Main Memory Management

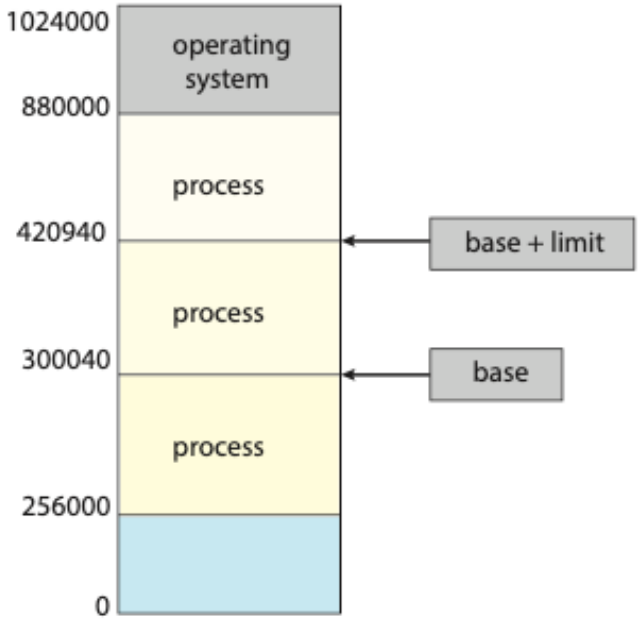


Figure 9.1 A base and a limit register define a logical address space.

FRAMES: Fixed sized chunks of **PHYSICAL** memory. These are referred to by **FRAME NUMBER** by the OS. A **FRAME TABLE** is an OS data structure that keeps track of the properties of each frame.

PAGES: Fixed sized chunks of **LOGICAL** memory. They are referred to by **PAGE NUMBER** by the process. A **PAGE TABLE** is an OS data structure associated with each process that keeps track of what **FRAMES** the process "owns"

PAGE SIZE == FRAME SIZE
PAGE TABLE size is often variable (processes have different memory sizes)
FRAME_TABLE size is determined by the size of physical memory

Processes do NOT access memory directly. They make reference to "virtual" addresses that go from zero to some limit. The contents of the page table are used to translate virtual addresses into physical addresses.

Exercise 01: A typical frame size for a Linux machine is 4K (4096 bytes of memory). For a machine with 16 GB of memory, how many frames are there?

Hint:

2^{30} = 1 GB	2^{33} = 8 GB
2^{31} = 2 GB	2^{34} = 16 GB
2^{32} = 4 GB	2^{35} = 32 GB

Exercise 01: A typical frame size for a Linux machine is 4K (4096 bytes of memory). For a machine with 16 GB of memory, how many frames are there?

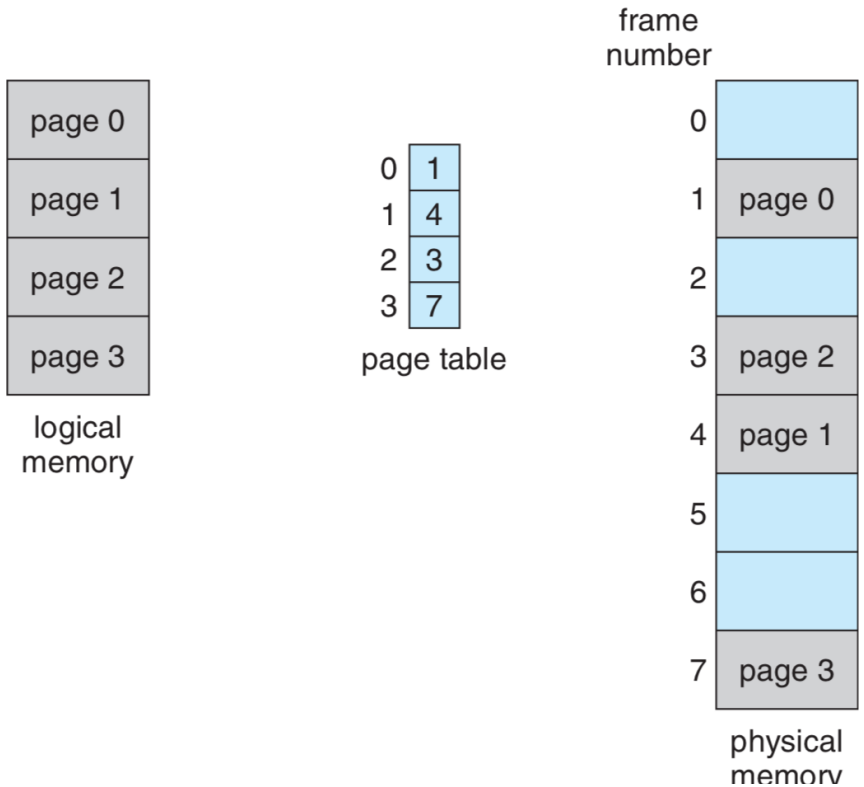
Hint:

2^{30} = 1 GB	2^{33} = 8 GB
2^{31} = 2 GB	2^{34} = 16 GB
2^{32} = 4 GB	2^{35} = 32 GB

2^{34} = 16 GB 2^{12} = 4 KB

$2^{34} / 2^{12} = 2^{(32-12)} = 2^{22}$ = 419304 = 4MB of pages

University of CINCINNATI



LOGICAL ADDRESSES range from 0 to some max value. The logical address is broken into two fields. If the frame (page) size is 2^n and the address has *m* bits, then the first *(m-n)* bits is the PAGE number. The remaining *n* bits is the displacement INSIDE the page where the specific word lives.

p is the page number and is represented in *(m-n)* bits.
d is the page displacement (offset) and is represented in *n* bits.

Exercise 2: For a 32 bit logical address and a page (frame) size of 4K, what is *n* and how many pages (frames) could be addressed?

Exercise 2: For a 32 bit logical address and a page (frame) size of 4K, what is *n* and how many pages (frames) could be addressed?

m = 32 *n* = 12 (2^{12} = 4K)

(m – n) = 32 - 12 = 20

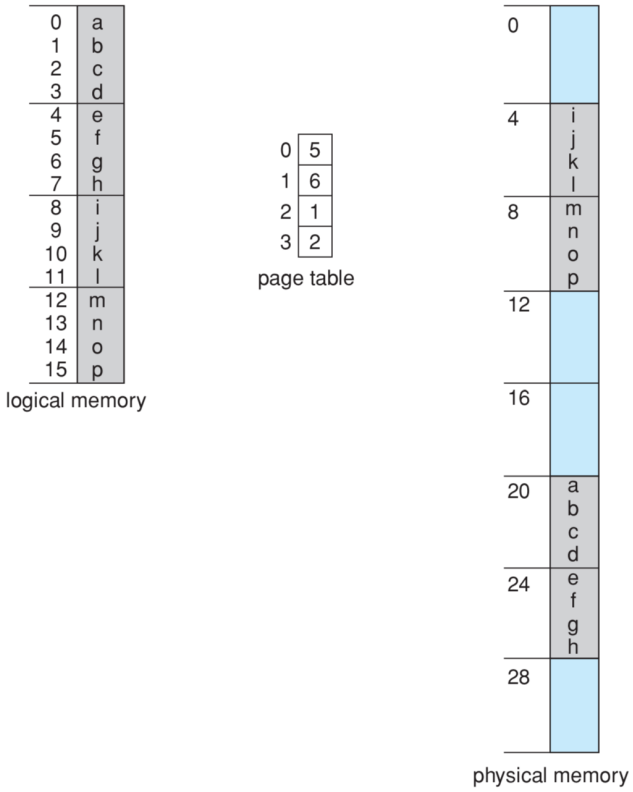
2^{20} = 1048576 = 1 MB of page table entries

Exercise 2: For a 32 bit logical address and a page (frame) size of 4K, what is *n* and how many pages (frames) could be addressed?

m = 32 *n* = 12 (2^{12} = 4K)

(m – n) = 32 - 12 = 20

2^{20} = 1048576 = 1 MB of page table entries



Paging example for a 32-byte memory with 4-byte pages.

