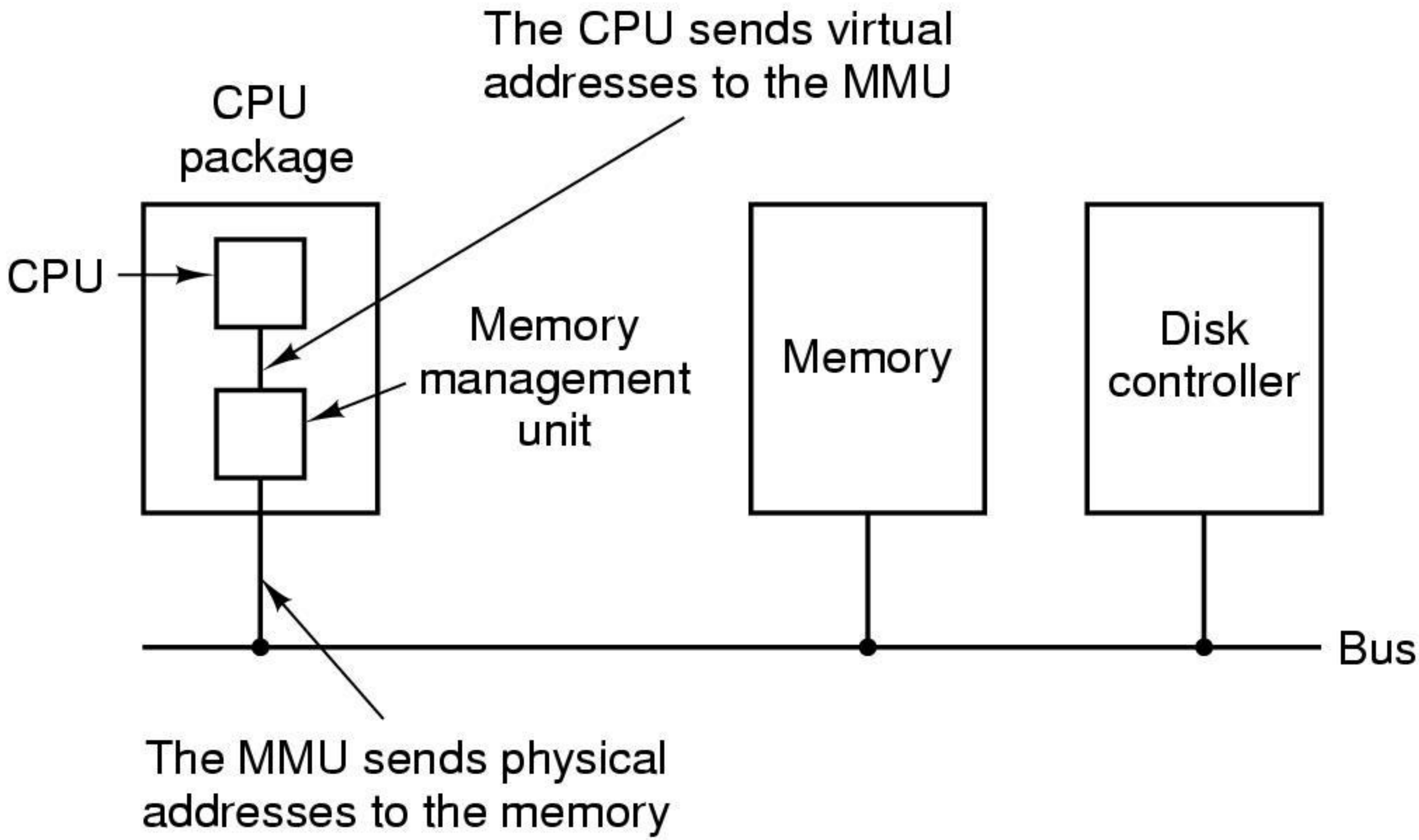
# Chapter 3

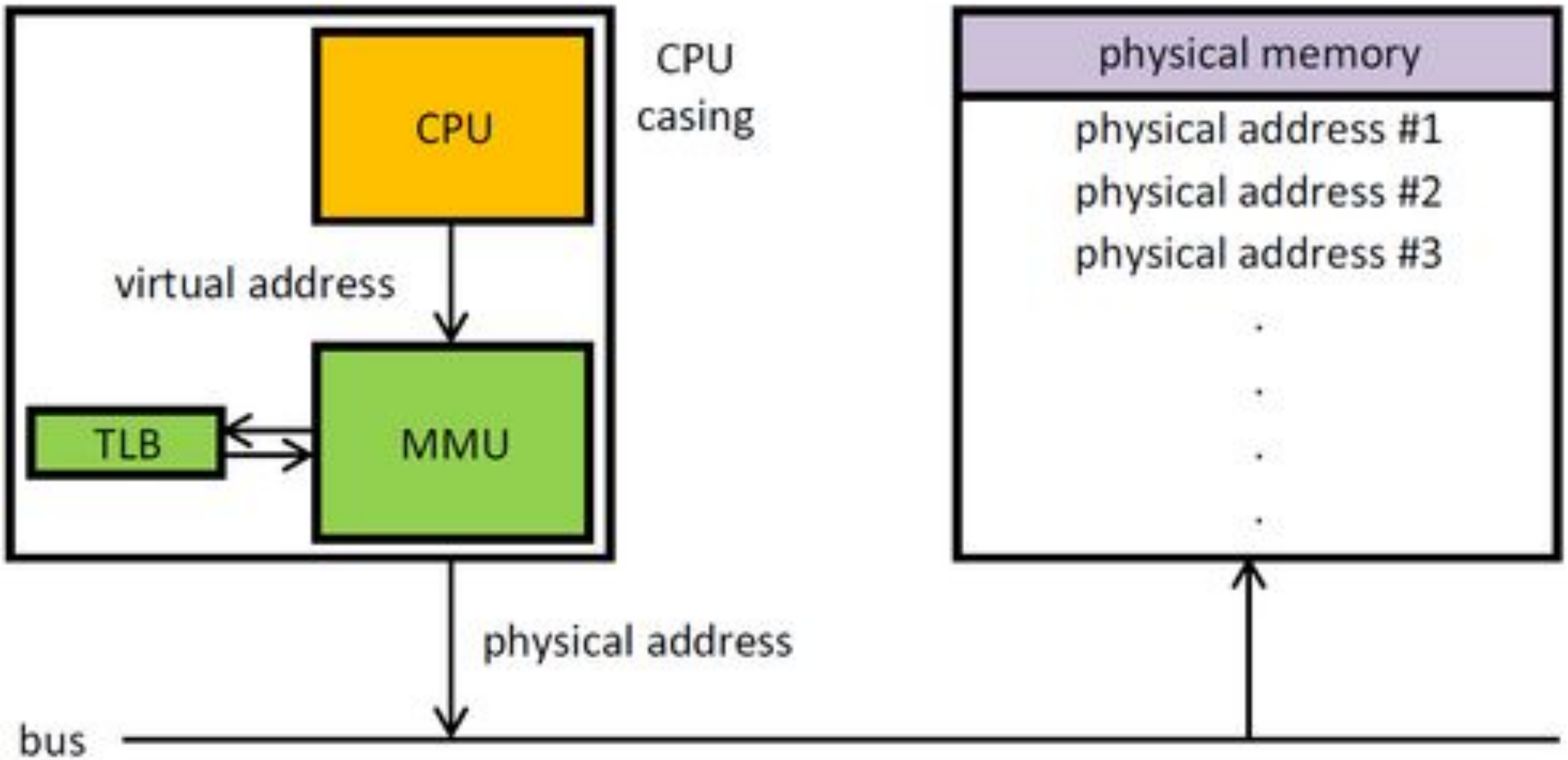
# Memory Management

# Virtual Memory

- a technique that allows the execution of processes that are not completely in memory.
- **abstracts** main memory into an *extremely large*, *uniform* array of storage as viewed by the programmers
- major advantages
  - programs can be larger than physical memory.
  - works just fine in a multiprogramming system, with pieces of many programs in memory at once.
    - While a program is waiting for piece of itself to be read in, the CPU can be given to another process.

