

# Virtual Memory (VM)

## Ch 8.3

Memory Management  
Address Translation  
Paging  
Hardware Support  
VM and Cache

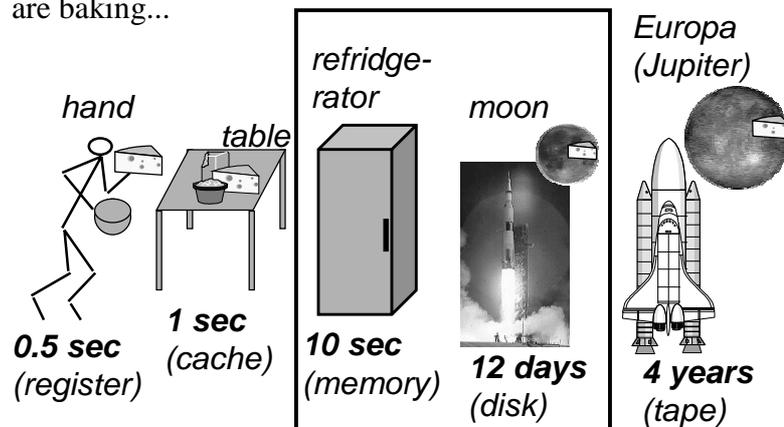
23.9.2002

Copyright Teemu Kerola 2002

1

## Teemu's Cheesecake

Register, on-chip cache, memory, disk, and tape speeds relative to times locating cheese for the cheese cake you are baking...



23.9.2002

Copyright Teemu Kerola 2002

2

## Virtual Memory

(virtuaalimuisti)

- Problem: How can I make my (main) memory as big as my disk drive?
- Answer: Virtual memory
  - keep only most probably referenced data in memory, and rest of it in disk
    - disk is much bigger and slower than memory
    - address in machine instruction may be different than memory address
    - need to have efficient address mapping
    - most of references are for data in memory
  - joint solution with HW & SW

23.9.2002

Copyright Teemu Kerola 2002

3

## Other Problems Often Solved with VM <sup>(3)</sup>

- If you must want to have many processes in memory at the same time, how do you keep track of memory usage?
- How do you prevent one process from touching another process' memory areas?
- What if a process needs more memory than we have?

23.9.2002

Copyright Teemu Kerola 2002

4

## Memory Management Problem <sup>(4)</sup>

- How much memory for each process?
  - is it fixed amount during the process run time or can it vary during the run time?
- Where should that memory be?
  - in a continuous or discontinuous area?
  - is the location the same during the run time or can it vary dynamically during the run time?
- How is that memory managed?
- How is that memory referenced?

23.9.2002

Copyright Teemu Kerola 2002

5

## Partitioning <sup>(3)</sup>

- How much physical memory for each process?
- Static (fixed) partitioning (staattiset tai kiinteät partitiot)
  - amount of physical memory determined at process creation time
  - continuous memory allocation for partition
- Dynamic partitioning (dynaamiset partitiot)
  - amount of physical memory given to a process varies in time
    - due to process requirements (of this process)
    - due to system (I.e., other processes) requirements

23.9.2002

Copyright Teemu Kerola 2002

6

## Static Partitioning

- Equal size - give everybody the same amount
  - fixed size - big enough for everybody
    - too much for most
    - need more? Can not run!
- Unequal size
  - sizes predetermined
- Variable size
  - Size determined at process creation time

23.9.2002

Copyright Teemu Kerola 2002

7

## Fragmentation

- Internal fragmentation (sisäinen pirstoutuminen)
  - unused memory inside allocated block
  - e.g., equal size fixed memory partitions
- External fragmentation (ulkoinen pirstoutuminen)
  - enough free memory, but it is splintered as many un-allocatable blocks
  - e.g., unequal size partitions or dynamic fixed size (variable size) memory partitions

23.9.2002

Copyright Teemu Kerola 2002

8

## Dynamic Partitioning <sup>(3)</sup>

- Process must be able to run with varying amounts of main memory
  - all of memory space is **not** in physical memory
  - need some minimum amount of memory
- New process?
  - reduce amount of memory for some (lower priority) processes
- Not enough memory for some process?
  - reduce amount of memory for some (lower priority) processes
  - kick (swap) out some (lower priority) process

23.9.2002

Copyright Teemu Kerola 2002

9

## Address Mapping <sup>(4)</sup>

(osoitteen muunnos)

Pascal, Java:

```
while (...)
  X := Y+Z;
```

Symbolic Assembler:

```
loop: LOAD    R1, Y
      ADD     R1, Z
      STORE   R1, X
```

Textual machine language:

```
1312: LOAD    R1, 2510
      ADD     R1, 2514
      STORE   R1, 2600
```

(addresses relative to 0)

Execution time:

```
101312: LOAD  R1,102510
        ADD   R1,102514
        ADD   R1,102600
```

(real, actual!)

