(Offload) ASAN via Software Managed Virtual Memory

Johannes Doerfert <jdoerfert@llnl.gov>
Ethan McDonough <ethanluismcdonough@gmail.com>
Vidush Singhal <singhav@purdue.edu>

Setting the Stage

Work in Progress ⟶ Room for Improvement ⟶ Approximate & Limited Results

Thanks

Aaron Jarmusch <jarmusch@udel.edu> — NVIDIA measurements

Rahulkumar Gayatri <rgayatri@lbl.gov> — KOKKOS testing & measurements

Prologue

```
>>> nvim src/simulation.cpp
```

```
>>> nvim src/simulation.cpp
>>> ninja -C build
```

```
>>> nvim src/simulation.cpp
>>> ninja -C build
>>> ./build/bin/science -max
```

```
>>> nvim src/simulation.cpp
>>> ninja -C build
>>> ./build/bin/science -max

Starting to produce maximal science…
```

```
>>> nvim src/simulation.cpp
>>> ninja -C build
>>> ./build/bin/science -max

Starting to produce maximal science…
AMDGPU fatal error 1: Memory access fault by GPU 2 (agent 0x557136cb9fd0)
at virtual address 0x7f3780422000. Reasons: Unknown (0)
```

```
>>> nvim src/simulation.cpp
>>> ninja -C build
>>> ./build/bin/science -max

Starting to produce maximal science…
Display last 2 kernels launched:
Kernel 0: 'omp target in openmc::process_collision_events() @ 352
Kernel 1: 'omp target in openmc::process_surface_crossing_events() @ 323
AMDGPU fatal error 1: Memory access fault by GPU 2 (agent 0x557136cb9fd0)
at virtual address 0x7f3780422000. Reasons: Unknown (0)
```

Main Cast

ASAN

the code snitch reporting stale pointers and buffer overflows

- instrumentation + runtime
- shadow memory & redzones
- staple since LLVM 3.1 (~2012)

GPUs

the AI accelerators that can do graphics and general compute

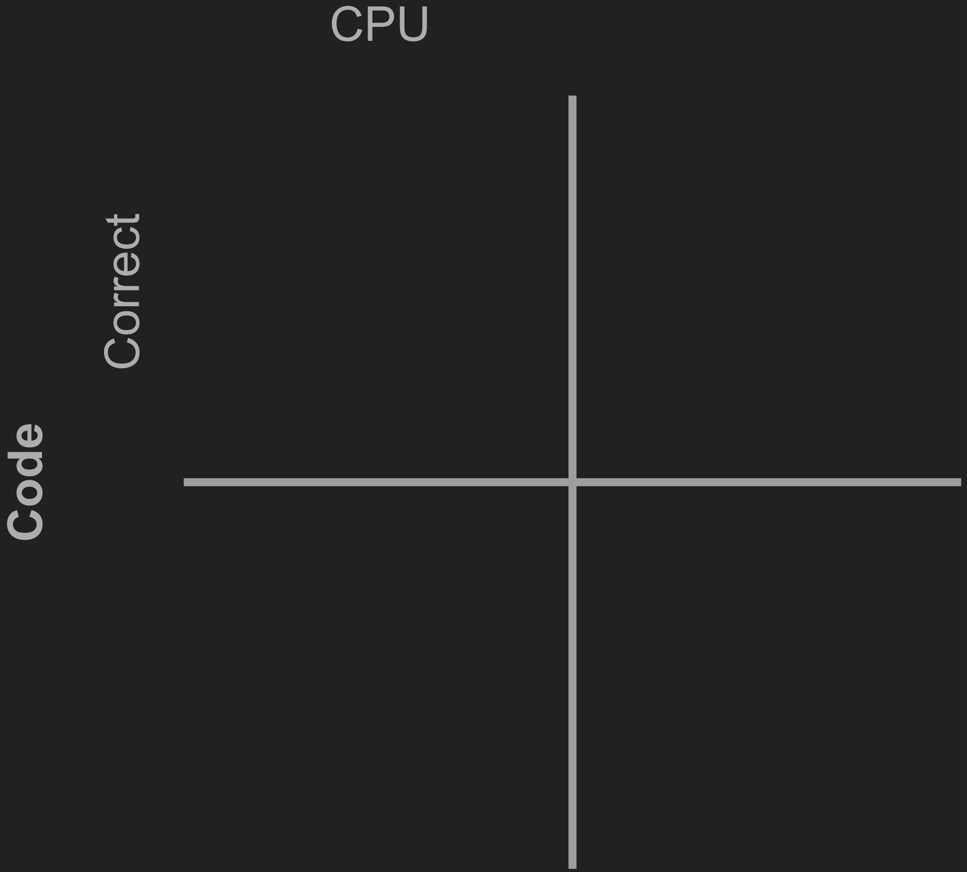
- powerhouse of modern HPC
- we initially focus on AMD GPUs
- dedicated memory is assumed

