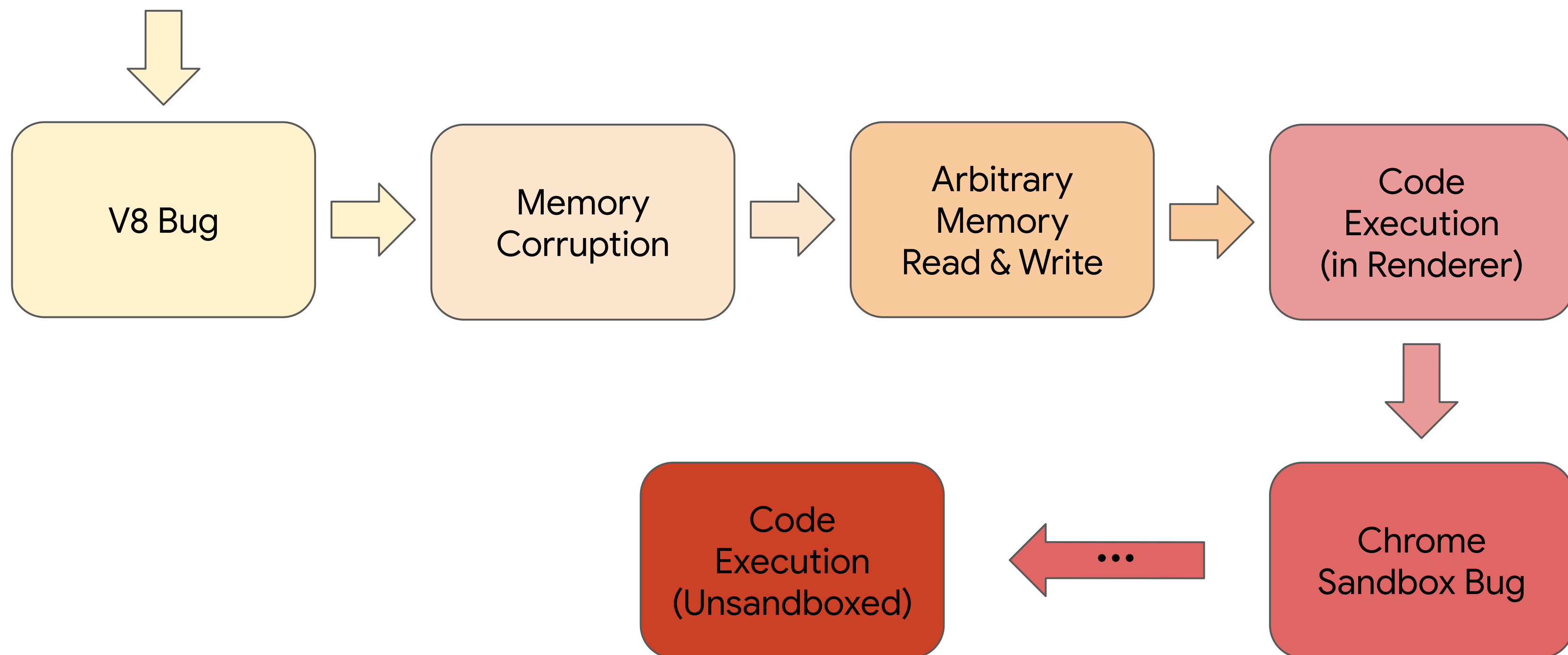
The V8 Heap Sandbox

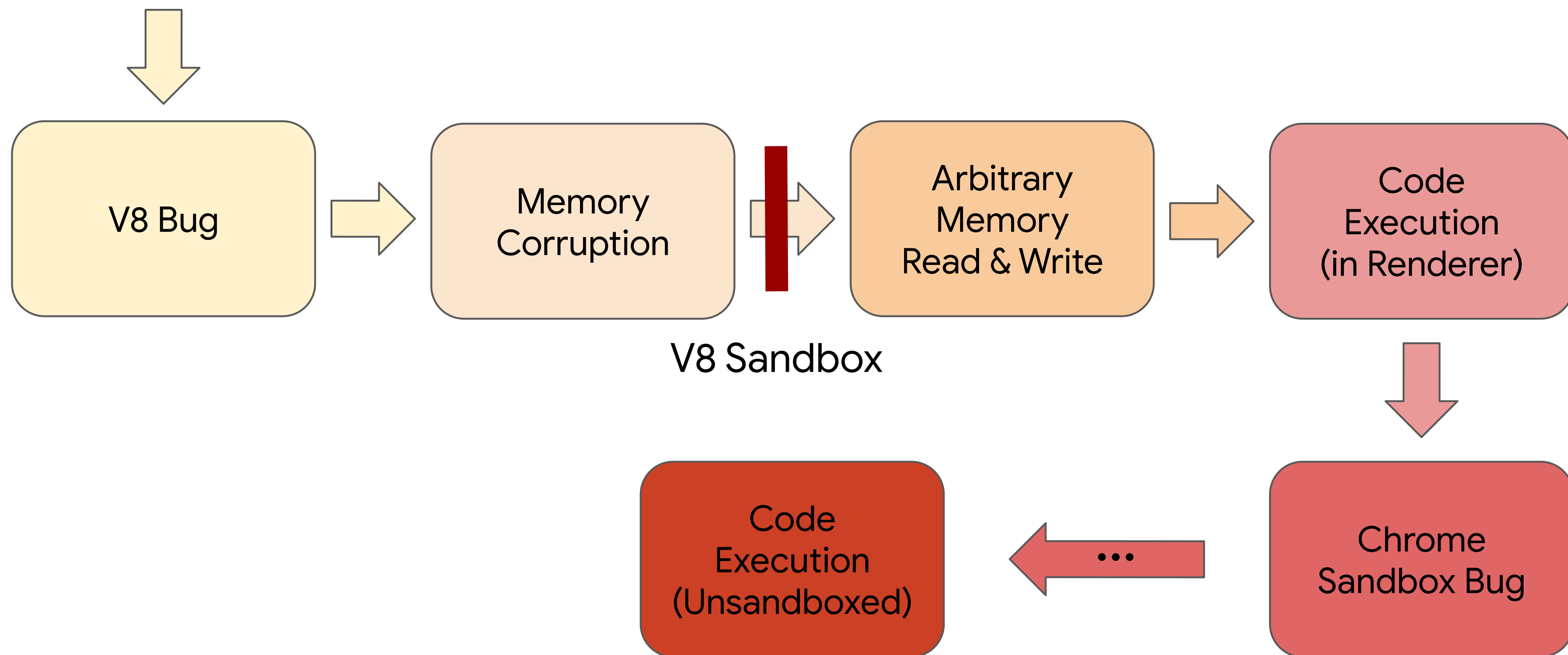
OffensiveCon 2024

Samuel Groß - Google V8 Security

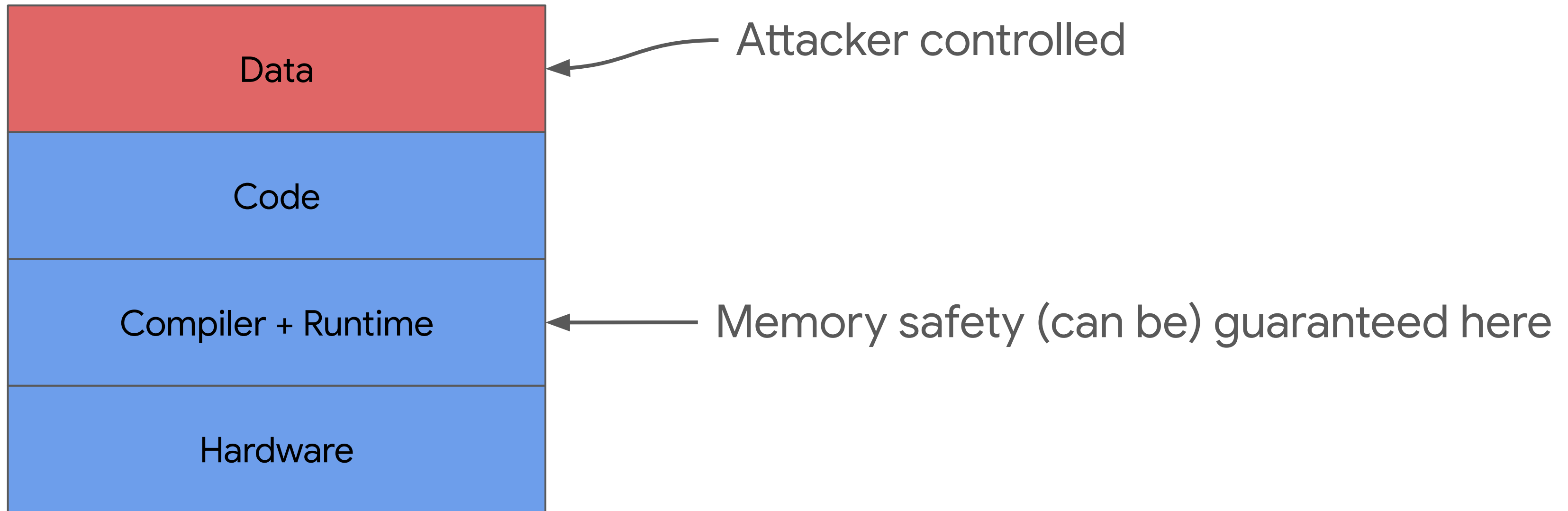
Typical Exploit Flow



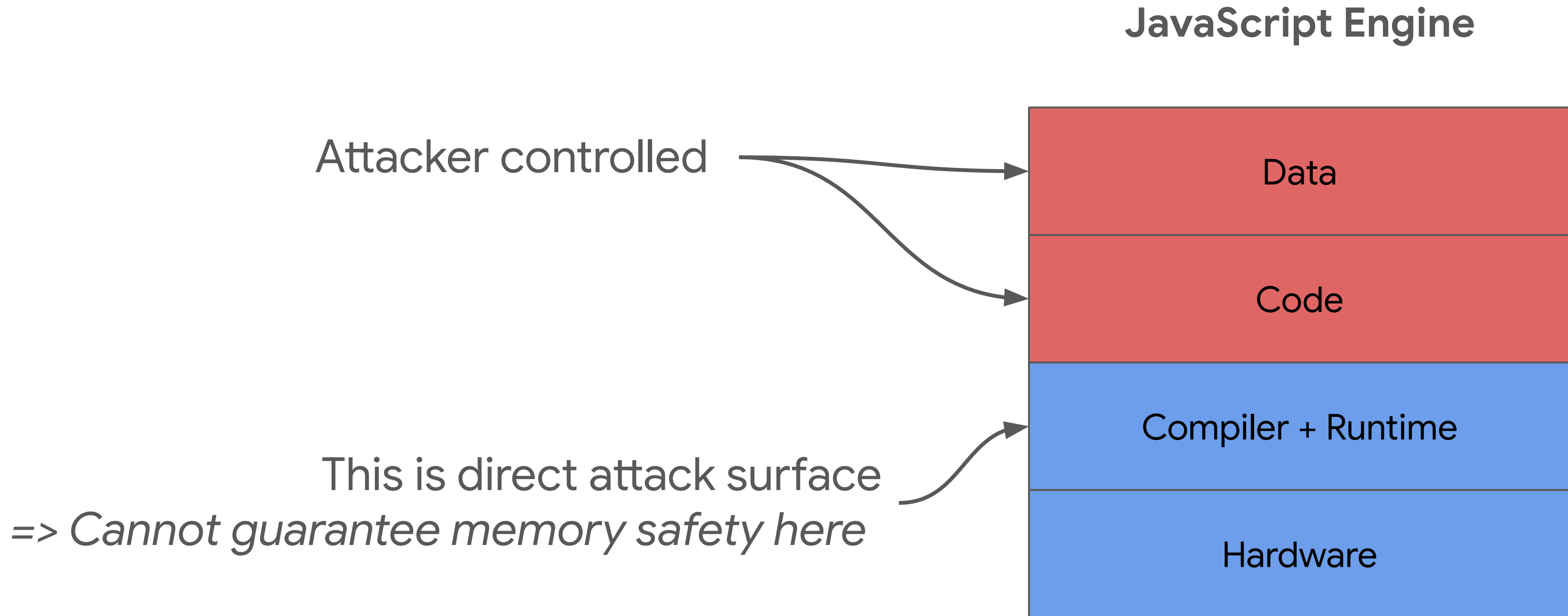
Typical Exploit Flow



Why JavaScript Engine Security is hard



Why JavaScript Engine Security is hard



Why JavaScript Engine Security is hard

