

Shadow bytes around the buggy address:

0x00016ddaab00:	00	00	00	00	00	00	00	00	00
0x00016ddaab80:	00	00	00	00	00	00	00	00	00
0x00016ddaac00:	00	00	00	00	00	00	00	00	00
0x00016ddaac80:	00	00	00	00	00	00	00	00	00
0x00016ddaad00:	00	00	f1	f1	f1	f1	00	00	00
=>0x00016ddaad80:	00	[f3]	f3	f3	f3	f3	f3	00	00
0x00016ddaae00:	00	00	00	00	00	00	00	00	00

Address Sanitizer (ASan) Internals

Adapted from Serebryany et al. (2012)

Motivation

**The problem: C lets us do too
much with memory.**

C lets us do too much with memory.

```
[samuel-skean@armDebianVM:~$ valgrind ./a.out
==2031== Memcheck, a memory error detector
==2031== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2031== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==2031== Command: ./a.out
==2031==
Hi this is your number: 8
==2031== Invalid read of size 4
==2031==    at 0x108814: main (silly.c:14)
==2031==   Address 0x1fff0007bc is on thread 1's stack
==2031==  20 bytes below stack pointer
==2031==
Your number, sir: 8
```

C lets us do too much with memory.

- Buffer overflow (/underflow)
 - Heap
 - Stack
 - Global
- Use-after-free

```
[samuel-skean@armDebianVM:~$ valgrind ./a.out
==2031== Memcheck, a memory error detector
==2031== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2031== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==2031== Command: ./a.out
==2031==
Hi this is your number: 8
==2031== Invalid read of size 4
==2031==    at 0x108814: main (silly.c:14)
==2031==   Address 0x1ffff007bc is on thread 1's stack
==2031==  20 bytes below stack pointer
==2031==
Your number, sir: 8
```


C lets us do too much with memory.

- Buffer overflow (/underflow)
 - Heap
 - Stack
 - Global
- Use-after-free
- Use after function return?

```
[samuel-skean@armDebianVM:~$ valgrind ./a.out
==2031== Memcheck, a memory error detector
==2031== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2031== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==2031== Command: ./a.out
==2031==
Hi this is your number: 8
==2031== Invalid read of size 4
==2031==    at 0x108814: main (silly.c:14)
==2031==   Address 0x1fff0007bc is on thread 1's stack
==2031==  20 bytes below stack pointer
==2031==
Your number, sir: 8
```


C lets us do too much with memory.

- Buffer overflow (/underflow)
 - Heap
 - Stack
 - Global
- Use-after-free
- Use after function return?

```
[samuel-skean@armDebianVM:~$ valgrind ./a.out
==2031== Memcheck, a memory error detector
==2031== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2031== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==2031== Command: ./a.out
==2031==
Hi this is your number: 8
==2031== Invalid read of size 4
==2031==    at 0x108814: main (silly.c:14)
==2031== Address 0x1fff0007bc is on thread 1's stack
==2031== 20 bytes below stack pointer
==2031==
Your number, sir: 8
==2031==
==2031== HEAP SUMMARY:
==2031==    in use at exit: 4 bytes in 1 blocks
==2031== total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==2031==
==2031== LEAK SUMMARY:
==2031==    definitely lost: 4 bytes in 1 blocks
```

We can do better than Valgrind.

We can do better than Valgrind.

- Faster

We can do better than Valgrind.

- Faster
 - Less Intrusive - fewer dependencies on language

We can do better than Valgrind.

- Faster
 - Less Intrusive - fewer dependencies on language
- Accurate?

We can do better than Valgrind.

- Faster
 - Less Intrusive - fewer dependencies on language
- Accurate?
 - Well, still no false positives.

The Approach

redzone1

mem1

redzone2

mem2

redzone3

How ASan lays out memory:



**What if we could have memory
about our memory?**



**YO DAWG, I HEARD YOU LIKE
CRASHING**

**What if we could have memory
about our memory?**

**SO I PUT WINDOWS IN YOUR CAR SO
YOU CAN CRASH WHILE YOU CRASH**



What if we could have memory
about our memory?

