



Internal Memory Cache

Stallings: Ch 4, Ch 5

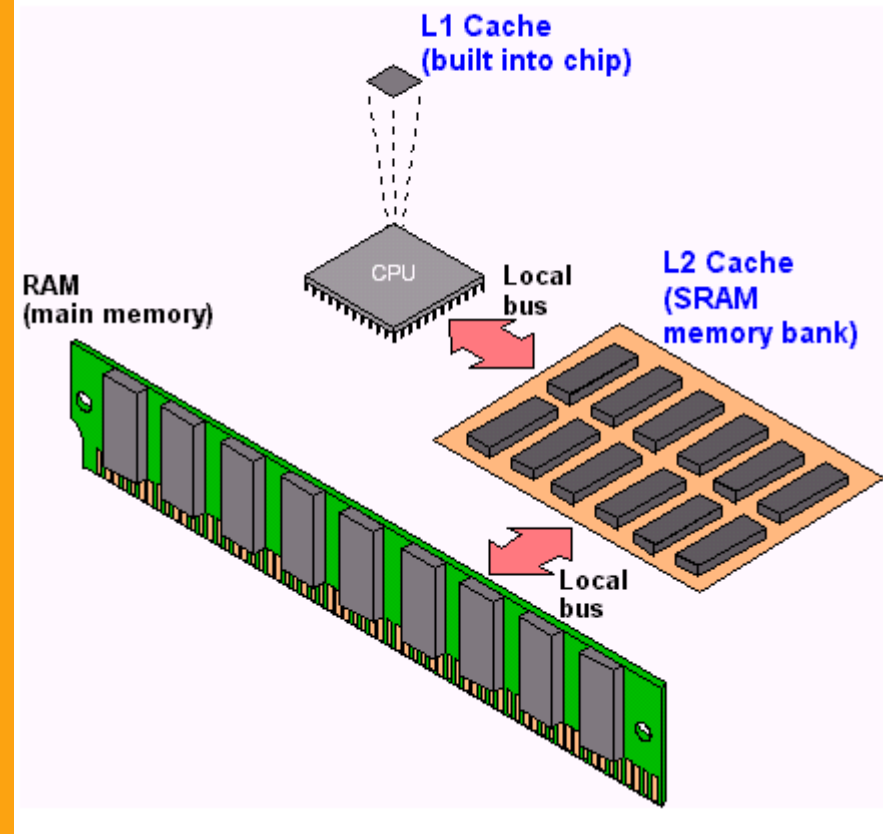
Key Characteristics

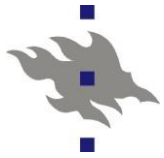
Locality

Cache

Main Memory

From Computer Desktop Encyclopedia
© 1999 The Computer Language Co. Inc.

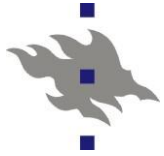




Key Characteristics of Memories / Storage

Location	Performance
Processor	Access time
Internal (main)	Cycle time
External (secondary)	Transfer rate
Capacity	Physical Type
Word size	Semiconductor
Number of words	Magnetic
Unit of Transfer	Optical
Word	Magneto-Optical
Block	Physical Characteristics
Access Method	Volatile/nonvolatile
Sequential	Erasable/nonerasable
Direct	Organization
Random	
Associative	

(Sta06 Table 4.1)



Goals

- I want my memory lightning fast
- I want my memory to be gigantic in size

- Register access viewpoint
 - data access as fast as HW register
 - **data size as large as memory**

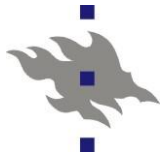
- Memory access viewpoint
 - data access as fast as memory
 - **data size as large as disk**

cache

HW solution

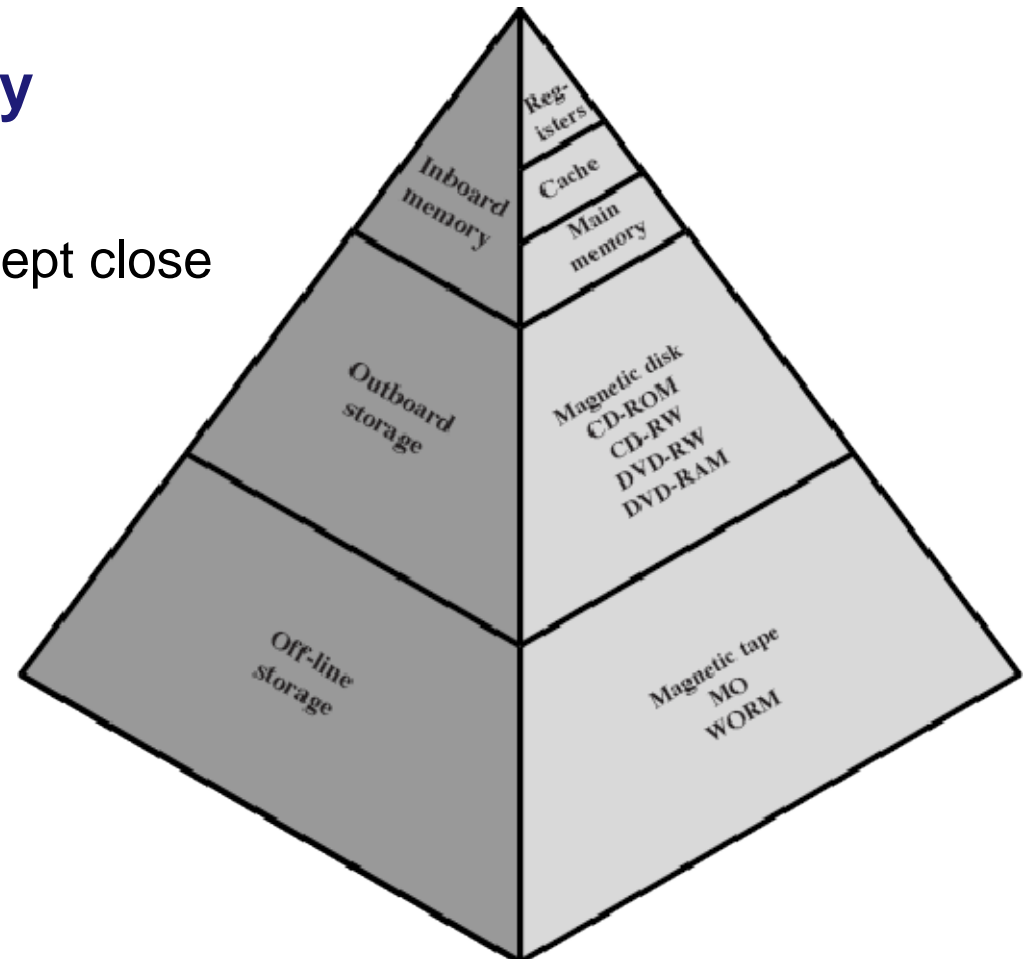
virtual
memory

HW help for
SW solution



Memory Hierarchy

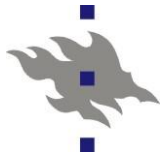
- Most often needed data kept close
- Access to small data sets can be made fast
 - simpler circuits
 - smaller gate delays
- Faster ~ more expensive
- Large can be bigger and cheaper (per B)



up: smaller, faster,
more expensive, more frequent access

down: bigger, slower,
less expensive, less frequent access

(Sta06 Fig 4.1)



Principle of locality (*paikallisuus*)

- In any given time period, memory references occur only to a small subset of the whole address space
= The reason why memory hierarchies work

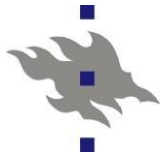
Prob (small data set) = 99%
Prob (the rest) = 1%

“Cost” (small data set) = 2 μ s
“Cost” (the rest) = 20 μ s

$$\text{Aver cost} = 99\% * 2 \mu\text{s} + 1\% * 20 \mu\text{s} = 2.2 \mu\text{s}$$

- Average cost is close to the cost of small data set
- How to determine data for that small set?
- How to keep track of it?

