Free-Space Management

Chapter 17

Previously in CS212…

- We solved internal fragmentation (space between stack and heap) with the concept of segments.
  - But we now caused external fragmentation and wasted space in between segments

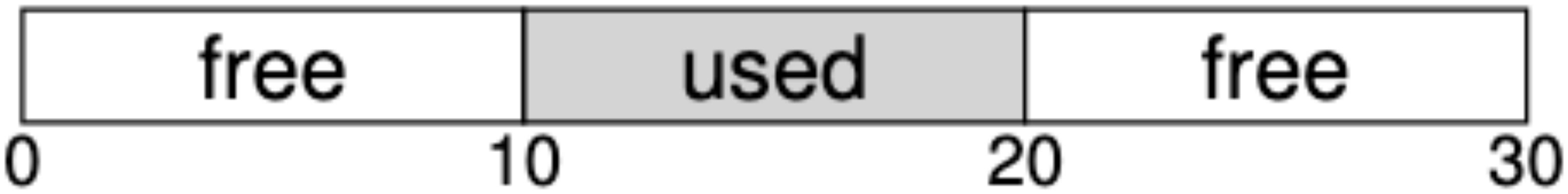
- Discussed how we can expand the base and bounds registers to accommodate segments
  - Changes this makes to the way we translate virtual addresses to physical addresses

- But we really haven't talked about how we decide where things go in memory and how we keep track…

Recap Segmentation and Fragmentation

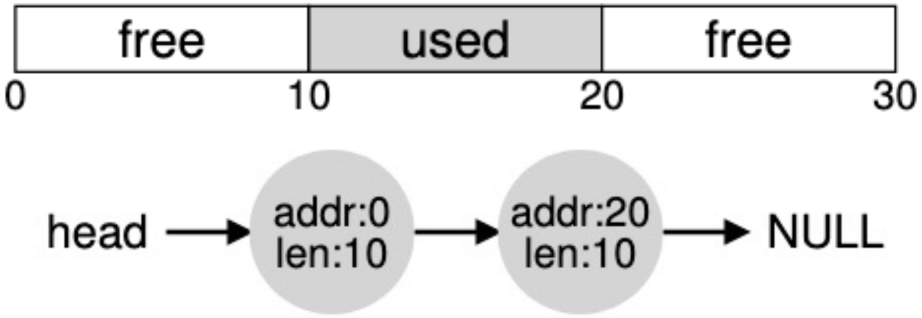
- Segmentation allows for reserved memory to be of varying sizes

- This means that sometimes while the total amount of free space in memory would allow for a segment to be created, there are no contiguous sections large enough fit the segment (**external fragmentation**)



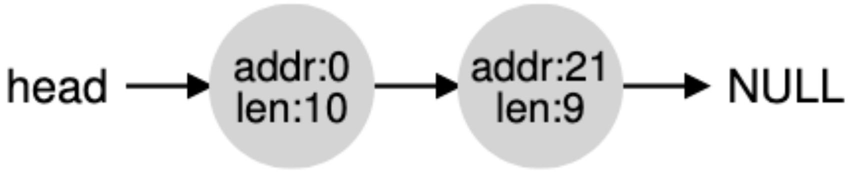
- While I 20K free total, I only have two 10K sections I can use

Splitting and Coalescing



- **Splitting**
  - If the free list has some space to accommodate a smaller memory request, it will divide up a larger space.
  - A one-byte request filled by the second free space changes the free list

Splitting

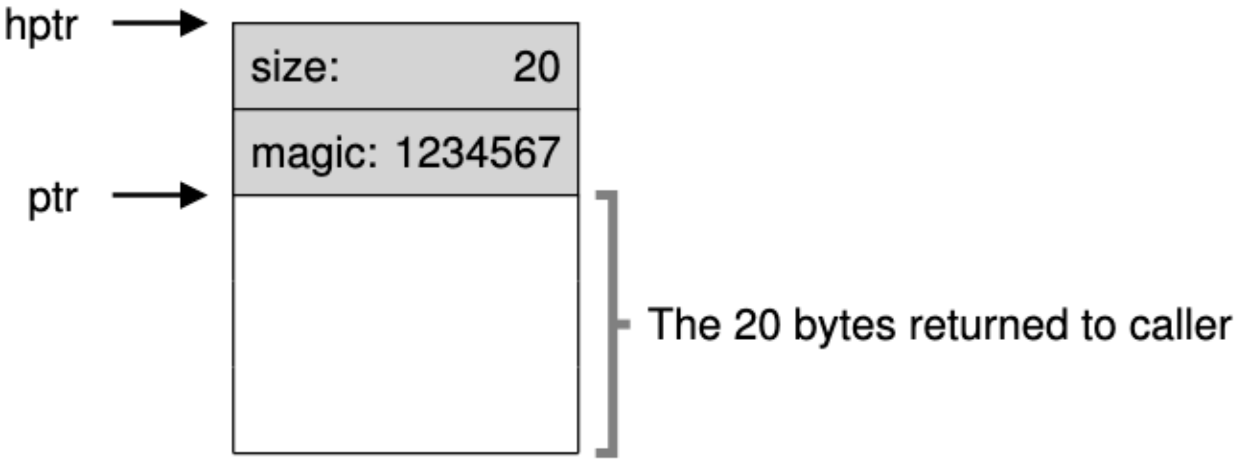


- **Coalescing**
  - Merge contiguous free space in the free list
  - Free the used ten bytes

