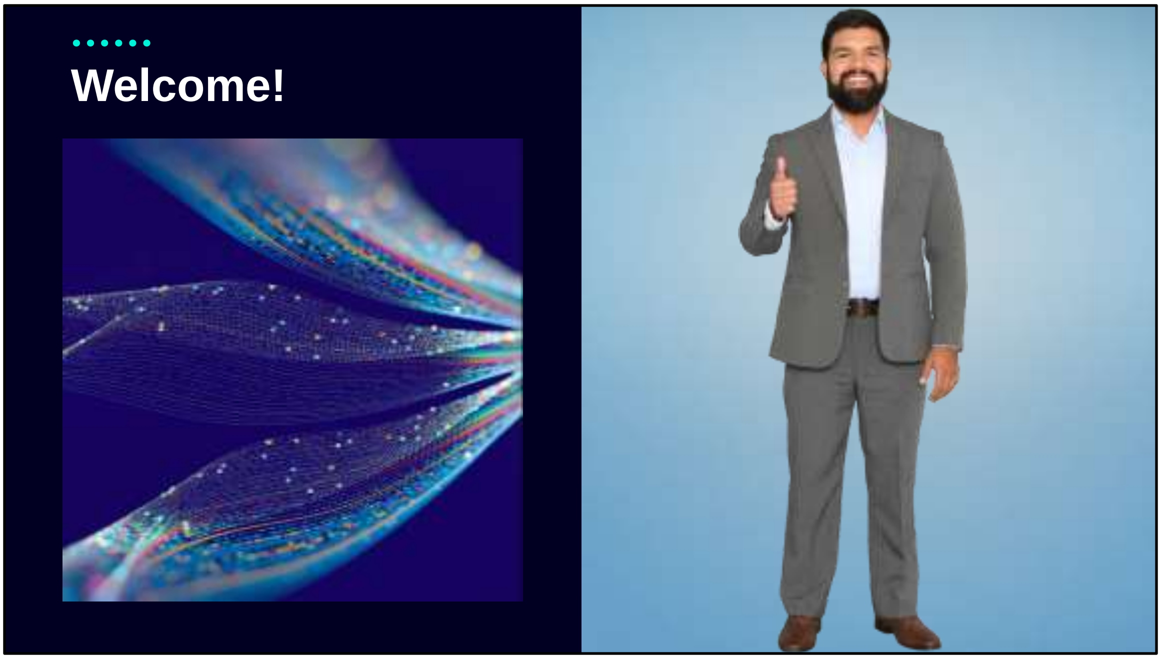
Memory Partitioning



Welcome!

ON SCREEN

Welcome back! In this video, we will look into Memory Partitioning. Let’s get started!

[CLICK – Next Slide]

Memory Partitioning

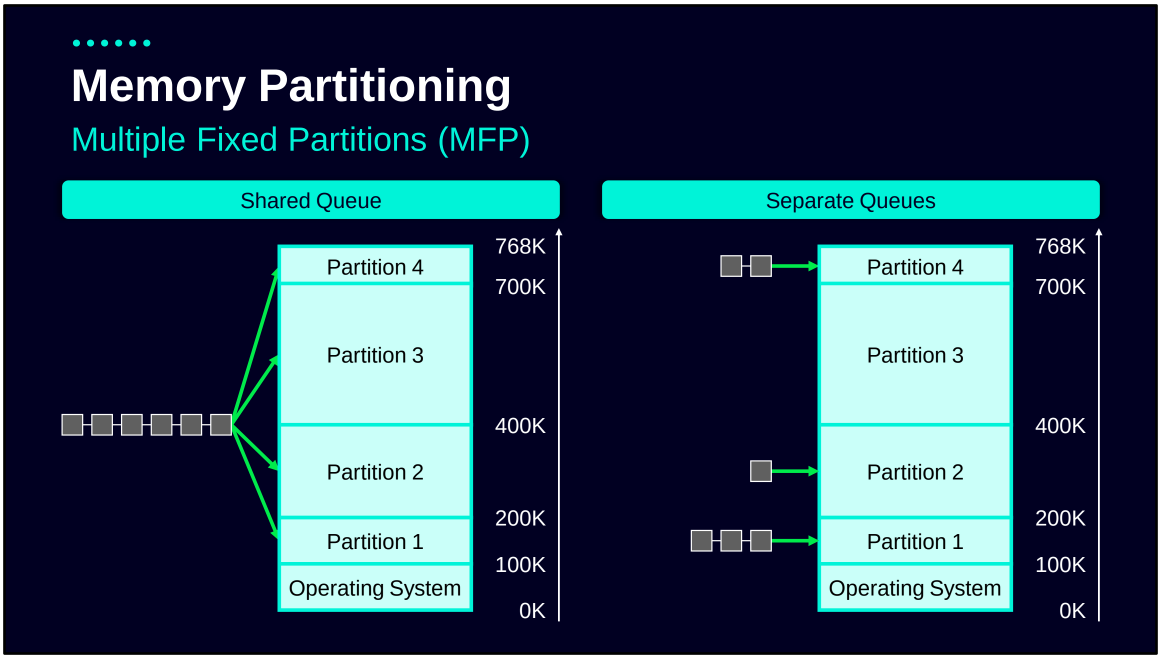
- Multiple Fixed Partitions (MFP)
- Multiple Variable Partitions (MVP)