23.9.2002

Copyright Teemu Kerola 2002

10

### Address Mapping <sup>(2)</sup>

logical address

Textual machine language:

1312:      LOAD      R1, 2510

Execution time:

101312:    LOAD      R1, 102510    or

101312:    LOAD      R1, 2510        ??

+100000?
physical address (constant?)
logical addr

- Want: R1 ← Mem[102510] or Mem[2510] ?

- Who makes the mapping? When?

23.9.2002
Copyright Teemu Kerola 2002
11

### Address Mapping <sup>(2)</sup>

- At program load time
  - loader (lataaja)
  - static address binding (staattinen osoitteiden sidonta)
- At program execution time
  - cpu
  - with every instruction
  - dynamic address binding (dynaaminen osoitteiden sidonta)
  - swapping
  - virtual memory

23.9.2002
Copyright Teemu Kerola 2002
12

## Swapping <sup>(4)</sup>

(heittovaihto)

- Keep all memory areas for all running and ready-to-run processes in memory
- New process
  - find continuous memory partition and swap the process in
- Not enough memory?
  - Swap some (lower priority) process out
- Some times can swap in only (runnable) portions of one process
- Address map: add base address

23.9.2002

Copyright Teemu Kerola 2002

13

## VM Implementation <sup>(2)</sup>

- Methods
  - base and limit registers
  - segmentation
  - paging
  - segmented paging, multilevel paging
- Hardware support
  - MMU - Memory Management Unit
    - part of processor
    - varies with different methods
  - Sets limits on what types of virtual memory (methods) can be implemented using this HW

23.9.2002

Copyright Teemu Kerola 2002

14

## Base and Limit Registers (2)

- Continuous memory partitions
  - one or more (4?) per process
  - may have separate base and limit registers
    - code, data, shared data, etc
    - by default, or given explicitly in each mem. ref.
- *BASE* and *LIMIT* registers in MMU
  - all addresses logical in machine instructions
  - address mapping for address (x):
    - check:  $x < LIMIT$
    - physical address:  $BASE+x$

23.9.2002

Copyright Teemu Kerola 2002

15

## Segmentation (4)

- Process address space divided into (relatively large) logical segments
  - code, data, shared data, large table, etc
  - object, module, etc
- Each logical segment is allocated its own continuous physical memory segment
- Memory address has two fields

011001 1010110000  
 segment byte offset (lisäys)

23.9.2002

Copyright Teemu Kerola 2002

16

## Segment. Address Mapping <sup>(3)</sup>

- Segment table
  - maps segment id to physical segment base address and to segment size
- Physical address
  - find entry in segment table
  - check: byte offset < segment size
  - physical address: base + byte offset
- Problem: variable size segments
  - External fragmentation, lots of memory management

23.9.2002

Copyright Teemu Kerola 2002

17

## Paging <sup>(4)</sup>

- Process address space divided into (relatively small) equal size pages
  - address space division is not based on logical entities, only on fixed size chunks designed for efficient implementation
- Each page is allocated its own physical page frame in memory
  - any page frame will do!
- Internal fragmentation
- Memory addresses have two fields

01100110 10110000

page      byte offset      (lisäys)

23.9.2002

Copyright Teemu Kerola 2002

18

## Paged Address Mapping

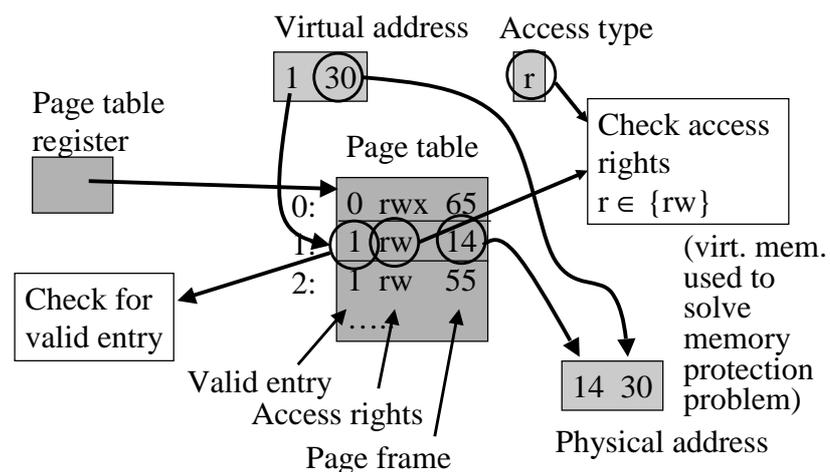
- Page table
  - maps page nr to physical page frame
- Physical address
  - find entry in page table (large array in memory)
  - get page frame, I.e., page address
  - physical address: page address + byte offset

