Virtual Memory

Virtual memory is a memory management technique that creates an **illusion of a large memory space** for programs, even if the actual physical memory is smaller. It allows programs to use more memory than is physically available by temporarily transferring data between **RAM** and **disk storage**.

- **Logical Address Space**: Programs operate in a logical address space, which is larger than physical memory.
- **Page Mapping**: The operating system uses a **page table** to map logical addresses to physical memory locations or to disk storage (swap space).

Advantages of Virtual Memory

1. **Efficient Memory Usage**: Programs only load the necessary parts into RAM, reducing memory wastage.
2. **Support for Larger Programs**: Enables execution of programs larger than the physical memory.
3. **Multiprogramming**: Allows multiple programs to run simultaneously, sharing physical memory.

Demand Paging

Demand paging is a lazy loading technique where program pages are loaded into memory **only when they are accessed**.

How Demand Paging Works

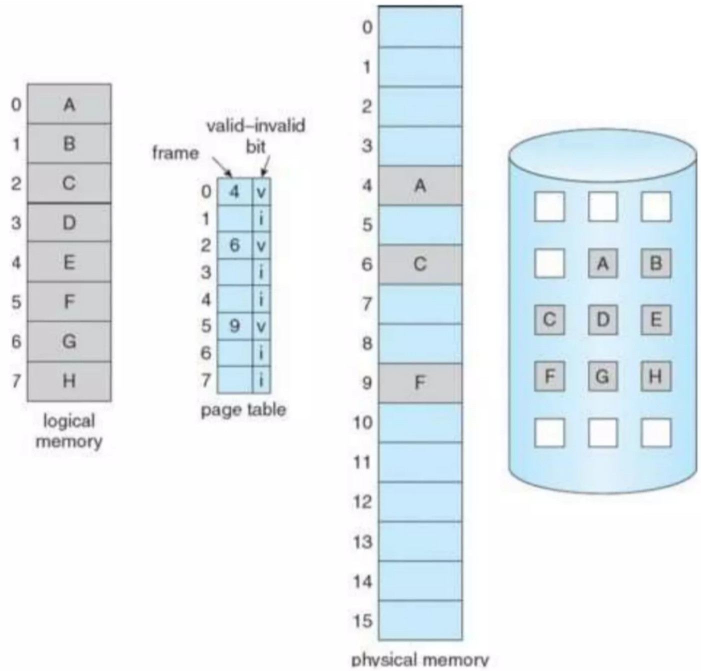
1. When a program requests a page, the operating system checks if it is in memory.
2. If the page is not in memory, a **page fault** occurs.
3. The operating system retrieves the page from secondary storage (disk) and loads it into physical memory.

Page Fault

A **page fault** occurs when a program tries to access a page that is not currently in physical memory.

Steps During a Page Fault

1. **Trap**: The CPU traps to the operating system.
2. **Check**: The operating system checks the page table to determine if the page is valid.
3. **Load Page**: If valid, the page is fetched from disk and loaded into memory.
4. **Update Page Table**: The page table is updated with the new page's location in memory.
5. **Resume Execution**: The process resumes from the point where the page fault occurred.



Performance of Demand Paging

The performance of demand paging depends on the **page fault rate**, which measures the frequency of page faults.

1. **Page Fault Rate (p)**:
   - **p = 0**: No page faults (ideal case).
   - **p = 1**: Every access results in a page fault (worst case).
2. **Effective Access Time (EAT)**:
   The effective access time combines the cost of accessing memory and handling page faults:

   EAT = (1–p) × Memory Access Time + p × (Page Fault Service Time)

Effective Access Time (EAT)

$$\begin{aligned} EAT = (1 - p) \text{ x memory access} \\ + p \text{ (page fault overhead} \\ \text{+ swap page out} \\ \text{+ swap page in} \\ \text{+ restart overhead} \\ ) \end{aligned}$$

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then
  EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!

High page fault rates significantly degrade performance due to the slow disk access involved in handling faults.

Page Replacement

When physical memory is full and a new page needs to be loaded, the operating system must replace an existing page using a **page replacement algorithm**.

Common Page Replacement Algorithms

1. **FIFO (First-In, First-Out)**:
    - The oldest page in memory is replaced.
    - Simple but may lead to **Belady's Anomaly** (more frames can increase page faults).

	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	0	0	7	7	7
F2		0	0	0	0	3	3	3	2	2	2	2	2	2	2	1	1	1	1	1	0	0
F3			1	1	1	1	0	0	0	3	3	3	3	3	3	3	2	2	2	2	2	1
	X	X	X	X	✓	X	X	X	X	X	X	✓	✓	✓	✓	X	X	✓	✓	X	X	X

2. **Optimal Replacement**:

- Replaces the page that will not be needed for the longest time.
- Requires future knowledge, so it is theoretical.

3. **LRU (Least Recently Used)**:
    - Replaces the page that has not been used for the longest time.
    - Effective but requires additional hardware or data structures.