Programming Languages on the Web, Now and the Future

Michael Lippautz / mlippautz@
Google

Programming Language Implementation Summer School
Bertinoro, Italy, 2023

The web

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <script type="text/javascript">
      async function fetchAndLogMovies() {
        const response =
            await fetch("http://example.com/movies.json");
        const movies = await response.json();
        console.log(movies);
      }
    </script>
  </head>
  <body onload="fetchAndLogMovies()">
  </body>
</html>
```

The web

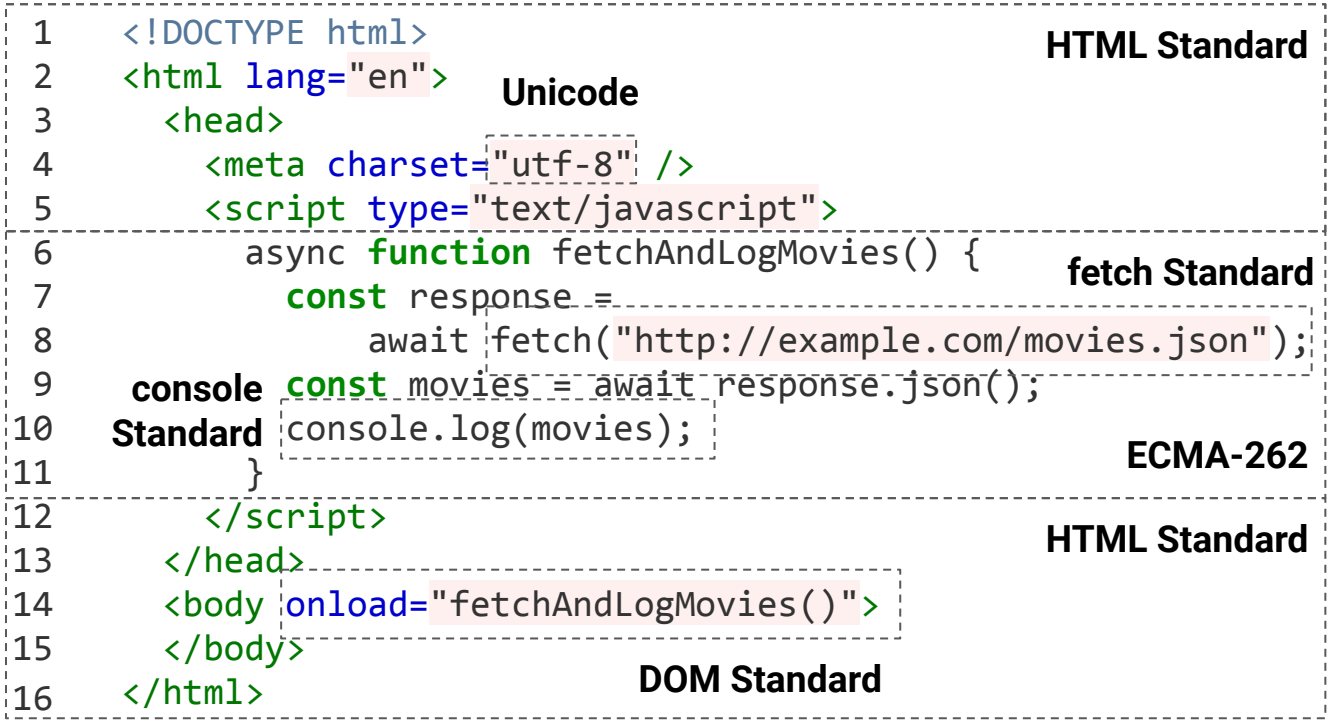
```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8" />
5       <script type="text/javascript">
6         async function fetchAndLogMovies() {
7           const response =
8               await fetch("http://example.com/movies.json");
9           const movies = await response.json();
10          console.log(movies);
11        }
12      </script>
13    </head>
14    <body onload="fetchAndLogMovies()">
15    </body>
16  </html>
```

HTML Standard

ECMA-262

HTML Standard

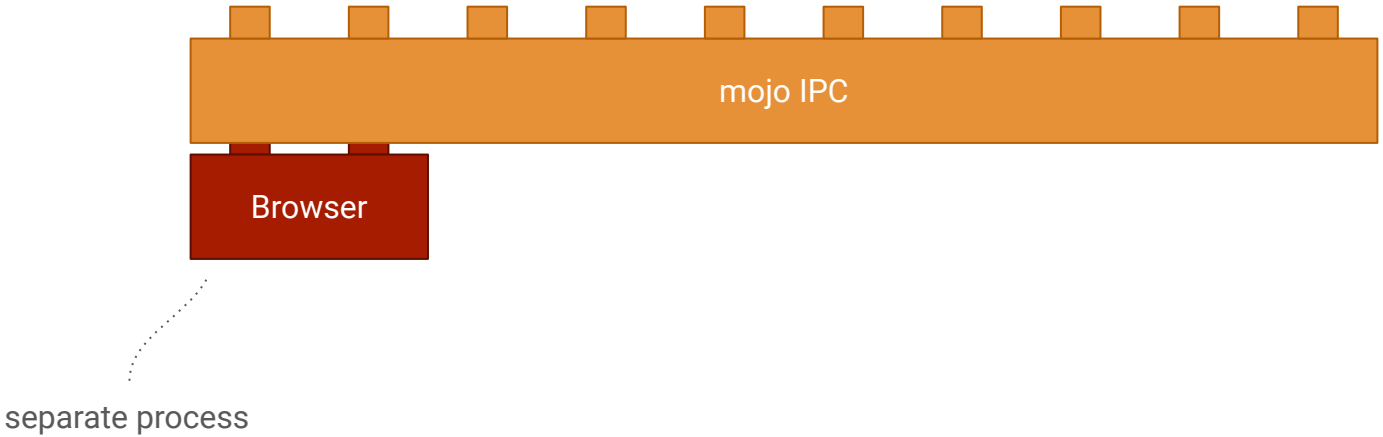
The web



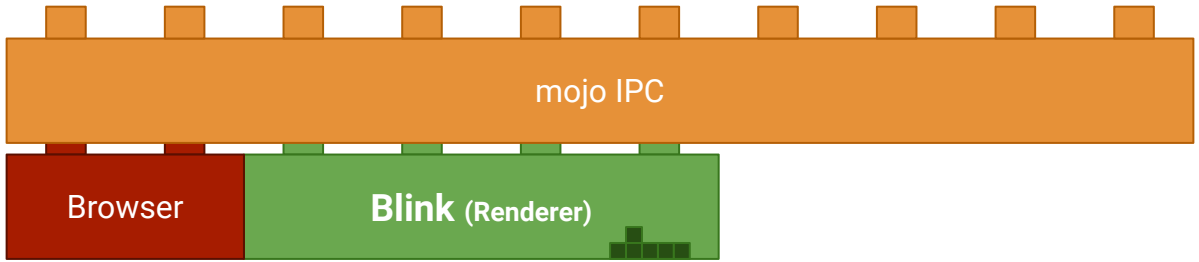

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <script type="text/javascript">
      async function fetchAndLogMovies() {
        const response =
            await fetch("http://example.com/movies.json");
        const movies = await response.json();
        console.log(movies);
      }
    </script>
  </head>
  <body onload="fetchAndLogMovies()">
  </body>
</html>
```

… and many more (CSS, WebGL/WebGPU, WebAssembly, …)

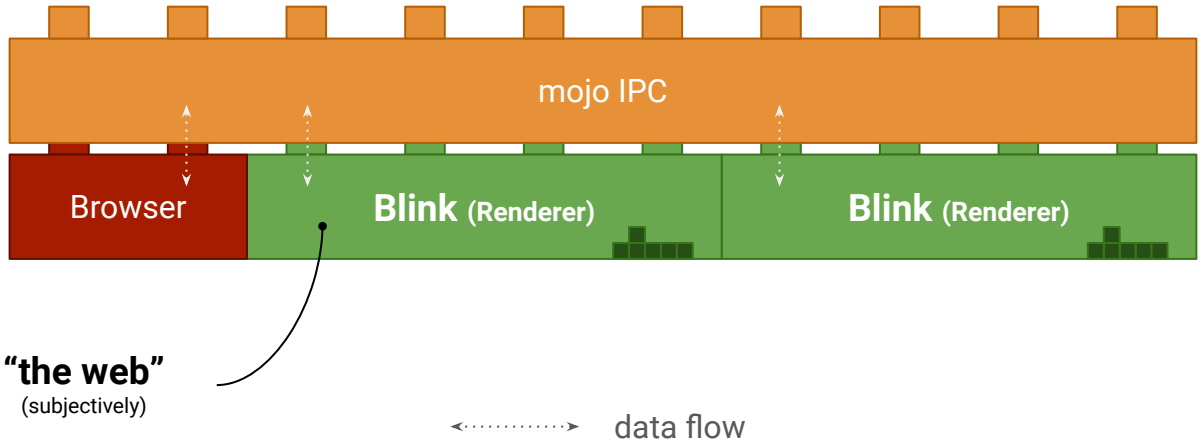
Chrome architecture



Chrome architecture

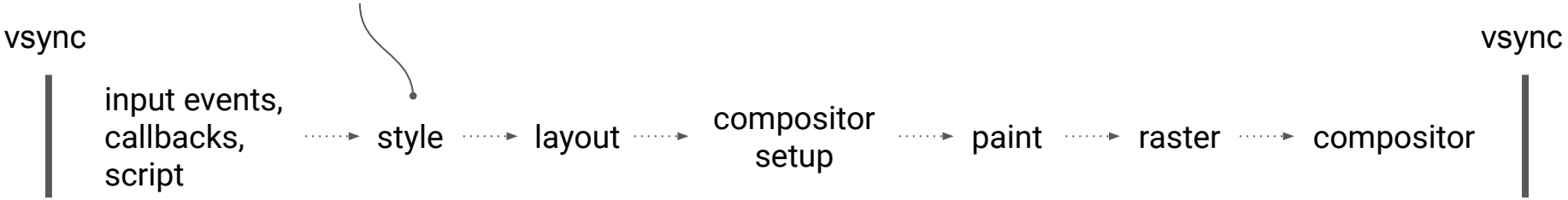


Chrome architecture

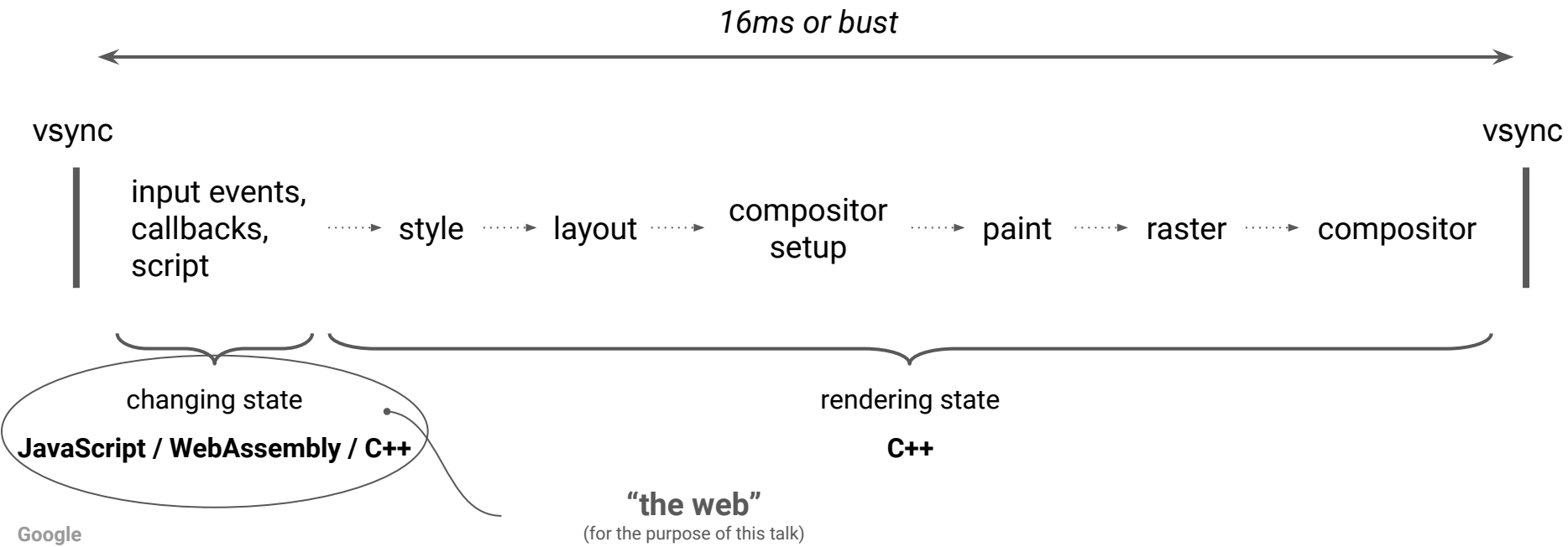


Interlude: Rendering 101

https://faultlore.com/blah/text-hates-you/



Interlude: Rendering 101



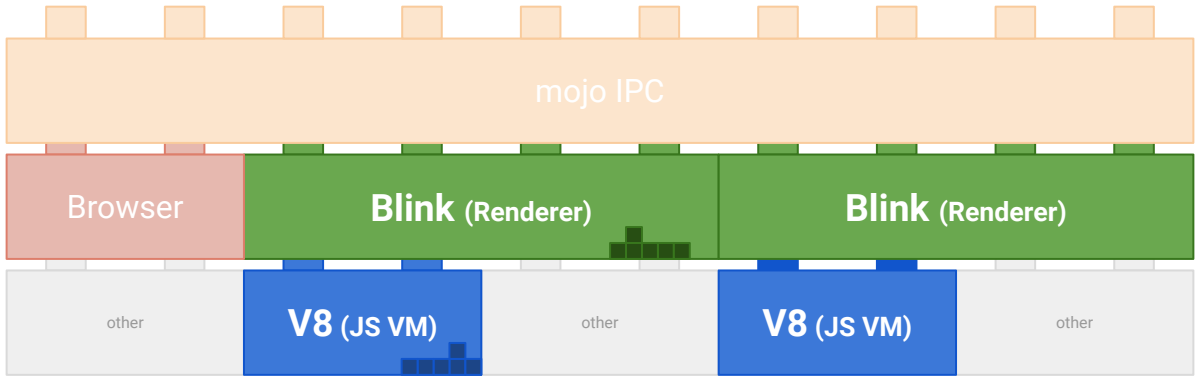
The web environment

- Flexible
  - Hardware (Phones, Laptops, Desktops, Fridges, …)
  - User programs (simple webpage, Photoshop, Figma, Docs, Gmail, …)
- Backwards compatible
  - https://www.spacejam.com/1996/
- Hostile: Adversaries cause real damage
  - https://blog.google/threat-analysis-group/countering-threats-north-korea/

Performance profile:

- Latency critical (16ms)
- Startup in the order of seconds

Chrome architecture



A virtual machine for JavaScript and WebAssembly

Basic execution pipeline

1. Fetch/load
2. Parsing
3. Interpreter
4. Type feedback

JIT compilation

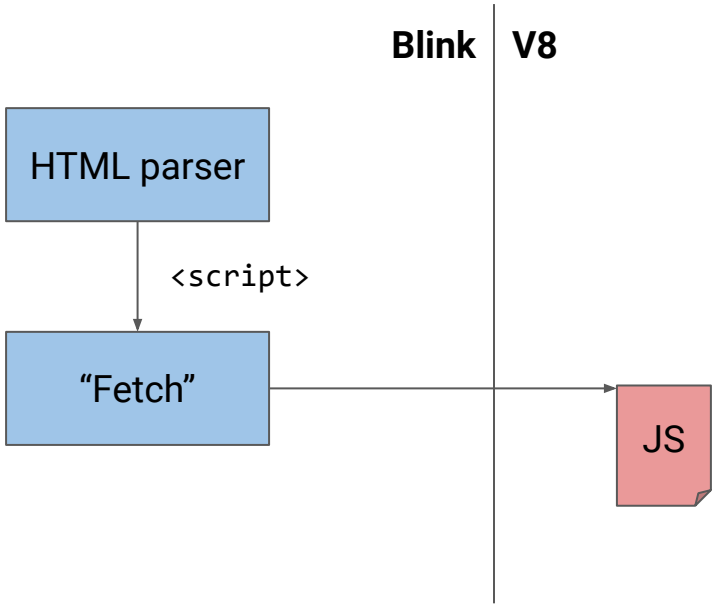
5. Sparkplug
6. Maglev
7. TurboFan

Life of a

```
<script src="index.js"></script>
```

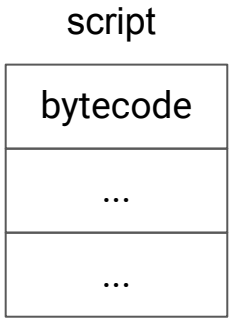
There exist various versions of such talks available in Chrome University on Youtube. ***This one is new!***

Acquire the script



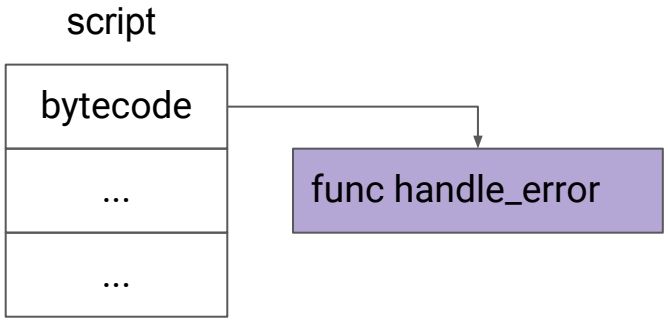
Parse and compile: An overview

```
function handle_error() { /* ... */ }

function event_handler() {
  /* ... */
  if (error) handle_error();
  /* ... */
}

(function setup() { /* ... */ })();

document.addEventHandler(
    event, event_handler);
```



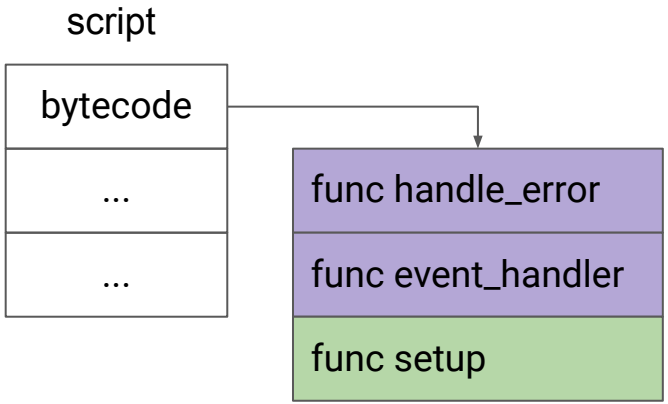
Parse and compile: An overview

```
function handle_error() { /* ... */ }

function event_handler() {
  /* ... */
  if (error) handle_error();
  /* ... */
}

(function setup() { /* ... */ })();

document.addEventHandler(
    event, event_handler());
```



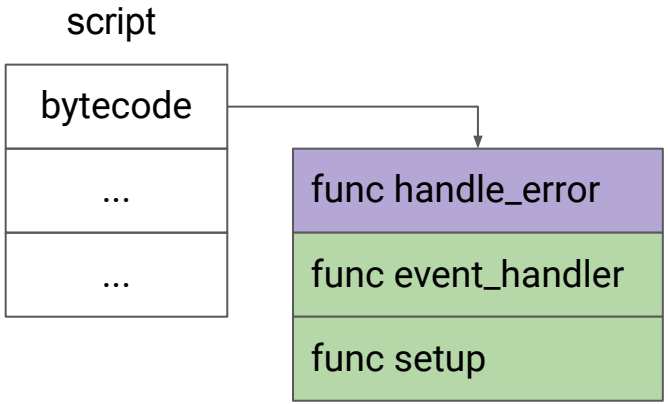
Parse and compile: An overview

```
function handle_error() { /* ... */ }

function event_handler() {
  /* ... */
  if (error) handle_error();
  /* ... */
}

(function setup() { /* ... */ })();

document.addEventHandler(
    event, event_handler);
```



Parse and compile: An overview

```
function handle_error() { /* ... */ }

function event_handler() {
  /* ... */
  if (error) handle_error();
  /* ... */
}

(function setup() { /* ... */ })();

document.addEventHandler(
    event, event_handler);
```



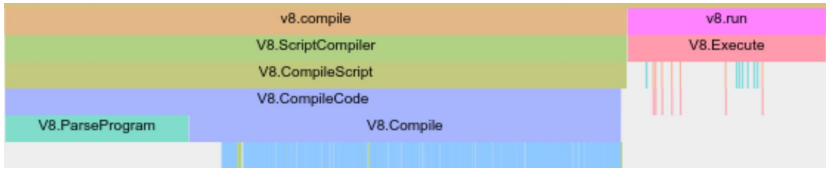
Caching

Parse and compile often unnecessary!

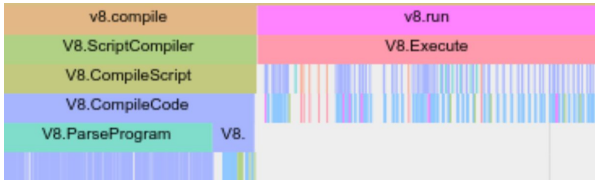
- V8: In memory cache in case we see the same script again
- Blink: On disk cache storing compiled data

Performance

time (order of ms)



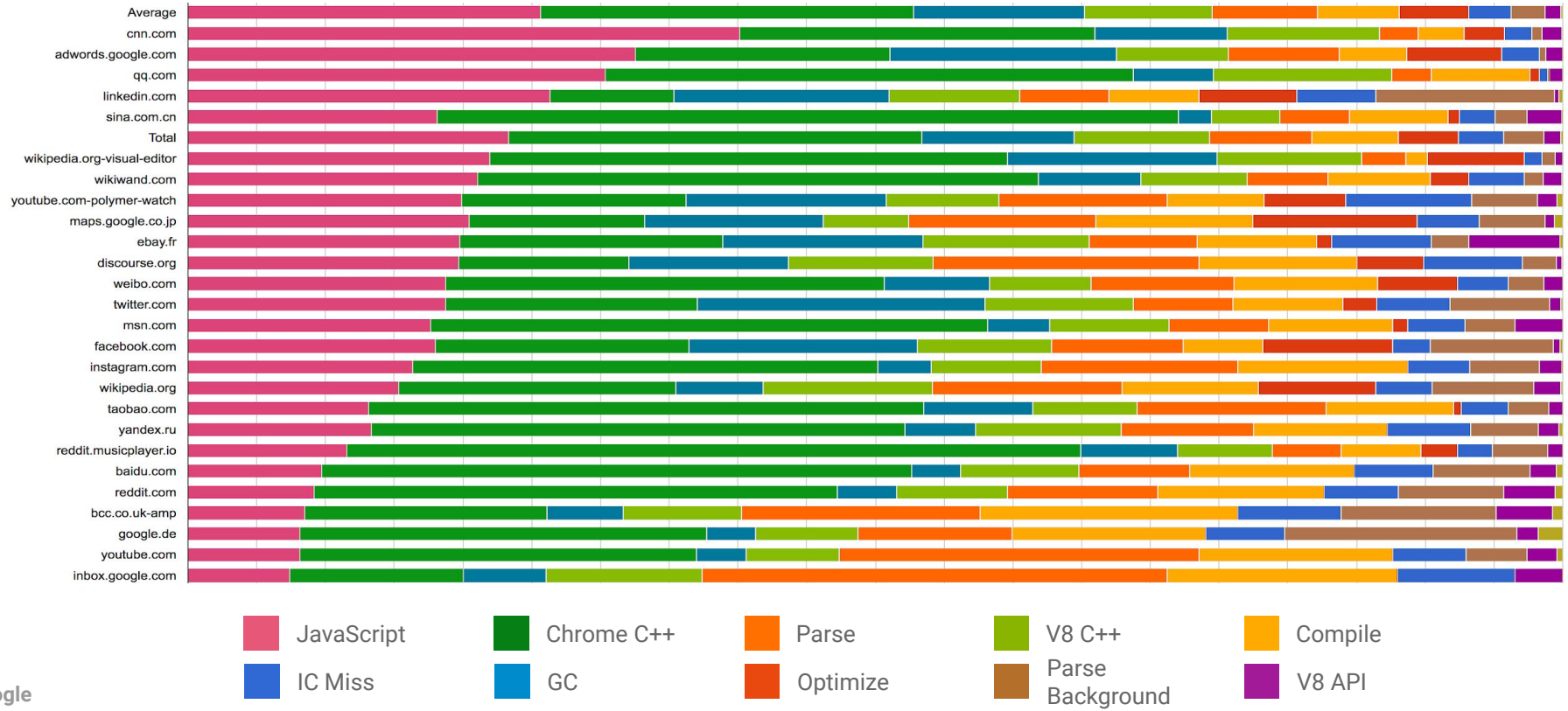
full parse and compile

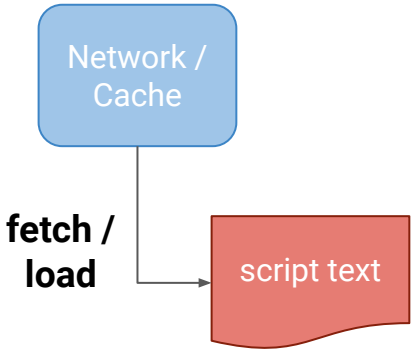


lazy compile

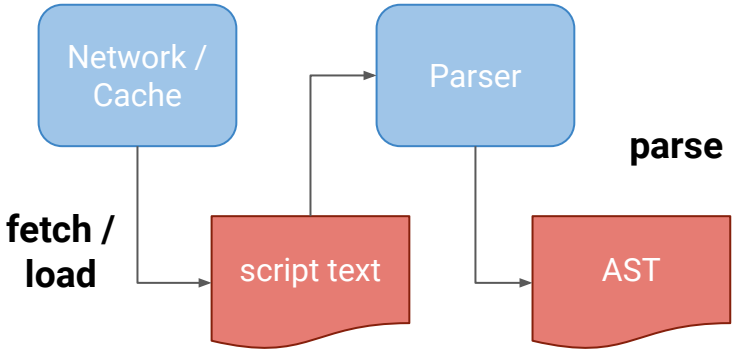
cache

Where is time spent these days?





V8 architecture (tbc)



V8 architecture (tbc)

Parsing in V8

- Pre-parse and full parse
- Recursive descent parser
  - (non-terminal symbols are delegated to matching procedures)

- Input: Tokens
- Output: abstract syntax tree (AST)

Parsing in V8: JS grammar is ambiguous

- No rewinding
- No unbounded look ahead

CoverParenthesizedExpressionAndArrowParameterList

```
let x = (
```

```
let x = (a, b) => { return a + b };
```

```
let x = (a, 3);
```

Cover grammar: Permissive symbols that keep internal state to signal when branching is possible

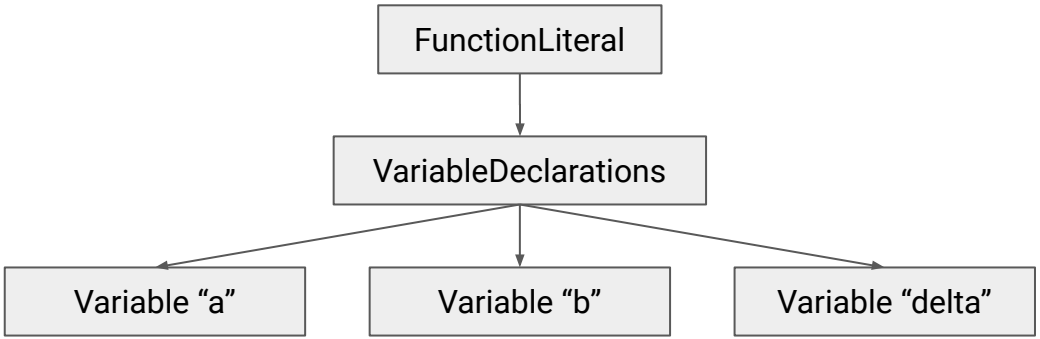
https://v8.dev/blog/understanding-ecmascript-part-4

Parsing: Creating an Abstract Syntax Tree (AST)

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```

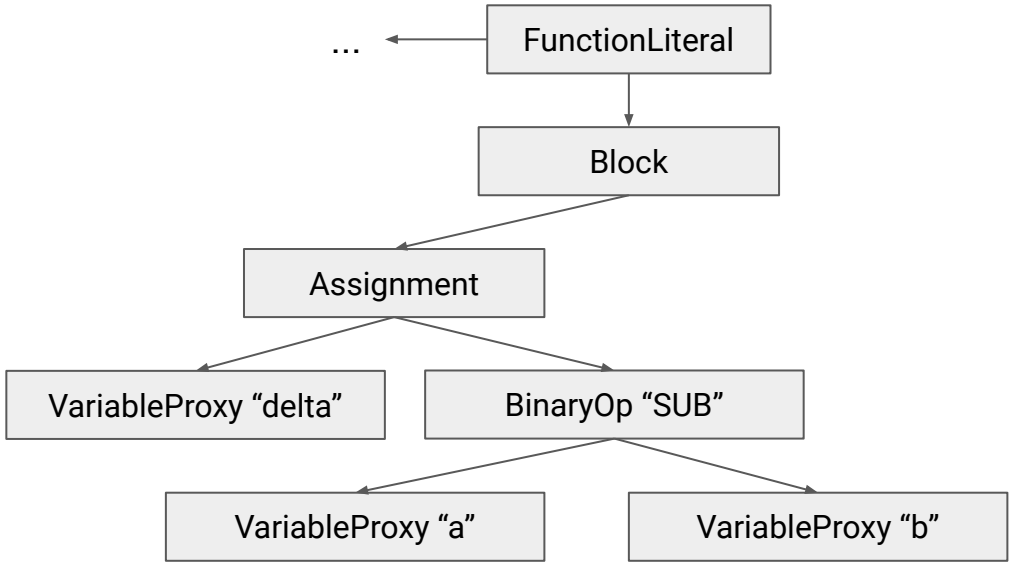
Parsing: Creating an Abstract Syntax Tree (AST)

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```



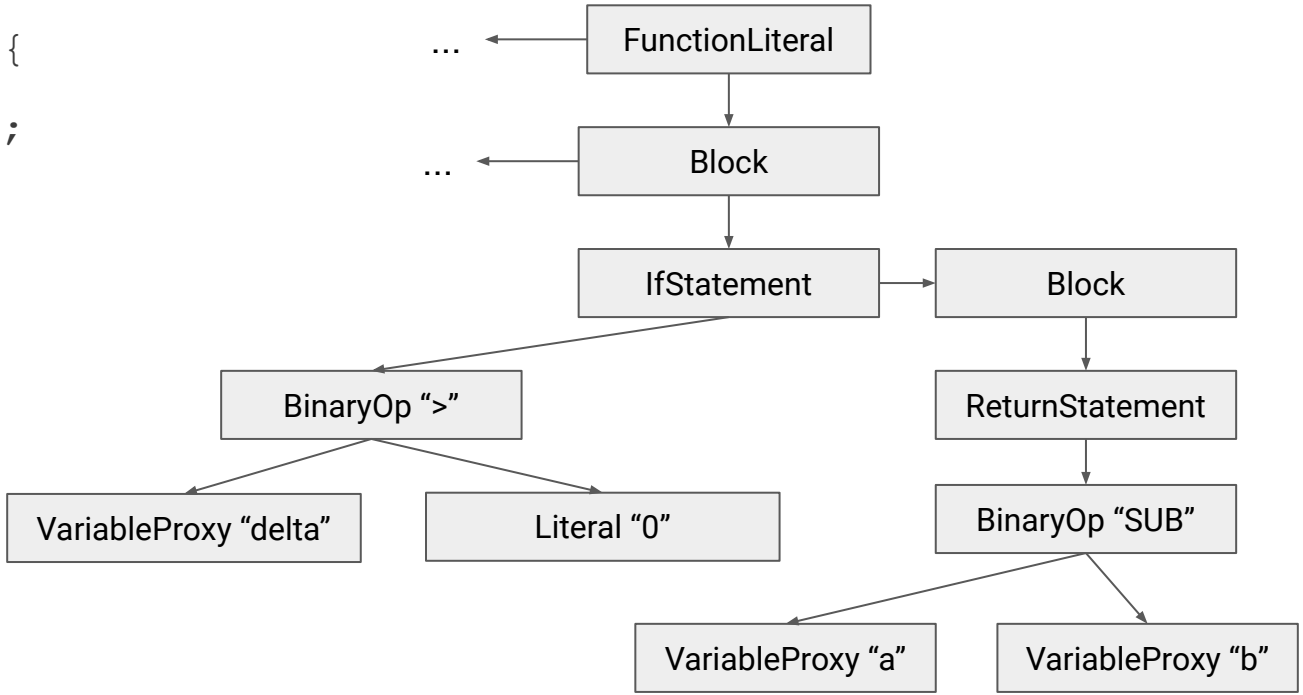
Parsing: Creating an Abstract Syntax Tree (AST)

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```



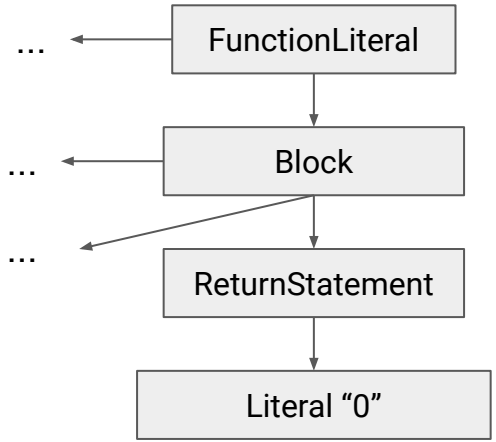
Parsing: Creating an Abstract Syntax Tree (AST)

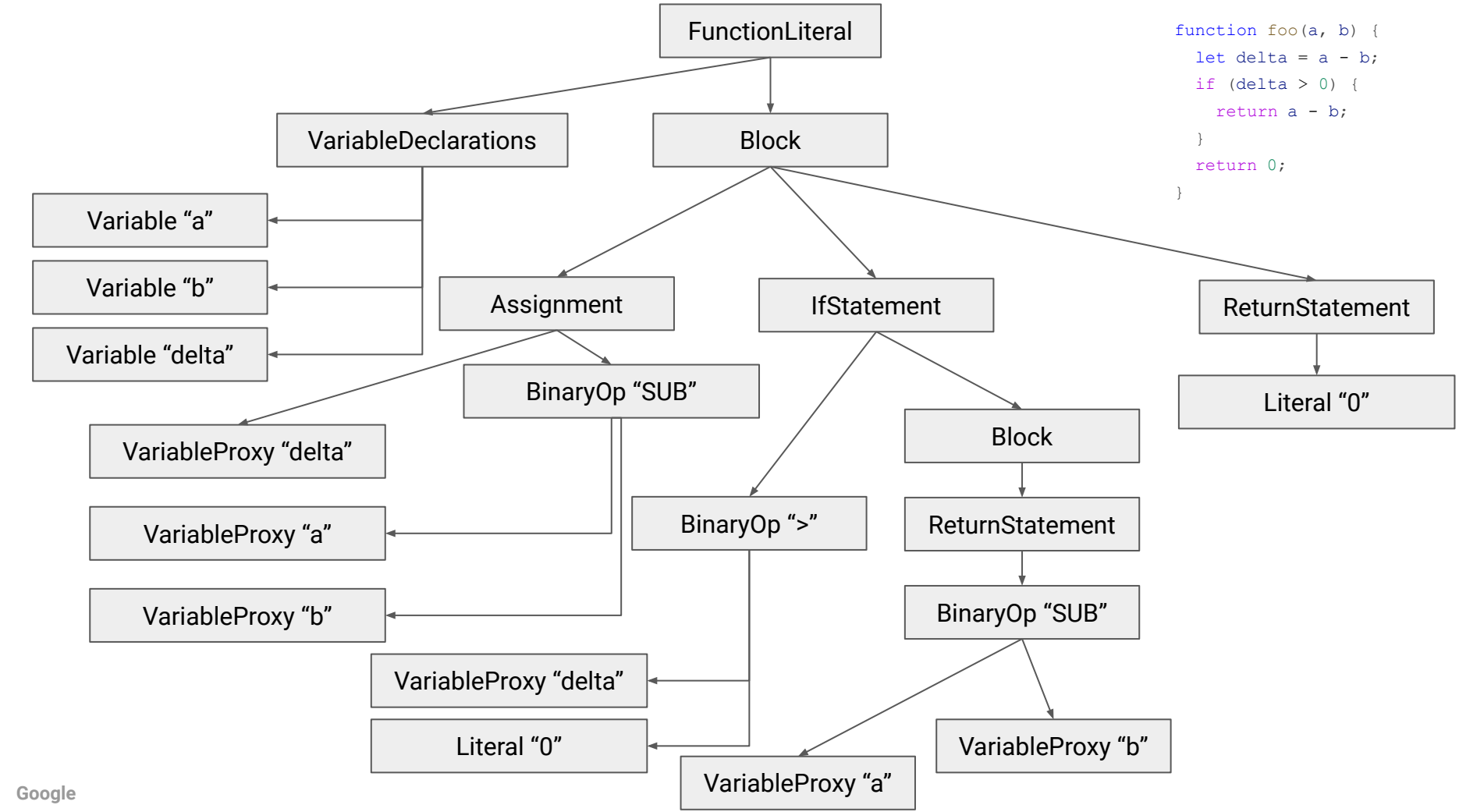
```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```



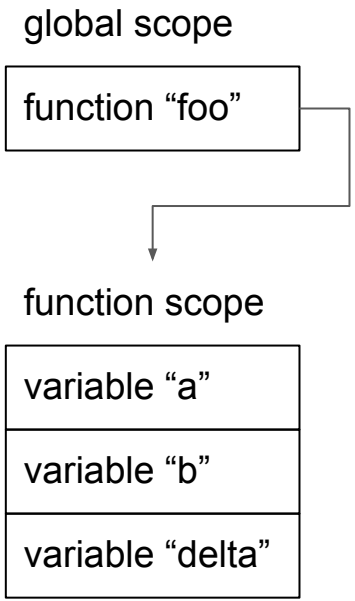
Parsing: Creating an Abstract Syntax Tree (AST)

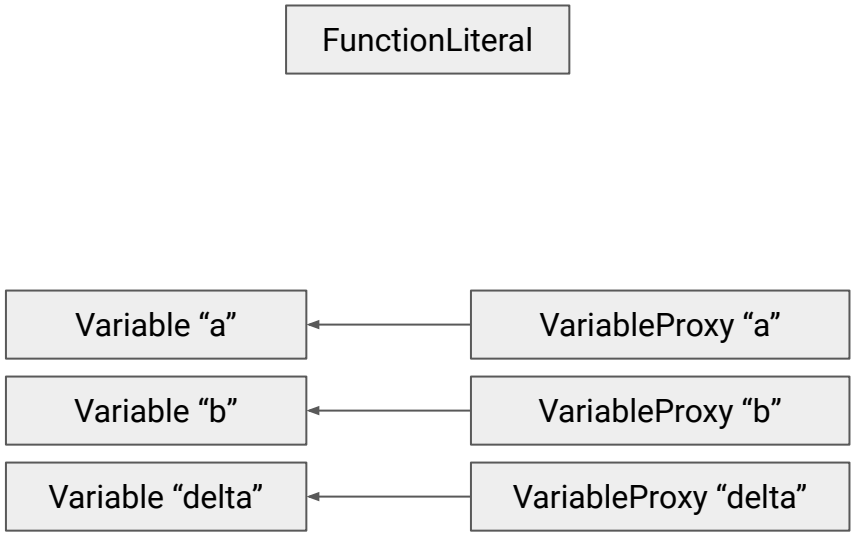
```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```



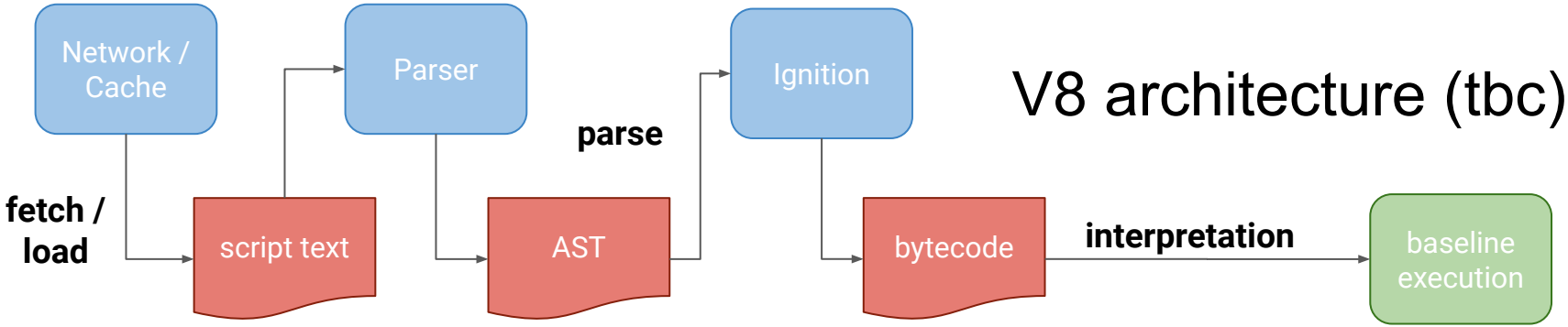

FunctionLiteral
VariableDeclarations
Block
Variable “a”
Variable “b”
Variable “delta”
Assignment
IfStatement
ReturnStatement
BinaryOp “SUB”
Literal “0”
VariableProxy “delta”
Block
VariableProxy “a”
BinaryOp “>”
ReturnStatement
VariableProxy “b”
BinaryOp “SUB”
VariableProxy “delta”
Literal “0”
VariableProxy “b”
VariableProxy “a”
function foo(a, b) {
let delta = a - b;
if (delta > 0) {
return a - b;
}
return 0;
}

Scope analysis





V8 architecture (tbc)



Ignition interpreter

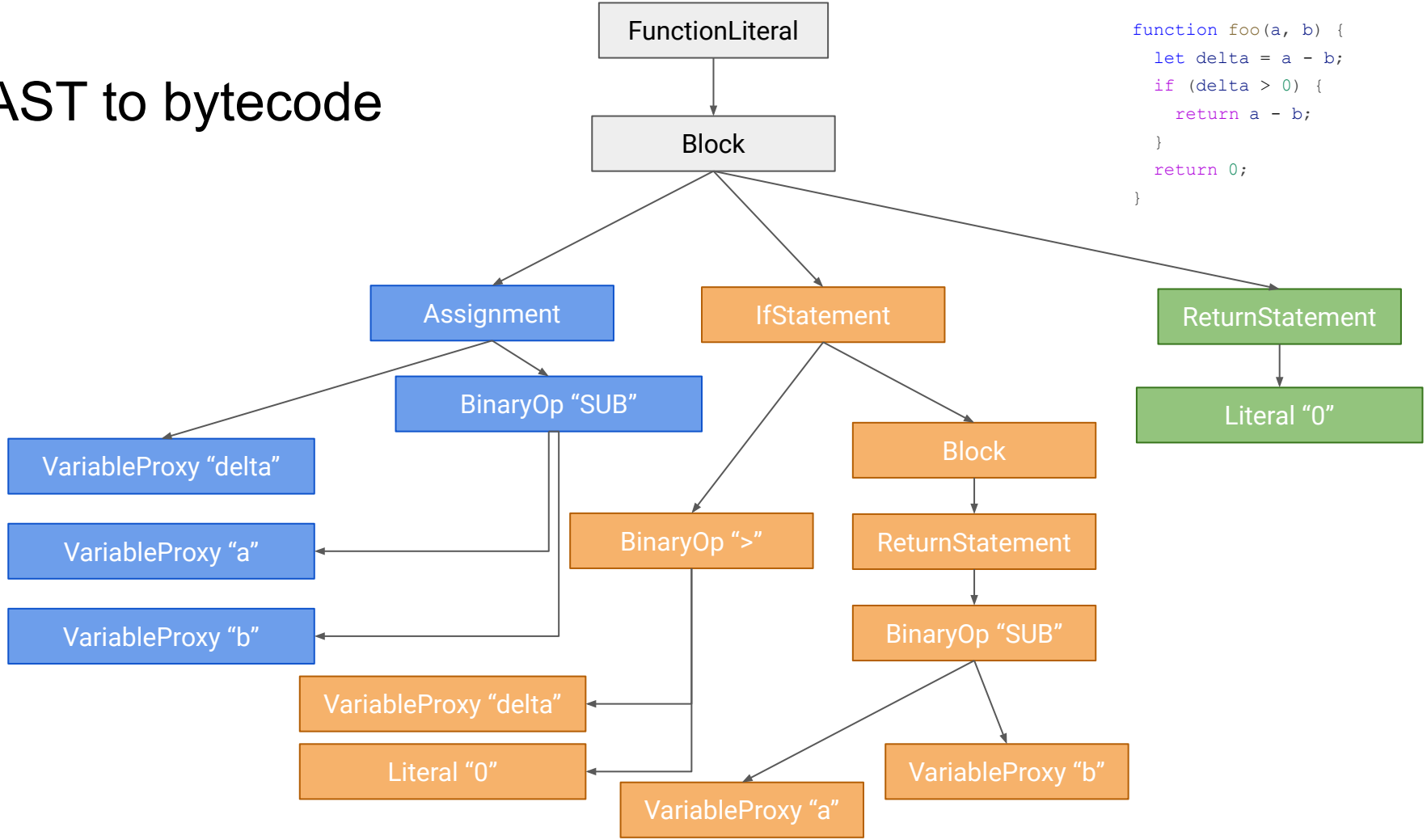
Bytecode generation

- Bytecode is generated by walking the AST
  - (Source positions are recorded for debugging purposes)
- Bytecode is used as input for later compiler tiers
  - No need to build an AST again
- Bytecode operation match JS abstraction
  - E.g., “load property x from object y”
  - Avoids dispatch overhead

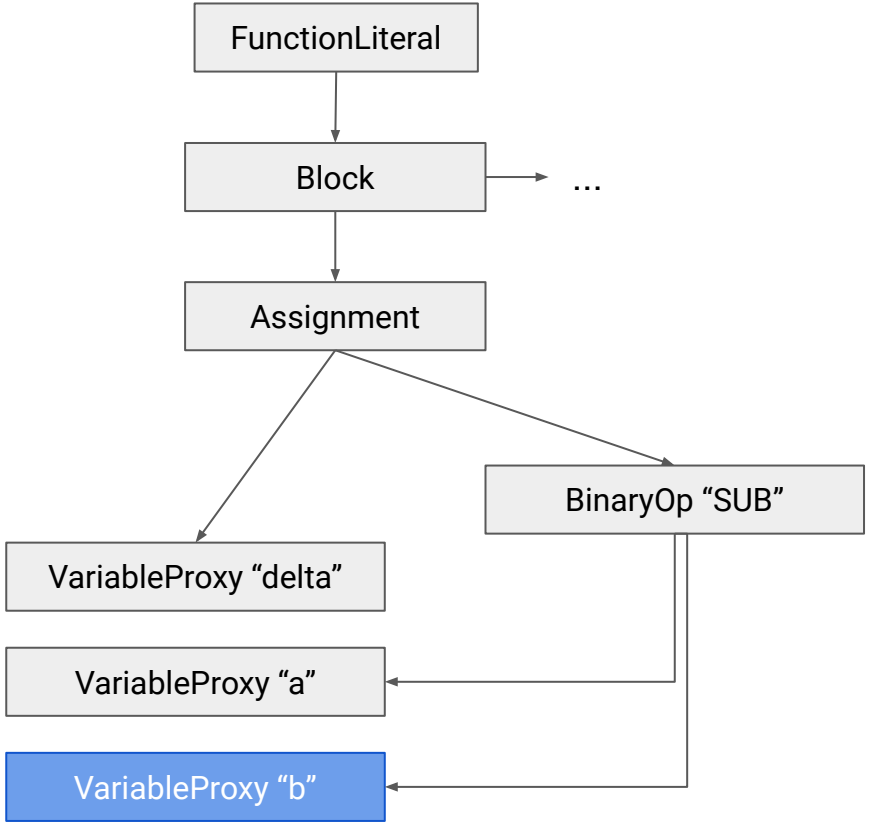
Interpretation

- Interpreter uses indirect threaded dispatch
  - Each bytecode handler ends with dispatch of next bytecode
- Register machine
  - Implicit accumulator register used in most bytecodes
  - Accumulator gets a lot of benefits of stack-based interpretation
- Implemented in compiler backend (to avoid writing it in assembly for 9 architectures)

