**Memory Management**

**Introduction**

Memory management in Linux involves the efficient allocation, tracking, and release of memory resources. The Linux kernel uses a virtual memory system, where each process is given its own isolated address space, with the kernel mapping it to physical memory. It employs paging to manage memory, dividing it into fixed-size blocks called pages. Linux also uses a combination of swap space and RAM to handle memory overflows. The system ensures that processes get the memory they need while minimizing fragmentation, and it reclaims unused memory through garbage collection techniques like page reclamation.

**Segmented Memory**

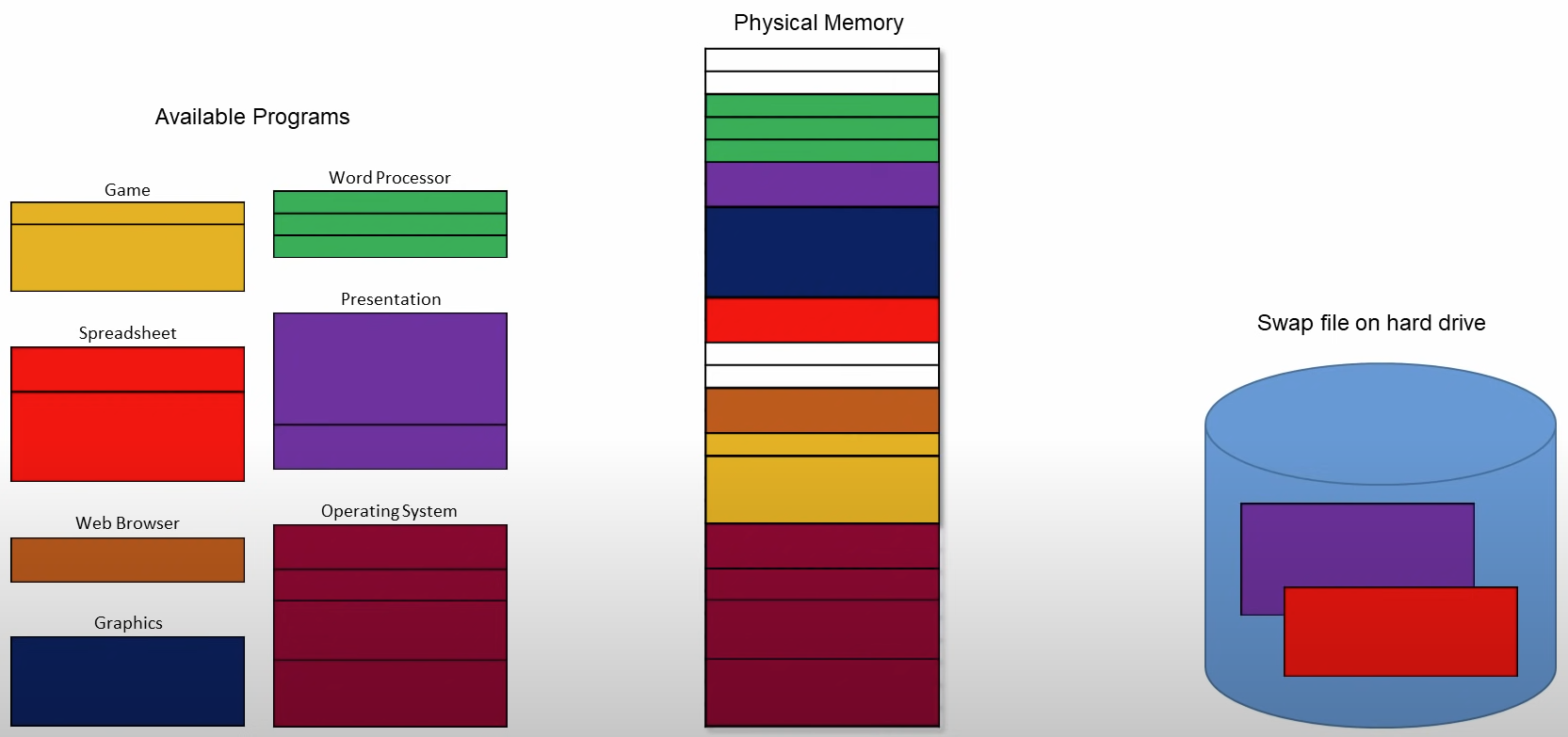
1. Segments are swapped between primary memory (RAM) and secondary memory (ROM)
2. Program segment corresponds to blocks of program code such as procedures or functions.
3. Data segment corresponds to data structures like stack, queue, graph, etc.
4. The OS knows about start and size of the segment in memory.
5. Segmentation can result in memory fragmentation (unused space in between segments).
6. A segment in memory can be only replaced by a segment of the same or smaller size.
7. Large memory segments may not be allowed in the memory very often.

**Paged Memory**

1. Memory is split up into small, equally sized sections called pages (4 Kb in size).
2. A single application may occupy multiple pages that are not necessarily contiguous.
3. Each application has its own view of memory (i.e.) it has its own virtual address space called virtual memory.
4. A page table records where the different pages of a program are located in the physical memory. It is used to map virtual pages to physical memory.
5. The virtual memory is divided into pages and the physical memory is divided into frames.
6. When a process tries to access a page that is not currently in physical memory, a page fault occurs. The operating system then loads the required page from disk into RAM, replacing an existing page if necessary
7. If RAM is full, the operating system swaps out pages that are not actively used to disk (swap space) to free up space for other pages. This allows the system to handle larger workloads than can fit into physical memory at once

**Memory Access in Earlier Days**

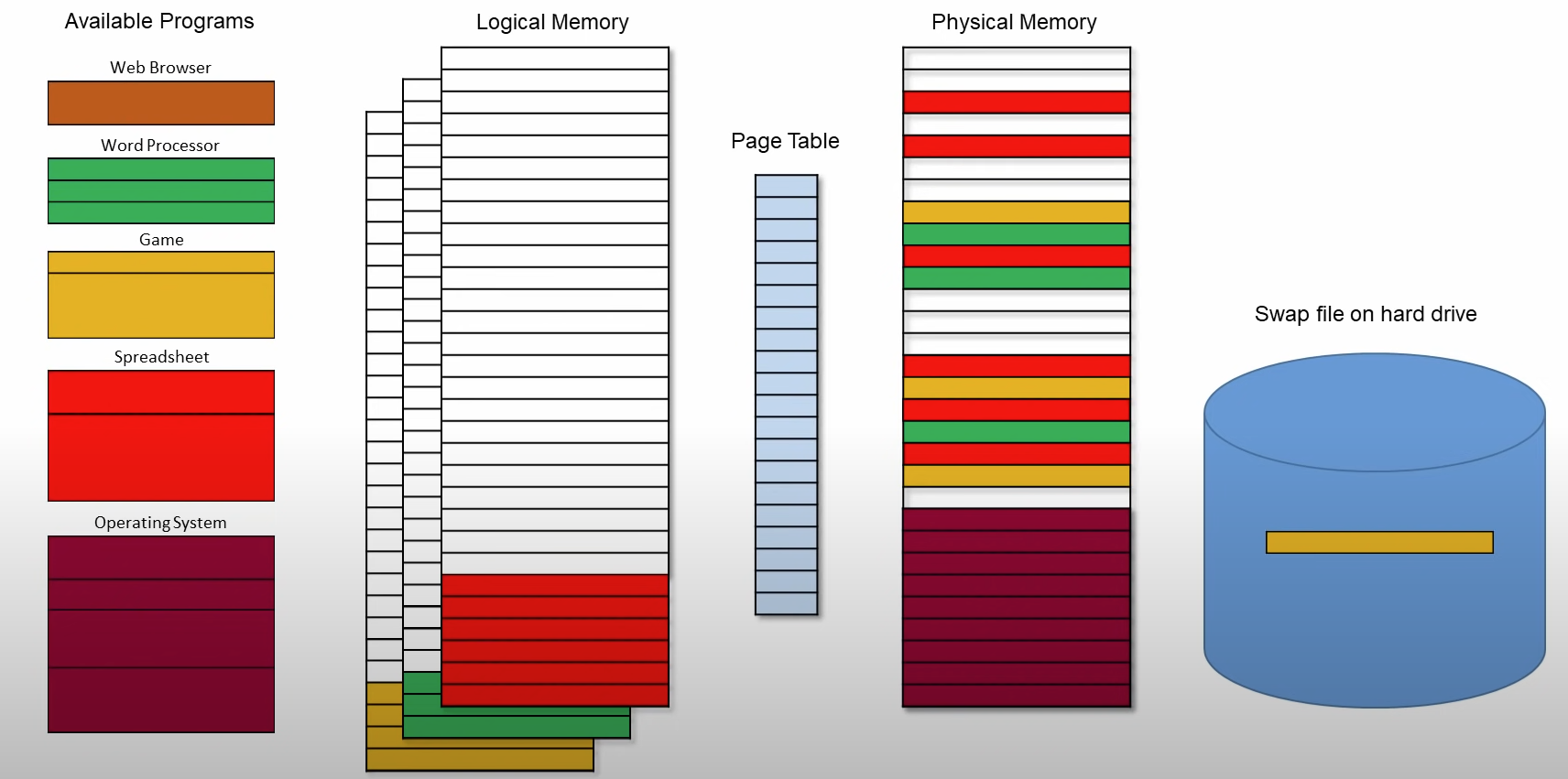
1. When a program (Game, MS Word, MS PPT, etc) is executed, an instance of the running program called a process is created by the kernel.



1. The program data is loaded into the physical memory (RAM) from the HDD/SSD for faster access.
2. The process then occupies the space available in the physical memory (RAM) in contiguous fashion (Segmented Memory) and the OS knows which program requires how much memory and it allocates it to the respective process.
3. The OS also stores the starting segment and ending segment of the process to access the data of the process.
4. If the physical memory is occupied completely, any idle process is sent back to the main memory (HDD/SSD) and the space is given to the new process which asks for the memory, as all processes occupied contiguous space if there was some free space available that was not contiguous it could be allocated to this new process.
5. Also earlier the physical memory had a single address space where all processes occupied its space in the same address space, this made processes corrupt or interfere with the data of other processes.
6. To overcome these issues paged memory was introduced.

**Memory Access at Present Time**

1. When we open google chrome, the OS loads the browser’s essential data (only code required to initiate the program) into physical memory and the rest of the pages are in swap space in the disk.



1. Google chrome has its own virtual address space which is divided into pages.
2. The operating system uses a page table to map virtual addresses to physical memory. The page table keeps track of which virtual pages correspond to which physical frames in RAM.
3. When Chrome needs to access a specific memory location (e.g., loading the user interface, displaying web pages), it generates a virtual memory address. If the required page is already in RAM (cached in physical memory), the system simply accesses the data from the physical memory address.
4. If the page is not in RAM (e.g., it was swapped out to the hard disk or hasn't been loaded yet), the system triggers a page fault. The operating system then loads the missing page into physical memory from the disk (swap space) or another part of RAM, and updates the page table accordingly.