The Shadow Mapping

Address of shadow byte =
(Address of normal byte $\gg 3$) + Offset

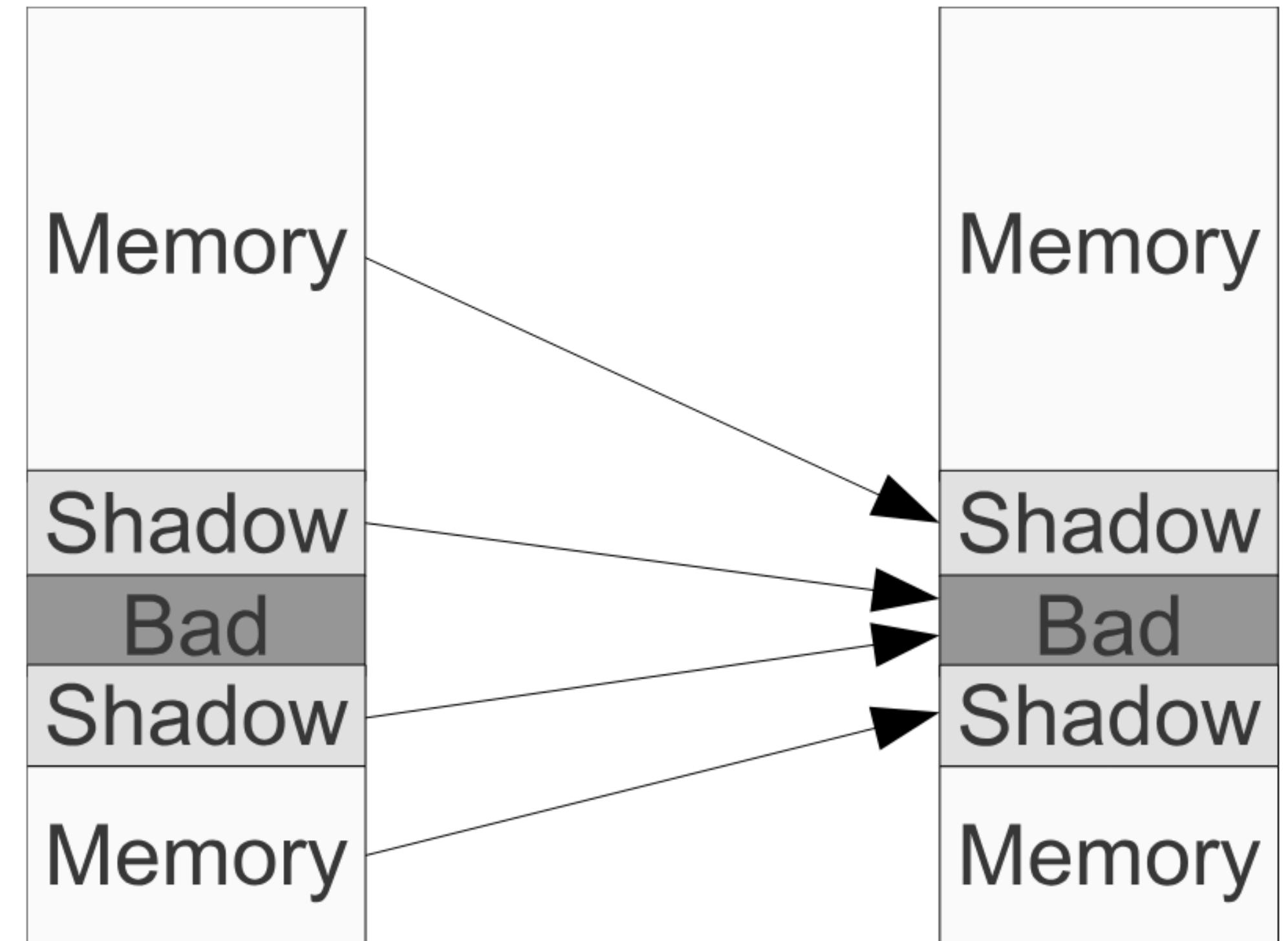
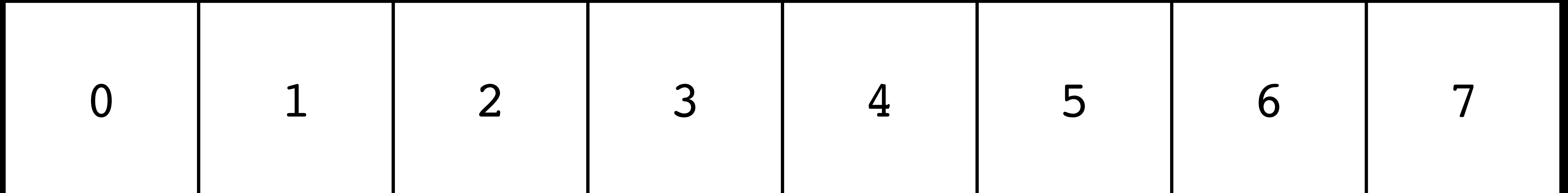


