CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Benchmarking JS

Vyacheslav Egorov

Conference: May 11-12 / Workshops: 13-14

Java VM

V8

Dart VM


complexity

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

```
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 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;
}
```

19MB/s

```
function U32TO8_LE(x, i, u) {
    //
    //  _   _ ____  _____   _    ____  __  __ 
    // | | | / ___|| ____| / \  / ___||  \/  |
    // | | | \___ \|  _|  / _ \ \___ \| |\/| |
    // | |_| |___) | |___ / ___ \ ___) | |  | |
    //  \___/|____/|_____/_/   \_\____/|_|  |_|
    //
    // ^^^^^^ 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

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.

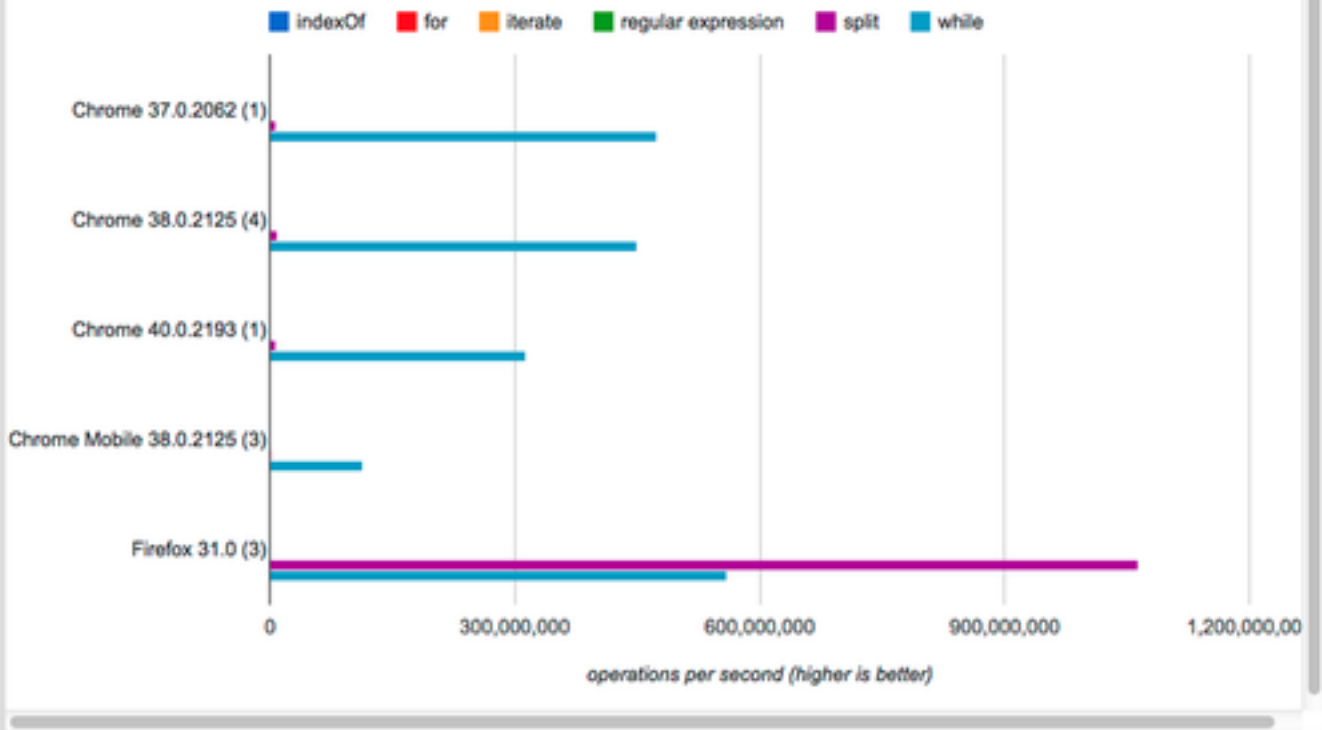
Run tests

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

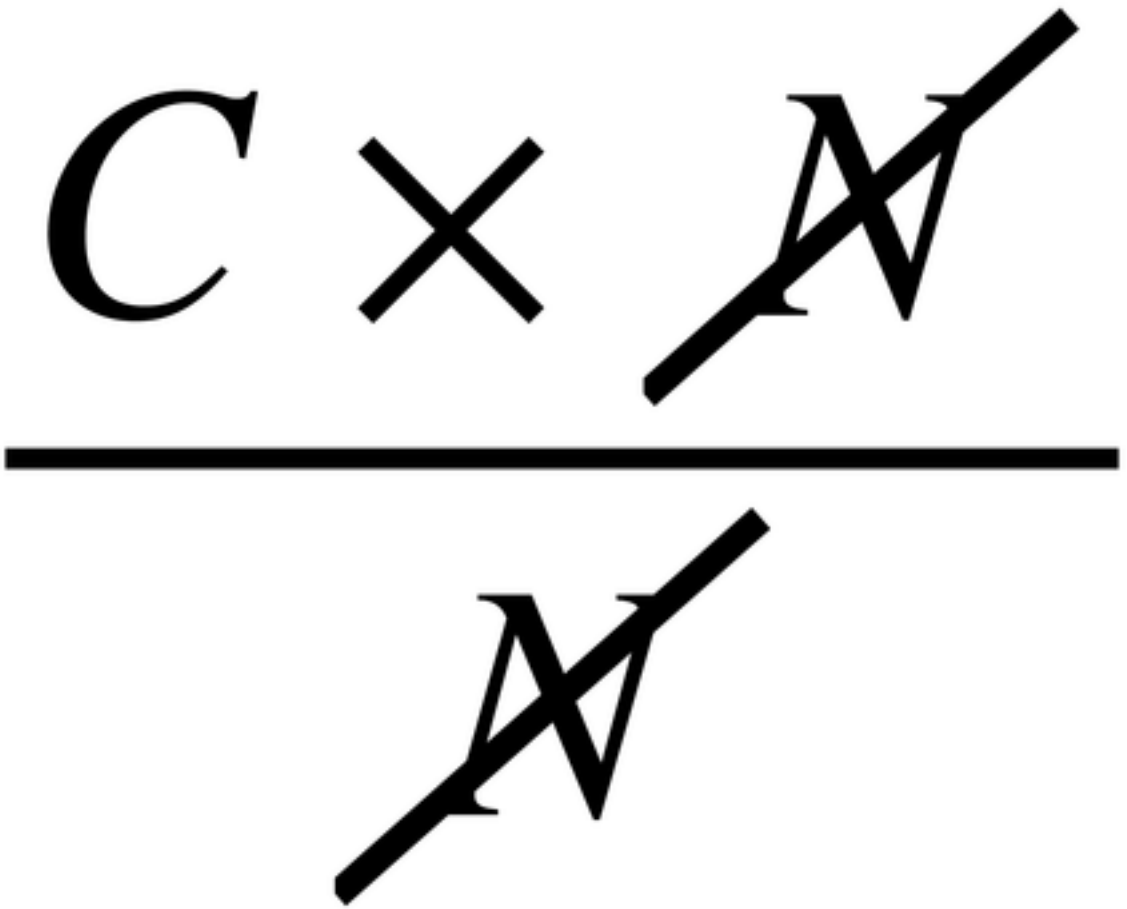
the cost of OP?

