



TurboFan: A new code generation architecture for V8

Benedikt Meurer
Google Munich
[@bmeurer](https://twitter.com/bmeurer)





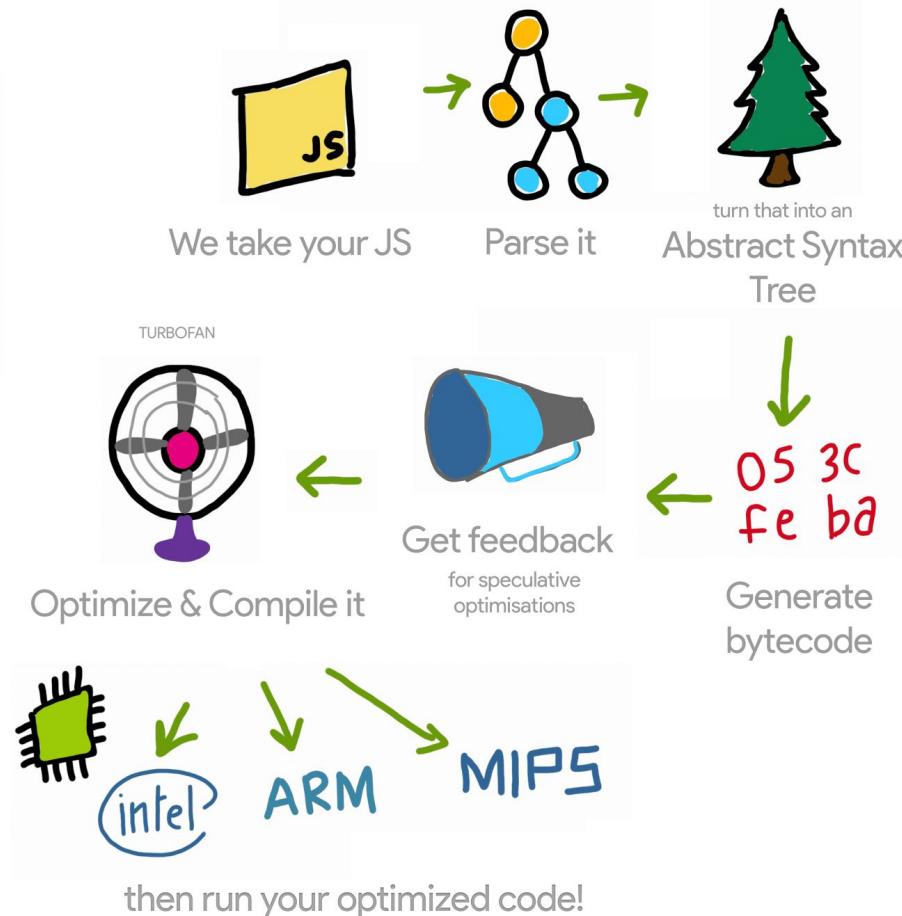
Ignition and TurboFan

enabled in

Chrome Dev / Canary!



How
Works



By @addyosmani

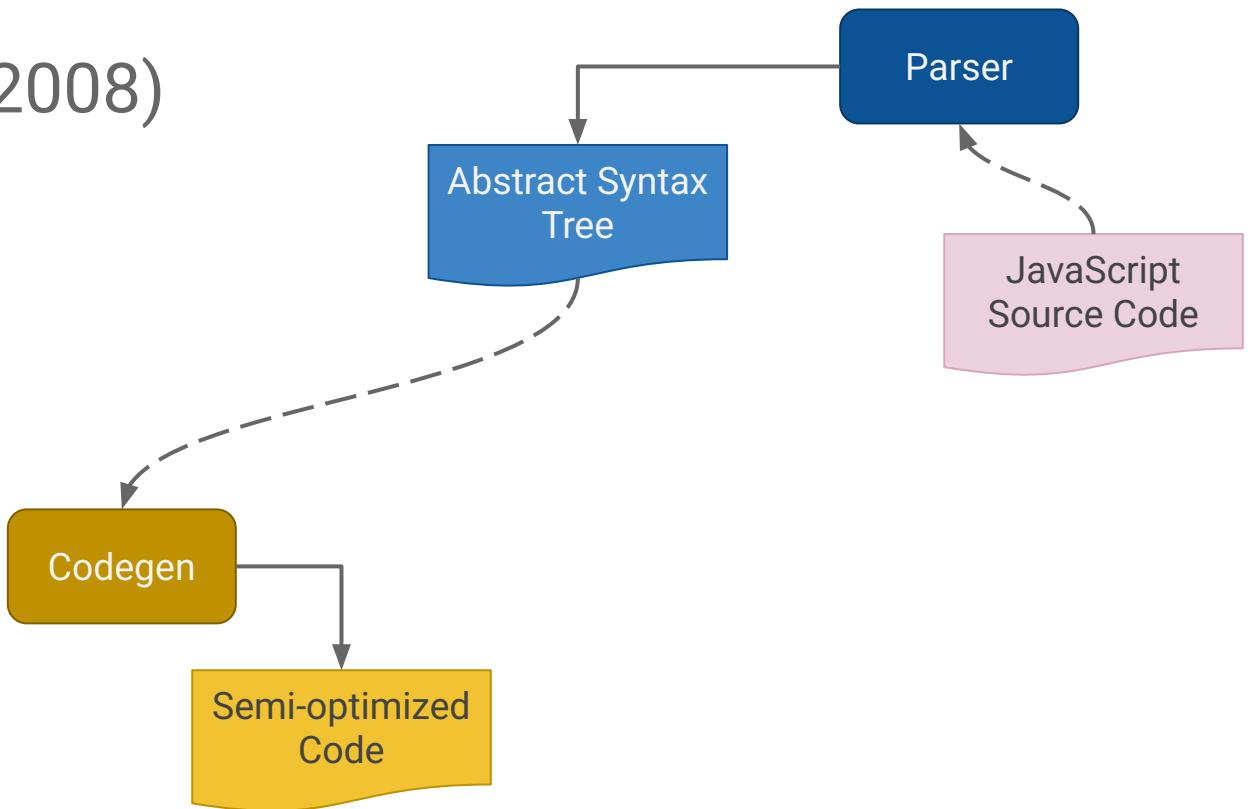
Why a new code generation architecture?

- Improve baseline performance
- Make performance predictable
- Reduce page load time
- Reduce memory usage
- Reduce complexity

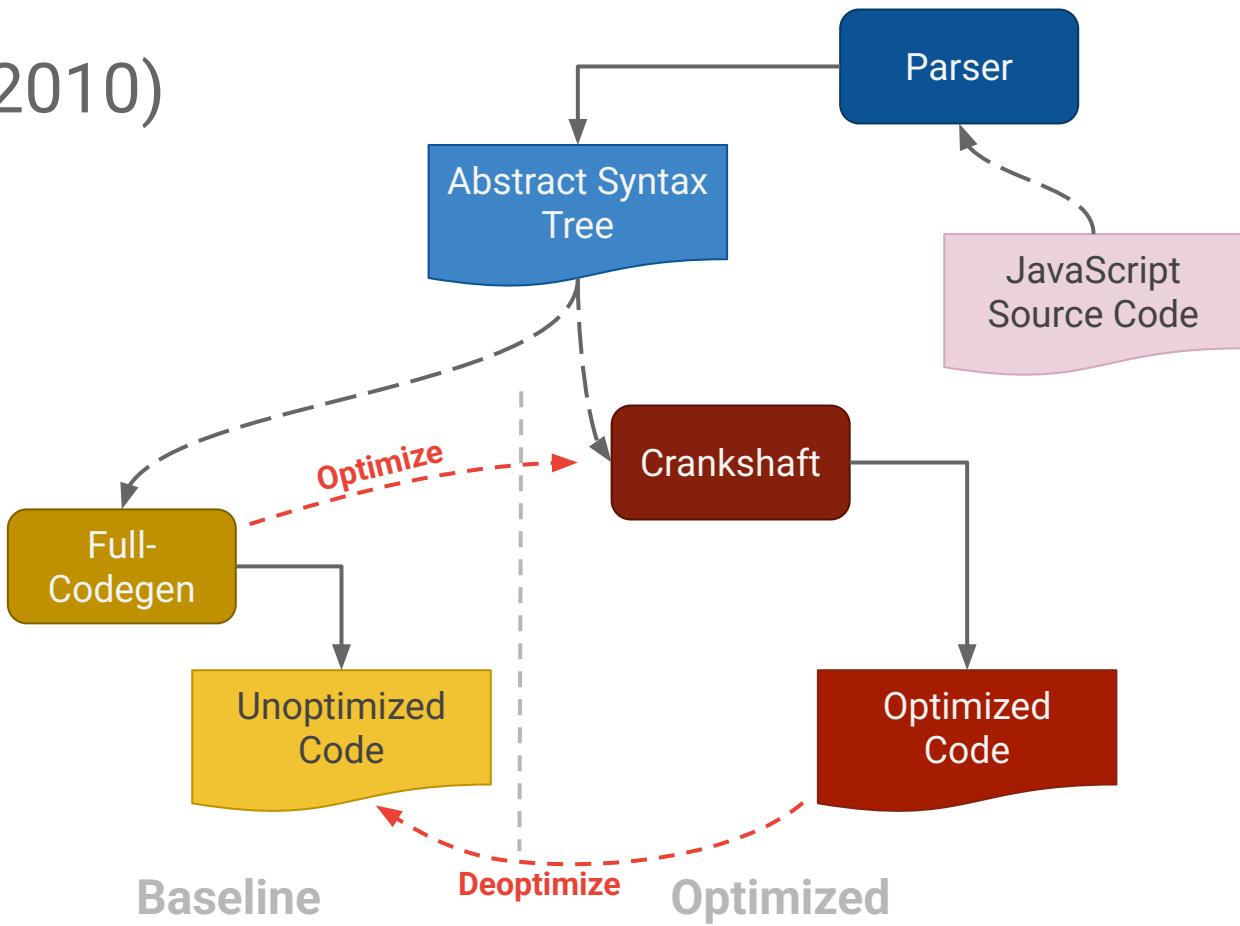


A bit of history...

