Operating Systems

0107451

Chapter 3 Memory Management

Dr. Naeem Odat

Tafial Technical University
Department of Computer and Communications Engineering

Memory Management

Introduction

- Memory is a general term that includes cache memory, main memory (RAM), secondary storage.
- The subject of this part is the main memory.
- Memory management is the responsibility of a **memory manager** in an operating system.
- Memory manger job is to efficiently manage memory:
  - keep track of which parts of memory are in use,
  - allocate memory to processes when they need it,
  - and deallocate it when they are done.

Memory Management

Introduction

- Ideal World (for the programmer):
  - I'm the only process in the world
  - I have a huge amount of memory at my disposal
- Real World
  - Many processes in the system
  - Not enough memory for them all

Memory Management

Goal of Memory Management

Have an abstraction for memory that presents the ideal world view to the programmer, yet implement it on a real system.

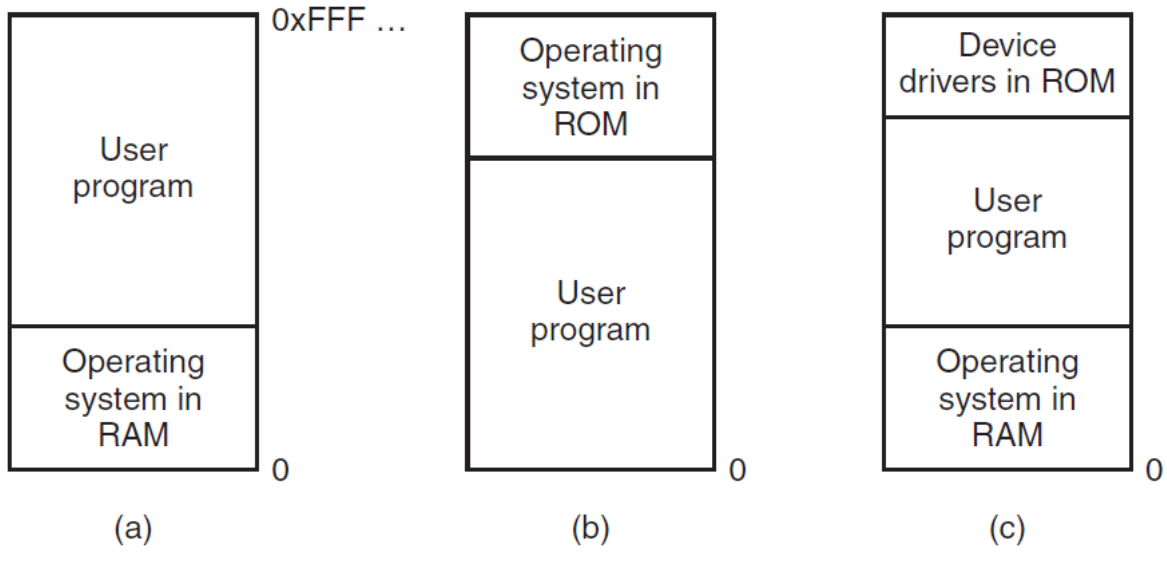
3.1 No Memory abstractions

A process loaded into memory sees the physical address. Two issues should be solved:

1. Protection of OS and other processes from the loaded process.
2. Relocation of address.

Memory organization

The main memory can be organized with OS in three or more ways.



One way to get some parallelism in a system with no memory abstraction is to program with multiple threads.

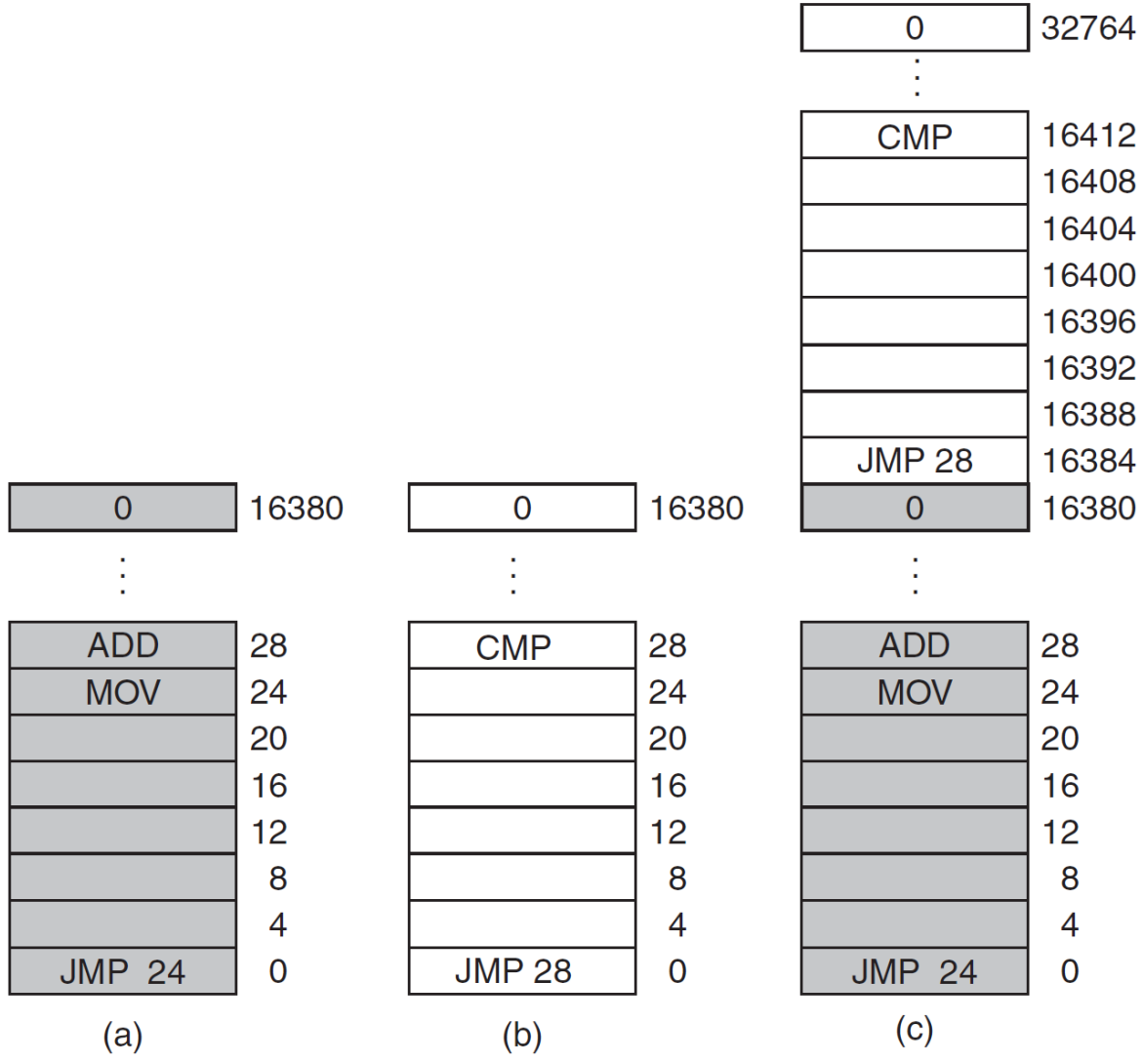
Running Multiple Programs

No Memory Abstraction

1. OS saves the entire contents of memory to a disk file, then bring in and run the next program (swapping).
2. With the addition of some special hardware, it is possible to run multiple programs concurrently without swapping (IBM solution). IBM solution (Lock-and-Key):
   - 2.1 Memory is divided into 2kbyte blocks each with 4 bit protection key, held in a special CPU registers.
   - 2.2 PSW has a protection key. If a program attempts to access memory with different protection key, it traps into OS.
   - 2.3 Drawback: Crash because of protection.
   - 2.4 IBM solved this by static relocation by modifying the addresses on the fly while loading. (Not a generic solution)

