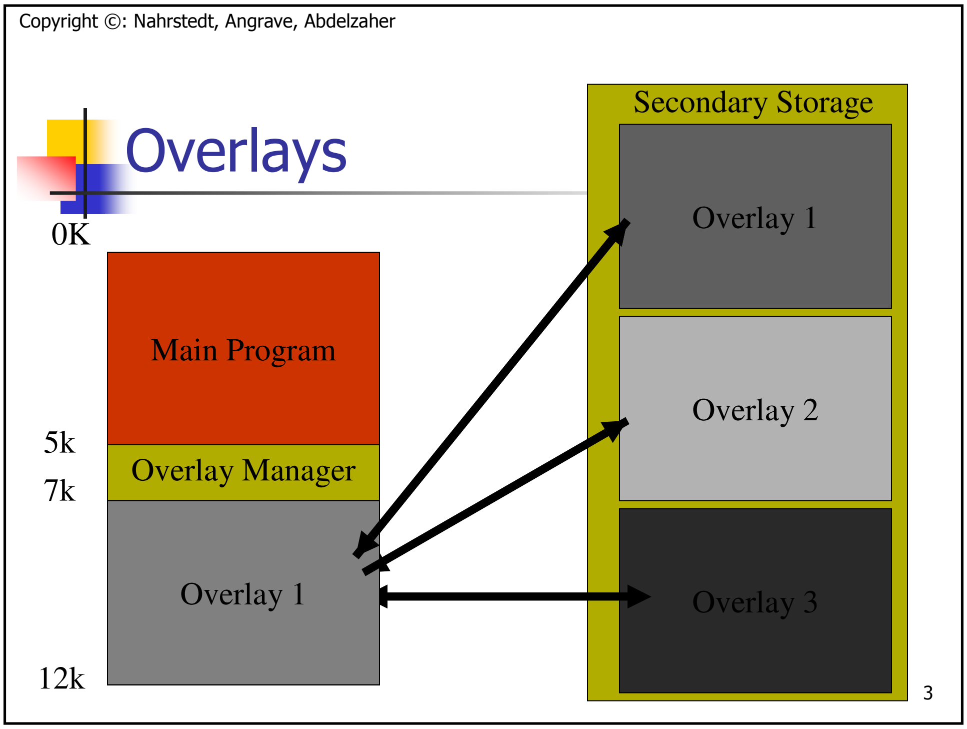
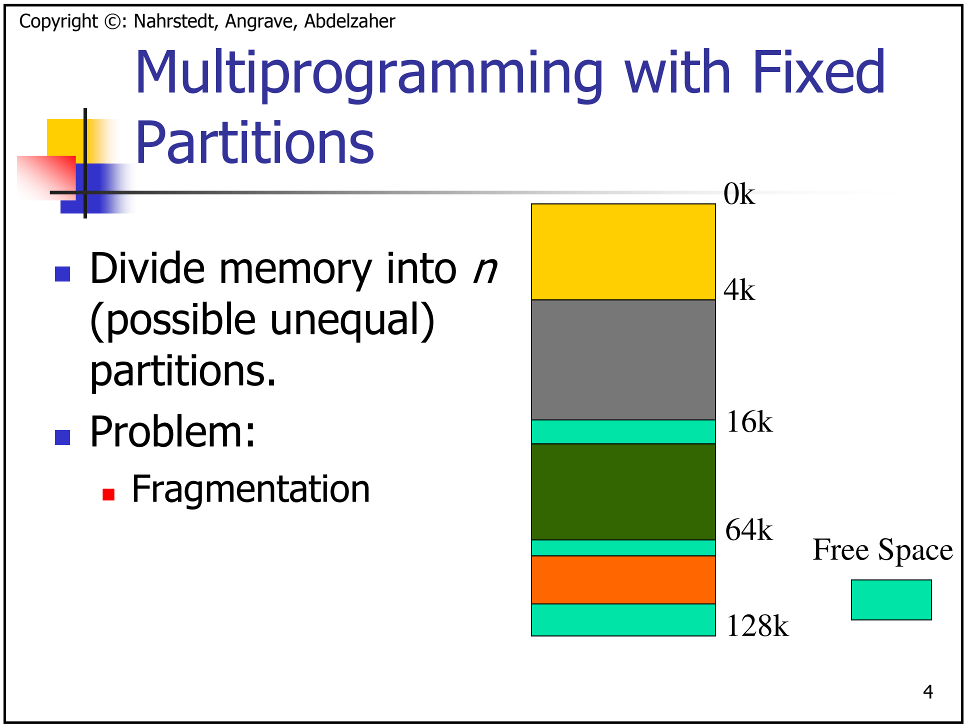
Memory

Memory Allocation

- Compile for overlays
- Compile for fixed Partitions
  - Separate queue per partition
  - Single queue
- Relocation and variable partitions
  - Dynamic contiguous allocation (bit maps versus linked lists)
- Fragmentation issues
- Swapping
- Paging



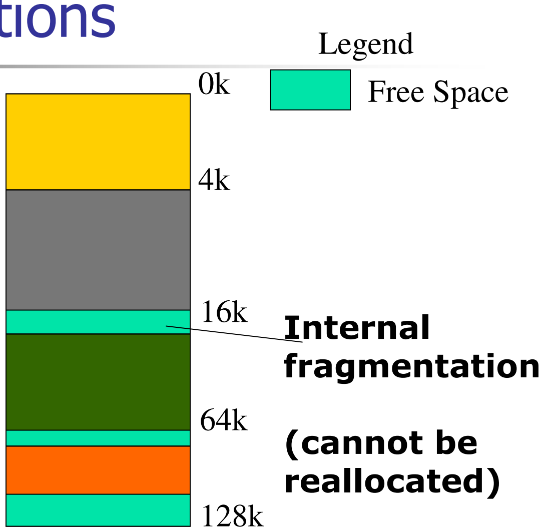
Overlays



Multiprogramming with Fixed Partitions

- Divide memory into *n* (possible unequal) partitions.
- Problem:
  - Fragmentation

Fixed Partitions



Internal fragmentation

(cannot be reallocated)

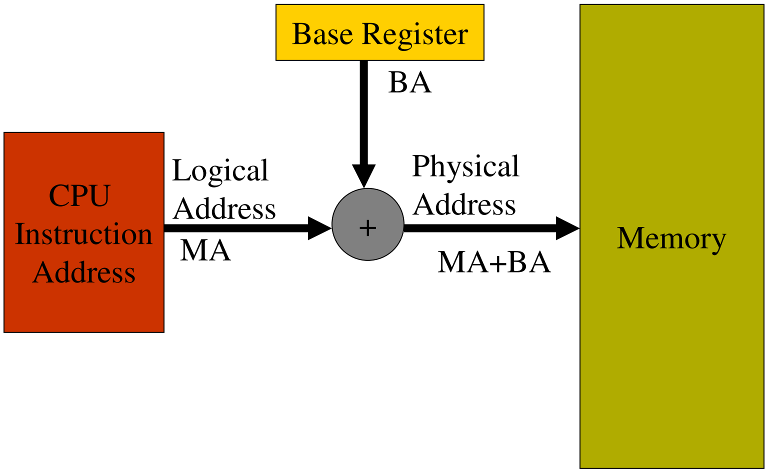
Fixed Partition Allocation Implementation Issues

- Separate input queue for each partition
  - Requires sorting the incoming jobs and putting them into separate queues
  - Inefficient utilization of memory
    - when the queue for a large partition is empty but the queue for a small partition is full. Small jobs have to wait to get into memory even though plenty of memory is free.
- One single input queue for all partitions.
  - Allocate a partition where the job fits in.
    - Best Fit
    - Worst Fit
    - First Fit

Relocation

- Correct starting address when a program starts in memory
- 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.
- Logical addresses, Virtual addresses
  - Logical address space , range (0 to max)
- Physical addresses, Physical address space
  - range (R+0 to R+max) for base value R.
- User program never sees the real physical addresses
- Memory-management unit (MMU)
  - map virtual to physical addresses.
- Relocation register
  - Mapping requires hardware (MMU) with the base register

Relocation Register


Base Register

BA

CPU Instruction Address

Logical Address

MA

+

Physical Address

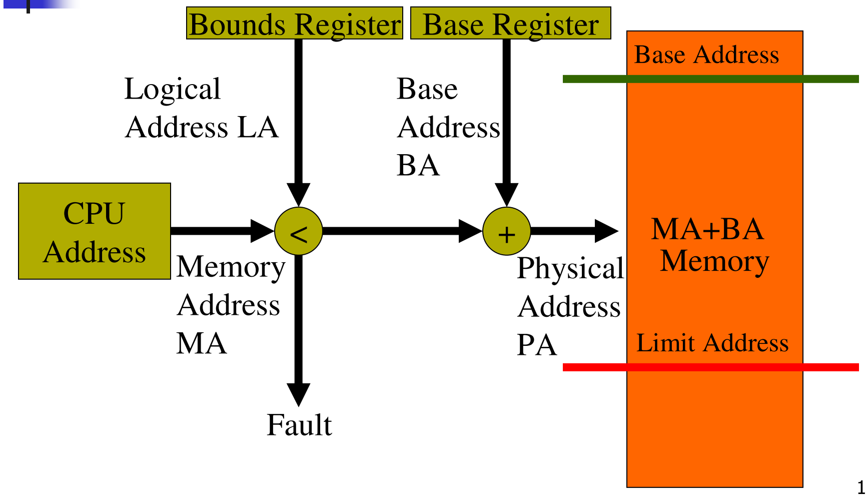
MA+BA

Memory

Question 1 - Protection

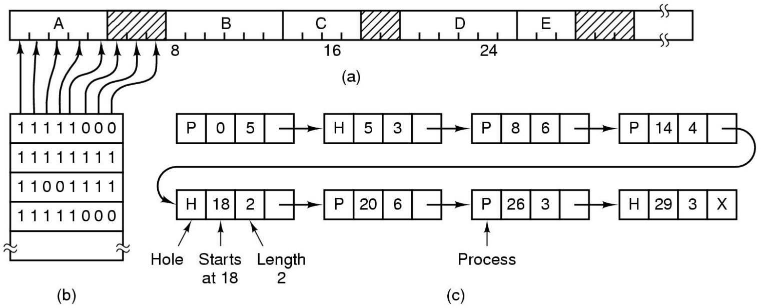
- Problem:
  - How to prevent a malicious process to write or jump into other user's or OS partitions
- Solution:
  - Base bounds registers