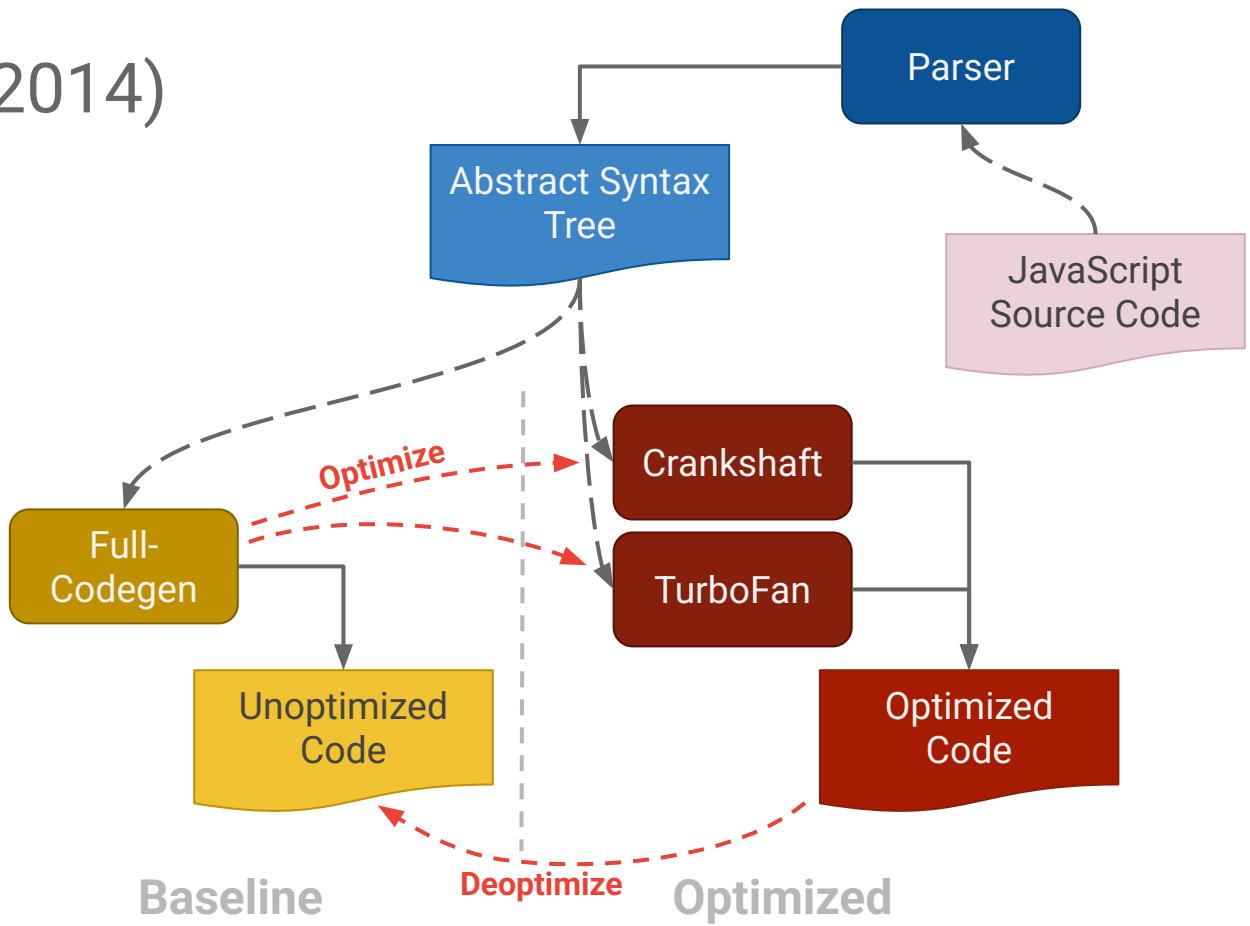
Compiler pipeline (2008)



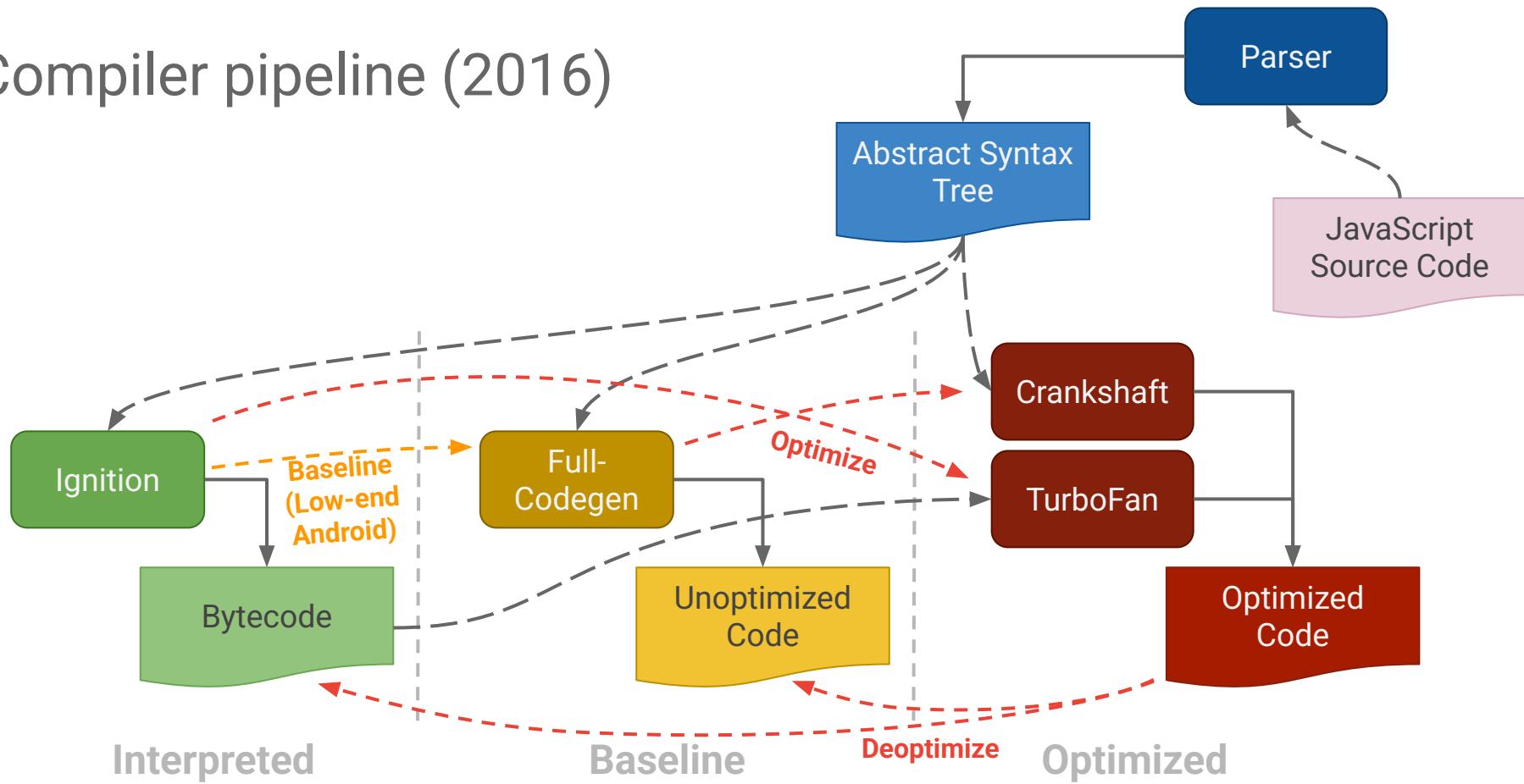
Compiler pipeline (2010)



Compiler pipeline (2014)

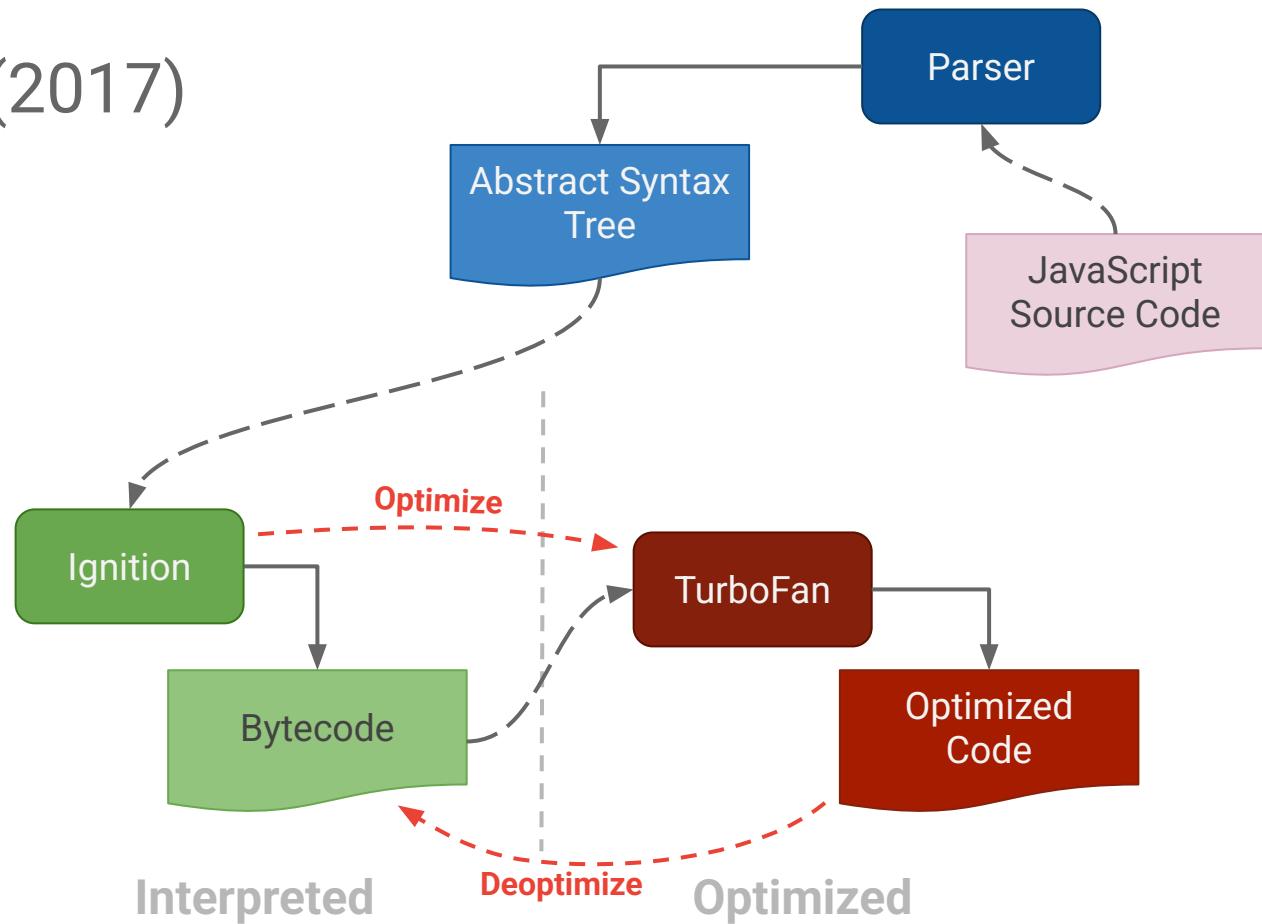


Compiler pipeline (2016)



