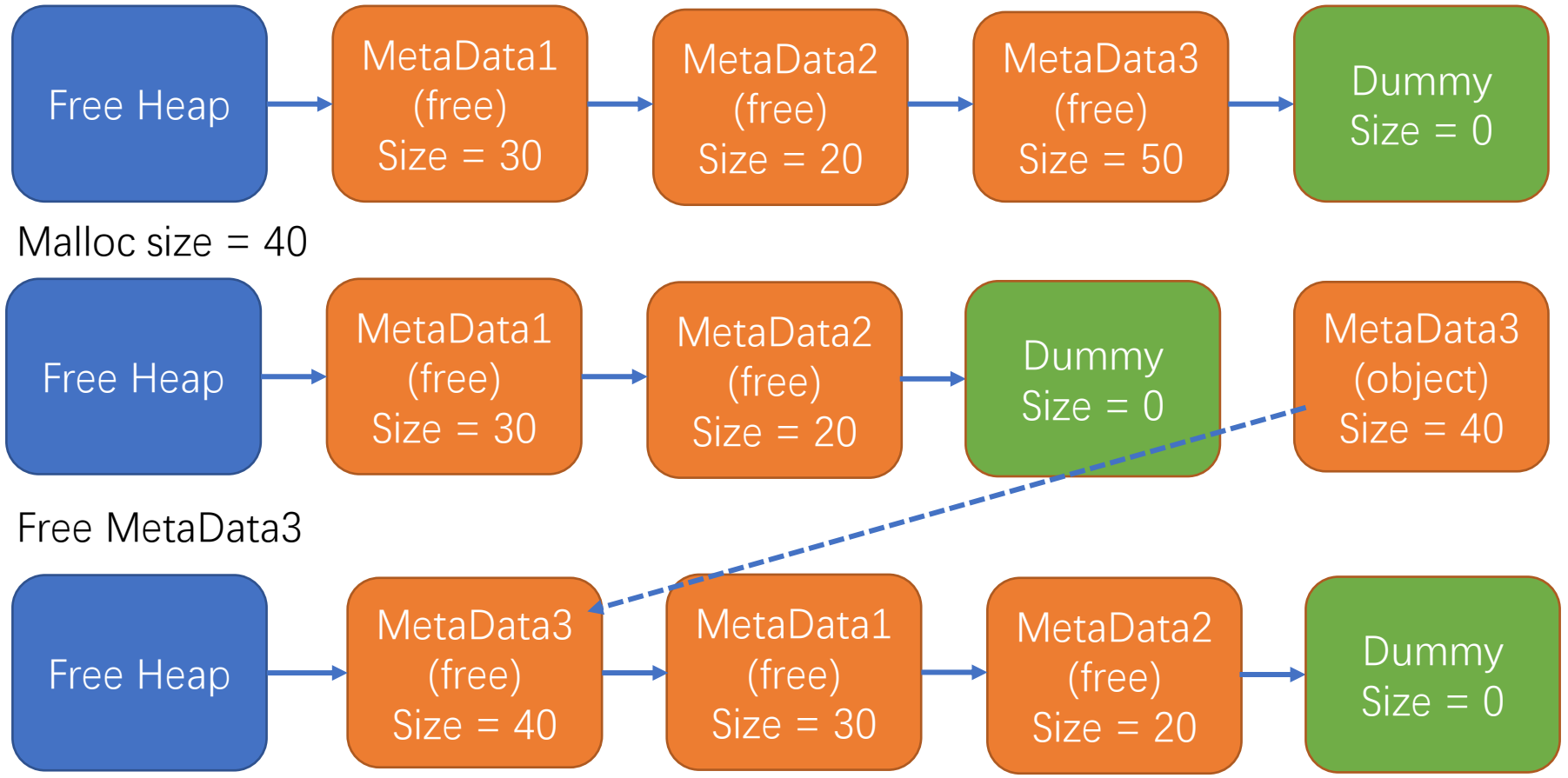
Simple Malloc



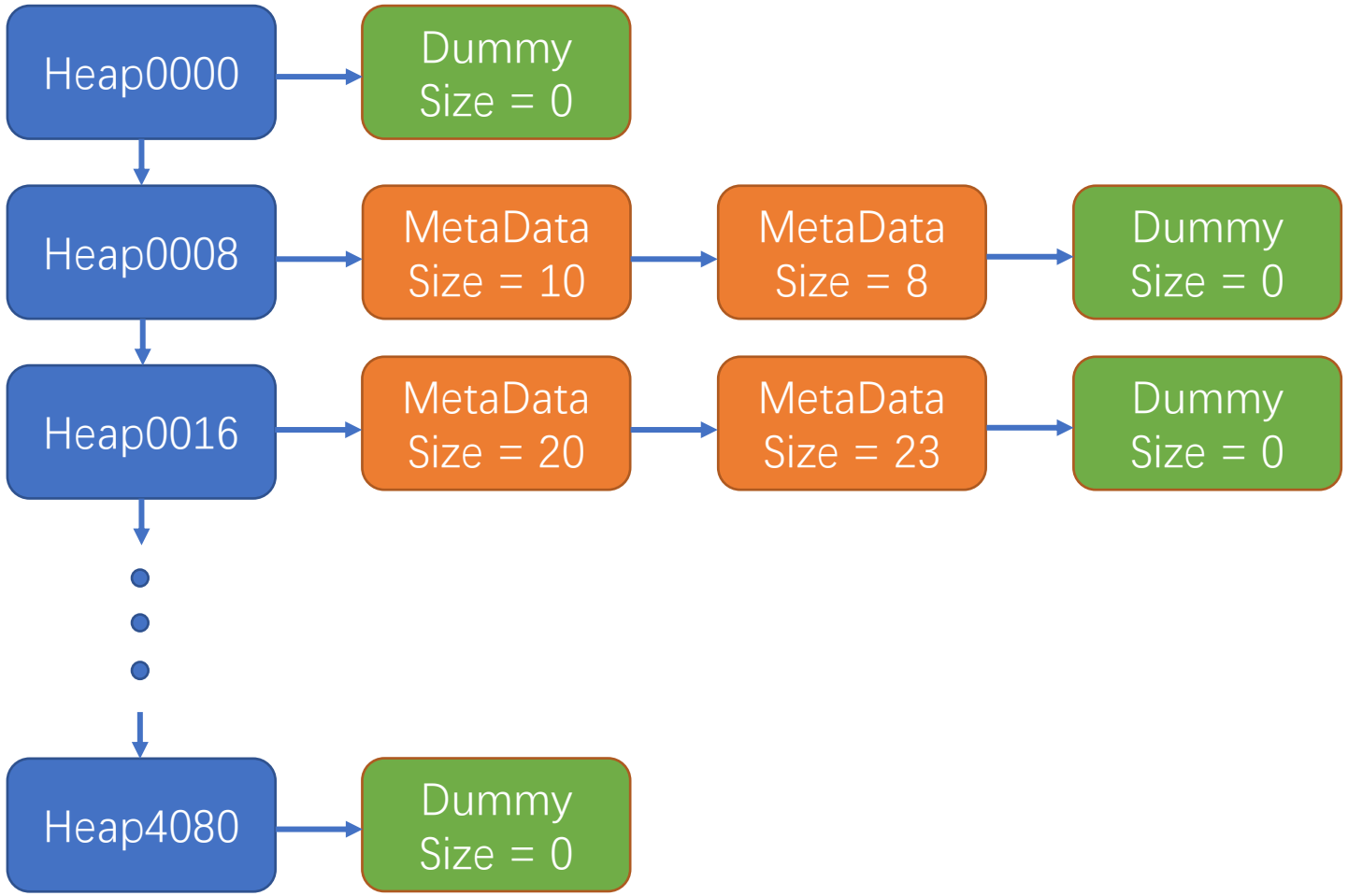
Drawback

(1) Search for a proper MetaData is slow

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

(3) When we free a data, the MetaData size will be the object size, but not the previous size.

My Malloc (Design)



