Operating System Support

Lecture Content

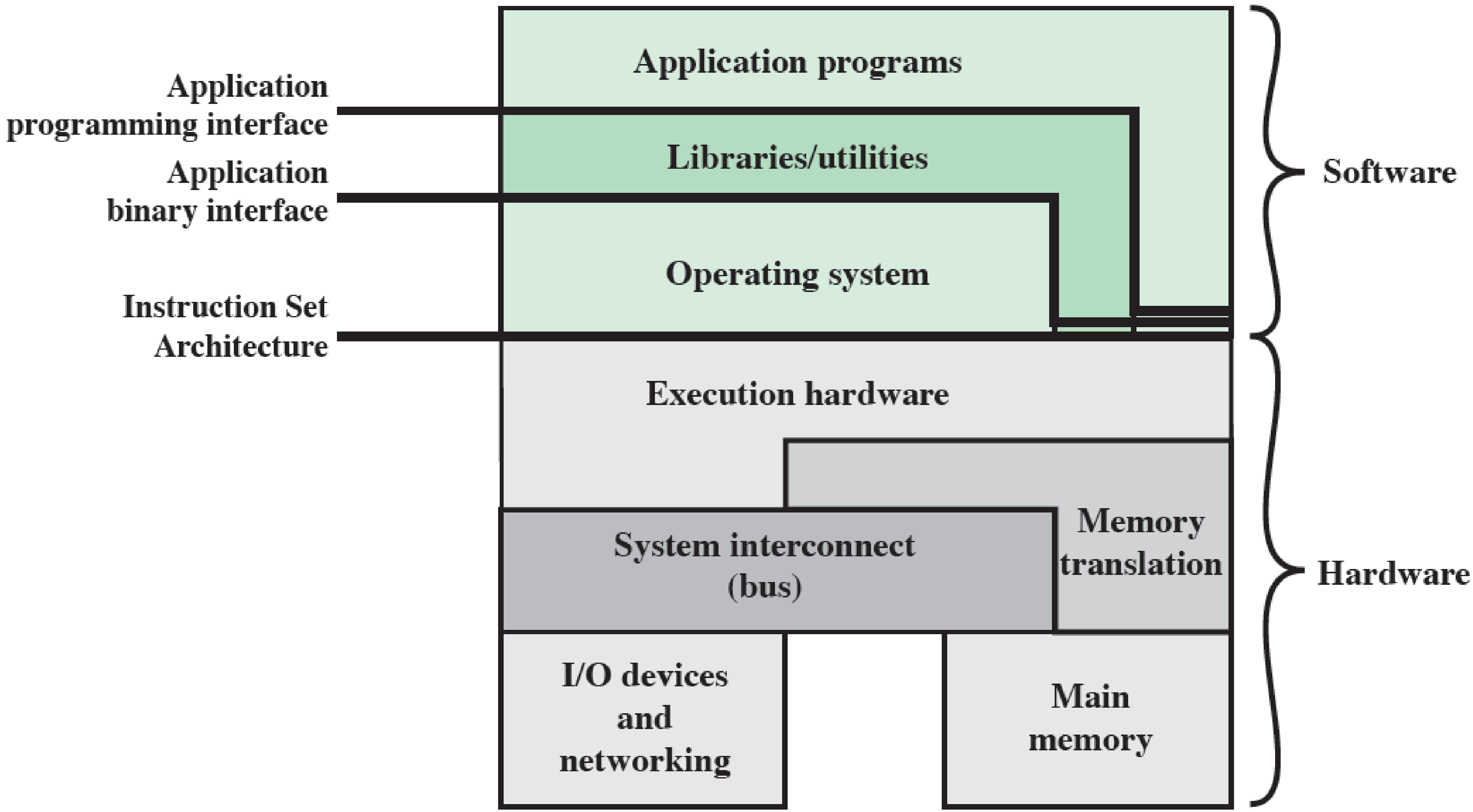
- OS in General
- Memory Management
- Virtual Memory
- Segmentation
- Memory Management of the Pentium II

Operating System Support

Learning Objectives

- Understand the OS from the point of view of the CPU
- Understand the concepts and necessity of scheduling
- Understand paging and segmentation
- Understand the concepts of virtual memory

Computer Hard and Software Structure



Key Interfaces For Computers

- **Instruction set architecture (ISA)**
  - Defines the machine language instructions of a computer
  - Boundary between hardware and software
- **Application binary interface (ABI)**
  - Defines a standard for binary portability across programs
  - Defines the system call interface to the operating system
- **Application programming interface (API)**
  - Gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls
  - Using an API enables application software to be ported easily to other systems that support the same API

Small Overview over the x86_64 Linux ABI

- Functions Calls:
  - Integer Parameters: %rdi, %rsi, %rdx, %rcx, %r8, %r9
  - Float Parameters: %xmm0-%xmm7
  - Further Parameters on the stack
  - Return Values: %rax, %rdx, %xmm0, %xmm1
  - Callee registers: %rsp, %rbx, %r12-%r15
- Memory and Stack Design
- Recommended reading:

  http://www.x86-64.org/documentation/abi.pdf

Operating System as Resource Manager

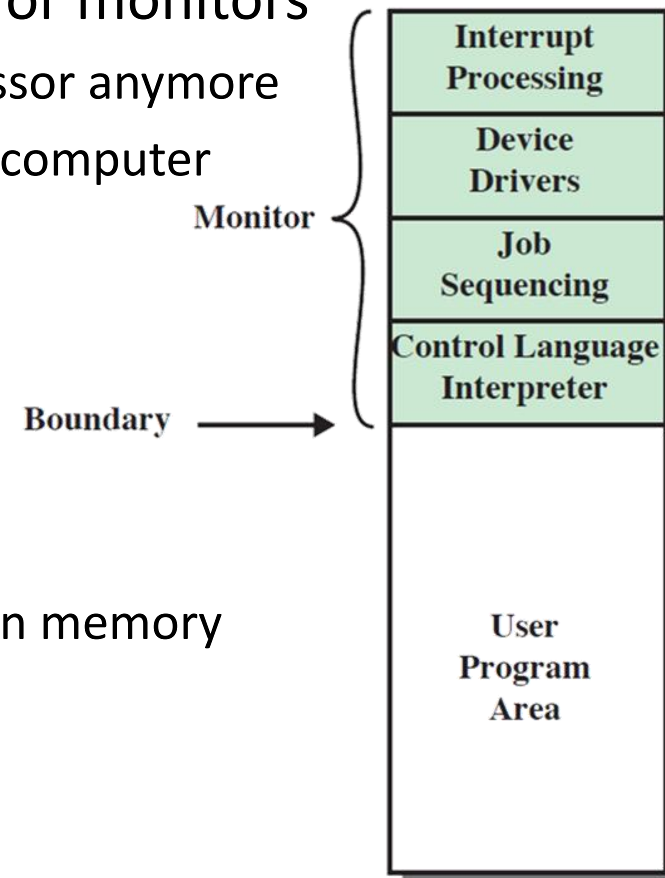
- A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions
- The OS is responsible for managing these resources
- The OS as a control mechanism is unusual in two respects:
  - The OS functions in the same way as ordinary computer software – it is a program executed by the processor
  - The OS frequently relinquishes control and must depend on the processor to allow it to regain control

