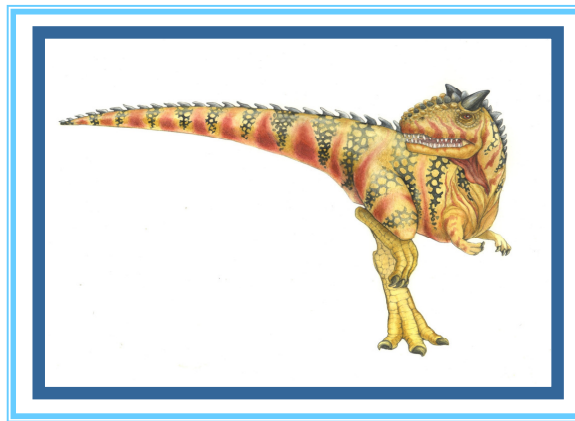# Memory Management

Basic Concepts

Physical memory

   - Hardware memory shared by all programs

   - Organized for performance (memory hierarchy, cache)

   - Vary in size on each machine

Logical memory

   - Abstraction of contiguous memory address space

   - Simple memory model for programming

   - Same size for every process

OS tasks for Memory Management

Provide an abstraction of logical memory for programs

Chapter 8

Provide a virtual memory that is larger than physical memory.

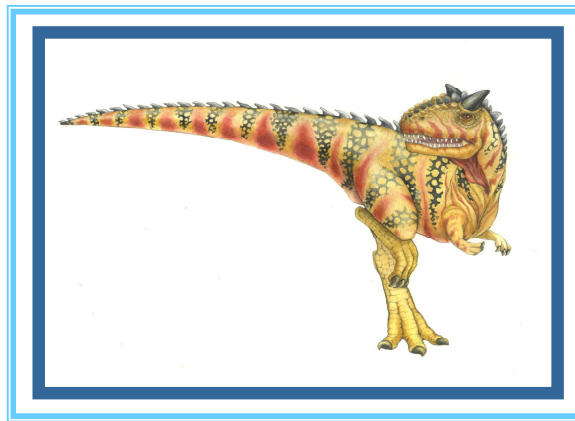Chapter 9

Others

Protection from other processes

Memory sharing between processes

Memory-mapped I/O

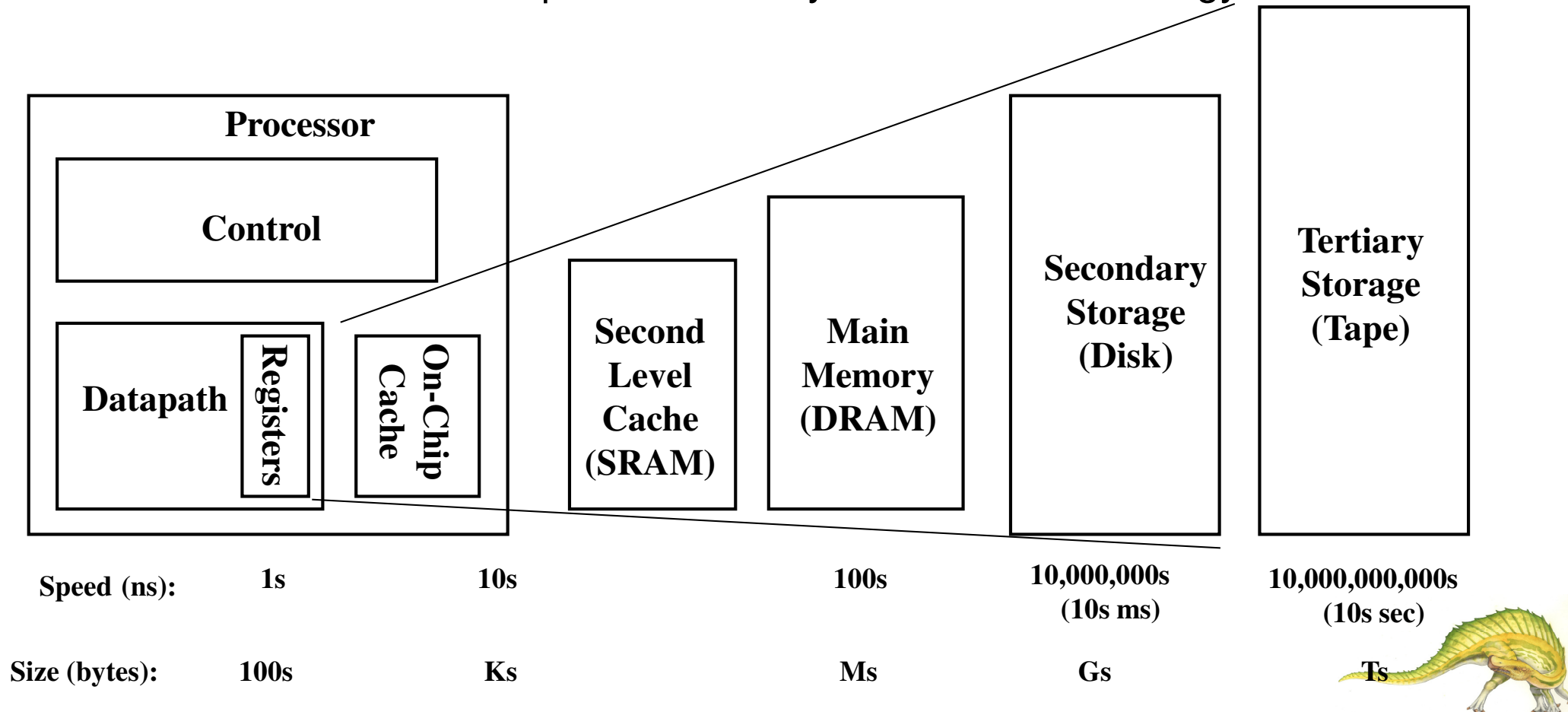# Chapter 8: Memory-Management Strategies

