CMPT 300
Operating System I
Memory Management – Chapter 9

Dr. Hazra Imran

Summer 2022

Outline

- Introduction
- Virtualization
- Registers
- Segmentation
- Swapping
- Fragmentation
  - Free Space Management
- Paging

Logical address vs Physical address

virtual address

- Logical address – generated by the CPU ____________

- Physical address – address seen by the MMU __________

- The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.

LAS & PAS same in compile-time and load time address binding schemes

LAS & PAS differ in execution-time based schemes

Recap

Main-Memory Management

- MM is a large array of words or bytes.
- Each word or byte has its own address.
- MM is a repo of quickly accessible data shared by the CPU and I/O devices.

The major activities of an operating system regarding MM are:

- Keep track of which part of memory is currently being used and by whom.
- Decide which processes are loaded into memory when memory space becomes available.
- Allocate and deallocate memory space as needed.

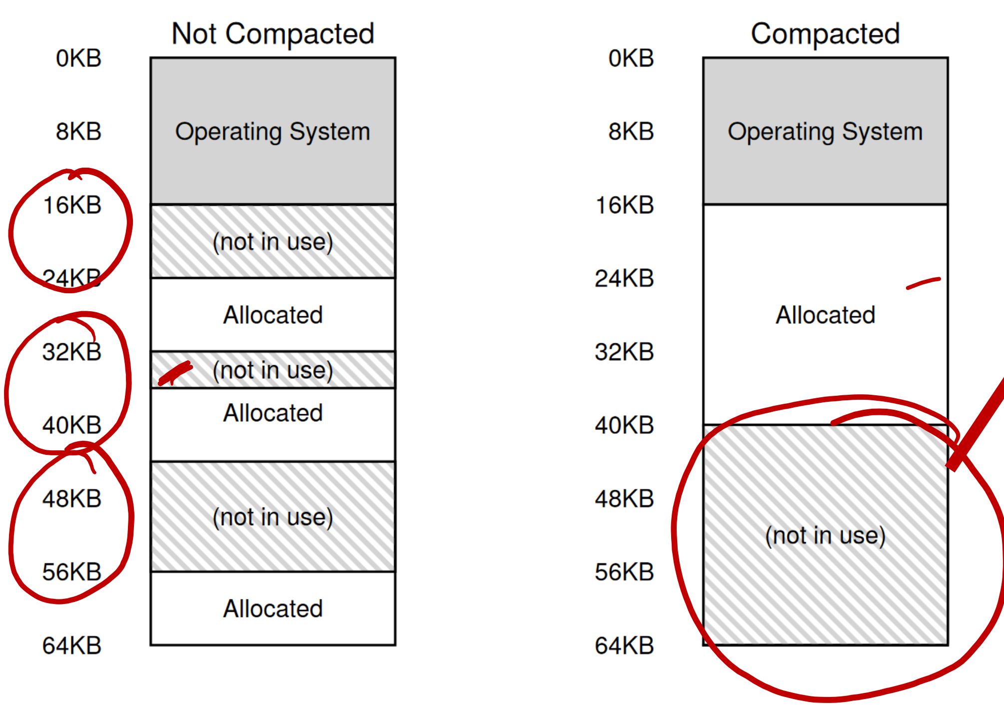
External Fragmentation

Total memory space enough for allocation, but it is not contiguous

Solution - Compaction:

- Copy data to one contiguous chunk of memory
- Main disadvantage of compaction: slow

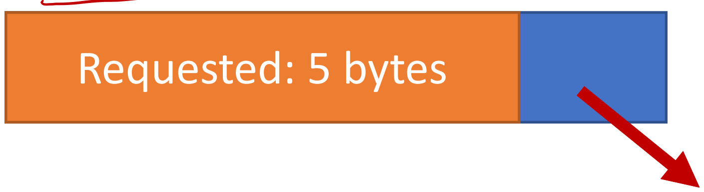
20 KB



Good for accommodating a max 24KB address space

Internal Fragmentation

- Allocated memory may be larger than requested
  - E.g., asked for 5 bytes (through malloc), allocated 8 bytes



3 extra bytes left unused

- Occurs when the system has fixed allocation sizes
- The hole is *internal* to the allocated memory
  - Compaction **cannot** help: OS/allocator has no control once the memory is handled to the requesting process

Memory Allocation Mechanisms

- Techniques used by contiguous allocation policies
  - Best-fit, worst-fit, first-fit…
- Useful for both OS and user-space memory allocators
  - E.g., malloc (malloc + free interfaces)
- Mechanisms
  - Splitting
  - Coalescing
  - Tracking allocated memory
  - Tracking free memory

Splitting

Find a free chunk of memory, split it into two

- Useful when requested memory size is smaller than the size of free memory

1. Initially: 30 bytes space in total, 10 bytes already allocated

Free list: Head → Address: 0, Length: 10 → Address: 20, Length: 10 → Null

2. After splitting the first 10 bytes for a new allocation of 5 bytes:

Free list: Head → Address: **5**, Length: **5** → Address: 20, Length: 10 → Null

Coalescing

Merge returned (adjacent) memory chunks into a big chunk

- Might be required for bigger allocations

1. Initially, 3 chunks being used (allocated)

Free list: Head → Address: 5 Length: 5 → Address: 20 Length: 5 → Null

2. First and third chunk freed:

3. After coalescing adjacent free chunks:

Head → Address: 0 Length: 10 → Address: 10 Length: 15 → Null

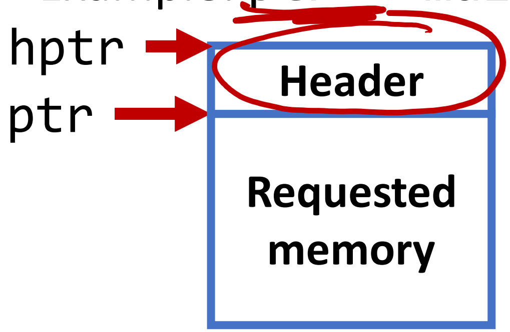
These two can also be merged

Tracking Allocated Memory

Memory allocator interfaces:

