Patterns of VM Design

Patterns of VM Design

me@mrale.ph

@mraleph

V8

Dart VM

Excelsior JET

V8

Dart VM

Excelsior JET

V8

Dart VM

```
Vector.prototype.length = function () {
  return Math.sqrt(this.x * this.x +
                   this.y * this.y);
};
```

representation
resolution
redundancy


fast ⊕ slow


fast ⊆ slow

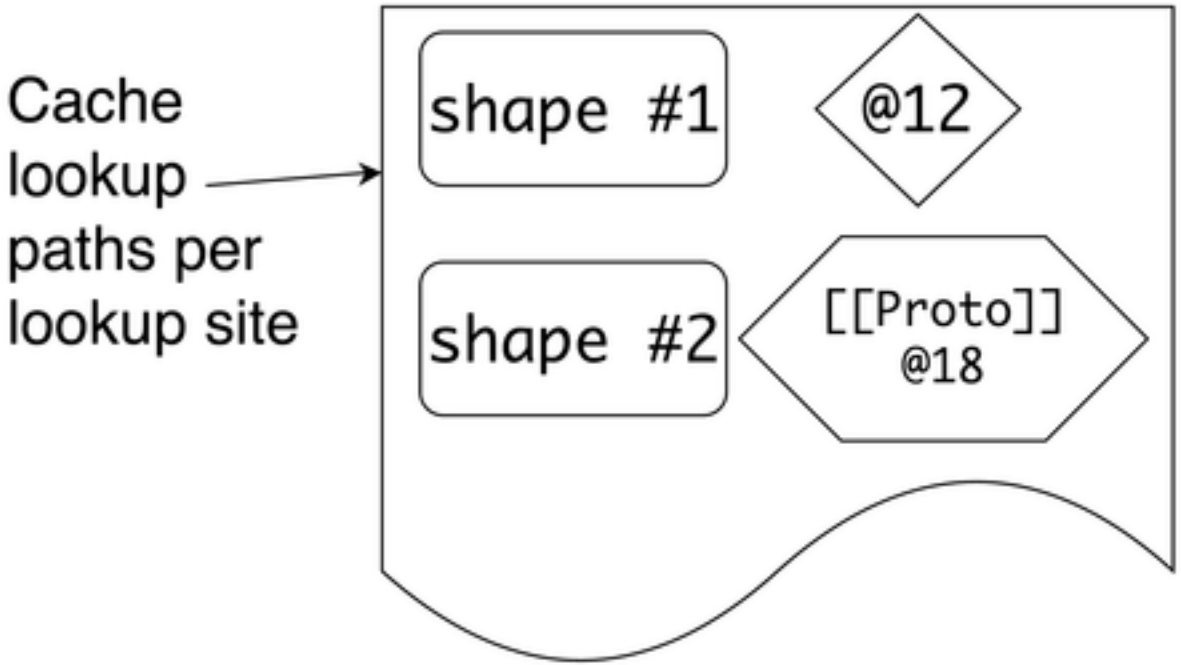
obj.prop

"guess where property is"

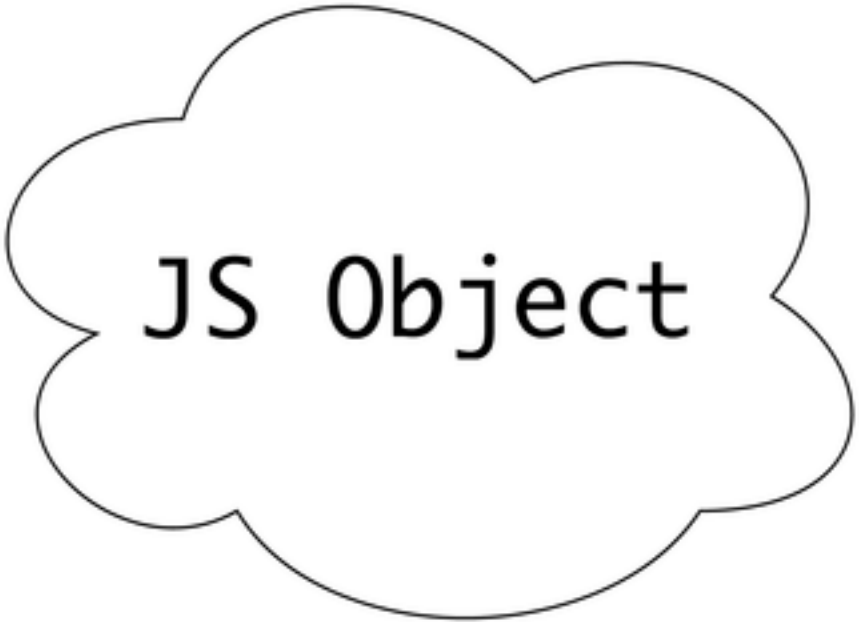
"remember where property was"

inline caching

obj.prop

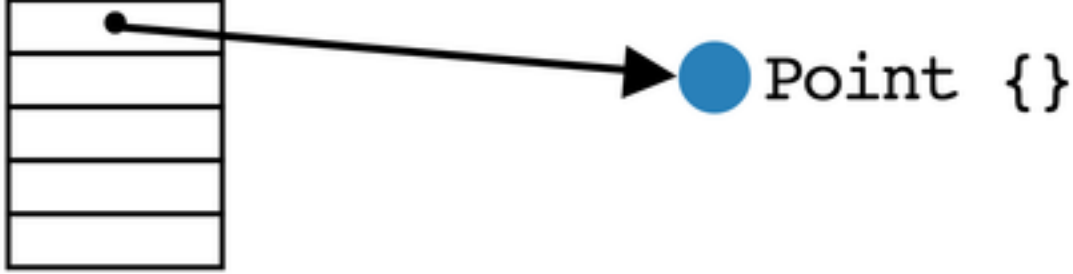

Cache
lookup
paths per
lookup site
shape #1
@12
shape #2
[[Proto]]
@18

"Shape?!"

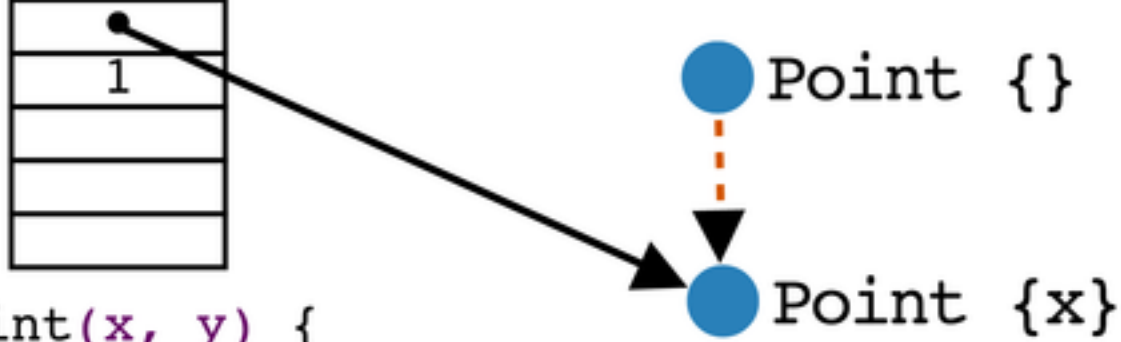

JS Object

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

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

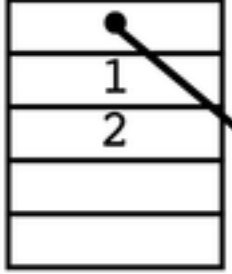
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

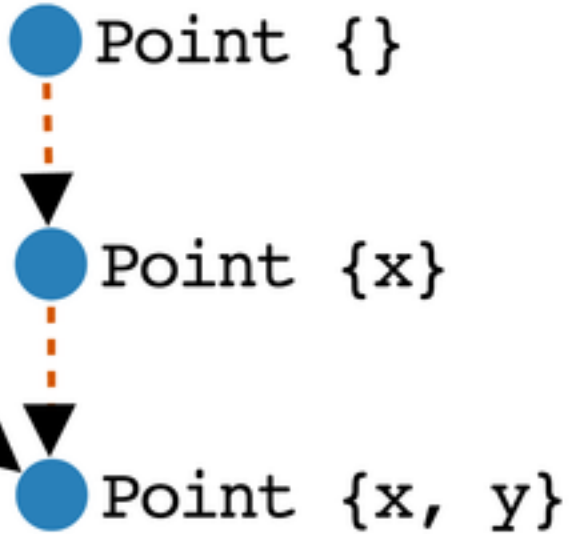



```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

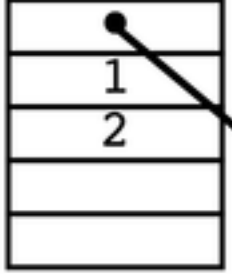


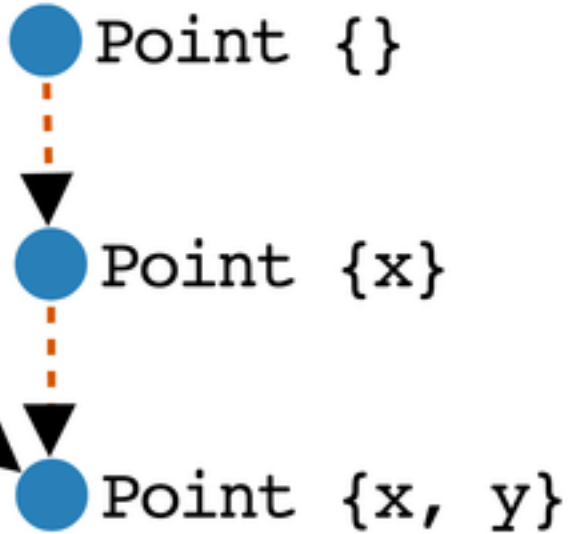

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```



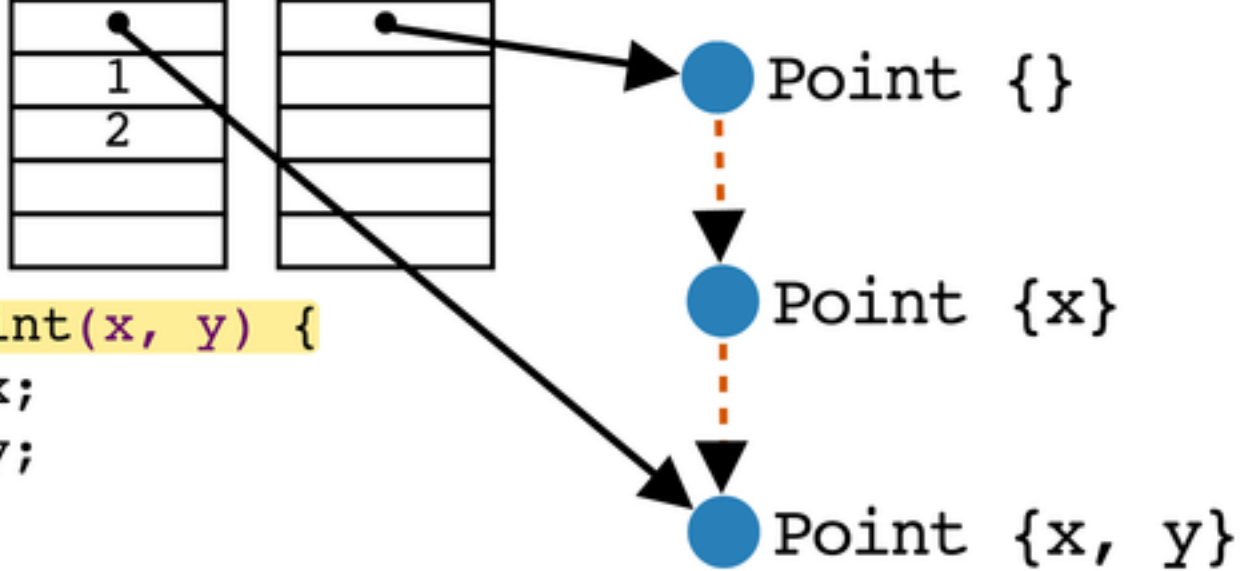



