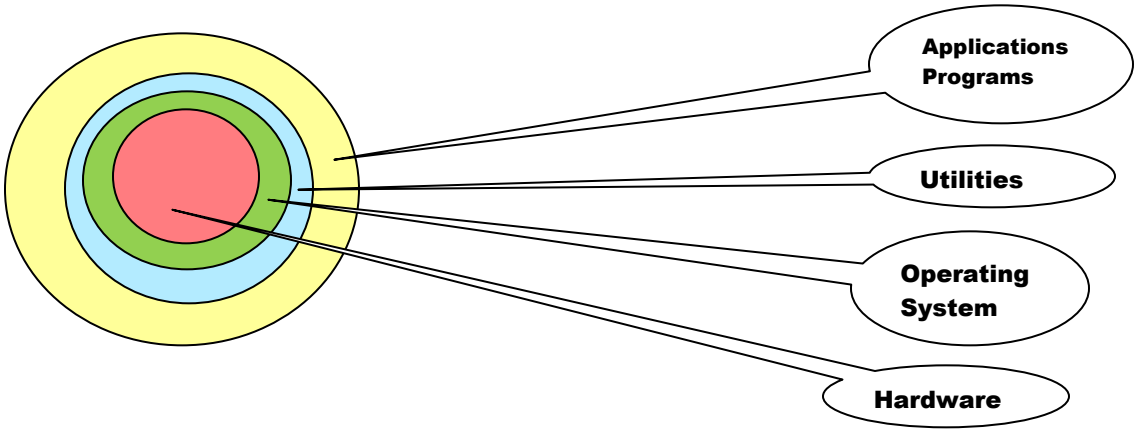
Operating Systems

Interface between User & Computer Hardware



Utilities

- Program Creation
  - Editors
  - Compilers
  - Debuggers
- File Manipulation
  - File Manipulation
  - File Deletion

Program Execution

- Link-Loaders
- Run-Time Management

I/O Device Access & Control

- Storage File Format Structures
- Access Management
- Protection Schemes

Memory Resident File Access & Control

- Memory Resident File Format Structures
- Access Management
- Protection Schemes

