Memory Hierarchy and Cache Ch 4.1-3

Memory Hierarchy

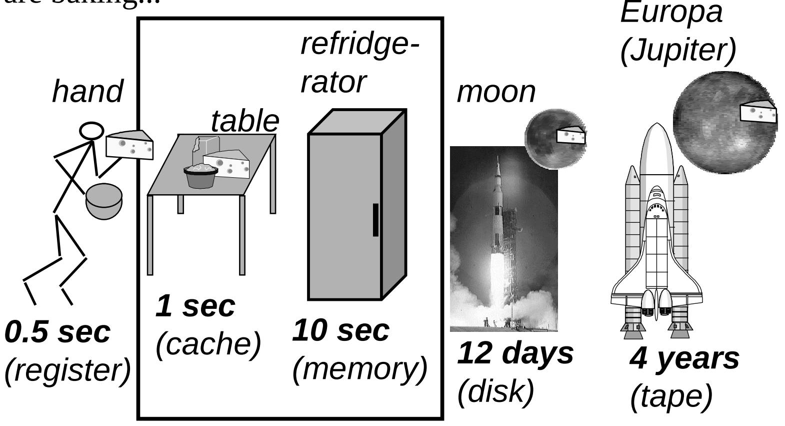
Main Memory

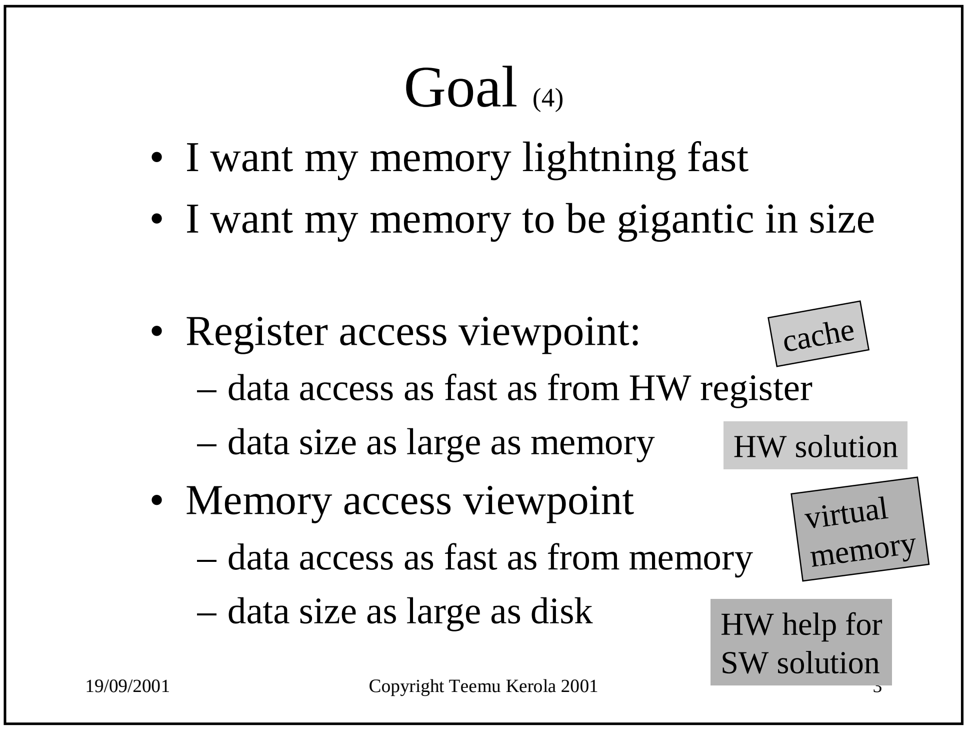
Cache

Implementation

Teemu’s Cheesecake

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





Goal (4)

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

- Register access viewpoint:
  - data access as fast as from HW register
  - data size as large as memory
- Memory access viewpoint
  - data access as fast as from memory
  - data size as large as disk