# Paging : A Technique used in Virtual Memory System





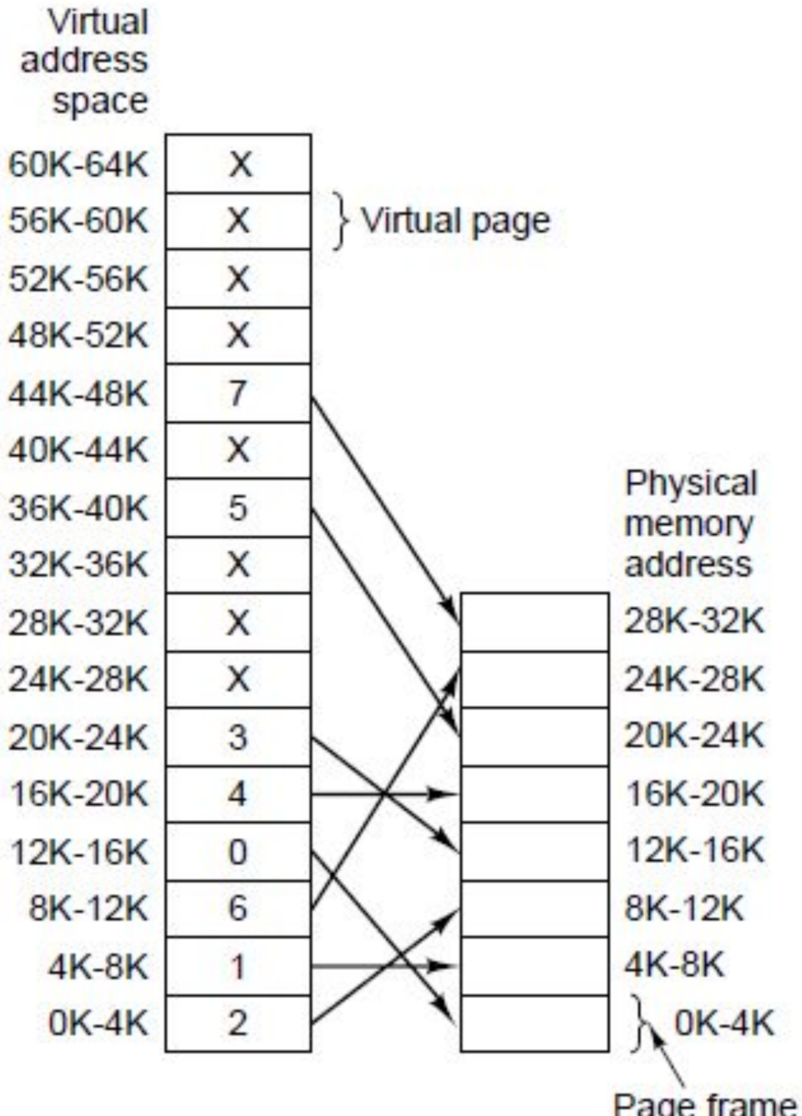

CPU: Central Processing Unit

MMU: Memory Management Unit

TLB: Translation lookaside buffer

# Paging : A Technique used in Virtual Memory System

- program-generated addresses are called virtual addresses and form the virtual address space
- The virtual address space is divided into fixed-size units called pages.
  - Each page is a contiguous range of addresses
  - These pages are mapped onto physical memory
- The corresponding units in the physical memory are called page frames.
- Transfers between RAM and disk are always in whole pages




