Memory Management

Dr Tran Duc Minh and Dr Hung Tran

DATCOM Lab
Faculty of Data Science and Artificial Intelligence
College of Technology, National Economics University
Email: minhdt@neu.edu.vn, hung.tran@neu.edu.vn

April 21, 2025

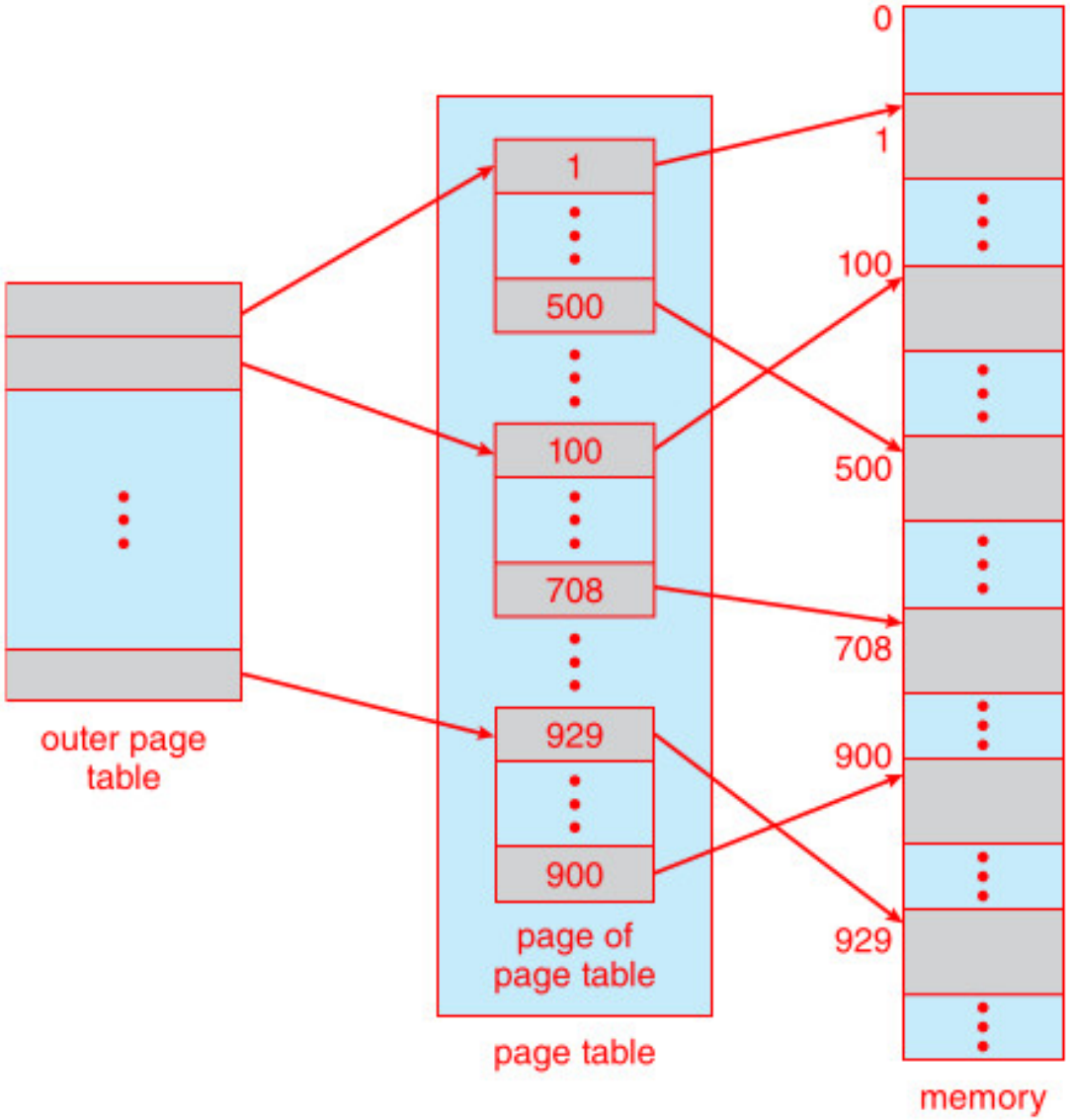
Outline

1. Structure of the Page Table
2. Swapping

Hierarchical Paging

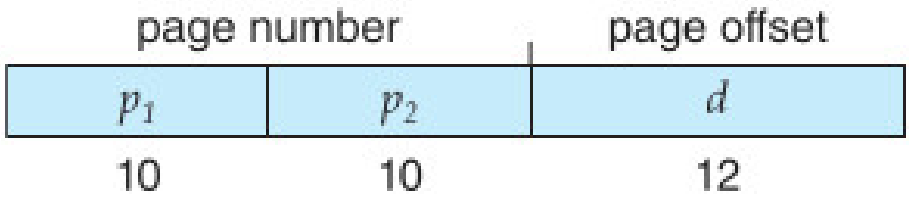
- Most modern computer systems support a large logical address space (2^{32} to 2^{64}).
- In such an environment, the page table itself becomes excessively large.
- For example: Consider a system with a 32-bit logical address space.
  - If the page size is 4 KB (2^{12}), then a page table may consist of over 1 million entries ($2^{20} = 2^{32}/2^{12}$).
