Operating Systems: Main Memory – Part II

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 9)

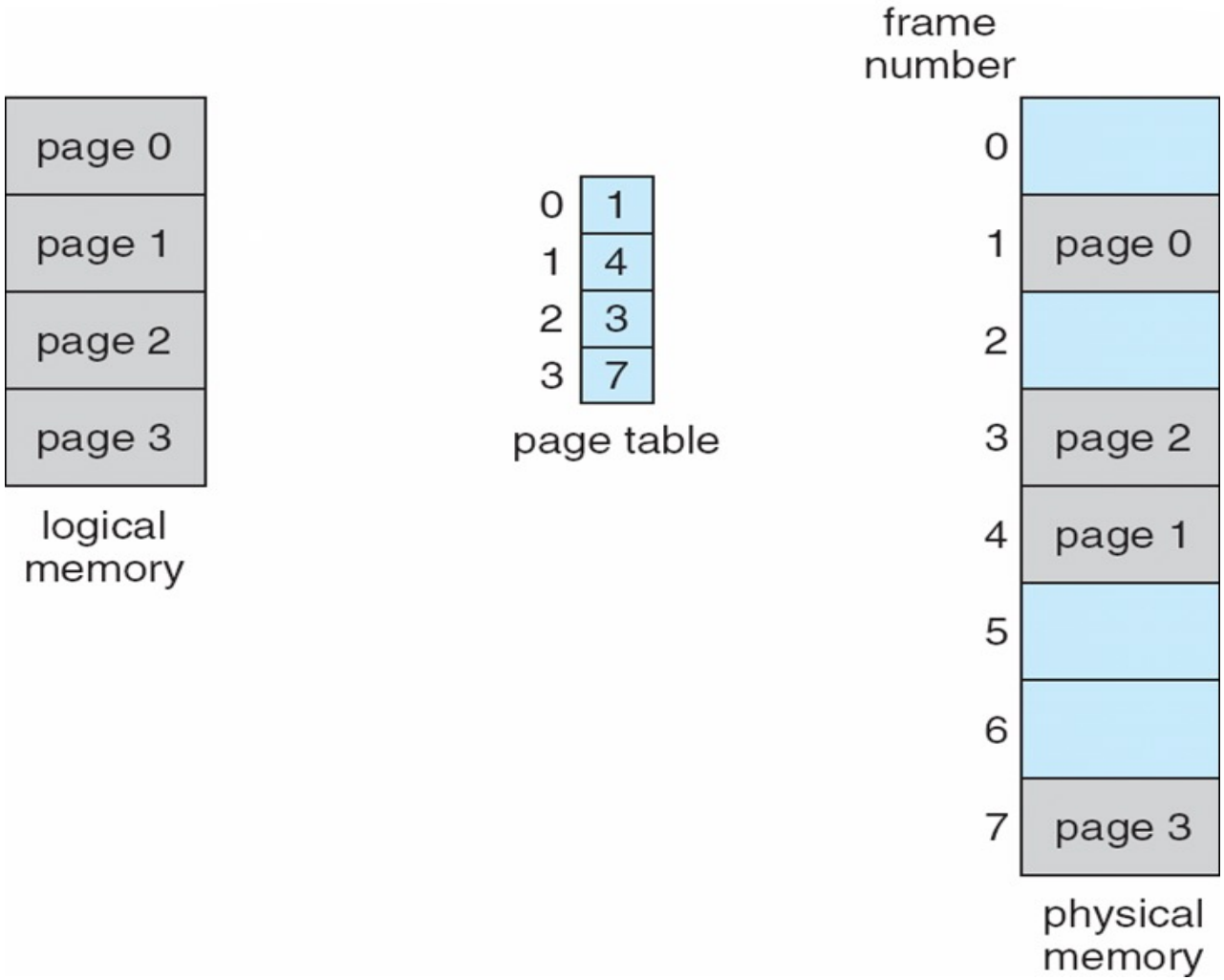
Paging

- **Paging** - a memory-management scheme that permits a process's physical address space to be **non-contiguous**.
- Logical address space of a process is divided into fixed blocks called **pages**
- Physical address; that is main memory is divided into fixed-sized blocks called **frames**.
- **Page size = Frame size**

Paging

- Pages of a process are scattered in main memory and stored in frames.
- Each executing process has its own page table.
- **Page table** – keeps track of the pages residing in main memory
  - Indexed by page numbers
  - Stores the frame number in which the page loaded.