Memory Hierarchy of a Modern Computer System
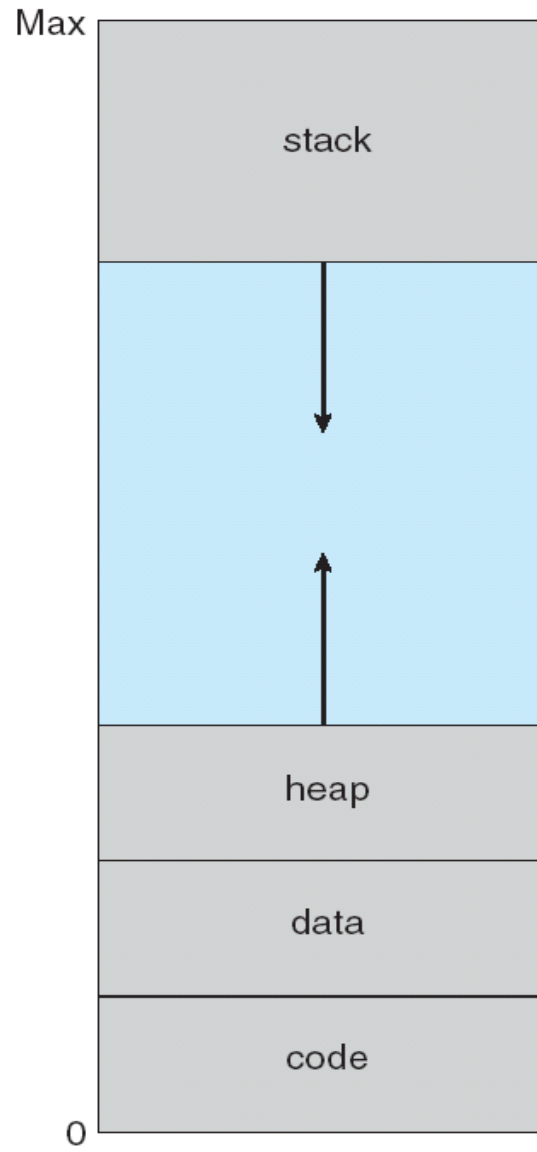
By taking advantage of the principle of **locality**:

Present the user with as much memory as is available in the cheapest technology.

Provide access at the speed offered by the fastest technology.

	Processor	Second Level Cache (SRAM)	Main Memory (DRAM)	Secondary Storage (Disk)	Tertiary Storage (Tape)
Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

Processor contains: Control, Datapath (Registers, On-Chip Cache)
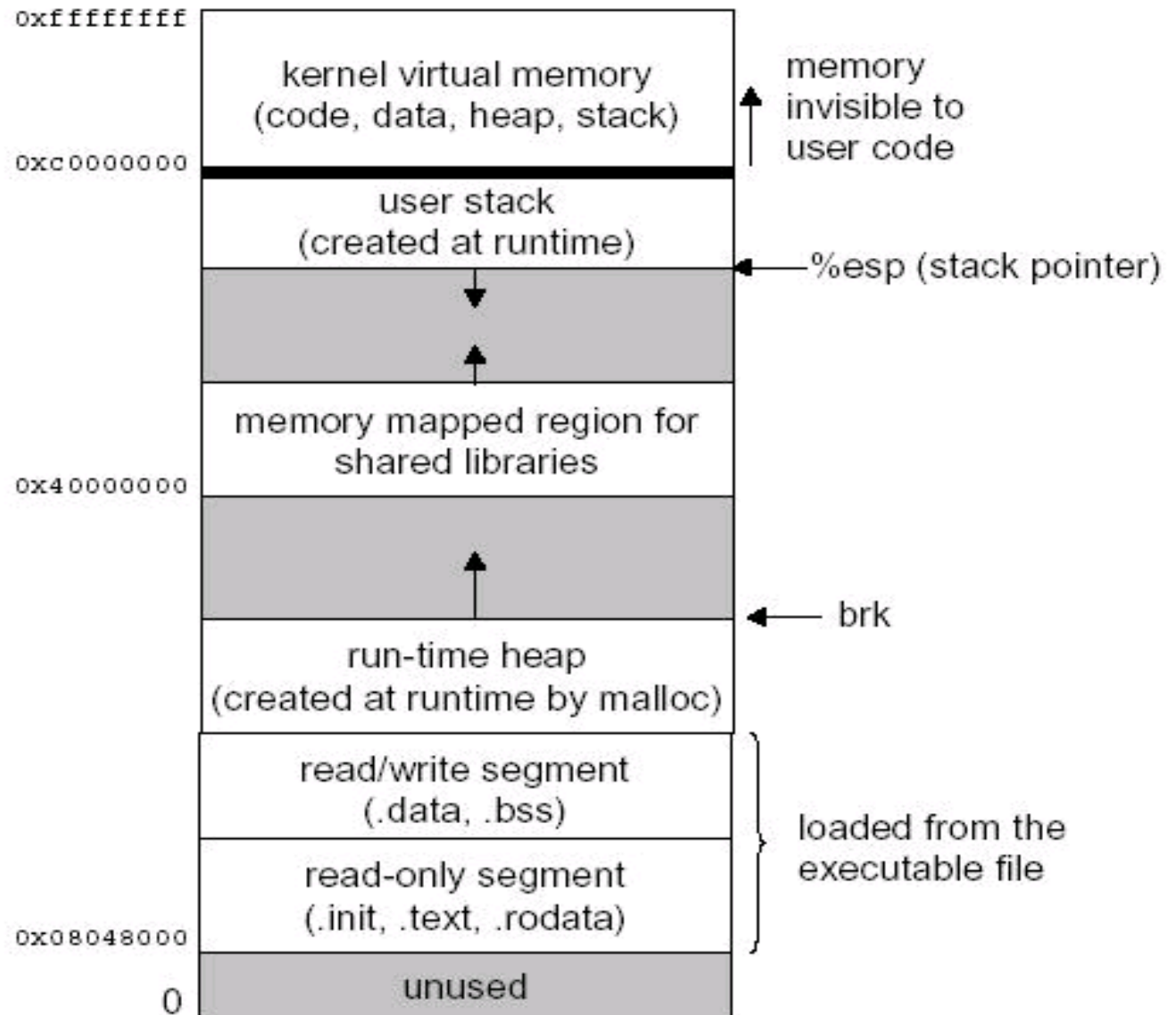
Process Memory Layout

