Sandboxing V8

Samuel Groß - saelo@google.com

What?

Lightweight, in-process sandbox for V8

Attacker then needs:

V8 Vulnerability +

V8 Sandbox escape +

Chrome Sandbox escape

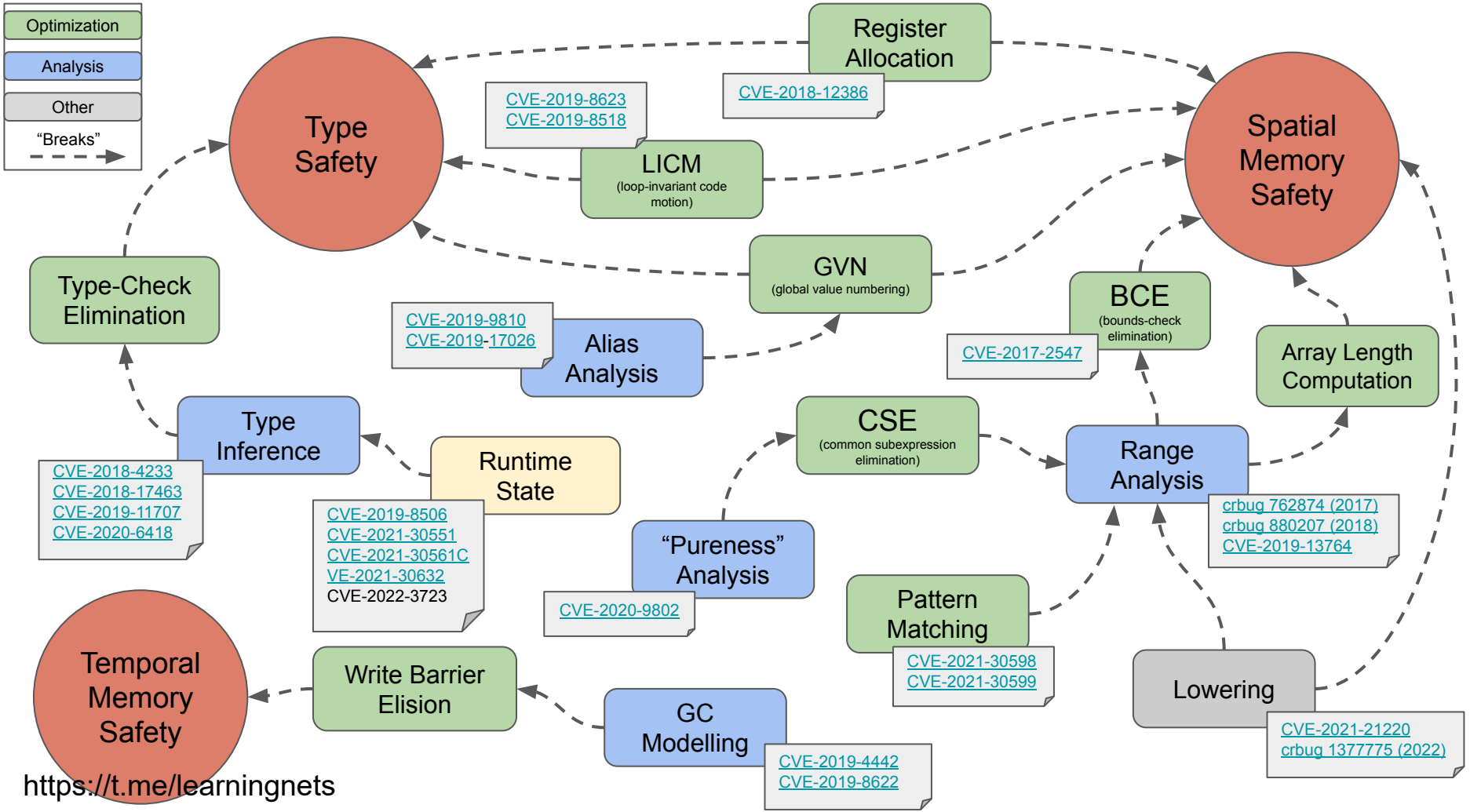
Why?

