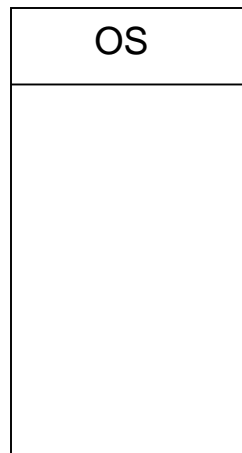


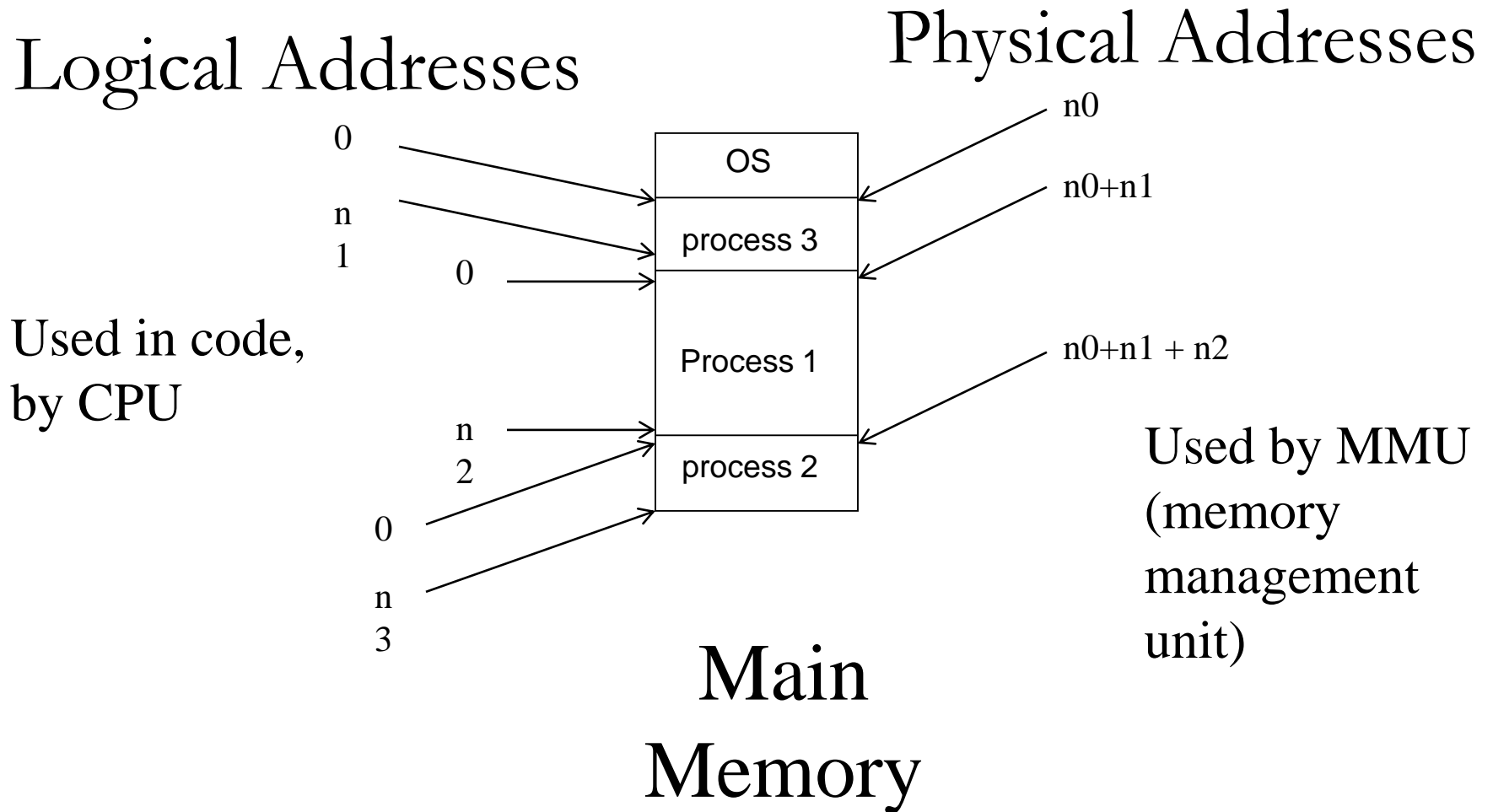
Main Memory (II)