Paging Model of Logical and Physical Memory



Paging – Some facts

- Page size are generally 4 KB or 8 KB (may range from 4KB – 1GB).
  - Power of 2 as it makes address translation easy.
- The page size (like the frame size) is defined by the hardware

Advantages:

- Simple to implement.
- Avoids external fragmentation but suffers from internal fragmentation.

Logical address and paging

- Logical address space = 2^4 = 16 bytes.
  - What is the range of addresses in this logical space?
  - How many bits do you need to represent a logical address in this space?

Logical address and paging

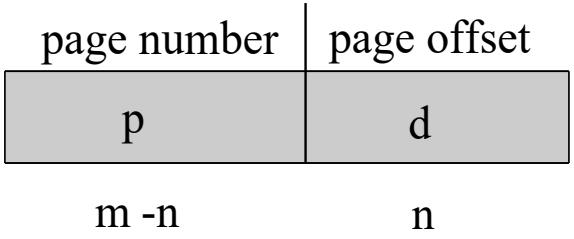
Logical address space = 2^4 = 16 bytes.

- Range of addresses in this logical space = 0..15(2^4-1)
- Bits needed to represent an address in this space = 4
- If the page size is be 4 bytes, then the logical address space is divided into 4 pages.

Logical addresses	Logical address in binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Paging - Address Translation

- Logical address now consists of of two parts:



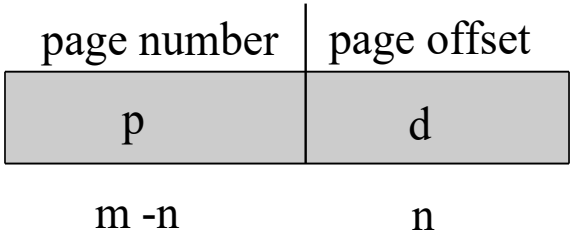
  - **Page number** (***p***) – used as an index into a **page table**
  - **Page offset** (***d***) – is the offset/displacement within the page.

- Page offset - ranges from 0 to page size -1
- Page size = frame size => page offset = frame offset.
- **Physical memory address = frame number + page offset.**

Address Translation Scheme - Generalized

For given logical address space size = 2^m, page size = 2^n and physical address space = 2^r

- Number of bits to represent logical address =
- Number of bits to represent physical address =
- Number of bits to represent offset =
- Number of pages =
- Number of bits to represent a page number =
- Number of frames =
- Number of bits to represent a frame number =



Paging and Address Translation Example

Logical address space = 2^4 = 16 bytes

- Number of bits to represent logical address =
- *Page size = 2^2 = 4 bytes*
- Number of bits to represent offset =
- Number of bits to represent a page number =
- *Total # of pages =*

