6.5. Virtual Memory

Term	Description
Virtual Memory	A logical memory space large enough to hold an entire application or multiple applications.
Virtual Address	A virtual address is the address generated by the CPU. The logical or program address that the process uses. Whenever the CPU generates an address, it always in terms of the virtual address space.
Physical Address	The real address in main store. Main store is also called main memory. Main memory is also called physical memory.
Mapping	The mechanism by which virtual addresses are translated into physical addresses.
Page Frames	The equal-size chunks or blocks into which main memory (physical memory) is divided.
Pages	The blocks into which virtual memory (the logical address space) is divided, each equal in size to a page frame. Virtual pages are stored on disk until needed.
Paging	The process of copying a virtual page from disk to a page frame in main memory.
Fragmentation	In the process allocating and reclaiming storage logically related data become physically separated and split into small pieces that are stored in a non-sequential locations.
Page Fault	An event that occurs when a requested page is not in main memory and must be copied into memory from disk.
Page File	The page file is the implementation of the virtual memory on hard disk, stored as pages.

- Main memory and virtual memory are divided into equal sized pages.
- The entire address space required by a process need not be in memory at once. Some parts can be on disk, while others are in main memory.
- Further, the pages allocated to a process do not need to be stored contiguously--either on disk or in memory.
- In this way, only the needed pages are in memory at any time, the unnecessary pages are in slower disk storage.

6.5.1 Paging

- Allocate physical memory to processes in fixed-size blocks called pages and keep track of where the various pages of the process reside by recording information in a **page table**.
- The page table has N rows, where N is the number of virtual pages in the process.
- Each row of the page table contains a **valid bit.** The valid bit is set to one (**1**) if the page is in main store, otherwise it is set to zero (**0**).

Term	Description
Page Field	The most significant part of the virtual address that specifies the page in which the address lies.
Offset Field	The least significant part of the virtual address that specifies the offset in the page to the desired byte.

Address Translation:

1. Extract the page number from the virtual address.
2. Extract the offset from the virtual address.
3. Translate the page number into a physical page frame number by accessing the page table.
    3.1. Look up the page number in the page table (using the virtual page number as an index.
    3.2 Check the valid for that page.
        3.2.1. If the valid bit = 0, the system generates a page fault and the operating system must intervene to:
            3.2.1.1. Locate the desired page on disk.
            3.2.1.2. Find a free page frame (this may necessitate removing a page from memory and copying it back to disk if there are no free page frames.
            3.2.1.3. Copy the desired page into the free page frame in main store.
            3.2.1.4. Update the page table. (The virtual page just brought in must have its frame number and valid bit in the page table modified. If a page was removed, its valid bit must be set to zero (0).
            3.2.1.5. Resume execution of the process causing the page fault, continuing to step 3.2.2.
        3.2.2. If the valid bit = 1, the page is in memory.
            3.2.2.1. Replace the virtual page number with the actual frame number.
            3.2.2.2. Access the desired data at the offset in the physical page frame by adding the offset to the frame number for the given virtual page.