General

Linux

Sample C Program

```c
#include <math.h>

int global;

int f(int farg)
{
        int flocal;


}


main()
{
    int local;
    int *dynamic;
    dynamic = malloc(10, sizeof(int));
    f(1);
    local = sqrt(2.0);
}
```

Addresses

Size of executable file = 314740 bytes

address of f()	1020c	⎫
address of main()	10244	⎪
address of sqrt()	10358	⎬ Code segment
address of printf()	13128	⎪
address of malloc()	31218	⎭
address of global	5004c	Data segment
value of dynamic	50068	Heap segment
address of f()'s flocal	ffbefb4c	⎫
address of f()'s farg	ffbefba4	⎬ Stack segment
address of main()'s local	ffbefbc4	⎭

Logical vs. Physical Address Space

Logical address – address in program instructions that CPU executes; also referred to as **virtual address**

Physical address – address used by memory unit to access physical memory (RAM)
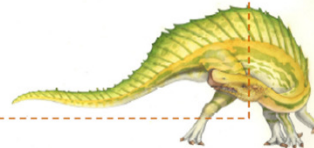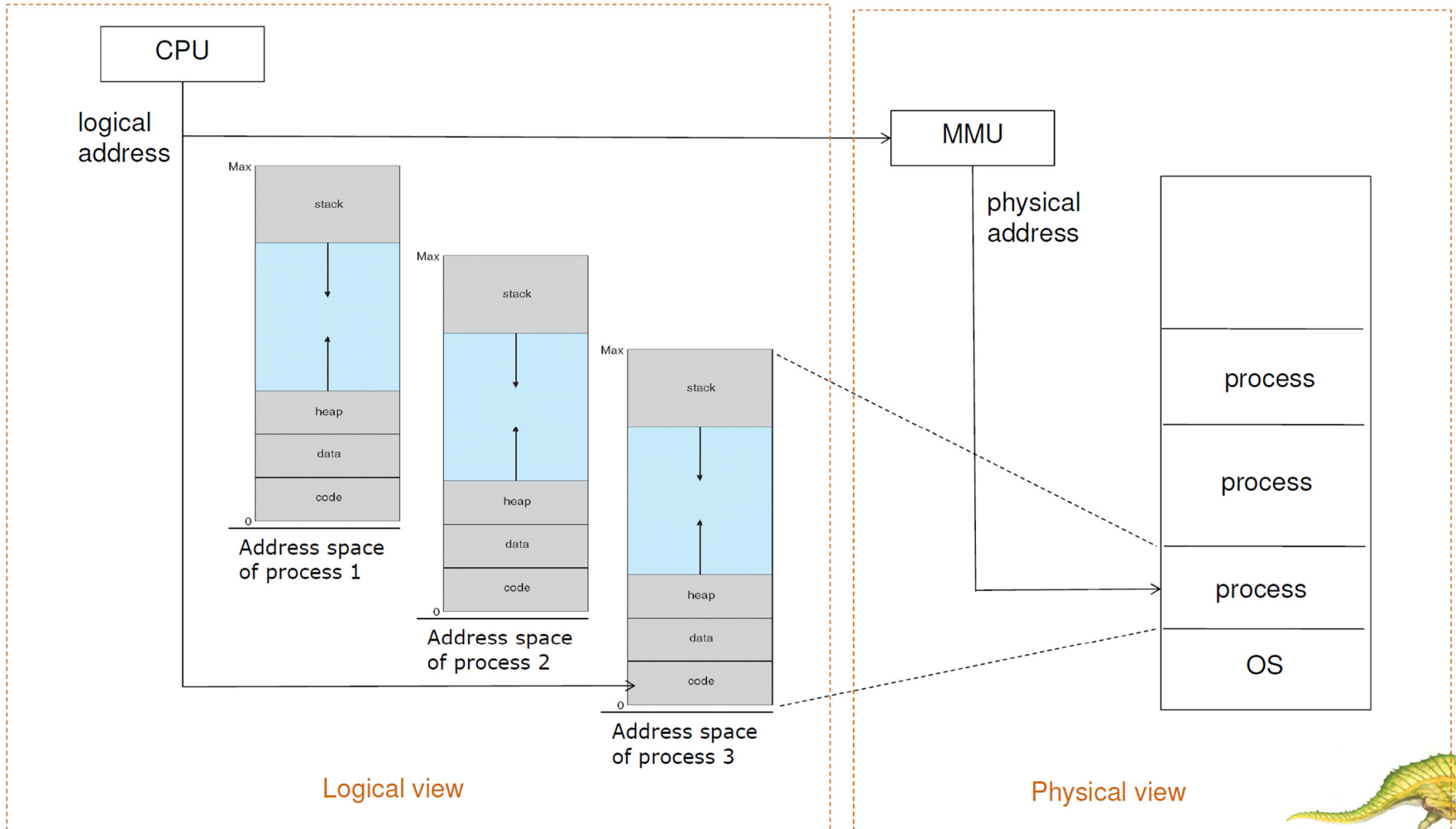
Every logical address must be translated into physical address by Memory Management Unit (MMU).