JavaScript engines are very attractive for attackers…

Fundamental problem: JS engine bugs are often *2nd order vulnerabilities*

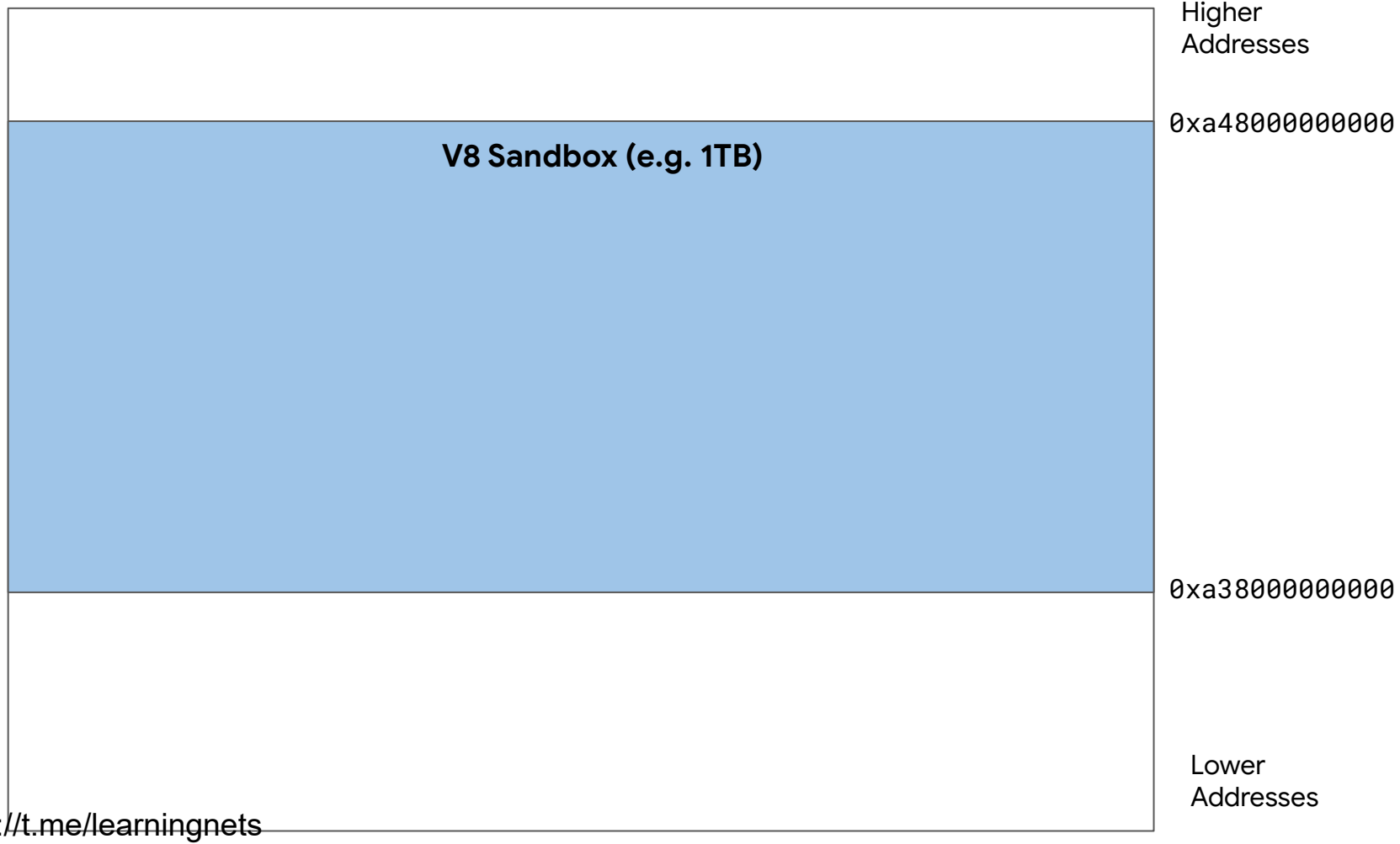
> Root cause is a *logic issue* in a compiler or the runtime environment...
>
> which can then be exploited for (more or less arbitrary) memory corruption at runtime

