

# **CS241 System Programming Memory Management (IV)**

Klara Nahrstedt

Lecture 31

4/10/2006



# Content

- Multi-level Paging
- Inverted Page Tables
- Paging Policies
  - Fetching
  - Placement
  - Replacement
- Demand Paging
- Replacement Policies
  - Optimal Replacement
  - FIFO Replacement
    - Belady Problem

# Administrative

- MP4 is posted, due April 17, 2006
- Quiz 9 is April 14, 2006
  - Topics for Quiz 9:
    - Basic memory management (T: Chapter 4.1)
    - Swapping (T: Chapter 4.2)
    - Paging (T: Chapter 4.3..1-4.3.3)

# Multilevel Page Tables

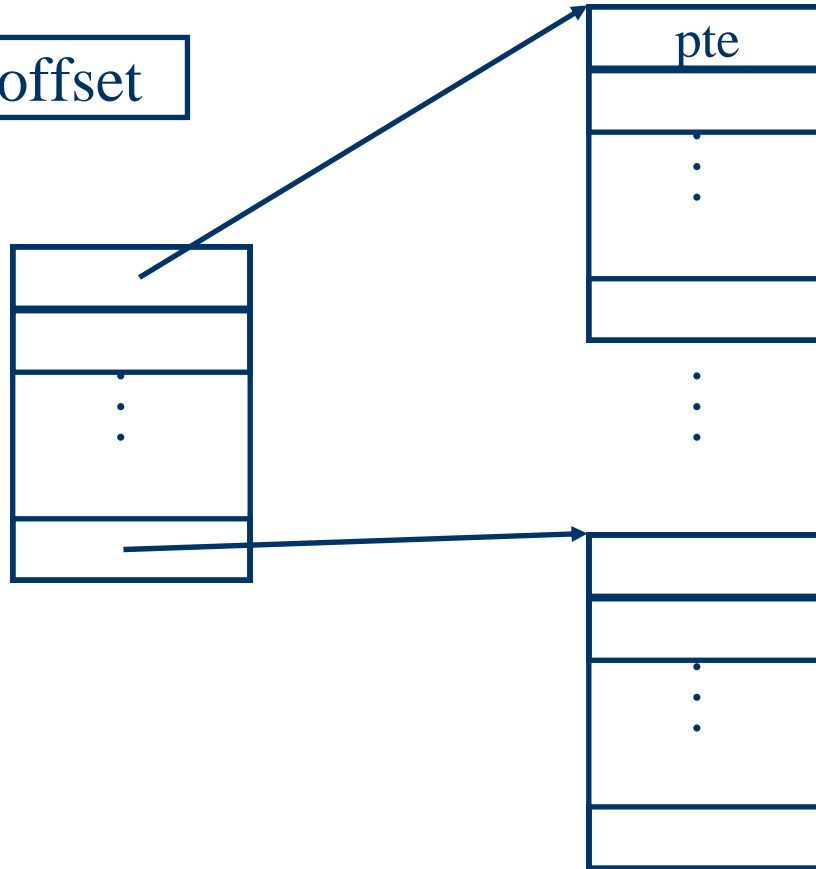
- Since the page table can be very large, one solution is to page the page table
- Divide the page number into
  - An index into a page table of second level page tables
  - A page within a second level page table
- Advantage
  - No need to keep all page tables in memory all the time
  - Only recently accessed memory's mapping need to be kept in memory, the rest can be fetched on demand

