



introduction to profiling Node.js applications

Patrick Mueller, NodeSource

introduction to profiling Node.js applications

Patrick Mueller [@pmuellr](#), [muellerware.org](#)
senior node engineer at [NodeSource](#)

<http://pmuellr.github.io/slides/2015/12-profiling-node-intro>

<http://pmuellr.github.io/slides/2015/12-profiling-node-intro/slides.pdf>

<http://pmuellr.github.io/slides/> (all of Patrick's slides)

what kind of profiling?

- **performance** with V8's CPU profiler
- **memory** with V8's heap snapshots

profiling performance

what does V8's CPU profiler do?

- trigger profiler on / off
- when on, at regular intervals, V8 will capture current stack trace, with time stamp, and source file / line numbers
- when turned off, profiler will aggregate the information, and produce a JSON data structure for analysis tools

understanding CPU profiling

- intro: [Google Developers: Speed Up JavaScript Execution](#)
- provides times spent executing functions:
 - **self time** - time to run the function, **not** including any functions that it called
 - **total time** - time to run the function, including any functions that it called

time-line from Chrome Dev Tools

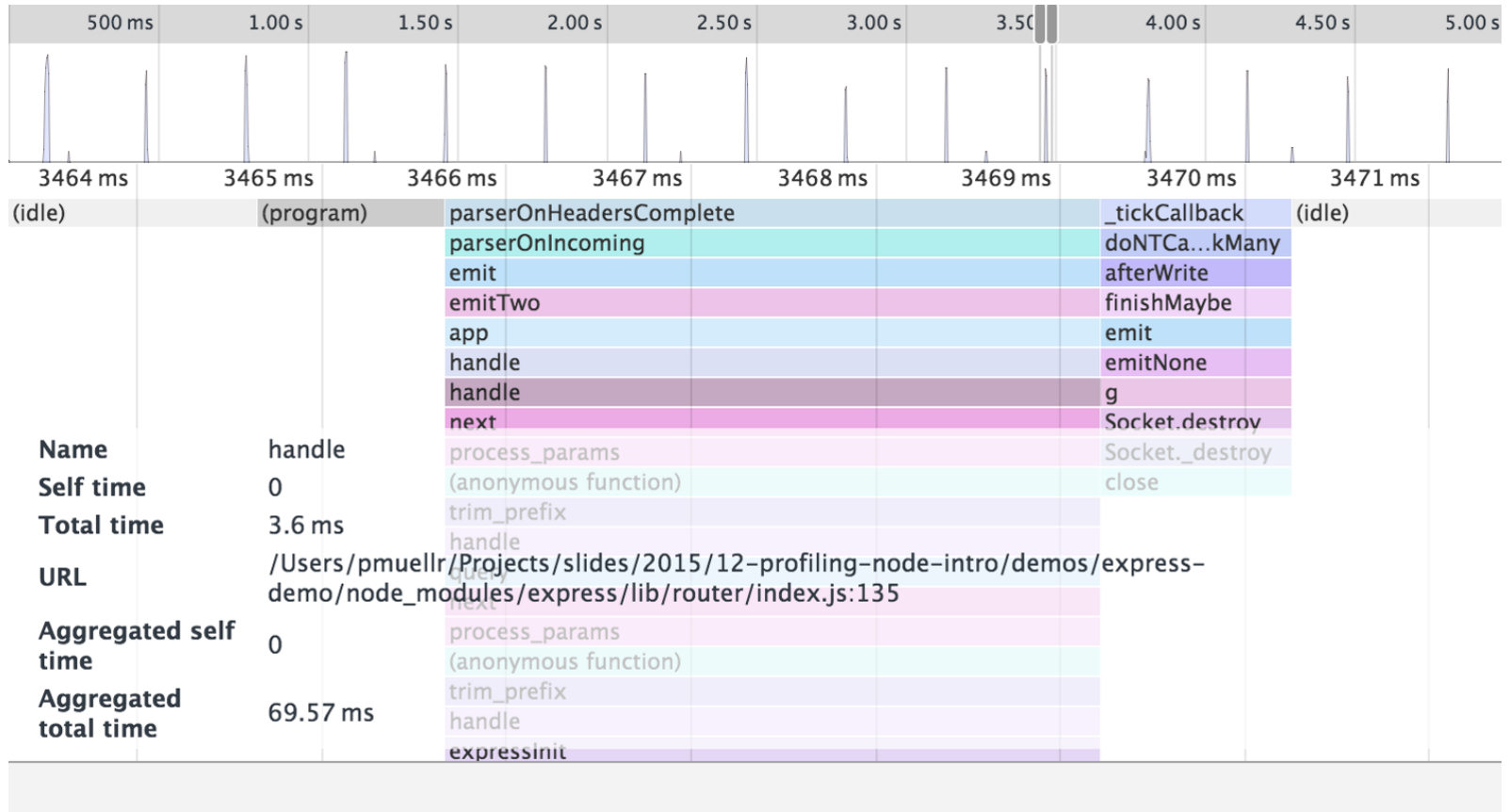
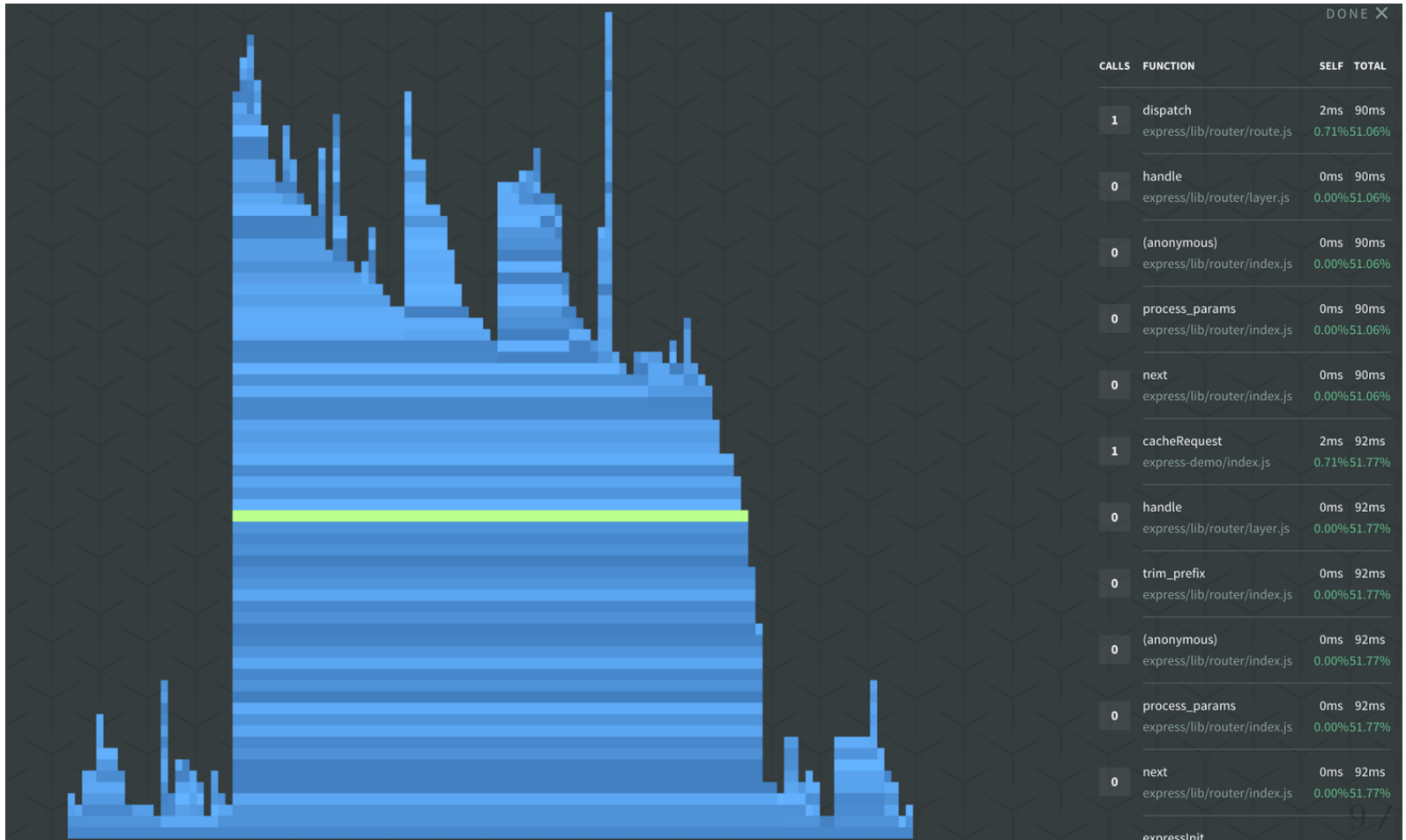