ON SCREEN

When looking at memory partitioning, the goal is to decide how to split memory into usable spaces. **[CLICK]**

We will look at two approaches to partition memory: Multiple Fixed Partitions and Multiple Variable Partitions.

[CLICK – Next Slide]



Memory Partitioning

Multiple Fixed Partitions (MFP)

Shared Queue

Partition 4

Partition 3

Partition 2

Partition 1

Operating System

768K

700K

400K

200K

100K

0K

Separate Queues

Partition 4

Partition 3

Partition 2

Partition 1

Operating System

768K

700K

400K

200K

100K

0K

OFF SCREEN

Memory in multi-programmed environments must be partitioned and allocated to processes.
When allocated, each process occupies one of the existing partitions.
One approach is to use Multiple Fixed Partitions. **[CLICK]**

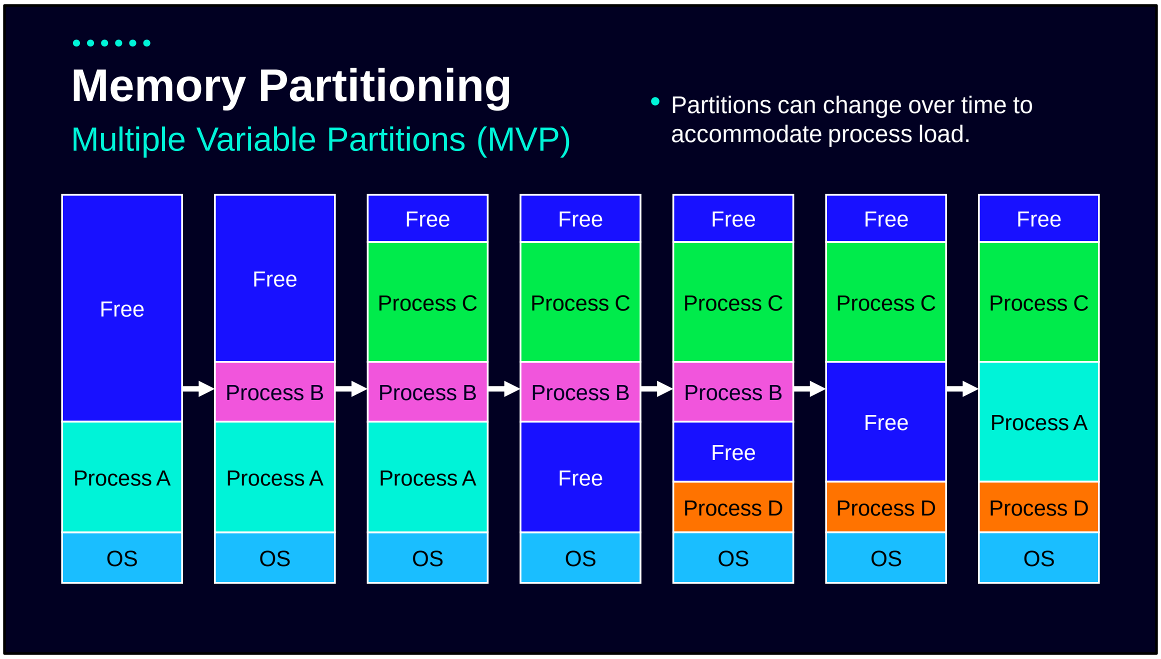
As the name indicates, in this approach, partitions do not change sizes. Partitions may have different sizes, but their sizes are fixed. **[CLICK]**

