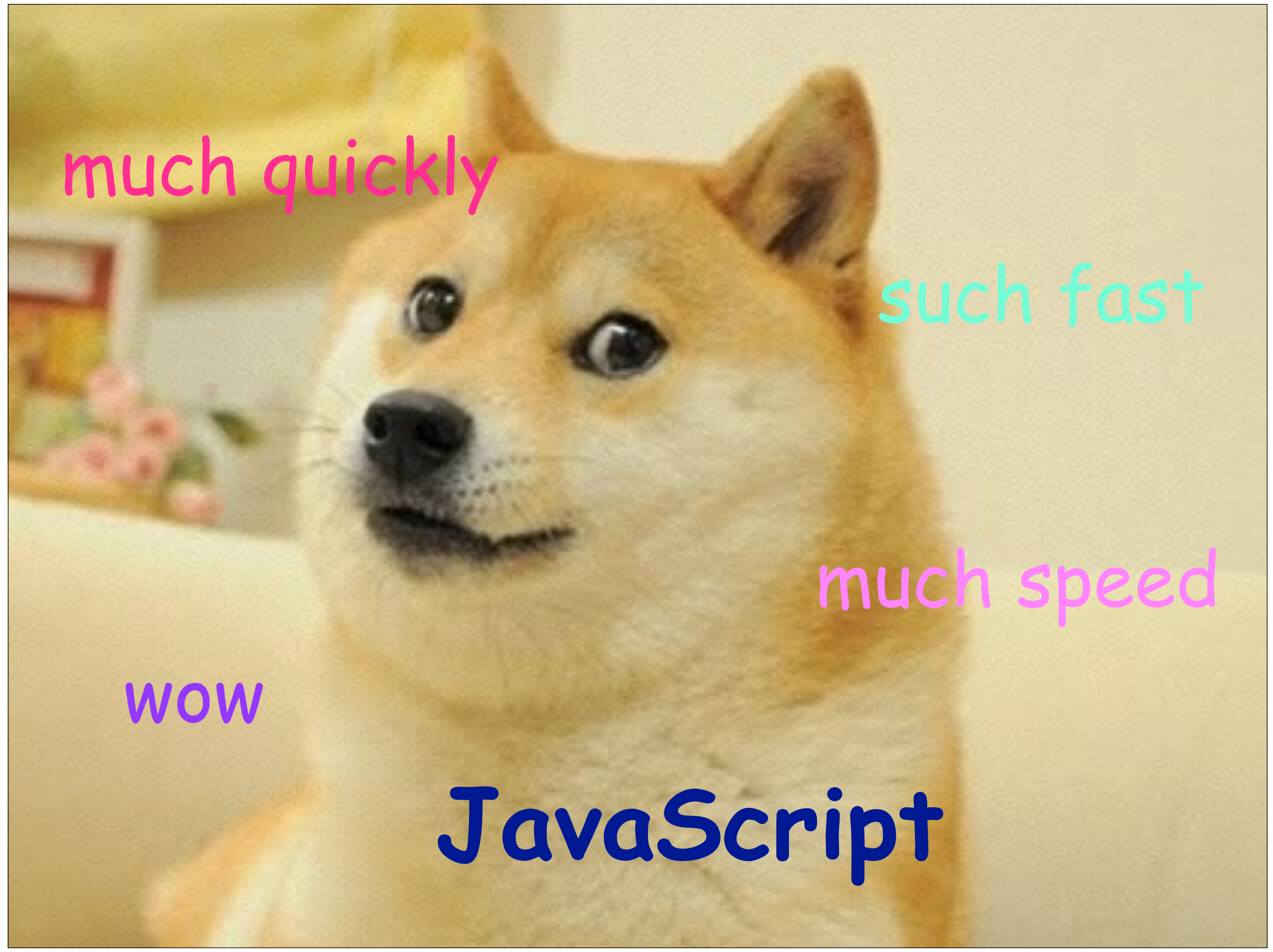
Performance Profiling for V8

Dr. Franziska Hinkelmann, Google V8

Good morning. My name is Franziska Hinkelmann. I’m a software engineer on the V8 team. and I’ll talk about profiling V8.



I think we can all agree that JavaScript is incredibly powerful. Not only in WHAT in can do, but also how fast it can do it. It's amazing, that with JavaScript, a scripting language, you can run enterprise node servers and large websites like Facebook or youtube. And this performance of course comes from the JavaScript engines. So in this talk, we'll look at what V8 does to be this fast and how to profile these optimizations.

- Browser: ChakraCore, JSC, Spidermonkey, V8
- Node.js: ChakraCore, V8
- Electron: V8
- IoT: Duktape, JerryScript

