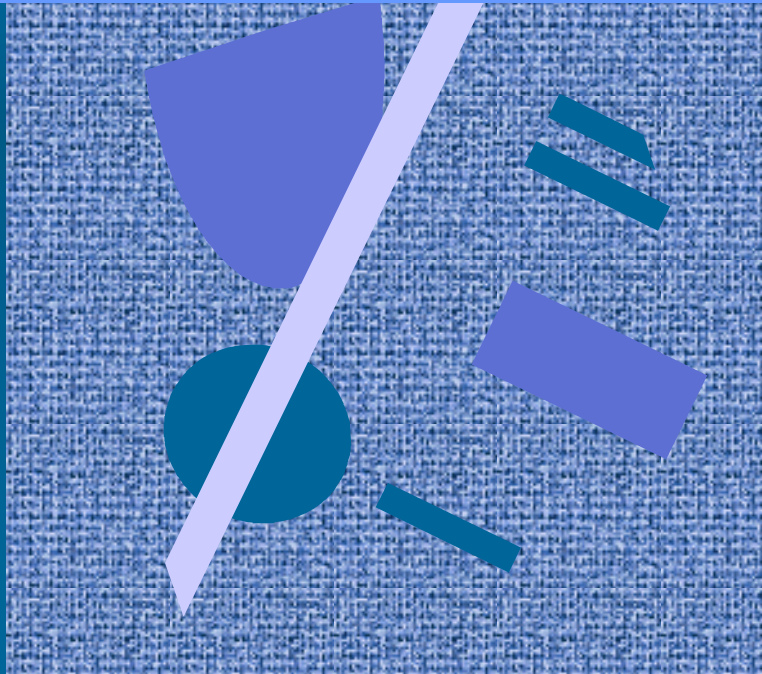


Virtual Memory (VM)

Ch 7.3



Memory Management
Address Translation
Paging
Hardware Support
VM and Cache

Virtual Memory

(virtuaalimuisti)

Ch 7.3

- 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 data references are for data in memory

Other Problems Often Solved with VM ⁽³⁾

- 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 there is?

Memory Management Problem (4)

- 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 during the run time?
- How is that memory managed?
- How is that memory referenced?

Partitioning ⁽³⁾

- How much physical memory for each process?
- Static (fixed) partitioning (staattiset partitiot)
 - amount of physical memory determined at process creation time
- 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

Static Partitioning

- Equal size - give everybody the same amount

Fig. 7.14

- fixed size - big enough for everybody
- need more? Can not run!
- internal fragmentation

(sisäinen pirstoutuminen)

- Unequal size

- external fragmentation

(ulkoinen pirstoutuminen)

Fig. 7.15

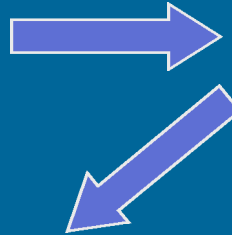
Dynamic Partitioning ⁽³⁾

- Process must be able to run with different amounts of main memory
 - all of memory space is **not** in physical 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

Address Mapping ⁽⁴⁾ (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!)

Address Mapping

Textual machine language: logical address

1312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1,102510 or

101312: LOAD R1, 2510 ??

physical address (constant?)

- Want: $R1 \leftarrow \text{Mem}[102510]$ or $\text{Mem}[2510]$?
- Who makes the mapping? When?