23.9.2002

Copyright Teemu Kerola 2002

19

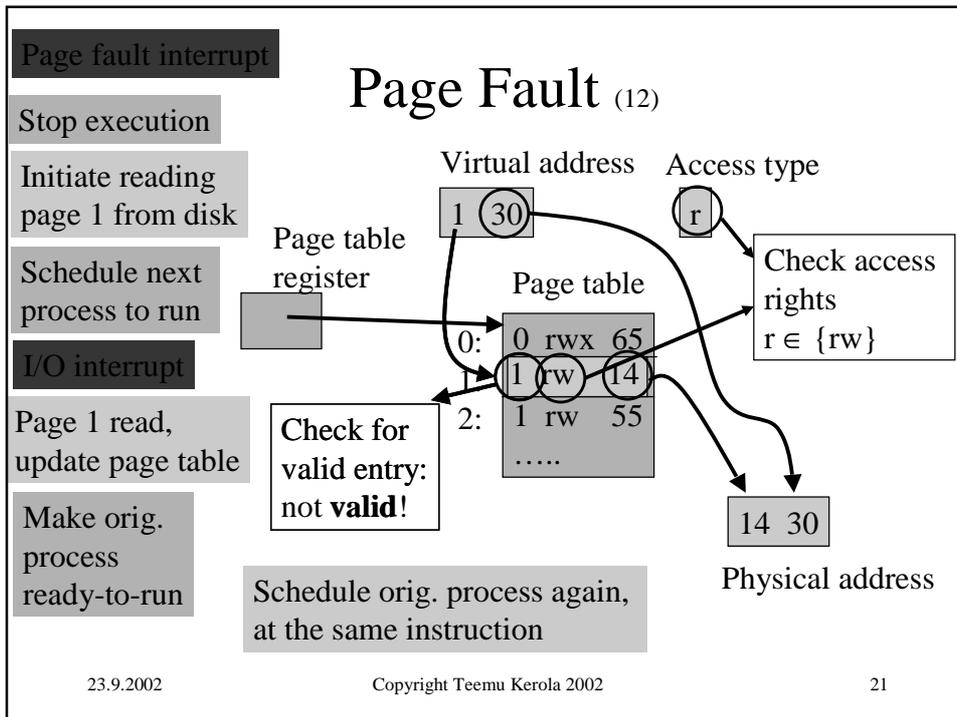
## Paged Address Translation (4)



23.9.2002

Copyright Teemu Kerola 2002

20



- ### Paging (3)
- Physical memory partitioning
    - discontinuous areas
  - Page tables
    - located in memory
    - can be very big, and each process has its own
      - entry for each page in address space
  - Inverted page table
    - entry for each page in memory
    - less space, more complex hashed loc
- Fig. 8.15 (Fig. 7.16 [Stal99])
- (Fig. 7.18 [Stal99]) Fig. 8.17
- 23.9.2002 Copyright Teemu Kerola 2002 22

## Address Translation <sup>(3)</sup>

- MMU does it for every memory access
  - code, data
  - more than once per machine instruction!
- Can not access page tables in memory every time - it would be too slow!
  - too high cost to pay for virtual memory?
- MMU has a “cache” of most recent address translations
  - TLB - Translation Lookaside Buffer
  - 99.9% hit ratio?

(osoitteennuunnos-  
taulukko)

