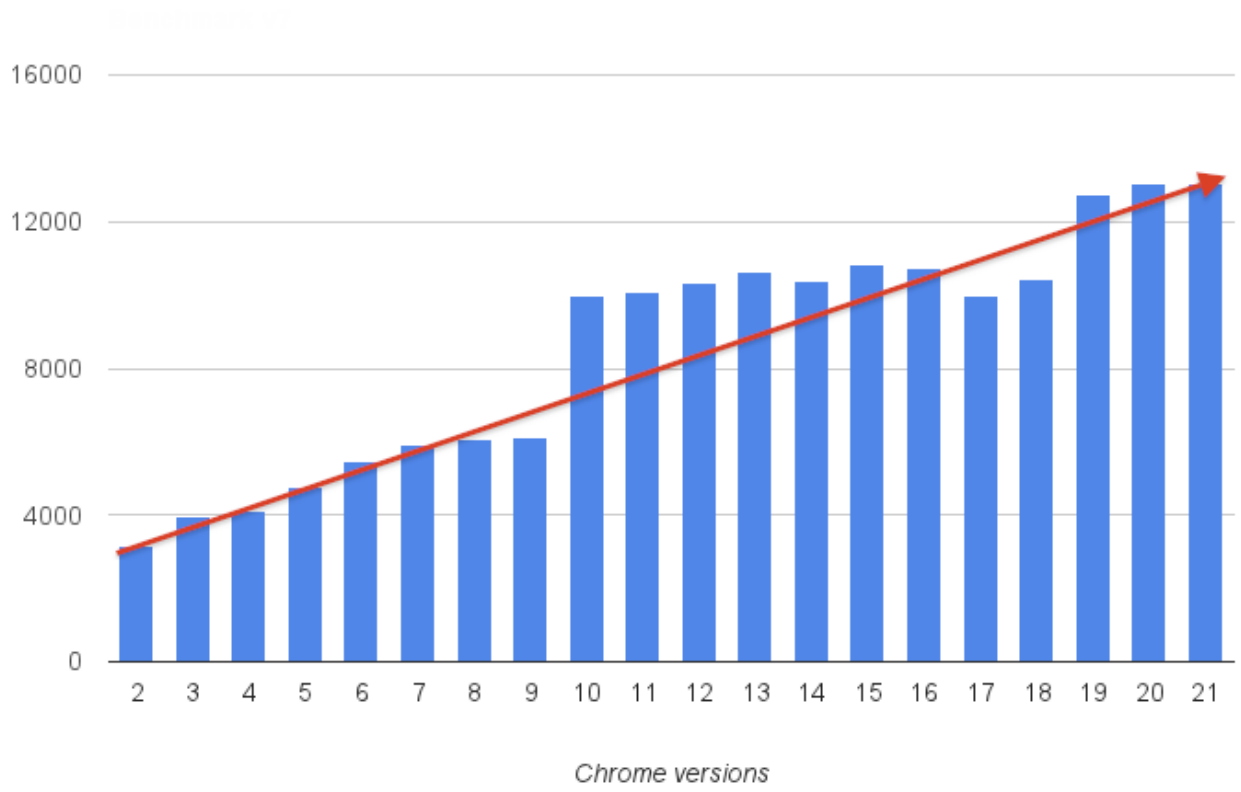
V8

迷渡

@justjavac

https://github.com/justjavac

V8's Score on V8Bench (version 7)



动态语言如何进行快速算术

导读

- V8中数字的表示
- 动态语言的算术运算为何慢
- 解释器、非优化编译器
- 类型反馈、优化编译器
- 去优化（Deoptimization）
- 截断分析
- 编译器的挑战
- 语言设计讨论
- 新的数字类型提案（Int64，BigNum，SIMD）

Encoding of numbers in V8 (x64)

小整数（Small integers） “tagged pointers”.

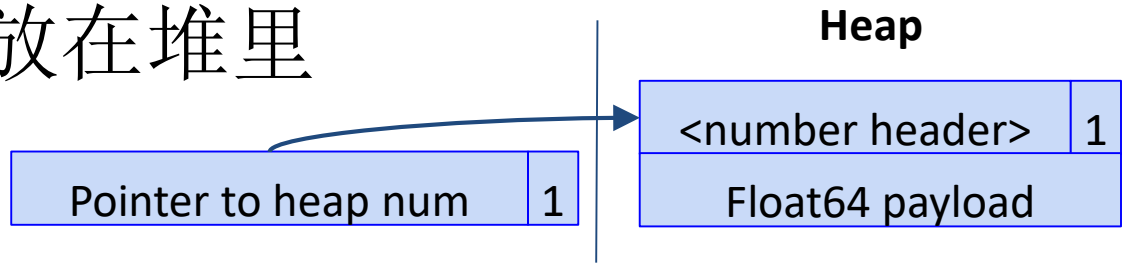
整数: int32 | 0000:0000

引用: high 63-bits of pointer | 1

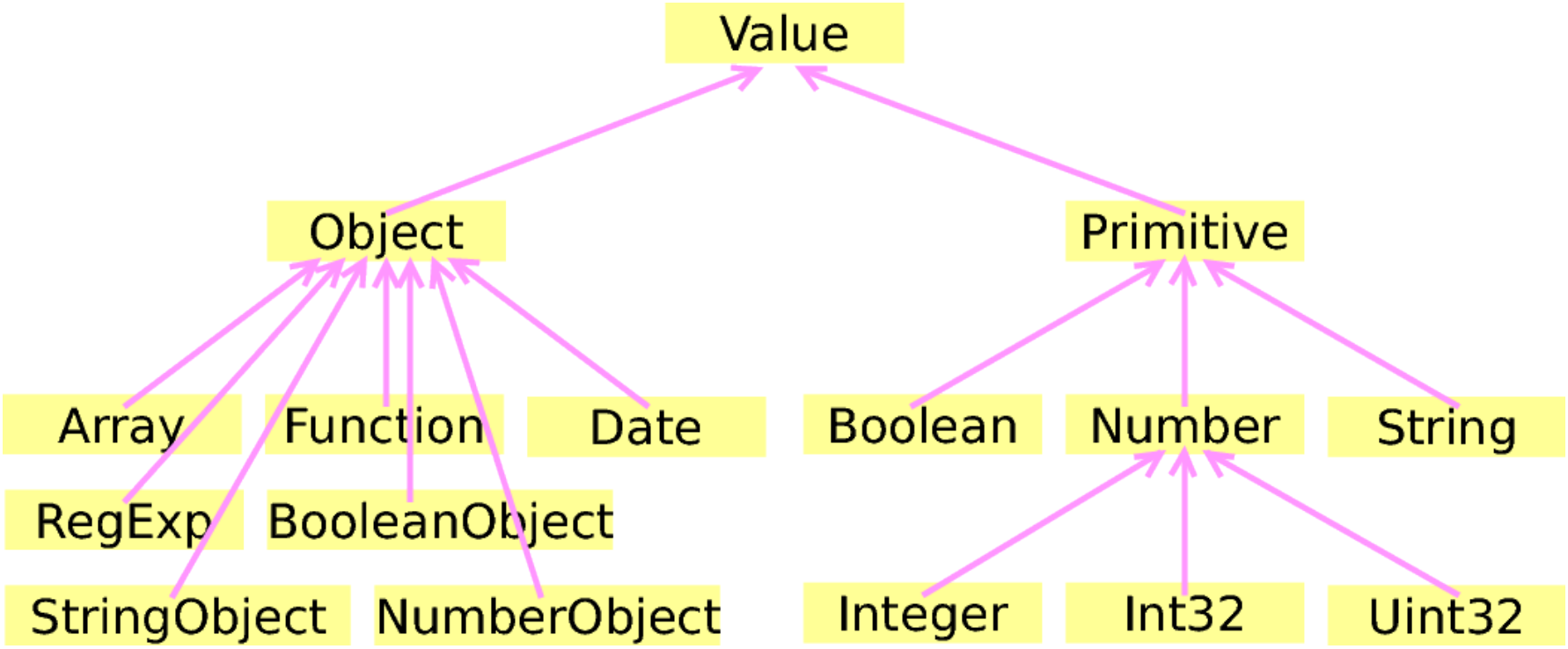
例如, 整数 42 编码为 0x0000002a00000000,

指针 0x12345678 编码为 0x12345679

非整数数值存放在堆里



Data type in V8



javascript中的“加法”

- 为什么 ++[[]][+[]]+[+[]] = 10？
- {} + {} 等于多少?
- 为什么 [1,2] + [3,4] 不等于 [1,2,3,4]？

加法操作

12.8.3 The Addition Operator (+)

NOTE The addition operator either performs string concatenation or numeric addition.

12.8.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be the result of evaluating *MultiplicativeExpression*.
4. Let *rval* be ? GetValue(*rref*).
5. Let *lprim* be ? ToPrimitive(*lval*).
6. Let *rprim* be ? ToPrimitive(*rval*).
7. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ? ToString(*lprim*).
    b. Let *rstr* be ? ToString(*rprim*).
    c. Return the String that is the result of concatenating *lstr* and *rstr*.
8. Let *lnum* be ? ToNumber(*lprim*).
9. Let *rnum* be ? ToNumber(*rprim*).
10. Return the result of applying the addition operation to *lnum* and *rnum*. See the Note below 12.8.5.