```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

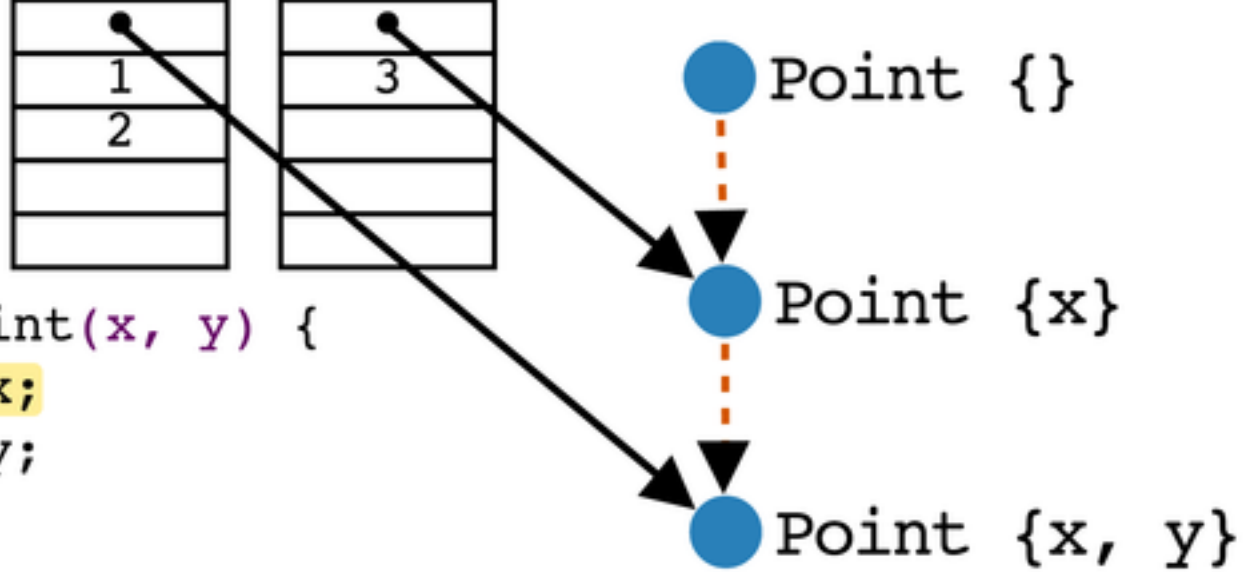




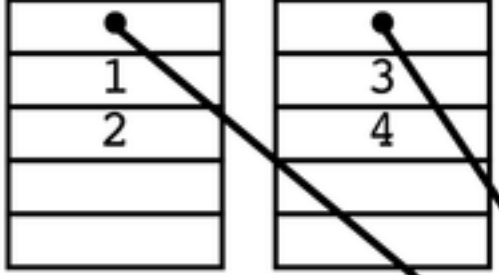

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

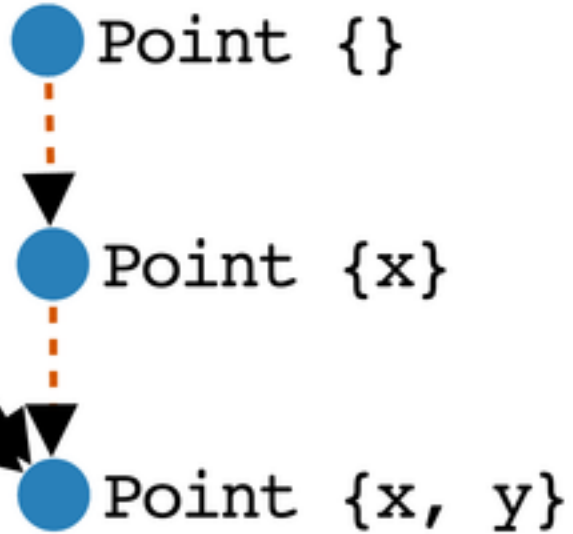



```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

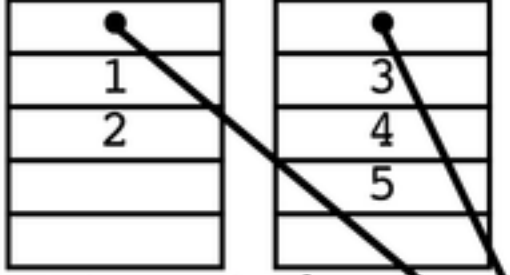



```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```

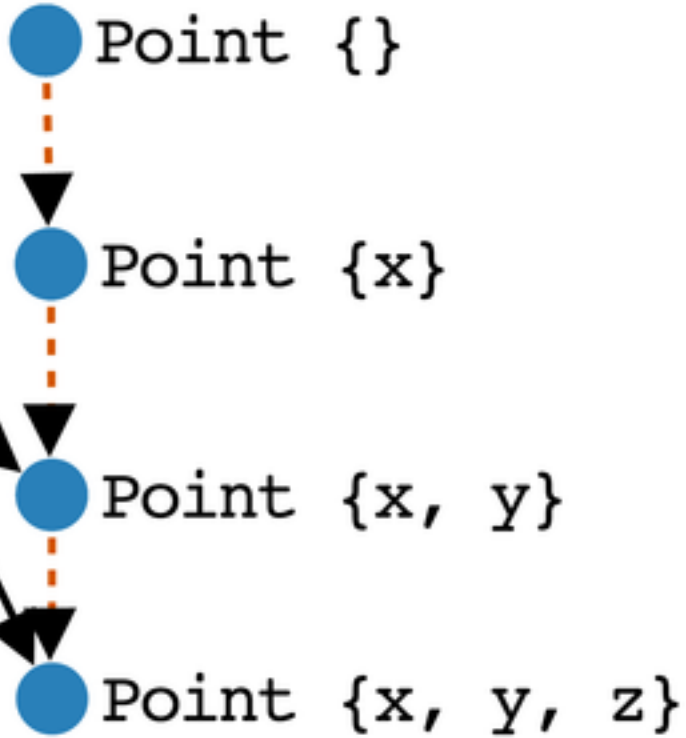





```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
```




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

var p1 = new Point(1, 2);
var p2 = new Point(3, 4);
p2.z = 5;
```



V8 hidden classes

≈ *maps* from Self VM

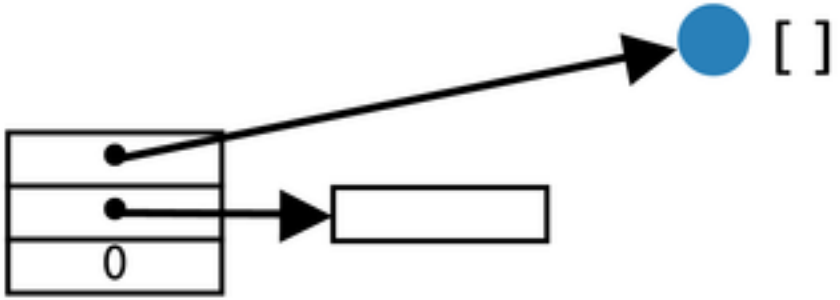
static structure

approximated

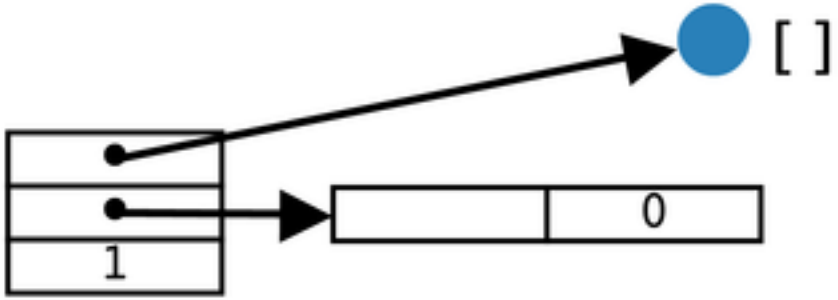
dynamically

```
var arr = [];
for (var i = 0; i < 101; i++)
  arr[i] = Math.sqrt(i);
```

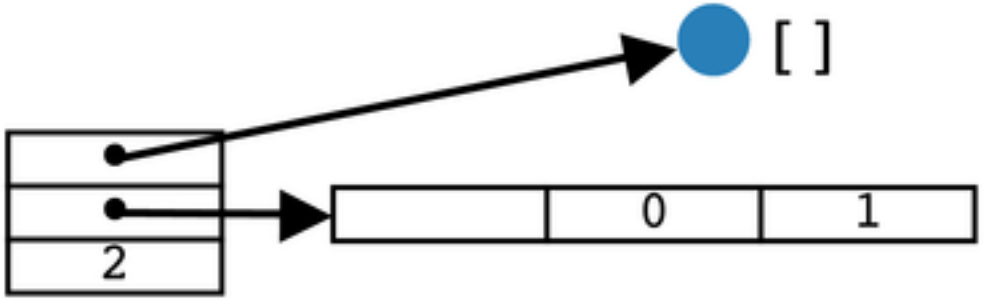
```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