Base Bounds Registers



Contiguous Allocation and Variable Partitions:

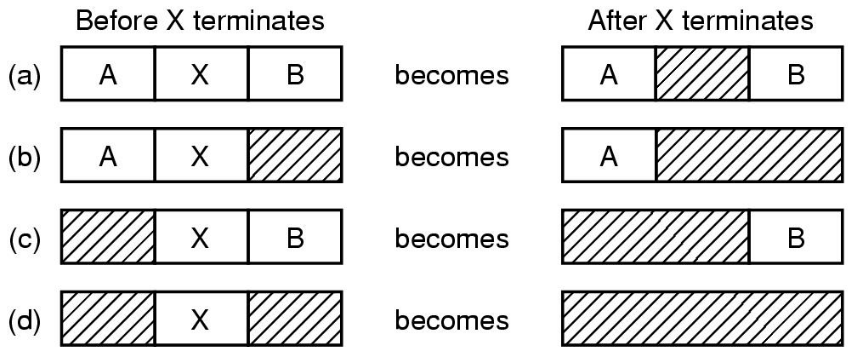
Bit Maps versus Linked Lists



- **Part of memory with 5 processes, 3 holes**
  - **tick marks show allocation units**
  - **shaded regions are free**
- **Corresponding bit map**

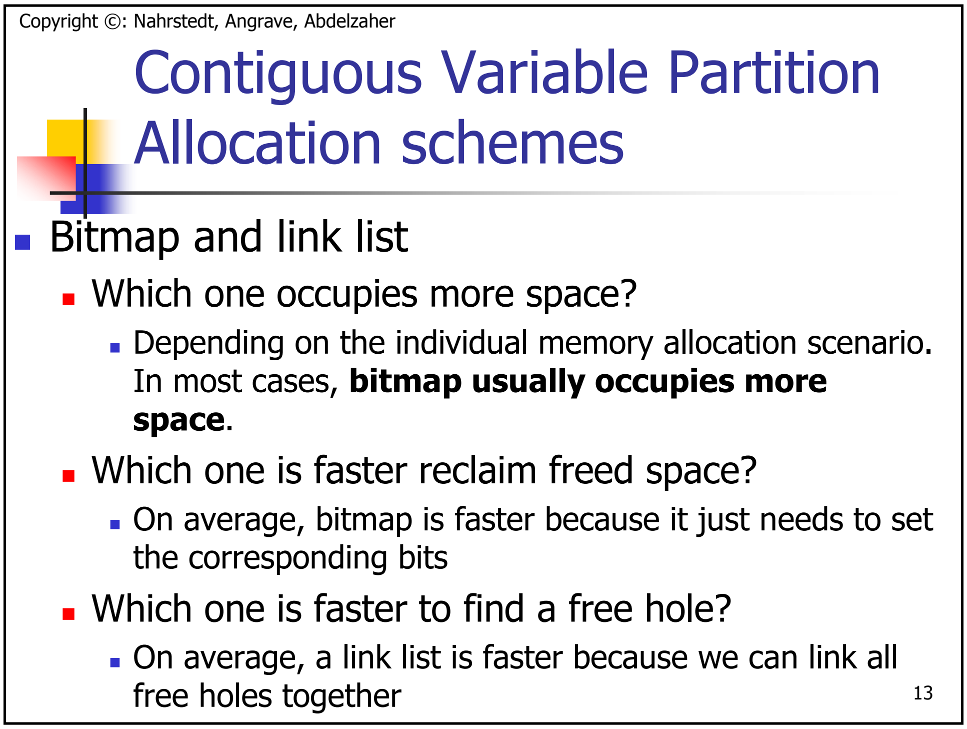
11

More on Memory Management with Linked Lists



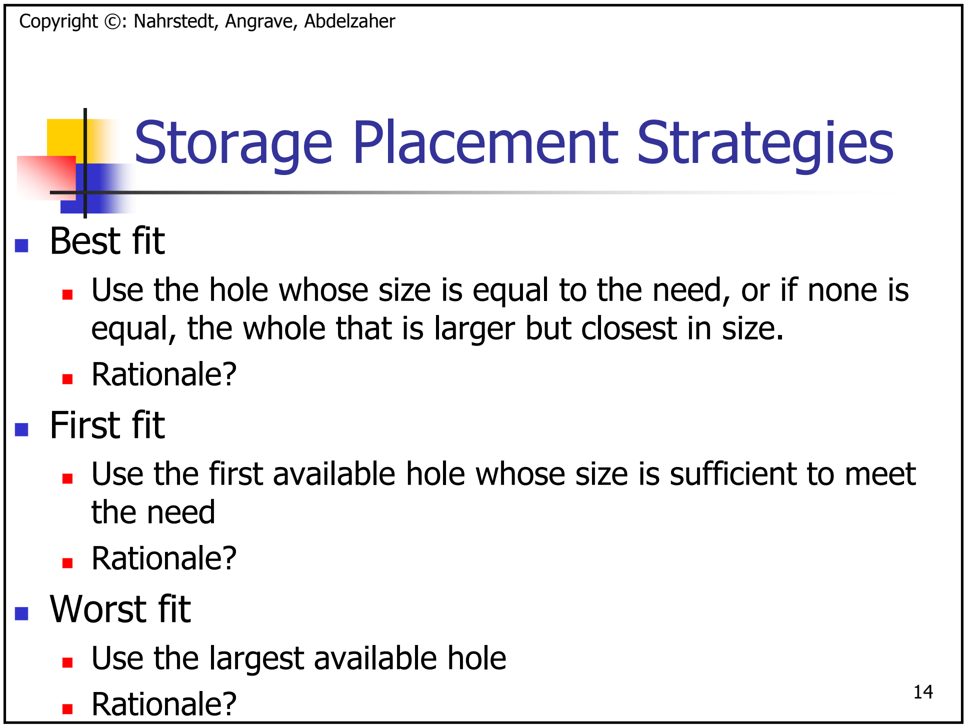
- Four neighbor combinations for the terminating process X

12



Contiguous Variable Partition Allocation schemes

- Bitmap and link list
  - Which one occupies more space?
    - Depending 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



Storage Placement Strategies

- Best fit
  - Use the hole whose size is equal to the need, or if none is equal, the whole 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?

Storage Placement Strategies

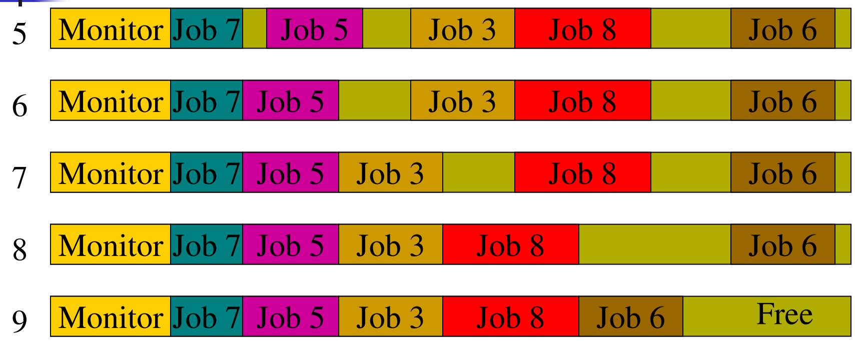
- Every placement strategy has its own problem
  - Best fit
    - Creates small holes that cant be used
  - Worst Fit
    - Gets rid of large holes making it difficult to run large programs
  - First Fit
    - Creates average size holes

How Bad Is Fragmentation?

