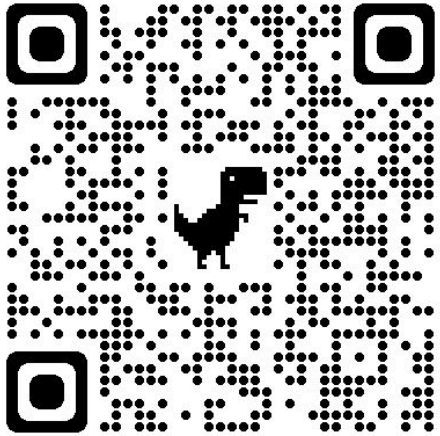
N|Solid Workshop

A powerful Node.js observability tool

NodeSource

Rafael Gonzaga

- Open Source Engineer
  at **Node|Source**
- Performance & Security
- Made in Brazil 🇧🇷



_rafaelgss

RafaelGSS

rafaelgss

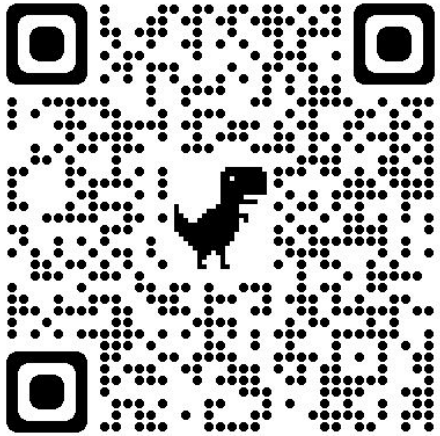
Open Source

- Node.js Technical Steering Committee (**TSC**) member
- Node.js Security Team & Node.js Releaser
- Former Fastify & Clinic.js maintainer

Juan Arboleda

- Senior Software Engineer

  at **Node|Source**



soyjuanarbol

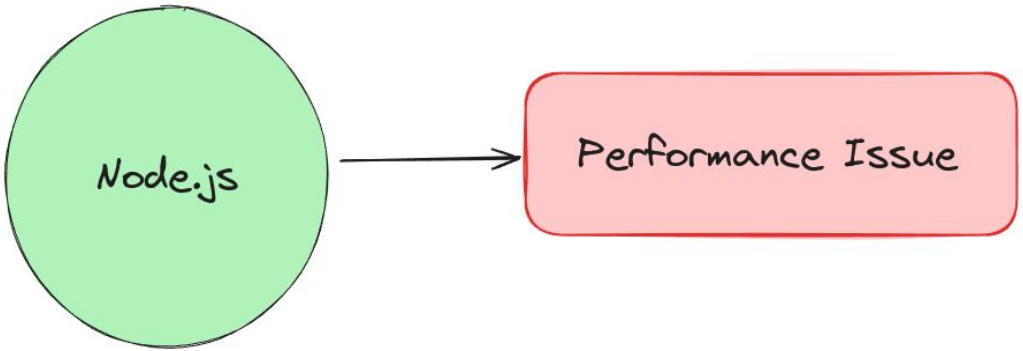
juanarbol

Open Source

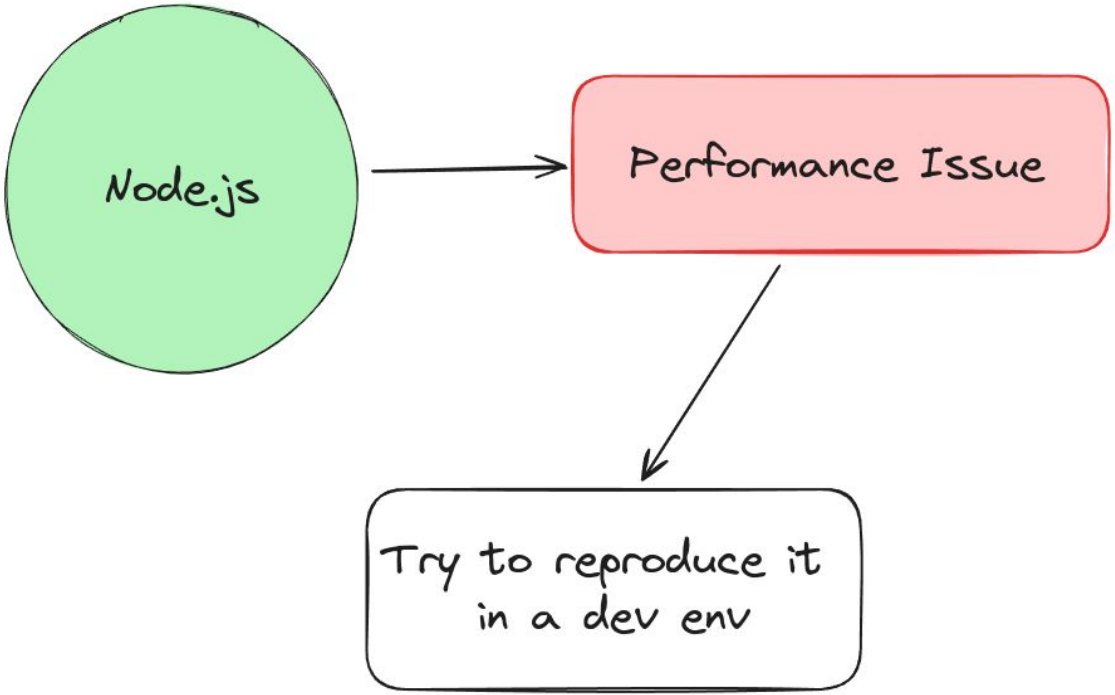
- Node.js
- V8
- Libuv
- Linux

Monitoring and finding performance bottlenecks are HARD

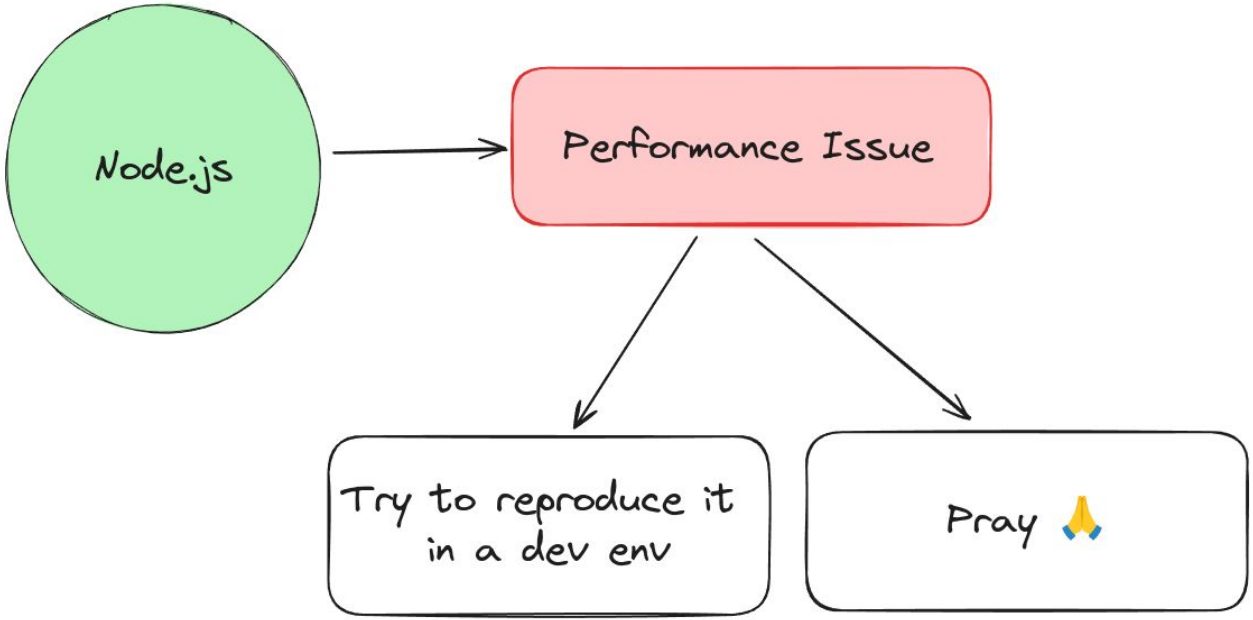
Solving a Performance Issue



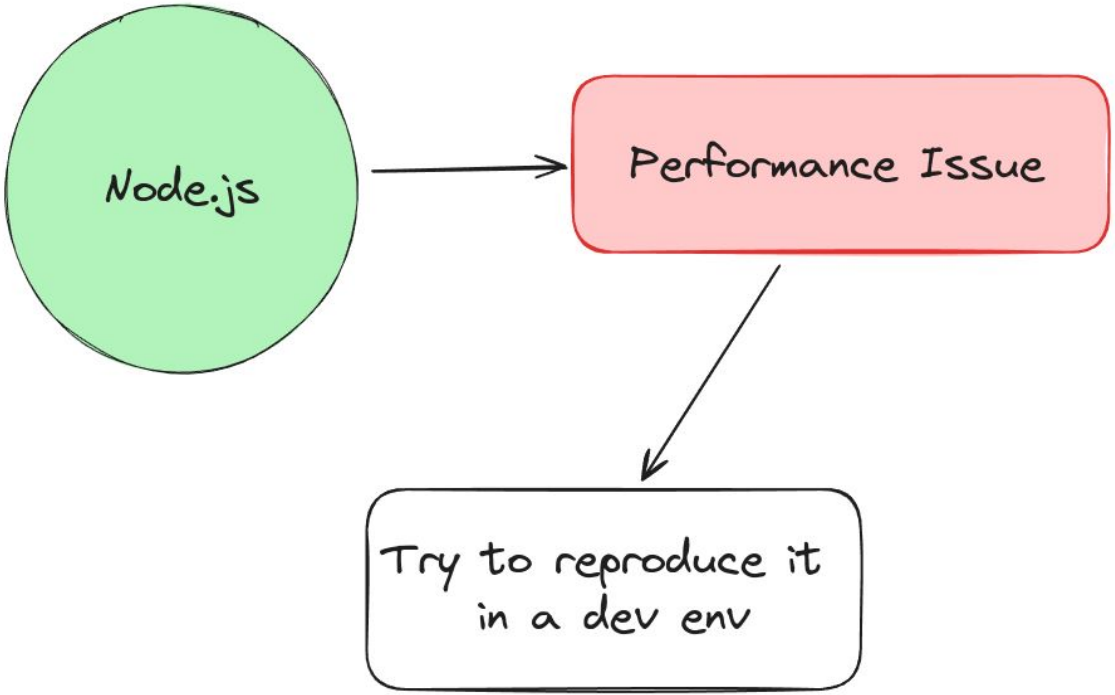
Solving a Performance Issue



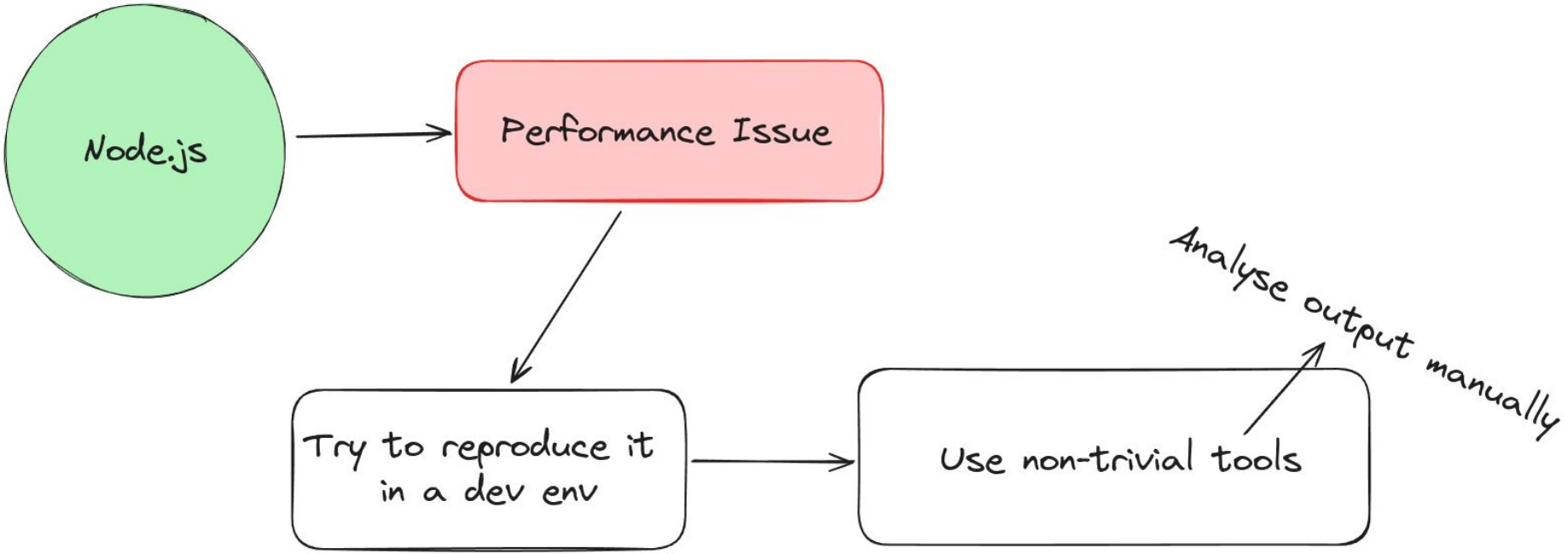
Solving a Performance Issue



Solving a Performance Issue



Solving a Performance Issue



That's what N|Solid is for!

A production and development monitoring tool

- Monitor your applications in real time
- Generate snapshots and analyse it using N|S Console
- Profile your application in production for a short period of time
- See Node.js specific events such as GC, Event Handlers, Task Queue (Libuv) and more!

1. Create your NSolid account

Juan will be passing by your tables with a card machine, 40k COP per person.

1. Create your NSolid account

~~Juan will be passing by your tables with a card machine, 40k COP per person:~~

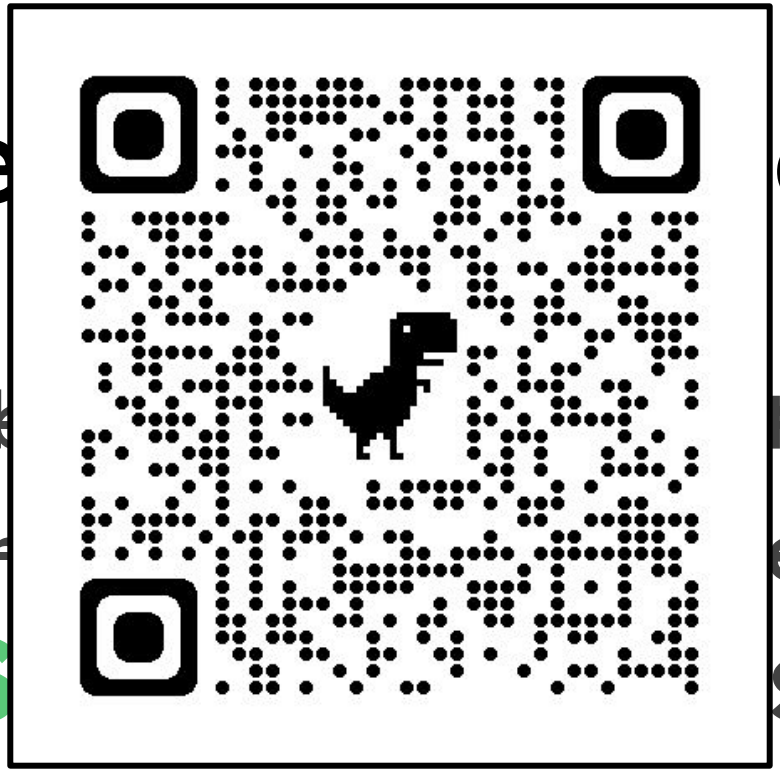
N|S is FREE to use!

1. Create account

~~Juan will k tables with a~~

~~card mach erson:~~

N|S se!



https://docs.nodesource.com/

Serveless and Node.js applications are supported!

NODE.JS APPLICATION

SERVERLESS FUNCTION

Serveless and Node.js applications are supported!

