Address-Aware Query Caching for Symbolic Execution

David Trabish

Shachar Itzhaky

Noam Rinetzky

Tel-Aviv University, Israel

Technion, Israel

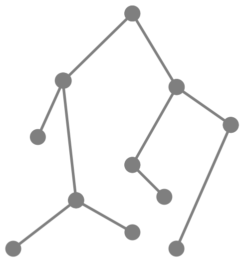
ICST 2021

Symbolic Execution: Introduction

- Systematic program analysis technique
- Many applications:
  - Test generation
  - Bug finding
  - ...
- Active research area
- Used in industry

Main Challenges

Path Explosion



Constraint Solving

$$x = 1 \wedge z > 1 \wedge select(a_2, 7) = 1$$

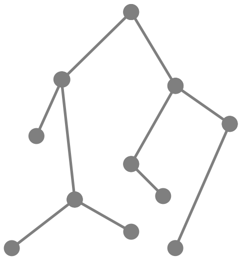
$$y > 10 \wedge z > 1 \wedge z + y < 77$$

$$a > b + 23 \wedge c - a > 56$$

$$w > s * 6 \wedge t > w$$

Main Challenges

Path Explosion



Constraint Solving

$$x = 1 \wedge z > 1 \wedge select(a_2, 7) = 1$$

$$y > 10 \wedge z > 1 \wedge z + y < 77$$

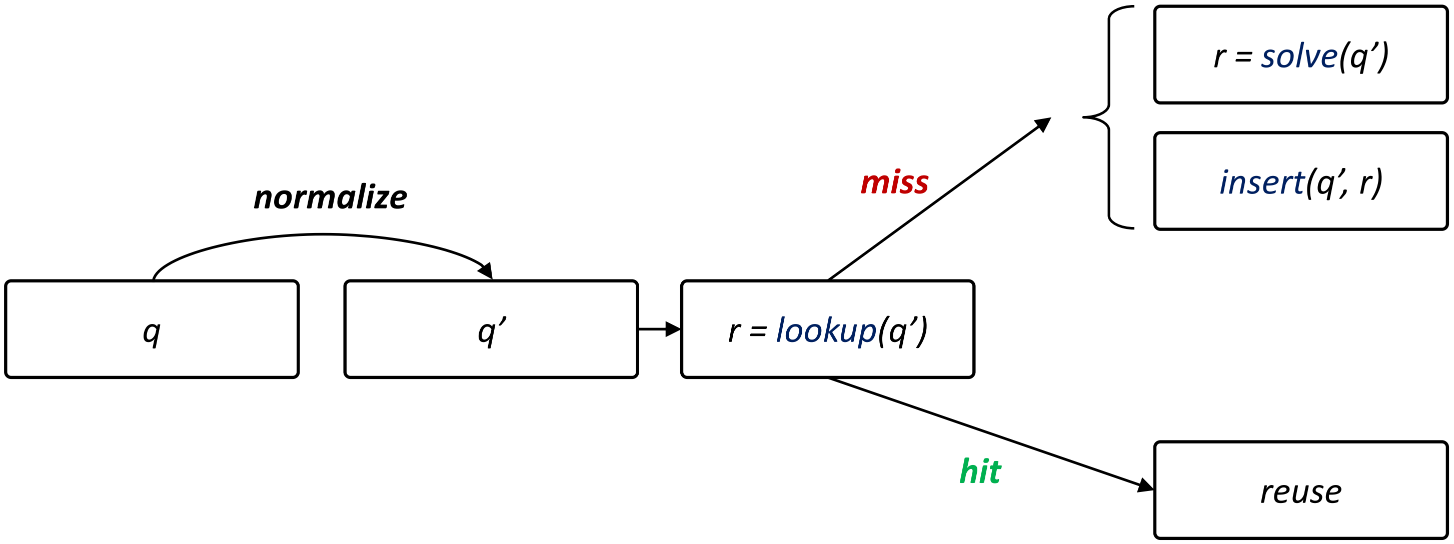
$$a > b + 23 \wedge c - a > 56$$

$$w > s * 6 \wedge t > w$$

Query Caching

- Constraint solving is a main bottleneck
- A common mitigation used by many tools – caching queries!

Query Caching



Query Caching

$z > 7 \land x - 2 > y$

Query	Result
…	…
…	…
…	…

Query Caching

$$\underbrace{z > 7 \land x - 2 > y}_{} \xrightarrow{\textit{normalize}} z > 7 \land x - y > 2$$

Query	Result
…	…
…	…
…	…

Query Caching

$$z > 7 \land x - 2 > y \xrightarrow{\textit{normalize}} z > 7 \land x - y > 2$$

Query	Result
…	…
…	…
…	…

miss

Query Caching

$$z > 7 \wedge x - 2 > y \xrightarrow{\textbf{\textit{normalize}}} z > 7 \wedge x - y > 2$$

Query	Result
…	…
…	…
…	…
$z > 7 \wedge x - y > 2$	*SAT*

Query Caching

normalize

$z > 7 \wedge x - 2 > y \xrightarrow{\textit{normalize}} z > 7 \wedge x - y > 2$

$z > 7 \wedge y + 3 \leq x$

Query	Result
…	…
…	…
…	…
$z > 7 \wedge x - y > 2$	*SAT*

Query Caching

$$z > 7 \land x - 2 > y \xrightarrow{\textit{normalize}} z > 7 \land x - y > 2$$

$$z > 7 \land y + 3 \leq x \xrightarrow{\textit{normalize}} z > 7 \land x - y > 2$$

Query	Result
…	…
…	…
…	…
$z > 7 \land x - y > 2$	*SAT*

Query Caching

$$z > 7 \wedge x - 2 > y \xrightarrow{\textit{normalize}} z > 7 \wedge x - y > 2$$

$$z > 7 \wedge y + 3 \leq x \xrightarrow{\textit{normalize}} z > 7 \wedge x - y > 2$$

Query	Result
…	…
…	…
…	…
$z > 7 \wedge x - y > 2$	*SAT*

hit

Query Caching

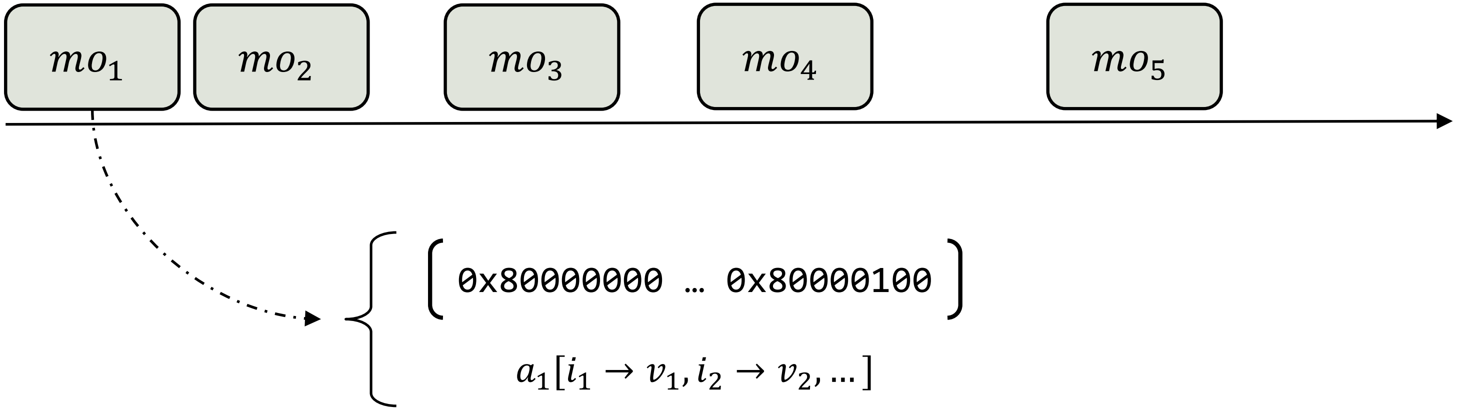
- Incomplete solution
- There are queries which are:
    - Equivalent
    - Can’t be reduced to the same normal form