Plot

Architecture

Code

Architecture



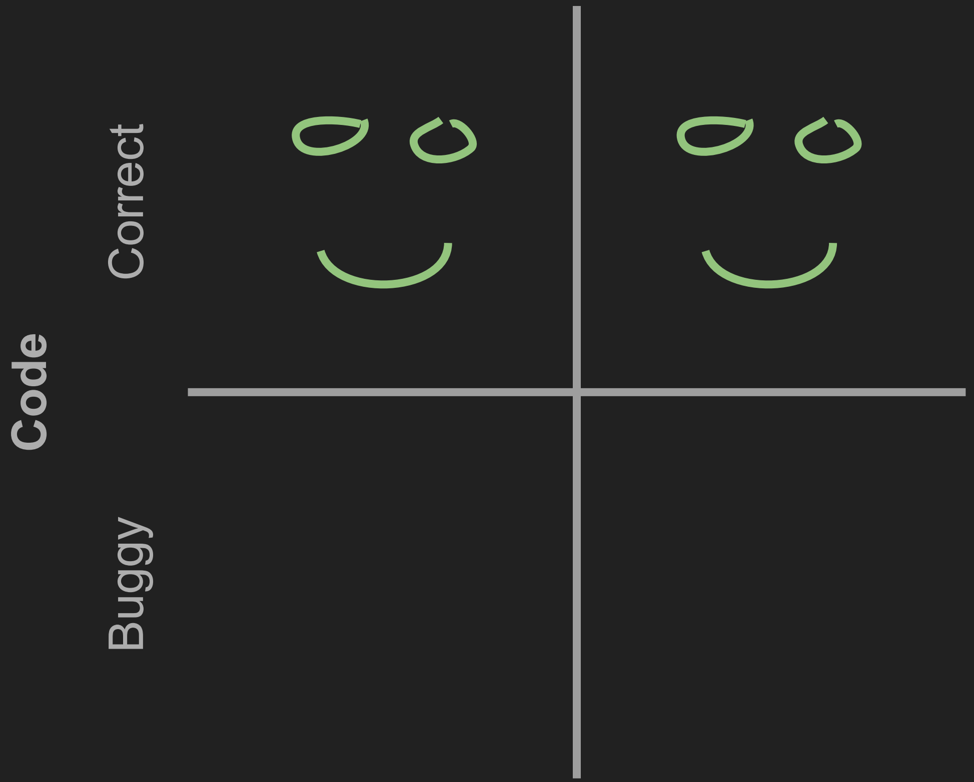
CPU

Correct

Code

Architecture

Code	CPU	GPU
Correct		
Buggy		



Architecture

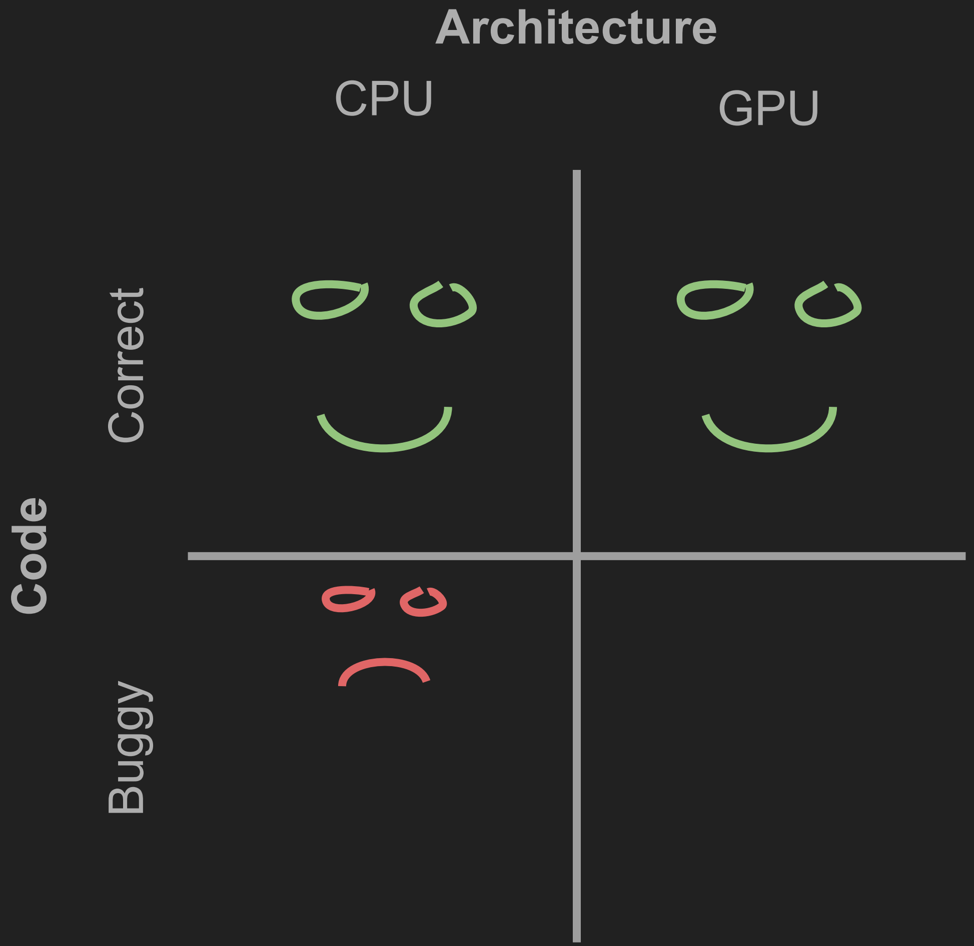
CPU

GPU

Code

Correct

Buggy



Architecture

CPU

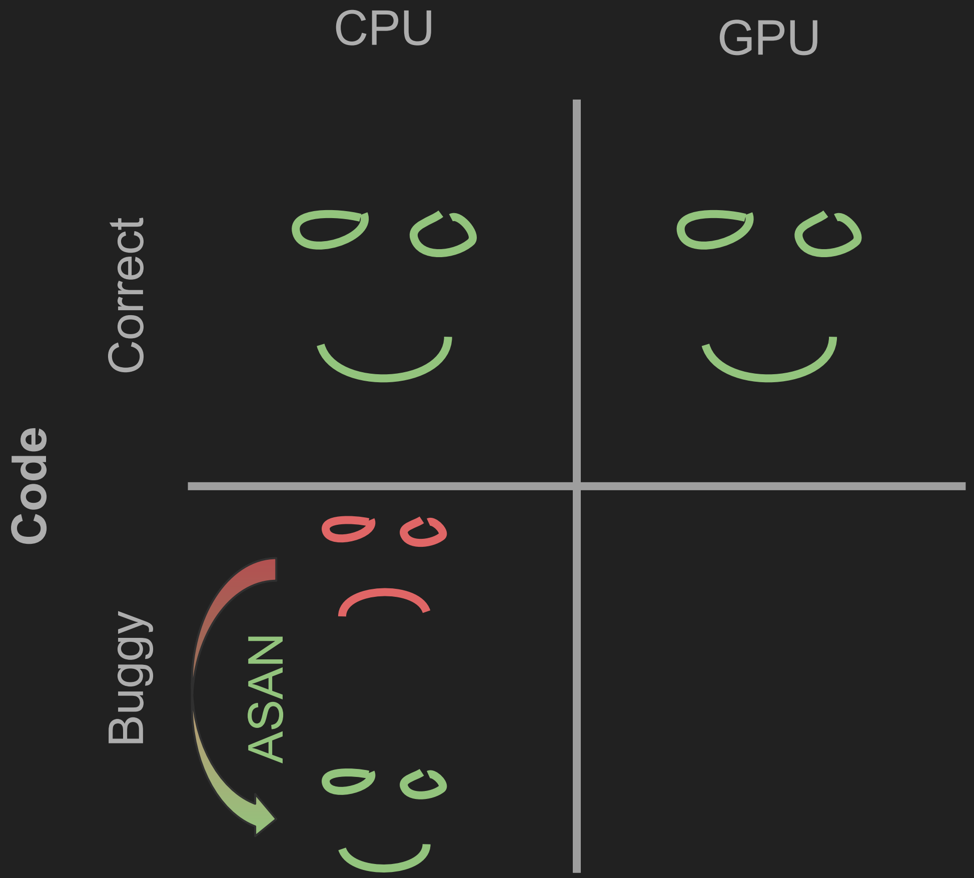
GPU

Code

Correct

Buggy

Architecture



CPU

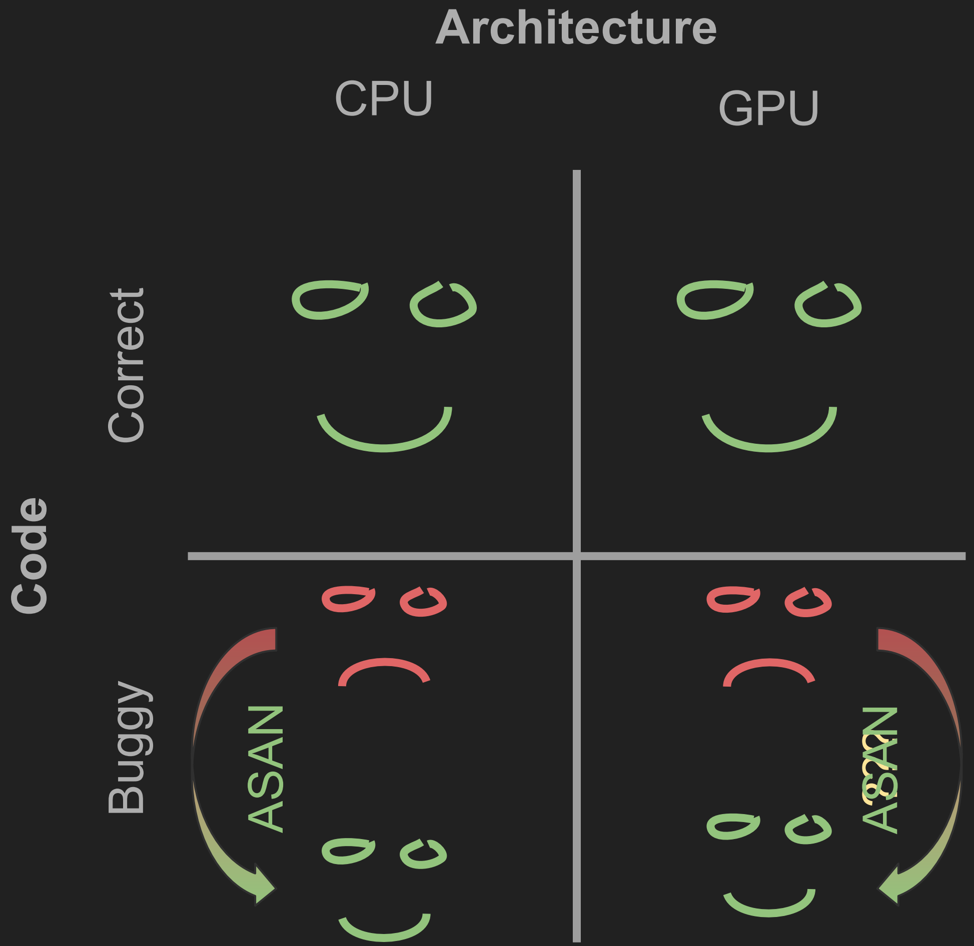
GPU

Code

Correct

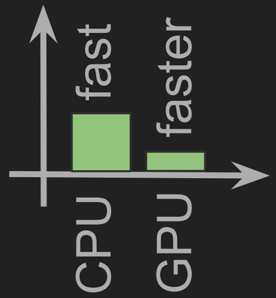
Buggy

ASAN

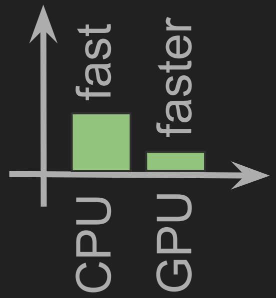

Architecture
CPU
GPU
Code
Correct
Buggy
ASAN
ASAN
???

Complication

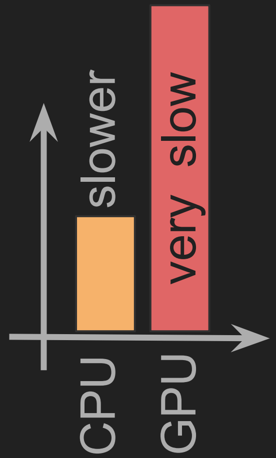
code **compile** executable **run**



code → **compile** → executable → **run**



code → **compile and sanitize** → executable → **run**

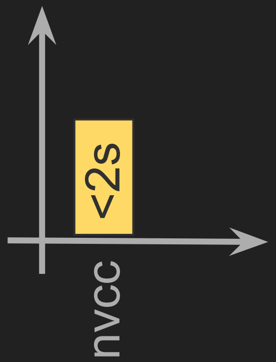


Situation

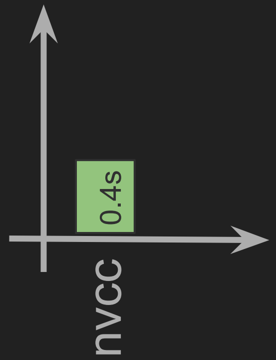
XSBench

- continuous energy macroscopic neutron cross section lookup
- event-based lookup, unionized grid, large H-M size (~5.5GB)
- proxy application for OpenMC
- CUDA/HIP/OpenMP offload versions

V100 - OpenMP



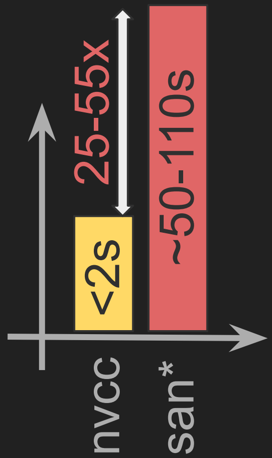
V100 - CUDA



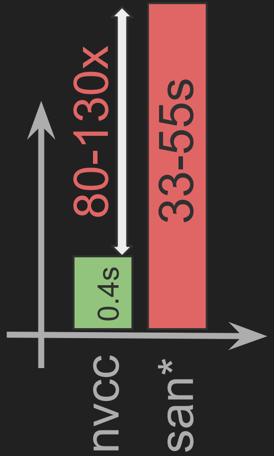
XSBench

- continuous energy macroscopic neutron cross section lookup
- event-based lookup, unionized grid, large H-M size (~5.5GB)
- proxy application for OpenMC
- CUDA/HIP/OpenMP offload versions