“加法”运算结果

	undefined	boolean	number	string	function	object	null	array
undefined	number	number	number	string	string	string	number	string
boolean	number	number	number	string	string	string	number	string
number	number	number	number	string	string	string	number	string
string	string	string	string	string	string	string	string	string
function	string	string	string	string	string	string	string	string
object	string	string	string	string	string	string	string	string
null	number	number	number	string	string	string	number	string
array	string	string	string	string	string	string	string	string

V8的算数运算

快速模式：直接调用二进制代码assembly

- 小整数
- 堆区的数值
- 怪异类型 - undefined, null, true, false.
- 字符串 (字符串连接运算)

对象（object）运算使用 C++ 实现（慢）

快速模式

- 代码：a + b

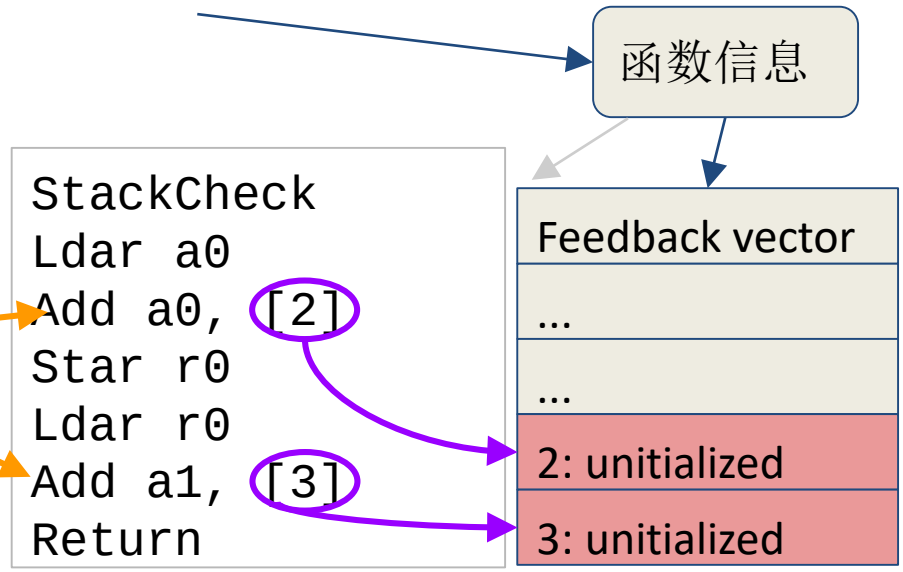
- **常规编译：**
- mov eax, a
- mov ebx, b
- call RuntimeAdd

- **IC 编译：**
- mov eax, a
- mov ebx, b
- add eax, ebx

Type feedback

二元运算符可以收集函数的 type feedback 信息.

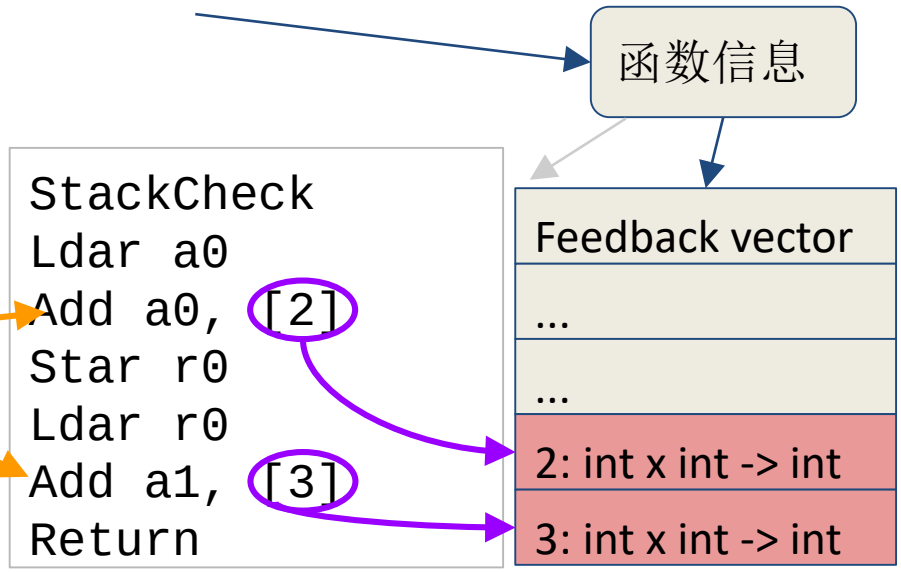
```
function f(a, b){
    var c = a + a;
    return b + c;
}
```



Type feedback

二元运算符可以收集函数的 type feedback 信息.

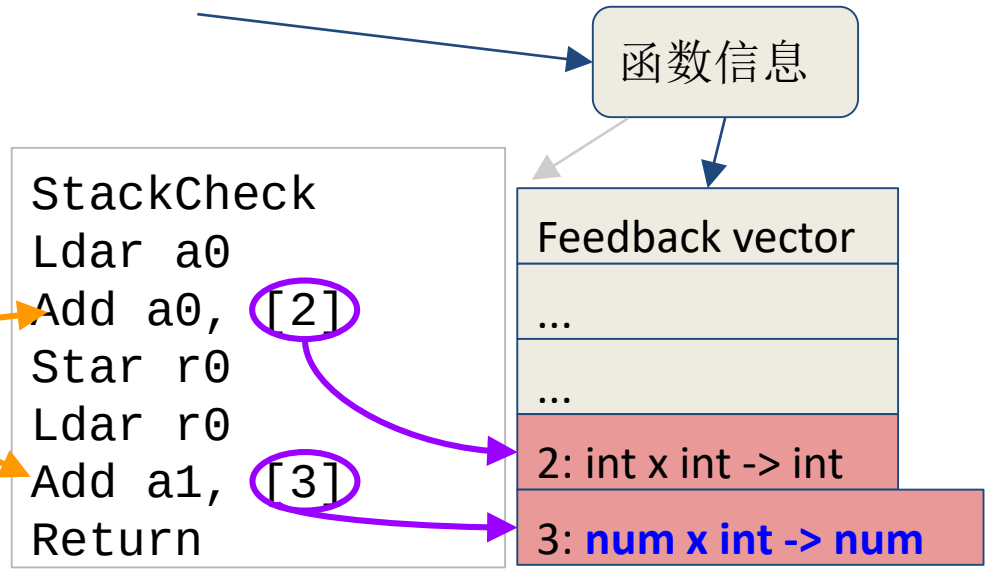
```
function f(a, b){
    var c = a + a;
    return b + c;
}
f(1, 2);
```



Type feedback

二元运算符可以收集函数的 type feedback 信息.

```
function f(a, b){
    var c = a + a;
    return b + c;
}
f(1, 2);
f(1, 0.1)
```



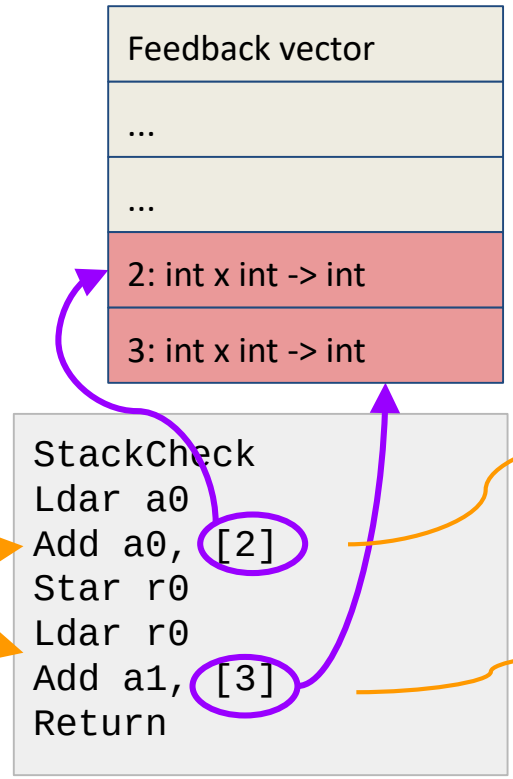
Optimizing compiler

- 使用 type feedback 做动态检查
- 一般而言，在编译阶段提前检查
- 检查之后，使用该类型作为动态类型
- 如果检查失败，去优化（deoptimize）
  - 去优化之后，可能会使用解释器运行中间码

Compile with types

对输入进行检查、相加、返回结果。

```
function f(a, b){
   var c = a + a;
   return b + c;
}
 f(1, 2);
 f(1, 2);
```



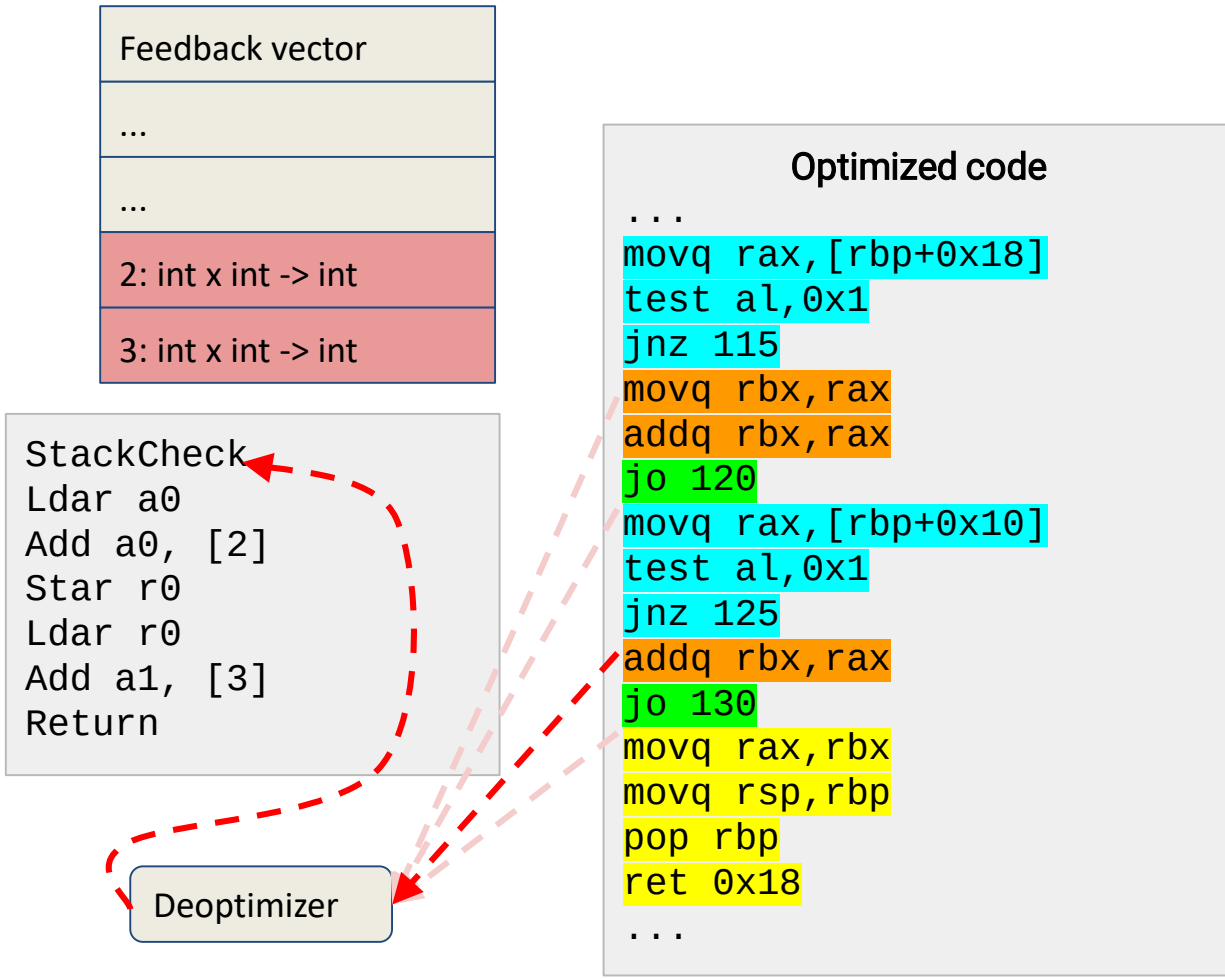
Optimized code

```
...
movq rax,[rbp+0x18]
test al,0x1
jnz 115
movq rbx,rax
addq rbx,rax
jo 120
movq rax,[rbp+0x10]
test al,0x1
jnz 125
addq rbx,rax
jo 130
movq rax,rbx
movq rsp,rbp
pop rbp
ret 0x18
...
```

