WEEK 4

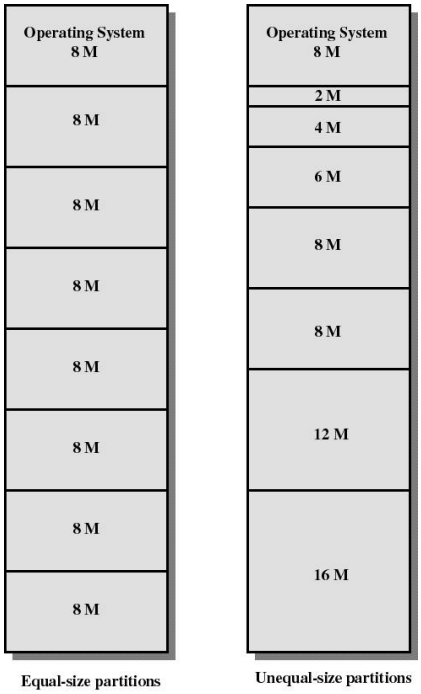
Memory management and virtual memory

Stallings, Chapters 7 & 8.1

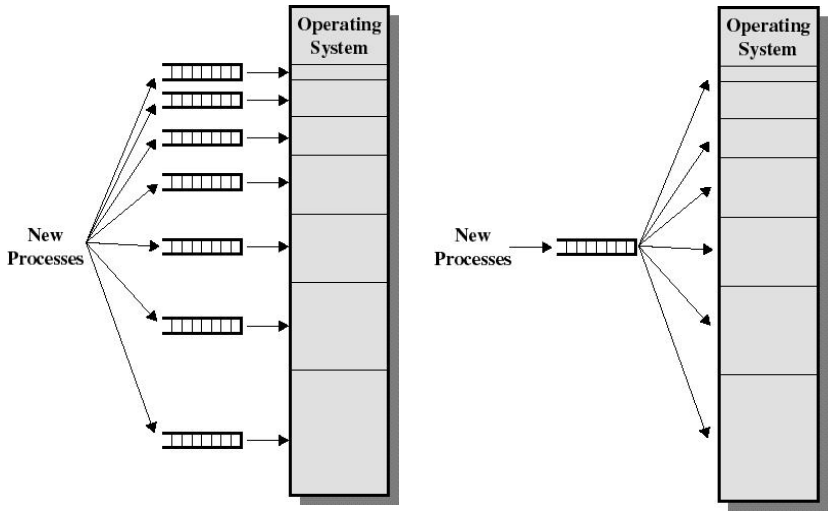
1

Fixed partitions

- OS splits the memory to partitions
- Whole process must fit to a partition
- Program sizes must be at most the size of the partitions
- Running larger programs required special effort from the programmer
- SWAPPING out to make place for another process
- internal fragmentation