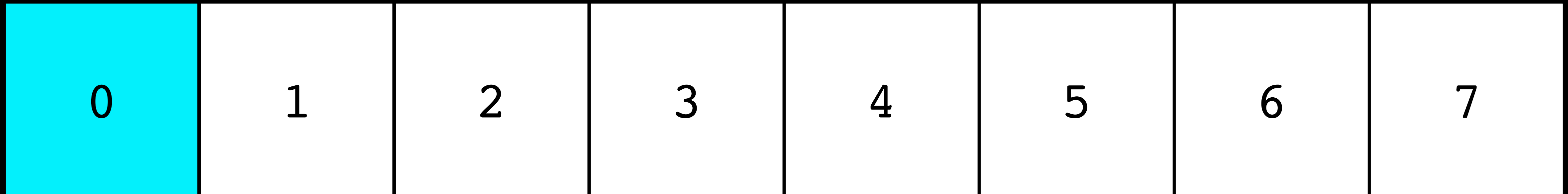
Figure 1: AddressSanitizer memory mapping.

A chunk of 8 bytes, 8-byte aligned



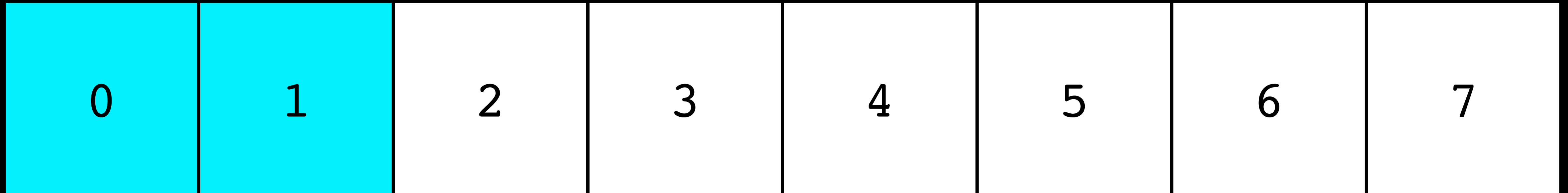
0 allocated bytes

A chunk of 8 bytes, 8-byte aligned



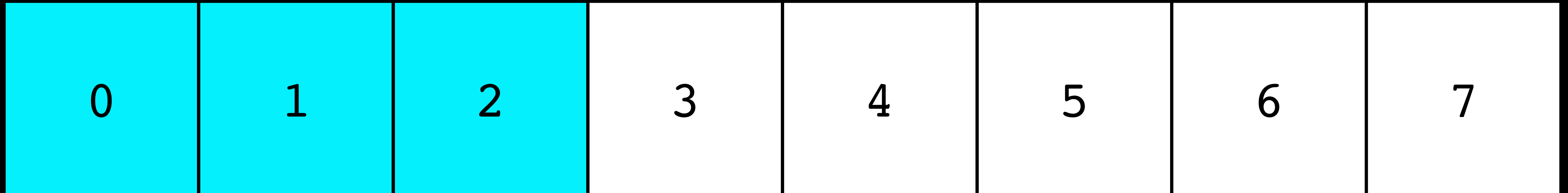
1 allocated byte

A chunk of 8 bytes, 8-byte aligned



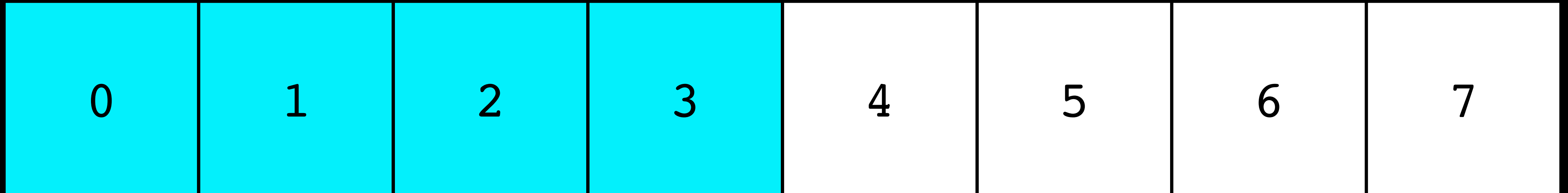
2 allocated bytes

A chunk of 8 bytes, 8-byte aligned



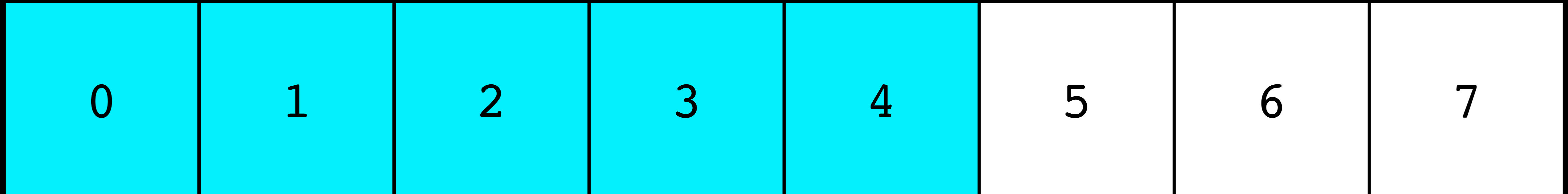
