COA

Chapter 08: | OS Support – Paging continued

Prof. Dr. Mostafa Elhosseini

https://youtube.com/drmelhosseini

AGENDA

- Page Table Structure
- Translation Lookaside Buffer
- Inverted page table

Hardware Support

- Each operating system has its own methods for storing page tables.
  - Most allocate a **page table for each process**.
- A **pointer** to the page table is stored with the other **register** values in the process control block.
  - When the **dispatcher** is told to start a process, it must **reload** the user registers.

Hardware Support

- The hardware implementation of the page table can be done in several ways
  - The page table is implemented as a **set of dedicated registers**
  - **Instructions to load or modify** the page-table registers are, of course, **privileged**
  - The **DEC PDP-11** is an example of such an architecture
  - The address consists of 16 bits, and the page size is 8 KB. The page table thus consists of **eight entries** that are kept in fast registers

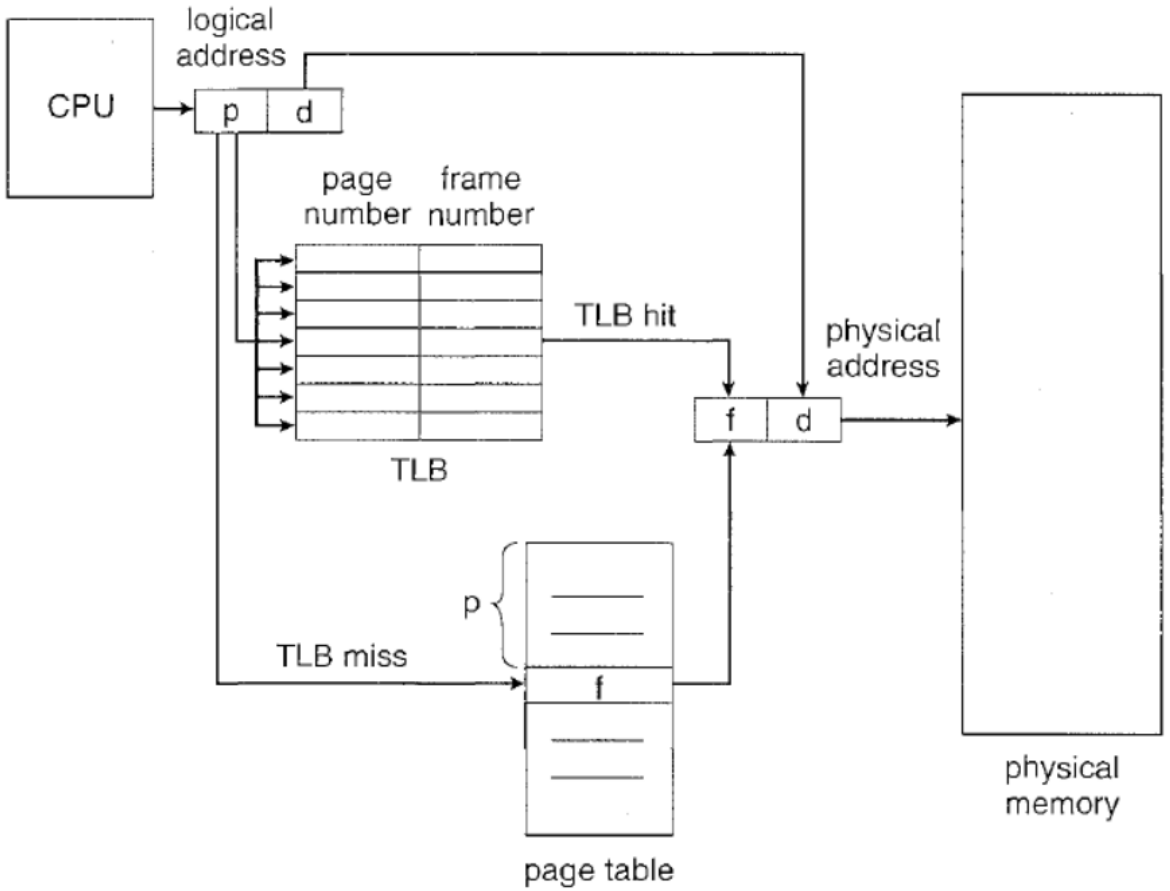
Hardware Support

- The **use of registers** for the page table is satisfactory if the page table is reasonably **small**.
- Most contemporary computers, however, allow the page table to be very large (for example, **1 million entries**).
  - The page table is kept in main memory, and a Page Table Base Register **PTBR** points to the page table
- The **problem** with this approach is the **time required** to access a user memory location