Coalescing

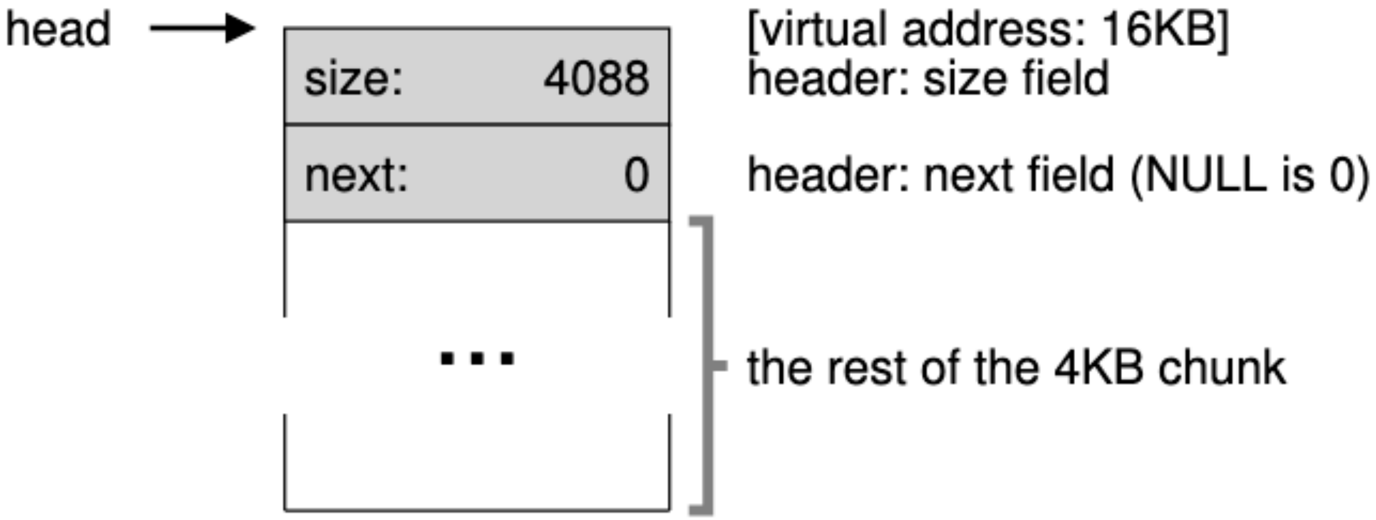
Determining Memory Size

- Malloc takes up slightly more space than requested to accommodate a **header block**
- This block (minimally) contains:
  - **Size** for quick pointer arithmetic for the used region
  - **Magic** for data integrity