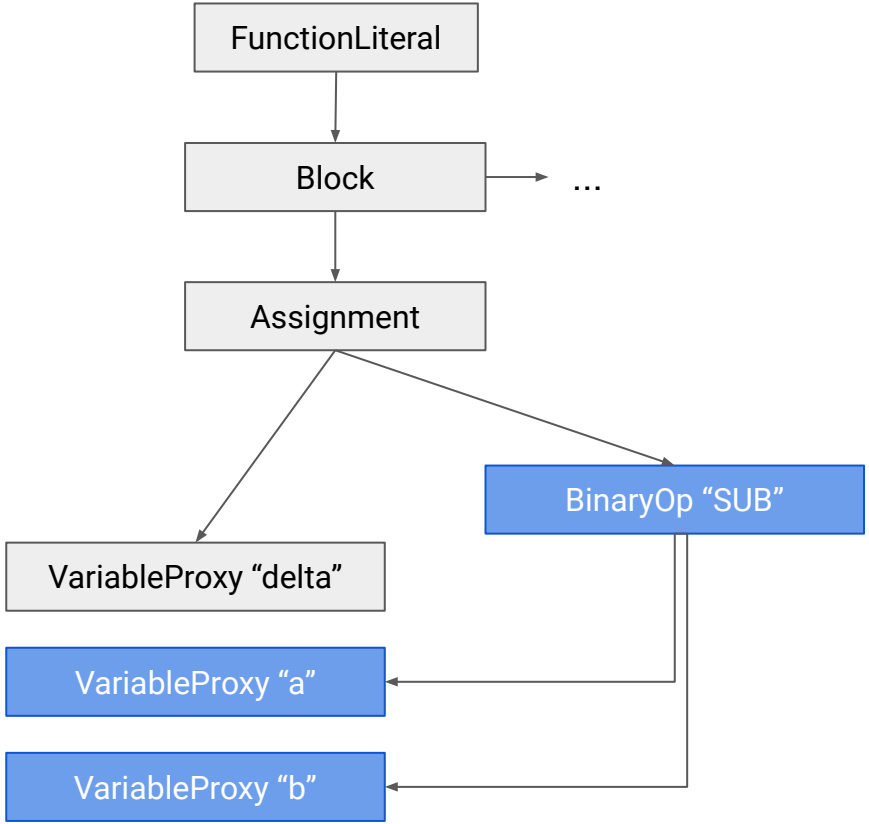
AST to bytecode



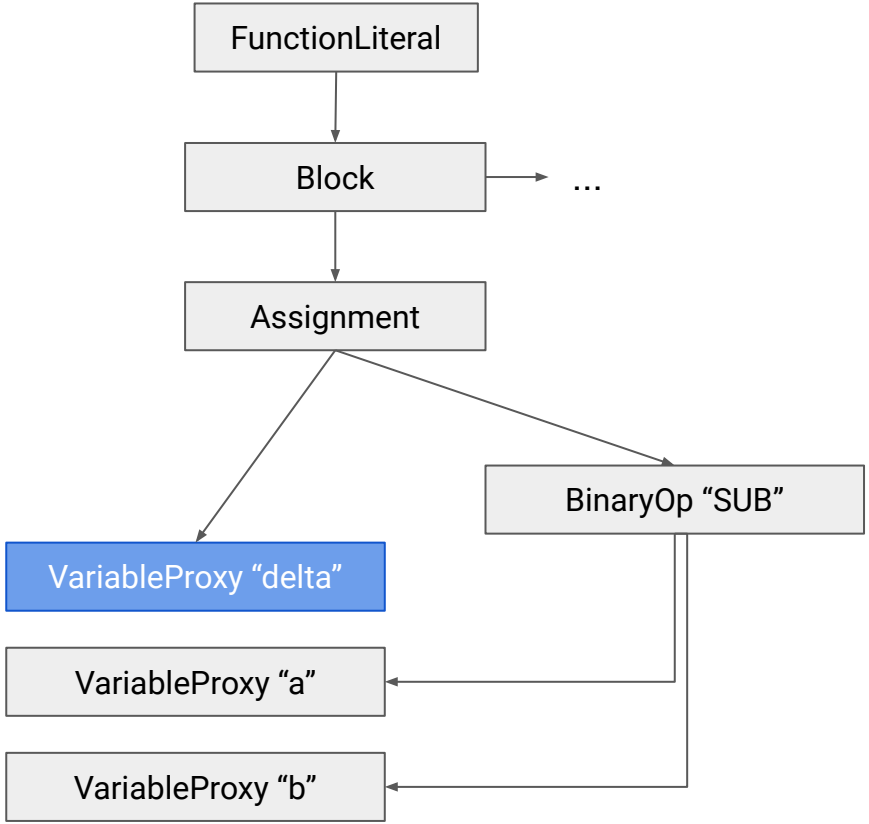

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
```


FunctionLiteral
Block
...
Assignment
BinaryOp “SUB”
VariableProxy “delta”
VariableProxy “a”
VariableProxy “b”

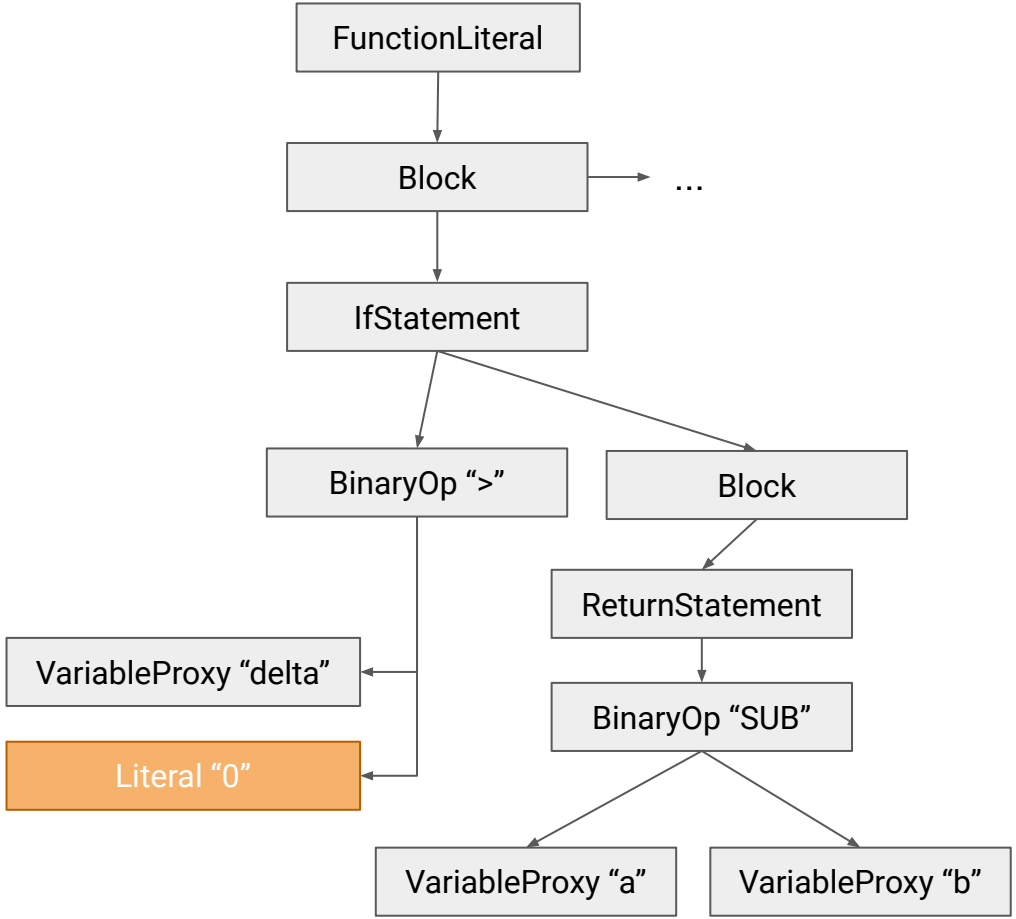
```
0 Ldar a1
```



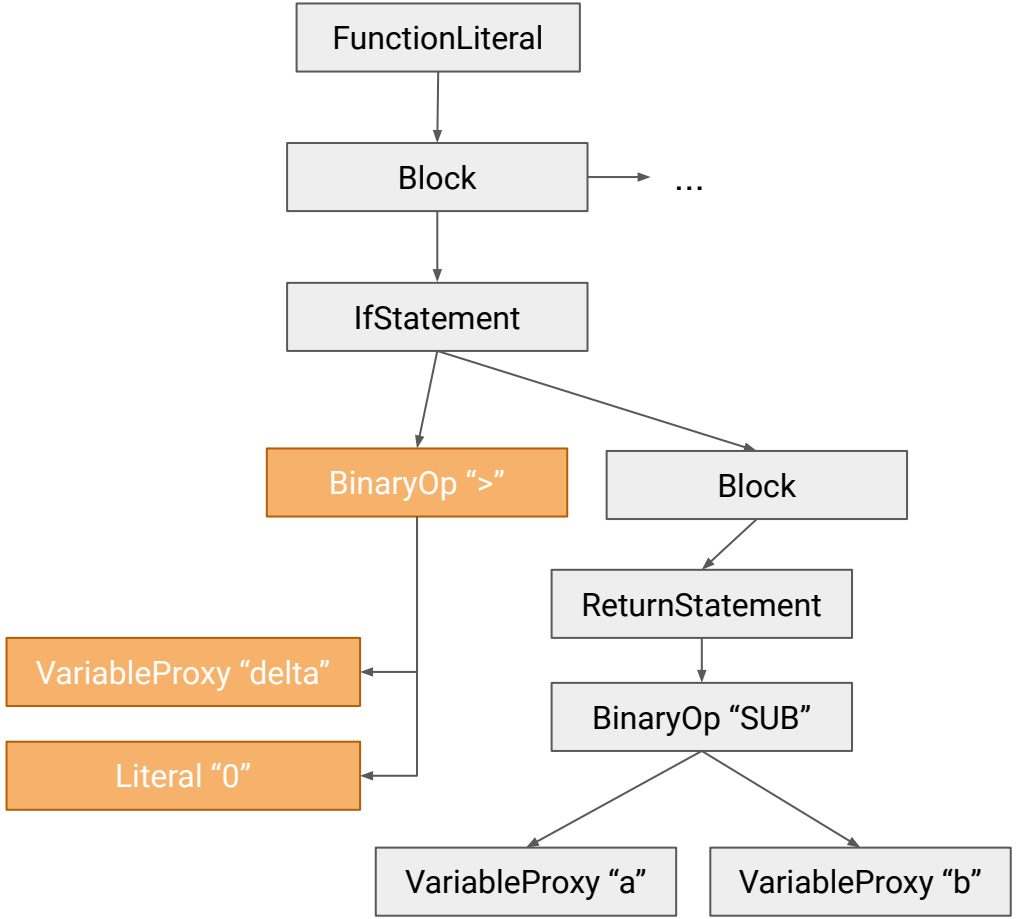

```
0 Ldar a1
2 Sub a0
```



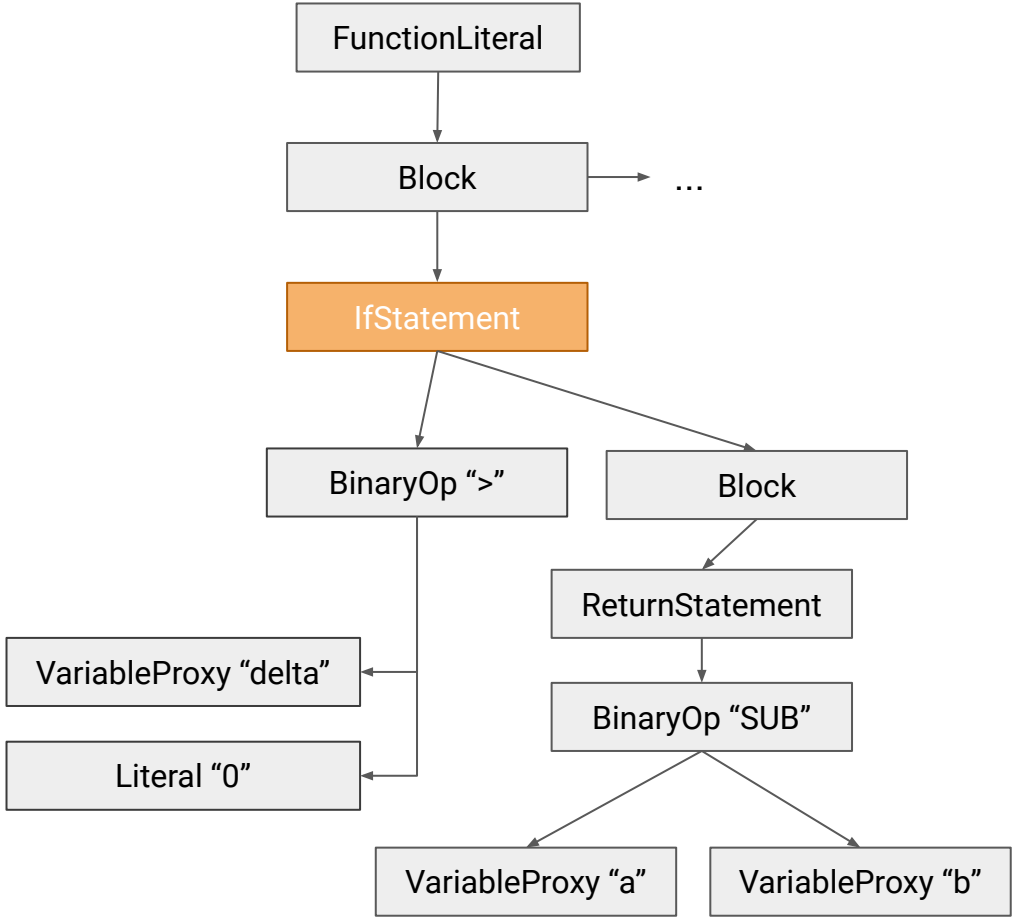

```
0 Ldar a1
2 Sub a0
5 Star0
```



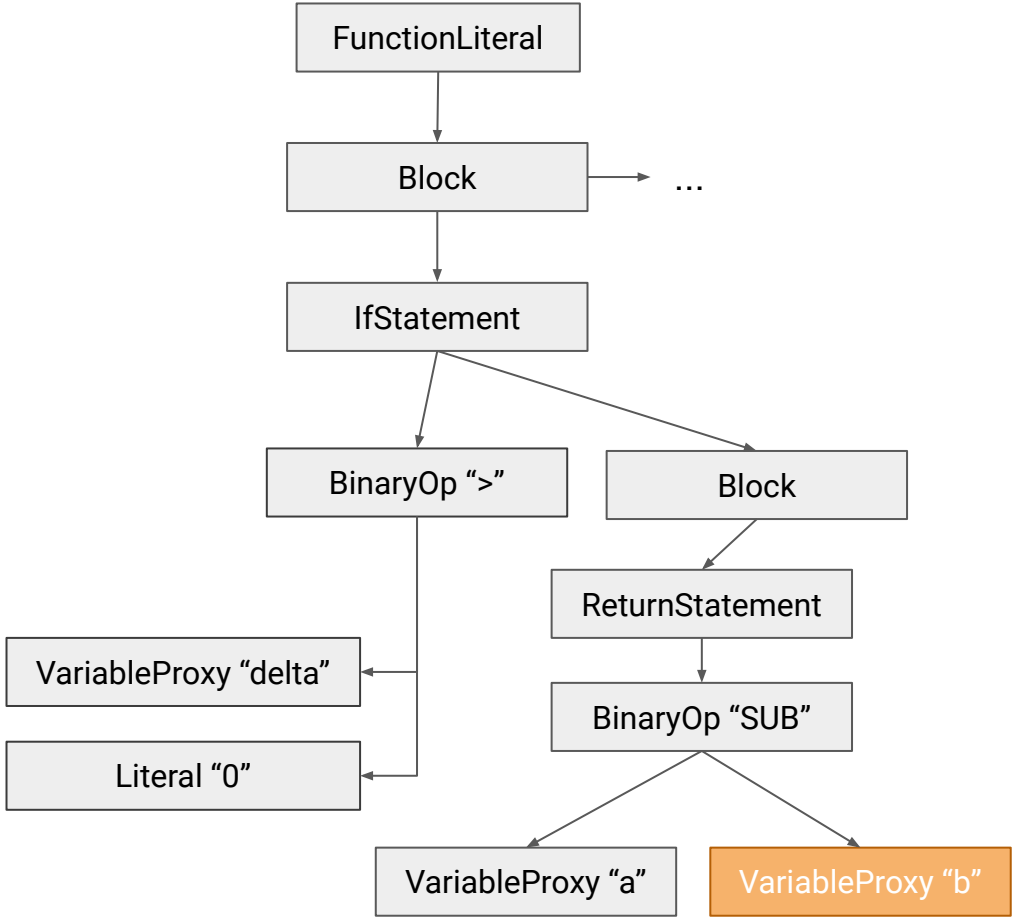

```
0 Ldar a1
2 Sub a0
5 Star0
6 LdaZero
```



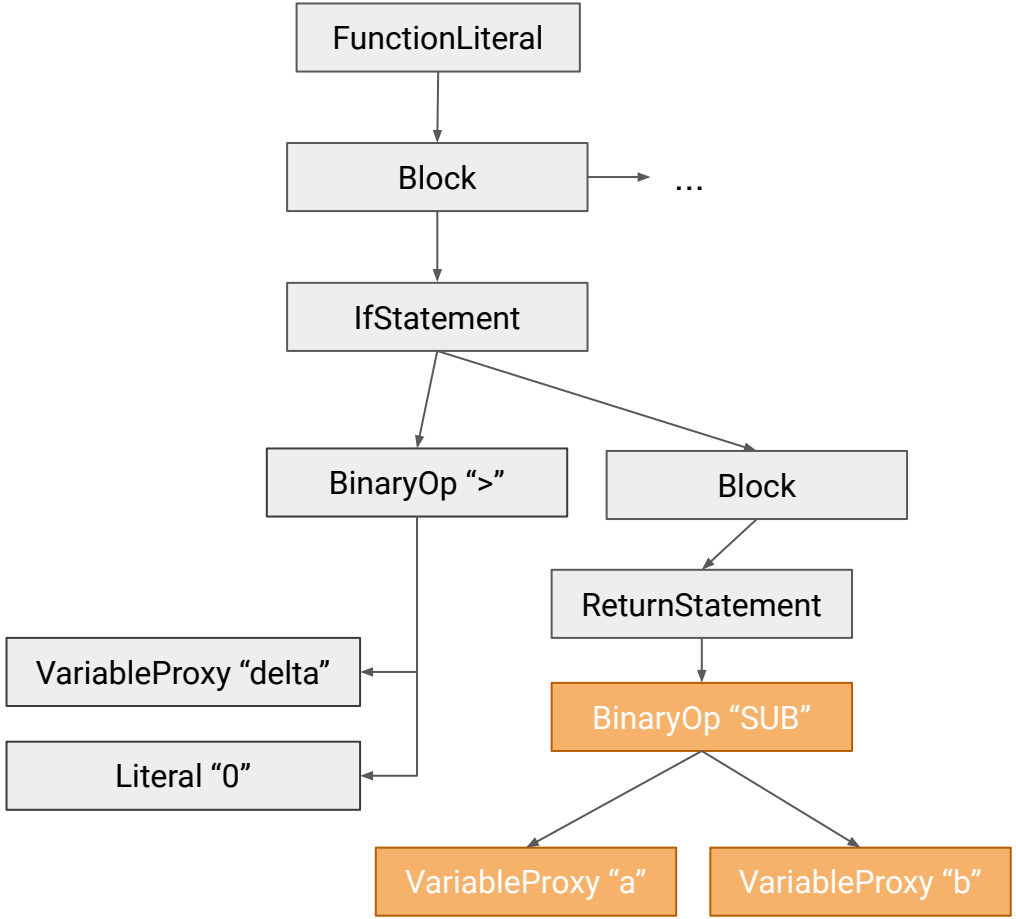

```
0 Ldar a1
2 Sub a0
5 Star0
6 LdaZero
7 TestGreaterThan r0
```



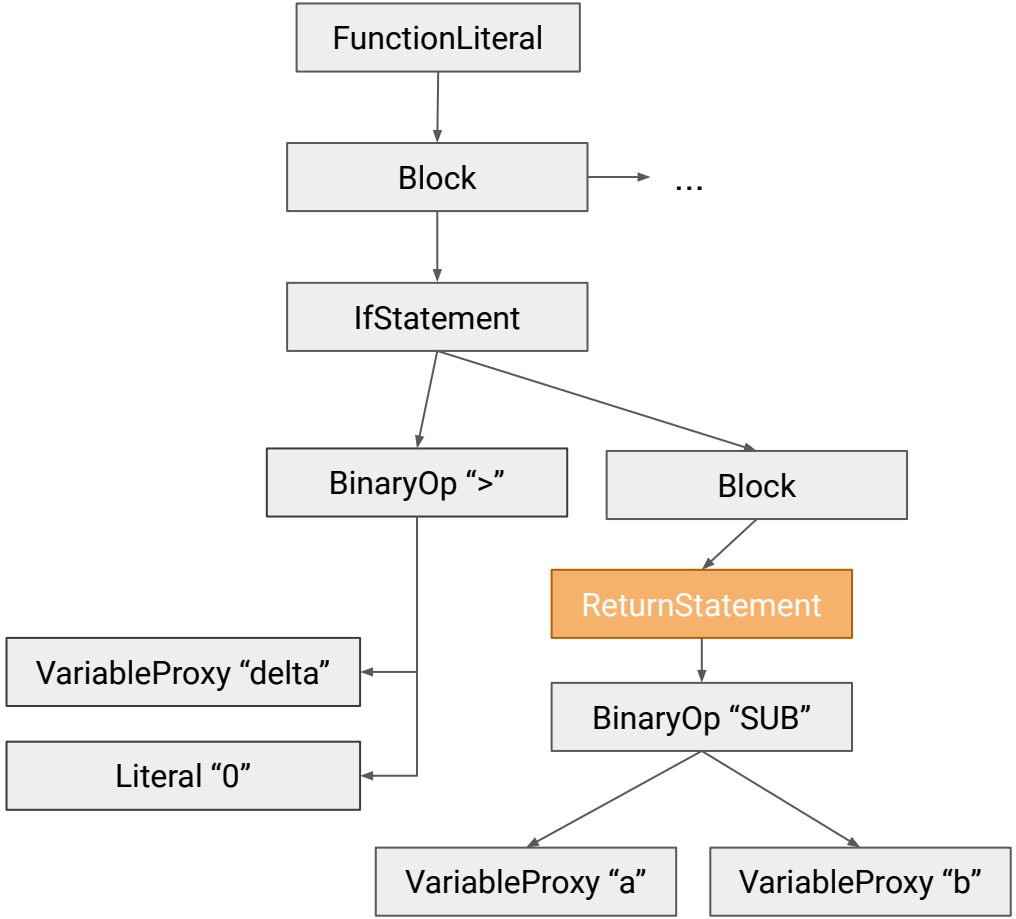

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
```



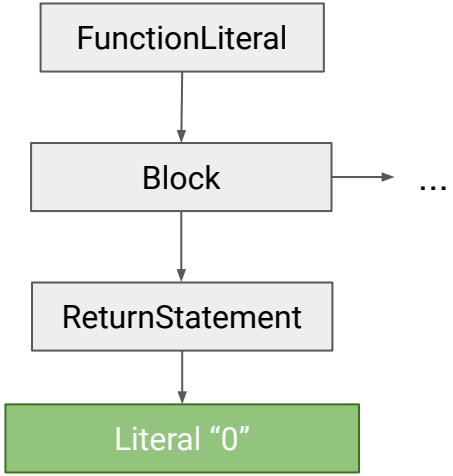

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
```



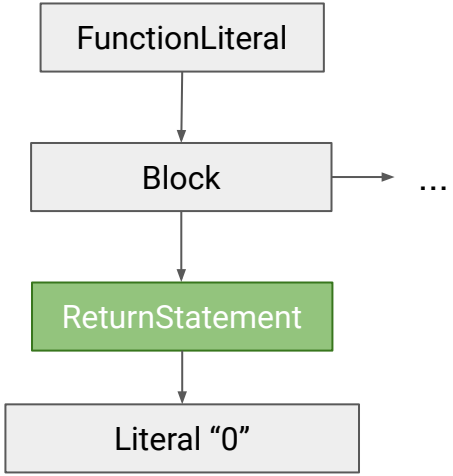

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
```




```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
```




```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
```




```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	undefined
accumulator	undefined

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	undefined
accumulator	1

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	undefined
accumulator	2

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	2

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	0

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	true

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	true

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	1

Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	2

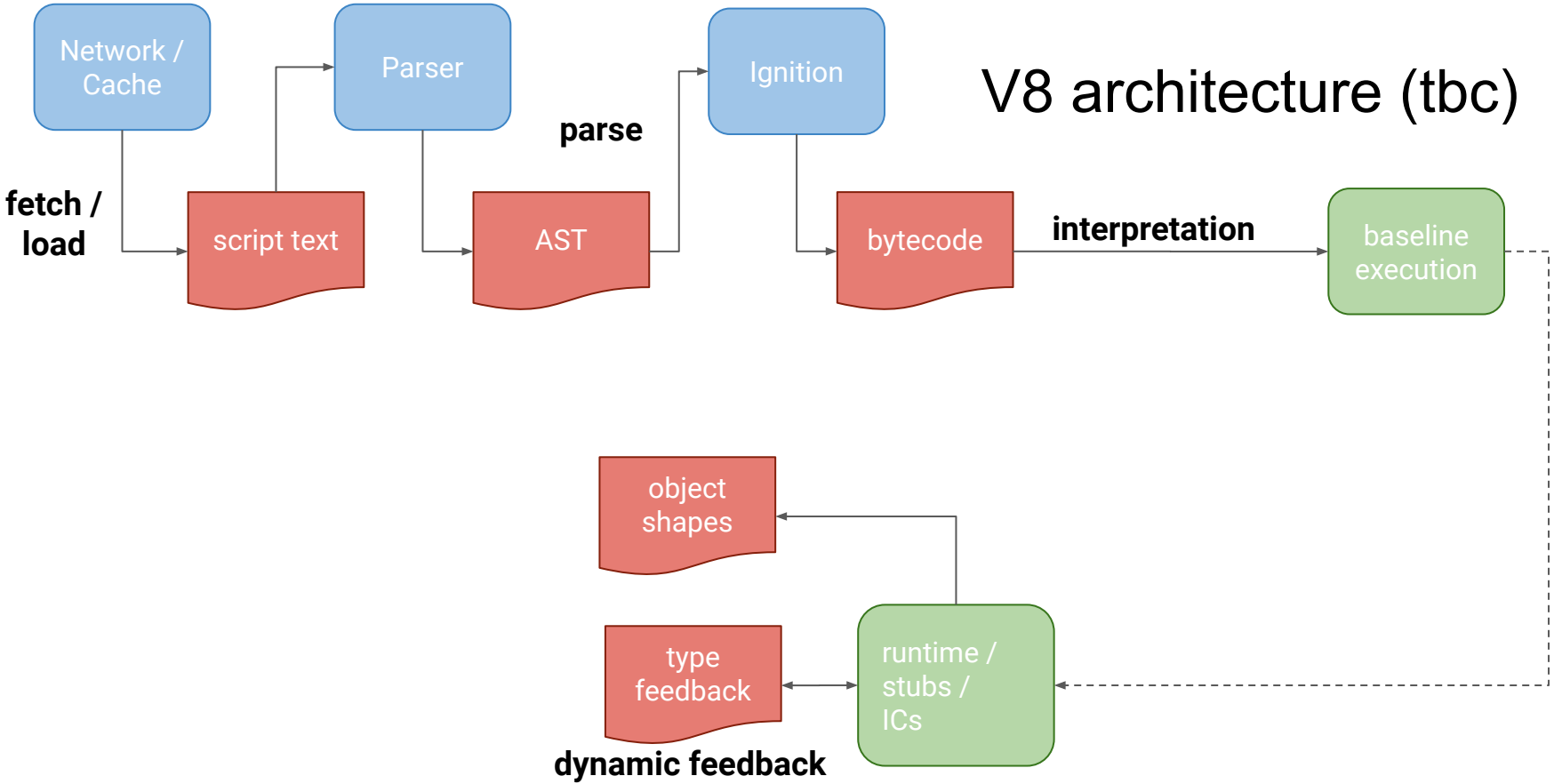
Interpreting bytecode

```
function foo(a, b) {
  let delta = a - b;
  if (delta > 0) {
    return a - b;
  }
  return 0;
}
foo(3, 1);
```

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

a0 [a]	3
a1 [b]	1
r0 [local]	2
accumulator	2

V8 architecture (tbc)



JavaScript “+”

```
function add(a, b) {
  return a + b;
}
```

JavaScript “+”

```
function add(a, b) {
  return a + b;
}
add(1, 2);              // 3
add(1.2, 3.14);         // 4.34
add(“hello”, “world”); // “helloworld”
add(1, true);           // 2
add(“foo”, true);       // “footrue”
var bar = {toString:() => “bar”};
add(“foo”, bar);        // “foobar”
```

Integer addition

Floating point addition

String addition

Type coercion

toString() / valueOf()

JavaScript “+” – Semantics

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the Note below 12.7.5.

NOTE 1 No hint is provided in the calls to ToPrimitive in steps 7 and 9. All standard objects except Date objects handle the absence of a hint as if the hint Number were given; Date objects handle the absence of a hint as if the hint String were given. Exotic objects may handle the absence of a hint in some other manner.

NOTE 2 Step 11 differs from step 5 of the Abstract Relational Comparison algorithm (7.2.11), by using the logical-or operation instead of the logical-and operation.

JavaScript “+” – Semantics

operator +

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the Note below 12.7.5.

NOTE 1 No hint is provided in the calls to ToPrimitive in steps 7 and 9. All standard objects except Date objects handle the absence of a hint as if the hint Number were given; Date objects handle the absence of a hint as if the hint String were given. Exotic objects may handle the absence of a hint in some other manner.

NOTE 2 Step 11 differs from step 5 of the Abstract Relational Comparison algorithm (7.2.11), by using the logical-or operation instead of the logical-and operation.

ToPrimitive

Table 9 — ToPrimitive Conversions

Input Type	Result
Completion Record	If *input* is an abrupt completion, return *input*. Otherwise return ToPrimitive(*input*.[[value]]) also passing the optional hint *PreferredType*.
Undefined	Return *input*.
Null	Return *input*.
Boolean	Return *input*.
Number	Return *input*.
String	Return *input*.
Symbol	Return *input*.
Object	Perform the steps following this table.

When Type(*input*) is Object, the following steps are taken:

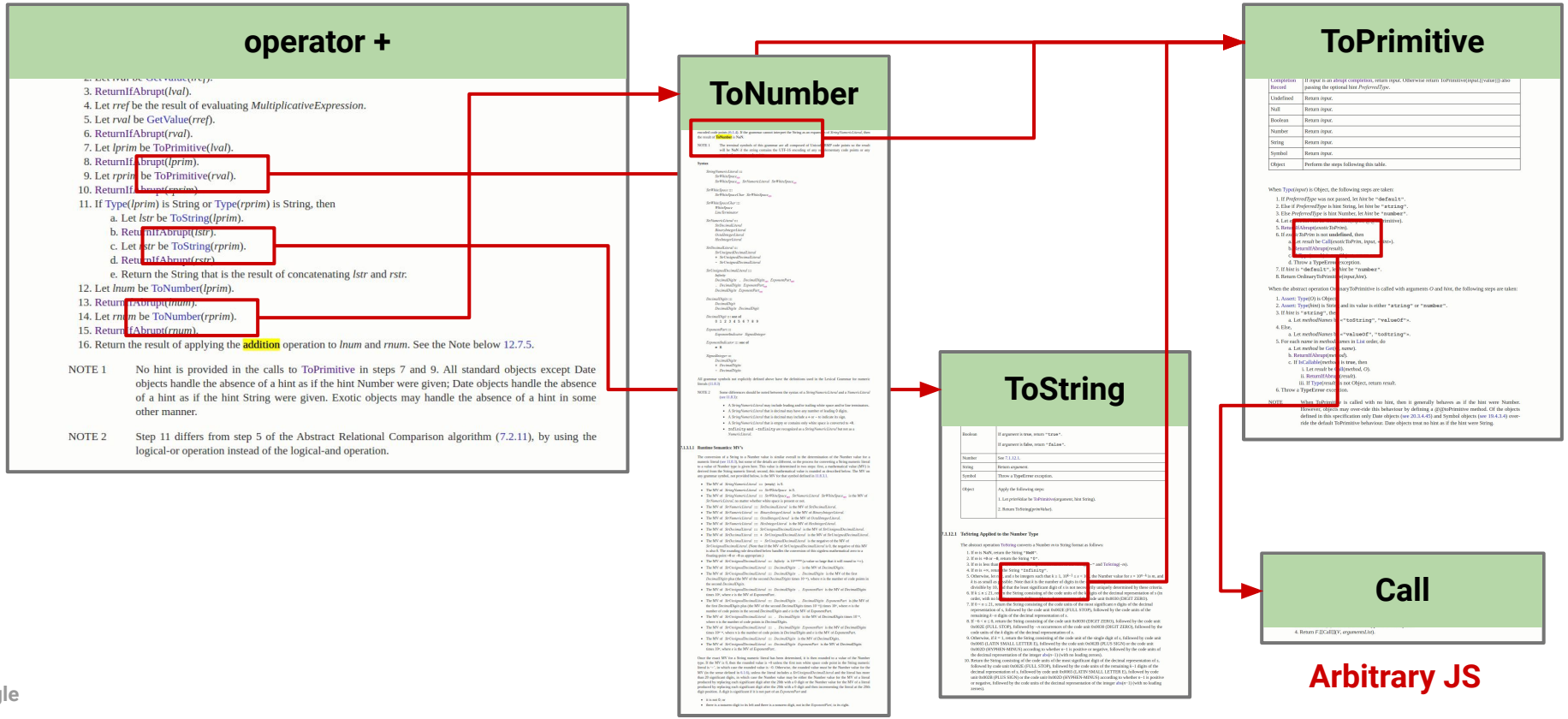
1. If *PreferredType* was not passed, let *hint* be `"default"`.
2. Else if *PreferredType* is hint String, let *hint* be `"string"`.
3. Else *PreferredType* is hint Number, let *hint* be `"number"`.
4. Let *exoticToPrim* be GetMethod(*input*, @@toPrimitive).
5. ReturnIfAbrupt(*exoticToPrim*).
6. If *exoticToPrim* is not **undefined**, then
   a. Let *result* be Call(*exoticToPrim*, *input*, «*hint*»).
   b. ReturnIfAbrupt(*result*).
   c. If Type(*result*) is not Object, return *result*.
   d. Throw a **TypeError** exception.
7. If *hint* is `"default"`, let *hint* be `"number"`.
8. Return OrdinaryToPrimitive(*input*,*hint*).

When the abstract operation OrdinaryToPrimitive is called with arguments *O* and *hint*, the following steps are taken:

1. Assert: Type(*O*) is Object
2. Assert: Type(*hint*) is String and its value is either `"string"` or `"number"`.
3. If *hint* is `"string"`, then
   a. Let *methodNames* be «`"toString"`, `"valueOf"`».
4. Else,
   a. Let *methodNames* be «`"valueOf"`, `"toString"`».
5. For each *name* in *methodNames* in List order, do
   a. Let *method* be Get(*O*, *name*).
   b. ReturnIfAbrupt(*method*).
   c. If IsCallable(*method*) is **true**, then
      i. Let *result* be Call(*method*, *O*).
      ii. ReturnIfAbrupt(*result*).
      iii. If Type(*result*) is not Object, return *result*.
6. Throw a **TypeError** exception.

NOTE When ToPrimitive is called with no hint, then it generally behaves as if the hint were Number. However, objects may over-ride this behaviour by defining a @@toPrimitive method. Of the objects defined in this specification only Date objects (see 20.3.4.45) and Symbol objects (see 19.4.3.4) over-ride the default ToPrimitive behaviour. Date objects treat no hint as if the hint were String.

JavaScript “+” – Semantics

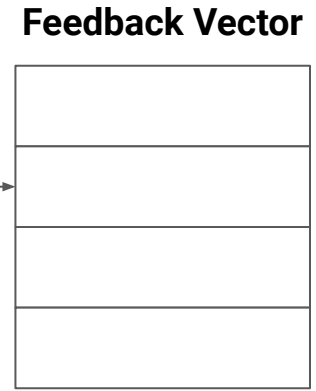


Type feedback: Inline caches

- Placed at various locations
  - E.g., binary operator +, property load
- Record type information
- Specialization based on types

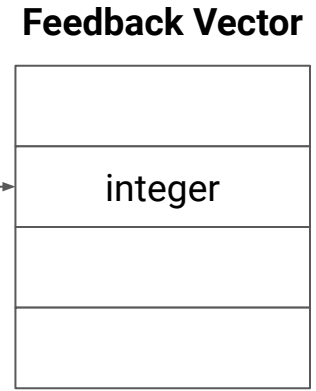
Type feedback: JavaScript “+”

```
function add(a, b) {
  return a + b;
}
```



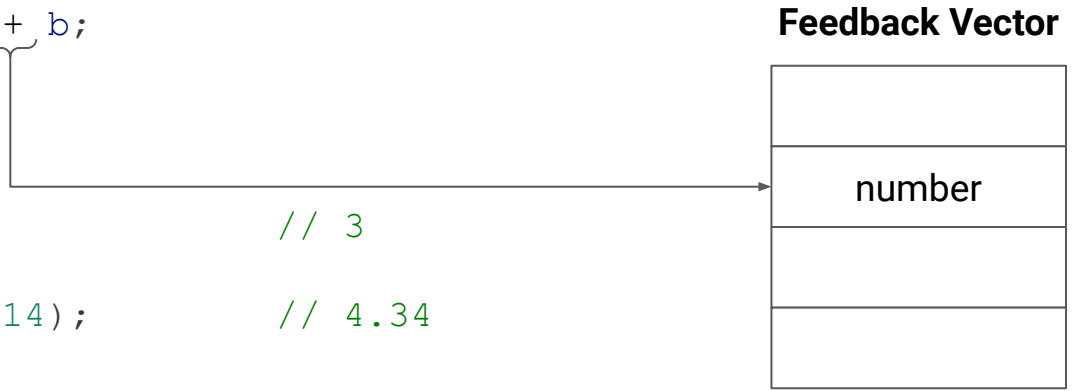
Type feedback: JavaScript “+”

```
function add(a, b) {
  return a + b;
}

add(1, 2);                // 3
```



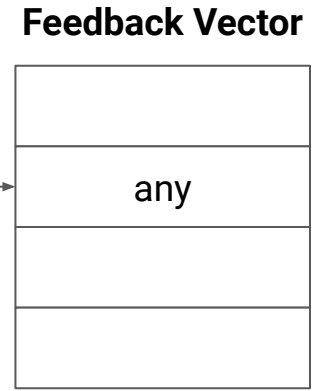
Type feedback: JavaScript “+”

```
function add(a, b) {
  return a + b;
}

add(1, 2);              // 3

add(1.2, 3.14);         // 4.34
```



Type feedback: JavaScript “+”

```
function add(a, b) {
  return a + b;
}

add(1, 2);                // 3

add(1.2, 3.14);           // 4.34

add(“hello”, “world”); // “helloworld”
```



Type feedback: Object loads

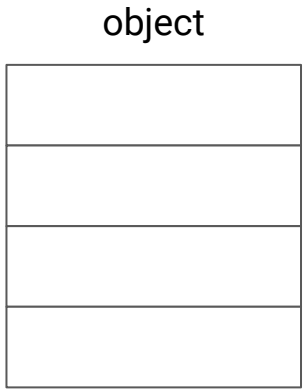
```
function loadX(point) {
  return p.x;
}
```

Object shapes in V8

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```

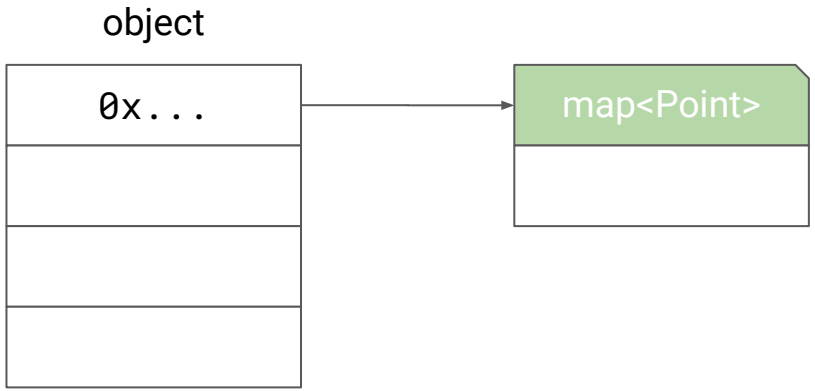
Object shapes in V8

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```



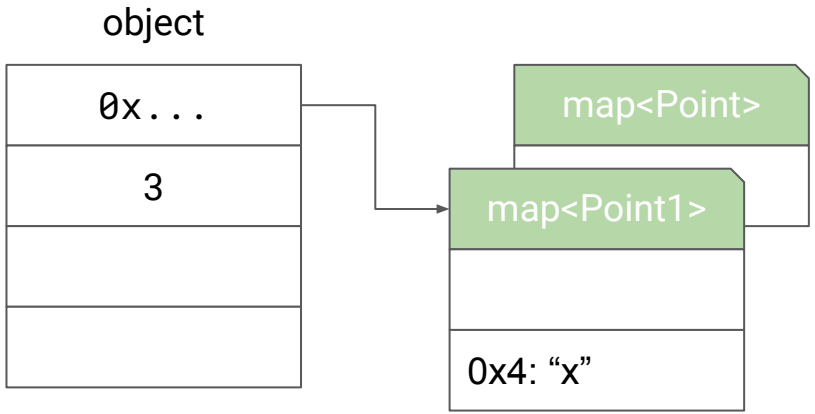
Object shapes in V8

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```



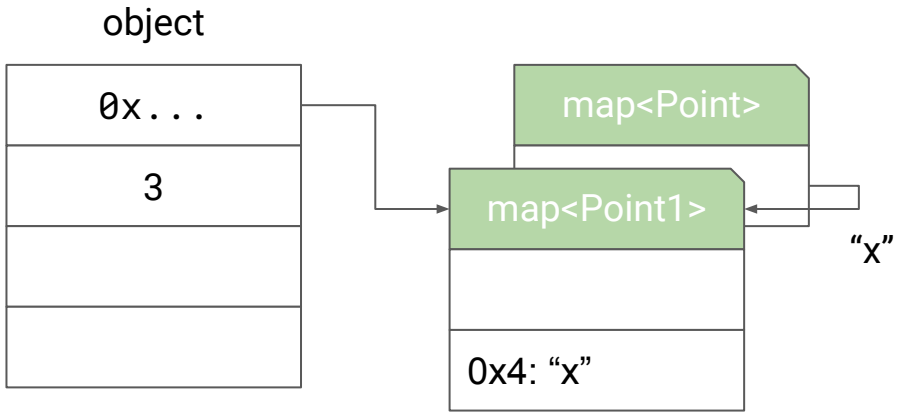
Object shapes in V8

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```



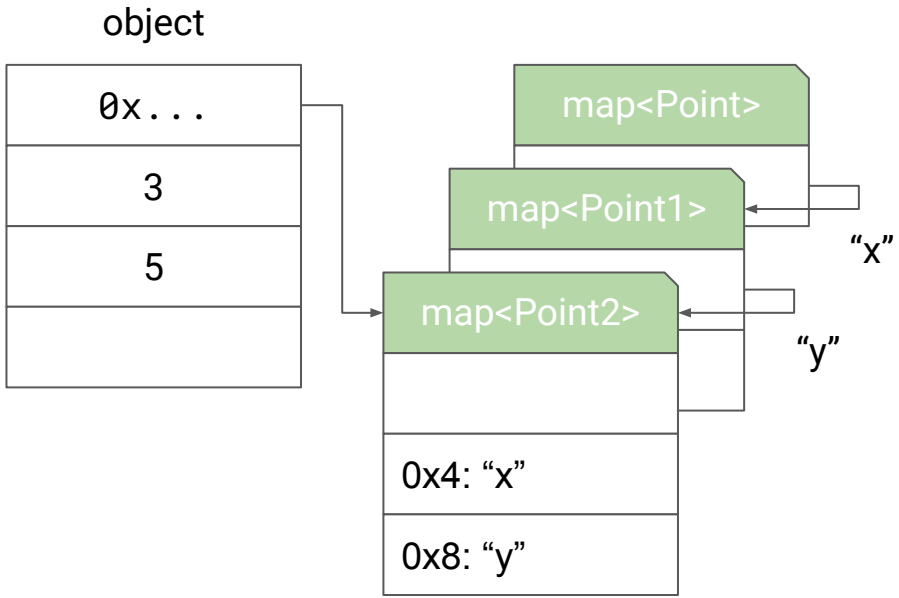
Object shapes in V8

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```



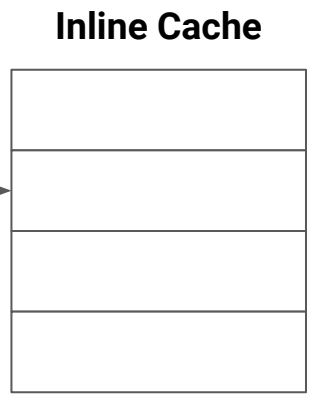
Object shapes in V8

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

var point = new Point(3, 5);
```



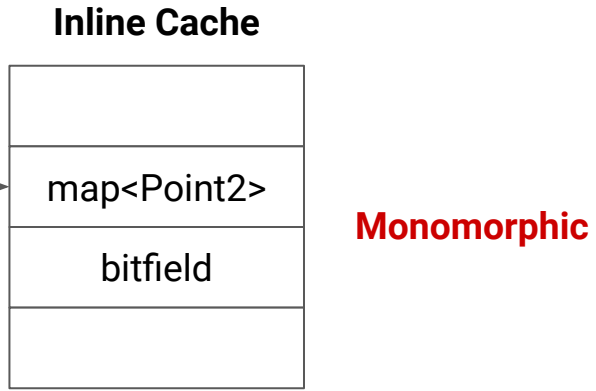
Type feedback: Object loads

```
function loadX(point) {
  return p.x;
}
```



