CS:3620 Operating Systems

Memory Allocation Algorithms

Variable sized allocation

- Given a block of memory, how do we allocate it to satisfy various memory allocation requests?
- This problem must be solved in the C library
  - Allocates one or more pages from kernel via *brk*/*sbrk* or *mmap* system calls
  - Gives out smaller chunks to user programs via malloc
- This problem also occurs in the kernel
  - Kernel must allocate memory for its internal data structures

Variable sized allocation: headers

- Consider a simple implementation of *malloc*
- Every allocated chunk has a header with info like size of chunk
  - Why store size? We should know how much to free when *free()* is called

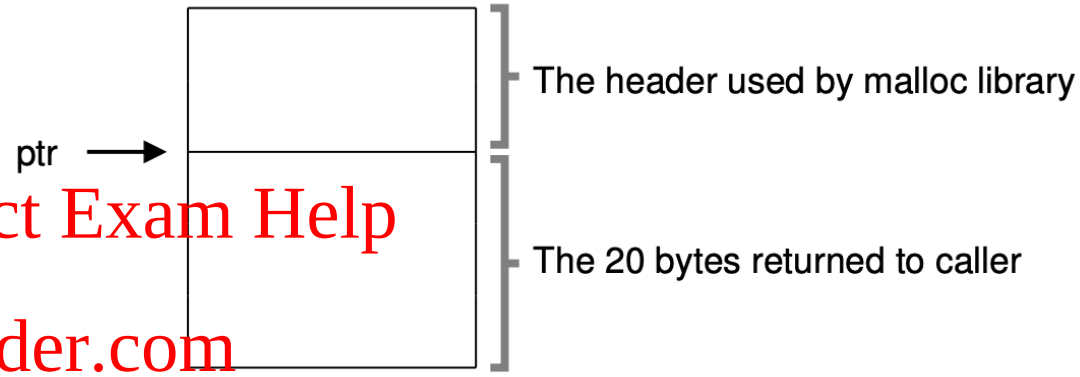


Figure 17.1: **An Allocated Region Plus Header**

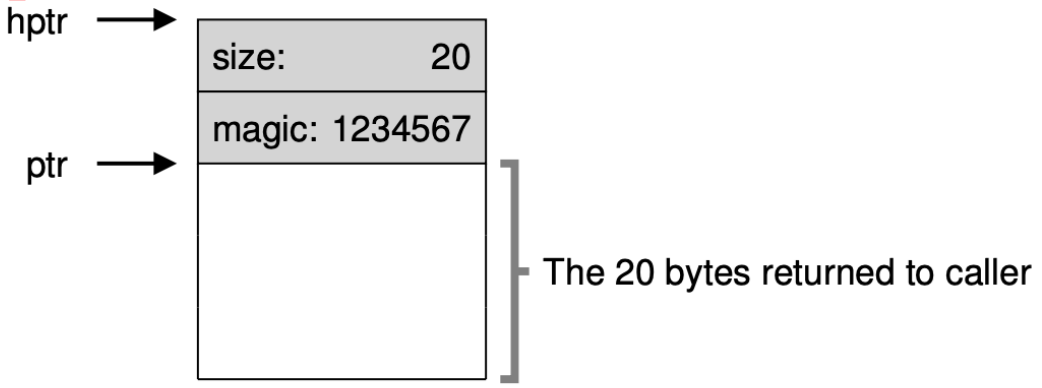


Figure 17.2: **Specific Contents Of The Header**

Free list

- Free space is managed as a list
  - Pointer to the next free chunk is embedded within the free chunk
- The library tracks the head of the list
  - Allocations happen from the head