NODE.JS APPLICATION

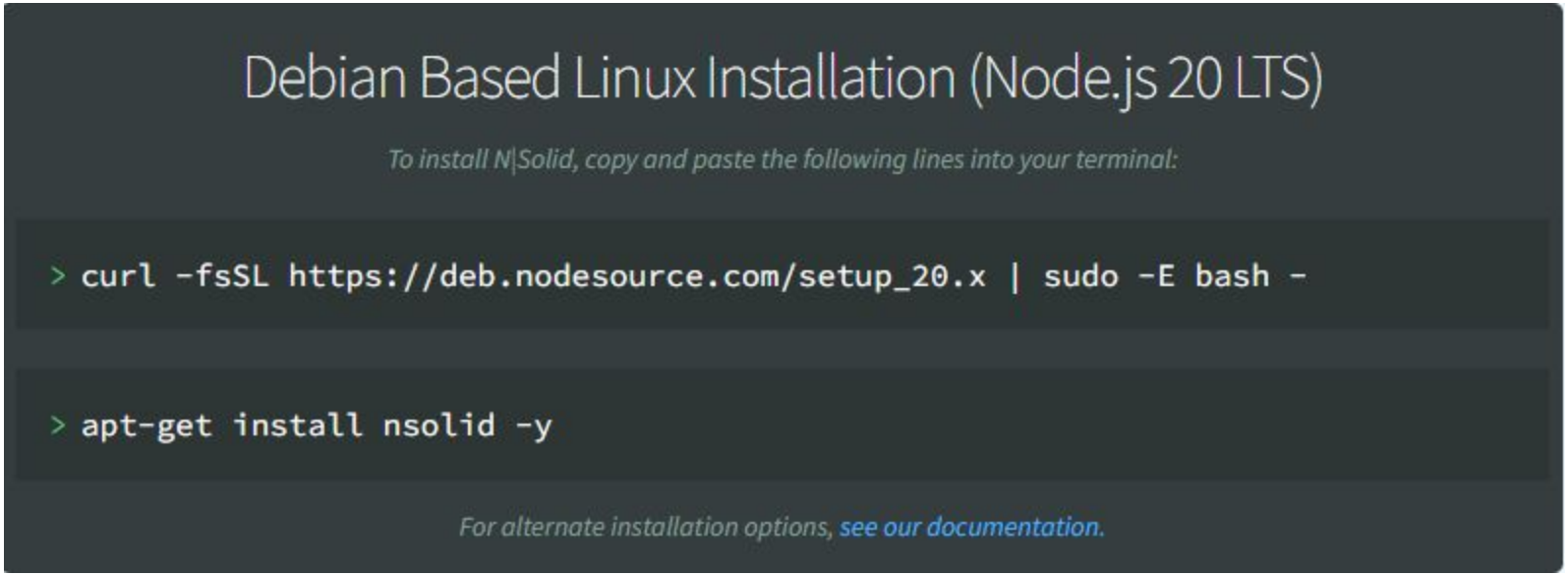
SERVERLESS FUNCTION

Install the **Runtime** or use it from **npm**

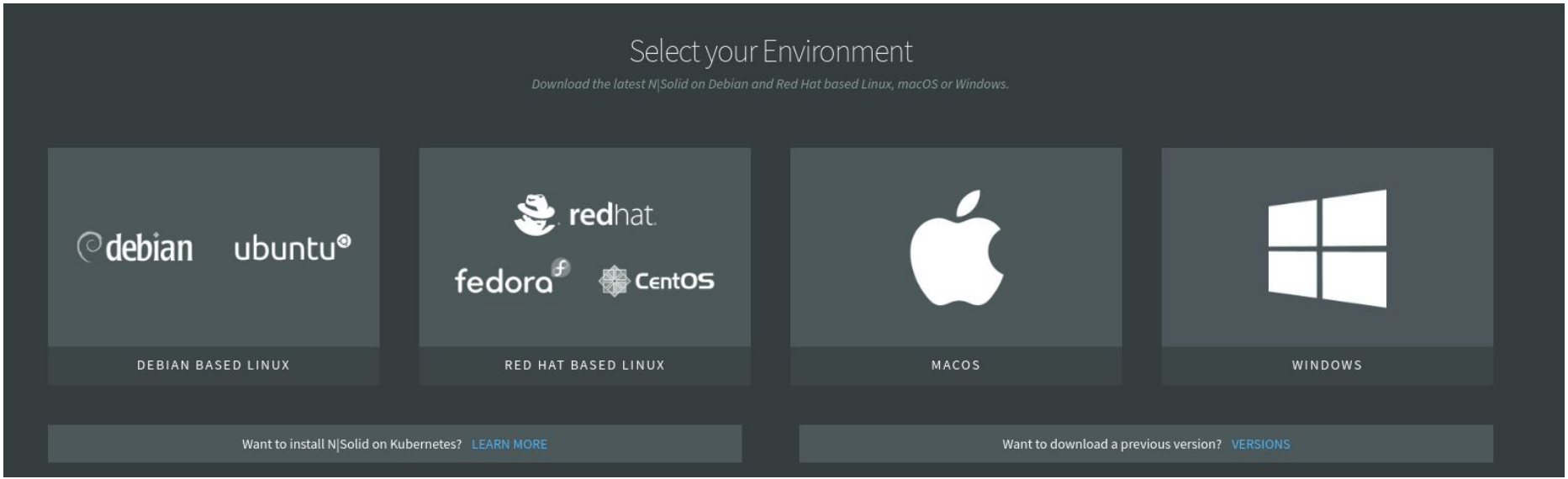
INSTALL RUNTIME

PROVIDE NPM INSTRUCTIONS

Linux installation



MacOS/Windows installation



Install as npm package

```
1 npm i -S nsolid
```

Select your SaaS plan

N|Solid is the best way to observe & monitor Node.js applications

Solid|4

4 PROCESSES
1 USERS

FREE

SELECT

Solid|8

8 PROCESSES
5 USERS

$99/mo

SELECT

Solid|12

12 PROCESSES
8 USERS

$199/mo

SELECT

Solid|E

25 to
UNLIMITED
PROCESSES

Custom Pricing

CONTACT US

Quick Install Wizard

FROM

NODESOURCE

URL for the user console

```
> https://905fe4b3-[redacted].saas.nodesource.io/welcome?n=f7d
```

VISIT NOW!

You can connect processes using NSOLID_SAAS as an environment variable with this value:

```
> [redacted]e4d3.prod.proxy.saas
```

Quick Install Wizard

FROM

NODESOURCE

URL for the user console

```
> https://905fe4b3-[redacted].saas.nodesource.io/welcome?n=f7d
```

VISIT NOW!

You can connect processes using NSOLID_SAAS as an environment variable with this value:

```
> [redacted]e4d3.prod.proxy.saas
```

git clone

nodesource/nsolid-workshop

⭐ Star it

CPU Example

Running CPU Example

```
git clone https://github.com/nodesource/nsolid-workshop.git
cd nsolid-workshop
npm install

cd 1-cpu/
```

Running CPU Example

```
NSOLID_SAAS=xxxxxxx.prod.proxy.saas.nodesource.io:9001 nsolid 1-slow-server.js
```

OR

```
  "nsolid": {
    "saas": "xxxxxxxxxxx"
  }
}
```

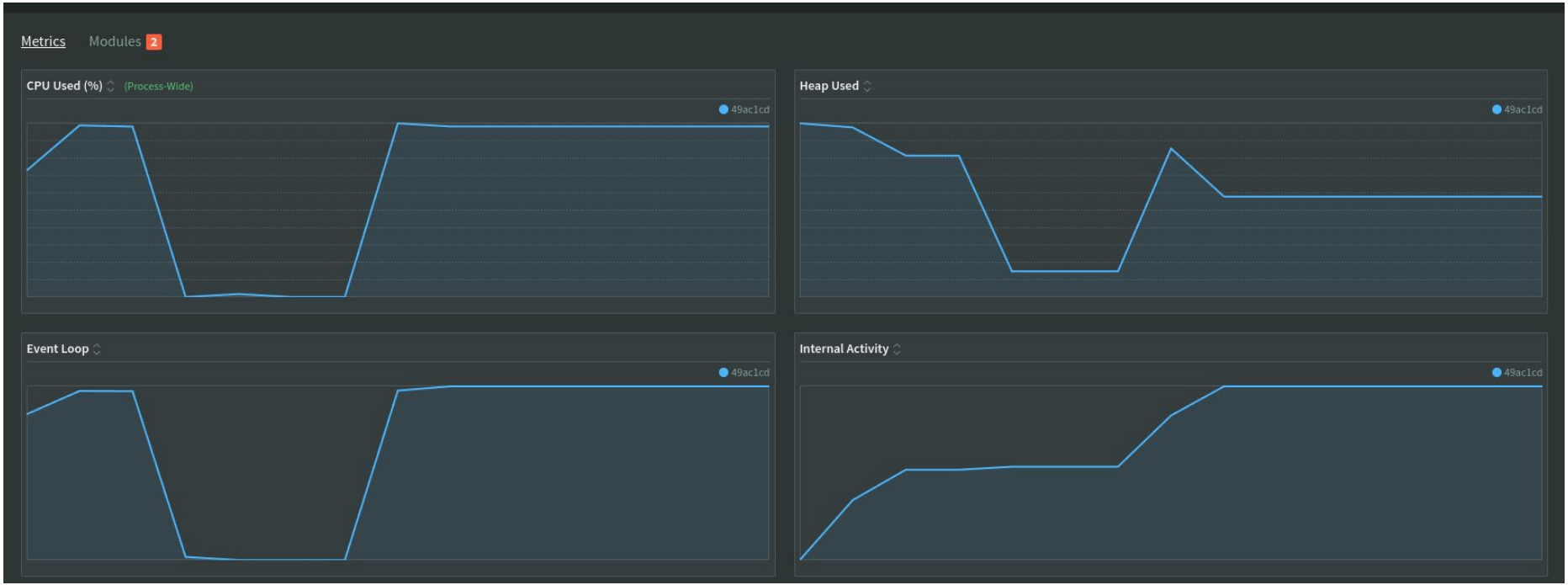
Running Benchmark

```
node benchmark.js
```

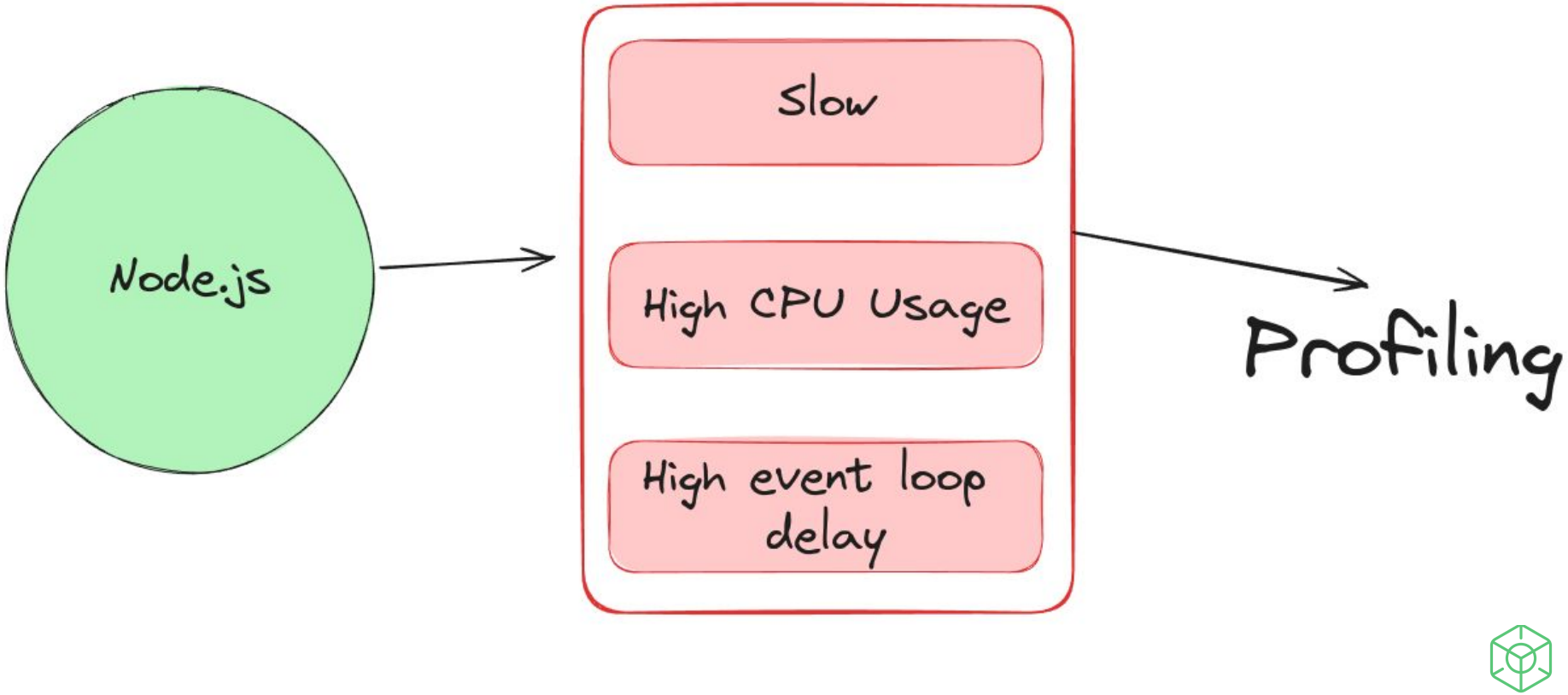
```
$ node benchmark.js

┌─────────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┐
│ Stat    │ 2.5% │ 50%  │ 97.5%│ 99%  │ Avg  │ Stdev│ Max  │
├─────────┼──────┼──────┼──────┼──────┼──────┼──────┼──────┤
│ Latency │ 14 ms│ 23 ms│ 26 ms│ 27 ms│23.6 ms│31.39 ms│1414 ms│
└─────────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┘

┌─────────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┐
│ Stat    │ 1%   │ 2.5% │ 50%  │ 97.5%│ Avg  │ Stdev│ Min  │
├─────────┼──────┼──────┼──────┼──────┼──────┼──────┼──────┤
│ Req/Sec │ 3,211│ 3,211│ 4,231│ 4,307│4,143.91│297.61│3,211 │
├─────────┼──────┼──────┼──────┼──────┼──────┼──────┼──────┤
│ Bytes/Sec│2.14 MB│2.14 MB│2.82 MB│2.87 MB│2.76 MB│199 kB│2.14 MB│
└─────────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┘

Req/Bytes counts sampled once per second.
# of samples: 11

46k requests in 11.03s, 30.4 MB read
```

What do you spot by looking at the graphs?

Analyze Console

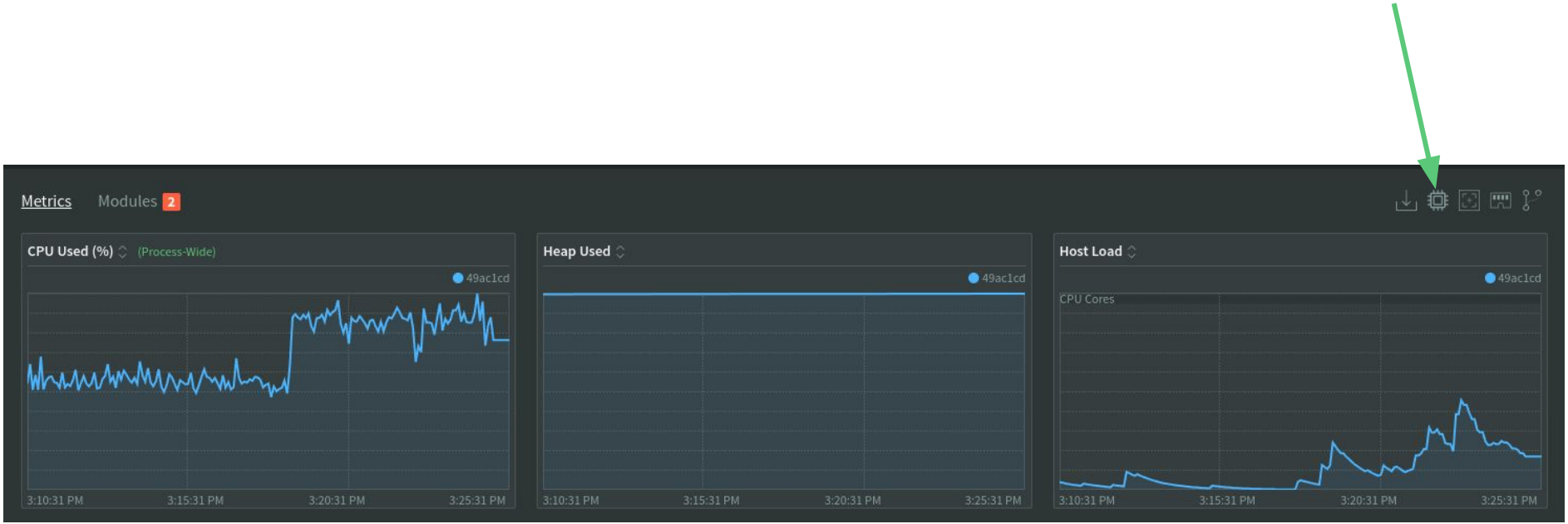


When is Profiling a good choice?