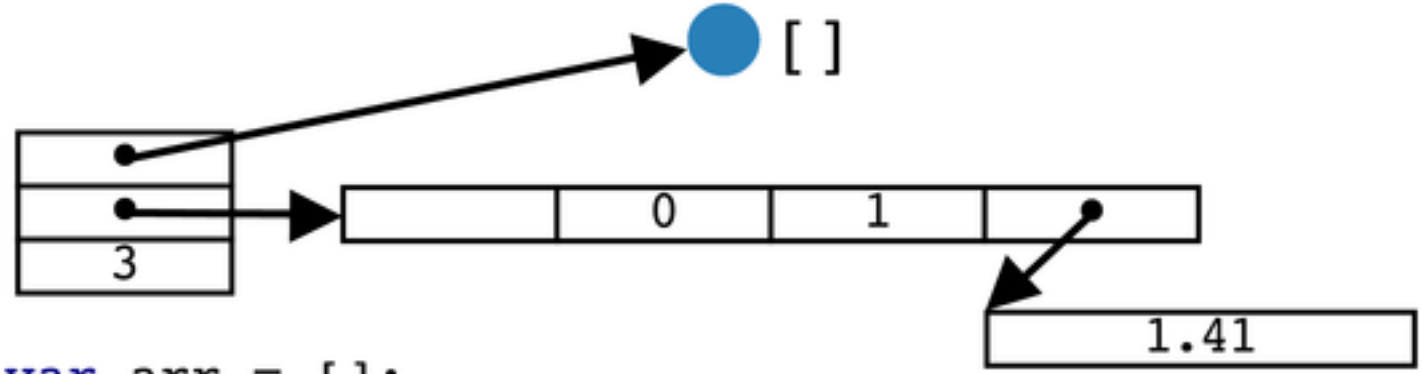

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



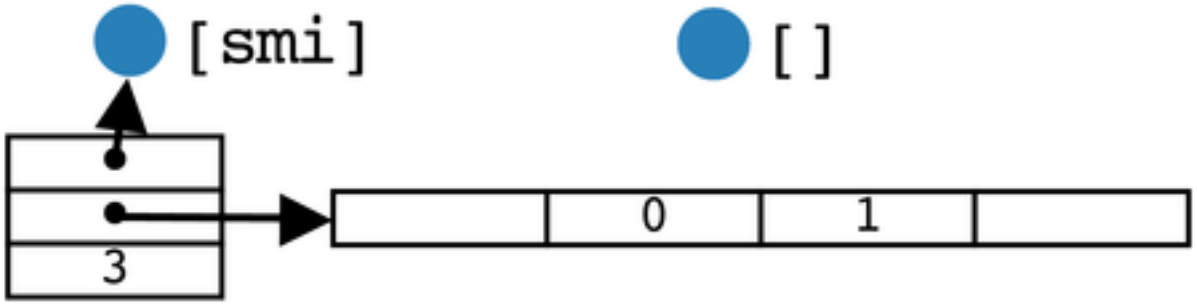

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



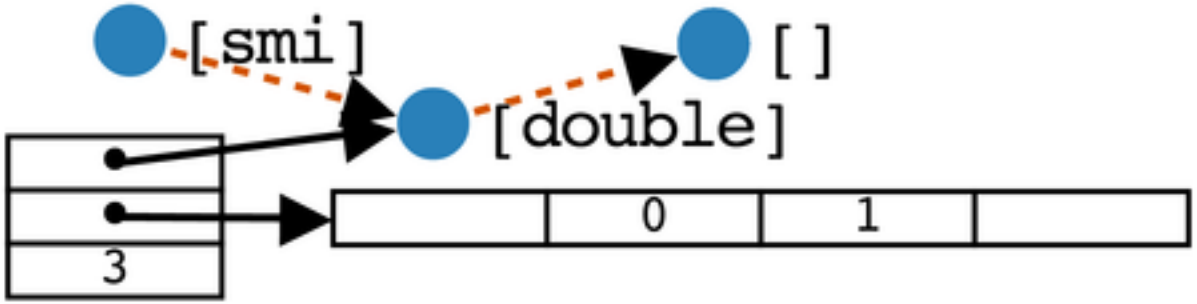

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



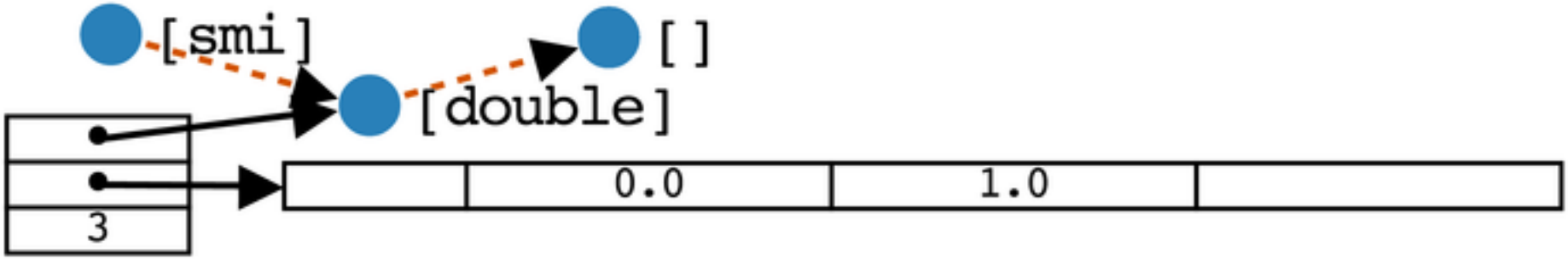

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



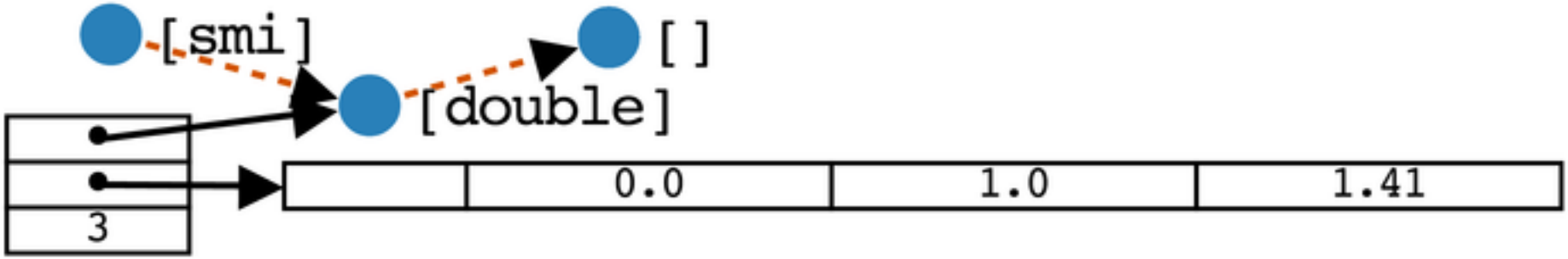

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



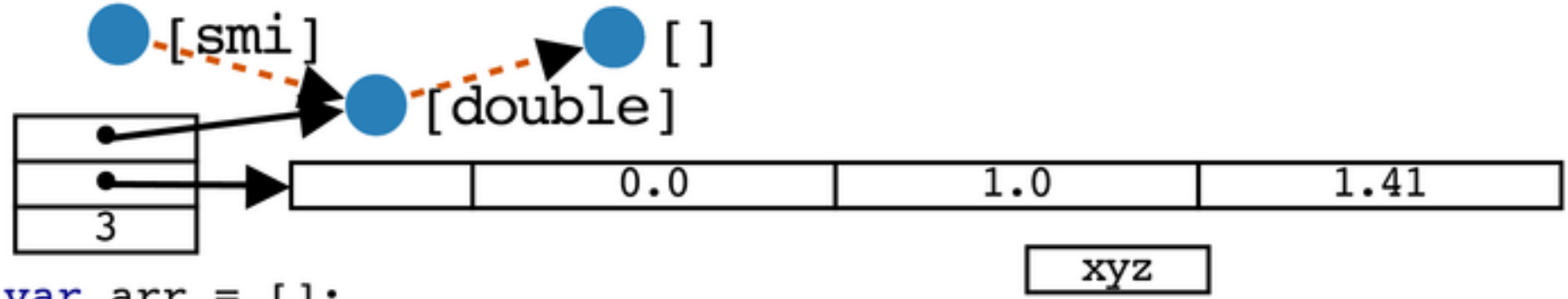

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



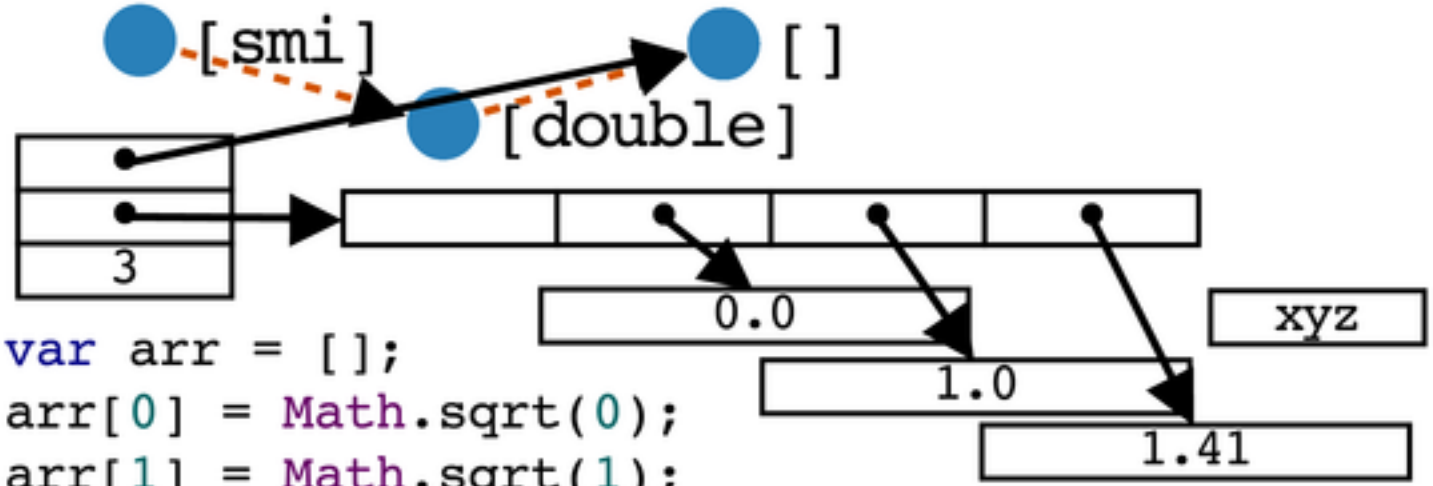

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```



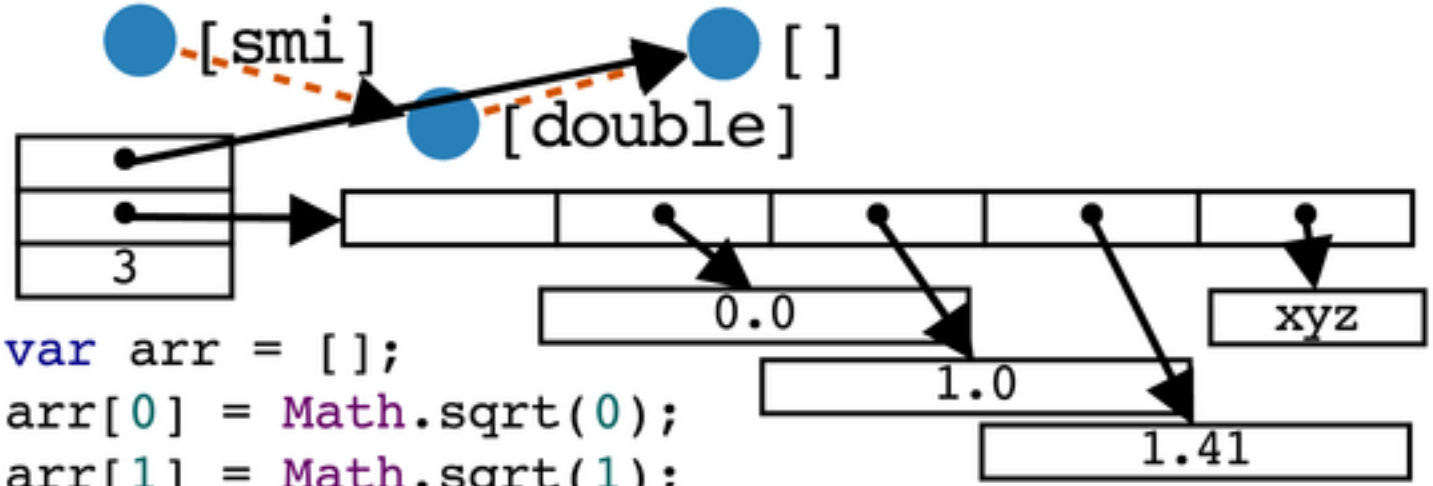

```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
```