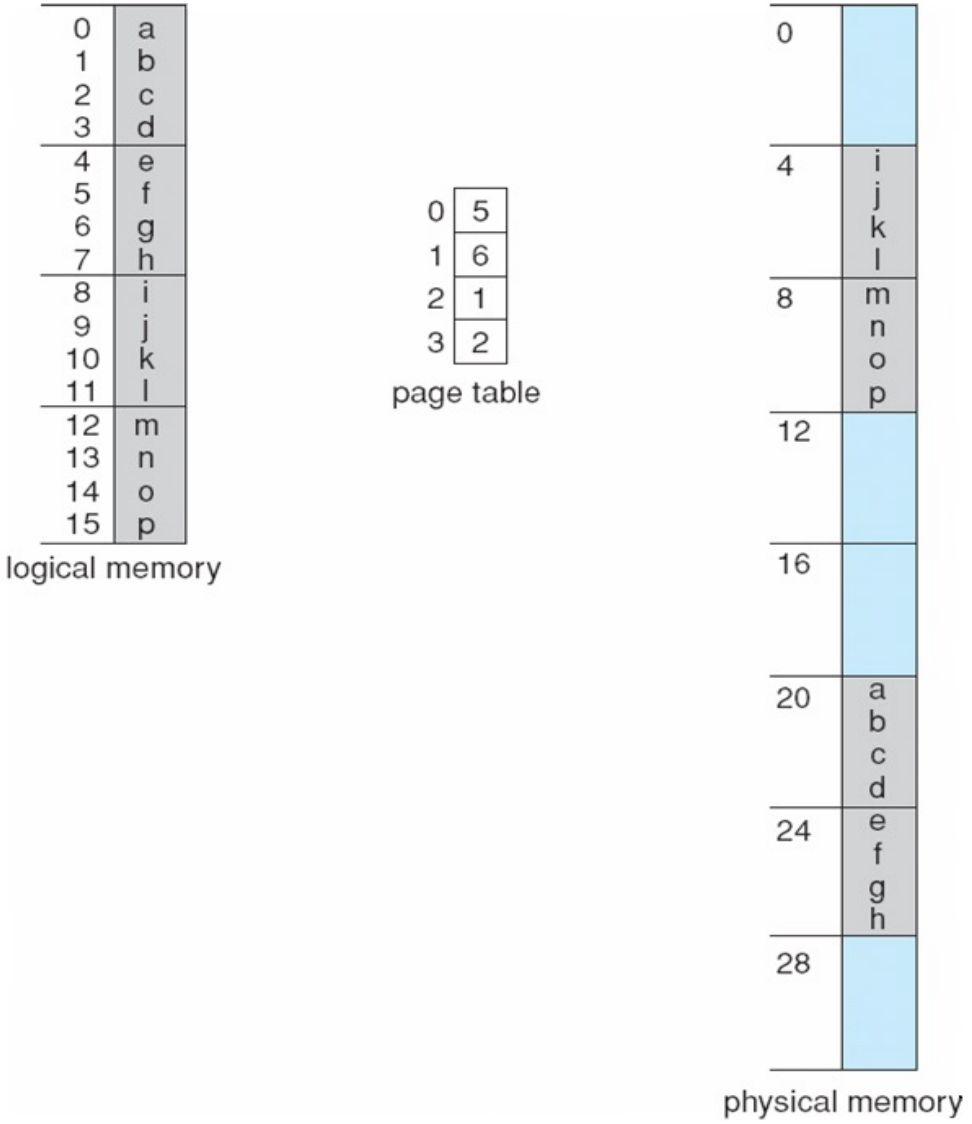
Memory/Physical address size = 2^5 = 32 bytes

- Number of bits to represent physical address =
- *Total # of frames =*
- Number of bits to represent a frame number =

Logical addresses	P number	d (offset)
0	00	00
1	00	01
2	00	10
3	00	11
4	01	00
5	01	01
6	01	10
7	01	11
8	10	00
9	10	01
10	10	10
11	10	11
12	11	00
13	11	01
14	11	10
15	11	11

Paging and Address Translation Whole Picture!

