JAVASCRIPT PERFORMANCE THROUGH THE SPYGLASS

VYACHESLAV EGOROV

Excelsior JET

V8

Dart VM

LuaJIT

Excelsior JET

V8

Dart VM

LuaJIT

COMPLEXITY

Why is let slower than var in a for loop in nodejs?

16

3

I have written a very simple benchmark:

```
console.time('var');
for (var i = 0; i < 100000000; i++) {}
console.timeEnd('var')


console.time('let');
for (let i = 0; i < 100000000; i++) {}
console.timeEnd('let')
```

If you're running Chrome, you can try it here (since NodeJS and Chrome use the same JavaScript engine, albeit usually slightly different versions):

▶ Show code snippet

And the results amaze me:

```
var: 89.162ms
let: 320.473ms
```

I have tested it in Node 4.0.0 && 5.0.0 && 6.0.0 and the proportion between `var` and `let` is the same for each node version.

Could someone please explain to me what is the reason behid this seemingly odd behaviour?

javascript node.js ecmascript-6 v8

share edit close flag

edited 16 hours ago

T.J. Crowder

450k ● 71 ● 702 ● 843

asked 17 hours ago

Jan Osch

194 ● 11

no choice

[ask on StackOverflow]

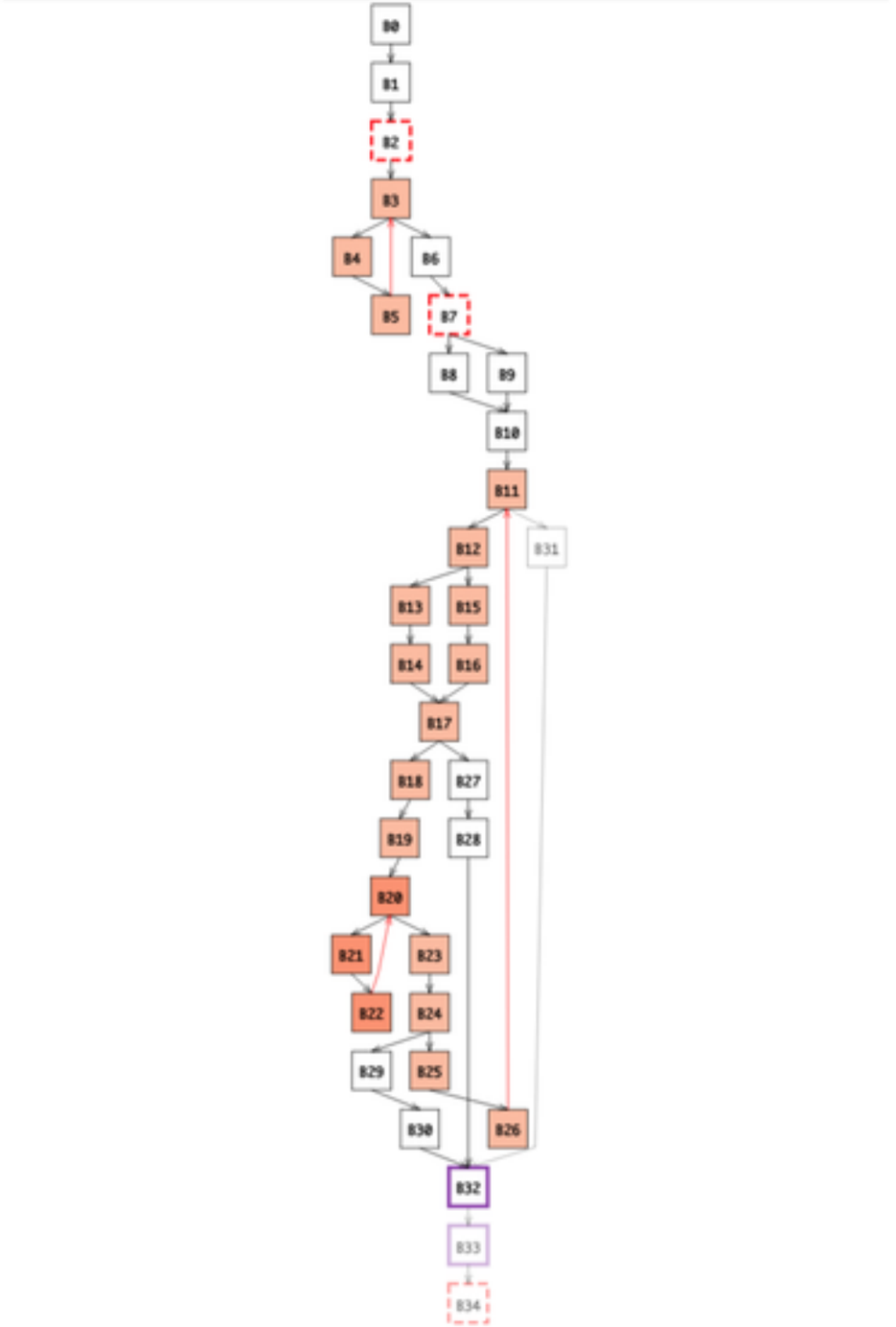
profile

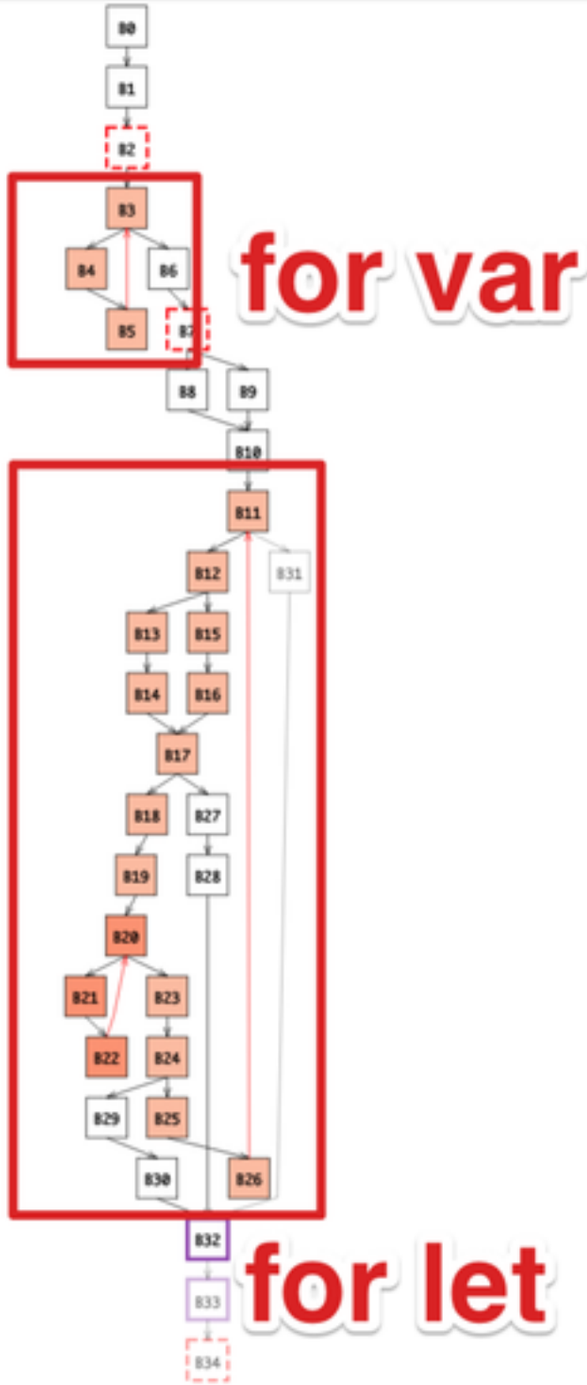
and/or

read code

Crankshaft ⇒ IRHydra

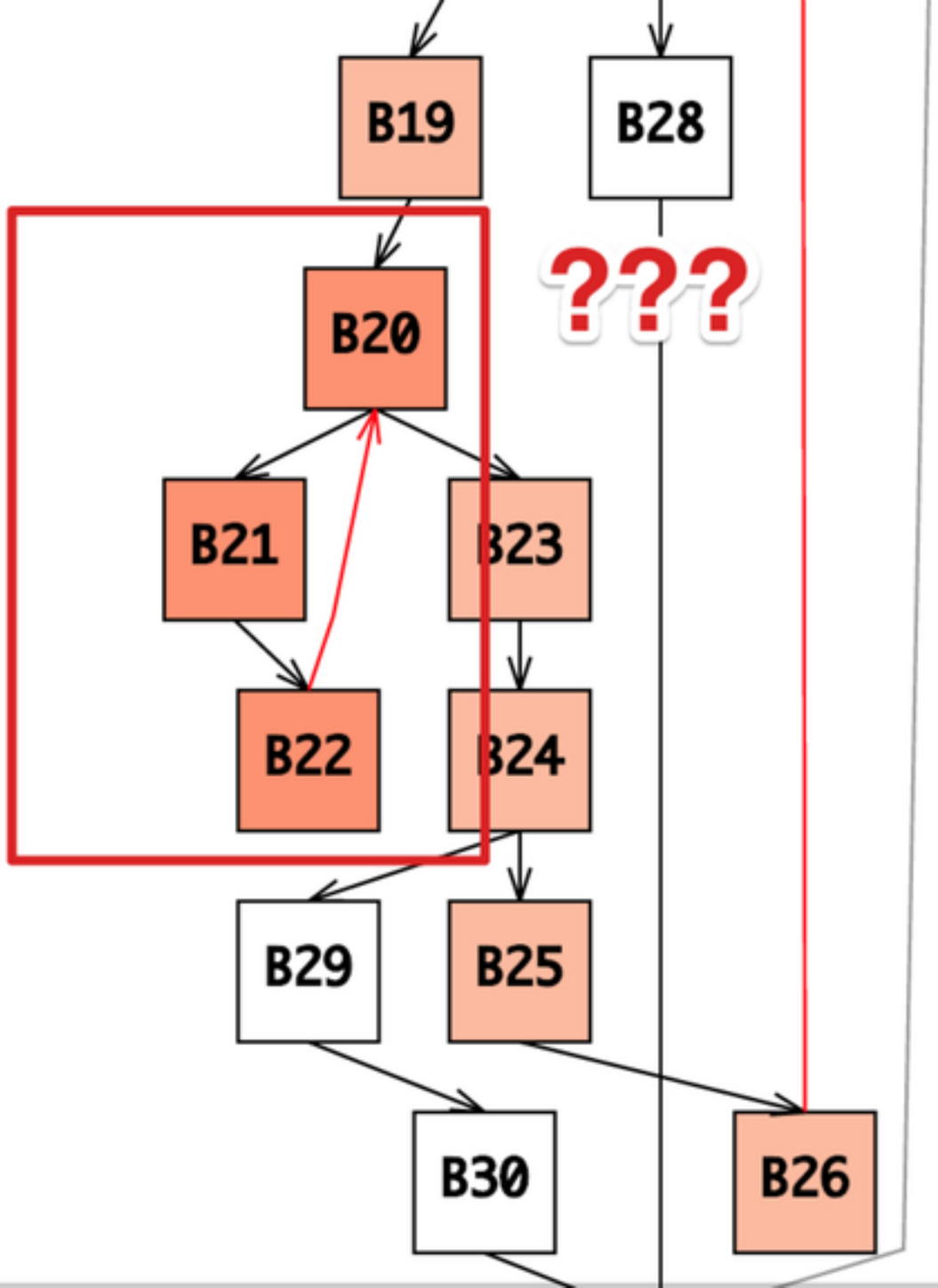
[http://mrale.ph/irhydra/2]