# Multilevel Page Tables

Virtual address



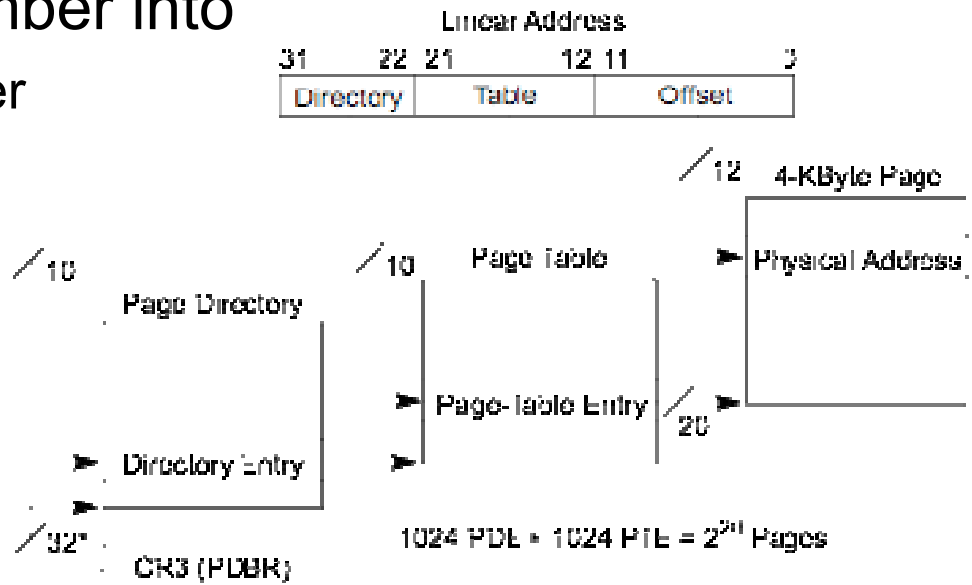
Directory



What does this buy us? Sparse address spaces and easier paging

# Example Addressing on a Multilevel Page Table System

- A logical address (on 32-bit x86 with 4k page size) is divided into
  - A page number consisting of 20 bits
  - A page offset consisting of 12 bits
- Divide the page number into
  - A 10-bit page number
  - A 10-bit page offset

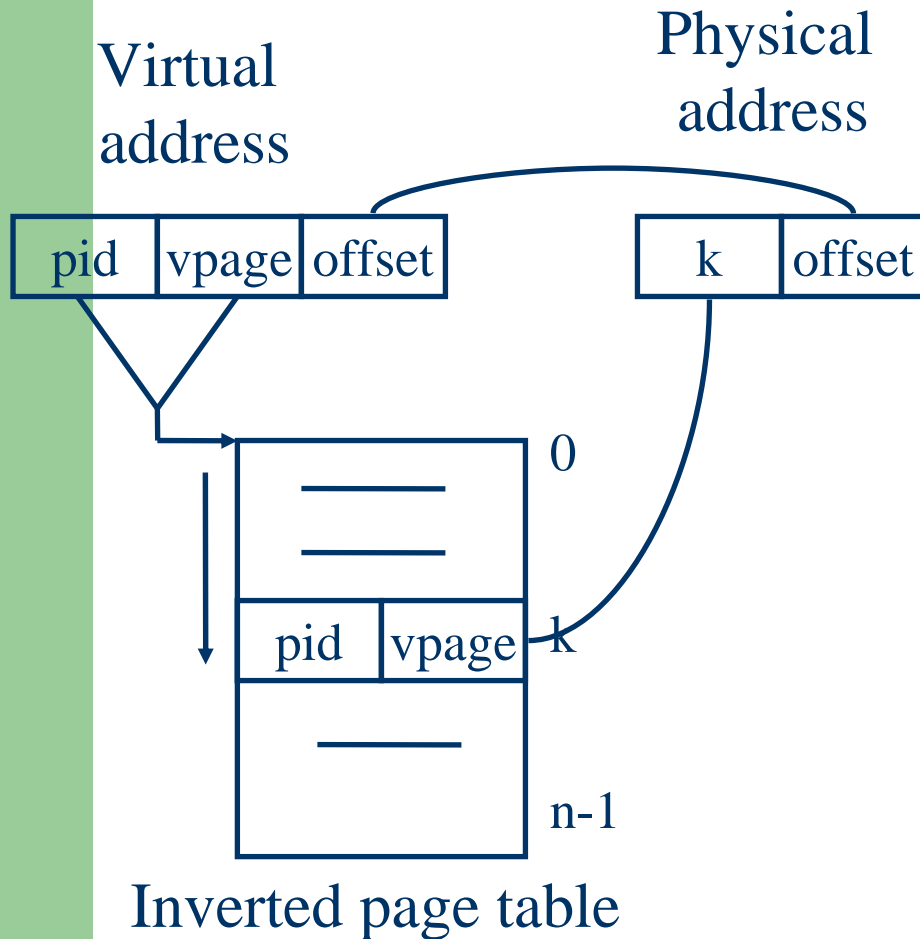


\*32 bits aligned onto a 4-KByte boundary.

