Imperial College London

SOFTWARE RELIABILITY GROUP

COM SYS Communication and Distributed Systems

RWTH AACHEN UNIVERSITY

A Deterministic Memory Allocator for Dynamic Symbolic Execution

Daniel Schemmel, Julian Büning, Frank Busse, Martin Nowack, Cristian Cadar

Dynamic Symbolic Execution

```
int x = input();
if (x == 0) {
    abort();
} else {
    return x;
}
```

Dynamic Symbolic Execution

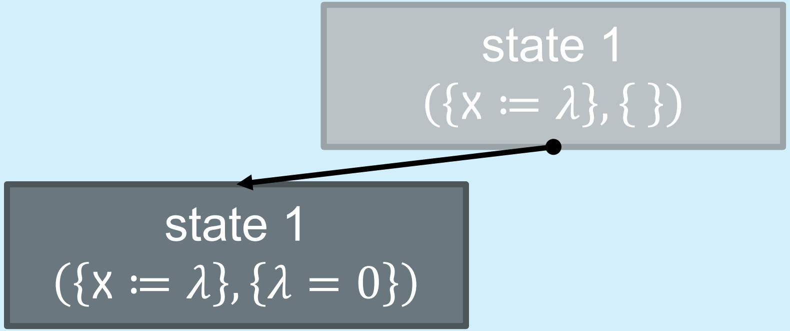
```
int x = input();
if (x == 0) {
    abort();
} else {
    return x;
}
```

state 1

$(\{x := \lambda\}, \{\ \})$

Dynamic Symbolic Execution

```
int x = input();
if (x == 0) {
    abort();
} else {
    return x;
}
```



state 1

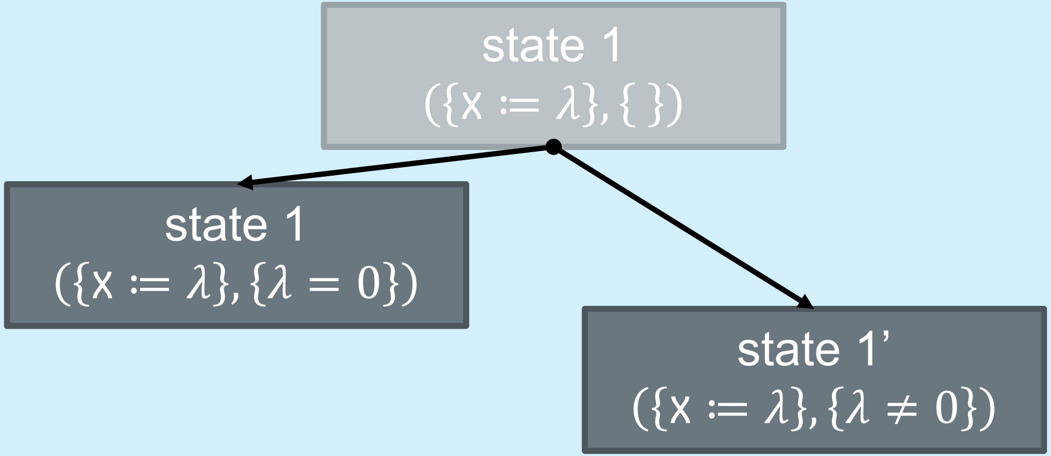
$(\{x := \lambda\}, \{\,\})$

state 1

$(\{x := \lambda\}, \{\lambda = 0\})$

Dynamic Symbolic Execution

```
int x = input();
if (x == 0) {
    abort();
} else {
    return x;
}
```



state 1
$(\{x := \lambda\}, \{\ \})$

state 1
$(\{x := \lambda\}, \{\lambda = 0\})$

state 1’
$(\{x := \lambda\}, \{\lambda \neq 0\})$

Address Translation in KLEE

- What is the address of `x`?

Address Translation in KLEE

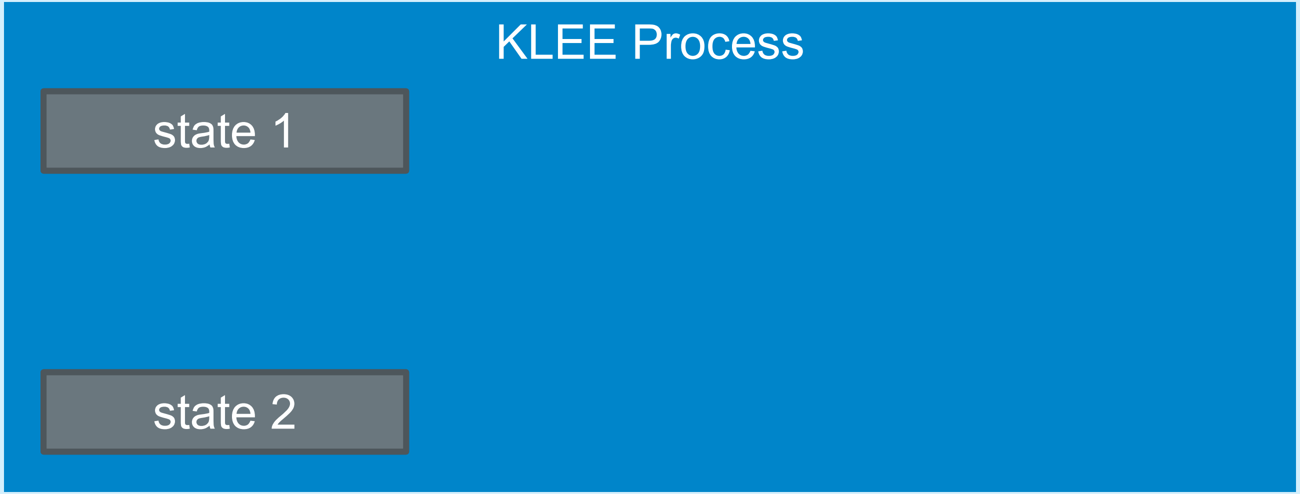
- What is the address of `x`?