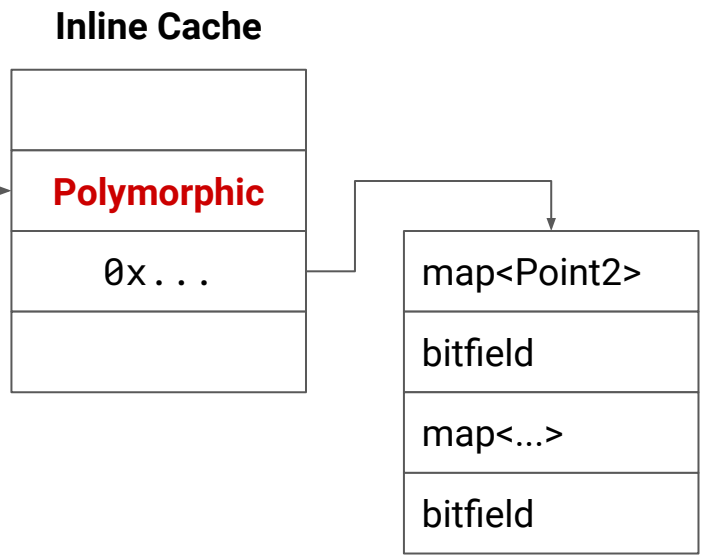
Type feedback: Object loads

```
function loadX(point) {
  return p.x;
}
loadX(new Point(1, 2));
```



Type feedback: Object loads

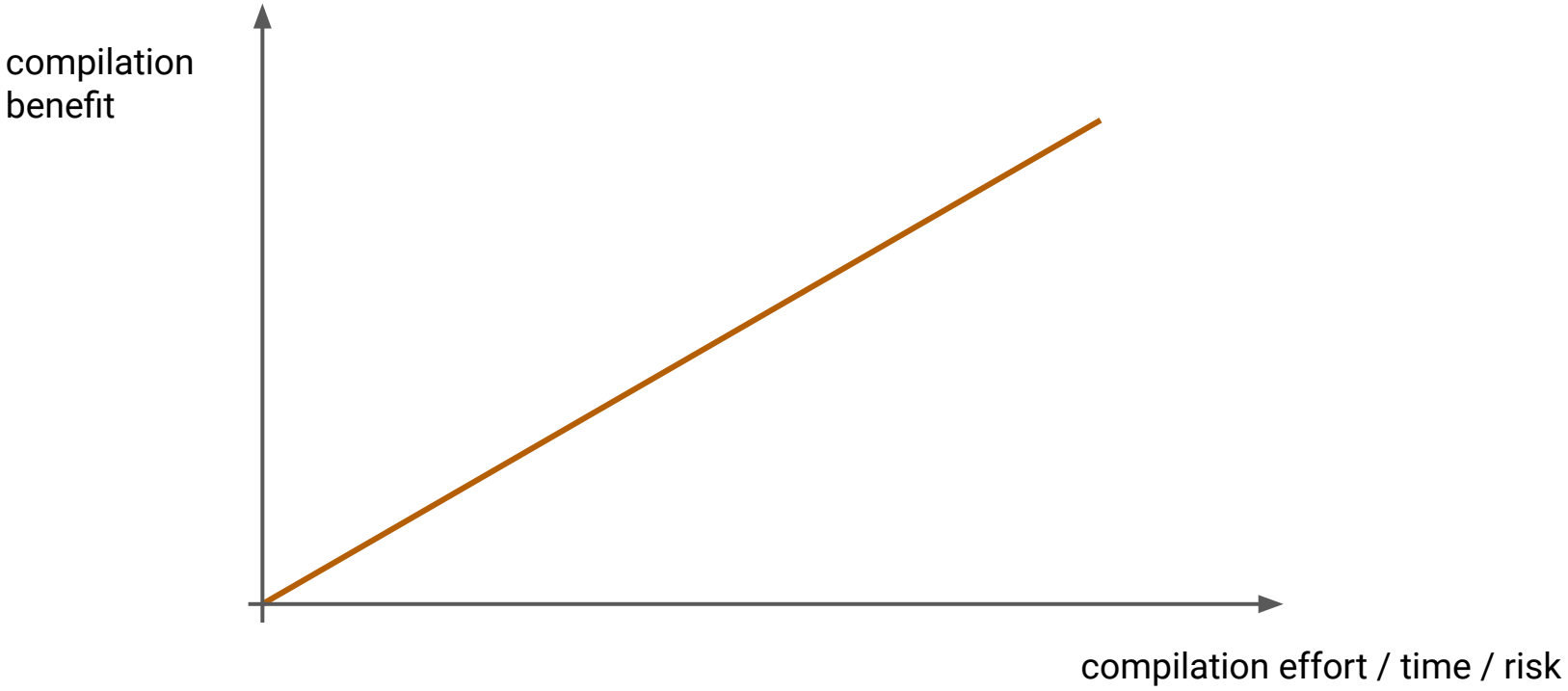
```
function loadX(point) {
  return p.x;
}
loadX(new Point(1, 2));
loadX({x: 1, bar: 2});
```



Just-In-Time (JIT) compilation

- “Just in time”
  - Few ms for the web
  - Often minutes for other applications (e.g. server)
- On the web compilation decisions have huge impact
  - Window of opportunity for the broad web is often only a few ms
    - Exception: Larger applications
  - Not compiling means being slow
  - Compiling too early with wrong assumptions results in deoptimization and missed opportunities
- Way out: **Compilers need to pay for themselves in their use cases**

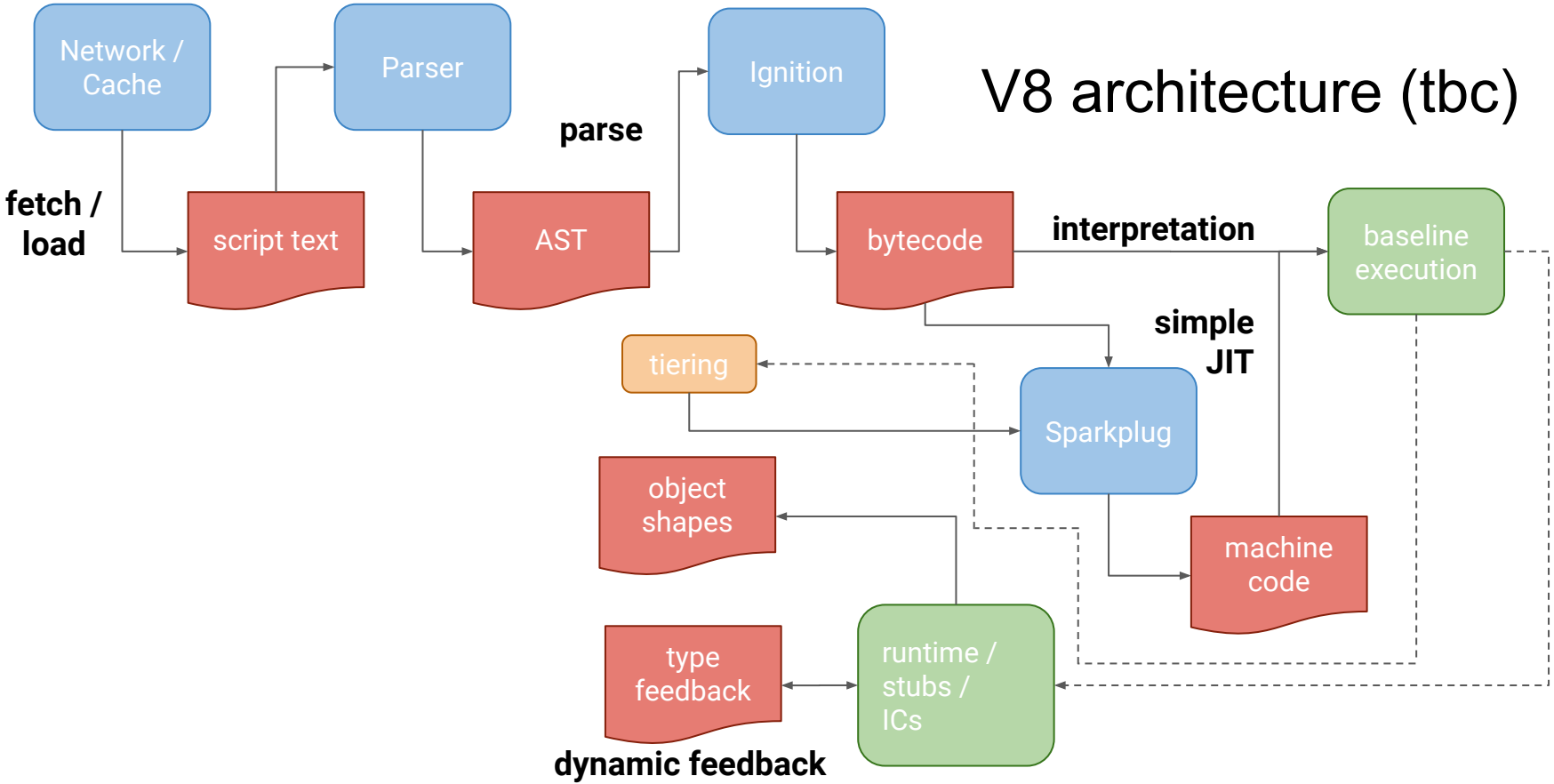
Compiler tiers



Compiler tiers

Compiler tiers

V8 architecture (tbc)



Sparkplug: A non-optimizing JIT compiler

- Motivation: A fast compiler that can pay for itself

```
// The Sparkplug compiler (abridged).
for (; !iterator.done(); iterator.Advance()) {
  VisitSingleBytecode();
}
```

- Input: Bytecode
- Output: Machine code

https://v8.dev/blog/sparkplug

Sparkplug: A non-optimizing JIT compiler

- Compiles individual bytecodes
- Two passes
  - Finding back edges
  - Code generation
- Relies on built-in functions for the actual operation
  - E.g., “+” or loading a property
  - In practice, this always means calling out to built-in functions
- In essence: Serialization of interpreter execution in native code

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

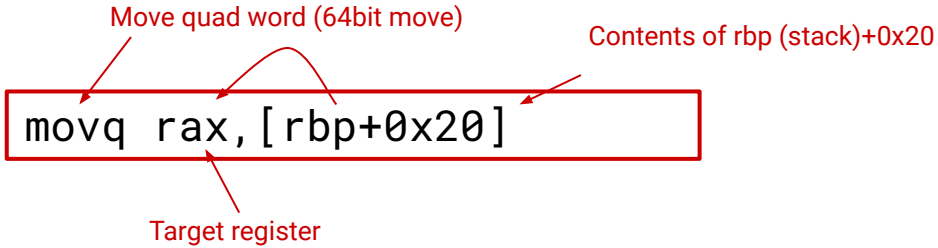
Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

- We will just accept V8's stack layout without going into details
- `rbx`: Register where builtin functions expect their feedback (mostly, not always)

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```



Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
cmp rax,0xdf9
jnz 0x55c5e00046c4
jmp 0x55c5e00046e7
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
cmp rax,0xdf9
jnz 0x55c5e00046c4
jmp 0x55c5e00046e7
movq rax,[rbp+0x20]
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
cmp rax,0xdf9
jnz 0x55c5e00046c4
jmp 0x55c5e00046e7
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
movl rbx,0x2
call 0x55c5be6d0bc0  (Subtract_Baseline)
```

Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
cmp rax,0xdf9
jnz 0x55c5e00046c4
jmp 0x55c5e00046e7
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
movl rbx,0x2
call 0x55c5be6d0bc0  (Subtract_Baseline)
movl rbx,0x3
movq rcx,0xfffffee
jmp 0x55c5be471e00  (BaselineLeaveFrame)
```

Eventually returning from the function. Conveniently (by design) rax (return register) holds the value of the accumulator which is what we want to return.

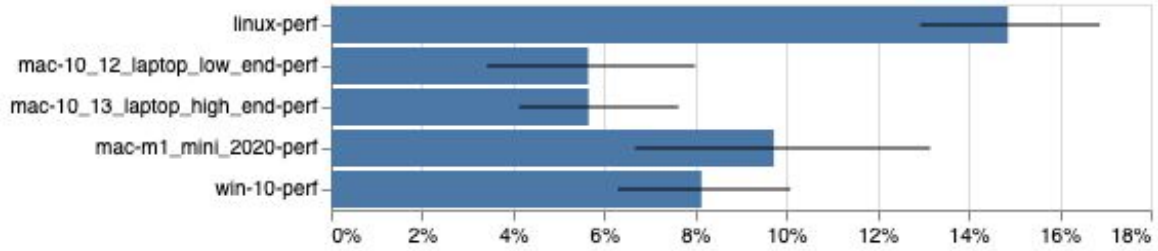
Sparkplug: Compilation

```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
xorl rbx,rbx
call 0x55c5be6d0bc0  (Subtract_Baseline)
movq [rbp-0x30],rax
xorl rax,rax
movq rdx,[rbp-0x30]
movl rbx,0x1
call 0x55c5be722a80  (GreaterThan_Baseline)
cmp rax,0xdf9
jnz 0x55c5e00046c4
jmp 0x55c5e00046e7
movq rax,[rbp+0x20]
movq rdx,[rbp+0x18]
movl rbx,0x2
call 0x55c5be6d0bc0  (Subtract_Baseline)
movl rbx,0x3
movq rcx,0xfffffee
jmp 0x55c5be471e00  (BaselineLeaveFrame)
```

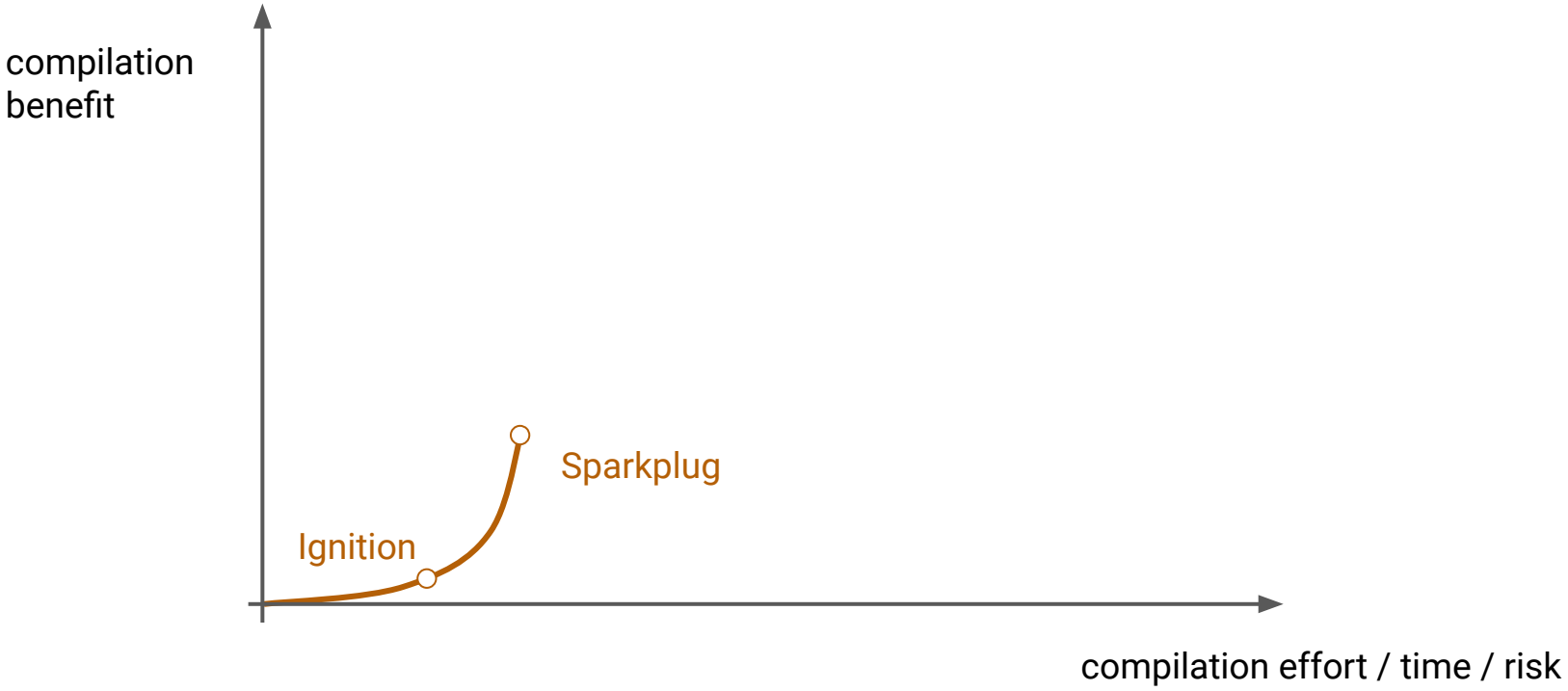
Sparkplug was implemented in 4 days

Results running with Sparkplug on Speedometer2

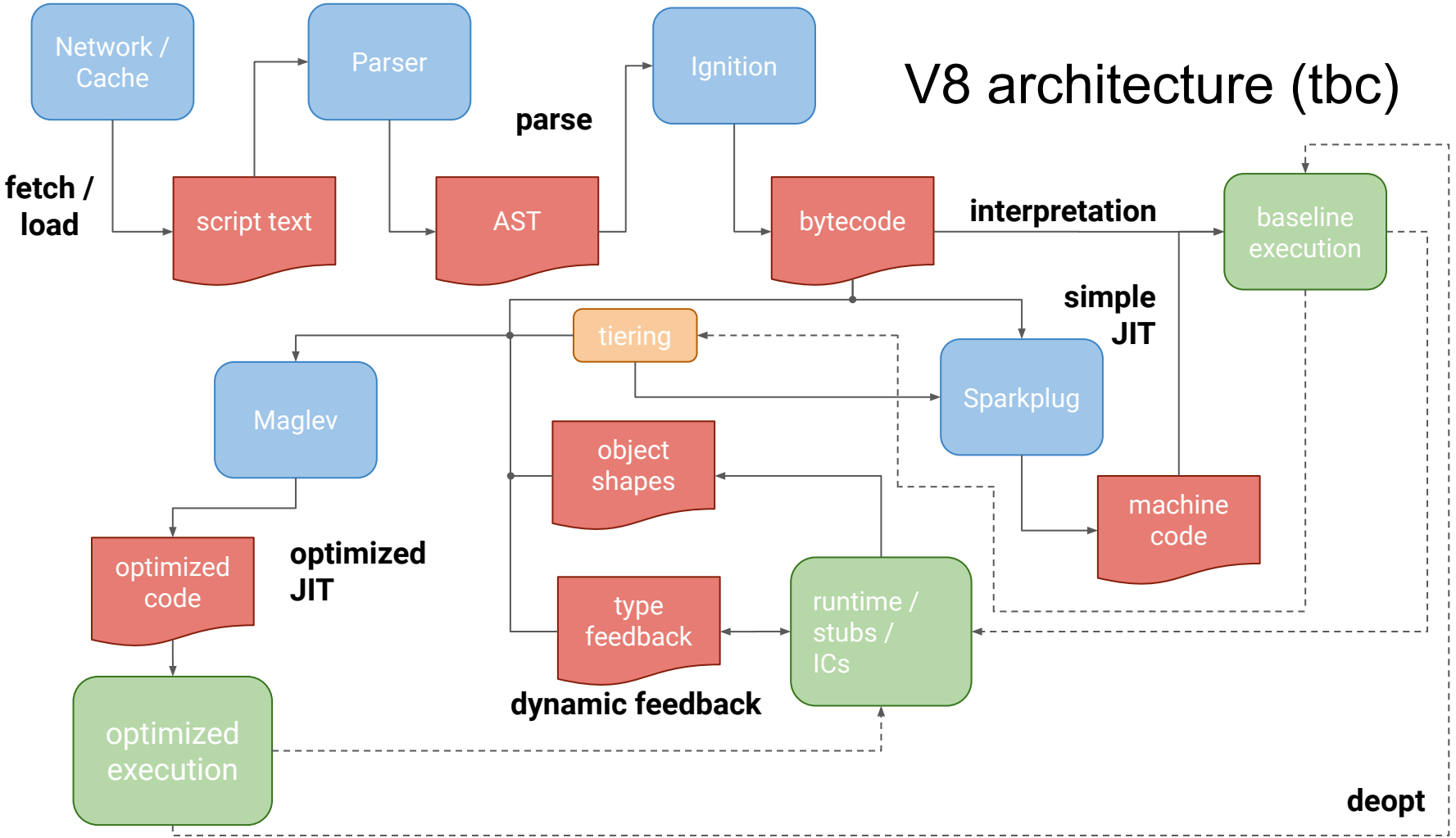


Compared against baseline which back then ran with Ignition and TurboFan.

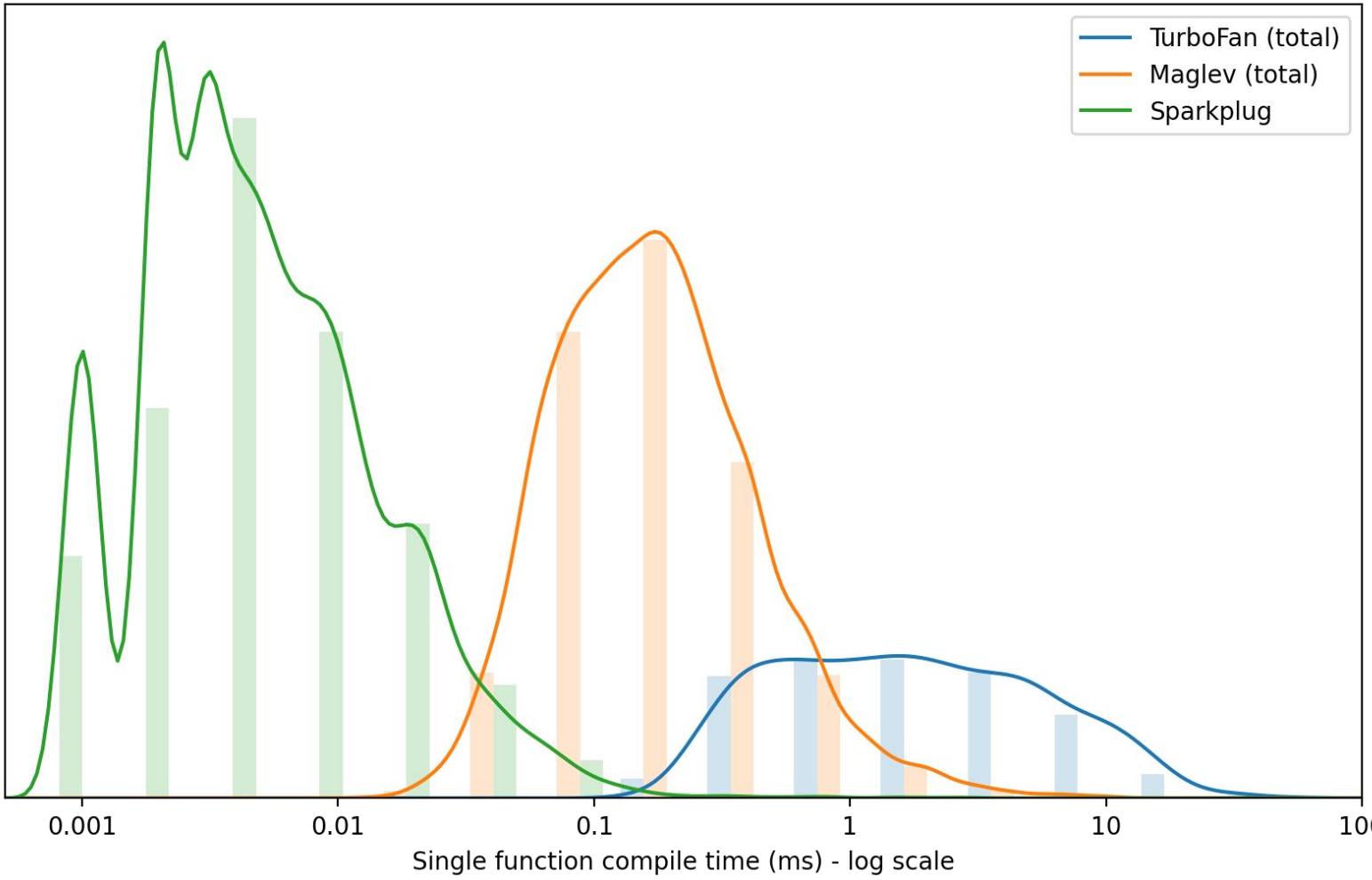
Compiler tiers



V8 architecture (tbc)



Maglev: The fastest optimizing compiler in V8


TurboFan (total)
Maglev (total)
Sparkplug
0.001
0.01
0.1
1
10
100
Single function compile time (ms) - log scale

Maglev prioritizes “fast over perfect”

- Have optimized code quicker
  … for smoother loading
- Compile more
  … with the same resources
- Compile earlier
  … since it’s not as expensive to be wrong

Design goal: Do (almost) everything in one linear pass.

Maglev graph building

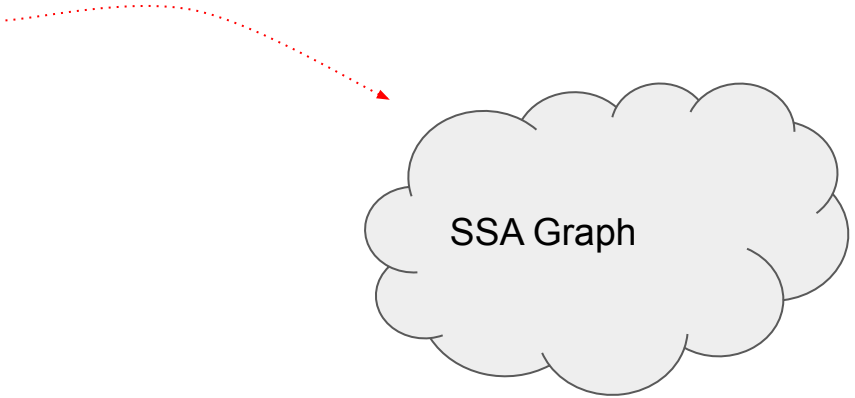
```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Maglev graph building

Interpreter Frame State
pc =
r0 =
acc =

Known Node Aspects

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```



Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 0
r0 =
acc = 2

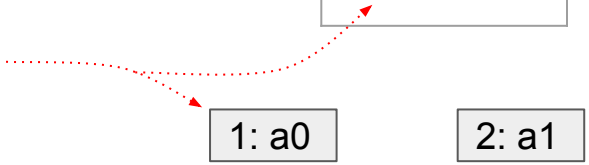
Known Node Aspects

1: a0

2: a1

Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```



Known Node Aspects

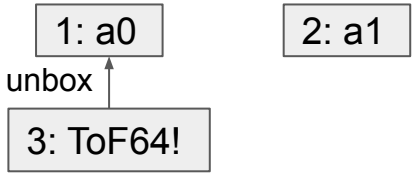
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 1
r0 =
acc = 2

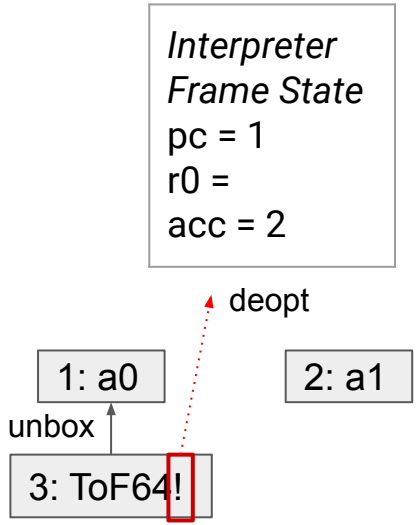
Known Node Aspects

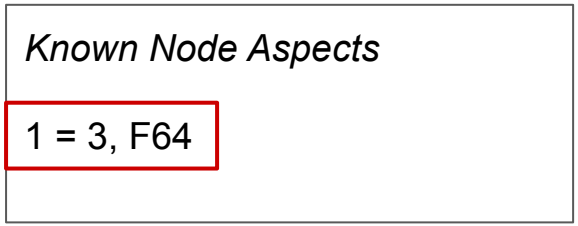
1 = 3



Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```





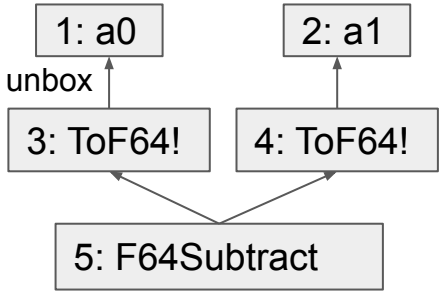
Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```

Interpreter Frame State
pc = 1
r0 =
acc = 5

Known Node Aspects

1 = 3, F64
2 = 4, F64



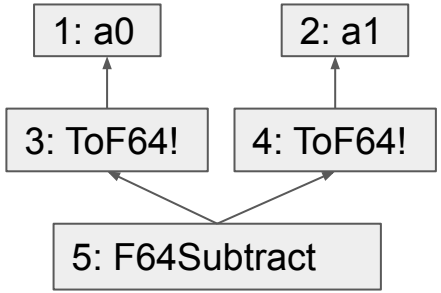
Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```

Interpreter Frame State
pc = 2
r0 = 5
acc = 5

Known Node Aspects

1 = 3, F64
2 = 4, F64



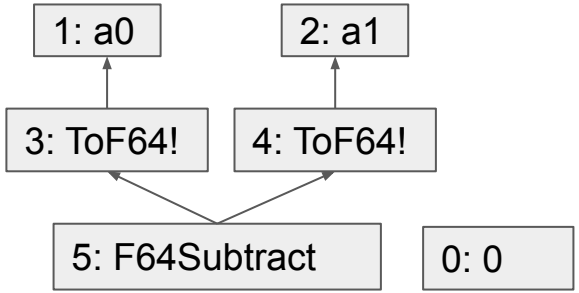
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 3
r0 = 5
acc = 0

Known Node Aspects

1 = 3, F64
2 = 4, F64



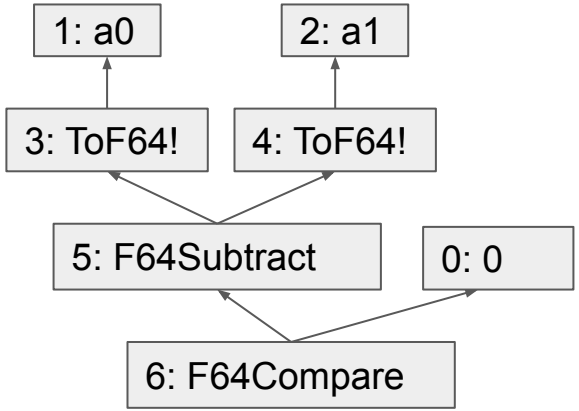
Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```

Interpreter Frame State
pc = 4
r0 = 5
acc = 6

Known Node Aspects

1 = 3, F64
2 = 4, F64



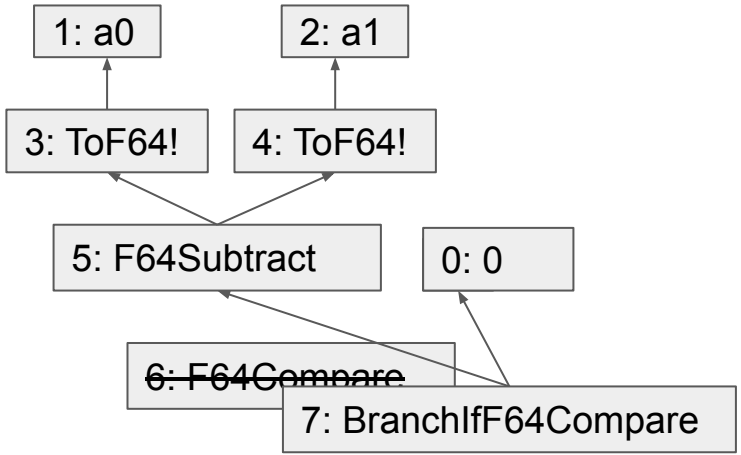
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 5
r0 = 5
acc = 6

Known Node Aspects

1 = 3, F64
2 = 4, F64



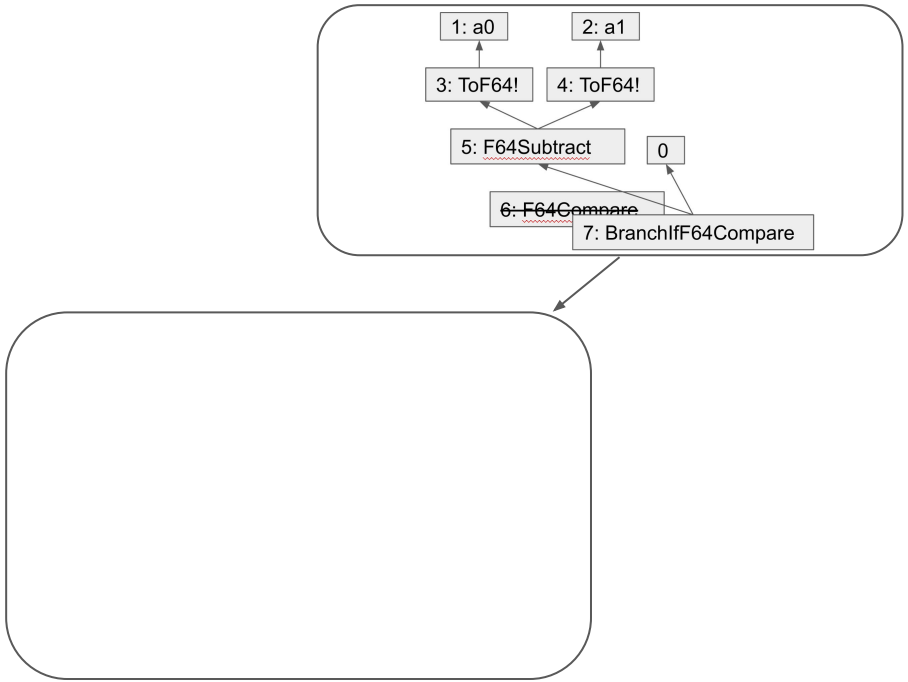
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 5
r0 = 5
acc = 6

Known Node Aspects

1 = 3, F64
2 = 4, F64



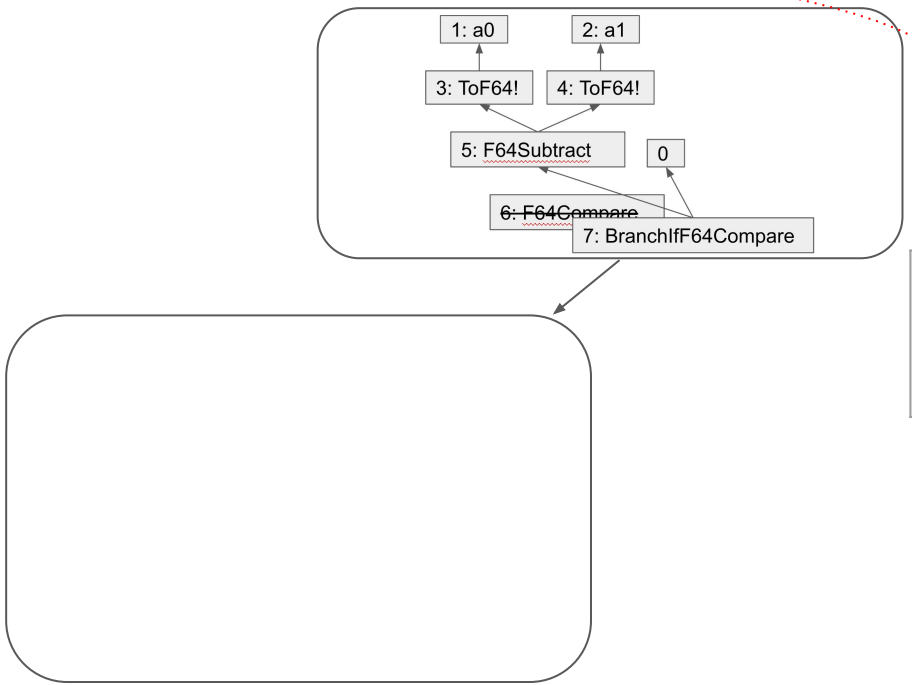
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 5
r0 = 5
acc = 6

Known Node Aspects

1 = 3, F64
2 = 4, F64



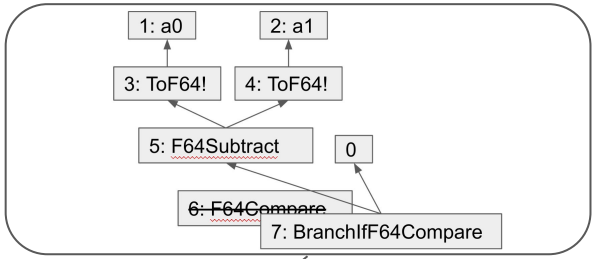
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 6
r0 = 5
acc = 2

Known Node Aspects

1 = 3, F64
2 = 4, F64



Interpreter Frame State
pc = 5
r0 = 5
acc = 6

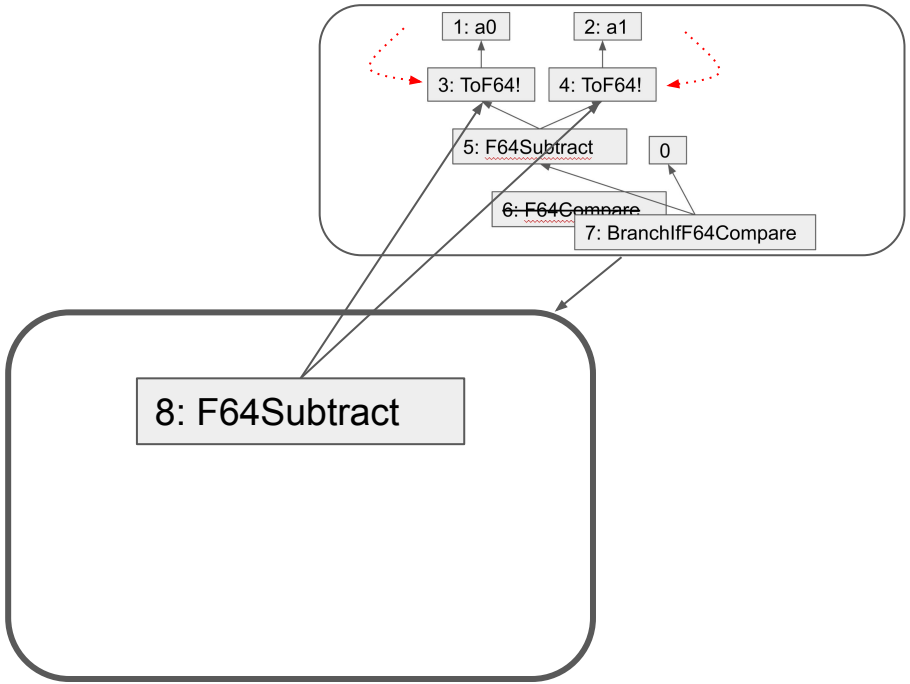
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 7
r0 = 5
acc = 8

Known Node Aspects

1 = 3, F64
2 = 4, F64



Interpreter Frame State
pc = 5
r0 = 5
acc = 6

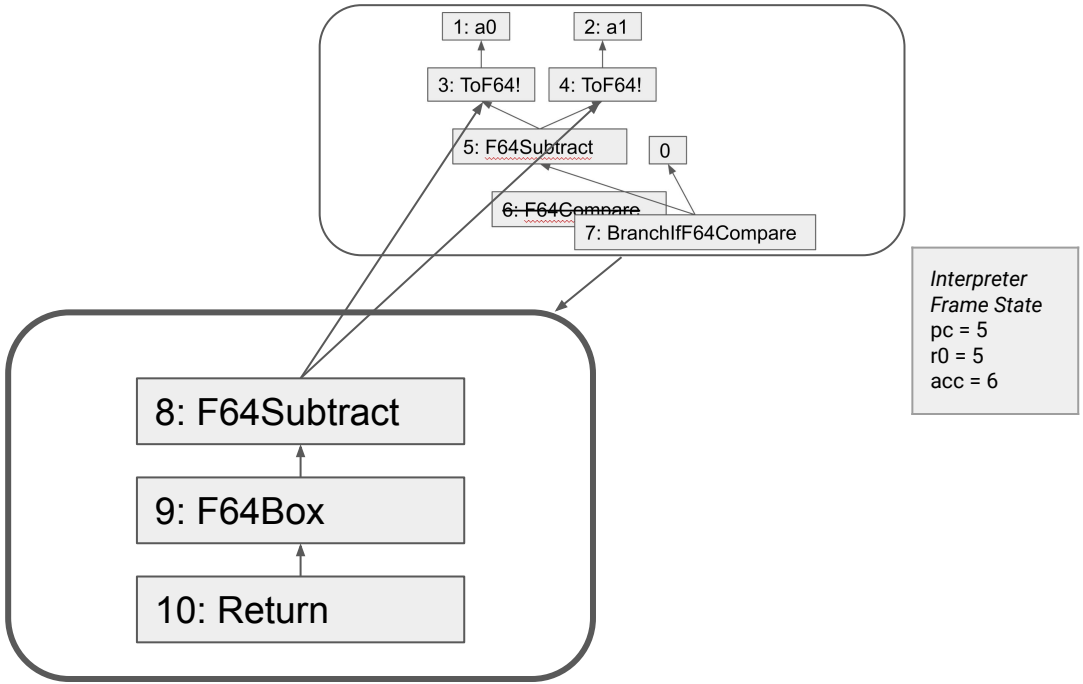
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 8
r0 = 5
acc = 10

Known Node Aspects

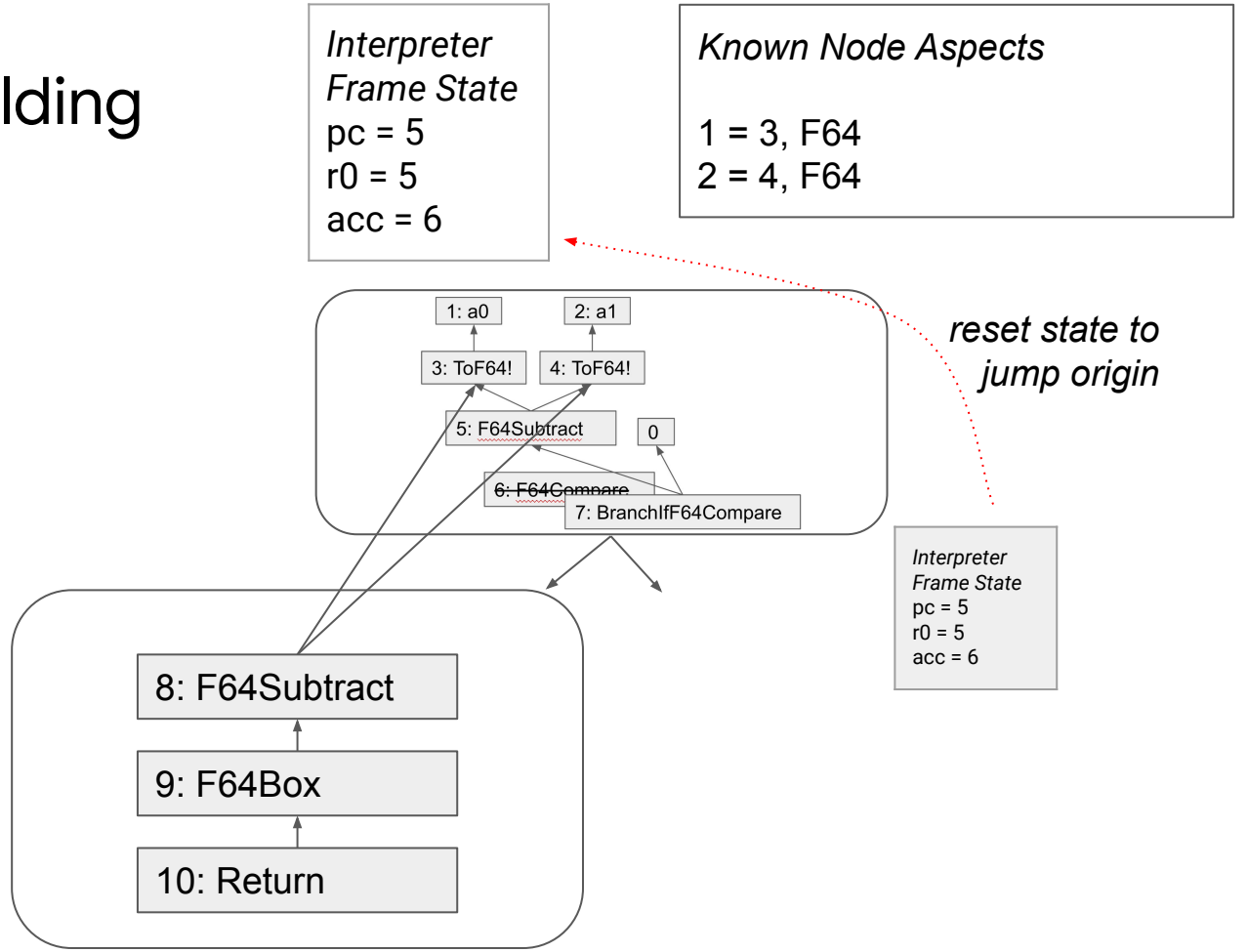
1 = 3, F64
2 = 4, F64



Interpreter Frame State
pc = 5
r0 = 5
acc = 6

Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```



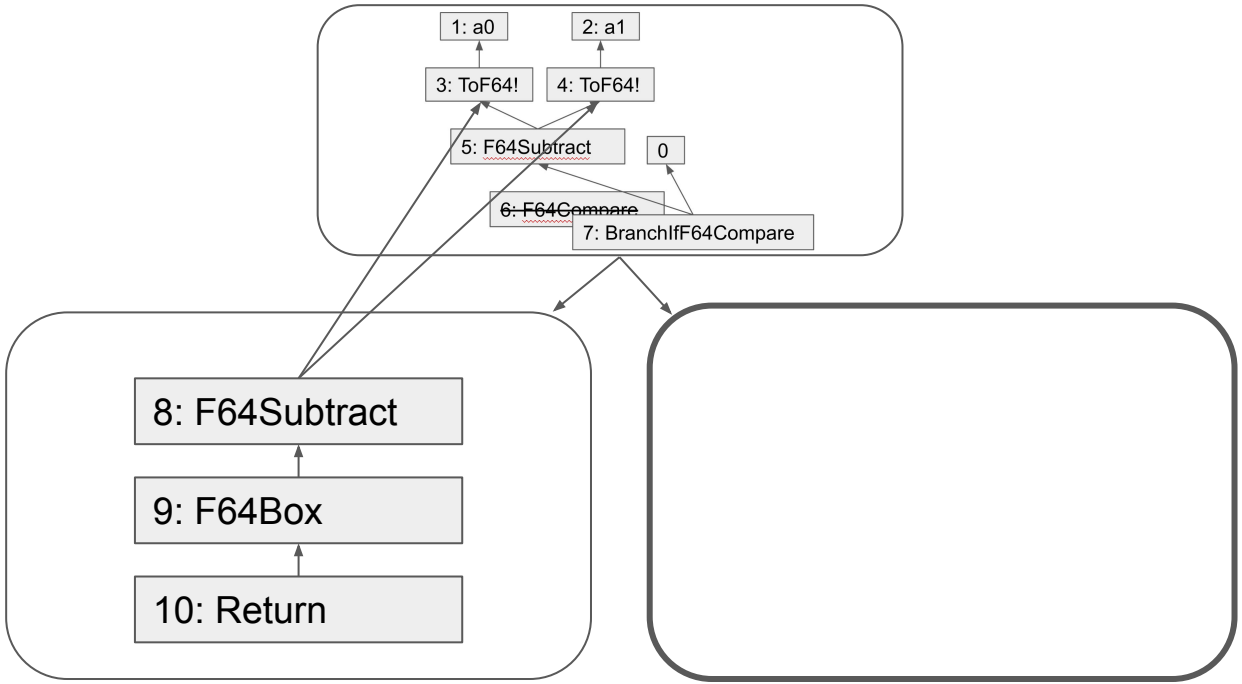
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 5
r0 = 5
acc = 6

Known Node Aspects

1 = 3, F64
2 = 4, F64



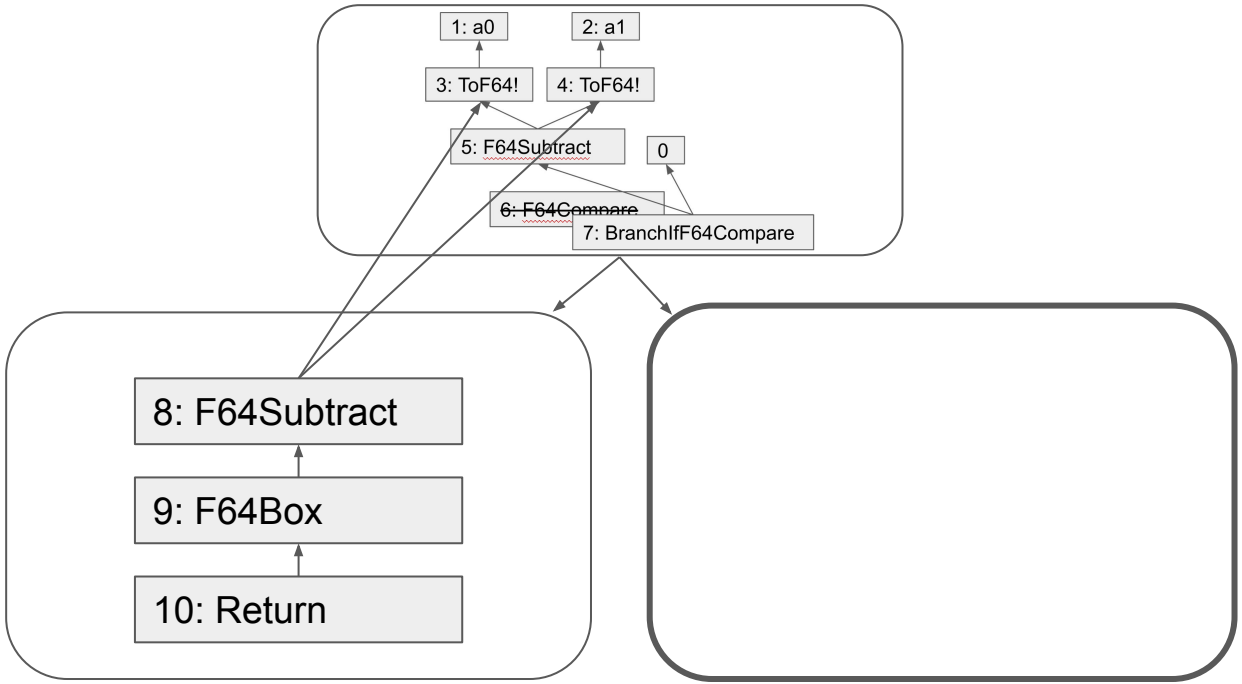
Maglev graph building

