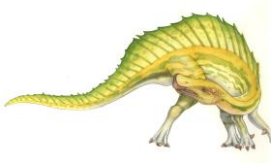
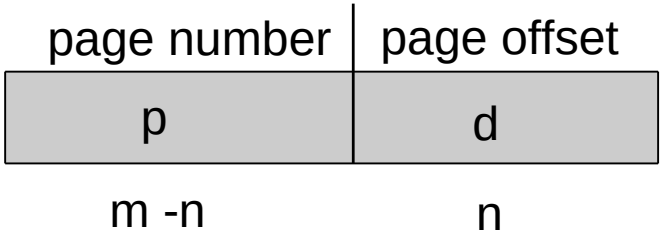
Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

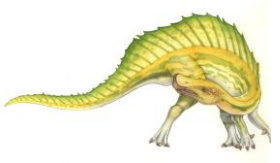


Address Translation Scheme

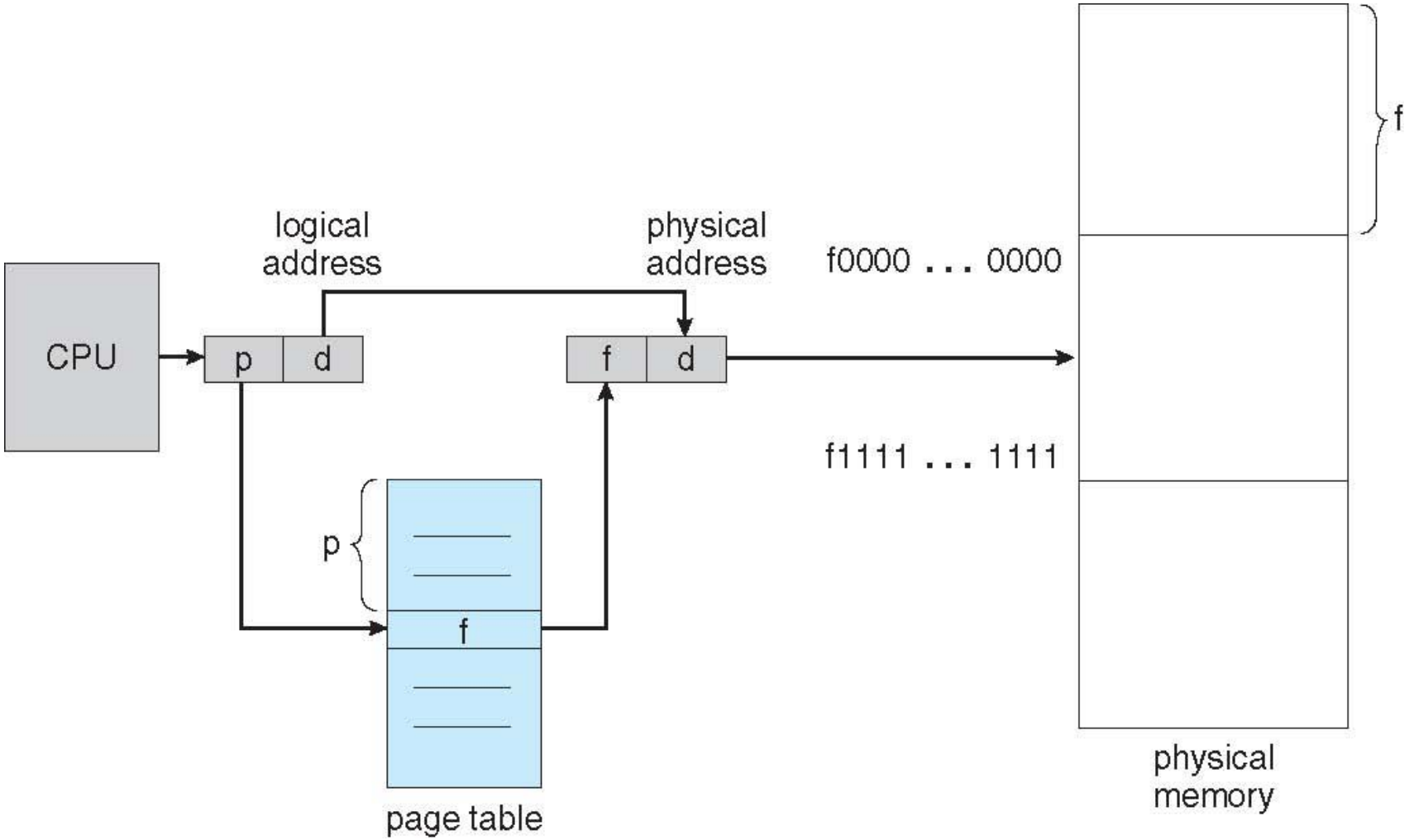
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