KLEE Process

Imperial College London

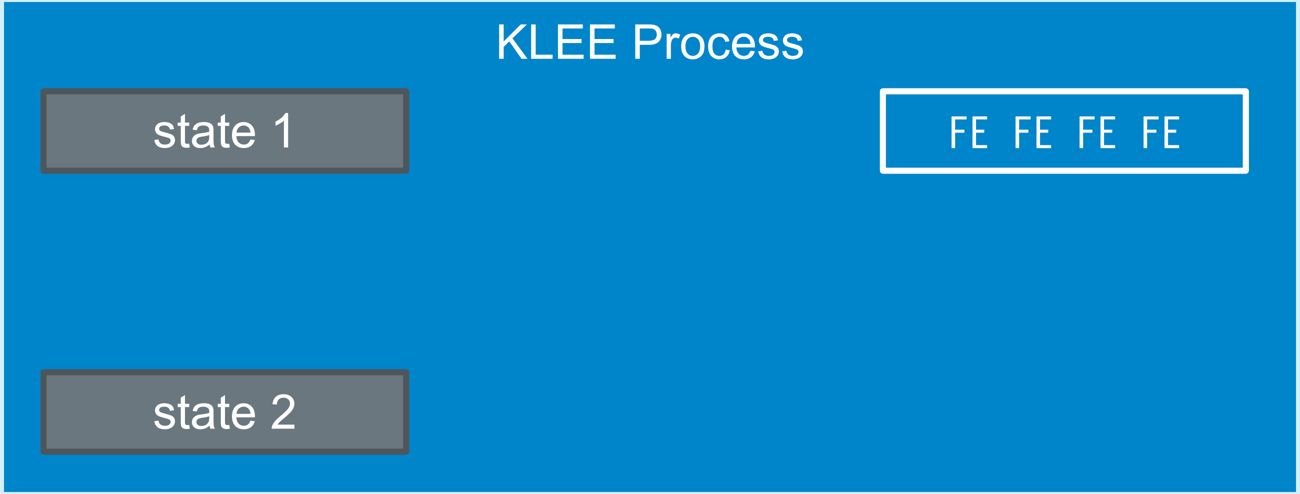
Address Translation in KLEE

- What is the address of `x`?



Address Translation in KLEE

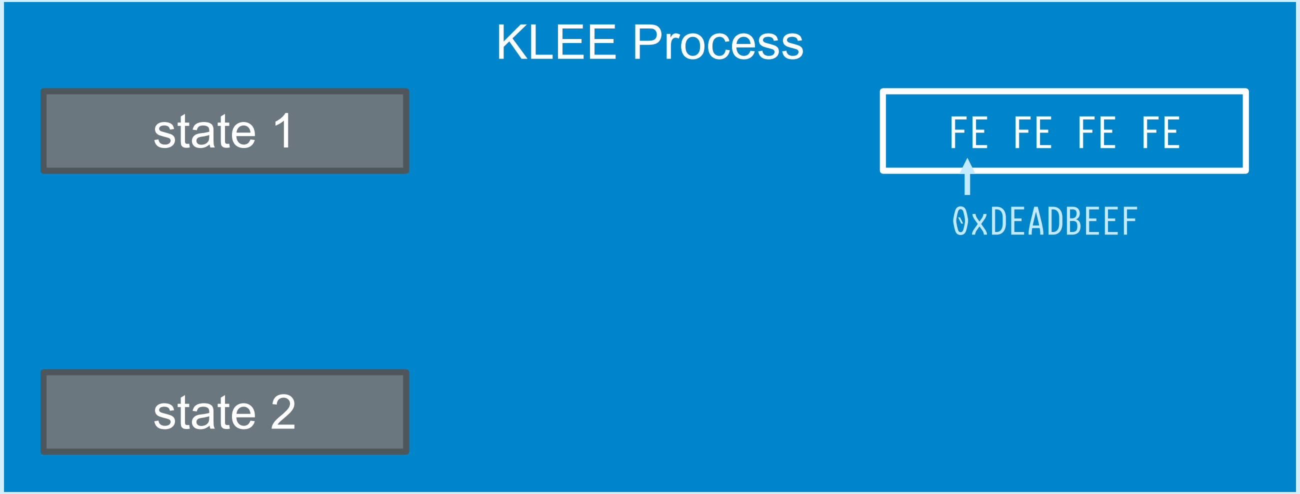
- What is the address of `x`?



Imperial College London

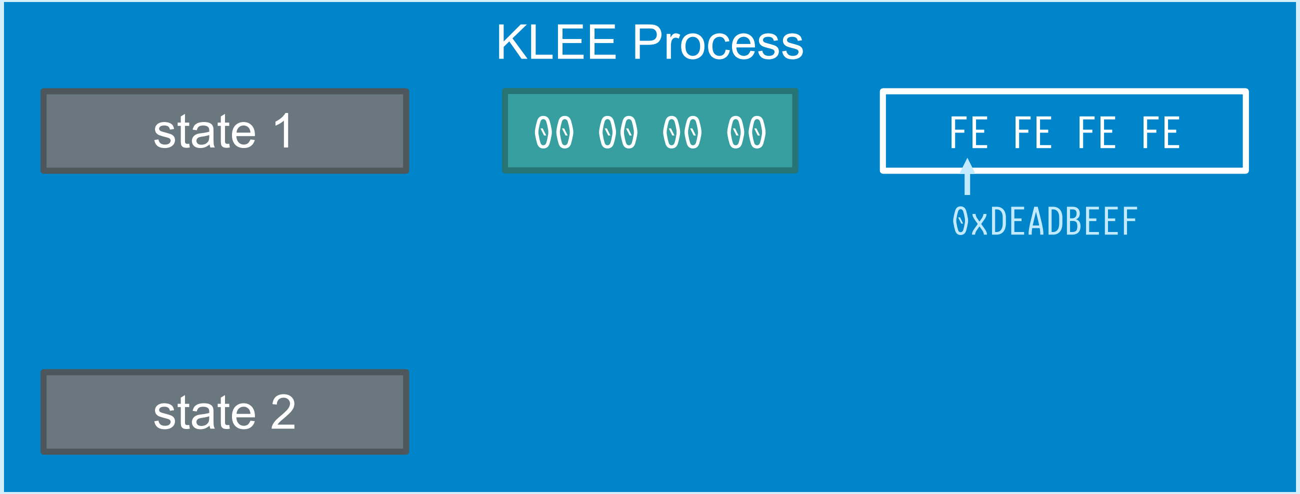
Address Translation in KLEE

- What is the address of `x`?