```
 0 Ldar a1
 1 Sub a0
 2 Star0
 3 LdaZero
 4 TestGreaterThan r0
 5 JumpIfFalse
 6 Ldar a1
 7 Sub a0
 8 Return
 9 LdaZero
10 Return
```

Interpreter Frame State
pc = 9
r0 = 5
acc = 0

Known Node Aspects

1 = 3, F64
2 = 4, F64



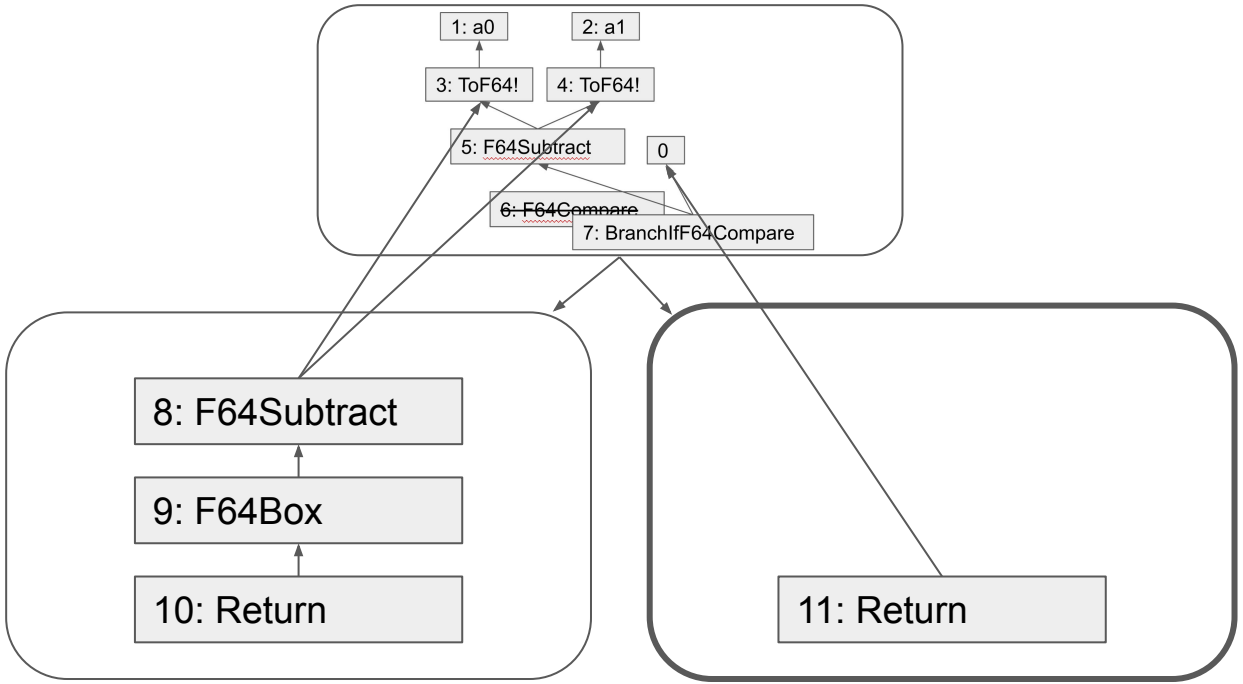
Maglev graph building

```
0  Ldar a1
1  Sub a0
2  Star0
3  LdaZero
4  TestGreaterThan r0
5  JumpIfFalse
6  Ldar a1
7  Sub a0
8  Return
9  LdaZero
10 Return
```

Interpreter Frame State
pc = 10
r0 = 5
acc = 11

Known Node Aspects

1 = 3, F64
2 = 4, F64

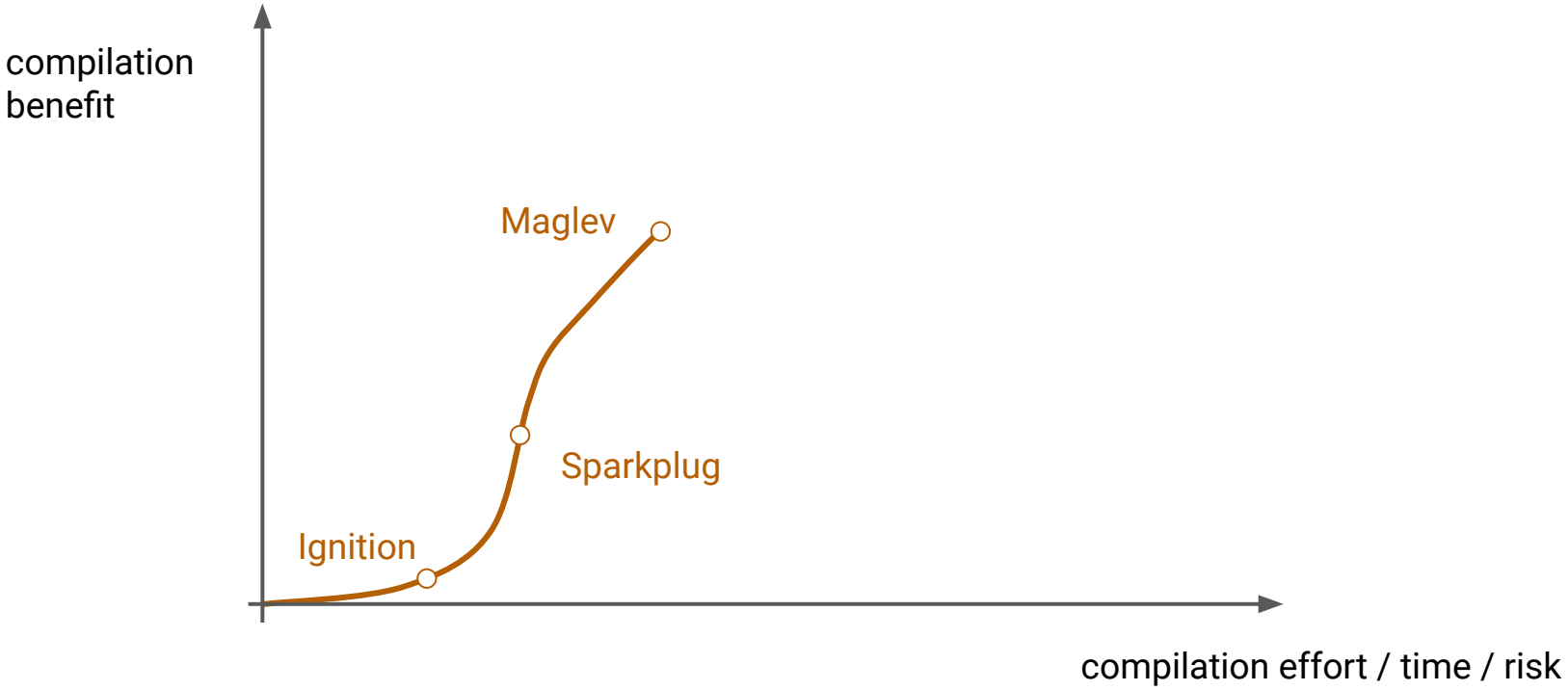


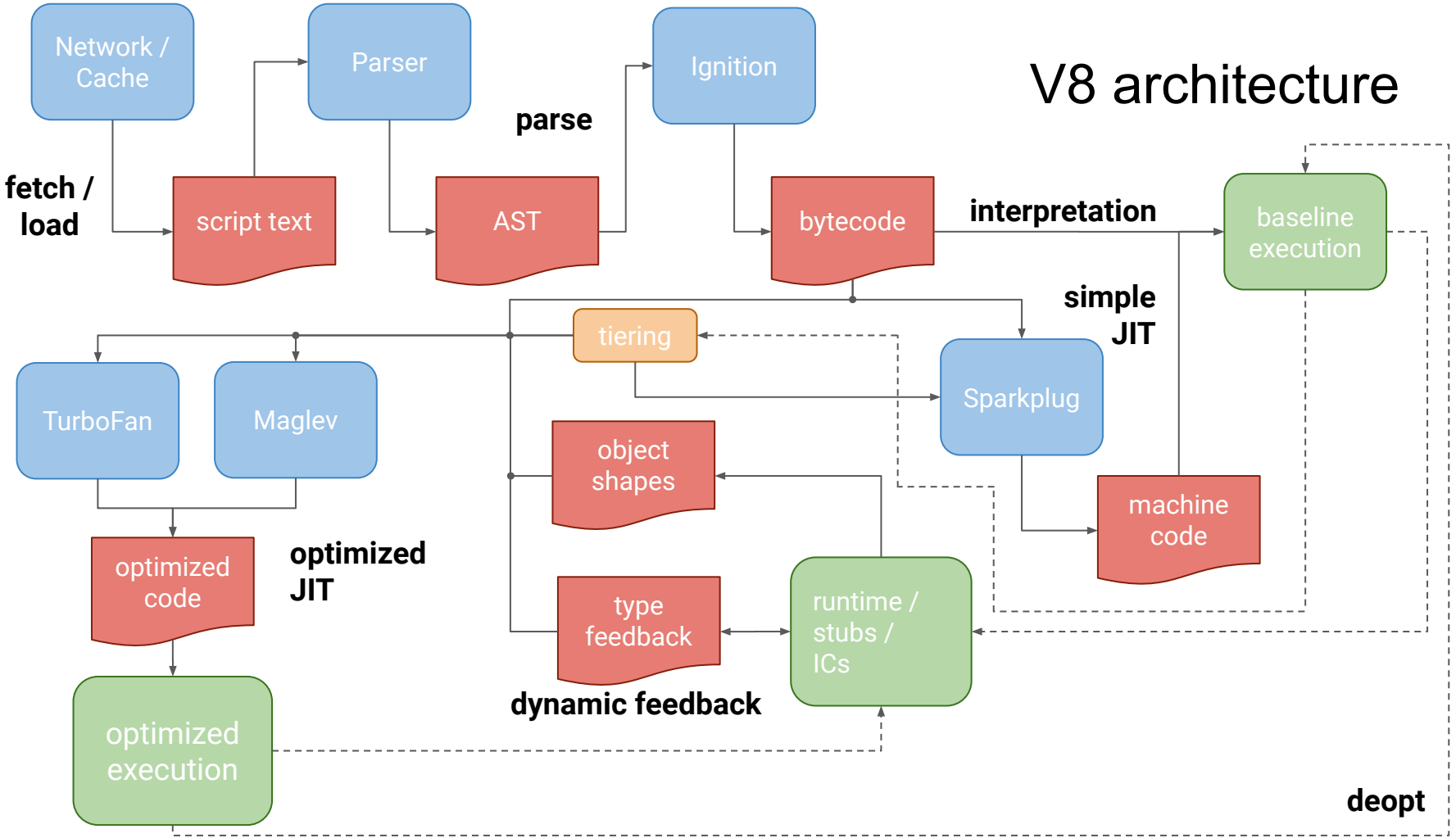
Maglev passes

1. **Graph building, inlining, representation selection, check elimination, branch elimination, load elimination, context specialization, speculative lowering, …**
2. Representation analysis (tagged/float64/int32) for phis
3. Live range analysis, decompression elimination, other preparation for regalloc & codegen
4. Register allocation, codegen

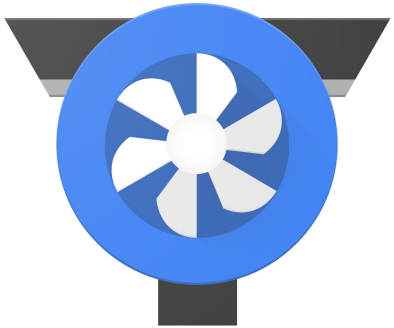
All passes use one and the same IR!

Compiler tiers




V8 architecture
Network / Cache
Parser
Ignition
parse
fetch / load
script text
AST
bytecode
interpretation
baseline execution
simple JIT
tiering
Sparkplug
TurboFan
Maglev
object shapes
machine code
optimized code
optimized JIT
type feedback
runtime / stubs / ICs
dynamic feedback
optimized execution
deopt

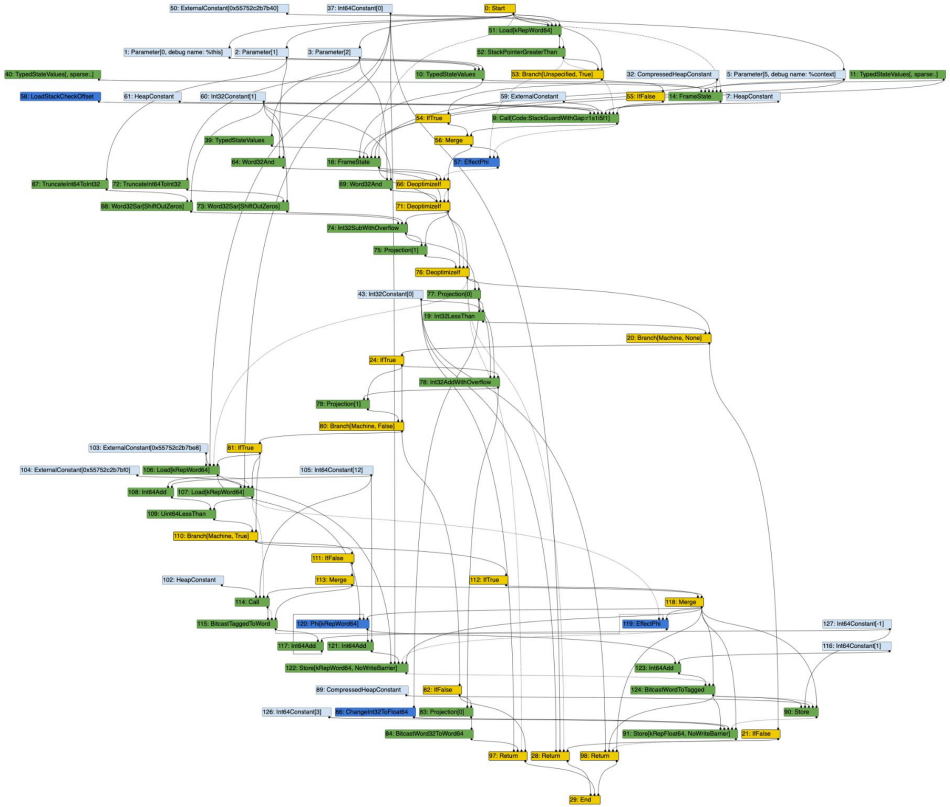
TurboFan: V8's last tier compiler



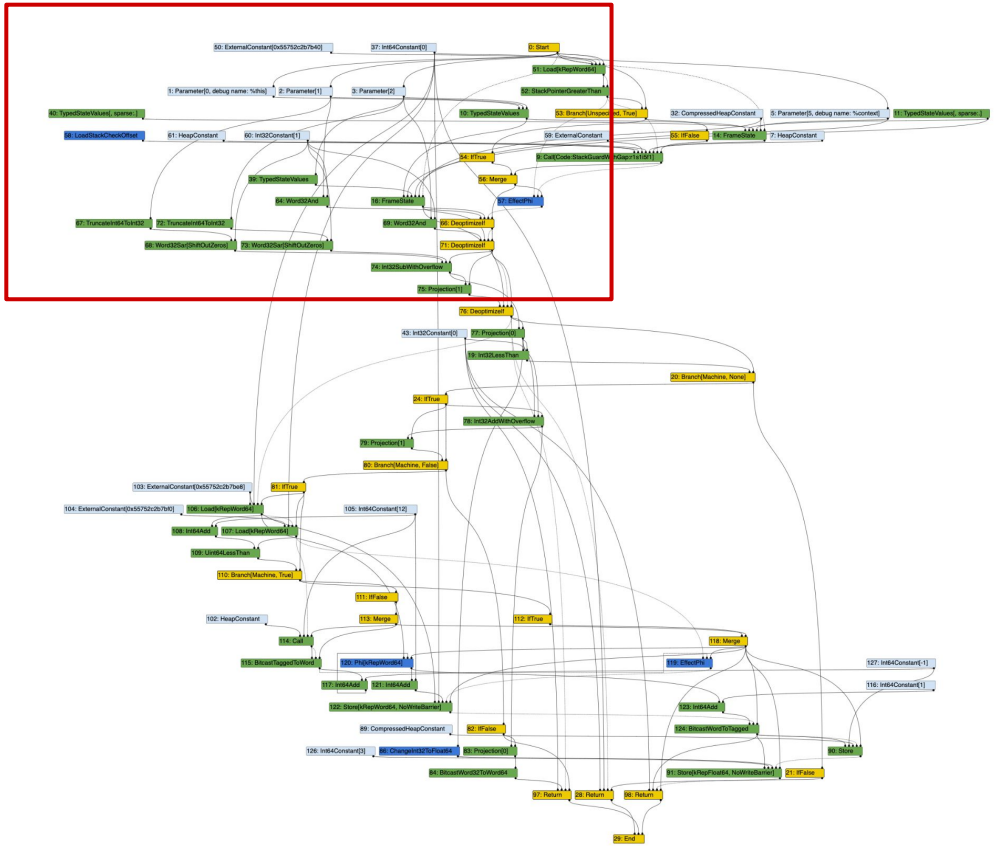
- Sea of nodes based compiler
  - Multiple different graphs that are overlaid
- Many optimization passes
  - Load/store elimination
  - Range checks
  - CSE
  - Dead code elimination
  - Constant propagation
  - Loop peeling
  - …

- Input: Bytecode
- Output: Machine code

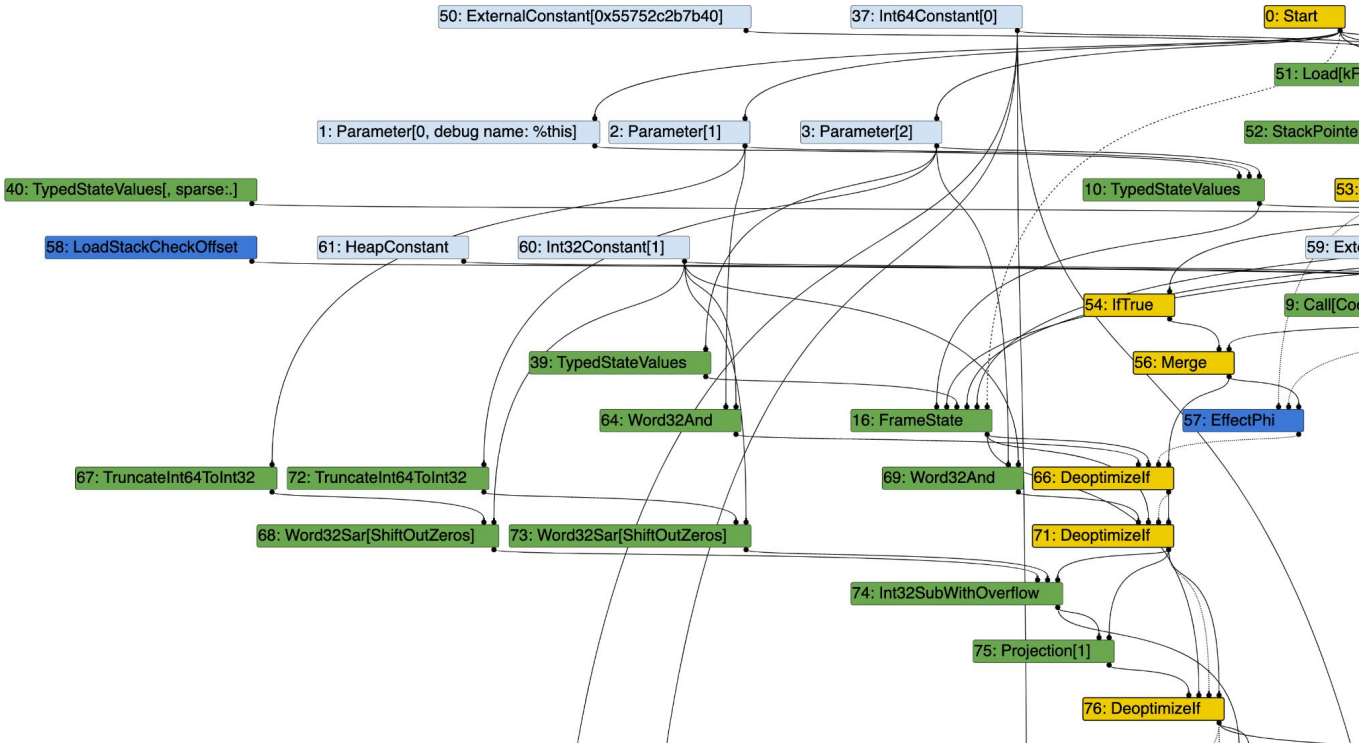
TurboFan: Sea of nodes



TurboFan: Sea of nodes



TurboFan: Sea of nodes



```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

```
    42  movq rcx,[rbp+0x18]
    46  testb rcx,0x1
    49  jnz 0x55758a1c415f  <+0x11f>
    4f  movq rdi,[rbp+0x20]
    53  testb rdi,0x1
    57  jnz 0x55758a1c4163  <+0x123>
    5d  movq r8,rdi
    60  sarl r8, 1
    63  movq r9,rcx
    66  sarl r9, 1
    69  subl r9,r8
    6c  jo 0x55758a1c4167  <+0x127>
    72  testl r9,r9
    75  jg 0x55758a1c40d7  B5 <+0x97>
B4:
    7b  xorl rax,rax
    7d  movq rcx,[rbp-0x18]
    81  movq rsp,rbp
    84  pop rbp
    85  cmpq rcx,0x3
    89  jg 0x55758a1c40ce  <+0x8e>
    8b  ret 0x18
    8e  pop r10
    90  leaq rsp,[rsp+rcx*8]
    94  push r10
    96  retl
```

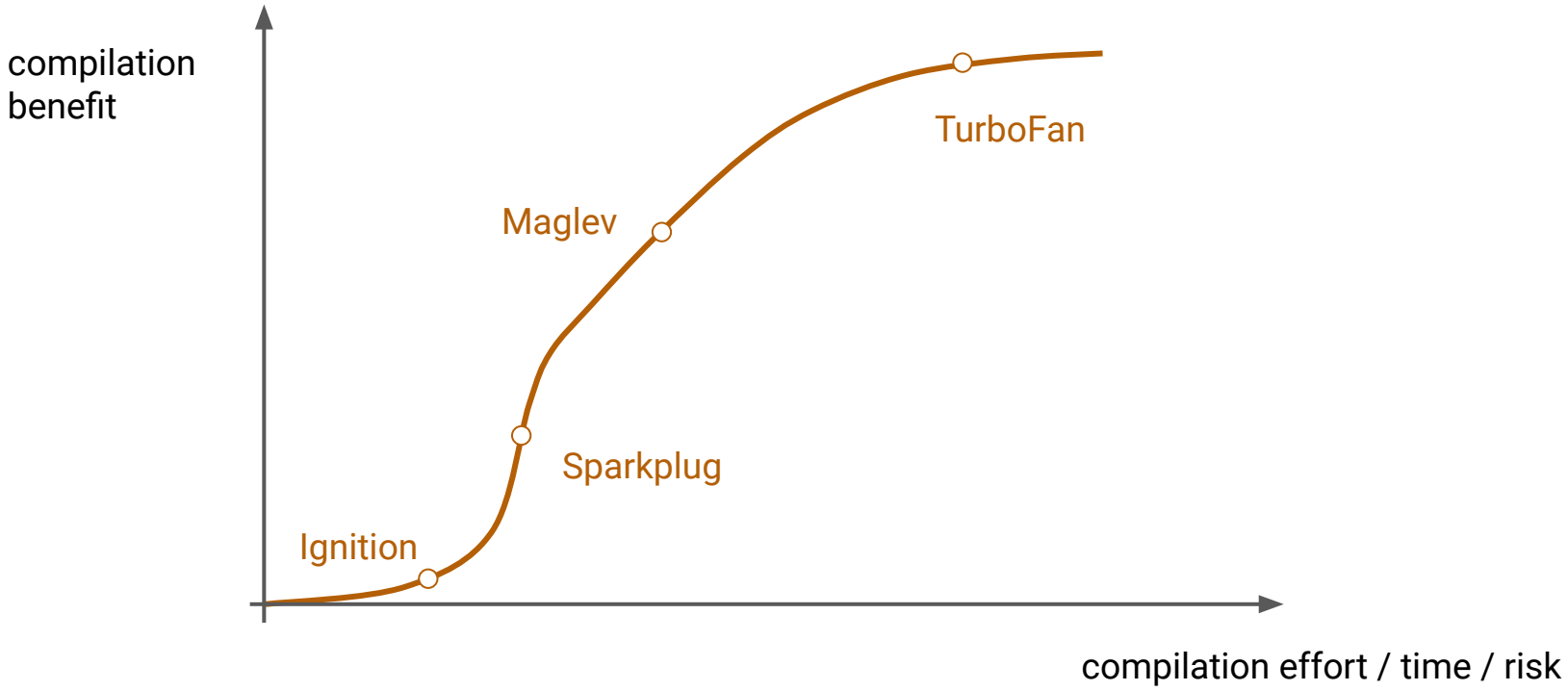
```
 0 Ldar a1
 2 Sub a0
 5 Star0
 6 LdaZero
 7 TestGreaterThan r0
10 JumpIfFalse [8] (18)
12 Ldar a1
14 Sub a0
17 Return
18 LdaZero
19 Return
```

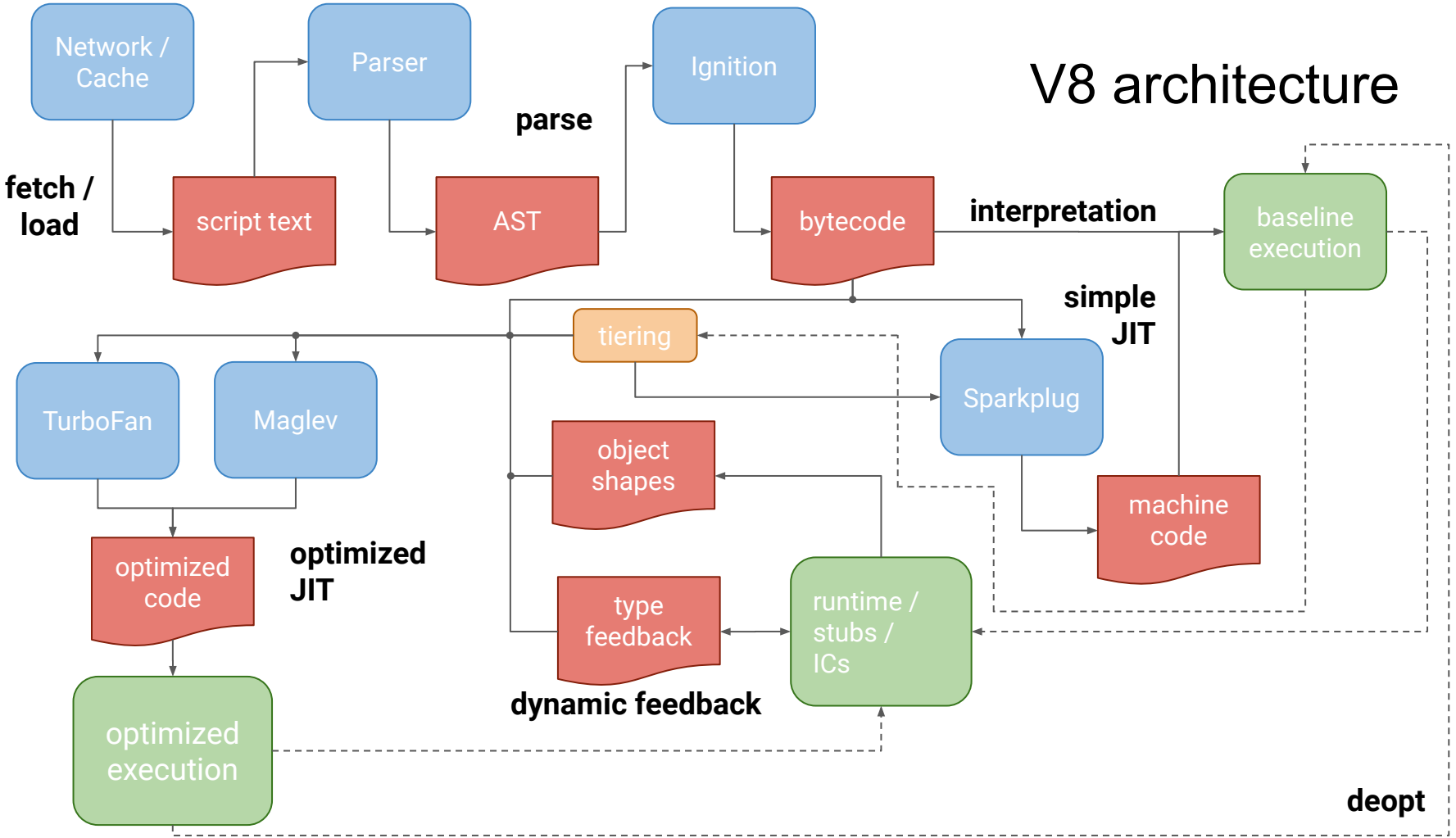
```
    42  movq rcx,[rbp+0x18]
    46  testb rcx,0x1
    49  jnz 0x55758a1c415f  <+0x11f>
    4f  movq rdi,[rbp+0x20]
    53  testb rdi,0x1
    57  jnz 0x55758a1c4163  <+0x123>
    5d  movq r8,rdi
    60  sarl r8, 1
    63  movq r9,rcx
    66  sarl r9, 1
    69  subl r9,r8
    6c  jo 0x55758a1c4167  <+0x127>
    72  testl r9,r9
    75  jg 0x55758a1c40d7  B5 <+0x97>
B4:
    7b  xorl rax,rax
    7d  movq rcx,[rbp-0x18]
    81  movq rsp,rbp
    84  pop rbp
    85  cmpq rcx,0x3
    89  jg 0x55758a1c40ce  <+0x8e>
    8b  ret 0x18
    8e  pop r10
    90  leaq rsp,[rsp+rcx*8]
    94  push r10
    96  retl
```

Only a single integer subtraction in the instruction stream

No calls to helpers!

Compiler tiers




V8 architecture
Network / Cache
Parser
Ignition
parse
fetch / load
script text
AST
bytecode
interpretation
baseline execution
simple JIT
tiering
Sparkplug
TurboFan
Maglev
object shapes
machine code
optimized code
optimized JIT
type feedback
runtime / stubs / ICs
dynamic feedback
optimized execution
deopt

Garbage collection in V8

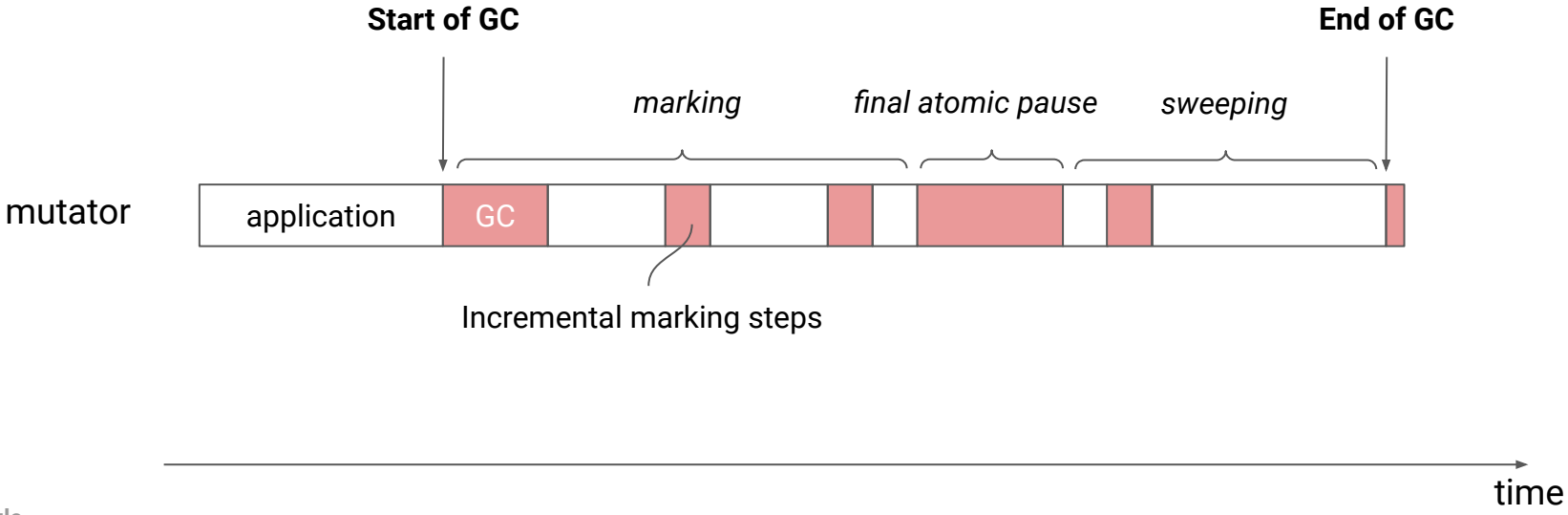
- Goals
  - Low latency (ideally sub ms)
  - “High throughput” (100s MB/s of JS)
  - Permissive in using memory when needed (foreground); aggressive shrinking when unused (background)
- Generational heap layout
  - Young generation for newly allocated data objects (~32M)
  - Old generation for long-lived objects (~2G)
- Minor GC: Only young generation
- Major GC: Both generations

Garbage collection in V8

- Goals
  - Low latency (ideally sub ms)
  - “High throughput” (100s MB/s of JS)
  - Permissive in using memory when needed (foreground); aggressive shrinking when unused (background)
- Generational heap layout
  - Young generation for newly allocated data objects (~32M)
  - Old generation for long-lived objects (~2G)
- Minor GC: Only young generation
- Major GC: Both generations

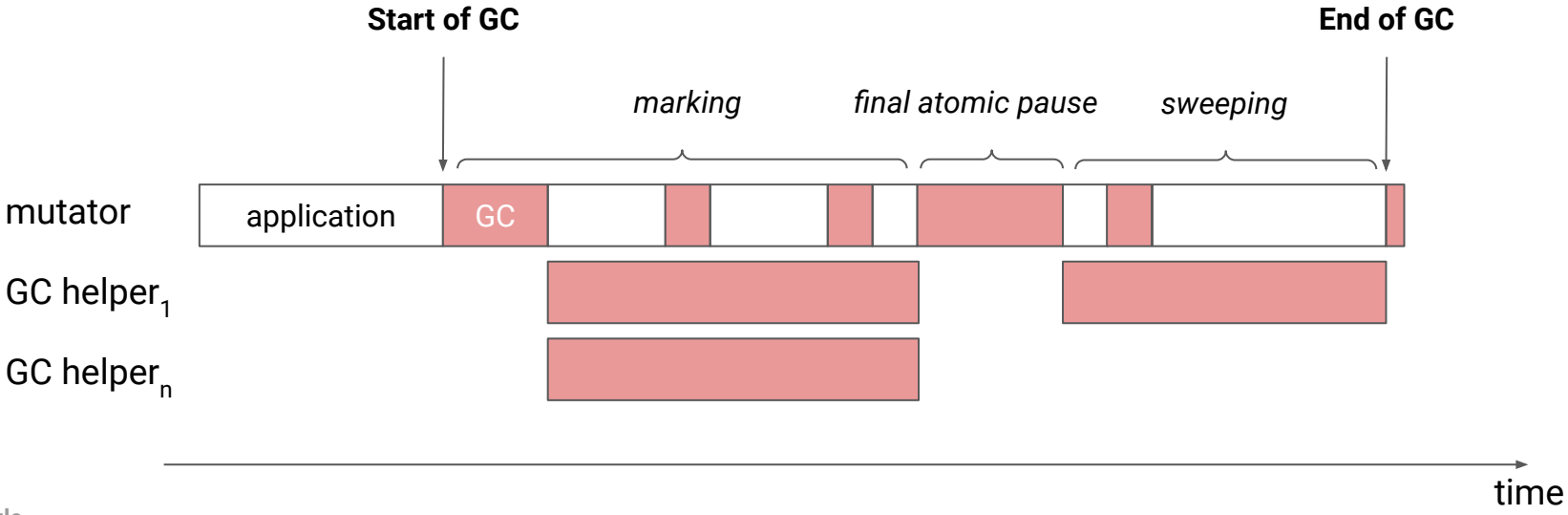
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread (for now)



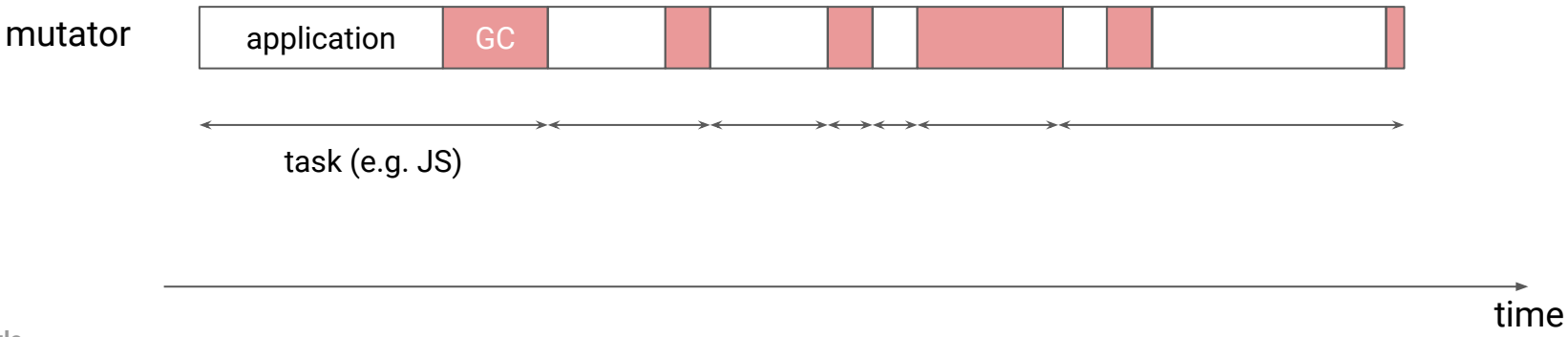
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread (for now)
- Many garbage collection threads
- Mostly concurrent



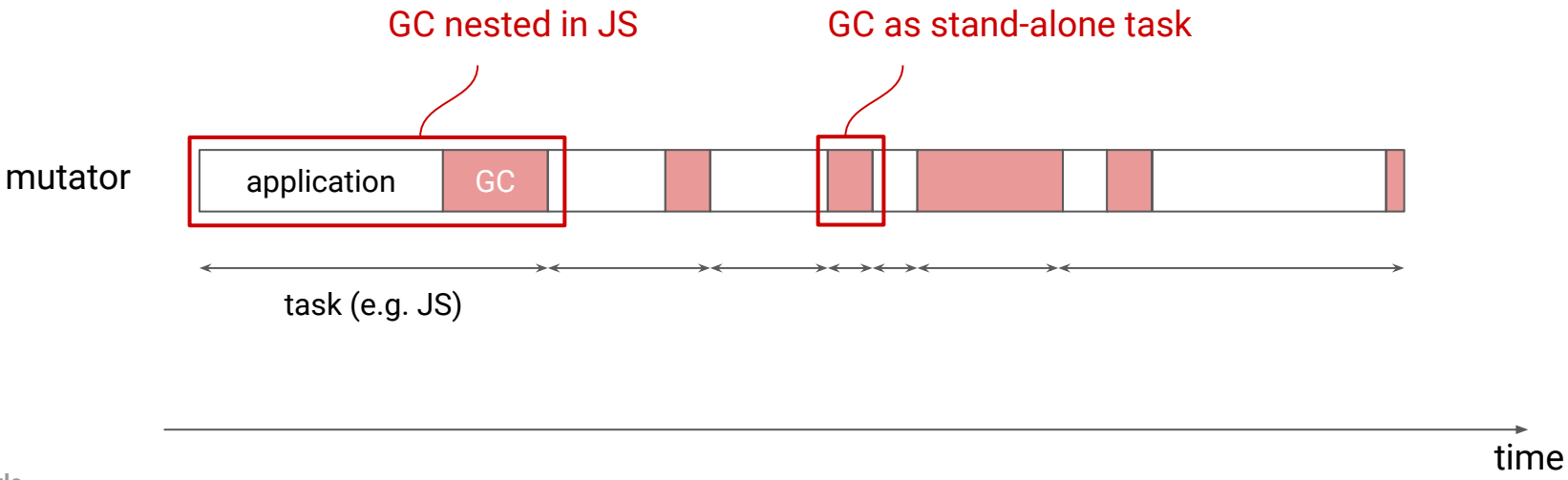
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread
- Renderer architecture: Message loop processing tasks



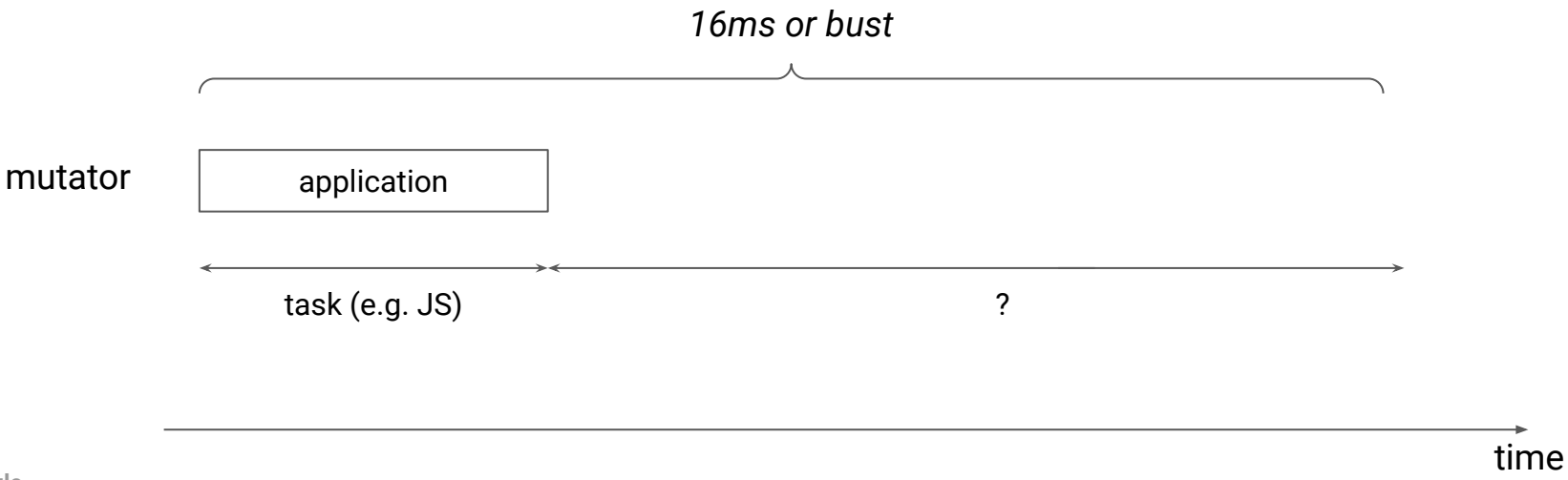
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread
- Renderer architecture: Message loop processing tasks