October 6, 2017


- ❏ Main memory is usually divided into two partitions:
 - ❏ Resident operating system, usually held in low memory with interrupt vector
 - ❏ User processes then held in high memory

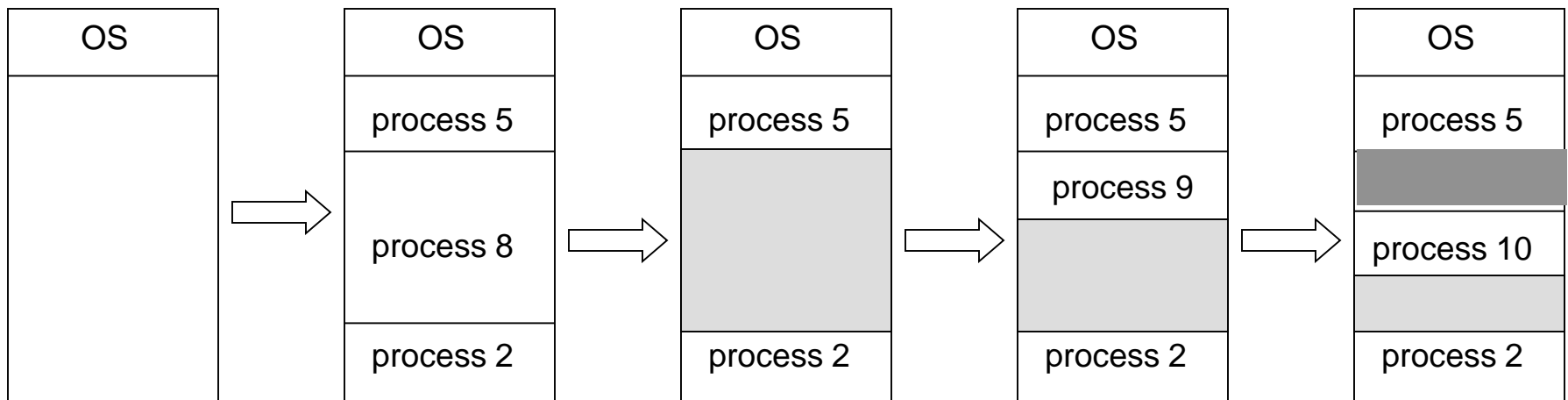


How to Allocate User Memory Space: Contiguous Allocation





Limitation of Contiguous Allocation: Fragmentation

 **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous



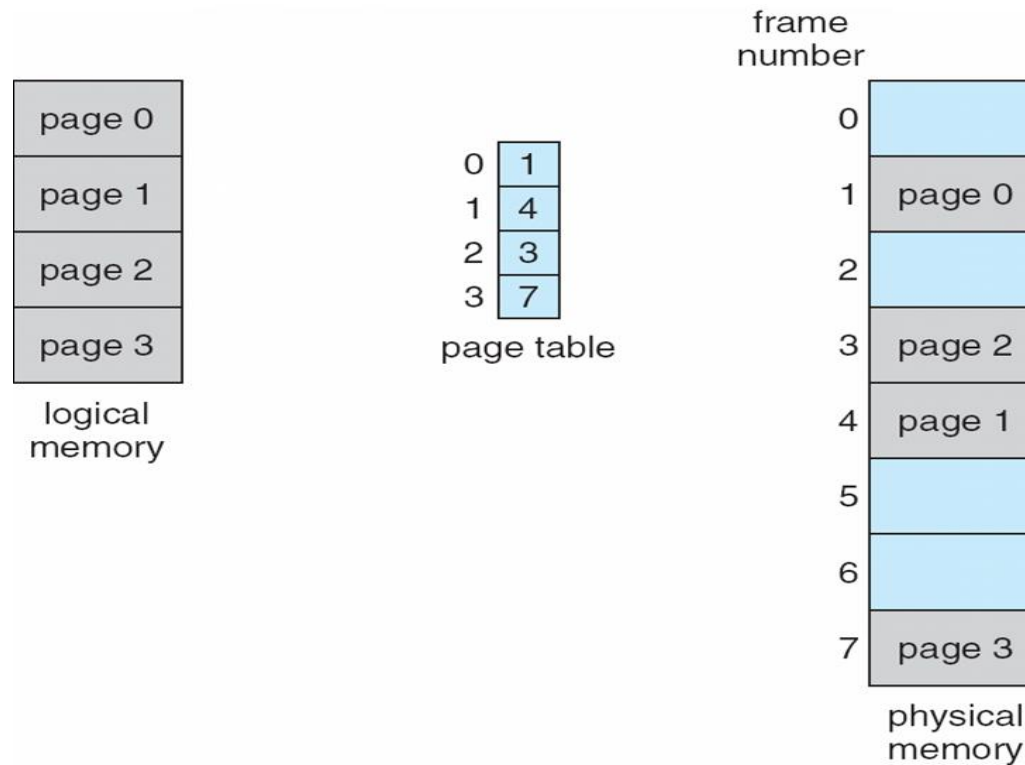
