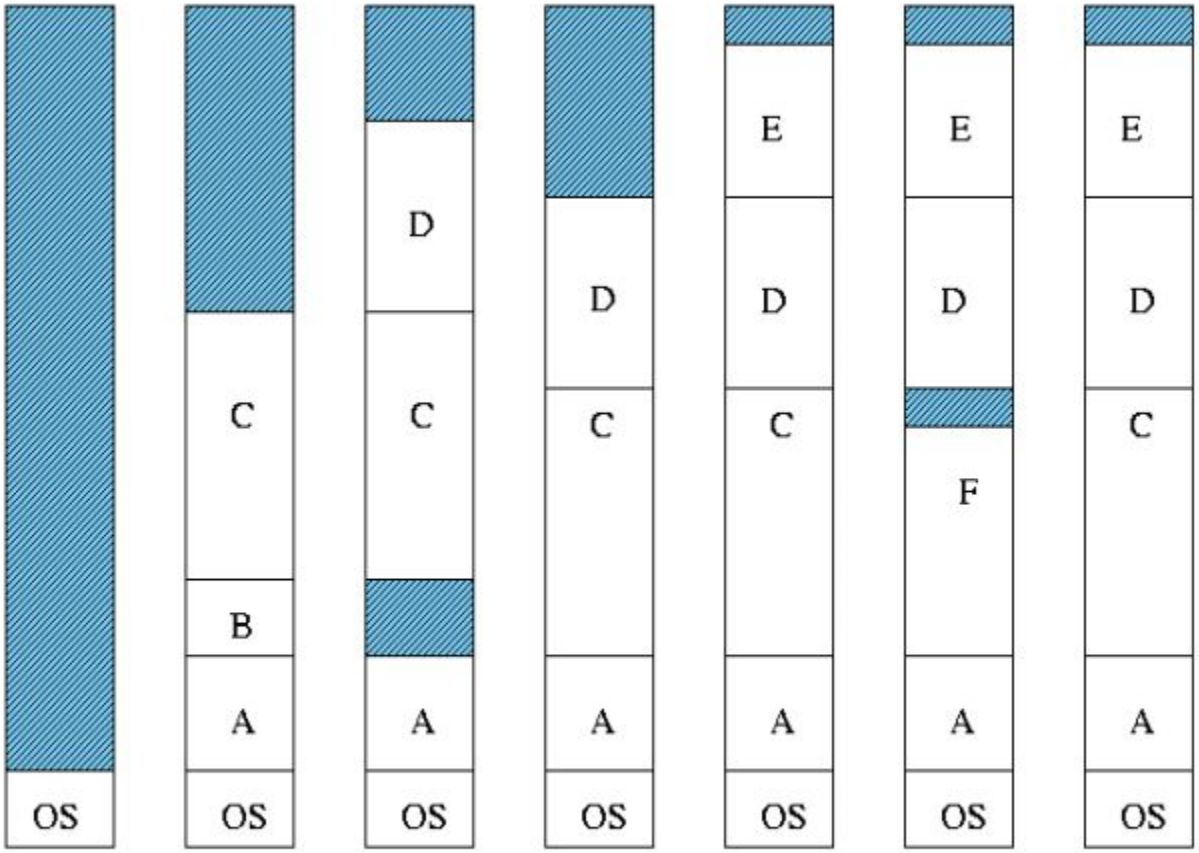
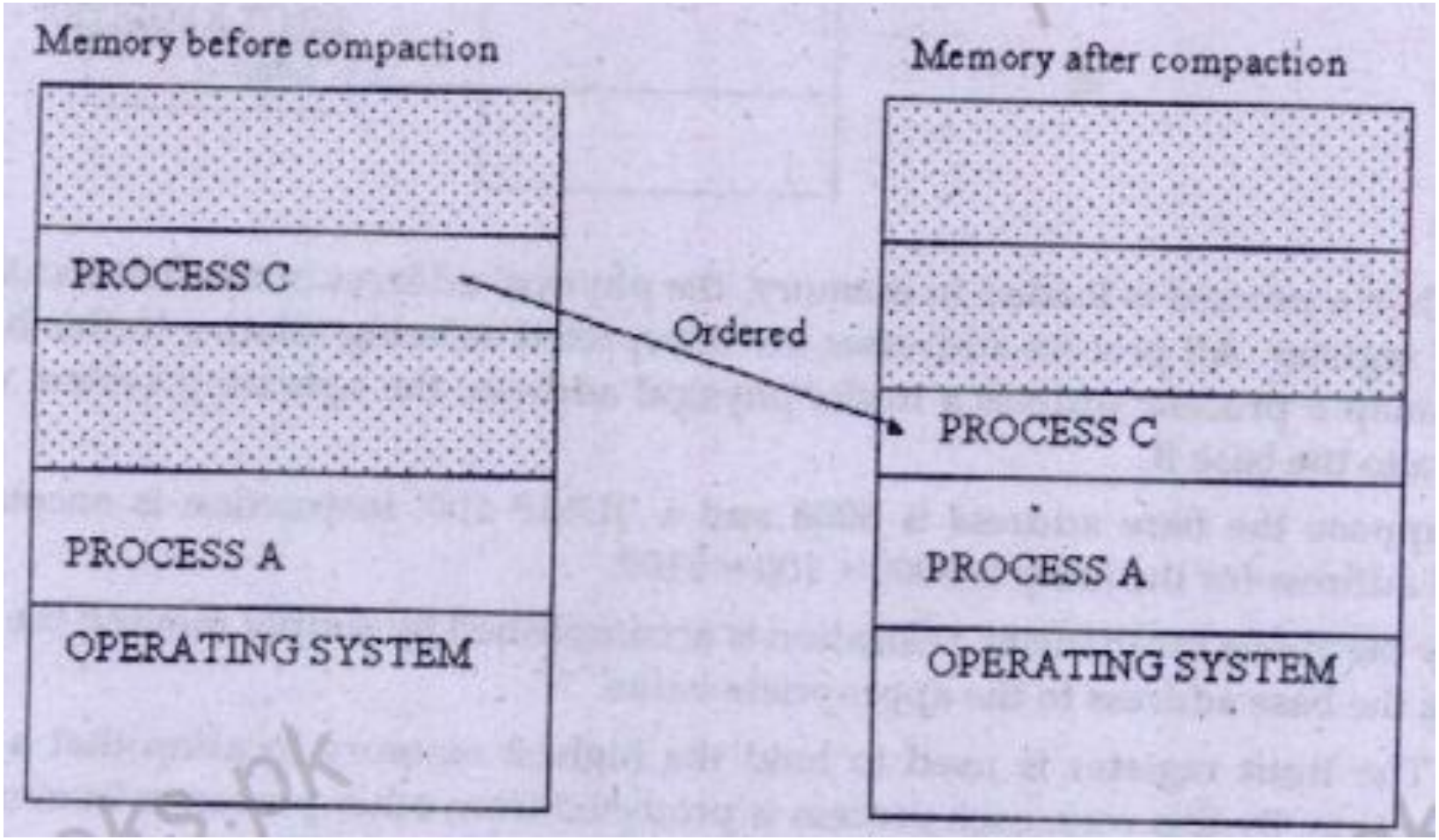
Operating Systems

Memory management PART2
Non-Contiguous Memory management scheme

Paging and Segmentation



Multiprorgamming with Variable Tasks (MVT), external fragmentation, and compaction


Memory before compaction
PROCESS C
PROCESS A
OPERATING SYSTEM
Ordered
Memory after compaction
PROCESS C
PROCESS A
OPERATING SYSTEM

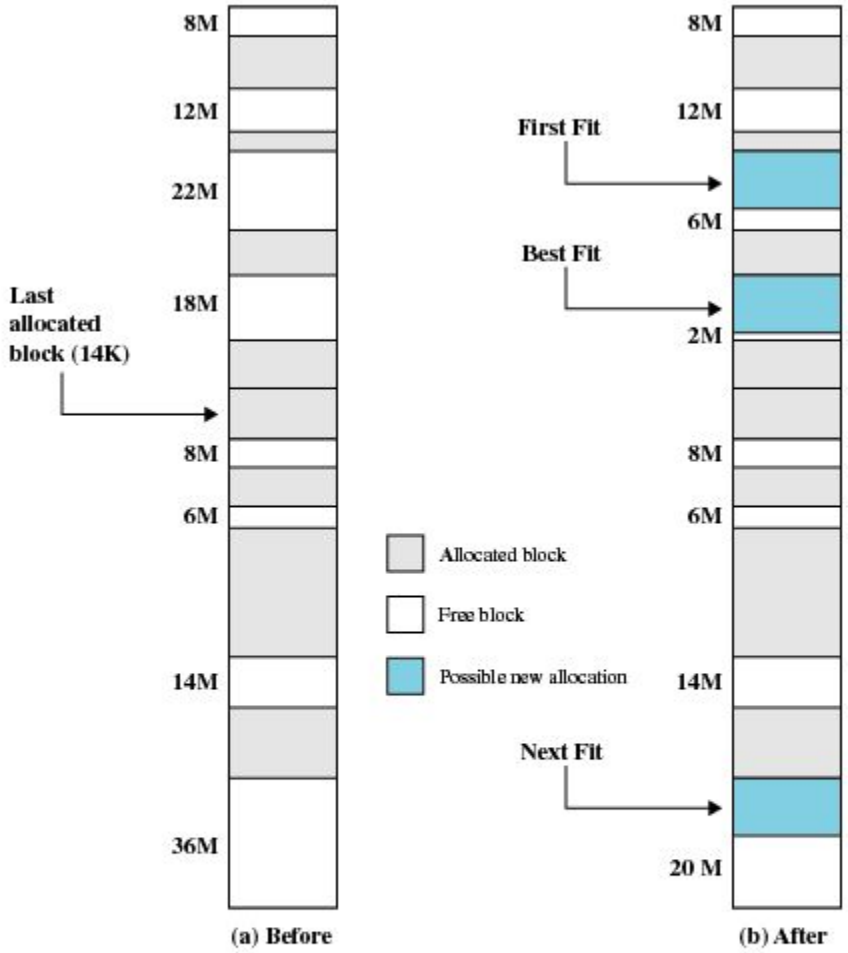


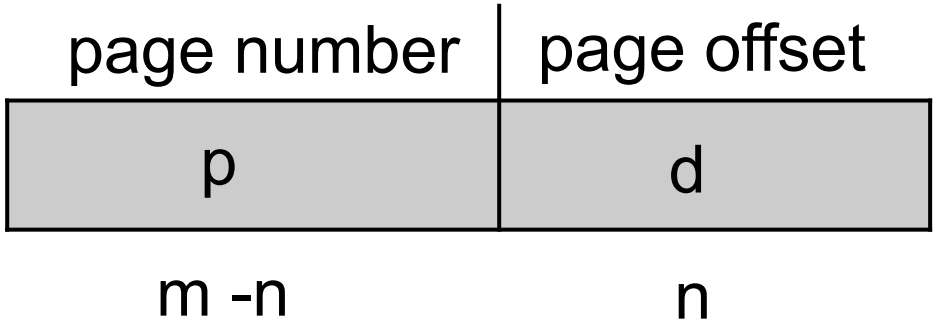
Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Overview

- Paging
- Page Tables
- TLB
- Hierarchical Pages
- Hashed Pages
- Inverted Pages
- Segmentation