Running Multiple Programs

No Memory Abstraction



The Relocation Problem

- Your Program is compiled and linked (generates absolute addresses)
- When being loaded into memory, can't predict which address it will be loaded in
- How to ensure correct memory addresses are used, and to manage protection?
- One solution, at loading time:
  - Relocate processes in a free memory hole. (Doesn't ensure protection.)
  - Update all the addresses to reflect correct physical addresses. (Distinguishing between constants and addresses is a problem.) (**Static Relocation.**)

Example on indistinguishable addresses

```
MOV AX, 1000H        B8 0010
JMP AX               FF E0
MOV AX, 2000H        B8 0020
JMP AX               FF E0
```

3.2 Address Space Concept

Address Space

- An address space is the set of addresses that a process can use to address memory.
- Each process has its own address space, independent of those belonging to other processes (despite the same address values).

3.2 Address Space Concept

Base and Limit Registers

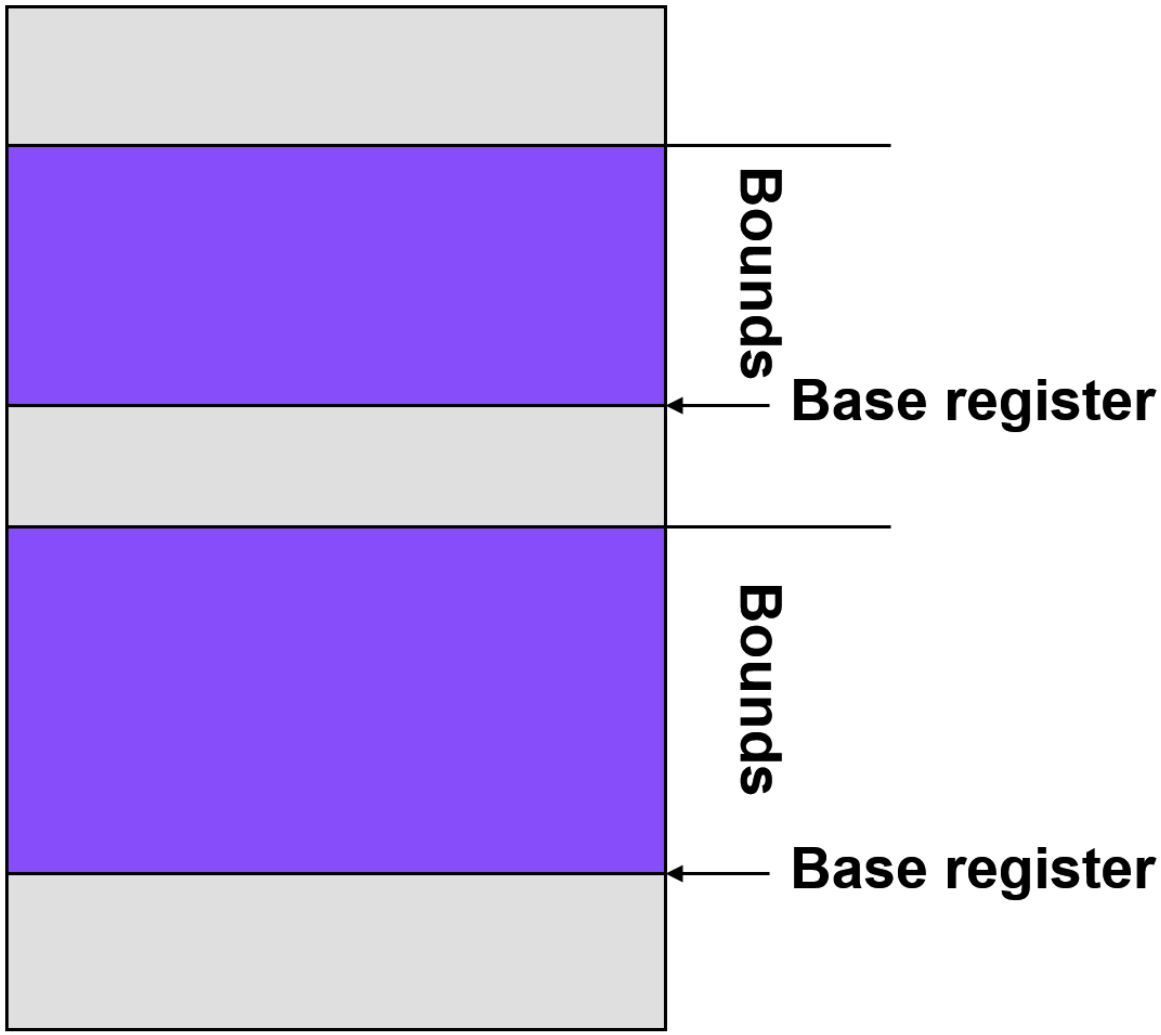
- To give each program its own address space, **base and limit registers** are used.
- It is a version of **dynamic relocation**.
- Each CPU is equipped with two special hardware registers, **base and limit** registers.
- When a process is run, the base register is loaded with the physical address where its program begins in memory and the limit register is loaded with the length of the program.

Base and Limit (Bound) Registers

- Every time a process references memory, either to fetch an instruction or read or write a data word:
  - The CPU hardware automatically adds the base value to the address generated by the process before sending the address out on the memory bus.
  - Simultaneously, it checks whether the address offered is equal to or greater than the value in the limit register, in which case a fault is generated and the access is aborted.
- A **disadvantage** is the need to perform an addition (slow because of carry-propagation) and a comparison (fast) on every memory reference.