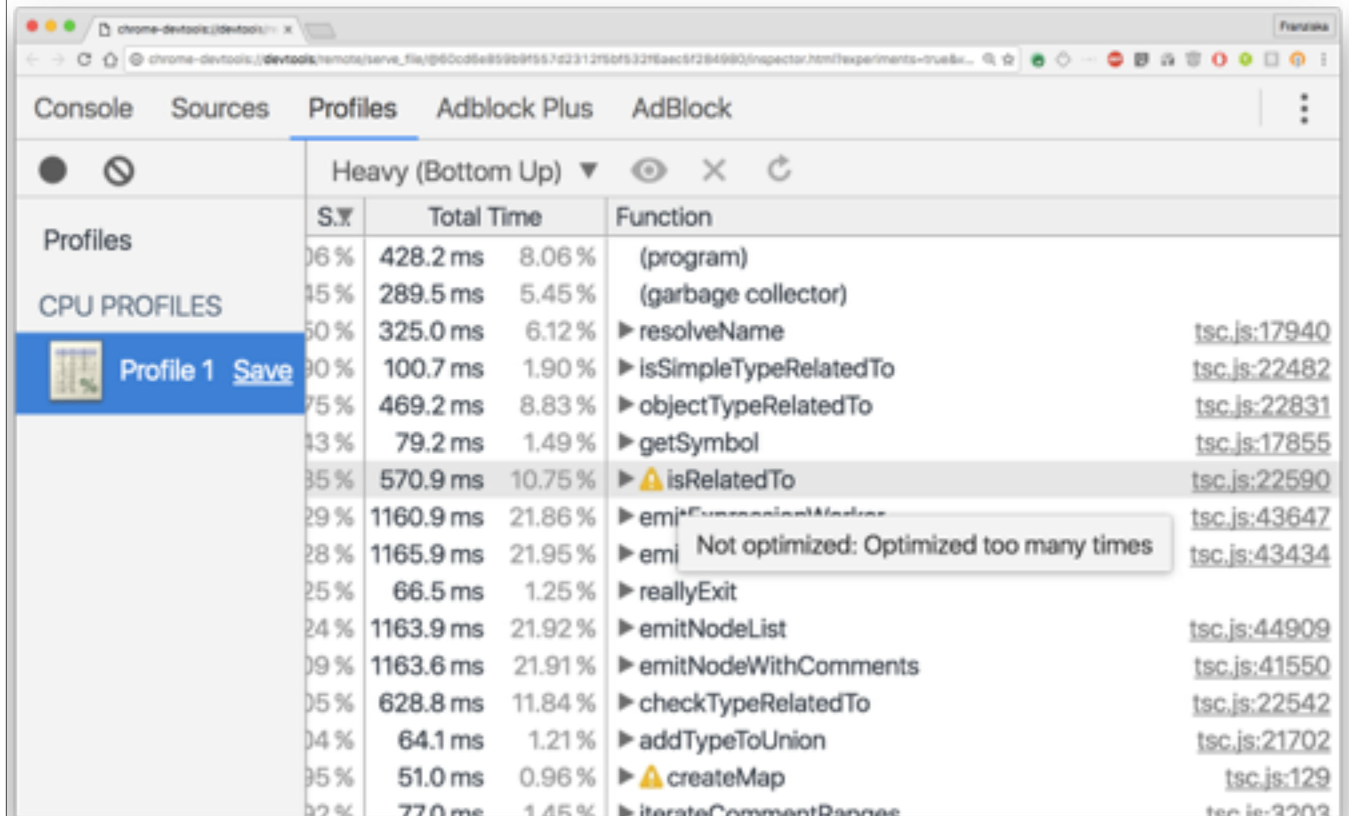
to put v8 in context.

Profiling V8

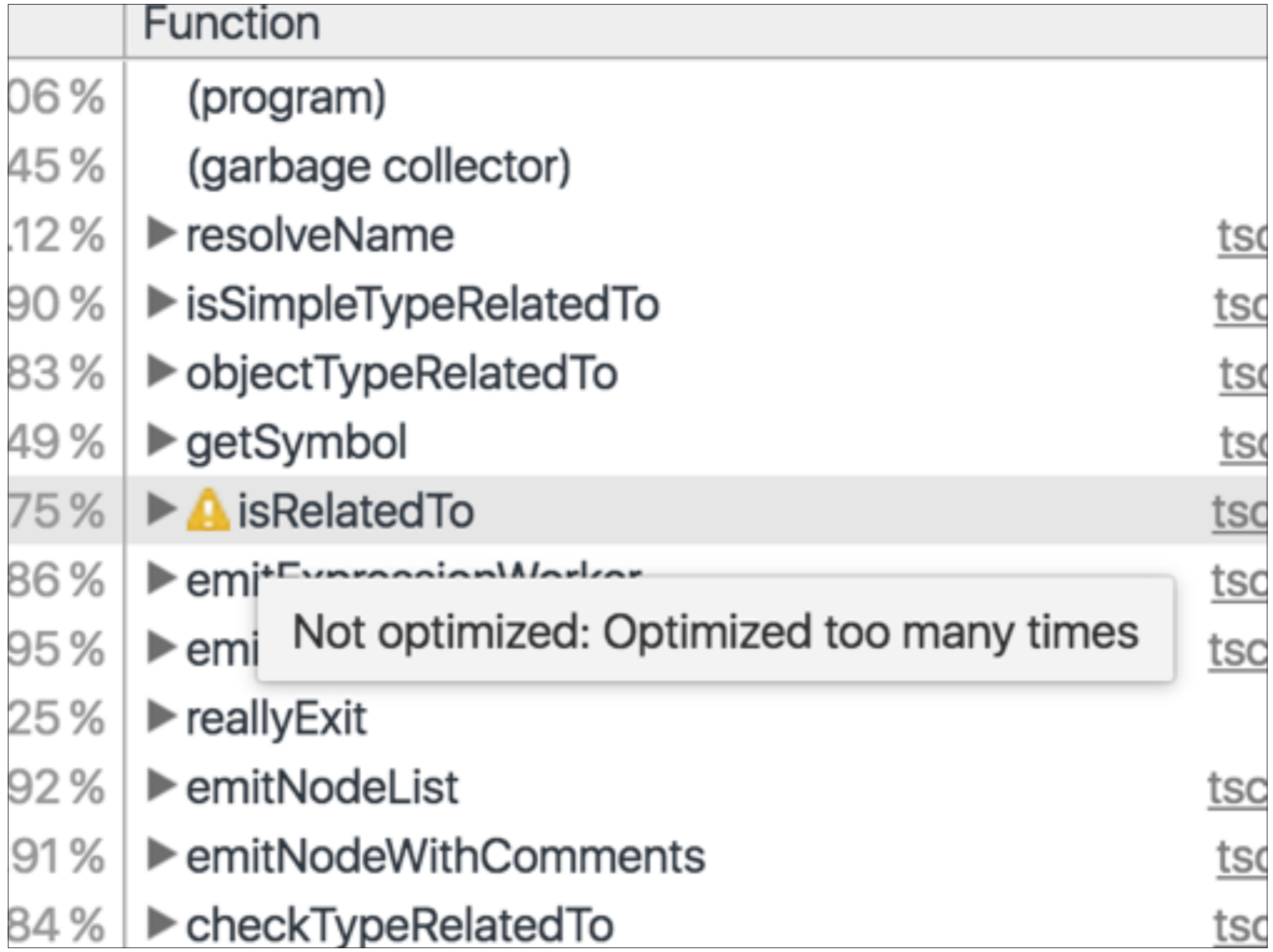
- Just in time (JIT) compilation
- Inline Caches (IC)
- Optimizing compiler
- Machine code

In order to talk successfully about profiling, we have to look at some key concepts in V8.
I'm hoping to give you some insight into the inner workings of V8 and some tools to profile, or TRACE, what's going on in the VM. These tools are what we use to understand and improve performance internally in V8 (and make changes in V8) - but you can also use them to understand better how V8 is dealing with your code. And maybe improve performance.