Sta06 Fig 4.2



Principle of locality

- In any given time period
 - memory references occur only to a small subset of the whole address space



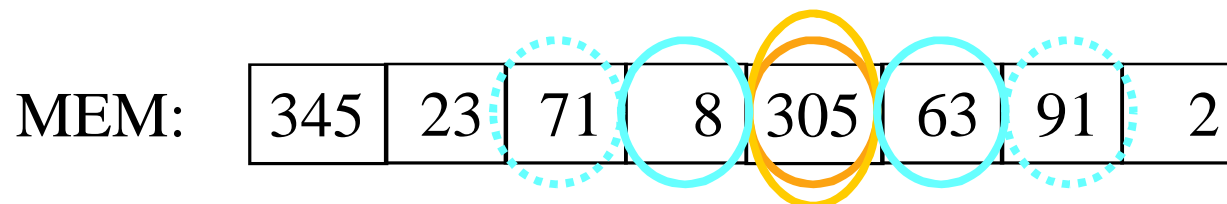
- Temporal locality (*ajallinen*)

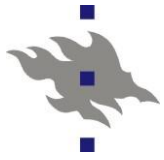
- it is likely that a data item referenced a short time ago will be referenced again soon



- Spatial locality (*alueellinen*)

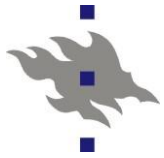
- it is likely that a data items close to the one referenced a short time ago will be referenced soon





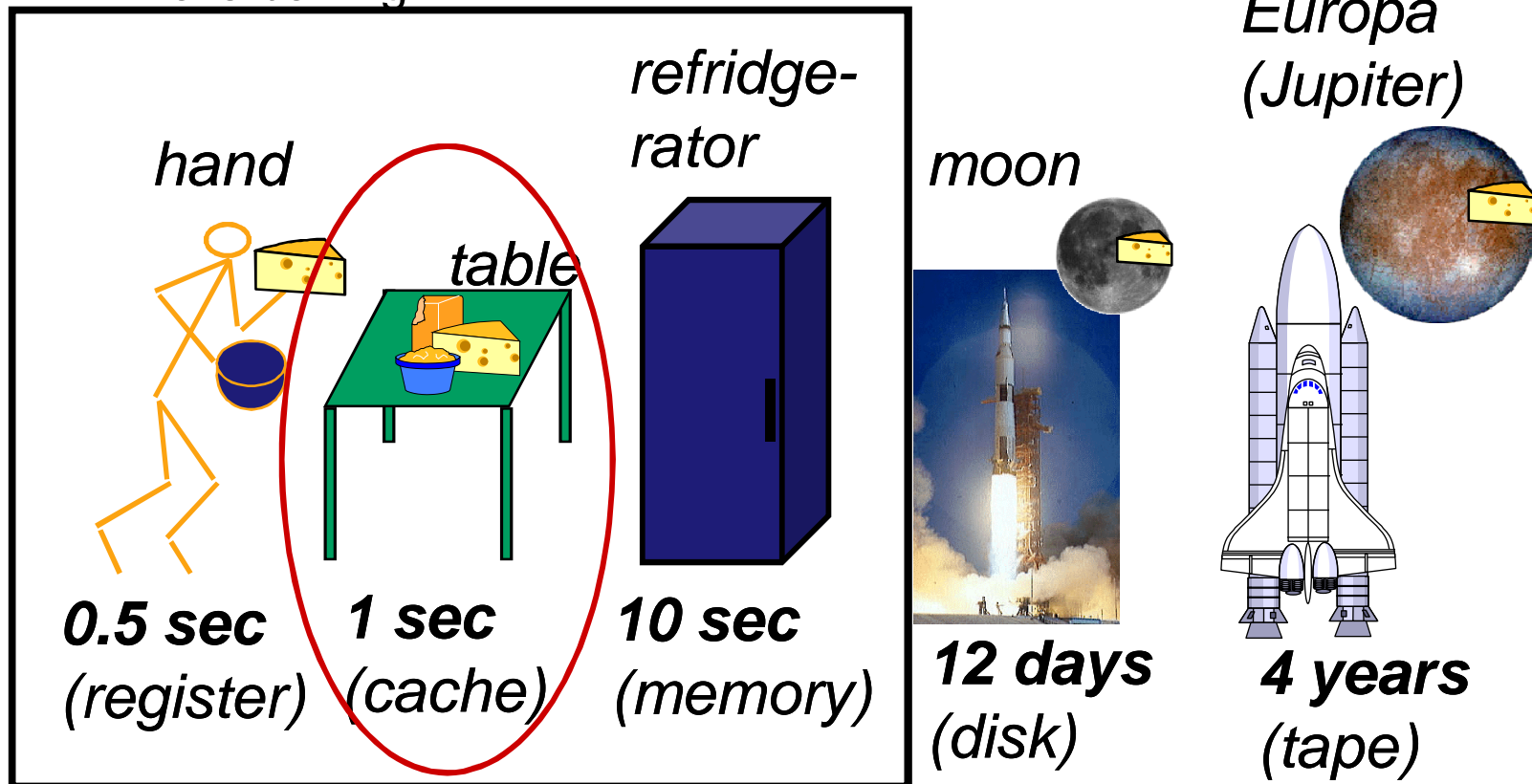
Computer Organization II

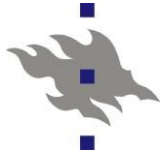
Cache



Teemu's Cheesecake

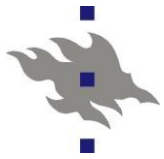
Register, on-chip cache, memory, disk, and tape speeds relative to times locating cheese for the cheese cake you are baking...



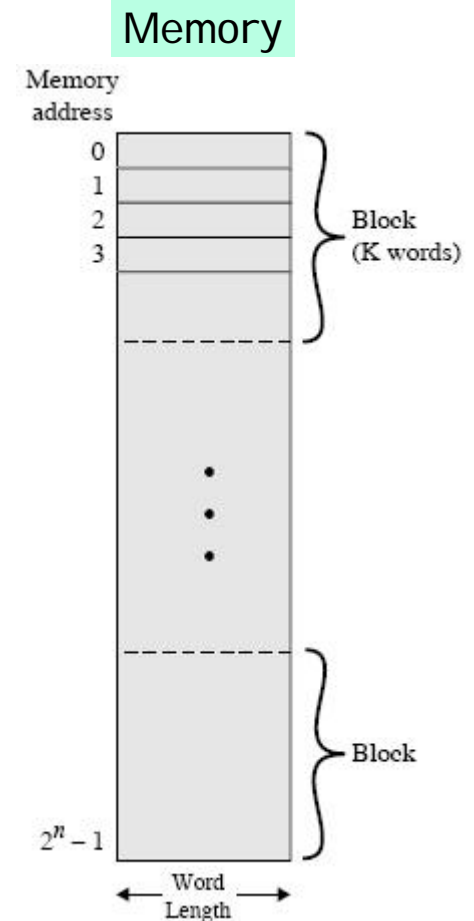
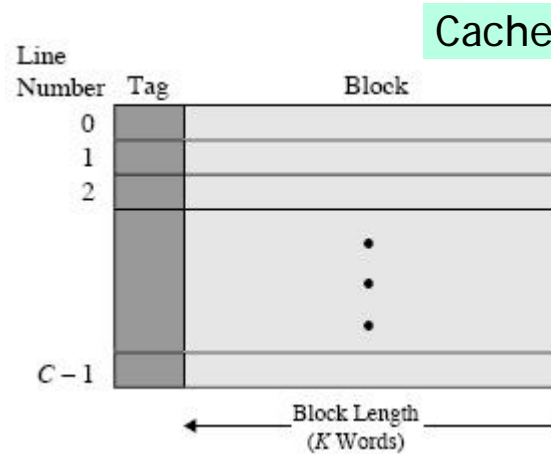
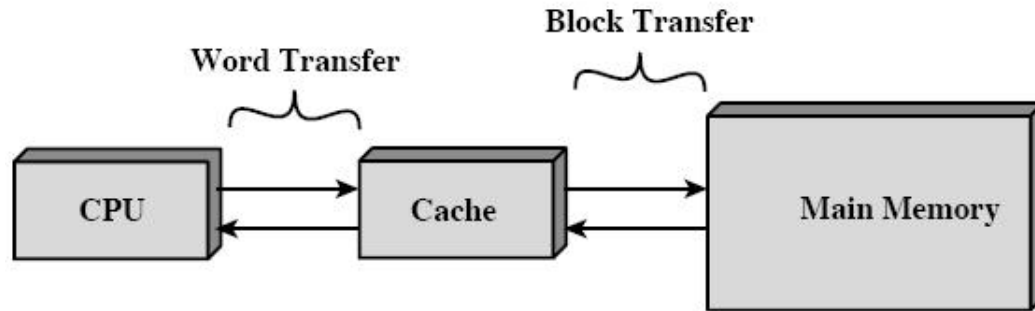


Cache Memory (*välimuisti*)

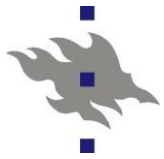
- How to access main memory as fast as registers?
- Locality → Use (CPU) cache!
 - Keep most probably referenced data in fast cache close to processor, and rest in memory
 - Most of data accesses only to cache
 - hit ratio 0.9-0.99
 - Cache is much smaller than main memory
 - Cache is (much) more expensive (per byte) than memory



