High Speed Memory

This talk is largely taken from:

https://www.gatevidyalay.com/paging-memory-management-operating-system/

Overview

- **Why use paging**
- **Paging implementation**

Why use paging?

- **In the stone age, programs ran using physical memory address.**
- **There were reasons for this**
  - Computers were single process (ran programs one at a time)
  - Memory was generally static (stacks and heap memory were not part of the early versions of FORTRAN and COBOL), which allowed faster memory access.
  - There was no support for any other type of memory
- **This lead to many issues:**
  - Program memory could not be relocated, so Multiprocessing was difficult because the addresses needed were often already used.
  - On the IBM 360 there was a base address register where to look for the memory.

How to fix these limitations

- **What we want to fix**
  - Allow multiple programs to be run in memory by allowing programs to be relocated in memory
  - Multiple programs that total more than the system memory can be run concurrently
  - Allow memory to kept in non-contiguous memory locations
  - Build translations of program memory into real addresses
  - Security so that a program cannot access the memory of another program.
  - Ability to run programs larger than the physical memory available
  - Allocate programs with the memory they need

Segmentation vs Paging

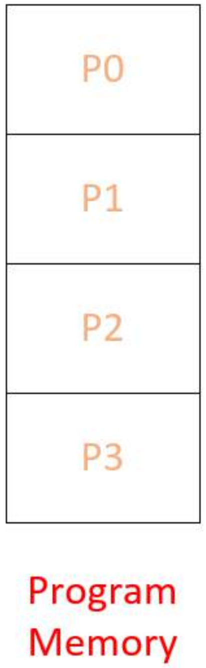
- **Segmentation**
  - Segmentation divides memory into blocks that the program can allocate.
  - It suffered from external fragmentation
  - Most famous was early x86 architecture, which used fixed 64k segments (because it had a 16 bit address). Arrays and other data and code were limited to 64k.
- **Paging**
  - Memory is mapped to fixed size pages.
  - All memory is seen as flat, and virtual addresses converted to real memory address.
  - Requires extra hardware and access time, but thought (known) to be worth it.

To Implement either paging or segmentation

- A base address in real memory is specified for the page or segment in memory.
- Program address is translated to memory address by calculating using the segment or page number to get the base address, and adding an offset.

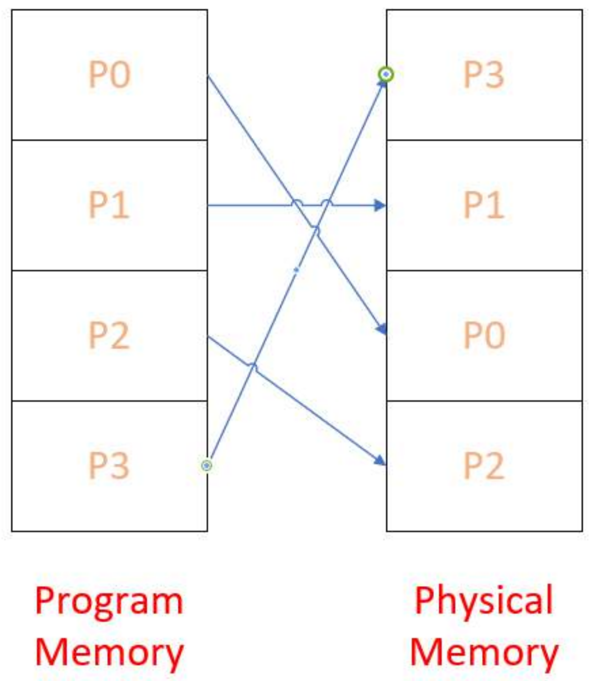
Paging implemented (page 1)

- **Memory is still flat, but is divided into pages. Memory is flat.**
  - Paging partitioning scheme



Paging implemented (page 3)