Chrome DevTools CPU Profile



JavaScript CPU Profile
Heap snapshot, timeline, profile


Function
06 %
(program)
45 %
(garbage collector)
12 %
resolveName
90 %
isSimpleTypeRelatedTo
83 %
objectTypeRelatedTo
49 %
getSymbol
75 %
isRelatedTo
86 %
95 %
Not optimized: Optimized too many times
25 %
reallyExit
92 %
emitNodeList
91 %
emitNodeWithComments
84 %
checkTypeRelatedTo

JS is dynamically typed

- Not statically typed (Like C++, Java, Rust).

```
var obj = {
      x: 1,
      y: 1
};
delete obj.x;
obj.z = 1;
```

Properties?

- Type information only available at runtime.

Just In Time (JIT) Compilation

Generate machine code during runtime, not **ahead of time** (AOT).

Property Access

```
function load(obj) {
  return obj.x;
}
```

- TypeError
- undefined
- prototype chain
- proxy
- side effects if accessor

console.log, array.prototype.length
proxy: get trap

9.1.8.1 OrdinaryGet (*O*, *P*, *Receiver*)

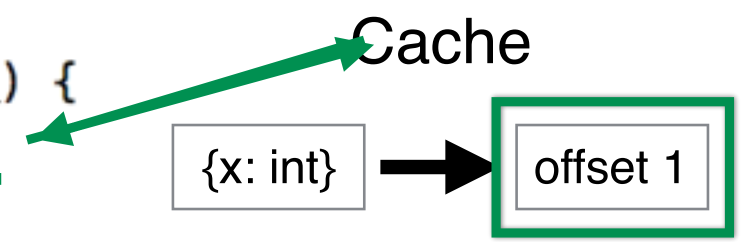
When the abstract operation OrdinaryGet is called with Object *O*, proper ECMAScript language value *Receiver*, the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**.
2. Let *desc* be ? *O*.[[GetOwnProperty]](*P*).
3. If *desc* is **undefined**, then
    a. Let *parent* be ? *O*.[[GetPrototypeOf]]().
    b. If *parent* is **null**, return **undefined**.
    c. Return ? *parent*.[[Get]](*P*, *Receiver*).
4. If IsDataDescriptor(*desc*) is **true**, return *desc*.[[Value]].
5. Assert: IsAccessorDescriptor(*desc*) is **true**.
6. Let *getter* be *desc*.[[Get]].
7. If *getter* is **undefined**, return **undefined**.
8. Return ? Call(*getter*, *Receiver*).

EcmaScript specification

```
obj.x
```

```
function load(obj) {
    return obj.x;
}
```




```
load({x: 5});

load({x: 17});
```

9.1.8.1 Ord[illegible]ryGet (O, P, Receiver)

When the a[illegible]ct operation OrdinaryGet is called with Object O, property key P, and ECMAScript la[illegible]ge value Receiver, the following steps are taken:

1. Assert: IsProperty[illegible](P) is true.
2. Let desc be ? O.[[GetO[illegible]Property]](P).
3. If desc is undefined, then
    a. Let parent be ? O.[[GetPr[illegible]eOf]]().
    b. If parent is null, return undefi[illegible]
    c. Return ? parent.[[Get]](P, Receiver).
4. If IsDataDescriptor(desc) is true, return desc.[illegible]alue]].
5. Assert: IsAccessorDescriptor(desc) is true.
6. Let getter be desc.[[Get]].
7. If getter is undefined, return undefined.
8. Return ? Call(getter, Receiver).

Inline Cache (IC)

```
function load(obj) {
    if(cond) {
        return obj.x;
    } else {
        return obj.x + 1;
    }
}
```

Shape of object = map = hidden class

{shape of object} → {fast path to property}

{x: int} → offset 1

Optimizing compiler

- Modern engines have optimizing compilers
- Basic compiler runs first and collects information, “hot functions” are then compiled by optimizing compiler