Fixed partitions: allocation

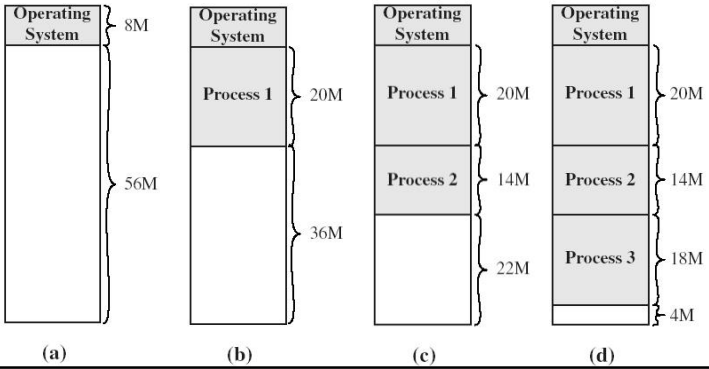


Kuva 7.3

3

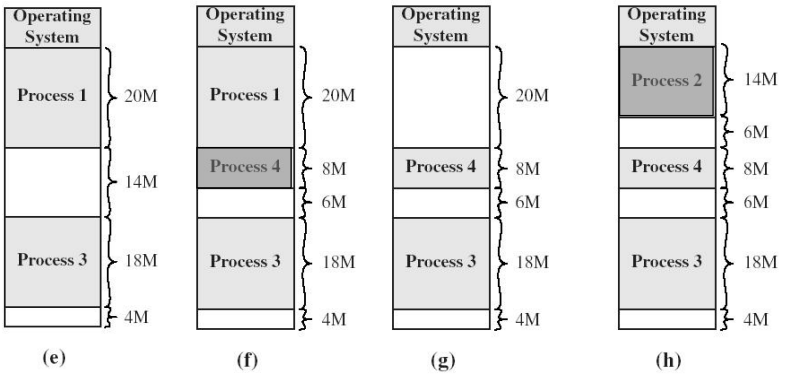
Dynamic partitioning

- No fixed predetermined partition sizes
- Each process gets only the amount of memory it needs
- **external fragmentation**
- OS must occasionally reorganize the memory **(compaction)**



4

Dynaamic partitions



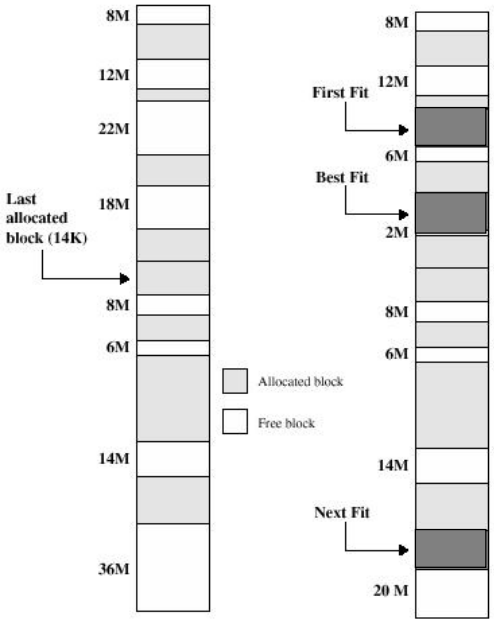
- Needed space for process 4 -> swap out process 2
- External fragments: 6M + 6M + 4M = 14M

5

Allocation Kuva 7.5

Where to place the new process?

- Goal: avoid external fragmentation and compation
- Some alternatives:
- Best-fit
- First-fit
- Next-fit



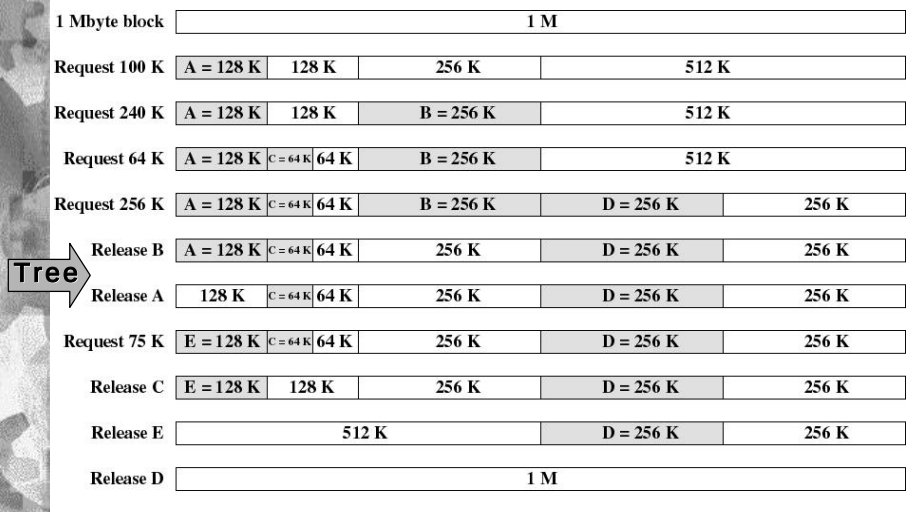
Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Buddy System

- Compromise: Fixed partition sizes, dynamic splitting
- Partition sizes are 2:n potenssissa
  - Some fixed smallest allocation size
- Allocation and combining efficient
- List for each separate size
- Allocation
  - If correct size not available, split a larger one
- Release
  - When two adjacent same sized areas free combine them