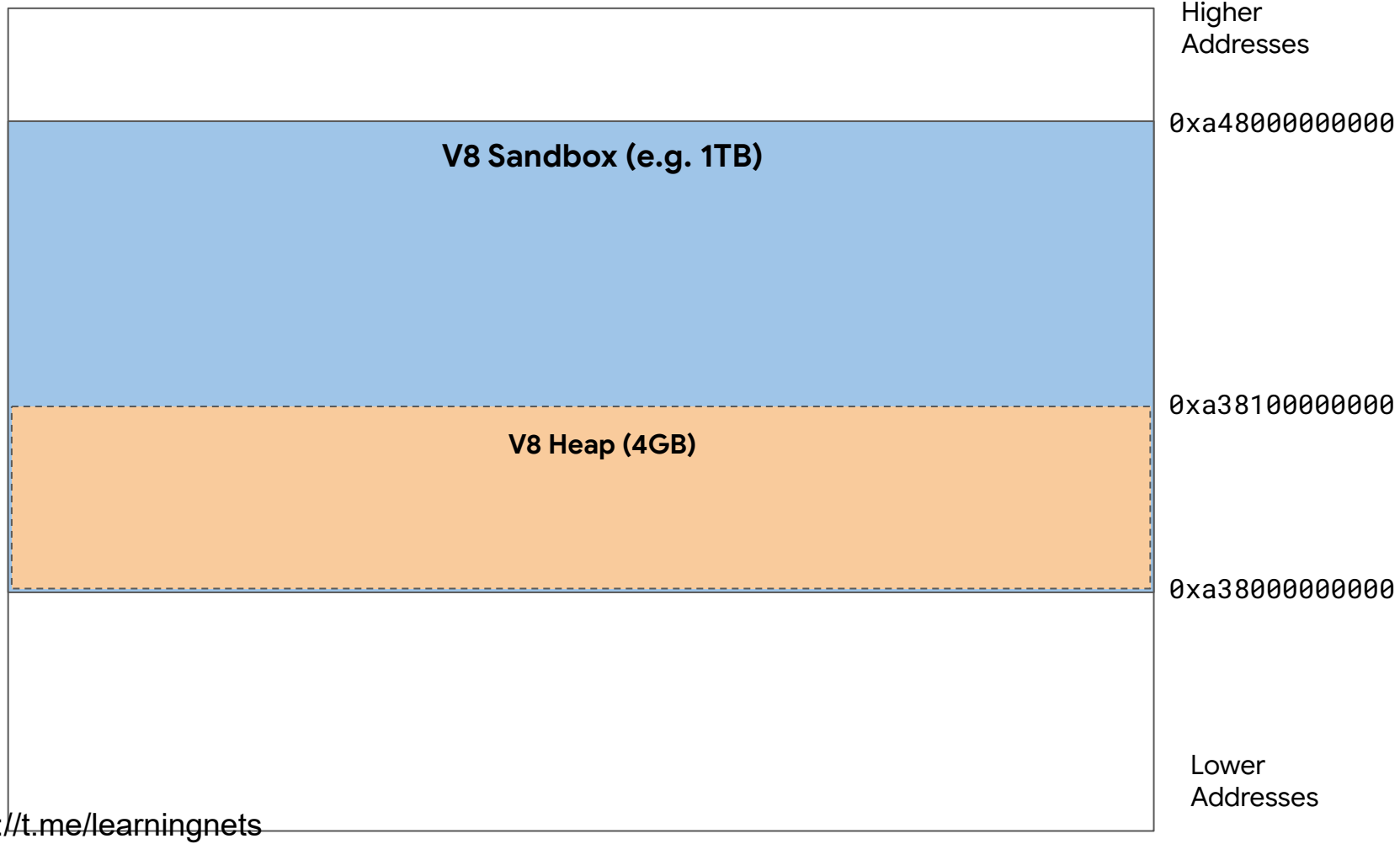
=> Powerful “superbugs”

