haha v8 engine go brrrrr

Syed Faraz Abrar

@farazsth98

whoami

Cyber Security Undergraduate @ Curtin University

Security Researcher @ elttam

Currently focusing on Chrome

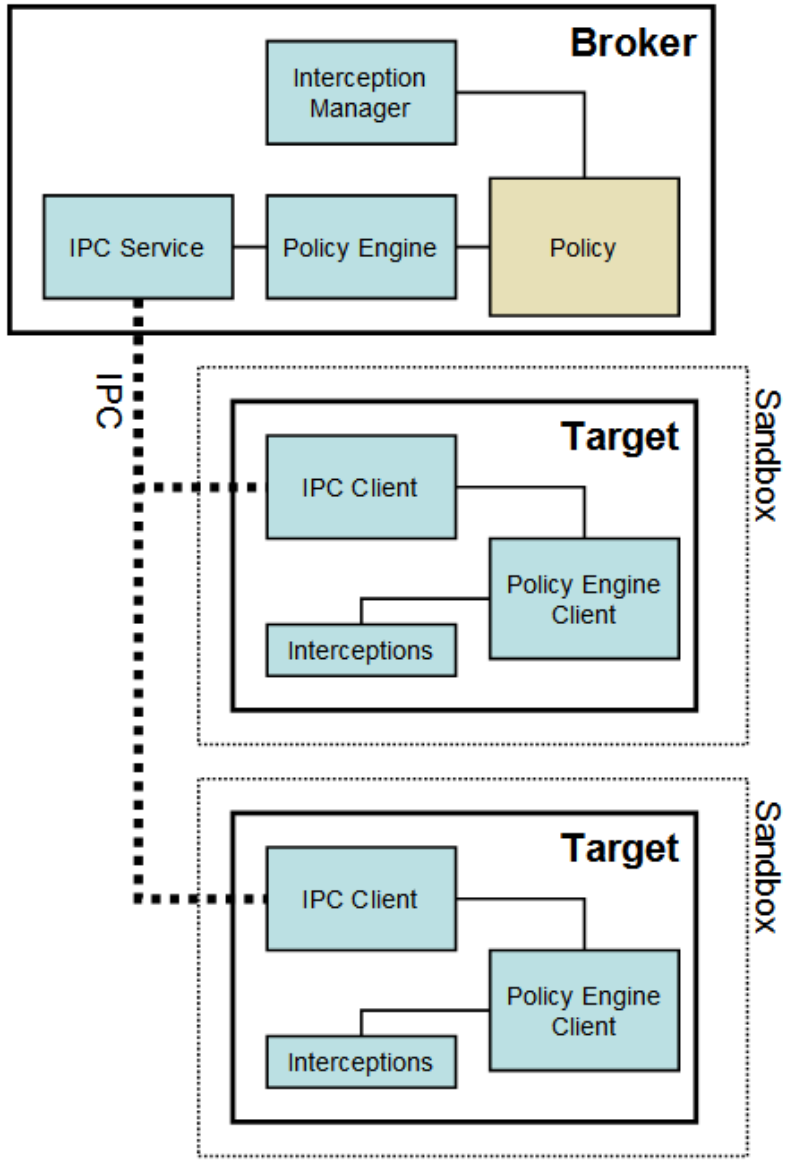
@farazsth98

TL;DR

- Background information on browsers and sandboxing
- Internals of a couple of V8 components
- Analysis of a vulnerability for each of them
  - Focusing on bug hunting, not exploitation
- Tips on approaching V8 research

Browsers

Modern browsers are composed of several processes.

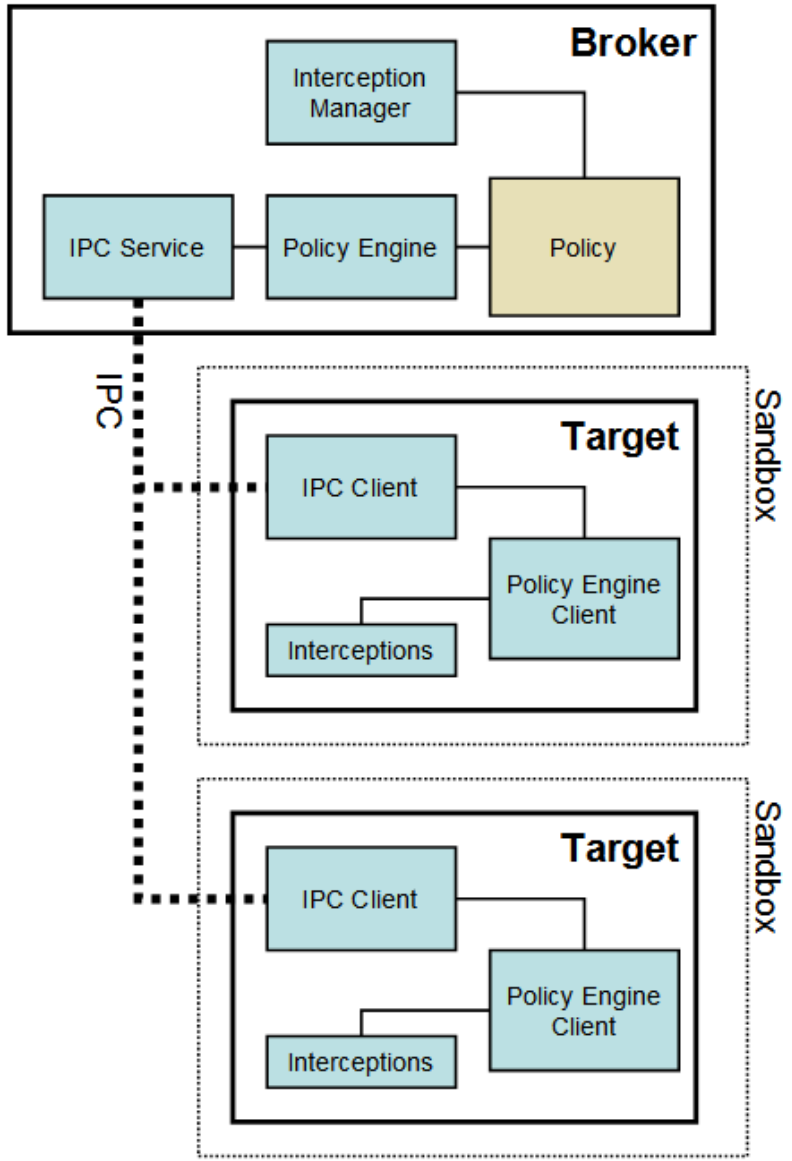


https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

Browsers

Modern browsers are composed of several processes.

The main *Broker* process has the most privileges, and handles creating all other *Target* processes.



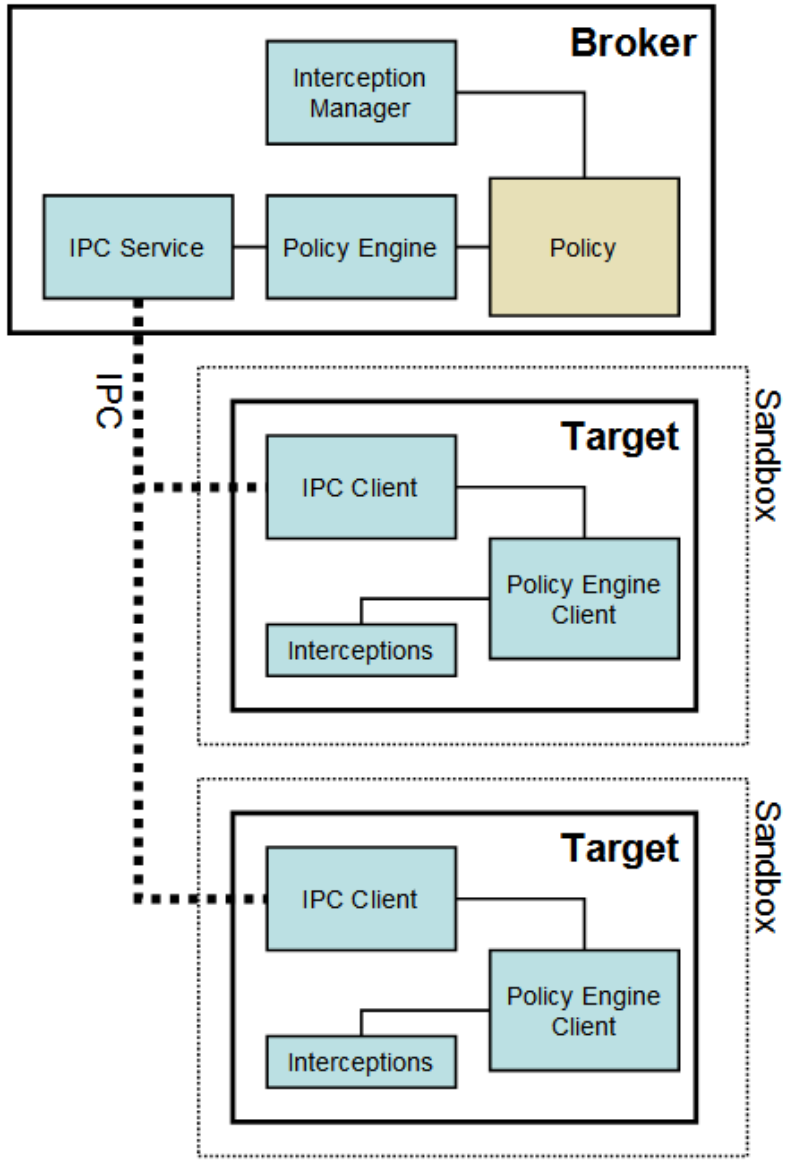
https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

Browsers

Modern browsers are composed of several processes.

The main *Broker* process has the most privileges, and handles creating all other *Target* processes.

These *Target* processes are considered untrusted by default.



https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

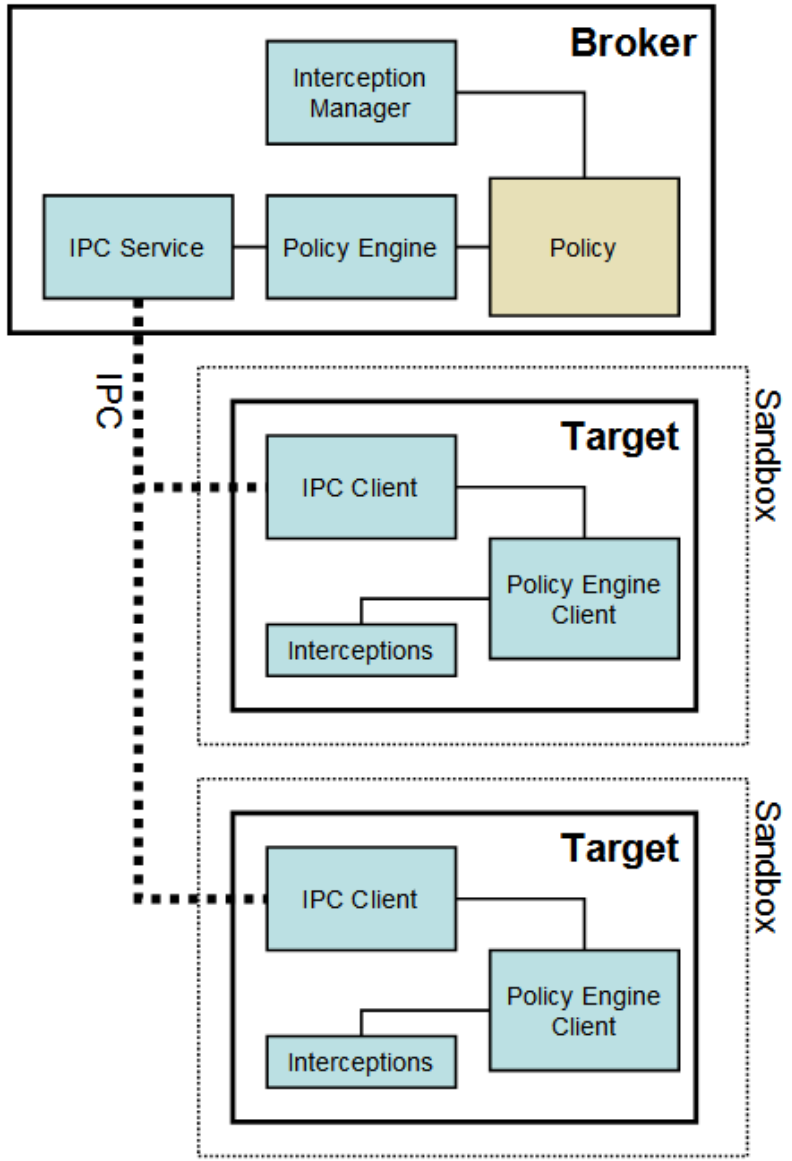
Browsers

Modern browsers are composed of several processes.

The main *Broker* process has the most privileges, and handles creating all other *Target* processes.

These *Target* processes are considered untrusted by default.

Renderer processes, **GPU** processes, **DRM** processes, etc, are all examples of *Target* processes.



https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

Browsers

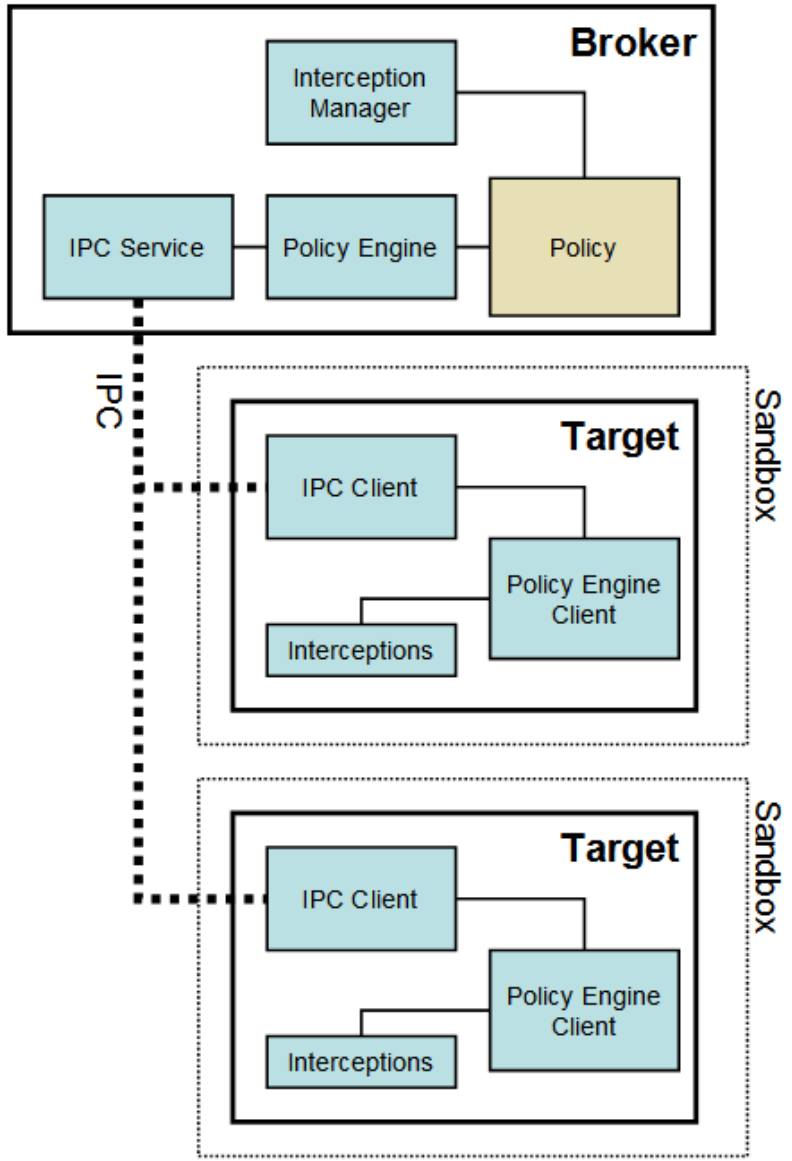
Modern browsers are composed of several processes.

The main *Broker* process has the most privileges, and handles creating all other *Target* processes.

These *Target* processes are considered untrusted by default.

Renderer processes, **GPU** processes, **DRM** processes, etc, are all examples of *Target* processes.

All communication is done through IPC.



https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

Browsers

Modern browsers are composed of several processes.