7

Buddy System



Kuva 7.6

8

Buddy: tree representation

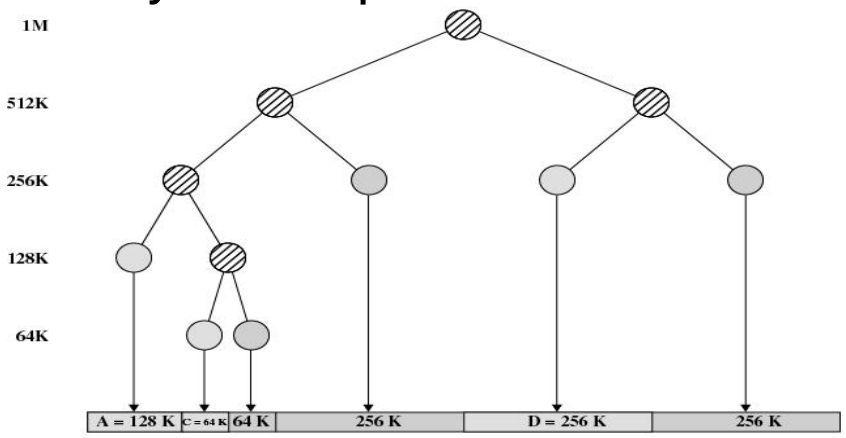
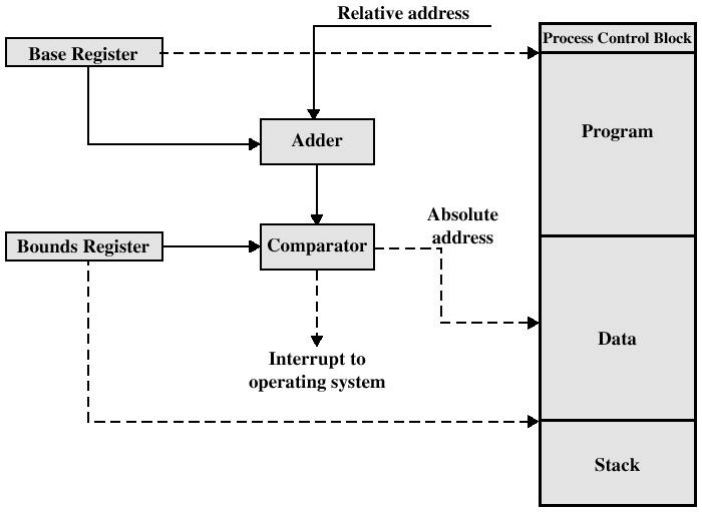


Figure 7.7 Tree Representation of Buddy System

9

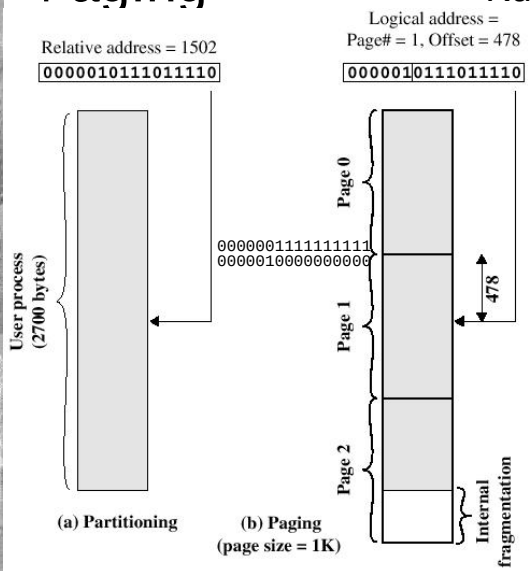
Address translations (fixed part., dynamic part. and buddy system)