Address Mapping (2)

- At program load time
 - loader
 - static address binding
- At program execution time
 - cpu
 - with every instruction
 - dynamic address binding
 - swapping
 - virtual memory

(lataaja)

(staattinen
osoitteiden sidonta)

(dynaaminen
osoitteiden sidonta)

Swapping (4)

(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

VM Implementation (2)

- Methods
 - base and limit registers
 - segmentation
 - paging
 - segmented paging
- Hardware support
 - MMU - Memory Management Unit
 - varies with different methods

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
- *BASE* and *LIMIT* registers in MMU
 - all addresses logical in machine instructions
 - address mapping for address (x):
 - check: $x < LIMIT$
 - physical address: $BASE+x$

Segmentation (5)

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

011001 1010110000

segment byte offset

(lisäys)

Segmentation Address Mapping

- 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

Paging

- Process address space divided into (relatively small) equal size pages
 - no logical entities, only physical
- 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

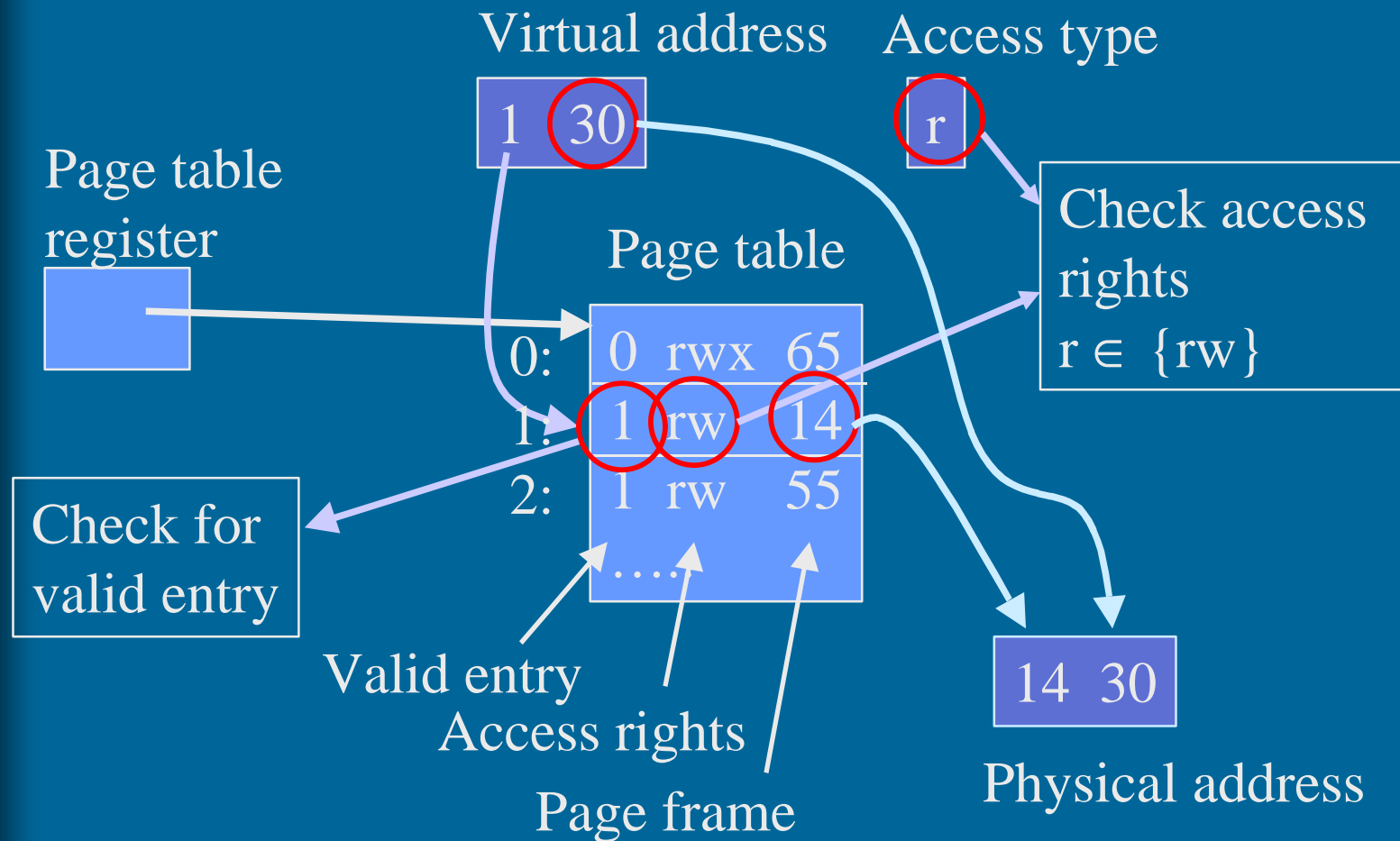
page byte offset

(lisäys)