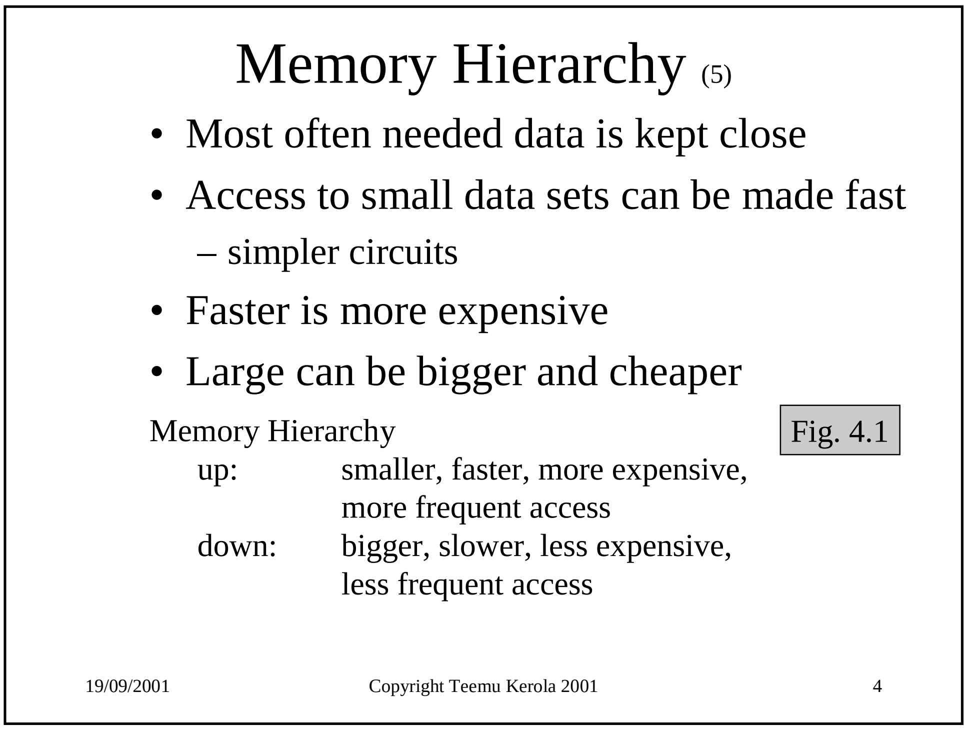
cache

HW solution

virtual memory

HW help for SW solution

19/09/2001 Copyright Teemu Kerola 2001 3



Memory Hierarchy (5)

- Most often needed data is kept close
- Access to small data sets can be made fast
  - simpler circuits
- Faster is more expensive
- Large can be bigger and cheaper

Fig. 4.1

Memory Hierarchy

up:	smaller, faster, more expensive, more frequent access
down:	bigger, slower, less expensive, less frequent access

19/09/2001 Copyright Teemu Kerola 2001 4

Principle of locality (7)

(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 µs
Cost (the rest) = 20 µs

Aver cost 99% * 2 µs + 1% * 20 µs = 2.2 µs

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

Principle of locality (5)

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

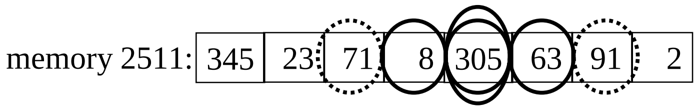
(paikallisuus)

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

(ajallinen paikallisuus)

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

(alueellinen paikallisuus)



Memory

- Random access semiconductor memory
  - give address & control, read/write data
- ROM, PROMS

Table 4.2

  - system startup memory, BIOS (Basic Input/Output System)
    - load and execute OS at boot
  - also random access
- RAM
  - “normal” memory accessible by CPU

RAM

E.g., $0.12 / MB (year 2001)?

- Dynamic RAM, DRAM
  - simpler, slower, denser, bigger (bytes per chip)
  - main memory?

E.g., 60 ns access

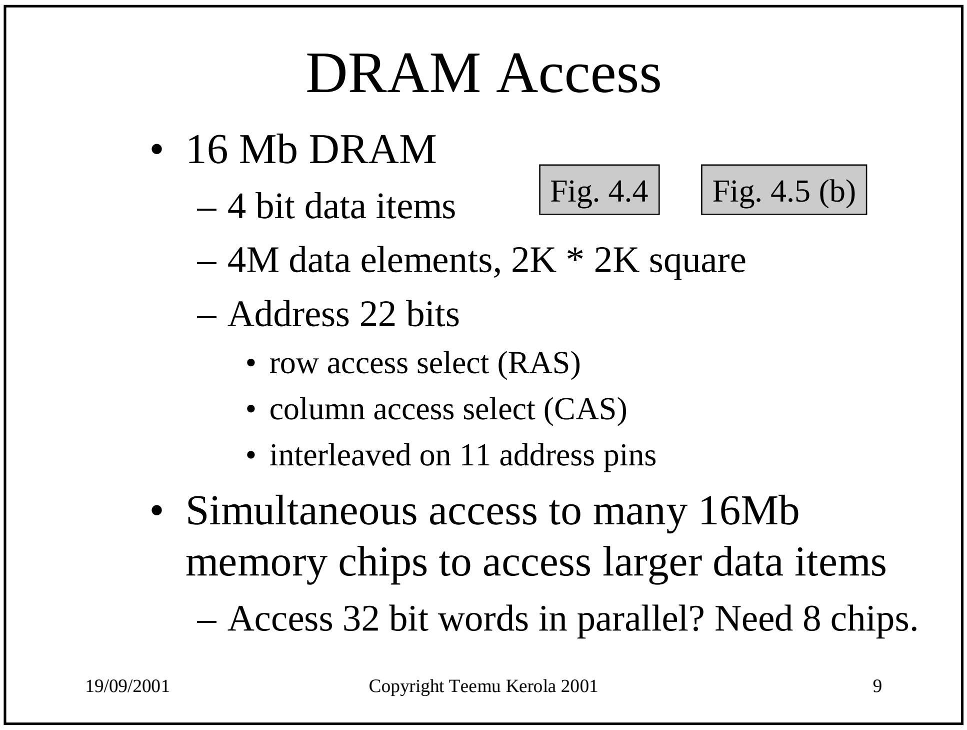
  - periodic refreshing required
  - refresh required after read
- Static RAM, SRAM

E.g., $0.50 / MB (year 2001)?

