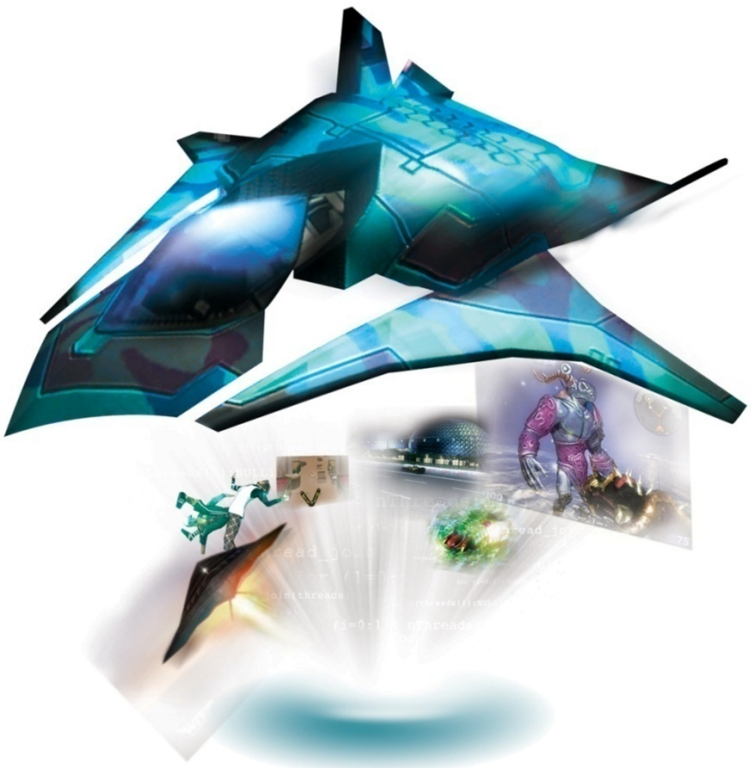
DM2112
DIGITAL ENTERTAINMENT SYSTEMS

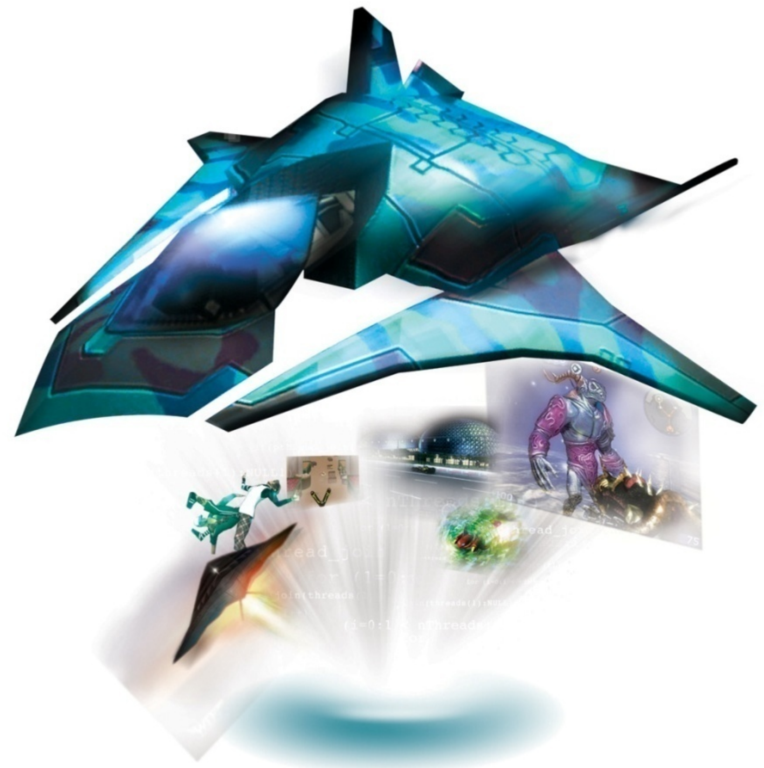


Operating System Concepts Pt 2

Today's Menu

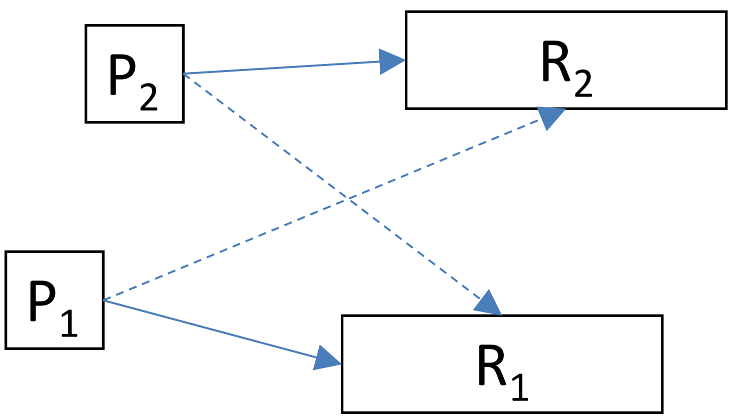
- Deadlocks
- Introduction to Memory Management
  - Swapping
  - Paging