Address Translation in KLEE

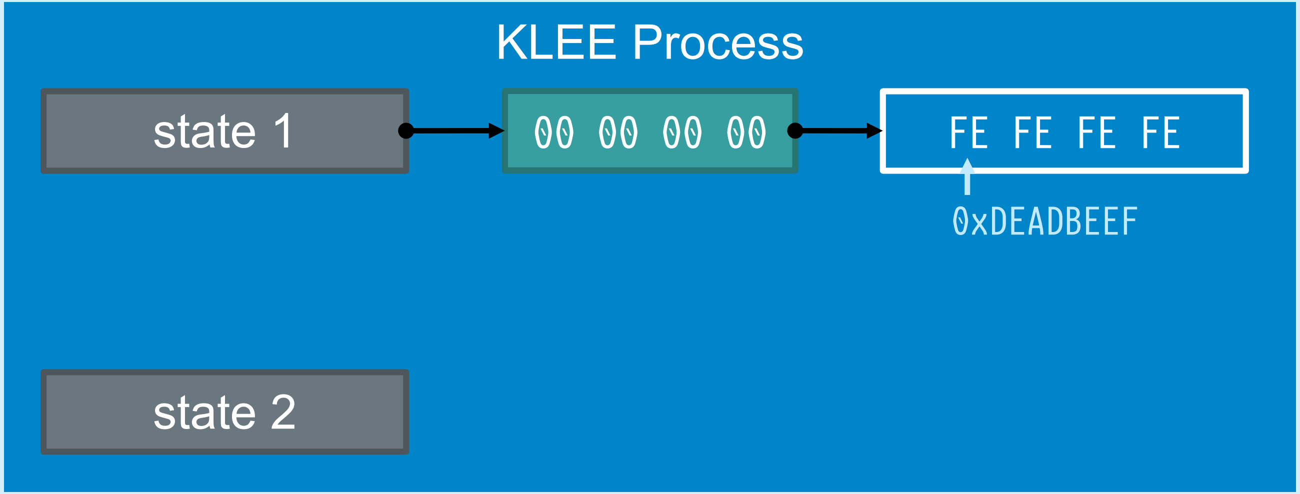
- What is the address of `x`?



Imperial College London

Address Translation in KLEE

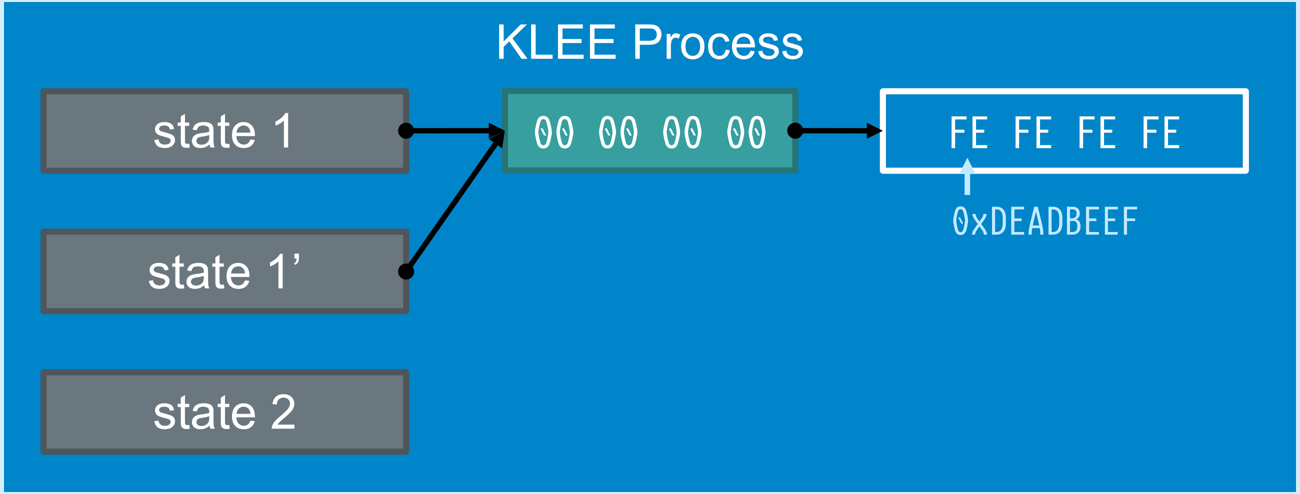
- What is the address of `x`?



Imperial College London

Address Translation in KLEE

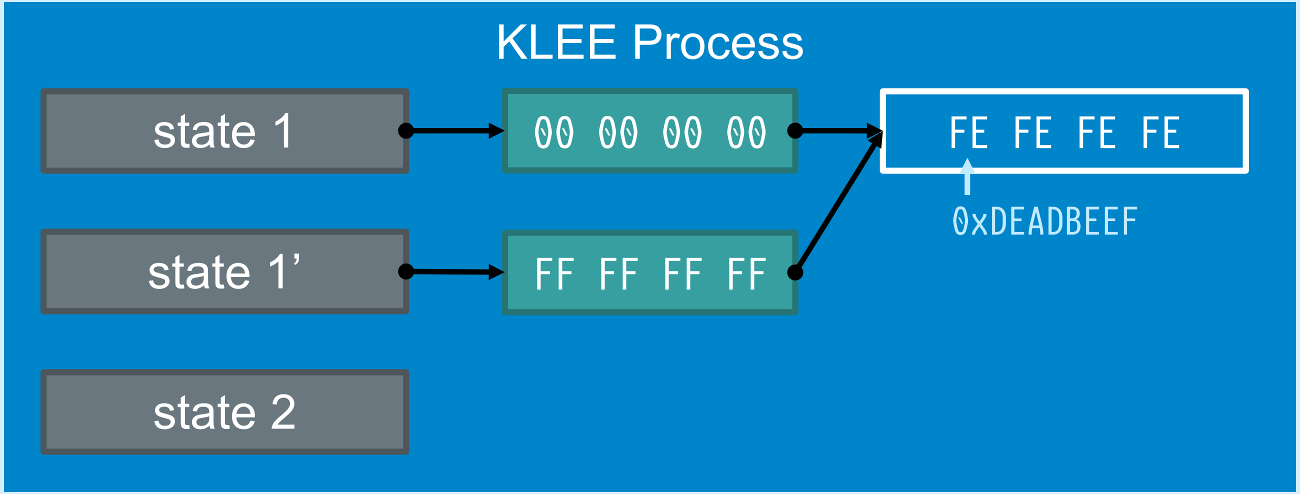
- What is the address of `x`?