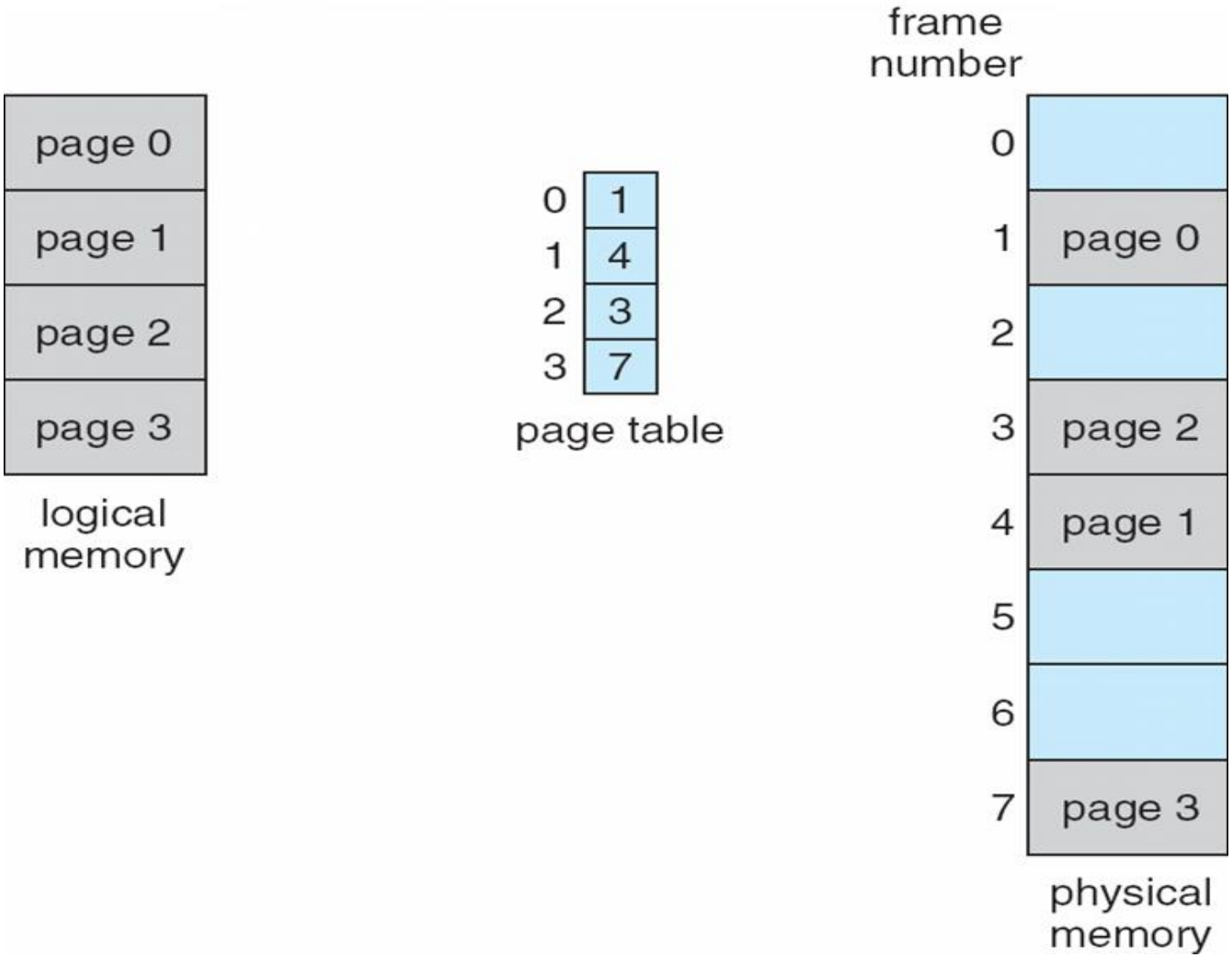
  - For given logical address space 2^m and page size 2^n