Compile with types

对输入进行检查、相加、返回结果。

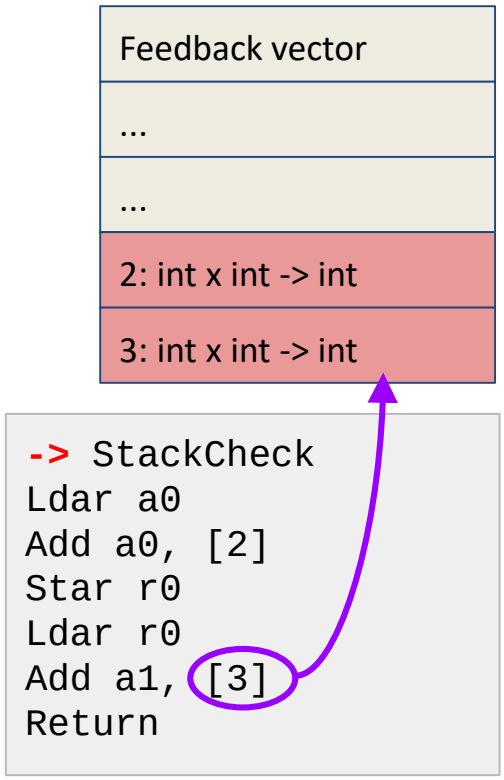
```
function f(a, b){
   var c = a + a;
   return b + c;
}
f(1, 2);
f(1, 2);
f(1, 0.1);//Deopt!
```



去优化Deoptimization

对输入进行检查、相加、返回结果。

```
function f(a, b){
   var c = a + a;
   return b + c;
}
 f(1, 2);
 f(1, 2);
 f(1,
0.1);//Deopt!
```



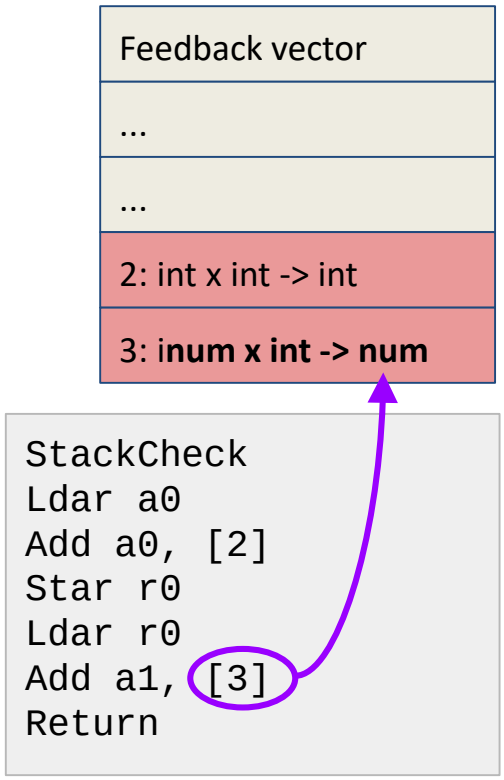
Deoptimizer

- 生成一个未优化的帧

去优化Deoptimization

对输入进行检查、相加、返回结果。

```
function f(a, b){
   var c = a + a;
   return b + c;
}
 f(1, 2);
 f(1, 2);
 f(1,
0.1);//Deopt!
```



Deoptimizer

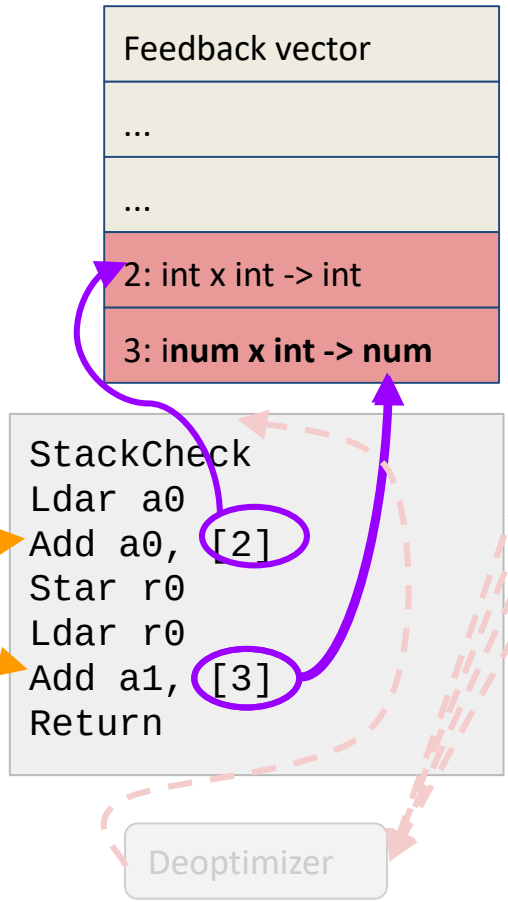
- 生成一个未优化的帧

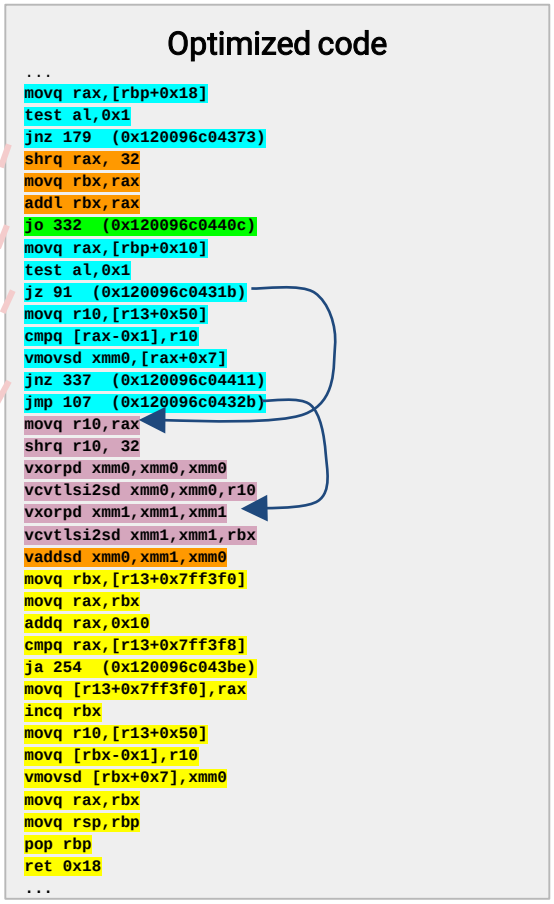
当运行加法运算时更新 Type feedback 信息。

去优化Deoptimization

对输入进行检查、相加、返回结果。

```
function f(a, b){
   var c = a + a;
   return b + c;
}
 f(1, 2);
 f(1, 2);
 f(1,
0.1);//Deopt!
```





要避免“去优化”

- 去优化的消耗是很大的！
  - 主要是因为重新优化（re-optimization）的消耗是很大的！

- 如果我们不恰当的使用类型反馈信息，那么我们就会陷入去优化的怪圈（**deoptimization loop**）：

  函数不停的去优化，然后再重新优化，直到我们达到了重优化的次数限制，这时我们的函数将再也不会被V8引擎优化。

Deoptimization loop example

```
function f(x, y) {   // 首次运行：x=1, y=undefined
    if (x) y = x + 1;  // Integer feedback for +
    // Type feedback 猜测 y 为 integer
    // 后面的代码会把 y 作为 integer
    // Crankshaft 会把 y 当作 int32
    // 如果 y 不是 int32，那么引擎做去优化
    if (x) y = y + 2;  // Integer feedback for +
    return y;
  }
```

Deoptimization loop example

```
function f(x, y) {   // 首次 x=1, 然后 x=0
    if (x) y = x + 1;  // Integer feedback for +
    // 如果 y 不是 int32，那么引擎做去优化
    if (x) y = y + 2;  // Integer feedback for +
    return y;
}

f(1, undefined);  // 首先 feedback, 然后 optimize.
// Crankshaft 进行去优化，最终优化被禁止

for (var i = 0; i < 1000000; i++) {
   f(0, undefined);
}
```

Can we do better?