10

Paging

Kuva 7.11

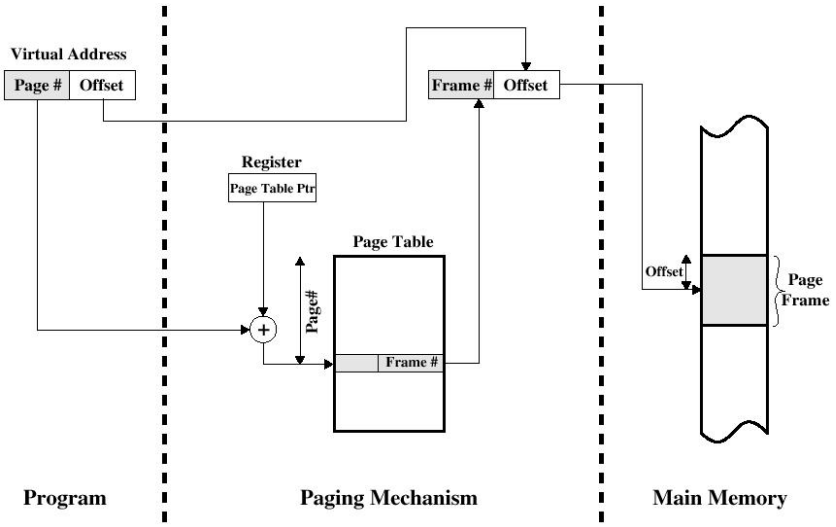


- Each process has own page table
  - Contains the locations (frame numbers) of allocated frames
- OS maintains a table (or list) of page frames, to know which are unallocated

11

Address translation

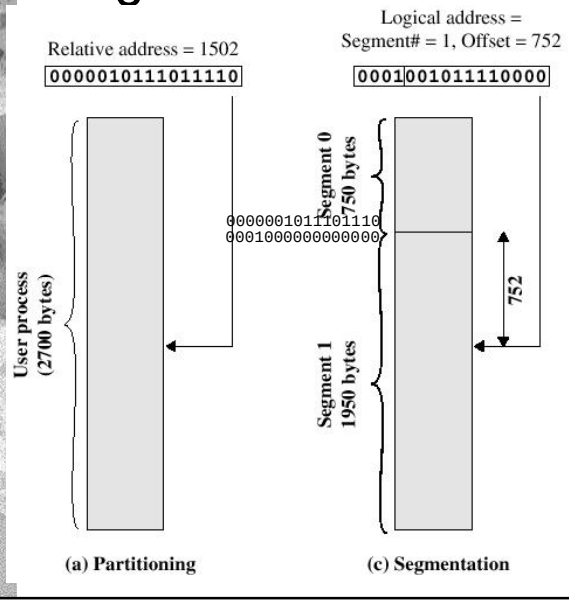
Kuva 8.3



12

Segmentation

Kuva 7.11

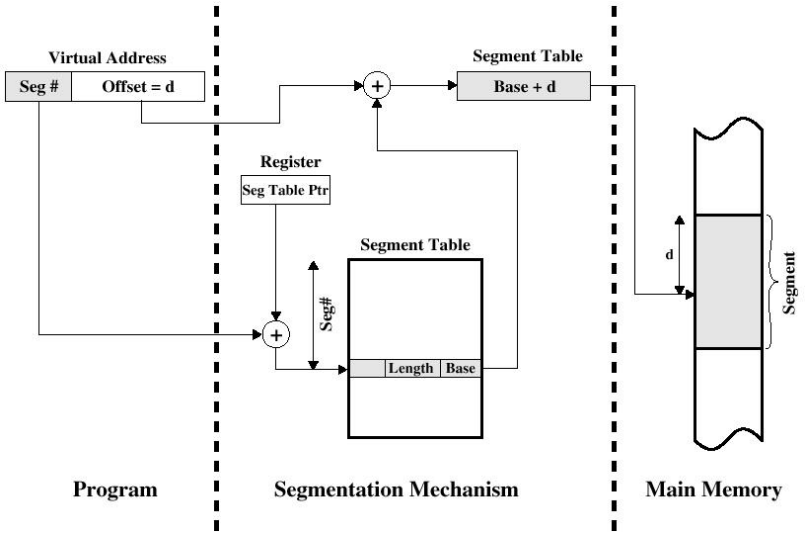


- Programmer (or compiler) determines the logical, unequal-sized segments
- Still using logical addresses (segment, offset)
- Usually a maximum size for each segment given
- Segments can be freely located by OS
- OS maintains a segment table for each process

13

Address translation (segm.)