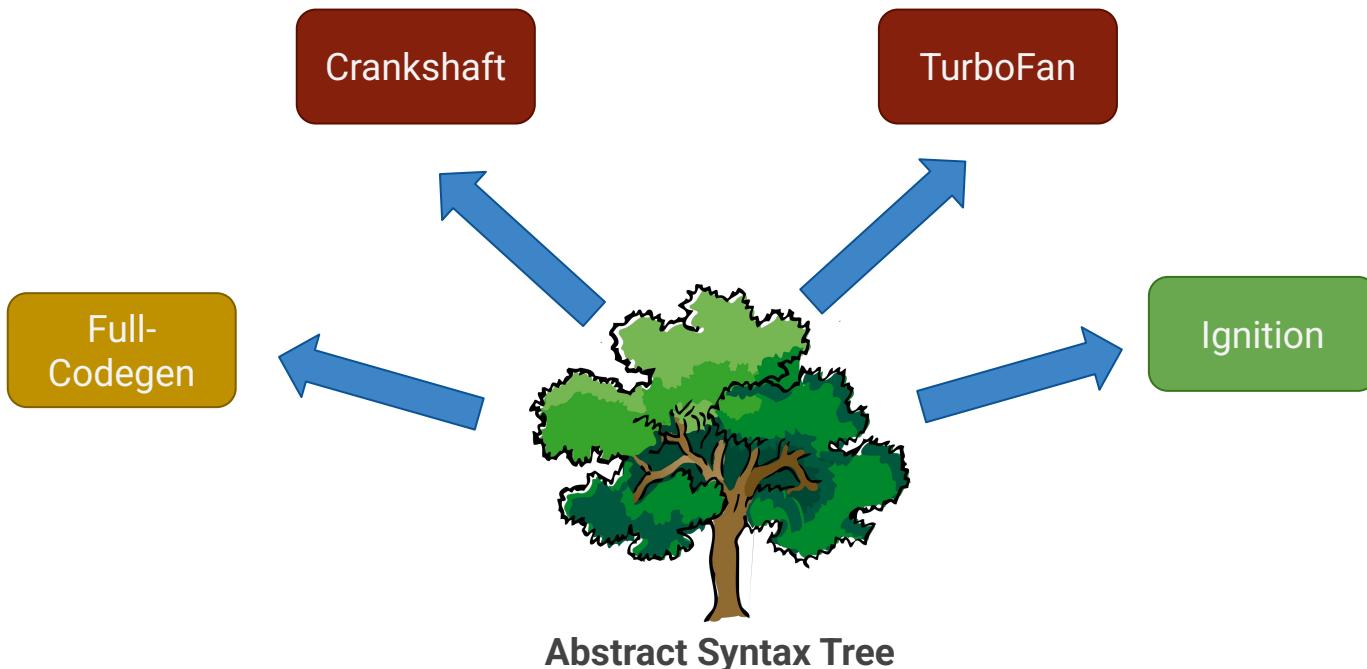


Compiler pipeline (2017)



Reduce complexity

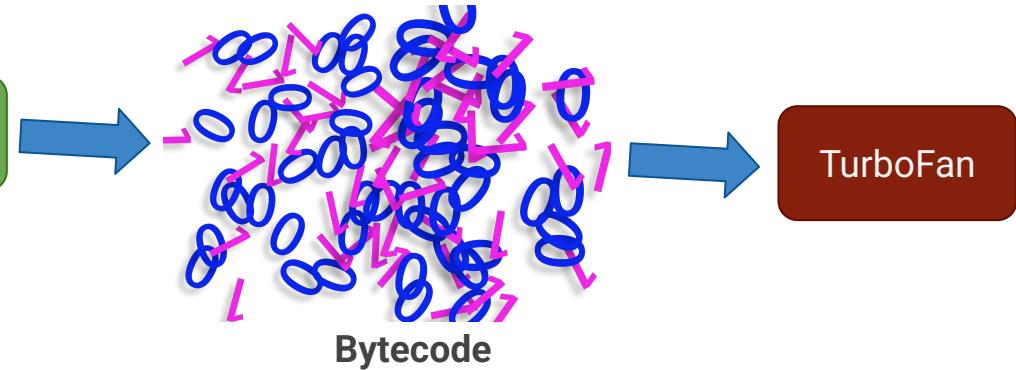
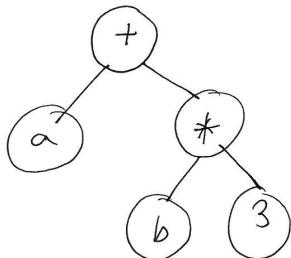
Abstract Syntax Tree



Abstract Syntax Tree



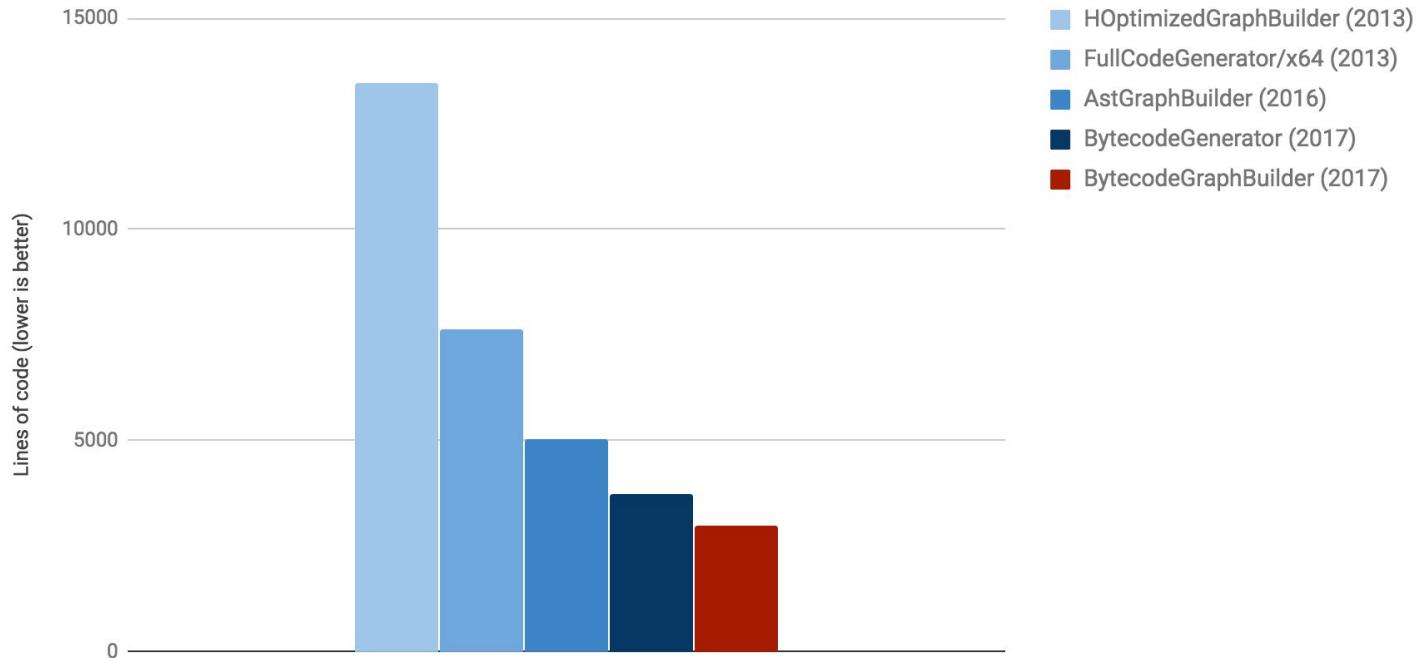
Abstract Syntax Tree



0 : 87	StackCheck
1 : 03 03	LdaSmi [3]
3 : 2e 02 02	Mul a1, [2]
6 : 2c 03 03	Add a0, [3]
9 : 8b	Return

const f = (a, b) => a + b * 3;