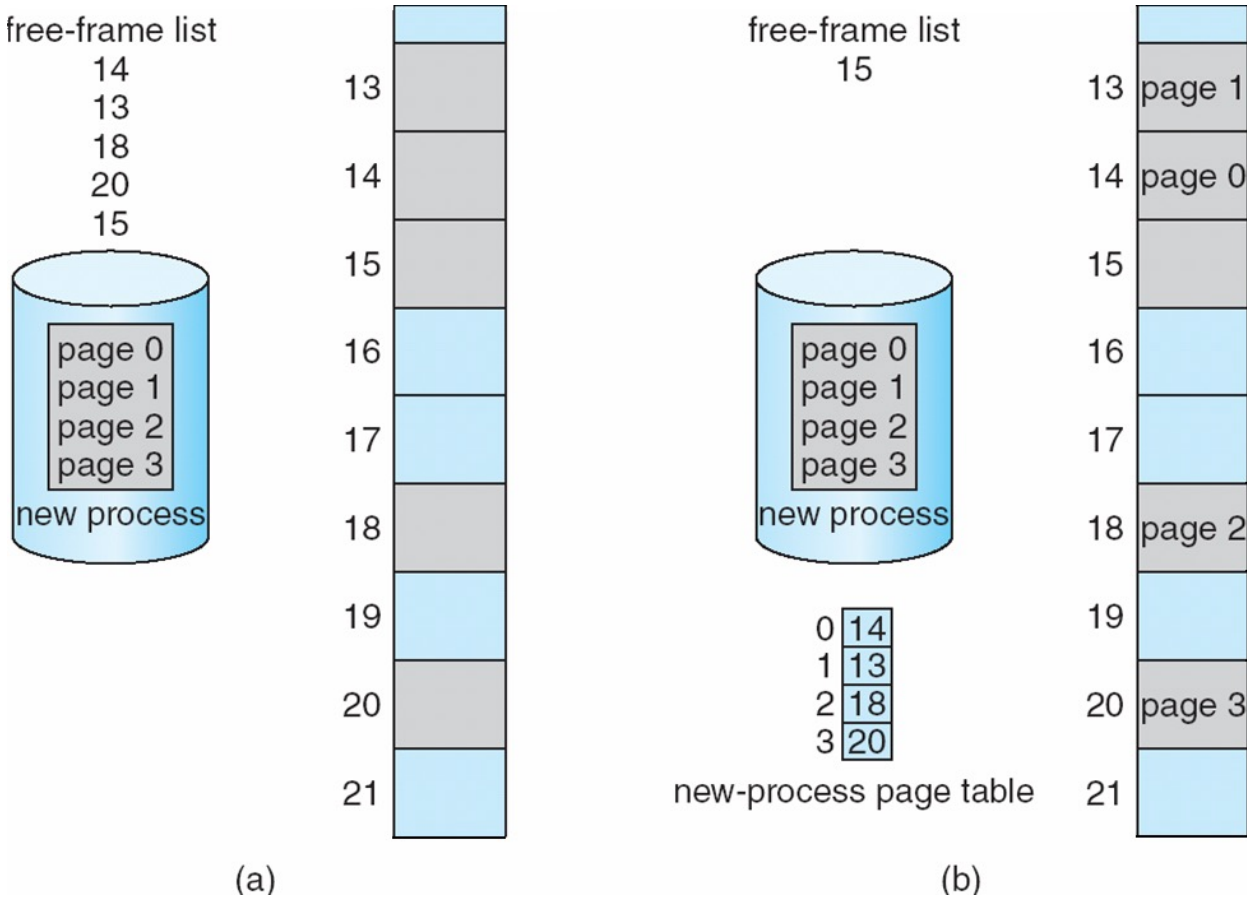
- CPU generates Logical address 3 [0011] – page# - 00 and offset = 11
- Page 0 resides in the frame# 5 [101]
- Logical address 3 maps to physical address = 10111(5*4 + 3) = 20 + 3 = 23.



Frames in Memory

- The operating system manages physical memory.
  - Needs to know which frames are allocated, which frames are available, how many total frames there are, etc.
  - To answer the above questions OS maintains a data structure called a **frame table (free-frame list)**.

Allocating Frames



Before allocation

After allocation

Implementation of Page Table

- For small size page tables – provide dedicated registers.
- For large page tables – **page table kept in main memory**
  - **Page-table base register** (**PTBR**) points to the starting address of the page table in main memory.
  - In this scheme every data/instruction access requires two memory accesses
    - One for the page table and one for the data / instruction!