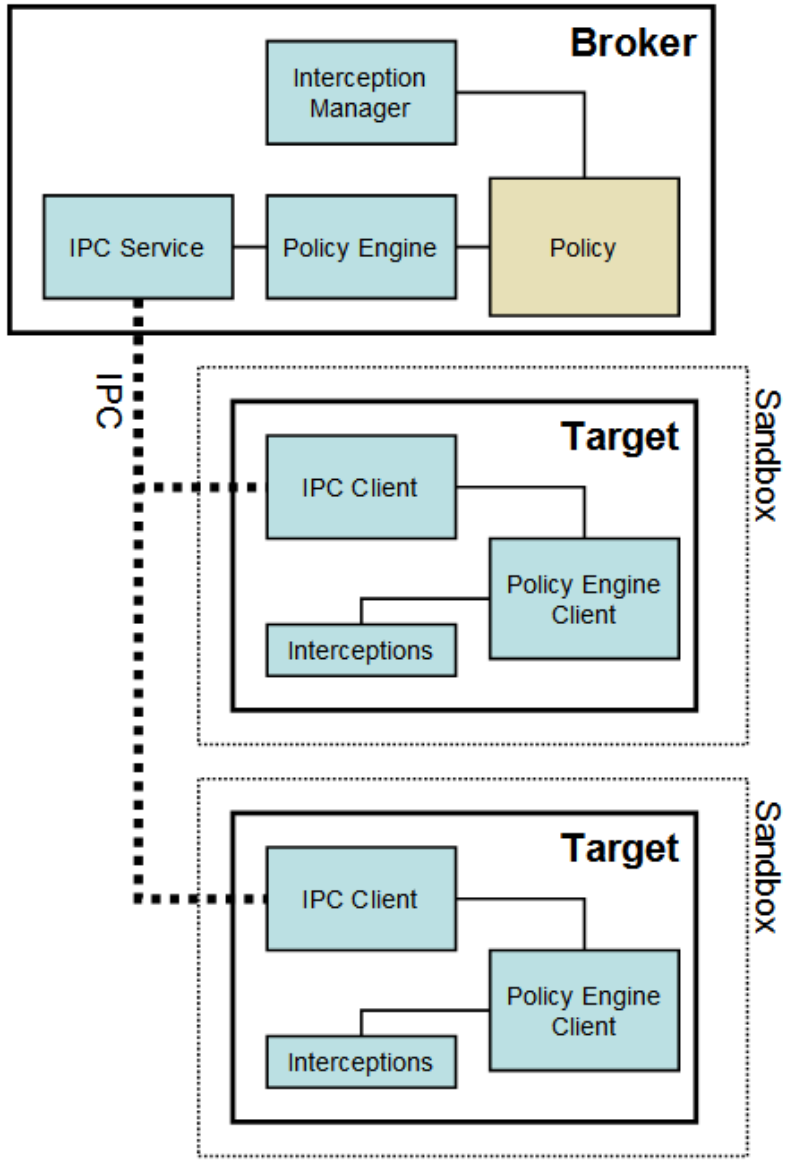
The main *Broker* process has the most privileges, and handles creating all other *Target* processes.

These *Target* processes are considered untrusted by default.

Renderer processes, **GPU** processes, **DRM** processes, etc, are all examples of *Target* processes.

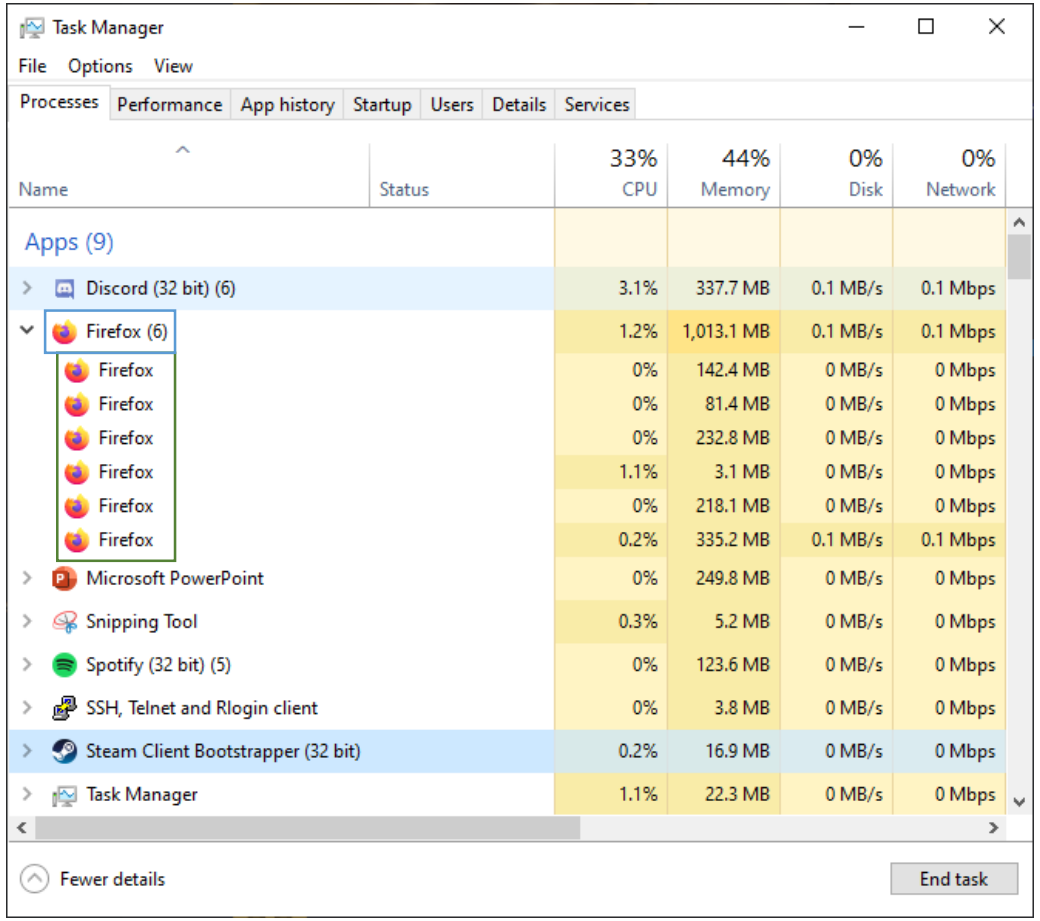
All communication is done through IPC.

Child processes are always sandboxed.



https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

Multiprocess Architecture



@farazsth98

Sandboxing (`chromium/src/sandbox/`)

Sandboxed processes have a number of different restrictions.

- Cannot access the file system
- Cannot perform arbitrary system calls
- Cannot create more child processes
- And some more…

Sandboxing (`chromium/src/sandbox/`)

Sandboxed processes have a number of different restrictions.

- Cannot access the file system
- Cannot perform arbitrary system calls
- Cannot create more child processes
- And some more…

Sandboxing is implemented differently in Windows and Linux.

- Restricted tokens, Job objects, Desktop objects, and Integrity levels used on Windows
- Seccomp-BPF, namespaces, Setuid sandbox, SELinux, and AppArmor used on Linux

Sandboxing (`chromium/src/sandbox/`)

Sandboxed processes have a number of different restrictions.

- Cannot access the file system
- Cannot perform arbitrary system calls
- Cannot create more child processes
- And some more…

Sandboxing is implemented differently in Windows and Linux.

- Restricted tokens, Job objects, Desktop objects, and Integrity levels used on Windows
- Seccomp-BPF, namespaces, Setuid sandbox, SELinux, and AppArmor used on Linux

The *Browser* process intercepts all system calls made by the sandboxed processes and handles them.

Motivation for attacking V8

The end goal is to always exploit the Chrome *Broker* process.

Motivation for attacking V8

The end goal is to always exploit the Chrome *Broker* process.

Multiple vulnerabilities are needed to achieve this.

Motivation for attacking V8

The end goal is to always exploit the Chrome *Broker* process.

Multiple vulnerabilities are needed to achieve this.

One of the most exposed processes is the **Renderer** process.

Motivation for attacking V8

The end goal is to always exploit the Chrome *Broker* process.

Multiple vulnerabilities are needed to achieve this.

One of the most exposed processes is the **Renderer** process.

V8 is Chrome’s JavaScript engine, and it runs within every renderer process.

Motivation for attacking V8

The end goal is to always exploit the Chrome *Broker* process.

Multiple vulnerabilities are needed to achieve this.

One of the most exposed processes is the **Renderer** process.

V8 is Chrome’s JavaScript engine, and it runs within every renderer process.

The enormous complexity of V8 means it contains entirely new and unique vulnerability classes.

V8 Heap

V8 uses a compressed heap in x86_64.

V8 Heap

V8 uses a compressed heap in x86_64.

All pointers are stored as 32-bit addresses.

V8 Heap

V8 uses a compressed heap in x86_64.

All pointers are stored as 32-bit addresses.

The *base address* of the heap is stored in the *r13* register (called the *root register*).

V8 Heap

V8 uses a compressed heap in x86_64.

All pointers are stored as 32-bit addresses.

The *base address* of the heap is stored in the *r13* register (called the *root register*).

The *base address* is added to pointers prior to dereferencing.

Value Representation

V8 uses a technique called **pointer tagging** to distinguish between heap pointers and small integers.

Value Representation

V8 uses a technique called **pointer tagging** to distinguish between heap pointers and small integers.

There are two types of *tagged representations*: A *Smi* (short for **Small Integer**), and a *HeapObject*.

Value Representation

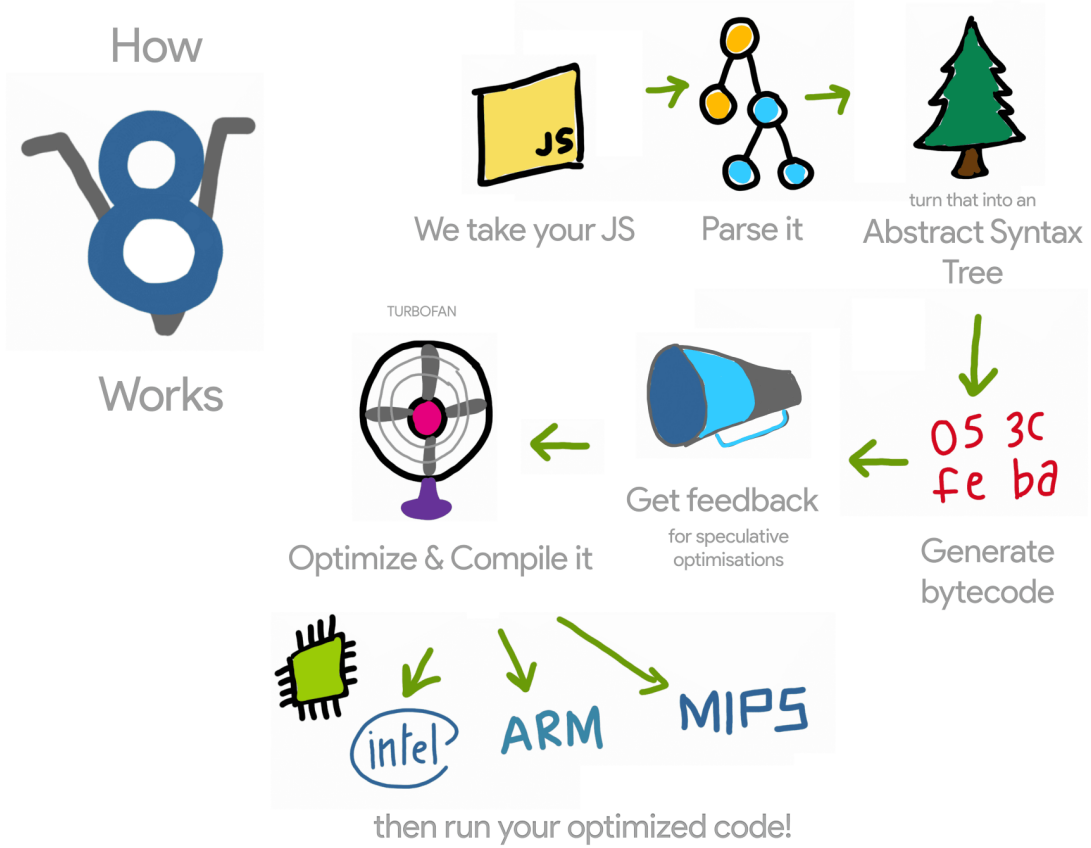
V8 uses a technique called **pointer tagging** to distinguish between heap pointers and small integers.

There are two types of *tagged representations*: A *Smi* (short for **Small Integer**), and a *HeapObject*.

`v8/src/objects/objects.h`

```
// Formats of Object::ptr_:
//  Smi:        [31 bit signed int] 0
//  HeapObject: [32 bit direct pointer] (4 byte aligned) | 01
```


V8 Pipeline

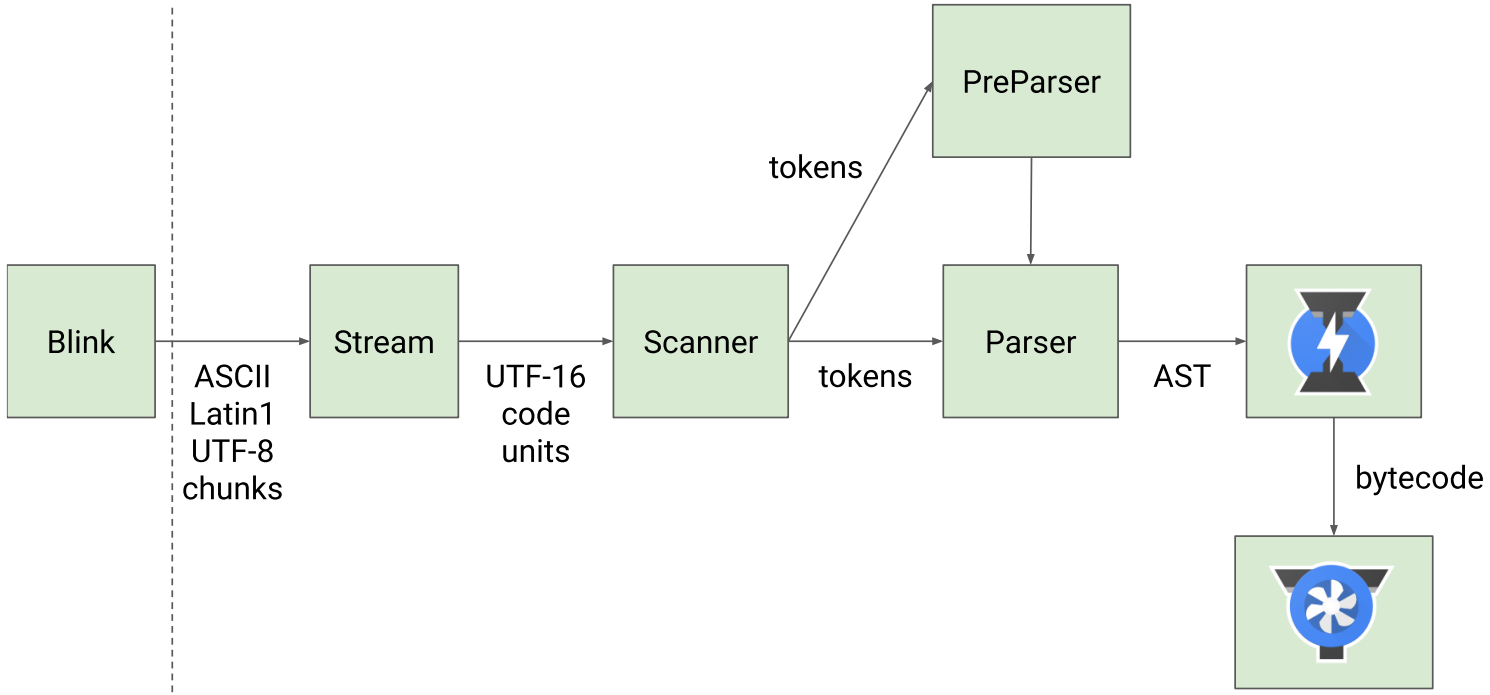


By @addyosmani

https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Parsing (`v8/src/parsing`)

JavaScript code comes in from the network as UTF-8 encoded text.

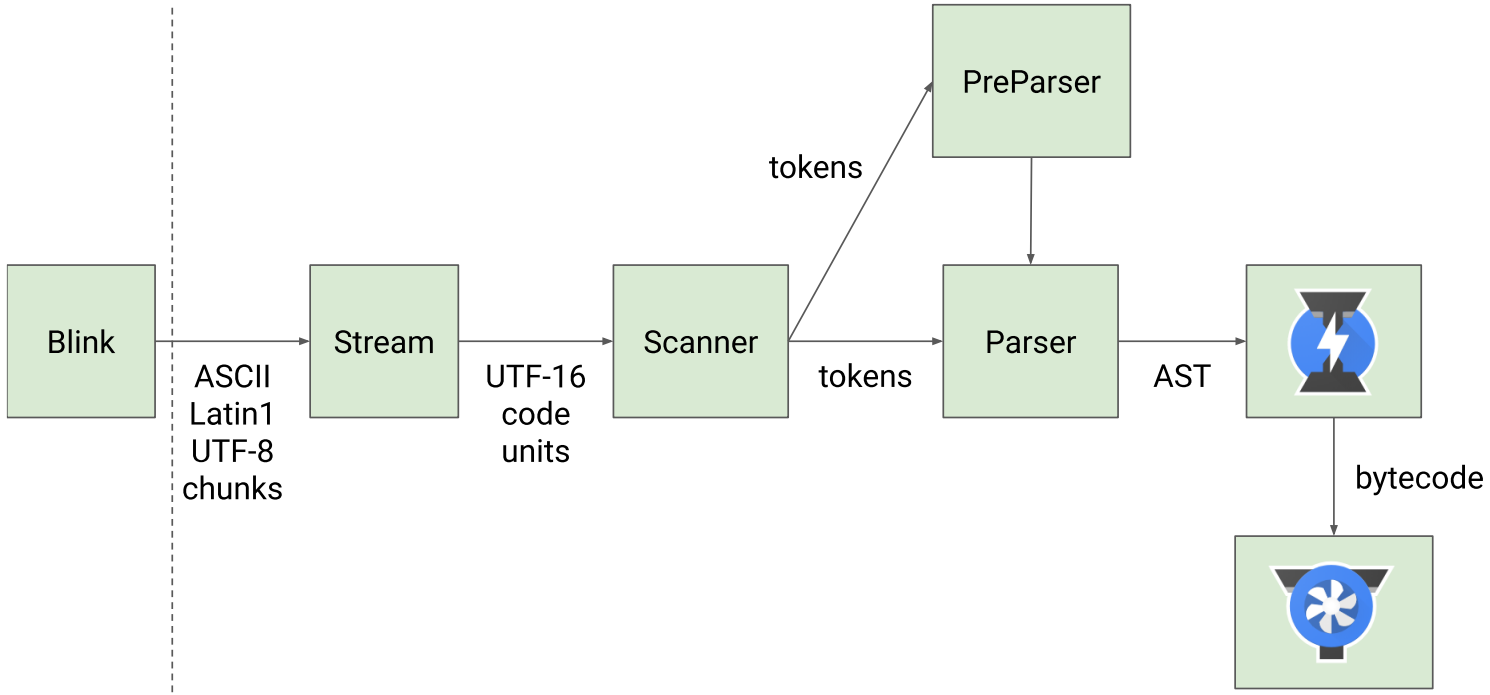


https://v8.dev/img/scanner/overview.svg

Parsing (`v8/src/parsing`)

JavaScript code comes in from the network as UTF-8 encoded text.