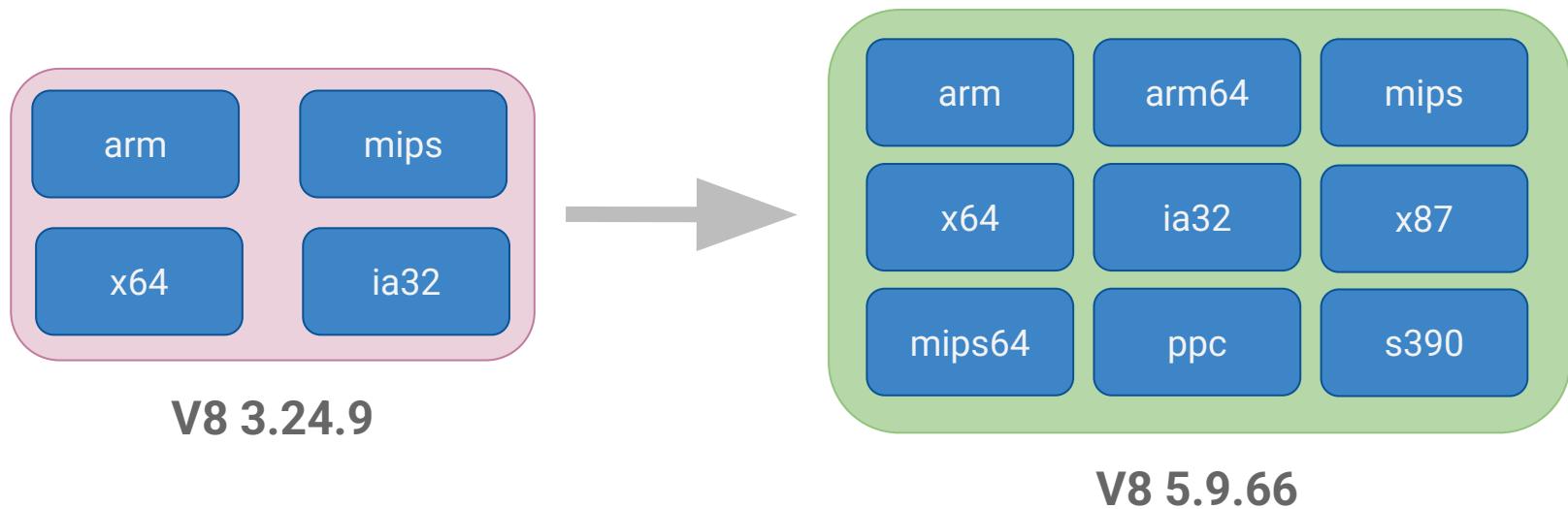
Abstract Syntax Tree



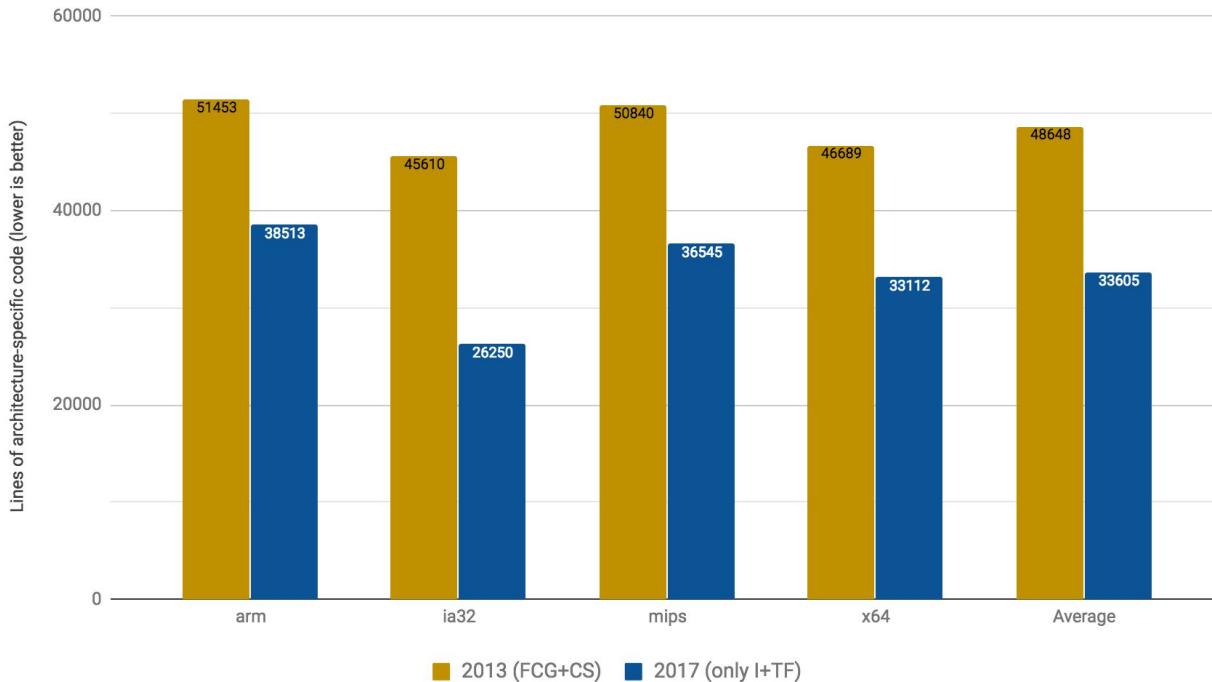
Supported architectures



Supported architectures



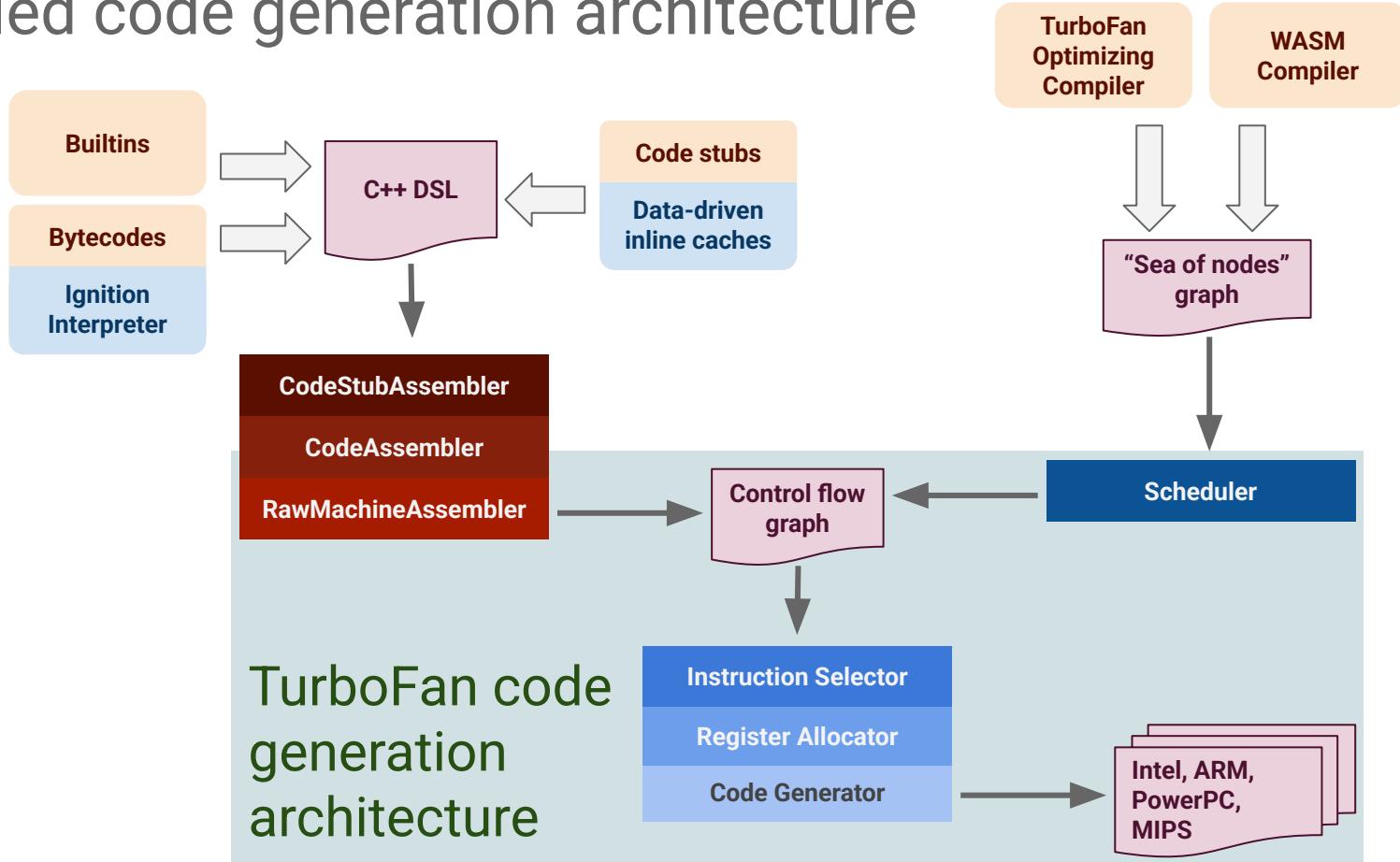
Architecture-specific LOCs



29%

less architecture-specific code from 2013 to 2017

Unified code generation architecture



Unified code generation architecture

- (not just an) Optimizing compiler
- Interpreter bytecode handlers
- Builtins (Object.create, Array.prototype.indexOf, etc.)
- Code stubs / IC subsystem
- WebAssembly code generation

Unified code generation architecture

- More opportunities for performance improvements
- Fewer bugs (no more register allocation by hand)
- Easier to port to new architectures

Predictable Performance

Optimization Killers

The screenshot shows a web browser window with three tabs open:

- Optimization killers**: A GitHub page with the URL <https://github.com/petkaantonov/bluebird/wiki/Optimization-killers>. It contains sections like "Introduction" and "Some V8 background". It also includes a snippet of assembly code:

```
mov eax, a
mov ebx, b
call RuntimeAdd
```

A note below says: "In other words it just calls a runtime function. If a and b are numbers, something like this:"
- V8 bailout reasons**: A GitHub page with the URL <https://github.com/v8/v8-bailout-reasons>. It has sections like "What this is about" and "Bailout reasons". The "Bailout reasons" section lists items such as "Assignment to parameter in arguments object", "Bad value context for arguments value", etc.
- Performance Tips for JavaScript**: A page from <https://www.html5rocks.com/en/tutorials/speed/v8/>. It features a "Table of Contents" on the left and a main content area on the right. The main content starts with "Therefore:" followed by a bullet point and a code block:
 - Put perf-sensitive code into a nested function if you have try {} catch {} blocks:

```
function perf_sensitive() {
  // Do performance-sensitive work here
}

try {
  perf_sensitive()
} catch (e) {
  // Handle exceptions here
}
```

This guidance will probably change in the future, as we enable try/catch blocks in the optimizing compiler. You can examine how the optimizing compiler is bailing out on functions by using the "--trace-opt" option with d8 as above, which gives you more information on which functions were bailed out:

```
d8 --trace-opt primes.js
```

De-optimization

Finally, the optimization performed by this compiler is speculative - sometimes it

Optimization Killers

- Generators and **async** functions
- **for**-**of** and destructuring
- **try-catch** and **try-finally**
- Compound **let** or **const** assignments
- Object literals that contain **__proto__**, or **get** or **set** declarations.
- **debugger**, **with** statements
- Literal calls to **eval()**
- ...

