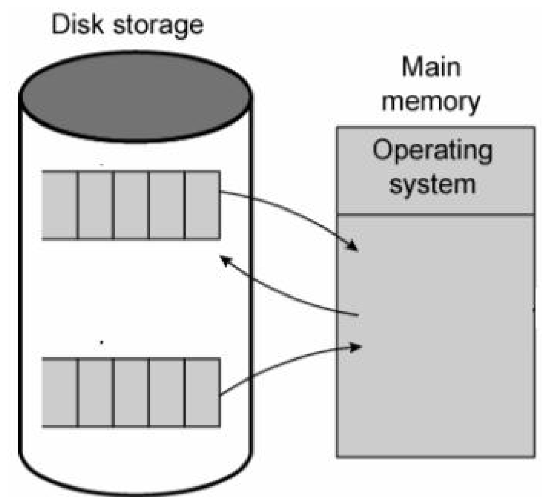
Tietokoneen rakenne

Muistin-hallinta

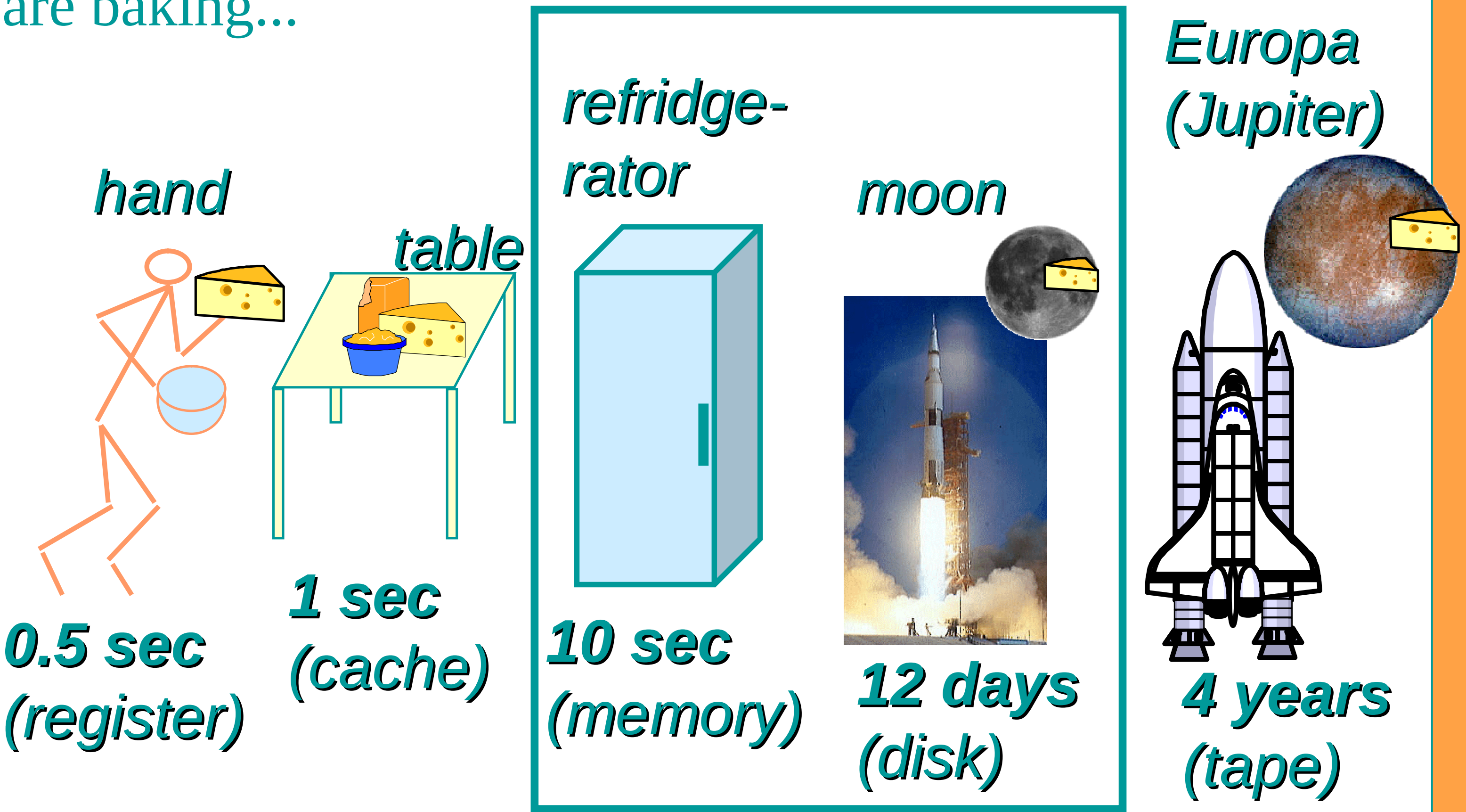


Stallings: Ch 8.3-8.6

- Heittovaihto vs. Virtuaalimuisti
- Esim: Pentium

Teemu’s Cheesecake

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



Virtual Memory

(virtuaalimuisti)

- **Problem: How can I make my (main) memory as big as my disk drive?**
- **Answer: Virtual memory**
  - keep only <u>most probably referenced data</u> 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

Other Problems Often Solved with VM

- 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?

Memory Management Problem

- **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?**

Partitioning

- 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

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

Sta06 Fig. 8.13 (a)

Sta06 Fig. 8.13 (b)

Sta06 Fig. 8.14

Fragmentation

- **Internal fragmentation** (sisäinen pirstoutuminen)
  - unused memory inside allocated block
  - e.g., equal size fixed memory partitions

    Sta06 Fig. 8.13 (a)
- **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

    Sta06 Fig. 8.13 (b)

    Sta06 Fig. 8.14

Dynamic Partitioning

- **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

Address Mapping (4)

(osoitteen muuntos)

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 (2)

logical address

Textual machine language:

1312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1,102510 or

101312: LOAD R1, 2510 ??

physical address (constant?)

logical addr

- Want: R1 ← Mem[102510] or Mem[2510] ?
- Who makes the mapping? When?

Address Mapping (2)

- 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

Heittovaihto (swapping)

Lisätietoja KJ-kurssilla

- **Prosessilla yhtenäinen muistialue**
  - Prosessi joko muistissa tai levyllä
  - Prosessinkuvaaja (PCB) aina muistissa
- **Ajonaikainen osoitemuunnos**
  - Looginen osoite ž fyysinen muistiosoite
- **Laitteistotuki = MMU**
  - Kanta- ja rajarekisteri
  - "Bounds exceeded"-keskeytys
- **KJ**
  - Kirjanpito vapaista muistialueista
  - Prosessien siirto levyltä muistiin / muistista levylle
  - Prosessin vaihto: kanta- ja rajarekisterin asetus
  - Virheellinen muistiviite: tapa prosessi

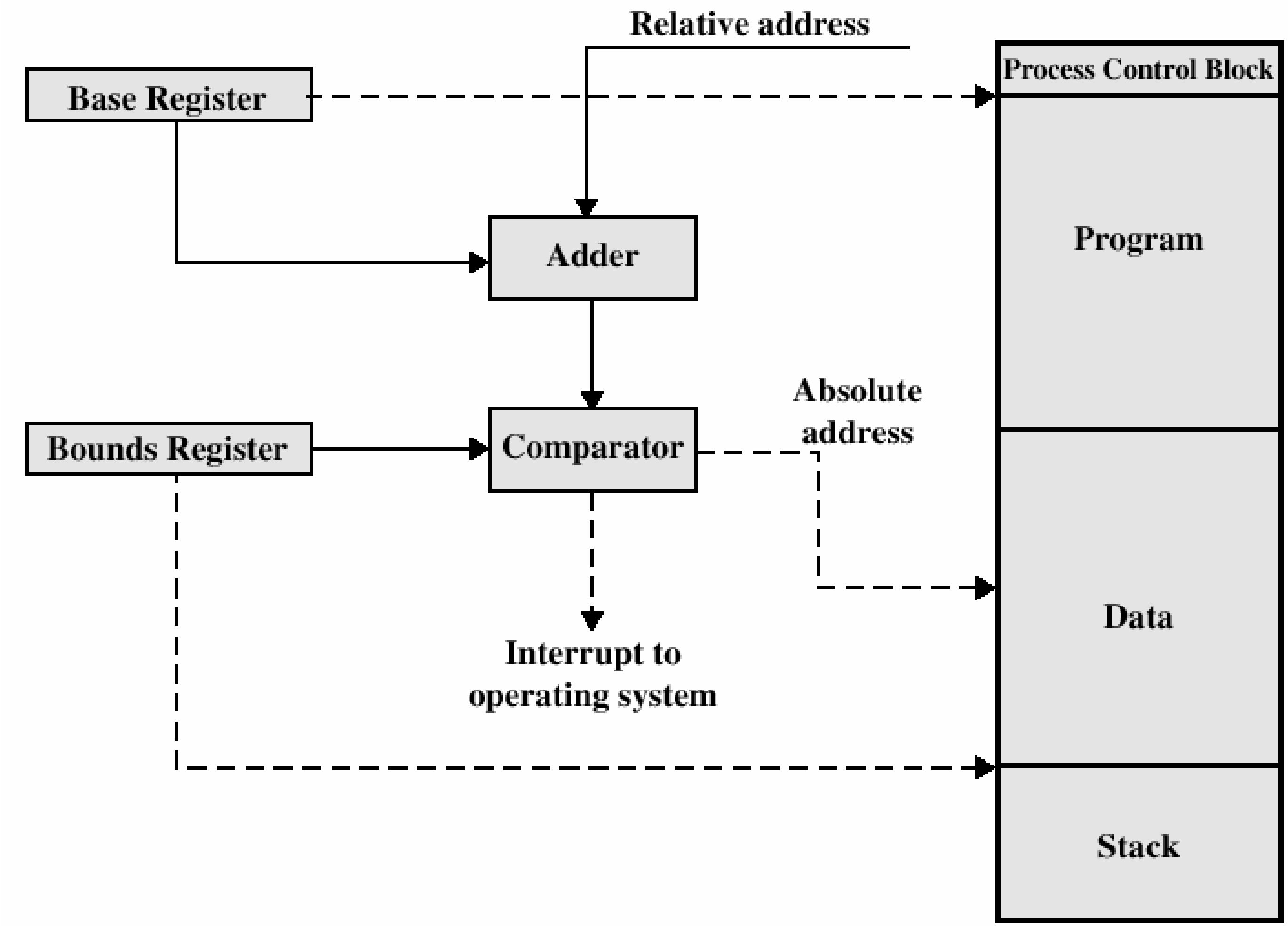
VM Implementation

- **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

Base and Limit Registers

- **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
  - Exec. time address mapping for address (x):
    - Check: $x < LIMIT$
    - Physical address: $BASE+x$

Osoitteenmuunnos rajarekistereitä käyttäen



(Sta05 Operating Systems Fig 7.8)

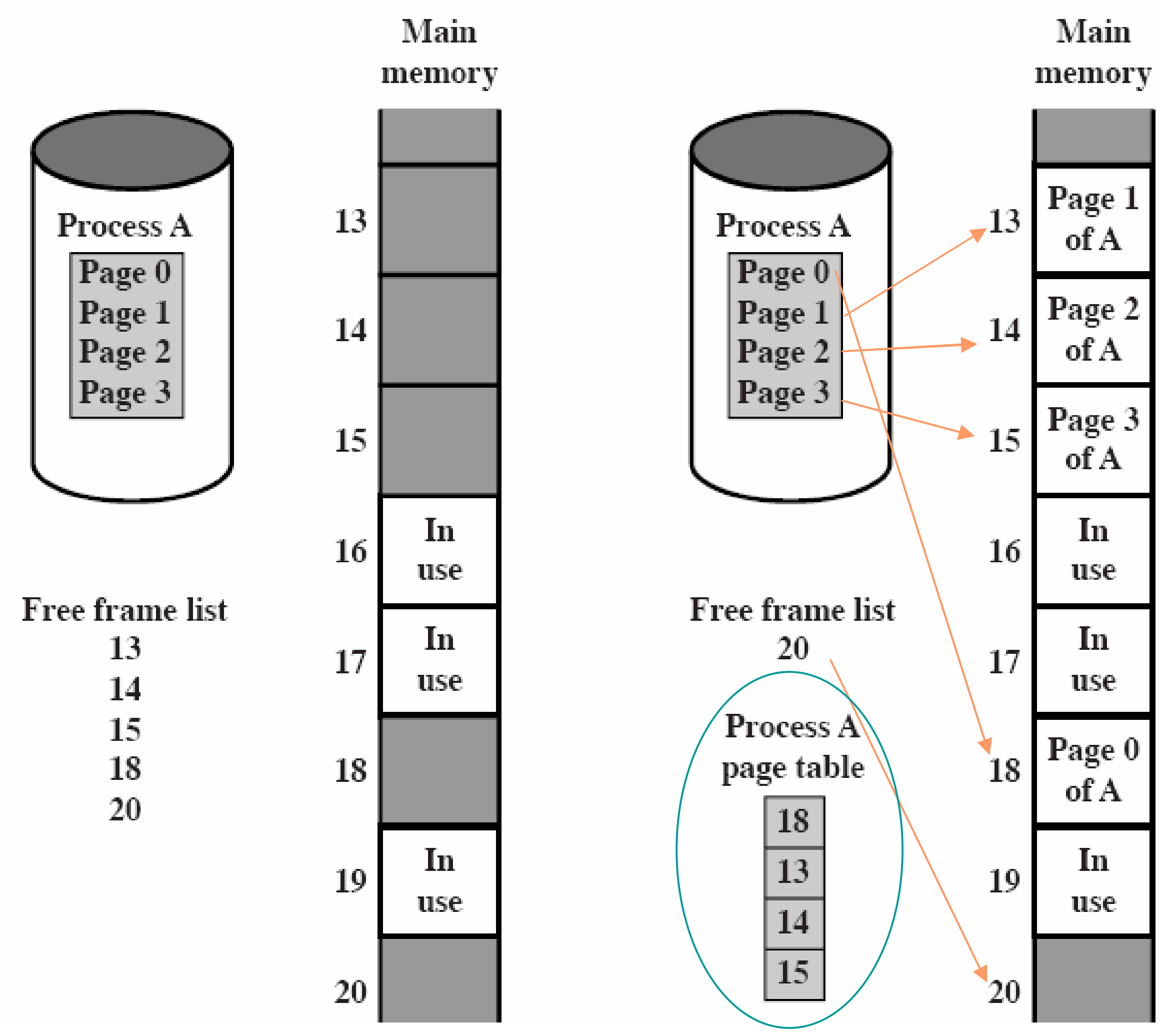
Virtuaalimuisti

- Vain tarvittavat prosessin palat muistissa, ei tarvitse sijaita peräkkäin muistissa
  - Tarvenouto
- Palojen koko?
  - Vakiokokoiset palat = Sivutus
  - Palojen koko vaihtelee = Segmentointi
  - Yhdistettynä = Sivutettu segmentointi
- KJ:n kirjanpito
  - Sivutilataulu (page frame table)
    - Mitkä sivutilat vapaita, mitkä varattuja?
  - Jokaisella prosessilla oma sivutaulu (page table)
    - Onko sivu muistissa vai levyllä? Presence-bitti
    - Missä sivutilassa sivu majailee?
    - Muuta kontrollitietoa? Viitebitit: Modified, R=Referenced

KJ kurssin pureskeltavaa

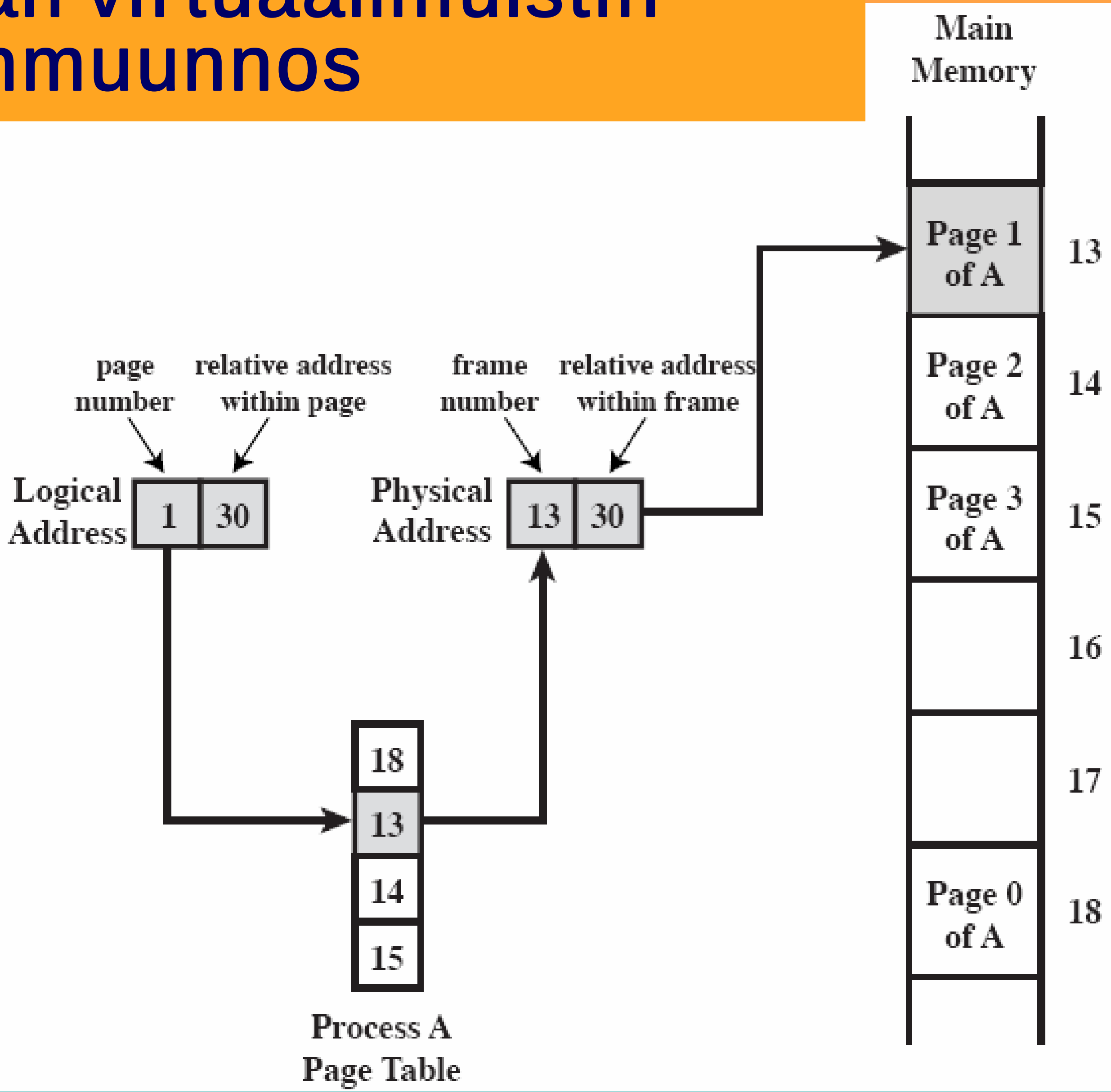
Sivutus "yleisintä" ð nyt vain siitä

Virtuaalimuisti: Sivutus



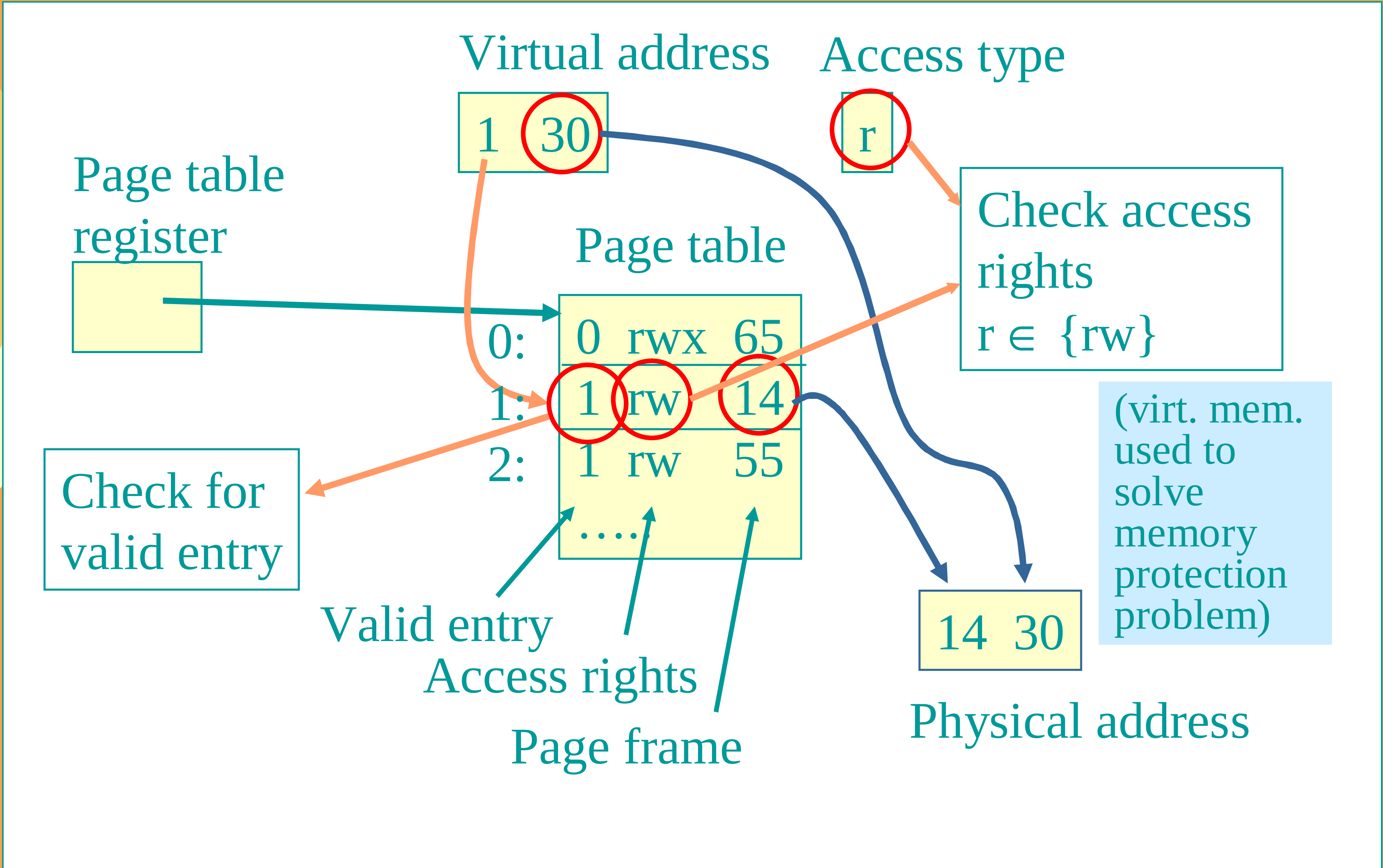
(Sta06 Fig 8.15)

Sivuttavan virtuaalimuistin osoitteenmuunnos



(Sta06 Fig 8.16)

Paged Address Translation (4)



Page fault interrupt

Page Fault (12)

Stop execution

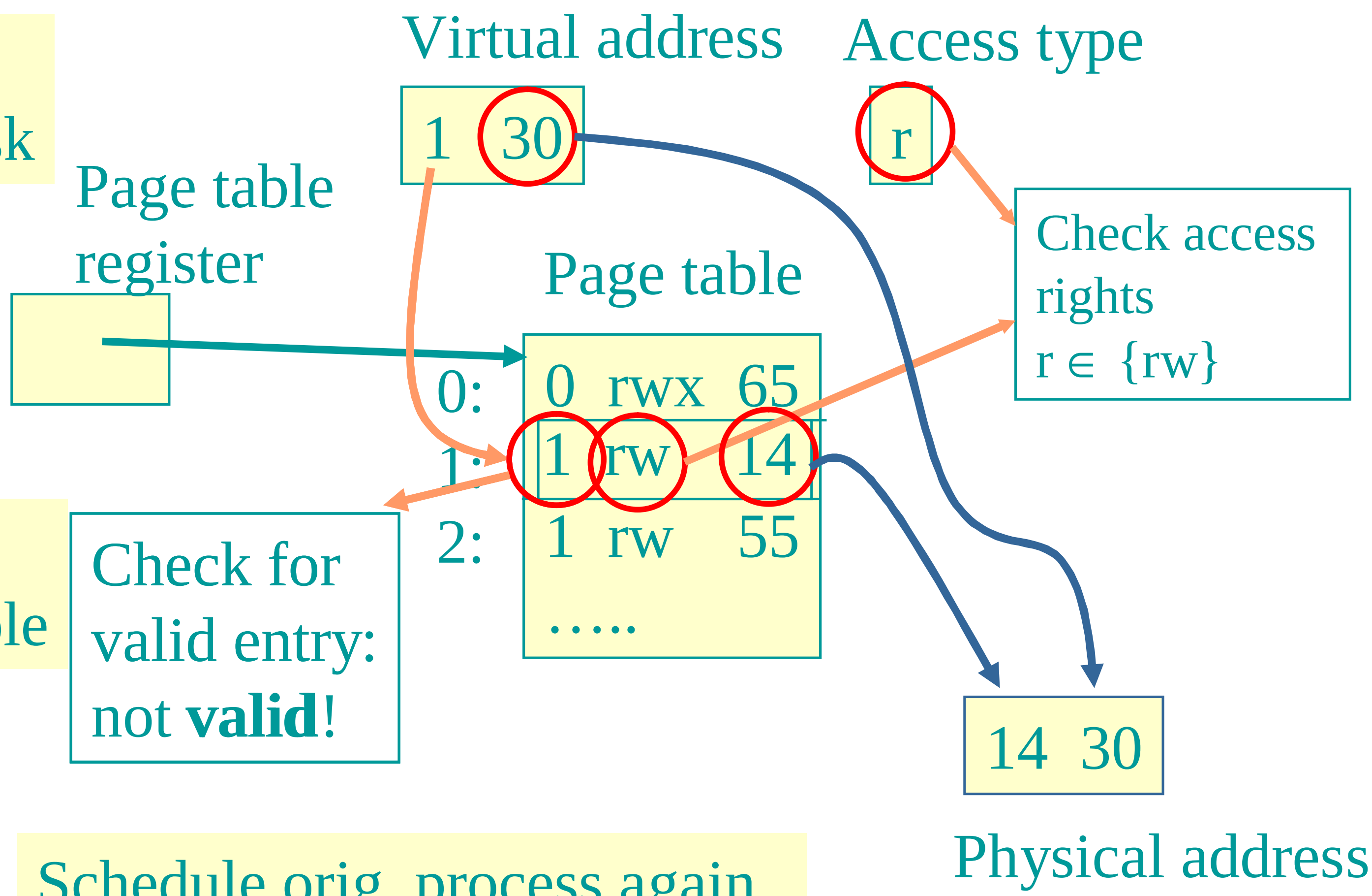
Initiate reading page 1 from disk

Schedule next process to run

I/O interrupt

Page 1 read, update page table

Make orig. process ready-to-run



Schedule orig. process again, at the same instruction

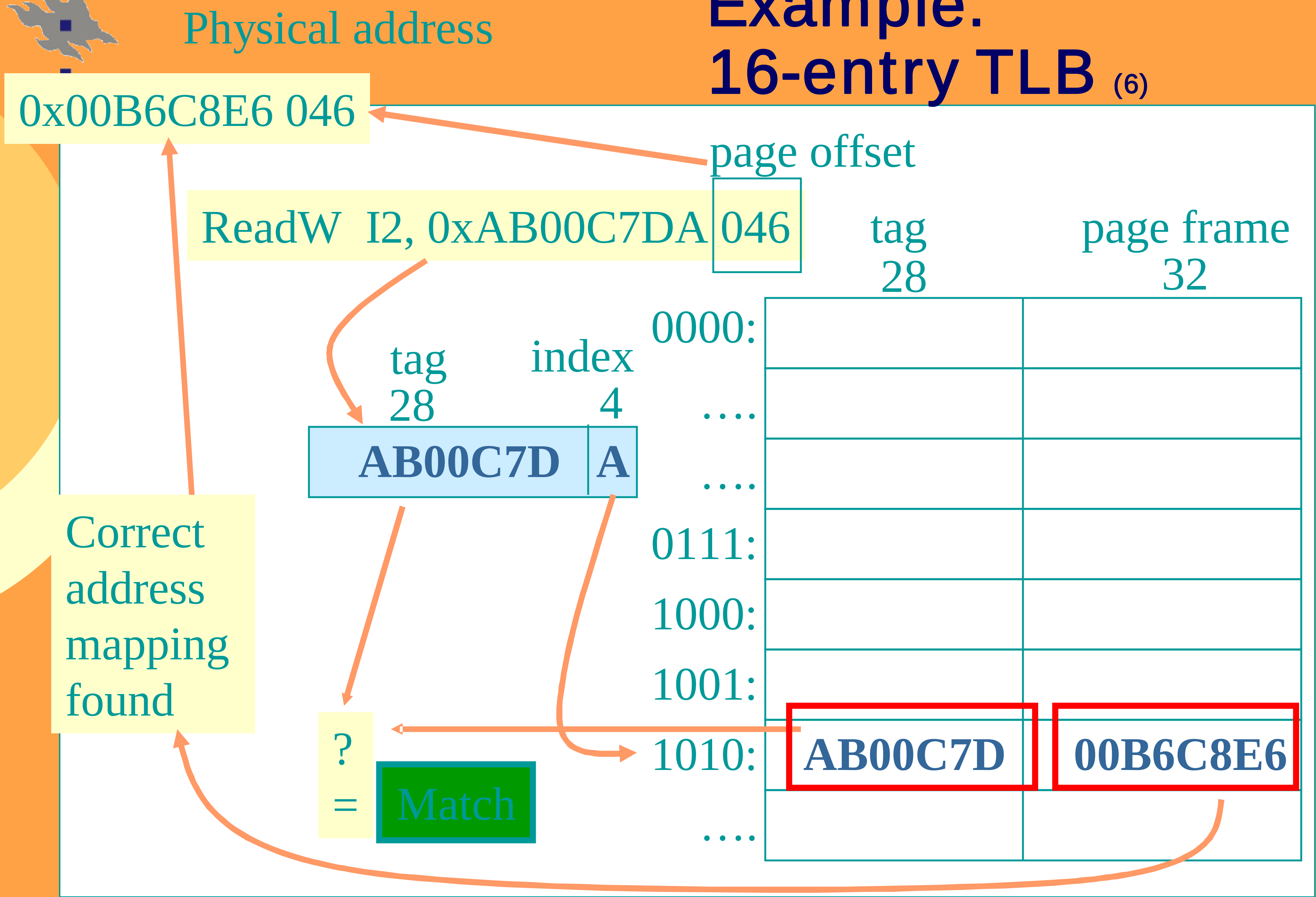
Virtuaalimuisti: TLB

Translation Lookaside Buffer

- Osoitemuunnos jokaiselle muistiviittaukselle, vähintään kerran per käsky
- Sivutaulun alkio muistissa

  = ylimääräinen muistinouto?
  - Liian hidasta
- Ratkaisu
  - Paikallisuus! Sitähän tarvitaan het'kohta uudestaan
  - Pidä tallessa edellisissä muunnoksissa tarvitut sivutaulun alkiot
- TLB, osoitemuunnospuskuri
  - Vrt. välimuisti
  - Nopea rekisterijoukko (esim. Pentium: 32 alkiota)
  - Assosiatiivinen haku
  - Osumatodennäköisyys 99.9% ? (eli lähes aina!)

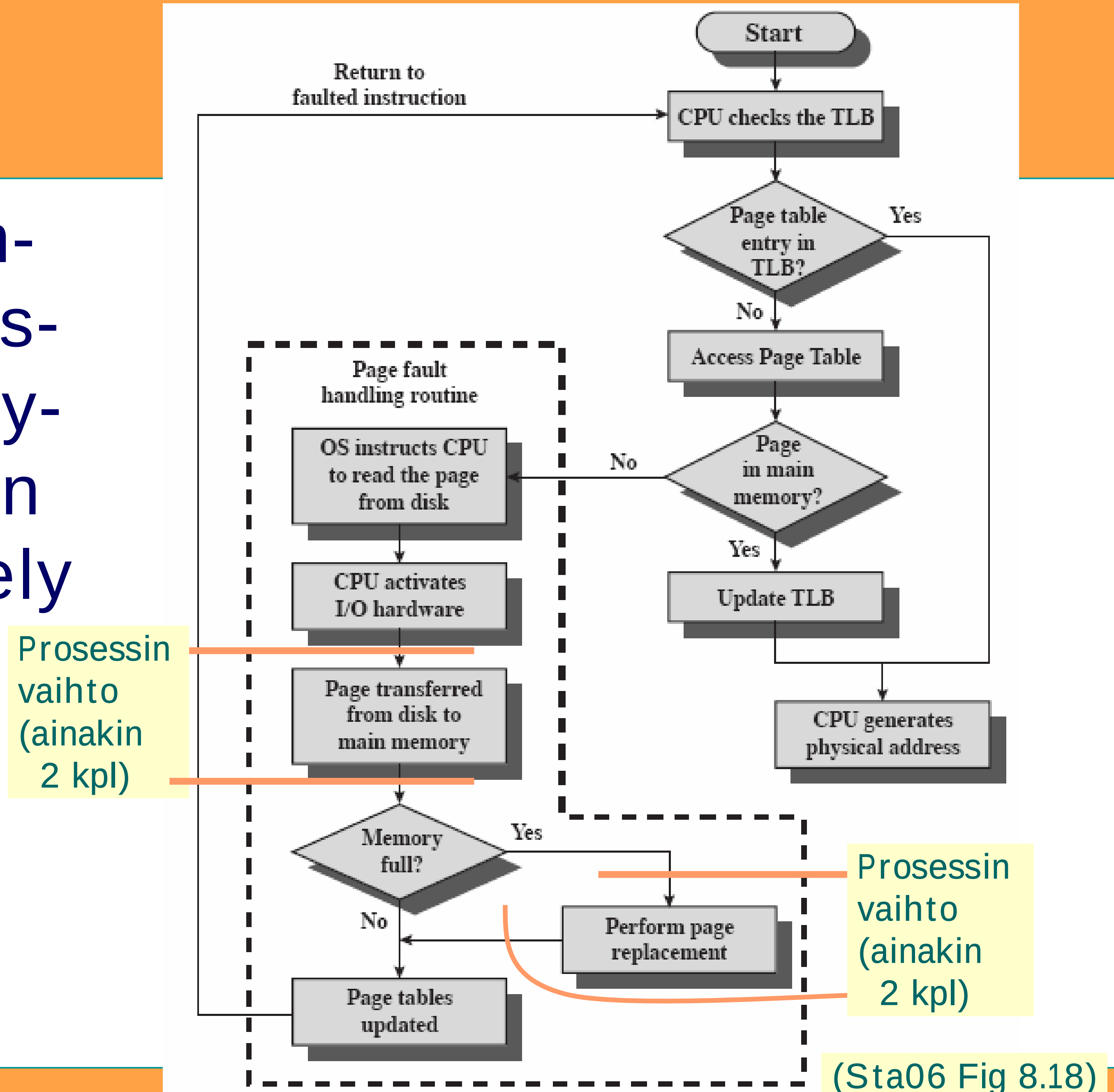
Example: 16-entry TLB (6)



Translation Lookaside Buffer

- “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?

Sivun-puutos-keskey-tyksen käsittely

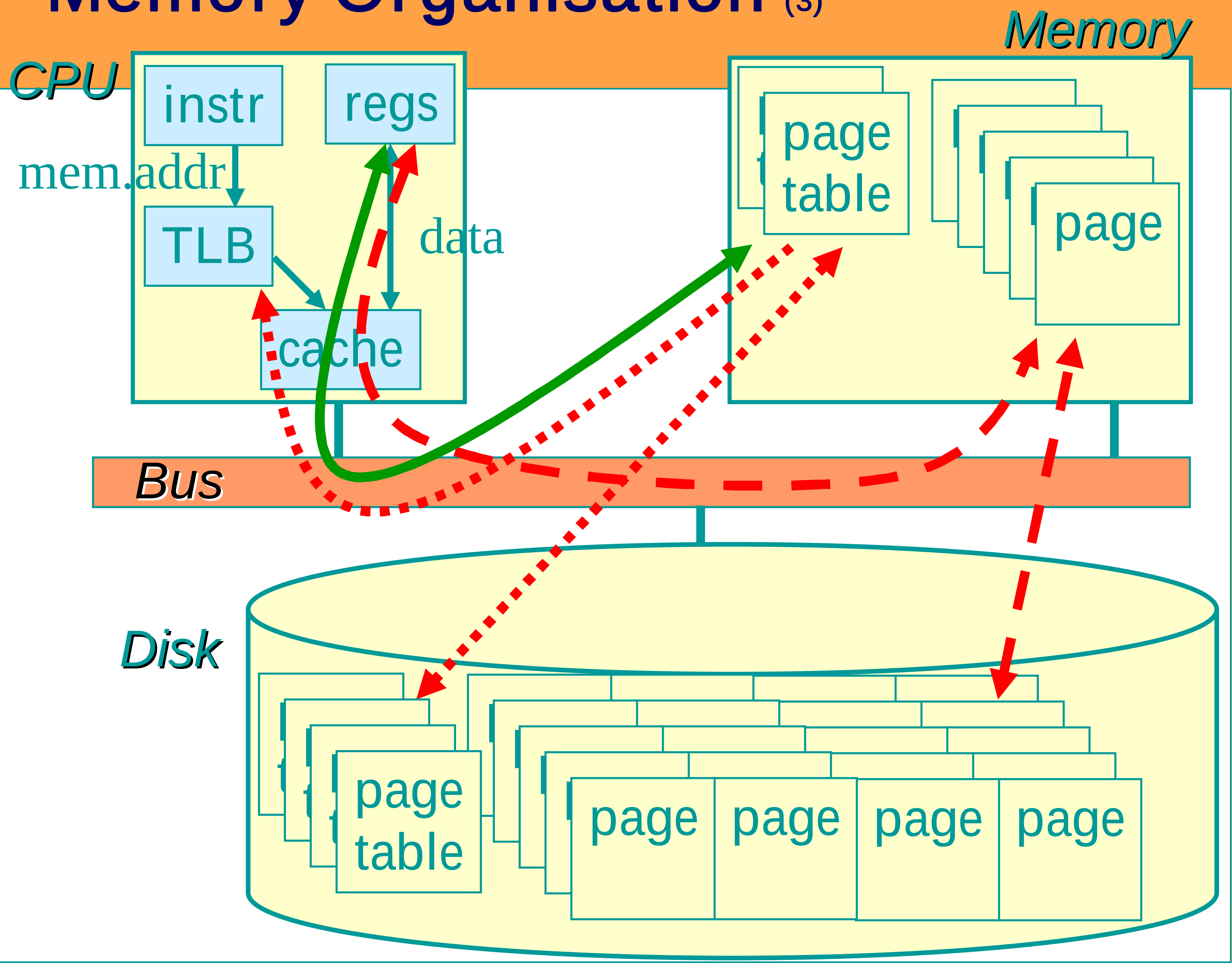


(Sta06 Fig 8.18)

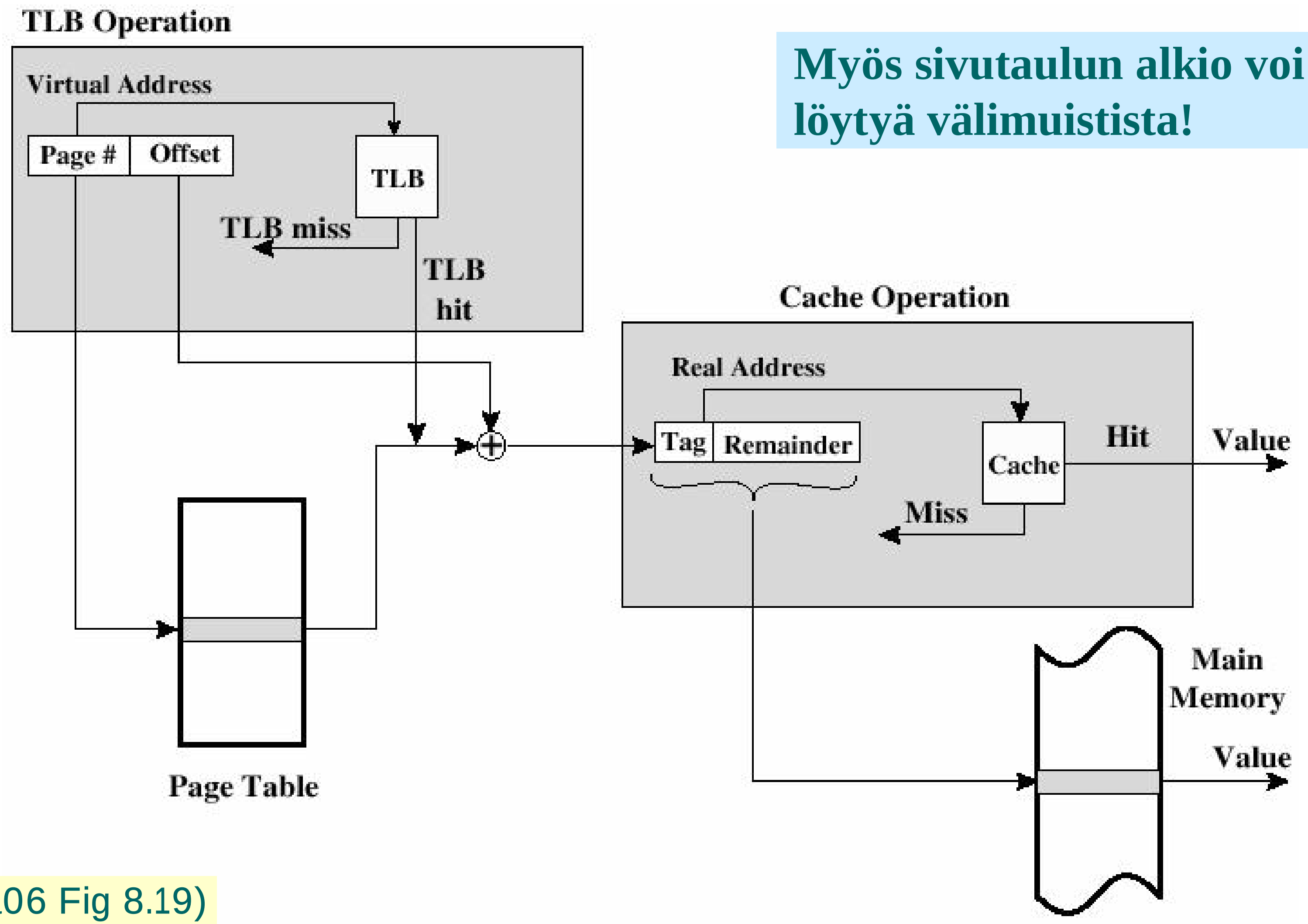
Virtuaalimuisti

- **Laitteistotuki = MMU ja sen erikoisrekisterit**
  - PTR (page table register)
    - Sivutaulun fyysinen alkuosoite (tuotu PCB:stä)
  - TLB (translation lookaside buffer)
    - Aiemmissa osoitteenmuunnoksissa käytettyjä tietoja = sivutaulun alkioita
  - “Page fault” –keskeytys
  - Viitebittien ylläpito
- **Prosessin vaihtuessa**
  - PTR • sivutaulun fyysinen alkuosoite
  - TLB:n vanha sisältö invalidoitava
    - TLB:n alkioissa Validi-bitit
    - Muuttuneet alkiot takaisin muistiin (“cache block”)

Memory Organisation (3)



TLB ja välimuisti



Myös sivutaulun alkio voi löytyä välimuistista!

(Sta06 Fig 8.19)

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

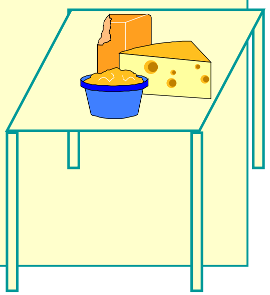
Cache Miss

- 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

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 (?)

Korvauspolitiikka

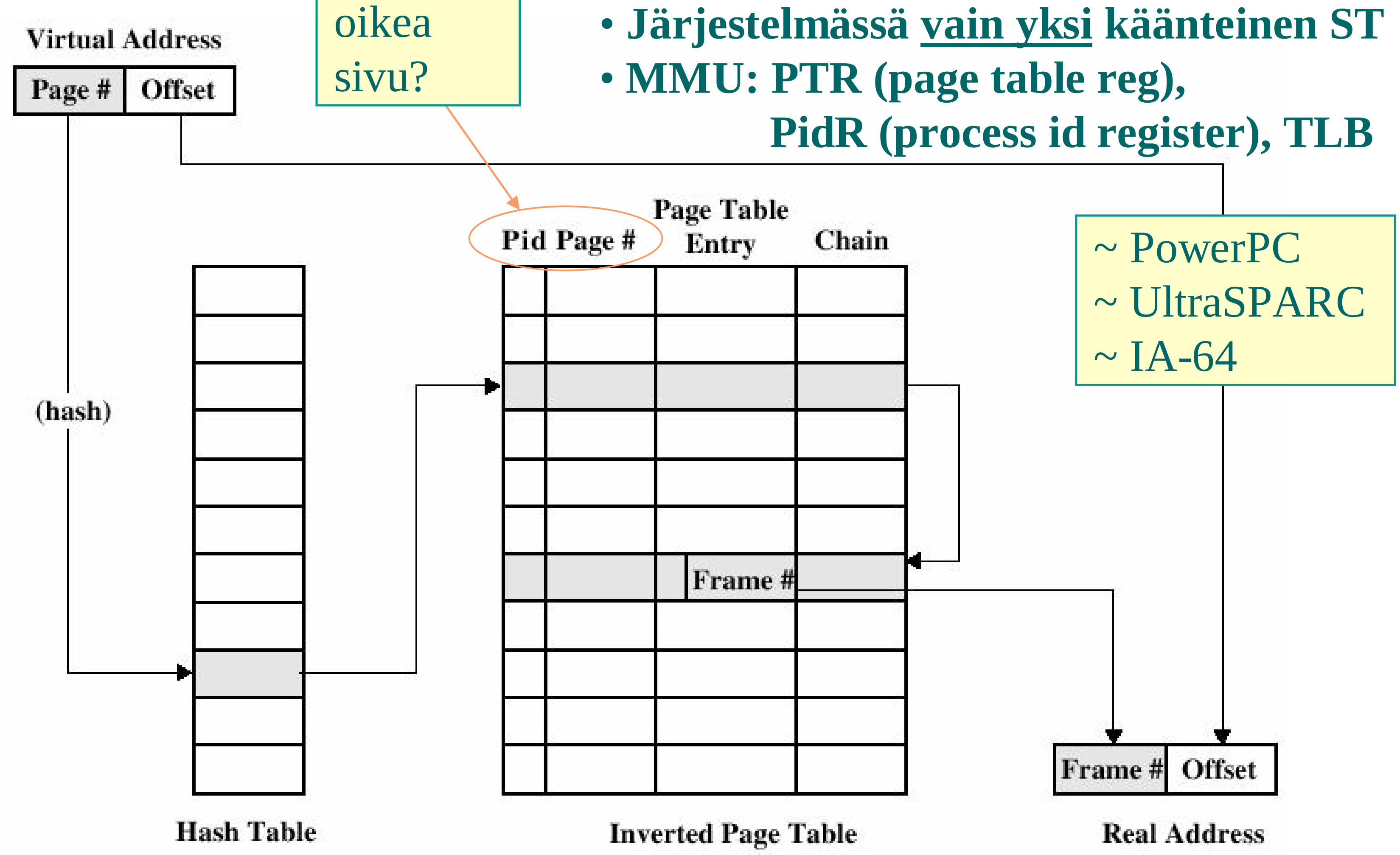
- Mikä sivu korvataan muistista, jos muistitilasta puutetta?

- Lokaalit / globaalit algoritmit
  - Prosessin omien sivujen joukosta
  - Kaikkein prosessien sivujen joukosta
- Algoritmeja
  - Clock, Second change, LRU, ...
- MMU
  - aseta viitattaessa Referenced=1,
  - aseta Modified=1, jos sivun sisältö muuttuu
- KJ
  - Nollaa bitit aika-ajoin
  - Korvaa tarvittaessa sellainen, jossa R=0, M=0
  - M=1 ð kirjoita muuttunut sivu levylle ennen uusiokäyttöä

KJ-kurssilla tarkemmin

Käänteinen sivutaulu

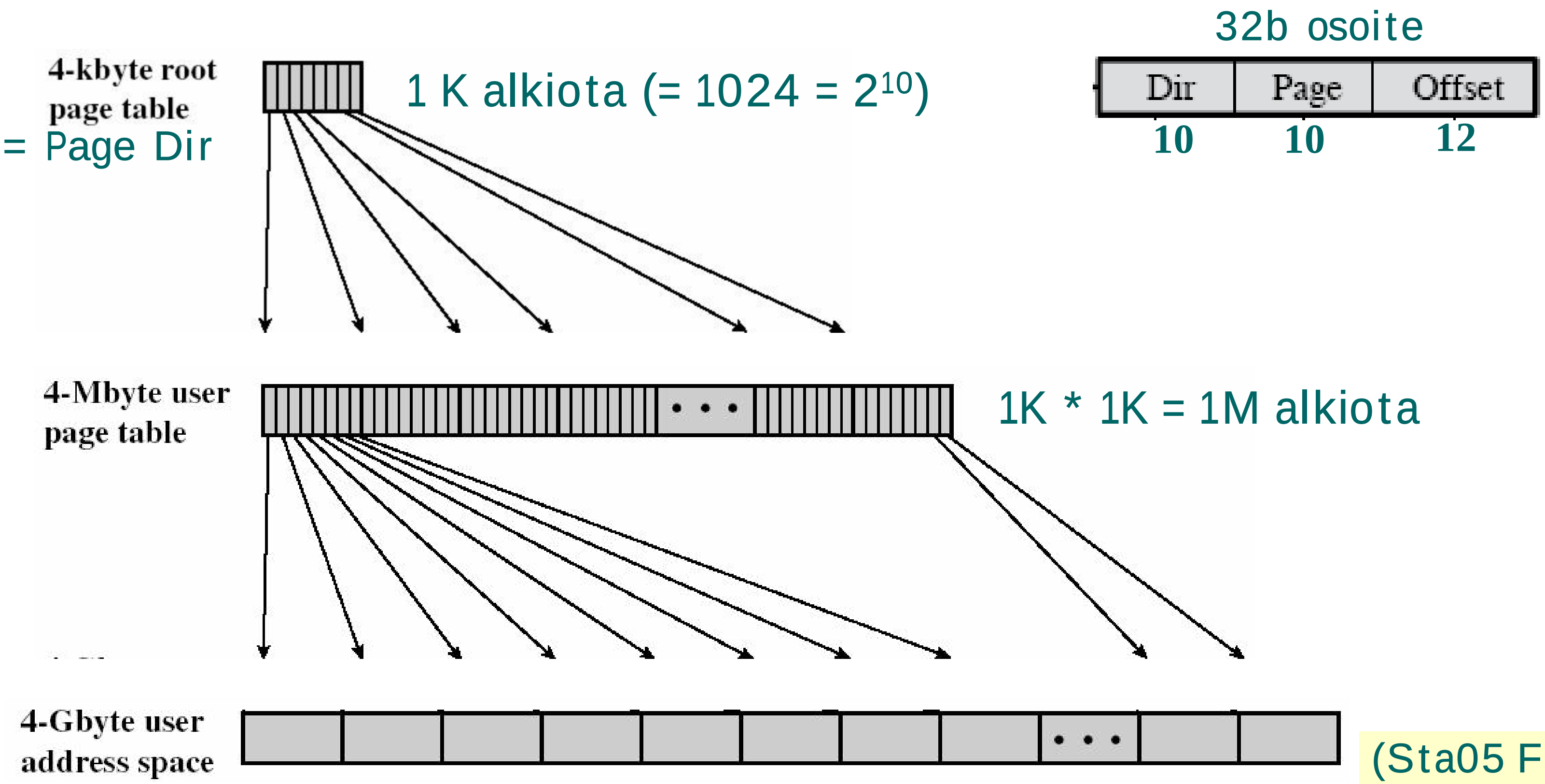
- Järjestelmässä vain yksi käänteinen ST
- MMU: PTR (page table reg), PidR (process id register), TLB



(Sta06 Fig 8.17 täydennettynä)

Monitasoinen sivutaulu (3)

- **Monet järjestelmät sallivat suuren virtuaaliosoiteavaruuden**
  - Myös ST jaetaan sivuihin, ja ST:n osia levyllä
  - Ylimmän tason ST mahtuu yhteen sivuun, aina muistissa



(Sta05 Fig 8.4)

Virtual Memory Policies (3)

- **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)

Tietokoneen rakenne

Esimerkiksi

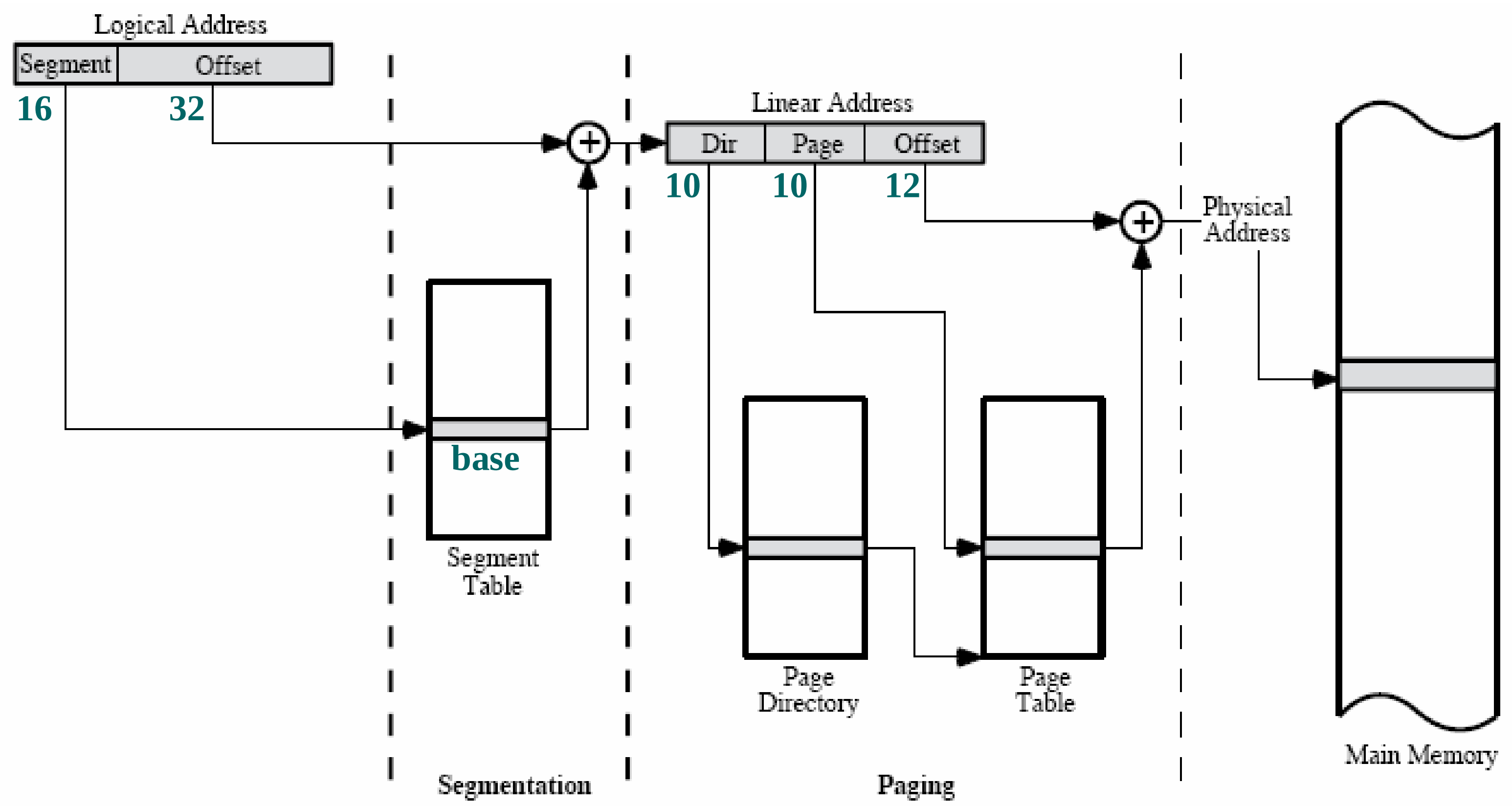
Pentium (IA-32)

Ks. myös Tan06

Pentiumin tuki muistinhallinnalle

- **Unsegmented unpaged, max 2^{32} = 4 GB**
  - Virtuaaliosoite = fyysinen osoite
  - Tehokas ð käyttöä reaaliaikajärjestelmissä
- **Unsegmented paged (Sivuttava), max 4 GB**
  - Lineaarinen osoiteavaruus
  - Sivu 4KB tai 4MB
  - Käyttöoikeudet sivukohtaisesti
- **Segmented unpaged (Segmentoiva), max 2^{48} = 64 TB**
  - Useita segmenttejä ð useita lineaarisia osoiteavaruuksia
  - Käyttöoikeudet segmenttikohtaisesti
- **Segmented paged (Sivuttava segmentointi), max 64 TB**
  - Muistinhallinta sivutusta käyttäen
  - Käyttöoikeudet segmenttikohtaisesti

Pentium: Osoitemuunnos

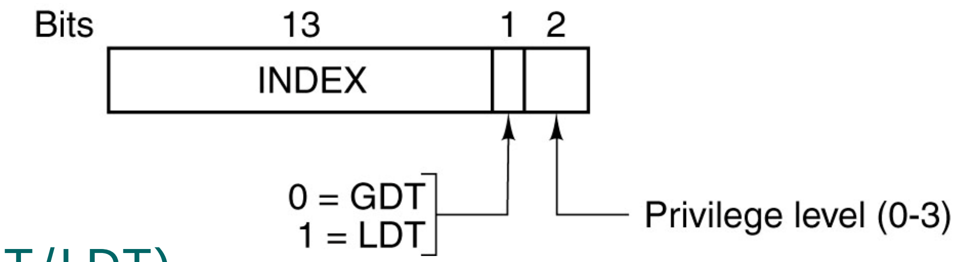


(Sta06 Fig 8.21)

- Jos Paging=Enabled, käytä sivutauluja, muuten lineaarinen osoite = fyysinen osoite (KJ, esim. laiteajurit?)
- Kontrollirekisterit ks. s. 444 [Sta06]

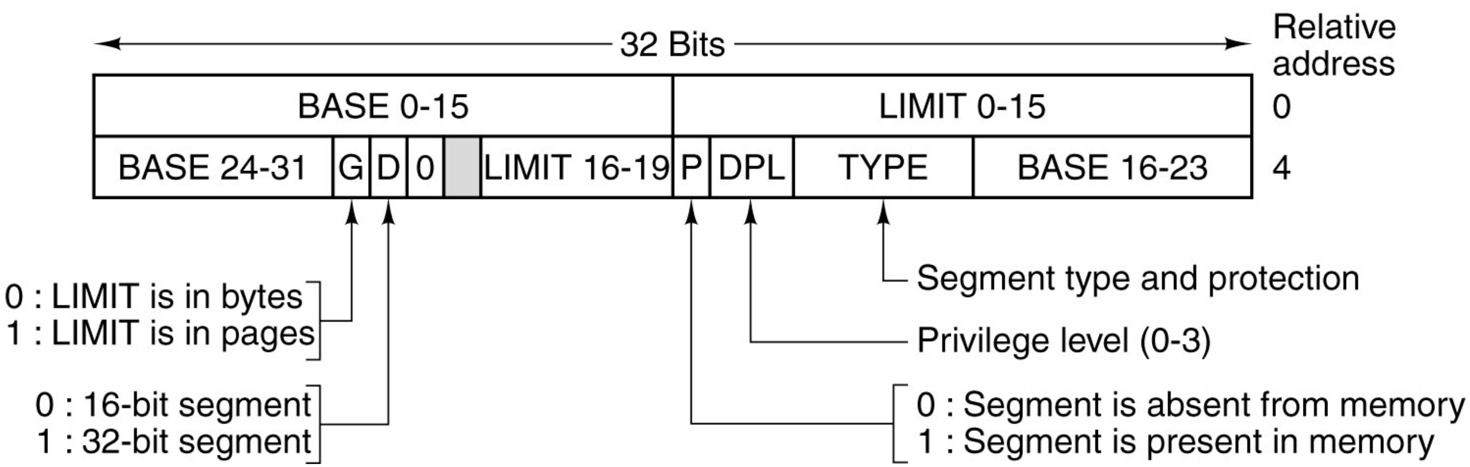
Pentium: Osoitemuunnos

- **Segment selector**
  - Global / Local
  - Segmentin numero
    - Global/Local Descriptor Table (GDT/LDT)



- **Segment descriptor**

(Tan06 Fig 6-12, 6-13)



Pentium: Segmenttikuvaaja

Segment Descriptor (Segment Table Entry)

Base

Defines the starting address of the segment within the 4-GByte linear address space.

D/B bit

In a code segment, this is the D bit and indicates whether operands and addressing modes are 16 or 32 bits.

Descriptor Privilege Level (DPL)

Specifies the privilege level of the segment referred to by this segment descriptor.

Granularity bit (G)

Indicates whether the Limit field is to be interpreted in units by one byte or 4 KBytes.

Limit

Defines the size of the segment. The processor interprets the limit field in one of two ways, depending on the granularity bit: in units of one byte, up to a segment size limit of 1 MByte, or in units of 4 KBytes, up to a segment size limit of 4 GBytes.

S bit

Determines whether a given segment is a system segment or a code or data segment.

Segment Present bit (P)

Used for nonpaged systems. It indicates whether the segment is present in main memory. For paged systems, this bit is always set to 1.

Type

Distinguishes between various kinds of segments and indicates the access attributes.

(Sta06 Table 8.5)

Pentium: Sivutaulu

Page Directory Entry and Page Table Entry

Accessed bit (A)

This bit is set to 1 by the processor in both levels of page tables when a read or write operation to the corresponding page occurs.

Dirty bit (D)

This bit is set to 1 by the processor when a write operation to the corresponding page occurs.

Page Frame Address

Provides the physical address of the page in memory if the present bit is set. Since page frames are aligned on 4K boundaries, the bottom 12 bits are 0, and only the top 20 bits are included in the entry. In a page directory, the address is that of a page table.

Page Cache Disable bit (PCD)

Indicates whether data from page may be cached.

Page Size bit (PS)

Indicates whether page size is 4 KByte or 4 MByte.

Page Write Through bit (PWT)

Indicates whether write-through or write-back caching policy will be used for data in the corresponding page.

Present bit (P)

Indicates whether the page table or page is in main memory.

Read/Write bit (RW)

For user-level pages, indicates whether the page is read-only access or read/write access for user-level programs.

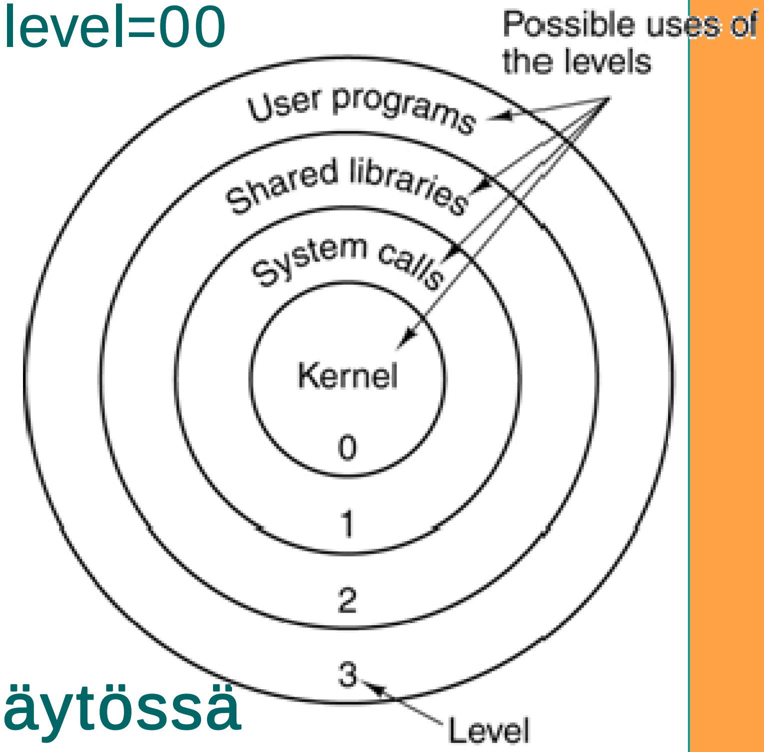
User/Supervisor bit (US)

Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).

(Sta06 Table 8.5)

Pentium: Käyttöoikeudet

- *Privilege level* CPU:n tilarekisterissä PSW:ssä
  - 00=korkein, 11 = matalin
  - Korkeampi voi viitata alemmille
  - Etuoikeutetut käskyt vain, kun level=00
- Käyttöoikeus rajattavissa
  - Segmentin valinta
    - RPL, requested privilege level
  - Segmenttikuvaaja
    - DPL, descriptor privilege level
    - Type: koodi/data? ž R/W
  - Sivutaulu
    - R/W-bitti
- Linux ja Windows: vain kaksi käytössä



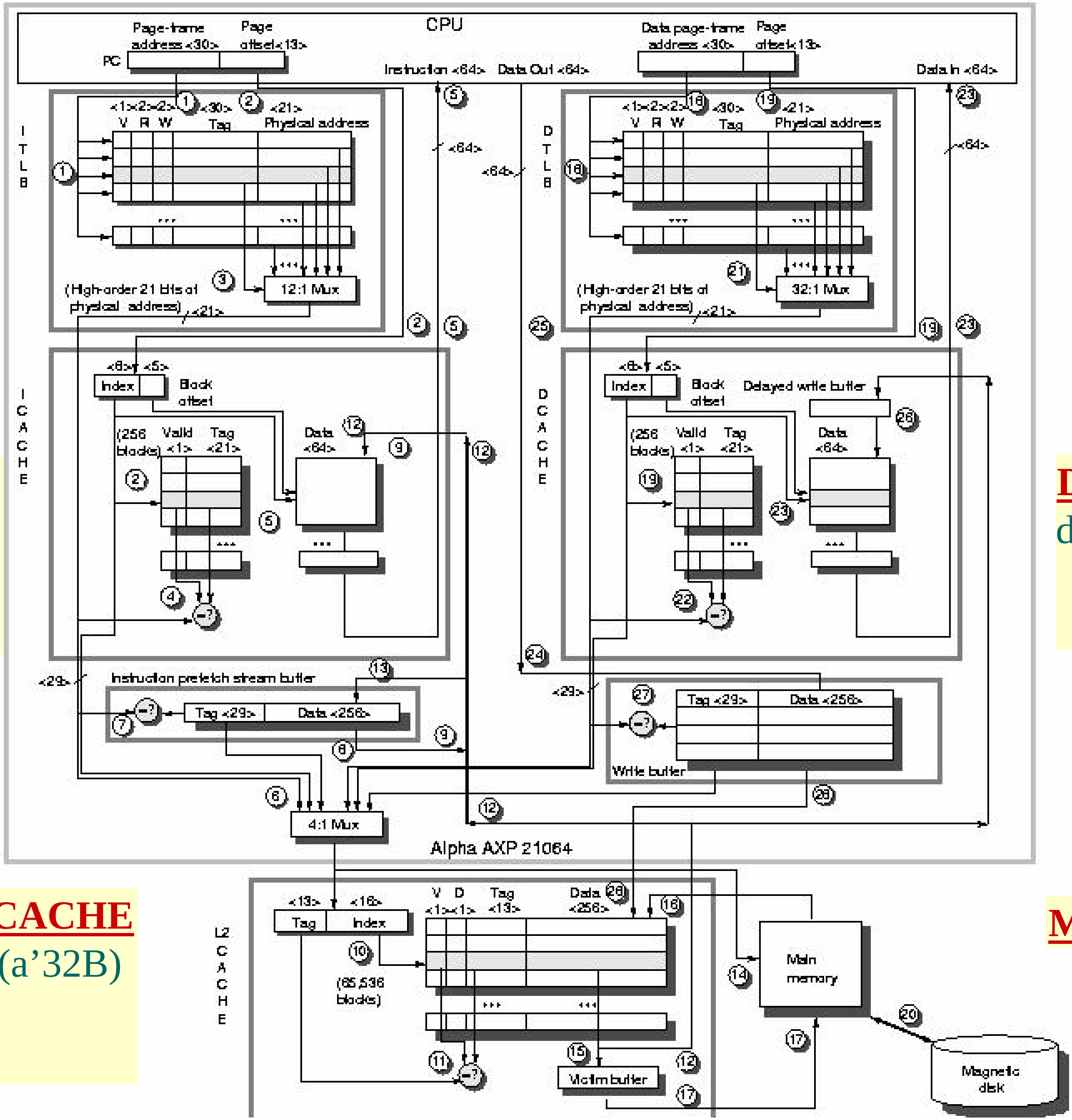
(Tan06 Fig 6-16)

Hennessy-Patterson:Computer Architecture, Fig 5.47 Alpha AXP

Instr. TLB
fully assoc,
12 entries

Instr. CACHE
direct mapped,
8 KB,
256 lines (a'32B)

Unified Level 2 CACHE
2 MB, 64K lines (a'32B)
direct mapped,
write-back



Data TLB
fully assoc,
32 entries

Data CACHE
direct mapped,
8 KB,
256 lines

Main Memory

Disk

Kertauskysymyksiä

- Mitä laitteistotason tukea tarvitaan VM:n toteuttamiseksi?
- Miten sivutus ja segmentointi eroavat toisistaan?
- Miksi ne joskus yhdistetään?
- Miten TLB ja välimuisti suhtautuvat toisiinsa?