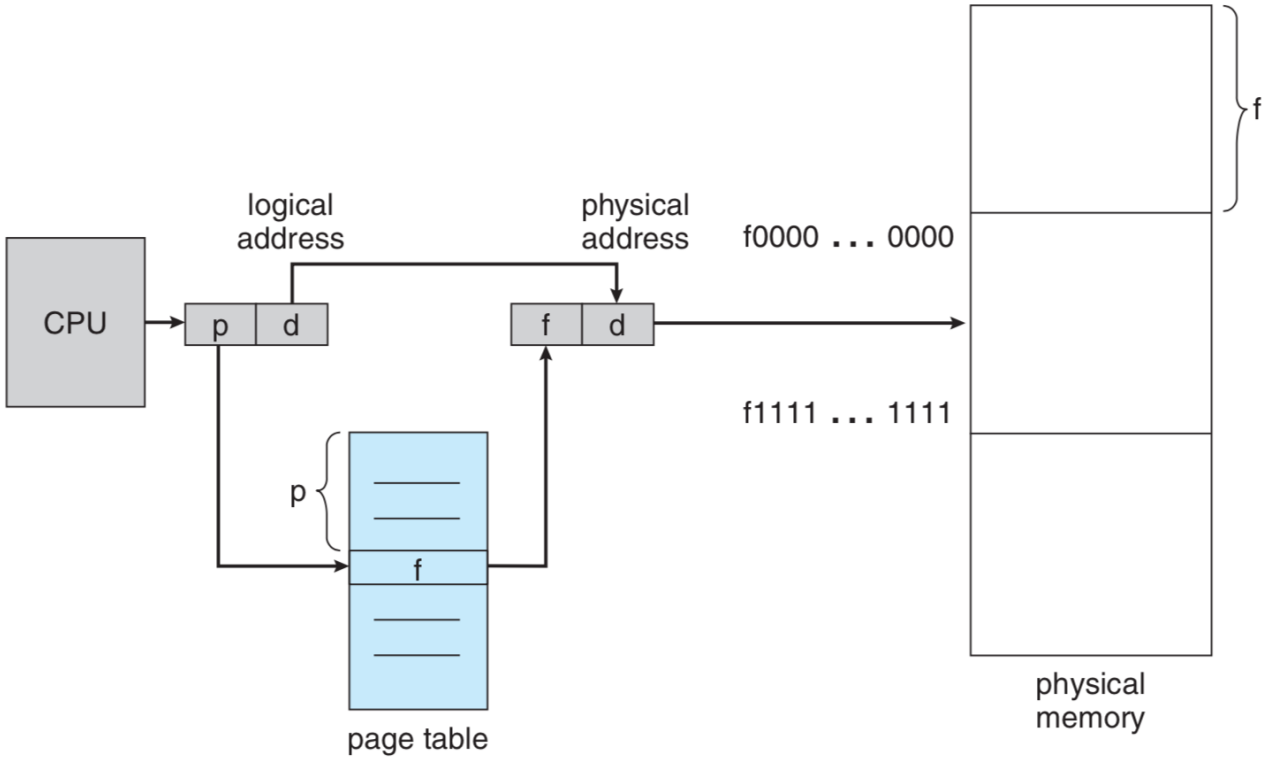
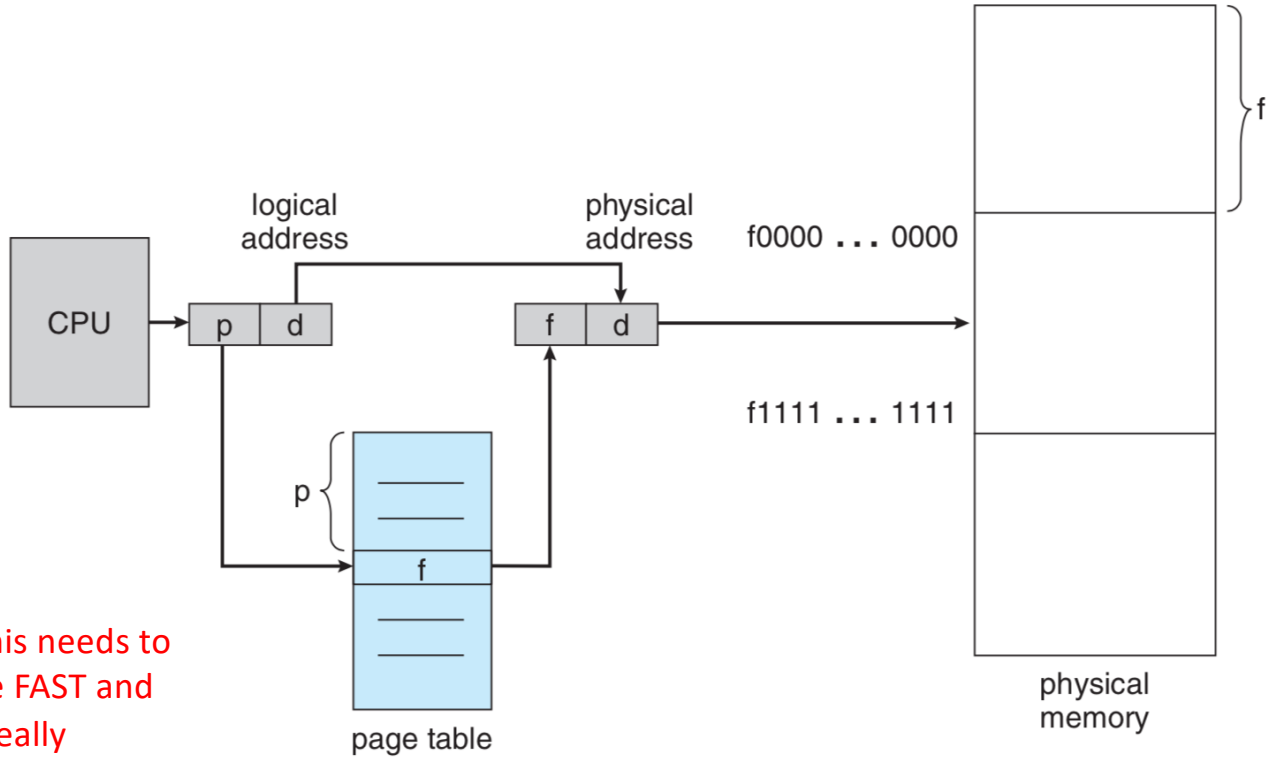