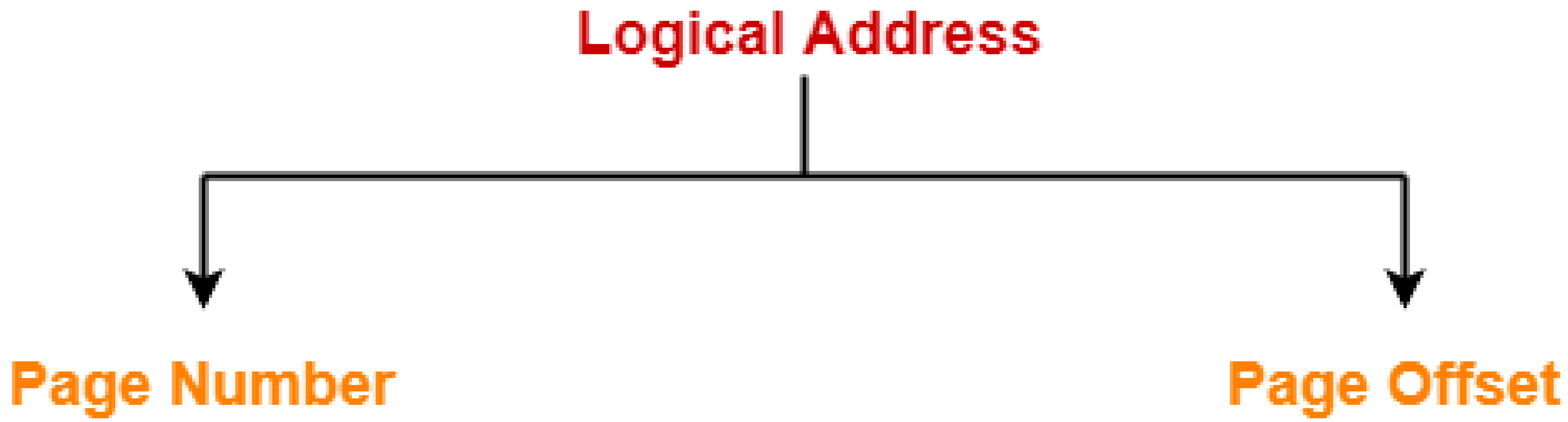
- **Example:Consider a process divided into 4 pages P0, P1, P2 and P3.**
- **In physical memory these pages are not contiguous**



Paging implemented (page 4)

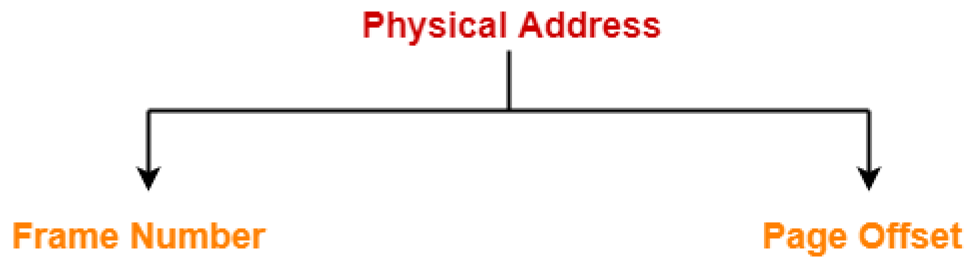
- **Translating logical address into physical address**
  - CPU always generates a logical address.
  - A physical address is needed to access the main memory.
- **Step 1**
  - CPU generates a logical address consisting of two parts-
    - Page Number
    - Page Offset
  - Page Number specifies the specific page of the process from which CPU wants to read the data.
  - Page Offset specifies the specific word on the page that CPU wants to read.

