ECE437/CS481

M07B: MEMORY MANAGEMENT

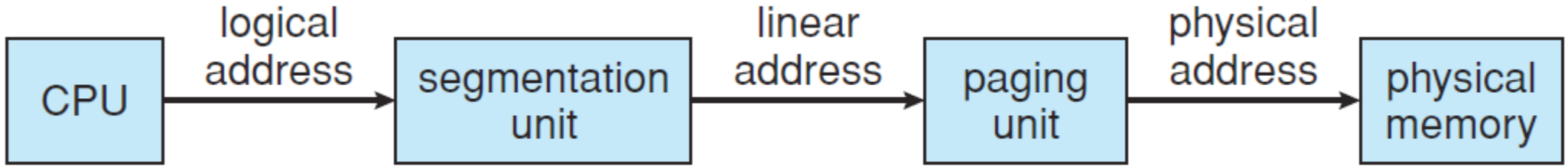
CHAPTER 8.7 & 9.8

Xiang Sun

The University of New Mexico

IA-32 architecture

- Memory management in IA-32 systems is divided into two components—segmentation and paging

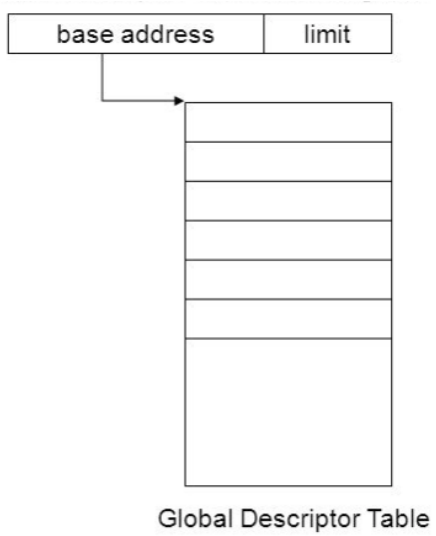


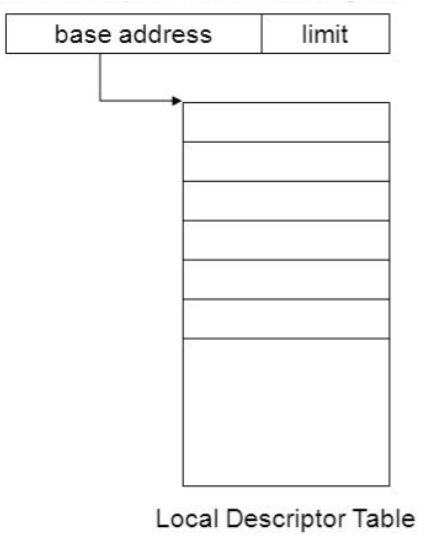
  - The CPU generates logical addresses, which are given to the segmentation unit.
  - The segmentation unit produces a linear address for each logical address.
  - The linear address is then given to the paging unit, which in turn generates the physical address in main memory.

IA-32 architecture

❑ IA-32 Segmentation (logical address → linear address)

- The logical address space of a process is divided into two partitions
  - ✓ Private partition
    - consists of up to 8K segments that are private to a process
    - information about the private partition is kept in the local descriptor table (LDT)
  - ✓ Global partition
    - consists of up to 8K segments that are shared among all the processes
    - information about the global partition is kept in the global descriptor table (GDT)
  - ✓ Each entry in the LDT/GDT consists of an 8-byte segment descriptor with information about a particular segment, including the base address and limit of the segment