```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
arr[3] = "xyz";
```

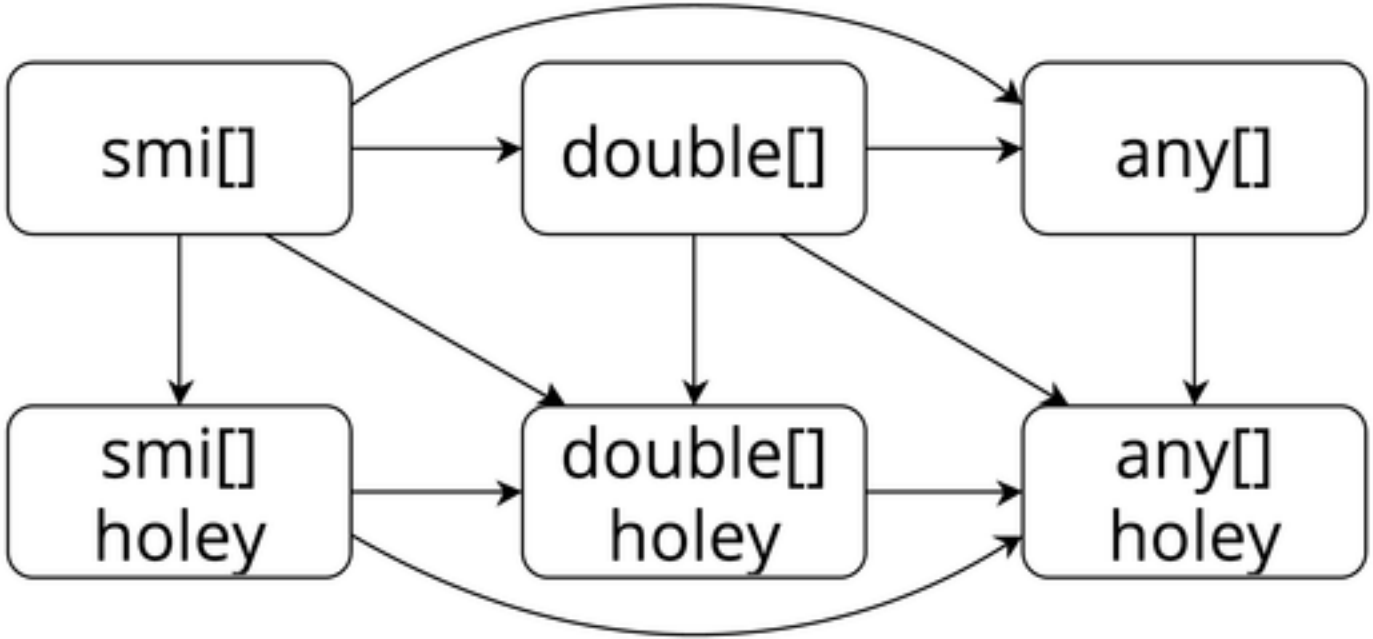



```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
arr[3] = "xyz";
```

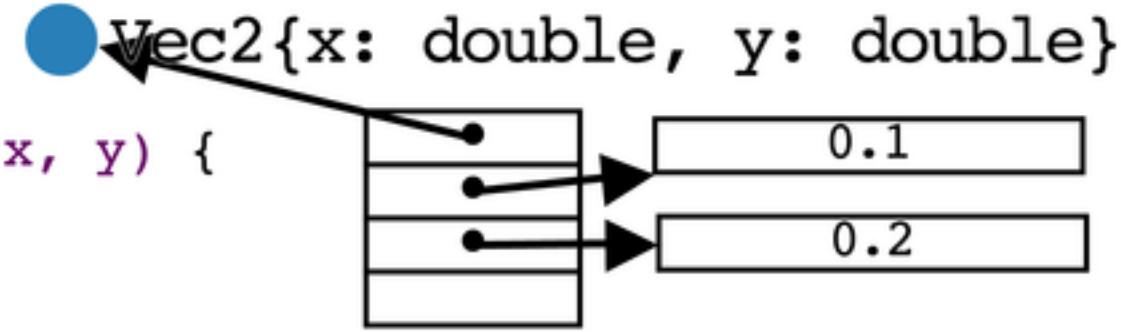



```
var arr = [];
arr[0] = Math.sqrt(0);
arr[1] = Math.sqrt(1);
arr[2] = Math.sqrt(2);
arr[3] = "xyz";
```

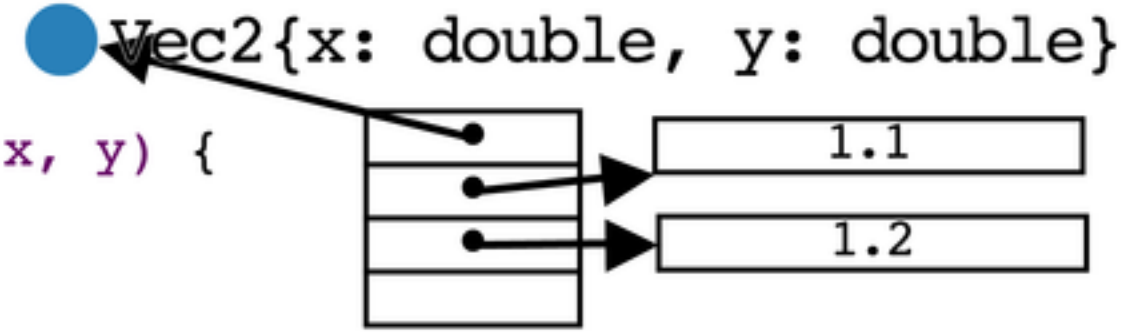
tracking denseness and unboxing


smi[]
double[]
any[]
smi[]
holey
double[]
holey
any[]
holey

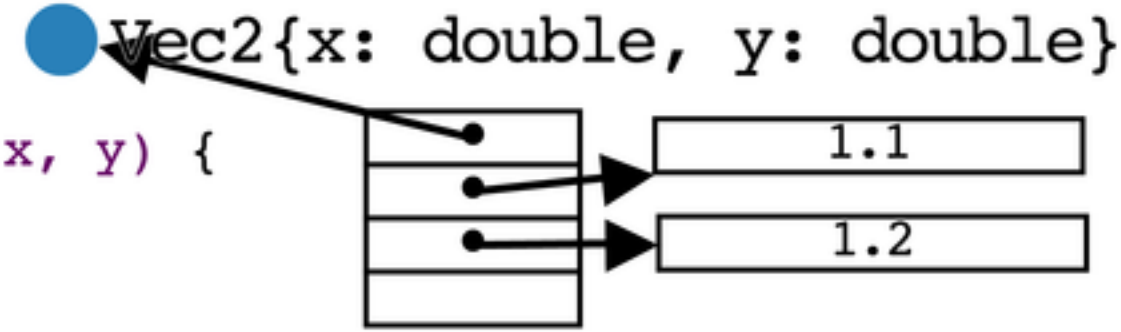
```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2);
v.x += 1;
v.y += 1;
```



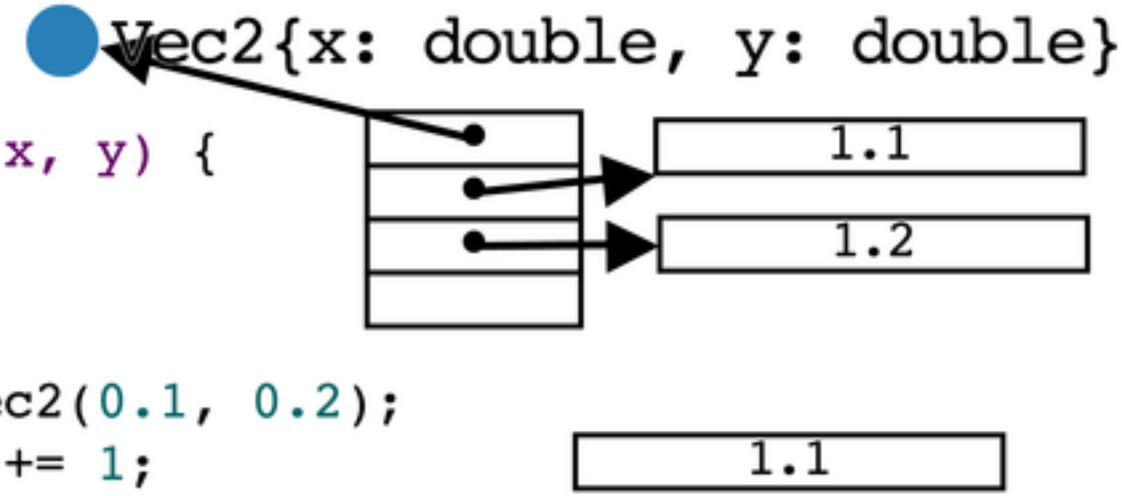

```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2);
v.x += 1;
v.y += 1;
```




```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2);
v.x += 1;
v.y += 1;
```

