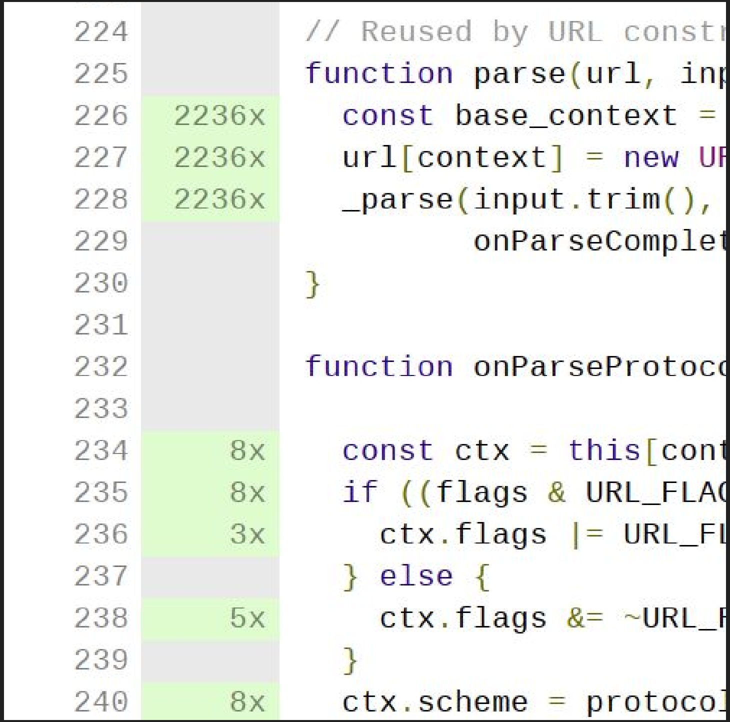
Ignition + TurboFan

A New Era for Node.js Performance

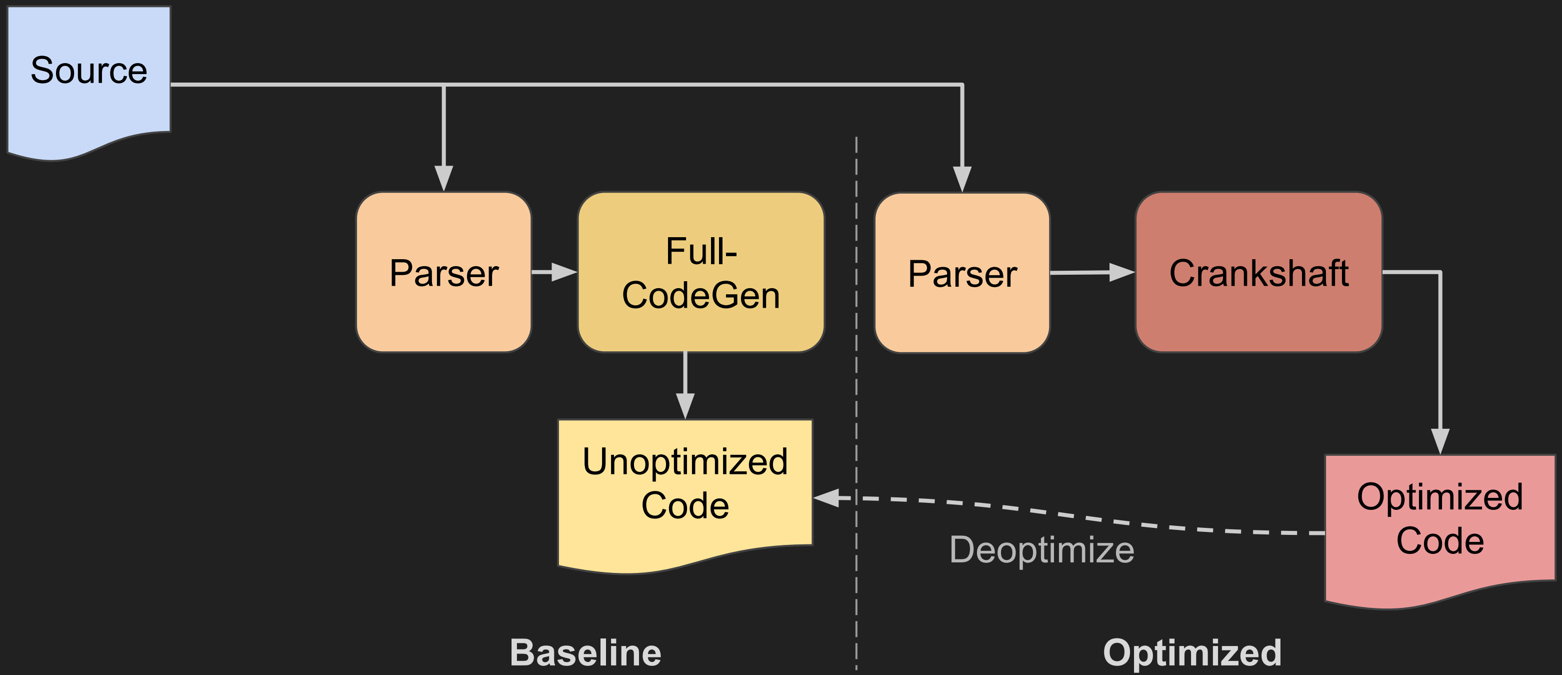
JIT compilation for JavaScript

- JIT = Just-in-Time: Source code is compiled to machine code as it is executed
- Compilation performance matters
    - Unlike Ahead-of-Time languages (C, C++, Java, Go)
- Multi-tiered compiler pipeline: only optimize hot code paths