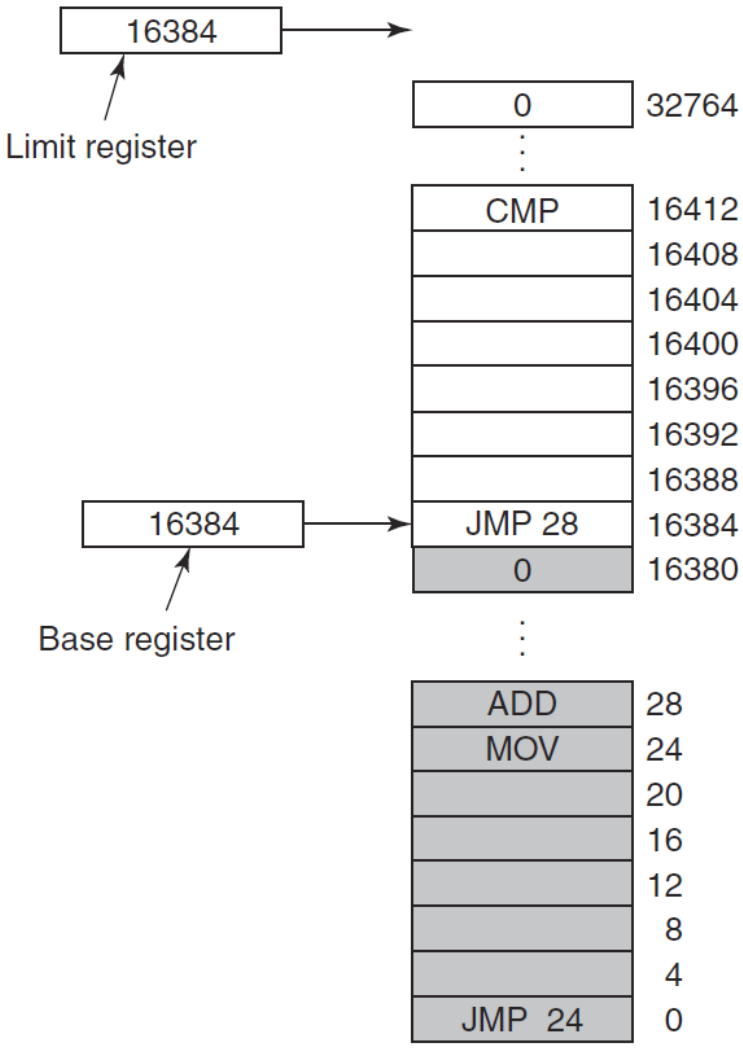
3.2 Address Space Concept

Base and Bound Registers



3.2 Address Space Concept

Base and Limit Registers

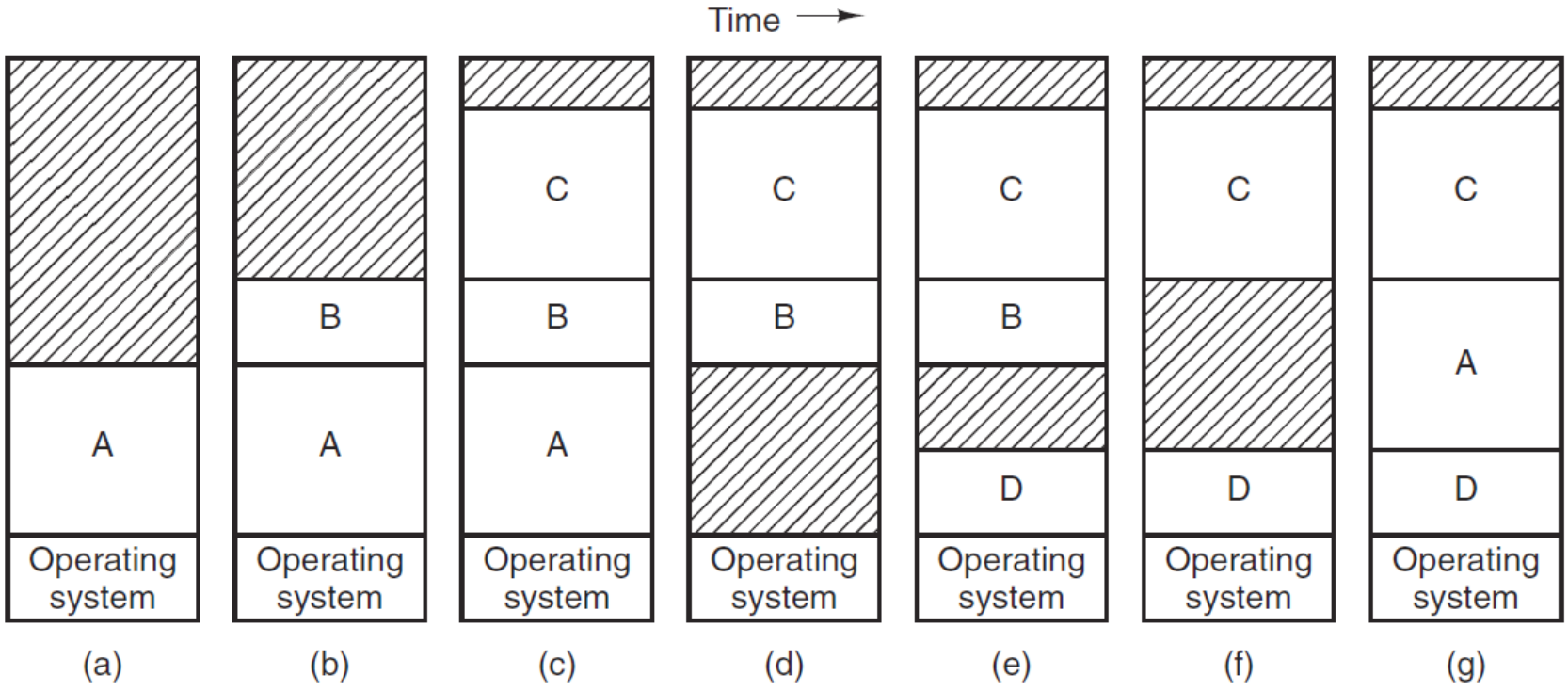


Memory Overloading

- Since physical memory is limited, eventually, it becomes full. Then an overload happens.
- Two general approaches to dealing with memory overload have been developed over the years.
  1. The simplest strategy, called **swapping**, consists of bringing in each process in its entirety, running it for a while, then putting it back on the disk.
  2. The other strategy, called **virtual memory**, allows programs to run even when they are only partially in main memory