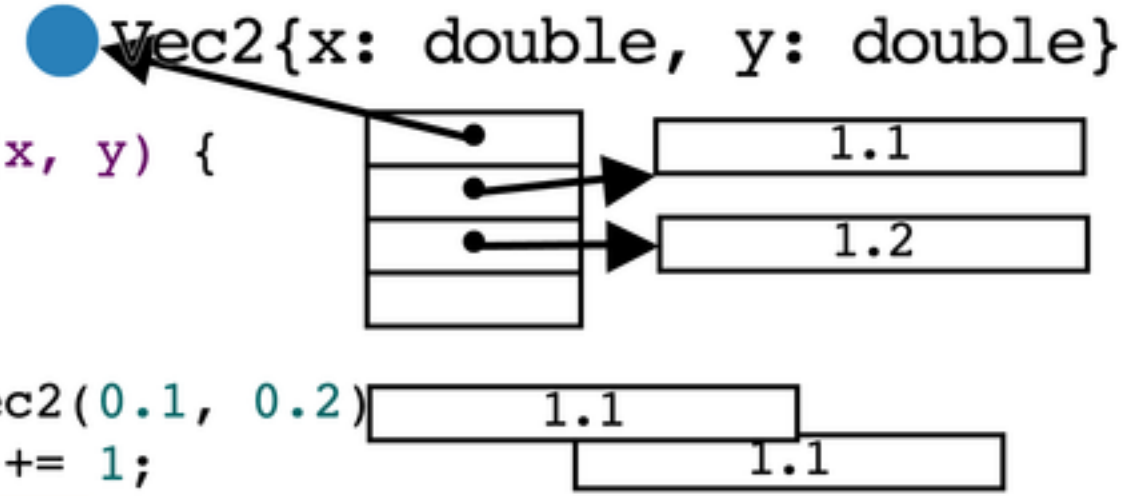



```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2);
v.x += 1; v.y += 1;
while (true) v.x;
```

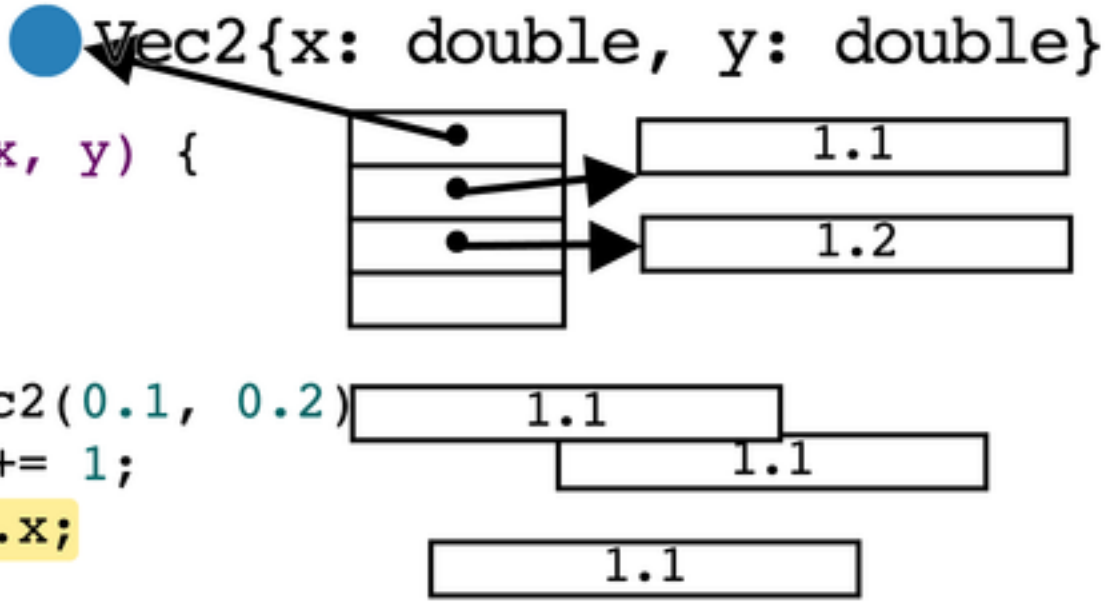



```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2);
v.x += 1; v.y += 1;
while (true) v.x;
```

```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2)
v.x += 1; v.y += 1;
while (true) v.x;
```



```
function Vec2(x, y) {
  this.x = x;
  this.y = y;
}
var v = new Vec2(0.1, 0.2)
v.x += 1; v.y += 1;
while (true) v.x;
```

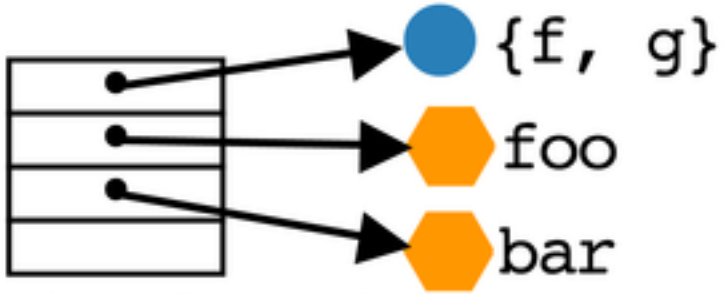


mutable boxes

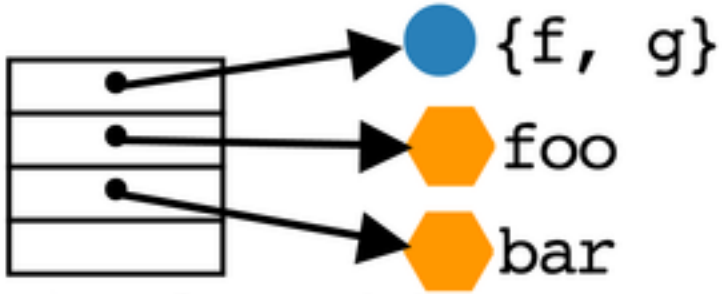
beneficial if you can read unboxed

```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
```

```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
// How hidden class of K.prototype looks like?
```

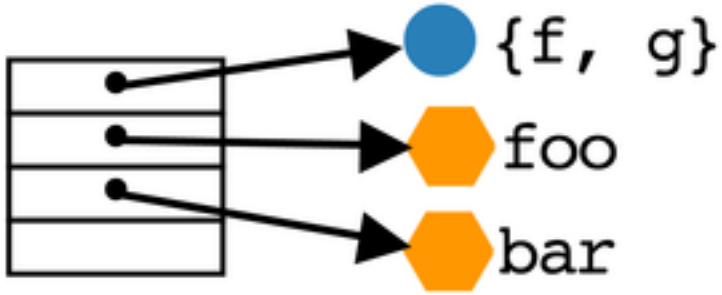



```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
// How hidden class of K.prototype looks like?
```

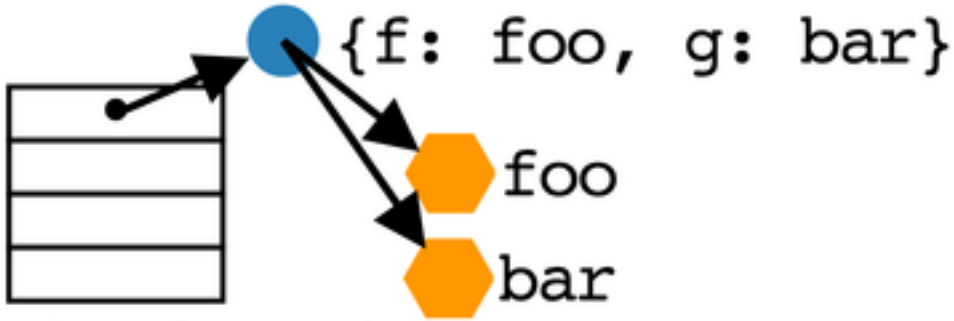



```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
// Want it to be more 'class'-like
```

promote functions to hidden class




```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
```




```
function K() { }
K.prototype.f = function foo() { };
K.prototype.g = function bar() { };
// Now it's more like vtbl!
```

"amazing technology!!1"

"amazing technology!!1"

has issues

"if there is a metaspace there is a metaGC problem"

it's still a heuristic

```
var o1 = {};
o1.f = function () { };
var o2 = {};
o2.f = function () { };

for (var i = 0; i < 1e7; i++) o1.f();
for (var i = 0; i < 1e7; i++) o2.f();
```