DEADLOCKS



The Deadlock Problem

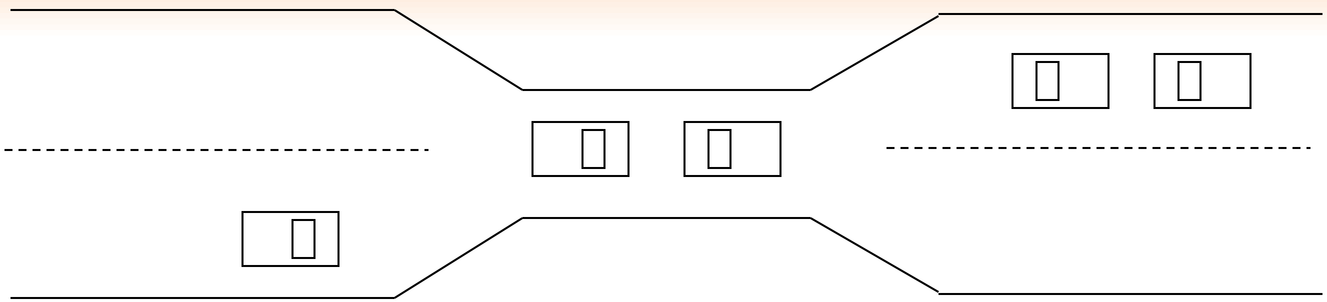
- Occurs when a process waits for another process to release resource.
  - Both requires addition resource to proceed
  - Processing becomes stalled



P_1 grabs R_1	P_2 grabs R_2
P_1 waits for R_2	P_2 waits for R_1

  - Example: process (or processes) consistently uses 100% CPU but no meaningful progress (does not appear to do anything)

Bridge Crossing Example



- Traffic only in one direction
- Each section of a bridge can be viewed as a resource
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
- Several cars may have to be backed up if a deadlock occurs
- Starvation is possible
- Note – Most OSes do not prevent or deal with deadlocks

Resource System Model

- Resource types R_1, R_2, . . ., R_m

  Can be CPU cycles, memory space, I/O devices

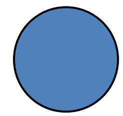
- Each resource type R_i has W_i instances.

- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

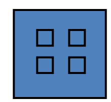
- Issues commonly arise when certain resources cannot be shared or such sharing would result in data corruption.