Javascript code

```
var l = a[i] & 0x3fff,
    h = a[i++]>>14,
    m = xh*l+h*xl;
     l = ((m&0x3fff)<<14 …
```

Asm.js code

```
e=...|0;K=...|0; ...
S=e+K+c+b+f+E+D+m+o+g+a+C+B+A+z+y+R|0
```

根本不需要做范围检测，因为位运算只对低32位有效。

截断（Truncations）

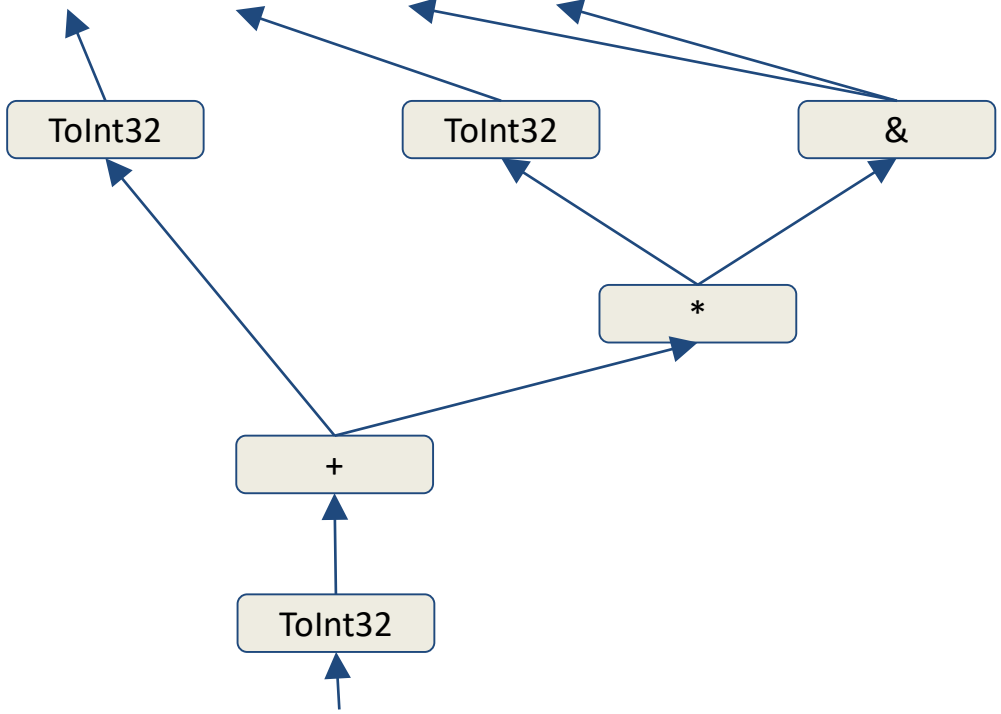
在 **`(x + y)|0` 运算时，我们只关心低 32 位的结果。**即使 x, y 都是int52，我们也只关心 x 和 y 的低 32 位。

表达式 `+a[i]` 不区分 a[i] = undefined 和 a[i] = NaN。在稀疏数组中，我们会读取到 NaN! 而不是 undefined。

表达式 **`c ? x : y`** 也不需要区分 **c** = 1 和 **c** = true。

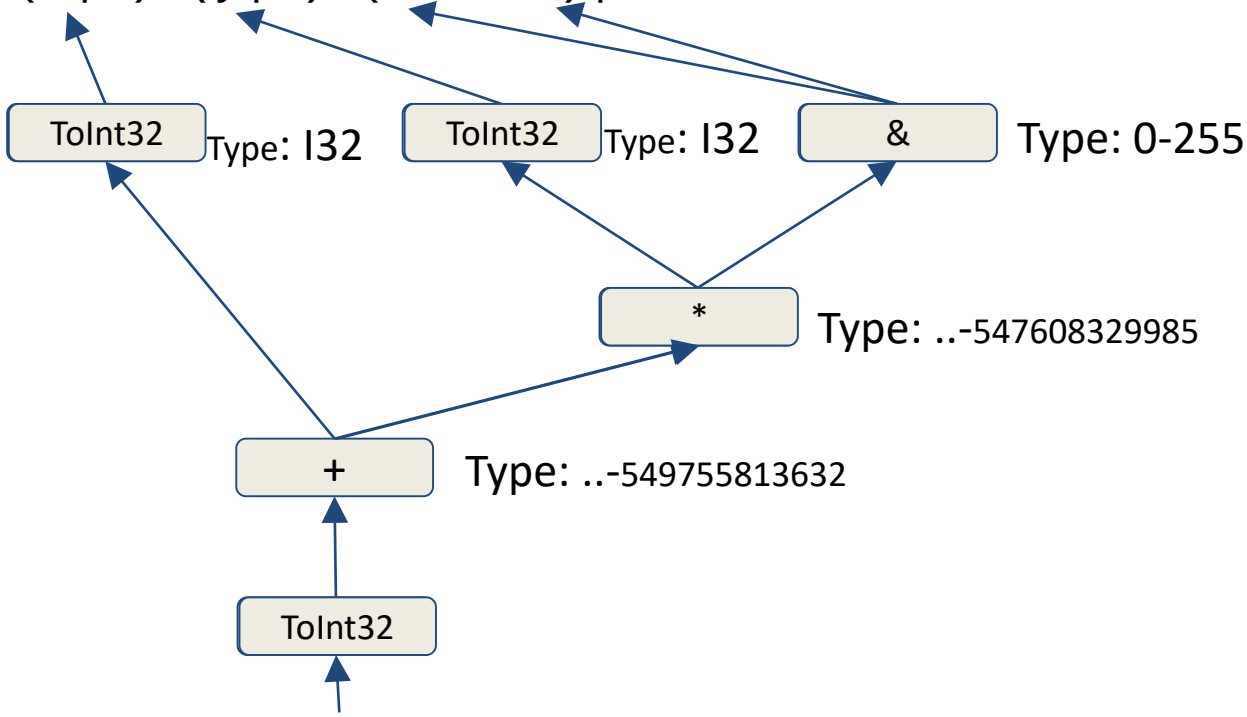
Example 1

```
var a = (x|0)+(y|0)*(z&0xff)|0
```



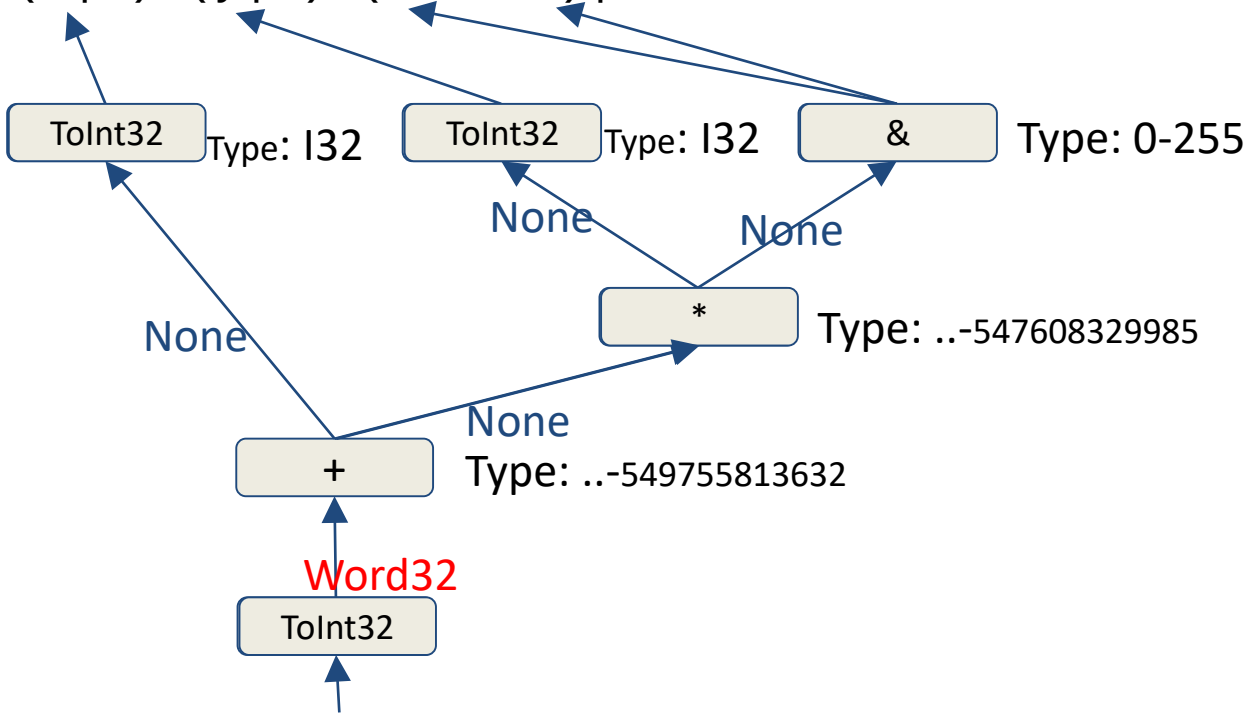
Example 1 (类型上限)

```
var a = (x|0)+(y|0)*(z&0xff)|0
```



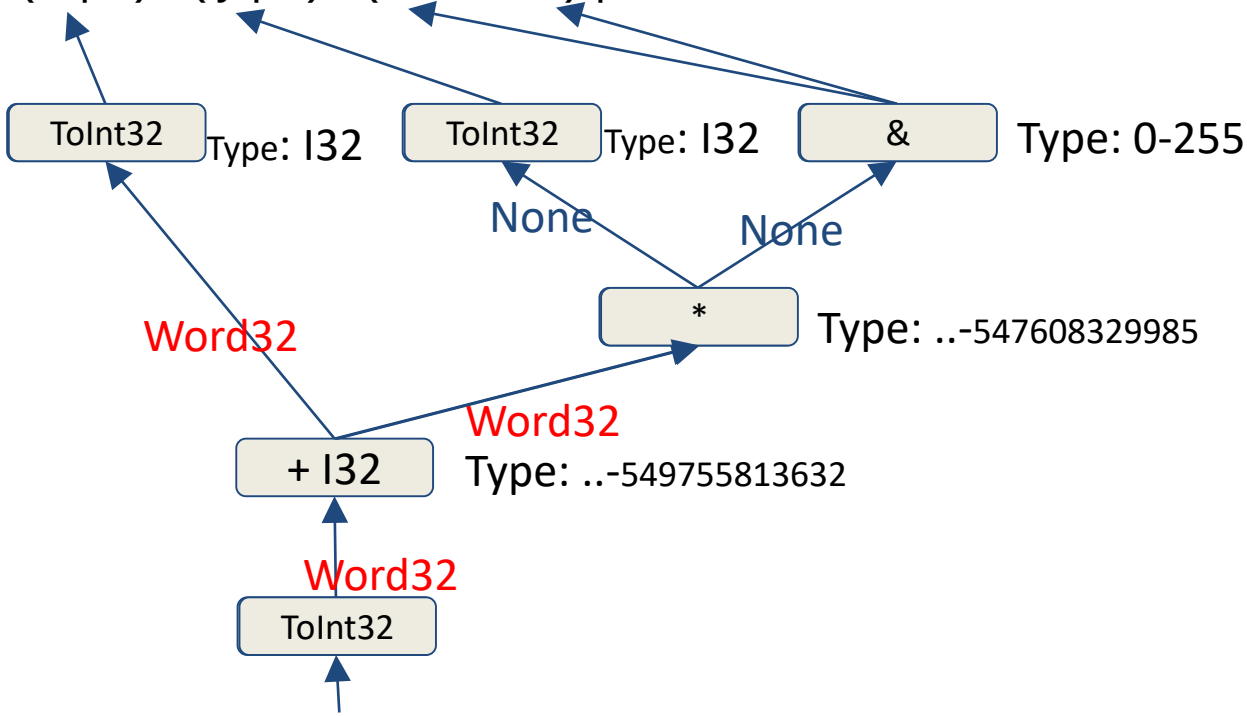
Example 1 (截断传播)

```
var a = (x|0)+(y|0)*(z&0xff)|0
```



