

# Memory

## Memory Learning Objectives

- Overlays & Fixed Partitions
  - Internal Fragmentation
  - Why separate queues are inefficient
- Virtual Addresses
  - Relocation using Base Register
- Dynamic contiguous allocation
  - Bitmaps versus linked lists
  - Allocation Schemes (Best, First, Worst, Next)

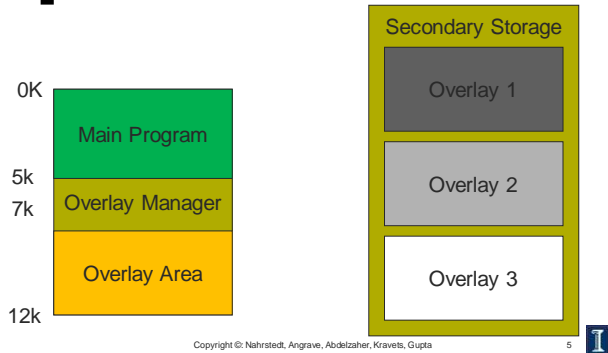
## Memory Management

- The ideal world has memory that is
  - Very large
  - Very fast
  - Very cheap
  - Non-volatile (doesn't go away when power is turned off)
- The real world has memory that is
  - Very large
  - Very fast
  - Affordable!
  - ⇒ Pick any two...
- Memory management goal
  - Make the real world look as much like the ideal world as possible

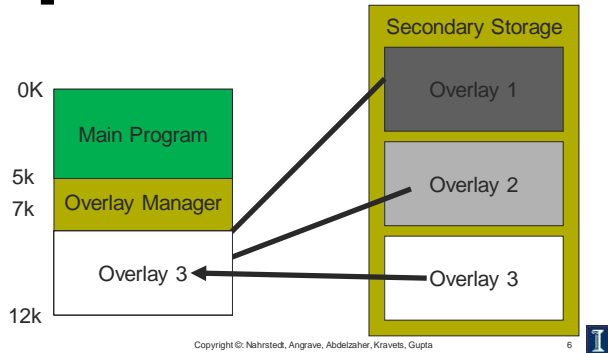
## Memory Management

- Goal
  - Layout the programs in memory as needed
- Potential issues
  - Utilization
  - Protection
  - Flexibility

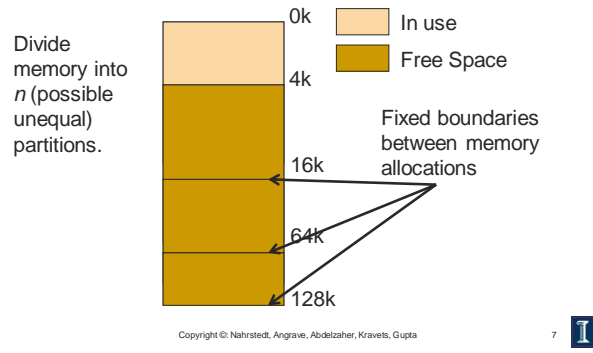
# [Overlays]



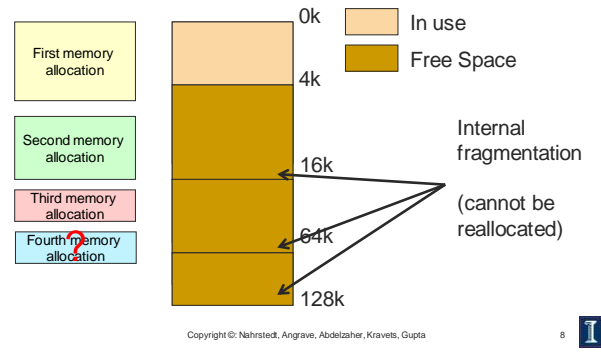
# [Overlays]



# [Multiple Fixed Partitions]

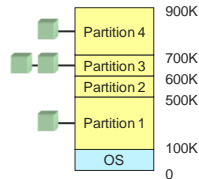


# [Multiple Fixed Partitions]



## Fixed Partition Implementation

- Separate input queue for each partition
  - Put incoming jobs into separate partition queues
    - Requires sorting the incoming jobs and putting them into separate queues
  - Inefficient utilization of memory
    - Small jobs still wait