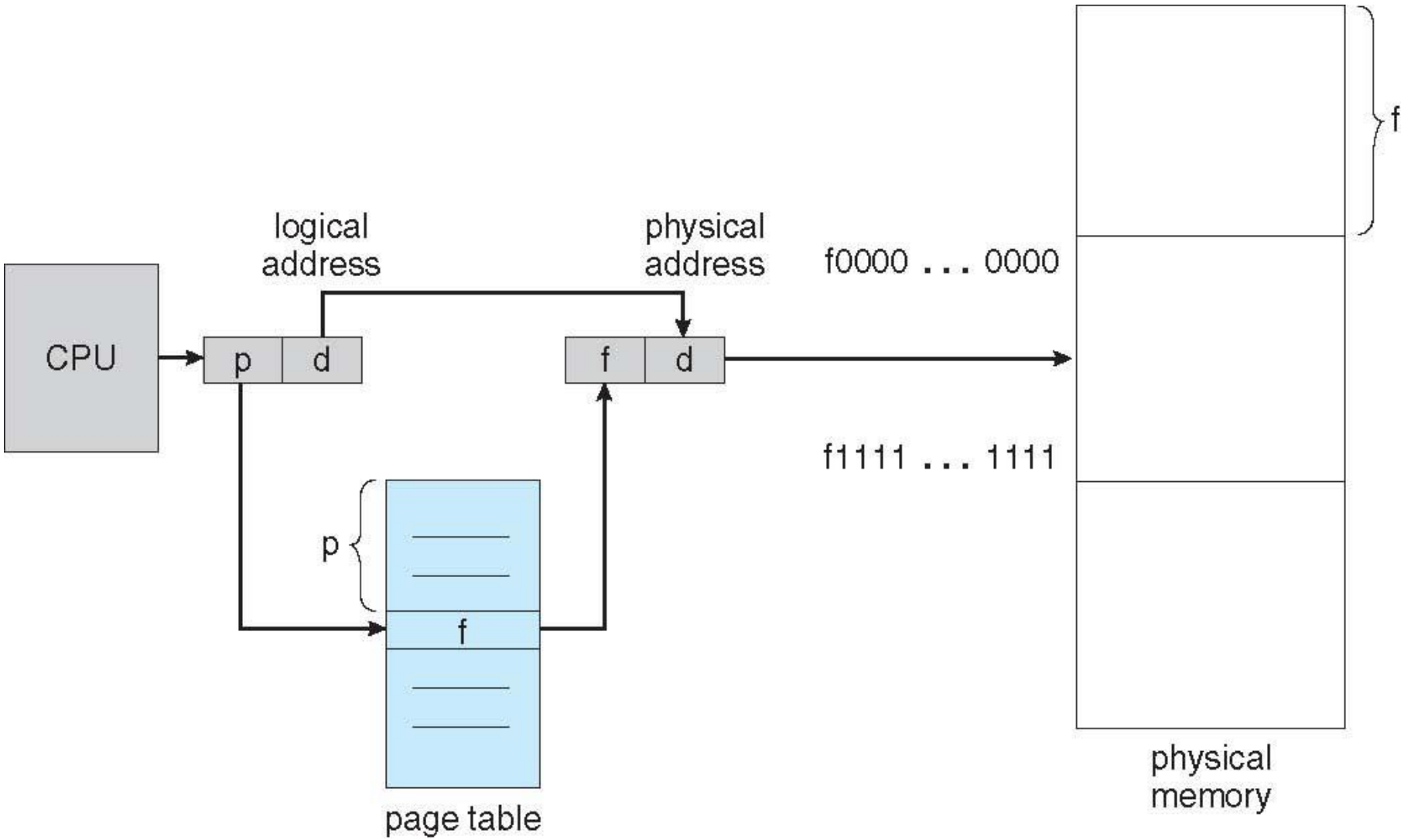
Address Translation Scheme

- Address generated by CPU 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

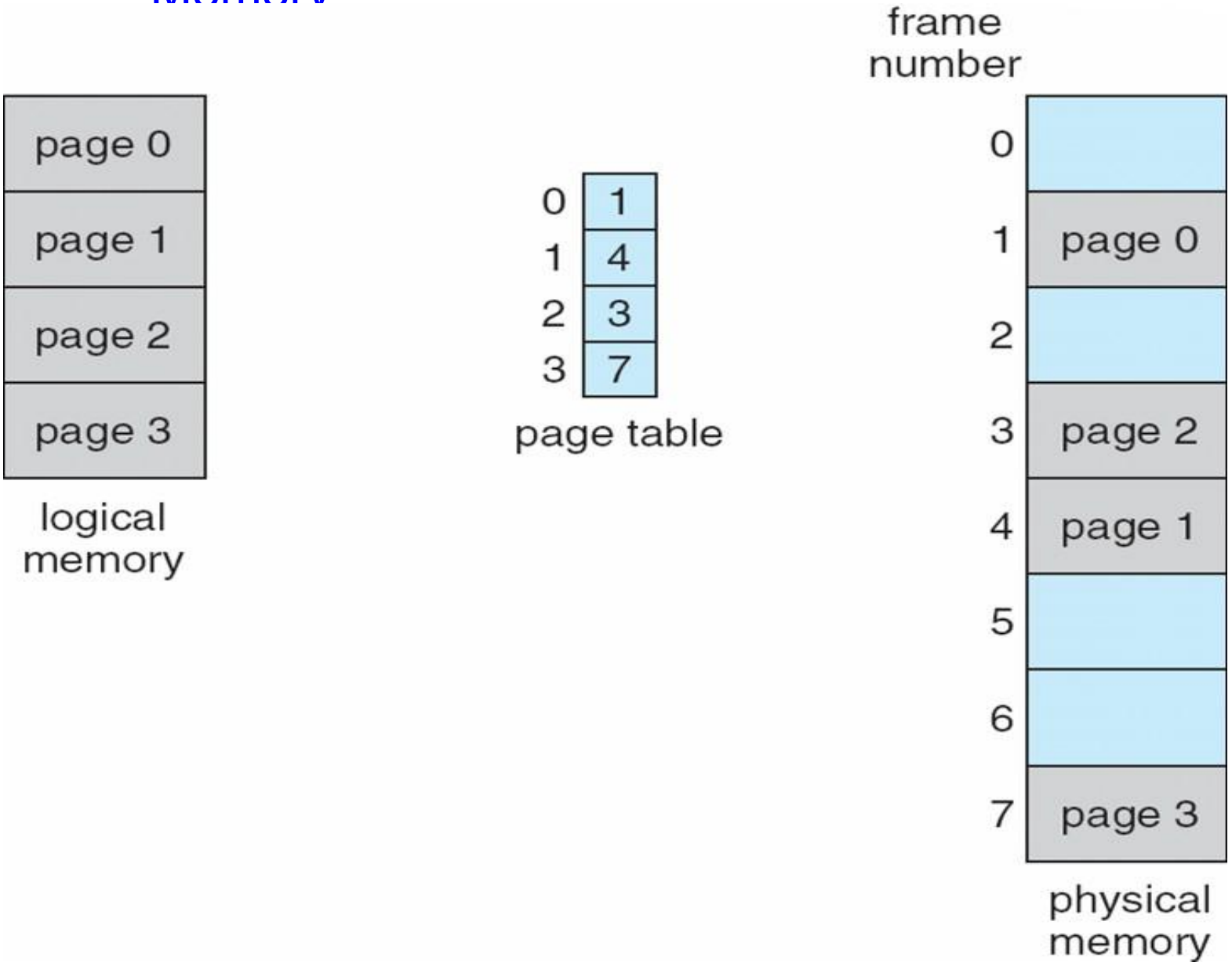


  - For given logical address space 2^m and page size 2^n

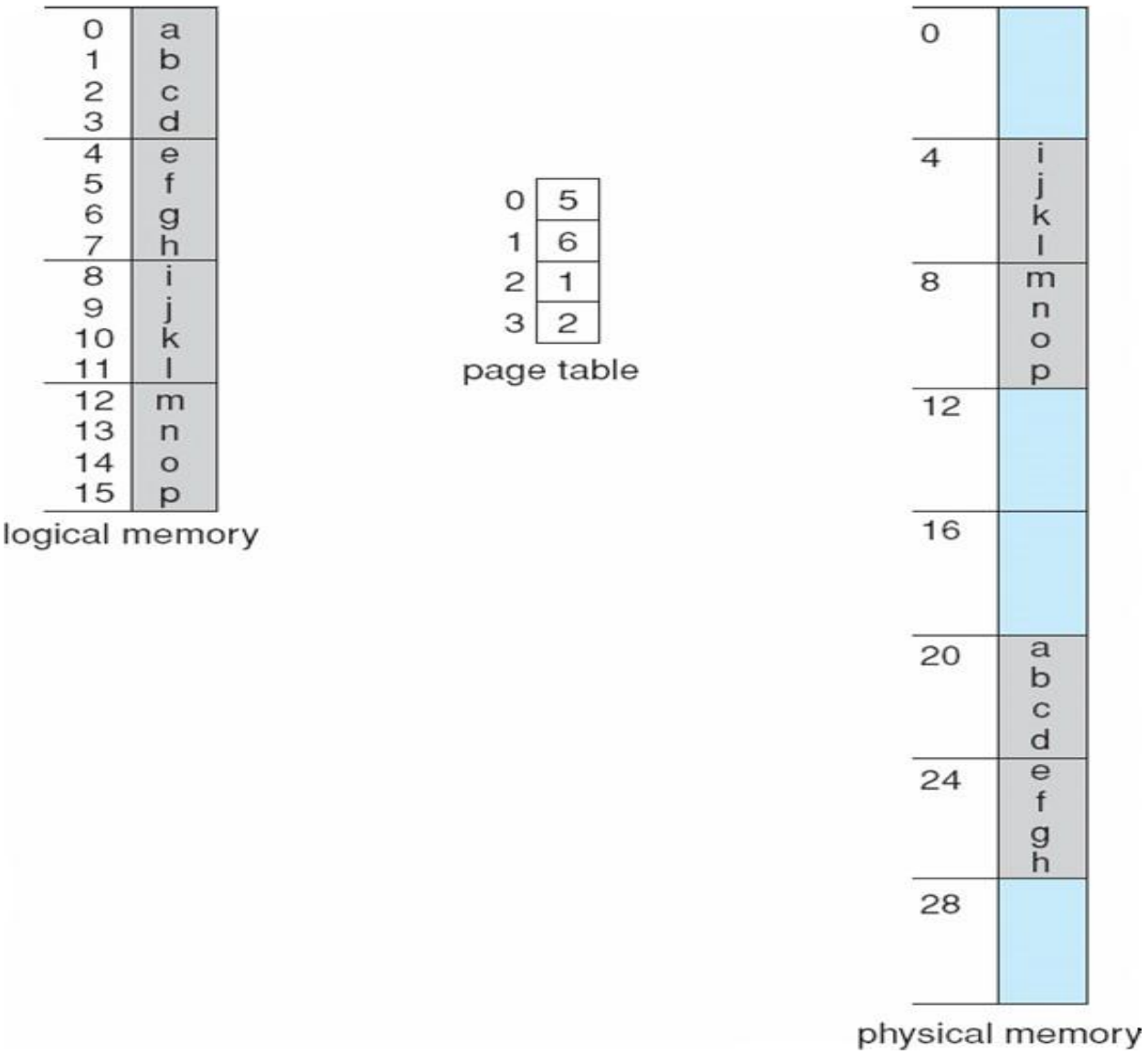
Paging Hardware



Paging Model of Logical and Physical Memory



Paging Example



n=2 and *m*=4 32-byte memory and 4-byte pages

Logical address				
Page	Displacement			
	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