Cache

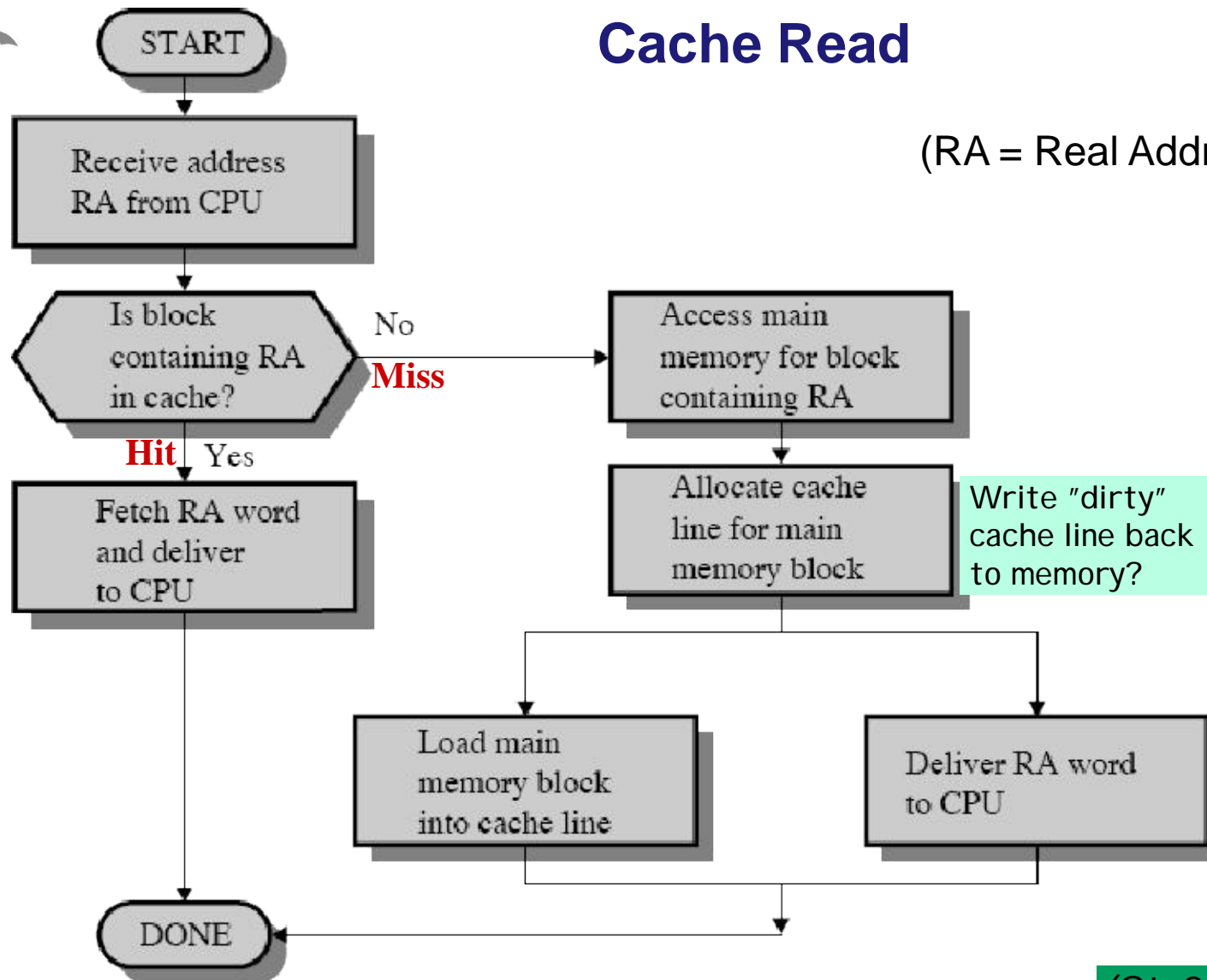


(Sta06 Fig 4.3, 4.4)



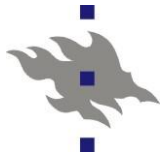
Cache Read

(RA = Real Address)

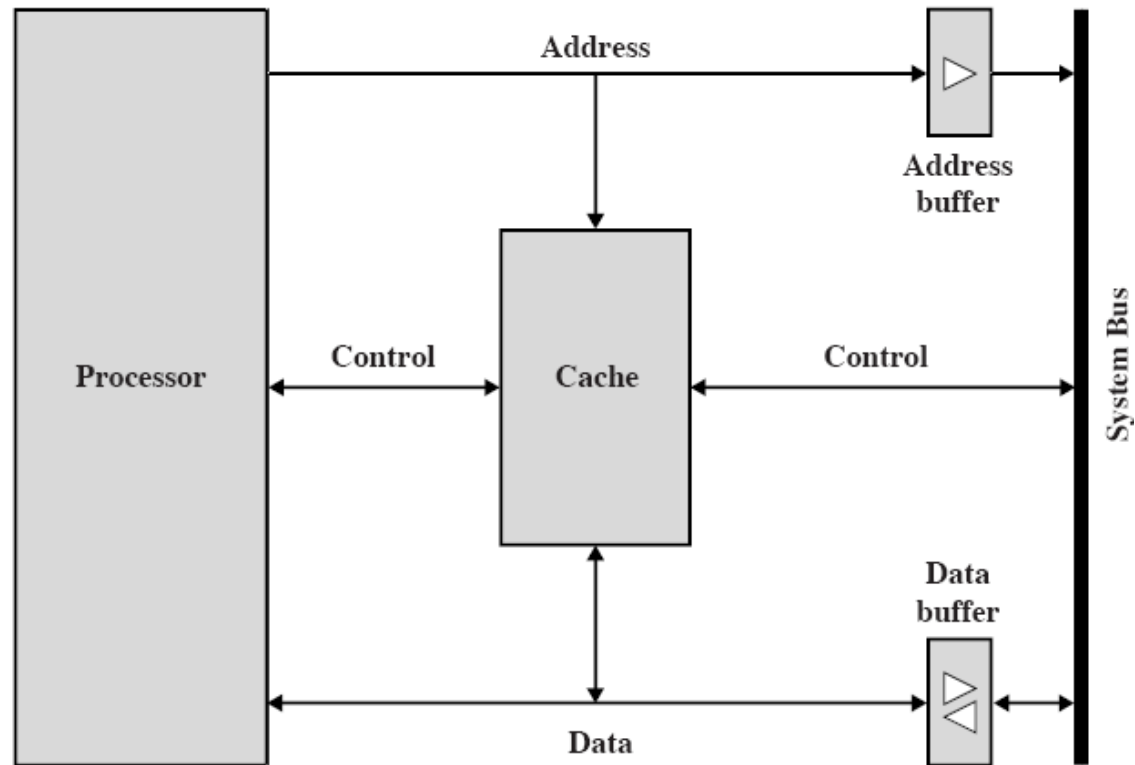


Write "dirty" cache line back to memory?

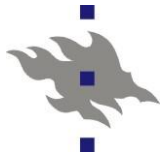
(Sta06 Fig 4.5)



Cache Organization



(Sta06 Fig 4.6)



Cache Design

Cache Size	Write Policy
Mapping Function	Write through
Direct	Write back
Associative	Write once
Set Associative	Line Size
Replacement Algorithm	Number of caches
Least recently used (LRU)	Single or two level
First in first out (FIFO)	Unified or split
Least frequently used (LFU)	
Random	

■ Cache Size & Line Size

- Many blocks help for temporal locality
- Large blocks help for spatial locality
- Larger cache is slower
- Multi-level cache

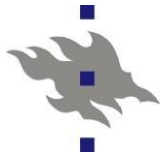
Typical sizes:

L1: 8 KB - 64 KB

L2: 256 KB - 8 MB

L3: 2 MB - 48 MB

(Sta09Table 4.3)



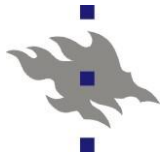
Mapping

- Which block contains the memory location?
- Is the block in cache?
- Where is it located?

- Solutions
 - direct mapping (*suora kuvaus*)
 - fully associative mapping (*täysin assosiatiivinen*)
 - set associative mapping (*joukkoassosiatiivinen*)