- Statistical arguments - Random sizes
- First-fit
- Given N allocated blocks
- 0.5∗N blocks will be lost because of fragmentation

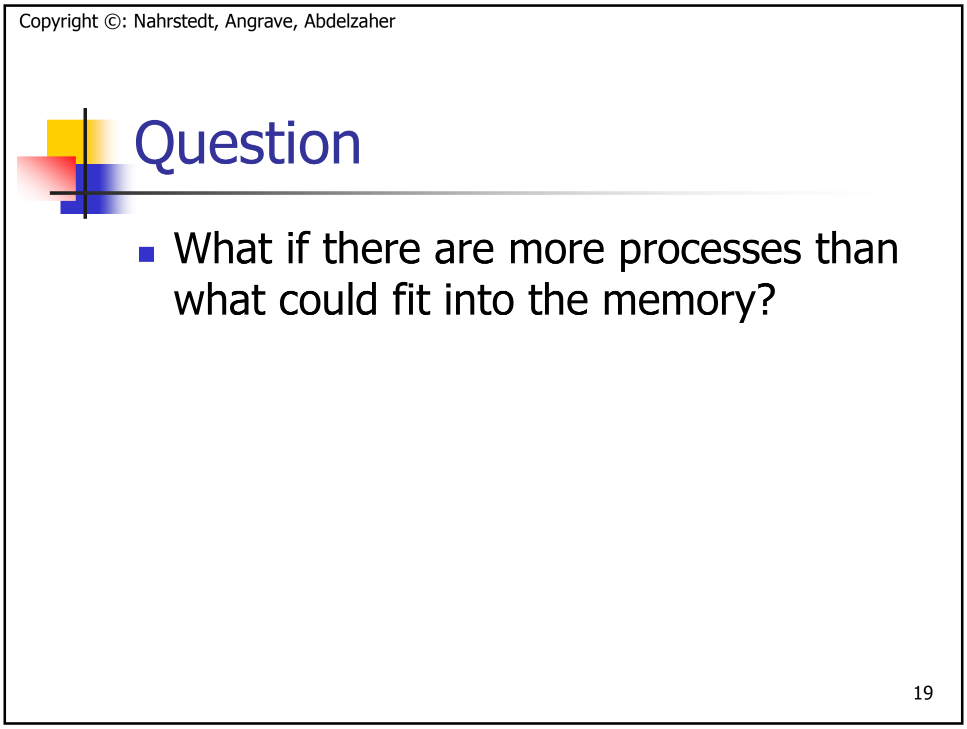
- Known as 50% RULE

Solve Fragmentation w. Compaction



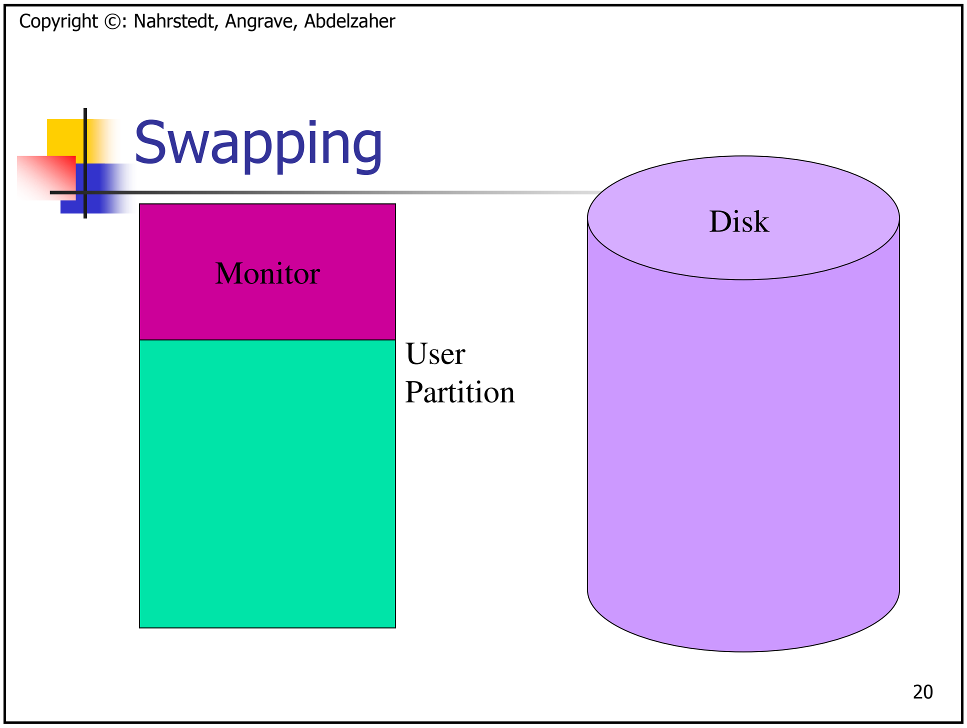
Storage Management Problems

- Fixed partitions suffer from
  - internal fragmentation
- Variable partitions suffer from
  - external fragmentation
- Compaction suffers from
  - overhead

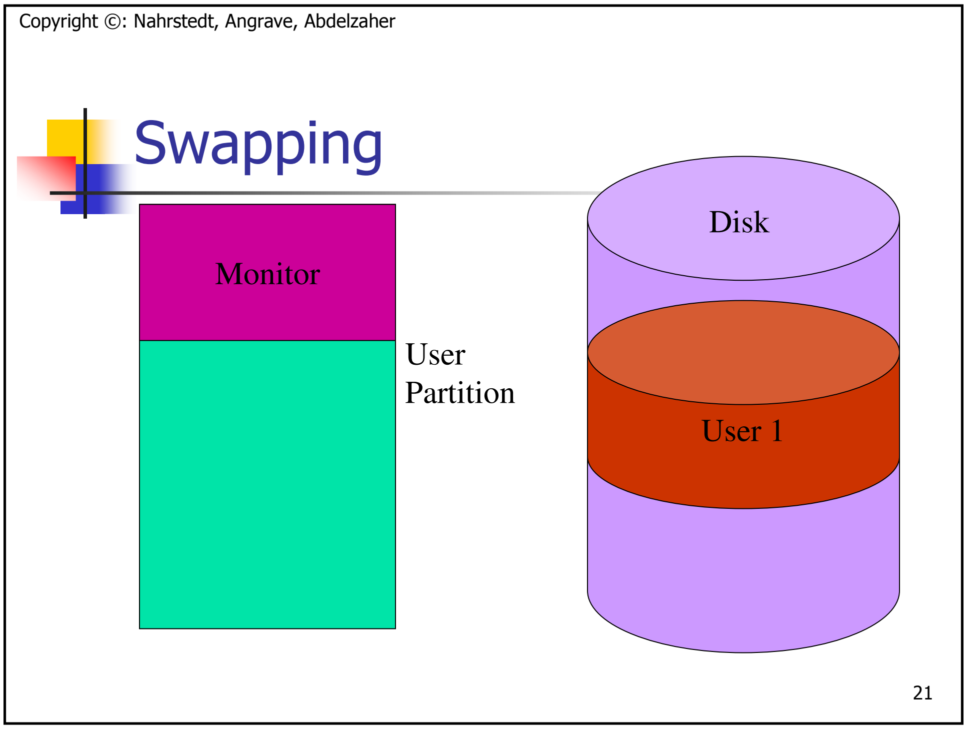


Question

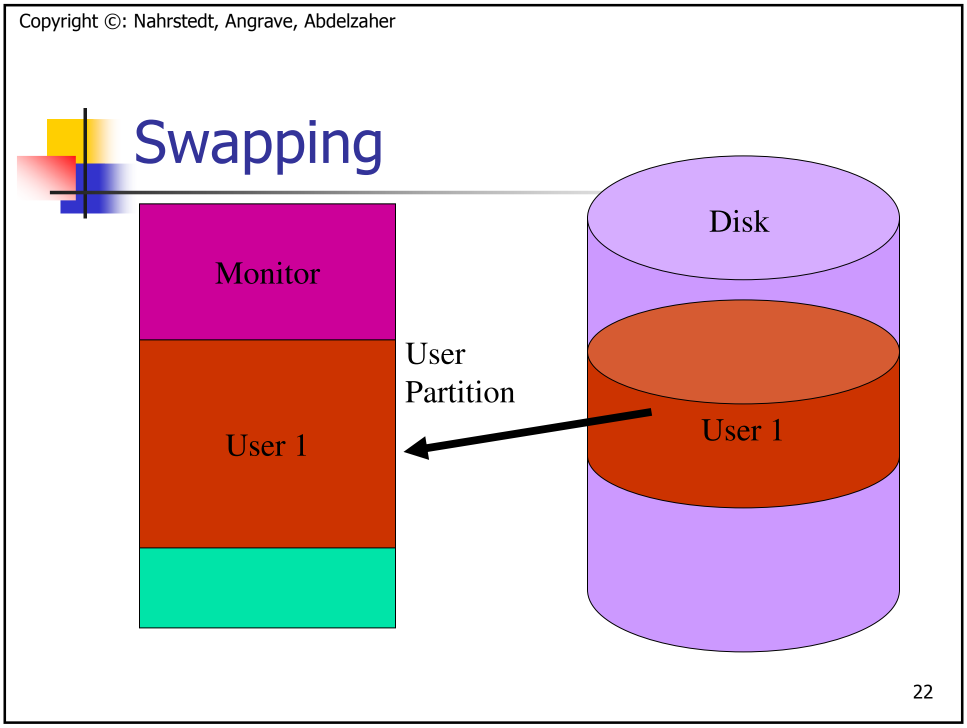
- What if there are more processes than what could fit into the memory?