Logical memory

0	1
1	4
2	3
3	7

Page table

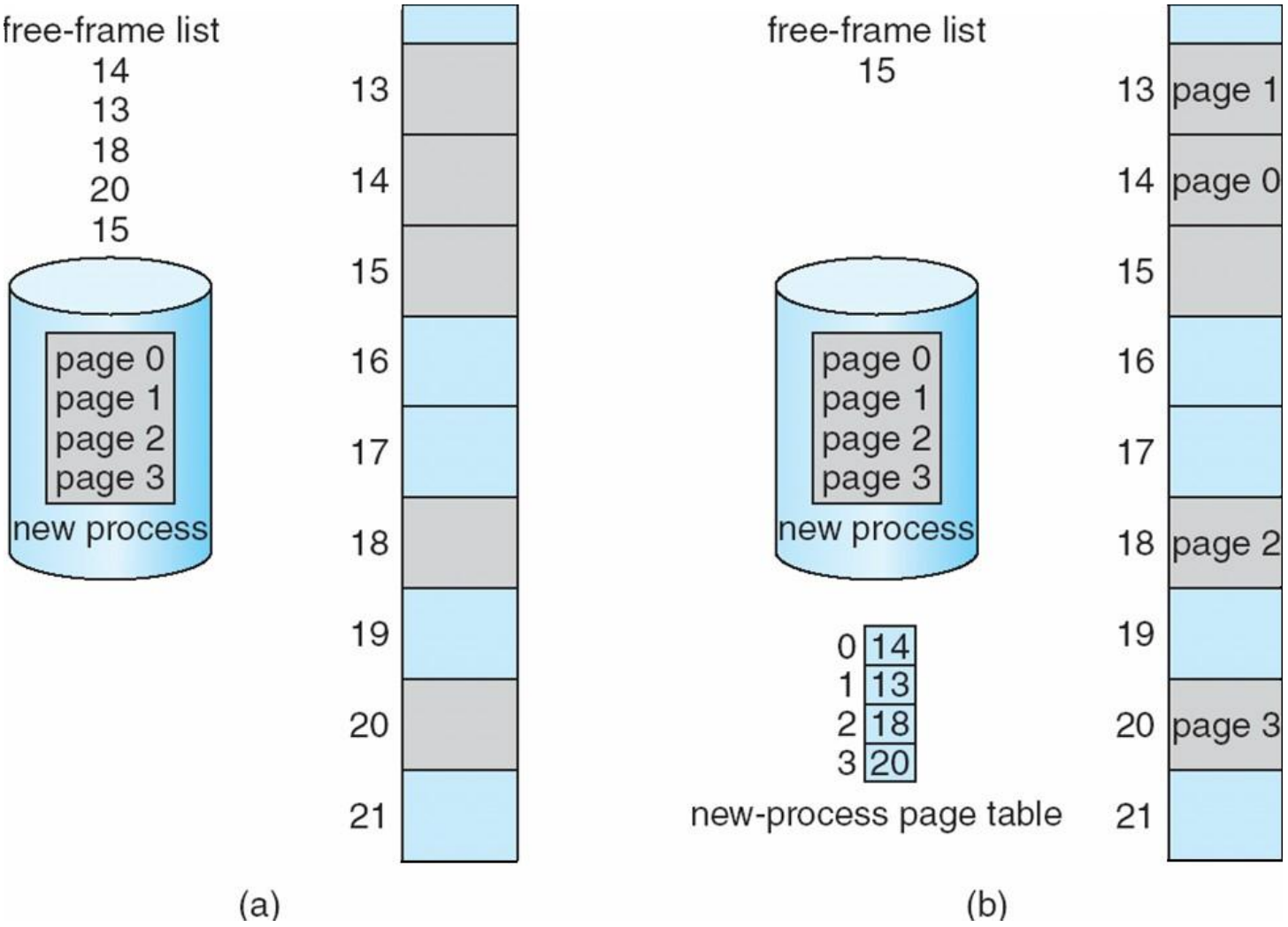
Frame	Address			
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31

Physical memory

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory

Free Frames

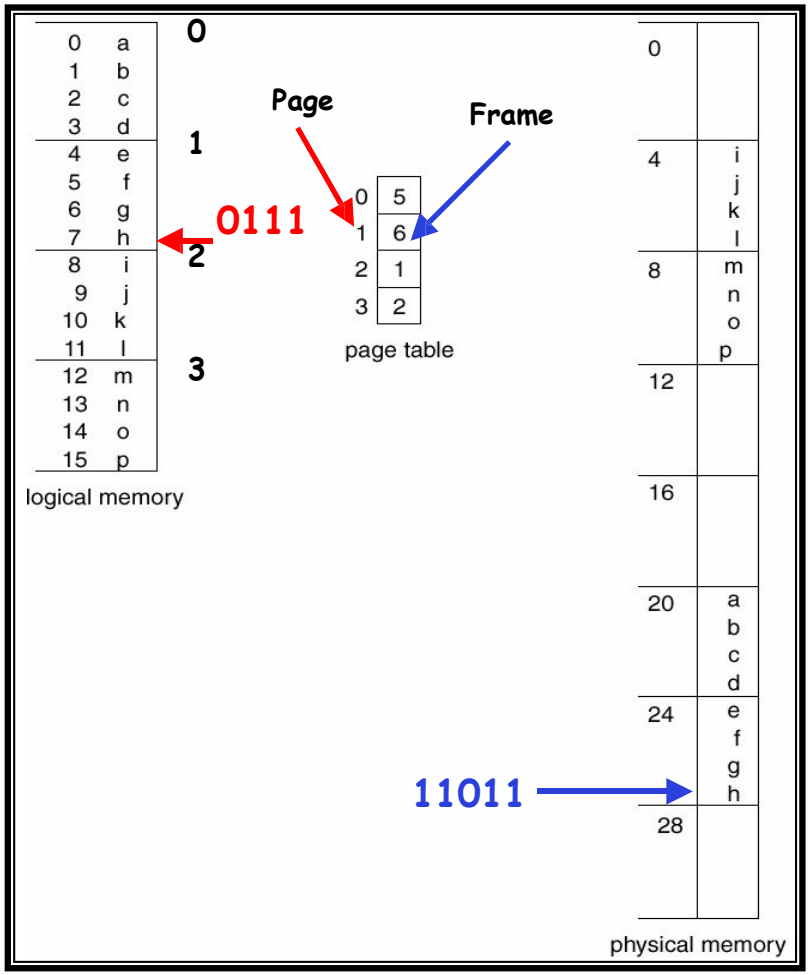


Before allocation

After allocation

Paging Example

- Page size = 4 bytes
- Process address space
  = 4 pages
- Physical address space
  = 8 frames
- Logical address: (1,3)
  = 0111
- Physical address: (6,3)
  = 11011



Addressing in Paging

- **Logical address space** of 16 pages of 1024 words each, mapped into a physical memory of 32 frames.
- **Logical address** size?
- **Physical address** size?
- Number of bits for p, f, and d?

Addressing in Paging

- No. of bits for **p** = 4 bits
- No. of bits for **f** = 5 bits
- No. of bits for **d** = 11 bits

Addressing in Paging

Logical address size = |p| + |d|
= 4+11
= 15 bits

Physical address size = |f| + |d|
= 5+11
= 16 bits

Page Table Size

Page table size = NP * PTES

where NP is the number of pages in the process address space and PTES is the page table entry size (equal to |f| based on our discussion so far).

Page table size = 16 * 5 bits

= 16 bytes

Another Example

- Logical address = 32-bit
- Process address space = 2^{32} B
  = 4 GB
- Main memory = RAM = 512 MB
- Page size = 4K
- Maximum pages in a process address space = 2^{32} / 4K
  = 1M

Another Example

- |d| = 12 bits
- |p| = 32 – 12 = 20 bits
- No. of frames = 512 M / 4 K
  = 128 K
- |f| = 17 bits

- Physical address = 17+12 bits

Page Table Entries in Page Table

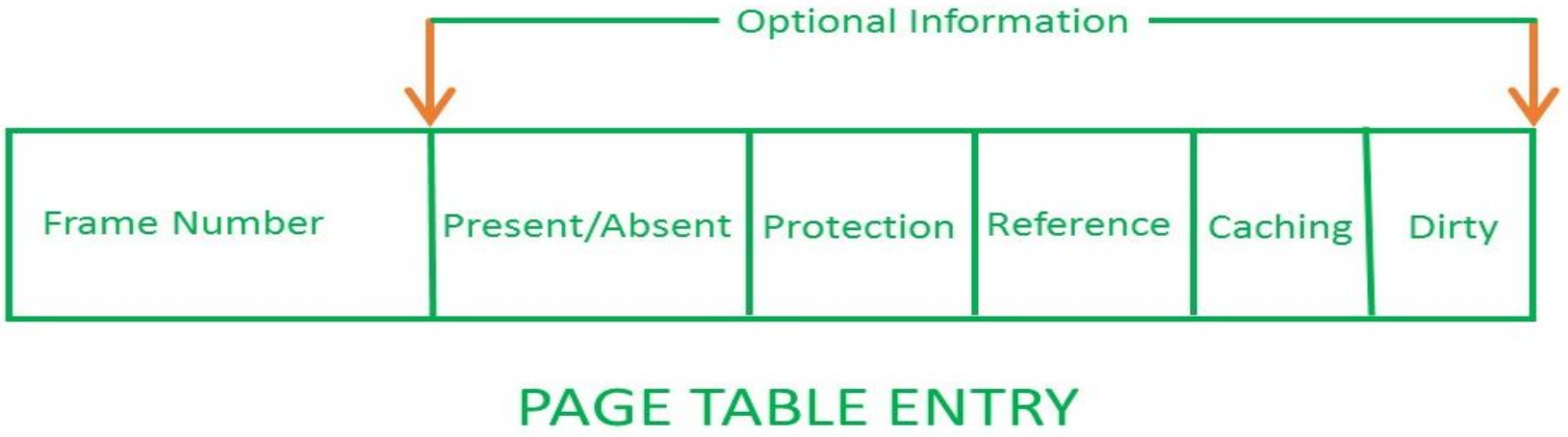
Frame Number – It gives the frame number in which the current page you are looking for is present.
The number of bits required depends on the number of frames. Frame bit is also known as address translation bit.
`Number of bits for frame = Size of physical memory/frame size`

Other Optional bits

1. **Present/Absent bit** – Present or absent bit says whether a particular page you are looking for is present or absent. In case if it is not present, that is called Page Fault. It is set to 0 if the corresponding page is not in memory. Used to control page fault by the operating system to support virtual memory. Sometimes this bit is also known as **valid/invalid** bits.
2. **Protection bit** – Protection bit says that what kind of protection you want on that page. So, these bit for the protection of the page frame (read, write etc).
3. **Referenced bit** – Referenced bit will say whether this page has been referred in the last clock cycle or not. It is set to 1 by hardware when the page is accessed.

Page Table Entries in Page Table

4. **Caching enabled/disabled** – Some times we need the fresh data. this bit **enables or disable** caching of the page.
5. **Modified bit** – Modified bit says whether the page has been modified or not. Modified means sometimes you might try to write something on to the page. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk or it has to be written back or it has to be saved back. It is set to 1 by hardware on write-access to page which is used to avoid writing when swapped out. Sometimes this modified bit is also called as the **Dirty bit**.



Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two memory access problem can be solved
  - by the use of a special fast-lookup hardware cache
  - called **associative memory** or **translation look-aside buffers** (**TLBs**)

