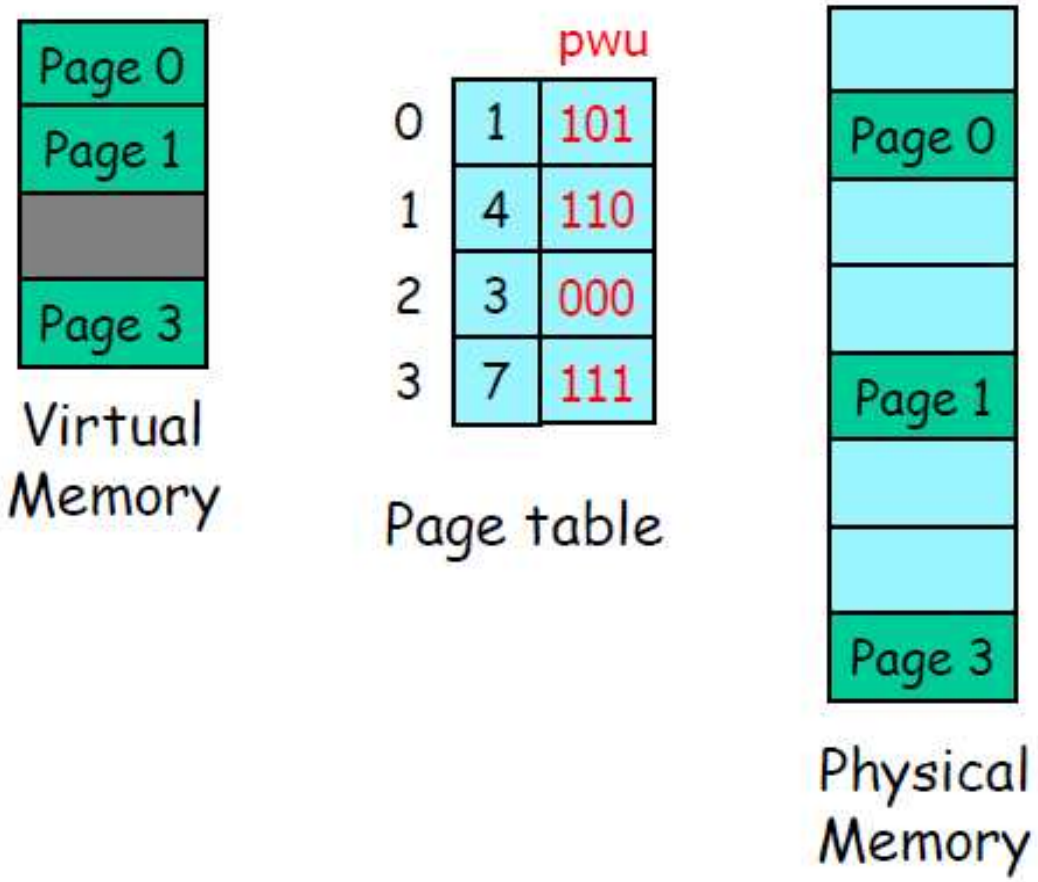
Memory Management (4)