```
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;
}
```

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



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

repetitions start with `str === ""`

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

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

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

jsperf 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

$$C_u N_u + C_o N_o$$

[unoptimized + optimized version]

[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 ☺


µbench
VM

```
<script>
  Benchmark.prototype.setup = function() {
    var test1 =
'owijfiojwefiojewijewoijewoijofiejioejffwiejijiefwiowefjoiwejiewjfiewfoiwejewifjwefijewfiojoewfjwefiowejfiewfjiewfjiewfj
ieowfjewfijewiiewfjweiojewfiojewfioejfiewfjiewfo';
    var test2 =
'owijfiojwefiojewijewoijewoijofiejioejffwiejijiefwiowefjoiwejiewjfiewfoiwejewifjwefijewfiojoewfjwefiowejfiewfjiewfjiewfj
ieowfjewfijewiiewfjweiojewfiojewfioejfiewfjiewfo';
    function outsideScope() {
      return test1 + test2;
    }
    function insideScope() {
      return
'owijfiojwefiojewijewoijewoijofiejioejffwiejijiefwiowefjoiwejiewjfiewfoiwejewifjwefijewfiojoewfjwefiowejfiewfjiewfjiewfj
ieowfjewfijewiiewfjweiojewfiojewfioejfiewfjiewfo' +
'owijfiojwefiojewijewoijewoijofiejioejffwiejijiefwiowefjoiwejiewjfiewfoiwejewifjwefijewfiojoewfjwefiowejfiewfjiewfjiewfj
ieowfjewfijewiiewfjweiojewfiojewfioejfiewfjiewfo';
    }
  };
</script>
```

Test runner

Ready to run.

Run tests

Testing in Chrome 34.0.1847.131 on OS X 10.9.2		
Test		**Ops/sec**
Inside function scope	`insideScope();`	ready
Outside function scope	`outsideScope();`	ready