3 allocated bytes

A chunk of 8 bytes, 8-byte aligned



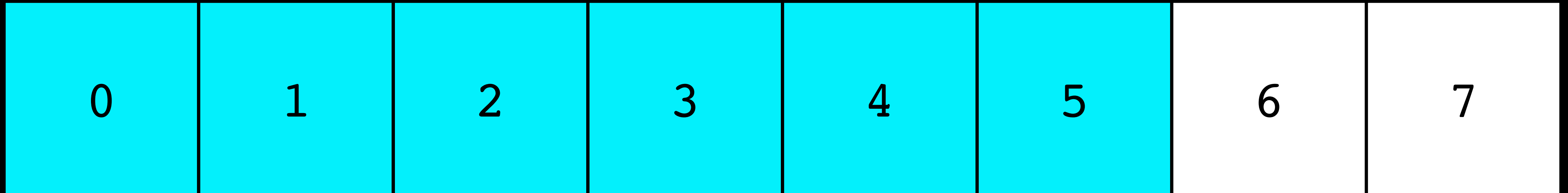
4 allocated bytes

A chunk of 8 bytes, 8-byte aligned



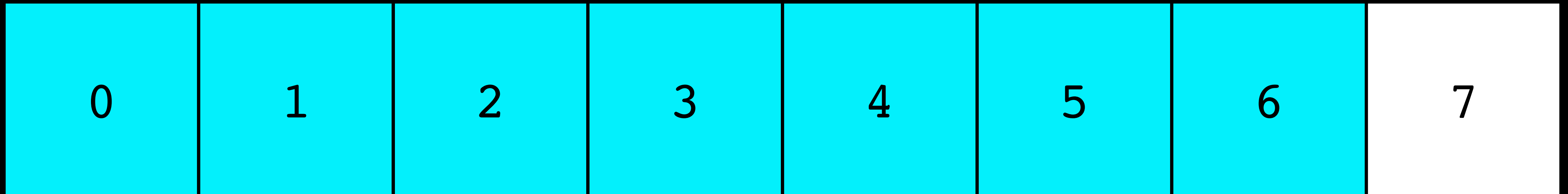
5 allocated bytes

A chunk of 8 bytes, 8-byte aligned



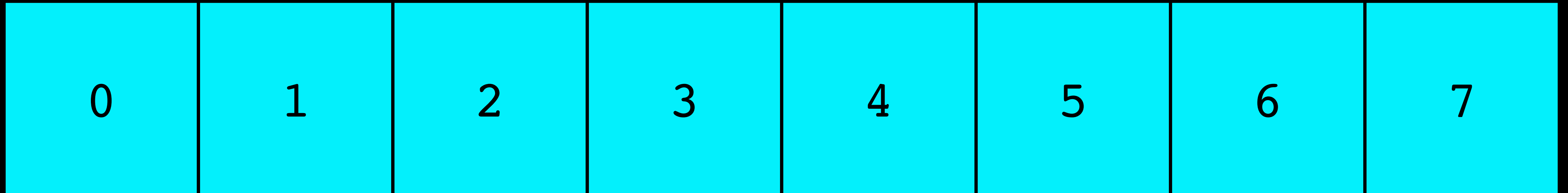
6 allocated bytes

A chunk of 8 bytes, 8-byte aligned



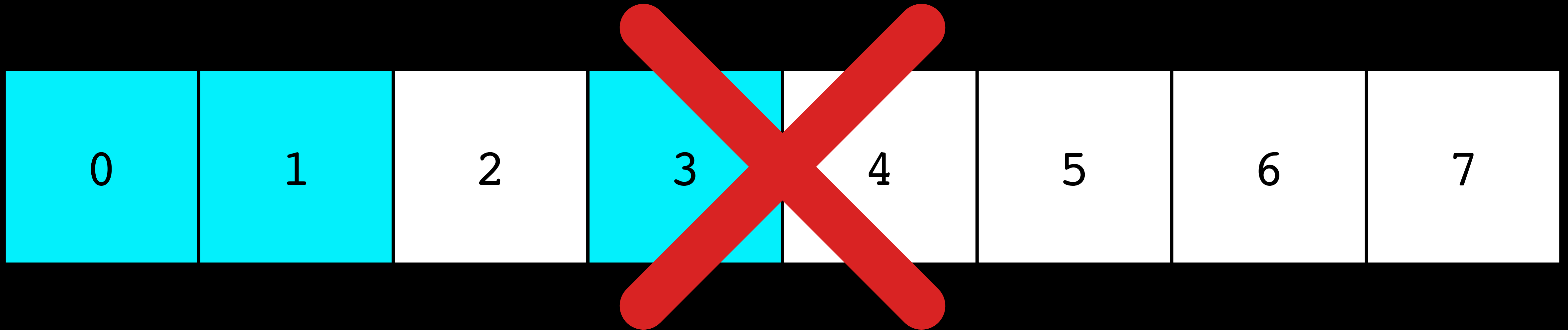
7 allocated bytes

A chunk of 8 bytes, 8-byte aligned



8 allocated bytes

A chunk of 8 bytes, *Not* 8-byte aligned



Every Memory Load/Store Gets This Before it?

```
ShadowAddr = (Addr >> 3) + Offset;  
if (*ShadowAddr != 0)  
    ReportAndCrash(Addr);
```

Every Memory Load/Store Gets This Before it

```
ShadowAddr = (Addr >> 3) + Offset;  
k = *ShadowAddr;  
if (k != 0 && ((Addr & 7) + AccessSize > k))  
    ReportAndCrash(Addr);
```

What We Looked At + Feedback



<https://forms.gle/cj5FfdcnacixxAb37>