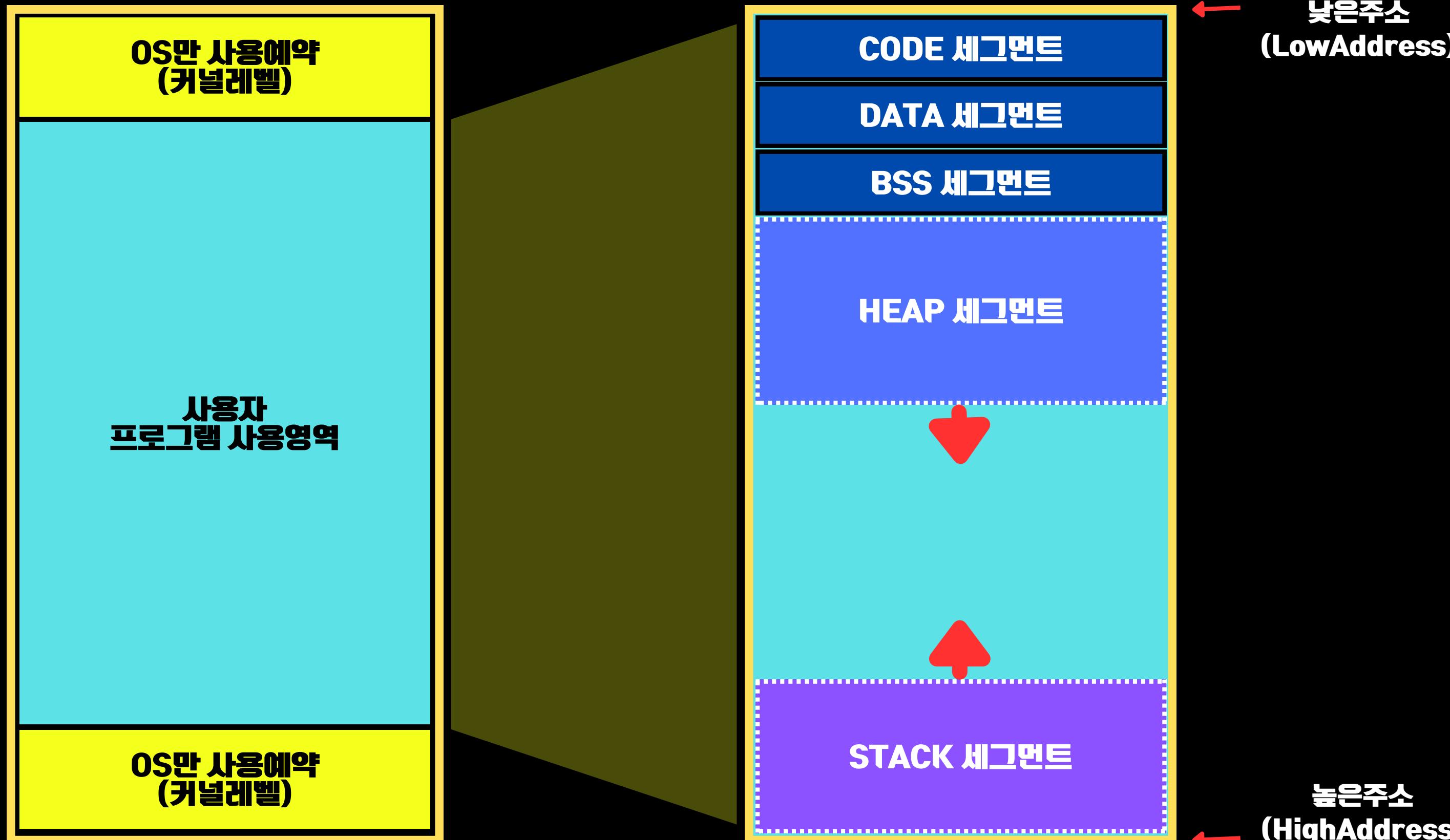




변수와 메모리

Memory Layout



Memory Layout

세그먼트

- ✓ 메모리에 적재되는 데이터를 용도별로 나누어 저장하는 영역

CODE 세그먼트

- ✓ 프로그램의 실행 가능한 코드 데이터
- ✓ TEXT 세그먼트라고도 함

CODE 세그먼트

DATA 세그먼트

BSS 세그먼트

HEAP 세그먼트



STACK 세그먼트

낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

DATA 세그먼트

✓ 초기화가 된 정적변수, 상수나 전역변수, 상수

BSS 세그먼트

✓ 초기화가 안 된 정적변수나 전역변수

```
...  
C언어  
  
/**  
 * DATA 세그먼트  
 */  
  
// 정적 변수  
static int i = 100;  
// 전역 변수  
int j = 200;
```

```
...  
C언어  
  
/**  
 * BSS 세그먼트  
 */  
  
// 정적 변수  
static int i;  
// 전역 변수  
int j;
```



Memory Layout

HEAP 세그먼트

- ✓ 프로그램 실행 중 동적으로 메모리를 할당하는 데이터가 위치하는 곳
- ✓ 영역의 크기가 고정되어 있지 않고 낮은 주소에서 높은 주소 방향으로 확장 됨

```
...  
Javascript  
  
const obj = {  
  name: 'CodingMax',  
  info: {  
    age: 30,  
    address: 'Seoul',  
    country: 'Korea'  
  }  
};
```



낮은주소
(LowAddress)

STACK 세그먼트

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
...  
Javascript  
  
function a() {  
    return 1;  
}  
function b() {  
    return 2;  
}  
function sum() {  
    const n1 = a();  
    const n2 = b();  
    return n1 + n2;  
}  
  
const result = sum();  
console.log(result);
```



낮은주소
(LowAddress)

높은주소
(HighAddress)

Memory Layout

STACK 세그먼트

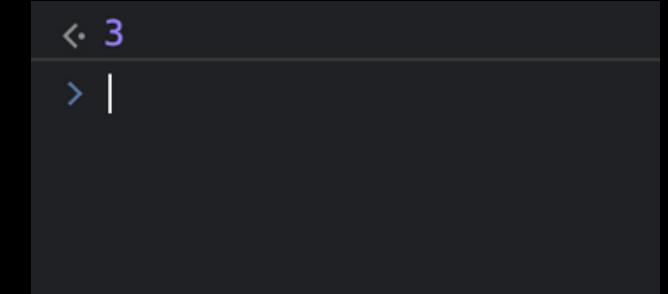
- ✓ 프로세스가 사용하는 스택 영역
- ✓ 함수 스택 프레임(지역변수, 함수 파라미터).
LIFO(Last In – First Out)
- ✓ 영역의 크기가 고정되어 있지 않고 높은 주소
에서 낮은 주소 방향으로 확장 됨

```
... Javascript
function a() {
  return 1;
}
function b() {
  return 2;
}
function sum() {
  const n1 = a();
  const n2 = b();
  return n1 + n2;
}

const result = sum();
console.log(result);
```



콘솔에 출력하고
프로그램 종료



높은주소
(HighAddress)



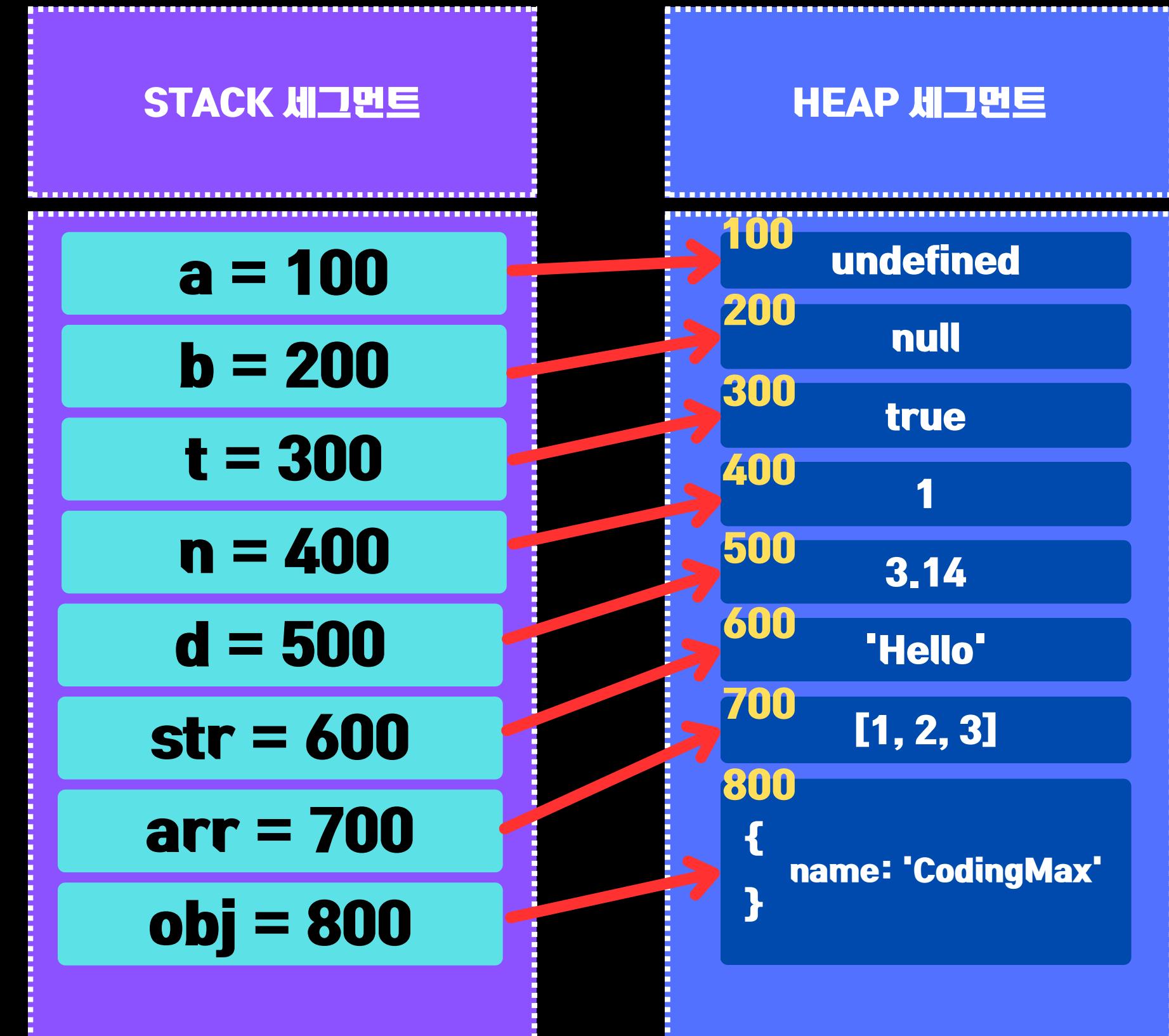
변수와 메모리

v8 blog

JavaScript values in V8 are represented as objects and allocated on the V8 heap,
no matter if they are objects, arrays, numbers or strings.
This allows us to represent any value as a pointer to an object.

자바스크립트 Level

```
...  
Javascript  
  
const a = undefined;  
  
const b = null;  
  
const t = true;  
  
const n = 1;  
  
const d = 3.14;  
  
const str = 'Hello';  
  
const arr = [1, 2, 3];  
  
const obj = {  
  name: 'CodingMax'  
};
```



8 Level

undefined
null
boolean

```
function TestPrimitive() {
    this.a1 = undefined;
    this.b1 = null;
    this.t1 = true;
    this.f1 = false;
    this.a2 = undefined;
    this.b2 = null;
    this.t2 = true;
    this.f2 = false;
}

const myTestPrimitive = new TestPrimitive();
```

The screenshot shows the Chrome DevTools Memory tab with a heap snapshot taken. The constructor for `TestPrimitive` is expanded, revealing its internal structure. Red arrows highlight several nodes where primitive values are stored:

- `a1`: `undefined` @1227
- `b1`: `null` @1039
- `t1`: `true` @1213
- `f1`: `false` @853
- `a2`: `undefined` @1227
- `b2`: `null` @1039
- `t2`: `true` @1213
- `f2`: `false` @853
- `map`: `Object` @112153
- `__proto__`: `Object` @112153
- `1`: `"object"` @1051
- `0`: `"null"` @1039
- `1`: `"object"` @1051
- `0`: `"null"` @1039
- `1`: `"undefined"` @1227
- `2`: `"undefined"` @1227
- `1`: `heap number` @331
- `2`: `"undefined"` @1227
- `1`: `heap number` @331
- `1`: `"boolean"` @721
- `0`: `"false"` @853
- `1`: `"boolean"` @721
- `0`: `"false"` @853
- `1`: `"boolean"` @721
- `0`: `"true"` @1213
- `1`: `"boolean"` @721
- `0`: `"true"` @1213

8 Level

undefined
null
boolean

