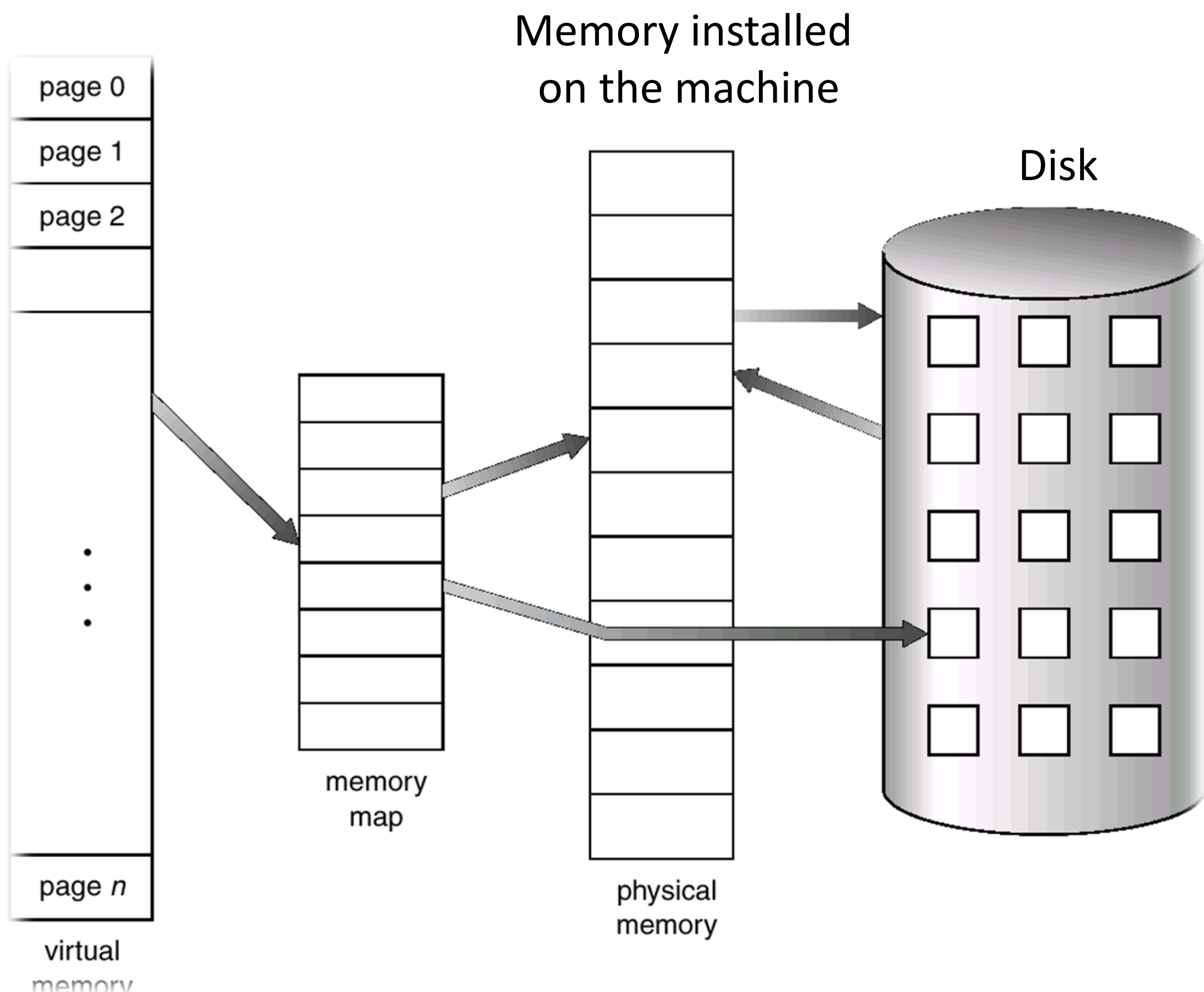
Operating Systems