```
$ d8 concat.js
Inside  x 198,893,290 ops/sec ±1.08%
Outside x 674,248,118 ops/sec ±1.08%
Fastest is Outside
```

«quick! move all your strings out of functions»

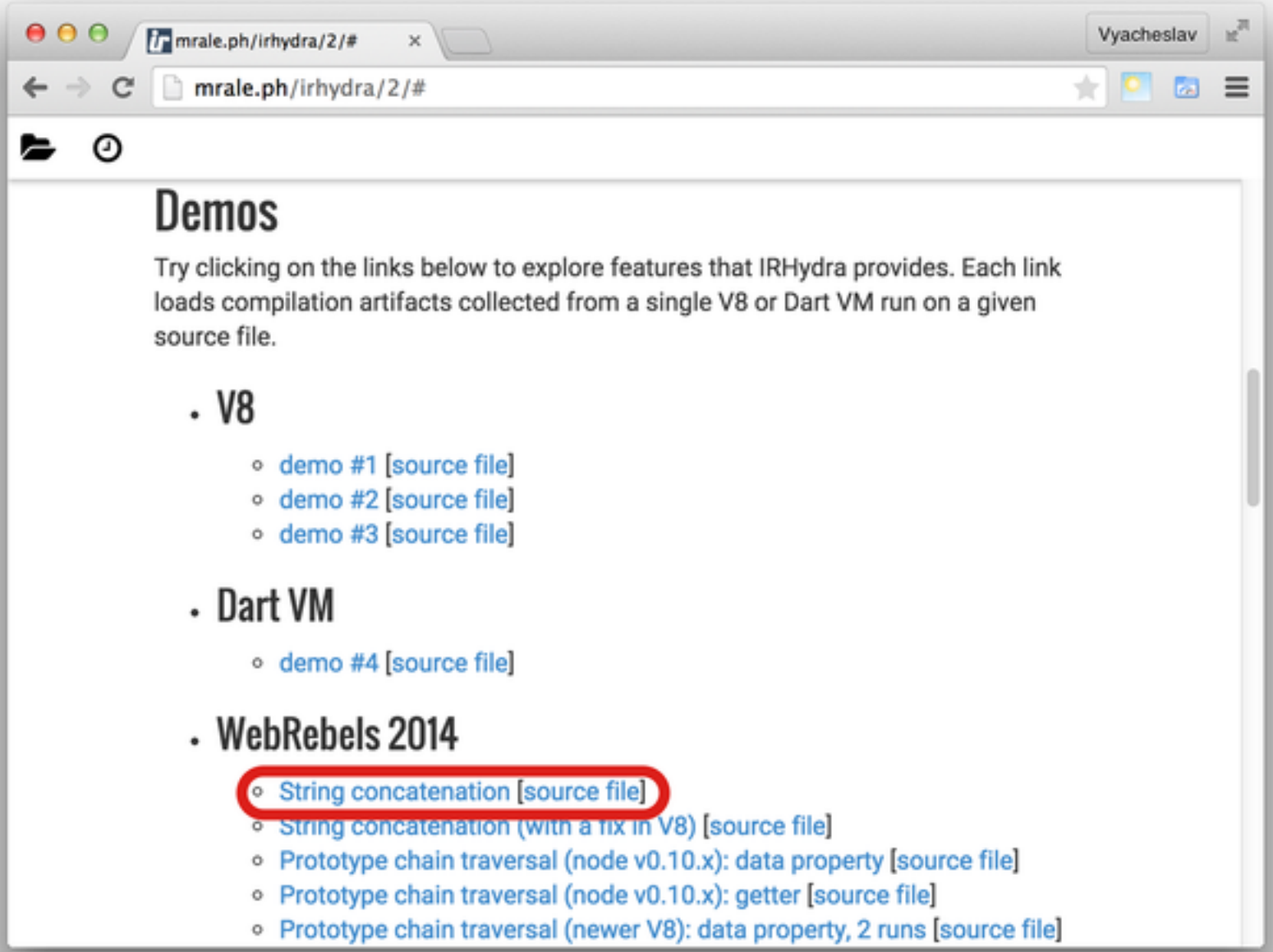
time to look at the code

we expect both variants to be DCEd

IRHydra^2

all samples from this talk
are demos within the tool

why `Outside` is fast(er)?


mrale.ph/irhydra/2/#
Vyacheslav
mrale.ph/irhydra/2/#
Demos
Try clicking on the links below to explore features that IRHydra provides. Each link loads compilation artifacts collected from a single V8 or Dart VM run on a given source file.
V8
demo #1 [source file]
demo #2 [source file]
demo #3 [source file]
Dart VM
demo #4 [source file]
WebRebels 2014
String concatenation [source file]
String concatenation (with a fix in V8) [source file]
Prototype chain traversal (node v0.10.x): data property [source file]
Prototype chain traversal (node v0.10.x): getter [source file]
Prototype chain traversal (newer V8): data property, 2 runs [source file]


Benchmark
Outside
result
outsideScope
Benchmark
Outside
outsideScope
Benchmark
Outside
String
outsideScope
OPTIMIZED
FUNCTIONS
select method

IR GRAPH SOURCE Hide Disassembly deopt @v164

Benchmark
Outside

result

outsideScope

Benchmark
Outside

outsideScope

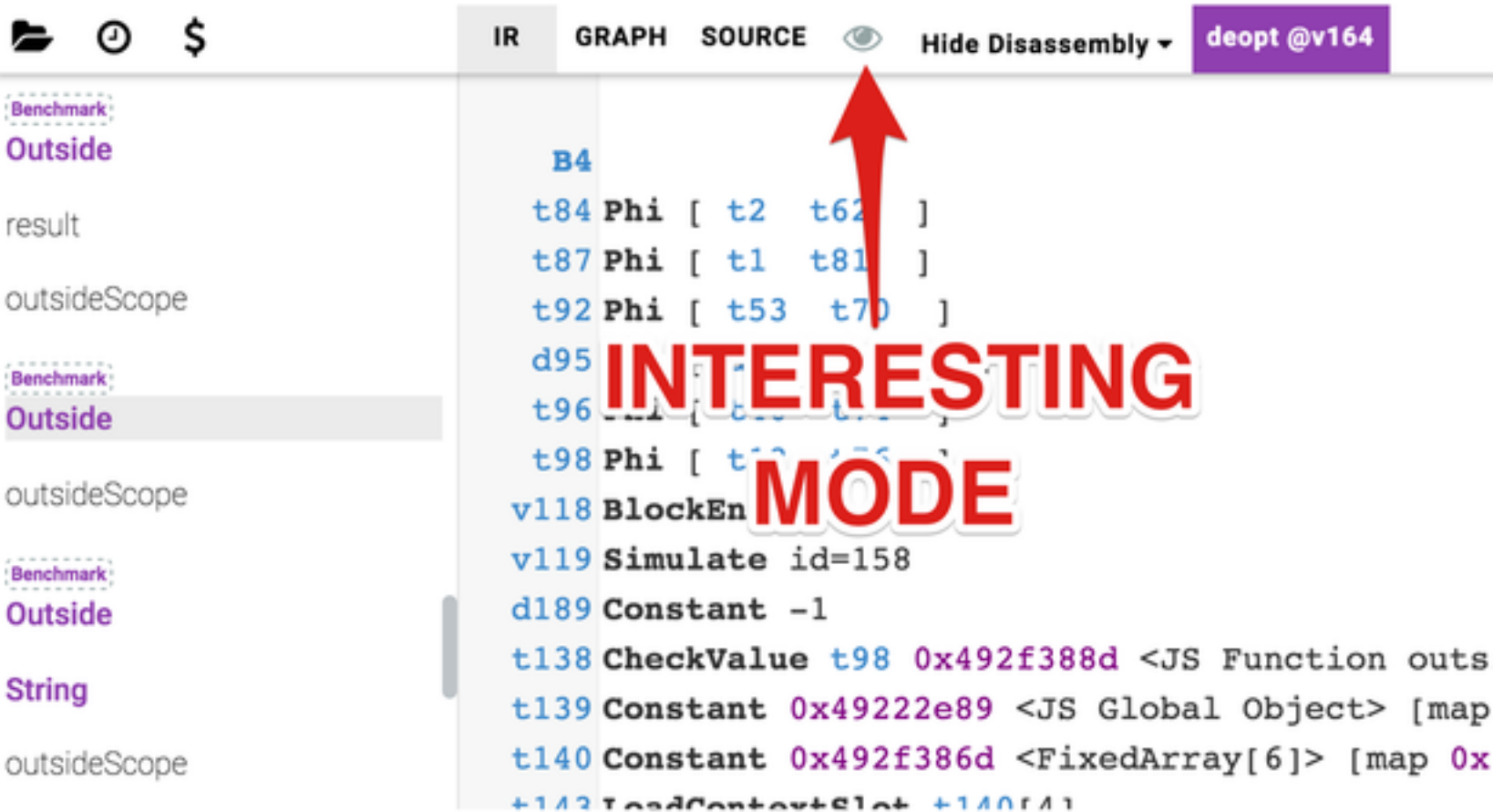
Benchmark
Outside

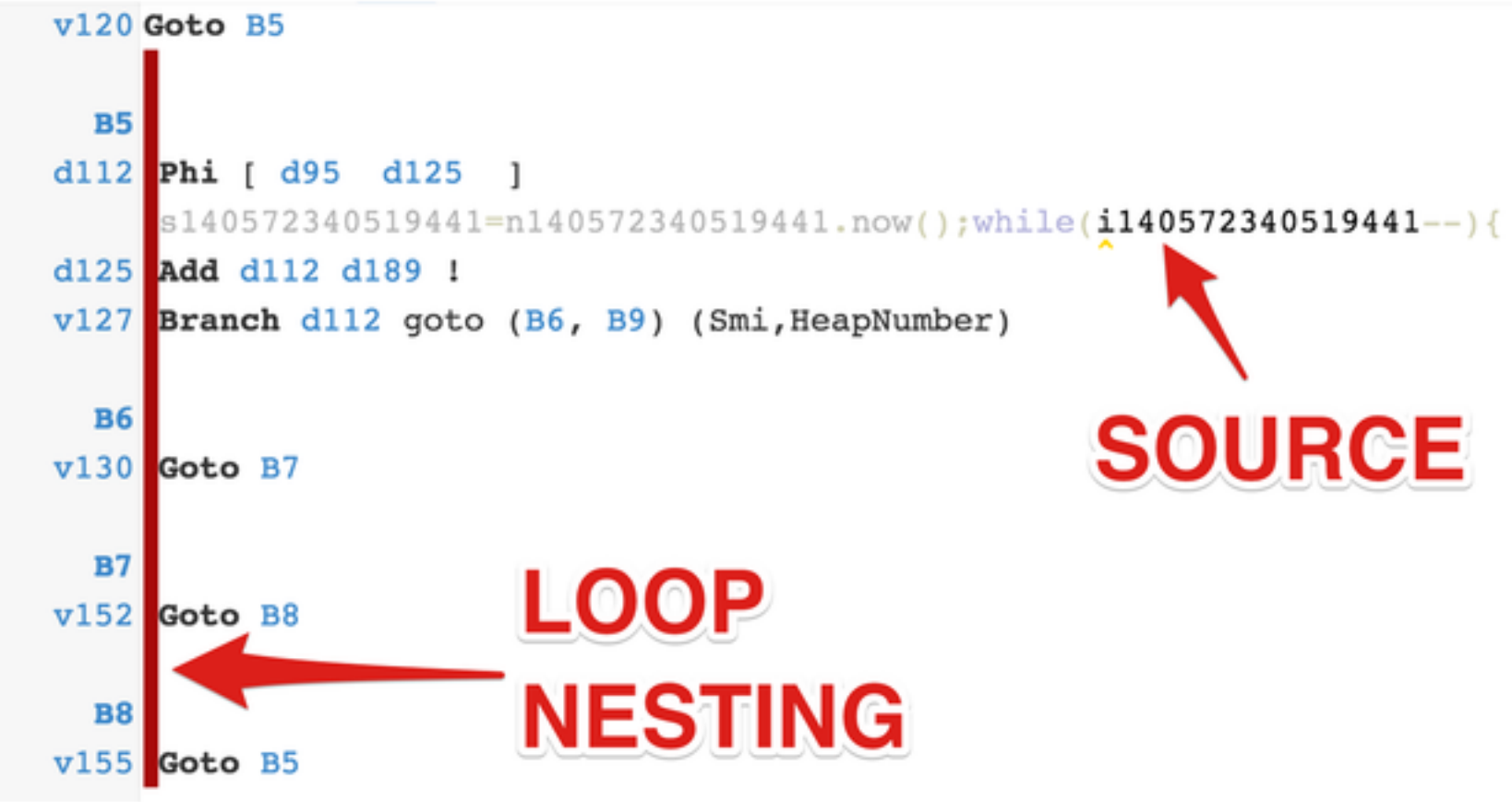
String

outsideScope