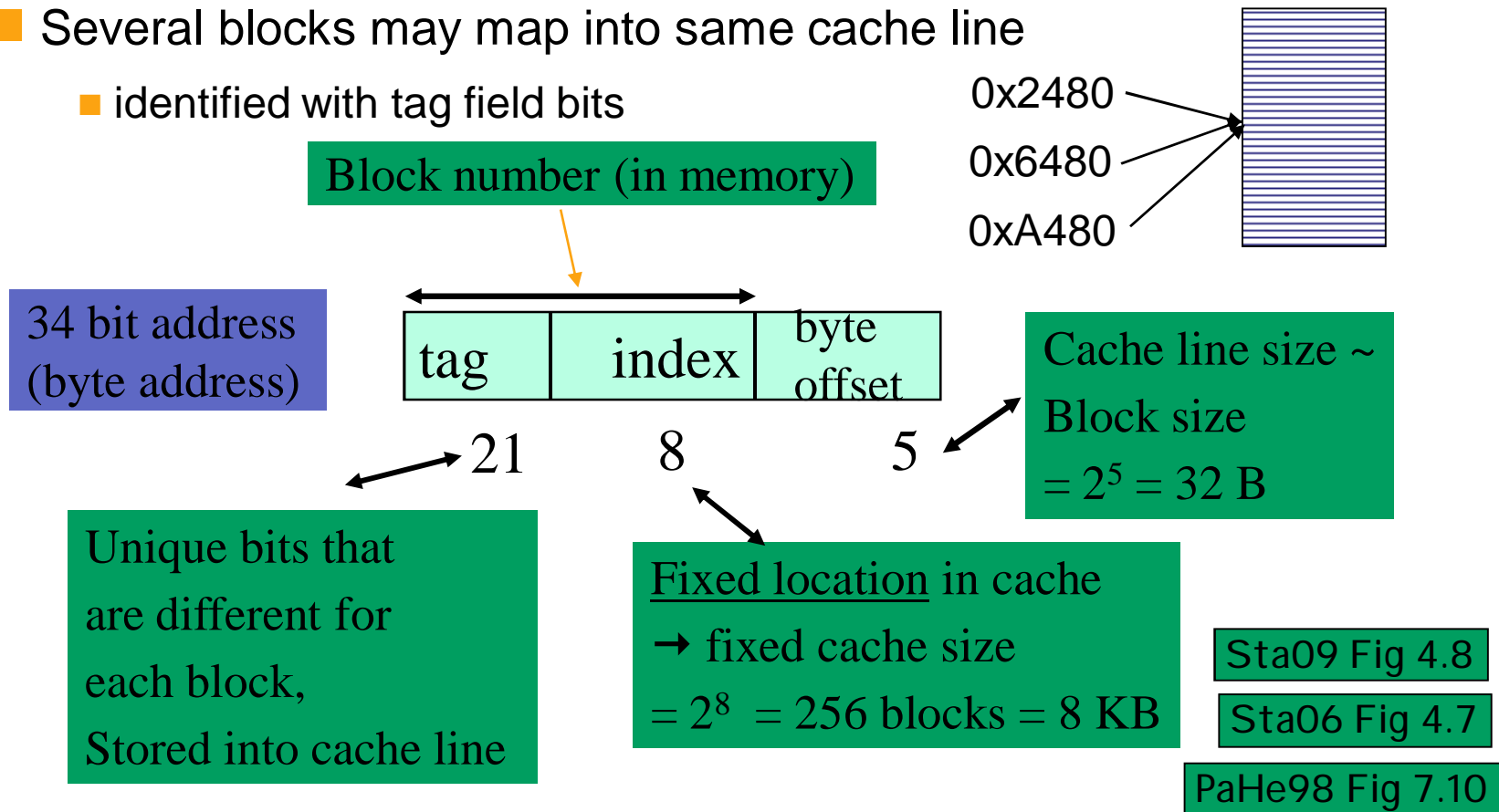
Cache simulation tools:

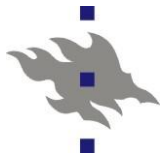
<http://www.ecs.umass.edu/ece/koren/architecture/Cache/frame0.htm>



Direct Mapping

- Each block has only one possible location (line) in cache
 - determined by index field bits
- Several blocks may map into same cache line
 - identified with tag field bits





Direct Mapping Example

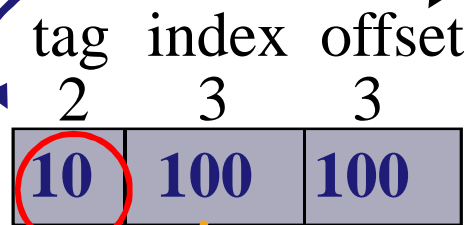
Word = 4B (here)

Block size = $2^3 = 8$ bytes = 64 bits

Cache line size

ReadW I2, 0xA4

8 bit address
(byte address)

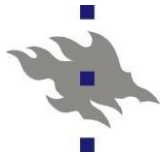


	tag	block, 64b
000:		
001:		
010:		
011:	01	54 A7 00 91 23 66 32 11
100:	11	77 55 55 66 66 22 44 22
101:	01	65 43 21 98 76 65 43 32
110:		

No match

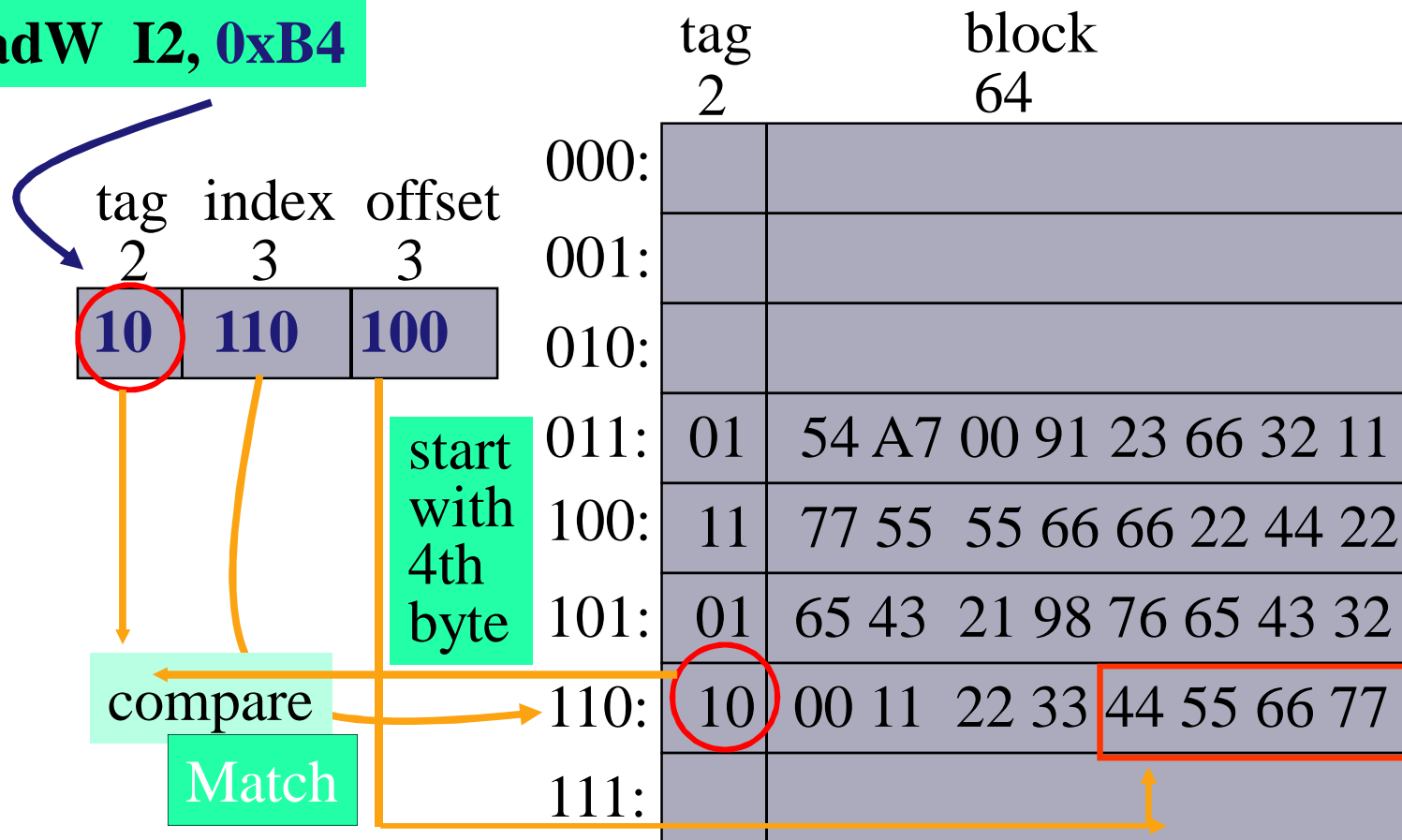
compare

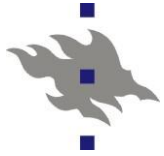
Read new memory block from memory address $0xA0=1010\ 0000$ to cache location 100, update tag, and then continue with data access



