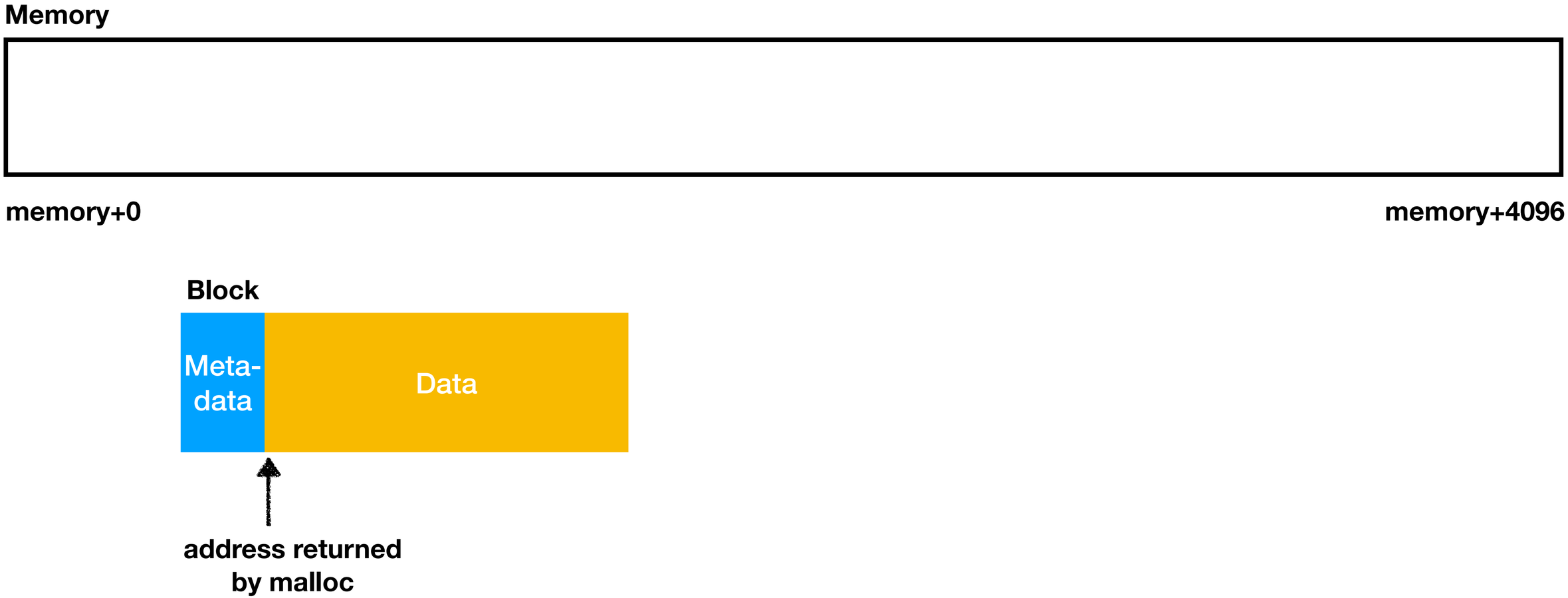
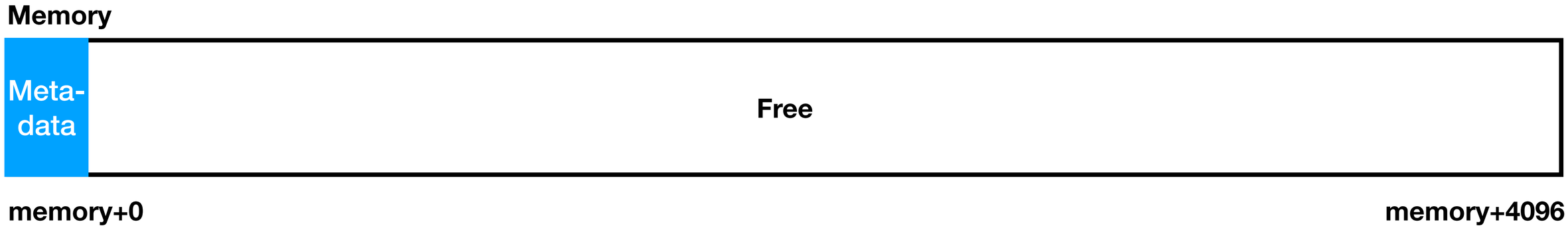
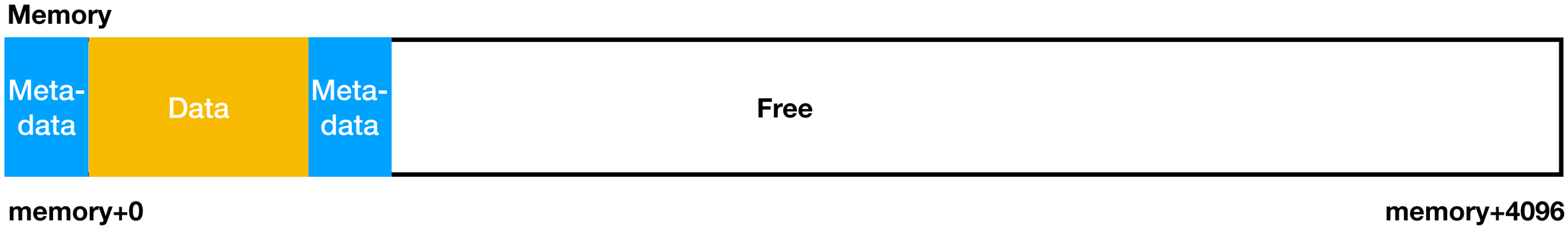