```
  B4
 t84 Phi [ t2  t62  ]
 t87 Phi [ t1  t81  ]
 t92 Phi [ t53  t70  ]
 d95 Phi [ d190  d191  ]
 t96 Phi [ t40  t74  ]
 t98 Phi [ t13  t76  ]
v118 BlockEntry
v119 Simulate id=158
d189 Constant -1
t138 CheckValue t98 0x492f388d <JS Function outs
t139 Constant 0x49222e89 <JS Global Object> [map
t140 Constant 0x492f386d <FixedArray[6]> [map 0x
t143 LoadContextSlot t140[4]
```

the forest hidden behind the trees


IR
GRAPH
SOURCE
Hide Disassembly
deopt @v164
Benchmark
Outside
result
outsideScope
Benchmark
Outside
outsideScope
Benchmark
Outside
String
outsideScope
B4
t84 Phi [t2 t62]
t87 Phi [t1 t81]
t92 Phi [t53 t70]
d95
t96
t98 Phi [
v118 BlockEn
v119 Simulate id=158
d189 Constant -1
t138 CheckValue t98 0x492f388d <JS Function outs
t139 Constant 0x49222e89 <JS Global Object> [map
t140 Constant 0x492f386d <FixedArray[6]> [map 0x
INTERESTING
MODE


v120 Goto B5
B5
d112 Phi [d95 d125]
s140572340519441=n140572340519441.now();while(i140572340519441--){
d125 Add d112 d189 !
v127 Branch d112 goto (B6, B9) (Smi,HeapNumber)
B6
v130 Goto B7
B7
v152 Goto B8
B8
v155 Goto B5
SOURCE
LOOP
NESTING

```
v120 Goto B5

  B5
d112 Phi [ d95  d125  ]
     s140572340519441=n140572340519441.now();while(i140572340519441--){
d125 Add d112 d189 !
v127 Branch d112 goto (B6, B9) (Smi,HeapNumber)

  B6
v130 Goto B7

  B7
v152 Goto B8

  B8
v155 Goto B5
```

`Outside` is fast

at *doing nothing*

V8 failed to constant fold

[bug!]

lets fix it!