The user program deals with logical addresses; it never sees the real physical addresses
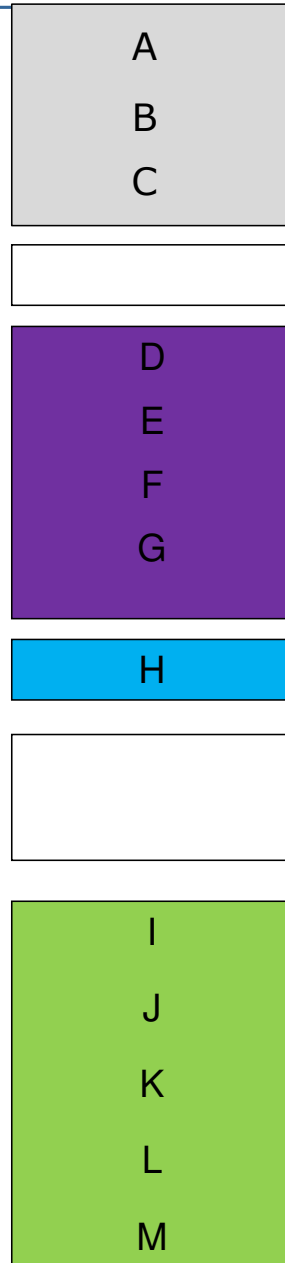
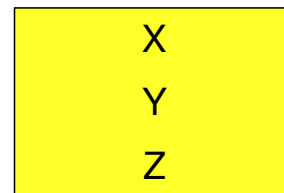# Address Translation

MEMORY ALLOCATION

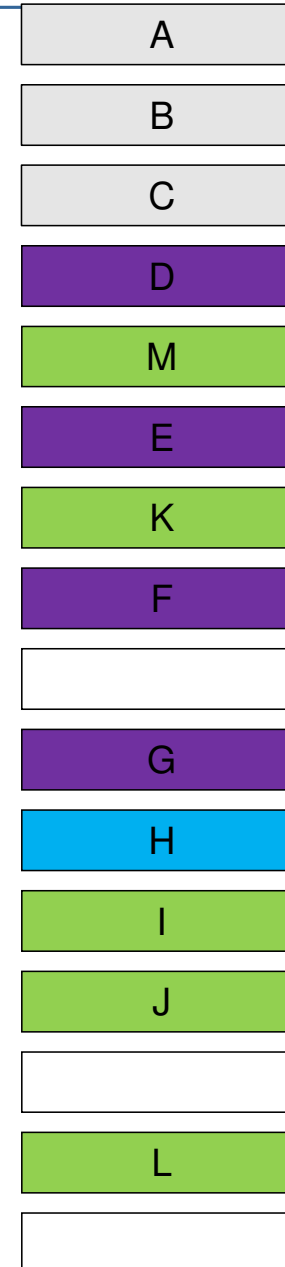| A |
| B |
| C |

| D |
| E |
| F |
| G |

| H |

| I |
| J |
| K |
| L |
| M |

Cannot find space of new process

| X |
| Y |
| Z |

| A |
| B |
| C |
| D |
| M |
| E |
| K |
| F |

| G |
| H |
| I |
| J |

| L |

Can find space of new process

| X |
| Y |
| Z |
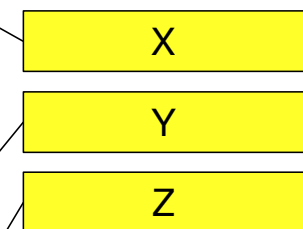
Paging

Physical memory is divided into fixed-sized blocks called **frames**

Logical memory is divided into blocks of <u>same size</u> called **pages.**

Size of frames and pages is power of 2, e.g. 512, 1K, 2K, **4K**, 8K bytes

OS keeps track of all free frames

To run a program of size n pages, need to find n free frames and load program

Use a page table is used to translate logical to physical addresses

Address Translation Scheme

Logical Address is divided into:

Page number (*p*) – used as an index into a *page table* which contains base address of each page in physical memory

Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
p	*d*
$m-n$ bits	n bits

For given logical address space 2^m *and page size* 2^n

Example: page no. and offset

Address width	32 bits
Page size	4K bytes
Offset	12 bits
Page number	20 bits

	page	offset
f()	10	20c
main()	10	244
sqrt()	10	358
printf()	13	128
malloc()	31	218
global	50	04c
flocal	ffbef	b4c
farg	ffbef	ba4