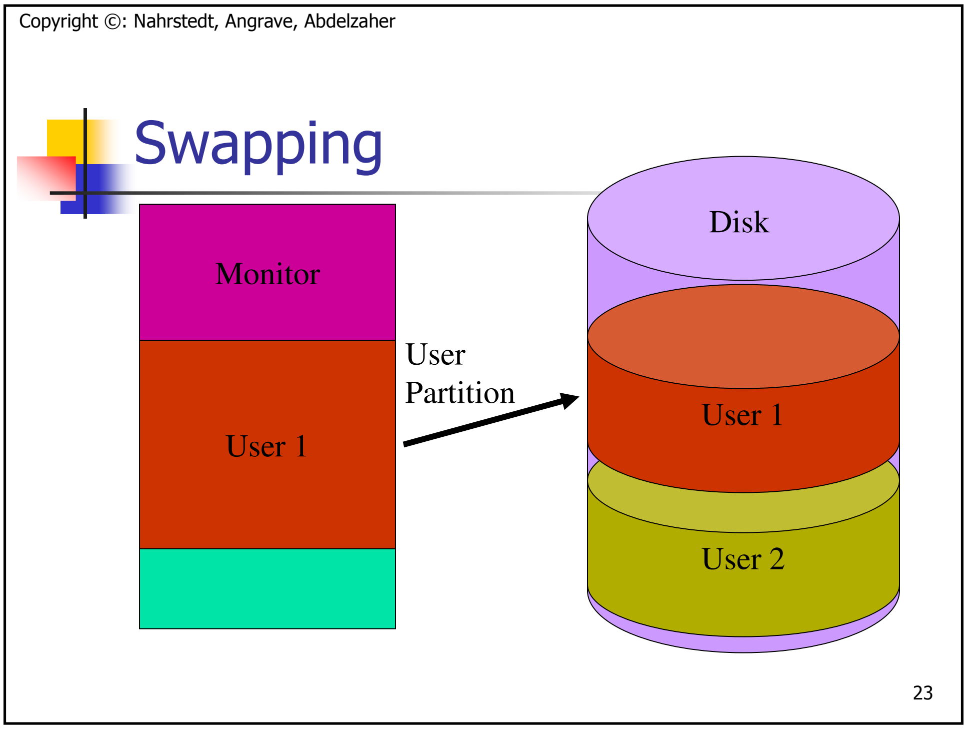
Swapping



Swapping



Swapping



Swapping

Monitor

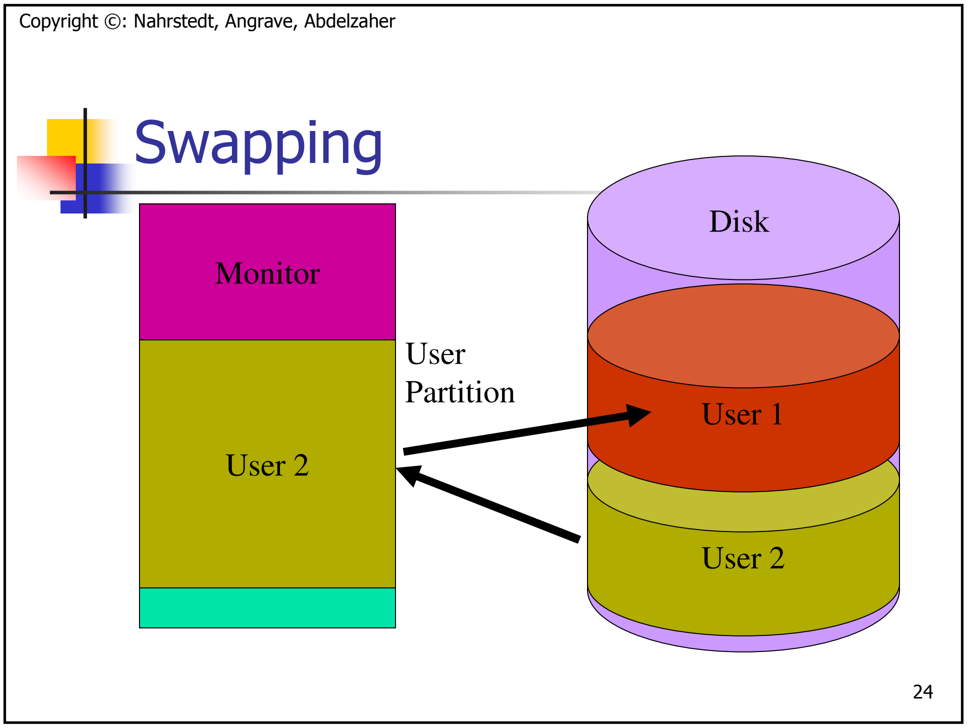
User 1

User Partition

Disk

User 1

User 2



Swapping

Monitor

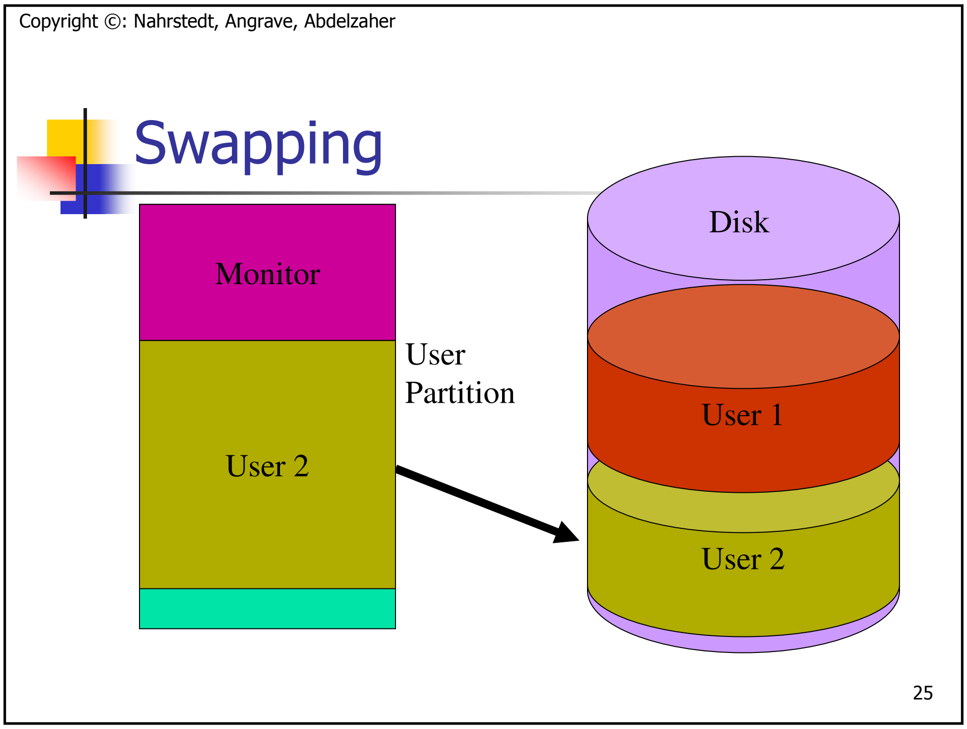
User 2

User Partition

Disk

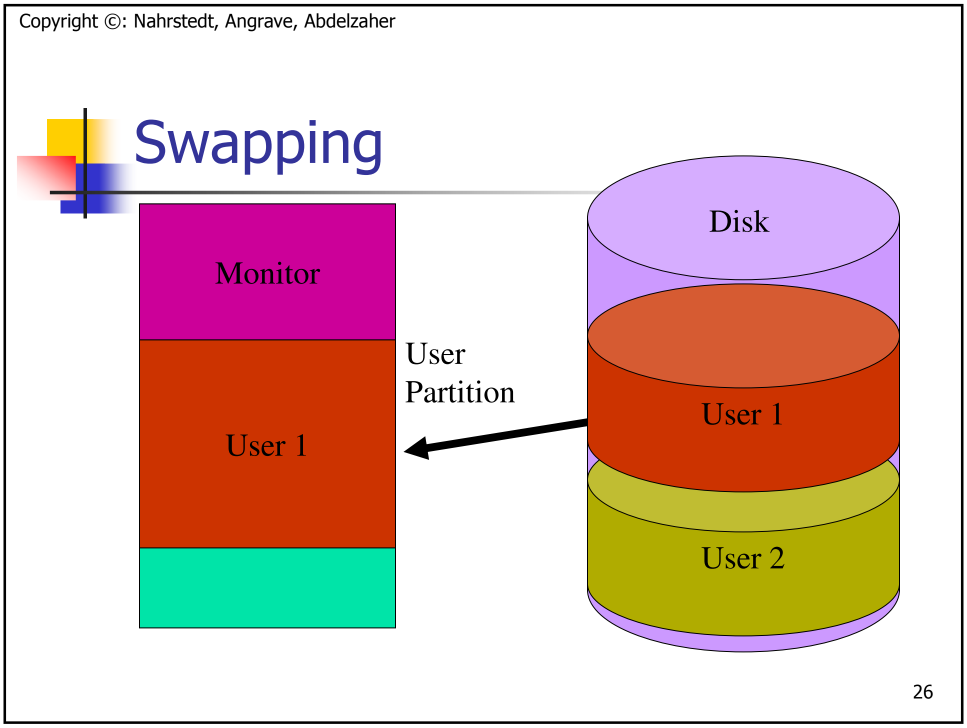
User 1

User 2



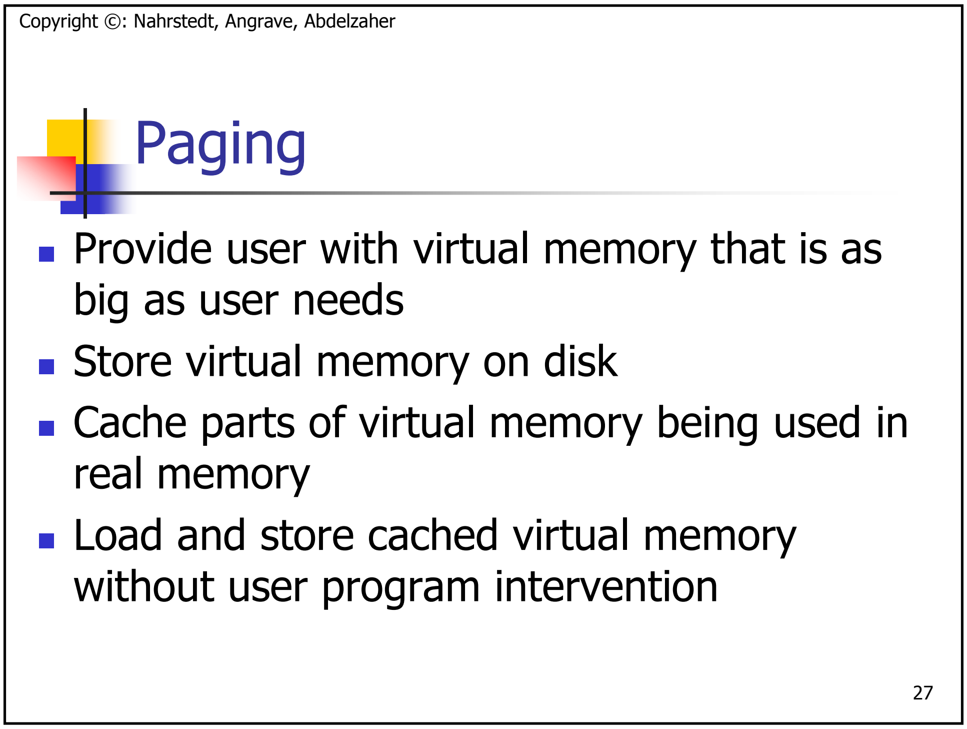
Swapping

25



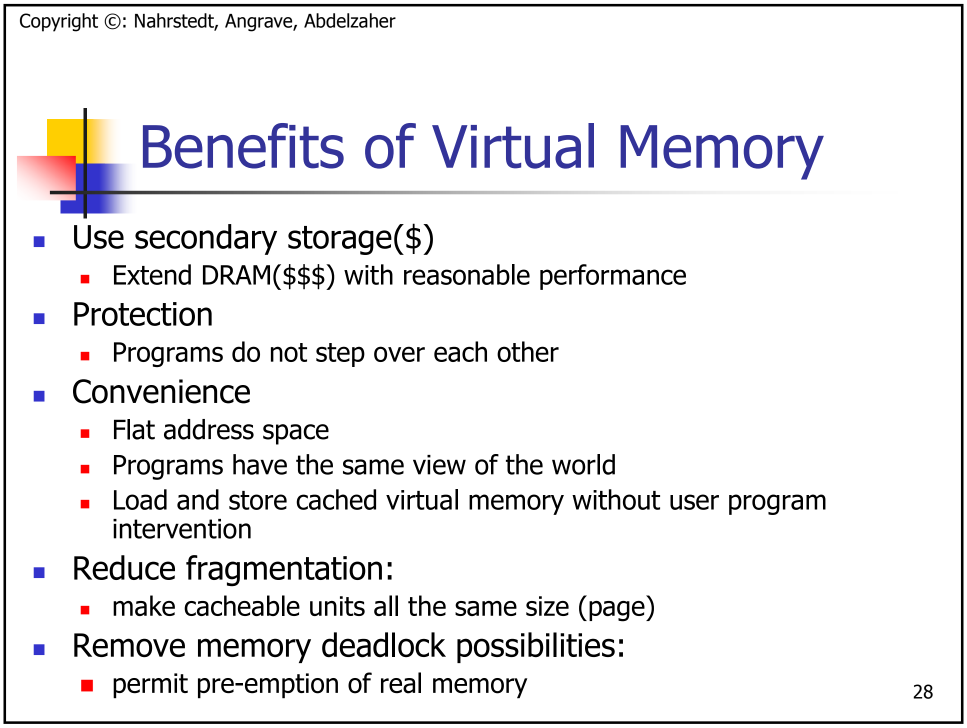
Swapping

26



