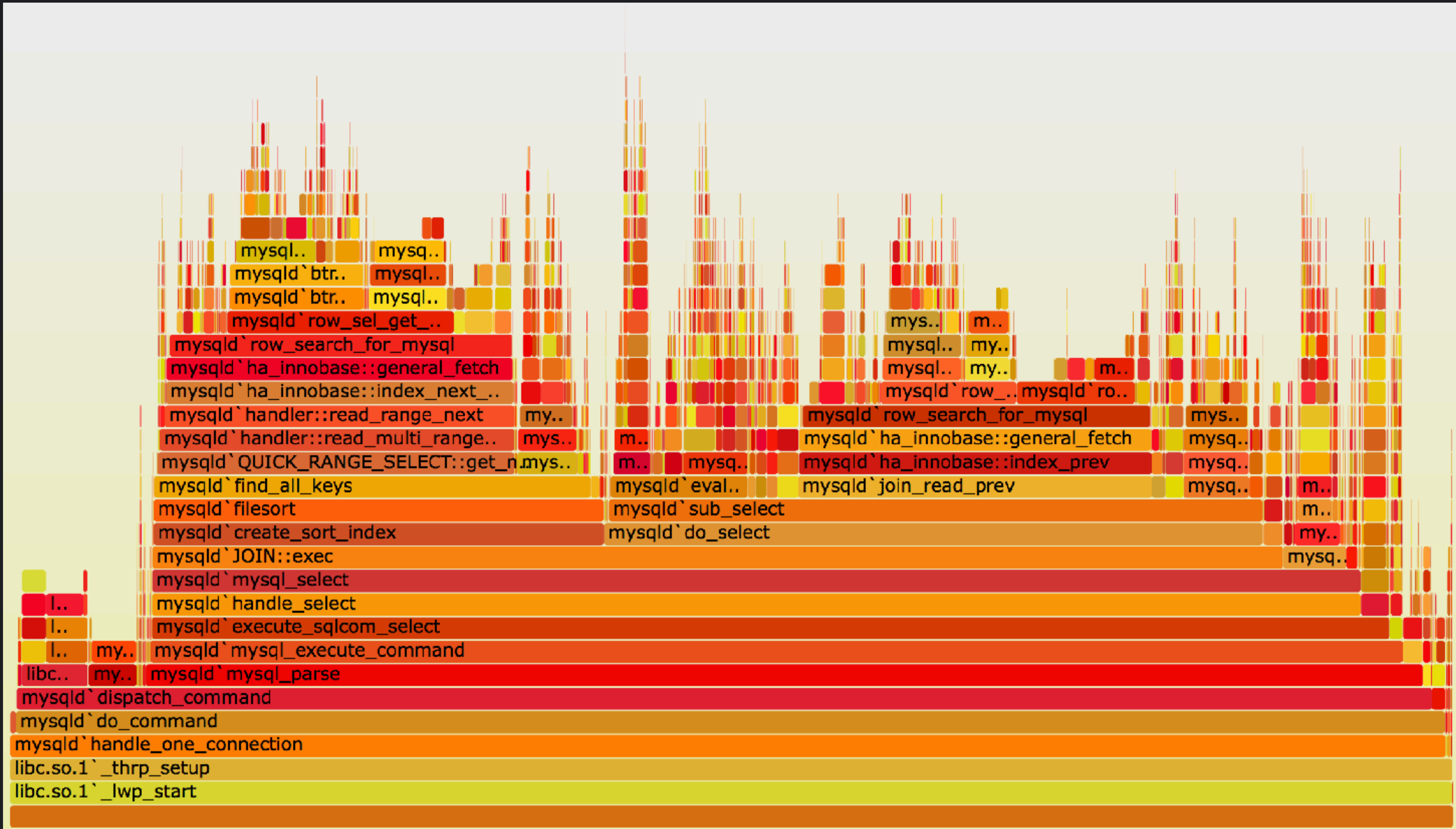


Flame Graph

Width of bar: Relative frequency of which we saw function on the CPU



Flame Graph



We'll use it to sample
stack traces from
our node process



PERF (1)

Performance analysis tools for Linux

Sample Stack Traces

`perf record -F 99 -p <pid> -g -- sleep 30`

Record profile to file

99 samples per second

Process to sample

Record full stack trace

Run for 30 seconds

The diagram shows the command `perf record -F 99 -p <pid> -g -- sleep 30` with four yellow arrows pointing to specific options and their meanings:

- An arrow from `-F` points to the text "99 samples per second".
- An arrow from `-p` points to the text "Process to sample".
- An arrow from `-g` points to the text "Record full stack trace".
- An arrow from `-- sleep 30` points to the text "Run for 30 seconds".
- An additional arrow from the text "Record profile to file" points to the `record` command itself.

Sample Stack Traces

```
perf record -F 99 -p <pid> -g -- sleep 30
```



```
perf.data
```

\$> perf script

watson@ubuntu:~/1-cpu\$



Sample Stack Trace

```
751dcc v8::internal::Compiler::Compile (/usr/local/bin/node)
a84a96 v8::internal::Runtime_CompileLazy (/usr/local/bin/node)
223055f092a7 [unknown] (/tmp/perf-27806.map)
223055f34338 [unknown] (/tmp/perf-27806.map)
22305609a84f [unknown] (/tmp/perf-27806.map)
223056021453 [unknown] (/tmp/perf-27806.map)
223056020ebb [unknown] (/tmp/perf-27806.map)
223055f09895 [unknown] (/tmp/perf-27806.map)
223056098b9f [unknown] (/tmp/perf-27806.map)
22305609669f [unknown] (/tmp/perf-27806.map)
223055f3b7c3 [unknown] (/tmp/perf-27806.map)
223055f2508f [unknown] (/tmp/perf-27806.map)
84f994 v8::internal::Execution::Call (/usr/local/bin/node)
570a99 v8::Function::Call (/usr/local/bin/node)
57ec51 v8::Function::Call (/usr/local/bin/node)
c86a79 node::AsyncWrap::MakeCallback (/usr/local/bin/node)
cc9fd7 node::Parser::on_headers_complete (/usr/local/bin/node)
f0444b http_parser_execute (/usr/local/bin/node)
```