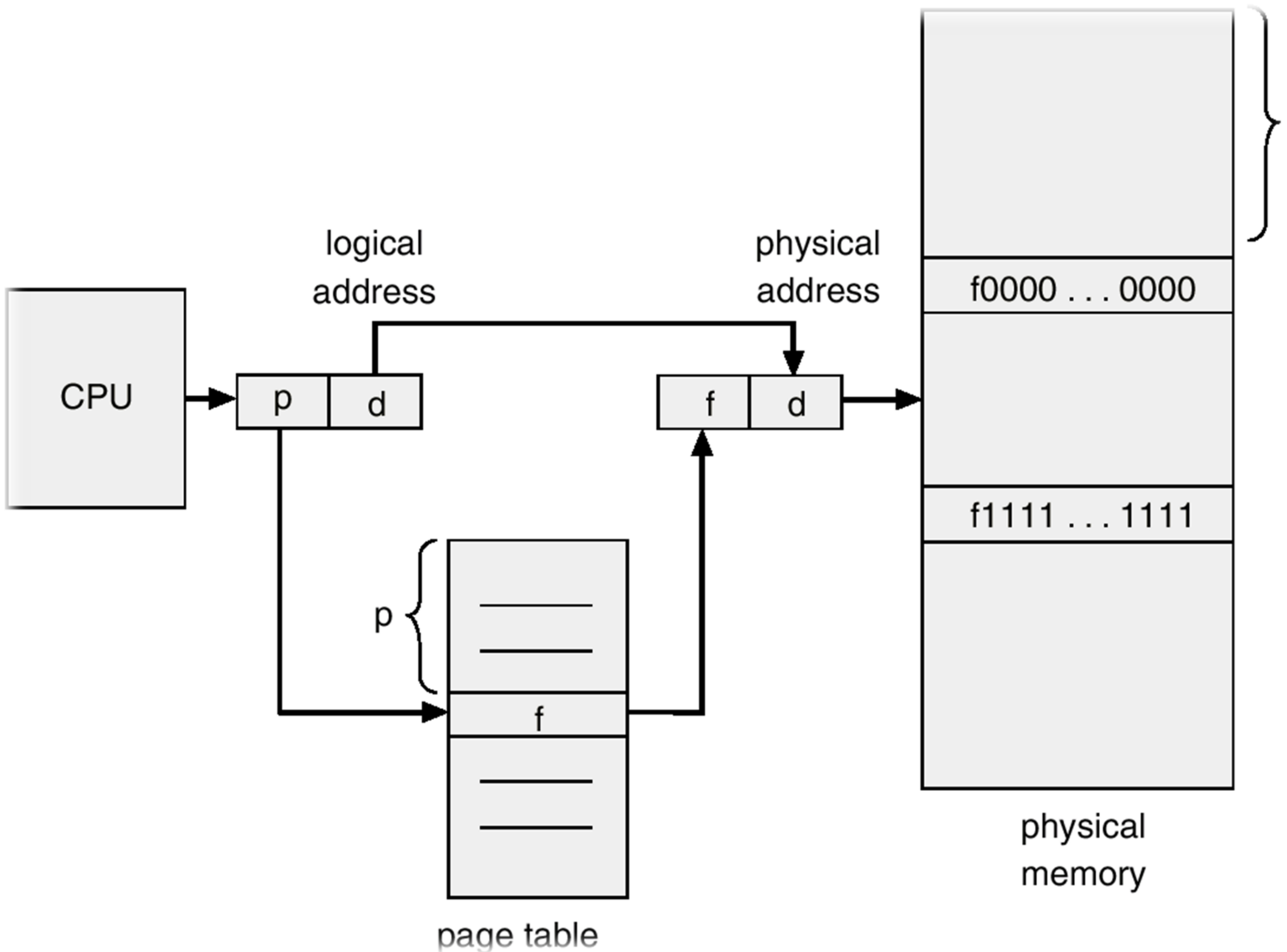
Memory Management III

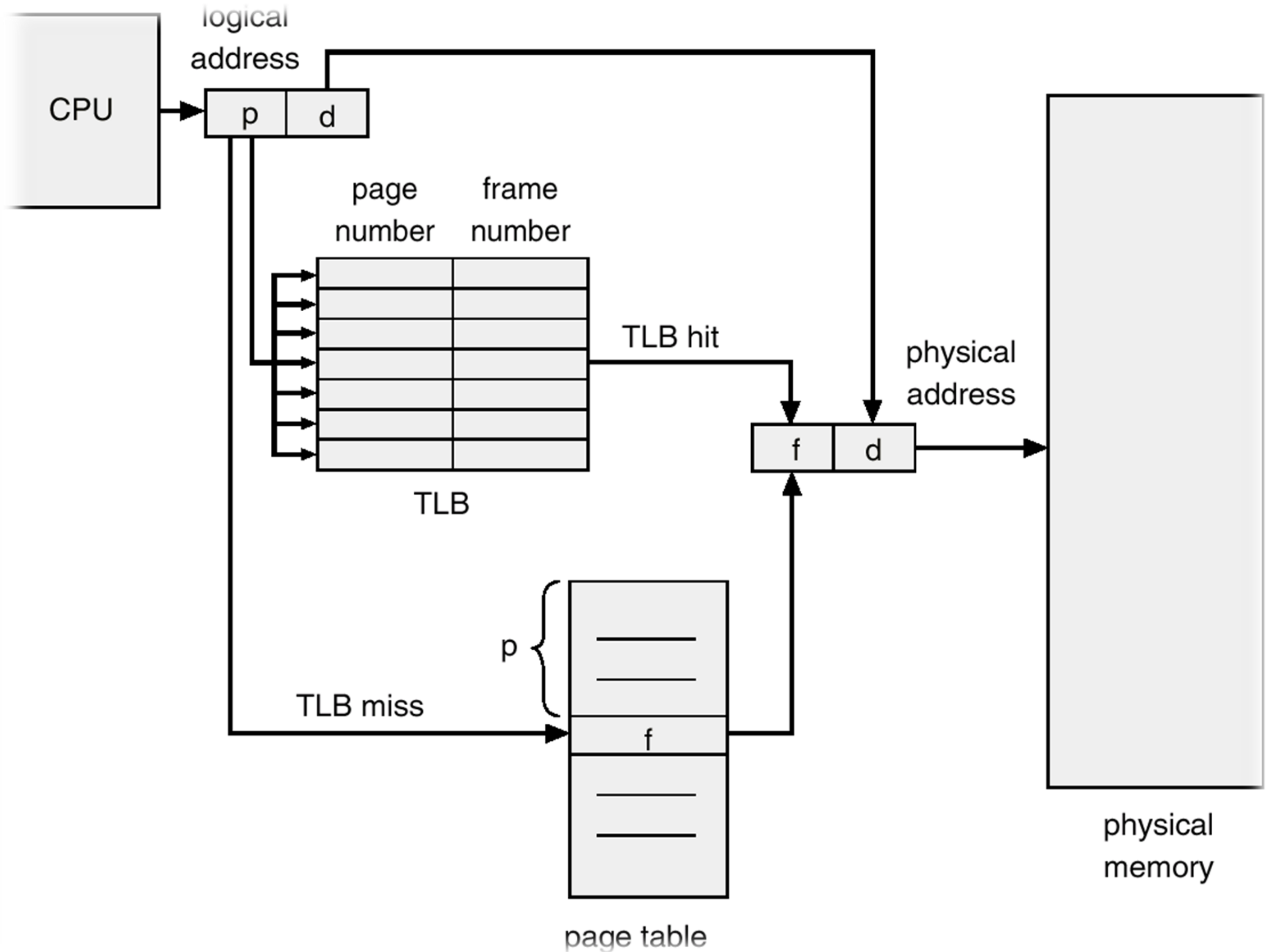
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

