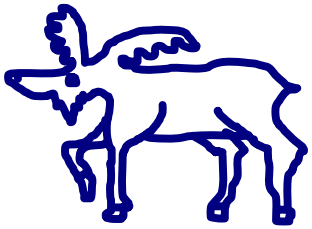
Lecture 24
Virtual Memory

Byung-gi Kim

School of Computer Science and Engineering

Soongsil University



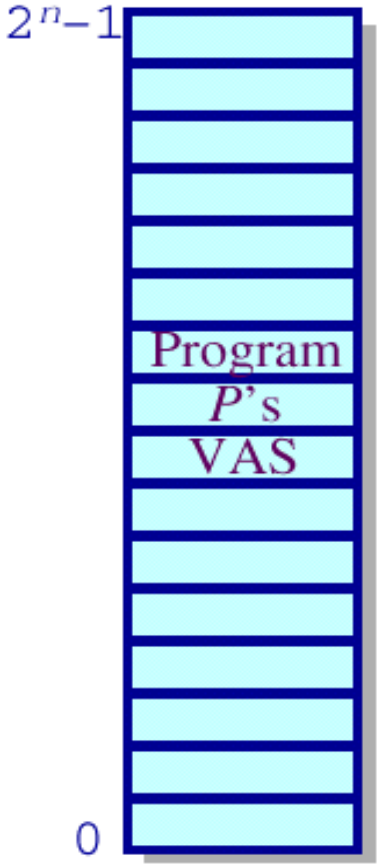
5. Large and Fast: Exploiting Memory Hierarchy

5.4 Virtual Memory

- **Virtual memory**
  - Use main memory as a "cache" for the secondary storage
  - Managed jointly by CPU hardware and the operating system
- **Motivations**
  - Illusion of having more physical main memory
  - Protection from illegal memory access
  - Controlled code and data sharing among processes
- **Goal of virtual memory**
  - Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

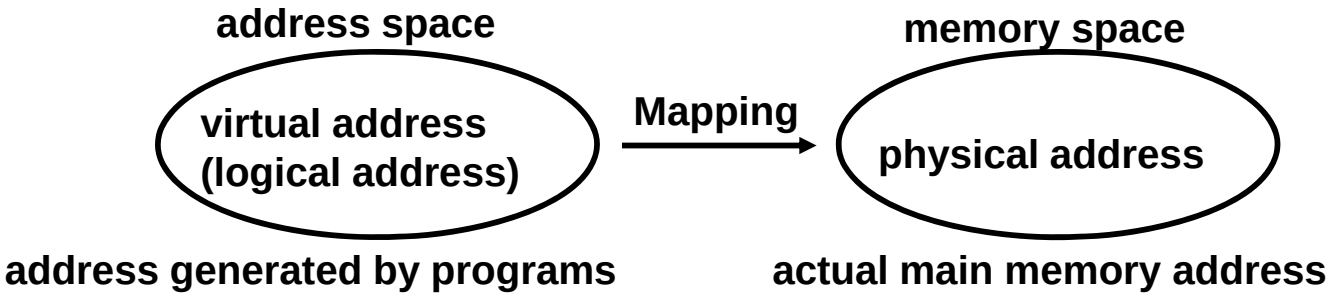
Virtual Memory Concept

- **Hide all physical aspects of memory from users.**
  - Memory is a logically unbounded *virtual (logical) address space* of 2^n bytes.
  - Only portions of virtual address space are in physical memory at any one time.
- **Relocation**
  - Advantage of virtual memory
  - Simplifies program loading for execution
  - Noncontiguous allocation
  - Only to find a sufficient number of pages in main memory



Address Mapping

- **Address space (logical) and memory space (physical)**



- **Virtual addresses**
  - Used by the processor (program, user)
- **Physical addresses**
  - Used to access main memory
- **Page**
  - Virtual memory block
- **Page fault**
  - Virtual memory miss

Memory Mapping (or Address Translation)

- Change a virtual page number to a physical page number
- Performed by a combination of hardware and software

