

MAM: A Memory Allocation Manager for GPUs

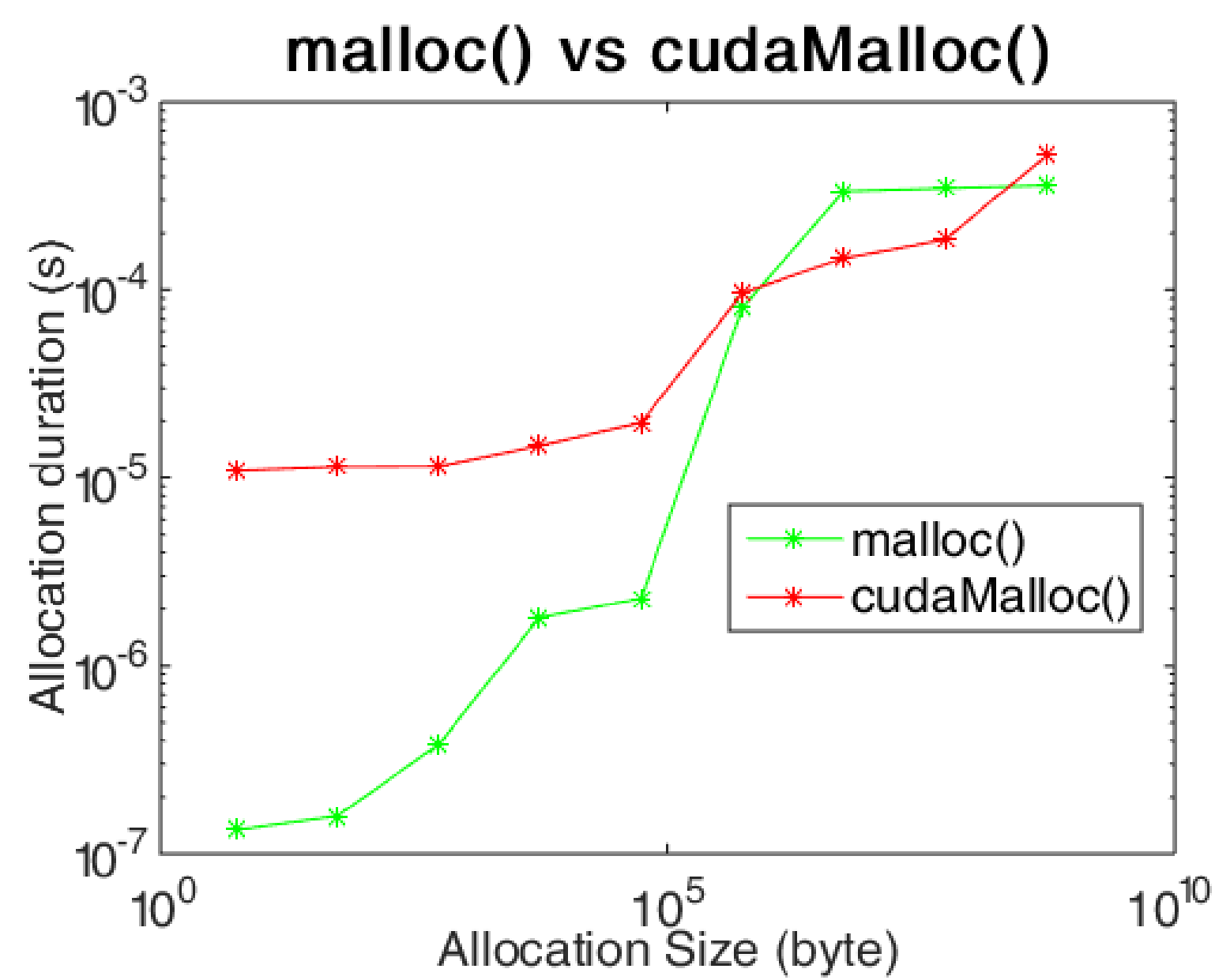
Can Aknesil

Computer Science and Engineering, Koç University, İstanbul
caknesil13@ku.edu.tr

Didem Unat

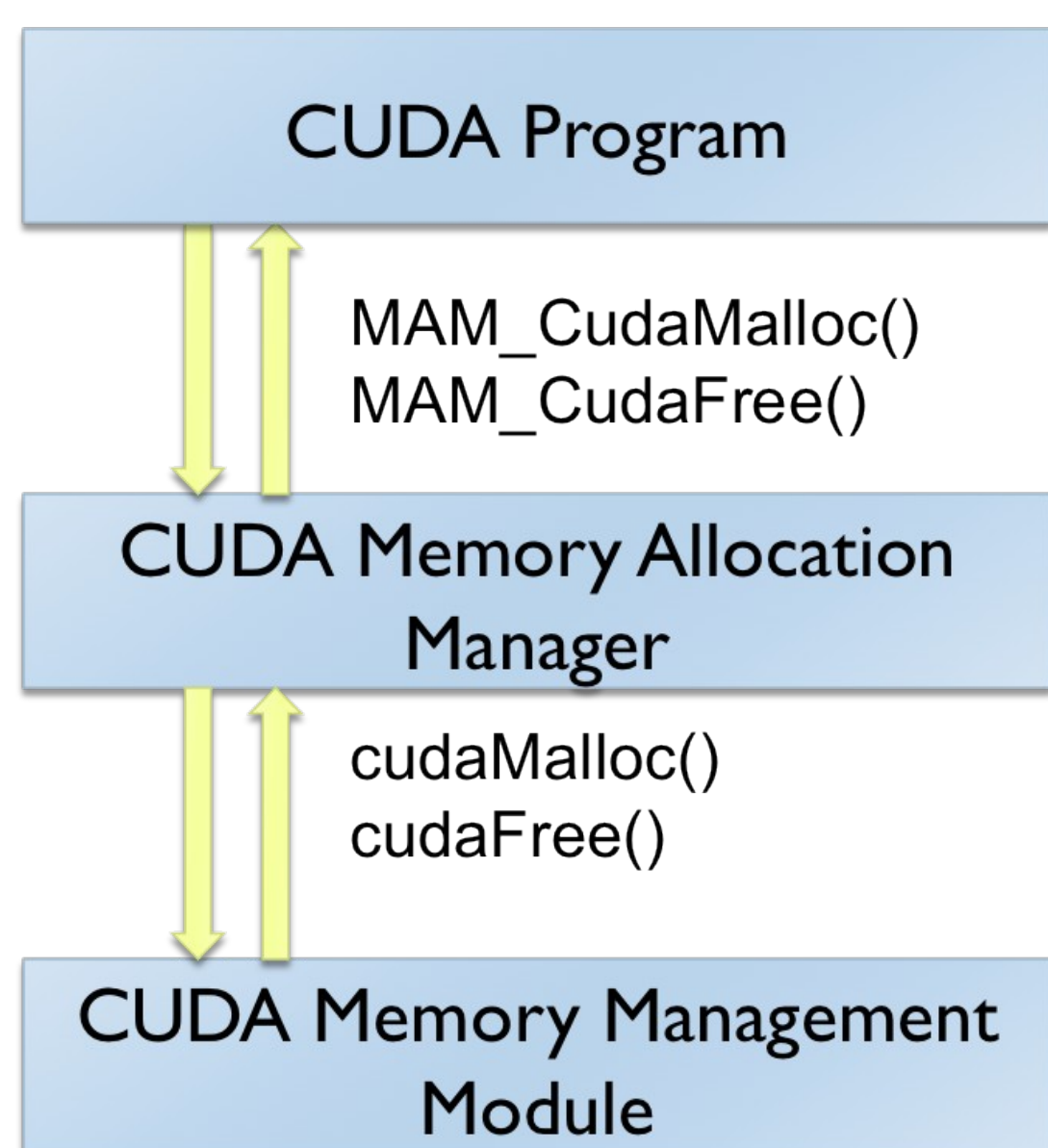
Computer Science and Engineering, Koç University, İstanbul
dunat@ku.edu.tr

Motivation



- `cudaMalloc()` is slow especially when allocation size is large.
- Applications requiring repetitive allocations may reduce the performance.
- We develop a memory management library primarily for GPUs.

What is MAM ?



- MAM is a C library that provides an abstraction layer between the programmer and the memory management module of CUDA environment.
- MAM removes the overhead of memory allocation and deallocation.

How to use MAM ?