JavaScript frames

command: perf script

Create perf map files

```
node --perf_basic_prof_only_functions server.js
```

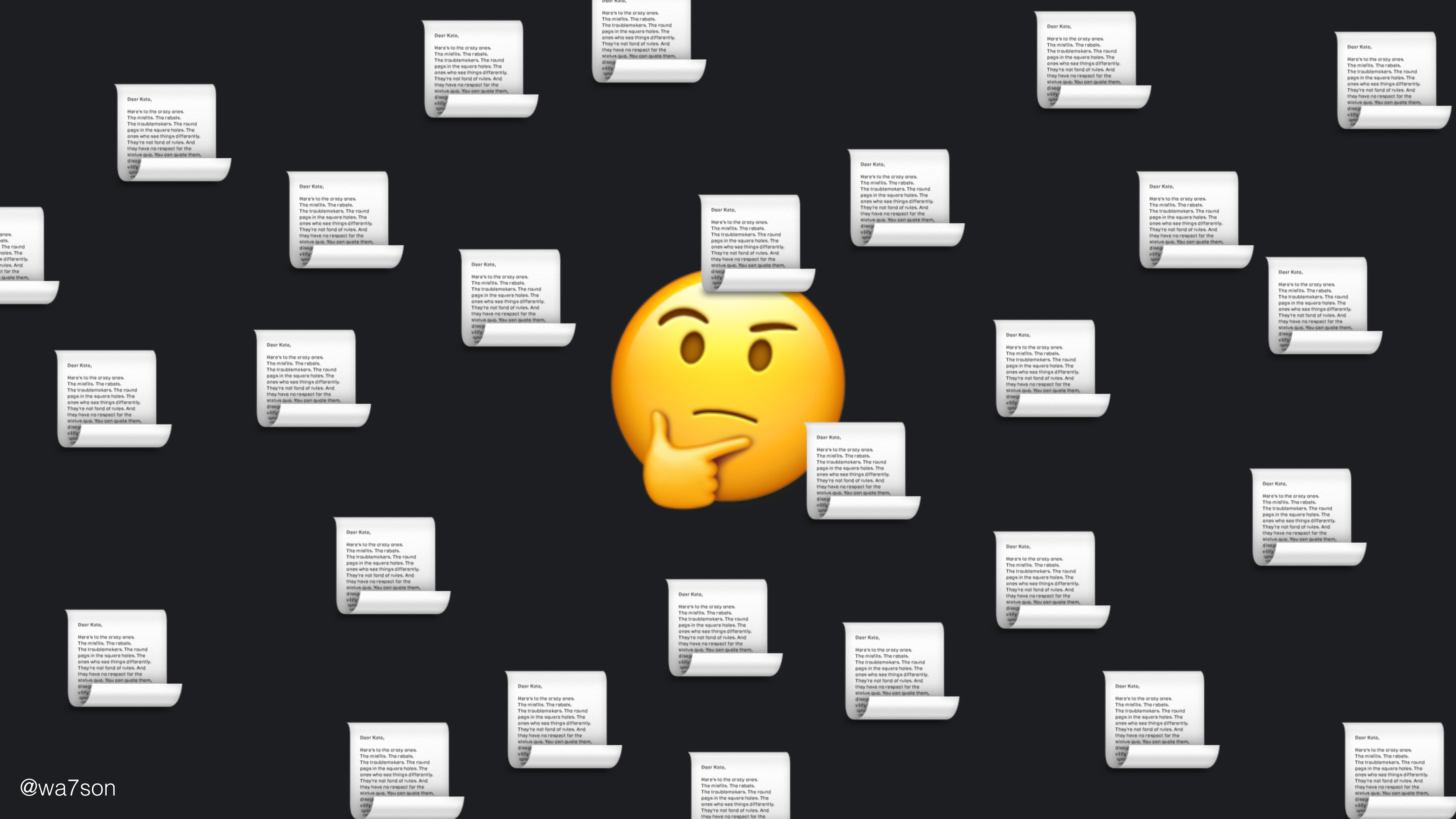


```
/tmp/perf-<pid>.map
```


Sample Stack Trace with JavaScript frames

```
5e6f22 v8::internal::(anonymous namespace)::HandleApiCallHelper<false> (/usr/local/bin/node)
5e759e v8::internal::Builtin_HandleApiCall (/usr/local/bin/node)
17712b1092a7 [unknown] (/tmp/perf-27742.map)
17712b29c49d LazyCompile:~pbkdf2 crypto.js:571 (/tmp/perf-27742.map)
17712b109895 [unknown] (/tmp/perf-27742.map)
17712b29c02f LazyCompile:~exports.pbkdf2Sync crypto.js:562 (/tmp/perf-27742.map)
17712b2ac721 LazyCompile:~foo /home/watson/1-cpu/cpu.js:13 (/tmp/perf-27742.map)
17712b29b4a3 LazyCompile:~ /home/watson/1-cpu/cpu.js:4 (/tmp/perf-27742.map)
17712b2221b3 LazyCompile:~emitTwo events.js:104 (/tmp/perf-27742.map)
17712b221cdb LazyCompile:~emit events.js:136 (/tmp/perf-27742.map)
17712b109895 [unknown] (/tmp/perf-27742.map)
17712b29979f LazyCompile:~parserOnIncoming _http_server.js:463 (/tmp/perf-27742.map)
17712b29729f LazyCompile:~parserOnHeadersComplete _http_common.js:45 (/tmp/perf-27742.map)
17712b13b7c3 [unknown] (/tmp/perf-27742.map)
17712b12508f [unknown] (/tmp/perf-27742.map)
84f994 v8::internal::Execution::Call (/usr/local/bin/node)
570a99 v8::Function::Call (/usr/local/bin/node)
57ec51 v8::Function::Call (/usr/local/bin/node)
c86a79 node::AsyncWrap::MakeCallback (/usr/local/bin/node)
cc9fd7 node::Parser::on_headers_complete (/usr/local/bin/node)
f0444b http_parser_execute (/usr/local/bin/node)
```

command: perf script



Cue for Demo

Recap

perf

- `apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r``
- `node --perf_basic_prof_only_functions server.js`
- `perf record -F 99 -p <pid> -g -- sleep 30`
- `perf script > stacks.out`