```
13  namespace v8 {
14    namespace internal {
15
16      #include "torque-generated/src/objects/oddball-tq.inc"
17
18      // The Oddball describes objects null, undefined, true, and false.
19  ↗ | class Oddball : public PrimitiveHeapObject {
```

● ● ● C++

class Undefined : public Oddball {
public:
 DECL_CAST(Undefined)
 OBJECT_CONSTRUCTORS(Undefined, Oddball);
};

● ● ● C++

class Null : public Oddball {
public:
 DECL_CAST(Null)
 OBJECT_CONSTRUCTORS(Null, Oddball);
};

● ● ● C++

class Boolean : public Oddball {
public:
 DECL_CAST(Boolean)
 V8_INLINE bool ToBool(Isolate* isolate) const;
 OBJECT_CONSTRUCTORS(Boolean, Oddball);
};

● ● ● C++

class True : public Boolean {
public:
 DECL_CAST(True)
 OBJECT_CONSTRUCTORS(True, Boolean);
};

● ● ● C++

class False : public Boolean {
public:
 DECL_CAST(False)
 OBJECT_CONSTRUCTORS(False, Boolean);
};

8 Level

undefined
null
boolean

1

```
// Defines all the read-only roots in Heap.
#define STRONG_READ_ONLY_ROOT_LIST(V)
/* Cluster the most popular ones in a few cache lines here at the top. */
/* The first 32 entries are most often used in the startup snapshot and */
/* can use a shorter representation in the serialization format. */
V(Map, free_space_map, FreeSpaceMap)
V(Map, one_pointer_filler_map, OnePointerFillerMap)
V(Map, two_pointer_filler_map, TwoPointerFillerMap)
V(Oddball, uninitialized_value, UninitializedValue)
V(Undefined, undefined_value, UndefinedValue)
V(Hole, the_hole_value, TheHoleValue)
V	Null null_value NullValue)
```

v8 /src/root/root.h

2

```
Node* WasmGraphBuilder::UndefinedValue() {
    return LOAD_ROOT(UndefinedValue, undefined_value);
}
```

3

```
#define LOAD_ROOT(RootName, factory_name)
    (parameter_mode_ == kNoSpecialParameterMode)
        ? graph()->NewNode(mcggraph()->common()->HeapConstant(
            isolate_->factory()->factory_name()))
        : gasm_->LoadImmutable(
            MachineType::Pointer(), BuildLoadIsolateRoot(),
            IsolateData::root_slot_offset(RootIndex::k##RootName)))
```

4

```
Node* JSGraph::HeapConstant(Handle<HeapObject> value) {
    Node** loc = cache_->FindHeapConstant(value);
    if (*loc == nullptr) {
        *loc = graph()->NewNode(common()->HeapConstant(value));
    }
    return *loc;
}
```

```
Node** CommonNodeCache::FindHeapConstant(Handle<HeapObject> value) {
    return heap_constants_->Find(base::bit_cast<intptr_t>(value.address()));
}
```

```
namespace compiler {
class CommonNodeCache final {
public:
    // Return all nodes from the cache.
    void GetCachedNodes(ZoneVector<Node*>* nodes);

private:
    Int32NodeCache int32_constants_;
    Int64NodeCache int64_constants_;
    Int32NodeCache tagged_index_constants_;
    Int32NodeCache float32_constants_;
    Int64NodeCache float64_constants_;
    IntPtrNodeCache external_constants_;
    IntPtrNodeCache pointer_constants_;
    Int64NodeCache number_constants_;
    IntPtrNodeCache heap_constants_;
    RelocInt32NodeCache relocatable_int32_constants_;
    RelocInt64NodeCache relocatable_int64_constants_;
};
```

```
// A cache for nodes based on a key. Useful for implementing canonicalization of
// nodes such as constants, parameters, etc.
template <typename Key, typename Hash = base::hash<Key>,
          typename Pred = std::equal_to<Key>>
class EXPORT_TEMPLATE_DECLARE(V8_EXPORT_PRIVATE) NodeCache final {
public:
    explicit NodeCache(Zone* zone) : map_(zone) {}
    ~NodeCache() = default;
    NodeCache(const NodeCache&) = delete;
    NodeCache& operator=(const NodeCache&) = delete;
```

Oddball 은 Singleton

8 Level String

```
Javascript String

function TestString() {
    this.s1 = '';
    this.s2 = 'a';
    this.s3 = 'Hello';
    this.s4 = '안녕하세요';
    this.s5 = '😊';
    this.t1 = '';
    this.t2 = 'a';
    this.t3 = 'Hello';
    this.t4 = '안녕하세요';
    this.t5 = '😊';
}
const myTestString = new TestString();
```

The screenshot shows the Chrome DevTools Memory tab with a heap snapshot named 'Snapshot 1'. The 'Profiles' section shows the 'Constructor' for the 'TestString' object. Inside, fields like 's1', 's2', etc., are listed, each pointing to a specific string value. These values are represented as various types of strings (SeqString, InternalizedString, etc.) at different memory locations.

```
V8 :: src/objects/objects.h

// - PrimitiveHeapObject ①
// - String
//   - SeqString
//     - SeqOneByteString
//     - SeqTwoByteString
//   - SlicedString
//   - ConsString
//   - ThinString
//   - ExternalString
//     - ExternalOneByteString
//     - ExternalTwoByteString
// - InternalizedString ②
//   - SeqInternalizedString
//     - SeqOneByteInternalizedString
//     - SeqTwoByteInternalizedString
//   - ConsInternalizedString
//   - ExternalInternalizedString
//     - ExternalOneByteInternalizedString
//     - ExternalTwoByteInternalizedString
```

① **string은 1바이트로 표현가능한 유니코드인
지 판단해 적절한 타입으로 힙에 할당한다.**

② **힙에 할당한 문자열값은 최적화를 위해
재사용한다.**



변수와 메모리

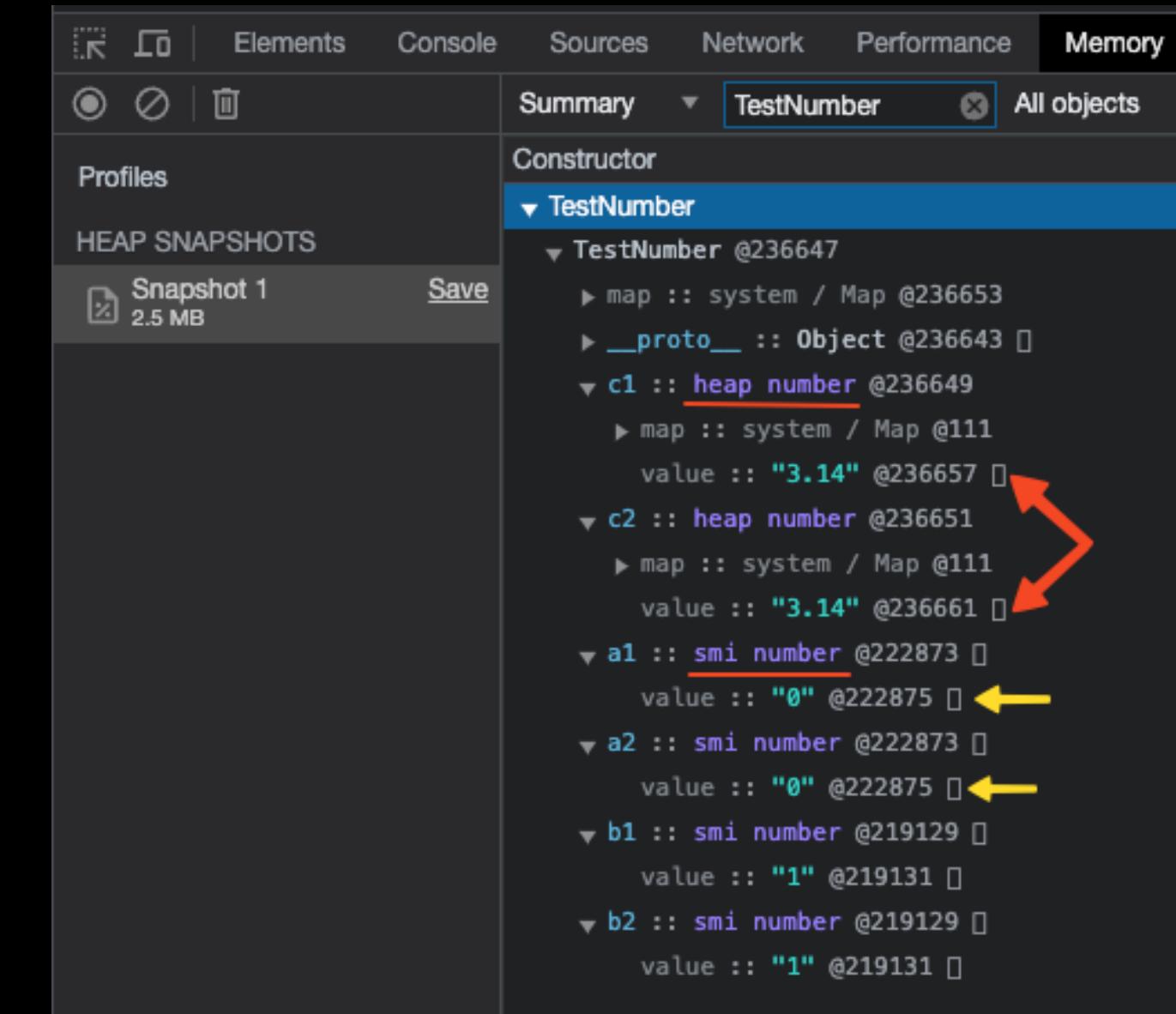
8 Level Number

```

●●● Javascript
function TestNumber() {
    this.a1 = 0;
    this.b1 = 1;
    this.c1 = 3.14;
    this.a2 = 0;
    this.b2 = 1;
    this.c2 = 3.14;
}
const myTestNumber = new TestNumber();

●●● Reuse Heap Number
function reuseHeapNumber() {
    return 3.14;
}

```



- ① **number는 Heap Number 와 smi 로 구분된다.**
- ② **Heap Number는 공유될 수도 있고 안 될 수도 있다. 예제처럼 변경 가능한 Heap Number는 재사용하지 않는다. reuseHeapNumber처럼 변경 불가능할 때만 재사용된다.**
- ③ **smi는 스냅샷에서 Heap 주소가 같아 공유되는 것 같지만 공유되지 않는다. 비트 표현이 같아서 같은 Heap 주소로 보일 뿐이다.**

8 Level Number – smi

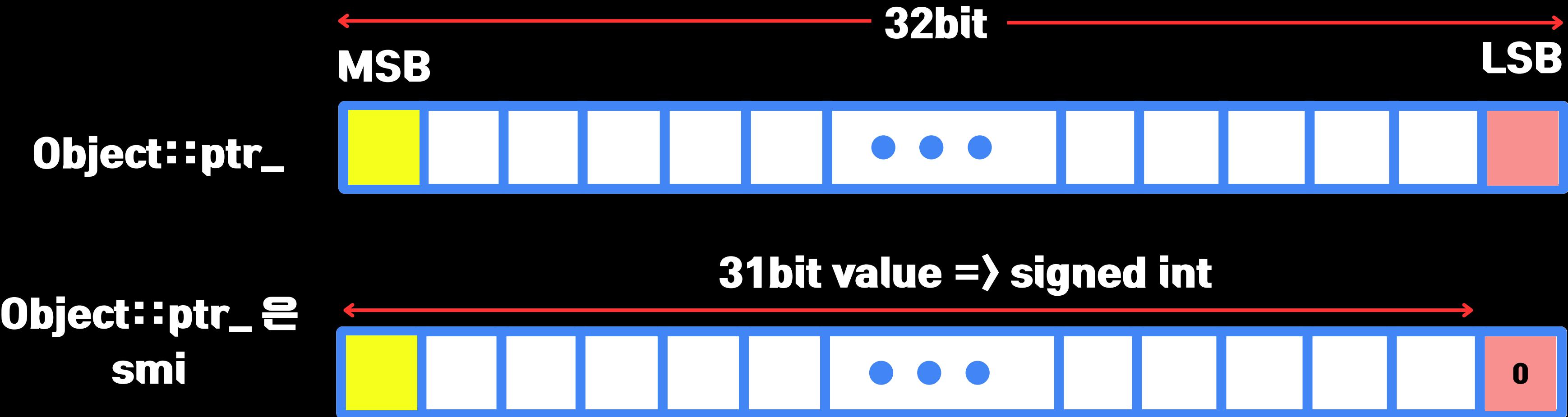
smi = small integer