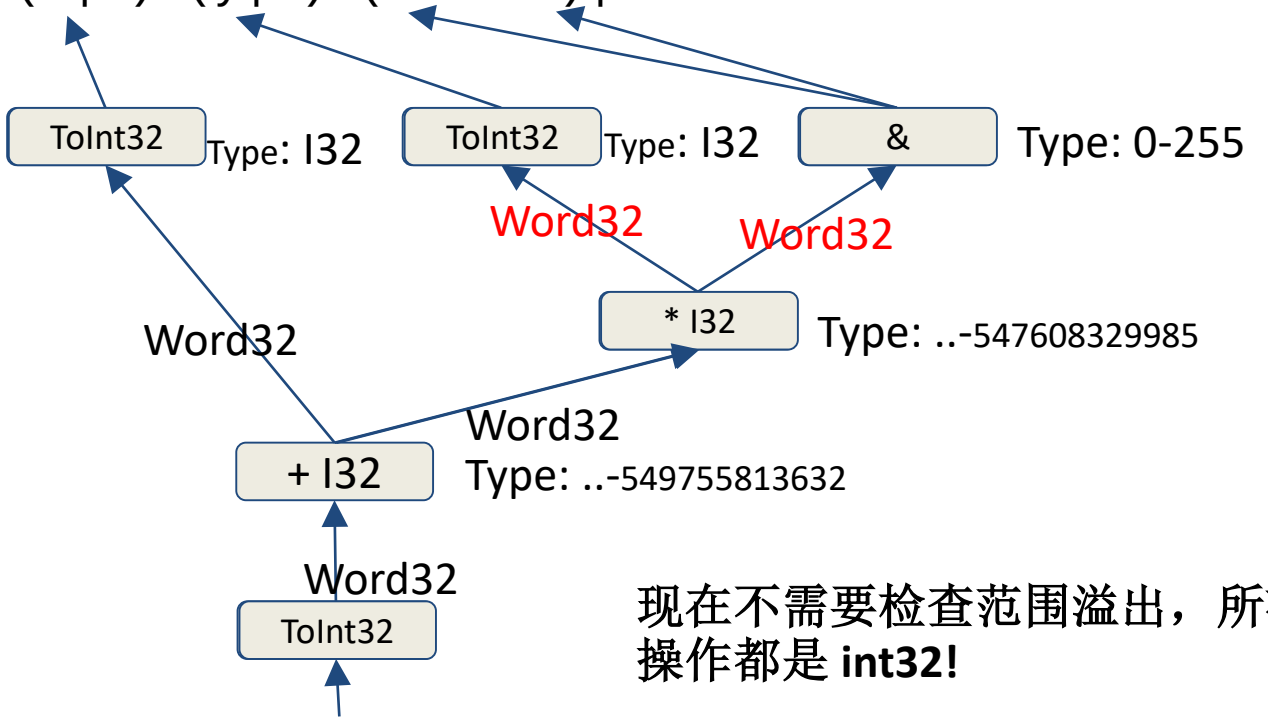
Example 1 (截断传播)

```
var a = (x|0)+(y|0)*(z&0xff)|0
```



Example 1 (截断传播)

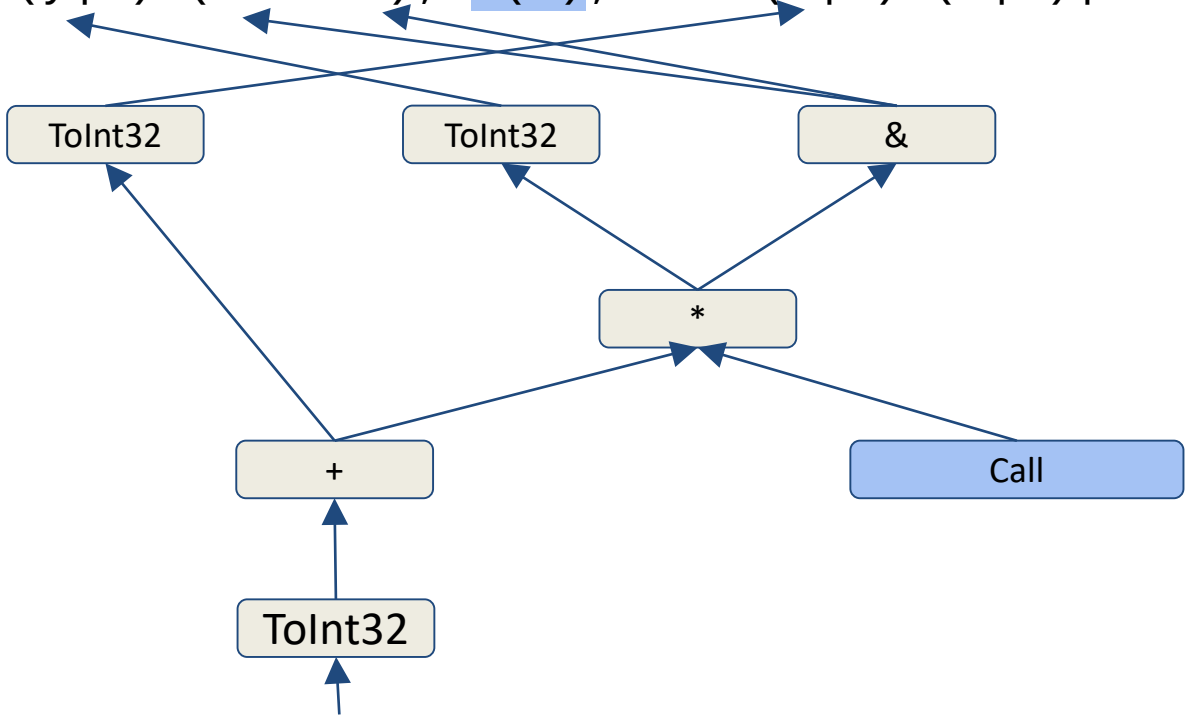
```
var a = (x|0)+(y|0)*(z&0xff)|0
```



现在不需要检查范围溢出，所有的操作都是 **int32!**

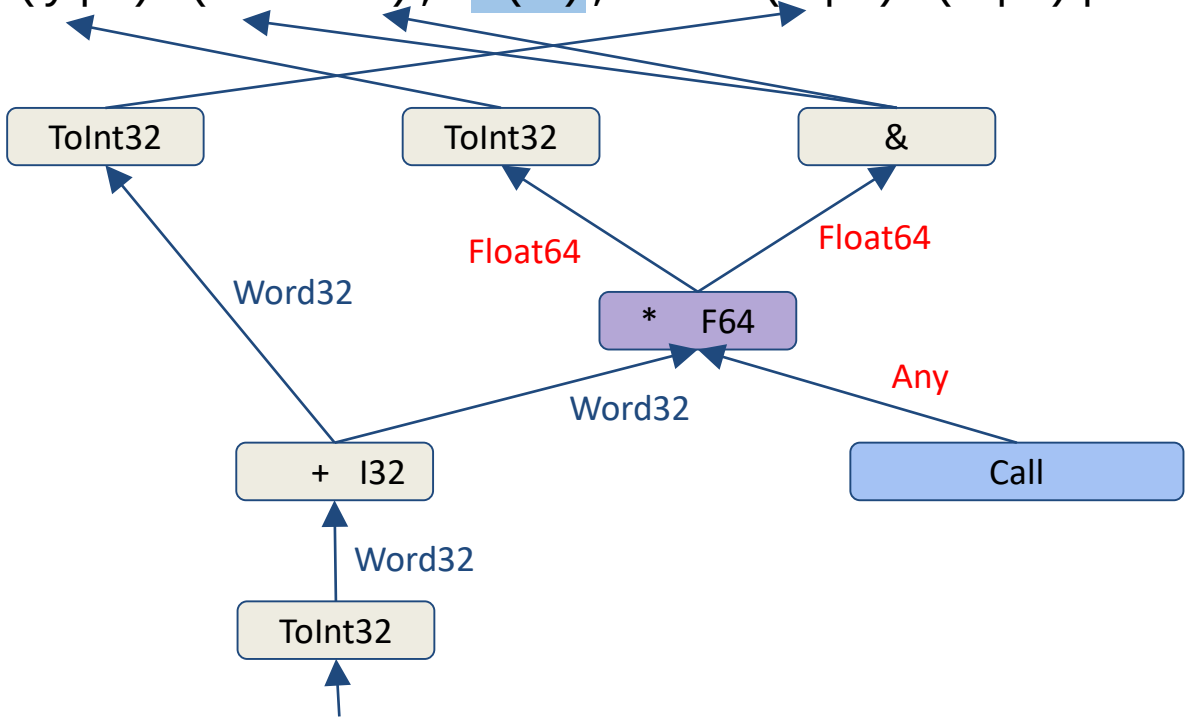
Example 2

```
var a = (y|0)*(z&0xff); f(a); a = (x|0)+(a|0)|0
```