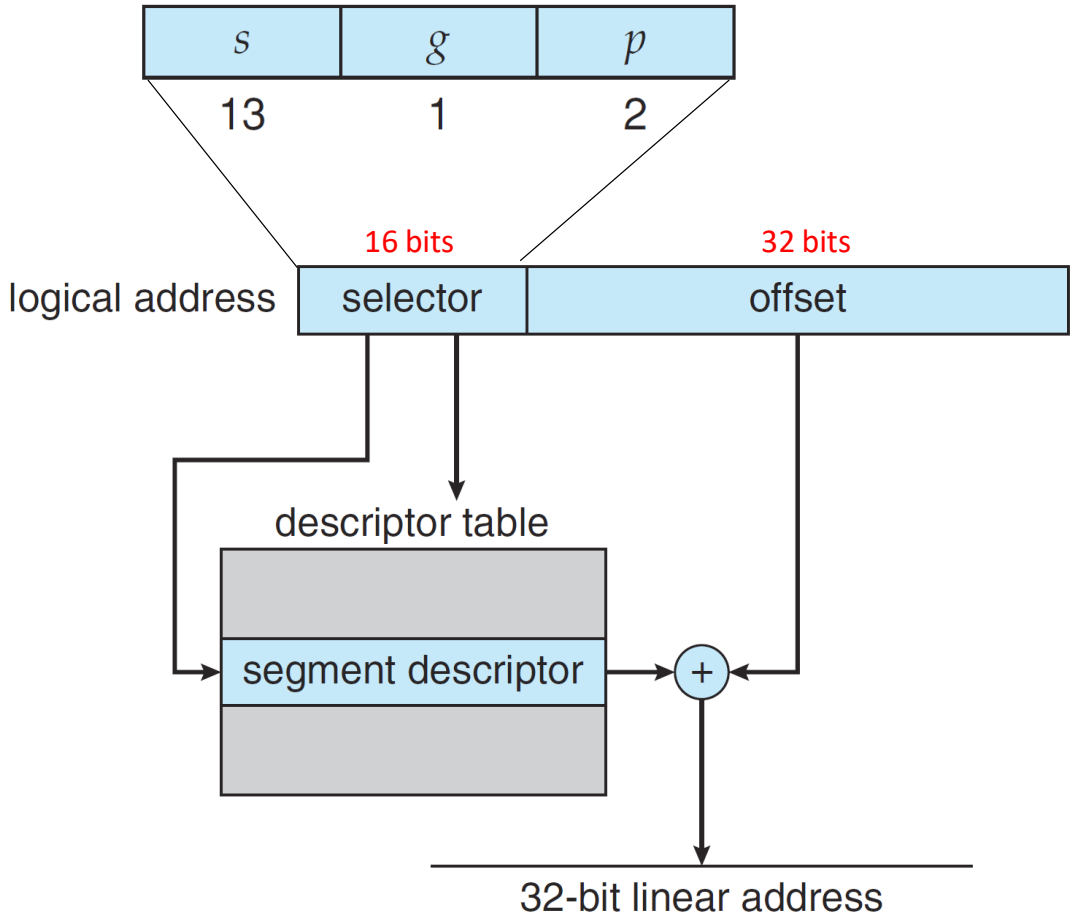




IA-32 architecture

IA-32 Segmentation

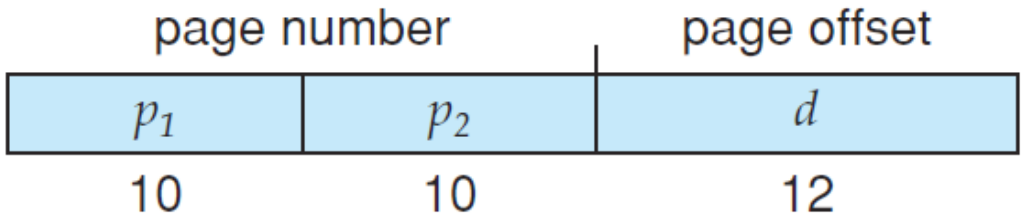
- logical address is a pair <selector, offset>, where the selector is a 16-bit number
  - ✓ s designates the segment number
  - ✓ g indicates whether the segment is in the GDT or LDT
  - ✓ p deals with protection
  - ✓ The segment number in the logical address points to the appropriate entry (segment descriptor) in the LDT/GDT
  - ✓ The base and limit information in the segment descriptor are used to generate a 32-bit linear address



IA-32 architecture

IA-32 Paging

- The IA-32 architecture allows a page size of either 4 KB or 4 MB
- For 4KB page size setup, IA-32 uses a two-level paging scheme