Imperial College London

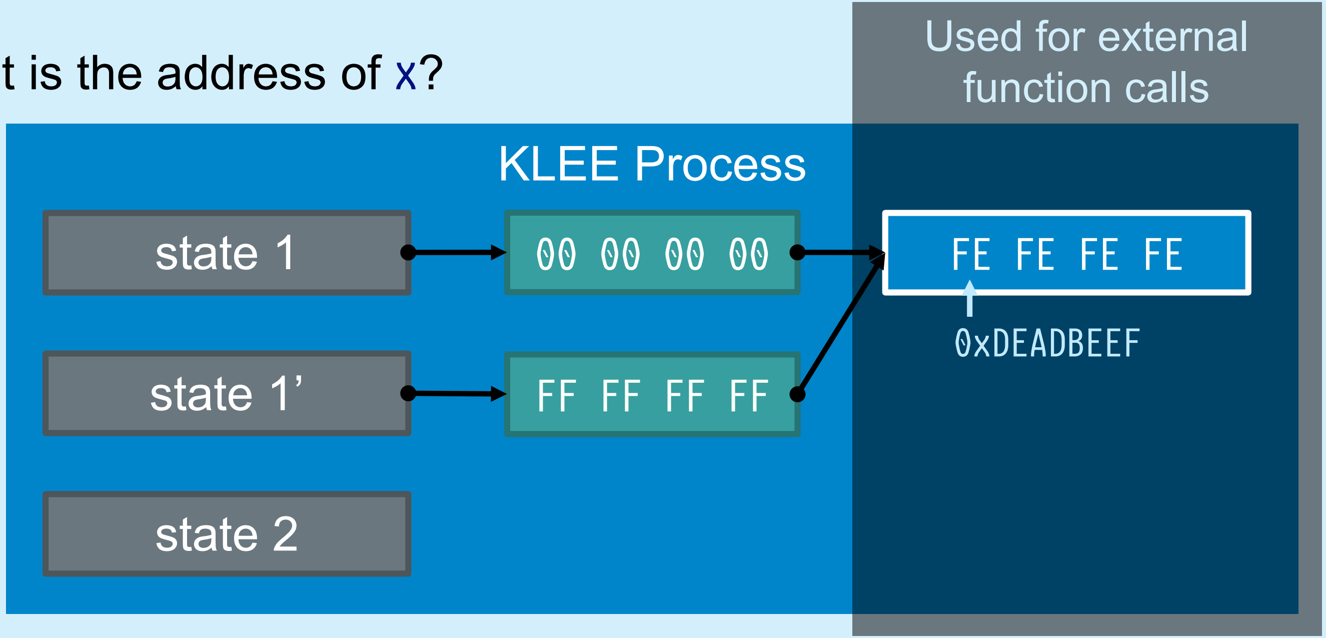
Address Translation in KLEE

- What is the address of x?



Address Translation in KLEE

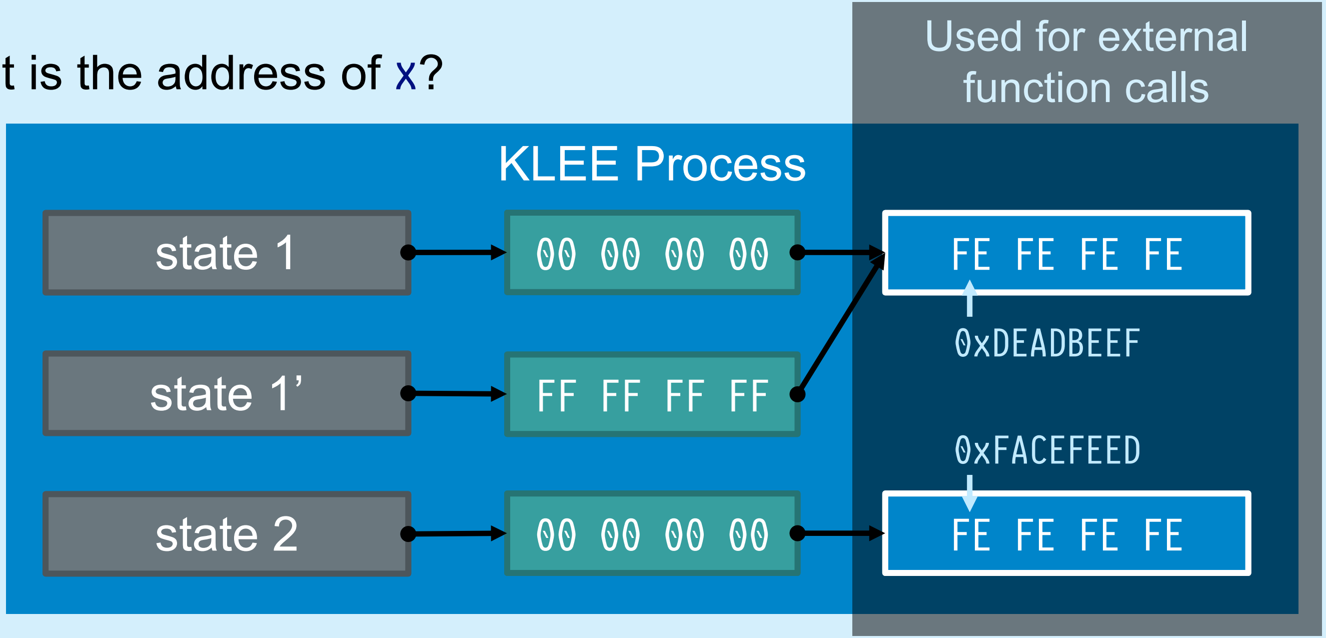
- What is the address of x?



Imperial College London

Address Translation in KLEE

- What is the address of `x`?