There are two ways to allocate process to partitions: using a shared Queue or using a Separate Queues. **[CLICK]**

In the Shared Queue approach, processes enter a single queue and are allocated to the first slot available.
The downside is that a small program may take a big partition, wasting memory space. **[CLICK]**

In the Separate queues approach, each partition has its own queue. Processes may enter the queue that best fit their memory requirement needs.
So, a process that requires a small memory partition will enter a queue for a small partition.
This avoids the situation where a small process is allocated to a big partition.
However, there is a downside. It is possible to have processes waiting in their queues while there are partitions not being used. This also represents a waste of resources.

Several old systems, including personal computers, used the multiple fixed partitions approach. Later, this strategy was replaced by multiple variable partition approaches.
[CLICK – Next Slide]



Memory Partitioning

Multiple Variable Partitions (MVP)

- Partitions can change over time to accommodate process load.

Free Process A OS	Free Process B Process A OS	Free Process C Process B Process A OS	Free Process C Process B Free OS	Free Process C Process B Free Process D OS	Free Process C Free Process D OS	Free Process C Process A Process D OS

OFF SCREEN

As mentioned, an alternative to fixed partitions is variable partitions. **[CLICK]**

With multiple variable partitions, partitions can change sizes over time to accommodate varying process loads. **[CLICK]**

Let’s look at how multiple variable partitions work by analyzing an example.

Initially, we have memory allocated to the OS and to process A. [CLICK]

At some point, process B enters memory. Note that the size of the partition allocated match the needs of process B. [CLICK]

And, another process, process C, is loaded into memory too. [CLICK]

At some point Process A is removed from memory. The memory area that was occupied by process A is now Free. [CLICK]

Next, process D is loaded into memory. Note that process D occupies a portion of the partition that had been occupied by process A. That partition changed sizes. Now, part of that partitioni holds process D and part remains free. [CLICK]

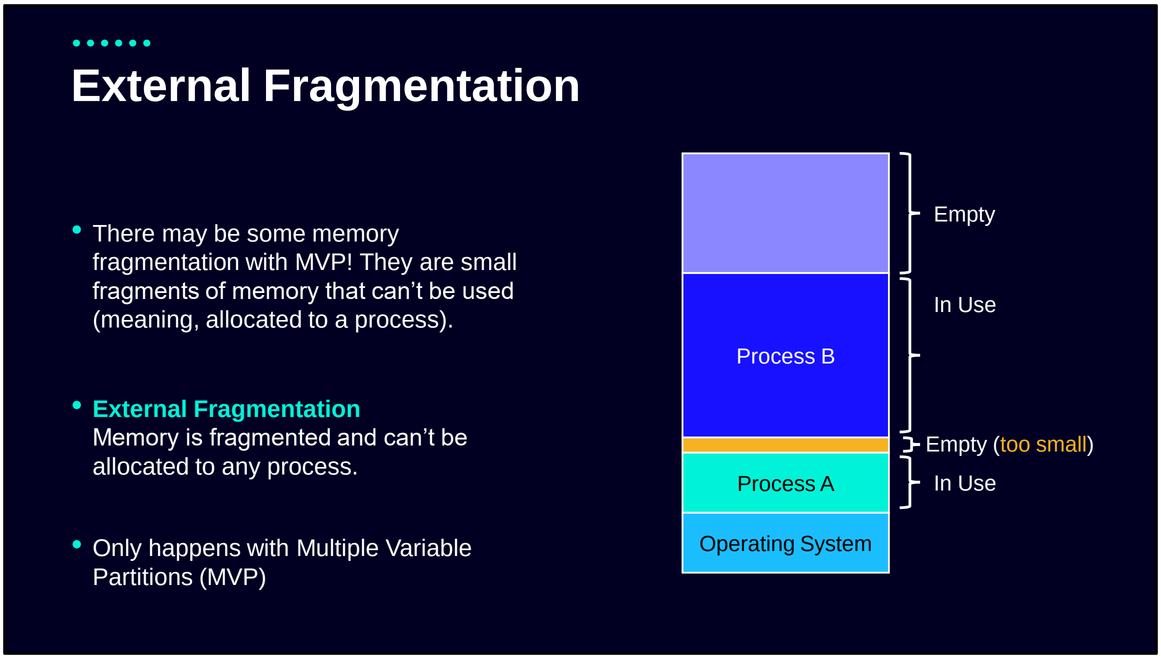
Then, process B terminates, and another memory area becomes available. [CLICK]