Optimization + IC

=

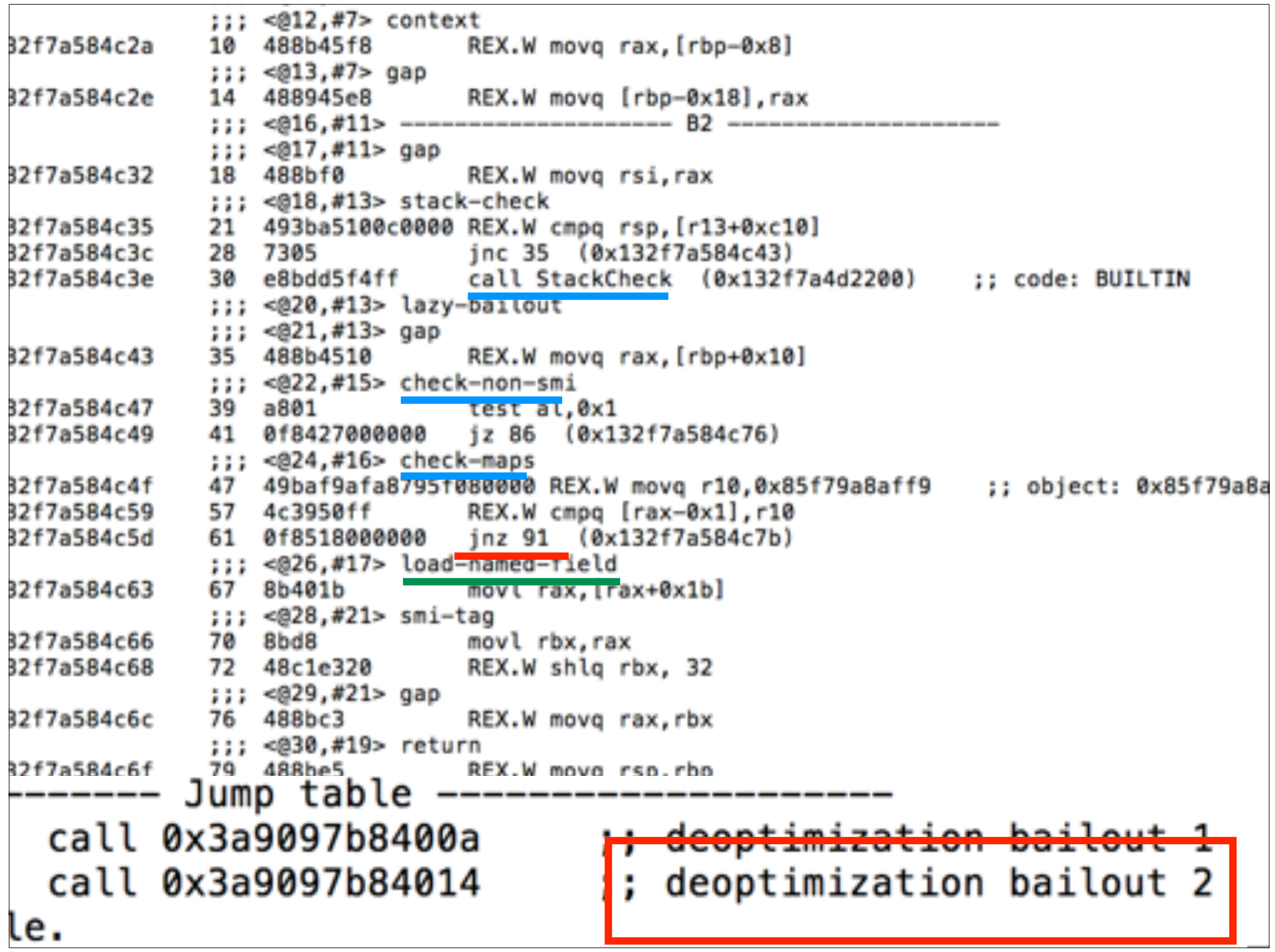
Speed

Optimized Machine Code

```
function load(obj) {
    return obj.x;
}
```

```
$ d8 --allow-natives-syntax —trace-opt —print-opt-code —code-comments load-opt.js
 [compiling method 0x9508e1f30c1 <JS Function load (SharedFunctionInfo 0xc3433e59a11)> using Crankshaft]
[optimizing 0x9508e1f30c1 <JS Function load (SharedFunctionInfo 0xc3433e59a11)> - took 5.019, 0.103,
0.089 ms]
```

If I have lost you, NOW is a good time to get back in. Because we're going to look at some actual machine code. Ready!?



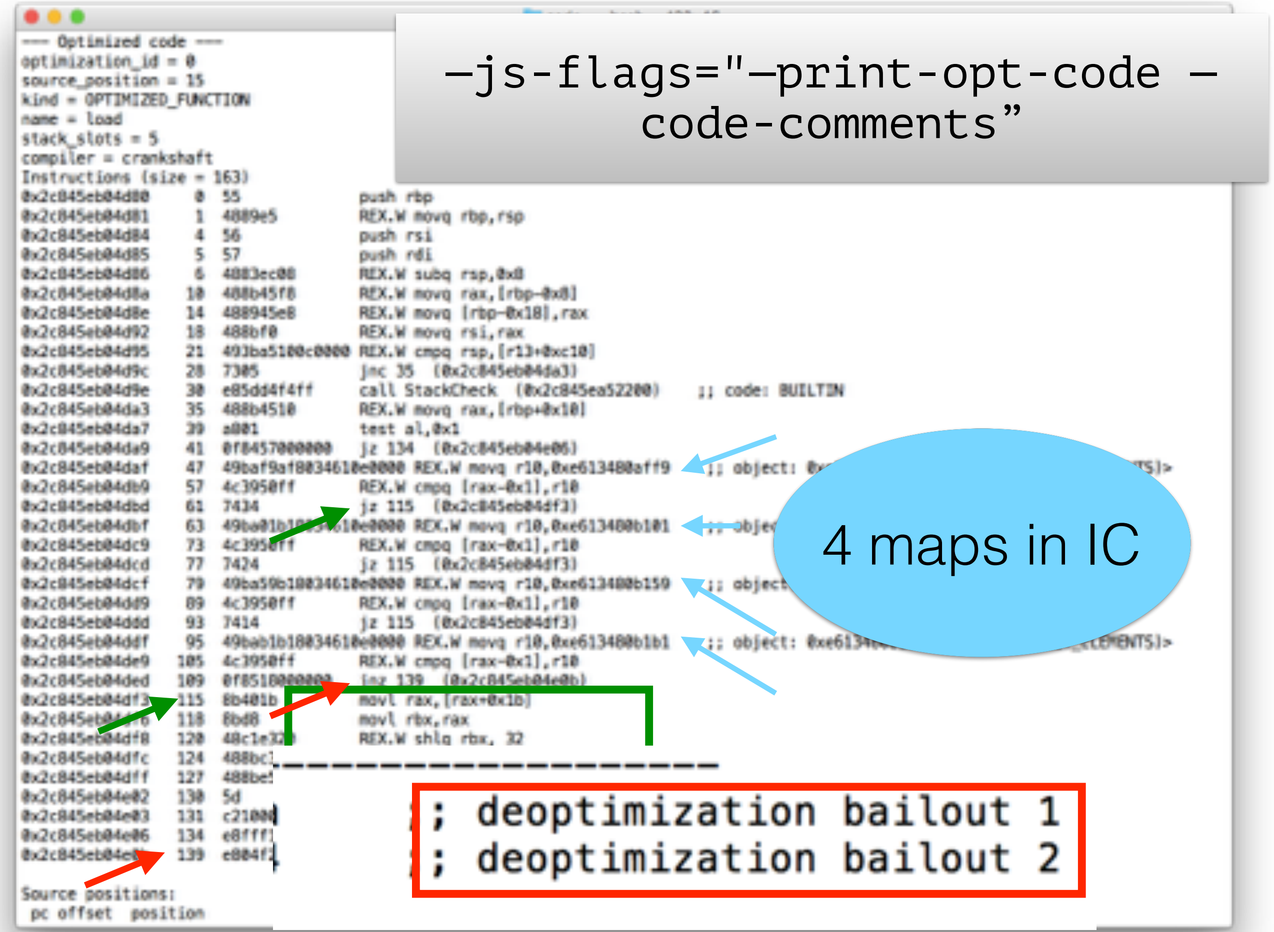
I think it's so exciting, that you can you can see and understand what's going on on the lowest level of your code. If this looks confusing, bare with me, I will explain. Long story short: JavaScript engines compile your JS Source down to machine code.

