Relocation and Addresses

- As we have seen, processes may be swapped in and out of main memory and thus may occupy different partitions in the memory. Also, when compaction is used, processes are shifted while in memory. Therefore, memory locations (of instructions and data) are not fixed during execution.

- There is a distinction made between several different types of addresses.

Relocation and Addresses

Logical

- Reference to a memory location independent of the current assignment of data to memory.

  Translation must be made to the physical address.

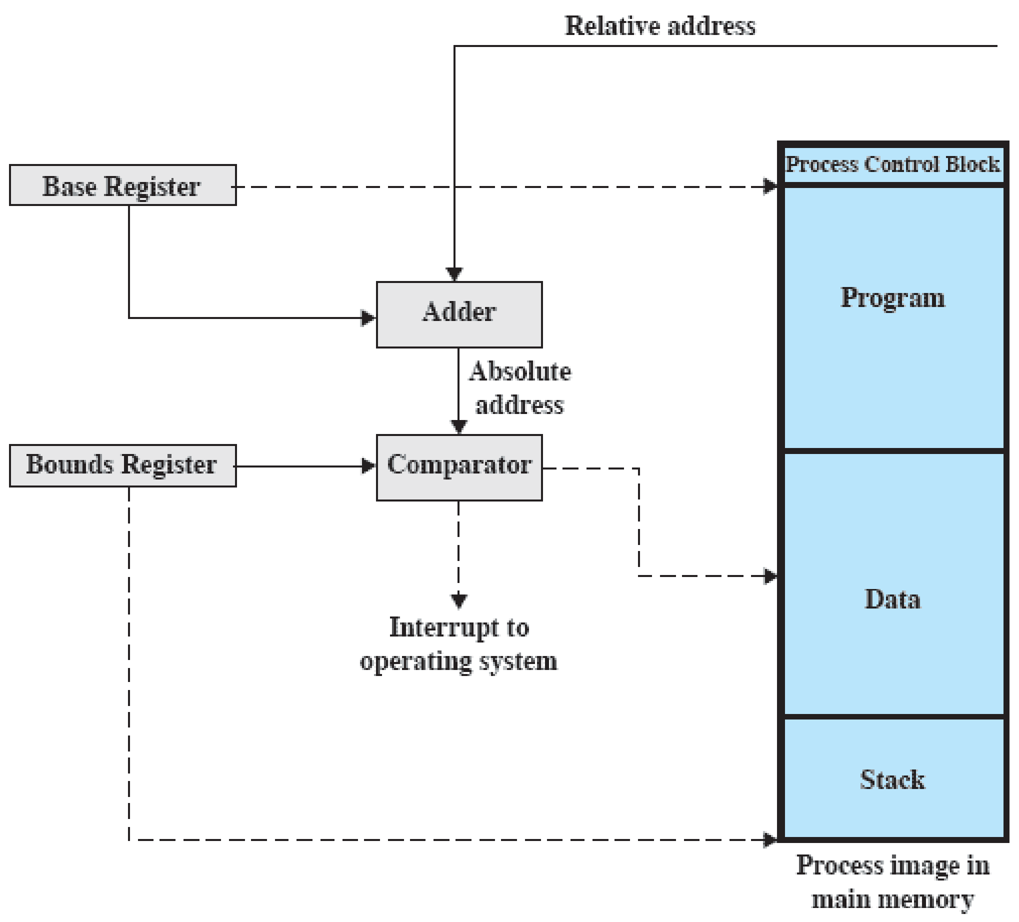
Relative

- Address expressed as a location relative to some known point.

Physical

- Absolute address or actual location in main memory.

Relative Addressing



Relative Addressing (cont.)

- When a process is assigned to the Running state - a special processor register (called the **base** register) is loaded with the starting address in main memory of the program.
- There is also a **bounds** register that indicates the ending location of the program.
- During the course of execution of the process, relative addresses are encountered.
- **These include the contents of the instruction register, instruction addresses that occur in branch and call instructions, and data addresses that occur in load and store instructions.**

Relative Addressing (cont.)

Each relative address goes through two steps

1. The value in the base register is added to the relative address to produce an absolute address.

2. The resulting address is compared to the value in the bounds register.

If the address is within bounds, then the instruction execution may proceed. Otherwise, an interrupt is generated to the operating system.

This allows programs to be swapped in and out of memory during the course of execution. It also provides a measure of protection. Each process image is isolated by the contents of the base and bounds registers and safe from unwanted accesses by other processes.

Paging

- **Fixed-sized partitioning** (equal + unequal partitions) is inefficient in the use of memory as it results in internal fragmentation
- **Variable-size partitioning** results in external fragmentation.
- In Paging, main memory is partitioned into equal fixed-size chunks that are relatively small – **frames**.
- **Each process is also divided into small fixed-size chunks of the same size known as pages.**
- Each page is assigned to a frame.
- Frames do not have to be contiguous.

Paging (cont.)

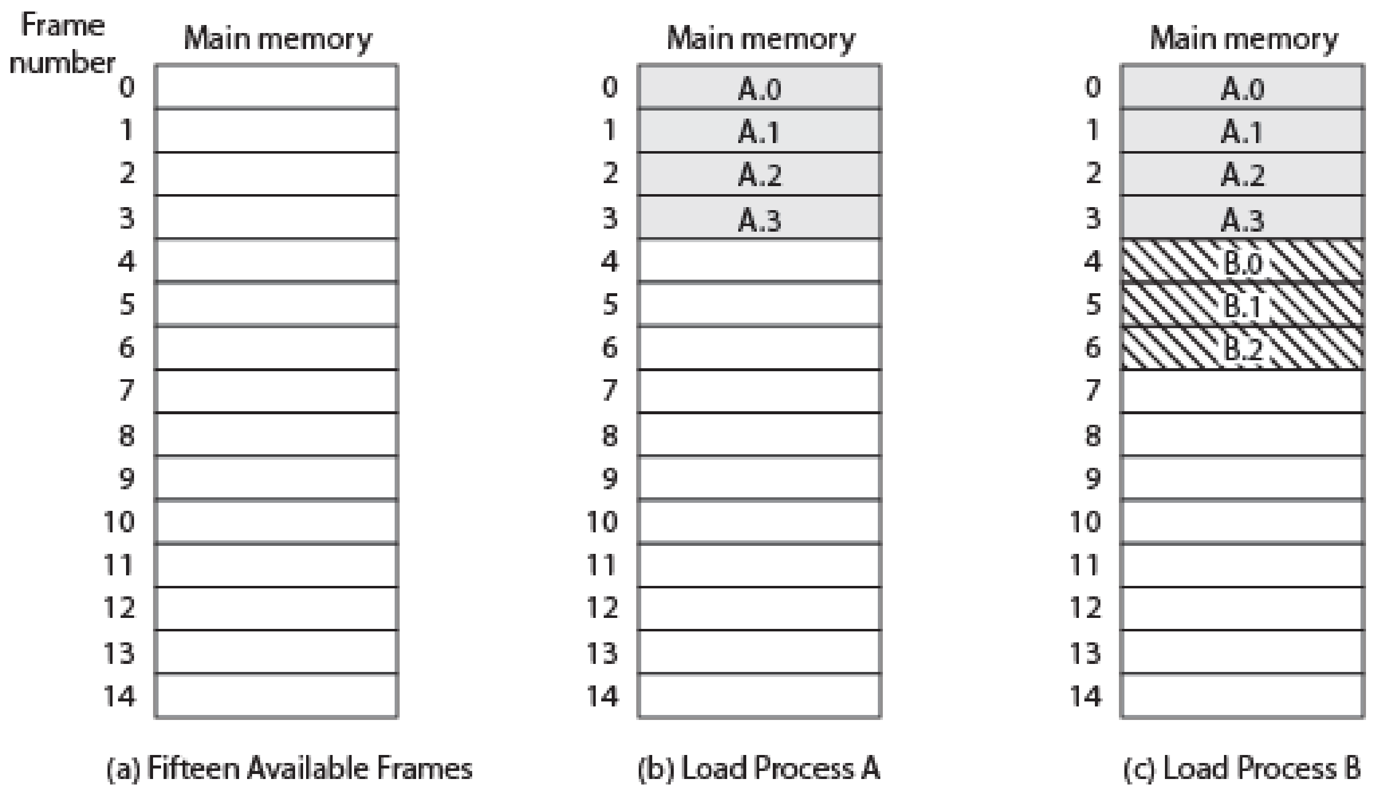
- There are four processes A, B, C and D each requiring space in memory.
- A list of free frames is maintained by the OS. In the diagram below, process A consists of four pages.
- The OS finds four free frames and loads the four pages of process A into the four frames.
- **Process B, consisting of three pages, and process C, consisting of four pages, are subsequently loaded.**
- **Process B is suspended and is swapped out of main memory**.