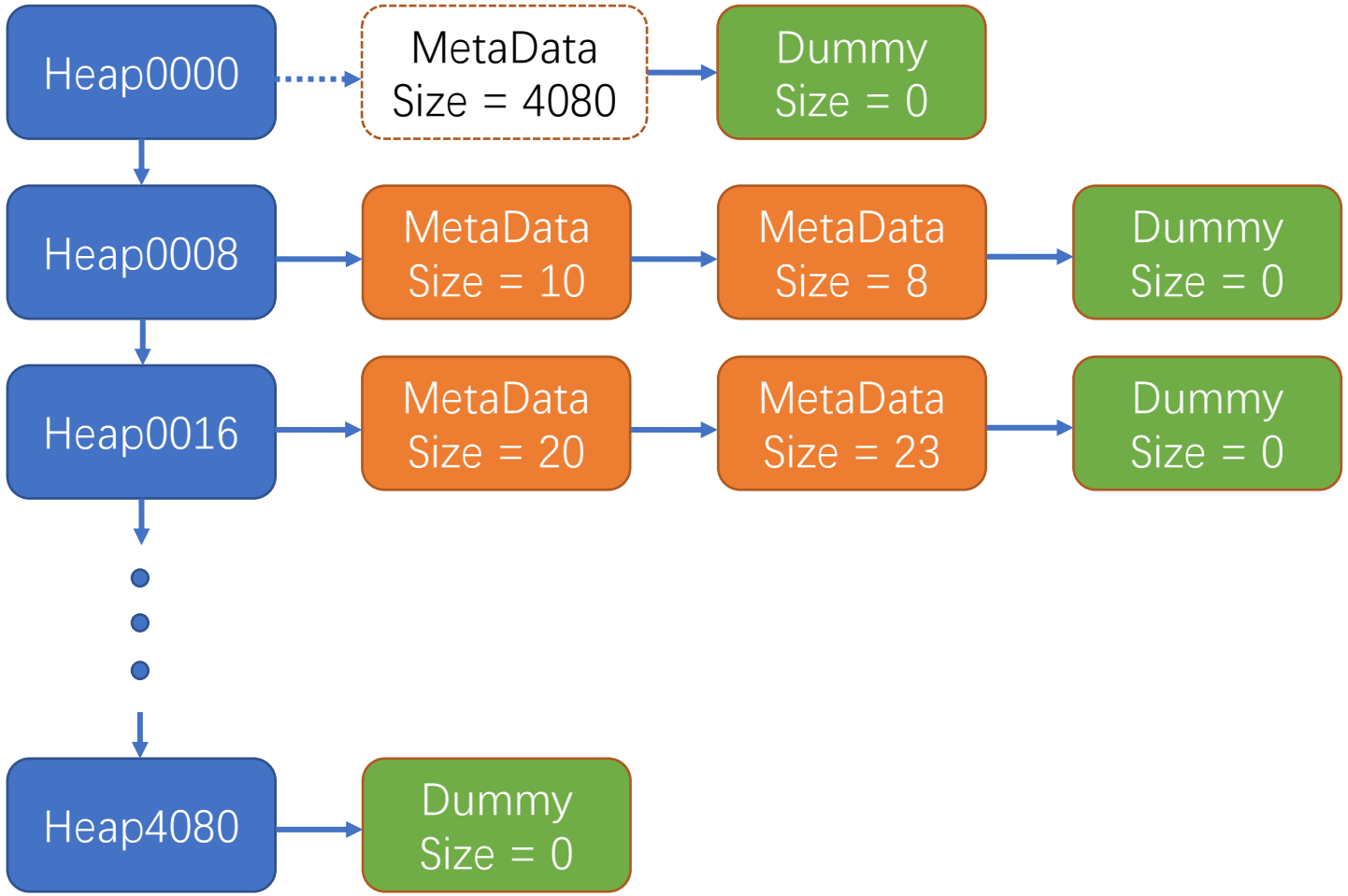
In My Malloc, heaps are linked and we only put metadata whose size > heap size and size < next heap size a into the heap.
e.x. Heap0008->Heap0016->Heap0024
Heap0008 can contain size = 8~15
Heap0016 can contain size = 16~23
**(If we allocate new memory, we put the new metadata into Heap0000)

Sizeof(Metadata) = 16
Sizeof(Heap) = 32

We have n additional heaps so we need first allocate new memory and use 32*n for them in the initialization.