0x

- `npm install 0x -g`
- `0x -c gen stacks.out > flamegraph.html`

Alternative Generators

[github.com / brendangregg / FlameGraph](https://github.com/brendangregg/FlameGraph)

[github.com / thlorenz / flamegraph](https://github.com/thlorenz/flamegraph)

Bonus Tip #1

npm install blocked

```
var blocked = require('blocked')

blocked(function (ms) {
  console.log('BLOCKED FOR %sms', ms | 0)
})
```

Bonus Tip #2

```
npm install block-trace -g
```


Bonus Tip #2

```
npm install block-trace -g
```

```
console.log('Waiting 1s ...')  
  
setTimeout(function () {  
    console.log('Spinning the CPU now!')  
    while (1) {}  
}, 1000)
```

Bonus Tip #2

```
npm install block-trace -g
```

```
console.log('Waiting 1s ...')

setTimeout(function () {
  console.log('Spinning the CPU now!')
  while (1) {}
}, 1000)
```

```
block-trace node example.js
```

Bonus Tip #2

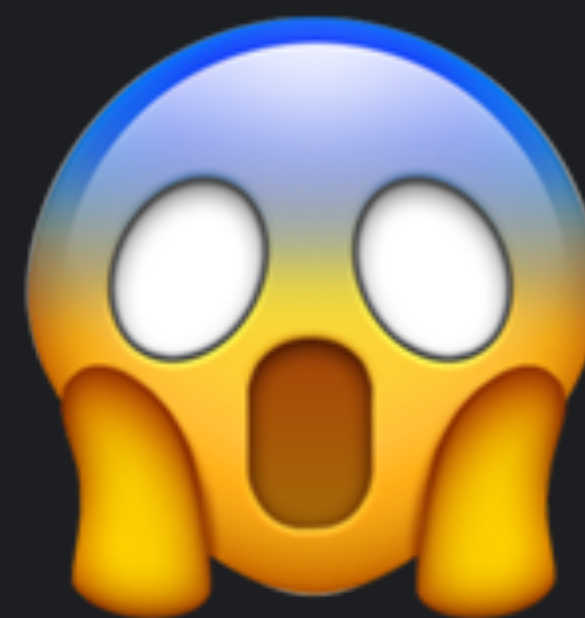
```
npm install block-trace -g
```

```
block-trace node example.js
```

```
Waiting 1s ...  
Spinning the CPU now!  
Error: CPU is blocked  
    at /Users/watson/code/node_modules/block-trace/example.js:5:10  
    at Timer.listOnTimeout (timers.js:92:15)
```


A black and white photograph of a supermarket aisle. In the foreground, a person wearing a full-body panda costume is walking away from the camera. To the left, a man in a light-colored short-sleeved shirt and dark trousers is standing and looking towards the right. Behind him, another person is partially visible. The aisle is filled with shelves of various products, and a shopping cart is visible in the middle ground. A semi-transparent red rectangle is overlaid in the center of the image, containing the text "Post-mortem Debugging" in white.

Post-mortem Debugging



Reproducibility

```
function fn () {  
    // ...  
}  
  
{  
    foo: 42,  
    bar: [1, 2, 3],  
    baz: {...}  
}
```

fn1 ()

fn2 ()

fn3 ()

Core Dump





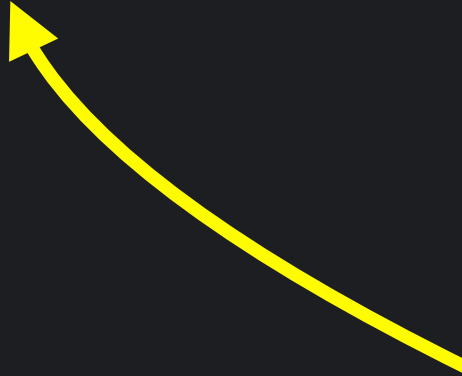


Tell Node.js to make a core dump

```
node --abort_on_uncaught_exception server.js
```

Tell your OS to allow core dumps

```
ulimit -c unlimited
```



Max size of
core dump

~/core.<pid>

48	54	54	50	2f	31	2e	31	20	32	30	30	20	4f	4b	0d
0a	44	61	74	65	3a	20	32	30	31	36	2d	31	32	2d	31
31	0d	0a	0d	0a	43	6f	6e	67	72	61	74	75	6c	61	74
69	6f	6e	73	21	20	59	6f	75	20	73	68	6f	75	6c	64
20	61	70	70	6c	79	20	66	6f	72	20	6a	6f	62	20	61
74	20	4f	70	62	65	61	74	20	3a	29	0d	0a	57	65	27
72	65	20	61	6c	77	61	79	73	20	6c	6f	6f	6b	69	6e
67	20	66	6f	72	20	74	61	6c	65	6e	74	65	64	20	64
65	76	65	6c	6f	70	65	72	73	2e	0d	0a	45	6d	61	69
6c	20	72	6f	6e	40	6f	70	62	65	61	74	2e	63	6f	6d