Resource Allocation Graph

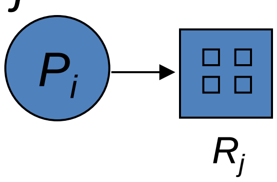
- Process



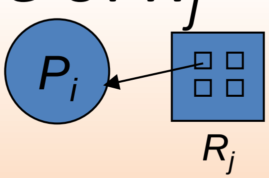
- Resource Type with 4 instances



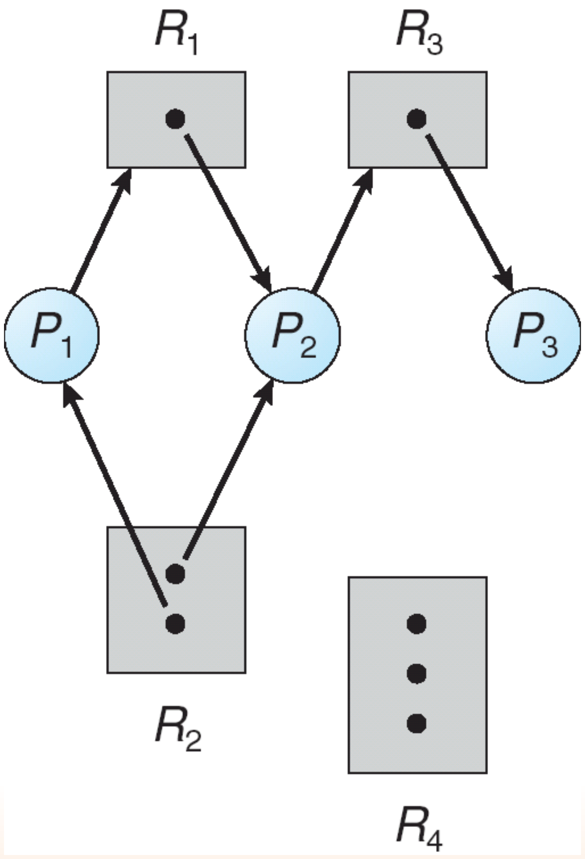
- P_i requests instance of R_j



- P_i is holding an instance of R_j



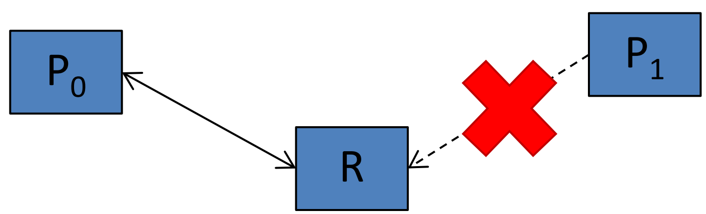
Example of a Resource Allocation Graph



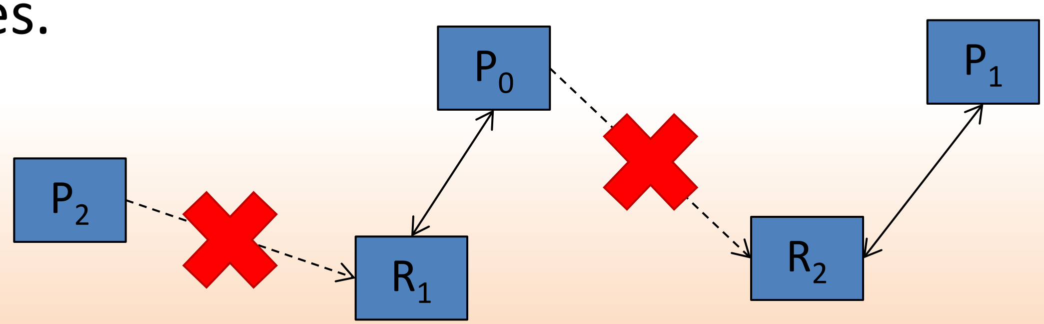
Deadlock Characterization

- A deadlock can arise if 4 conditions are present simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource.

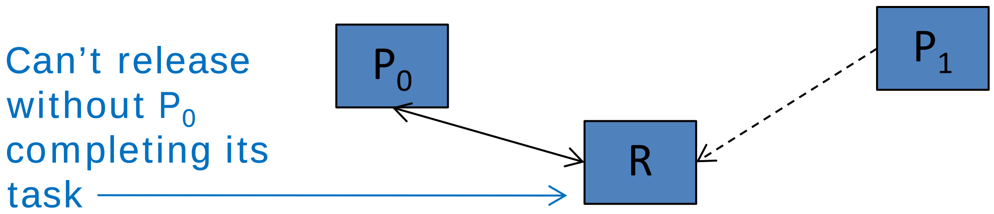


- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.

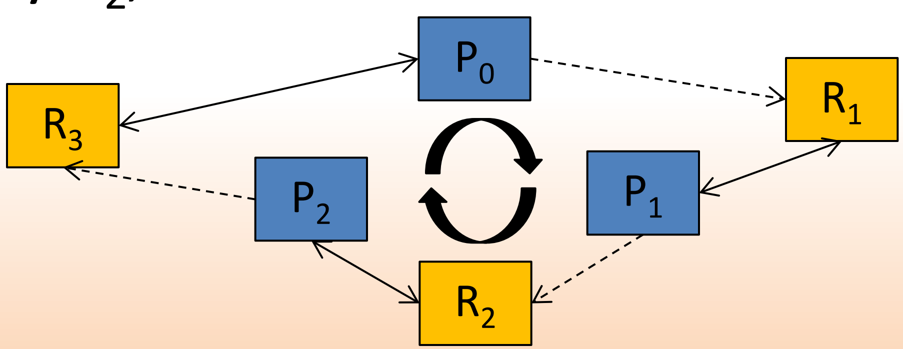


Deadlock Characterization (Continued)

- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task



- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, etc



Methods for Handling Deadlocks

1. Ensure that the system will *never* enter a deadlock state (Deadlock Avoidance & Prevention).

2. Allow the system to enter a deadlock state and then recover (Deadlock Recovery).

3. Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX (Terminate on Detection).

For this module, we will only look at Prevention.

Deadlock Prevention

- **Mutual exclusion** – Removing the mutual exclusion condition means that no process may have exclusive access to a resource.

- **Hold and wait** – This condition may be removed by requiring processes to request all the resources they will need before starting up (or before embarking upon a particular set of operations).
  - This advance knowledge is often difficult to satisfy, and is an inefficient use of resources
  - Another way is to require processes to release all their resources before requesting all the resources they will need. This is also often not practical.

Deadlock Prevention (Continued)

- **No preemption** – Must be able to force a process to stop using a resource.
  - Difficulty is that a process must be able to have a resource for a certain amount of time, or else the processing outcome may be inconsistent

- **Circular wait** – A hierarchy can be used to determine which processes can use which resources first.
  - Interrupts can also be disabled during critical sections (i.e. during times when a program code accesses a shared resource like data or a device