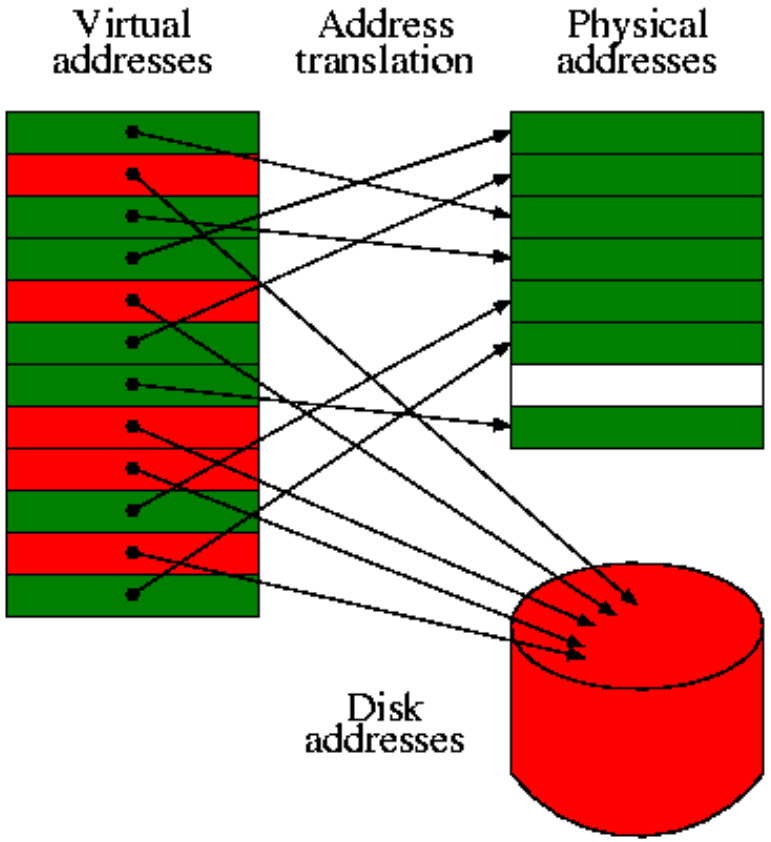


Figure 5.19

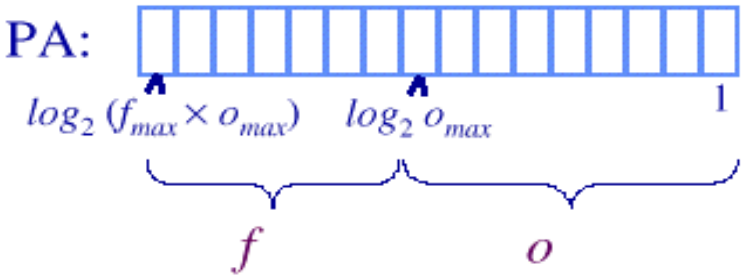
Paging

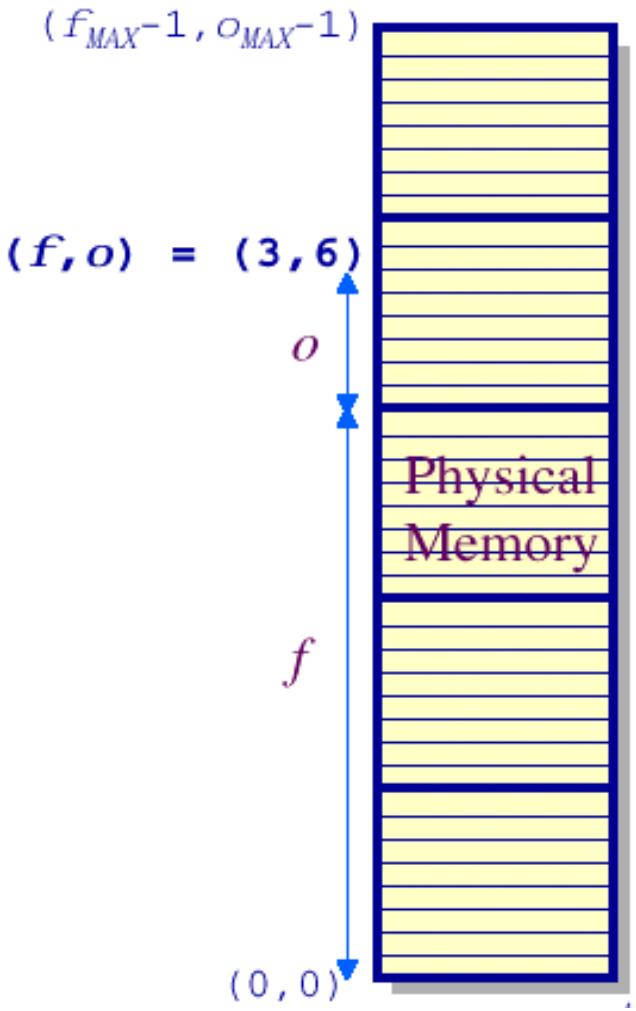
- Physical memory is divided into equal sized *page frames.*
  - size of page = size of frame
- Physical memory address is a pair *(f, o).*