Optimization Killers - arguments object

```
const arr = new Array(4000);
function mymax() { return Math.max.apply(undefined, arguments); }
for (let i = 0; i < 2000; ++i) mymax(...arr);
```

```
$ node --trace-opt --trace-deopt apply.js
```

```
...
```

```
[deoptimizing (DEOPT eager): begin 0x34e770317ea1 <JS Function mymax
(SharedFunctionInfo 0x2793cb1f4031> (opt #5) @2, FP to SP delta: 24, caller sp:
0x7fff5fbf6378]
```

```
...
```

```
[disabled optimization for 0x2793cb1f4031 <SharedFunctionInfo mymax>, reason:
Optimized too many times]
```

```
$
```

Too many arguments!

Optimization Killers - arguments object

```
var callbacks = [
  function sloppy() {},
  function strict() { "use strict"; }
];

function dispatch() {
  for (var l = callbacks.length, i = 0; i < l; ++i) {
    callbacks[i].apply(null, arguments);
  }
}
```

[disabled optimization for ... <SharedFunctionInfo dispatch>,
reason: Bad value context for arguments value]

Optimization Killers

Setup

```
1  const N = 1000;~  
2  const arr = Array.apply(null, { length: N }).map(Number.call, Number);~  
3  let total = 0;~
```

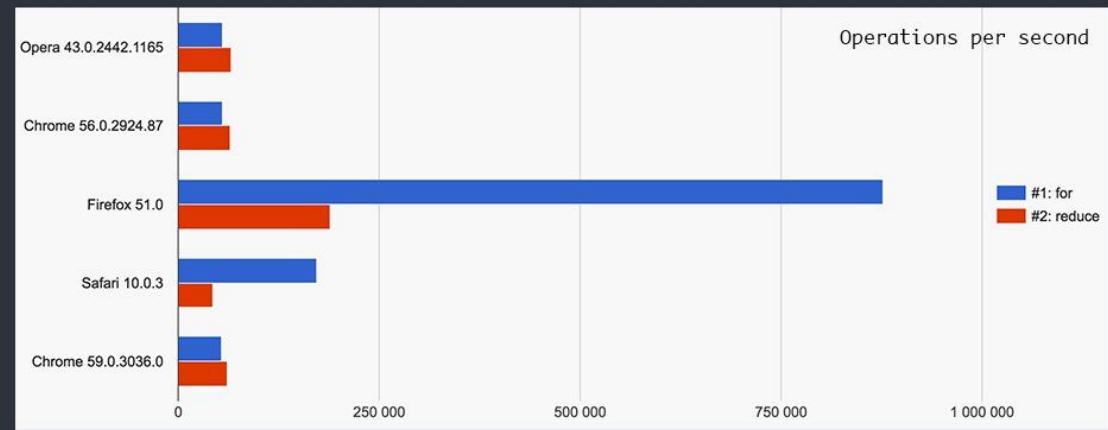
For Loop

```
1  for(let i = 0; i < arr.length; i++) {~  
2    total += arr[i];~  
3  }~
```

Array.prototype.reduce

```
1  total = arr.reduce((a, b) => a + b, 0);~
```

Benchmark results (higher is better)



Baseline Performance

Baseline Performance - Builtins



Shubheksha
@ScribblingOn

TIL I learnt a simple for loop is faster than forEach in JS, thanks to [@TheLarkInn](#). Now trying to find out why.



Aaron Powell
@slace

[@jdalton](#) watched the video. Reasons make sense. I find it strange that [].foreach is slower than while loops (my tests agree, still strange)

Follow



Jani Tarvainen
@velmu

Follow

"[Node.js 6.0] native Promise performance is still around six times slower than Bluebird"
[blog.wikimedia.org/2017/02/17/nod...](http://blog.wikimedia.org/2017/02/17/nodejs-native-promises/) /cc
@PetkaAntonov #nodejs

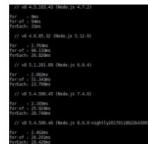
Follow



Sean Larkin
@TheLarkInn

Following

Do I have some fun low hanging perf fruit for someone looking to make [#webpack](#) build faster.
Change forEach to for() with let 😍😍



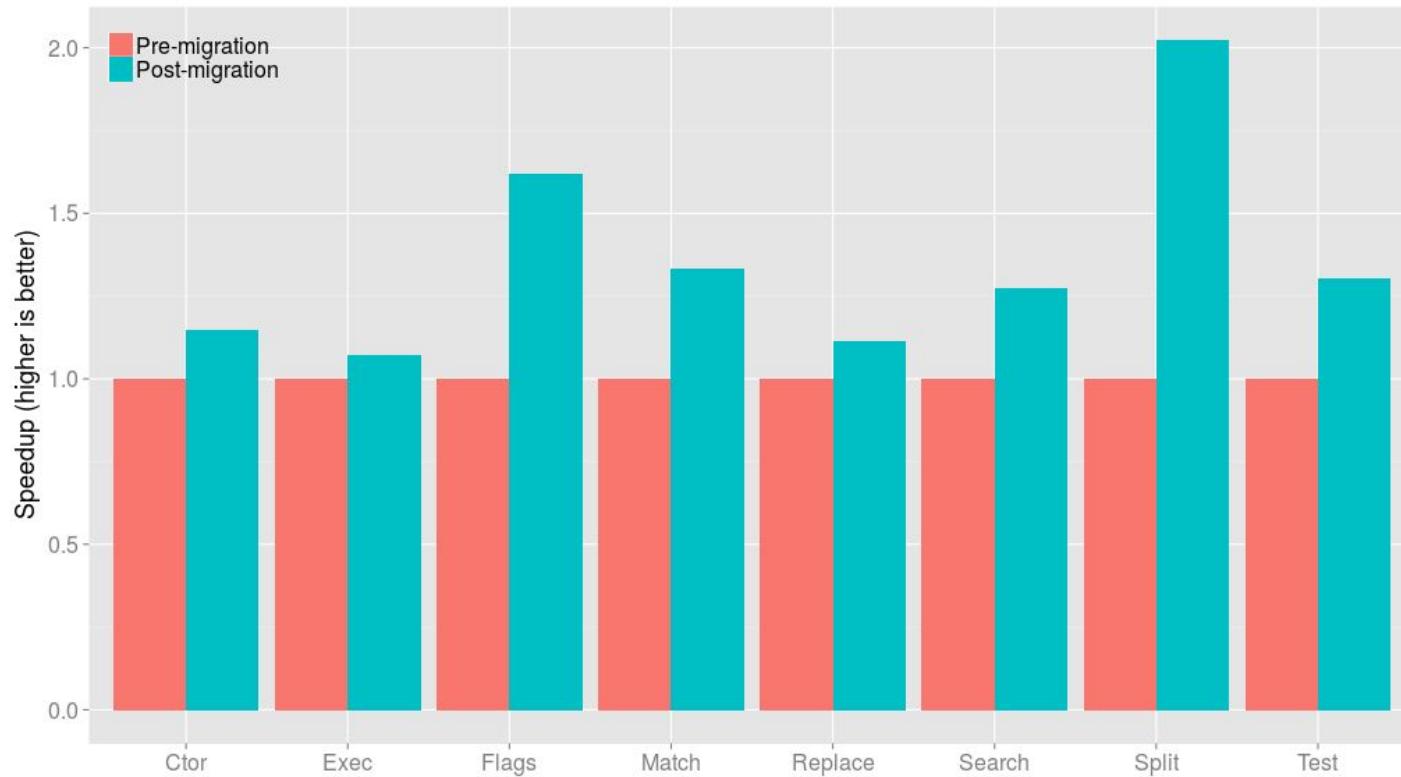
vmb @vsemozhetbyt

@bmeurer @TheLarkInn @ScribblingOn Seems v8 in @bmeurer's bench has let/const cycle optimization. Gist/results (var)
gist.github.com/vsemozhetbyt/7...