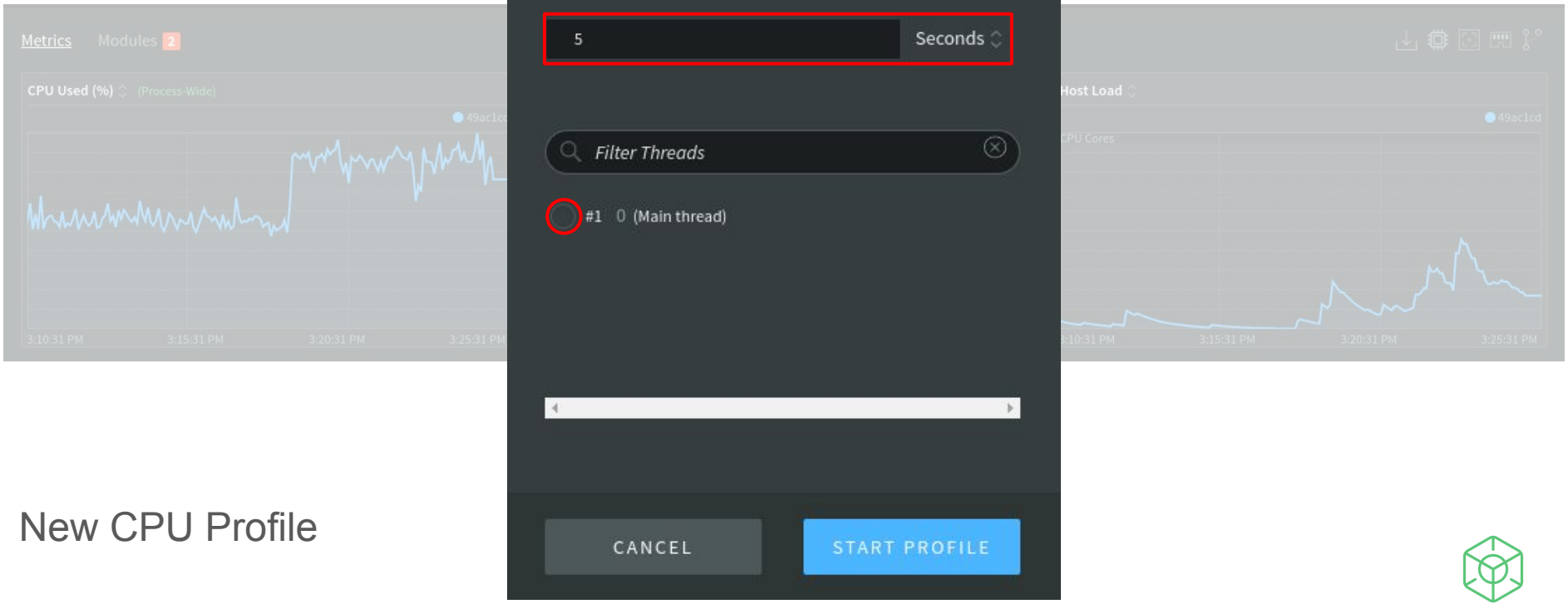


Change benchmark duration for a precise Profiling

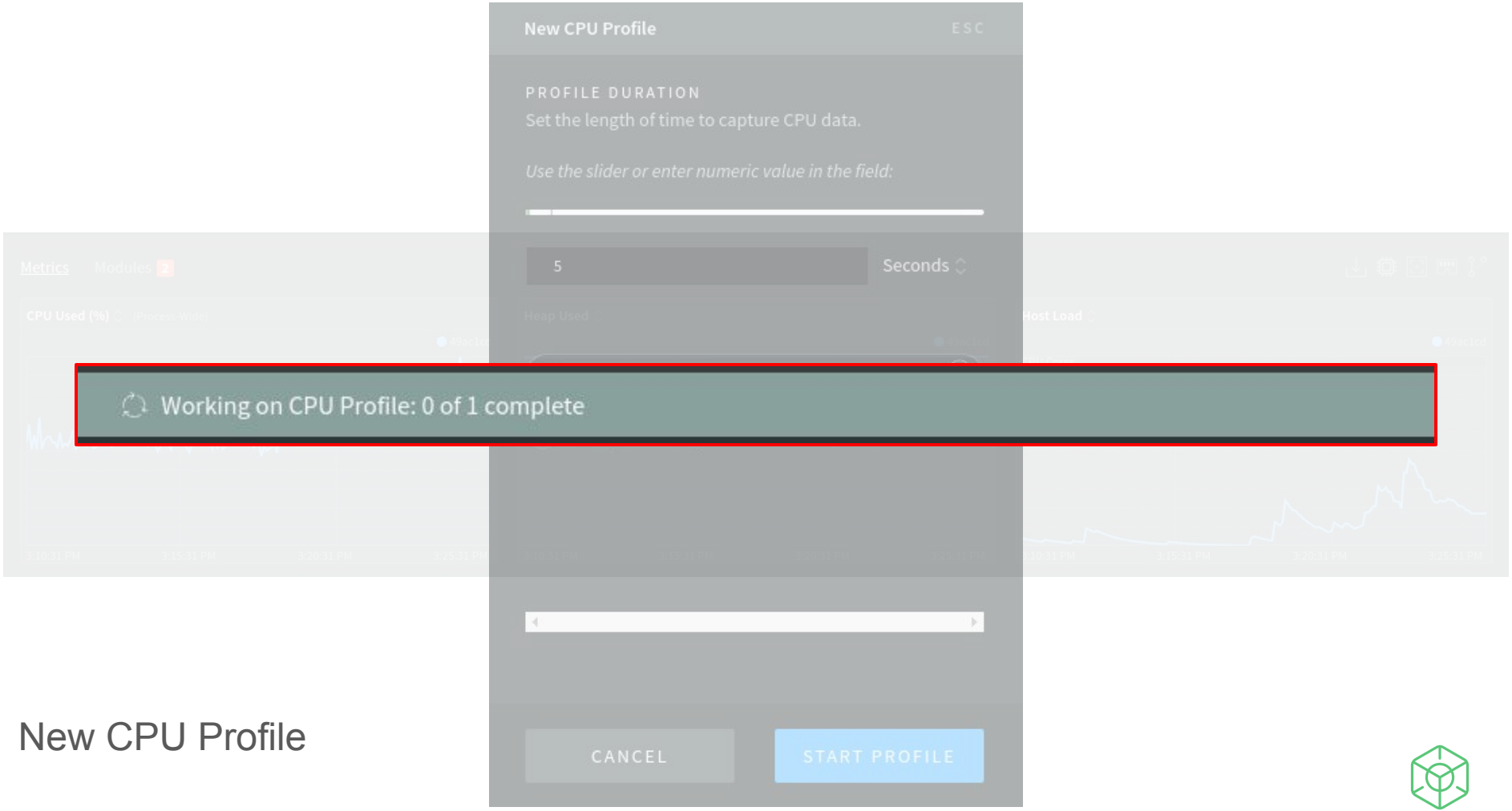
```
9 M 1-cpu/benchmark.js
8 @@ -6,7 +6,7 @@ async function main () {
7     const result = await autocannon({
6       url: 'http://localhost:3000',
5       connections: 100,
4 -     duration: 10,
3 +     duration: 60,
2     })
1     console.log(autocannon.printResult(result));
```