- Allocation: `void *allocate(int size);` No “size” parameter
  - E.g., malloc
- Deallocation: `void deallocate(void *memory);`
  - E.g., free
  - Size information needed for managing free space (“holes”)
  - Solution: store extra information (e.g., size) in a header block together with the allocated memory

Example: `ptr = malloc(10);`



Returned the caller: `ptr`

Used by allocator: `hptr = ptr - sizeof(Header)`

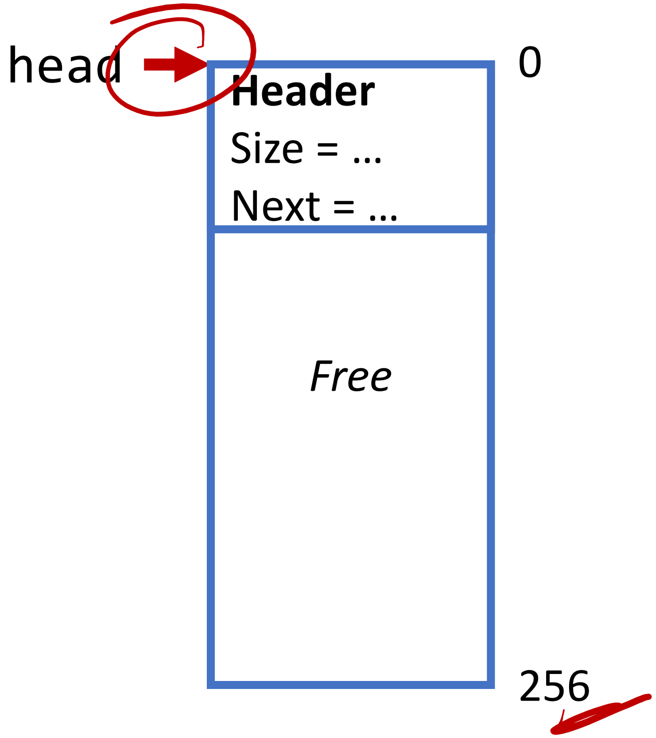
- Header size included in free list
- Not visible to user

Tracking Free Memory

- Free list needed for tracking free memory chunks
  - Require memory to store the nodes, too
  - But there is no allocator to use – we are building one!
  - Solution: embed the list in memory

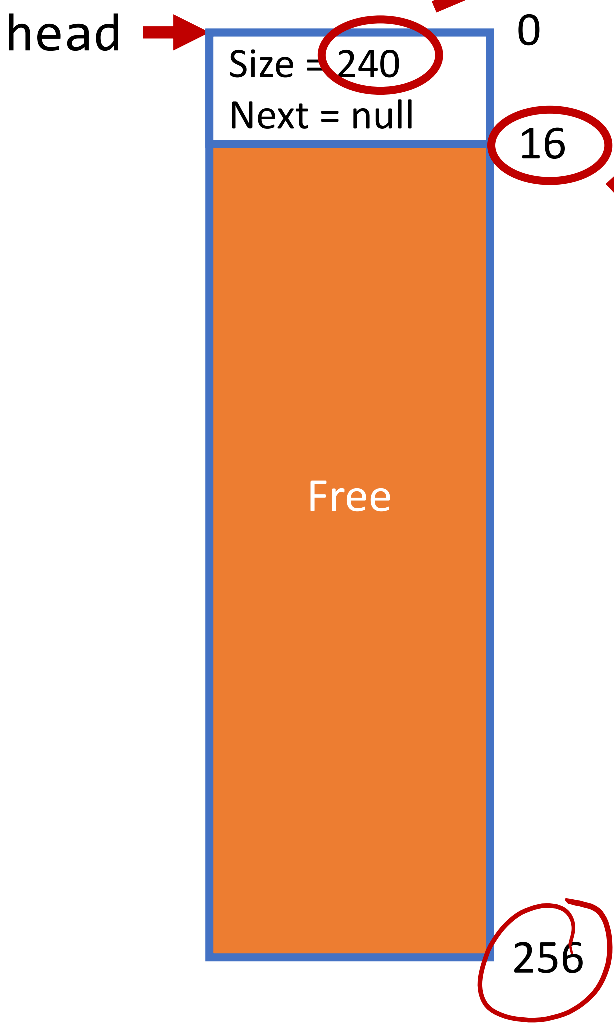
Example:

- Total memory (heap) size: 256 bytes
- Initially no allocation
  - Conceptually, the "free list" has only one entry
  - Head pointer points to the beginning of the memory space
- Use split/coalescing techniques for allocation/deallocation