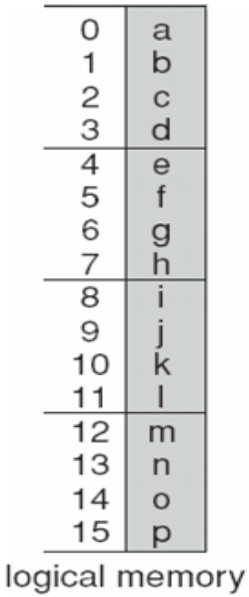
Paging Hardware

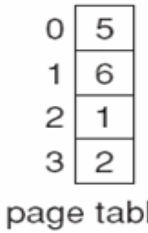


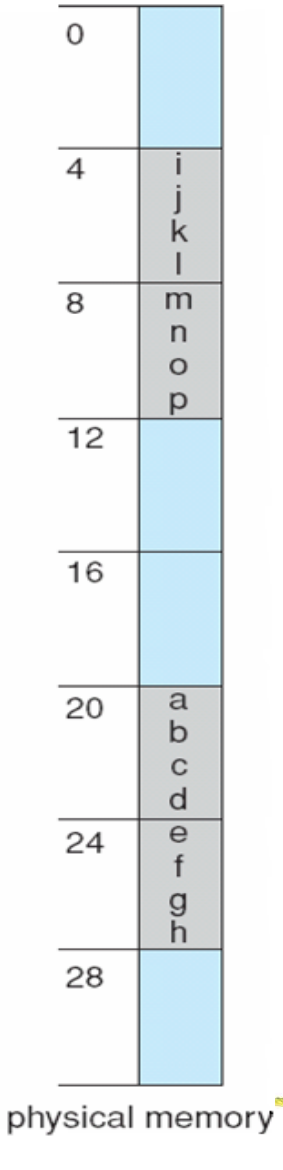
Paging Model of Logical and Physical Memory



Paging Example



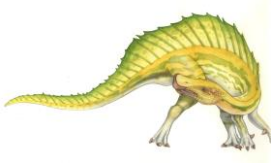




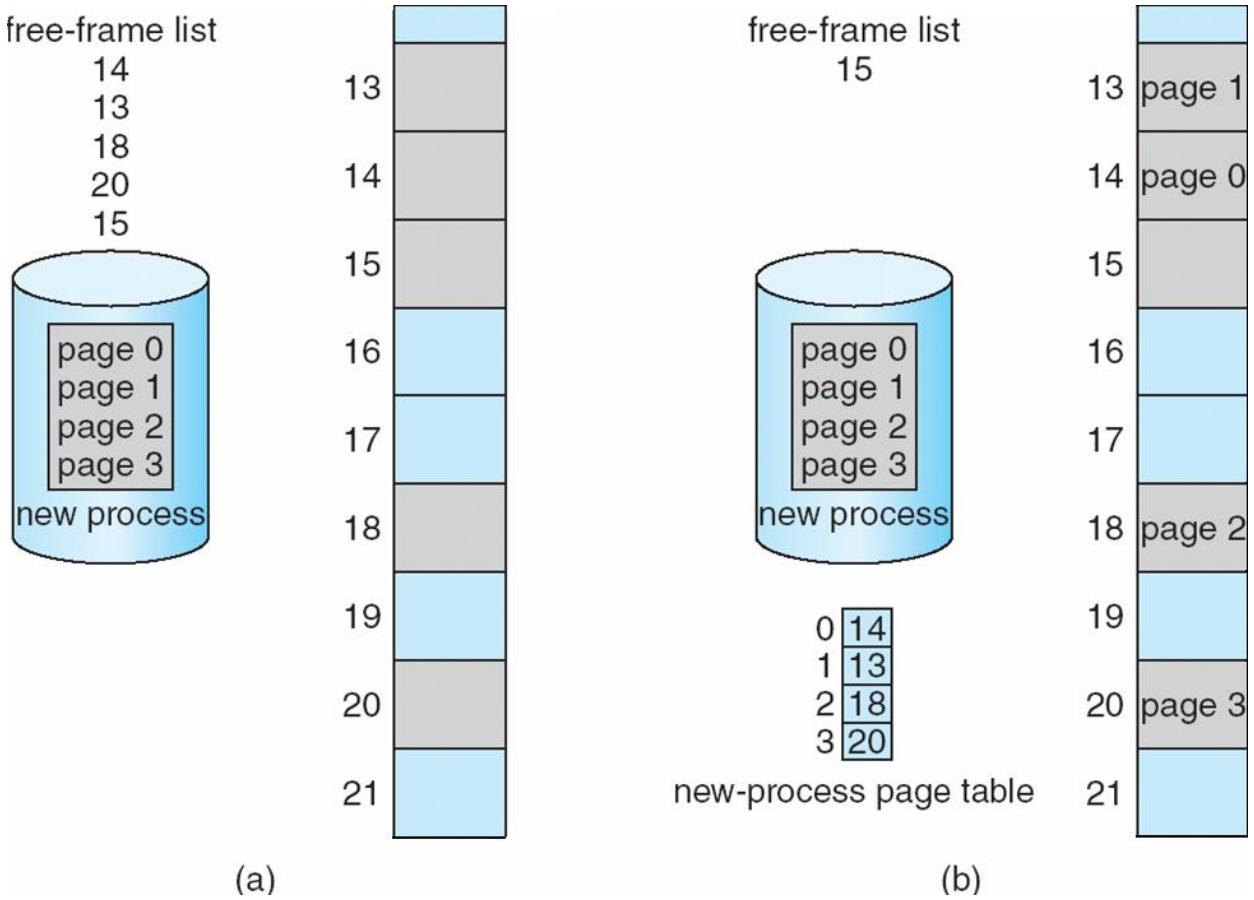
n=2 and *m*=4 32-byte memory and 4-byte pages