Paging (cont.)

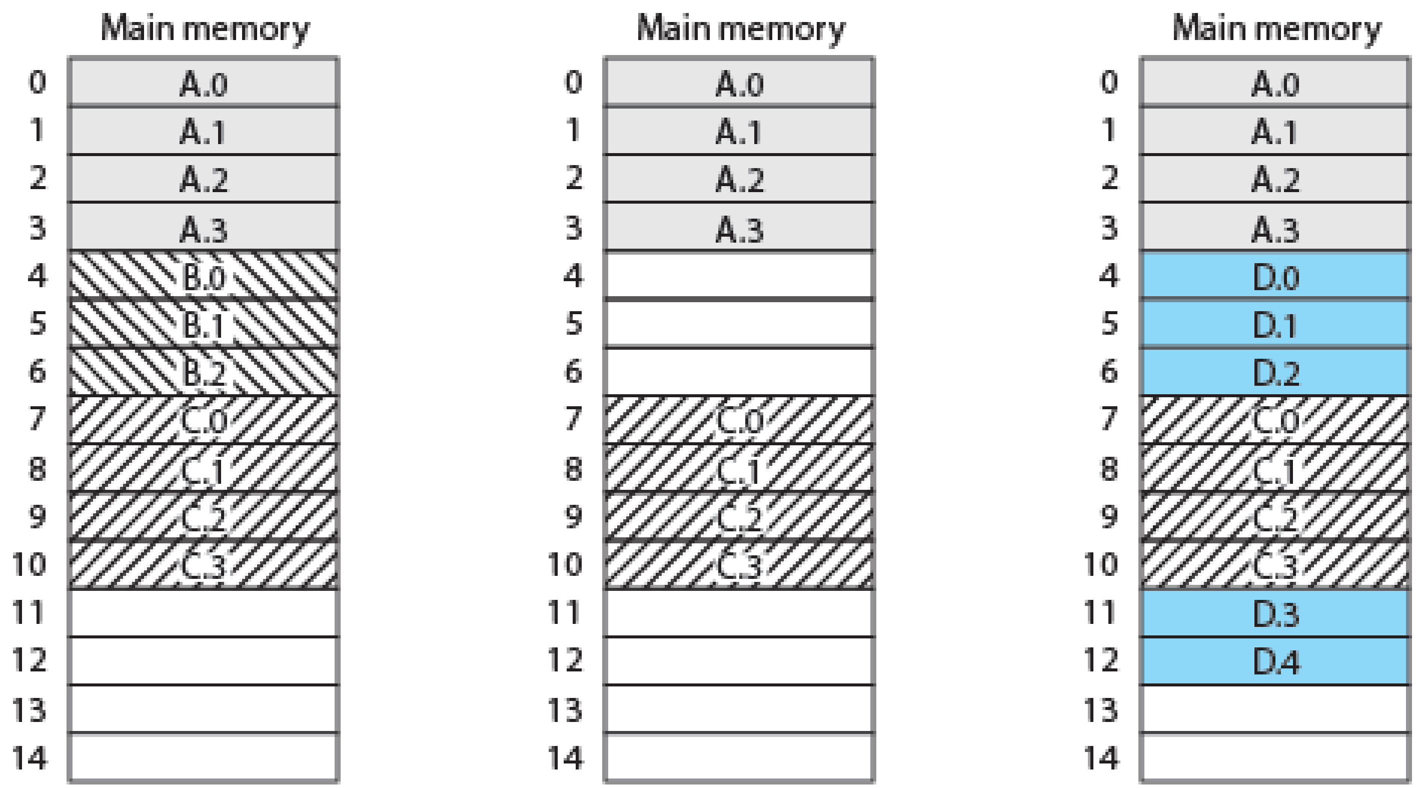
- The OS needs to bring in a new process, process D, which consists of five pages. D is not stored in contiguous frames but a new method is required to evaluate its address (the base register address is not sufficient as the chunks of the process are not in order).

- The operating system needs to maintain a **page table** for each process – a means by which the logical address can be translated to a physical address. The page table needs to hold the frame address for each page.

Loading Pages

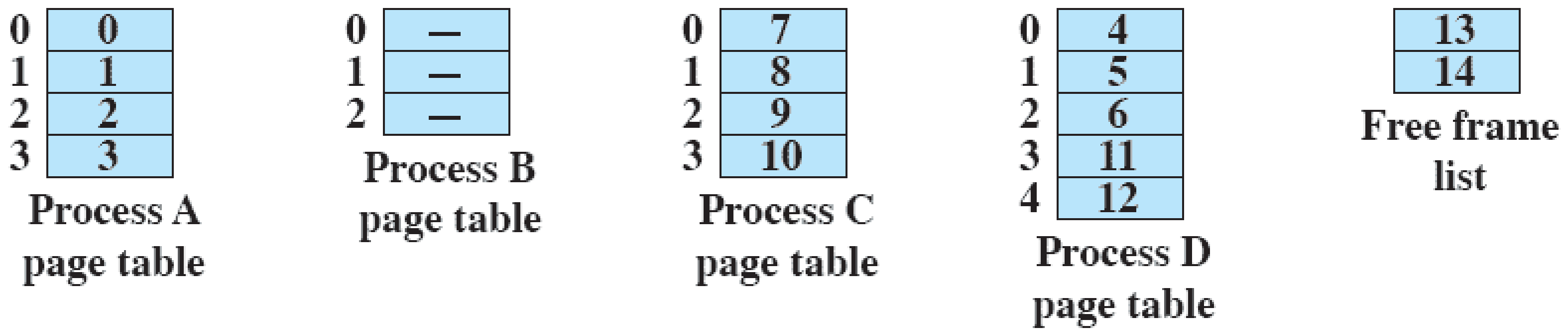


Loading Pages (cont.)



(d) Load Process C (e) Swap out B (f) Load Process D

Loading Pages (cont.)