Back then …

- From the late 1940s to the mid-1950s the programmer interacted directly with the computer hardware – there was no OS
- Problems:
  - **Scheduling**
    - Sign-up sheets were used to reserve processor time
    - This could result in wasted computer idle time if the user finished early
    - If problems occurred the user could be forced to stop before resolving the problem
  - **Setup Time**
    - A single program could involve:
      - Loading the compiler plus the source program into memory
      - Saving the compiled program
      - Loading and linking together the object program and common functions

Batch Operating Systems

- The wasted time due to scheduling and setup time was unacceptable
- Advent of simple batch operating systems or monitors
  - Programmer has no direct access to the processor anymore
  - The user submits the job on cards or tape to a computer operator
  - Put on an input device, for use by the monitor
- Point of View of the Monitor
  - Controls the sequence of events
  - Thus, most of the monitor has to be in the main memory
  - This is called **resident monitor**



Point of View of the Processor

- Processor executes instructions from the monitor
  - These instructions cause the next job to be read into main memory
  - The processor executes the instruction in the user’s program until it encounters an ending or error condition
  - Either event causes the processor to fetch its next instruction from the monitor program
- The monitor handles setup and scheduling
  - A batch of jobs is queued up and executed with no idle time
  - Job control language (JCL)
    - Special type of programming language controlling the monitor

- **Monitor, or batch OS, is simply a computer program and relies on the processors ability to fetch instructions from various portions of main memory and to seize and relinquish control alternately**

Desirable CPU Features

- Memory Protection
- Timer
- Privileged Instructions
- Interrupts

Desirable CPU Features

- **Memory Protection**
  - User program must not alter the memory area containing the monitor
  - The processor hardware should detect an error and transfer control to the monitor
  - The monitor aborts the job, prints an error message, and loads the next job
- **Timer**
- **Privileged Instructions**
- **Interrupts**

Desirable CPU Features

- **Memory Protection**
- **Timer**
  - Used to prevent a job from monopolizing the system
  - If the timer expires an interrupt occurs and control returns to monitor
- **Privileged Instructions**
- **Interrupts**

Desirable CPU Features

- **Memory Protection**
- **Timer**
- **Privileged Instructions**
  - Can only be executed by the monitor
  - If the processor encounters such an instruction while executing a user program an error interrupt occurs
  - I/O instructions are privileged so the monitor retains control of all I/O devices
- **Interrupts**

Desirable CPU Features

- **Memory Protection**
- **Timer**
- **Privileged Instructions**
- **Interrupts**
  - Gives the OS more flexibility in relinquishing control to and regaining control from user programs

Multiprogramming and Time Sharing Systems

- **Multiprogramming Batch Processing**
  - The monitor executes several programs at the same time
  - More efficient, as wait times can be used by the processor
  - Nevertheless, there is no interaction possible

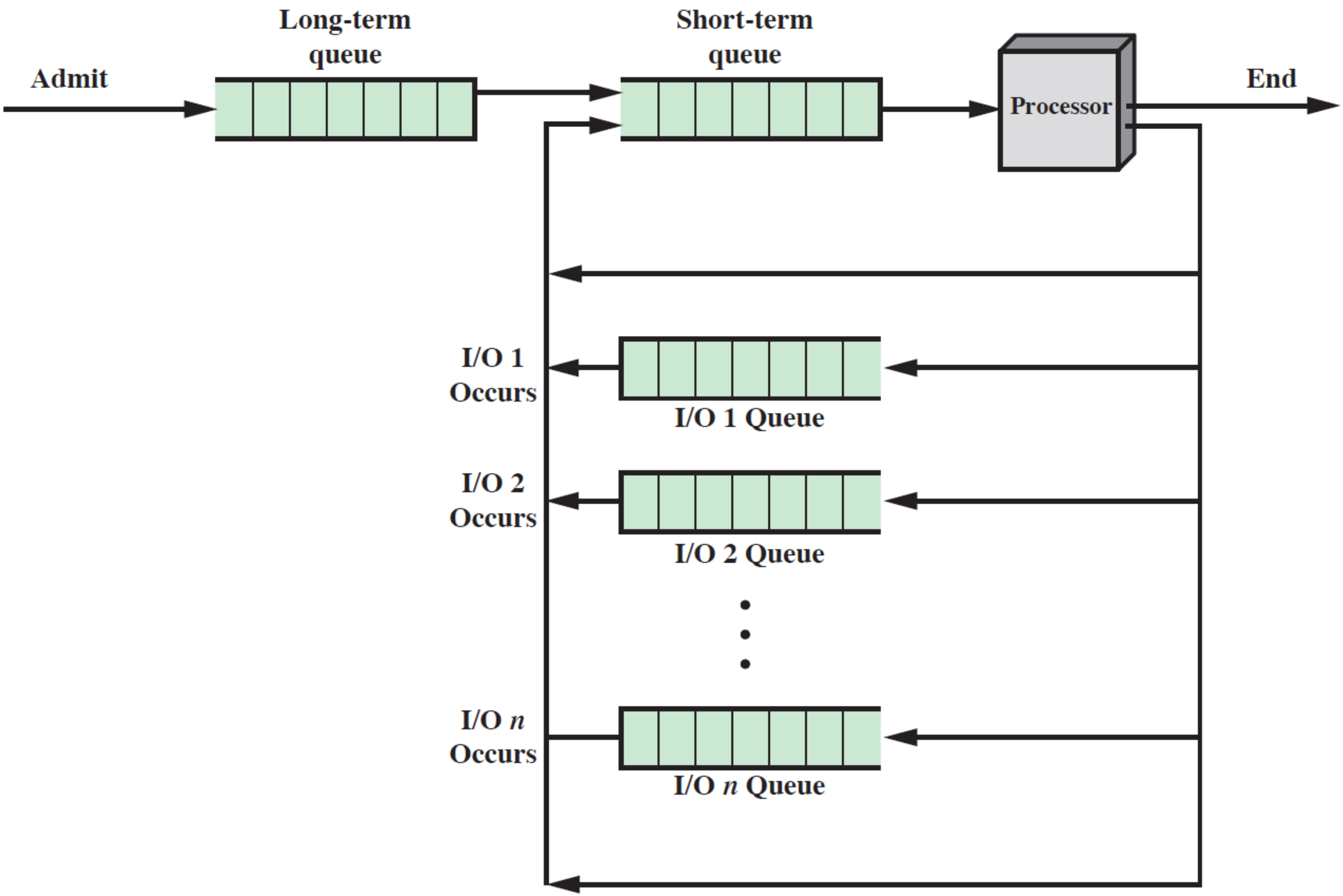
- **Time Sharing Systems**
  - Used when the user interacts directly with the computer
  - Processor’s time is shared among multiple users
  - Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Scheduling