Finally process A is loaded into memory once again.

Note that with MVP, partitioins change size over time. In other words, partitions are variable, hence the name multiple variable partitions. [CLICK]

This looks interesting… but there is a problem with this approach…

[CLICK – Next Slide]



External Fragmentation

- There may be some memory fragmentation with MVP! They are small fragments of memory that can't be used (meaning, allocated to a process).
- **External Fragmentation**
  Memory is fragmented and can't be allocated to any process.
- Only happens with Multiple Variable Partitions (MVP)

OFF SCREEN

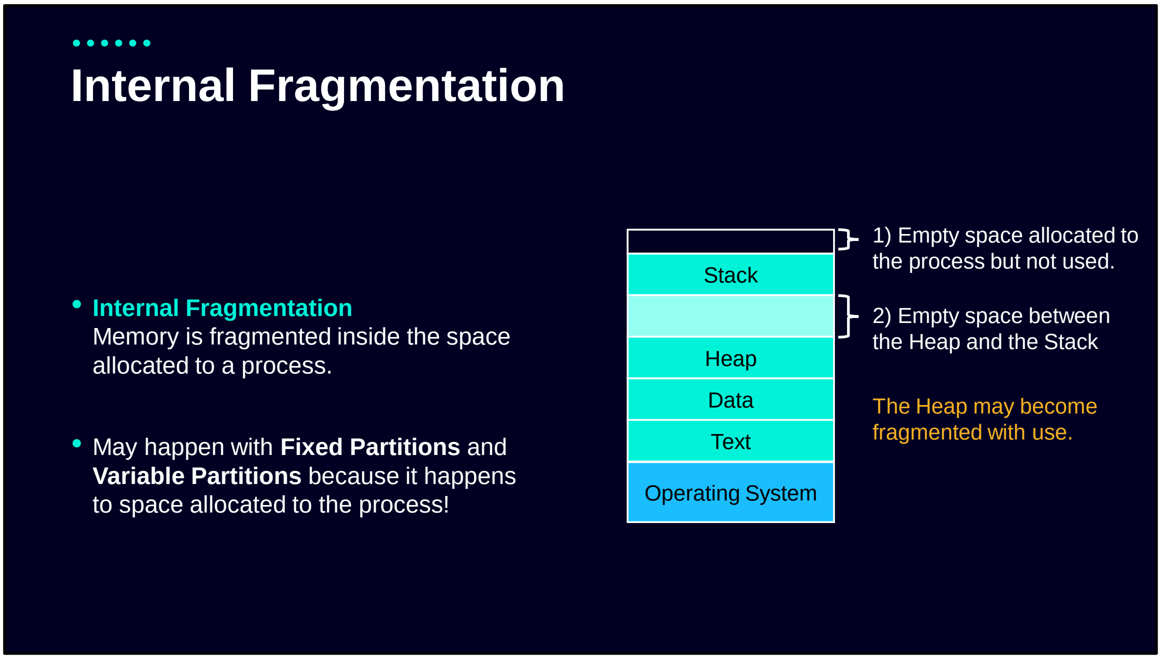
With MVP, there may be some memory fragmentation!
Memory fragmentation occurs when there are small fragments of memory that are so small that can't be used. **[CLICK]**

With MVP, as processes enter and leave memory, free space may remain between allocated partitions.
If the free space is too small to be used, fragmentation has occurred.
[CLICK]

More specifically, this is called External Fragmentation.

Remember: External Fragmentation occurs with memory that can't be allocated to any process because it is too small.