Paged Address Mapping

- Page table
 - maps page nr to physical page frame
- Physical address:
 - find entry in page table
 - physical address: page address + byte offset

Paged Address Translation (4)



Page fault interrupt

Page Fault (12)

Stop execution

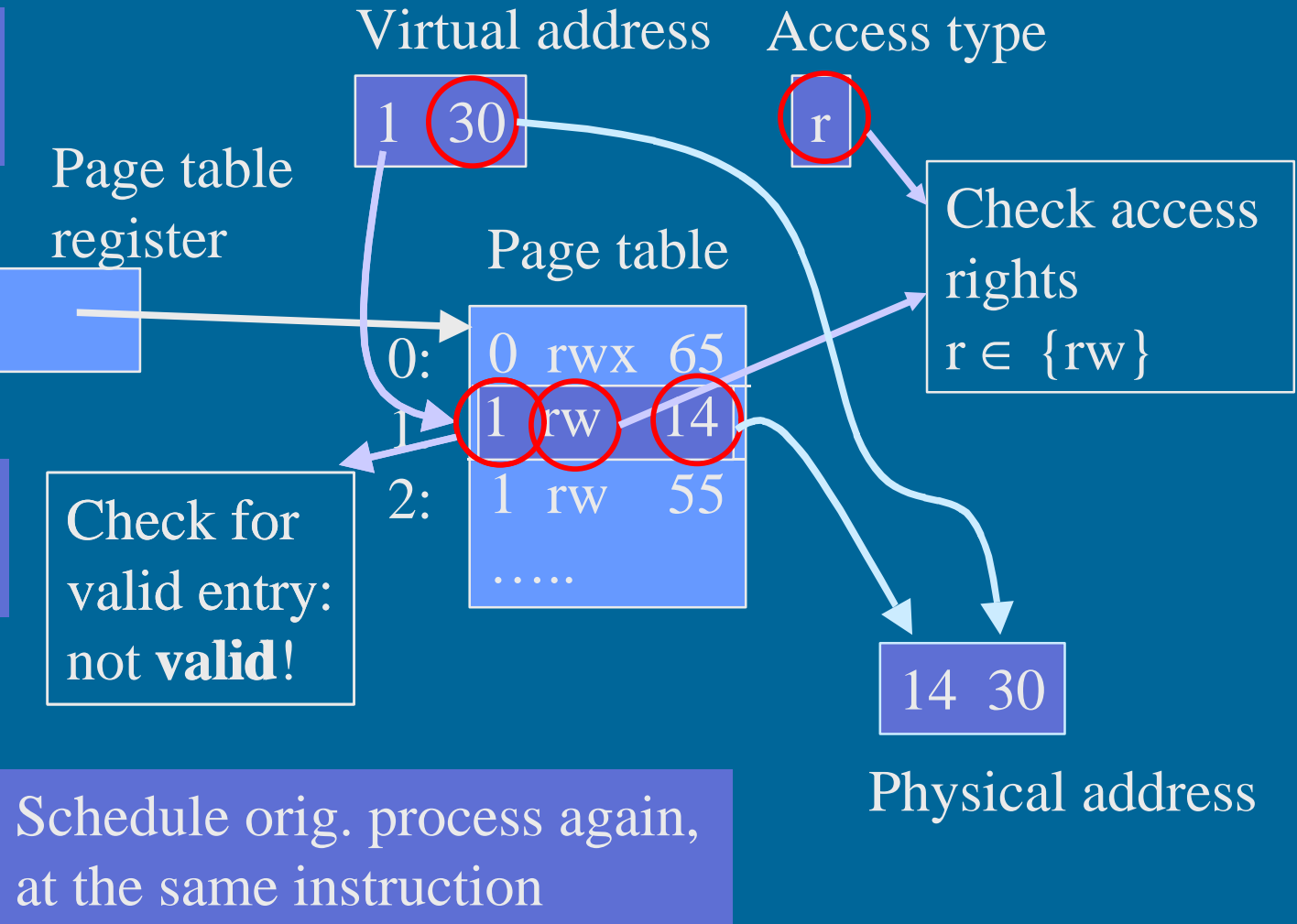
Initiate reading page 1 from disk

Schedule next process to run

I/O interrupt

Page 1 read, update page table

Make orig. process ready-to-run