- The key to multiprogramming

- **Long-Term Scheduling**
  - The decision to add to the pool of processes to be executed
- **Medium-Term Scheduling**
  - The decision to add to the number of processes that are partially or fully in main memory
- **Short-Term Scheduling**
  - The decision as to which available process will be executed by the processor
- **I/O Scheduling**
  - The decision as to which process's pending I/O request shall be handled by an available I/O device

Scheduling


Long-term queue
Short-term queue
Admit
Processor
End
I/O 1 Occurs
I/O 1 Queue
I/O 2 Occurs
I/O 2 Queue
I/O n Occurs
I/O n Queue

Operating System Support
Lecture Content

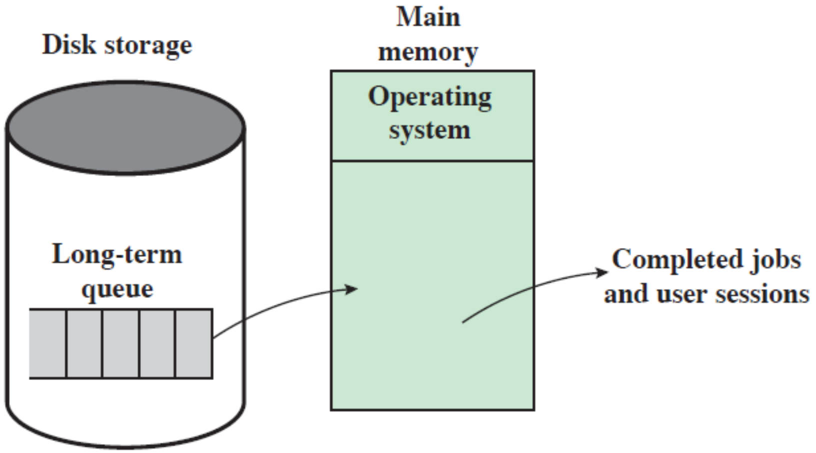
- OS in General
- Memory Management
- Virtual Memory
- Segmentation
- Memory Management of the Pentium II

Memory Management

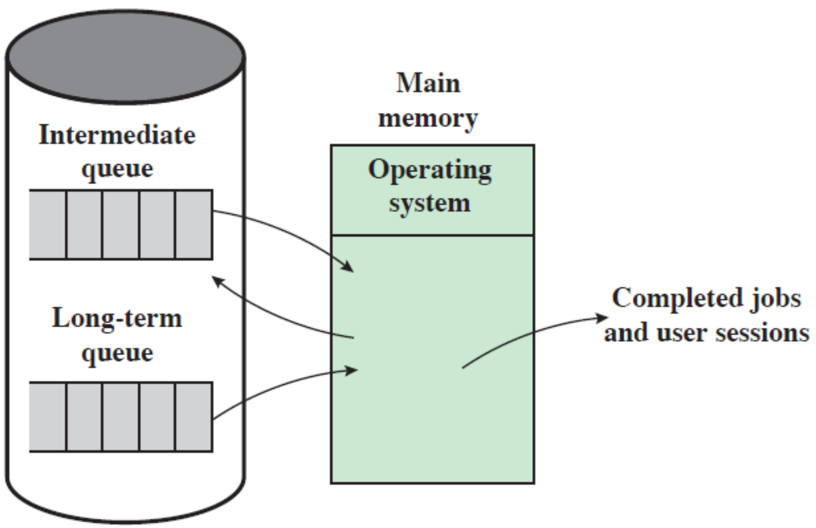
- Programs being executed have to be in main memory
- In order to accommodate more and more programs, we have to increase the amount of memory available

- But there are two flaws in this approach:
  - First, main memory is expensive, even today
  - Second, the appetite of programs for memory has grown as fast as the cost of memory has dropped
  - So larger memory results in larger processes, not more processes