Dr. Jun Zheng

CSE325 Principles of Operating Systems

10/25/2019

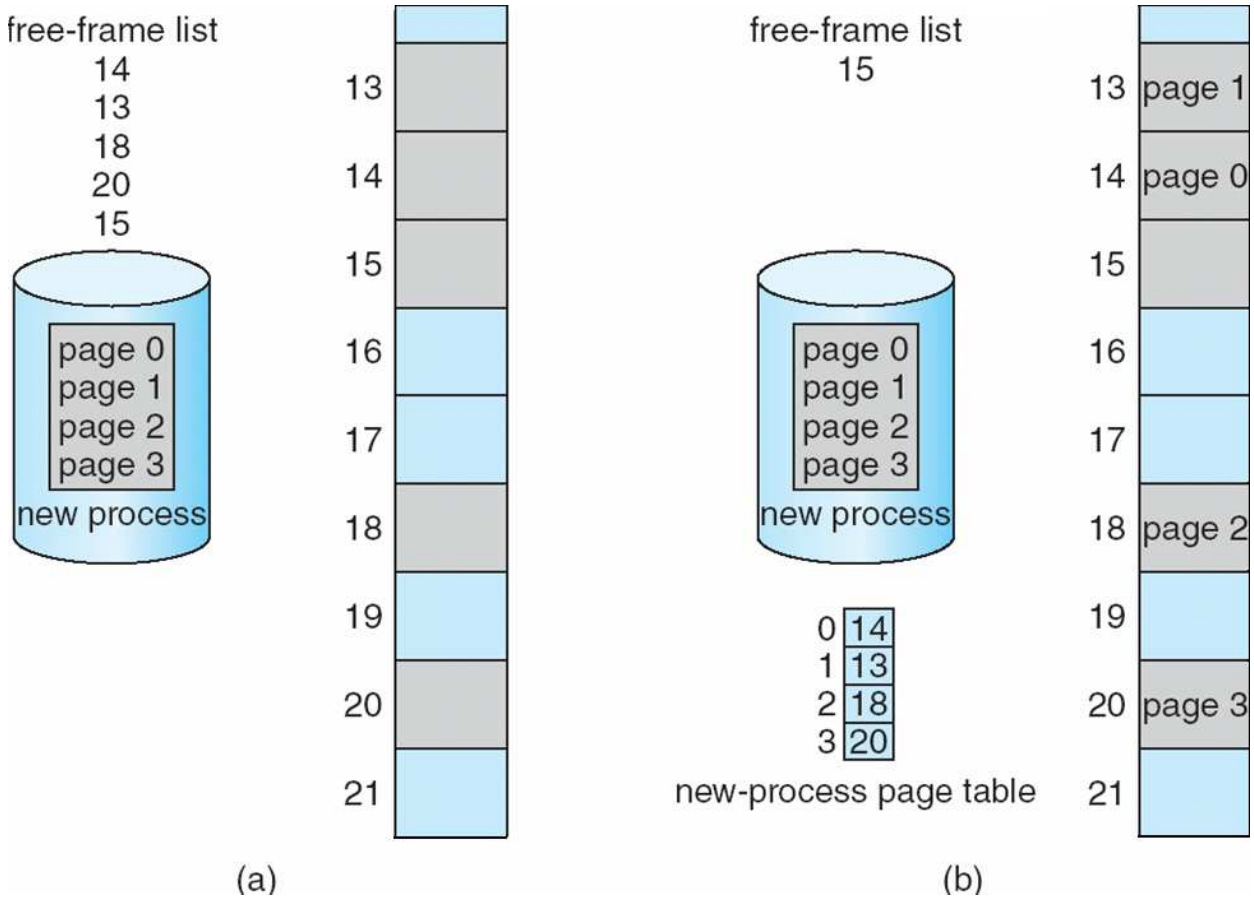
Page Protection Example



Page Allocation

- Free frame management
  - E.g., can put page on a free list
- Allocation policy
  - E.g., one page at a time, from head of free list
- xv6: `kalloc.c`

Page Allocation



Before allocation

After allocation

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Implementation of Page Table