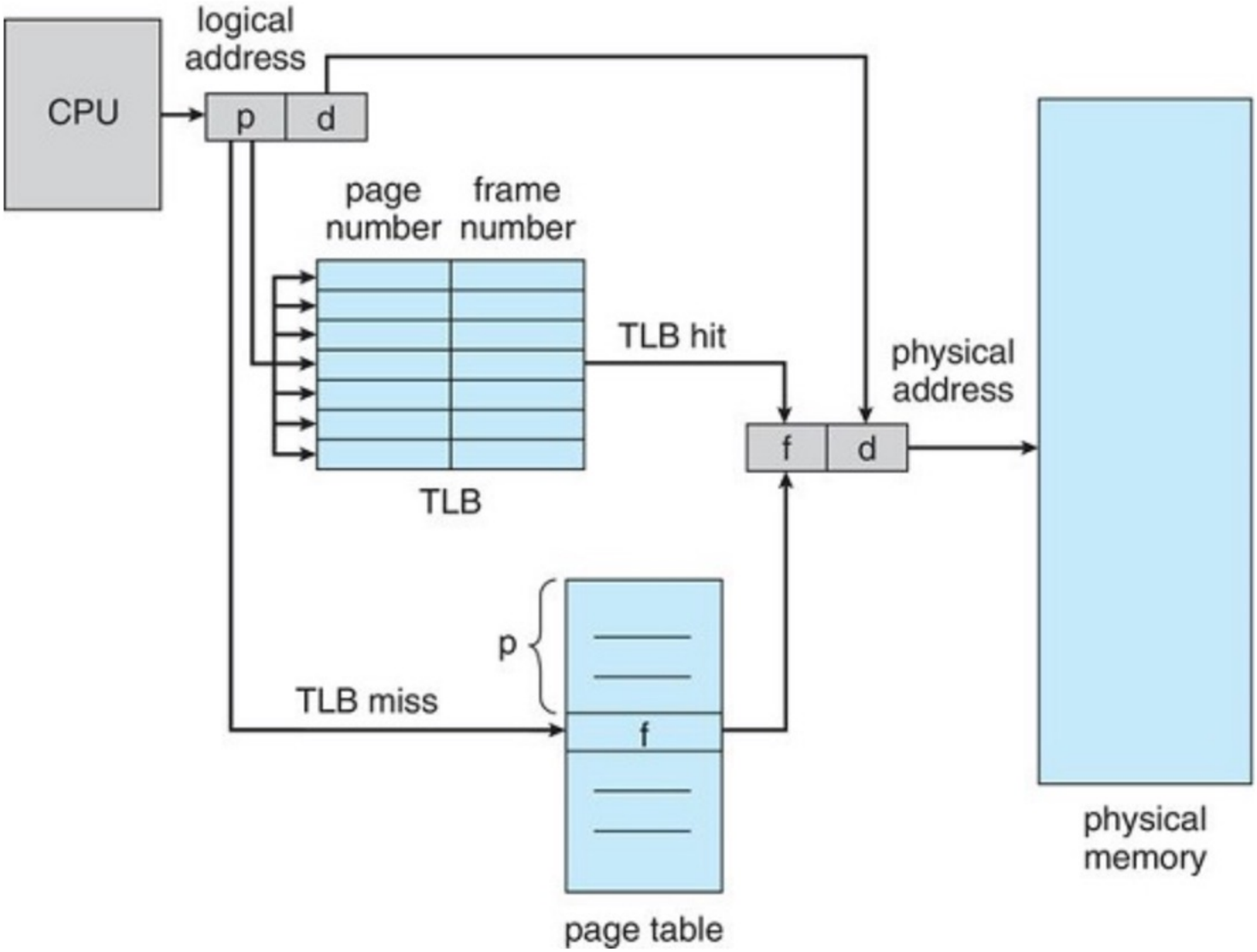
Implementation of Page Table

- To mitigate the two-memory access problem current machines, use a special fast-lookup hardware cache called **translation look-aside buffers** (**TLBs**)
  - TLB is associative, high-speed memory.
  - TLBs typically are small (64 to 1,024 entries)

Translation Look-aside Buffer (TLB)