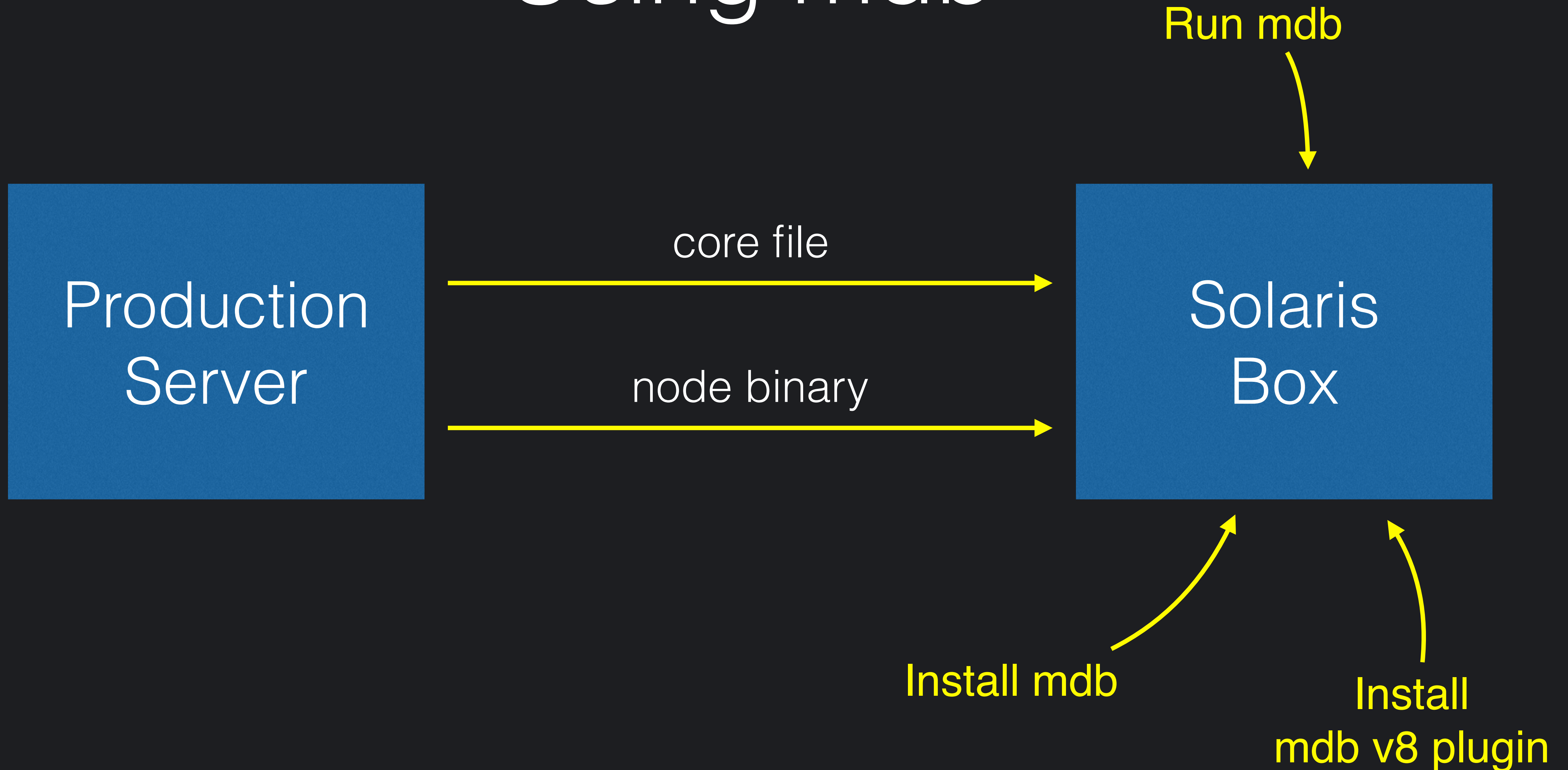
mdb

Modular Debugger - *An extensible, low-level debugger developed by Sun Microsystems for the Solaris 7 operation system*

Using mdb

- Solaris variant
- mdb (runs only on Solaris and friends)
- mdb v8 - https://github.com/joyent/mdb_v8
- core file
- node binary

Using mdb



Autopsy

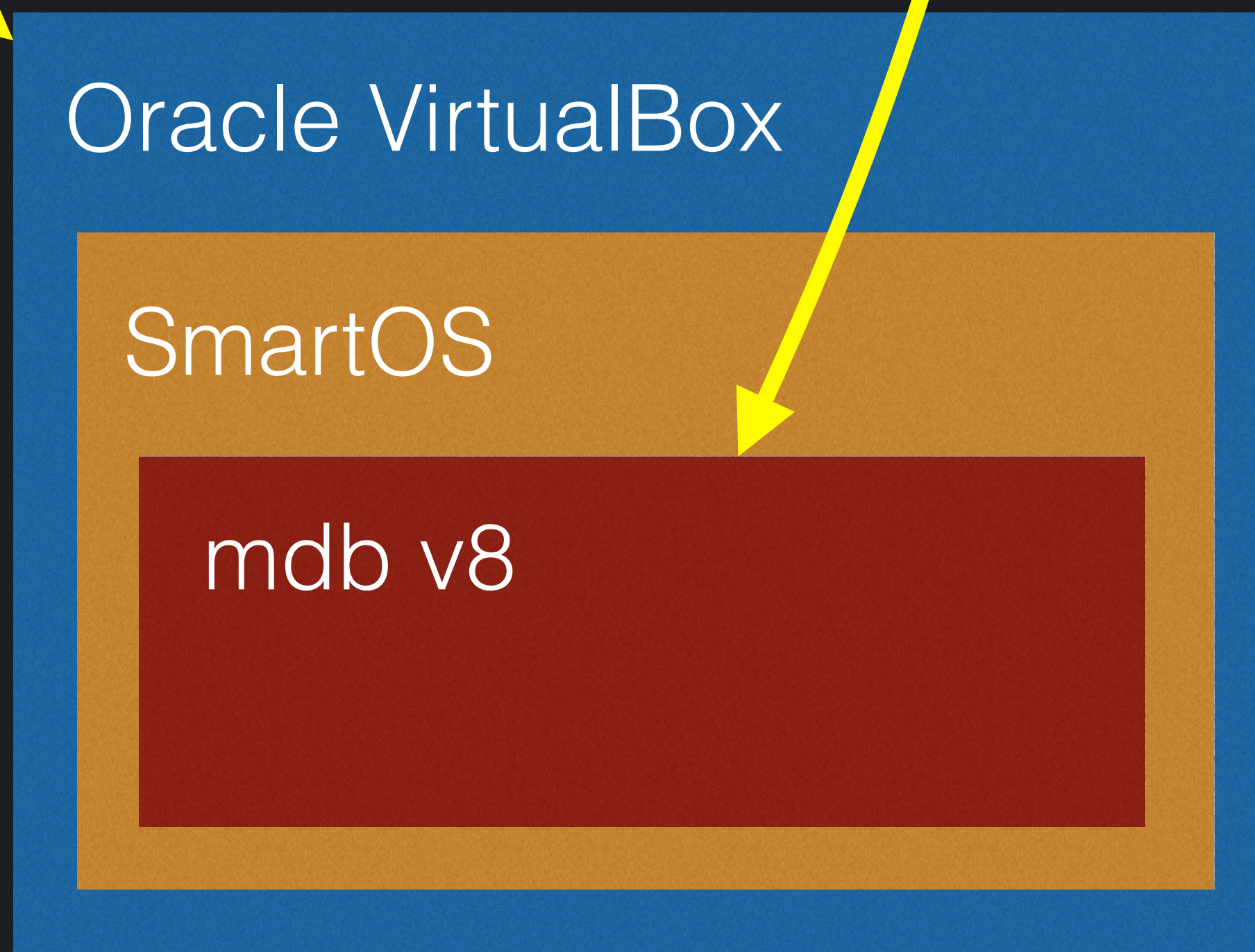
```
JS Commands > ::dmods -l ! grep js  
V8 Commands > ::dmods -l ! grep v8  
Useful: ::status, ::dump, ::help <cmd>
```