```
--- src/hydrogen.cc (revision 21348)
+++ src/hydrogen.cc (working copy)
@@ -9639,6 +9639,14 @@
       return left;
     }

+    if (FLAG_fold_constants &&
+        left->IsConstant() && HConstant::cast(left)->HasStringValue() &&
+        right->IsConstant() && HConstant::cast(right)->HasStringValue()) {
+      return AddUncasted<HStringAdd>(
+          left, right, allocation_mode.GetPretenureMode(),
+          STRING_ADD_CHECK_NONE, allocation_mode.feedback_site());
+    }
+
     // Register the dependent code with the allocation site.
     if (!allocation_mode.feedback_site().is_null()) {
       ASSERT(!graph()->info()->IsStub());
```

```
$ d8 concat.js
Inside  x 758,530,478 ops/sec ±1.82%
Outside x 674,248,118 ops/sec ±1.14%
Fastest is Inside
```

both are now doing nothing!

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({prop: 10}))))));
                  // LISP tribute ^^^^^
```

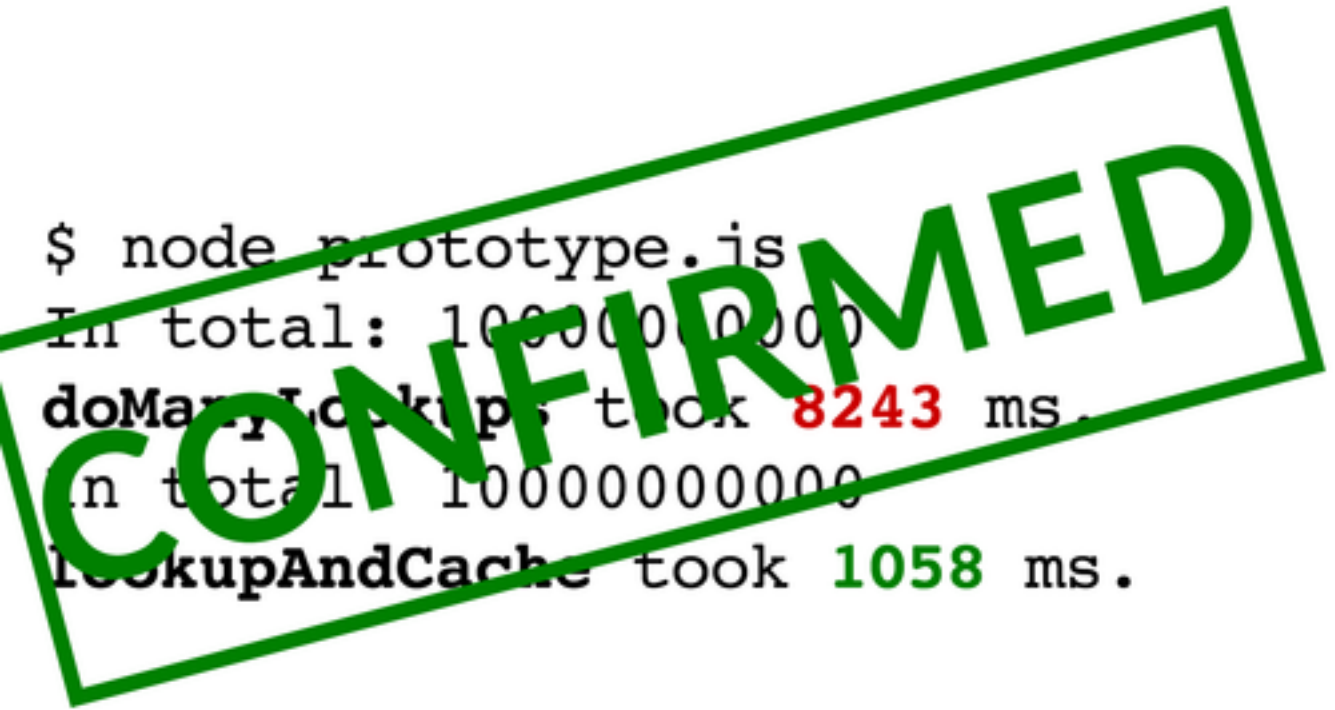
```
function doManyLookups() {
  var counter = 0;
  for(var i = 0; i < 1000; i++)
    for(var j = 0; j < 1000; j++)
      for(var k = 0; k < 1000; k++)
        counter += obj.prop;
  print('In total: ' + counter);
}
```

```
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: 10000000000
doManyLookups took 8243 ms.
In total: 10000000000
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    Constant 10  range[10,10,m0=0]                483647,1000,m0=0]
v113 Simulate id=86 pop 2 / var[2] = d110, var
v114 Goto B11
```

```
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=0
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

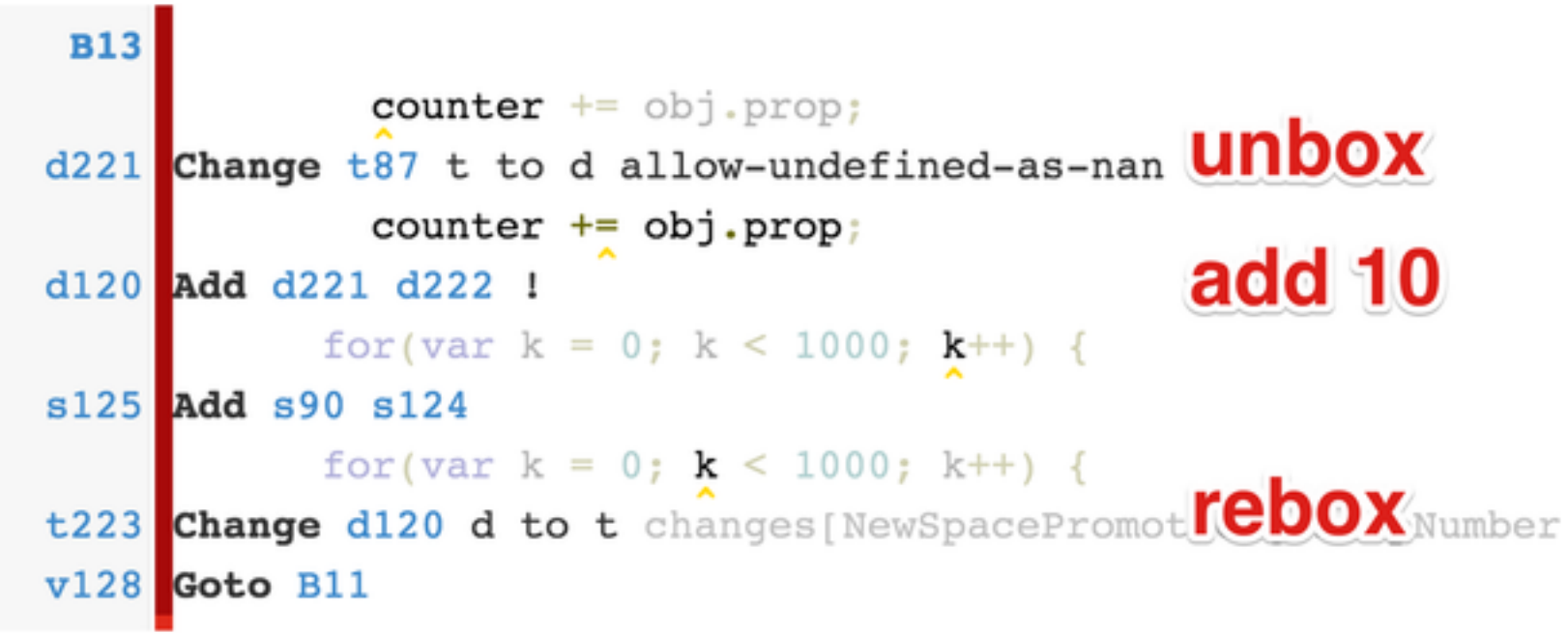
what if we run benchmark twice?