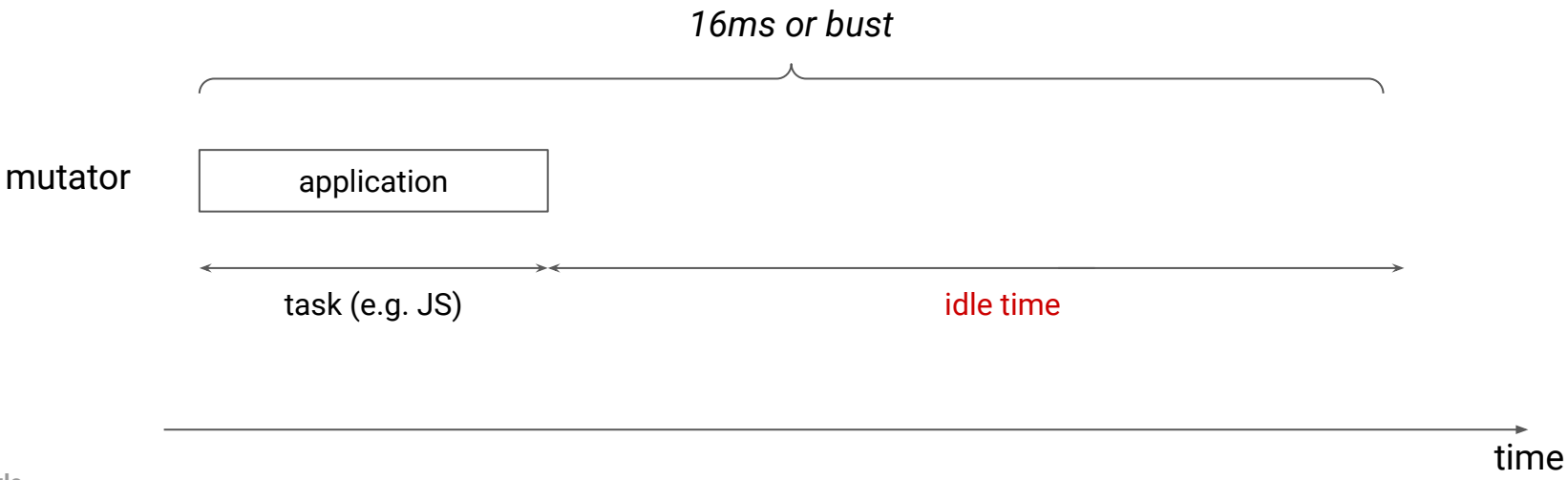
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread
- Renderer architecture: Message loop processing tasks



Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread
- Renderer architecture: Message loop processing tasks



Major garbage collection in V8: Mark-Sweep-Compact

- PLDI’16
- Maintain separate idle queues in Chrome’s scheduler to use for GC

Idle Time Garbage Collection Scheduling

Ulan Degenbaev[+] Jochen Eisinger[+] Manfred Ernst[#] Ross McIlroy[*] Hannes Payer[+]

Google Germany[+], UK[*], USA[#]

{ulan,eisinger,ernstm,rmcilroy,hpayer}@google.com

Major garbage collection in V8: Mark-Sweep-Compact

- PLDI'16
- Maintain separate idle queues in Chrome's scheduler to use for GC
- Worked really well until everybody in Chrome started using it

Idle Time Garbage Collection Scheduling

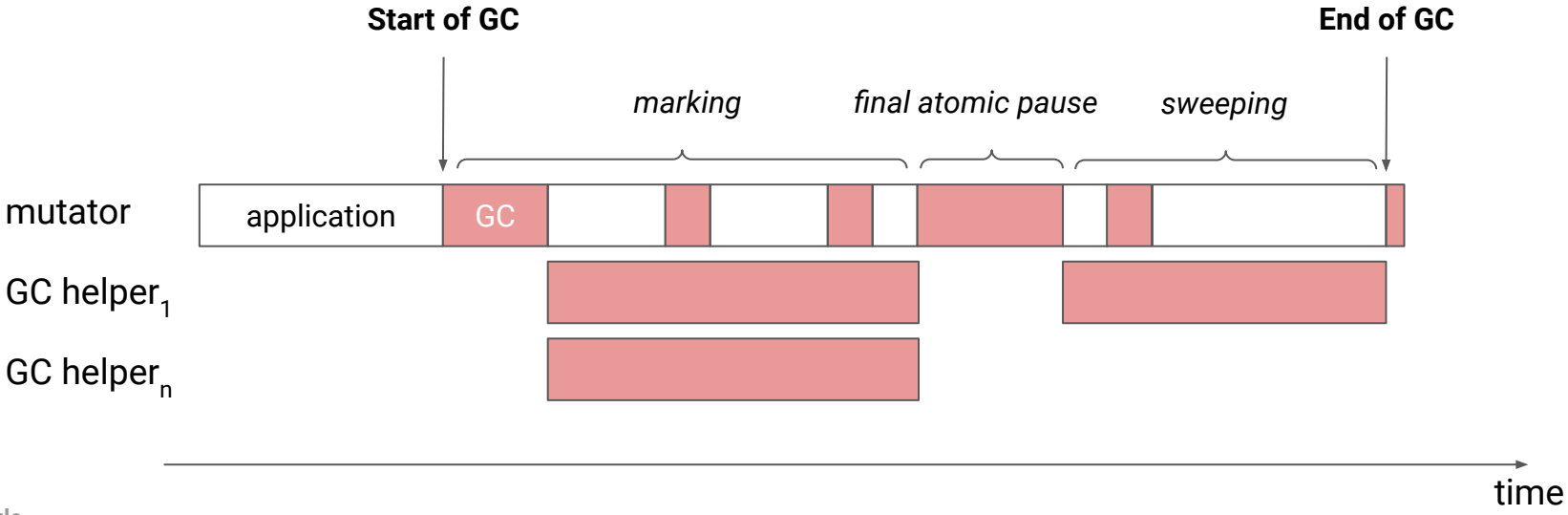
Ulan Degenbaev[+] Jochen Eisinger[+] Manfred Ernst[#] Ross McIlroy[*] Hannes Payer[+]

Google Germany[+], UK[*], USA[#]

{ulan,eisinger,ernstm,rmcilroy,hpayer}@google.com

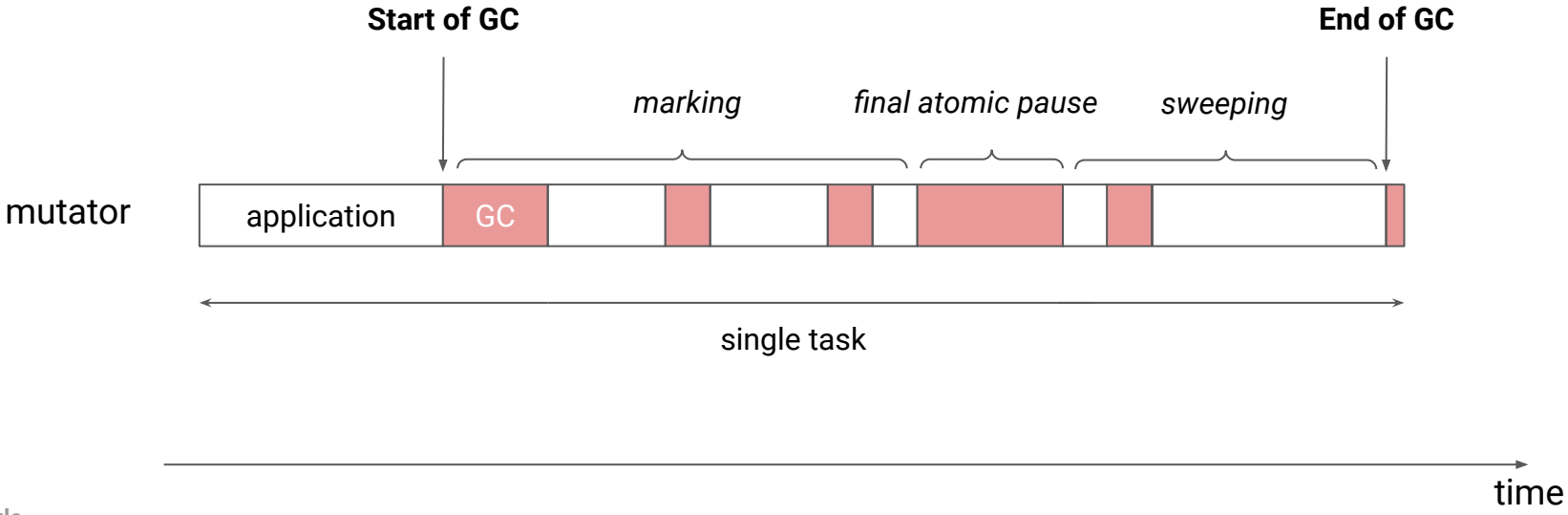
Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread (for now)
- Many garbage collection threads
- Mostly concurrent

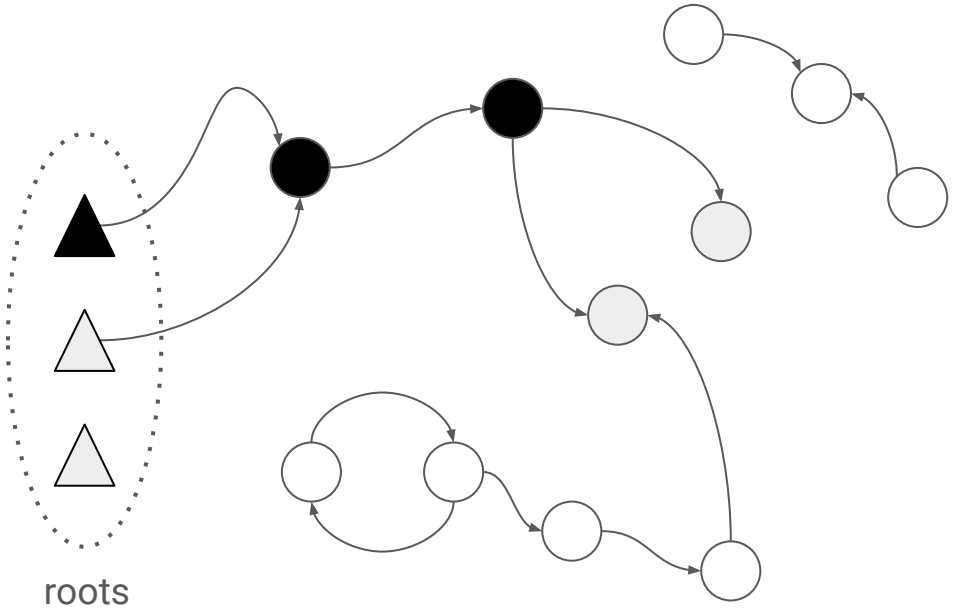


Major garbage collection in V8: Mark-Sweep-Compact

- Web: Single mutator thread (for now)
- Many garbage collection threads
- Mostly concurrent but fall back to incremental in case no tasks are available



Recap: Tri-color marking



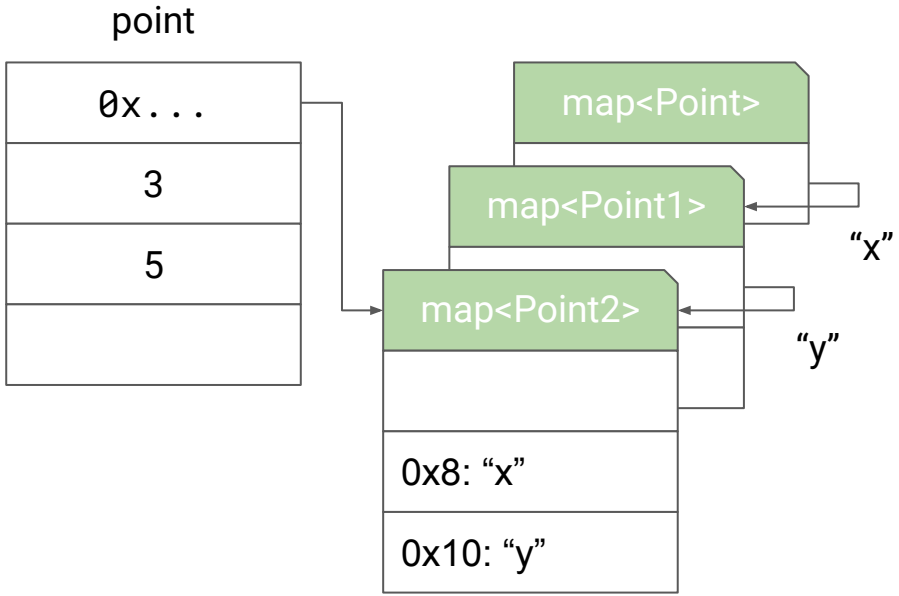
- **white**: not discovered yet
- **grey**: discovered, pushed onto the worklist
- **black**: fully processed

Strong invariant: No black to white references.

Concurrent marking in V8

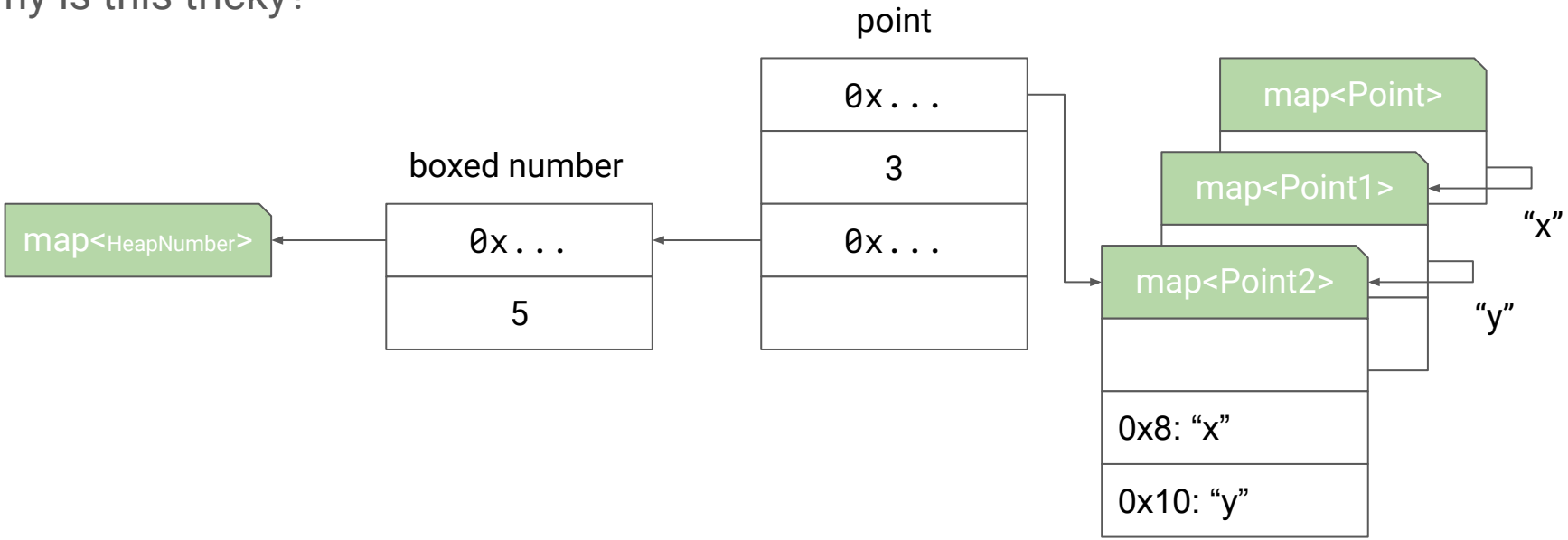
Why is this tricky?

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var point = new Point(3, 5);
```



Concurrent marking in V8

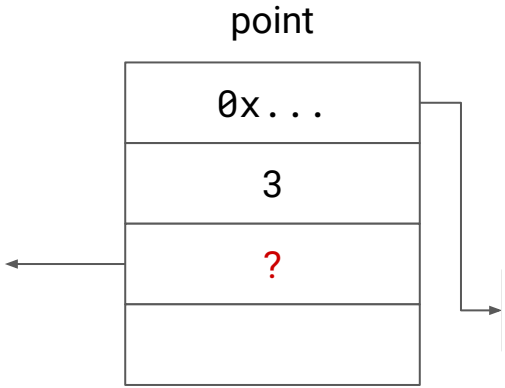
Why is this tricky?

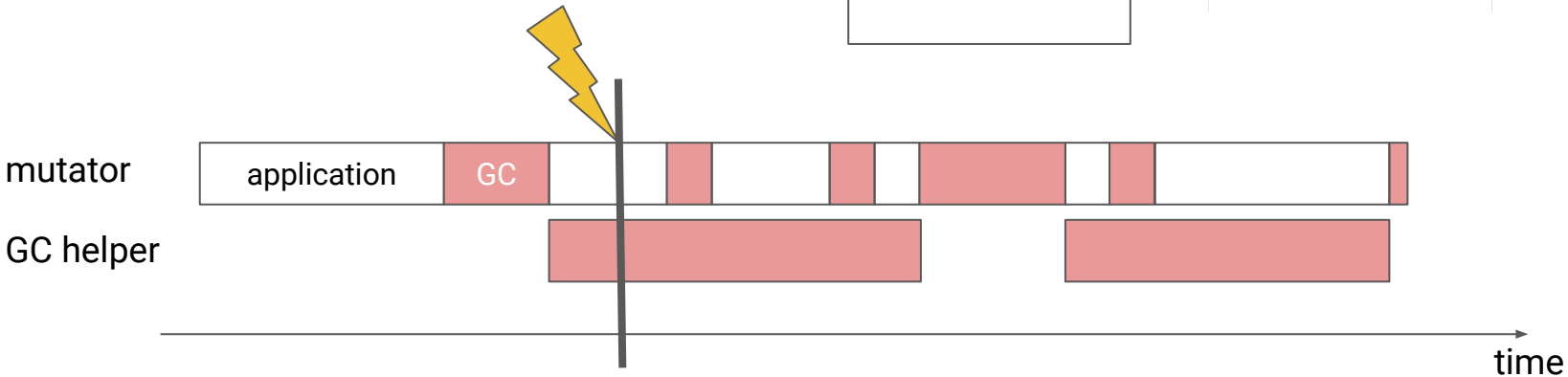


Concurrent marking in V8

Why is this tricky?

A slot in an object may change its representation.





Concurrent marking in V8

Marker

```
Mark():
  while (pop obj):
    load map = obj.map
    Visit(map, obj)

Visit(map, obj):
  snapshot = []
  for s in tagged_slots(map):
    load p = obj.slot[s]
    snapshot.add(p)
  if CAS obj.color grey => black:
    for p in snapshot:
      if CAS p.color white => grey:
        push p
```

Mutator

```
TypeChange(obj, new_map):
  CAS obj.color white => grey
  CAS obj.color grey => black
  store obj.map = new_map
  ...
```

synchronization

Concurrent marking in V8

Marker

```
Mark():
  while (pop obj):
    load map = obj.map
    Visit(map, obj)

Visit(map, obj):
  snapshot = []
  for s in tagged_slots(map):
    load p = obj.slot[s]
    snapshot.add(p)
  if CAS obj.color grey => black:
    for p in snapshot:
      if CAS p.color white => grey:
        push p
```

Mutator

```
TypeChange(obj, new_map):
  CAS obj.color white => grey
  CAS obj.color grey => black
  store obj.map = new_map
  ...
```

synchronization

Concurrent marking in V8

Marker

```
Mark():
  while (pop obj):
    load map = obj.map
    Visit(map, obj)

Visit(map, obj):
  snapshot = []
  for s in tagged_slots(map):
    load p = obj.slot[s]
    snapshot.add(p)
  if CAS obj.color grey => black:
    for p in snapshot:
      if CAS p.color white => grey:
        push p
```

Mutator

```
TypeChange(obj, new_map):
  CAS obj.color white => grey
  CAS obj.color grey => black
  store obj.map = new_map
  ...
```

synchronization

Concurrent marking in V8

Marker

```
Mark():
  while (pop obj):
    load map = obj.map
    Visit(map, obj)

Visit(map, obj):
  snapshot = []
  for s in tagged_slots(map):
    load p = obj.slot[s]
    snapshot.add(p)
  if CAS obj.color grey => black:
    for p in snapshot:
      if CAS p.color white => grey:
        push p
```

Mutator

```
TypeChange(obj, new_map):
  CAS obj.color white => grey
  CAS obj.color grey => black
  store obj.map = new_map
  ...
```

synchronization

Barriers

Original Dijkstra insertion barrier

for reasons of simplicity, the mutator shall do so independently of the color of the new edge's source.

```
store obj.slot[x] = p
if CAS p.color white=>grey:
  push p
```

Operating Systems
R.S. Gaines Editor

On-the-Fly Garbage Collection: An Exercise in Cooperation

Edsger W. Dijkstra
Burroughs Corporation

Leslie Lamport
SRI International

A.J. Martin, C.S. Scholten, and E.F.M. Steffens
Philips Research Laboratories

Barriers

Original Dijkstra insertion barrier

performance in V8

for reasons of simplicity, the mutator shall do so independently of the color of the new edge's source.

```
store obj.slot[x] = p
if CAS p.color white=>grey:
  push p
```

Operating Systems
R.S. Gaines
Editor

On-the-Fly Garbage Collection: An Exercise in Cooperation

Edsger W. Dijkstra
Burroughs Corporation

Leslie Lamport
SRI International

A.J. Martin, C.S. Scholten, and
E.F.M. Steffens
Philips Research Laboratories

Concurrent marking in V8

- ISMM'19
- Manual proofs

Invariant 4.4 (Reference Safety). *The value passed to the FollowReference is a tagged value, that is* $\mathcal{A}(value) \in \{Ref, Smi\}$.

Concurrent Marking of Shape-Changing Objects

Ulan Degenbaev
Google
Germany
ulan@google.com

Michael Lippautz
Google
Germany
mlippautz@google.com

Hannes Payer
Google
Germany
hpayer@google.com

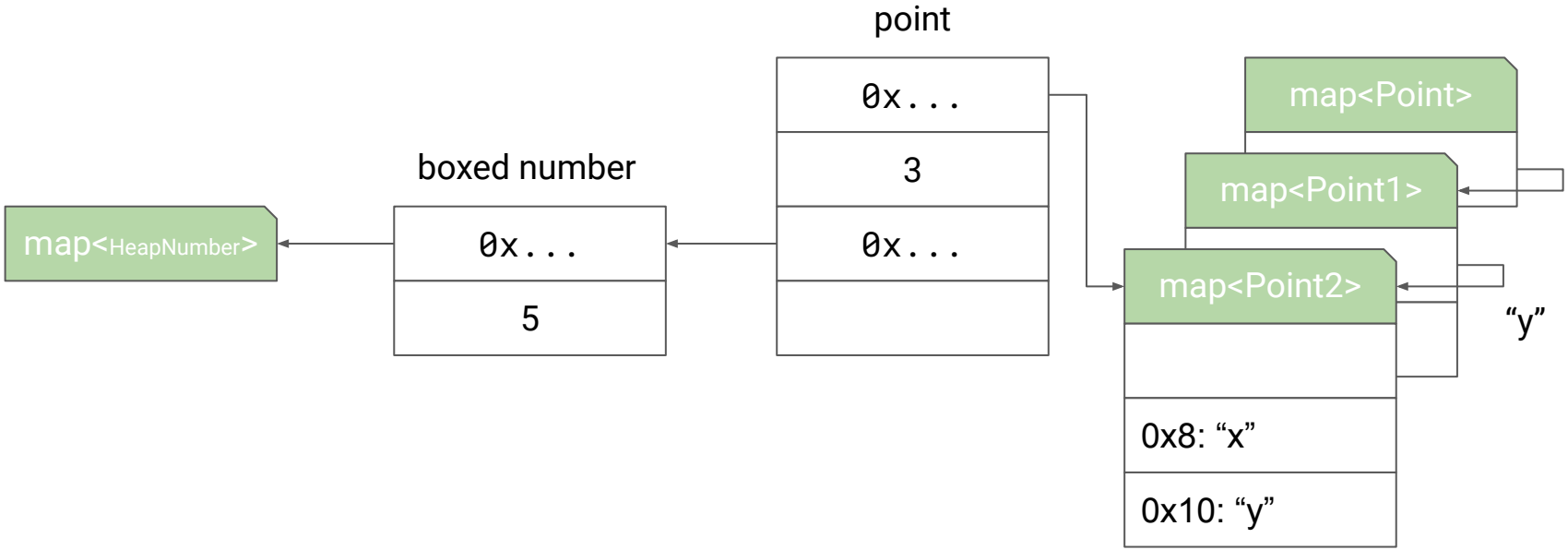
Double unboxing is long gone!

Interlude: Pointer compression

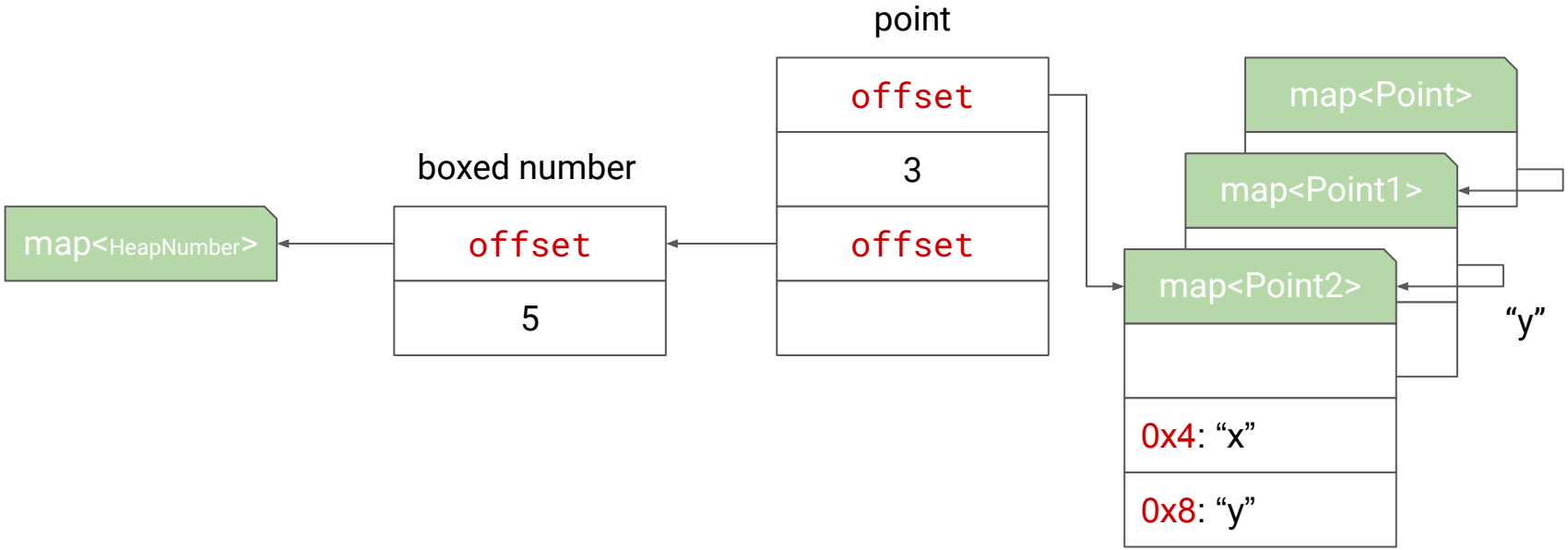
It is absolutely idiotic to have 64-bit pointers when I compile a program that uses less than 4 gigabytes of RAM. When such pointer values appear inside a struct, they not only waste half the memory, they effectively throw away half of the cache.

– Knuth 2008

Interlude: Pointer compression



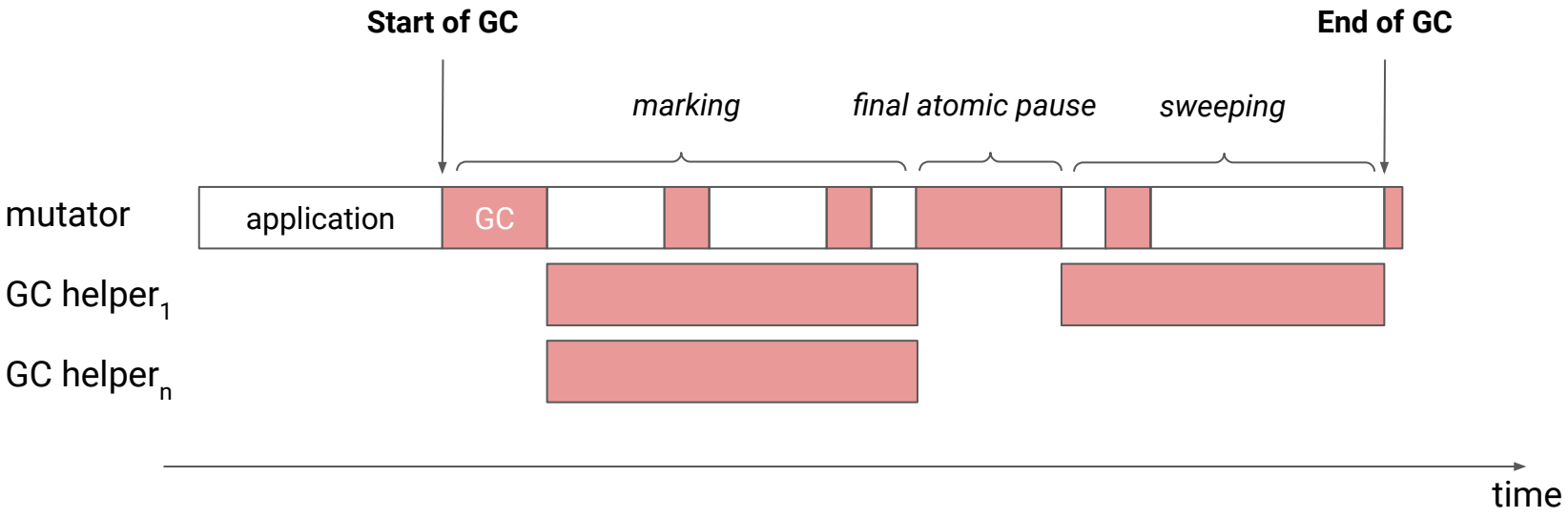
Interlude: Pointer compression



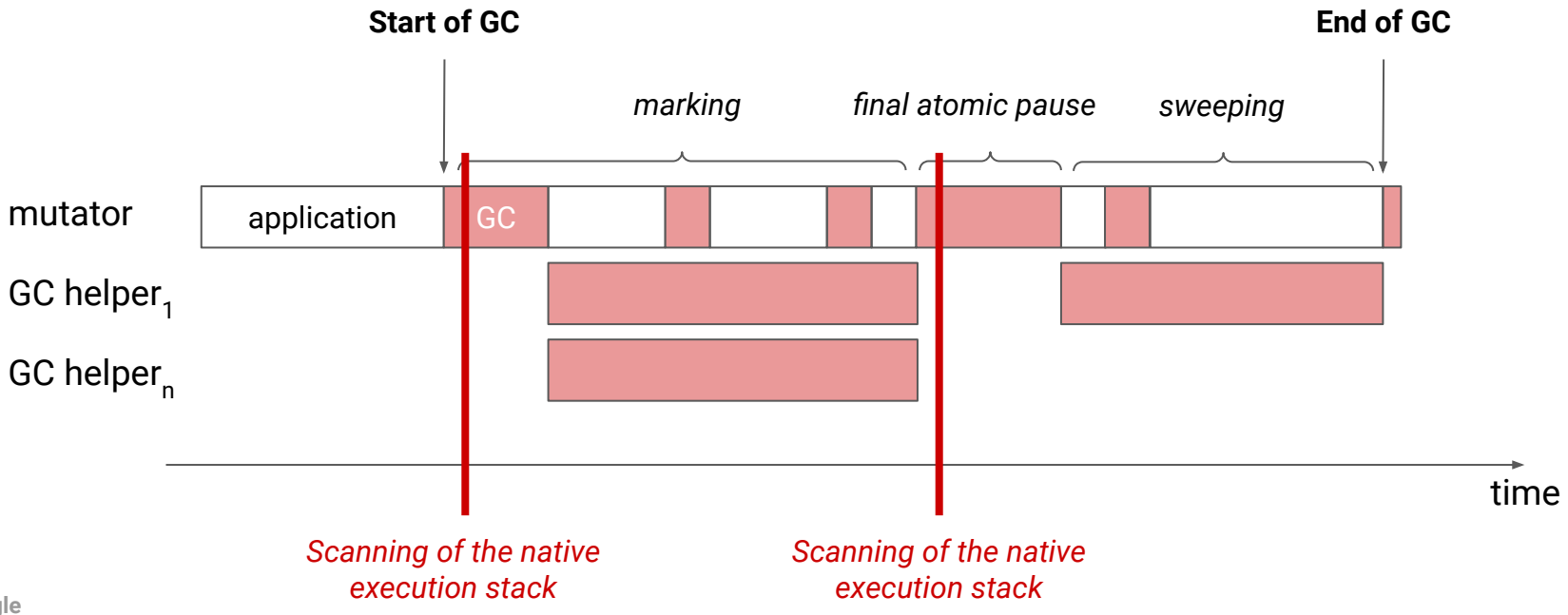
Interlude: Pointer compression

- Only available on 64bit platforms
- Heap is limited to 4GiB in size
- Pointers are compressed into `base + offset`
- Offset is 32 bit in size

Another case study: Minimizing stack scanning



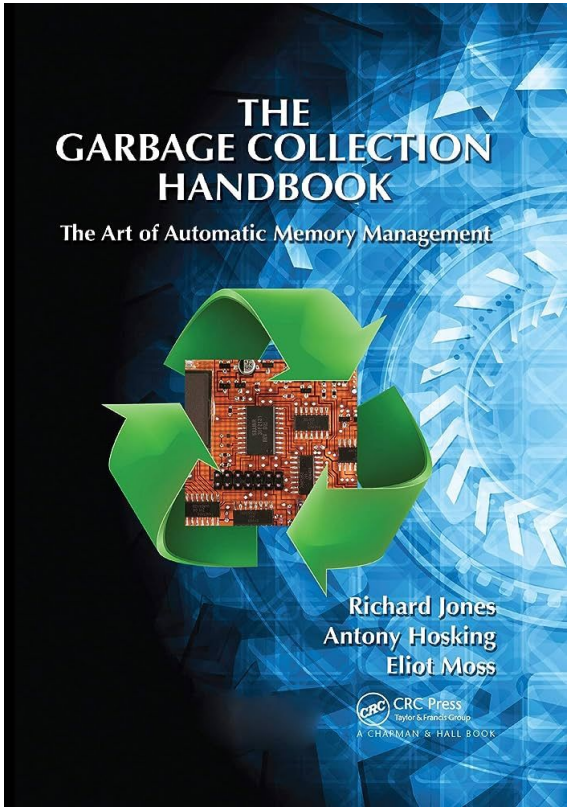
Another case study: Minimizing stack scanning



Another case study: Minimizing stack scanning

Literature

- Snapshot-at-the-beginning garbage collection
- Use deletion barrier [Yuasa]

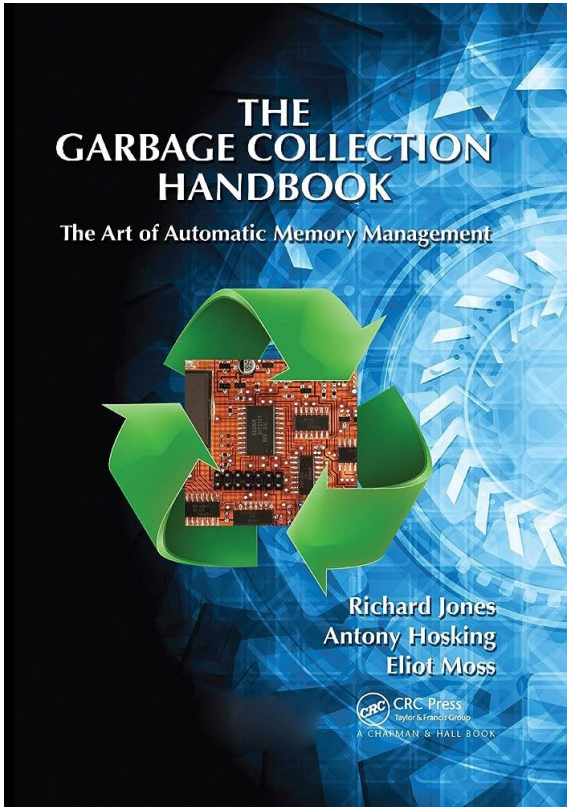


Another case study: Minimizing stack scanning

Literature

- Snapshot-at-the-beginning garbage collection
- Use deletion barrier [Yuasa]

Hooray!



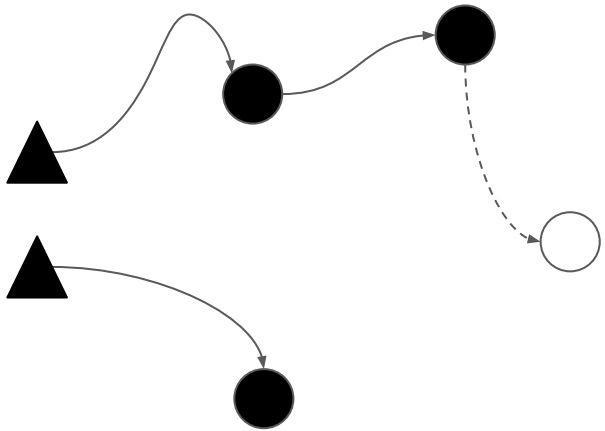
Another case study: Minimizing stack scanning

Here’s what literature doesn’t tell you: Doesn’t work with weak references

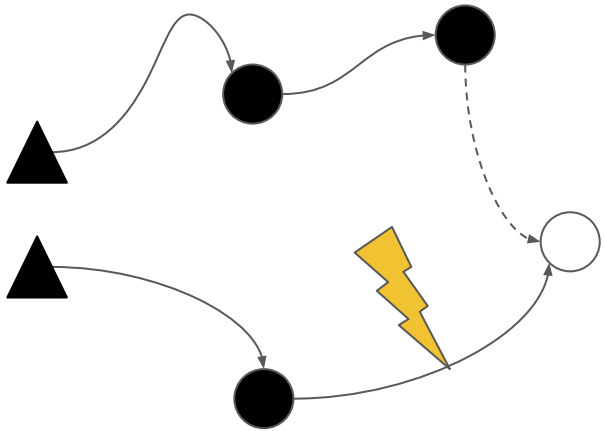
Why?

- Deletion barrier assumes liveness witness
- Weak references are not a liveness witness

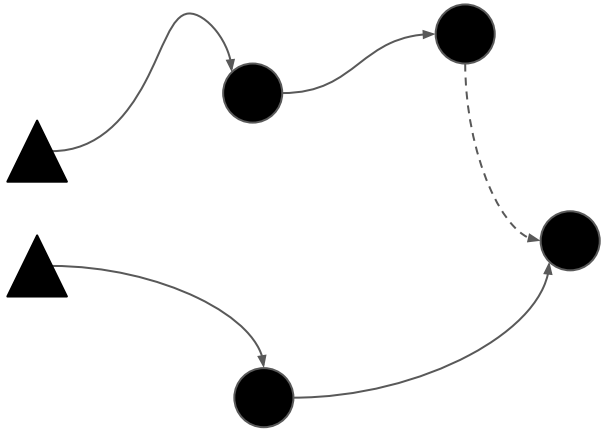
Deletion barrier is not sufficient for weak handling



Deletion barrier is not sufficient for weak handling

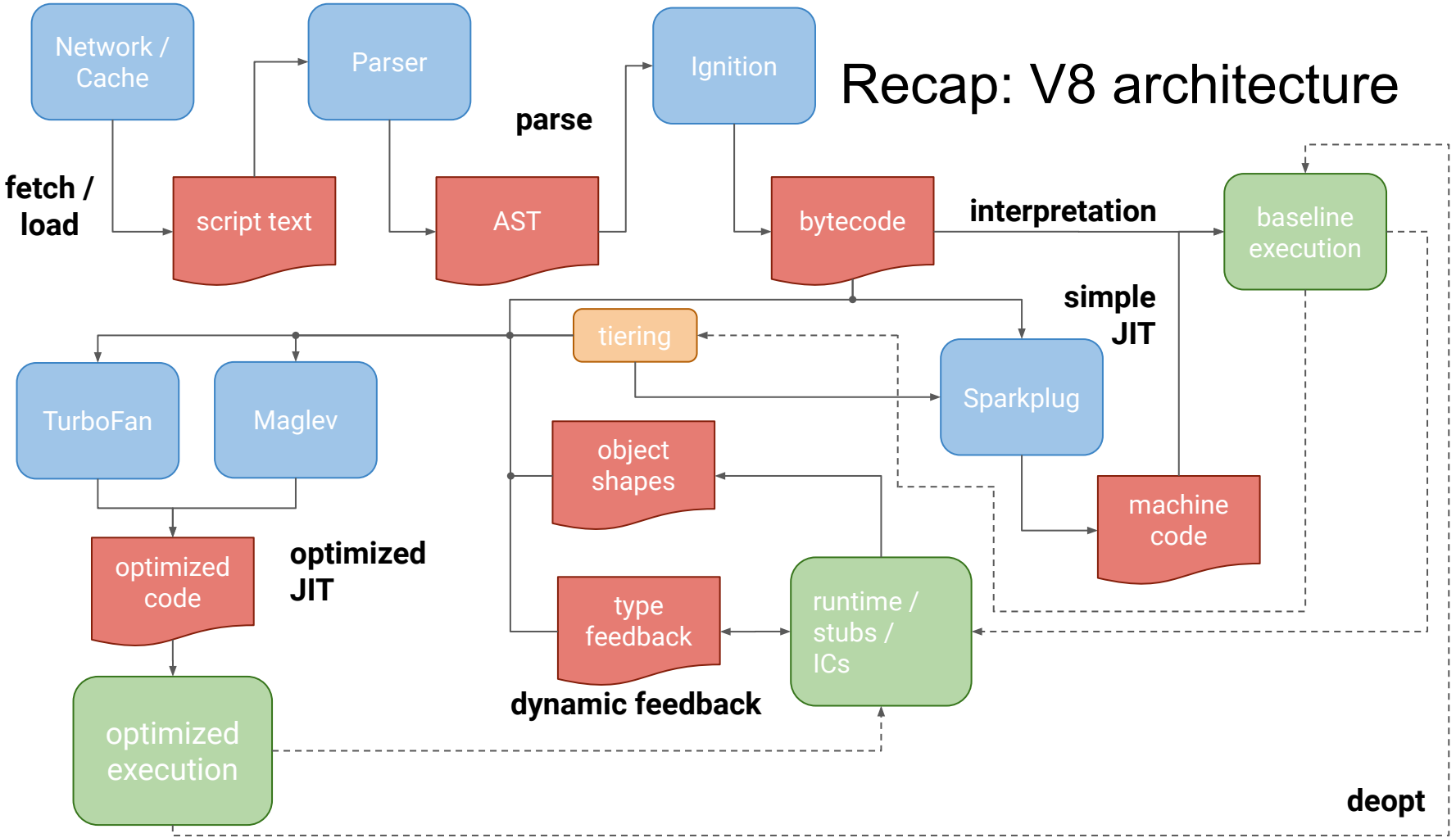


Deletion barrier + read barrier on weak references



Retrofitting a read barrier is hard

Recap: V8 architecture



Current exploration: Conservative stack scanning

- V8 has precise stack information of stack layout
- Expensive to maintain (handle abstraction)
- Minor GC implemented via semi-space copy

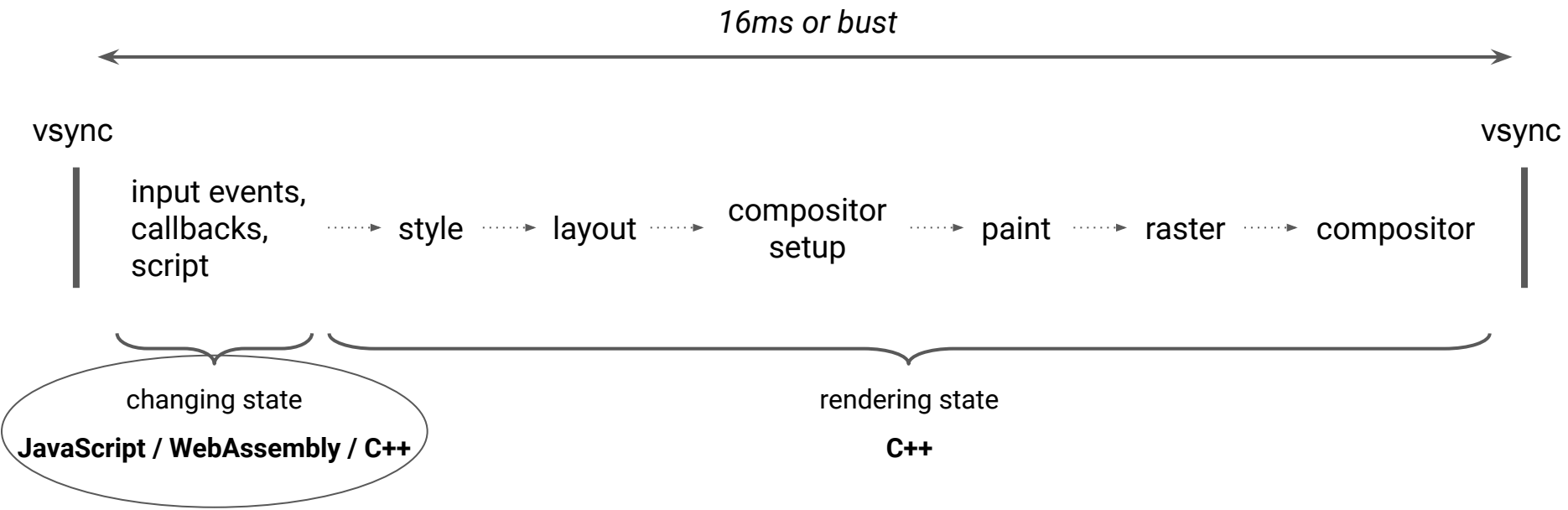
Going forward

1. Switch to non-moving young generation GC
    - Use sticky mark bits
2. Ditch handles and rely on conservatively scanning the stack to find pointers

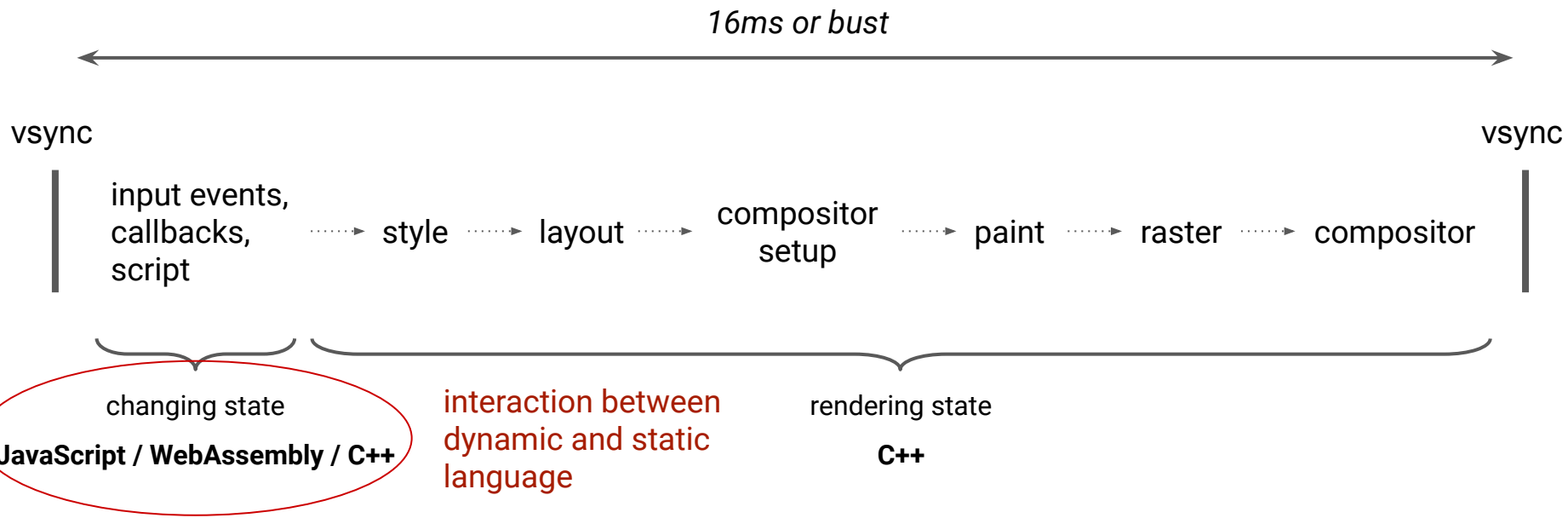
GC is not a solved problem! (but there's a lot of literature out there to appreciate)

Changing gears: JavaScript FFI

Recall: Rendering

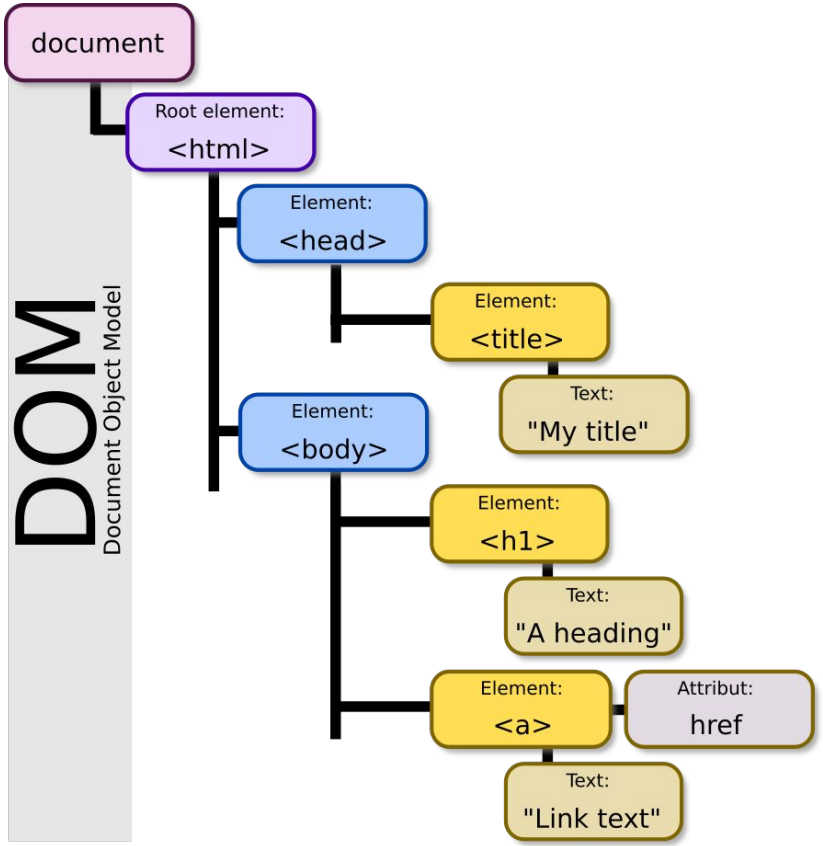


Recall: Rendering



Changing state: DOM

- Document Object Model (DOM)
- Cross-platform language-independent representation of HTML
- Web Interface Definition Language (IDL)
- Encoded in C++ and picked up by the rendering pipeline

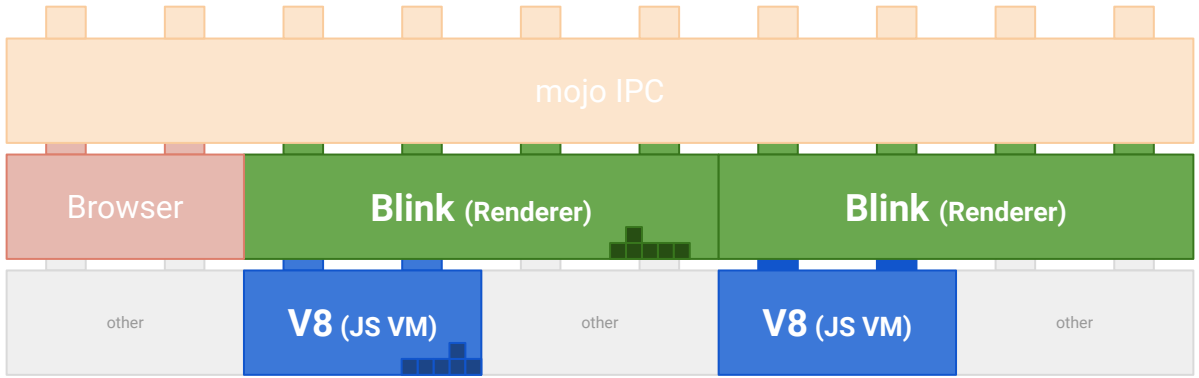


Changing state: JavaScript

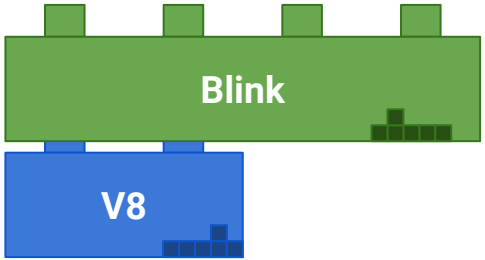
- Lingua franca of the web
- Used to interact with the DOM
- Untrusted code executed in an isolated environment, the virtual machine of a browser