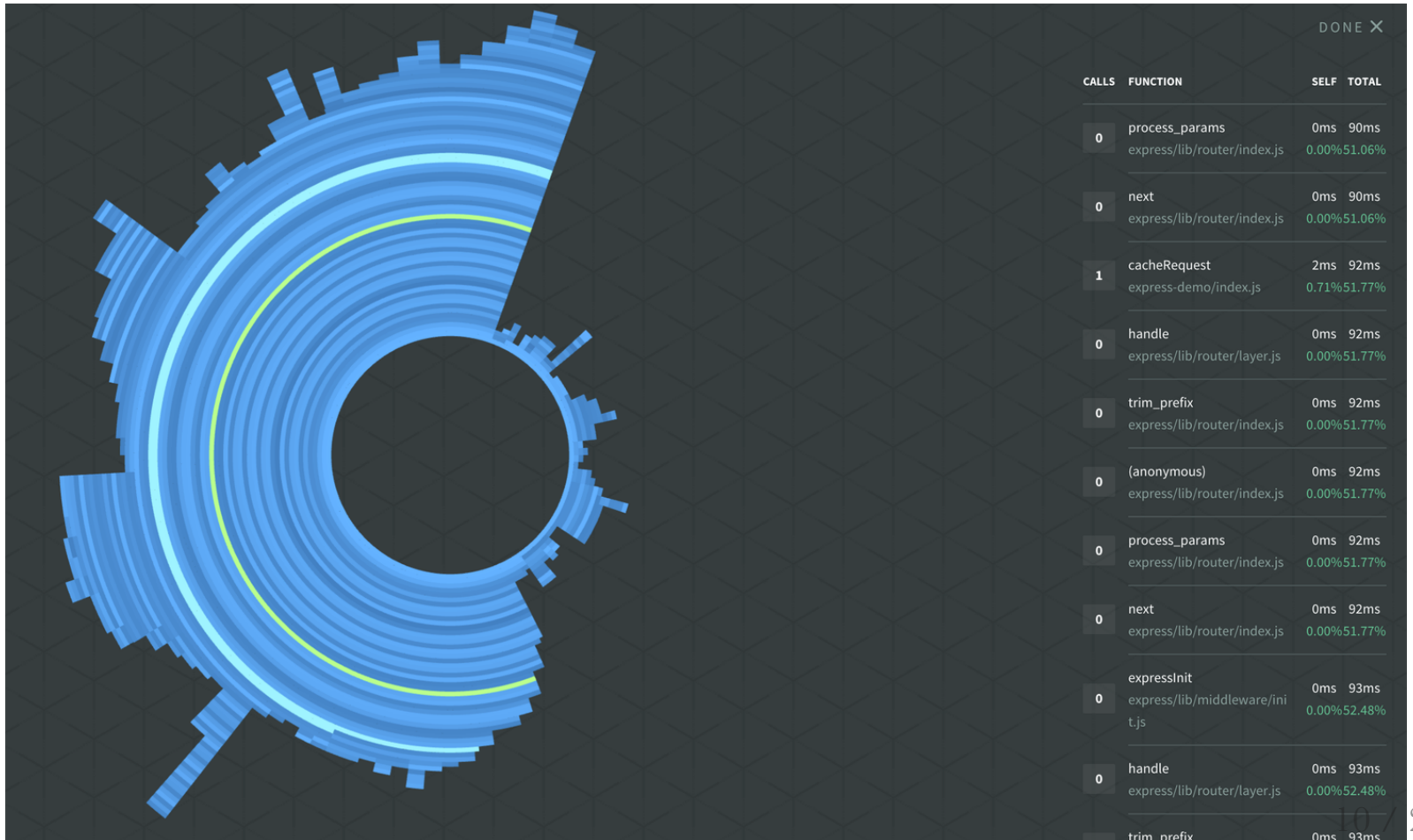
table from Chrome Dev Tools

Self ▼	Total	Function	
4831.8 ms	4831.8 ms	(idle)	(program):-1
16.3 ms 9.22%	16.3 ms 9.22%	(program)	(program):-1
12.5 ms 7.09%	12.5 ms 7.09%	(garbage collector)	(program):-1
10.0 ms 5.67%	13.8 ms 7.80%	▶ c	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
7.5 ms 4.26%	8.8 ms 4.96%	▶ Lexer.next	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
6.3 ms 3.55%	6.3 ms 3.55%	▶ spawn	(program):-1
3.8 ms 2.13%	3.8 ms 2.13%	▶ now	(program):-1
3.8 ms 2.13%	6.3 ms 3.55%	▶ pp.eat	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	18.8 ms 10.64%	▶ pp.parseExprSubsc...	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	2.5 ms 1.42%	▶ pp.finishNode	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	2.5 ms 1.42%	systemStats	nsolid.js:227
2.5 ms 1.42%	2.5 ms 1.42%	▶ posix.dirname	path.js:528
2.5 ms 1.42%	30.0 ms 17.02%	▶ parse	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	95.0 ms 53.90%	▶ app	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	8.8 ms 4.96%	▶ OutgoingMessage.end	http outgoing.js:513
2.5 ms 1.42%	2.5 ms 1.42%	▶ ServerResponse.writeHead	http server.js:159
2.5 ms 1.42%	3.8 ms 2.13%	▶ Agent.addRequest	http agent.js:109
2.5 ms 1.42%	87.5 ms 49.65%	▶ render	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