f — frame number (f_{max} frames)

o — frame offset (o_{max} bytes/frames)

Physical address = $o_{max} \times f + o$





Address in Virtual Memory System

- **Address = virtual page number || page offset**

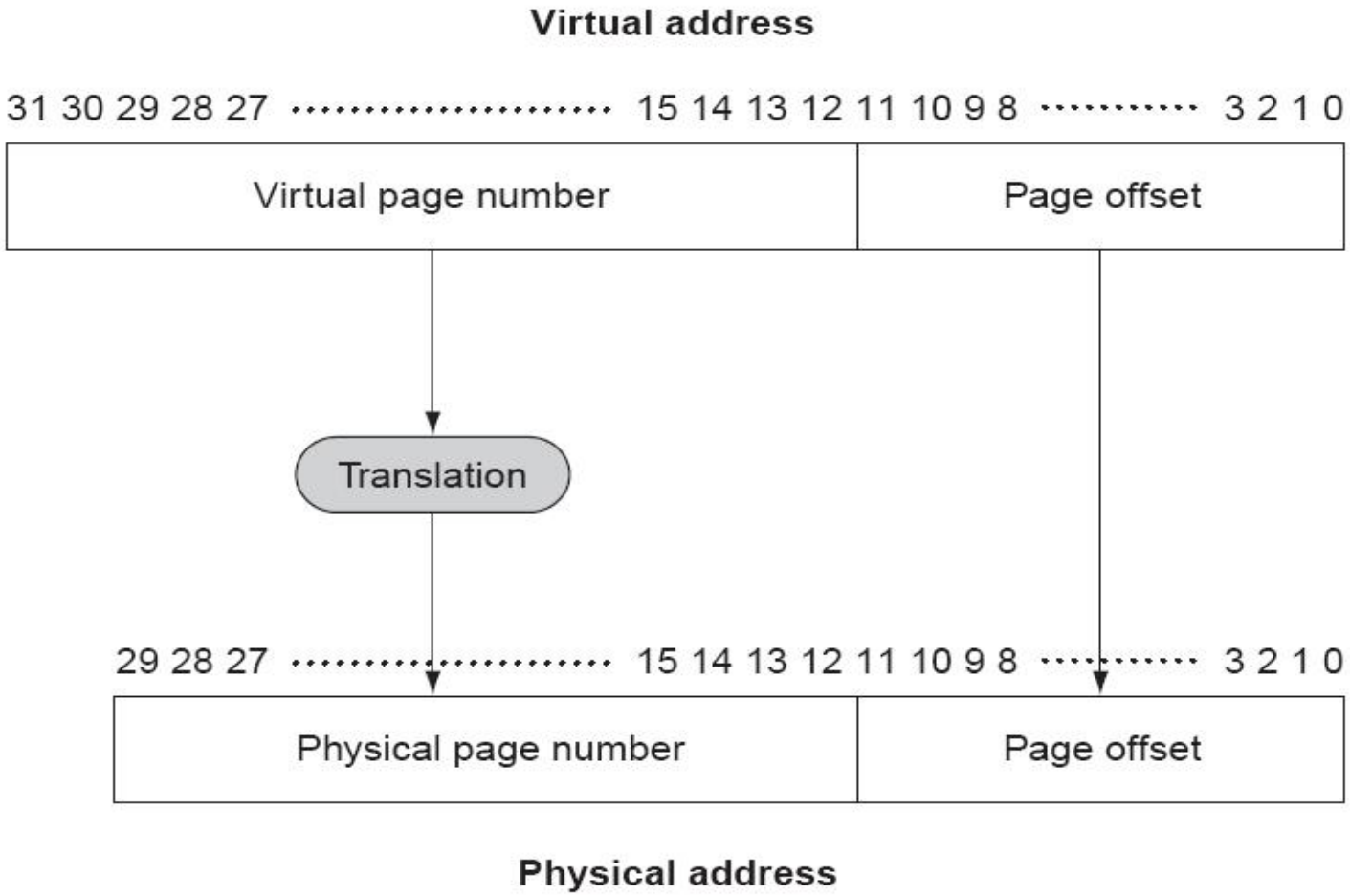


Figure 5.20

Design Considerations

- **Very large miss penalty (millions of cycles)**

1. Large page size

- Should be large enough to amortize the high disk access time
- Typically 4KB～16KB, recently 32KB or 64KB
- 1KB pages in the new embedded systems