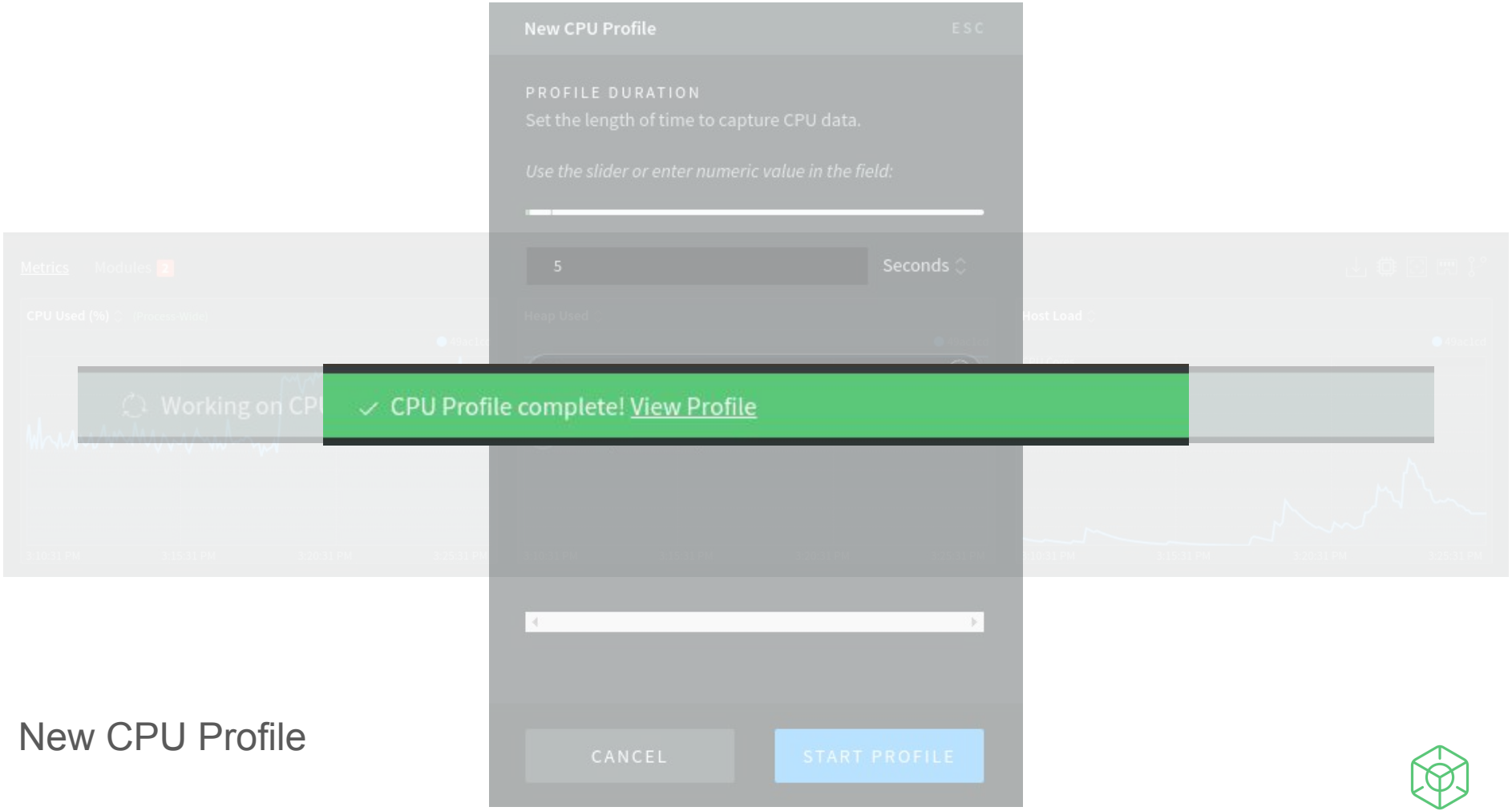
New CPU Profile



New CPU Profile

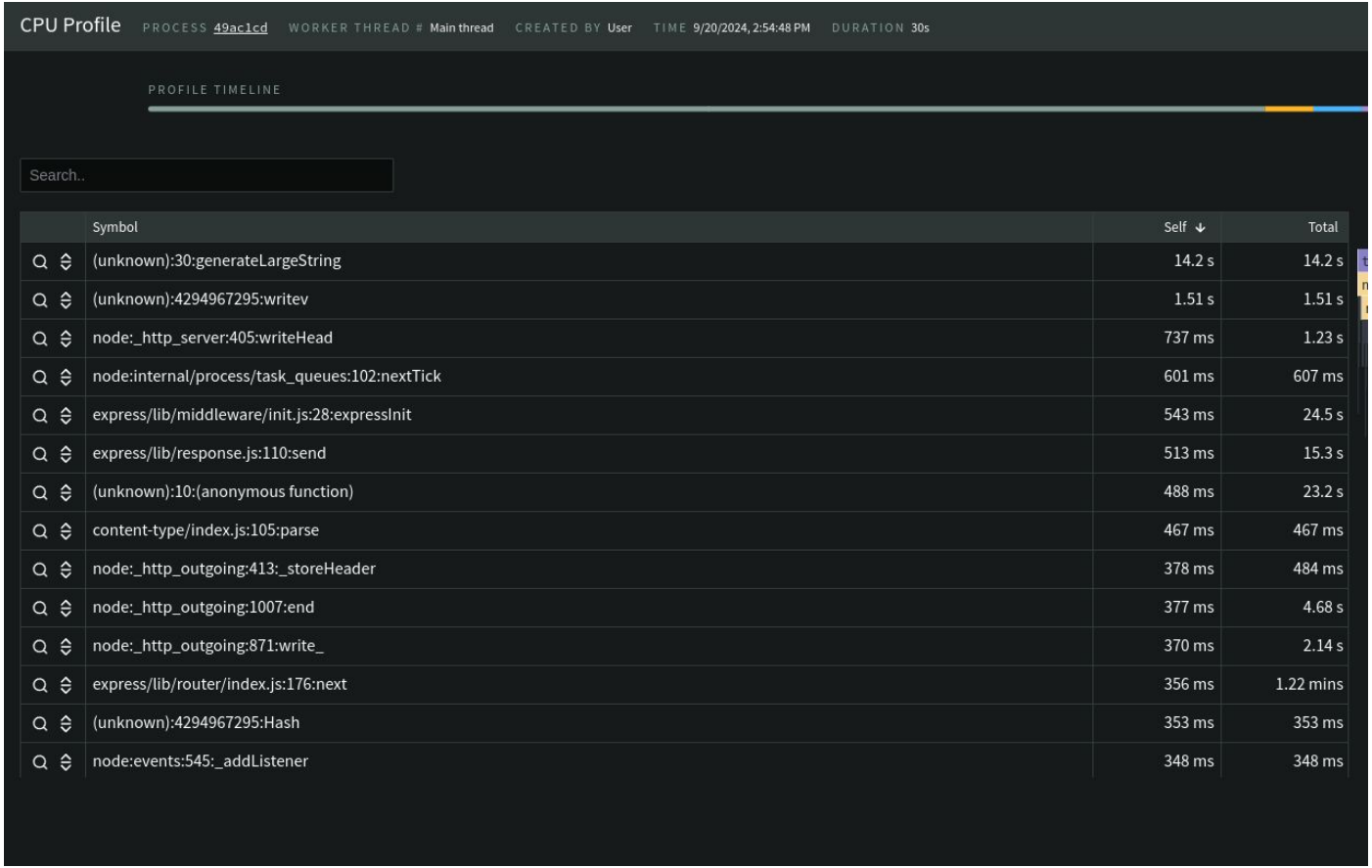


New CPU Profile



New CPU Profile

CPU Profiling Results

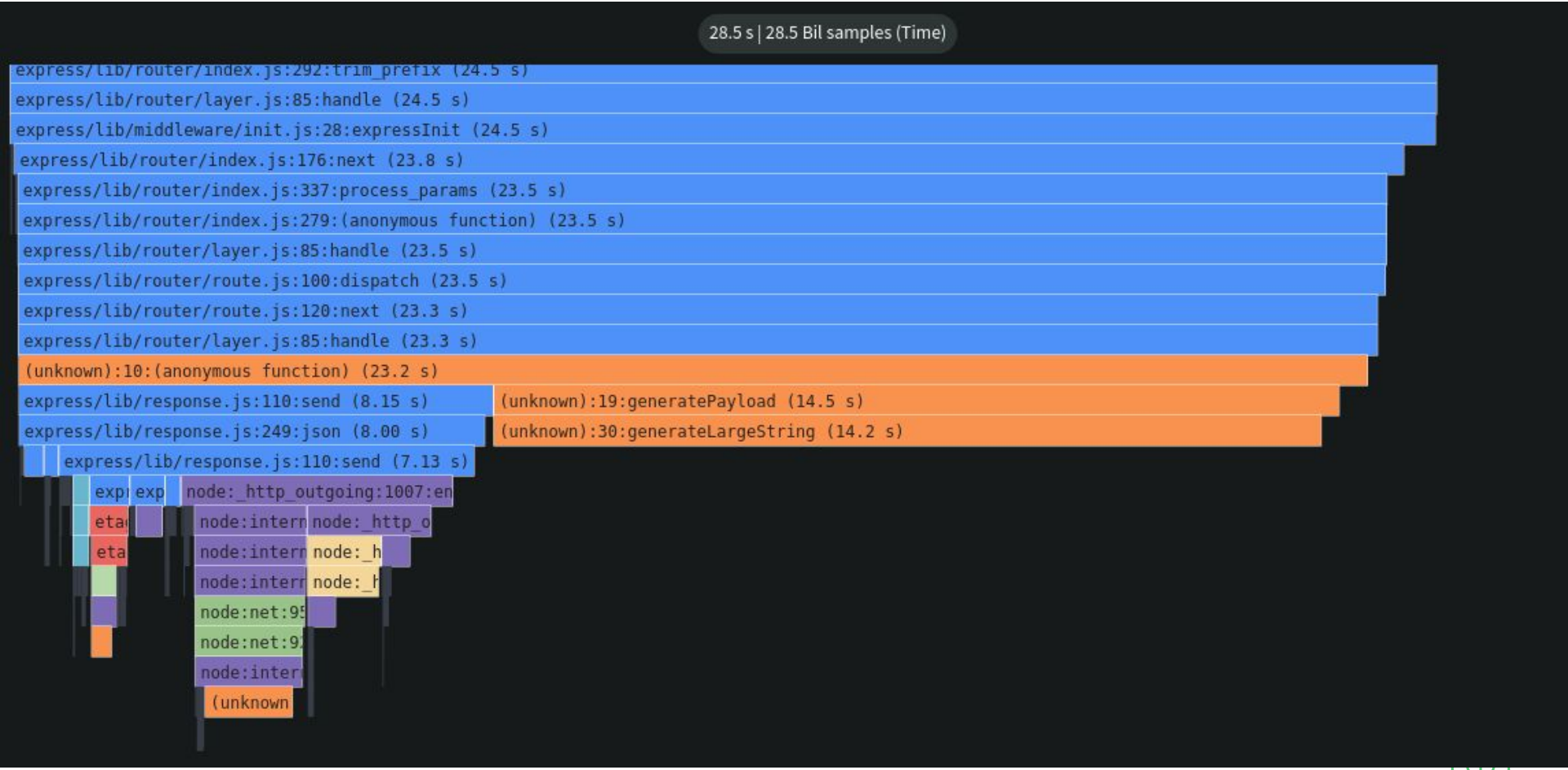


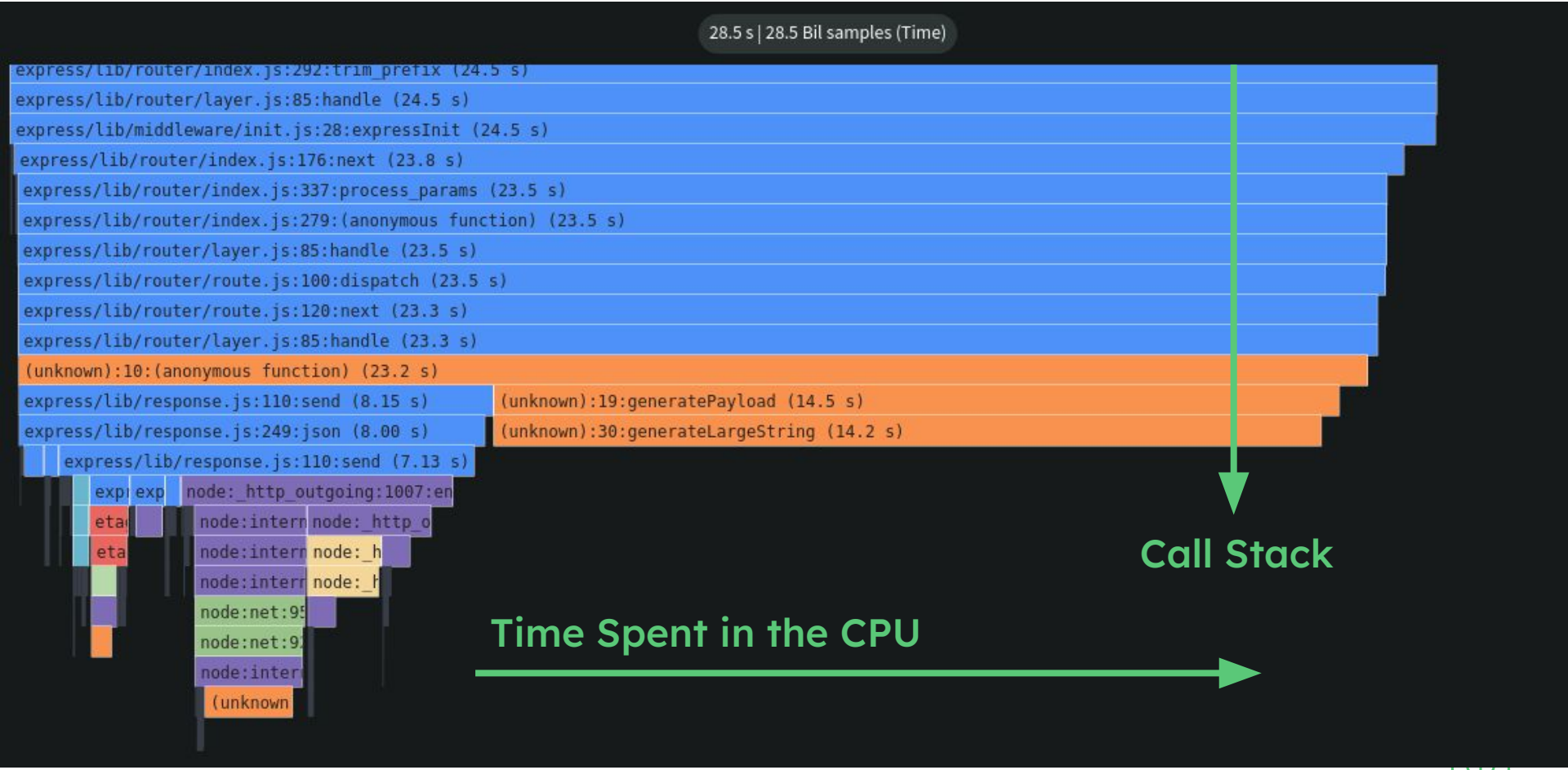
CPU Profile PROCESS 49ac1cd WORKER THREAD # Main thread CREATED BY User TIME 9/20/2024, 2:54:48 PM DURATION 30s

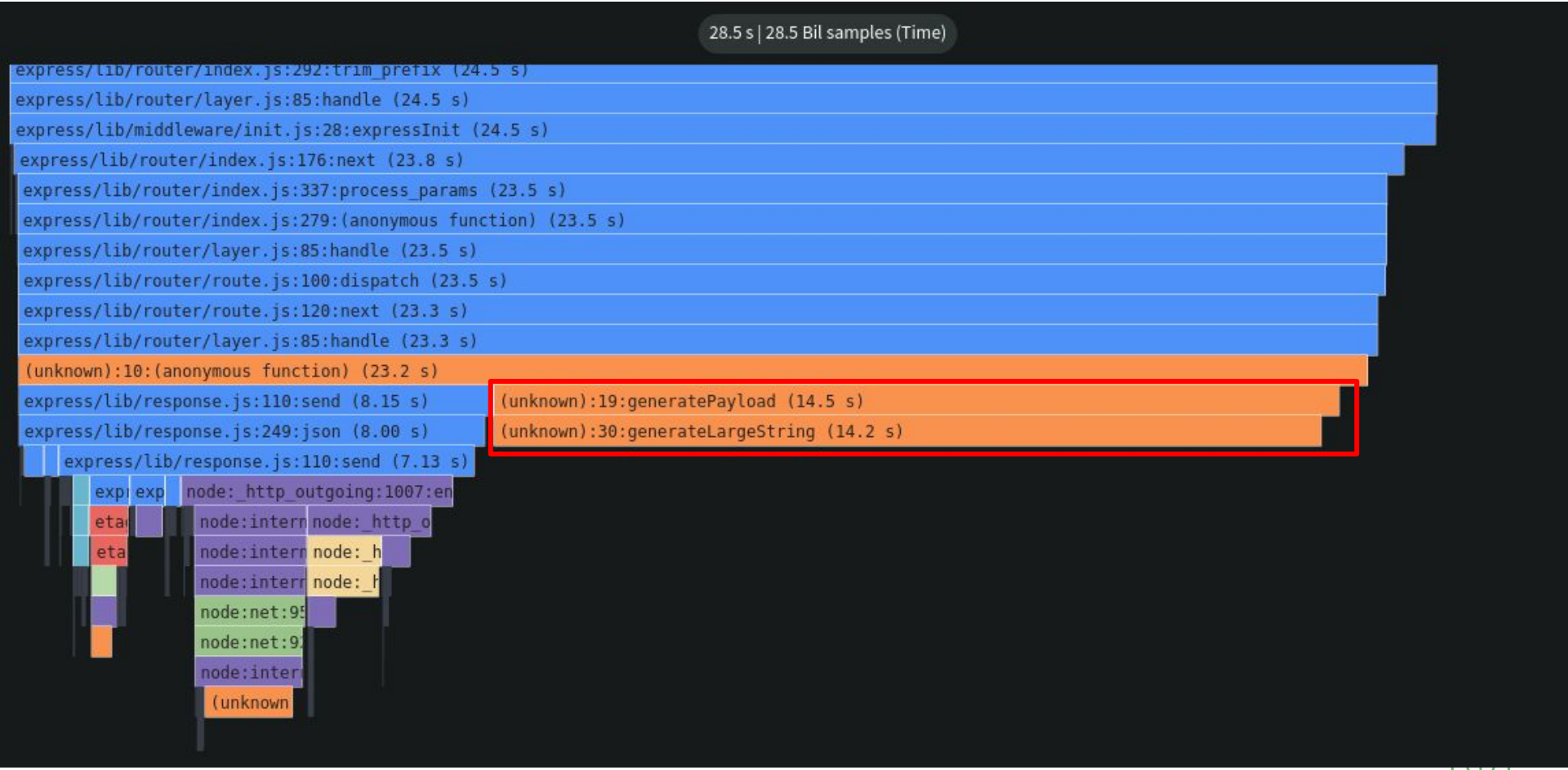
PROFILE TIMELINE

Search..

Symbol	Self ↓	Total
(unknown):30:generateLargeString	14.2 s	14.2 s
(unknown):4294967295:writev	1.51 s	1.51 s
node:_http_server:405:writeHead	737 ms	1.23 s
node:internal/process/task_queues:102:nextTick	601 ms	607 ms
express/lib/middleware/init.js:28:expressInit	543 ms	24.5 s
express/lib/response.js:110:send	513 ms	15.3 s
(unknown):10:(anonymous function)	488 ms	23.2 s
content-type/index.js:105:parse	467 ms	467 ms
node:_http_outgoing:413:_storeHeader	378 ms	484 ms
node:_http_outgoing:1007:end	377 ms	4.68 s
node:_http_outgoing:871:write_	370 ms	2.14 s
express/lib/router/index.js:176:next	356 ms	1.22 mins
(unknown):4294967295:Hash	353 ms	353 ms
node:events:545:_addListener	348 ms	348 ms


28.5 s | 28.5 Bil samples (Time)
express/lib/router/index.js:292:trim_prefix (24.5 s)
express/lib/router/layer.js:85:handle (24.5 s)
express/lib/middleware/init.js:28:expressInit (24.5 s)
express/lib/router/index.js:176:next (23.8 s)
express/lib/router/index.js:337:process_params (23.5 s)
express/lib/router/index.js:279:(anonymous function) (23.5 s)
express/lib/router/layer.js:85:handle (23.5 s)
express/lib/router/route.js:100:dispatch (23.5 s)
express/lib/router/route.js:120:next (23.3 s)
express/lib/router/layer.js:85:handle (23.3 s)
(unknown):10:(anonymous function) (23.2 s)
express/lib/response.js:110:send (8.15 s)
(unknown):19:generatePayload (14.5 s)
express/lib/response.js:249:json (8.00 s)
(unknown):30:generateLargeString (14.2 s)
express/lib/response.js:110:send (7.13 s)
node:_http_outgoing:1007:en
node:intern
node:_http_o
node:intern
node:_h
node:intern
node:net:9
node:net:9
node:inter
(unknown


28.5 s | 28.5 Bil samples (Time)
express/lib/router/index.js:292:trim_prefix (24.5 s)
express/lib/router/layer.js:85:handle (24.5 s)
express/lib/middleware/init.js:28:expressInit (24.5 s)
express/lib/router/index.js:176:next (23.8 s)
express/lib/router/index.js:337:process_params (23.5 s)
express/lib/router/index.js:279:(anonymous function) (23.5 s)
express/lib/router/layer.js:85:handle (23.5 s)
express/lib/router/route.js:100:dispatch (23.5 s)
express/lib/router/route.js:120:next (23.3 s)
express/lib/router/layer.js:85:handle (23.3 s)
(unknown):10:(anonymous function) (23.2 s)
express/lib/response.js:110:send (8.15 s)
(unknown):19:generatePayload (14.5 s)
express/lib/response.js:249:json (8.00 s)
(unknown):30:generateLargeString (14.2 s)
express/lib/response.js:110:send (7.13 s)
node:_http_outgoing:1007:en
node:intern
node:_http_o
node:intern
node:_h
node:intern
node:net:9
node:net:9
node:inter
(unknown
Call Stack
Time Spent in the CPU


28.5 s | 28.5 Bil samples (Time)
express/lib/router/index.js:292:trim_prefix (24.5 s)
express/lib/router/layer.js:85:handle (24.5 s)
express/lib/middleware/init.js:28:expressInit (24.5 s)
express/lib/router/index.js:176:next (23.8 s)
express/lib/router/index.js:337:process_params (23.5 s)
express/lib/router/index.js:279:(anonymous function) (23.5 s)
express/lib/router/layer.js:85:handle (23.5 s)
express/lib/router/route.js:100:dispatch (23.5 s)
express/lib/router/route.js:120:next (23.3 s)
express/lib/router/layer.js:85:handle (23.3 s)
(unknown):10:(anonymous function) (23.2 s)
express/lib/response.js:110:send (8.15 s)
(unknown):19:generatePayload (14.5 s)
express/lib/response.js:249:json (8.00 s)
(unknown):30:generateLargeString (14.2 s)
express/lib/response.js:110:send (7.13 s)
node:_http_outgoing:1007:en
node:intern
node:_http_o
node:intern
node:_h
node:intern
node:net:9
node:net:9
node:inter
(unknown

Memory Example

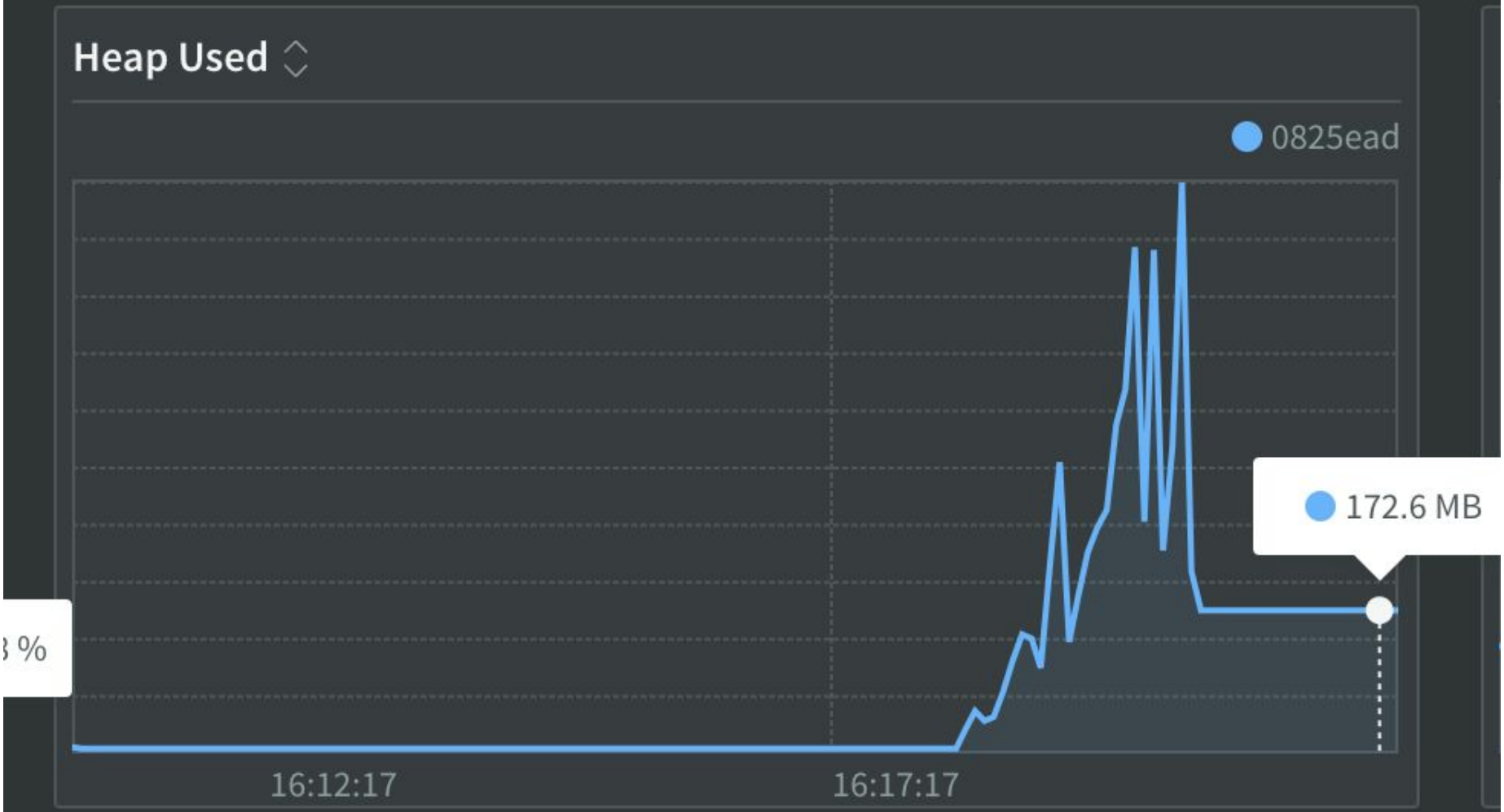
Running Memory Example

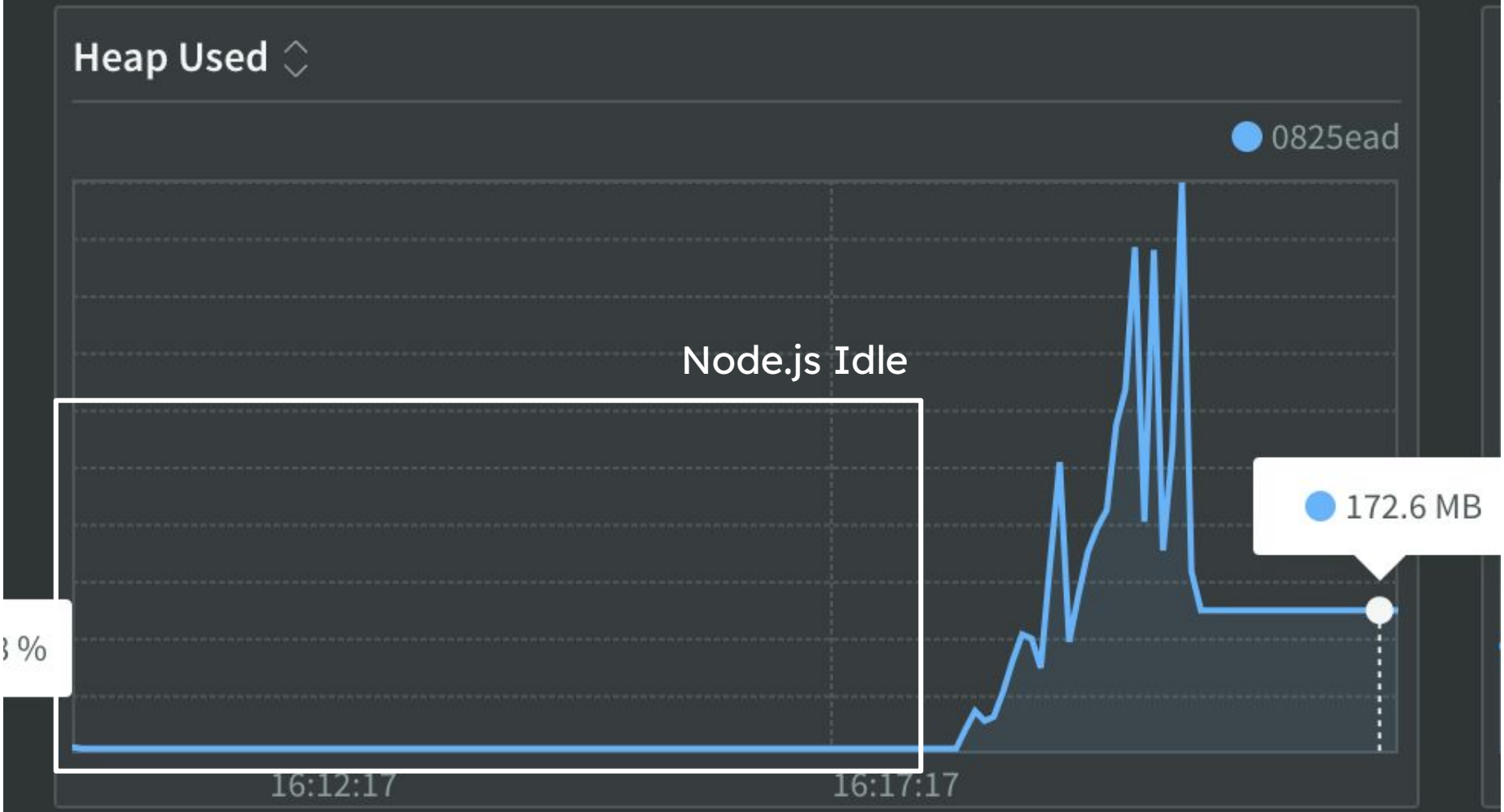
```
$ cd 2-memory/

$ nsolid index.js
Server is running on port 3000
In a separate terminal run: node benchmark.js
```