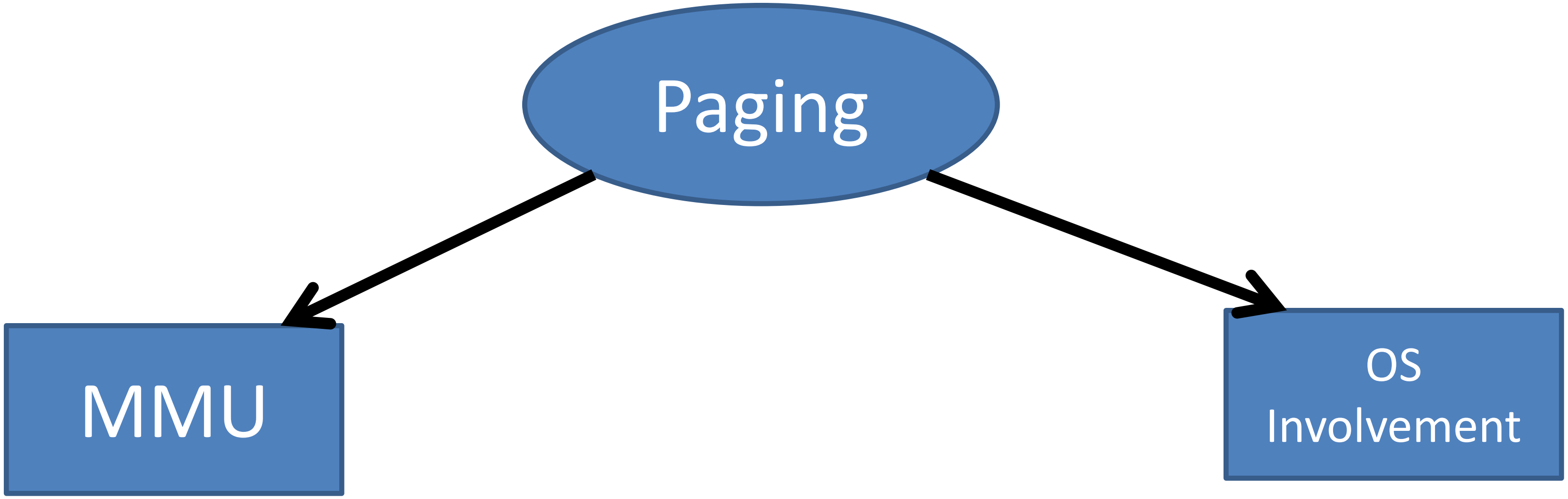
Summary of Paging

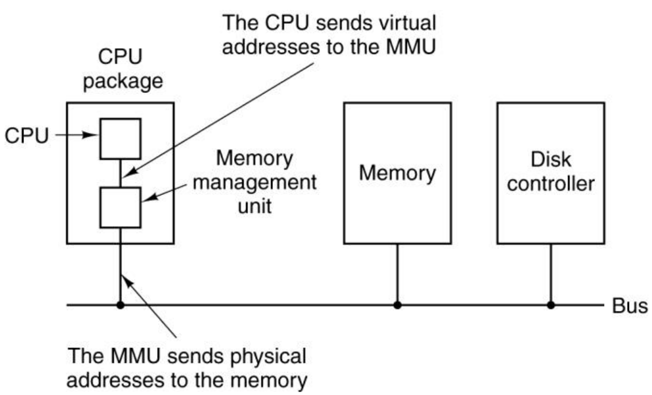
- Virtual address space bigger than physical memory
- Mapping virtual address to physical address
- Virtual address space divided into fixed-size units called pages
- Physical address space divided into fixed-size units called ~~pages~~ frames
- Virtual address space of a process can be non-contiguous in physical address space


Memory installed
on the machine
page 0
page 1
page 2
page n
virtual
memory
memory
map
physical
memory
Disk


logical address
physical address
CPU
p
d
f
d
f0000 . . . 0000
f1111 . . . 1111
p
f
page table
physical memory


CPU
logical address
p
d
page number
frame number
TLB hit
physical address
f
d
TLB
p
TLB miss
f
page table
physical memory

