

Contents

1. Introduction
2. System Structures
3. Process Concept
4. Multithreaded Programming
5. Process Scheduling
6. Synchronization
7. Deadlocks
8. Memory-Management Strategies
9. Virtual-Memory Management
10. File System
11. Mass-Storage Structures
12. I/O Systems
13. Protection, Security, Distributed Systems₉₈



Chapter 8

Memory-Management Strategies

Memory Management

- Motivation
 - Keep several processes in memory to improve a system's performance
- Selection of different memory management methods
 - Application-dependent
 - Hardware-dependent
- Memory – A large array of words or bytes, each with its own address.
 - Memory is always too small!

Memory Management

- The Viewpoint of the Memory Unit
 - A stream of memory addresses!
- What should be done?
 - Which areas are free or used (by whom)
 - Decide which processes to get memory
 - Perform allocation and de-allocation
- Remark:
 - Interaction between CPU scheduling and memory allocation!

Background

- Address Binding – binding of instructions and data to memory addresses

Binding Time

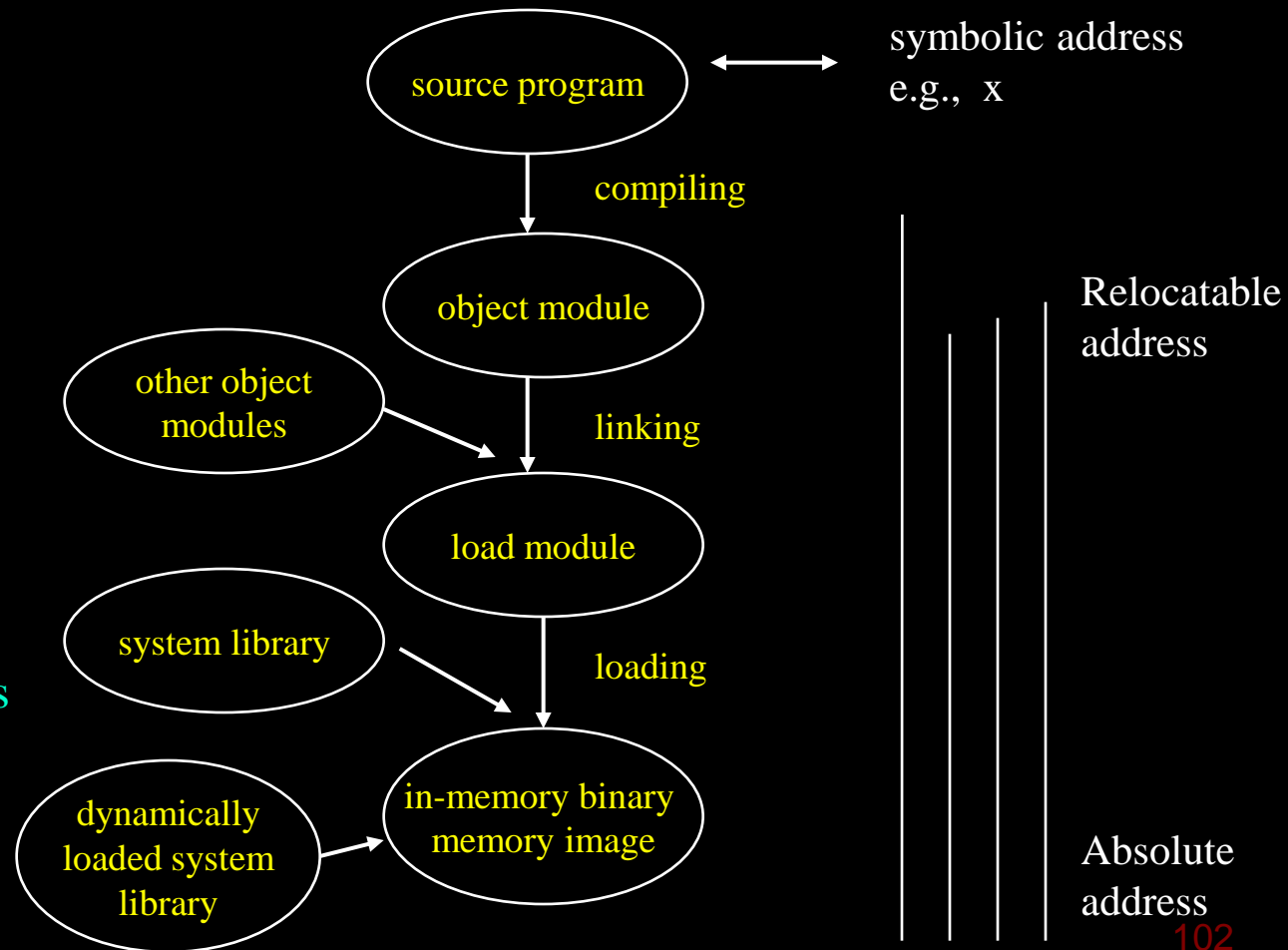
Known at compile time,
where a program will be in
memory - “absolute code”
MS-DOS *.COM

At load time:

- All memory reference by a program will be translated
- Code is relocatable
- Fixed while a program runs

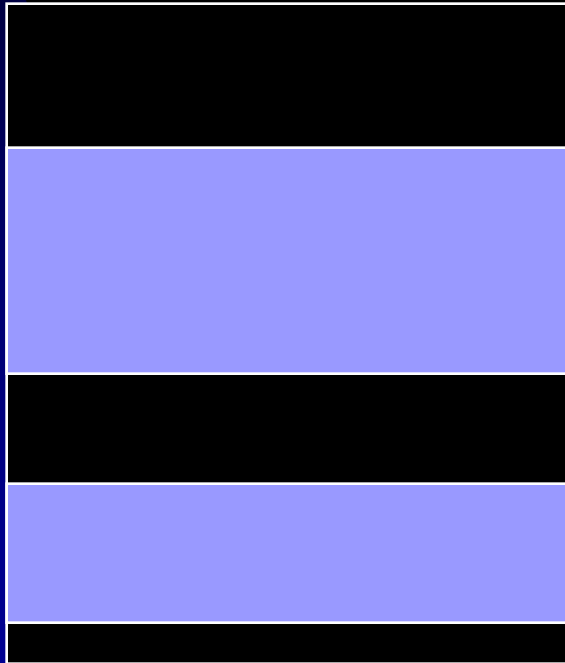
At execution time

- binding may change as a program run



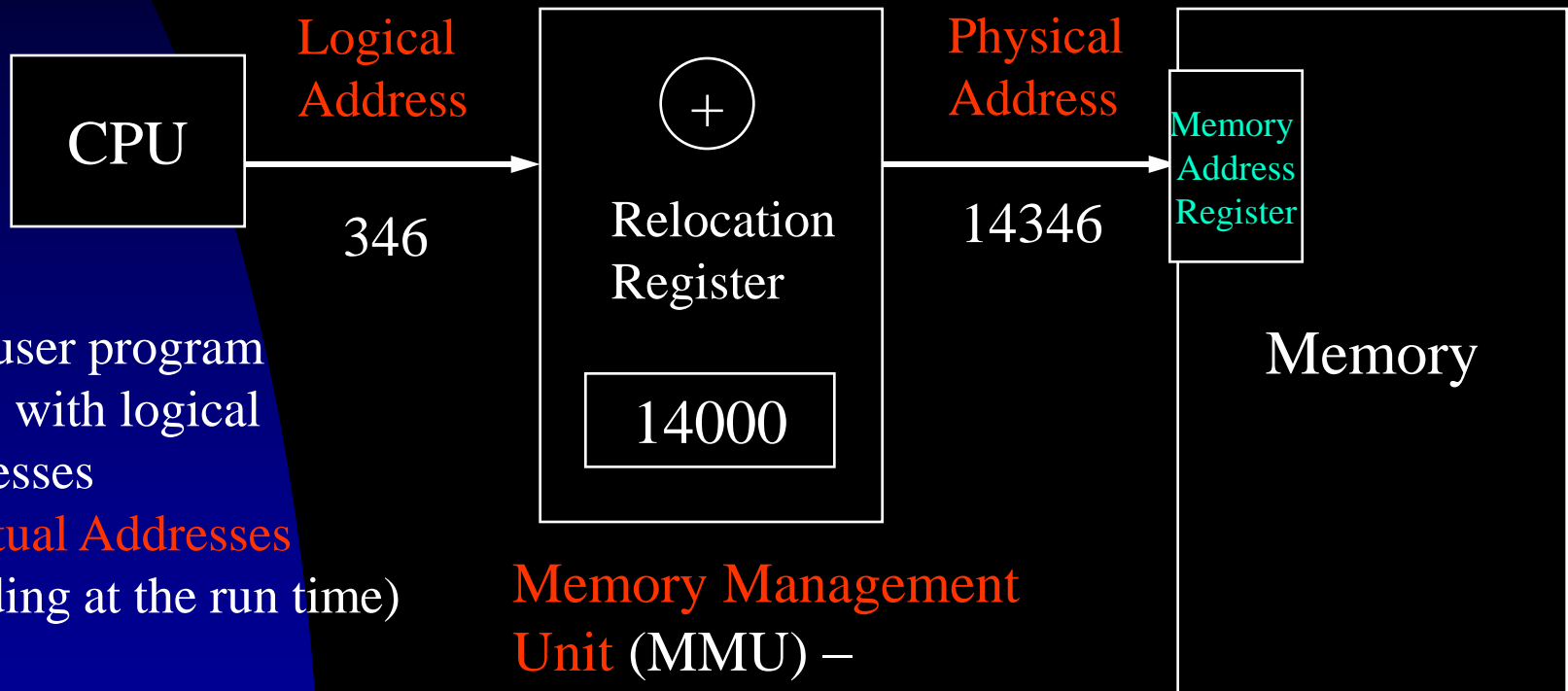
Background

Main
Memory



- Binding at the Compiling Time
 - A process must execute at a specific memory space
- Binding at the Load Time
 - Relocatable Code
- Process may move from a memory segment to another → binding is delayed till run-time

Logical Versus Physical Address



The user program deals with logical addresses

- **Virtual Addresses**
(binding at the run time)

Memory Management Unit (MMU) –
“Hardware-Support”

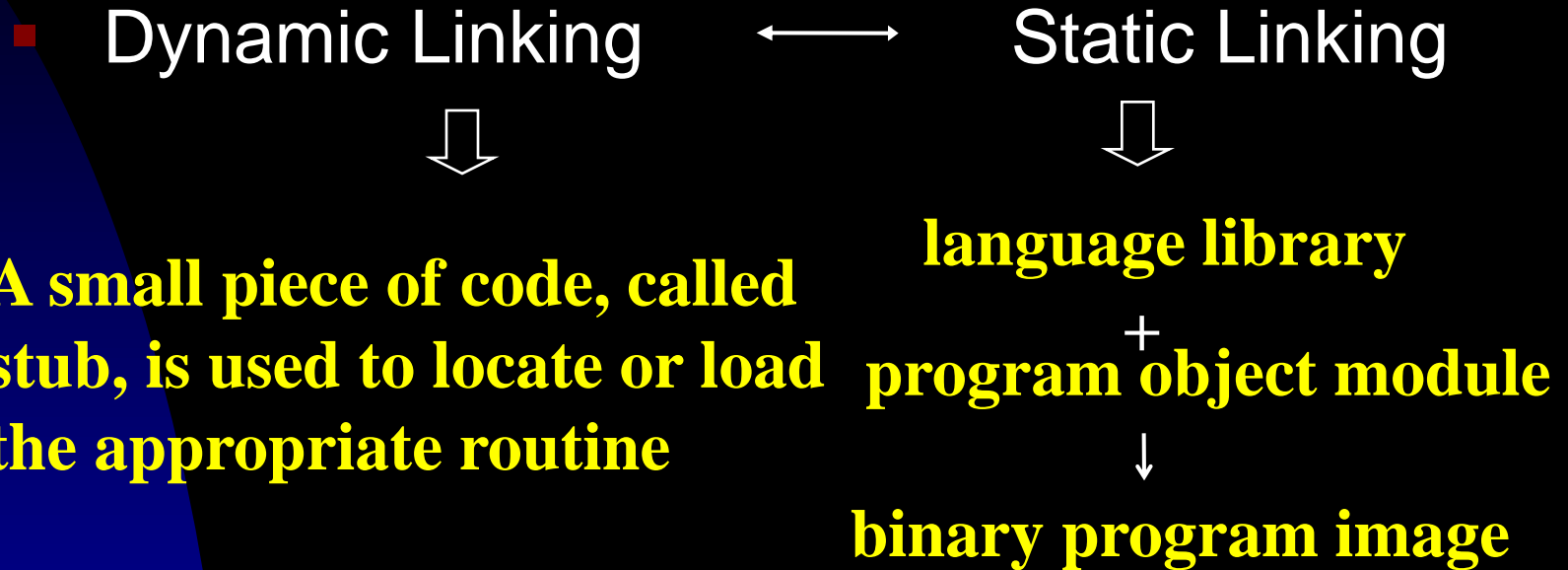
Logical Versus Physical Address

- A logical (physical) address space is the set of logical (physical) addresses generated by a process. Physical addresses of a program is transparent to any process!
- MMU maps from virtual addresses to physical addresses. Different memory mapping schemes need different MMU's that are hardware devices. (slow down)
- Compile-time & load-time binding schemes results in the collapsing of logical and physical address spaces.