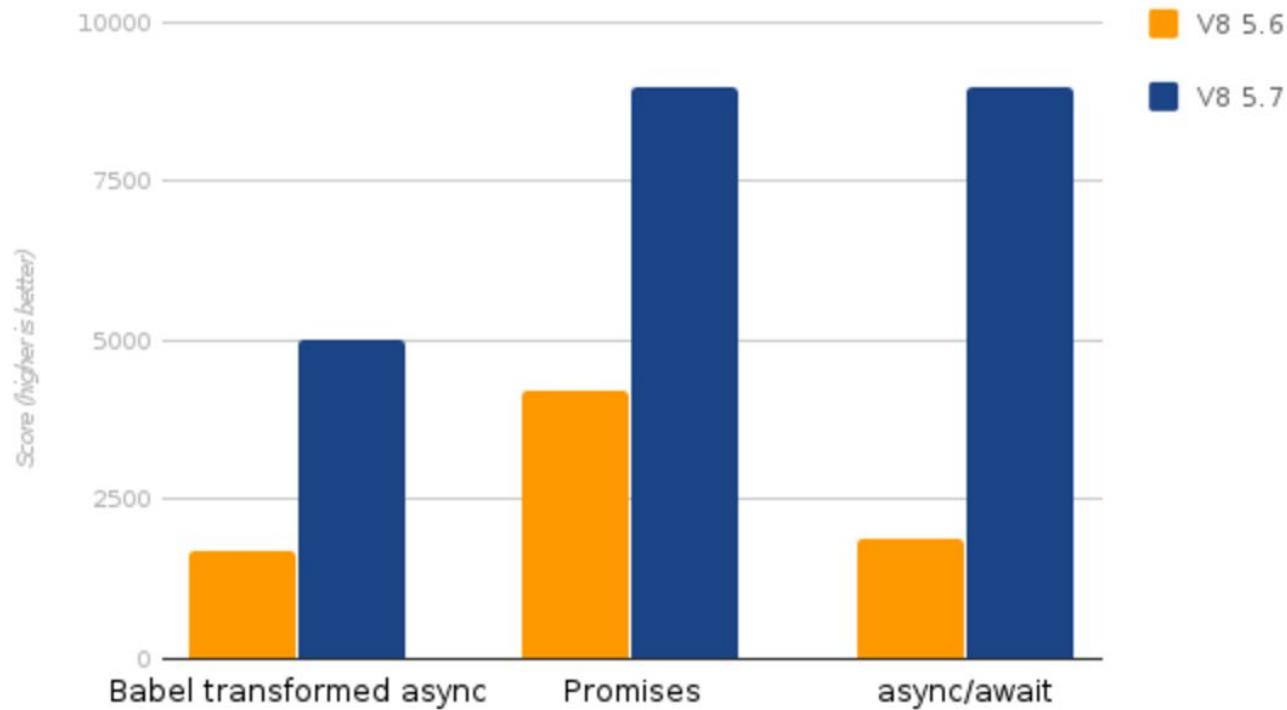
Baseline Performance - Builtins

- Expectation mismatch: “builtins should be fast”
- TurboFan’s CodeStubAssembler basis for fast builtins
- Ideally no performance cliff

Baseline Performance - RegExp Builtins



Baseline Performance - Promises & async/await



Baseline Performance

- Baseline performance matters
- Optimize too early is costly
 - Might deoptimize quickly because type feedback not stable yet
 - Optimization is expensive
- Generated code quality depends on quality and stability of type feedback

Improve Startup Time

Average Page



Improve Startup Time

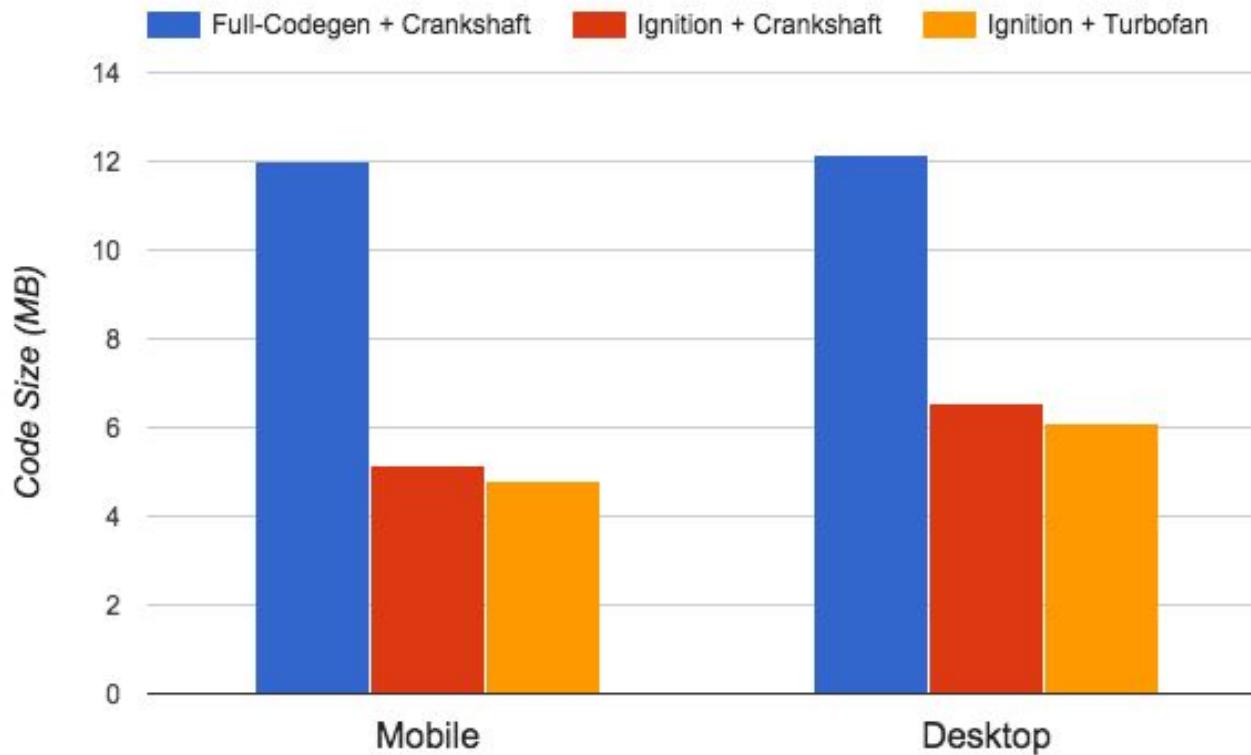
- Bytecode faster to generate
- Better suited for smaller icache - (low-end) mobile
- Parse only once, optimize from bytecode
- Optimize less aggressively - better baseline performance
- Data-driven ICs reduce slow path cost

Reduce Memory Usage

Reduce Memory Usage

- Memory is precious on mobile
- Android devices with 512MiB - 1GiB are common
- Emerging markets
- Ignition code is up to **8x** smaller than Full-Codegen code (arm64)

Reduce Memory Usage



Test

Chrome Dev / Canary

The screenshot shows a Google Chrome window with two tabs open. The main tab is a landing page for Google Chrome Canary, featuring a large 'Download Chrome' button and information about experimental features like SharedArrayBuffer support and V8 features. The second tab is titled 'chrome://flags/#enable-v8-future' and displays several experimental flags with dropdown menus. One flag, 'Experimental Features in V8 JavaScript execution', is set to 'Enabled'. Another flag, 'GPU rasterization', is set to 'Default'. A message at the bottom indicates changes will take effect after relaunch.

Secure <https://www.google.com/chrome/browser/canary.html?platform=mac>

Get on the bleeding edge

Google Chrome Canary has the new features you need. Be forewarned: it's designed for developers and can break things. It's not for everyone.

Download Chrome

For Mac OS X
You can also download Chrome for [Windows](#), [Linux](#), or [Android](#).

chrome://flags/#enable-v8-future

Experimental enabled SharedArrayBuffer support in JavaScript. Mac, Windows, Linux, Chrome OS, Android
Enable SharedArrayBuffer support in JavaScript. [#shared-array-buffer](#)

Default

Experimental Features in V8 JavaScript execution Mac, Windows, Linux, Chrome OS, Android
Enable experimental features in V8 for JavaScript execution. [#enable-v8-future](#)

Enabled

Disable the new JavaScript Compilation Pipeline Mac, Windows, Linux, Chrome OS, Android
Disable V8's new Ignition interpreter and TurboFan compiler for JavaScript execution. [#disable-v8-ignition-turbo](#)

Default

GPU rasterization Mac, Windows, Linux, Chrome OS, Android
Use GPU to rasterize web content. Requires impl-side painting. [#enable-gpu-rasterization](#)

Default

GPU rasterization MSAA sample count. Mac, Windows, Linux, Chrome OS, Android

Your changes will take effect the next time you relaunch Google Chrome.

RELAUNCH NOW

Node vee-eight-lkgr (Ubuntu/x86_64 build)

github.com/v8/node

The screenshot shows the 'Builder: V8 - node.js integration (stats)' page. It displays the following sections:

- Currently Building:**
 - 4494 ETA: 02:53:53 [5 mins, 32 secs] [Running for 19 mins, 38 secs] build and test node.js
- Pending Build Requests:**
 - (Mar 23 02:29:01, waiting 19 mins, 20 secs) 90d28637dc127e2a87c72d95317dcc56bce9a603 (wiktorg@google.com)
- Recent Builds:**

Time	Revision	Result	Build #	Info
Mar 23 01:54	8aa3459f2098d6a489d357b7837170fd9872fc67	success	#4493	Build successful
Mar 22 20:22	e1fb93b8fbe860bdbb94834d12ba90b27bbc9a2	success	#4492	Build successful

The screenshot shows the 'Builder: V8 - node.js integration' logs page. It lists the following build steps:

- configure node.js - install
- makedirs install directory
- build and install node.js
- zipping
- gsutil upload
- Archive link
- cleanup archive
- rmmtree archive directory
- recipe result

A red arrow points from the 'Build successful' message in the recent builds table on the left to the 'download' step in the logs on the right, highlighting the connection between the build outcome and the specific log entry.

