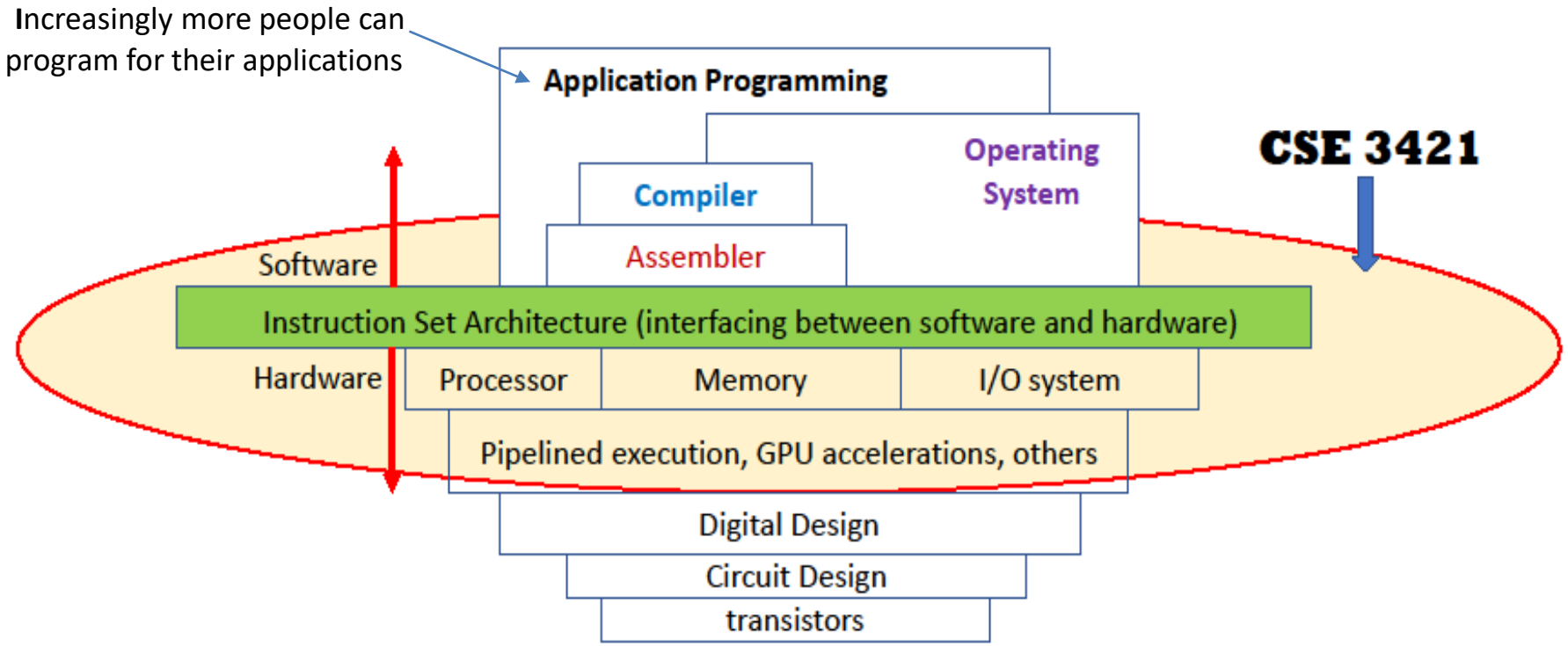
CSE3421
Introduction to Computer Architecture

Virtual Memory

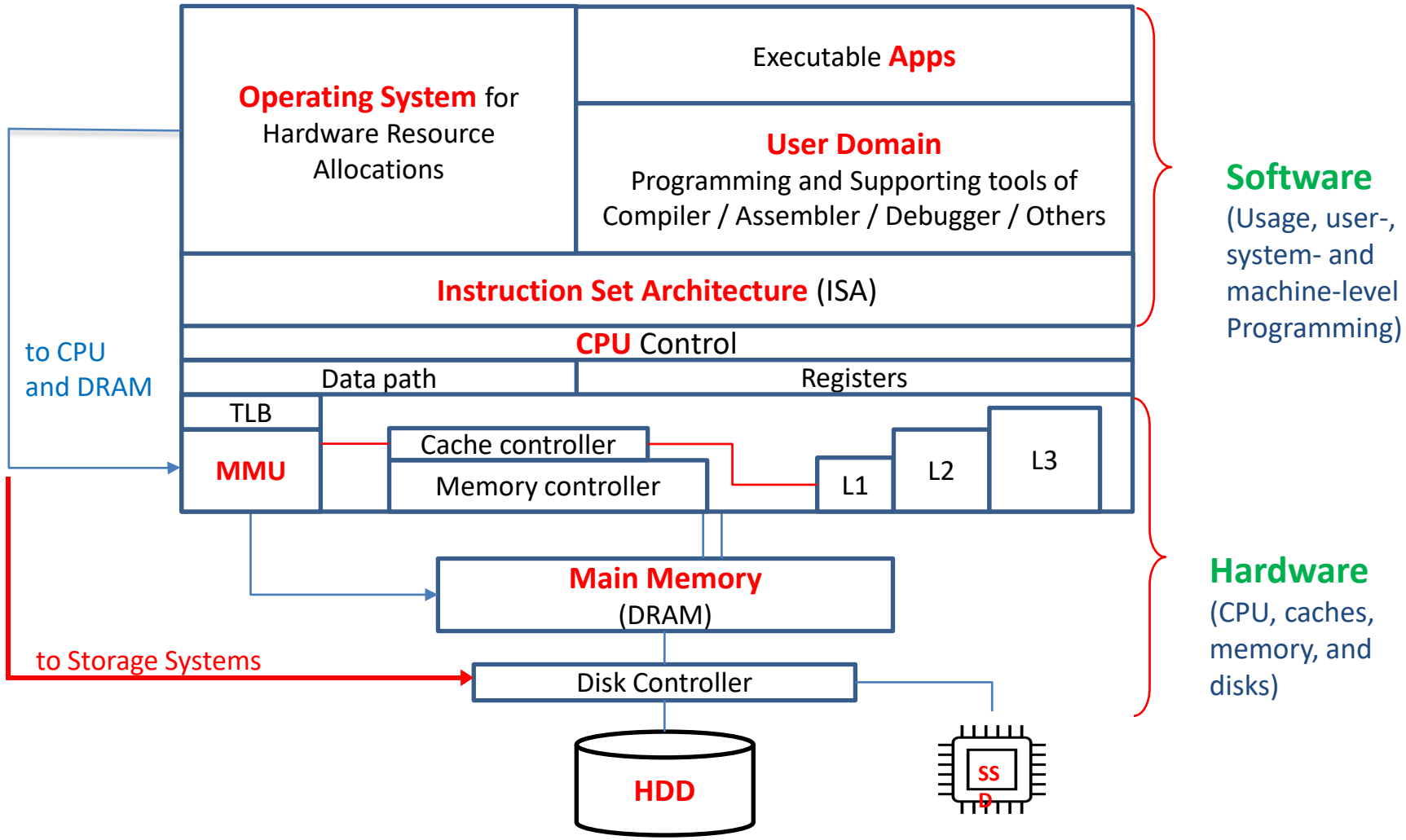
Xiaodong Zhang

Review: Where is CSE 3421 in Computing Ecosystem?



- We cover three spaces:
  - Virtual space: Assembly code and data stored in disks
  - Physical memory space: CPU, pipeline, cache, and memory system
  - Storage space: virtual memory, page table, file systems, hard disks and SSD

The Entire Hardware-Software Stack (where is your expertise?)



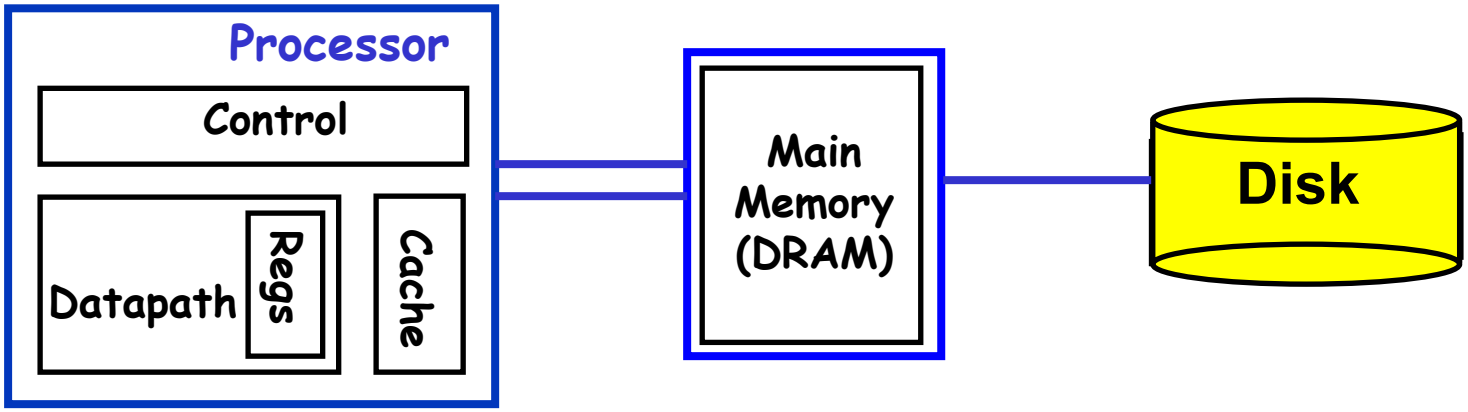
Why Virtual Memory?

1. Allow a program to be written without memory size constraints
2. Multiprogramming in memory so that context switches can occur
3. Relocation: Parts of the program can be placed at different locations in the memory instead of a big chunk (contiguous allocation)

Virtual Memory

I. Main Memory holds many programs (processes) running at same time

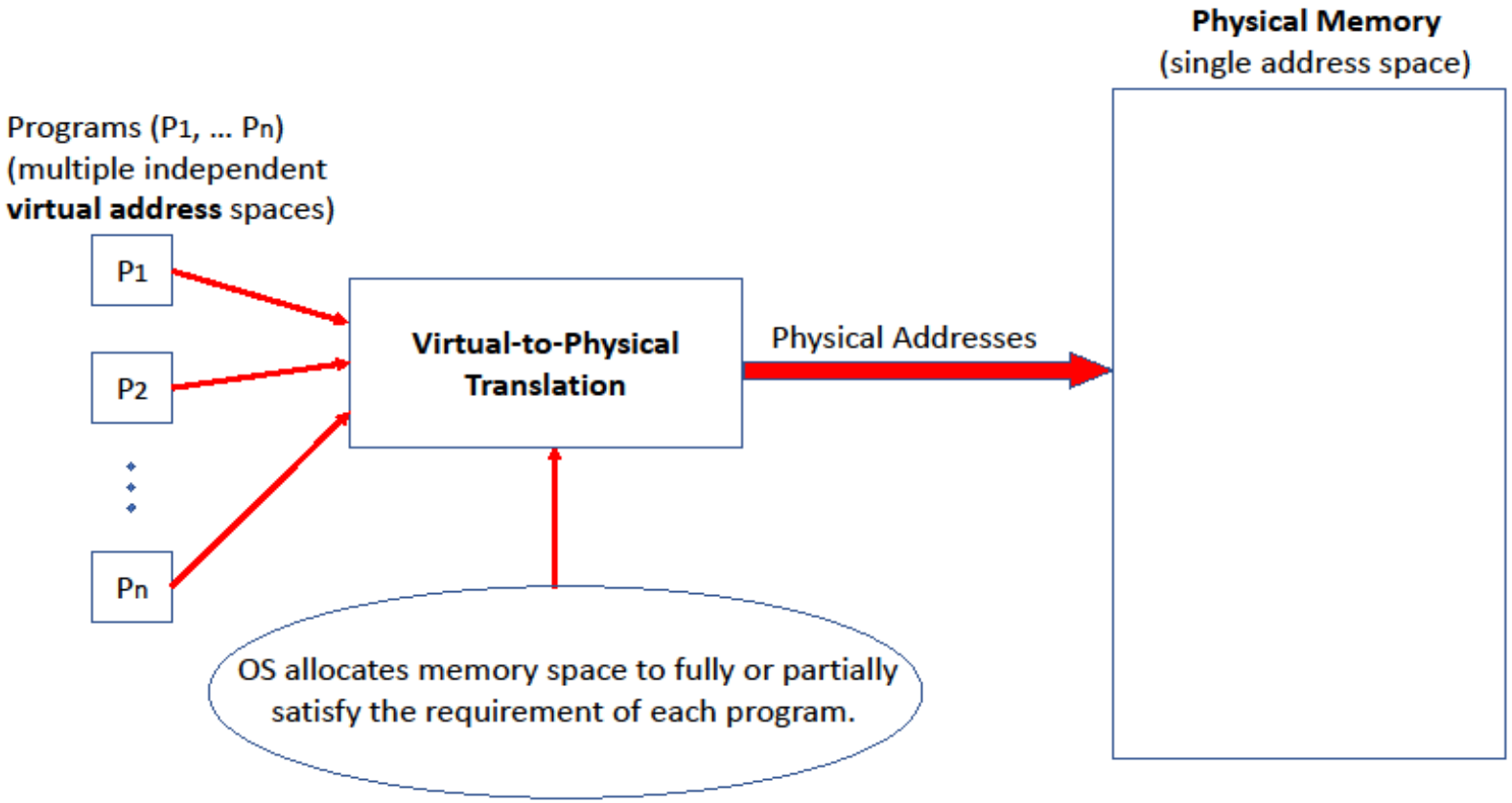
II. use Main Memory as a kind of “cache” for disk