Address-Dependent Queries

- Queries that depend on **numerical address values**
- First, some background…

Background: Address Space

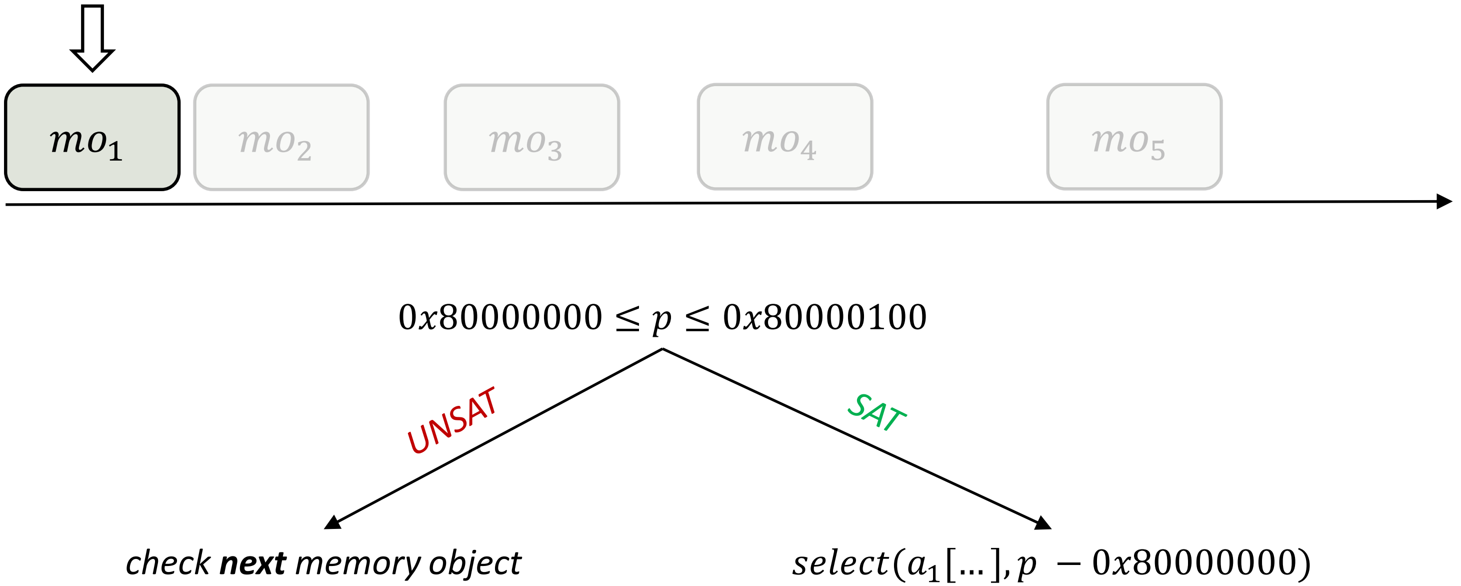
- Total order of memory objects
- Each memory object is associated with:
  - Unique address interval
  - Unique SMT array



Background: Pointer Resolution

When dereferencing a pointer expression ***p***:

- Scan the address space



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} true$

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0$

Address-Dependent Queries

```c
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

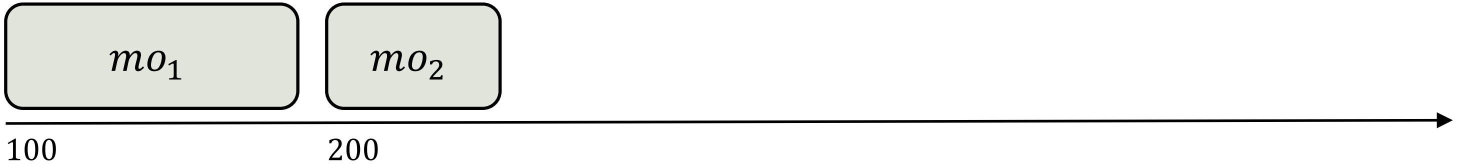
$pc \stackrel{\text{def}}{=} z \leq 0$

Address-Dependent Queries