```
••• V8 :: src/objects/objects.h

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

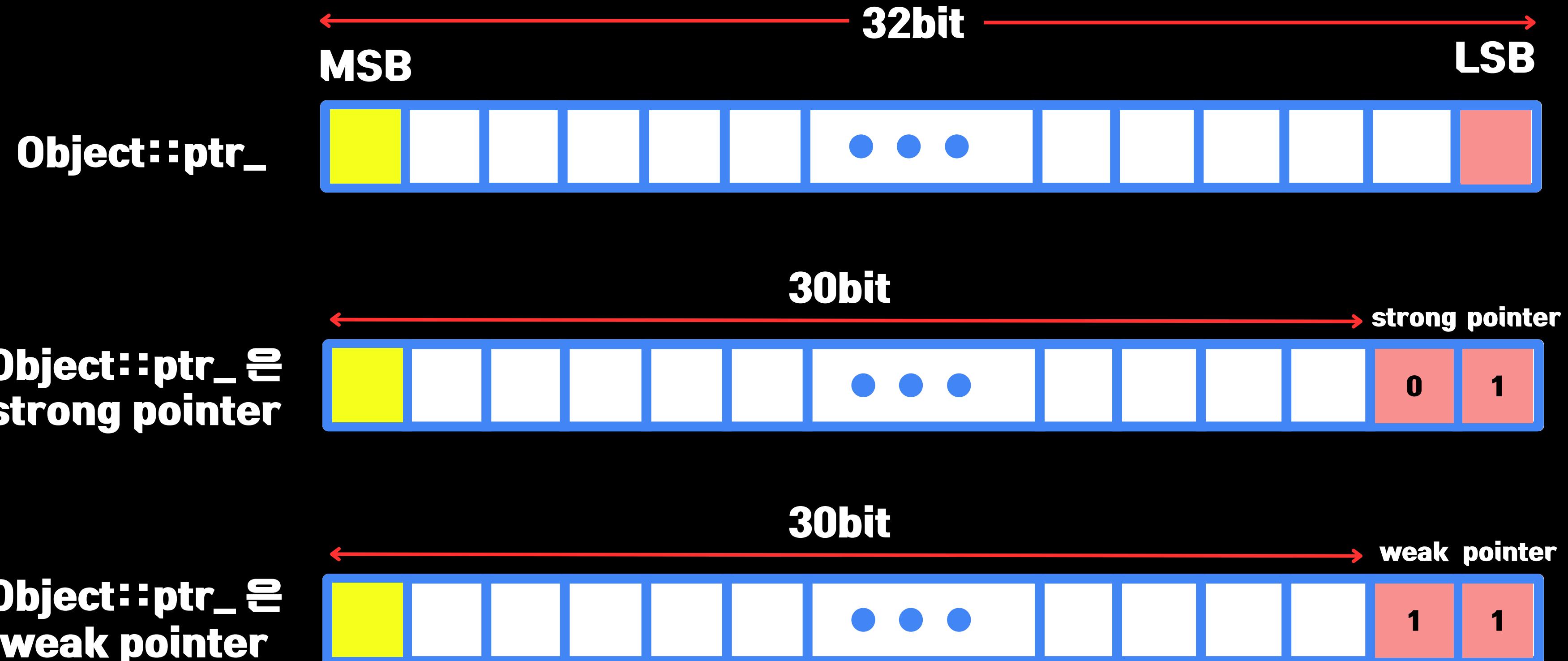
- ① **포인터 태깅(Pointer Tagging)**을 사용해 특정 비트값을 바탕으로 객체를 가리키는 `ptr_` 변수를 **smi** 값과 **Heap Number** 객체를 가리키는 포인터 주소값으로 사용한다.
- ② 32비트 머신에서 **smi**는 31비트 `signed int`
- ③ 32비트 머신에서 **Heap Number**를 가리키는 포인터라면 32비트 포인터

8 Level Number - smi



- ① LSB가 0 이면 `Object::ptr_` 을 31bit signed int 값을 저장하는 변수로 사용
- ② 따라서 smi 는 heap 에 생성되지 않고 바로 `Object::ptr_` 변수에 값이 쓰인다.

8 Level Number – HeapNumber



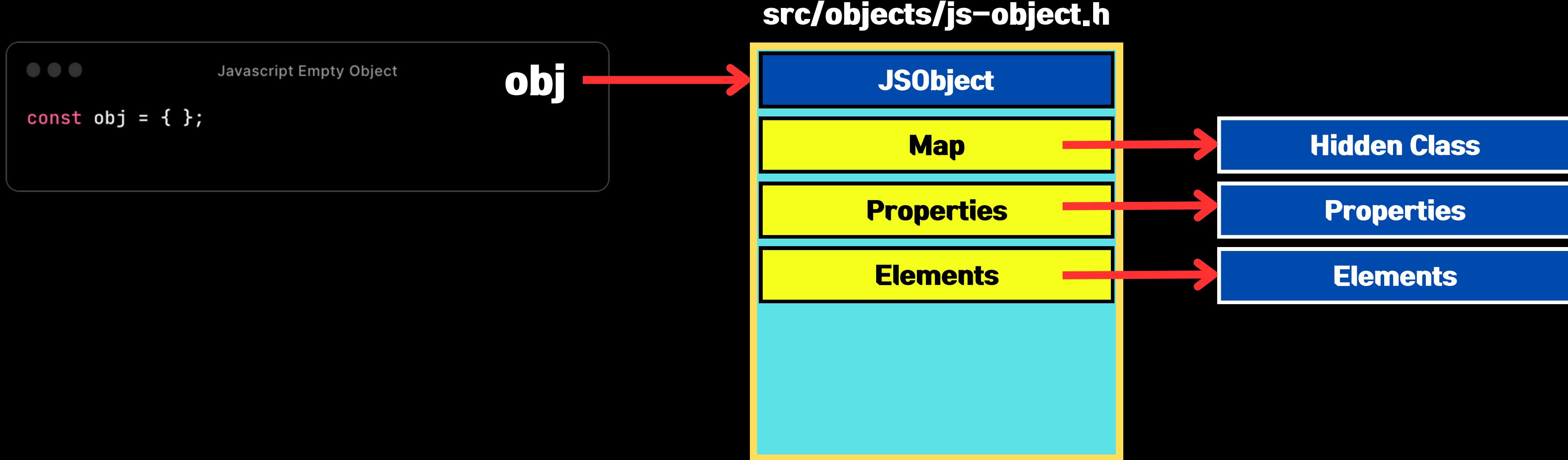
8 Level Number - 64BIT





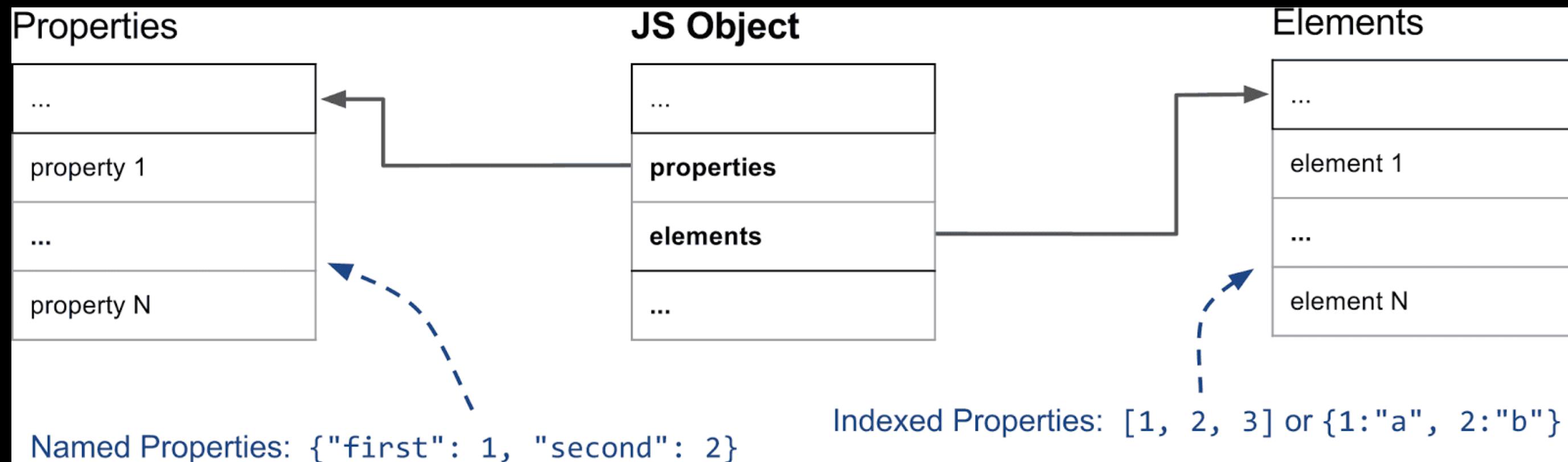
변수와 메모리

8 Level Object



- ① 자바스크립트 객체를 만들면 V8 내부에서 `JSObject`가 힙에 생성되고 `obj` 변수가 해당 객체를 가리킨다.
- ② 각 `JSObject`는 기본적으로 3가 포인터를 가지고 있다.
 - ✓ `Map`은 `Hidden Class`를 가리키는 포인터. 객체의 모양(속성 및 크기 정보 등) 정보를 갖는다.
 - ✓ `Properties`는 이름이 있는 속성들을 담고 있는 객체를 가리키는 포인터
 - ✓ `Elements`는 숫자 인덱스형을 키로 갖는 속성들을 담고 있는 객체를 가리키는 포인터

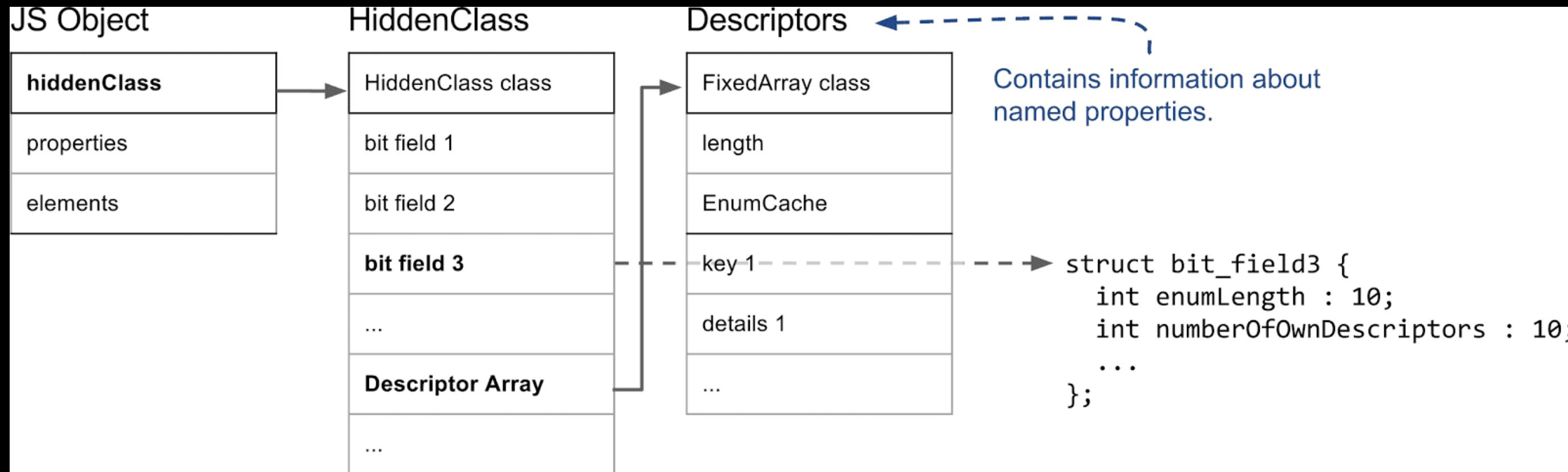
V8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① 이를 있는 속성 즉, 키가 문자열인 속성은 Properties에 저장된다.
- ② 키가 숫자인 속성은 Elements에 저장된다.