same hidden class

$\Rightarrow$ same structure

inline caching

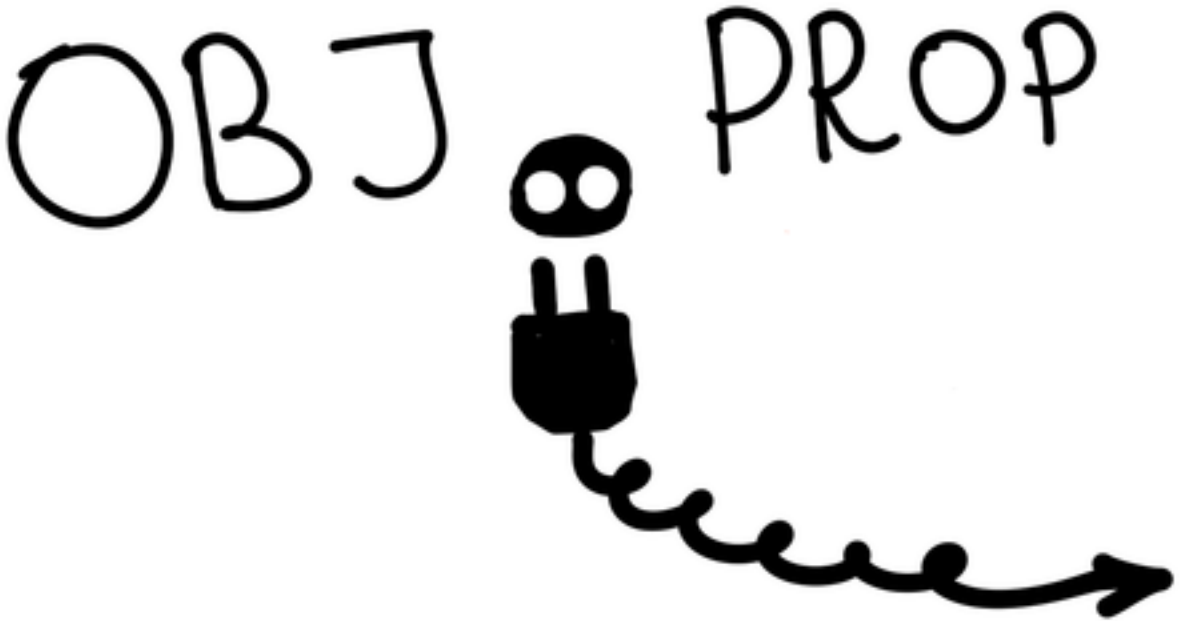
In V8

In V8

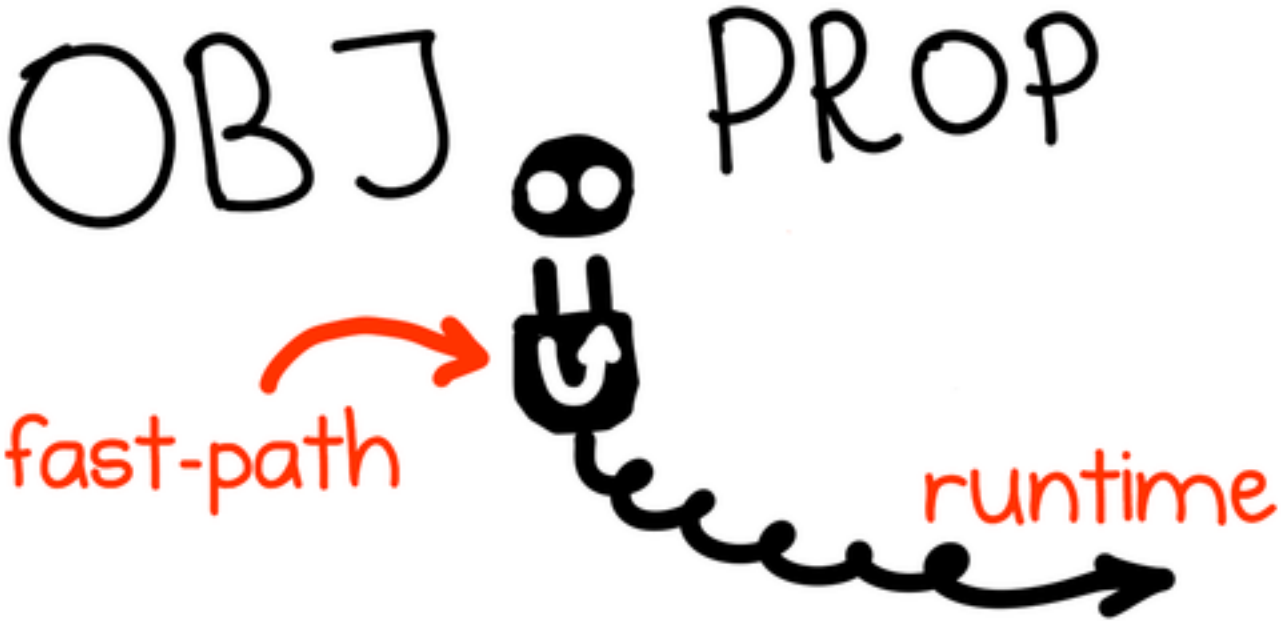
In V8 (American Version)

In V8

In V8



In V8



```
;; Compiled code
mov eax, obj
mov ecx, "foo"
call LoadIC_Initialize
```

```
// Runtime system.
function LoadIC_Initialize(obj, prop) {
  var lookupResult = obj.lookup(prop);
  patch(lookupResult.compile());
  return lookupResult.value;
}
```

```
// Runtime system.
function LoadIC_Initialize(obj, prop) {
  var lookupResult = obj.lookup(prop);
  patch(lookupResult.compile());
  return lookupResult.value;
}
```

```
;; Compiled LoadIC Stub
0xabcdef:
cmp [eax - 1], klass0
jnz LoadIC_Miss
mov eax, [eax + 11]
ret

;; Compiled code
mov eax, obj
mov ecx, "foo"
call 0xabcdef ;; patched!
```

```
;; Compiled LoadIC Stub
0xabcdef:
cmp [eax - 1], klass0
jnz LoadIC_Miss
mov eax, [eax + 11]
ret

;; Compiled code
mov eax, obj
mov ecx, "foo"
call 0xabcdef ;; patched!
```

```
;; Compiled LoadIC Stub
0xabcdef:
cmp [eax - 1], klass0
jnz LoadIC_Miss
mov eax, [eax + 11]
ret

;; Compiled code
mov eax, obj
mov ecx, "foo"
call 0xabcdef ;; patched!
```

```
;; Compiled LoadIC Stub
0xabcdef:
cmp [eax - 1], klass0
jnz LoadIC_Miss
mov eax, [eax + 11]
ret

;; Compiled code
mov eax, obj
mov ecx, "foo"
call 0xabcdef ;; patched!
```

```
;; Compiled LoadIC Stub
0xabcdef:
cmp [eax - 1], klass0
jnz LoadIC_Miss
mov eax, [eax + 11]
ret

;; Compiled code
mov eax, obj
mov ecx, "foo"
call 0xabcdef ;; patched!
```

Everything is an IC stub

- property accesses
- element accesses
- method calls
- special method calls
- global variables lookup
- arithmetic operations

In V8

Dart?

Dart?

~~code patching~~

```
function ICStub($data, receiver, ...) {
  var target = $data.get(receiver.klass)
  if (target === null) {
    target = HandleMiss($data, receiver, ...)
  }
  return target(receiver, ...)
}
```

fields access?

everything is a call

`o.x`	`o.get:x()`
`o.x = v`	`o.set:x(v)`

everything is a call

`o.x`	`o.get:x()`
`o.x = v`	`o.set:x(v)`

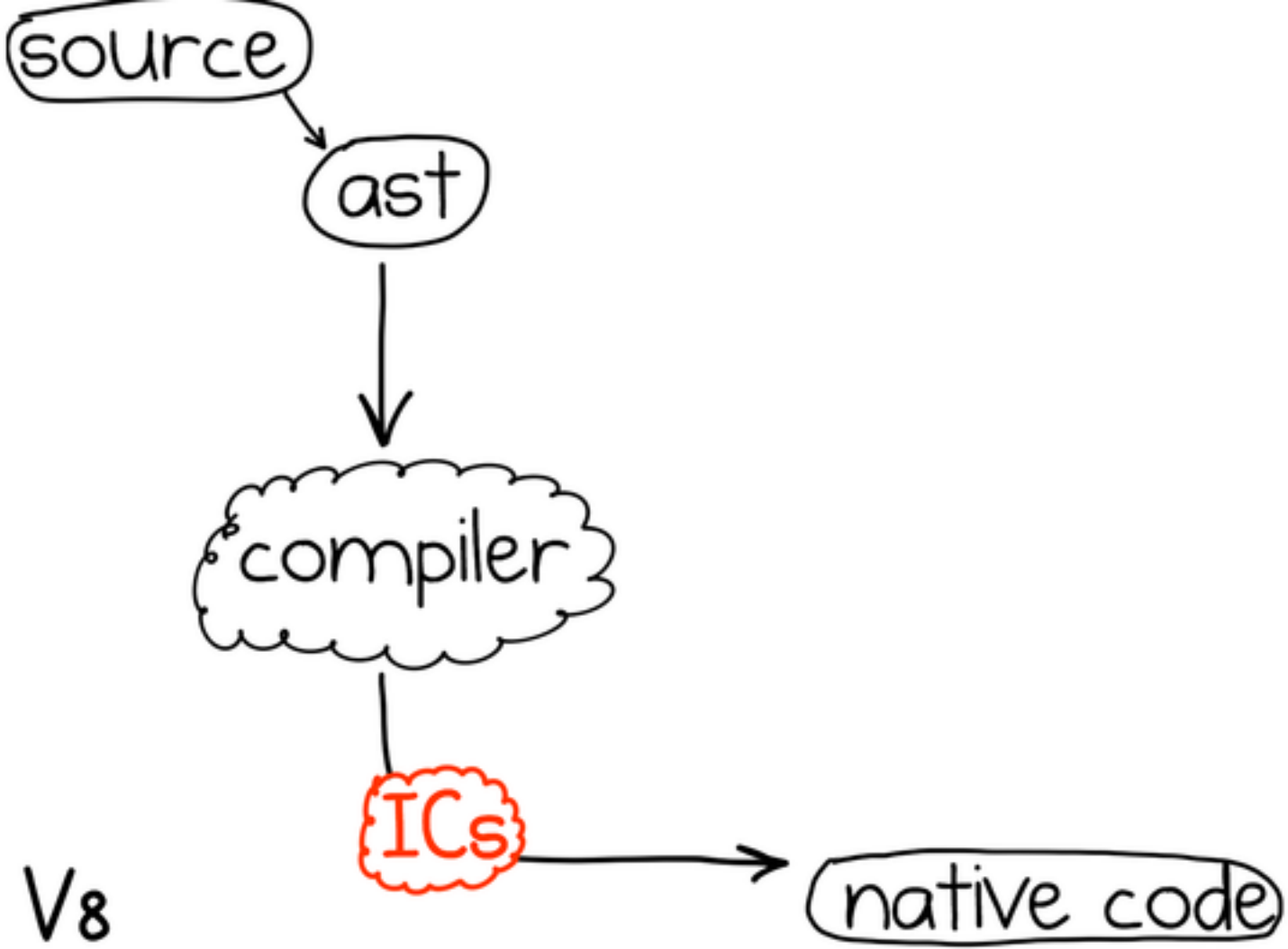
uniform = good.