Logical Address

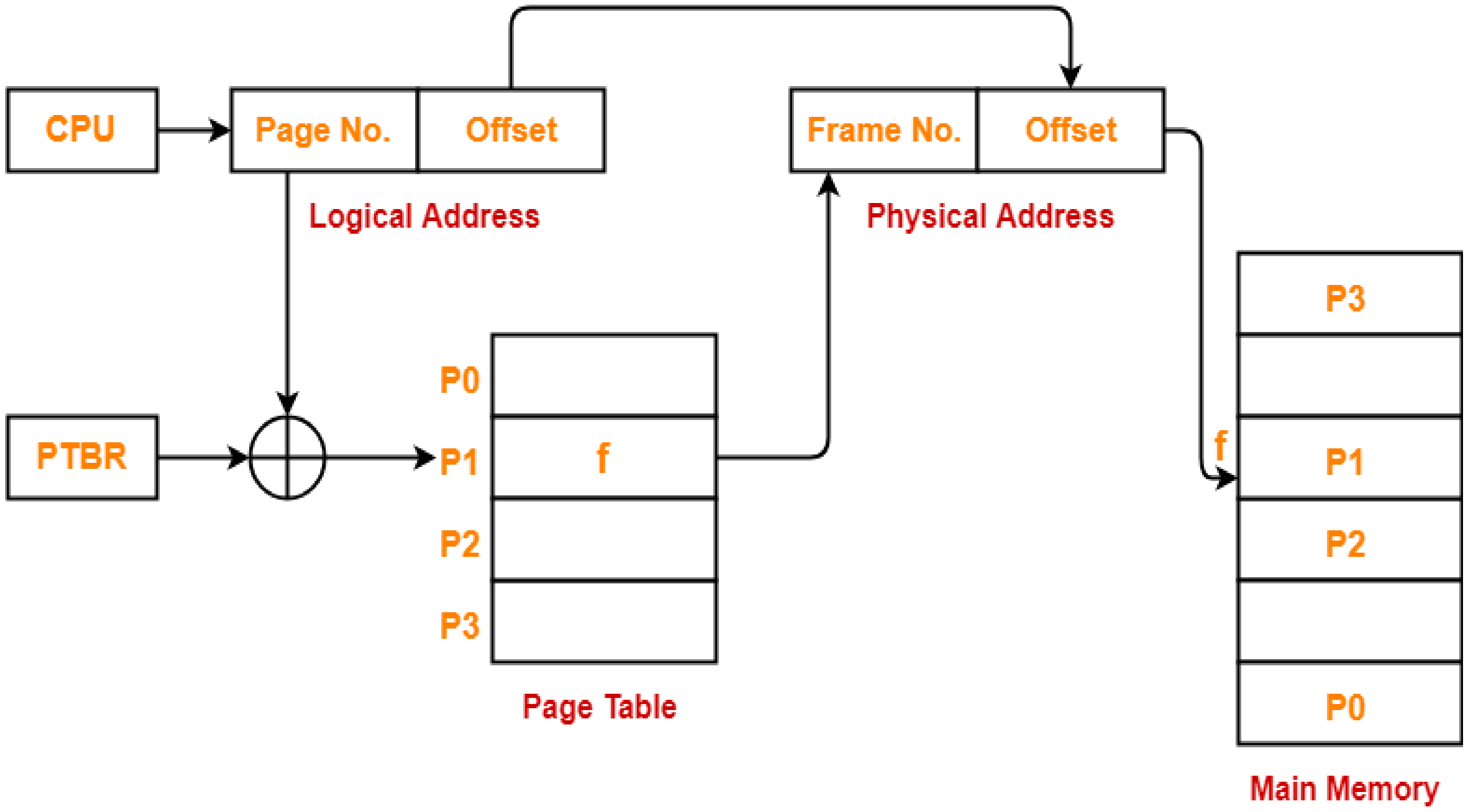


Paging implemented (page 5)

- **Step 2**
  - A page table (one per process) provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.
    - Frame number specifies the specific frame where the required page is stored.
    - Page Offset specifies the specific word that has to be read from that page.



Paging implemented (page 6)