Tracking Free Memory

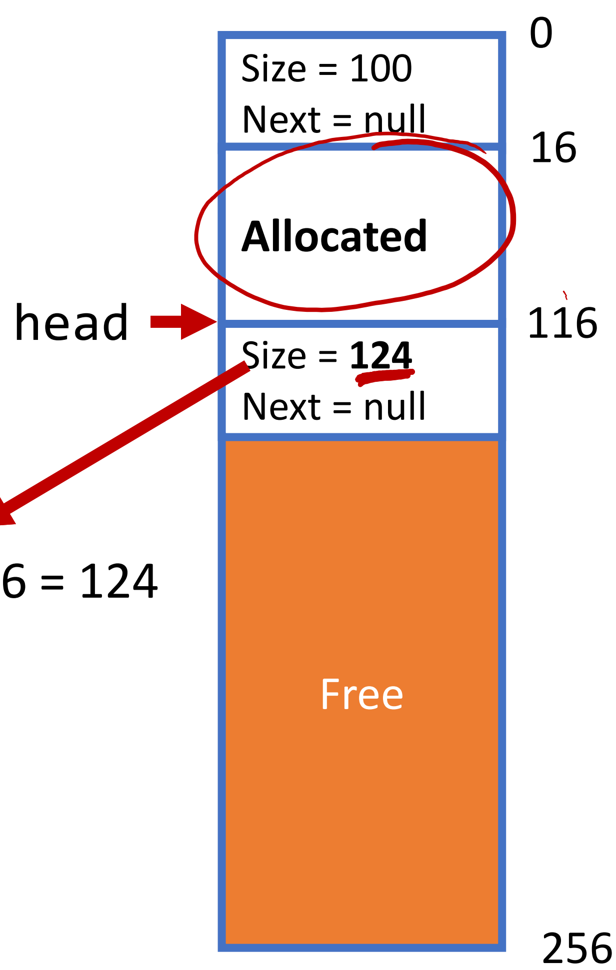
Initial state:



Size of actual free memory:
256 – 16 = 240

Header size

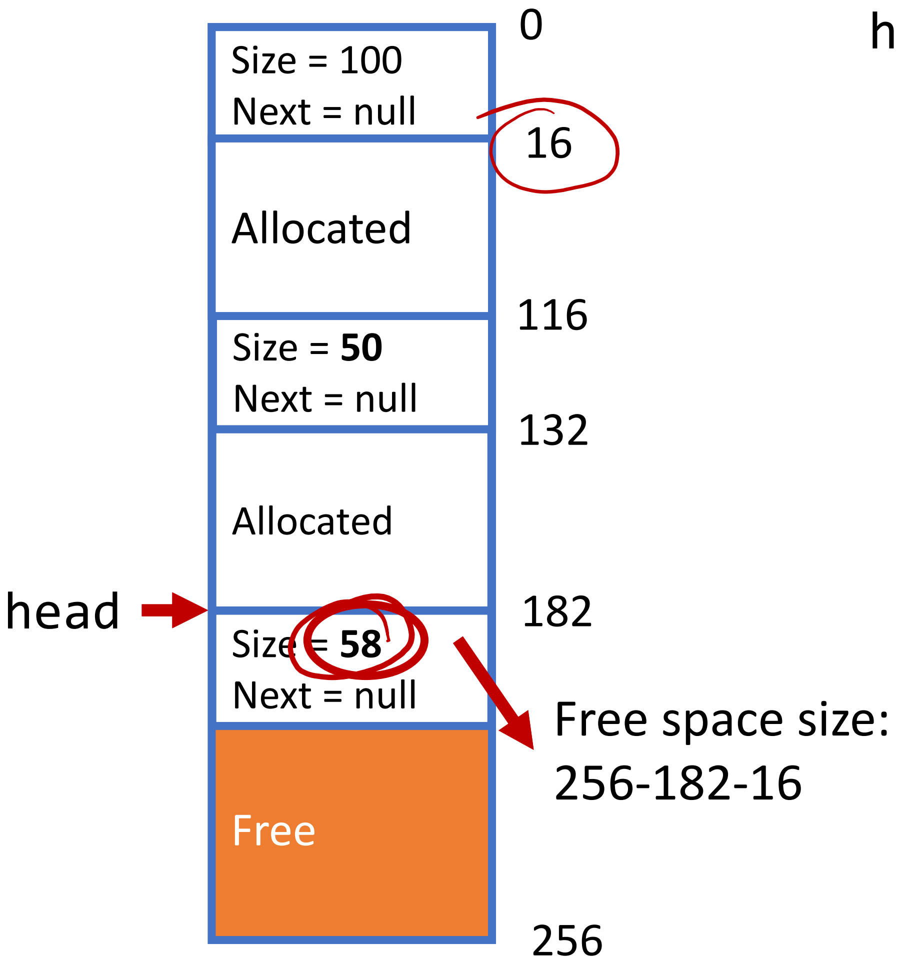
After allocating 100 bytes:



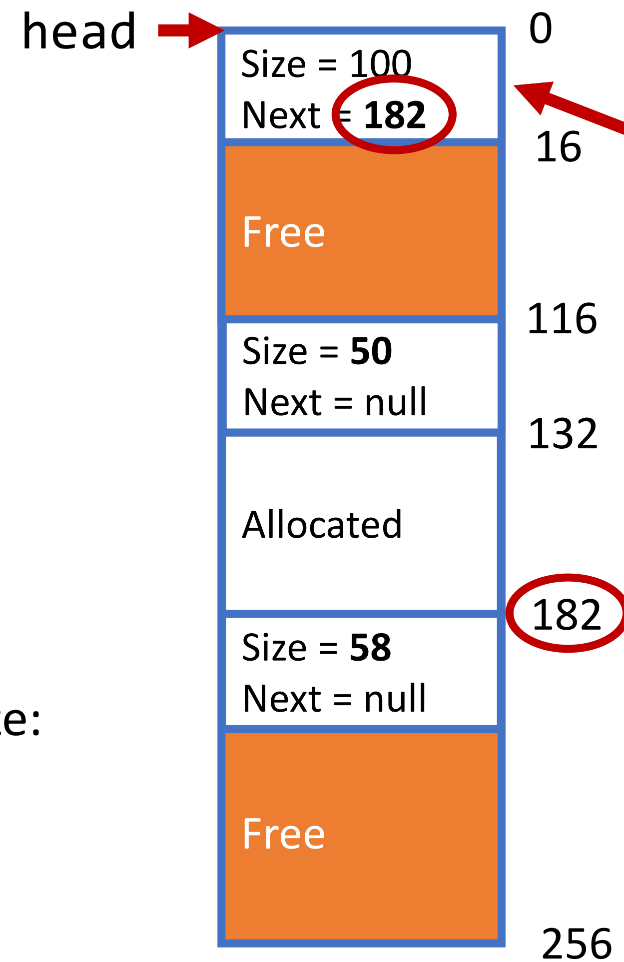
(256 – 116) – 16 = 124

Tracking Free Memory

Allocating another 50 bytes:



Freeing the first allocated chunk (100 bytes)



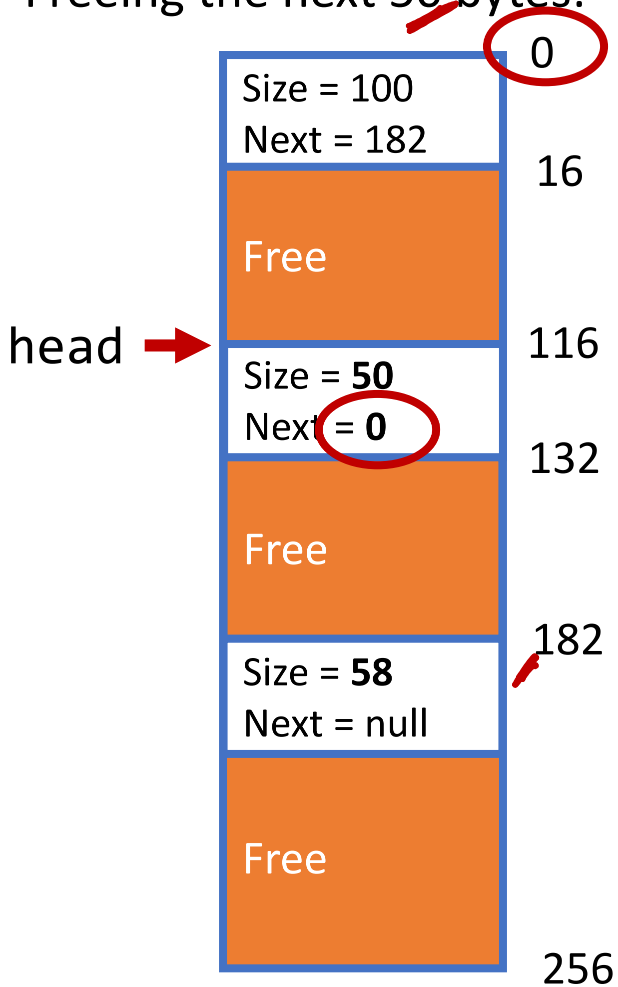
next points to previous head at 182

Always insert at list head

- Freed chunk becomes new head

Tracking Free Memory

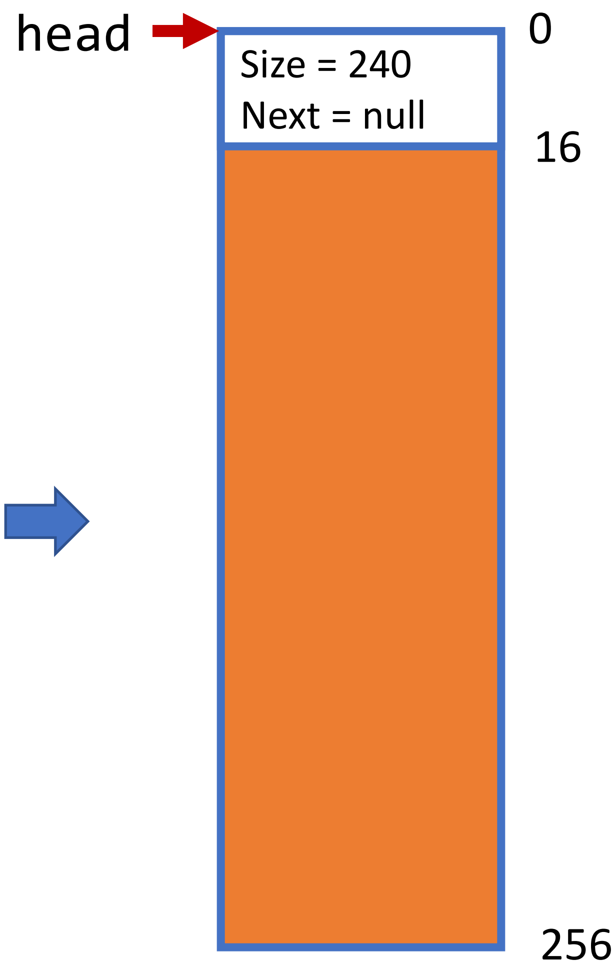
Freeing the next 50 bytes:



Q: Allocate 150 bytes?

A: No way to allocate – space fragmented in three pieces. Coalescing needed.

Coalescing:



Let’s summarized

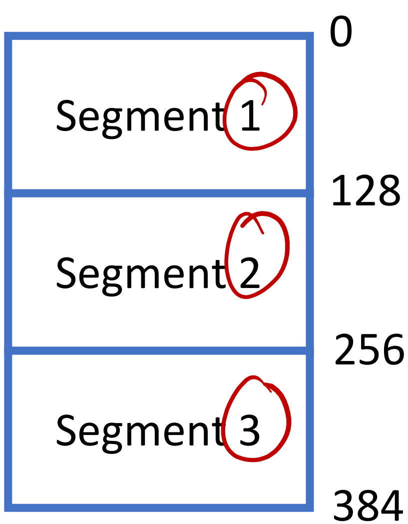
- Objective: Allocate a contiguous slab of memory to each process so that they can store their address space
- The mechanisms are “easy”
  - MMU for memory protection and relocatable code Segmentation
  - Dynamic Memory / Dynamic Loading / Dynamic Linking for smaller address spaces
  - ...
- But finding a good policy is really hard
  - For process picking, hole picking, placement in hole
- Fragmentation is unavoidable and wastes RAM