Programming Space and its Execution Space

- A program is adaptively running in computer architecture in a most effective resource allocation way
- A compiler sets each program into its own virtual address space (stored in disks)
- VM translates the program's virtual address space to **physical memory addresses**, enforcing **protection** of a program's address space from other programs.
- VM allows each user's program to exceed the size of physical memory.
- The unit of in VM is a page, a miss in VM is page fault

A Big Picture of Virtual Memory for multiprogramming



- Multiple programs want to run in a computer (competing in the same area)
- OS knows the program/data space and create a page table for each program (table assignment by reservation)
- Not all the data sets are moved to memory (most hungry guests first)
- Running the program (an usher leads the group to the table)

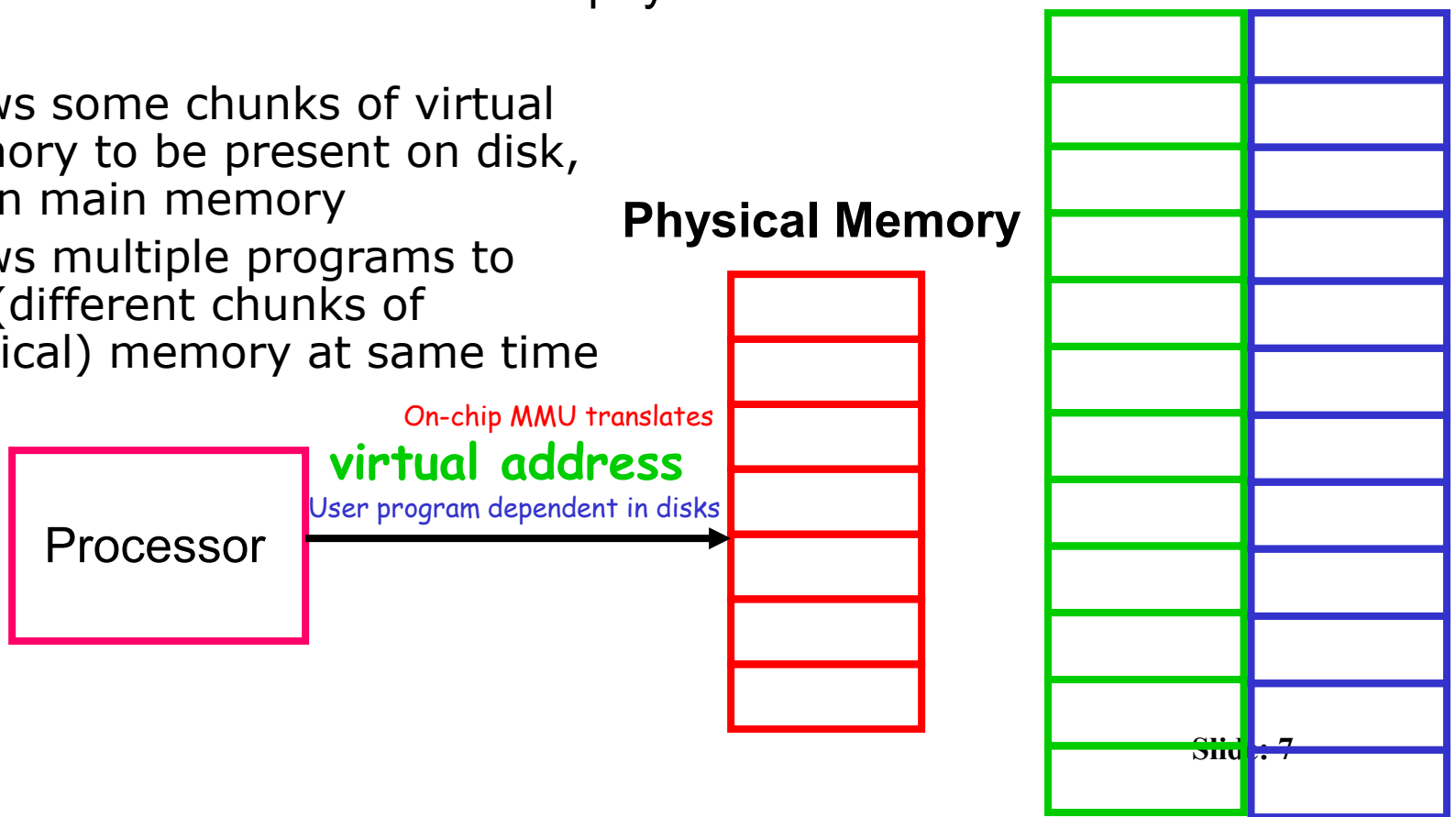
Virtual to Physical Memory mapping

Each process has its own private "virtual address space" (e.g., 2^{32} Bytes); CPU actually generates "virtual addresses"

Each computer has a "physical memory address space" (e.g., 128 MB DRAM); also called "real memory"

Address translation: from virtual to physical addresses

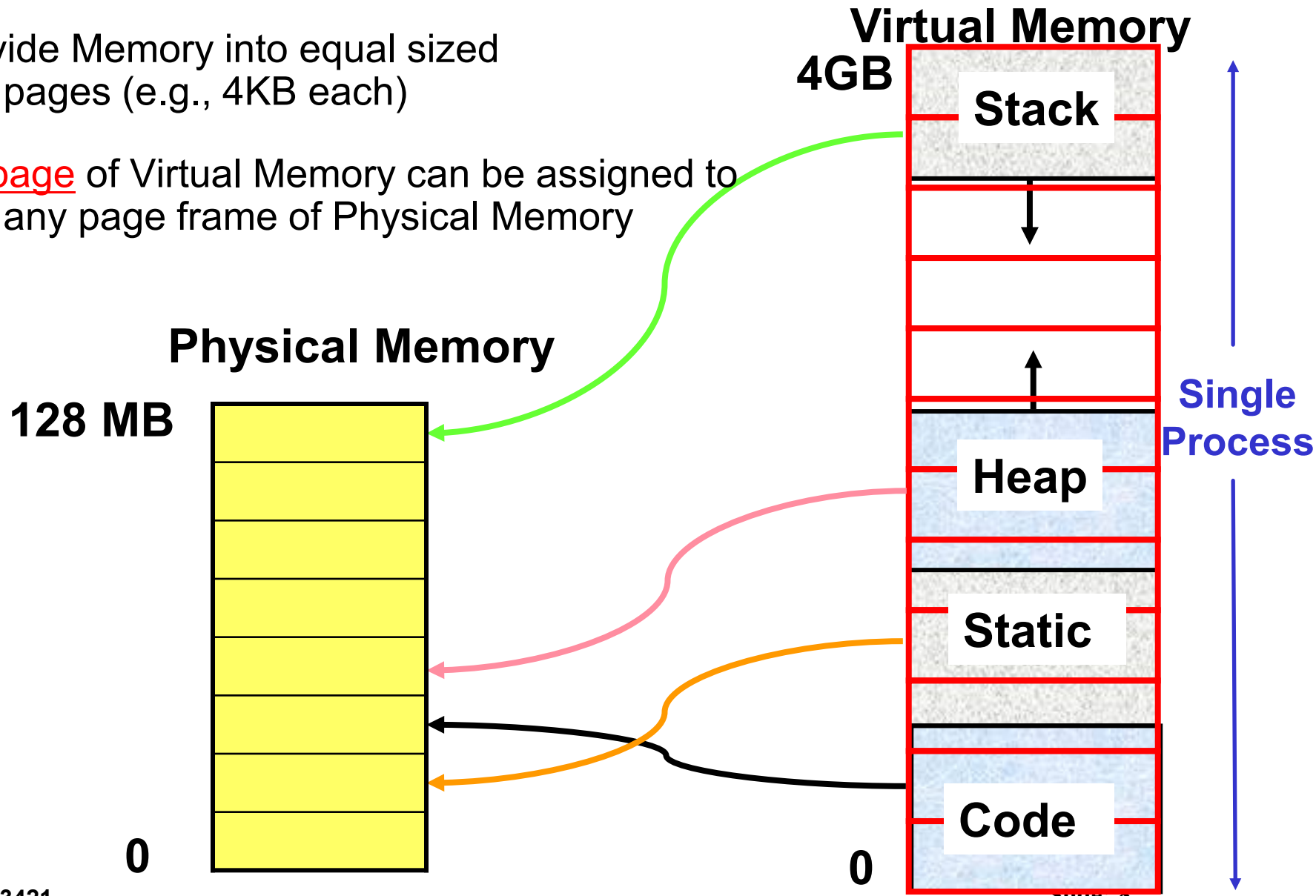
- Allows some chunks of virtual memory to be present on disk, not in main memory
- Allows multiple programs to use (different chunks of physical) memory at same time



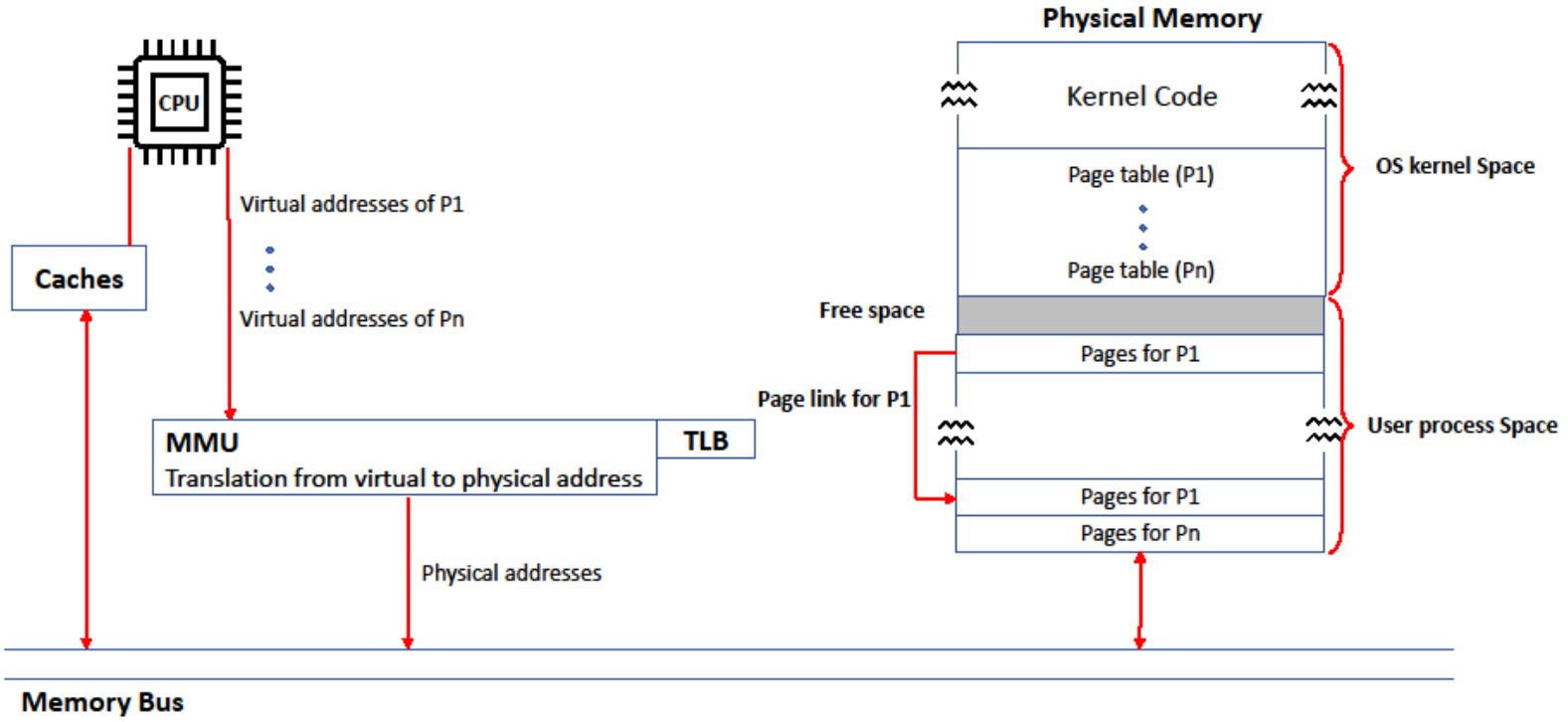
Mapping Virtual Memory to Physical Memory

Divide Memory into equal sized pages (e.g., 4KB each)

A page of Virtual Memory can be assigned to any page frame of Physical Memory



Virtual Space, MMU, Physical Space, Page Tables



- Process loader in OS reads executable code to set up virtual address space
- OS uses MMU (memory management unit) for address translations, which monitors usage of physical memory. MRU translations are cached in TLB
- Each running program has its own page table
- Pages can be allocated contiguously (Pn), or non-contiguously (P1)

How to Perform Address Translation? (memory address again)

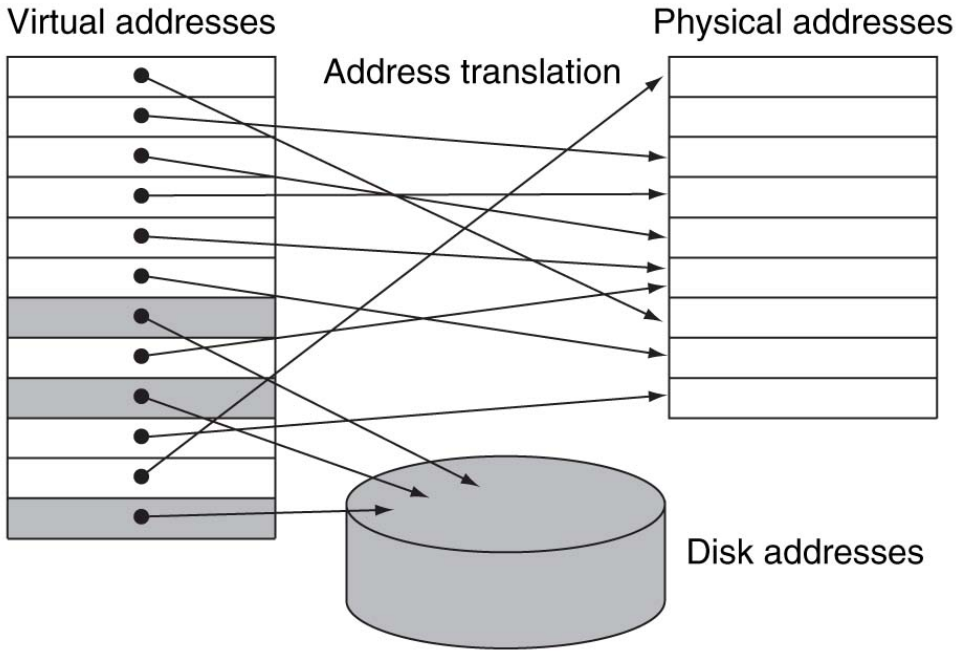
VM divides memory into equal sized pages

Address translation maps entire pages

virtual address

Virtual Page Number	Page Offset

Offsets within the pages remains the same



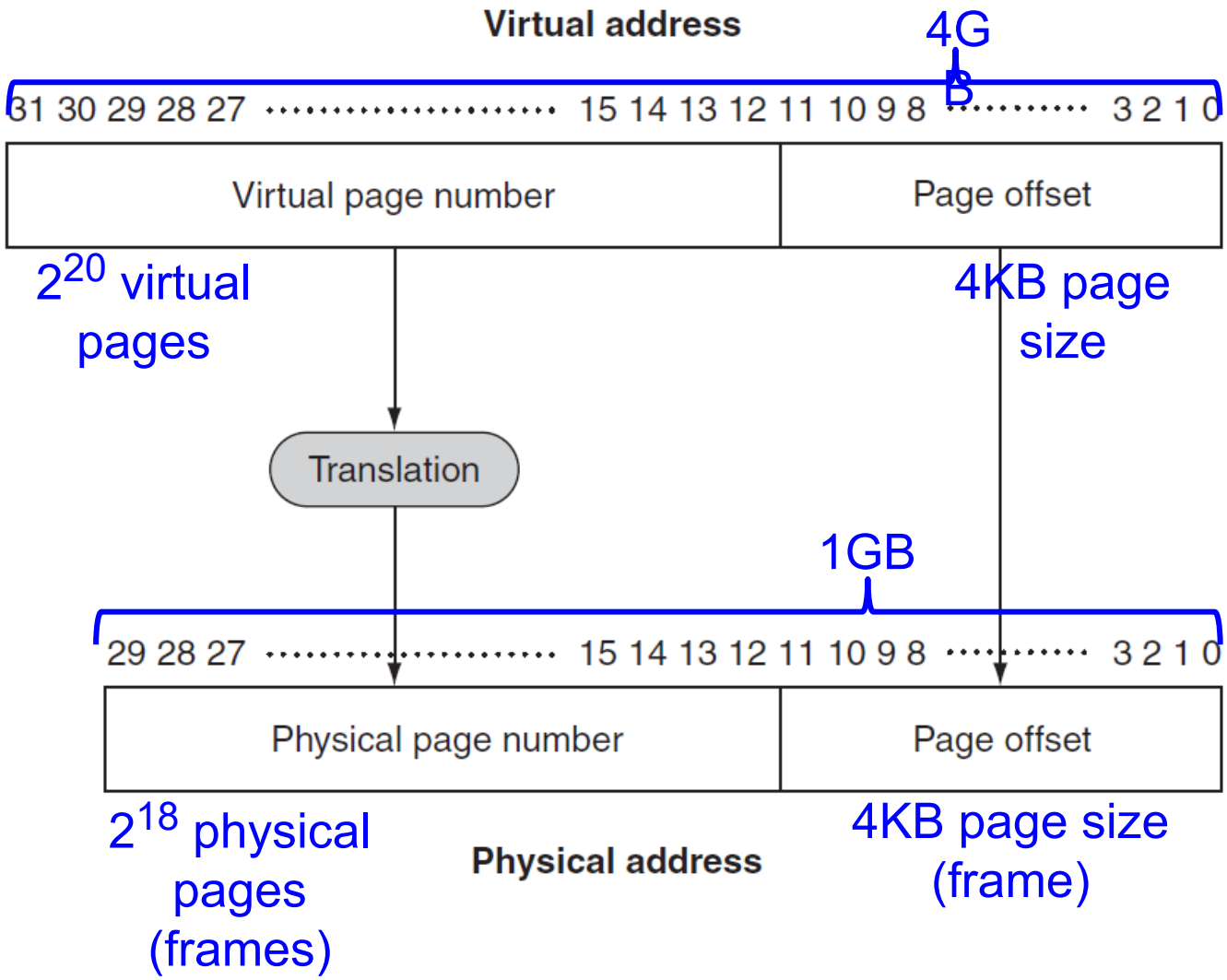
Page size is a power of two

Virtual address separates into:

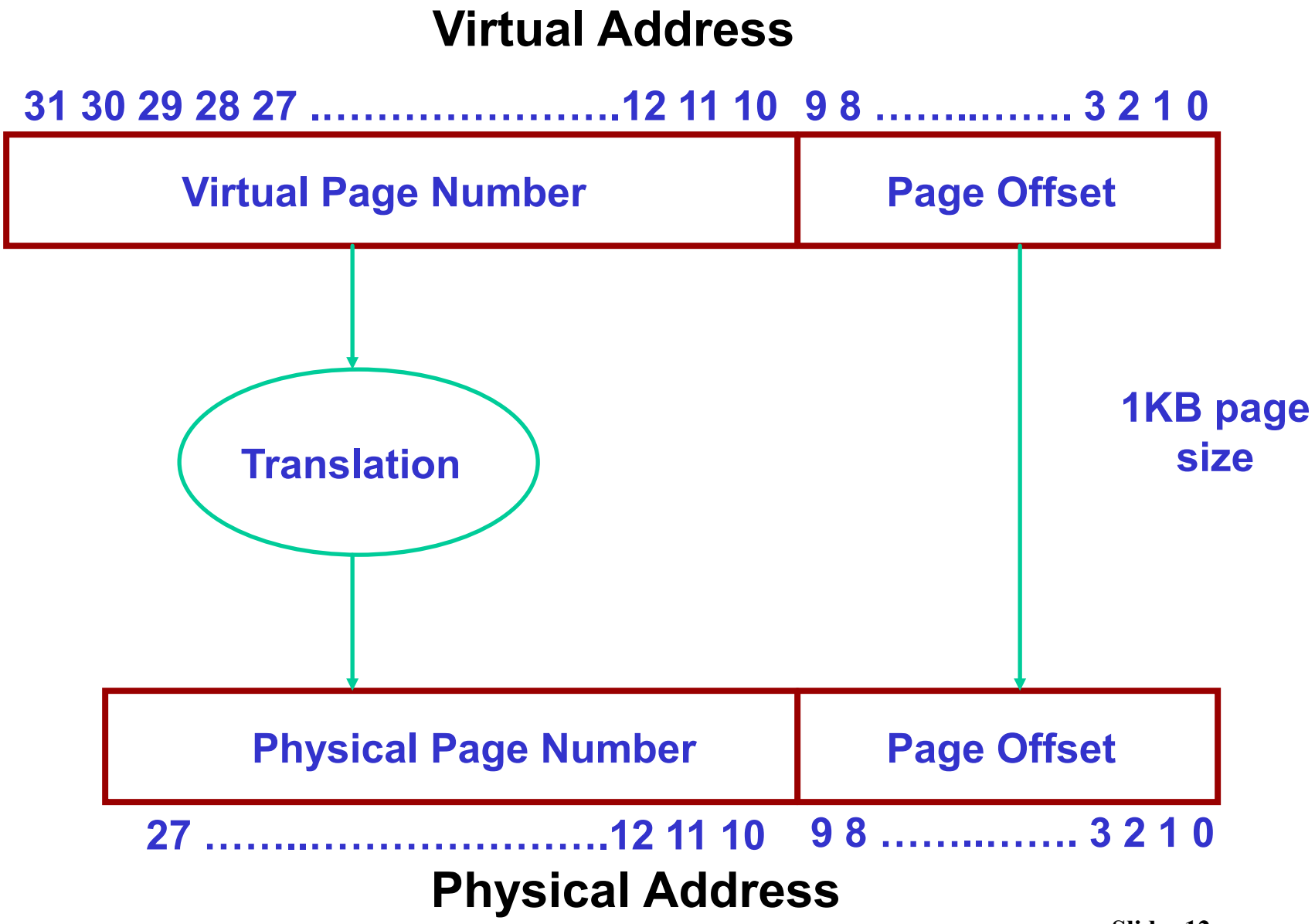
Index field and offset fields

Where are tags? (No need, due to **1 table for 1 process**)

Virtual to Physical Address Mapping



Mapping Virtual to Physical Address (1 K Page)



The Page Table

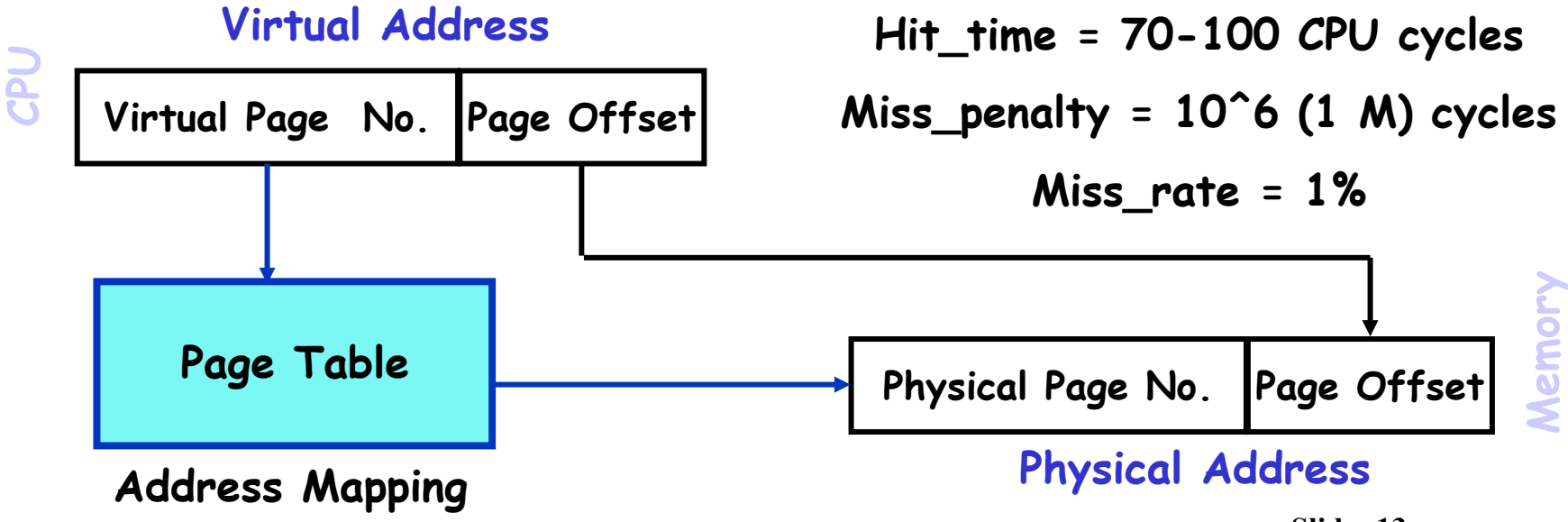
Want fully associative page placement

How to locate the physical page? A hardware called Memory Management Unit (MMU) handles this to create a page table

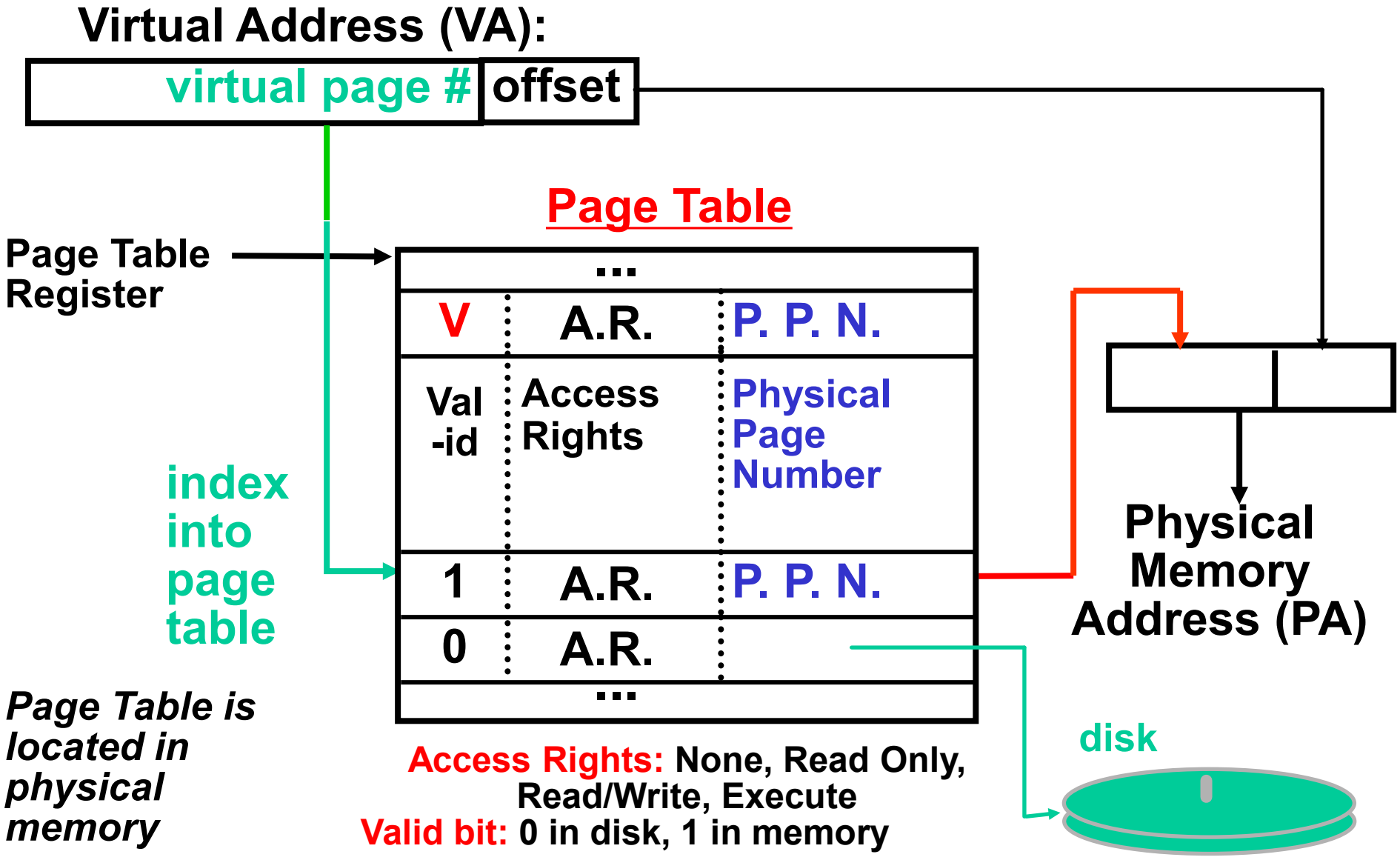
A **page table** is a data structure which contains the mapping of virtual pages to physical pages

There are several different ways, all up to the operating system, to keep and update this data

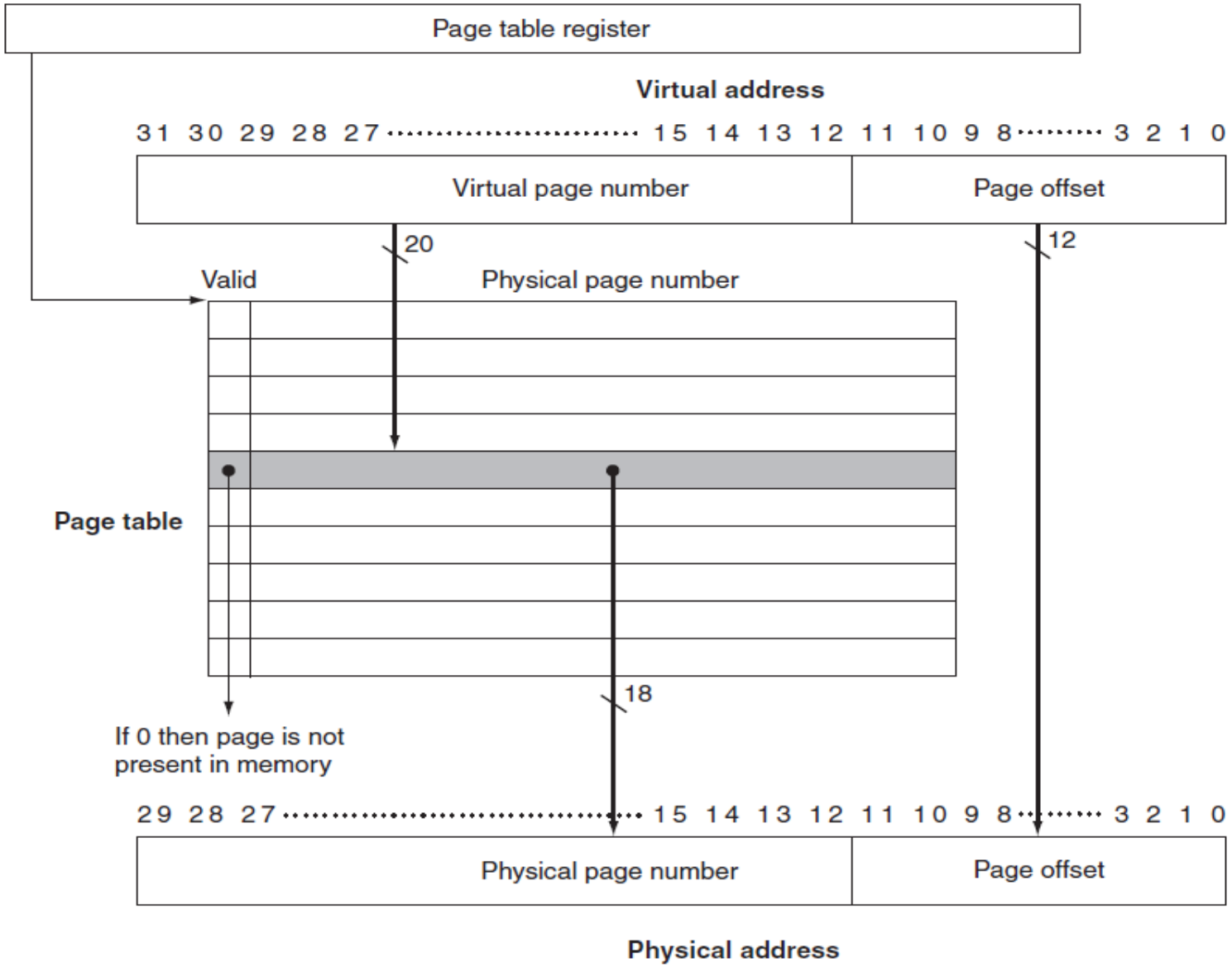
Each process running in the system has its own page table