Dynamic Loading

- Dynamic Loading
 - A routine will not be loaded until it is called. A relocatable linking loader must be called to load the desired routine and change the program's address tables.
 - Advantage
 - Memory space is better utilized.
 - Users may use OS-provided libraries to achieve dynamic loading

Dynamic Linking



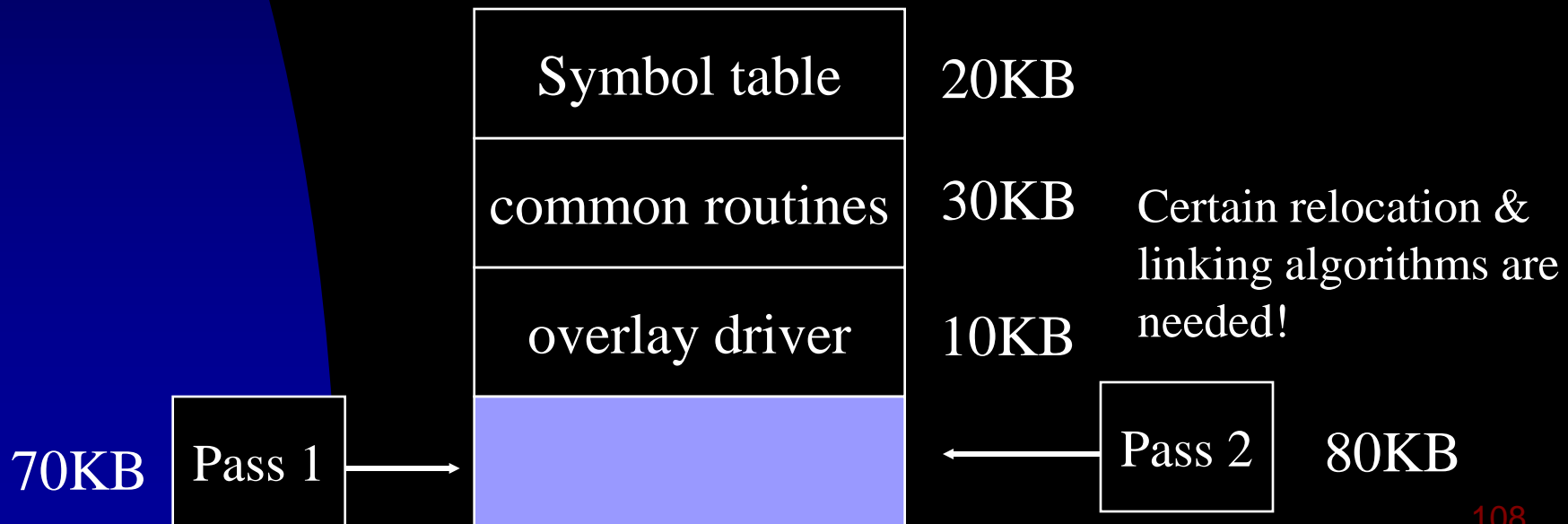
Advantage

Save memory space by sharing the library code among processes → Memory Protection & Library Update!

Simple

Overlays

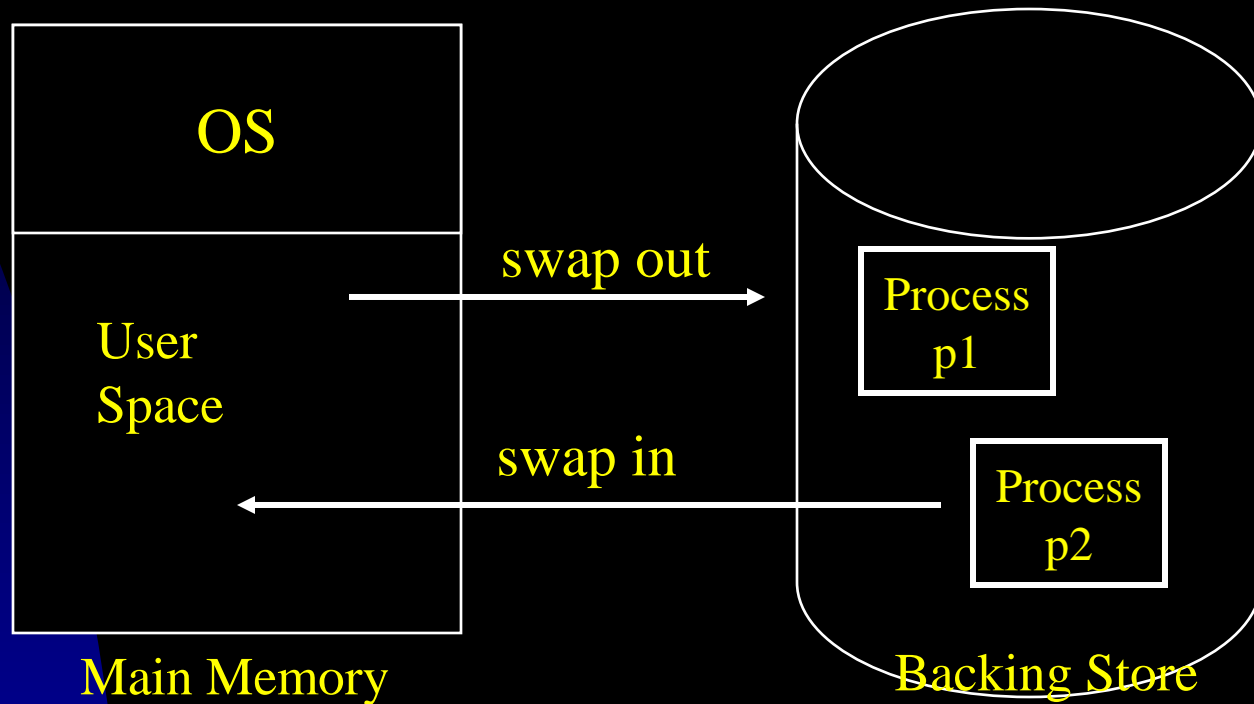
- Motivation
 - Keep in memory only those instructions and data needed at any given time.
 - Example: Two overlays of a two-pass assembler



Overlays

- Memory space is saved at the cost of run-time I/O.
- Overlays can be achieved w/o OS support:
 - ⇒ “absolute-address” code
- However, it’s not easy to program a overlay structure properly!
 - ⇒ Need some sort of automatic techniques that run a large program in a limited physical memory!

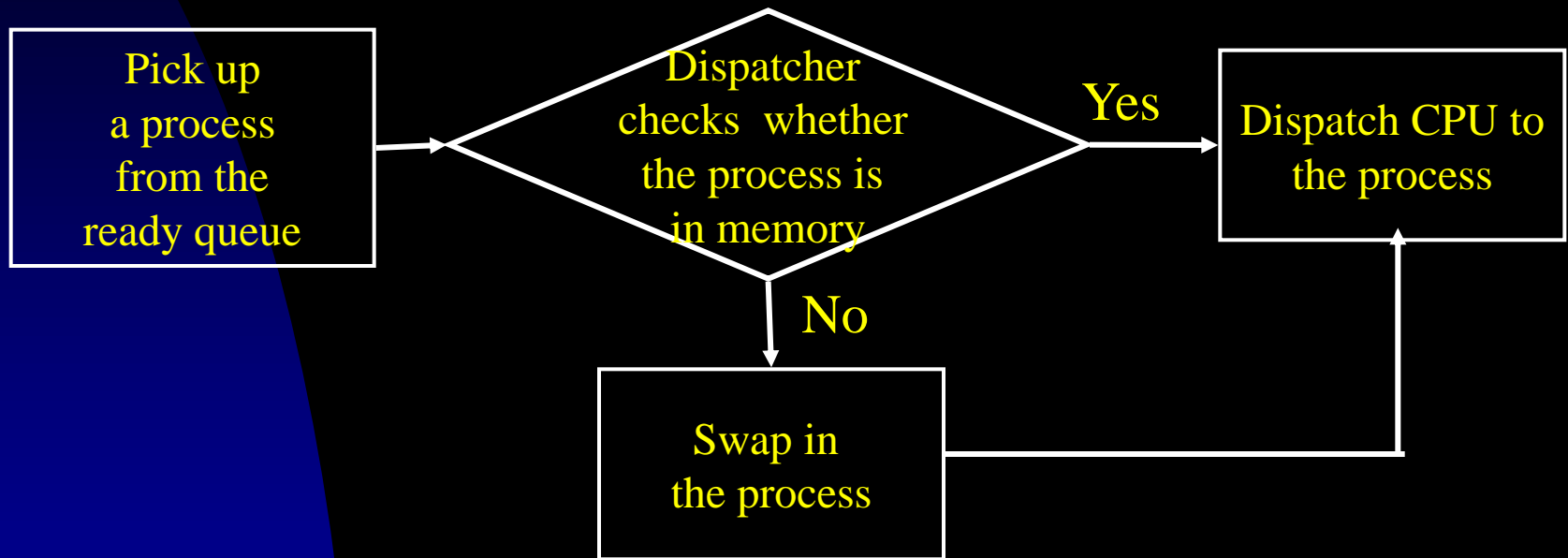
Swapping



Should a process be put back into the same memory space that it occupied previously?
↔ Binding Scheme?!

Swapping

■ A Naive Way

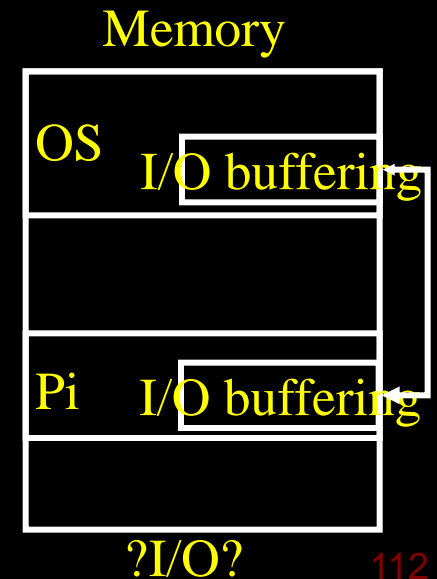


Potentially High Context-Switch Cost:

$$2 * (\underbrace{10000\text{KB}/50\text{MBps}}_{\text{Transfer Time}} + \underbrace{8\text{ms}}_{\text{Latency Delay}}) = 416\text{ms}$$

Swapping

- The execution time of each process should be long relative to the swapping time in this case (e.g., 416ms in the last example)!
- Only swap in what is actually used. \Rightarrow Users must keep the system informed of memory usage.
- Who should be swapped out?
 - “Lower Priority” Processes?
 - Any Constraint?
 \Rightarrow System Design



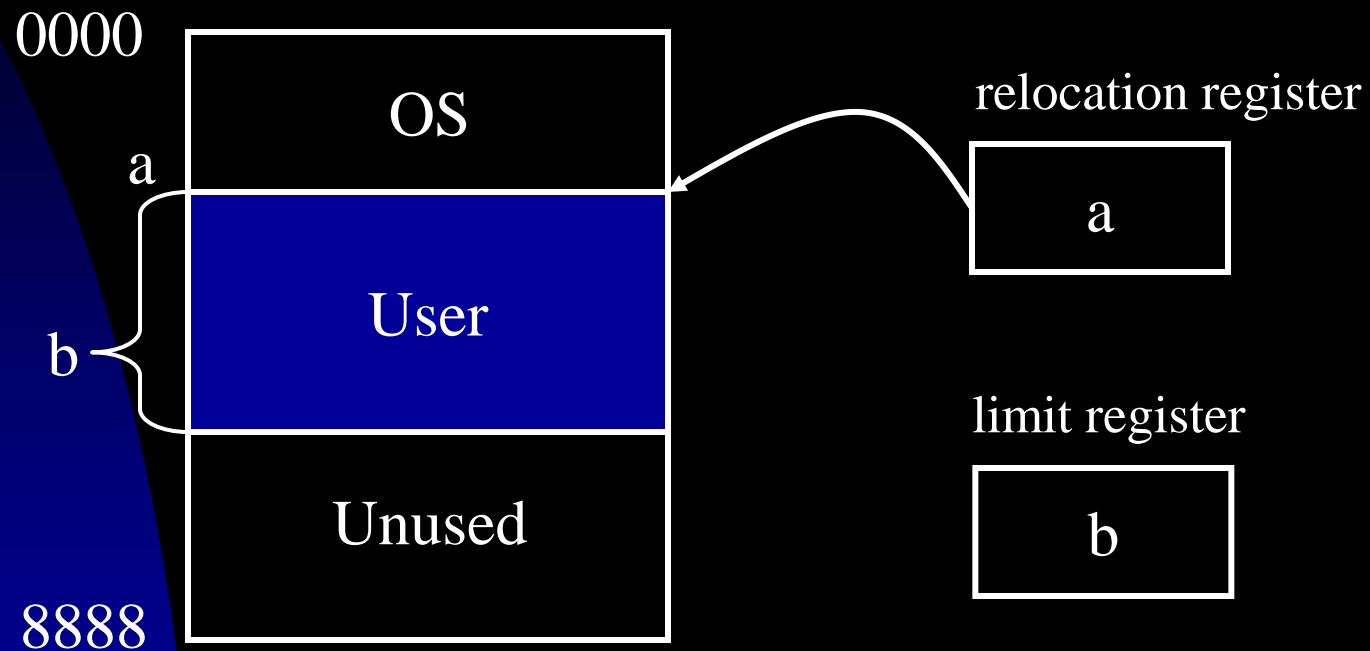
Swapping

- Separate swapping space from the file system for efficient usage
- Disable swapping whenever possible such as many versions of UNIX – Swapping is triggered only if the memory usage passes a threshold, and many processes are running!
- In Windows 3.1, a swapped-out process is not swapped in until the user selects the process to run.

Swapping

- Mobile systems do not support swapping in general but might have paging (iOS and Android)
 - Limited space for storage
 - Limited writes due to endurance constraints
- Strategies for Memory Management of Mobile Systems
 - Applications voluntarily relinquish allocated memory (iOS).
 - Terminate applications when there is no sufficient memory (application state storing?)