System Access & Control

- System-Wide Access
- Resource Access

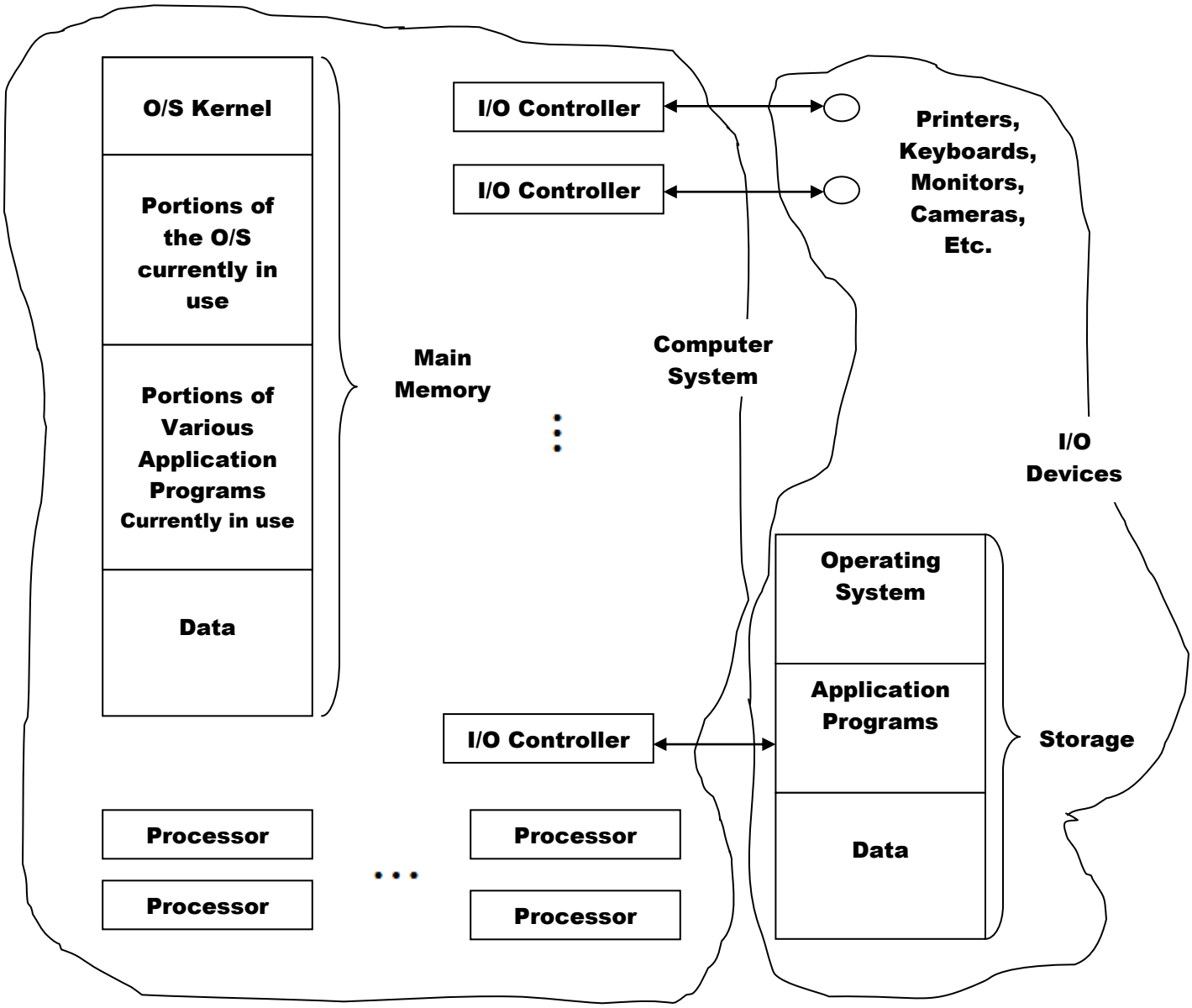
Error Detection & Response Mechanism

- Error Detection
- Error Correction
- Response to Unrecoverable Error

Accounting

- System Usage Collection
- System Performance Tuning
- Forecasting Enhancement Requirements
- Billing Users for Usage

Resource Manager



Operation

- **Allocation of Main Memory is made jointly by both the O/S and Memory Management Hardware**
- **O/S controls access to I/O devices by Application Programs**
- **O/S controls access to and use of files**
- **O/S controls access to and use of the processors, i.e., how much time can be allocated to the execution of a particular Application Program**

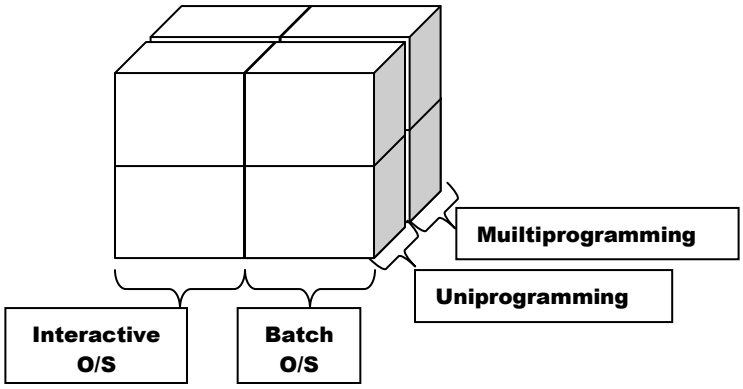
Classification of Operating Systems

Interactive O/S
- **Keyboard & Monitor Access to O/S**
- **Immediate, i.e., Interactive, Communication with the Program during Execution Phase**

Batch O/S
- **Multiple Programs from a diverse set of Users are batched together and submitted for execution**
- **Results are printed for each User Program to be read offline**
- **Example: Batch Updating of Bank Account Master Files at the conclusion of a Transaction Period (end of day)**

Uniprogramming
- **Single Program is loaded into Main Memory**
- **Processor concentrates on executing that program to completion**