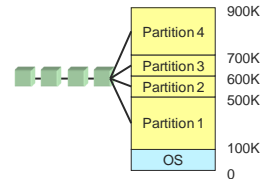
Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

9



## Fixed Partition Implementation

- Solution: One single input queue for all partitions.
  - Allocate a partition where the job fits and use...
    - Best Fit
    - Worst Fit
    - First Fit



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

10



## Virtual addresses

- "Any programming problem can be solved by adding a level of indirection."
- Logical address
  - Address generated by the CPU
  - Virtual address
- Physical address
  - Address seen by the memory unit

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

11



## Virtual addresses

- "Different jobs will run at different addresses"
  - When a program is linked, the linker must know at what address the program will begin in memory.
  - Program never sees physical address
  - Correct starting address when a program starts in memory

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

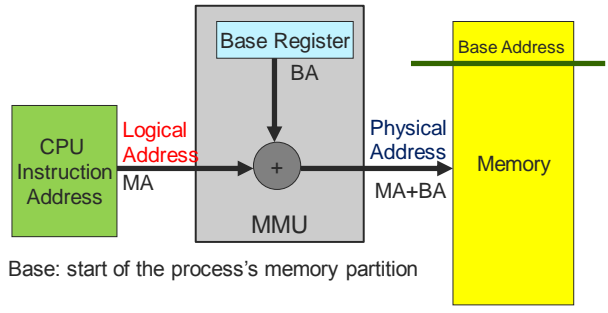
12



# Base Register

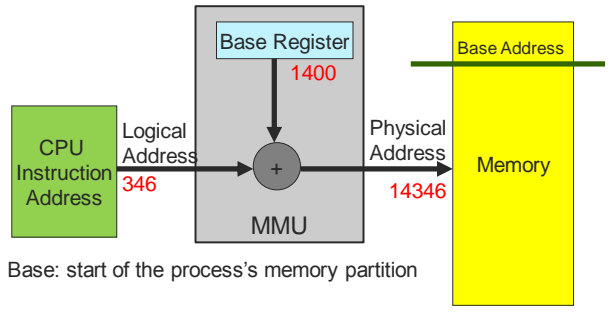
- Logical or "Virtual" addresses
  - Logical address space
  - Range: 0 to max
- Physical addresses
  - Physical address space
  - Range: R+0 to R+max for base value R
- How:
  - Memory-management unit (MMU)
    - Map virtual to physical addresses.
  - Relocation register
    - Mapping requires hardware (MMU) with the base register

# Relocation Register



Base: start of the process's memory partition

# Relocation Register



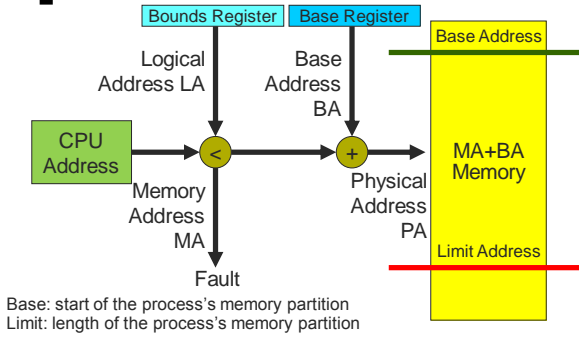
Base: start of the process's memory partition

# Protection

- Problem
  - How to prevent a malicious process from writing or jumping into other user's or OS partitions
- Solution
  - Base bounds registers



## Base Bounds Registers



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

17

## Memory Management

- Goal
  - Keep track of free / allocated memory regions
- Mechanisms
  - Bitmaps
    - One bit in map corresponds to a fixed-size region of memory
  - Linked lists
    - Each entry in the list corresponds to a contiguous region of memory

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

18

## Bit Maps and Linked Lists

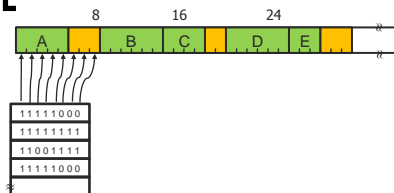


- Part of memory with 5 processes, 3 holes
  - Tick marks show allocation units
  - Green regions are free