Paging – Memory Management Strategy Adopted by Modern OSes

Key Ideas:

-  Logical address space of a process remains contiguous but the physical address space of it needs not be contiguous
-  Process is allocated physical memory whenever the latter is available

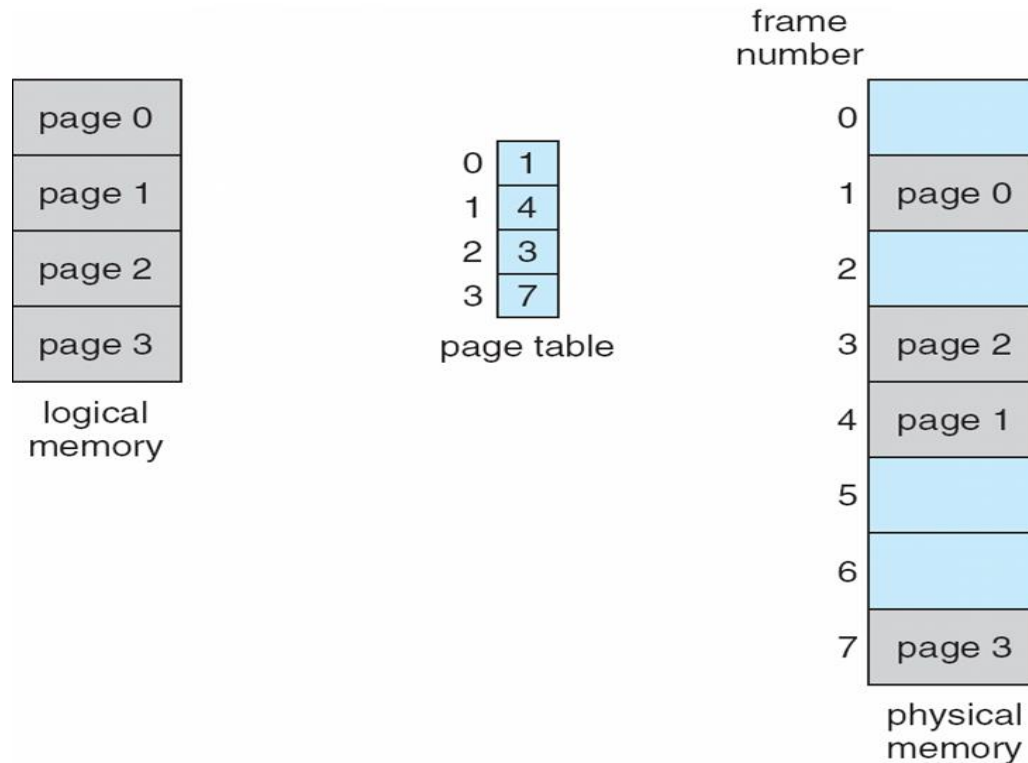
Paging: Key Ideas

- ❏ Divide physical memory (user memory part) into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- ❏ Divide logical memory space of a process into blocks of same size called **pages**