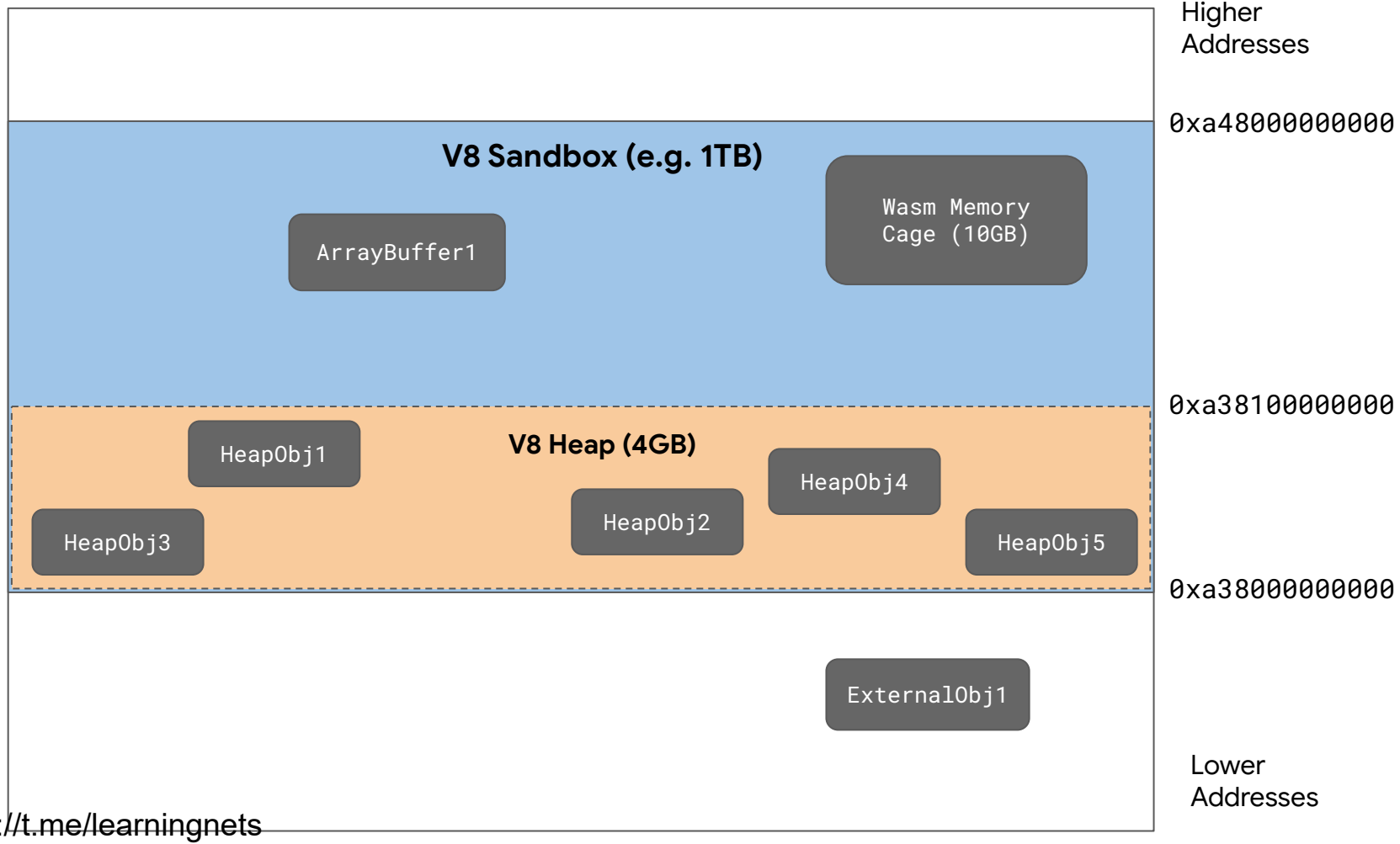