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

19

## Bit Maps and Linked Lists

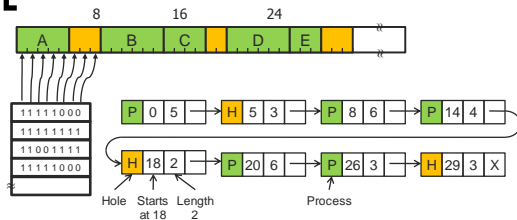


- Part of memory with 5 processes, 3 holes
  - Tick marks show allocation units
  - Green regions are free

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

20

## Bit Maps and Linked Lists



- Part of memory with 5 processes, 3 holes
  - Tick marks show allocation units
  - Green regions are free

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

21



## Partition Allocation schemes

- Bitmap vs. link list
  - Which one occupies more space?
    - Depends on the individual memory allocation scenario
    - In most cases, bitmap usually occupies more space.
  - Which one is faster reclaim freed space?
    - On average, bitmap is faster because it just needs to set the corresponding bits
  - Which one is faster to find a free hole?
    - On average, a link list is faster because we can link all free holes together

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

22



## Storage Placement Strategies

- Best fit
  - Use the hole whose size is equal to the need, or if none is equal, the hole that is larger but closest in size.
  - Rationale?
- First fit
  - Use the first available hole whose size is sufficient to meet the need
  - Rationale?
- Worst fit
  - Use the largest available hole
  - Rationale?

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

23



## Example

- Consider a swapping system in which memory consists of the following hole sizes in memory order:
  - 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K.
  - Which hole is taken for successive segment requests of:
    - 12K
    - 10K
    - 9K

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

24



## [ Example ]

- Consider a swapping system in which memory consists of the following hole sizes in memory order:
  - 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K.
  - Which hole is taken for successive segment requests of:
    - 12K
    - 10K
    - 9K

First fit: 20K, 10K, 18K.	Best fit: 12K, 10K, 9K.	Worst fit: 20K, 18K, and 15K.
---------------------------------	-------------------------------	-------------------------------------

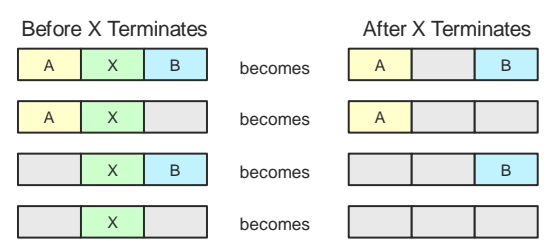
## [ Storage Placement Strategies ]

- Best fit
  - Produces the smallest leftover hole
  - Creates small holes that cannot be used
- Worst Fit
  - Produces the largest leftover hole
  - Difficult to run large programs
- First Fit
  - Creates average size holes
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

## [ Fragmentation ]

- External Fragmentation
  - Memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation
  - Allocated memory may be slightly larger than requested memory
  - The size difference is memory internal to a partition, but not being used

## [ Memory Management: Process Termination ]



- Four neighbor combinations for the termination of process X

## How Bad Is Fragmentation?

- Statistical arguments - Random sizes
- First-fit
  - Given N allocated blocks
  - $0.5 * N$  blocks will be lost because of fragmentation
- Known as 50% RULE

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

29



## Compaction

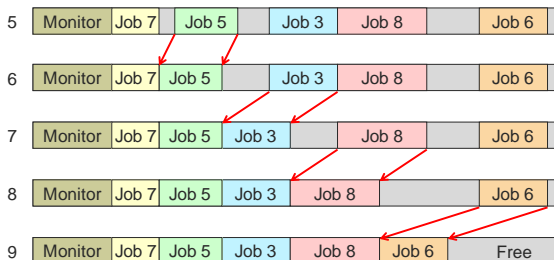
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

30



## Solve Fragmentation w. Compaction



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

31



## Storage Management Problems

- Fixed partitions suffer from
  - Internal fragmentation
- Variable partitions suffer from
  - External fragmentation
- Compaction suffers from
  - Overhead