2.5 ms 1.42%	2.5 ms 1.42%	▶ slice	buffer.js:609
2.5 ms 1.42%	108.8 ms 61.70%	▶ emit	events.js:116
1.3 ms 0.71%	91.3 ms 51.77%	▶ cacheRequest	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
1.3 ms 0.71%	1.3 ms 0.71%	▶ posix.join	path.js:474
1.3 ms 0.71%	1.3 ms 0.71%	▶ pp.readString	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
1.3 ms 0.71%	7.5 ms 4.26%	▶ base.NewExpressio...	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
1.3 ms 0.71%	1.3 ms 0.71%	▶ _tokentype.types.b...	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
1.3 ms 0.71%	11.3 ms 6.38%	▶ pp.parseExprList	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos
1.3 ms 0.71%	5.0 ms 2.84%	▶ pp.parselident	/Users/pmuellr/Projects/slides/2015/12-profiling-node-intro/demos

flame graph from N | Solid



sunburst from N | Solid



how can you get CPU profiles?

- [npm v8-profiler](#) (requires instrumenting your code)
- [npm node-inspector](#)
- [StrongLoop arc](#)
- [NodeSource N|Solid](#)

demo time!

expecting faster response time in app when load testing with **ab** - **what's slowing down this app?**

- [source for the express-demo](#)
- see the instructions in [demos/README.md](#)
- using N|Solid - [getting started info](#)

profiling memory

what are V8 heap snapshots?

