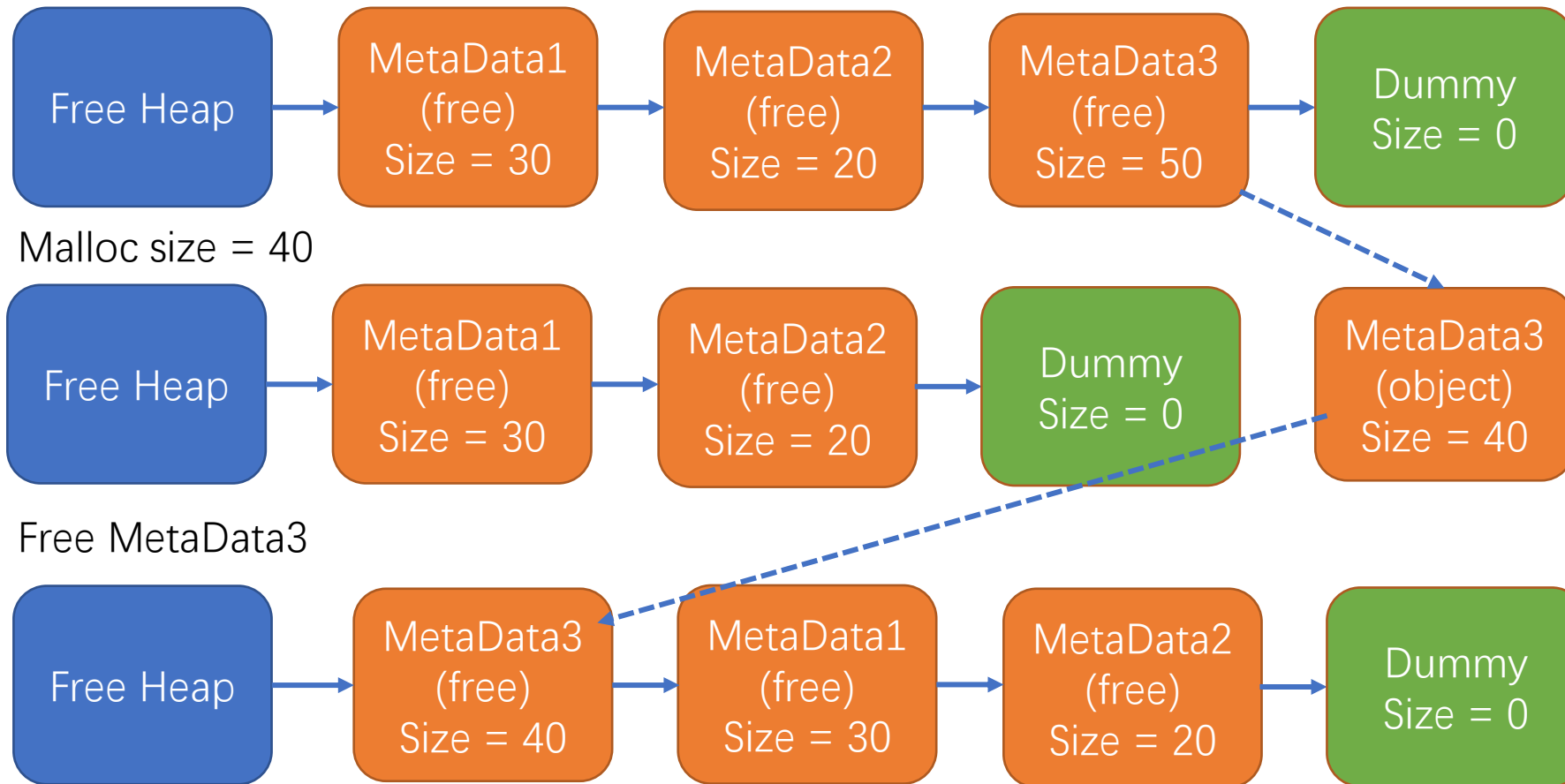


# Simple Malloc

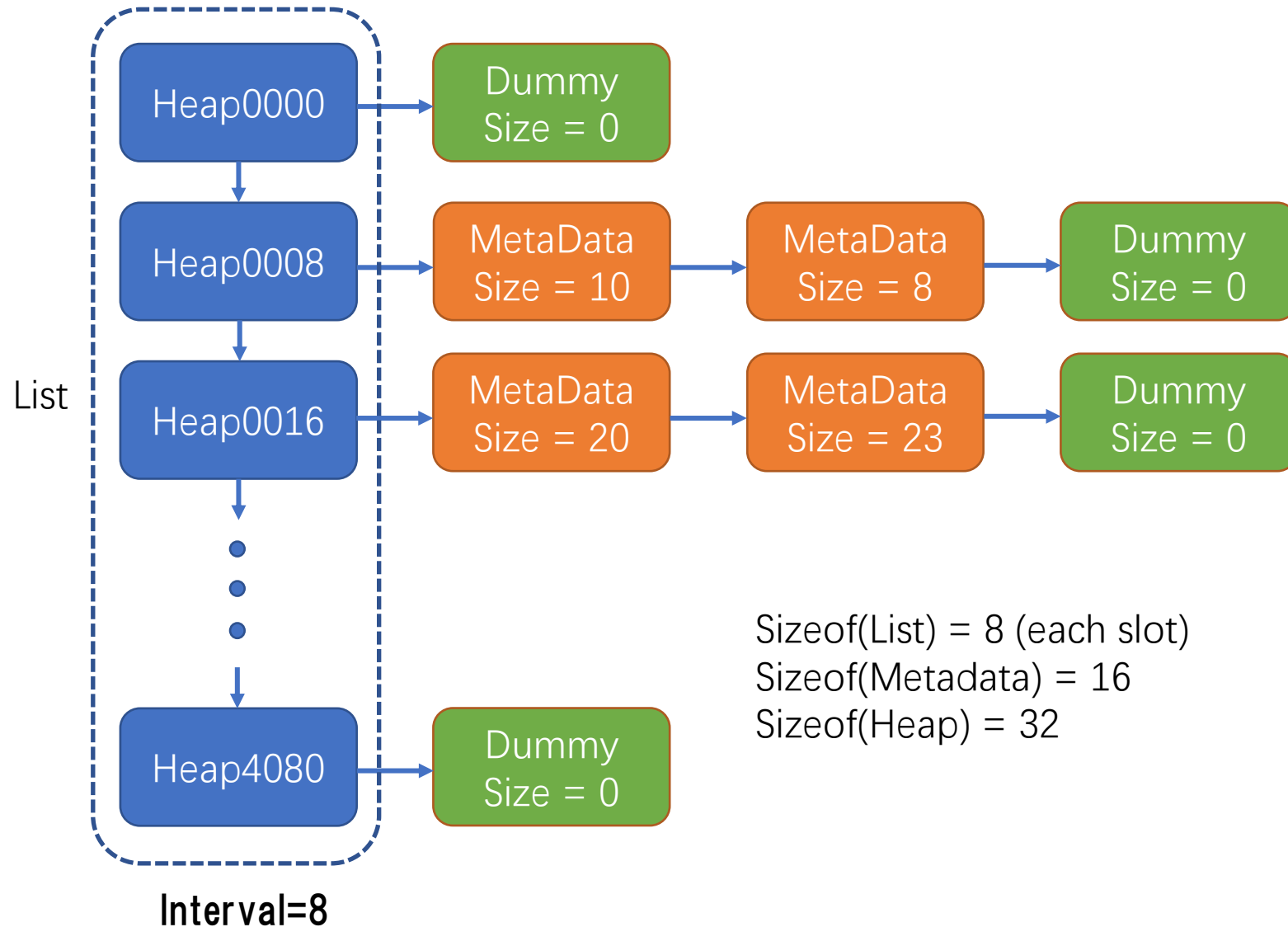


Drawback

(1) Search for a proper MetaData is slow

(2) The first available slot may not be the most proper one.