Imperial College London

The Need for Deterministic Memory Allocation

- For experiments to be repeatable, memory allocation must be repeatable

Imperial College London

The Need for Deterministic Memory Allocation

- For experiments to be repeatable, memory allocation must be repeatable
- Advanced symbolic execution techniques benefit from or outright require deterministic execution

The Need for Deterministic Memory Allocation

- For experiments to be repeatable, memory allocation must be repeatable

- Advanced symbolic execution techniques benefit from or outright require deterministic execution
  - POR-SE [Symbolic partial-order execution for testing multi-threaded programs. Schemmel et al. CAV 2020]

The Need for Deterministic Memory Allocation

- For experiments to be repeatable, memory allocation must be repeatable
- Advanced symbolic execution techniques benefit from or outright require deterministic execution
  - POR-SE [Symbolic partial-order execution for testing multi-threaded programs. Schemmel et al. CAV 2020]
  - SYMLIVE [Symbolic liveness analysis of real-world software. Schemmel et al. CAV 2018]

Imperial College
London

The Need for Deterministic Memory Allocation

- For experiments to be repeatable, memory allocation must be repeatable

- Advanced symbolic execution techniques benefit from or outright require deterministic execution
  - POR-SE [Symbolic partial-order execution for testing multi-threaded programs. Schemmel et al. CAV 2020]
  - SymLive [Symbolic liveness analysis of real-world software. Schemmel et al. CAV 2018]
  - MoKLEE [Running symbolic execution forever. Busse et al. ISSTA 2020]

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:

KDALLOC

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
  1. Support for external calls (addresses valid in host process)

KDALLOC

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
    1. Support for external calls (addresses valid in host process)
    2. Cross-run determinism (multiple runs should behave the same)

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
  1. Support for external calls (addresses valid in host process)
  2. Cross-run determinism (multiple runs should behave the same)
  3. Cross-path determinism (multiple paths should behave the same)

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
    1. Support for external calls (addresses valid in host process)
    2. Cross-run determinism (multiple runs should behave the same)
    3. Cross-path determinism (multiple paths should behave the same)
    4. Spatially distanced allocations (misindexing an array should trap)

Imperial College London

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
  1. Support for external calls (addresses valid in host process)
  2. Cross-run determinism (multiple runs should behave the same)
  3. Cross-path determinism (multiple paths should behave the same)
  4. Spatially distanced allocations (misindexing an array should trap)
  5. Temporally distanced allocations (use-after-free should trap)

KDAlloc

- An allocator specifically for dynamic symbolic execution can do better!
- Important properties:
  1. Support for external calls (addresses valid in host process)
  2. Cross-run determinism (multiple runs should behave the same)
  3. Cross-path determinism (multiple paths should behave the same)
  4. Spatially distanced allocations (misindexing an array should trap)
  5. Temporally distanced allocations (use-after-free should trap)
  6. Stability (minor changes should not snowball)

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state
  - This region is only used to provide addresses and for external calls

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state
  - This region is only used to provide addresses and for external calls
  - Object data is already state-dependent

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state
  - This region is only used to provide addresses and for external calls
  - Object data is already state-dependent

- Categorize allocations to reduce snowball effect

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state
  - This region is only used to provide addresses and for external calls
  - Object data is already state-dependent

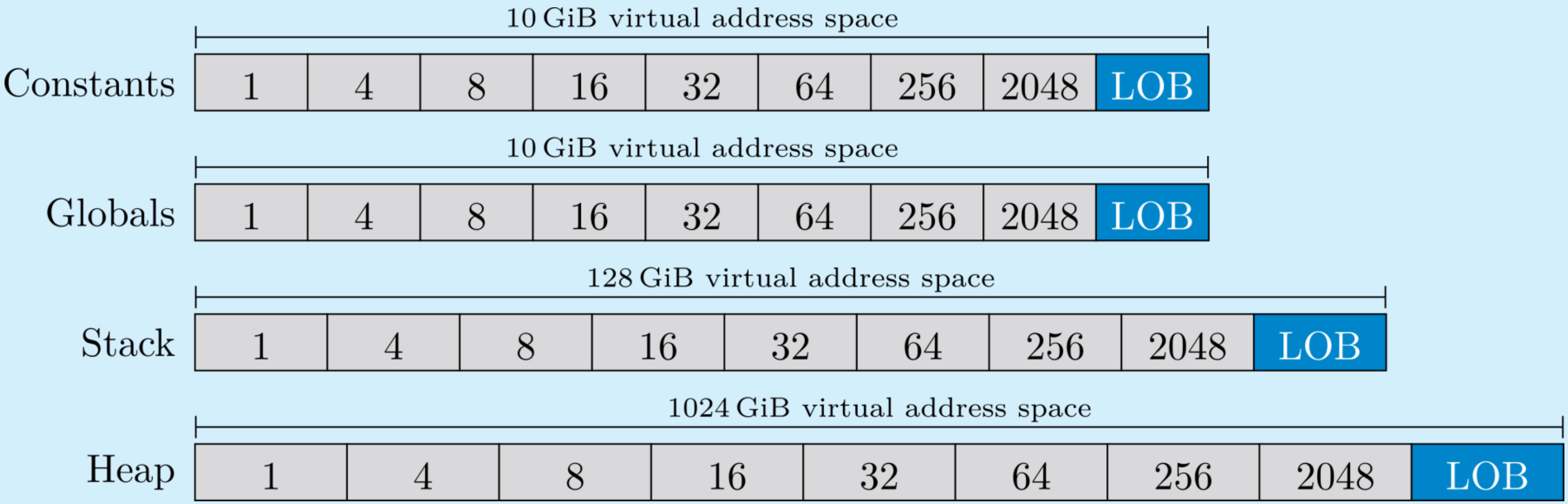
- Categorize allocations to reduce snowball effect
  - Multiple allocators, especially to disconnect stack and heap

General Architecture

- `mmap` one large region and attach forkable metadata to the initial state
  - This region is only used to provide addresses and for external calls
  - Object data is already state-dependent

- Categorize allocations to reduce snowball effect
  - Multiple allocators, especially to disconnect stack and heap
  - Binned allocations