Swapping

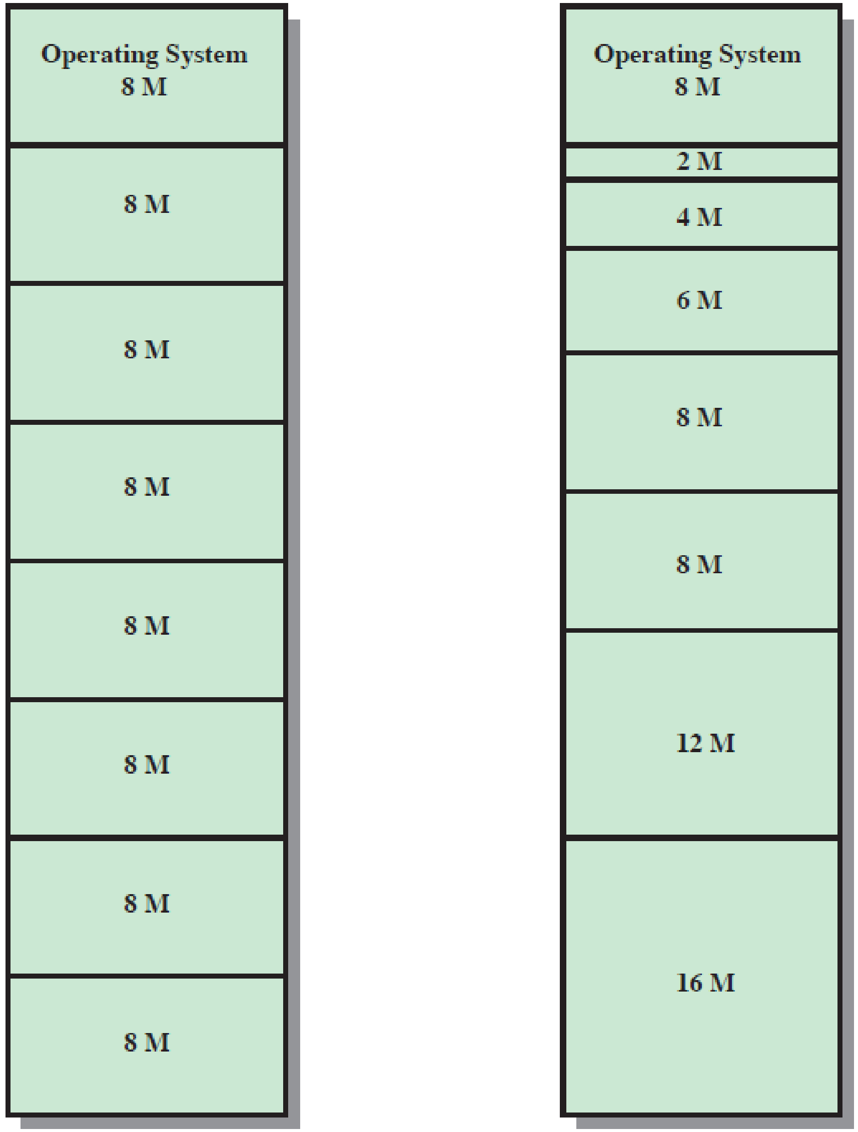


(a) Simple job scheduling



(b) Swapping

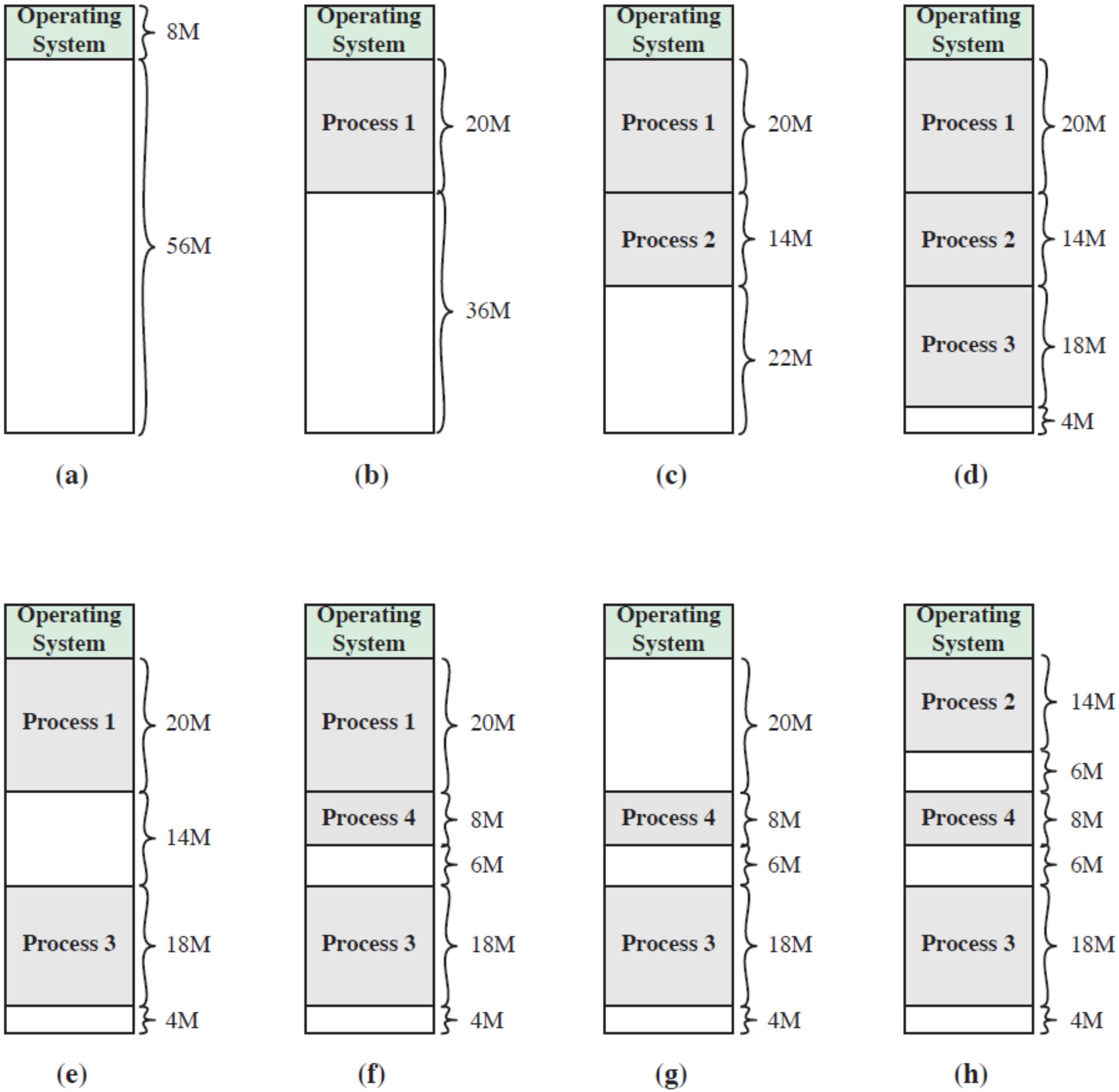
Fixed-Size Partitions



(a) Equal-size partitions

(b) Unequal-size partitions

Dynamic Partitioning



Observations

- It is not likely that a program is in the same place in main memory each time it is swapped in
- A process in memory consists of instructions and data
  - Addresses of data items
  - Addresses of instructions, used for branching instructions

- To solve this problem, programs only use logical addresses
- **Logical Address**
  - expressed as a location relative to the beginning of the program
- **Physical Address**
  - an actual location in main memory
- **Base Address**
  - current starting location of the process

Limitations

- With a simple virtual address, we can fit a program and its data anywhere in the memory

- **But**:
  - Do we really know how much memory do we need?
  - What, if the program requires more space during runtime, but the space is already occupied?
  - Where to place the stack?
  - …