## Address translation

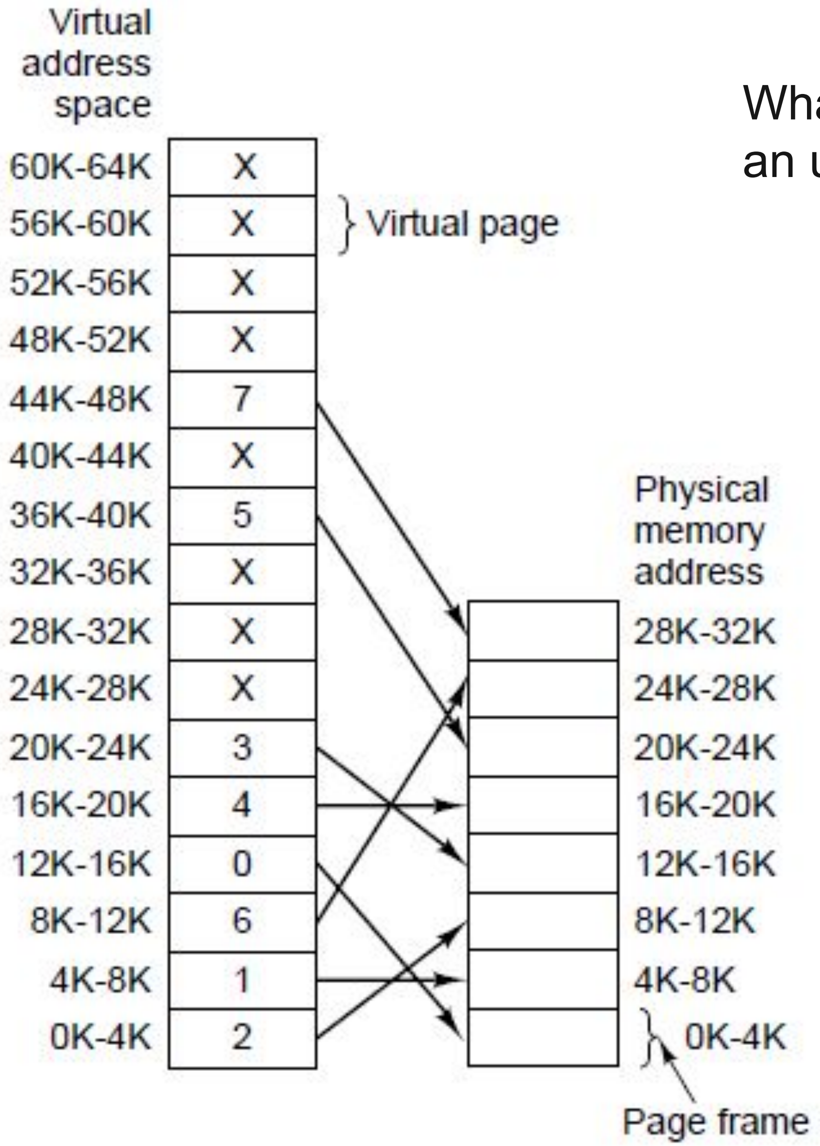
$$\textit{virtual address} \xrightarrow{\textit{page table}} \textit{physical address}$$

$$Page\, 0 \xrightarrow{map\, to} Frame\, 2$$

$$0_{virtual} \xrightarrow{map\, to} 8192_{physical}$$

$$\begin{matrix} 20500_{vir} \\ (20k+20)_{vir} \end{matrix} \xrightarrow{map\, to} \begin{matrix} 12308_{phy} \\ (12k+20)_{phy} \end{matrix}$$

# Page Fault




What happens if the program references an unmapped addresses???

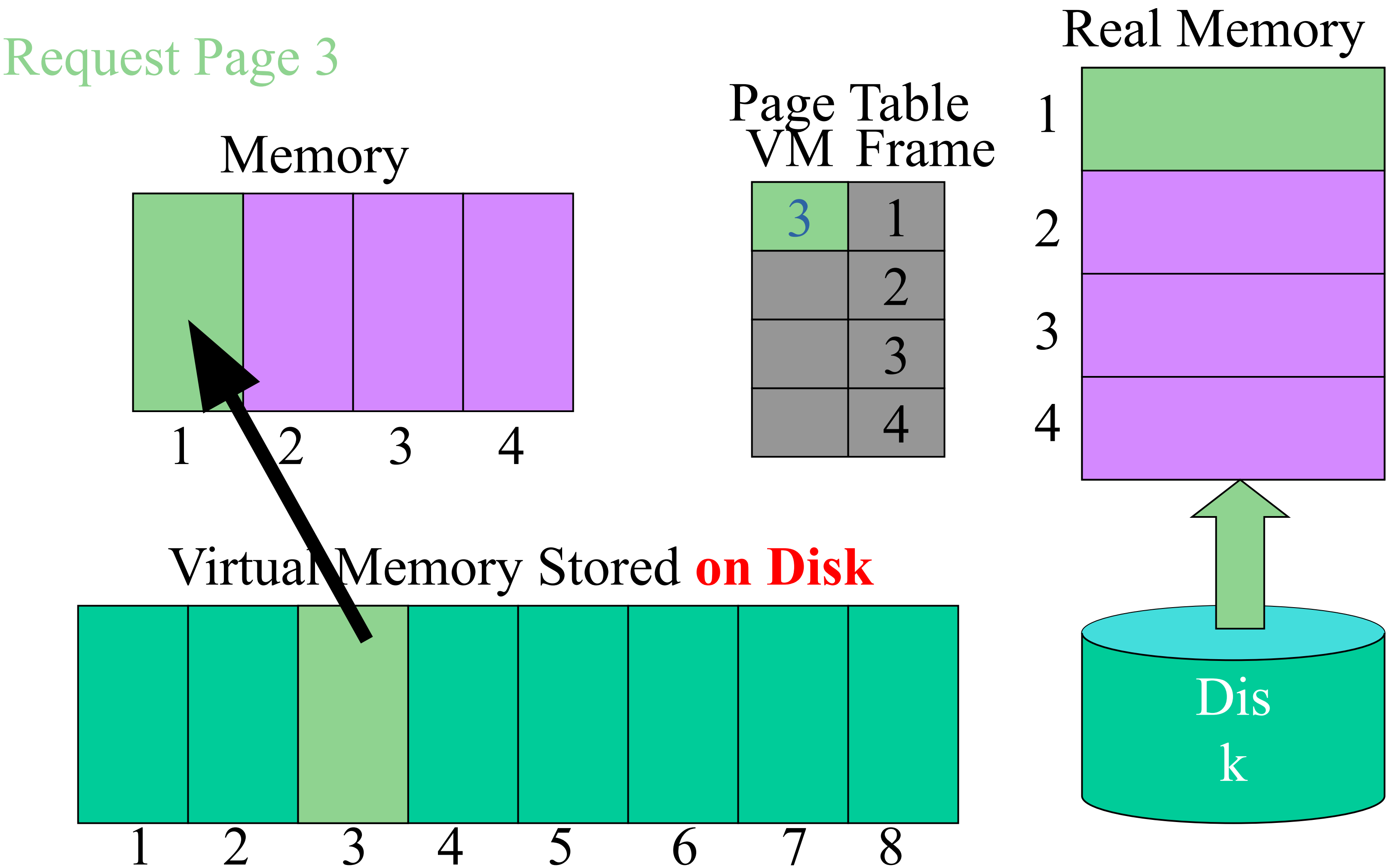
$\boxed{\text{MOV REG, 32780}}$?