V100 - OpenMP



V100 - CUDA

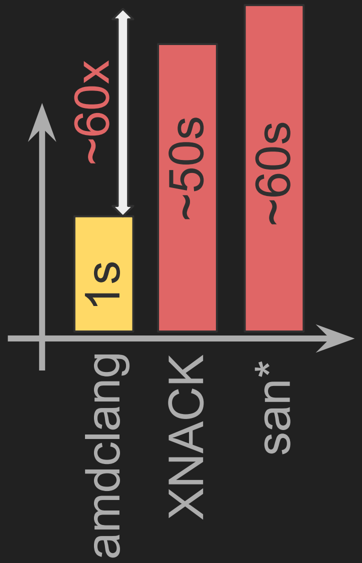


* NVIDIA Compute Sanitizer

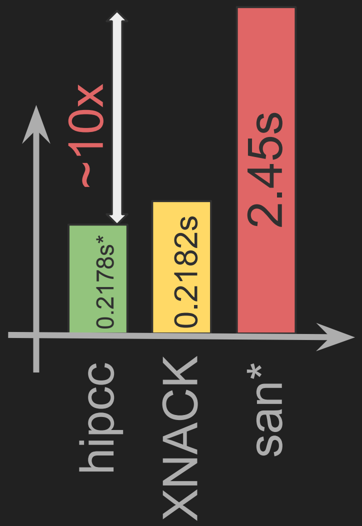
XSBench

- continuous energy macroscopic neutron cross section lookup
- event-based lookup, unionized grid, large H-M size (~5.5GB)
- proxy application for OpenMC
- CUDA/HIP/OpenMP offload versions

MI-250X - OpenMP



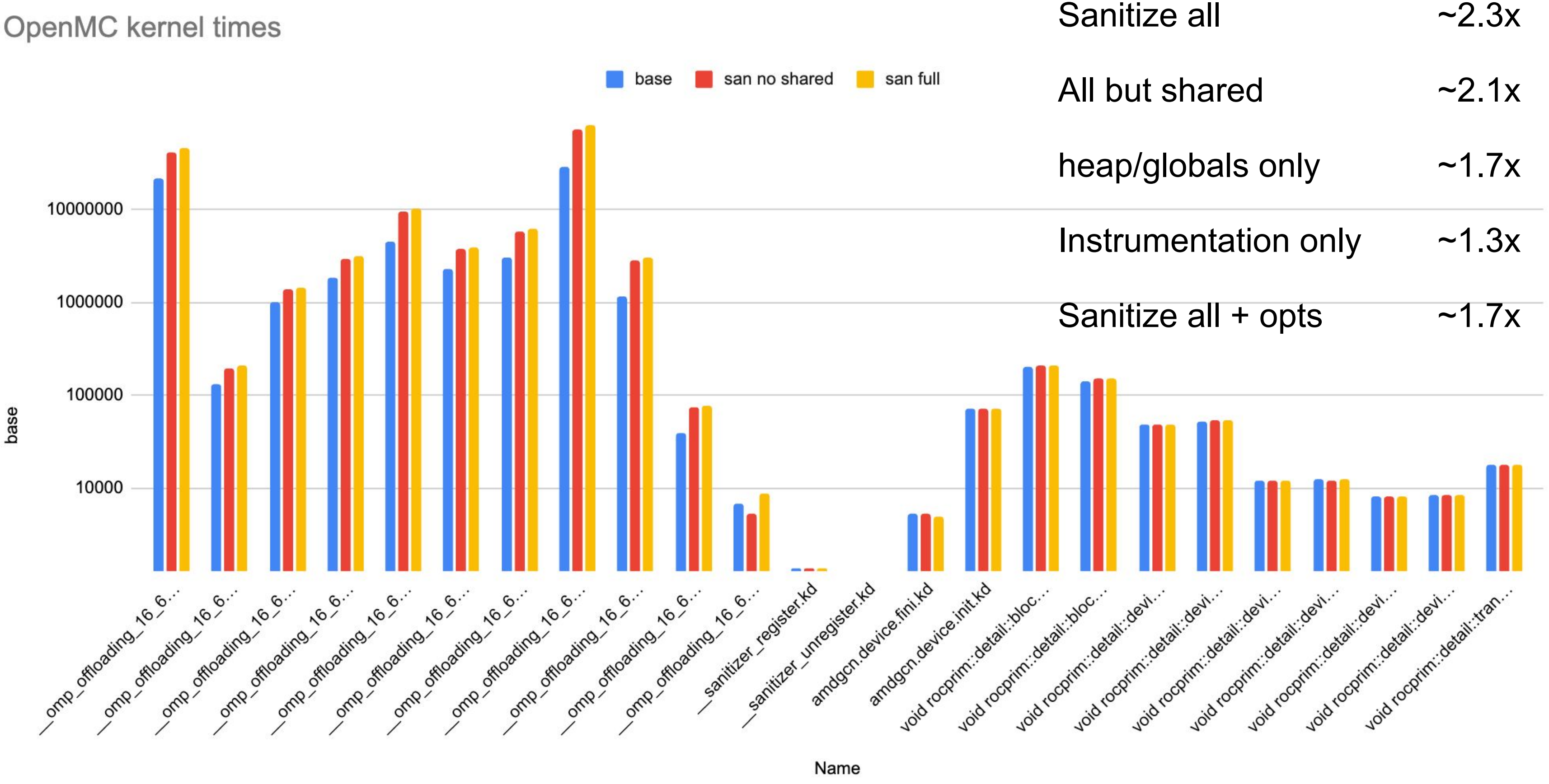
MI-250X - HIP



* AMD ASAN for GPU incl. XNACK

XSBench

- continuous energy macroscopic neutron cross section lookup
- event-based lookup, unionized grid, large H-M size (~5.5GB)
- proxy application for OpenMC
- CUDA/HIP/OpenMP offload versions



Sanitize all	~2.3x
All but shared	~2.1x
heap/globals only	~1.7x
Instrumentation only	~1.3x
Sanitize all + opts	~1.7x

