Introduction to Paging

Outline

- Introduction to Paging Concepts

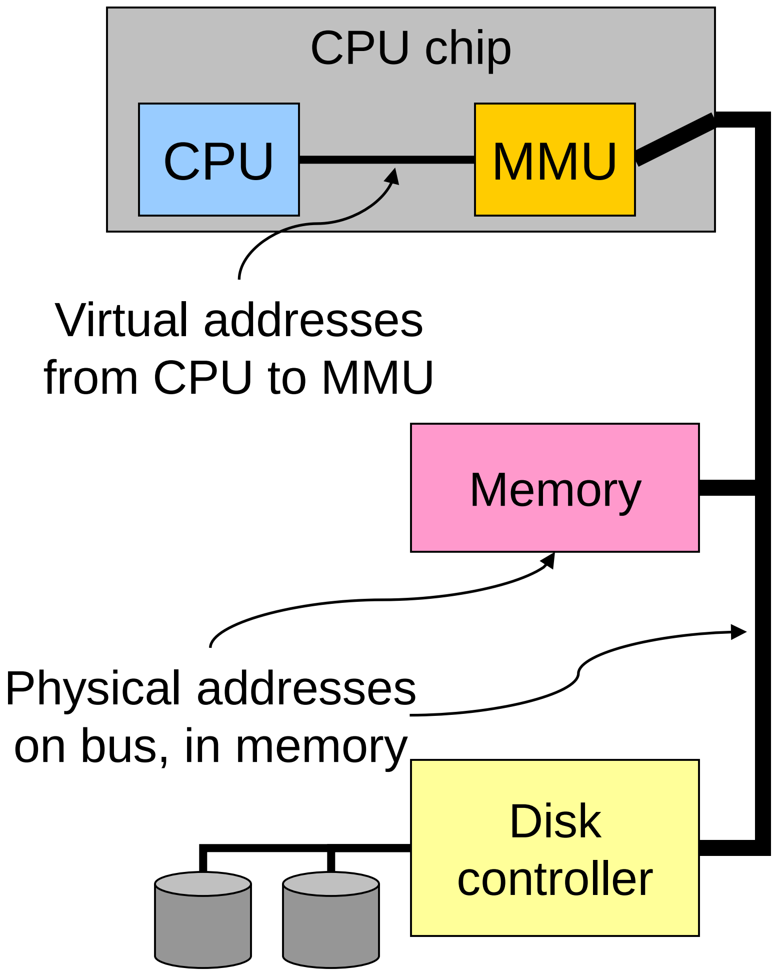
Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks
- The chunks of a process are called pages and chunks of memory are called frames
- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page

Virtual memory

- Basic idea: allow the OS to hand out more memory than exists on the system.
- Keep recently used stuff in physical memory
- Move less recently used stuff to disk
- Keep all of this hidden from processes
  - Processes still see an address space from 0 – max address
  - Movement of information to and from disk handled by the OS without process help
- Virtual memory (VM) especially helpful in multiprogrammed system
  - CPU schedules process B while process A waits for its memory to be retrieved from disk