[CLICK – Next Slide]



Internal Fragmentation

- **Internal Fragmentation**
  Memory is fragmented inside the space allocated to a process.

- May happen with **Fixed Partitions** and **Variable Partitions** because it happens to space allocated to the process!

1) Empty space allocated to the process but not used.

2) Empty space between the Heap and the Stack

The Heap may become fragmented with use.

OFF SCREEN

Another type of fragmentation is Internal Fragmentation.
Internal fragmentation happens when memory that is allocated to a process is not used due to one of two factors:
allocated memory becomes fragmented and too small to be used (this can happen with the Heap)
or because more memory was allocated to the process than was necessary.

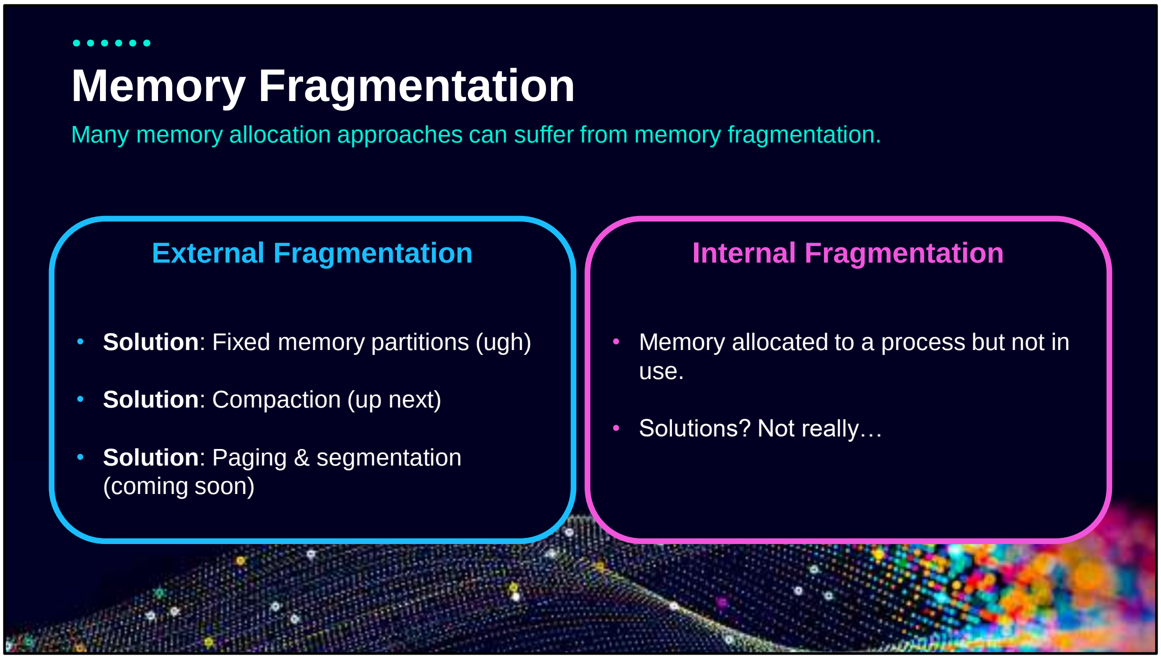
Remember: with Internal Fragmentation, memory is fragmented inside a process.
[CLICK]

An allocation strategy may assign too much space for a process. The unnecessary allocated memory space is a form of internal fragmentation since the current process allocated will not use some of the memory and since it is allocated, no other process will use it either.

The memory between the Heap and the Stack of a process may become fragmented as a process executes.

The Heap, for example, may become fragmented as variables and other structures are allocated and deallocated. Small memory spaces may be available, but are too small to be used, therefore they are a form of internal fragmentation.

[CLICK – Next Slide]



Memory Fragmentation

Many memory allocation approaches can suffer from memory fragmentation.

External Fragmentation

- **Solution**: Fixed memory partitions (ugh)
- **Solution**: Compaction (up next)
- **Solution**: Paging & segmentation (coming soon)

Internal Fragmentation

- Memory allocated to a process but not in use.
- Solutions? Not really…

OFF SCREEN

Different memory allocation approaches can cause memory fragmentation.

But can we fix it? Yes! **[CLICK]**

In the case of external memory fragmentation, we can fix it with Fixed Memory Partitions… But we know there are problems with that too.