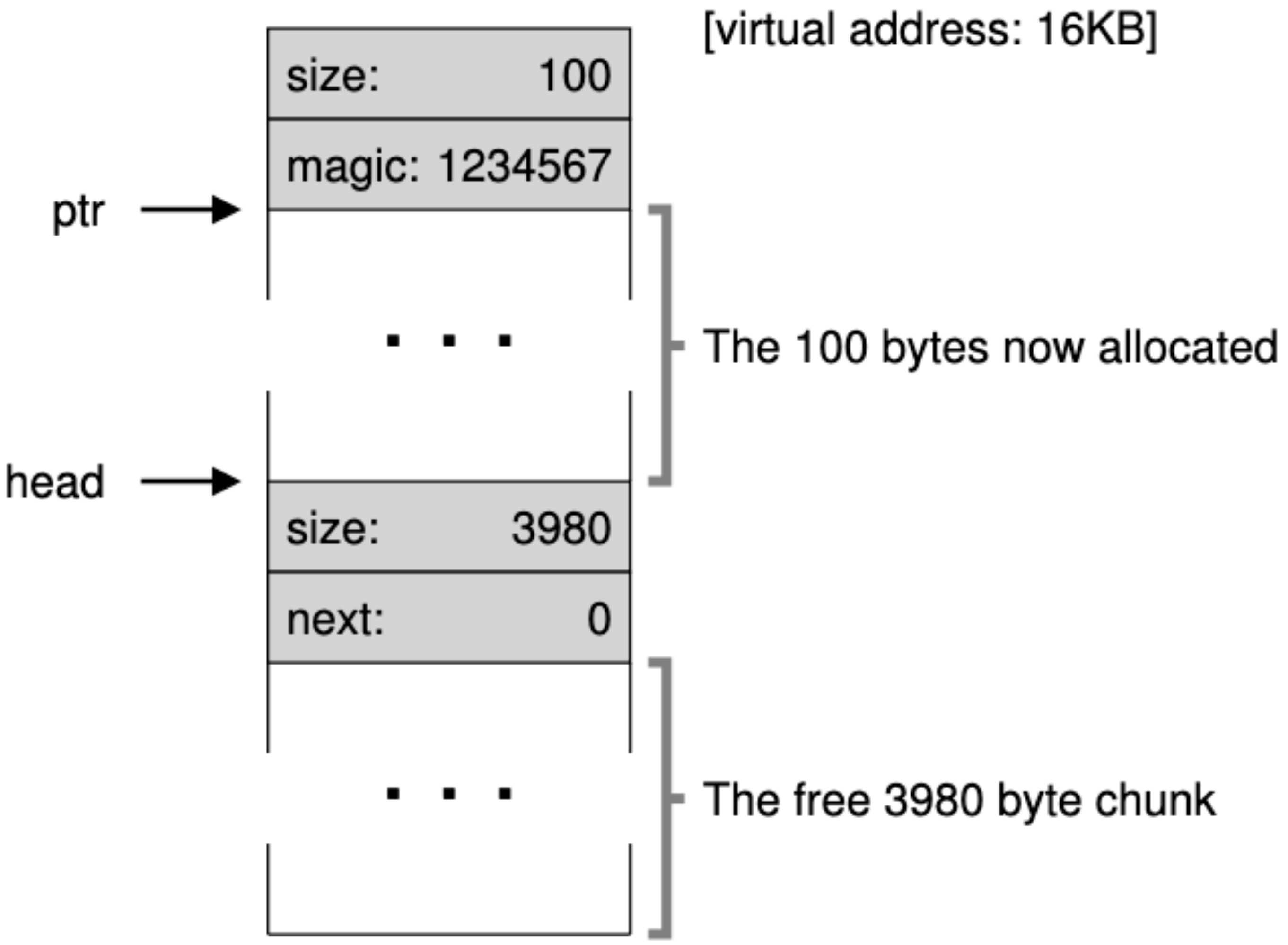
Example

- We setup a free list in the Heap
- Size is 4K (little less due to the header)
- head is the pointer for the free list