Paging: Key Ideas

- Pages are mapped to frames one-by-one; process-specific **page table** records the mapping and facilitates the logical to physical address translation
- OS keeps track of free frames and allocates frames to new/swap-in processes



Address Translation Scheme

Logical address generated by CPU is divided into:

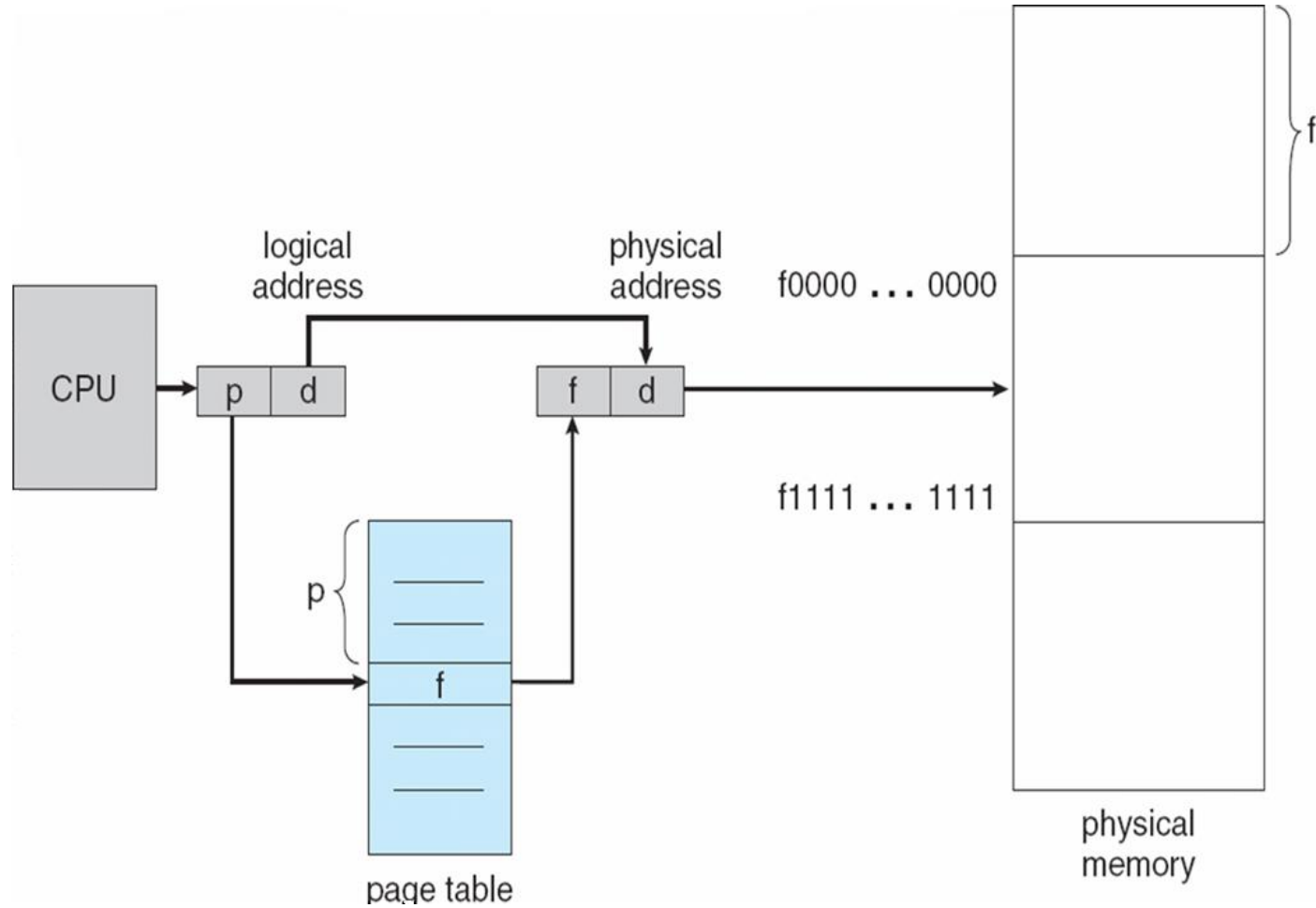
Page number (p) – used as an index into a *page table* which contains base address of each page in physical memory