Memory
memory+0
memory+4096
Block
Meta-
data
Data
address returned
by malloc

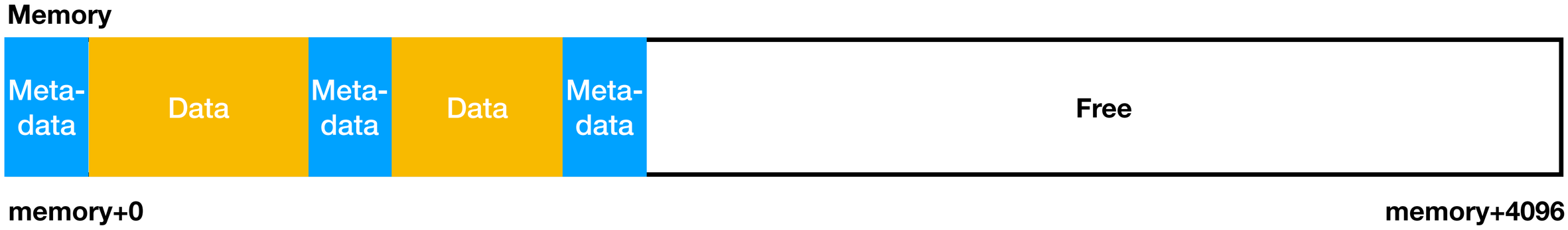
When the program starts, memory should be a single free block



Each time we allocate, we find the first available free block large enough for our request and *split* it



Each time we allocate, we find the first available free block large enough for our request and *split* it



Each time we allocate, we find the first available free block large enough for our request and *split* it

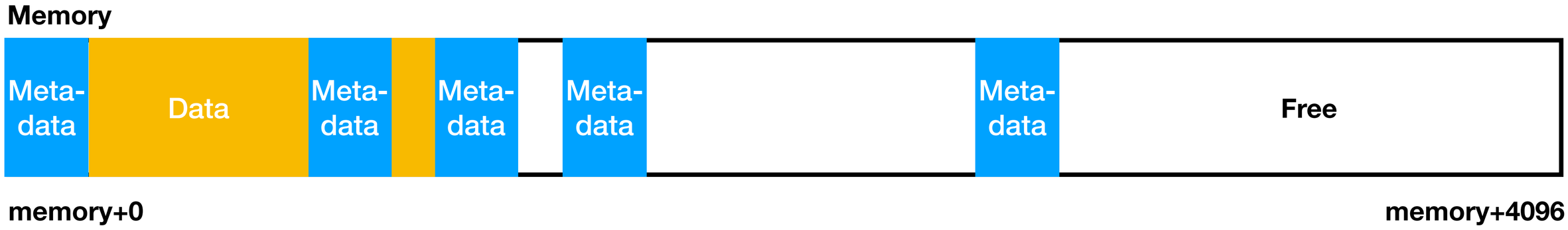
When we free a block, we mark the block as free