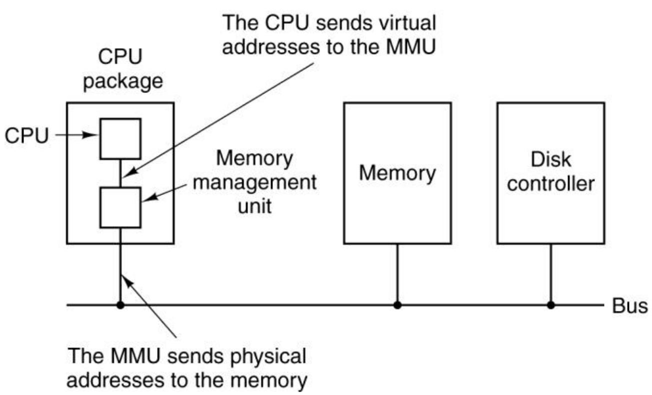

Paging
MMU
OS
Involvement


The CPU sends virtual
addresses to the MMU
CPU
package
CPU
Memory
management
unit
Memory
Disk
controller
Bus
The MMU sends physical
addresses to the memory

Paging

MMU

OS Involvement



?

OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
- When process exits
- When page fault occurs

OS Involvement With Paging

- When a new process is created
  - Determine how large the program and data will be (initially)
  - Create page table
  - Allocate space in memory for page table
  - Record info about page table and swap area in process table
- When a process is scheduled for execution
- When process exits
- When page fault occurs

OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
- When process exits
- When page fault occurs

OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
  - MMU reset for the process
  - TLB flushed
  - Process table made current
- When process exits
- When page fault occurs

OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
- When process exits
  - OS releases the process page table
  - Frees its pages and disk space
- When page fault occurs

OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
- When process exits
- When page fault occurs

Page Fault Handling

1. The hardware:
   - Saves program counter
   - Traps to kernel
2. An assembly routine saves general registers and calls OS
3. OS tried to discover which virtual page is needed
4. OS checks address validation and protection and assign a page frame (page replacement may be needed)

Page Fault Handling

5. If page frame selected is dirty
   - Page scheduled to transfer to disk
   - Frame marked as busy
   - OS suspends the current process
   - Context switch takes place
6. Once the page frame is clean
   - OS looks up disk address where needed page is
   - OS schedules a disk operation
   - Faulting process still suspended
7. When disk interrupts indicates page has arrived
   - OS updates page table

Page Fault Handling

8. Faulting instruction is backed up to its original state before page fault and PC is reset to point to it.
9. Process is scheduled for execution and OS returns to the assembly routine.
10. The routine reloads registers and other state information and returns to user space.

Interesting Scenario: Virtual Memory & I/O Interaction

- Process issues a syscall to read a file into a buffer
- Process suspended while waiting for I/O
- New process starts executing
- This other process gets a page fault
- If paging algorithm is global there is a chance the page containing the buffer be removed from memory.
- The I/O operation of the first process will write some data into the buffer and some other on the just-loaded page!

Interesting Scenario: Virtual Memory & I/O Interaction

- Process issues a syscall to read a file into a buffer
- Process suspended while waiting for I/O
- New process starts executing
- This other process gets a page fault
- If paging algorithm is global there is a chance the page containing the buffer be removed from memory.
- The I/O operation of the first process will write some data into the buffer and some other on the just-loaded page!

One solution: Locking (pinning) pages engaged in I/O so that they will not be removed.

Backing Store (i.e. the disk)

- Swap area: must not be accessed by user.
- Associated with each process the disk address of its swap area; stored in the process table
- Before process starts swap area must be initialized
  - One way: copy all process image into swap area [static swap area]
  - Another way: don't copy anything and let the process swap out [dynamic]
- Instead of disk partition, one or more preallocated files within the normal file system can be used [Windows uses this approach.]

Conclusions

- Memory management has two goals:
  - Protecting processes from each other.
  - Give each process the illusion that it has the whole memory as a huge continuous address space for itself.
- Technique used to do that: paging