```
<script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
    document.body.appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script>
```

Chrome architecture

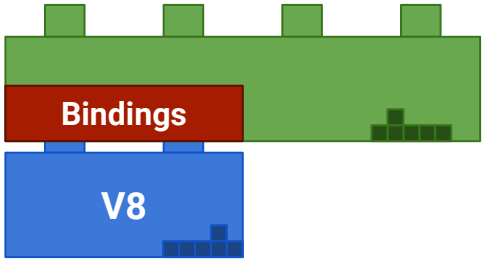


Changing state



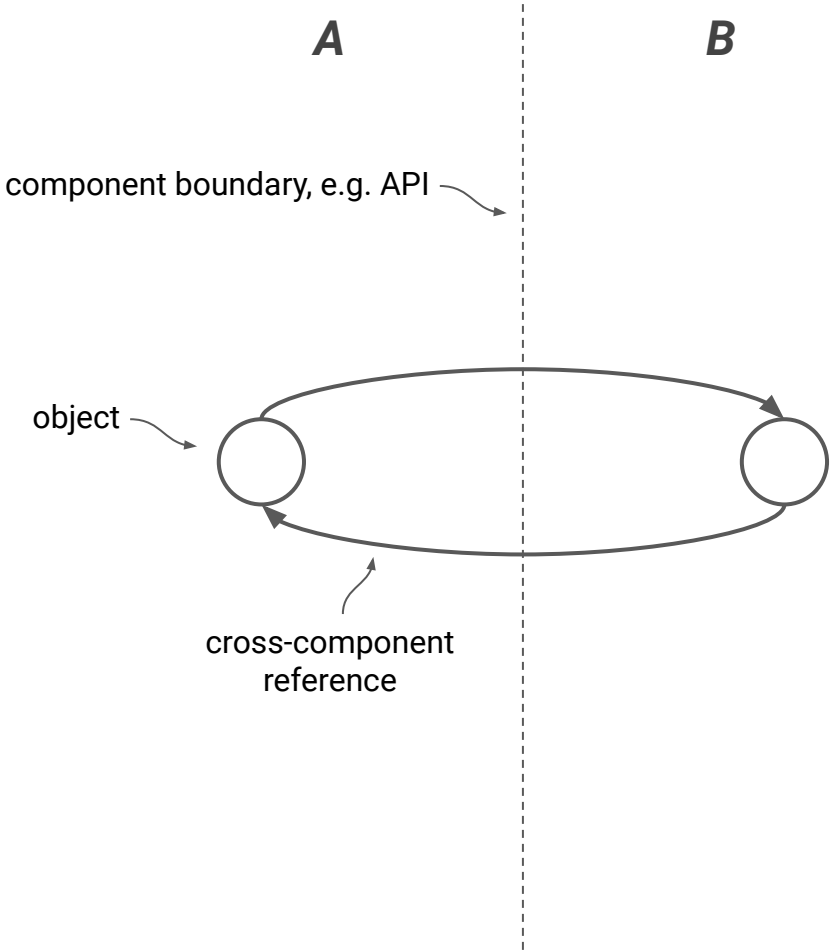
Changing state

- Bindings layer glues JS to C++
- Hard API boundary to V8



Problem

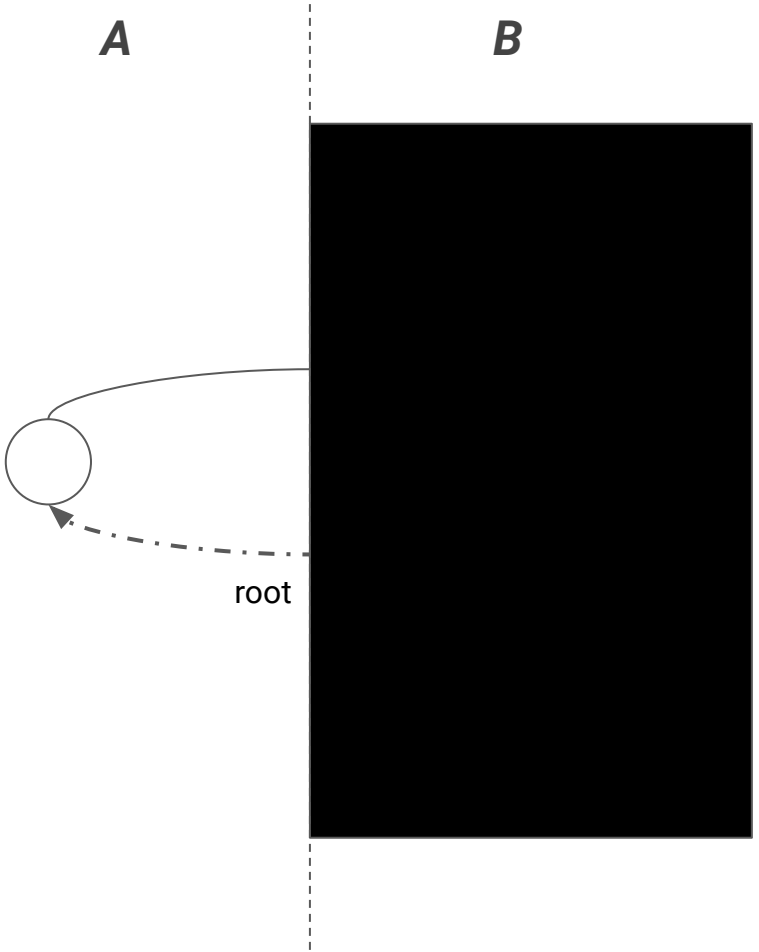
- Software split in components with hard boundaries
- Ways to compose arbitrary object graphs across components
- No static ownership of memory
- Potentially differently managed environments



Problem

- Software split in components with hard boundaries
- Ways to compose arbitrary object graphs across components
- No static ownership of memory
- Potentially differently managed environments

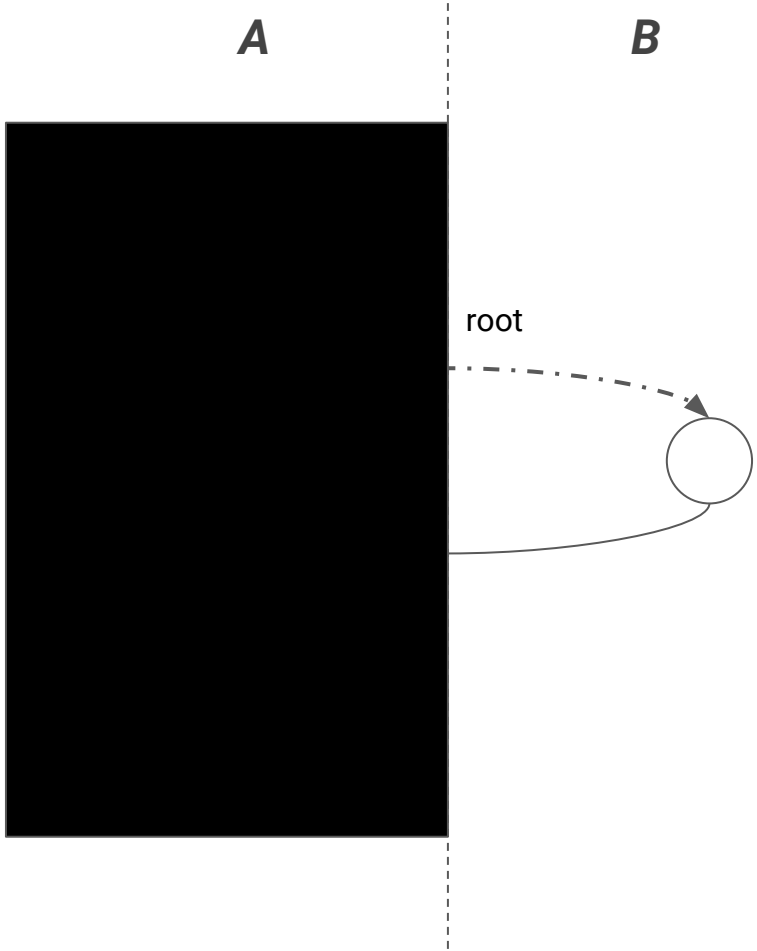
Unknown references create roots into environment



Problem

- Software split in components with hard boundaries
- Ways to compose arbitrary object graphs across components
- No static ownership of memory
- Potentially differently managed environments

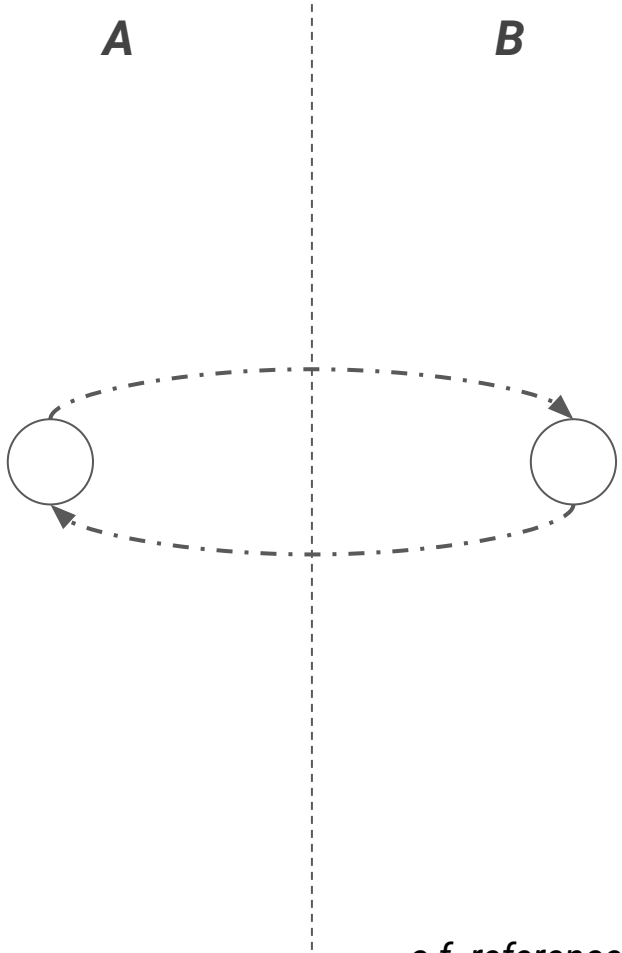
Unknown references create roots into environment



Problem

- Software split in components with hard boundaries
- Ways to compose arbitrary object graphs across components
- No static ownership of memory
- Potentially differently managed environments

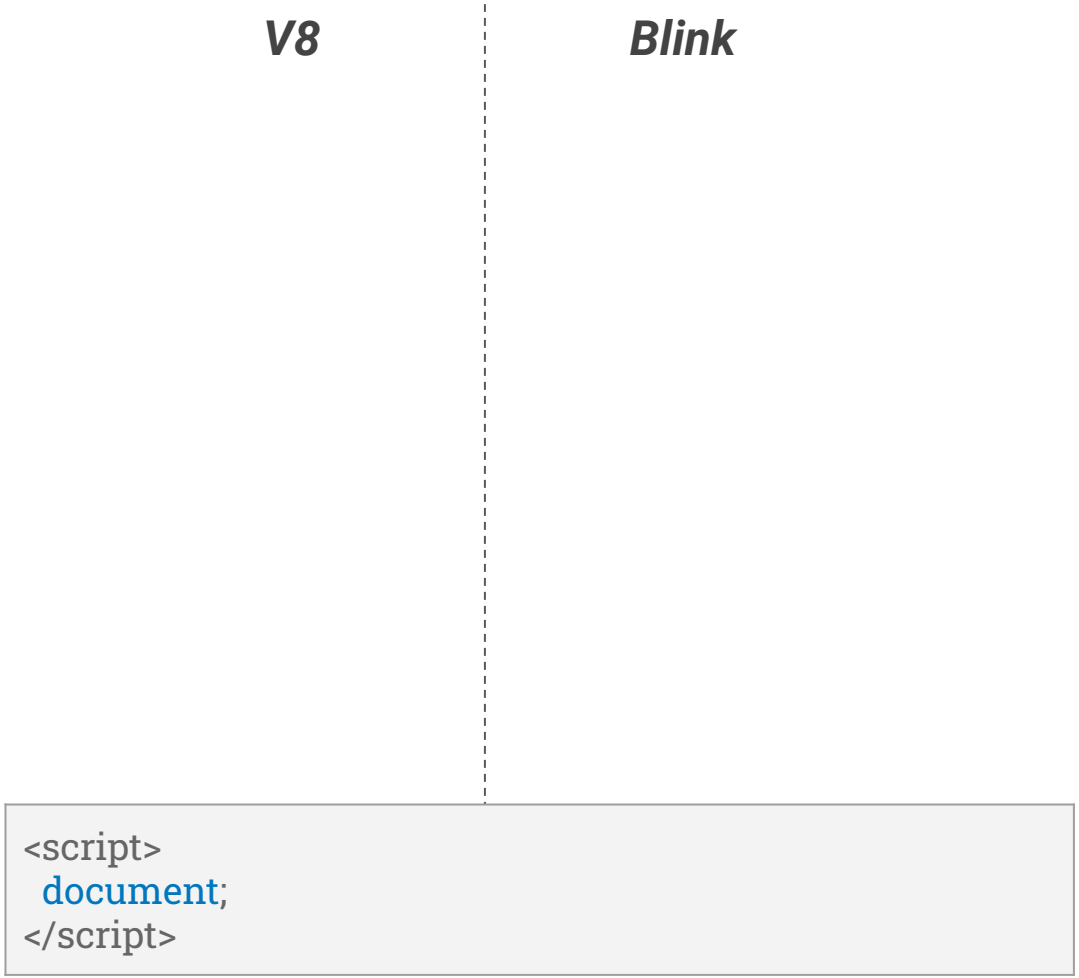
Unknown references create roots into environment



c.f. reference counting cycles

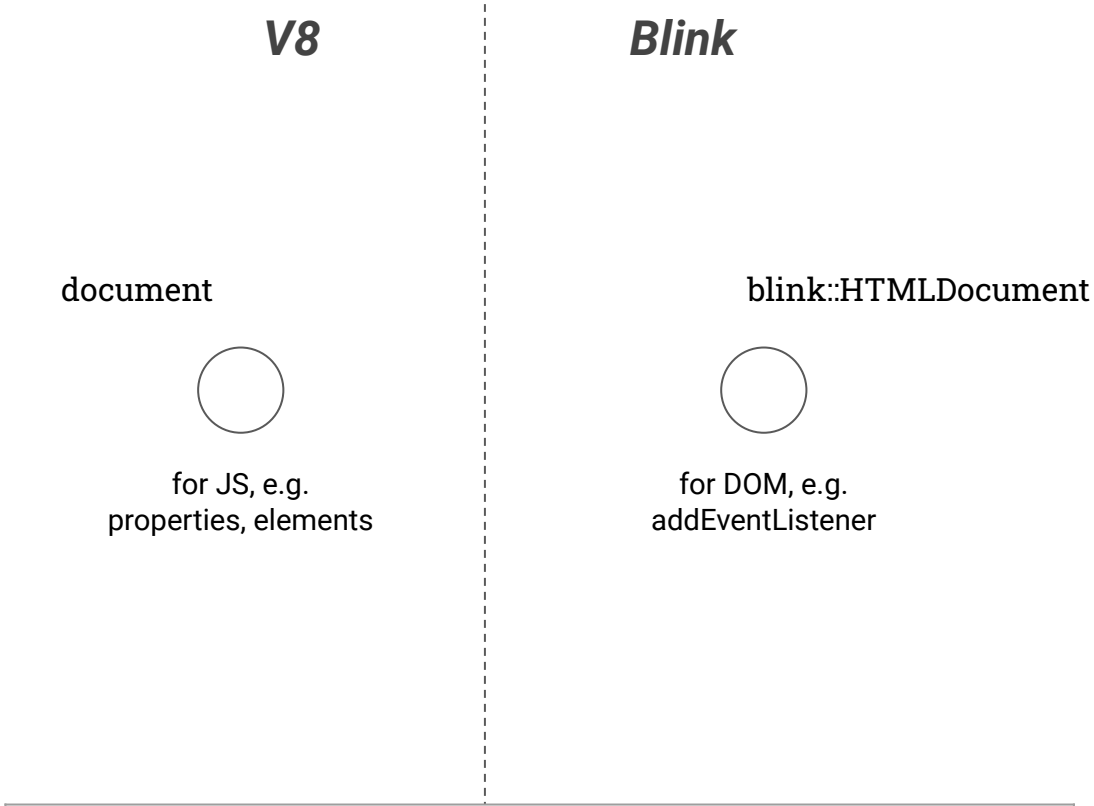
The bond

- JavaScript ⇔ DOM



The bond

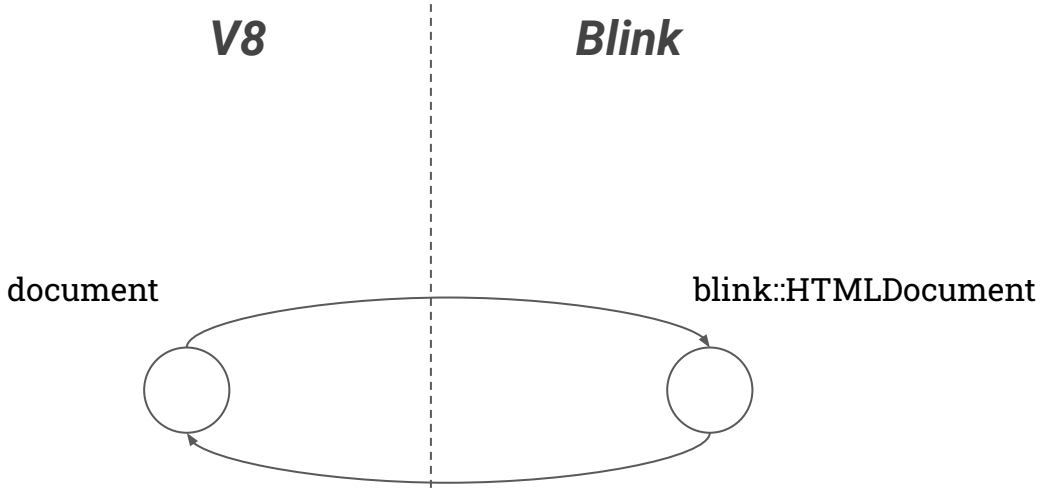
- JavaScript ⇔ DOM
- Objects come in halves




```
<script>
  document;
</script>
```

The bond

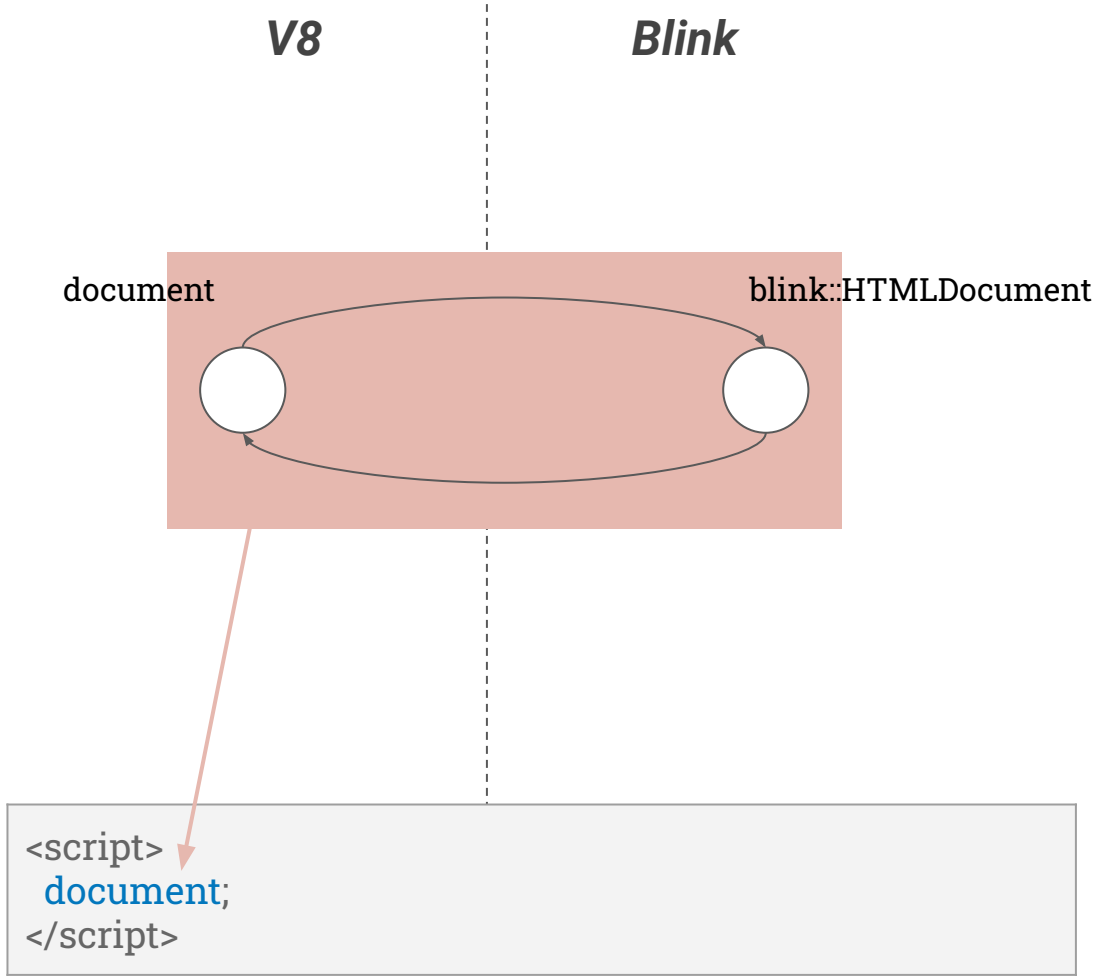
- JavaScript ⇔ DOM
- Objects come in halves
- Reference each other




```
<script>
  document;
</script>
```

The bond

- JavaScript ⇔ DOM
- Objects come in halves
- Reference each other



Is this an actual problem?

Example

Mixing server-side and client-side rendering

```
<!DOCTYPE html>
<head><script>
 function createDiv() {
  let newDiv =
     document.createElement("div");
  document.body
     .appendChild(newDiv);
 }

 document.addEventListener(
    "DOMContentLoaded", createDiv);
</script></head>
<body>
 <span></span>
</body>
</html>
```

V8

Blink

Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
     document.body
         .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```

Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
       document.createElement("div");
     document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```

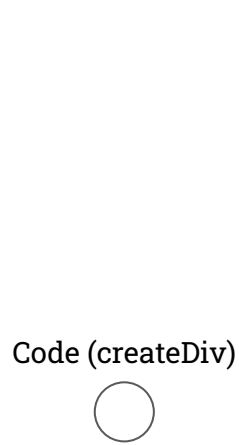
V8

Blink

blink::HTMLDocument

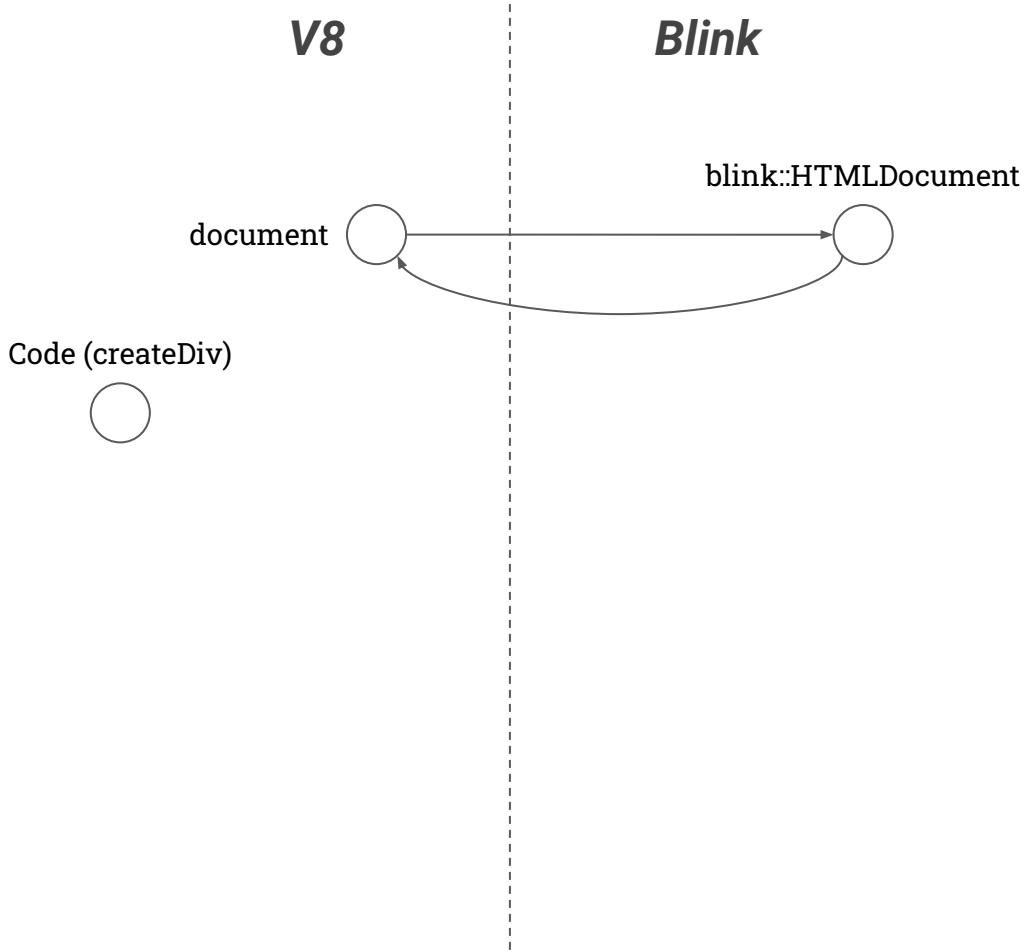
Example

```
<!DOCTYPE html>
<head><script>
 function createDiv() {
  let newDiv =
     document.createElement("div");
   document.body
      .appendChild(newDiv);
 }

 document.addEventListener(
    "DOMContentLoaded", createDiv);
</script></head>
<body>
 <span></span>
</body>
</html>
```



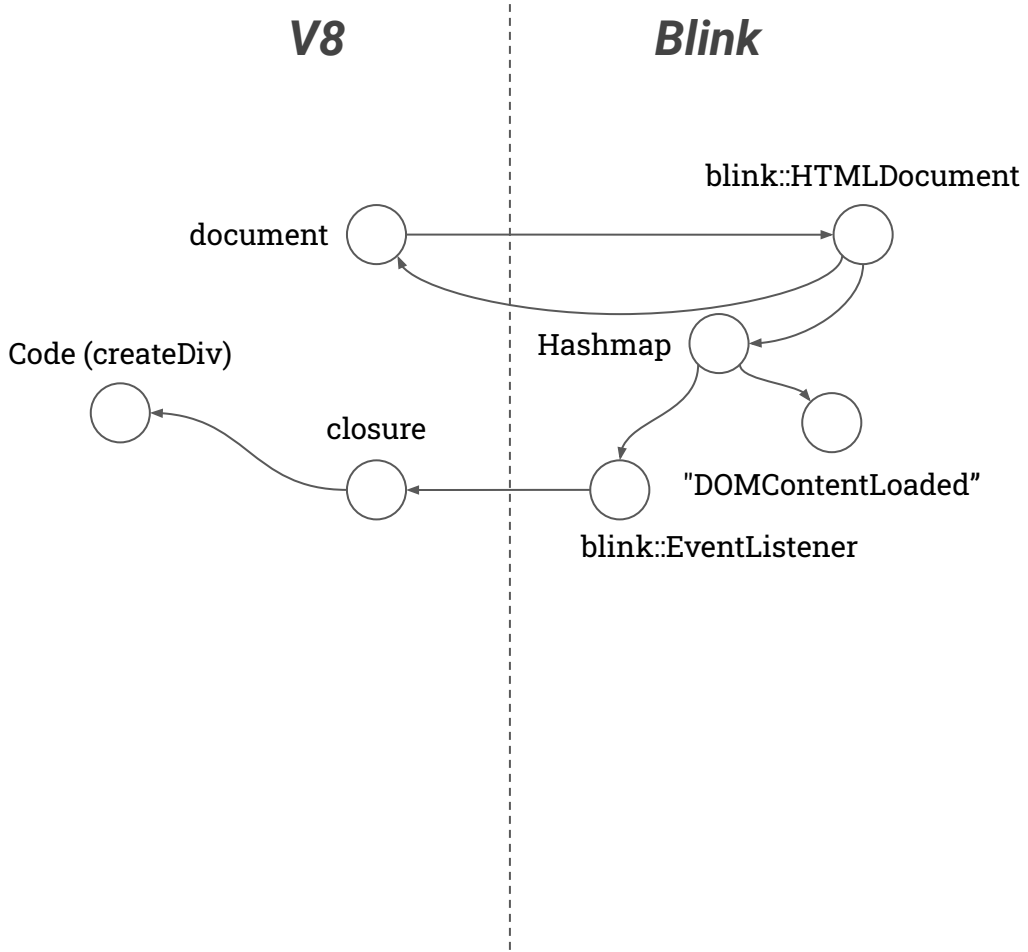
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
    document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



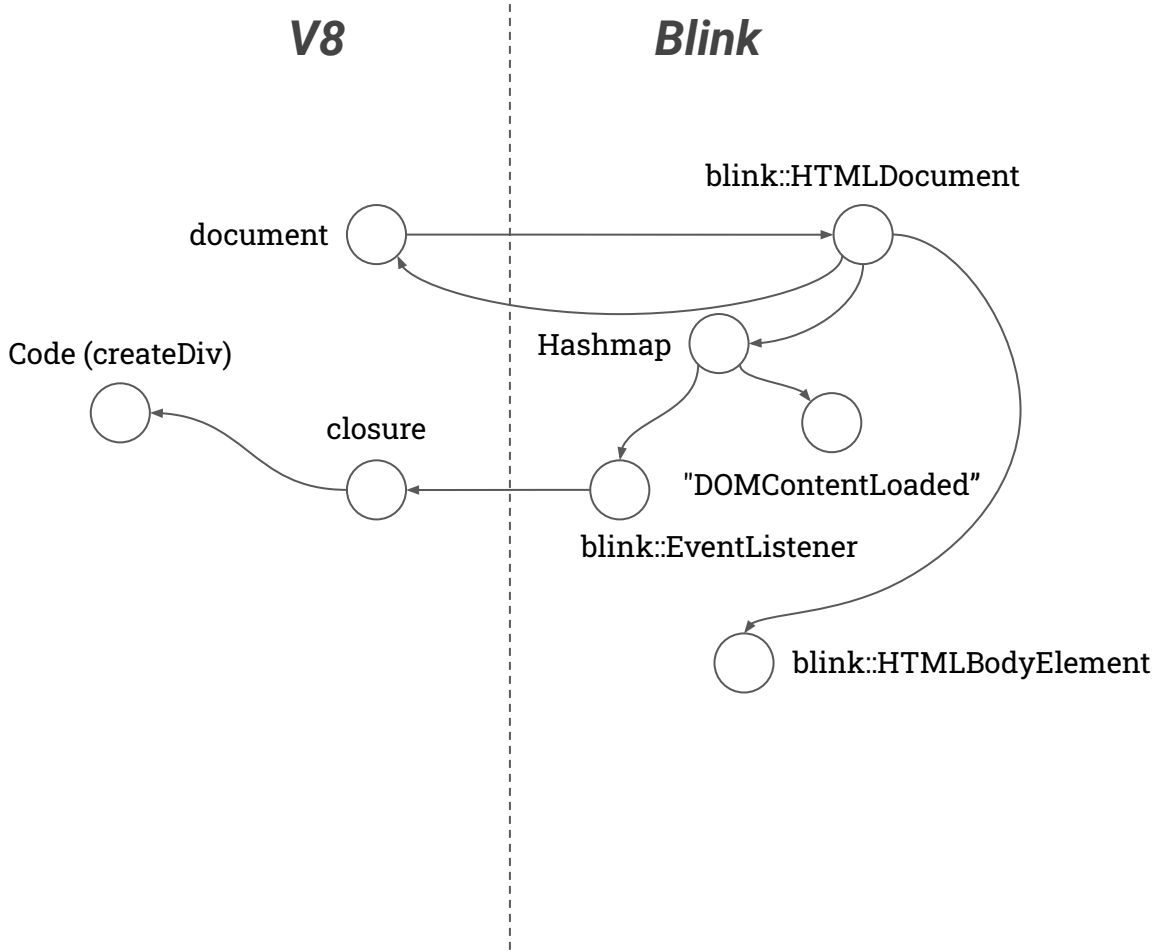
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
       document.createElement("div");
     document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
     "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



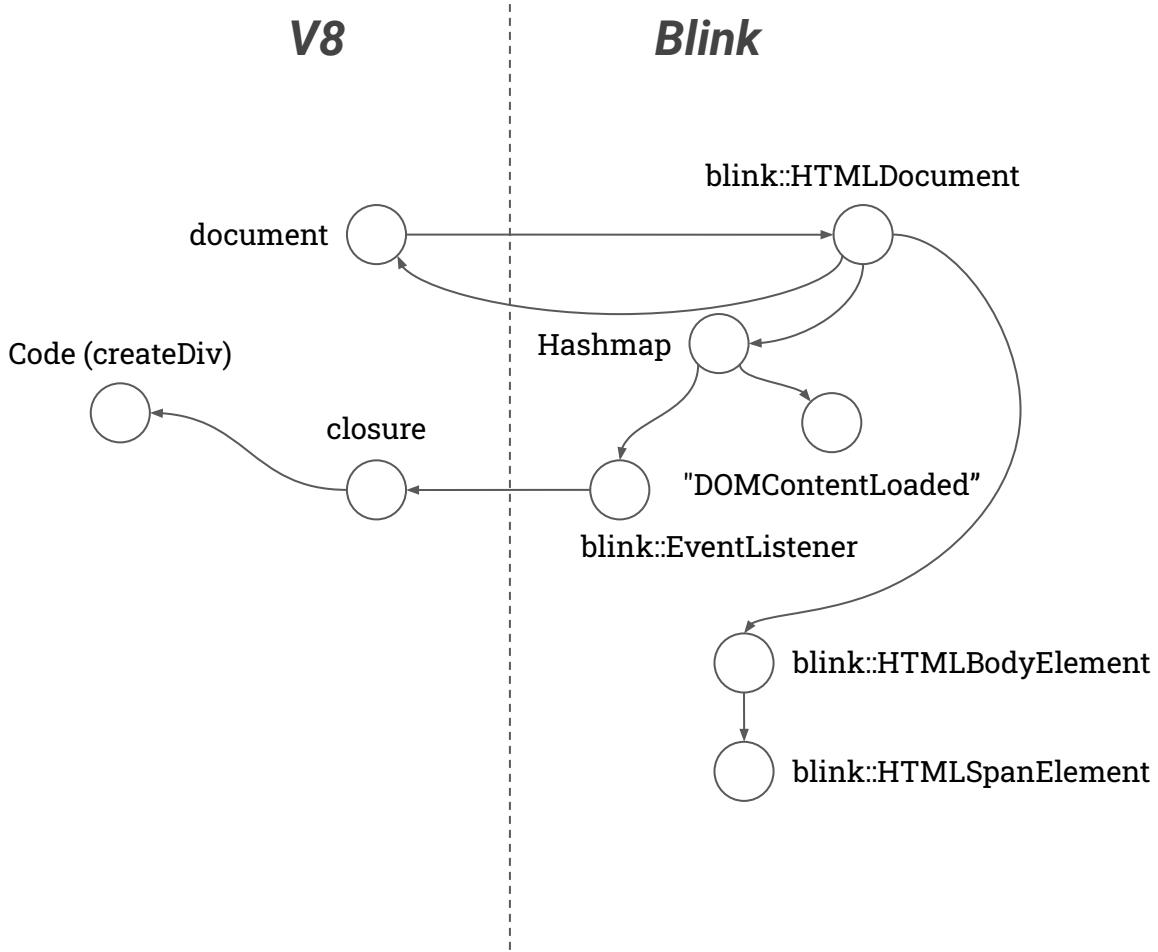
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
       document.createElement("div");
     document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



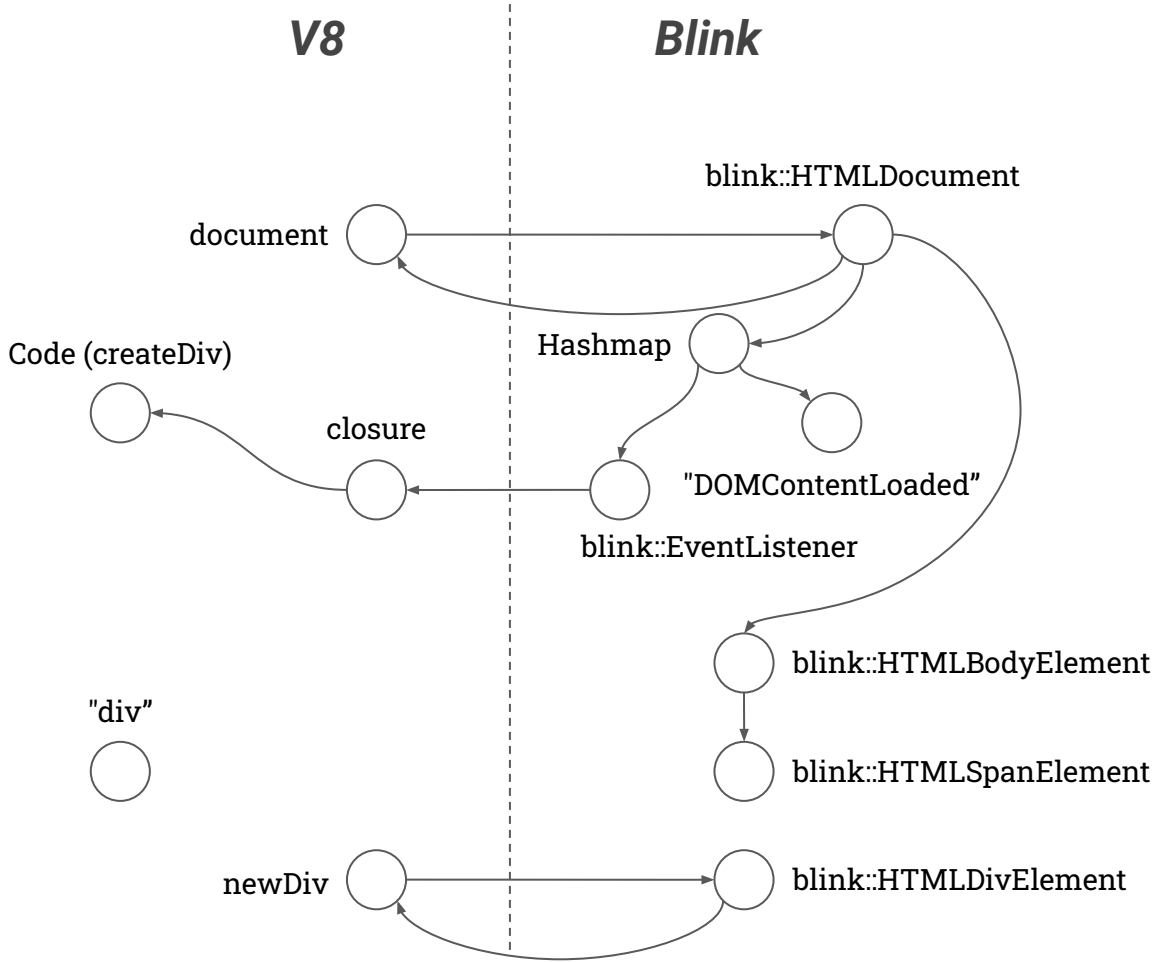
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
       document.createElement("div");
     document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



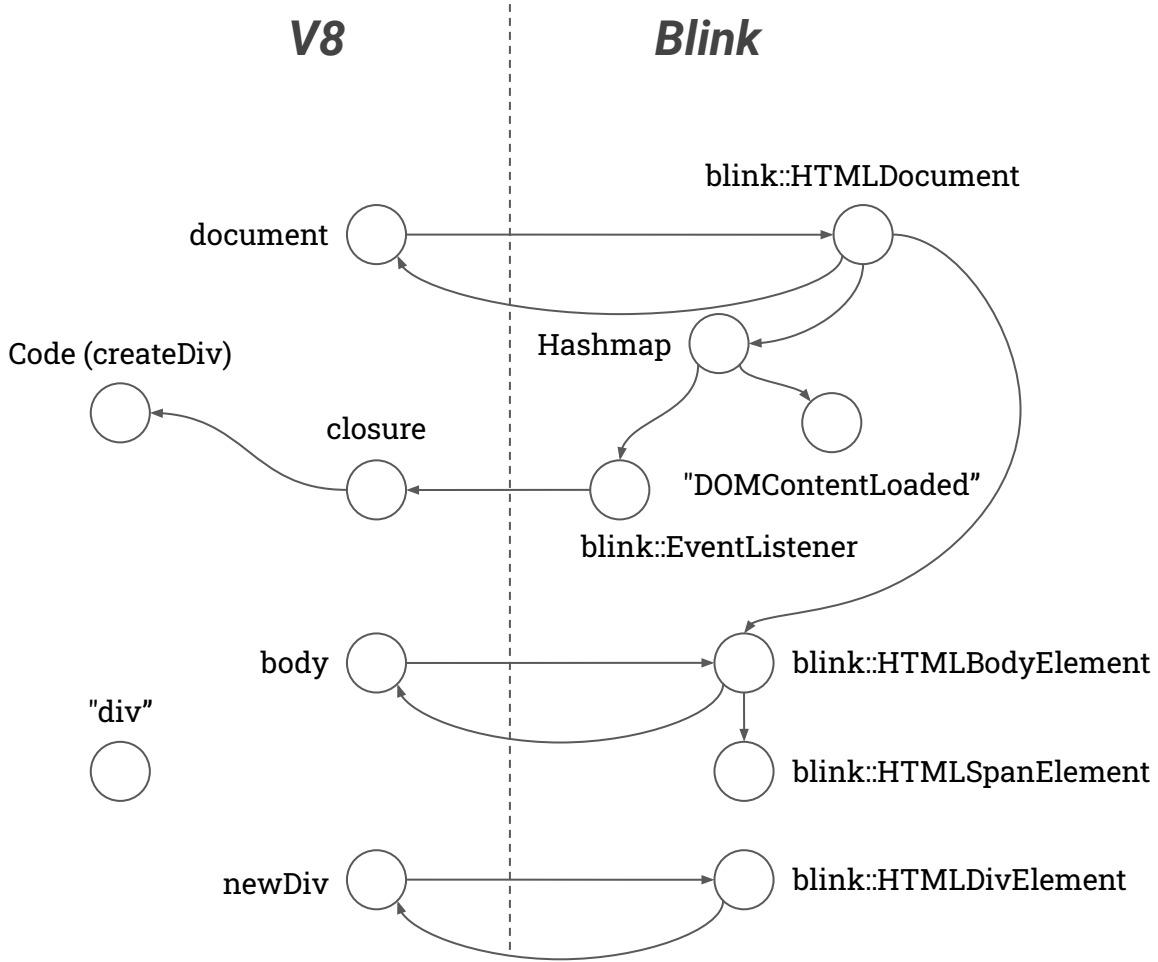
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
      document.createElement("div");
    document.body
      .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



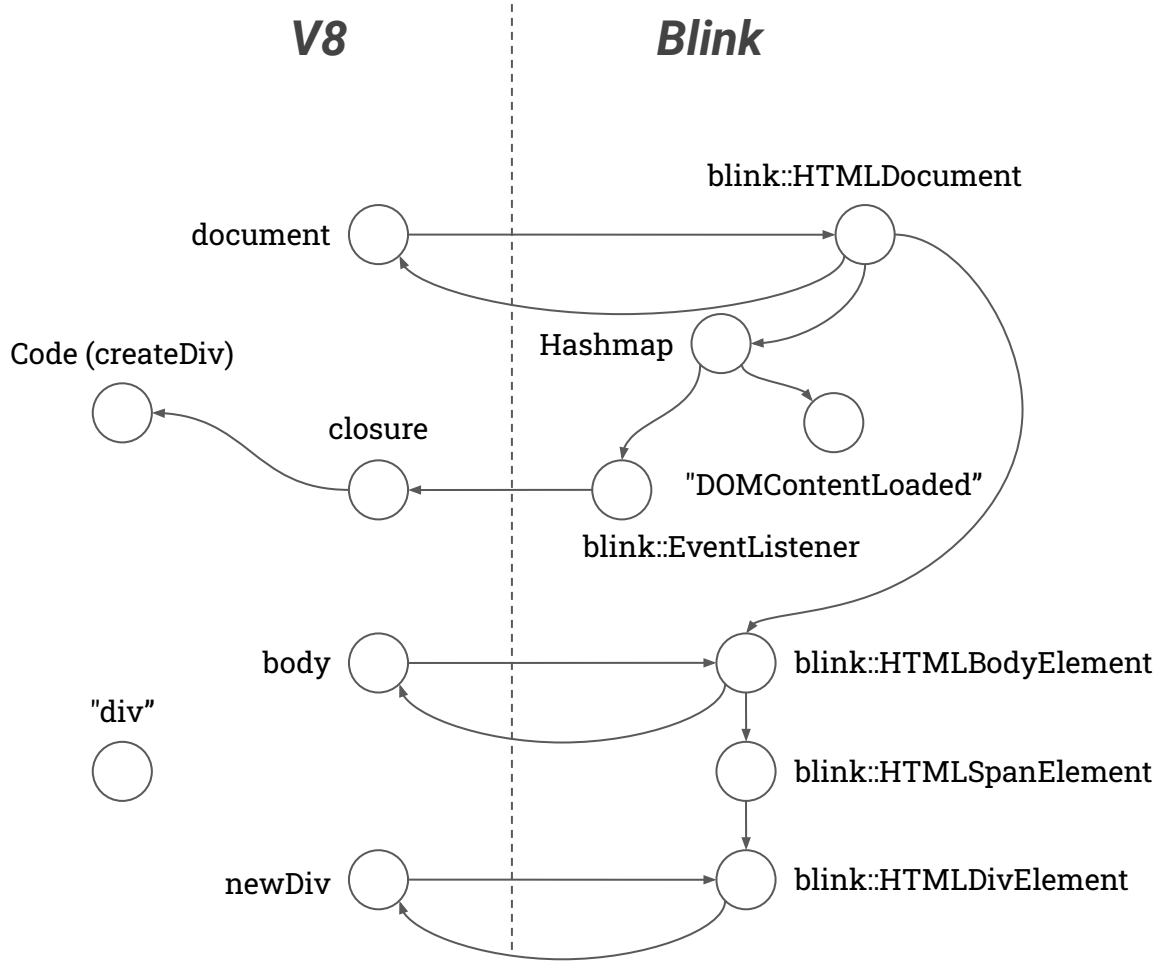
Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
    document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



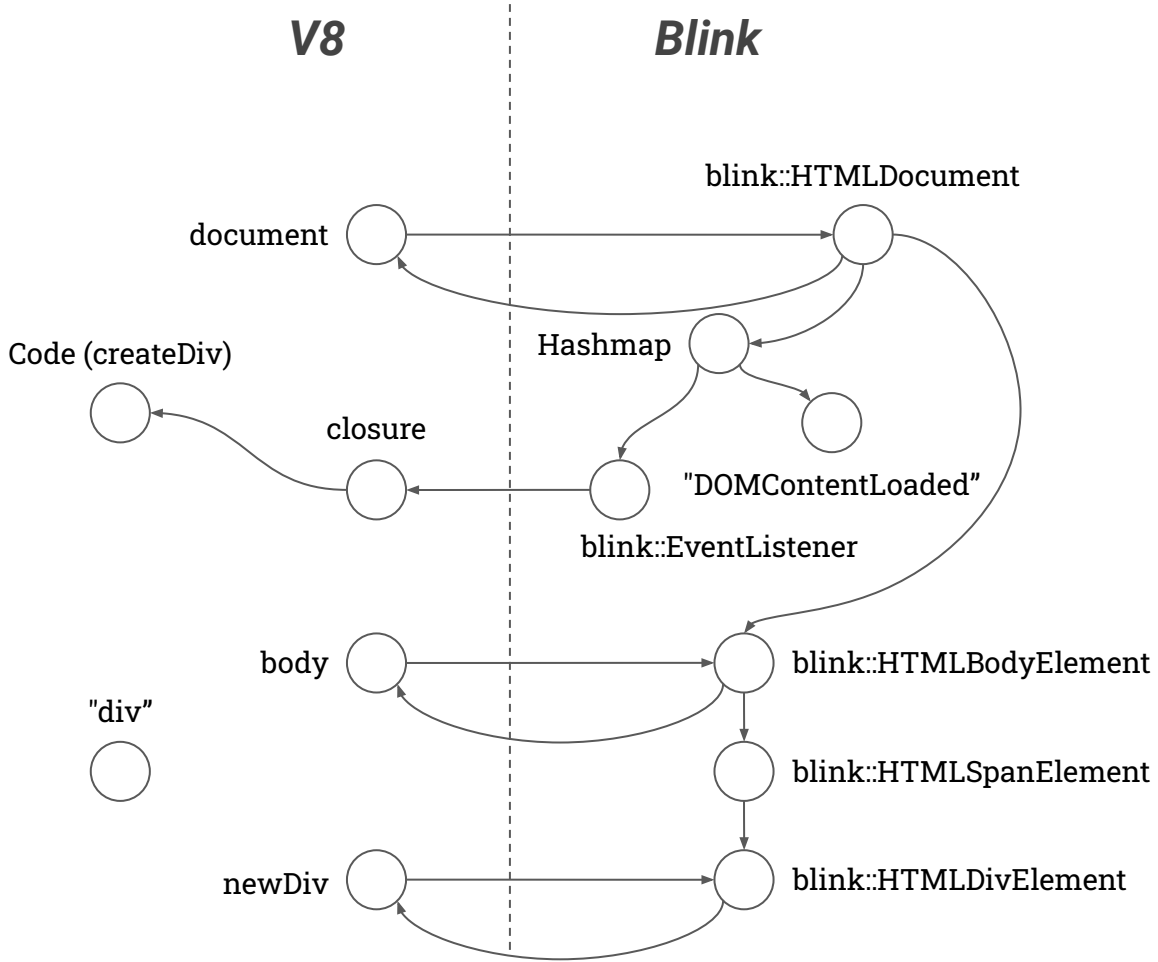
Example

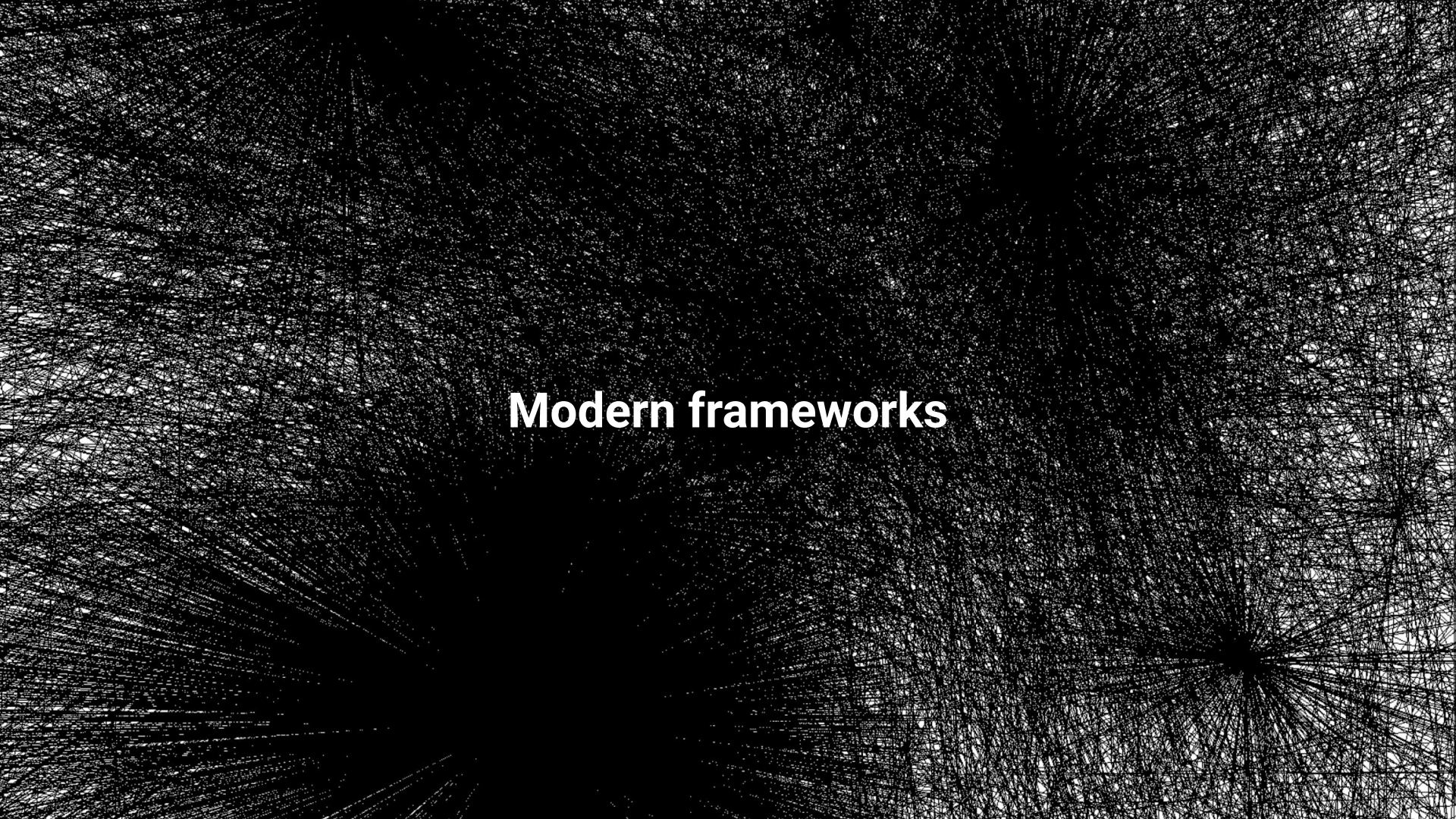
```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
     document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```