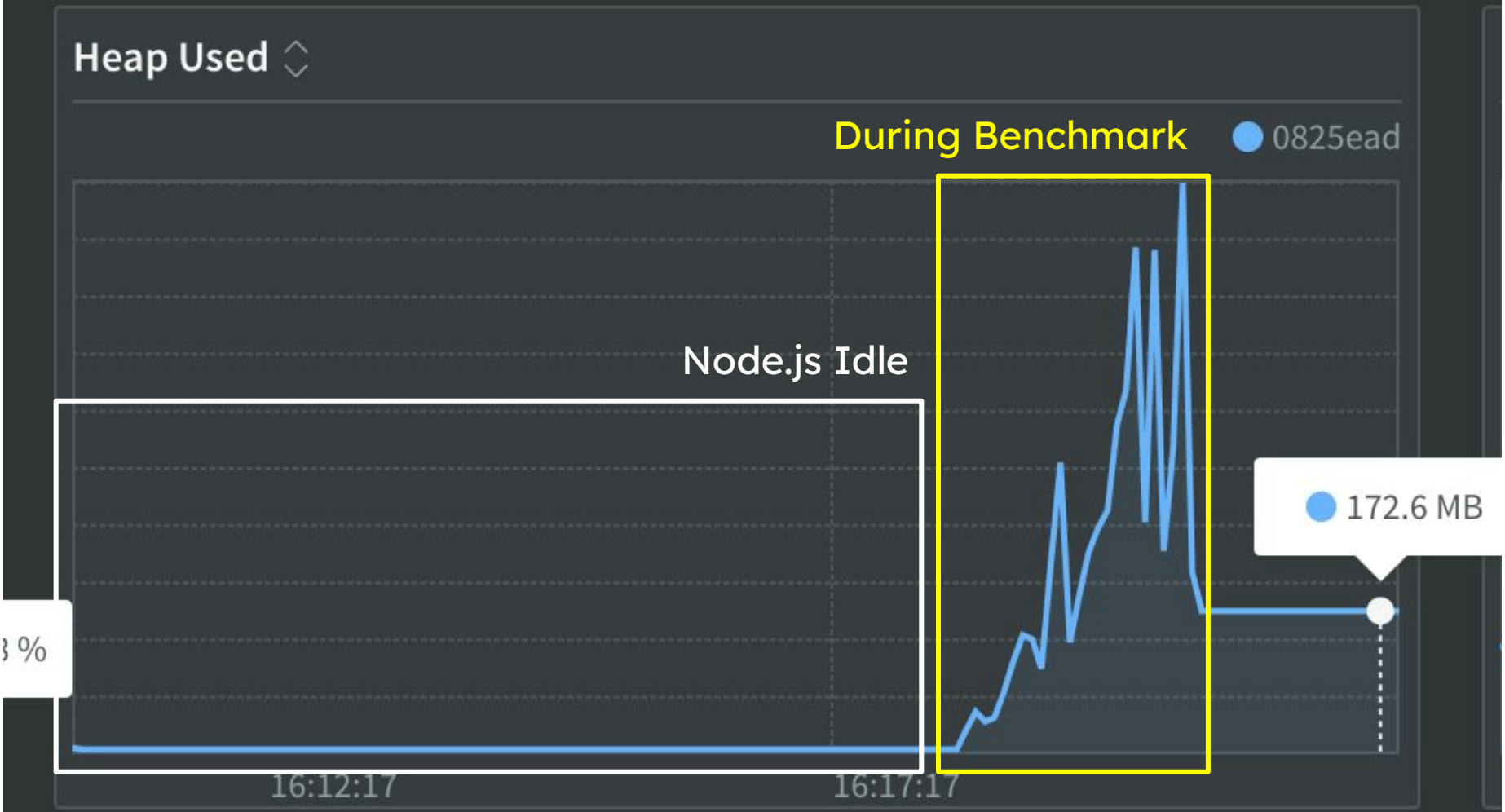
Running Memory Example

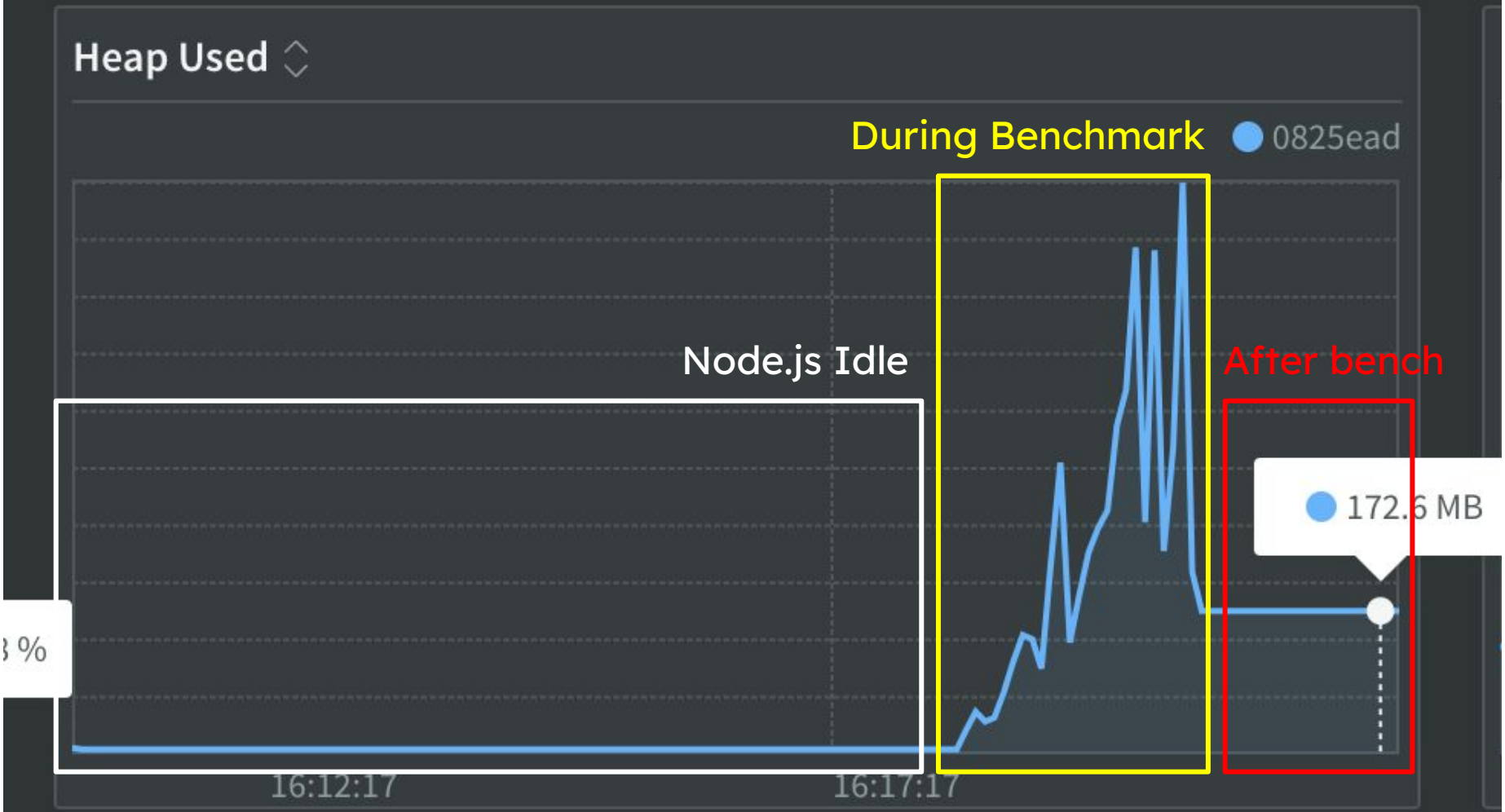
Separate Terminal

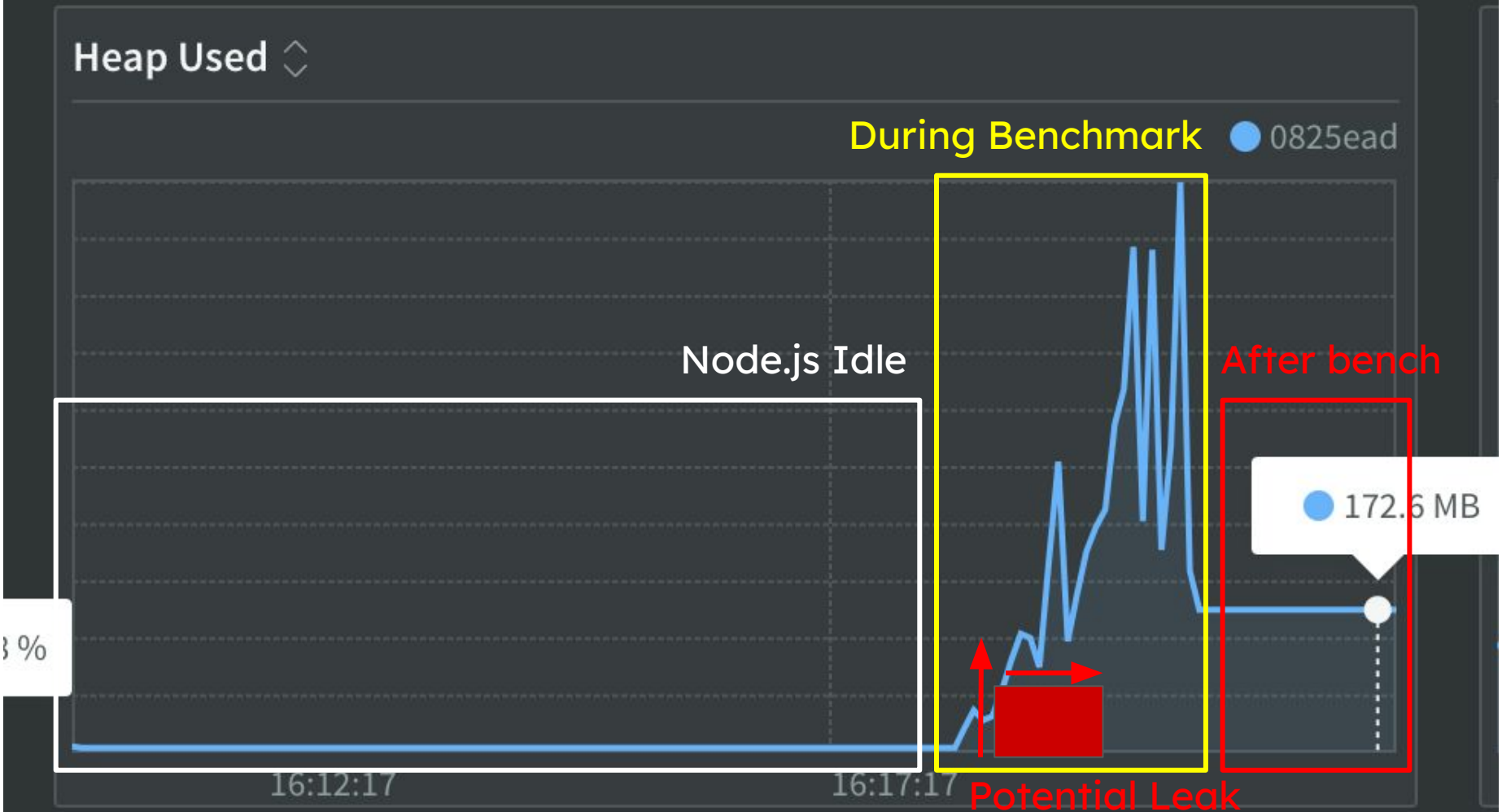
What do you spot by looking at the graphs?

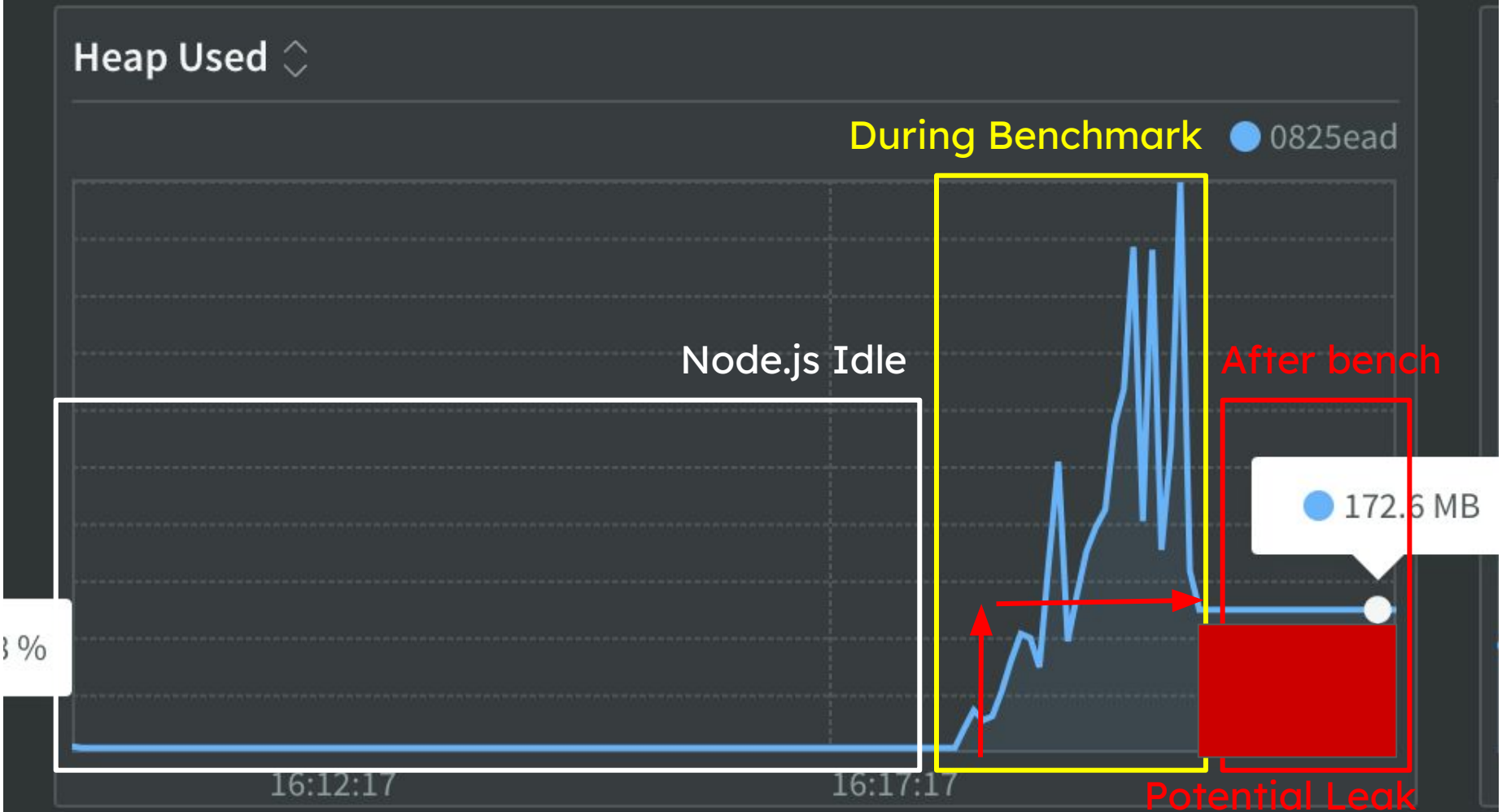

Heap Used
0825ead
172.6 MB
16:12:17
16:17:17


Heap Used
0825ead
Node.js Idle
172.6 MB
16:12:17
16:17:17

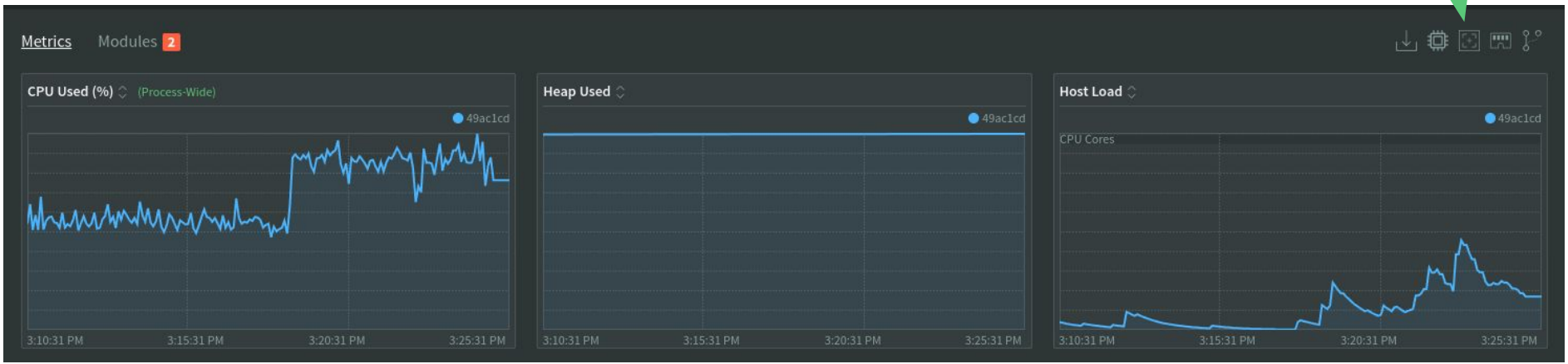

Heap Used
During Benchmark
0825ead
Node.js Idle
172.6 MB
16:12:17
16:17:17


Heap Used
During Benchmark
0825ead
Node.js Idle
After bench
172.6 MB
16:12:17
16:17:17