3.2 Address Space Concept

Swapping

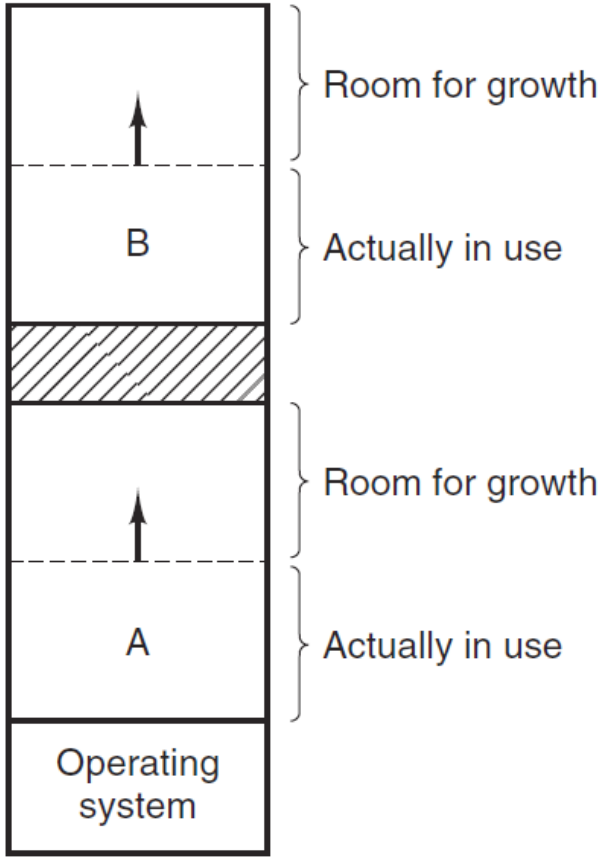


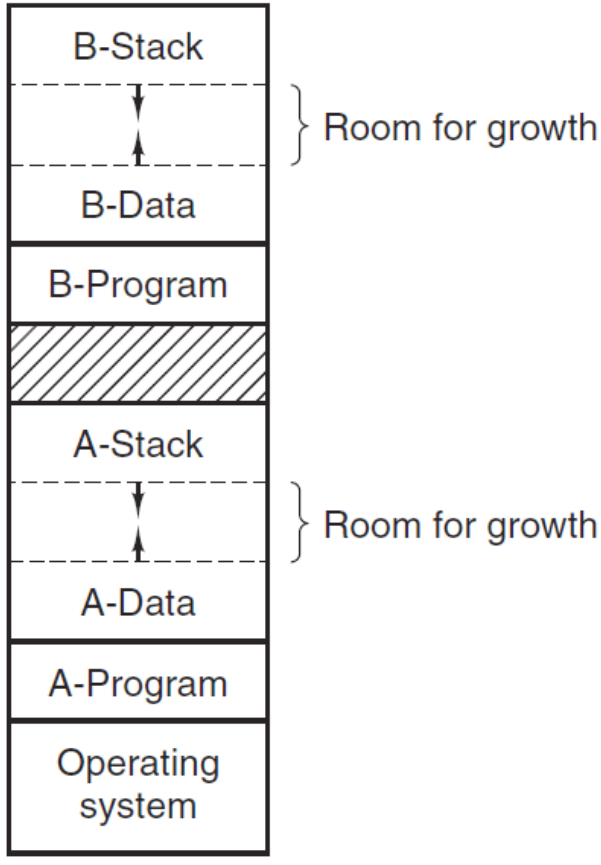
Swapping - Overhead

- The size of memory allocated for the process can be the same of the program size if the program does not grow dynamically.
- If the program needs memory as it runs, enough amount of memory, larger than its size is allocated
- To reduce the overhead associated with moving or swapping processes that no longer fit in their allocated memory.

3.2 Address Space Concept

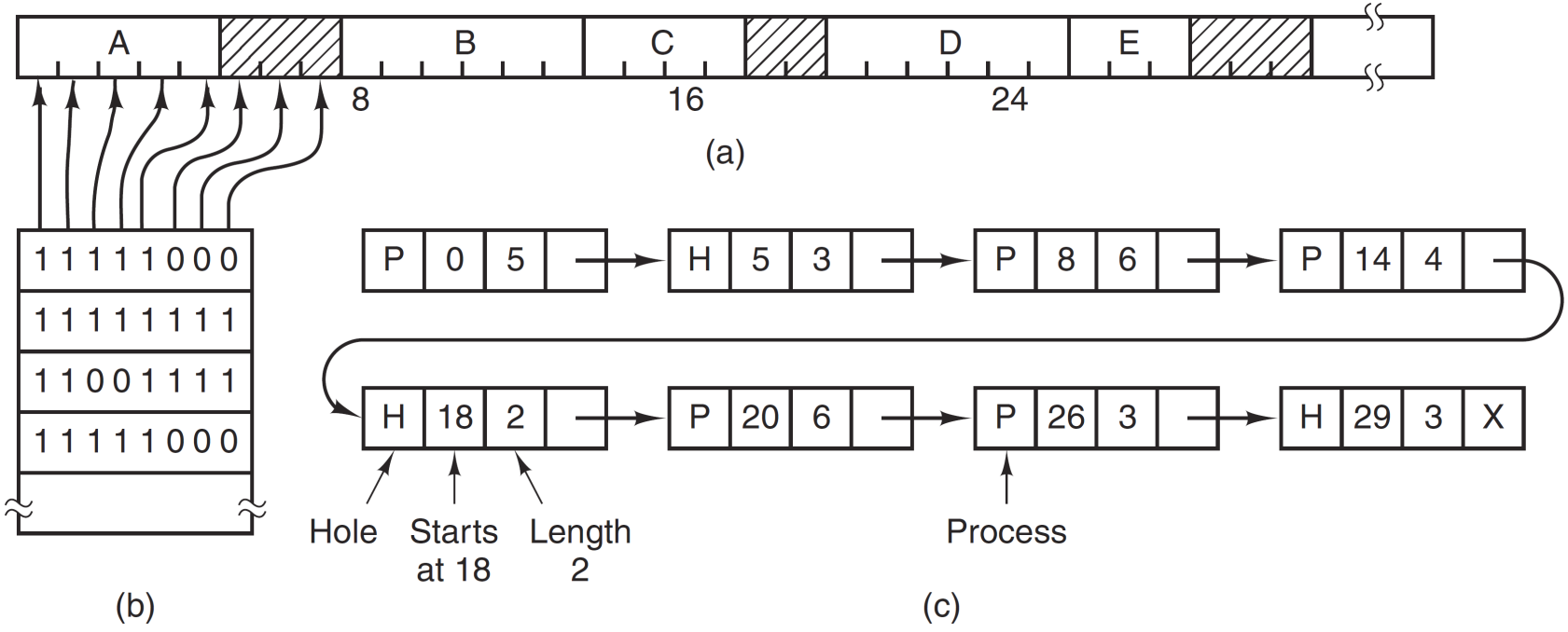
Swapping - Overhead





Managing Free Memory

In general terms, there are two ways to keep track of memory usage: **bitmaps** and **free lists**.



Memory Management with Bitmaps

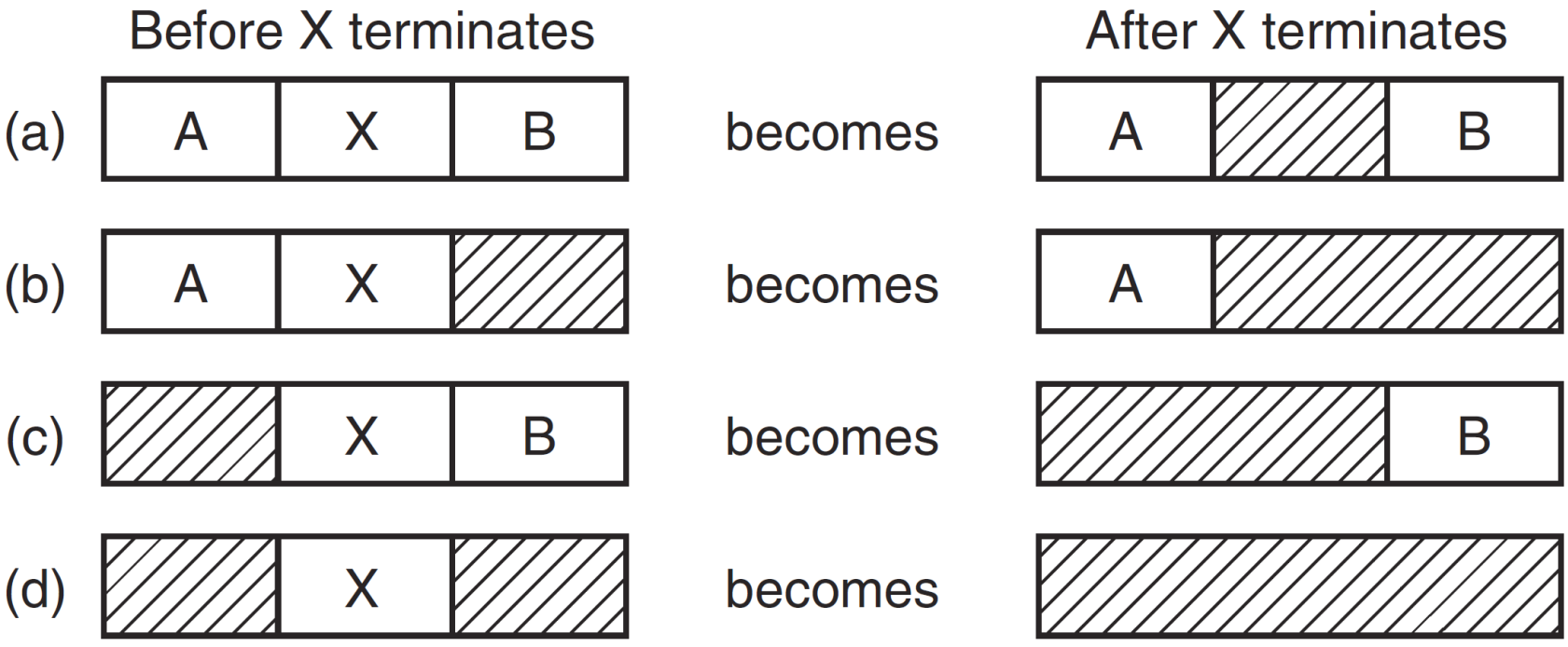
- With a bitmap, memory is divided into allocation units as small as a few words and as large as several kilobytes.
- Corresponding to each allocation unit is a bit in the bitmap, which is 0 if the unit is free and 1 if it is occupied (or vice versa).
- The main problem is searching a bitmap for a run of a given length is a slow operation.