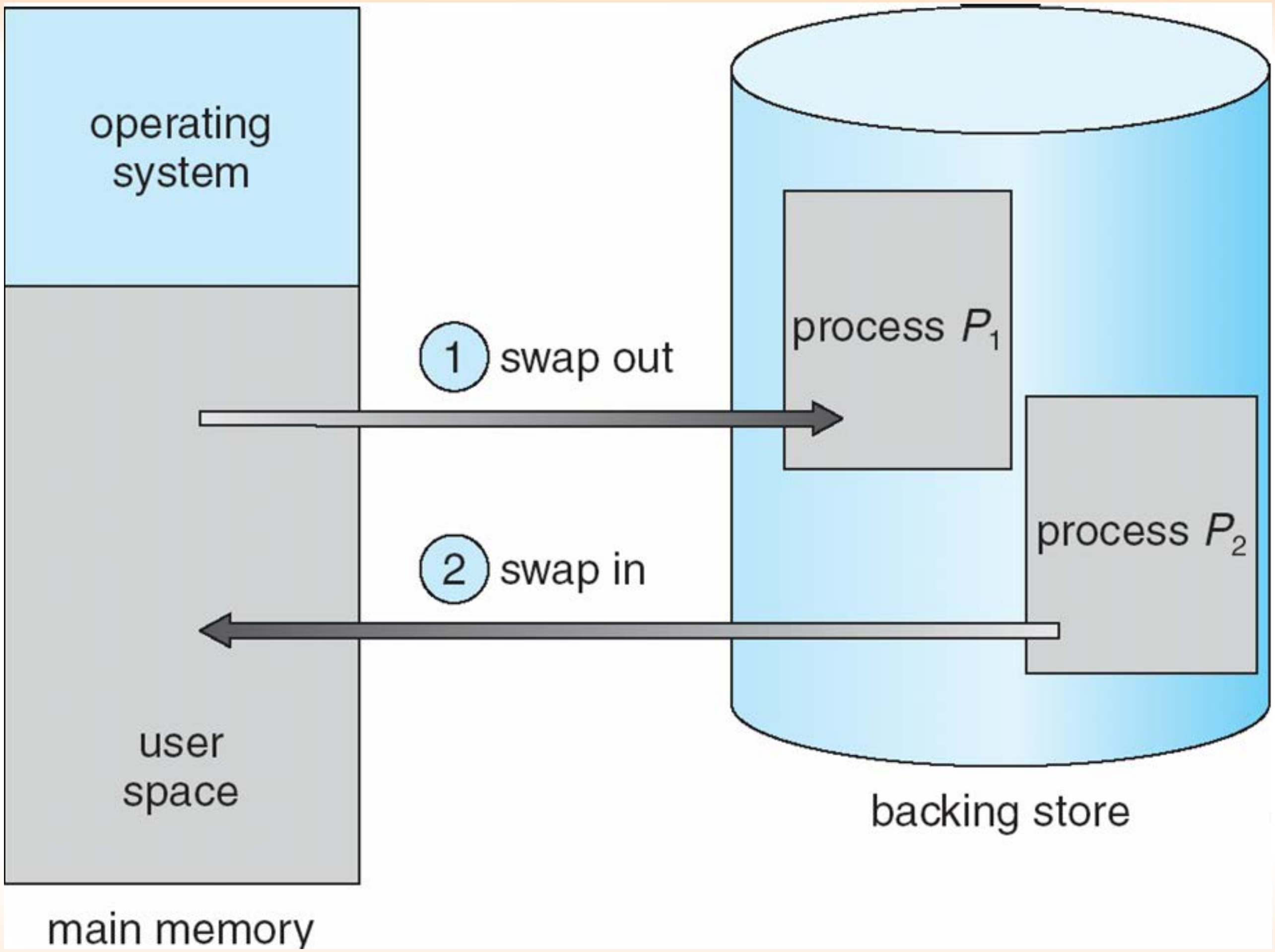
INTRODUCTION TO MEMORY MANAGEMENT

Swapping

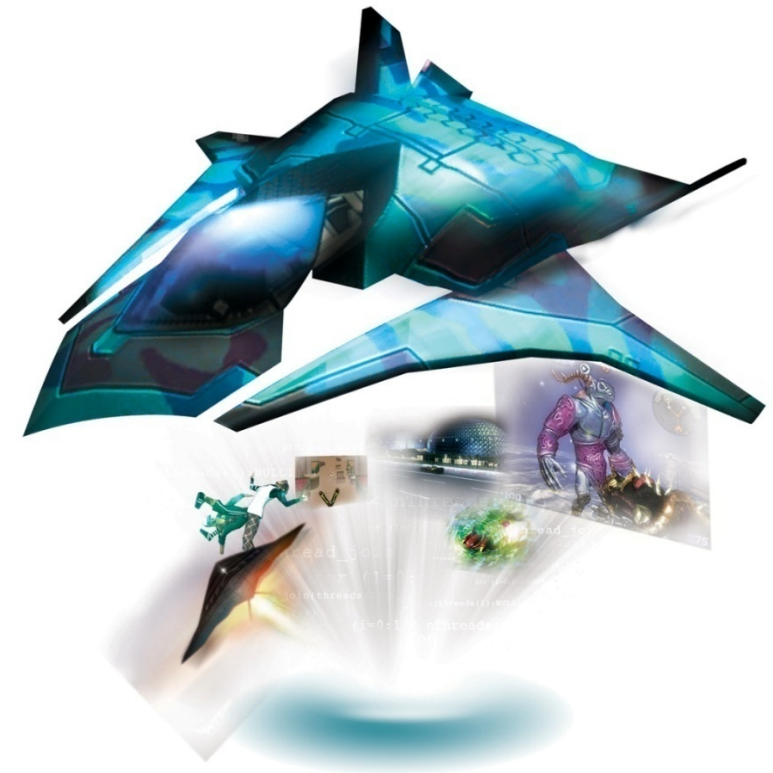
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images