Page fault & swapping

# Page Fault

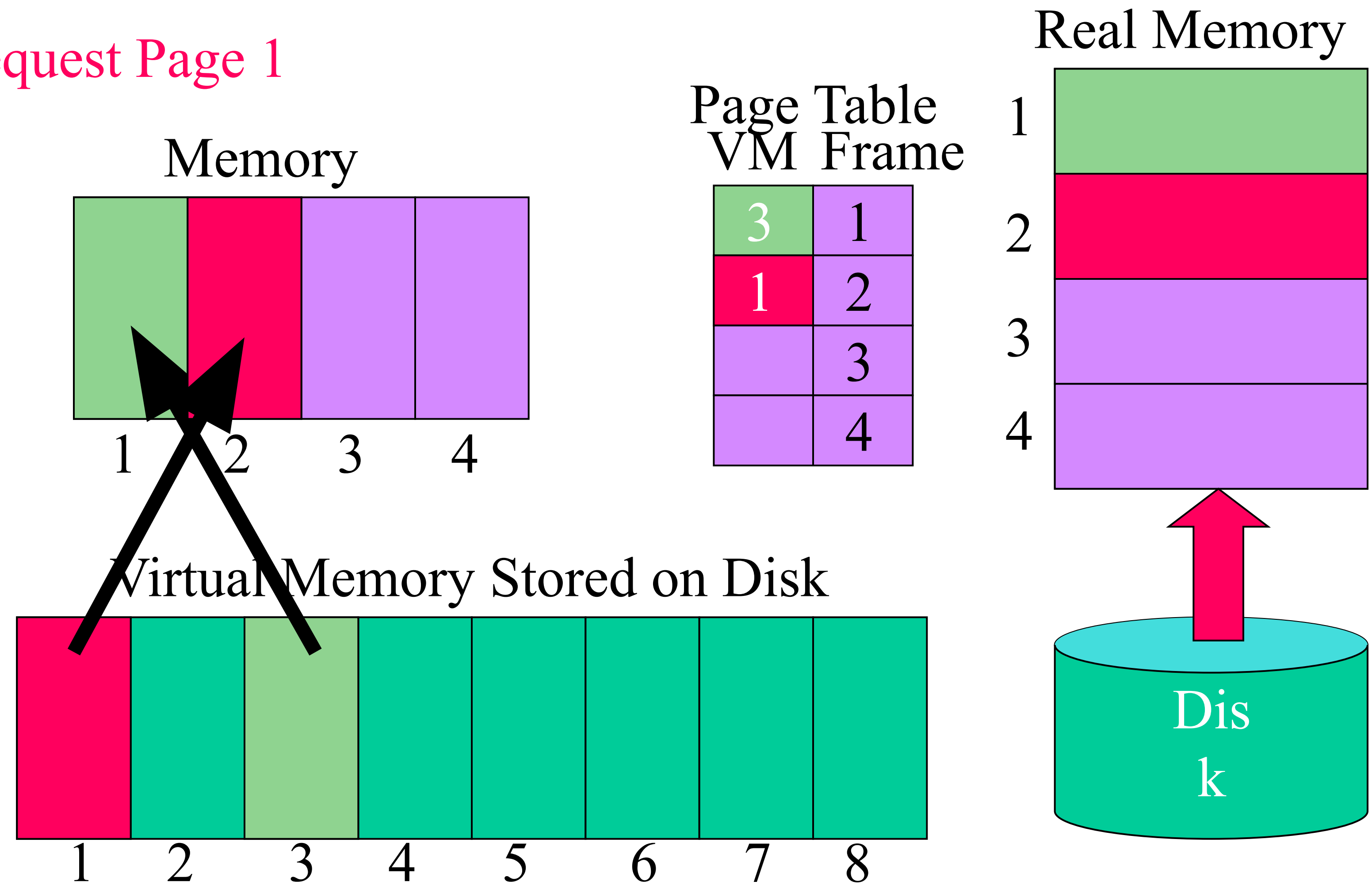
- If the MMU notices that the requested virtual page is unmapped it causes the CPU to trap to the operating system.
- This trap is called a page fault.
- The operating system
  - picks a little-used page frame and writes its contents back to the disk (if needed).
  - fetches the page just referenced into the page frame just freed
  - changes the map, and restarts the trapped instruction.