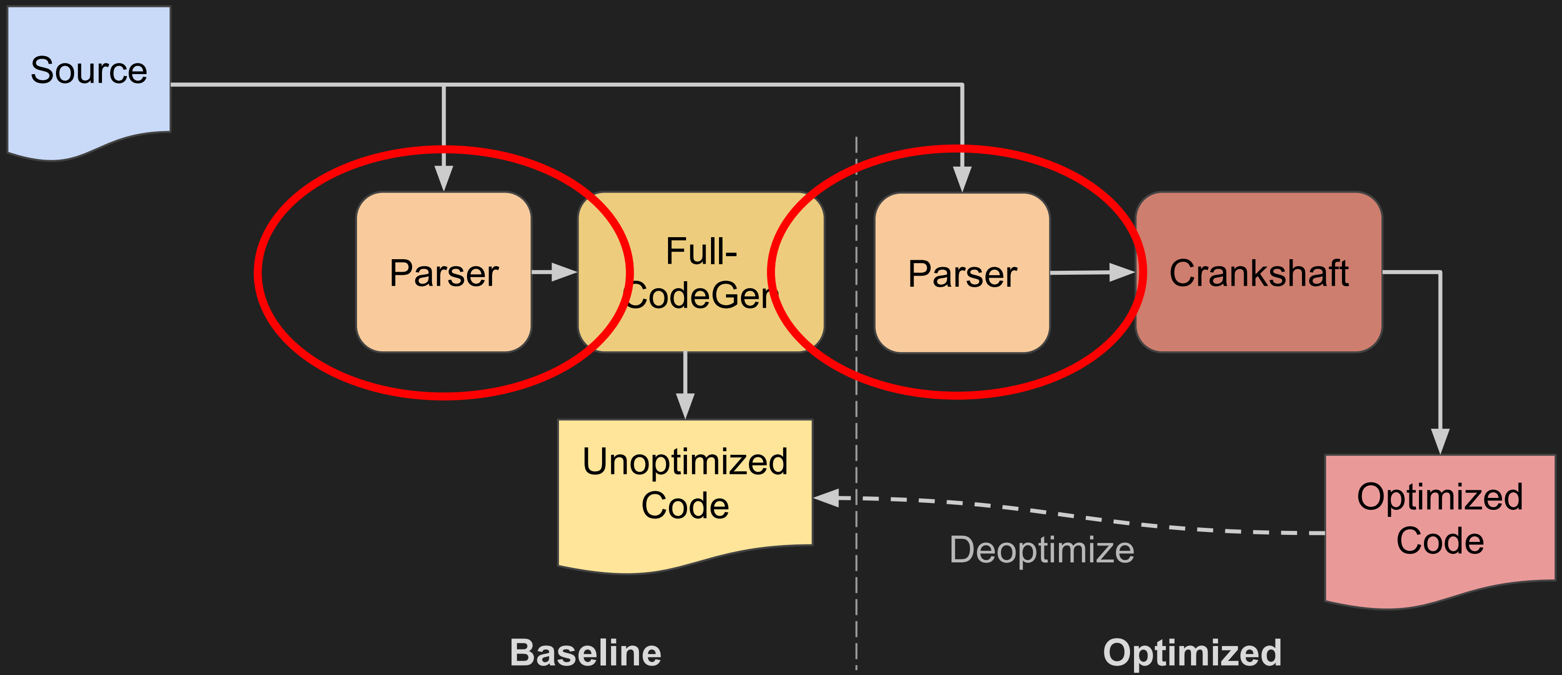
```
       // Reused by URL constr
       function parse(url, inp
2236x    const base_context = 
2236x    url[context] = new UR
2236x    _parse(input.trim(), 
                onParseComplet
       }

       function onParseProtoco

   8x    const ctx = this[cont
   8x    if ((flags & URL_FLAG
   3x      ctx.flags |= URL_FL
         } else {
   5x      ctx.flags &= ~URL_F
         }
   8x    ctx.scheme = protocol
```

