Lesson 13

Virtual Memory

Objectives

- Define Virtual Memory and its benefits
- Illustrate how pages are loaded into memory using demand paging.

Virtual memory

- It is a technique that uses main memory as a "cache" for secondary storage
- It allows efficient and safe sharing of memory among multiple programs, such as for the memory needed by multiple virtual machines for Cloud computing, and to remove the programming burdens of a small, limited amount of main memory
- Code needs to be in memory to execute, but entire program rarely used
    - Error code, unusual routines, large data structures
- Entire program code not needed at same time

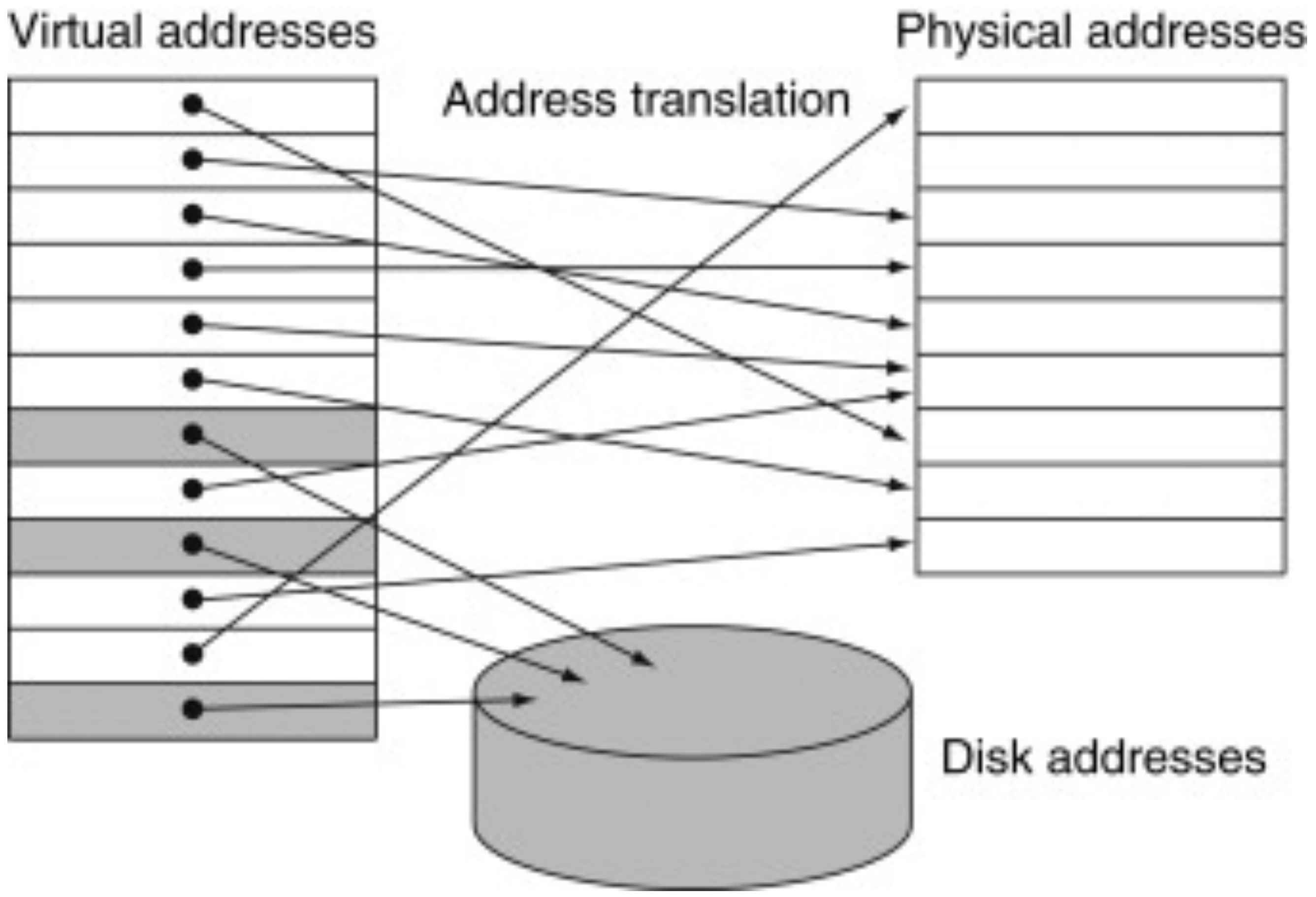
Virtual Memory

- Address space—a separate range of memory locations accessible only to this program
- Virtual memory implements the translation of a program's address space to *physical addresses.* This translation process enforces *protection* of a program's address space from other virtual machines.
- Memory Management Unit(MMU) maps logical addresses to physical addresses
- Logical address is an address generated by the CPU
- Logical memory is also called a virtual address
- *Physical address*: An address in main memory.
- ***Virtual address***: An address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.