Implementation of Page Table

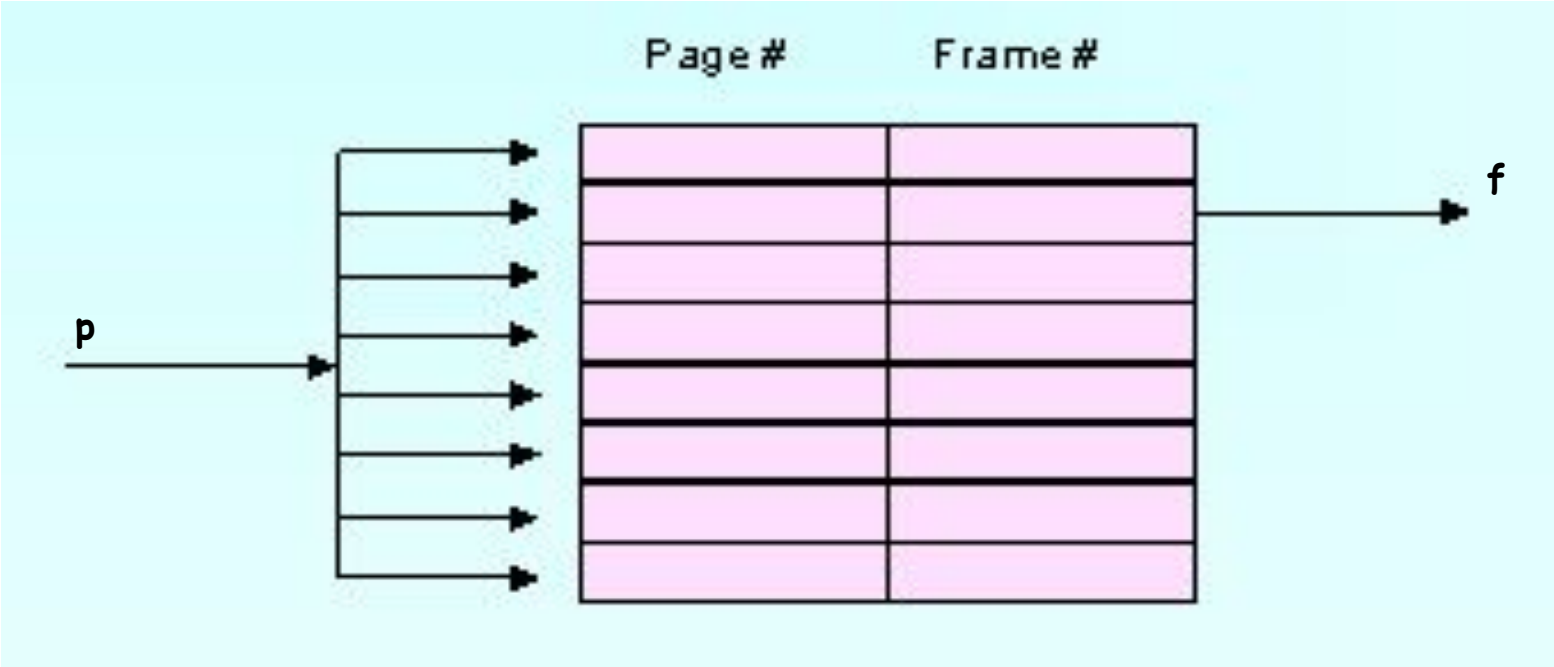
- Keep page table in the main memory
- Page table base register (PTBR)
- $T_{effective} = 2T_{mem}$
- $T_{effective}$ is not acceptable
- Use a special, small, fast lookup hardware, called translation look-aside buffer (TLB)
- Typically 64–1024 entries
- An entry is (key, value)
- Parallel search for key; on a hit, value is returned

Implementation of Page Table

- (key,value) is (p,f) for paging
- For a logical address, (p,d), TLB is searched for **p**. If an entry with a key **p** is found, we have a hit and **f** is used to form the physical address. Else, page table in the main memory is searched.

TLB

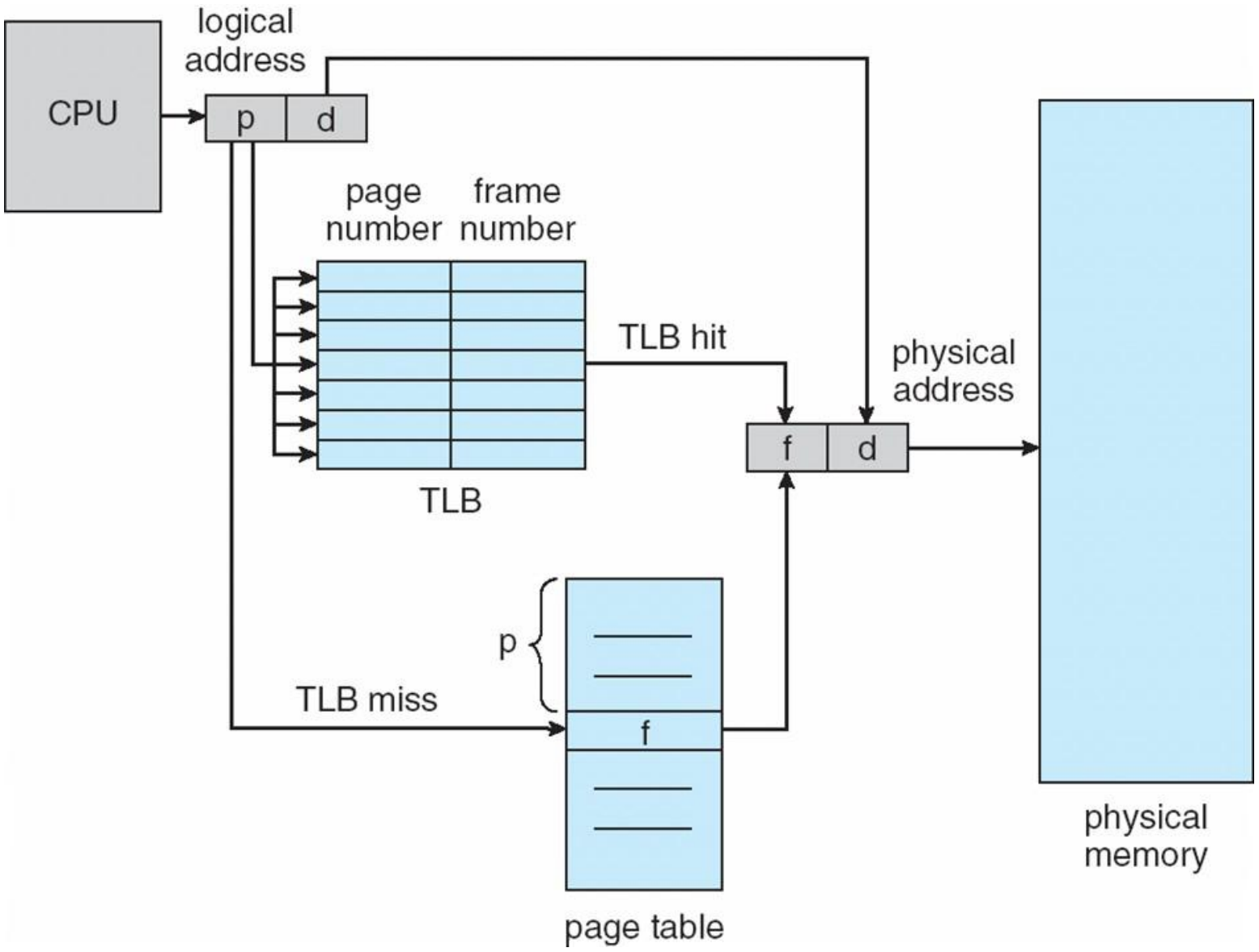
Logical address:
(p, d)



Implementation of Page Table

- The TLB is loaded with the (p,f) pair so that future references to p are found in the TLB, resulting in improved hit ratio.
- On a context switch, the TLB is flushed and is loaded with values for the scheduled process.

Paging Hardware With TLB



Performance of Paging

- $T_{effective}$ on a hit = $T_{mem} + T_{TLB}$
- $T_{effective}$ on a miss = $2T_{mem} + T_{TLB}$
- If HR is hit ratio and MR is miss ratio, then

$$T_{effective} = HR\,(T_{TLB} + T_{mem}) + MR\,(T_{TLB} + 2T_{mem})$$

Example

- T_{mem} = 100 nsec
- T_{TLB} = 20 nsec
- Hit ratio is 80%
- $T_{effective}$ = ?

$T_{effective}$ = 0.8 (20 + 100) + 0.2 (20 + 2x100)
= 140 nanoseconds

Example

- T_{mem} = 100 nsec
- T_{TLB} = 20 nsec
- Hit ratio is 98%
- $T_{effective}$ = ?

$T_{effective}$ = 0.98 (20 + 100) + 0.02 (20 + 2x100)
= 122 nanoseconds

Structure of the Page Table

- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    - That amount of memory used to cost a lot
    - Don’t want to allocate that contiguously in main memory
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical/ Multi-level Page Tables

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

WHY WE NEED Multilevel paging:

The need for multilevel paging arises when-

- The size of page table is greater than the frame size.
- As a result, the page table cannot be stored in a single frame in main memory.
- We then page the page table

Multilevel Paging

Frame Size = 2MB , Page Table Size = 8MB, Main Memory = 64MB

Page Table 8MB

0	f0
1	f1
2	f2
3	f3
4	f4
5	f5
6	f6
7	f7

✗ Can't Fit In single Frame

Page Table(0)

0	f8	0	f0
		1	f1
1	f9	2	f2
		3	f3
2	f10	4	f4
		5	f5
3	f11	6	f6
		7	f7

✓ Fit In 4 Frame

Page Table(1)

0	f0	0	f8	0 f0
				1 f1
		1	f9	2 f2
				3 f3
1	f1	2	f10	4 f4
				5 f5
		3	f11	6 f6
				7 f7

✓ Fit In 2 Frame

Hence

Page Table (0) 8MB

0	f8
1	f9
2	f10
3	f11

Page Table (1) 4MB

0	f12
1	f13

Page Table (2) 2MB

0	F14

	Main Memory
	OS
f0	Process1 Page0
f1	Process1 Page1
f2	Process1 Page2
f3	Process1 Page3
f4	Process1 Page4
f5	Process1 Page5
f6	Process1 Page6
f7	Process1 Page7
f8	Page Table(0) Page0
f9	Page Table(0) Page1
f10	Page Table(0) Page2
f11	Page Table(0) Page3
f12	Page Table(1) Page0
f13	Page Table(1) Page1
f14	Page Table(2)
	. .
f31	

EXAMPLE: 1

Illustration of Multilevel Paging-

Consider a system using paging scheme where-
Logical Address Space = 4 GB
Physical Address Space = 16 TB
Page size = 4 KB

Now, let us find how many levels of page table will be required.

Number of Bits in Physical Address-

Size of main memory
= Physical Address Space
= 16 TB
= 2^{44} B
Thus, Number of bits in physical address = 44 bits

Number of Frames in Main Memory-

Number of frames in main memory
= Size of main memory / Frame size
= 16 TB / 4 KB
= 2^{32} frames
Thus, Number of bits in frame number = 32 b

Number of Bits in Page Offset-

We have,

Page size

= 4 KB

= 2^{12} B

Thus, Number of bits in page offset = 12 bits

Alternatively,

Number of bits in page offset

= Number of bits in physical address – Number of bits in frame number

= 44 bits – 32 bits

= 12 bits

So, Physical address is-

Number of Pages of Process-

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 2^{20} pages

Now, we can observe-

- The size of inner page table is greater than the frame size (4 KB).
- Thus, inner page table can not be stored in a single frame.
- So, inner page table has to be divided into pages.

Inner Page Table Size-

Inner page table keeps track of the frames storing the pages of process.

Inner Page table size

= Number of entries in inner page table x Page table entry size

= Number of pages the process is divided x Number of bits in frame number

= 2^{20} x 32 bits

= 2^{20} x 4 bytes

= 4 MB

Number of Pages of Inner Page Table-

Number of pages the inner page table is divided

= Inner page table size / Page size

= 4 MB / 4 KB

= 2^{10} pages

Now, these 2^{10} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table-

Number of page table entries in one page of inner page table

= Page size / Page table entry size

= Page size / Number of bits in frame number

= 4 KB / 32 bits

= 4 KB / 4 B

= 2^{10}

Number of Bits Required to Search an Entry in One Page of Inner Page Table-

One page of inner page table contains 2^{10} entries.

