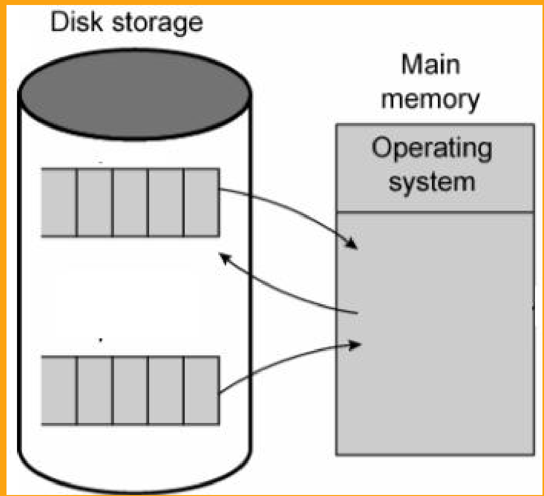
Lecture 4

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Memory Management (*Muistinhallinta*)

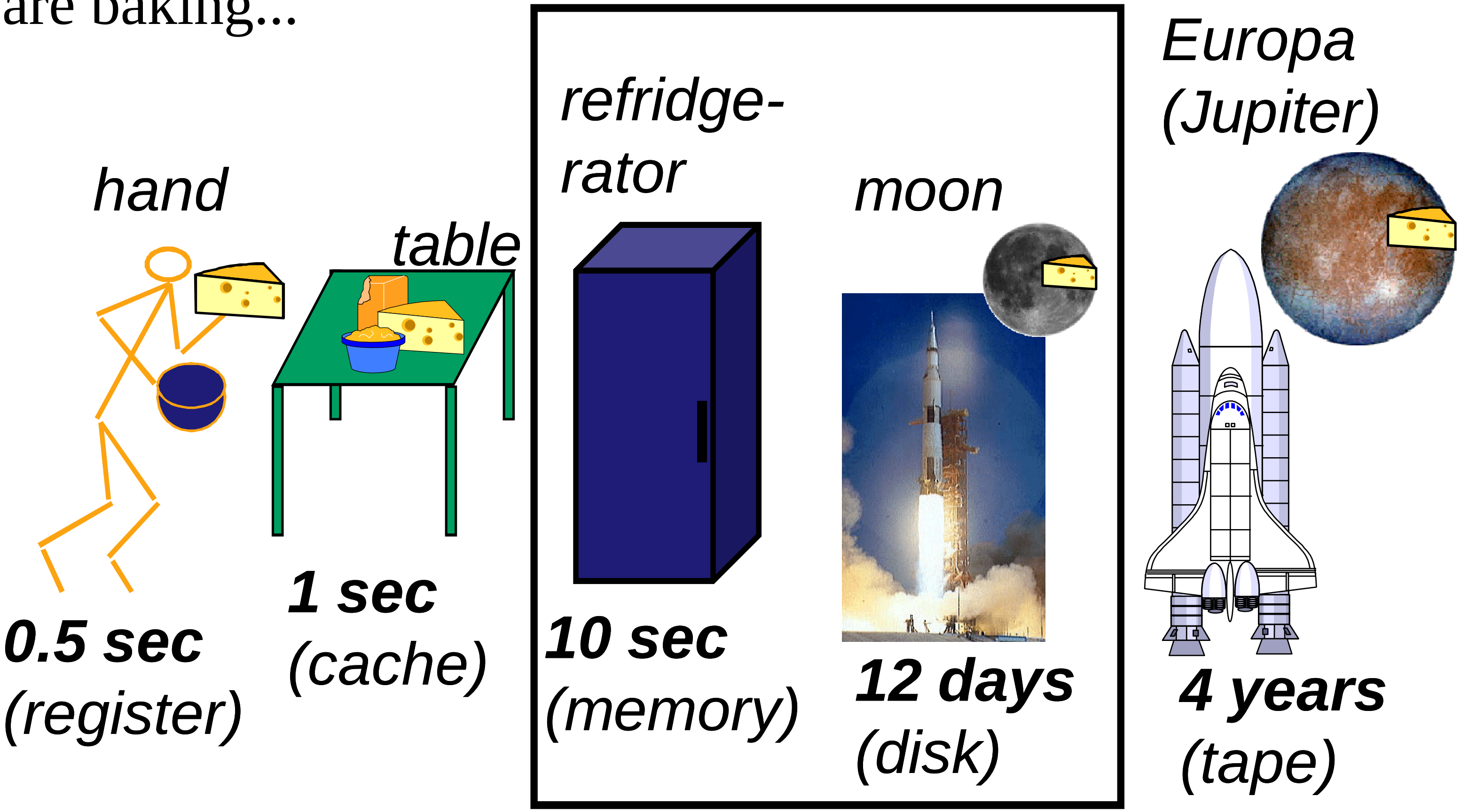
Stallings: Ch 8.3-8.6

Memory management
Swapping vs. virtual memory
Hardware and software support
Example: 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 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

Other Problems Often Solved with VM

- If you 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 available?

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 (*kiinteät partitiot, kiinteä ositus*)
  - 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
  - Can not combine
- Variable size
  - Size determined at process creation time

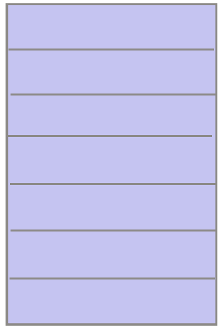


Fig. 8.13 (a) [Sta06]

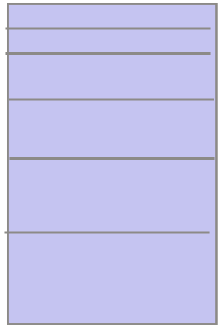


Fig. 8.13 (b) [Sta06]

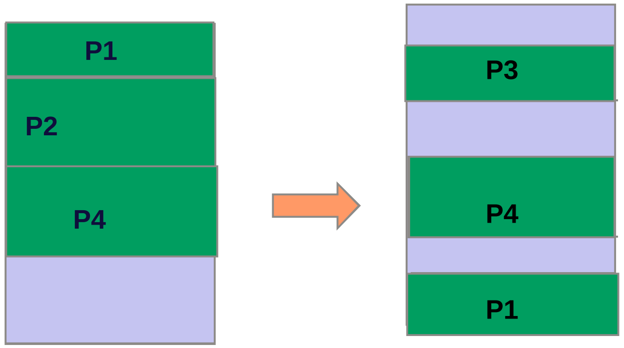


Fig. 8.14 [Sta06]

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?
  - If necessary 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

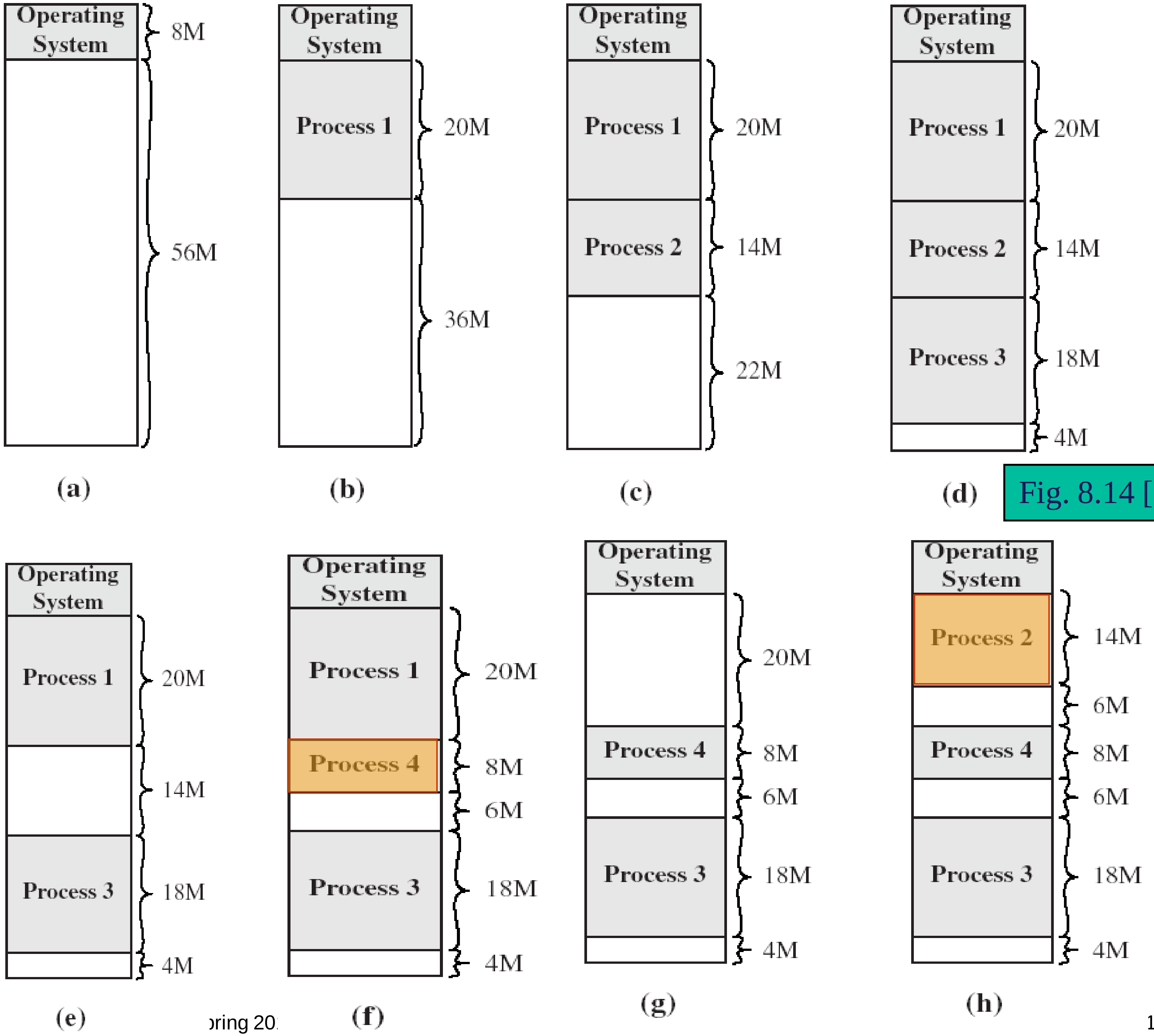


Fig. 8.14 [Sta06]