Translating Logical Address into Physical Address

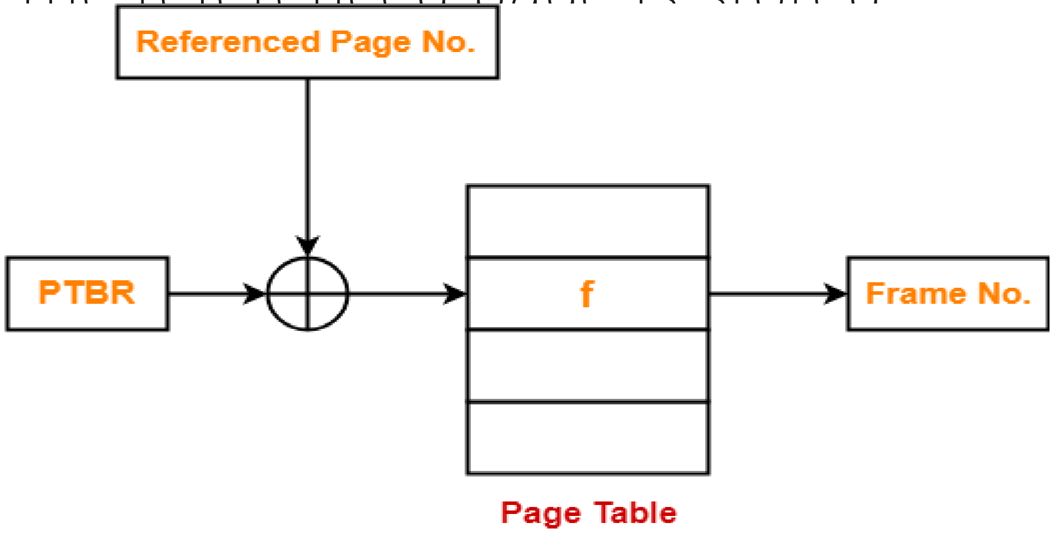
Paging advantages and disadvantages

- **The advantages of paging are**
  - It allows program relocation
  - It allows to store parts of a single process in a non-contiguous fashion.
  - It solves the problem of external fragmentation.
- **The disadvantages of paging are**
  - It suffers from internal fragmentation.
  - There is an overhead of maintaining a page table for each process.
  - The time taken to fetch the instruction increases since now two memory accesses are required.

Reading a Page Table Entry

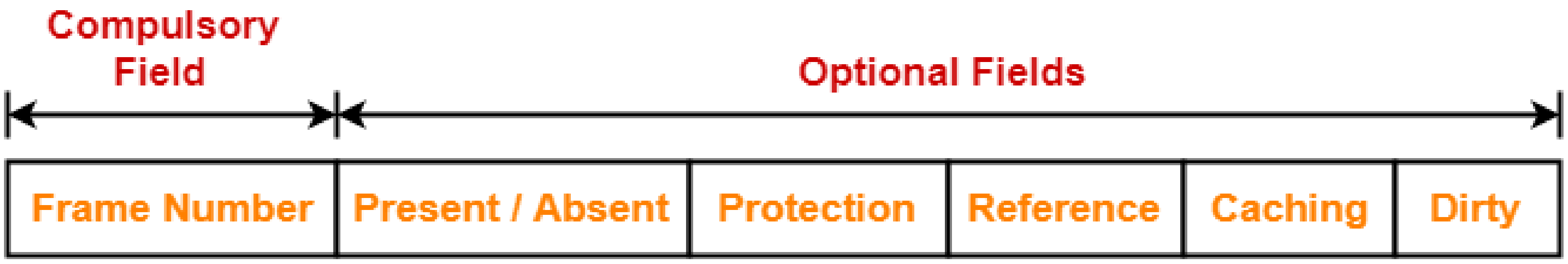
Page Table Base Register (PTBR) provides the base address of the page table.

- The base address of the page table is added with the page number referenced by the CPU.
- It gives the entry of the page table containing the frame number where the referenced page is stored



Obtaining Frame Number Using Page Table

Page Table Entry – Frame definition



Page Table Entry Format

Frame Number

- Frame number specifies the frame where the page is stored in the main memory.
- The number of bits in frame number depends on the number of frames in the main memory.

Present / Absent Bit

- **This bit is also sometimes called as valid / invalid bit.**
- **This bit specifies whether that page is present in the main memory or not.**
- **If the page is not present in the main memory, then this bit is set to 0 otherwise set to 1.**
    - If the required page is not present in the main memory, then it is called as Page Fault.
    - A page fault requires page initialization.
    - The required page has to be initialized (fetched) from the secondary memory and brought into the main memory

Protection Bit

- This bit is also sometimes called as “Read / Write bit“.
- This bit is concerned with the page protection.
- It specifies the permission to perform read and write operation on the page.
- If only read operation is allowed to be performed and no writing is allowed, then this bit is set to 0.
- If both read and write operation are allowed to be performed, then this bit is set to 1.

Reference Bit

- **Reference bit specifies whether that page has been referenced in the last clock cycle or not.**
- **If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.**
  - Reference bit is useful for page replacement policy.
  - A page that has not been referenced recently is considered a good candidate for page replacement in LRU page replacement policy.

Caching Enabled / Disabled

- **This bit enables or disables the caching of page.**
- **Whenever freshness in the data is required, then caching is disabled using this bit.**
- **If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.**

Dirty Bit

- **This bit is also sometimes called as “Modified bit“.**
- **This bit specifies whether that page has been modified or not.**
- **If the page has been modified, then this bit is set to 1 otherwise set to 0.**
  - In case the page is modified,
    - Before replacing the modified page with some other page, it has to be written back in the secondary memory to avoid losing the data.
    - Dirty bit helps to avoid unnecessary writes.
    - This is because if the page is not modified, then it can be directly replaced by another page without any need of writing it back to the disk.

Formulas for Paging – Main Memory

- Physical Address Space = Size of main memory
- Size of main memory = Total number of frames x Page size
- Frame size = Page size
- If number of frames in main memory = 2^X, then number of bits in frame number = X bits
- If Page size = 2^X Bytes, then number of bits in page offset = X bits
- If size of main memory = 2^X Bytes, then number of bits in physical address = X bits

Formulas for Paging – For Process

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

Formulas for Paging – For Page Table

- Size of page table = Number of entries in page table x Page table entry size
- Number of entries in pages table = Number of pages the process is divided
- Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

Formulas for Paging – Important notes

- In general, if the given address consists of ‘n’ bits, then using ‘n’ bits, 2^n locations are possible.
- Then, size of memory = 2^n x Size of one location.
- If the memory is byte-addressable, then size of one location = 1 byte.
- Thus, size of memory = 2^n bytes.
- If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.
- Thus, size of memory = 2^n x m bytes.

Formulas for Paging – For Page Table

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

Translation Look-aside buffer

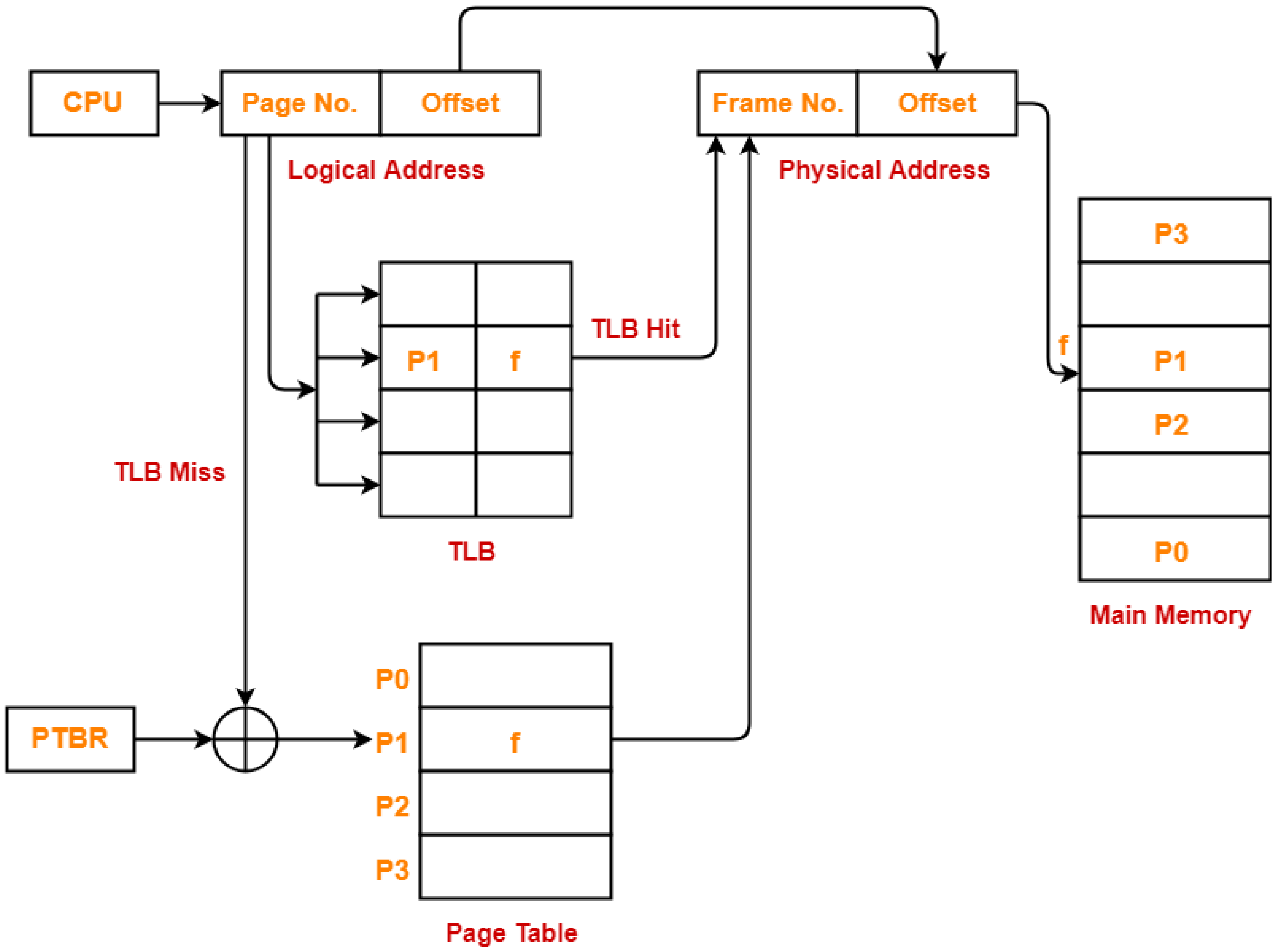
- **Translating Logical Address into Physical Address**
  - In a paging scheme using TLB,, the logical address generated by the CPU is translated into the physical address using following steps
  - Step-01: CPU generates a logical address consisting of two parts-
    - Page Number
    - Page Offset
  - Step-02:TLB is checked to see if it contains an entry for the referenced page number.
    - The referenced page number is compared with the TLB entries all at once.

Now two cases are possible

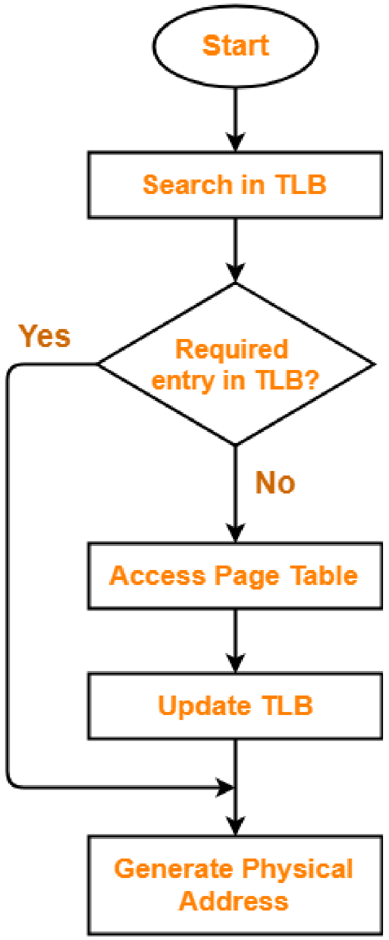
- Case-01: If there is a TLB hit-In this case, TLB entry is used to get the corresponding frame number for the referenced page number
- Case-02: If there is a TLB miss-In this case, page table is used to get the corresponding frame number for the referenced page number. Then, TLB is updated with the page number and frame number for future references.

Finally

- **Step-03:After the frame number is obtained, it is combined with the page offset to generate the physical address. Then, physical address is used to read the required word from the main memory.**



Translating Logical Address into Physical Address


Start
Search in TLB
Required entry in TLB?
Yes
No
Access Page Table
Update TLB
Generate Physical Address

Flowchart

Important Points

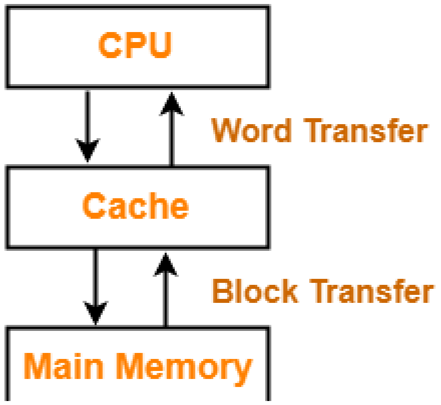
- **Point-01: Unlike page table, there exists only one TLB in the system. So, whenever context switching occurs, the entire content of TLB is flushed and deleted. TLB is then again updated with the currently running process.**
- **Point-02: When a new process gets schedule, initially, TLB is empty. So, TLB misses are frequent. With every access from the page table, TLB is updated. After some time, TLB hits increases and TLB misses reduces.**
- **Point-03: The time taken to update TLB after getting the frame number from the page table is negligible. Also, TLB is updated in parallel while fetching the word from the main memory.**

Advantages and Disadvantages

- **Advantages-**
  - TLB reduces the effective access time.
  - Only one memory access is required when TLB hit occurs.
- **Disadvantages-**
  - After some time of running the process, when TLB hits increases and process starts to run smoothly, a context switching occurs. The entire content of the TLB is flushed. Then, TLB is again updated with the currently running process. This happens again and again.
  - TLB can hold the data of only one process at a time.
  - When context switches occur frequently, the performance of TLB degrades due to low hit ratio.
  - As it is a special hardware, it involves additional cost.

Caching

- Cache Memory is a Random Access Memory that can be accessed at a much faster speed than main memory.
- Cache memory lies on the path between the CPU and the main memory. It facilitates the transfer of data between the processor and the main memory at the speed which matches to the speed of the proce RAM vs DRAM).

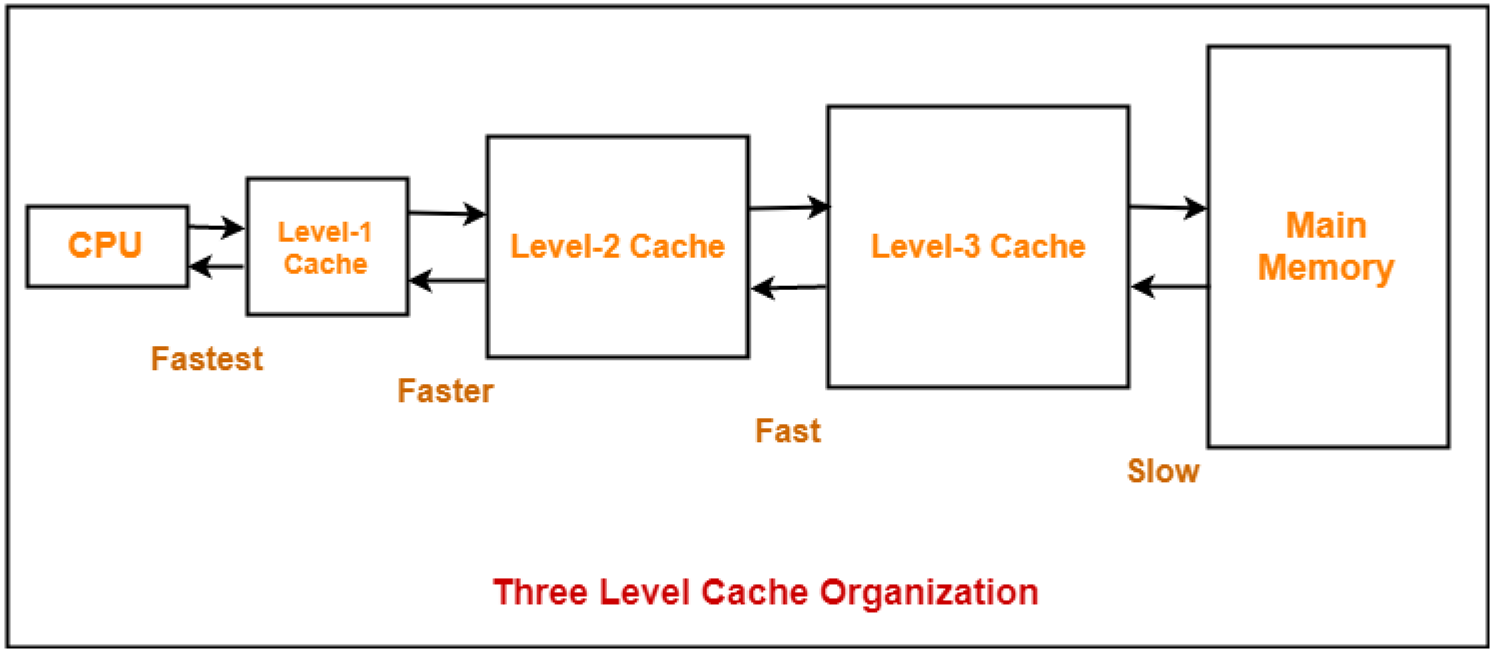


How caching works

- The memory subsystem first looks in the cache memory for the needed data. If it is found, it is called a ***cache hit***, and data is retrieved from cache.
- **In parallel with the checking of the cached, the memory subsystem starts the request for the data from memory. If the data is found in cache, this memory read is halted. If it is not found in cache (known as a cache miss), the read continued.**
- **Some cache access value:**

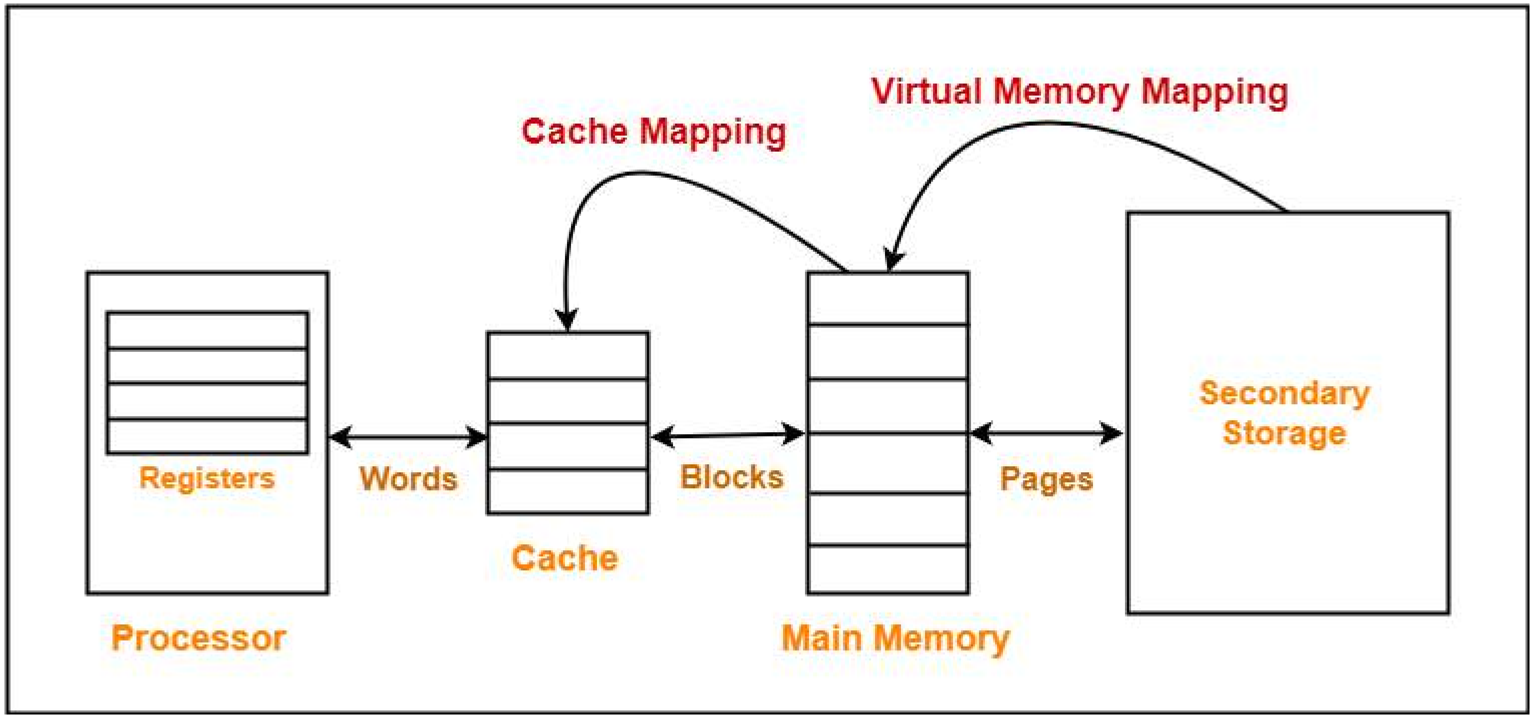
•local L1 CACHE hit,	~4 cycles (2.1 - 1.2 ns)
•local L2 CACHE hit,	~10 cycles (5.3 - 3.0 ns)
•local L3 CACHE hit, line unshared	~40 cycles (21.4 - 12.0 ns)
•local L3 CACHE hit, shared line in another core	~65 cycles (34.8 - 19.5 ns)
•local L3 CACHE hit, modified in another core	~75 cycles (40.2 - 22.5 ns)
•local DRAM	~60 ns
•remote DRAM	~100 ns

Multilevel Cache



Three Level Cache Organization

Cache Mapping



Types of Cacheing architecture

- **Direct Mapping**
  - How to do a Modulo operation quickly
  - No need for a replacement algorithm
- **Fully associative mapping**
  - Any block can map to any cache block (think of the TLB)
  - Needs a replacement algorithm
- **K-way Set Associative Mapping**
  - Blocks map to groups of K blocks that are fully associative.