Direct Mapping Example 2

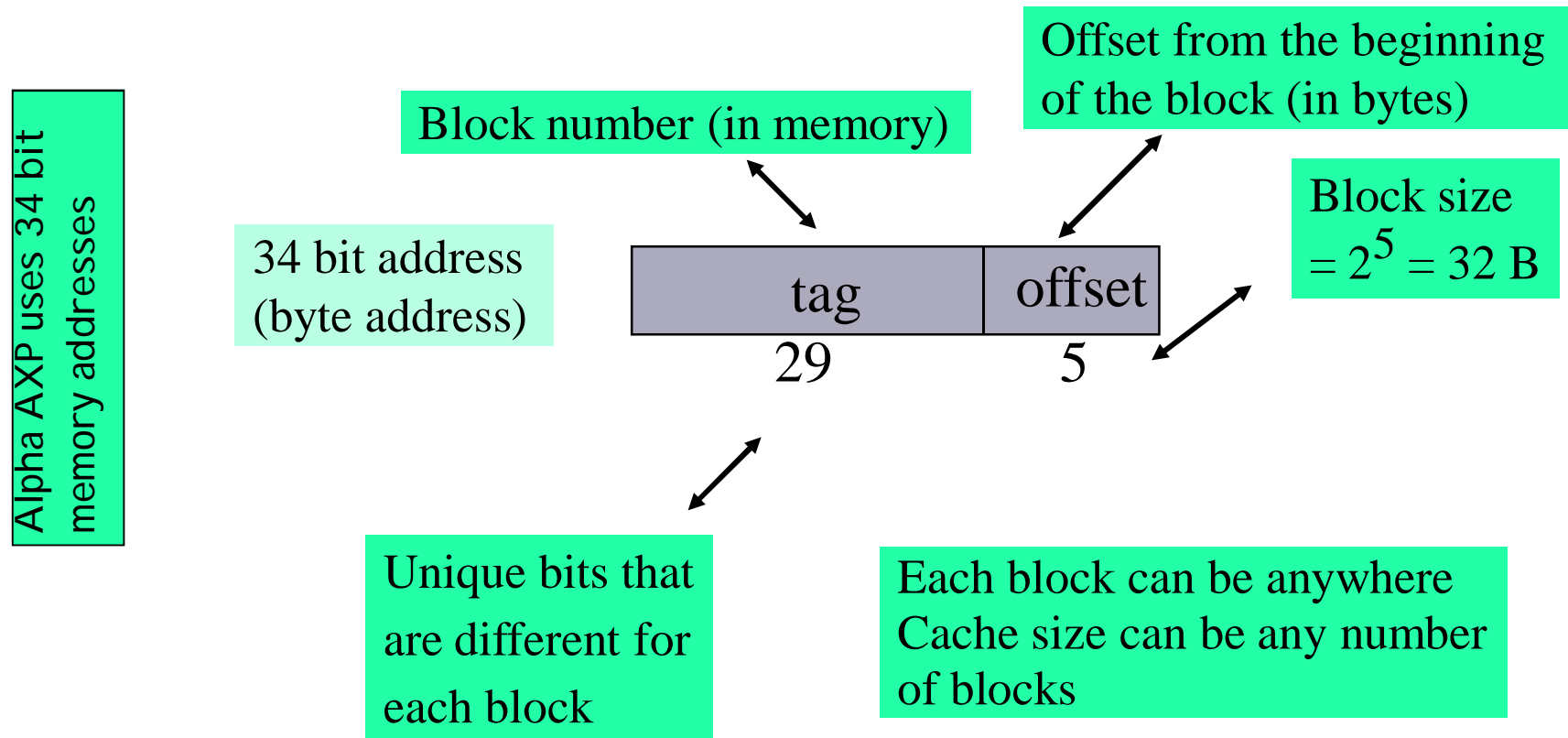
ReadW I2, 0xB4



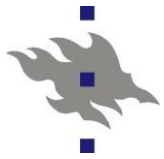


Fully Associative Mapping (6)

