A Pointer Tracking Memory Model for KLEE

Felix Rath,
Daniel Schemmel, Oscar Soria Dustmann, Klaus Wehrle

https://comsys.rwth-aachen.de 1st KLEE Workshop, London, UK, 2018-04-20

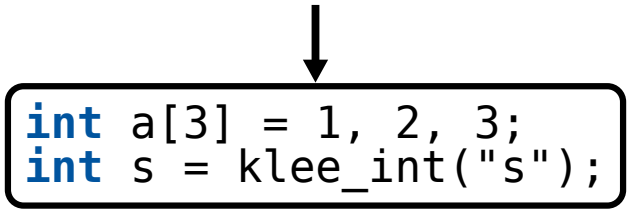
COM SYS | RWTH AACHEN UNIVERSITY

Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```

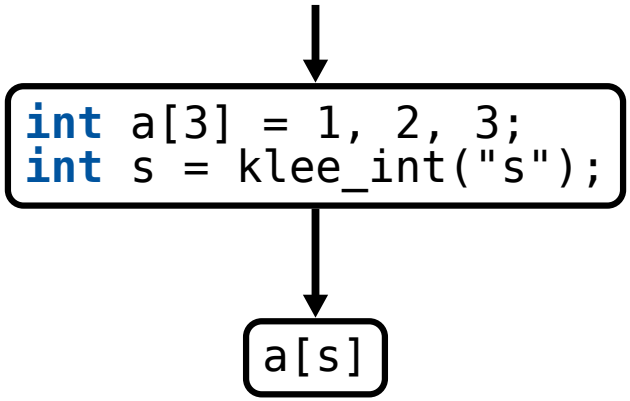
Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```



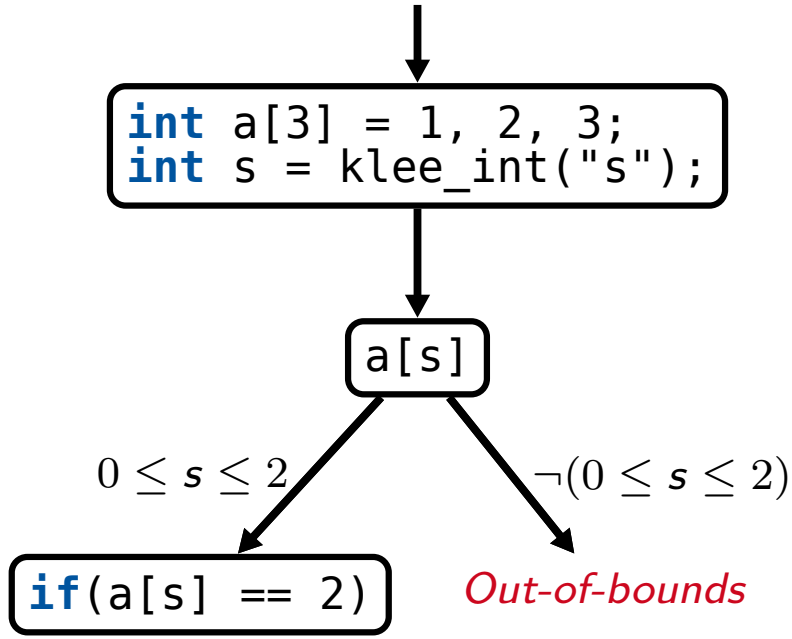
Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```



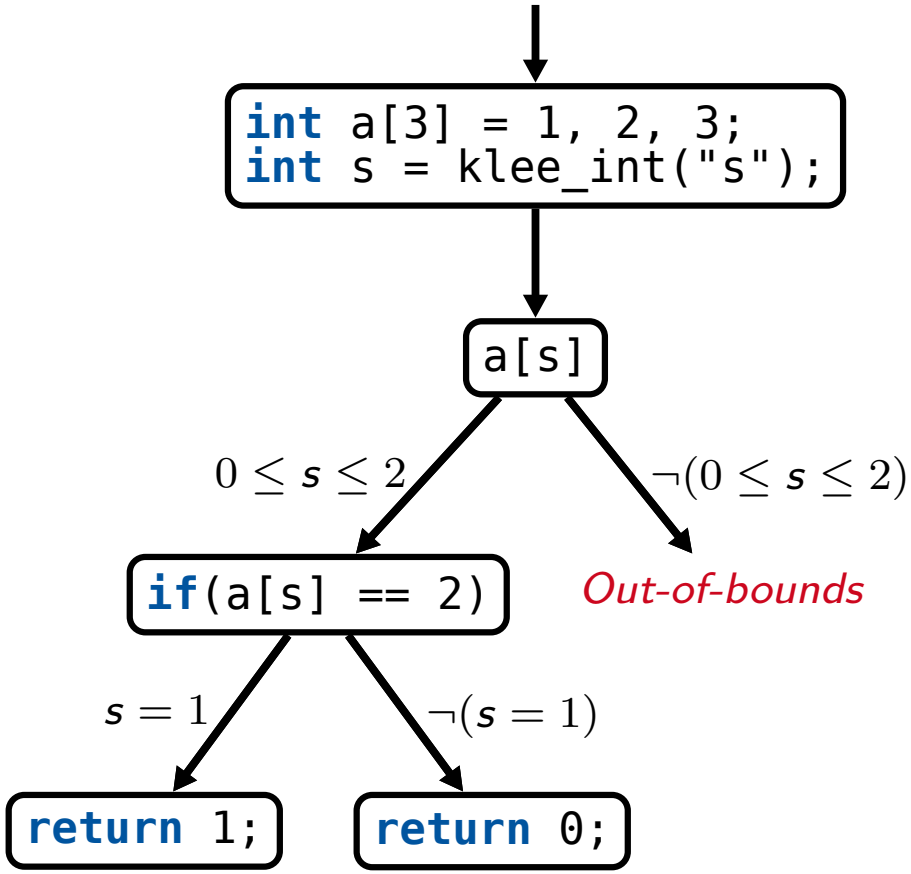
Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```



Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```

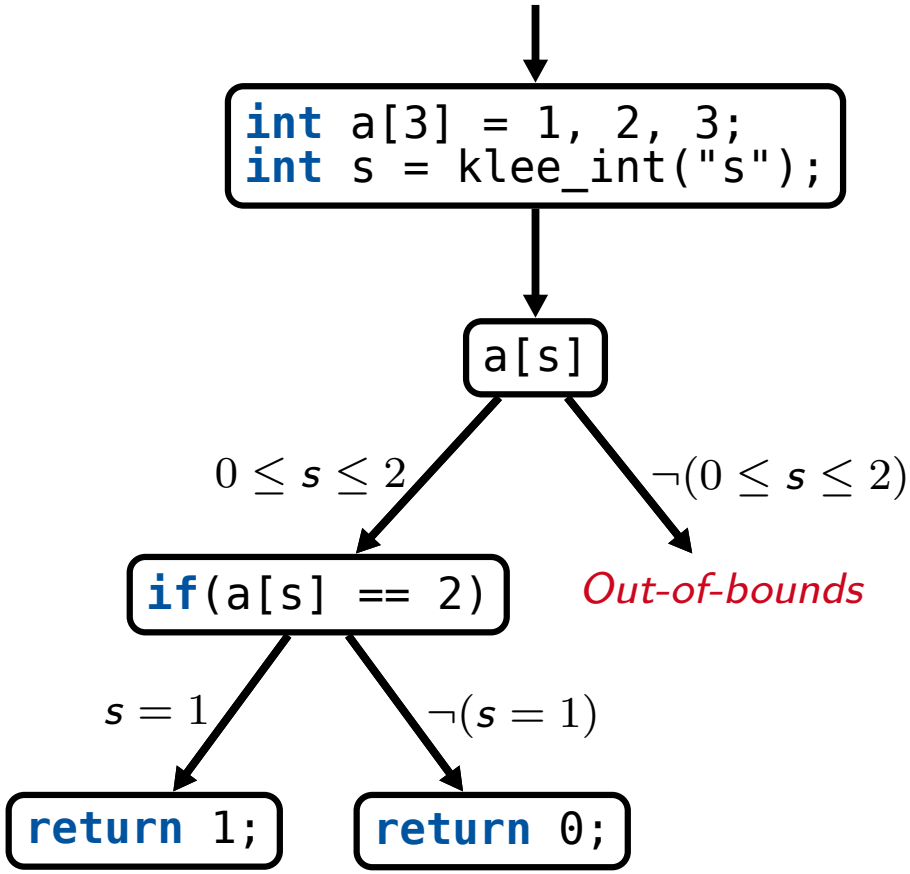


Motivation

```
1 int main(void) {
2   int a[3] = {1, 2, 3};
3   int s = klee_int("s");
4
5   if(a[s] == 2)
6     return 1;
7   else
8     return 0;
9 }
```