Paging Hardware
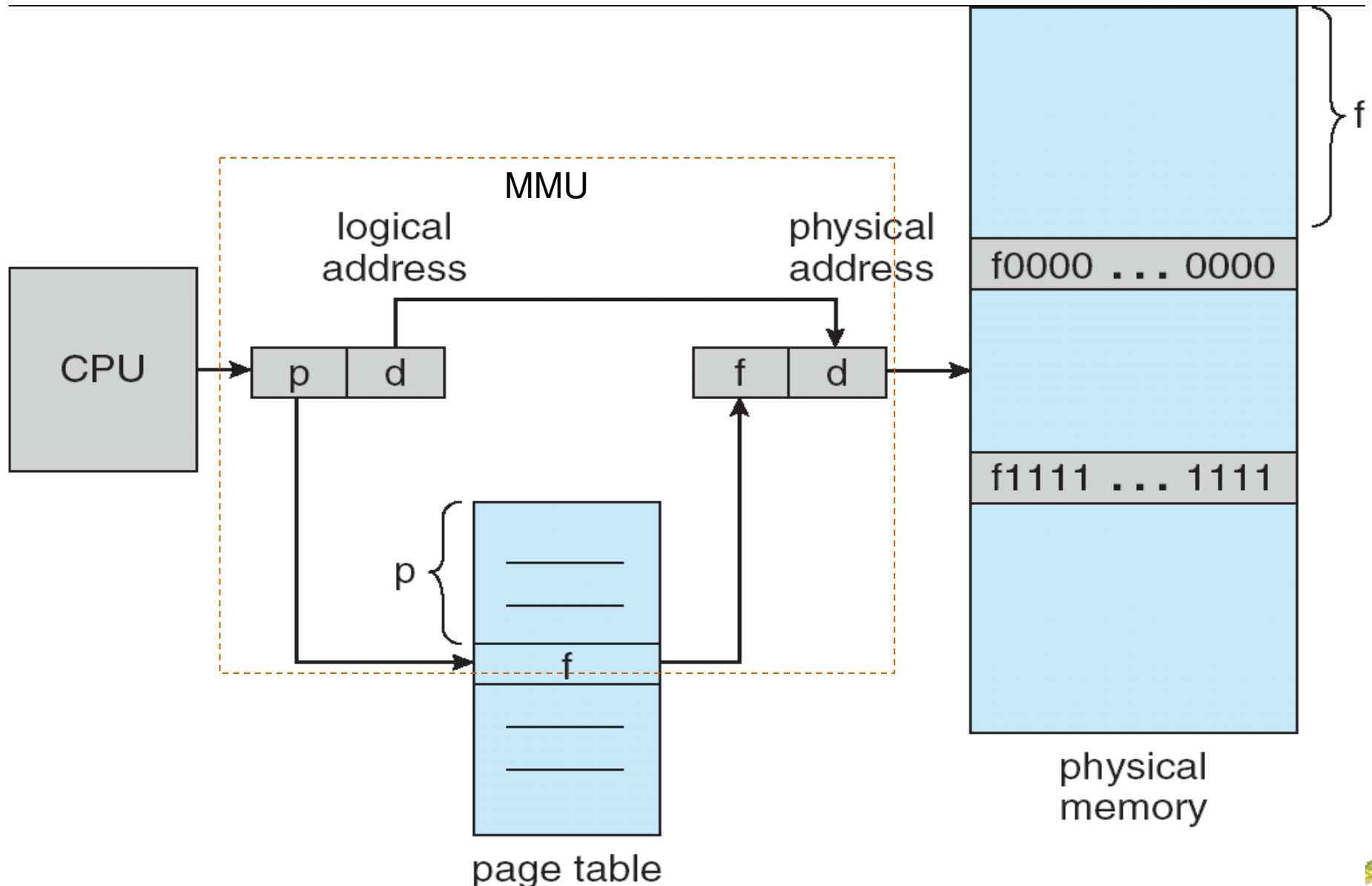
Paging Model of Logical and Physical Memory

page 0
page 1
page 2
page 3
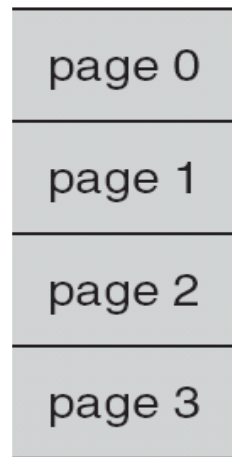
logical memory