We can also do something that is called compaction. Another solution is Paging and

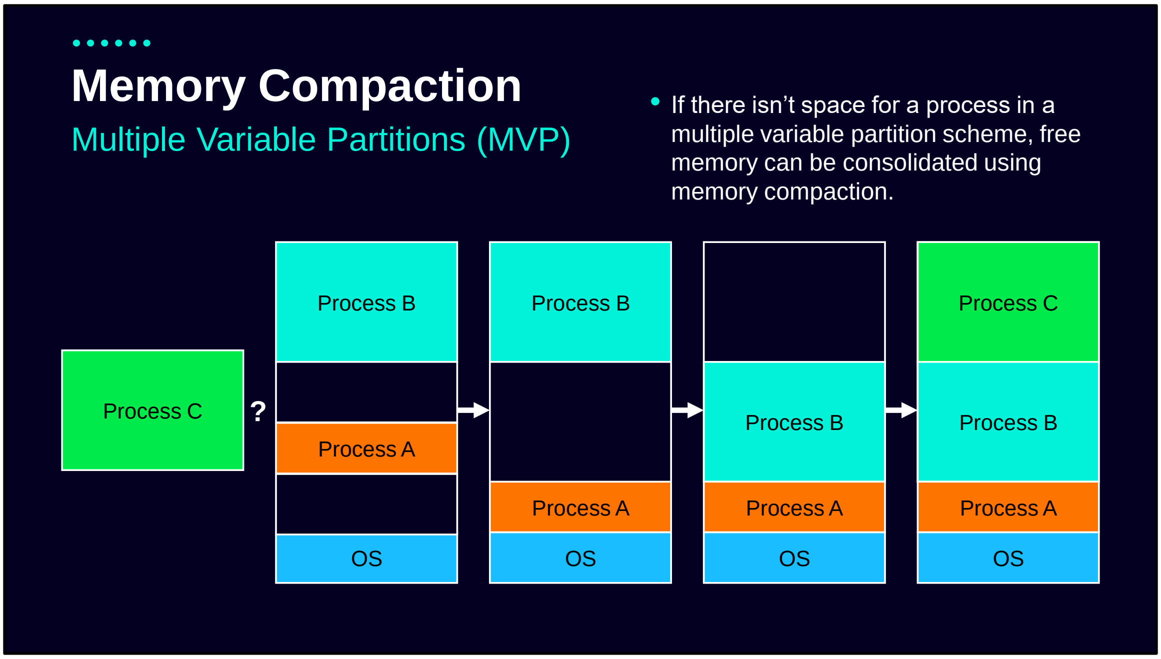
Segmentation. **[CLICK]**

Using multiple variable partitions is a form of avoiding the allocation of excessive memory to a process. So, it is a “fix” for this type of internal fragmentation.

Unfortunately, there aren’t practical solutions for Internal Fragmentation between the Heap and the Stack. Although, compaction has been explored as a solution for this type of Internal fragmentation.

So, let’s look at Compaction next.

[CLICK – Next Slide]



Memory Compaction

Multiple Variable Partitions (MVP)

- If there isn’t space for a process in a multiple variable partition scheme, free memory can be consolidated using memory compaction.

OFF SCREEN

If there isn’t free space for a process in a multiple variable partition scheme, free memory can be consolidated using Memory Compaction. This is equivalent to performing the defragmentation of the memory. [CLICK]

Suppose we have a multiple variable partition allocation strategy, with the configuration as shown. [CLICK]

Currently, there isn't a partition that could hold Process C in memory.
However, we can perform compaction. [CLICK]

With compaction we aim to group all partitions, elimitating empty spaces between them. [CLICK]

Process A, for example, was moved to the bottom of the memory space, right after the OS memory area. [CLICK]