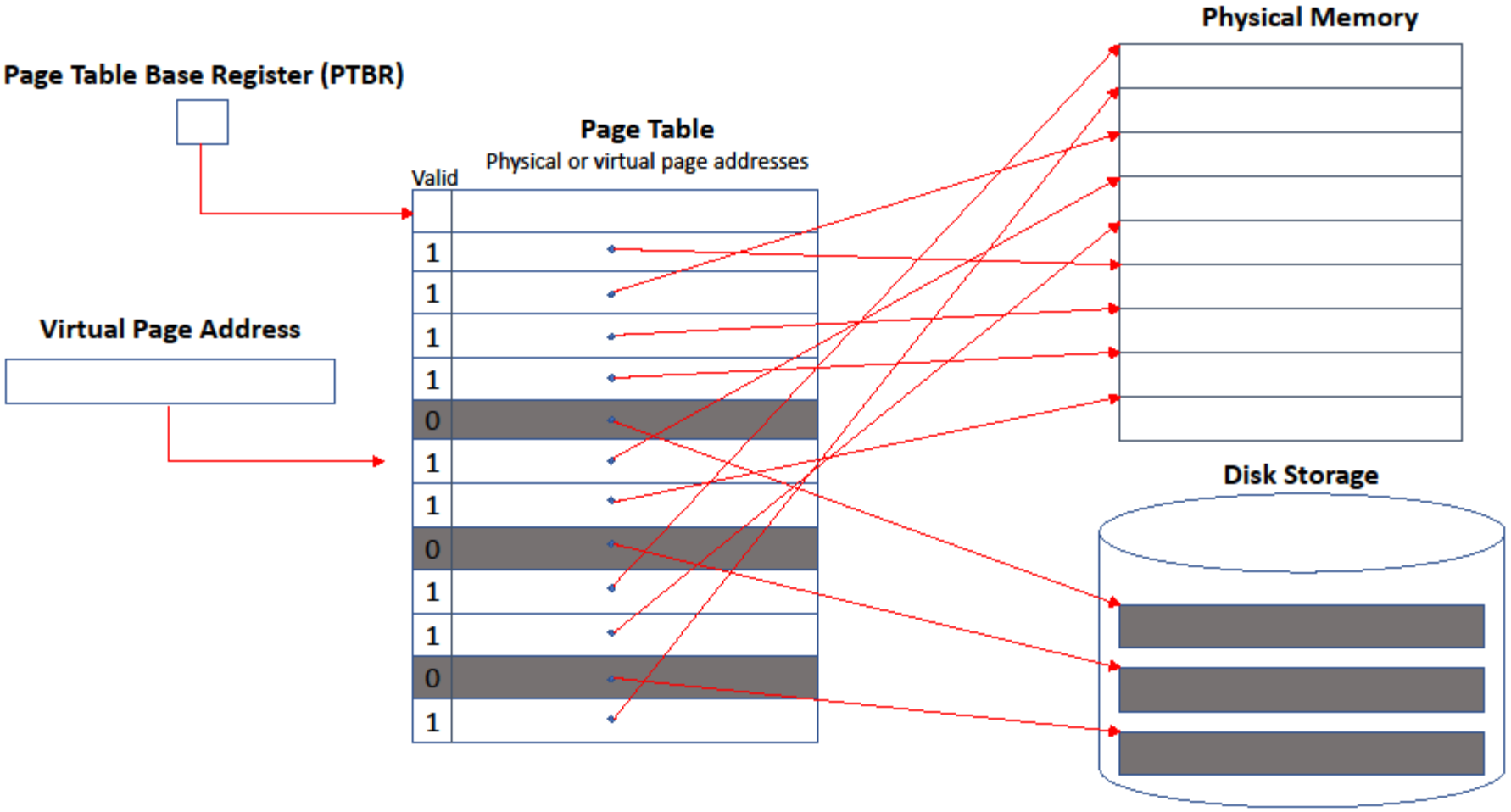
Inside Page Table



Each Process has a Page Table Register



Another view of Page Table for a Process



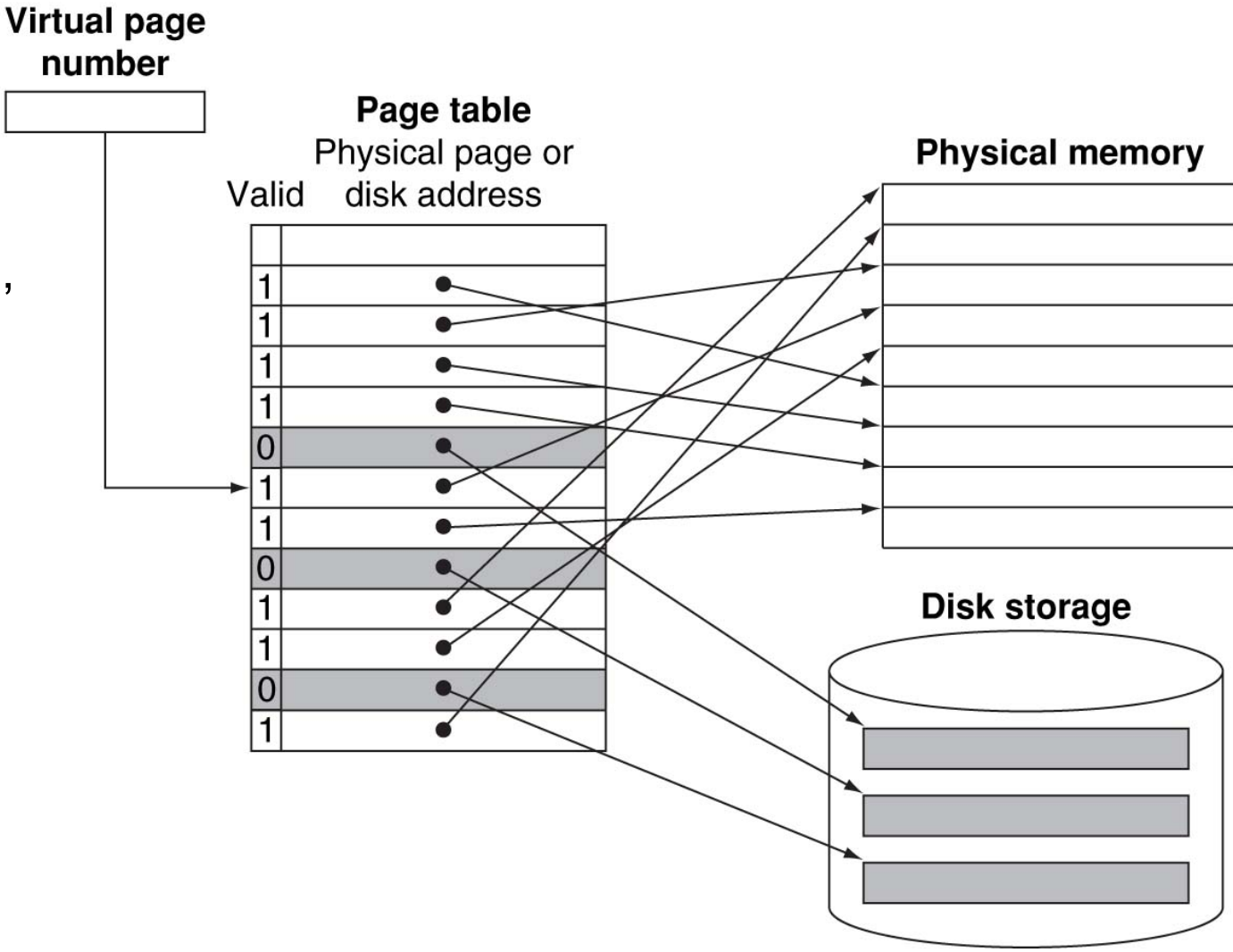
Page Table Register is a part of a process

- Page table base register (PTBR) in hardware points to the start of the page table for a process
  - In a context switch, OS changes PTBR for another process
  - For TLB, each process is given address space number (ASN)
- The page table + program counter (PC) + registers (PTBR, ASNR) specify the state of a program (process). (A person's state: home address (PT), current status (PC) and ID (reserved registers))
- In multiprogramming environment, the state is saved for another program use CPU, restoration of the state enables the program to continue
- A process is active when it is in possession of CPU, otherwise it is inactive
- OS activates a process by loading the process's state

Valid bit determines where the page is (disk (0) or memory (1))

OS creates a space on disk for all the pages of the process, called **swap space**

OS also creates a data structure to record and monitor the activities of page accesses, which will be used by management, such as LRU or LIRS



Handling Page Faults

- A page fault is like a cache miss
  - Must find it in lower level of hierarchy
- If valid bit is zero, the Physical Page Number points to a page on disk
- When OS starts new process, it creates space (swapping space) on disk for all the pages of the process, sets all valid bits in page table to zero
- Physical Page Numbers to point to disk called Demand Paging - pages of the process are loaded from disk only as needed

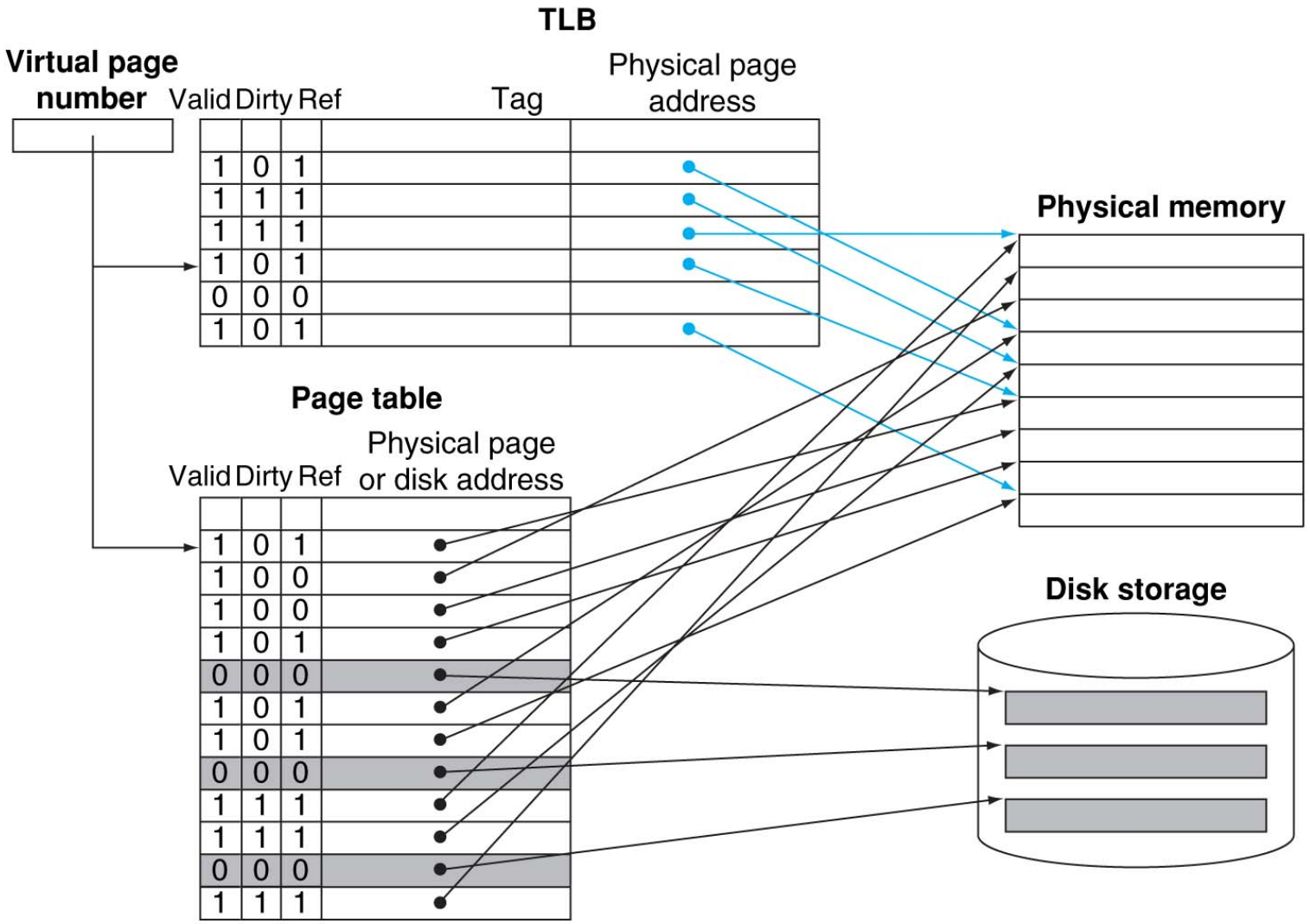
Comparing the 2 hierarchies

Cache	Virtual Memory
Block or Line	Page
Miss	Page Fault
Block Size: 16-64Bytes	Page Size: 4K-16KBytes
Placement: Direct Mapped, N-way Set Associative (multicolumn)	Fully Associative (OS control)
Replacement: approx. LRU of Random	LRU, LIRS, Clock Clock-pro
Write Thru or Write Back	Write Back
How to Manage: Hardware	Hardware + Software (Operating System)

How to Translate (map) Fast?

- Problem: Virtual Memory requires two memory accesses!
  - (1) to translate Virtual Address into Physical Address (page table lookup) - Page Table is in physical memory
  - (2) for the given memory address,to transfer the actual data (hopefully cache hit)
- Observation: since there is locality in pages of data, must be locality in virtual addresses of those pages!
- Why not create a cache of virtual to physical address translations to make translation fast? (smaller is faster)
- For historical reasons, such a “page table cache” is called a Translation Lookaside Buffer, or TLB
- Address Space Number is the process space ID for TLB to cache the page table entries of multiple processes

TLB (Translation Lookaside Table) and Page Table



Typical TLB Format

Virtual Page Number	Physical Page Number	Valid	Ref	Dirty	Access Rights
"tag"	"address"				

Since TLB only stores a small part of page table, virtual page # is a unique "tag" for each lookup

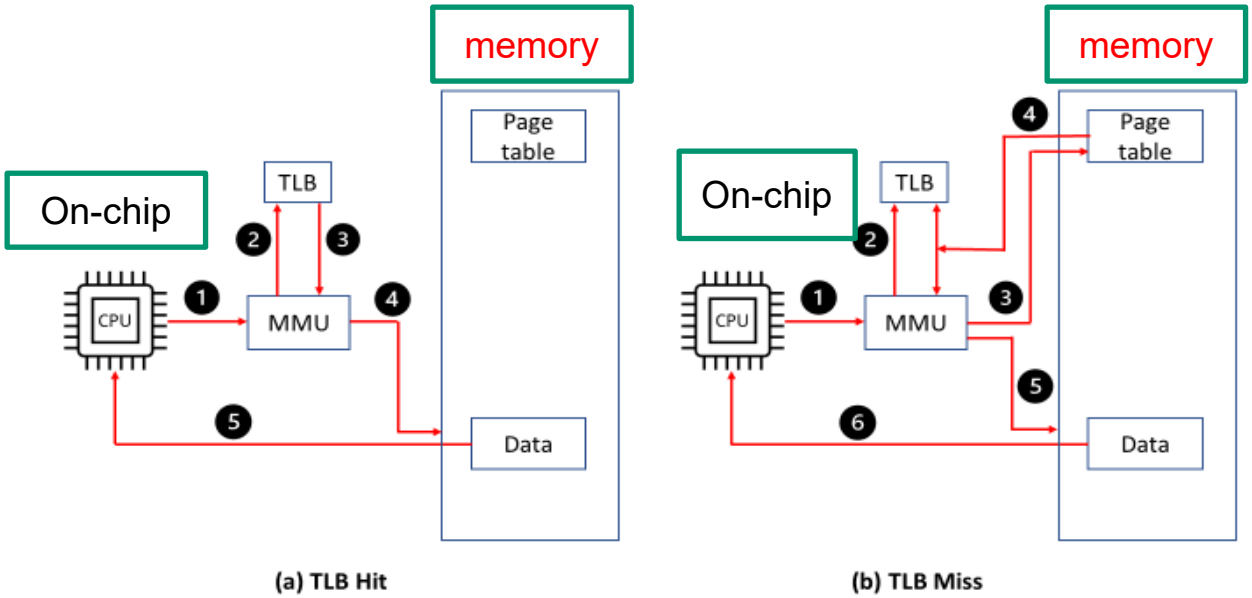
Dirty: since use write back, need to know whether or not to write page to disk when replaced

Ref: Used to calculate LRU on replacement

Reference bit - set when page accessed; OS periodically sorts and moves the referenced pages to the top & resets all Ref bits

Valid bit=1 in memory, =0 in disk. TLB access time is L1 latency

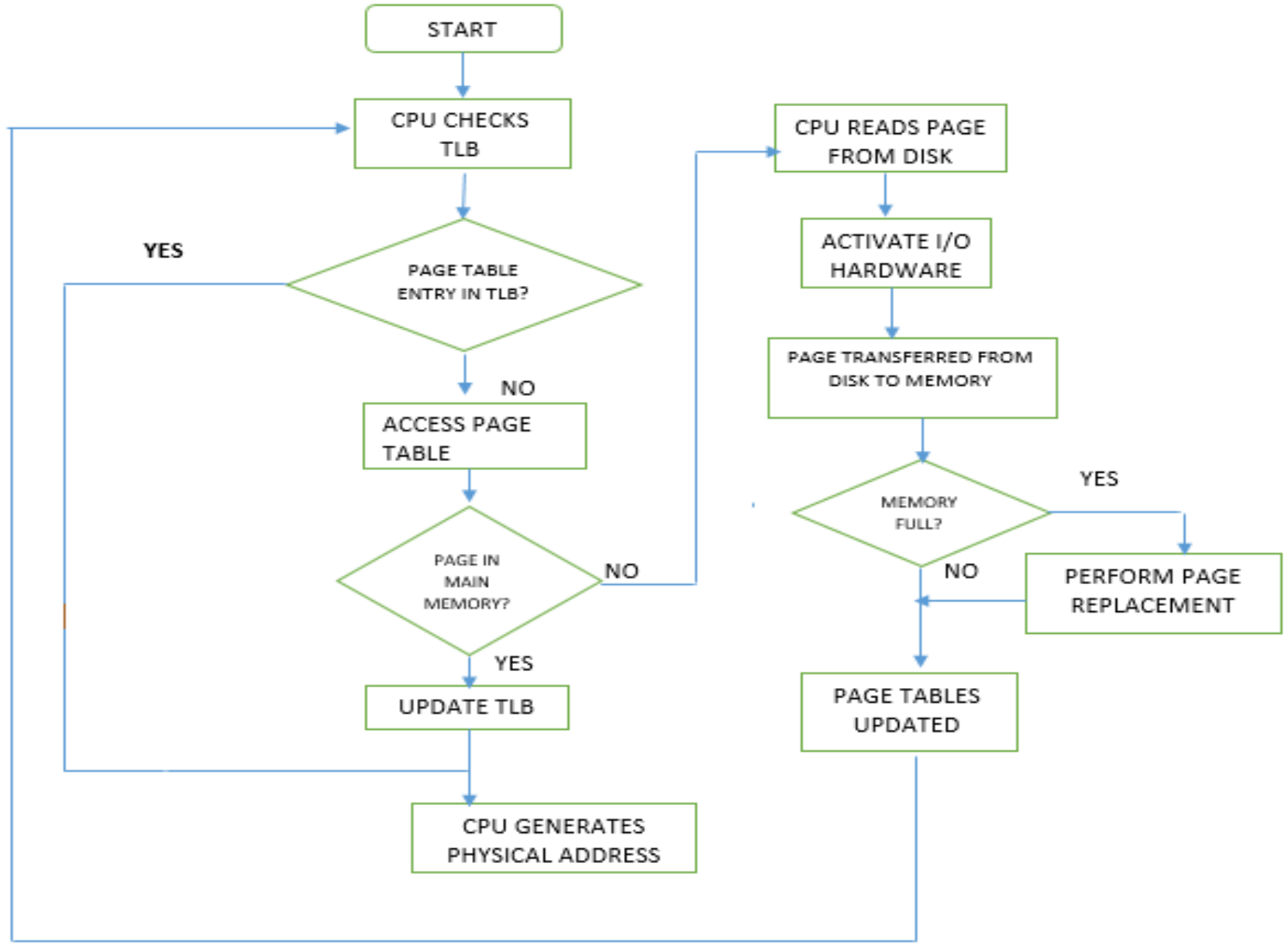
TLB Hit vs TLB Miss



- TLB hit: 1. CPU issues a virtual address, (2) MMU checks TLB, (3) TLB hit, (4) MMU accesses targeted memory page, and (5) page is delivered to CPU

- TLB miss: 1. CPU issues a virtual address, (2) MMU checks TLB, (3) TLB miss and MMU accesses page table, (4) physical address is sent to MMU, TLB is updated, (5) MMU accesses targeted memory page, and (6) page is delivered to CPU

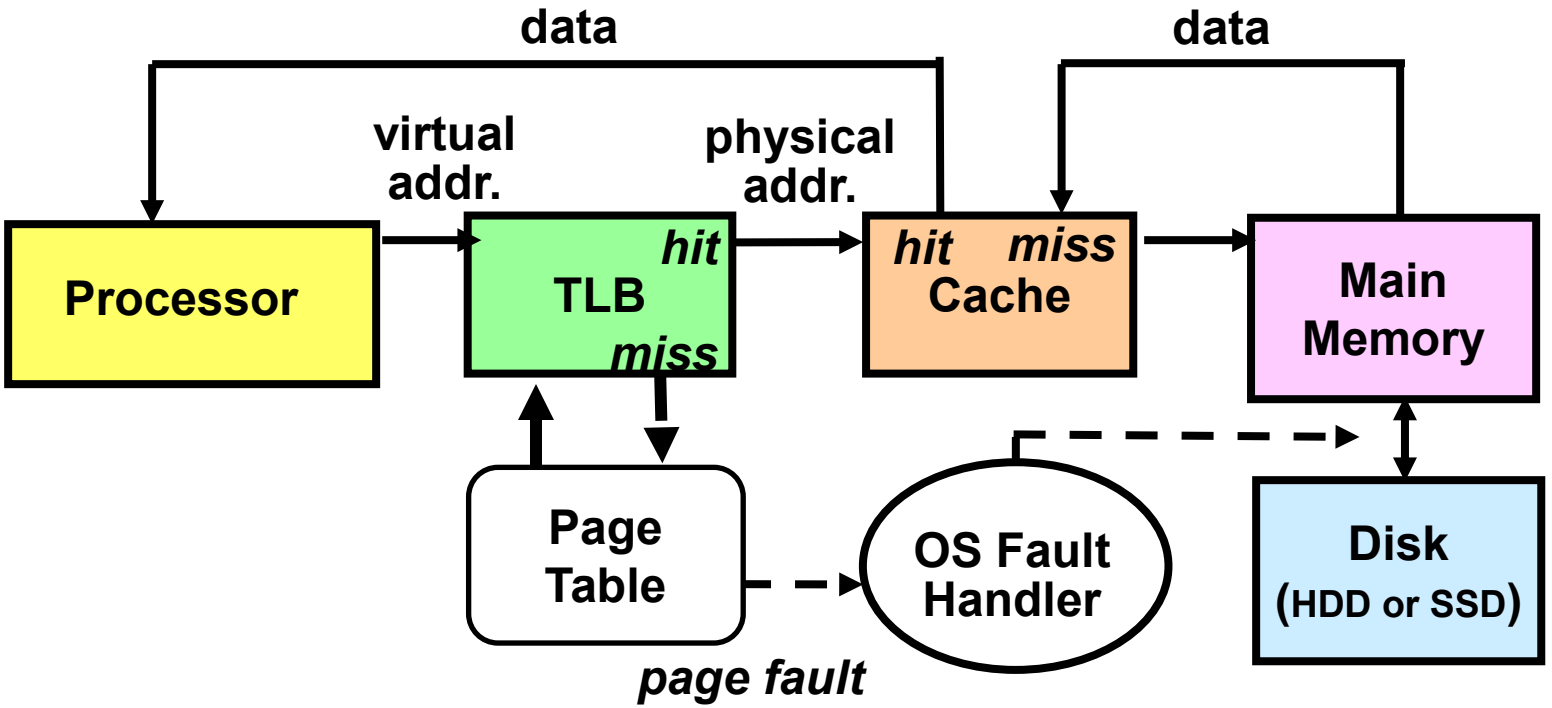
Data flow involving TLB, page table, cache …



Another view of Translation Look-Aside Buffers

TLB is usually small, typically 32-512 entries

Like any other cache, the TLB can be fully associative, set associative, or direct mapped