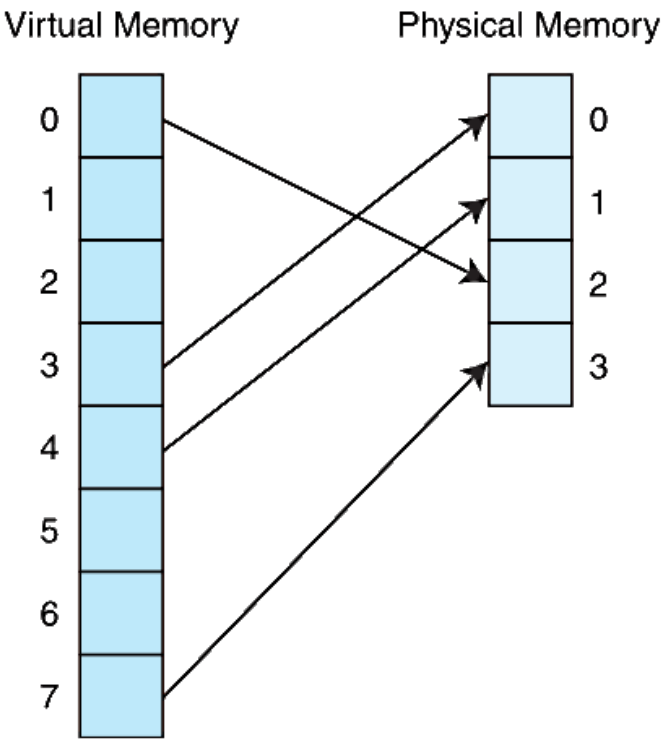


Figure 6.18 – left side

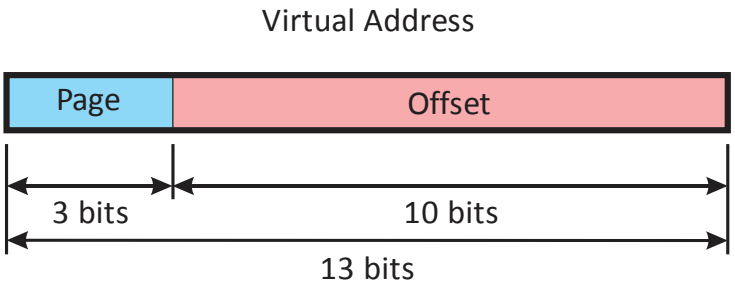
Page Page Table

	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Figure 6.18 – right side

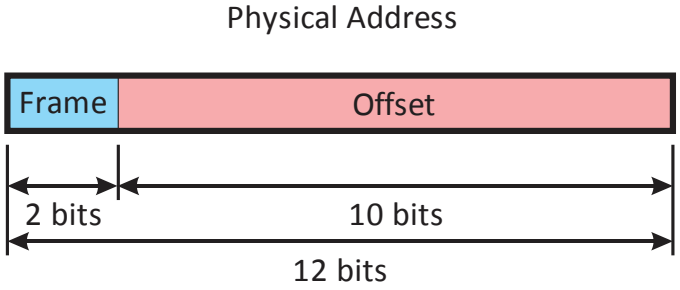
Example:

- Virtual address space is 8K bytes ($2^{13} = 8192$ bytes), making the **virtual address 13 bits**.
- Define the page size to be 1K bytes ($2^{10} = 1024$ bytes), making the page **offset 10 bits**



Format for a 13-Bit Virtual Address with $2^{13} = 8192$ bytes

- Main store occupies $2^{12} = 4096$ bytes making a **Physical** (or real) **Address 12 bits**.
- The page size remains the same, 1K bytes ($2^{10} = 1024$ bytes), making the page **offset 10 bits**
- Given a 12-bit Physical Address and an offset of 10 bits, makes the **Frame** Address **2 bits**.
- $2^3 = 8$ Virtual pages are mapped to $2^2 = 4$ physical (or real) page frames.



Format for 12-Bit Physical Address with $2^{12} = 4096$ bytes

Page Table

Page	Frame	Valid Bit
0	-	0
1	3	1
2	0	1
3	-	0
4	-	0
5	1	1
6	2	1
7	-	0

Example Page Table

- Find the Physical Address that corresponds to the Virtual address **0x1553**
    1. Convert the Virtual Address to binary separating the Page and Offset portions of the address.
       1 01 01 0101 0011
       Page= 101=5
       Offset= 01 0101 0111=0x153

    2. Find the Frame in which the Page is mapped by employing the Page as an index into the Page Table and extracting the Frame stored in the entry given by the Page Number.
       Page 5 is mapped to Frame 1.

    3. Find the Physical Address by replacing the 3-bit Virtual Page Number with the 2-bit Physical Frame Number
       Physical Address= Frame+Offset=01 + 01 0101 0111
       Physical Address = 0101 0101 0111=0x553

- Find the Physical Address that corresponds to the Virtual address **0x1004**
    1. Convert the Virtual Address to binary separating the Page and Offset portions of the address.
       1 00 00 0000 0100
       Page= 100=4
       Offset= 00 0000 0100=0x004

    2. Find the Frame in which the Page is mapped by employing the Page as an index into the Page Table and extracting the Frame stored in the entry given by the Page Number.
       Page 4 is not mapped.