```
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
unbox
counter += obj.prop;
d120 Add d221 d222 !
add 10
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
rebox
Number
v128 Goto B11

`'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.
```

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

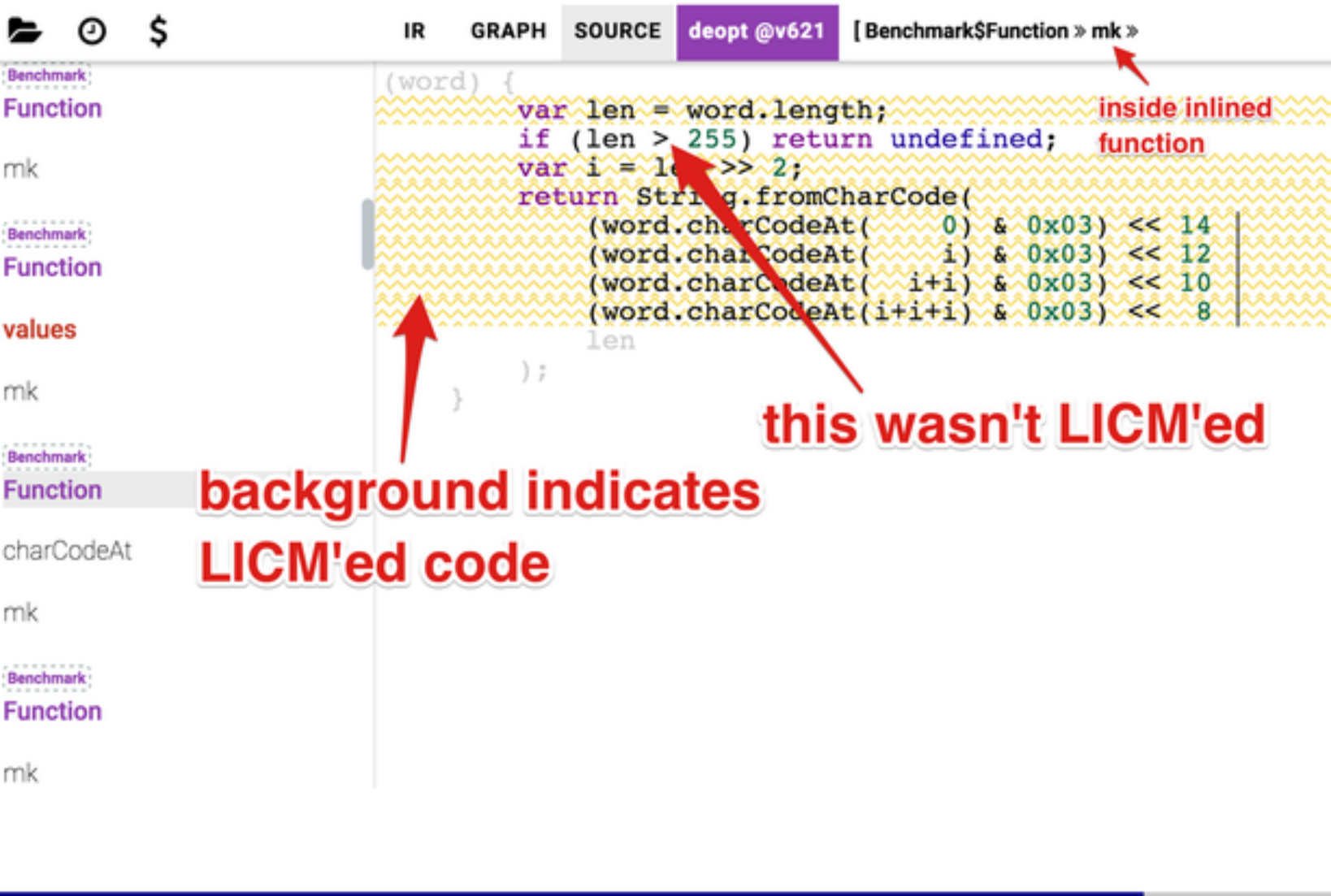
— function call does not go through an IC, but method call does

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

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

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;
inlining marker
var key;
key = mk('www.wired.com');
key = mk('www.youtube.com');
key = mk('scorecardresearch.com');
key = mk('www.google-analytics.com');
// no operation performed


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.char odeAt(i) & 0x03) << 12
(word.charCdeAt(i+i) & 0x03) << 10
(word.charCodeAt(i+i+i) & 0x03) << 8
len
);
}
inside inlined function
this wasn't LICM'ed
background indicates LICM'ed code

measuring *almost*
empty loop again!

can function call be faster than method call?