Thus,

Number of bits required to search a particular entry in one page of inner page table = 10 bits

Outer Page Table Size-

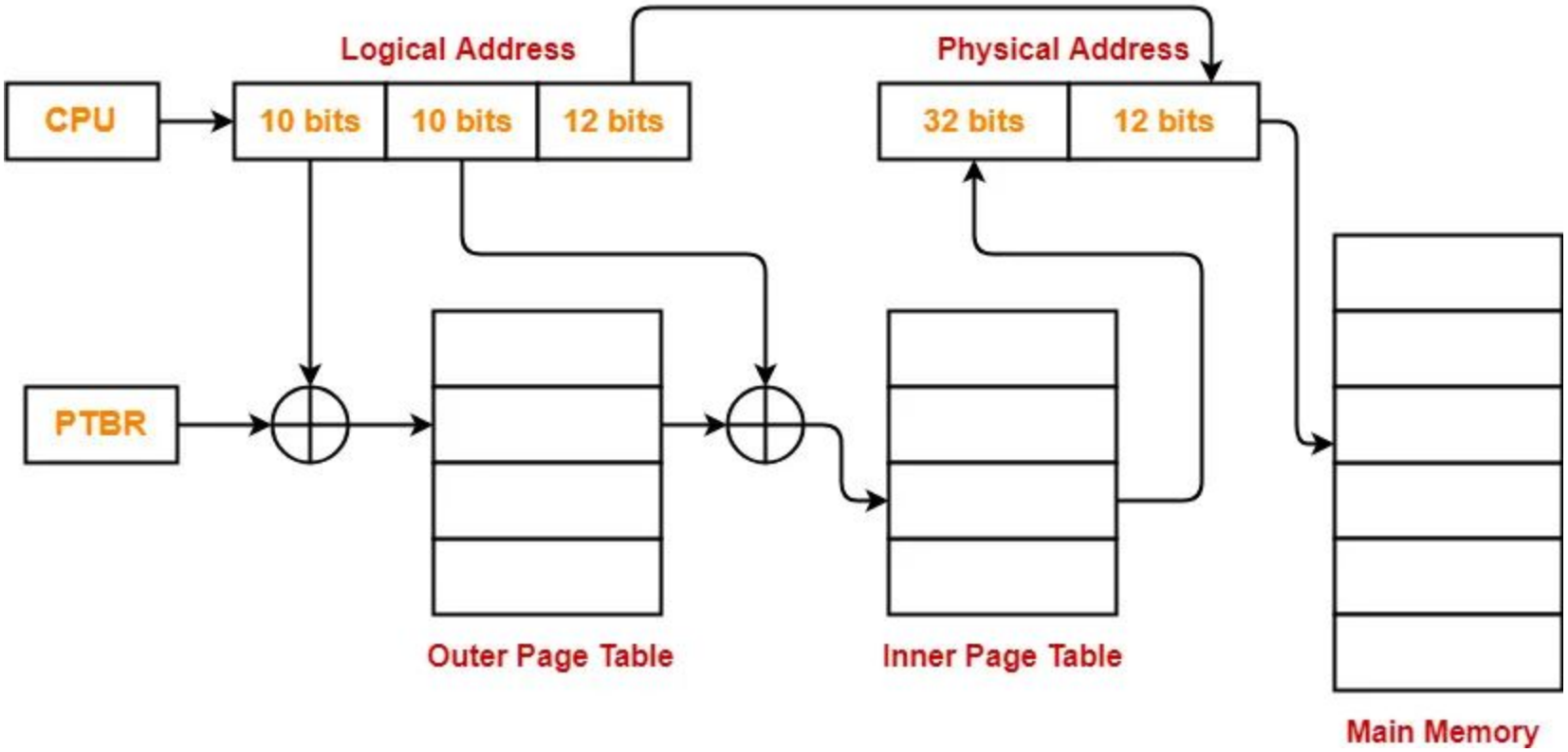
Outer page table is required to keep track of the frames storing the pages of inner page table.
Outer Page table size
= Number of entries in outer page table x Page table entry size
= Number of pages the inner page table is divided x Number of bits in frame number
= 2^{10} x 32 bits
= 2^{10} x 4 bytes
= 4 KB

Now, we can observe-
- The size of outer page table is same as frame size (4 KB).
- Thus, outer page table can be stored in a single frame.
- So, for given system, we will have two levels of page table.
- Page Table Base Register (PTBR) will store the base address of the outer page table.

Number of Bits Required to Search an Entry in Outer Page Table-

Outer page table contains 2^{10} entries.
Thus,
Number of bits required to search a particular entry in outer page table = 10 bits

The paging system will look like as shown below:-



EXAMPLE: 2

Consider a system using multilevel paging scheme. The page size is 16 KB. The memory is byte addressable and virtual address is 48 bits long. The page table entry size is 4 bytes.

Find-

1. How many levels of page table will be required?
2. Give the divided physical address and virtual address.

Solution-

Given-
- Virtual Address = 48 bits
- Page size = 16 KB
- Page table entry size = 4 bytes

Number of Bits in Frame Number-

We have,
Page table entry size
= 4 bytes
= 32 bits
Thus, Number of bits in frame number = 32 bits

Number of Frames in Main Memory-

We have, Number of bits in frame number = 32 bits
Thus,
Number of frames in main memory
= 2^{32} frames

Size of Main Memory-

Size of main memory
= Total number of frames x Frame size
= 2^{32} x 16 KB
= 2^{46} B
= 64 TB
Thus, Number of bits in physical address = 46 bits

Number of Bits in Page Offset-

We have,
Page size
= 16 KB
= 2^{14} B
Thus, Number of bits in page offset = 14 bits

Alternatively,
Number of bits in page offset
= Number of bits in physical address – Number of bits in frame number
= 46 bits – 32 bits
= 14 bits

Process Size-

Number of bits in virtual address = 48 bits
Thus,
Process size
= 2^{48} bytes
= 256 TB

Number of Pages of Process-

Number of pages the process is divided
= Process size / Page size
= 256 TB / 16 KB
= 2^{48} B / 2^{14} B
= 2^{34} pages

Inner Page Table Size-

Inner page table keeps track of the frames storing the pages of process.
Inner Page table size
= Number of entries in inner page table x Page table entry size
= Number of pages the process is divided x Page table entry size
= 2^{34} x 4 bytes
= 2^{36} bytes
= 64 GB

Now, we can observe-

- The size of inner page table is greater than the frame size (4 KB).
  64GB> 4KB
- Thus, inner page table can not be stored in a single frame.
- So, inner page table has to be divided into pages.

Number of Pages of Inner Page Table-

Number of pages the inner page table is divided
= Inner page table size / Page size
= 64 GB / 16 KB
= 2^{36} B / 2^{14} B
= 2^{22} pages

Now, these 2^{22} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table-

Number of page table entries in one page of inner page table
= Page size / Page table entry size
= 16 KB / 4 B
= 2^{12} entries

Number of Bits Required to Search an Entry in One Page of Inner Page Table-

One page of inner page table contains 2^{12} entries.
Thus,
Number of bits required to search a particular entry in one page of inner page table = 12 bits

Outer Page Table-1 Size-

Outer page table-1 is required to keep track of the frames storing the pages of inner page table.

Outer Page table-1 size

= Number of entries in outer page table-1 x Page table entry size

= Number of pages the inner page table is divided x Page table entry size

= 2^{22} x 4 bytes

= 16 MB

Now, we can observe-

- The size of outer page table-1 is greater than the frame size (4 KB).
  16MB >4KB
- Thus, outer page table-1 can not be stored in a single frame.
- So, outer page table-1 has to be divided into pages.

Number of Pages of Outer Page Table-1

Number of pages the outer page table-1 is divided
= Outer page table-1 size / Page size
= 16 MB / 16 KB
= 2^{10} pages
Now, these 2^{10} pages of outer page table-1 are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Outer Page Table-1

Number of page table entries in one page of outer page table-1
= Page size / Page table entry size
= 16 KB / 4 B
= 2^{12} entries

Number of Bits Required to Search an Entry in One Page of Outer Page Table-1

One page of outer page table-1 contains 2^{12} entries.
Thus,
Number of bits required to search a particular entry in one page of outer page table-1 = 12 bits

Outer Page Table-2 Size-

Outer page table-2 is required to keep track of the frames storing the pages of outer page table-1.
Outer Page table-2 size
= Number of entries in outer page table-2 x Page table entry size
= Number of pages the outer page table-1 is divided x Page table entry size
= 2^{10} x 4 bytes
= 4 KB

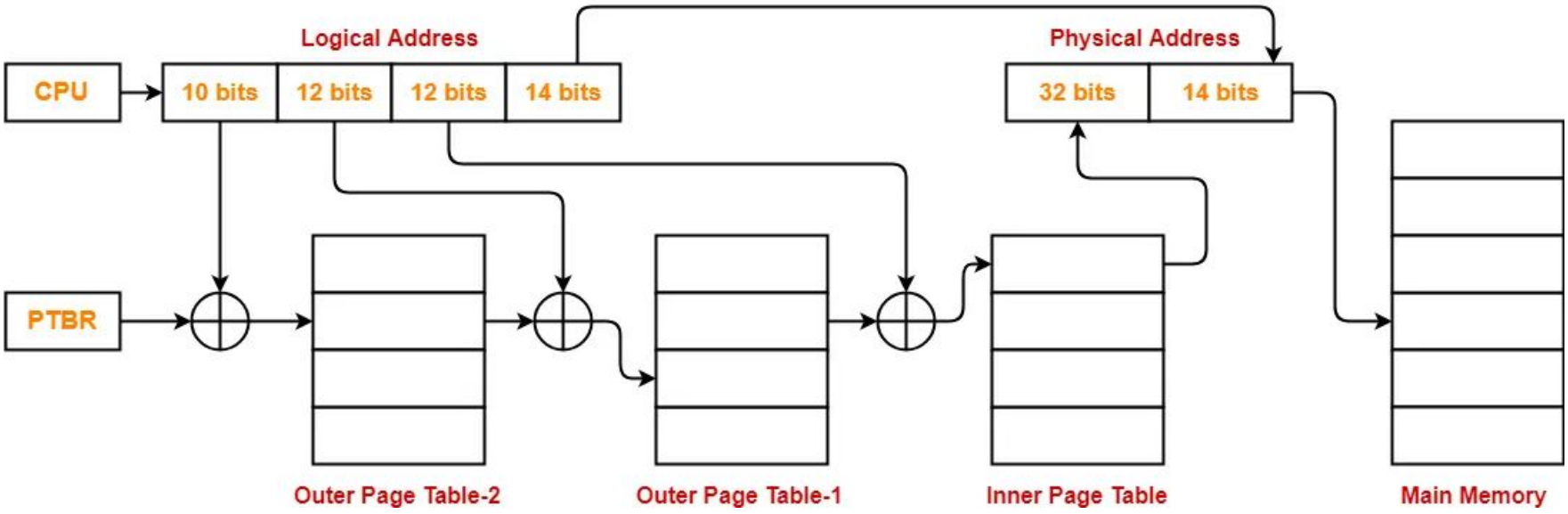
Now, we can observe-

- The size of outer page table-2 is less than the frame size (16 KB).
  4KB< 16KB
- Thus, outer page table-2 can be stored in a single frame.
- In fact, outer page table-2 will not completely occupy one frame and some space will remain vacant.
- So, for given system, we will have three levels of page table.
- Page Table Base Register (PTBR) will store the base address of the outer page table-2.

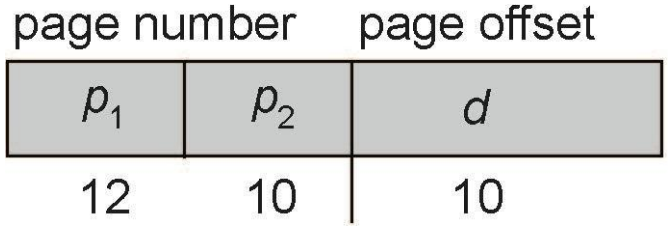
Number of Bits Required to Search an Entry in Outer Page Table-2

Outer page table-2 contains 2^{10} entries.
Thus,
Number of bits required to search a particular entry in outer page table-2 = 10 bits