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

| page number | page offset |
|-------------|-------------|
| p | d |
| $m - n$ | n |

For given logical address space 2^m , *page size* 2^n and *maximum number of pages per process* 2^{m-n}

Paging Hardware



Paging Example

| | |
|----|---|
| 0 | a |
| 1 | b |
| 2 | c |
| 3 | d |
| 4 | e |
| 5 | f |
| 6 | g |
| 7 | h |
| 8 | i |
| 9 | j |
| 10 | k |
| 11 | l |
| 12 | m |
| 13 | n |
| 14 | o |
| 15 | p |

logical memory

| | |
|---|---|
| 0 | 5 |
| 1 | 6 |
| 2 | 1 |
| 3 | 2 |

page table

| | |
|----|------------------|
| 0 | |
| 4 | i j k l |
| 8 | m n o p |
| 12 | |
| 16 | |
| 20 | a b c d |
| 24 | e f g h |
| 28 | |

physical memory

32-byte memory and
4-byte pages

How to find the
physical
addresses for
logical addresses
7 and 12?

Efficiency Limitation & Solution


- ❏ 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)**

Associative Memory

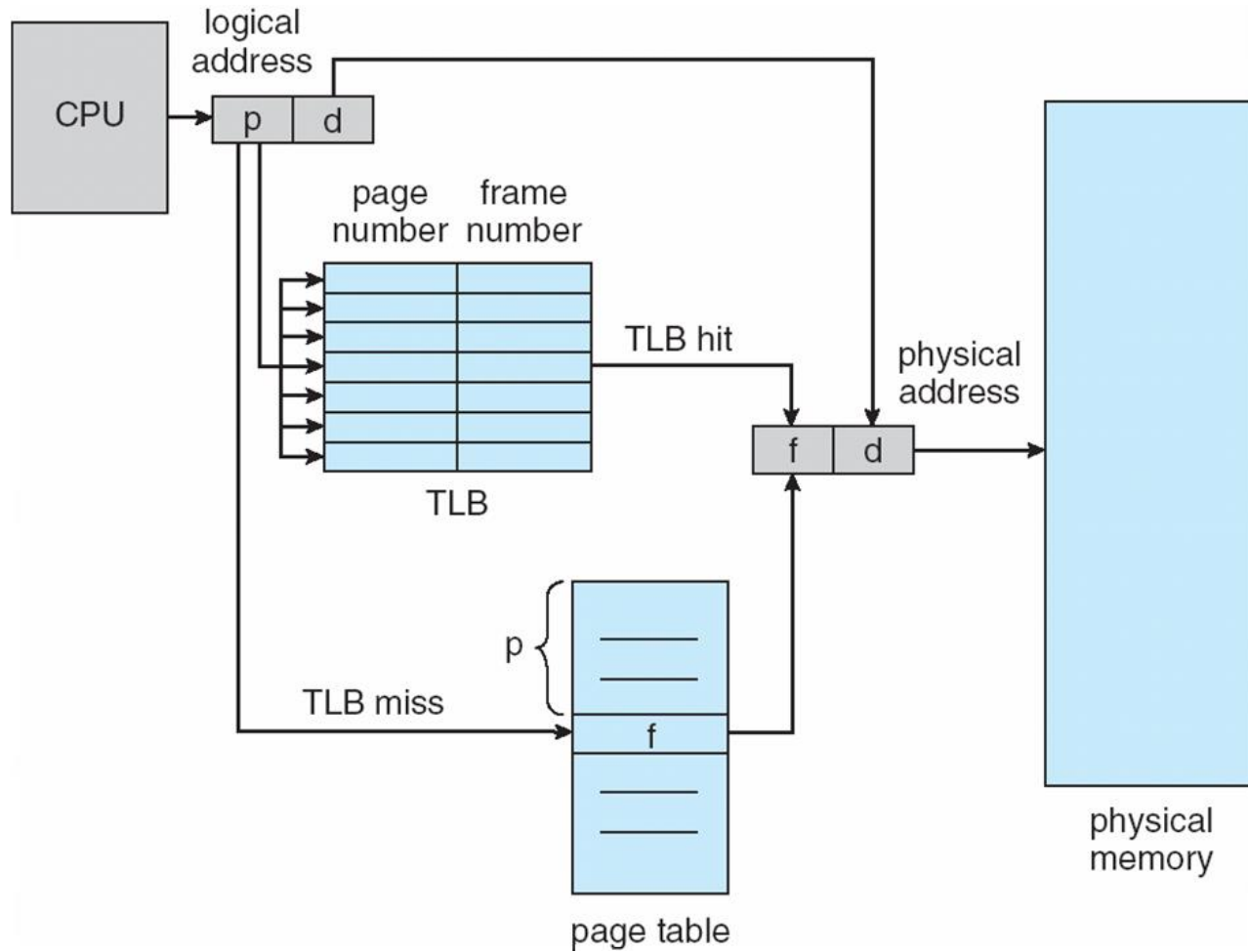
 Associative memory – parallel search

| Page # | Frame # |
|--------|---------|
| | |
| | |
| | |
| | |

 Input page number p

 If p is in associative register, get frame # out

Paging Hardware With TLB



Paging Example

| | |
|----|---|
| 0 | a |
| 1 | b |
| 2 | c |
| 3 | d |
| 4 | e |
| 5 | f |
| 6 | g |
| 7 | h |
| 8 | i |
| 9 | j |
| 10 | k |
| 11 | l |
| 12 | m |
| 13 | n |
| 14 | o |
| 15 | p |

logical memory

| | |
|---|---|
| 0 | 5 |
| 1 | 6 |
| 2 | 1 |
| 3 | 2 |

page table

| | |
|----|------------------|
| 0 | |
| 4 | i j k l |
| 8 | m n o p |
| 12 | |
| 16 | |