Multiprogramming
- **Multiple Programs are loaded into Main Memory**
- **Processor executes the programs concurrently, i.e., switches between programs to maximize processor utilization**



Early Systems
- **No Operating Systems**
- **Console**
  - **Set of Display Lights to indicate the state of the processor**
  - **Single Row of Display Lights used to indicate a single command written in binary numbers**
  - **Set of Toggle Switches to set the Display Lights for a single command written in binary code**
- **Input Device – Card Reader**
- **Program Creation**
  - **Written on Coding Pads**
  - **Transcribed to Hollerith Cards via a Card Punch Machine (cast iron)**
  - **Program in Hollerith Card Format read into the Main Memory by the use of a Card Reader**
  - **Programs were written to absolute memory locations**
  - **Entire program had to loaded into memory**
  - **Program could be written to Magnetic Tape by the use of a Tape Drive in write mode**
  - **Program could be retrieved from the magnetic tape by the use of a Tape Drive in read mode**
- **Program Execution**
  - **Initiated by a special "GO" switch on the console**
  - **Error condition – program halted; status could be investigated by inspecting registers and main memory**
  - **Normal Completion – output to printer or Card Punch**

Scheduling

- **Sign Up Sheet**
  - **Blocks of Time – 15 minutes**
  - **waste of processing time – sign up for more time than required**
  - **Inefficient use of time – need more time than allotted – forced to stop -- necessitates return at a different time – results in inefficient use of programmers time**

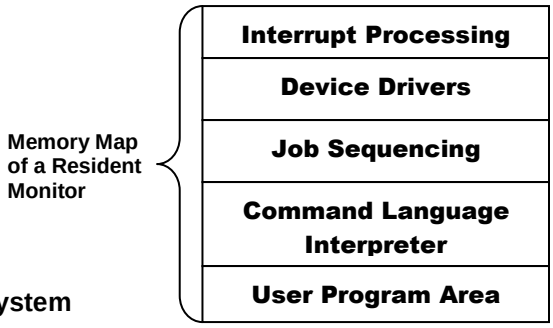
Later Early Systems

- **Setup Time**
  - **Single Program (job)**
  - **Loading Compiler**
  - **Loading High Level language Source Code**
  - **Saving the Object Code after compilation**
  - **Loading & Linking Program Object Code and Library Function Object Codes**

Attaching/Detaching	Loading/Unloading
• Magnetic Tape Drives • Card Readers	• Magnetic Tape Drives • Card Readers

- **Serial Processing – users had access to the computer System in Series**

- **Software Tools**
  - **Libraries of Common Functions**
  - **Linkers**
  - **Loaders**
  - **Debuggers**
  - **I/O Driver Routines**



Simple Batch Operating Systems (Monitors)

- **User**
  - **does not have direct access to the Computer System**
  - **submits job to an operator (cards/tape)**
- **Operator**
  - **batches jobs together sequentially**
  - **submits entire batch to monitor for processing**
- **Monitor controls the sequence of events**

Resident Monitor – portion of the Monitor that must always be in the main memory

Non-resident Monitor – portion of the Monitor that remains in storage until required
- **Utilities**
- **Common Functions**

Resident Monitor determines when to read in a new job

Monitor
- **allocates memory for a new job**
- **reads a new job in from storage**
- **inserts control information at end of job requiring job to restore control to the Monitor**
- **passes control to the new job**
- **at the end of the job, control passes back to the Resident Monitor**

Monitor prints results of the previous execution for the User

Monitor schedules a new job

- **Processor (at a certain point in time)**
  - **is executing instructions from a portion of the Monitor residing in Main Memory that cause the next job to be read into Main Memory**
  - **encounters a branch instruction that leads the processor to start executing the new job**
  - **encounters either an end statement or an error condition ➔ causing the processor to fetch its next instruction from specified locations in the Monitor code**

Control is Passed to a Job
- **Processor is fetching and executing instructions from the user code**

Control is Returned to the Monitor
- **Processor is fetching and executing instructions from the user code**