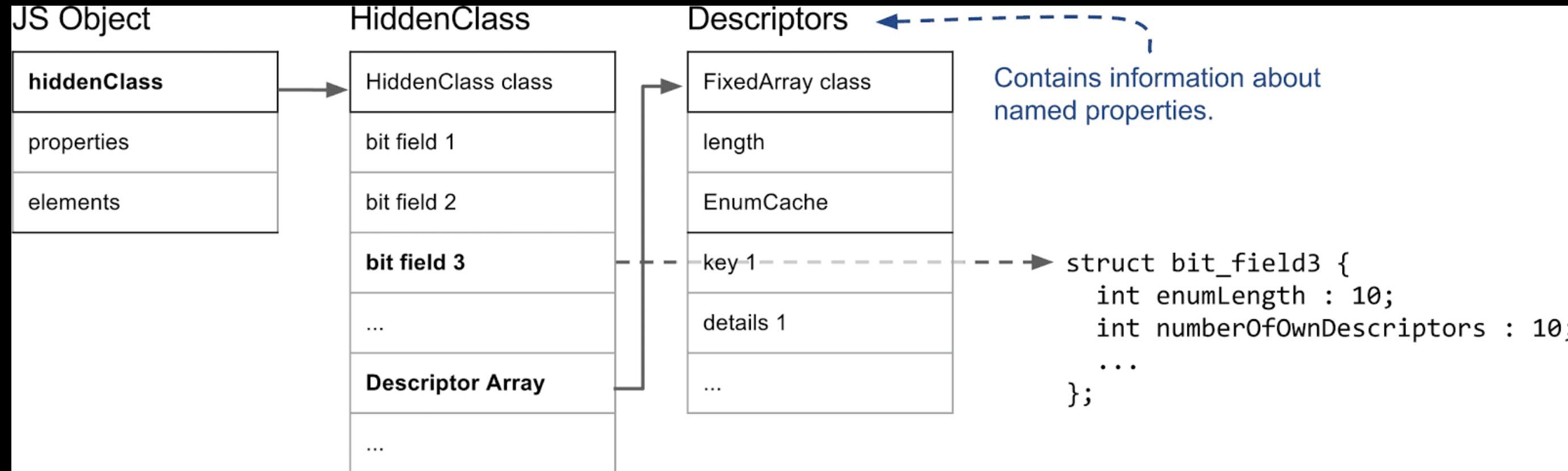
8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① **HiddenClass**는 객체 속성 갯수, 객체의 Prototype 포인터 등 메타데이터를 가지고 있다.
- ② 자바스크립트의 객체는 런타임에 속성이 추가, 삭제 될 수 있기 때문에 객체의 최종 모양을 미리 알 수 있는 방법이 없다.
- ③ 그래서 하든클래스를 사용해 동적으로 변경되는 객체의 모양에 대한 정보를 관리한다.

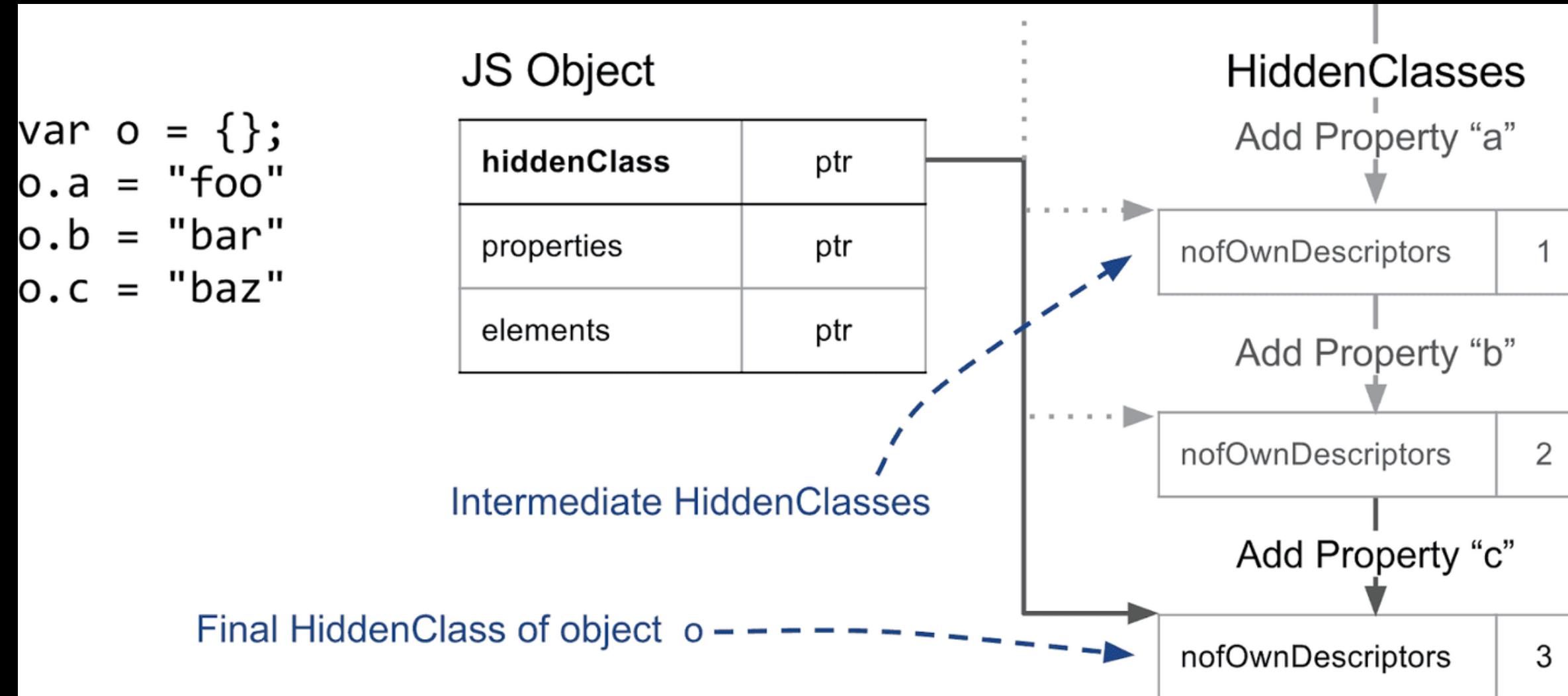
8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① 세번째 비트 필드가 가장 중요하다. 속성 갯수를 담고 있고 Descriptor Array 포인터를 가지고 있기 때문이다.
- ② Descriptor Array는 속성의 이름 정보와 값이 메모리 어느 위치에 있는지를 가지고 있어 매우 중요하다.

8 Level Object

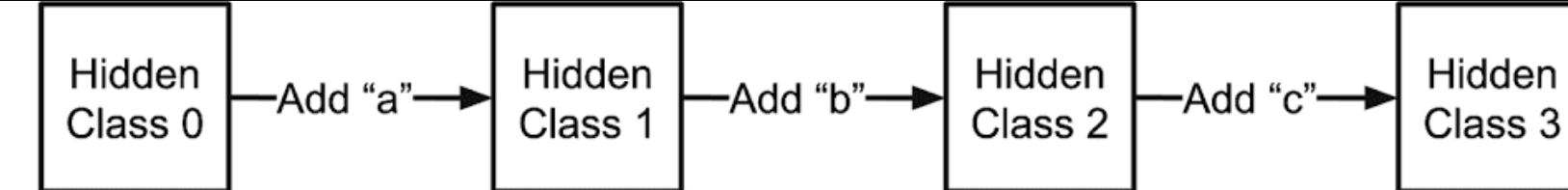


이미지출처: <https://v8.dev/blog/fast-properties>

- ① 객체에 새로운 속성을 추가할 때마다 하든클래스가 생성된다.
- ② JSObject는 최종 하든클래스를 포인트한다.
- ③ 최종 하든클래스의 nOfOwnDescriptor 값이 3으로 3개 속성에 대한 정보를 담고 있다.

V8 Level Object

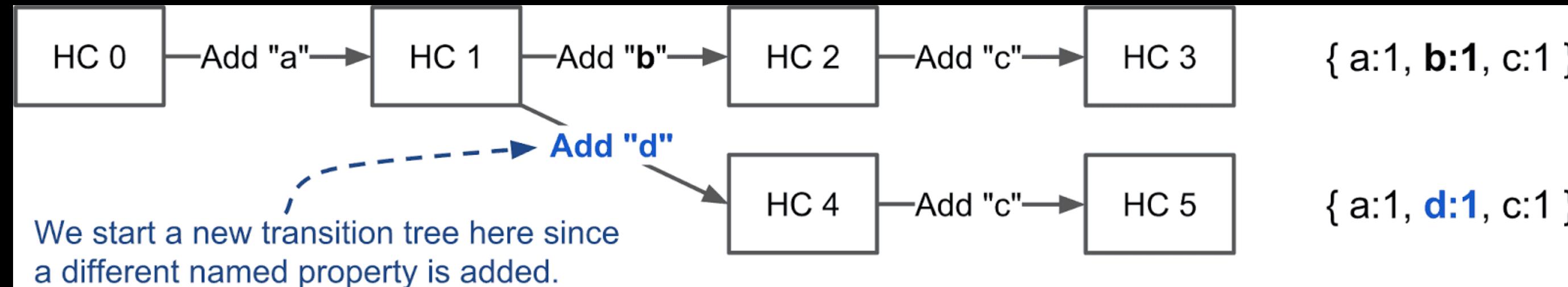
```
{ a:"foo", b:"bar", c:"baz" }  
{ a:"foo", b:"bar", c:"baz", 0:"foo1" }  
{ a:"foo", b:"bar", c:"baz", 0:"foo1", 1:"foo2" }
```



이미지출처: <https://v8.dev/blog/fast-properties>

- ① 단, 배열 인덱스처럼 숫자를 기로 갖는 속성을 추가하면 하든클래스를 새로 생성하지 않는다.
- ② Descriptor Array 성능 때문으로 객체의 속성은 대부분이 이름이 있고 정수형 인덱스 속성은 잘 사용하지 않는다.
- ③ `obj[0] = 10; obj[1] = 20;` 처럼 숫자키를 갖는 속성을 추가할 수는 있지만 잘 사용하지 않는다. 배열이 있기 때문이다.

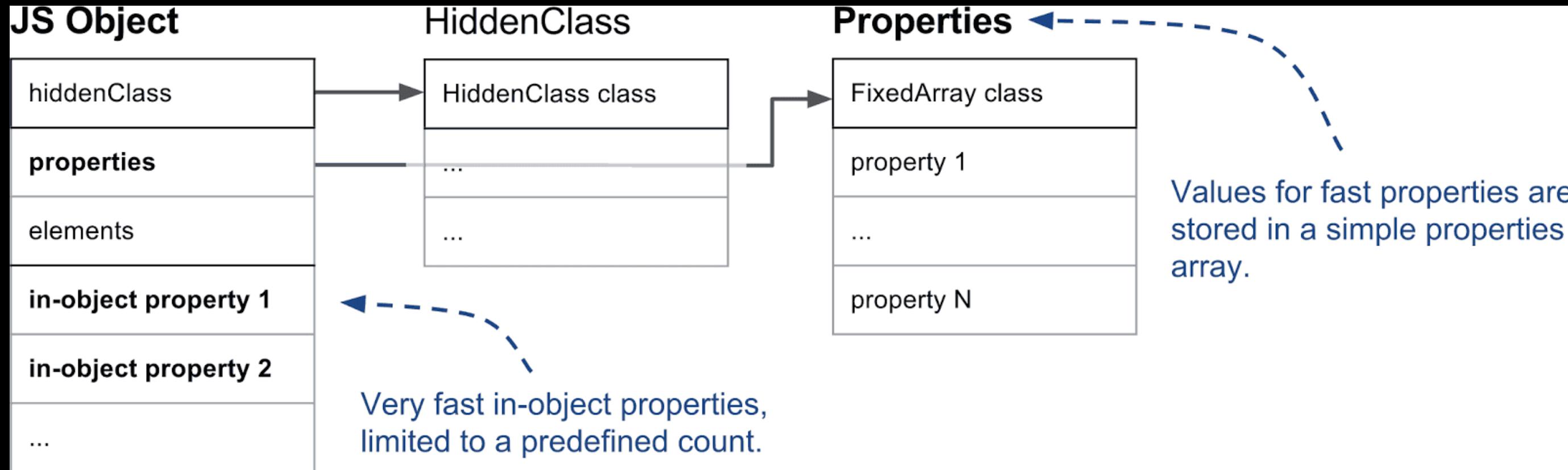
8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① 히든클래스는 다른 객체와 공유될 수 있다.
- ② 새로운 빈 객체에 'a' 속성을 추가하면 H0과 H1을 공유한다.
- ③ 여기에서 새로운 속성 'd'를 추가하면 히든클래스 루트에서 새로운 브랜치를 만들어 히든클래스 HC4를 만든다.
- ④ 'c'를 추가하면 새로운 히든클래스 브랜치에 히든클래스 HC5를 만든다. 그래서 새로 만든 객체는 최종적으로 HC5를 히든클래스로 사용한다.

V8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① **in-object:** 객체 자체가 담고 있는 속성으로 가장 접근이 빠른 속성이다. **in-object** 속성은 객체마다 다르다. 예로 **Symbol** 등이 **in-object**로 구현되어 있다.
- ② **in-object** 속성은 객체를 통해 **direct**로 접근할 수 있다.
- ③ **in-object** 속성이 아닌 일반(Normal) 속성은 객체의 **properties** 포인터를 통해 간접 접근할 수 있다.