  - Assuming that each entry consists of 4 bytes, each process may need up to 4 MB of physical address space for the page table alone.
- We would not want to allocate the page table contiguously in main memory.
- One simple solution to this problem is to divide the page table into smaller pieces.
- We can accomplish this division in several ways.

Two-Level Paging Algorithm



Two-Level Paging Algorithm

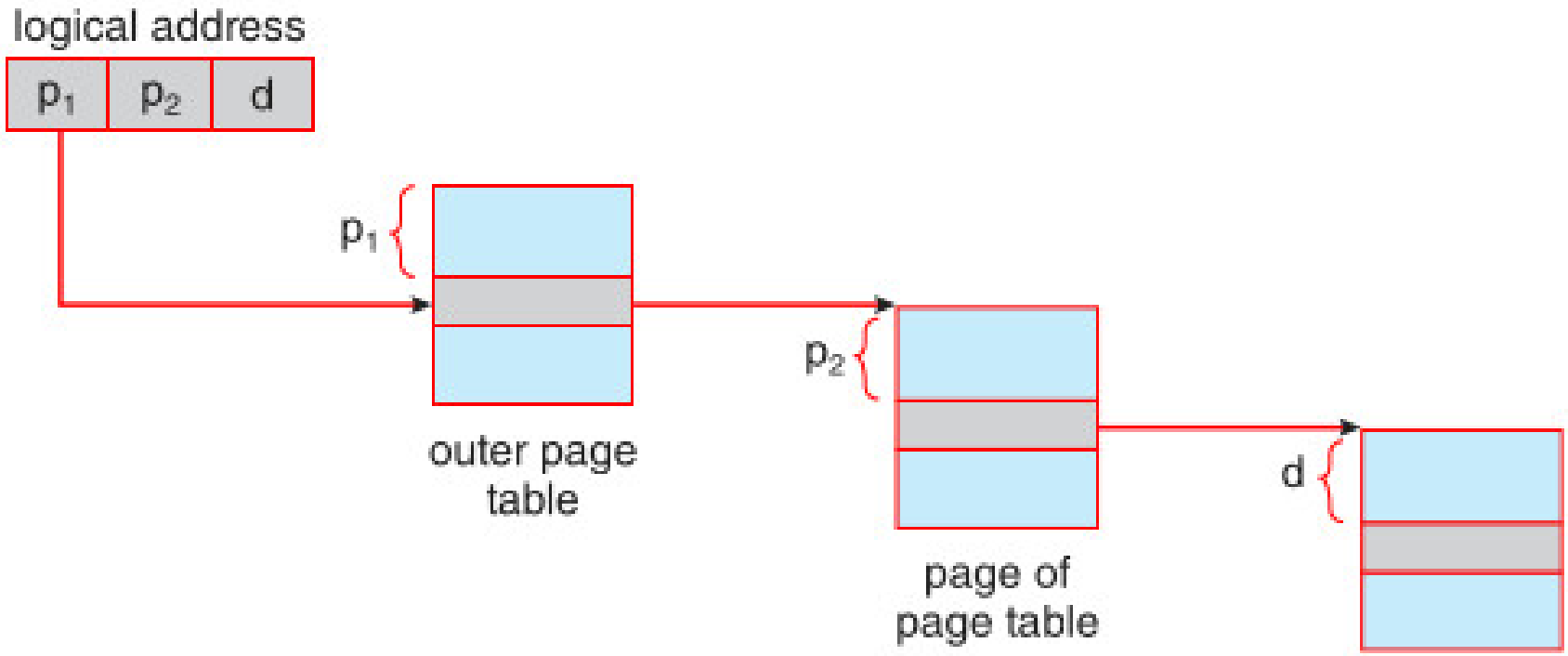
- Consider a system with a 32-bit logical address space and a page size of 4 KB.
- A logical address is divided into a page number (20 bits) and a page offset (12 bits).
- Since we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset. Thus, a logical address is structured as follows:



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.

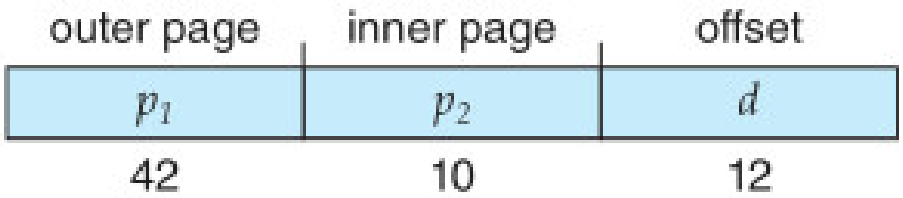
- Since address translation works from the outer page table inward, this scheme is known as a **forward-mapped** page table.

Two-Level Paging Algorithm



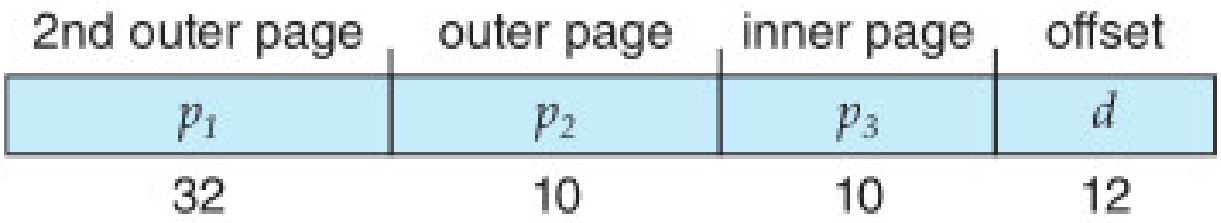
Two-Level Paging in 64-bit Systems

- For a system with a 64-bit logical address space, a two-level paging scheme is no longer appropriate.
- With a page size of 4 KB (2^{12}), the page table consists of up to 2^{52} entries.
- In a two-level paging scheme, the inner page tables can contain 2^{10} 4-byte entries.
- The outer page table would consist of 2^{42} entries, or 2^{44} bytes.



Three-Level Paging in 64-bit Systems

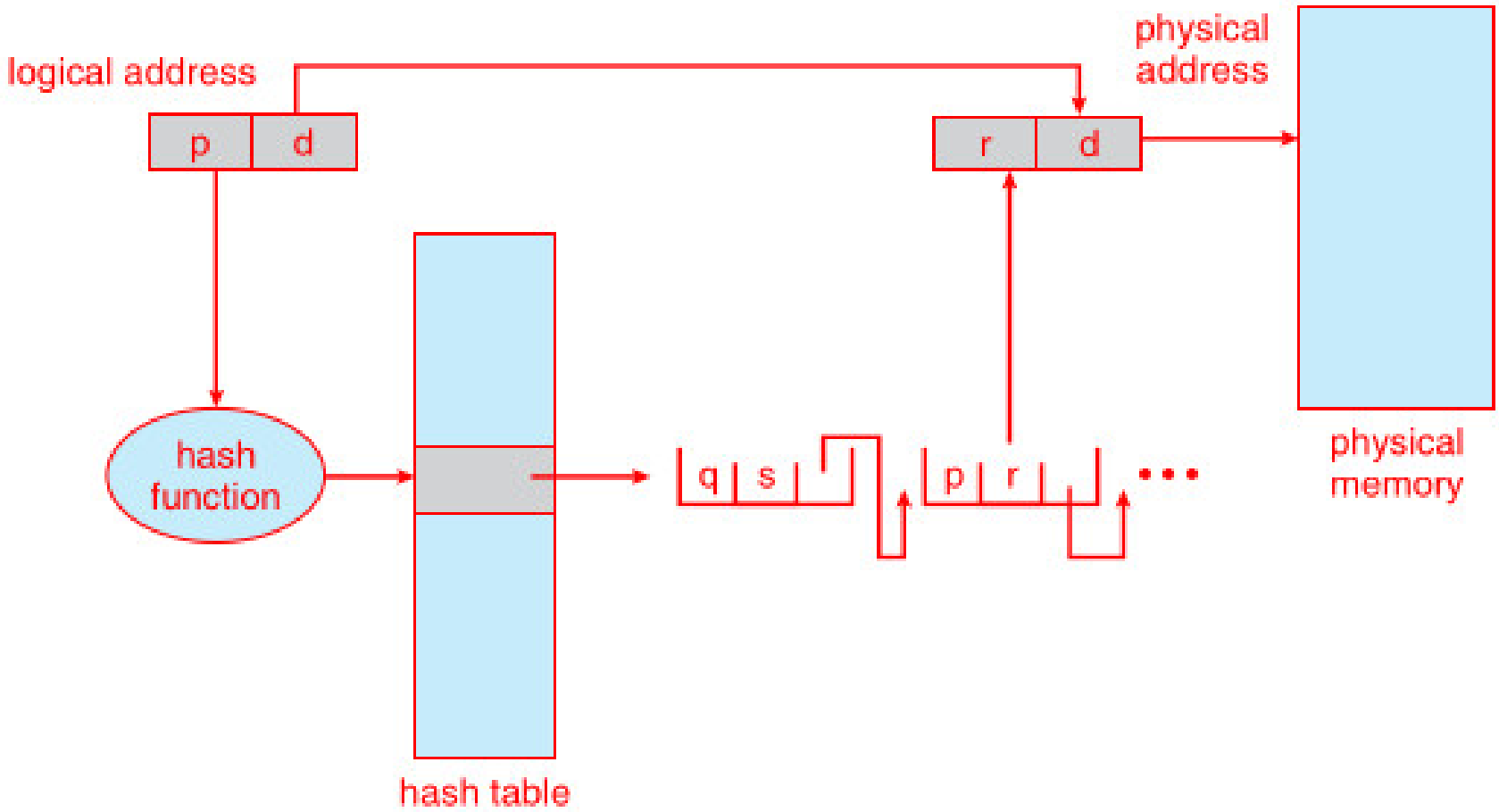
- For example: We can page the outer page table, giving a three-level paging scheme.
  - Suppose the outer page table is made of standard-size pages (2^{10} entries, or 2^{12} bytes).
  - In this case, a 64-bit address space still results in an outer page table of 2^{34} bytes (16 GB) in size.
    - ★ And possibly 4 memory access to get to one physical memory location



Hashed Page Table for Large Address Spaces

- Address spaces larger than 32 bits is to use a **hashed page table** with the hash value is 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,
  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 of the linked list.
  - If there is a match, the corresponding page frame (field 2) is used to form the desired physical address.
  - If no match, subsequent entries in the linked list are searched for a matching virtual page number.

Hashed Page Table for Large Address Spaces



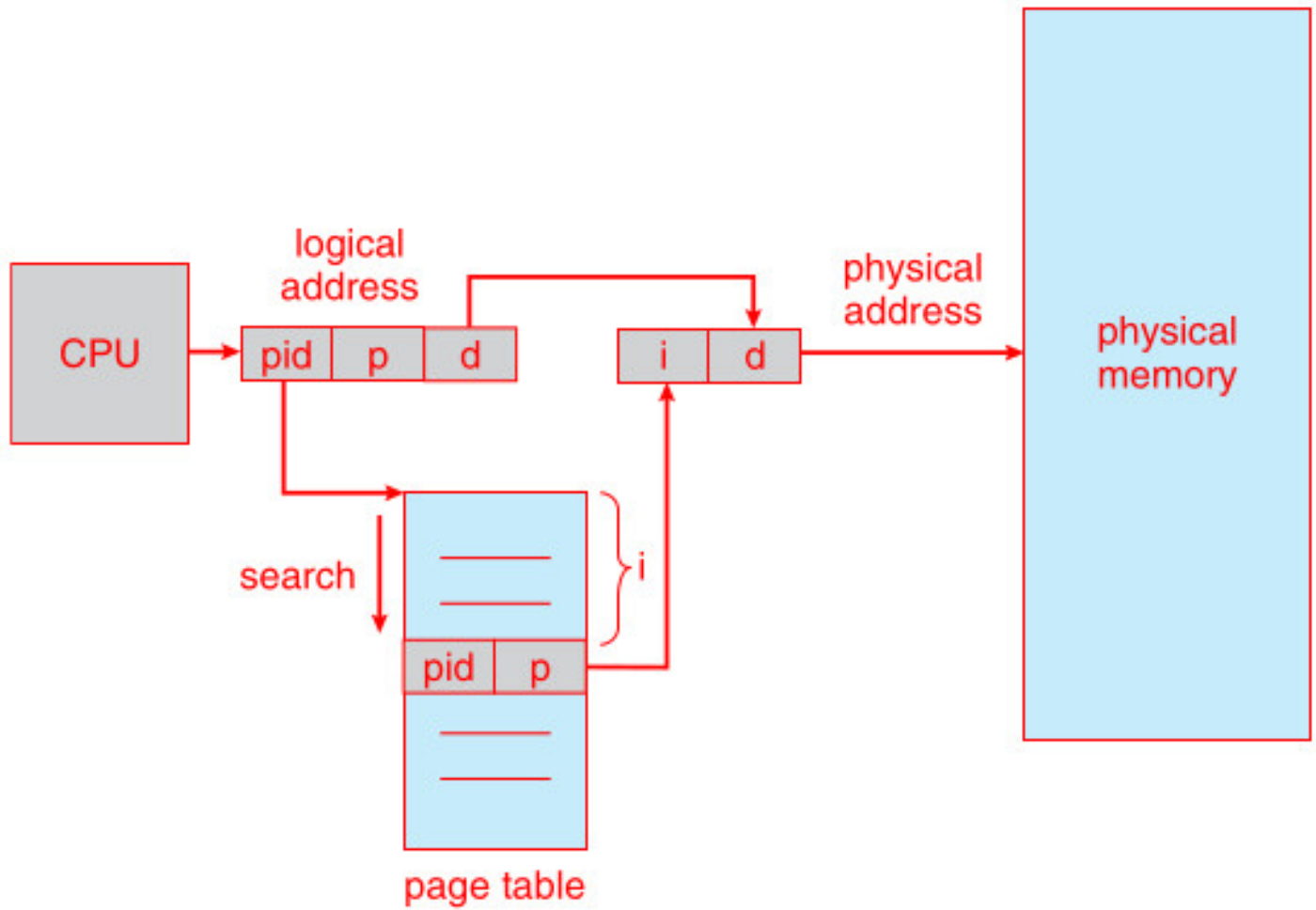
Clustered Page Tables for 64-bit Systems

- A variation of the hashed page table scheme that is useful for 64-bit address spaces.
- This variation uses **clustered page tables**, which are similar to hashed page tables except that each entry in the hash table refers to several pages (such as 16) rather than a single page.
- A single page-table entry can store the mappings for multiple physical-page frames.
- Clustered page tables are particularly useful for **sparse** address spaces, where memory references are noncontiguous and scattered throughout the address space.

Inverted Page Tables

- **Traditional Page Tables**: One entry per virtual page per process, leading to high memory usage due to many unused entries.
- **Inverted Page Table**:
  - One entry per physical frame, not per virtual page.
  - Each entry stores the virtual page number and process ID (PID).
  - Size depends on physical memory, not virtual address space.
- **Advantages**: Reduces memory consumption for page tables.
- **Operation**: OS searches the table to map virtual addresses to physical frames.
- **Usage**: Common in systems like 64-bit UltraSPARC and PowerPC, often with an address-space identifier.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

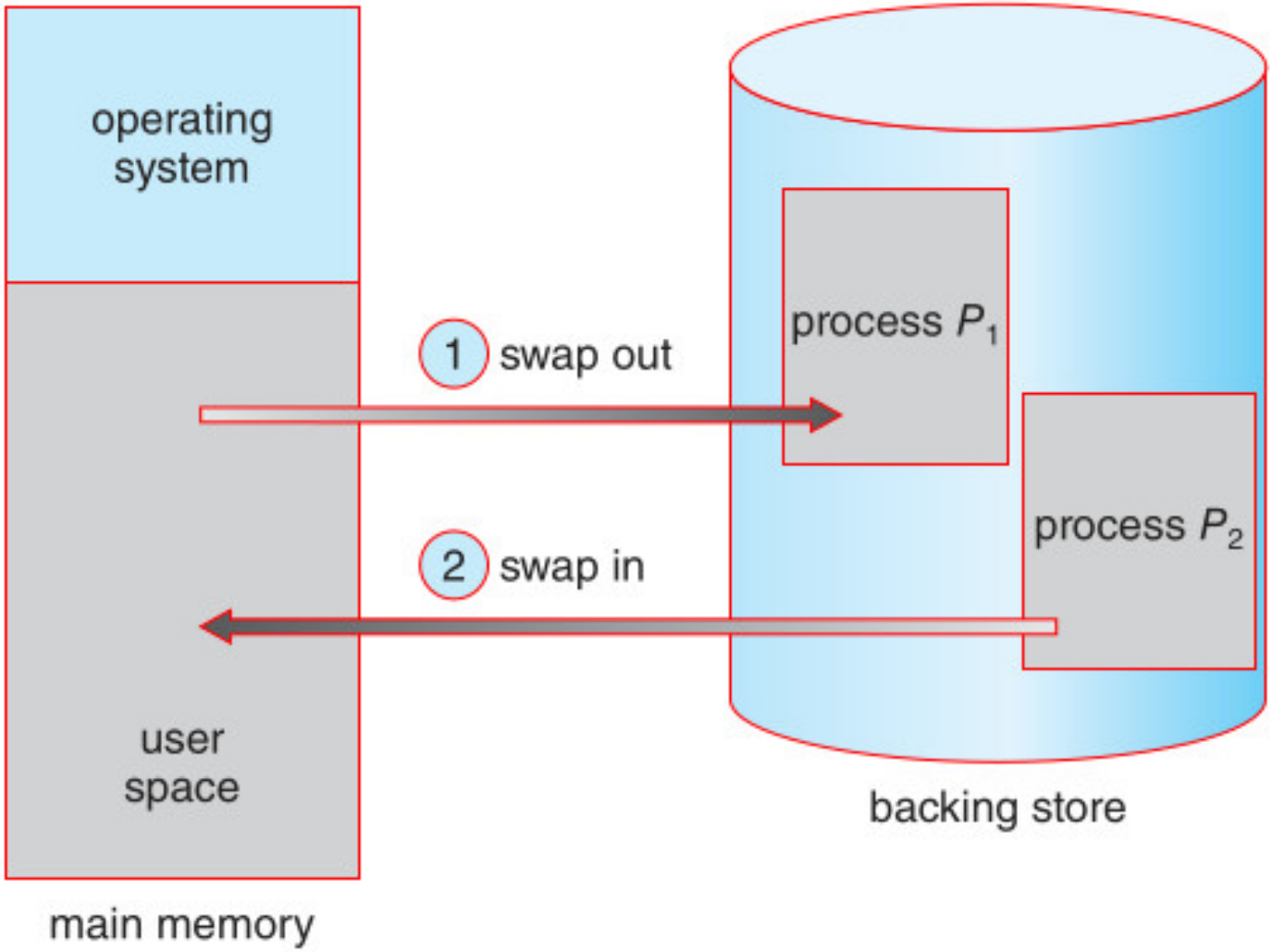
Inverted Page Tables



Standard swapping

- **Purpose**: Allows a process (or part) to be temporarily moved from memory to a backing store and back for execution.
- **Benefit**: Enables total physical address space of processes to exceed physical memory, increasing multiprogramming.
- **Standard Swapping**:
  - Moves entire processes between main memory and a **backing store**.
  - Backing store must be large, provide direct access, and store process data structures (including per-thread data for multithreaded processes).
- **OS Role**: Maintains metadata for swapped-out processes to restore them when swapped back in.
- **Advantages**:
  - Allows memory oversubscription to accommodate more processes than physical memory can hold.
  - Frees memory from idle processes for active ones.
- **Process**: Idle processes are swapped out; if they become active, they are swapped back in.

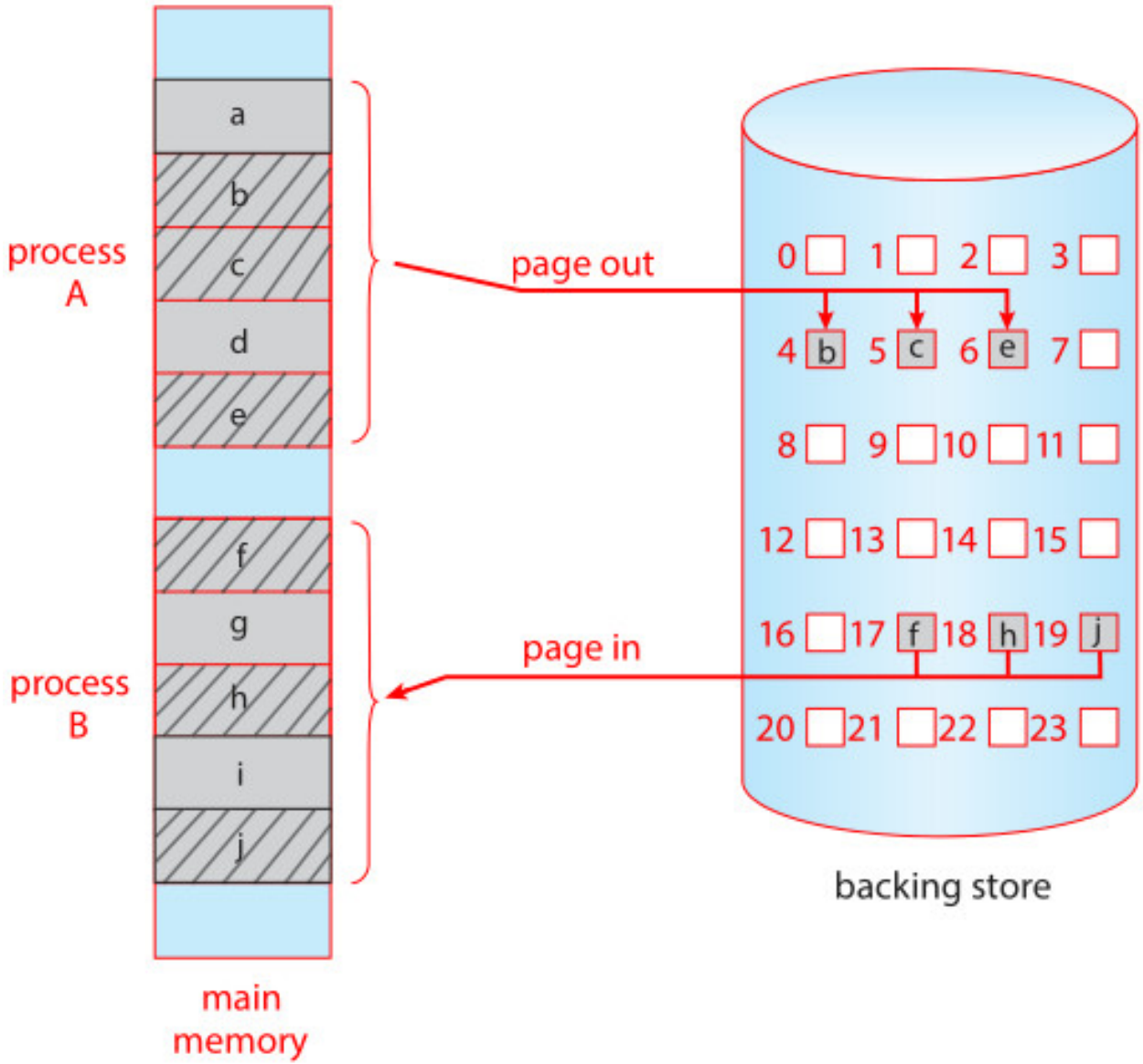
Standard swapping



Swapping with Paging

- Modern systems (Linux, Windows) swap individual pages, not entire processes.
- More efficient: only a small number of pages are swapped, still allows memory oversubscription.
- **Terminology**:
  - "Swapping" refers to standard swapping.
  - "Paging" refers to swapping with paging.
- **Operations**:
  - *Page out*: Moves a page to the backing store.
  - *Page in*: Retrieves a page from the backing store.

Standard swapping



Swapping on Mobile Systems

- **Mobile Systems**: Avoid swapping due to flash memory constraints:
  - Limited storage space compared to hard disks.
  - Poor throughput between main memory and flash memory.
- **iOS Strategy**:
  - Requests apps to voluntarily free memory when low.
  - Read-only data (e.g., code) can be removed and reloaded from flash memory.
  - Modified data (e.g., stack) are never removed.
  - Apps failing to free enough memory may be terminated.
- **Android Strategy**:
  - Similar to iOS: may terminate processes if memory is low.
  - Saves application state to flash memory before termination for quick restarts.
- **Developer Responsibility**: Mobile developers must carefully manage memory to avoid excessive usage or leaks.

Questions