Translation look-aside buffer (TLB)

- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts: a **key** (or tag) and a **value**.
- The **search is fast**; the hardware, however, is **expensive**.
- Typically, the **number of entries** in a TLB is **small**, often numbering between 64 and 1,024
- The TLB **contains only a few** of the page-table entries
  - If the page number is found, its frame number is immediately available
  - If the page number is not in the TLB (known as a TLB miss)

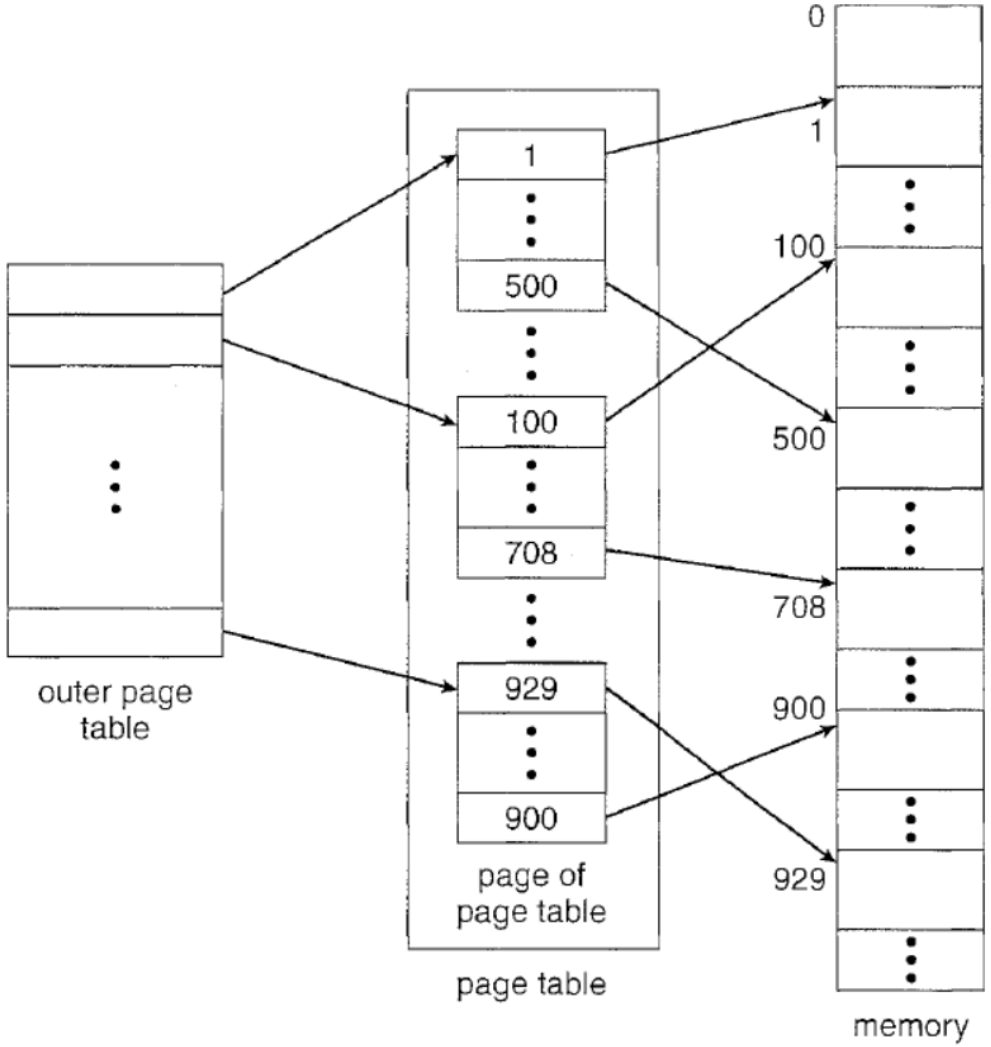
Translation look-aside buffer (TLB)