Schematic View of Swapping



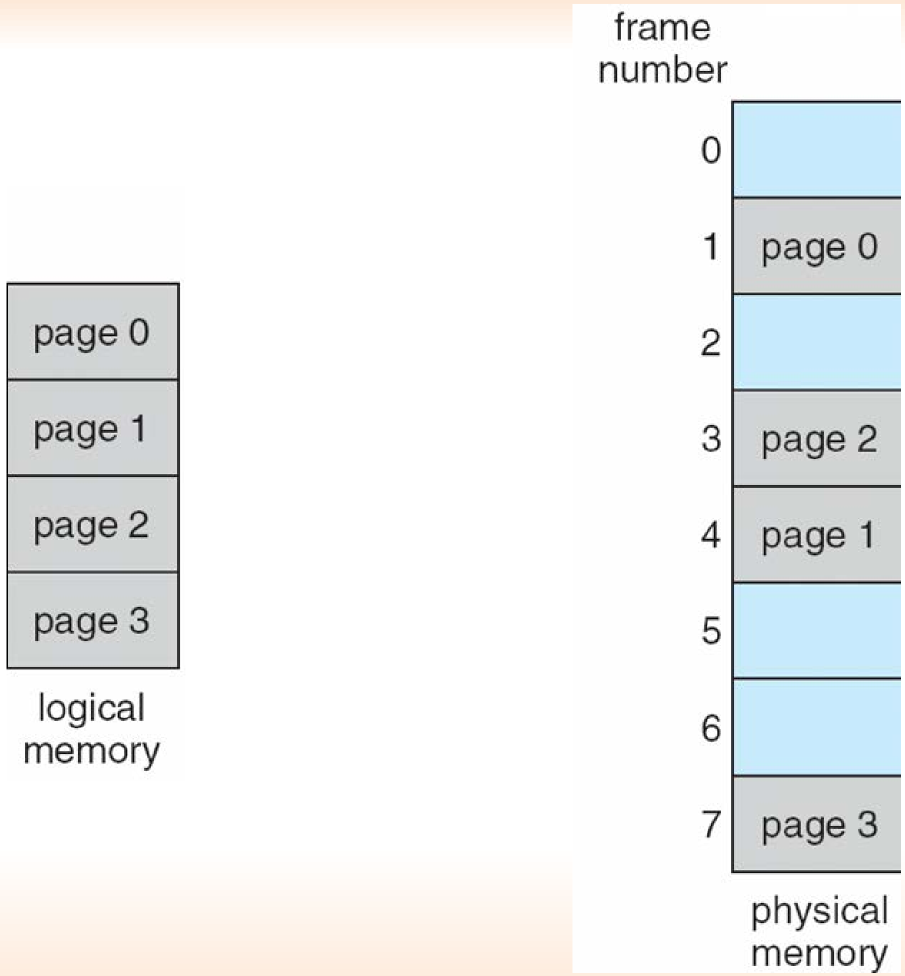
PAGING



Paging

- A process is allocated physical memory whenever available.
- Logical memory is the memory that a process “thinks” it has.
- Physical memory is divided into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes).
- Logical memory is divided into blocks each of same size called **pages.**