```c
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0$

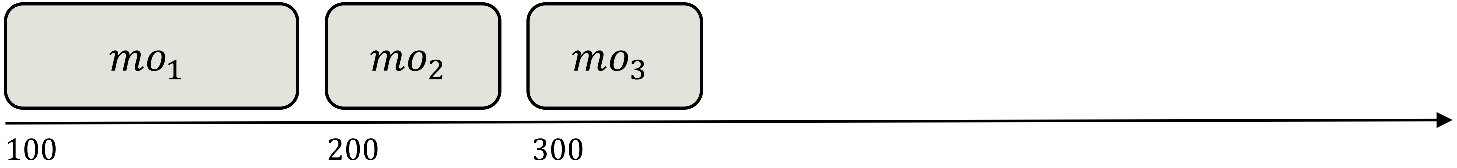


Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0$

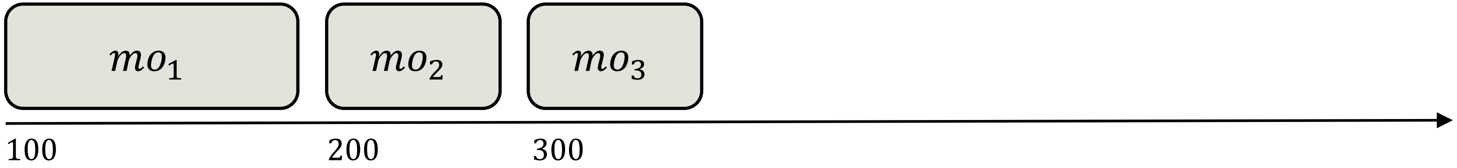


Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \land i < 2 \land j < 2$



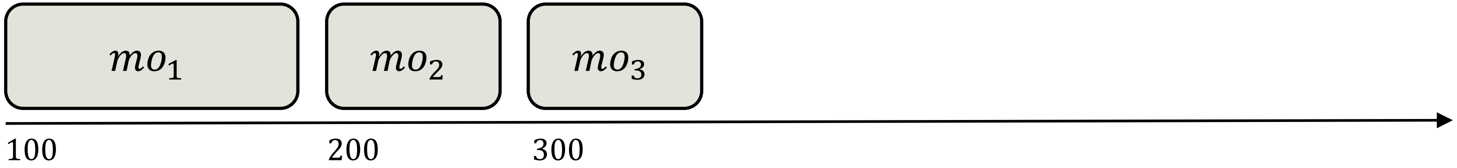
Address-Dependent Queries

```c
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i)$

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \land i < 2 \land j < 2$



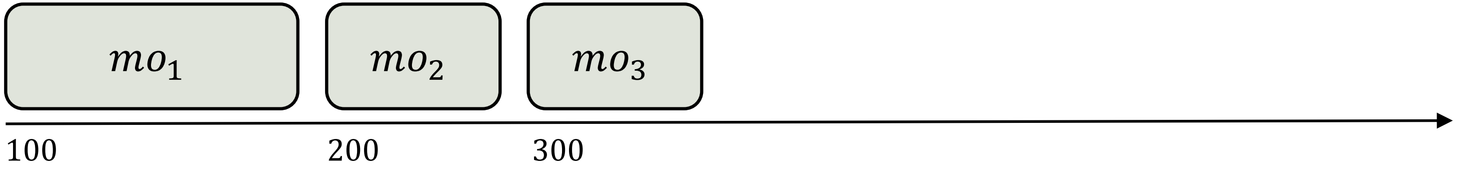
Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$$select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$$

path constraints:

$$pc \stackrel{\text{def}}{=} z \leq 0 \land i < 2 \land j < 2$$



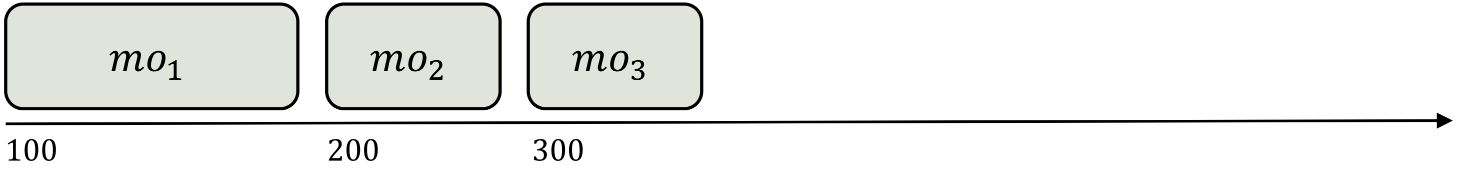
Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$$

path constraints:

$$pc \stackrel{\text{def}}{=} z \leq 0 \wedge i < 2 \wedge j < 2$$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

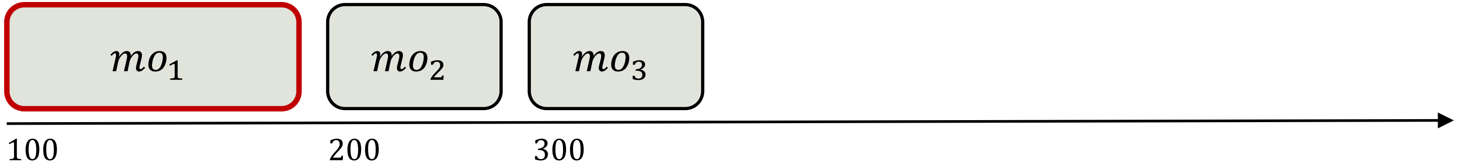
$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \ \wedge \ i < 2 \ \wedge \ j < 2$

resolution query:

$pc \wedge 100 \leq p < 116$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$p \overset{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$

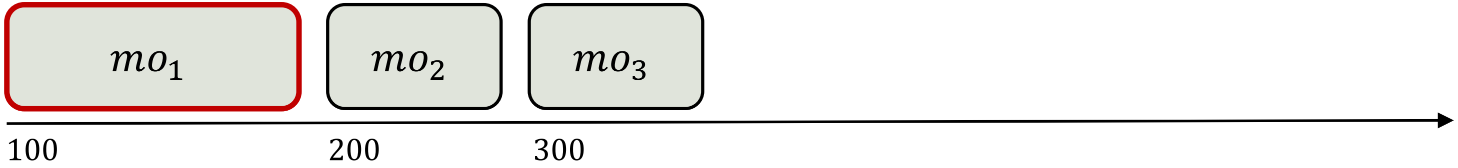
path constraints:

$pc \overset{\text{def}}{=} z \leq 0 \wedge i < 2 \wedge j < 2$

resolution query:

$pc \wedge 100 \leq p < 116$

UNSAT



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

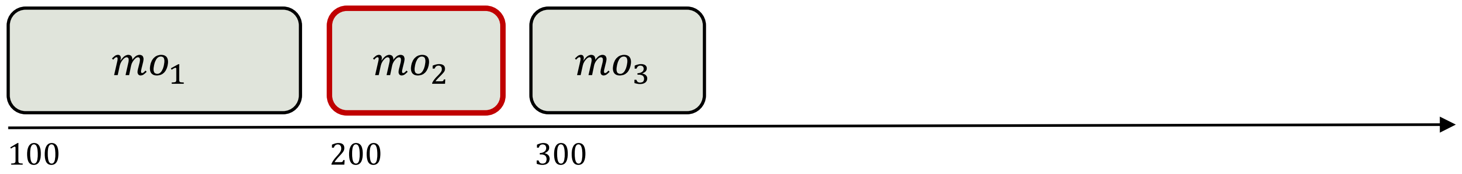
$$p \overset{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$$

path constraints:

$$pc \overset{\text{def}}{=} z \leq 0 \ \wedge \ i < 2 \ \wedge \ j < 2$$

resolution query:

$$pc \wedge 200 \leq p < 202$$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$$

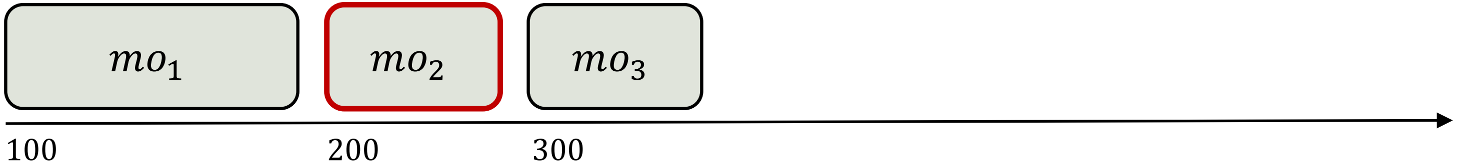
path constraints:

$$pc \stackrel{\text{def}}{=} z \leq 0 \ \land \ i < 2 \ \land \ j < 2$$

resolution query:

$$pc \land 200 \leq p < 202$$

SAT



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

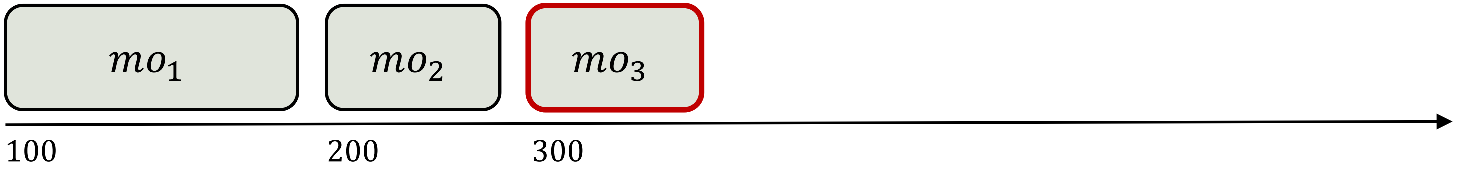
$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \ \land \ i < 2 \ \land \ j < 2$

resolution query:

$pc \land 300 \leq p < 302$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

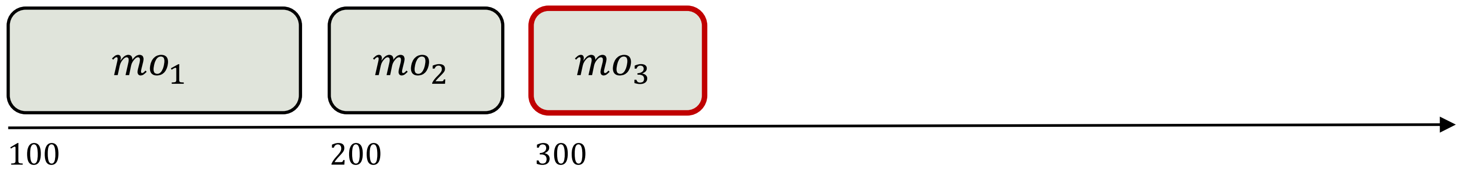
path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \land i < 2 \land j < 2$

resolution query:

$pc \land 300 \leq p < 302$

SAT



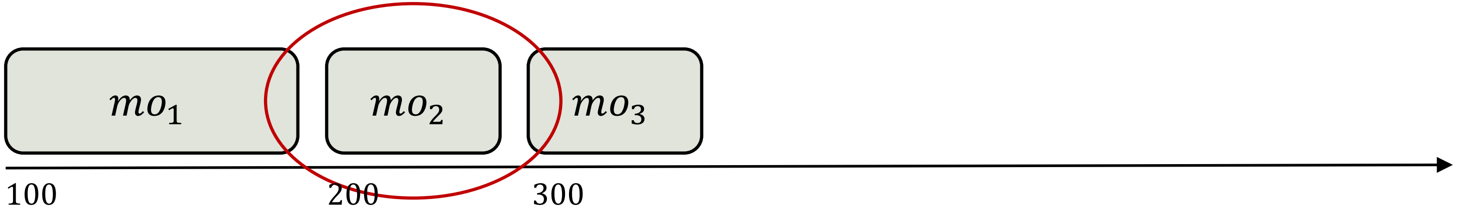
Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$$

path constraints:

$$pc \stackrel{\text{def}}{=} z \leq 0 \ \wedge \ i < 2 \ \wedge \ j < 2$$



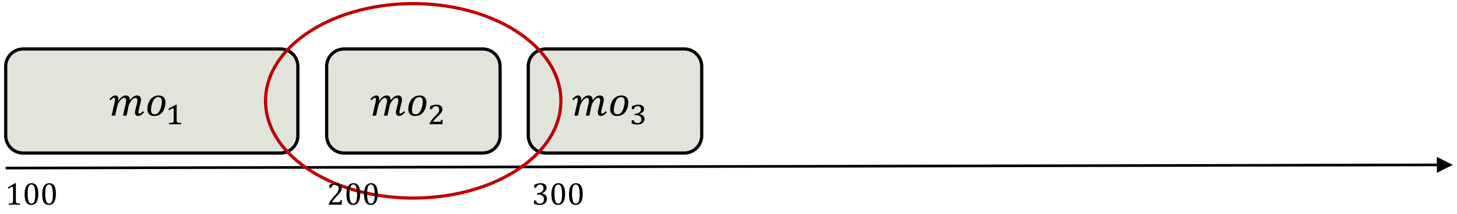
Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow 200{,}1 \rightarrow 300], i) + j$

path constraints:

$pc \stackrel{\text{def}}{=} z \leq 0 \;\wedge\; i < 2 \;\wedge\; j < 2 \;\wedge\; 200 \leq p \leq 202$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

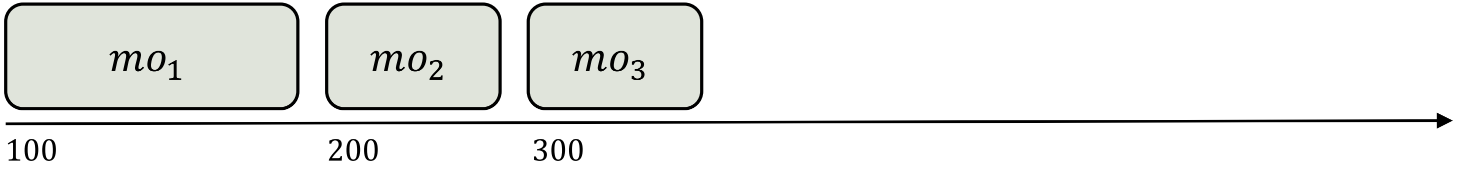
$$p \overset{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$$

path constraints:

$$pc \overset{\text{def}}{=} z \leq 0 \ \land \ i < 2 \ \land \ j < 2 \ \land \ 200 \leq p \leq 202$$

query:

$$pc \ \land \ select(a_2, p - 200) = 7$$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

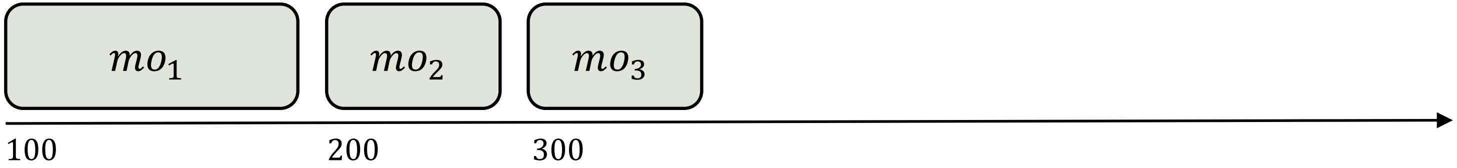
$$p \overset{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$$

path constraints:

$$pc \overset{\text{def}}{=} z \leq 0 \ \wedge \ i < 2 \ \wedge \ j < 2 \ \wedge \ 200 \leq p \leq 202$$

query:

$$pc \ \wedge \ select(a_2, p - 200) = 7$$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

What happens when $z > 0$*?*

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} true$

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z > 0$

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

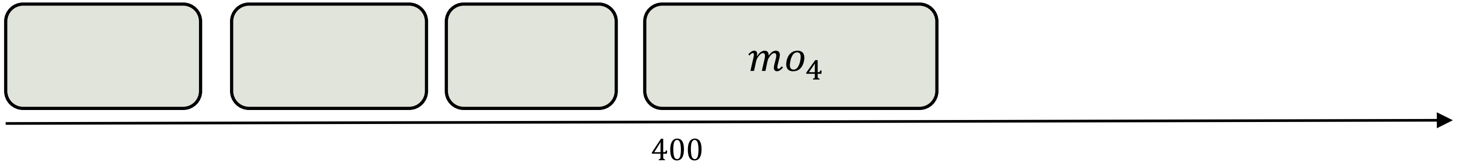
$pc \stackrel{\text{def}}{=} z > 0$

Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z > 0$

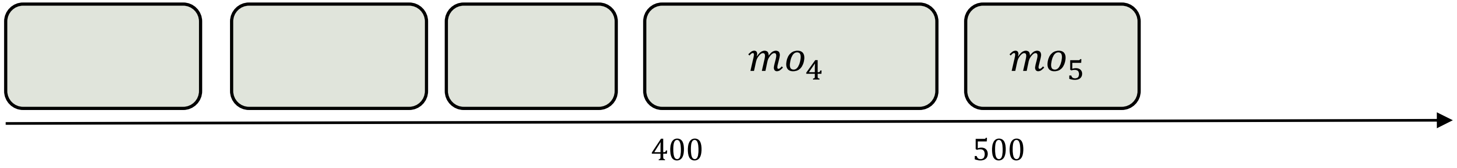


Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z > 0$

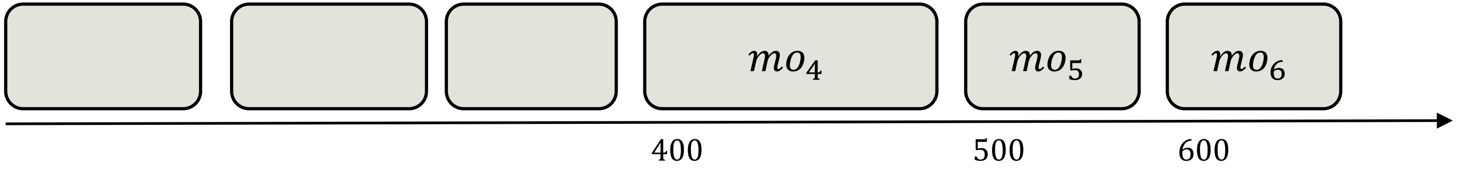


Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

path constraints:

$pc \stackrel{\text{def}}{=} z > 0$



Address-Dependent Queries

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

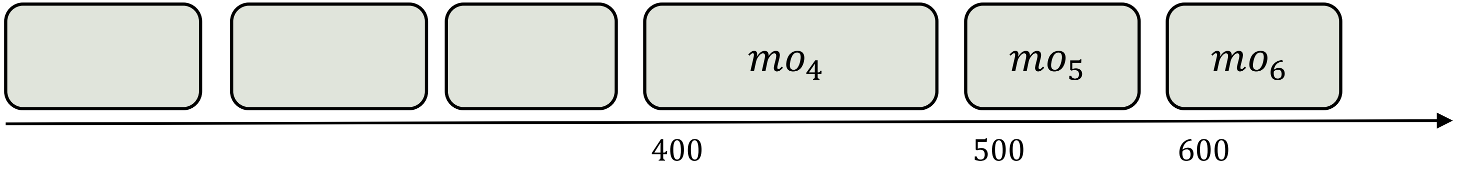
$$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow 500, 1 \rightarrow 600], i) + j$$

path constraints:

$$pc \stackrel{\text{def}}{=} z > 0 \ \wedge \ i < 2 \ \wedge \ j < 2 \ \wedge 500 \leq p \leq 502$$

query:

$$pc \ \wedge \ select(a_5, p - 500) = 7$$



Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

$pc \stackrel{\text{def}}{=} z \leq 0 \land\ i < 2\ \land\ j < 2\ \land 200 \leq p \leq 202$

query:

$pc\ \land\ select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to 500, 1 \to 600], i) + j$

$pc \stackrel{\text{def}}{=} z > 0\ \land\ i < 2\ \land\ j < 2\ \land 500 \leq p \leq 502$

query:

$pc\ \land\ select(a_5, p - 500) = 7$

Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

$pc \stackrel{\text{def}}{=} z \leq 0 \land i < 2 \land j < 2 \land 200 \leq p \leq 202$

query:

$pc \land select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to 500, 1 \to 600], i) + j$

$pc \stackrel{\text{def}}{=} z > 0 \land i < 2 \land j < 2 \land 500 \leq p \leq 502$

query:

$pc \land select(a_5, p - 500) = 7$

z can be sliced away!

Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 200 \leq p \leq 202$

query:

$pc \ \wedge \ select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow 500, 1 \rightarrow 600], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 500 \leq p \leq 502$

query:

$pc \ \wedge \ select(a_5, p - 500) = 7$

z can be sliced away!

Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land 200 \leq p \leq 202$

query:

$pc \ \land \ select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to 500, 1 \to 600], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land 500 \leq p \leq 502$

query:

$pc \ \land \ select(a_5, p - 500) = 7$

- Identical queries up to **address values**

Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \to 200, 1 \to 300], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 200 \leq p \leq 202$

query:

$pc \ \wedge \ select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to 500, 1 \to 600], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 500 \leq p \leq 502$

query:

$pc \ \wedge \ select(a_5, p - 500) = 7$

- Identical queries up to **address values**
- Equivalent

Address-Dependent Queries

$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow 200, 1 \rightarrow 300], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 200 \leq p \leq 202$

query:

$pc \ \wedge \ select(a_2, p - 200) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow 500, 1 \rightarrow 600], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge 500 \leq p \leq 502$

query:

$pc \ \wedge \ select(a_5, p - 500) = 7$

- Identical queries up to **address values**
- Equivalent
- No common normal form → **query caching fails!**

Goal

Apply query caching for address-dependent queries

- Expression representation
- Matching algorithm

Expression Representation

- Need to distinguish between integer and address values
- Can be achieved using the **relocatable addressing model**
  - *Relocatable Addressing Model for Symbolic Execution (ISSTA 2020)*

Relocatable Addressing Model

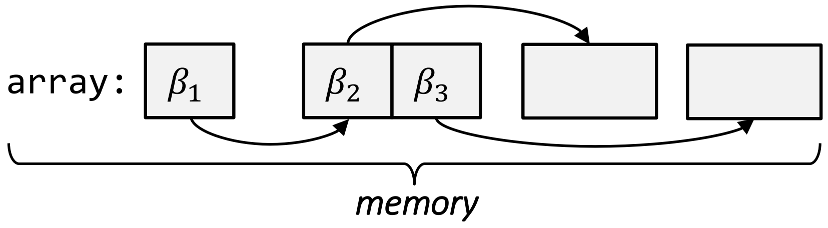
- Allocated addresses are **symbolic** values, rather than concrete
- Maintain **address constraints** to preserve the **non-overlapping** property
- Address constraints are substituted when a query is sent to the solver

Relocatable Addressing Model

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```

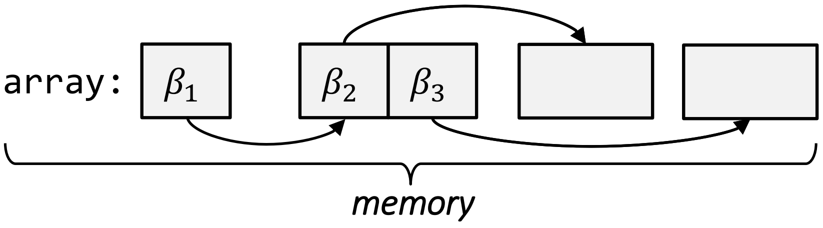
Relocatable Addressing Model

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```



Relocatable Addressing Model

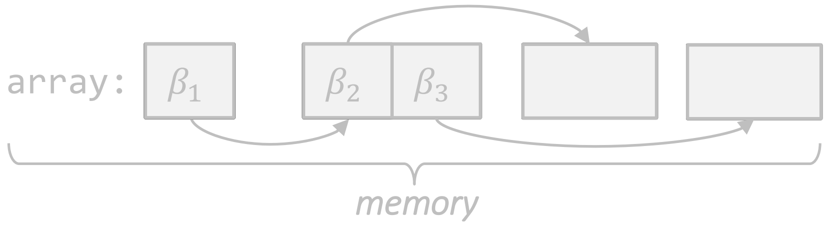
```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```