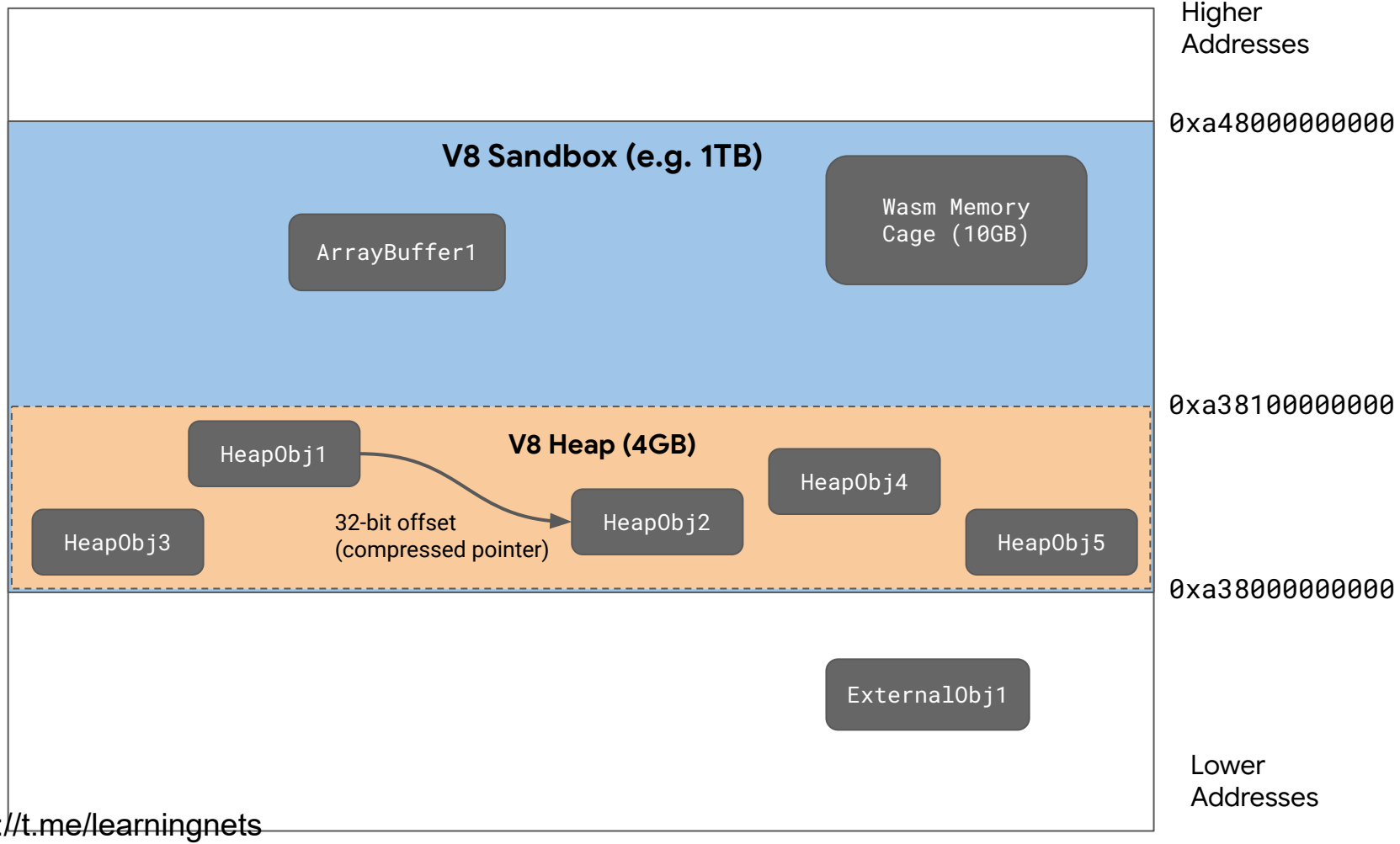
How?