# Paging Request



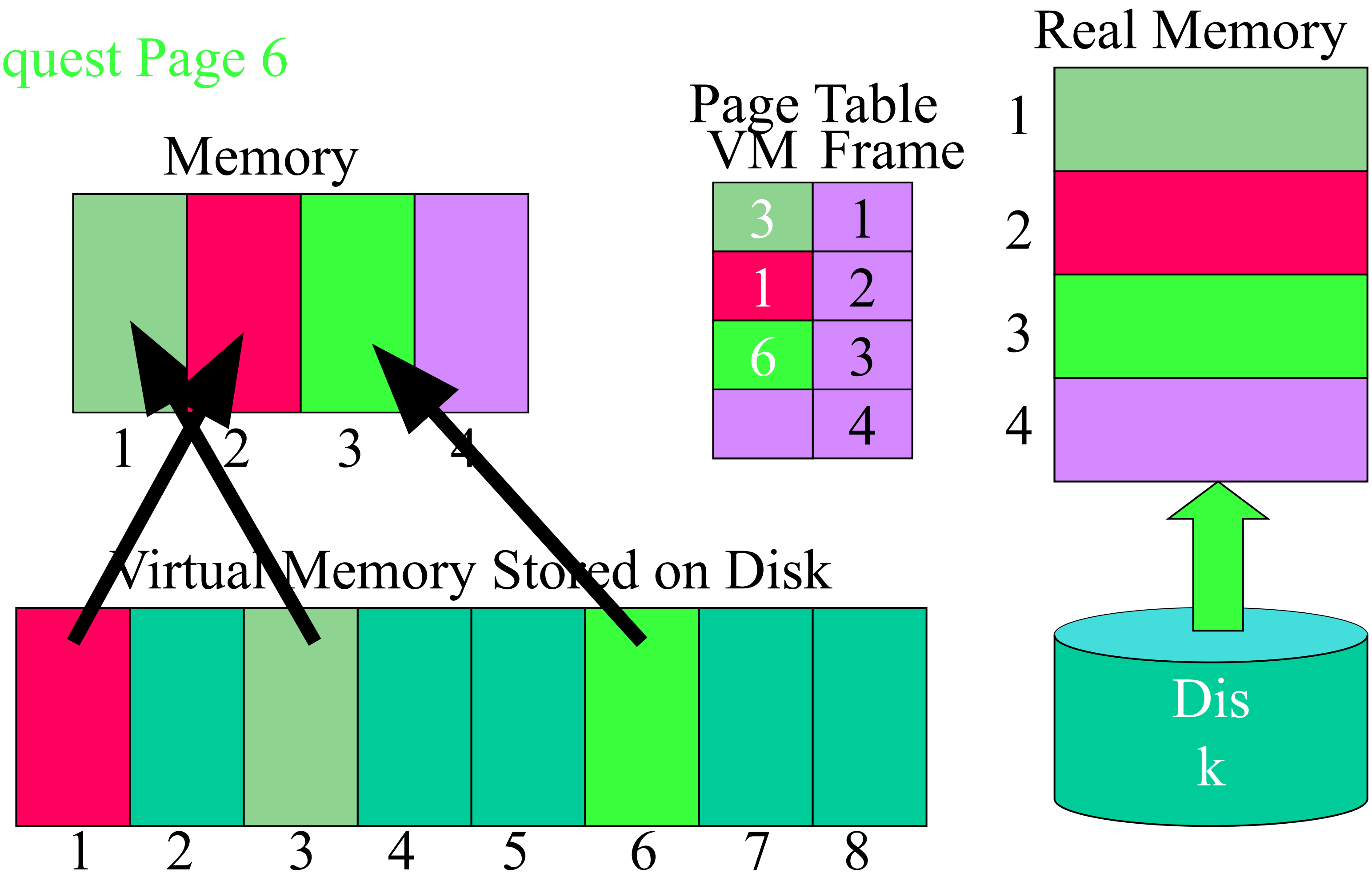
# Paging

Request Page 1



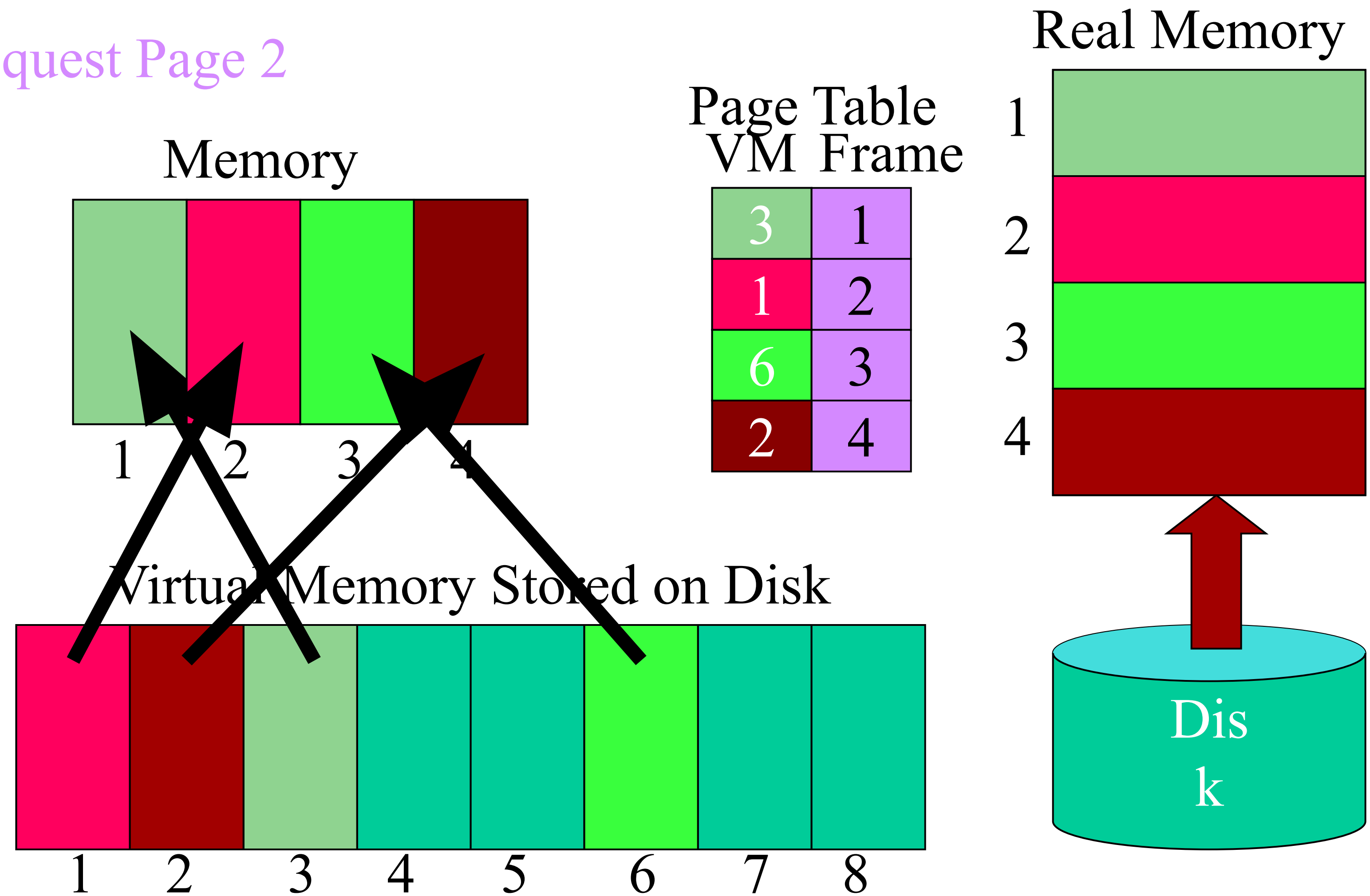
# Paging

Request Page 6



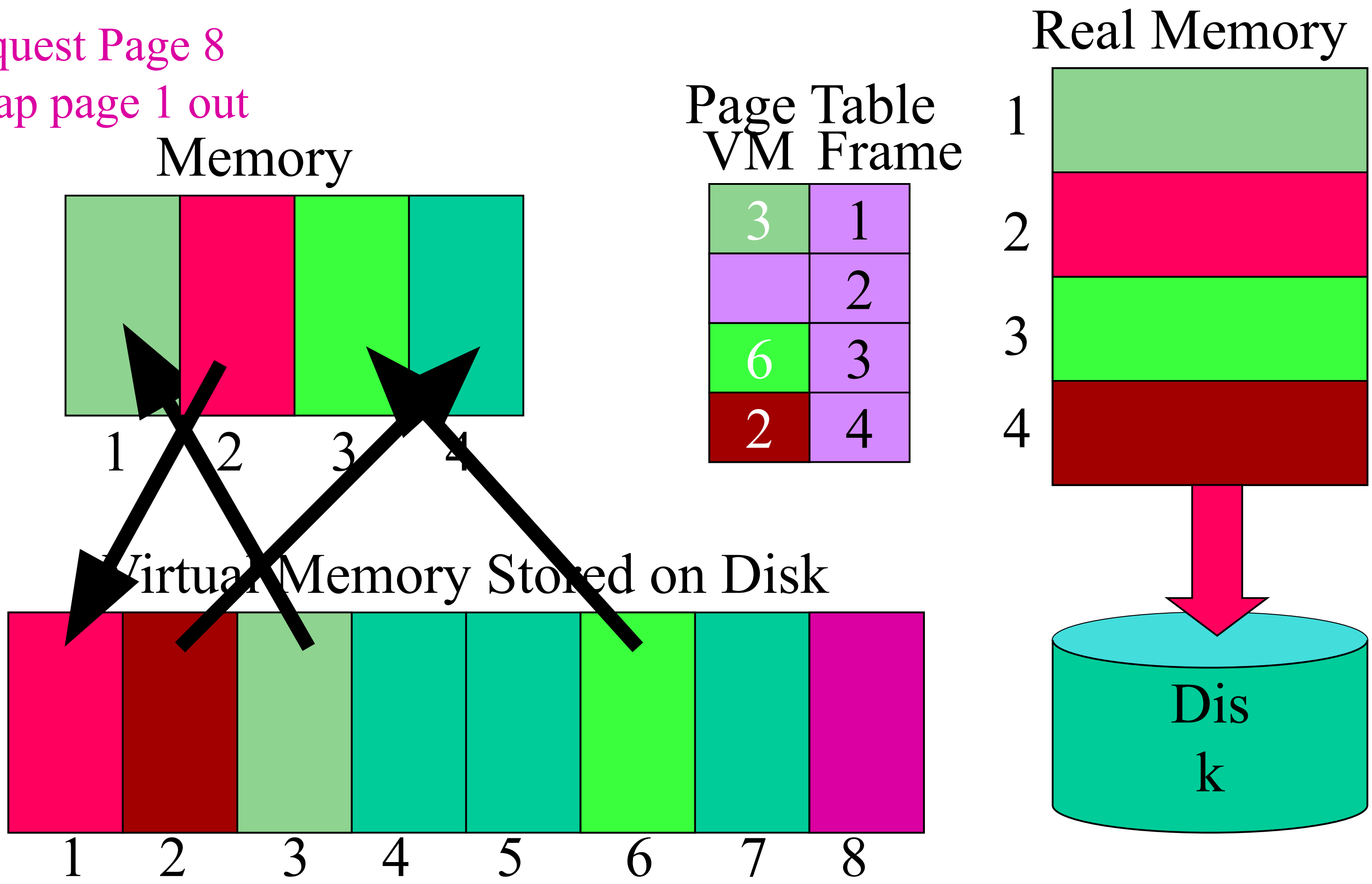