Fragmentation (*pirstoutuminen*)

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

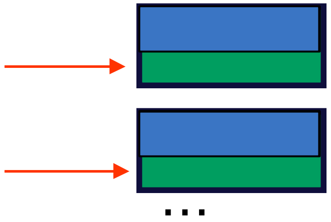


Fig. 8.13 (a) [Sta06]

- 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

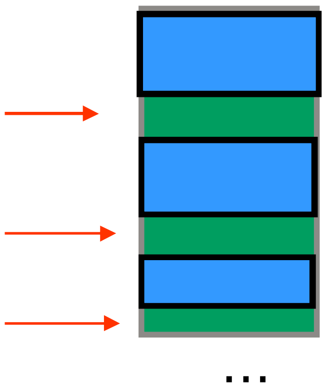


Fig. 8.13 (b) [Sta06]

Fig. 8.14 [Sta06]

Address Mapping (*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

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, address translation

- 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 (*heittovaihto*)
  - Virtual memory

Swapping (*heittovaihto*)

More on OS course

- Process has continuous memory area
  - Process fully in memory or on disk
  - Process control block, PCB (*prosessinkuvaaja*) always in memory
- Address translation at execution time (*ajonaikainen*)
  - Logical address → physical memory address
- Memory management unit , MMU, - **hardware** support
  - Base and limit registers (*Kanta- ja rajarekisteri*)
  - “Bounds exceeded”-interrupt
- Operating System (OS) (*käyttöjärjestelmä)*
  - Bookkeeping about unallocated (free) memory areas
  - Process swapping between memory and disk
  - Process switch: set new values to base and limit registers
  - Illegal (unauthorized) memory access: kill the process

Virtual Memory Implementation (*Virtuaalimuistitoteutus*)

- Methods
  - Base and limit registers (*kanta- ja rajarekisterit*)
  - Segmentation (*segmentointi*)
  - Paging (*sivutus*)
  - 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: $0 \le x < LIMIT$
    - Physical address: BASE +x

From Comp. Org I

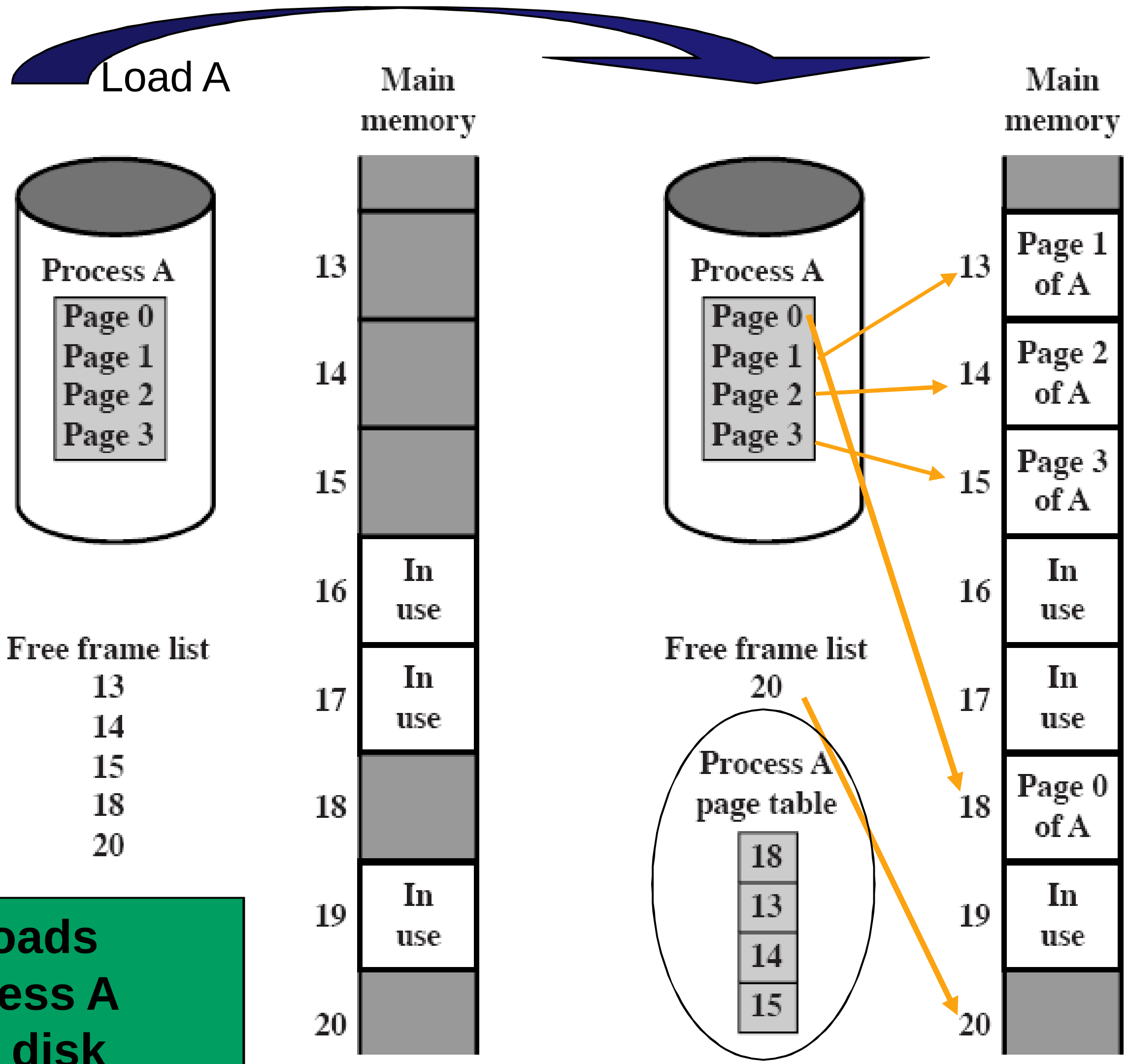
Virtual memory

OS course content

- Only needed chunks in the memory, no need to be contiguous
  - Demand paging (*Tarvenouto*)
- Chunk size?
  - Fixed size = Paging
  - Variable size = Segmentation
  - Combined = Paged segments
- OS bookkeeping (*KJ:n kirjanpito*)
  - Page frame table (*sivutilataulu*)
    - Which page frames are free, which are occupied
  - Each process has its own page table (*sivutaulu*)
    - Page in memory or on disk? Presence-bit
    - In memory, which page frame contains this page?
    - Other control? Bits: Modified, Referenced

Paging "most common"
⇨ here only paging

Virtual Memory: Paging (*sivutus*)