2. Reducing page fault rate

- Fully associative placement of pages

3. Using clever software algorithms

- Because the overhead is small compared to disk access time

4. Write-back

- Too long a disk write time

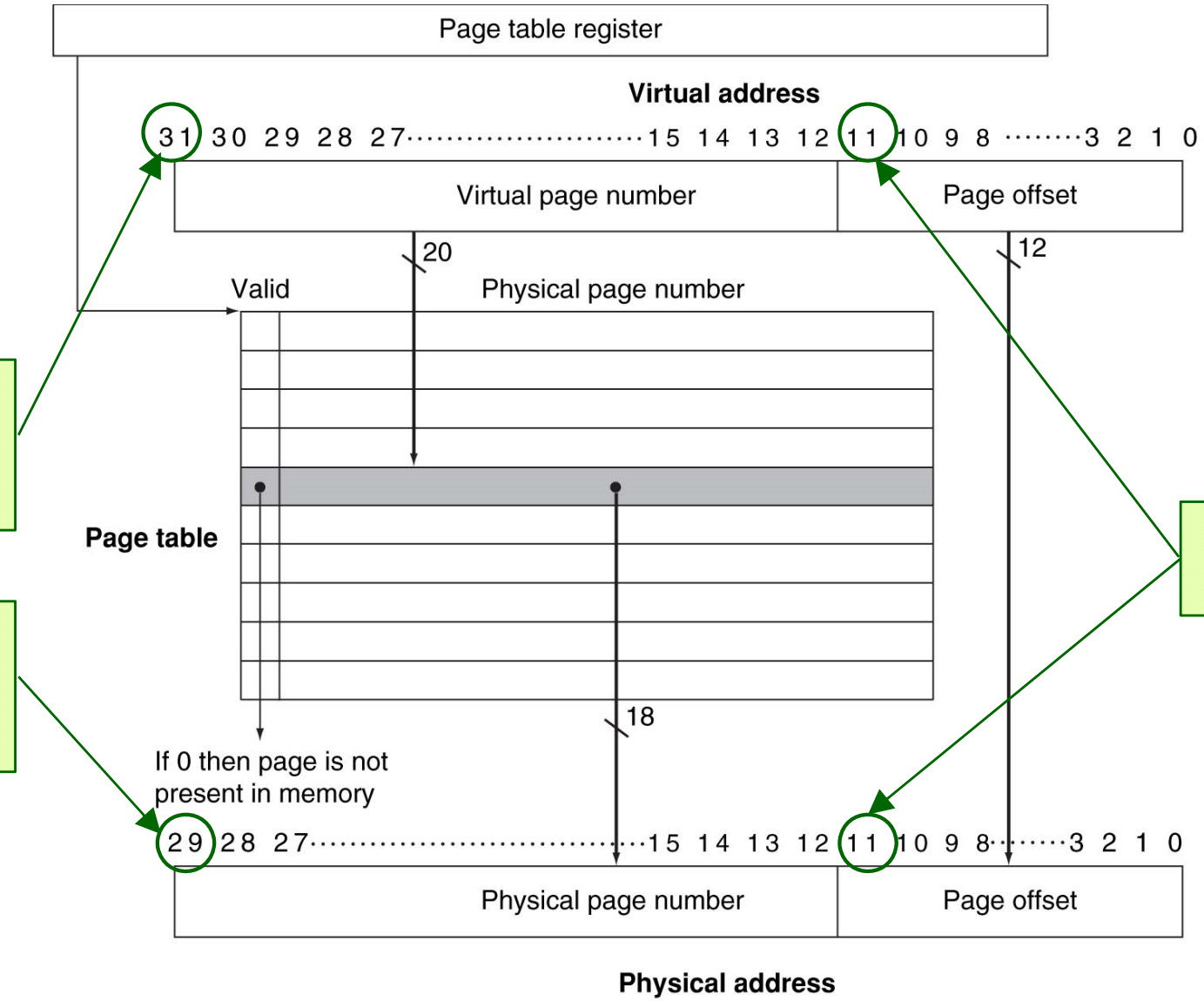
Elaboration

- **Paging ... fixed-size blocks**
- **Segmentation ... variable-size blocks**
  - Address = (segment number, offset)
  - Addition rather than concatenation
  - Supporting more powerful methods of protection and sharing in an address space
  - Splitting address space into logically separate pieces

Placing a Page and Finding It Again

- **Fully associative mapping**
  - Reducing the page fault rate by optimizing the page placement
  - A virtual page can be mapped to any physical page.