Node vee-eight-lkgr (source build)

```
git clone https://github.com/v8/node.git node-v8
cd node-v8
git checkout vee-eight-lkgr
./configure --prefix "$HOME/Applications/node-vee-eight-lkgr"
make install
```

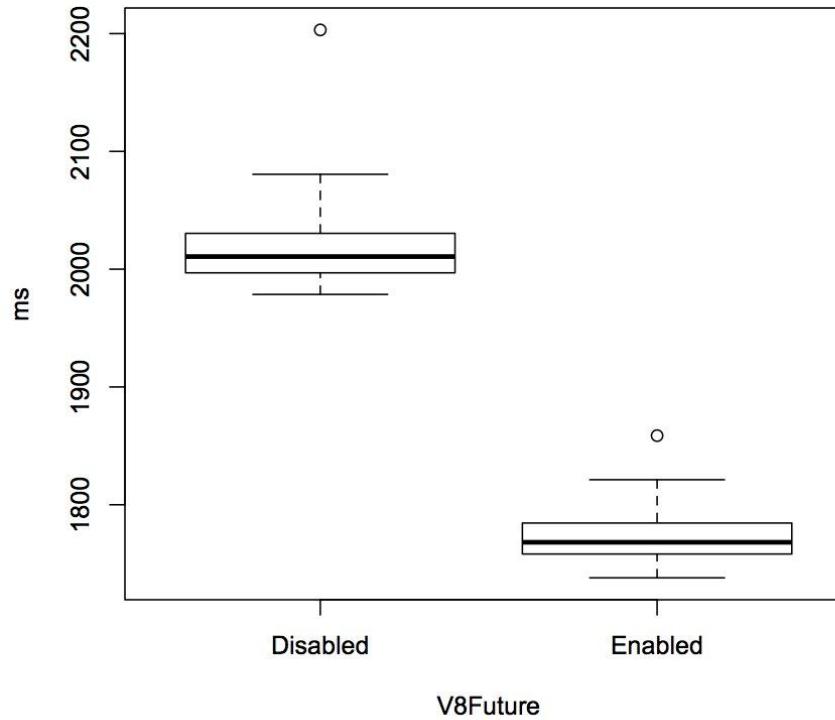
Performance Results



Page Load Time (Top25 Websites)



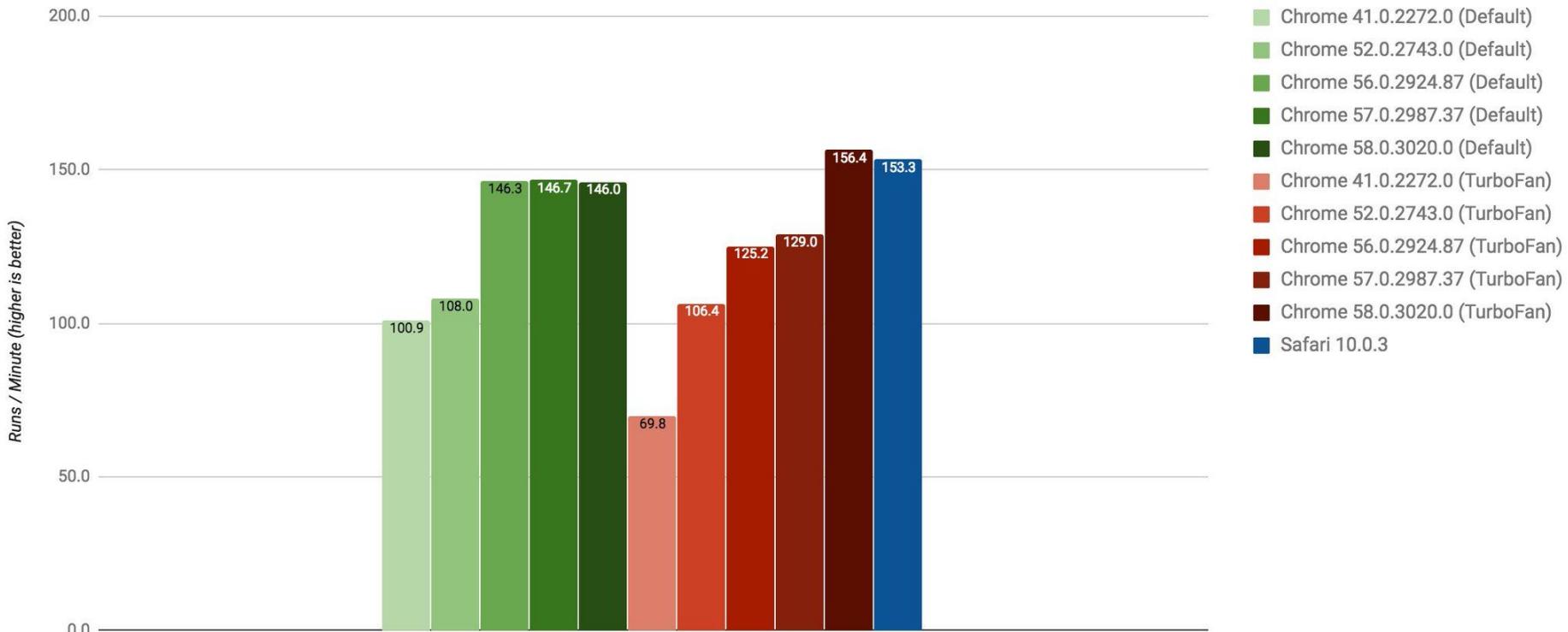
Page Load Time (LinkedIn Feed)



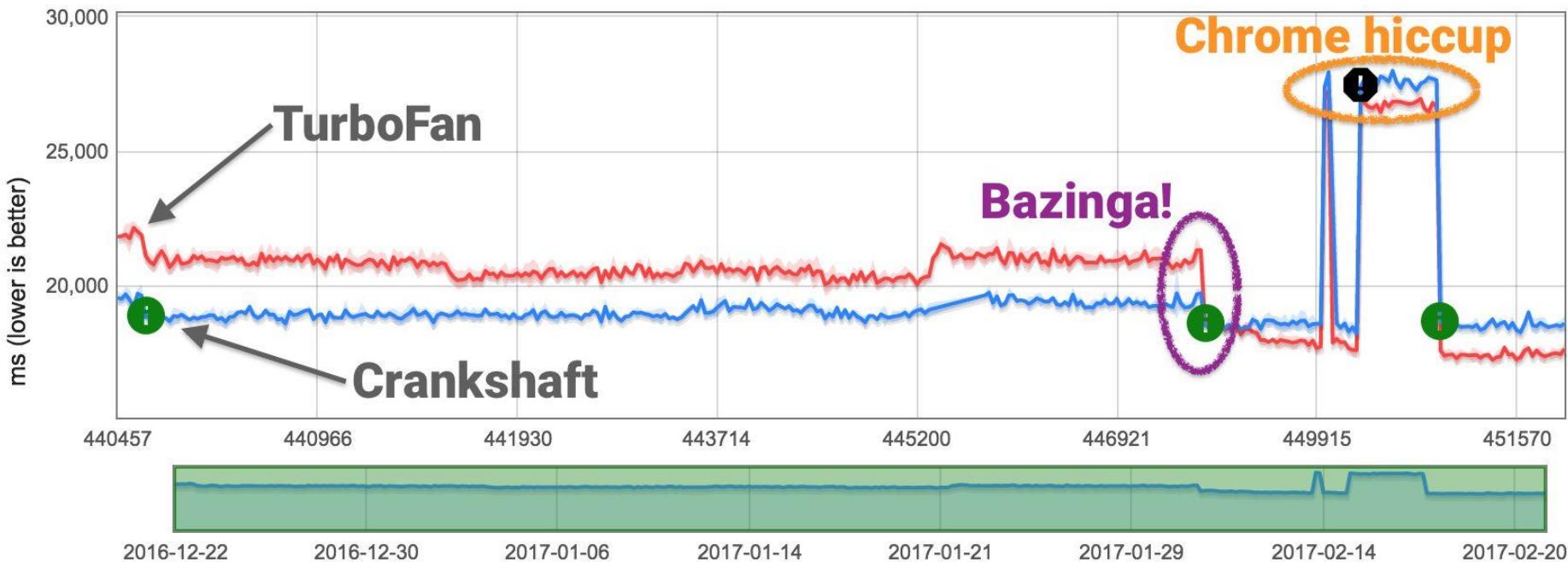
Page Load Time (Script Execution)



Framework Performance (Speedometer)



Framework Performance (Speedometer)



Framework Performance (Ember)

HTML Text

Render Complex List (HTML)

Name	Speed	Error	Samples	Mean
Render Complex List (HTML) <small>2.10.0</small>	45.93 / sec	± 31.84%	434	21.77 ms

HTML Text

Render Complex List (HTML)

Name	Speed	Error	Samples	Mean
Render Complex List (HTML) <small>2.10.0</small>	25.89 / sec	± 136.14%	267	38.63 ms

Render Complex List

Name	Speed	Error	Samples	Mean
Render Complex List <small>2.10.0</small>	49.07 / sec	± 27.19%	452	20.38 ms

Render Complex List

Name	Speed	Error	Samples	Mean
Render Complex List <small>2.10.0</small>	27.18 / sec	± 124.19%	281	36.80 ms