# My Malloc (Design)



In My Malloc, heaps are linked and we only put metadata into the heap that: heap size  $\leq$  size  $<$  next heap size.

e.x. Heap0008  $\rightarrow$  Heap0016  $\rightarrow$  Heap0024  
Heap0008 can contain size = 8~15  
Heap0016 can contain size = 16~23

\*\*

If we allocate new memory, we put the new metadata into Heap0000

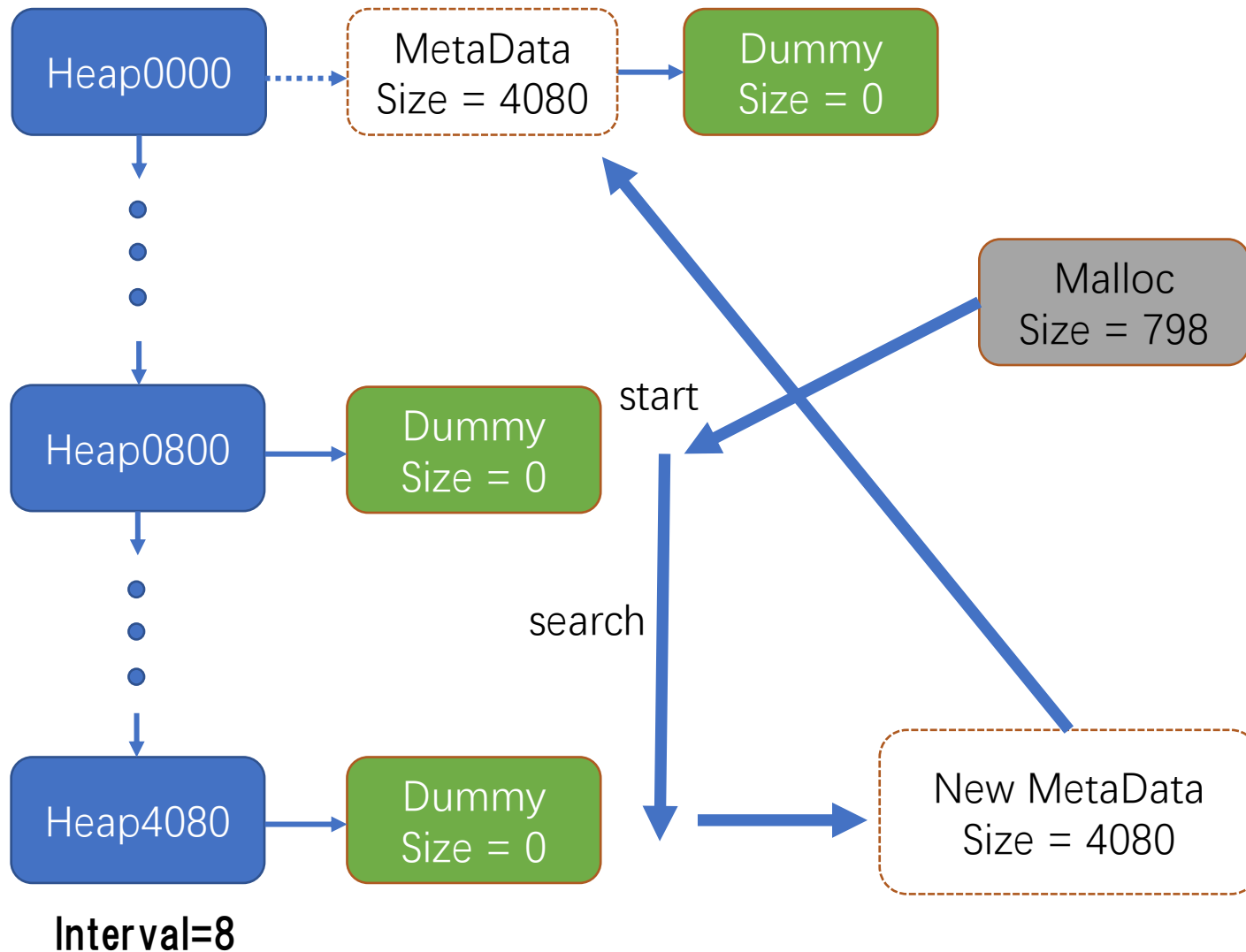
We have  $n$  additional heaps so we need first allocate new memory and use size =  $32 * n$  for heaps.