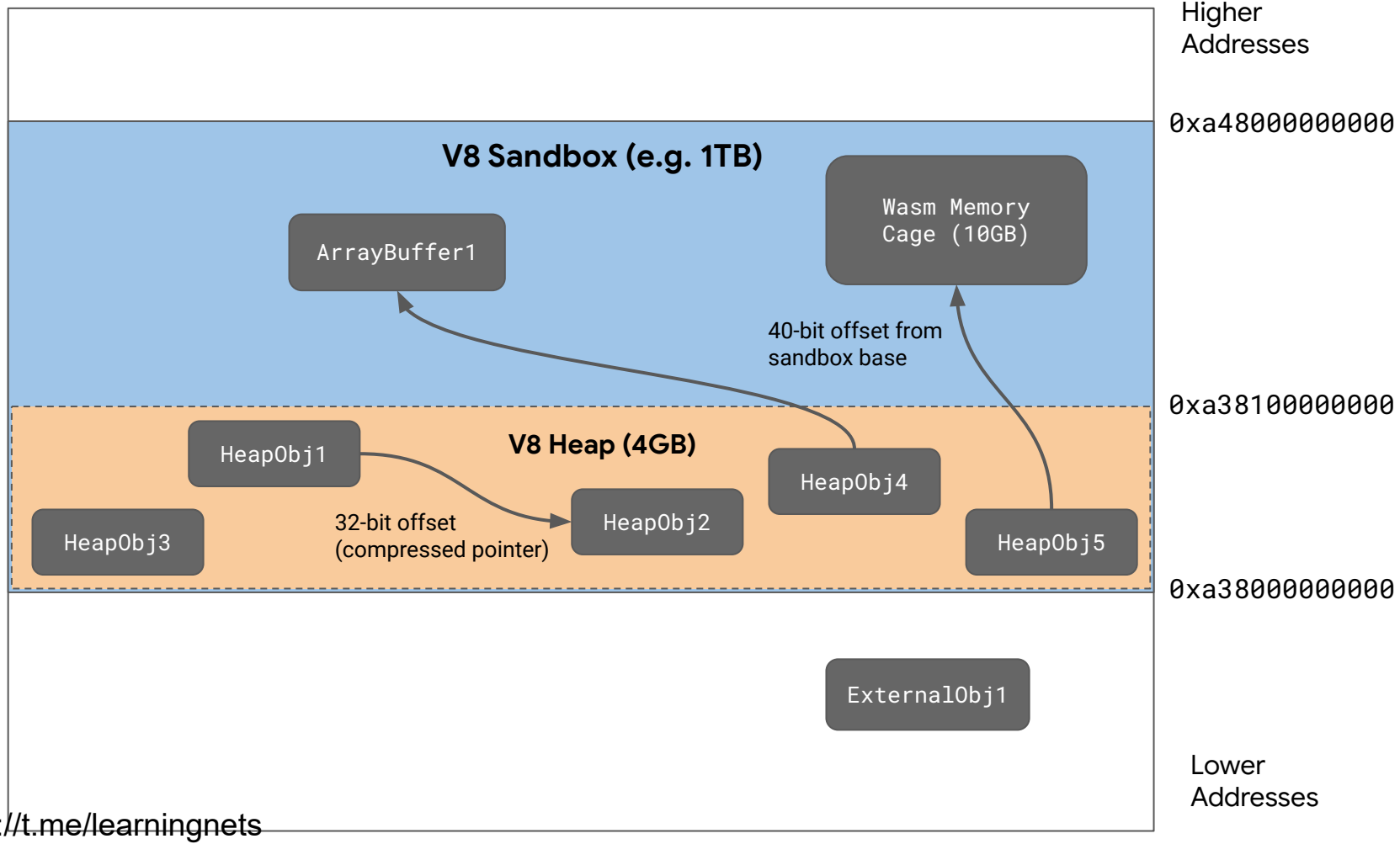
Basically: No more Pointers in V8! :)