Memory Layout for KDAlloc



10 GiB virtual address space

Constants: 1 | 4 | 8 | 16 | 32 | 64 | 256 | 2048 | LOB

10 GiB virtual address space

Globals: 1 | 4 | 8 | 16 | 32 | 64 | 256 | 2048 | LOB

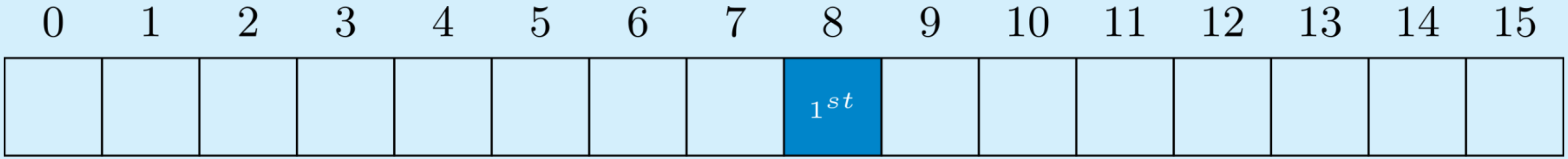
128 GiB virtual address space

Stack: 1 | 4 | 8 | 16 | 32 | 64 | 256 | 2048 | LOB

1024 GiB virtual address space

Heap: 1 | 4 | 8 | 16 | 32 | 64 | 256 | 2048 | LOB

Slot Allocator for Sized Bins: Spatially Distanced

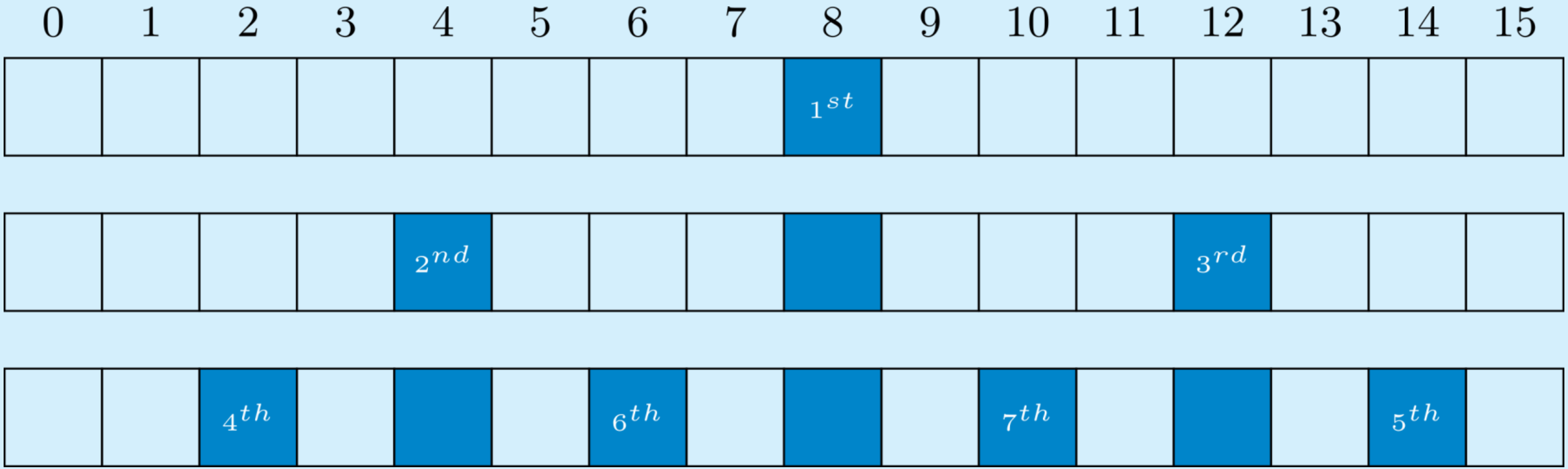


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
								1^{st}							

Slot Allocator for Sized Bins: Spatially Distanced

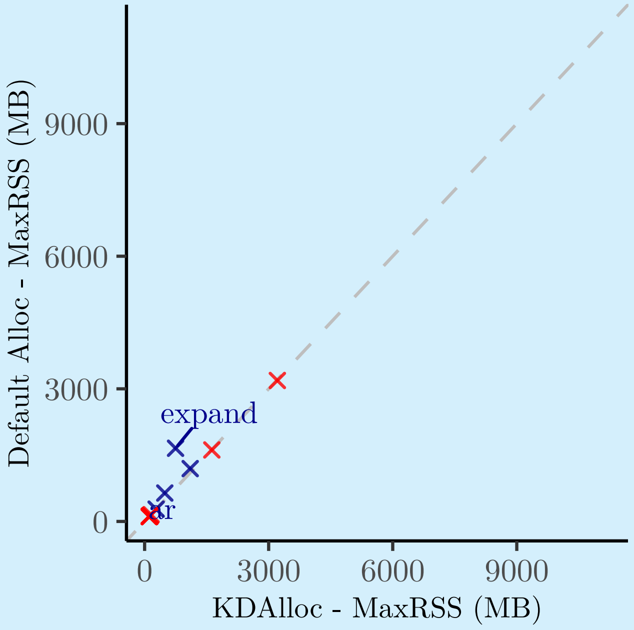
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
								1st							
				2nd				■				3rd			

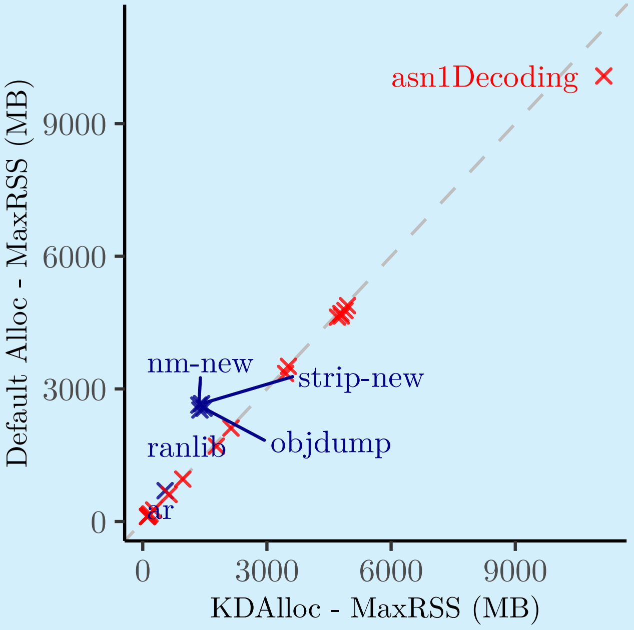
Slot Allocator for Sized Bins: Spatially Distanced