```
// for (var item in list[i]) { ... }
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[i]) { ... }
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[i]) { ... }
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[i]) { ... }
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

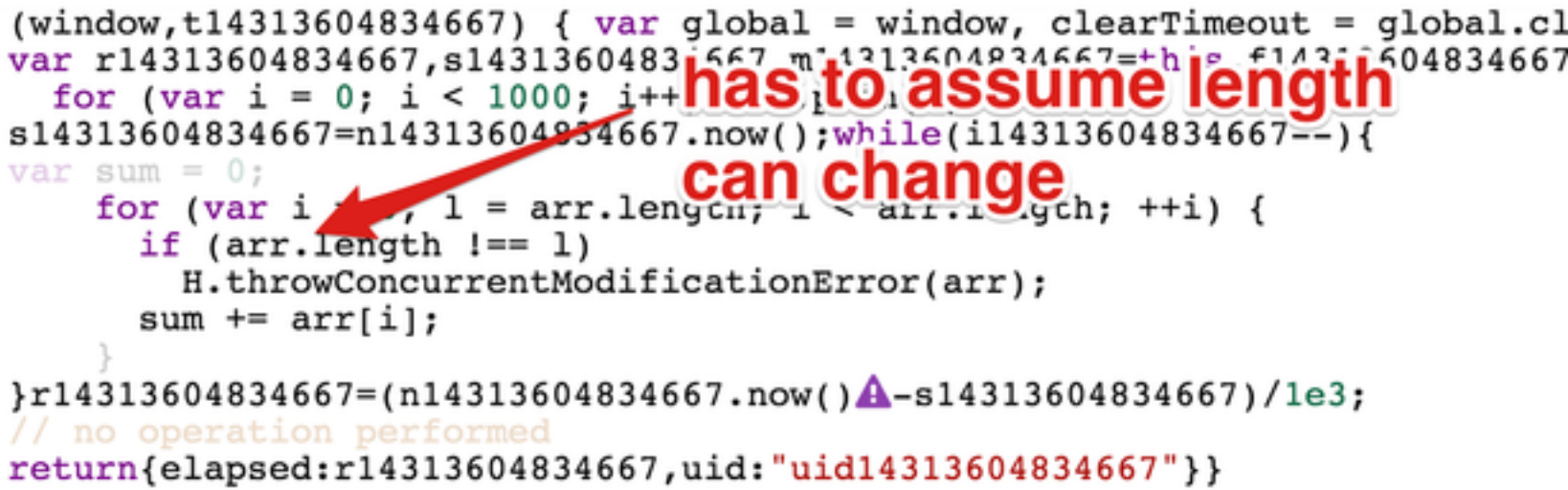
`(0,o.f)()`

in the code that never executes

```
(window,t143136048346667) { var global = window, clearTimeout = global.cl
var r143136048346667,s143136048346667,m143136048346667=this,f14313604834667
  for (var i = 0; i < 1000; i++) arr.push(i);
s143136048346667=n143136048346667.now();while(i143136048346667--){
var sum = 0;
    for (var i = 0, l = arr.length; i < arr.length; ++i) {
      if (arr.length !== l)
        H.throwConcurrentModificationError(arr);
      sum +=arr[i];
    }
}r143136048346667=(n14[illegible]34667)/1e3;
// no operation performed
return{elapsed:r143136[illegible]
```