n will depend on the interval

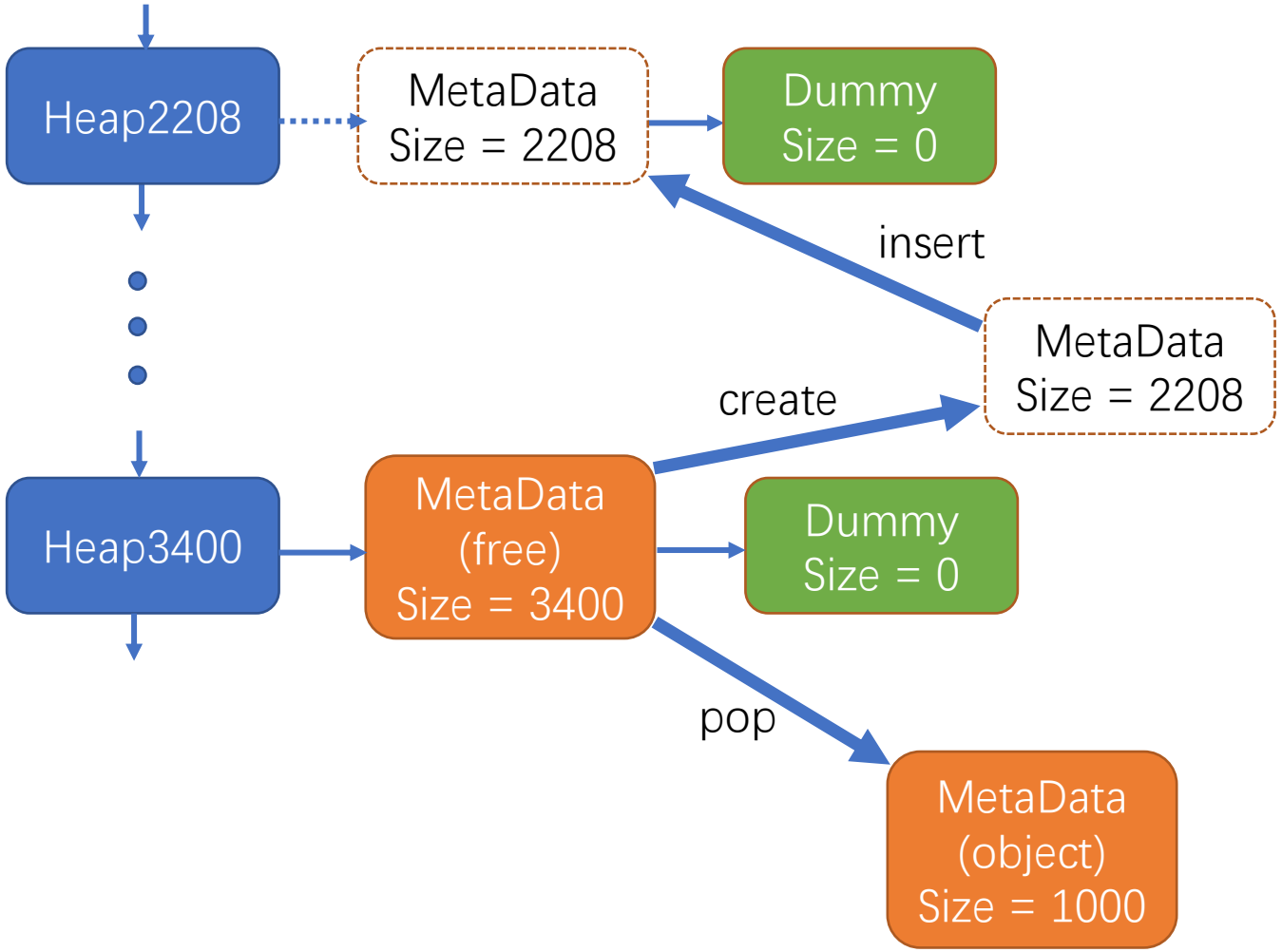
My Malloc (Malloc and allocate new)



When we malloc a space, like size = 15.
We start from Heap0000,
While (1) or (2)
(1) heap size < request size
(2) heap->next == Dummy (no MetaData)
We go to the next heap.
So when we stop, the current heap must have a MetaData and its size >= object size.

If we viewed all heaps but 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 heap.

My Malloc (Arrange object)