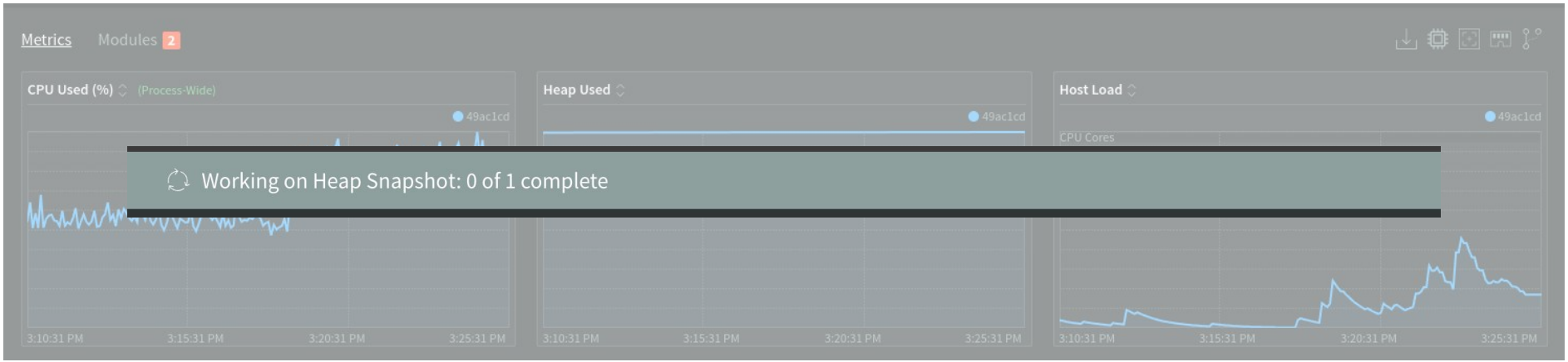

Heap Used
During Benchmark
0825ead
Node.js Idle
After bench
172.6 MB
16:12:17
16:17:17
Potential Leak


Heap Used
During Benchmark
0825ead
Node.js Idle
After bench
172.6 MB
16:12:17
16:17:17
Potential Leak

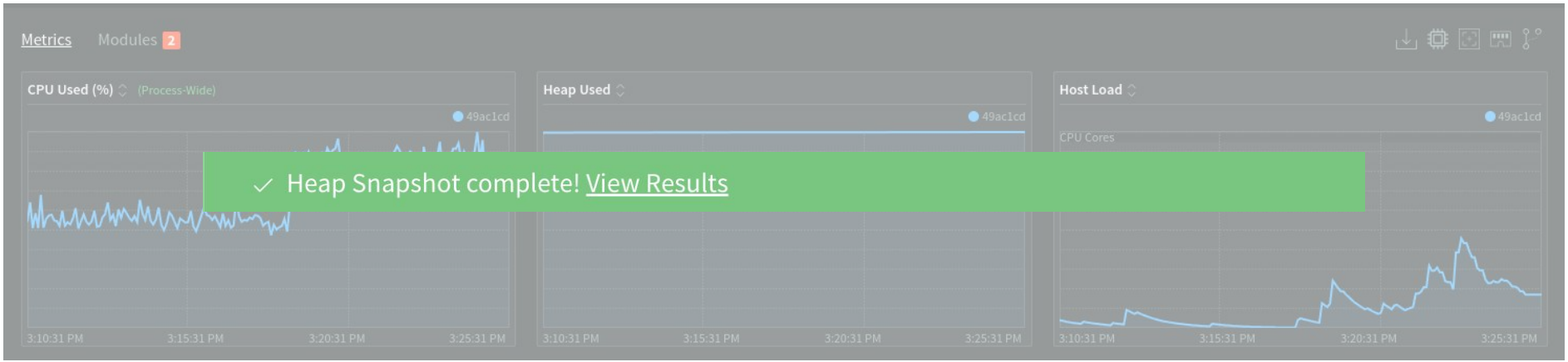
Potential Leak identified what about now?