$$\underbrace{\beta_1 = 100, \beta_2 = 200, \beta_3 = 300}_{\textit{address constraints}}$$

Relocatable Addressing Model

```
int z; // symbolic
if (z > 0)
  allocate_objects();

char **array;
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;

int i,j; // symbolic: i<2,j<2
if (array[i][j] == 7)
  do_something();
```



$$\underbrace{\beta_1 = 100, \beta_2 = 200, \beta_3 = 300}_{\textit{address constraints}}$$

$$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$$

$$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$$

query:

$$pc \ \wedge \ select(a_2, p - \beta_2) = 7$$

Relocatable Addressing Model

$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \land \ select(a_2, p - \beta_2) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow \beta_5, 1 \rightarrow \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \land \ select(a_5, p - \beta_5) = 7$

Relocatable Addressing Model

$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \;\wedge\; j < 2 \;\wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \;\wedge\; select(a_2, p - \beta_2) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \;\wedge\; j < 2 \;\wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \;\wedge\; select(a_5, p - \beta_5) = 7$

- Can distinguish between integers and address values

Relocatable Addressing Model

$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_2 \le p \le \beta_2 + 2$

query:

$pc \ \land \ select(a_2, p - \beta_2) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_5 \le p \le \beta_5 + 2$

query:

$pc \ \land \ select(a_5, p - \beta_5) = 7$

- Can distinguish between integers and address values
- Identical up to renaming
    - $\beta_2 \leftrightarrow \beta_5, \ \beta_3 \leftrightarrow \beta_6$

Relocatable Addressing Model

$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \wedge \ select(a_2, p - \beta_2) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow \beta_5, 1 \rightarrow \beta_6], i) + j$

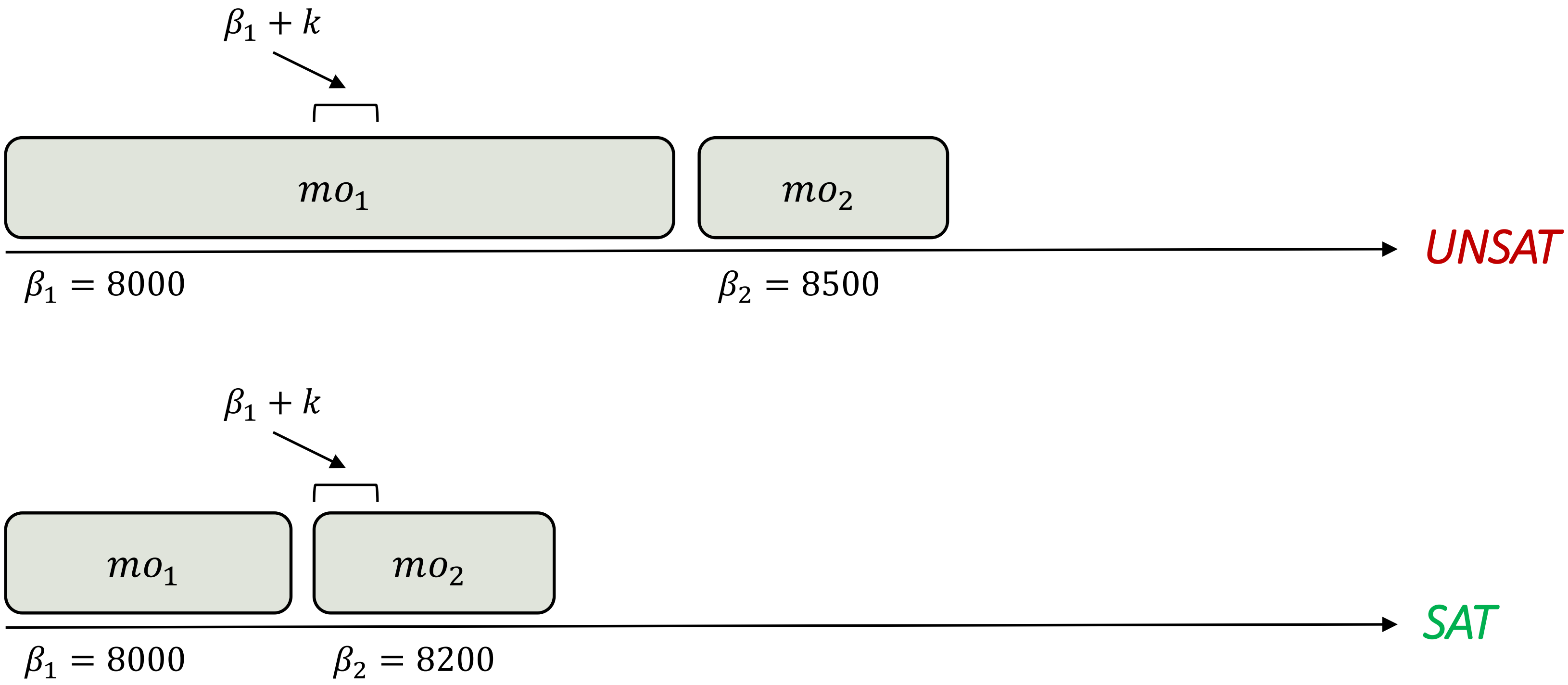
$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \wedge \ select(a_5, p - \beta_5) = 7$

- Can distinguish between integers and address values
- Identical up to renaming
  - $\beta_2 \leftrightarrow \beta_5, \ \beta_3 \leftrightarrow \beta_6$
- Is it enough?

$$100 \le k \le 101 \land \beta_2 \le \beta_1 + k < \beta_2 + 100$$



Address-Agnostic Queries

Definition:

A query ***q*** is address-agnostic if:

- Its satisfiability doesn’t change under isomorphic address spaces

Address-Agnostic Queries

Property:

- No undefined behavior → generated queries are address agnostic

Check if two address-dependent queries are **equivalent** by:

- Checking expression isomorphism (identical up to renaming)
- Checking address space isomorphism

$$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j$$

$$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$$

query:

$$pc \ \wedge \ select(a_2, p - \beta_2) = 7$$

$$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow \beta_5, 1 \rightarrow \beta_6], i) + j$$

$$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$$

query:

$$pc \ \wedge \ select(a_5, p - \beta_5) = 7$$

$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \wedge \ select(a_2, p - \beta_2) = 7$

$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \wedge \ select(a_5, p - \beta_5) = 7$

$$\beta_2 \leftrightarrow \beta_5$$
$$\beta_3 \leftrightarrow \beta_6$$

$$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$$

$$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \le p \le \beta_2 + 2$$

query:

$$pc \ \wedge \ select(a_2, p - \beta_2) = 7$$

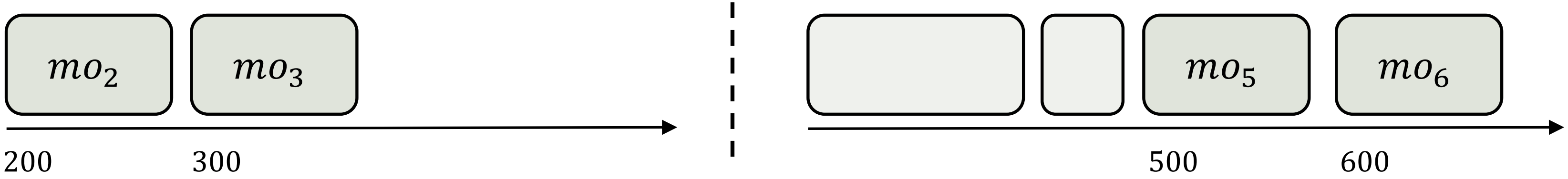
$$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$$

$$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \le p \le \beta_5 + 2$$

query:

$$pc \ \wedge \ select(a_5, p - \beta_5) = 7$$

$$\beta_2 \leftrightarrow \beta_5$$
$$\beta_3 \leftrightarrow \beta_6$$



$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \land \ select(a_2, p - \beta_2) = 7$