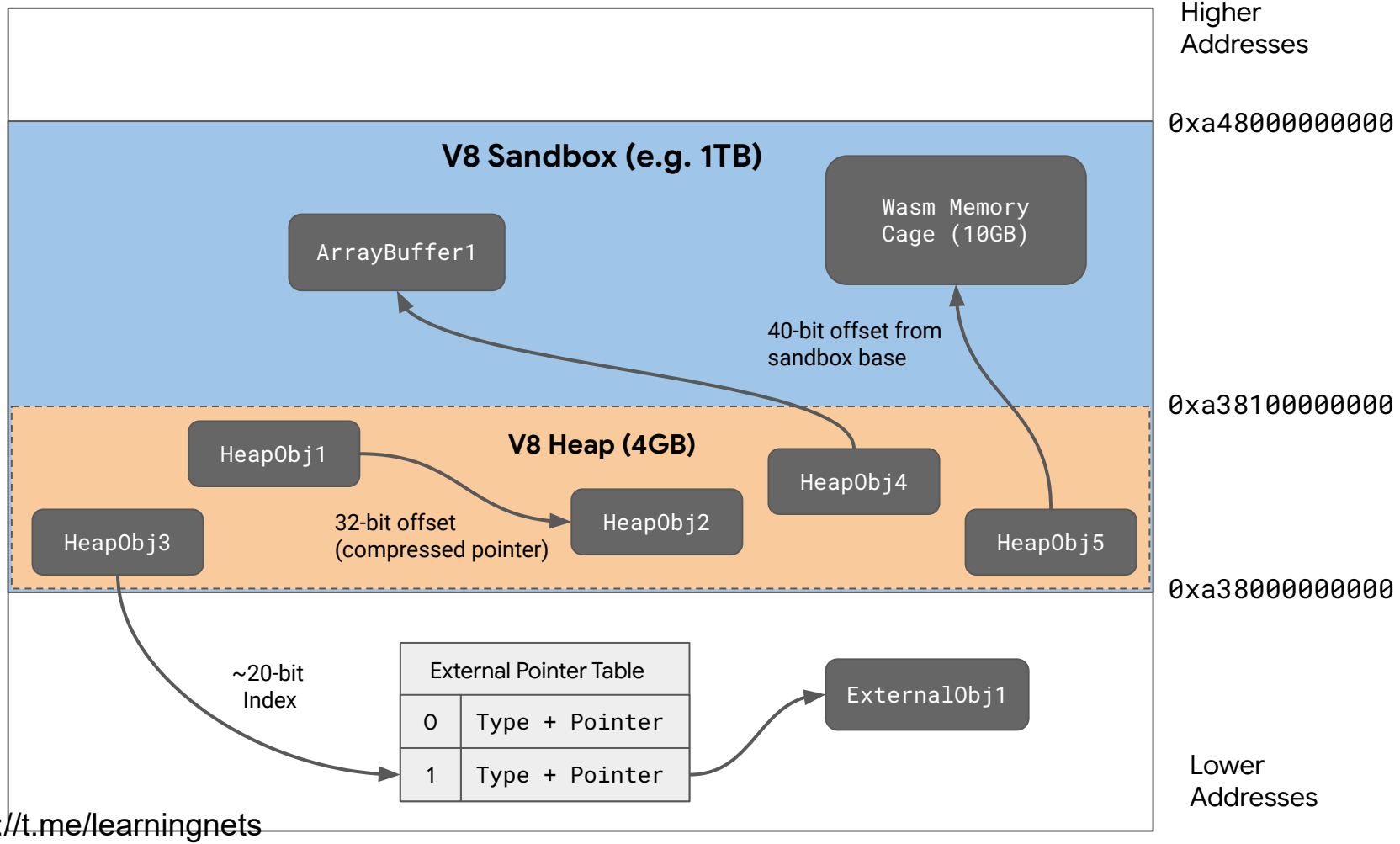










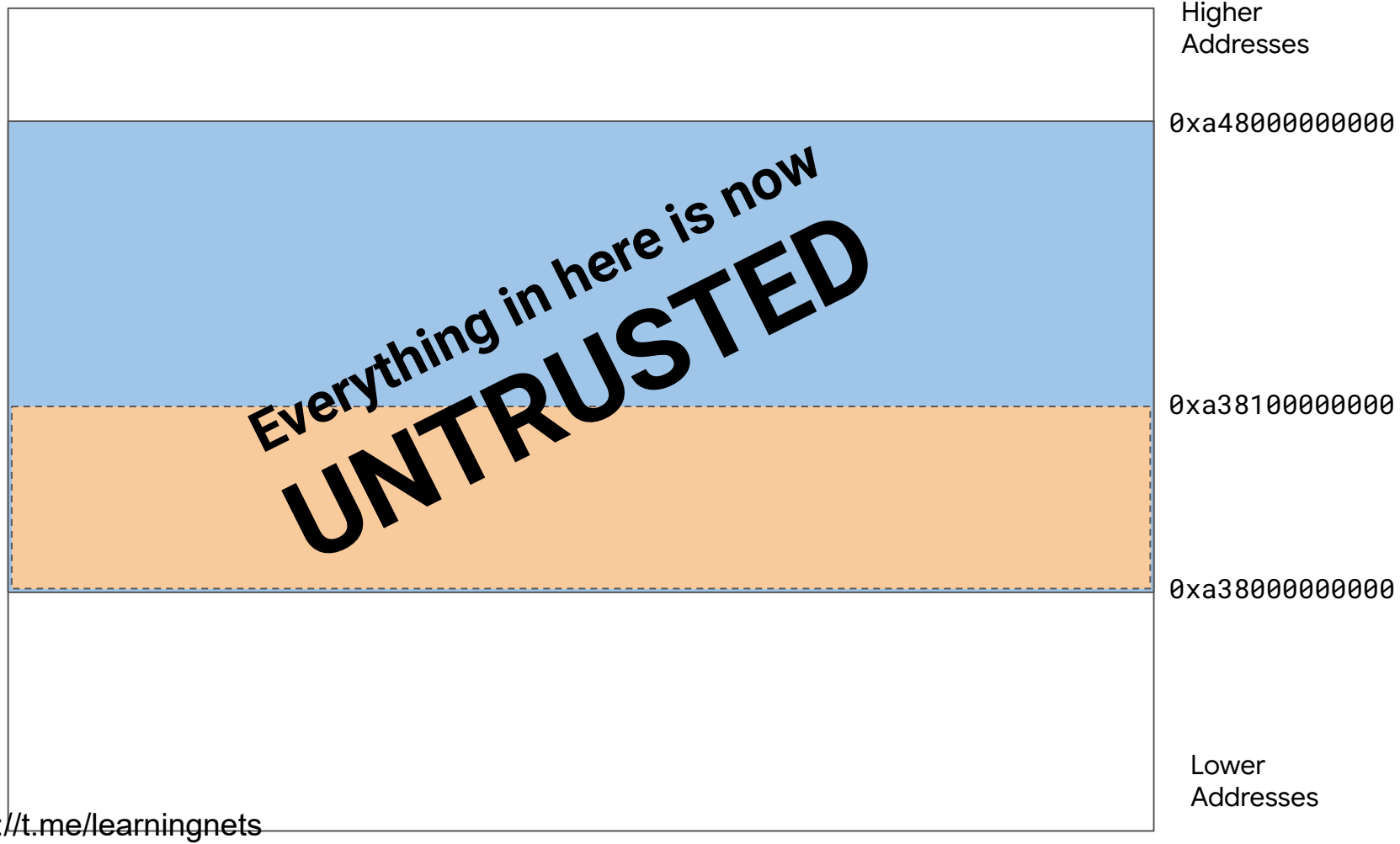


Performance?

- **Goal: < 2% performance overhead (currently ~1.5%)**
- Sandbox primitives are designed to be as performant as possible:
  - Offsets only require a shift+add (1 instruction on arm64, 2 on x64) to turn into a full pointer
  - External pointers require ~1 additional memory load + bitwise AND for the type check
  - External pointer table supports efficient GC and compaction
  - Everything else basically has no impact on performance
- Still, performance impact is noticeable…

```
void TurboAssembler::DecodeSandboxedPointer(const Register& value) {
  // Sandbox overhead: single shift+add instruction
  Add(value, kPtrComprCageBaseRegister,
      Operand(value, LSR, kSandboxedPointerShift));
}
```

Implications (Beyond V8)



Implications Beyond V8

```
int idx = ...;

v8_obj->SetInternalField(0, v8_num(idx));

...

...

uint idx = v8_obj->GetInternalField(0).to_uint();

return array[idx];
```

Implications Beyond V8

```
int idx = ...;

v8_obj->SetInternalField(0, v8_num(idx));

...

...

uint idx = v8_obj->GetInternalField(0).to_uint();

return array[idx];
```

Implications Beyond V8

```
int idx = ...;

v8_obj->SetInternalField(0, v8_num(idx));

...

...

uint idx = v8_obj->GetInternalField(0).to_uint();

CHECK_LT(idx, size_of_array);     // Simple fix ...

// ... and a fuzzer will find this easily, at least ...

return array[idx];
```

Goals of the V8 Sandbox

Goals of the V8 Sandbox