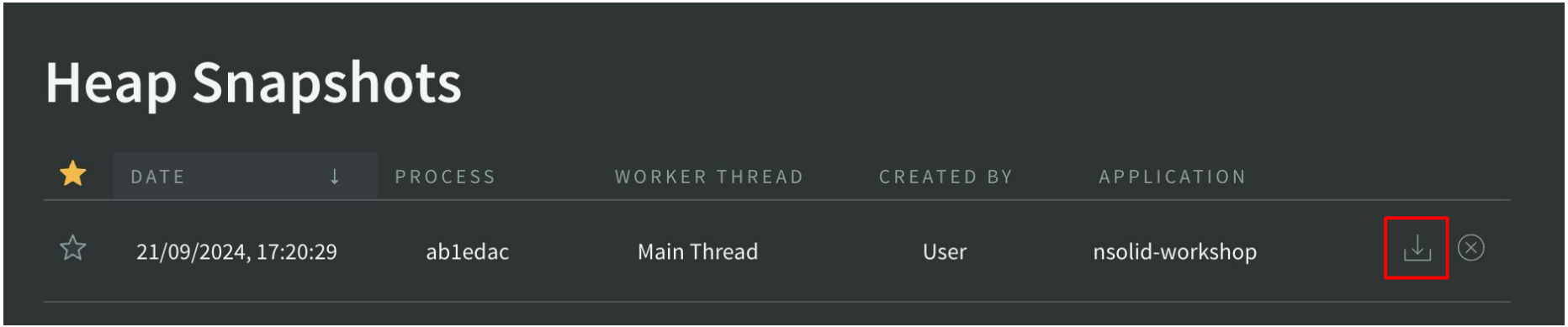
New Heap Snapshot



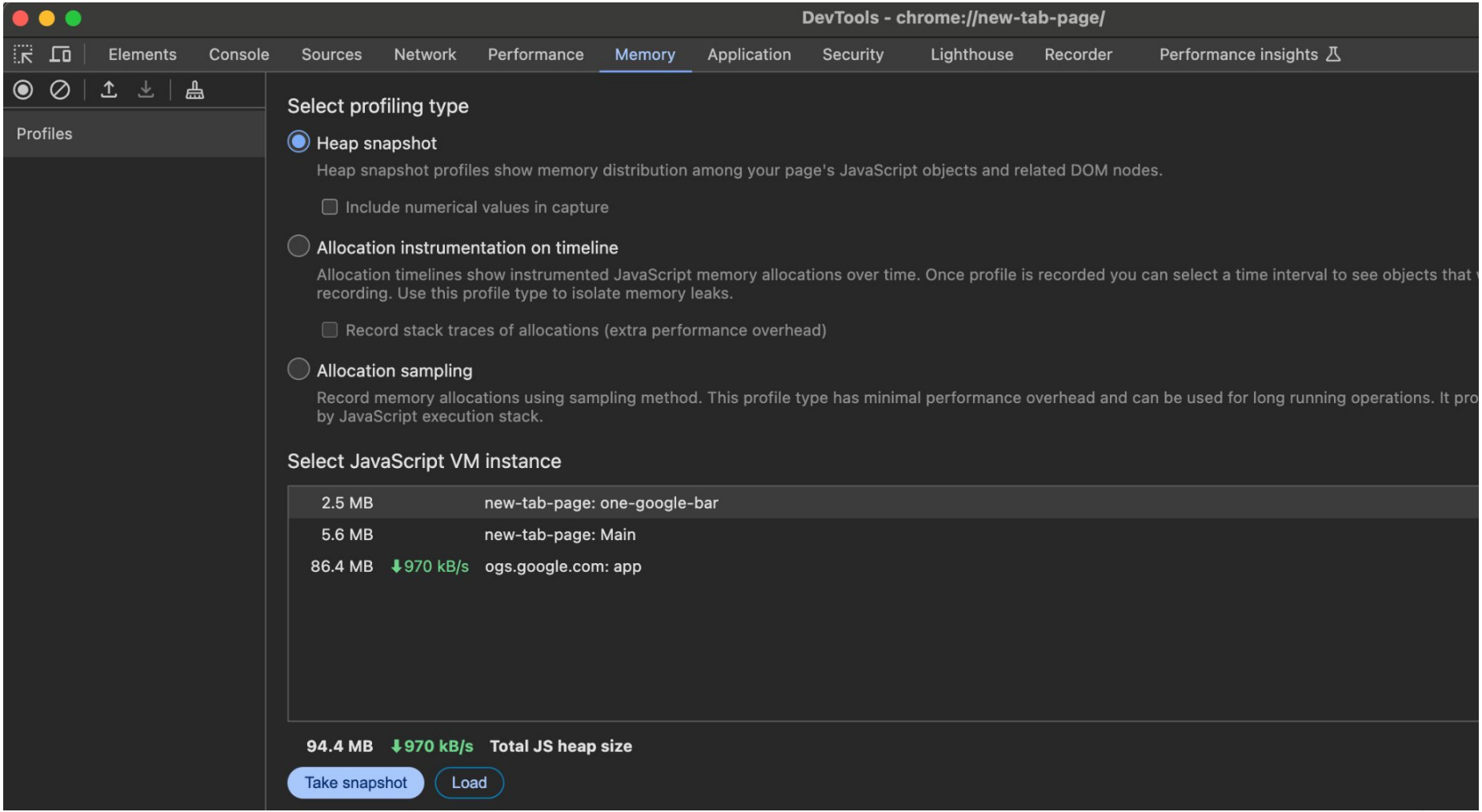
New Heap Snapshot

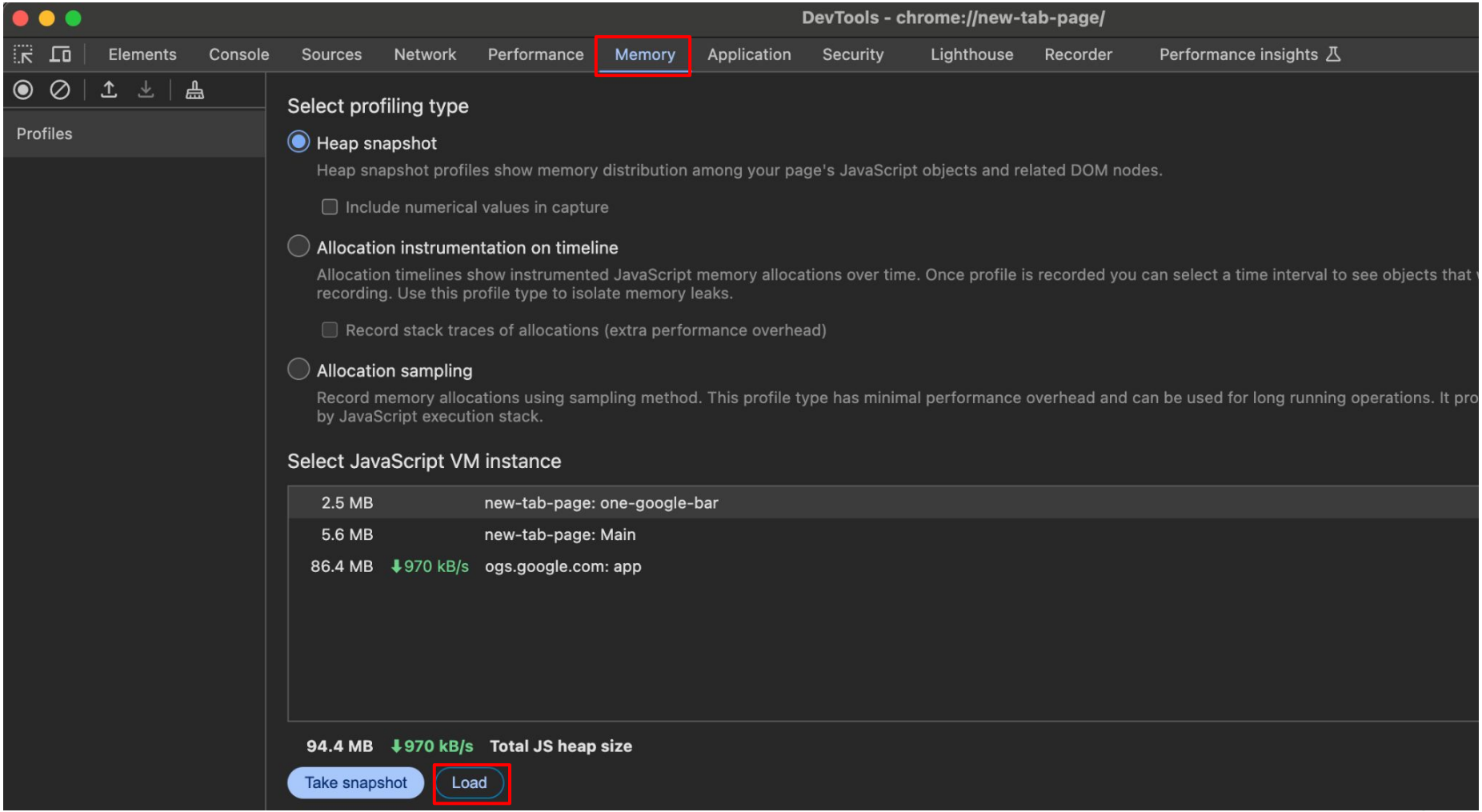


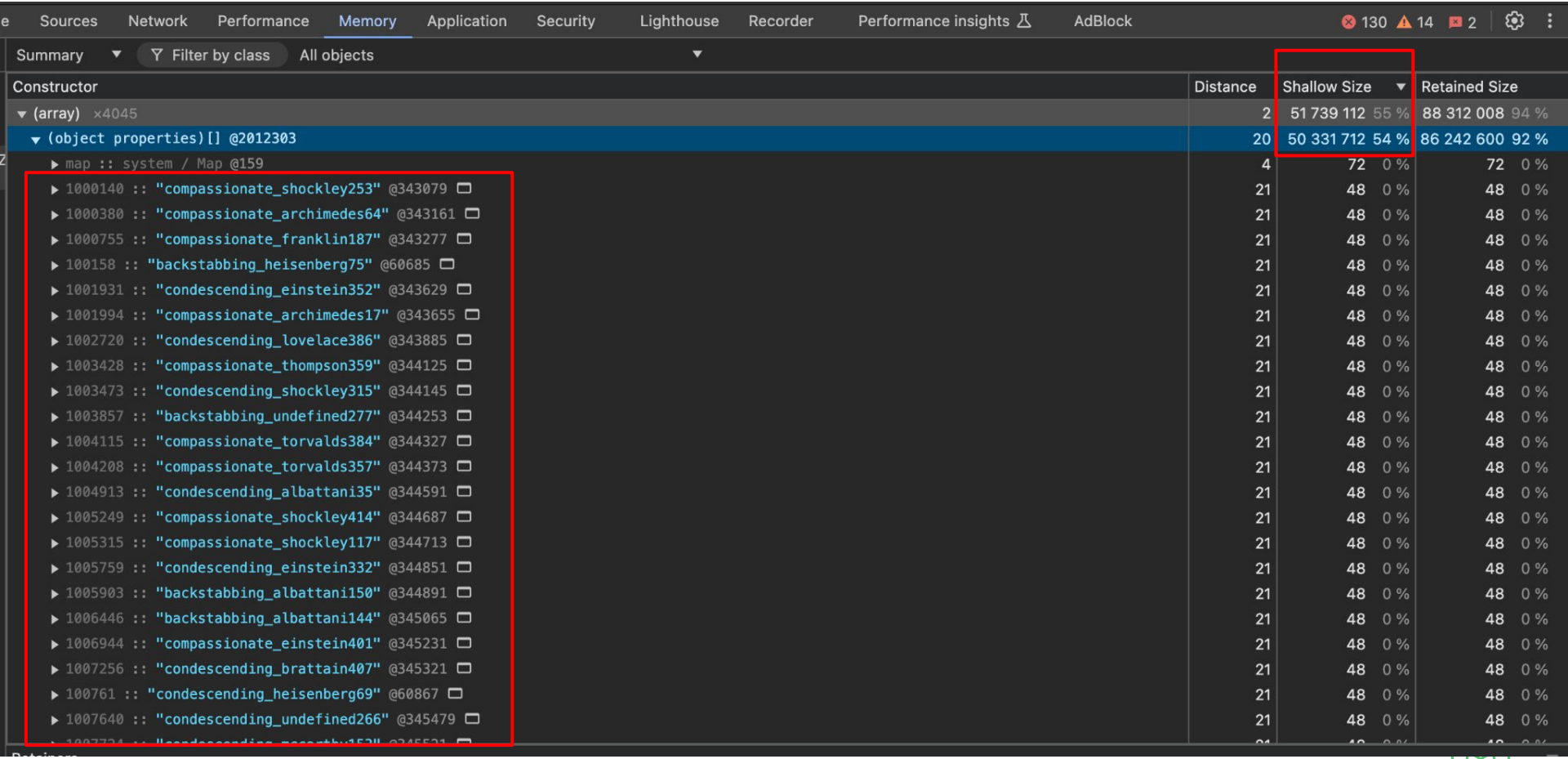
New Heap Snapshot



Reading Snapshots


DevTools - chrome://new-tab-page/
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
Profiles
Select profiling type
Heap snapshot
Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.
Include numerical values in capture
Allocation instrumentation on timeline
Allocation timelines show instrumented JavaScript memory allocations over time. Once profile is recorded you can select a time interval to see objects that
recording. Use this profile type to isolate memory leaks.
Record stack traces of allocations (extra performance overhead)
Allocation sampling
Record memory allocations using sampling method. This profile type has minimal performance overhead and can be used for long running operations. It pro
by JavaScript execution stack.
Select JavaScript VM instance
2.5 MB new-tab-page: one-google-bar
5.6 MB new-tab-page: Main
86.4 MB ⬇970 kB/s ogs.google.com: app
94.4 MB ⬇970 kB/s Total JS heap size
Take snapshot
Load


DevTools - chrome://new-tab-page/
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
Profiles
Select profiling type
Heap snapshot
Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.
Include numerical values in capture
Allocation instrumentation on timeline
Allocation timelines show instrumented JavaScript memory allocations over time. Once profile is recorded you can select a time interval to see objects that
recording. Use this profile type to isolate memory leaks.
Record stack traces of allocations (extra performance overhead)
Allocation sampling
Record memory allocations using sampling method. This profile type has minimal performance overhead and can be used for long running operations. It pro
by JavaScript execution stack.
Select JavaScript VM instance
2.5 MB new-tab-page: one-google-bar
5.6 MB new-tab-page: Main
86.4 MB ⬇970 kB/s ogs.google.com: app
94.4 MB ⬇970 kB/s Total JS heap size
Take snapshot
Load


Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
AdBlock
130
14
2
Summary
Filter by class
All objects
Constructor
Distance
Shallow Size
Retained Size
(array) ×4045 2 51 739 112 55 % 88 312 008 94 %
(object properties)[] @2012303 20 50 331 712 54 % 86 242 600 92 %
map :: system / Map @159 4 72 0 % 72 0 %
1000140 :: "compassionate_shockley253" @343079 21 48 0 % 48 0 %
1000380 :: "compassionate_archimedes64" @343161 21 48 0 % 48 0 %
1000755 :: "compassionate_franklin187" @343277 21 48 0 % 48 0 %
100158 :: "backstabbing_heisenberg75" @60685 21 48 0 % 48 0 %
1001931 :: "condescending_einstein352" @343629 21 48 0 % 48 0 %
1001994 :: "compassionate_archimedes17" @343655 21 48 0 % 48 0 %
1002720 :: "condescending_lovelace386" @343885 21 48 0 % 48 0 %
1003428 :: "compassionate_thompson359" @344125 21 48 0 % 48 0 %
1003473 :: "condescending_shockley315" @344145 21 48 0 % 48 0 %
1003857 :: "backstabbing_undefined277" @344253 21 48 0 % 48 0 %
1004115 :: "compassionate_torvalds384" @344327 21 48 0 % 48 0 %
1004208 :: "compassionate_torvalds357" @344373 21 48 0 % 48 0 %
1004913 :: "condescending_albattani35" @344591 21 48 0 % 48 0 %
1005249 :: "compassionate_shockley414" @344687 21 48 0 % 48 0 %
1005315 :: "compassionate_shockley117" @344713 21 48 0 % 48 0 %
1005759 :: "condescending_einstein332" @344851 21 48 0 % 48 0 %
1005903 :: "backstabbing_albattani150" @344891 21 48 0 % 48 0 %
1006446 :: "backstabbing_albattani144" @345065 21 48 0 % 48 0 %
1006944 :: "compassionate_einstein401" @345231 21 48 0 % 48 0 %
1007256 :: "condescending_brattain407" @345321 21 48 0 % 48 0 %
100761 :: "condescending_heisenberg69" @60867 21 48 0 % 48 0 %
1007640 :: "condescending_undefined266" @345479 21 48 0 % 48 0 %

Constructor	Distance	Shallow Size	Retained Size
▼ (array) ×4045	2	51 739 112 55 %	88 312 008 94 %
▼ (object properties)[] @2012303	20	50 331 712 54 %	86 242 600 92 %
▶ map :: system / Map @159	4	72 0 %	72 0 %
▶ 1000140 :: "compassionate_shockley253" @343079	21	48 0 %	48 0 %
▶ 1000380 :: "compassionate_archimedes64" @343161	21	48 0 %	48 0 %

Retainers

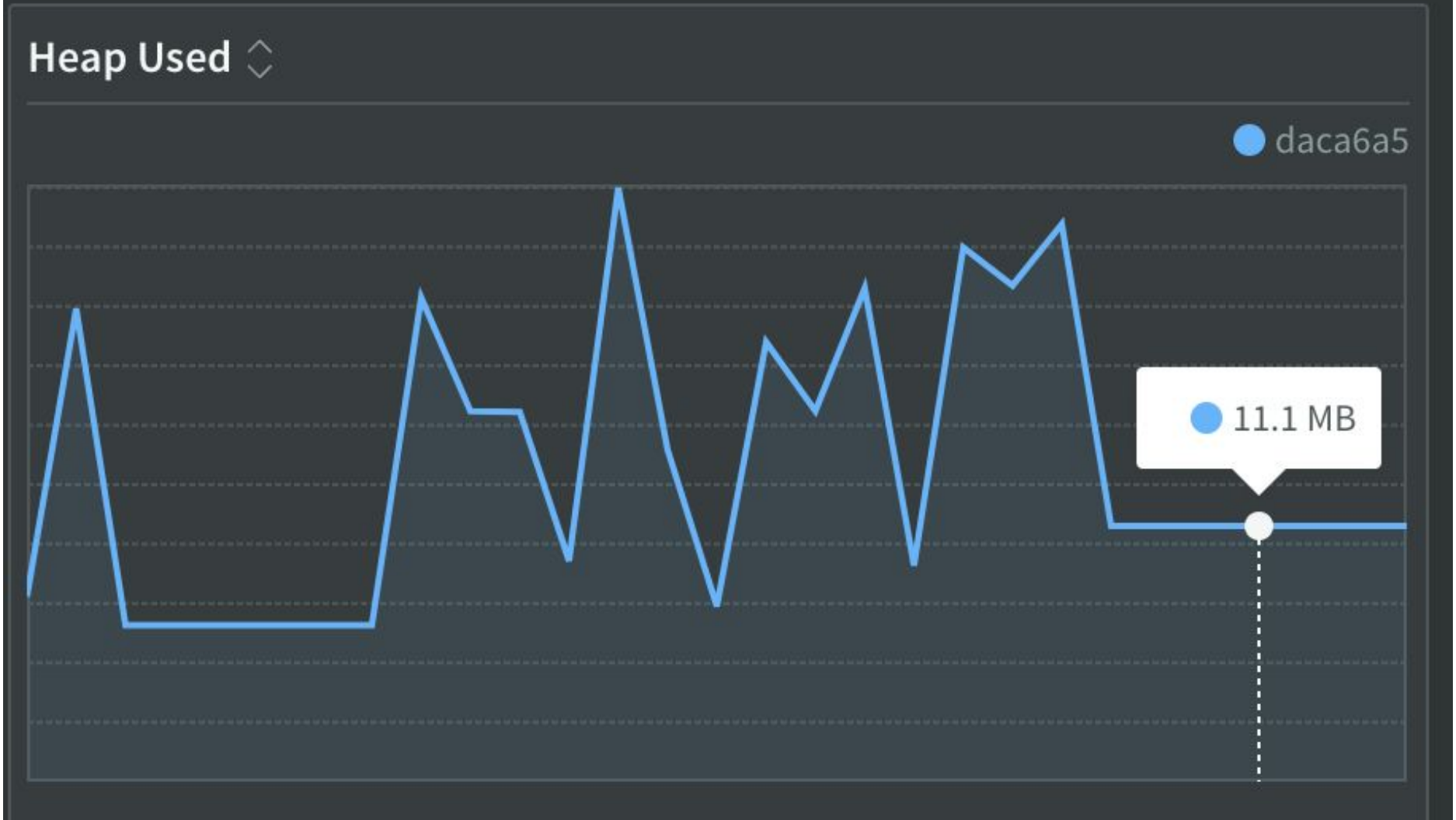
Object	Distance	Shallow Size	Retained Size
▼ 1000140 in (object properties)[] @2012303	20	50 331 712 54 %	86 242 600 92 %
▼ properties in Object @6621	19	24 0 %	86 242 624 92 %
▼ 350 / deoptimization literal in (Stack roots) @17	–	0 0 %	944 0 %
[7] in (GC roots) @3	–	0 0 %	479 192 1 %
▶ names in system / Context @6565	18	64 0 %	86 246 872 92 %
▶ 158030 in (internalized strings) @5	–	0 0 %	0 0 %

```
 4 function name () {
 5   let result = namesGenerator();
 6   if (names[result]) {
 7     result += names[result]++;
 8   }
 9   names[result] = 1;
10   return result;
11 }
```

```
 4 function name () {
 5   let result = namesGenerator();
 6   if (names[result]) {
 7     result += names[result]++;
 8   }
 9   names[result] = 1;
10   return result;
11 }
```

Infinity array

```
diff --git a/2-memory/index.js b/2-memory/index.js
index 82e3bf3..6eac308 100644
--- a/2-memory/index.js
+++ b/2-memory/index.js
@@ -25,10 +25,11 @@ const names = {};
 function name () {
   let result = namesGenerator();
   if (names[result]) {
-    result += names[result]++;
+    names[result]++;
+  } else {
+    names[result] = 1;
   }
-  names[result] = 1;
-  return result;
+  return result + names[result];
 }
```


Heap Used
daca6a5
11.1 MB

N|Solid can also generate **heap samples** and **track allocations**

New Heap Profile ESC

PROFILE DURATION

Set the length of time to capture heap data.

Use the slider or enter numeric value in the field:

5 Seconds

TYPE OF PROFILE

If you don't select a thread from the list, the main thread will be selected automatically.

Filter Threads

#1 0 (Main thread)

Heap Sample

Best for production use. Tracks heap objects allocated during function calls, but other memory grahp information.

Track Allocations

Best for deep memory leaks analysis. This option performs a Heap Snapshot and combines the functionality of Heap Snapshot and the Sampling profiler to give you the most detailed memory allocation, but with larger impact to the application.

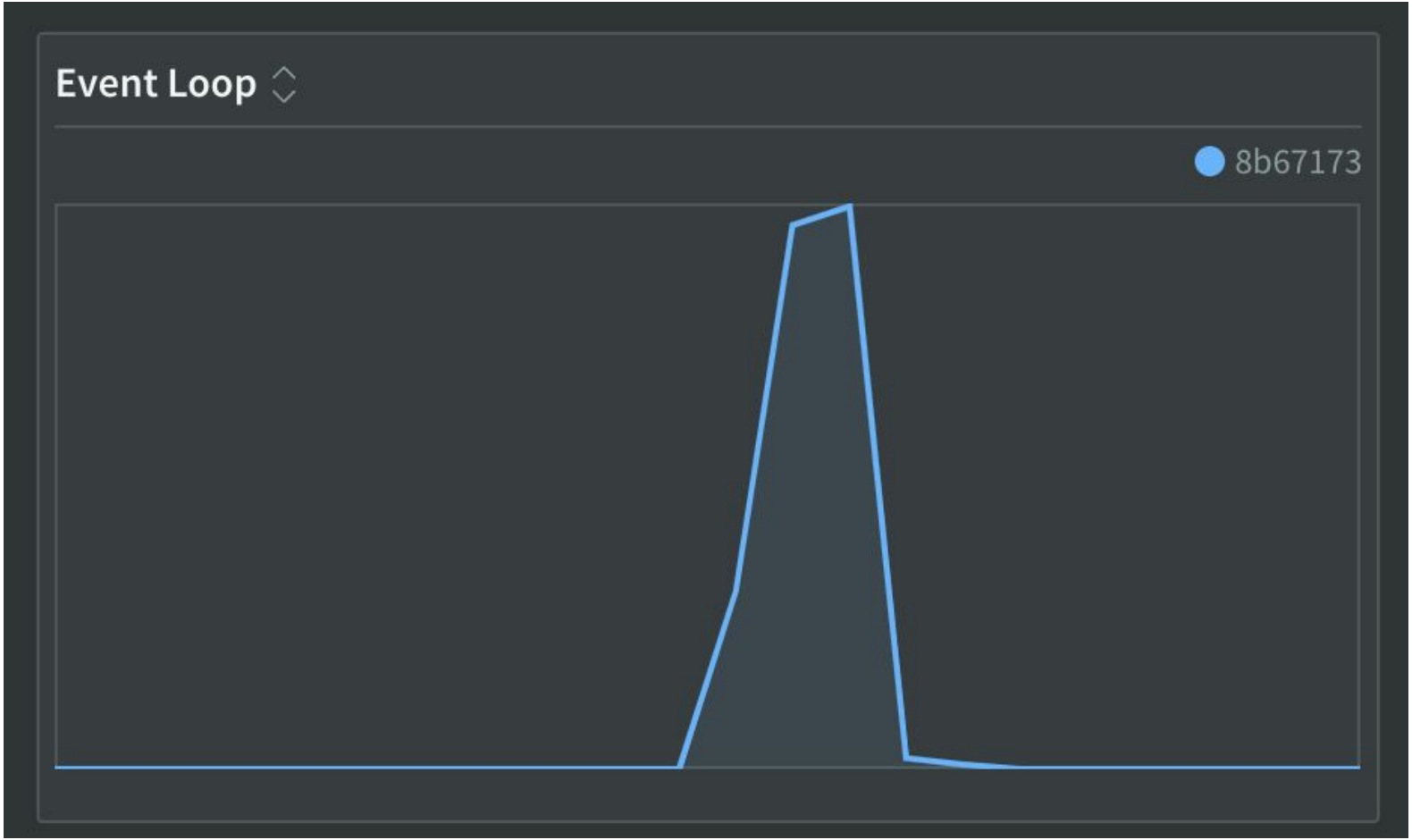
CANCEL START PROFILE

I/O Example

Running IO Sync Example

```
$ cd 3-io/

$ nsolid sync.js
Server is running on port 3000
In a separate terminal run: node benchmark.js
```

What do you spot by looking at the graphs?