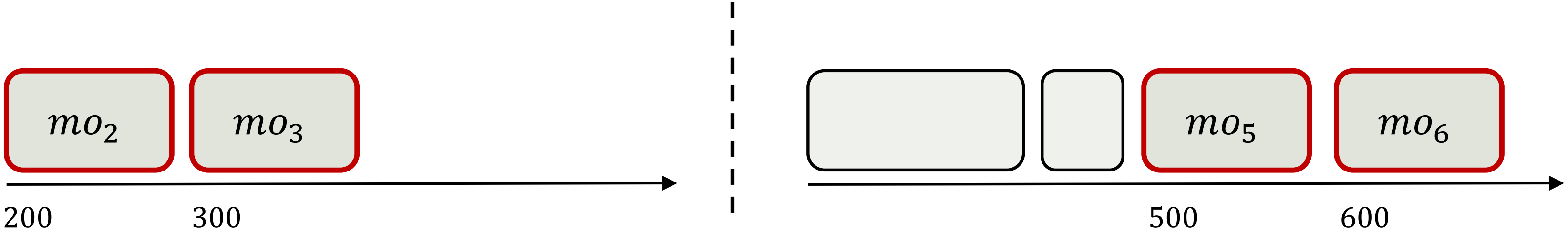
$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow \beta_5, 1 \rightarrow \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \land \ j < 2 \ \land \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \land \ select(a_5, p - \beta_5) = 7$

$$\beta_2 \leftrightarrow \beta_5$$
$$\beta_3 \leftrightarrow \beta_6$$



$p \stackrel{\text{def}}{=} select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \wedge \ select(a_2, p - \beta_2) = 7$

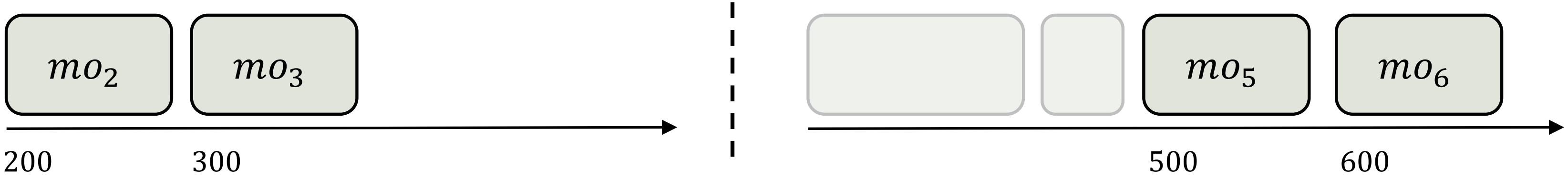
$p \stackrel{\text{def}}{=} select(a_4[0 \rightarrow \beta_5, 1 \rightarrow \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \wedge \ select(a_5, p - \beta_5) = 7$

$$\beta_2 \leftrightarrow \beta_5$$
$$\beta_3 \leftrightarrow \beta_6$$



$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \wedge \ select(a_2, p - \beta_2) = 7$

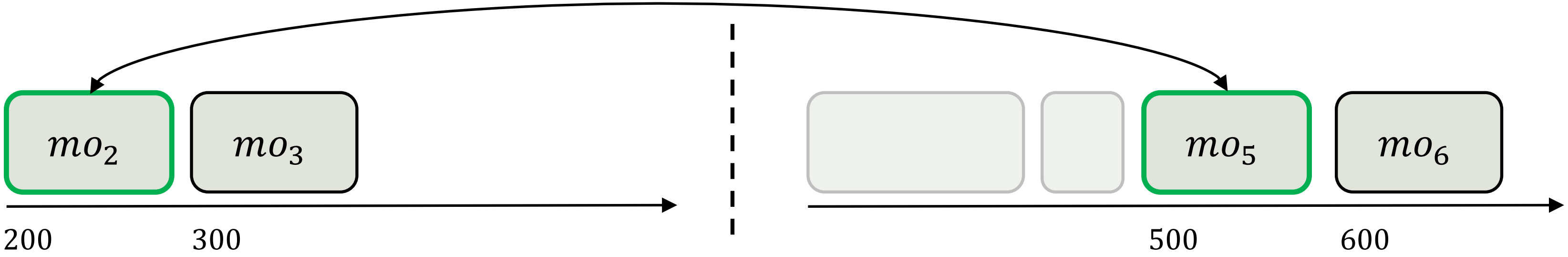
$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \wedge \ select(a_5, p - \beta_5) = 7$

$\beta_2 \leftrightarrow \beta_5$
$\beta_3 \leftrightarrow \beta_6$



$|mo_2| = |mo_5|$

$p \stackrel{\text{def}}{=} select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_2 \leq p \leq \beta_2 + 2$

query:

$pc \ \wedge \ select(a_2, p - \beta_2) = 7$

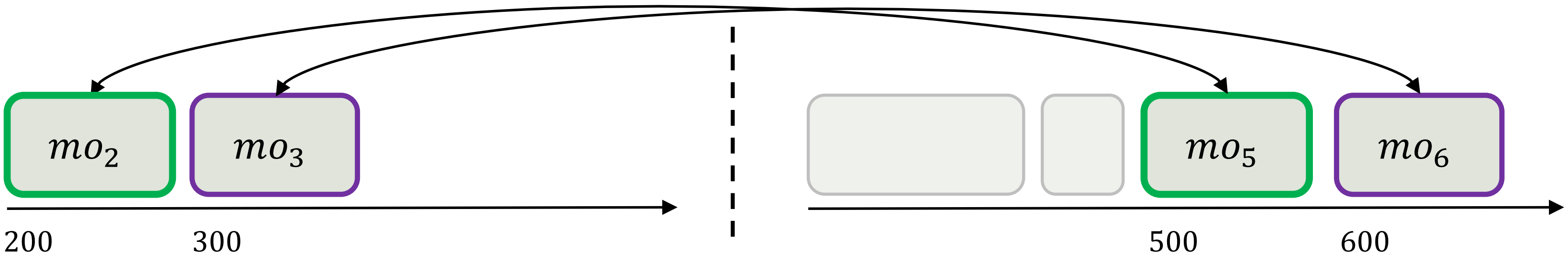
$p \stackrel{\text{def}}{=} select(a_4[0 \to \beta_5, 1 \to \beta_6], i) + j$

$pc \stackrel{\text{def}}{=} i < 2 \ \wedge \ j < 2 \ \wedge \beta_5 \leq p \leq \beta_5 + 2$

query:

$pc \ \wedge \ select(a_5, p - \beta_5) = 7$

$$\beta_2 \leftrightarrow \beta_5$$
$$\beta_3 \leftrightarrow \beta_6$$



$$|mo_2| = |mo_5|$$
$$|mo_3| = |mo_6|$$

Limitations

Undefined behavior

- Branch depends on address space layout
- Crossing boundaries using pointer arithmetic

```
char *p = malloc(10);
char *q = malloc(50);

if (p > q) {
  …
}

if (*(p + 100) == *q) {

}
```

Implementation

- On top of KLEE
- Query caching
  - Standard approach uses hash table
  - We need **address-agnostic** hash function

$$h(select(a_1[0 \to \beta_1, 1 \to \beta_2 + 2]) = 17) \neq h(select(a_1[0 \to \beta_3, 1 \to \beta_4 + 2]) = 17)$$

Implementation

- On top of KLEE
- Query caching
  - Standard approach uses hash table
  - We need **address-agnostic** hash function

$$\forall \beta, \beta'. h'(\beta) = h'(\beta')$$

Implementation

- On top of KLEE
- Query caching
  - Standard approach uses hash table
  - We need **address-agnostic** hash function

$$\forall \beta, \beta'. h'(\beta) = h'(\beta')$$

$$\Downarrow$$

$$h'(select(a_1[0 \to \beta_1, 1 \to \beta_2 + 2]) = 17) \quad = \quad h'(select(a_1[0 \to \beta_3, 1 \to \beta_4 + 2]) = 17)$$

Evaluation

Compare two querying caching approaches:

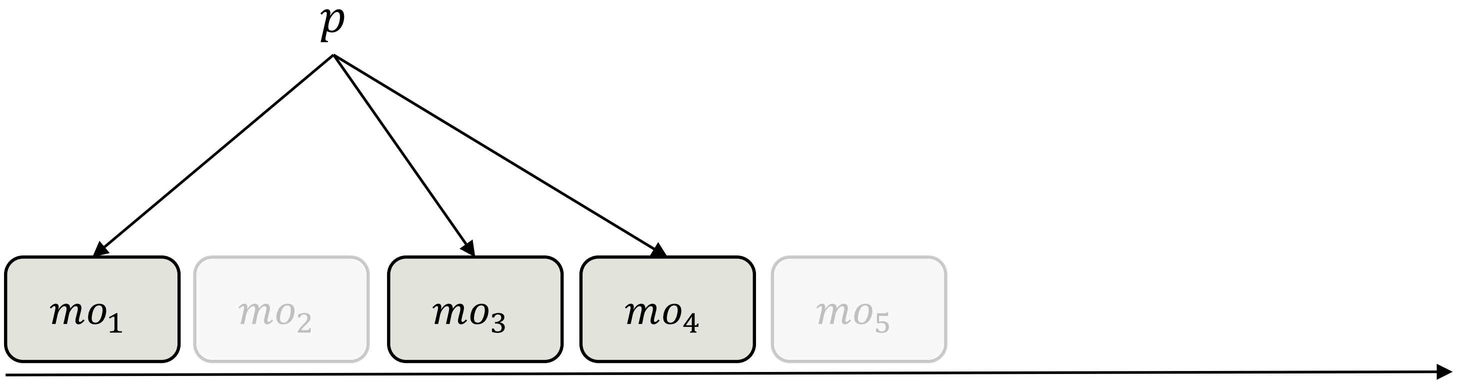
- Standard (syntactic) *(Base)*
- Address-Aware *(AA)*

Under two memory models:

- Forking *(FMM)*
  - Vanilla KLEE
- Dynamically Segmented *(DSMM)*
  - *Relocatable Addressing Model for Symbolic Execution (ISSTA 2020)*

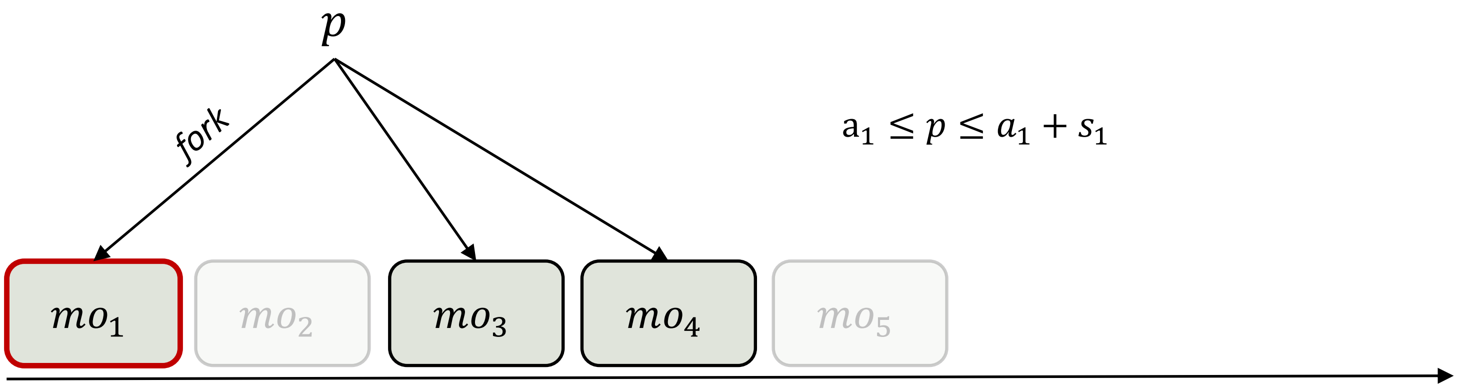
Forking Memory Model

- Fork for each pointed memory object
- Used in vanilla KLEE



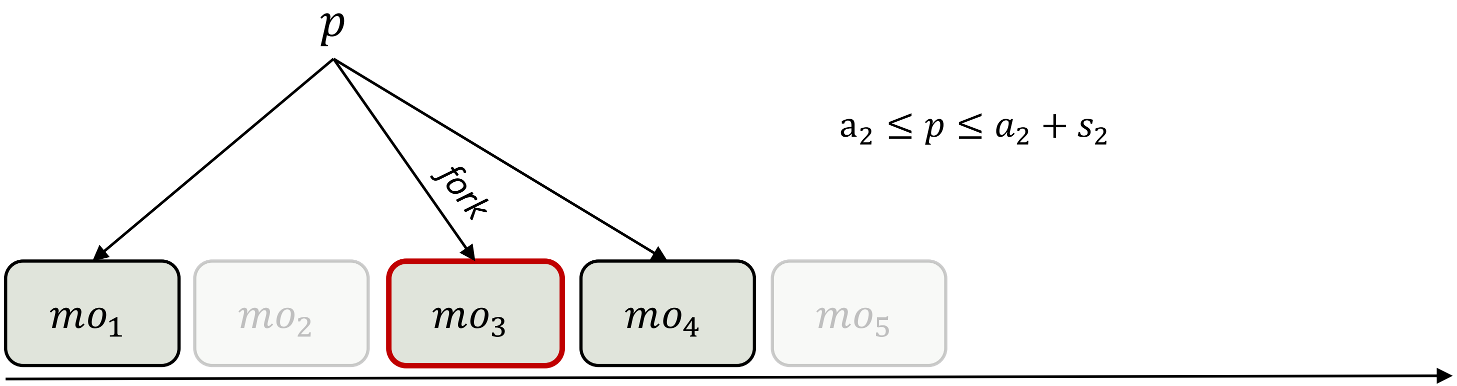
Forking Memory Model

- Fork for each pointed memory object
- Used in vanilla KLEE



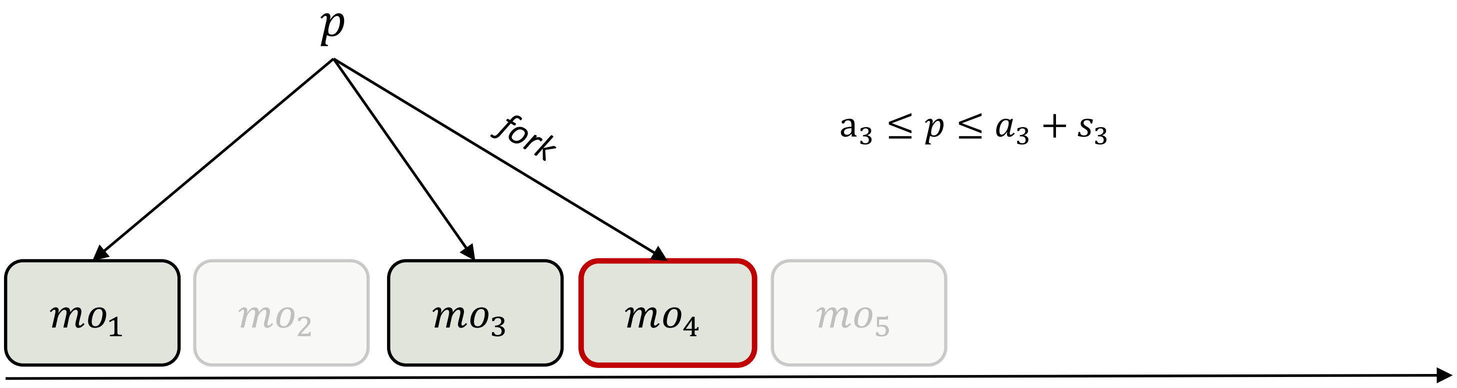
Forking Memory Model

- Fork for each pointed memory object
- Used in vanilla KLEE



Forking Memory Model

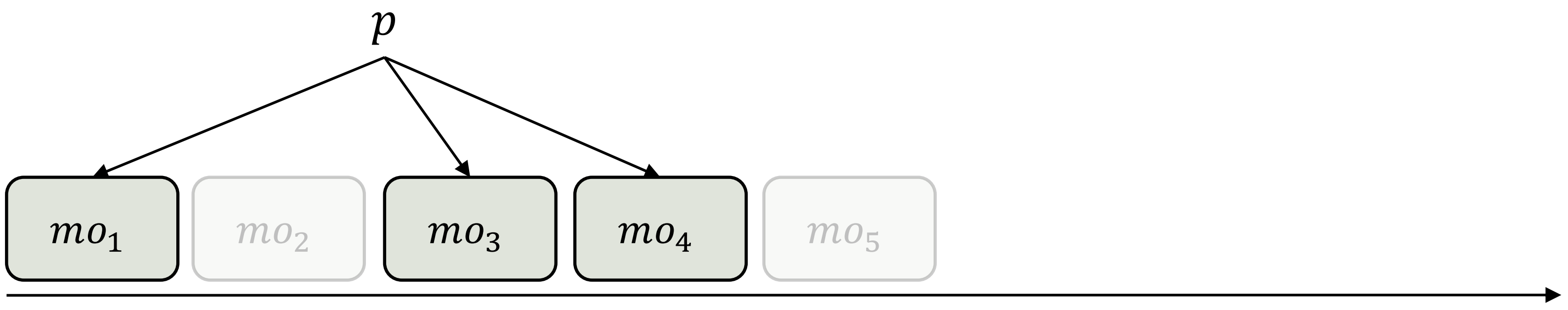
- Fork for each pointed memory object
- Used in vanilla KLEE