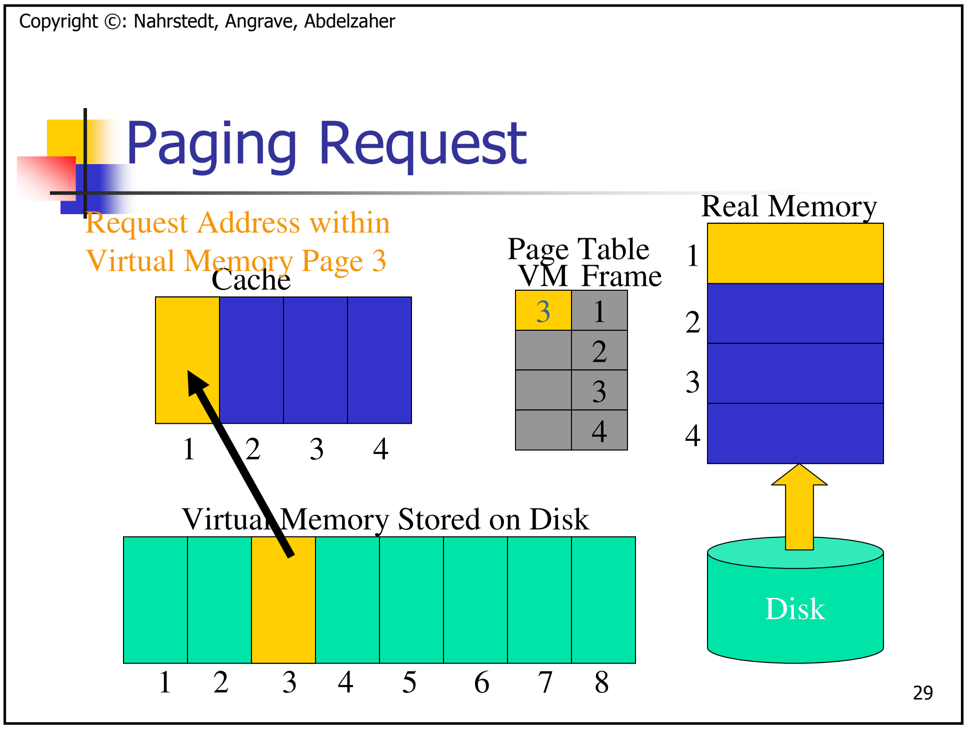
Paging

- Provide user with virtual memory that is as big as user needs
- 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



Benefits of Virtual Memory

- Use secondary storage($)
  - Extend DRAM($$$) with reasonable performance
- Protection
  - Programs do not step over each other
- 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)
- Remove memory deadlock possibilities:
  - permit pre-emption of real memory