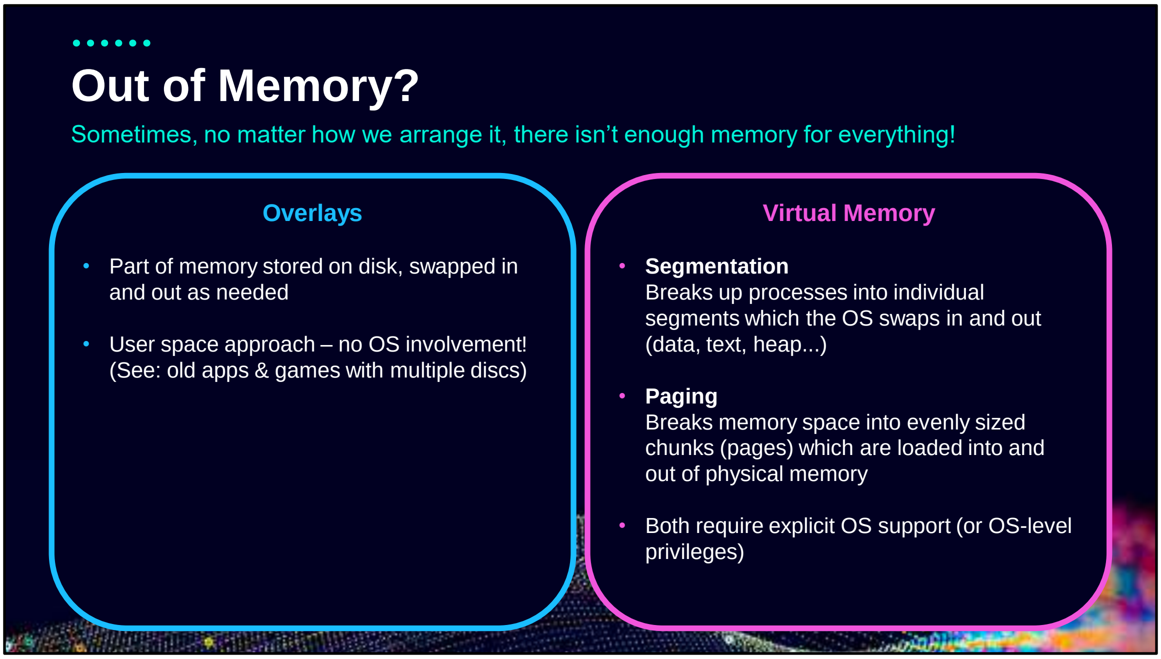
Process B has also been moved [CLICK]

There is now no free memory between the OS and Process A and between processes A and B. Compaction has been executed. [CLICK][CLICK]

Note that now, process C can be loaded. A partition in memory can be allocated that is large enough to hold process C.

Compaction has also been used with disk drives. Running drive defragmentation was something common to recover some performance from a slowing down computers that used magnetic hard disks

[CLICK – Next Slide]



Out of Memory?

Sometimes, no matter how we arrange it, there isn’t enough memory for everything!

Overlays

- Part of memory stored on disk, swapped in and out as needed
- User space approach – no OS involvement! (See: old apps & games with multiple discs)

Virtual Memory

- **Segmentation**
  Breaks up processes into individual segments which the OS swaps in and out (data, text, heap...)
- **Paging**
  Breaks memory space into evenly sized chunks (pages) which are loaded into and out of physical memory
- Both require explicit OS support (or OS-level privileges)

OFF SCREEN

Sometimes, no matter how we arrange it, there isn’t enough memory for everything!

So, we can use “fake memory” approaches in such situations. **[CLICK]**

There are two approaches that virtually extend the memory space that process can use:

Overlays and virtual memory. **[CLICK]**

With Overlays, part of memory is stored on disk, and swapped in and out as needed. The programmer is responsible for defining what stays in memory and what is moved back to disk. **[CLICK]**

Overlays are a user space approach with no OS involvement.

This approach has been used by old apps and games with multiple disks.