Program: pages

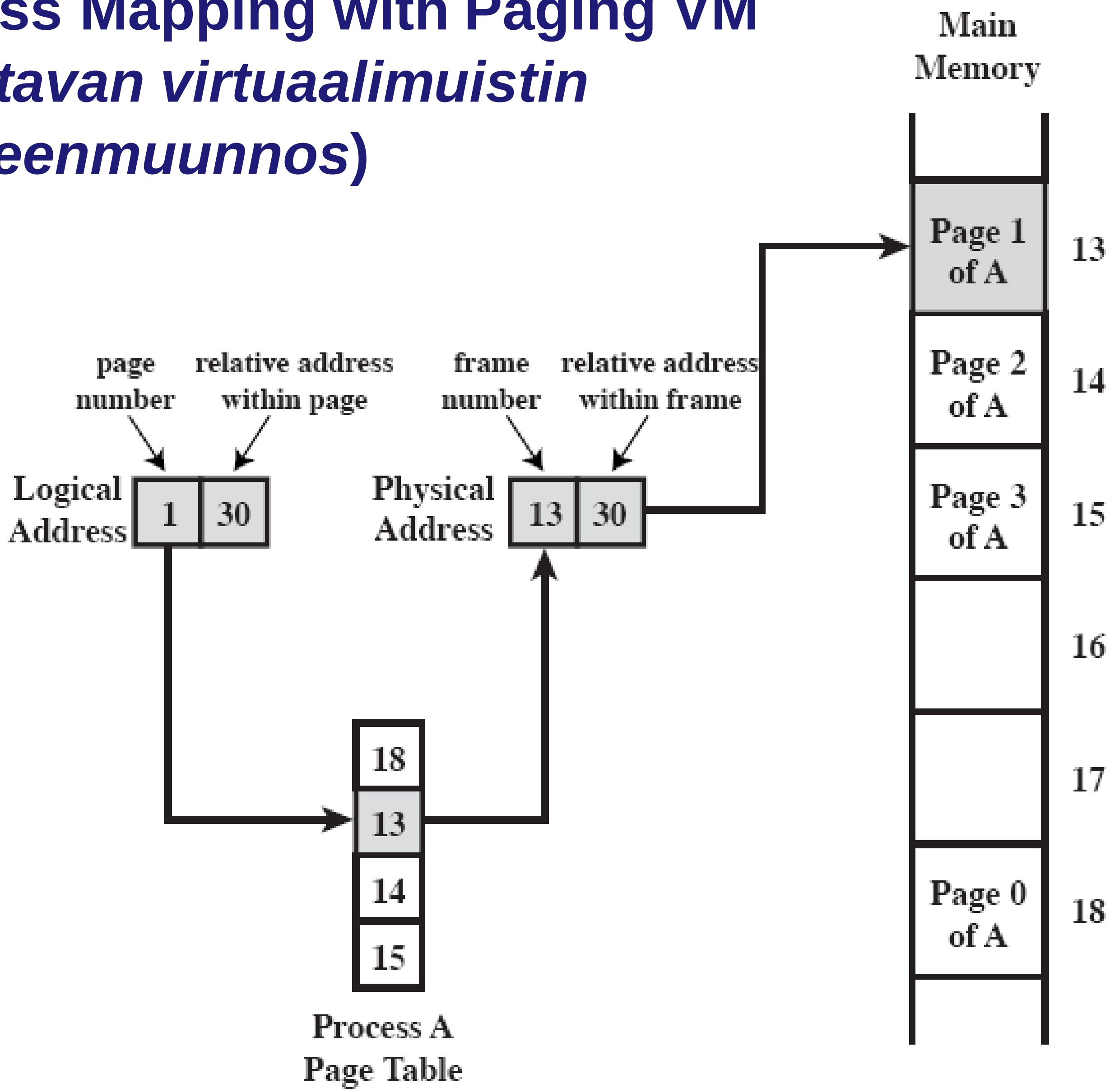
Memory: frames

OS loads process A from disk

Process A in main memory

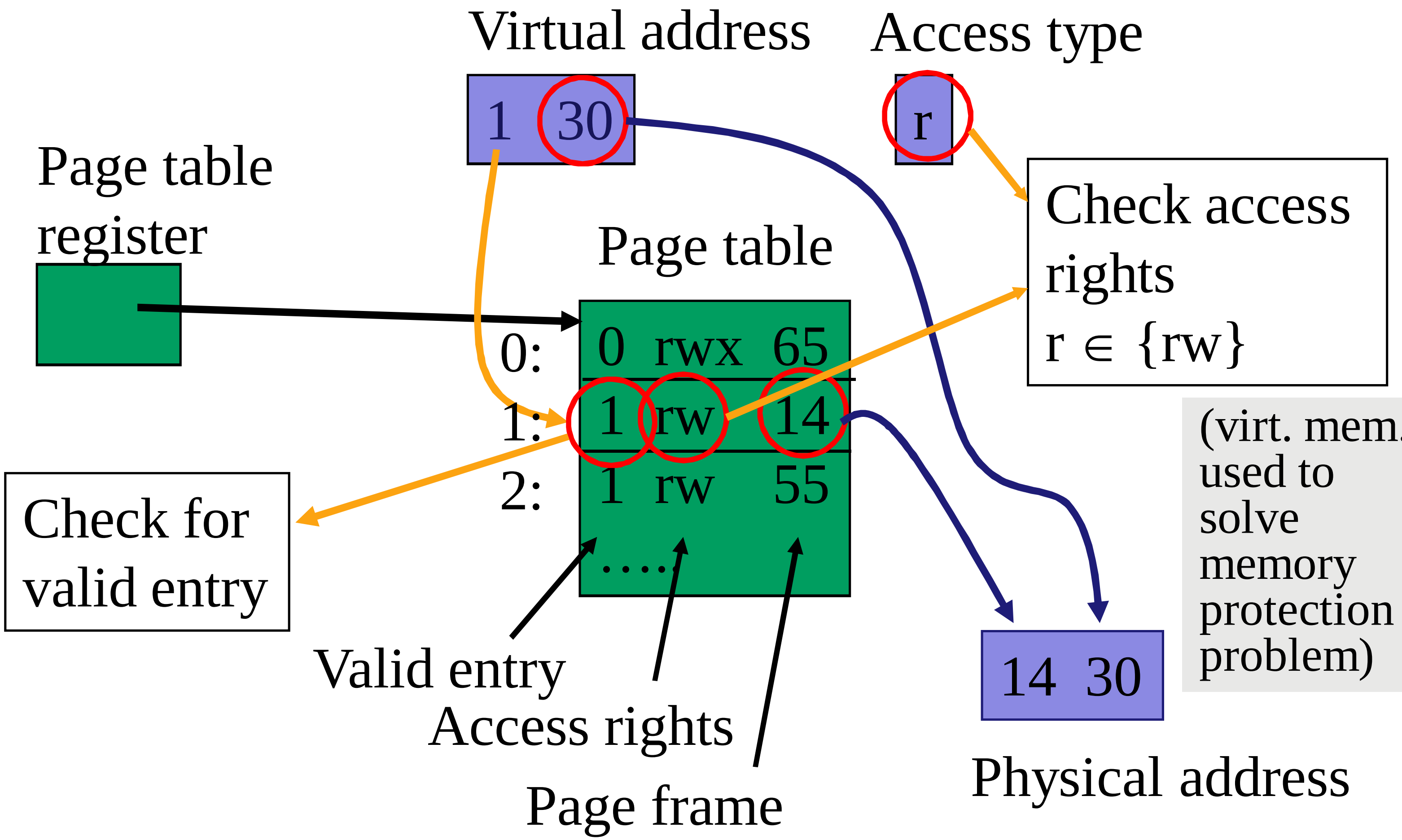
(Sta06 Fig 8.15)

Address Mapping with Paging VM (*Sivuttavan virtuaalimuistin osoitteenmuunnos*)



(Sta06 Fig 8.16)

Paged Address Translation



(virt. mem. used to solve memory protection problem)