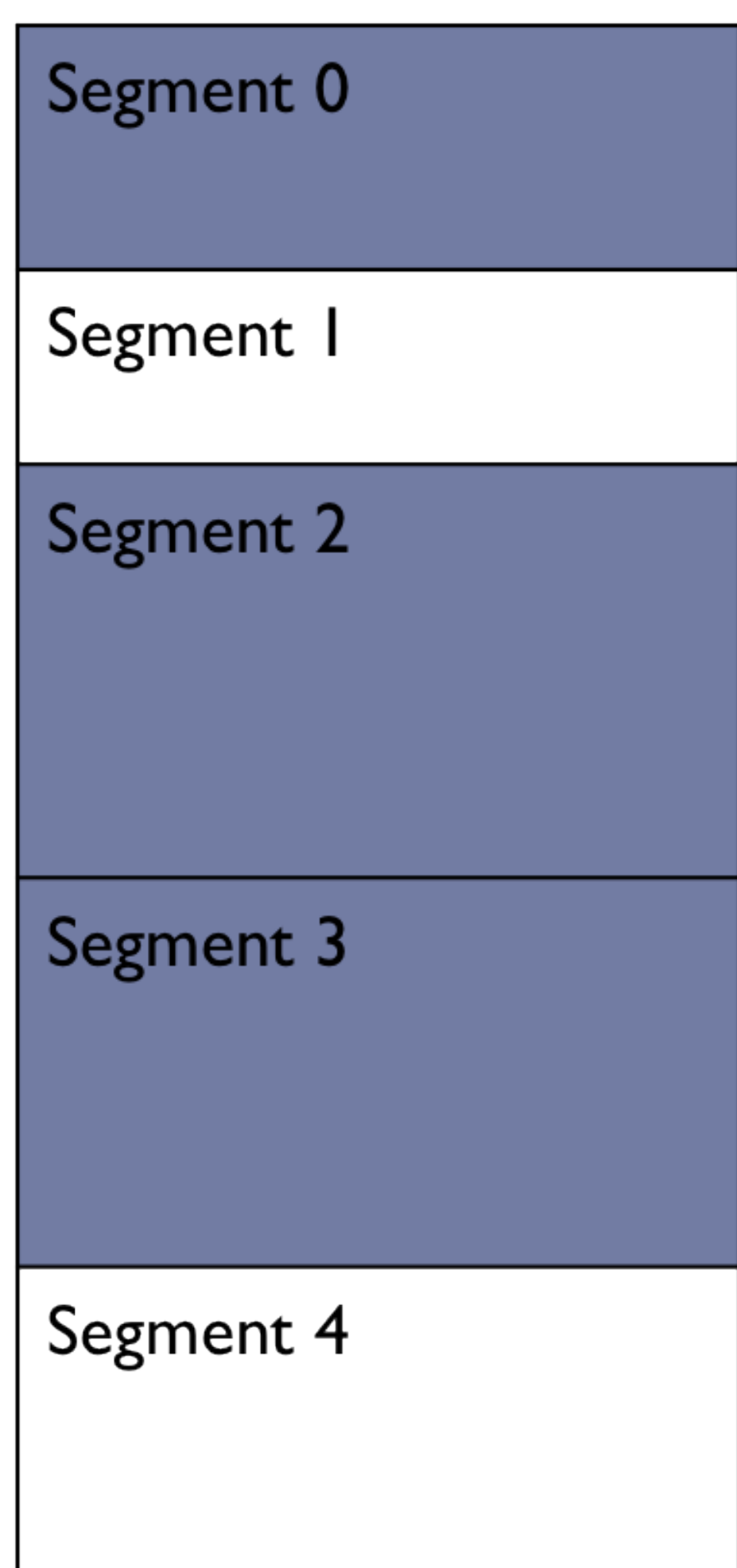
WITHOUT MAM

```
cudaMalloc(&ptr, size);
...
cudaFree(ptr);
```

WITH MAM

```
MAM_Create();
MAM_CudaMalloc(&ptr, size);
...
MAM_CudaFree(ptr);
MAM_Destroy();
```

How MAM Works ?



- When `MAM_Create()` is called, a large continuous *chunk* of memory is allocated on the device.
- The size of the *chunk* is expected to be greater than the total size of the memory that will be allocated by the CUDA program at a time instance.
- During allocation and deallocation, MAM manipulates *segments* of the *chunk* by creating and destroying them.
- Each *segment* is nothing but a small sized structure stored in the host memory.

```
struct segment {
    void *basePtr;
    size_t size;
    char isEmpty;
    /* data related to data structures */
};
```

Blue: Segments being used
White: Empty segments

Data Structures in MAM

The Pointer Tree

- Stores empty segments sorted according to their sizes.
- Red-Black Tree is used.

The Size Tree-Dictionary

- Stores empty segments sorted according to their sizes
- Red-Black Tree is used.

Allocation Algorithm

procedure ALLOCATE

```
Find a best-fitting empty segment from the tree-dictionary  $O(\log n)$ 
Mark the segment as filled  $O(1)$ 
if The segment perfectly fits  $O(1)$  then
    Remove segment from tree-dictionary  $O(\log n)$ 
else
    Resize it  $O(1)$ 
    Remove it from tree-dictionary  $O(\log n)$ 
    Create a new empty segment  $O(1)$ 
    Insert it in pointer-tree & tree-dictionary  $O(\log n)$ 
end if
Return the base pointer of filled segment  $O(1)$ 
end procedure
```

Deallocation Algorithm

procedure DEALLOCATE

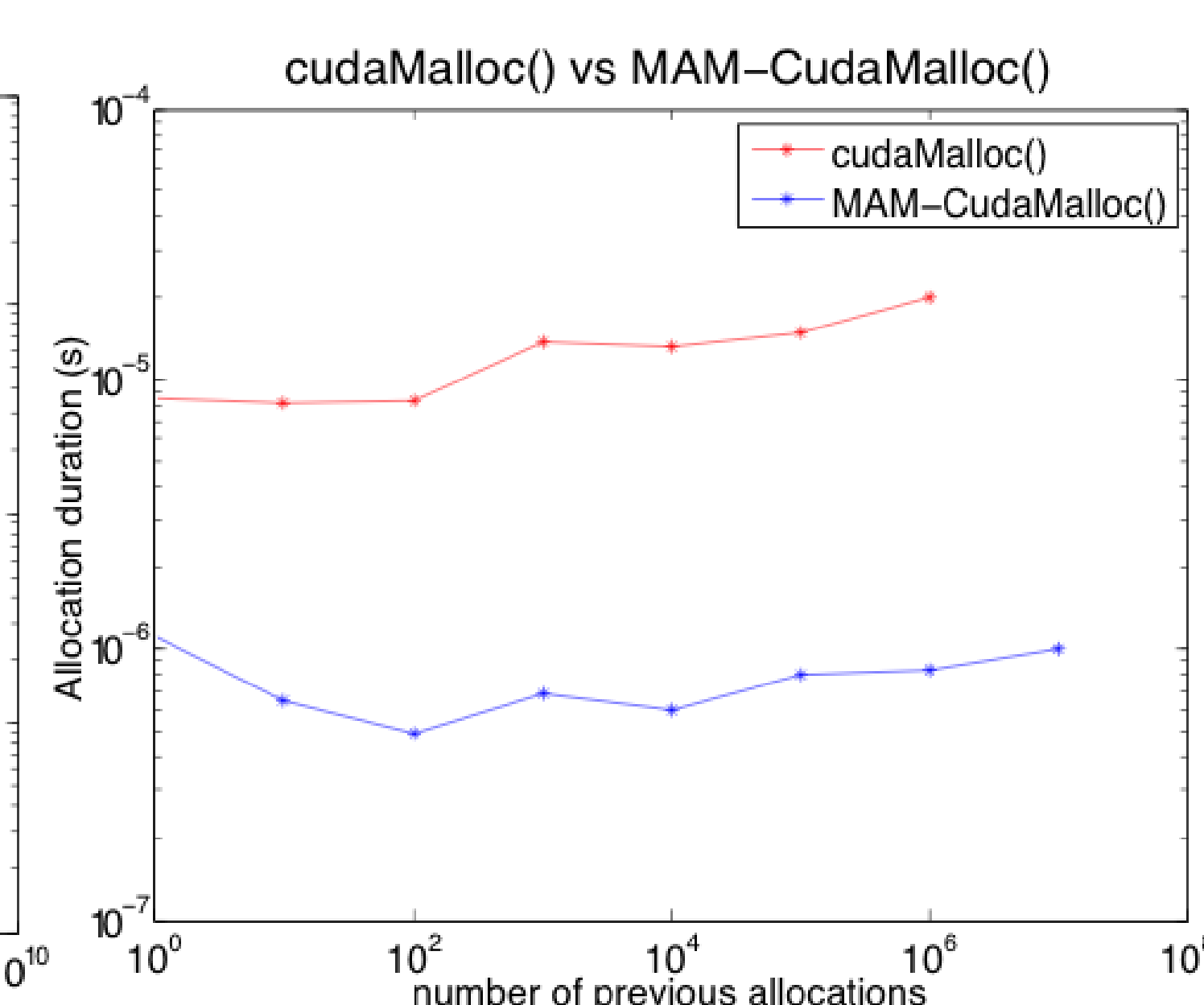
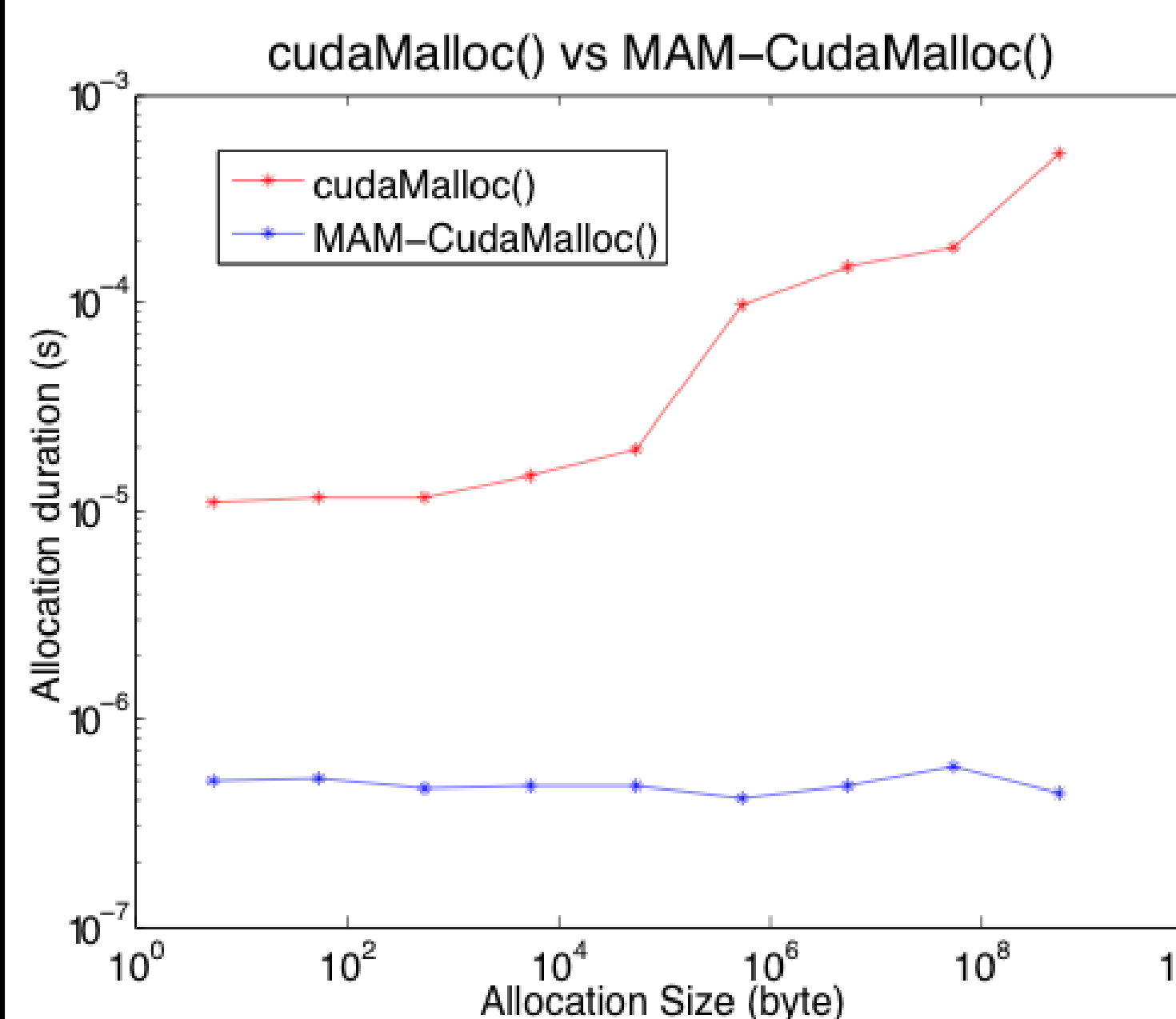
```
Find the segment in the pointer-tree  $O(\log n)$ 
Mark the segment as empty  $O(1)$ 
Get previous and next segments  $O(\log n)$ 
if the previous segment is empty  $O(1)$  then
    Remove the segment being newly emptied from pointer-tree and tree-dictionary  $O(\log n)$ 
    Destroy the segment being newly emptied  $O(1)$ 
    Resize previous segment  $O(1)$ 
    Replace it in tree-dictionary  $O(\log n)$ 
    Assign it to the variable stored the destroyed segment  $O(\log n)$ 
end if
// repeat the similar procedure for next segment.
end procedure
```

For both algorithms,

- Time complexity: $O(\log n)$
- Space complexity: $O(n)$

where n = number of segments

Performance Results



- MAM performs better in the measurement which is according to the allocation size. While the allocation duration of `cudaMalloc()` increases linearly, the duration of `MAM_CudaMalloc()` stays constant.
- MAM performs better also in the measurement which is according to the number of previously allocated segments.