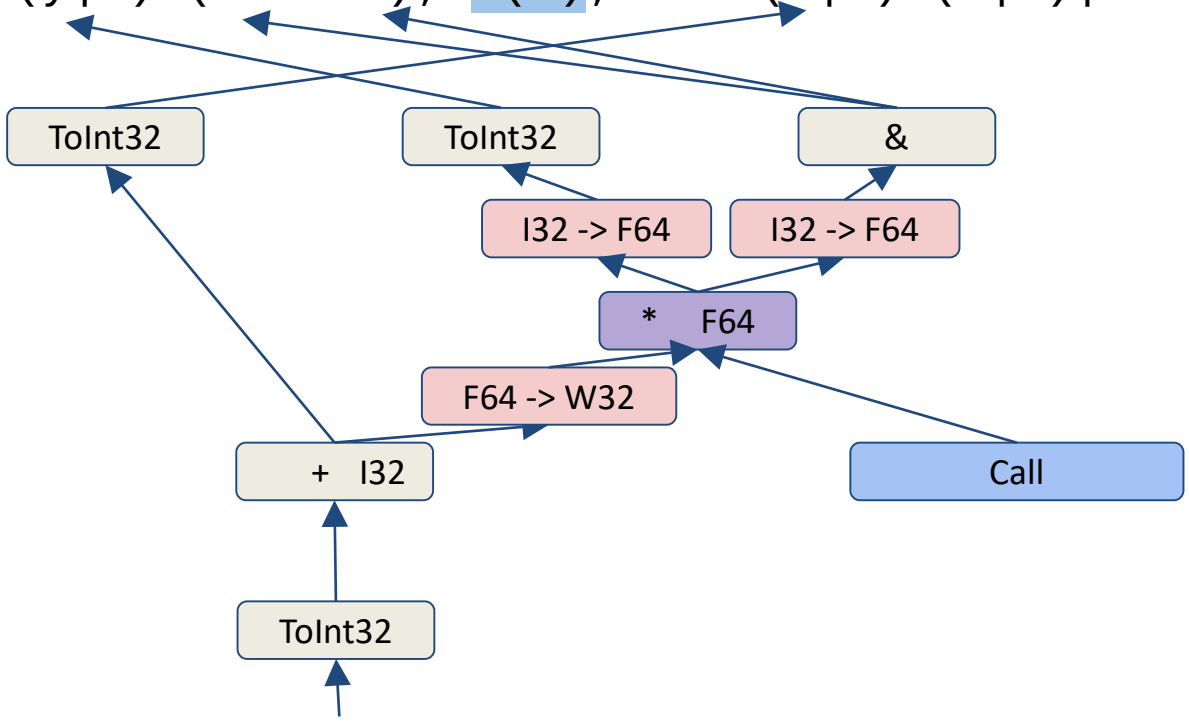
Example 2

```
var a = (y|0)*(z&0xff); f(a); a = (x|0)+(a|0)|0
```



Example 2

```
var a = (y|0)*(z&0xff); f(a); a = (x|0)+(a|0)|0
```



“截断”的其他用途

截断还可以用于其他优化:

- 从 double 到 integer 转换时的负零检查
- 乘法运算的负零检查
- 读取数组元素时的 *undefined* 检查
- 使引擎能更精准地表示类型

截断传播只在 V8 的 Turbofan 编译器有效。

面临的挑战

目前，引擎首先进行截断分析，而类型反馈不影响截断。

例如，（ x + y | 0 ）中 x 和 y 将会被作为整型。
理想情况下，使用x和y的类型反馈，然后进行 int32加法。

然而，很多情况下，最明智的选择往往是“更差”的表示法。

例如，a + b + 0.5 应该是 float64，即使 a，b 被反馈为整型。

未来方向

JavaScript 可以使用任意的精确的整数

- 我们可以更加精准的控制V8引擎生成的代码!

也许以后会有 (U)Int64 或 BigNum 类型?

SIMD 对于 JavaScript (或其他动态语言)或许有意义?

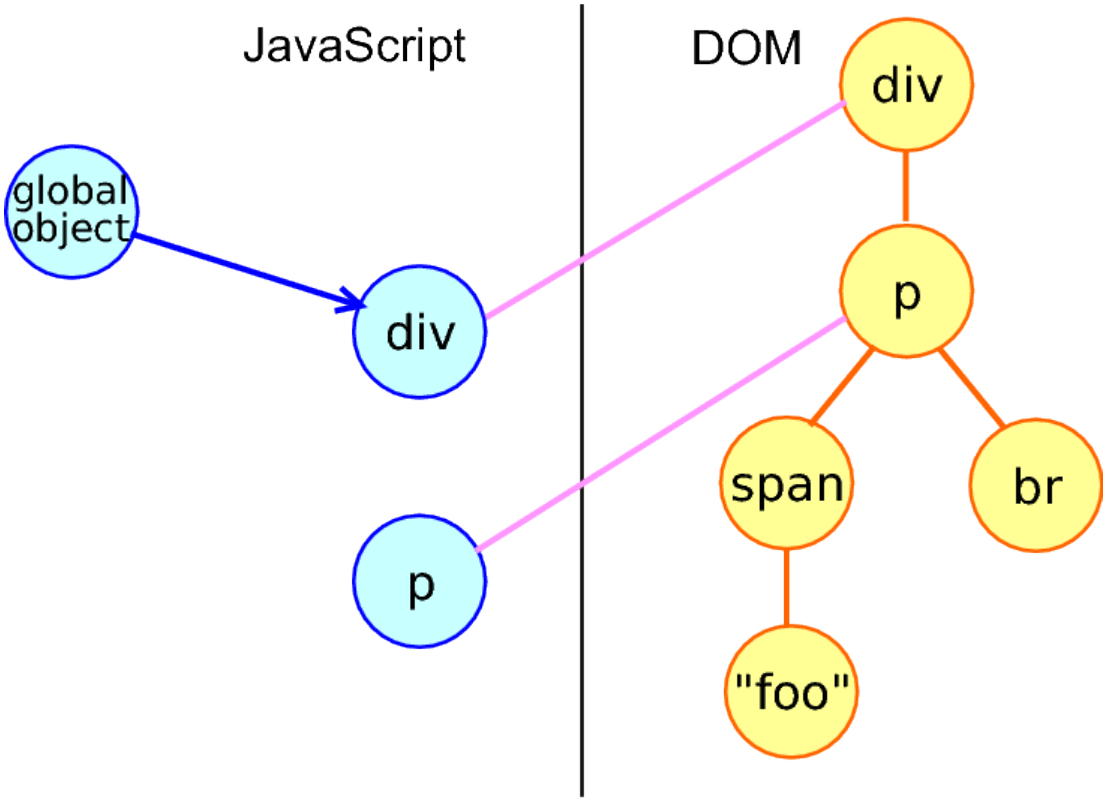
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SIMD

其它

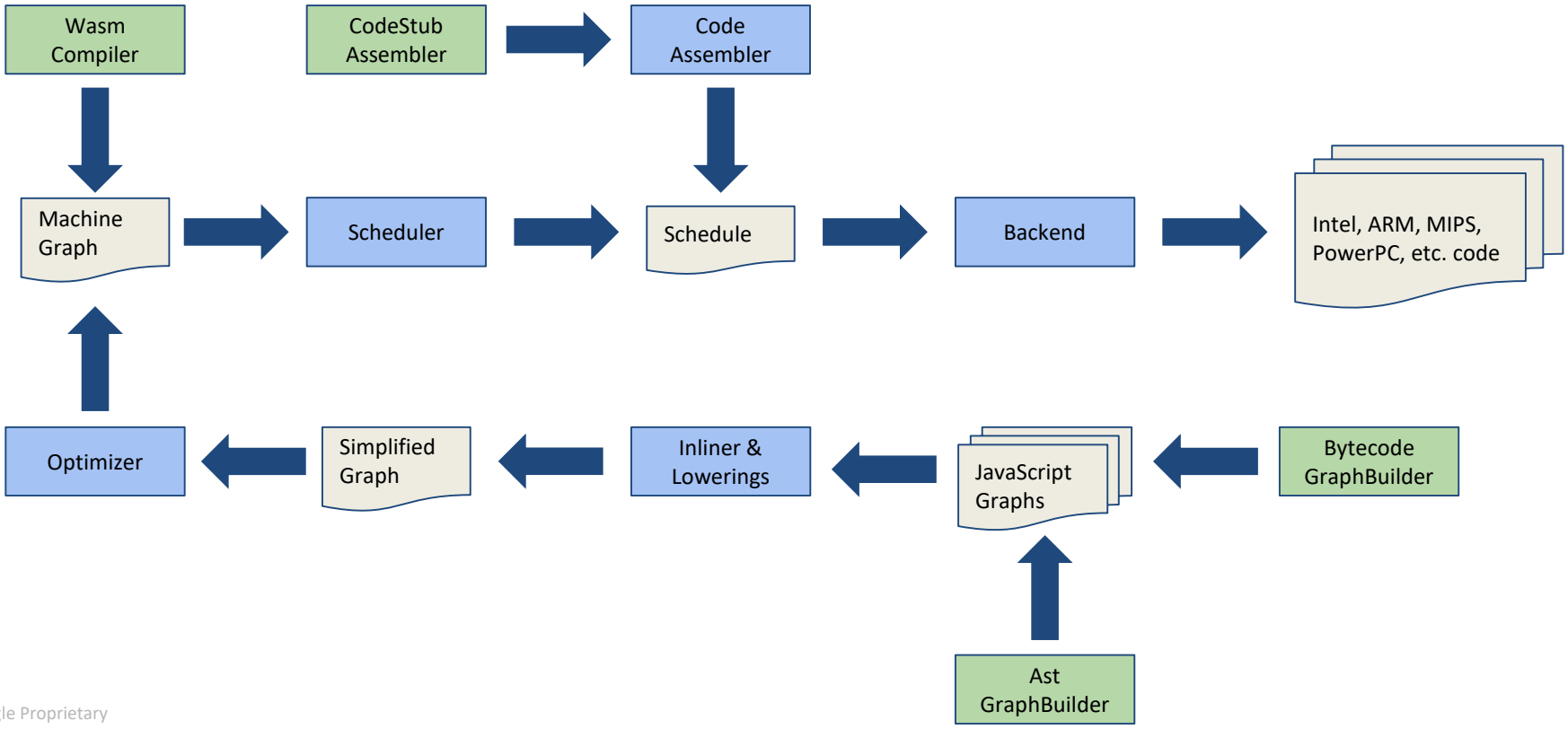
- V8 Binding
- TurboFan 架构
- Intermediate Representation
- Hidden Classes
- Inline Cache

V8 Binding： JS object 和 DOM 对象

```
div = document.createElement("div");
div.innerHTML = "<p><span>foo</span><br></p>";
div.firstChild;
```