Figure 8.10 Paging hardware.

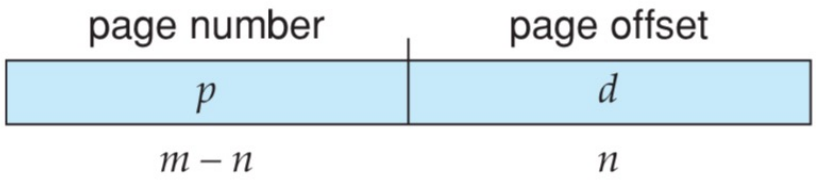


This needs to be FAST and ideally hardware supported

Figure 8.10 Paging hardware.

DEDICATED REGISTERS: Each page table entry COULD be represented by a base register in the CPU with one register per process page. This is not really done except for small memory spaces.

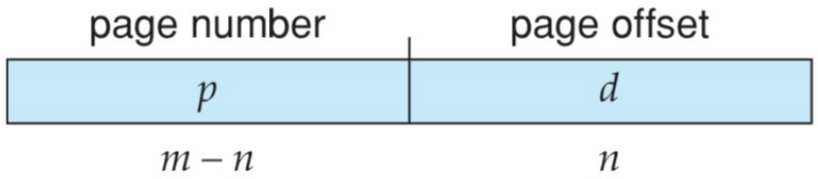
Exercise 3: The PDP-11 had a 16 bit address space and a page size of 8KB. Compute how many frames were available and the maximum size of a page table.



University of CINCINNATI

DEDICATED REGISTERS: Each page table entry COULD be represented by a base register in the CPU with one register per process page. This is not really done except for small memory spaces.

Exercise 3: The PDP-11 had a 16 bit address space and a page size of 8KB. Compute how many frames were available and the maximum size of a page table.



$m = 16$ $n = 13$ $(2^{13} = 8K)$

$(m - n) = 16 - 13 = 3$

$2^3 = 8 = 8$ pages of page table entries… this would mean that 8 base registers would be required… which just so happens to be what the PDP-11 had.

University of CINCINNATI

Remember this? Would you want to use dedicated base registers to solve the page table problem here?

Exercise 2: For a 32 bit logical address and a page (frame) size of 4K, what is *n* and how many pages (frames) could be addressed?

m = 32 *n* = 12 (2^{12} = 4K)

(m – n) = 32 - 12 = 20

2^{20} = 1048576 = 1 MB of page table entries

Page Table Base Register (PTBR): The hardware maintains a SINGLE register that points to a block of memory that contains the page table. The actual page table is in memory, the PTBR points to its first slot.

Exercise 4: Why is this good? Why is this bad?

Good: I don't need a million registers. I just need one. Also, changing ONE register instead of many makes process context switches faster.