Again, we take the first block large enough for our request and split it

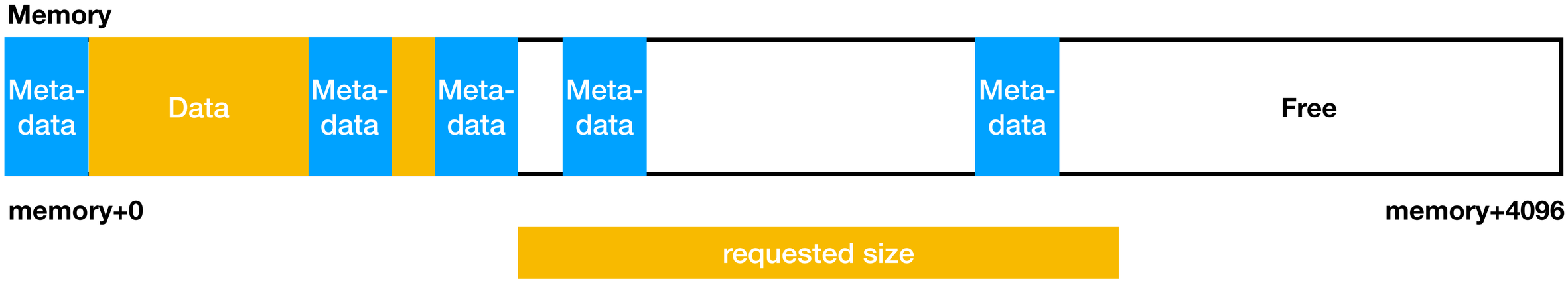
“Large enough” means we also have space for a new metadata block!

It's okay if malloc returns a region larger than requested, but malloc cannot return one smaller than requested

How should we handle consecutive free blocks?



How should we handle consecutive free blocks?



If we just mark blocks unused, we may have a situation where we cannot find a large enough block, even though there is clearly enough contiguous free memory

At some point, we need to *coalesce* adjacent free blocks

How should we handle consecutive free blocks?

If we just mark blocks unused, we may have a situation where we cannot find a large enough block, even though there is clearly enough contiguous free memory

At some point, we need to *coalesce* adjacent free blocks

How should we handle consecutive free blocks?

If we just mark blocks unused, we may have a situation where we cannot find a large enough block, even though there is clearly enough contiguous free memory

At some point, we need to *coalesce* adjacent free blocks

This could be done by malloc or free; it's up to you

What information do we need in the metadata?

Short answer: everything we need to implement malloc and free, and no more

Possibilities:
block status (free/used)
block length
pointer to next block
pointer to previous block

But maybe we don't need the block length *and* the pointer, since blocks are always contiguous

Real implementations use the previous block pointer to avoid traversing the entire list,
but if you always traverse the whole list anyway, you don't need it

Questions to Consider

- The argument to free must always be the address of the start of a block
  - Your free needs to detect incorrect pointers and report them
  - How could you check whether an address is valid?
- Your own implementations of malloc and free must be *memory-safe* and never go past the end of your array
  - Do you need to explicitly indicate that some block is the last block?

Reminders

- The real malloc and free have to store all their metadata in memory, along with the data reserved by their callers
  - Any information you store must be in the array (local variables are okay, but anything global must be in the array)
- Arrays are a lie! You may say you are allocating 4096 chars, but it's really 4096 *bytes* that you can use to store anything — in the end, everything is bytes!
  - static char memory[4096];
  - struct header *p = (struct header *) memory; // fine!
  - p = p->next; // makes sense!