V8’s original JIT engines



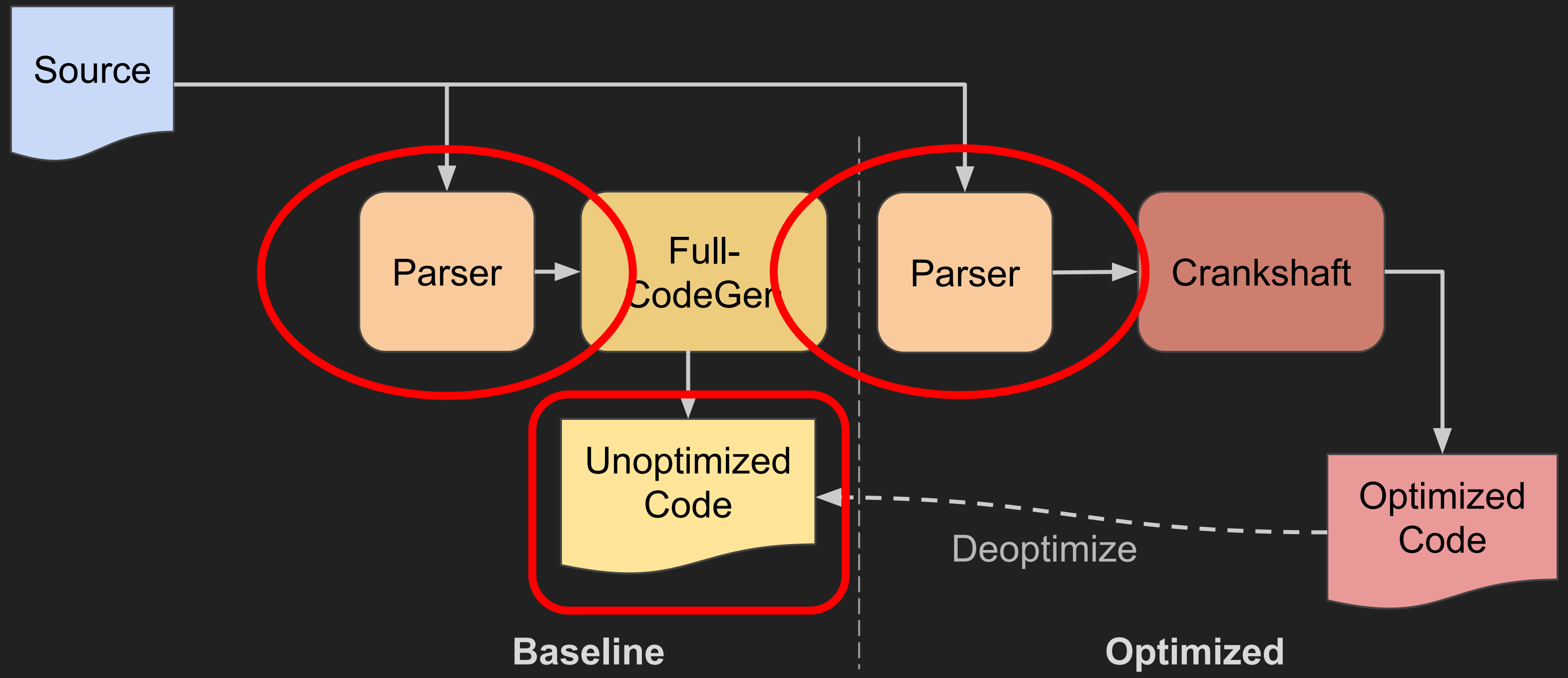
Credit: McIlroy, Ross, et al.

V8’s original JIT engines



Credit: McIlroy, Ross, et al.

V8’s original JIT engines



Credit: McIlroy, Ross, et al.

FullCodeGen

- V8’s *baseline* compiler
- All JavaScript code runs through FullCodeGen first
- JavaScript → Unoptimized machine code
- Acceptable performance, but:
- *Hugely* memory inefficient

```
(a, b) => a + b + 100
```

```
  0  55              push rbp
  1  4889e5          REX.W movq rbp,rsp
  4  56              push rsi
  5  57              push rdi
  6  493ba5880a0000  REX.W cmpq rsp,[r13+0xa88]
 13  7305            jnc 20
 15  e84c11efff      call StackCheck
 20  ff7518          push [rbp+0x18]
 23  488b4510        REX.W movq rax,[rbp+0x10]
 27  5a              pop rdx
 28  e8ff50edff      call 0x309dd0c20b80
 33  90              nop
 34  50              push rax
 35  48b80000000064000000 REX.W movq rax,0x6400000000
 45  5a              pop rdx
 46  e8ed50edff      call 0x309dd0c20b80
 51  90              nop
 52  48bbf9c3760003340000 REX.W movq rbx,0x34030076c3f9
 62  83430bd1        addl [rbx+0xb],0xd1
 66  791f            jns 99
 68  50              push rax
 69  e89610efff      call InterruptCheck
 74  58              pop rax
 75  48bbf9c3760003340000 REX.W movq rbx,0x34030076c3f9
 85  49ba0000000000180000 REX.W movq r10,0x180000000000
 95  4c895307        REX.W movq [rbx+0x7],r10
 99  c9              leave
100  c21800          ret 0x18
103  498b45a8        REX.W movq rax,[r13-0x58]
107  e9c4ffffff      jmp 52
```