0	1
1	4
2	3
3	7

page table

0
1 page 0
2
3 page 2
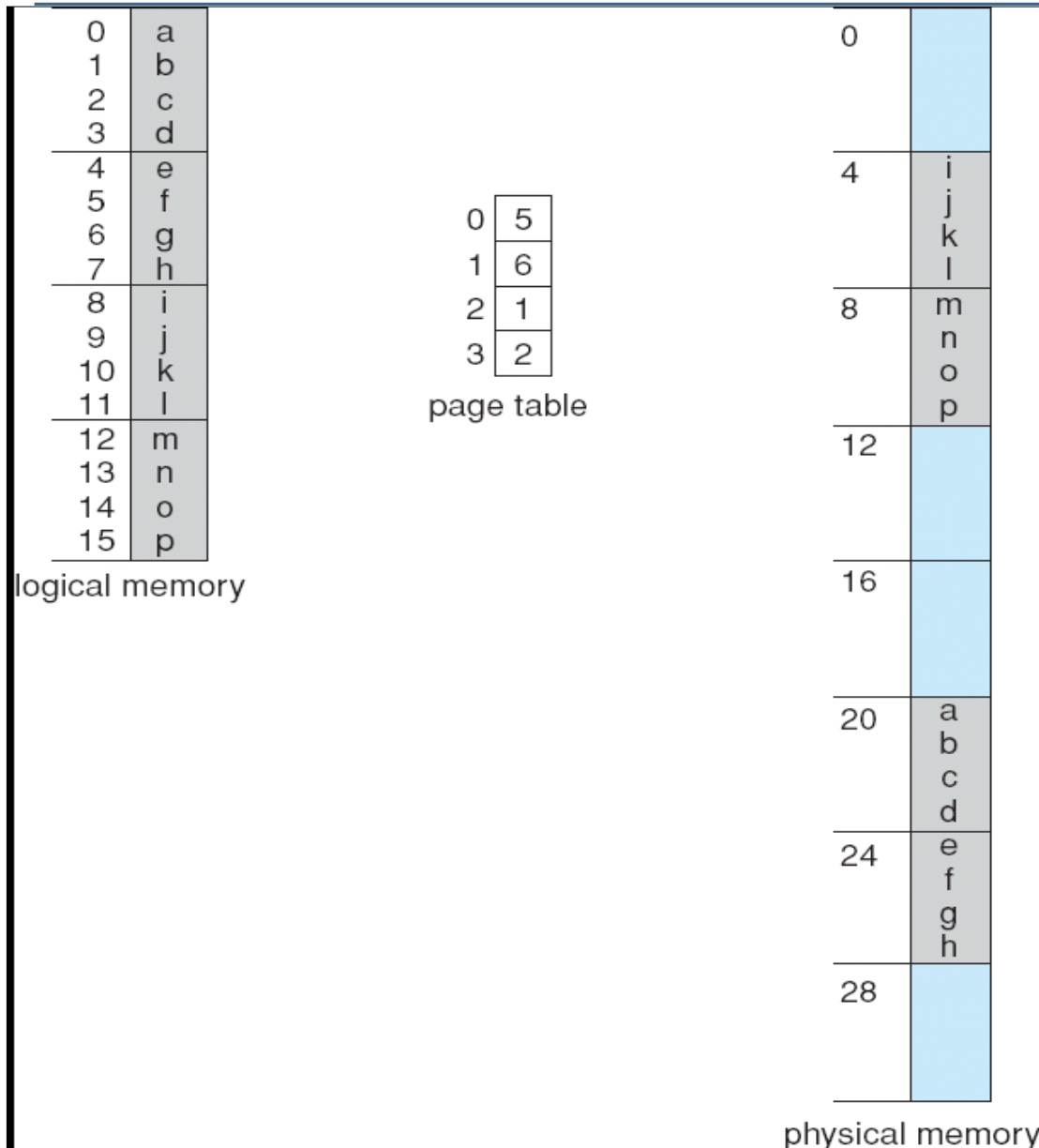4 page 1
5
6
7 page 3

physical memory

page table
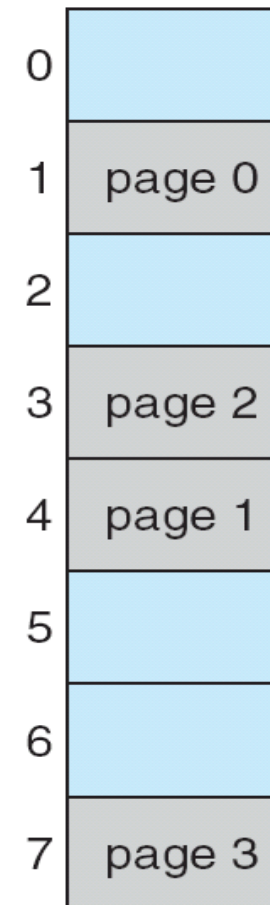
logical memory

physical memory

16-byte logical memory

4-byte page size → 4 pages

4-bit address → 2-bit page number, 2-bit offset

32-byte physical memory
4-byte frame size → 8 frames
5-bit address → 3-bit frame number, 2-bit offset

Ex.
```
Logical address    =  1001
Page number        =  10
Frame              = 001
Offset             =    01
Physical address   = 00101
Data               = 'j'
```

Translation Look-Aside Buffer

Page table is kept in main memory

Two memory access problem: Every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The problem can be solved by using a special hardware cache called **translation look-aside buffers (TLBs)**

Keep recently used page table entries

Address translation (p, d)

Search for p in TLB

If p is found (TLB hit), get frame no. out

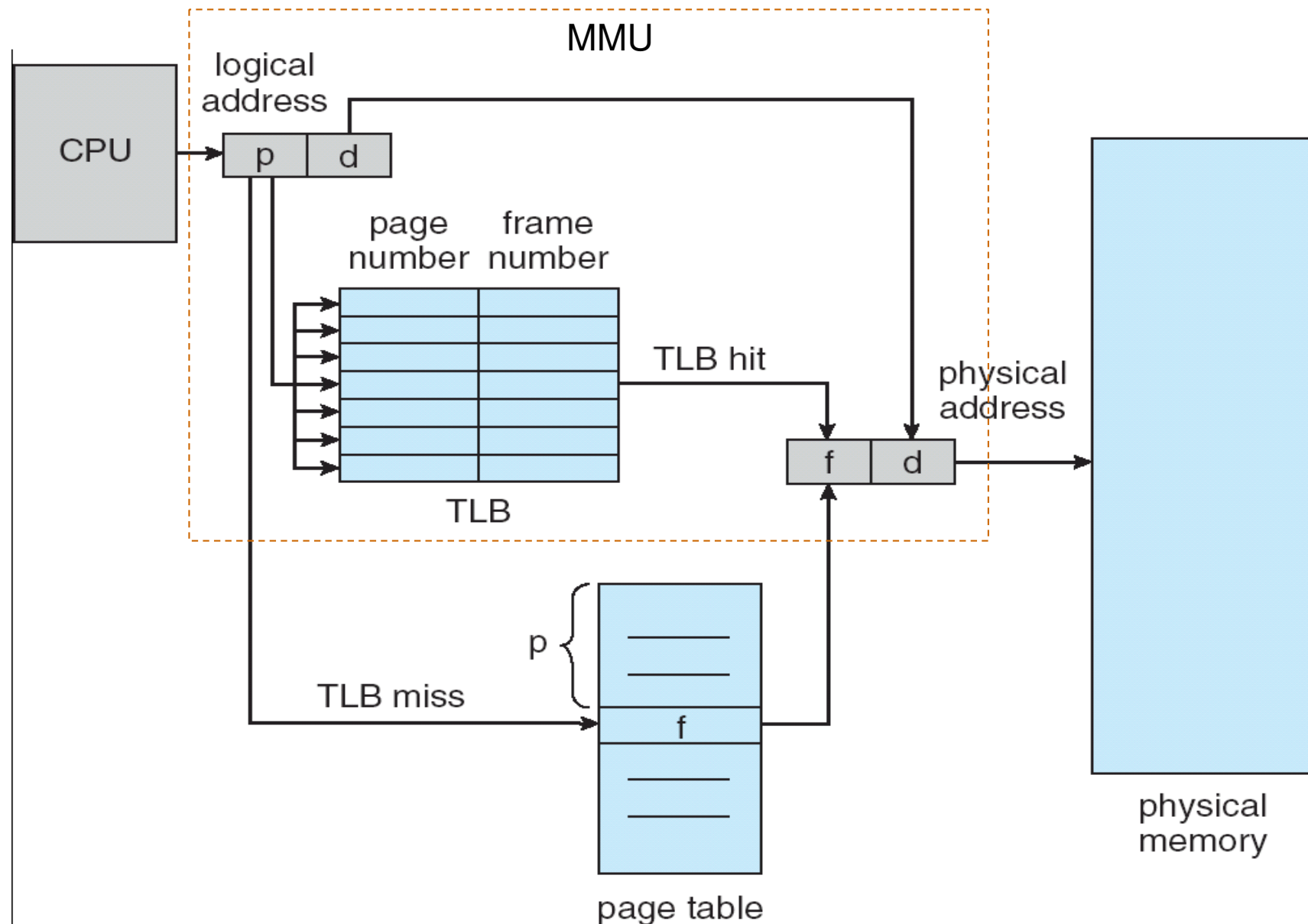Otherwise (TLB miss), get frame # from page table in memory

Paging Hardware With TLB

Effective Access Time

Associative Lookup = ε time unit

Memory access time = m time unit

Hit ratio – percentage of times that a page number is found in TLB

Hit ratio = α

Effective Access Time (EAT)

$$\text{EAT} = (\varepsilon + m)\,\alpha + (\varepsilon + 2m)(1 - \alpha)$$

$$= \varepsilon\alpha + m\,\alpha + \varepsilon + 2m - \varepsilon\,\alpha - 2m\alpha$$

$$= 2m - \alpha m + \varepsilon$$

Structure of the Page Table

For 32-bit virtual address space with 4 KB page size, page table may have up to 1 million ($2^{32}/2^{12}$) entries (per process!)

Intel Itanium 2 processor supports 64-bit virtual address space and 50-bit physical address space with different page sizes (4kB, 8kB, 16kB, 256kB, 1MB, 4MB, 16MB, 64MB and 256MB)

Current operating systems support up to 50-bit virtual address space

A lot of pages between heap and stack is unused, but page table entries are still allocated for them (if page table is an array)

Aim: reduce size of page table

Common page table structures

    Hierarchical Paging

    Hashed Page Tables

    Inverted Page Tables

Hierarchical Page Tables

Break up the logical address space into multiple page tables
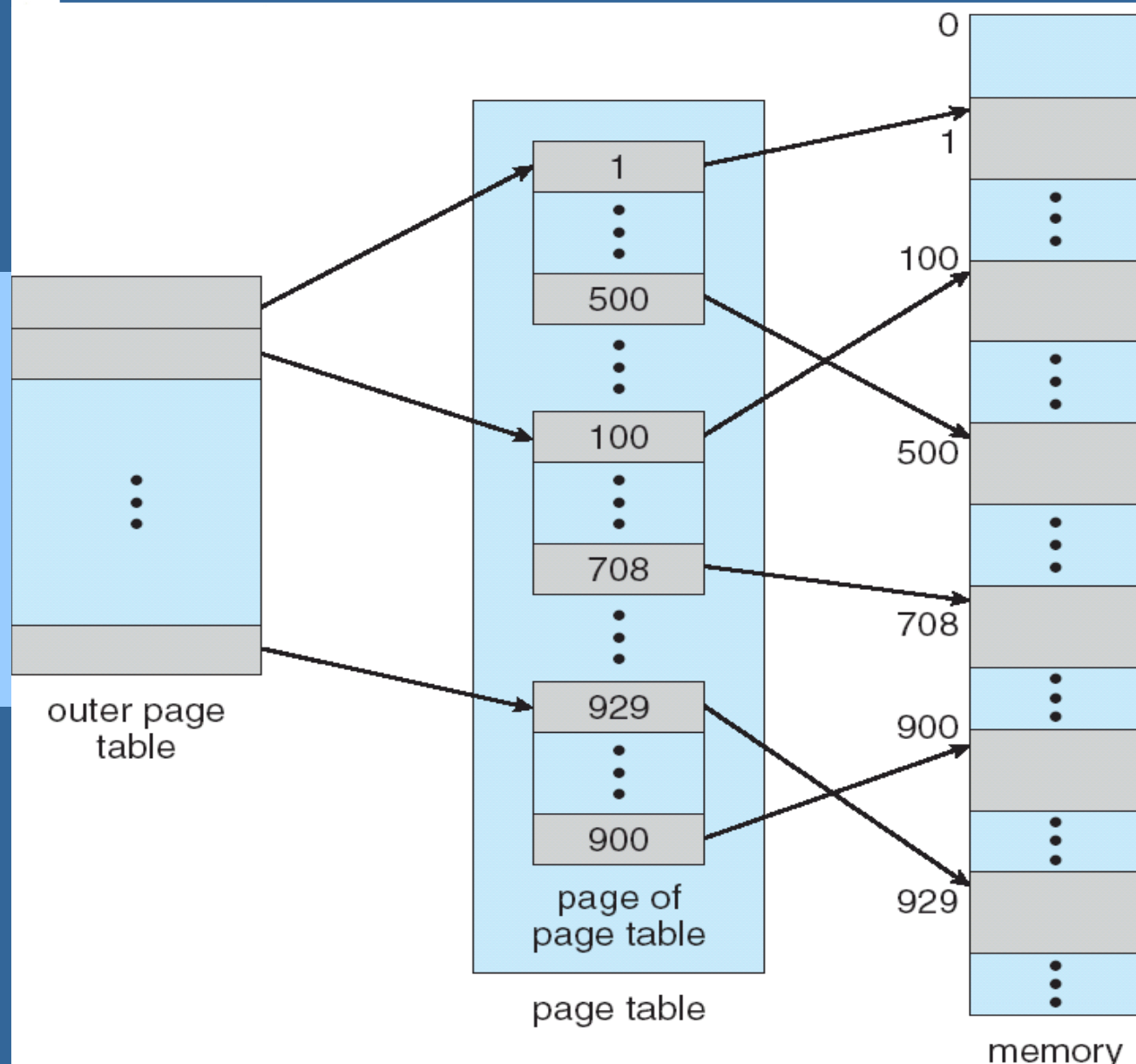
A simple technique is a two-level page table

Page table is also paged

Two-Level Page-Table Scheme

Outer page table
- contains all entries for the whole logical address space
- each entry associates to a lot of pages
- reduce number of entries
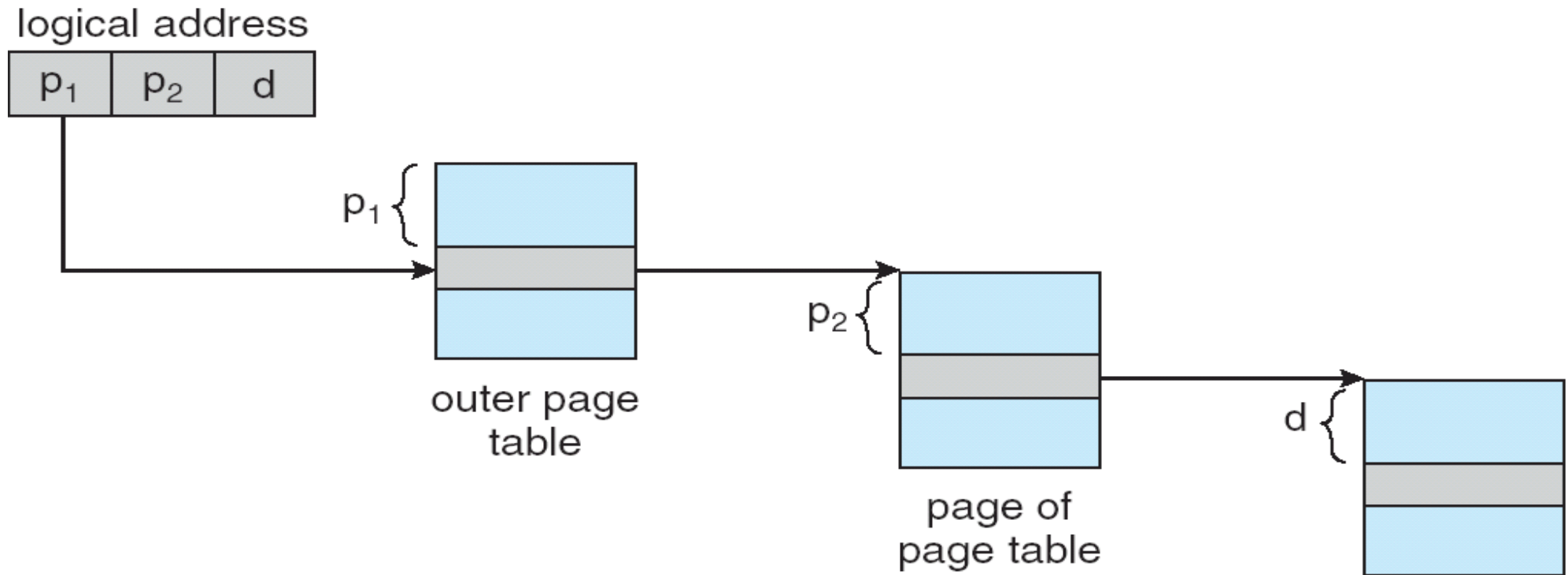- entries for unused memory area are null

page table
- contains pages of page table for only used pages
  - dynamically allocated
  - reduce number of entries

Two-Level Address-Translation

Two-Level Paging Example

A logical address (on 32-bit machine with 1K page size) is divided into:

- a page number consisting of 22 bits
- a page offset consisting of 10 bits

Since the page table is paged, the page number is further divided into:

- a 12-bit page number
- a 10-bit page offset

Thus, a logical address is:

page number		page offset
p_1	p_2	d
12	10	10

where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Outer page table has 4K entries

page table contains pages of page table, each has 1K entries

Three-level Paging Scheme

outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12