Kuva 8.12



14

WEEK 4

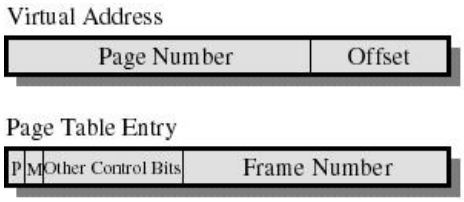
Virtual memory

Stallings, Chapter 8

15

Page table

Kuva 8.2a



P= present bit
M = Modified bit

- Each process has its own page table
- Each entry has a present bit, since not all pages need to be in the memory all the time -> page faults
- Remember the locality principle
- Logical address space can be much larger than the physical

16

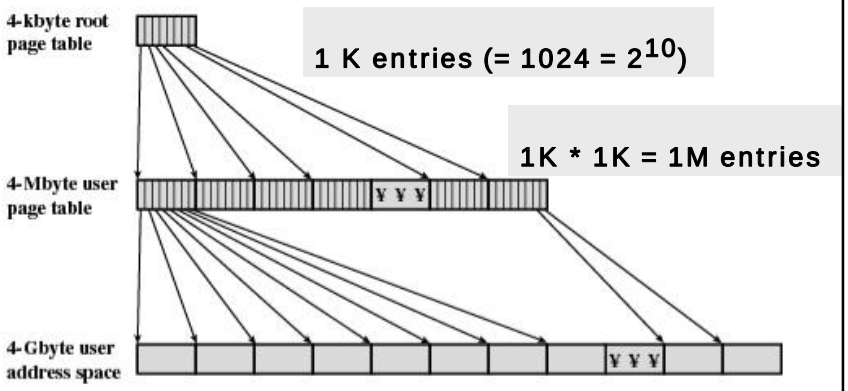
Structure of page table

- Large virtual address space
  - Logical address could be 32- or 64-bits
- Each process has a large page table
  - Using 32-bit address and frame size 4KB (12 bit offset), means 2^{20} = 1M of page tables entries for a single process
  - Each entry requires several bytes (lets say 4 bytes), so the final size of page table could be for example 4 MB
- Thus the page table is divided to several pages also and part of it can be on the disk
  - We only need the part of the page table in the memory that covers the pages used currently in the execution of the process

17

Two-level hierarchical page table

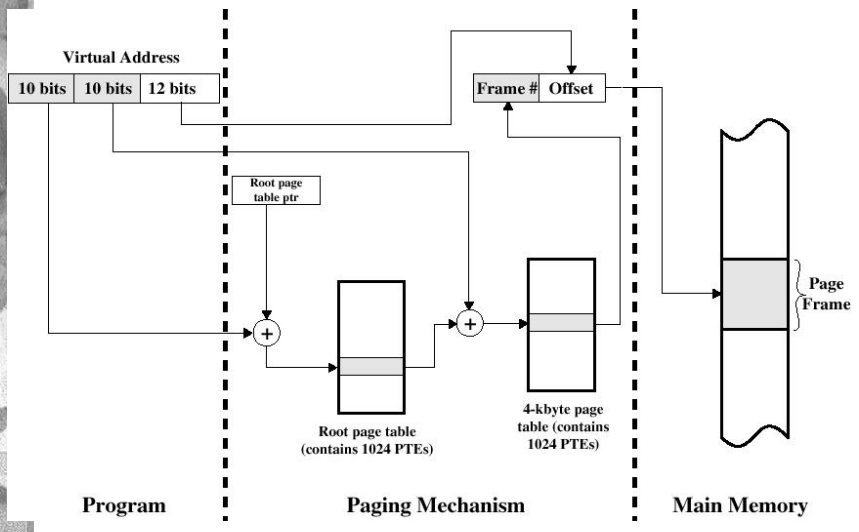
- Top most level in one page and always in the memory