# Multilevel Paging and Performance

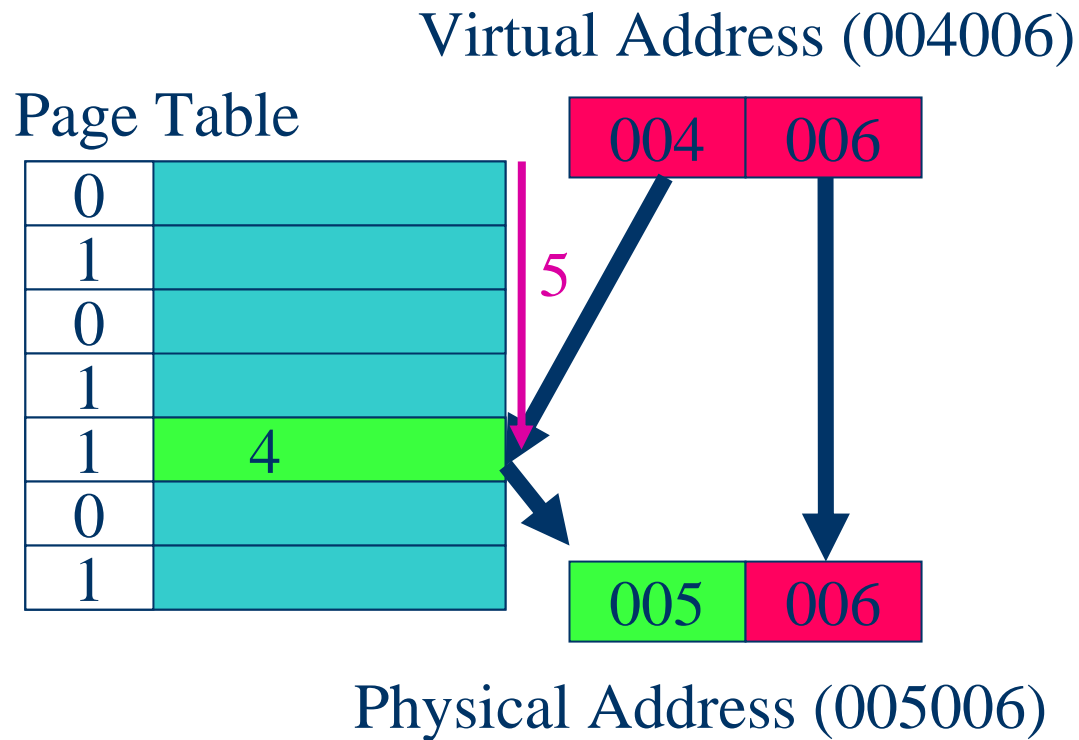
- Since each level is stored as a separate table in memory, converting a logical address to a physical one with a three-level page table may take four memory accesses. Why?