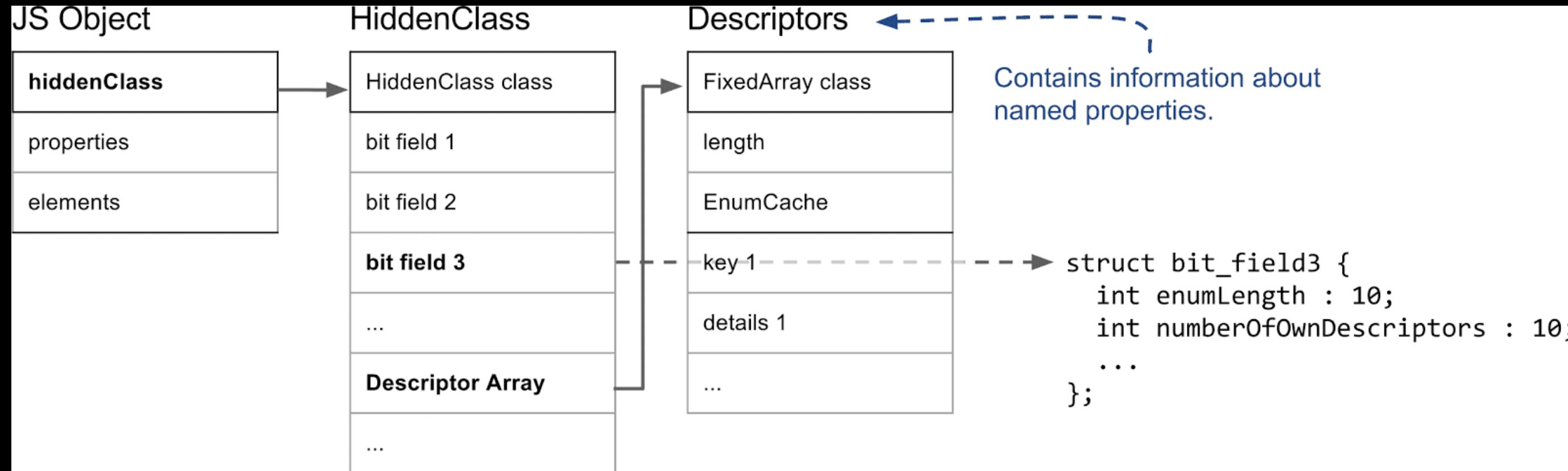
...

V8 src/objects/js-array.h

```
class JSArrayIterator: ... {
public:
    DECL_PRINTER(JSArrayIterator)
    DECL_VERIFIER(JSArrayIterator)

    // [kind]: the [[ArrayIterationKind]] inobject property.
    inline IterationKind kind() const;
    inline void set_kind(IterationKind kind);
```

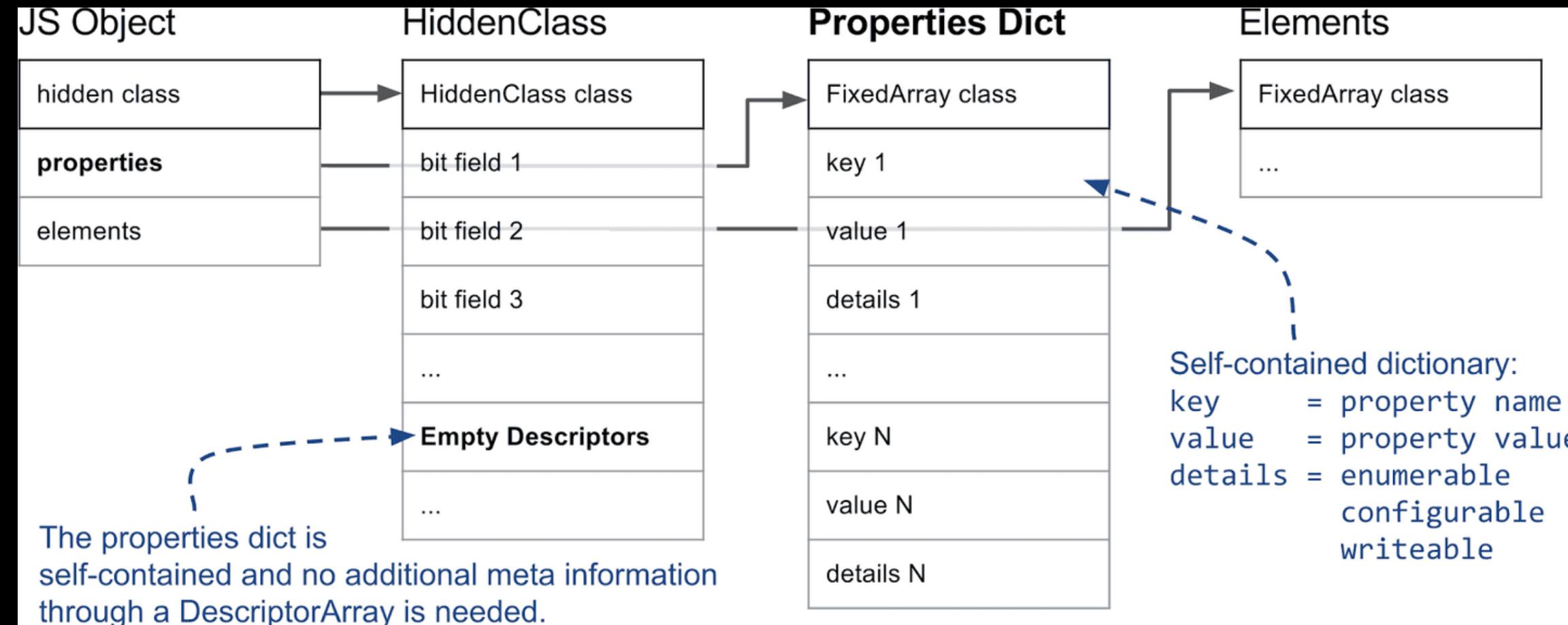
8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① **Fast 속성**은 값에 직접 접근할 수 있는 속성. 예) **string** 값, **number** 값, **boolean** 값, **undefined**, **null**
- ② **Fast 속성**은 **HiddenClass**의 자체 속성 스토어인 **Descriptor Array**를 사용한다.
- ③ 속성 이름이 문자열이어도 **Descriptor Array**의 **Descriptor**는 인덱스를 통해 접근할 수 있다.
- ④ 그래서 **Fast 속성**이라고 한다.

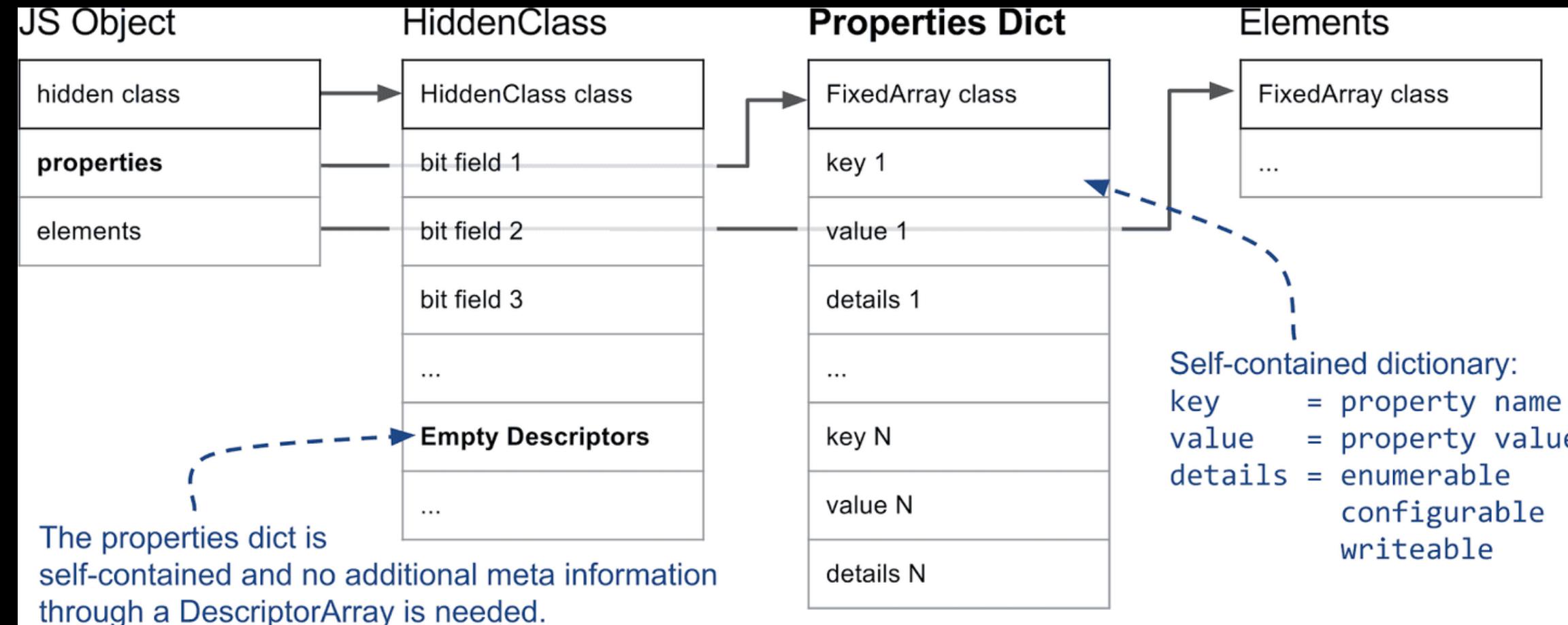
8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

- ① 객체, 배열, 함수와 같은 객체를 값으로 갖는 속성은 **Descriptor**를 사용하지 않고 속성 사전을 사용해 저장한다.
- ② 속성 값인 객체에 속성의 추가와 삭제가 빈번하다면 **Descriptor Array**와 **Hidden Class** 관리에 많은 메모리와 시간이 필요하다.
- ③ 그래서 V8은 **slow-property** 개념을 사용한다. 속성과 메타데이터가 **Descriptor**와 **Hidden Class**가 아닌 **Property 사전(Dict)**에 모두 저장하여 **Hidden Class** 업데이트 없이 속성을 추가하고 삭제할 수 있도록 한다.

8 Level Object



이미지출처: <https://v8.dev/blog/fast-properties>

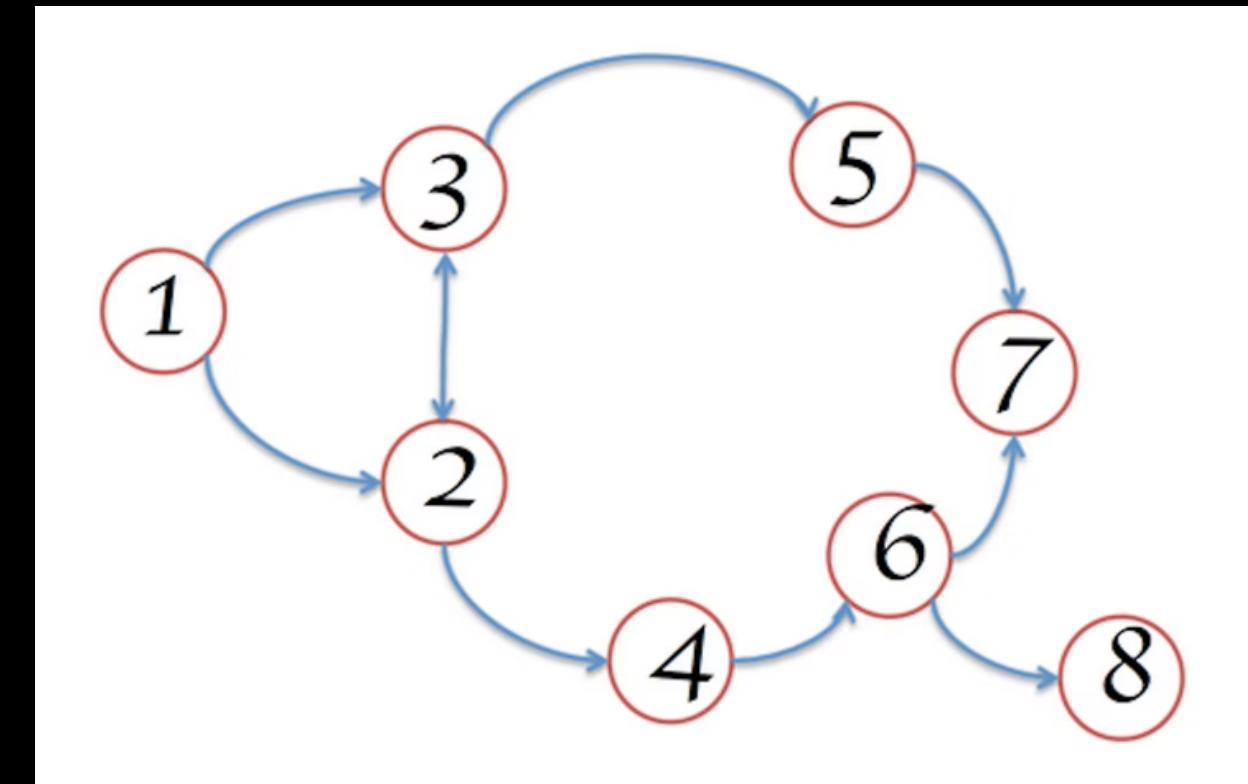
- ① **in-object 속성은 객체 자체에 저장된다.**
- ② **fast 속성은 Property Store에 존재하고 모든 메타 데이터는 HiddenClass에 있는 Descriptor Array에 저장된다.**
- ③ **slow 속성은 자체 보유한 속성 사전에 존재하며 HiddenClass를 사용하지 않는다.**