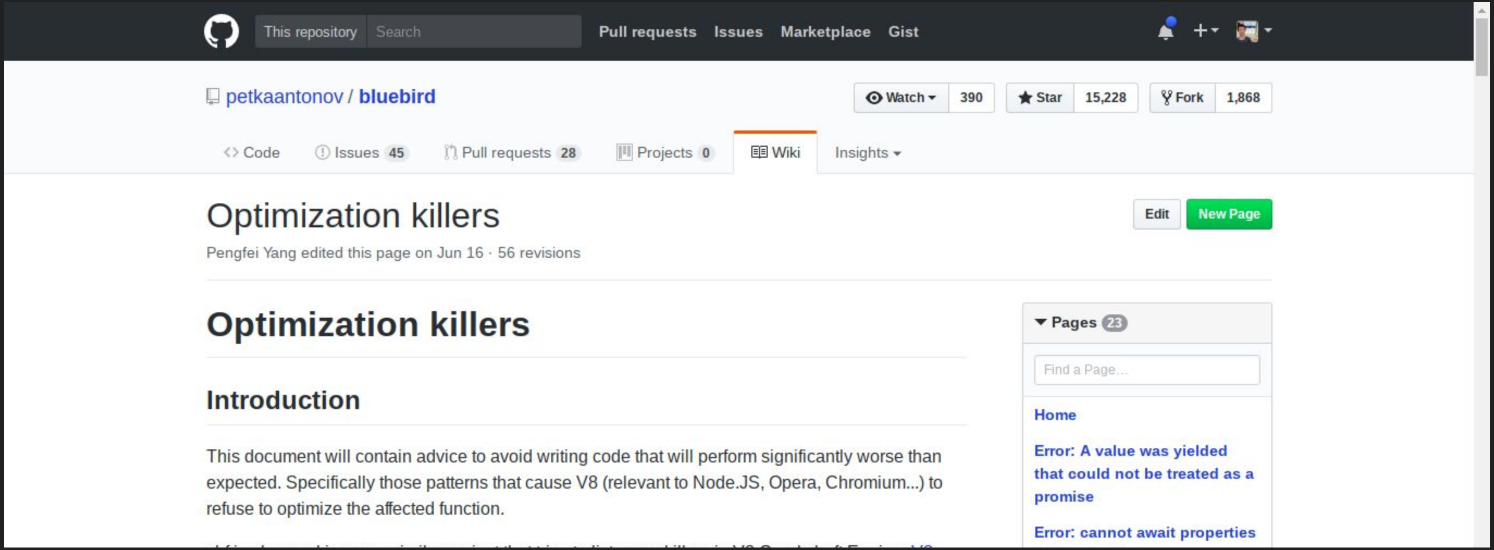
```
  0  55               push rbp
  1  4889e5           REX.W movq rbp,rsp
  4  56               push rsi
  5  57               push rdi
  6  493ba5880a0000   REX.W cmpq rsp,[r13+0xa88]
 13  7305             jnc 20
 15  e84c11efff       call StackCheck
 20  ff7518           push [rbp+0x18]
 23  488b4510         REX.W movq rax,[rbp+0x10]
 27  5a               pop rdx
 28  e8ff50edff       call 0x309dd0c20b80
```

14 bytes $^{\text{(minified)}}$ $\rightarrow$ 116 bytes

```
 52  48bbf9c3760003340000 REX.W movq rbx,0x34030076c3f9
 62  83430bd1         addl [rbx+0xb],0xd1
 66  791f             jns 99
 68  50               push rax
 69  e89610efff       call InterruptCheck
 74  58               pop rax
 75  48bbf9c3760003340000 REX.W movq rbx,0x34030076c3f9
 85  49ba0000000000180000 REX.W movq r10,0x180000000000
 95  4c895307         REX.W movq [rbx+0x7],r10
 99  c9               leave
100  c21800           ret 0x18
103  498b45a8         REX.W movq rax,[r13-0x58]
107  e9c4ffffff       jmp 52
```

Crankshaft: “Optimization Killers”

- Crankshaft emits optimized machine code for hot functions
- Support for certain JS features was never added to Crankshaft
- Unpredictable performance



Optimization killers

Pengfei Yang edited this page on Jun 16 · 56 revisions

Optimization killers

Introduction

This document will contain advice to avoid writing code that will perform significantly worse than expected. Specifically those patterns that cause V8 (relevant to Node.JS, Opera, Chromium...) to refuse to optimize the affected function.