Example

```
<!DOCTYPE html>
<head><script>
  function createDiv() {
    let newDiv =
        document.createElement("div");
    document.body
        .appendChild(newDiv);
  }

  document.addEventListener(
      "DOMContentLoaded", createDiv);
</script></head>
<body>
  <span></span>
</body>
</html>
```




Modern frameworks

Cross-component garbage collection (CC GC)

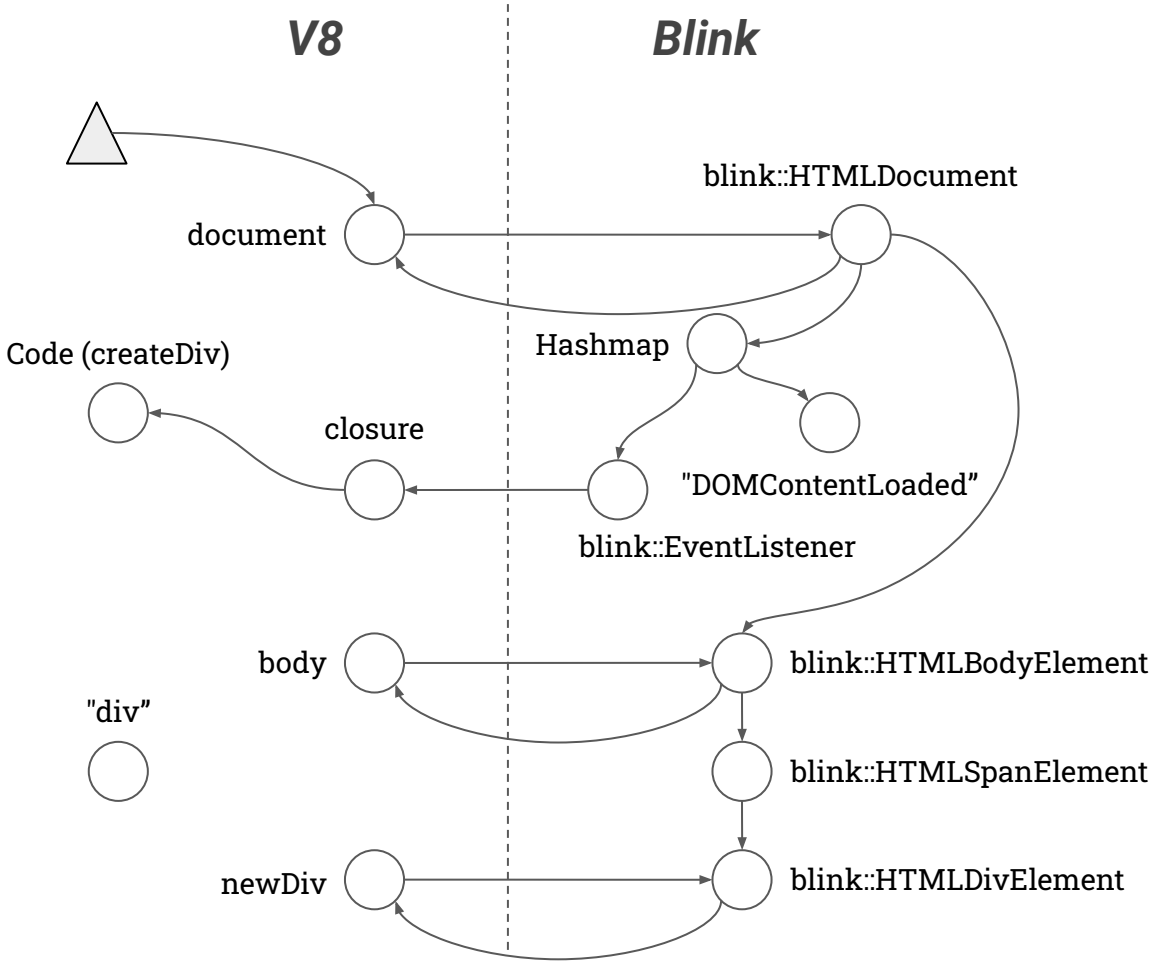
- OOPSLA'18
- Tracing garbage collectors in V8 and Blink for JS and C++, respectively
- Renderer garbage collections
  - Start at C++ and JS root sets
  - At component boundary: Delegate processing of object to specialized GC
- Allows specialized GCs to have different optimizations as long as they use tracing and rely on same invariants
  - Dijkstra insertion barrier

Cross-Component Garbage Collection

ULAN DEGENBAEV, Google, Germany
JOCHEN EISINGER, Google, Germany
KENTARO HARA, Google, Japan
MARCEL HLOPKO, Google, Germany
MICHAEL LIPPAUTZ, Google, Germany
HANNES PAYER, Google, Germany

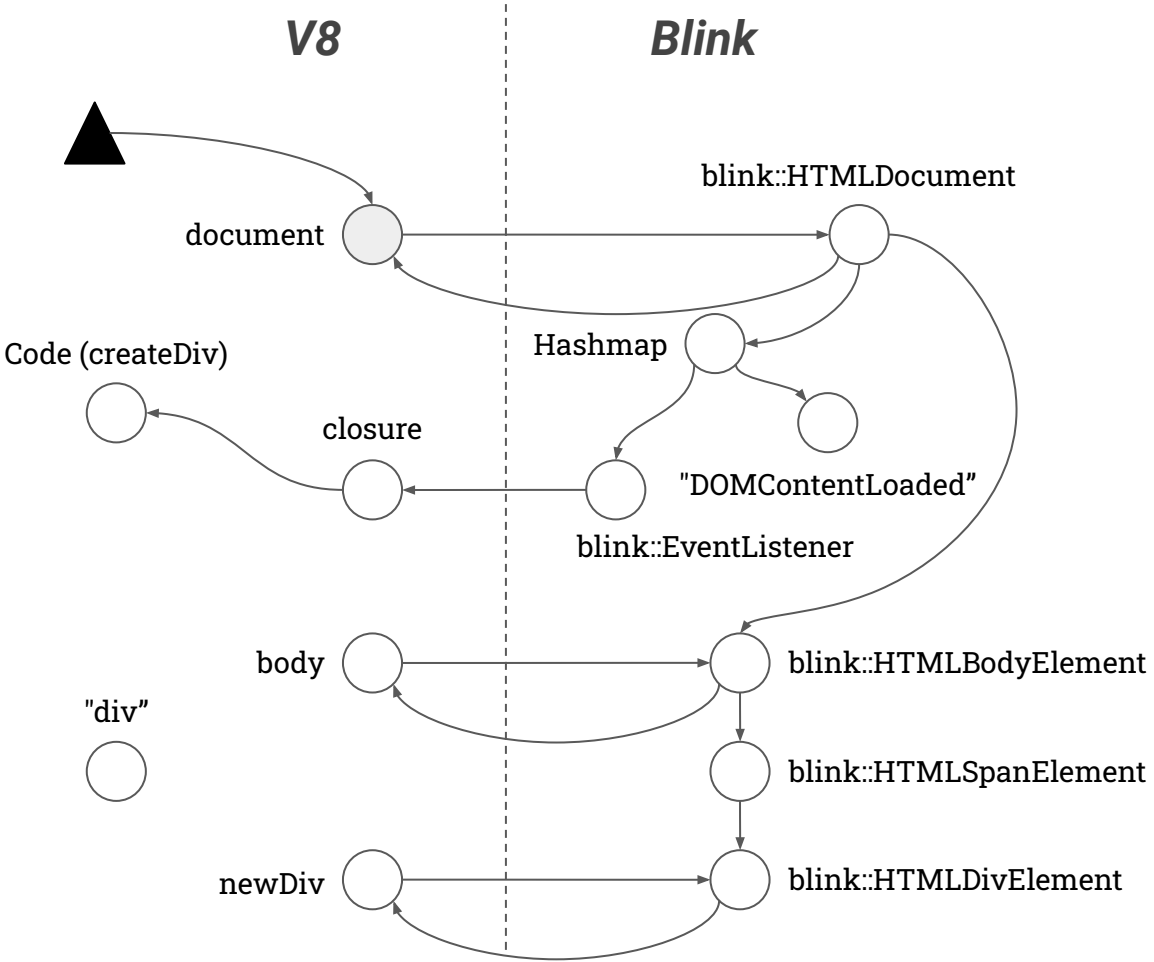
CC GC

- Start at roots



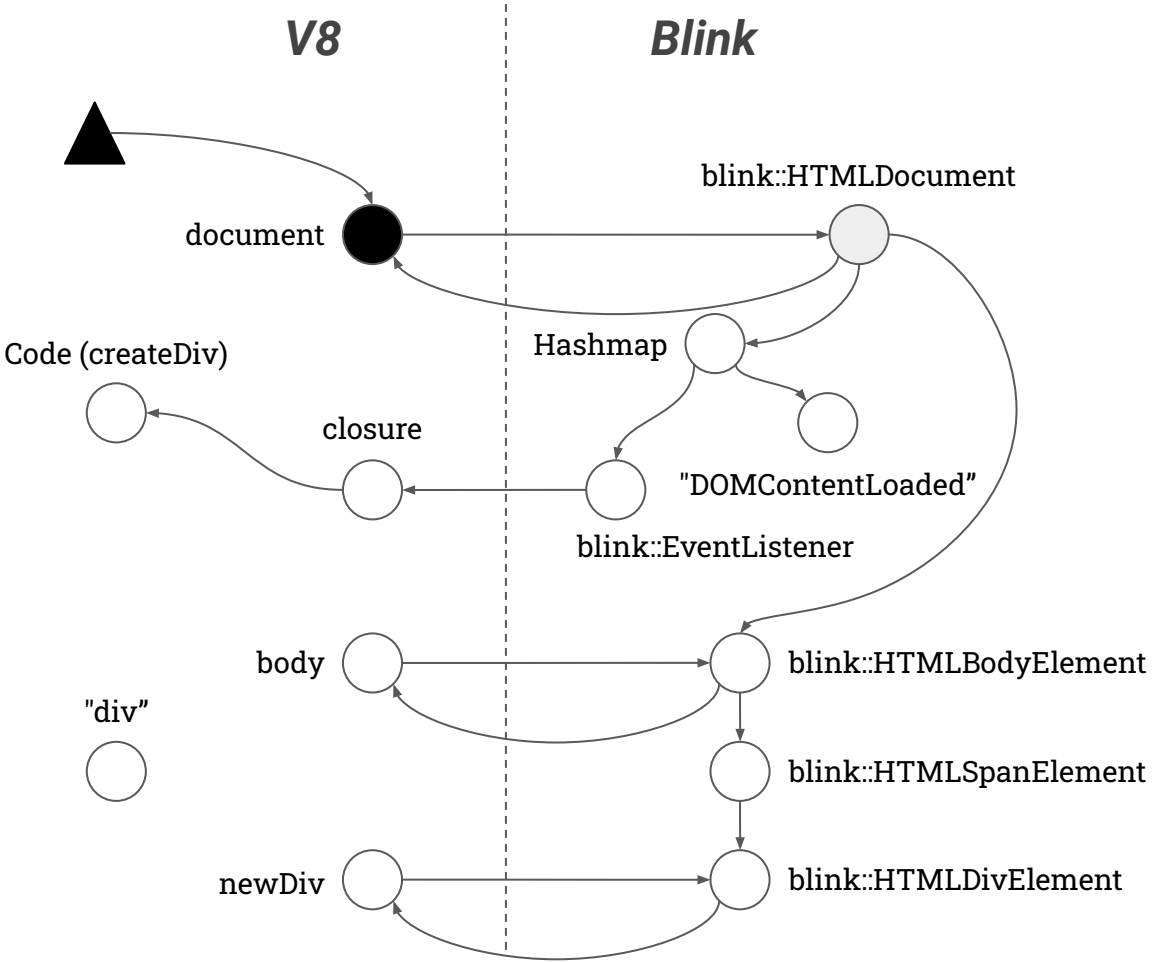
CC GC

- Start at roots
- Delegate processing to specialized GC



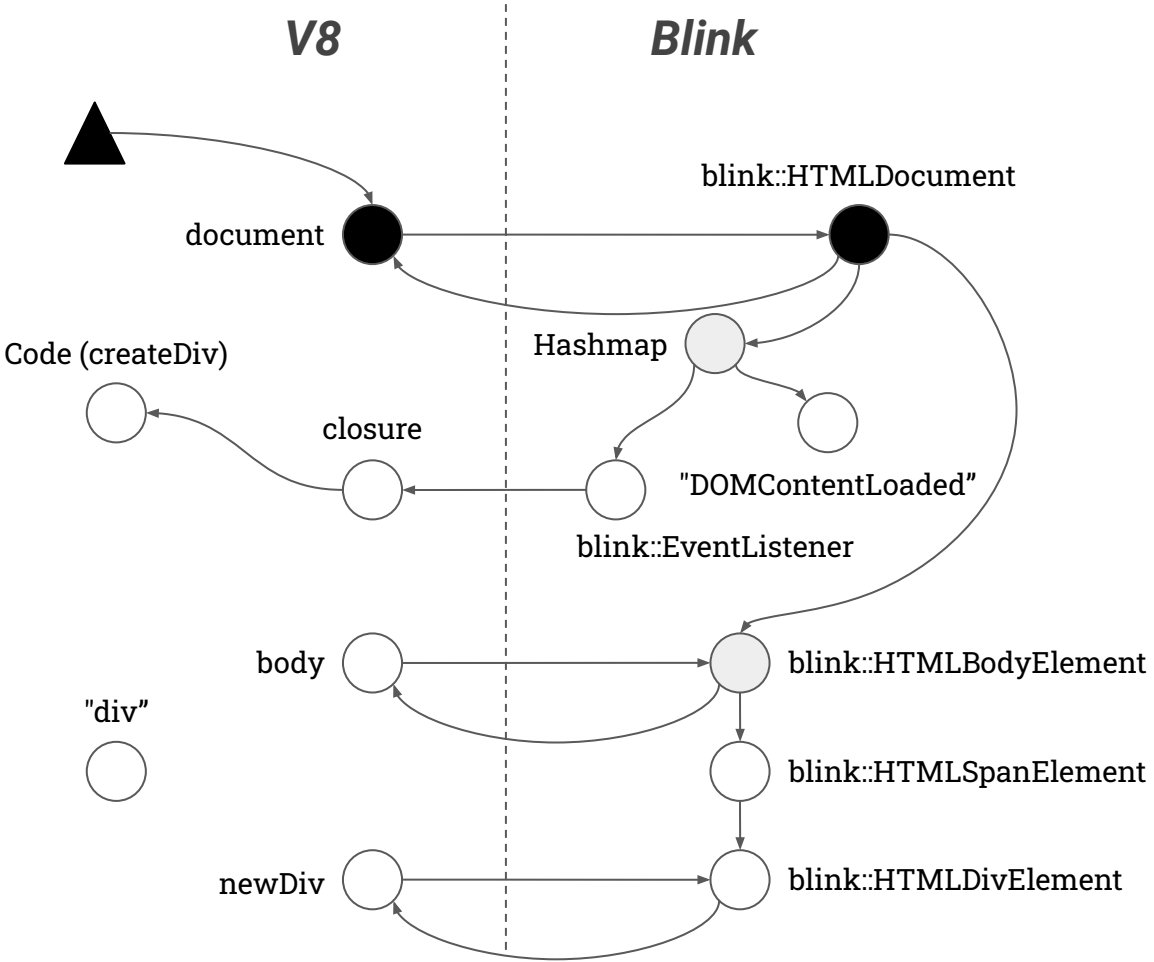
CC GC

- Start at roots
- Delegate processing to specialized GC
- Continue in components where there's work



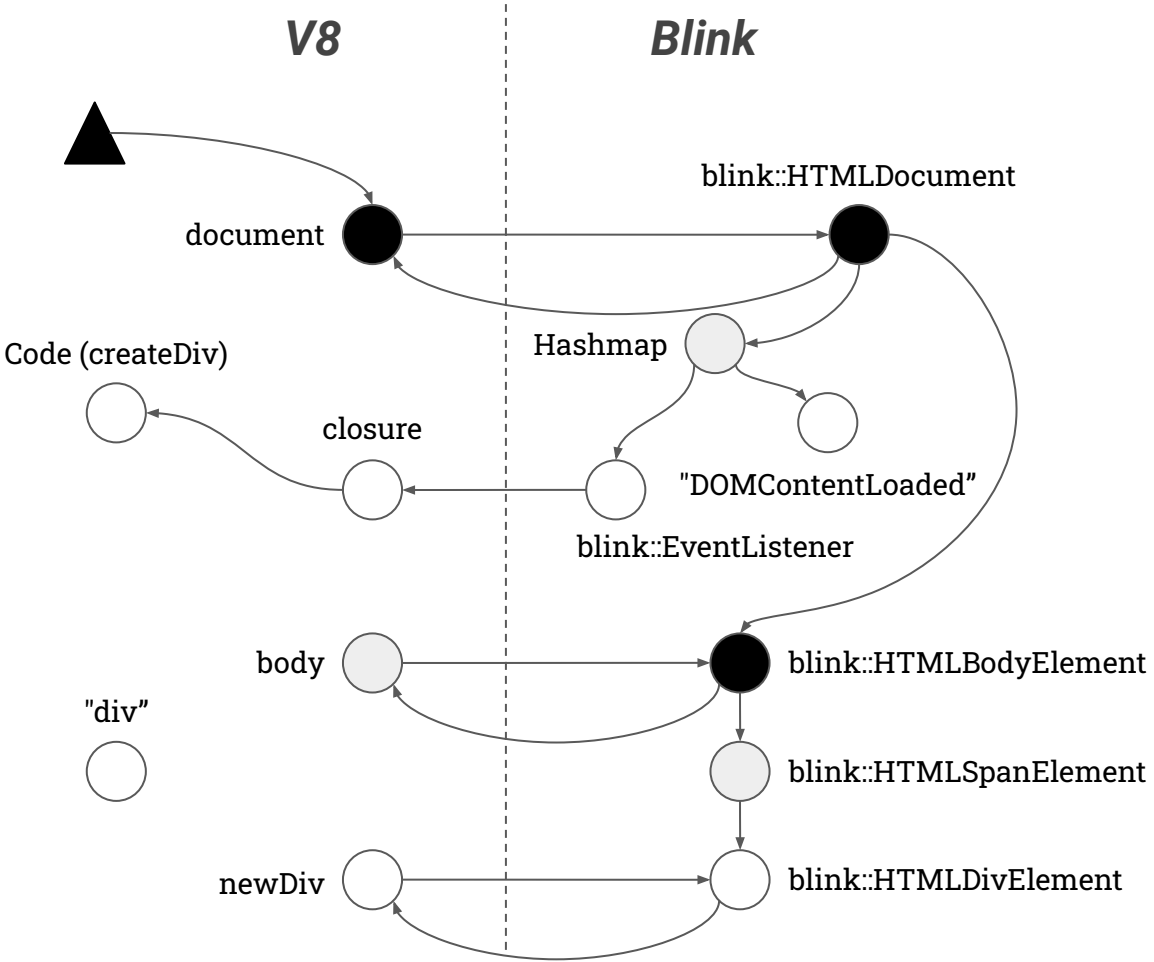
CC GC

- Start at roots
- Delegate processing to specialized GC
- Continue in components where there's work



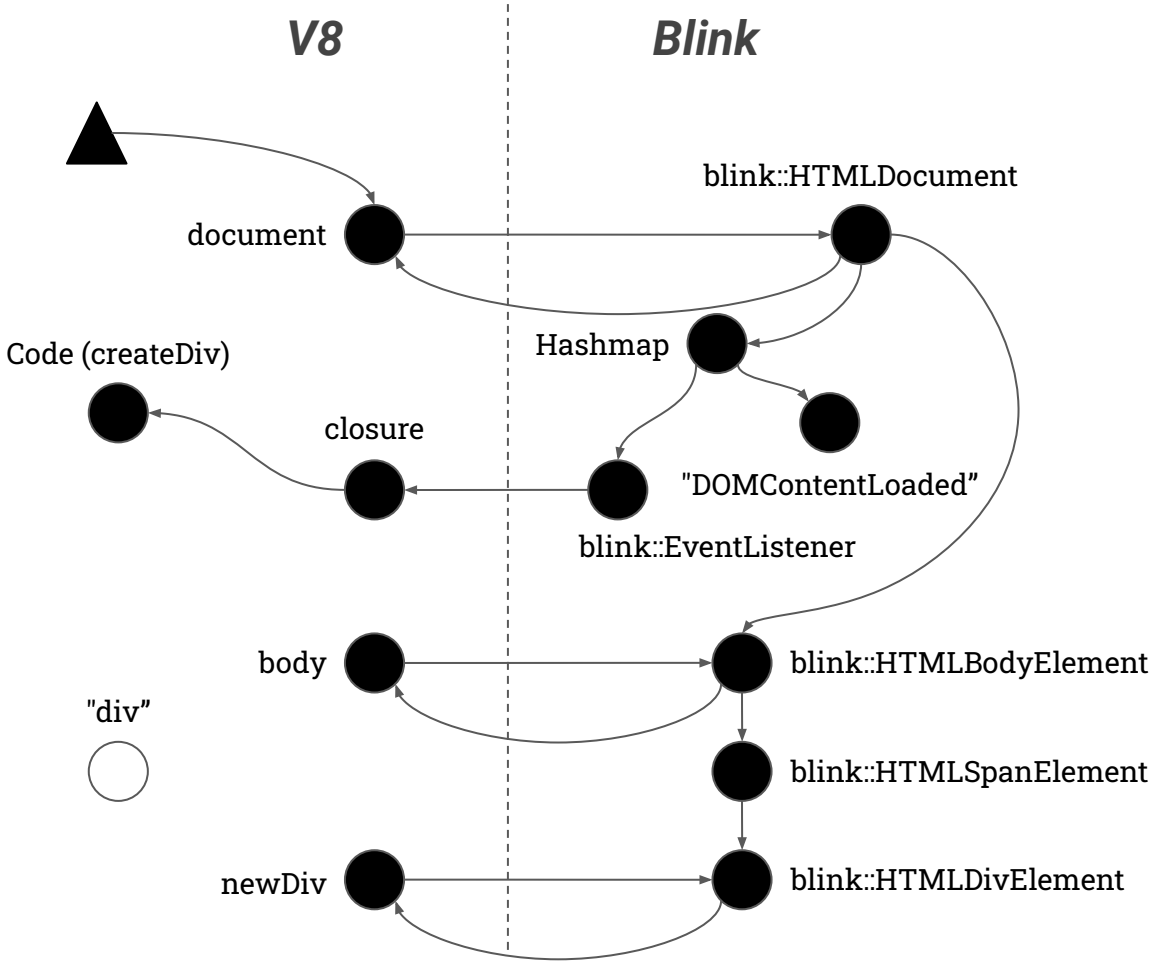
CC GC

- Start at roots
- Delegate processing to specialized GC
- Continue in components where there's work



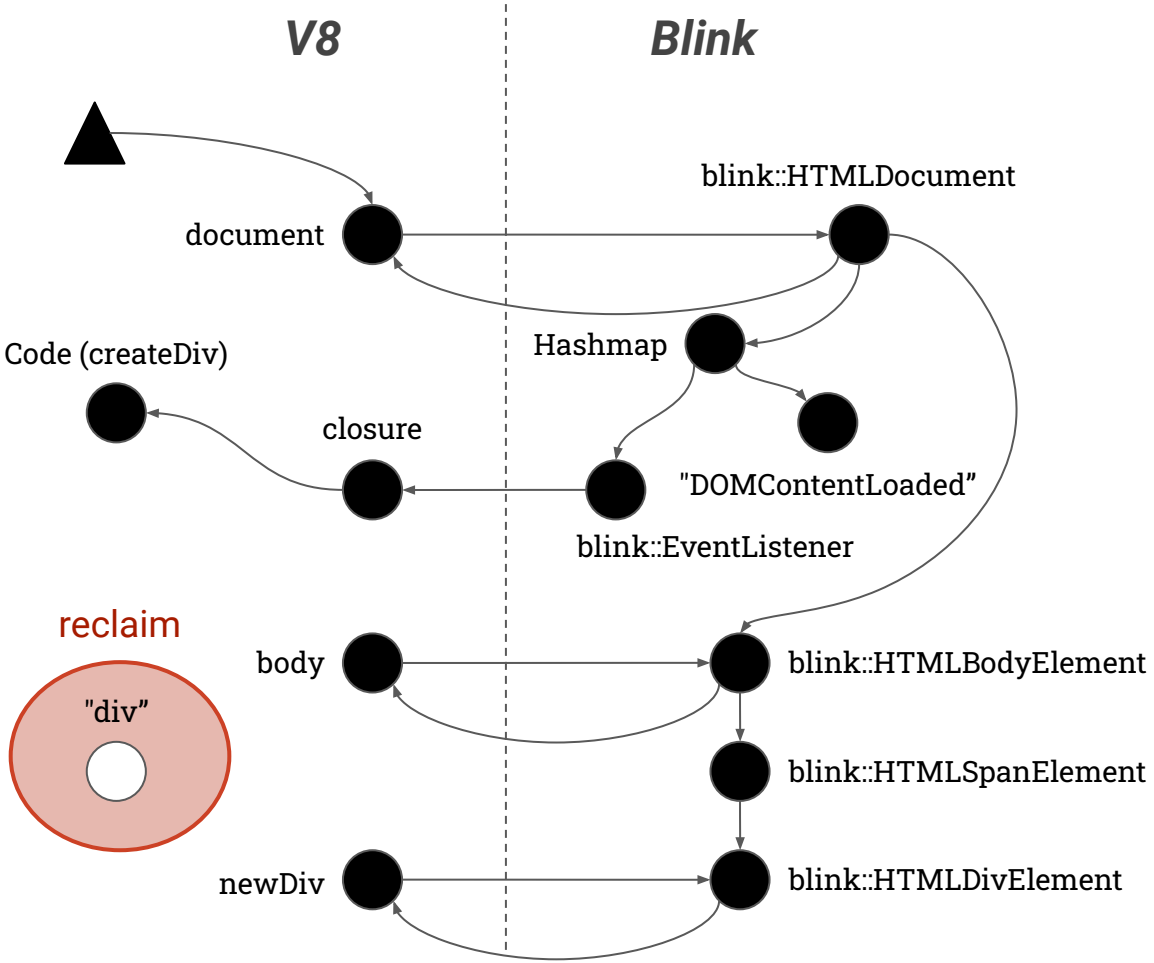
CC GC

- Start at roots
- Delegate processing to specialized GC
- Continue in components where there's work
- Fixed-point over "special objects", e.g. ephemerons



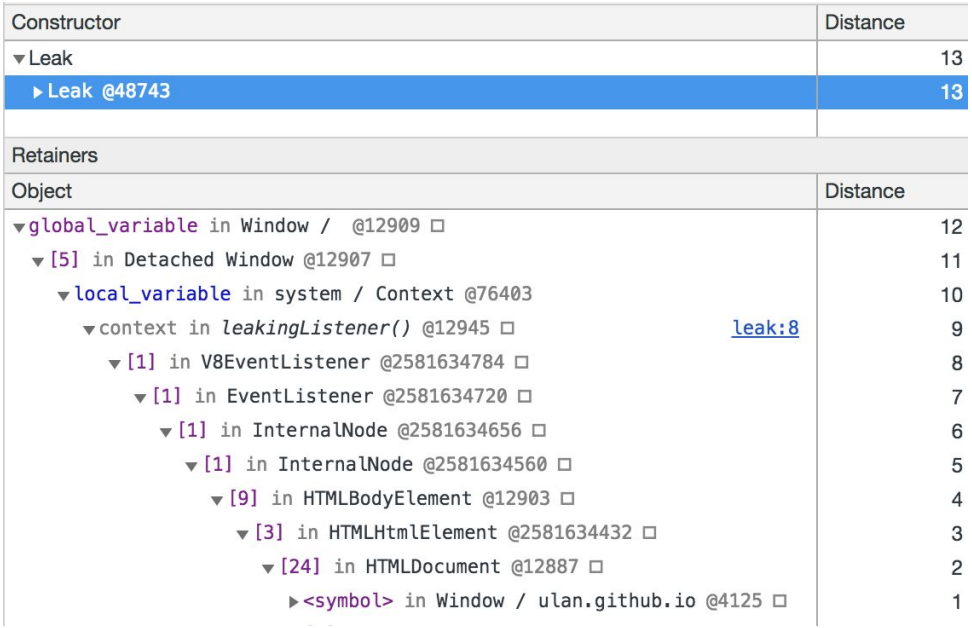
CC GC

- Start at roots
- Delegate processing to specialized GC
- Continue in components where there's work
- Fixed-point over "special objects", e.g. ephemerons
- Delegate reclamation to specialized GC



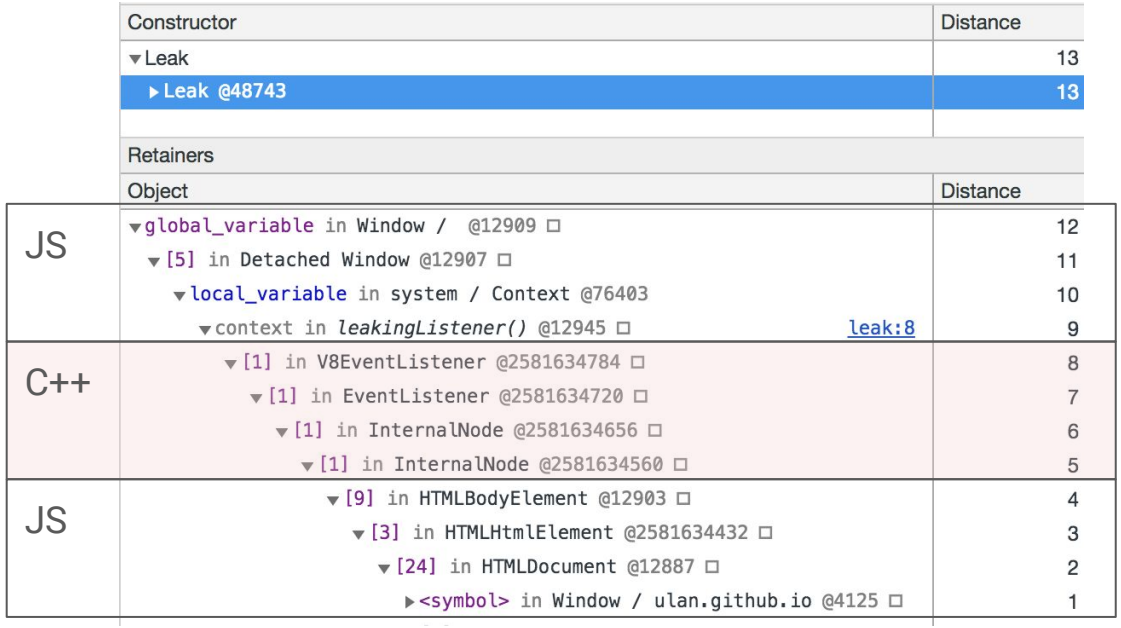
Tooling

- Basic capabilities in both worlds:
  - Objects
    - Identity
    - Naming
  - Edges



Tooling

- Basic capabilities in both worlds:
  - Objects
    - Identity
    - Naming
  - Edges



Garbage collection in C++

```
class Foo : public GarbageCollected<Foo> {
 public:
  void Trace(Visitor* v) const override {
    v->Trace(ref_);
    v->Trace(v8_ref_);
  }

 private:
  Member<Foo> ref_;
  v8::TracedReference<v8::Value> v8_ref_;
};
```

Garbage collection in C++

```
class Foo : public GarbageCollected<Foo> {
 public:
  void Trace(Visitor* v) const override {
    v->Trace(ref_);
    v->Trace(v8_ref_);
  }

 private:
  Member<Foo> ref_;
  v8::TracedReference<v8::Value> v8_ref_;
};
```

Object description exposed to user code

Garbage collection in C++

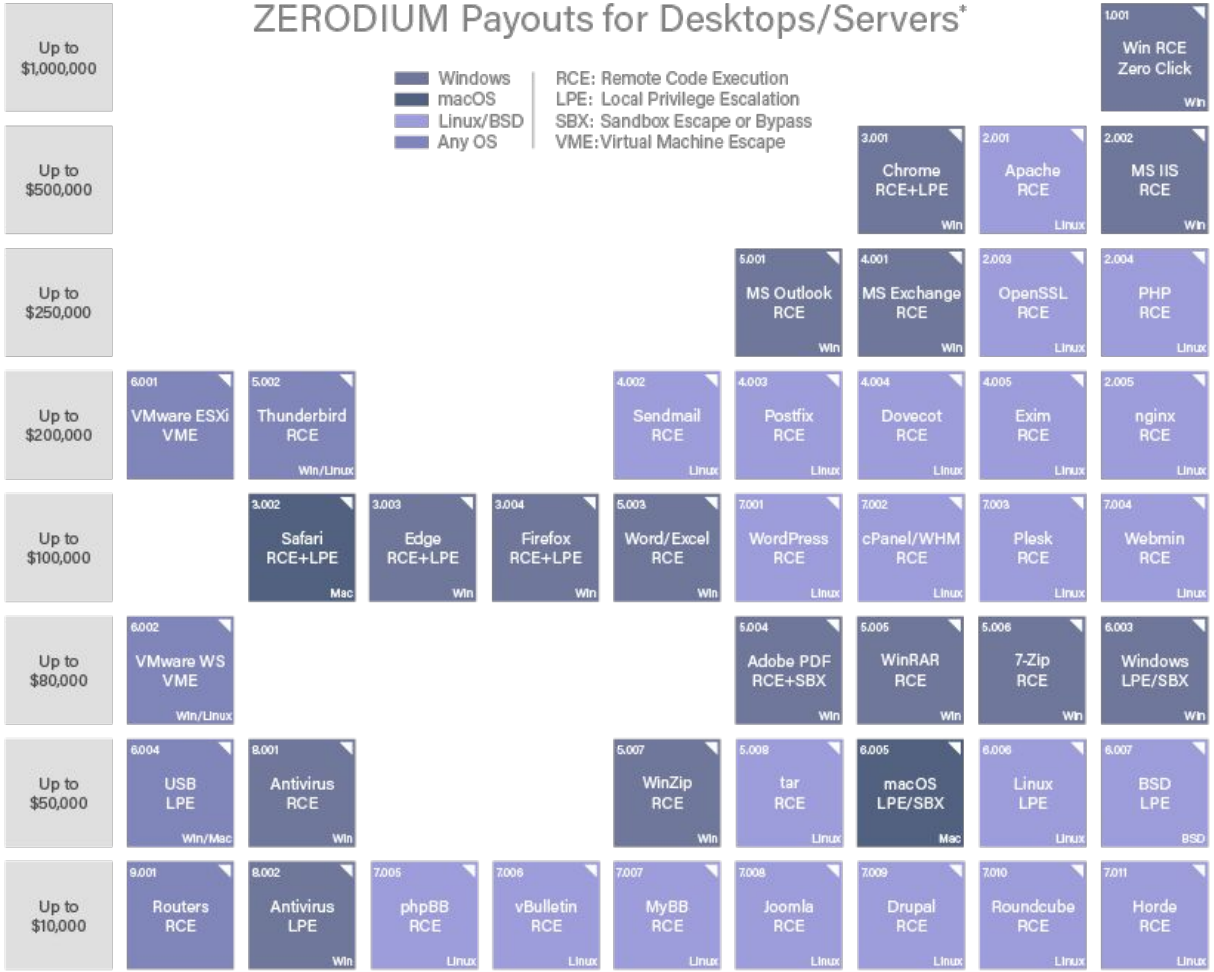
```
Foo() : ref_(PublishObject(this)) {}
```