Expected: 3 Test Cases

Actual: *17* Test Cases



Resolving a[s]

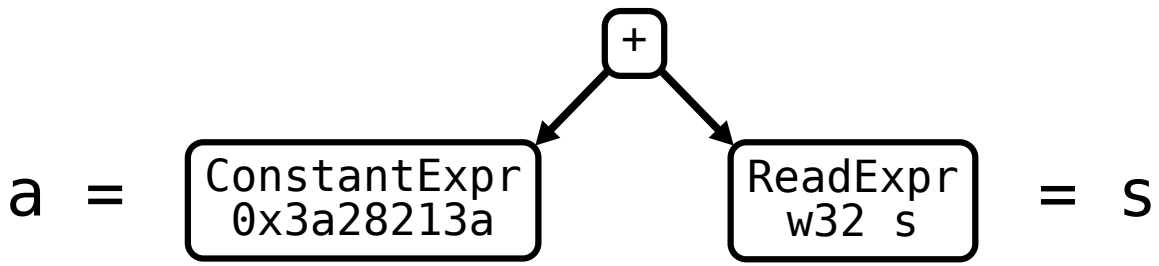
a[s]

Resolving a[s]

```
a[s] = *(a+s)
```

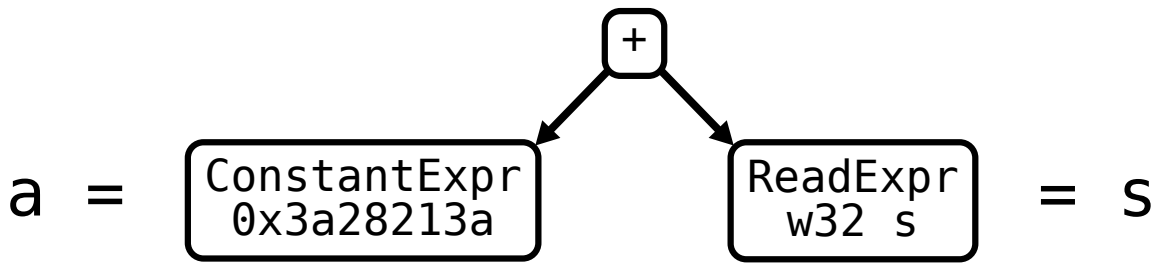
Resolving a[s]

```
a[s] = *(a+s)
```



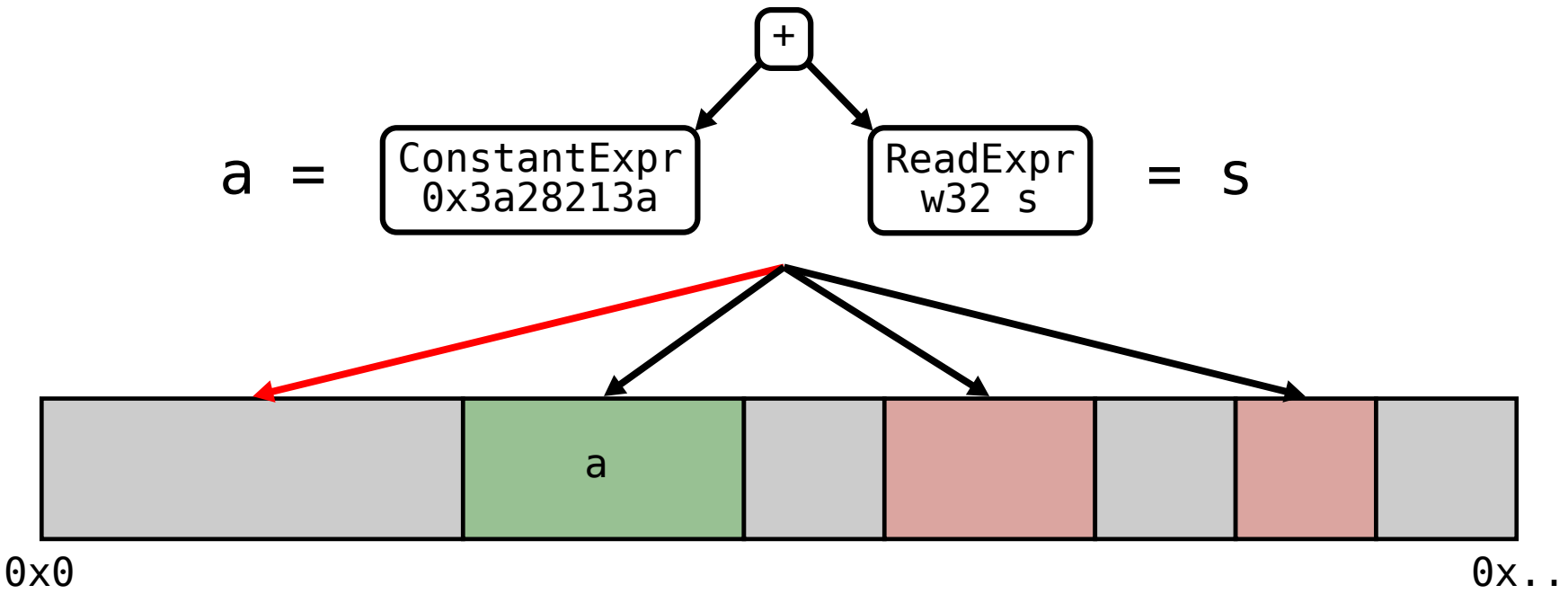
Resolving a[s]

```
a[s] = *(a+s)
```



Resolving a[s]

a[s] = *(a+s)



Pointer Tracking

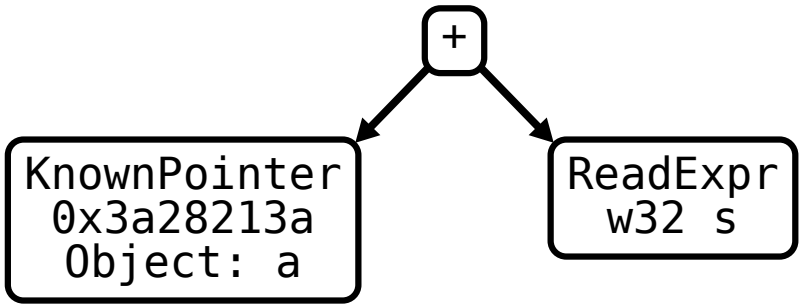
Idea:

- Mark the return value of an allocation as a pointer: `KnownPointer`
- Also remember which object is pointed to
- Use this information for address resolution

Pointer Tracking

Idea:

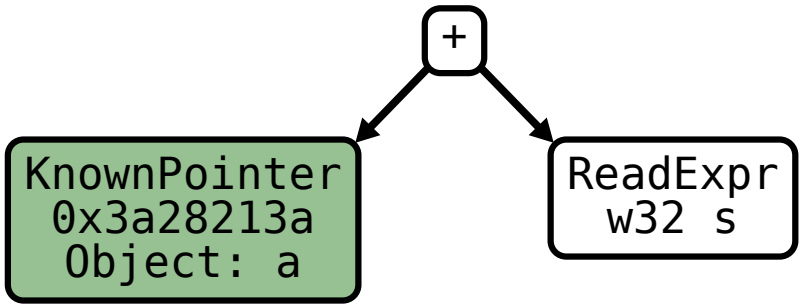
- Mark the return value of an allocation as a pointer: `KnownPointer`
- Also remember which object is pointed to
- Use this information for address resolution



Pointer Tracking

Idea:

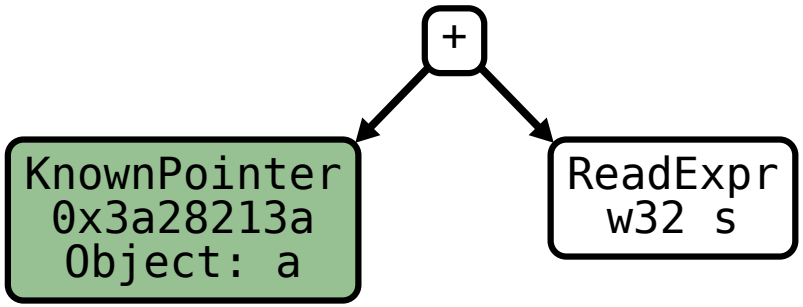
- Mark the return value of an allocation as a pointer: `KnownPointer`
- Also remember which object is pointed to
- Use this information for address resolution



Pointer Tracking

Idea:

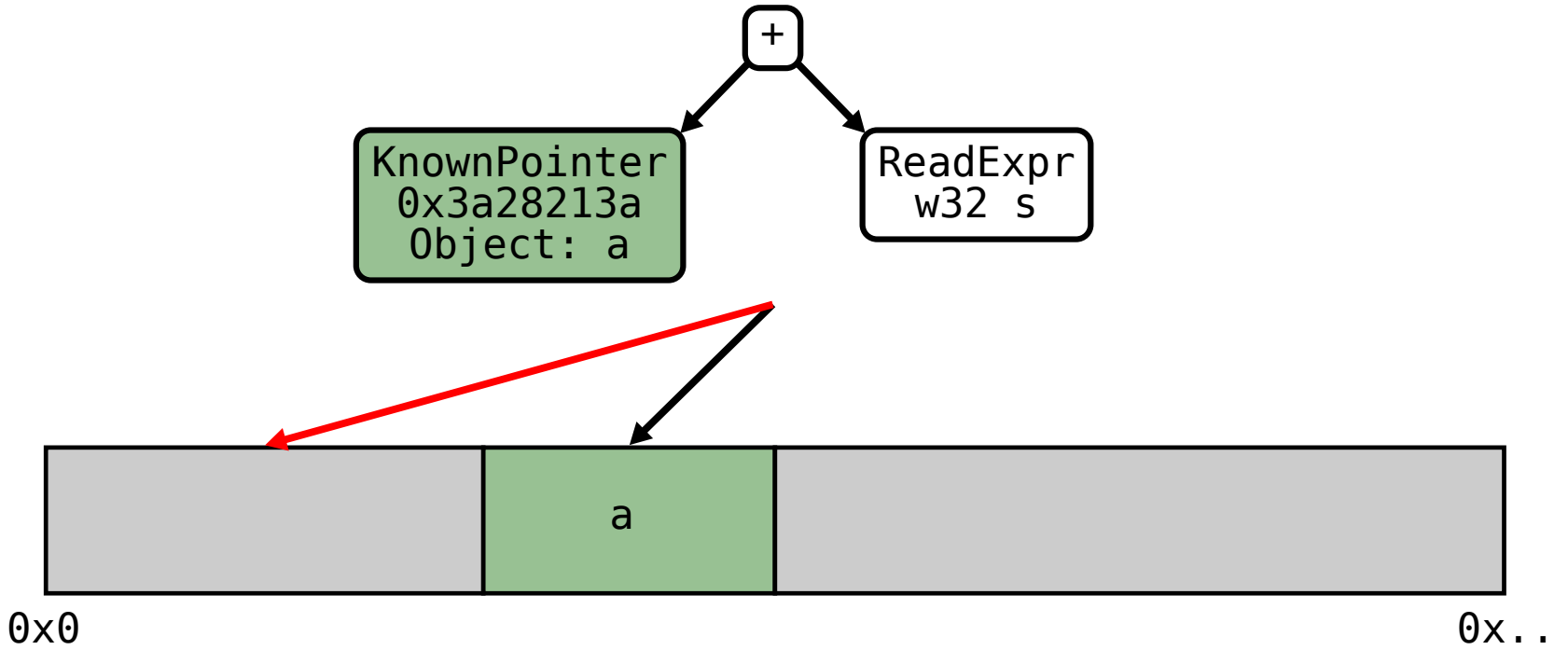
- Mark the return value of an allocation as a pointer: `KnownPointer`
- Also remember which object is pointed to
- Use this information for address resolution



Pointer Tracking

Idea:

- Mark the return value of an allocation as a pointer: `KnownPointer`
- Also remember which object is pointed to
- Use this information for address resolution



What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130)` + `KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`
- Undefined behaviour according to C standard

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`
- Undefined behaviour according to C standard
- x86_64 does not care anymore

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

- GCC 8 will be adding similar analyses

What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
- Result: `KnownPointer(a, 0x3a282135)`

Pointer subtractions of different objects

- `KnownPointer(a, 0x..) - KnownPointer(b, 0x..)`
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

- GCC 8 will be adding similar analyses
- `-fsanitize=pointer-subtract, -fsanitize=pointer-compare`

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
- Result: `KnownValue(5)`

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
- Result: `KnownValue(5)`

BUT

- `KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)`?

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
- Result: `KnownValue(5)`

BUT

- `KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)`?
- `(&,|,~,<<,>>,%,*,/)`?

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
- Result: `KnownValue(5)`

BUT

- `KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)`?
- `(&,|,~,<<,>>,%,*,/)`?
- Actually appear in real code (e.g., xor-pointer swap)

Offsets and Bitops

But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
- Result: `KnownValue(5)`

BUT

- `KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)`?
- `(&,|,~,<<,>>,%,*,/)`?
- Actually appear in real code (e.g., xor-pointer swap)
- `KnownPointer` and `KnownValue` not expressive enough

MaybePointer

Insight

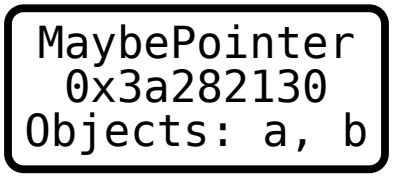
- We can't always know if the result of an operation will be a pointer.
- Might only become obvious after subsequent operations.

MaybePointer

Insight

- We can't always know if the result of an operation will be a pointer.
- Might only become obvious after subsequent operations.

Solution:



MaybePointer

Insight

- We can't always know if the result of an operation will be a pointer.
- Might only become obvious after subsequent operations.

Solution:

```
MaybePointer
0x3a282130
Objects: a, b
```

- Remember information about all involved objects
- No clear association with one specific object

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed

Pointer Tracking: External Function Calls

External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed
- → `ConstantExpr`

Pointer Tracking: Summary

`KnownPointer(Object)`

A value that is known to point to `Object`.

Pointer Tracking: Summary

`KnownPointer(Object)`

A value that is known to point to `Object`.

`MaybePointer(ObjectA, ObjectB, ...)`

A value that was somehow created through operations on pointers into `ObjectA` and `ObjectB`, and might currently point into any of them or not.

Pointer Tracking: Summary

`KnownPointer(Object)`

A value that is known to point to `Object`.

