ITP 30002 Operating System

Segmentation

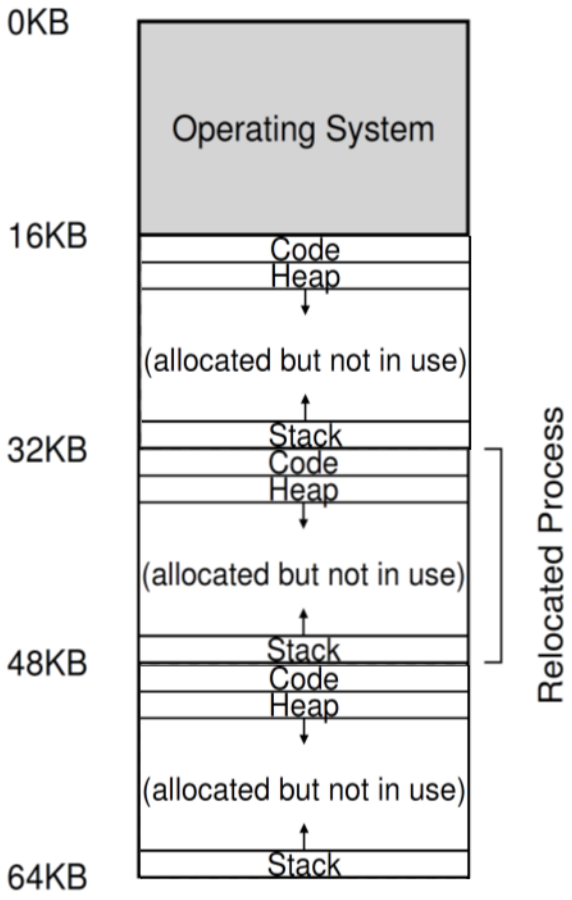
OSTEP Chapters 16 & 17

Shin Hong

Motivation

- **Internal fragmentation problem**: a process typically uses a small portion of an address space sparsely, thus, the rest of the address space remains unused and wasted

- **Redundant data**: The same piece of code may be loaded redundantly if multiple processes use it

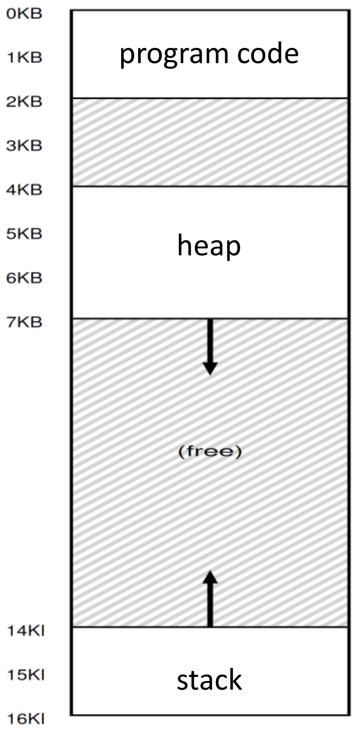


Approach

- Ideas
  1. split an address space into multiple pieces and manage each seperately
  2. allocate the memory to each piece depending on the actual needs
- Have multiple **segments** and restrict the memory access of a process to the segments only
  - a segment is a continuous memory region defined by a pair of base and bounds addresses
  - the available portion of the address space of a process can be represented as a set of segments
  - a segment is initially defined at a loading time
  - a segment can be relocated and extended/shrinked
- Let multiple processes have a same segment if the data is to be shared

Segmentation

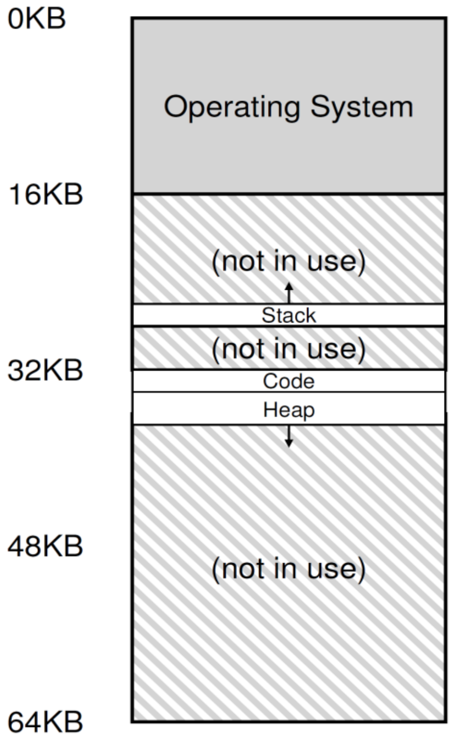
- A process has three segments, **code**, **heap** and **stack** in an address space
- MMU is required to have three base-bounds pairs, and a mechanism to identify which segment a memory access is on
  - MMU will find the base value and add it to the offset part of a virtual address
- OS can change bounds on demand of a process to extend/shrink a segment



Segment	Base	Size
Code	32K	2K
Heap	34K	3K
Stack	28K	2K

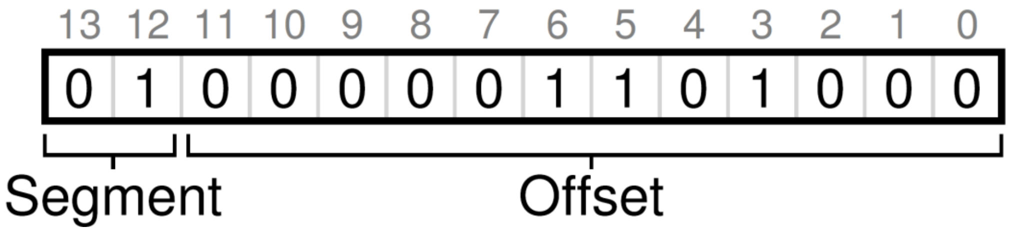
100 (0x0064) → 32K + 100 (0x8064)

4200 (0x1068) → 34K + 104 (0x8868)



Which Segment A Memory Access is On?

- By virtual address
  - use top few bits as an segmentation indicator
  - ex. virtual address 4200



00: code
01: heap
11: stack

- By instruction type
  - referring the code segment if the address is derived from PC
  - referring the stack segment if the address is derived from stack pointer
  - referring the heap segment otherwise

Segment Attributes

- A bit to indicate whether a segment grows forward or backward
  - ex.

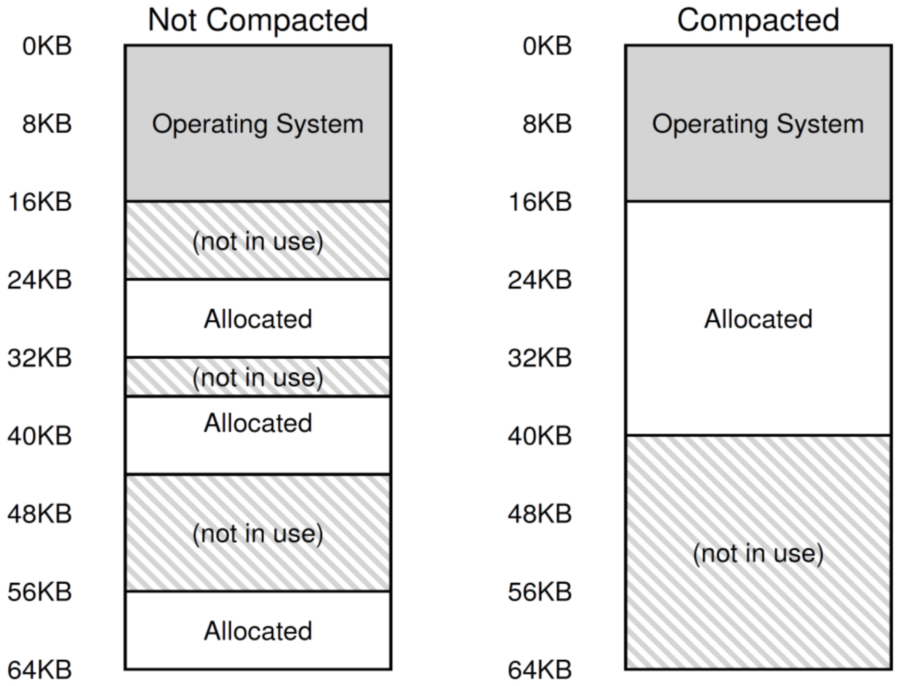
Segment	Base	Size (max 4K)	Grows Positive?
$Code_{00}$	32K	2K	1
$Heap_{01}$	34K	3K	1
$Stack_{11}$	28K	2K	0

- A bit to indicate whether a segment is for read-write or read-only
  - ex.

Segment	Base	Size (max 4K)	Grows Positive?	Protection
$Code_{00}$	32K	2K	1	Read-Execute
$Heap_{01}$	34K	3K	1	Read-Write
$Stack_{11}$	28K	2K	0	Read-Write

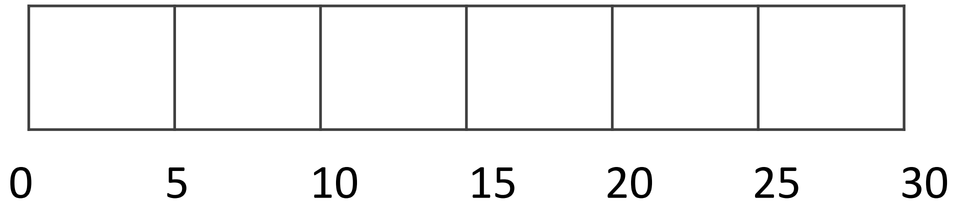
Operating System Supports

- context switching
- serving requests to grow/shrink a segment
- managing free space in physical memory to deal with external fragmentation problem
  - compaction
  - free-list management algorithms are applied



Free-Space Management

- External fragmentation hurts memory utilization
  - commonly happens for dynamic allocation of variable-length memory units
  - example. with a 30-bytes address space
    - alloc 15 bytes as A
    - alloc 10 bytes as B
    - free A (15 bytes)
    - alloc 5 bytes as C
    - alloc 15 bytes as D