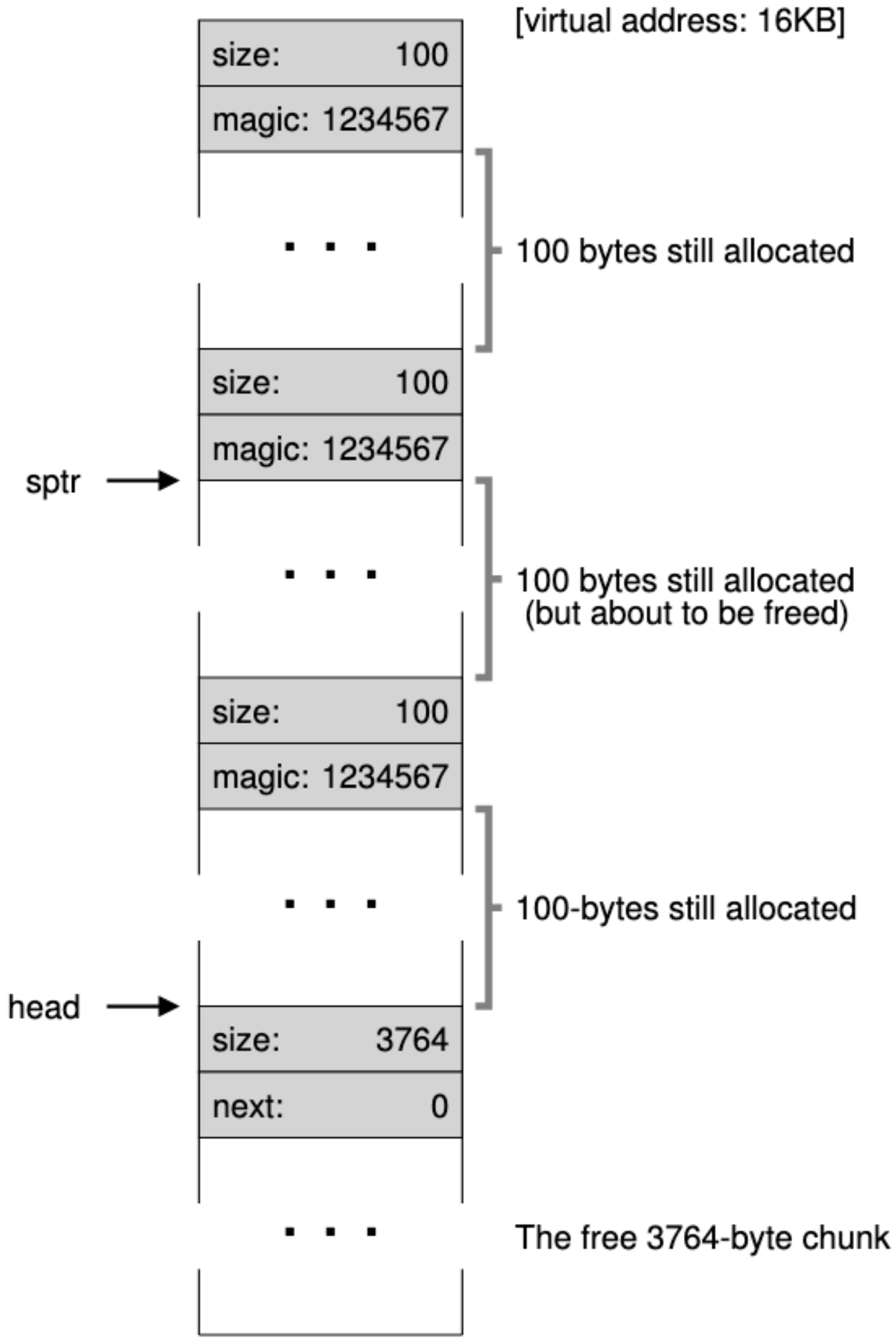
Example

- A program mallocs 100 bytes worth of data
- 8 extra bytes for the header
- We split the free space and update the size of the free list header