| 20 | a b c d |
| 24 | e f g h |
| 28 | |

physical memory

32-byte memory and
4-byte pages

How to find the
physical
addresses for
logical addresses
7 and 12?

TLB

| | |
|---|---|
| 1 | 6 |
| 2 | 1 |

Effective Access Time

- ❏ Associative Lookup = ϵ microsecond
- ❏ Assume memory cycle time is t microseconds
- ❏ Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- ❏ Hit ratio = α
- ❏ **Effective Access Time** (EAT)

$$\begin{aligned} \text{EAT} &= (t + \epsilon) \alpha + (2t + \epsilon)(1 - \alpha) \\ &= 2t + \epsilon - \alpha t \end{aligned}$$

TLB Hardware is shared by multiple processes

- ❏ Problem: when a process is swapped out and a new process is swapped in, the content of the TLB becomes outdated
- ❏ Solutions:
 - ❏ flush the TLB when process switch; or
 - ❏ store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

Implementation of Page Table



- ❏ Page table is kept in main memory (kernel space)
- ❏ **Page-table base register (PTBR)** points to the page table
- ❏ **Page-table length register (PTLR)** indicates size of the page table
- ❏ Memory protection implemented by
 - ❏ PTLR specifies the length of page table (the number of pages for a process)
 - ❏ If the page number of an address \geq the value of PTLR, the logical address is invalid

Shared Pages

Shared code

-  One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

Private code and data

-  Each process keeps a separate copy of the code and data
-  The pages for the private code and data can appear anywhere in the logical address space