8

Figure 8.4 A Two-Level Hierarchical Page Table

Address translation with two levels Kuva 8.5



19

Inverted page table

20

Inverted page table

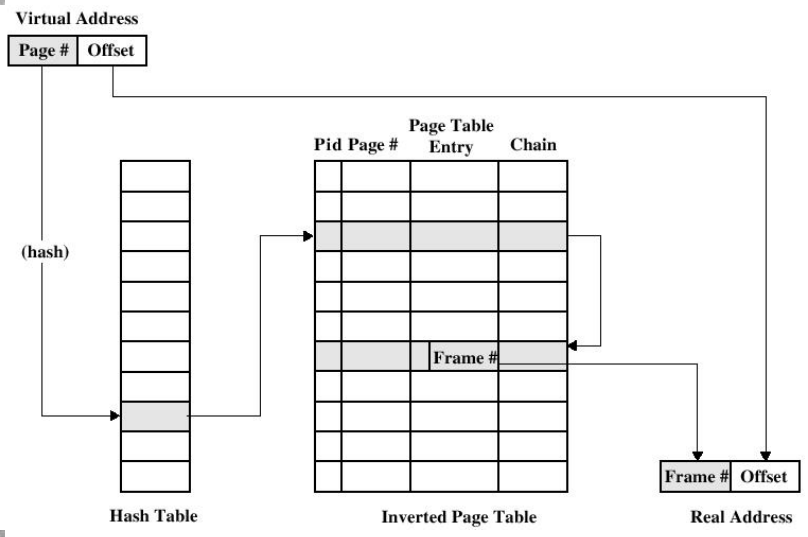
- Physical memory often smaller than the virtual address space of processes
- Invert booking: Store for each page frame what page (of which process) is stored there
  - Only one global table (inverted page table), one entry for each page frame.
- Seach for the page based on the content of the table
  - Ineffcient, if done sequentially,
  - Use hash to calculate the location, start search from there
- If page not found, page fault
- Useful, only if TLB is large