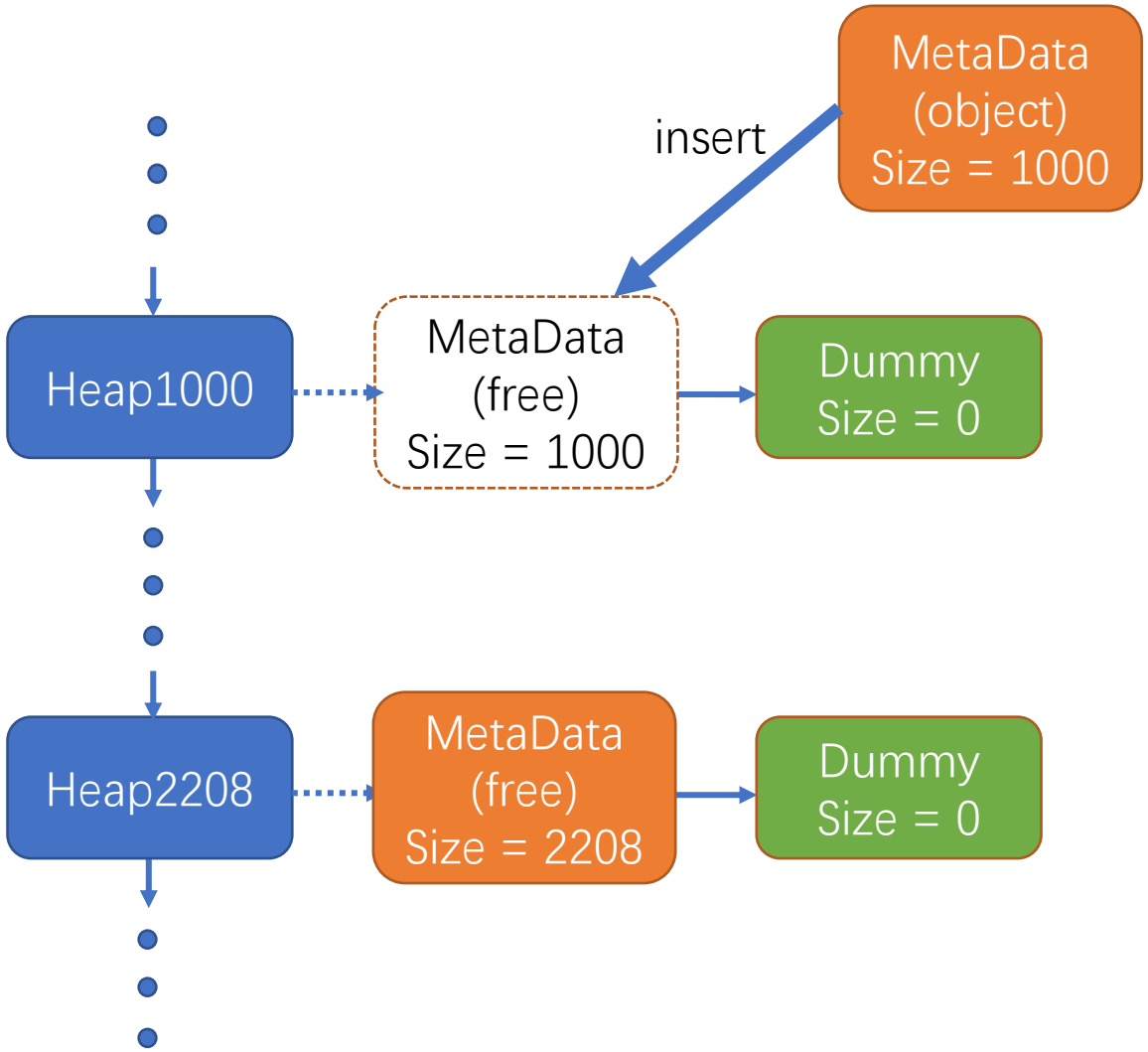


For example, when object size = 1000 and we find the proper MetaDate (size = 3400). Similar to simple malloc, we first use 1000 and the remaining size = 2400, then we use 16 for creating a new MetaData, and the rest become 2214.

Next, we will search a proper heap for 2214, that is Heap2208. 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 need to change the size, because when we free the MetaDate, its size would not change.

My Malloc (Free object)



Similar to the last page.

	Interval=1	Interval=2	Interval=4	Interval=8	Interval=16	Interval=24	Interval=32	Interval=64
C1	290ms/70%	145ms/70%	93ms/70%	54ms/70%	57ms/70%	34ms/70%	36ms/70%	43ms/70%
C2	170ms/39%	100ms/39%	58ms/39%	43ms/39%	35ms/39%	31ms/39%	28ms/39%	41ms/39%
C3	173ms/51%	102ms/51%	63ms/51%	45ms/51%	37ms/45%	43ms/17%	63ms/6%	59ms/4%
C4	986ms/72%	609ms/72%	295ms/72%	158ms/72%	98ms/71%	81ms/68%	74ms/61%	63ms/45%
C5	768ms/72%	397ms/72%	210ms/72%	124ms/72%	105ms/71%	66ms/68%	69ms/62%	58ms/46%