Backstory

fixed-sized
red zones

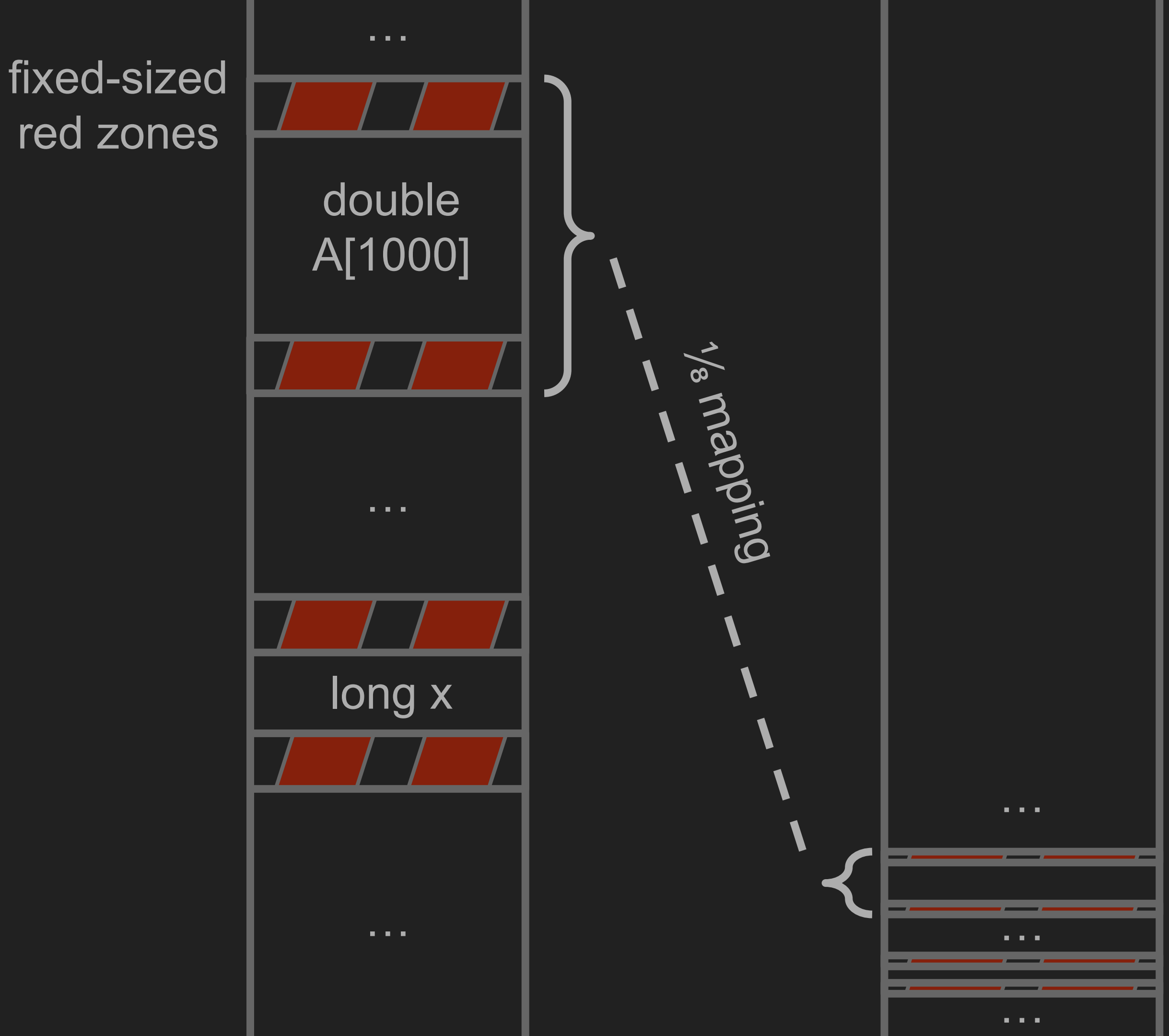
…

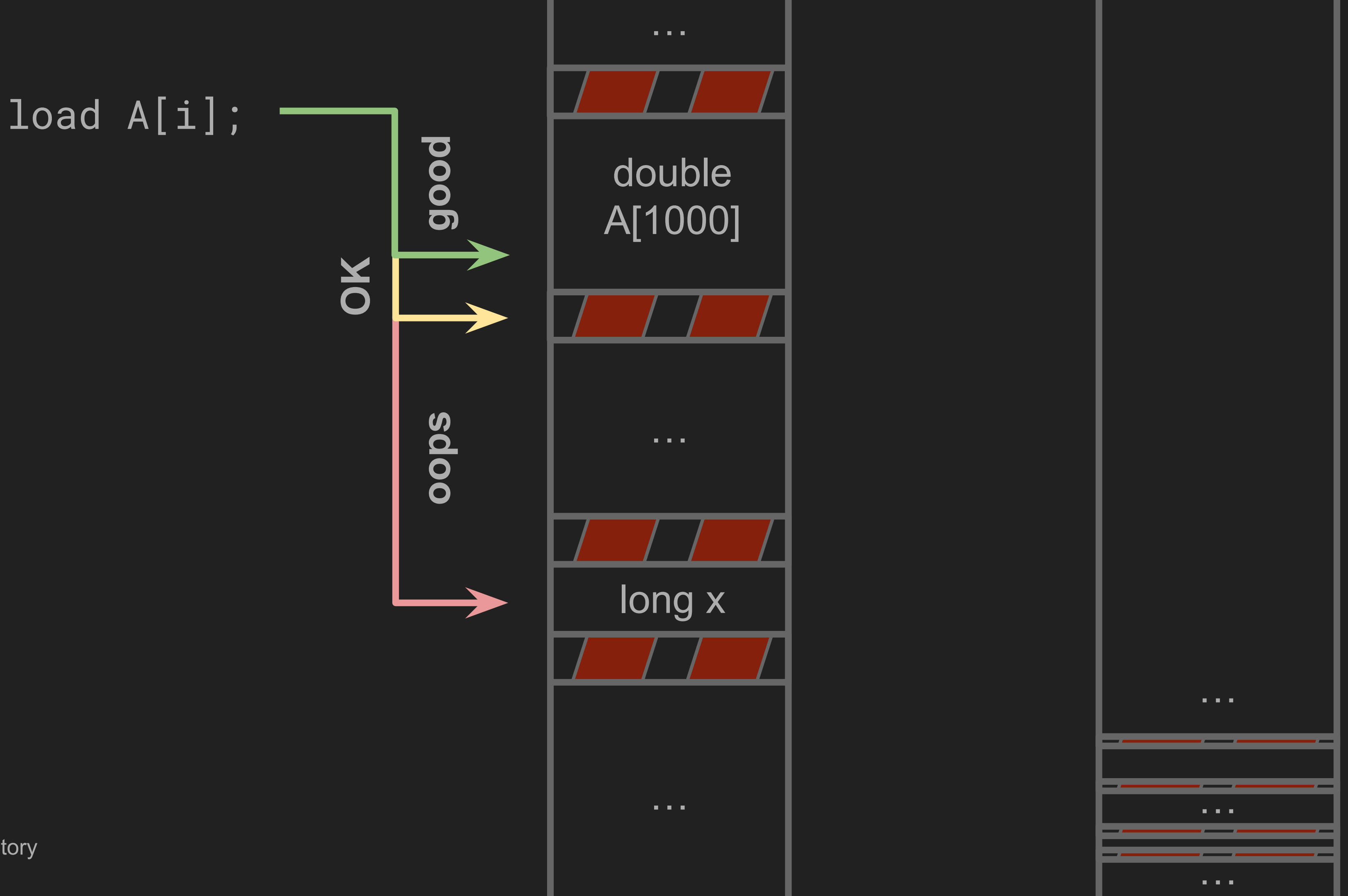
double
A[1000]