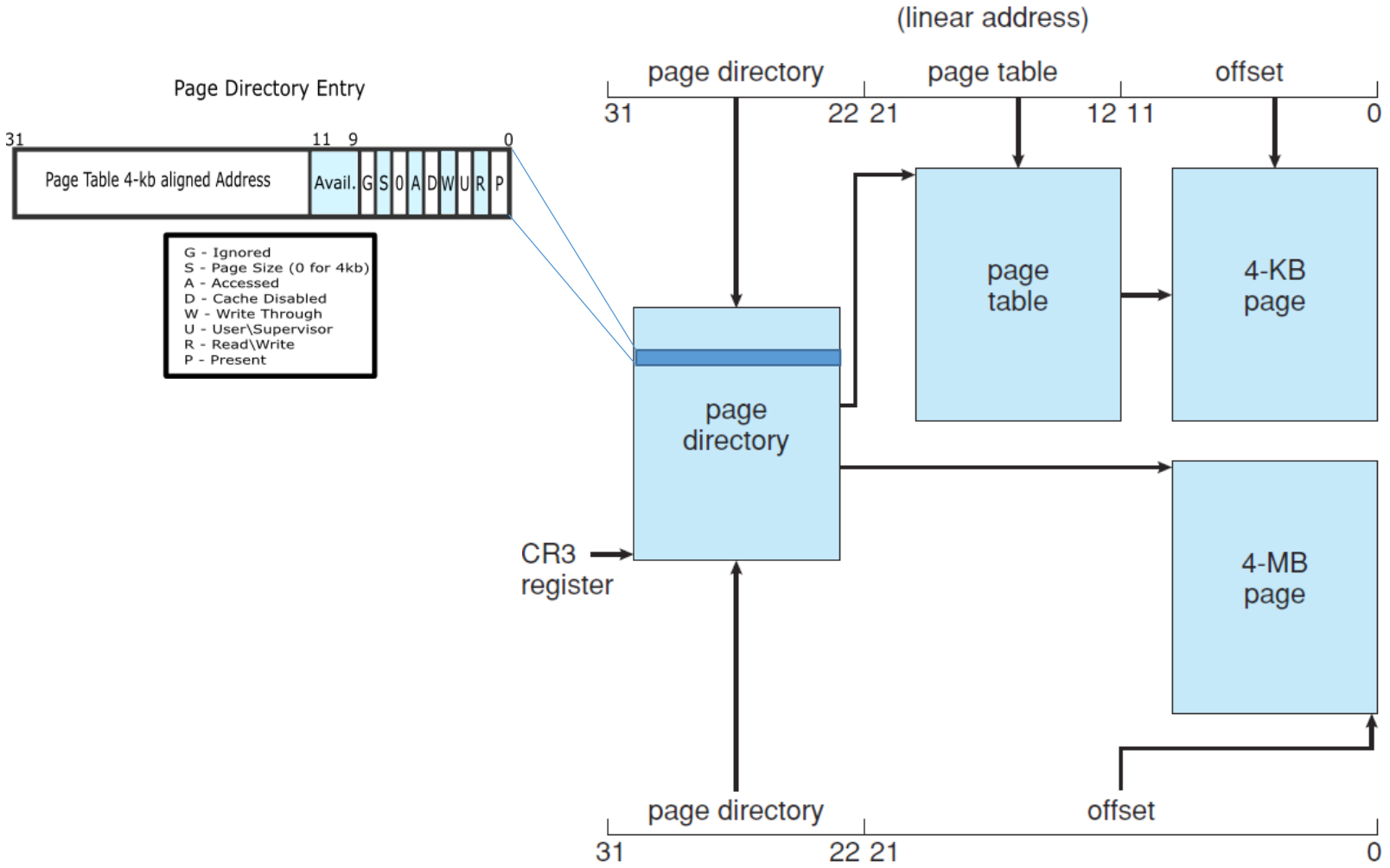
  - ✓ 10 bits (p_1) are used to index the outer page table (which is termed as page directory in IA-32)
  - ✓ 10 bits (p_2) are used to index the inner page table
  - ✓ 12 bits are the offset for a 4KB page

- For 4MB page size setup, IA-32 uses a one-level paging scheme
  - ✓ 10 bits (p_1) are used to index the page table (page directory)
  - ✓ 22 bits are the offset for a 4MB page