The paging system will look like as shown below-

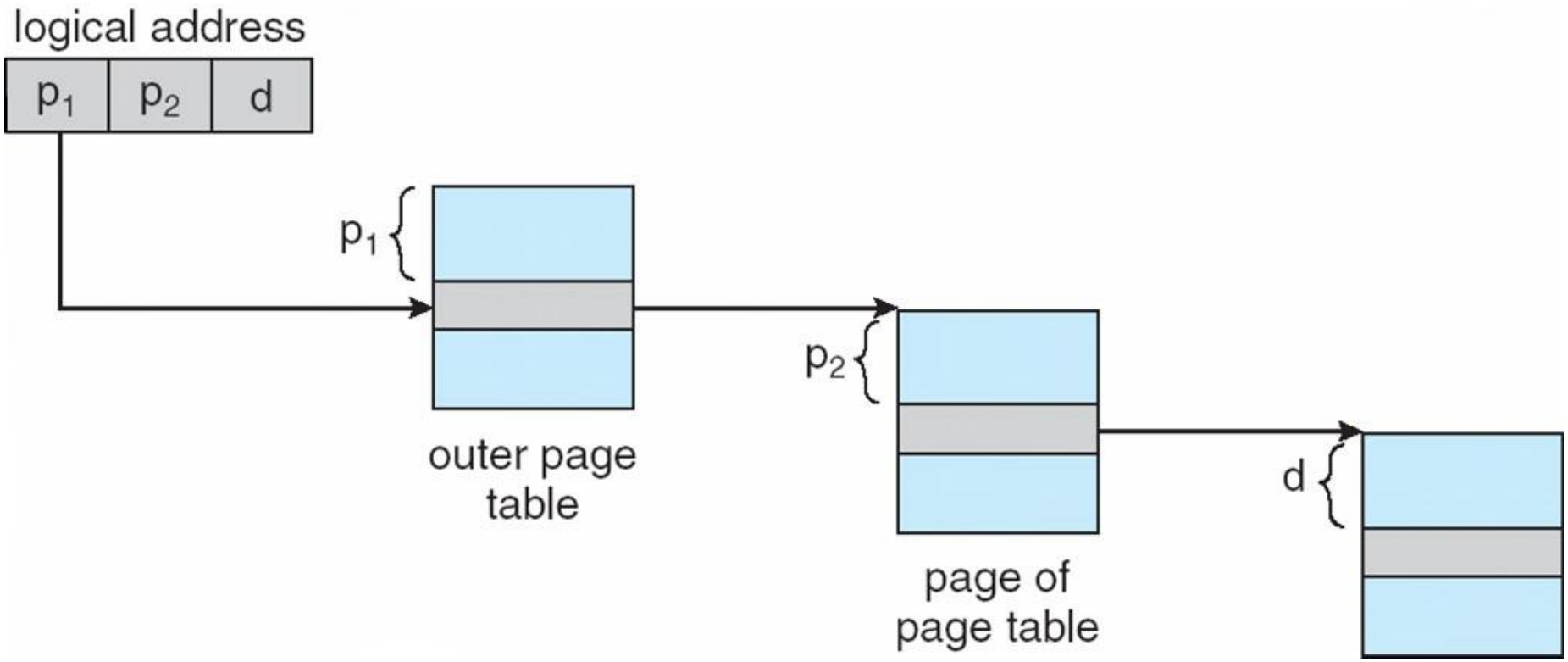


Two-Level Paging Example

- 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 as follows:
- 

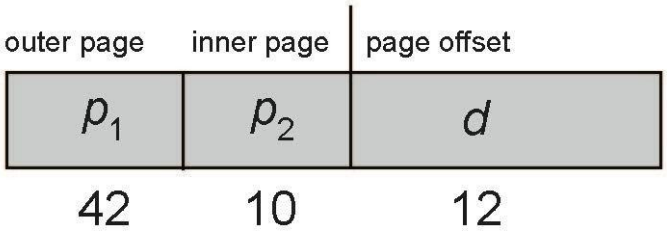
- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

Address-Translation Scheme



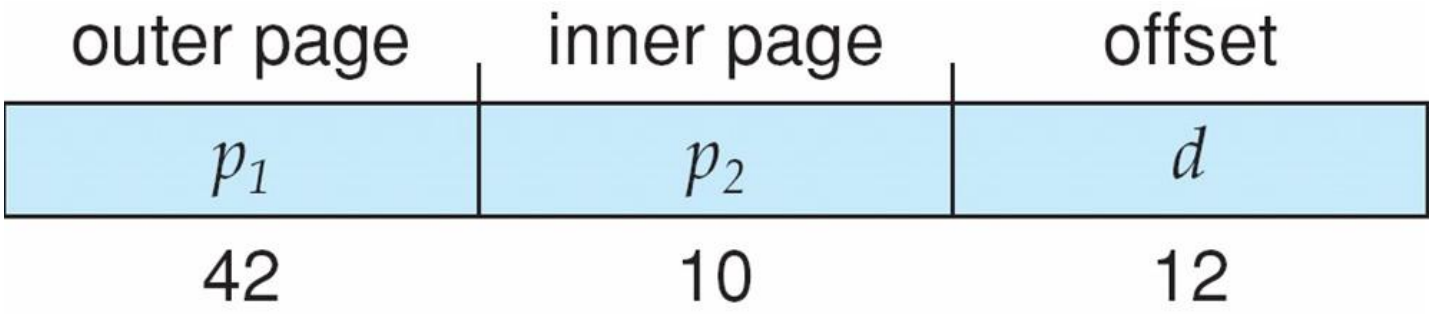
64-bit Logical Address Space

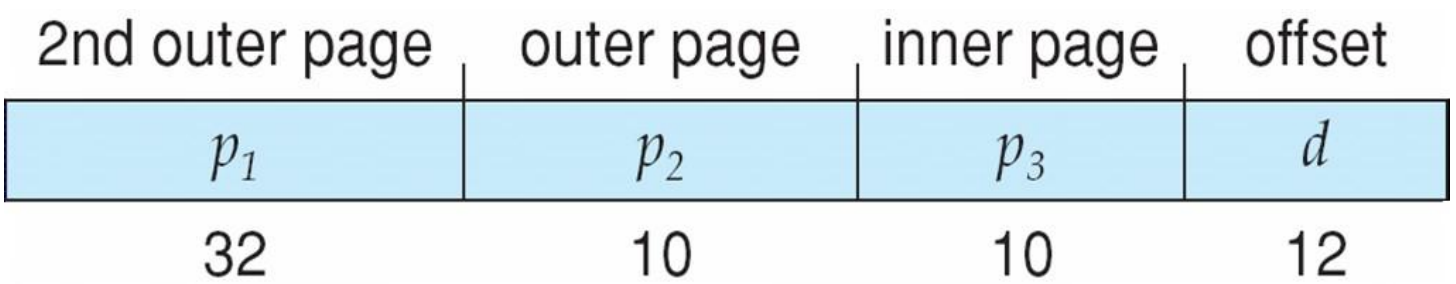
- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like



  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

Three-level Paging Scheme

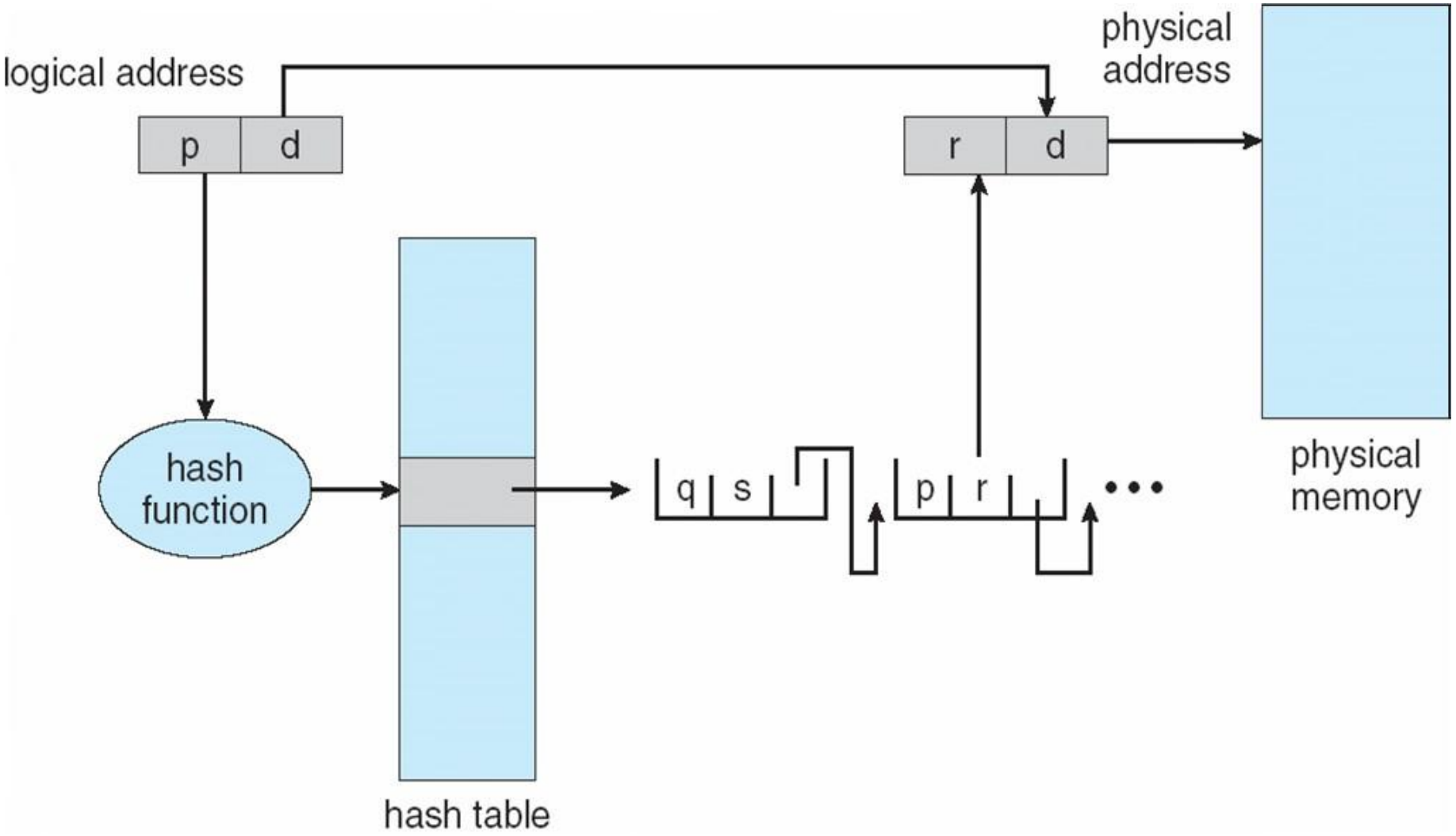




2- Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains
  - (1) the virtual page number
  - (2) the value of the mapped page frame
  - (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables**
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory

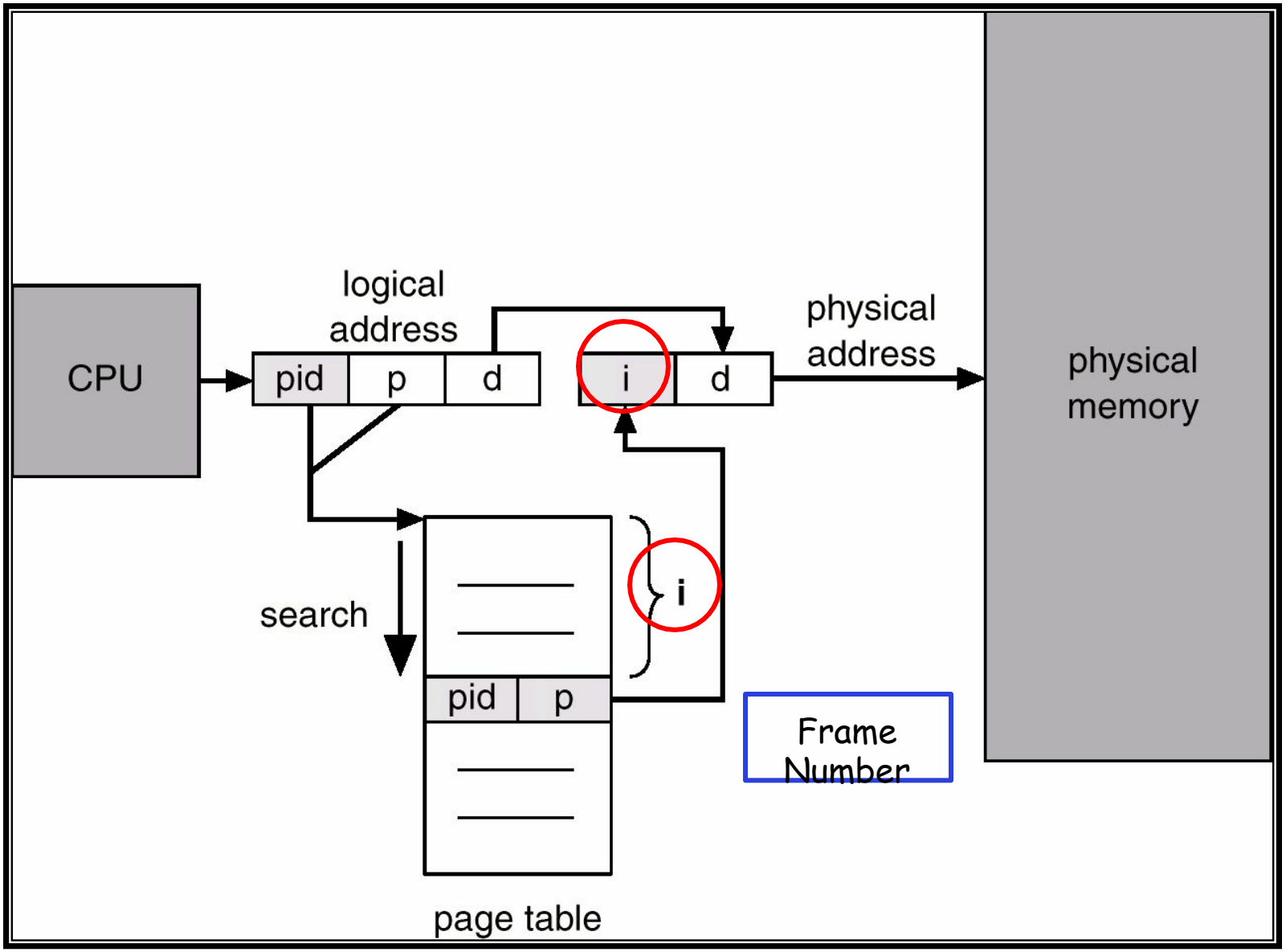
Hashed Page Table



3- Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages,
  - track all physical pages
- One entry for each real page of memory
- Entry consists of
  - the virtual address of the page stored in that real memory location,
  - information about the process that owns that page
- Decreases memory needed to store each page table
  - but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one/few page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