Bad: SPEED is terrible. Every logical memory access in user mode requires at least two physical memory accesses. One to read the table to load the base and one to do the user's operation

Page Table Base Register (PTBR): The hardware maintains a SINGLE register that points to a block of memory that contains the page table. The actual page table is in memory, the PTBR points to its first slot.

Exercise 4: Why is this good? Why is this bad?

Page Status / Protection Bits

In addition to FRAME number, page table entries can also contain additional status bits that tell what can, should, or needs, to be done when a page is accessed.

Read/Write: The page is readable only or readable and writeable by the process

Valid/Invalid: The page table entry is “valid” in that the corresponding virtual address actually “active” in the virtual address space. A process trying to access an invalid page generates a trap to the kernel.

Transition Look-Aside Buffer (TLB)+ (PTBR): A TLB is an "associative memory". Associative memories are addressed by a key… NOT by an address. The TLB holds RECENT page lookups. If you give it a page number it knows, it will give you the frame number VERY quickly. If not, the CPU will go to the page table in memory to get the frame. That "new lookup" goes into the TLB to speed up future lookups. TLB MUST be flushed between processes

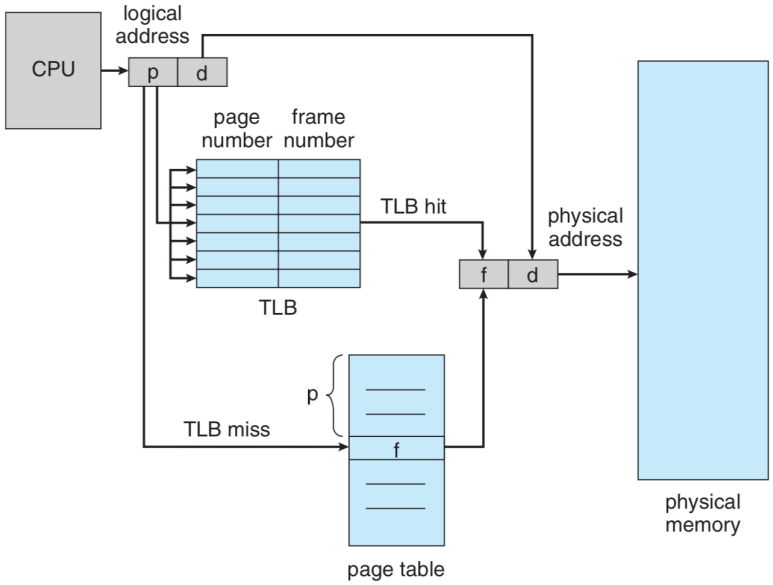


Figure 8.14 Paging hardware with TLB.