<https://github.com/nearform/autopsy>

AUTOPSY

VirtualBox is a
Prerequisite

SSH ALL THE THINGS!!!



SmartOS + mdb
are installed by
Autopsy

Cue for Demo

Recap 1/2

Production Server

- `ulimit -c unlimited`
- `node --abort-on-uncaught-exception server.js`

Recap 2/2

Solaris

- `mdb node core`

mdb

- `::load v8` *Load v8 features*
- `::jsstack -vn0` *Show js stack with details*
- `<addr>::jsprint` *Output content of a memory address*

See also

[github.com / tjfontaine / lldb-v8](https://github.com/tjfontaine/lldb-v8)

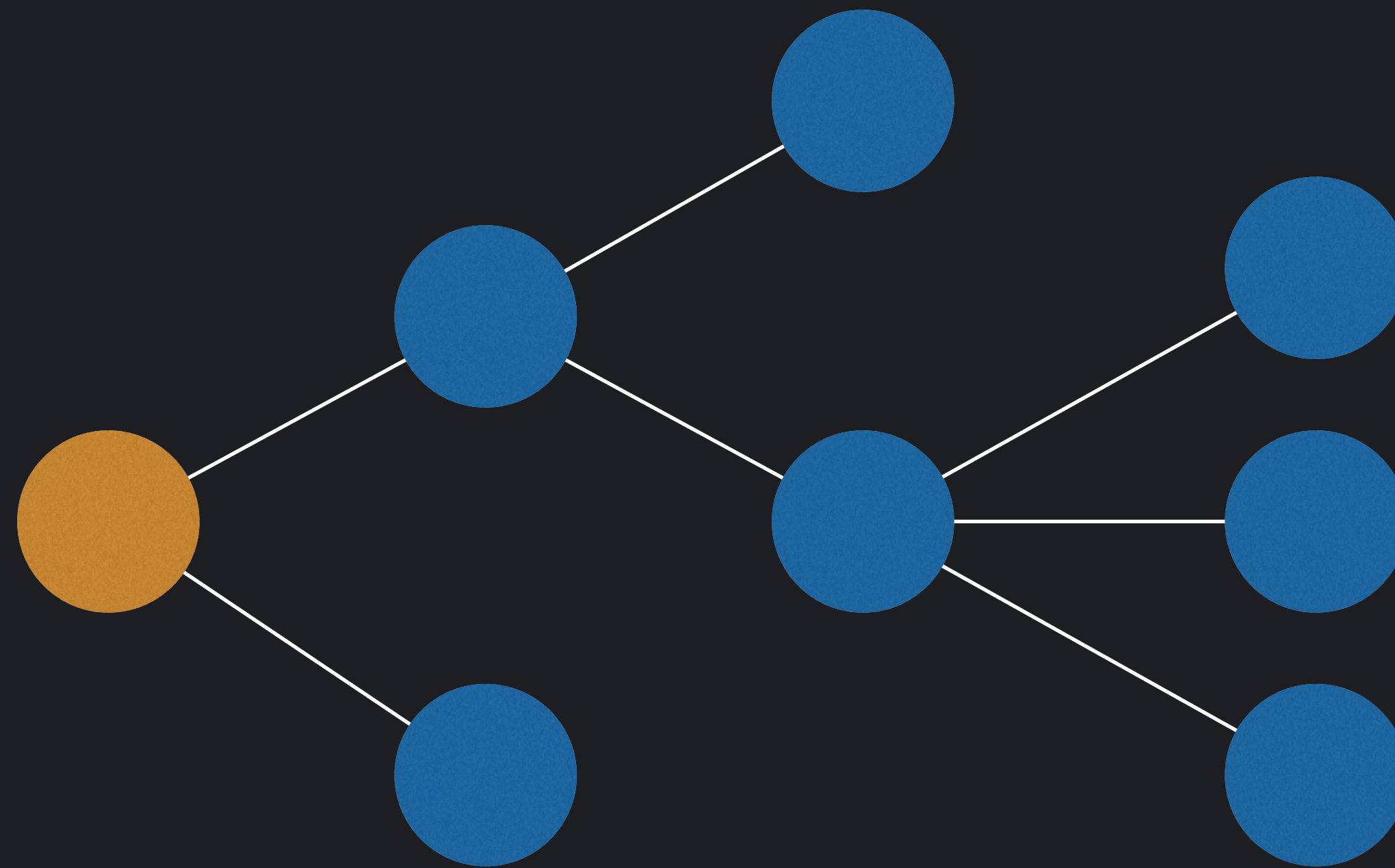
[github.com / nodejs / llnode](https://github.com/nodejs/llnode)

[github.com / nodejs / post-mortem](https://github.com/nodejs/post-mortem)