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

32



## [ Question ]

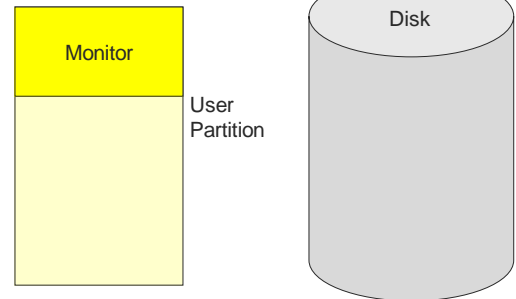
- What if there are more processes than what could fit into the memory?
- Swapping
- Memory allocation changes as
  - Processes come into memory
  - Processes leave memory
    - Swapped to disk
    - Complete execution

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

33



## [ Swapping ]

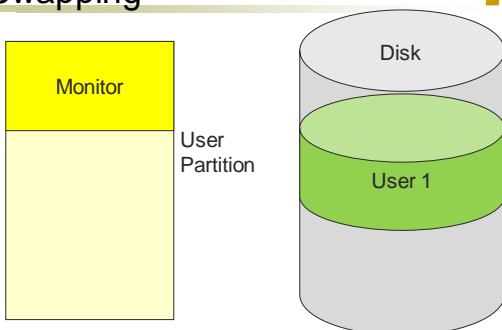


Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

34



## [ Swapping ]

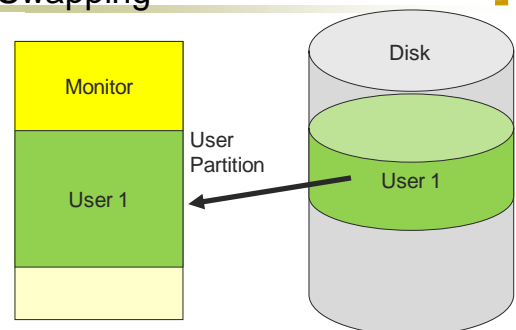


Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

35



## [ Swapping ]

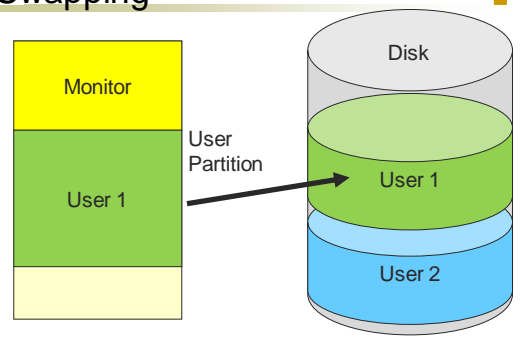


Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

36



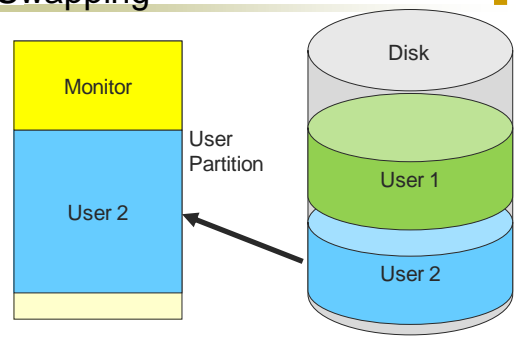
# [ Swapping ]



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

37

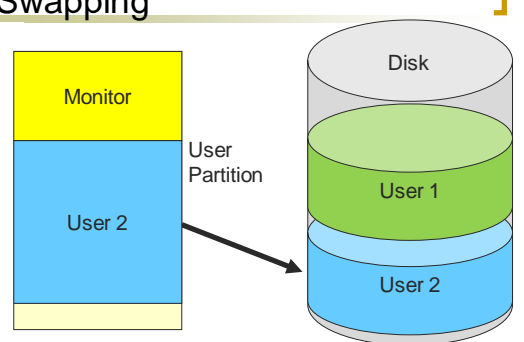
# [ Swapping ]



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

38

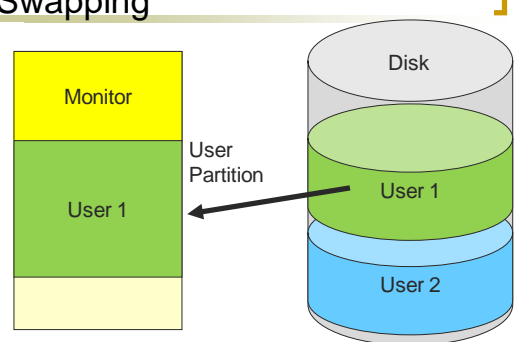
# [ Swapping ]