…

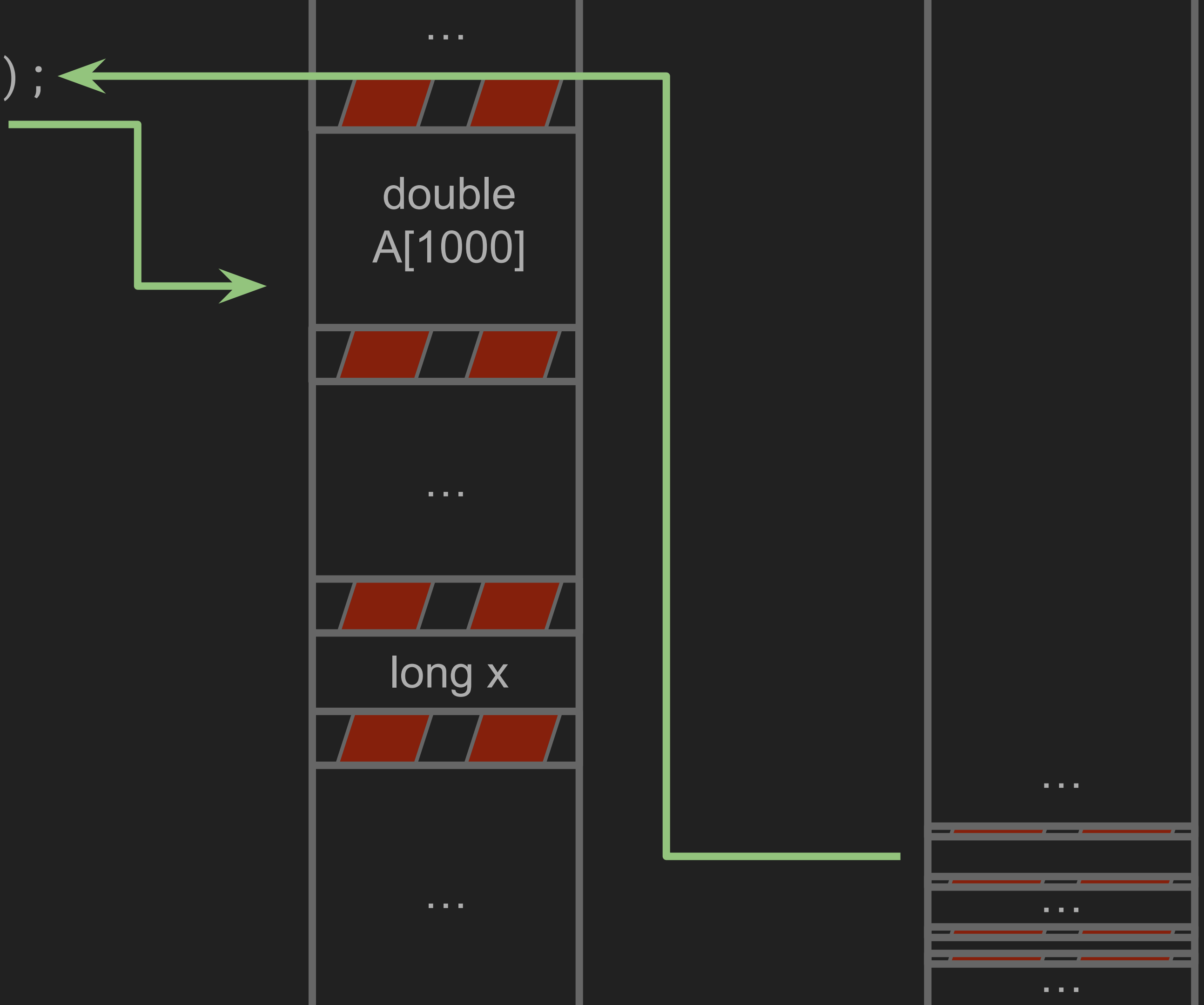
long x

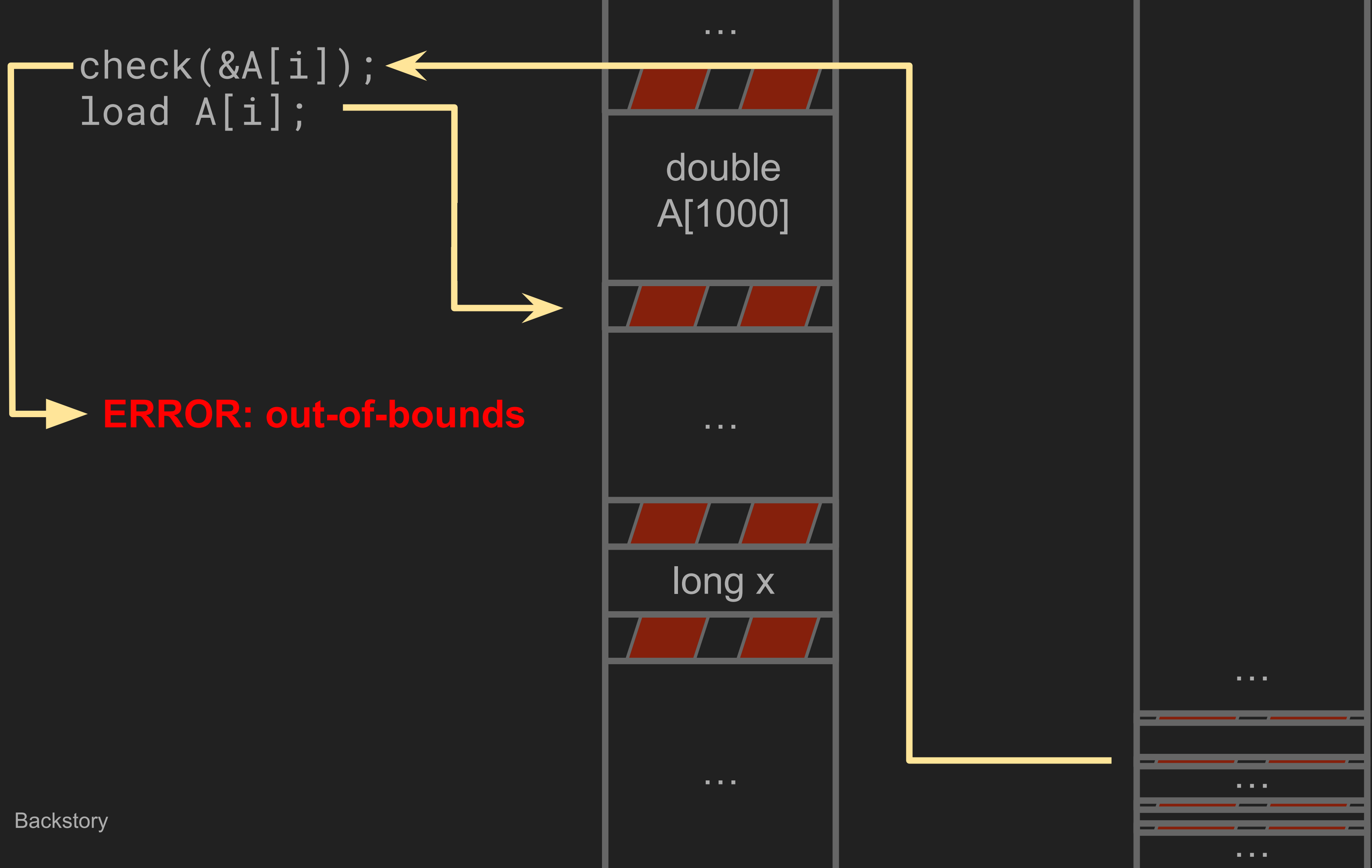
…


…
fixed-sized
red zones
double
A[1000]
…
long x
…
⅛ mapping
…
…
…


load A[i];
good
OK
oops
...
double
A[1000]
...
long x
...
...
...
...

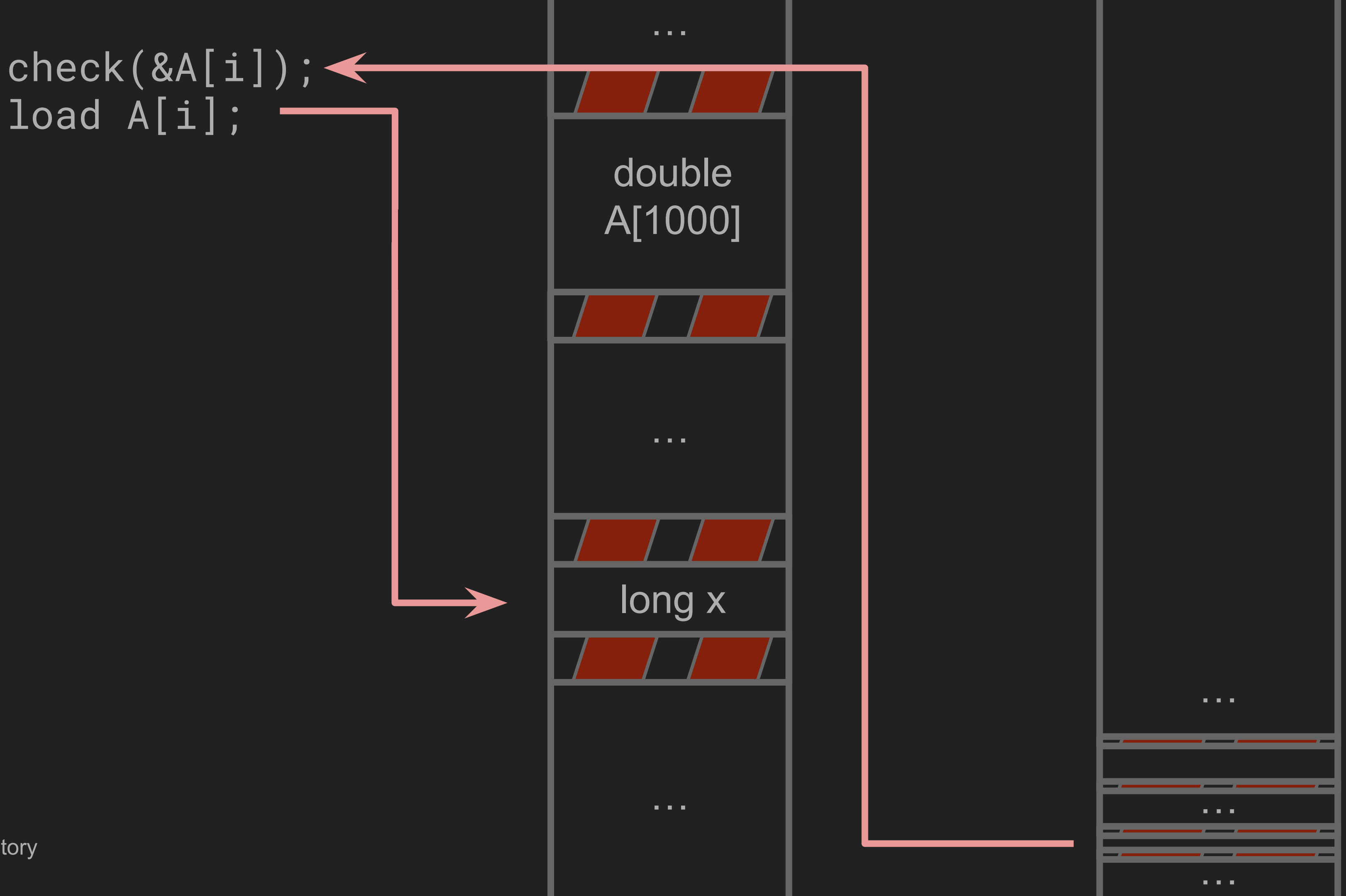
```
check(&A[i]);
load A[i];
```





```
check(&A[i]);
load A[i];
```

ERROR: out-of-bounds



```
check(&A[i]);
load A[i];
```

Recap

CPUs and GPUs are different

- open vs closed environment
- high vs low total memory
- 1 memory vs 3+ memories