IA-32 architecture

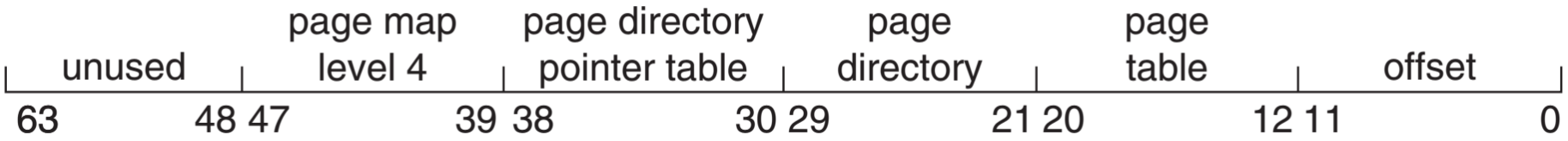
IA-32 Paging

- The CR3 register (page-table base register) points to the page directory for the current process
- One entry in the page directory is the page size (PS) flag (8th bit).
  - If PS=1, the page directory points directly to the 4-MB page frame



X86-64 architecture

- Intel developed IA-64 architecture--Itanium
  - The IA-64 Architecture was NOT widely adopted since IA-64 does not compatible with IA-32
- AMD also developed similar architecture, called X86-64 (AMD64) architecture
  - The 86-64 architecture offers several backward compatibility modes to execute IA-32 software. Intel adopted the architecture after the failure of Itanium.
- Support for a 64-bit address space yields 2^{64} addressable memory space.
- In practice, fewer than 64 bits are used for address representation in current designs. The x86-64 architecture currently provides a 48-bit logical address with support for page sizes of 4 KB, 2 MB, or 1 GB using four levels of paging hierarchy.



Kernel Memory Management

- Kernel memory is treated differently from user memory (e.g., paging scheme)
  - Different from the user space memory requests, the kernel memory requests
    - ✓ may high-frequency, small allocations (e.g., for kernel data structures, process control blocks, etc.) and large continuous blocks for certain operations (like I/O buffers, network packets, etc.).
    - ✓ need to be fast and low-latency.
  - Two methods to manage free kernel memory
    - ✓ Buddy system
    - ✓ Slab allocation

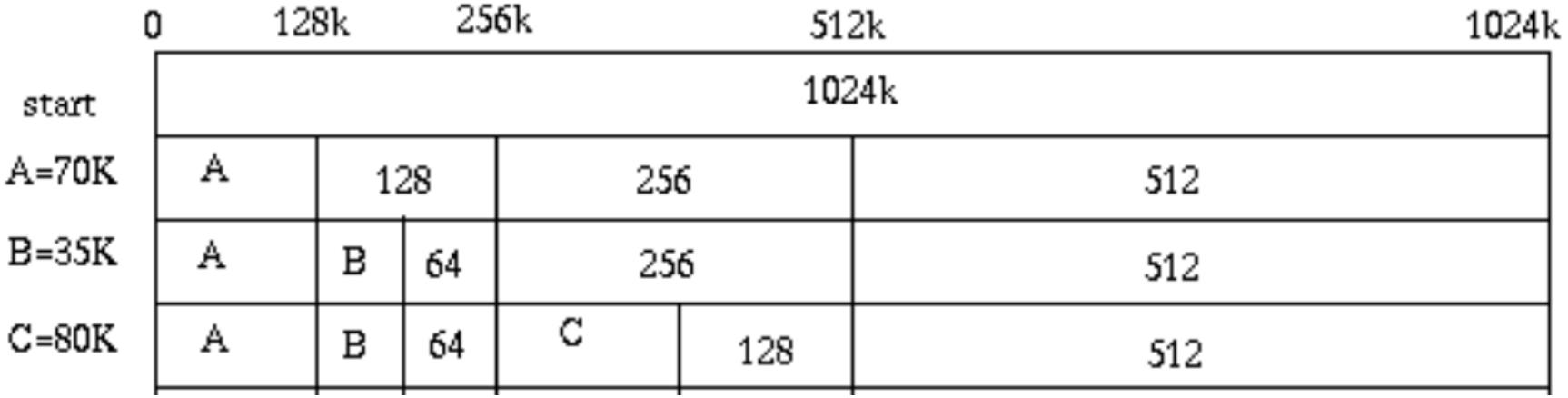
Kernel Memory Management

- Kernel memory allocation—Buddy system
  - Basic idea: the size of partition/page can be varied within a range
  - Used for UNIX kernel memory allocation
  - Detail metric
    - ✓ The size of a memory block could be 2^k, where $L \leq k \leq U$.
    - ✓ 2^L is the smallest block that can be allocated.
    - ✓ 2^U is the largest block that can be allocated.
    - ✓ Once there is an incoming memory request, a suitable block size would be searched by the order of 2^U … 2^L