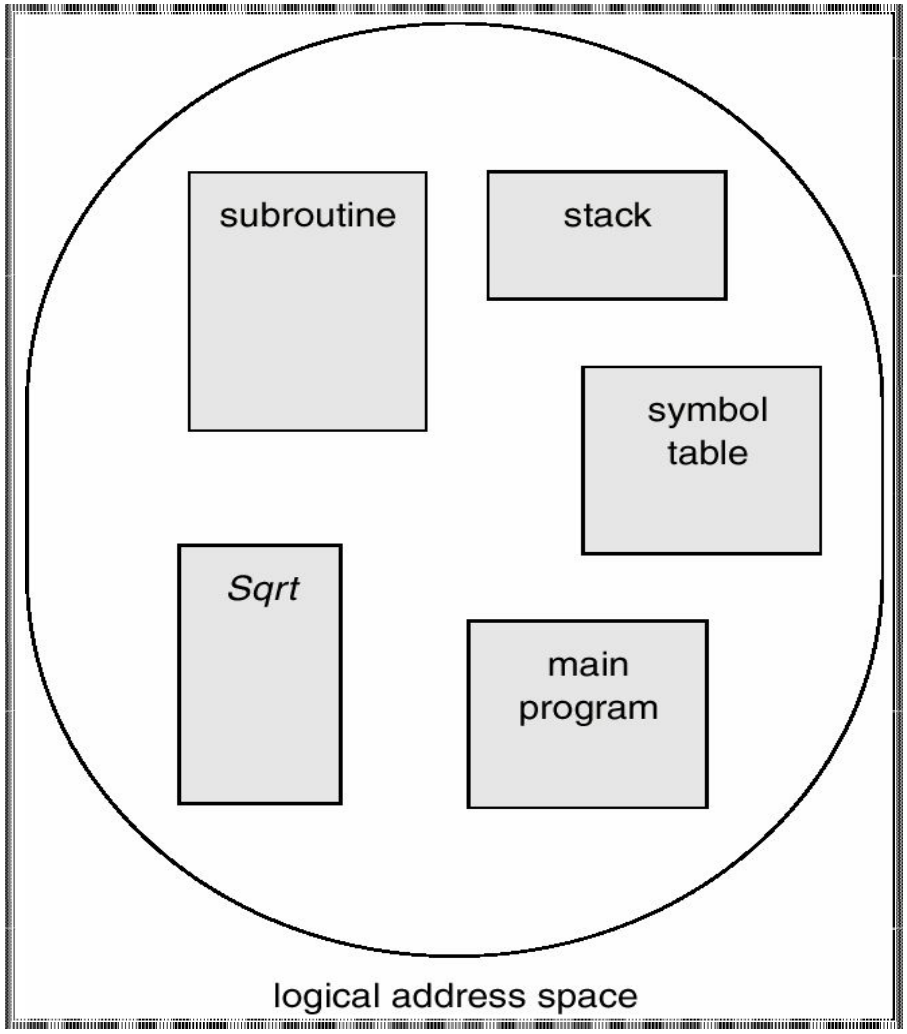
Inverted Page Table Architecture



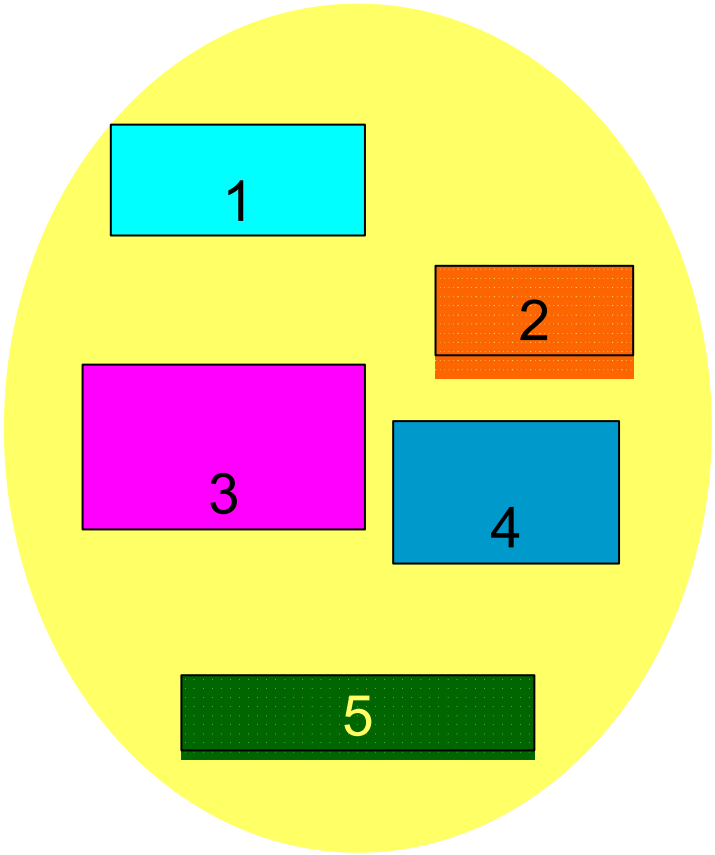
Segmentation

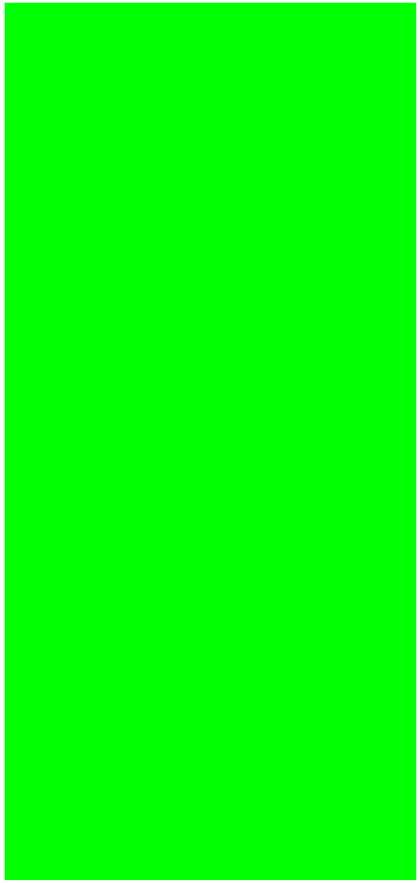
- A memory management scheme that supports programmer's view of memory.
- A segment is a logical unit such as: main program, procedure, function, method, object, global variables, stack, symbol table
- A program is a collection of segments