`MaybePointer(ObjectA, ObjectB, ...)`

A value that was somehow created through operations on pointers into `ObjectA` and `ObjectB`, and might currently point into any of them or not.

`KnownValue`

A value that is known to **not** be a pointer.

Pointer Tracking: Summary

`KnownPointer(Object)`

A value that is known to point to `Object`.

`MaybePointer(ObjectA, ObjectB, ...)`

A value that was somehow created through operations on pointers into `ObjectA` and `ObjectB`, and might currently point into any of them or not.

`KnownValue`

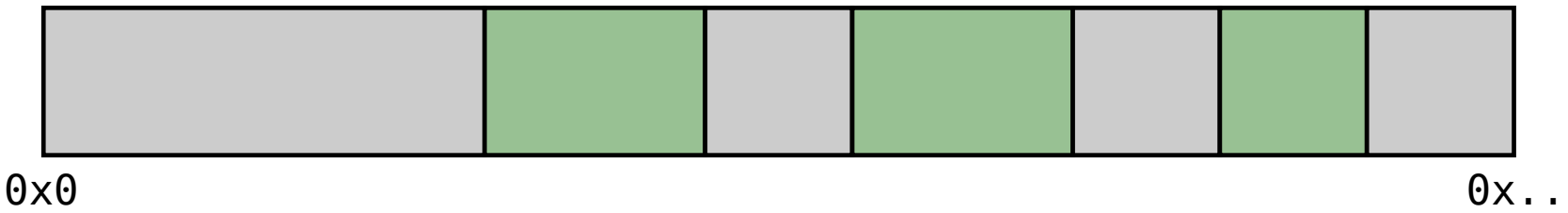
A value that is known to **not** be a pointer.

`ConstantExpr`

A value that might be a pointer into any object, or might not be a pointer at all. Results from external function calls.

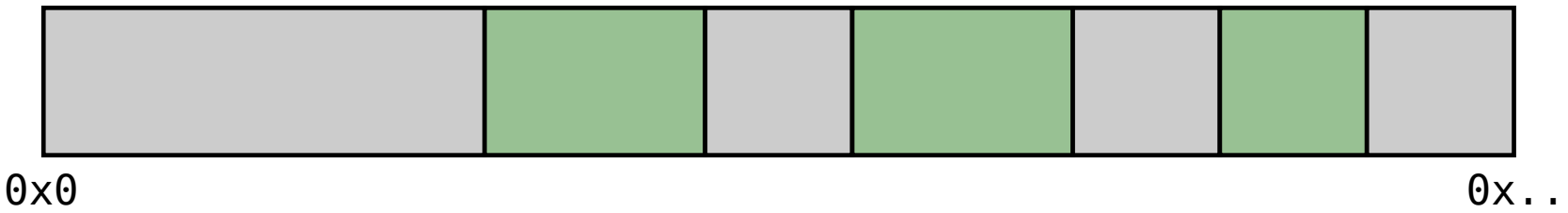
Old Memory Model

Before: Single address space, containing all objects



Old Memory Model

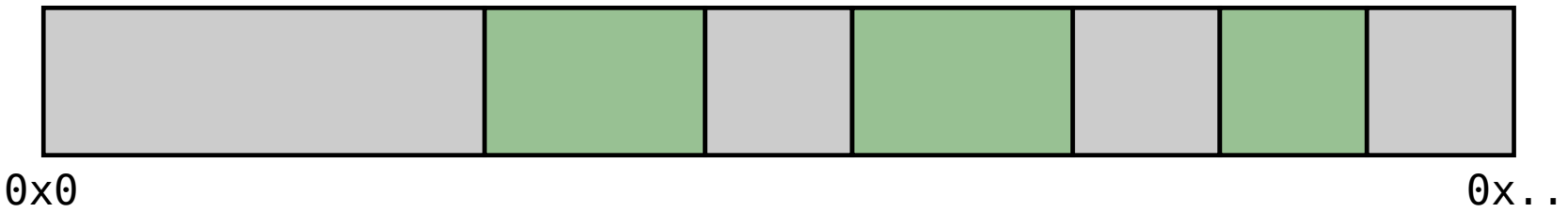
Before: Single address space, containing all objects



- Address resolution based solely on values

Old Memory Model

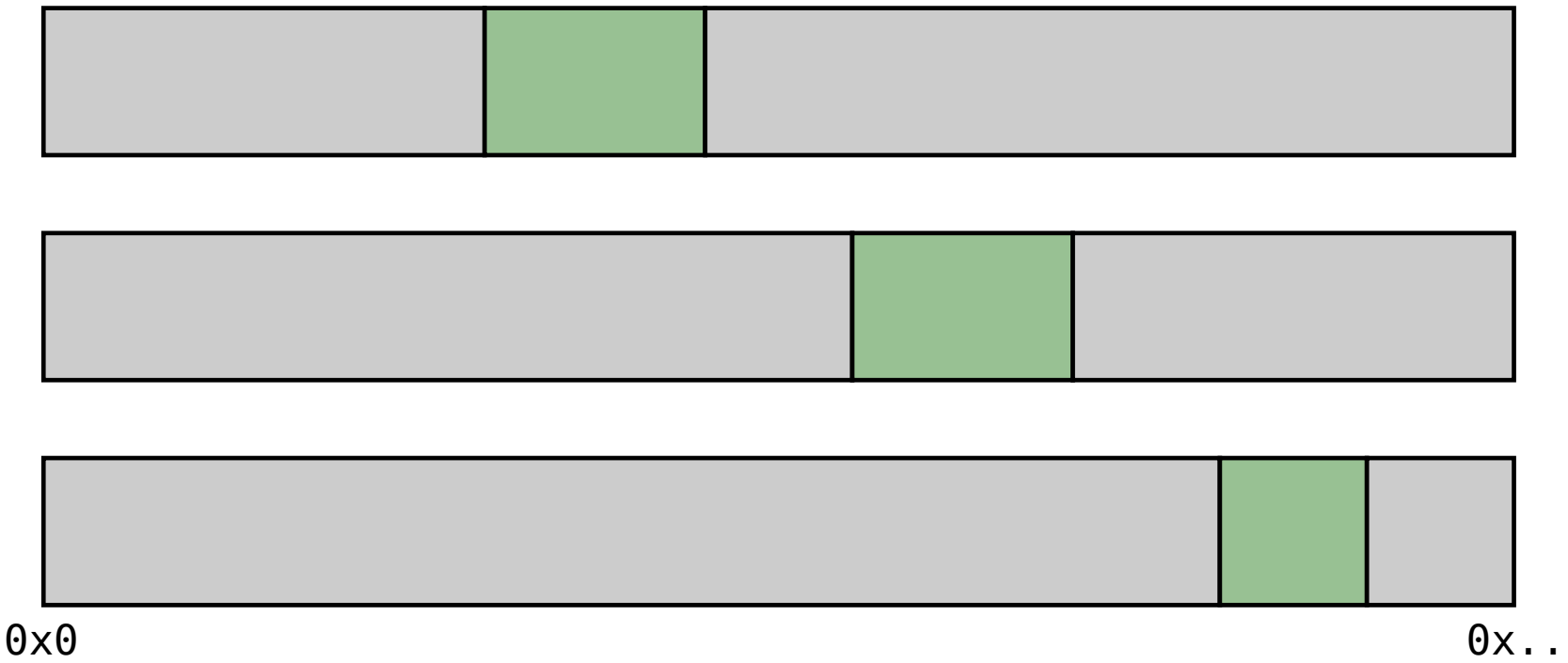
Before: Single address space, containing all objects