21

Inverted page table

4. ed

Kuva 8.6



22

5. ed

Inverted page table v.2

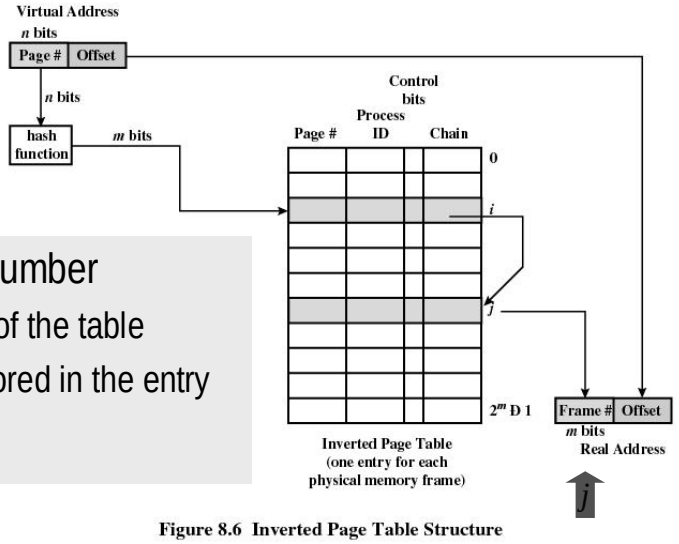


Figure 8.6 Inverted Page Table Structure

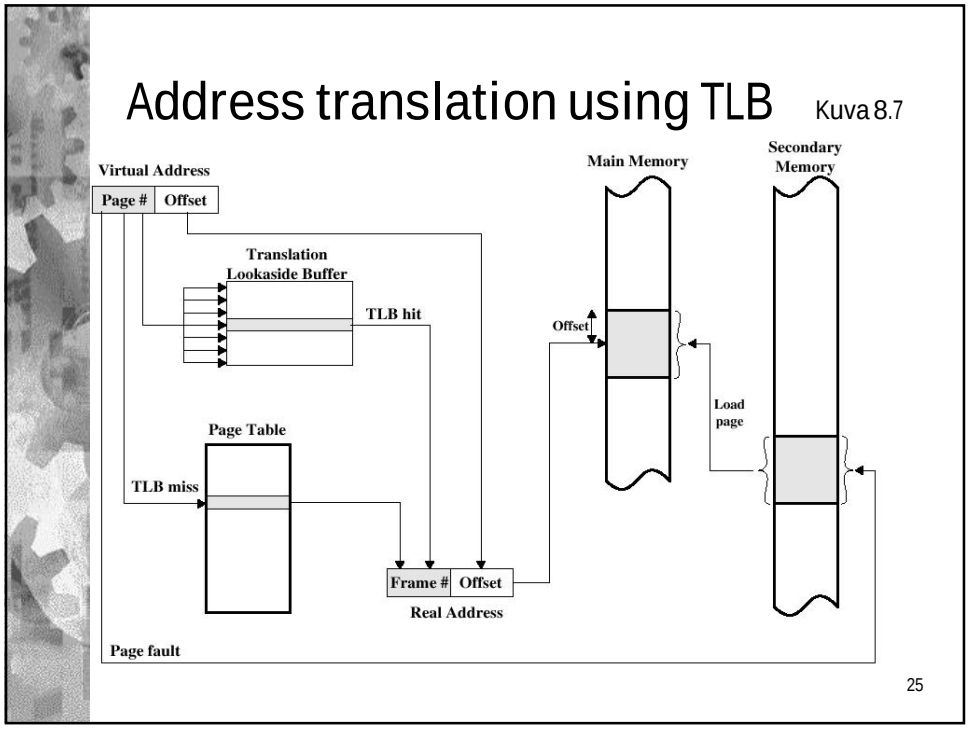
- Frame number
  - Index of the table
  - Not stored in the entry

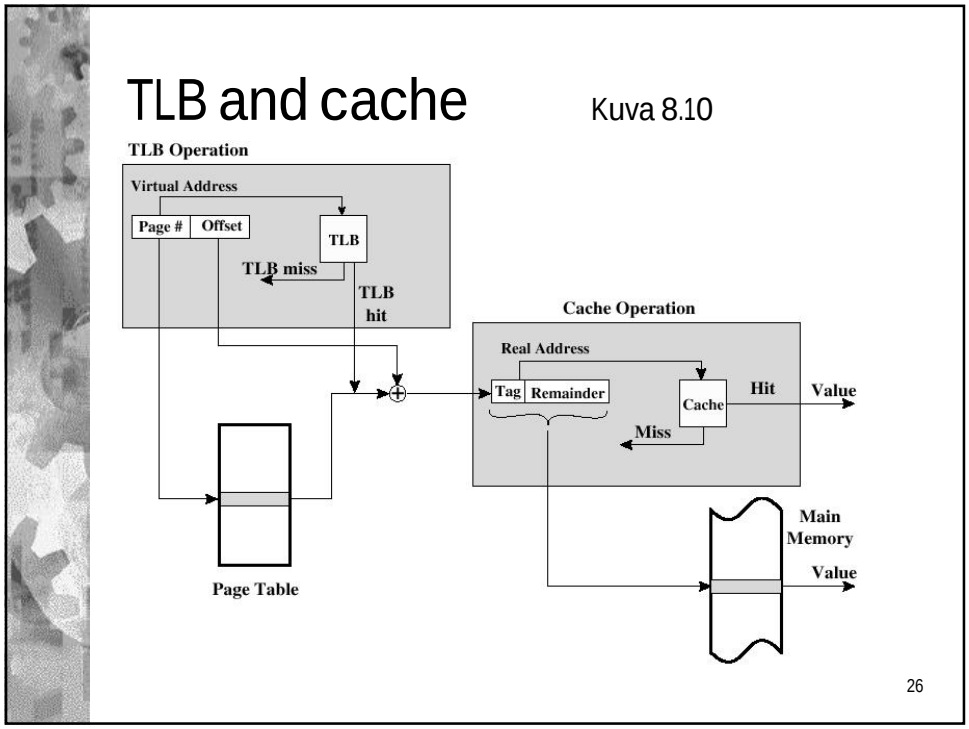
23

TLB – translation lookaside buffer

- Part of memory management unit (MMU)
  - Cache for used page table entries to avoid extra memory access during address translation
- Associative search
  - Compare with all elements at the same time (fast)
- Each TLB element contains: page number, page table entry, validity bit
- Each process uses the same page numbers 0, 1, 2, …, stored on different page frames
  - è TLB must be cleared during process switch
  - è At least clear the validity bits (this is fast)