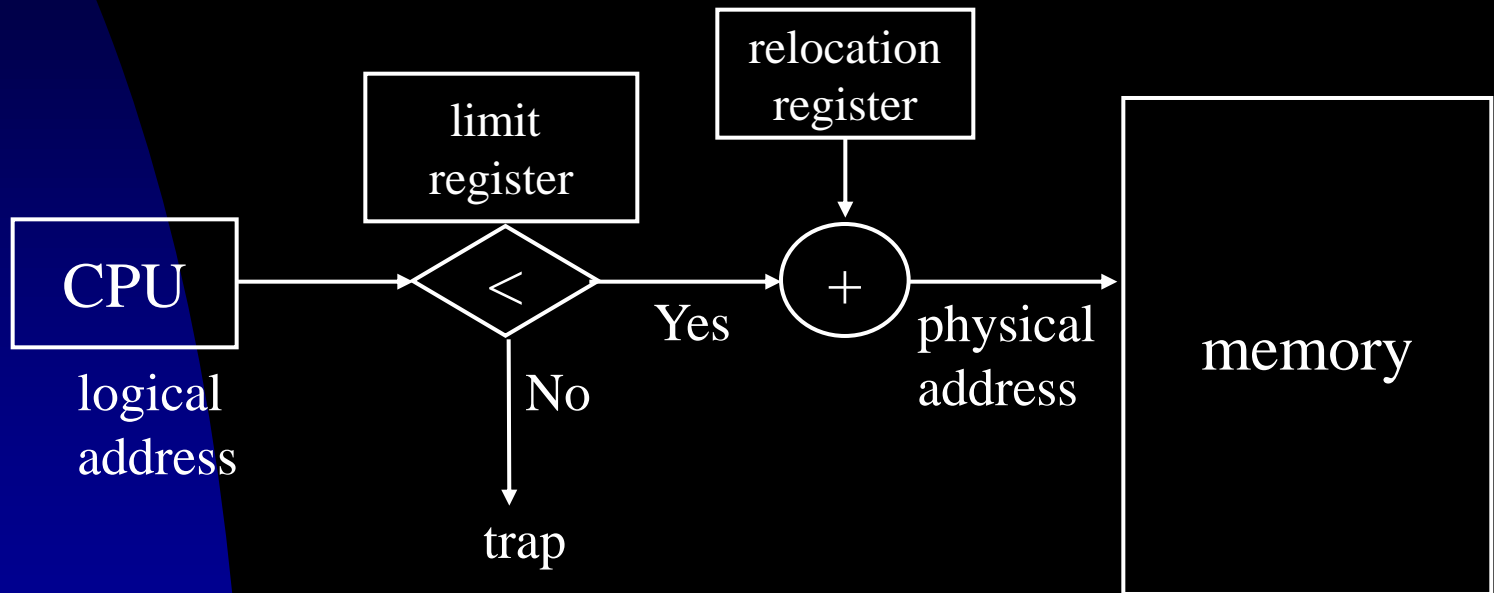
Contiguous Allocation – Single User



- A single user is allocated as much memory as needed
- Problem: Size Restriction → Overlays (MS/DOS)

Contiguous Allocation – Single User

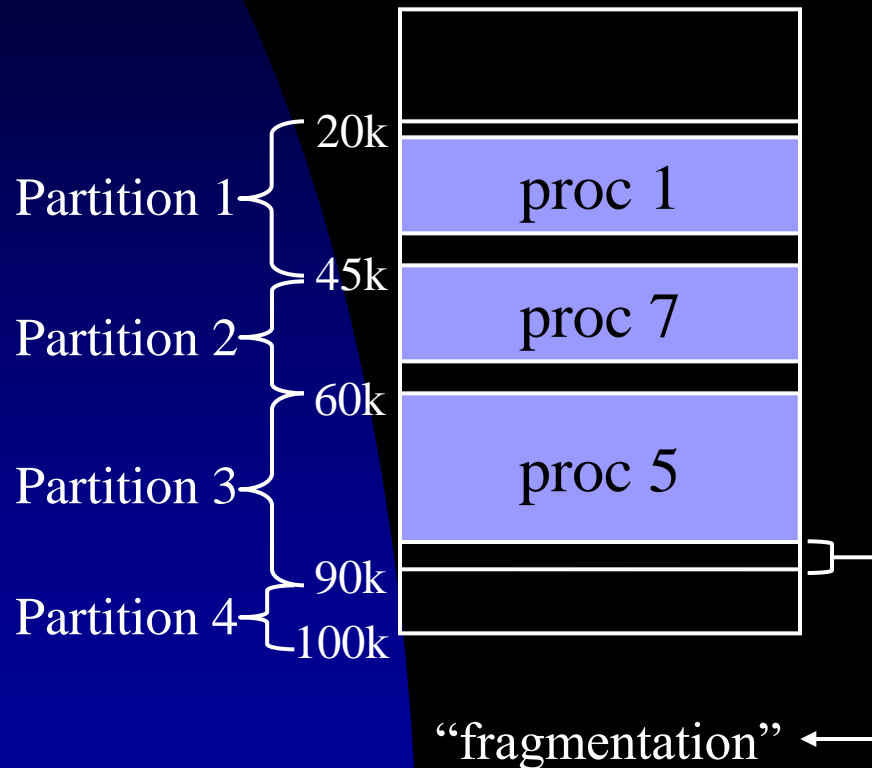
- Hardware Support for Memory Mapping and Protection



Disadvantage: Wasting of CPU and Resources
∴ No Multiprogramming Possible

Contiguous Allocation – Multiple Users

■ Fixed Partitions



- Memory is divided into fixed partitions, e.g., OS/360 (or MFT)
- A process is allocated on an entire partition
- An OS Data Structure:

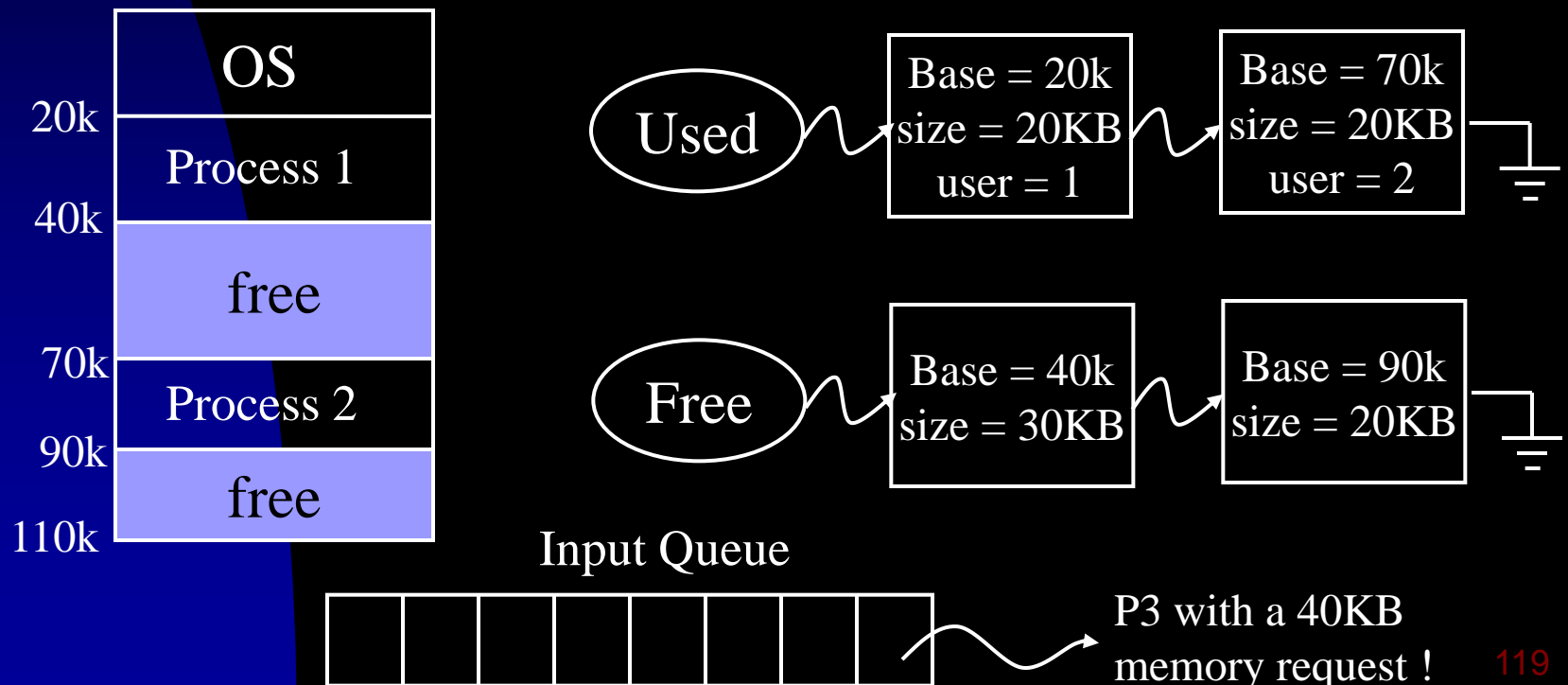
| Partitions | | | |
|------------|------|----------|--------|
| # | size | location | status |
| 1 | 25KB | 20k | Used |
| 2 | 15KB | 45k | Used |
| 3 | 30KB | 60k | Used |
| 4 | 10KB | 90k | Free |

Contiguous Allocation – Multiple Users

- Hardware Supports
 - Bound registers
 - Each partition may have a protection key (corresponding to a key in the current PSW)
- Disadvantage:
 - Fragmentation gives poor memory utilization !

Contiguous Allocation – Multiple Users

- Dynamic Partitions
 - Partitions are dynamically created.
 - OS tables record free and used partitions



Contiguous Allocation – Multiple Users

- Solutions for dynamic storage allocation :
 - First Fit – Find a hole which is big enough
 - Advantage: Fast and likely to have large chunks of memory in high memory locations
 - Best Fit – Find the smallest hole which is big enough. → It might need a lot of search time and create lots of small fragments !
 - Advantage: Large chunks of memory available
 - Worst Fit – Find the largest hole and create a new partition out of it!
 - Advantage: Having largest leftover holes with lots of search time!

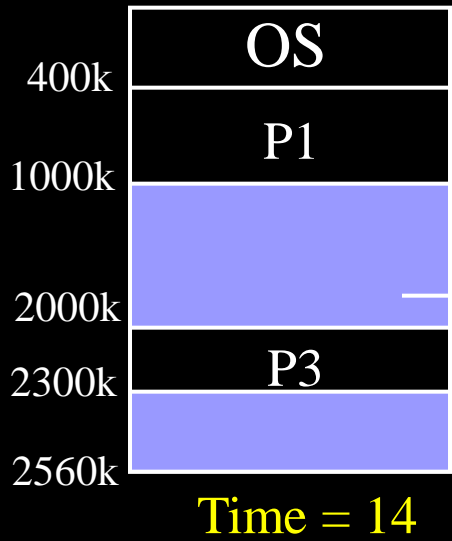
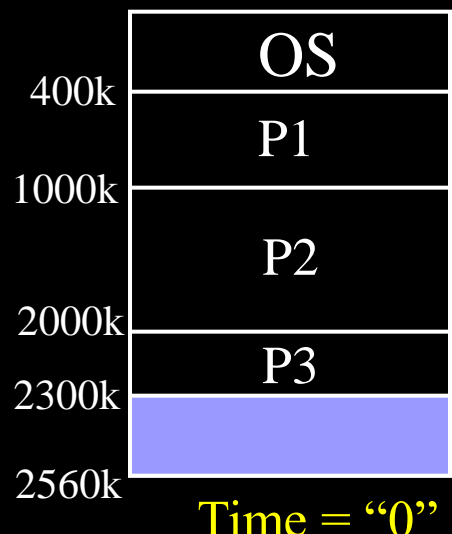
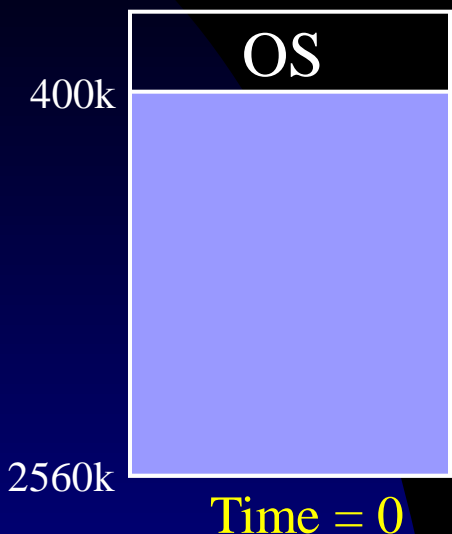
Better
in Time
and Storage
Usage

Contiguous Allocation Example – First Fit

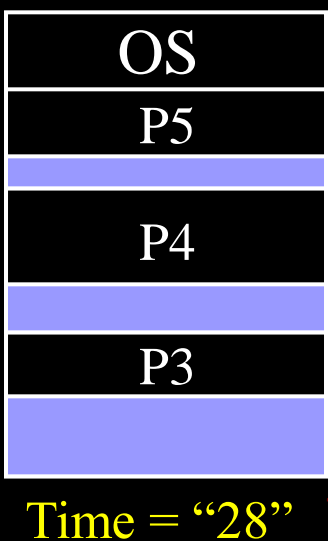
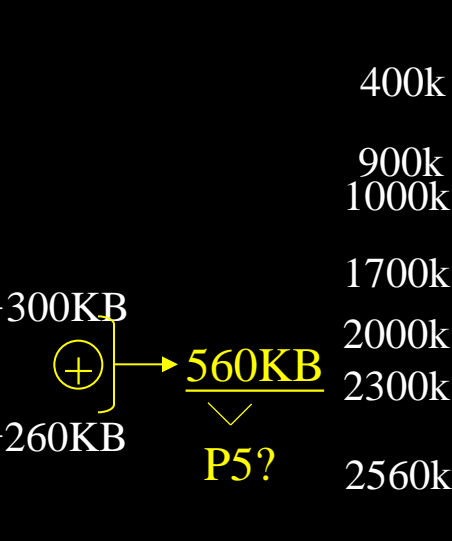
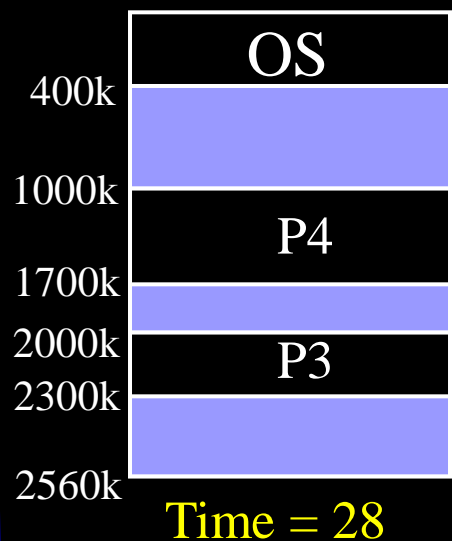
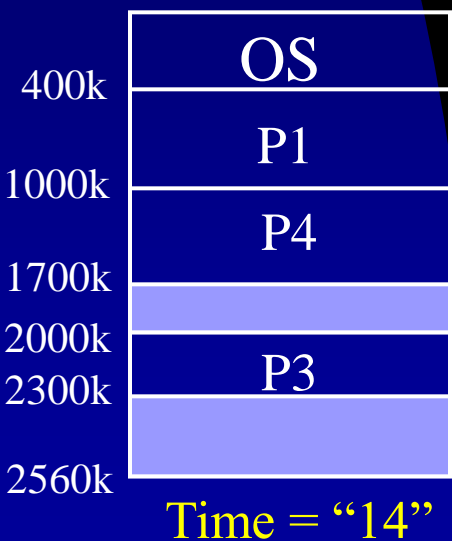
(RR Scheduler with Quantum = 1)

A job queue

| Process | Memory | Time |
|---------|--------|------|
| P1 | 600KB | 10 |
| P2 | 1000KB | 5 |
| P3 | 300KB | 20 |
| P4 | 700KB | 8 |
| P5 | 500KB | 15 |



P2 terminates & frees its memory



560KB
P5?