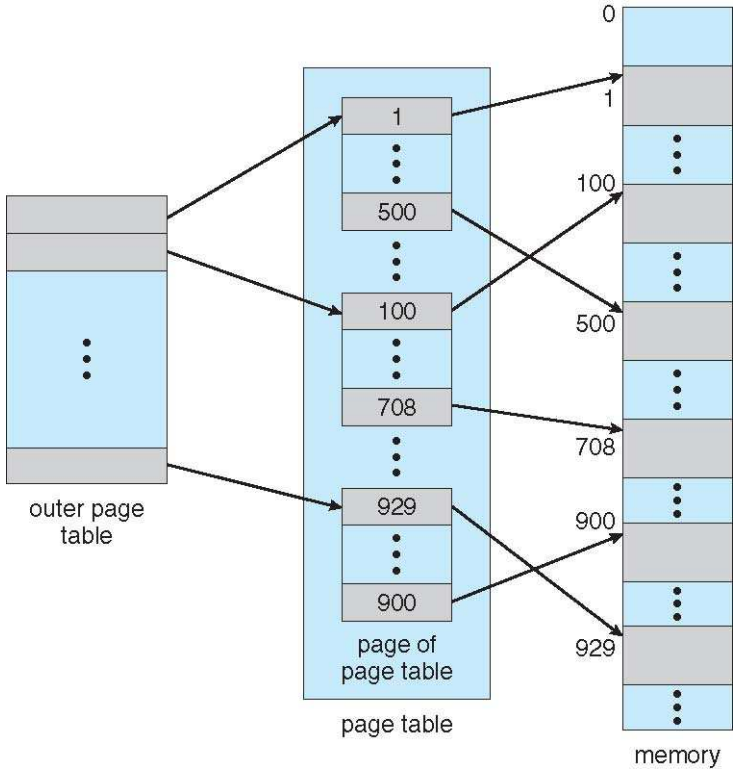
- Page table is kept in main memory
  - **Page-table base register (PTBR)** points to the page table
    - x86: `cr3`
  - **Page-table length register (PTLR)** indicates size of the page table
  - OS stores base in process control block (PCB)
  - OS switches PTBR on each context switch
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Structure of the Page Table

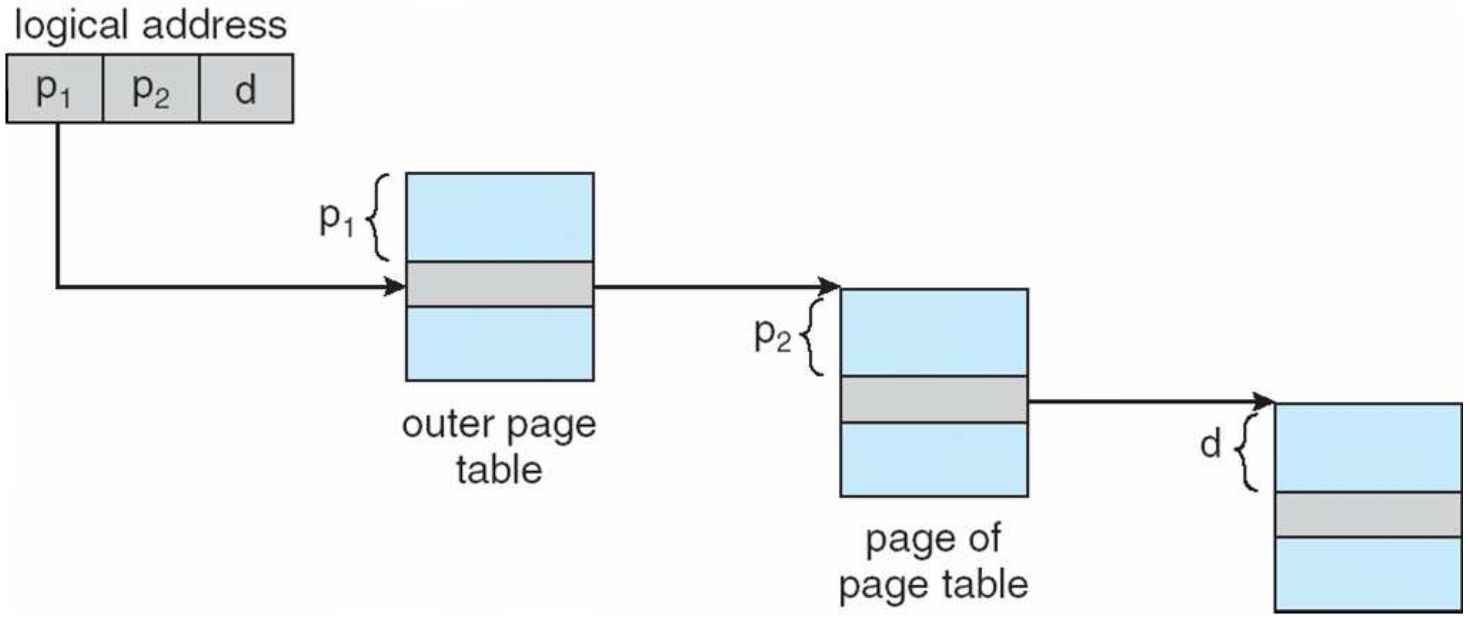
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries (2^{32} / 2^{12})
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    - That amount of memory used to cost a lot
    - Don't want to allocate that contiguously in main memory
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