- **Page table**
  - A full table that indexes the memory
  - Contains the virtual to physical address translation
  - Resides in memory
  - Each process has its own page table.
  - Indexed with the virtual page number
  - Page table entry (PTE)
    - Physical page number
    - Valid bit (or residence bit or presence bit)
  - Page table register
    - Starting address of the page table

Address Translation Hardware



Virtual address space = 2^{32} bytes

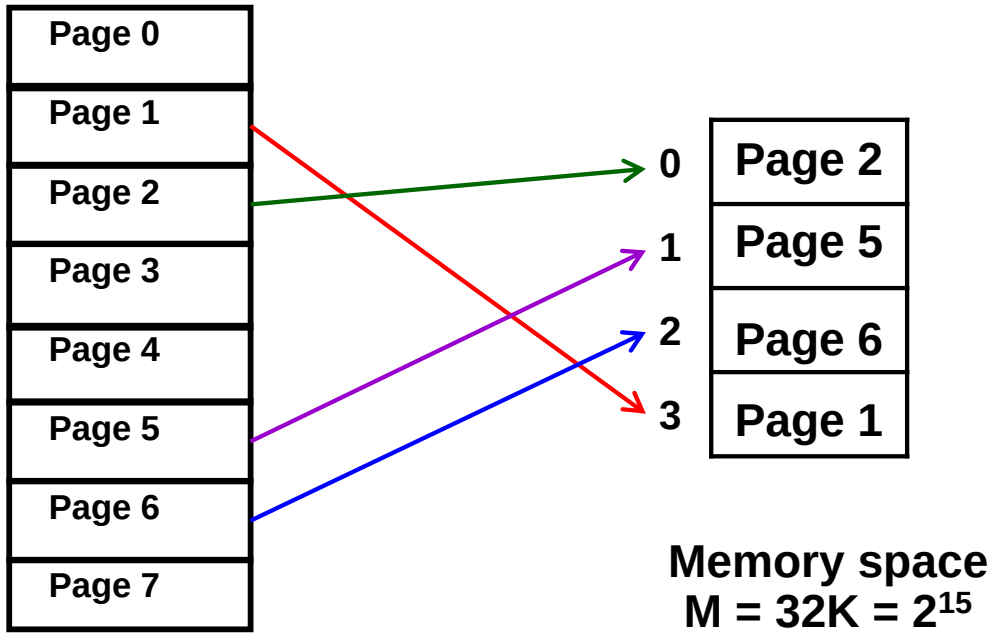
Physical address space = 2^{30} bytes

Page size = 2^{12} bytes

Figure 5.21

Example: Prob. 5 of 2011-1 Terminal Exam

- Virtual address space = 64KB = 2^{16} B
- Physical memory space = 32KB = 2^{15} B
- Page size = 8KB = 2^{13} B