Operating System Support

Lecture Content

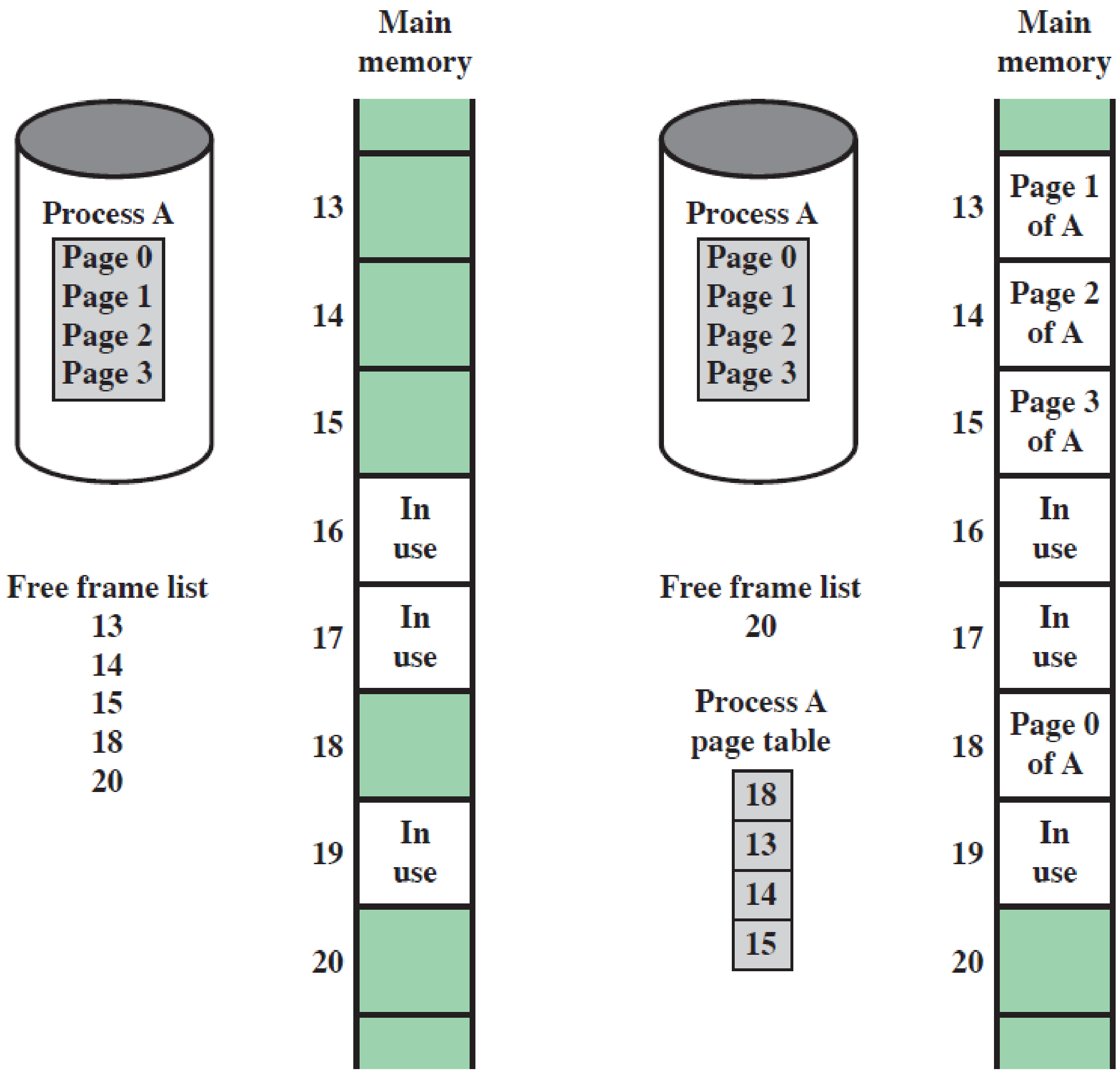
- OS in General
- Memory Management
- Virtual Memory
- Segmentation
- Memory Management of the Pentium II

Drawbacks of Fixed-Size or Dynamic Portioning

- Both unequal fixed-size and variable-size partitions are inefficient in the use of memory

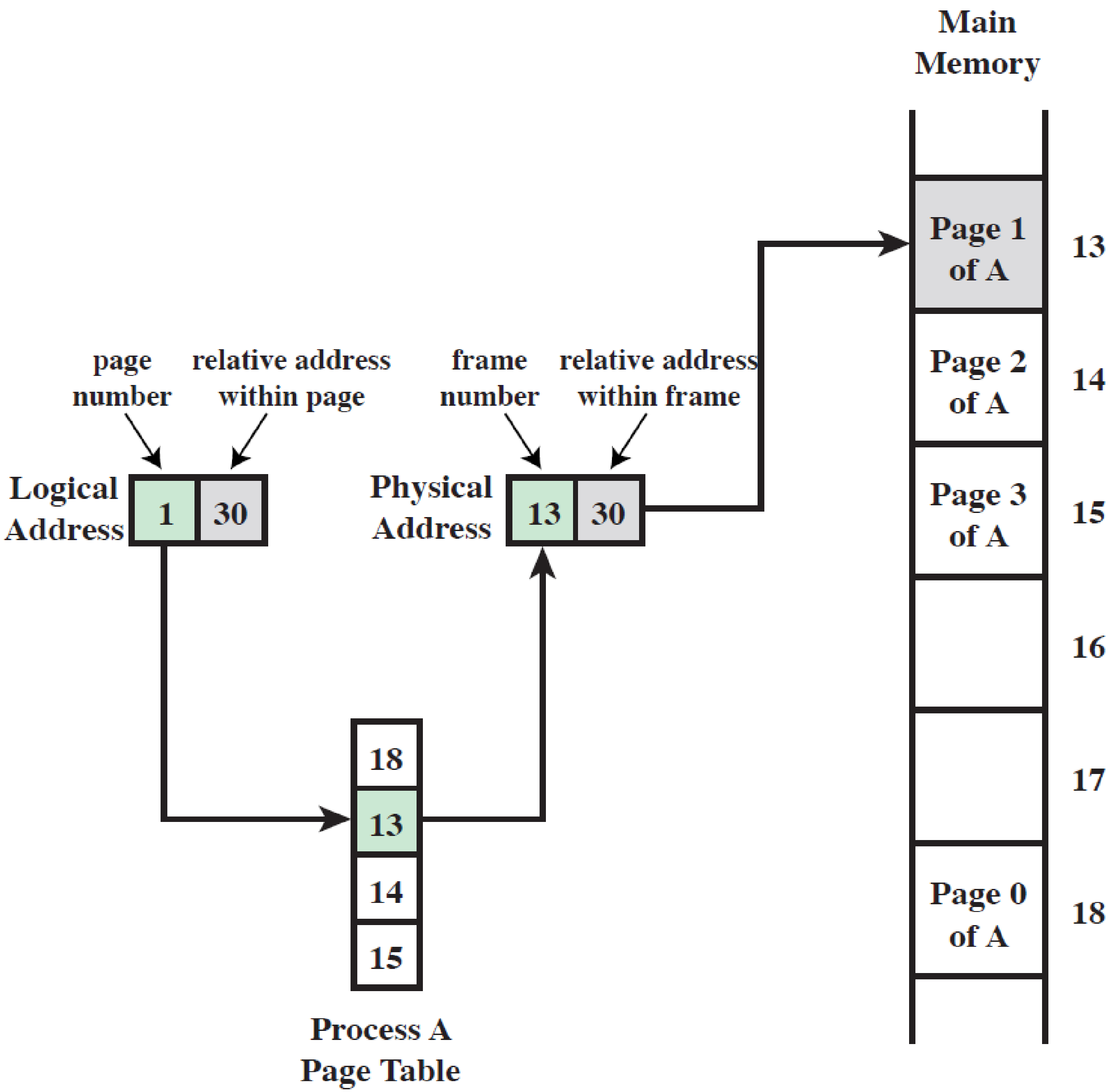
- Solution: Paging
  - Memory is partitioned into equal fixed-size small chunks
  - Each process is also divided into small fixed-size chunks
  - Then the chunks of a program, known as pages, are assigned to available chunks of memory, known as frames, or page frames
  - Wasted space in memory for a process is only a fraction of the last page
  - The list of free frames is maintained by the OS

