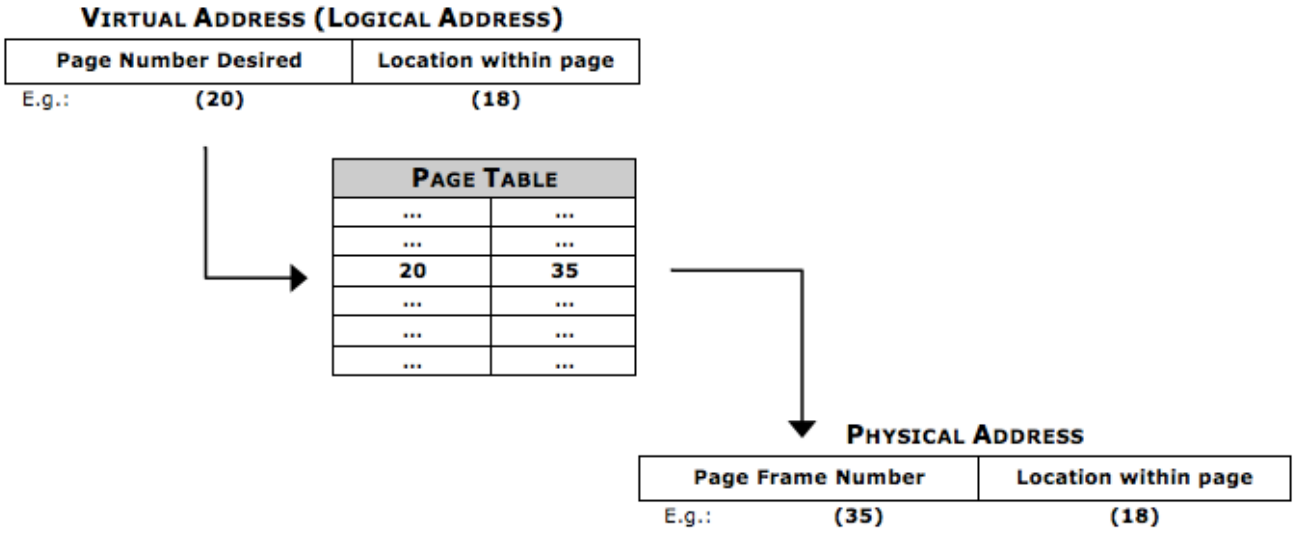
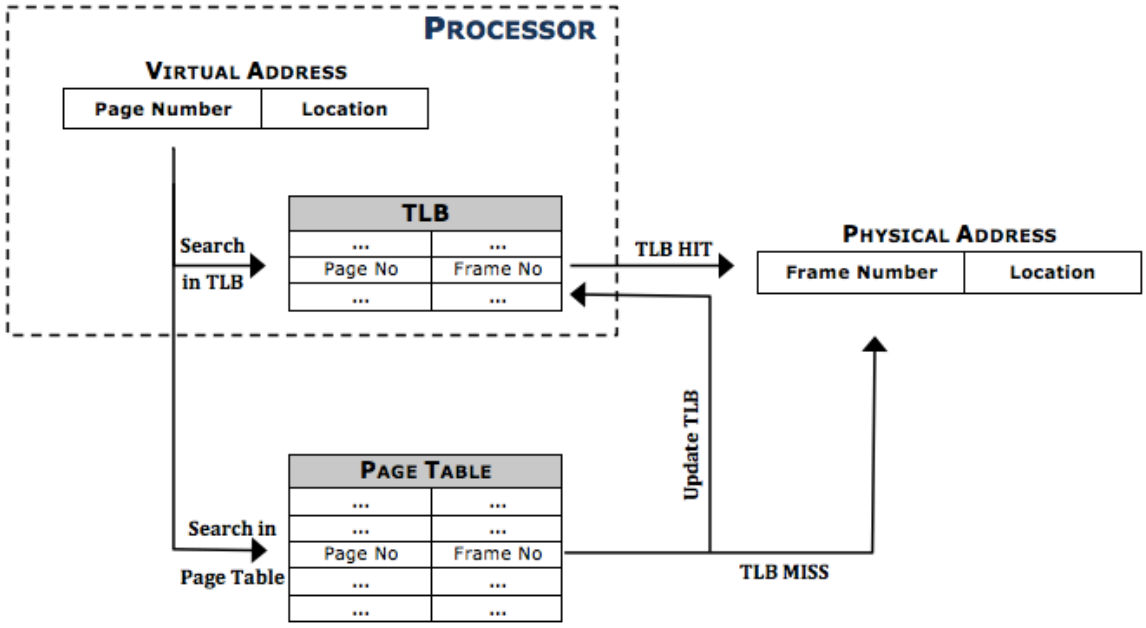
VIRTUAL MEMORY MANAGEMENT USING PAGING

1) The **Virtual Memory space** is divided into **equal size blocks** called **"Pages"**.
2) The **Physical Memory space** (also called Main Memory) is also divided into **equal size blocks** called **Page Frames** (also simply called pages).
3) **Pages are loaded** from Virtual Memory **into any** available page frame of Physical Memory.
4) Consider example of 80386 Processor. Physical Memory = 4 GB. Page Size = 4 KB.
   No of pages in Physical Memory = 4 GB / 4 KB = 1M. ($2^{32} / 2^{12} = 2^{20}$)
5) When a page is needed, Processor **checks** if the **desired page** is **present** in the **Physical Memory**.
6) If found, it is called a **"HIT"** and the operation is performed on the Physical Memory.
7) If the **desired page** is **not present** in the **Physical Memory** its called a **MISS** or a **Page Fault.**
8) On a Page Fault the desired page is loaded form Virtual Memory into Physical Memory.
9) If no empty page frame is available in Physical Memory, then a **"Page Replacement"** is performed.
10) An old page in Physical Memory is replaced with a new desired page from the Virtual Memory.
11) Popular **algorithms** like **FIFO** (First in first out), **LRU** (Least recently used) or **LFU** (Least frequently used) are used to determine which page of the Physical Memory must be replaced.
12) If there are **too many page faults**, the system is said to be **Thrashing**. This must be avoided.
13) While replacing, the **"Dirty Bit"** is checked to determine if a page is **modified** in Physical Memory.
14) **If Dirty bit = 1, then the page has been modified** (is "Dirty") and hence must be copied back into Virtual Memory before being replaced else the modified information will be lost.
15) **If Dirty bit = 0, then the page can be directly replaced**.
16) Since **any page** of Virtual Memory can be **loaded into any page frame** of Physical Memory, a **"Page Table"** is required to give the **mapping between** Virtual Memory **page number** and Physical Memory **page frame number**.
17) **Virtual Address** can be divided in two parts: **Desired Page Number | Location in page**
18) **Physical Address** can be divided in two parts: **Present Page Frame Number | Location in page**
19) The Page Table relates the desired page number to the page frame number (if present) in the Physical Memory. This is how address translation is performed.
20) **But this process is very slow**. To access any location, we must first access the corresponding entry in the page table, then access the page.
21) **Page Table is also in the memory**. This means, to access any memory location we must make an **additional trip** to the memory just to access the page table.
22) To speed up the process a **"Translation Look-aside Buffer"** is used (called TLB).
23) **The TLB is an on chip cache, present inside the processor.**
24) **It stores 32 most recently used Page table entries.**
25) This makes subsequent access to these pages (whose information is cached in the TLB) **much faster as there is no need to access the page table**. Processor can directly obtain the starting address of the page frame from the TLB and hence directly access the page.
26) Due to principle of **"Locality of reference"** most systems get a **Hit ratio of >98%** on the TLB, thus making the operations very fast.
27) An application may require several pages. All are not loaded into Physical Memory at once. Instead, pages are loaded as and when they are required by the processor. This is called **Demand Paging**.

PAGING MECHANISM:



TRANSLATION LOOK-ASIDE BUFFER (TLB)



PAGING V/S SEGMENTATION

	PAGING	SEGMENTATION
1	Paging is a **physical division** of the memory.	Segmentation is a **logical division** of the memory.
2	Pages are of a **fixed size**.	Segments are of **variable sizes**.
3	**Fragmentation** (wastage of unused space) **is high** as pages are of **fixed size.**	**Fragmentation is low** as segments are of **variable size.**
4	Completely **hidden from programmer**.	**Controlled by programmer.**
5	Does **not** offer **much control over protection mechanism**	As segments are controlled by the programmer, we can assign **different privilege levels to segments**. Hence provides better control over protection mechanism.
6	**Does not differentiate between code and data.**	Code and data segments are **treated differently**.
7	Pages are loaded based on **algorithms** such as **FIFO, LRU, LFU** etc.	Segments are loaded based on **algorithms** such as **First Fit, Best Fit** etc.
8	Does **not allow sharing** of code or data between different processes/ tasks.	Segments can be either global or local. **Global segments can be shared** among all tasks.