The **Stream** converts this to UTF-16, after which the **Scanner** tokenizes the code.



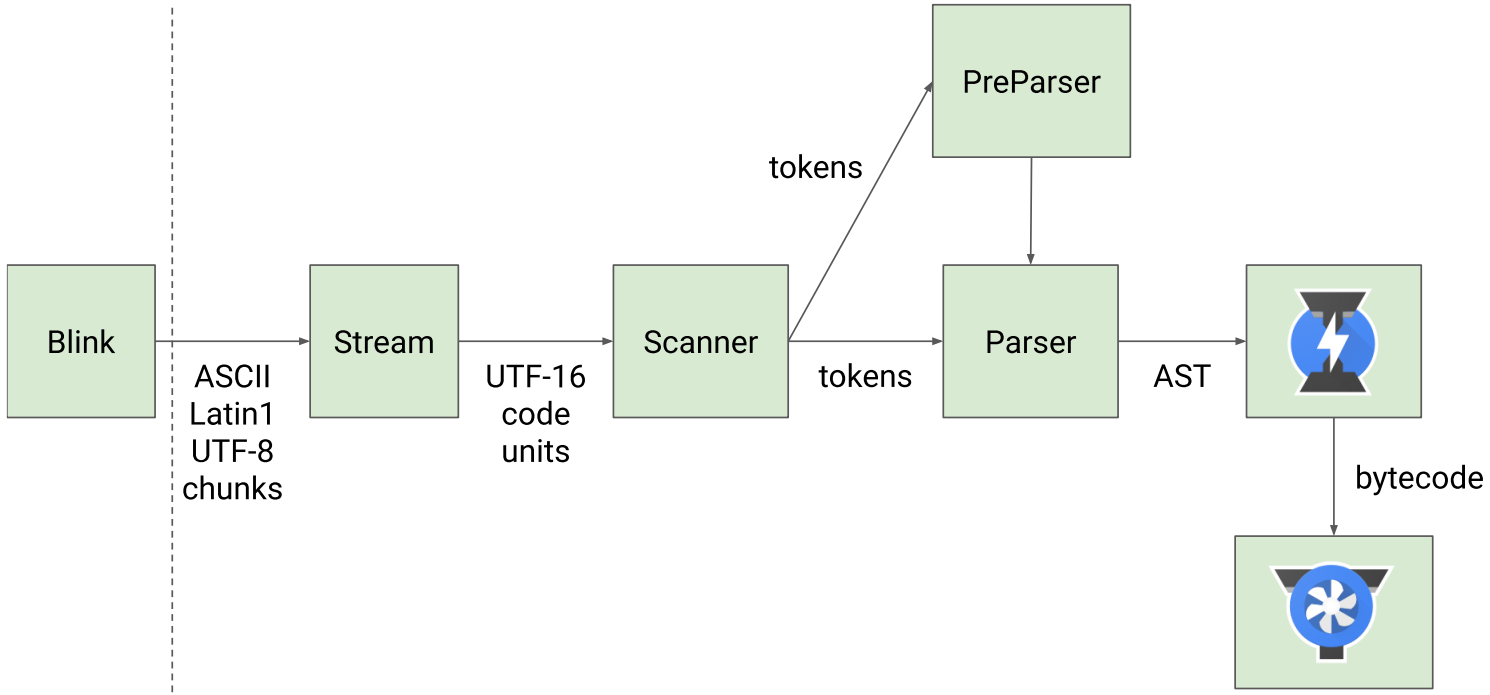
https://v8.dev/img/scanner/overview.svg

Parsing (`v8/src/parsing`)

JavaScript code comes in from the network as UTF-8 encoded text.

The **Stream** converts this to UTF-16, after which the **Scanner** tokenizes the code.

The **Preparser** is used for *lazy parsing*, where it skips parsing any *functions*, but makes sure they're syntactically valid.



https://v8.dev/img/scanner/overview.svg

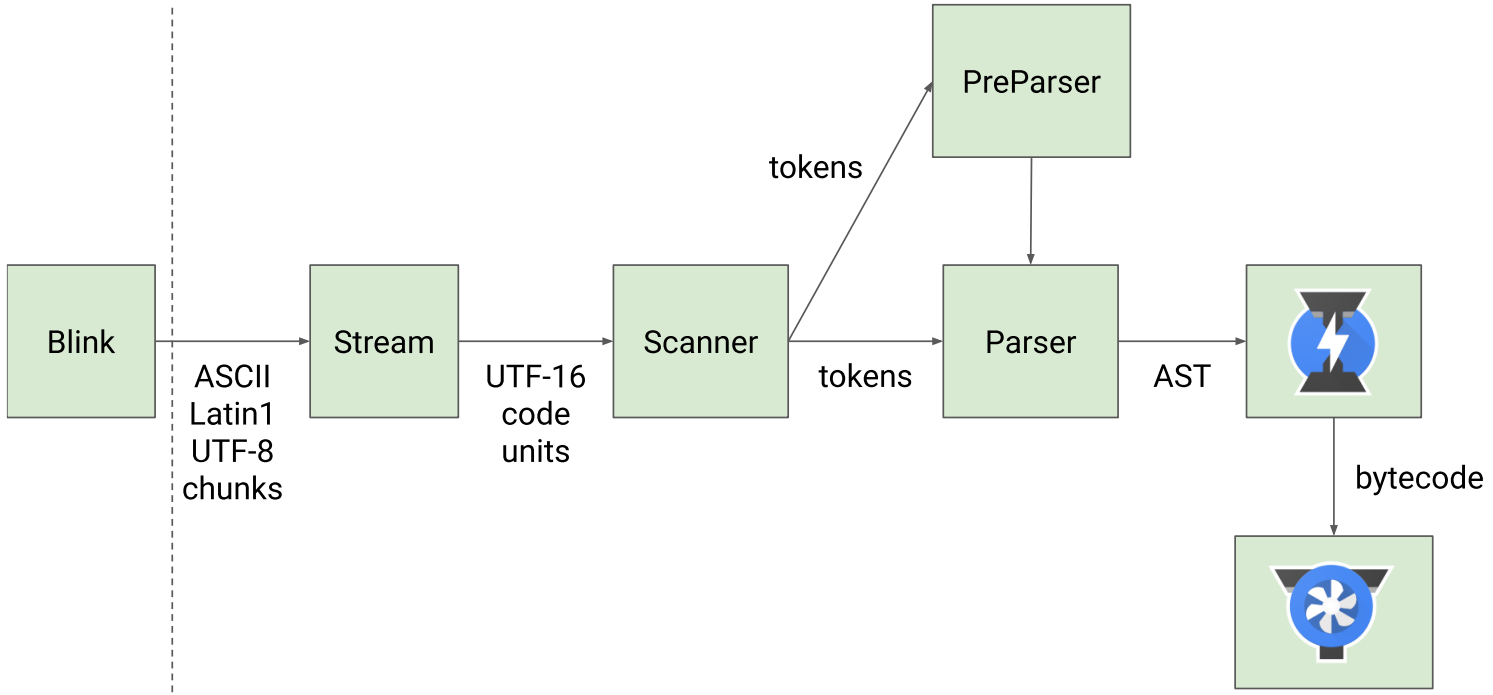
Parsing (`v8/src/parsing`)

JavaScript code comes in from the network as UTF-8 encoded text.

The **Stream** converts this to UTF-16, after which the **Scanner** tokenizes the code.

The **Preparser** is used for *lazy parsing*, where it skips parsing any *functions*, but makes sure they're syntactically valid.

The **Parser** simply the code into an *Abstract Syntax Tree* (AST). The **Parser** might also parse functions that were *Preparsed* previously if they are now required to run.

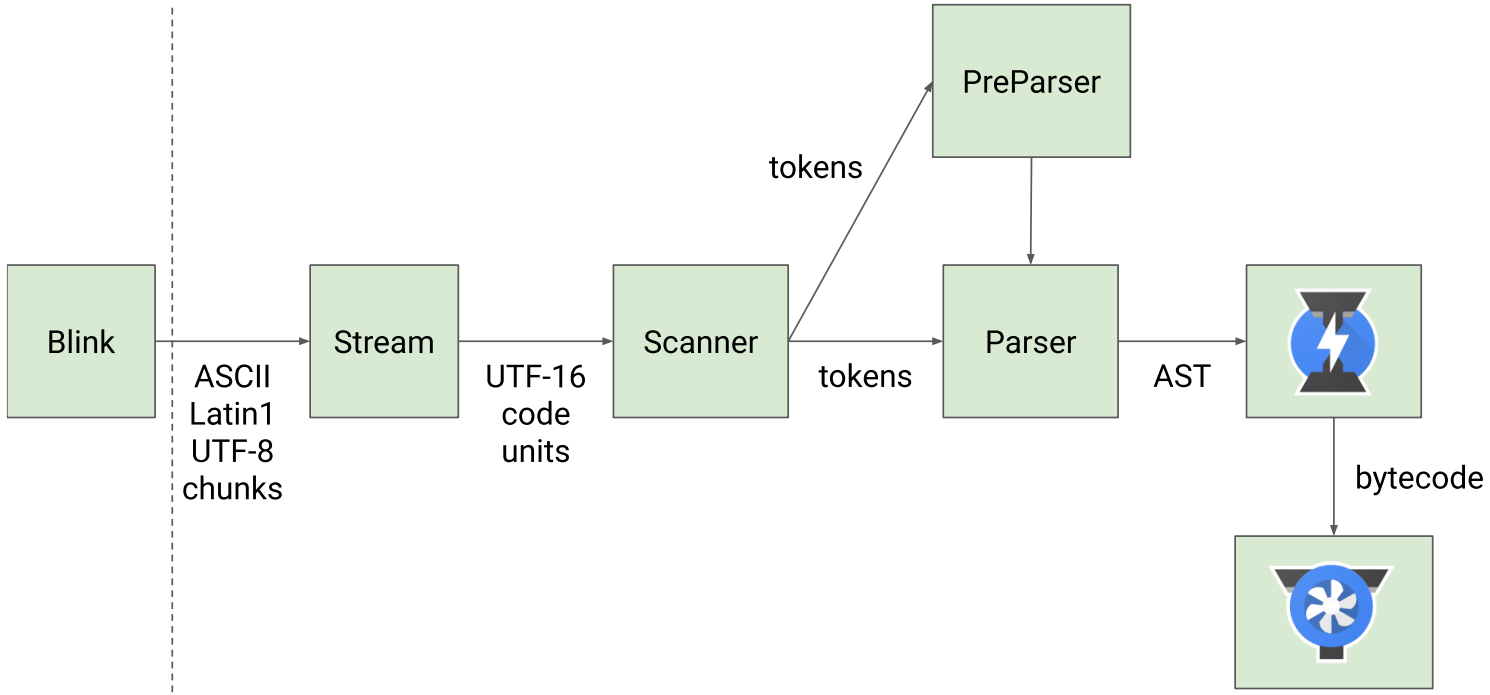


https://v8.dev/img/scanner/overview.svg

@farazsth98

Parsing (`v8/src/parsing`)

The AST is then passed into **Ignition**, which is V8's bytecode compiler / interpreter.

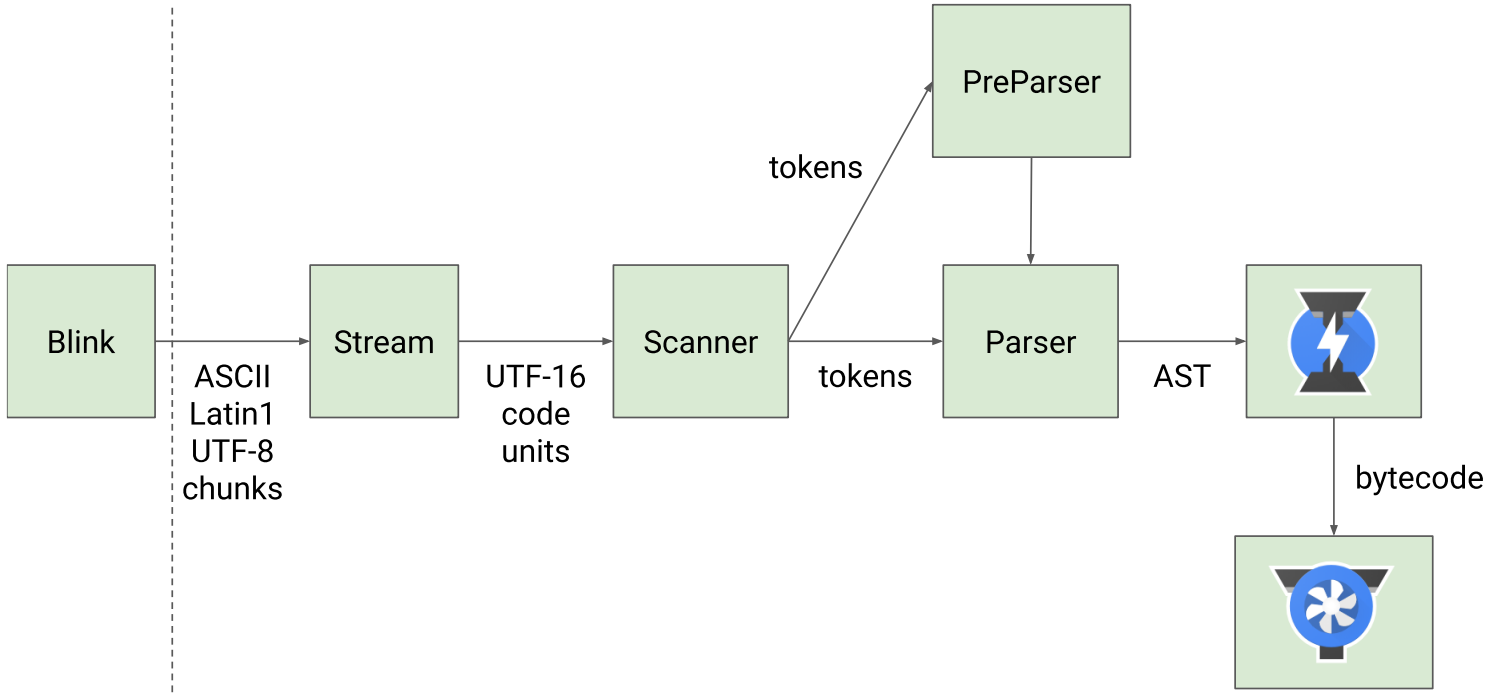


https://v8.dev/img/scanner/overview.svg

Parsing (`v8/src/parsing`)

The AST is then passed into **Ignition**, which is V8's bytecode compiler / interpreter.

The Ignition *compiler* first compiles the AST into bytecode.



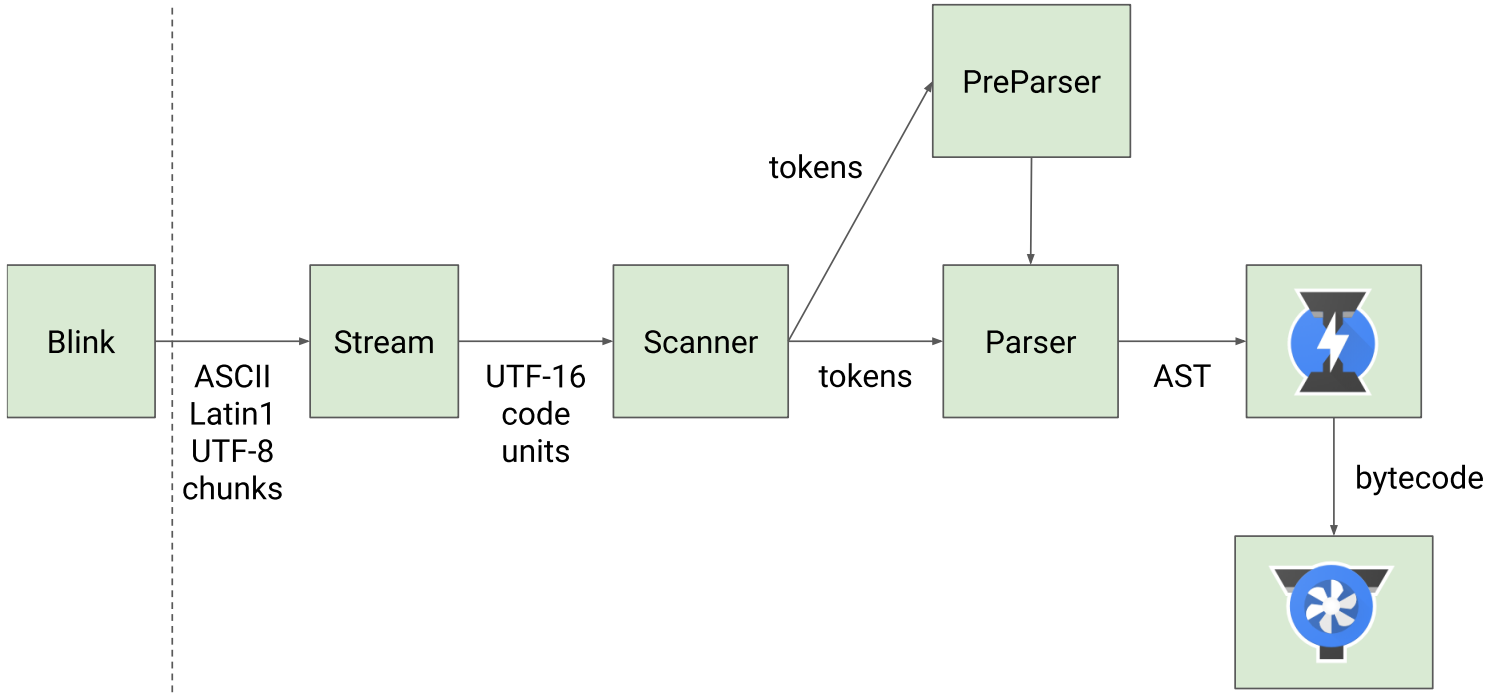
https://v8.dev/img/scanner/overview.svg

Parsing (`v8/src/parsing`)

The AST is then passed into **Ignition**, which is V8's bytecode compiler / interpreter.