Virtual Memory

- *Protection*: A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other's data. These mechanisms also isolate the operating system from a user process.
- ***Address translation***: Also called ***address mapping***. The process by which a virtual address is mapped to an address used to access memory.
- It allows a single user program to exceed the size of primary memory

Virtual Memory That is Larger Than Physical Memory



Pages

- The processor generates virtual addresses while the memory is accessed using physical addresses.
- Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page. A virtual memory block is called a page
- A virtual page can be absent from main memory and not mapped to a physical address; in that case, the page resides on disk
- A virtual memory miss is called a page fault.
- **Page fault**: An event that occurs when an accessed page is not present in main memory.

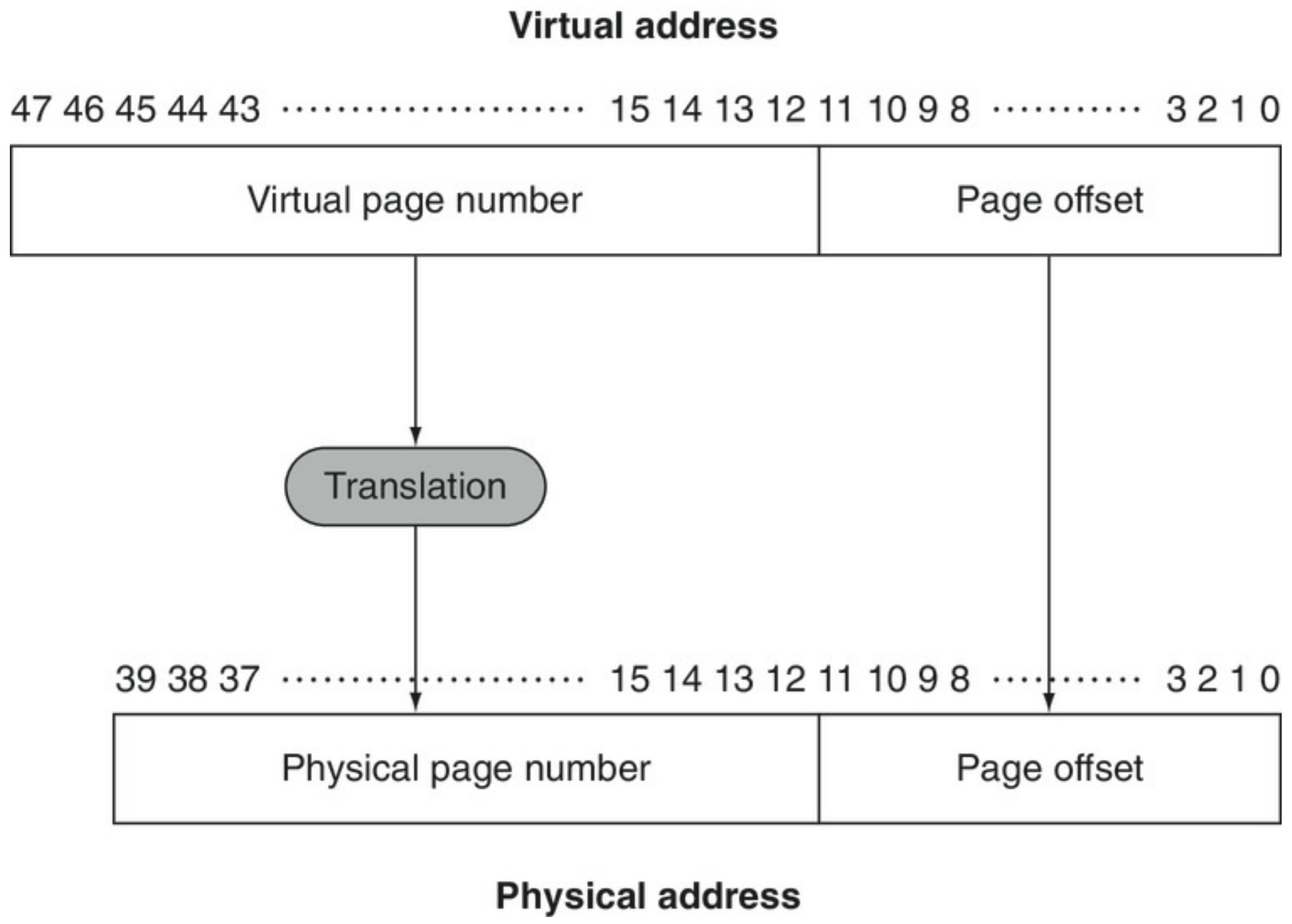
Relocation

- Virtual memory loads programs for execution by providing relocation.
- Relocation maps the virtual addresses used by a program to different physical addresses before the addresses are used to access memory.
- Program can load anywhere in the main memory

Virtual Address

- In virtual memory, the address is broken into a *virtual page number* and a *page offset*.

Mapping from a virtual to a physical address



Segmentation

- Paging uses fixed-size blocks
- Segmentation uses a variable-size block
- In segmentation, an address consists of two parts: a segment number and a segment offset. The segment number is mapped to a physical address, and the offset is added to find the actual physical address.

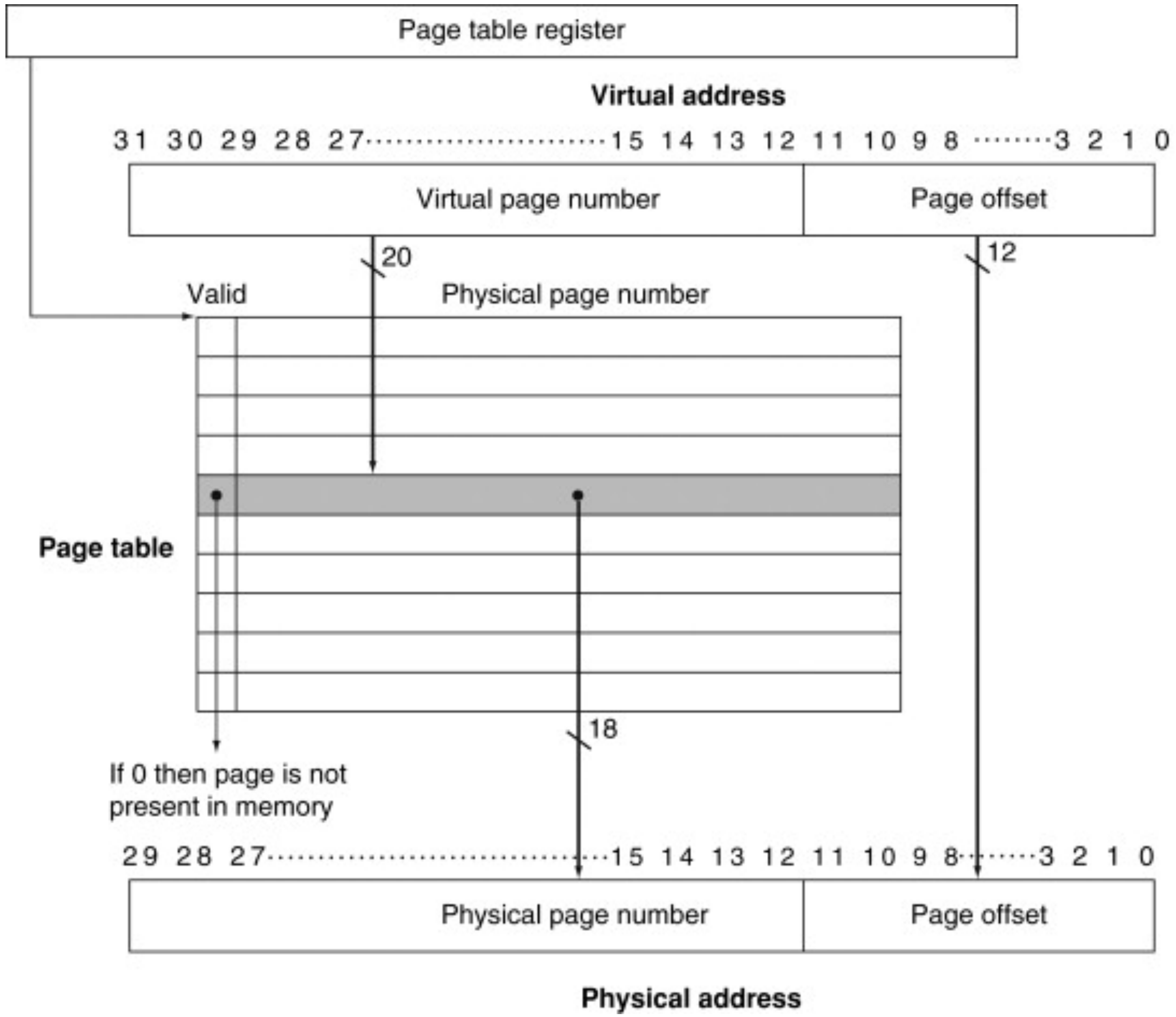
Virtual Memory Implementation

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Placing a page and finding it again

- We locate pages by using a table that indexes the memory; this structure is called a *page table*, and it resides in memory
- ***Page table***: The table containing the virtual to physical address translations in a virtual memory system.
- A page table is indexed with the page number from the virtual address to discover the corresponding physical page number.

Page table



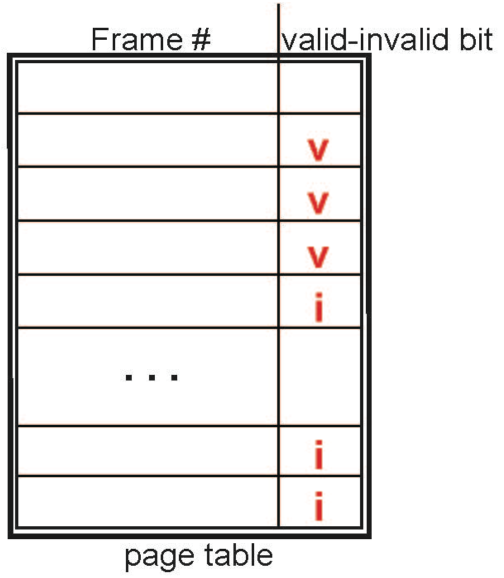
Page table register
Virtual address
31 30 29 28 27 ··· 15 14 13 12 11 10 9 8 ··· 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not present in memory
29 28 27 ··· 15 14 13 12 11 10 9 8 ··· 3 2 1 0
Physical page number
Page offset
Physical address

The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address

- A 32-bit address

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory – **memory resident**, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



- During MMU address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault

Page faults

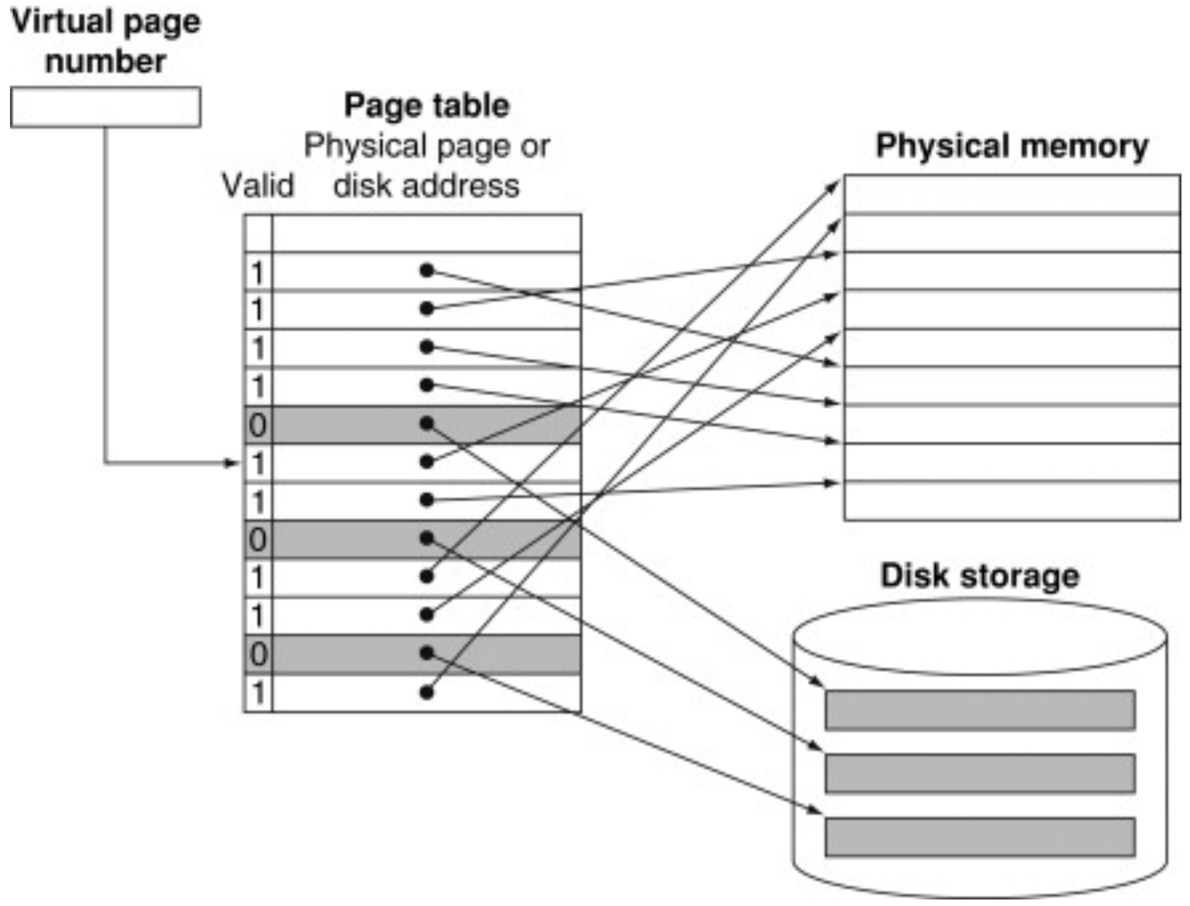
- If the valid bit for a virtual page is off, a page fault occurs

1. Operating system looks at another table to decide:
    - Invalid reference $\Rightarrow$ abort
    - Just not in memory
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault

Swap Space

- **Swap space**: The space on the disk reserved for the full virtual memory space of a process.

The page table maps each page in virtual memory to either a page in main memory or a page stored on disk

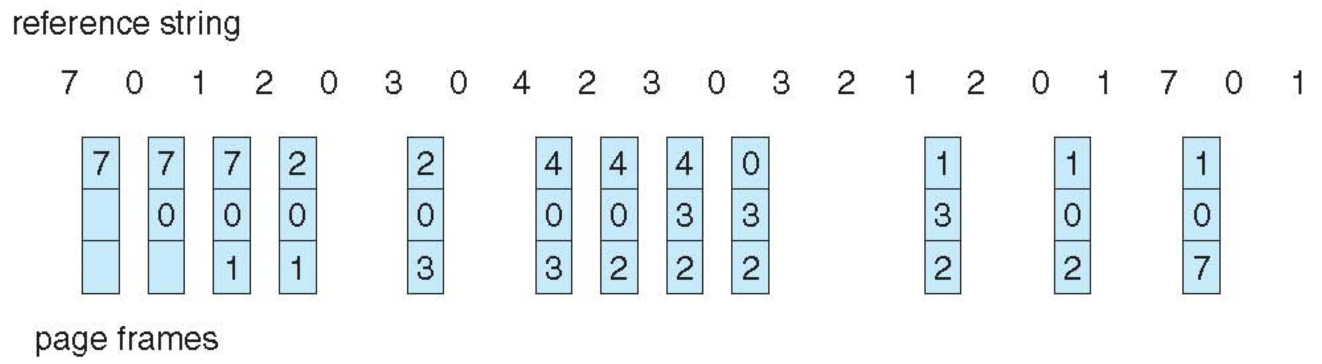


Page Replacement

- When a page fault occurs, if all the pages in main memory are in use, the operating system must choose a page to replace.
- Operating systems follow the *least recently used* (LRU) replacement scheme

LRU

- The operating system searches for the least recently used page, assuming that a page that has not been used in a long time is less likely to be needed than a more recently accessed page.
- The replaced pages are written to swap space on the disk.



- 12 pages faults occur above

LRU and **Reference bit**

- **Reference bit**: Also called **use bit**. A field that is set whenever a page is accessed and that is used to implement LRU or other replacement schemes.
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however

Computing total page table size

- With a 32-bit virtual address, 4 KiB pages, and 4 bytes per page table entry, we can compute the total page table size:

$$\text{Number of page table entries} = \frac{2^{32}}{2^{12}} = 2^{20}$$

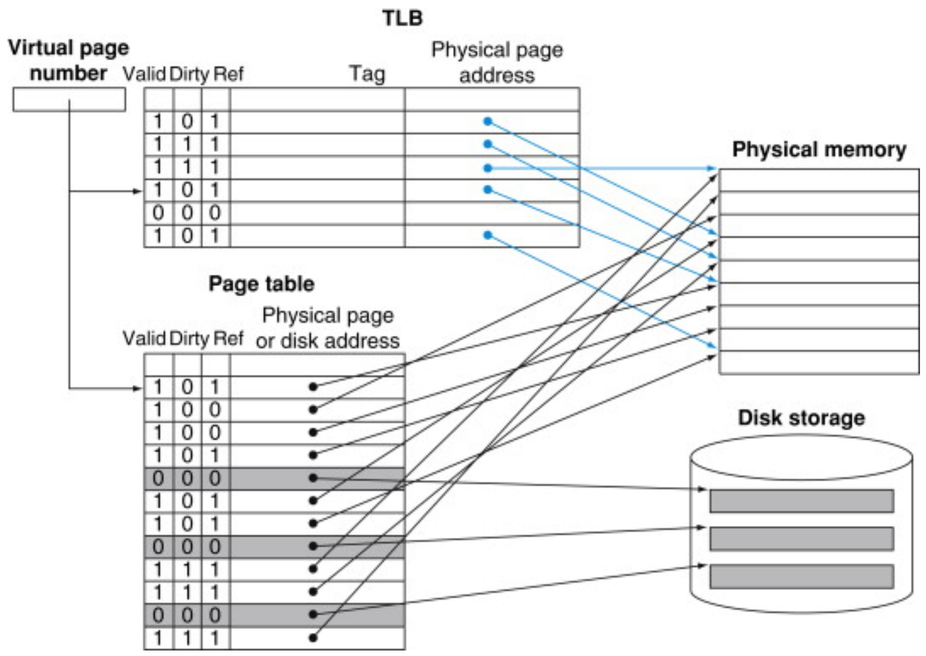
$$\text{Size of page table} = 2^{20}\ \text{page table entries} \times 2^{2}\,\frac{\text{bytes}}{\text{page table entry}} = 4\ \text{MiB}$$

The TLB

- Since the page tables are stored in main memory, every memory access by a program can take at least twice as long. one memory access to obtain the physical address and a second access to get the data
- Modern processors include a special cache that keeps track of recently used translations. This special address translation cache is traditionally referred to as a *translation-lookaside buffer* (TLB)

Translation-lookaside buffer (TLB)

- A cache that keeps track of recently used address mappings to try to avoid an access to the page table.



Reading

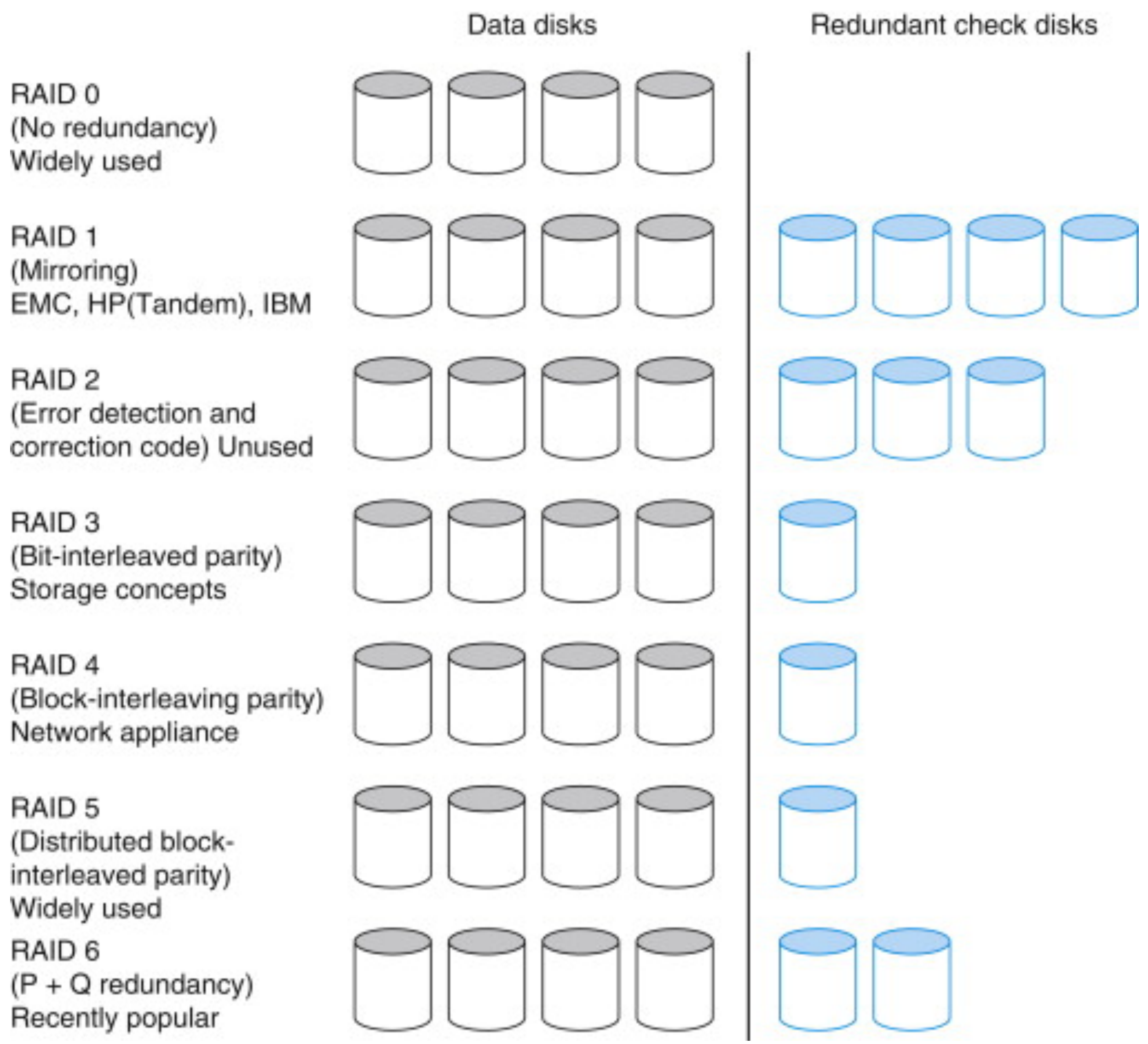
- Virtual Memory Reading 5.7

End of Virtual Memory

Redundant Arrays of Inexpensive Disks (RAID)

- An organization of disks that uses an array of small and inexpensive disks so as to increase both performance and reliability.

RAID


Data disks
Redundant check disks
RAID 0
(No redundancy)
Widely used
RAID 1
(Mirroring)
EMC, HP(Tandem), IBM
RAID 2
(Error detection and
correction code) Unused
RAID 3
(Bit-interleaved parity)
Storage concepts
RAID 4
(Block-interleaving parity)
Network appliance
RAID 5
(Distributed block-
interleaved parity)
Widely used
RAID 6
(P + Q redundancy)
Recently popular

No redundancy (RAID 0)

- spreading data over multiple disks, called *striping*
- ***Striping***: Allocation of logically sequential blocks to separate disks to allow higher performance than a single disk can deliver.

Mirroring (RAID 1)

- *RAID 1 is also called mirroring* or *shadowing*, uses twice as many disks as does RAID 0
- Whenever data is written to one disk, that data is also written to a redundant disk, so that there are always two copies of the information.
- If a disk fails, the system just goes to the "mirror" and reads its contents to get the desired information.
- Mirroring is the most expensive
- ***Mirroring***: Writing identical data to multiple disks to increase data availability.

Error detecting and correcting code (RAID 2)

- RAID 2 borrows an error detection and correction scheme most often used for memories

Error detection code

- A code that enables the detection of an error in data, but not the precise location and, hence, correction of the error

Hamming code

- Hamming used a *parity code* for error detection
- In a parity code, the number of 1s in a word is counted; the word has odd parity if the number of 1s is odd and even otherwise. When a word is written into memory, the parity bit is also written (1 for odd, 0 for even). That is, the parity of the N+1 bit word should always be even. Then, when the word is read out, the parity bit is read and checked. If the parity of the memory word and the stored parity bit do not match, an error has occurred.

Parity code for error detection

- Calculate the parity of a byte with the value 31_{ten} and show the pattern stored to memory. Assume the parity bit is on the right. Suppose the most significant bit was inverted in memory, and then you read it back. Did you detect the error? What happens if the two most significant bits are inverted?