Fragmentation – Dynamic Partitions

- External fragmentation occurs as small chunks of memory accumulate as a by-product of partitioning due to imperfect fits.
- Statistical Analysis For the First-Fit Algorithm:
 - 1/3 memory is unusable – 50-percent rule
- Solutions:
 - Merge adjacent free areas.
 - Compaction
 - Compact all free areas into one contiguous region
 - Requires user processes to be relocatable

Any optimal compaction strategy???

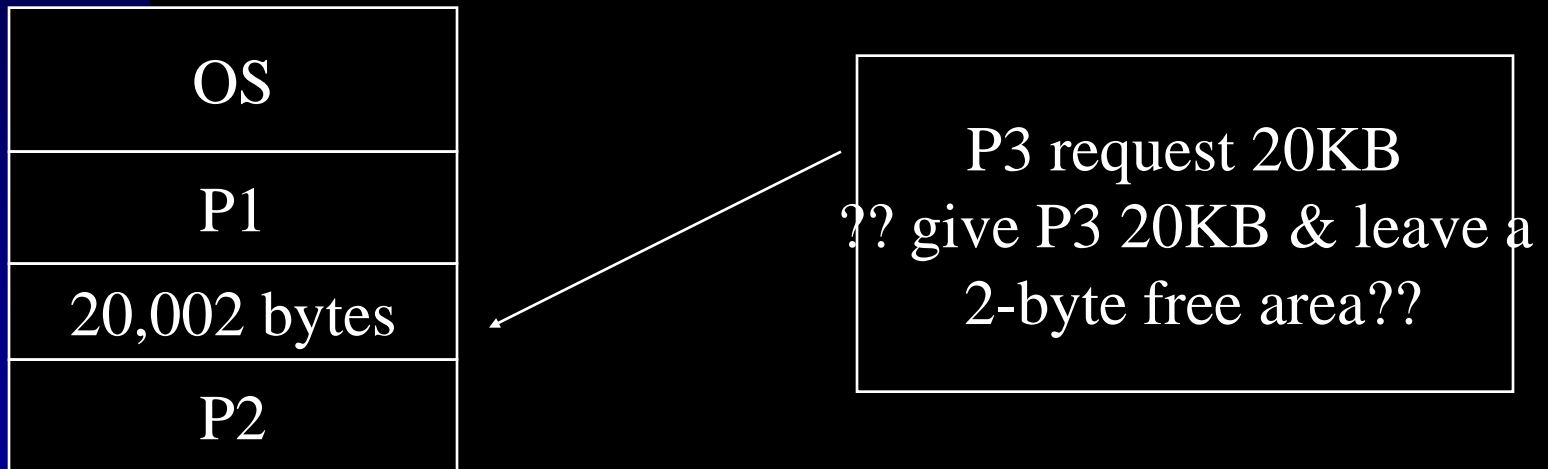
Fragmentation – Dynamic Partitions



- Cost: Time Complexity $O(n!)$??!
- Combination of swapping and compaction
 - Dynamic/static relocation

Fragmentation – Dynamic Partitions

- Internal fragmentation:
A small chunk of “unused” memory internal to a partition.



Reduce free-space maintenance cost

→ Give 20,002 bytes to P3 and have 2 bytes as an internal fragmentation!

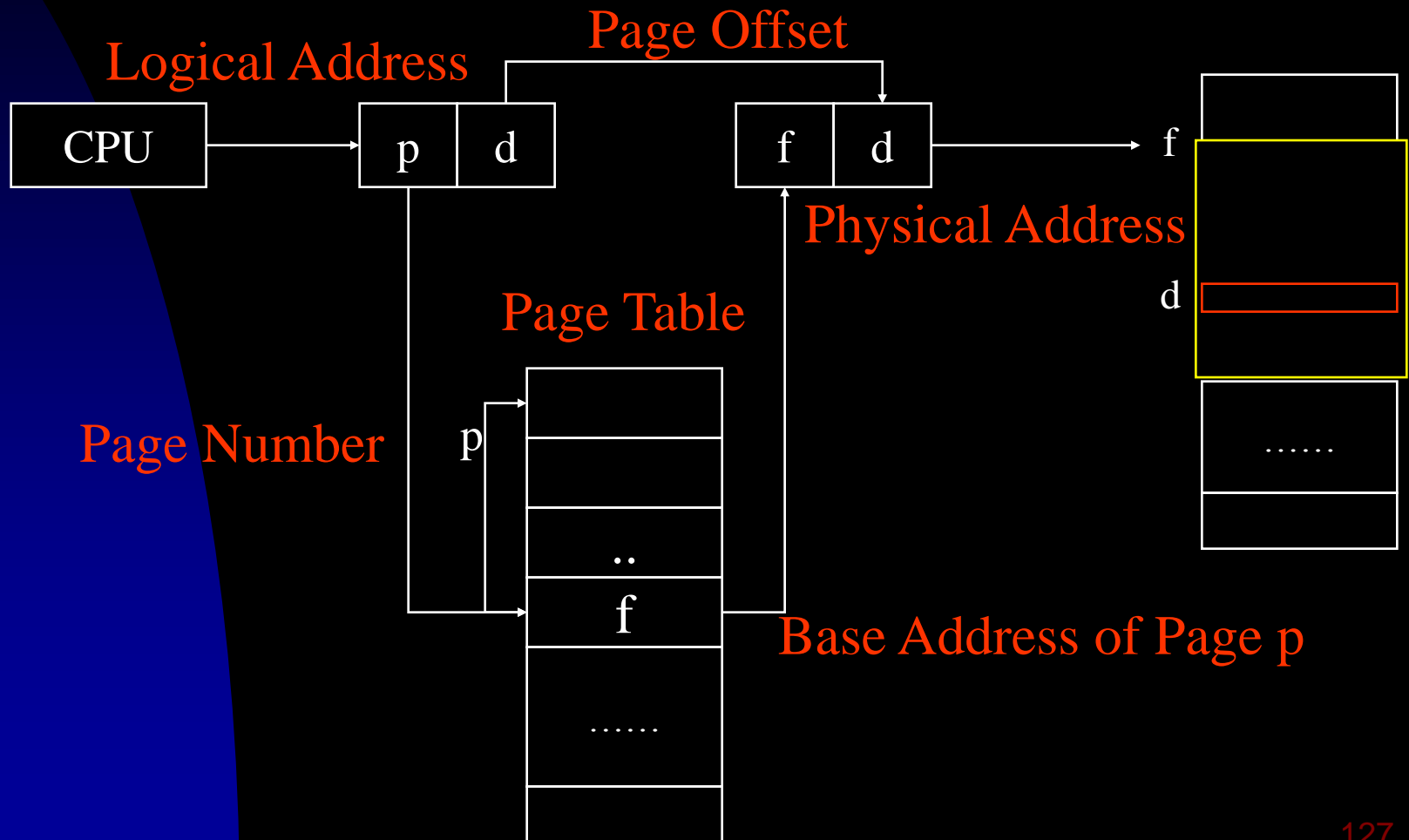
Fragmentation – Dynamic Partitions

- Dynamic Partitioning:
 - Advantage:
 - ⇒ Eliminate fragmentation to some degree
 - ⇒ Can have more partitions and a higher degree of multiprogramming
 - Disadvantage:
 - Compaction vs Fragmentation
 - The amount of free memory may not be enough for a process! (contiguous allocation)
 - Memory locations may be allocated but never referenced.
 - Relocation Hardware Cost & Slow Down
- ⇒ Solution: Paged Memory!

Paging

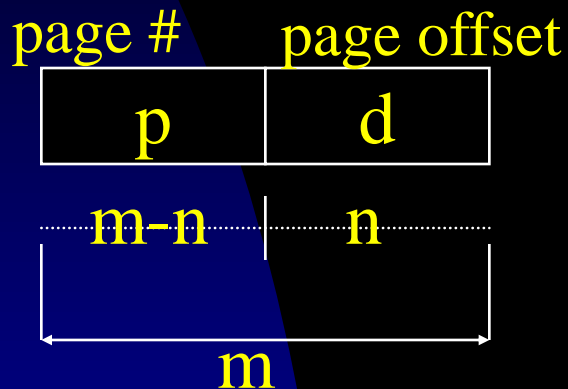
- Objective
 - Users see a logically contiguous address space although its physical addresses are throughout physical memory
- Units of Memory and Backing Store
 - Physical memory is divided into fixed-sized blocks called *frames*.
 - The logical memory space of each process is divided into blocks of the same size called *pages*.
 - The backing store is also divided into blocks of the same size if used.

Paging – Basic Method



Paging – Basic Method

- Address Translation



max number of pages: 2^{m-n}

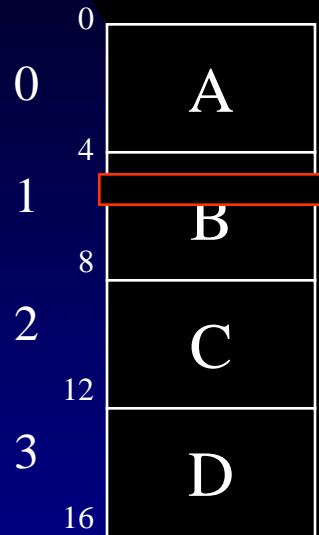
Logical Address Space: 2^m

Physical Address Space: ???

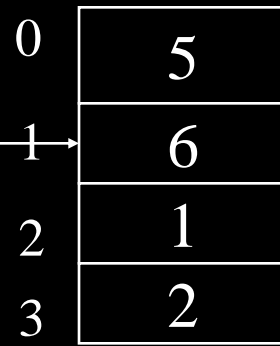
- A page size tends to be a power of 2 for efficient address translation.
- The actual page size depends on the computer architecture. Today, it is from 512B or 16KB.

Paging – Basic Method

Page



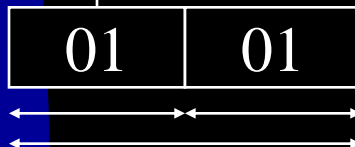
Logical
Memory



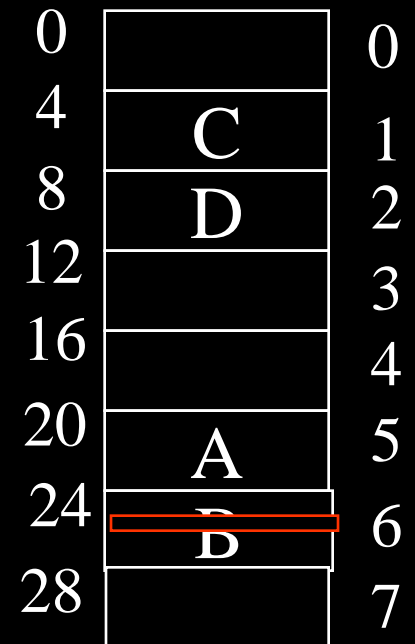
Page
Table

Logical Address

$$1 * 4 + 1 = 5$$



Frame



Physical Memory

Physical Address
= $6 * 4 + 1 = 25$

Paging – Basic Method

- No External Fragmentation
 - Paging is a form of dynamic relocation.
 - The average internal fragmentation is about one-half page per process
- The page size generally grows over time as processes, data sets, and memory have become larger.
 - 4-byte page table entry & 4KB per page → $2^{32} * 2^{12}B = 2^{44}B = 16TB$ of physical memory

Page Size

Disk I/O
Efficiency

Page Table
Maintenance

Internal
Fragmentation

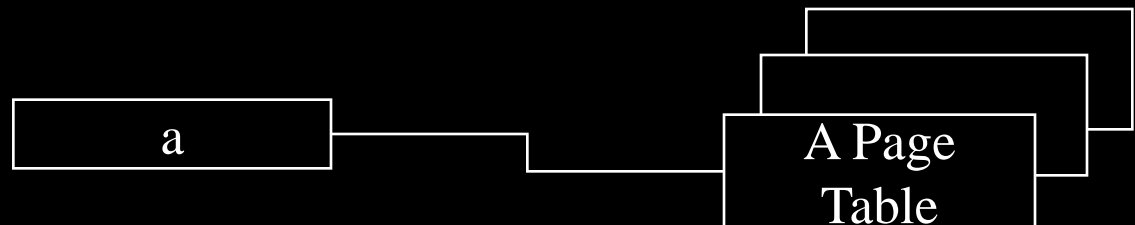
* Example: 8KB or 4MB for Solaris.

Paging – Basic Method

- Page Replacement:
 - An executing process has all of its pages in physical memory.
- Maintenance of the Frame Table
 - One entry for each physical frame
 - The status of each frame (free or allocated) and its owner
- The page table of each process must be saved when the process is preempted. → Paging increases context-switch time!