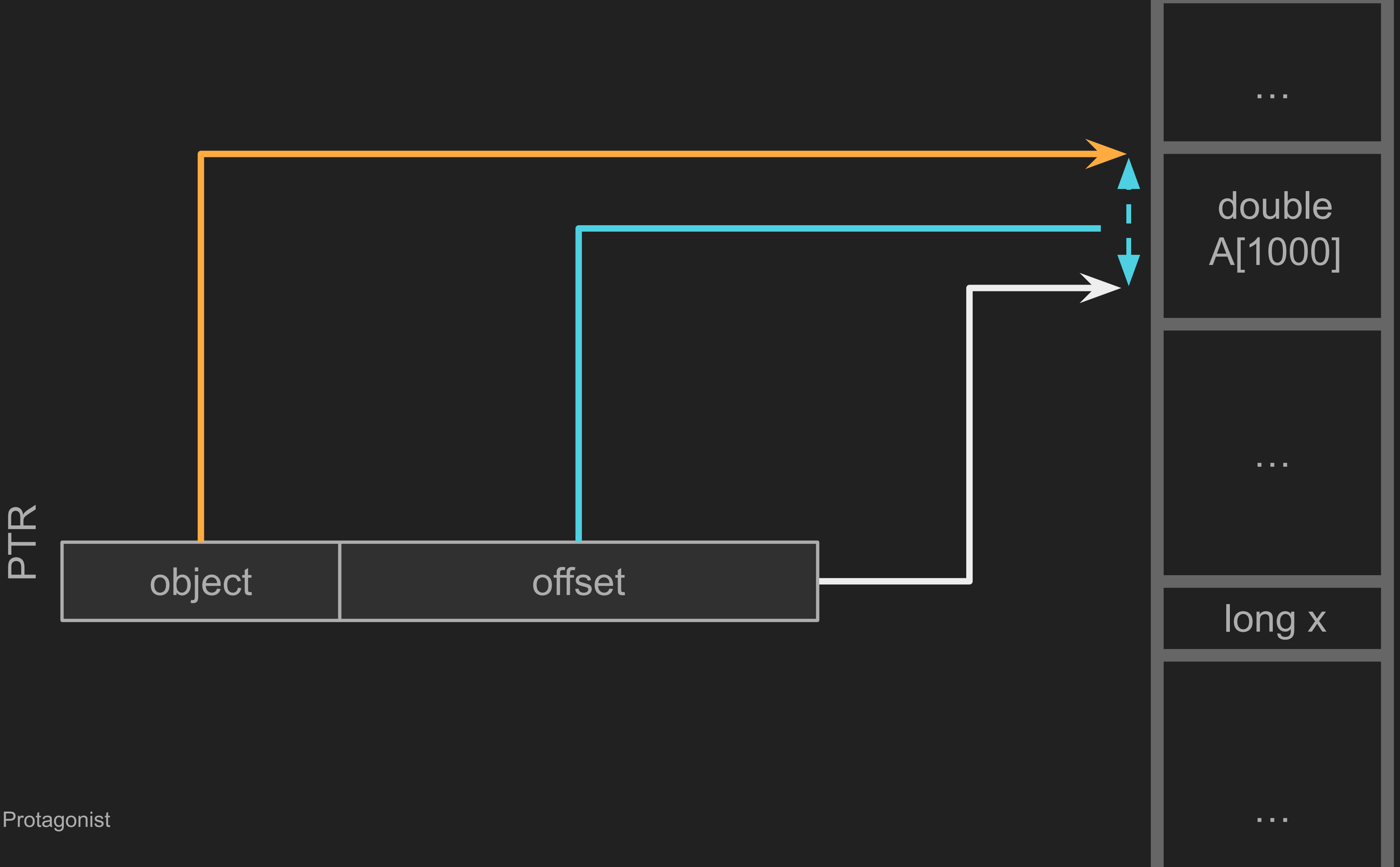
ASAN uses extra memory, accesses

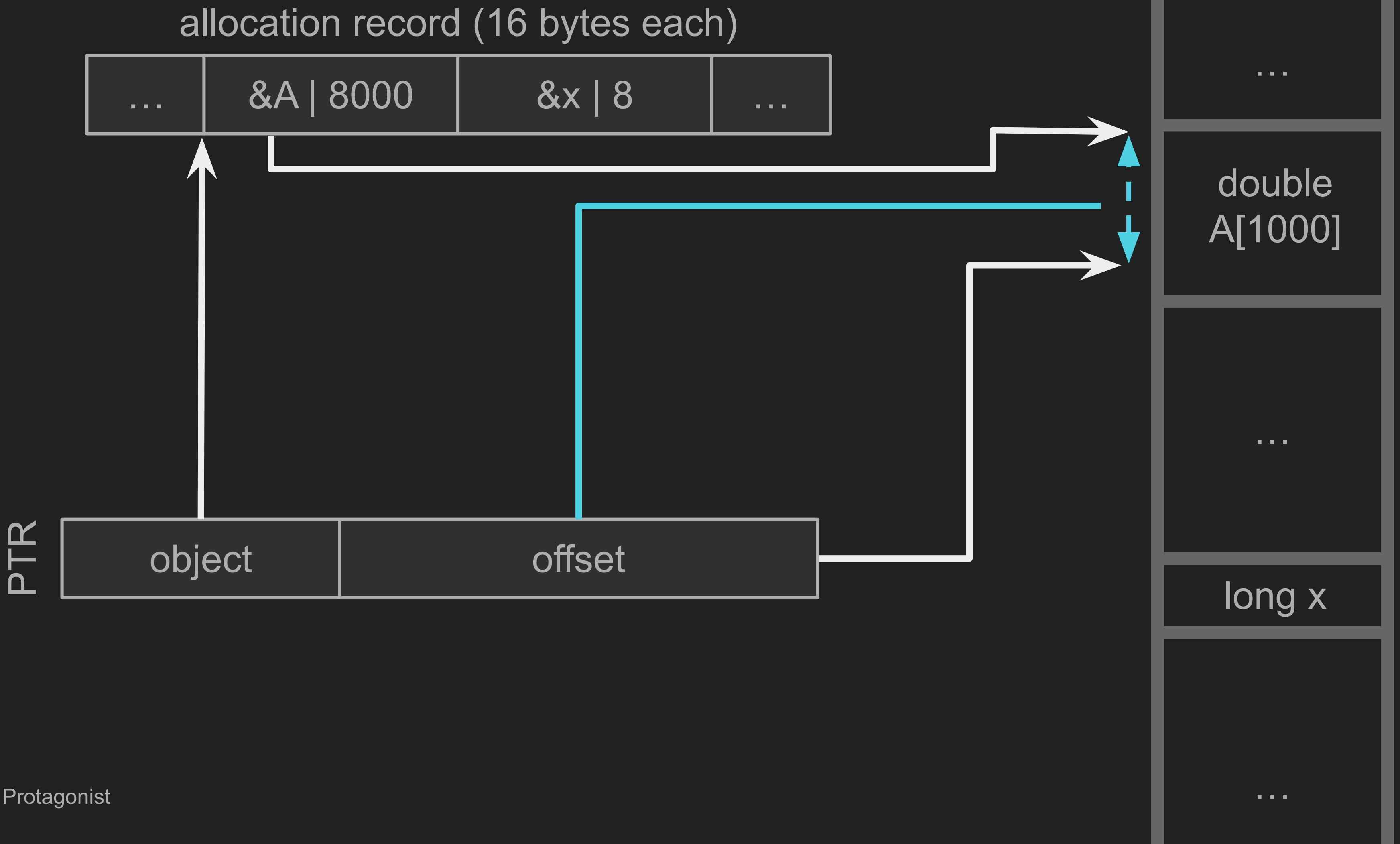
- >1.25x memory per allocation
- 2x number of accesses
- false negatives possible

overheads of 1-2 orders of magnitude

- ASAN increases compile time
- vendor products only,
  non-portable, varying results

Protagonist


PTR
object
offset
…
double A[1000]
…
long x
…


allocation record (16 bytes each)
…
&A | 8000
&x | 8
…
PTR
object
offset
…
double A[1000]
…
long x
…

```
void increment(double *A, int N) {

  for (int I = 0; I < N; ++I) {

    A[I]++;
  }
}
```

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {

    A[I]++;
  }
}
```

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check(Offset + I, Size);
    A[I]++;
  }
}
```

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

allocation record (16 bytes each)

<table><tr><td>…</td><td>&A | 8000</td><td>&x | 8</td><td>…</td></tr></table>

PTR

<table><tr><td>object</td><td>offset</td></tr></table>

allocation record (16 bytes each)

<table>
<tr><td>…</td><td>&A | 8000</td><td>&x | 8</td><td>…</td></tr>
</table>

- on-device registration for each allocation
- 16 bytes load at base pointer definition (`__lookup`)
- bit operations and comparison for each access (`__check`)
- **bad** for 32-bit shared and stack pointers

PTR	object	magic	offset

allocation record (16 bytes each)

…	&A | 8000	&x | 8	…

- on-device registration for each allocation
- 16 bytes load at base pointer definition (`__lookup`)
- bit operations and comparison for each access (`__check`)
- **bad** for 32-bit shared and stack pointers

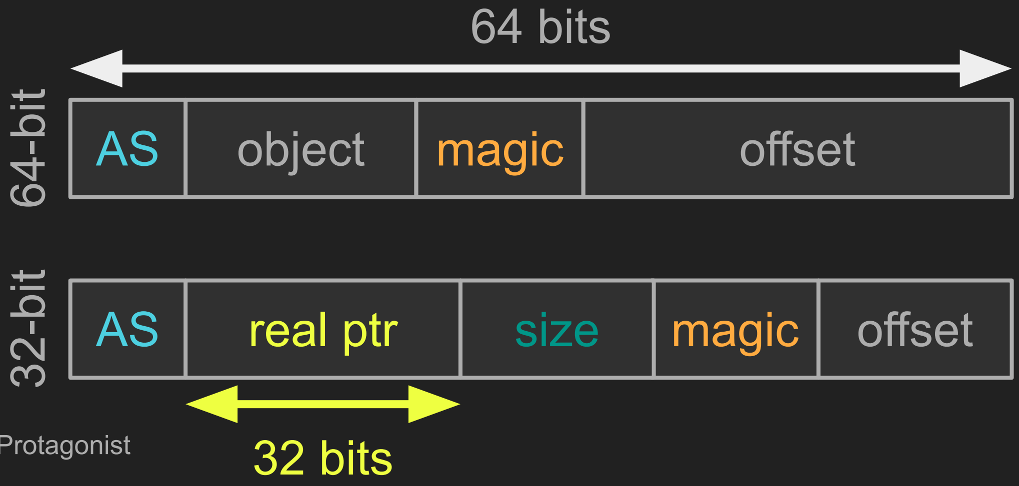
PTR

AS	object	magic	offset

allocation record (16 bytes each)

…	&A | 8000	&x | 8	…

- on-device registration for each allocation
- 16 bytes load at base pointer definition (`__lookup`)
- bit operations and comparison for each access (`__check`)
- **bad** for 32-bit shared and stack pointers