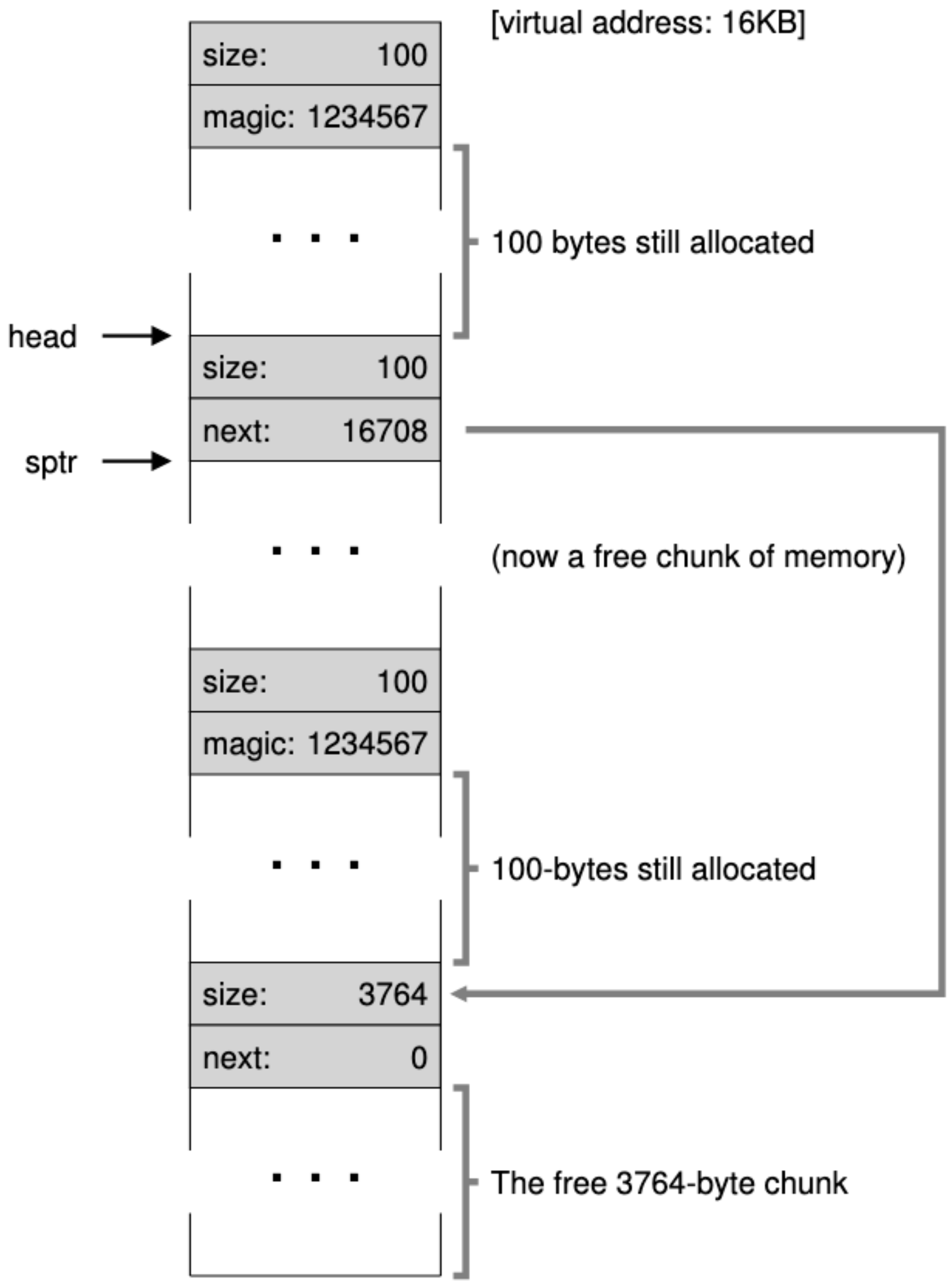
Example

- Two more requests for 100 bytes are made
- Head pointer is updated



Example

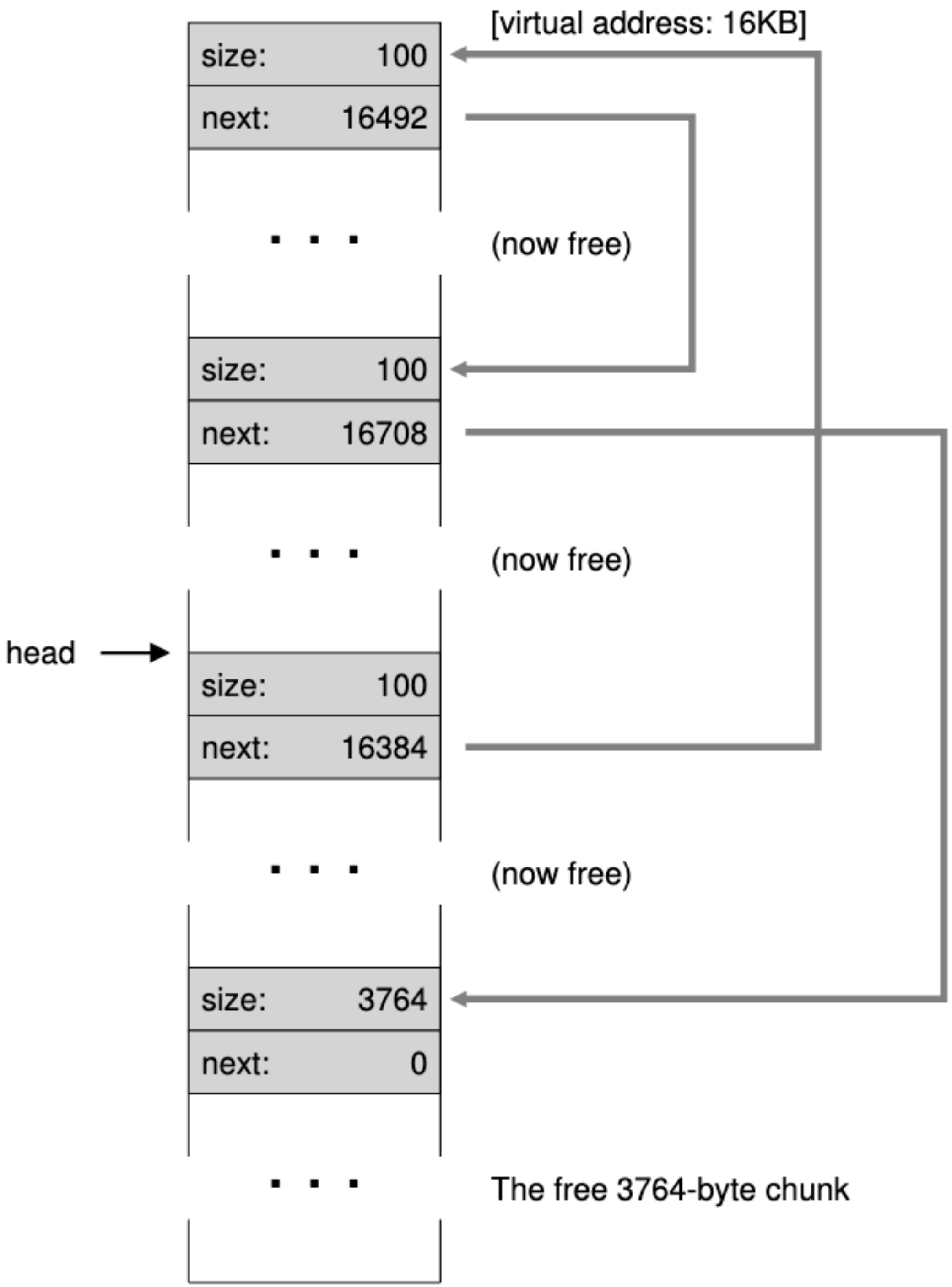
- Middle chunk is freed
- Head is moved to reference that newly free 100 bytes
- The next free space pointer is updated
- 100 bytes of external fragmentation



Example

- As the remaining memory is freed, the list pointers are updated

- Coalescing the free spaces is necessary in the free list to restore the heap's actual capacity