# Inverted Page Tables



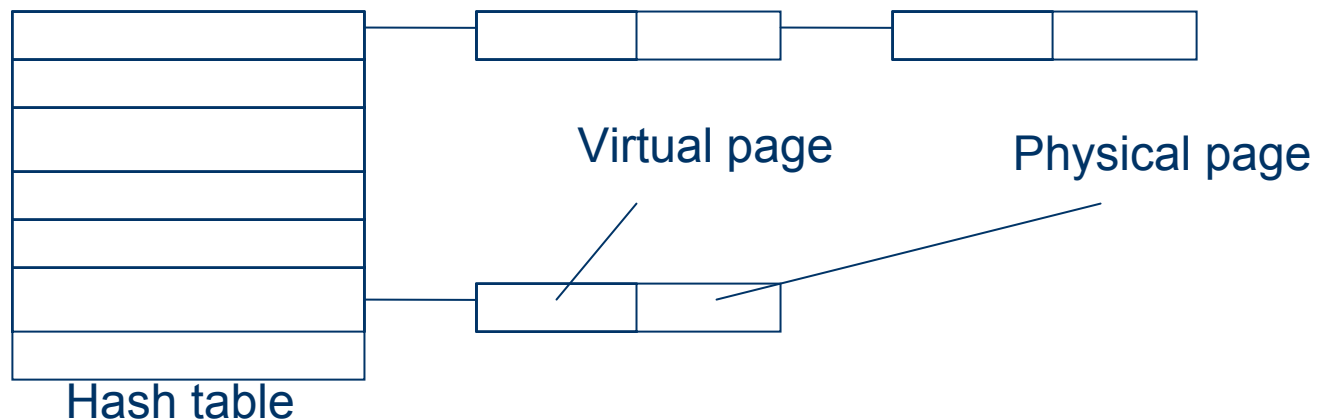
- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
  - Trade off space for time
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

# Inverted Page Table



# Inverted Page Table Implementation

- TLB is same as before
- TLB miss is handled by software
- In-memory page table is managed using a hash table
  - Number of entries  $\geq$  number of physical frames
  - Not found: page fault