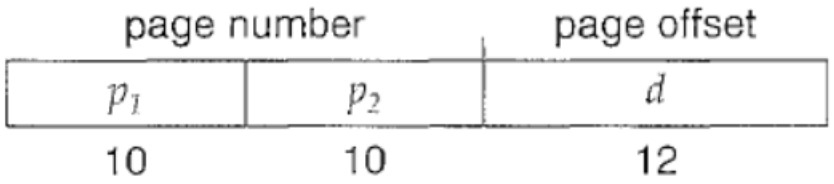
Two-level paging algorithm

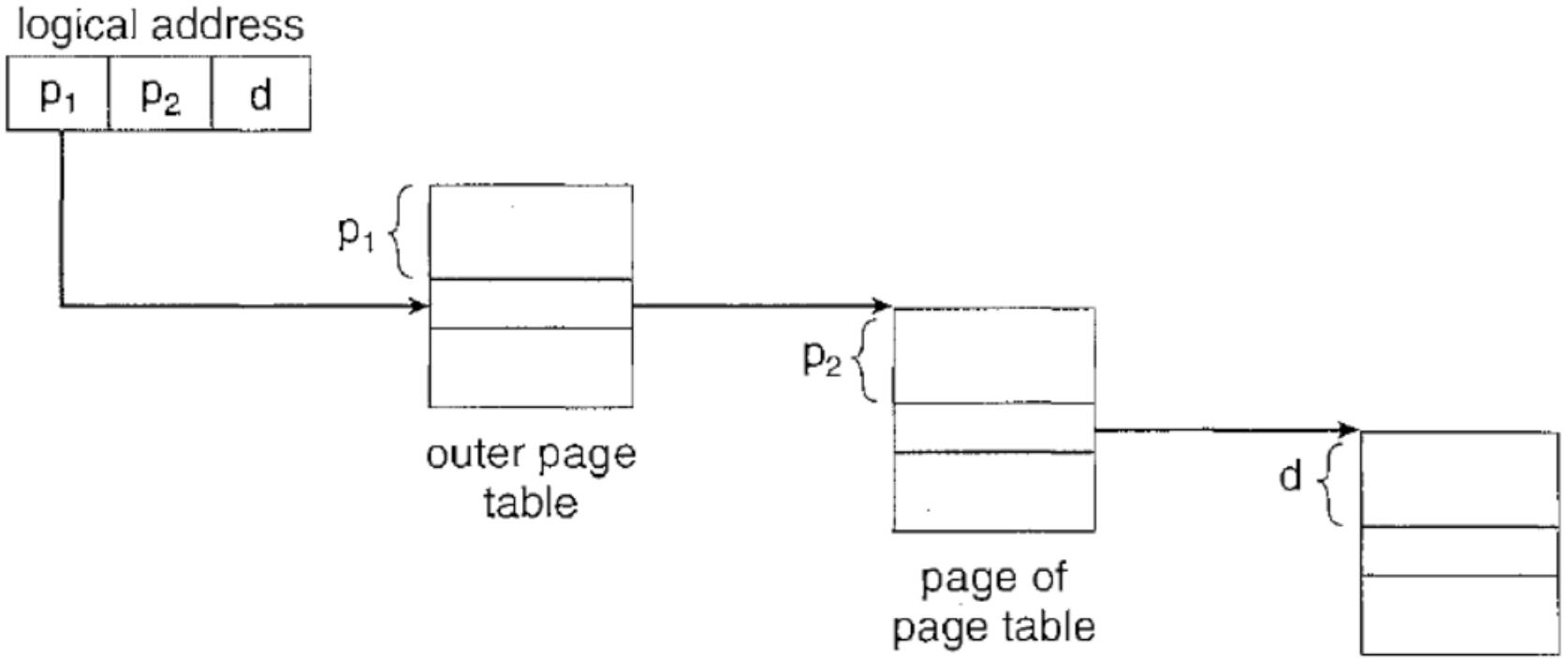
- The page table itself is also paged
  - For example, consider again the system with a **32-bit logical address space** and a **page size of 4 KB**.
  - A logical address is divided into a page number consisting of **20 bits** and a page offset consisting of **12 bits**.
  - Because we **page the page table**, the **page number** is further **divided** into a **10-bit page number** and a **10-bit page offset**.