- Each block can be in any cache line
 - tag must be complete block number

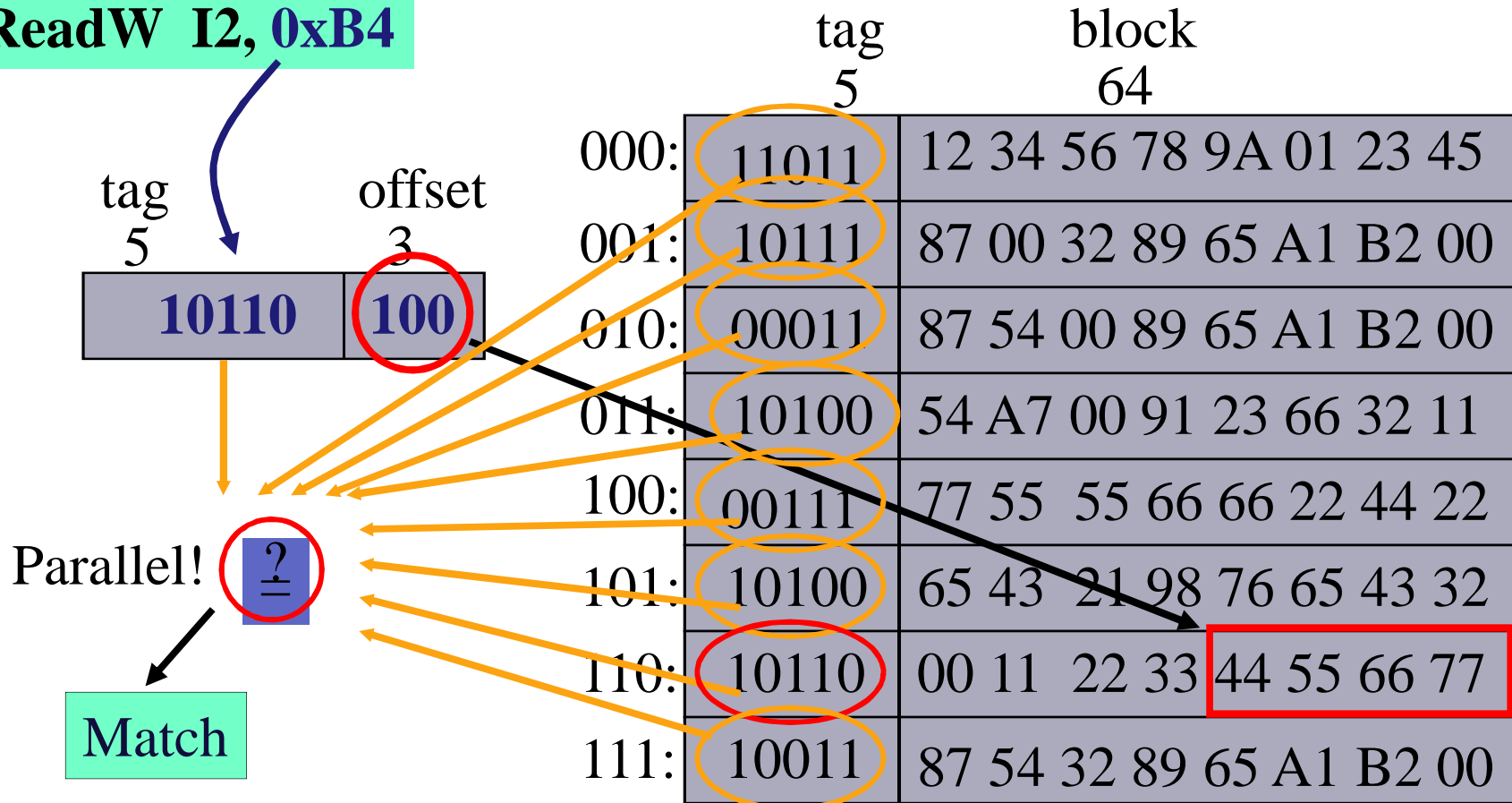


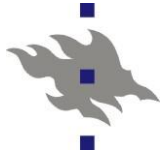
Sta06 Fig 4.9



Fully Associative Example

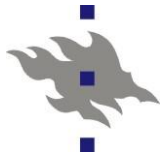
ReadW I2, 0xB4





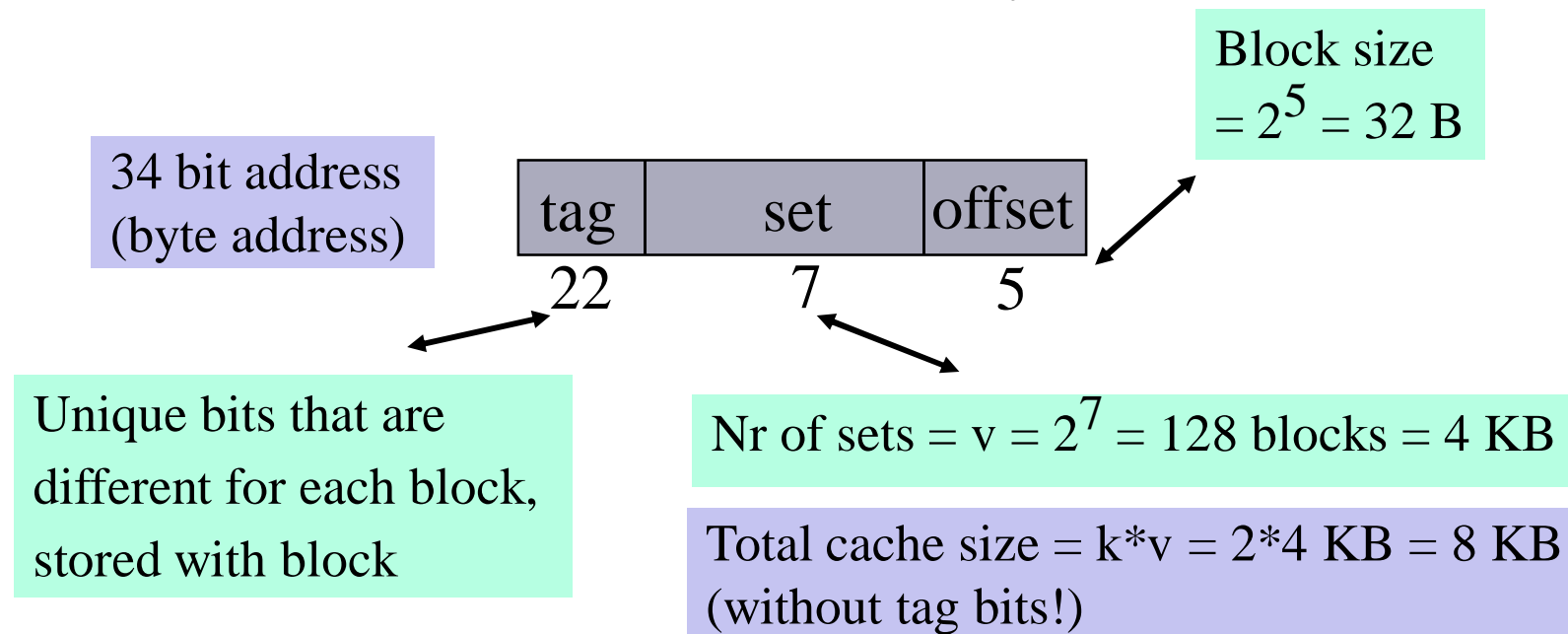
Fully Associative Mapping

- Lots of circuits
 - tag fields are long - wasted space?
 - each cache line tag must be compared parallelly
 - with the memory address tag
 - lots of wires, comparison circuits
 - large surface area on chip
- Final comparison “or” has large gate delay
 - did any of these 64 comparisons match?
 - $\log_2(64) = 6$ levels of binary OR-gates
 - how about 262144 comparisons?
 - 18 levels?
- \Rightarrow Can use it only for small caches



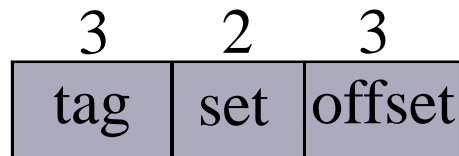
Set Associative Mapping

- With set size $k=2$, each cache entry contain 2 blocks
 - Use set (set index) field to find the cache entry
 - Use tag to determine if the block belongs to the set
 - Use offset to find the proper byte in the block



2-way Set Associative Cache

- $k=2 \rightarrow$ Two blocks in each set (= in one cache entry)
- 4 sets \rightarrow 2 bits for set index
- 2 words in a block = 8 Bytes \rightarrow 3 bits for byte offset
- 3 bits for tag



8 bit address
(byte address)

set	tag	block	tag	block
00:	110	12 34 56 78 9A 01 23 45	011	77 55 55 66 66 22 44 22
01:	110	87 00 32 89 65 A1 B2 00	101	65 43 21 98 76 65 43 32
10:	100	87 54 00 89 65 A1 B2 00	101	00 11 22 33 44 55 66 77
11:	101	54 A7 00 91 23 66 32 11	111	00 11 22 33 44 55 66 77
	3	64	3	64



2-way Set Assoc. Cache Example

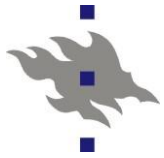
ReadW I2, 0xB4



Parallel!

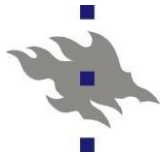
?

Match



Set Associative Mapping

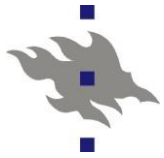
- Set associative cache with set size $k=2$
 - = 2-way cache (common)
 - Degree of associativity = nbr of blocks in a set = v
 - Large degree of associativity?
 - More data items in one set
 - Less “collisions” within set
 - Final comparison (matching tags?) gate delay?
 - Maximum (nr of cache lines)
 - ⇒ fully associative mapping
 - Minimum (1)
 - ⇒ direct mapping
- Whole cache is one set!
- Each cache line is a set!



Cache Replacement Algorithm

- Which cache block to replace to make room for new block from memory?

- Direct mapping: trivial
- First-In-First-Out (FIFO)?
- Least-Frequently-Used (LFU)?
- Random?
- Which one is best / possible?
 - Chip area?
 - Fast? Easy to implement?



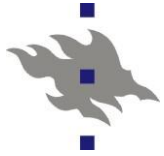
Cache Write Policy – memory writes?

- Write through (*läpikirjoittava*)
 - Each write goes always to cache and memory
 - Each write is a cache miss!
- Write back (*lopuksi/takaisin kirjoittava*)
 - Each write goes only to cache
 - Write cache block back to memory
 - only when it is replaced in cache
 - Memory may have stale (old) data
 - cache coherence problem (*eheys, yhdenmukaisuus, yhtäpitävyys*)
- Write once (*"vain kerran kirjoittava?"*)
 - Write-invalidate Snoopy-cache coherence protocol for multiprocessors
 - Write invalidates data in other caches
 - Write to memory at replacement time, or when some other cache needs it (has read/write miss)

A bit set

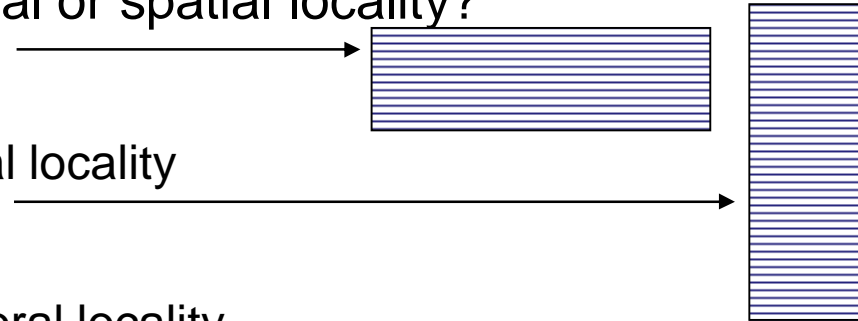
Coherence problems:

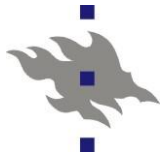
- More users of the same data: memory valid? cache valid?
- multiple processors with own caches



Cache Line Size

- How big cache line?
- Optimise for temporal or spatial locality?
 - bigger cache line
 - ⇒ better for spatial locality
 - more cache lines
 - ⇒ better for temporal locality
- Best size varies with program or program phase?
- Best size different with code and data?
- 2-8 words?
 - word = 1 float??