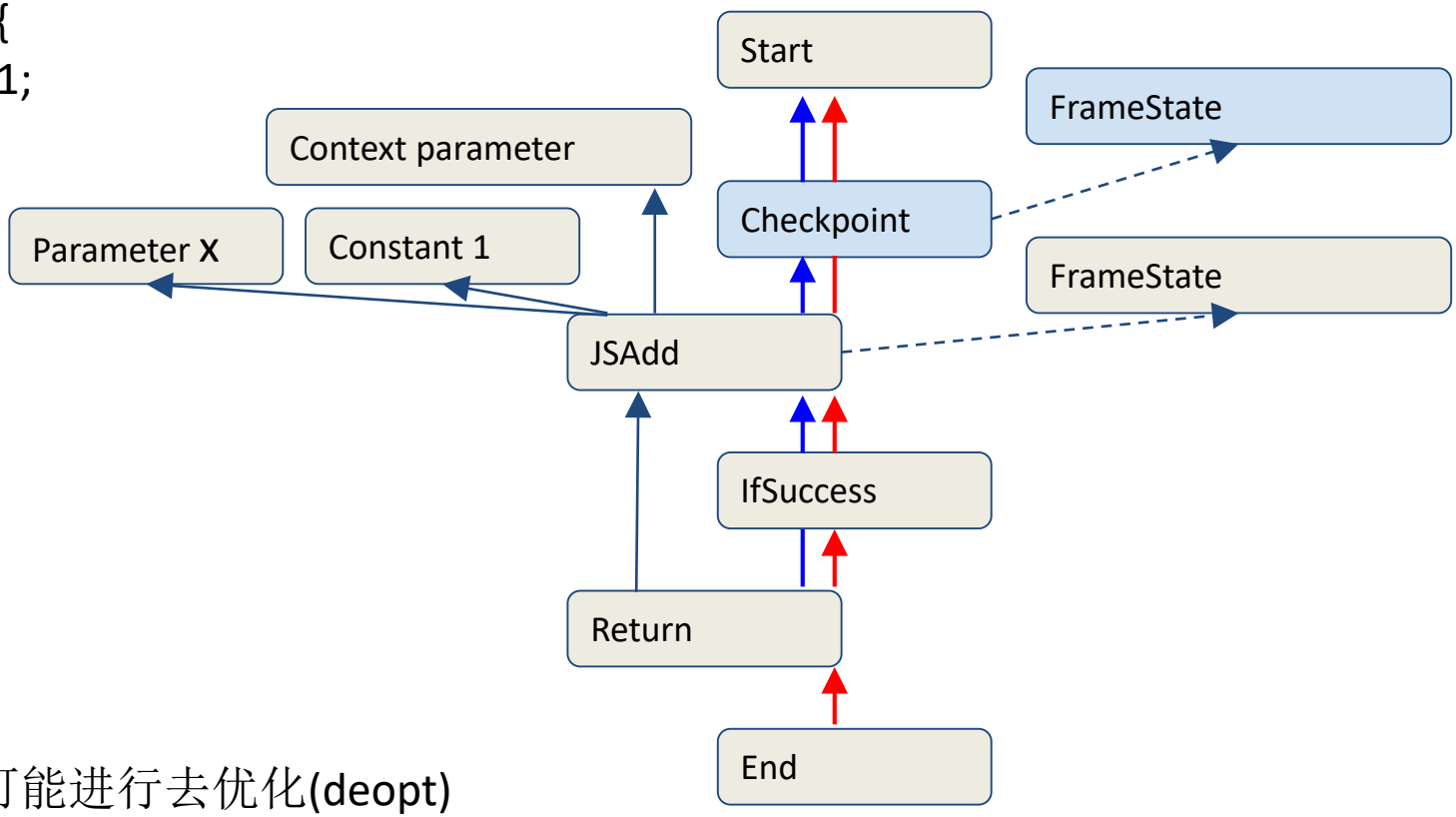


TurboFan 架构



TurboFan IR

```
function f(x) {
    return x + 1;
}
```



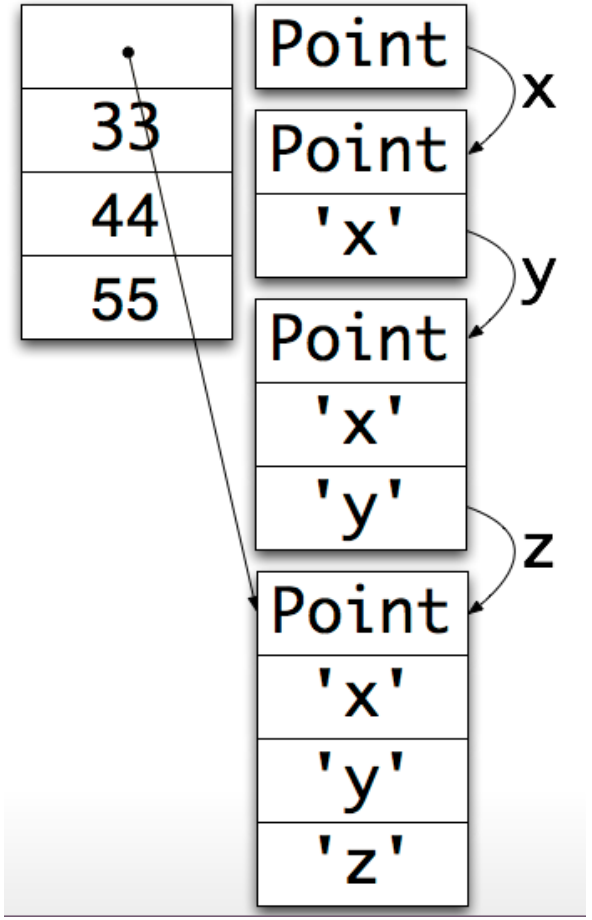
加法运算有可能进行去优化(deopt)

Hidden Classes

JAVASCRIPT

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(11, 22);
var p2 = new Point(33, 44);
p2.z = 55
// warning! p1 and p2 now have
// different hidden classes
```



送福利


知乎 LIVE
全部 Live
热门精选
课程与专题
我的
前端工程师的入门与进阶
justjavac ›
时间（星期六）
2017-04-22 20:00
1578 人已参与
送给好友
赞助并参与活动（¥9.99）
Live 简介
我是 justjavac（迷渡），前端开发工程师， 8 年前端开发经验。之前在 GDG 、稀土掘金、 D-Day 等进行过多次前端技术分享。曾组织翻译《 JavaScript Quirks 》，正在出版《代码之谜》。 JSON API 中文规范维护者， Flaurm 中文社区创始人。平时混迹于 GitHub ，参与众多开源项目。
本次 Live 我将带来前端开发... 展开全部

前端工程师的入门与进阶x10

Talk is cheap.
Show me the code.

整数相加

```
function add(obj) {
    return obj.prop + obj.prop;
}

const length = 1000 * 1000;

const o = { prop: 1 };

for (let i = 0; i < length; i++) {
    add(o);
}
```

d8 --trace-opt-verbose add-of-ints.js

- [marking 030ADD65 <JS Function test (SharedFunctionInfo 03E3A99D)> **for recompilation**, reason: **small function**, ICs with typeinfo: 3/3 (100%), generic ICs: 0/3 (0%)]
- [compiling method 030ADD65 <JS Function test (SharedFunctionInfo 03E3A99D)> using Crankshaft]
- [optimizing 030ADD65 <JS Function test (SharedFunctionInfo 03E3A99D)> - took 0.033, 0.067, 0.537 ms]
- [**completed optimizing** 030ADD65 <JS Function test (SharedFunctionInfo 03E3A99D)>]
- [marking 030ADA75 <JS Function (SharedFunctionInfo 03E3A8E9)> for recompilation, reason: small function, ICs with typeinfo: 3/3 (100%), generic ICs: 0/3 (0%)]

混合相加

```
function test( obj ) {
  return obj.prop + obj.prop;
}

let a = { prop: 'a' }, b = { prop: [] }, i = 0;

while ( i++ < 10000 ) {
  test( Math.random() > 0.5 ? a : b );
}
```

d8 --trace-opt-verbose add-of-mixed.js

- [marking 03F723E9 <JS Function valueOf (SharedFunctionInfo 03F32765)> for recompilation, reason: small function, ICs with typeinfo: 0/0 (100%), gener
- ic ICs: 0/0 (0%)]
- [marking 03F6C499 <JS Function toString (SharedFunctionInfo 03F3687D)> for recompilation, reason: small function, ICs with typeinfo: 4/5 (80%), generic ICs: 0/5 (0%)]
- [compiling method 03F723E9 <JS Function valueOf (SharedFunctionInfo 03F32765)> using Crankshaft]
- [compiling method 03F6C499 <JS Function toString (SharedFunctionInfo 03F3687D)> using Crankshaft]
- [optimizing 03F723E9 <JS Function valueOf (SharedFunctionInfo 03F32765)> - took 0.031, 0.112, 0.037 ms]
- [completed optimizing 03F723E9 <JS Function valueOf (SharedFunctionInfo 03F32765)>]
- [optimizing 03F6C499 <JS Function toString (SharedFunctionInfo 03F3687D)> - took 0.073, 0.211, 0.076 ms]
- [completed optimizing 03F6C499 <JS Function toString (SharedFunctionInfo 03F3687D)>]

- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, **not enough type info**: 2/3 (66%)]
- [not yet optimizing test, **not enough type info**: 2/3 (66%)]
- [marking 037ADADD <JS Function (SharedFunctionInfo 0453A8F5)> for recompilation, reason: small function, ICs with typeinfo: 7/7 (100%), generic ICs:
- 0/7 (0%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]
- [not yet optimizing test, not enough type info: 2/3 (66%)]

--trace-deopt

```
function test( obj ) {
  return obj.prop + obj.prop;
}

let a = { prop: 'a' }, b = { prop: [] }, i = 0;

while ( i++ < 10000 ) {
  test( i !== 8000 ? a : b );
}
```

- [deoptimizing (DEOPT eager): begin 035ADE39 <JS Function test (SharedFunctionInfo 0433ABA1)> (opt #0) @3, FP to SP delta: 12, caller sp: 0x00c2f548]
- reading input frame test => node=4, args=2, height=1; inputs:
- 0: 0x035ade39 ; [fp - 8] 035ADE39 <JS Function test (SharedFunctionInfo 0433ABA1)>
- 1: 0x03d8a395 ; [fp + 12] 03D8A395 <JS Global Object>
- 2: 0x035adec1 ; ecx 035ADEC1 <an Object with map 0450DB69>
- 3: 0x03d6b2b9 ; [fp - 12] 03D6B2B9 <FixedArray[173]>
- translating frame test => node=4, height=0
- 0x00c2f544: [top + 20] <- 0x03d8a395 ; 03D8A395 <JS Global Object> (input #1)
- 0x00c2f540: [top + 16] <- 0x035adec1 ; 035ADEC1 <an Object with map 0450DB69> (input #2)
- 0x00c2f53c: [top + 12] <- 0x04433dac ; caller's pc
- 0x00c2f538: [top + 8] <- 0x00c2f564 ; caller's fp
- 0x00c2f534: [top + 4] <- 0x03d6b2b9 ; context 03D6B2B9 <FixedArray[173]> (input #3)
- 0x00c2f530: [top + 0] <- 0x035ade39 ; function 035ADE39 <JS Function test (SharedFunctionInfo 0433ABA1)> (input #0)
- [deoptimizing (eager): end 035ADE39 <JS Function test (SharedFunctionInfo 0433ABA1)> @3 => node=4, pc=0x044340e5, caller sp=0x00c2f548, state=NO_REGISTERS, took 5.152 ms]
- [removing optimized code for: test]
- [evicting entry from optimizing code map (notify deoptimized) for 0433ABA1 <SharedFunctionInfo test>]

Garbage Collection