- When a logical address is generated by the CPU, its page number is presented as a data item to TLB.
- This data is checked with all the entries in TLB **simultaneously**.
- If page is found (called a **TLB Hit**) its corresponding frame# is returned.
- If page is not found in TLB (called a **TLB Miss**) page table is searched for the page and its corresponding frame# is retuned and entry for the page added in TLB.
- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch

Paging Hardware With TLB



TLB - Effective Access Time

- **Hit ratio (α):** is the percentage of times a page number of interest is found in TLB.
- Memory access time = ***m***
- TLB access time = t
- EAT = $(\boldsymbol{m + t})\ \alpha + (2\boldsymbol{m + t})(1 - \alpha)$

TLB - Effective Access Time Example

- **Suppose memory access time (*m*)** = 100ns
- If we fail to find the page in TLB then accessing memory is 2m =200ns.
- Ignoring TLB access time **t**
  - If hit-ratio (α) = 80%
    - EAT = 0.80 x 100 + 0.20 x 200 = 120ns
  - Consider more realistic hit ratio -> α = 99%
    - EAT = 0.99 x 100 + 0.01 x 200 = 101ns!

Structure of the Page Table

- Consider a 32-bit logical address space as on modern computers, and page size of 4 KB (2^{12})

Page number | Page offset

20 bits	12 bits

- No of entries in a single level page table = 2^{20} entries
- If each page entry is 4 bytes:
  - Size of page table = $4*2^{20}$ = 4 MB!
  - Don't want to allocate that contiguously in main memory!

Structure of the Page Table Cont…

- Common techniques for structuring a page table:
  - **Hierarchical paging**
  - Hashed page table (used for logical address spaces > 2^{32})
  - Inverted page table

Multi- Level/Hierarchical Paging

- **Multi- Level/Hierarchical Paging** – is a technique in which the page table is paged; that is, broken down into multiple page tables of the same size as the page.
- Need an outer page table to track these pages!
- With varying sizes of the logical address space and pages, we could have a very large outer page table.
  - In this case, we repeat this process by paging the outer page table (thus introducing another layer of page table).

Multi- Level/Hierarchical Paging Example

- From the example previously shown the size of a single level page table = $4*2^{20}$ bytes.
- Number of pages of the page table =