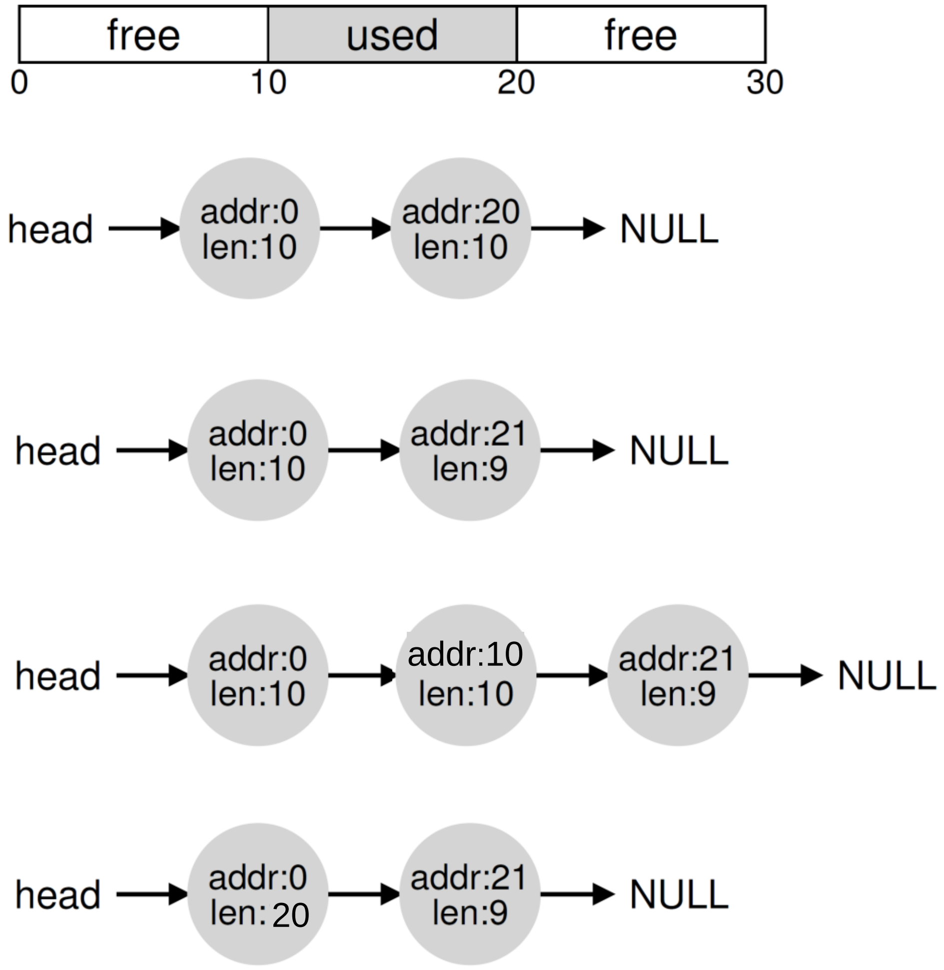


Free List

- maintains free space as a linked list of free chunks (i.e., continuous unused memory regions)

- to allocate a chunk of size *m*
  - find a node of a free chunk whose size is greater than or equal to *m*
  - split the free chunk if its size is greater than *m*

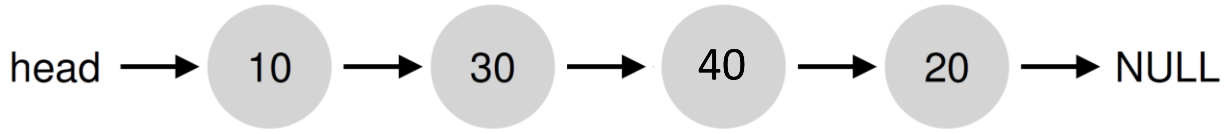
- to free an allocated memory chunk,
  - add a node of the newly freed chunk to the free list
  - merge adjacent chunks into a single larger chunk (coalescing)



Allocation Strategies

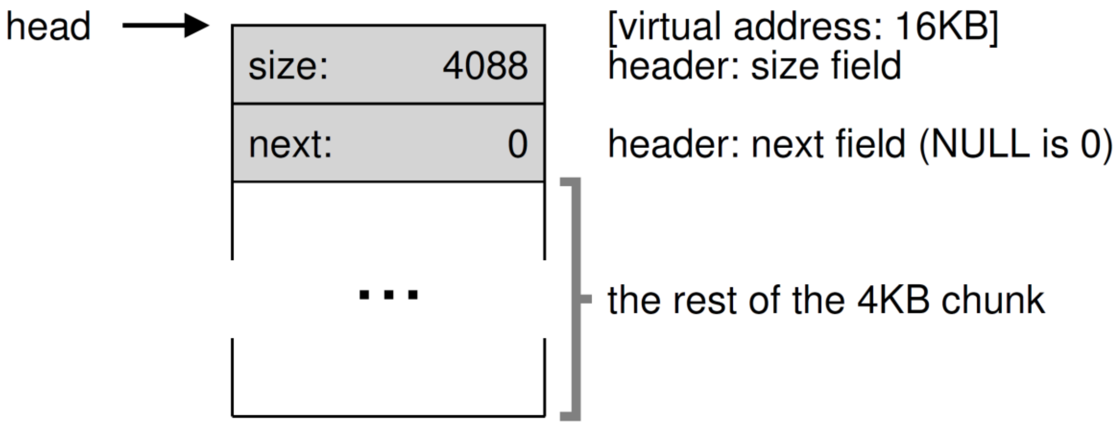
- The ideal allocator should be fast and minimize fragmentation.
- **Best fit**: search through the free list and find the smallest chunks that are large enough to afford the memory request (i.e., closest to what the user asks)
- **Worst fit**: find the largest chunk in order to keep large chunks remain in the free list
- **First fit**: find the first chunk that is large enough to afford the requested memory
- **Next fit**: conduct the first fit search from the node where the last search was stopped (not from the beginning of the free list)