A page fault occurs.
Since all the Physical Frames are occupied, one must be replaced.
Select a Frame to be replaced.
We have no information on which to make a selection.
Select a random Frame – Frame 0
Mark the Valid Bit in the Page Table at Index 2 0 indicating that that Page Frame is no longer valid.
Replace the entry in the Page Table at index position 4.
Assign the Frame Number to 0.
Assign the Valid Bit to 1

3. Find the Physical Address by replacing the 3-bit Virtual Page Number with the 2-bit Physical Frame Number
Physical Address= Frame+Offset=00 +00 0000 0100=
Physical Address = 0000 0000 0100=0x004

6.5.2 Effective Access Time Using Paging

- To access physical memory, two (2) accesses are required, one to translate the page address to a frame address, and one to access the actual memory desired.

Symbol	Meaning
EAT	Effective Access Time – the time required to access memory using a virtual memory system and paging.
T_{ms}	Time to access main store or main memory. A typical value in 2014 is 200ns.
P_{ms}	Probability that the desired data are in main store. A typical value is 0.99
T_d	Time to access hard disk. A typical value is 10ms
P_d	Probability that the desired data are on the hard disk. $P_d = 1 - P_{ms}$

$$EAT = P_{ms} \times T_{ms} + (1 - P_{ms}) \times P_d$$

Assuming typical values and two memory references:

$$EAT = 0.99 \times (200ns + 200ns) + 0.01 \times 10ms$$
$$EAT = 100.396 \times 10^{-6} = 100.396\mu s$$

- Even if we had no page faults, the EAT would be 400ns because memory is always read twice: First to access the page table, and second to load the page from memory.
- Because page tables are read constantly, it makes sense to keep them in a special cache called a *translation look-aside buffer* (TLB).
- TLBs are a special associative cache that stores the mapping of virtual pages to physical pages.