Segmentation

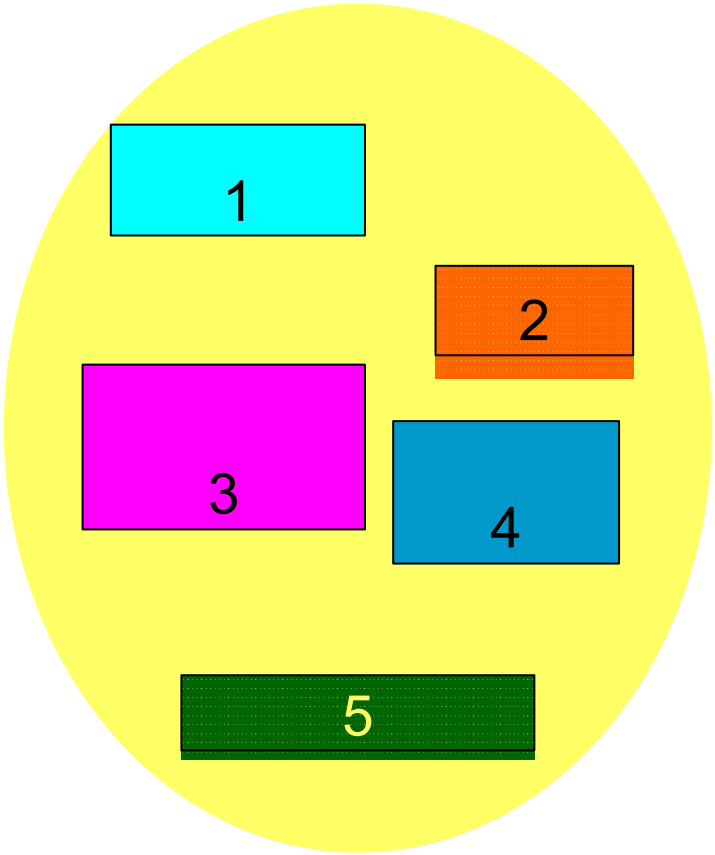


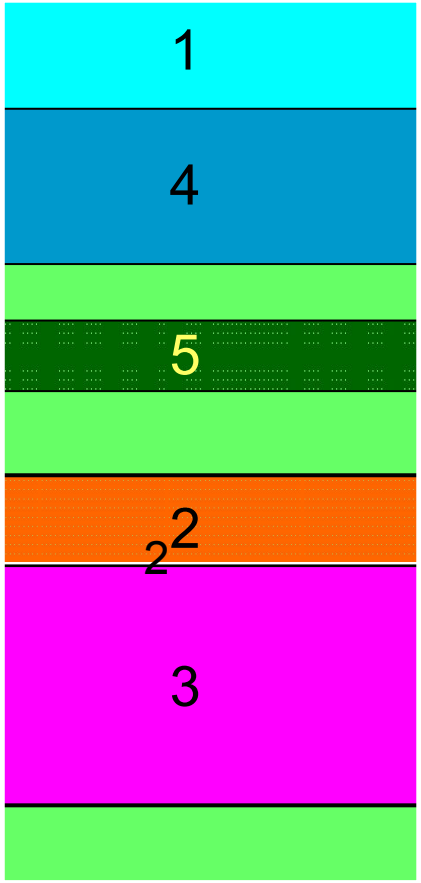
Segmentation



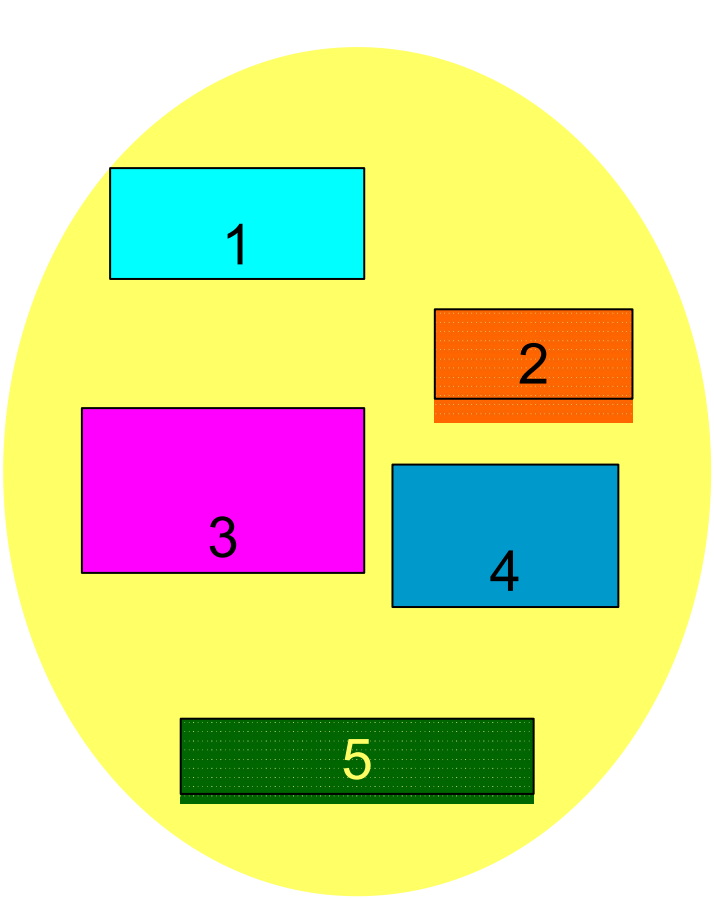


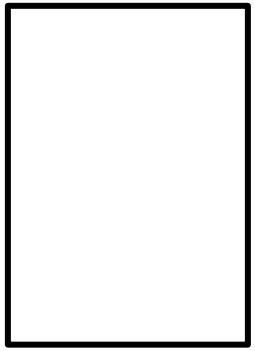
Segmentation



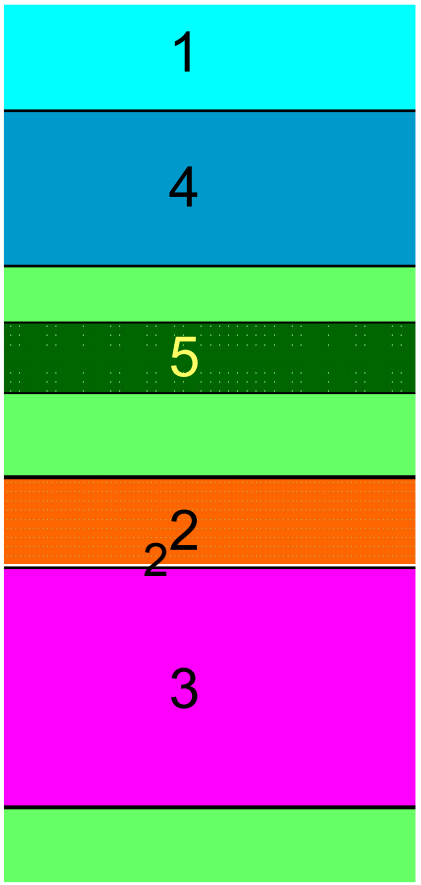


Segmentation





segment table



Segmentation

- Logical address **consists of a two tuple:**

  <segment-number, offset>

- Segment table – **maps two-dimensional logical addresses to physical addresses**

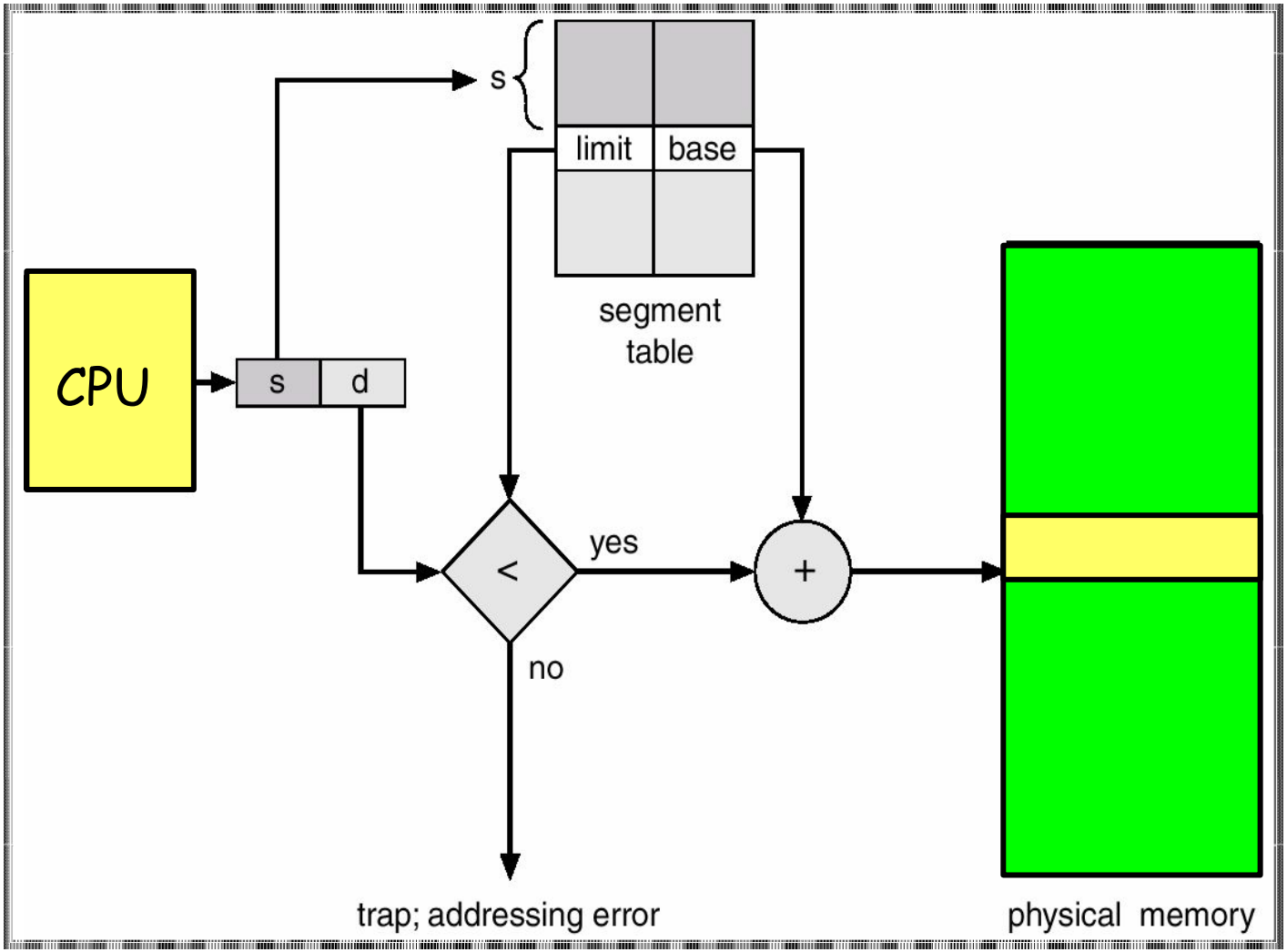
Segmentation

- **Each segment table entry has:**
  - **base – contains the starting physical address where the segments reside in memory.**
  - **limit – specifies the length of the segment.**

Segmentation

- Segment-table base register (STBR) points to the segment table's location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program
- Segment number s is legal if s < STLR

Segmentation

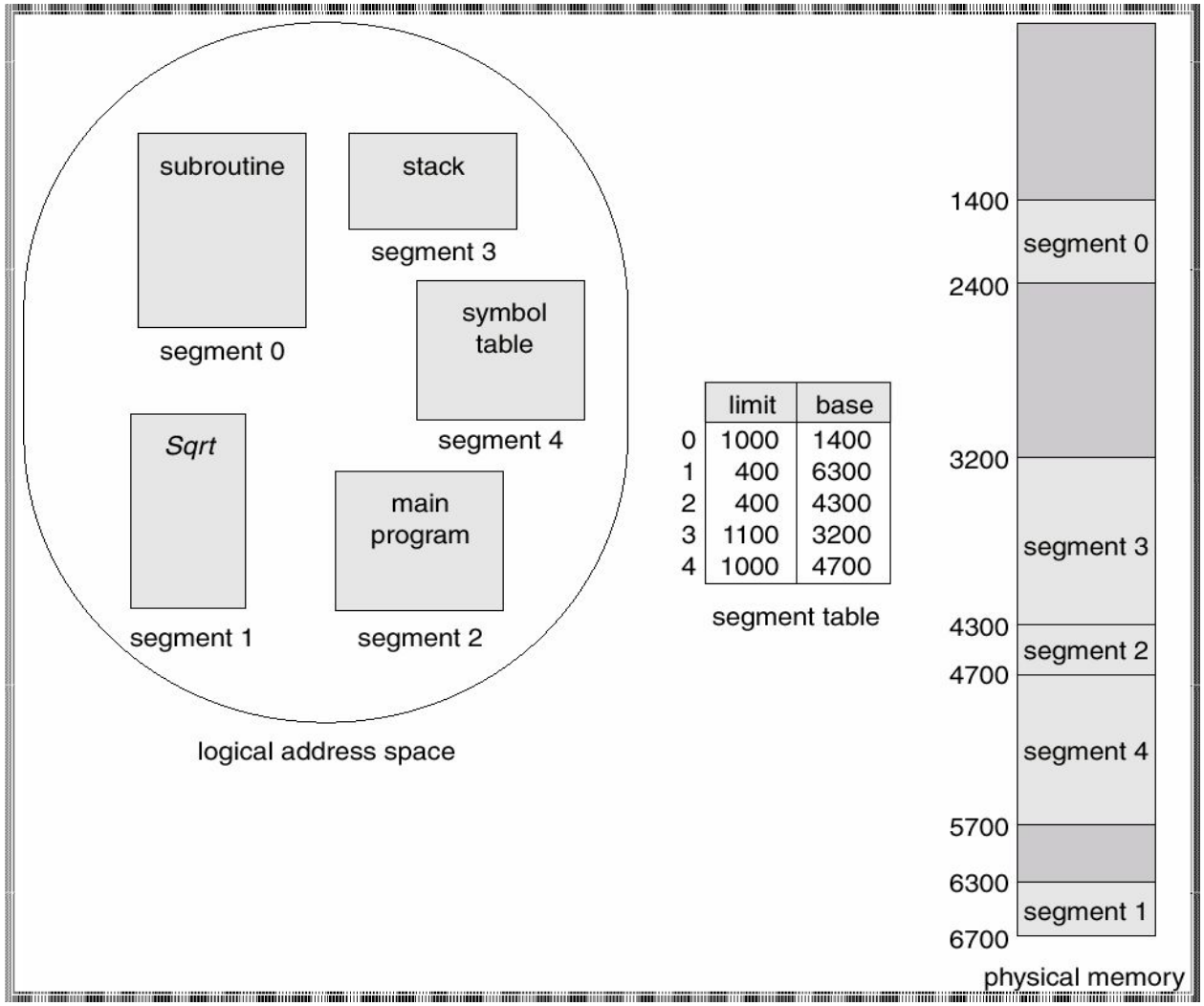


Segmentation Architecture

- Dynamic Storage Allocation
  - **First fit**
  - **Best fit**
  - **Worst fit**

  - External fragmentation

Example


subroutine
stack
segment 3
segment 0
symbol table
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit
base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

Address Translation

- **Logical and Physical Addresses**
  - (2, 399) – PA: 4300+399 = 4699
  - (4, 0) – PA: 4700+0 = 4700
  - (4, 1000) ⇒ trap
  - (3, 1300) ⇒ trap
  - (6, 297) ⇒ trap

Issues with Segmentation

- **Reduce external fragmentation by compaction**
  - **Shuffle segments to place free memory together in one block.**
  - **Compaction is possible *only* if relocation is dynamic, and is done at execution time.**

Issues with Segmentation

- I/O problem
  - **Latch job in memory while it is involved in I/O.**
  - **Do I/O only into OS buffers**
- **Very large segments ⇒ page program segments—**
  paged segmentation