We can use list to search faster, we also need size =  $8 * n$  for the list.

$$n = 4096 / \text{interval}$$

$$\text{init\_size} = 40 * n = (40 / \text{interval}) * 4096$$

# My Malloc (Malloc and allocate new)



When we malloc a space, like size = 798. We first view Heap0000, if it is vacant, then start from Heap0800.

While

heap->next == Dummy (vacant)

We go to the next heap.

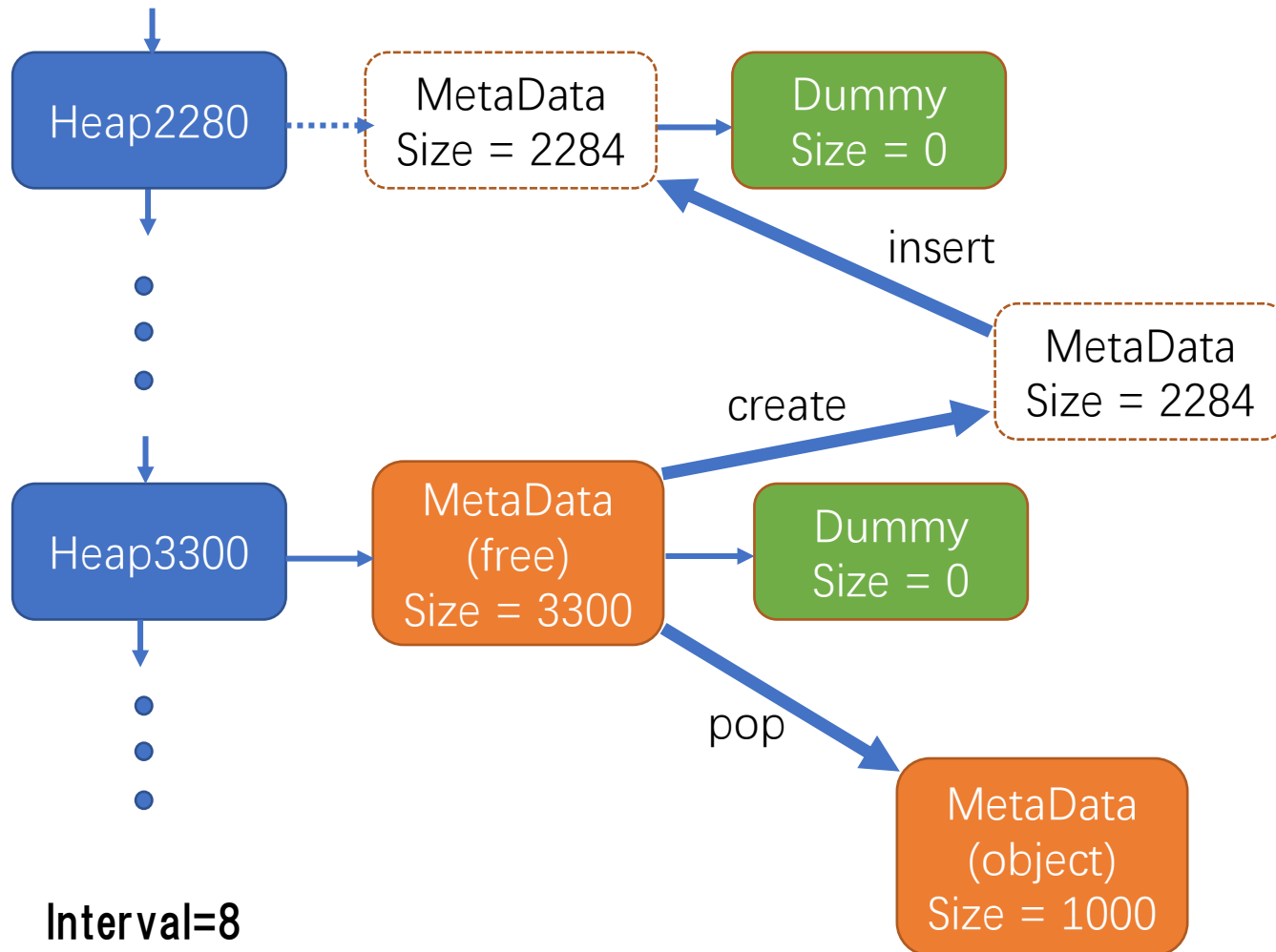
So when we stop, the current heap must have a MetaData whose size  $\geq$  object size.

If we viewed the last heap (Heap4080) but still find no proper MetaData.

We allocate for new memory (size = 4096) and create a new MetaData (size = 4080) in Heap0000.

So that when we re-run the malloc function, we can find this MetaData at the first time.

# My Malloc (Arrange object)



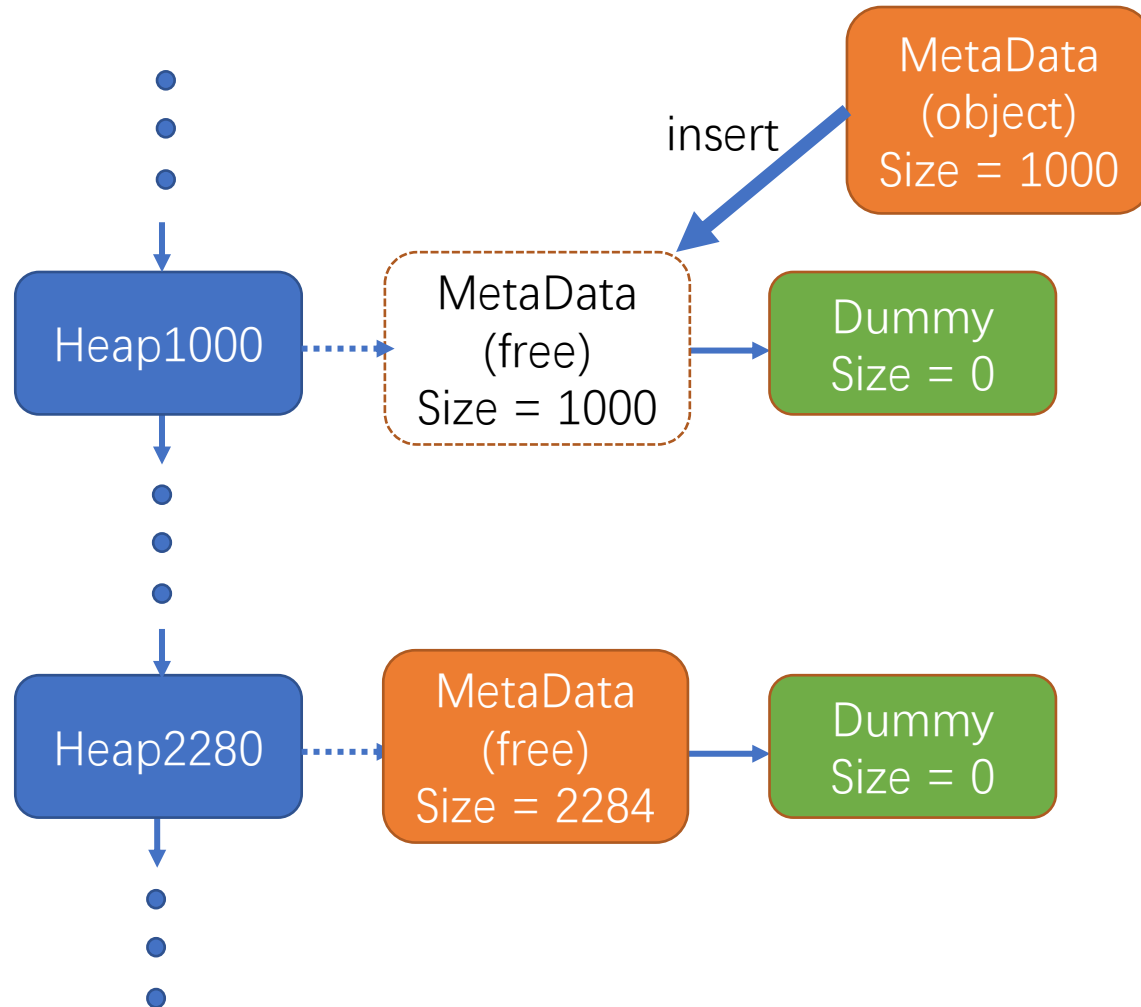
For example, when object size = 1000 and we find the proper MetaDate (size = 3300). Similar to simple malloc, we first use 1000 and the remaining size = 2300, then we use 16 for creating a new MetaData, and the rest become 2284.

Next, we will search a proper heap for 2284, that is Heap2280. So that we can insert the new MetaData.

\*\*

If the remaining size < 24, which can not contain a metadata(16) + object(>=8). We do not

# My Malloc (Free object)



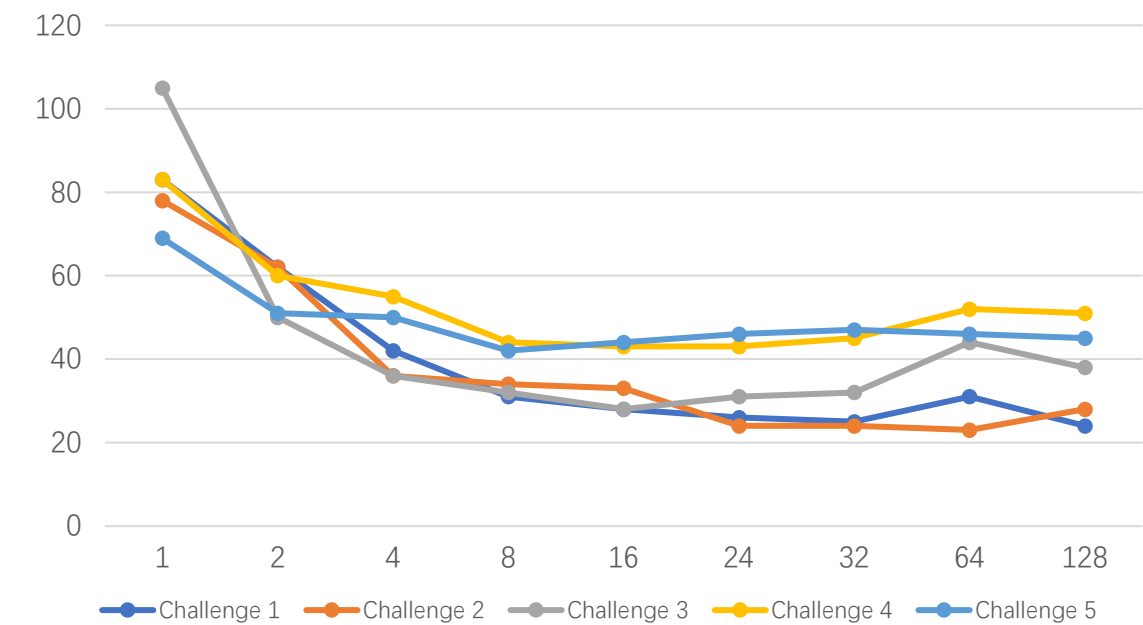
Interval=8

Similar to the last page.

Have no time to use `munmap_to_system` to release the space.

	Interval=1	Interval=2	Interval=4	Interval=8	Interval=16	Interval=24	Interval=32	Interval=64	Interval=128
C1	83ms/61%	62ms/66%	42ms/68%	31ms/70%	28ms/70%	26ms/70%	25ms/70%	31ms/70%	24ms/70%
C2	78ms/24%	62ms/29%	36ms/34%	34ms/39%	33ms/39%	24ms/39%	25ms/39%	23ms/39%	28ms/39%
C3	85ms/36%	50ms/42%	36ms/46%	32ms/51%	28ms/43%	31ms/30%	32ms/12%	44ms/4%	38ms/3%
C4	83ms/70%	60ms/71%	55ms/72%	44ms/72%	43ms/70%	43ms/67%	45ms/61%	52ms/45%	51ms/30%
C5	69ms/70%	51ms/71%	50ms/72%	42ms/72%	44ms/71%	46ms/68%	47ms/62%	46ms/46%	45ms/31%

Time-Interval



Utilization-Time