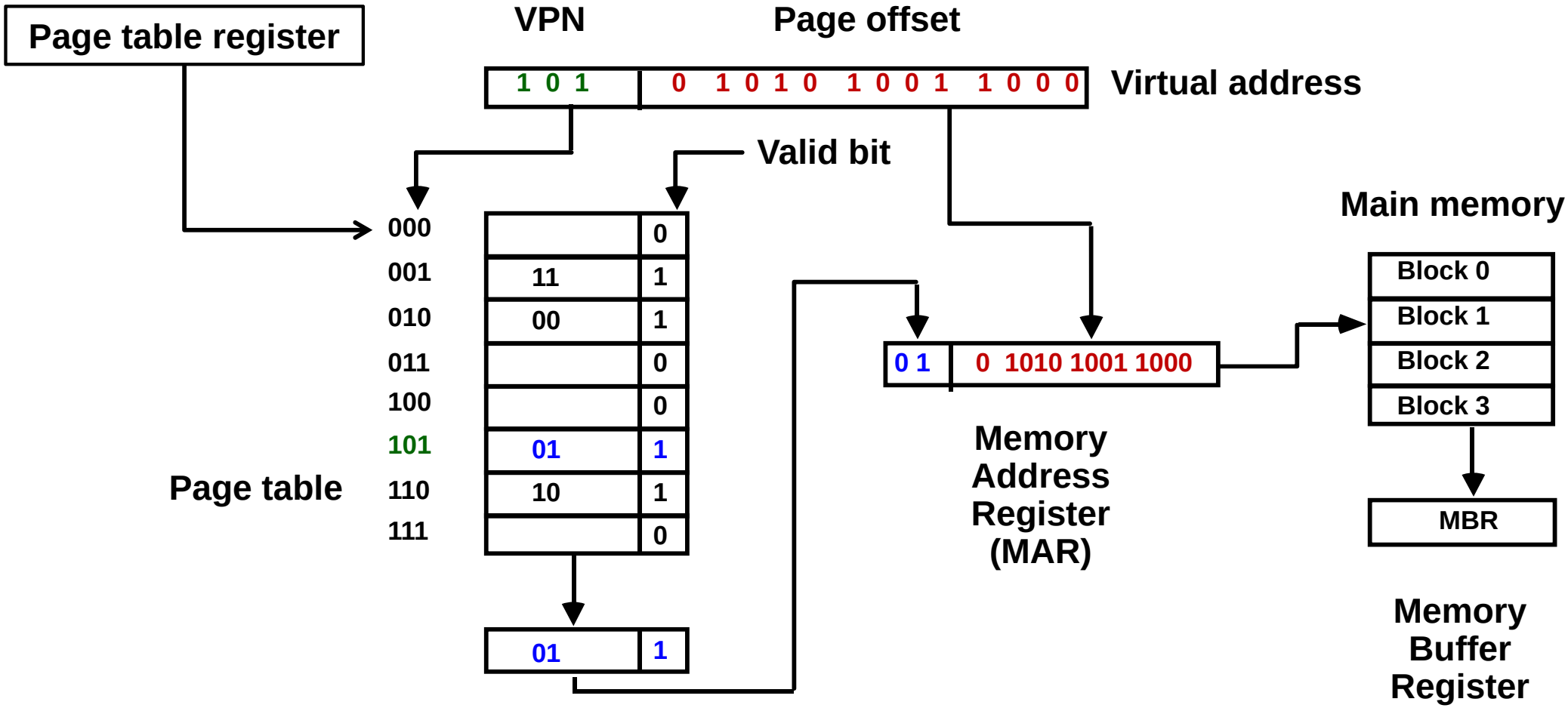
Address space
N = 64K = 2^{16}

Memory space
M = 32K = 2^{15}

	V	PPN
000	0	10
001	1	11
010	1	00
011	0	01
100	0	11
101	1	01
110	1	10
111	0	01

Page table

Address Translation Example



Reference String (in byte address)

- **4010, 5100, 0048, 6F08, 33B4, 1084, AAB0, 2770**

VA			PA	
Hex	Binary	VPN	PPN	Hex
4010	0100 0000 0001 0000	010	**00**	**0010**
5100	0101 0001 0000 0000	010	**00**	**1100**
0048	0000 0000 0100 1000	000	**page fault**	
6F08	0110 1111 0000 1000	011	**page fault**	
33B4	0011 0011 1011 0100	001	**11**	**73B4**
1084	0001 0000 1000 0100	000	**10**	**5084**
AAB0	1010 1010 1011 0000	101	**page fault**	
2770	0010 0111 0111 0000	001	**11**	**6770**