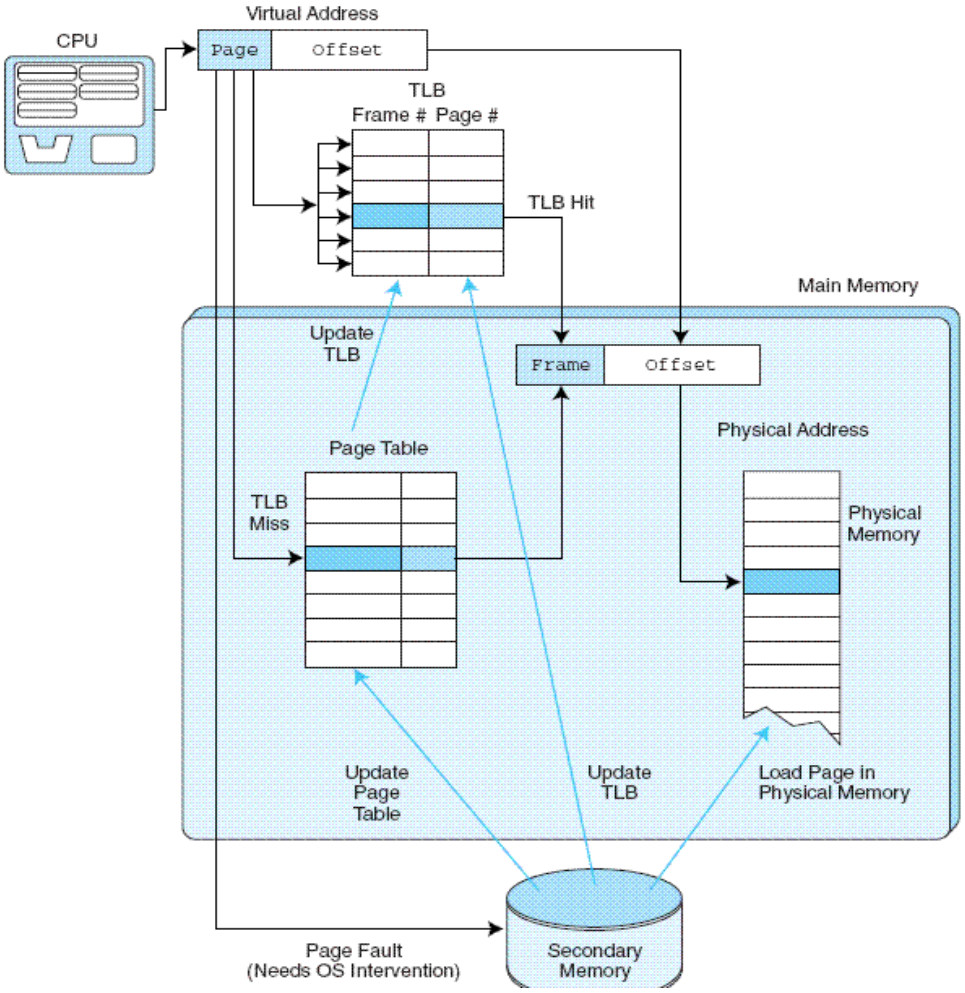


Figure 6.27 Using the TLB

1. Extract the page number from the virtual address.
2. Extract the offset from the virtual address.
3. Search for the virtual page number in the TLB.
4. If the (virtual page #, page frame #) pair is found in the TLB, add the offset to the physical frame number and access the memory location.
5. If there is a TLB miss, go to the page table to get the necessary frame number.

   If the page is in memory, use the corresponding frame number and add the offset to yield the physical address.
6. If the page is not in main memory, generate a page fault and restart the access when the page fault is complete.

6.5.3 Putting It All Together: Using Cache, TLBs, and Paging

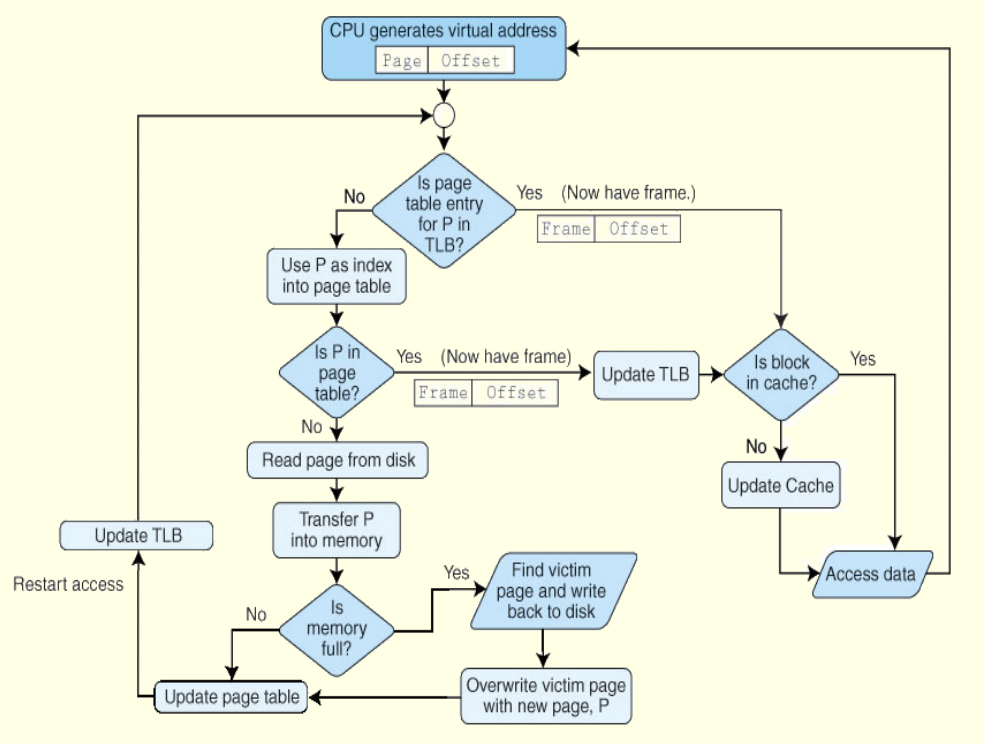


Figure 6.28 Putting It All Together: the TLB, Page Table, Cache, and Main Memory

6.5.4 Advantages and Disadvantages of Paging and Virtual Memory

Disadvantages:

- Adds an extra memory reference when accessing data.
- Even with a TLB, the mapping of the virtual page number to a real memory page frame still incurs and translation overhead.
- Memory is required to store the page table. For example, imagine a 32-bit virtual address, 1GB of physical memory – a 30-bit physical memory address, and a 4096 byte page – a 12-bit offset. A virtual memory address has a 20-bit page field and a 12-bit offset field. The page table must have 2^{20} entries where each entry stores 18-bit frame numbers and 1-bit valid flags. Each entry must occupy 3 bytes. Hence the page table occupies $3 \times 2^{20} = 3{,}145{,}728$ bytes – not insignificant penalty.
- The TLB is special and costly to implement.

Advantages

- Programs are no longer restricted by the amount of physical memory that is available.
- Programmers are not required to manage memory. *When programmers manage memory they manage it badly*.
- Virtual memory permits more programs to execute concurrently, thus, increasing CPU utilization and system throughput.

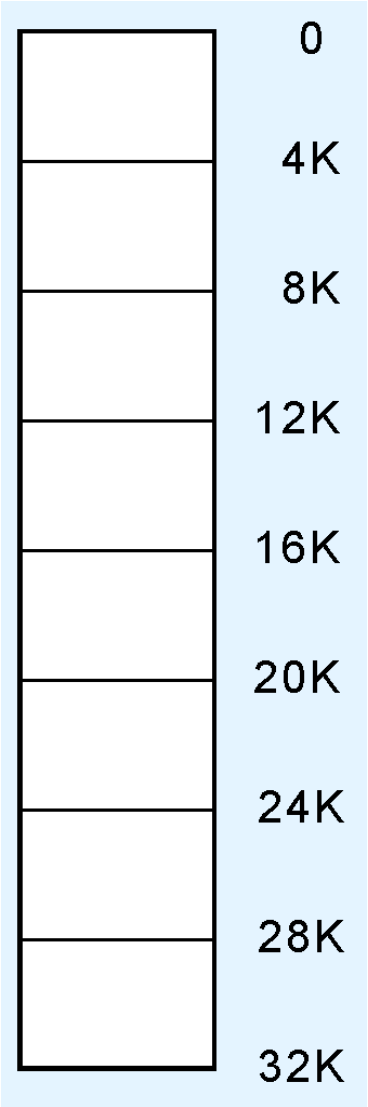
- A paging system enables a rudimentary protection system. With a few bits, protection can be specified as "this page belongs to user X, no others are permitted access" or "this page belongs to user X but you can read it."

6.5.5 Segmentation

- Another approach to virtual memory is the use of ***segmentation***.
- Instead of dividing memory into equal-sized pages, virtual address space is divided into variable-length **segments**, often under the control of the programmer.
- A segment is located through its entry in a segment table, which contains the segment's memory location and a bounds limit that indicates its size.
- After a page fault, the operating system searches for a location in memory large enough to hold the segment that is retrieved from disk.
- Both paging and segmentation can cause fragmentation.
- Paging is subject to *internal* fragmentation because a process may not need the entire range of addresses contained within the page. Thus, there may be many pages containing unused fragments of memory.
- Segmentation is subject to *external* fragmentation, which occurs when contiguous chunks of memory become broken up as segments are allocated and deallocated over time.

6.5.5.1 Internal Fragmentation

- Internal fragmentation only occurs in paging systems.