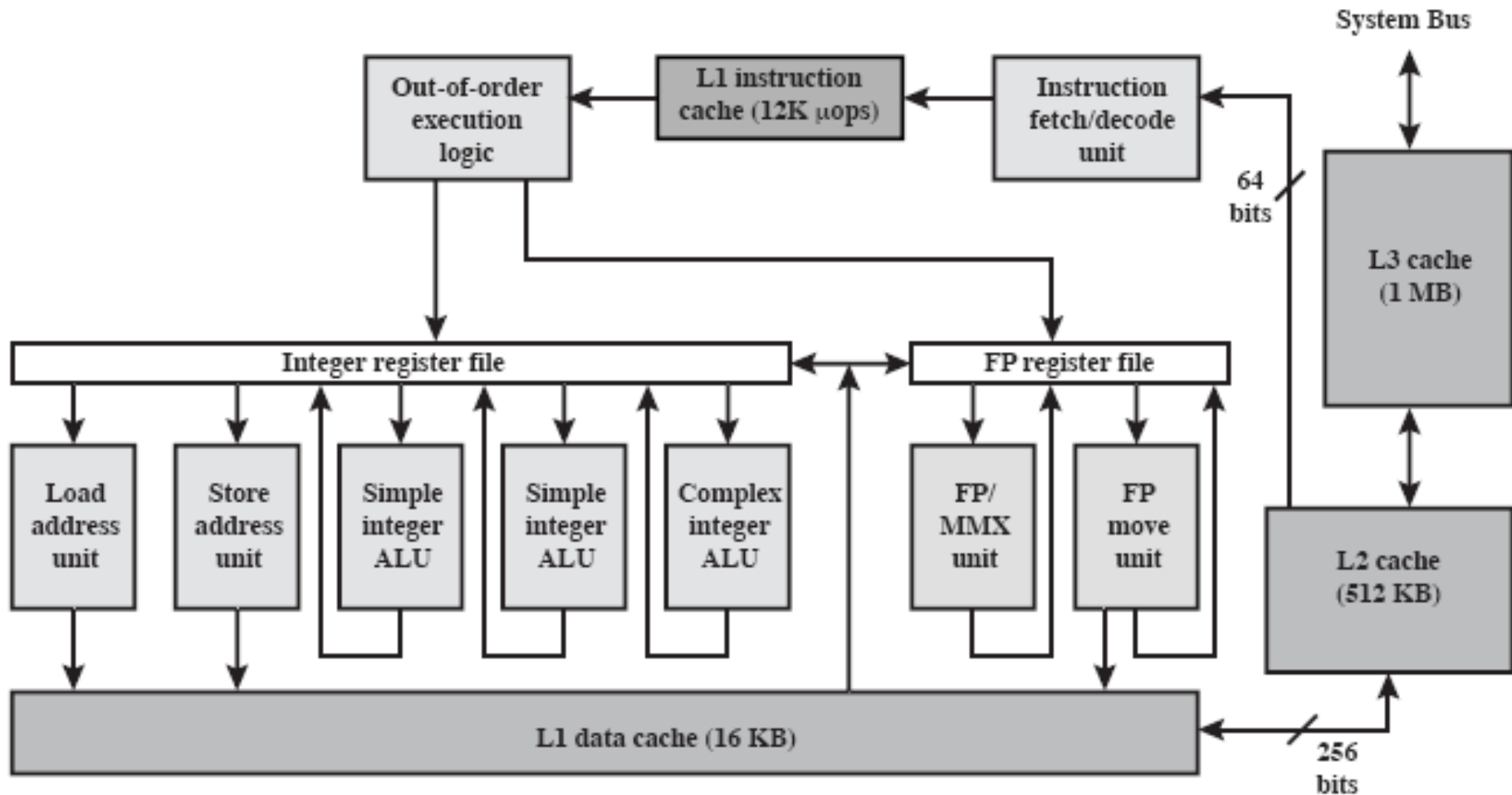




Types and Number of Caches

- Same cache for data and code, or not?
 - Data references and code references behave differently
- Unified vs. split cache (*yhdistetty/erilliset*)
 - split cache: can optimise structure separately for data and code
 - Trend towards split caches: Pentium, Power PC, ARM.. (instruction pipelining)
- One cache too large for best results
- Multiple levels of caches
 - L1 on same chip as CPU
 - L2 on same package or chip as CPU
 - older systems: same board
 - L3 on same board as CPU

Example: Pentium 4 Block Diagram

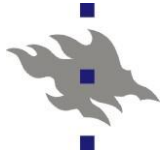


L1: split, 4-way set-associative, line size 64 B

L2, L3: unified, 8-way set-associative, line size 128 B

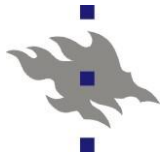
(Sta09 Fig 4.18)

(Sta06 Fig 4.13)



Computer Organization II

Main Memory



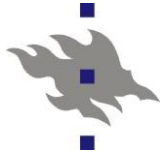
Main Memory Types

(katoava,
haihtuva)

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

(Sta06 Table 5.1)

- Random access semiconductor memory
 - Direct access to each memory cell
 - Access time same for all cells



RAM

■ Dynamic RAM, DRAM

Analog: Charge on capacitors

- Periodic refreshing required
- Refresh required after read
- Simpler, slower, denser, bigger (bytes per chip)
- Access time ~ 60 ns
- Main memory? (early systems)

■ Static RAM, SRAM

Digital: flip-flop gates

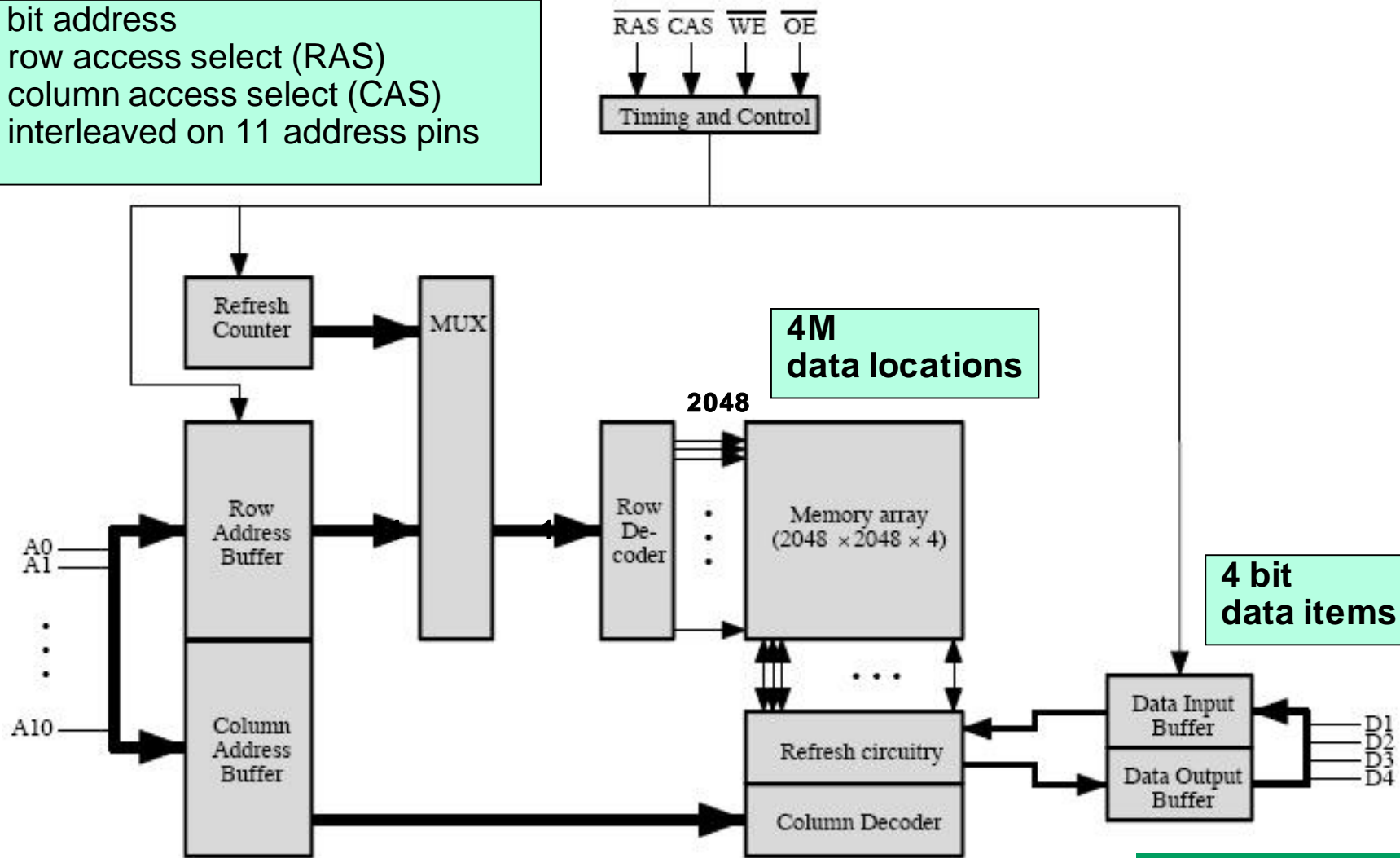
- No periodic refreshing needed
- Data remains until power is lost
- More complex (more chip area/byte), faster, smaller
- Access time ~ 2-5 ns
- Level 2 cache?



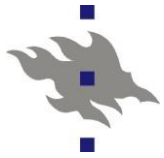
DRAM Access, 16 Mb DRAM (4M x 4)

22 bit address

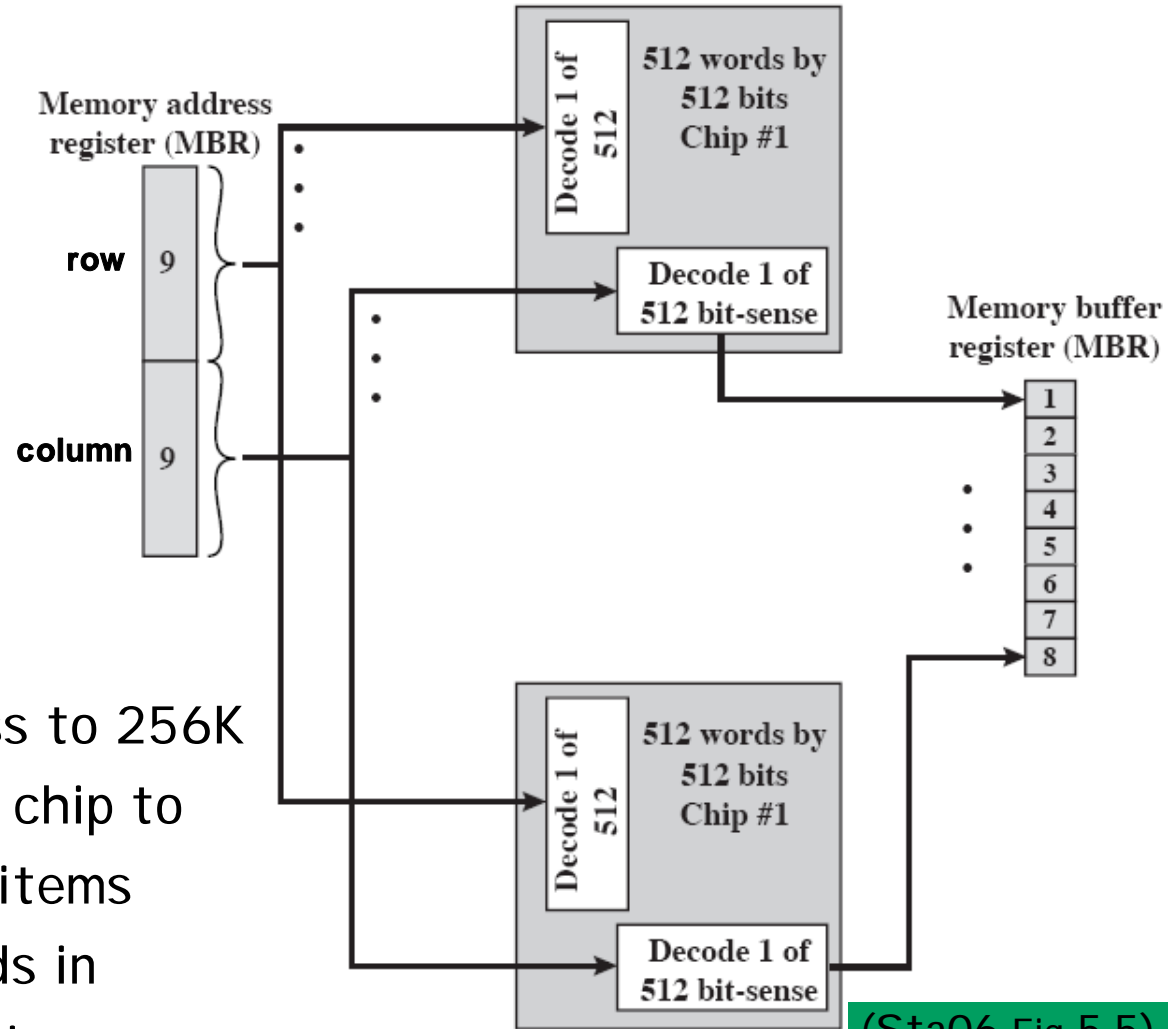
- row access select (RAS)
- column access select (CAS)
- interleaved on 11 address pins