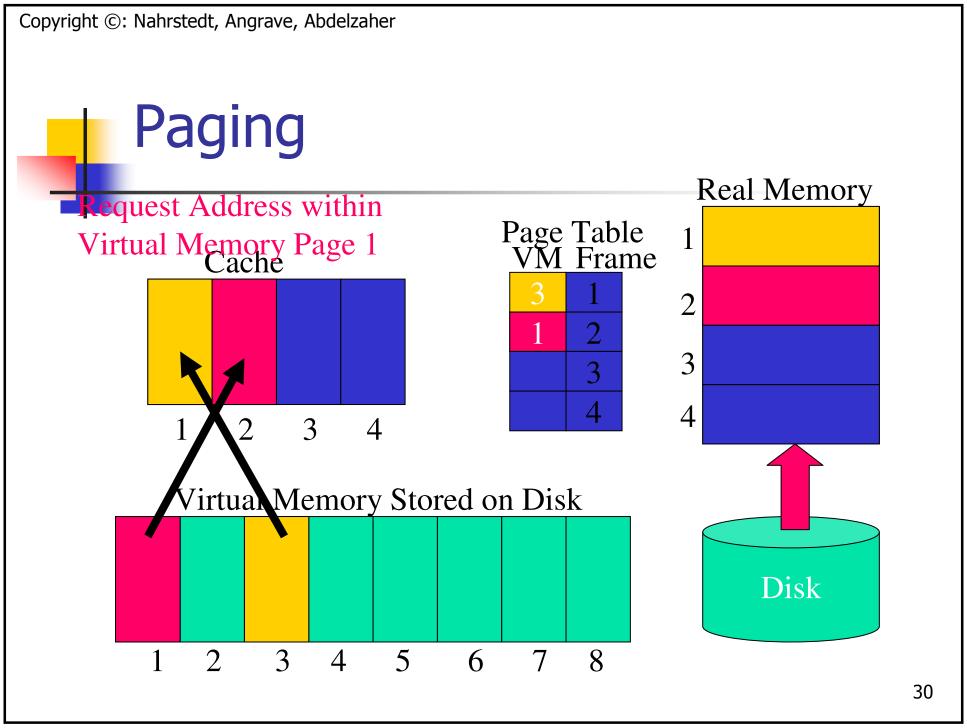


Paging Request

Request Address within Virtual Memory Page 3

Page Table

VM	Frame
3	1
	2
	3
	4

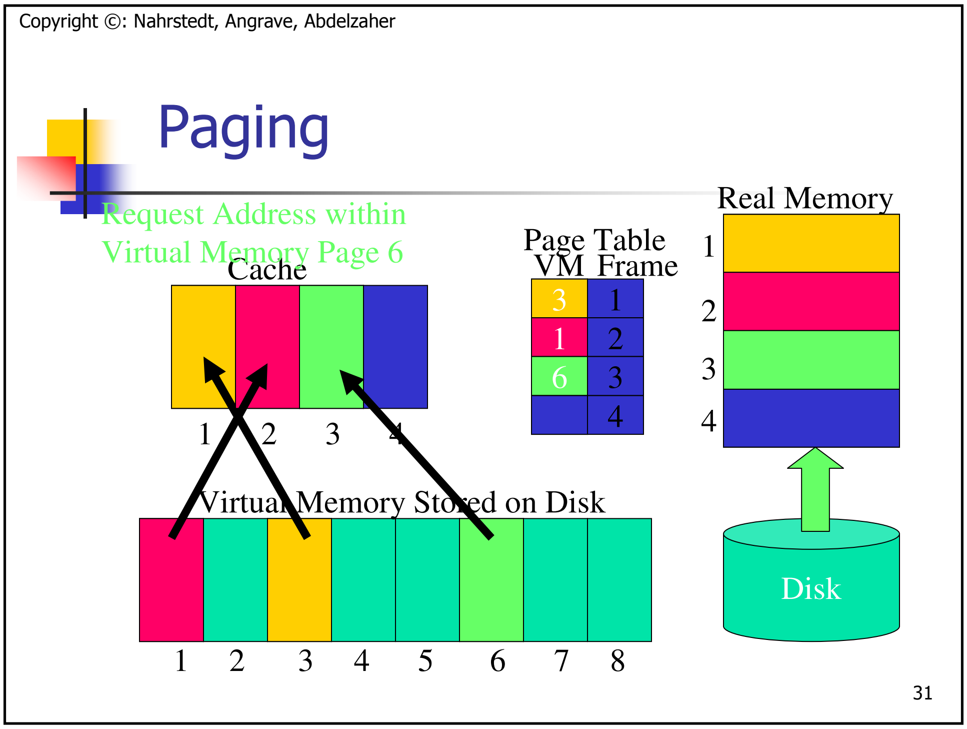


Paging

Request Address within Virtual Memory Page 1

Page Table

VM	Frame
3	1
1	2
	3
	4



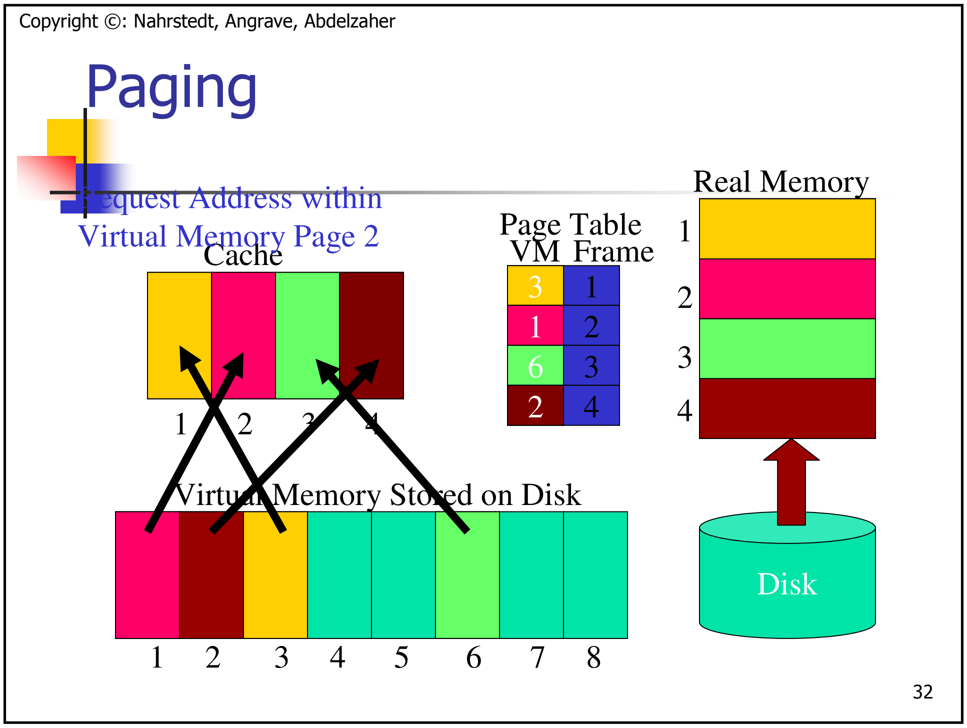
Paging

Request Address within Virtual Memory Page 6

Page Table

VM	Frame
3	1
1	2
6	3
	4

31



Paging

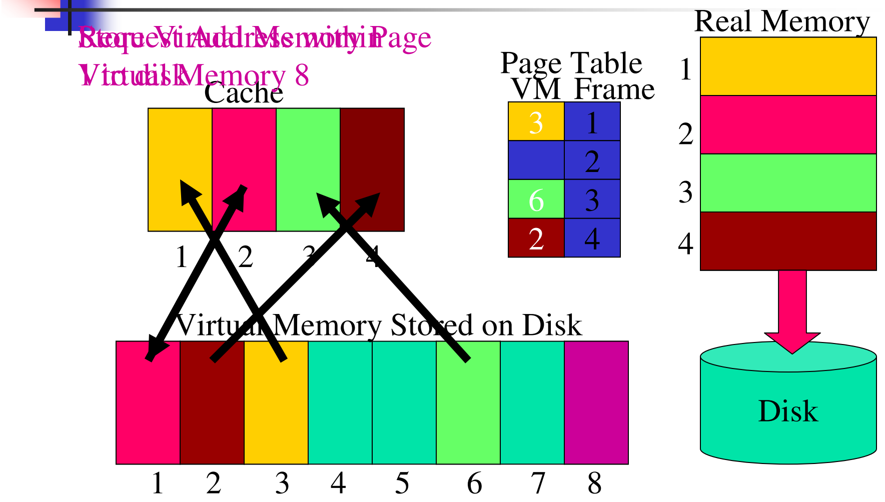
Request Address within Virtual Memory Page 2