Paging



(a) Before (b) After

Paging


Main
Memory
Page 1
of A
13
Page 2
of A
14
Page 3
of A
15
16
17
Page 0
of A
18
page
number
relative address
within page
frame
number
relative address
within frame
Logical
Address
1
30
Physical
Address
13
30
18
13
14
15
Process A
Page Table

Paging

- Paging extends the concept of logical addresses
- A simple base address will no longer suffice
- Rather, the OS maintains a page table for each process
- The page table shows the frame location for each page of the process

- The processor translates the logical address into a physical address
- The processor must know how to access the page table of the current process
- Presented with a logical address (page number, relative address), the processor uses the page table to produce a physical address (frame number, relative address)

Virtual Memory

- Each page of a process is brought in only when it is needed
  - Principle of locality
  - If the program references to a page not in main memory, a page fault is triggered which tells the OS to bring in the desired page
- **Advantages**
  - More processes can be maintained in memory
  - Time is saved because unused pages are not swapped in and out of memory
- **Disadvantages**
  - Page replacement: A new page throws out an existing page
  - This replacements can get inefficient
  - Thrashing
    - When the processor spends most of its time swapping pages rather than executing instructions

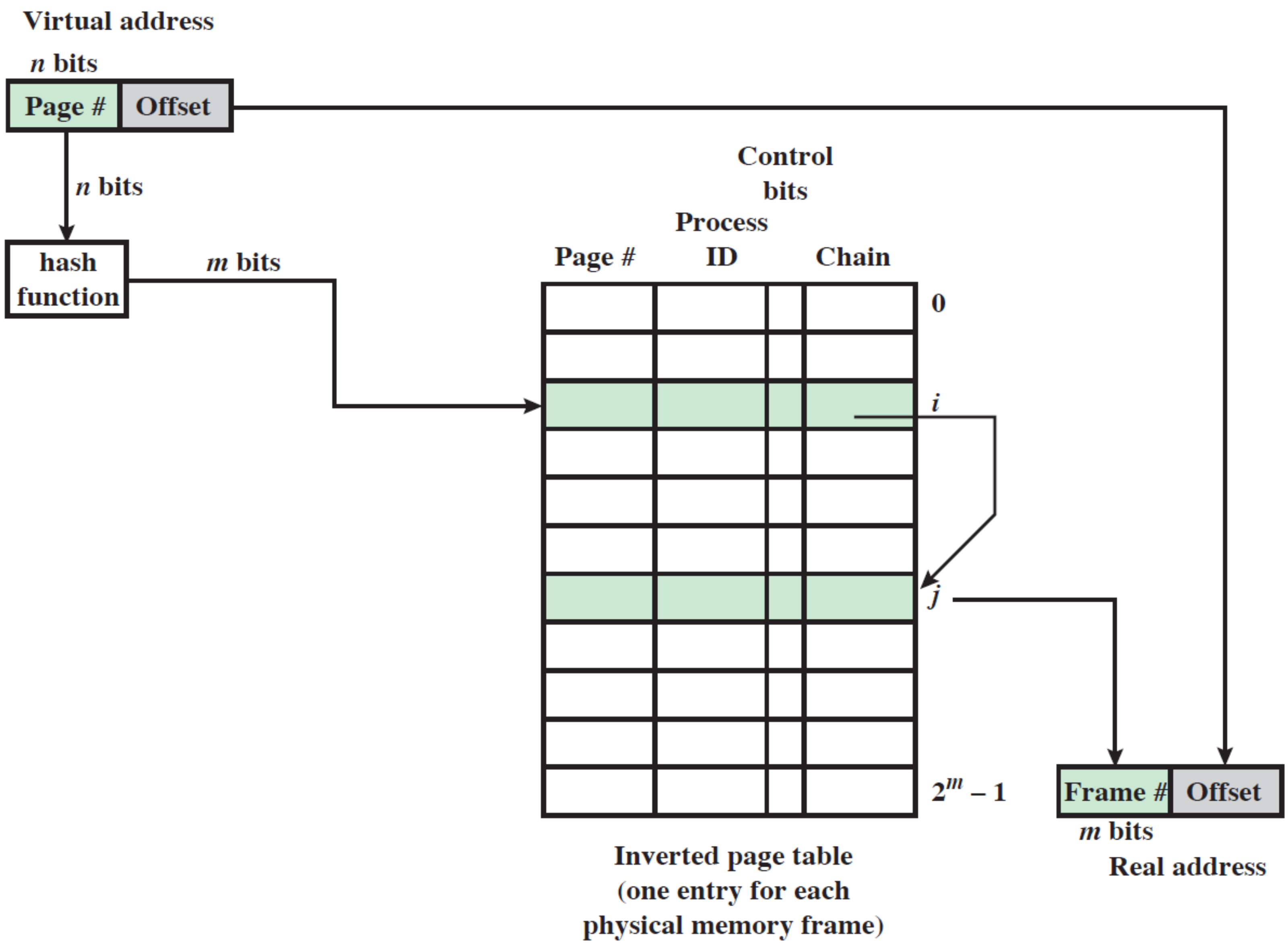
Handling the Page Tables

- In most systems, there is one page table per process
- Allowing each process to have up to 2^{31} = 2 Gbytes of virtual memory using 2^9 = 512-byte pages results in 2^{22} page table entries