Rise of “CrankshaftScript” and microoptimizations

- Unpredictable performance leads to JS code tuned specific for Crankshaft
- `.forEach()` → `for` loop
- `isNaN(a)` → `a !== a`
- `a === ''` → `a.length === 0`
- `arguments` copying
- Isolated functions for `try {} catch (err) {}`

Crankshaft: Rigid structure

- Crankshaft was never designed for extension
- Difficult to implement ES6+ features on top of Crankshaft
  - E.g. generator functions, async functions
- Performance-sensitive code forced to stay on ES5
  - E.g. `const` → `var`

EventEmitter.prototype.emit

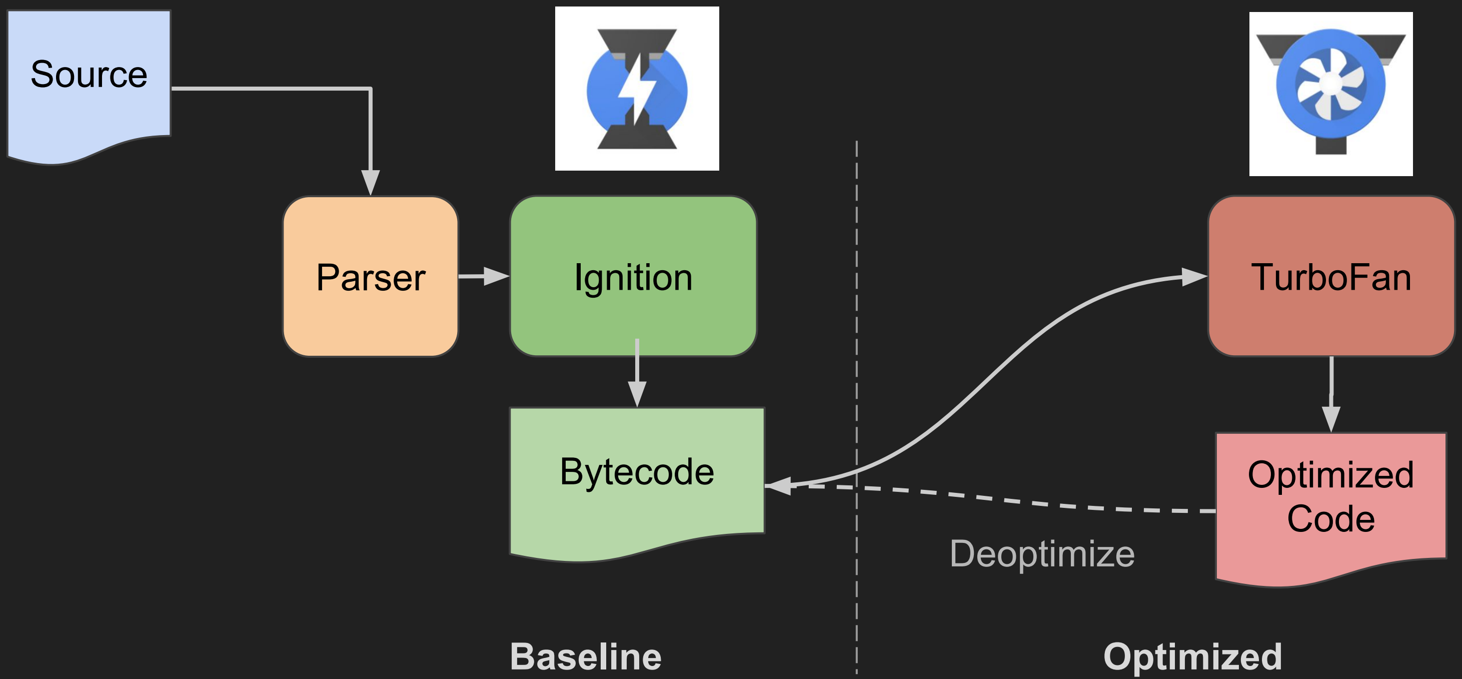
```
  var isFn = typeof handler === 'function';
  len = arguments.length;
  switch (len) {
    // fast cases
    case 1:
      emitNone(handler, isFn, this);
      break;
    case 2:
      emitOne(handler, isFn, this, arguments[1]);
      break;
    case 3:
      emitTwo(handler, isFn, this, arguments[1], arguments[2]);
      break;
    case 4:
      emitThree(handler, isFn, this, arguments[1], arguments[2], arguments[3]);
      break;
    // slower
    default:
      args = new Array(len - 1);
      for (i = 1; i < len; i++)
        args[i - 1] = arguments[i];
      emitMany(handler, isFn, this, args);
  }
```

EventEmitter.prototype.emit

```
  var isFn = typeof handler === 'function';
```

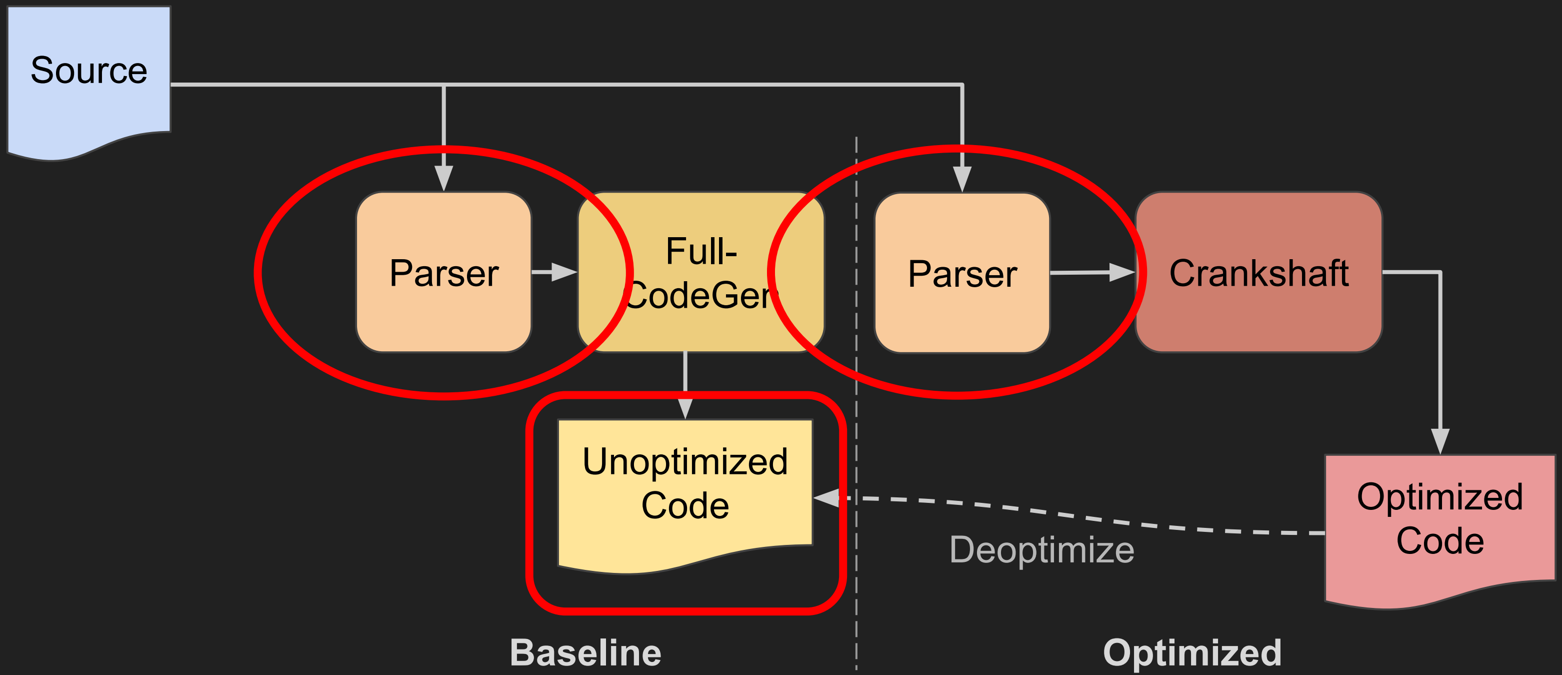
```
      emitMany(handler, isFn, this, args);
```

V8’s answer: Ignition + TurboFan



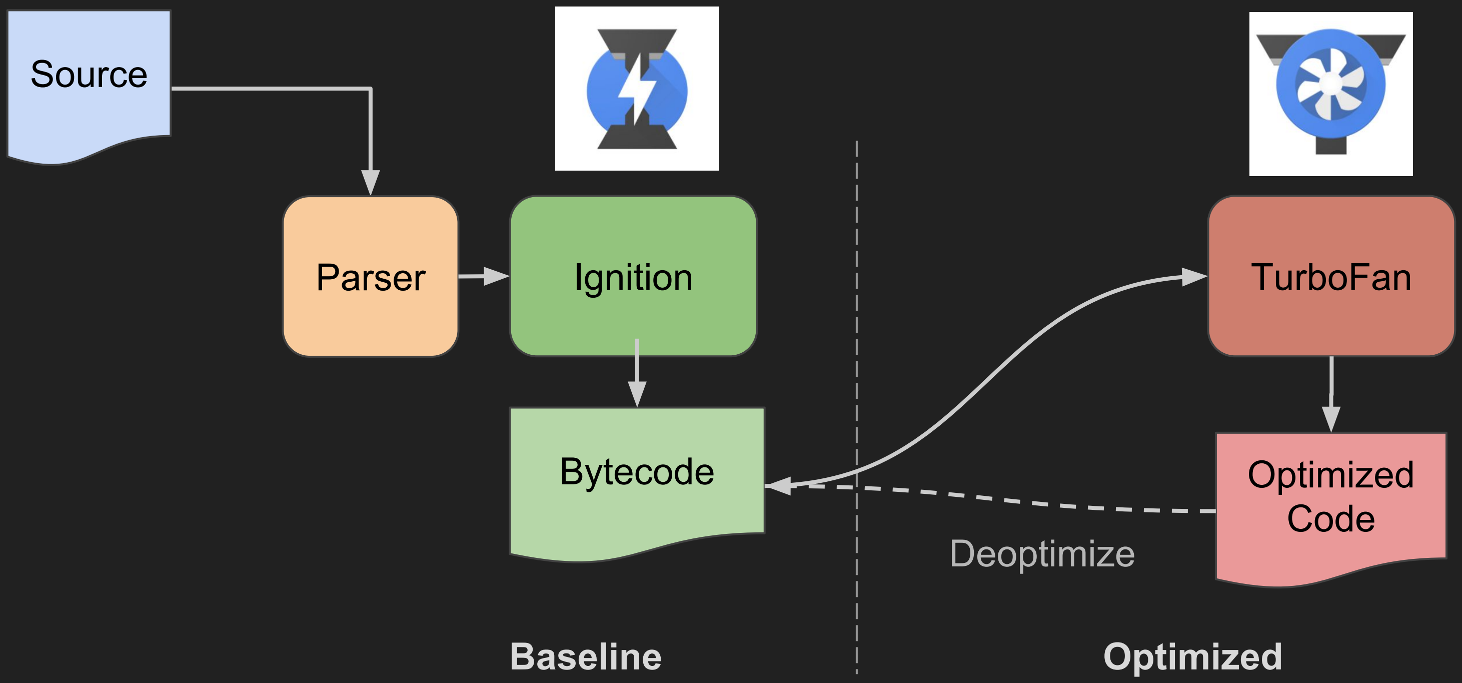
Credit: McIlroy, Ross, et al.

V8's original JIT engines



Credit: McIlroy, Ross, et al.

Ignition + TurboFan



Credit: McIlroy, Ross, et al.

Ignition: Memory-conserving interpreter

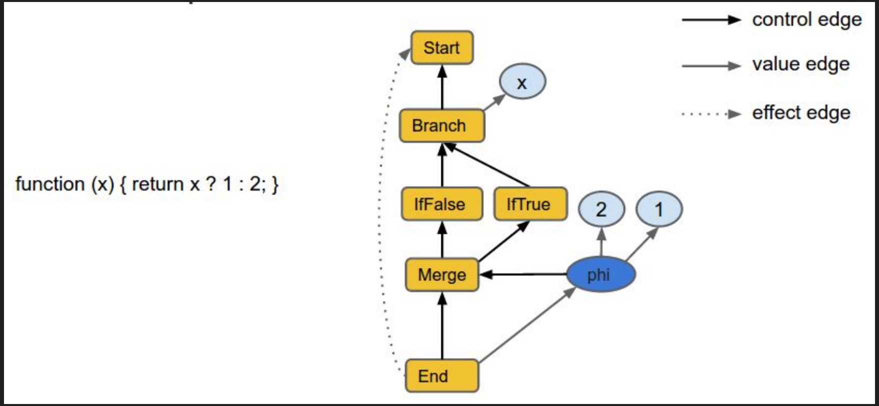
```
(a, b) =>
  a + b + 100
```

```
0 : 93          StackCheck
1 : 1e 02       Ldar a1
3 : 2c 03 03    Add a0, [3]
6 : 37 64 04    AddSmi [100], [4]
9 : 97          Return
```

14 bytes (minified) $\rightarrow$ 10 bytes

TurboFan: Advanced optimizing compiler

- Graph-based optimizations



- Layered architecture: More flexibility for new features
- Optimizes *everything*

TurboFan: No more “CrankshaftScript”

- `.forEach()` → `for` loop
- `isNaN(a)` → `a !== a`
- `a === ''` → `a.length === 0`
- `arguments` copying
- `const` → `var`
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

- `.forEach()` → `for` loop ✗ `.forEach()` will soon be inlined
- `isNaN(a)` → `a !== a`
- `a === ''` → `a.length === 0`
- `arguments` copying
- `const` → `var`
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

- `.forEach()` → for loop ✗ `.forEach()` will soon be inlined
- `isNaN(a)` → `a !== a` ✗ `isNaN()` is now inlined
- `a === ''` → `a.length === 0`
- `arguments` copying
- `const` → `var`
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

- `.forEach()` → for loop ✗ `.forEach()` will soon be inlined
- `isNaN(a)` → `a !== a` ✗ `isNaN()` is now inlined
- `a === ''` → `a.length === 0` ✗ Performance is now identical
- `arguments` copying
- `const` → `var`
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