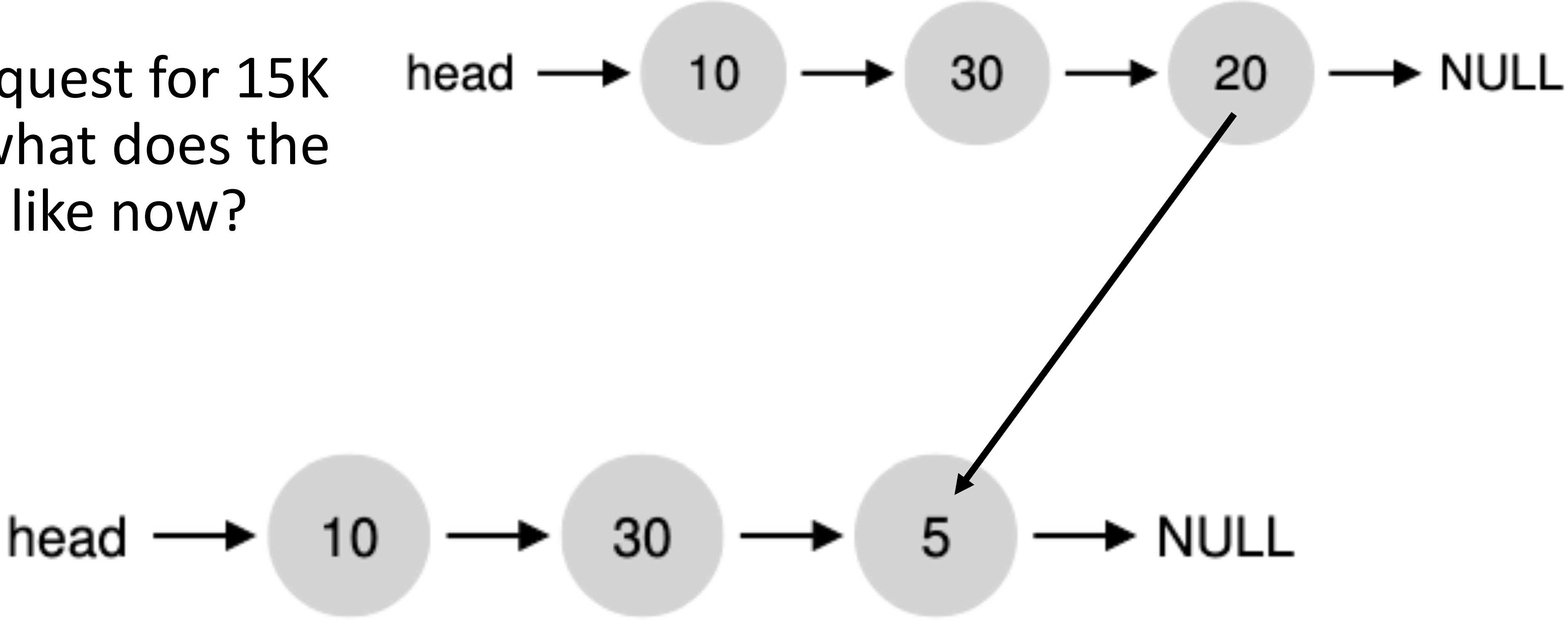
Memory Allocation Strategies

- **Best Fit**
  - Find space in free list as big or bigger than requested and return the smallest
  - Naïve approach has heavy performance penalty from searching
- **Worst Fit**
  - Find the largest chunk, split it, and return the requested amount
  - Costly search and poor performing with excess fragmentation
- **First Fit**
  - Find the first block big enough to fit and return the requested amount
  - Lower overhead, can use address-based ordering for free space to further reduce overhead and fragmentation
- **Next Fit**
  - Keep an extra pointer to in the list to where the last free space was allocated
  - Spread the free space more uniformly
  - Like first fit otherwise

Example - Best Fit

20 is the smallest space that fits

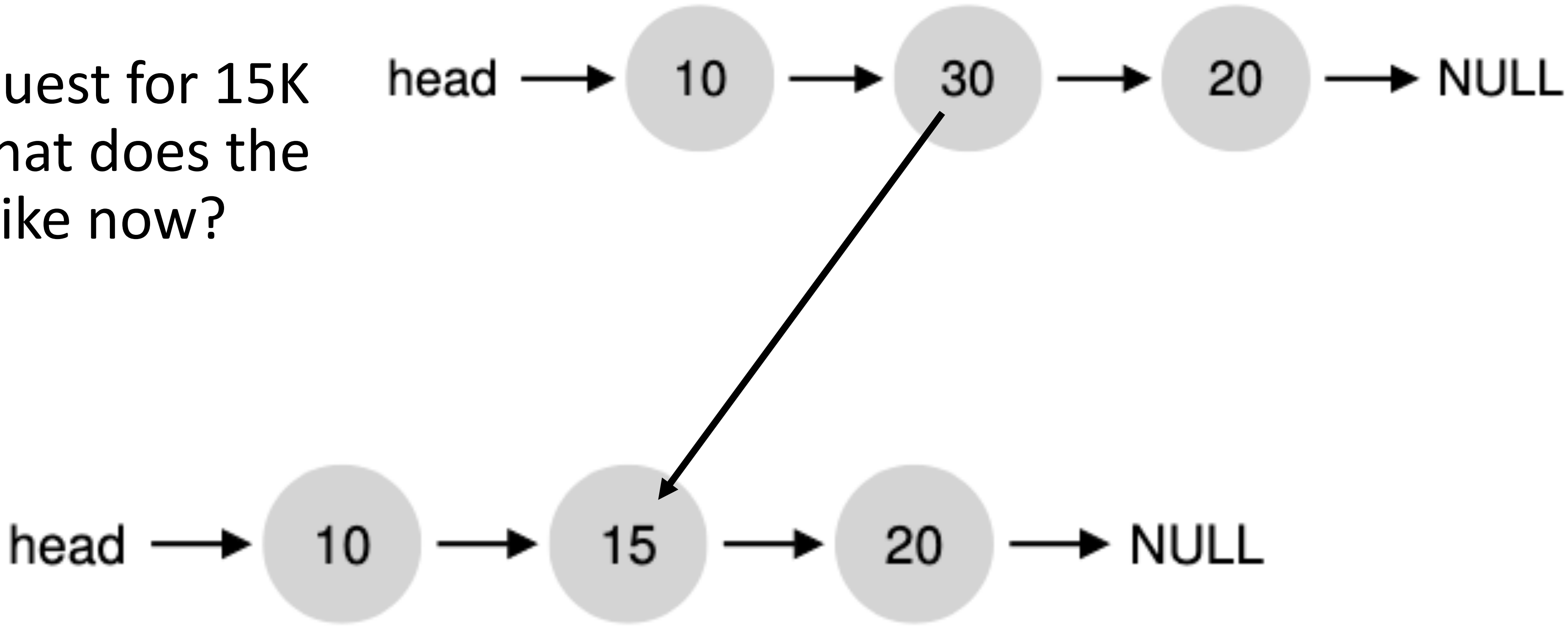
- Assume a request for 15K was made, what does the free list look like now?