never executed and

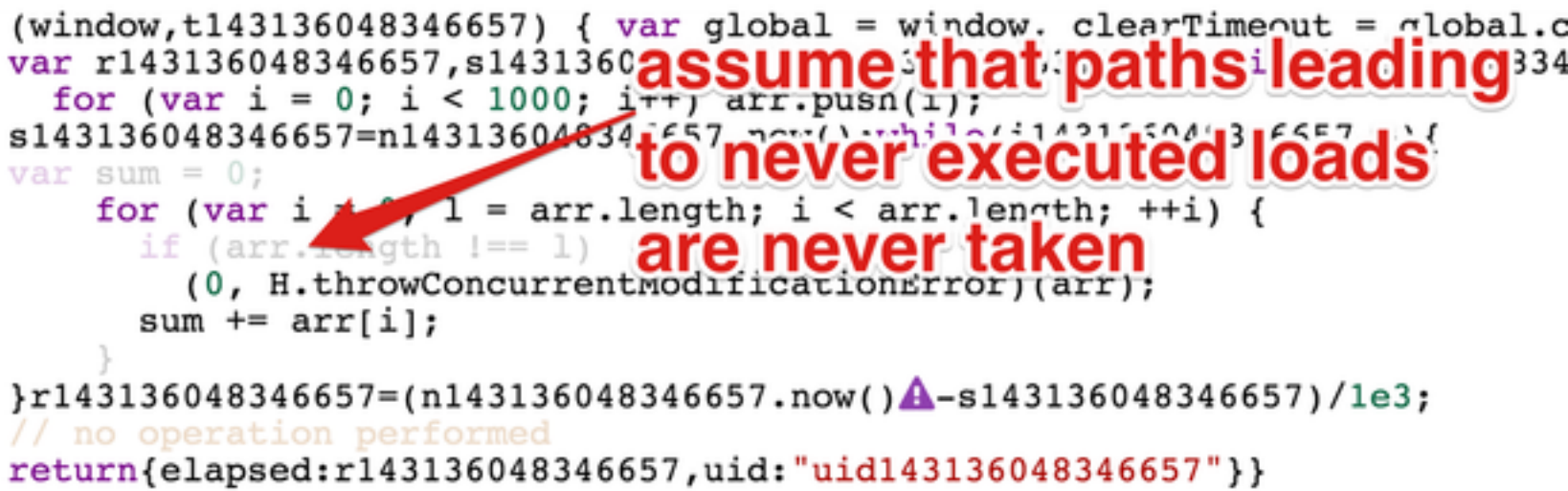
V8 doesn't know where it goes


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

```
(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 perfor
return{elapsed:r143136
```

also never executed

but property loads are special


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=n1431360483
hoisted
;while(i143136048346657--){
var sum = 0;
for (var i = 0, l = arr.length; i < arr.length; ++i) {
if (arr.length !== l)
(0, H.throwCon
rentModificationError)(arr);
sum += arr[i];
}
folded away
}r143136048346657=(n143136048346657.now()-s143136048346657)/1e3;
// no operation performed
return{elapsed:r143136048346657,uid:"uid143136048346657"}}

algorithms first

μbenchmarks last

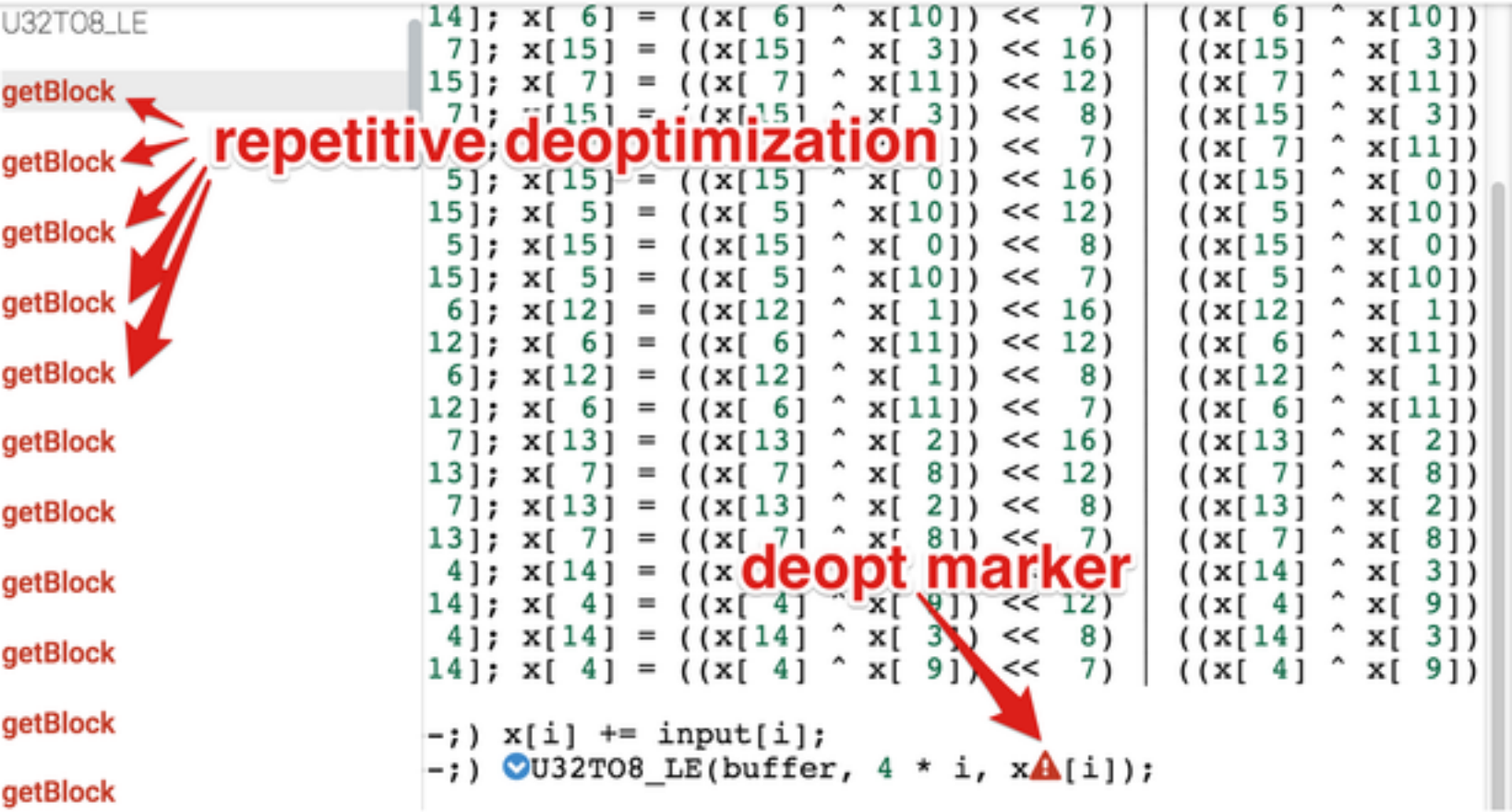
THANK YOU

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
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])
12]; x[7] = ((x[7] ^ x[11]) << 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 | ((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]);

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

talk to VM people

report bugs