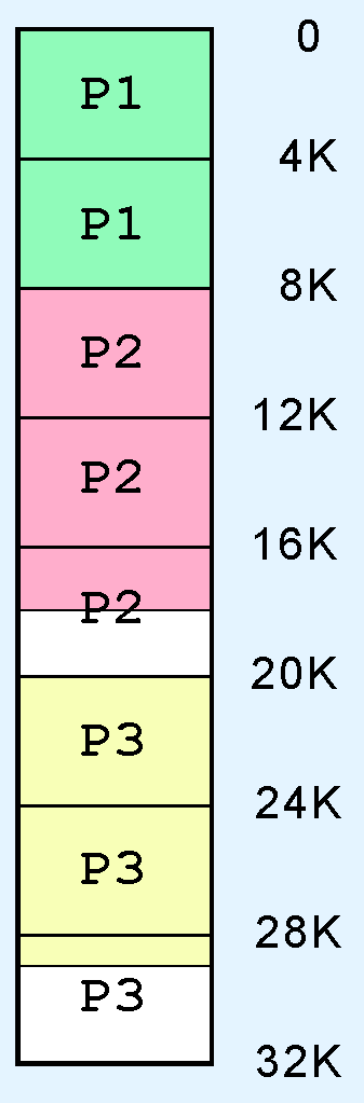
Main Store for a Small Computer

- The 32K memory is divided into 8 page frames of 4K each.
- A schematic of this configuration is shown above.
- The numbers at the right are memory frame addresses.

Process Name	Memory Needed
P1	8K
P2	10K
P3	9K
P4	4K

Memory Resources Needed for Four Processes

- Suppose there are four processes waiting to be loaded into the system with memory requirements as shown in the table.
- We observe that these processes require 31K of memory.



P1	8K
P2	10K
P3	9K
P4	4K

Memory Allocation for Three of the Four Processes

- When the first three processes are loaded, memory looks like this:
- All of the frames are occupied by three of the processes.
- Despite the fact that there are enough free bytes in memory to load the fourth process, P4 has to wait for one of the other three to terminate, because there are no unallocated frames.
- This is an example of *internal fragmentation.*