Paging

- Physical memory partitioning
 - discontinuous areas
- Page tables
 - each process has its own
 - located in memory
 - can be very big
 - entry for each page in address space
- Inverted page table
 - entry for each page in memory

Fig. 7.16

Fig. 7.18

Address Translation ⁽³⁾

- 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?

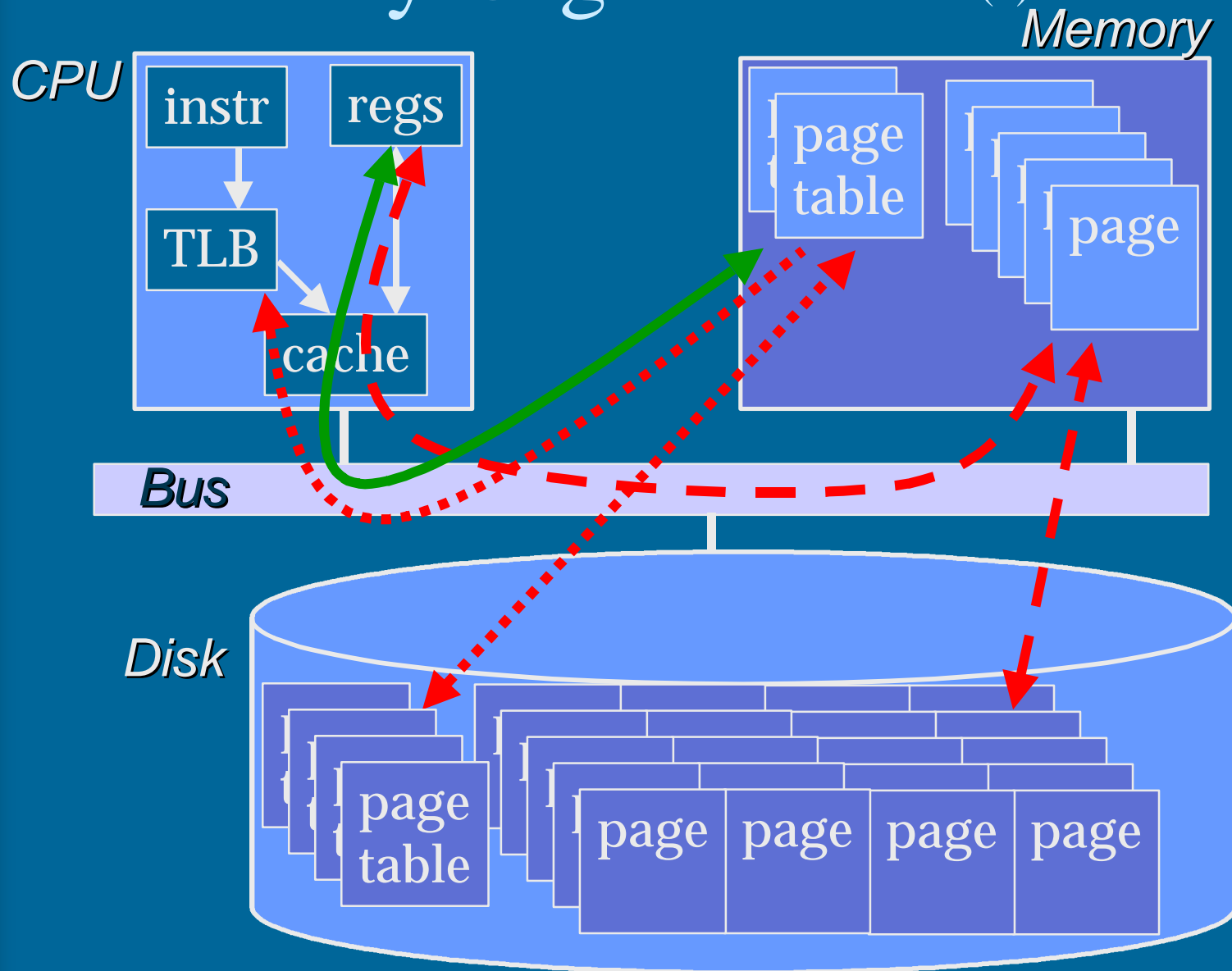
(osoitteen-
muunnos-
taulukko)