B0
B1
B2
B3
B4
B6
B5
B7
B8
B9
B10
B11
B12
B31
B13
B15
B14
B16
B17
B18
B27
B19
B28
B20
B21
B23
B22
B24
B29
B25
B30
B26
B32
B33
B34



for var

for let


B19
B28
???
B20
B21
B23
B22
B24
B29
B25
B30
B26

```
# Get the source as described at:
# https://chromium.googlesource.com/v8/v8.git
$ fetch v8
...
$ cd v8
$ make x64.debug
...
```

```
# Get the source as described at:
# https://chromium.googlesource.com/v8/v8.git
$ fetch v8
...
$ cd v8
$ make x64.debug
...
$ out/x64.debug/d8 --print-ast test.js
```

```
. . FOR at -1
. . . YIELD COUNT 0
. . . BODY at -1
. . . . BLOCK at -1
. . . . . BLOCK NOCOMPLETIONS at -1
. . . . . . EXPRESSION STATEMENT at -1
. . . . . . . INIT at -1
. . . . . . . . VAR PROXY local[0] (mode = LET) "i"
. . . . . . . . VAR PROXY local[2] (mode = TEMPORARY) ".for"
. . . . . . IF at -1
. . . . . . . CONDITION at -1
. . . . . . . . EQ at -1
. . . . . . . . . VAR PROXY local[3] (mode = TEMPORARY) ".for"
. . . . . . . . . LITERAL 1
. . . . . . . THEN at -1
. . . . . . . . EXPRESSION STATEMENT at -1
. . . . . . . . . ASSIGN at -1
. . . . . . . . . . VAR PROXY local[3] (mode = TEMPORARY) ".for"
. . . . . . . . . . LITERAL 0
. . . . . . . ELSE at 200
. . . . . . . . EXPRESSION STATEMENT at 200
. . . . . . . . . POST INC at 200
. . . . . . . . . . VAR PROXY local[0] (mode = LET) "i"
. . . . . . EXPRESSION STATEMENT at -1
. . . . . . . ASSIGN at -1
. . . . . . . . VAR PROXY local[4] (mode = TEMPORARY) ".for"
. . . . . . . . LITERAL 1
. . . . . . IF at 186
. . . . . . . CONDITION at 186
. . . . . . . . LT at 186
. . . . . . . . . VAR PROXY local[0] (mode = LET) "i"
. . . . . . . . . LITERAL 1e+08
. . . . . . . THEN at -1
. . . . . . . . EMPTY at -1
. . . . . . . ELSE at -1
. . . . . . . . BREAK at -1
. . . . . FOR at 168
. . . . . . YIELD COUNT 0
```

```
VAR PROXY local[0] (mode = LET) "i"
VAR PROXY local[2] (mode = TEMPORARY) ".for"
```

```
local[0] (mode = LET) "i"
local[2] (mode = TEMPORARY) ".for"
```

```
$ git grep '"\.for"'
src/ast/ast-value-factory.h:
      F(dot_for, ".for")
```

```
$ git grep dot_for
src/parsing/parser.cc:  const AstRawString* temp
src/parsing/parser.cc:                  scope_->NewTem
src/parsing/parser.cc:                scope_->NewTempo
```

```
// We are given a for statement
// of the form
//
//  labels: for (let/const x = i;
//                    cond;
//                    next) body
//
// and rewrite it as follows.
```

```
// We are given a for statement of the form
//
//  labels: for (let/const x = i; cond; next) body
//
// and rewrite it as follows.  Here we write {{ ... }} for init-blocks, ie.,
// blocks whose ignore_completion_value_ flag is set.
//
//  {
//    let/const x = i;
//    temp_x = x;
//    first = 1;
//    undefined;
//    outer: for (;;) {
//      let/const x = temp_x;
//      {{ if (first == 1) {
//           first = 0;
//         } else {
//           next;
//         }
//         flag = 1;
//         if (!cond) break;
//      }}
//      labels: for (; flag == 1; flag = 0, temp_x = x) {
//        body
//      }
//      {{ if (flag == 1)  // Body used break.
//           break;
//      }}
//    }
//  }
```

internals are important

internals are irrelevant

```
function find(val){
  function index(value) {
    return [1, 2, 3].indexOf(value);
  }

  return index(val);
}
```

```
function find(val){
  function index(value) {
    return [1, 2, 3].indexOf(value);
  }

  return index(val);
}
// => 5464 op/ms
```

```
var arr = [1, 2, 3];
function find(val){
  function index(value) {
    return arr.indexOf(value);
  }

  return index(val);
}
```

```
var arr = [1, 2, 3];
function find(val){
  function index(value) {
    return arr.indexOf(value);
  }

  return index(val);
}
// => 14084 op/ms
```

«ARRAY ALLOCATION IS EXPENSIVE!»

```
var makeArr = () => [1, 2, 3];
function find(val){
  function index(value) {
    return makeArr().indexOf(value);
  }

  return index(val);
}
```

```
var makeArr = () => [1, 2, 3];
function find(val){
  function index(value) {
    return makeArr().indexOf(value);
  }

  return index(val);
}
// => 12048 op/ms
```

```
var makeArr = () => [1, 2, 3];
function find(val){
  function index(value) {
    return makeArr().indexOf(value);
  }

  return index(val);
}
// => 12048 op/ms
```

```
$ perf record d8 --perf-basic-prof test.js
$ perf report
```

```
14.80% v8::internal::Factory::NewFunction
 9.98% v8::internal::Factory::NewFunctionFromSha
 8.74% *InnerArrayIndexOf native array.js:1019
 7.24% v8::internal::Factory::NewFunctionFromSha
 5.79% v8::internal::Runtime_NewClosure
 4.84% Builtin:ArgumentsAdaptorTrampoline
 4.44% v8::internal::Heap::AllocateRaw
 4.30% v8::internal::Factory::New
 3.74% v8::internal::SharedFunctionInfo::SearchO
 3.20% ~index test.js:2
```

```
14.80% v8::internal::Factory::NewFunction
 9.98% v8::internal::Factory::NewFunctionFromSha
 8.74% *InnerArrayIndexOf native array.js:1019
 7.24% v8::internal::Factory::NewFunctionFromSha
 5.79% v8::internal::Runtime_NewClosure
 4.84% Builtin:ArgumentsAdaptorTrampoline
 4.44% v8::internal::Heap::AllocateRaw
 4.30% v8::internal::Factory::New<v8::internal::
 3.74% v8::internal::SharedFunctionInfo::SearchO
 3.20% ~index test.js:2
```

```
14.49% *InnerArrayIndexOf native array.js:1019
10.47% LoadIC:A load IC from the snapshot
 8.45% ~index b2.js:12
 8.05% Stub:FastNewClosureStub
 5.77% Builtin:ArgumentsAdaptorTrampoline
 5.23% *find2 b2.js:11
```

```
14.49% *InnerArrayIndexOf native array.js:1019
10.47% LoadIC:A load IC from the snapshot
 8.45% ~index b2.js:12
 8.05% Stub:FastNewClosureStub
 5.77% Builtin:ArgumentsAdaptorTrampoline
 5.23% *find2 b2.js:11
```

`FastNewClosureStub`
did not support allocation of
functions with literals inside

`FastNewClosureStub` did not support allocation of functions with literals inside

[now it actually can, example is from February 2016]

Strange JavaScript performance

6

2

When I was implementing ChaCha20 in JavaScript, I stumbled upon some strange behavior.