23.9.2002

Copyright Teemu Kerola 2002

23

## Translation Lookaside Buffer <sup>(3)</sup>

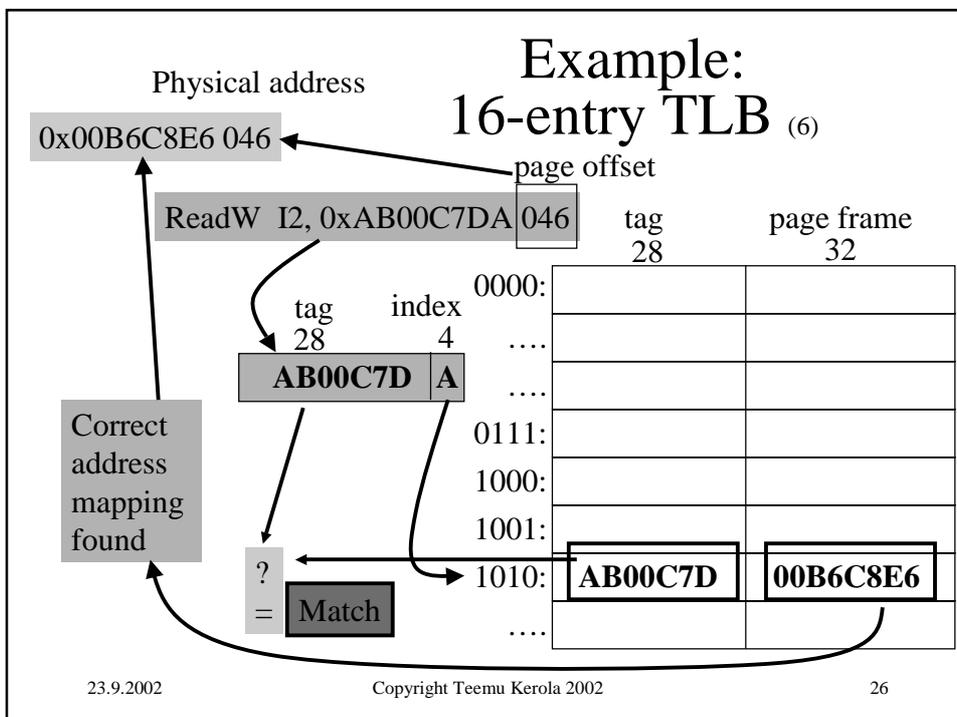
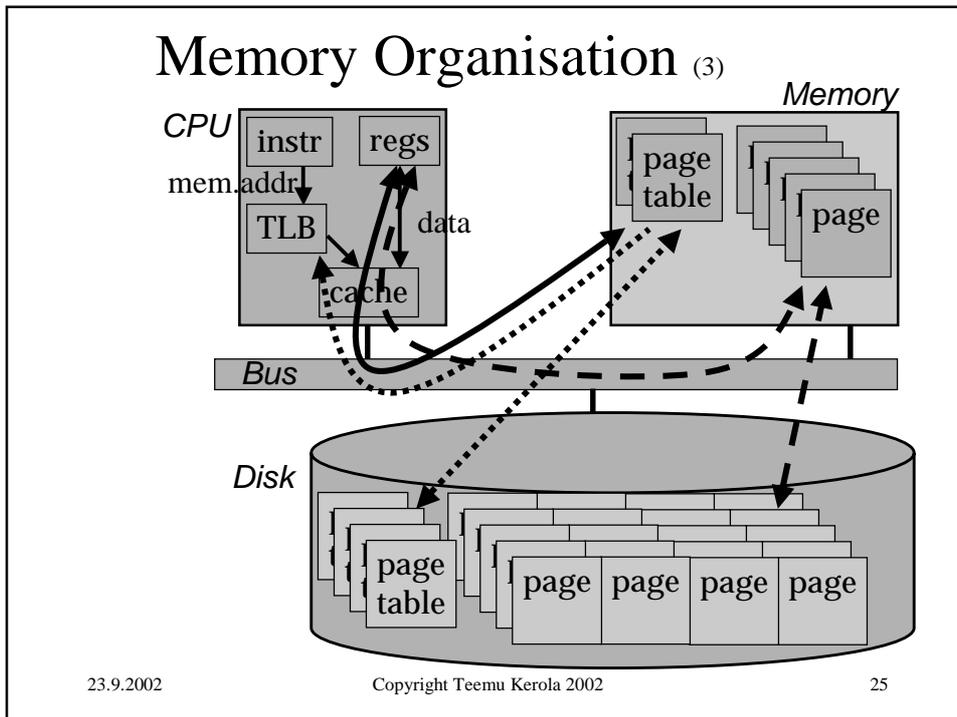
Fig. 8.18 (Fig. 7.19 [Stal99])

- “Hit” on TLB?
  - address translation is in TLB - real fast
- “Miss” on TLB?
  - must read page table entry from memory
  - takes time
  - cpu waits idle until it is done
- Just like normal cache, but for address mapping
  - implemented just like cache
  - instead of cache line data have physical address
  - split TLB? 1 or 2 levels?