The Ignition *compiler* first compiles the AST into bytecode.

Finally, the Ignition *interpreter* interprets and executes this bytecode.



https://v8.dev/img/scanner/overview.svg

CVE-2019-5790 – LiteralBuffer Integer Overflow

Integer overflow within the *Scanner* found by Dimitry Fourny of Bluefrostsecurity.

CVE-2019-5790 – LiteralBuffer Integer Overflow

Integer overflow within the *Scanner* found by Dimitry Fourny of Bluefrostsecurity.

The `Scanner::Scan` method starts off by calling `Scanner::ScanSingleToken` to read in a single token.

CVE-2019-5790 – LiteralBuffer Integer Overflow

Integer overflow within the *Scanner* found by Dimitry Fourny of Bluefrostsecurity.

The `Scanner::Scan` method starts off by calling `Scanner::ScanSingleToken` to read in a single token.

Let's take a String as an example. We get the following call stack:

```
Scanner::Scan -> Scanner::ScanSingleToken ->
Scanner::ScanString -> Scanner::AddLiteralChar
-> Scanner::LiteralBuffer::AddChar ->
Scanner::LiteralBuffer::AddTwoByteChar
```


CVE-2019-5790 – LiteralBuffer Integer Overflow

```
void Scanner::LiteralBuffer::AddTwoByteChar(uc32 code_unit) {
  DCHECK(!is_one_byte());
  if (position_ >= backing_store_.length()) ExpandBuffer();
  if (code_unit <=
      static_cast(unibrow::Utf16::kMaxNonSurrogateCharCode)) {
    *reinterpret_cast<uint16_t*>(&backing_store_[position_]) = code_unit;
    position_ += kUC16Size;
  } else {
    *reinterpret_cast<uint16_t*>(&backing_store_[position_]) =
        unibrow::Utf16::LeadSurrogate(code_unit);
    position_ += kUC16Size;
    if (position_ >= backing_store_.length()) ExpandBuffer();
    *reinterpret_cast<uint16_t*>(&backing_store_[position_]) =
        unibrow::Utf16::TrailSurrogate(code_unit);
    position_ += kUC16Size;
  }
}
```

@farazsth98

CVE-2019-5790 – LiteralBuffer Integer Overflow

```
void Scanner::LiteralBuffer::ExpandBuffer() {
  Vector new_store = Vector::New(NewCapacity(kInitialCapacity));
  MemCopy(new_store.start(), backing_store_.start(), position_);
  backing_store_.Dispose();
  backing_store_ = new_store;
}
```

`kInitialCapacity = 16;`

```
int Scanner::LiteralBuffer::NewCapacity(int min_capacity) {
  int capacity = Max(min_capacity, backing_store_.length());
  int new_capacity = Min(capacity * kGrowthFactor, capacity + kMaxGrowth);
  return new_capacity;
}
```

`kGrowthFactor = 4; kMaxGrowth = 1024 * 1024;`

What if `(capacity * kGrowthFactor)` overflows and becomes less than `(capacity + kMaxGrowth)`?

@farazsth98

CVE-2019-5790 – LiteralBuffer Integer Overflow

```
int length() const {
  DCHECK_GE(std::numeric_limits<int>::max(), length_); // max length check
  return static_cast<int>(length_);
}
```

A `Vector` cannot have a size greater than $2^{31}-1$.

Proof of Concept:

```
let s = String.fromCharCode(0x4141).repeat(0x10000001) + "A";
s = "'"+s+"'";
eval(s);
```

```
capacity = 2 * 0x10000001 = 0x20000002
capacity * kGrowthFactory = 0x20000002 * 4 = 0x80000008
capacity + kMaxGrowth = 0x20000002 + (1024 * 1024) = 0x20100002
```

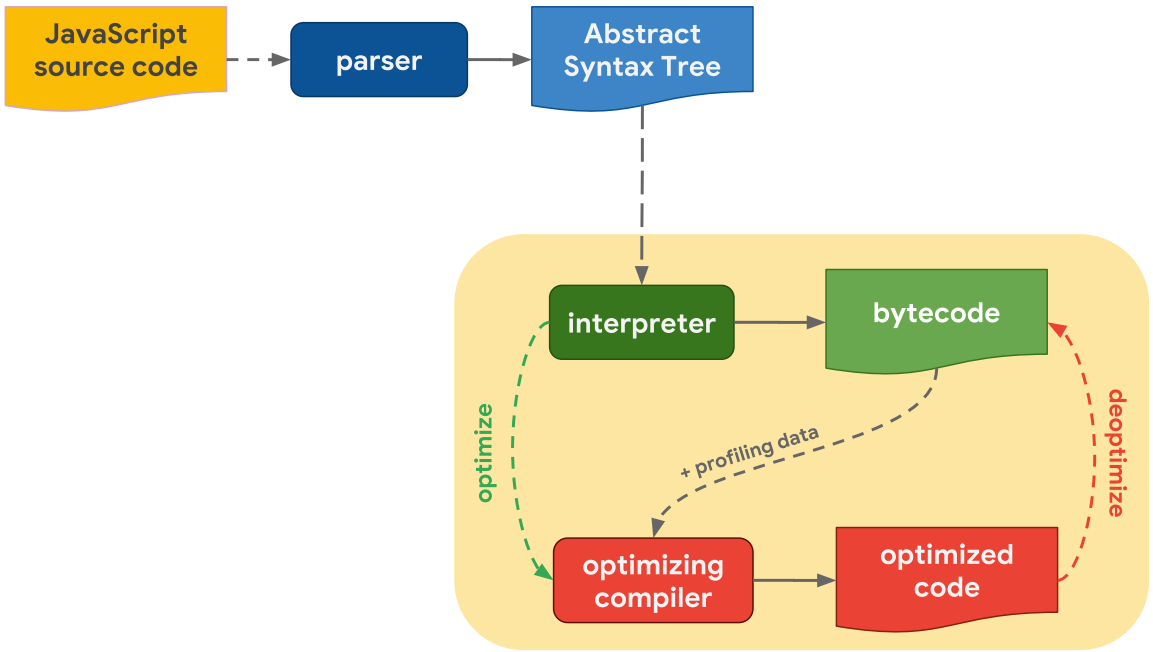
32-bit signed integer max value = $2^{31}-1$ = 0x7FFFFFFF

The first calculation overflows and yields a smaller value than the original capacity. This will in turn result in a heap overflow when the `MemCopy` is called later on.

@farazsth98

Ignition (`v8/src/ignition`)

Ignition is V8's fast low-level register-based interpreter.

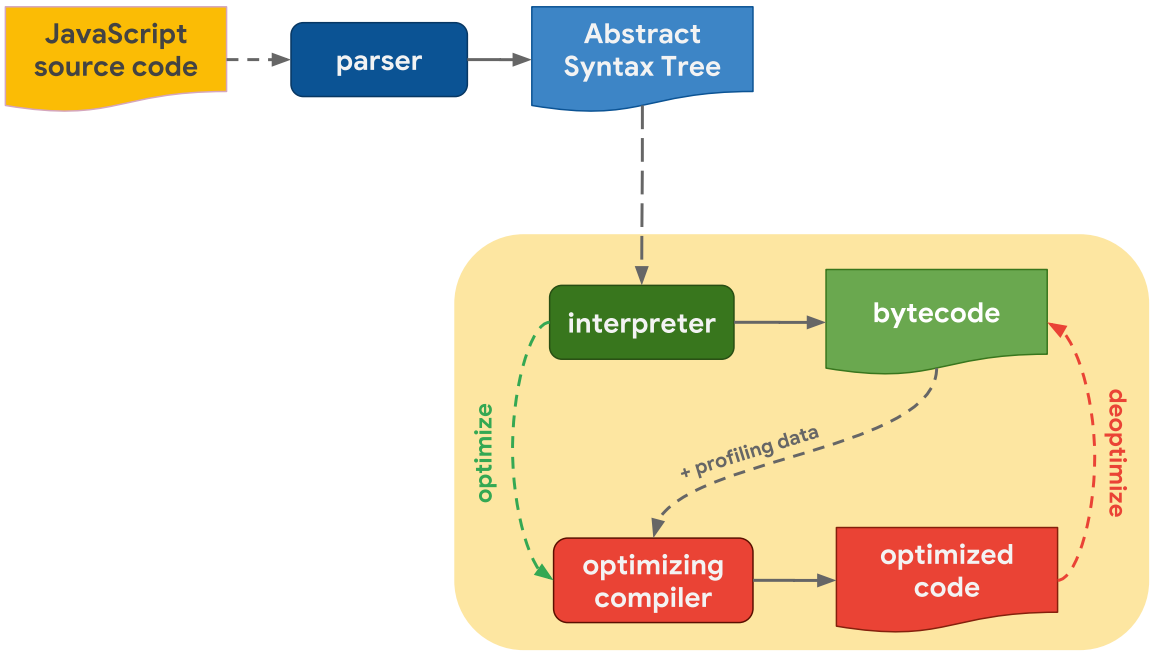


https://mathiasbynens.be/notes/shapes-ics

Ignition (`v8/src/ignition`)

Ignition is V8's fast low-level register-based interpreter.

It takes the AST generated by the **Parser** and compiles it to V8-unique bytecode.



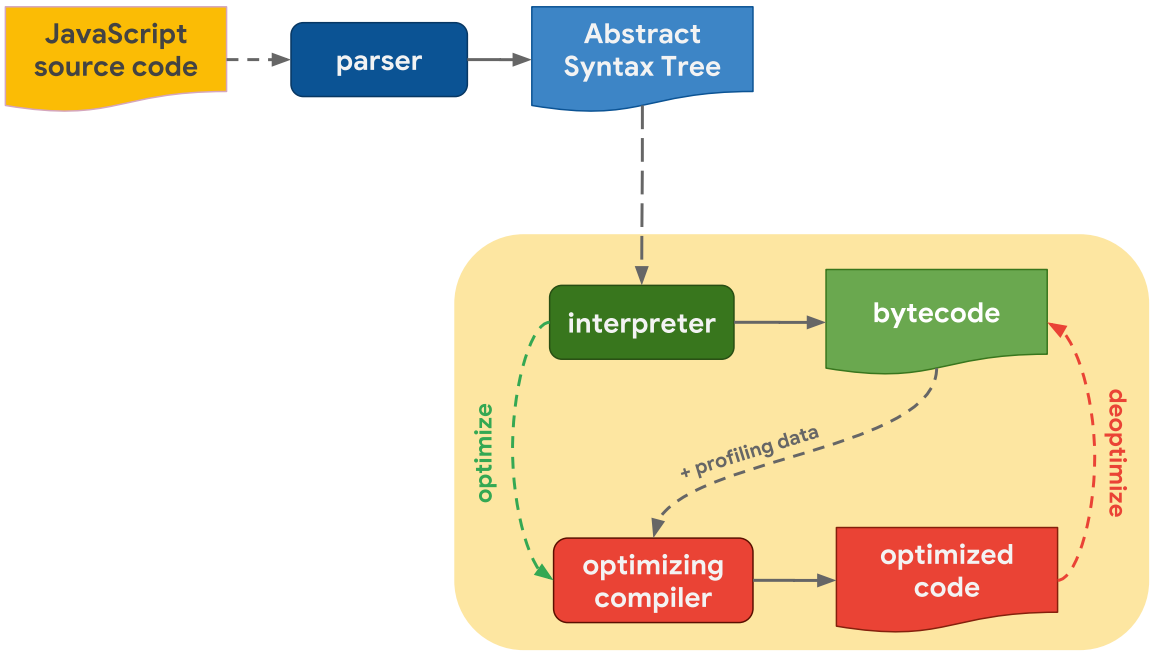
https://mathiasbynens.be/notes/shapes-ics

Ignition (`v8/src/ignition`)

Ignition is V8's fast low-level register-based interpreter.

It takes the AST generated by the **Parser** and compiles it to V8-unique bytecode.

Ignition has a number of different registers that all bytecodes will explicitly use as inputs and outputs.



https://mathiasbynens.be/notes/shapes-ics

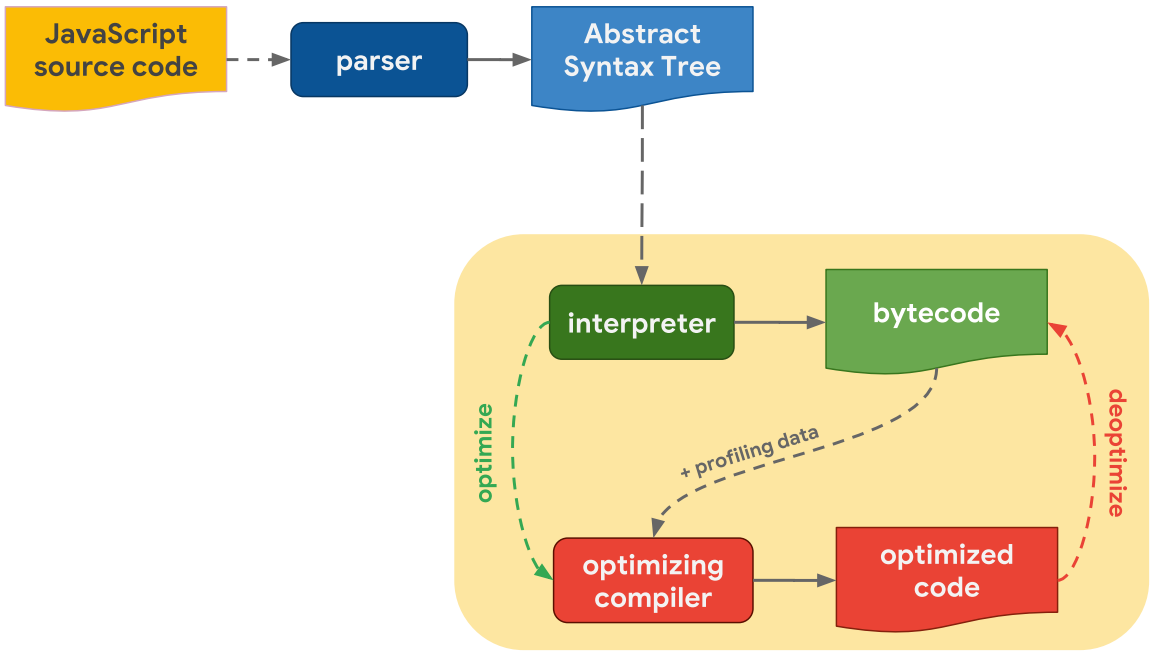
Ignition (`v8/src/ignition`)

Ignition is V8's fast low-level register-based interpreter.

It takes the AST generated by the **Parser** and compiles it to V8-unique bytecode.

Ignition has a number of different registers that all bytecodes will explicitly use as inputs and outputs.