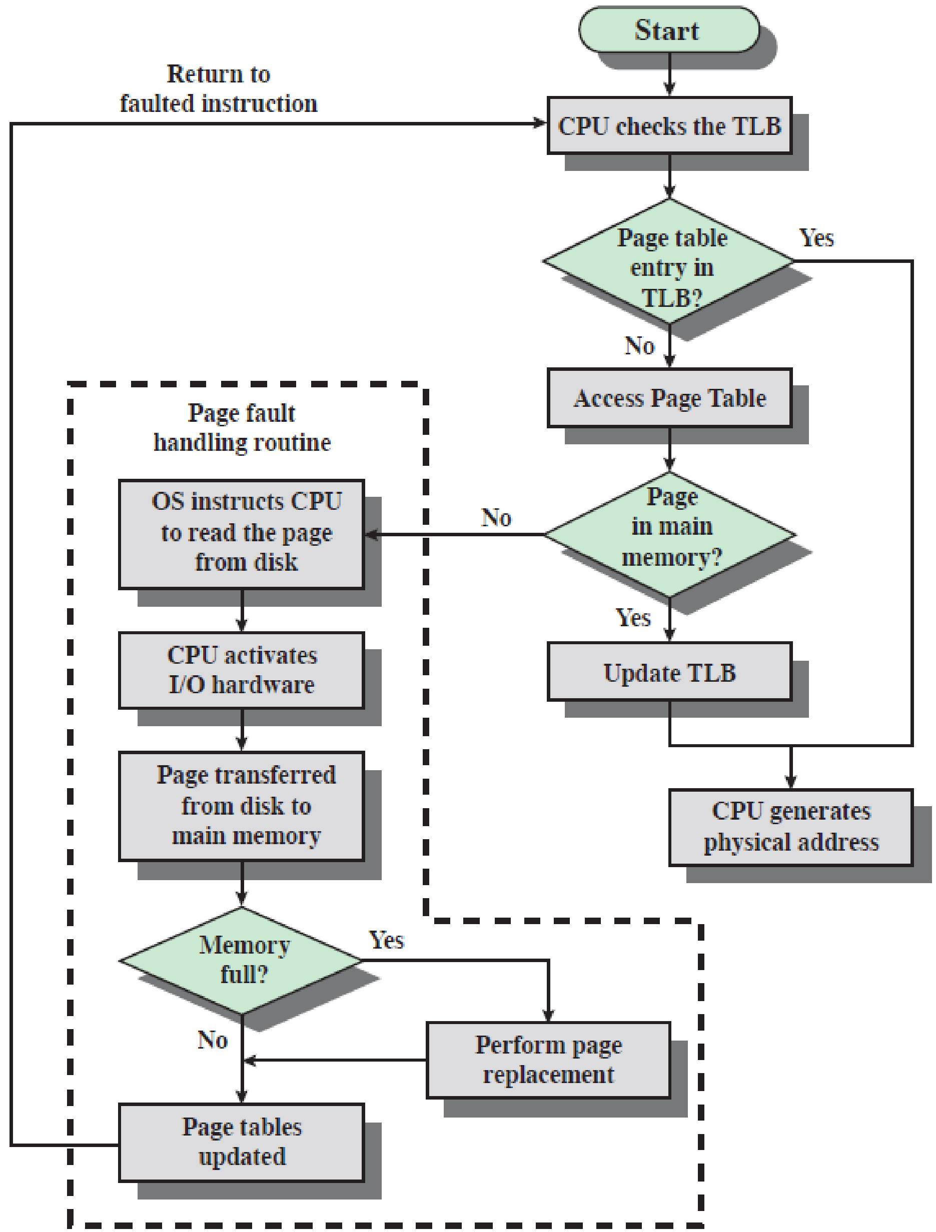
- To overcome this problem
  - store page tables in virtual memory rather than real memory
    - This means that page tables are subject to paging just as other pages are

  - Two-level scheme
    - There is a page directory, in which each entry points to a page table
    - Typically, the maximum length of a page table is restricted to be equal to one page