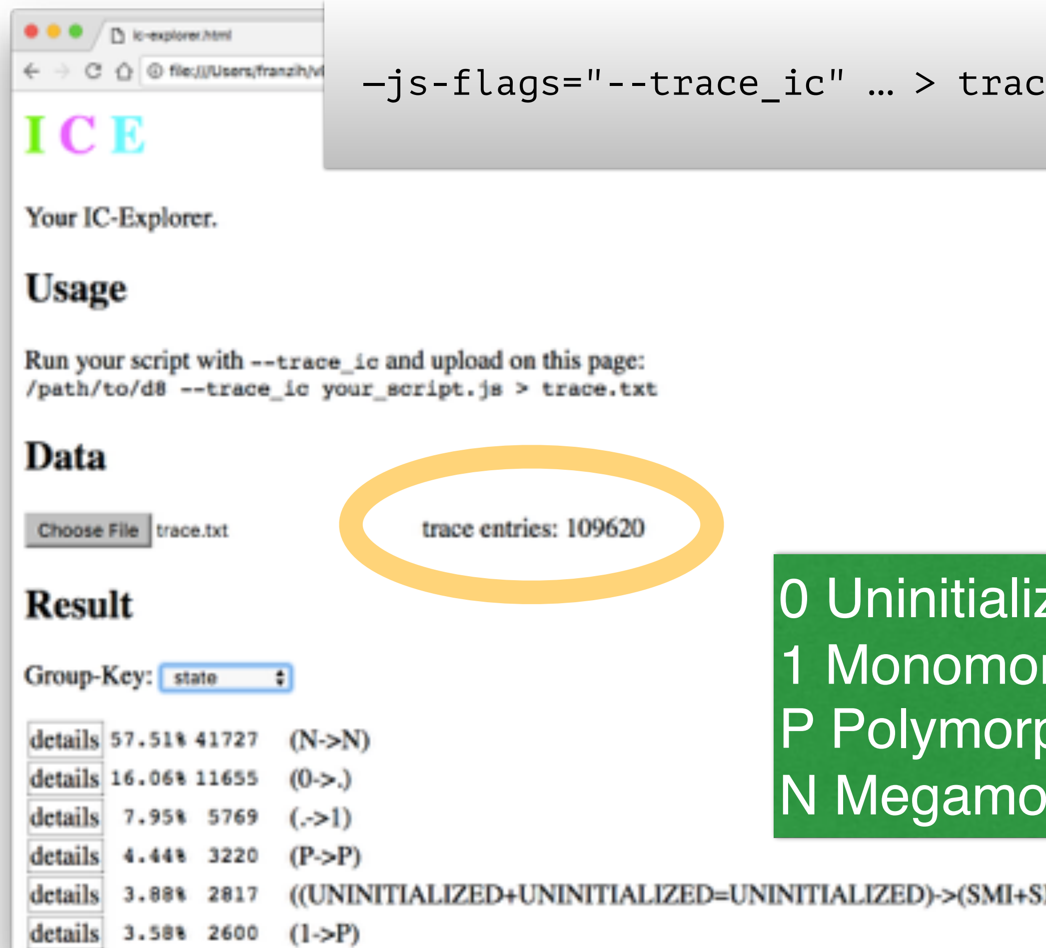
IC States

- Uninitialized
- Monomorphic: 1 map
- Polymorphic: 2-4 maps
- Megamorphic: more than 4 maps

```
               ;;; <@12,#7> context
32f7a584c2a    10  488b45f8       REX.W movq rax,[rbp-0x8]
               ;;; <@13,#7> gap
32f7a584c2e    14  488945e8       REX.W movq [rbp-0x18],rax
               ;;; <@16,#11> -------------------- B2 --------------------
               ;;; <@17,#11> gap
32f7a584c32    18  488bf0         REX.W movq rsi,rax
               ;;; <@18,#13> stack-check
32f7a584c35    21  493ba5100c0000 REX.W cmpq rsp,[r13+0xc10]
32f7a584c3c    28  7305           jnc 35  (0x132f7a584c43)
32f7a584c3e    30  e8bdd5f4ff     call StackCheck  (0x132f7a4d2200)    ;; code: BUILTIN
               ;;; <@20,#13> lazy-bailout
               ;;; <@21,#13> gap
32f7a584c43    35  488b4510       REX.W movq rax,[rbp+0x10]
               ;;; <@22,#15> check-non-smi
32f7a584c47    39  a801           test al,0x1
32f7a584c49    41  0f8427000000   jz 86  (0x132f7a584c76)
               ;;; <@24,#16> check-maps
32f7a584c4f    47  49baf9afa8795f080000 REX.W movq r10,0x8            5f79a8a
32f7a584c59    57  4c3950ff       REX.W cmpq [rax-0x1],r10
32f7a584c5d    61  0f8518000000   jnz 91  (0x132f7a584c7b)
               ;;; <@26,#17> load-named-field
32f7a584c63    67  8b401b         movl rax,[rax+0x1b]
               ;;; <@28,#21> smi-tag
32f7a584c66    70  8bd8           movl rbx,rax
32f7a584c68    72  48c1e320       REX.W shlq rbx, 32
               ;;; <@29,#21> gap
32f7a584c6c    76  488bc3         REX.W movq rax,rbx
               ;;; <@30,#19> return
32f7a584c6f    79  488be5         REX.W movq rsp,rbp
32f7a584c72    82  5d             pop rbp
32f7a584c73    83  c21000         ret 0x10
               ;;; -------------------- Jump table --------------------
32f7a584c76    86  e88ff3d7ff     call 0x132f7a30400a     ;; deoptimization bailout 1
32f7a584c7b    91  e894f3d7ff     call 0x132f7a304014     ;; deoptimization bailout 2
               ;;; Safepoint table.
```