My first version was build like this (let's call it "Encapsulated Version"):

```
function quarterRound(x, a, b, c, d) {
    x[a] += x[b]; x[d] = ((x[d] ^ x[a]) << 16) | ((x[d] ^ x[a]) >>> 16);
    x[c] += x[d]; x[b] = ((x[b] ^ x[c]) << 12) | ((x[b] ^ x[c]) >>> 20);
    x[a] += x[b]; x[d] = ((x[d] ^ x[a]) <<  8) | ((x[d] ^ x[a]) >>> 24);
    x[c] += x[d]; x[b] = ((x[b] ^ x[c]) <<  7) | ((x[b] ^ x[c]) >>> 25);
}

function getBlock(buffer) {
    var x = new Uint32Array(16);

    for (var i = 16; i--;) x[i] = input[i];
    for (var i = 20; i > 0; i -= 2) {
        quarterRound(x, 0, 4, 8,12);
        quarterRound(x, 1, 5, 9,13);
        quarterRound(x, 2, 6,10,14);
        quarterRound(x, 3, 7,11,15);
        quarterRound(x, 0, 5,10,15);
        quarterRound(x, 1, 6,11,12);
        quarterRound(x, 2, 7, 8,13);
        quarterRound(x, 3, 4, 9,14);
    }
    for (i = 16; i--;) x[i] += input[i];
    for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]);
    input[12]++;
    return buffer;
}
```

To reduce unnecessary function calls (with parameter overhead etc.) I removed the `quarterRound` -function and put it's contents inline (it's correct; I verified it against some test vectors):

```
function getBlock(buffer) {
    var x = new Uint32Array(16);
```

ChaCha20

```
function getBlock(buffer) {
  var x = new Uint32Array(16);

  for (var i = 16; i--;) x[i] = input[i];
  for (var i = 20; i > 0; i -= 2) {
    quarterRound(x, 0, 4, 8,12);
    quarterRound(x, 1, 5, 9,13);
    quarterRound(x, 2, 6,10,14);
    quarterRound(x, 3, 7,11,15);
    quarterRound(x, 0, 5,10,15);
    quarterRound(x, 1, 6,11,12);
    quarterRound(x, 2, 7, 8,13);
    quarterRound(x, 3, 4, 9,14);
  }
  for (i = 16; i--;) x[i] += input[i];
  for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]);
  input[12]++;
  return buffer;
}
```

19MB/s

```
function getBlock(buffer) {
  var x = new Uint32Array(16);

  for (var i = 16; i--;) x[i] = input[i];
  for (var i = 20; i > 0; i -= 2) {
    quarterRound(x, 0, 4, 8,12);
    quarterRound(x, 1, 5, 9,13);
    quarterRound(x, 2, 6,10,14);
    quarterRound(x, 3, 7,11,15);
    quarterRound(x, 0, 5,10,15);
    quarterRound(x, 1, 6,11,12);
    quarterRound(x, 2, 7, 8,13);
    quarterRound(x, 3, 4, 9,14);
  }
  for (i = 16; i--;) x[i] += input[i];
  for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]);
  input[12]++;
  return buffer;
}
```

```
function getBlock(buffer) {
  var x = new Uint32Array(16);

  for (var i = 16; i--;) x[i] = input[i];
  for (var i = 20; i > 0; i -= 2) {
    // quarterRound(x, 0, 4, 8,12);
    x[ 0] += x[ 4]; x[12] = ((x[12] ^ x[ 0]) << 16) | ((
    x[ 8] += x[12]; x[ 4] = ((x[ 4] ^ x[ 8]) << 12) | ((
    x[ 0] += x[ 4]; x[12] = ((x[12] ^ x[ 0]) <<  8) | ((
    x[ 8] += x[12]; x[ 4] = ((x[ 4] ^ x[ 8]) <<  7) | ((
    // ... so on ...
  }
  for (i = 16; i--;) x[i] += input[i];
  for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]);
  input[12]++;
  return buffer;
}
```

2MB/s

«I HEARD INLINE MUST IMPROVE PERFORMANCE!»

```
function U32TO8_LE(x, i, u) {
    x[i]   = u; u >>>= 8;
    x[i+1] = u; u >>>= 8;
    x[i+2] = u; u >>>= 8;
    x[i+3] = u;
}
```

```
function U32T08_LE(x, i, u) {
    //
    //  `MM'      `M' 6MMMMb\ `MMMMMMMMM              dM.      6MMMMb\ `MMb     dMM'
    //   MM        M 6M'    `  MM       \            ,MMb     6M'    `  MMM.   ,PMM
    //   MM        M MM        MM                    d'YM.    MM        M`Mb   d'MM
    //   MM        M YM.       MM     ,             ,P `Mb    YM.       M YM. ,P MM
    //   MM        M  YMMMMb   MMMMMMM             d'  YM.    YMMMMb   M `Mb d' MM
    //   MM        M      `Mb  MM    `            ,P   `Mb        `Mb  M  YM.P  MM
    //   MM        M       MM  MM                 d'    YM.        MM  M  `Mb'  MM
    //   YM        M       MM  MM                ,MMMMMMMMb        MM  M   YP   MM
    //    8b      d8 L    ,M9  MM       /        d'      YM. L    ,M9  M   `'   MM
    //     YMMMMM9  MYMMMM9  _MMMMMMMMM        _dM_     _dMM_MYMMMM9  _M_      _MM_
    //
    x[i]   = u; u >>>= 8;
    x[i+1] = u; u >>>= 8;
    x[i+2] = u; u >>>= 8;
    x[i+3] = u;
}
```

19MB/s

```
function U32TO8_LE(x, i, u) {
    // ____     ___  ____   __________              _        ____   ___       ___
    // `MM'     `M' 6MMMMb\ `MMMMMMMMM             dM.      6MMMMb\ `MMb     dMM'
    //  MM       M 6M'    `  MM      \            ,MMb     6M'    `  MMM.   ,PMM
    //  MM       M MM        MM                   d'YM.    MM        M`Mb   d'MM
    //  MM       M YM.       MM    ,             ,P `Mb    YM.       M YM. ,P MM
    //  MM       M  YMMMMb   MMMMMMM             d'  YM.    YMMMMb   M `Mb d' MM
    //  MM       M      `Mb  MM    `            ,P   `Mb        `Mb  M  YM.P  MM
    //  MM       M       MM  MM                 d'    YM.        MM  M  `Mb'  MM
    //  YM       M       MM  MM                ,MMMMMMMMb        MM  M   YP   MM
    //   8b     d8 L    ,M9  MM      /         d'      YM. L    ,M9  M   `'   MM
    //    YMMMMM9  MYMMMM9  _MMMMMMMMM       _dM_     _dMM_MYMMMM9  _M_      _MM_
    // ^^^^^^ Don't remove this comment it makes the code 10x faster. ^^^^^^^^^^^
    x[i]   = u; u >>>= 8;
    x[i+1] = u; u >>>= 8;
    x[i+2] = u; u >>>= 8;
    x[i+3] = u;
}
```

we say performance
we mean jsperf.com

we say performance

we mean jsperf.com

[or at least we used to]

Count the occurrences of a single character in a string.

Preparation code

```
<script>
  Benchmark.prototype.setup = function() {
    var matcher = /e/g;
    var testString = "We the People of the United States, in Order to form a more perfect Union, establish
Justice, insure domestic Tranquility, provide for the common defence,promote the general Welfare, and secure
the Blessings of Liberty to ourselves and our Posterity, do ordain and establish this Constitution for the
United States of America.";
  };
</script>
```

Test runner

Ready to run.

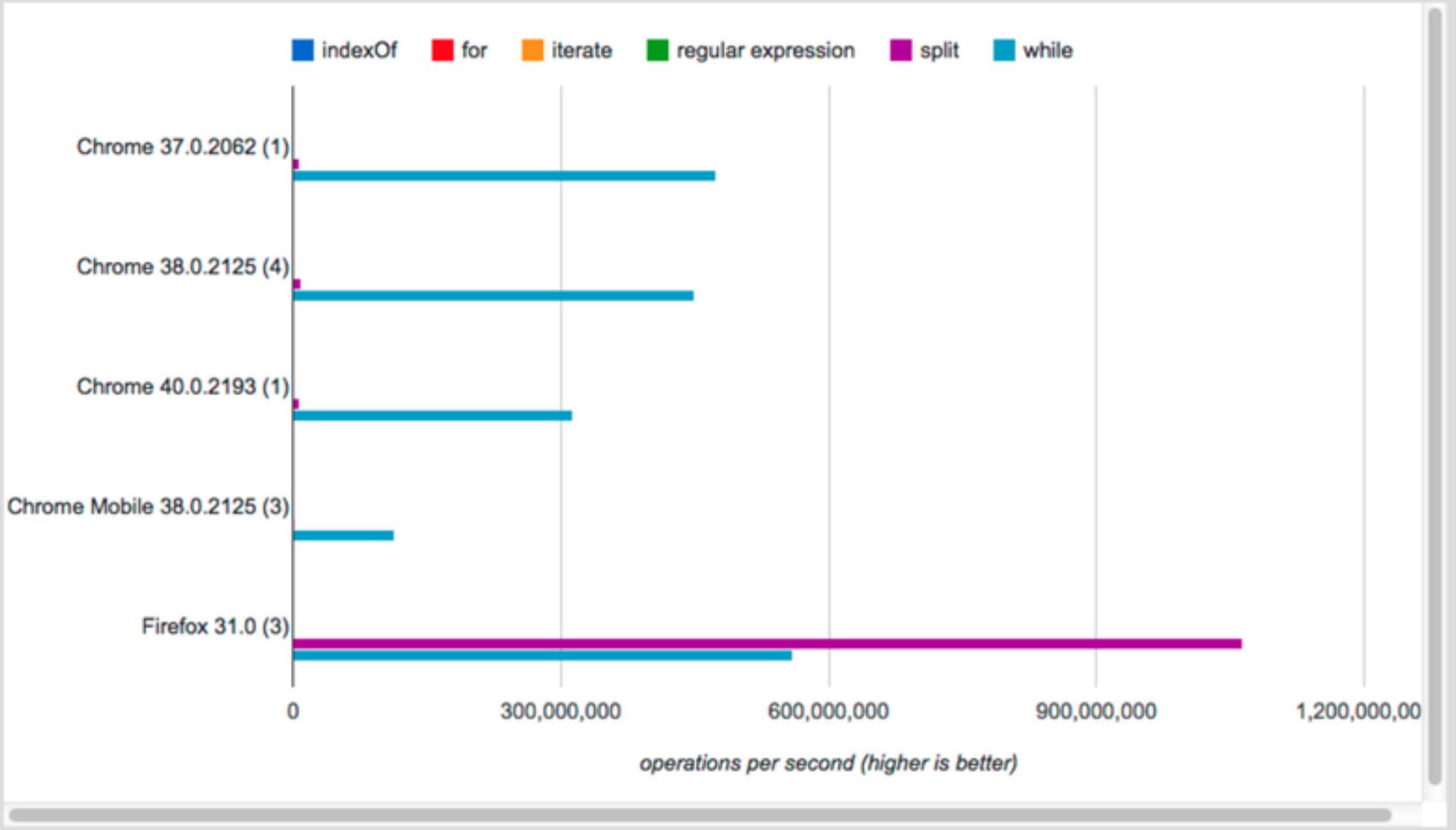
Run tests

Testing in Chrome 41.0.2224.3 on OS X 10.9.5		
	Test	**Ops/sec**
regular expression	`testString.match(matcher).length;`	ready
split	`testString.split("e").length - 1;`	ready