- Compiler-based memory safety doesn’t work if compiler is attack surface
  - => Any logic bug can potentially turn into memory corruption
- Disabling optional compilers solves only a part of the problem
  - Plenty of bugs elsewhere (e.g. runtime) …
  - … and it is very slow :(

=> Writing a high-performance, memory-safe JS engine is **hard**

High-performance, memory-safe JavaScript engine?

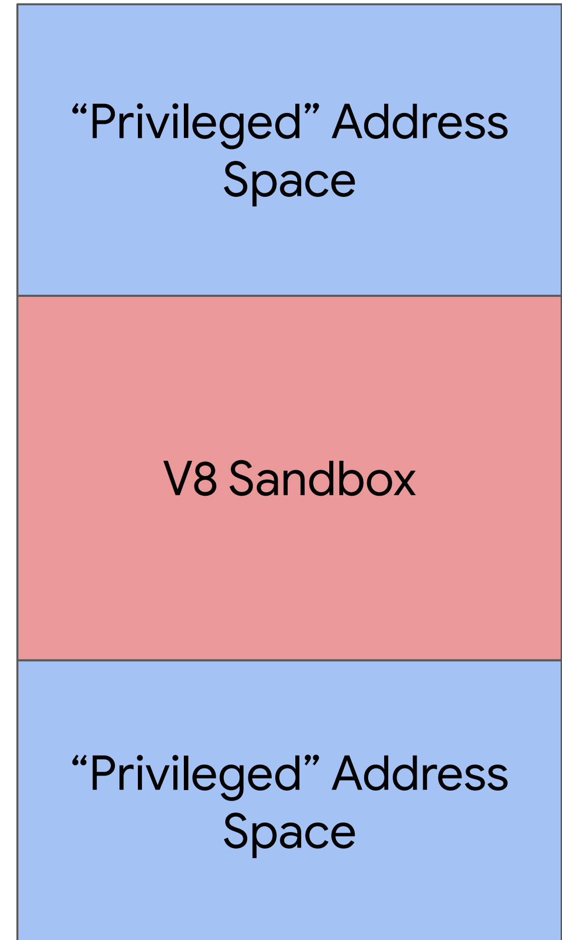
A different approach...