Kernel Memory Management

❑ Kernel memory allocation—Buddy system

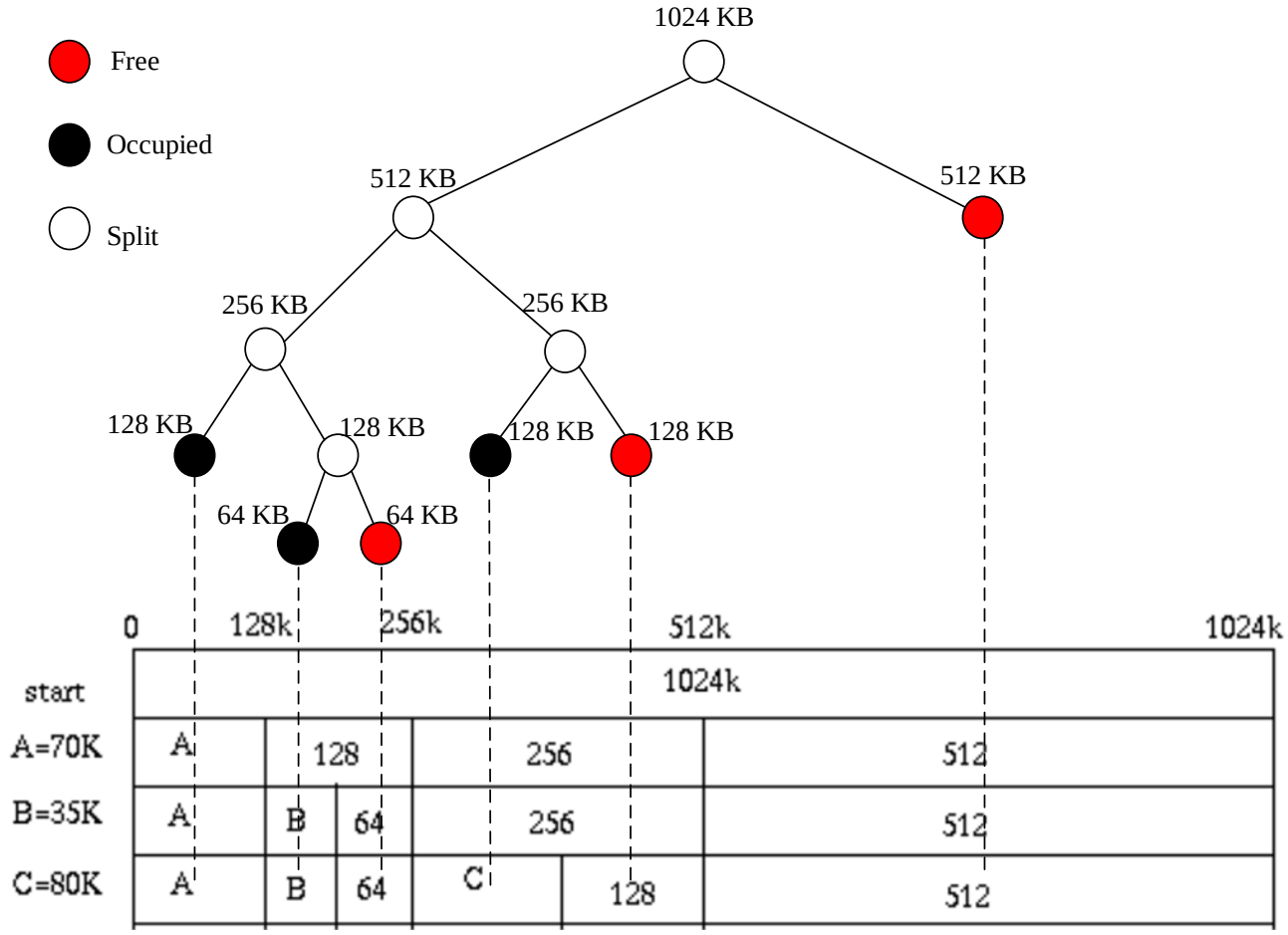
- Assume that the free kernel memory space is 1024 KB.
- There are three incoming memory requests, i.e., A, B, and C, and the requested memory size is 70 KB, 35 KB, and 80 KB, respectively. The kernel will allocate the memory to the requests sequentially.
- The memory space is divided into two buddies, each 512 KB in size.
- One of the buddies is further divided into to buddies, each 256 in size.
- The process continues until a smallest buddy (which can satisfy the requirement of the request) is found.