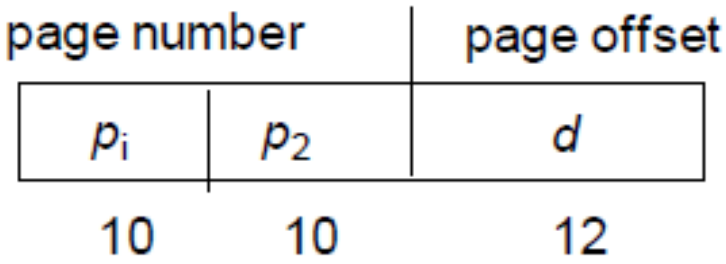


Address Translation with Hierarchical Page Table



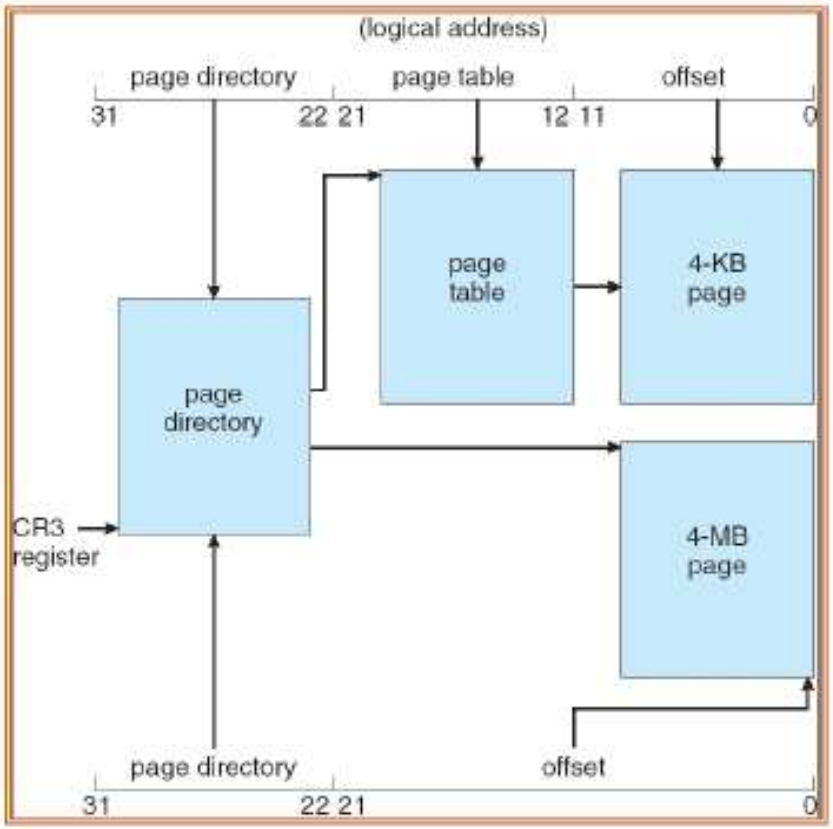
x86 Page Translation

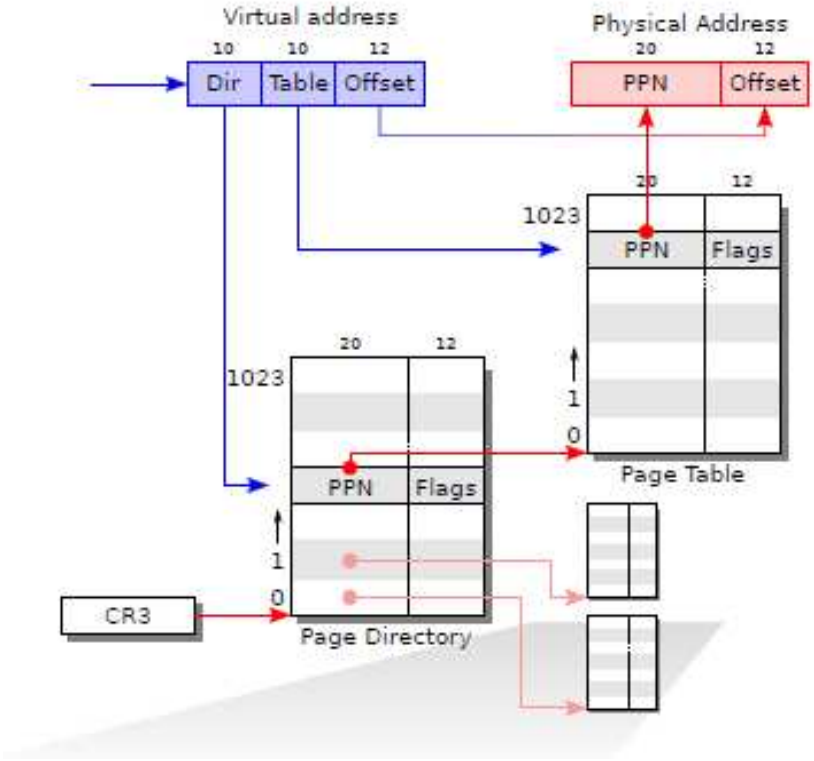
- 32-bit address space, 4 KB page
  - 4KB page -> 12 bits for page offset
- How many bits for 2nd-level page table?
  - Desirable to fit a 2nd-level page table in one page
  - 4KB/4B = 1024 -> 10 bits for 2nd-level page table
- Address bits for top-level page table: 32 – 12 – 10 = 10



NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

x86 Paging Architecture

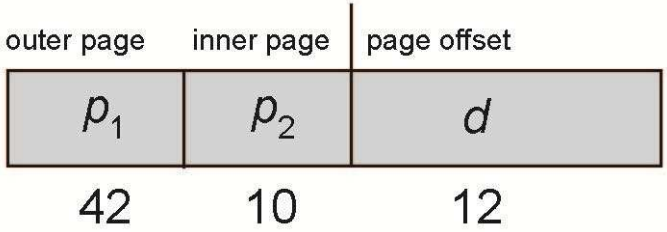




NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

64-bit Logical Address Space

- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like

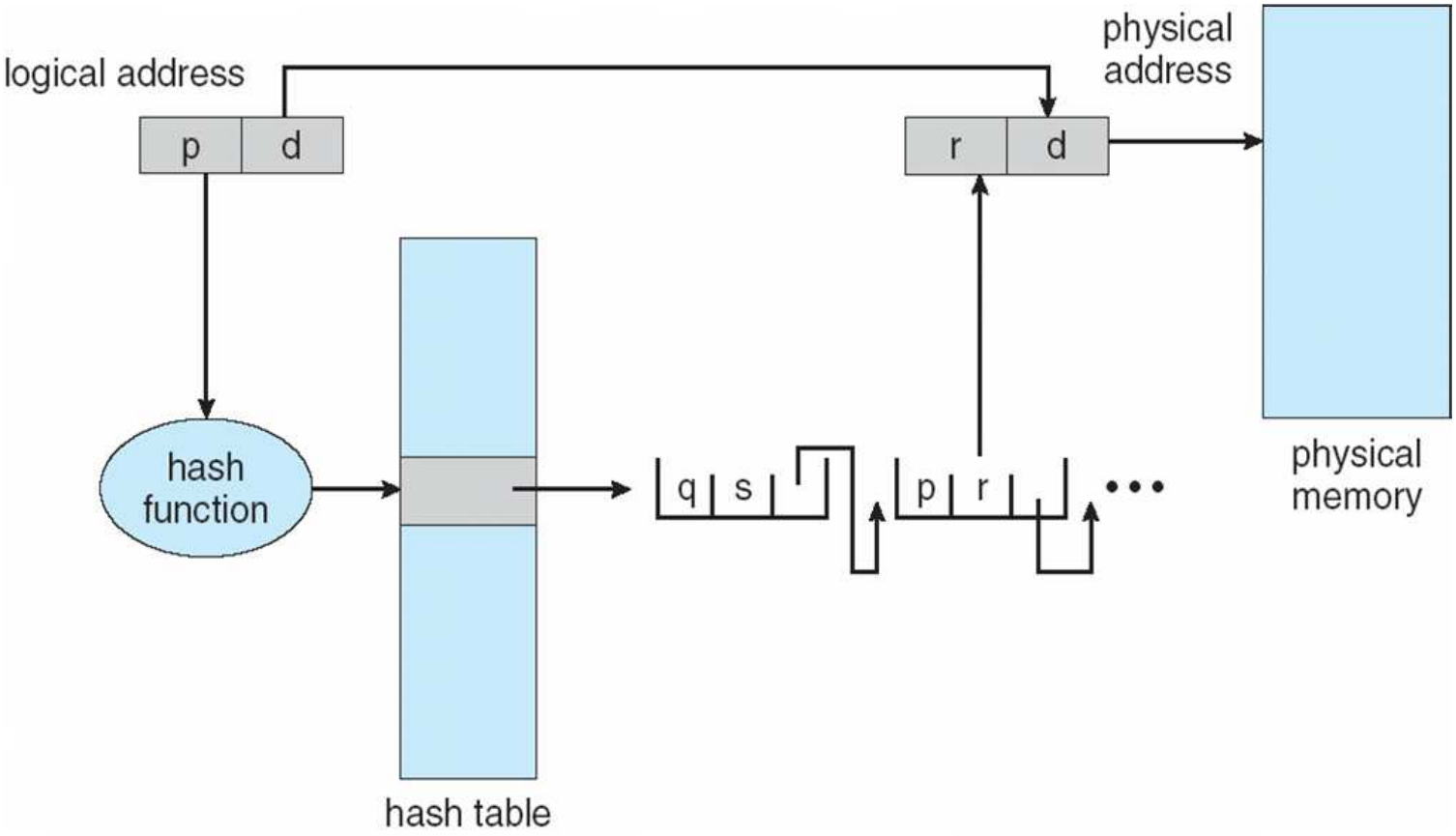


  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

Hashed Page Tables

- Common in address spaces > 32 bits
- The logical page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains (1) the logical page number (2) the value of the mapped page frame (3) a pointer to the next element
- Logical page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables**
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)