A special register known as the **accumulator** register is used for calculations and return values (similar to RAX in x86).



https://mathiasbynens.be/notes/shapes-ics

Bytecode Execution

```
function add(x, y) {
  return x + y;
}

console.log(add(1, 2));
```

```
$ out/Debug/d8 --print-bytecode add.js
…
[generated bytecode for function: add]
Parameter count 3
Frame size 0
   12 E> 0x37738712a02a @    0 : 94                StackCheck
   23 S> 0x37738712a02b @    1 : 1d 02             Ldar a1
   32 E> 0x37738712a02d @    3 : 29 03 00          Add a0, [0]
   36 S> 0x37738712a030 @    6 : 98                Return
Constant pool (size = 0)
Handler Table (size = 16)
```


https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Bytecode Execution

```
function add(x, y) {
  return x + y;
}

console.log(add(1, 2));
```

```
$ out/Debug/d8 --print-bytecode add.js
…
[generated bytecode for function: add]
Parameter count 3
Frame size 0
   12 E> 0x37738712a02a @    0 : 94                StackCheck
   23 S> 0x37738712a02b @    1 : 1d 02             Ldar a1
   32 E> 0x37738712a02d @    3 : 29 03 00          Add a0, [0]
   36 S> 0x37738712a030 @    6 : 98                Return
Constant pool (size = 0)
Handler Table (size = 16)
```


https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Bytecode Execution

```
function add(x, y) {
  return x + y;
}

console.log(add(1, 2));
```

```
$ out/Debug/d8 --print-bytecode add.js
…
[generated bytecode for function: add]
Parameter count 3
Frame size 0
   12 E> 0x37738712a02a @    0 : 94                StackCheck
   23 S> 0x37738712a02b @    1 : 1d 02             Ldar a1
   32 E> 0x37738712a02d @    3 : 29 03 00          Add a0, [0]
   36 S> 0x37738712a030 @    6 : 98                Return
Constant pool (size = 0)
Handler Table (size = 16)
```


https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Bytecode Execution

```
function add(x, y) {
   return x + y;
}

console.log(add(1, 2));
```

```
$ out/Debug/d8 --print-bytecode add.js
…
[generated bytecode for function: add]
Parameter count 3
Frame size 0
   12 E> 0x37738712a02a @    0 : 94                StackCheck
   23 S> 0x37738712a02b @    1 : 1d 02             Ldar a1
   32 E> 0x37738712a02d @    3 : 29 03 00          Add a0, [0]
   36 S> 0x37738712a030 @    6 : 98                Return
Constant pool (size = 0)
Handler Table (size = 16)
```


https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Bytecode Execution

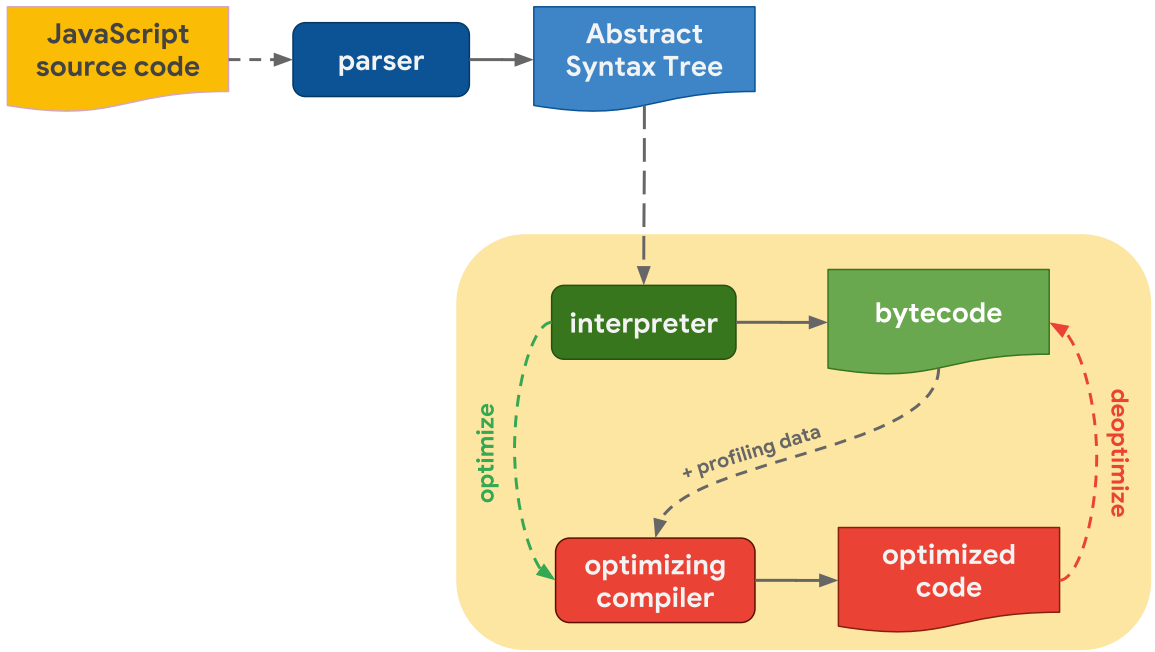
```
function add(x, y) {
  return x + y;
}

console.log(add(1, 2));
```

```
$ out/Debug/d8 --print-bytecode add.js
…
[generated bytecode for function: add]
Parameter count 3
Frame size 0
   12 E> 0x37738712a02a @    0 : 94                StackCheck
   23 S> 0x37738712a02b @    1 : 1d 02             Ldar a1
   32 E> 0x37738712a02d @    3 : 29 03 00          Add a0, [0]
   36 S> 0x37738712a030 @    6 : 98                Return
Constant pool (size = 0)
Handler Table (size = 16)
```


https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

TurboFan (`v8/src/compiler`)

When a function is ran several times, it becomes *hot*.

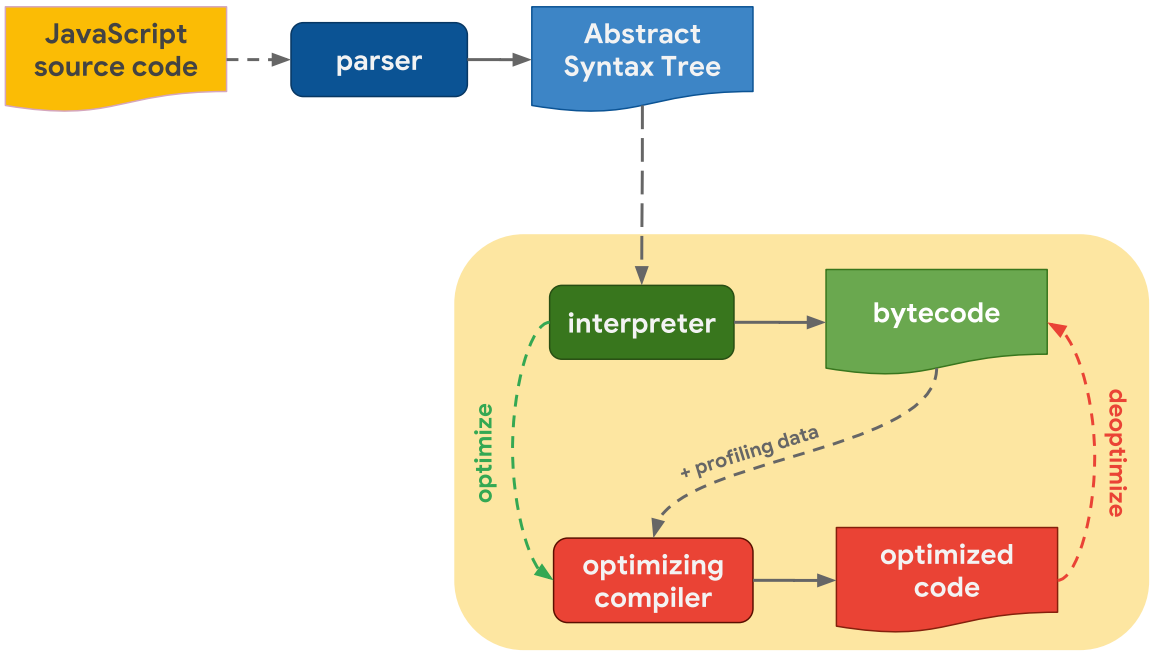


https://mathiasbynens.be/notes/shapes-ics

TurboFan (`v8/src/compiler`)

When a function is ran several times, it becomes *hot*.

Using type feedback and profiling information collected by the interpreter, **TurboFan** speculates that the function only deals with a certain type of value.



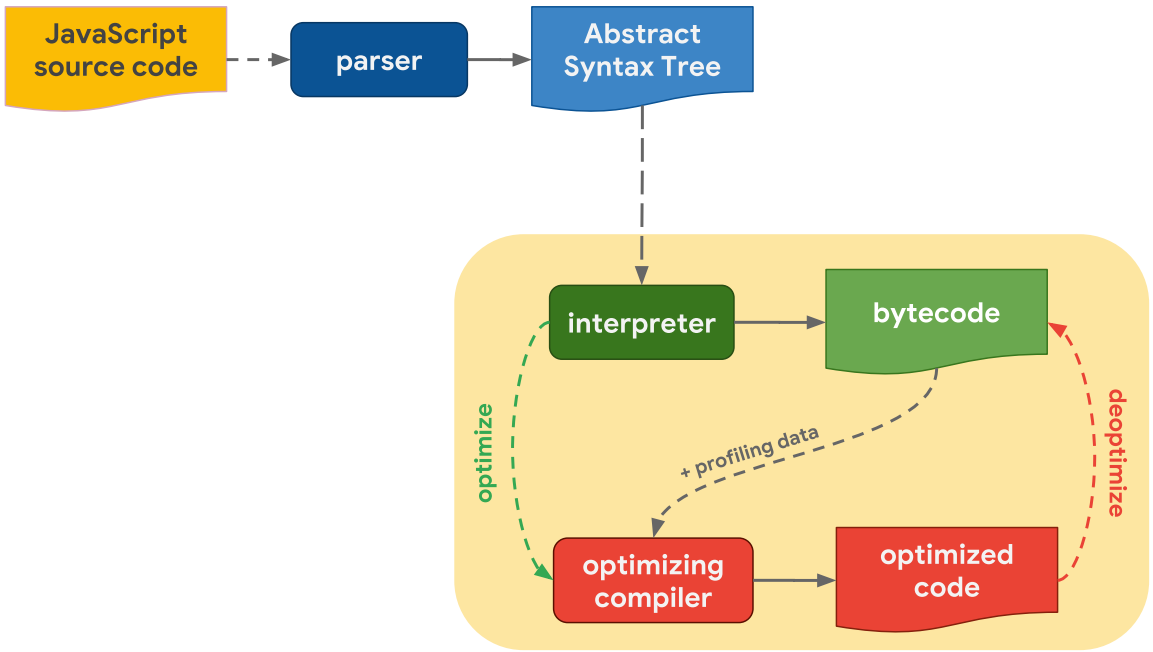
https://mathiasbynens.be/notes/shapes-ics

TurboFan (`v8/src/compiler`)

When a function is ran several times, it becomes *hot*.

Using type feedback and profiling information collected by the interpreter, **TurboFan** speculates that the function only deals with a certain type of value.

It uses these speculations as assumptions to produce highly optimized machine code.



https://mathiasbynens.be/notes/shapes-ics

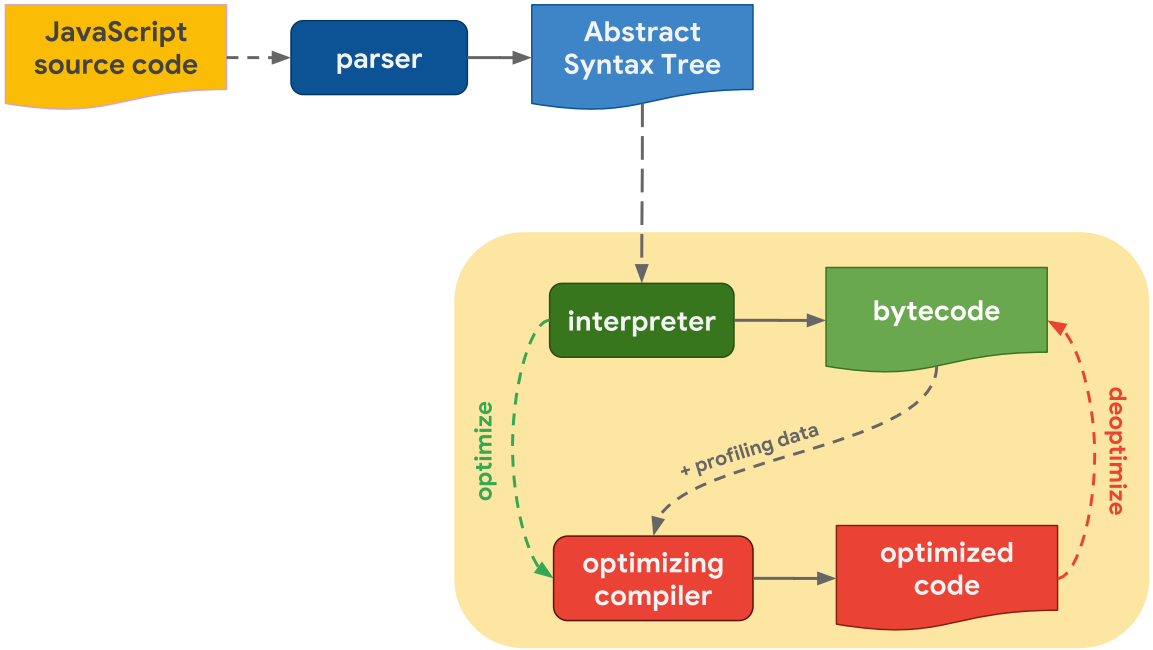
TurboFan (`v8/src/compiler`)

When a function is ran several times, it becomes *hot*.

Using type feedback and profiling information collected by the interpreter, **TurboFan** speculates that the function only deals with a certain type of value.

It uses these speculations as assumptions to produce highly optimized machine code.

If at any point these assumptions fail to hold true, the code is deoptimized and execution goes back to **Ignition**.



https://mathiasbynens.be/notes/shapes-ics

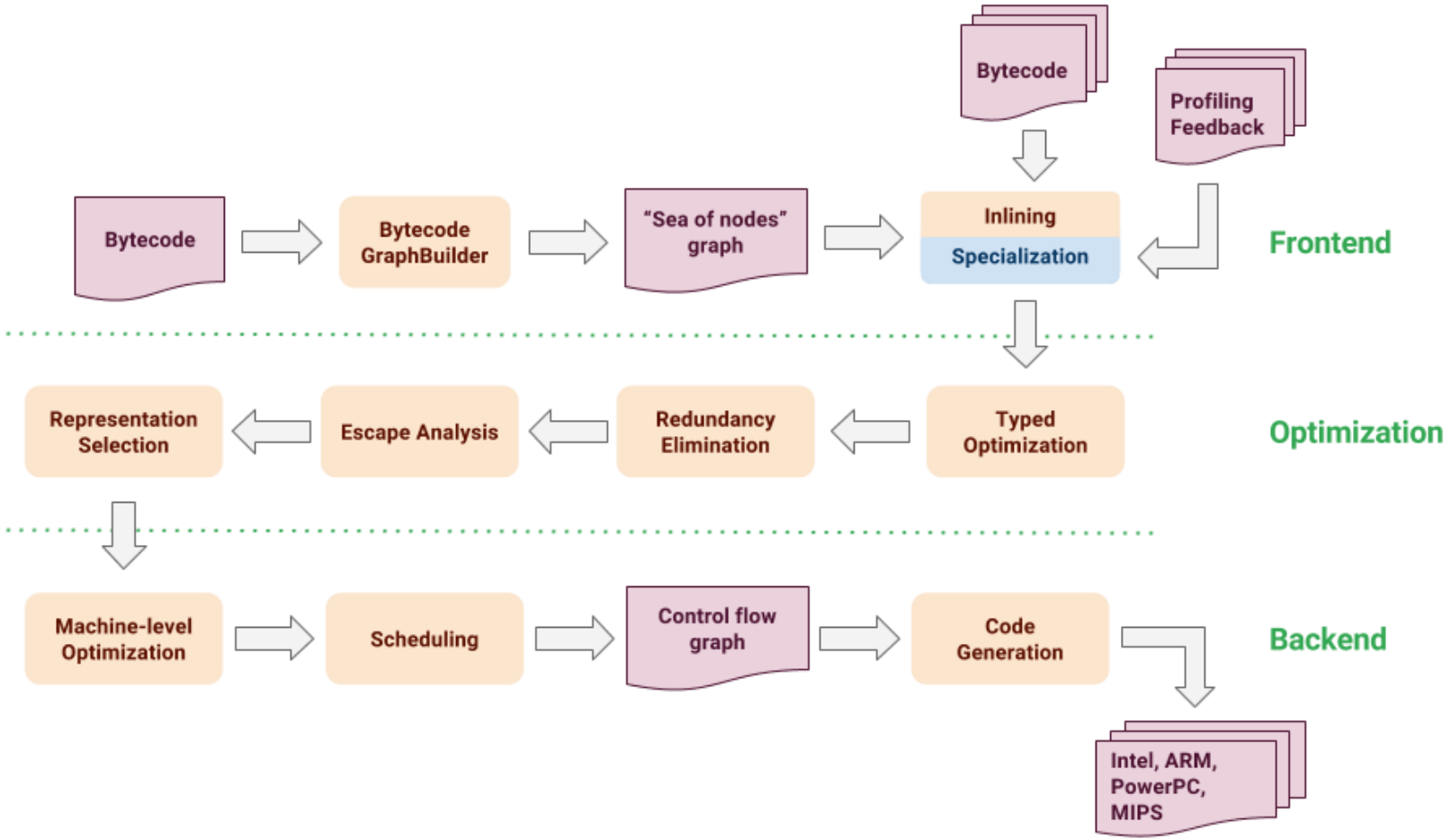
Speculative Optimization

```
function add(x, y) {
    return x + y;
}

for (var i = 0; i < 10000; i++) {
    add(1, 2);
}

add(1.1, 2.2);
```

```
add(1,2); // BinaryOp of SignedSmall
add(1,2); // Stored in Feedback Vector
…
add(1,2); // Optimized for Smis
…
add(1.1, 2.2); // Deoptimize to interpreter
```

@farazsth98

Speculative Optimization



@farazsth98

https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Shapes

Shapes (a.k.a *HiddenClasses*) are used to store information about objects, specifically how their properties are to be accessed.

Shapes

Shapes (a.k.a *HiddenClasses*) are used to store information about objects, specifically how their properties are to be accessed.

They also track *ElementsKinds* for arrays.