- Address resolution based solely on values
- No idea which values are pointers

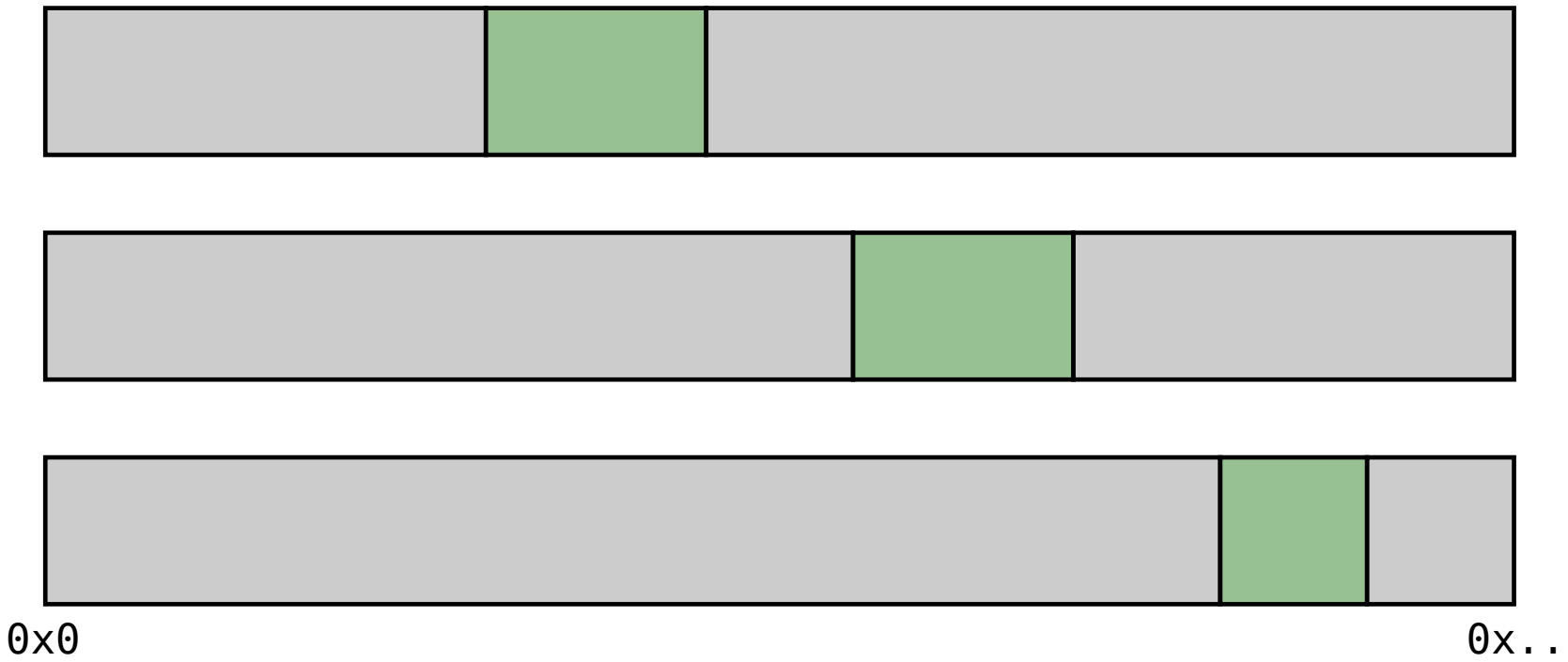
New Memory Model

Now: One full-size address space per object



New Memory Model

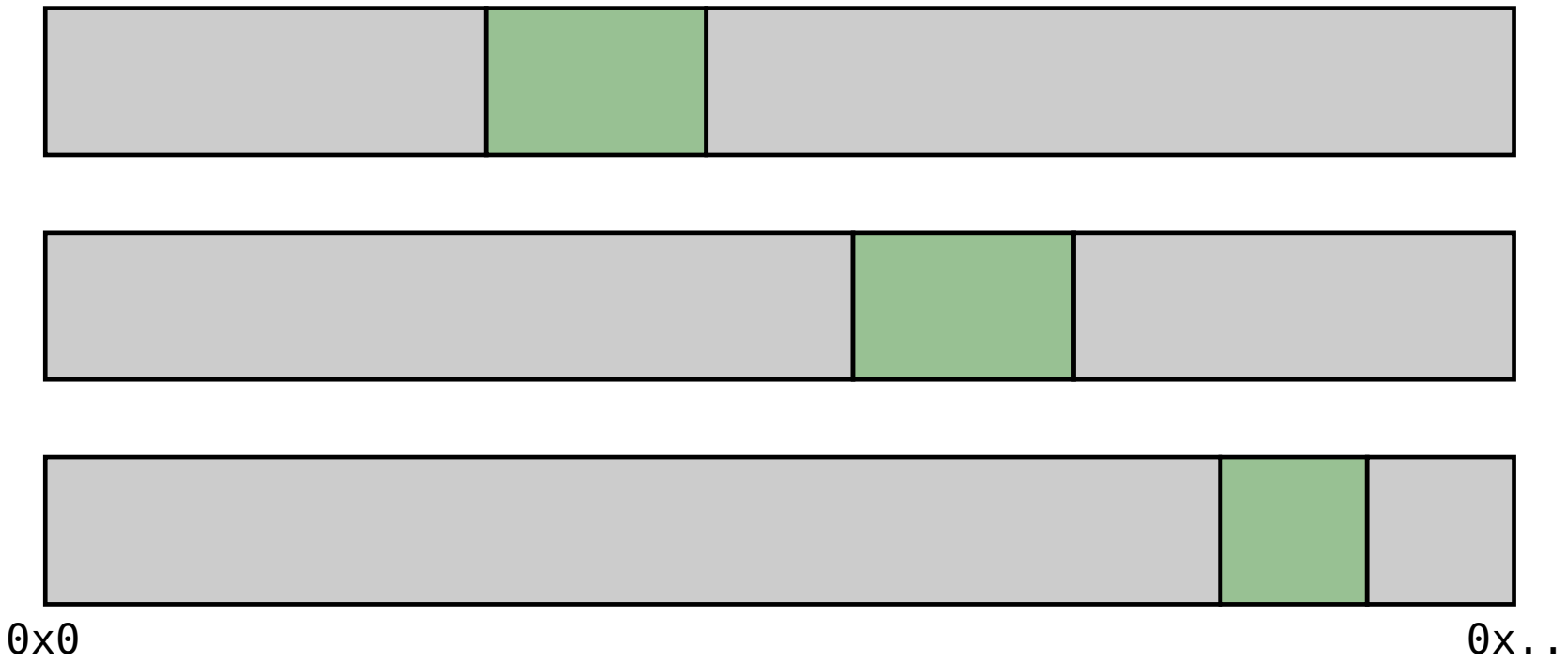
Now: One full-size address space per object



- Address resolution based on tracked pointers

New Memory Model

Now: One full-size address space per object



- Address resolution based on tracked pointers
- Much better knowledge about pointers and pointees

Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

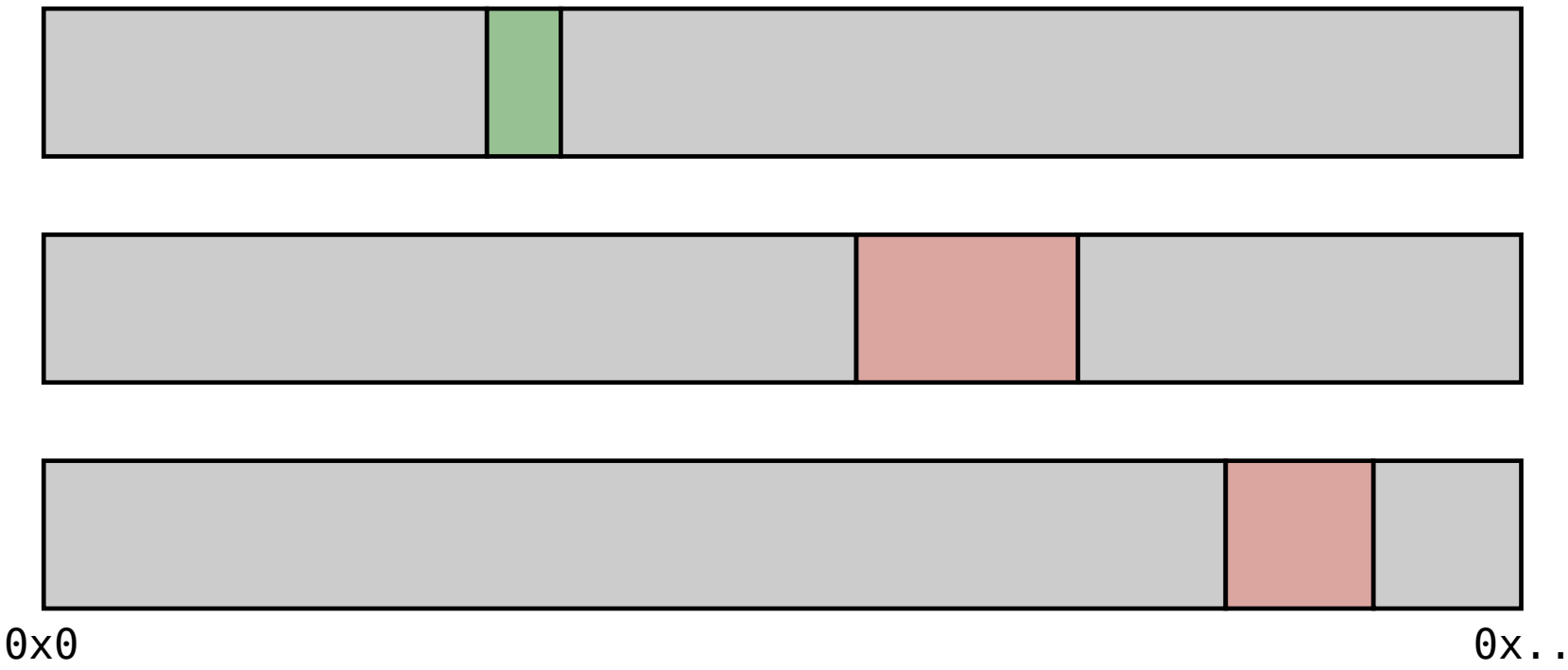
Possible solution: Lazy growth

Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

Possible solution: Lazy growth

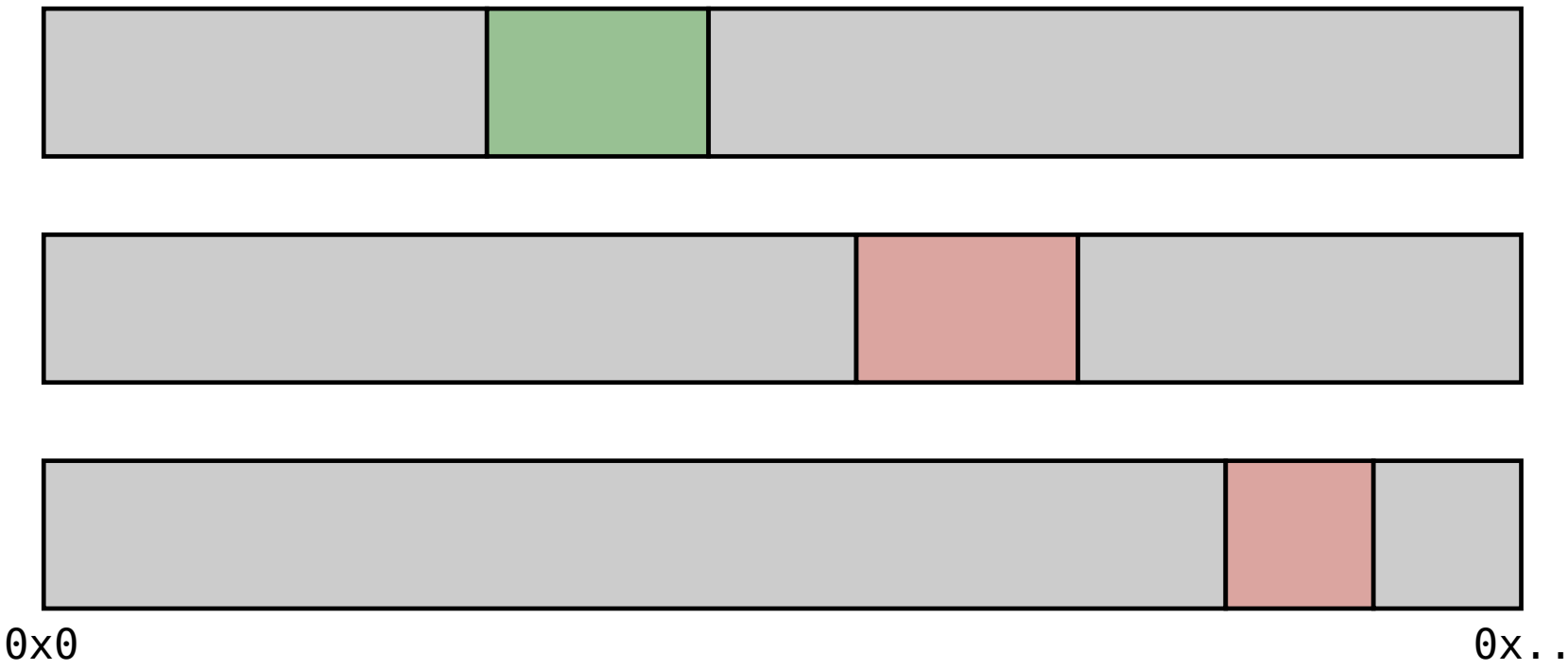


Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

Possible solution: Lazy growth

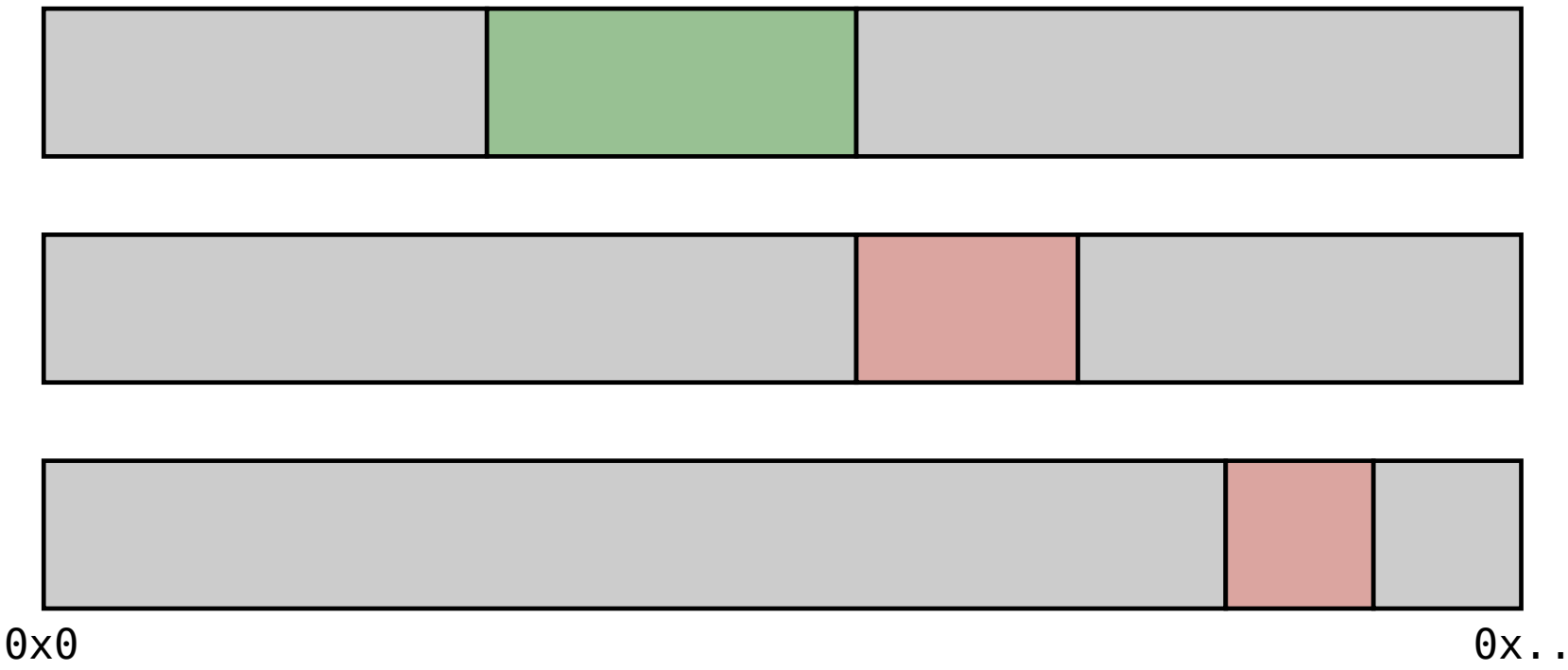


Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

Possible solution: Lazy growth

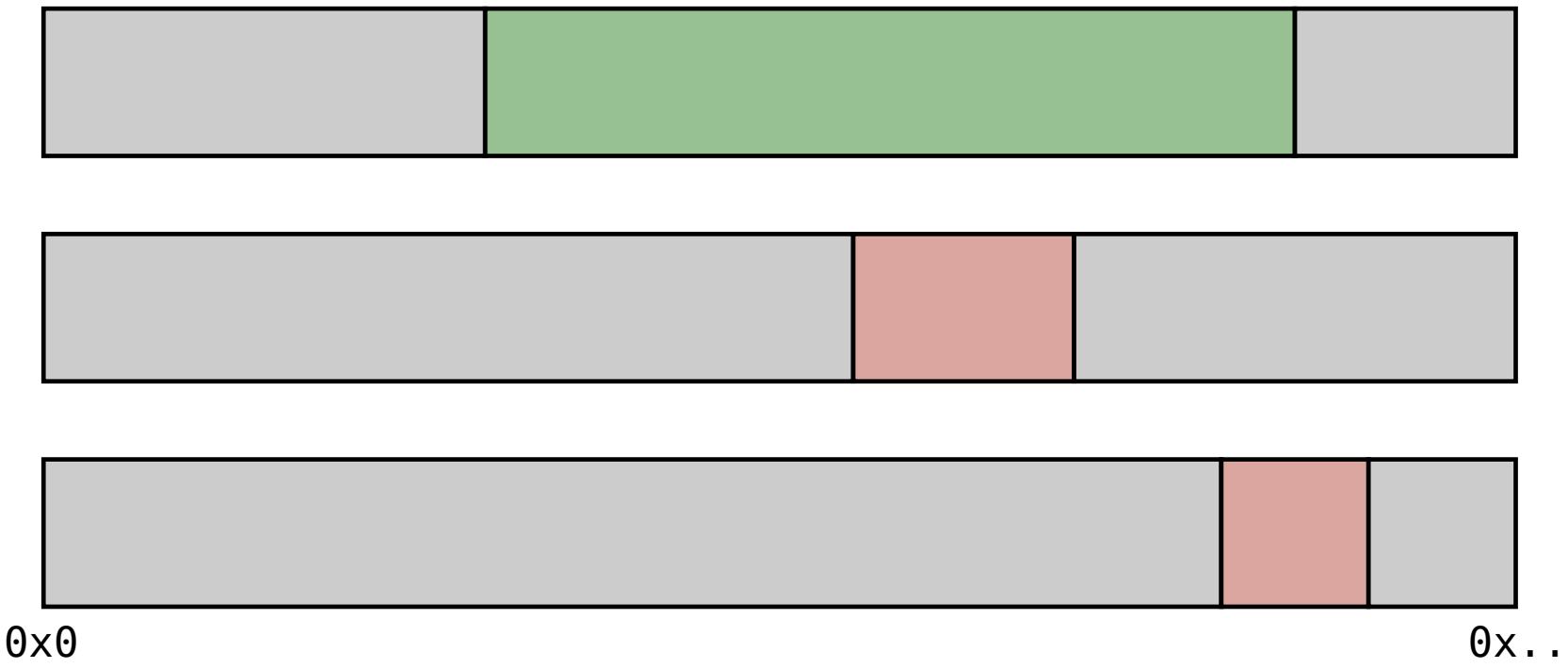


Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → `Error: concretized symbolic size`

Possible solution: Lazy growth

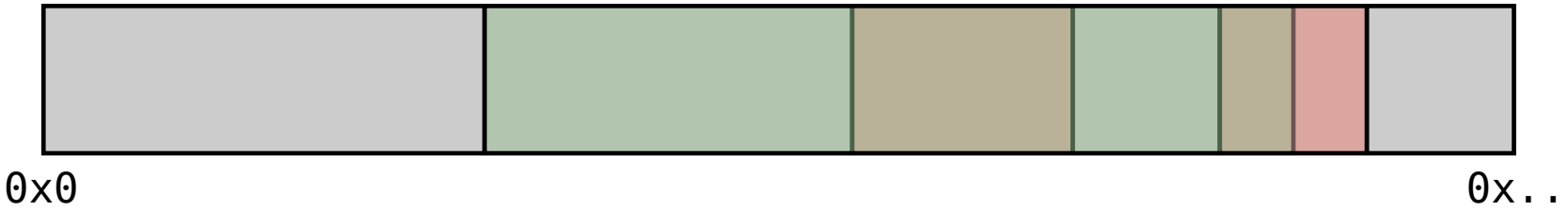


External Function Calls revisited

EFCs require us to flatten our address space:

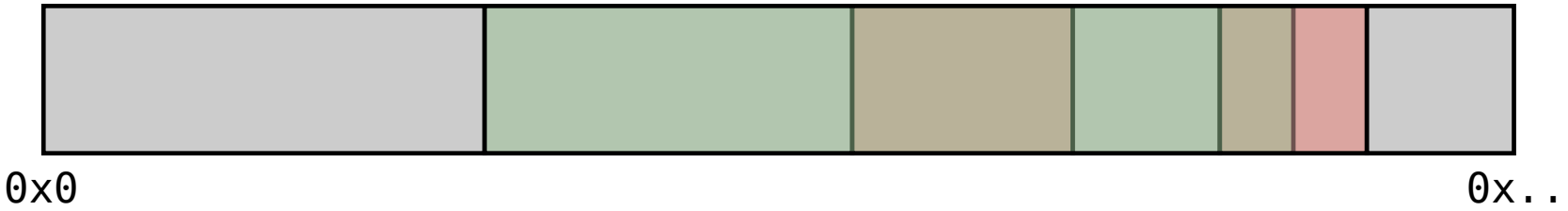
External Function Calls revisited

EFCs require us to flatten our address space:



External Function Calls revisited

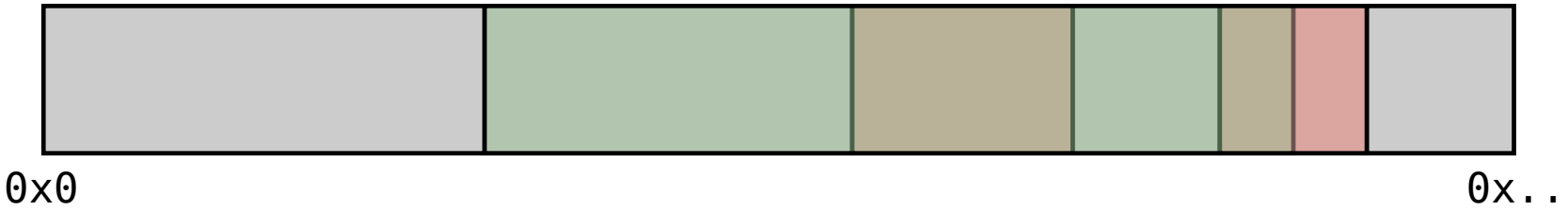
EFCs require us to flatten our address space:



- "Problematic"

External Function Calls revisited

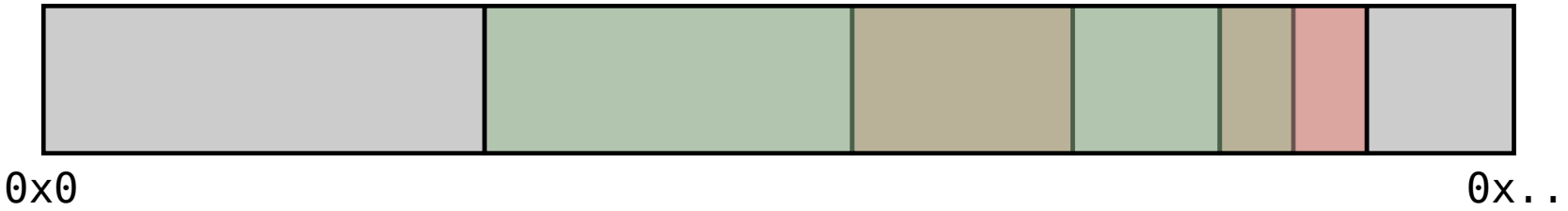
EFCs require us to flatten our address space:



- "Problematic"
- Quick fix: Re-allocate to a new position

External Function Calls revisited

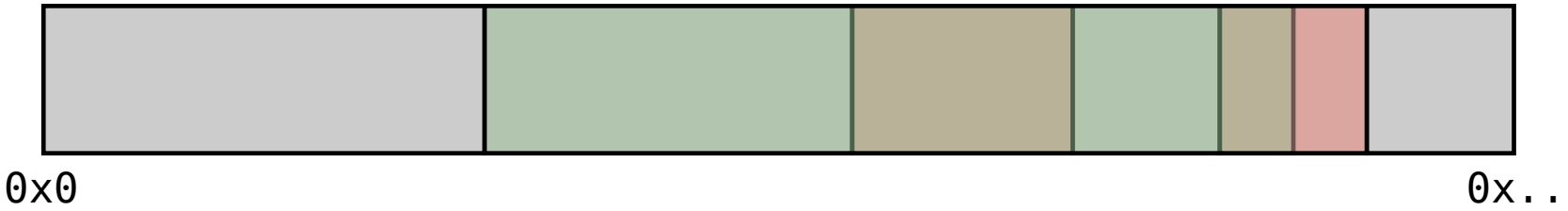
EFCs require us to flatten our address space:



- "Problematic"
- Quick fix: Re-allocate to a new position
- Does not work either ("changing the past")

External Function Calls revisited

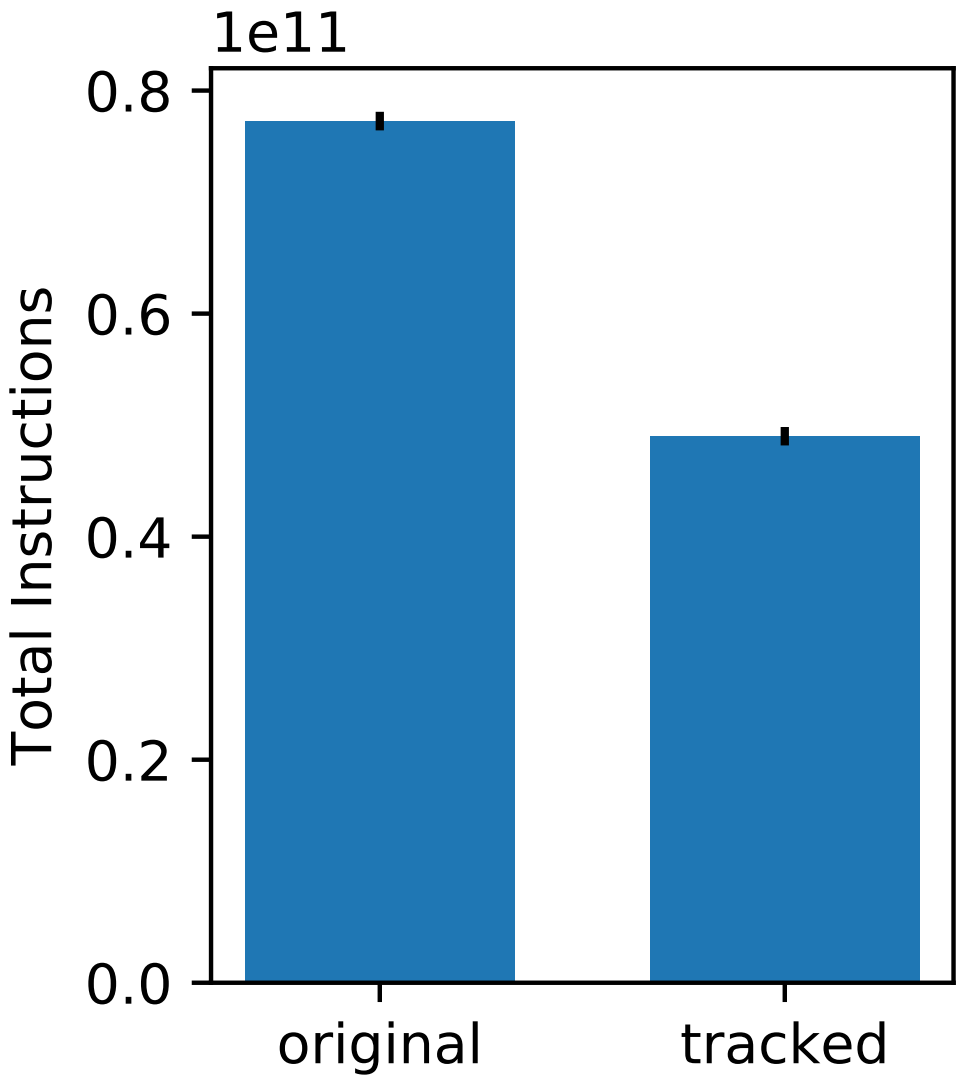
EFCs require us to flatten our address space:



- "Problematic"
- Quick fix: Re-allocate to a new position
- Does not **easily** work either ("changing the past")

First Experiments

- 25 iterations, one hour each
- Able to run all 107 coreutils
- 80 without `ConstantExprs`
- No `MaybePointers` with more than four objects
- Four subtractions found, all due to one `realloc` in uclibc
- Somewhat high overhead (unoptimized)



Regex_old

```
1 UCHAR_T* old_buffer = cur_buffer;
2 cur_buffer = realloc(cur_buffer, new_size);
3 // ...
4 if(cur_buffer != old_buffer) {
5   int offset = cur_buffer - old_buffer;
6   FIXUP_POINTER(foo, offset);
7   FIXUP_POINTER(bar, offset);
8 // ...
```

Summary

KLEE's memory model:

- Single address space
- No knowledge about pointers

Summary

KLEE's memory model:

- Single address space
- No knowledge about pointers

Pointer tracking:

- Keep track of pointers
- Some operations and external calls require overapproximation
- `KnownPointer, MaybePointer, KnownValue, ConstantExpr`
- Basic symbolic allocation sizes

Summary

KLEE's memory model:

- Single address space
- No knowledge about pointers

Pointer tracking:

- Keep track of pointers
- Some operations and external calls require overapproximation
- `KnownPointer, MaybePointer, KnownValue, ConstantExpr`
- Basic symbolic allocation sizes

Evaluation:

- Works on the coreutils
- Found one case of an illegal pointer subtraction