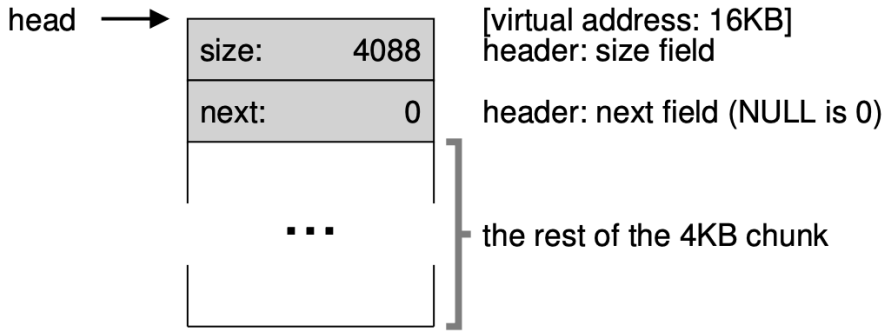


Figure 17.3: **A Heap With One Free Chunk**

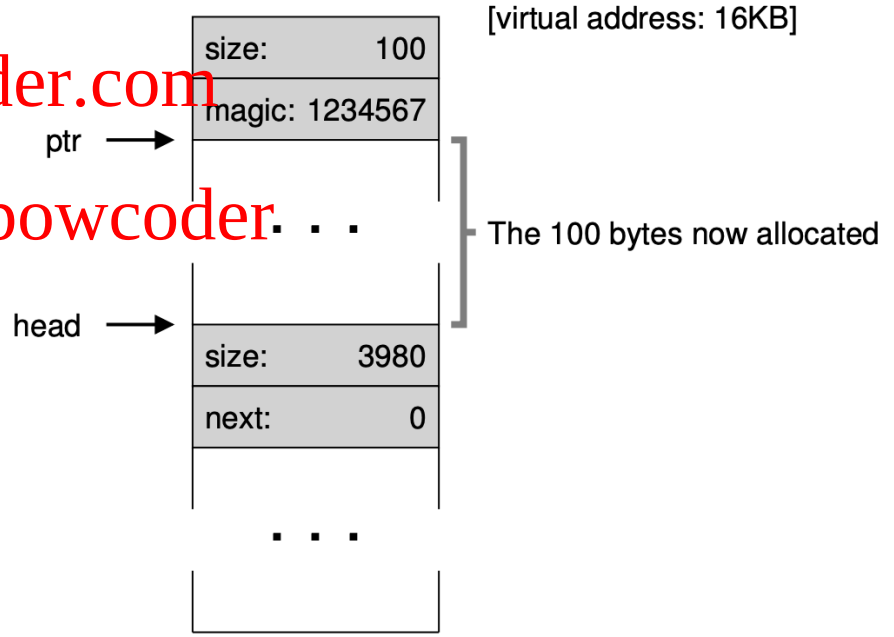
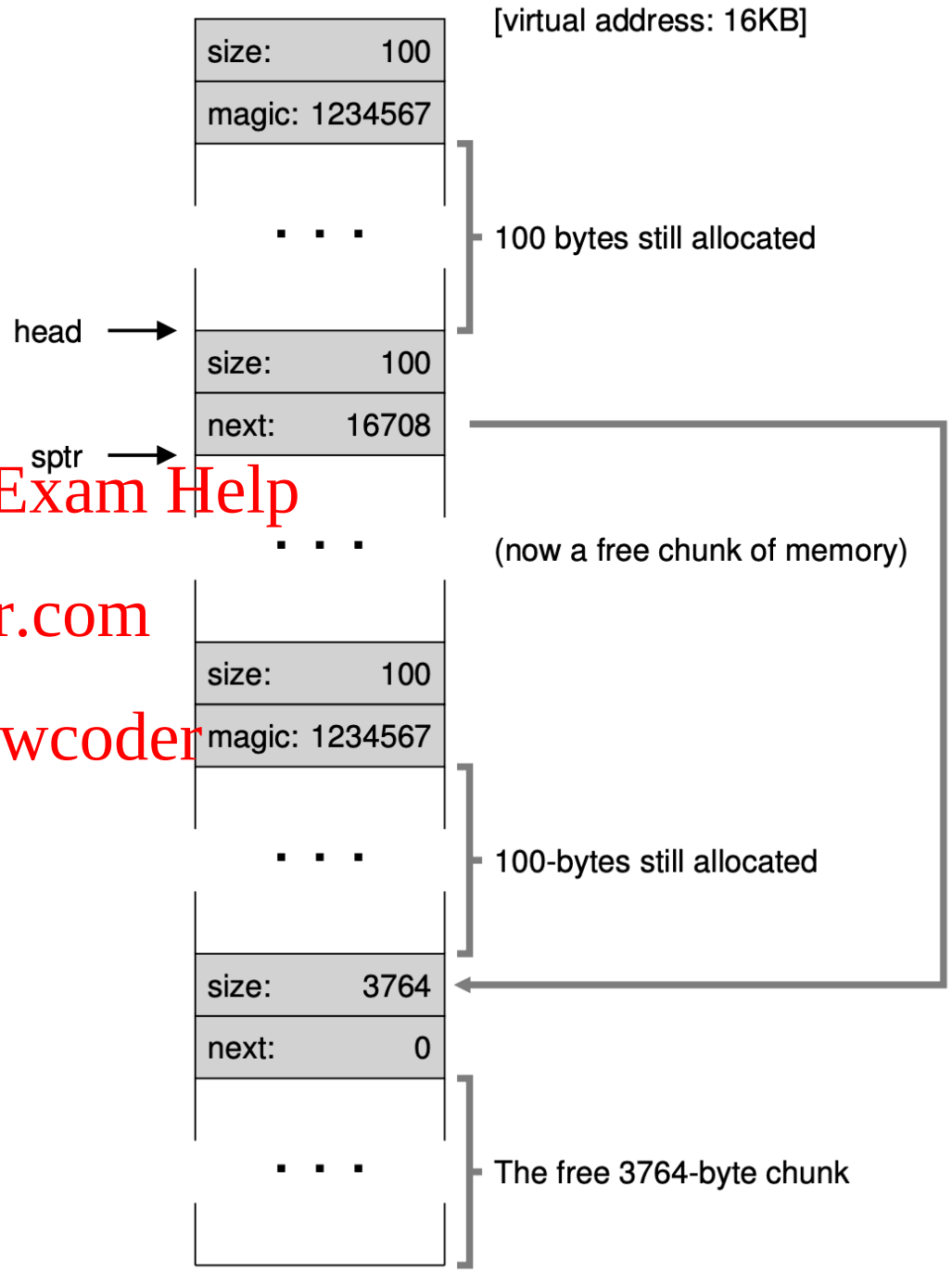