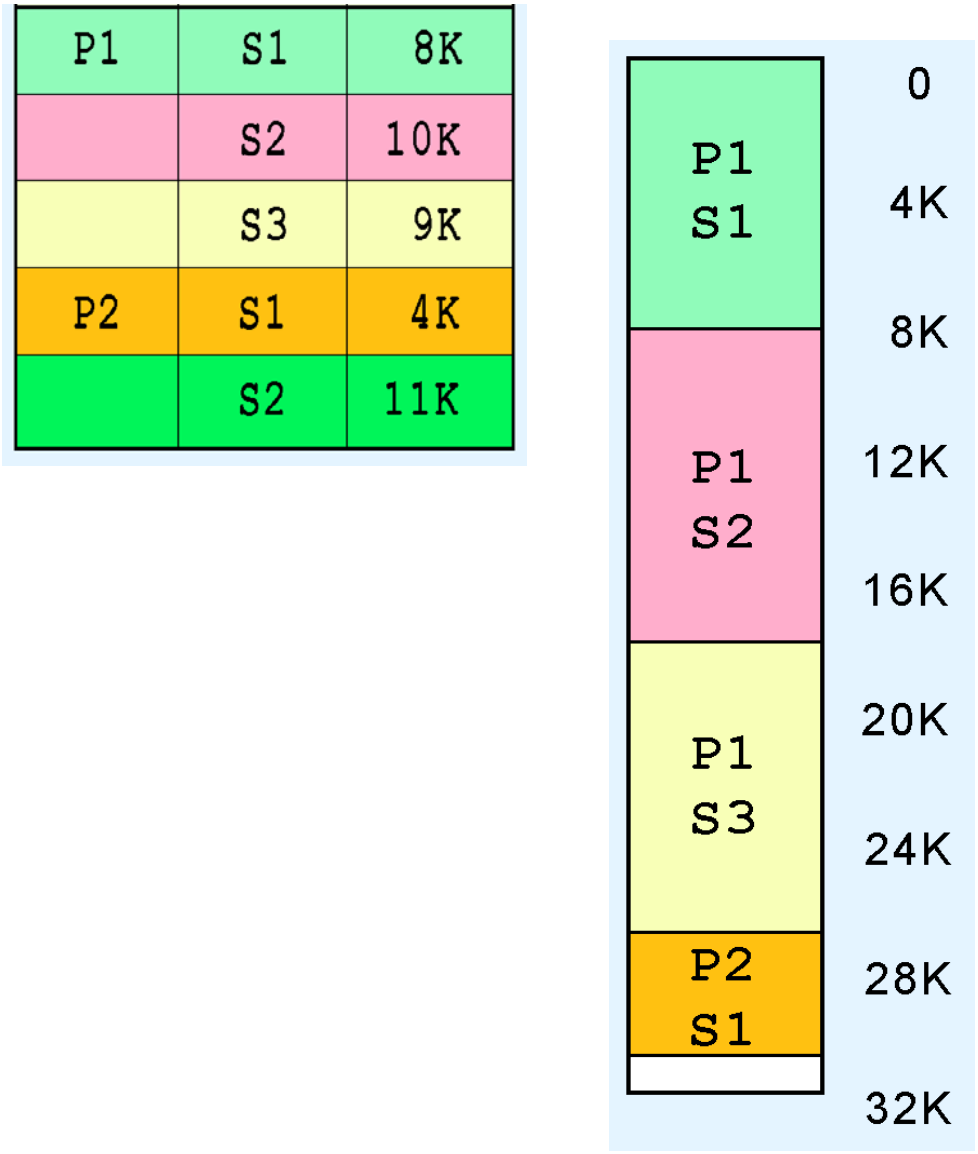
6.5.5.1 External Fragmentation

- External fragmentation only occurs in segmented systems

Process Name	Segment	Memory Needed
P1	S1	8K
	S2	10K
	S3	9K
P2	S1	4K
	S2	11K

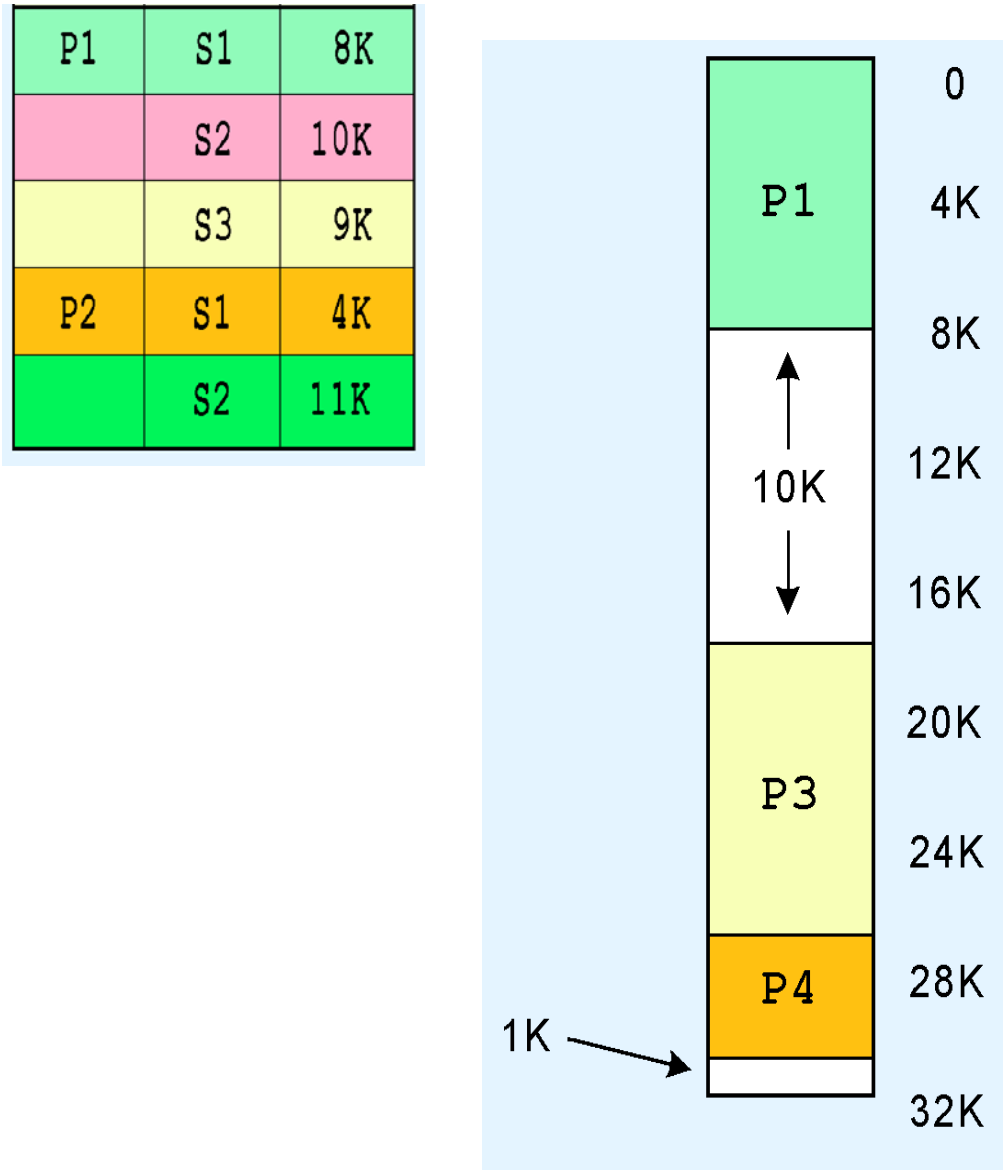
Memory Resources Needed for Two Processes and Five Segments