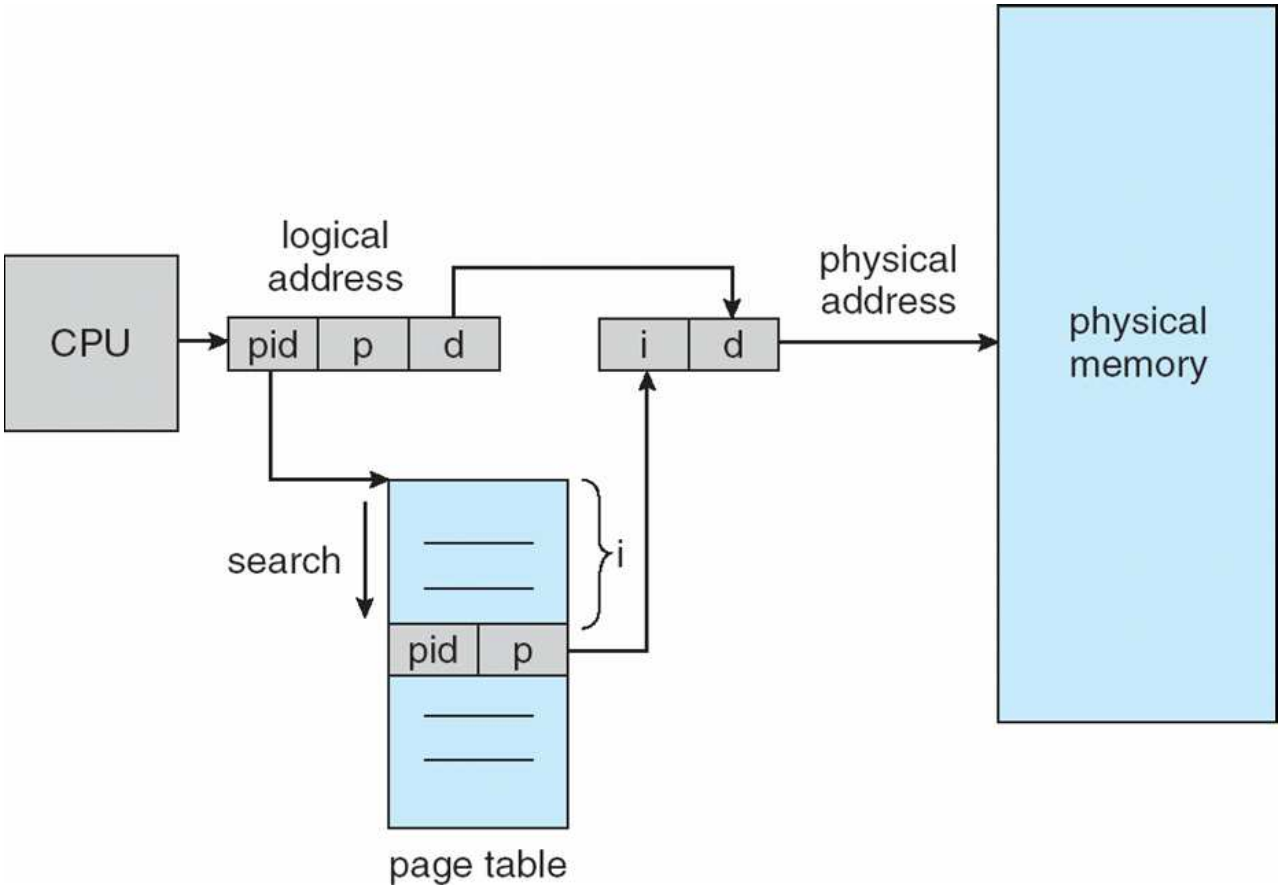
Hashed Page Table



Inverted Page Table

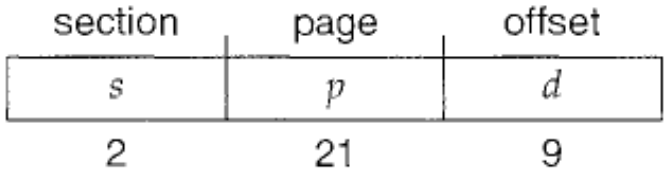
- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a logical address to the shared physical address

Inverted Page Table Architecture



Example

The VAX architecture supports a variation of two-level paging. The VAX is a 32-bit machine with a page size of 512 bytes. The logical address space of a process is divided into four equal sections, each of which consists of 2^{30} bytes. Each section represents a different part of the logical address space of a process. The first 2 high-order bits of the logical address designate the appropriate section. The next 21 bits represent the logical page number of that section, and the final 9 bits represent an offset in the desired page.



How many memory operations are performed when a user program executes a memory-load operation?

Example

Consider a computer system with a 32-bit logical address and 4-KB (2^{12}B) page size. The system supports up *to* 512MB (2^{29}B) of physical memory. How many entries are there in each of the following?

(1) A conventional single-level page table

(2) An inverted page table

Answer

(1) # of pages= # of entries =????

Size of logical address space $= 2^m$ = # of pages × page size

»» 2^{32} = # of pages × 2^{12}

# of pages =2^{32} / 2^{12}= 2^{20} pages

(2)

Size of physical address space = # of frames × frame size

(frame size = page size)

For inverted page table, # of frames = # of entries

2^{29}= # of frames × 2^{12}

# of frames =2^{29} / 2^{12}= 2^{17}

# of entries = 2^{17}

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY