KOÇ UNIVERSITY

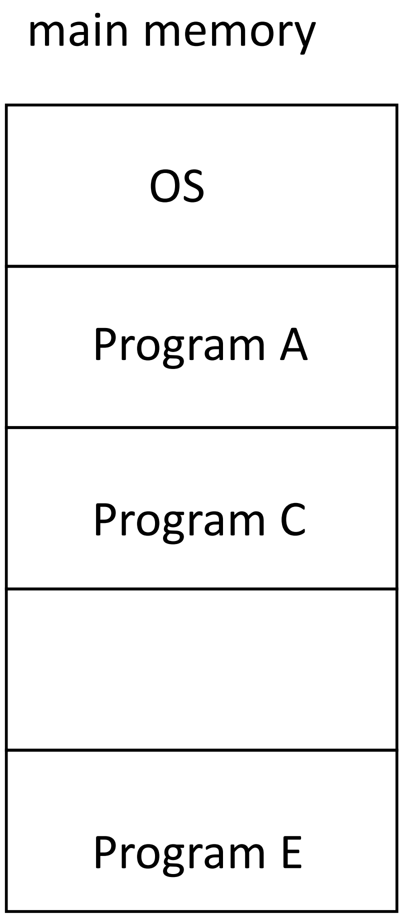
Virtual Memory Management

Didem Unat

Lecture 20

COMP 304 Operating Systems

Main Memory



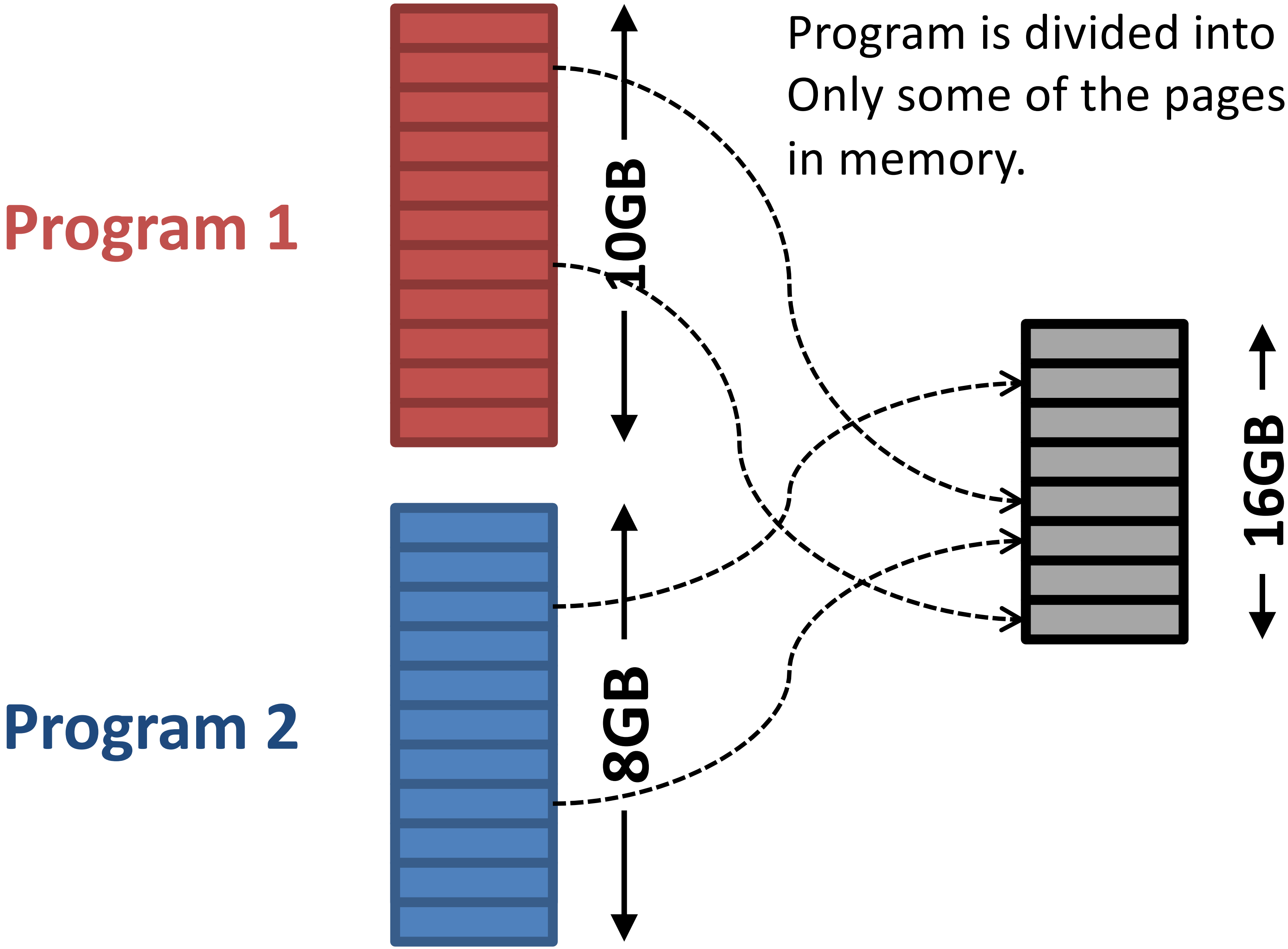
- Both data and code reside in main memory for a program
- In main memory, there are multiple programs (processes) exist at the same time
- Operating System (OS) as well needs some space in the main memory

Memory

- What happens if your program or its data is larger than the main memory?
- How do you protect a program's data in the memory from other programs?
- Virtual Memory
  - Use main memory as a "cache" for secondary memory (disk)
  - Allows efficient and safe sharing of memory among multiple programs
  - Provides the ability to easily run programs larger than the size of physical memory
  - Automatically handles bringing in data from disk

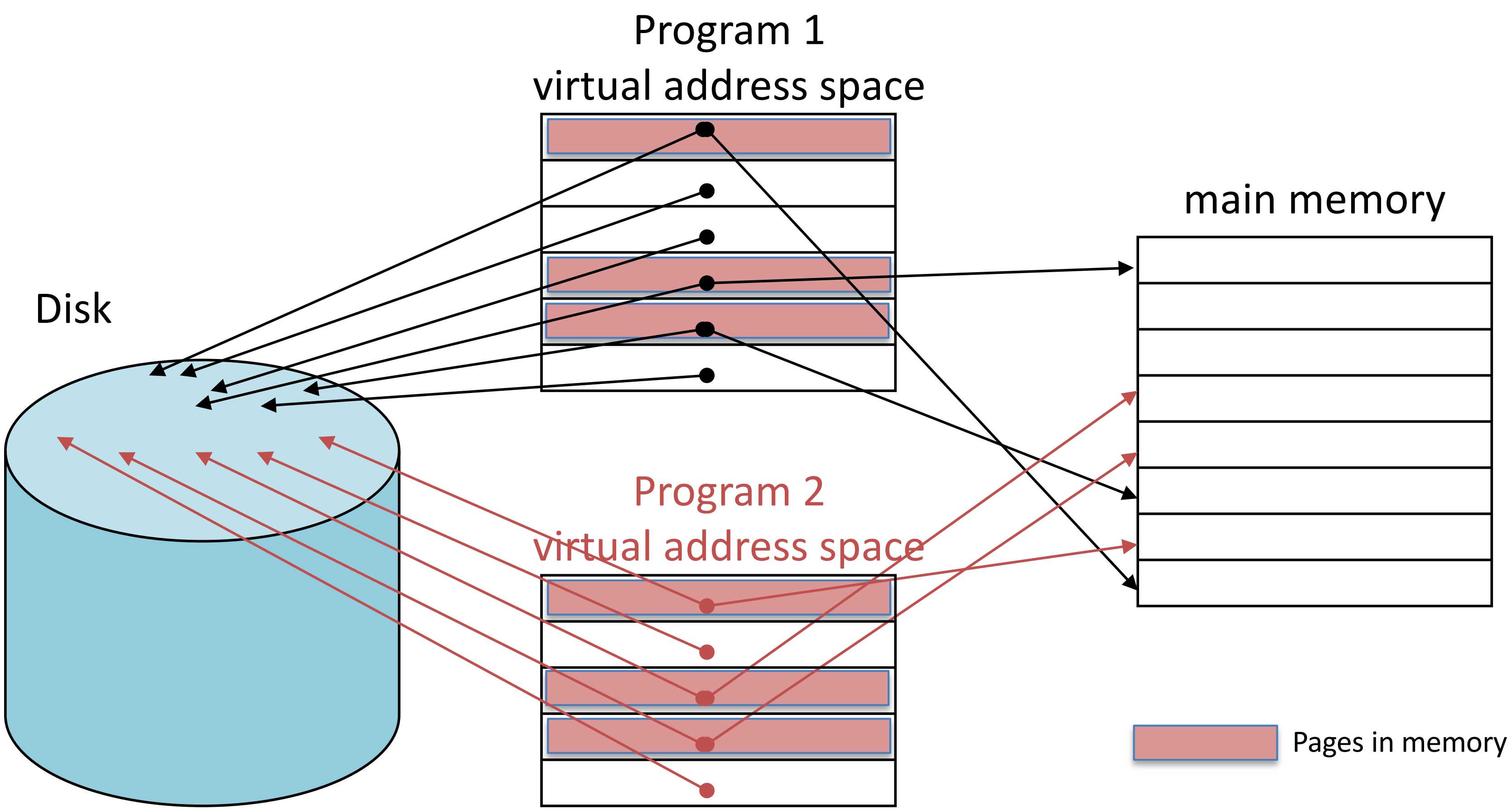
Paging

Program is divided into pages.
Only some of the pages reside in memory.



Two Programs Sharing Physical Memory

- Each program is compiled into its own address space – a "virtual" address space
  - Address space is divided into pages
- All these pages are in disk but only some of them are in main memory (physical memory) during program execution



Overview of Paging (cont'd)

1. Based on the notion of a **virtual address space**
   - A large, contiguous address space that is only an illusion
     - Virtual address space >> Physical address space
   - Each "program" gets its own separate virtual address space
     - Each **process**, not each thread

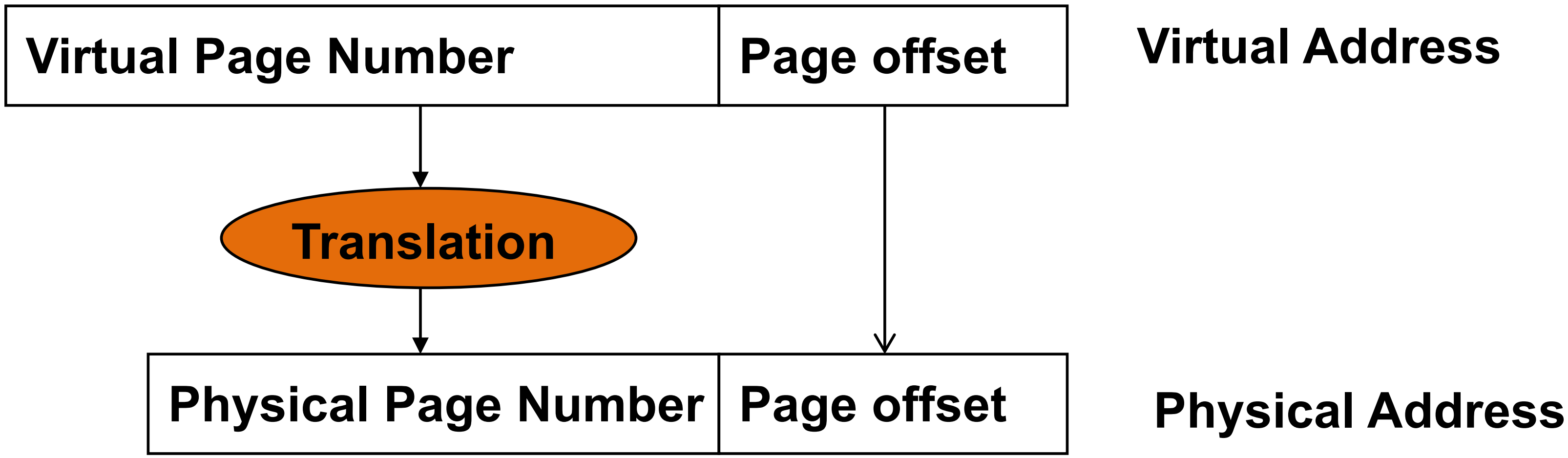
2. Divide the address spaces into fixed-sized **pages**
   - **Virtual page**: A "chunk" of the virtual address space
   - **Physical page**: A "chunk" of the physical address space
     - Also called a **frame**
   - Size of virtual page == Size of physical page

Overview of Paging (cont'd)

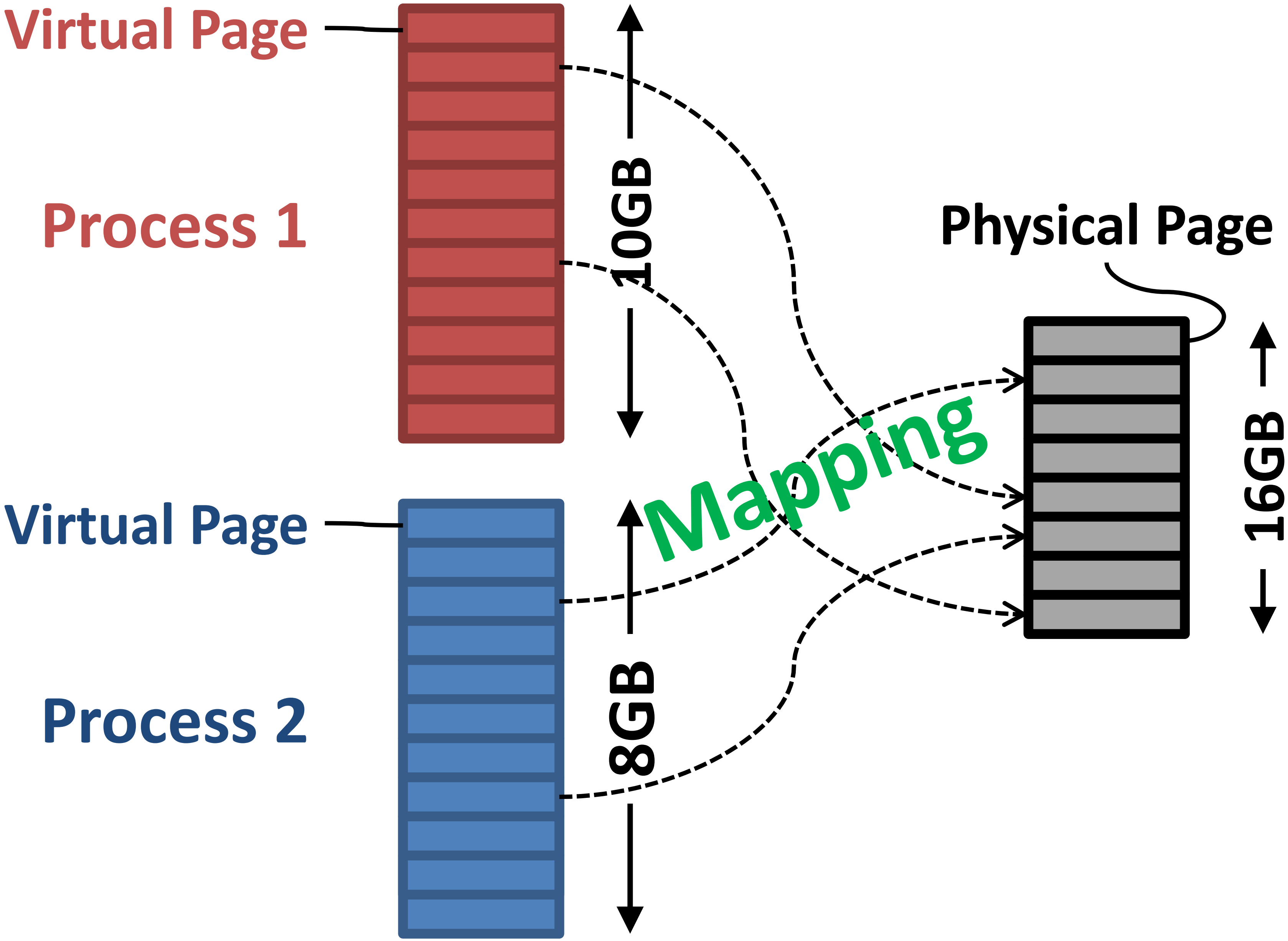
3. **Map** virtual pages to physical pages
   - By itself, a virtual page is merely an illusion
     - Cannot actually store anything
     - Needs to be backed-up by a physical page
   - Before a virtual page can be accessed …
     - It must be paired with a physical page
     - I.e., it must be **mapped** to a physical page
     - This mapping is stored somewhere (at OS memory space)
   - On every subsequent access to the virtual page …
     - Its mapping is looked up
     - Then, the access is directed to the physical page

Virtual and Physical Addresses

- So each memory request *first* requires an address translation from the virtual space to the physical space
- Translation is done by a combination of hardware and OS support

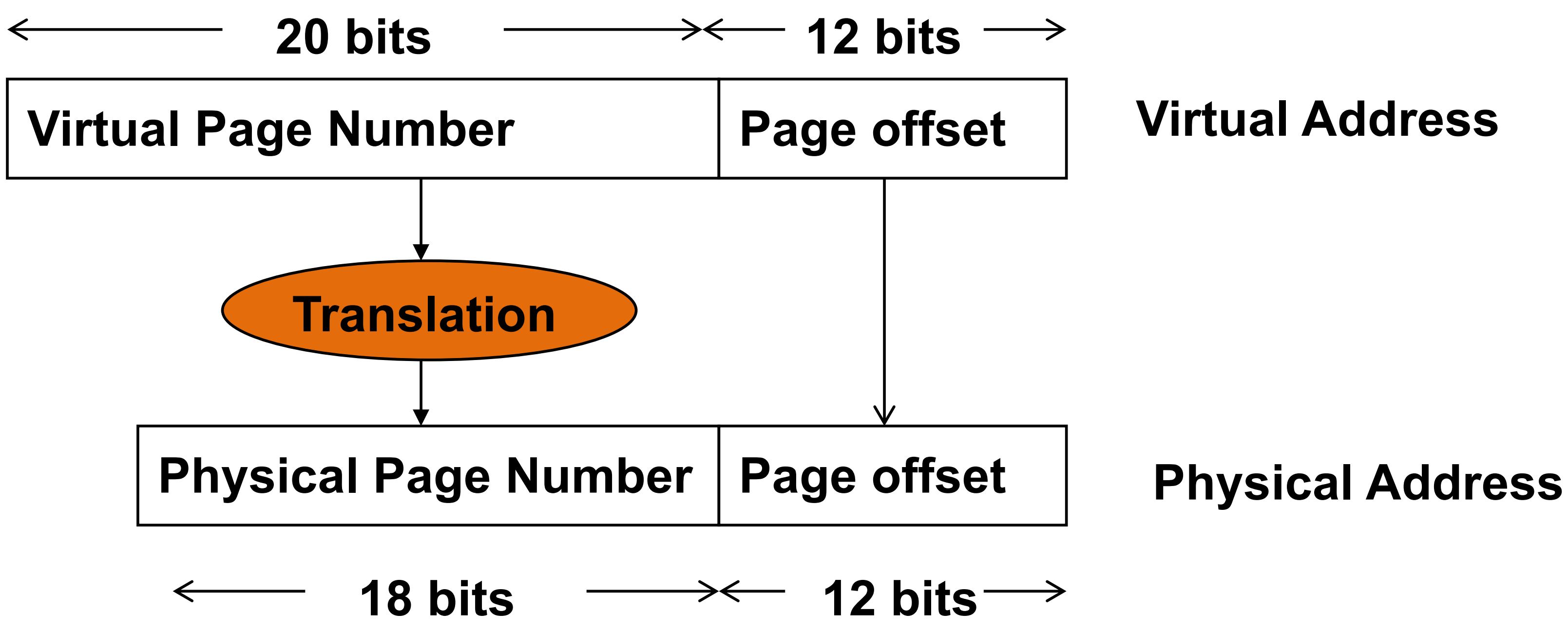


Overview of Paging (cont'd)



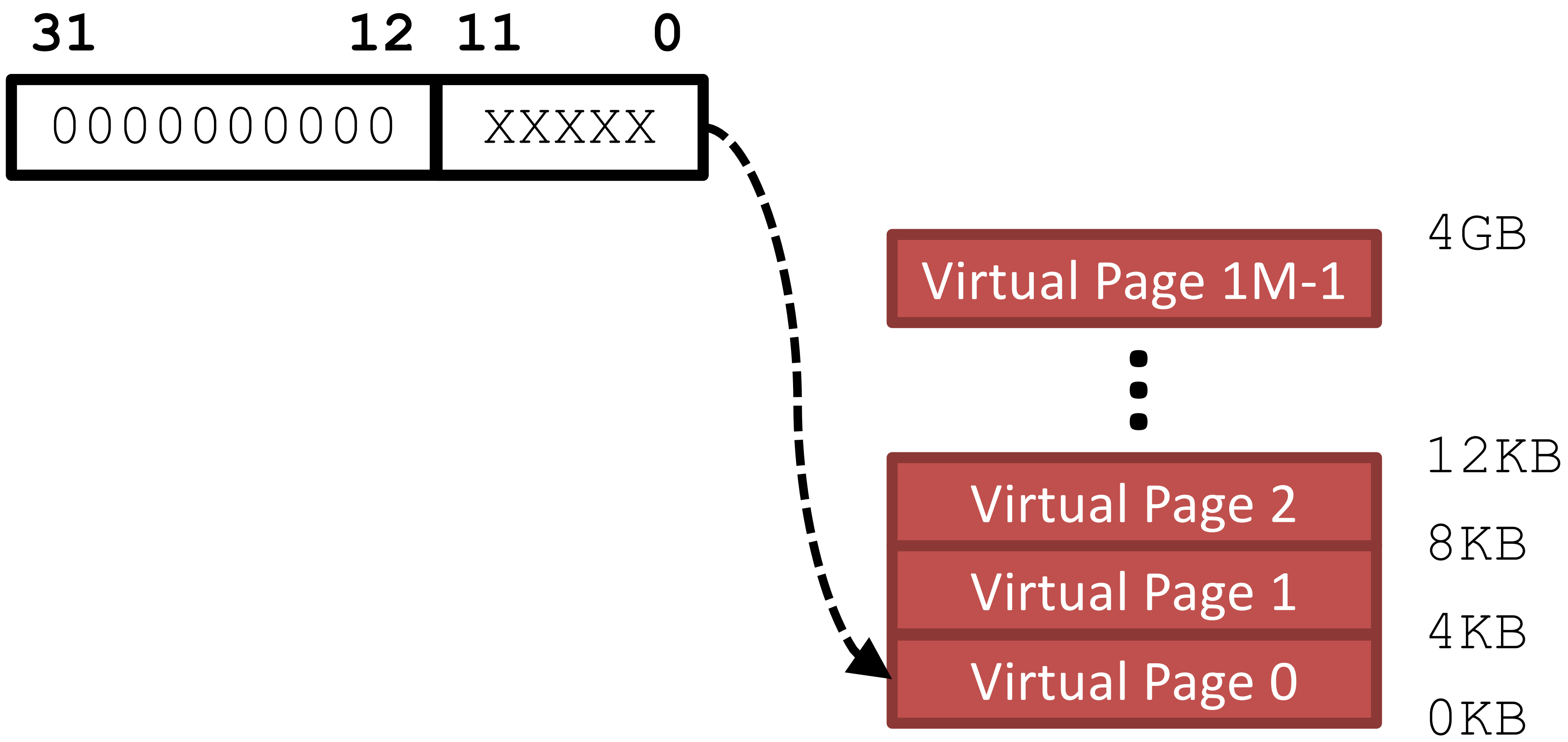
Virtual and Physical Addresses

- What is the page size?
  - 2^12 = 4 KB
- How many pages are allowed in physical memory?
  - 2^18 pages, thus physical address space = #pages * page size = 1 GB
- How many pages are allowed in virtual address space?
  - 2^20 pages, virtual address space = #pages * page size = 4 GB



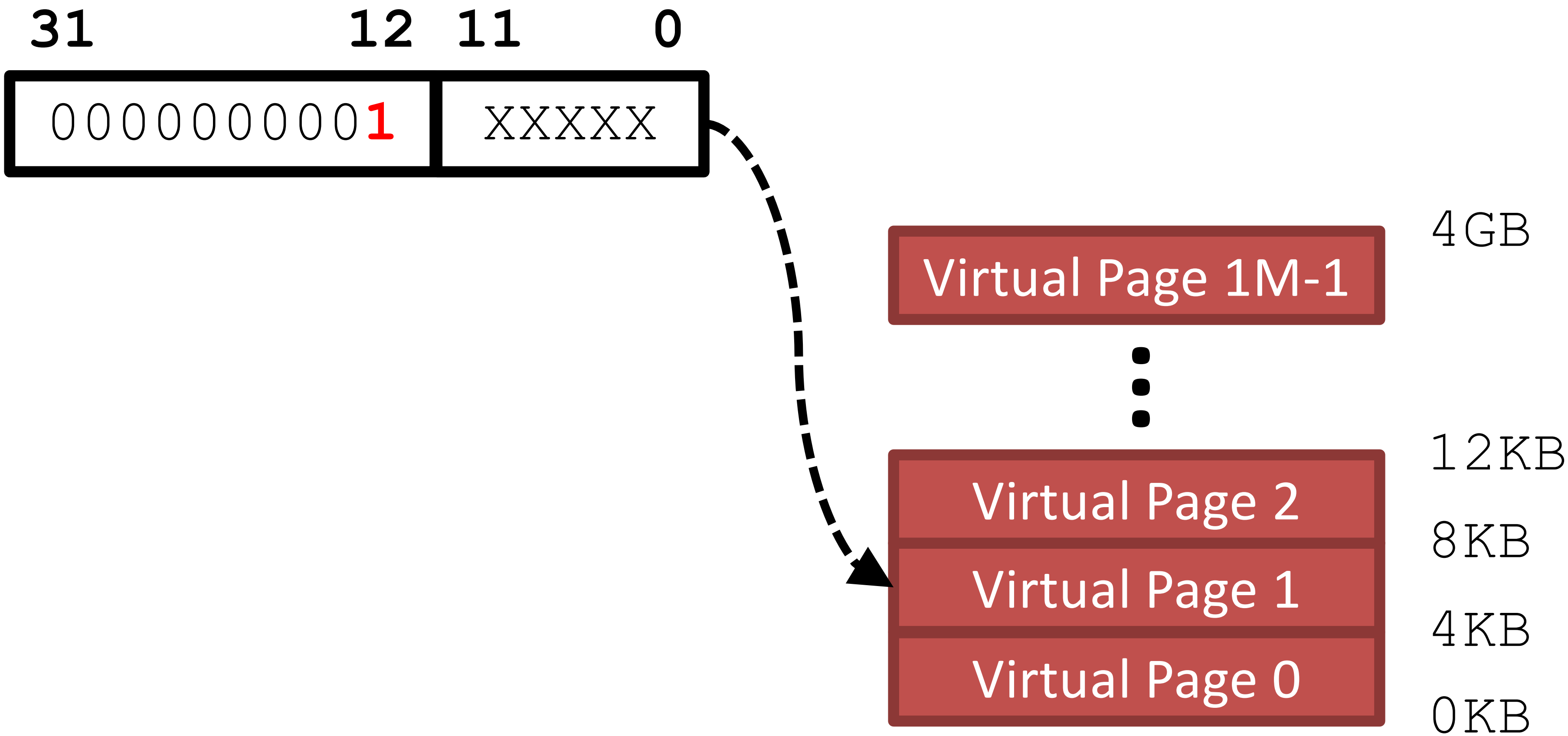
Intel 80386: Virtual Pages

32-bit Virtual Address



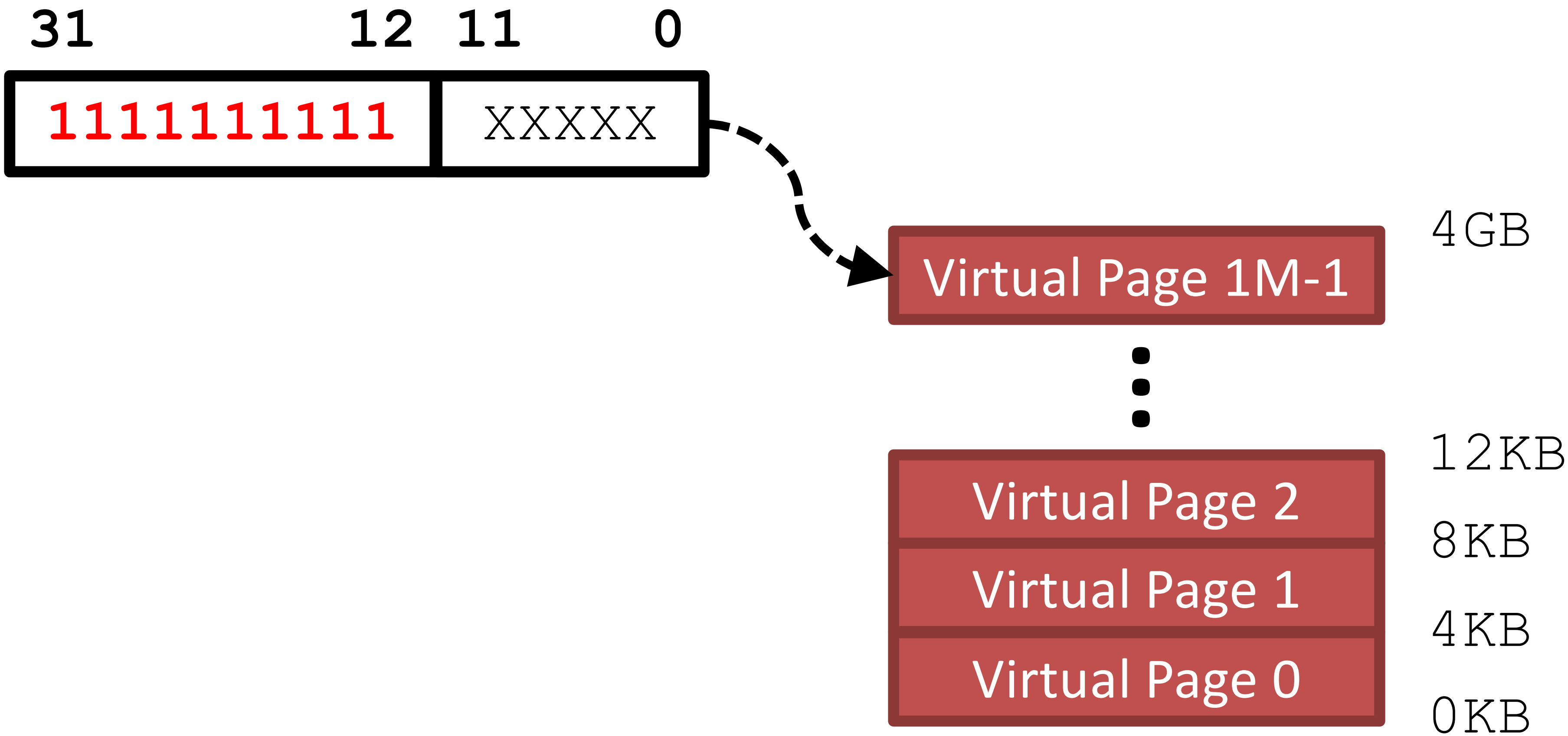
Intel 80386: Virtual Pages

32-bit Virtual Address

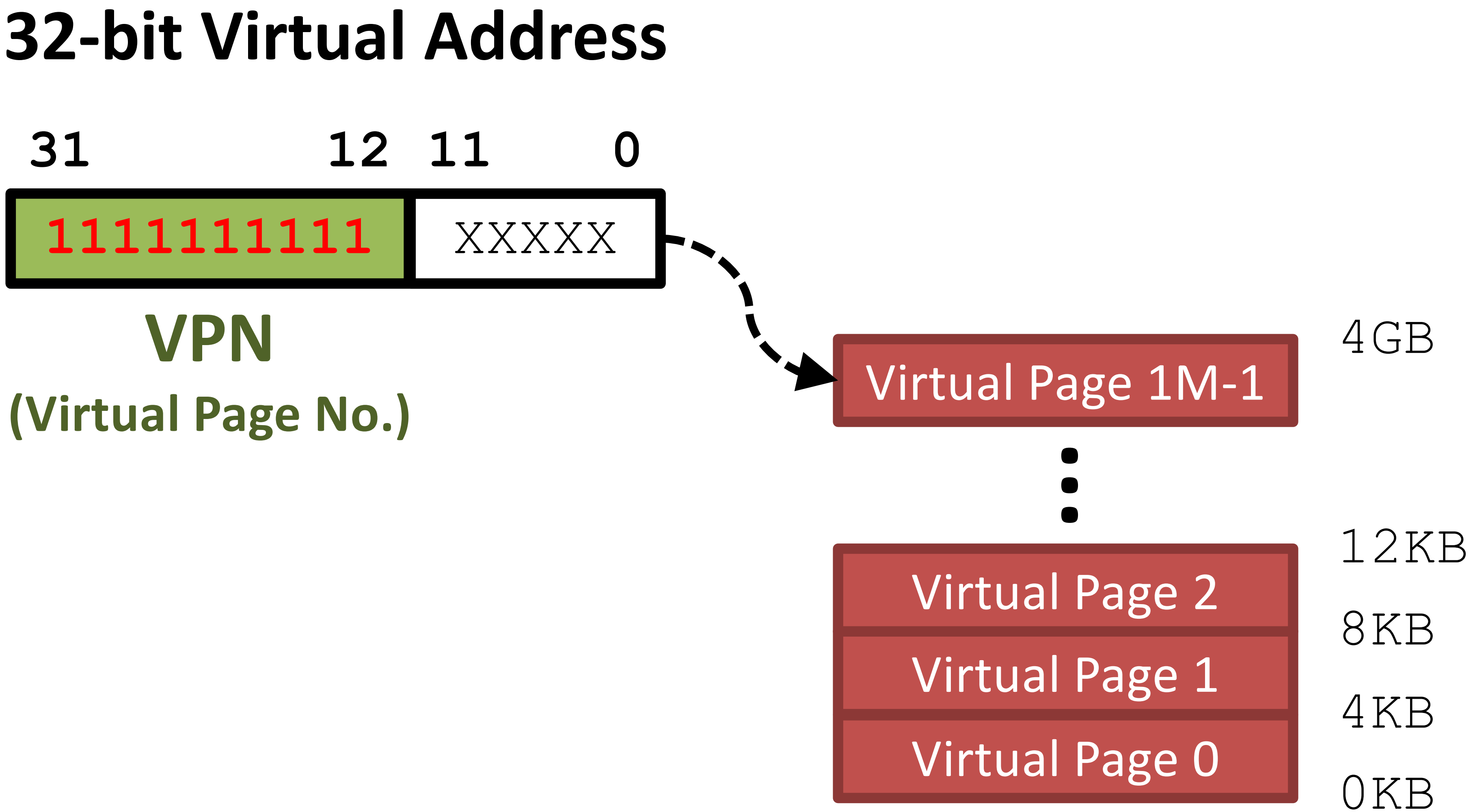


Intel 80386: Virtual Pages

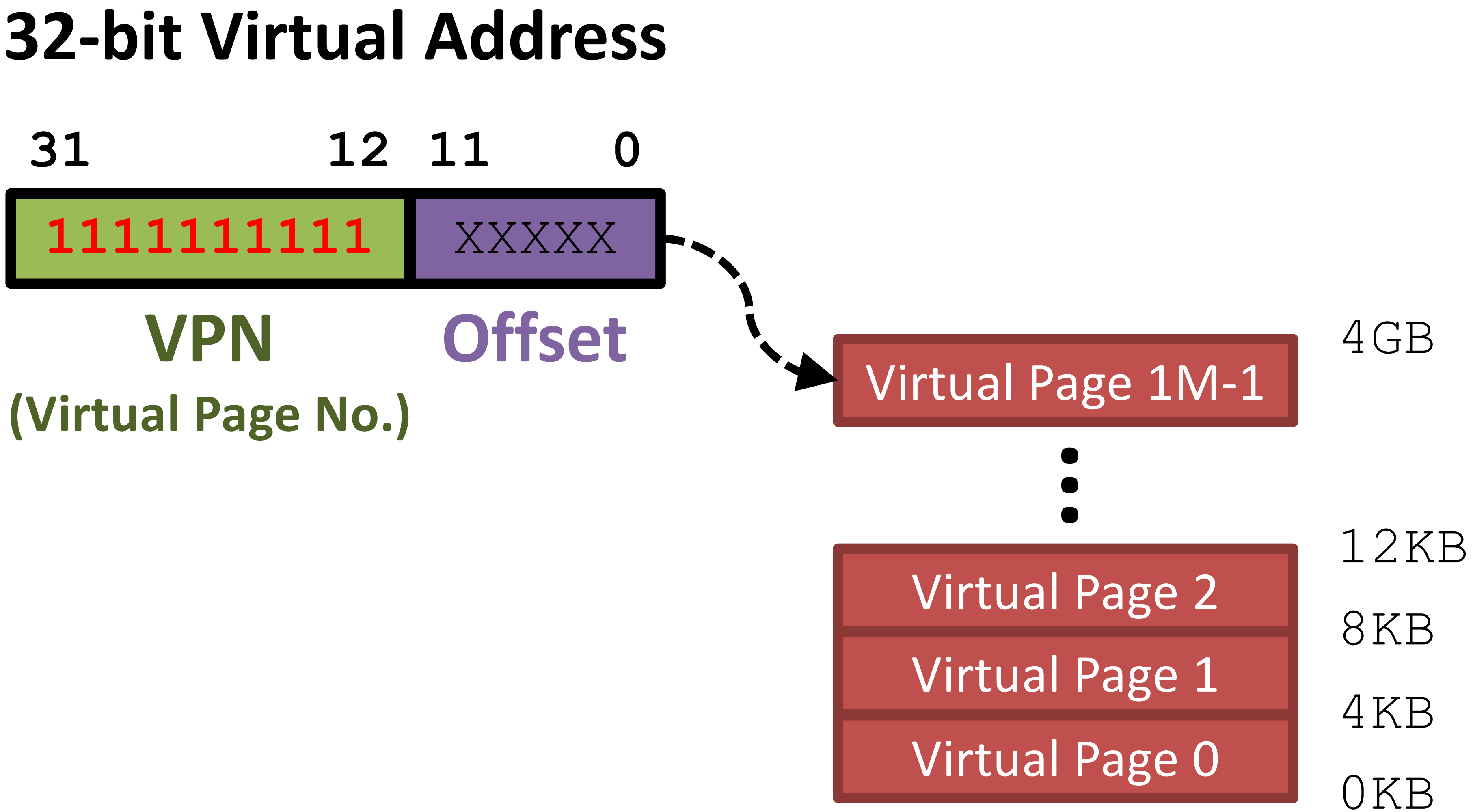
32-bit Virtual Address



Intel 80386: Virtual Pages

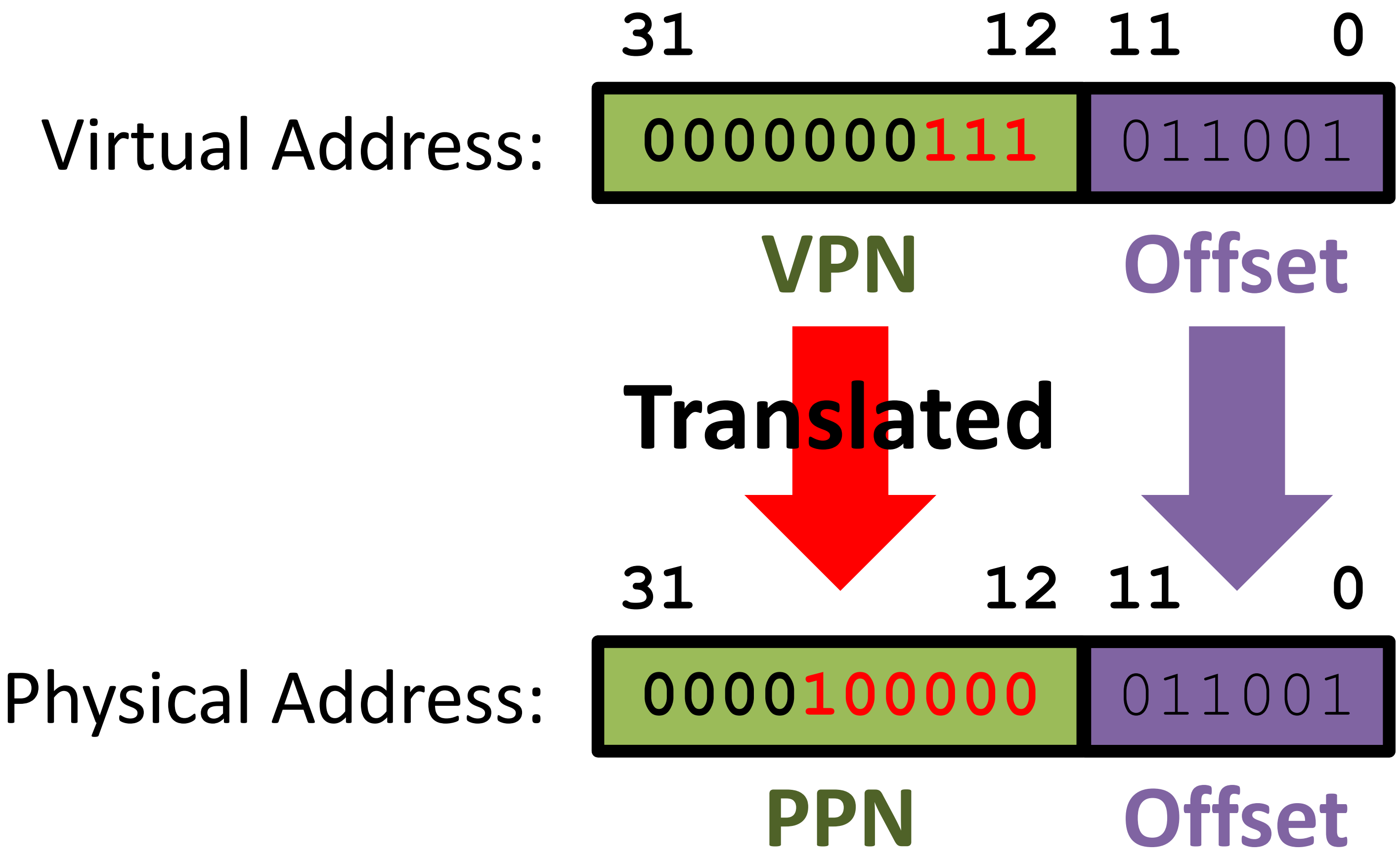


Intel 80386: Virtual Pages



Intel 80386: Translation

- Assume: Virtual Page 7 is mapped to Physical Page 32
- For an access to Virtual Page 7 …



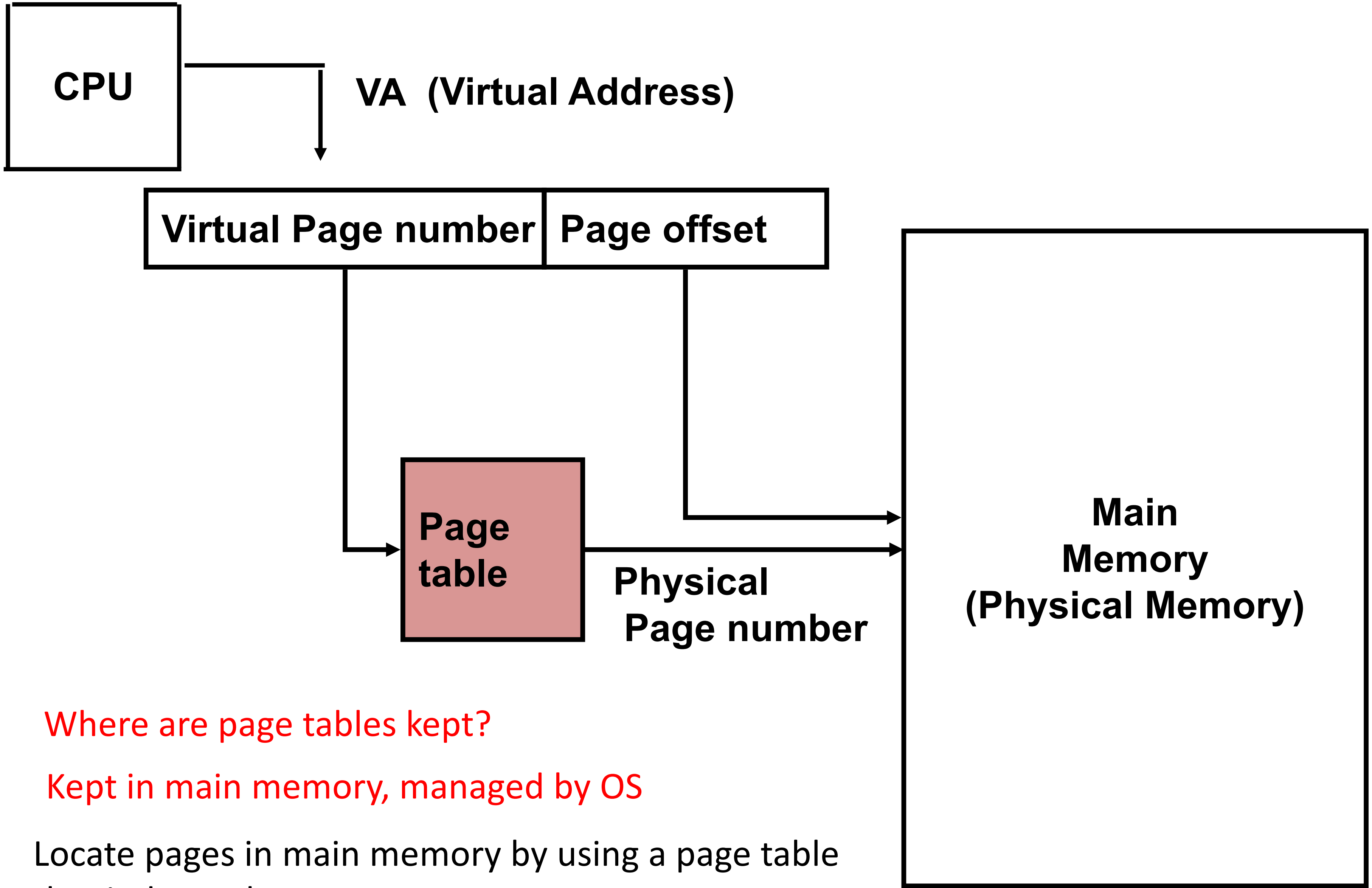
VPN → PPN

- How to keep track of VPN → PPN mappings?
  - VPN 65 → PPN 981,
  - VPN 3161 → PPN 1629,
  - VPN 9327 → PPN 524, …