- **Job Control Language (JCL)**
  - **provides control instructions to the monitor**
  - **FORTRAN**
    - **each instruction and each data item is coded on a separate**
      - **Hollerith card or**
      - **record on tape**
    - **$JOB -- indicates the beginning of a new job**
    - **$FTN -- specifies which FORTRAN compiler should be loaded**
      - **selected compiler produces object code – object code stored**
        - ✓ **in memory ➔ "compile, load & go" operation**
        - ✓ **on tape ➔ $LOAD card is required to place code in memory**

$FTN passes control from Monitor to the FORTRAN compiler
Upon termination of the compiler operation, control passes back to the Monitor

    - **if a $LOAD card is detected, the Monitor invokes the loader which loads the object code into the space previously held by the FORTRAN compiler; the Monitor transfers control to the object program**
    - **if a $LOAD card is not detected, the Monitor transfers control to the object program**

- **Hardware Features (absent from early hardware)**
  - **Memory Protection**
    - **exclude user programs from writing in the Monitor memory space**
  - **Timer**
    - **prohibit any single job from executing beyond a specified time limit**
  - **Privileged Instructions**
    - **instructions provided for use by the Monitor code only**
    - **processor encountering a privilege instruction in a user program, passes control to the Monitor, which then executes an error interrupt**
  - **Interrupts**
    - **ability of the hardware to regain control from an executing program to process an exceptional situation**

Processor time
&
memory space

are both shared by the

user program
&
Monitor

➔ **Utilization of the system is improved**

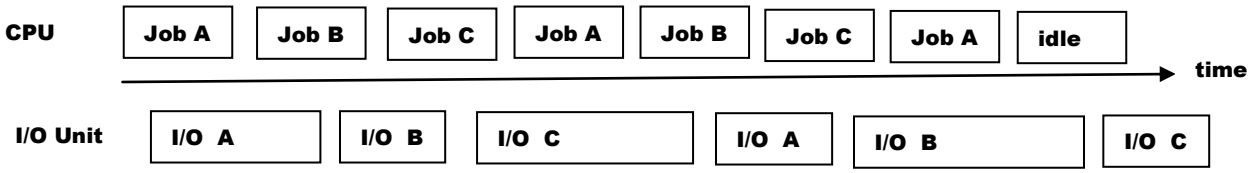
Multiprogrammed Batch Systems

- Idle Processor
  - CPU speed >> I/O device speed
  - single job available ➔ CPU must wait for I/O completion

- Multitasking
  - multiple tasks within the same program
  - processed concurrently

- Multiprogramming
  - multiple processes from multiple programs
  - processed concurrently

- Multiprogramming/Multitasking
  - multiple jobs are available for execution, CPU
    - encounters an I/O operation on current job
    - switches to alternate job while waiting for I/O completion



Efficient Batch Processing

- Hardware Requirements
  - I/O Interrupt Processing
  - Direct Memory Access (DMA)
  - Memory Management

- Time-Sharing Systems
  - Transaction Processing – reservation systems
  - Multiple Interactive Jobs
  - O/S interleaves the execution of each user program in a short burst (quantum) of time
  - N users ➔ each user has effective control of a computer with slightly less than $\frac{1}{N}$ the speed of the actual computer (overhead costs of context switching)

Direct User Interaction with the Executing Process

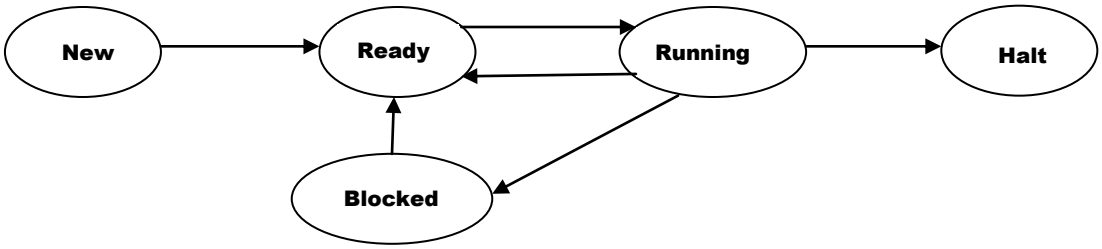
Scheduling

- Long-Term Scheduling (infrequent exedcution)
  - determines which programs are admitted for processing
    - allocated memory space
    - allocated process control block (becomes a process)
    - process control block placed in the Ready Queue
      - ✓ batch system ➔ PCB held on disk in a disk queue
      - ✓ some systems ➔ PCB swapped-out to raw disk (no file management system)
  - considers the degree of multiprogramming & the size of remaining free memory