Event Loop
8b67173

Rule #1
Don’t block the
event loop

please

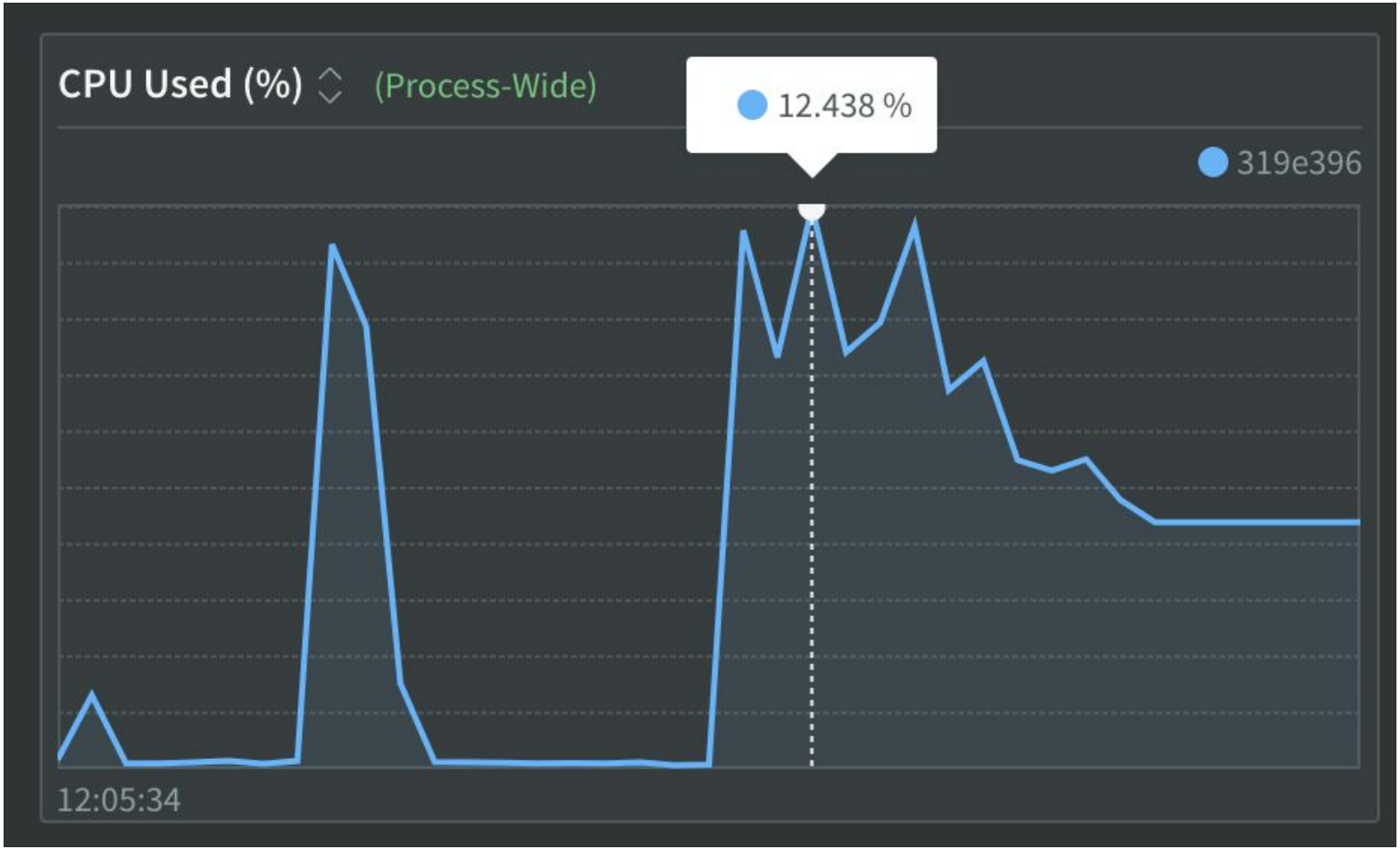
Using --trace-sync-io

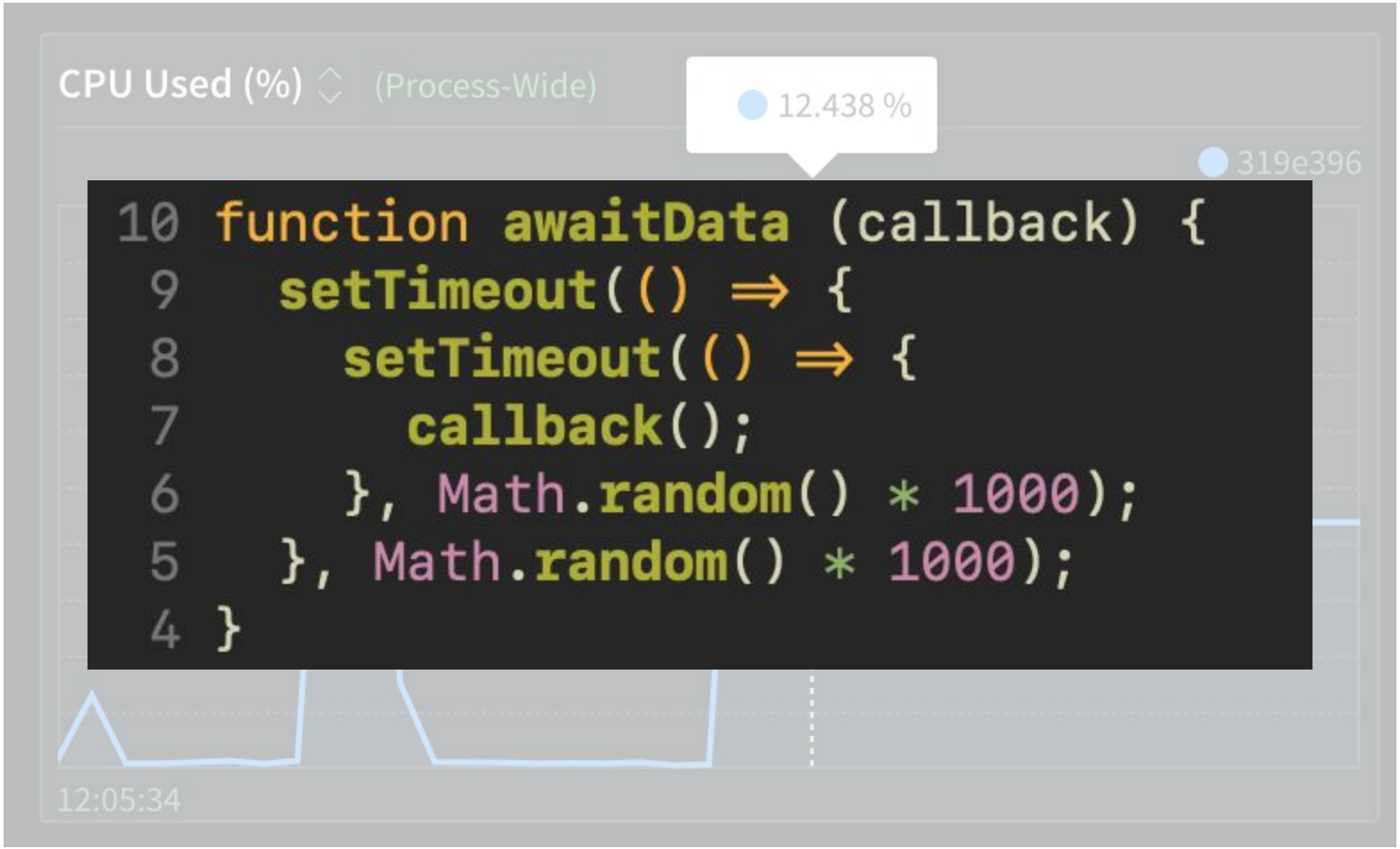
```
$ nsolid --trace-sync-io sync.js

(nsolid:68275) WARNING: Detected use of sync API
    at openSync (node:fs:573:18)
    at syncOperation (/Users/rafaelgss/repos/os/nsolid-workshop/3-io/sync.js:26:21)
    at /Users/rafaelgss/repos/os/nsolid-workshop/3-io/sync.js:15:3
    at handle (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/layer.js:95:5)
    at next (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/route.js:149:13)
    at dispatch (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/route.js:119:3)
    at handle (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/layer.js:95:5)
    at /Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/index.js:284:15
    at process_params (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/index.js:346:12
    at next (/Users/rafaelgss/repos/os/nsolid-workshop/node_modules/express/lib/router/index.js:280:10)
```

Running IO Promise Example

```
$ nsolid promises.js
Server is running on port 3000
In a separate terminal run: node benchmark.js
```

What do you spot by looking at the graphs?


CPU Used (%)
(Process-Wide)
12.438 %
319e396
12:05:34




```
10 function awaitData (callback) {
 9   setTimeout(() => {
 8     setTimeout(() => {
 7       callback();
 6     }, Math.random() * 1000);
 5   }, Math.random() * 1000);
 4 }
```

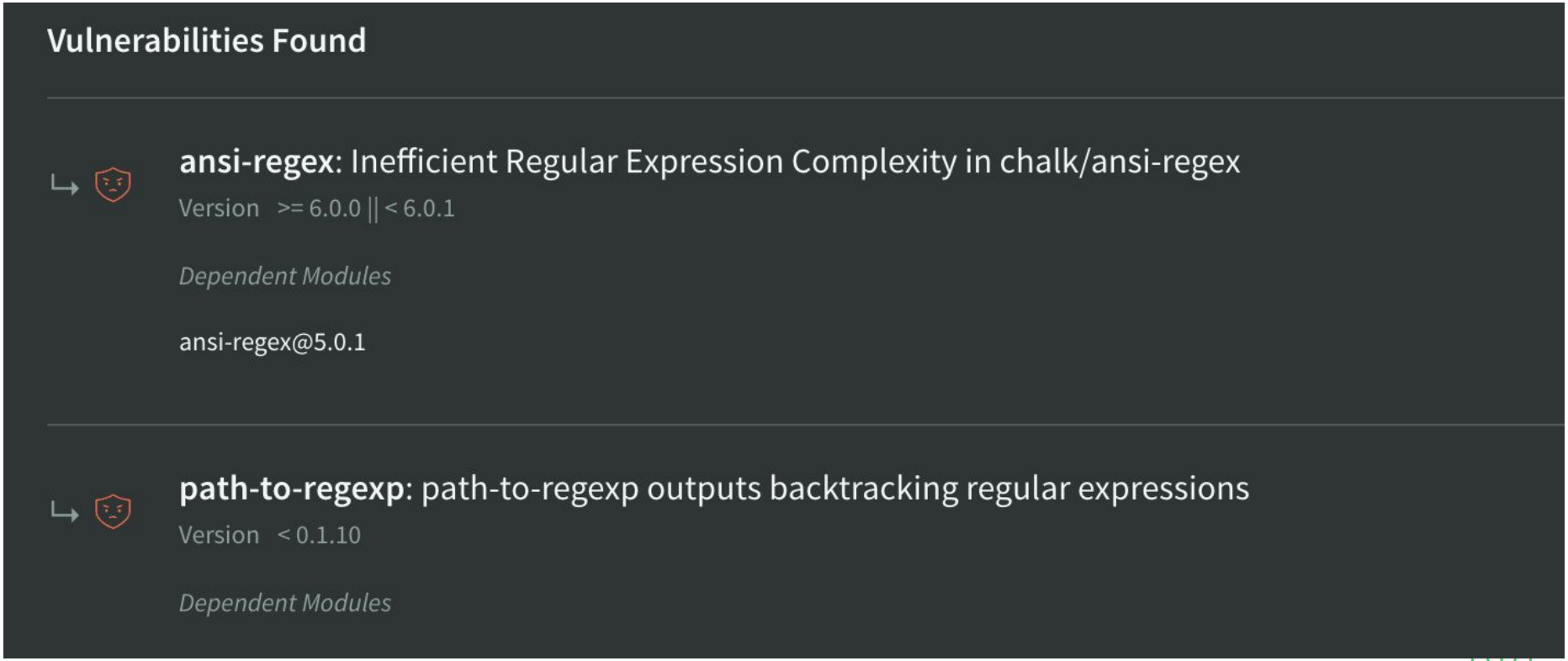
When running a benchmark, it’s expected that your **CPU load** is **HIGH**

N|Solid can do more!

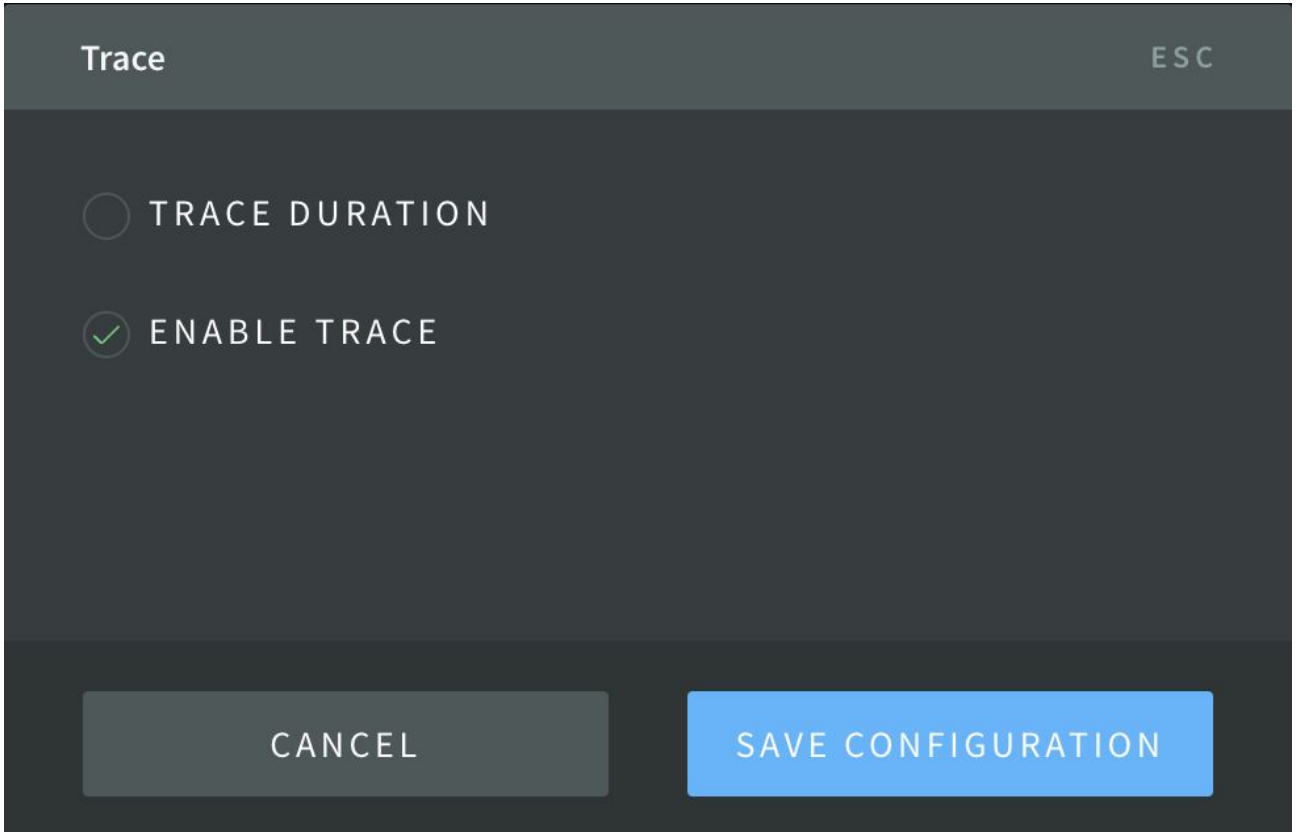
and faster

NODESOURCE

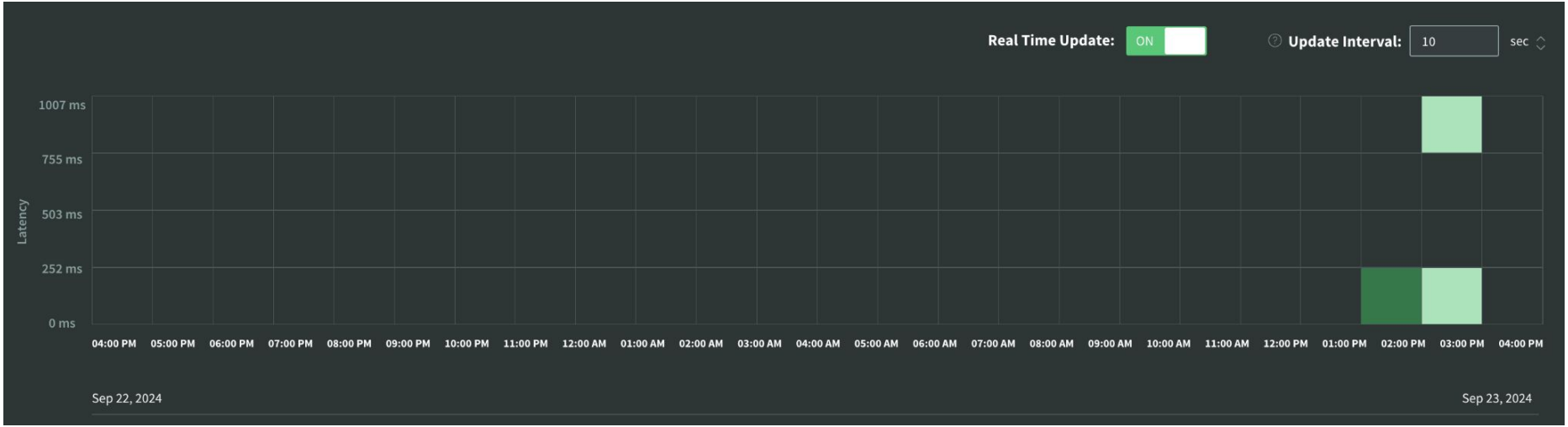
Monitor vulnerabilities of your modules

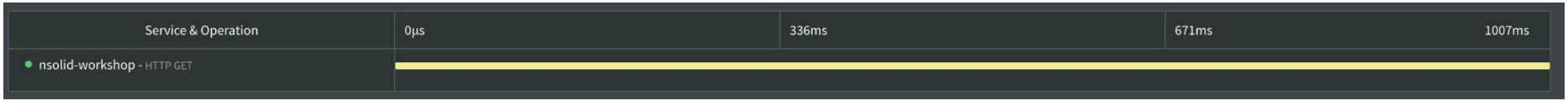


Built-in tracing system



Built-in tracing system





Thank you!

Gracias!