변수와 메모리

8 Level Object Size

Object Graph



이미지출처: <https://developer.chrome.com/docs/devtools/memory-problems/memory-101/>

```

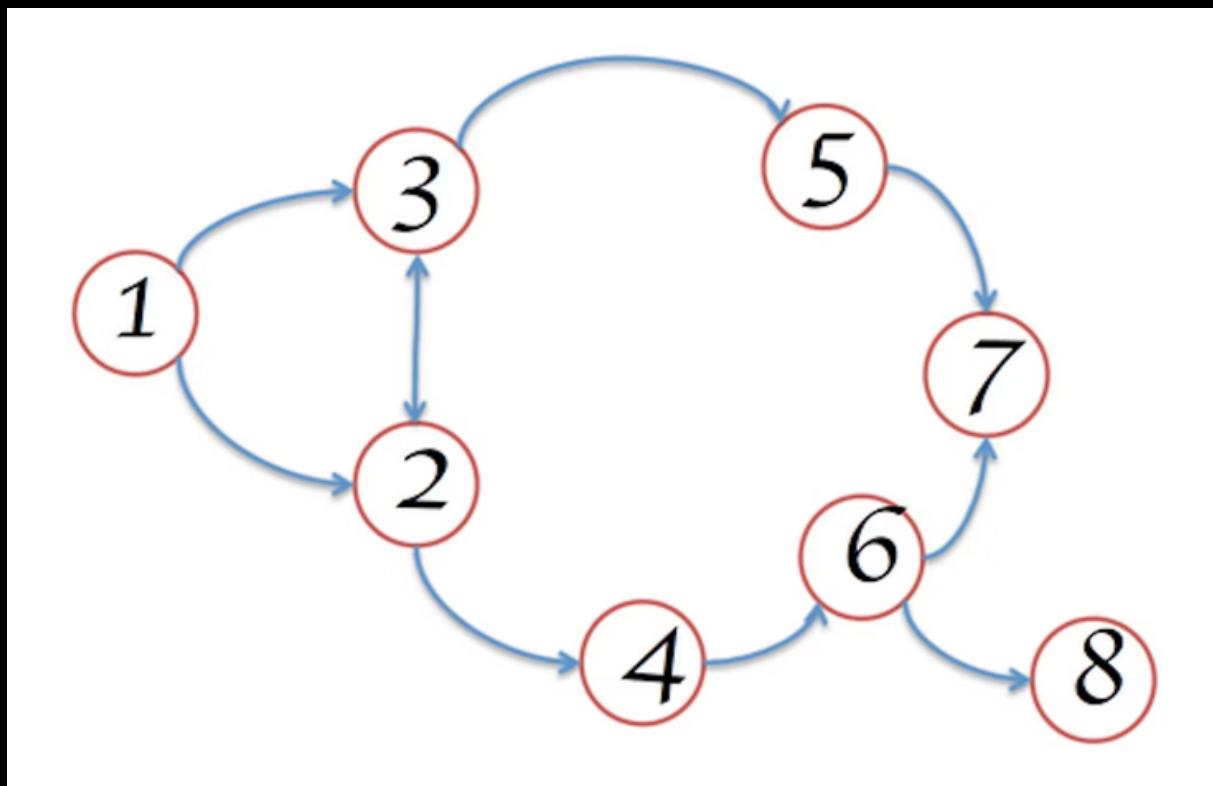
Javascript Object Size

function TestObjectSize() {
    this.age = 10;
}
const myObj = new TestObjectSize();
  
```

Constructor	Distance	Shallow Size	Retained Size
TestObjectSize	4	48 0 %	124 0 %
TestObjectSize @377475	4	48 0 %	124 0 %
__proto__ :: Object @377475	3	28 0 %	124 0 %
map :: system / Map @377475	5	40 0 %	68 0 %
age :: smi number @347829	3	0 0 %	0 0 %

8 Level Object Size

Object Graph



Constructor	Distance	Shallow Size	Retained Size
▼ TestObjectSize	4	48 0 %	124 0 %
▼ TestObjectSize @377475	4	48 0 %	124 0 %
▶ __proto__ :: Object @377475	3	28 0 %	124 0 %
▶ map :: system / Map @377475	5	40 0 %	68 0 %
▶ age :: smi number @347829	3	0 0 %	0 0 %

이미지출처: <https://developer.chrome.com/docs/devtools/memory-problems/memory-101/>

- ① **Shallow Size:** 객체 그 자체만의 크기
- ② **Retained Size:** 객체의 GC root로부터 출발하여 도달할 수 있는 모든 종속된 객체까지 모두 메모리에서 삭제했을 때의 크기

8 Level Object Size

Javascript Object Size

```
function TestObjectSize() {
    this.age = 10;
}
const myObj = new TestObjectSize();
```

Constructor	Distance	Shallow Size	Retained Size
▼ TestObjectSize	4	48 0 %	124 0 %
▼ TestObjectSize @377475	4	48 0 %	124 0 %
► __proto__ :: Object @377473	3	28 0 %	124 0 %
► map :: system / Map @377474	5	40 0 %	68 0 %
► age :: smi number @347829	3	0 0 %	0 0 %

Javascript Object Size

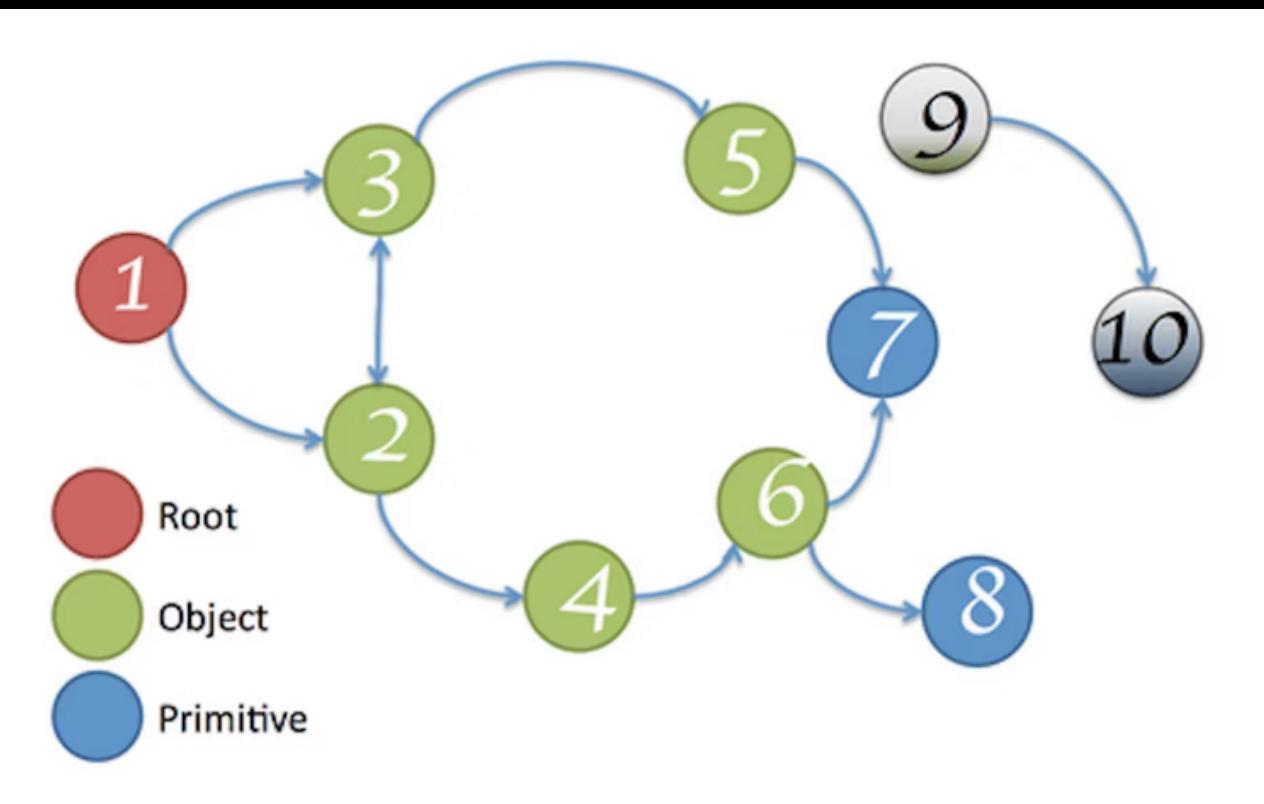
```
function TestObjectSize() {
    this.age = 10;
}
const myObj = new TestObjectSize();
myObj.name = 'CodingMax';
```

Constructor	Distance	Shallow Size	Retained Size
▼ TestObjectSize	4	48 0 %	236 0 %
▼ TestObjectSize @377475	4	48 0 %	236 0 %
► map :: system / Map @381269	5	40 0 %	156 0 %
► __proto__ :: Object @377473	3	28 0 %	124 0 %
► name :: "CodingMax" @378527	5	24 0 %	24 0 %
► age :: smi number @378845	3	0 0 %	0 0 %

① Shallow Size: 객체 그 자체만의 크기

② Retained Size: 객체의 GC root로부터 출발하여 도달할 수 있는 모든 종속된 객체까지 모두 메모리에서 삭제했을 때의 크기

8 Level Garbage collection



Constructor

```

▼ TestObjectSize
  ▼ TestObjectSize @377475
    ► map :: system / Map @381269
    ► __proto__ :: Object @377473 []
    ► name :: "CodingMax" @378527 [] ←
    ► age :: smi number @378845 []
  
```

Retainers

Object

```

► 1 in (constant pool) @382093
► 1 in (constant pool) @382043
► 232 / DevTools console in (Global handles) @29
► 4257 in (Internalized strings) @5
▼ name in TestObjectSize @377475 ←
  ▼ myObj in system / Context @371111 ←
    ▼ 3 in (internal array)[] @365233
      ▼ script_context_table in system / NativeContext @345349
        ▼ 14 / gin::ContextHolder::context_ in (Global handles) @29
          [13] in (GC roots) @3 ←
    ► 441 / V8PerContextData::context_ in (Global handles) @29
    ► 443 / extensions::ScriptContext::v8_context_ in (Global handles) @29
  
```

Constructor

```

▼ TestObjectSize
  ▼ TestObjectSize @377475
    ► map :: system / Map @381269
    ► __proto__ :: Object @377473 []
    ► name :: "CodingMax" @378527 []
    ► age :: smi number @378845 [] ←
  
```

Retainers

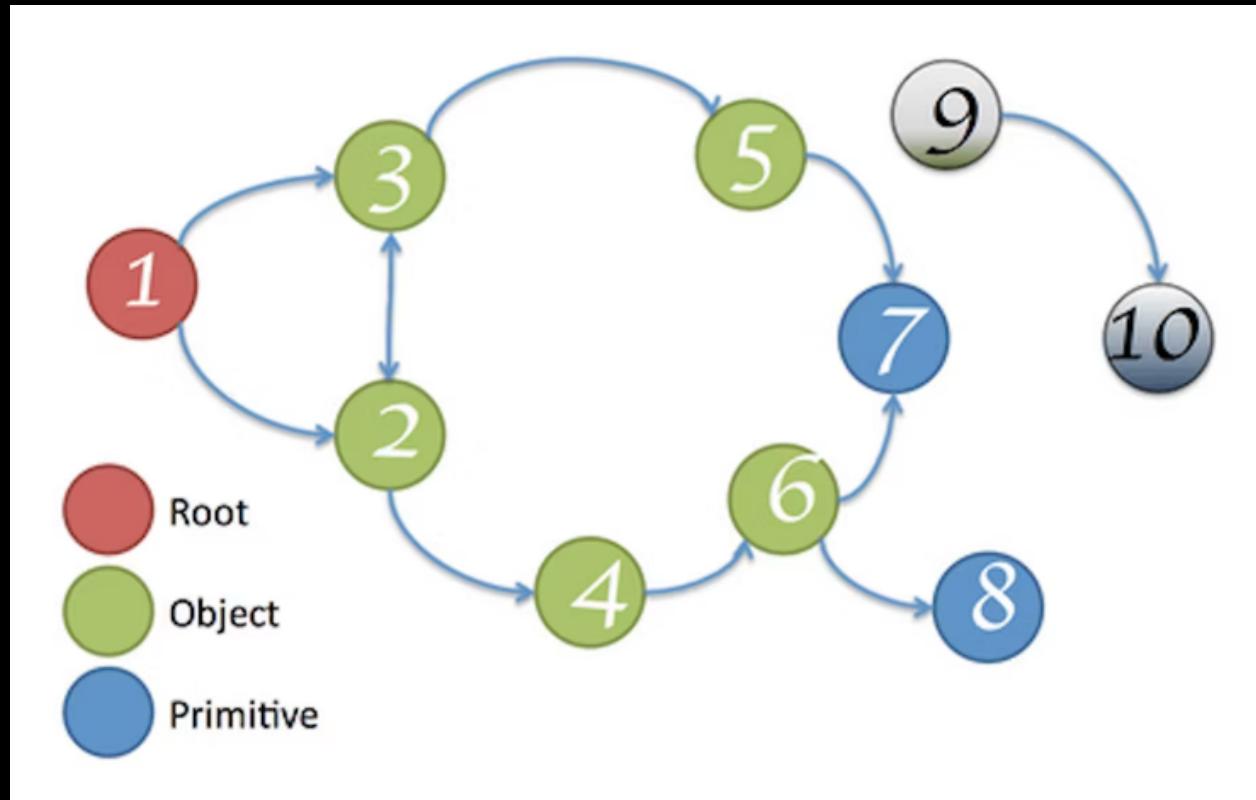
Object

```

► <symbol> in SyntaxError @345481
► <symbol> in SyntaxError @345481
► DOCUMENT_TYPE_NODE in Node @350997 []
► DOCUMENT_TYPE_NODE in Node @175923 []
► DOCUMENT_TYPE_NODE in Node() @347231 []
► DOCUMENT_TYPE_NODE in Node() @174853 []
► INUSE_ATTRIBUTE_ERR in DOMException @347837 []
► INUSE_ATTRIBUTE_ERR in DOMException() @347131 []
► NAMESPACE_RULE in CSSRule @373911 []
► NAMESPACE_RULE in CSSRule() @346161 []
► SVG_LENGTHTYPE_PC in SVGLength() @346807 []
► SVG_LENGTHTYPE_PC in SVGLength @358925 []
► SVG_PRESERVEASPECTRATIO_XMAXYMAX in SVGPreserveAspectRatio() @346841 []
► SVG_PRESERVEASPECTRATIO_XMAXYMAX in SVGPreserveAspectRatio @358617 []
▼ age in TestObjectSize @377475 ←
  ▼ myObj in system / Context @371111 ←
    ▼ 3 in (internal array)[] @365233
      ▼ script_context_table in system / NativeContext @345349
        ▼ 14 / gin::ContextHolder::context_ in (Global handles) @29
          [13] in (GC roots) @3 ←
    ► 441 / V8PerContextData::context_ in (Global handles) @29
    ► 443 / extensions::ScriptContext::v8_context_ in (Global handles) @29
  
```

- ① GC Root는 V8 엔진 Native 코드에서 외부에 있는 자바스크립트 객체를 참조하기 위한 핸들. 예를 들어, GlobalThis로 브라우저에서는 window 가 GC Root가 된다.
- ② Primitive 는 항상 Leaf 가 된다. Primitive는 참조형이 아니어서 다른 객체를 참조할 수 없기 때문이다.

8 Level Garbage collection



Constructor	Distance	S
▼ TestObjectSize	4	
▼ TestObjectSize @377475	4	
► map :: system / Map @381269	5	
► __proto__ :: Object @377473 □	3	
► name :: "CodingMax" @378527 □	5	
► age :: smi number @378845 □	3	
Retainers		
Object	Distance	S
► 232 / DevTools console in (Global handles) @29	-	
► 4257 in (Internalized strings) @5	-	
► 1 in (constant pool) @382093	-	
► 1 in (constant pool) @382043	-	
▼ name in TestObjectSize @377475	4	
▼ myObj in system / Context @371111	3	
► 3 in (internal array) [] @365233	3	
▼ script_context_table in system / NativeContext @345349	2	
▼ 14 / gin::ContextHolder::context_ in (Global handles) @29	-	
[13] in (GC roots) @3	-	

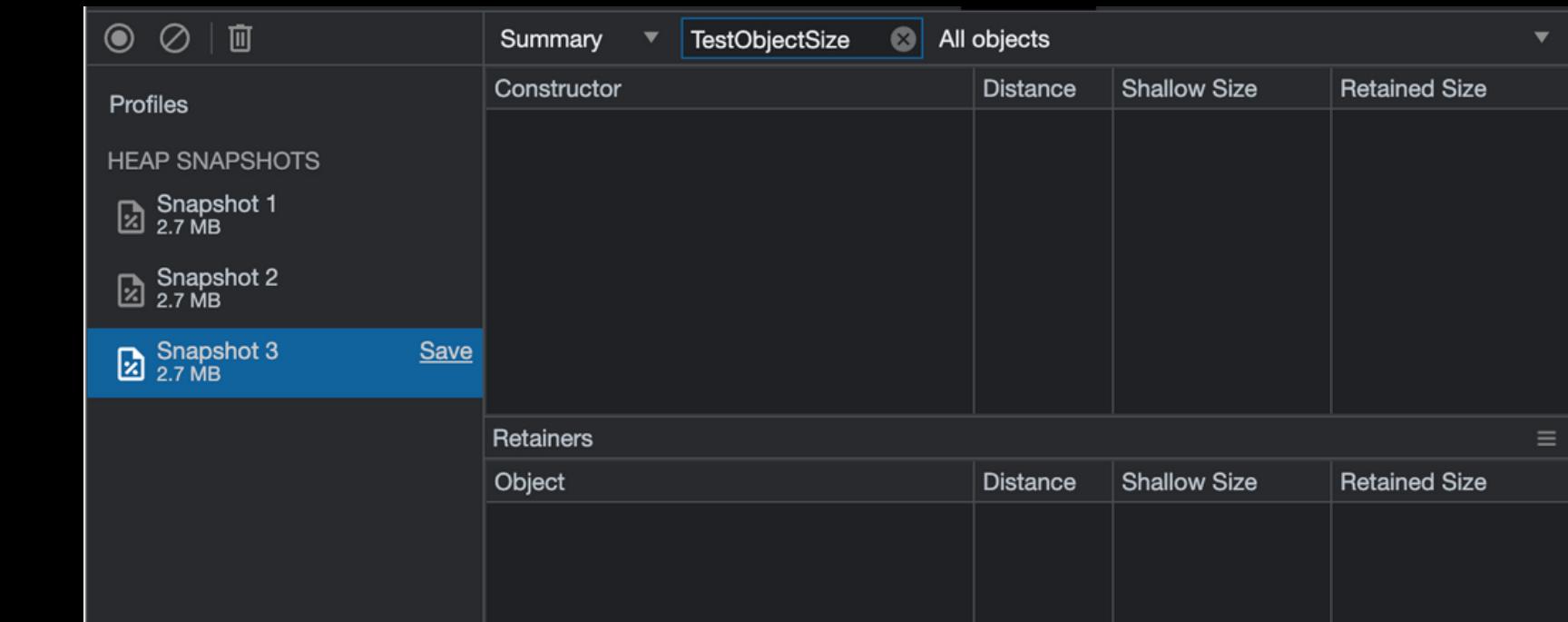
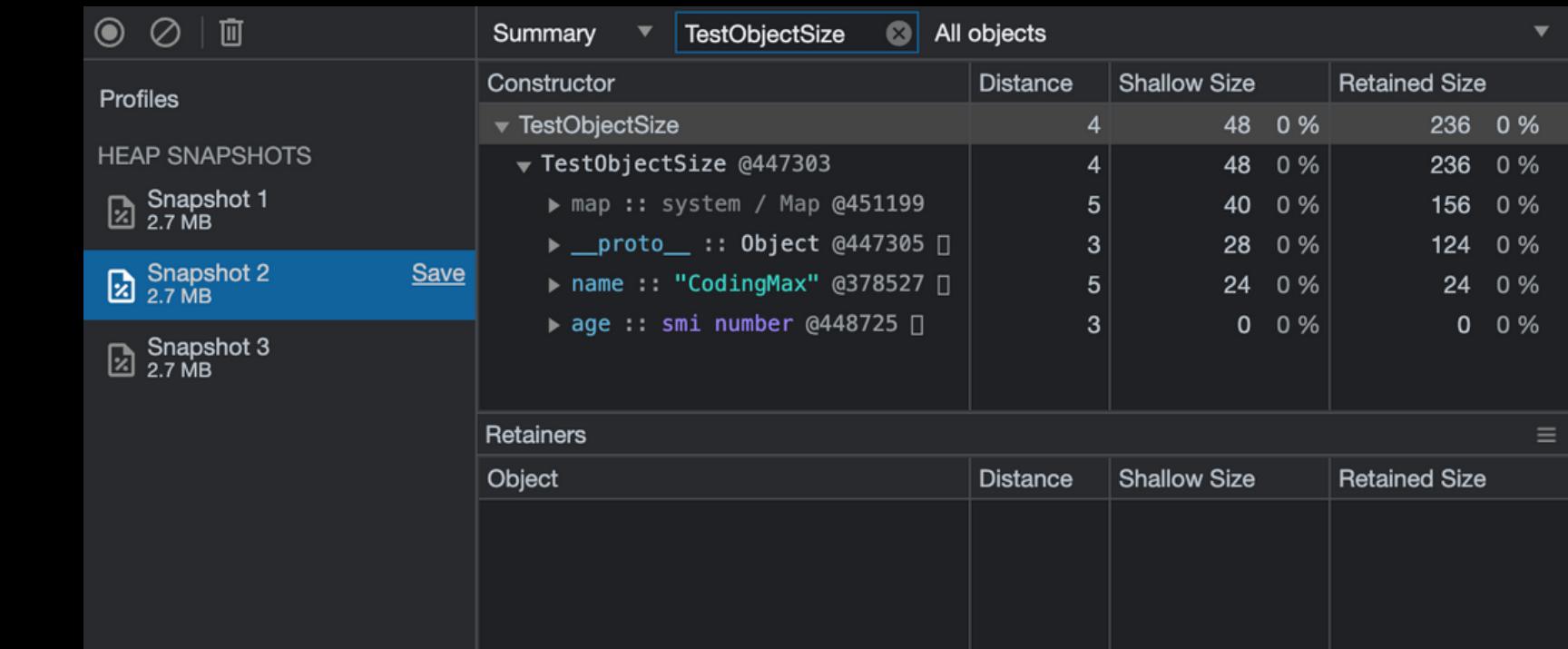
- ① Distance는 GC Root에서 해당 속성까지의 도달 거리
- ② GC Root에서 도달할 수 없다면 GC 대상이 된다. 그래서 9, 10번이 GC된다.

8 Level Garbage collection

```
Test GC

function TestObjectSize() {
    this.age = 10;
}
let myObj = new TestObjectSize();
myObj.name = 'CodingMax';

// GC
myObj = null;
```



- ① **myObj = null** 을 설정하면 GC Root에서
도달할 수 없기 때문에
myObj와 속성 name, age 모두 GC 된다.

8 Level Garbage collection

```
Test GC

function TestObjectSize() {
    this.age = 10;
}
let myObj = new TestObjectSize();
myObj.name = 'CodingMax';

// Prevent name property to be GC.
let objNameHolder = new TestObjectSize();
objNameHolder.name = myObj.name;
myObj = null;
```

Summary ▾ TestObjectSize ✎ All objects				
Constructor	Distance	Shallow Size	Retained Size	
▼ TestObjectSize	4	48 0 %	200 0 %	
▼ TestObjectSize @545935	4	48 0 %	200 0 %	
▶ __proto__ :: Object @544871	3	28 0 %	124 0 %	
▶ map :: system / Map @544885	5	40 0 %	120 0 %	
▶ name :: "CodingMax" @378527	5	24 0 %	24 0 %	
▶ age :: smi number @528939	3	0 0 %	0 0 %	

Retainers				
Object	Distance	Shallow Size	Retained Size	
▼ name in TestObjectSize @545935	4	48 0 %	200 0 %	
▼ myObj in system / Context @533041	3	20 0 %	220 0 %	
▶ context in TestObjectSize() @5:	2	32 0 %	520 0 %	
▶ TestObjectSize in Window / :	1	36 0 %	1 338 384 48 %	
▶ constructor in Object @54487	3	28 0 %	124 0 %	

Summary ▾ TestObjectSize ✎ All objects				
Constructor	Distance	Shallow Size	Retained Size	
▼ TestObjectSize ×2	4	96 0 %	96 0 %	
▼ TestObjectSize @545935	4	48 0 %	48 0 %	
▶ map :: system / Map @544885	5	40 0 %	164 0 %	
▶ __proto__ :: Object @544871	3	28 0 %	124 0 %	
▶ name :: "CodingMax" @378527	5	24 0 %	24 0 %	
▶ age :: smi number @553901	3	0 0 %	0 0 %	
▼ TestObjectSize @556541	5	48 0 %	48 0 %	
▶ map :: system / Map @544885	5	40 0 %	164 0 %	
▶ __proto__ :: Object @544871	3	28 0 %	124 0 %	
▶ name :: "CodingMax" @378527	5	24 0 %	24 0 %	
▶ age :: smi number @553901	3	0 0 %	0 0 %	

Retainers				
Object	Distance	Shallow Size	Retained Size	
▼ objNameHolder in system / Context	4	20 0 %	108 0 %	
▶ 4 in (internal array) [] @556355	3	40 0 %	472 0 %	

- 1 새로운 객체를 만들고 `myObj.name` 속성을 `objNameHolder.name` 속성으로 참조한다.

8 Level Garbage collection

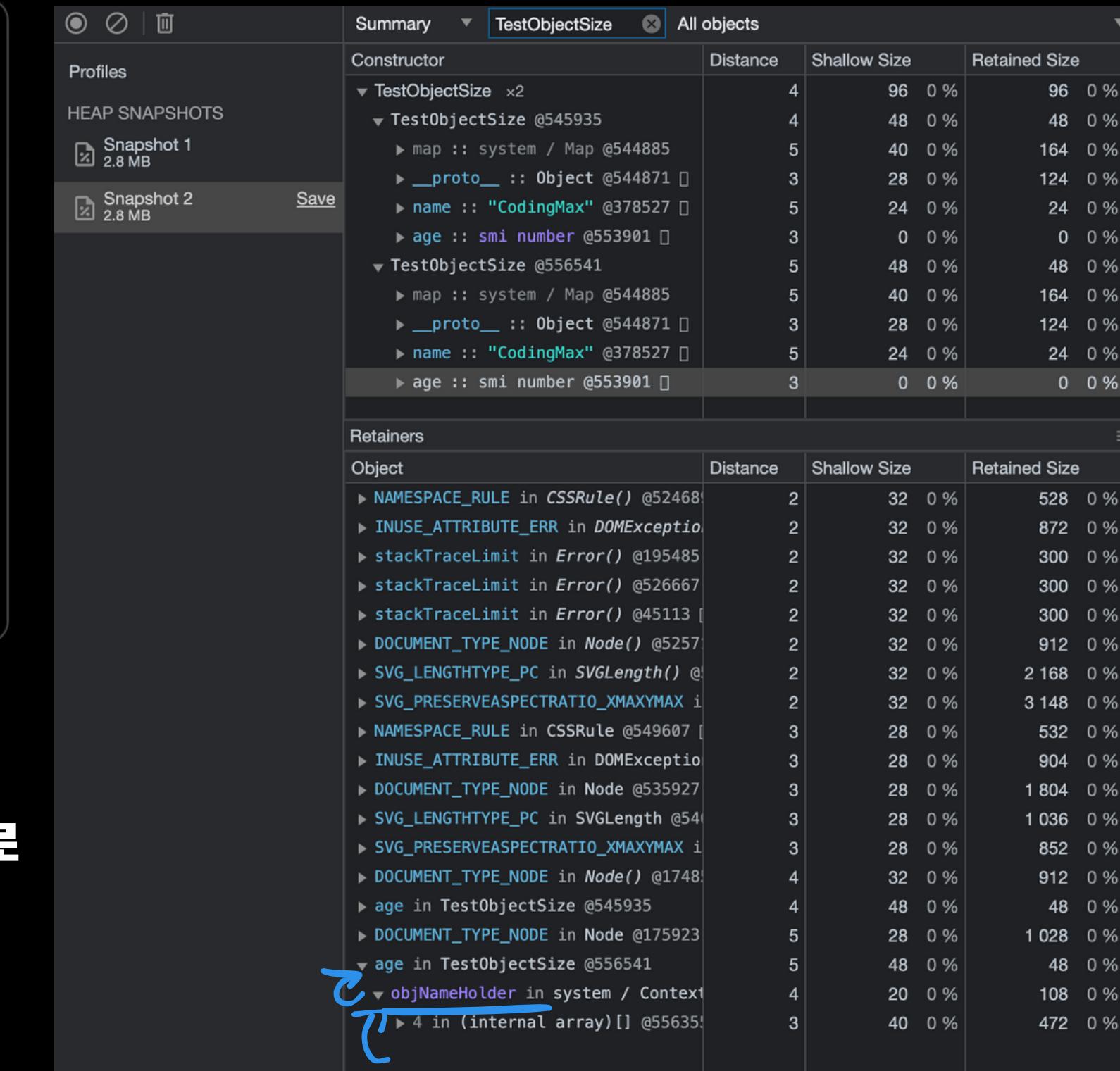
```
Test GC

function TestObjectSize() {
    this.age = 10;
}
let myObj = new TestObjectSize();
myObj.name = 'CodingMax';

// Prevent name property to be GC.
let objNameHolder = new TestObjectSize();
objNameHolder.name = myObj.name;
myObj = null;
```

1

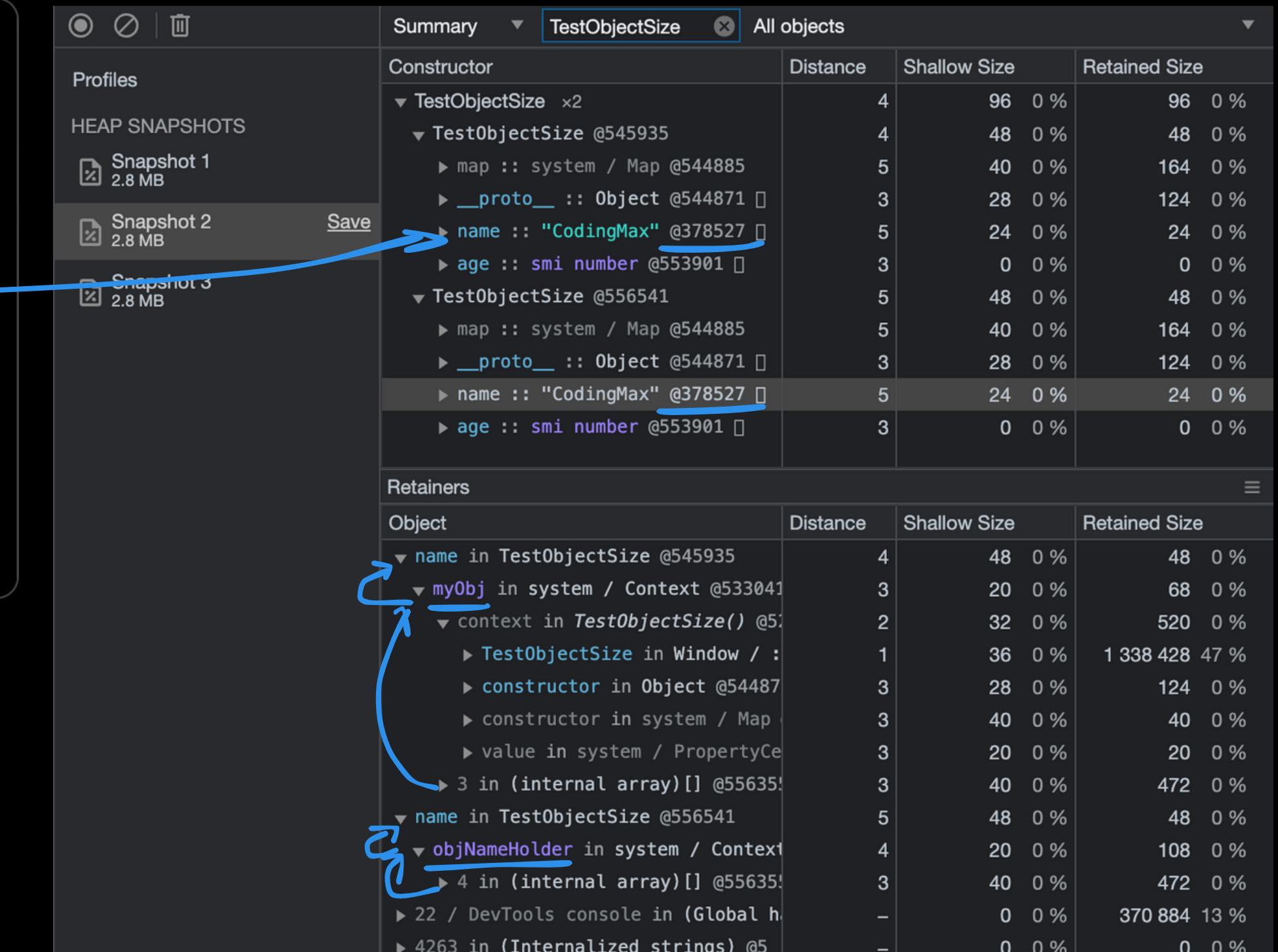
objNameHolder의 age속성은 objNameHolder에 속해 있다. 그래서 GCRoot는 objNameHolder.age로 접근할 수 있다.



8 Level Garbage collection

```
Test GC

function TestObjectSize() {
    this.age = 10;
}
let myObj = new TestObjectSize();
myObj.name = 'CodingMax';
// Prevent name property to be GC.
let objNameHolder = new TestObjectSize();
objNameHolder.name = myObj.name;
myObj = null;
```



- 1 objNameHolder의 name 속성은 myObj에 속해 있고 objNameHolder.name이 참조하고 있다. 그래서 GCRoot는 myObj.name과 objNameHolder.name 두 경로로 name에 접근할 수 있다.
- 2 name 속성의 V8 힙 주소는 두 객체 모두 378527이다

V8 Level Garbage collection

```

Test GC

function TestObjectSize() {
    this.age = 10;
}

let myObj = new TestObjectSize();
myObj.name = 'CodingMax';

// Prevent name property to be GC.
let objNameHolder = new TestObjectSize();
objNameHolder.name = myObj.name;
myObj = null;

```

- 1 myObj는 GC되었지만 name 속성은 objNameHolder를 통해 접근할 수 있기 때문에 GC 되지 않고 objNameHolder 속성으로 남아 있다.
- 2 name 속성의 V8 힙 주소는 그대로 378527이다.

	Summary		TestObjectSize	
Profiles	Constructor	Distance	Shallow Size	Retained Size
HEAP SNAPSHOTS	TestObjectSize	5	48 0 %	244 0 %
Snapshot 1	TestObjectSize @556541	5	48 0 %	244 0 %
Snapshot 2	▶ map :: system / Map @544885	6	40 0 %	164 0 %
Snapshot 3	▶ __proto__ :: Object @544871	3	28 0 %	124 0 %
	▶ name :: "CodingMax" @378527	6	24 0 %	24 0 %
	▶ age :: smi number @557451	3	0 0 %	0 0 %
	Save			
Retainers	Object	Distance	Shallow Size	Retained Size
	objNameHolder in system / Context	4	20 0 %	304 0 %
	▶ 4 in (internal array)[] @556355	3	40 0 %	668 0 %

	Summary		TestObjectSize	
Profiles	Constructor	Distance	Shallow Size	Retained Size
HEAP SNAPSHOTS	TestObjectSize	5	48 0 %	244 0 %
Snapshot 1	TestObjectSize @556541	5	48 0 %	244 0 %
Snapshot 2	▶ map :: system / Map @544885	6	40 0 %	164 0 %
Snapshot 3	▶ __proto__ :: Object @544871	3	28 0 %	124 0 %
	▶ name :: "CodingMax" @378527	6	24 0 %	24 0 %
	▶ map :: system / Map (OneByte)	4	40 0 %	40 0 %
	▶ age :: smi number @557451	3	0 0 %	0 0 %
	Save			
Retainers	Object	Distance	Shallow Size	Retained Size
	name in TestObjectSize @556541	5	48 0 %	244 0 %
	▶ objNameHolder in system / Context	4	20 0 %	304 0 %
	▶ 4 in (internal array)[] @556355	3	40 0 %	668 0 %
	script_context_table in system	2	1 144 0 %	87 520 3 %
	▶ 4263 in (Internalized strings) @5	-	0 0 %	0 0 %
	▶ 1 in (constant pool) @451665	-	32 0 %	56 0 %
	▶ 1 in (constant pool) @451529	-	32 0 %	56 0 %
	▶ 1 in (constant pool) @480793	-	32 0 %	56 0 %

즐거운 코딩 경험
〈CODINGMAX/〉