Example – Worst Fit

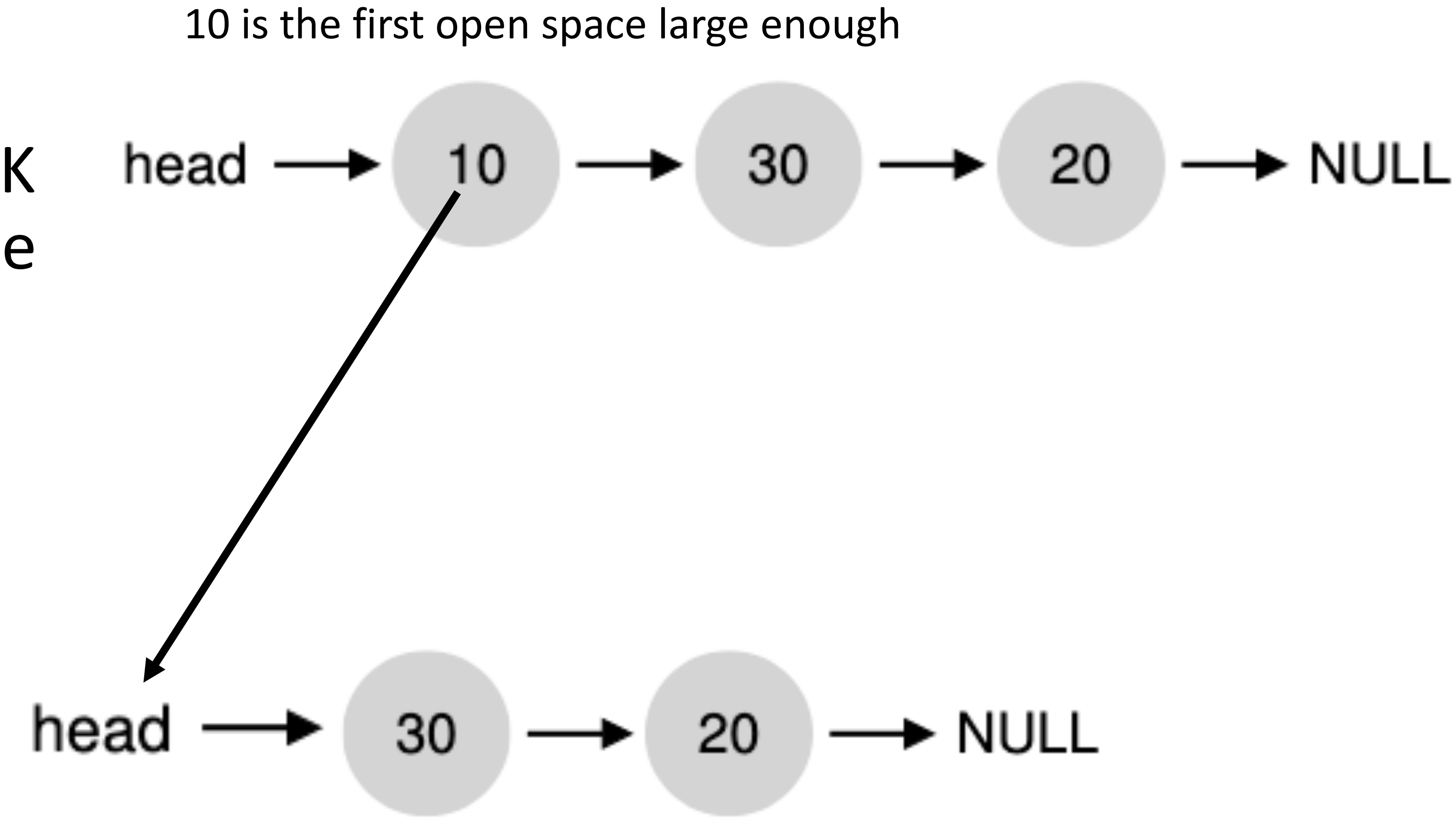
- Assume a request for 15K was made, what does the free list look like now?