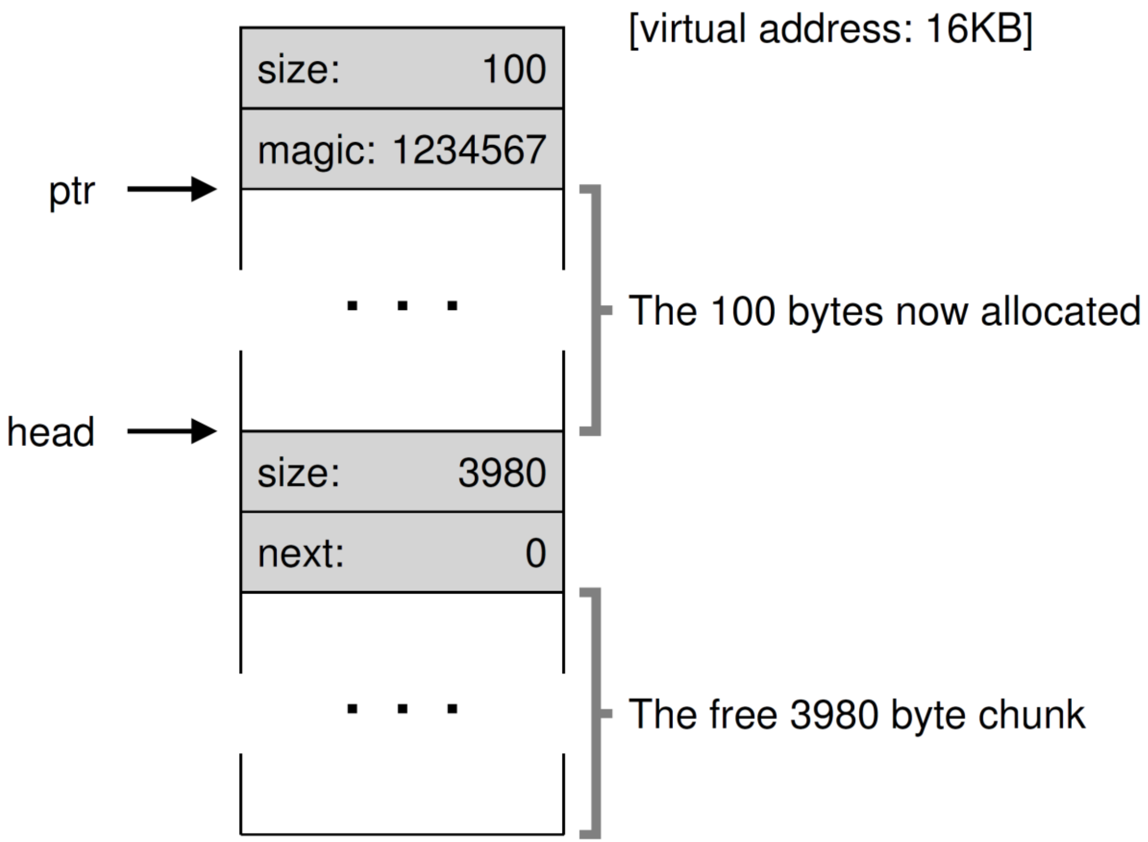
Ex. the user asks 18

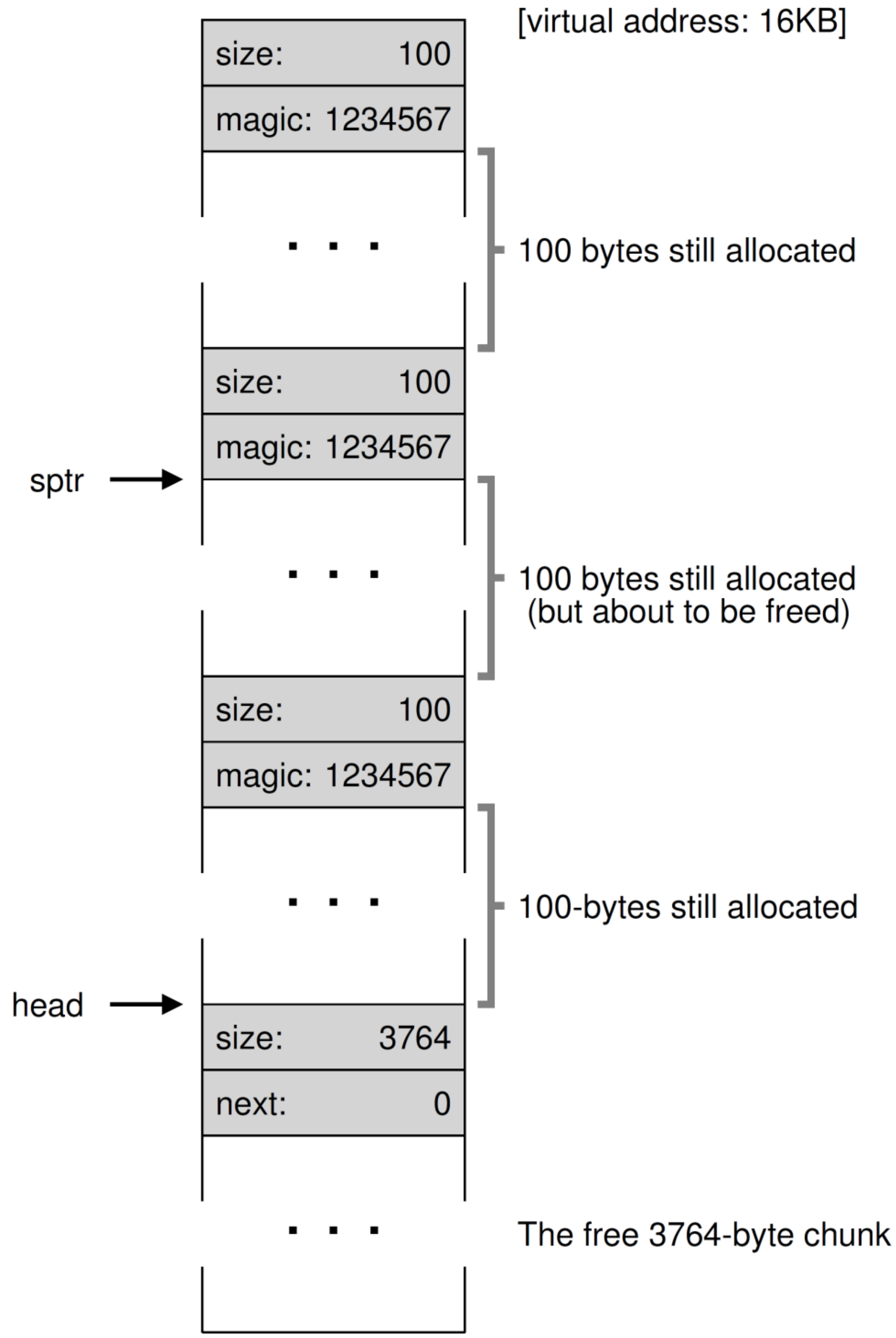


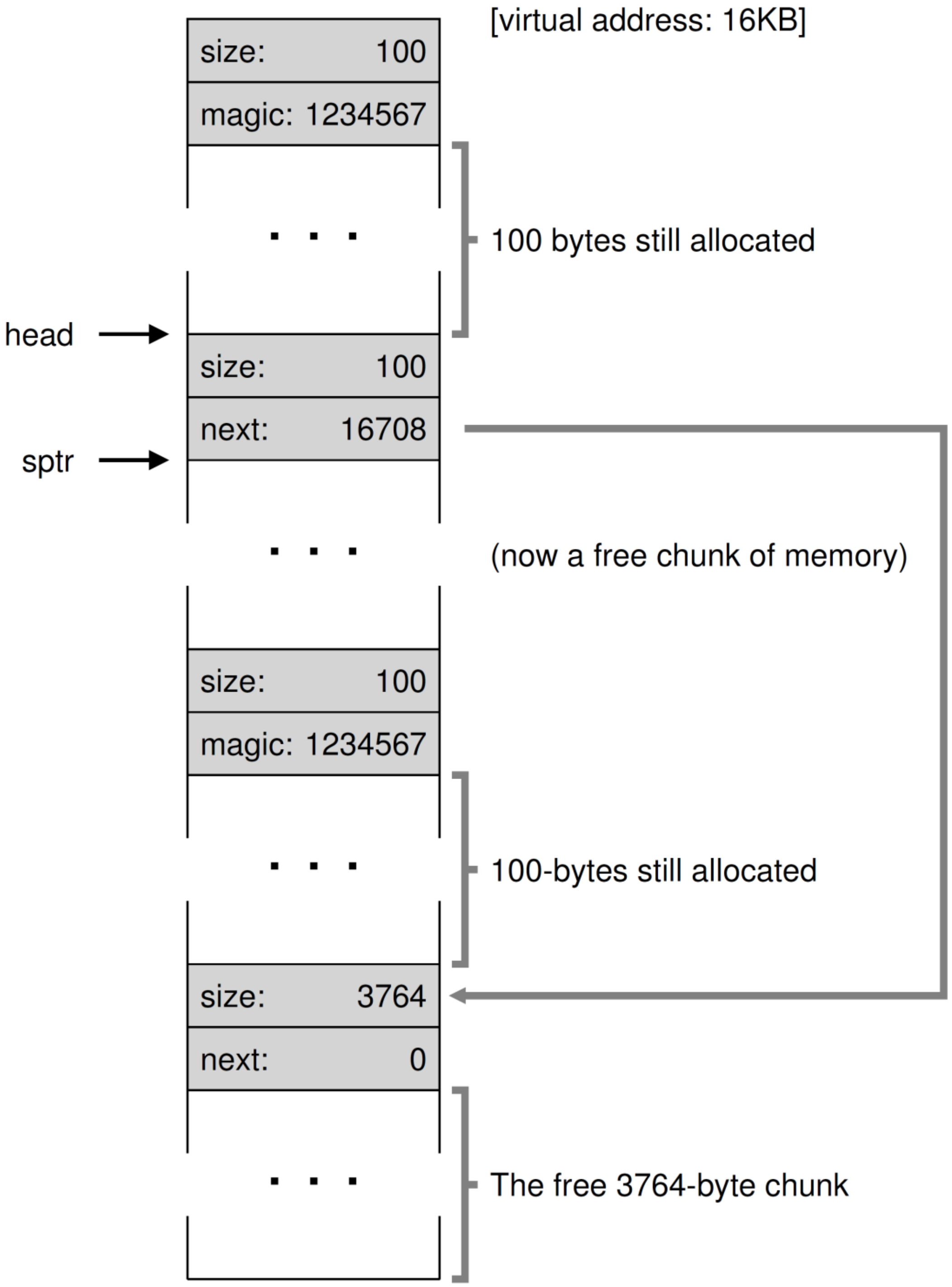
Embedding A Free List

- Example: 4096-byte memory






[virtual address: 16KB]
size: 100
magic: 1234567
100 bytes still allocated
size: 100
magic: 1234567
sptr
100 bytes still allocated
(but about to be freed)
size: 100
magic: 1234567
100-bytes still allocated
head
size: 3764
next: 0
The free 3764-byte chunk


[virtual address: 16KB]
size: 100
magic: 1234567
100 bytes still allocated
head
size: 100
next: 16708
sptr
(now a free chunk of memory)
size: 100
magic: 1234567
100-bytes still allocated
size: 3764
next: 0
The free 3764-byte chunk

on

ystem