Two-level paging algorithm



Two-level paging algorithm

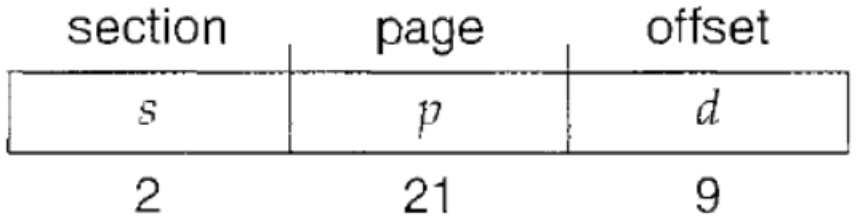




Two-level paging algorithm

- Because address translation works from the outer page table inward, this scheme is also known as a **forward-mapped page table**
  - 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**.

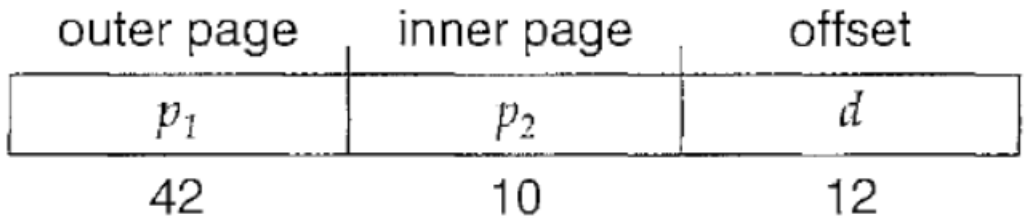
Two-level paging algorithm



- The next **21 bits** represent the logical page number of that section, and the **final 9 bits** represent an offset in the desired page
- By partitioning the page table, the operating system can **leave partitions unused** until a process needs them
- **Even when this scheme is used**, the size of a one-level page table for a VAX process using one section is **2^{21} bits * 4** bytes per entry= **8MB**.
- To further reduce main-memory use, the VAX **pages the user-process page tables**

Two-level paging algorithm

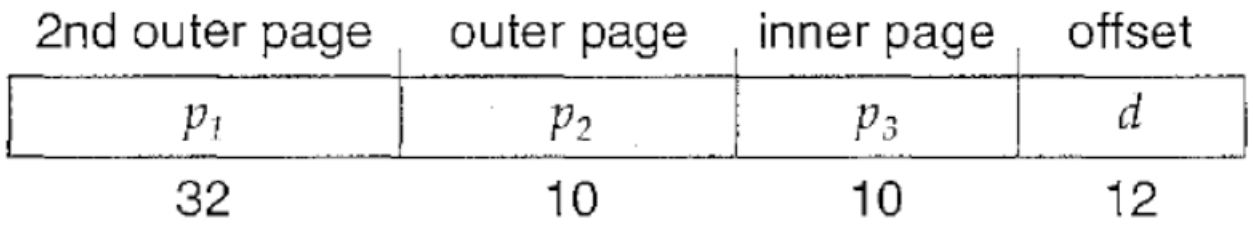
- For a system with a **64-bit logical address space**, a **two-level paging scheme is no longer appropriate**.
- To illustrate this point, let us suppose that the page size in such a system is 4 KB (2^{12})
- the page table consists of up to 2^{52} entries



- The outer page table consists of 2^{42} entries, or 2^{44} bytes. The obvious way to avoid such a large table is to **divide the outer page table into smaller pieces**

Three-level paging algorithm

- Suppose that the outer page table is made up of standard-size pages (2^{10} entries, or 2^{12} bytes). In this case, a 64-bit address space is still **daunting**:

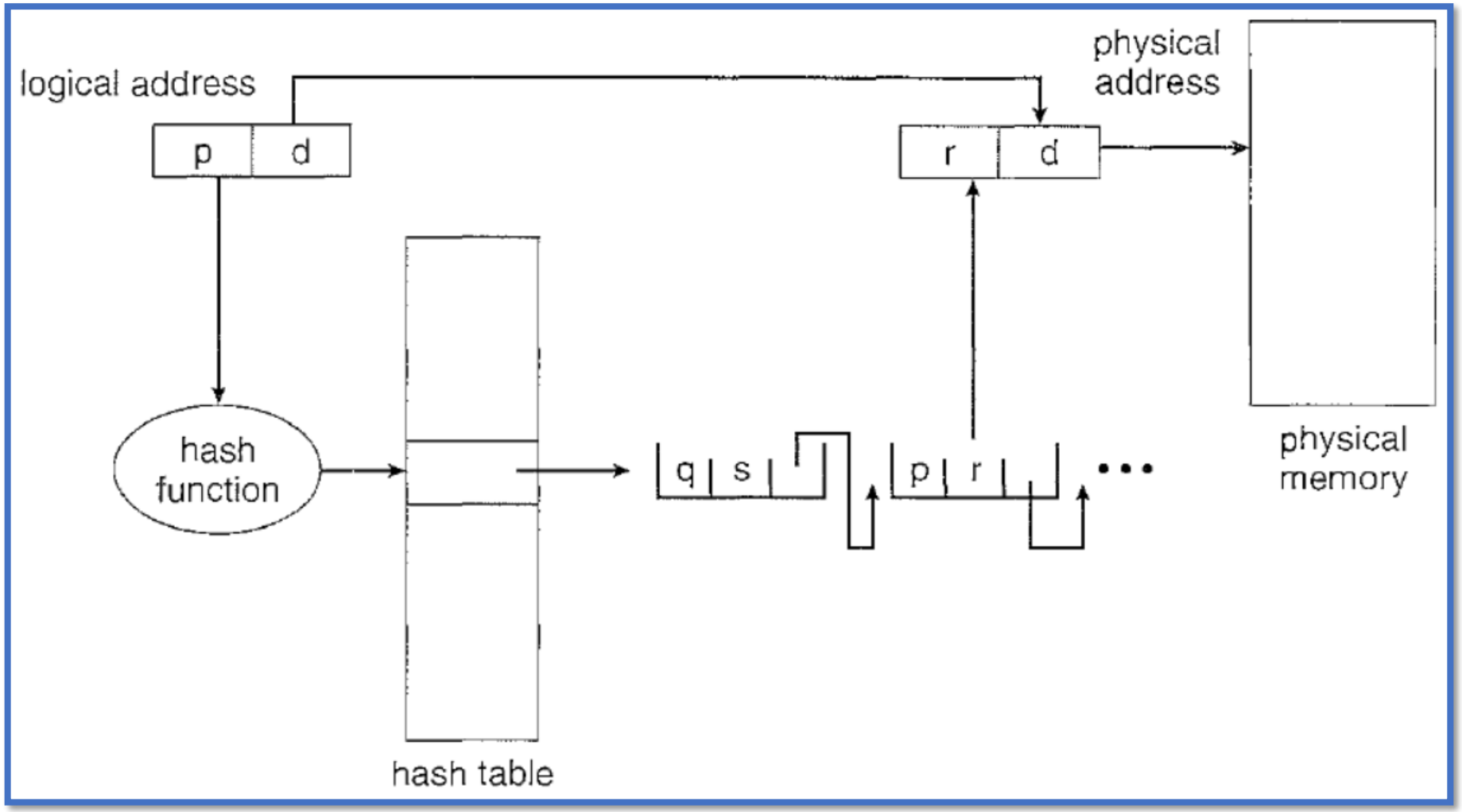


- The outer page table is still 2^{34} bytes in size
- The 64-bit UltraSPARC would require **seven levels** of paging-a **prohibitive number** of memory accesses.
- **Hierarchical** page tables are generally considered **inappropriate** for **64-bit** architectures