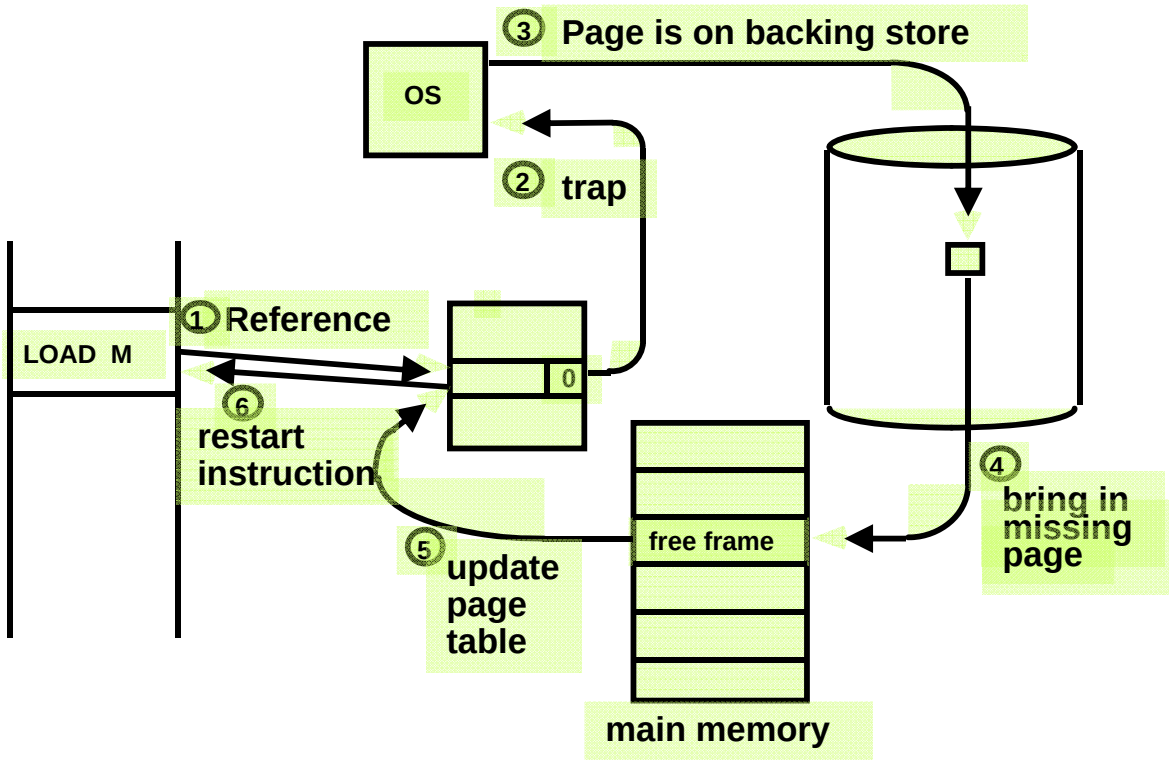
Page table							
000	001	010	011	100	101	110	111
	11	**00**			**01**	**10**	
		00					
		00					
10							
			01				
	11						
10							
					01		
	11						

Final Page Table

000	1	10
001	1	11
010	1	00
011	0	01
100	0	11
101	1	01
110	0	10
111	0	01

Page Faults

- **Page fault**
  - Occurs when the valid bit for a virtual page is off
  - OS must be given control through exception mechanism.
  - Find the page in the next level and decide where to place the requested page in memory



Processor architecture should provide the ability to restart any instruction after a page fault.

Handling a Page Fault

1. Trap to the OS by page fault exception
2. Save the user registers and program state
3. Determine that the exception was a page fault
4. Check the reference was legal and determine the location of the page on the disk
5. Choose a victim using page replacement algorithm
   - Writeback to disk if dirty block
6. Issue a read from disk to the free frame
7. While waiting, the CPU may be allocated to some other process
8. Interrupt from disk (I/O completed)
9. Save the registers and program state for the other user
10. Determine that the interrupt was from disk
11. Correct the page tables (the desired page is now in memory)
12. Wait for the CPU to be allocated to this process again
13. Restore the user registers, program state, and new page table, then resume the interrupted instruction.