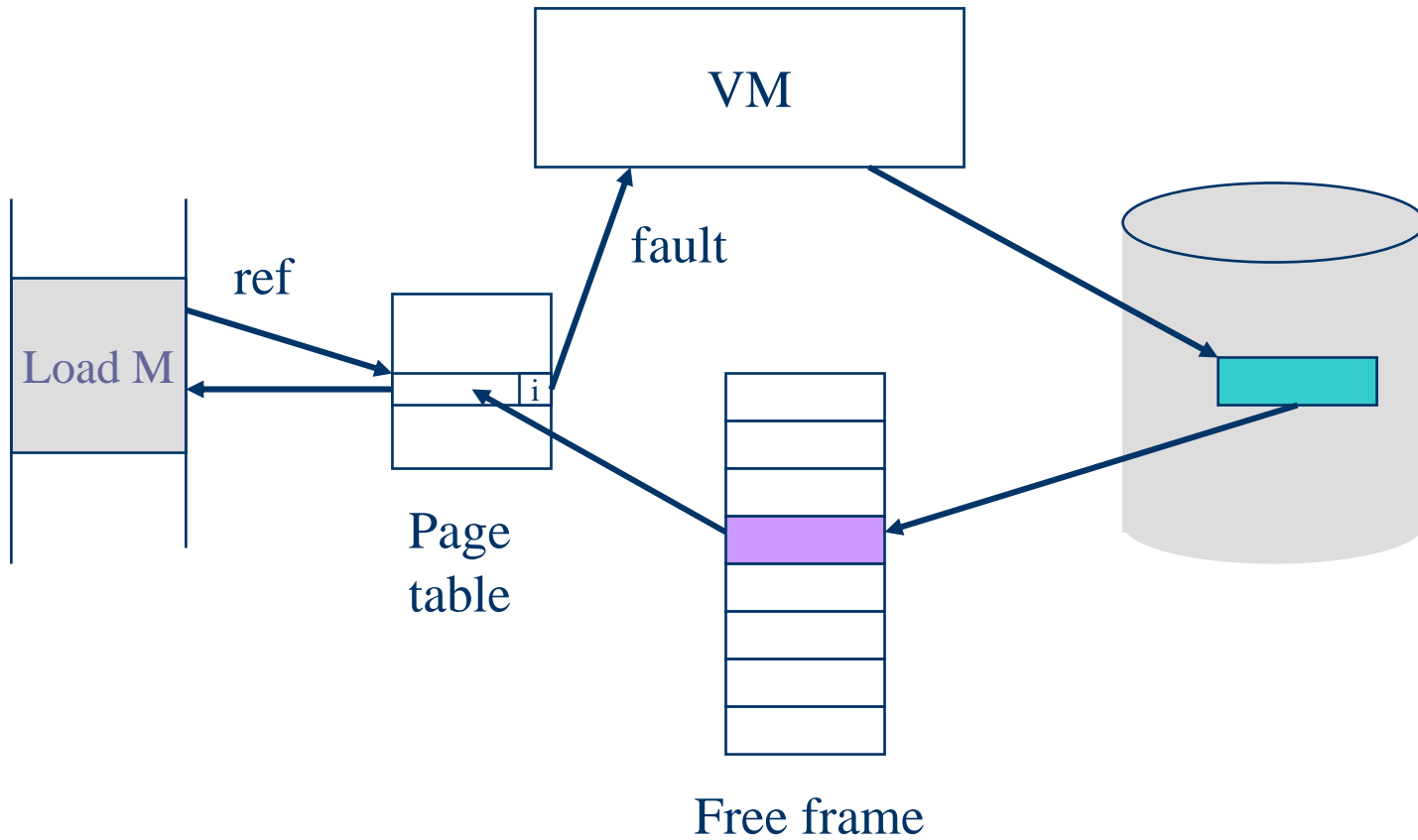
# Paging Policies

- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Placement Strategies
  - When a page is brought into primary storage, where is it to be put?
- Replacement Strategies
  - Which page now in primary storage is to be removed from primary storage when some other page or segment is to be brought in and there is not enough room.

# Demand Paging

- Algorithm Never bring a page into primary memory until its needed.
  1. Page fault
  2. Check if a valid virtual memory address. Kill job if not.
  3. If valid reference, check if its cached in memory already (perhaps for some other process.) If so, skip to 7).
  4. Find a free page frame.
  5. Map address into disk block and fetch disk block into page frame. Suspend user process.
  6. When disk read finished, add vm mapping for page frame.
  7. If necessary, restart process.

# Demand Paging Example



# Page Replacement

1. Find location of page on disk
2. Find a free page frame
  1. If free page frame use it
  2. Otherwise, select a page frame using the page replacement algorithm
  3. Write the selected page to the disk and update any necessary tables
3. Read the requested page from the disk.
4. Restart the user process.
5. It is necessary to be careful of synchronization problems. For example, page faults may occur for pages being paged out.

# Issue: Eviction

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don't
    - Hardware has a dirty bit for each page frame indicating this page has been updated or not
  - Where do you write? To “swap space”
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren't likely to be used again

# Terminology

- **Reference string**: the memory reference sequence generated by a program.
- **Paging** – moving pages to (from) disk
- **Optimal** – the best (theoretical) strategy
- **Eviction** – throwing something out
- **Pollution** – bringing in useless pages/lines

# Page Replacement Strategies

- **The Principle of Optimality**
  - Replace the page that will not be used again the farthest time in the future.
- **Random page replacement**
  - Choose a page randomly
- **FIFO - First in First Out**
  - Replace the page that has been in primary memory the longest
- **LRU - Least Recently Used**
  - Replace the page that has not been used for the longest time
- **LFU - Least Frequently Used**
  - Replace the page that is used least often
- **NRU - Not Recently Used**
  - An approximation to LRU.
- **Working Set**
  - Keep in memory those pages that the process is actively using.

# Principal of Optimality

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first references, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- This algorithm provides a basis for comparison with other schemes.
- Impractical because it needs future references
- If future references are known
  - should not use demand paging
  - should use pre paging to allow paging to be overlapped with computation.