```
function strToArray( str ) {
  let i = 0,
    len = str.length,
    arr = new Uint16Array( str.length );
  for ( ; i < len; ++i ) {
    arr[ i ] = str.charCodeAt( i );
  }
  return arr;
}

let i = 0, str = 'V8 is the collest';

while ( i++ < 1e5 ) {
  strToArray( str );
}
```

d8 --trace-gc gc.js

- [14484:01BF4CA8] 9 ms: Scavenge 1.7 (2.5) -> 1.6 (3.5) MB, 0.7 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 11 ms: Scavenge 1.7 (3.5) -> 1.7 (4.5) MB, 0.6 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 131 ms: Scavenge 3.2 (7.5) -> 2.7 (9.5) MB, 1.8 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 196 ms: Scavenge 4.2 (9.5) -> 2.7 (9.5) MB, 6.7 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 223 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 8.0 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 250 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 8.1 / 0.0 ms [allocation failure].
- [14484:01BF4CA8] 275 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 8.0 / 0.0 ms [allocation failure].

no-gc

```
function strToArray( str, bufferView ) {
  let i = 0,
    len = str.length;
  for ( ; i < len; ++i ) {
    bufferView[ i ] = str.charCodeAt( i );
  }
  return bufferView;
}

let i = 0,
  str = 'V8 is the coolest',
  buffer = new ArrayBuffer( str.length * 2 ),
  bufferView = new Uint16Array( buffer );

while ( i++ < 1e5 ) {
  strToArray( str, bufferView );
}
```

- [23052:01D277F8] 9 ms: Scavenge 1.7 (2.5) -> 1.6 (3.5) MB, 0.8 / 0.0 ms [allocation failure].
- [23052:01D277F8] 10 ms: Scavenge 1.7 (3.5) -> 1.7 (4.5) MB, 0.6 / 0.0 ms [allocation failure].

full gc

```
function strToArray( str ) {
  var i = 0,
    len = str.length,
    arr = new Uint16Array( str.length );
  for ( ; i < len; ++i ) {
    arr[ i ] = str.charCodeAt( i );
  }
  return arr;
}

var i = 0, str = 'V8 is the coolest', arr = [];

while ( i++ < 1e6 ) {
  strToArray( str );
  if ( i % 100000 === 0 ) {
    // 数组里面存放大对象 huge object
    arr.push( new Uint16Array( 100000000 ) );
    // 5% 概率释放数组
    Math.random() > 0.95 && ( arr.length = 0 );
  }
}
```

d8 --trace-gc gc.js

- [8664:01E15D10] 9 ms: Scavenge 1.7 (2.5) -> 1.6 (3.5) MB, 0.6 / 0.0 ms [allocation failure].
- [8664:01E15D10] 11 ms: Scavenge 1.7 (3.5) -> 1.7 (4.5) MB, 0.7 / 0.0 ms [allocation failure].
- [8664:01E15D10] 71 ms: Scavenge 3.2 (7.5) -> 2.7 (9.5) MB, 1.8 / 0.0 ms [allocation failure].
- [8664:01E15D10] 94 ms: Scavenge 4.2 (9.5) -> 2.7 (9.5) MB, 6.3 / 0.0 ms [allocation failure].
- [8664:01E15D10] 119 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 7.6 / 0.0 ms [allocation failure].
- [8664:01E15D10] 144 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 7.4 / 0.0 ms [allocation failure].
- [8664:01E15D10] 169 ms: Scavenge 4.6 (9.5) -> 2.7 (9.5) MB, 8.3 / 0.0 ms [allocation failure].
- [8664:01E15D10] 193 ms: Mark-sweep 4.6 (9.5) -> 2.6 (9.5) MB, 5.0 / 0.0 ms [allocation failure] [**promotion limit reached**].
- [8664:01E15D10] 230 ms: Scavenge 4.5 (9.5) -> 2.6 (9.5) MB, 16.9 / 0.0 ms [allocation failure].
- [8664:01E15D10] 258 ms: Scavenge 4.5 (9.5) -> 2.6 (9.5) MB, 9.2 / 0.0 ms [allocation failure].
- [8664:01E15D10] 282 ms: Scavenge 4.5 (9.5) -> 2.6 (9.5) MB, 7.7 / 0.0 ms [allocation failure].
- [8664:01E15D10] 303 ms: Mark-sweep 4.5 (9.5) -> 2.6 (9.5) MB, 4.8 / 0.0 ms [allocation failure] [**promotion limit reached**].
- [8664:01E15D10] 339 ms: Scavenge 4.5 (9.5) -> 2.6 (9.5) MB, 19.2 / 0.0 ms [allocation failure].
- [8664:01E15D10] 365 ms: Scavenge 4.5 (9.5) -> 2.6 (9.5) MB, 8.7 / 0.0 ms [allocation failure].

--allow-natives-syntax

```
function factorial( n ) {
  return n === 1 ? n : factorial( --n );
}

var i = 0;

while ( i++ < 1e8 ) {
  factorial( 10 );
  i % 1e7 === 0 && %CollectGarbage( null );
}
```

d8 --allow-natives-syntax --trace-gc

- [21068:00294D18] 7 ms: Scavenge 1.7 (2.5) -> 1.6 (3.5) MB, 0.7 / 0.0 ms [allocation failure].
- [21068:00294D18] 8 ms: Scavenge 1.7 (3.5) -> 1.7 (4.5) MB, 0.6 / 0.0 ms [allocation failure].
- [21068:00294D18] 561 ms: Mark-sweep 2.8 (7.5) -> 2.6 (10.5) MB, 2.9 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 1060 ms: Mark-sweep 2.6 (10.5) -> 2.5 (9.5) MB, 3.8 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 1563 ms: Mark-sweep 2.5 (9.5) -> 2.5 (9.5) MB, 2.2 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 2063 ms: Mark-sweep 2.5 (9.5) -> 2.4 (9.5) MB, 1.9 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 2564 ms: Mark-sweep 2.4 (9.5) -> 2.4 (9.5) MB, 1.8 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 3058 ms: Mark-sweep 2.4 (9.5) -> 2.4 (6.5) MB, 1.9 / 0.0 ms [%CollectGarbage] [GC in old space requested].
- [21068:00294D18] 3561 ms: Mark-sweep 2.4 (6.5) -> 2.4 (6.5) MB, 1.9 / 0.0 ms [%CollectGarbage] [GC in old space requested].

Hidden Classes

- **function** Class(val) {
- **this**.prop = val;
- }

- **var** a = **new** Class('foo');
- **var** b = **new** Class('bar');

- print(**%HaveSameMap(a, b)**);

- b.prop2 = 'baz';

- print(**%HaveSameMap(a, b)**);

d8 --allow-natives-syntax

- true
- false

尾调用优化

- Syntactic Tail Calls (STC)

- https://github.com/tc39/proposal-ptc-syntax

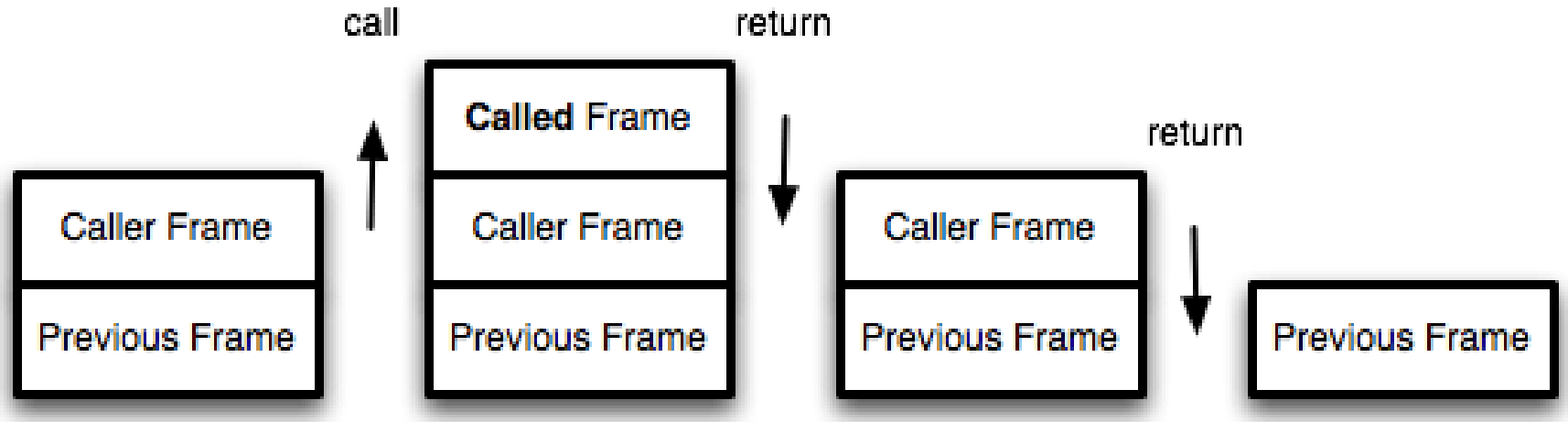
累加

```
function sum(x) {
        if (x === 1) {
            return x;
        }
        return x + sum(x - 1);
}
```

sum(5)

- sum(5)
- 5 + sum(4)
- 5 + (4 + sum(3))
- 5 + (4 + (3 + sum(2)))
- 5 + (4 + (3 + (2 + sum(1))))
- 5 + (4 + (3 + (2 + 1)))
- 5 + (4 + (3 + 3))
- 5 + (4 + 6)
- 5 + 10
- 15

调用栈


call
return
Called Frame
Caller Frame
Previous Frame
Caller Frame
Previous Frame
return
Caller Frame
Previous Frame
Previous Frame

存在的问题

```
function foo(n) {
  return bar(n*2);
}

function bar() {
  throw new Error();
}

try {
  foo(1);
} catch(e) {
  print(e.stack);
}
```

- 不使用**PTC：**
- Error
- at bar
- at foo
- at Global Code

- 使用**PTC：**
- Error
- at bar
- at Global Code

解决方式：显式指定

- return continue
- !return
- #function()

<questions />

by @jjc