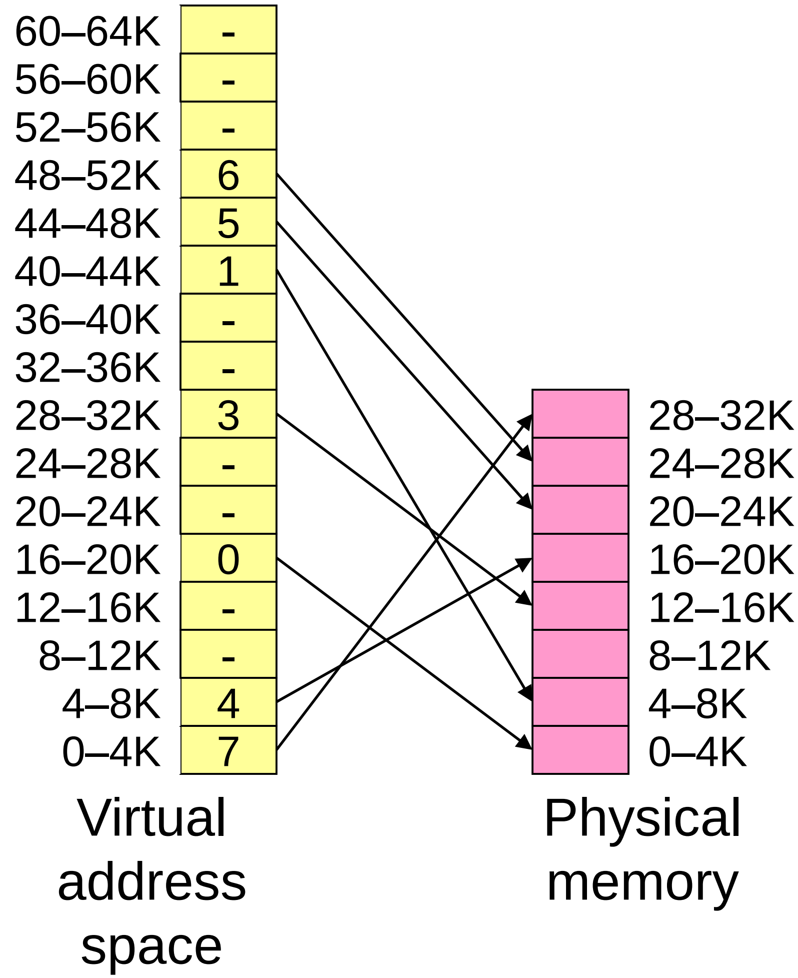
Virtual and physical addresses



- Program uses *virtual addresses*
  - Addresses local to the process
  - Hardware translates virtual address to *physical address*
- Translation done by the ***Memory Management Unit***
  - Usually on the same chip as the CPU
  - Only physical addresses leave the CPU/MMU chip
- Physical memory indexed by physical addresses

Paging and page tables

- Virtual addresses mapped to physical addresses
  - Unit of mapping is called a *page*
  - All addresses in the same virtual page are in the same physical page
  - *Page table entry* (PTE) contains translation for a single page
- Table translates virtual page number to physical page number
  - Not all virtual memory has a physical page
  - Not every physical page need be used
- Example:
  - 64 KB virtual memory
  - 32 KB physical memory



Page Example

0	0
1	1
2	2
3	3

Process A

0	7
1	8
2	9

Process B

Frame Number	
0	A0
1	A1
2	A2
3	A3
4	
5	
6	
7	B0
8	B1
9	B2
10	
11	
12	
13	
14	

Paging

- A program requires N free frames
- There is a need to set up a page table to translate logical to physical addresses
- Internal fragmentation

Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number (p) – used as an index into a page table which contains base address of each page in physical memory

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

  - Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit
  - The number of logical address bits is m

Address Translation Scheme

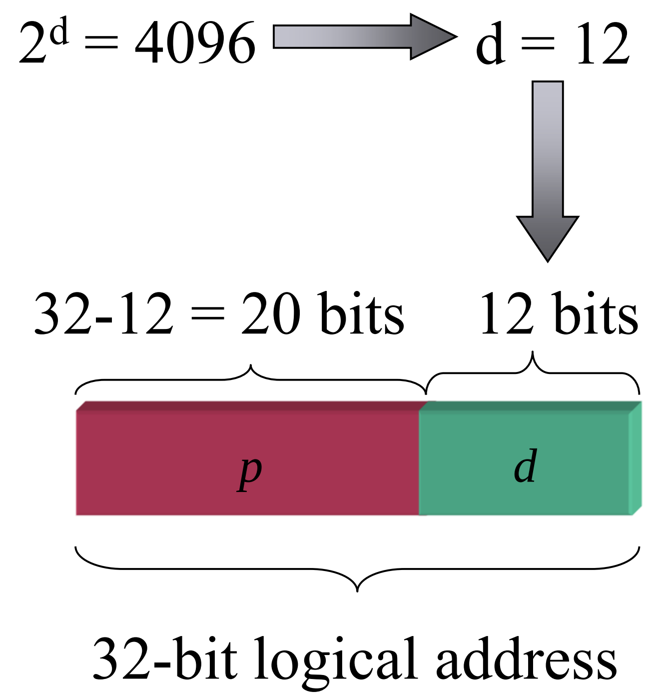
- If the number of logical address bits is m and the number of bits in the offset is n then
  - Logical address space is 2^m
  - Page size (number of entries) is 2^n
- Example: $m = 4$, $n = 2$
  - Number of address is 16
  - Number of entries in a page is 4

