Virtual Memory
- At any time there are many processes in the computer, each of which uses only a small amount of vast address space.
- This small part is mapped onto physical memory.
- There are protection mechanisms in place to prevent one process accessing memory allocated to another process.
- Motivation for virtual memory came from trying to fit large programs into a small memory – not all data/instruction blocks need to be in memory at all time.

Virtual Memory Organization
- Virtual memory can be divided into fixed size blocks (pages) or variable size blocks (segments).
- Pages are easier to manipulate but a large portion of a page might be unused by the program (internal fragmentation).
- Segments provide better usage of data loaded into main memory but fragment the main memory (external fragmentation).

Virtual Memory Organization
- Virtual memory simplifies loading program for the execution – it can run in any memory space.
- Relationship between virtual memory and main memory is similar to the relationship between the main memory and cache memory but terminology is different:
  - Block → Page or Segment. Page number is derived from the data address, segment number requires separate address information.
  - Cache miss → Page fault.
  - Translation of virtual to physical address is called address translation or mapping.
  - Replacement is controlled by OS (not by hardware like in case of caches).

Where Can a Page Be Placed In Memory
- Page fault penalty (miss penalty) is very expensive – involves disk rotation.
- A sophisticated placement algorithm (such as fully associative) reduces miss rate and does not much increase hit time.

How to Find a Page In Memory
- Page table holds the physical address of the page or the segment.
  - Indexed by a page or segment number.
  - For segmentation, segment offset is added to table information to get the address of data in memory.
  - For paging, page offset is concatenated to table information to get the address of data in memory.
Page Table

- Page number
- Page offset
- Virtual address
- Page frame number
- Physical address to main memory

Inverted Page Table

- Page number
- Page offset
- Virtual address
- Inverted page table
- Physical address to main memory

Which Page to Replace on a Miss
- To minimize page faults, most operating systems replace least recently used (LRU) page.
- To facilitate easy discovery of LRU page, most computers provide a use bit or reference bit that is set each time a page is accessed and periodically cleared.
- This does not help us find exactly the LRU page, but a set of pages that are least recently used.

What Happens on a Write
- Since it is very expensive to access disk each time data is written, all virtual memory systems are write back.
- Each page includes a dirty bit to avoid unnecessary writes.

Techniques for Fast Address Translation
- Page tables are usually large and it would take a lot of time and hardware to search them entirely every time we need to access memory.
- One access to page table (in memory) to find physical address and one access to get data from memory.
- We can remember past translations to avoid accessing page table.
- Store past translations in a separate cache – Translation Lookaside Buffer (TLB).

TLB
- Usually fully associative.
- An entry holds:
  - A tag (unique identifier, this is a portion of virtual address)
  - Physical address (page frame number)
  - Protection bit
  - Valid bit
  - Sometimes dirty bit for the page and use bit for the page
- To reduce TLB misses due to context switches, each entry has an 8-bit address space number.
Selecting Page Size
- Smaller pages require larger page table → increase hit time
- We should favor larger pages
- Transferring larger pages from disk is more efficient
- Larger pages reduce the number of TLB misses
- The only drawback is that larger pages are underutilized – a large portion of them is never accessed

VM and Cache Access Diagram
- Page size is 8KB
- TLB is direct mapped with 256 entries
- L1 cache is direct mapped and size is 8KB
- L2 cache is direct mapped and size is 4MB
- Block size is 64B for L1 and L2
- Virtual address is 64 bits and physical address is 41 bits long

VM and Cache Access Diagram With Optimization So L1 Holds Virtual Tags

VM Protection
- Multiprogramming leads to a concept of a process – a program that has its own data space
- Many processes share CPU and memory - one process owns the CPU at one time
- Change of ownership is called context switch
- A process must operate correctly through context switches
- One process cannot access memory that holds data from another process (say through escaped pointer)

VM Protection
- Simplest approach to make sure process accesses cannot “escape” is to check every address a process wants to access against two registers Base and Bound which define start and end of this process’ space in main memory
- System processes can change those registers but user’s cannot
- There is also page level protection so that memory portions holding executable code cannot be overwritten by this code
- A read-only bit is assigned to each page