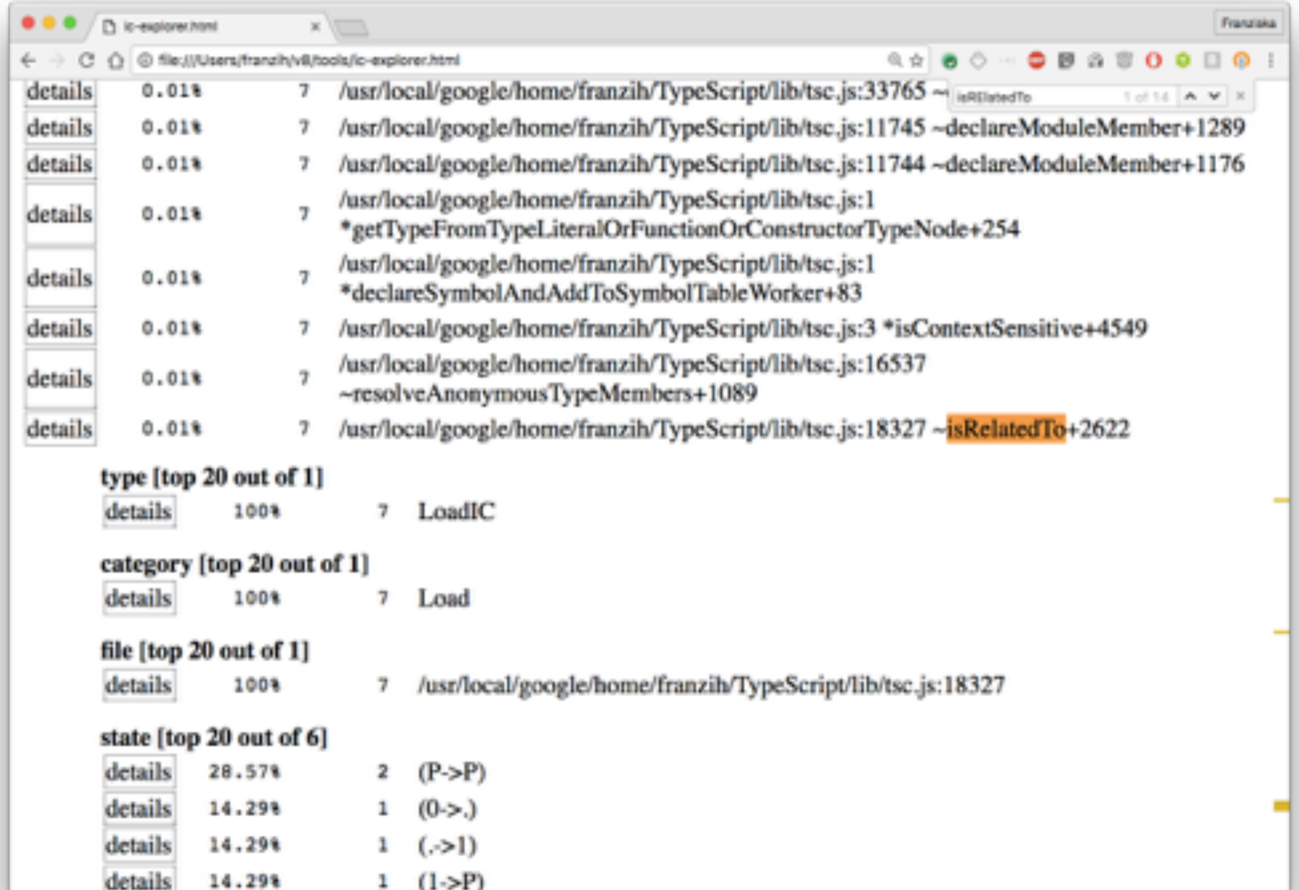
1 map in IC


–js-flags="–print-opt-code –code-comments”
4 maps in IC
; deoptimization bailout 1
; deoptimization bailout 2



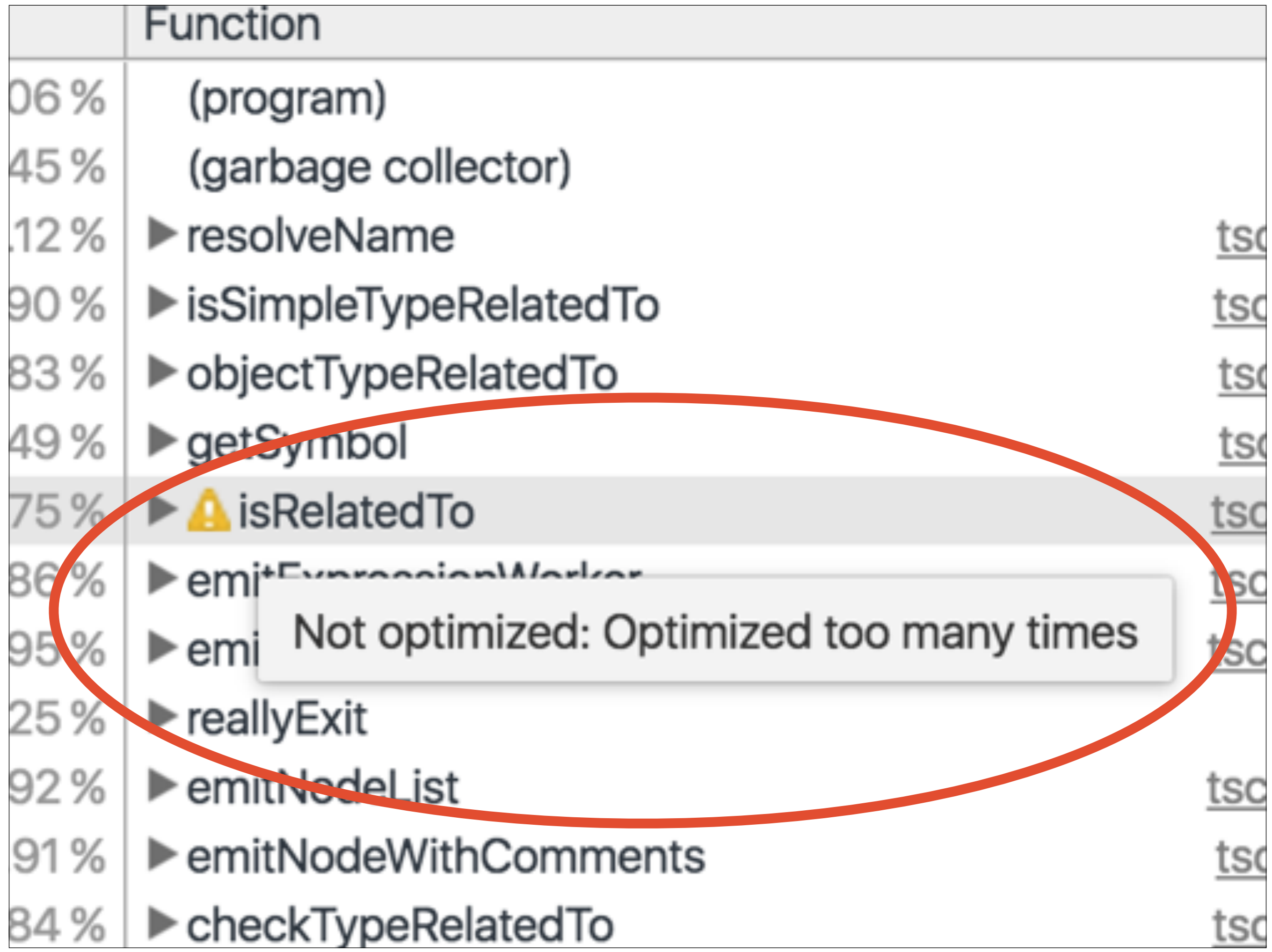

```
–js-flags="--trace_ic" … > trace.txt
```