Memory Management with Linked Lists

- Another way of keeping track of memory is to maintain a linked list of allocated and free memory segments.
- A segment either contains a process or is an empty hole between two processes.
- Each entry in the list specifies a hole (H) or process (P), the address at which it starts, the length, and a pointer to the next item.
- Double-linked list is better in this management.

3.2 Address Space Concept

Memory Management with Linked Lists



Memory Management with Linked Lists

When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a created process (or an existing process being swapped in from disk).

1. First Fit
2. Next Fit
3. Best Fit
4. Worst Fit
5. Quick Fit. Maintains separate lists for some of the more common sizes requested.

Virtual Memory

- There is a need to run programs that are **too large to fit in memory**
- There is certainly a need to have systems that can support multiple programs running simultaneously, each of which fits in memory but all of which collectively exceed memory.
- Swapping is not an attractive option, since access time of a typical SATA disk is long.
- A program that is larger than the physical memory has to use **Overlays**. Which is not easy to be done.

Virtual Memory

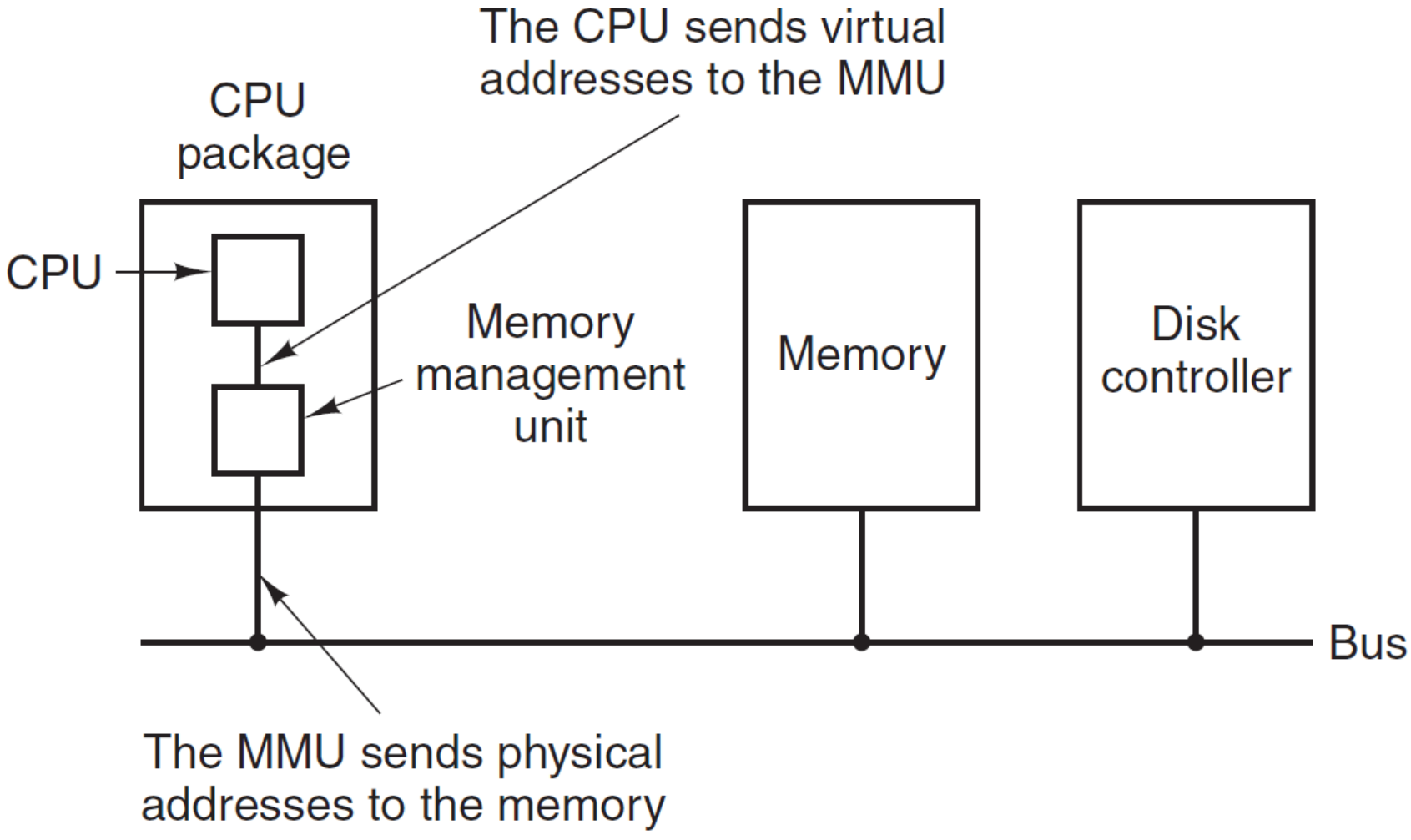
- The basic idea behind virtual memory is that each program has its own address space, which is broken up into chunks called **pages**.
- Each page is a contiguous range of addresses.
- These pages are mapped into **physical frames**.
- When a process references a page, a mapping is done on the fly if it is in memory, if it is not, the OS brings it to memory and restart the instruction that referenced it.

Paging

- A program-generated addresses are called **virtual addresses** and form the **virtual address space**.
- Virtual address goes to an **MMU** (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses.
- The virtual address space consists of fixed-size units called **pages**.
- The corresponding units in the physical memory are called **page frames**.
- The pages and page frames are generally the same size.