- **Bottom Line**: Contiguous memory allocation is difficult

Contiguous vs. Non-contiguous Allocation

Contiguous allocation

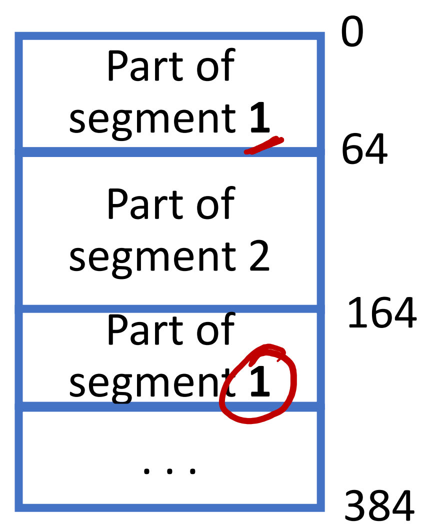
- Addresses for a segment or process must be contiguous



Segmentation

Non-contiguous allocation

- Paging: underlying physical memory and segment decoupled



Segmentation with paging

Outline

- Introduction
- Virtualization
- Registers
- Segmentation
- Swapping
- Fragmentation
  - Free Space Management
- **Paging**

Paging

PM - frames
LM - pages

- Process is allocated memory wherever it is available
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is the power of 2 (typically 512 bytes to few GBs)
  - OS keeps track of all free frames
- Divide logical memory into blocks of the same size called **pages**
- A page can be put at any frame
- For a program of size n pages, need to find n free frames
- Use page tables to translate logical to physical addresses

Paging

External frag — No
Inter frag — Yes

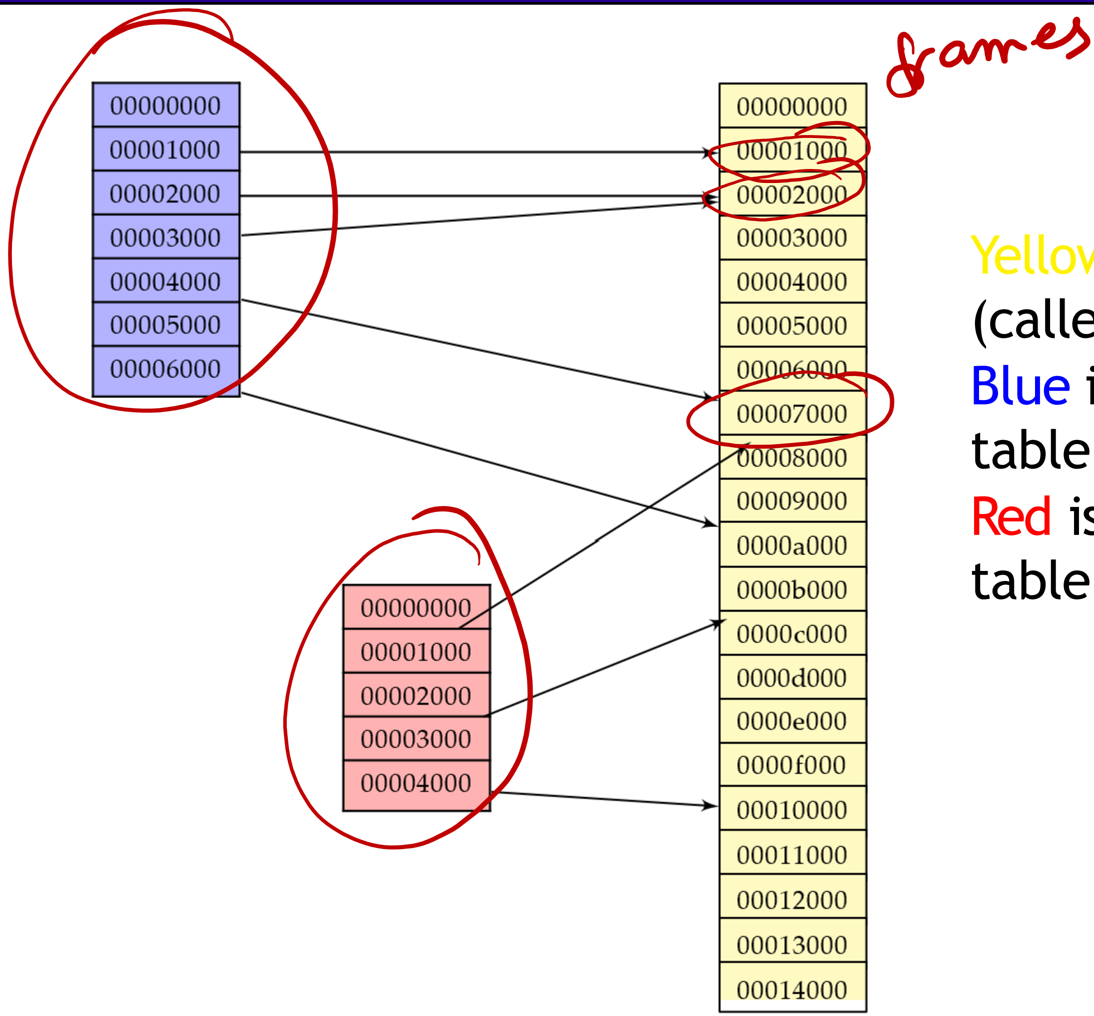
- Permits physical address space of a process to be physically non-contiguous
- The process is allocated physical memory whenever the physical memory is available – no external fragmentation
    - What about Internal fragmentation?
- This avoids compaction.

program
page P2 P3

- Physical blocks – Frame
- Logical blocks – Pages
- Size of frames and pages is defined by hardware (power of 2 to ease calculations)
- An address is determined by:
- page number (index into table) + offset

  <-- mapped into --> base address (from table) + offset.