0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D page table

13
14

Free frame list

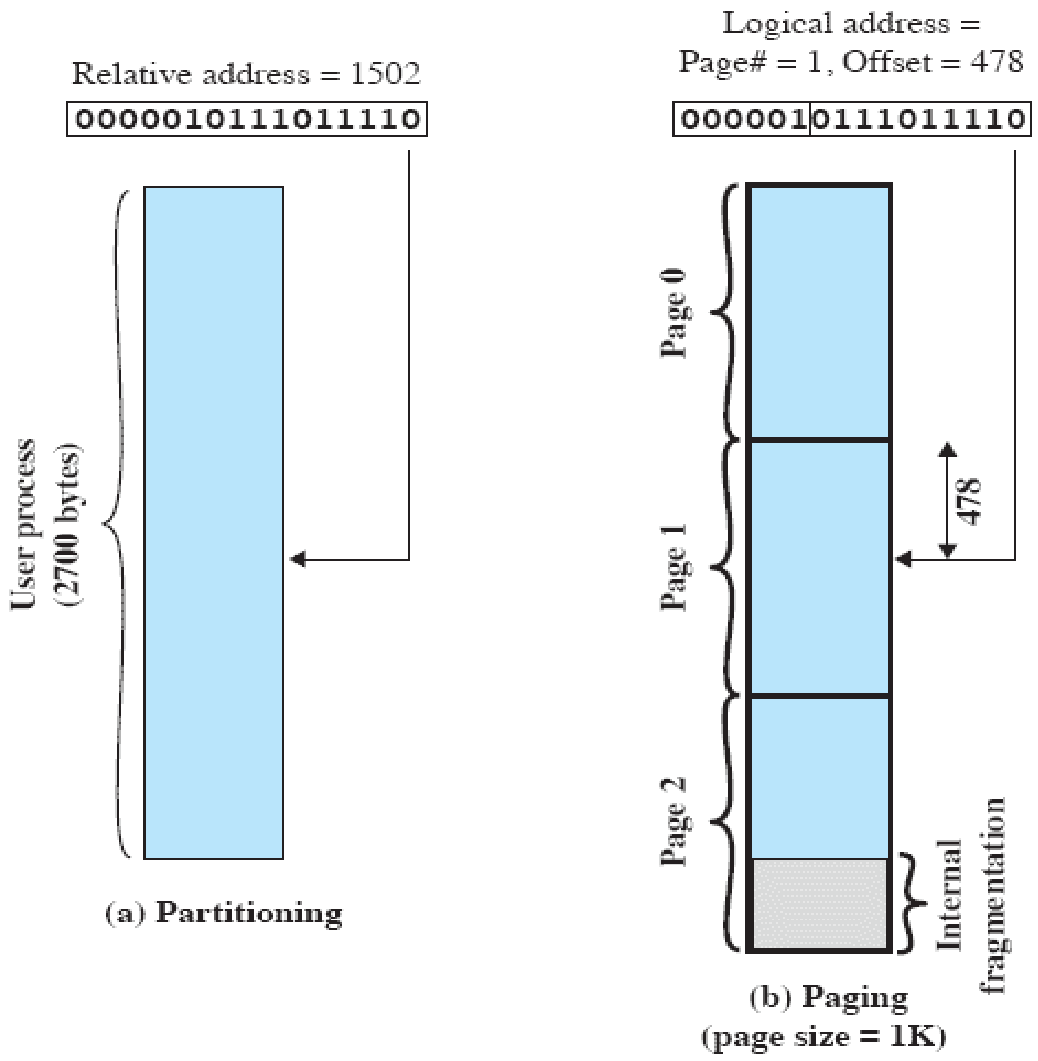
The page table contains one entry for each page of the process, so that the table is easily indexed by the page number (starting at page 0).

Each page table entry contains the number of the frame in main memory that holds the corresponding page.

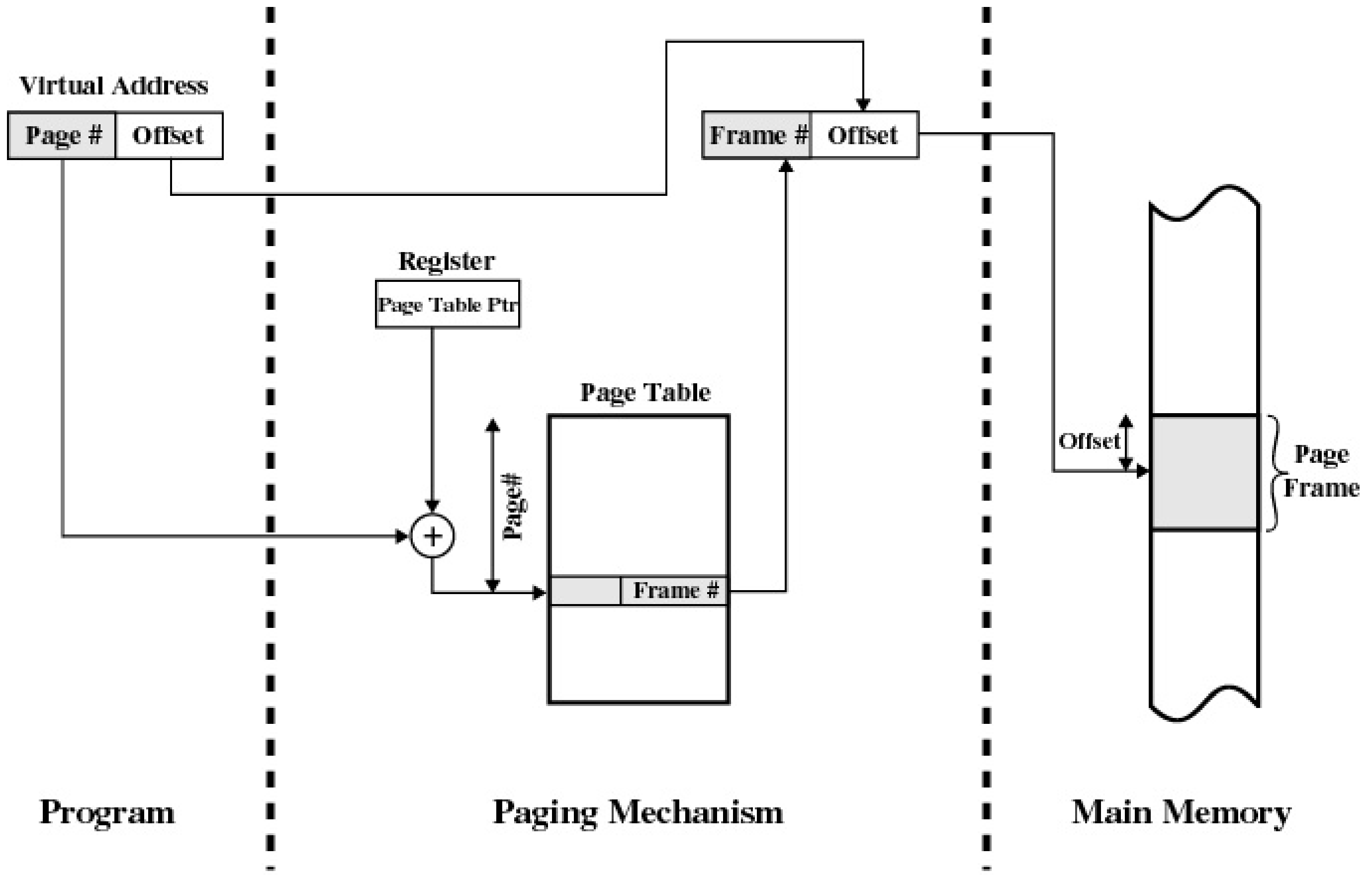
In addition, the OS maintains a single free-frame list of all the frames in main memory that are currently unoccupied and available for pages.