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

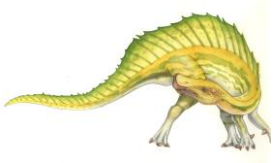


Before allocation

After allocation

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 (Cont.)

- 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
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

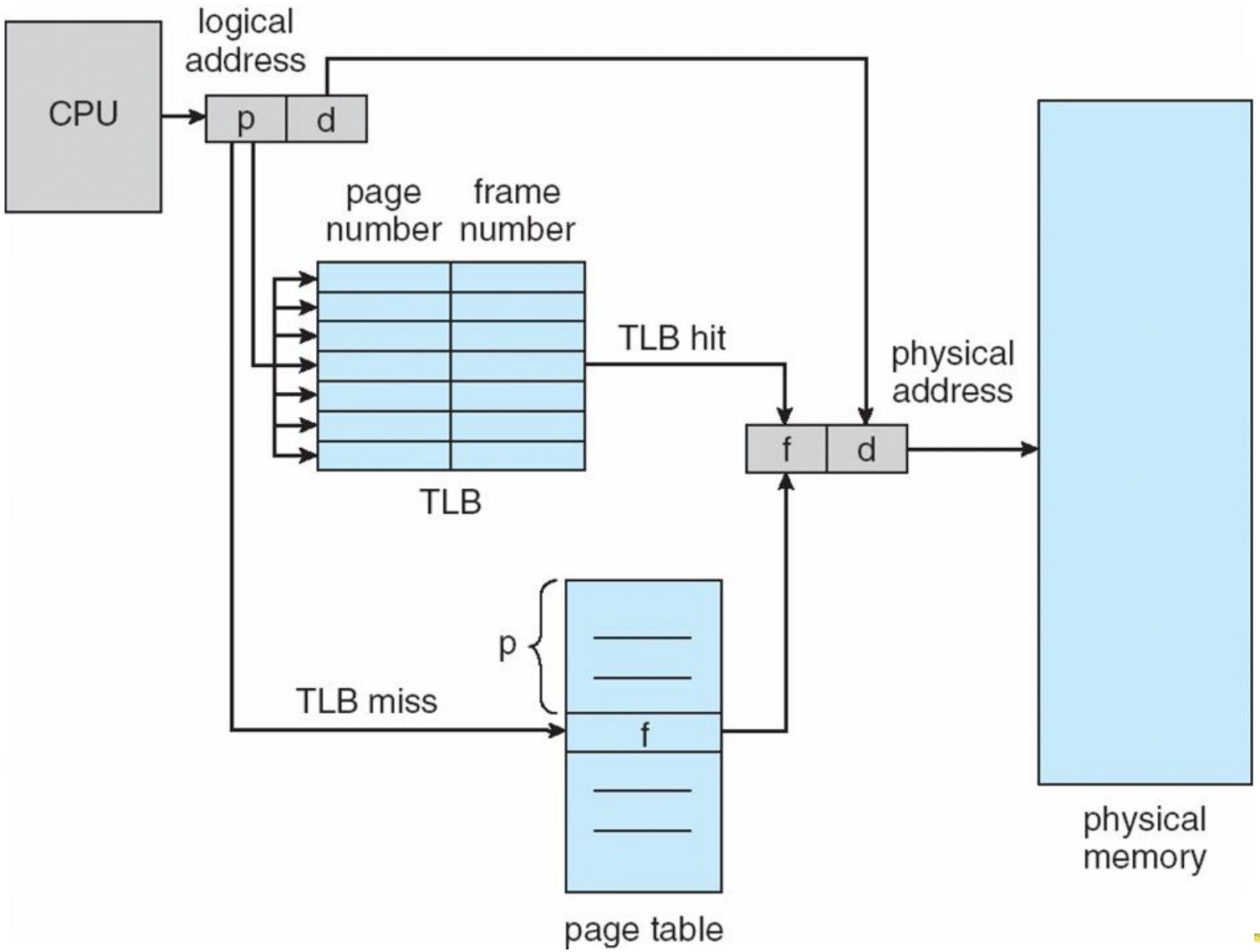
Associative Memory

- Associative memory – parallel search

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

Paging Hardware With TLB

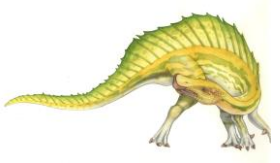


Effective Access Time

- Associative Lookup = ε time unit
  - Can be < 10% of memory access time
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- **Effective Access Time** (**EAT**)

$$EAT = \alpha\,(\varepsilon+1) + (1-\alpha)\,(\varepsilon+1+1)$$
$$= 2 + \varepsilon - \alpha$$

- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.80 x (100+20) + 0.20 x (20+100+100) = 140ns
- Consider more realistic hit ratio -> α = 99%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.99 x (20+100) + 0.01 x (20+100+100) = 120.8ns



Paging

Q. 1 Consider a logical address space of 8 pages of 1024 words each, mapped onto a physical memory of 32 frames.