Inverted Page Table Structure



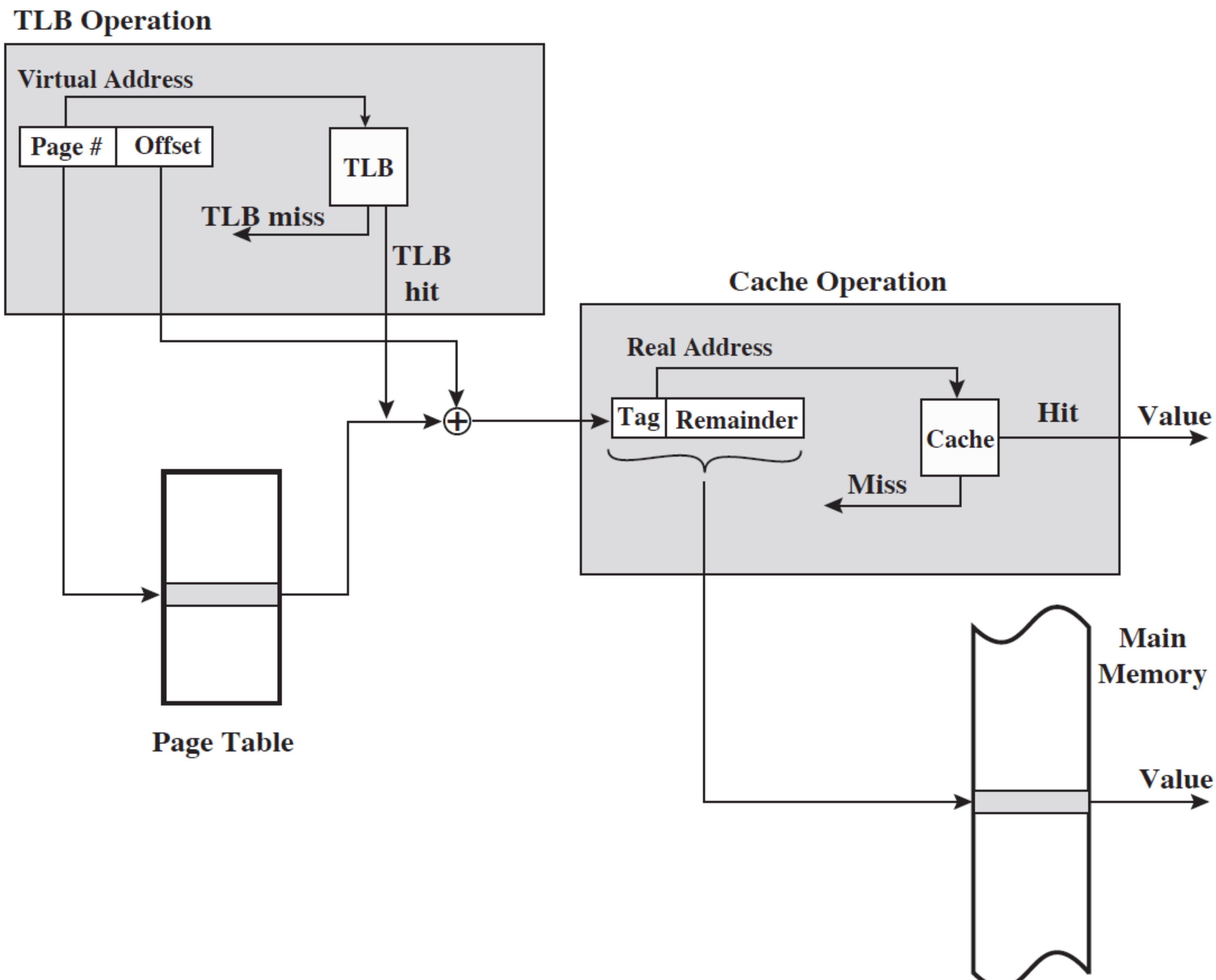
Paging and Translation Lookaside Buffer (TLB)



Where to put the cache?

- Where would you put the cache?
  - Before TLB using virtual addresses and PID?
  - After TLB using real addresses?
- What are the implications?

TLB & Cache


TLB Operation
Virtual Address
Page #
Offset
TLB
TLB miss
TLB
hit
Page Table
Cache Operation
Real Address
Tag
Remainder
Cache
Hit
Value
Miss
Main
Memory
Value

Operating System Support

Lecture Content

- OS in General
- Memory Management
- Virtual Memory
- Segmentation
- Memory Management of the Pentium II

Segmentation

- Usually visible to the programmer
- Provided as a convenience for organizing programs and data
- Means for associating privilege and protection attributes with instructions and data
- Allows the programmer to view memory as consisting of multiple address spaces or segments

Segmentation

- Usually visible to the programmer
- Provided as a convenience for organizing programs and data
- Means for associating privilege and protection attributes with instructions and data
- Allows the programmer to view memory as consisting of multiple address spaces or segments

- **Advantages**
  - Simplifies the handling of growing data structures
  - Allows programs to be altered and recompiled independently without requiring that an entire set of programs be re-linked and re-loaded
  - Lends itself to sharing among processes
  - Lends itself to protection

Different Operation Modes

- **Unsegmented Unpaged Memory**
- **Unsegmented Paged Memory**
- **Segmented Unpaged Memory**
- **Segmented Paged Memory**

Different Operation Modes

- **Unsegmented Unpaged Memory**
  - Virtual address is the same as the physical address
  - Useful in low-complexity, high performance controller applications
- **Unsegmented Paged Memory**
- **Segmented Unpaged Memory**
- **Segmented Paged Memory**

Different Operation Modes

- **Unsegmented Unpaged Memory**
- **Unsegmented Paged Memory**
  - Memory is viewed as a paged linear address space
  - Protection and management of memory is done via paging
  - Favored by some operating systems
- **Segmented Unpaged Memory**
- **Segmented Paged Memory**

Different Operation Modes

- **Unsegmented Unpaged Memory**
- **Unsegmented Paged Memory**
- **Segmented Unpaged Memory**
  - Memory is viewed as a collection of logical address spaces
  - Affords protection down to the level of a single byte
  - Guarantees that the translation table needed is on-chip when the segment is in memory
  - Results in predictable access times
- **Segmented Paged Memory**

Different Operation Modes

- **Unsegmented Unpaged Memory**
- **Unsegmented Paged Memory**
- **Segmented Unpaged Memory**
- **Segmented Paged Memory**
  - Segmentation is used to define logical memory partitions subject to access control, and paging is used to manage the allocation of memory within the partitions
  - Operating systems such as UNIX System V favor this view

Segmentation of the Pentium II

- Each virtual address consists of a 16-bit segment reference and a 32-bit offset
  - Two bits of segment reference deal with the protection mechanism
  - 14 bits specify segment

- Unsegmented virtual memory is 2^{32} = 4Gbytes
- Segmented virtual memory is 2^{46}=64 terabytes (Tbytes)
- Physical address space employs a 32-bit address for a maximum of 4 Gbytes
- Virtual address space is divided into two parts
  - One-half is global, shared by all processes
  - The remainder is local and is distinct for each process

Segment Protection

- Associated with each segment are two forms of protection:
  - Privilege level
  - Access attribute
- There are four privilege levels
  - Most protected (level 0)
  - Least protected (level 3)
- Privilege level of data segment is called "**classification**"
- Privilege level of a program is called "**clearance**"

- A program can only access data segments with classification $\leq$ clearance
- The privilege mechanism also limits the use of certain instructions

Paging on Pentium II

- Segmentation may be disabled
  - In which case linear address space is used

- Two level page table lookup
  - First, page directory
    - 1024 entries max
    - Splits 4 Gbyte linear memory into 1024 page groups of 4 Mbyte
    - Each page table has 1024 entries corresponding to 4 Kbyte pages
    - Can use one page directory for all processes, one per process or mixture
    - Page directory for current process always in memory
  - Use TLB holding 32 page table entries
  - Two page sizes available, 4k or 4M

Pentium II Address Translation Mechanism