Page Table

VM	Frame
3	1
1	2
6	3
2	4

32

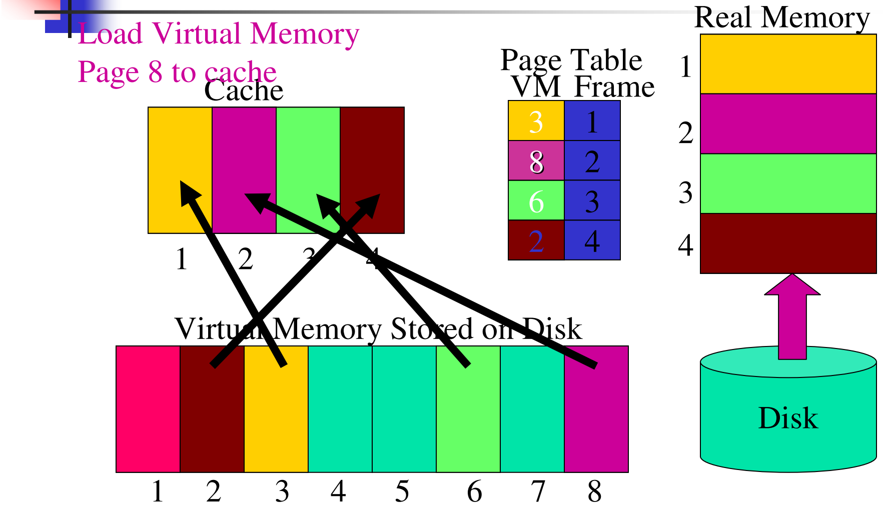
Paging



Request Address within Page
Virtual Memory 8

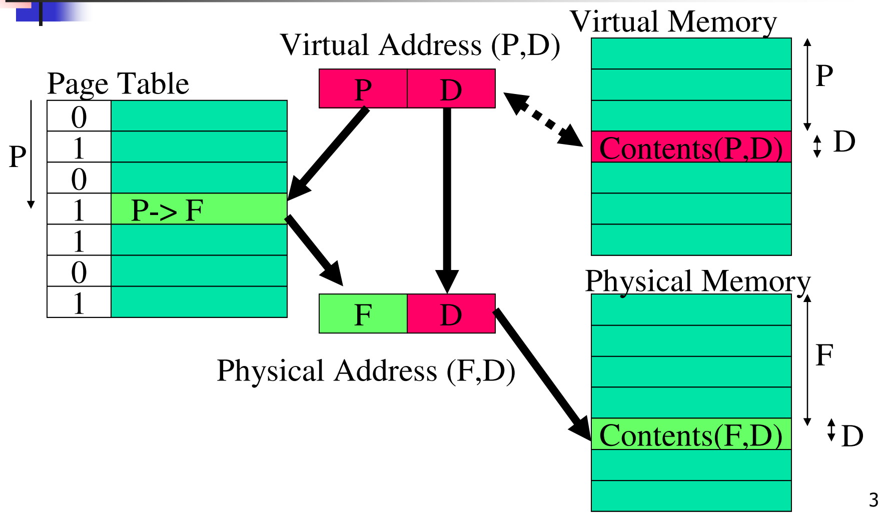
Store Virtual Memory
to disk

Paging

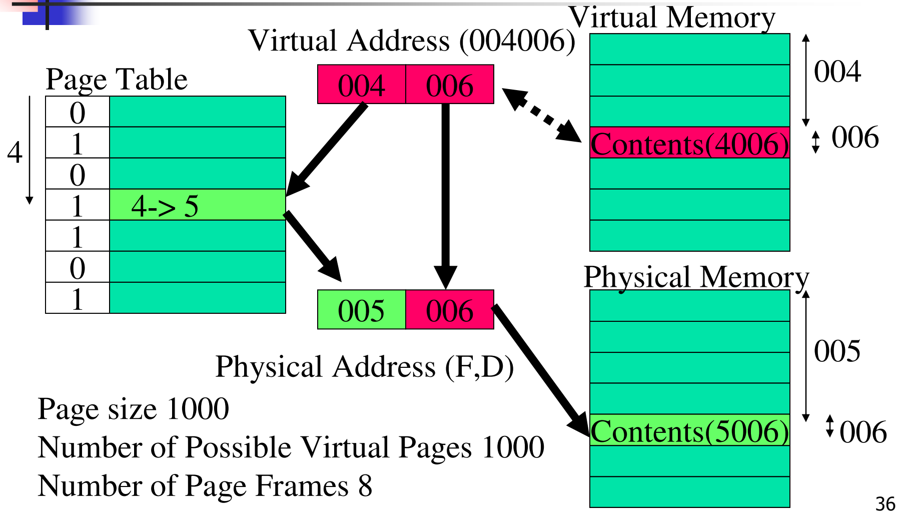


Load Virtual Memory
Page 8 to cache

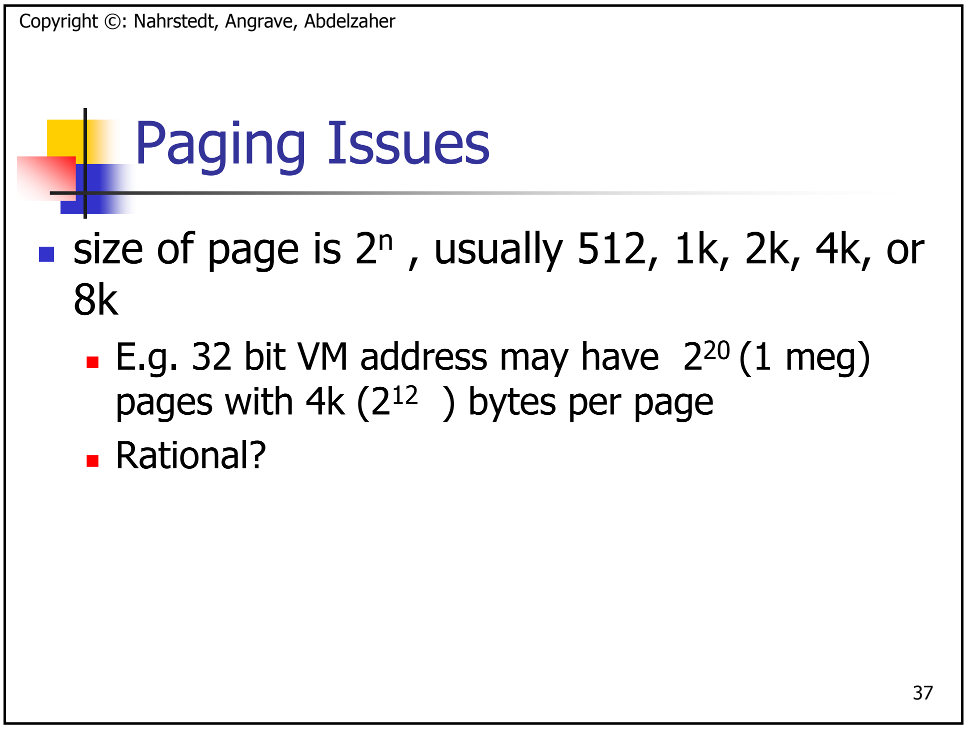
Page Mapping Hardware



Page Mapping Hardware

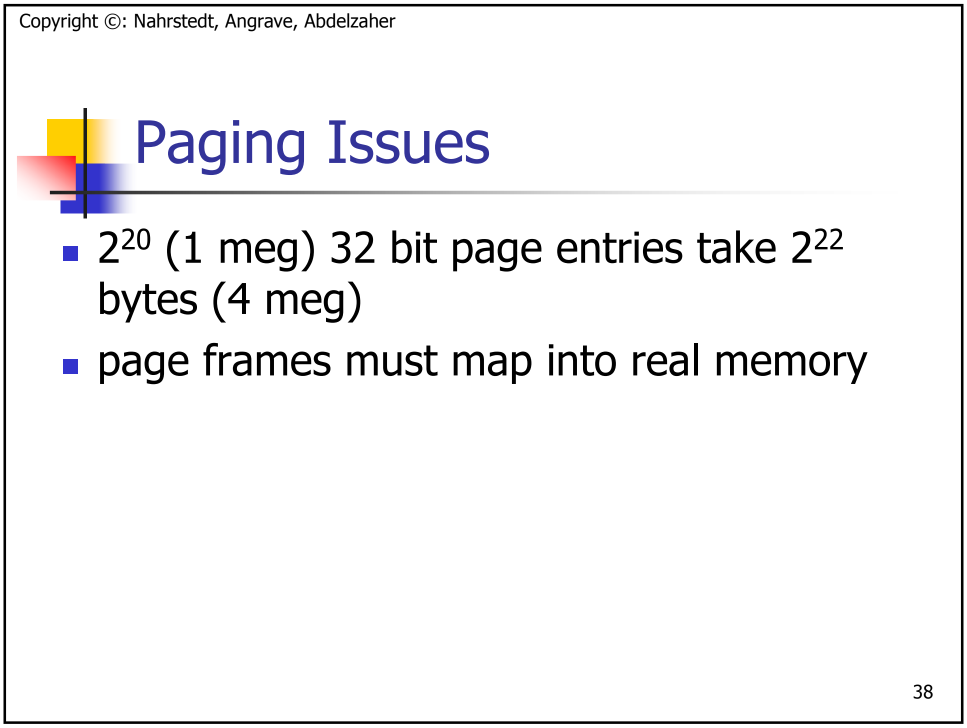


Page size 1000
Number of Possible Virtual Pages 1000
Number of Page Frames 8



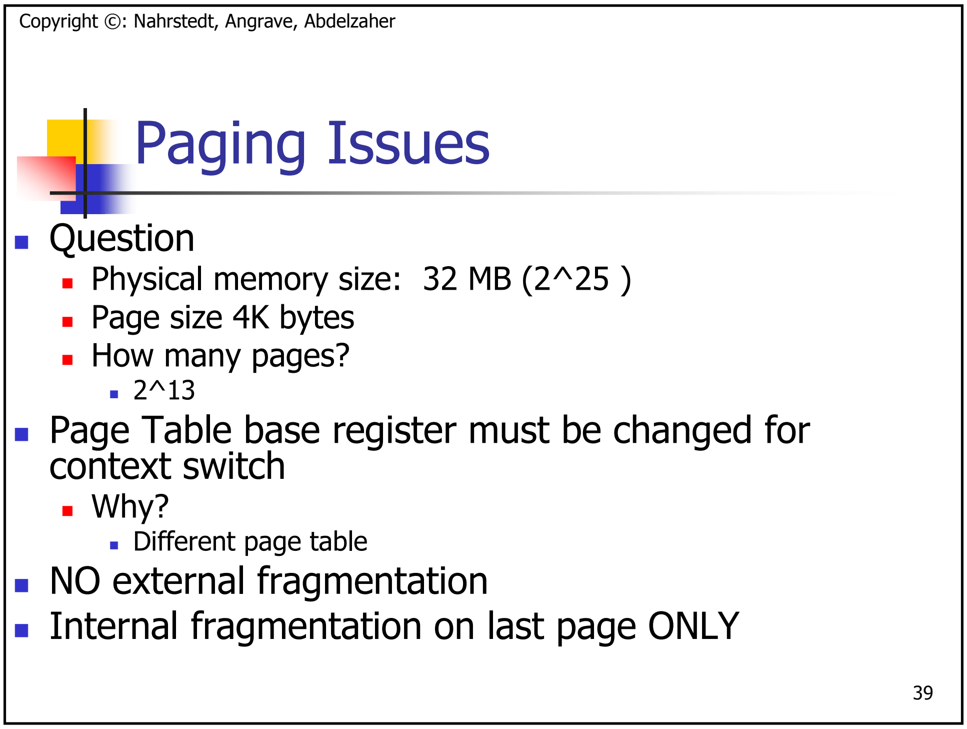
Paging Issues

- size of page is 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
  - Rational?



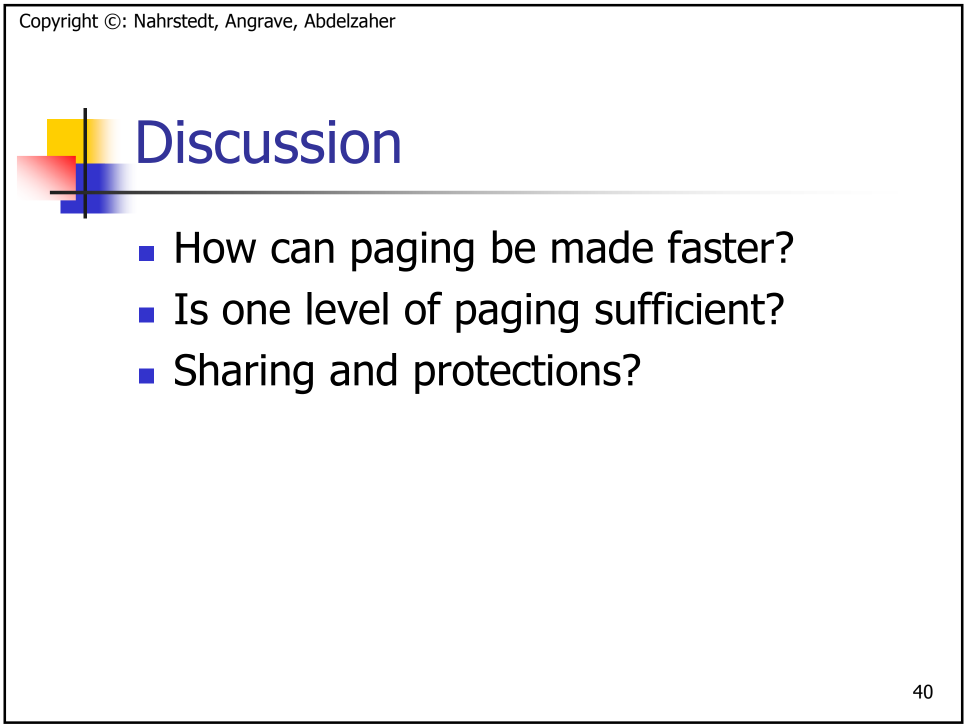
Paging Issues

- 2^{20} (1 meg) 32 bit page entries take 2^{22} bytes (4 meg)
- page frames must map into real memory