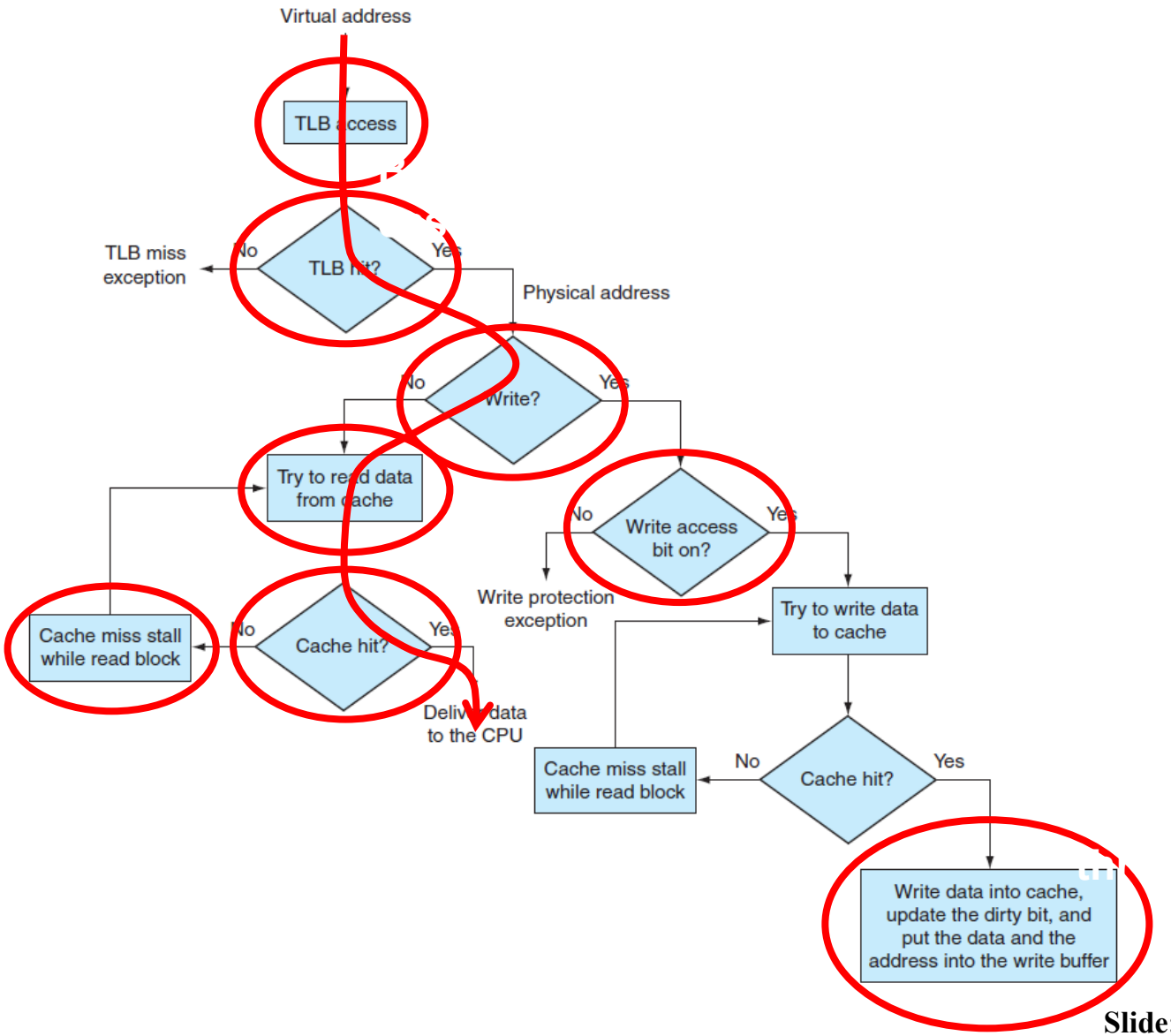


Accessing data with TLB

1. TLB hit
   - Using the memory address (v=1) to seek in cache and cache hit or
   - Cache miss, seek the page in the memory
   - Page is in disk (v=0), start a page fault operation:
     - find a victim page in memory, swap the page to memory from disk, update the page table, supply address to CPU to search cache or memory

2. TLB miss (search the page table in memory)
   - memory address is available (v=1), supply it to CPU in 1, or
   - Page is in disk (v=0), start a page fault operation:
     - find a victim page in memory, swap the page to memory from disk, update the page table, supply address to CPU in 1

Read and Cache Write-Through



Worse case in page accesses

- In the **worst case**, a reference can miss all: the TLB, the page table, and the cache.
- Consider all the seven combinations of the three events. State for each whether it can actually occur and under what circumstances.
- The three-hit case is not in the table, which is the best case.

TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	Possible, although the page table is never really checked if TLB hits.
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache.
miss	hit	miss	TLB misses, but entry found in page table; after retry, data misses in cache.
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
hit	miss	miss	Impossible: cannot have a translation in TLB if page is not present in memory.
hit	miss	hit	Impossible: cannot have a translation in TLB if page is not present in memory.
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory.

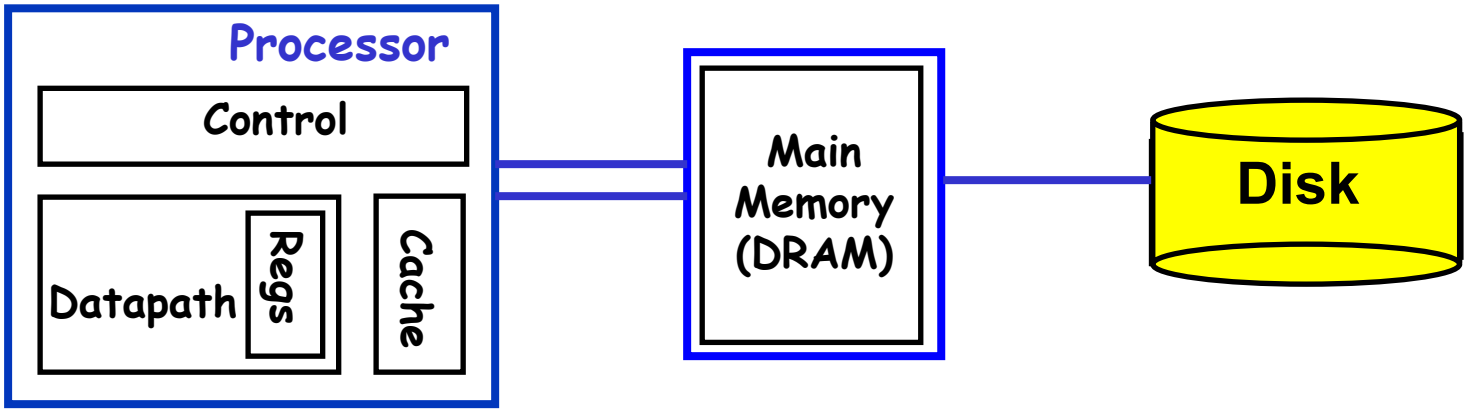
Why Virtual Memory?

1. Allow a program to be written without memory size constraints
2. Multiprogramming in memory so that context switches can occur
3. Relocation: Parts of the program can be placed at different locations in the memory instead of a big chunk (contiguous allocation)

Virtual Memory

I. Main Memory holds many programs (processes) running at same time

II. use Main Memory as a kind of “cache” for disk



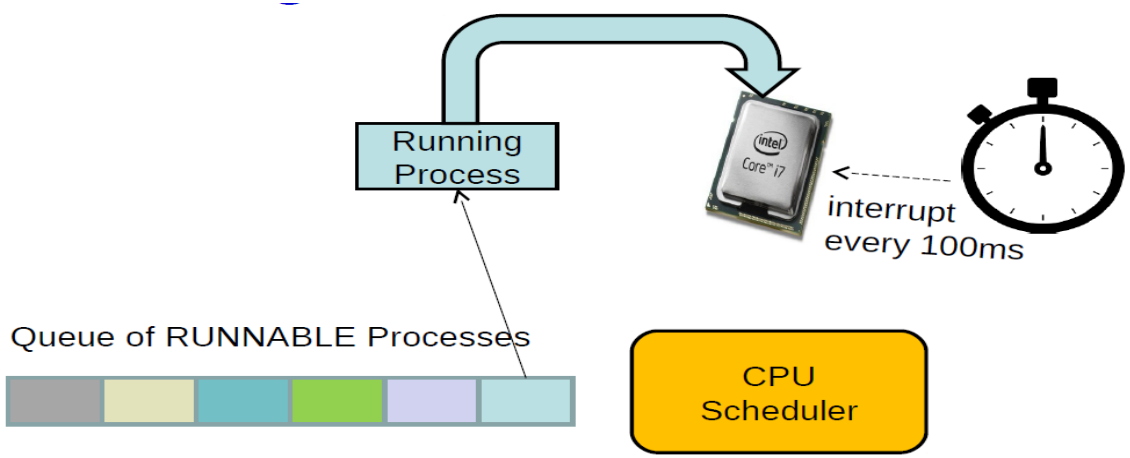
Multiprogramming in the views of users and OS

Program ≠ Process

Program	Process
code + static and global data	Dynamic instantiation of code + data + heap + stack + process state
One program can create several processes	A process is unique isolated entity

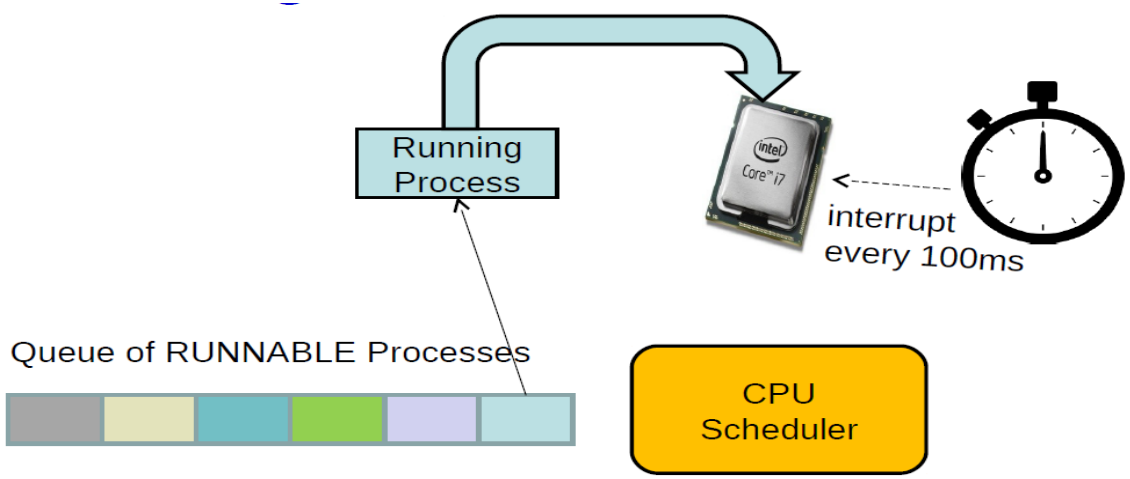
- A program is in virtual space
  - Programs are created by users/tools: Java, assembly, executable codes
  - Each program consists of multiple procedures and functions
- A process is running in the physical space
  - Its execution and space allocation and others are managed by OS with hardware support
  - Each process is assigned by a unique ID called PID