Figure 17.4: **A Heap: After One Allocation**

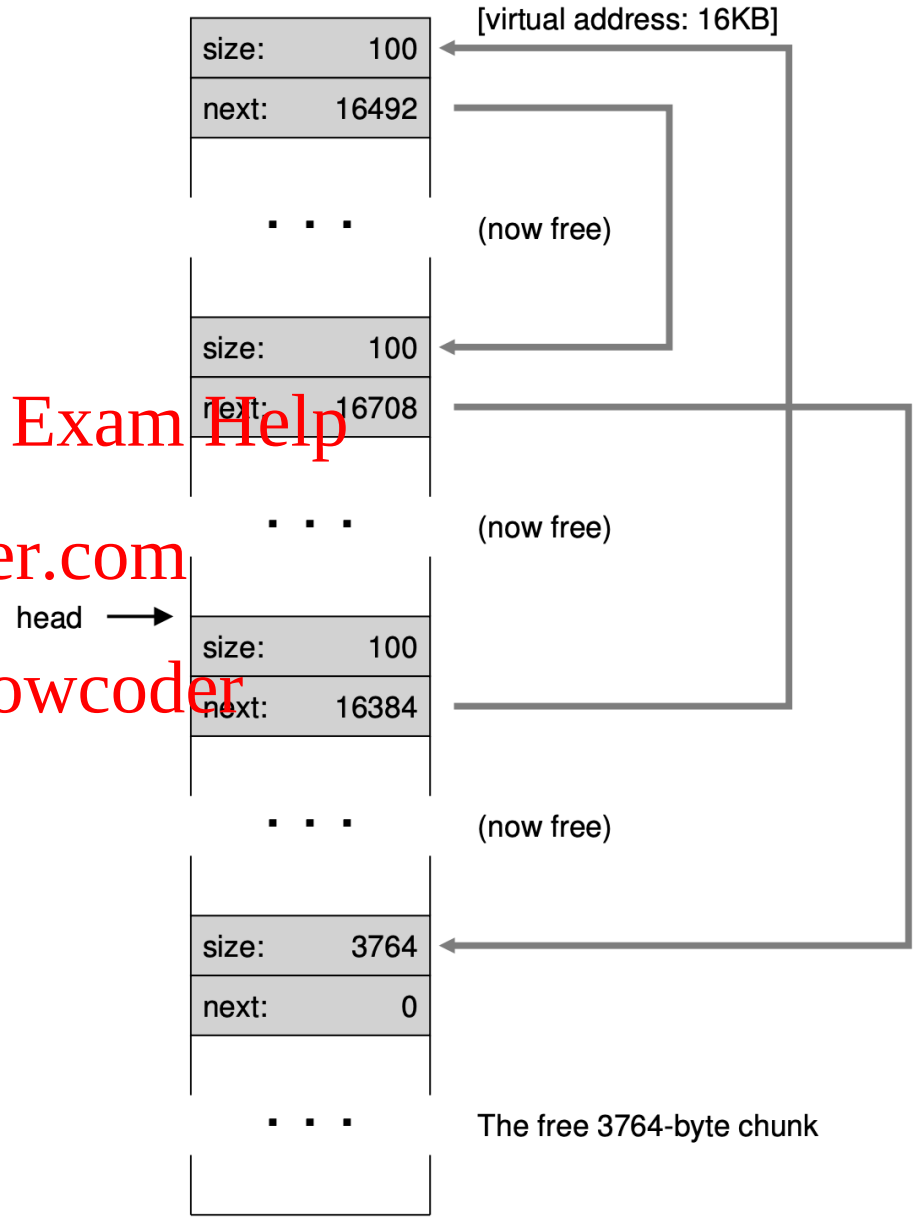
Fragmentation

- Suppose 3 allocations of size 100 bytes each happen. Then, the middle chunk pointed to by *sptr* is freed
- What is the free list?
  - It now has two non- contiguous elements
- Free space may be scattered around due to fragmentation
  - Cannot satisfy a request for 3800 bytes even though we have the free space