Partially initialized object may escape to GC

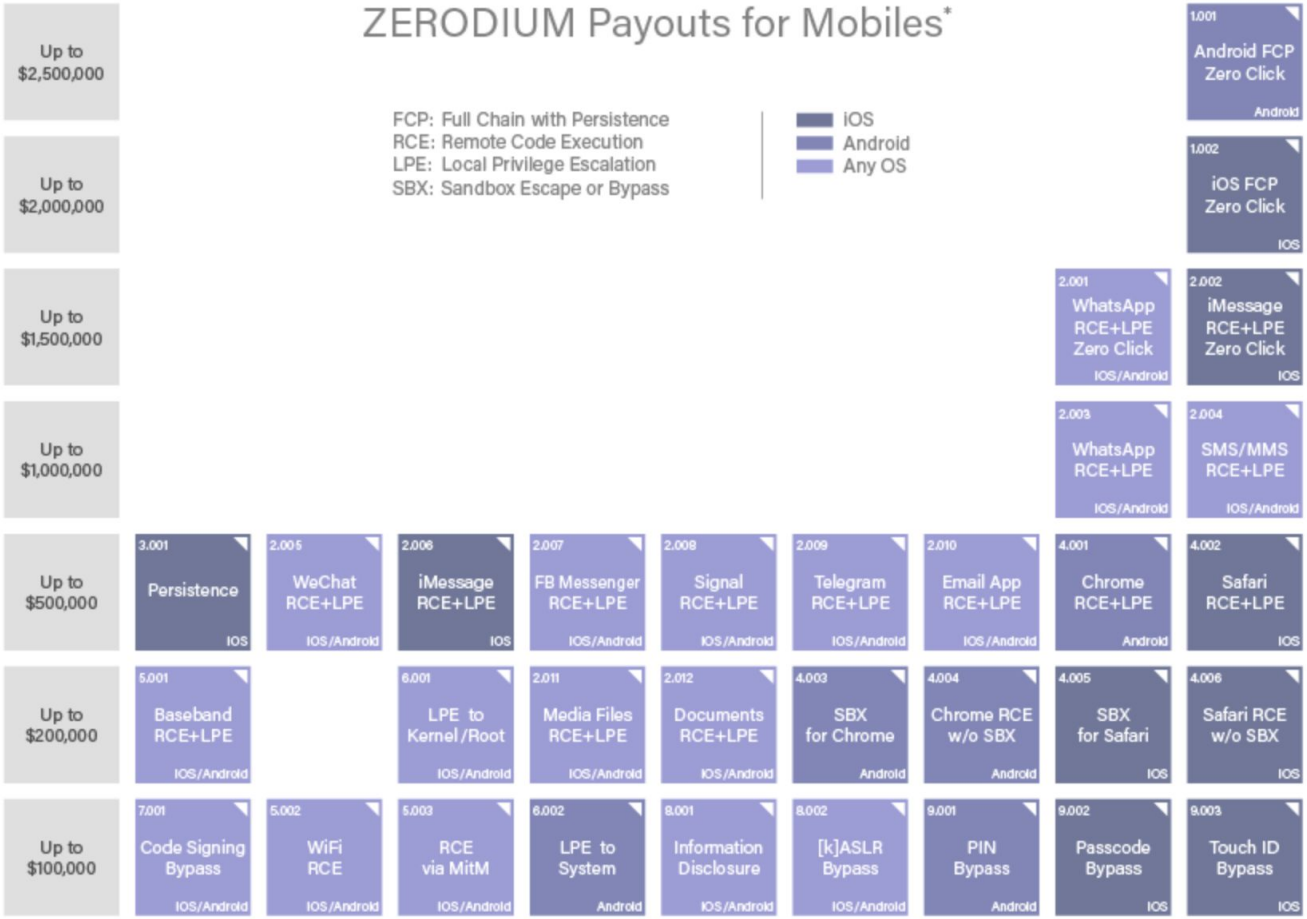
Security in V8/Chrome

	High-quality report with functional exploit	High-quality report	Baseline
Sandbox escape / Memory corruption in a non-sandboxed process	$40,000 [1]	$30,000 [1]	Up to $20,000 [1]
Universal Cross Site Scripting (includes Site Isolation bypass)	$20,000	$15,000	Up to $10,000
Memory Corruption in a highly privileged process (e.g. GPU or network processes)	$20,000	$15,000	Up to $10,000
Renderer RCE / memory corruption in a sandboxed process	$15,000	$10,000	Up to $7,000
Security UI Spoofing	$7,500	N/A [2]	Up to $3,000
User information disclosure	$5,000 - $20,000	N/A [2]	Up to $2,000
Web Platform Privilege Escalation	$5,000	$3,000	Up to $1,000
Exploitation Mitigation Bypass	$5,000	$3,000	Up to $1,000
Chrome OS	See the 'Chrome OS' section below		
Chrome Bisect Bonus	$500-$2,000 (see the 'Bisect Bonus' section below)		
Chrome Fuzzer Bonus	Up to $5,000 (see the 'Chrome Fuzzer Program' section below)		
Chrome Patch Bonus	$500 - $2,000		

ZERODIUM Payouts for Desktops/Servers*



** All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.*



** All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.*

Security in V8/Chrome

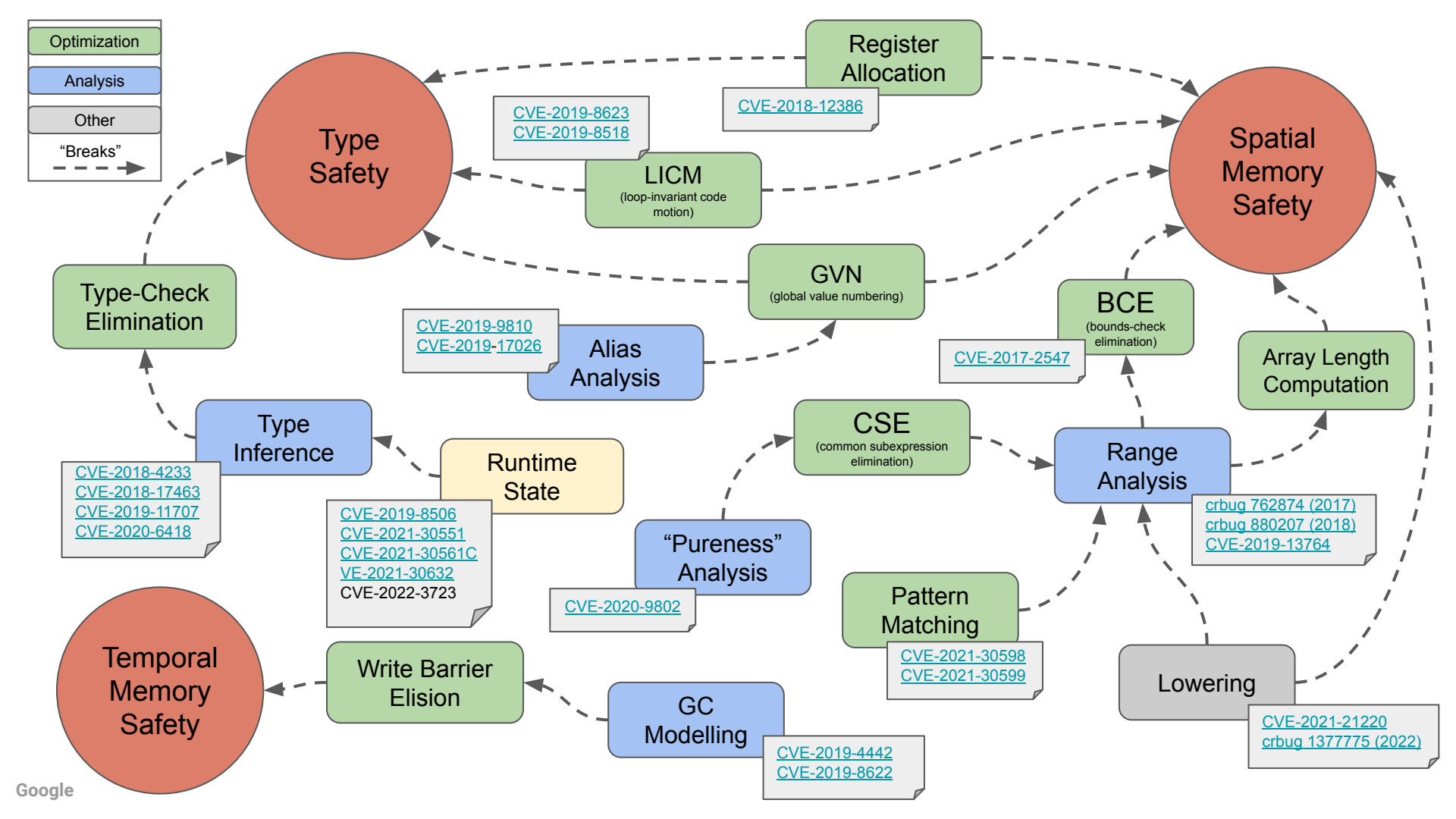
Defense in depth (layer on layer on layer on layer on…)

CI

- Sanitizers
- Testing/fuzzing

Architecture

- Chrome sandbox
- Process segregation
- Hardware features: MTE, PAC, CFI, ...
- Garbage collection
- Bindings (WebIDL)
- Runtime assertions (CHECK())
- Memory quarantines
- ...

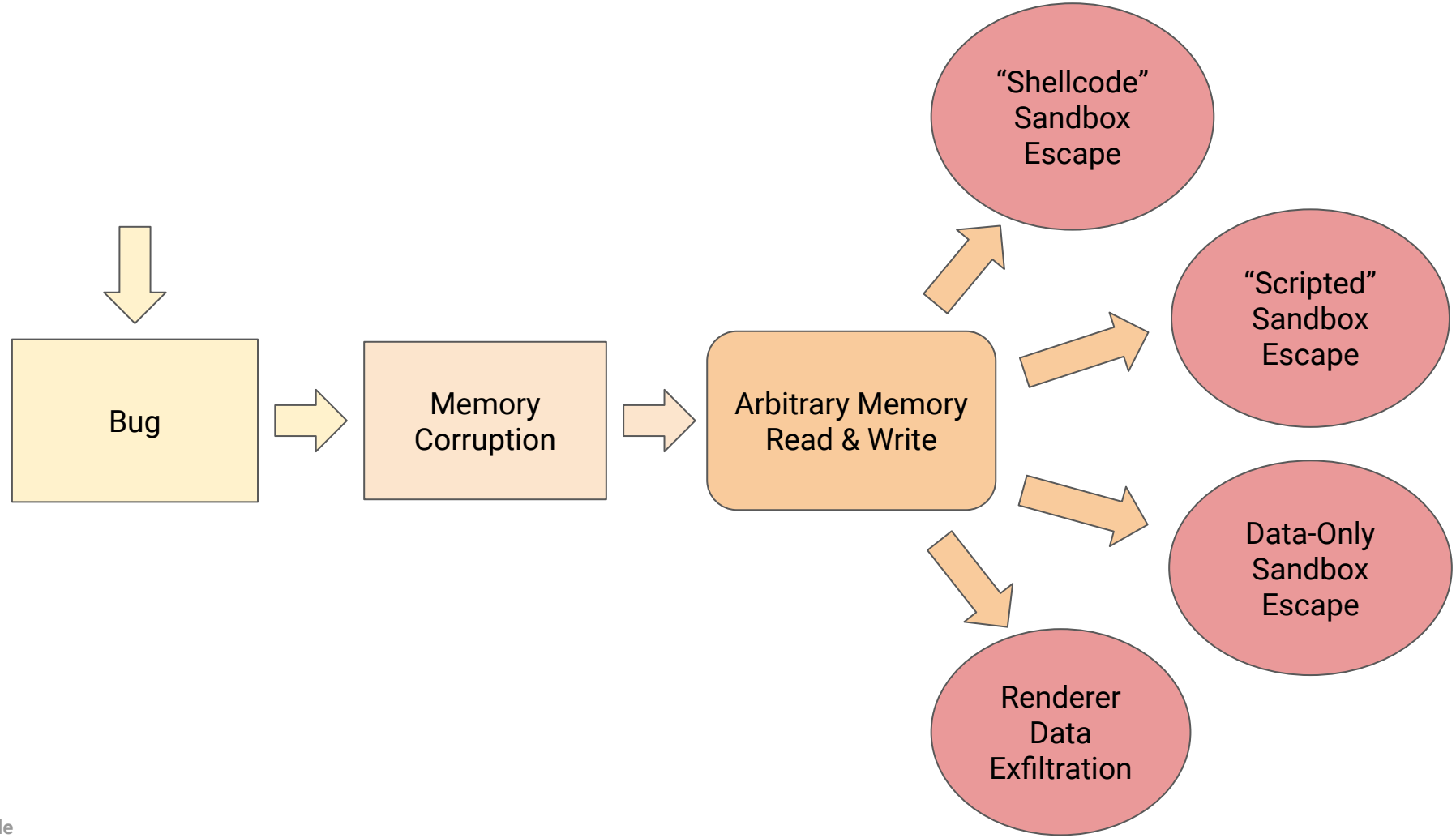

Optimization
Analysis
Other
“Breaks”
Type Safety
Spatial Memory Safety
Temporal Memory Safety
Register Allocation
CVE-2018-12386
CVE-2019-8623
CVE-2019-8518
LICM
(loop-invariant code motion)
GVN
(global value numbering)
Type-Check Elimination
CVE-2019-9810
CVE-2019-17026
Alias Analysis
BCE
(bounds-check elimination)
CVE-2017-2547
Array Length Computation
Type Inference
CVE-2018-4233
CVE-2018-17463
CVE-2019-11707
CVE-2020-6418
Runtime State
CVE-2019-8506
CVE-2021-30551
CVE-2021-30561C
VE-2021-30632
CVE-2022-3723
CSE
(common subexpression elimination)
Range Analysis
crbug 762874 (2017)
crbug 880207 (2018)
CVE-2019-13764
“Pureness” Analysis
CVE-2020-9802
Pattern Matching
CVE-2021-30598
CVE-2021-30599
Lowering
CVE-2021-21220
crbug 1377775 (2022)
Write Barrier Elision
GC Modelling
CVE-2019-4442
CVE-2019-8622
Google

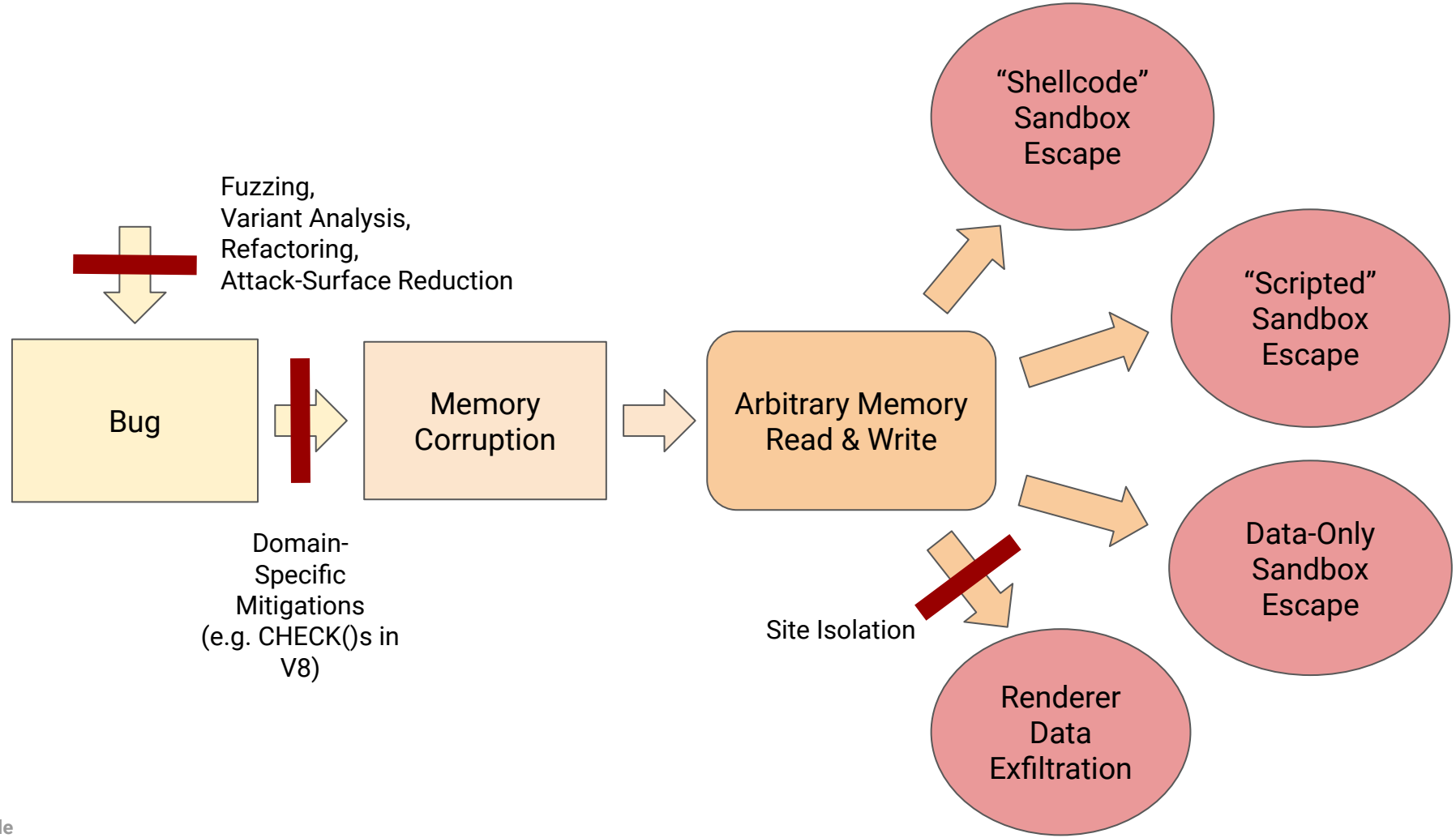
V8’s fundamental problem

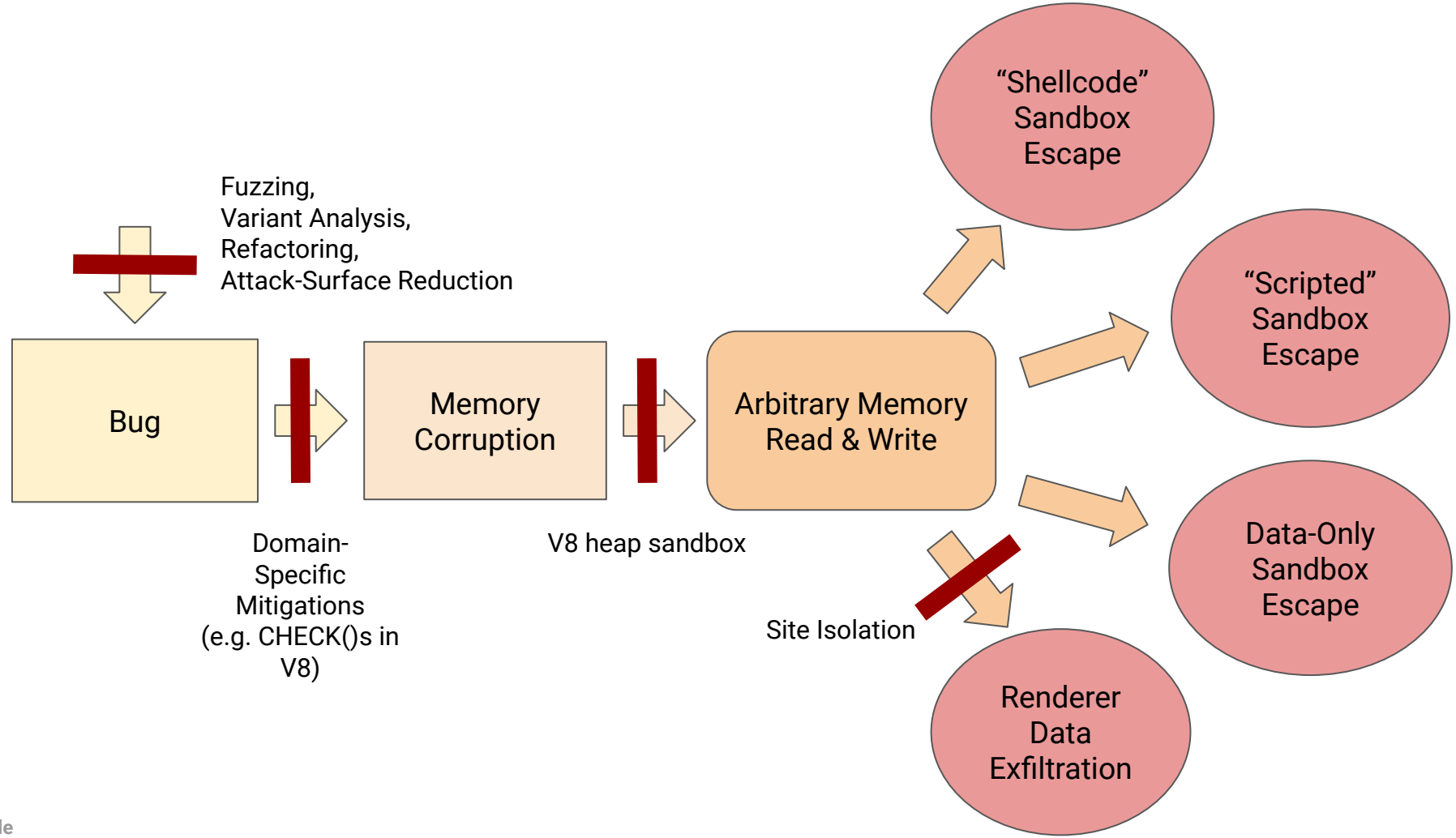
JIT bugs are essentially 2nd order vulnerabilities

- Root cause is a *logic* issue in the compiler/runtime environment
- ... which is then exploited to generate vulnerable machine code
- ... which can then be exploited for arbitrary memory corruption at runtime

Often cannot even be mitigated with latest hardware features (e.g. CFI).


Bug
Memory Corruption
Arbitrary Memory Read & Write
“Shellcode” Sandbox Escape
“Scripted” Sandbox Escape
Data-Only Sandbox Escape
Renderer Data Exfiltration


Fuzzing,
Variant Analysis,
Refactoring,
Attack-Surface Reduction
Bug
Memory Corruption
Domain-Specific Mitigations (e.g. CHECK()s in V8)
Arbitrary Memory Read & Write
Site Isolation
"Shellcode" Sandbox Escape
"Scripted" Sandbox Escape
Data-Only Sandbox Escape
Renderer Data Exfiltration


Fuzzing,
Variant Analysis,
Refactoring,
Attack-Surface Reduction
Bug
Memory Corruption
Arbitrary Memory Read & Write
"Shellcode" Sandbox Escape
"Scripted" Sandbox Escape
Data-Only Sandbox Escape
Renderer Data Exfiltration
Domain-Specific Mitigations (e.g. CHECK()s in V8)
V8 heap sandbox
Site Isolation

V8 heap sandbox

In-process sandbox to limit the impact of V8 vulnerabilities

Currently

V8 vulnerability + Chrome Sandbox escape

V8 heap sandbox

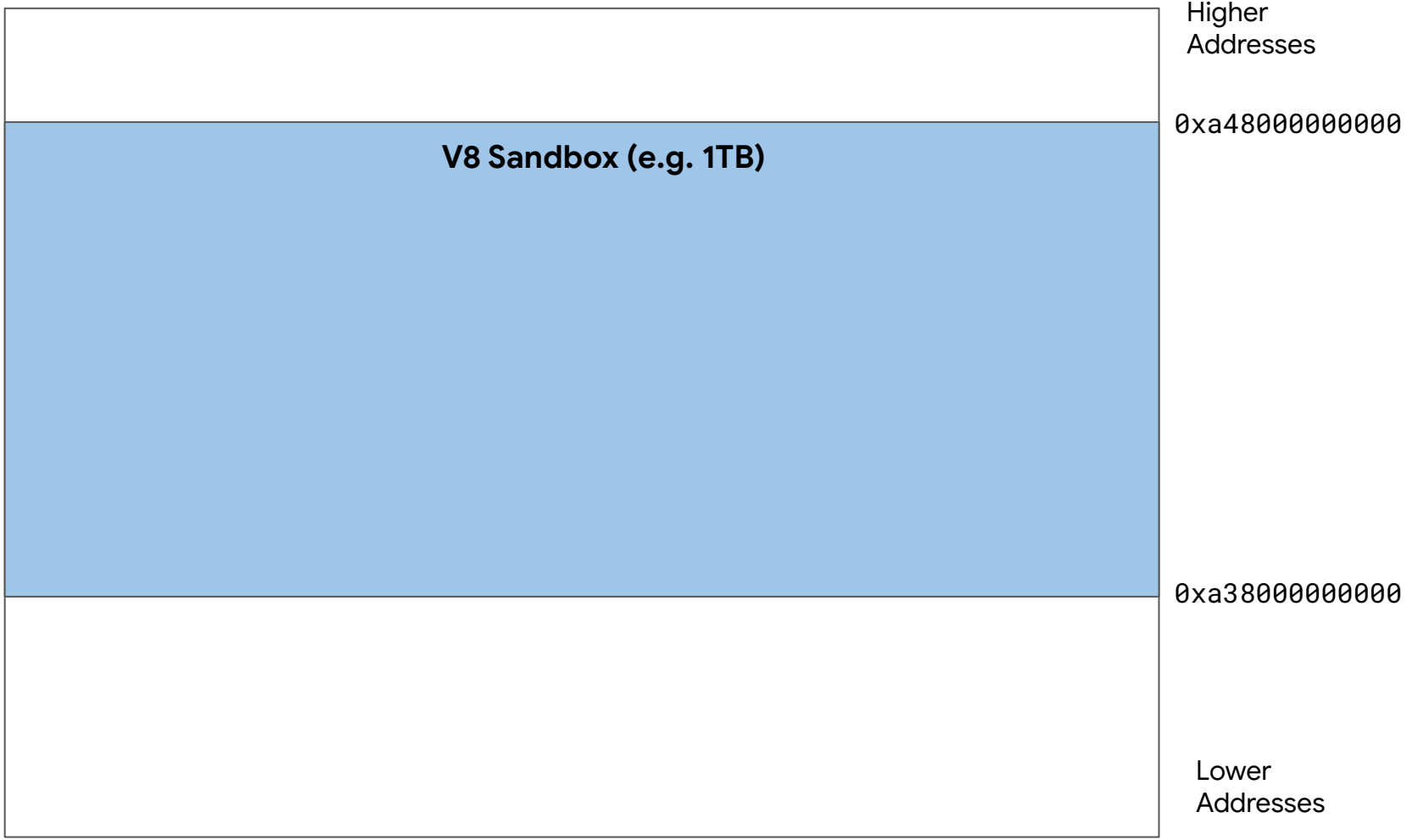
In-process sandbox to limit the impact of V8 vulnerabilities

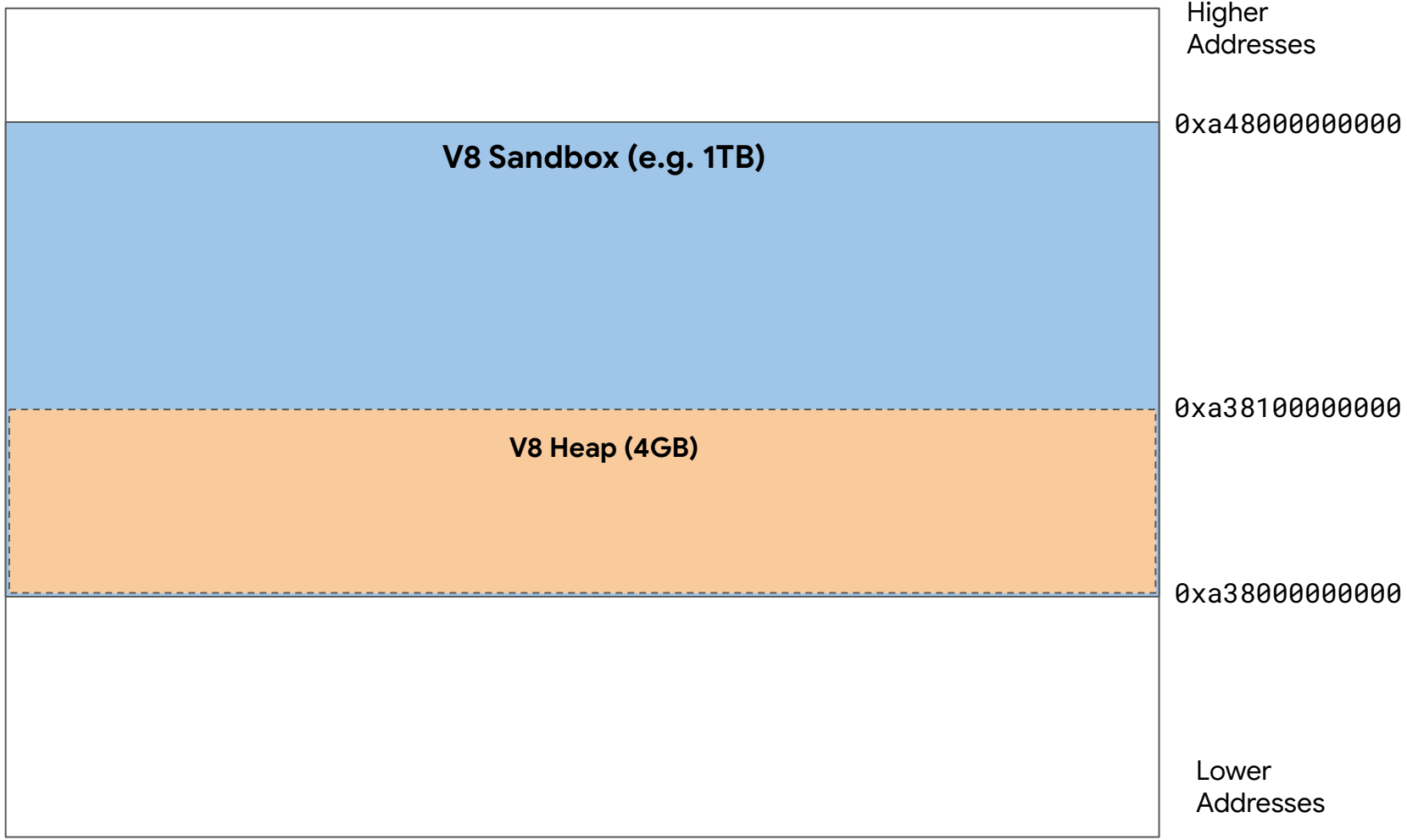
Future

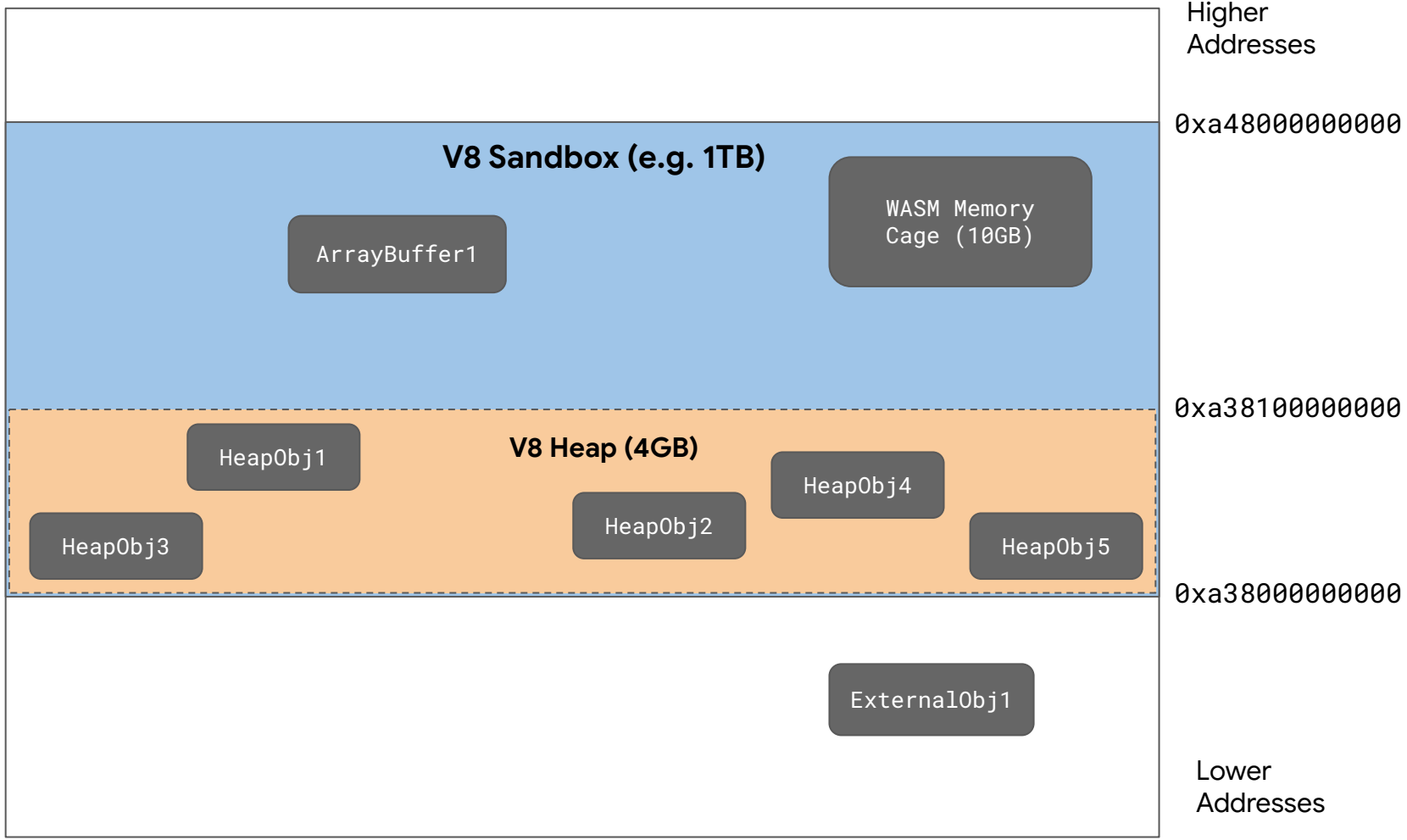
V8 vulnerability + V8 Heap Sandbox escape + Chrome Sandbox escape

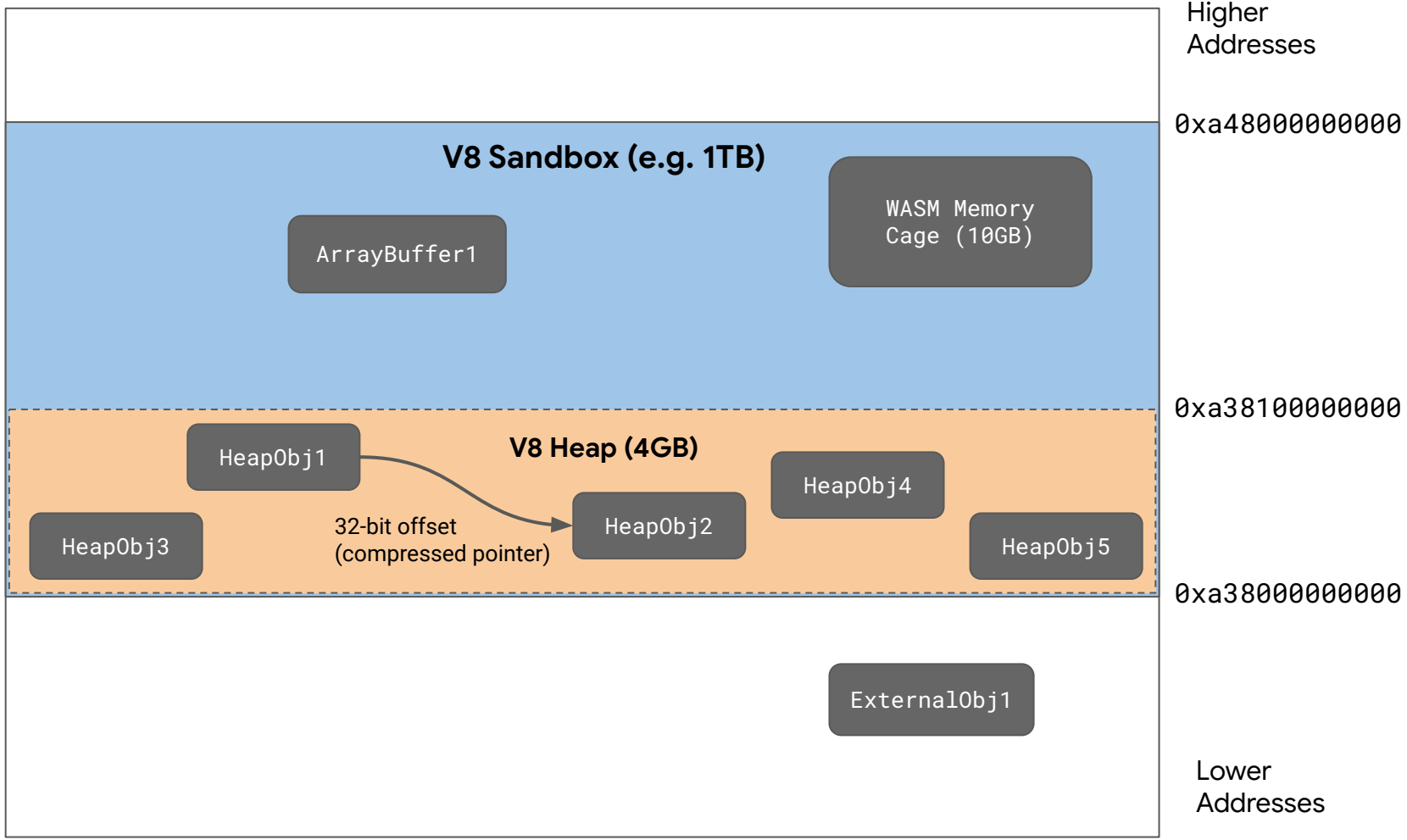
V8 heap sandbox: How?

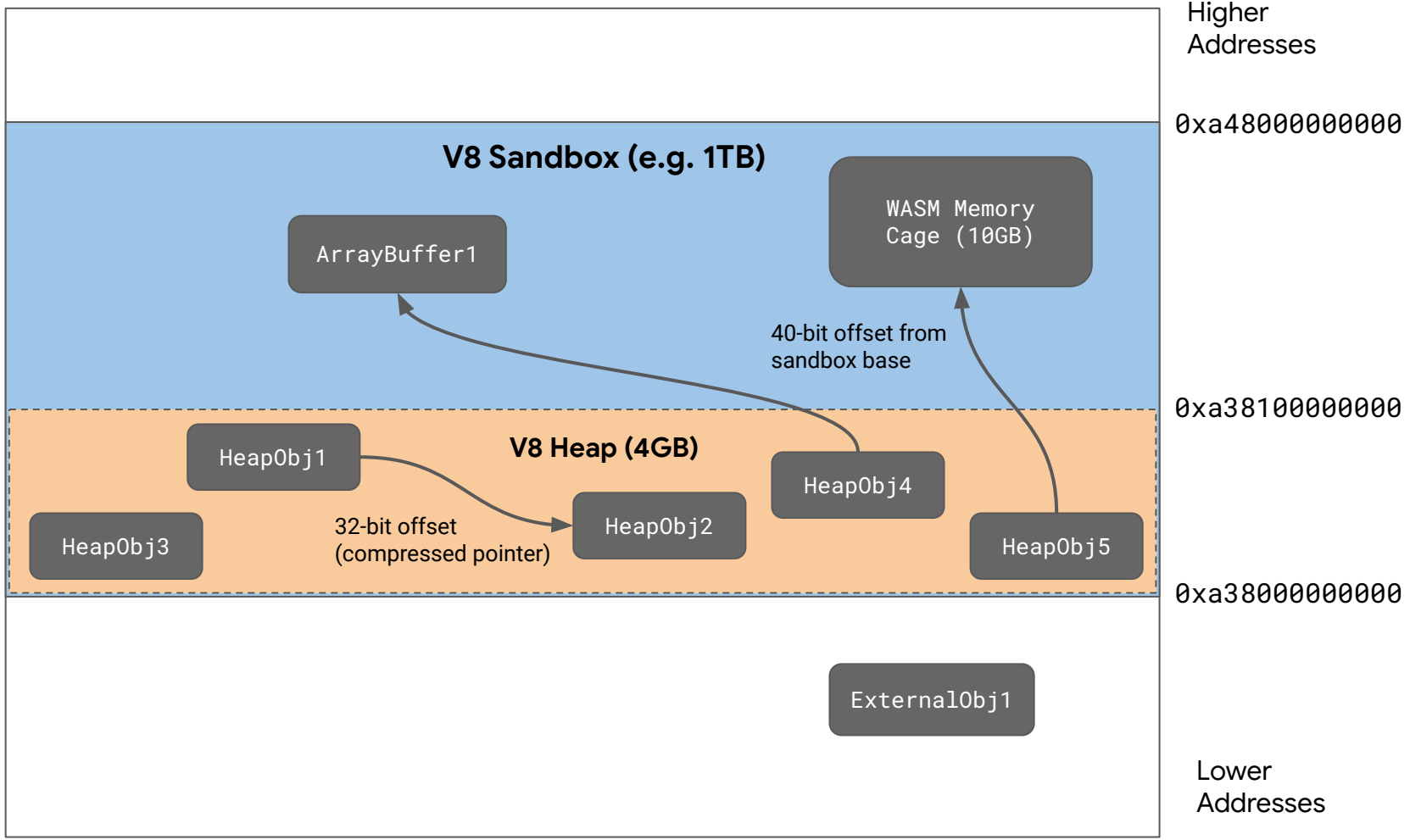
- Remove all pointers from the V8 heap
- On-heap pointers become 32-bit offsets (“compressed pointers”)
- Off-heap pointers become indices into external pointer table (“external pointers”)
- V8 exploit can then (in theory) only corrupt data inside the heap, not outside

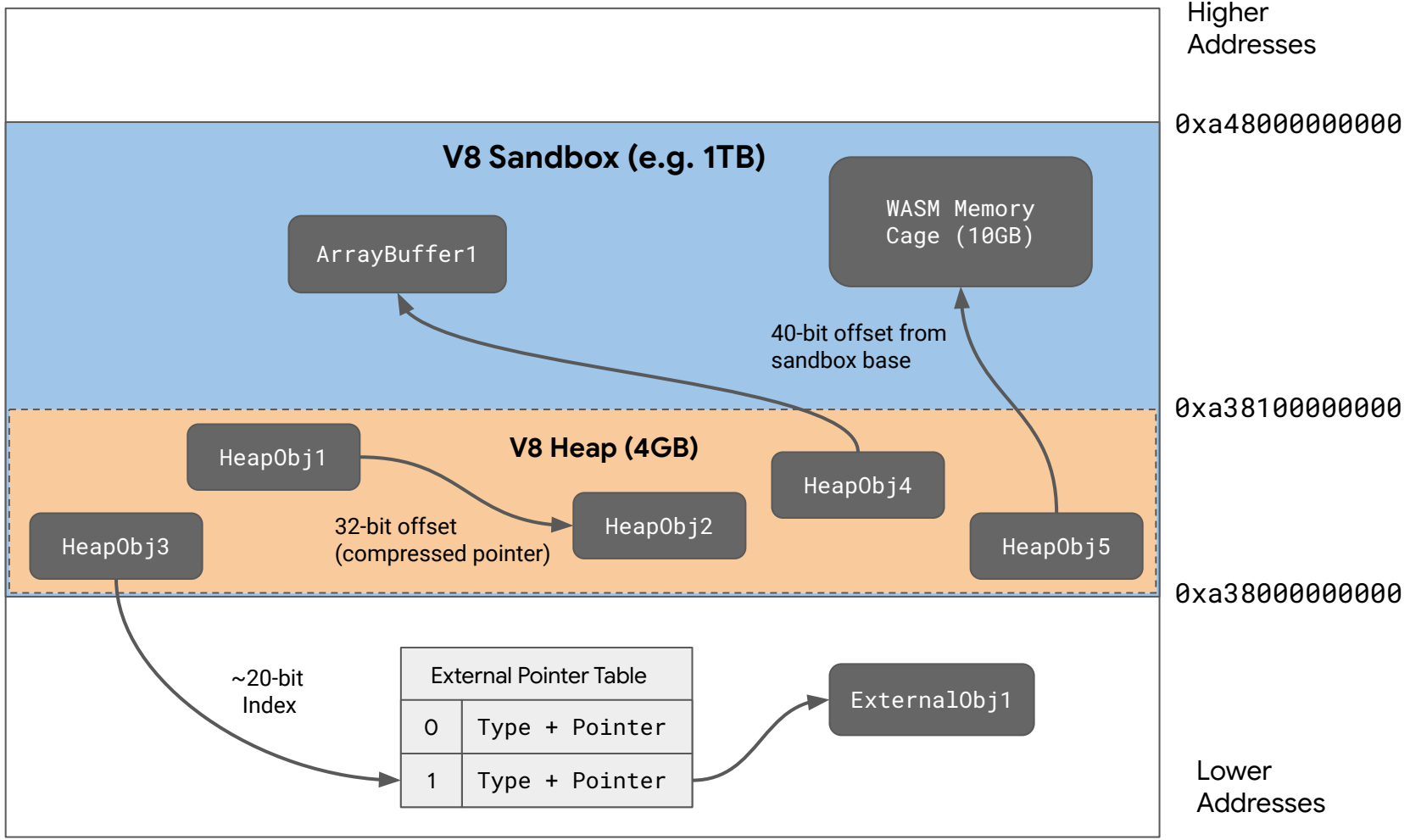

Higher
Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
0xa38000000000
Lower
Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
0xa38100000000
V8 Heap (4GB)
0xa38000000000
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
WASM Memory Cage (10GB)
ArrayBuffer1
0xa38100000000
V8 Heap (4GB)
HeapObj1
HeapObj4
HeapObj2
HeapObj3
HeapObj5
0xa38000000000
ExternalObj1
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
ArrayBuffer1
WASM Memory Cage (10GB)
0xa38100000000
V8 Heap (4GB)
HeapObj1
HeapObj4
HeapObj3
32-bit offset
(compressed pointer)
HeapObj2
HeapObj5
0xa38000000000
ExternalObj1
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
ArrayBuffer1
WASM Memory Cage (10GB)
40-bit offset from sandbox base
0xa38100000000
V8 Heap (4GB)
HeapObj1
HeapObj4
HeapObj3
32-bit offset (compressed pointer)
HeapObj2
HeapObj5
0xa38000000000
ExternalObj1
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (e.g. 1TB)
ArrayBuffer1
WASM Memory Cage (10GB)
40-bit offset from sandbox base
0xa38100000000
V8 Heap (4GB)
HeapObj1
HeapObj4
HeapObj3
32-bit offset (compressed pointer)
HeapObj2
HeapObj5
0xa38000000000
~20-bit Index
External Pointer Table
0
Type + Pointer
1
Type + Pointer
ExternalObj1
Lower Addresses

V8 heap sandbox

- Remove all pointers from the V8 heap
- Use compressed address scheme and external tables to refer to objects

Goal: Add to Vulnerability Reward Program (VRP) program

- Memory corruption API
  - Memory view over V8's heap
  - API emulating exploitation frameworks: AddressOf(), GetObjectAt()
- Modelled as game:
  - An attacker has control over the whole heap of V8
  - The game is won if the process observes a segfault outside of the heap

Programming Languages on the Web, Now and the Future

Michael Lippautz
Google

Thank you, questions?