# Paging

Request Page 2



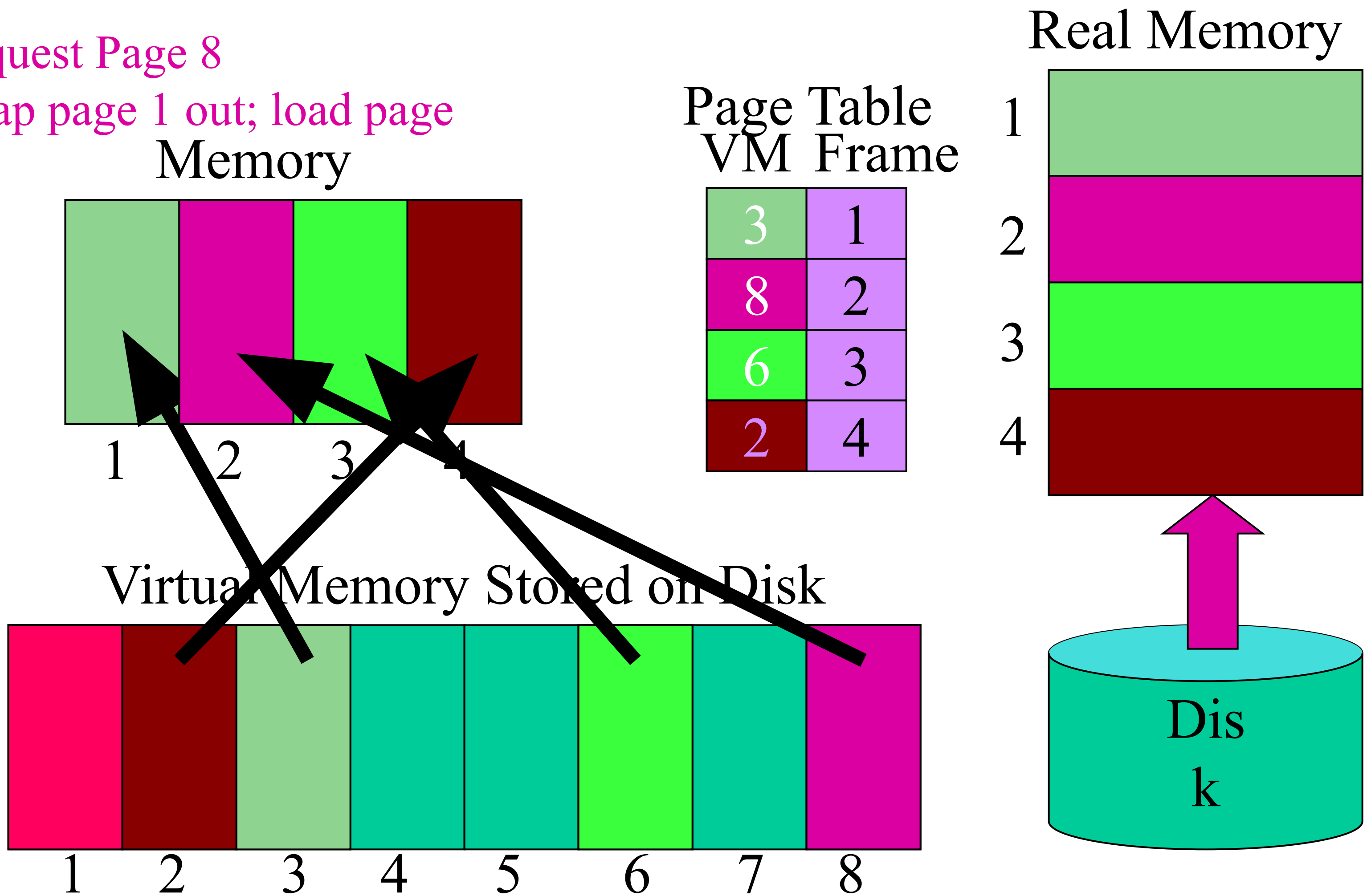
# Paging

Request Page 8
Swap page 1 out



# Paging

Request Page 8
Swap page 1 out; load page 8



# Inside the MMU: Page Table

**Address generated by