Idea:

- Accept that bugs will happen and that memory will be corrupted
- Limit which memory can be corrupted
- Make that a security boundary

=> Result: an in-process sandbox

Can corrupt memory here



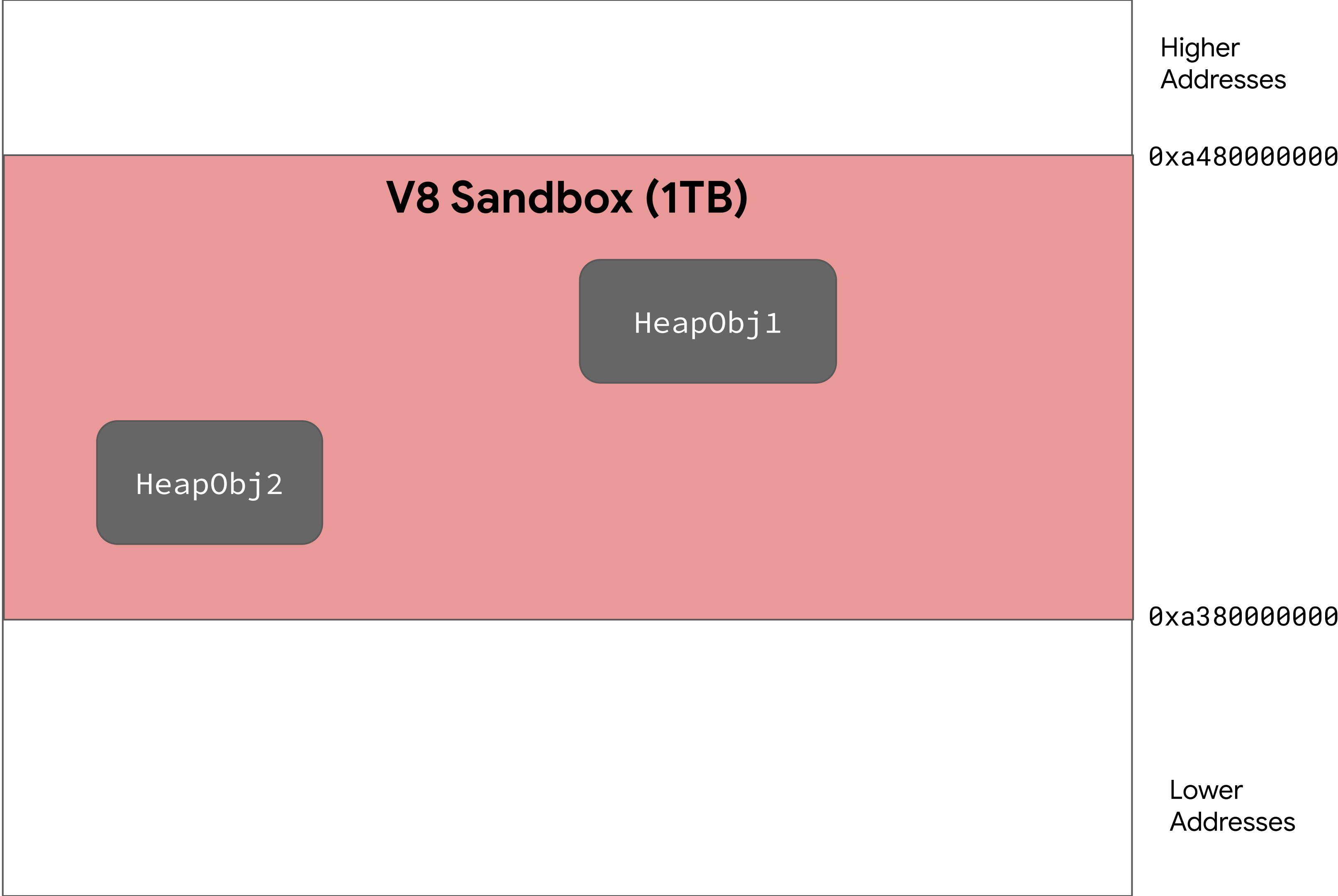
Higher Addresses

0xa48000000000

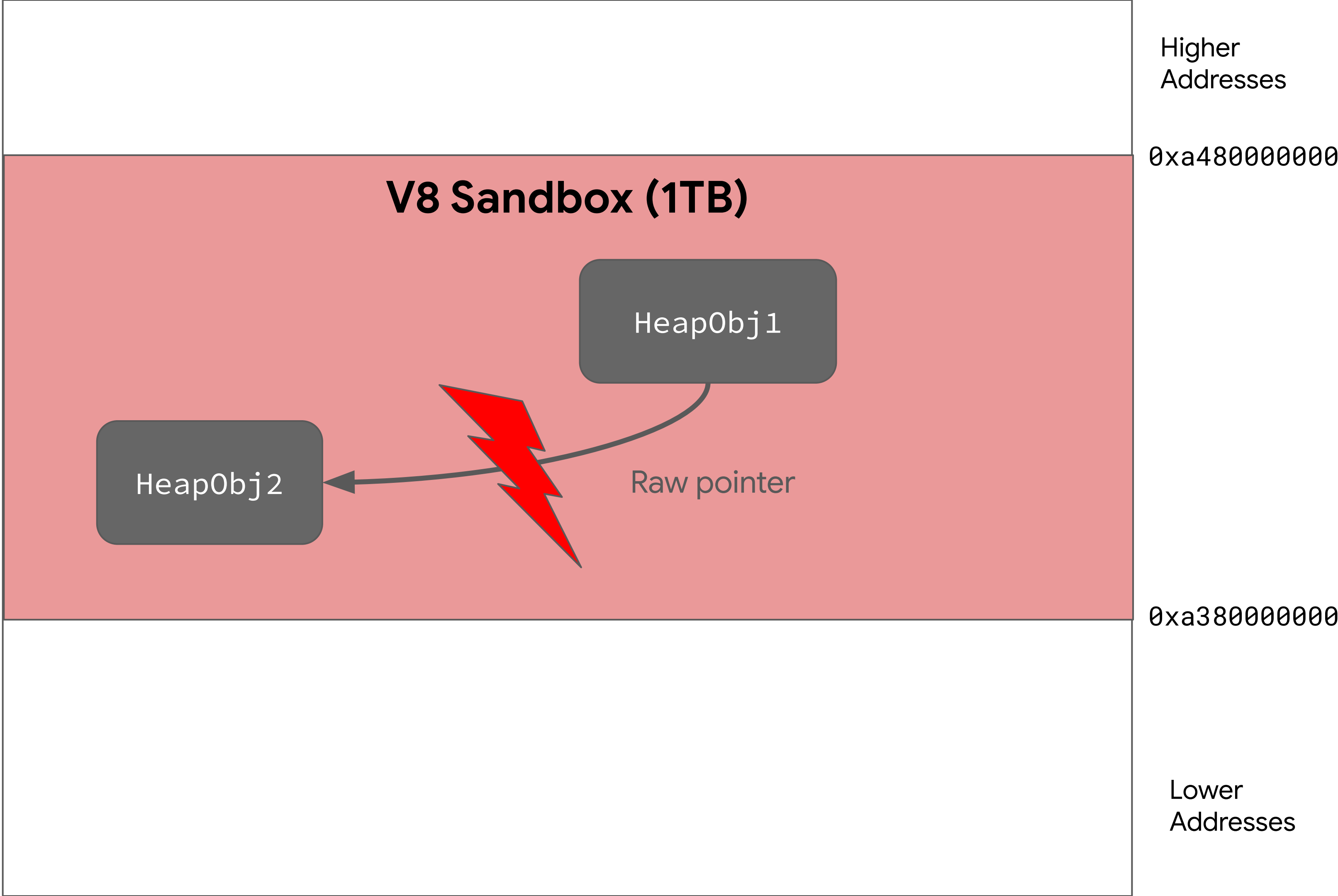
V8 Sandbox (1TB)