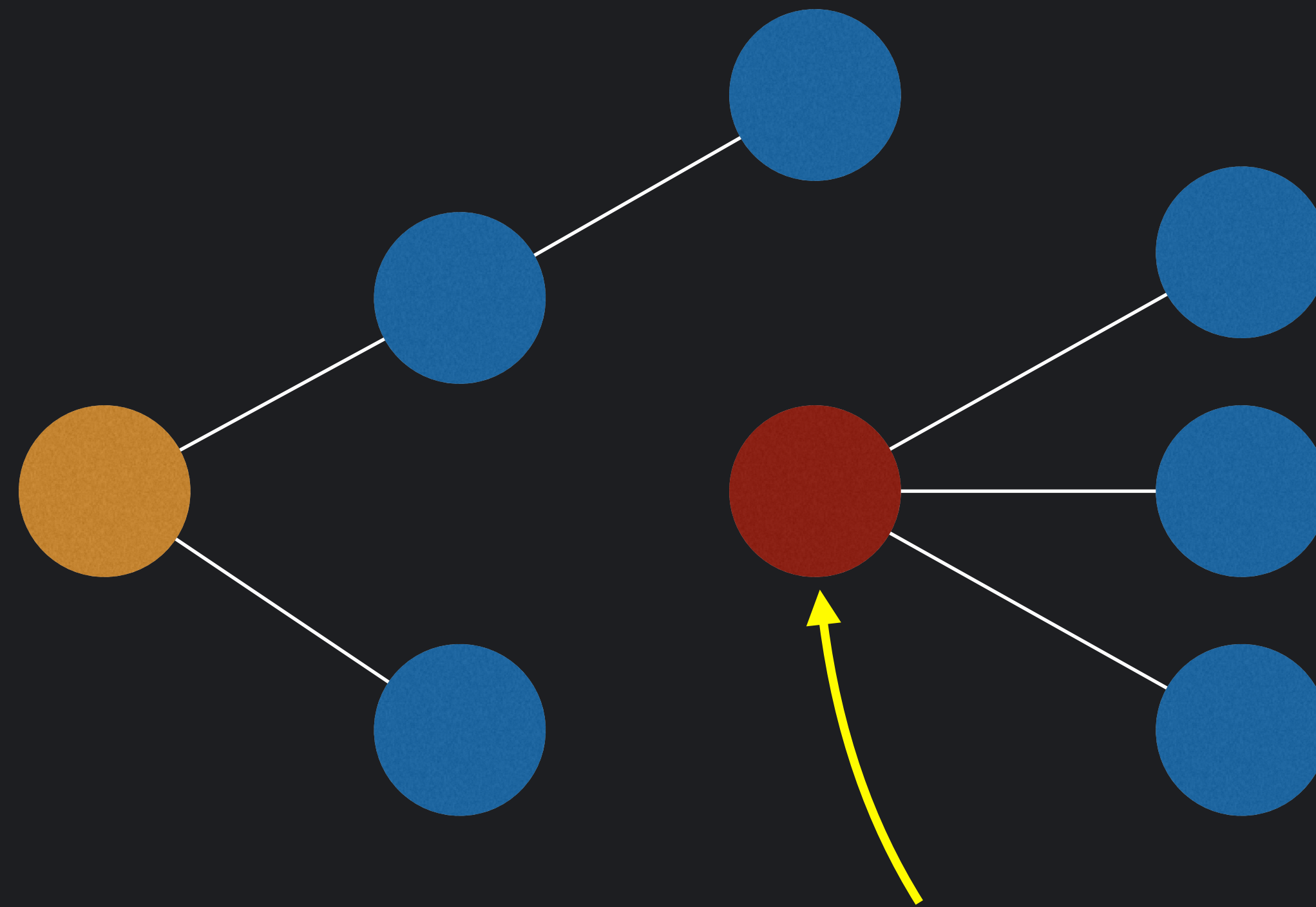
A person with dark hair, wearing a dark shirt, is pouring water from a clear plastic bottle into a glass. The background is a plain, light-colored wall. A semi-transparent red rectangle is overlaid on the image, containing the text "Memory Leaks" in white.

Memory Leaks

What are Memory Leaks in JS?