Shapes

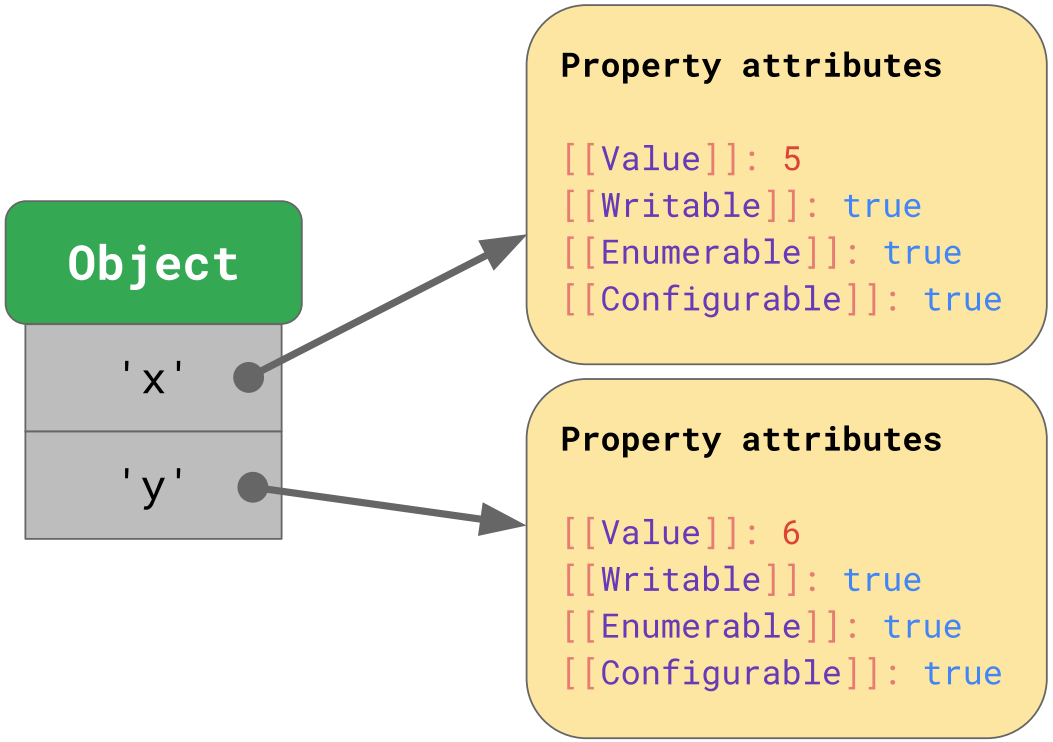
Shapes (a.k.a *HiddenClasses*) are used to store information about objects, specifically how their properties are to be accessed.

They also track *ElementsKinds* for arrays.

Shapes are created on-demand as needed, and shared between objects as much as possible.

Shapes

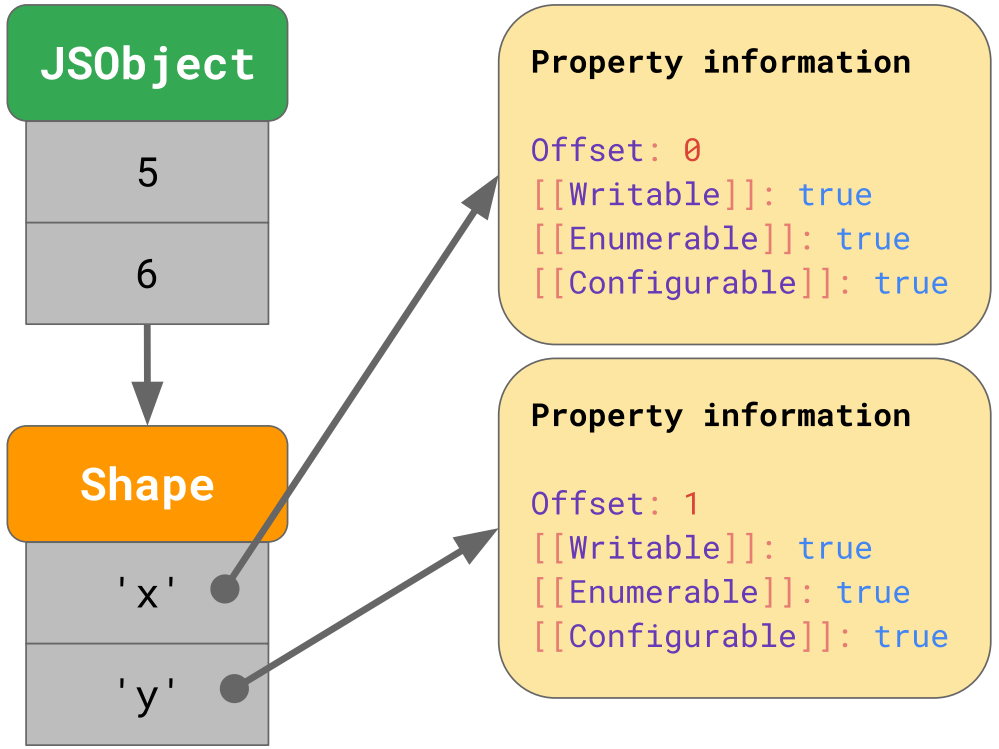
```
object = {
  x: 5,
  y: 6,
};
```



https://mathiasbynens.be/notes/shapes-ics

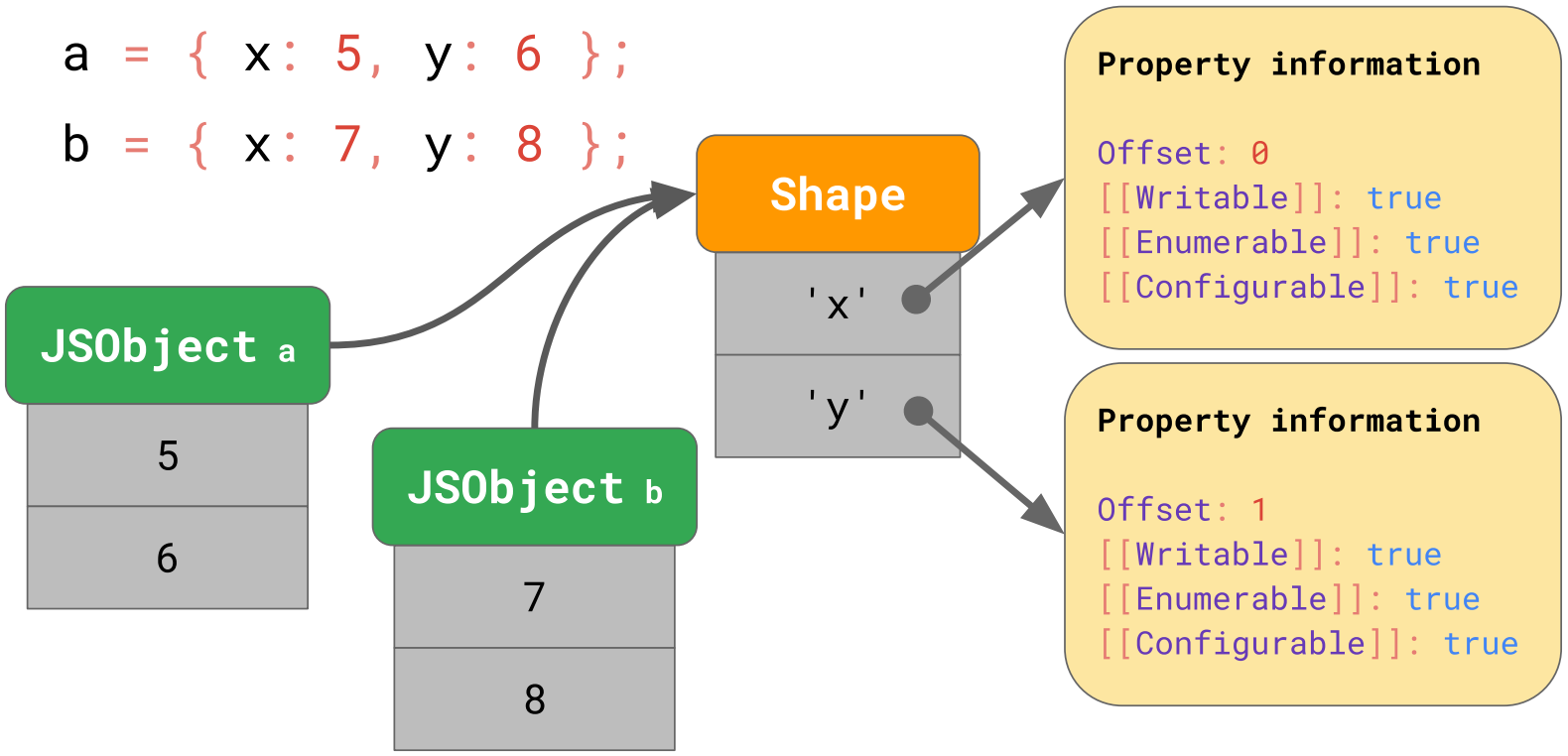
Shapes

```
object = {
  x: 5,
  y: 6,
};
```

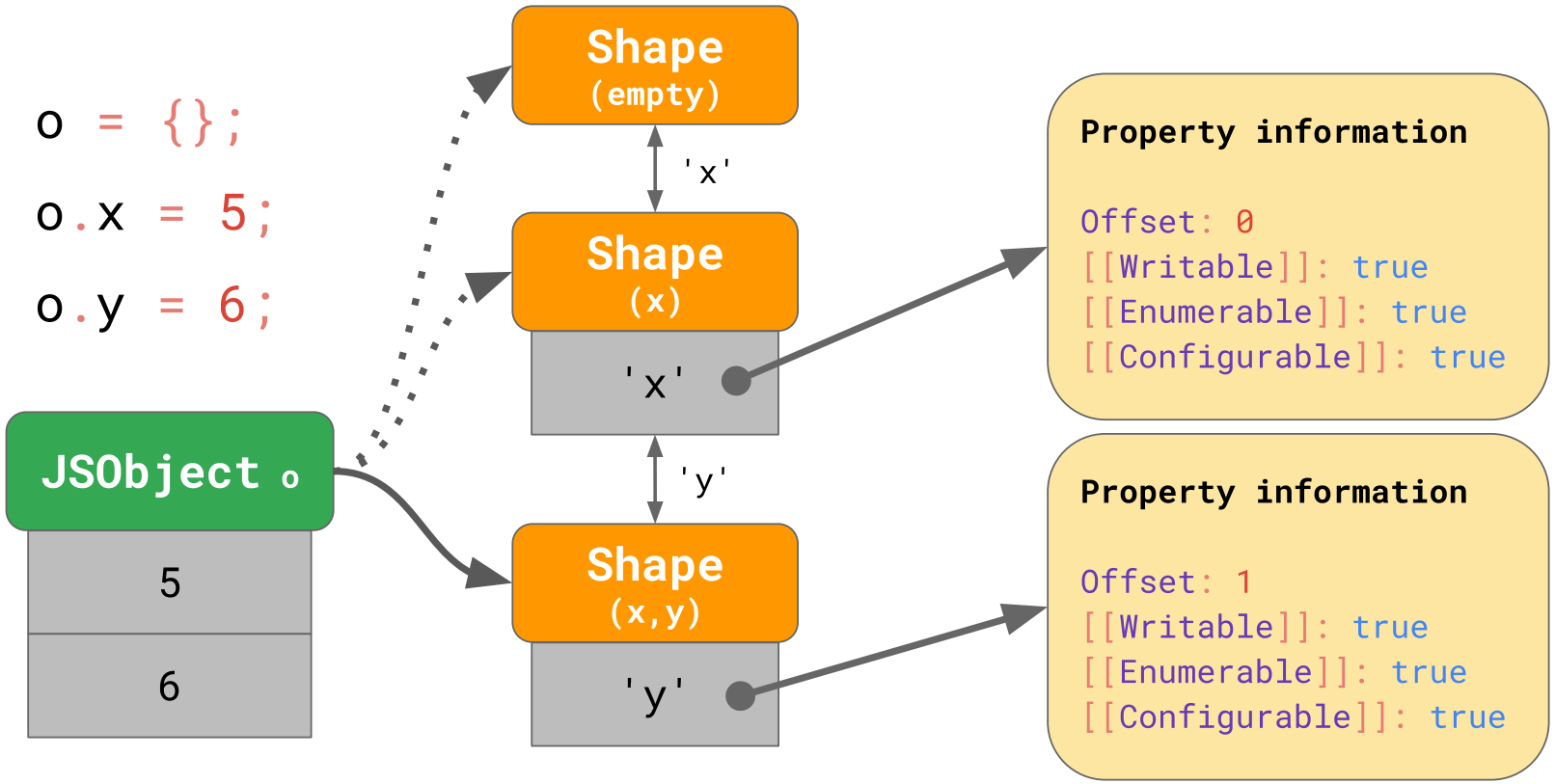


https://mathiasbynens.be/notes/shapes-ics

Shapes

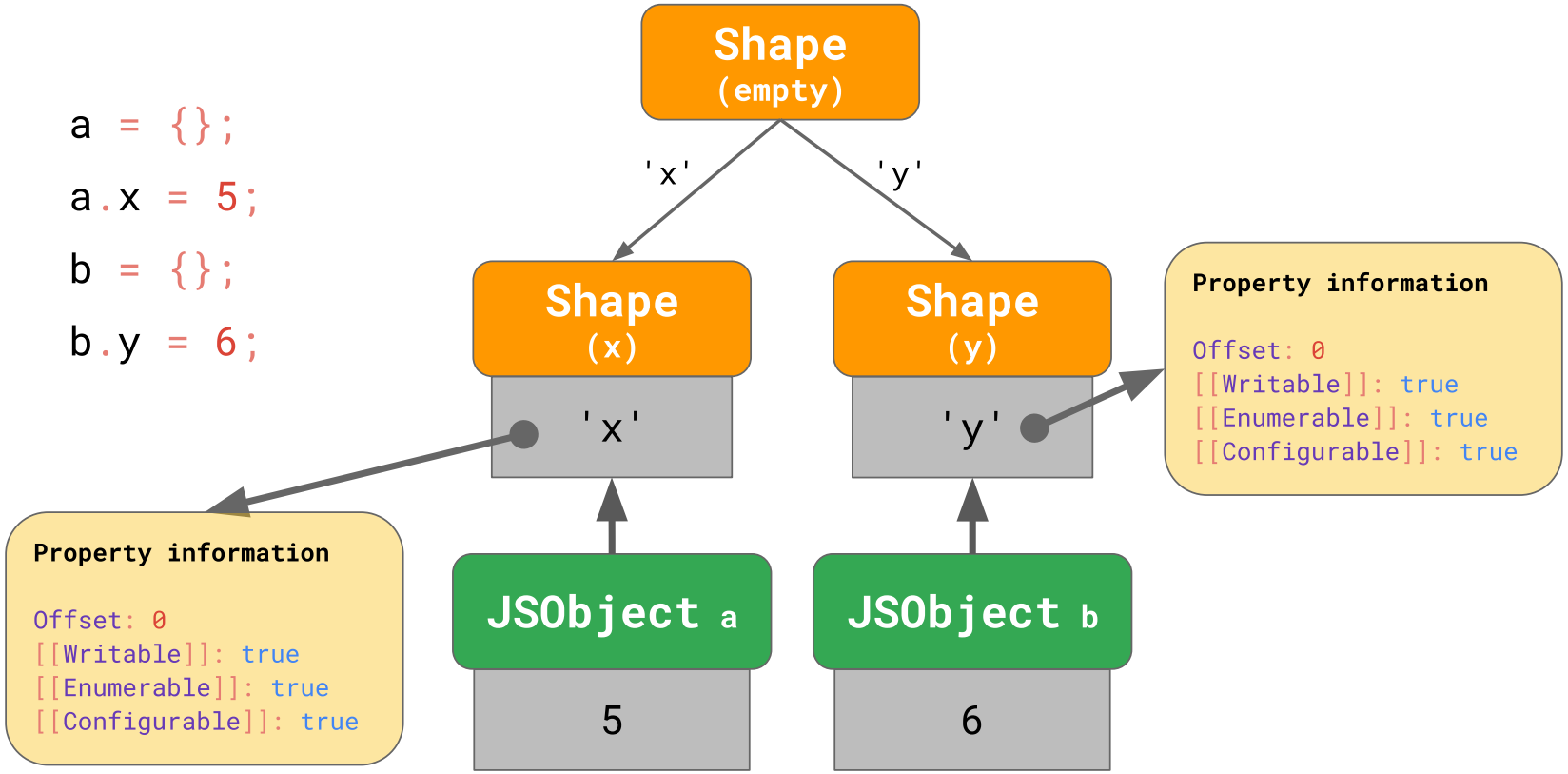

a = { x: 5, y: 6 };
b = { x: 7, y: 8 };
Shape
'x'
'y'
JSObject a
5
6
JSObject b
7
8
Property information
Offset: 0
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true
Property information
Offset: 1
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true