Simple paging is similar to fixed partitioning. The differences are that the partitions are small; a program may occupy more than one partition; and these partitions need not be contiguous.

Logical to Physical



Paging



Virtual Address
Page #
Offset
Frame #
Offset
Register
Page Table Ptr
Page Table
Page#
Frame #
Offset
Page Frame
Program
Paging Mechanism
Main Memory

Logical to Physical

- In this example, 16-bit addresses are used, and the page size is 1K i.e. 2^{10} = 1,024 bytes. The relative address 1502, in binary form, is 0000010111011110.

- With a page size of 1K, an offset field of 10 bits is needed, leaving 6 bits for the page number.

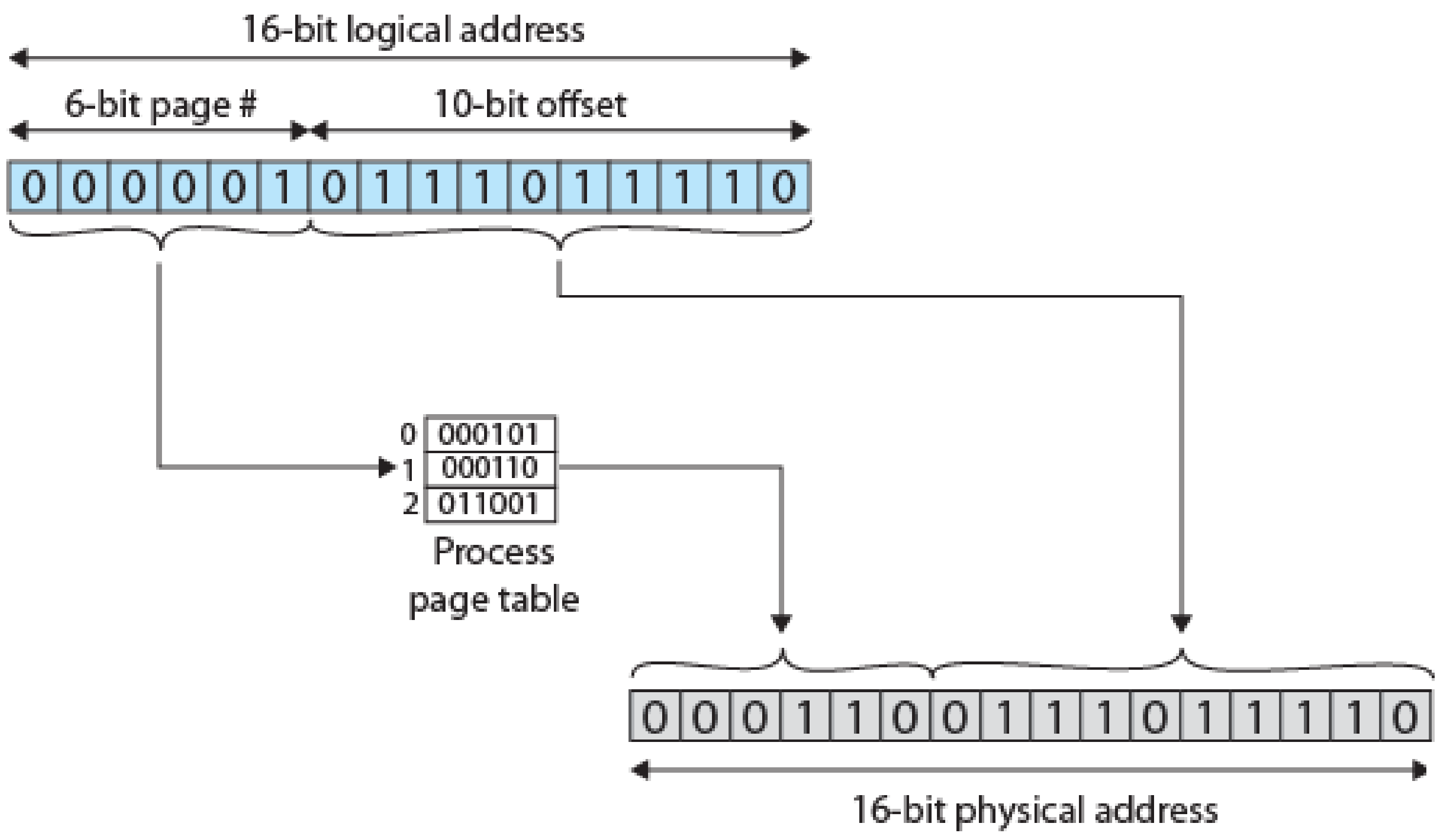
- Thus a program can consist of a maximum of 2^6 = 64 pages of 1K bytes each

- **The relative address 1502 corresponds to an offset of 478 (0111011110) on page 1 (000001), which yields the same 16-bit number, 0000010111011110.**

Paging Example

Convert the logical address 0000010111011110, which is page number 1, offset 478. Suppose that this page is residing in main memory frame 6 binary 000110. Then the physical address is frame number 6, offset 478 0001100111011110.

Paging Example (cont.)



Paging Example 2

Convert the following logical address to a physical address given the page table.

Logical address: 0011001100110110

0	1011
1	1100
2	1011
3	1001
4	1010

Paging Example 2 (cont.)

Number of bits:

page 4 offset 12

page no. 0011 offset: 001100110110

Index page 3: frame = 1001

Physical Address: 1001001100110110

Paging Exercise

Convert the following logical address to a physical address given the page table.

Logical address: 0001001100110111

0	**10110**
1	**11001**
2	**10111**
3	**10010**
4	**11010**

Memory Management (cont.)

Segmentation

Segmentation is similar to paging in some respects.

- The program and its associated data are divided into a number of **segments.**
- It is not required that all segments of all programs be of the same length, although there is a maximum segment length.
- A logical address using segmentation consists of two parts - the **segment number** and an **offset** – to calculate physical address.
- It also consists of a segment table for each process

Virtual Memory

- **Virtual memory** is a common feature of most operating systems on desktop & laptop computers. It has become so because it provides a big benefit for users at a very low cost.

- Computers have a fixed amount of RAM available for the CPU to use. Unfortunately, that amount of RAM is not usually enough to run all of the programs that most users expect to run at once.

- For example, if you load the operating system, an e-mail program, a Web browser, word processor, etc. into RAM simultaneously there may not be enough space to hold it all. If there were no virtual memory it would not be possible to hold all applications concurrently.

Virtual Memory (cont.)

- With virtual memory, what the computer can do is look at RAM for areas that have not been used recently and copy them onto the hard disk. This frees up space in RAM to load the new application / data. This makes the computer feel like it has unlimited RAM space and because hard disk space is so much cheaper than RAM chips, it has an economic advantage.

- An obvious disadvantage is that a machine that depends too much on virtual memory will be substantially slower.

Virtual Memory (cont.)