0 Uninitialized
1 Monomorphic
P Polymorphic
N Megamorphic

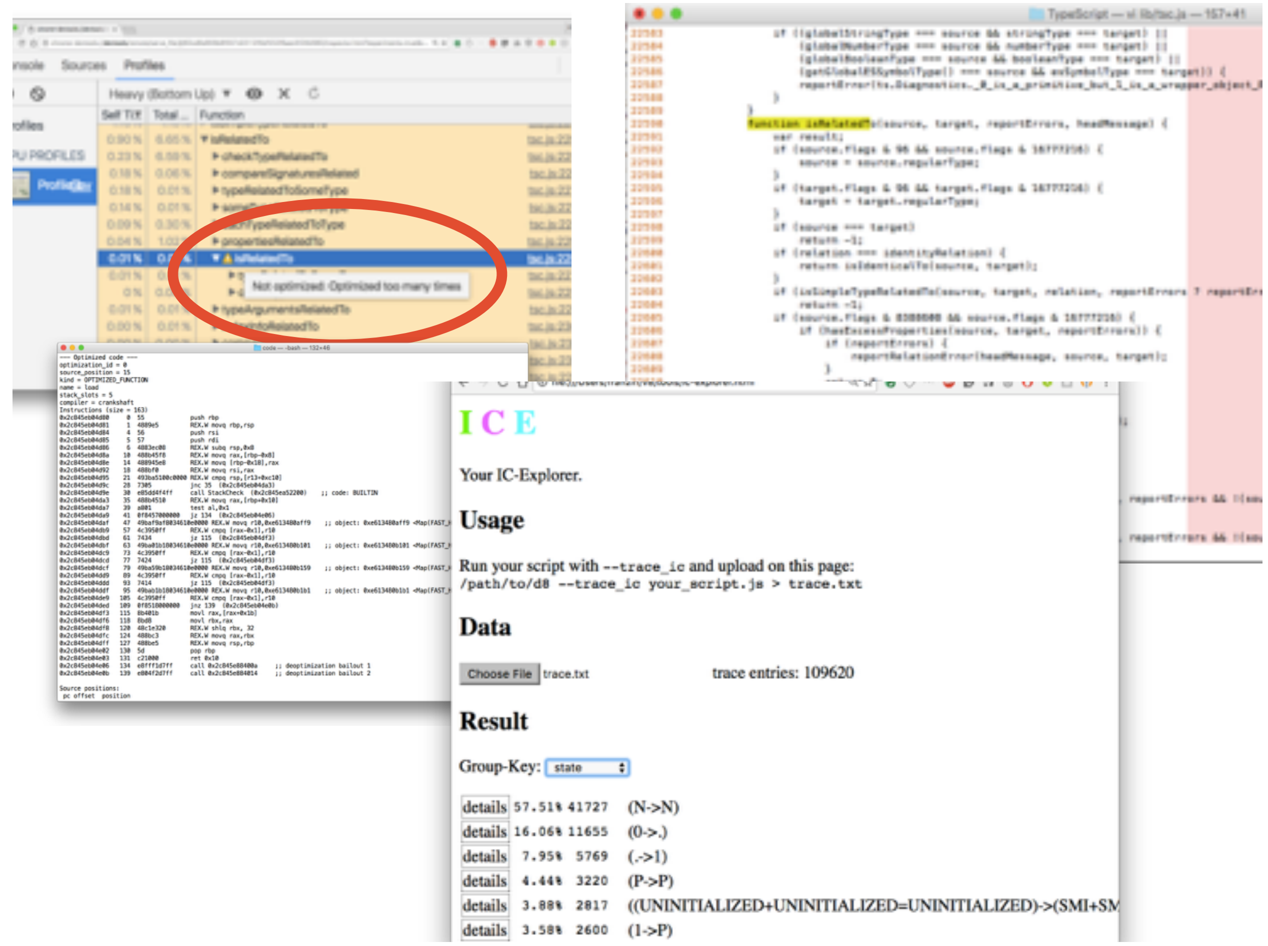

ic-explorer.html
file:///Users/franzih/v8/tools/ic-explorer.html
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:33765 ~
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:11745 ~declareModuleMember+1289
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:11744 ~declareModuleMember+1176
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:1 *getTypeFromTypeLiteralOrFunctionOrConstructorTypeNode+254
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:1 *declareSymbolAndAddToSymbolTableWorker+83
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:3 *isContextSensitive+4549
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:16537 ~resolveAnonymousTypeMembers+1089
details 0.01% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:18327 ~isRelatedTo+2622
type [top 20 out of 1]
details 100% 7 LoadIC
category [top 20 out of 1]
details 100% 7 Load
file [top 20 out of 1]
details 100% 7 /usr/local/google/home/franzih/TypeScript/lib/tsc.js:18327
state [top 20 out of 6]
details 28.57% 2 (P->P)
details 14.29% 1 (0->.)
details 14.29% 1 (.->1)
details 14.29% 1 (1->P)