Paging – Hardware Support

- Page Tables
 - Where: Registers or Memory
 - Efficiency is the main consideration!
 - The use of registers for page tables
 - The page table must be small!
 - The use of memory for page tables
 - Page-Table Base Register (PTBR)

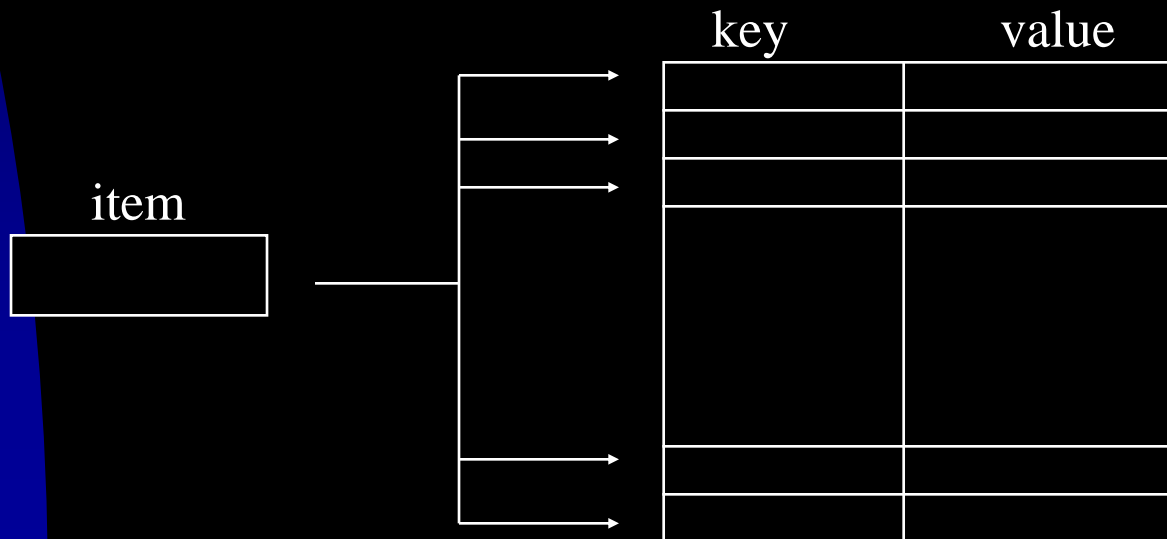


Paging – Hardware Support

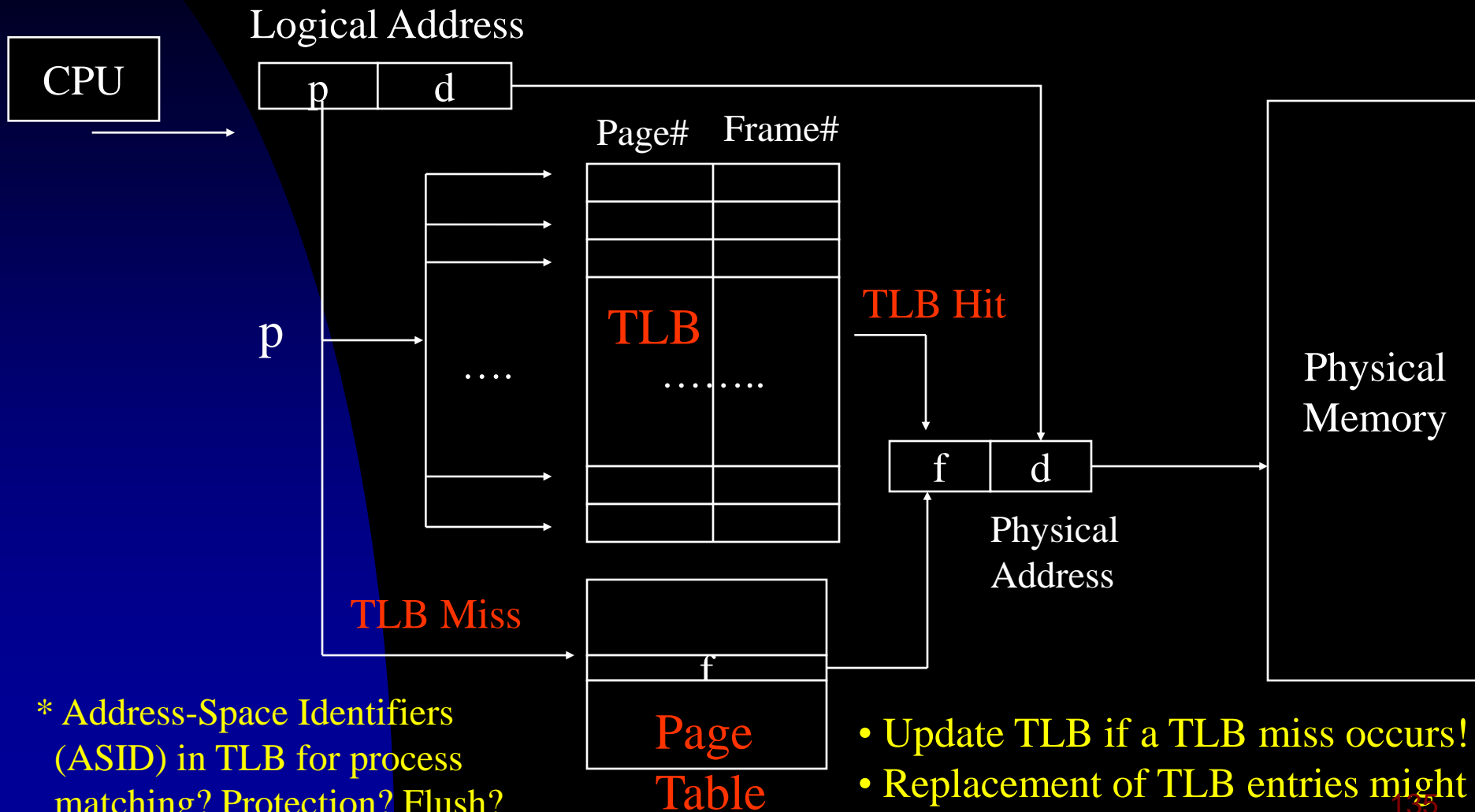
- Page Tables on Memory
 - Advantages:
 - The size of a page table is unlimited!
 - The context switch cost may be low if the CPU dispatcher merely changes PTBR, instead of reloading another page table.
 - Disadvantages:
 - Memory access is slowed by a factor of 2
 - Translation Look-aside buffers (TLB)
 - Associate, high-speed memory
 - (key/tag, value) – 16 ~ 1024 entries
 - Less than 10% memory access time

Paging – Hardware Support

- Translation Look-aside Buffers(TLB):
 - Disadvantages: Expensive Hardware and Flushing of Contents for Switching of Page Tables
 - Advantage: Fast – Constant-Search Time



Paging – Hardware Support



* Address-Space Identifiers (ASID) in TLB for process matching? Protection? Flush?

- Update TLB if a TLB miss occurs!
- Replacement of TLB entries might be needed.

Paging – Effective Memory Access Time

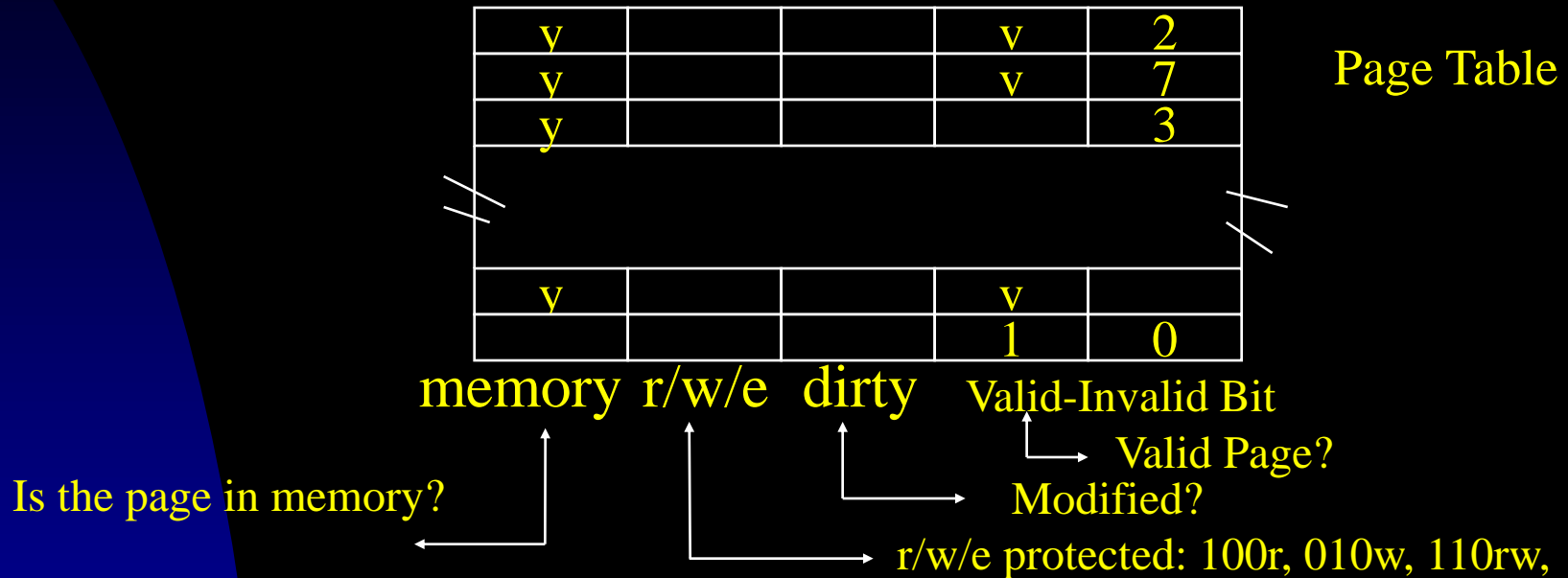
- Hit Ratio = the percentage of times that a page number is found in the TLB
 - The hit ratio of a TLB largely depends on the size and the replacement strategy of TLB entries!
- Effective Memory Access Time
 - $\text{Hit-Ratio} * (\text{TLB lookup} + \text{a mapped memory access}) + (1 - \text{Hit-Ratio}) * (\text{TLB lookup} + \text{a page table lookup} + \text{a mapped memory access})$

Paging – Effective Memory Access Time

- An Example
 - 20ns per TLB lookup, 100ns per memory access
 - Effective Access Time = $0.8 \times 120\text{ns} + 0.2 \times 220\text{ns} = 140\text{ ns}$, when hit ratio = 80%
 - Effective access time = $0.98 \times 120\text{ns} + 0.02 \times 220\text{ns} = 122\text{ ns}$, when hit ratio = 98%
- Intel 486 has a 32-register TLB and claims a 98 percent hit ratio.