- **Page Table:** A "lookup table" for the mappings
  - Can be thought of as an array
  - Each element in the array is called a **page table entry** (PTE)

```
uint32 PAGE_TABLE[1<<20];
PAGE_TABLE[65]=981;
PAGE_TABLE[3161]=1629;
PAGE_TABLE[9327]=524; ...
```

Address Translation



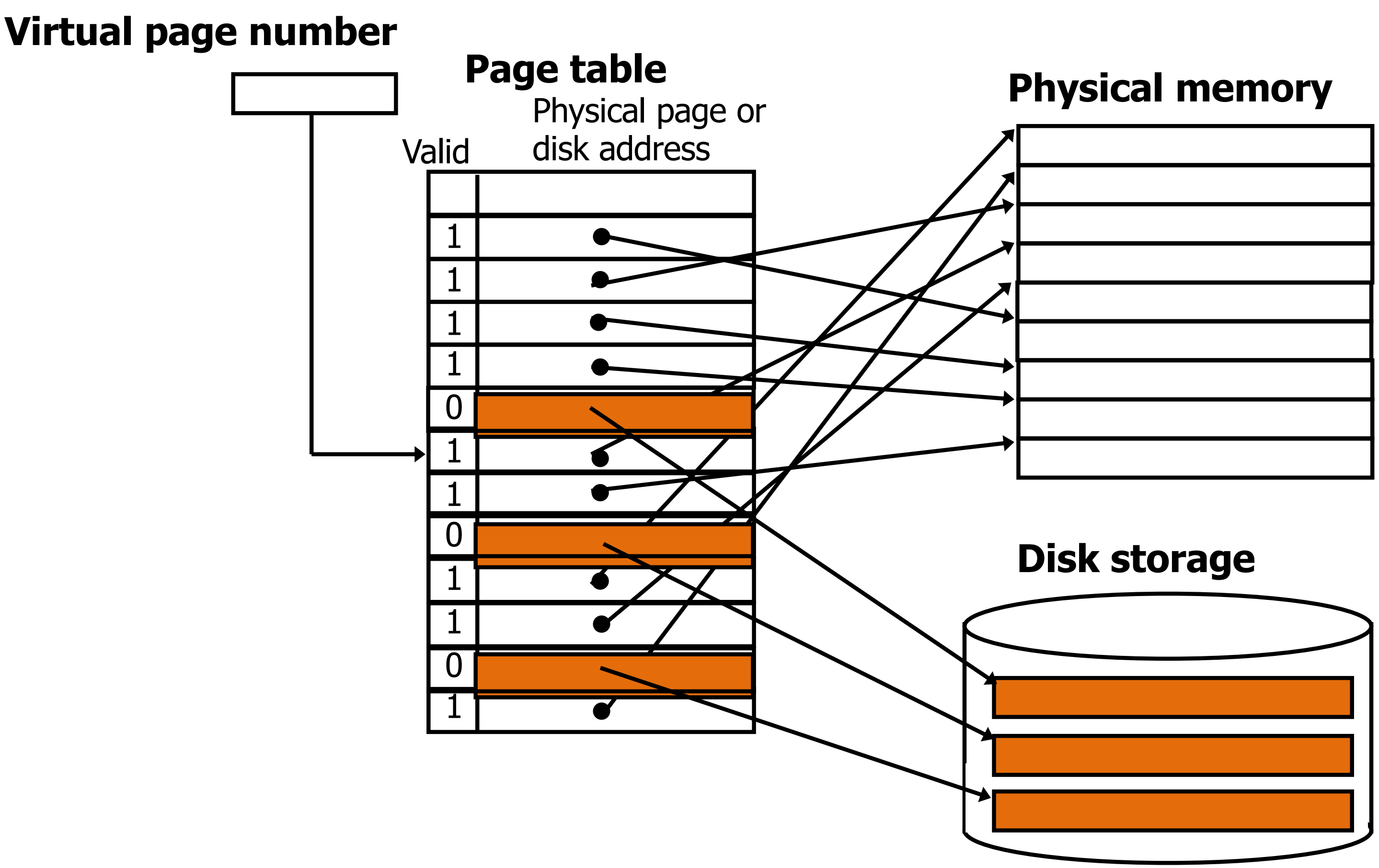
Where are page tables kept?

Kept in main memory, managed by OS

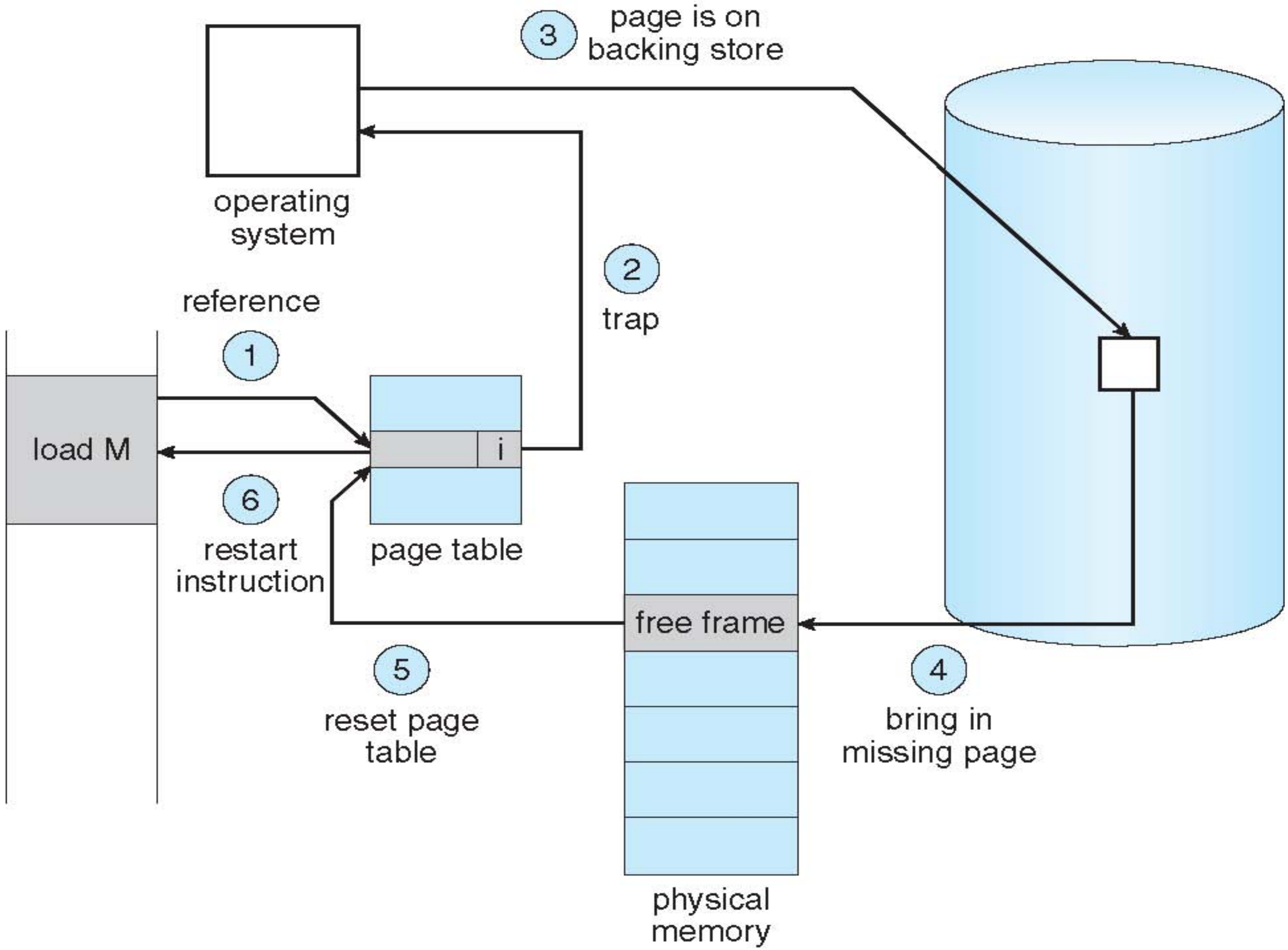
Locate pages in main memory by using a page table that indexes the memory

Page Fault

- If the valid bit of the page table is zero, this means that the page is **not in main memory**.
- In this case of a reference to an invalid page, then a page fault occurs, and the missing page is read in from disk.