30 is the largest space that fits



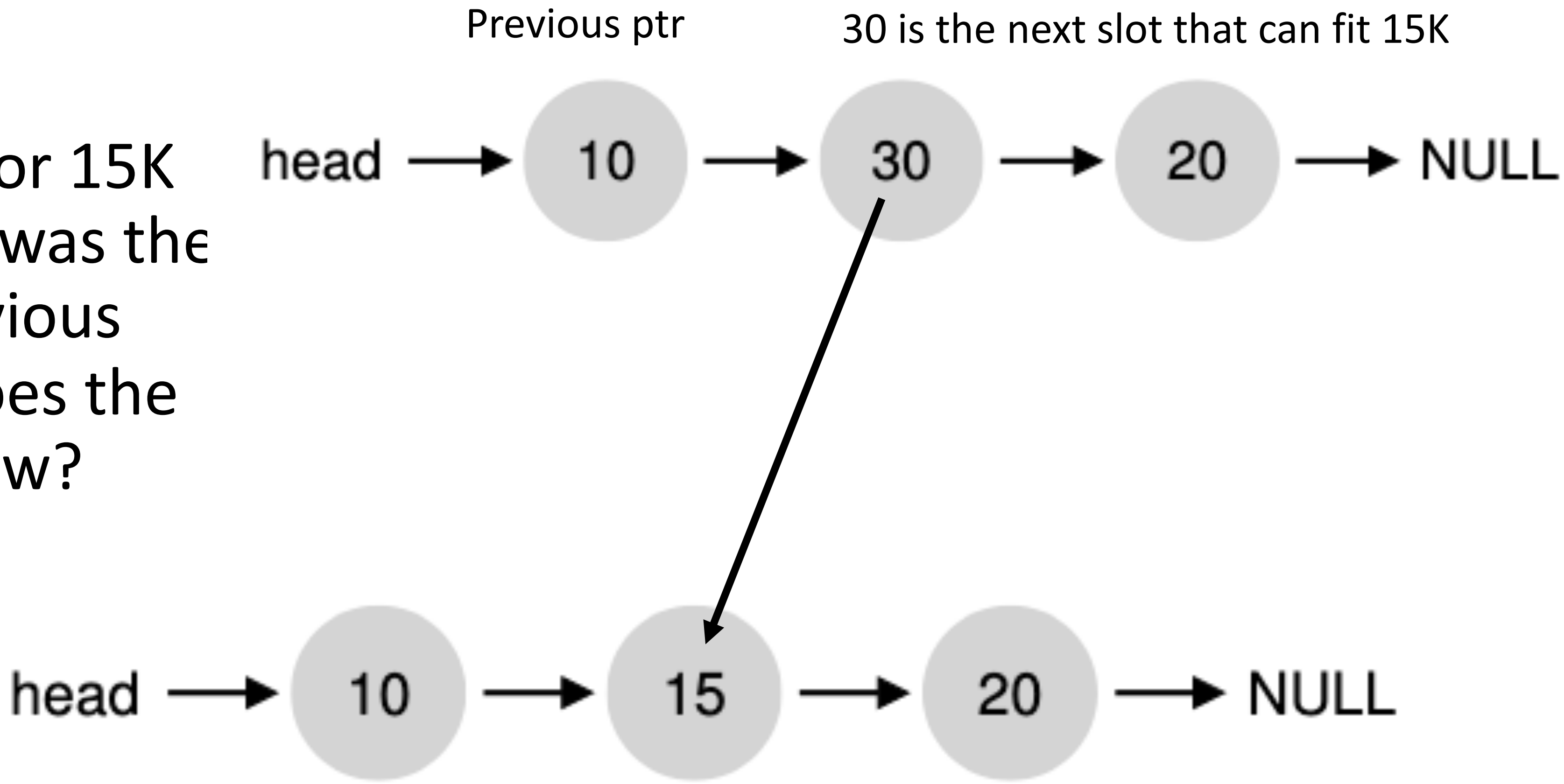
Example – First Fit

- Assume a request for 10K was made, what does the free list look like now?



Example – Next Fit

- Assume a request for 15K was made, and 10 was the location of the previous allocation. What does the free list look like now?



Segregated Lists

- Idea is to reserve a chunk of memory solely for common sized objects/requests
  - Easy to know if/where they fit, and minimize external fragmentation
- All other requests are served by a general memory allocating algorithm
- Slab allocator is an extension of this approach for storing kernel objects
  - Uses automatic reference counting (no pointers to the memory, must not be used)
  - Freed objects were preinitialized to reduce overhead.

Buddy Allocation

- When a request is made, recursively divide up free memory by two until a block that is big enough to fit is found
  - E.g. An additional division by two would be too small
  - Management of the free space is a tree
- Fixed size blocks lead to internal fragmentation
- Freeing memory is automatically coalesced by checking the "buddies" of the memory that was freed recursively and stopping when a buddy is in use

Next Time…

- We looked at some approaches to allocating memory when considering
- We saw that some better performing algorithms balance fixed size memory with some degree of flexibility
- We will look at a different model for distributing memory to our processes