Mapping logical => physical address

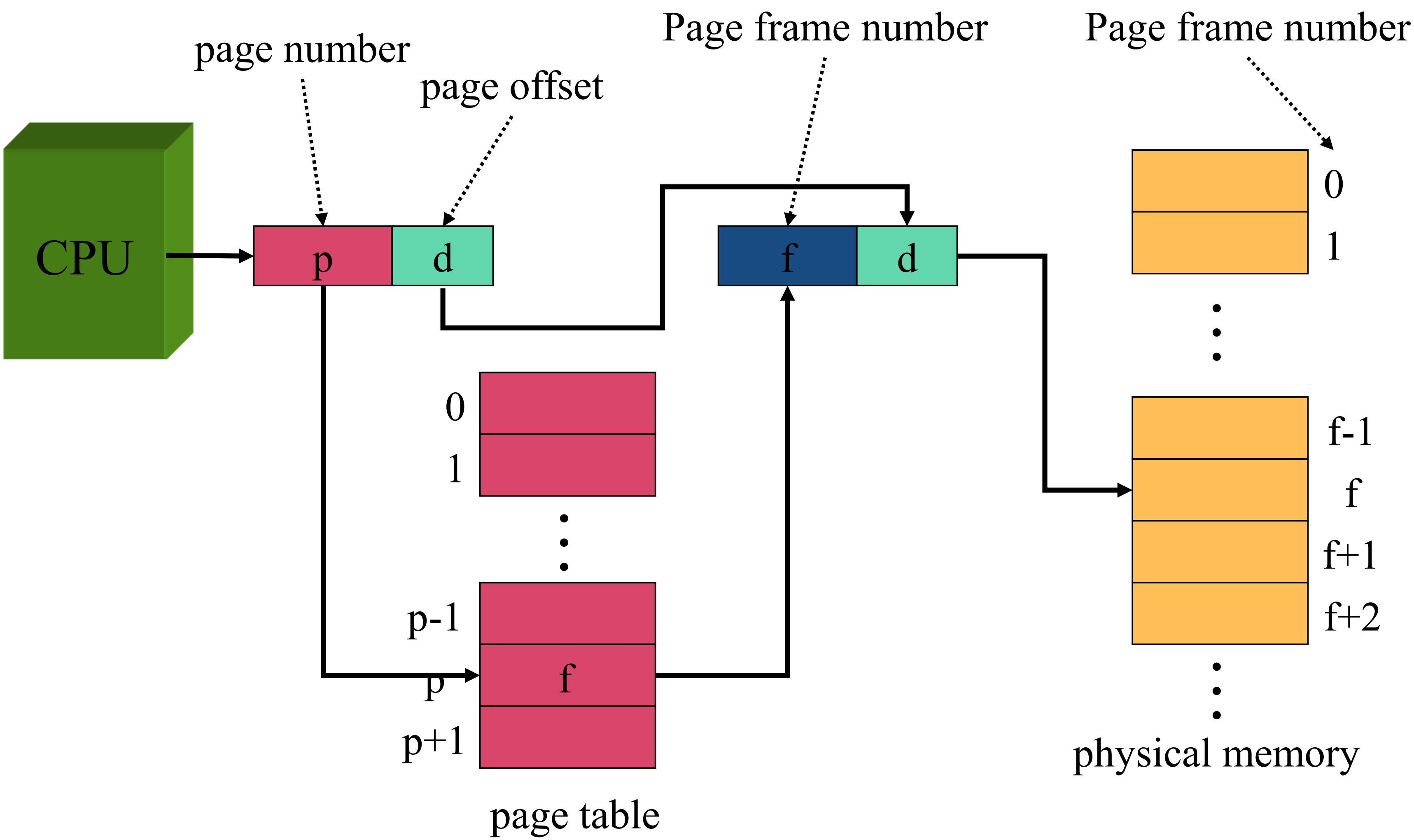
- Split address from CPU into two pieces
  - Page number (*p*)
  - Page offset (*d*)
- Page number
  - Index into page table
  - Page table contains base address of page in physical memory
- Page offset
  - Added to base address to get actual physical memory address
- Page size = 2^d bytes

Example:

- 4 KB (=4096 byte) pages
- 32-bit logical addresses



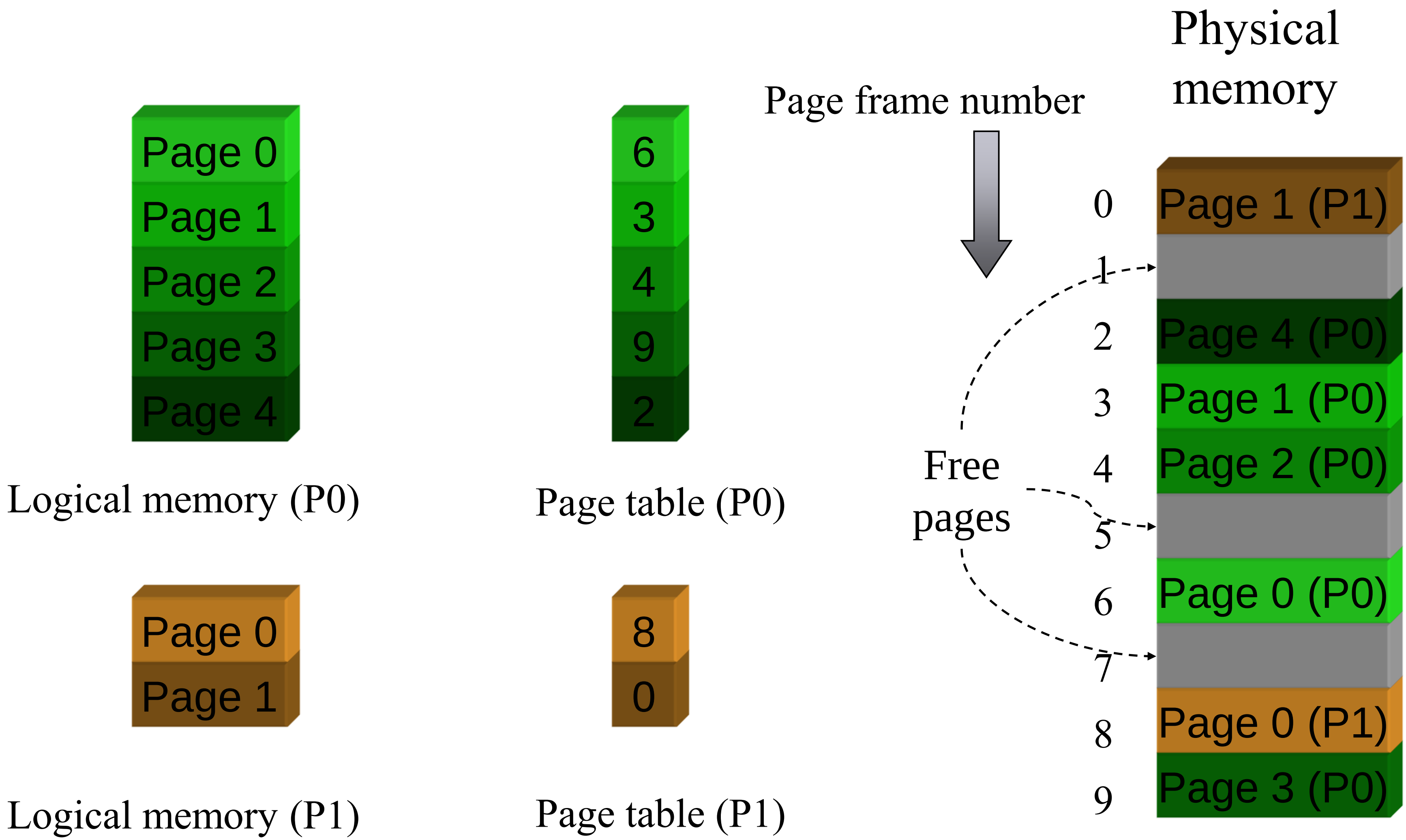
Address translation architecture



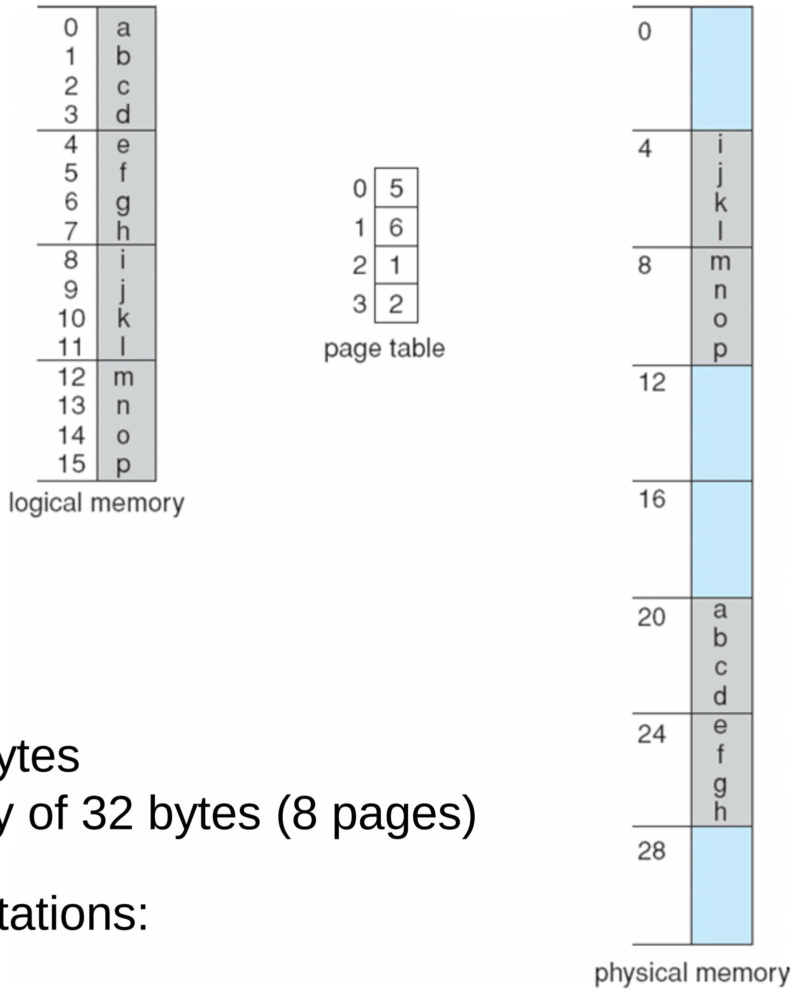
Paging Hardware

- The CPU issues a logical address (remember that all addresses are binary)
- The hardware extracts the page number, p, and the page offset d
- The page number, p, is used to index the page table
  - The entry in the page table consists of the frame number, f
- The actual address is the concatenation of the bits that make up f and d.

Memory & paging structures



Paging Example



m=4
n=2
Page size of 4 bytes
Physical memory of 32 bytes (8 pages)

Binary representations:
0 00
1 01
2 10
3 11

Paging Example

- First a little reminder. All addresses are in bits. The example on the previous page gives the base 10 equivalent of binary numbers

0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Example

- Logical address in binary form is 0000
  - From the bits we extract the page number as 00 and the offset as 00
  - Page number 00 corresponds to the first entry of the page table.
  - There we find that the corresponding frame is 5 (or 0101)
  - The physical address produced is the concatenation of the bits of the frame number and the offset
    - 010100
  - In base 10 we find that the physical address is 20 (5*4+0)

Example

- Logical address in binary form is 0100
  - From the bits we extract the page number as 01 and the offset as 00
  - Page number 01 corresponds to the second entry of the page table.
  - There we find that the corresponding frame is 6 (or 0110)
  - The physical address produced is the concatenation of the bits of the frame number and the offset
    - 011000
  - In base 10 we find that the physical address is 24(6*4+0)
- What do you think logical address 13 maps to?

Observations

- Let's look at logical addresses:
  - 0000 (0), 0001 (1), 0010(2) and 0011(3)
  - The first two bits (page number) are the same
  - This means that these logical addresses will be in the same frame.
  - The position in the frame is determined by the offset
- For a process its pages can be in any order
  - For our example page 1 is in a frame that appears "later" than the frame associated with page 2

Observations

- The logical address space and the physical address space DO NOT have to be the same size
- The logical address space can be larger then the physical address space (more on this later)

Paging

- No external fragmentation
  - Any free frame can be allocated to a process that needs it
- There may be some internal fragmentation
  - The memory requirements may not coincide with page boundaries
  - The last frame allocated may not be completely full

Paging

- Internal fragmentation is on average one half page per process
- Larger page sizes means more wasted space
- Smaller page sizes means larger tables

Summary

- This section introduced the concept of paging in memory