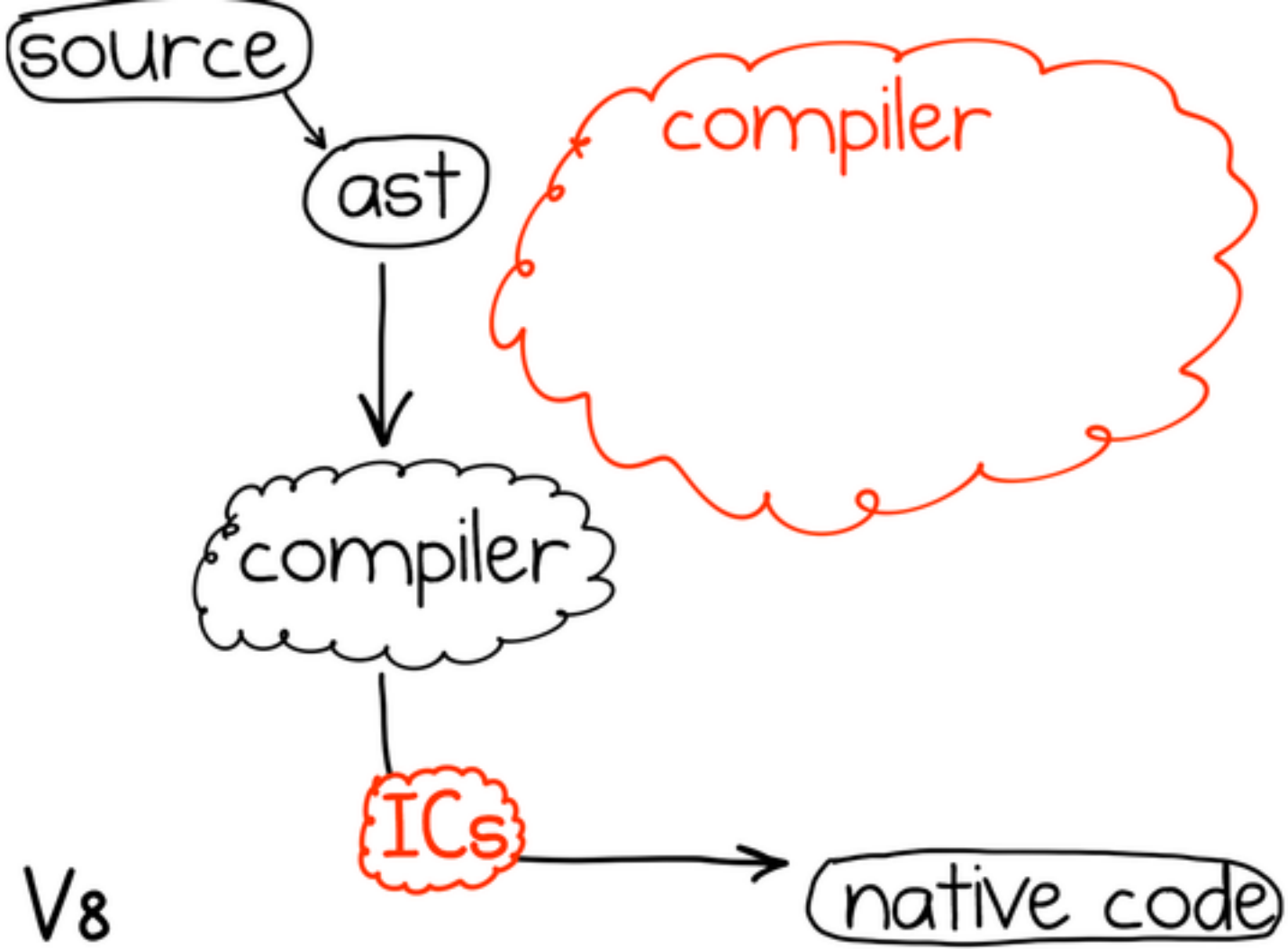
```
Vector.prototype.length = function () {
  return Math.sqrt(this.x * this.x +
                   this.y * this.y);
};
```

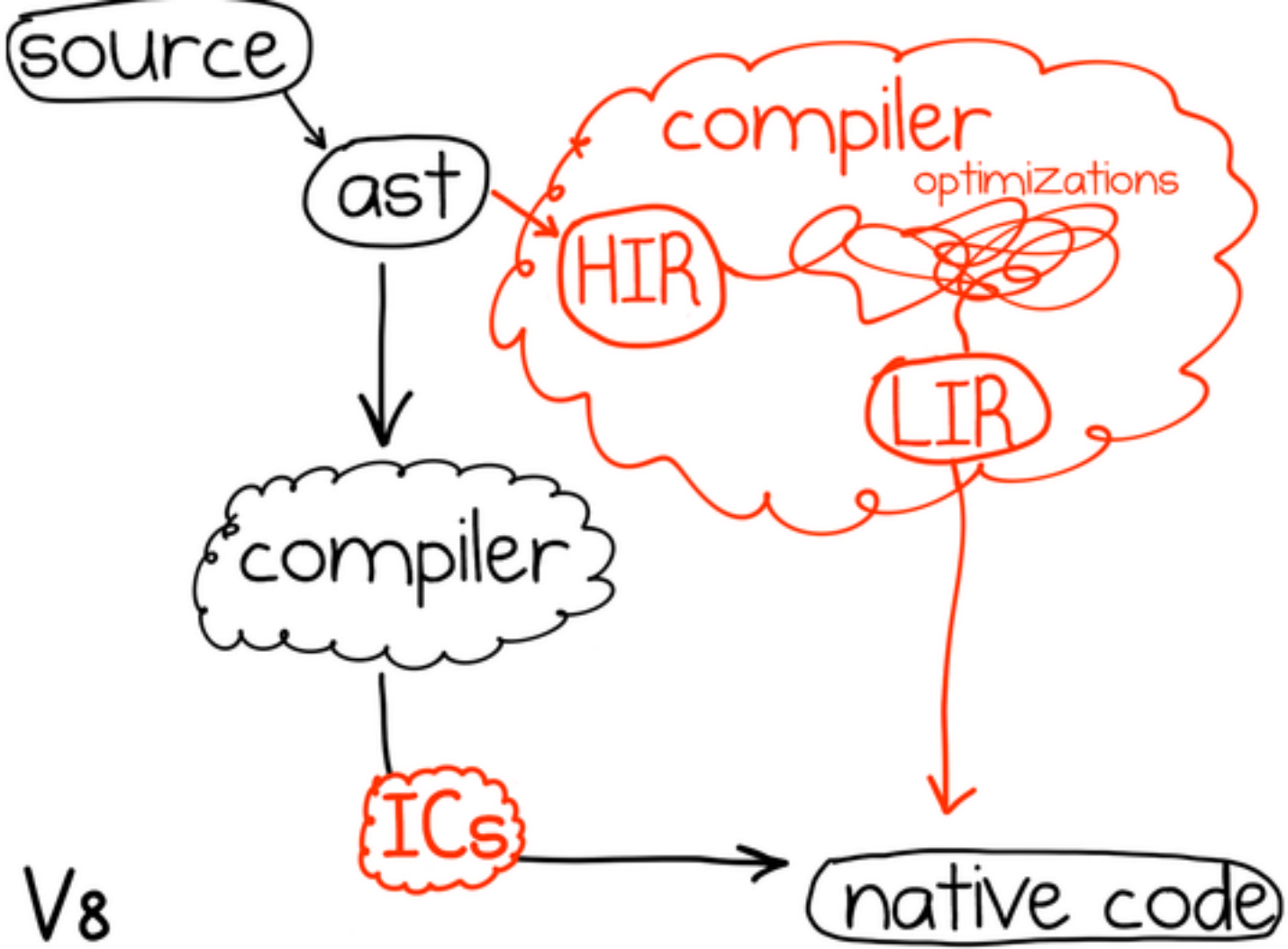
```
Vector.prototype.length = function () {
  return Math.sqrt(this.x * this.x +
                   this.y * this.y);
};
```

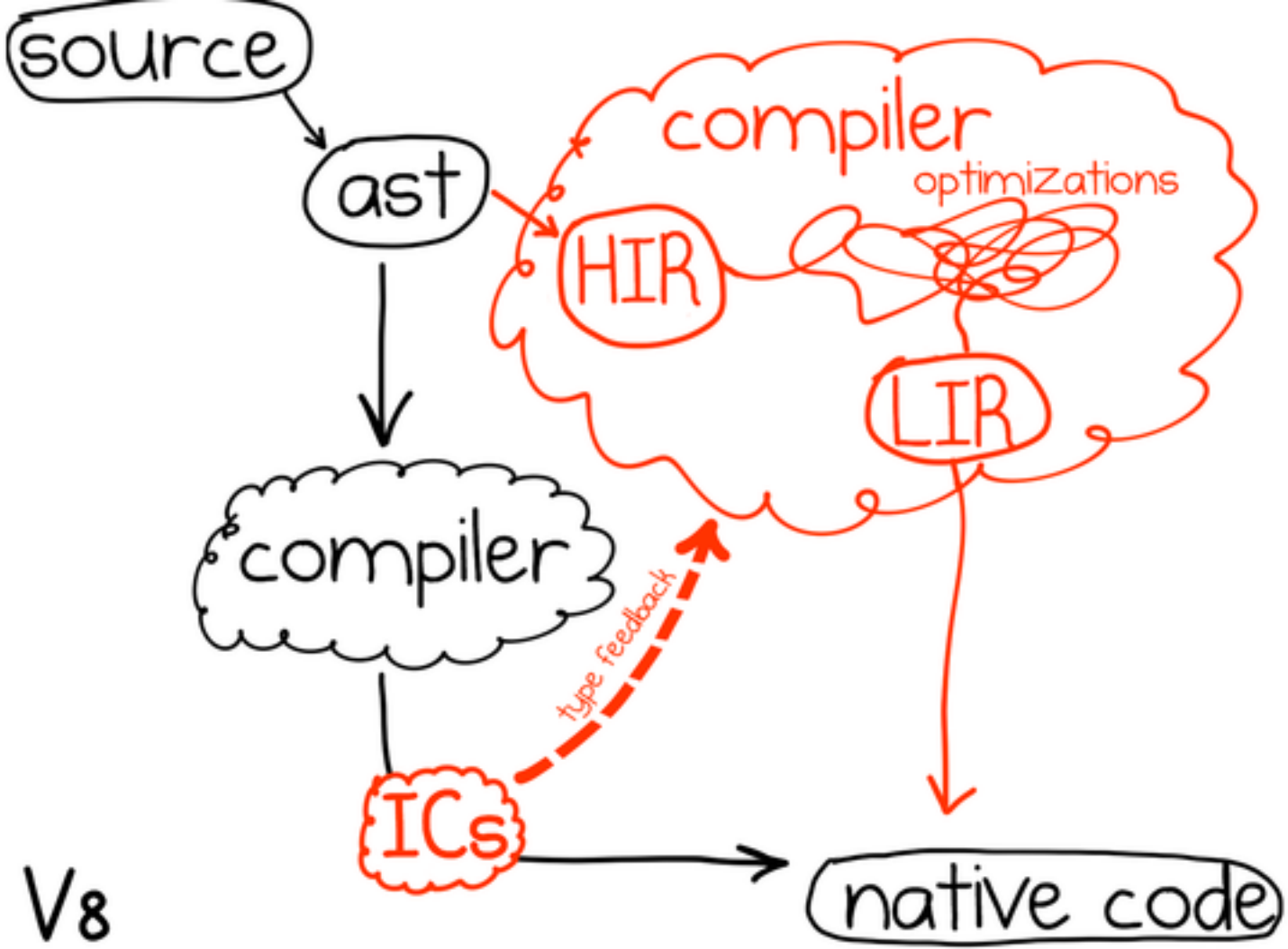
$$(\text{fast}_1 \oplus \text{slow}_1) +$$
$$(\text{fast}_2 \oplus \text{slow}_2) +$$
$$(\text{fast}_3 \oplus \text{slow}_3) + \ldots$$