3.3 Virtual Memory

Paging - MMU

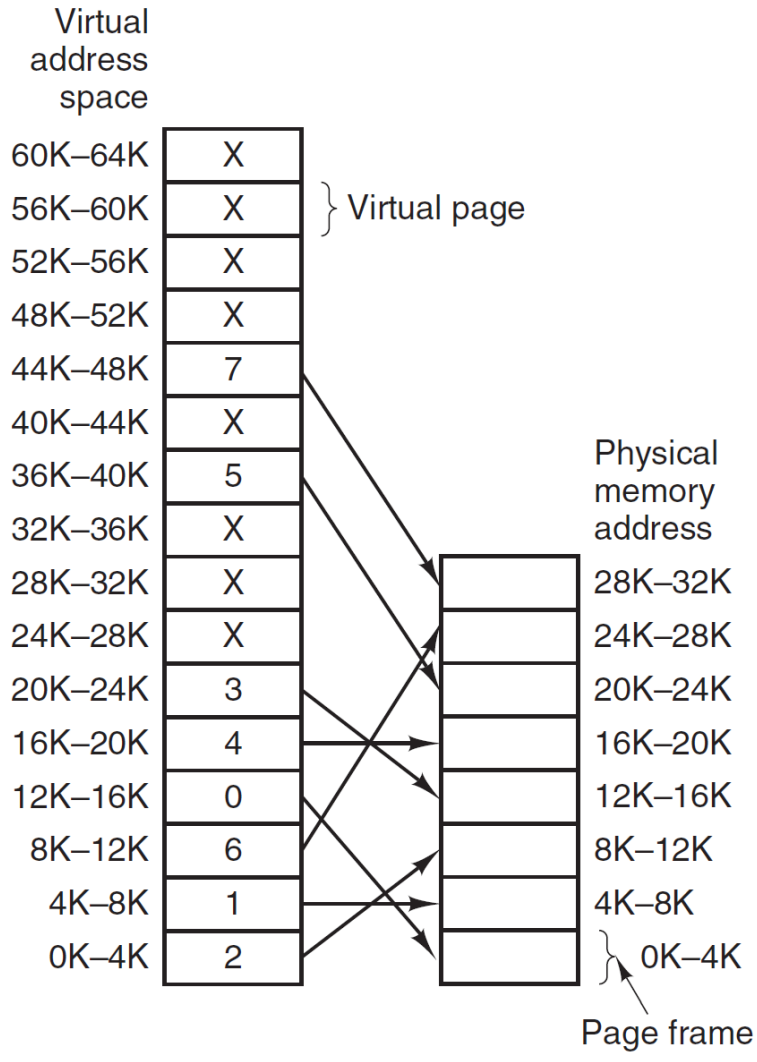


MMU - a closer look

- Lookup the virtual address.
- If present in physical memory, then fetch it.
- If not, then call upon the operating system ("page fault").
  - OS loads the required page into memory from secondary storage.
  - Instruction is re-started.

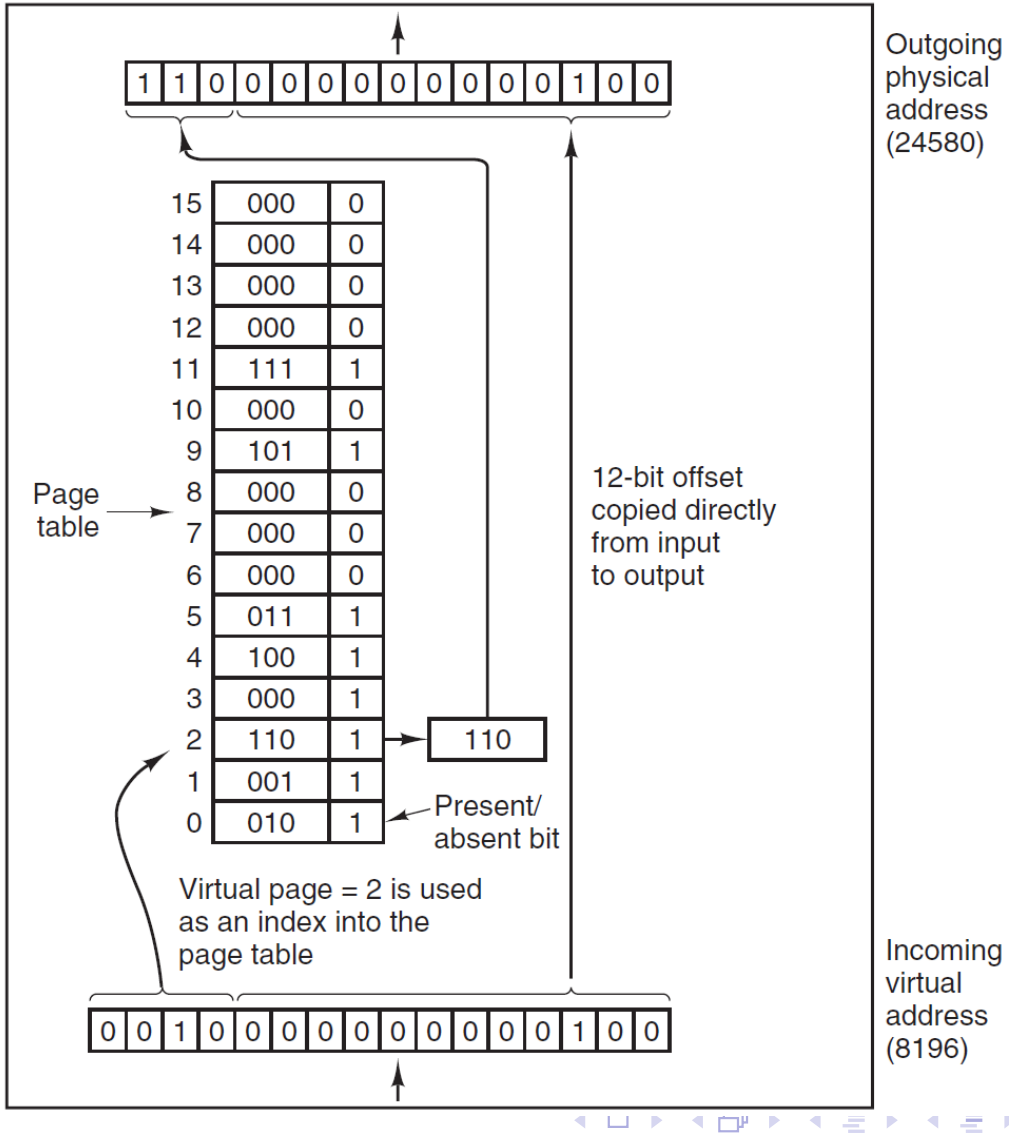
Paging - Example

- 64K Virtual Address Space. Divided into 16 pages, of 4K each.
- 32K Physical Memory. 8 pages of 4K each.



3.3 Virtual Memory

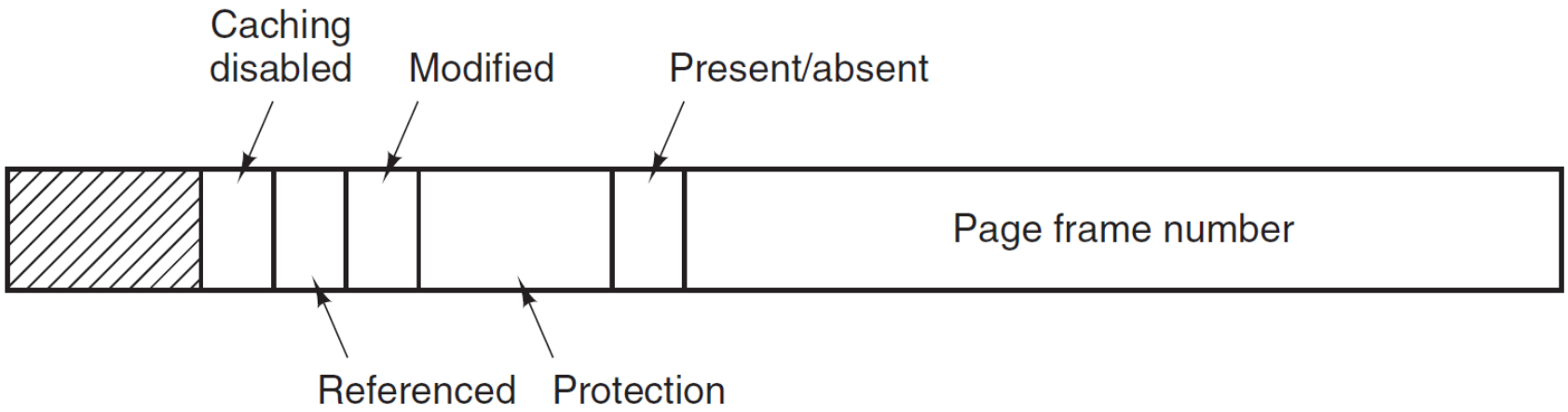
MMU Internals



Page Fault

- What if the required page is not in memory?
- A **page fault** happens.
- Page fault gives control to the OS.
  - Find a free page frame (might need to evict an existing page from memory, page replacement policy).
  - OS fetches the required page from the disk.
  - Modify the page table.
  - Instruction is restarted

Typical Page Table Entry



Address Translation Performance

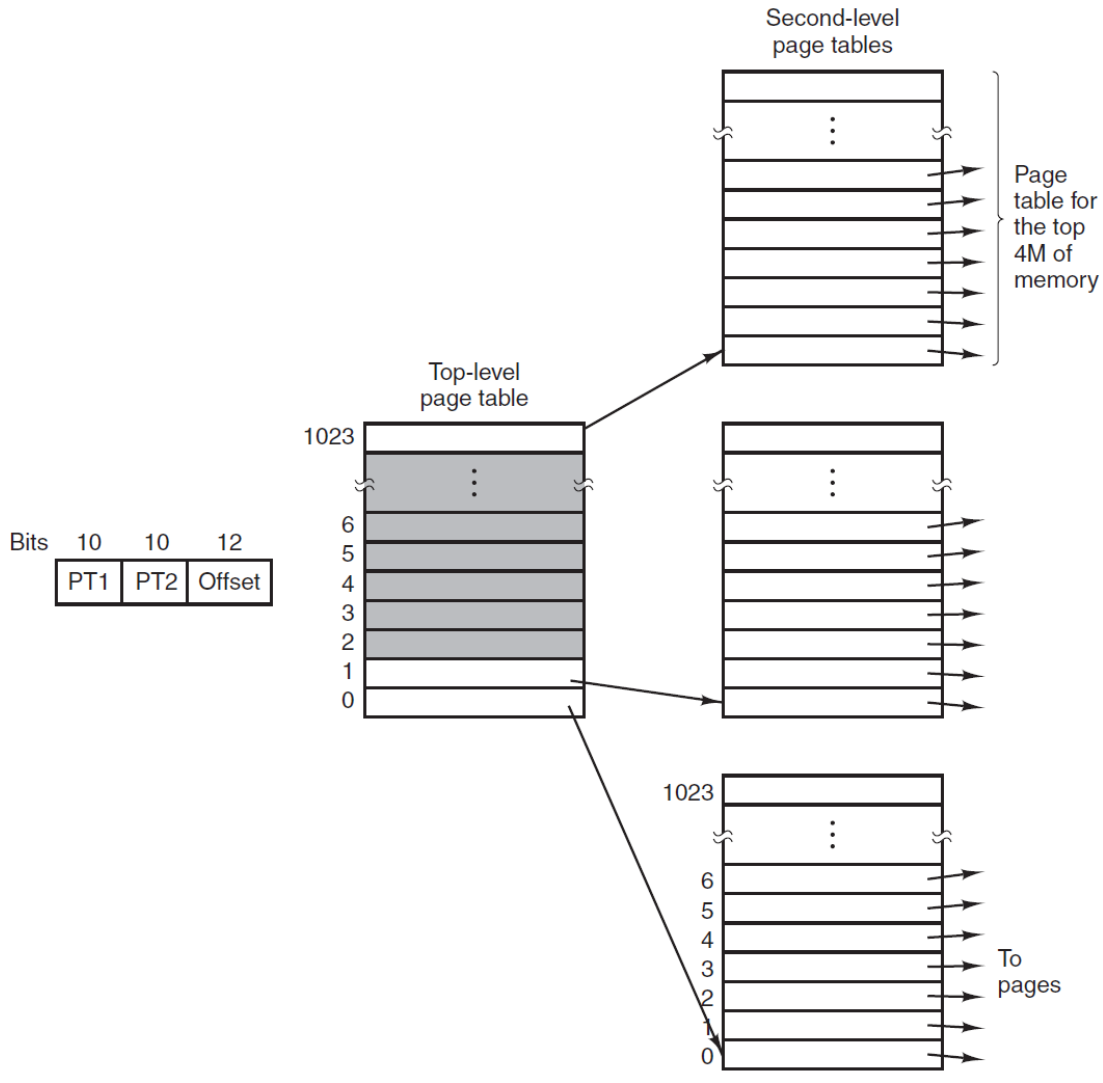
- The address translation is done on every memory reference.
- Maybe twice per instruction:
  - Instruction fetch
  - Fetch Memory operand
- The translation is better to be fast!

Address Translation Schemes

- Page Table – Most popular.
- Translation Lookaside Buffer (TLB) (Performance reasons, cache)
- Large address spaces:
  - Multilevel page tables.
  - Inverted Page Table.