Translation Lookaside Buffer ⁽³⁾

- “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

Fig. 7.19

Memory Organisation ⁽³⁾



TLB Example (6)

Physical address

0x00B6C8E6 046

page offset

ReadW I2, 0xAB00C7DA 046

tag
28

page frame
32

index	tag	page frame
0000:		
....		
....		
0111:		
1000:		
1001:		
1010:	AB00C7D	00B6C8E6
....		

AB00C7D | A

?
=
Match

Correct address mapping found

TLB and Cache ⁽³⁾

- Usually address translation first and then cache lookup
- 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

Fig. 7.20

TLB vs. Cache

TLB Miss

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

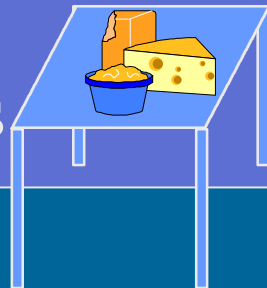
Cache Miss

- 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

TLB Misses vs. Page Faults

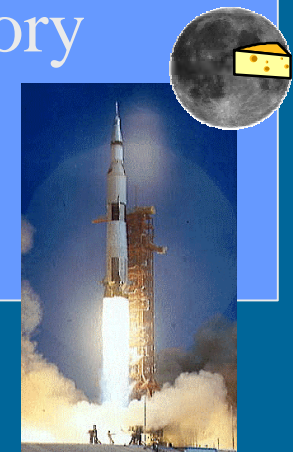
TLB Miss

- CPU waits idling
- HW implementation
- Data is copied from memory to TLB
- Delay 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 (?)



Virtual Memory Policies ⁽³⁾

- Fetch policy (noutopolitiikka)
 - demand paging: only when needed 1st time
 - working set: keep those needed 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
 - global, consider pages for all processes
 - dirty pages must be written to disk (likaiset, muutetut)

Page Replacement Policy (2)

- 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?

Page Replacement Policies

(sivunpoisto-
algoritmit)