Paging – Protection & Sharing

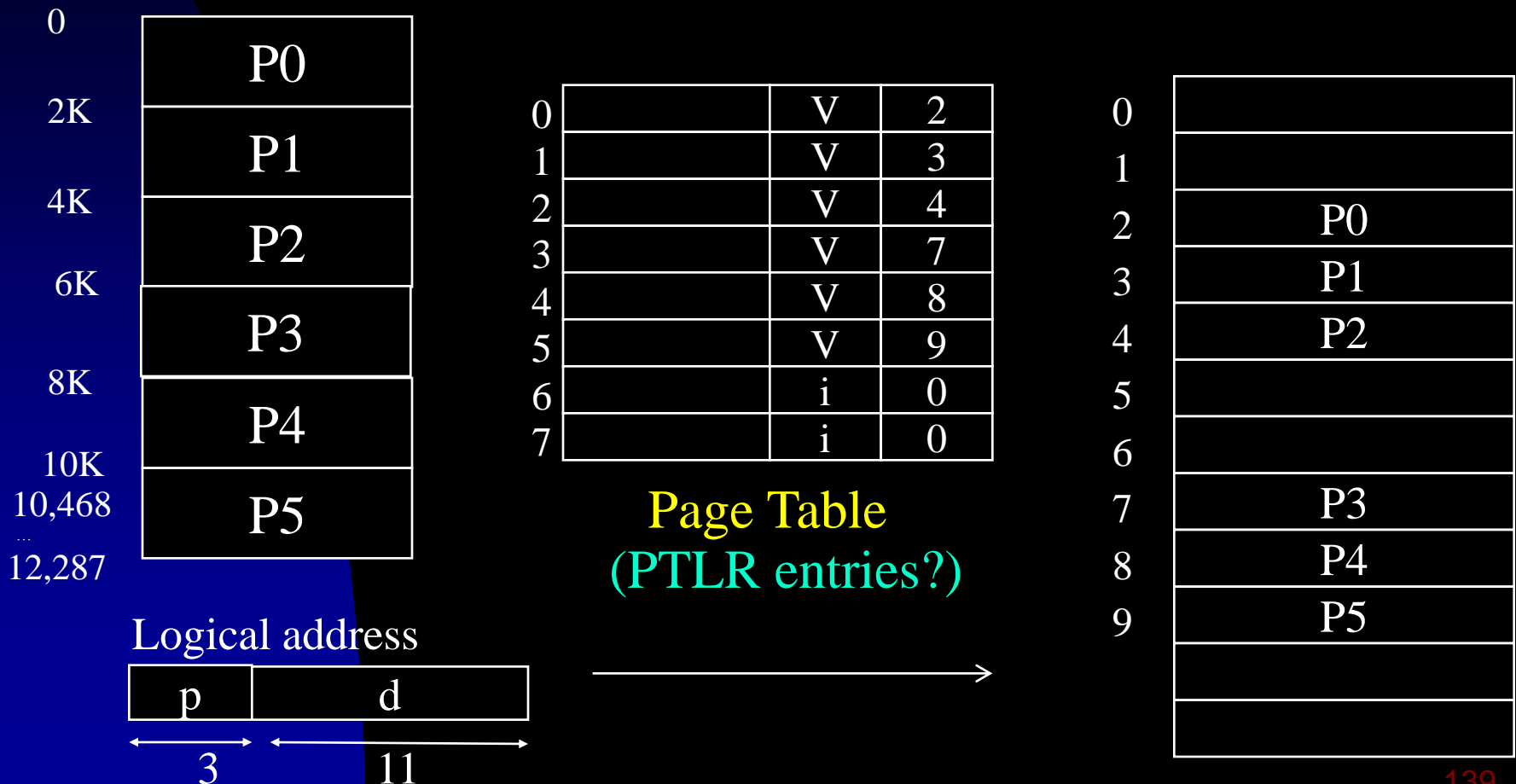
■ Protection



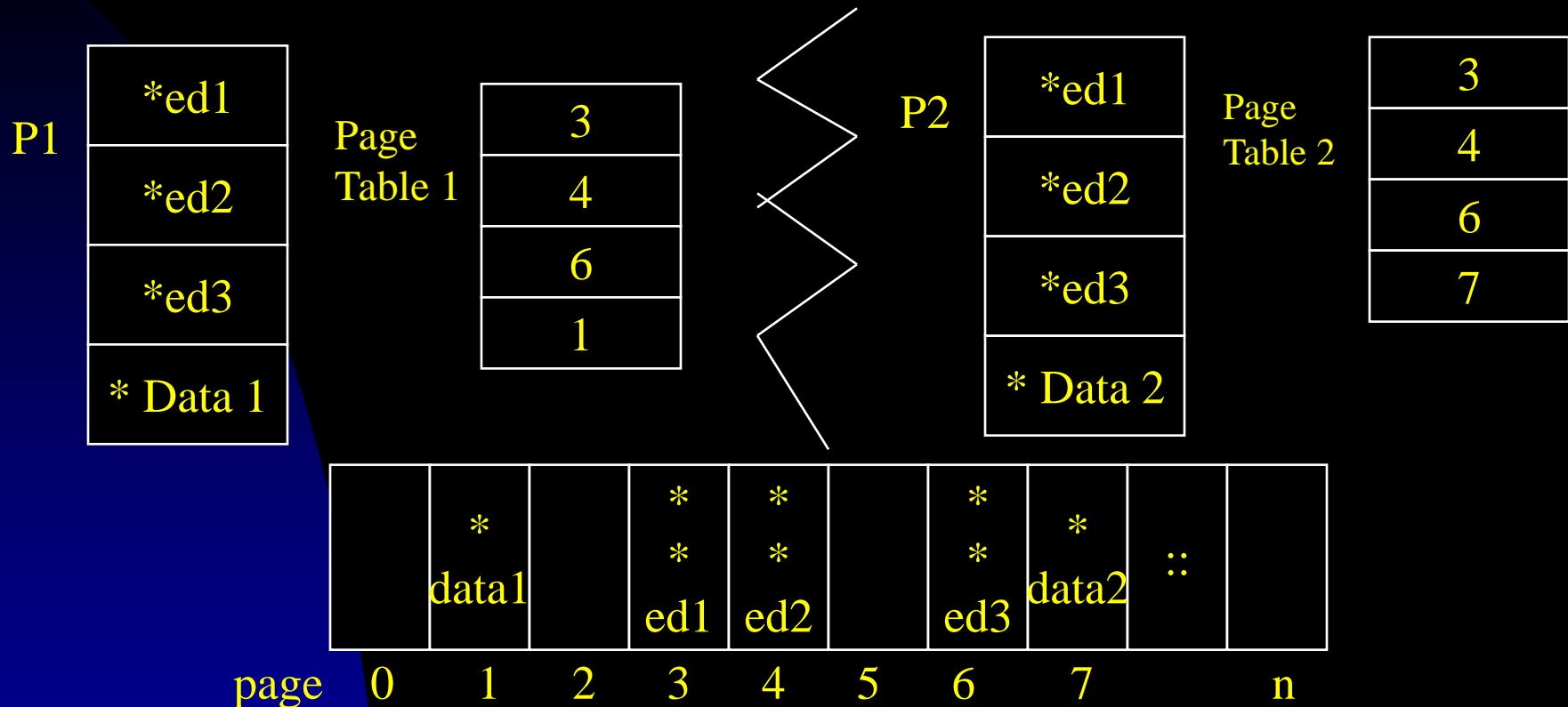
- Use a Page-Table Length Register (PTLR) to indicate the size of the page table.
- Unused Paged table entries might be ignored during maintenance.

Paging – Protection & Sharing

- Example: a 12287-byte Process ($16384=2^{14}$)



Paging – Protection & Sharing



- Procedures which are executed often (e.g., editor) can be divided into procedure + data. Memory can be saved a lot.
- The space of reentrant procedures can be saved! The non-modified nature of shared code must be enforced
- Address referencing inside shared pages could be an issue.

Multilevel Paging

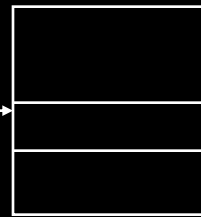
- Motivation
 - The logical address space of a process in many modern computer system is very large, e.g., 2^{32} to 2^{64} Bytes.
32-bit address $\rightarrow 2^{20}$ page entries \rightarrow 4MB
4KB per page 4B per entries page table
- \rightarrow Even the page table must be divided into pieces to fit in the memory!

Multilevel Paging – Two-Level Paging

Logical Address



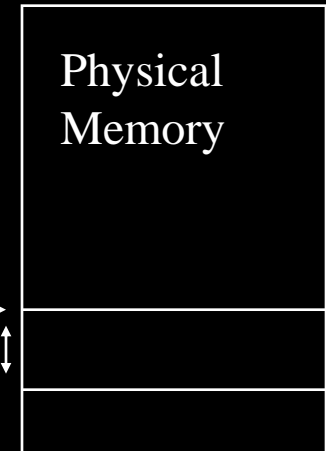
P1



P2



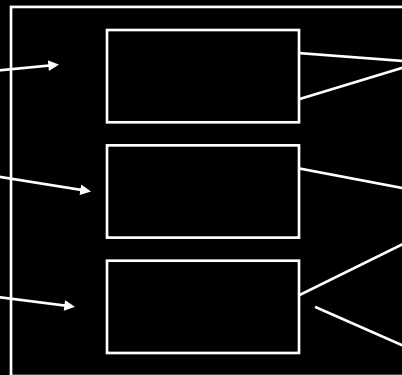
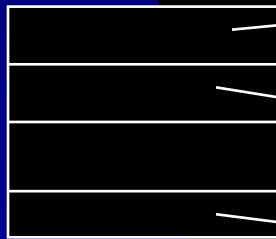
d



Outer-Page Table

A page of page table

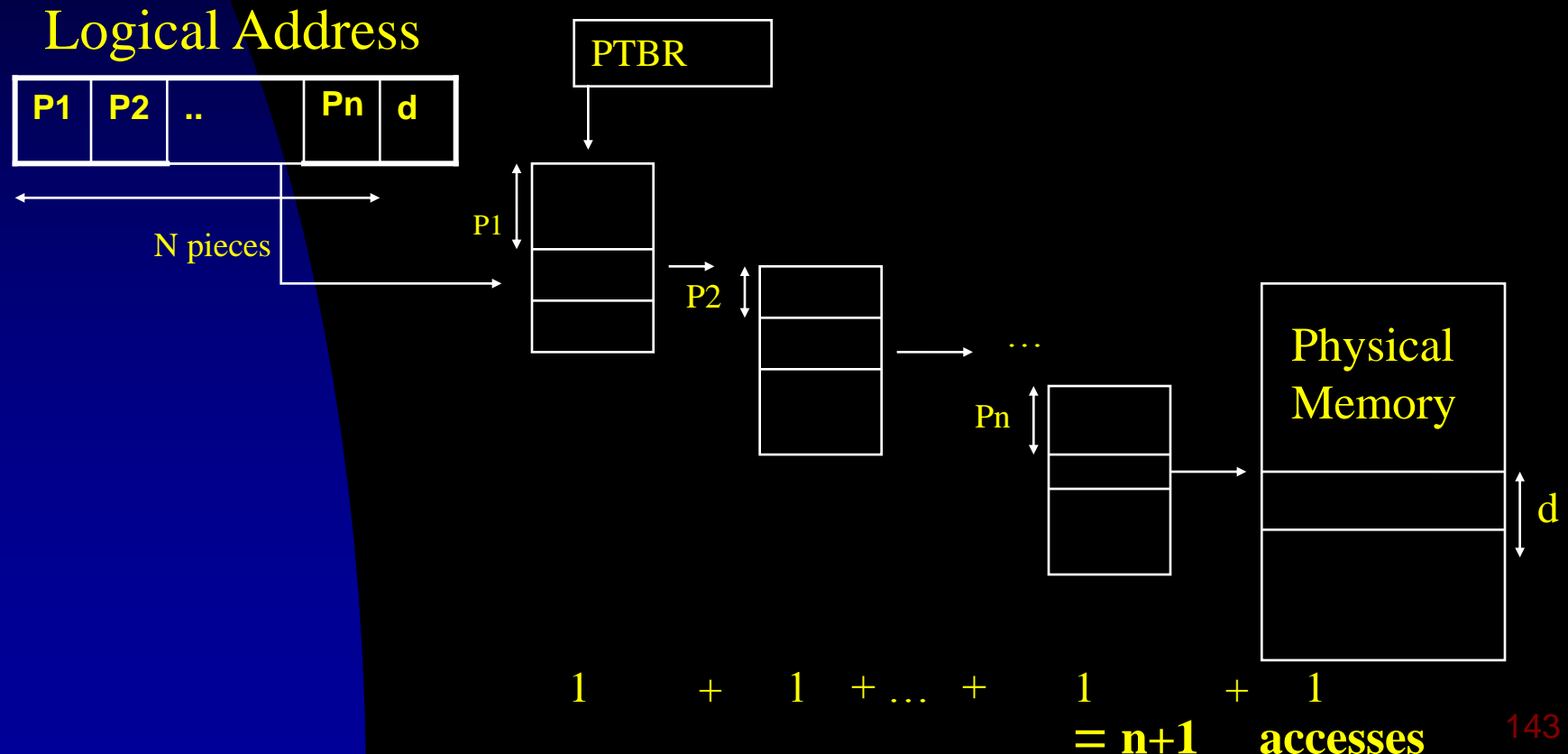
PTBR



Forward-Mapped Page Table

Multilevel Paging – N-Level Paging

- Motivation: Two-level paging is not appropriate for a huge logical address space!



Multilevel Paging – N-Level Paging

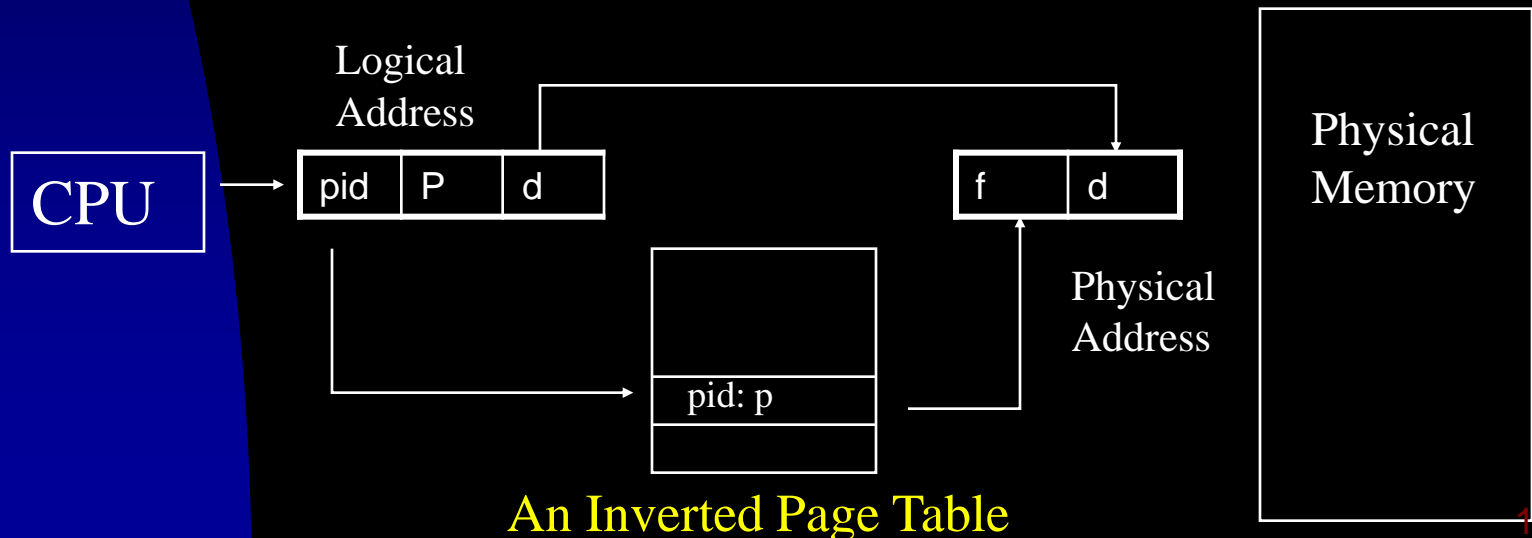
- Example
 - 98% hit ratio, 4-level paging, 20ns TLB access time, 100ns memory access time.
 - Effective access time = $0.98 \times 120\text{ns} + 0.02 \times 520\text{ns} = 128\text{ns}$
- SUN SPARC (32-bit addressing) → 3-level paging
- Motorola 68030 (32-bit addressing) → 4-level paging
- VAX (32-bit addressing) → 2-level paging

Hashed Page Tables

- Objective:
 - To handle large address spaces
- Virtual address \rightarrow hash function \rightarrow a linked list of elements
 - (virtual page #, frame #, a pointer)
- Clustered Page Tables
 - Each entry contains the mappings for several physical-page frames, e.g., 16.