a. How many bits are there in the logical address?

b. How many bits are there in the physical address?

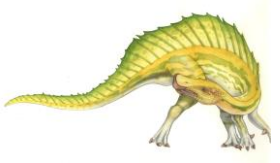
Quiz 2 (Section 1)

- Consider the following snapshot of a system in which five resources A, B, C, D and E are available. The system contains a total of 2 instances of A, 1 of resource B, 1 of resource C, 2 resource D and 1 of resource E.

	Allocation					*Request*					*Available*				
	A	*B*	*C*	*D*	*E*	*A*	*B*	*C*	*D*	*E*	*A*	*B*	*C*	*D*	*E*
P_0	1	0	1	1	0	0	1	0	0	1	2	1	1	2	1
P_1	1	1	0	0	0	0	0	1	0	1					
P_2	0	0	0	1	0	0	0	0	0	1					
P_3	0	0	0	0	0	1	0	1	0	1					

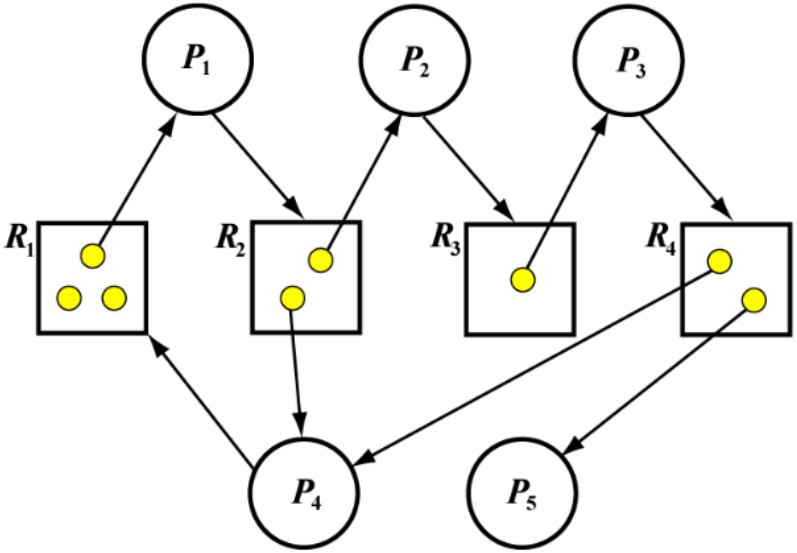
a) Convert this matrix representation to a resource allocation graph.

b) Use the deadlock detection algorithm to determine whether the system contains a deadlock. Which processes are involved in the deadlock?



Quiz 2 (Section 2)

- Consider the following resource allocation graph.



a) Convert it to the matrix representation (i.e., Allocation, request and Available).

b) Use the deadlock detection algorithm to determine whether the system contains a deadlock. Which processes are involved in the deadlock?