iterate	`var count = 0;`
	`for (var i = 0; i < testString.length; i++) {`
	`  if (testString.charAt(i) === "e") {`
	`  count += 1;`
	`  }`
	`}`
		ready

while	`var count = 0;`
	`while (testString.length) {`
	`  if (testString.charAt(0) === "e") {`
	`  count += 1;`
	`  }`
	`  testString = testString.slice(1);`
	`}`
		ready

Browserscope



Browserscope thinks you are using **Chrome 41.0.2224**

```
// split
str.split("e").length - 1;

// while
var count = 0;
while (str.length) {
  if (str.charAt(0) === "e") {
    count += 1;
  }
  str = str.slice(1);
}
```

```
// split (IT'S OVER 9000K OP/S ON FF!)
str.split("e").length - 1;

// while
var count = 0;
while (str.length) {
  if (str.charAt(0) === "e") {
    count += 1;
  }
  str = str.slice(1);
}
```

DOUBT

EVERYTHING

µbench 101

the cost of OP?

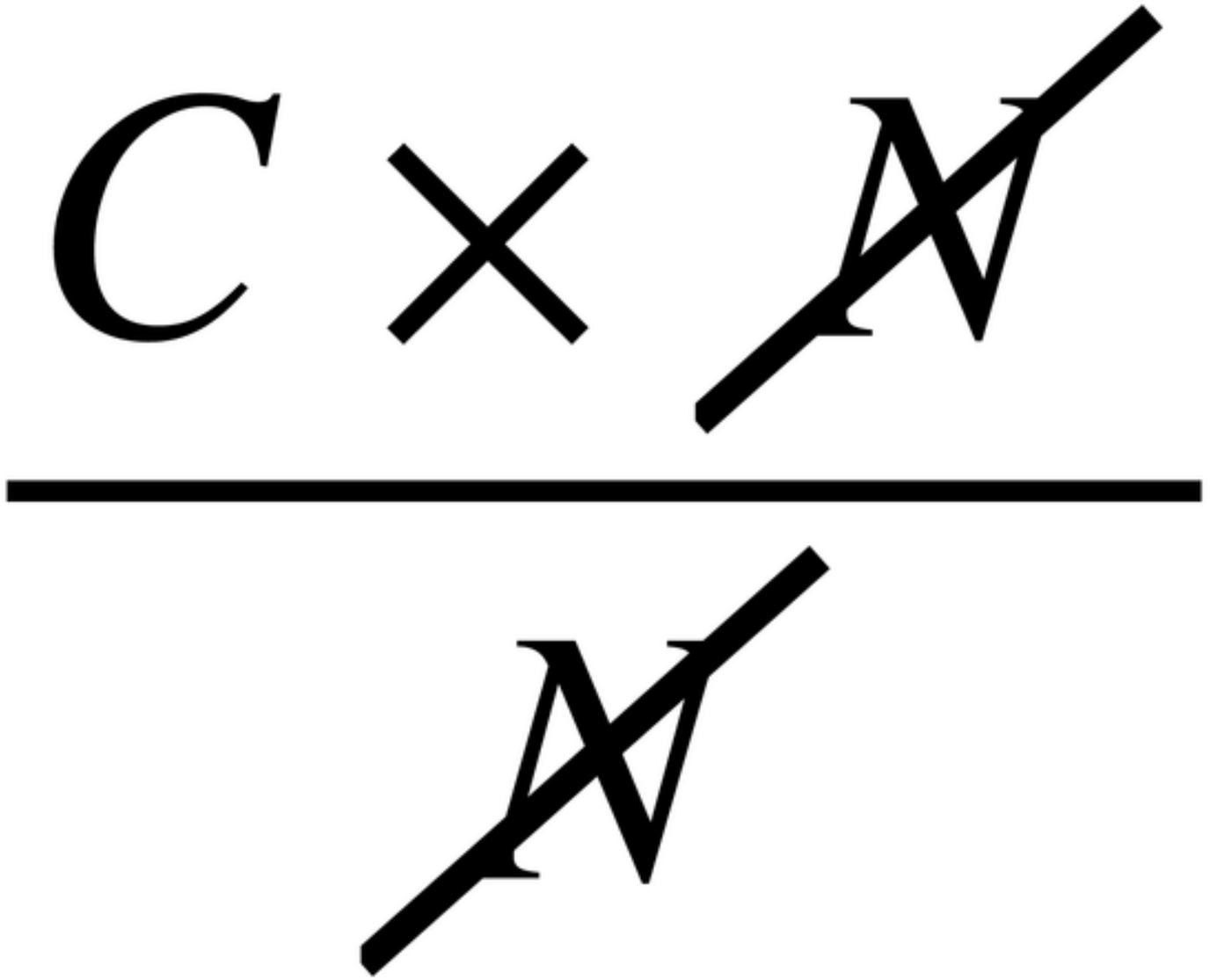
```
function benchmark() {
  var start = Date.now();
  /* OP */
  return (Date.now() - start);
}
```

what if OP faster than clock?

```
function benchmark(N) {
  var start = Date.now();
  for (var i = 0; i < N; i++) {
    /* OP */
  }
  return (Date.now() - start) / N;
}
```

$$C \times N$$

$$\frac{C \times N}{N}$$



```
while (str.length) {
  if (str.charAt(0) === "e") {
    count += 1;
  }
  str = str.slice(1);
}
```

```
function benchmark() {
  var str = "...";
  var start = Date.now();
  for (var i = 0; i < N; i++) {
    var count = 0;
    while (str.length) {
      if (str.charAt(0) === "e") {
        count += 1;
      }
      str = str.slice(1);
    }
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var str = "...";
  var start = Date.now();
  for (var i = 0; i < N; i++) {
    var count = 0;
    while (str.length) {
      if (str.charAt(0) === "e") {
        count += 1;
      }
      str = str.slice(1);
    }
  }  // here str === ""
  return (Date.now() - start) / N;
}
```

repetitions start with `str === ""`

$$C \times N$$

$$C \times 1 + 0 \times (N - 1)$$

$$\frac{C \times 1 + 0 \times (N - 1)}{N}$$

$$\frac{C}{N}$$

$$\frac{C}{N} \approx 0$$

benchmark.js misuse

OP should take the same time when repeatedly executed

Preparation code

```
<script>
  Benchmark.prototype.setup = function() {
    var i = '1561565', j,
    superParseInt = function(a) {
            return ~~parseInt(a, 10);
        };
  };
</script>
```

Test runner

Ready to run.

Run tests

Testing in Chrome 29.0.1547.76 on OS X 10.8.5		
Test		**Ops/sec**
Double Tilde	`j = ~~i;`	ready
ParseInt	`j = parseInt(i, 10);` `j = isNaN(j) ? 0 : j;`	ready
Including true	`j = i === true ? 1 : ~~parseInt(i, 10);`	ready
Slight change to parseInt	`j = ~~parseInt(i, 10);`	ready


Firefox 19.0 (1)
Firefox 21.0 (1)
Firefox 21.0 (1)
Double Tilde: 607,665,833 ops/sec

«~~ for the win!»


Firefox 19.0 (1)
Firefox 21.0 (1)
Firefox 21.0 (1)
Double Tilde: 607,665,833 ops/sec

NOPE

JITs are clever

JITs optimize

as program runs

$$C \times N$$

$$C_u N_u + C_o N_o$$

[unoptimized + optimized version]

$$\sum C_i N_i$$

[multiple unoptimized and optimized versions]

$$\sum C_i N_i + \frac{1}{\sqrt{2\pi}} \int C(\xi) e^{-\frac{\xi^2}{2}} d\xi$$

[math gets too complicated to approach analytically]

JITs are clever

to measure you need to *outsmart*

Disclaimer: On slides I will show optimizations as source-to-source transformations. In reality compilers operate on more sophisticated representations.

```
function benchmark() {
  var i = '1561565', j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = ~~i;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = ~~'1561565';
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = ~-1561566;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = 1561565;
  }
  return (Date.now() - start) / N;
}
```

constant propagation

«HEY, I CAN TRICK THE COMPILER!»

```
function benchmark() {
  var i = Date.now().toString(), j;
  //  ^ not a constant any more
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = ~~i;
  }
  return (Date.now() - start) / N;
}
```

«WRONG ANSWER, MCFLY!»