- 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

OS
Virtual Memory
Management

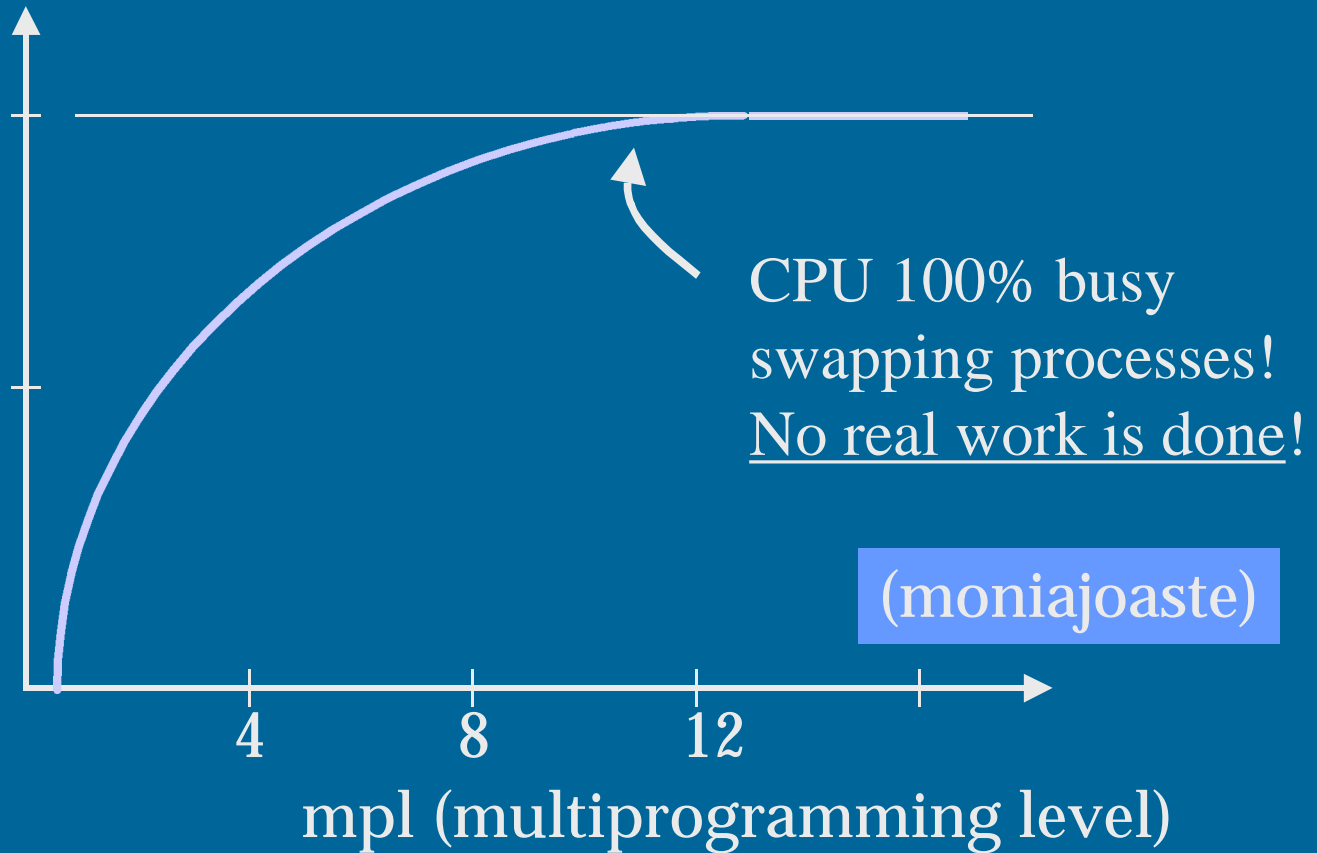
Trashing (3)

(ruuhkautuminen)

CPU 1.0
utilization

(käyttösuhde)

Higher mpl
⇒ less physical
memory
per process!



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

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

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 more memory
 - 1 page frame? 4 page frames?
 - If $T > U$ then take some memory away
 - 1 page frame?
 - If $L < T < U$ then keep current allocation

VM Summary (5)

- 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

-- End of Chapter 7.3: Virtual Memory --

Fig. 5.47 from
Hennessy-Patterson,
Computer Architecture

Alpha AXP 21064
memory hierarchy

Fully assoc, 12 entry
instruction TLB

8 KB, direct mapped,
256 line (each 32B)
instruction cache

2 MB, 64K line (each 32B)
direct mapped, unified,
write-back L2 cache

Fully assoc,
32 entry
data TLB

8 KB,
direct
mapped,
256 line
(each 32B)
data cache

main memory

paging disk (dma)