23.9.2002

Copyright Teemu Kerola 2002

24



## TLB and Cache <sup>(3)</sup>

- Usually address translation first Fig. 8.19  
and then cache lookup (Fig. 7.20 [Stal99])
- Cache can be based on virtual addresses
  - can do TLB and cache lookup simultaneously
  - faster
- Implementations are very similar
  - TLB often fully associative
    - optimised for temporal locality (of course!)

23.9.2002

Copyright Teemu Kerola 2002

27

## TLB vs. Cache

### TLB Miss

- CPU waits idling
- HW implementation
- Invisible to process
- Data is copied from memory to TLB
  - from page table data
  - from cache?
- Delay 4 (or 2 or 8?) clock cycles

### Cache Miss

- CPU waits idling
- HW implementation
- Invisible to process
- Data is copied from memory to cache
  - from page data
- Delay 4 (or 2 or 8?) clock cycles

23.9.2002

Copyright Teemu Kerola 2002

28

## TLB Misses vs. Page Faults

### TLB Miss

- CPU waits idling
- HW implementation
- Data is copied from memory to TLB (or from cache)
- Delay 1-4 (?) clock cycles



### Page Fault

- Process is suspended and cpu executes some other process
- SW implementation
- Data is copied from disk to memory
- Delay 30 ms (?)



23.9.2002

Copyright Teemu Kerola 2002

29

## Virtual Memory Policies <sup>(3)</sup>

- Fetch policy (noutopolitiikka)
  - demand paging: fetch page only when needed 1st time
  - working set: keep all needed pages in memory
  - prefetch: guess and start fetch early
- Placement policy (sijoituspolitiikka)
  - any frame for paged VM
- Replacement policy (poistopolitiikka)
  - local, consider pages just for this process for replacement
  - global, consider also pages for all other processes
  - dirty pages must be written to disk (likaiset, muutetut)

23.9.2002

Copyright Teemu Kerola 2002

30

## Page Replacement Policy <sup>(2)</sup>

- Implemented in SW
- HW support
  - extra bits in each page frame
  - M = Modified
  - R = Referenced
    - set (to 1) with each reference to frame
    - reset (to 0) every now and then
      - special (privileged) instruction from OS
      - automatically (E.g., every 10 ms)
  - Other counters?

23.9.2002

Copyright Teemu Kerola 2002

31

## Page Replacement Policies <sup>(6)</sup>

- OPT - optimal
- NRU - not recently used
- FIFO - first in first out
  - 2nd chance
  - clock
- Random
- LRU - least recently used
  - complex counter needed
- NFU - not frequently used

(sivunpoisto-  
algoritmit)

OS  
Virtual Memory  
Management

23.9.2002

Copyright Teemu Kerola 2002

32

## Thrashing

- Too high mpl
- Too few page frames per process
  - E.g., only 1000? 2000?
  - Less than its working set
- Once a process is scheduled, it will very soon reference a page not in memory
  - page fault
  - process switch

23.9.2002 Copyright Teemu Kerola 2002 33

## Trashing

(ruuhkautuminen)

CPU 1.0 utilization

(käyttösuhde)

Higher mpl  
=> less physical memory per process!

CPU 100% busy swapping processes!  
No real work is done!

(moniajoaste)

mpl (multiprogramming level)

- How much memory per process?  
- How much memory is needed?

23.9.2002 Copyright Teemu Kerola 2002 34

## Page Fault Frequency (PFF) Dynamic Memory Allocation

- Two bounds: L=Lower and U=Upper
- Physical memory split into fixed size pages
- At every page fault
  - T=Time since previous page fault
  - if  $T < L$  then give process more memory
    - 1 page frame? 4 page frames?
  - if  $U < T$  then take some memory away
    - 1 page frame?
  - if  $L < T < U$  then keep current allocation

23.9.2002

Copyright Teemu Kerola 2002

35

## Multi-level paging/segmentation

- Segmented paging
    - address logically split into segments and then physically into pages
    - protection may be at segment level
  - Multiple level paging
    - large address space may result in very large page tables
    - solution: multiple levels of page tables
- 01101 01100110 10110000  
 segm page byte offset
- Fig. 5.43 [HePa96]
- VM implementation may not utilize them all
  - VM implementation may seem to use more levels than there are (e.g., Linux 3 levels on 2-level Intel arch.)
    - nr of actual levels in mem. management macros

23.9.2002

Copyright Teemu Kerola 2002

36

## VM Summary

- How to partition memory?
  - Static or dynamic size (amount)
- How to allocate memory
  - Static or dynamic location
- Address mapping
- HW help (TLB) for address translation
  - before or concurrently with cache access?
- VM policies
  - fetch, placement, replacement

23.9.2002

Copyright Teemu Kerola 2002

37