Shapes

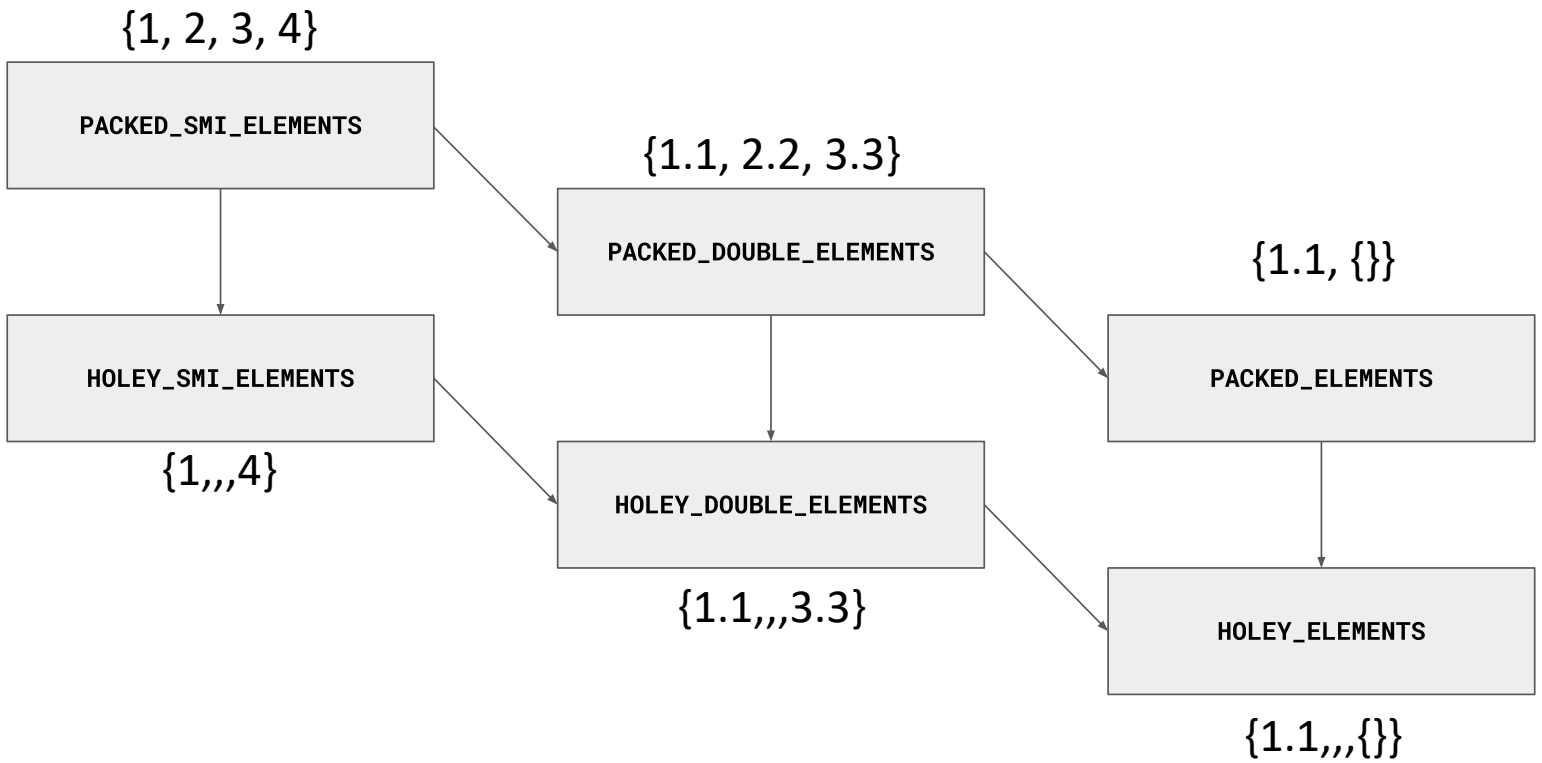

o = {};
o.x = 5;
o.y = 6;
JSObject o
5
6
Shape
(empty)
'x'
Shape
(x)
'x'
'y'
Shape
(x,y)
'y'
Property information
Offset: 0
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true
Property information
Offset: 1
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true

Shapes

```
a = {};
a.x = 5;
b = {};
b.y = 6;
```

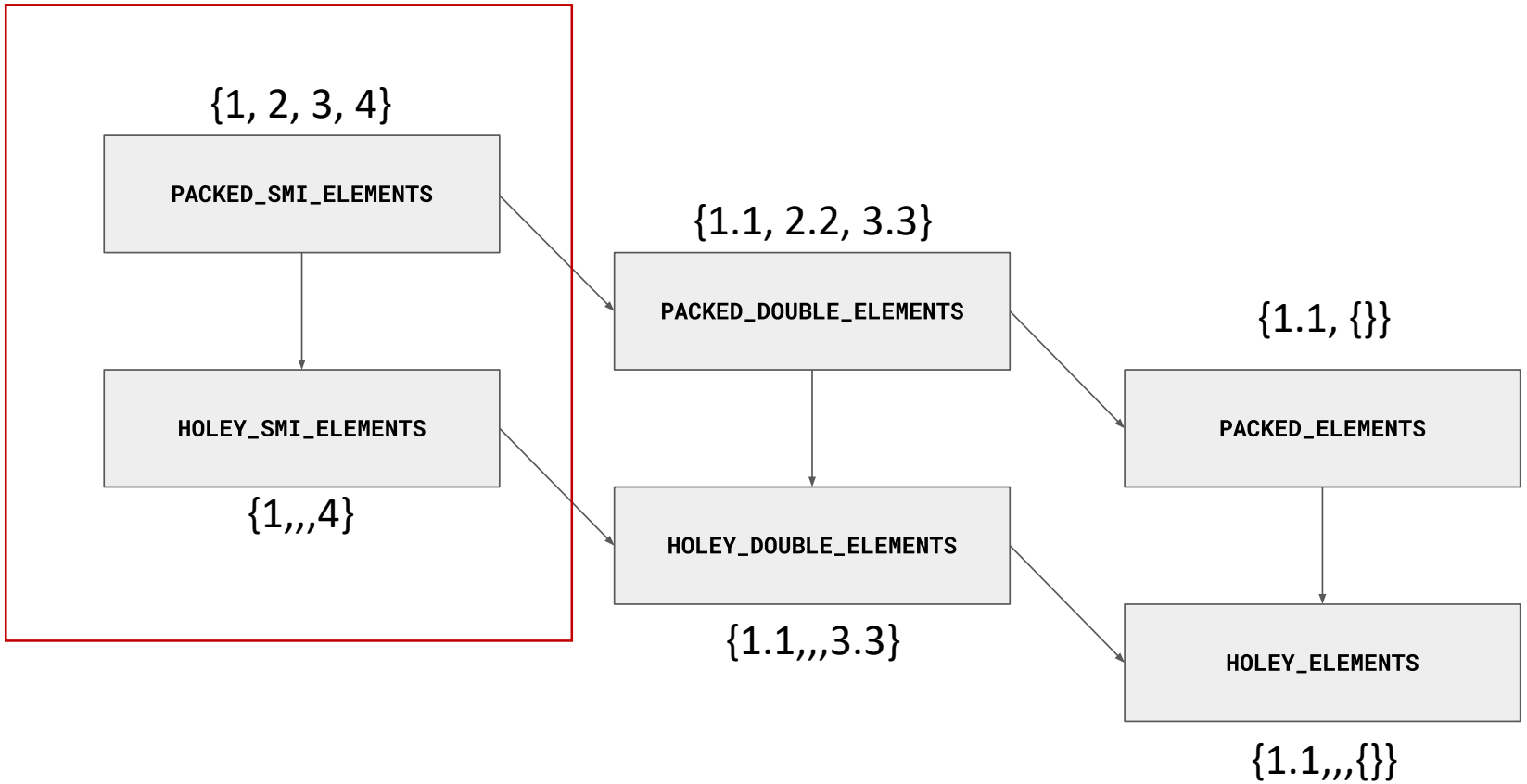


ElementsKind



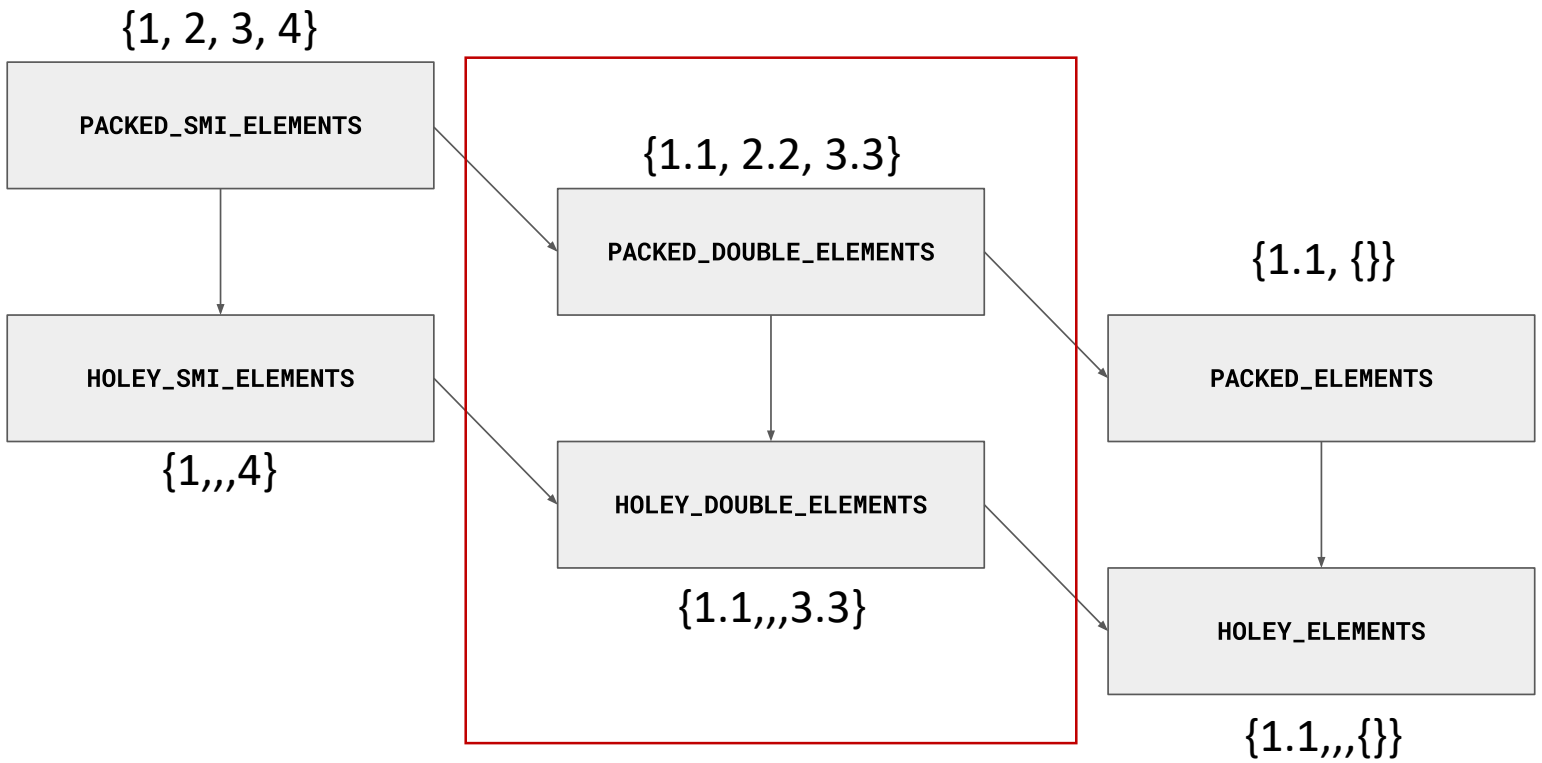
@farazsth98

ElementsKind



@farazsth98

ElementsKind



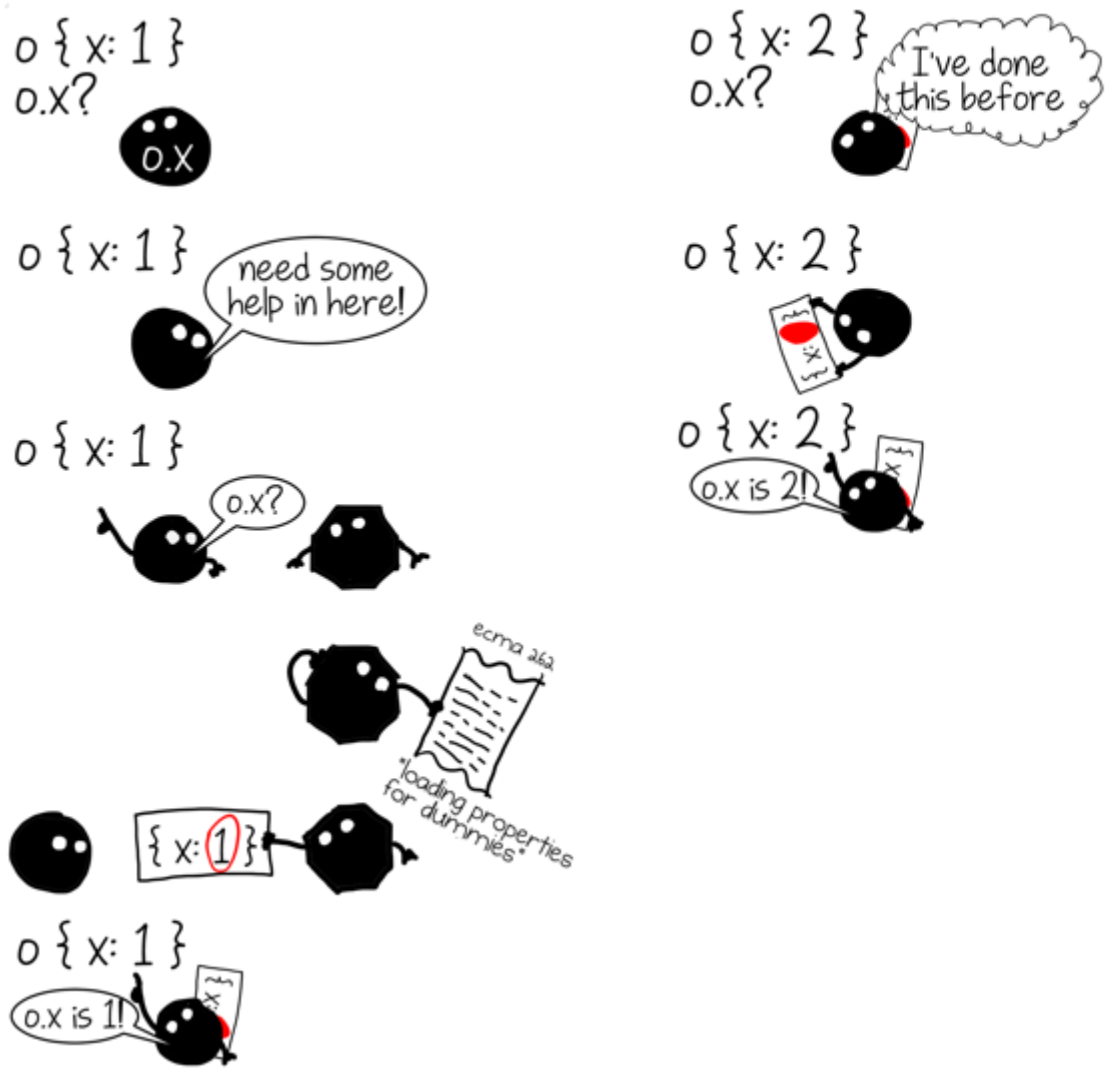
@farazsth98

Inline Caches

Inline Caches store information regarding property loads and stores.

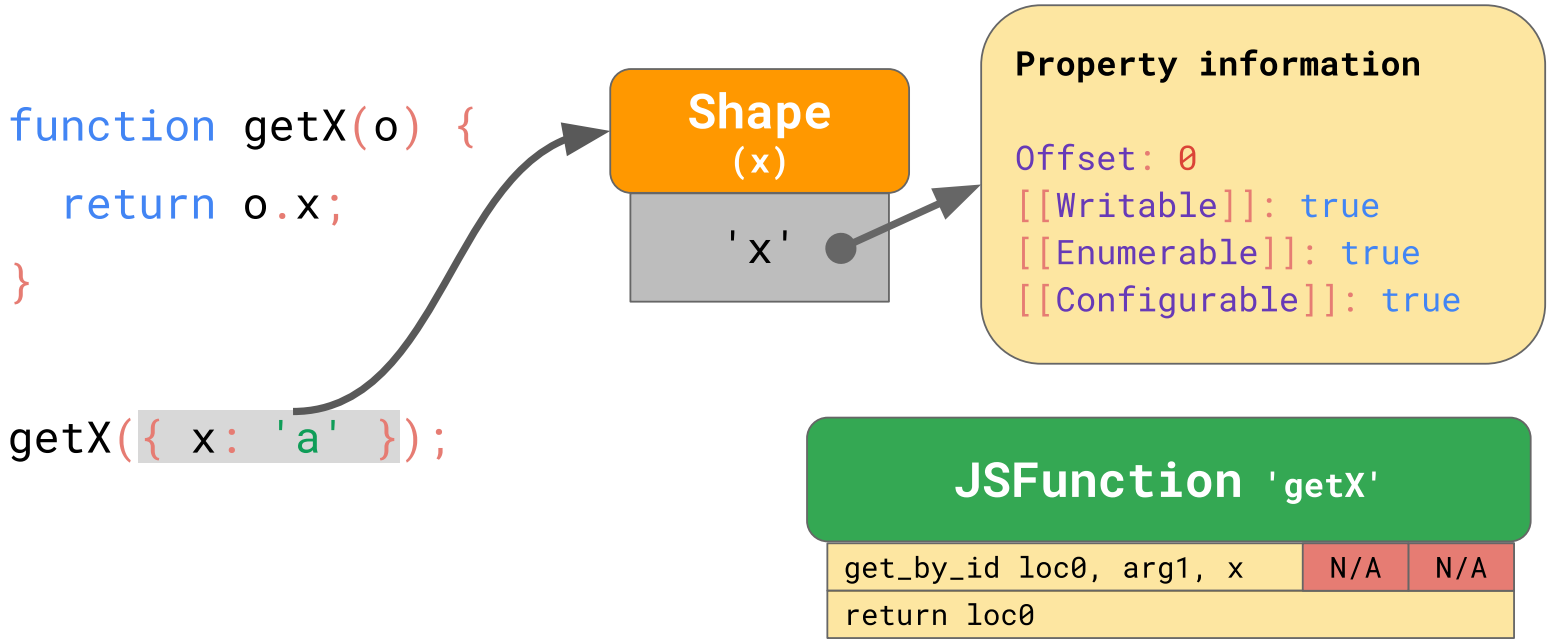
Any time an object's property is accessed within a function, the information regarding this access is stored in the function's **Inline Cache**.