Page Fault

Page fault interrupt

Stop execution

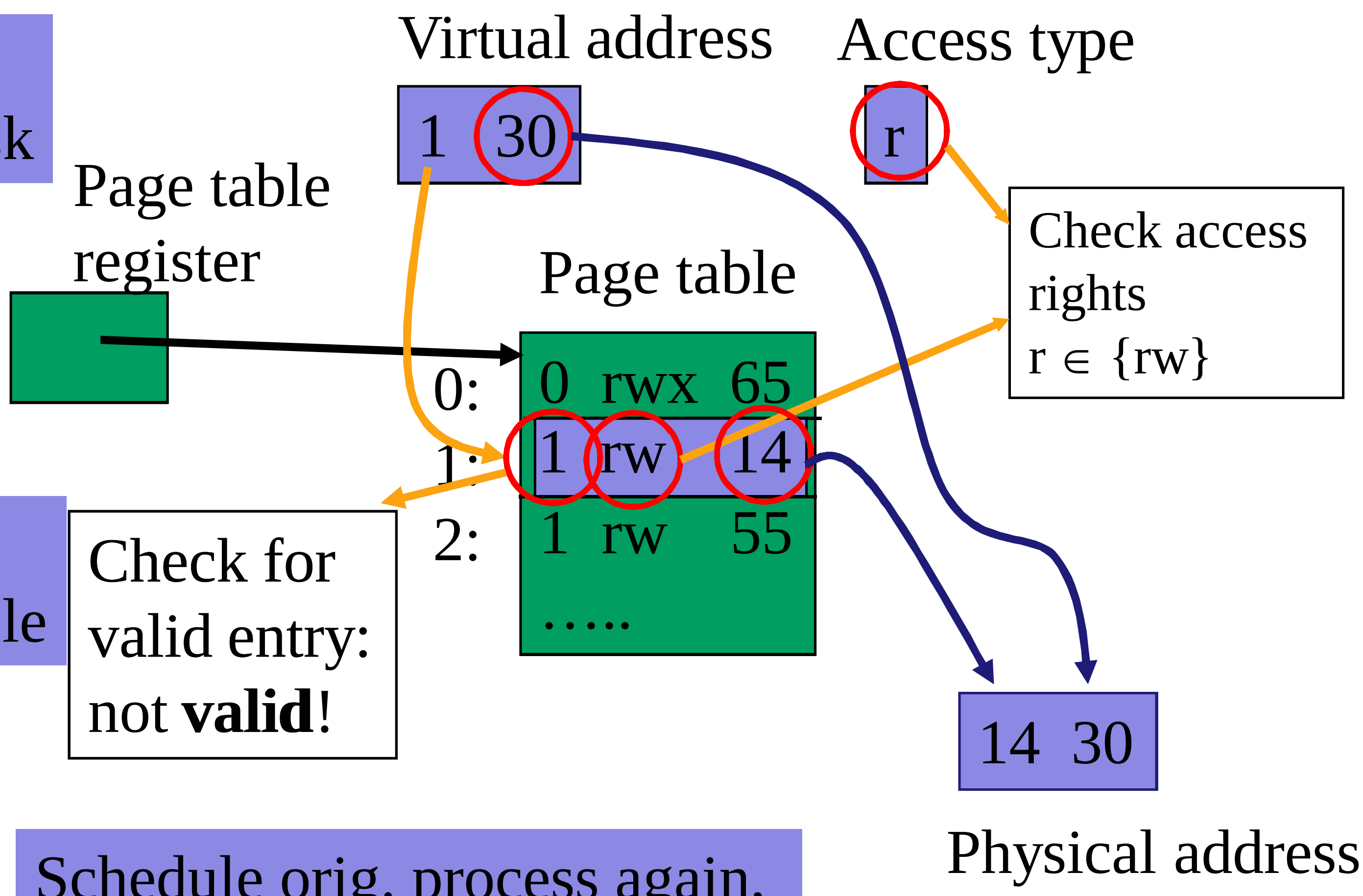
Initiate reading page 1 from disk

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

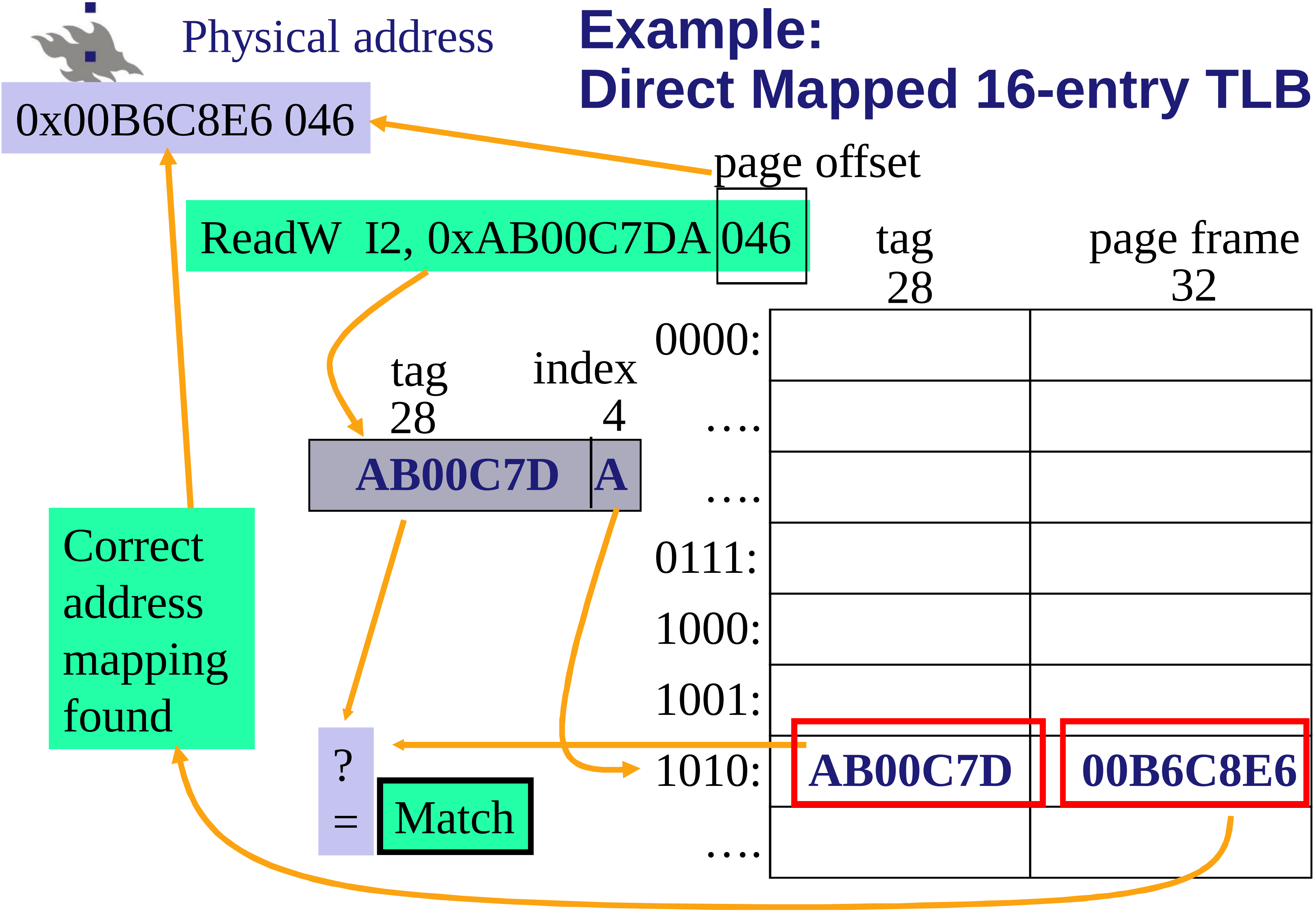
Virtual memory: Translation Lookaside Buffer (TLB) (*osoitteenmuunnospuskuri*)

- Address translation for each memory reference, at least once for each instruction
- Page table elements in memory

= extra (even more) memory access?

  - Too slow!
- Solution ←
  - Principle of Locality! Page table element needed soon again
  - Store recently used page table elements (of this process) on CPU's memory management unit
- TLB, translation lookaside Buffer
  - Just like cache
  - Fast set of registers (Pentium: 32 registers)
  - Associative search
  - Hit ratio (*Osumatodennäköisyys*) 99.9% ? (Almost always!)

Example: Direct Mapped 16-entry TLB

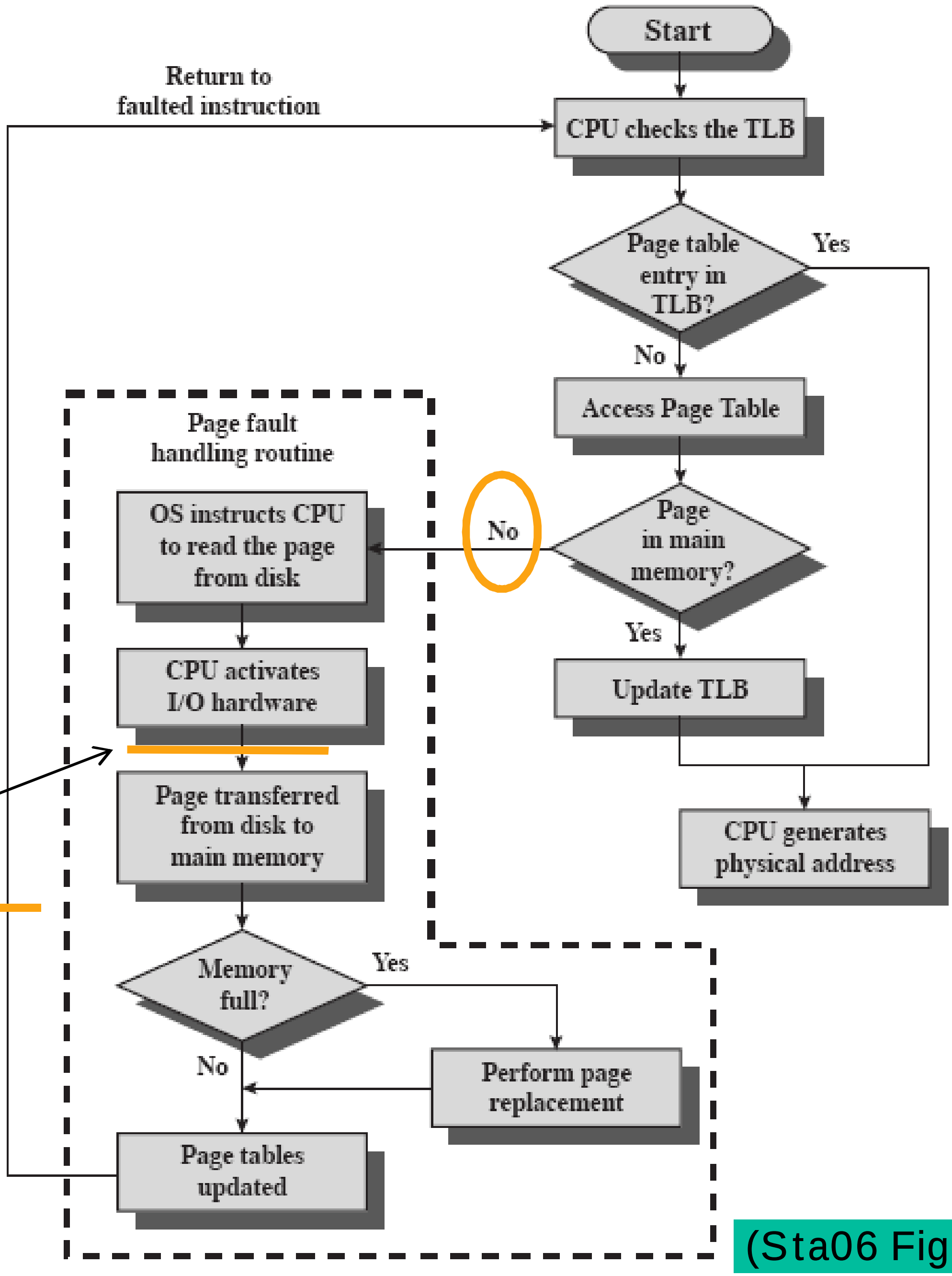


Translation Lookaside Buffer (TLB)

- "Hit" on TLB?
  - address translation is in TLB - real fast
- "Miss" on TLB?
  - must read page table entry from memory
  - takes time – not much, just a memory reference
    - Entry might be in cache!
  - 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?

Page Fault? (*Sivun-puutos-keskeytys*)

Process switch (minimum 2)