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    j = ~~i;
    //  ^^^ loop invariant
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  var j0 = ~~i;
  //  ^^^^^^^^ hoisted from the loop
  for (var n = 0; n < N; n++) {
    j = j0;
  }
  return (Date.now() - start) / N;
}
```

loop invariant code motion

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  var j0 = ~~i;
  for (var n = 0; n < N; n++) {
    j = j0;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  var j0 = ~~i;
  for (var n = 0; n < N; n++) {
    j = j0;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  var j0 = ~~i;
  for (var n = 0; n < N; n++) {
    j = j0;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(), j;
  var start = Date.now();
  var j0 = ~~i;
  for (var n = 0; n < N; n++) {
    j = j0;
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    /* sound of silence */
  }
  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var start = Date.now();
  /*
   * sound of silence
   */
  return (Date.now() - start) / N;
}
```

dead code elimination

```
// Remember this one?
// split (IT'S OVER 9000K OP/S ON FF!)
str.split("e").length - 1;
```

```
// Remember this one?
// split (IT'S OVER 9000K OP/S ON FF!)
str.split("e").length - 1;
// it's full of dead code.
```

optimizer eats μbenchmarks for breakfast

chew proof μbenchmarks?

do **not** try
to write them

2. no constants
3. no loop invariants
4. no dead code

1. **verify results**
2. no constants
3. no loop invariants
4. no dead code

```
function benchmark() {


  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {

    j = ~~i;
  }

  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(),
      i1 = Date.now().toString(), t;
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {

    j = ~~i;
  }

  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(),
      i1 = Date.now().toString(), t;
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    t = i; i = i1; i1 = t;
    j = ~~i;
  }

  return (Date.now() - start) / N;
}
```

```
function benchmark() {
  var i = Date.now().toString(),
      i1 = Date.now().toString(), t;
  var j;
  var start = Date.now();
  for (var n = 0; n < N; n++) {
    t = i; i = i1; i1 = t;
    j = ~~i;
  }
  if (i != j || i1 != j) throw "whoa?";
  return (Date.now() - start) / N;
}
```

not bad

(potentially)

still not

enough

```
for (var n = 0; n < N; n++) {
  t = i; i = i1; i1 = t;
  j = ~~i;
}
```

```
for (var n = 0; n < (N / 2); n++) {
  t = i; i = i1; i1 = t;
  j = ~~i;
  t = i; i = i1; i1 = t;
  j = ~~i;
}
if ((N % 2) === 1) {
  t = i; i = i1; i1 = t;
  j = ~~i;
}
```

```
for (var n = 0; n < (N / 2); n++) {
  j = ~~i1;
  t = i1;
  j = ~~i;
}
```

```
for (var n = 0; n < (N / 2); n++) {
  j = ~~i1; /* dead */
  t = i1;   /* dead */
  j = ~~i;  /* invariant */
}
```

loop unrolling

V8 doesn't do it

V8 doesn't do it

but I want to induce paranoia ☺️

presumption of performance

reasonable code
reasonably fast

confirmation bias

«prototype
chains are slow»

```
var obj =
  Object.create(
    Object.create(
      Object.create(
        Object.create(
          Object.create({prop: 10})))));
```

```
var obj =
  Object.create(
    Object.create(
      Object.create(
        Object.create(
          Object.create(
            Object.create({prop: 10})))));
                    // LISP tribute ^^^^^
```

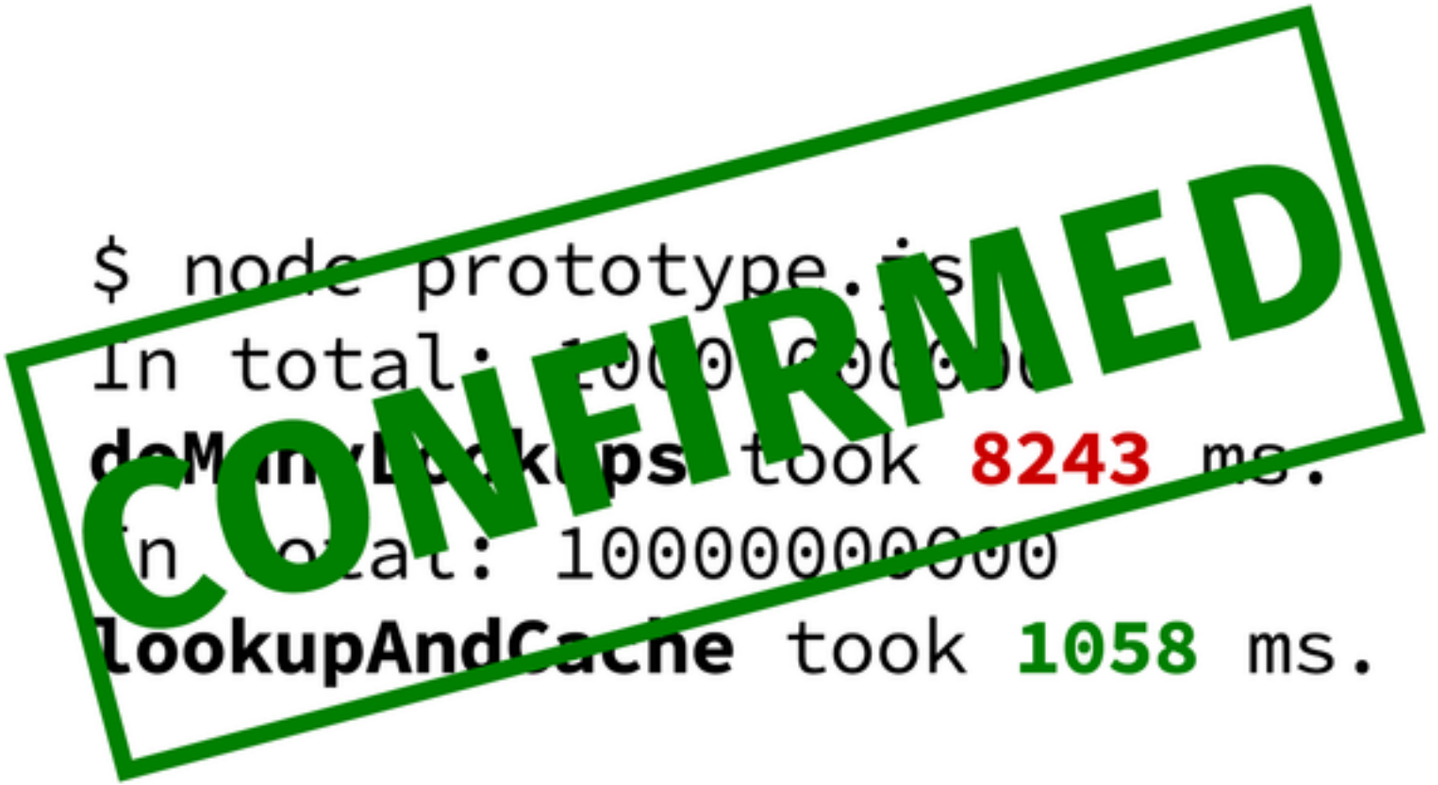
```
function doManyLookups() {
  var counter = 0;
  for(var i = 0; i < 1000; i++)
    for(var j = 0; j < 1000; j++)
      for(var k = 0; k < 1000; k++)
        counter += obj.prop;
  print('In total: ' + counter);
}
```

```
function lookupAndCache() {
  var counter = 0;
  var value = obj.prop;
  for(var i = 0; i < 1000; i++)
    for(var j = 0; j < 1000; j++)
      for(var k = 0; k < 1000; k++)
        counter += value;
  print('In total: ' + counter);
}
```

```
// State of art benchmark driver.
function measure(f) {
  var start = Date.now();
  f();
  var end = Date.now();
  print(f.name + ' took ' +
        (end - start) + ' ms.');
}
```

```
measure(doManyLookups);
measure(lookupAndCache);
```

```
$ node prototype.js
In total: 10000000000
doManyLookups took 8243 ms.
In total: 10000000000
lookupAndCache took 1058 ms.
```


$ node prototype.js
In total: 1000 000000
doManyLookups took 8243 ms.
In total: 1000000000
lookupAndCache took 1058 ms.
CONFIRMED

lets make it harder

```
-    Object.create({ prop: 10 }))))));
+    Object.create({ get prop () { return 10 } }))))));
```

```
$ node prototype.js
In total: 10000000000
doManyLookups took 1082 ms.
In total: 10000000000
lookupAndCache took 1061 ms.
```

«what kind of
voodoo is this?»

```
 B13
 v96 BlockEntry
 v97 Simulate id=90
 v98 StackCheck changes[NewSpacePromotion]
v103 Simulate id=113 push d81, push t99
v104 EnterInlined prop, id=4
v106 LeaveInlined
v107 Simulate id=111 push t105
v108 Goto B14


 B14
v109 BlockEntry
d110 Add d81 d147 !
i                                    483647,1000,m0=0]
v113 Simulate id=86 pop 2 / var[2] = d110, var
v114 Goto B11
```

inlined

Constant 10 range[10,10,m0=0]

```
B13
v96  BlockEntry
v97  Simulate id=90
v98  StackCheck changes[NewSpacePromotion]
t99  LoadContextSlot t30[4]
t100 LoadNamedGeneric t99.prop changes[*]
v101 Simulate id=111 push d81, push t100
d139 Change t100 t to d
d102 Add d81 d139 !
i104 Add i84 i103 range[-2147483647,1000,m0=C
v105 Simulate id=86 pop 2 / var[2] = d102, va
v106 Goto B11
```

very old V8 did not inline loads from data properties defined on prototypes

trying newer V8

```
$ d8 prototype.js
In total: 10000000000
doManyLookups took 1294 ms.
In total: 10000000000
lookupAndCache took 1189 ms.
```

```
$ d8 prototype.js
In total: 10000000000
doManyLookups took 1294 ms.
In total: 10000000000
lookupAndCache took 1189 ms.
```

prototype chain traversal got LICMed

… and now for something completely different

what if we run benchmark twice?

```
measure(doManyLookups);
measure(doManyLookups);
measure(lookupAndCache);
measure(lookupAndCache);
```

```
$ d8 prototype.js | grep took
doManyLookups took 1301 ms.
doManyLookups took 3408 ms.
lookupAndCache took 1204 ms.
lookupAndCache took 3406 ms.
```

what just happened here?