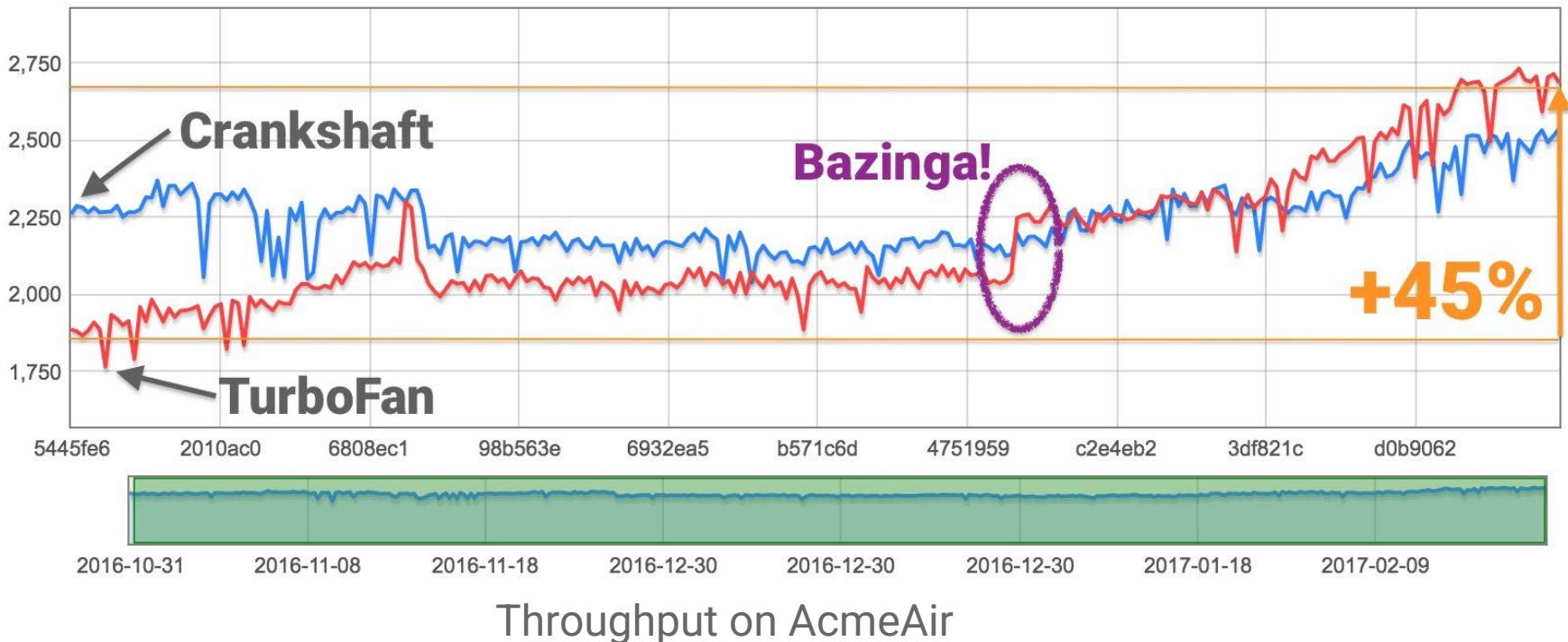


Jeff Atwood @codinghorror

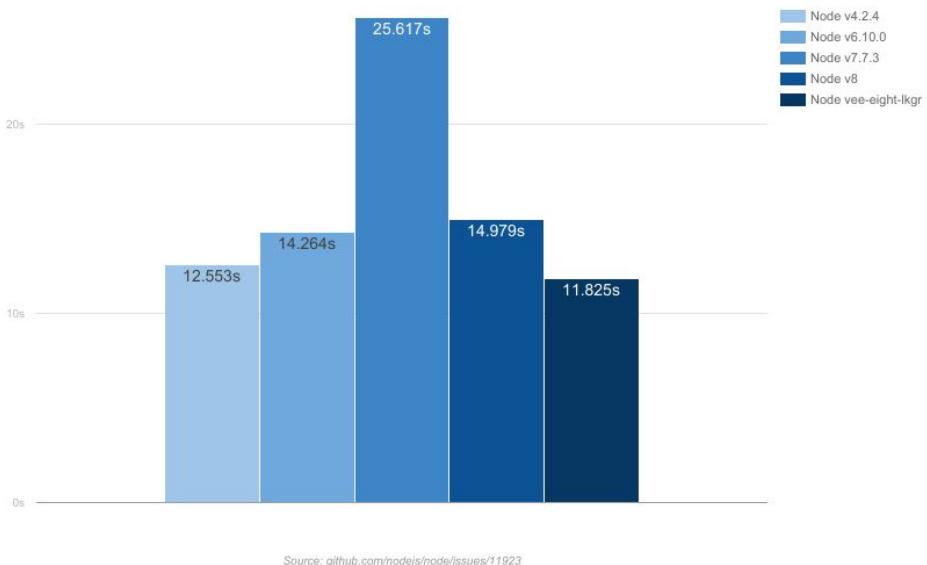
Following

Seeing some fantastic complex JS perf improvements (Discourse, Ember) in Chrome Canary!! cc @bmeurer

Node Server Workloads (AcmeAir)



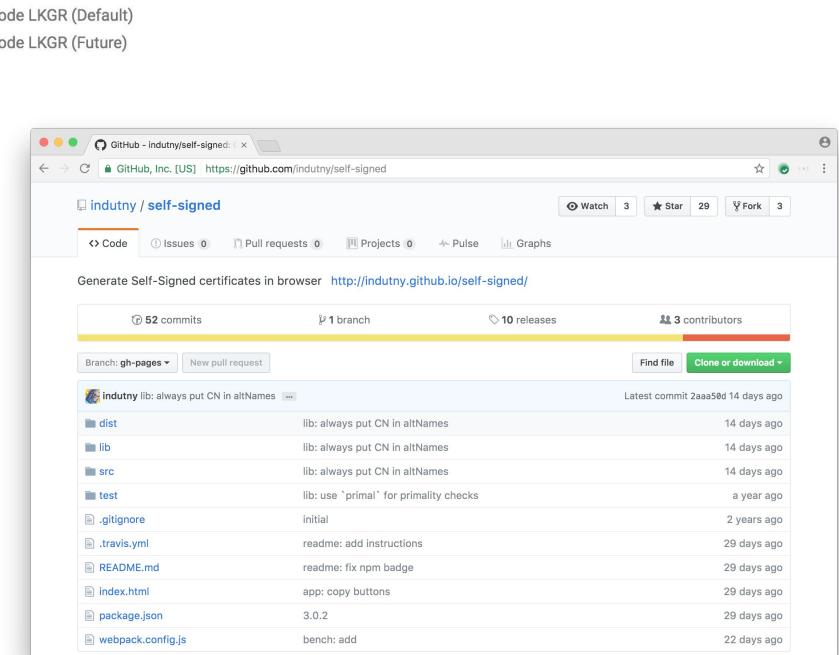
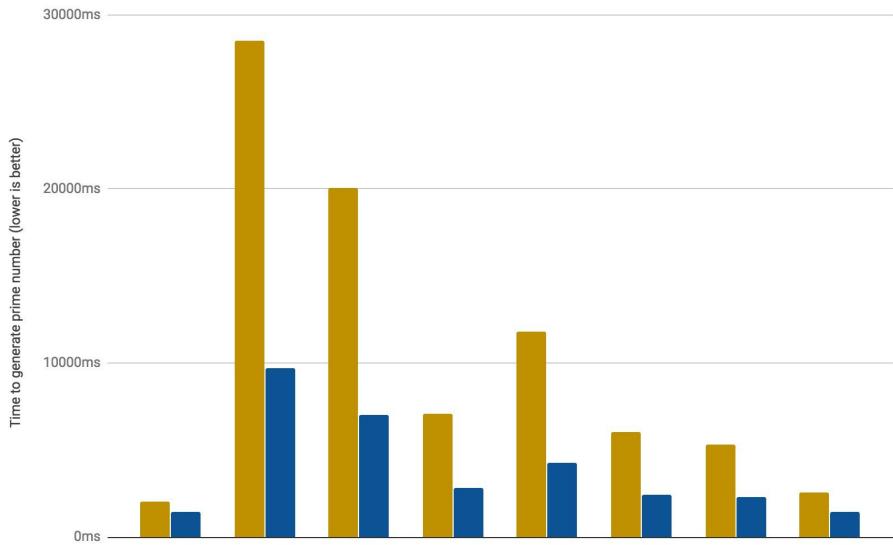
UglifyJS2



UglifyJS is a JavaScript parser, minifier, compressor or beautifier toolkit.

github.com/mishoo/UglifyJS2

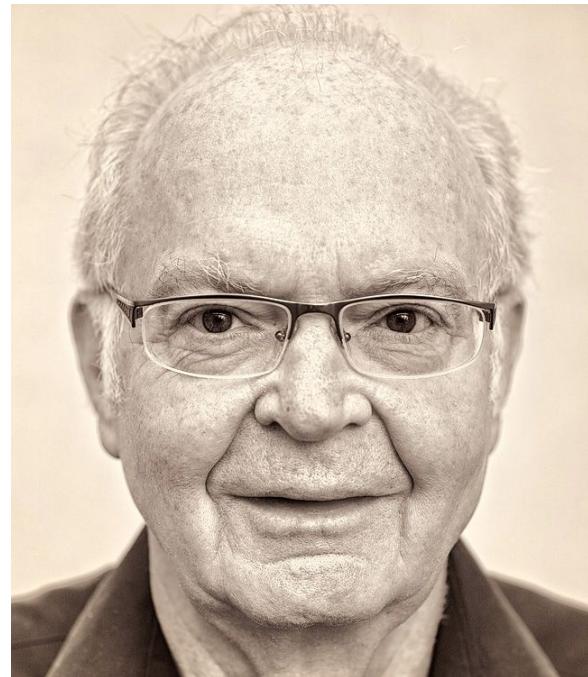
Cryptographic / Number Crunching Applications



Performance Advice

Performance Advice

“Premature
optimization is the
root of all evil”



Performance Advice

- Write idiomatic, declarative JavaScript
 - Appropriate language features
 - Handle exceptions where necessary
 - Use proper collections (Map, Set, WeakMap, WeakSet, etc.)
- Avoid (engine-specific) work-arounds
- File bug reports when something is slow

Performance Advice - Declarative JavaScript

Declarative

```
if (obj !== undefined) { return obj.x; }
```

```
....  
23 cmpq [r13-0x60],rax  
27 jz 72  
....
```

Obscure

```
if (obj) { return obj.x; }
```

```
....  
27 cmpq [r13-0x40],rax  
31 jz 128  
37 test al,0x1  
39 setzl bl  
42 movzxbl rbx,rbx  
45 cmpl rbx,0x0  
48 jnz 185  
54 cmpq [r13-0x38],rax  
58 jz 128  
64 movq rdx,[rax-0x1]  
68 testb [rdx+0xc],0x10  
72 jnz 128  
78 cmpq [r13+0x50],rdx  
82 jz 160  
....  
160 vmovsd xmm0,[rax+0x7]  
165 movq [rbp-0x18],rbx  
169 vxorpd xmm1,xmm1,xmm1  
173 vucomisd xmm1,xmm0  
177 jz 128  
179 movq rbx,[rbp-0x18]  
183 jmp 88  
185 movq [rbp-0x18],rbx  
189 cmpq rax,0x0  
193 jz 128  
195 movq rbx,[rbp-0x18]  
199 jmp 88  
....
```

~15% faster on avg!

Performance Advice - Declarative JavaScript

Declarative

```
function foo6(f, ...args) {  
  return f(...args);  
}
```

Obscure

```
function foo5(f) {  
  switch (arguments.length) {  
    case 1: return f();  
    case 2: return f(arguments[1]);  
    case 3: return f(arguments[1], arguments[2]);  
    default: {  
      var args = [];  
      for (var i = 1; i < arguments.length; ++i) {  
        args[i - 1] = arguments[i];  
      }  
      return f.apply(undefined, args);  
    }  
  }  
}
```

~4.5x faster on avg!

Thank You !