Logical vs. Physical Memory



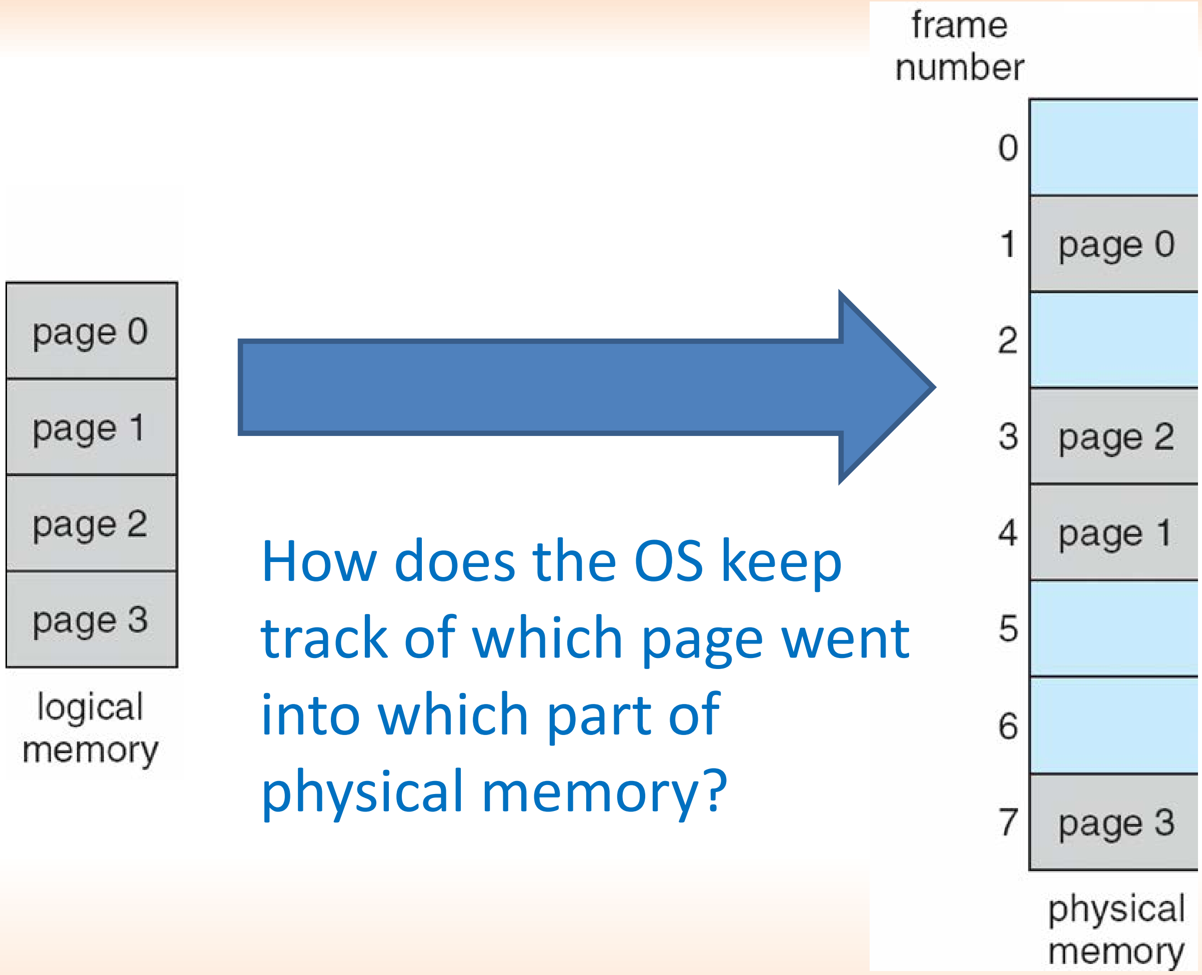
Paging

- The operating system has to:
  - Keep track of all free frames
  - Set up a page table to translate logical to physical addresses

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

- If not all the storage space in a frame is used, then there is **internal fragmentation**.

Address Translation



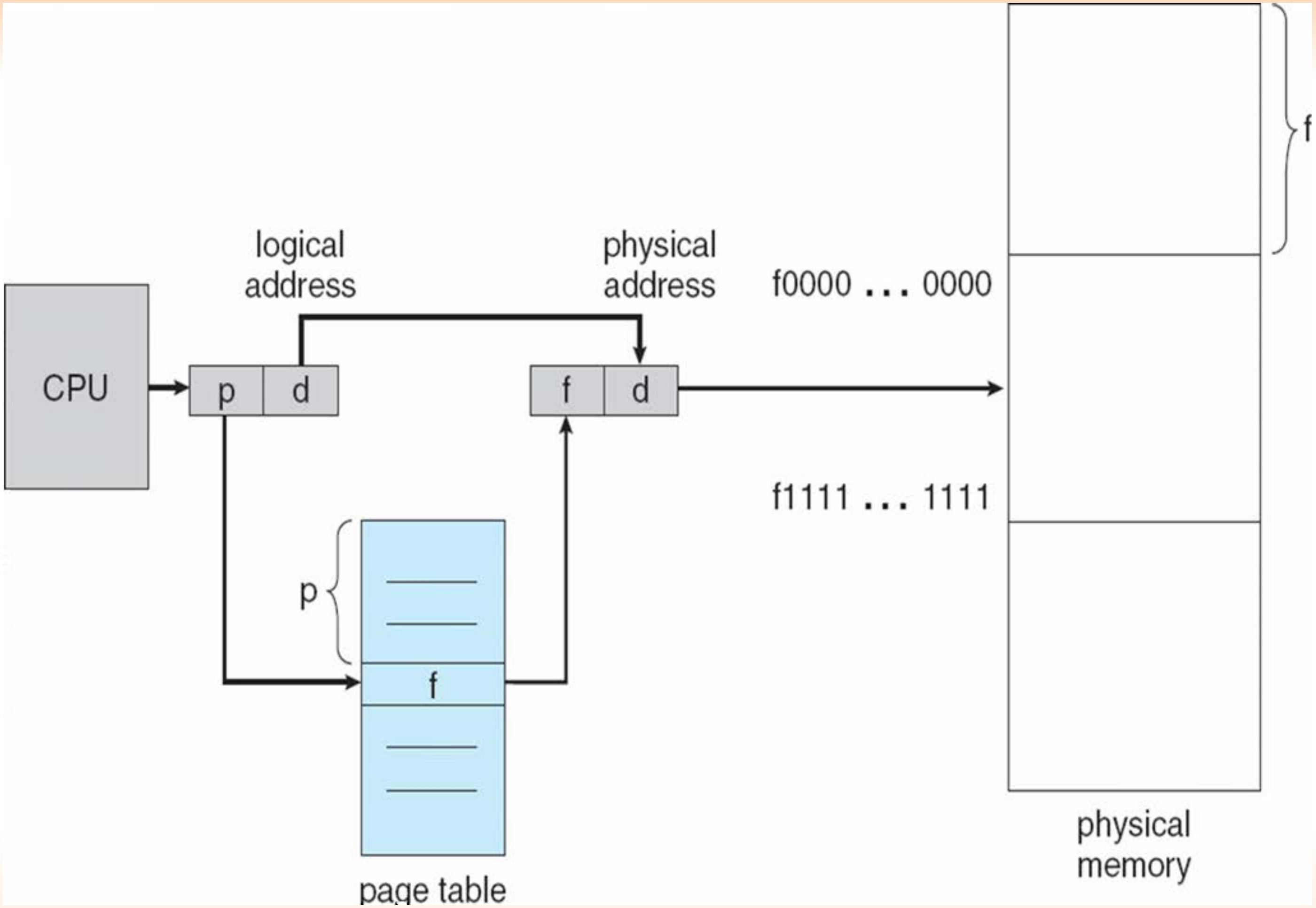
How does the OS keep track of which page went into which part of physical memory?