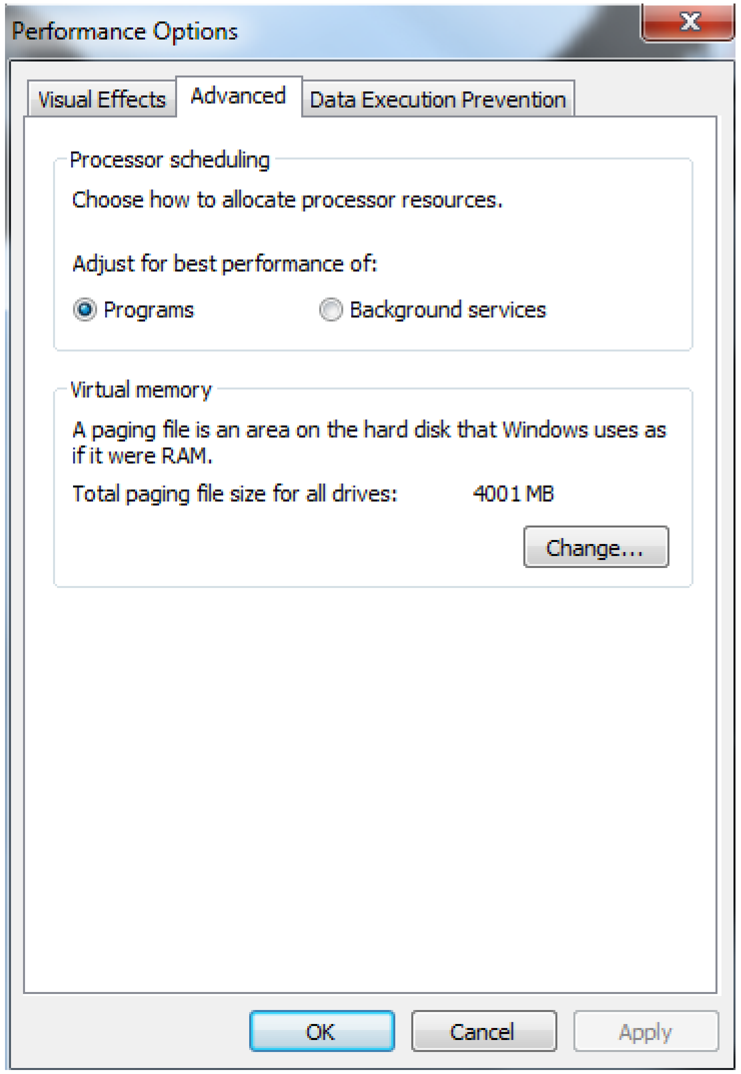
- The read/write speed of a hard drive is much slower than RAM, and the technology of a hard drive is not geared toward accessing small pieces of data at a time. If your system has to rely too heavily on virtual memory, you will notice a significant performance drop. The key is to have enough RAM to handle everything you tend to work on simultaneously then, the only time you "feel" the slowness of virtual memory is when there's a slight pause when you're changing tasks.

- When this is not the case, the operating system has to constantly swap data back and forth between RAM and the hard disk. This is called **thrashing**, and it can make your computer feel incredibly slow.

VM in Windows

- The area of the hard disk that stores the RAM image is called a **page file**.
- It has an intelligent **virtual memory manager** that uses a default setting to help Windows allocate hard drive space for virtual memory as needed. For most circumstances, this should be sufficient, but it can be manually configured, especially if there is more than one physical hard drive or speed-critical applications.
- In Windows, to look at this, select **System** from the control panel, select **Advanced** followed by **Performance**.

VM on Windows



Processor scheduling

Choose how to allocate processor resources.

Adjust for best performance of:

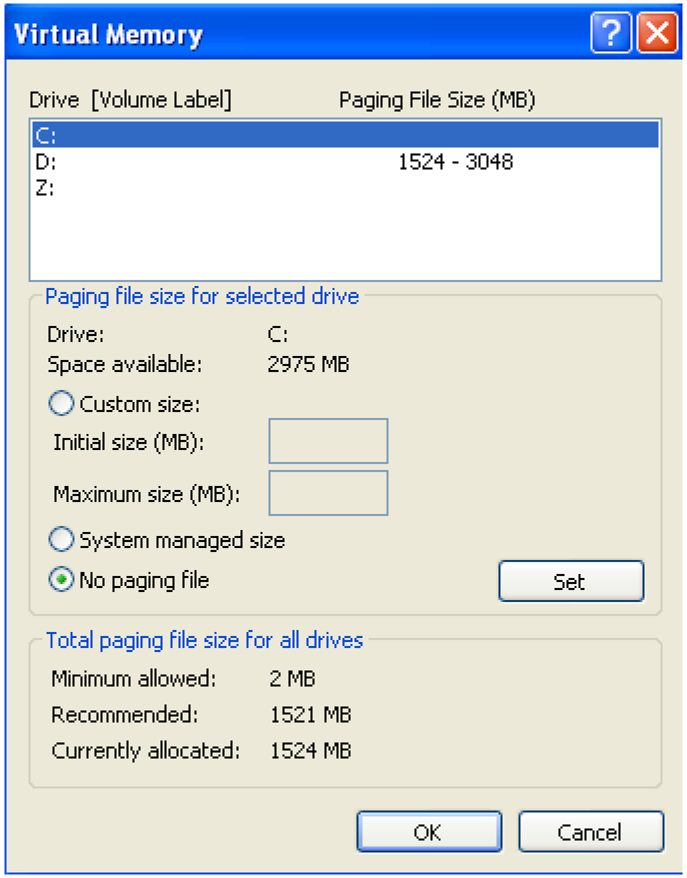
[✓ Programs] Background services

Virtual memory

A paging file is an area on the hard disk that Windows uses as if it were RAM.

Total paging file size for all drives: 4001 MB

To configure VM on Windows



Virtual Memory

Drive [Volume Label]	Paging File Size (MB)
C:	
D:	1524 - 3048
Z:	

Paging file size for selected drive

Drive:	C:
Space available:	2975 MB

Custom size:

Initial size (MB):	
Maximum size (MB):	

System managed size

[✓ No paging file]

Set

Total paging file size for all drives

Minimum allowed:	2 MB
Recommended:	1521 MB
Currently allocated:	1524 MB

OK Cancel

Linux Virtual Memory

The kernel will write the contents of a currently unused block of memory to the hard disk so that the memory can be used for another purpose.

When the original contents are needed again, they are read back into memory.

Reading and writing to the hard disk is slower (on the order of thousands of times slower) than using real memory, so the programs don't run as fast.