Hashed Page Table

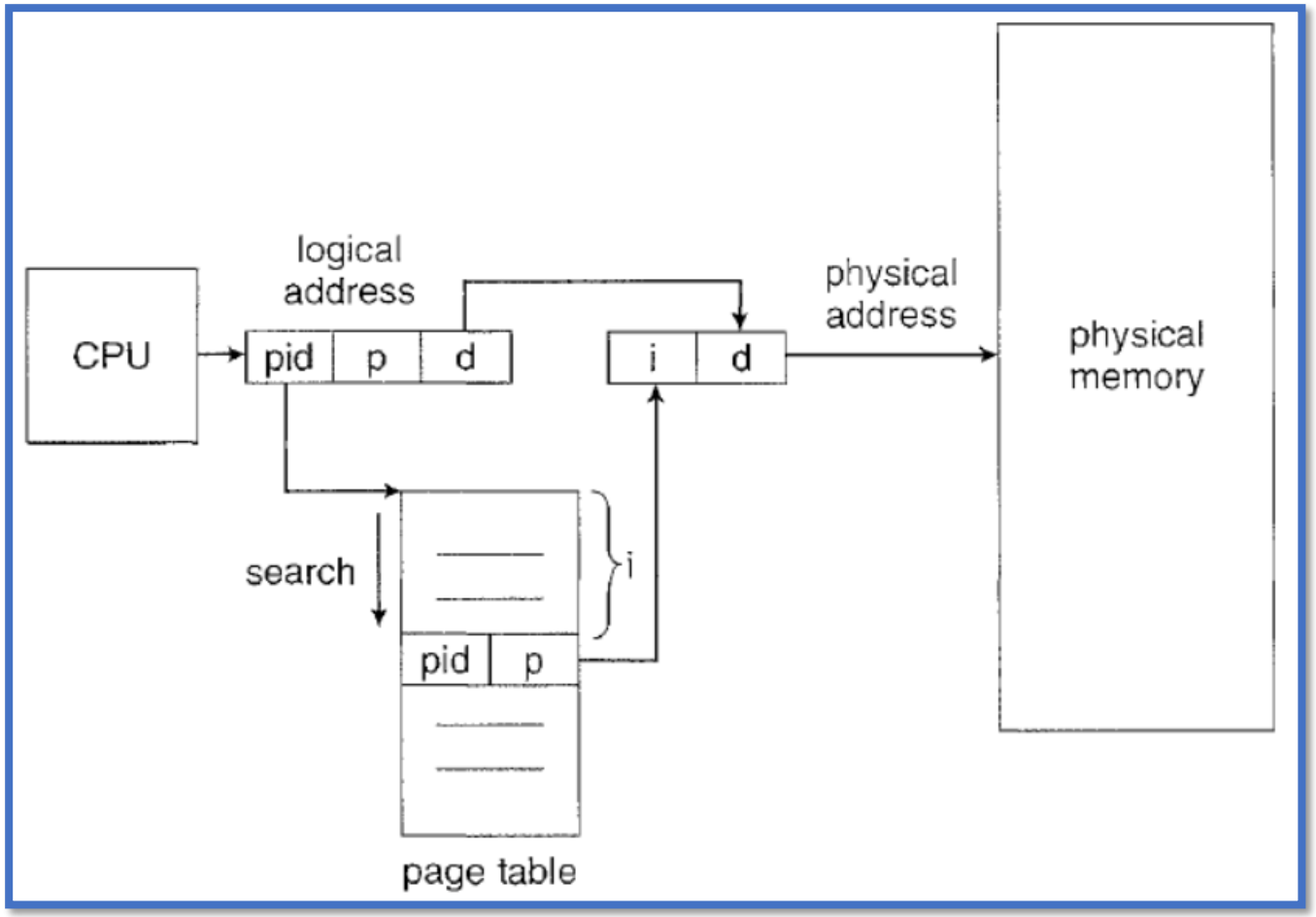
- Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions)
- Each element consists of three fields:
  - The virtual page number,
  - The value of the mapped page frame, and
  - A pointer to the next element in the linked list
- The **algorithm** 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

Hashed Page Table



Inverted Page Table

- An **inverted page table** has **one entry** for each **real page** (or **frame**) of memory
- **only one page table** is in the system, and it has only one entry for each page of physical memory



Inverted Page Table + Hash Table

- The **page number** portion of a virtual address is **mapped** into a **hash value** using a simple **hashing function**
- The **hash value** is a **pointer** to the **inverted page table**, which contains the page table entries.
- There is **one entry** in the **inverted page table** for each **real memory page** frame rather than one per virtual page.
  - Thus a **fixed proportion of real memory** is required for the tables **regardless** of the number of processes or virtual pages supported

Inverted Page Table + Hash Table

- Because more than one virtual address may map into the same hash table entry, a chaining technique is used for managing the overflow. The hashing technique results in chains that are typically short—between one and two entries