toString
doManyLookups
doManyLookups
doManyLookups
doManyLookups
lookupAndCache
deopt: smi overflow
deopt: int32 overflow
deopt: loop end
stabilized

```
B13
            counter += obj.prop;
d221 Change t87 t to d allow-undefined-as-nan
            counter += obj.prop;
d120 Add d221 d222 !
          for(var k = 0; k < 1000; k++) {
s125 Add s90 s124
          for(var k = 0; k < 1000; k++) {
t223 Change d120 d to t changes[NewSpacePromot Number
v128 Goto B11
```

unbox

add 10

rebox

`'In total: ' + counter` type-feedback leaks upwards into the loop and causes excessive boxing of the `counter` variable

workaround: hide + from representation inference

```
-    print('In total: ' + counter);
+    print('In total: ' + counter.toString());
```

```
$ d8 prototype.js | grep took
doManyLookups took 1298 ms.
doManyLookups took 1119 ms.
lookupAndCache took 1188 ms.
lookupAndCache took 982 ms.
```

```
Benchmark.prototype.setup = function ()
  var obj = { }

  var _a = 0;
  Object.defineProperty(obj, 'a', {
    get: function () { return _a; },
    set: function (val) { _a = val; }
  });
};
```

```
suite.add('Accessor', function () {
  obj.a = 2;
  obj.a;
});
```

```
$ v5.3/d8 concat.js
Accessor  x   3,840,922 ops/sec ±1.54%
```

```
$ v5.3/d8 concat.js
Accessor  x   3,840,922 ops/sec ±1.54%

$ v3.31/d8 concat.js
Accessor  x 874,731,387 ops/sec ±1.09%
```

```
$ v5.3/d8 concat.js
Accessor  x    3,840,922 ops/sec ±1.54%

$ v3.31/d8 concat.js
Accessor  x 874,731,387 ops/sec ±1.09%
```

```
function f() {
  var obj = { }
  var _a = 0;
  Object.defineProperty(obj, 'a', {
    get: function () { return _a; },
    set: function(val) { _a = val; }
  });
  for (var i = 0; i < 1e6; i++) {
    obj.a = 2;
    obj.a;
  }
}
```

```
f();  // =>   2ms
```

```
f();   // =>   2ms
f();   // => 287ms
```

```
f();   // =>    2ms
f();   // => 287ms
```

```
Object.defineProperty(obj, 'a', {
  /* ... */
});
print('obj has fast properties: ' +
      %HasFastProperties(obj));
```

```
$ d8 --allow-natives-syntax
obj has fast properties: true
f() took 2ms
obj has fast properties: false
f() took 287ms
```

hidden class
transition clash

```
var obj1 = { }
Object.defineProperty(obj1, 'a', {
  get: function () { /* ... */ },
  set: function(val) { /* ... */ }
});
var obj2 = { }
Object.defineProperty(obj2, 'a', {
  get: function () { /* ... */ },
  set: function(val) { /* ... */ }
});
// Initial hudden class for obj1/obj2 is
// but after defineProperty they have d
```

```
Object.defineProperty(obj, 'a', {
  /* ... */
});
Object.create(obj);
```

```
f();  // =>  4ms
f();  // =>  3ms
```

```
f();   // =>    4ms
f();   // =>    3ms
f();   // =>    3ms
f();   // =>    9ms
f();   // =>   28ms
```

```
function g() {
  var obj = { }
  var _a = 0;
  obj.a = {
    get: function () { return _a; },
    set: function(val) { _a = val; }
  };

  for (var i = 0; i < 1e6; i++) {
    obj.a.set(2);
    obj.a.get();
  }
}
```

```
g();   // =>  23ms
g();   // =>  15ms
g();   // =>  16ms
g();   // =>  22ms
g();   // =>  16ms
```

```
function Box(_a) {
  this._a = _a;
}

Box.prototype.get = function () {
  return this._a;
};

Box.prototype.set = function (val) {
  this._a = val;
};
```

```
function h() {
  var obj = { }
  obj.a = new Box(0);
  for (var i = 0; i < 1e6; i++) {
    obj.a.set(2);
    obj.a.get();
  }
}
```

```
h();  // =>   2ms
h();  // =>   1ms
h();  // =>   0ms
h();  // =>   1ms
h();  // =>   1ms
```

KNOW FUNDAMENTALS!

we can look into other VMs!

```
$ jsc test.js
f();  // =>   3ms
f();  // =>   3ms
f();  // =>   16ms
f();  // =>   17ms
f();  // =>   11ms
```

```
$ jsc --useConcurrentJIT=false \
      --showDisassembly=true   \
      test.js >& code.asm
```

```
$ jsc --useConcurrentJIT=false \
      --showDisassembly=true   \
      test.js >& code.asm
# there'll be three DFG versions of func
```

```
82:<!7:loc14> GetLocal(Untyped:@178, JS|MustGen|UseAsOther, Final, loc6(O<Final>/FlushedCell), machine:loc5, R:Stack(-7), bc#1
    0x22d0b7400516: mov -0x30(%rbp), %rax
84:<!0:-> CheckStructure(Cell:@82, MustGen, [%CS:Object], R:JSCell_structureID, Exits, bc#102)
    0x22d0b740051a: cmp $0xff, (%rax)
    0x22d0b7400520: jnz 0x22d0b7400871
85:< 2:loc13> GetGetterSetterByOffset(KnownCell:@82, KnownCell:@82, JS|UseAsOther, Othercell, id5{a}, 0, inferredType = Top, R
    0x22d0b7400526: mov 0x10(%rax), %rsi
86:< 1:loc9>  GetSetter(KnownCell:@85, JS|UseAsOther, Function, R:GetterSetter_setter, Exits, bc#102)
    0x22d0b740052a: mov 0x18(%rsi), %rdx
87:<!0:-> MovHint(Untyped:@82, MustGen, loc22, W:SideState, ClobbersExit, bc#102)
89:<!0:-> MovHint(Untyped:@81, MustGen, loc21, W:SideState, ClobbersExit, bc#102, ExitInvalid)
91:<!0:-> ExitOK(MustGen, W:SideState, bc#102)
92:<!0:-> CheckCell(Cell:@86, Untyped:@82, MustGen, <0x1035cc970, Function>, set#EoFqmc/<nogen>:[0x1035a1de0], Exits, bc#102)
    0x22d0b740052e: mov $0x1035cc970, %r11
    0x22d0b7400538: cmp %r11, %rdx
    0x22d0b740053b: jnz 0x22d0b7400887
--> set#EoFqmc:<0x103597620, bc#102, SetterCall, known callee: Cell: 0x1035cc970 (%BA:Function), ID: 49, numArgs+this = 2, sta
```

```
82:<!7:loc14> GetLocal(Untyped:@178, JS|MustGen|UseAsOther, Final, loc6(O<Final>/FlushedCell), machine:loc5, R:Stack(-7), bc#1
    0x22d0b7400516: mov -0x30(%rbp), %rax
84:<!0:-> CheckStructure(Cell:@82, MustGen, [%CS:Object], R:JSCell_structureID, Exits, bc#102)
    0x22d0b740051a: cmp $0xff, (%rax)
    0x22d0b7400520: jnz 0x22d0b7400871
85:< 2:loc13> GetGetterSetterByOffset(KnownCell:@82, KnownCell:@82, JS|UseAsOther, Othercell, id5{a}, 0, inferredType = Top, R
    0x22d0b7400526: mov 0x10(%rax), %rsi
86:< 1:loc9>  GetSetter(KnownCell:@85, JS|UseAsOther, Function, R:GetterSetter_setter, Exits, bc#102)
    0x22d0b740052a: mov 0x18(%rsi), %rdx
87:<!0:-> MovHint(Untyped:@82, MustGen, loc22, W:SideState, ClobbersExit, bc#102)
89:<!0:-> MovHint(Untyped:@81, MustGen, loc21, W:SideState, ClobbersExit, bc#102, ExitInvalid)
91:<!0:-> ExitOK(MustGen, W:SideState, bc#102)
92:<!0:-> CheckCell(Cell:@86, Untyped:@82, MustGen, <0x1035cc970, Function>, set#EoFqmc/<nogen>:[0x1035a1de0], Exits, bc#102)
    0x22d0b740052e: mov $0x1035cc970, %r11
    0x22d0b7400538: cmp %r11, %rdx
    0x22d0b740053b: jnz 0x22d0b7400887
--> set#EoFqmc:<0x103597620, bc#102, SetterCall, known callee: Cell: 0x1035cc970 (%BA:Function), ID: 49, numArgs+this = 2, sta
```

```
82 GetLocal(@178)
84 CheckStructure(@82)
85 GetGetterSetterByOffset(@82)
86 GetSetter(@85)
92 CheckCell(Cell:@86, ...)
--> set#EoFqmc:<... known callee ...>
```

```
80 CheckStructure(@78)
81 GetGetterSetterByOffset(@78)
82 GetSetter(@81)
88 Call(@82)
```

```
80 CheckStructure(@78)
81 GetGetterSetterByOffset(@78)
82 GetSetter(@81)
88 Call(@82)
```

NOW DESERT

method call
vs.
function call

```
function mk(word) {
  var len = word.length;
  if (len > 255) return undefined;
  var i = len >> 2;
  return String.fromCharCode(
    (word.charCodeAt(    0) & 0x03) << 14 |
    (word.charCodeAt(    i) & 0x03) << 12 |
    (word.charCodeAt(  i+i) & 0x03) << 10 |
    (word.charCodeAt(i+i+i) & 0x03) <<  8 |
    len
  );
}
```

```
Benchmark.prototype.setup = function() {
  function mk(word) {
    /* ... */
  }

  var MK = function() { };
  MK.prototype.mk = mk;
  var mker = new MK;
};
```

```
suite
  .add('Function', function() {
    var key = mk('www.wired.com');
    key = mk('www.youtube.com');
    key = mk('scorecardresearch.com');
    key = mk('www.google-analytics.com');
  })
  .add('Method', function() {
    var key = mker.mk('www.wired.com');
    key = mker.mk('www.youtube.com');
    key = mker.mk('scorecardresearch.com');
    key = mker.mk('www.google-analytics.com');
  })
```

```
$ d8 method-vs-function.js
Function x   4,149,776 ops/sec ±0.62%
Method   x 682,273,122 ops/sec ±0.72%
Fastest is Method
```

method call
is faster?

KEEP SECRET
WHAT I AM GOING
TO SHOW YOU NOW

```
--- a/method-function.js
+++ b/method-function.js
@@ -2,6 +2,9 @@
 load("../benchmark.js");

 Benchmark.prototype.setup = function() {
+    "Speed" + "your" + "JS" + "with" +
+    "this" + "one" + "weird" + "trick";
+
     function mk(word) {
         var len = word.length;
         if (len > 255) return undefined;
```

```
$ d8 method-vs-function.js
Function x 695,708,197 ops/sec ±0.38%
Method   x 692,496,013 ops/sec ±0.29%
Fastest is Function,Method
```

isFunction

no optimized instances!

select

filter by name

Function

Function was never optimized!

— heuristics that decide when to optimize are based (among other things) on the amount of initialized inline caches

— function call does not go through an IC, but method call does

— `Method` had enough initialized ICs to trigger optimizations, but `Function` didn't!

— Until fake + ICs were added in the `setup` section

— heuristics that decide when to optimize are based (among other things) on the amount of initialized inline caches

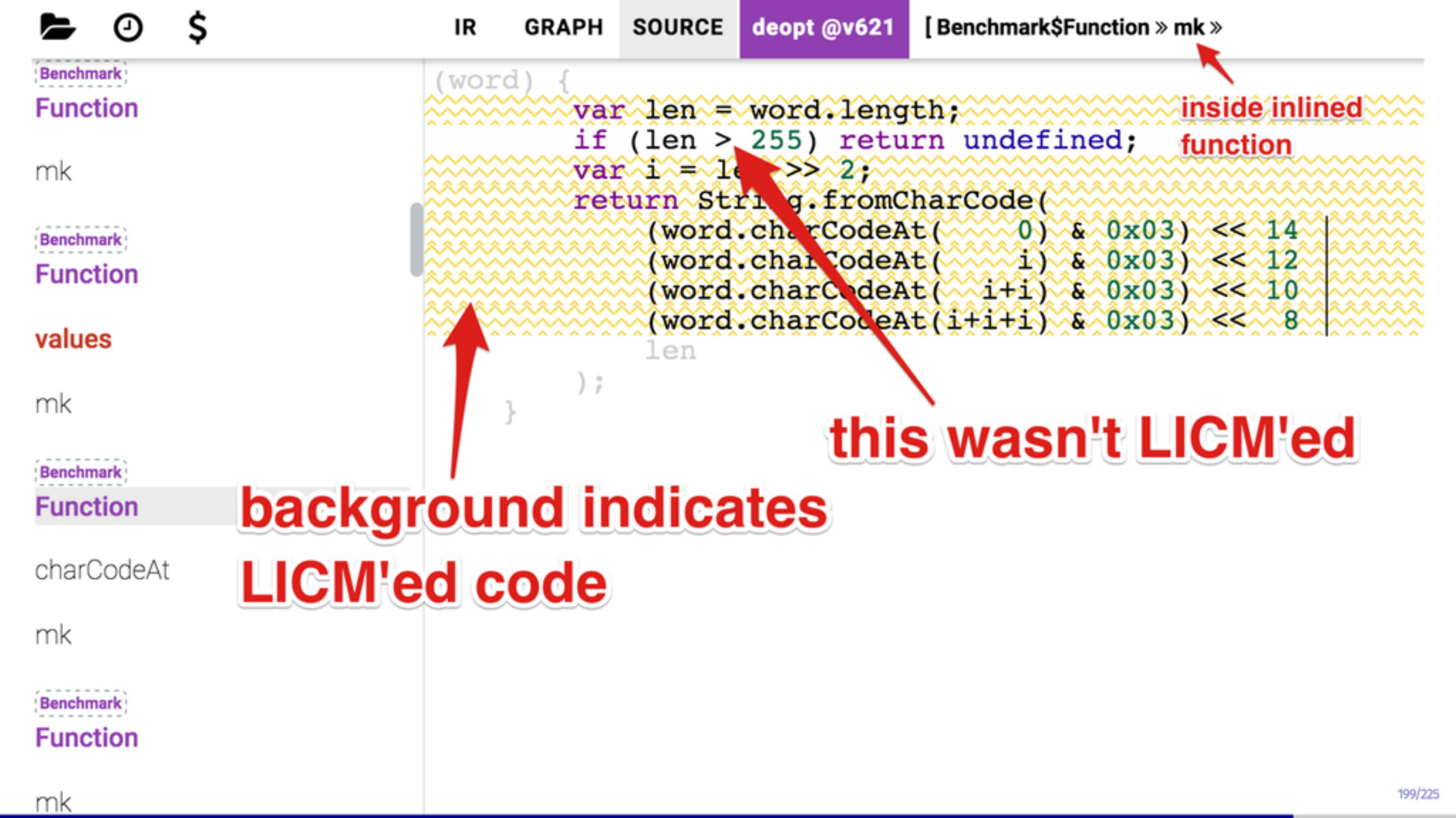
— function call does not go through IC, but method call does

— `Method` had enough initialized ICs to trigger optimizations, but `Function` didn't!

— Until fake + ICs were added in the `setup` section

yada yada


IR
GRAPH
SOURCE
deopt @v621
[Benchmark$Function »
Benchmark
Function
mk
Benchmark
Function
values
mk
Benchmark
Function
charCodeAt
mk
Benchmark
Function
mk
function mk(word) {
var len = word.length;
if (len > 255) return undefined;
var i = len >> 2;
return String.fromCharCode(
(word.charCodeAt(0) & 0x03) << 14
(word.charCodeAt(i) & 0x03) << 12
(word.charCodeAt(i+i) & 0x03) << 10
(word.charCodeAt(i+i+i) & 0x03) << 8
len
);
}
var MK = function() { };
MK.prototype.mk = mk;
var mker = new MK;
s140572296482617=n140572296482617.now();while(i140572296
var key;
key = mk('www.wired.com');
key = mk('www.youtube.com');
key = mk('scorecardresearch.com');
key = mk('www.google-analytics.com');
}r140572296482617=(n140572296482617.now()-s14057229648
// no operation performed
inlining marker
198/225


IR
GRAPH
SOURCE
deopt @v621
[Benchmark$Function » mk »
Benchmark
Function
mk
Benchmark
Function
values
mk
Benchmark
Function
charCodeAt
mk
Benchmark
Function
mk
(word) {
var len = word.length;
if (len > 255) return undefined;
var i = le >> 2;
return Stri g.fromCharCode(
(word.cha CodeAt(0) & 0x03) << 14
(word.cha CodeAt(i) & 0x03) << 12
(word.charC deAt(i+i) & 0x03) << 10
(word.charCo eAt(i+i+i) & 0x03) << 8
len
);
}
inside inlined function
this wasn't LICM'ed
background indicates LICM'ed code
199/225

measuring *almost*
empty loop again!

can function call be faster than method call?

```
// for (var item in list) { ... }
var sum = 0;
for (var i = 0, L = arr.length;
     i < arr.length;
     ++i) {
  if (arr.length !== L)
    H.throwConcurrentModificationError(arr);
  var item = list[i];
  sum += item;
}
```

```
// for (var item in list) { ... }
var sum = 0;
for (var i = 0, L = arr.length;
     i < arr.length;
     ++i) {
  // if (arr.length !== L)
  //   H.throwConcurrentModificationError(arr)
  var item = list[i];
  sum += item;
}
```

```
$ d8 iteration.js
WithCheck    x 396,792 ops/sec ±0.37%
WithoutCheck x 456,653 ops/sec ±0.82%
Fastest is WithoutCheck (by 18%)
```

```
// for (var item in list) { ... }
var sum = 0;
for (var i = 0, L = arr.length;
     i < arr.length;
     ++i) {
  if (arr.length !== L)
    H.throwConcurrentModificationError(arr);
  var item = list[i];
  sum += item;
}
```

```
// for (var item in list) { ... }
var sum = 0;
for (var i = 0, L = arr.length;
     i < arr.length;
     ++i) {
  if (arr.length !== L)
    (0,H.throwConcurrentModificationError)(arr
  var item = list[i];
  sum += item;
}
```

```
$ d8 iteration.js
WithCheck    x 396,792 ops/sec ±0.37%
WithoutCheck x 456,653 ops/sec ±0.82%
WithHack     x 462,698 ops/sec ±0.48%
Fastest is WithoutCheck,WithHack
```