Each PID is used during the OS Scheduling



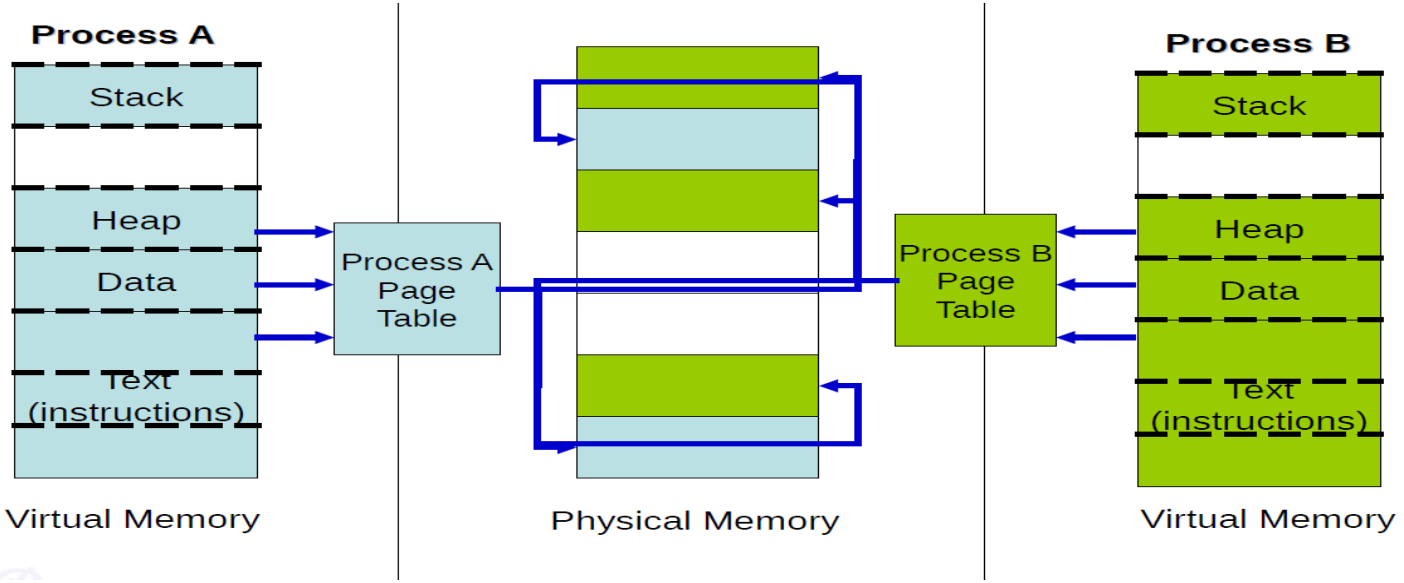
- Round robin scheduling is commonly used
  - A fixed time stamp is assigned to each process in each round
  - CPU is periodically interrupted to execute one process at a time in queue
- Priority scheduling: first execute labeled processes (e.g. sys. kernels)
- Opportunity scheduling: when a process waits for I/O, the idle CPU cycles are used for another process
- Context switch happens at a high cost for turning to a different process

"Equality vs Equity" in System Resource Allocation



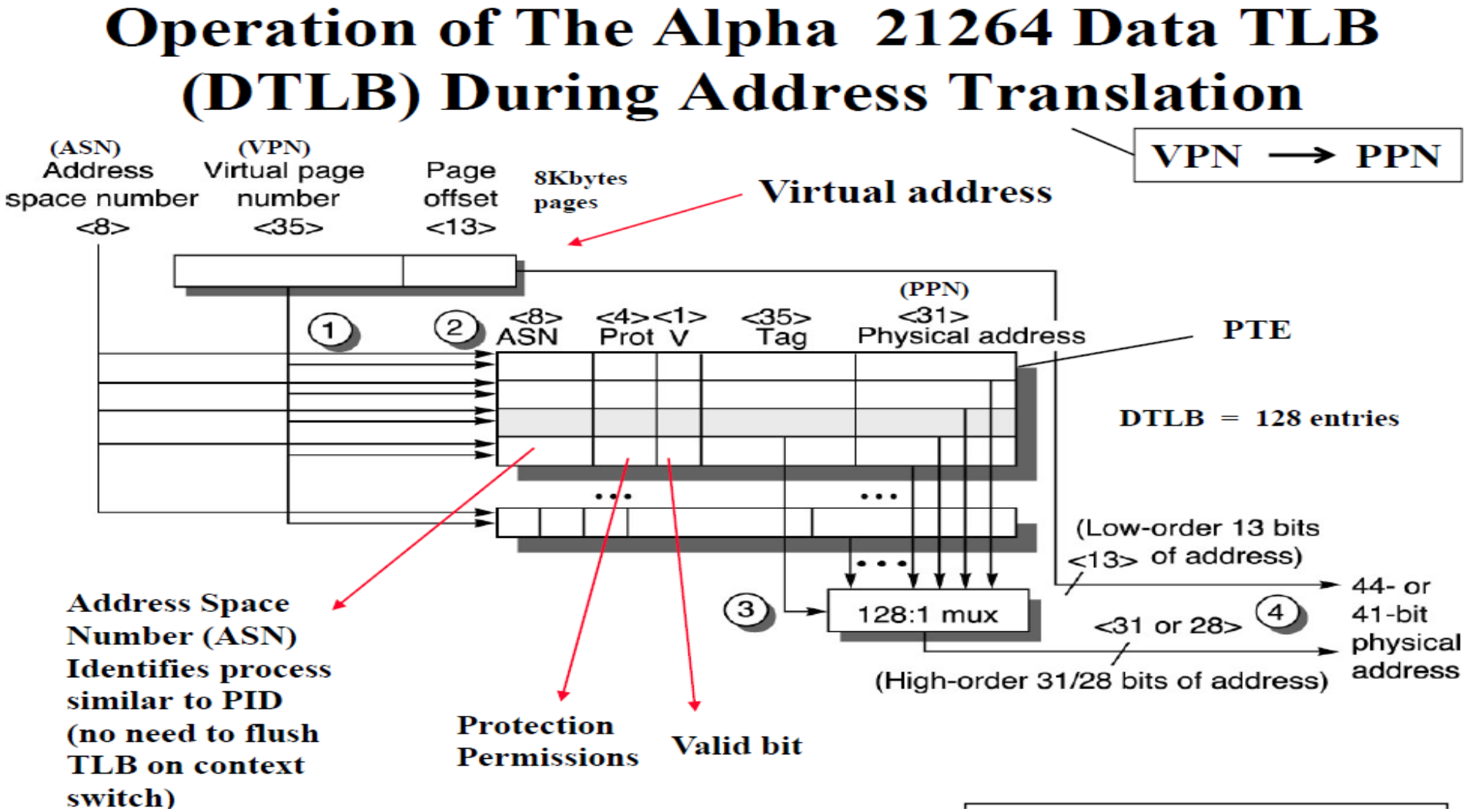
- Equality: treating each process equally
  - A fixed time stamp is assigned to each process in round robin scheduling
  - Each round process time and context switch overhead are the same
- Equity: addressing the unique property of each process
  - Priority scheduling: ranking processes based on their weights in systems
  - Opportunity scheduling: giving flexibility to scheduling to best utilize available cycles

Multiprogramming by Architecture Support



- Each process is given a Register to point the Page table
  - Page table base register (PTBR)
  - OS assigns different memory pages (frames) to different processes
  - Process A and Process B do not have shared frames
- A process dependent ID is address space number or ID
  - ASN or ASID : space ownership of a unique process

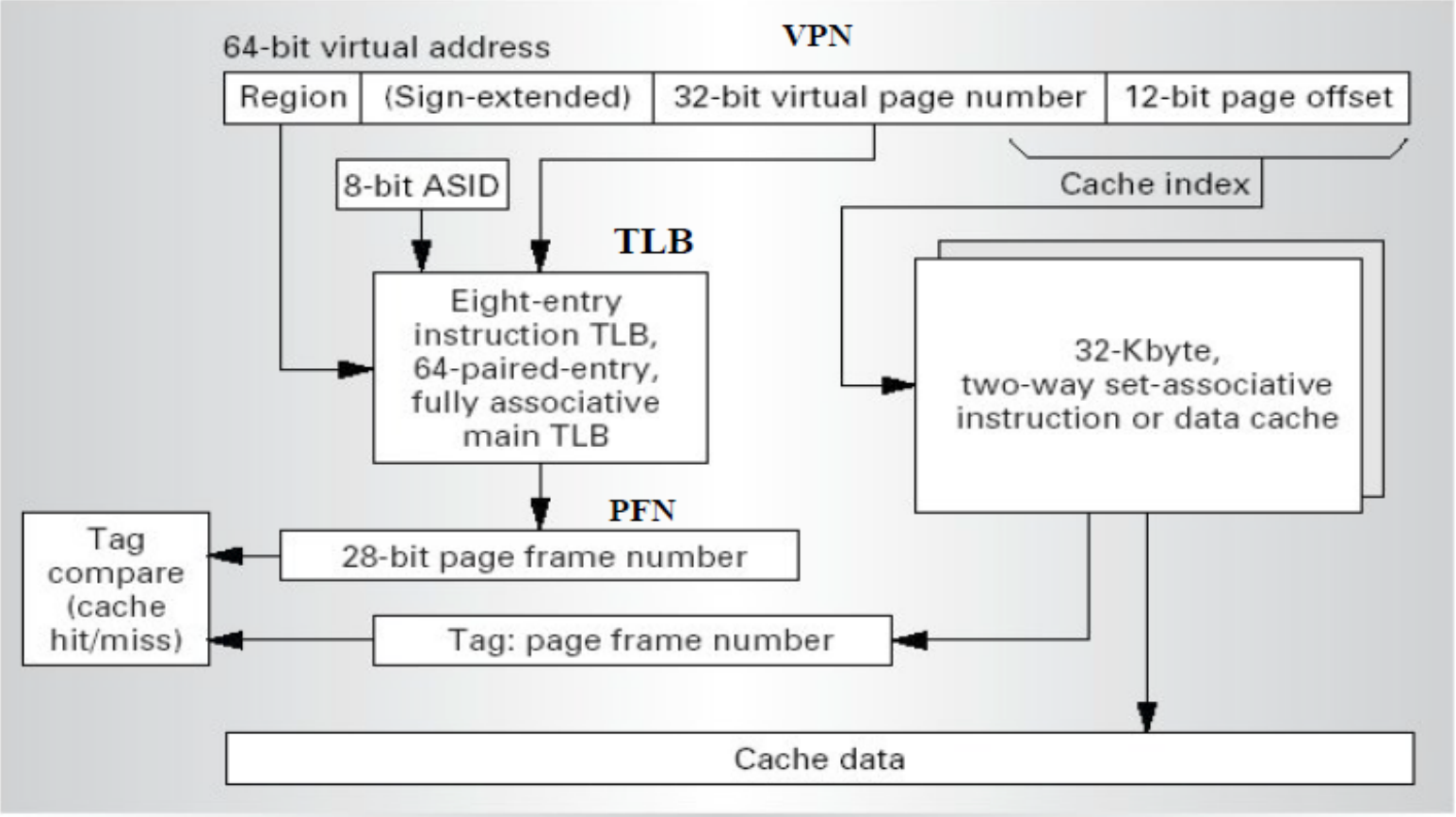
ASN or ASID is used in TLB



- Alpha 21264 is a RISC processor by DEC in 1996

ASN or ASID is used in TLB (continued)

MIPS R10000 Address Translation Mechanism



- MIPS 10000 is a RISC processor by MIPS Technologies in 1996

Where are data storages in the memory hierarchy ?
(1) Registers

- **Registers**
  - Also called processor registers or CPU registers
  - On chip and is closest to ALU, **32 - 64 bits** each
  - **Compiler** directly uses available registers during the code generation
  - Assembly programmers directly use registers
  - Registers can store base memory address, but **no mapping** between Registers and Memory.
  - Registers are not under the management of OS

  - **Registers is in the virtual space**

Where are data storages in the memory hierarchy ? (2) on-chip caches and off-chip DRAM

- Cache and memory are in physical space of execution
- Cache block size: **4 Bytes – 32+ Bytes**
- Memory page size: **4KB - 8KB**
- Virtual pages are allocated in physical memory pages by **OS**
- Virtual and physical pages are indexed in **page table**
- Both memory and cache **share the same memory address**
- For a given memory address, it means
  - a Byte, a starting address of a word, of a page, of a segment (multiple pages) in memory, which memory bank the page in
  - a block of direct-mapped, set-associative, fully-associative cache
  - If cache is partitioned by pages, which cache region it is

Users are always in the virtual space

- High-level language program (in C)

```
swap (int v[], int k)
(int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
)
```

- Assembly language program (for MIPS)

```
swap:   sll     $2, $5, 2
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
        sw      $16, 0($2)
        sw      $15, 4($2)
        jr      $31
```

- Machine (object, binary) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
 . . .
```