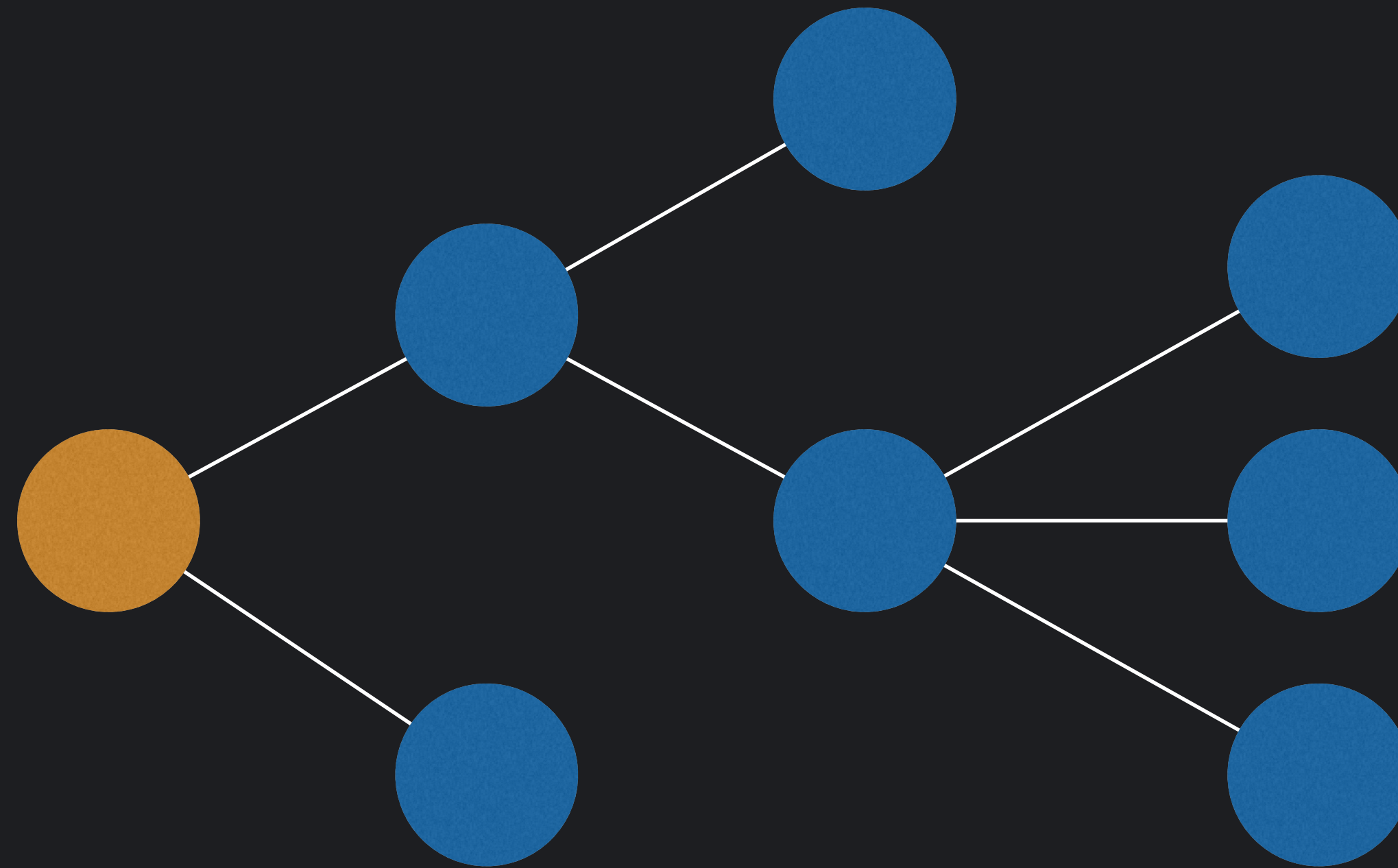


What are Memory Leaks in JS?

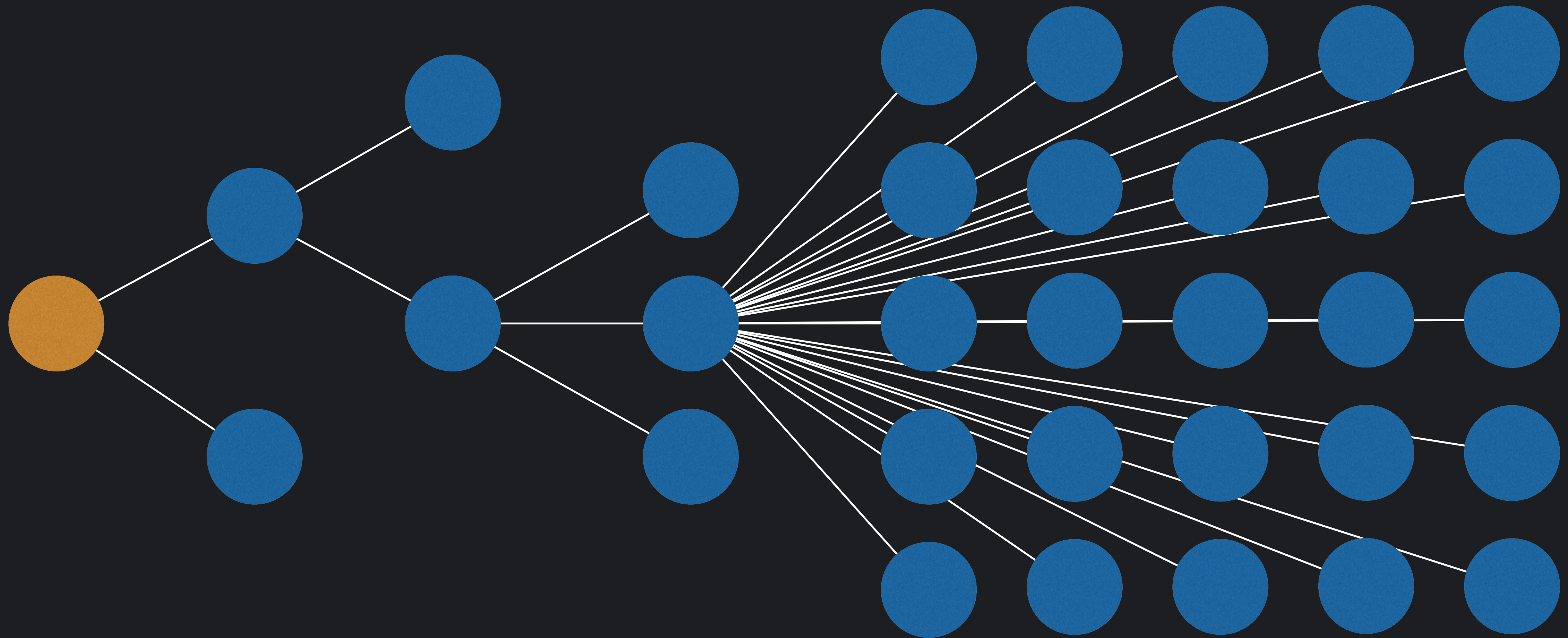


No memory leak
(will be garbage collected)

What are Memory Leaks in JS?