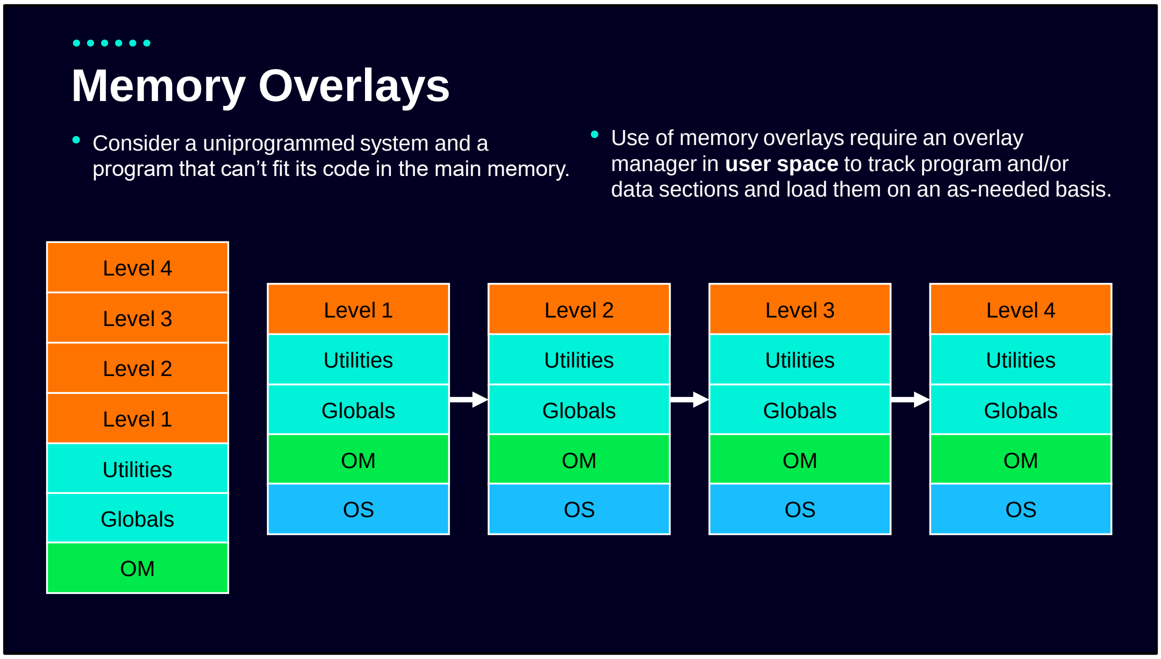
Now, let's look a virtual memory. Virtual memory is divided into Segmentation and Paging. **[CLICK]**

With segmentation, processes are divided into individual segments (such as the text, heap, and stack segments). In this strategy, the OS swaps segments of processes in and out of main memory. **[CLICK]**

Paging, on the other hand, breaks memory space into evenly sized chunks called pages) which are loaded in and out of main memory. **[CLICK]**

Paging and Segmentation require explicit OS support or OS-level privileges.

[CLICK – Next Slide]



Memory Overlays

- Consider a uniprogrammed system and a program that can't fit its code in the main memory.
- Use of memory overlays require an overlay manager in **user space** to track program and/or data sections and load them on an as-needed basis.

Level 4
Level 3
Level 2
Level 1
Utilities
Globals
OM

Level 1
Utilities
Globals
OM
OS

Level 2
Utilities
Globals
OM
OS

Level 3
Utilities
Globals
OM
OS

Level 4
Utilities
Globals
OM
OS

OFF SCREEN

Let's look at an example of Memory Overlays **[CLICK]**

Consider a uniprogrammed system and a program that can't fit its code in the main memory.
A typical example would be a game.
Not so long ago, it was common for games to require multiple disks.
It worked this way: once the player completed a given level, the user had to manually load the next disk into the system to continue playing the game. **[CLICK]**

In this example, let's consider a game with 4 levels.

Since the entire game does not fit in memory, the programmer decided to create overlays with the contents of each level of the game. [CLICK]

So, when the game is first loaded in memory, the Globals, Utilities, and level 1 of the game are loaded. Additionally, an overlay manager is also loaded. That is necessary to control overlays, since this approach is typically not directly supported by the

Operating System. [CLICK]

When the player completes level 1, another level must be loaded. Level 1 is removed from memory [CLICK]

And level 2 is loaded to memory [CLICK]

The same happens to the other levels. [3-Click]

Note that the programmer defines the size and contents of overlays and manages it without direct support from the OS. [CLICK]
In other words, the use of memory overlays require an overlay manager in user space to track program and/or data sections and load them on an as-needed basis

[CLICK – Next Slide]

Thank You For Watching

References

- James, M. (2015, August 25). *Screenshot of article* [Online Image]. I-Programmer. https://www.i-programmer.info/news/82-heritage/8920-twenty-years-ago-window-95-was-released.html