DSC204A: Scaleable Data Systems Winter 2025

10: Memory Virtualization & Intro to IPC

Instructor: Umesh Bellur *Scribe: Zijie Feng*

1 Memory Virtualization

1.1 Fragmentation

CPU only issues virtual addresses so a process of translation is needed to turn it into a physical address. Parts of the virtual address space containing the code, static data, heap, and stack, would need a place to exist on the physical memory.

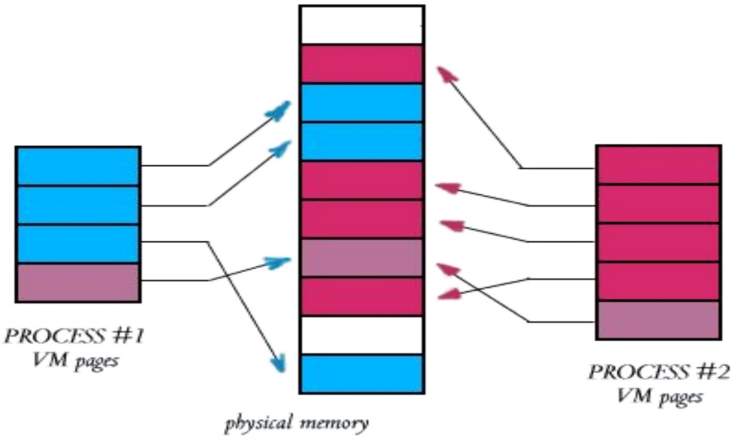
External fragmentation happens because of processes requiring variable amounts of memory space. Space might be wasted because of the ordering of processes living on the memory.

Internal fragmentation can happen when the OS assigns more memory to a process than it actually needs.

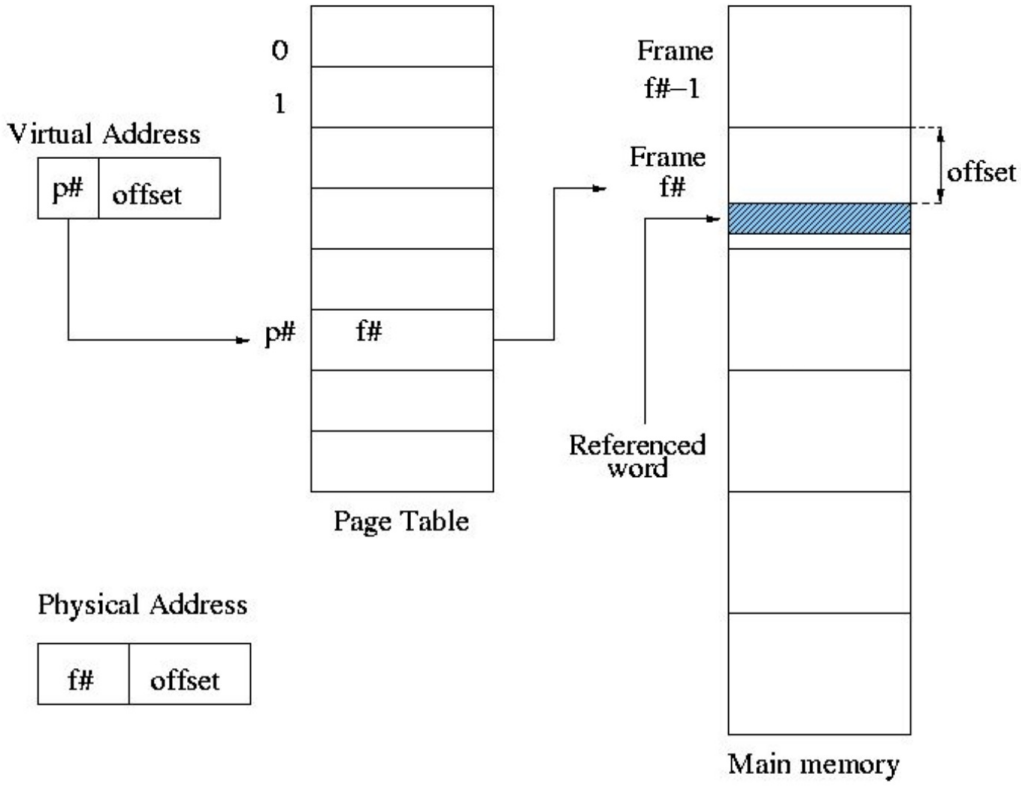
To fix the issue of fragmentation and improve memory efficiency, we can allocated fixed sized chunks (Pages) that are small. However, if the chunks are too small and we keep assigning new chunks all over the place for a process, it's going to take a lot of effort keeping track of all of them.

What size should we choose for the chunks, then? Through experiments, people have found that **4KB** and **8KB** are the optimal choices in most scenarios.

1.2 Paging & Address Translation



In the above figure, we can see an illustration of how the page assignment process works. Different processes have various VM pages corresponding to different locations on the physical memory and these physical locations might not be continuous. The physical location on the memory is where the CPU actually reads the data from.

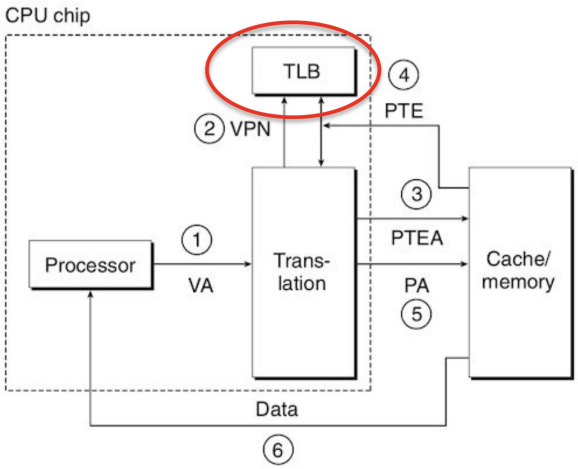


In the above figure, we can see what a virtual address is composed of and how it's used to find data located at a specific location on the physical memory. The virtual address contains a page number and an offset. The page number is used in the Page Table to determine the specific frame number on the memory and offset will then indicate where exactly the target is, on that frame.

The size of a virtual address is fixed so how should we determine how many bits should be used for the page number and how many for the offset? This is related to the specific size of a page we have designed in the system. For example, with the virtual address being 32-bit in length, and pages being all 4KB, 12 bits will be used to store offset in the virtual address since we need exactly 12 bits to represent 4KB.

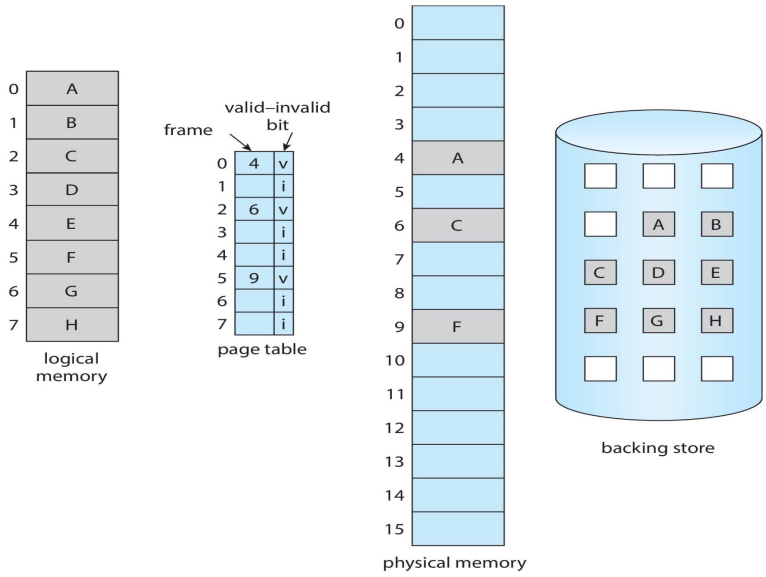
The page table is a very important structure in this process since it maintains records of the relationships between virtual addresses and physical addresses. This table itself also exists in memory.

1.3 Making the translation fast!



As we can imagine, the translation process from virtual to physical can be slow since we have to access the memory for the page table, look up the mapping, and then access the memory again for our target. One way to speed up the process would be using a cache, which involves a special hardware unit called the Translation Lookaside Buffer (TLB). This cache exists close to the processor and the OS would come here first to look for the mapping. Since it's cache, the concept of hit or miss still applies. If the desired mapping is not on the cache then we would still need to access the page table on the memory.

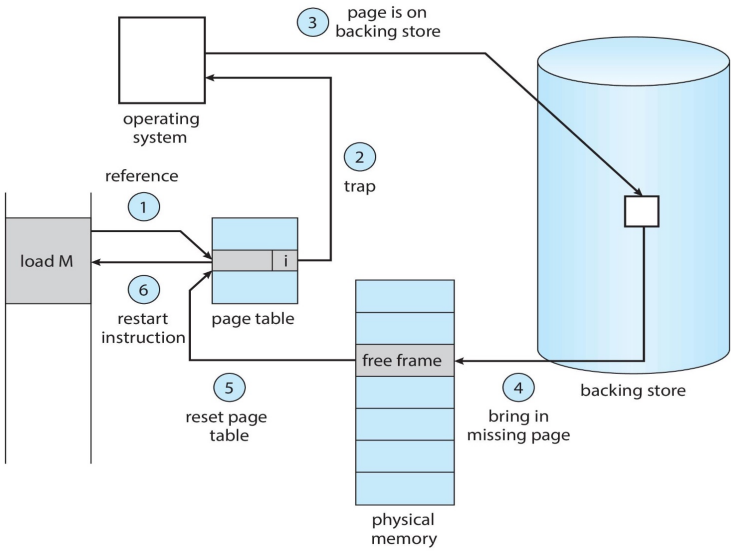
1.4 What happens when the page is NOT in memory?



It's possible for a process to keep growing and eventually needs more space than the physical memory has.

So we need to store on the disk as well. If a process needs anything that was previously stored on the disk, then that page will be put on the memory and accessed. As we can see in the above figure, in the page table, there is one extra bit in the end now, which is called the valid-invalid bit. This indicates the validity of a mapping to the physical memory, not the disk. When using the disk as backing store for the memory, this is called the **Swap Space** and its size and records are all maintained by the OS.

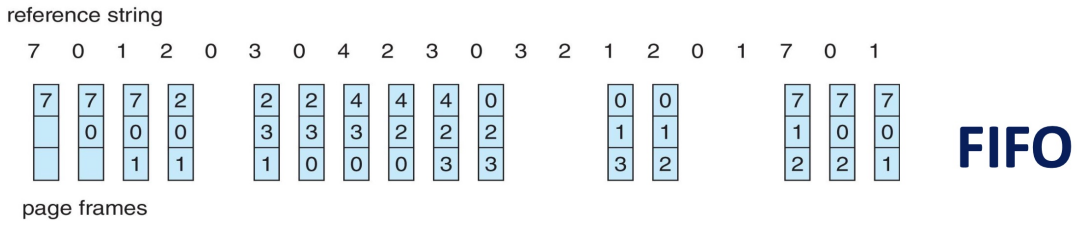
1.5 Handling a Page Fault!



When there is no more space on the memory for us to assign a new mapping, we need to get rid of some other mapping in order to make space and this is called a page fault and the process of handling it can be expensive. The process includes:

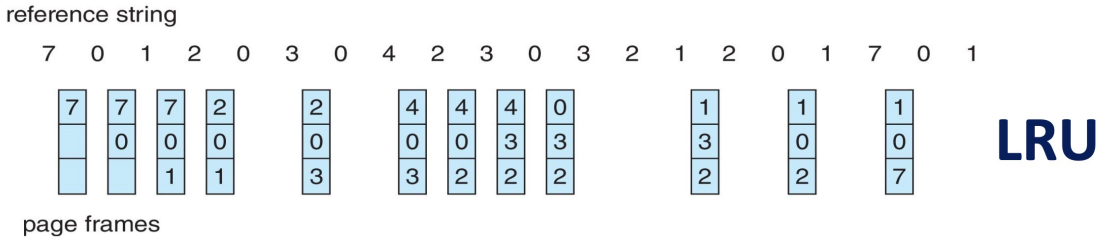
1. Request with a virtual address and page table indicates it's invalid
2. Inquiry with the OS
3. OS indicates the page is on the disk
4. Bring the page to a free frame on the memory or send another page currently on the frame back to the disk if there's no free frame
5. Reset page table since we got the page on the memory now
6. Restart instruction

In this process, when there is no free frame on the memory and we need to move an existing page on the memory to the disk, this moved page is called a "victim". If this victim page has never been written to by a process then we can even simply overwrite it.

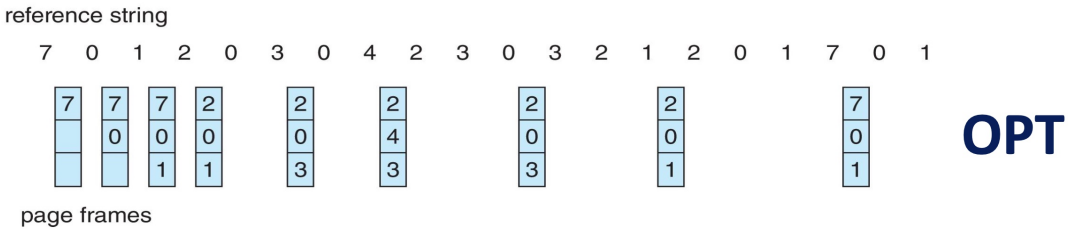


1.6 Which Page?

There are many different methods to determine which page we would choose as the victim page. First In First Out is one of them and it's quite simple, bumping out the earliest page for every page fault.



Least Recently Used (LRU) is another method to determine the victim page. It determines that we should kick out the least referenced page recently to improve efficiency. This is quite a popular method found in various Operating Systems. However, there is a slight downside for this method which is the fact that the system needs to keep track of how much each page is being utilized and referenced, which also takes effort.



Another ideal method would be OPT, which allows us to see future incoming pages to ensure we can keep as many useful pages on the memory as possible. In the above figure, you can see this method generates fewer page faults.

When there are multiple processes competing for memory spaces, causing a large amount of page faults, performance will be greatly affected and this is a phenomenon called thrashing.

1.7 The Working Set Model

Instead of blindly using a specific strategy for bumping pages in between memory and disk, the working set model can determine a working set of pages for each process. The OS can then keep these sets of pages on the memory and the other pages can be dropped to save space. This ensures lower possibility for thrashing and maximized CPU utilization.

2 Intro to IPC

2.1 The Problem

Each process has its own private address space that is protected by the OS. However, two processes might need to communicate with each other in order to know the status of the other process's execution. How should two processes communicate with each other? An IPC is built for this purpose.

2.2 Desirable Characteristics of IPC

1. Fast
2. Easy to use
3. Well defined synchronization model
4. Versatile
5. Easy to implement
6. Works remotely