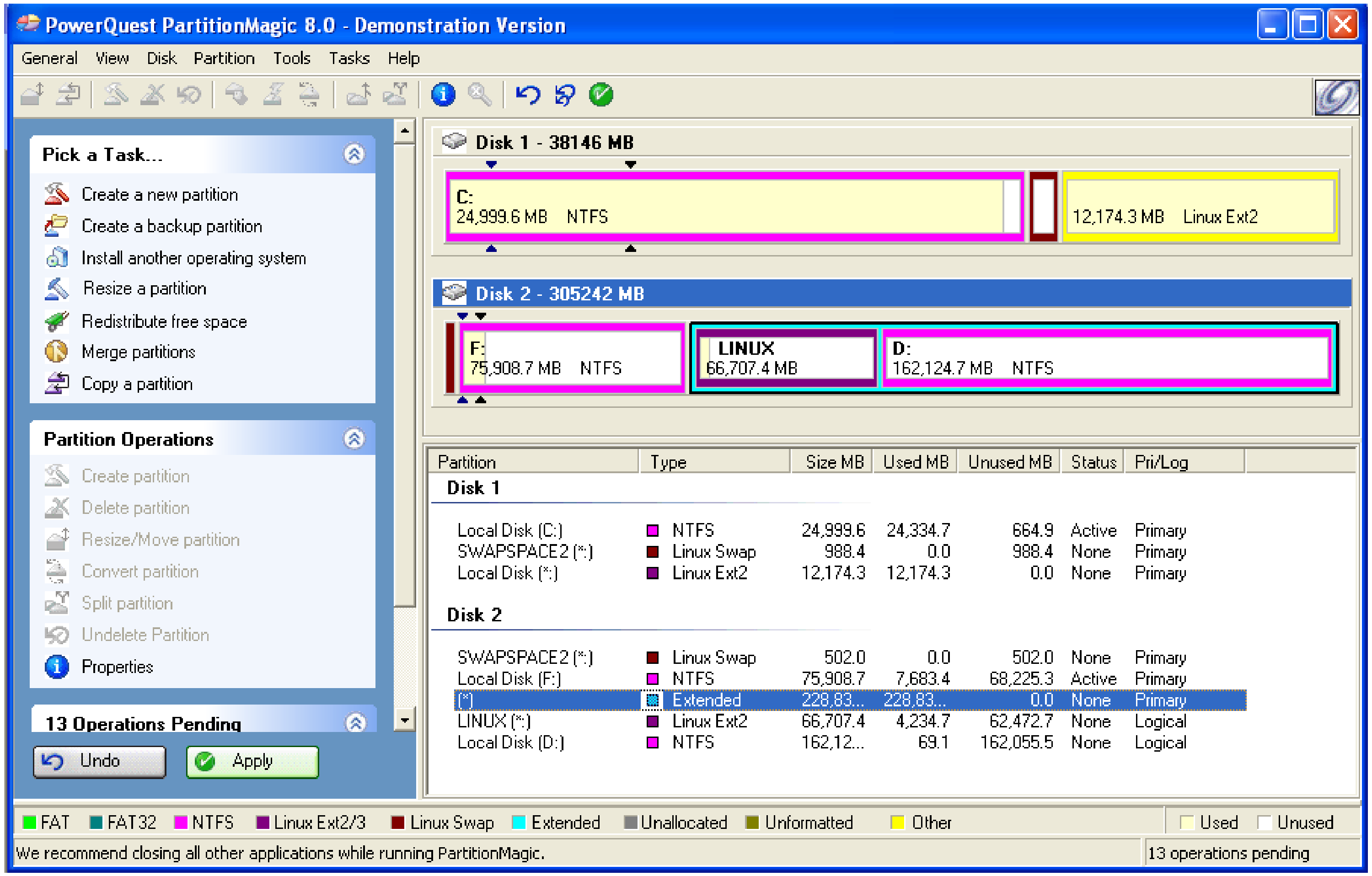
The part of the hard disk that is used as virtual memory is called the ***swap space***.

Linux Virtual Memory

- Swapping is the process whereby a page of memory is copied to the preconfigured space on the hard disk, called swap space, to free up that page of memory. The combined size of the physical memory and the swap space is the amount of virtual memory available.

- Linux has two forms of swap space: the swap partition and the swap file. The swap partition is an independent section of the hard disk used solely for swapping; no other files can reside there. The swap file is a special file in the file system that resides amongst your system and data files.

System with 2 disks each divided into a number of partitions



Partition	Type	Size MB	Used MB	Unused MB	Status	Pri/Log
Disk 1						
Local Disk (C:)	NTFS	24,999.6	24,334.7	664.9	Active	Primary
SWAPSPACE2 (*:)	Linux Swap	988.4	0.0	988.4	None	Primary
Local Disk (*:)	Linux Ext2	12,174.3	12,174.3	0.0	None	Primary
Disk 2						
SWAPSPACE2 (*:)	Linux Swap	502.0	0.0	502.0	None	Primary
Local Disk (F:)	NTFS	75,908.7	7,683.4	68,225.3	Active	Primary
(*)	Extended	228,83...	228,83...	0.0	None	Primary
LINUX (*:)	Linux Ext2	66,707.4	4,234.7	62,472.7	None	Logical
Local Disk (D:)	NTFS	162,12...	69.1	162,055.5	None	Logical

🟩 FAT 🟦 FAT32 🟪 NTFS 🟪 Linux Ext2/3 🟥 Linux Swap 🟦 Extended ⬜ Unallocated 🟫 Unformatted 🟨 Other ⬜ Used ⬜ Unused

We recommend closing all other applications while running PartitionMagic.

13 operations pending

Linux swap space

- If you want to take a look at your swap partition, try (as root) the command: **fdisk –l**

- Output might look something like:

```
 Device Boot      Start         End          Blocks       Id  System
/dev/sda1          2048         2314239      1156096    82  Linux swap
/dev/sda2          2314240    20971519   9328640  83  Linux
```

- The second line represents the main Linux partition, the first the swap partition. The numbers represent the starting and finishing block.

- **Main partition** has a size: **9328640**

- **Swap partition** has a size: **1156096**

Swap Space Size

The following has been recommended:

- **Desktop system** - use a swap space of double system memory, as it will allow you to run a large number of applications (many of which may will be idle and easily swapped), making more RAM available for the active applications.

- **Server** system - have a smaller amount of swap available (e.g. half of physical memory) so that you have some flexibility for swapping when needed.

- **Older desktop** machines (e.g. only 128MB), use as much swap space as you can spare, even up to 1GB.

Swap Files

Swapping to files is usually slower than swapping to a raw partition, so this is not the recommended permanent swapping technique.

Creating a swap file, however, can be a quick fix if you temporarily need more swap space.

Implementing Virtual Memory

- Two characteristics of simple paging (and simple segmentation) are the keys to virtual memory.

- Memory references are dynamically translated into physical addresses at run time i.e. a process may be swapped in and out of main memory such that it occupies different regions.