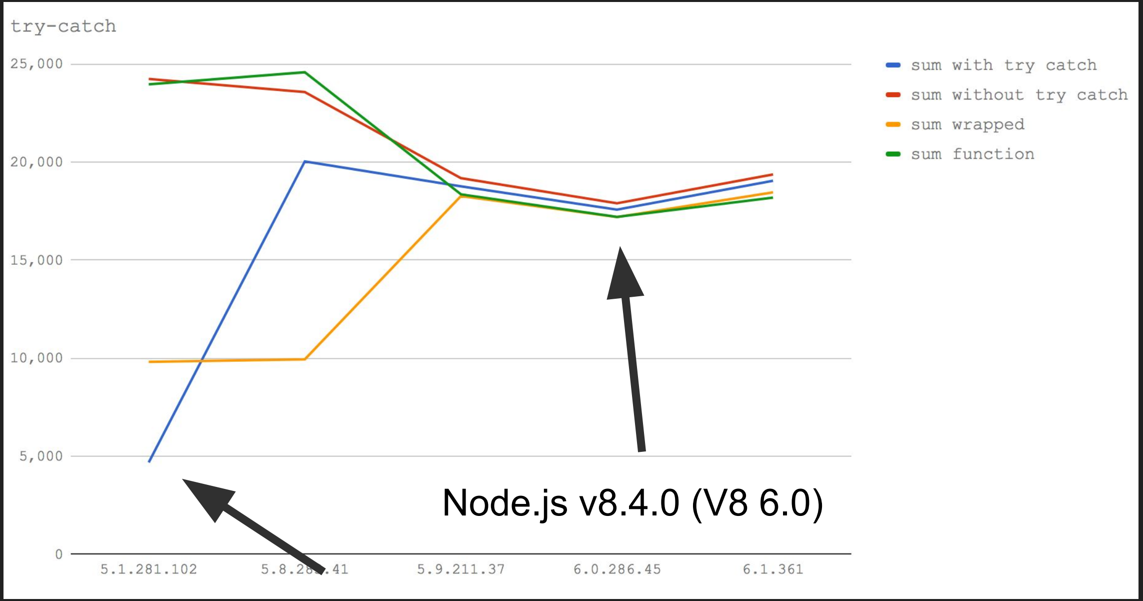
- `.forEach()` → `for` loop ✗ `.forEach()` will soon be inlined
- `isNaN(a)` → `a !== a` ✗ `isNaN()` is now inlined
- `a === ''` → `a.length === 0` ✗ Performance is now identical
- `arguments` copying ✗ `(...args)` is always faster than copying; `arguments` can also be used directly
- `const` → `var`
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

- `.forEach()` → `for` loop ✗ `.forEach()` will soon be inlined
- `isNaN(a)` → `a !== a` ✗ `isNaN()` is now inlined
- `a === ''` → `a.length === 0` ✗ Performance is now identical
- `arguments` copying ✗ `(...args)` is always faster than copying; `arguments` can also be used directly
- `const` → `var` ✗ `const` is faster than `var` once warmed up
- Isolated functions for `try {} catch (err) {}`

TurboFan: No more “CrankshaftScript”

- Isolated functions for `try {} catch (err) {}` ✗

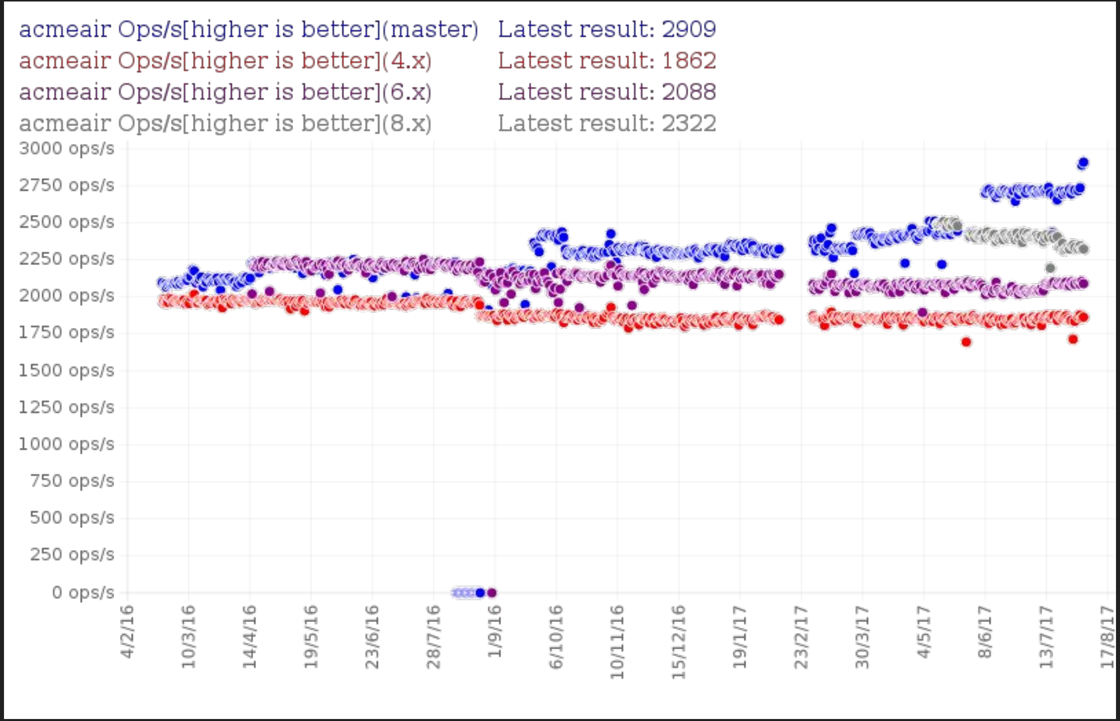


Credit: David Mark Clements & Matteo Collina

Node.js v6.11.2 (V8 5.1)

TurboFan: Discouraging *Microbenchmarks*

- Focus on **real-world performance** (performance of an entire application)



Write readable code.

Don’t concern yourself with microoptimizations – that’s V8’s job.