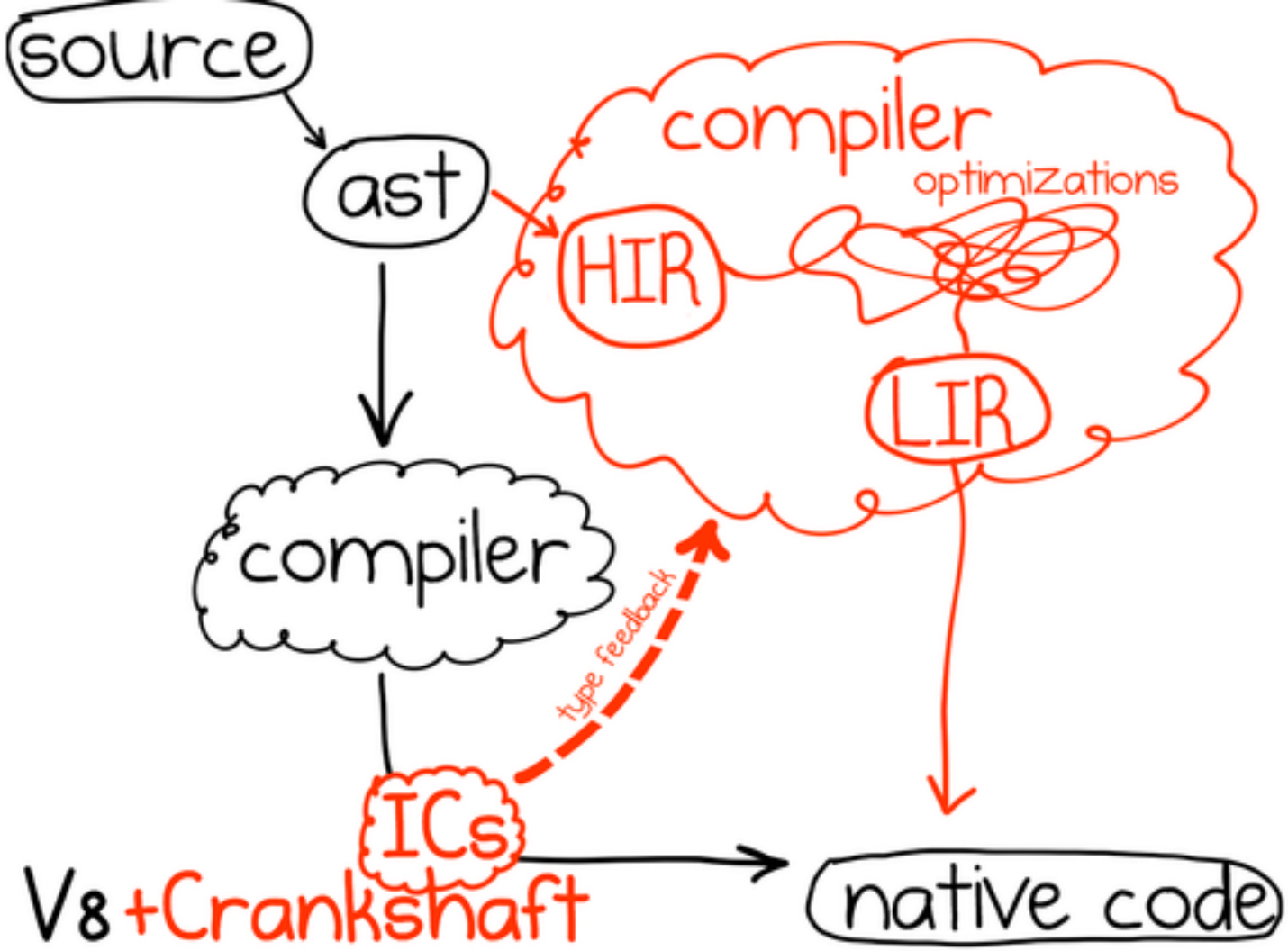
$$(\text{fast}_1 + \text{fast}_2 + \ldots) \oplus \text{slow}$$

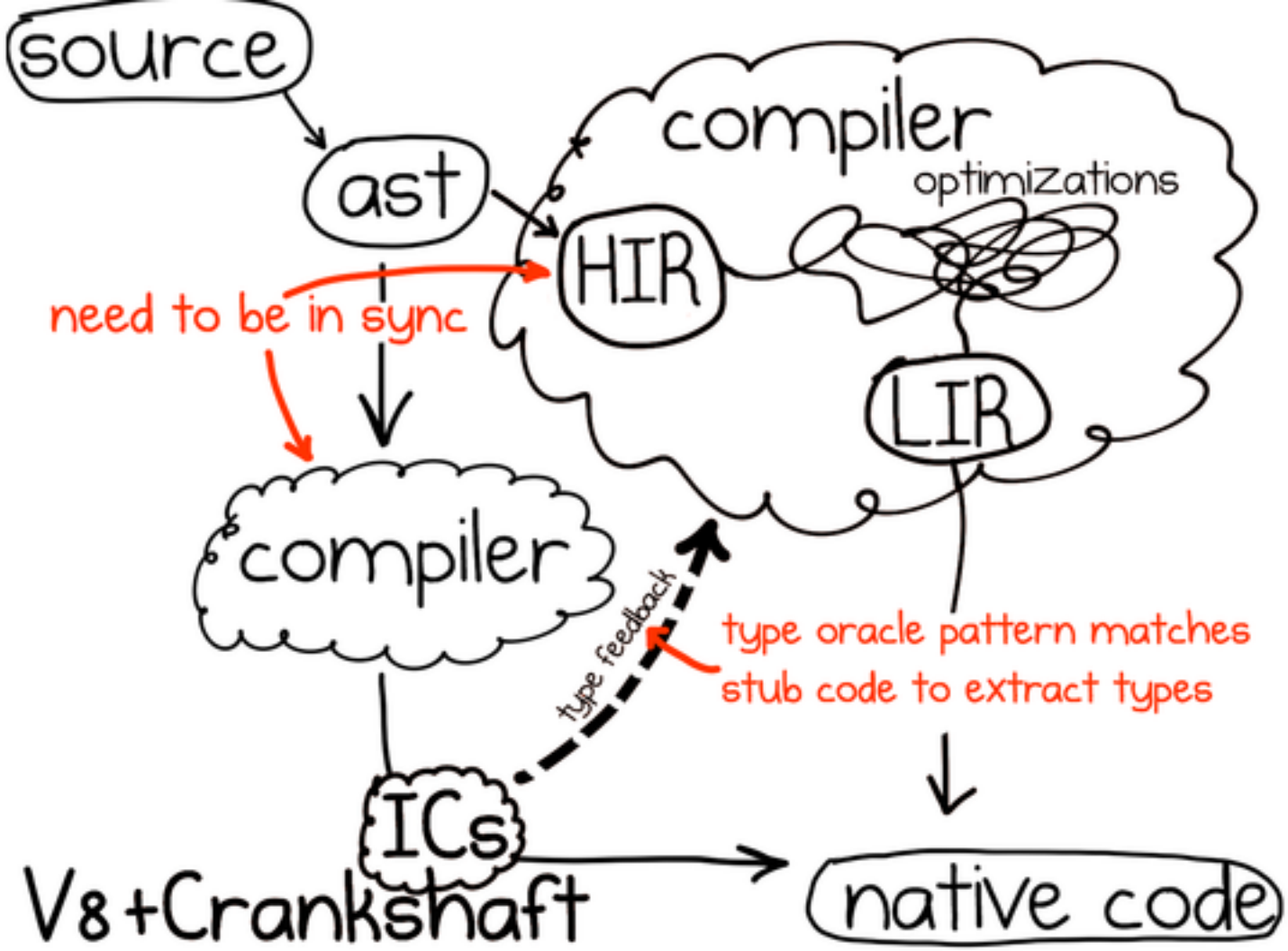

source
ast
compiler
ICs
native code
V8


source
ast
compiler
compiler
ICs
native code
V8


source
ast
compiler
optimizations
HIR
LIR
compiler
ICs
native code
V8


source
ast
compiler
optimizations
HIR
LIR
compiler
type feedback
ICs
native code
V8


source
ast
compiler
compiler
optimizations
HIR
LIR
type feedback
ICs
native code
V8 +Crankshaft


source
ast
compiler
optimizations
HIR
LIR
need to be in sync
compiler
type feedback
type oracle pattern matches
stub code to extract types
ICs
native code
V8+Crankshaft

Crankshaft (2010)

1. compile unoptimized code
2. feed hot functions into optimizer
3. speculate types based on IC states
4. apply classic optimizations
5. emit optimized code

Crankshaft (2010)

1. compile unoptimized code
2. feed hot functions into optimizer
3. **speculate** types based on IC states
4. apply classic optimizations
5. emit optimized code

checks in the code
verify speculations

failed check

⇒ jump to unoptimized code

code can depend on assumptions

[e.g. that some prototype chain does not change]

violated assumption

⇒ deopt dependant code

fast ⊕ slow

fast ⊕ slow

"it's a trap"

manual decomposition

maintenance burden

can derive fast-path (semi)automatically?

"if you have optimizing compiler - use it!"

"if you have optimizing compiler - use it!"

write IR instead of asm

"if you have optimizing compiler - use it!"

V8 stubs; Dart VM intrinsics, regexp

runtime speaks
wrong language

user-code?

```
getInferredTypeOf(element) {
  element = element.implementation;
  return typeInformations.putIfAbsent(element, () {
    return new ElementTypeInformation(element, this);
  });
}
```

```
getInferredTypeOf(element) {
  element = element.implementation;
  return typeInformations.putIfAbsent(element, () {
    return new ElementTypeInformation(element, this);
  });
}
```

```
class _HashMap {
  @jit.AlwaysInline
  putIfAbsent(key, @jit.AlwaysInline ifAbsent) {
    /* ... */
  }
}

@jit.NeverInline
getInferredTypeOf(element) {
  /* ... */
}
```

```
class ListMixin {
  forEach(f) { /* ... */ }
}

class List with ListMixin { }
class Int32List with ListMixin { }
class Float64List with ListMixin { }
```

```
class ListMixin {
  @jit.ReceiverPolymorphic
  forEach(f) {
    /* ... */
  }
}
```

warmup?

precook fast-paths

"Universal" VM

small yet expressive core language

optimization hints

fast ⊕ slow decomposition (guards, assumptions, deopts)

ability to precook fast paths at build step

Thank you