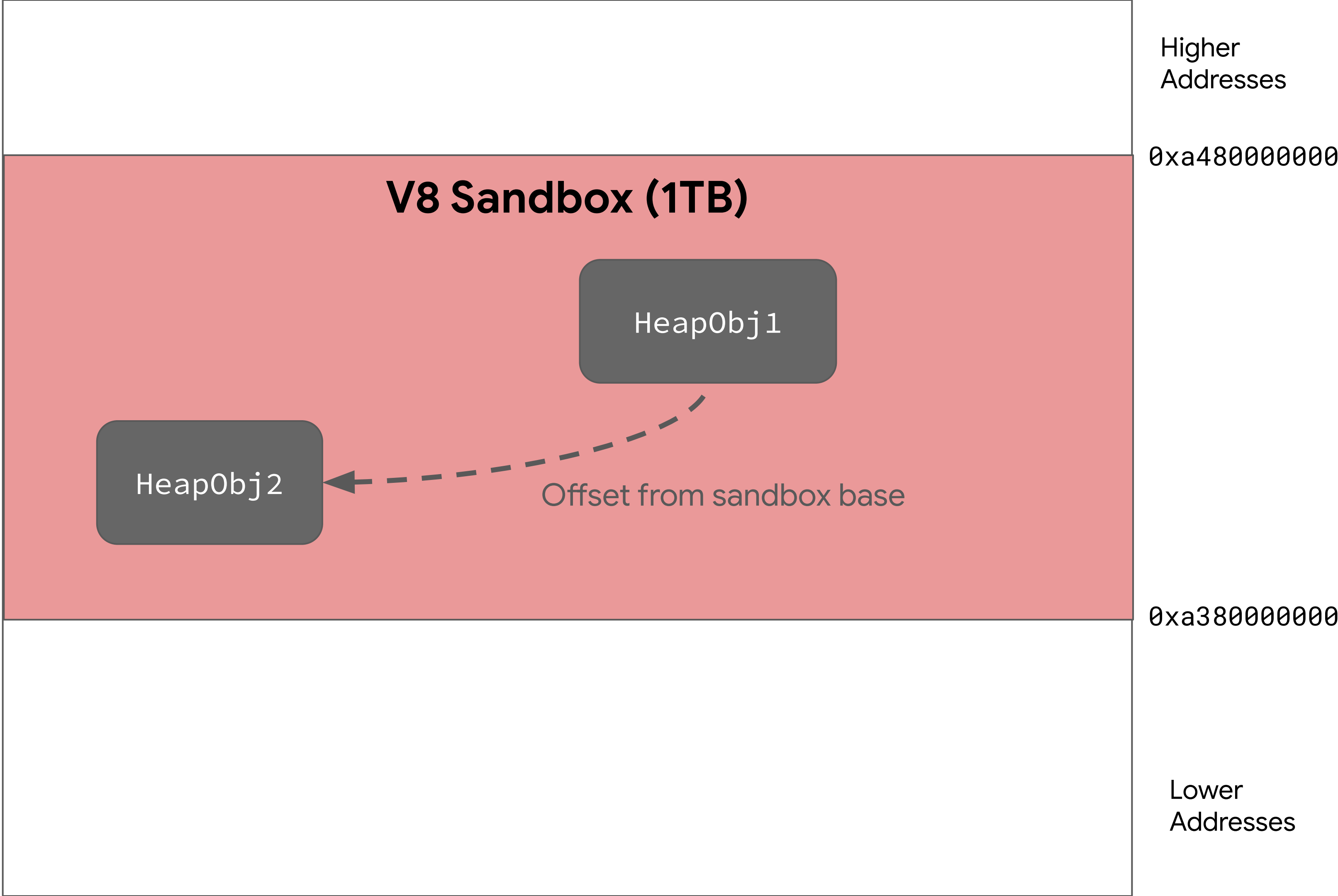
0xa38000000000

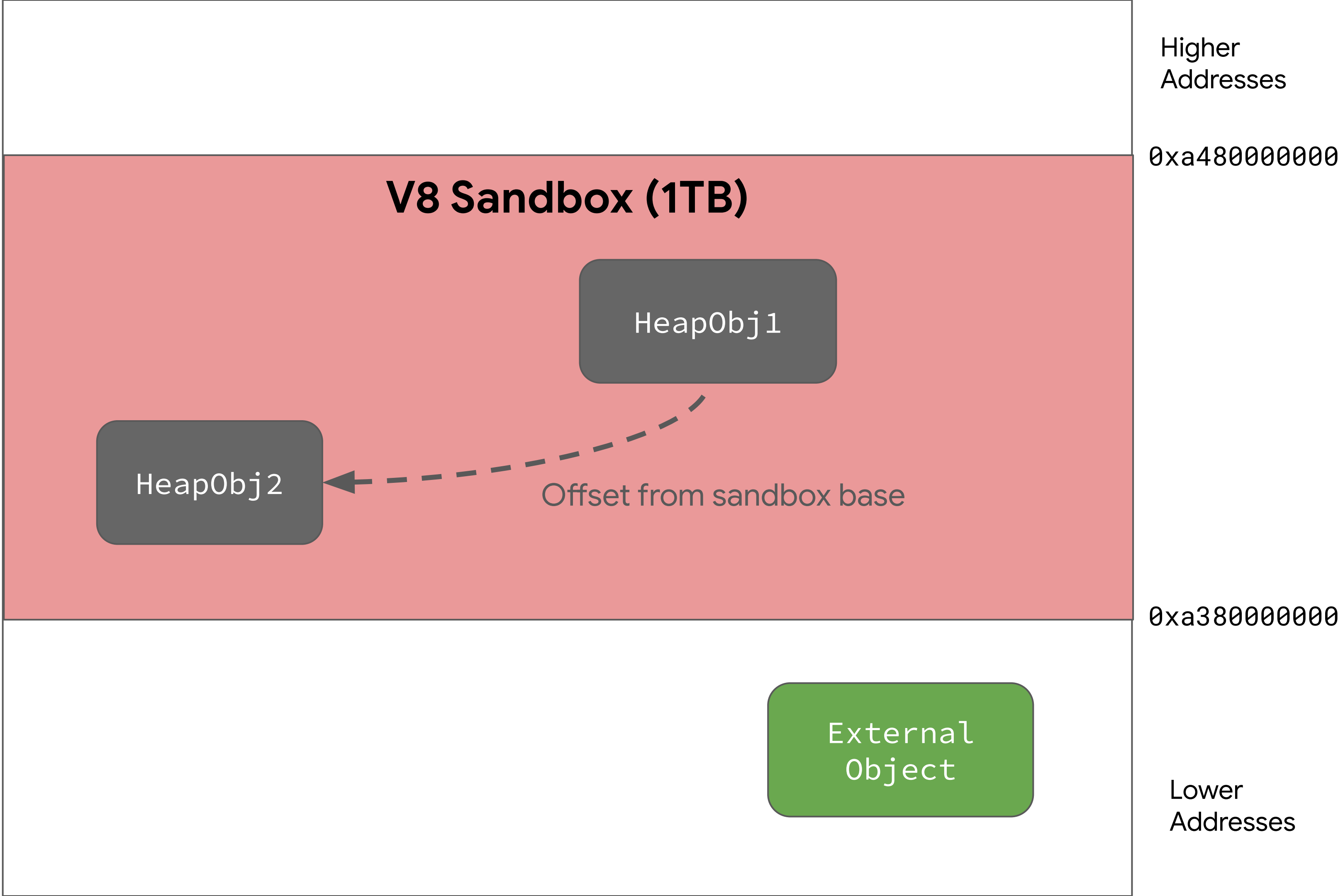
Lower Addresses



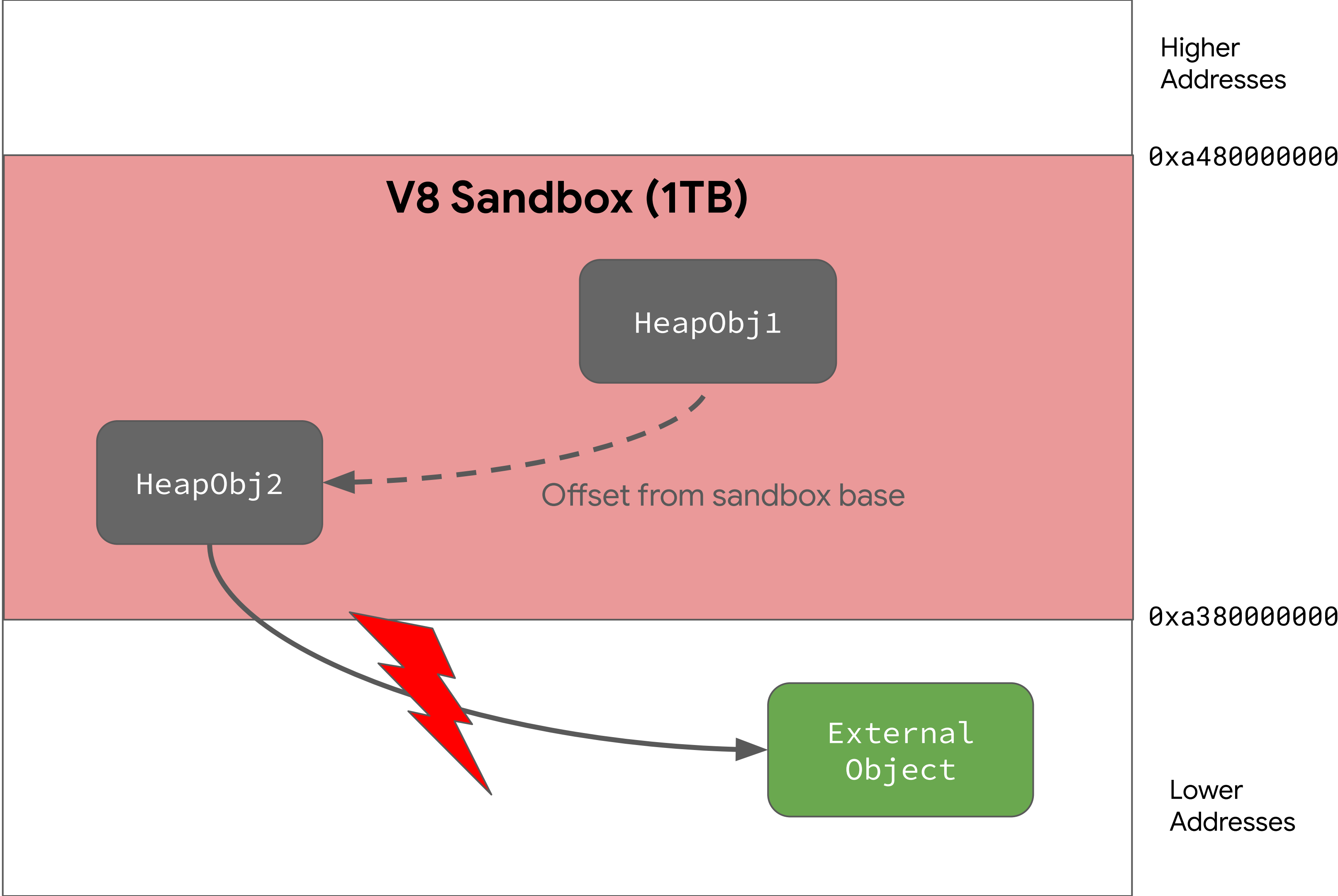
Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
HeapObj1
HeapObj2
0xa38000000000
Lower
Addresses


Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
HeapObj1
HeapObj2
Raw pointer
0xa38000000000
Lower
Addresses


Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
HeapObj1
HeapObj2
Offset from sandbox base
0xa38000000000
Lower
Addresses



Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
HeapObj1
HeapObj2
Offset from sandbox base
0xa38000000000
External
Object
Lower
Addresses


Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
HeapObj1
HeapObj2
Offset from sandbox base
0xa38000000000
External
Object
Lower
Addresses

Higher Addresses

0xa48000000000

V8 Sandbox (1TB)

HeapObj1

HeapObj2

Offset from sandbox base

Index

0xa38000000000

External Ptr Table	
0	Type + Pointer
1	Type + Pointer

External Object

Lower Addresses

Higher
Addresses

0xa48000000000

V8 Sandbox (1TB)

HeapObj1

HeapObj2

Index

Basically: ban all raw pointers!

0xa38000000000

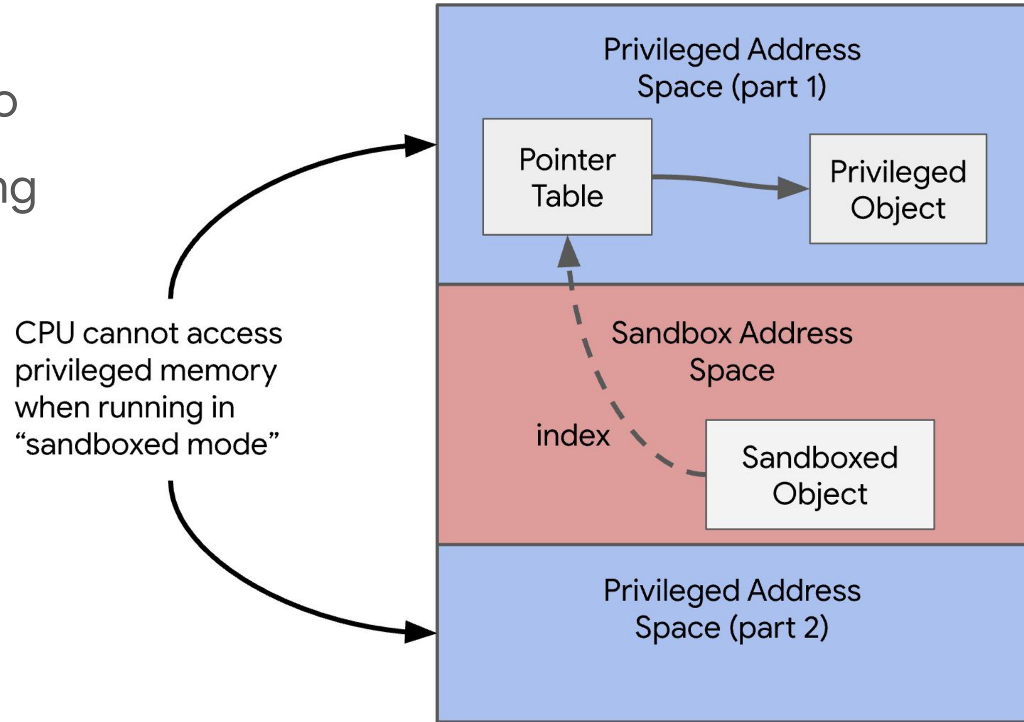
External Ptr Table	
0	Type + Pointer
1	Type + Pointer

External
Object

Lower
Addresses

Sandbox with Hardware Support?