- Medium-Term Scheduling
  - manages degree of multiprocessing
  - swaps jobs between raw disk memory
- Short-Term Scheduling (frequent execution)
  - decides which job to execute next
  - call the dispatcher, dispatcher executes the context switch between PCBs

Time-Sharing System – accepts all users until system is saturated; locks further entry
Algorithm for Accepting New Processes – balance CPU bound versus I/O Bound

Process States



- **New – admitted by Long-Term Scheduler, PCB allocated, OS needs to allocate memory, etc**
- **Ready – process is waiting access to processor, i.e., PCB is complete**
- **Running – process is executing**
- **Waiting – process is suspended, waiting for some system resource, e.g., I/O completion**
- **Halted – process has terminated, awaiting deallocation**

Process

- **program code**
- **contents of the process control block**

Process Control Block

- **Identifier (unique integer)**
- **State**
- **Priority**
- **PC contents**
- **Memory Pointers – start & end locations of allocated memory space**
- **Context Data – contents of all other registers**
  containing information important to the continued operation of the process
- **I/O Status Information**
- **Accounting Information**

Context Switch

- **current user process issues a service call, e.g., I/O request**
  - **process is suspended until call is completed**
- **current user process issues an interrupt, i.e., hardware-generated signal**
  - **process is suspended until interrupt is processed**
  - **privileged instruction, divide by zero, time quantum**
- **interrupt event unrelated to current process**
  - **process is suspended until interrupt is processed**

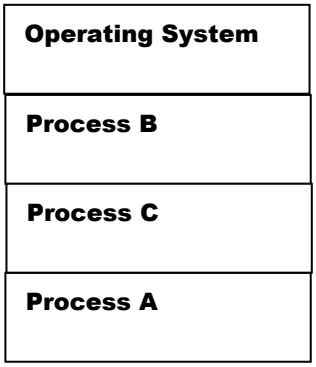
- **OS receives control of the at the interrupt handler if an interrupt occurred**
- **OS receives control of the at the service-call handler if an service call occurred**

Process Scheduling

- **Long Term Queue**
  - **disk resident**
  - **list of candidate jobs waiting to be selected by Long-Term Scheduler**
- **Short-Term Queue – Ready Queue**
- **Multiple I/O Queues – one for each I/O Device**

- **I/O Scheduling – upon completion, OS moves process from queue to Ready Queue**
  Selects a waiting process and signals I/O device to start execution

Memory Management



- Swapping
  - Ready Queue may be empty – all the processes may be in the I/O queues
  - Swap processes from the I/O queues to
    - an **Intermediate Queue** on disk
    - swap space, i.e., raw disk – partition with no file system (faster swapping)
  - OS selects a process from
    - the **Intermediate Queue** to place in the **Ready Queue**
    - swap space, i.e., raw disk – partition with no file system (faster swapping)

- Partitioning
  - Fixed Size
    - Equal Size
    - Unequal Size
  - Variable Size

Holes in Memory Areas of memory unusable because of small size
Compaction Moving processes to different memory locations to consolidate holes

- Addresses
  - Logical Address
    - location relative to the beginning of the program (location zero)
    - program instructions contain logical addresses
  - Physical Address
    - actual location in main memory
    - **base address**
      - ✓ physical address of the current starting location of process in physical memory
    - processor adds the **base address** to the logical address to obtain the physical address

Paging

- pages – small fixed size chunks of program
- frames – small fixed size chunks of memory