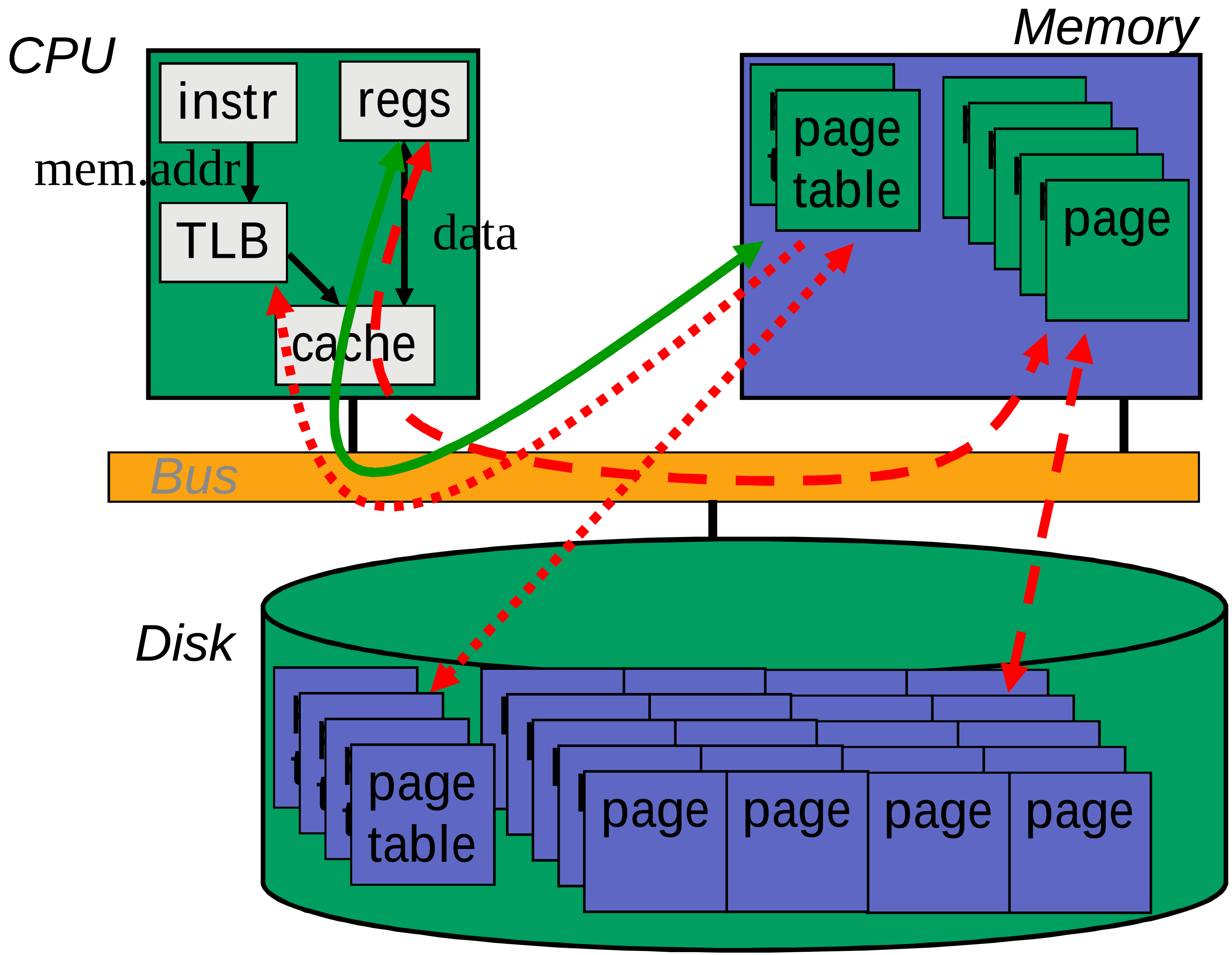


(Sta06 Fig 8.18)

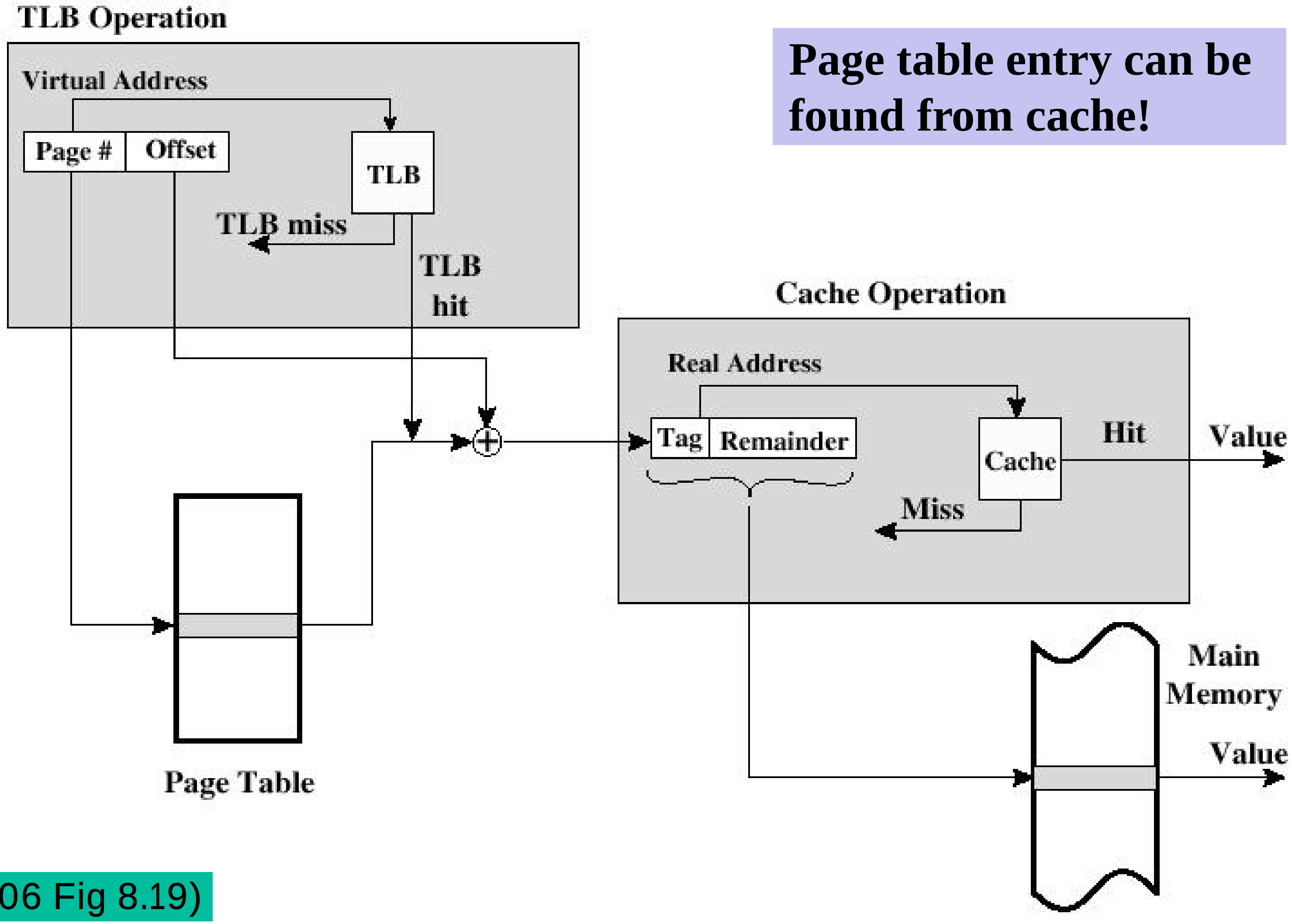
Virtual memory

- Hardware support: MMU and its special registers
  - PTR (page table register)
    - Physical start address of process page table (copied from PCB – process control block)
  - TLB (translation lookaside buffer)
    - Caches page table entries from earlier address mappings
  - “Page fault” –interrupt
  - Updating reference and modified bits
- Process switch
  - PTR ← Physical start address of process page table
  - Invalidate old TLB content (process specific)
    - Each location has valid bit
    - Changed elements back to memory (“cache block”)

Memory Organisation



TLB and cache



Page table entry can be found from cache!

(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

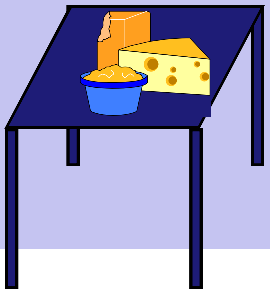
Cache Miss

- 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

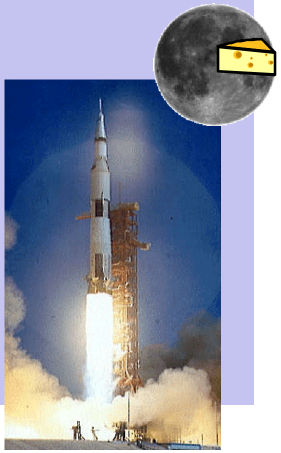
TLB Miss

- 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 processes
- SW implementation
- Data is copied from disk to memory
- Delay 1-4 (?) clock cycles



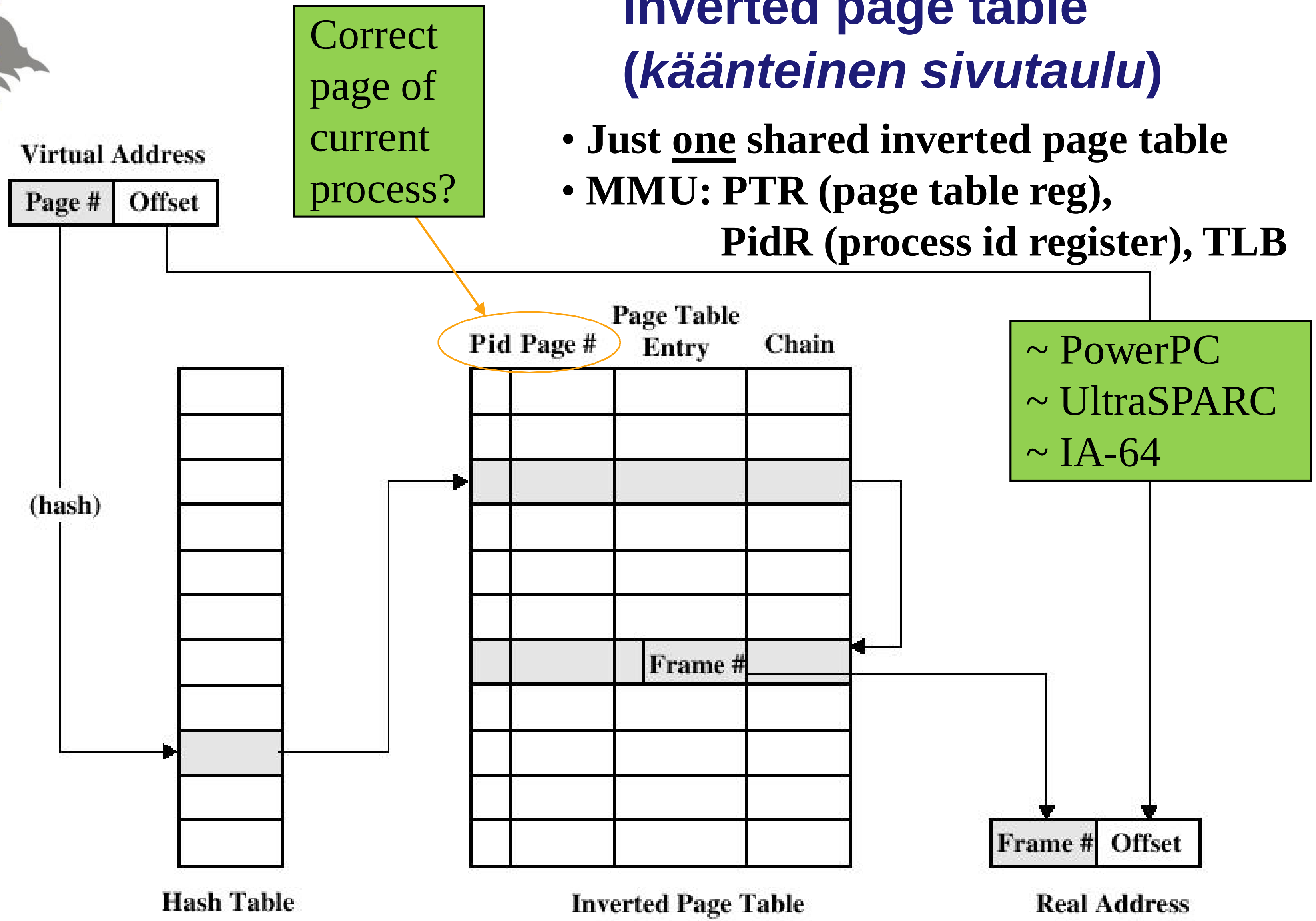
Replacement policy (*Korvauspolitiikka*)