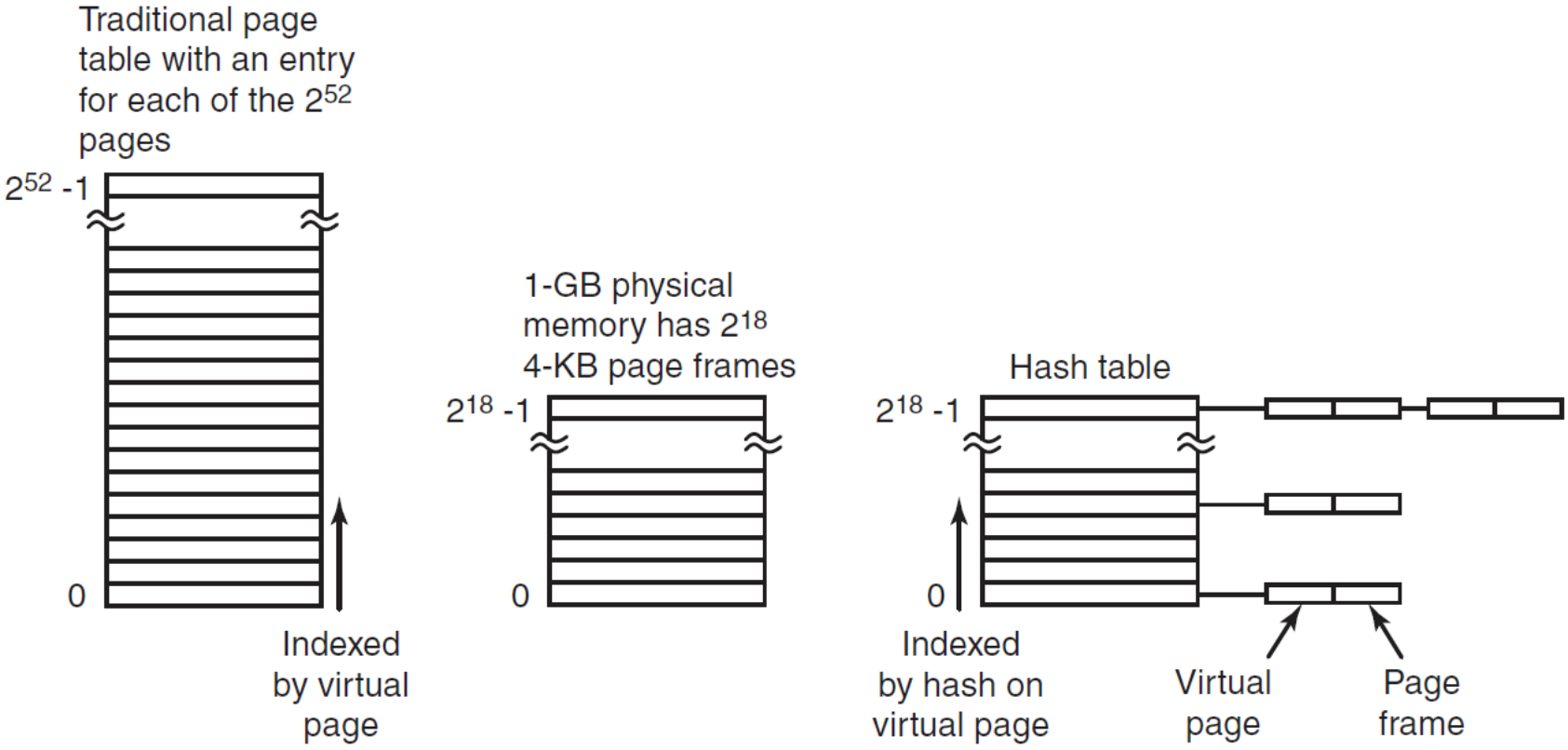
3.3 Virtual Memory

Multi Level Page Table



3.3 Virtual Memory

Inverted Page Table



Question

Where is the page table is located?

- Registers
- Memory

OS Issues

- Fetch policy - when and which pages to fetch into memory?
- Placement policy - where to place pages?
- Replacement policy - when and which page to evict from memory?

All combined in the handling of a page fault.

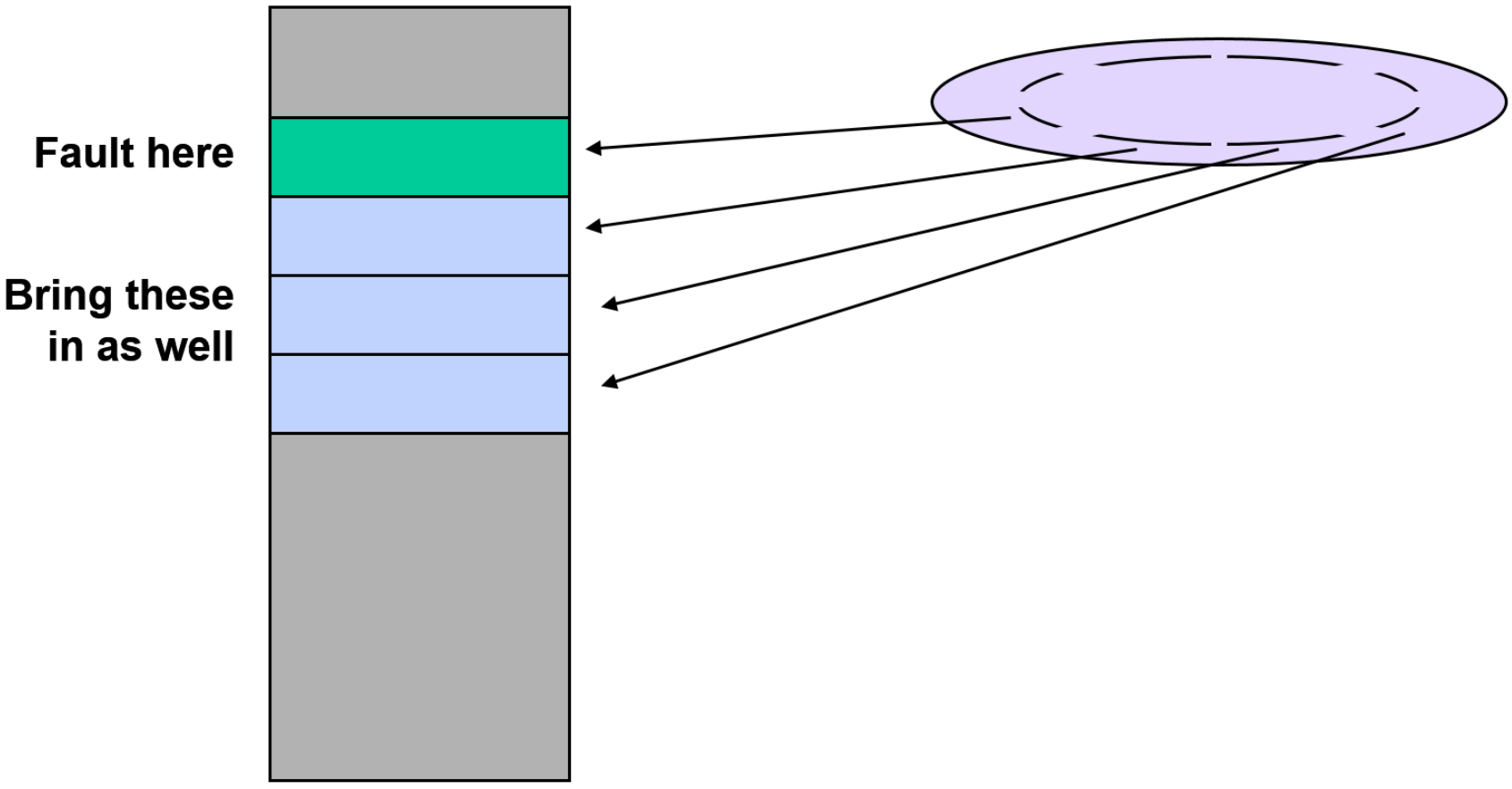
A simple Paging Scheme

- Fetch policy
  - start process off with no pages in primary storage.
  - bring in pages on demand (and only on demand) this is known as demand paging.

  Placement policy
  - it doesn't matter - put the incoming page in the first available page frame.

  Replacement policy
  - replace the page that has been in primary storage the longest (FIFO policy).

Improving the Fetch Policy



Improving the Replacement Policy

- When is replacement done?
  - doing it "on demand" causes excessive delays.
  - should be performed as a separate, concurrent activity
- Which pages are replaced?
  - FIFO policy is not good.
  - want to replace those pages least likely to be referenced soon

Use a pageout daemon.

The Pageout Daemon