Memory Consumption

DFS

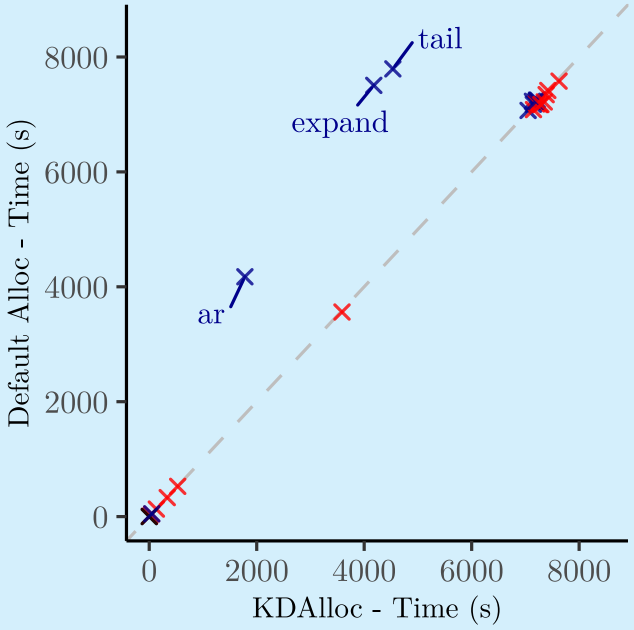


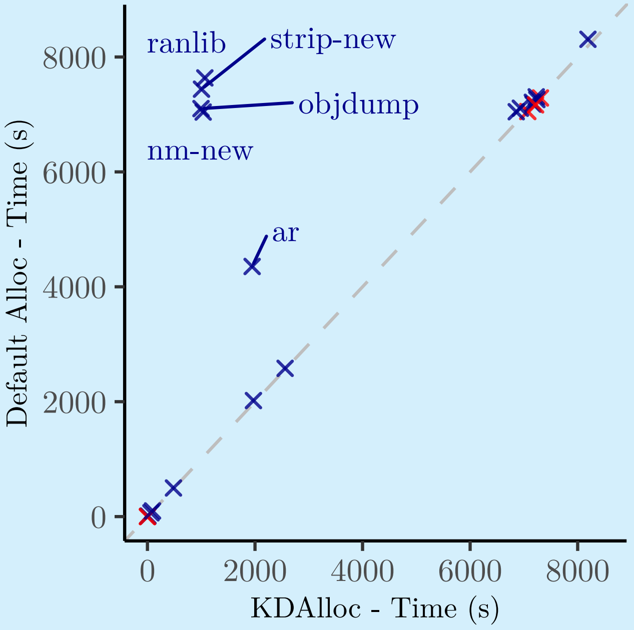


RndCov

Performance

DFS



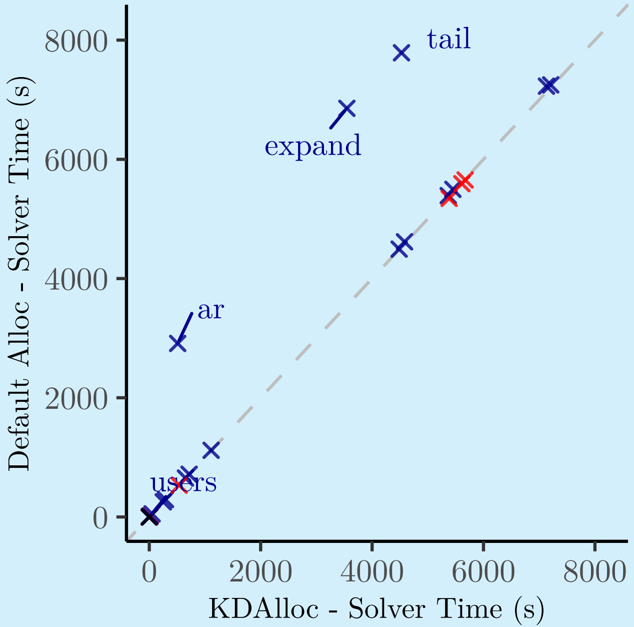


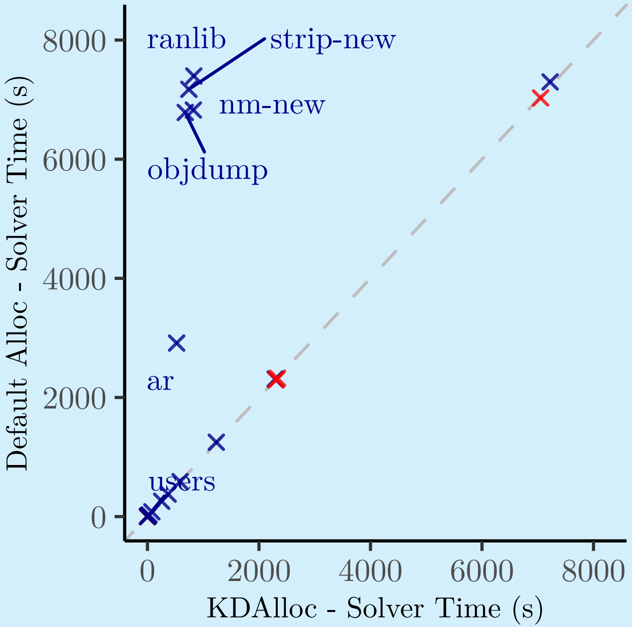
RNDCOV

Imperial College London

Solver Time

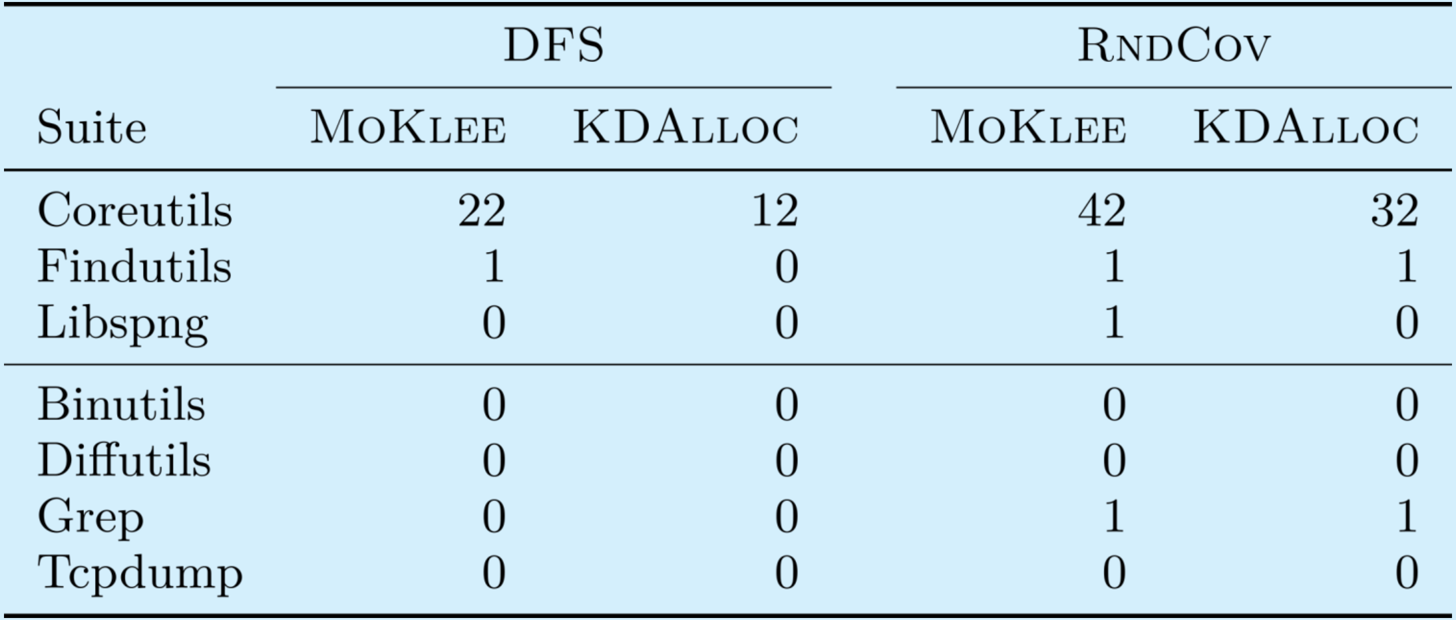
DFS





RndCov

MoKlee: Fewer Diverging Locations



	DFS		RndCov	
Suite	MoKlee	KDAlloc	MoKlee	KDAlloc
Coreutils	22	12	42	32
Findutils	1	0	1	1
Libspng	0	0	1	0
Binutils	0	0	0	0
Diffutils	0	0	0	0
Grep	0	0	1	1
Tcpdump	0	0	0	0

MoKlee: Fewer Divergences in `memmove`

```c
char* s = (char*)dest;
const char* p = (const char*)src;
if (p >= s) {
    while (n) {
        *s++ = *p++;
        --n;
    }
} else {
    while (n) {
        --n;
        s[n] = p[n];
    }
}
return dest;
```

MoKlee: Fewer Divergences in `memmove`

```
char* s = (char*)dest;
const char* p = (const char*)src;
if (p >= s) {
    while (n) {
        *s++ = *p++;
        --n;
    }
} else {
    while (n) {
        --n;
        s[n] = p[n];
    }
}
return dest;
```

MoKlee: Fewer Divergences in memmove

```
char* s = (char*)dest;
const char* p = (const char*)src;
if (p >= s) {
    while (n) {
        *s++ = *p++;
        --n;
    }
} else {
    while (n) {
        --n;
        s[n] = p[n];
    }
}
return dest;
```

- uClibc’s memmove is sensitive to memory layout

Imperial College London

Summary & Conclusion

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

- We implemented KDALLOC in KLEE and show:

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

- We implemented KDALLOC in KLEE and show:
  - Performance and memory consumption are not impacted negatively

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

- We implemented KDAlloc in KLEE and show:
  - Performance and memory consumption are not impacted negatively
  - Use-after-free detection is improved (general benefit)

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

- We implemented KDALLOC in KLEE and show:
  - Performance and memory consumption are not impacted negatively
  - Use-after-free detection is improved (general benefit)
  - Specific benefits for multiple DSE-based techniques