Page Table with Disk Addresses

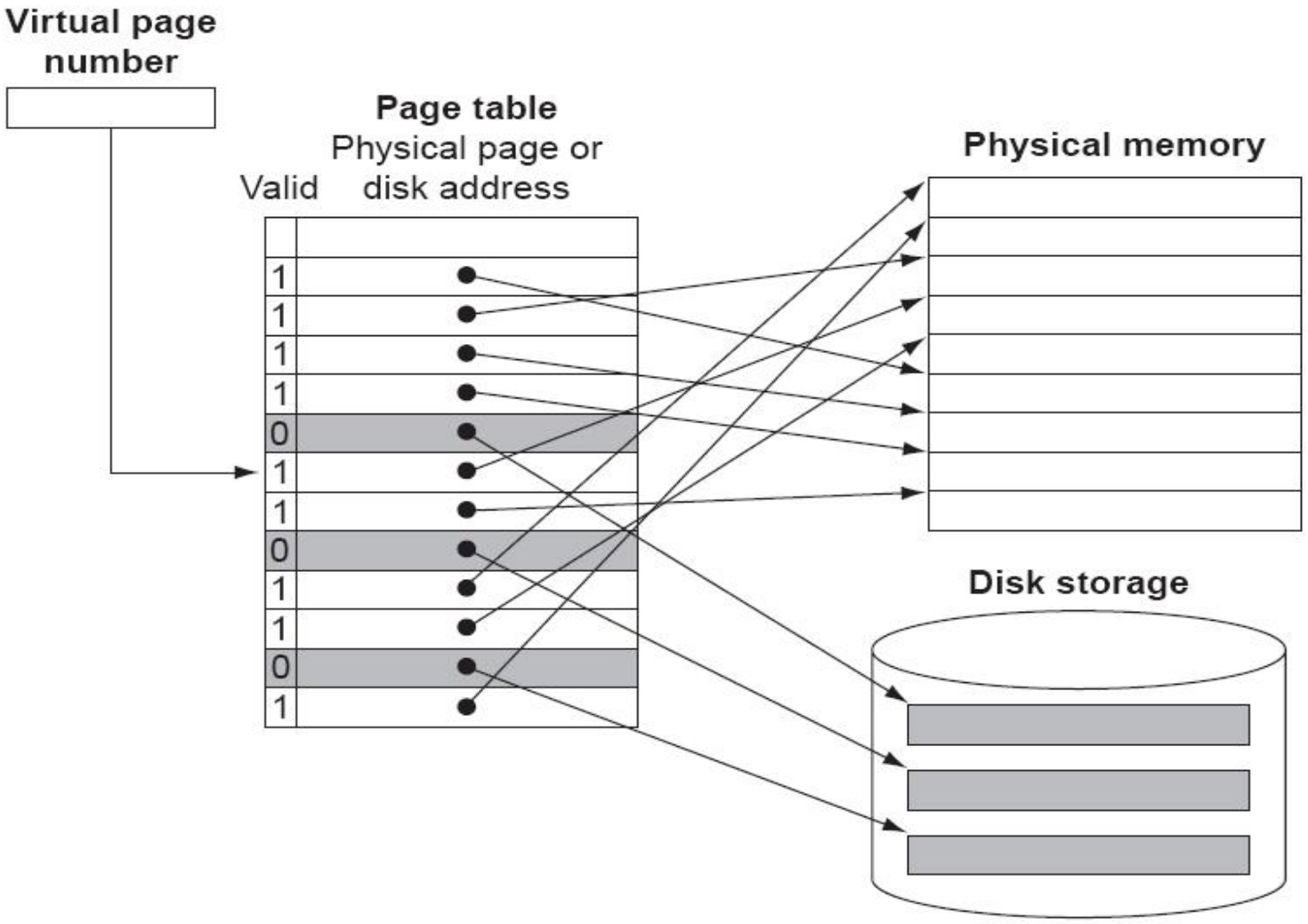


Figure 5.22

Elaboration

- **Page table size for 32-bit virtual address, 4 KB pages, 4 bytes per page table entry**

$$\frac{2^{32}}{2^{12}} \text{ pages} \times 4 \frac{\text{bytes}}{\text{page}} = 4 \text{ MB (for each program in execution)}$$

What About Writes?

- **Write-back**
  - Performing write into the page in memory
  - Copying the page back to disk when it is replaced
  - Dirty bit (=Modified bit)
    - added to page table entry