OS course

- Which page should be replaced, when there is not enough free page frames in main memory?
- Local/ global policy
  - Select from the processes own pages
  - Select from all pages (of all processes)
- Algorithm
  - Clock, Second change, LRU, …
- MMU
  - At page access set Referenced=1 (read)
  - set Modified=1, page content changed (write)
- OS
  - Reset Referenced and Modified "periodically"
  - Replace a page where R=0, M=0
  - M=1 ⇨ write the page to disk before reusing the page frame

Inverted page table
(*käänteinen sivutaulu*)

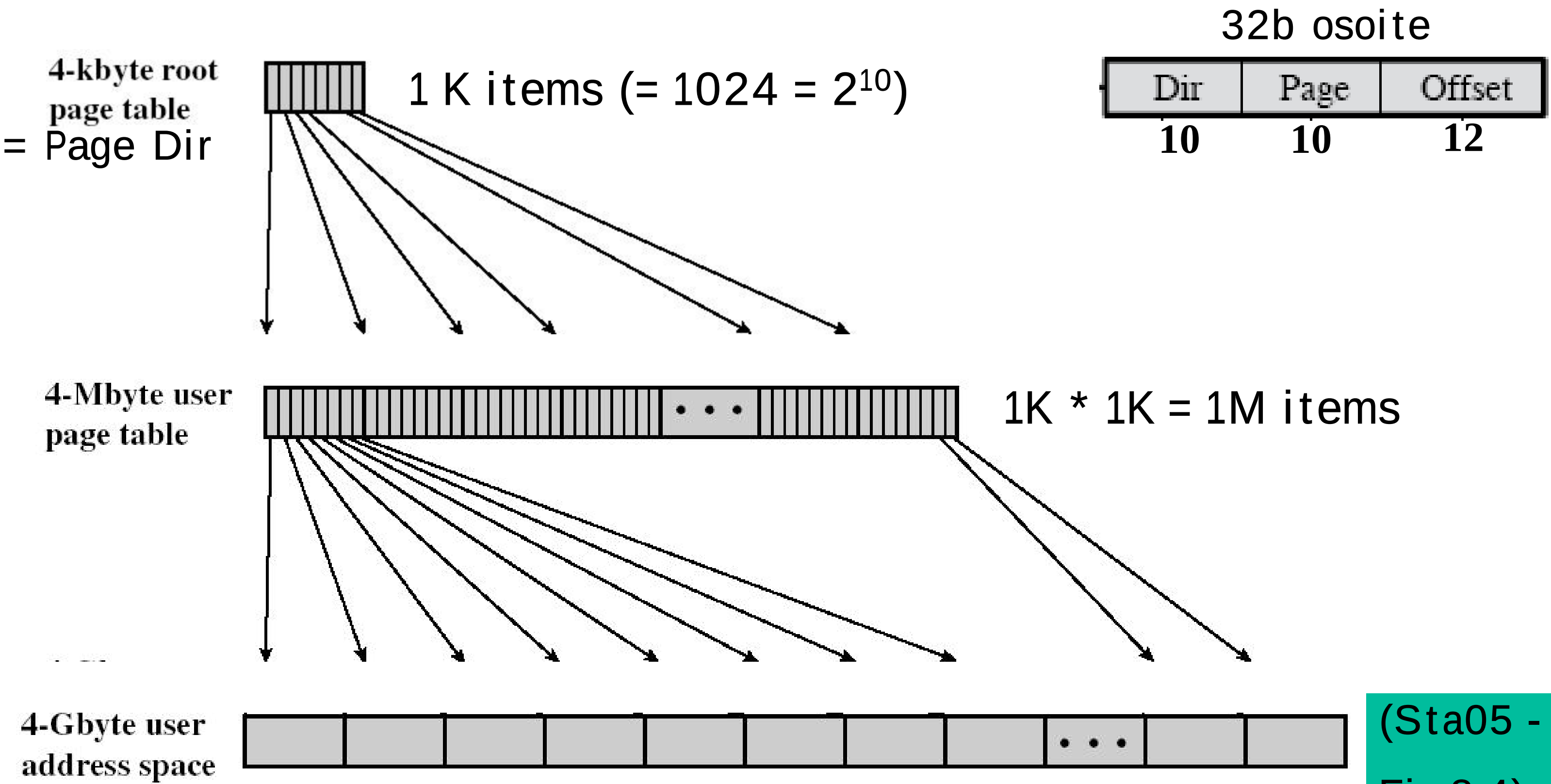
- Just <u>one</u> shared inverted page table
- MMU: PTR (page table reg),
  PidR (process id register), TLB



(Sta06 Fig 8.17)

Hierarchical page table (*monitasoinen sivutaulu*)

- Several systems allow large virtual address space
  - Page table split to pages, some of it on the disk
  - Top level of page table fits to one page, always in memory



(Sta05 - OS Fig 8.4)

Virtual Memory Policies

- 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 sivut*)

Example

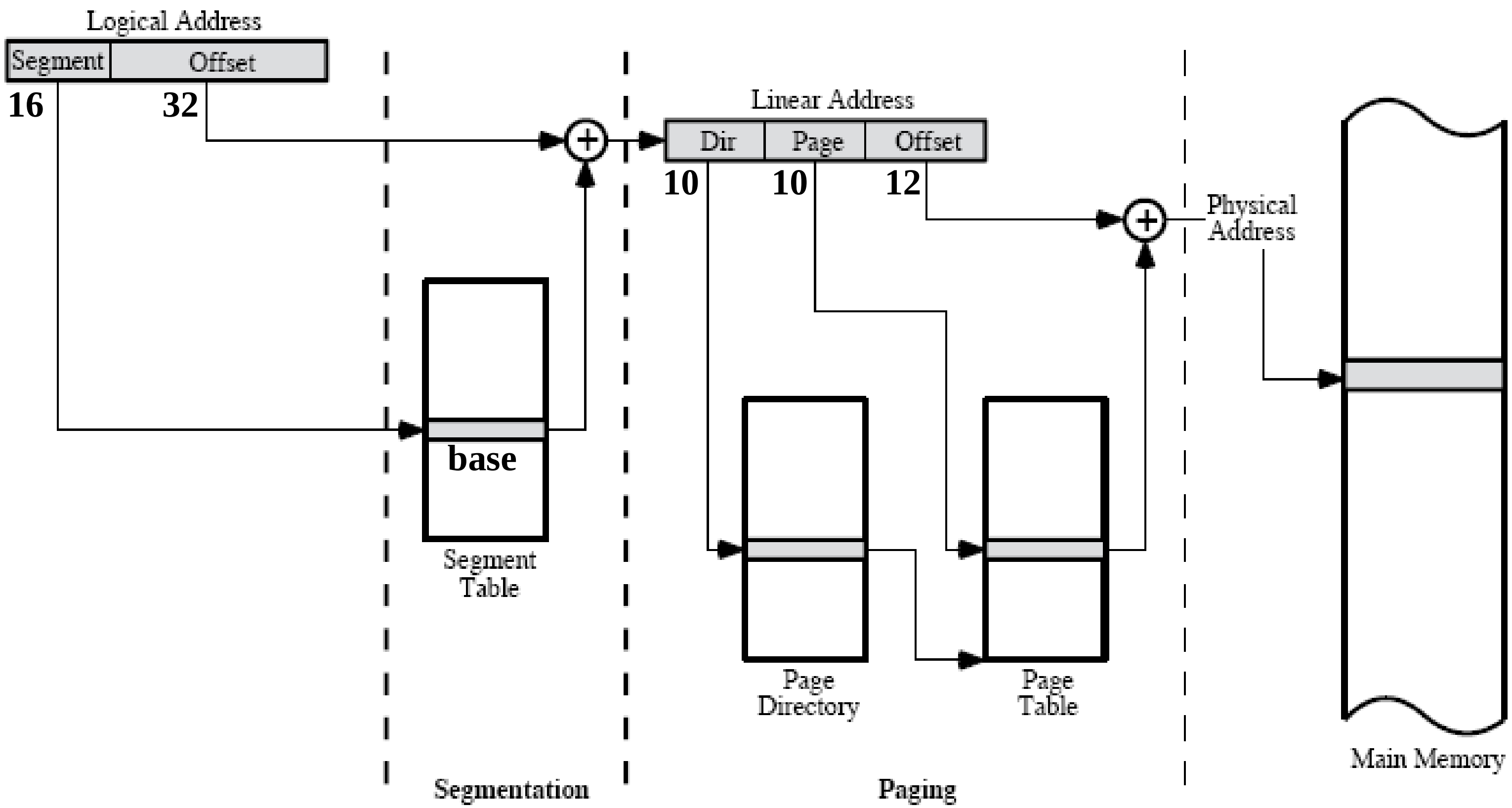
Pentium (IA-32)

See also Tan06

Pentium support for memory management

- Unsegmented unpaged, max 2^{32} = 4 GB
  - Virtual address = physical address
  - Efficient ⇨ feasible in real-time systems
- Unsegmented paged (*Sivuttava*), max 4 GB
  - Linear address space (*lineaarinen osoiteavaruus*)
  - Page and frame size: 4KB or 4MB
  - Protection frame based
- Segmented unpaged (*Segmentoiva*), max 2^{48} = 64 TB
  - Several segments ⇨ several linear memory spaces
  - Protection segment based
- Segmented paged (*Sivuttava segmentointi*), max 64 TB
  - Memory management using pages and page frames
  - Protection segment based

Pentium: Address translation

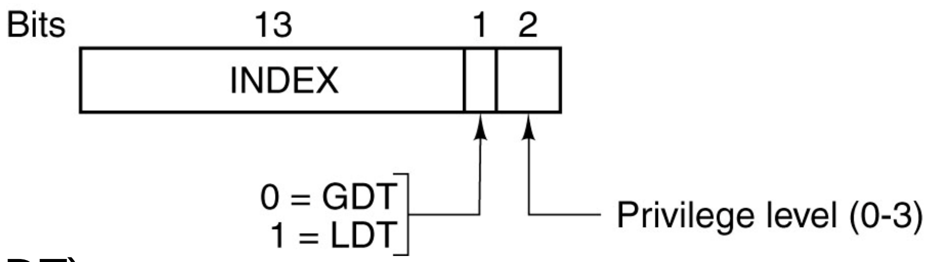


(Sta06 Fig 8.21)

- If *Paging=Enabled*, use page tables
  else linear address = physical address (OS, f.ex. Devide drivers?)
- Control registers (see further in the course book)

Pentium: Address translation

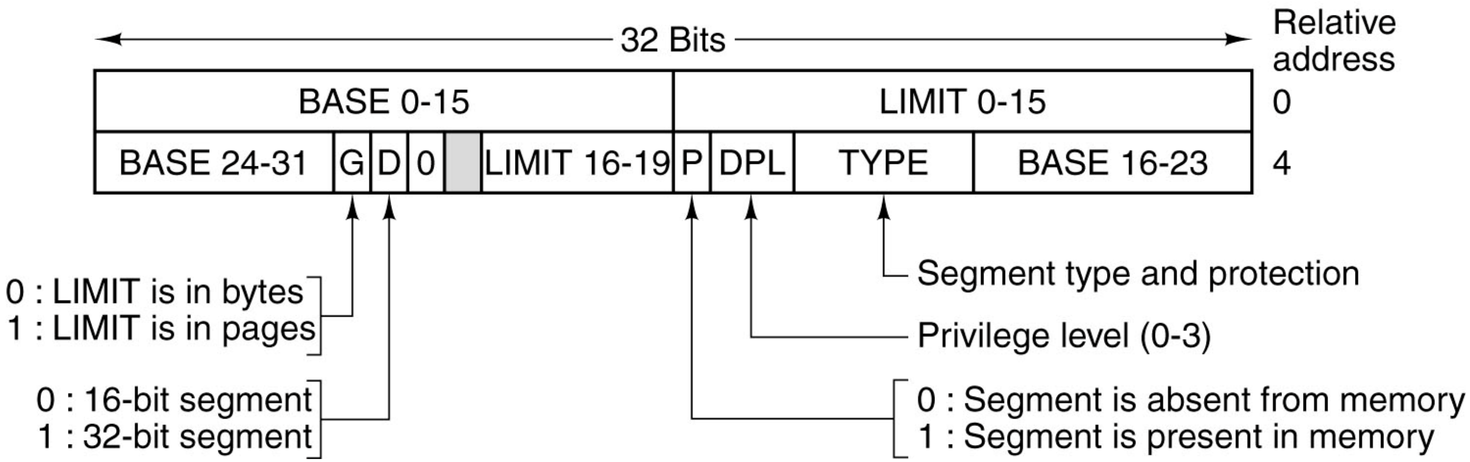
- Segment selector
  - Global / Local Table
  - Segment number
    - Global/Local Descriptor Table (GDT/LDT)
  - CS/DS selectors for code/data segments



(Tan06 Fig 6-12, 6-13)

Segment descriptor

Get from GDT or LDT
and store to registers on MMU



Pentium: Segment Desriptor (*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.

(Sta09 Table 8.5)

Pentium: Page Table (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)

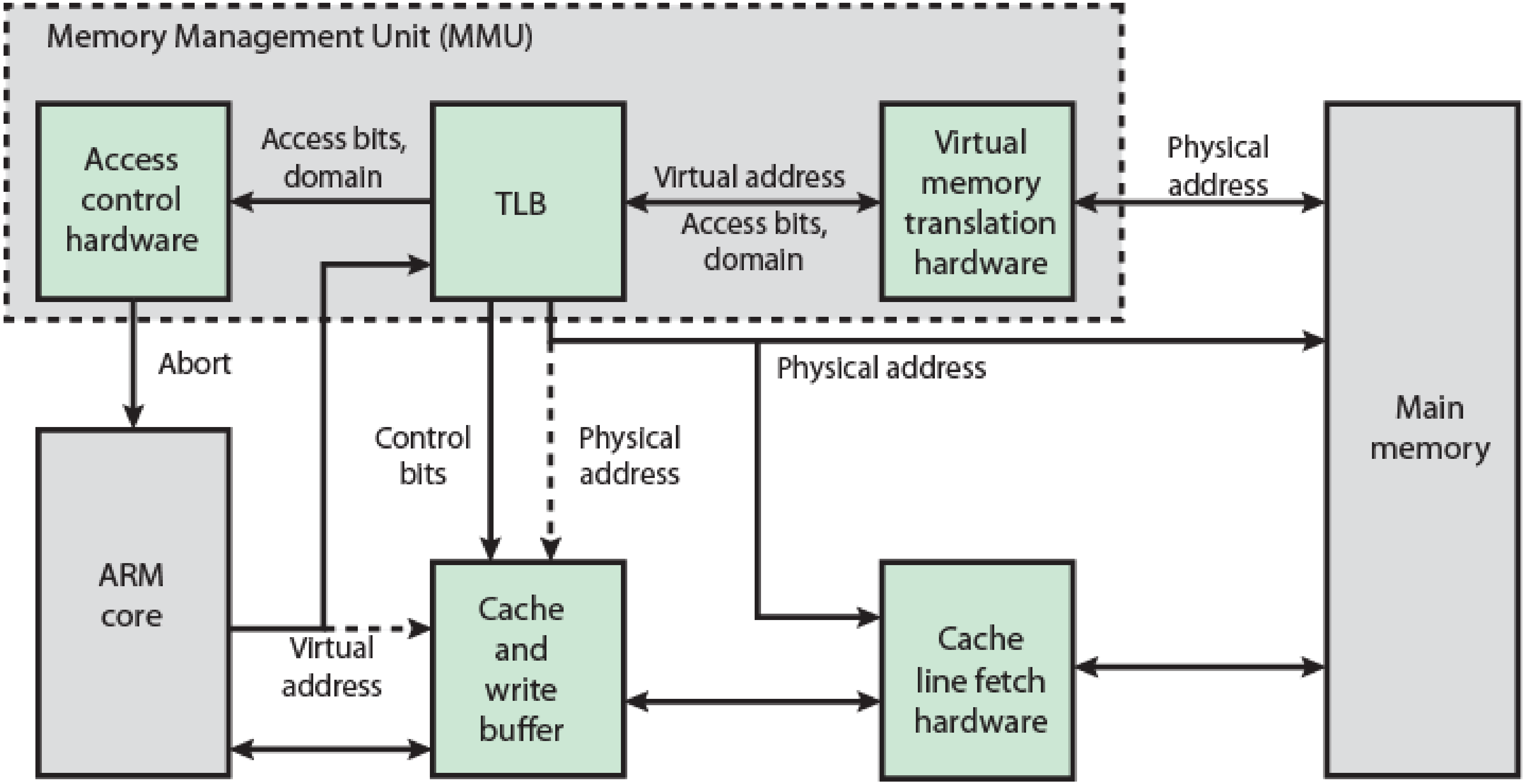
Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).

(Sta09 Table 8.5)

Example

ARM

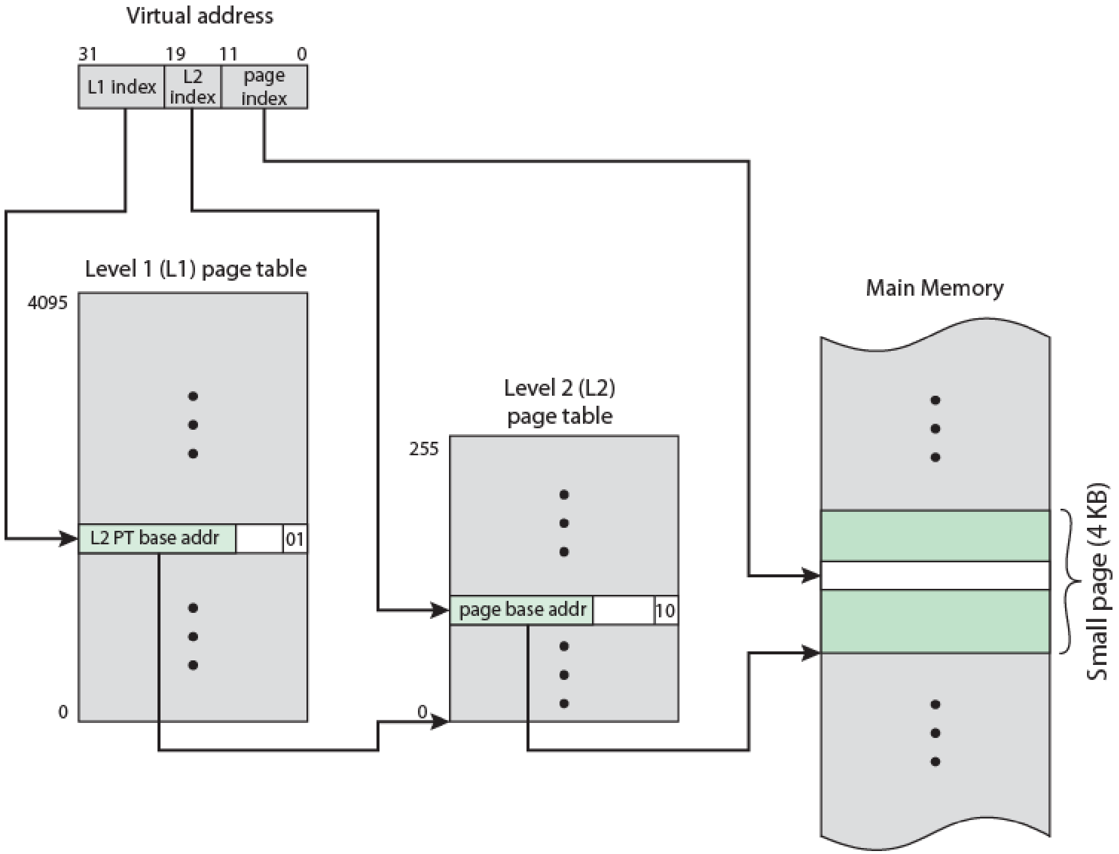
ARM Memory System Overview



(Sta09 Fig 8.22)

ARM Virtual Memory Address Translation for Small Pages - Diagram

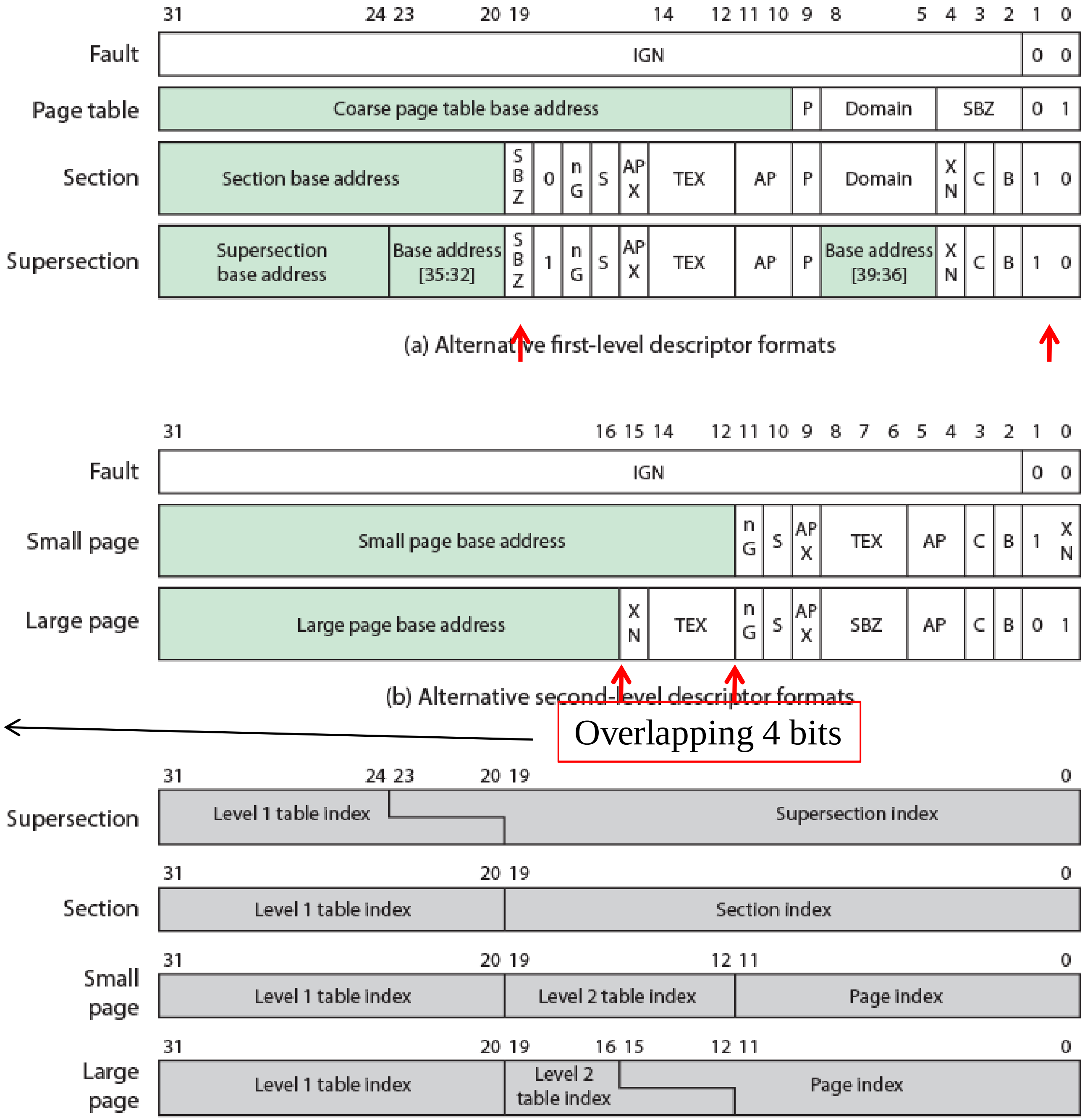
- Single L1 page table
  - 4K 32-bit entries
  - Each L1 entry points to L2 page table
- Each L2 page table
  - 256 32-bit entries
  - Each L2 entry points to 4-KB page in main memory
- 32-bit virtual address
  - 12 bit - L1
  - 8 bit - L2
  - 12 bit - offset (=page index)



(Sta09 Fig 8.23)

ARMv6 Memory Management Formats

Second level page table:
- Large page
entry replicated 16 times
-Mix of small and large pages allowed



(c) Virtual memory address formats

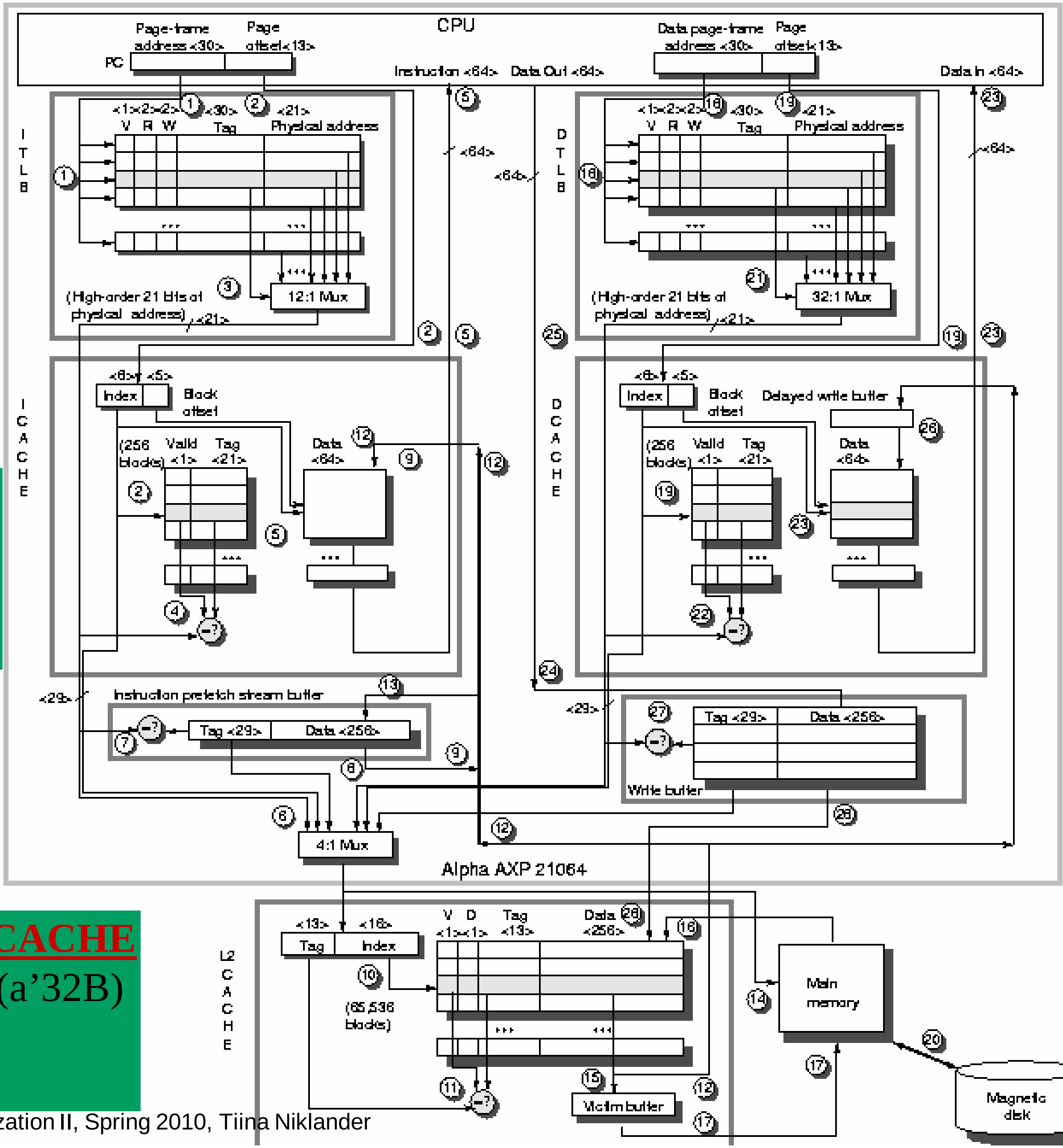
(Sta09 Fig 8.24)

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

Review Questions / Kertauskysymyksiä

- What hardware support is needed for virtual memory implementation?
- Differences of paging and segmentation?
- Why to combine paging and segmentation?
- Relationship of TLB and cache? Similarities, differences?

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