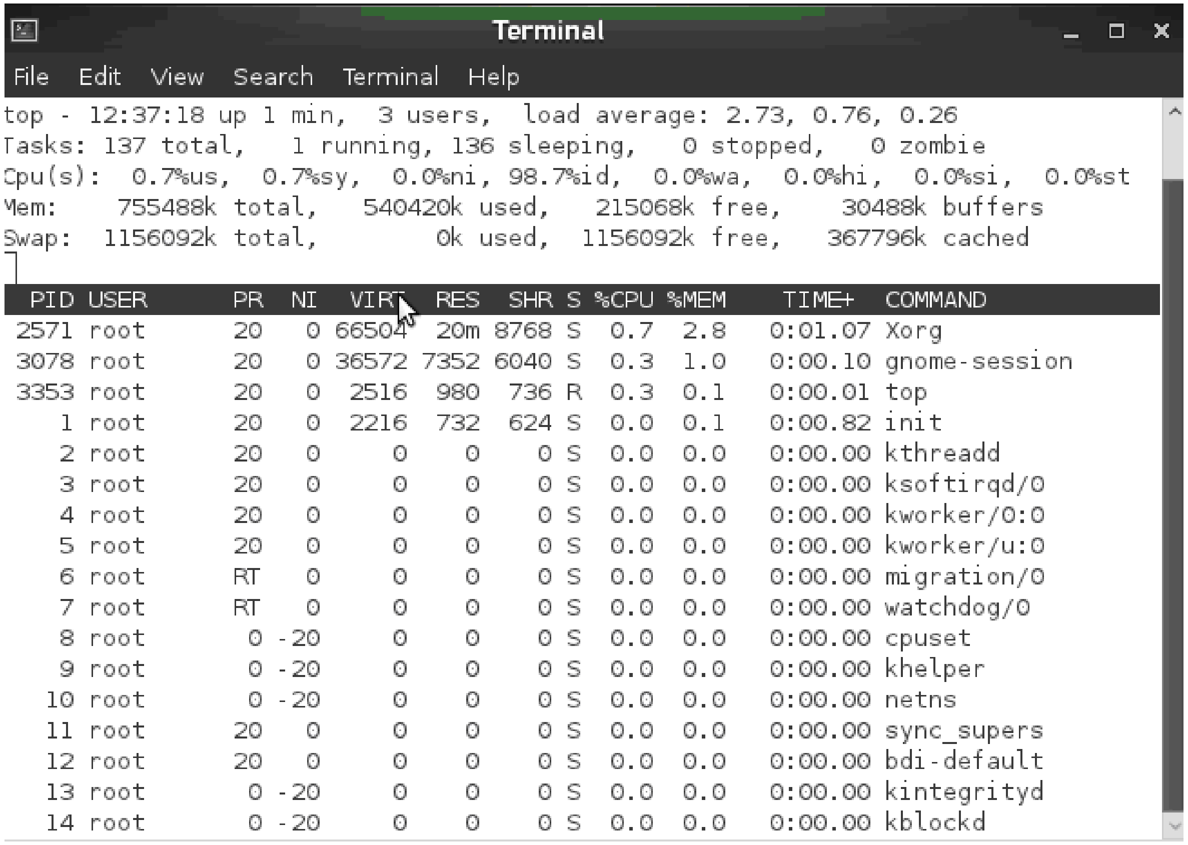
- A process may be broken up into pieces that do not need to located contiguously in main memory i.e. all pieces of a process do not need to be loaded in main memory during execution.

Implementing Virtual Memory

- **This means that all the pieces (pages or segments) need not be in main memory during execution.**

- As a process executes only in main memory, that memory is referred to as **real** memory. A programmer (or user) perceives a much larger memory - that on disk which is referred to as **virtual** memory. Virtual memory allows effective multiprogramming and removes many constraints of main memory.

Displaying Virtual Memory Usage



```
top - 12:37:18 up 1 min,  3 users,  load average: 2.73, 0.76, 0.26
Tasks: 137 total,   1 running, 136 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7%us,  0.7%sy,  0.0%ni, 98.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    755488k total,   540420k used,   215068k free,    30488k buffers
Swap:  1156092k total,        0k used,  1156092k free,   367796k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2571 root      20   0 66504  20m 8768 S  0.7  2.8   0:01.07 Xorg
 3078 root      20   0 36572 7352 6040 S  0.3  1.0   0:00.10 gnome-session
 3353 root      20   0  2516  980  736 R  0.3  0.1   0:00.01 top
    1 root      20   0  2216  732  624 S  0.0  0.1   0:00.82 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0
    5 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    8 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
    9 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   10 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.00 sync_supers
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.00 bdi-default
   13 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kintegrityd
   14 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
```

See slide 22 from Week 7

Displaying Virtual Memory Usage

This is the output of the top command see Week 2 notes.

It includes information on:

- **Mem -** contains statistics on memory usage, including total available memory, free memory, used memory, shared memory, and memory used for buffers.

- **Swap** - contains statistics on swap space, including total swap space, available swap space, and used swap space.

Displaying Virtual Memory Usage

VIRT -- Virtual Image (kb)

- The represents the total amount of **virtual** memory used by the task. It includes all code, data and shared libraries plus pages that have been swapped out.

SWAP -- Swapped size (kb)

- This is the swapped out portion of a task's total **virtual** memory image.

RES -- Resident size (kb)

- The non-swapped **physical** memory a task has used.

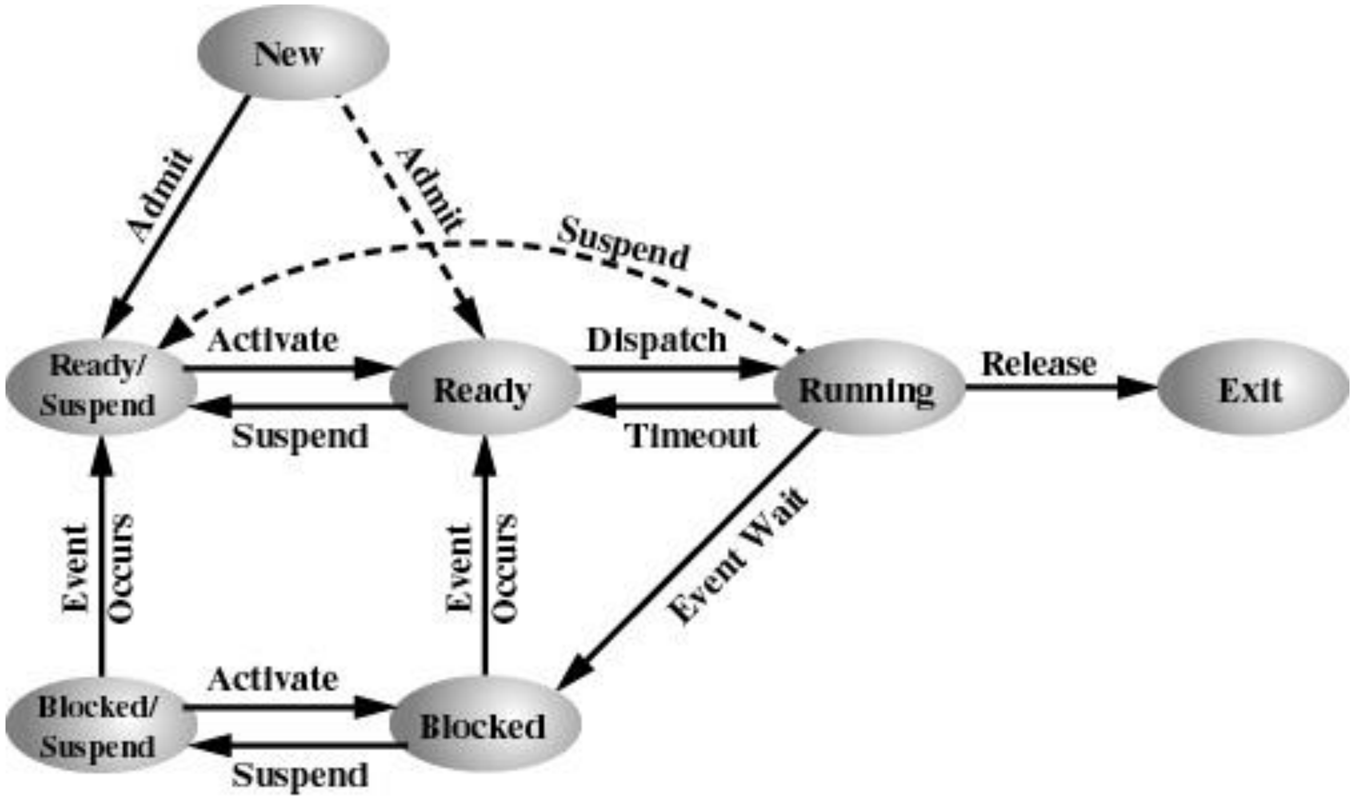
Program Execution

- Operating system brings into main memory a few pieces of the program.
- **Resident set** - portion of process that is in main memory.
- An interrupt is generated when an address is needed that is not in main memory.
- Operating system places the process in a blocking state.