# Optimal Example

12 references,  
7 faults

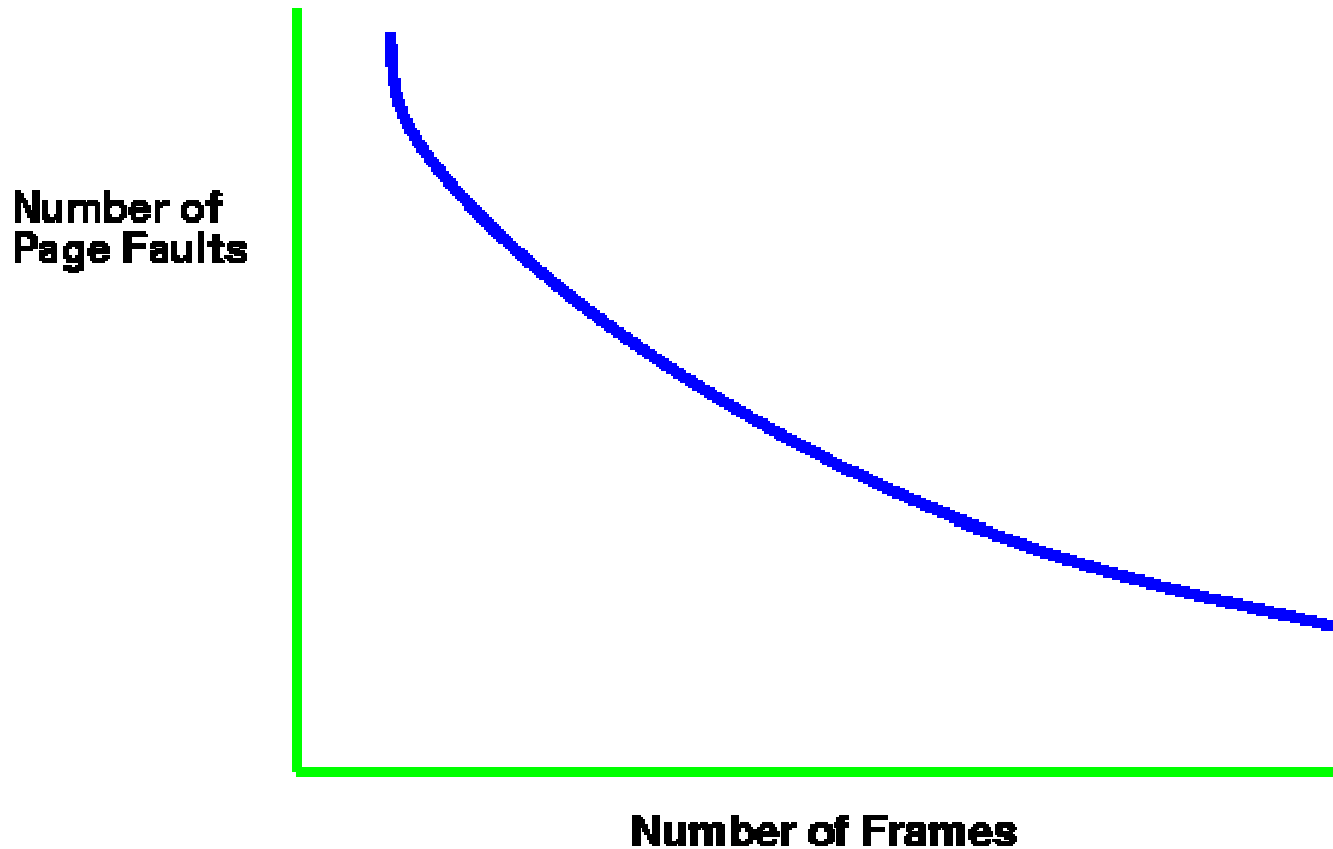
Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

# FIFO

12 references,  
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

# Paging Behavior with Increasing Number of Page Frames



# Belady's Anomaly (for FIFO)

As the number of page frames increase, so does the fault rate.

12 references,  
10 faults

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

# Summary

- Inverted page tables are very useful if page tables become too large (e.g., in 64 bit address space)
- Demand Paging is the basic strategy that the advanced memory management is based on
- Replacement strategies are of great importance to keep the space efficiently used