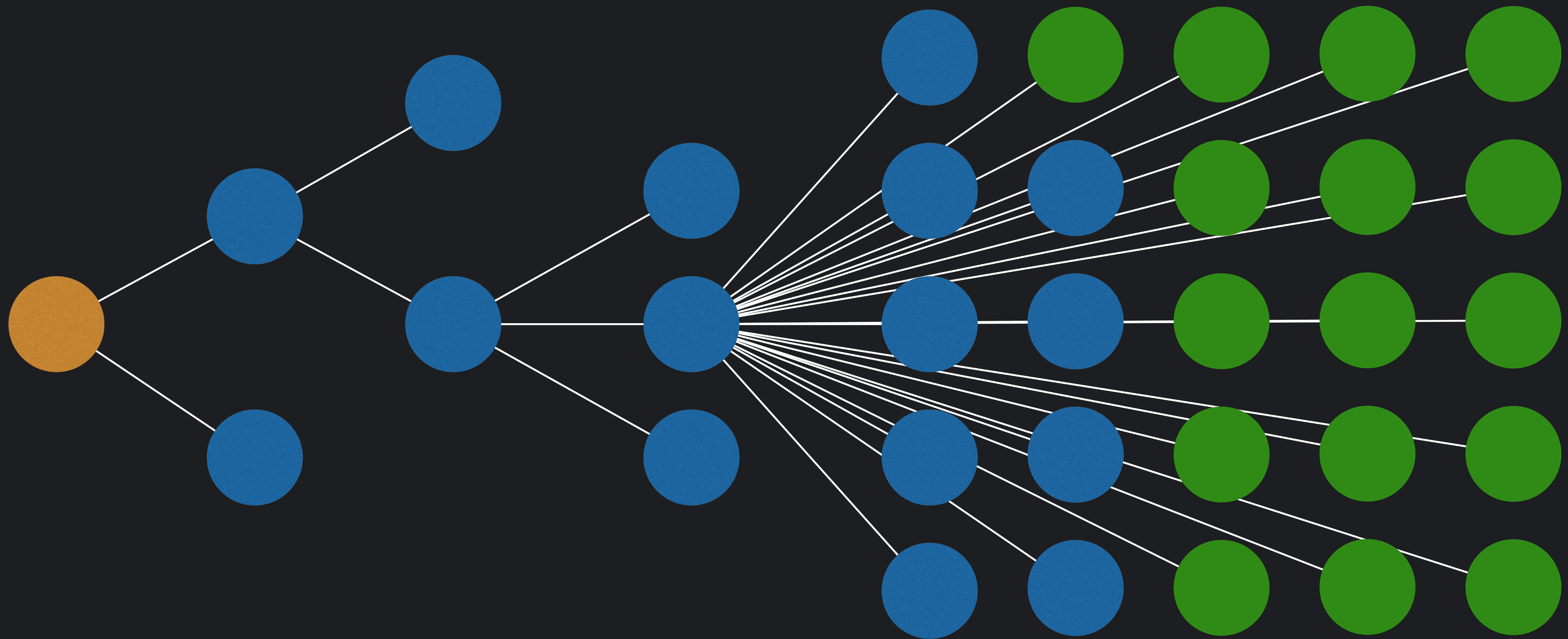
What are Memory Leaks in JS?



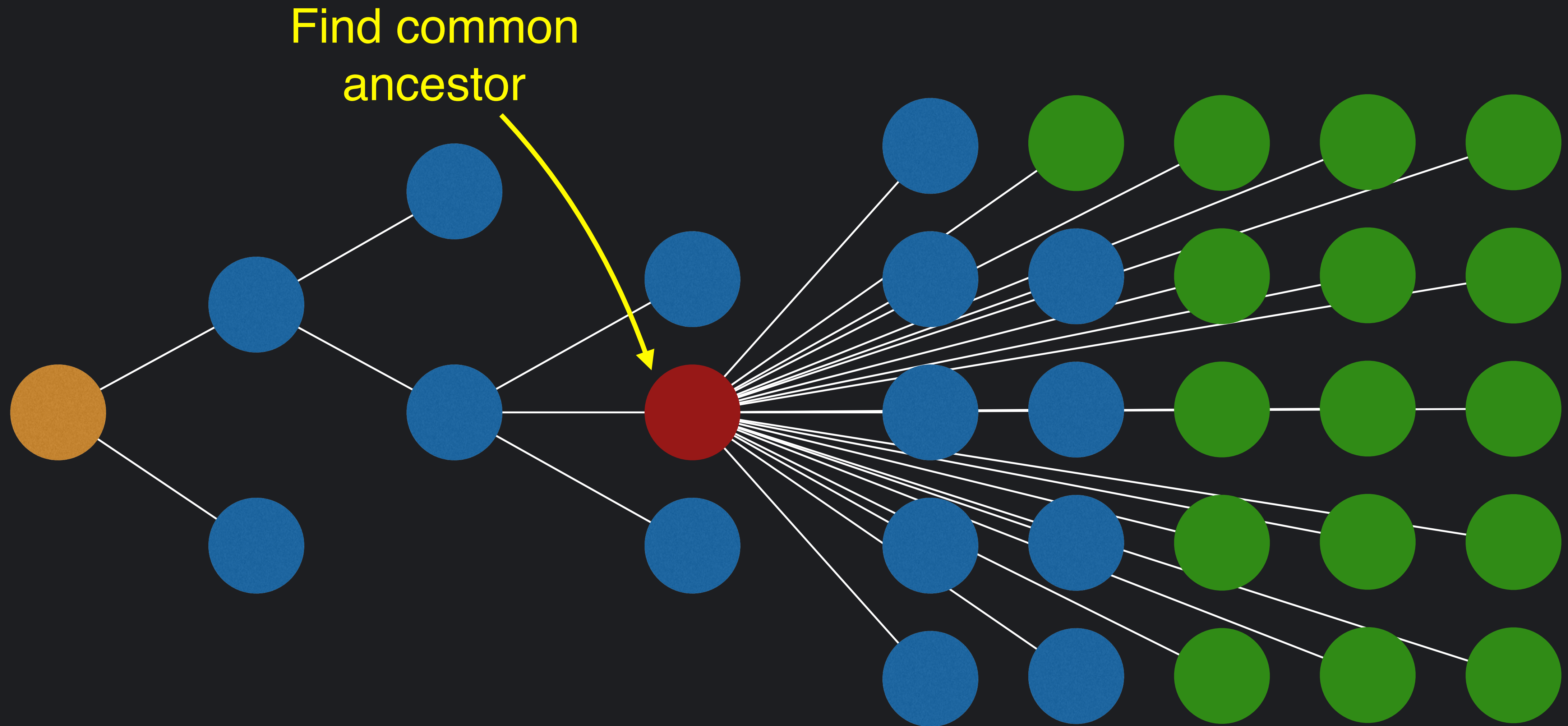
Q: How can we figure out which objects are leaking?

A: Heap dump diffing!

Heap Dump Diffing



Heap Dump Diffing



Cue for Demo

Recap

App

- `npm install heapdump --save`
- `require('heapdump')`

Production Server

- `kill -USR2 <pid>`

Google Chrome DevTools

- Profiles -> Load (2 dumps) -> Comparison

Alternative Approach

`gcore (1)`

Generate a core dump of a running program with process ID pid

Finding Leaks with gcore & mdb

Production Server

- `gcore <pid>`

Solaris

- `mdb node core`

mdb

- `::findjsobjects` *Scan entire heap, group + log objects by type*
- `::findjsobjects ! sort -k2` *Sort by count column*
- `<addr>::findjsobjects -r` *Find parent objects referring to <addr> object*

спасибо

@wa7son

github.com/watson

 opbeat

(Pssst, we're hiring)