Paging



Yellow is the physical RAM pages (called *frames*).

Blue is process 1's virtual page table.

Red is process 2's virtual page table.

Each process
↓
PT

Lets take an example

offset – house street – Page #

Assume:

<2,4000>

Virtual address is 32 bits, page offset is 12 bits, physical address is 30 bits

Logical address

31	11 0
Virtual page number *20 bits*	Page offset *12 bits* *4096*
2	*4000*

Logical Page number	Physical Frame #
0	2
1	0
2	14
…	
1048575	

2^{20} entries

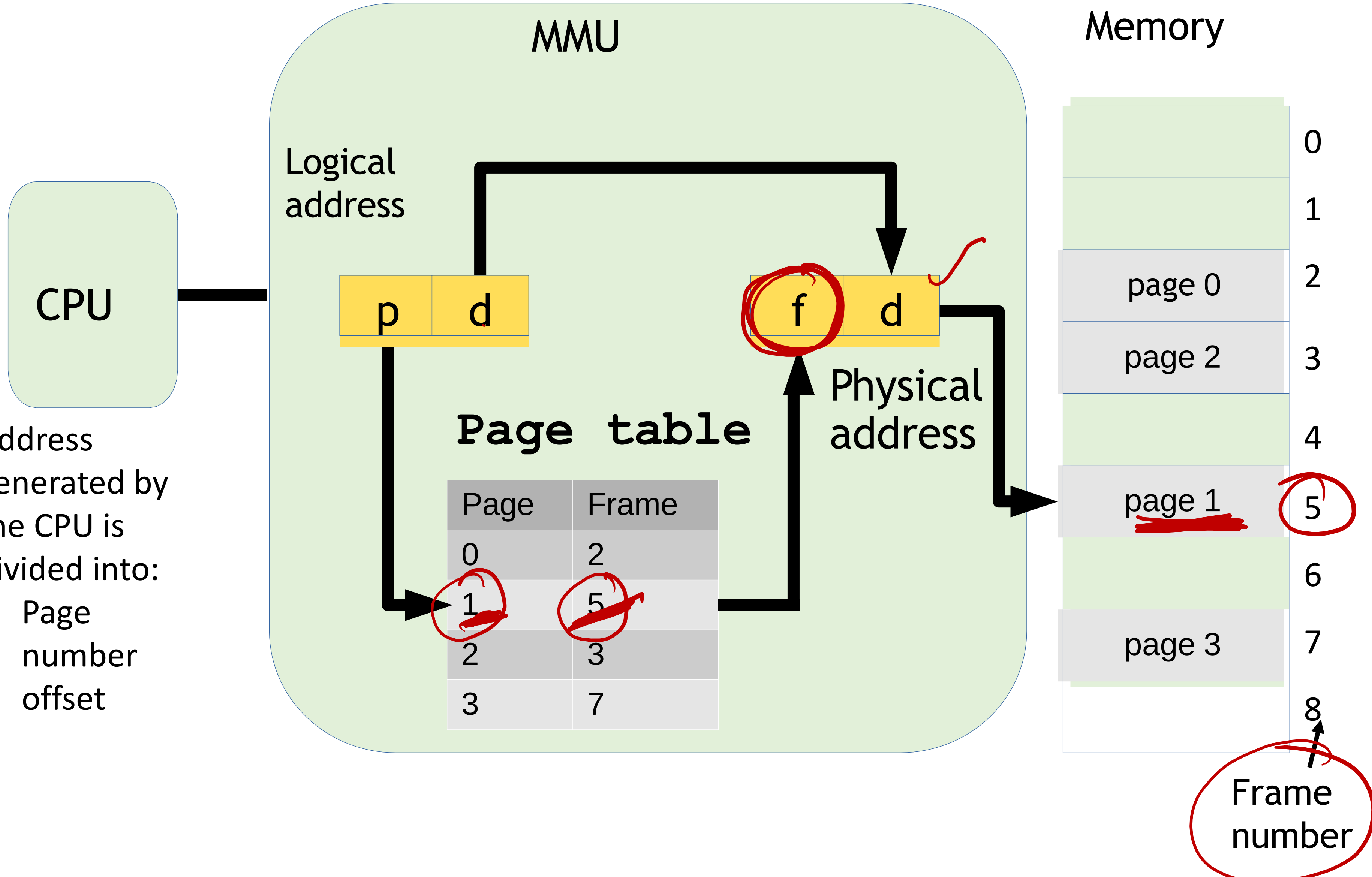
Physical address

29	11 0
page number 18 bits	Page offset 12 bits 4096 cells
14	*4000*

Page number	
0	
…	
14	0 1 2 … 4000 … 4095
…	
262,143	

2^{18} entries

Page address translation



Address generated by the CPU is divided into:

- Page number
- offset

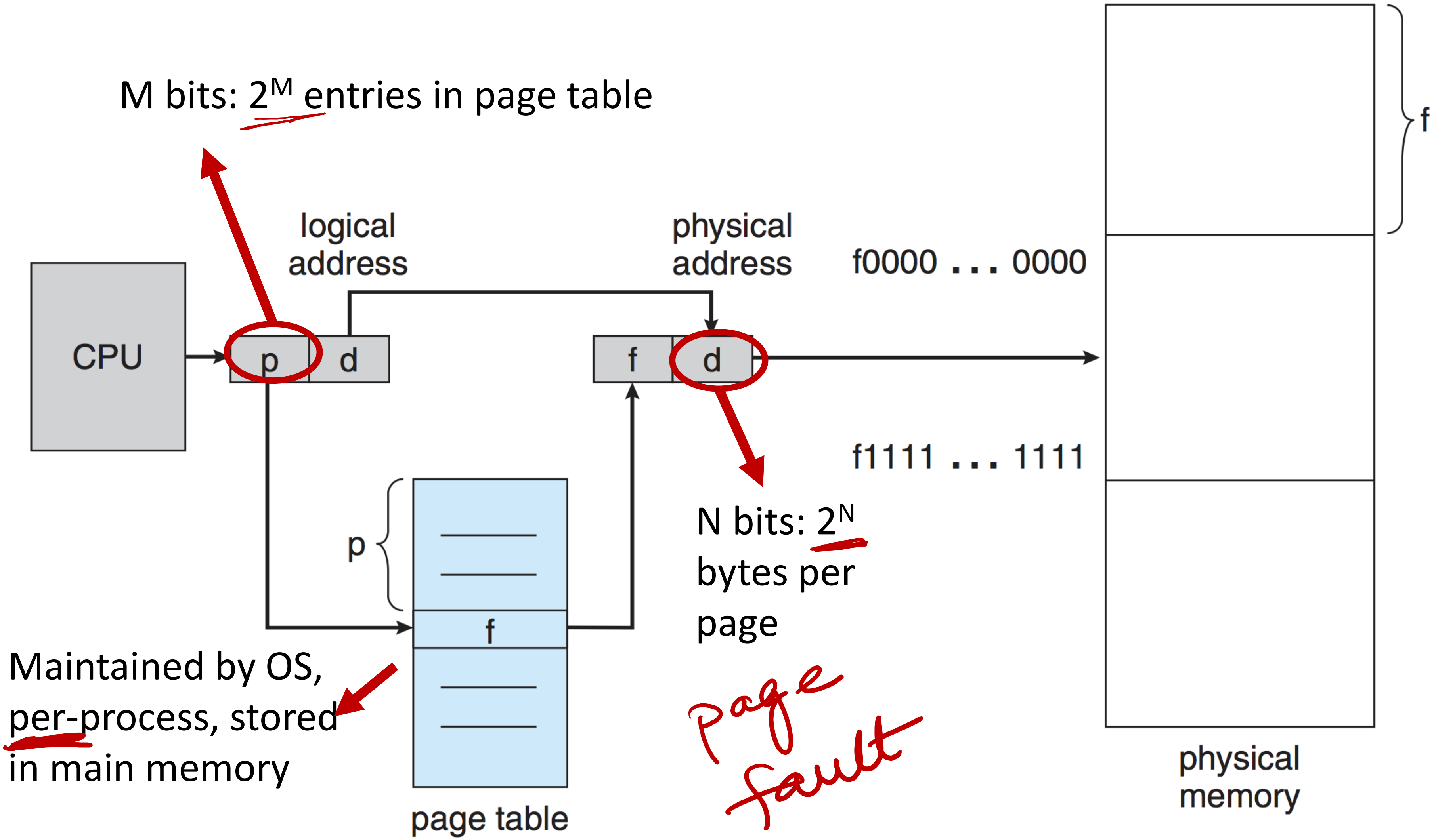
Virtual Addresses

- Address generated by CPU is divided into:
  - Page number (p): used as an index into a page table, which is an array of entries for translating page # to frame #
  - Page offset (d): combined with frame # addr to define the physical address that is sent to the memory unit

page number	page offset
p	***d***
$(m - n)$ bits	n bits

- Logical address space = 2^m bytes
- Page size = 2^n bytes
- Number of pages = 2^{m-n}

Address Translation in Paging



Try!

Given 4KB page size and a 32-bit virtual address, calculate

- m (# of bits for representing virtual address space) ___ 32
- n (# of bits for offset within a page) ___ 12
- the number of bits for representing page # ___ (32-12) 20