18% speedup by replacing

`o.f()`

with

`(0,o.f)()`

in the code that never executes

```
(window,t14313604834667) { var global = window, clearTimeout = global.cl
var r143136048346 67,s143136048346 67,m143136048346 67=this,f143136048346 67
  for (var i = 0; i < 1000; i++) arr.push(i);
s143136048346 67=n143136048346 67.now();while(i143136048346 67--){
var sum = 0;
    for (var i = 0, l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        H.throwConcurrentModificationError(arr);
      sum +=  arr[i];
    }
}r143136048346 67=(n143136048346 67.now()- 143136048346 67)/1e3;
// no operation performed
return{elapsed:r143136
```

never executed and

V8 doesn't know where it goes

```
(window,t14313604834667) { var global = window, clearTimeout = global.cl
var r14313604834667,s1431360483[illegible] m[illegible]=this.f1431[illegible]604834667
  for (var i = 0; i < 1000; i++[illegible]
s14313604834667=n14313604834667.now();while(i14313604834667--){
var sum = 0;
    for (var i [illegible], l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        H.throwConcurrentModificationError(arr);
      sum += arr[i];
    }
}r14313604834667=(n14313604834667.now()-s14313604834667)/1e3;
// no operation performed
return{elapsed:r14313604834667,uid:"uid14313604834667"}}
```

has to assume length

can change

```
(window,t143136048346657) { var global = window, clearTimeout = global.c
var r143136048346657,s143136048346657,m143136048346657=this,f14313604834
  for (var i = 0; i < 1000; i++) arr.push(i);
s143136048346657=n143136048346657.now();while(i143136048346657--){
var sum = 0;
    for (var i = 0, l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        (0, H.throwConcurrentModificationError)(arr);
      sum += ar[i];
    }
}r143136048346657=(n143136048346657.now()-s143136048346657)/1e3;
// no operation perform
return{elapsed:r143136
```

also never executed

but property loads are special

```
(window,t143136048346657) { var global = window. clearTimeout = global.c
var r143136048346657,s1431360                                          334
  for (var i = 0; i < 1000; i++) arr.push(i);
s143136048346657=n143136048346657.now();while(i143136048346657 ){
var sum = 0;
    for (var i = 0, l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        (0, H.throwConcurrentModificationError)(arr);
      sum += arr[i];
    }
}r143136048346657=(n143136048346657.now()-s143136048346657)/1e3;
// no operation performed
return{elapsed:r143136048346657,uid:"uid143136048346657"}}
```

assume that paths leading to never executed loads are never taken

```
(window,t143136048346657) { var global = window, clearTimeout = global.c
var r143136048346657,s143136048346657,m143136048346657=this,f14313604834
  for (var i = 0; i < 1000; i++) arr.push(i);
s143136048346657=n143136048346657.now();while(i143136048346657--){
var sum = 0;
    for (var i = 0, l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        (0, H.throwConcurrentModificationError)(arr);
      sum += arr[i];
    }
}r143136048346657=(n143136048346657.now()-s143136048346657)/1e3;
// no operation performed
return{elapsed:r143136048346657,uid:"uid143136048346657"}}
```

hoisted

folded away

algorithms first

μbenchmarks last

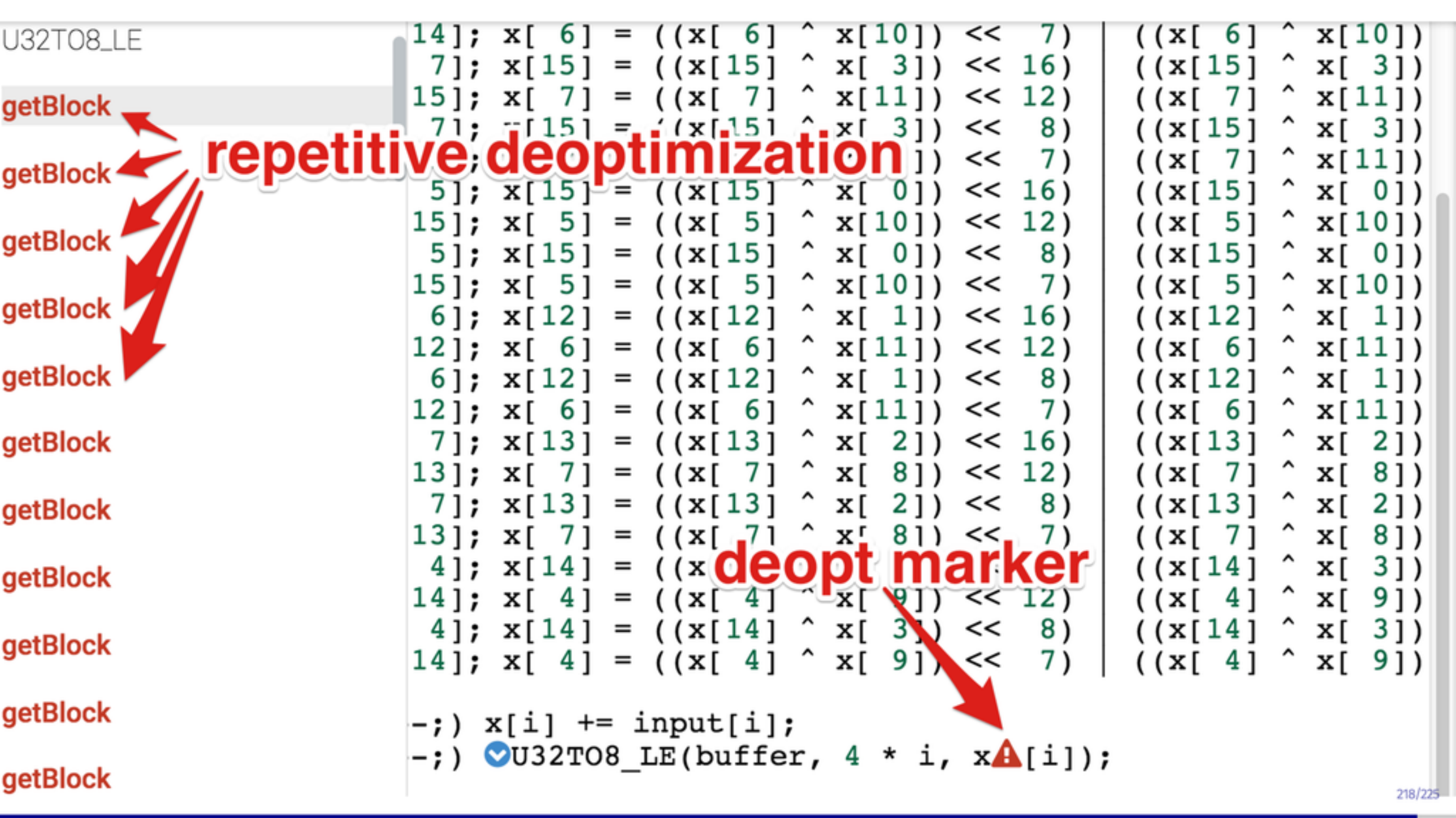
THANK YOU

WAIT!

what about *the first* example?

```
function getBlock(buffer) {
  var x = new Uint32Array(16);

  for (var i = 16; i--;) x[i] = input[i];
  for (var i = 20; i > 0; i -= 2) {
    // quarterRound(x, 0, 4, 8,12);
    x[ 0] += x[ 4]; x[12] = ((x[12] ^ x[ 0]) << 16) | ((
    x[ 8] += x[12]; x[ 4] = ((x[ 4] ^ x[ 8]) << 12) | ((
    x[ 0] += x[ 4]; x[12] = ((x[12] ^ x[ 0]) <<  8) | ((
    x[ 8] += x[12]; x[ 4] = ((x[ 4] ^ x[ 8]) <<  7) | ((
    // ... so on ...
  }
  for (i = 16; i--;) x[i] += input[i];
  for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]);
  input[12]++;
  return buffer;
}
```


U32TO8_LE
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
getBlock
repetitive deoptimization
deopt marker
14]; x[6] = ((x[6] ^ x[10]) << 7) | ((x[6] ^ x[10])
7]; x[15] = ((x[15] ^ x[3]) << 16) | ((x[15] ^ x[3])
15]; x[7] = ((x[7] ^ x[11]) << 12) | ((x[7] ^ x[11])
7]; x[15] = ((x[15] ^ x[3]) << 8) | ((x[15] ^ x[3])
0]) << 7) | ((x[7] ^ x[11])
5]; x[15] = ((x[15] ^ x[0]) << 16) | ((x[15] ^ x[0])
15]; x[5] = ((x[5] ^ x[10]) << 12) | ((x[5] ^ x[10])
5]; x[15] = ((x[15] ^ x[0]) << 8) | ((x[15] ^ x[0])
15]; x[5] = ((x[5] ^ x[10]) << 7) | ((x[5] ^ x[10])
6]; x[12] = ((x[12] ^ x[1]) << 16) | ((x[12] ^ x[1])
12]; x[6] = ((x[6] ^ x[11]) << 12) | ((x[6] ^ x[11])
6]; x[12] = ((x[12] ^ x[1]) << 8) | ((x[12] ^ x[1])
12]; x[6] = ((x[6] ^ x[11]) << 7) | ((x[6] ^ x[11])
7]; x[13] = ((x[13] ^ x[2]) << 16) | ((x[13] ^ x[2])
13]; x[7] = ((x[7] ^ x[8]) << 12) | ((x[7] ^ x[8])
7]; x[13] = ((x[13] ^ x[2]) << 8) | ((x[13] ^ x[2])
13]; x[7] = ((x[7] ^ x[8]) << 7) | ((x[7] ^ x[8])
4]; x[14] = ((x[14] ^ x[3])
14]; x[4] = ((x[4] ^ x[9]) << 12) | ((x[4] ^ x[9])
4]; x[14] = ((x[14] ^ x[3]) << 8) | ((x[14] ^ x[3])
14]; x[4] = ((x[4] ^ x[9]) << 7) | ((x[4] ^ x[9])
-;) x[i] += input[i];
-;) U32TO8_LE(buffer, 4 * i, x[i]);
218/225

repetitive deopt
is always V8 bug

— inlining U32TO8_LE used to break *safe-uint32* analysis

— adding long comment into U32TO8_LE disables inlining

— there are more sane workarounds

```
function getBlock(buffer) {
  var x = new Uint32Array(16);

  for (var i = 16; i--;) x[i] = input[i];
  for (var i = 20; i > 0; i -= 2) {
    // ...
  }
  for (i = 16; i--;) x[i] += input[i];
  for (i = 16; i--;) U32TO8_LE(buffer, 4 * i, x[i]|0);
  //                    immediately truncate to int32 ^^
  input[12]++;
  return buffer;
}
```

never assume that
a language feature
has to be slow

talk to VM people
report bugs