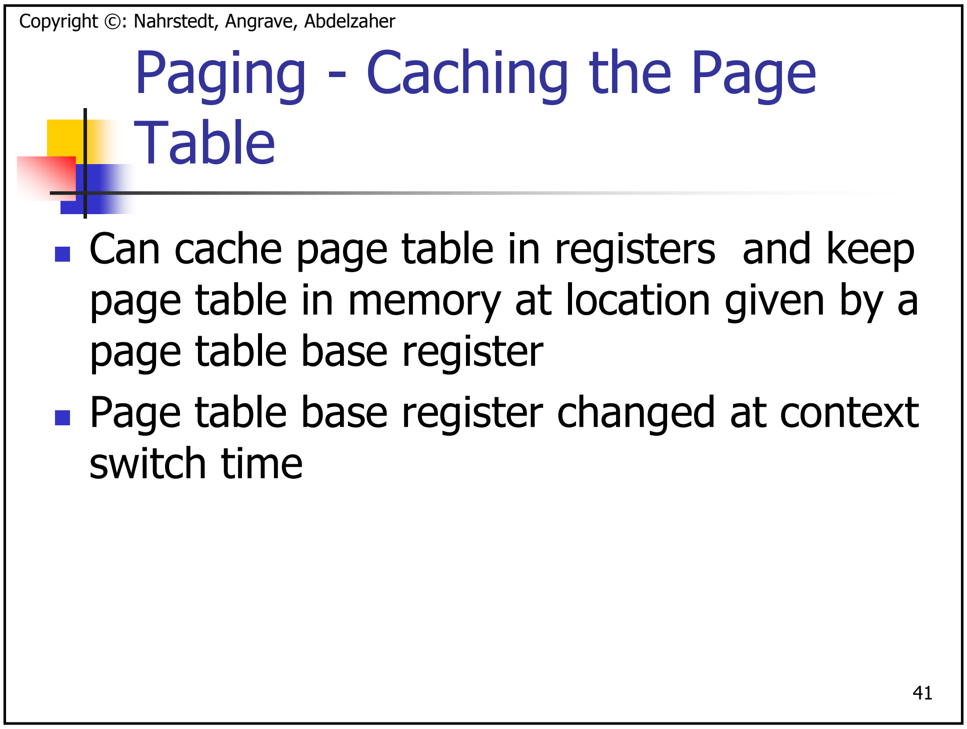
Paging Issues

- Question
  - 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



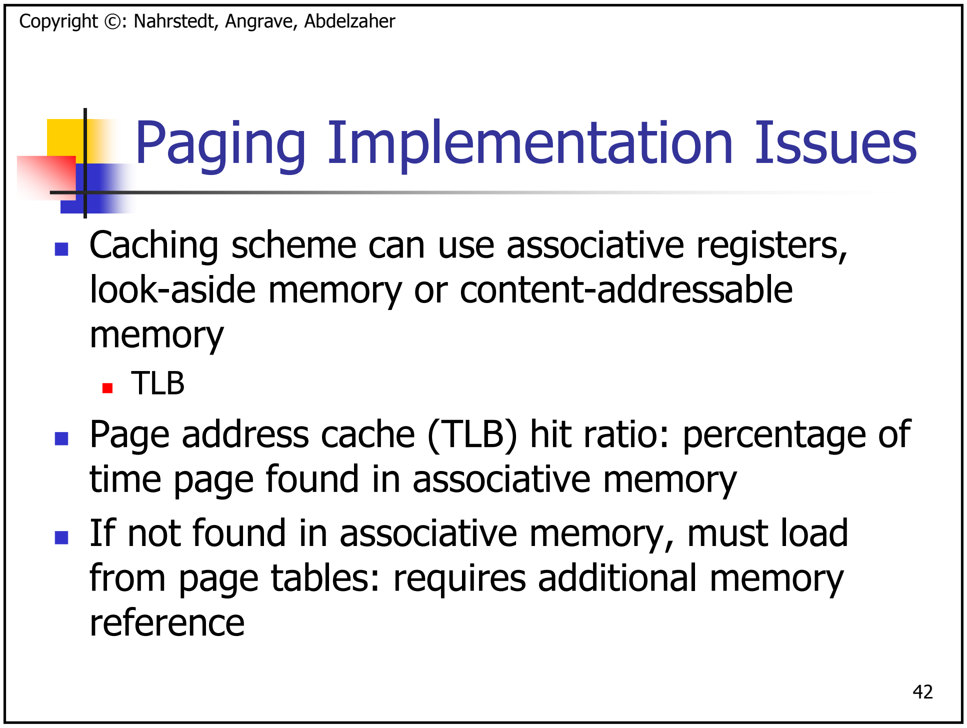
Discussion

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



Paging - Caching the Page Table

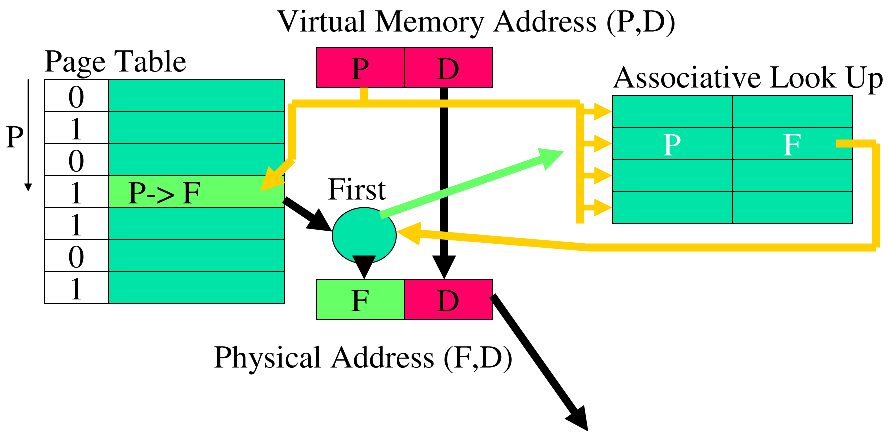
- Can cache page table in registers and keep page table in memory at location given by a page table base register
- Page table base register changed at context switch time



Paging Implementation Issues

- Caching scheme can use associative registers, look-aside memory or content-addressable memory
    - TLB
- Page address cache (TLB) hit ratio: percentage of time page found in associative memory
- If not found in associative memory, must load from page tables: requires additional memory reference

Page Mapping Hardware



43

Page Mapping Hardware

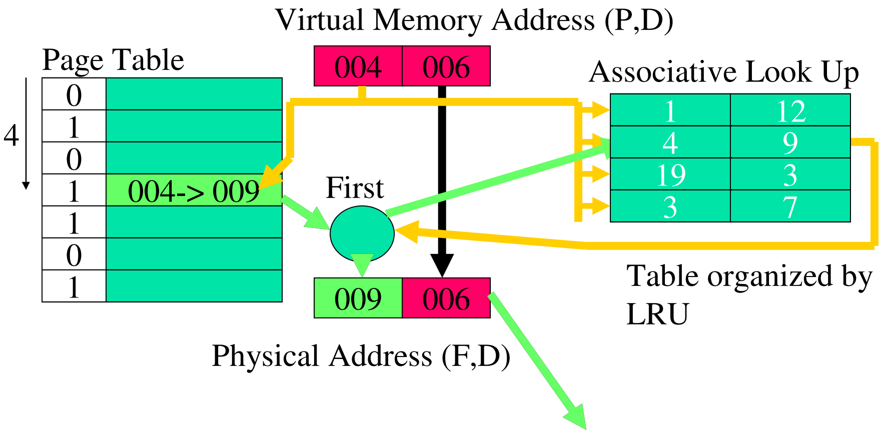
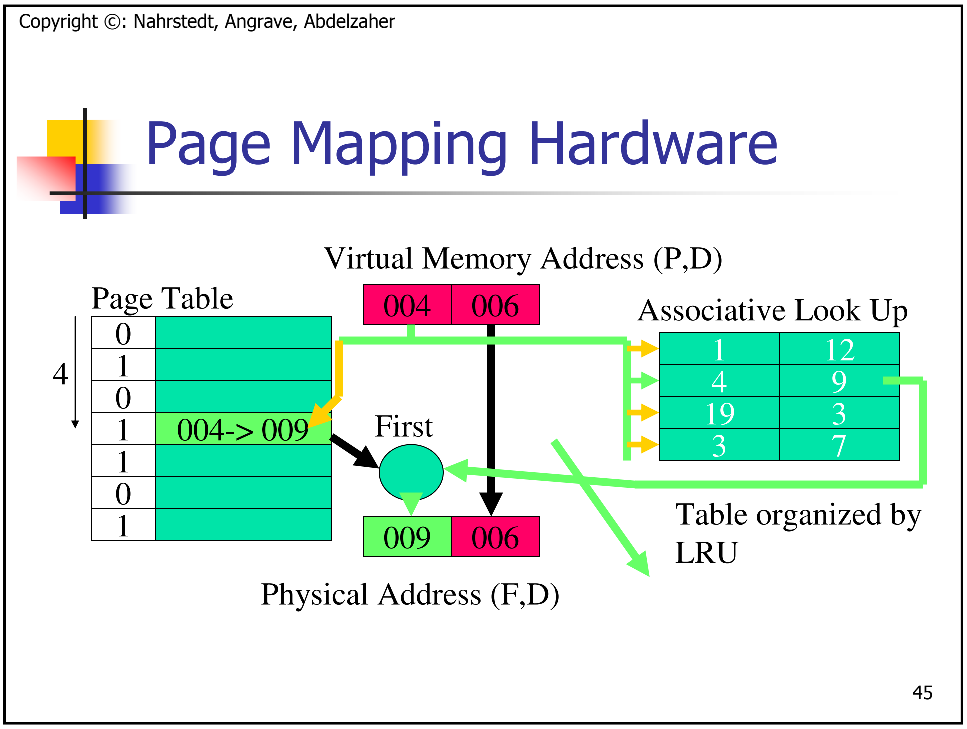


Table organized by LRU

44

Page Mapping Hardware



Virtual Memory Address (P,D)

004	006

Page Table

0	
1	
0	
1	004-> 009
1	
0	
1	

Associative Look Up

1	12
4	9
19	3
3	7

Table organized by LRU

009	006

Physical Address (F,D)