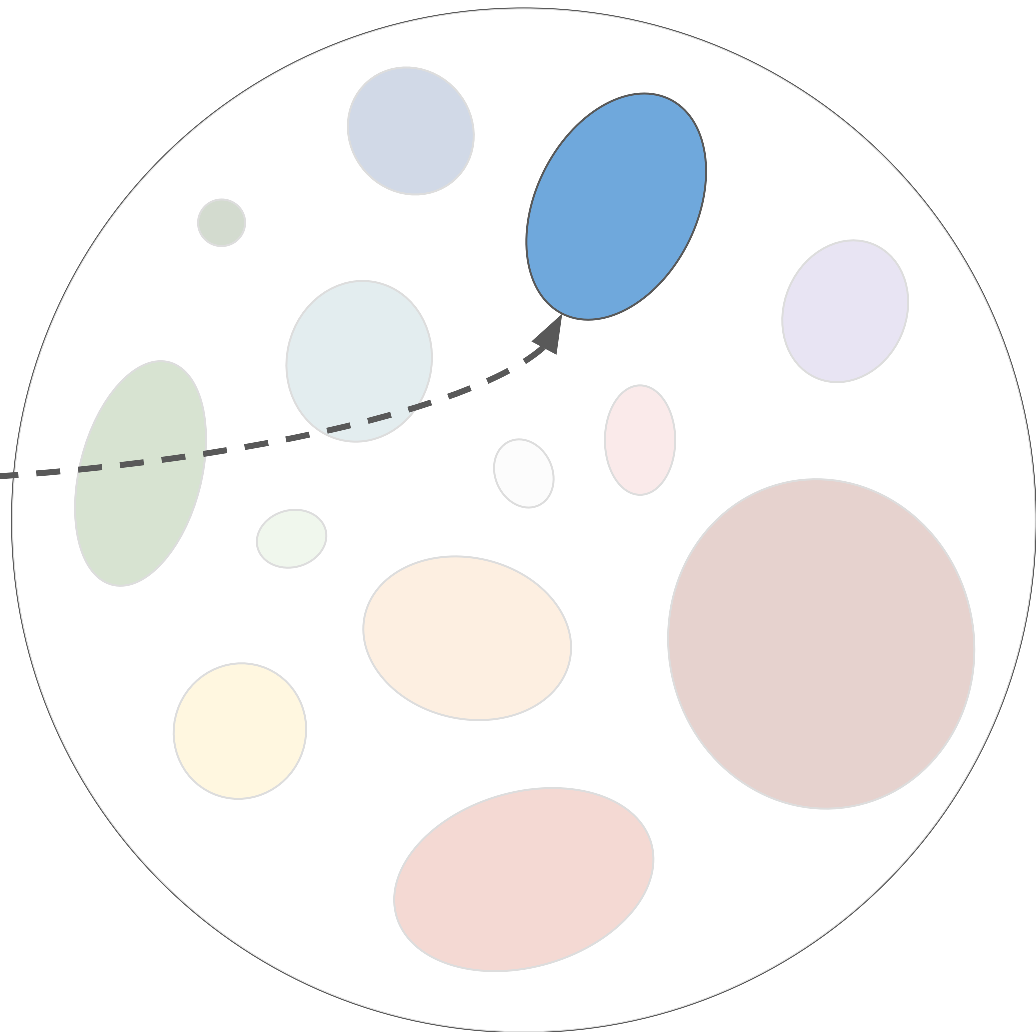
- In the future, may be possible to "drop privileges" when executing JS or Wasm code
- Would be very similar to userspace/kernel split
- Ideally: want to be able to run untrusted *machine code*



High-performance, memory-safe JavaScript engine!
(with a sandbox)

Lots of smaller, simpler problems

Performance



Performance

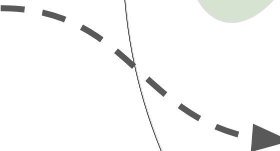
- Sandbox building blocks are fundamentally cheap
  - Offsets require just an additional add or shift+add instruction
  - Pointer table requires one additional memory load for external references
- => Benefit over other memory safety technologies
- Today: overhead of sandbox is only around 1% on popular benchmarks
  - => Can be (and is already) enabled by default!

```
ldr x3, [x0, #7]
```

Sandboxification (`x28` always contains the sandbox base)

```
ldr x3, [x0, #7]
add x3, x28, x3, lsr #24
```

Untrusted Indices



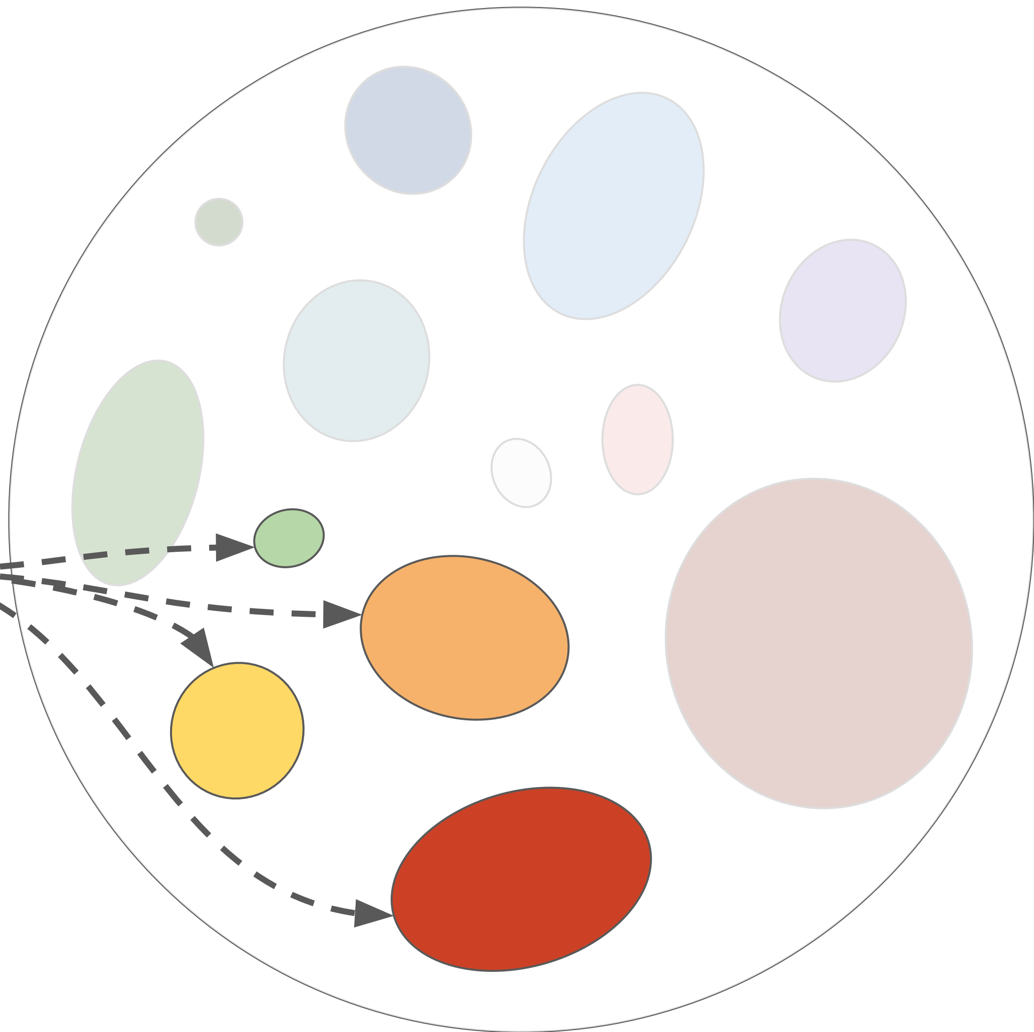
Untrusted Indices

```
Tagged<MyHeapObject> obj = ...;
int idx = obj->get_the_index();
int val = obj->get_the_value();
some_global_array[idx] = val;
```

Untrusted Indices

```
Tagged<MyHeapObject> obj = ...;
uint idx = obj->get_the_index();
int val = obj->get_the_value();
SBXCHECK(idx < some_global_array_size);
some_global_array[idx] = val;
```

Broken Invariants



Broken Invariants

```
std::vector<std::string> JSObject::GetPropertyNames() {
  int num_properties = TotalNumberOfProperties();
  std::vector<std::string> properties(num_properties);

  for (int i = 0; i < NumberOfInObjectProperties(); i++) {
    properties[i] = GetNameOfInObjectProperty(i);
  }

  // Deal with the other types of properties
  // ...
```

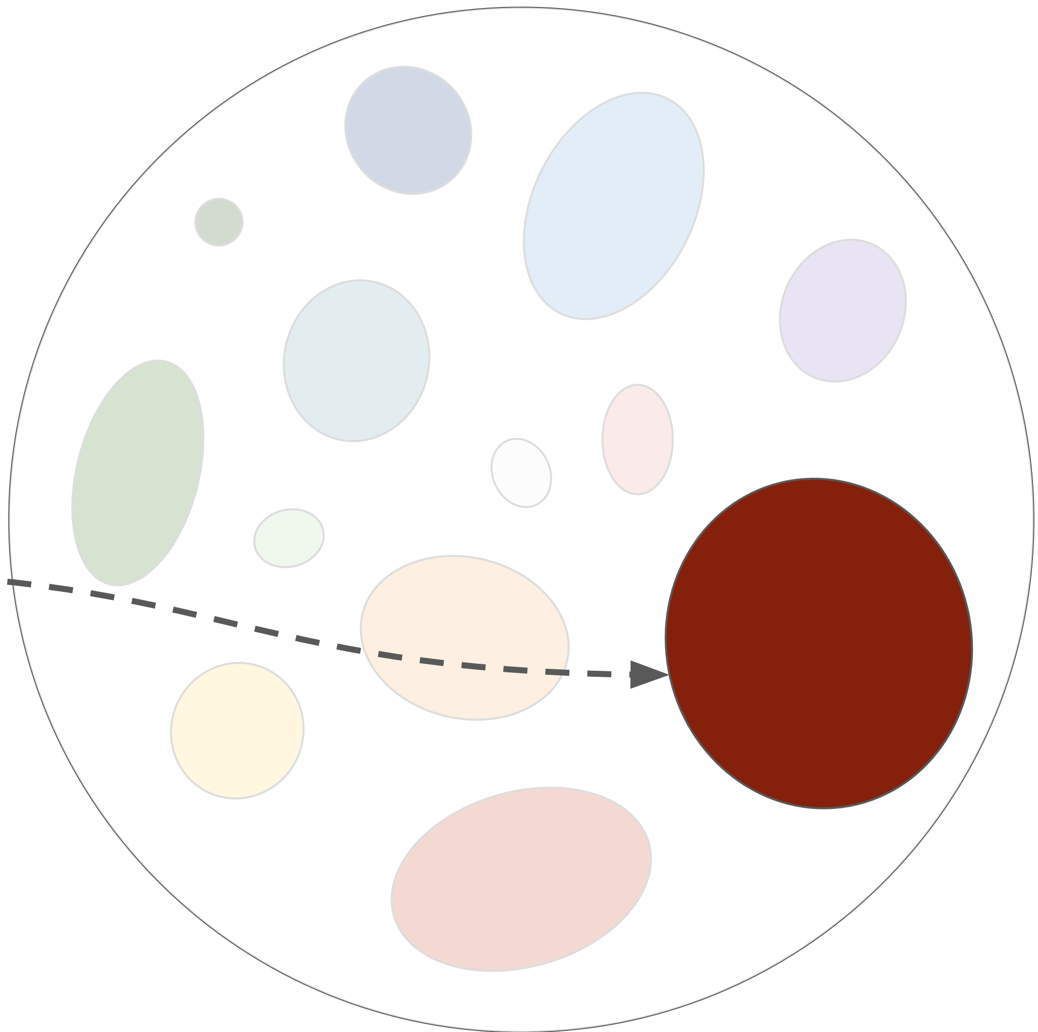
Broken Invariants

```
std::vector<std::string> JSObject::GetPropertyNames() {
  int num_properties = TotalNumberOfProperties();
  std::vector<std::string> properties(num_properties);

  for (int i = 0; i < NumberOfInObjectProperties(); i++) {
    SBXCHECK(i < properties.size());
    properties[i] = GetNameOfInObjectProperty(i);
  }

  // Deal with the other types of properties
  // ...
```

Sandbox CFI



Sandbox CFI

- Obvious: machine code cannot be inside the sandbox
    - => Move out of the sandbox

Sandbox CFI

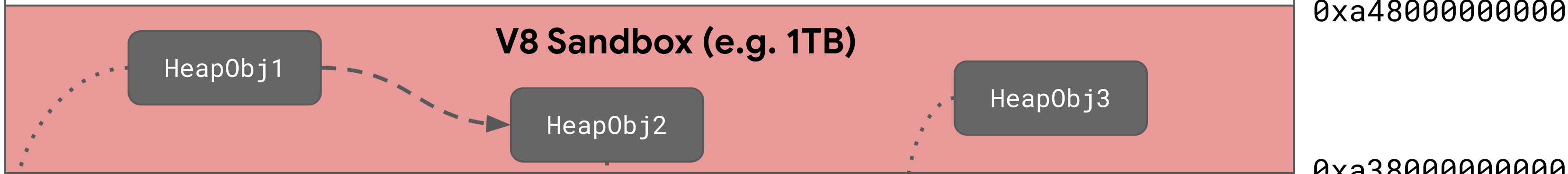
- Obvious: machine code cannot be inside the sandbox
  - => Move out of the sandbox
- Obvious: cannot have raw pointers to machine code inside the sandbox
  - => Use code pointer table indirection (essentially a form of CFI)

Sandbox CFI

- Obvious: machine code cannot be inside the sandbox
  - => Move out of the sandbox
- Obvious: cannot have raw pointers to machine code inside the sandbox
  - => Use code pointer table indirection (essentially a form of CFI)
- Less obvious: code metadata cannot be inside the sandbox
  - Can e.g. lead to code corruption when manipulated
  - => Move out of the sandbox

Sandbox CFI

- Obvious: machine code cannot be inside the sandbox
  - => Move out of the sandbox
- Obvious: cannot have raw pointers to machine code inside the sandbox
  - => Use code pointer table indirection (essentially a form of CFI)
- Less obvious: code metadata cannot be inside the sandbox
  - Can e.g. lead to code corruption when manipulated
  - => Move out of the sandbox
- Less obvious: interpreter bytecode cannot be in the sandbox
  - Causes stack corruption if manipulated
  - => Move out of sandbox and also reference via a pointer table



External Ptr Table	
0	Type + Pointer
1	Type + Pointer

Trusted Ptr Table	
0	Type + Pointer
1	Type + Pointer

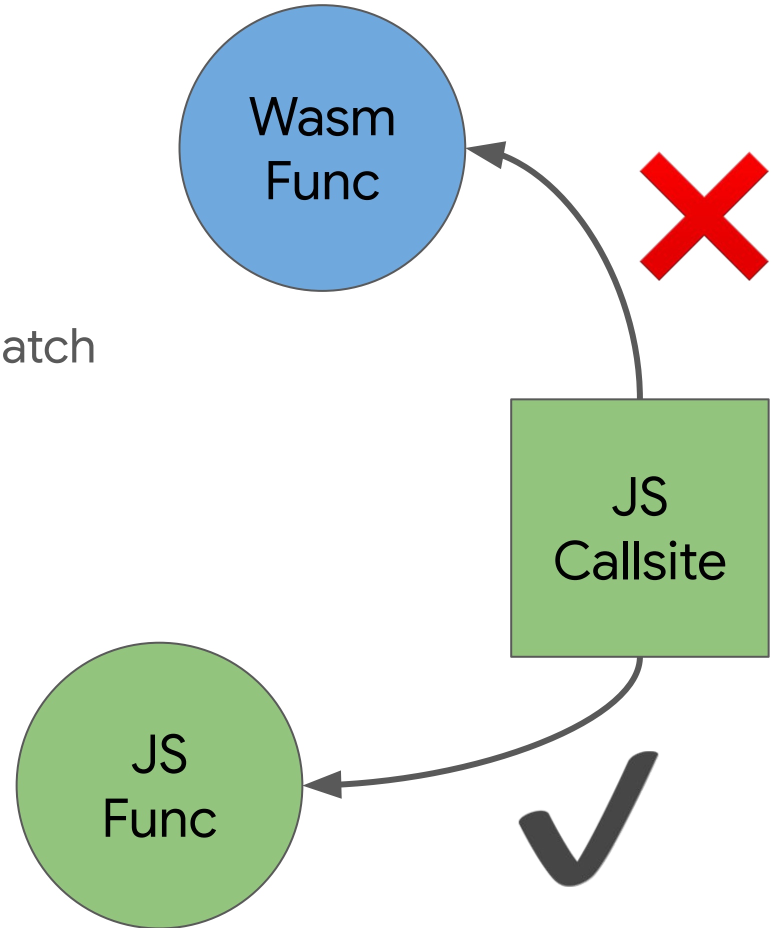
Code Ptr Table	
0	Pointer
1	Pointer

Sandbox CFI

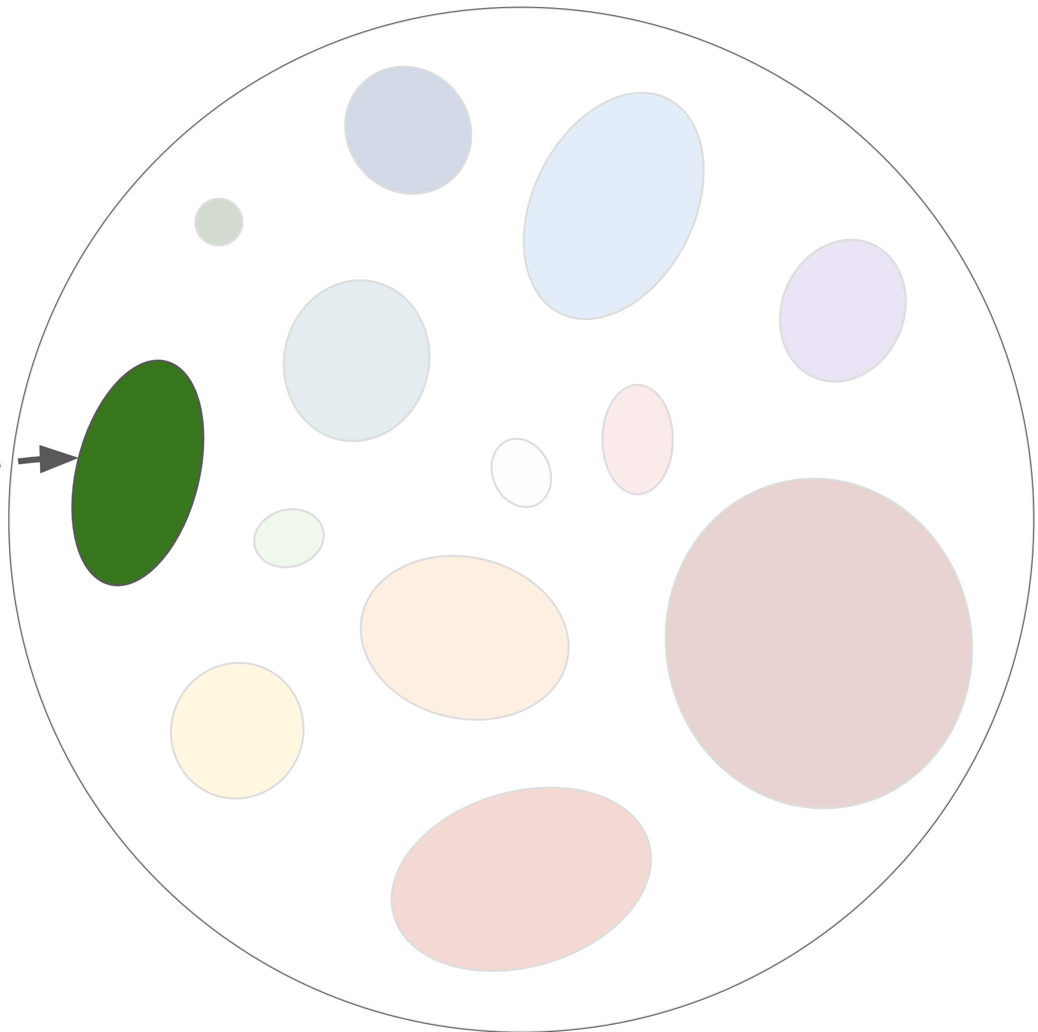
And more subtle issues in this area:

- Calling convention/signature mismatch
- Deoptimization and tier-up
- Desynchronized code references
- ...

=> Still work to do in this area



Testing



Testing

- Sandbox is *testable*
  - Clear attacker model + tools to develop and validate sandbox bypasses
- This enables:
  - automatic fuzzing
  - ability to write regression tests
  - inclusion in Chrome's bug bounty program (active since March 2024)

```
let memory = new Sandbox.MemoryView(0, kSize);
let dv = new DataView(memory);
// Full read+write to sandbox address space
dv.setUint8(0x41414141, 0x42);
```

Demo

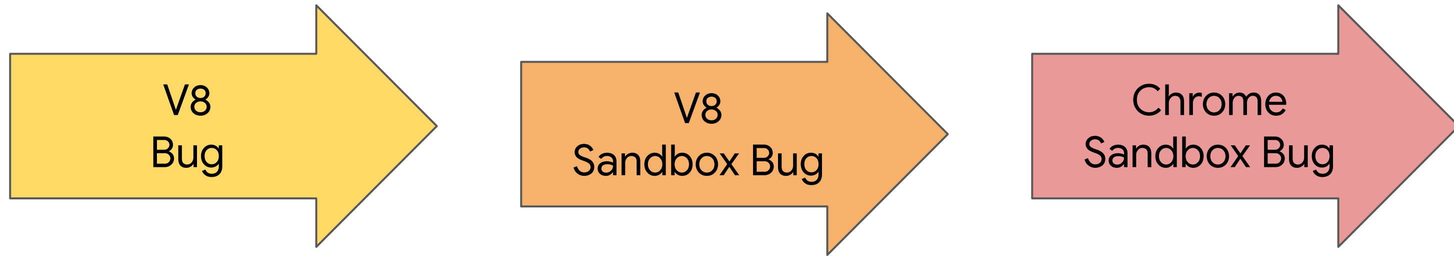
```
saelo v8/v8 [main] > cat demo.js
const kSize = 0x100000000;  // 4 GB
let memory = new DataView(new Sandbox.MemoryView(0, kSize));
memory.setUint32(0x41414141, 0x42424242);

saelo v8/v8 [main] > ./out/x64.sbxtst/d8 --s
```

```
saelo v8/v8 [main] > ./out/x64.sbxtst/d8 --sandbox-testing poc.js
```

Conclusion

Sandbox increases length of (typical) V8-based Chrome exploit chain



Key question: how hard is this new attack surface?

... Only one way to find out: build it, then see what happens :)

Resources

- Blog post: v8.dev/blog/sandbox
- README: src/sandbox/README.md
- Past sandbox bugs: v8-sandbox buganizer hotlist
- Sandbox VRP rules: g.co/chrome/vrp/#v8-sandbox-bypass-rewards

Questions?