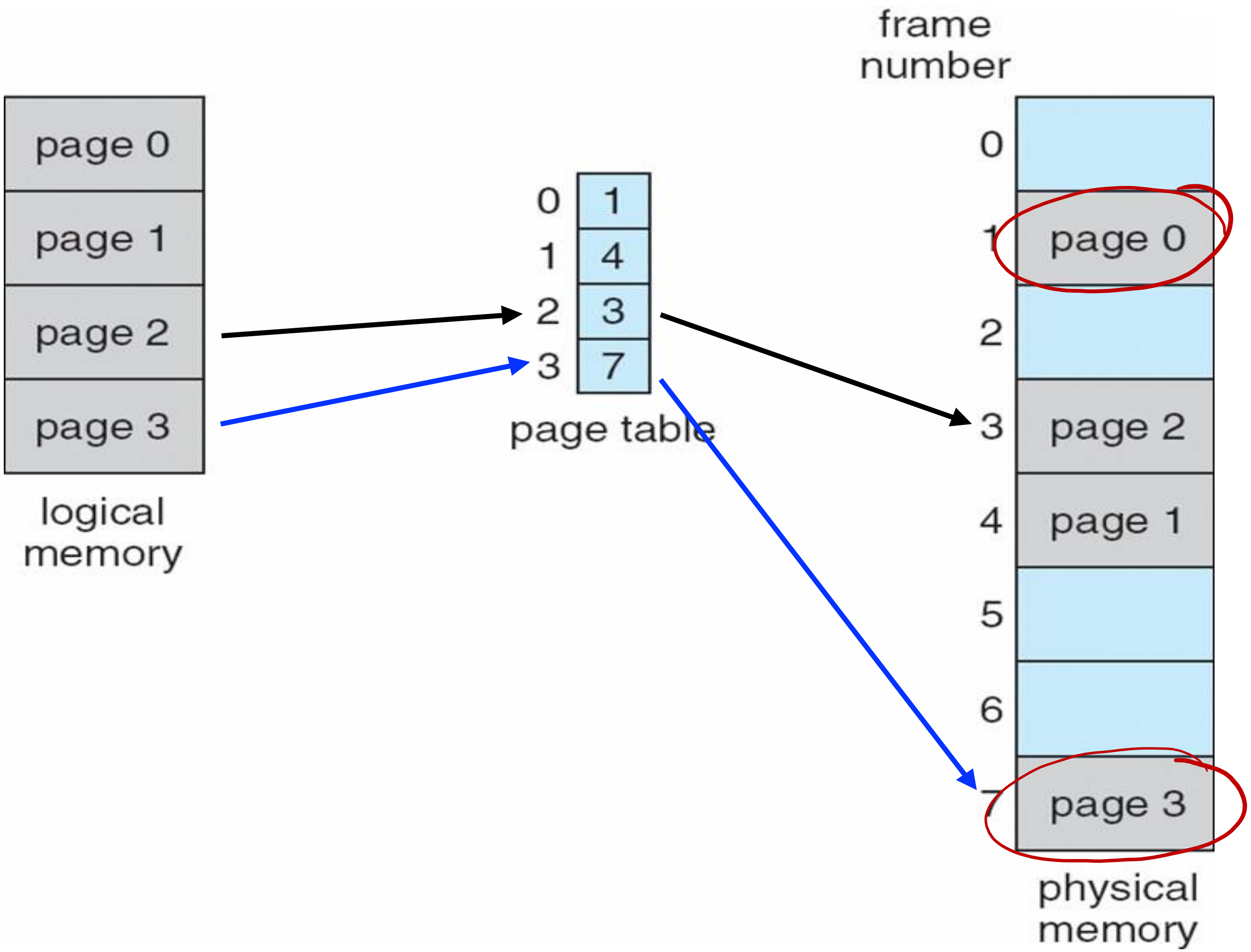
4KB = 4096 bytes

4KB

2^{12}

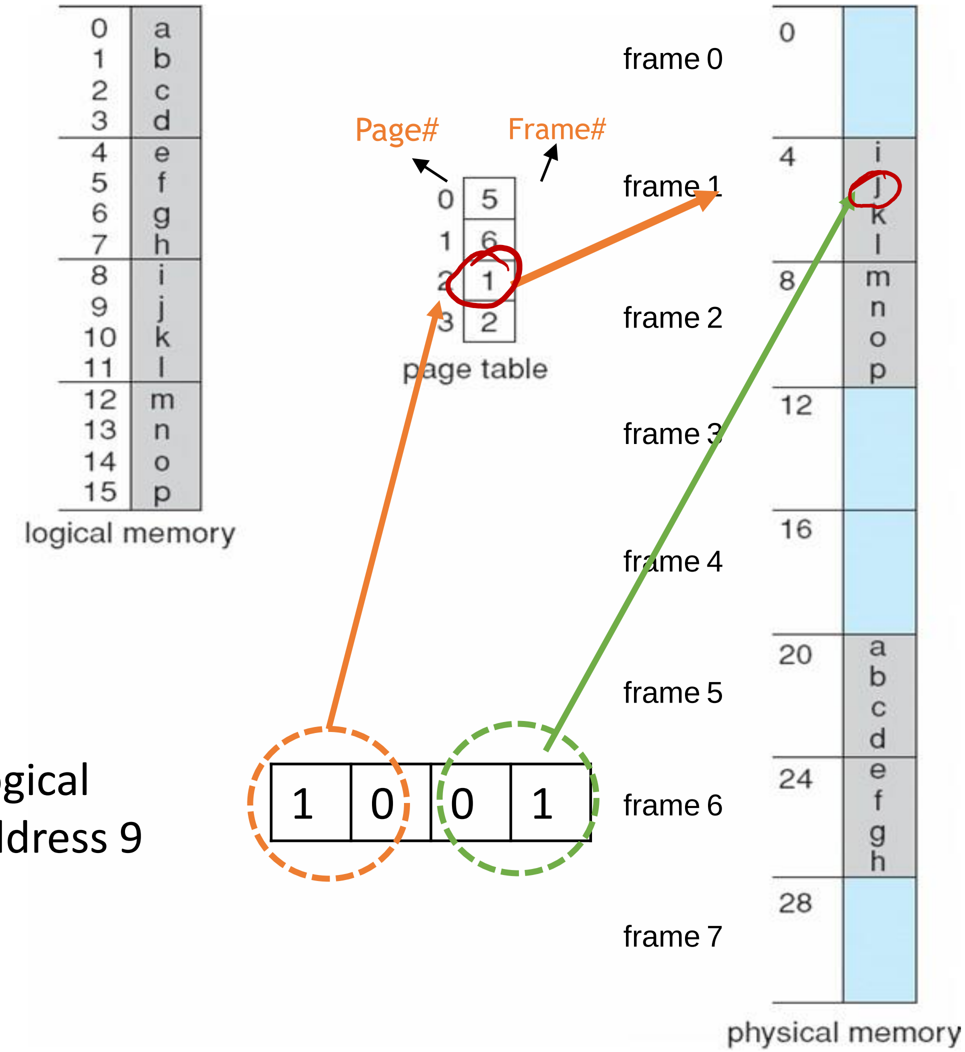
4KB = 4 x 1024

Paging Example



Paging Example

Logical address 9



Ques like:
Virtual address space: 16 bytes (m = 4)
Memory size per page: 4 bytes (n = 2)

(1) Infer the page table and
(2) use it to translate the virtual address

page number	page offset
p	*d*
m - n	*n*

Setup like
- Page size: 4-byte (2^2)
- Physical memory: 32-byte (2^5)
- Logical memory: 16-byte (2^4)

Address Translation with Paging

- For each process, there is an address **A**

page number = A / page_size

offset = A mod page_size

- This is the page number within the process address space
- e.g. address in the process, A = 10,000 and page size = 4k
- page number = 10000 / 4k = 10,000 / 4096 = 2.xxx = truncate to 2

- this is the distance from the beginning of the page
- page offset = 10000 mod 4k = 10,000 mod 4096 = 1908