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

39

# [ Swapping ]



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

40

## Limitations of Swapping

- Problems with swapping
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
    - External fragmentation
      - Lots of small free areas
    - Compaction
      - Reassemble larger free areas
  - Processes are either in memory or on disk: half and half doesn't do any good

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

41



## Virtual memory

- Basic idea
  - Allow the OS to hand out more memory than exists on the system
  - Keep recently used stuff in physical memory
  - Move less recently used stuff to disk
  - Keep all of this hidden from processes
- Process view
  - Processes still see an address space from 0 – max address
  - Movement of information to and from disk handled by the OS without process help

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

42



## Benefits of Virtual Memory

- Especially helpful in multiprogrammed system
  - CPU schedules process B while process A waits for its memory to be retrieved from disk
- Use secondary storage(\$)
  - Extend DRAM(\$\$\$) with reasonable performance
- Protection
  - Programs do not step over each other

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

43



## Benefits of Virtual Memory

- Convenience
  - Flat address space
  - Programs have the same view of the world
  - Load and store cached virtual memory without user program intervention
- Reduce fragmentation
  - Make cacheable units all the same size (page)

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

44



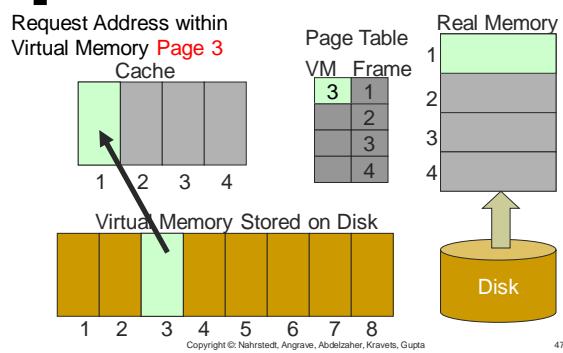
# [ Paging ]

- Paging is how an OS achieves VM
- Goal
  - Provide user with virtual memory that is as big as user needs
- Implementation
  - Store virtual memory on disk
  - Cache parts of virtual memory being used in real memory
  - Load and store cached virtual memory without user program intervention

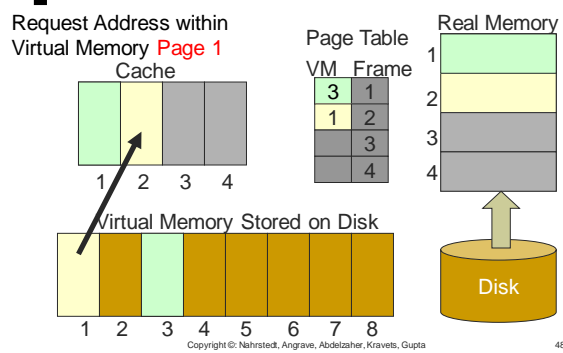
# [ Page Faults ]

- What happens when a program accesses a virtual page that is not mapped into any physical page?
  - Hardware triggers a page fault
- Page fault handler
  - Find any available free physical page
  - If none, evict some resident page to disk
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table

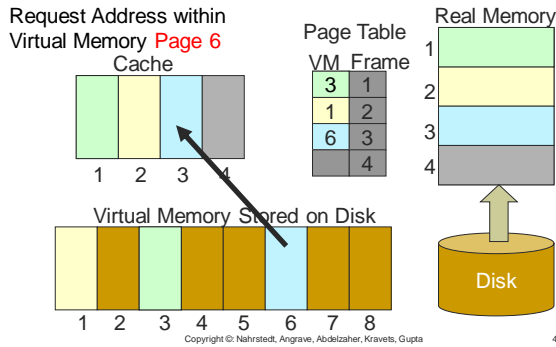
# [ Paging Request ]