24

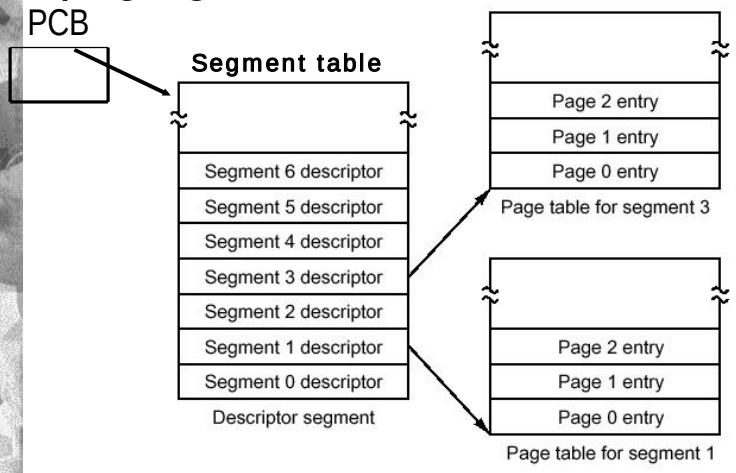

Address translation using TLB
Kuva 8.7
Virtual Address
Page #
Offset
Translation
Lookaside Buffer
TLB hit
Main Memory
Secondary
Memory
Offset
Load
page
Page Table
TLB miss
Frame #
Offset
Real Address
Page fault
25


TLB and cache
Kuva 8.10
TLB Operation
Virtual Address
Page #
Offset
TLB
TLB miss
TLB
hit
Cache Operation
Real Address
Tag
Remainder
Cache
Hit
Value
Miss
Main
Memory
Value
Page Table
26

Segmentointi ja sivutus yhdistettynä

27

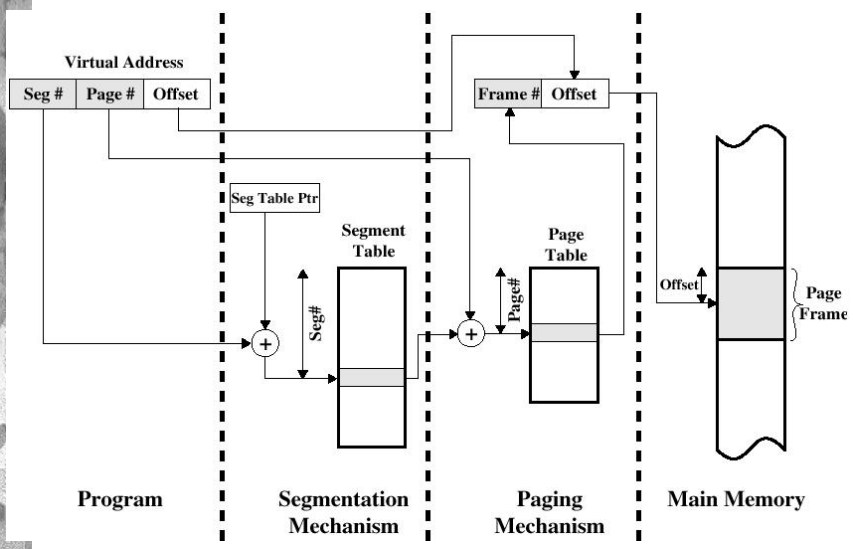
Combining segmentation and paging Tan01 4-39



28

Combined segmentation and paging

Kuva 8.13



29

Protection and sharing

No slides in English (sorry)

30