- Suppose that instead of frames, our 32K system uses segmentation.
- The memory segments of two processes is shown in the table at the right.
- The segments can be allocated anywhere in memory.



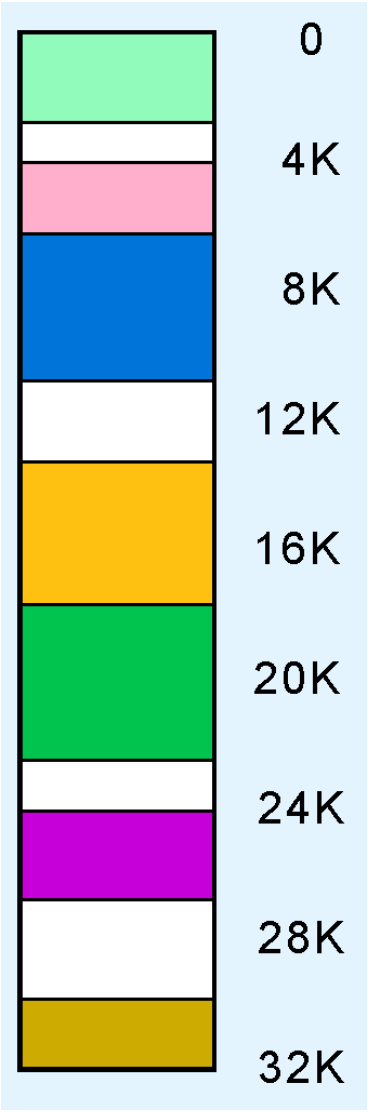
Memory Allocation for Two Processes

- All of the segments of P1 and one of the segments of P2 are loaded as shown at the right.
- Segment S2 of process P2 requires 11K of memory, and there is only 1K free, so it waits.



External Fragmentation

- Eventually, Segment 2 of Process 1 is no longer needed, so it is unloaded giving 11K of free memory.
- But Segment 2 of Process 2 cannot be loaded because the free memory is not contiguous.
- Over time, the problem gets worse, resulting in small unusable blocks scattered throughout physical memory.
- This is an example of *external fragmentation*.
- Eventually, this memory is recovered through compaction, and the process starts over.



External Fragmentation

- Large page tables are cumbersome and slow, but with its uniform memory mapping, page operations are fast. Segmentation allows fast access to the segment table, but segment loading is labor-intensive.
- Paging and segmentation can be combined to take advantage of the best features of both by assigning fixed-size pages within variable-sized segments.
- Each segment has a page table. This means that a memory address will have three fields, one for the segment, and another for the page, and a third for the offset.