# [ Paging Request ]

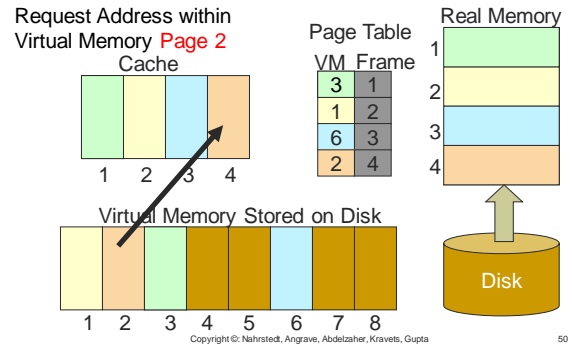


## [ Paging Request ]



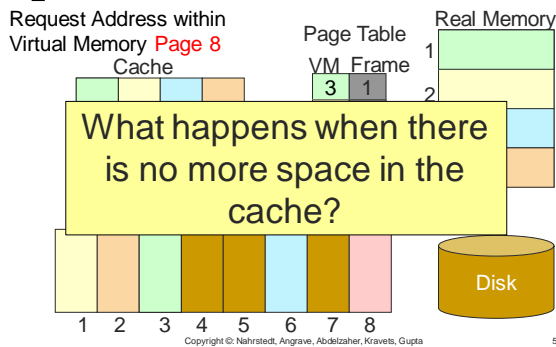
49

## [ Paging Request ]



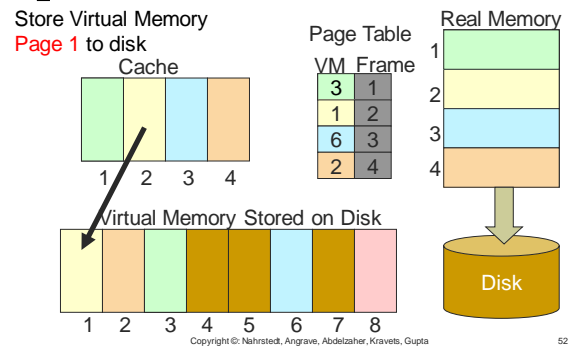
50

## [ Paging Request ]



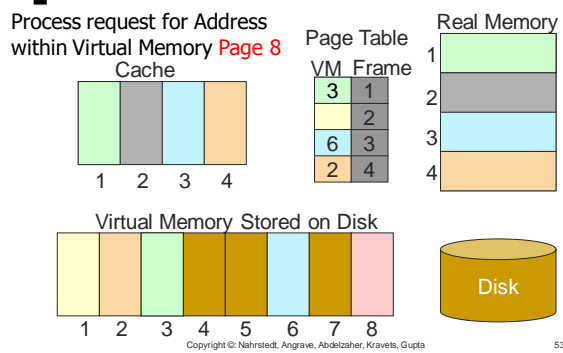
51

## [ Paging Request ]

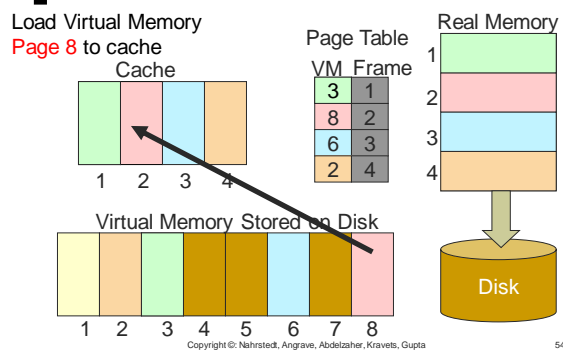


52

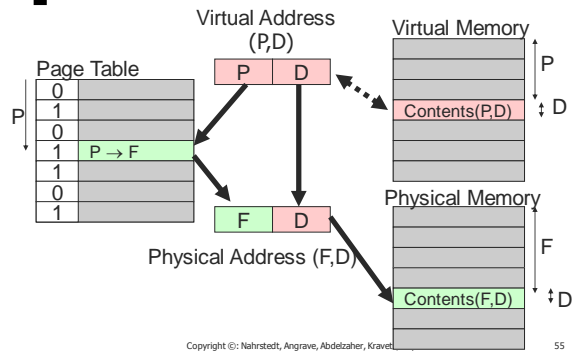
# [ Paging Request ]