Bit-interleaved parity (RAID 3)

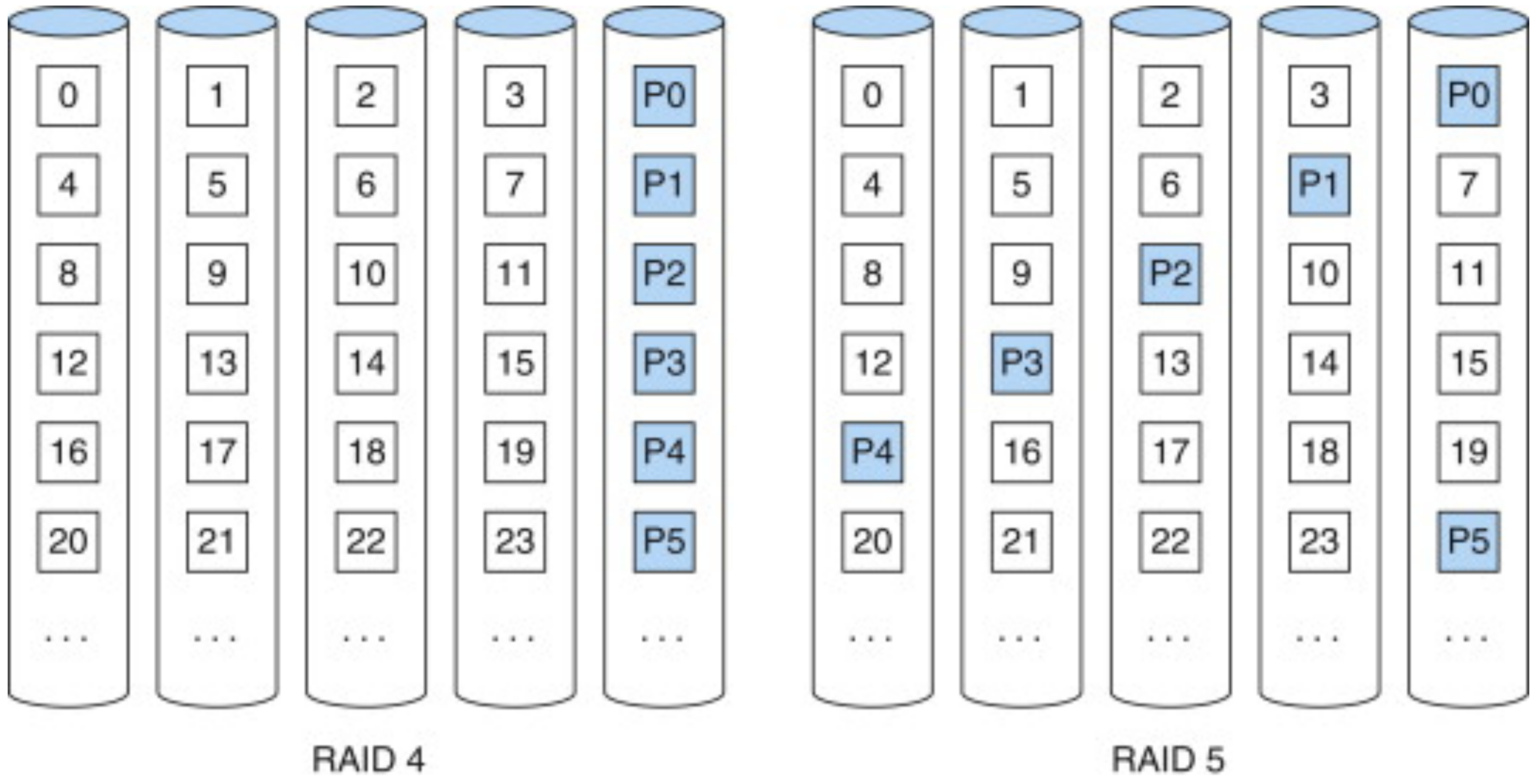
Block-interleaved parity (RAID 4)

- RAID 4 uses the same ratio of data disks and check disks as RAID 3, but they access data differently. The parity is stored as blocks and associated with a set of data blocks.

Distributed block-interleaved parity (RAID 5)

- The parity information is spread throughout all the disks so that there is no single bottleneck for writes.

RAID 4 VS RAID 5



P + Q redundancy (RAID 6)

•

Reading

- 5.10