← same size

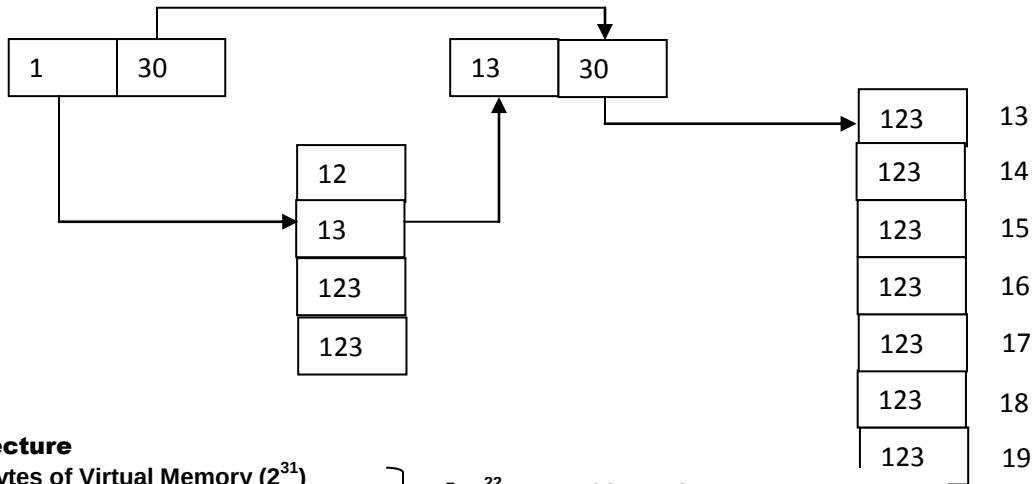
- No external fragmentation of memory
- Internal fragmentation ≤ ½ of last frame
- Page tables maintain list of frames in use
- OS also keeps a list of free frames

Demand Paging

- Each page of a process is allocated a frame in memory
  only when it is referenced from a previous frame
- Page Fault
- Page Replacement Algorithms
- Trashing
  - repeatedly throwing out pages just before they were to be referenced
- Real Memory
  - actual physical memory
- Virtual Memory
  - perceived memory requirements of entire process
  - may be larger than size of real memory
  - exists on disk storage

Page Table Structure

- Memory-Resident Page Table Location (starting position) is held in a register
- Logical Address [page #, offset]
- Page # indexes into page table producing the frame #
- Physical Address [frame #, offset]
- Memory Management Unit – hardware required
- One page table / process



VAX Architecture

- 2 GBytes of Virtual Memory (2^{31})
- 512 byte pages (2^9)