64 bits

64-bit

AS	object	magic	offset

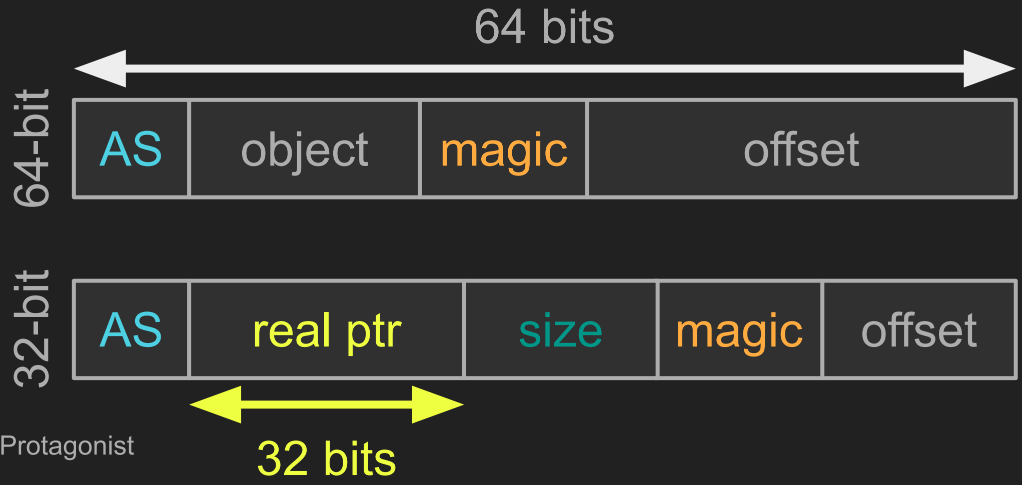
32-bit

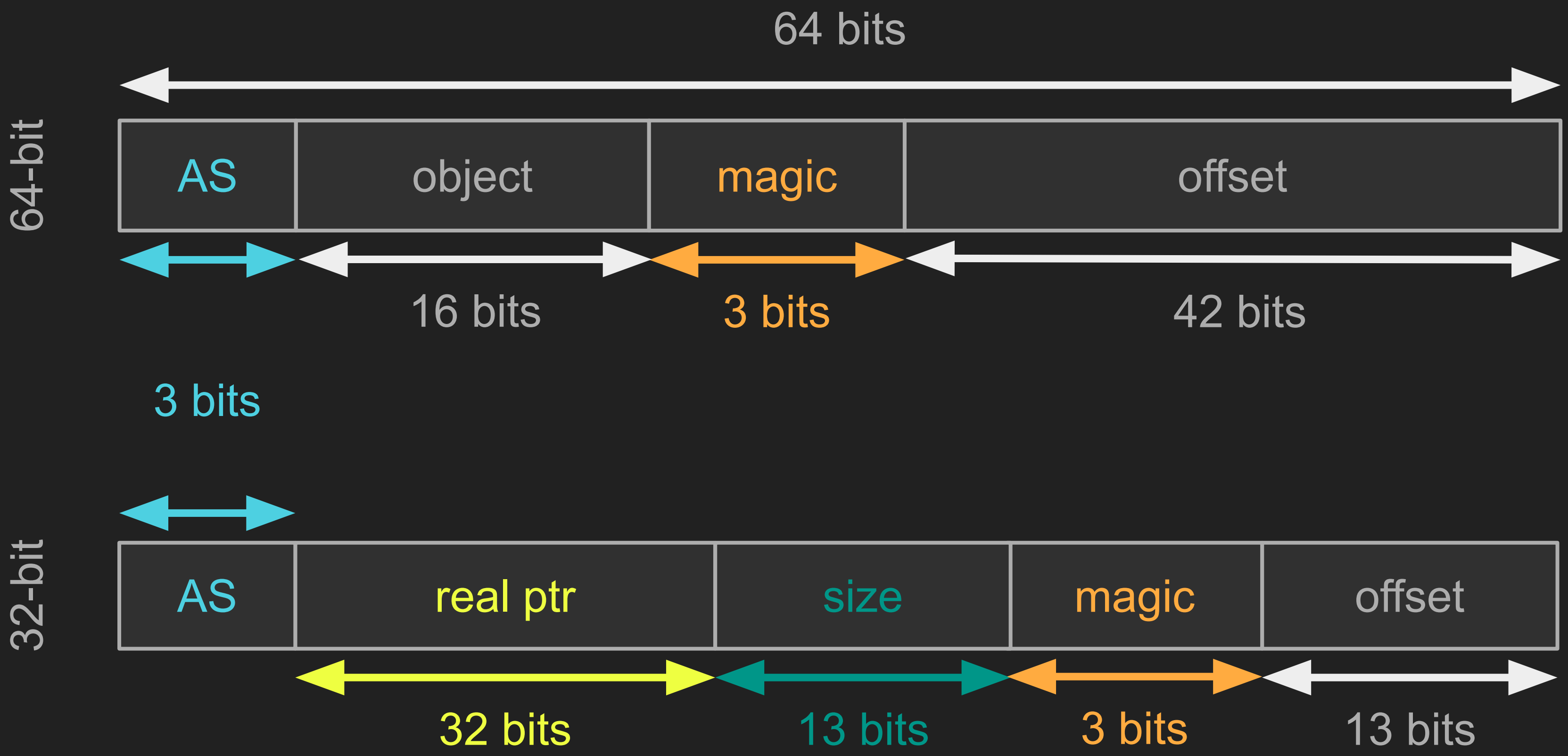
AS	real ptr	size	magic	offset

32 bits

allocation record (16 bytes each)

- on-device registration for each **heap** allocation
- **replace 32-bit pointers with 64-bit pointers** (remove AS)
- **AS check** + 16 bytes load at **heap** base pointer definition (`__lookup`)
- bit operations and comparison for each access (`__check`)
- **no memory overhead** for 32-bit shared and stack pointers




64 bits
64-bit
AS
object
magic
offset
16 bits
3 bits
42 bits
3 bits
32-bit
AS
real ptr
size
magic
offset
32 bits
13 bits
3 bits
13 bits

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
void increment(double *A, int N) {
  auto [AS, Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check(AS, Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
void increment(double AS(3) *A, int N) {
  auto [Base, Size, Offset] = __lookup_as3(A);
  for (int I = 0; I < N; ++I) {
    __check_as3(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
int* __[cuda]malloc(int N) {
  auto DevPtr = [cuda]malloc(N);
  return __register_heap(DevPtr, N);
}
```

```
double [__shared__] A[1000];

void increment(int N) {
  auto [Base, Size, Offset] = __lookup_asX(A);
  for (int I = 0; I < N; ++I) {
    __check_asX(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
double [__shared__] A[1000];
double [__shared__] *A.fake;
void increment(int N) {
  auto [Base, Size, Offset] = __lookup_asX(*A.fake);
  for (int I = 0; I < N; ++I) {
    __check_asX(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
double [__shared__] A[1000];
double [__shared__] *A.fake;

void __init_program() {         // called during startup
  A.fake = __register_global(A, 8000);
}

void __init_kernel() {          // called from kernel
  A.fake = __register_shared(A, 8000);
}
```

Supporting Characters

```
void increment(double *A, int N) {
  for (int I = 0; I < N; ++I)
    A[I]++;
}
```

avoid duplicate checks

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check(Offset + I, Size);
    (Base + Offset + I)[I]++;
  }
}
```

```
void shift(double *A, int N) {
  for (int I = 0; I < N - 2; ++I)
    A[I] = A[I+1] + A[I+2];
}
```

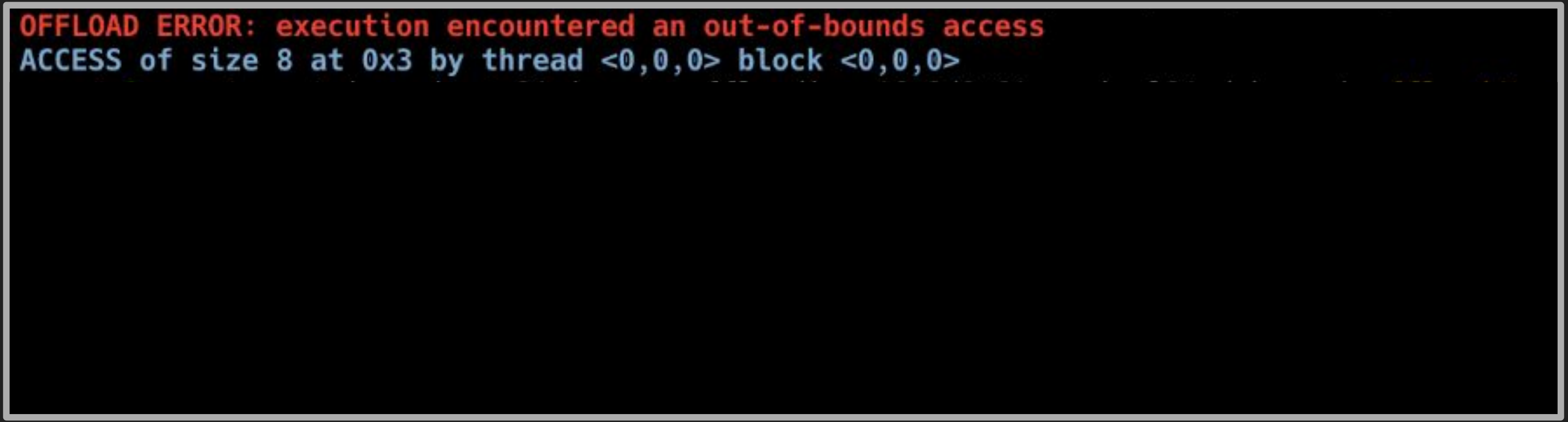
avoid “middle” checks

```
void shift(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N - 2; ++I) {
    __check(Offset + 0, Size);
    __check(Offset + 2, Size);
    (... + I)[I] = (... + I)[I+1] + (... + I)[I+2];
  }
}
```