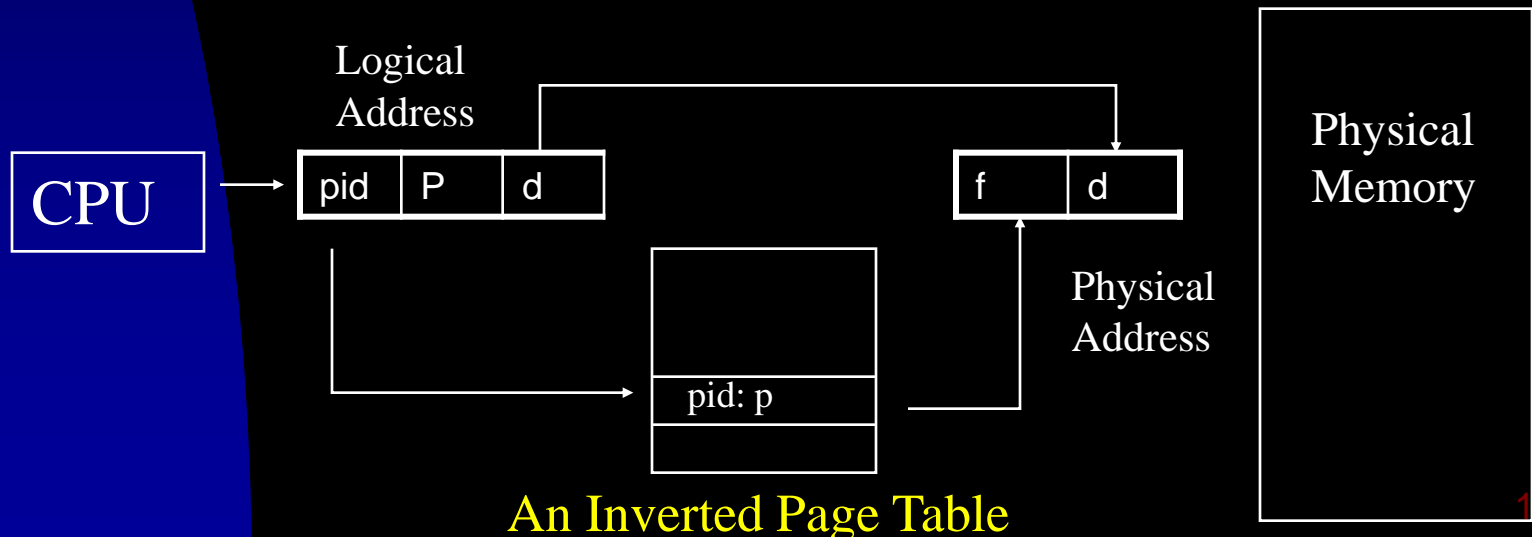
Inverted Page Table

- Motivation
 - A page table tends to be big and does not correspond to the # of pages residing in the physical memory.
- Each entry corresponds to a physical frame.
 - Virtual Address: <Process ID, Page Number, Offset>



Inverted Page Table

- Each entry contains the virtual address of the frame.
 - Entries are sorted by physical addresses.
 - One table per system.
- When no match is found, the page table of the corresponding process must be referenced.
- Example Systems: HP Spectrum, IBM RT, PowerPC, SUN 64-bit UltraSPARC

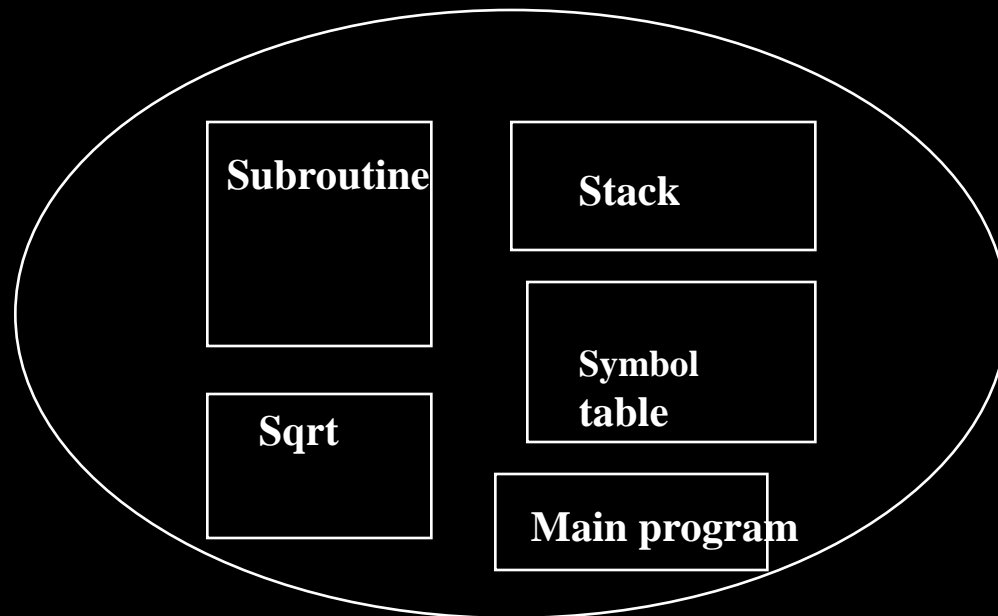


Inverted Page Table

- Advantage
 - Decrease the amount of memory needed to store each page table
- Disadvantage
 - The inverted page table is sorted by physical addresses, whereas a page reference is in a logical address.
 - The use of Hash Table to eliminate lengthy table lookup time: 1HASH + 1IPT
 - The use of an associate memory to hold recently located entries.
 - Difficult to implement with shared memory

Segmentation

- Segmentation is a memory management scheme that supports the user view of memory:
 - A logical address space is a collection of segments with variable lengths.

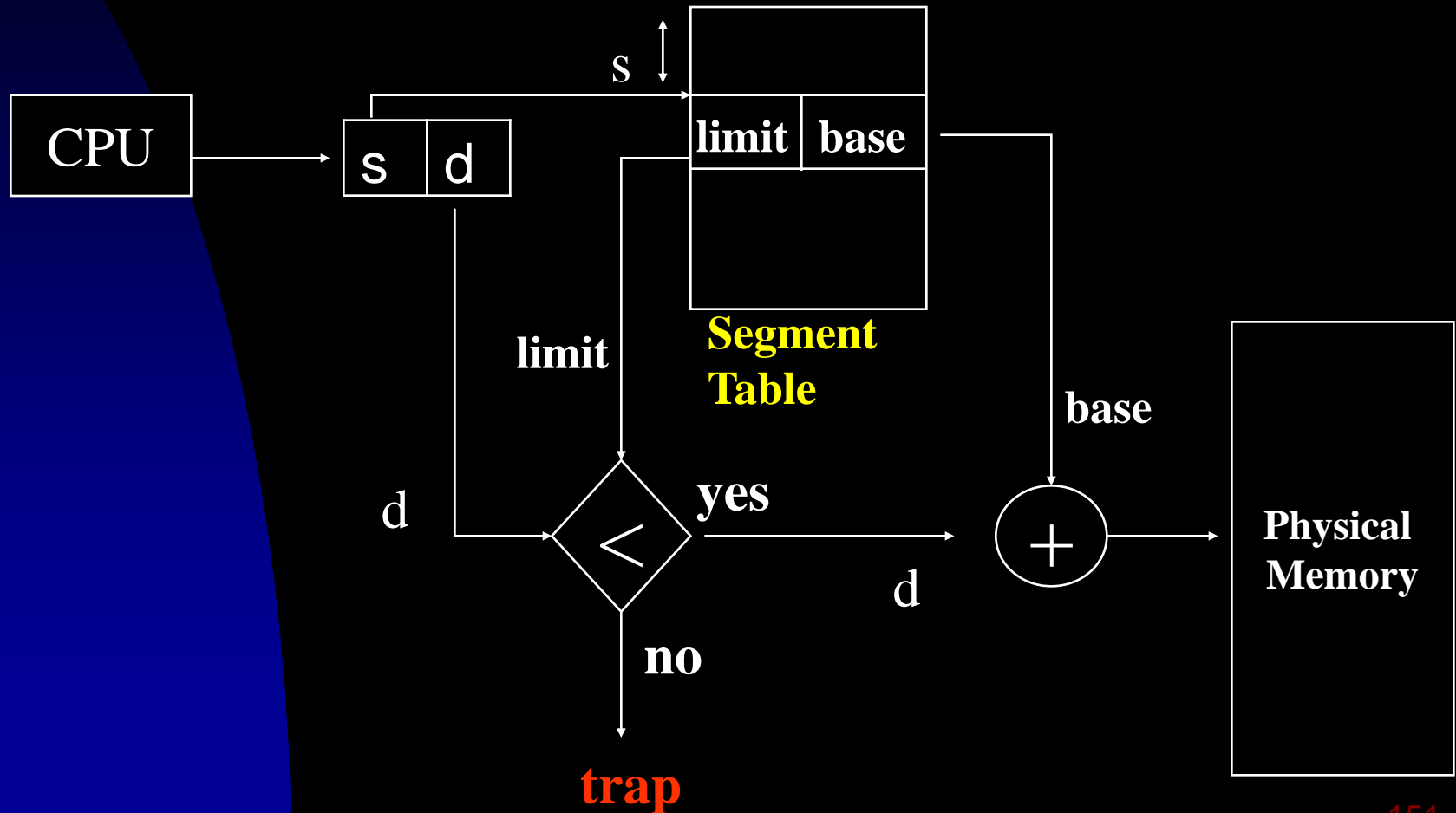


Segmentation

- Why Segmentation?
 - Paging separates the user's view of memory from the actual physical memory but does not reflect the logical units of a process!
 - Pages & frames are fixed-sized, but segments have variable sizes.
- For simplicity of representation,
<segment name, offset> → <segment-number, offset>

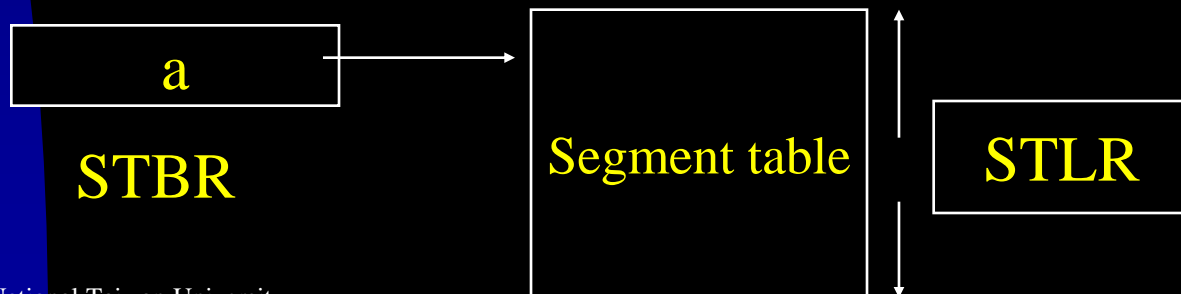
Segmentation – Hardware Support

■ Address Mapping



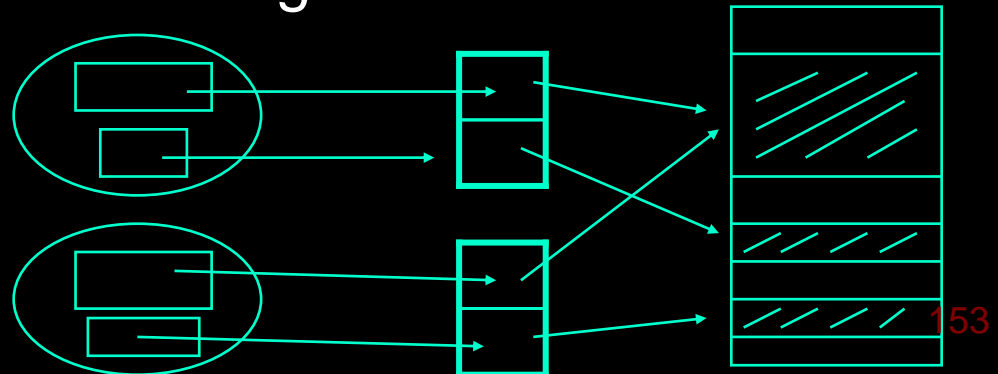
Segmentation – Hardware Support

- Implementation in Registers – limited size!
- Implementation in Memory
 - Segment-table base register (STBR)
 - Segment-table length register (STLR)
 - Advantages & Disadvantages – Paging
 - Use an associate memory (TLB) to improve the effective memory access time !
 - TLB must be flushed whenever a new segment table is used !



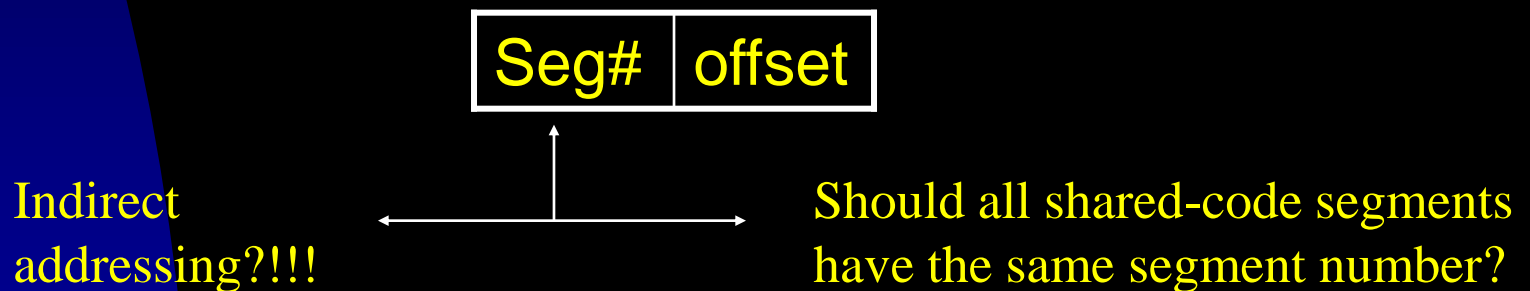
Segmentation – Protection & Sharing

- Advantage:
 - Segments are a semantically defined portion of the program and likely to have all entries being “homogeneous”.
 - Example: Array, code, stack, data, etc.
→ Logical units for protection !
 - Sharing of code & data improves memory usage.
 - Sharing occurs at the segment level.



Segmentation – Protection & Sharing

- Potential Problems
 - External Fragmentation
 - Segments must occupy contiguous memory.
 - Address referencing inside shared segments can be a big issue:



- How to find the right segment number if the number of users sharing the segments increase! → Avoid reference to segment #

Segmentation – Fragmentation

- Motivation:
 - Segments are of variable lengths!
 - Memory allocation is a dynamic storage-allocation problem.
 - best-fit? first-fit? worst-ft?
- External fragmentation will occur!!
 - Factors, e.g., average segment sizes

Size
↓
A byte

External
Fragmentation
↓
Overheads increases substantially!
(base+limit “registers”)

Segmentation – Fragmentation

- Remark:
 - Its external fragmentation problem is better than that of the dynamic partition method because segments are likely to be smaller than the entire process.
- Internal Fragmentation??

Segmentation with Paging

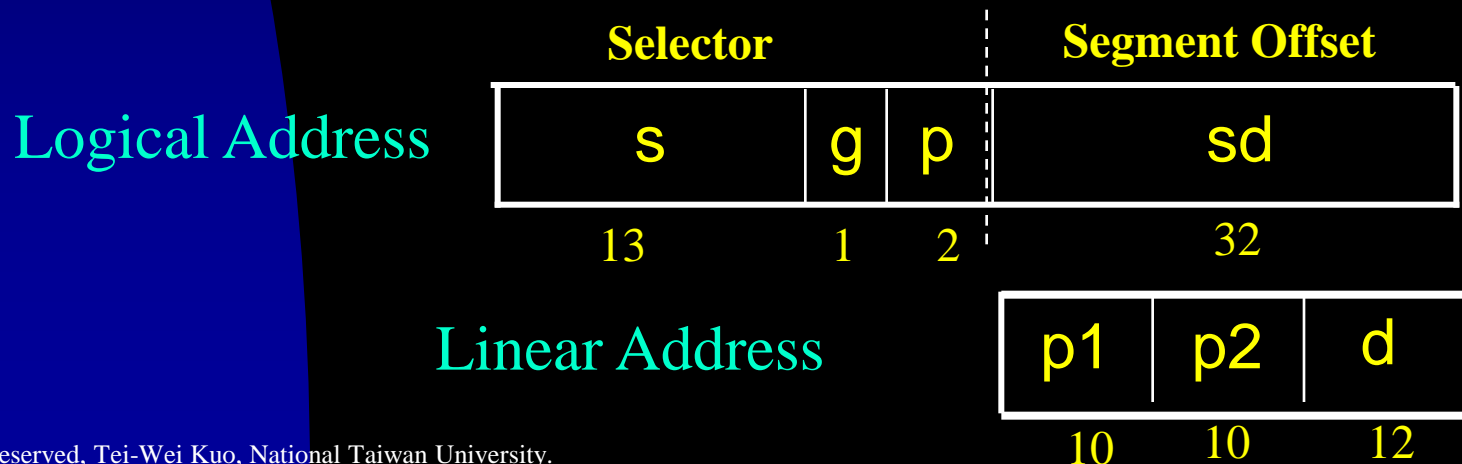
- Motivation :
 - Segmentation has external fragmentation.
 - Paging has internal fragmentation.
 - Segments are semantically defined portions of a program.
 - “Page” Segments !

Oracle SPARC Solaris

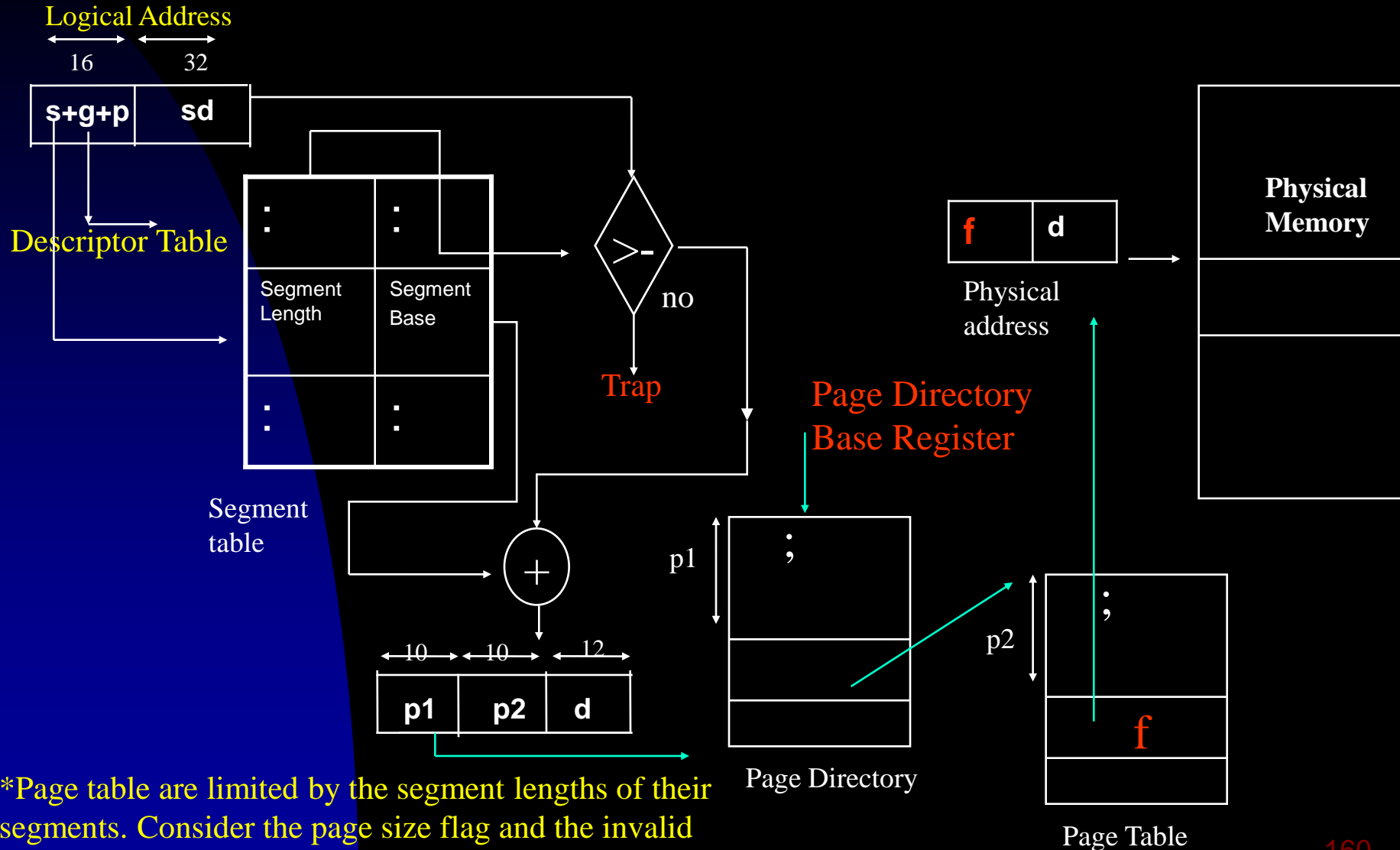
- Solaris over a 64-bit SPARC CPU computer
 - One hash table for the kernel and one for all user processes
 - Each entry has a base address and the number of pages for the entry
 - The procedure of a virtual memory translation: TLB → Translation Storage Buffer (TSB) → Interrupt happens to manipulate TSB and TLB

IA-32 Segmentation

- 8K Private Segments + 8K Public Segments
 - Page Size = 4KB or 4MB (*page size flag* in the page directory), Max Segment Size = 4GB
 - Tables:
 - Local Descriptor Table (LDT)
 - Global Descriptor Table (GDT)
 - 6 microprogram segment registers for caching



IA-32 Segmentation

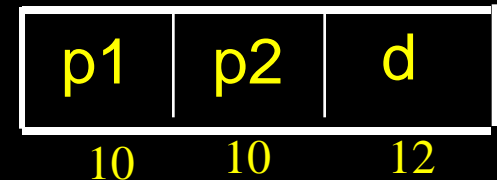


*Page table are limited by the segment lengths of their segments. Consider the page size flag and the invalid bit of each page directory entry.

IA-32 Paging

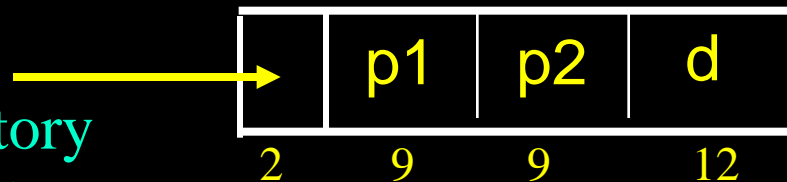
- A Two-Level Paging Scheme
 - Page Directory and Page_Size flag

Linear Address



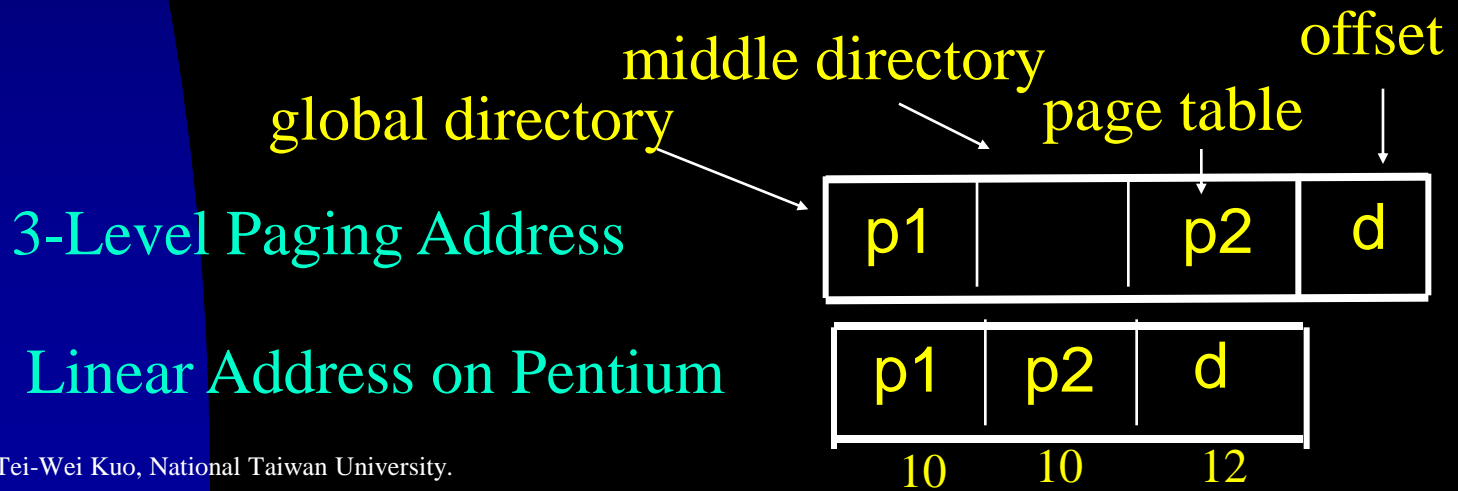
- Invalid Bit in the Page Dir
 - Swapping support to the disk
- Page Address Extension – 3 Levels

Page Directory
Pointer Table



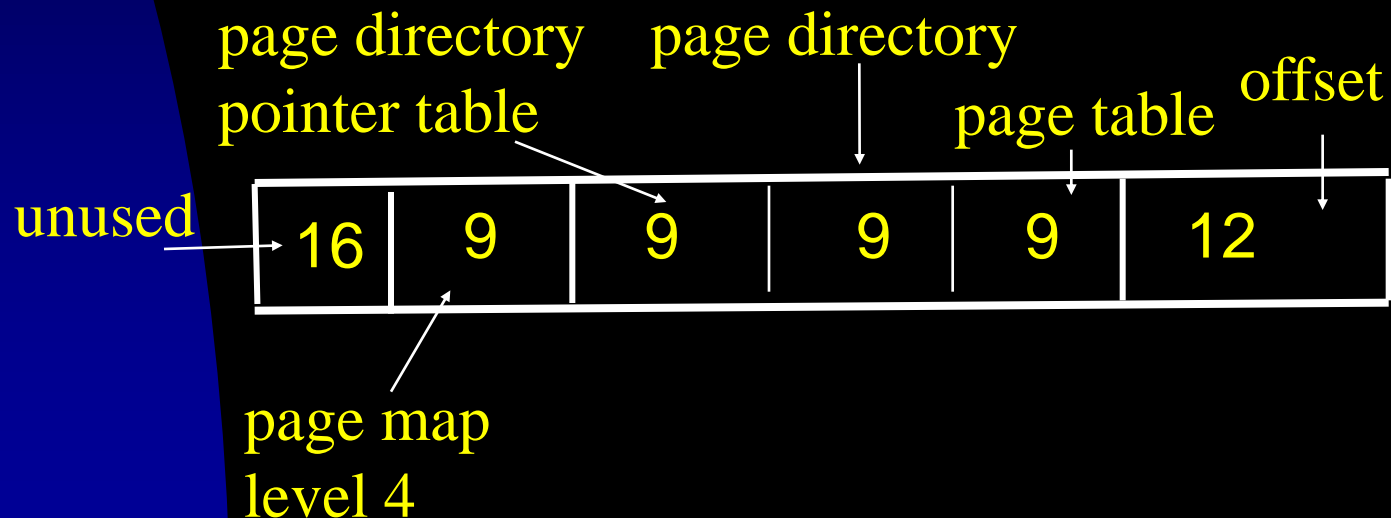
IA-32 and Linux

- Limitation & Goals
 - Supports over a variety of machines
 - Use segmentation minimally – GDT
 - One individual segment for the kernel code, kernel data, the user code, the user data, the task state segment, the default LDT
 - Protection: user and kernel modes



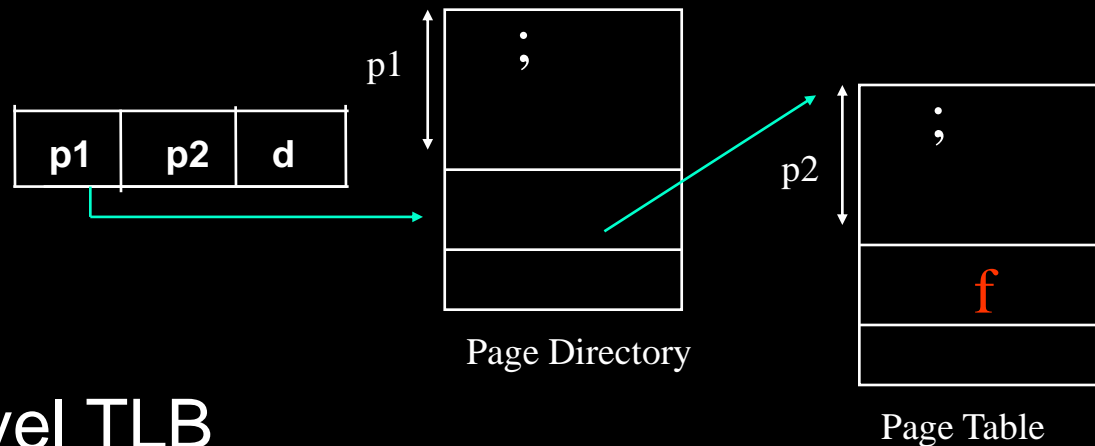
x86-64

- x86-64 is a 64-bit architecture based on IA-32 and proposed by AMD (and used by Intel).
- Page Sizes: 4KB, 2MB, or 1GB



32-Bit ARM

- Paging:
 - 2-Level Paging: 4KB or 16KB Pages
 - 1-Level Paging: 1MB or 16MB Sections



- 2-Level TLB
 - Micro TLBs – Data and Instructions
 - Main TLB

Paging and Segmentation

- To overcome disadvantages of paging or segmentation alone:
 - Paged segments – divide segments further into pages.
 - Segment need not be in contiguous memory.
 - Segmented paging – segment the page table.
 - Variable size page tables.
- Address translation overheads increase!
- An entire process still needs to be in memory at once!

→ Virtual Memory!!

Paging and Segmentation

- Considerations in Memory Management
 - Hardware Support, e.g., STBR, TLB, etc.
 - Performance
 - Fragmentation
 - Multiprogramming Levels
 - Relocation Constraints?
 - Swapping: +
 - Sharing?!
 - Protection?!