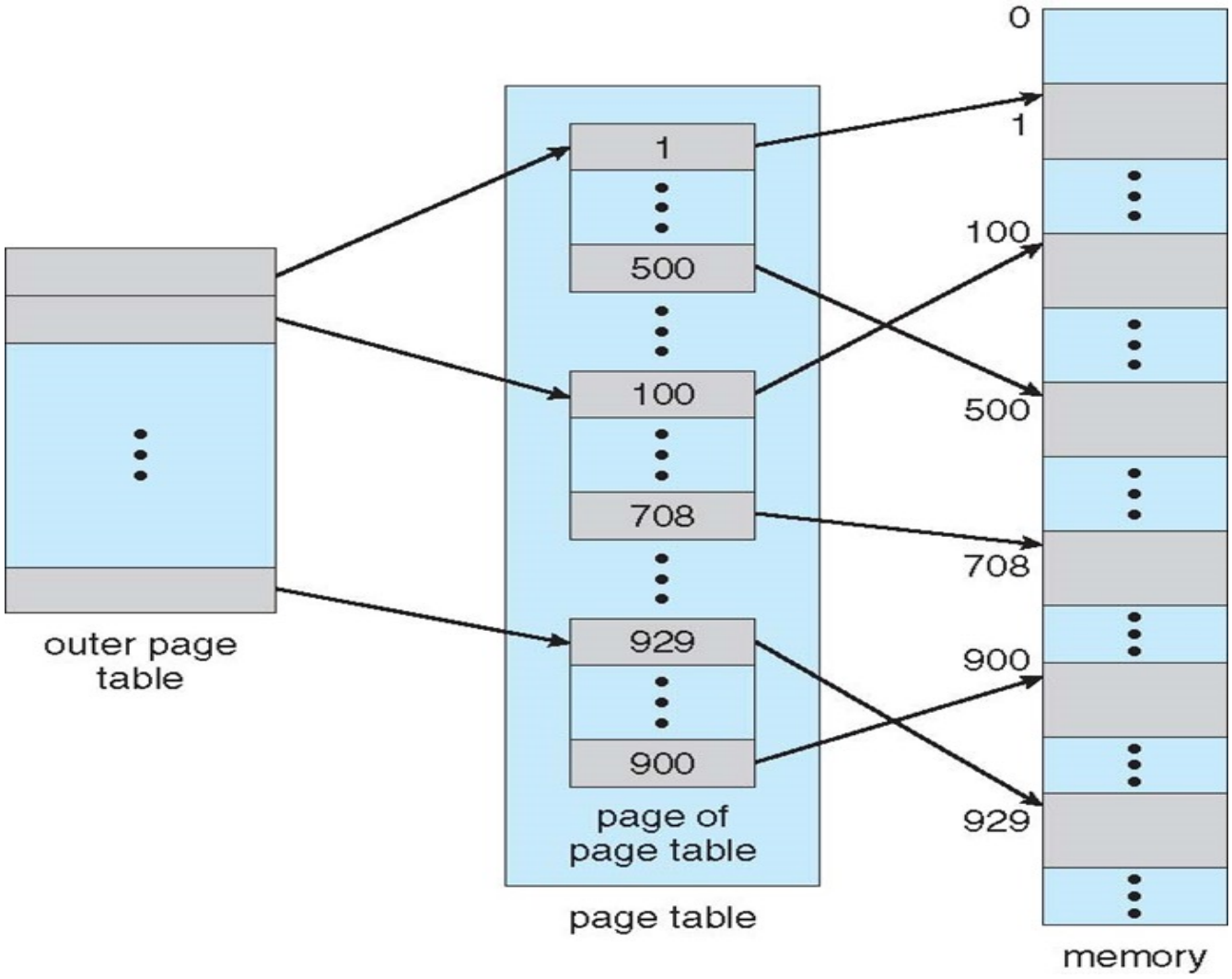
Size of the single level page table/size of a page

= $4*2^{20}/2^{12} = 2^{10}$

Multi- Level/Hierarchical Paging

- To track these pages of the page table we need a **second level page table** (or outer page table)!
- Size of this outer page table
  - = Number of pages of the page table * page entry size
  - = $2^{10}(2^2) = 2^{12}$ bytes = size of one page.
- Since there are two levels of page tables, this above technique is called **two-level paging**.

Two-Level Page-Table Scheme Illustration



Address Translation in two level paging scheme - Example

On a 32-bit machine with 4K page size (previous example)

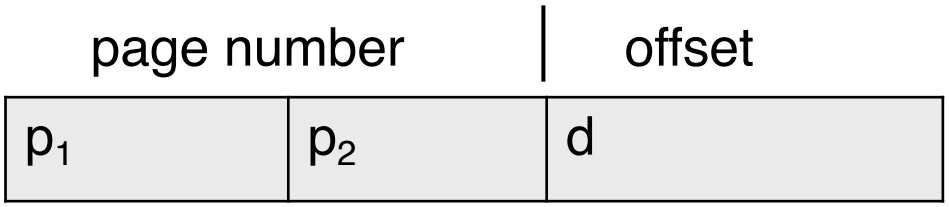
No. of page table entries in a page is (at most) = $2^{12}/4 = 2^{10}$.

The distribution of no. of bits to represent each level of the multi-level page table, and the page offset is as follows:

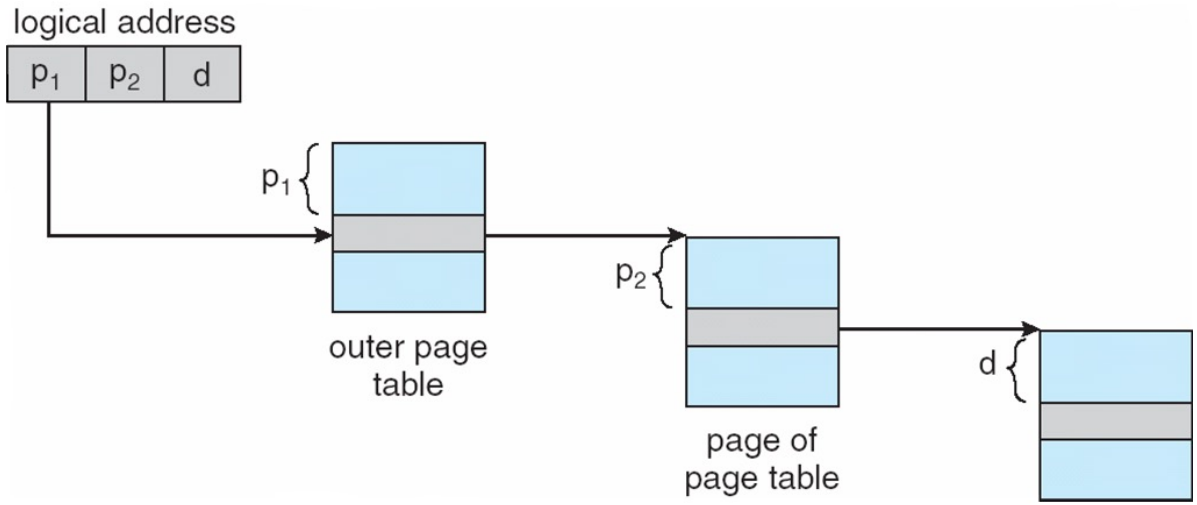
10	**10**	**12**

Address Translation in two level paging scheme - Generalized

- Since the page table is paged, the page number is further divided into:
  - p_1 is the displacement/offset in the outer page table, and
  - p_2 is the displacement/offset within the page of the inner page table
- Thus, a logical address is as follows:



Address Translation in two level paging scheme



- The starting address of the outer page table is stored in in PTBR.
- p_1 is the offset into this outer page table.
- We extract the address at this memory address, which is the starting address of the page of the page table.
- Then, p_2 is the offset into this inner page to read the frame number.
- Once we get the frame number, we use the offset d to compute the address of the actual instruction/data.

Hierarchical paging example

- Logical address space = 2^5 bytes
- Page size = 2^2 bytes
- Physical address space = 2^8 bytes
- Page table entry size = 8 bits = 1 bytes
- No. of pages to represent this logical space= 2^3
- Size of a single level page table = $2^3 * 1 = 2^3$
- No. of entries of a page of a page table = $2^2/1 = 2^2$
- No. of pages required to represent the page table = $2^3/2^2 = 2$.
- 1 outer page table needed to hold the addresses of these to pages.

Hierarchical paging example continued..

- Address Translation Scheme:
  - 1 bit to represent offset in the outer page table
  - 2 bit to represent offset in the inner page table
  - 2 bit to represent offset within the page.

1	2	2

Multi Level Hierarchical Paging

- Consider a system with logical address space 64-bit logical address space with page size 4KB (2^{12})
- If the page table entry size is 4 bytes, then number of page table entries in a page = $2^{12}/4 = 2^{10}$
- We need 10 bits to represent an offset in this page of a page table.
- Therefore, according to the two-level paging scheme the address translation scheme is as follows:

42	10	12

- The size of the outer page table = $2^{42} * 4 = 2^{44}$ bytes --- too big to store in a single outer page of the page table!

Multi Level Hierarchical Paging

- Two-level paging scheme is no longer appropriate!
- Three level page scheme; that is, page the outer page table?
  - Any page of the page table (inner or outer) can have only 2^{10} entries. Therefore, we will need 10 bits to represent this offset.

32	10	10	12

  - 2nd outer page also too big!
  - Keep paging the different levels of outer page table, till it can be contained in a single page.

2	10	10	10	10	10	12

Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all **physical pages (that is frames)**
- **Inverted page table** has one entry for each **frame** of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Example: Consider a computer system with a 32-bit logical address, 4-KB page size, and 24-bit physical address.

- A conventional, single-level page table contains 2^{20} entries.
- An inverted page table contains **2^{12} entries**!