$a_3 \leq p \leq a_3 + s_3$

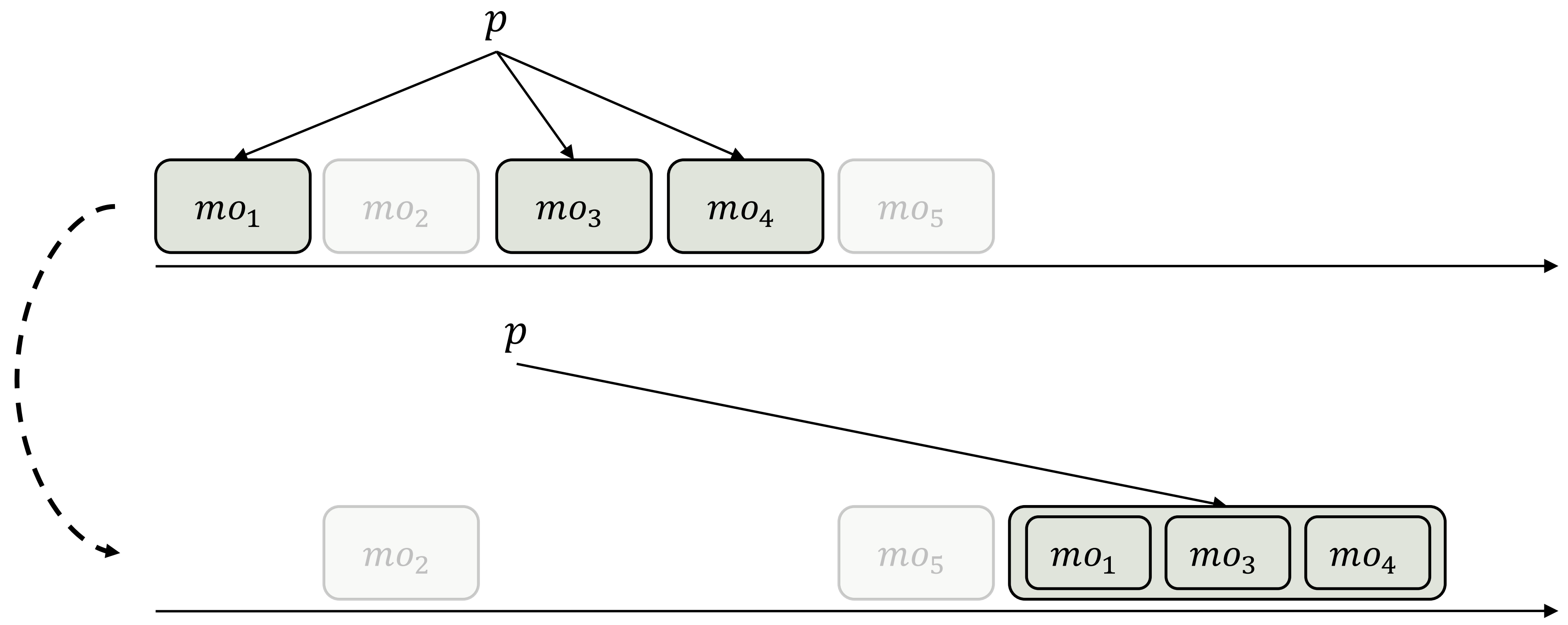
Dynamically Segmented Memory Model

- Merge memory objects to a **single segment**



Dynamically Segmented Memory Model

- Merge memory objects to a **single segment**



Empirical Validation

Validate query caching correctness:

- Consistency of query results
- Consistency of explored paths / coverage

Evaluation

Number of queries

Benchmark	FMM			DSMM		
	Base	AA	Ratio	Base	AA	Ratio
m4	10792	4265	2.53x	1600	1289	1.24x
make	347324	45471	7.63x	50558	9753	5.18x
sqlite	5622	4681	1.20x	14563	12993	1.12x
apr	445	300	1.48x	126	86	1.46x
libxml2	124782	6118	20.39x	124782	6118	20.39x
expat	89740	31747	2.82x	89736	31761	2.82x
bash	8538	4479	1.90x	7542	4098	1.84x
json-c	15364	5246	2.92x	2757	1523	1.81x

Evaluation

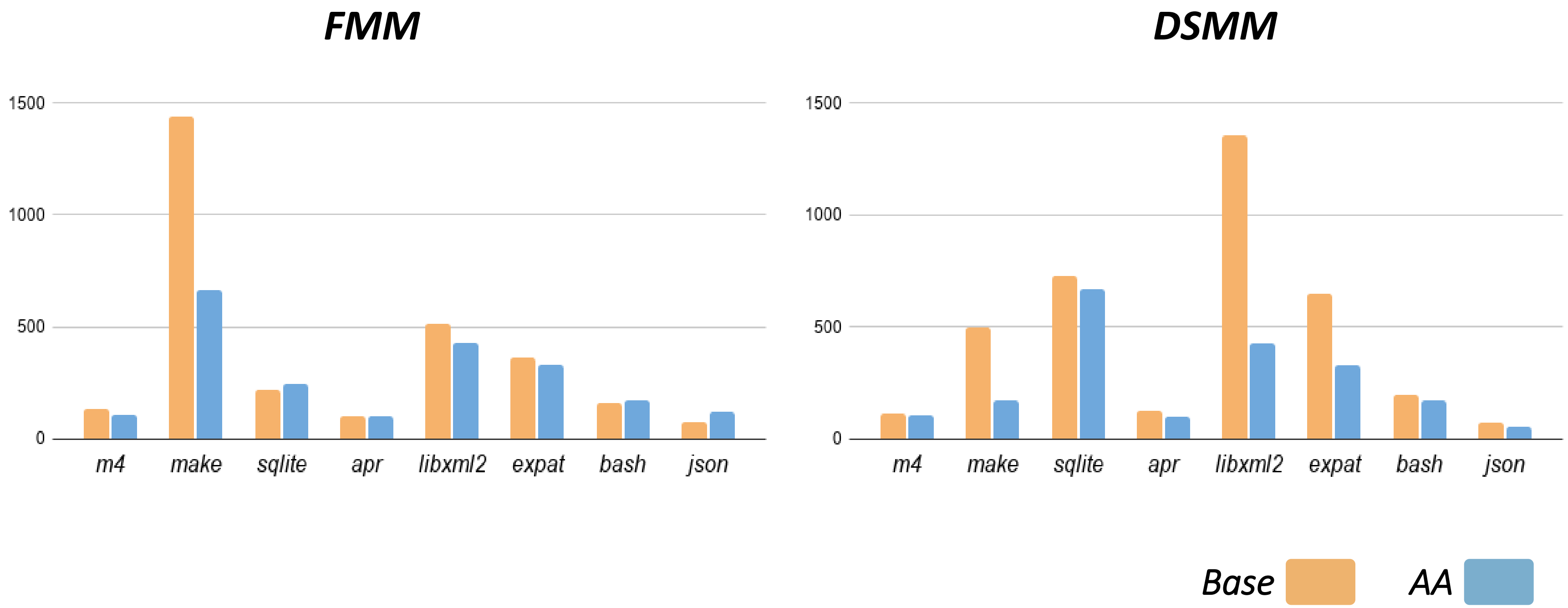
Analysis time

Benchmark	FMM			DSMM		
	Base	AA	Speedup	Base	AA	Speedup
m4	00:13:16	00:04:59	2.67x	00:19:17	00:14:55	1.29x
make	06:46:44	02:30:51	2.69x	03:56:42	01:47:23	4.11x
sqlite	00:17:20	00:14:24	1.20x	04:00:17	03:12:22	1.24x
apr	00:57:33	00:39:05	1.47x	00:20:20	00:13:39	1.49x
libxml2	02:33:33	00:17:09	8.96x	02:27:35	00:17:12	8.58x
expat	00:26:02	00:23:19	1.11x	00:25:13	00:23:06	1.09x
bash	02:37:48	01:23:30	1.88x	02:39:04	01:14:18	2.14x
json-c	00:31:36	00:13:20	2.37x	00:08:05	00:04:19	1.87x

hh:mm:ss

Evaluation

Memory usage *(MB)*



Evaluation

Runtime overhead:

- Programs **without** address-dependent queries
  - coreutils, libosip, libyaml
- FMM
  - Average: 6%
  - Max: 17% (libosip)
- DSMM
  - Negligible overhead

Conclusion

- Query caching technique for **address dependent** queries
- Significant performance improvement:
  - Number of queries
  - Analysis time
- Reasonably low overhead

Available on GitHub: https://github.com/davidtr1037/klee-aaqc