Kernel Memory Management

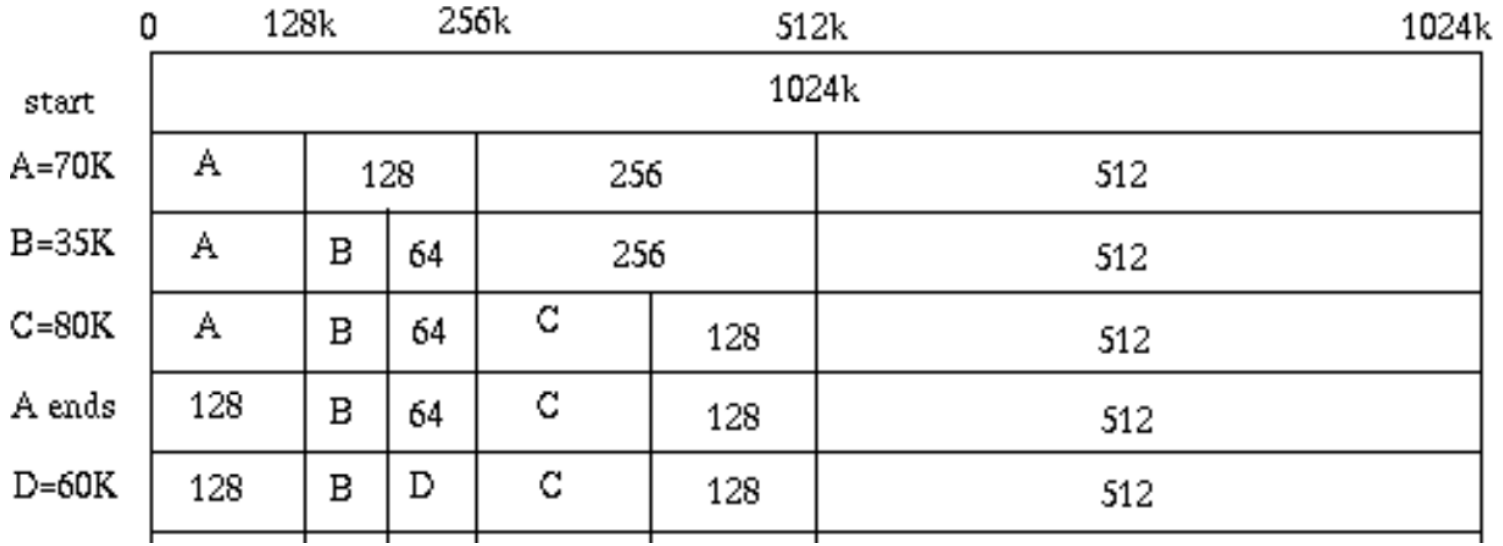
❑ Kernel memory allocation—Buddy system

- ➢ The buddy system can be represented as a binary tree.
  - ✓ Each node represents a buddy.
  - ✓ The nodes in the same level have the same memory size.
  - ✓ The memory size of a child node = the memory size of its parent node/2.
  - ✓ Each branch ends up to a free/occupied node.