- JSON file describing every reachable JavaScript object in the application; taking a snapshot always starts with a garbage collection
- JSON files are ... large; figure 2x heap memory allocated by Node.js
- triggered via single native V8 call - **TakeHeapSnapshot()**

understanding heap snapshots

- intro: [Google Developers: Viewing Heap Snapshots](#)
- object sizes/counts, grouped by constructor
 - **shallow size** - the size of memory held by an object itself
 - **retained size** - the size of memory that can be freed once an object is deleted

heapmap from Chrome Dev Tools

Constructor	Distance	Objects Count ▼	Shallow Size	Retained Size
▶ ReadableState	6	8 266 2%	1 587 072 4%	1 851 584 4%
▶ (concatenated string)	4	6 780 1%	271 200 1%	310 768 1%
▶ WritableState	6	4 134 1%	793 808 2%	1 324 416 3%
▼ TagRequest	10	4 133 1%	99 192 0%	99 376 0%
▶ TagRequest @4987	11		24 0%	24 0%
▶ TagRequest @4989	10		24 0%	208 0%
▶ TagRequest @5053	11		24 0%	24 0%
▶ TagRequest @5101	11		24 0%	24 0%
▶ TagRequest @5149	11		24 0%	24 0%
▶ TagRequest @5197	11		24 0%	24 0%
▶ TagRequest @7635	11		24 0%	24 0%
▶ TagRequest @7683	11		24 0%	24 0%
▶ TagRequest @9239	11		24 0%	24 0%

Retainers					
Object	Distance ▲	Shallow Size	Retained Size		
▼ __tag in IncomingMessage @4975	10	240 0%	6 320 0%		
▼ [19] in Array @166833	9	32 0%	25 882 456 58%		
▼ Requests in system / Context @71747	8	80 0%	25 882 768 58%		
▼ context in function pingServer() @71753	7	72 0%	1 584 0%		
▼ _repeat in Timeout @166413	6	144 0%	1 728 0%		
▼ _idlePrev in Timer @1105	5	32 0%	224 0%		
▼ [333] in @66753	4	56 0%	208 0%		
▼ lists in system / Context @37847	3	224 0%	960 0%		
▼ context in function () @5879	2	72 0%	72 0%		
▶ clearInterval in @583	1	48 0%	4 968 0%		
▶ value in system / PropertyCell @37929	3	32 0%	32 0%		
▶ clearInterval in @66411	4	56 0%	56 0%		
▶ 22 in (map descriptors)[] @65155	6	272 0%	272 0%		
▶ context in function () @5865	2	72 0%	72 0%		
▶ context in function () @5845	2	72 0%	72 0%		

what kind of output can you get?

- large JSON file - could be 100's of MB;
figure 2x allocated heap
- can "diff" snapshots to help identify leaks
- can drill into or out from references in
Chrome Dev Tools; references / referenced
by

how can you get heap snapshots?

- [npm v8-profiler](#) (requires instrumenting your code)
- [npm node-inspector](#)
- [StrongLoop arc](#)
- [NodeSource N|Solid](#)

demo time!

this app seems to be leaking memory - **what objects are leaking?**

- [source for the express-demo](#)
- see the instructions in [demos/README.md](#)
- using N|Solid - [getting started info](#)

profiling tips

profiling performance

- look for **width** in trace visualizations; height only shows stack trace which may not have any perf consequences
- "script" profiling a web server: start profile, run load tester, stop profile
- use node/v8 option **--no-use-inlining** to turn off function inlining; stack traces may make more sense (but no inlining!)

profiling memory

- easiest way to find a memory leak:
 - take a heap snapshot; run load tester; take another heap snapshot; diff in Chrome Dev Tools
- 'tag' objects you think might be leaking w/easy to find class:

```
req.__tag = new TagRequest()
```

fin

profiling Node.js applications