→ 2^{22} page table entries per process

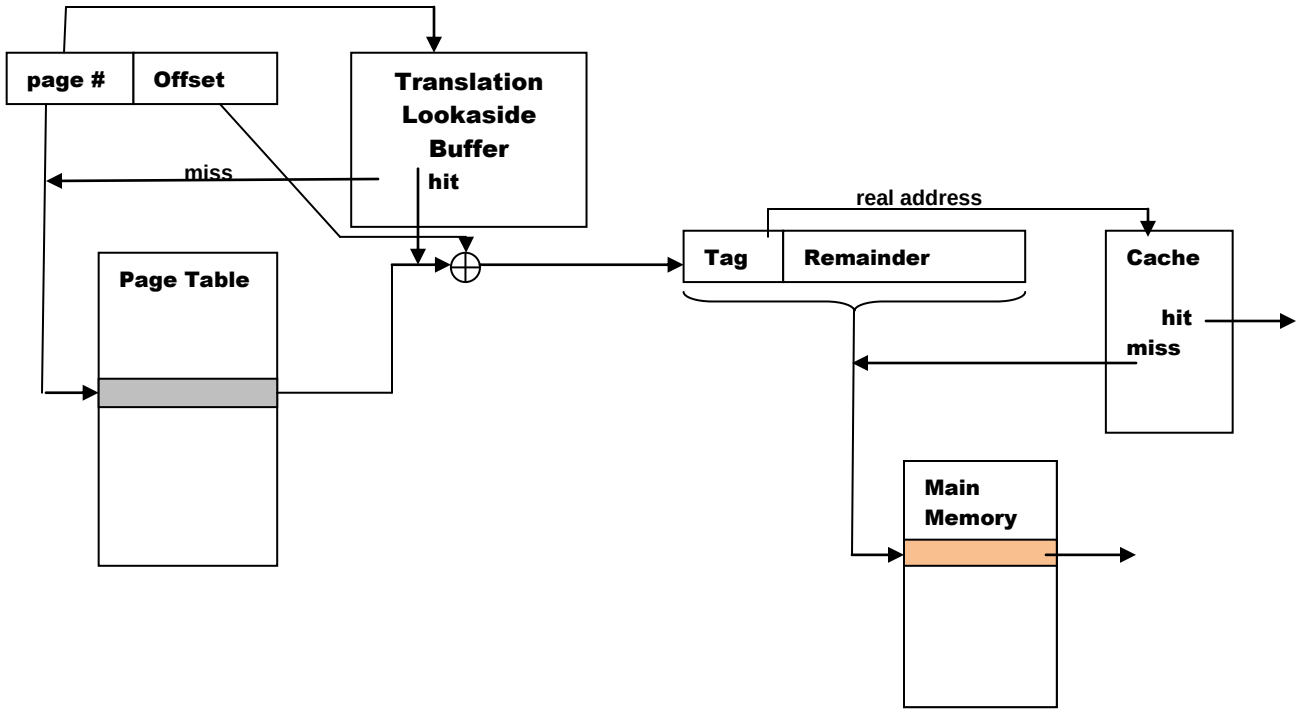
- page tables are stored in virtual memory, i.e., on disk
- page table length ≤ page length

Two-Level Paging Scheme

- Page Directory
  - each entry points to a page table
  - if (page table length ≤ page length) && (directory length ≤ page length) then
    total pages ≤ $(\text{page length})^2$

Inverted Page Table

- index the page table by **frame number** rather than by page number
- use **hash function** to map the page number into a location in the page table
- **hash collisions** are maintained in a linked list, i.e., a chain of entries
- size of an inverted page table << size of normal page table



Segmentation

- visible to programmer
- allows programmer to view memory multiple address segments
- segments are of dynamic size
- each segment may be assigned access and usage rights
- used to associate privilege & protection attributes to instructions & data
- memory reference (segment #, offset)

- data structures may be assigned to a segment
  OS may expand or shrink the size of the segment as needed
- segments may be changed & recompiled independently
- segments may be shared among different processes

Pentium Memory Management

- **Unsegmented Unpaged Memory – low-complexity high-performance controllers**
- **Unsegmented Paged Memory – BSD (Berkeley Unix Distributions)**
- **Segmented Unpaged Memory – predictable access times – embedded systems**
- **Segmented Paged Memory – Unix System V Distributions**

- **Segmentation**
  - **virtual address (Pentium logical address)**
    - **segment reference**
      - ✓ **2-bit protection mechanism**
      - ✓ **14-bit segment specification**
    - **offset 32-bit**
  - **unsegmented memory → virtual memory = 2^{32} bytes, i.e., 4 GBytes**
  - **segmented memory → virtual memory = 2^{46} bytes, i.e., 64 TBytes**
    - **½ virtual address space == global, i.e., used by all processes**
    - **½ virtual address space == local, i.e., distinct for each process**
  - **physical address (32-bit address) → 4 GBytes**
  - **segment protection**
    - **privilege levels**
      - ✓ **0 : kernel → memory management, protection, access control** ← **most protected**
      - ✓ **1 : kernel → remainder of OS modules**
      - ✓ **2 : specialized application subsystems with their own security mechanisms, e.g., DBMS, Office Automation, Software Engineering Environments**
      - ✓ **3 : application programs** ← **least protected**
    - **access attribute**
  -

Data Segment Privilege Level – Classification
Program Segment Privilege Level – Clearance

Process may only access Data Segments for which its Clearance is lower than or equal to (more privileged or same privilege) as the Classification of the Data Segment