Kernel Memory Management

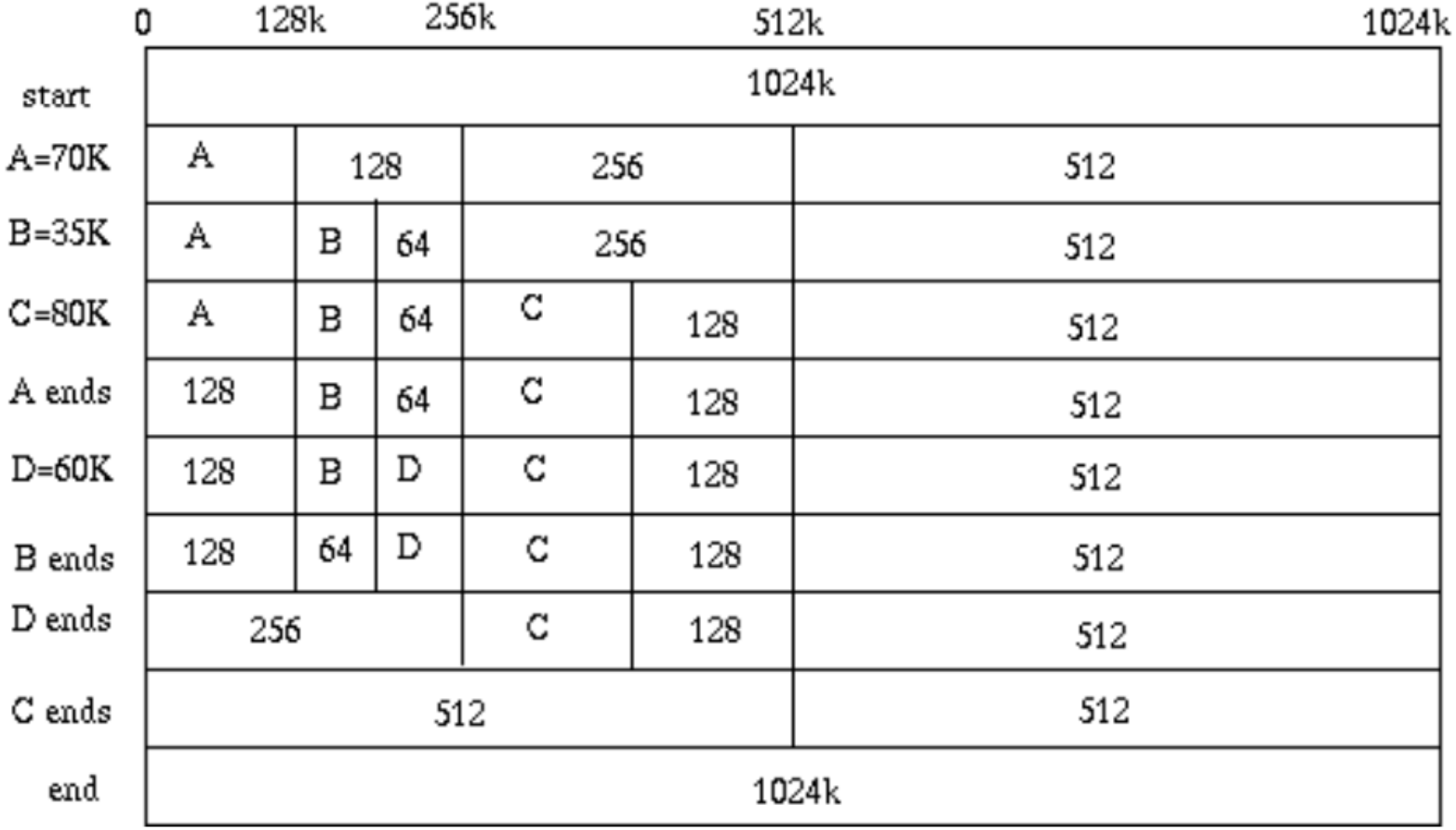
- Kernel memory allocation—Buddy system
  - What if A ends, and then a new incoming memory request D (with memory size 60 KB) shows up?
  - What if D=100 KB?



Kernel Memory Management

❑ Kernel memory allocation—Buddy system

- If occupied buddies are freed, the adjacent buddies can be combined together to form larger buddies----coalescing.



Kernel Memory Management

❑ Kernel memory allocation—Slab allocation

- Kernel modules and drivers often need to allocate temporary storage for non-persistent structures/objects (e.g., inode, PCBs, and mutex locks), which are uniform in size and are allocated and deallocated many times.
- Frequent allocation and deallocation is not efficient and may result in external fragmentation.
- Solution: frequently used data structures/objects tend to be allocated and freed often, so cache them. We need a global kernel caching allocator to manage individual caches.
- Slab allocator is design for the global kernel caching allocator.

Kernel Memory Management

❑ Kernel memory allocation—Slab allocation

- Three entities are defined in the Slab layer.
  - ✓ Cache: a cache stores the data structures with the same type.
    - For example, one cache is for all the PCBs, whereas another cache is for inode.
  - ✓ Slab: a cache is divided into multiple slabs. Each slab comprises one or more contiguous physical memory frames.
  - ✓ Object: each slab contains a number of objects, which are the data structures being cached.
  - ✓ Each slab is in one of three states
    - Full slab: all objects in a slab are allocated.
    - Empty slab: all objects in a slab are free.
    - Partial slab: some objects in a slab are free.