When optimized machine code is generated, the code speculates that the same object's property is accessed, and uses the information stored in the **Inline Cache** to access the property.



@farazsth98

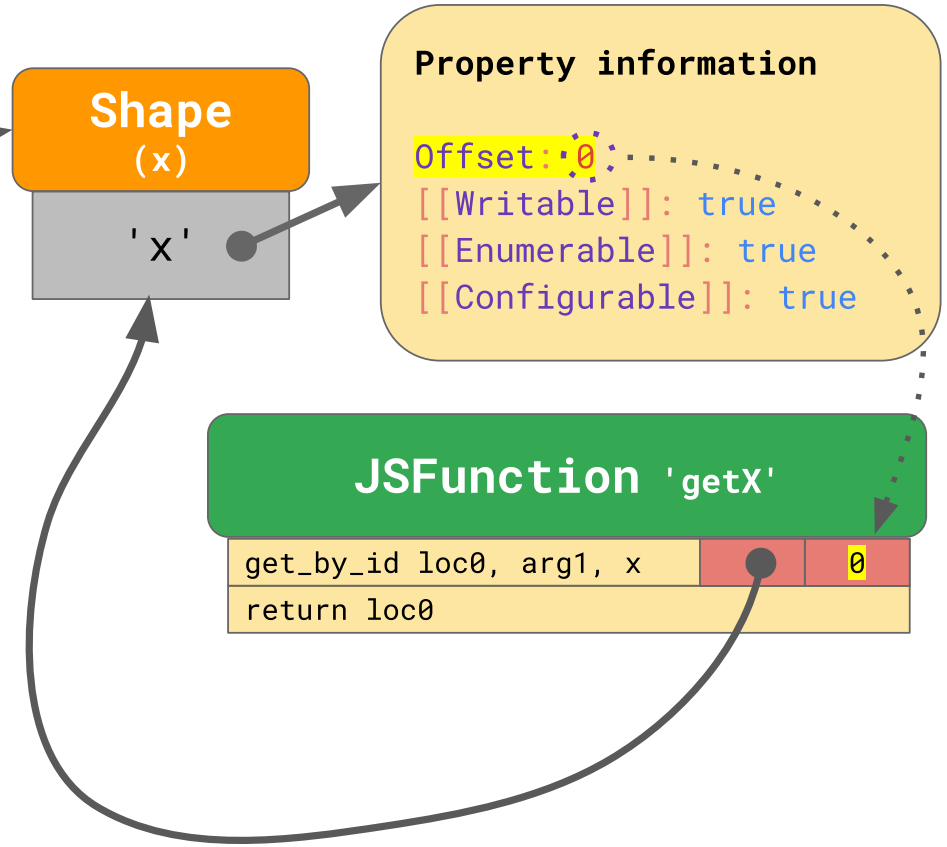
Inline Caches



https://mathiasbynens.be/notes/shapes-ics

Inline Caches

```
function getX(o) {
  return o.x;
}

getX({ x: 'a' });
```



https://mathiasbynens.be/notes/shapes-ics

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

Found by Stephen Röttger (@_tsuro) through manual code review, and by Samuel Groß (@5aelo) through fuzzing.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

Found by Stephen Röttger (@_tsuro) through manual code review, and by Samuel Groß (@5aelo) through fuzzing.

Great writeup by Stephen – Trashing The Flow of Data

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

Found by Stephen Röttger (@_tsuro) through manual code review, and by Samuel Groß (@5aelo) through fuzzing.

Great writeup by Stephen – Trashing The Flow of Data

Some more background information regarding **TurboFan** required.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

TurboFan will take the bytecode generated by **Ignition** and transform it into a “sea of nodes” graph.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

TurboFan will take the bytecode generated by **Ignition** and transform it into a “sea of nodes” graph.

The optimizations **TurboFan** performs are more easily done on the “sea of nodes” graph than on the AST.

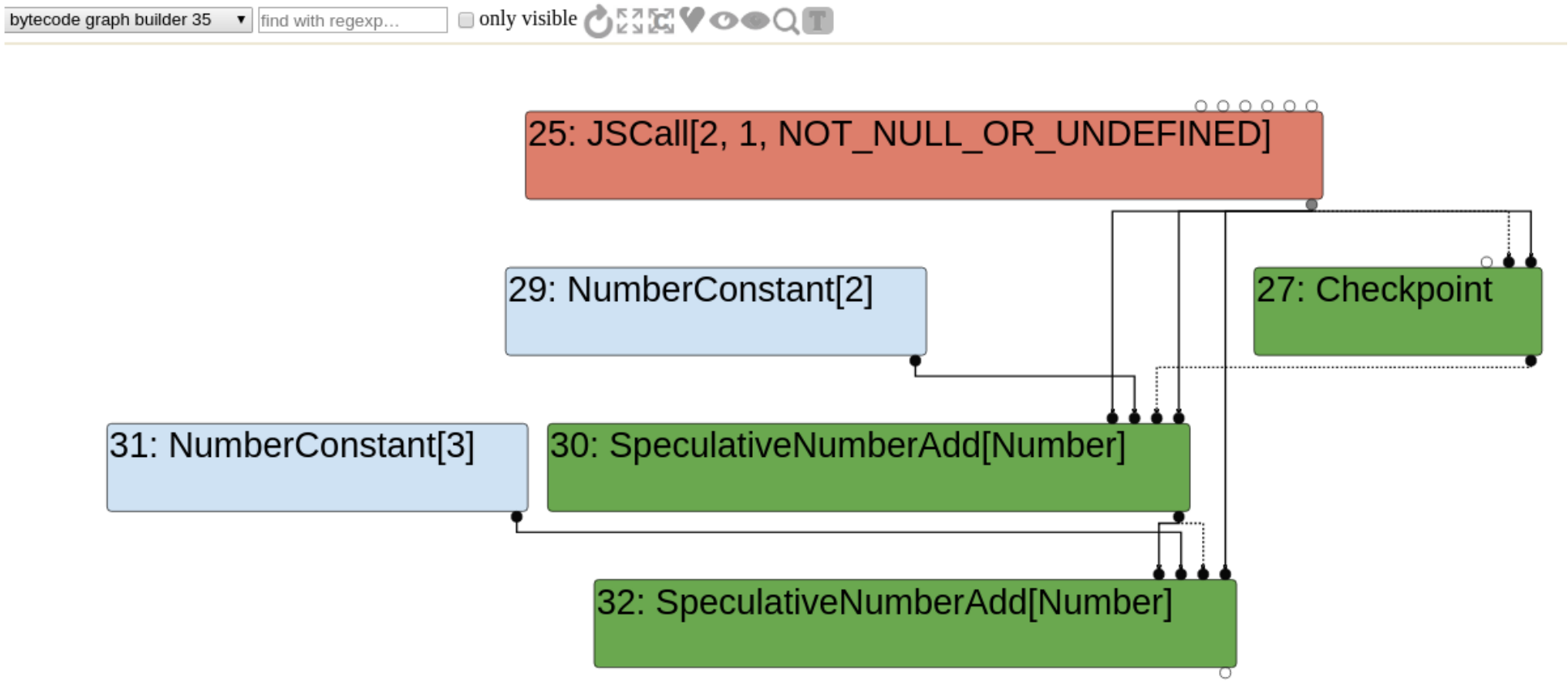
Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

TurboFan will take the bytecode generated by **Ignition** and transform it into a “sea of nodes” graph.

The optimizations **TurboFan** performs are more easily done on the “sea of nodes” graph than on the AST.

At the end, this “sea of nodes” graph is *lowered* down to machine code for a specific architecture.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`



@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

A node called the `CheckMaps` node can exist on this graph.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

A node called the `CheckMaps` node can exist on this graph.

When TurboFan generates optimized machine code, `CheckMaps` nodes are used to verify object maps before property accesses.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

A node called the `CheckMaps` node can exist on this graph.

When TurboFan generates optimized machine code, `CheckMaps` nodes are used to verify object maps before property accesses.

The `CheckMaps` node will cause the code to bail out to the interpreter if the map is not the one expected by TurboFan.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

A node called the `CheckMaps` node can exist on this graph.

When TurboFan generates optimized machine code, `CheckMaps` nodes are used to verify object maps before property accesses.

The `CheckMaps` node will cause the code to bail out to the interpreter if the map is not the one expected by TurboFan.

In a lot of cases, speculations made by TurboFan might cause it to eliminate `CheckMaps` nodes.

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

A node called the `CheckMaps` node can exist on this graph.

When TurboFan generates optimized machine code, `CheckMaps` nodes are used to verify object maps before property accesses.

The `CheckMaps` node will cause the code to bail out to the interpreter if the map is not the one expected by TurboFan.

In a lot of cases, speculations made by TurboFan might cause it to eliminate `CheckMaps` nodes.

If this is done incorrectly, *type confusions* might ensue between two objects that have different *Maps*.

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerable code was in `JSCallReducer::ReduceArrayIndexOfIncludes`.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerable code was in `JSCallReducer::ReduceArrayIndexOfIncludes`.

This function replaces the normal `Array.prototype.indexOf` and `Array.prototype.includes` with optimized machine code when it knows the *ElementsKind* of the array that the function is being called on.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerable code was in `JSCallReducer::ReduceArrayIndexOfIncludes`.

This function replaces the normal `Array.prototype.indexOf` and `Array.prototype.includes` with optimized machine code when it knows the *ElementsKind* of the array that the function is being called on.

The function first uses `InferReceiverMaps` to infer the map for the array.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerable code was in `JSCallReducer::ReduceArrayIndexOfIncludes`.

This function replaces the normal `Array.prototype.indexOf` and `Array.prototype.includes` with optimized machine code when it knows the *ElementsKind* of the array that the function is being called on.

The function first uses `InferReceiverMaps` to infer the map for the array.

At the end, the function bails out to the interpreter if the map is not known.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerable code was in `JSCallReducer::ReduceArrayIndexOfIncludes`.

This function replaces the normal `Array.prototype.indexOf` and `Array.prototype.includes` with optimized machine code when it knows the *ElementsKind* of the array that the function is being called on.

The function first uses `InferReceiverMaps` to infer the map for the array.

At the end, the function bails out to the interpreter if the map is not known.

Otherwise, it continues on and optimizes the function.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

This seems correct initially, until you realise that `InferReceiverMaps` actually has three possible return values.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

```
enum InferReceiverMapsResult {
kNoReceiverMaps,         // No receiver maps inferred.
kReliableReceiverMaps,   // Receiver maps can be trusted.
kUnreliableReceiverMaps  // Receiver maps might have changed (side-
effect).
};
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

This seems correct initially, until you realise that `InferReceiverMaps` actually has three possible return values.

`kUnreliableReceiverMaps` tells the compiler that although the map has been inferred, it is unreliable as it could have possibly been changed by side-effects of other operations.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

```
enum InferReceiverMapsResult {
kNoReceiverMaps,         // No receiver maps inferred.
kReliableReceiverMaps,   // Receiver maps can be trusted.
kUnreliableReceiverMaps  // Receiver maps might have changed (side-
effect).
};
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

This seems correct initially, until you realise that `InferReceiverMaps` actually has three possible return values.

`kUnreliableReceiverMaps` tells the compiler that although the map has been inferred, it is unreliable as it could have possibly been changed by side-effects of other operations.

If this happens, the caller *must* guard against possible map changes with a runtime check. One way to do this is by inserting a `CheckMaps` node into the graph.

```
Reduction JSCallReducer::ReduceArrayIndexOfIncludes(
  SearchVariant search_variant, Node* node) {
CallParameters const& p = CallParametersOf(node->op());
if (p.speculation_mode() == SpeculationMode::kDisallowSpeculation)
{
  return NoChange();
}

Node* receiver = NodeProperties::GetValueInput(node, 1);
Node* effect = NodeProperties::GetEffectInput(node);
Node* control = NodeProperties::GetControlInput(node);

ZoneHandleSet<Map> receiver_maps;
NodeProperties::InferReceiverMapsResult result =
    NodeProperties::InferReceiverMaps(broker(), receiver, effect,
                                      &receiver_maps);
if (result == NodeProperties::kNoReceiverMaps) return NoChange();

[...] // Continue with optimizing the function
```

```
enum InferReceiverMapsResult {
kNoReceiverMaps,         // No receiver maps inferred.
kReliableReceiverMaps,   // Receiver maps can be trusted.
kUnreliableReceiverMaps  // Receiver maps might have changed (side-
effect).
};
```

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerability occurs because the function fails to account for the case where the map that is inferred is unreliable.

Proof of concept:

```
function f(idx, arr) {
  // Transition to dictionary mode in the final invocation.
  arr.__defineSetter__(idx, ()=>{});
  // Will then read OOB.
  return arr.includes(1234);
}

f('', []);
f('', []);
%OptimizeFunctionOnNextCall(f);
f('1000000', []);
```

`__defineSetter__`, when called on an index, will immediately change the array's *ElementsKind* to `DICTIONARY_ELEMENTS`.

`%OptimizeFunctionOnNextCall` is a V8 native built-in function that causes TurboFan to compile the function that's called immediately after.

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerability occurs because the function fails to account for the case where the map that is inferred is unreliable.

Proof of concept:

```
function f(idx, arr) {
  // Transition to dictionary mode in the final invocation.
  arr.__defineSetter__(idx, ()=>{});
  // Will then read OOB.
  return arr.includes(1234);
}

f('', []);
f('', []);
%OptimizeFunctionOnNextCall(f);
f('1000000', []);
```

`__defineSetter__`, when called on an index, will immediately change the array's *ElementsKind* to `DICTIONARY_ELEMENTS`.

`%OptimizeFunctionOnNextCall` is a V8 native built-in function that causes TurboFan to compile the function that's called immediately after.

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerability occurs because the function fails to account for the case where the map that is inferred is unreliable.

Proof of concept:

```
function f(idx, arr) {
  // Transition to dictionary mode in the final invocation.
  arr.__defineSetter__(idx, ()=>{});
  // Will then read OOB.
  return arr.includes(1234);
}

f('', []);
f('', []);
%OptimizeFunctionOnNextCall(f);
f('1000000', []);
```

`__defineSetter__`, when called on an index, will immediately change the array's *ElementsKind* to `DICTIONARY_ELEMENTS`.

`%OptimizeFunctionOnNextCall` is a V8 native built-in function that causes TurboFan to compile the function that's called immediately after.

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The vulnerability occurs because the function fails to account for the case where the map that is inferred is unreliable.

Proof of concept:

```
function f(idx, arr) {
  // Transition to dictionary mode in the final invocation.
  arr.__defineSetter__(idx, ()=>{});
  // Will then read OOB.
  return arr.includes(1234);
}

f('', []);
f('', []);
%OptimizeFunctionOnNextCall(f);
f('1000000', []);
```

`__defineSetter__`, when called on an index, will immediately change the array's *ElementsKind* to `DICTIONARY_ELEMENTS`.

`%OptimizeFunctionOnNextCall` is a V8 native built-in function that causes TurboFan to compile the function that's called immediately after.

@farazsth98

Chromium Issue 944062 – Failing to insert Map checks in `Array.indexOf` and `Array.includes`

The patch:

```
+  // If we have unreliable maps, we need a map check.
+  if (result == NodeProperties::kUnreliableReceiverMaps) {
+    effect =
+        graph()->NewNode(simplified()->CheckMaps(CheckMapsFlag::kNone,
+                                                 receiver_maps, p.feedback()),
+                         receiver, effect, control);
+  }
+
```

Checks for `kUnreliableReceiverMaps` and inserts a `CheckMaps` node into the sea of nodes graph.

@farazsth98

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

Try your hand at an easy CTF challenge.

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

Try your hand at an easy CTF challenge.

Pick a *very* detailed writeup / bug report and analyse it.

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

Try your hand at an easy CTF challenge.

Pick a *very* detailed writeup / bug report and analyse it.

Slowly move onto lesser and lesser detailed bug reports.

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

Try your hand at an easy CTF challenge.

Pick a *very* detailed writeup / bug report and analyse it.

Slowly move onto lesser and lesser detailed bug reports.

Generalize older bugs into bug classes and look for similar ones.

How do you start on V8?

Watch some talks (like this one!) that explain some of the internals of V8.

Try your hand at an easy CTF challenge.

Pick a *very* detailed writeup / bug report and analyse it.

Slowly move onto lesser and lesser detailed bug reports.

Generalize older bugs into bug classes and look for similar ones.

Fuzz everything!

Useful links

https://chromereleases.googleblog.com/

https://bugs.chromium.org/p/chromium/issues/list

- `Security_Severity=High`
- `Security_Severity=Critical`
- `Component:Blink>Javascript`

https://source.chromium.org

Thanks to Thugcrowd for hosting this event!

Feel free to @ me on the discord and ask any questions

@farazsth98