Address Translation

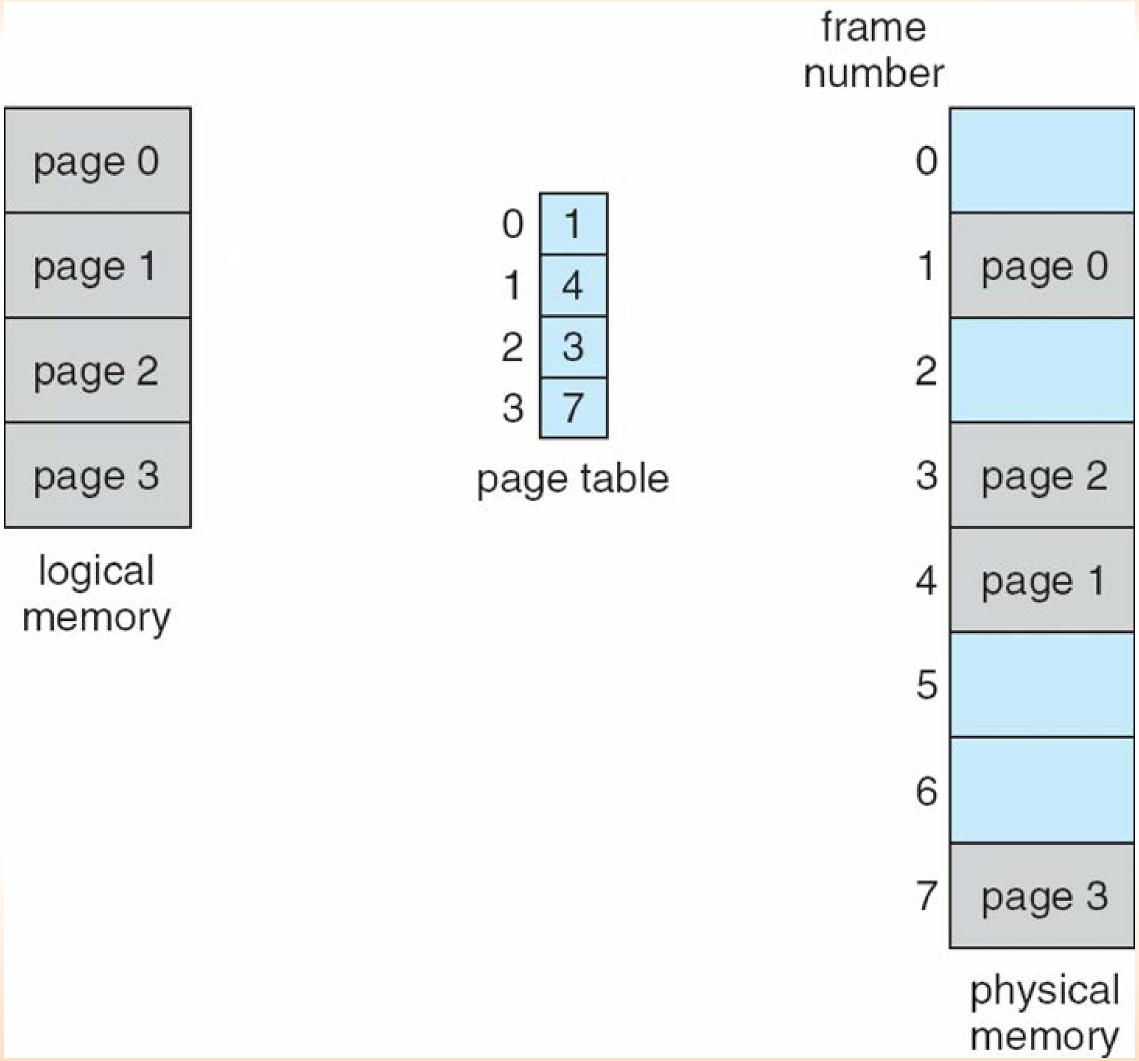
- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
p	*d*

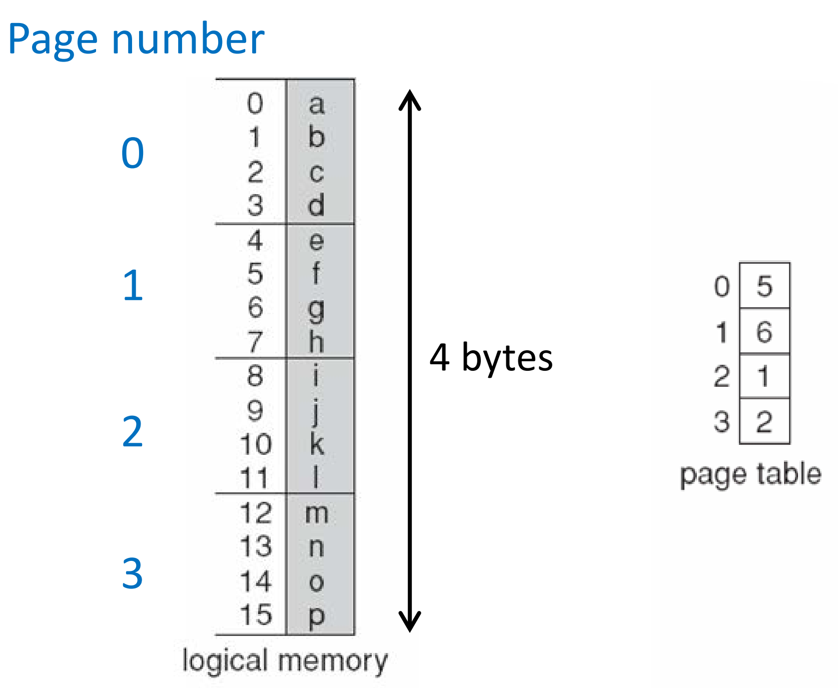
Paging Hardware

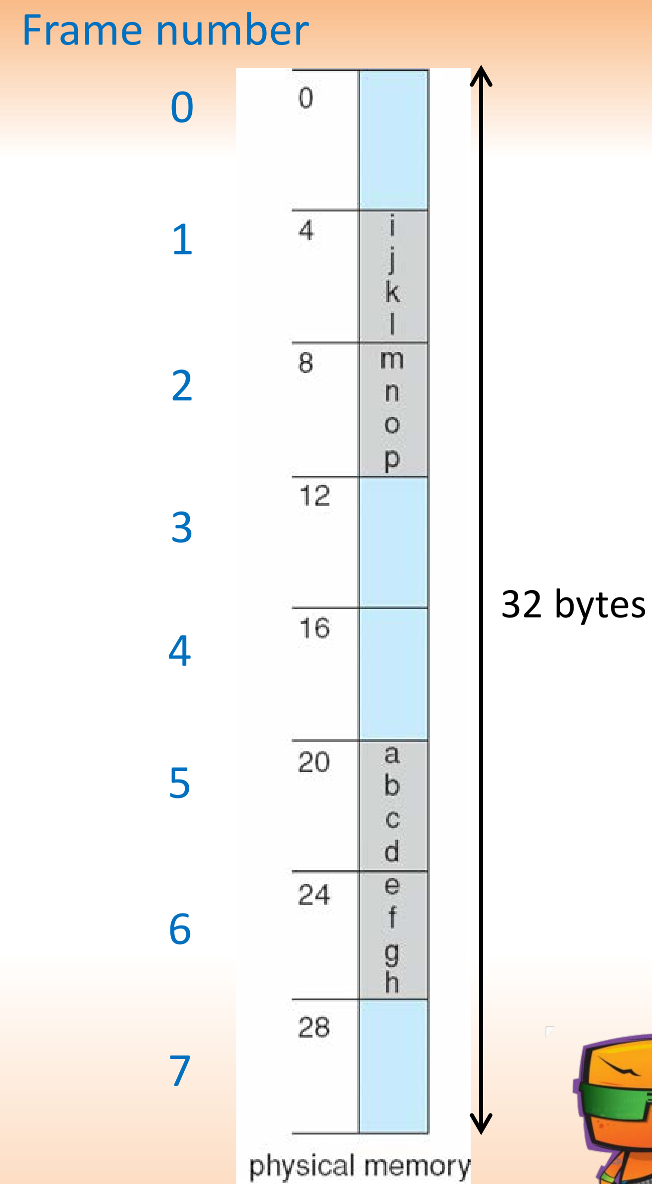


Paging Model of Logical and Physical Memory



Paging Example





32-byte memory and 4-byte pages

Nanyang Polytechnic

Acknowledgements

- Slides adapted from Operating System Concepts (8th Edition), by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin. John Wiley & Sons Inc., ISBN 0-470-12872-0
- http://en.wikipedia.org/wiki/Deadlock

More Next Time

- Questions?