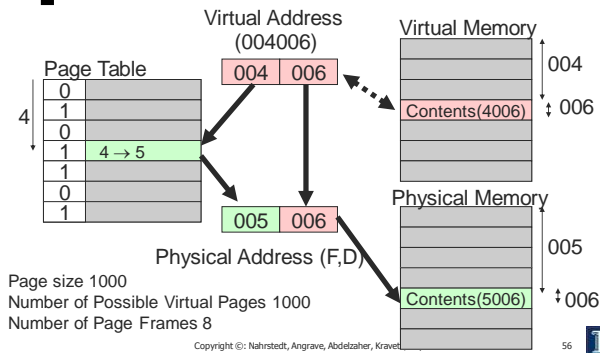
# [ Paging Request ]



# [ Page Mapping Hardware ]



# [ Page Mapping Hardware ]



## [ Paging Issues ]

- Page size
  - Typically  $2^n$ 
    - usually 512, 1k, 2k, 4k, or 8k
    - e.g. 32 bit VM address may have  $2^{20}$  (1 meg) pages with 4k ( $2^{12}$ ) bytes per page
  - $2^{20}$  (1 meg) 32 bit page entries take  $2^{22}$  bytes (4 meg)
  - Page frames must map into real memory

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

57



## [ Paging Issues ]

- Physical memory size: 32 MB ( $2^{25}$ )
  - Page size 4K bytes
  - How many pages?
    - $2^{13}$
- Page Table base register must be changed for context switch
  - Why?
    - Different page table
- NO external fragmentation
- Internal fragmentation on last page ONLY

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

58



## [ Discussion ]

- How can paging be made faster?
- Is one level of paging sufficient?
- Sharing and protections?

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

59



## [ Paging - Caching the Page Table ]

- Cache page table in registers
- Keep page table in memory
  - Location given by a *page table base register*
- Page table base register changed at context switch time

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

60



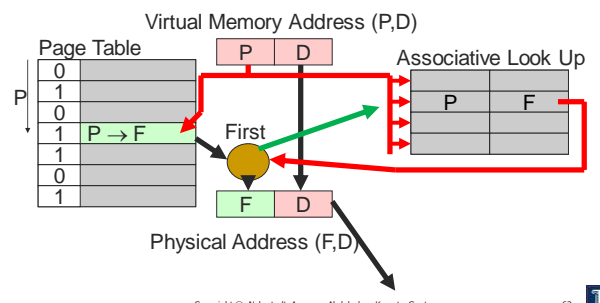
# Paging Implementation Issues

- Caching scheme
  - Associative registers, look-aside memory or content-addressable memory
  - TLB
- Page address cache (TLB) hit ratio
  - Percentage of time page found in associative memory
- Cache miss
  - If not found in associative memory, must load from page tables
  - Requires additional memory reference

Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta



# Page Mapping Hardware

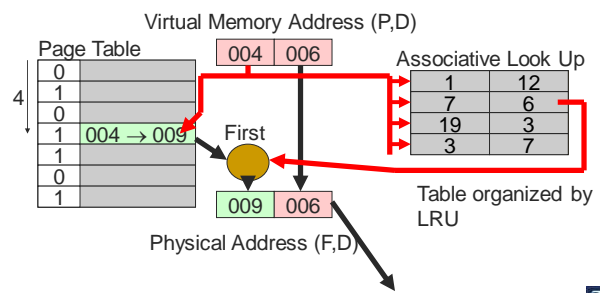


Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta



# Page Mapping Hardware

First access, retrieve page from page table

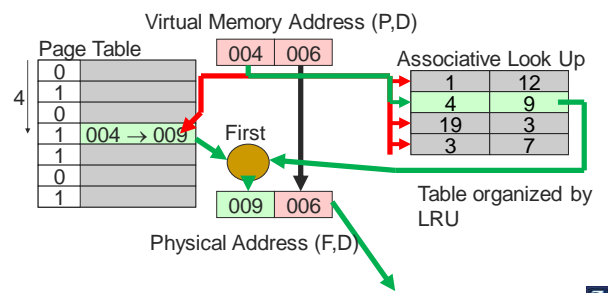


Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta



# Page Mapping Hardware

Second access, retrieve page from associative registers.



Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta