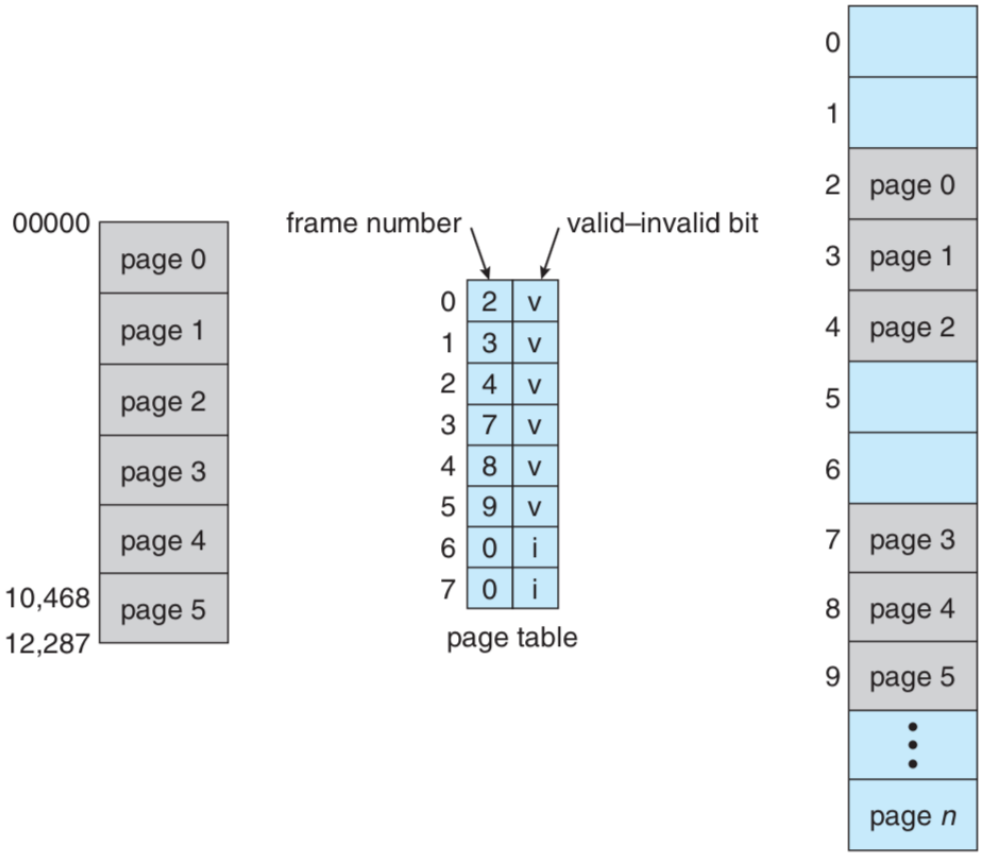


Figure 8.15 Valid (v) or invalid (i) bit in a page table.

University of CINCINNATI

Shared Pages

It is possible for page tables in DIFRERENT processes to point to the same physical frames. This is VERY useful for shared code segments.

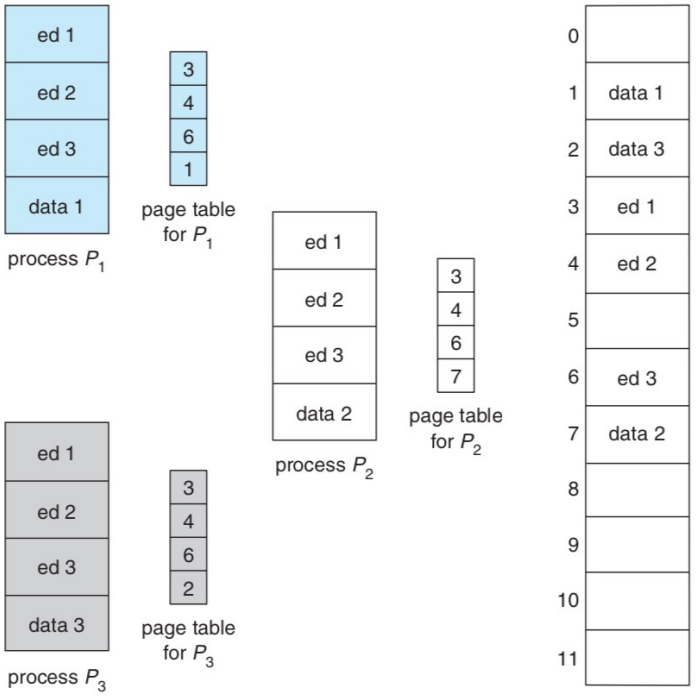
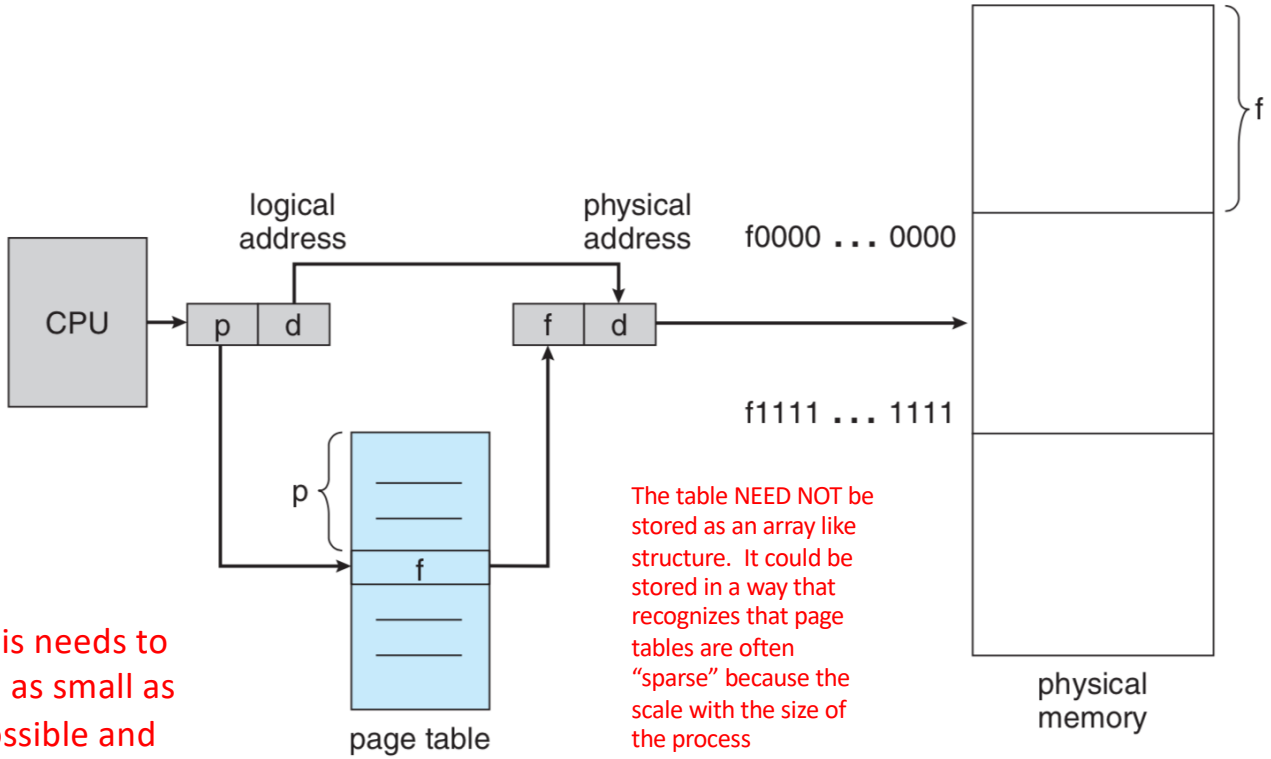


Figure 8.16 Sharing of code in a paging environment.

University of CINCINNATI



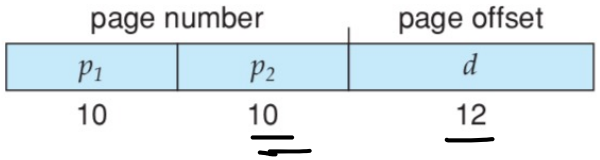
The table NEED NOT be stored as an array like structure. It could be stored in a way that recognizes that page tables are often "sparse" because the scale with the size of the process

This needs to be as small as possible and ideally hardware supported

Figure 8.10 Paging hardware.

Hierarchical Page Tables

It's a page table of page tables… or maybe a page table of page tables of page tables. Depends on how deep you want to stack….



where p_1 is an index into the outer page table and p_2 is the displacement within the page of the inner page table. The address-translation method for this architecture is shown in Figure 8.18. Because address translation works from the outer page table inward, this scheme is also known as a **forward-mapped page table**.

Hierarchical Page Tables

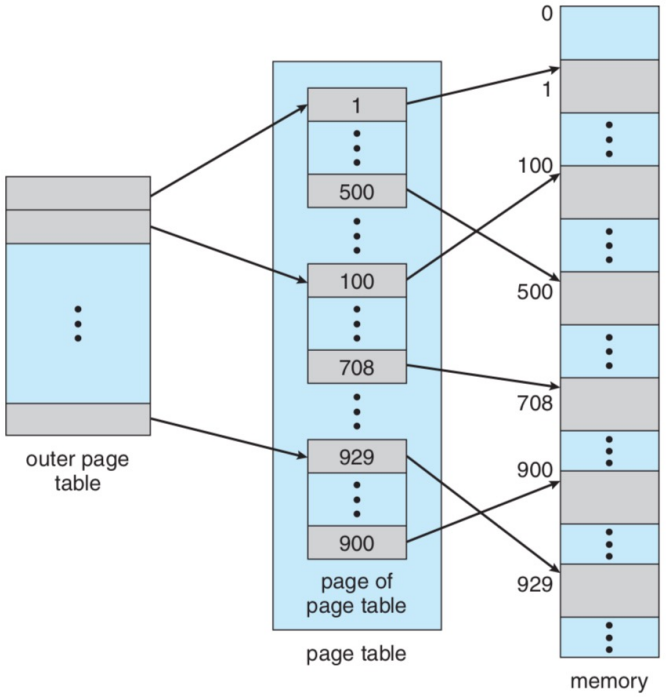


Figure 8.17 A two-level page-table scheme.

Exercise 5: Why is this good? Why is this bad?

Hashed Page Table

A common approach for handling address spaces larger than 32 bits is to use a **hashed page table**, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions). Each element consists of three fields: (1) the virtual page number, (2) the value of the mapped page frame, and (3) a pointer to the next element in the linked list.

The algorithm works as follows: The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared with field 1 in the first element in the linked list. If there is a match, the corresponding page frame (field 2) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. This scheme is shown in Figure 8.19.

Hashed Page Table

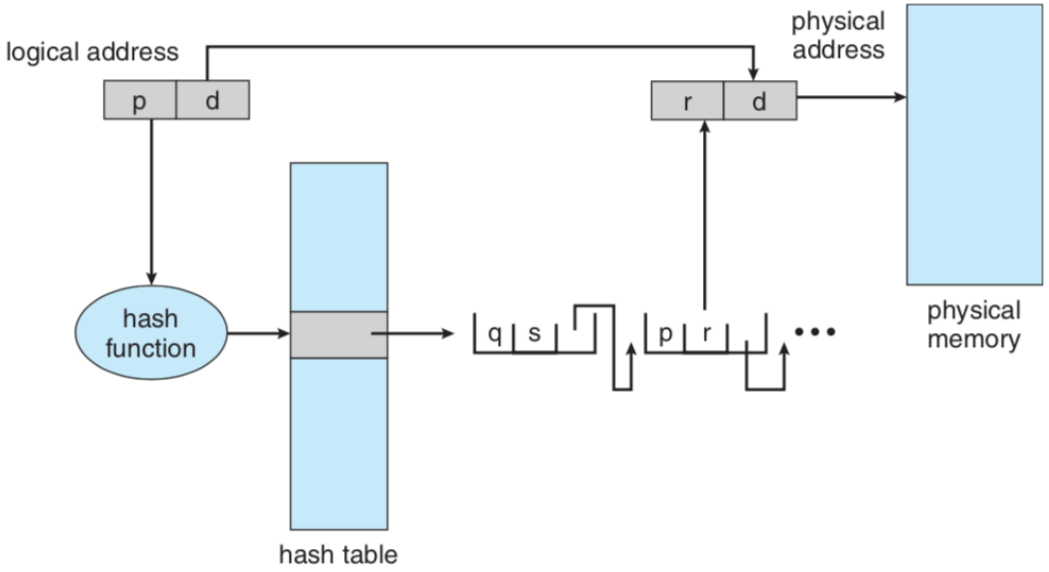
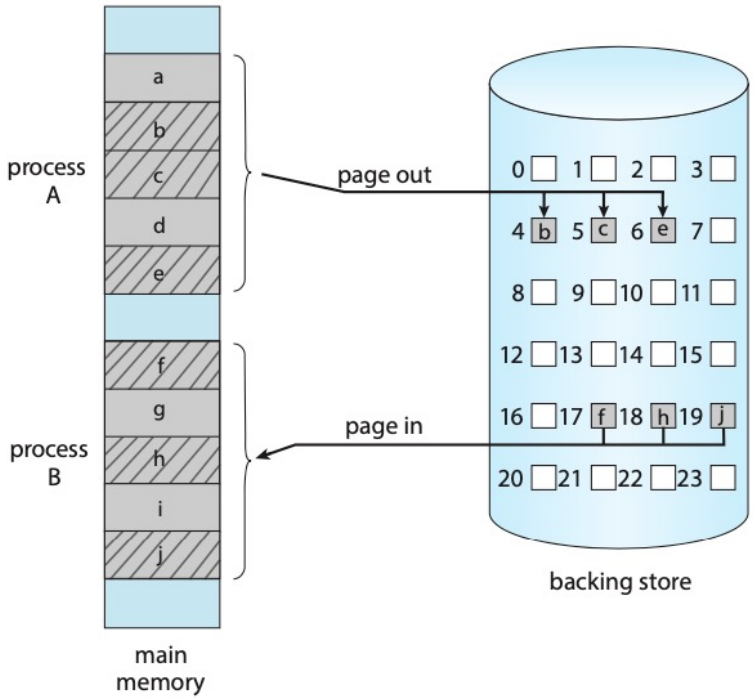


Figure 8.19 Hashed page table.

Paging (Swapping with Paging)



University of CINCINNATI