Summary & Conclusion

- The memory allocator has significant impact on dynamic symbolic execution

- We implemented KDAlloc in KLEE and show:
  - Performance and memory consumption are not impacted negatively
  - Use-after-free detection is improved (general benefit)
  - Specific benefits for multiple DSE-based techniques

- KDAlloc is becoming part of mainline KLEE!

Guaranteed Use-After-Free Behavior

```
char *mallocfree() {
    char *s = strdup("A");
    free(s);
    char *t = strdup("B");
    return s;
}
```

```
int main(void) {
    char *s = mallocfree();
    puts(s);
    return 0;
}
```

- KDAlloc guarantees detection when quarantine is enabled

Query Structure with KDAlloc

```
(Extract w32 0
    (Add w64 0xFFFFDDBC00000000 (Select w64 C 0x0000000000000000 0x0000224400000000)))
        ↓ Extract(Add): (Extract (Add x y)) → (Add (Extract x) (Extract y))
(Add w32 (Extract w32 0 0xFFFFDDBC00000000)
    (Extract w32 0 (Select w64 C 0x0000000000000000 0x0000224400000000)))
        (Extract w32 0 0xFFFFDDBC00000000) → 0x00000000 and ↓ Extract(Select)
(Add w32 0x00000000
    (Select w32 C (Extract w32 0 0x0000000000000000) (Extract w32 0 0x0000224400000000)))
        (Extract w32 0 0x0000000000000000) → 0x00000000
        (Extract w32 0 0x0000224400000000) → 0x00000000
(Add w32 0x00000000 (Select w32 C 0x00000000 0x00000000))
        (Select w32 C 0x00000000 0x00000000) → 0x00000000
(Add w32 0x00000000 0x00000000) = 0x00000000
```

Query Structure without KDAlloc

```
(Extract w32 0
    (Add w64 0xFFFFAAAAA7290C00 (Select w64 C 0x0000000000000000 0x0000555558D6F400)))
  ↓ Extract(Add): (Extract (Add x y)) → (Add (Extract x) (Extract y))
(Add w32 (Extract w32 0 0xFFFFAAAAA7290C00)
    (Extract w32 0 (Select w64 C 0x0000000000000000 0x0000555558D6F400)))
  (Extract w32 0 0xFFFFAAAAA7290C00) → 0xA7290C00 and ↓ Extract(Select)
(Add w32 0xA7290C00
    (Select w32 C (Extract w32 0 0x0000000000000000) (Extract w32 0 0x0000555558D6F400)))
  (Extract w32 0 0x0000000000000000) → 0x00000000
  (Extract w32 0 0x0000555558D6F400) → 0x58D6F400
(Add w32 0xA7290C00 (Select w32 C 0x00000000 0x58D6F400))
```