Steps in Handling a Page Fault



Page Fault (more in detailed)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

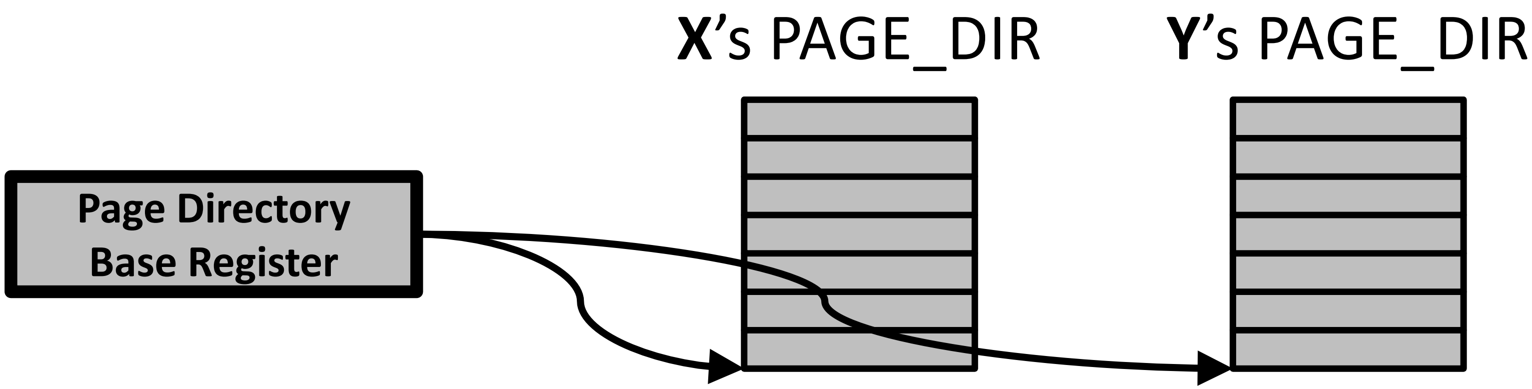
Determining Page Table Size

- Assume
  - 40-bit virtual address
  - 30-bit physical address
  - 8 KB pages
  - Each page table entry is one word (4 bytes)
- How large is the page table?
  - Page offset = 8 KB = 2^13 => 13 bit page offset
  - Virtual page number = 40 - 13 = 27 bits
  - Number of entries in Page Table= number of pages = 2^27
  - Total size = number of entries x bytes/entry

    = 2^27 x 4 = 512 Mbytes

Address Space of Processes

- Each process has its own virtual address space
  - Process **X**: text editor
  - Process **Y**: video player
  - **X** writing to its virtual address 0 does not affect the data stored in **Y**'s virtual address 0 (or any other address)
    - This was the entire purpose of virtual memory
  - *Each process has its own page directory and page tables*
    - When process Y starts running (process X stops running), the page directory base register's value must be updated

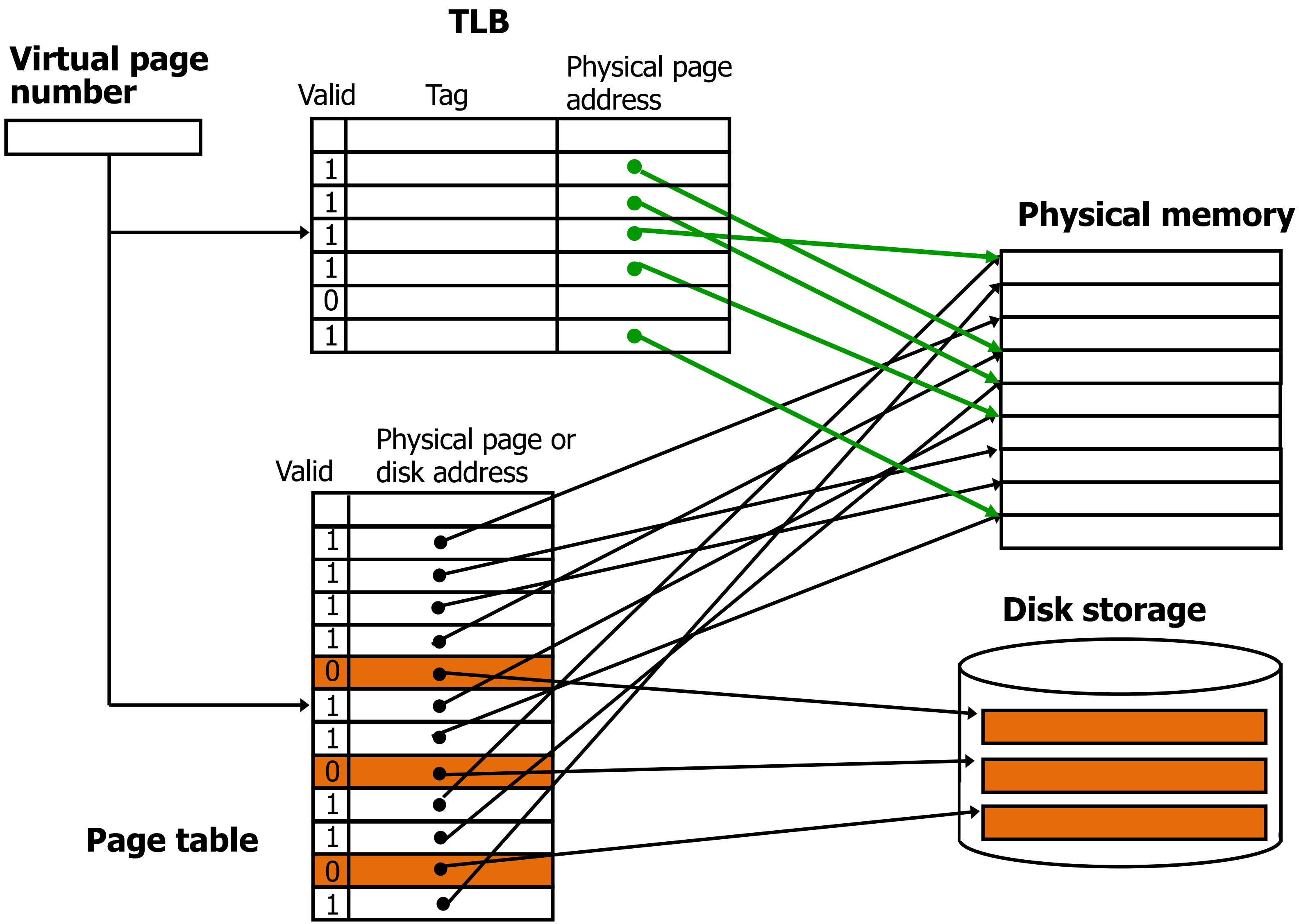


Accessing Page Tables

- Problem:
  - Need to translate Virtual addresses (VA) into Physical address (PA) for every load/store
  - Translation is done through page tables
  - Page tables are in memory!
- Accessing page tables is slow.
  - Must access memory for loads/stores-- even cache hits!
  - Worse, if translation is not completely in memory, may need to go to disk before hitting in cache!
    - Page tables are kept in pages as well
- What do we do?

Translation-Lookaside Buffer (TLB)

- A TLB acts as a cache for the page table, by storing physical addresses of pages that have been recently accessed.



Acknowledgments

- These slides are adapted from
  - *Computer Organization and Design, 5th Edition*, Patterson & Hennessy, © 2008, MK
  - Prof. Mary Jane Irwin
  - Prof. Gokhan Memik
  - Prof. Onur Mutlu
  - Wikipedia