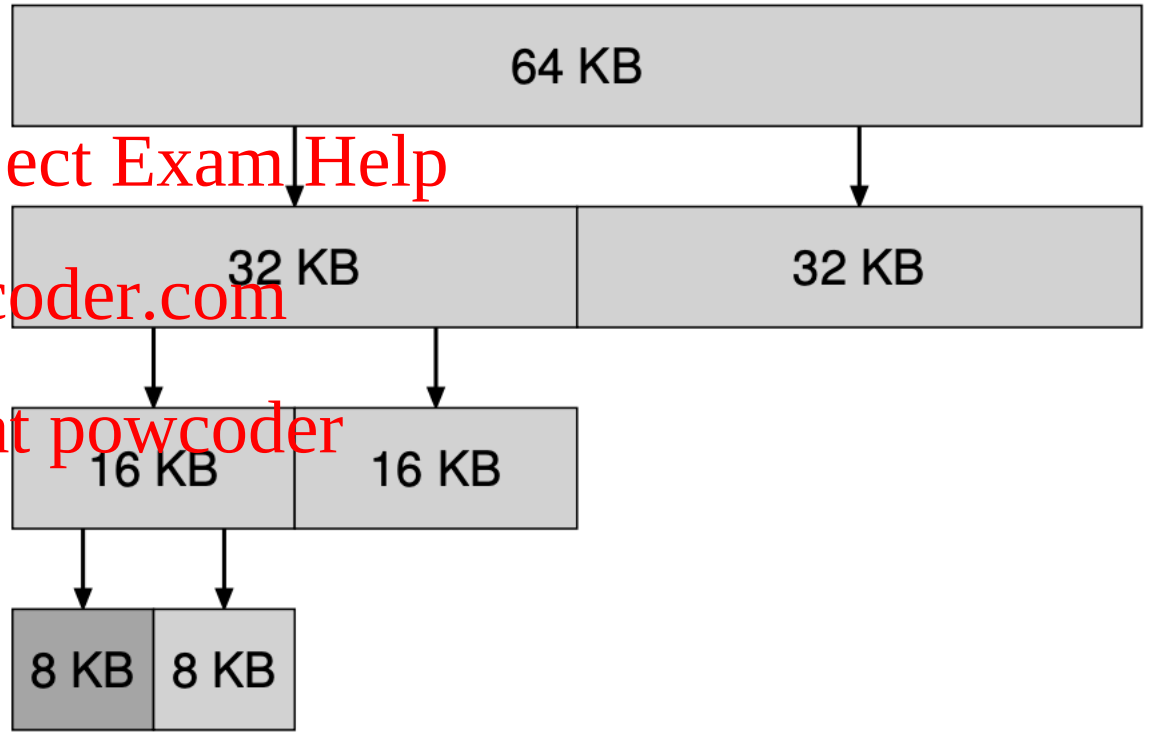
Splitting and Coalescing

- Suppose all the three chunks are freed
- The list now has a bunch of free chunks that are adjacent
- A smart algorithm would merge them all into a bigger free chunk
- Must split and coalesce free chunks to satisfy variable sized requests



Buddy allocation for easy coalescing

- Allocate memory in size of power of 2
  - E.g., for a request of 7000 bytes, allocate 8 KB chunk
- Why? 2 adjacent power-of-2 chunks can be merged to form a bigger power-of-2 chunk
  - E.g., if 8KB block and its “buddy” are free, they can form a 16KB chunk



Variable Size Allocation Strategies

- First fit: allocate first free chunk that is sufficient
- Best fit: allocate free chunk that is closest in size
- Worst fit: allocate free chunk that is farthest in size

- Example, consider this free list, and malloc(15)

- Best fit would allocate the 20-byte chunk

- Worst fit would allocate 30-byte chunk: remaining chunk is bigger and more usable

Fixed size allocations

- Memory allocation algorithms are much simpler with fixed size allocations
- Page-sized fixed allocations in kernel:
  - Has free list of pages
  - Pointer to next page stored in the free page itself
- For some smaller allocations (e.g., PCB), kernel uses a slab allocator
  - Object caches for each type (size) of objects
  - Within each cache, only fixed size allocation
  - Each cache is made up of one or more “slabs”
- Fixed size memory allocators can be used in user programs also (instead of generic malloc)

Disclaimer

- *These lecture slides are based on a slide set by Youjip Won (Hanyang University) and Mythili Vutukuru (IIT Bombay)*