--trace-opt –trace-deopt

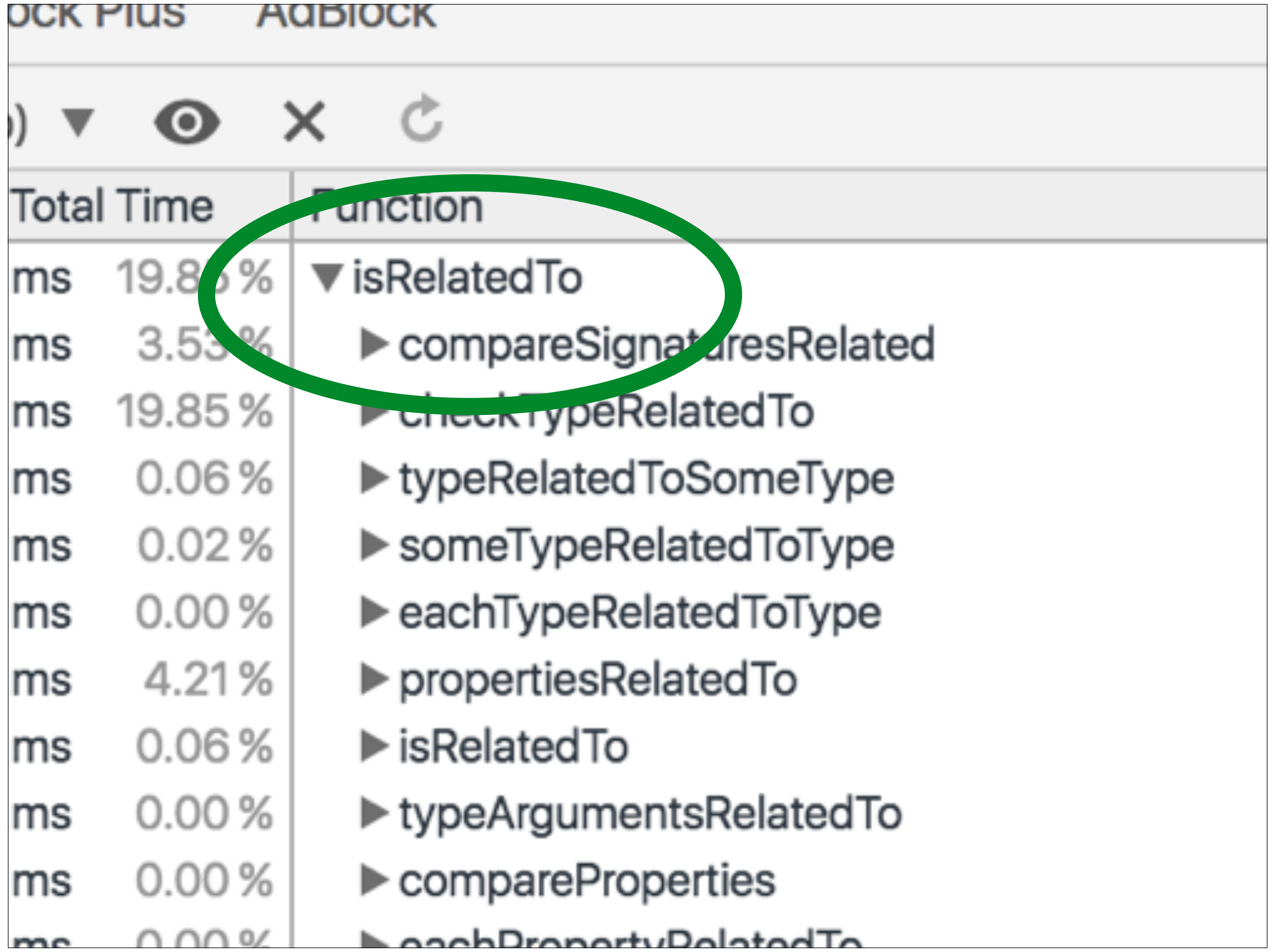
```
$ node --trace-opt –trace-deopt load-opt.js
[compiling method 0x1b9f780f3139 <JS Function
load (SharedFunctionInfo 0x3697a6859ad1)> using
Crankshaft]
[optimizing 0x1b9f780f3139 <JS Function load
(SharedFunctionInfo 0x3697a6859ad1)> - took
0.910, 0.052, 0.058 ms]
[evicting entry from optimizing code map (notify
deoptimized) for 0x3697a6859ad1
<SharedFunctionInfo load>]
```



back to the original problem: a function that is not optimized because optimized too many times when you look at the CPU profile.



Sources
Profiles
Heavy (Bottom Up)
Not optimized: Optimized too many times
I C E
Your IC-Explorer.
Usage
Run your script with --trace_ic and upload on this page:
/path/to/d8 --trace_ic your_script.js > trace.txt
Data
Choose File trace.txt
trace entries: 109620
Result
Group-Key: state
details 57.51% 41727 (N->N)
details 16.06% 11655 (0->.)
details 7.95% 5769 (.->1)
details 4.44% 3220 (P->P)
details 3.88% 2817 ((UNINITIALIZED+UNINITIALIZED=UNINITIALIZED)->(SMI+SM
details 3.58% 2600 (1->P)



AdBlock
Total Time
Function
ms
isRelatedTo
ms
compareSignaturesRelated
ms 19.85 %
checkTypeRelatedTo
ms 0.06 %
typeRelatedToSomeType
ms 0.02 %
someTypeRelatedToType
ms 0.00 %
eachTypeRelatedToType
ms 4.21 %
propertiesRelatedTo
ms 0.06 %
isRelatedTo
ms 0.00 %
typeArgumentsRelatedTo
ms 0.00 %
compareProperties

Be careful with optimizations!

- Don't "optimize" unless you must
- Measure first

Be careful with optimizations!

- V8 internals change
- Different in other engines

- $ chrome --js-flags="--trace-opt"
  - –trace-opt –trace-deopt
  - –print-opt-code
  - –trace-ic
- $ node –trace-ic …
- $ d8 (V8 shell)
- IC Explorer v8/tools/ic-explorer.html

@fhinkel franzih@google.com

I hope I was able to give you an overview how V8 works internally to be so fast and to give you some tools if you want to understand better what’s going on in your own websites.

If you have any questions, please talk to me during the breaks or feel free to reach out via email or twitter.

franzih@google.com

@fhinkel