Program Execution

- Piece of process that contains the logical address is brought into main memory
- Operating system issues a disk I/O Read request.
- Another process is dispatched to run while the disk I/O takes place.
- An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the **Ready** state.

Remember Process States (Week2)



(b) With Two Suspend States

Trashing

- Swapping out a piece of a process just before that piece is needed.

- The processor spends most of its time swapping pieces rather than executing user instructions.

- Thrashing slows everything down.

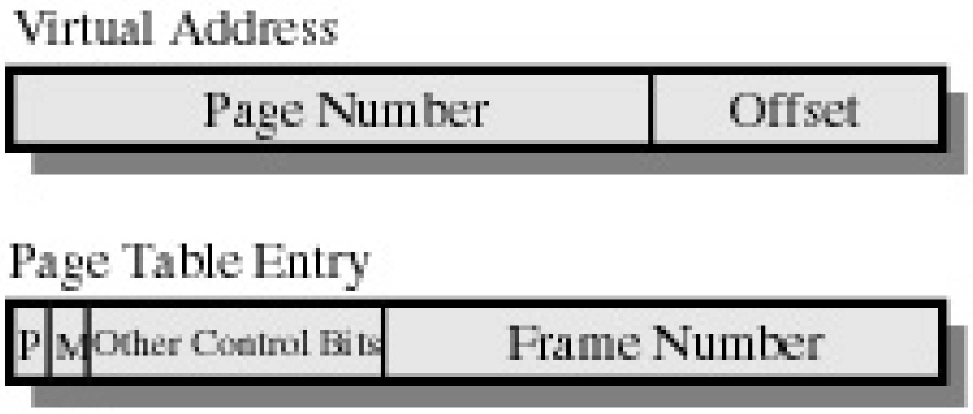
Principle of Locality

- Program and data references within a process tend to cluster.
- Only a few pieces of a process will be needed over a short period of time.
- Possible to make intelligent guesses about which pieces will be needed in the future.
- This suggests that virtual memory may work efficiently.

Virtual Memory Paging

- Each process has its own page table.
- Each page table entry contains the frame number of the corresponding page in main memory.
- A bit is needed to indicate whether the page is in main memory or not.
- A (modify) bit is needed to indicate if the page has been altered since it was last loaded into main memory.
- If no change has been made, the page does not have to be written to the disk when it needs to be swapped out.

Virtual Memory Paging



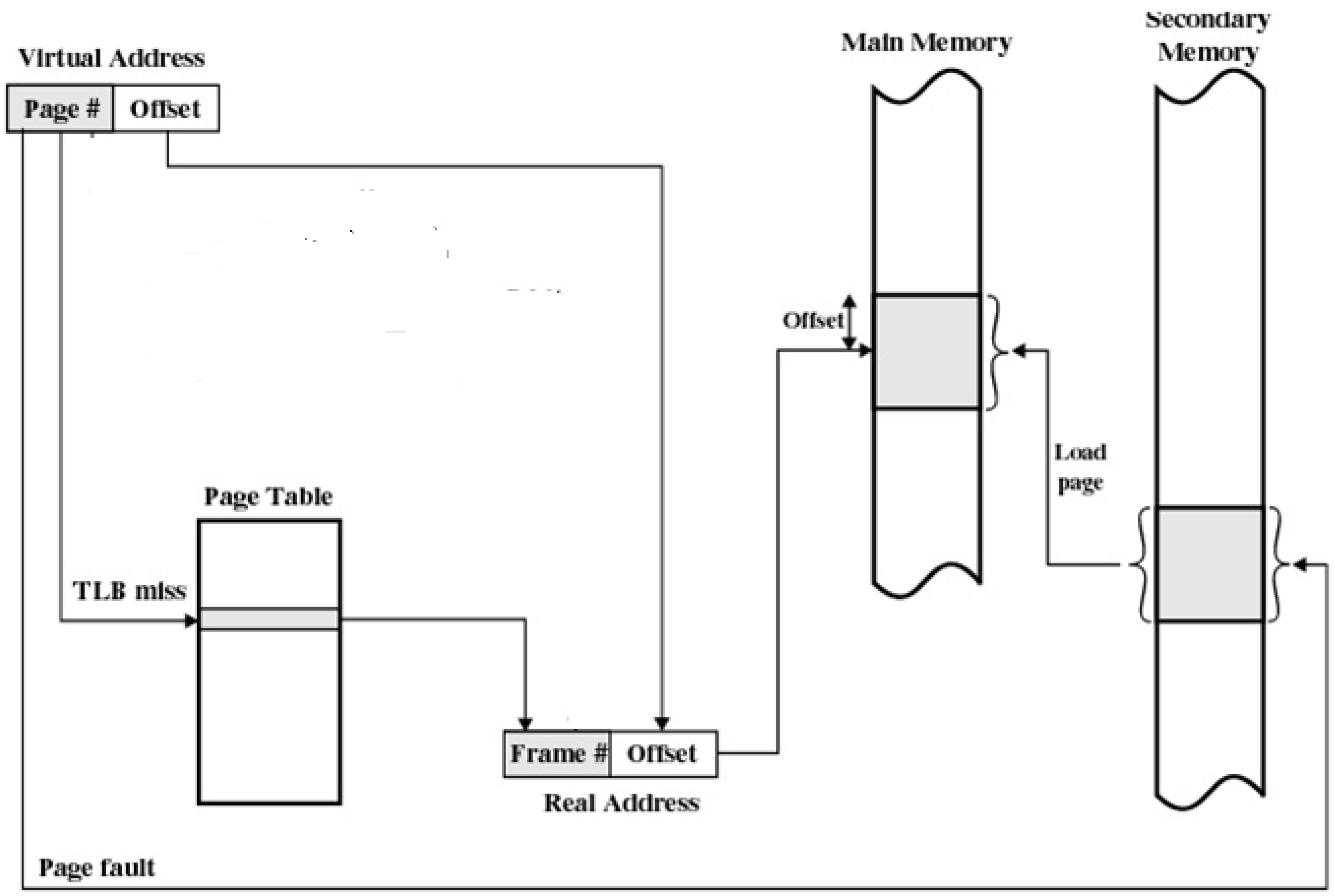
P indicates if the corresponding page is resident in main memory and **M** indicates if the contents have been altered since the page was last loaded.

[Other control bits may also be present e.g. protection, sharing].

Page Faults

- Page marked as not present
- CPU determines page isn’t in memory
- Interrupts the program, starts O.S. page fault handler
- O.S. verifies the reference is valid but not in memory
  - Otherwise reports illegal address
    - Swap out a page if needed
    - Read referenced page from disk + Update page table entry
    - Resume interrupted process (or possible switch to another process)

Virtual Memory Paging



Page Size

- Smaller page size, less amount of internal fragmentation and more pages required per process. More pages per process mean larger page tables. Larger page tables means large portion of page tables in virtual memory.

- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better.