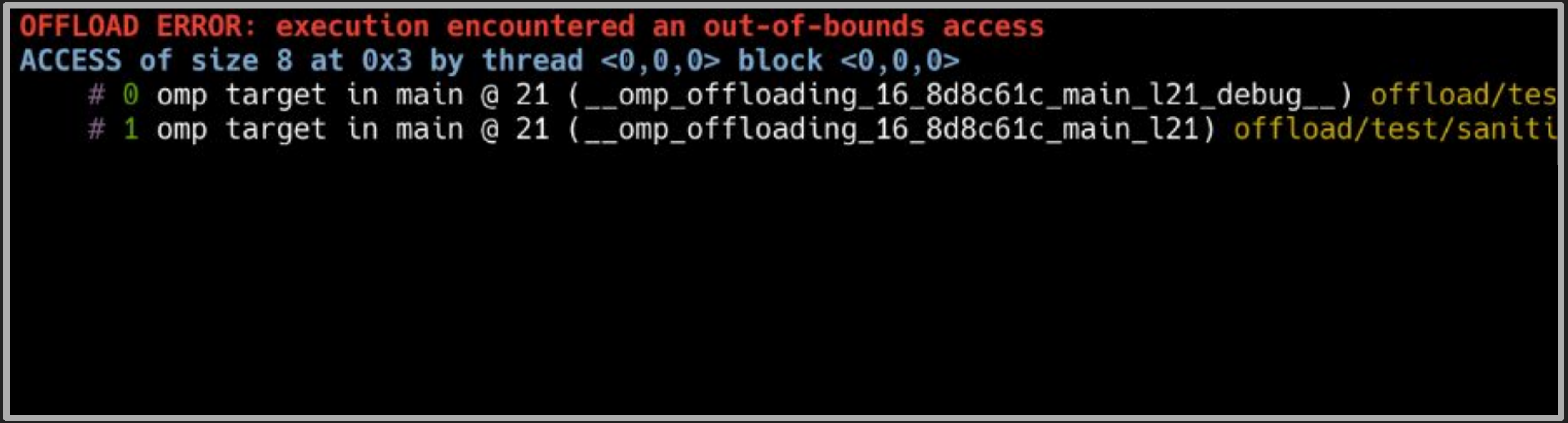
```
double [__shared__] A[1000];

double middle() {
  return A[500];
}
```

avoid checks for known good accesses

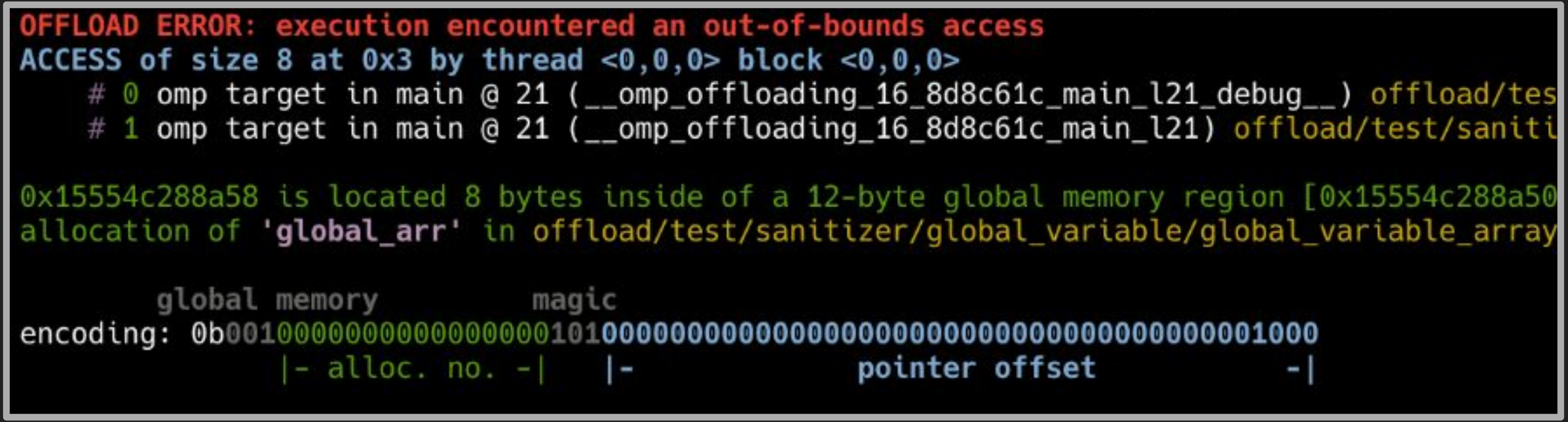
```
double [__shared__] A[1000];

double middle() {
  return A[500];
}
```



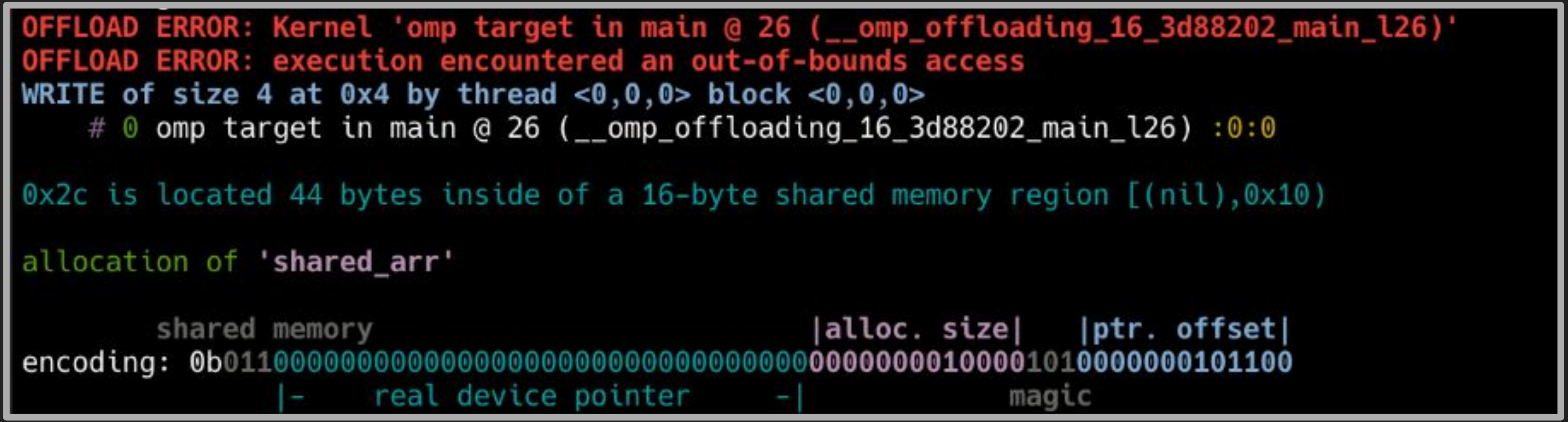
```
OFFLOAD ERROR: execution encountered an out-of-bounds access
ACCESS of size 8 at 0x3 by thread <0,0,0> block <0,0,0>
```



```
OFFLOAD ERROR: execution encountered an out-of-bounds access
ACCESS of size 8 at 0x3 by thread <0,0,0> block <0,0,0>
    # 0 omp target in main @ 21 (__omp_offloading_16_8d8c61c_main_l21_debug__) offload/tes
    # 1 omp target in main @ 21 (__omp_offloading_16_8d8c61c_main_l21) offload/test/saniti
```



```
OFFLOAD ERROR: execution encountered an out-of-bounds access
ACCESS of size 8 at 0x3 by thread <0,0,0> block <0,0,0>
    # 0 omp target in main @ 21 (__omp_offloading_16_8d8c61c_main_l21_debug__) offload/tes
    # 1 omp target in main @ 21 (__omp_offloading_16_8d8c61c_main_l21) offload/test/saniti

0x15554c288a58 is located 8 bytes inside of a 12-byte global memory region [0x15554c288a50
allocation of 'global_arr' in offload/test/sanitizer/global_variable/global_variable_array


        global memory         magic
encoding: 0b0010000000000000000101000000000000000000000000000000000000000001000
               |- alloc. no. -|   |-             pointer offset           -|
```



```
OFFLOAD ERROR: Kernel 'omp target in main @ 26 (__omp_offloading_16_3d88202_main_l26)'
OFFLOAD ERROR: execution encountered an out-of-bounds access
WRITE of size 4 at 0x4 by thread <0,0,0> block <0,0,0>
    # 0 omp target in main @ 26 (__omp_offloading_16_3d88202_main_l26) :0:0

0x2c is located 44 bytes inside of a 16-byte shared memory region [(nil),0x10)

allocation of 'shared_arr'

        shared memory                          |alloc. size|   |ptr. offset|
encoding: 0b011000000000000000000000000000000000000000010000101000000000101100
               |-    real device pointer     -|            magic
```

Synopsis

Reported Errors

allocation too large	✔	4TB for heap or 8KB for stack and shared
bad pointer	✔	magic mismatch, e.g., due to large out-of-bounds
out-of-bounds	✔	offset negative or larger than size
use-after-free	✔	size was negative due to deallocation (heap only)
wrong address space	✔	address space bits did not match compile time value

no false positives/negatives for out-of-bounds

low memory

avoid memory overhead

memory bound

avoid memory accesses, especially in loops

different memories

fast/cache memory
⇒ fast verification

stack memory (per thread)
⇒ no memory overhead

“slow”/large heap
⇒ fixed overhead per allocation

no false positives/negatives for out-of-bounds