CPU is divided into:**

Page number(p): an index into a *page table*

Page offset(d): to be copied into memory

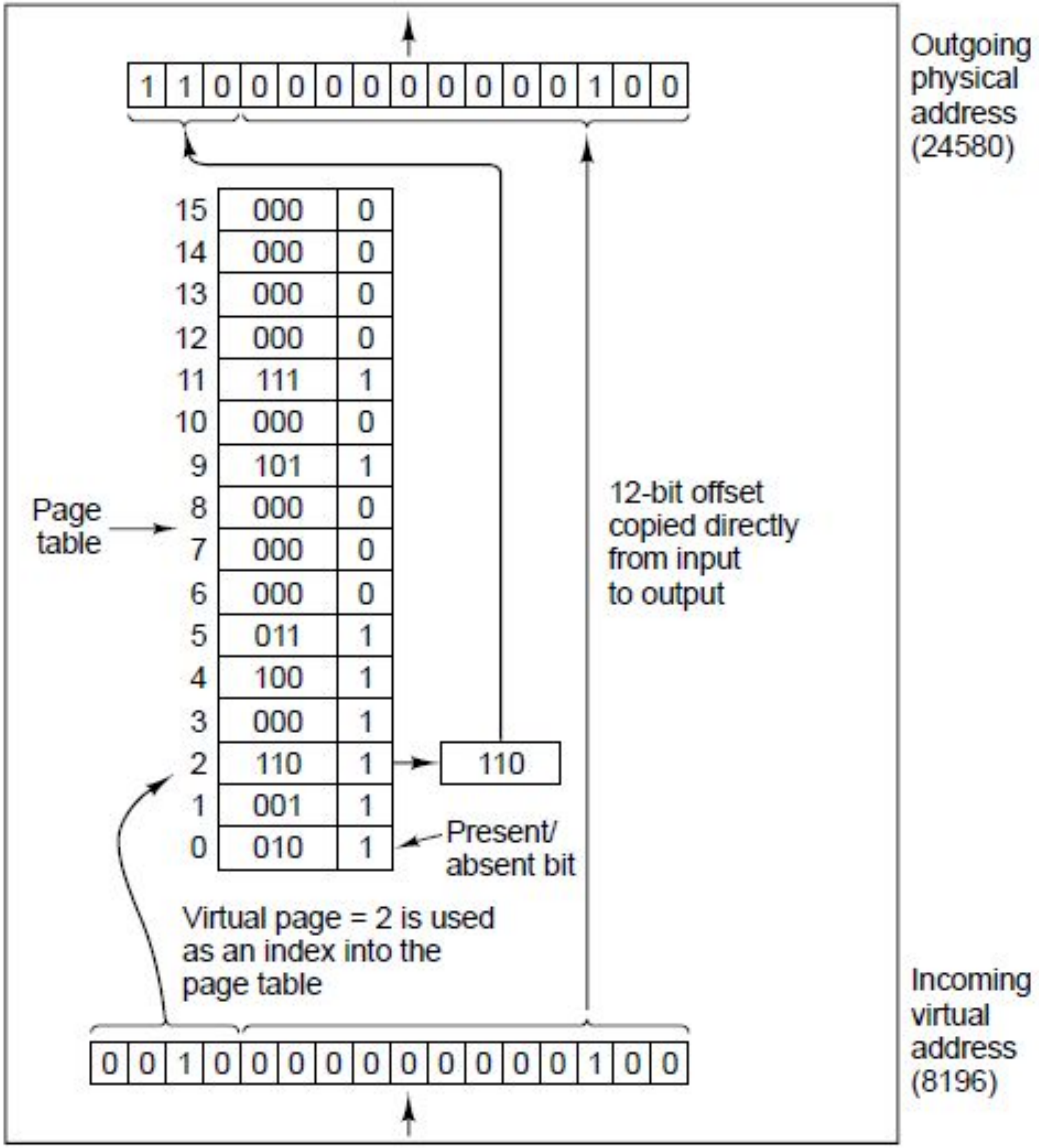
Given *logical address space* $2^m$ and *page size* $2^n$,

$$\text{number of pages} = \frac{2^m}{2^n} = 2^{m-n}$$

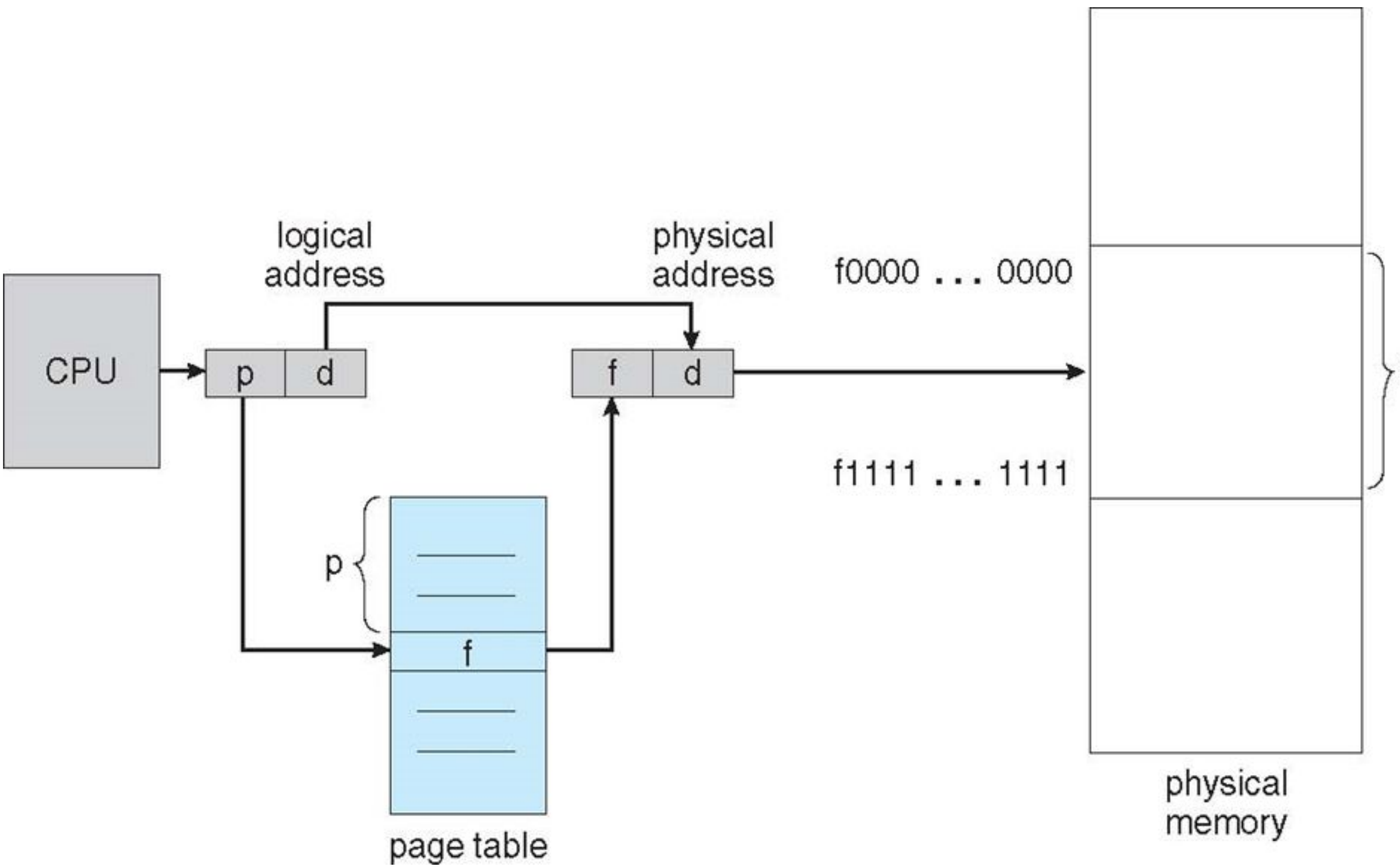
**Example: addressing to 0010000000000100**

$$\underbrace{\overbrace{0\,0\,1\,0}^{m-n=4}\ \overbrace{0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0}^{n=12}}_{m=16}$$

$$\text{page number} = 0010 = 2, \quad \text{page offset} = 000000000100$$

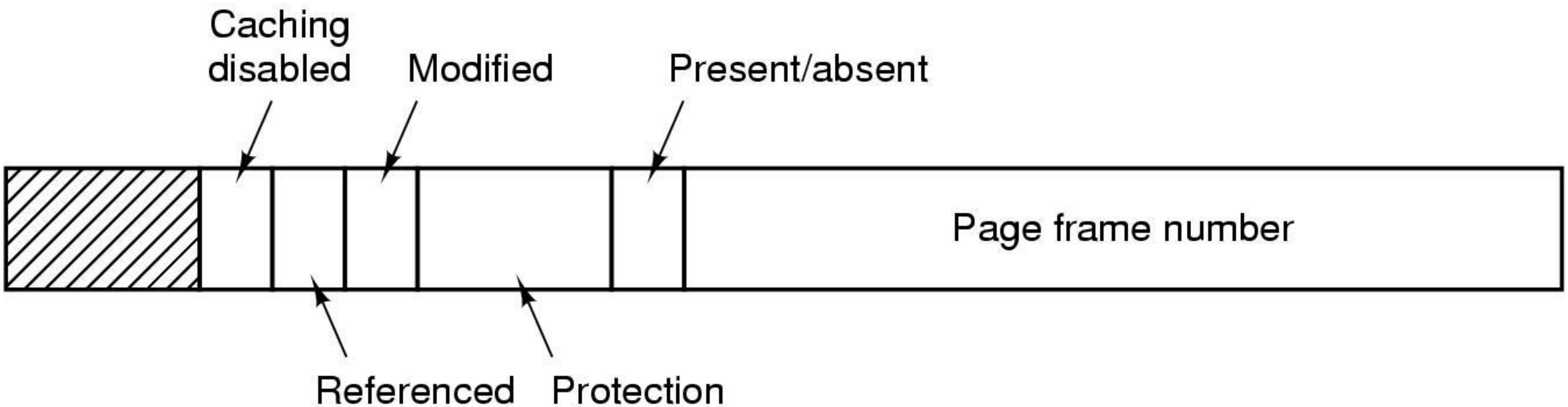



| | |
|---|---|
| Virtual pages: | 16 |
| Page size: | 4k |
| Virtual memory: | 64K |
| Physical frames: | 8 |
| Physical memory: | 32K |


CPU
logical address
p
d
physical address
f
d
f0000 . . . 0000
f1111 . . . 1111
f
p
f
page table
physical memory

# Page Tables Entry

- Entry for each virtual page in the page table



# Paging

- 2 major issues
  - The mapping from virtual address to physical address must be fast.
  - If the virtual address space is large, the page table will be large.