(Sta06 Fig 5.3)

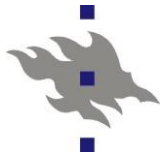


256-KB DRAM Memory Organization



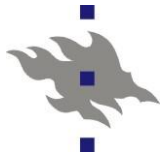
(Sta06 Fig 5.5)

- Simultaneous access to 256K 8-bit word memory chip to access larger data items
- Access 64-bit words in parallel? Need 8 chips.

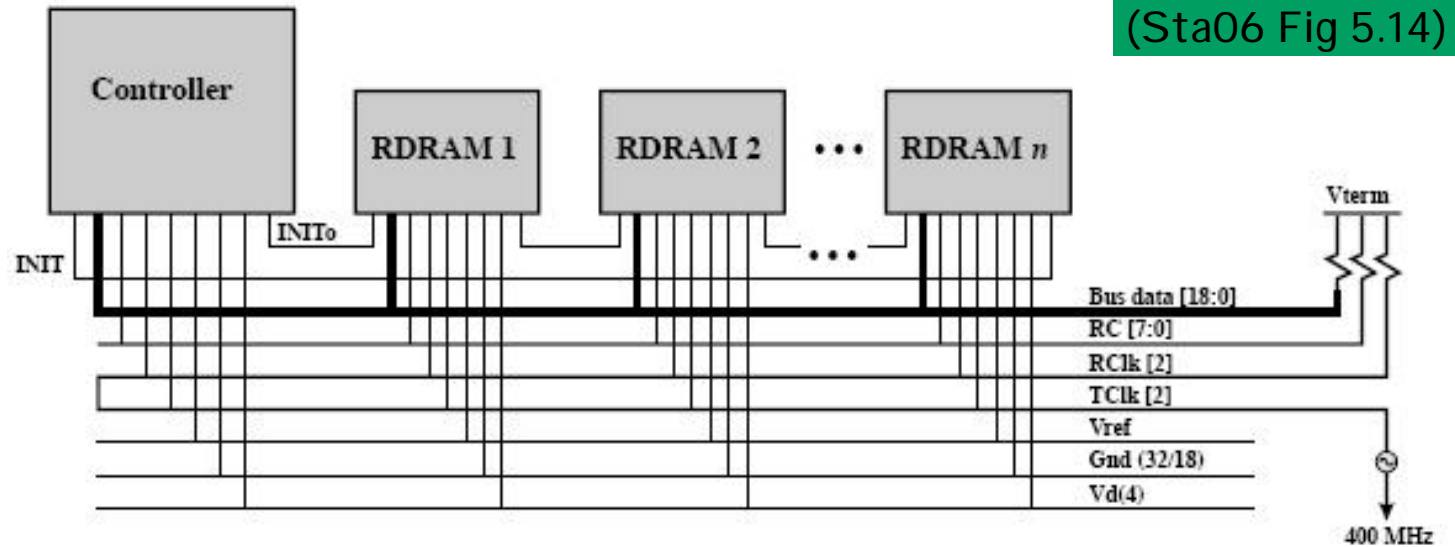


SDRAM (Synchronous DRAM)

- CPU clock synchronizes also the bus
 - Runs on higher clock speeds than ordinary DRAM
 - CPU knows how long it takes to make a reference,
 - can do other work while waiting
- 16 bits in parallel
 - Access 4 DRAMs (4 bits each) in parallel
 - Access time ~ 18 ns, transfer rate ~ 1.3 GB/s
- DDR SDRAM, double data rate
 - Current main memory technology
 - Supports transfers both on rising and falling edge of the clock cycle
 - Consumes less power
 - Access time ~ 12 ns, transfer rate ~ 3.2 GB/s

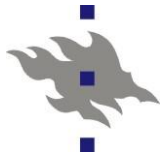


Rambus DRAM (RDRAM)



- Works with fast Rambus memory bus (800Mbps)
 - Controller + RDRAM modules
 - Access time ~ 12 ns, transfer rate ~ 4.8 GB/s
- Speed slows down with many memory modules
 - Serially connected on Rambus channel
 - Not good for servers with 1 GB memory (for now!)

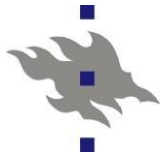
STI Cell processor



Flash memory

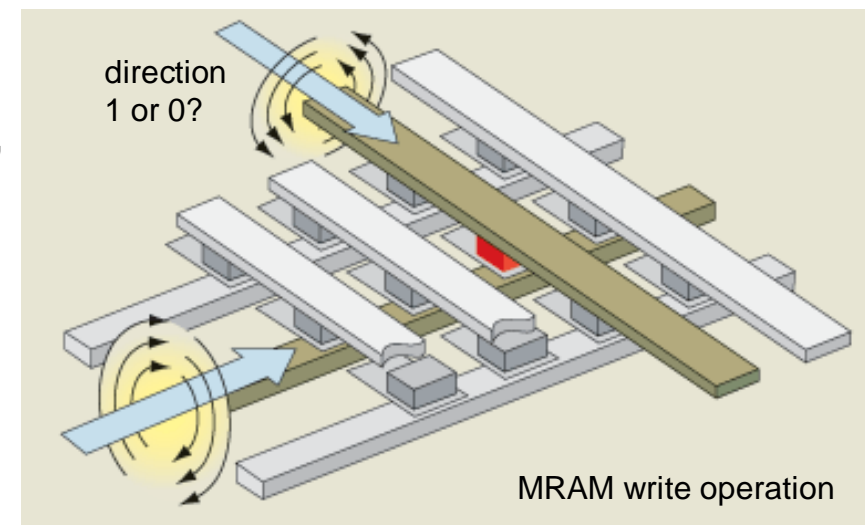
- Based on transistors that are separated by a thin oxide layer
 - Flash cell is analog, not digital storage:
uses different charge levels to store 2 (or more) bits
in each cell
- Non-volatile, data remains with power off
 - Electrical erasing in blocks = "flash"
 - Slow to write
 - Access time ~ 50 ns
- Used as a solid state storage
 - No moving parts
 - FlashBIOS in PC's, USB-memory
 - In phones, digital cameras, hand-held devices,.....



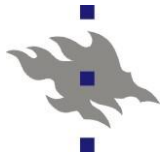


MRAM

- Magnetoresistive Random Access Memory (MRAM)
 - Data stored with magnetic fields on two plates
 - Magnetic field directions determine bit value
- Non-volatile, data remains with power off
 - Fast to read/write
 - No upper limit for write counts (Flash has upper limit)
 - Access time comparable to DRAM
 - Almost as fast as SRAM
- Future open
 - Small market share now
 - Expensive now (2006: \$25 4Mbit, 2008: \$15 4Mbit, Freescale)
 - Still under development
 - May replace flash in a few years
 - May replace SRAM later on
 - May replace DRAM and become "universal memory"



<http://www.research.ibm.com/journal/rd/501/maffitt.html>



Kertauskysymyksiä/Review questions

- Memory hierarchy and principle of locality?
- Different ways to use locality in cache solutions?
- Differences of associative and set associative mappings?
- Why to have separate caches for instructions and data ?

- Muistihierarkia ja paikallisuus?
- Millä tavoin paikallisuus huomioidaan välimuistiratkaisussa?
- Assosiatiivisen ja joukkoassosiatiivisen kuvauksen erot?
- Miksi käskyille on oma välimuisti ja datalle oma?