1. Introduce a new security boundary

    V8-based Chrome exploit chains now requires 3 instead of 2 vulnerabilities

Goals of the V8 Sandbox

1. Introduce a new security boundary

   V8-based Chrome exploit chains now requires 3 instead of 2 vulnerabilities

2. Reduce “2nd order” vulnerabilities to “1st order” ones

   V8 sandbox escape bug is likely a “standard” memory corruption bug

State of the V8 Sandbox (Dec. 2022)

State of the V8 Sandbox (Dec. 2022)

- Sandbox “foundation” launched this year
  - Sandbox Address Space and Sandboxed Pointers launched in M103 (~June)
  - External Pointer Table launched in M107 (~October)
  - Misc. smaller features and fixes (e.g Bounded Size) shipped throughout the year
- At this point, basically still a proof-of-concept

State of the V8 Sandbox (Dec. 2022)

- Sandbox “foundation” launched this year
  - Sandbox Address Space and Sandboxed Pointers launched in M103 (~June)
  - External Pointer Table launched in M107 (~October)
  - Misc. smaller features and fixes (e.g Bounded Size) shipped throughout the year
- At this point, basically still a proof-of-concept
- Up next:
  - Code pointer sandboxing (~= V8 sandbox CFI)
  - Fine-granular type tags for pointers to Blink objects (e.g. DOM nodes)
  - Building custom fuzzers for the sandbox attack surface
  - A ton of further fixes, code refactoring, etc.

Summary

V8 Sandbox architecture breaks down a hard problem (secure *and* fast JS engine) into lots of smaller problems with "relatively" easy solutions (OOB accesses, TOCTOU issues, UAFs, …).