  - more complex (more chip area/byte), faster, smaller (bytes per chip)

E.g., 5 ns access?

  - cache?
  - no periodic refreshing needed
  - data remains until power is lost

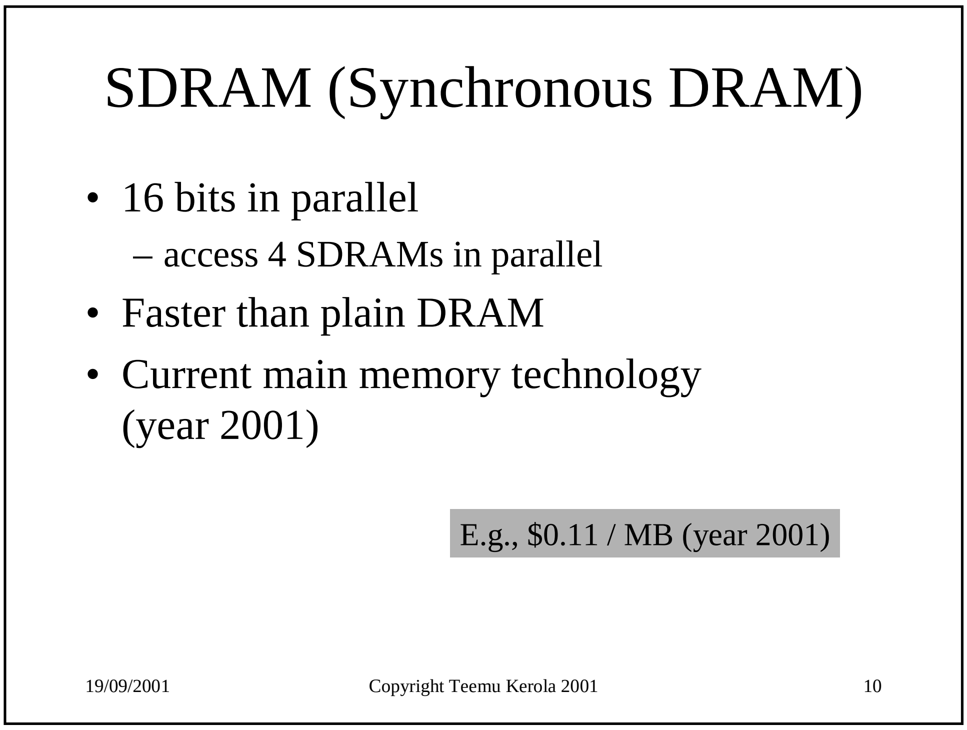


DRAM Access

- 16 Mb DRAM
  - 4 bit data items
  - 4M data elements, 2K * 2K square
  - Address 22 bits
    - row access select (RAS)
    - column access select (CAS)
    - interleaved on 11 address pins
- Simultaneous access to many 16Mb memory chips to access larger data items
  - Access 32 bit words in parallel? Need 8 chips.

Fig. 4.4

Fig. 4.5 (b)



SDRAM (Synchronous DRAM)

- 16 bits in parallel
  - access 4 SDRAMs in parallel
- Faster than plain DRAM
- Current main memory technology (year 2001)

E.g., $0.11 / MB (year 2001)

RDRAM (RambusDRAM)

- New technology, works with fast memory bus
  - expensive
  E.g., $0.40 / MB (year 2001)?
- Faster transfer rate than with SDRAM
  E.g., 1.6 GB/sec vs. 200 MB/sec (?)
- Faster access than SDRAM
  E.g., 38 ns vs. 44 ns
- Fast internal Rambus channel (800 MHz)
- Rambus memory controller connects to bus
- Speed slows down with many memory modules
  - serially connected on Rambus channel
  - not good for servers with 1 GB memory (for now!)
- 5% of memory chips (year 2000)

Cache Memory

(välimuisti)

- Problem: how can I make my (main) memory as fast as my registers?
- Answer: (processor) cache
  - – keep most probably referenced data in fast cache close to processor, and rest of it in memory
    - much smaller than main memory
    - (much) more expensive (per byte) than memory
    - most of data accesses to cache

90% 99%?

Fig. 4.13 Fig. 4.16

Memory references with cache (5)

- Data is in cache?

Hit

Fig. 4.15

Data is only in memory?
    Read it to cache
    CPU waits until data available

Miss

Many blocks (cache lines) help for temporal locality
    many different data items in cache

Fig. 4.14

Large blocks help for spatial locality
    lots of “nearby” data available

Fixed cache size?
    Select “many” or “large”?

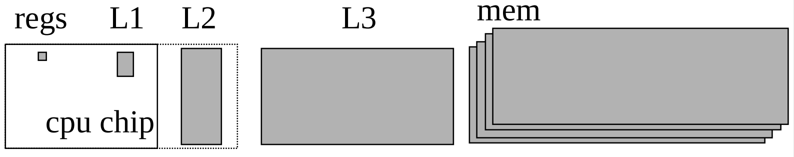
Cache Features (6)

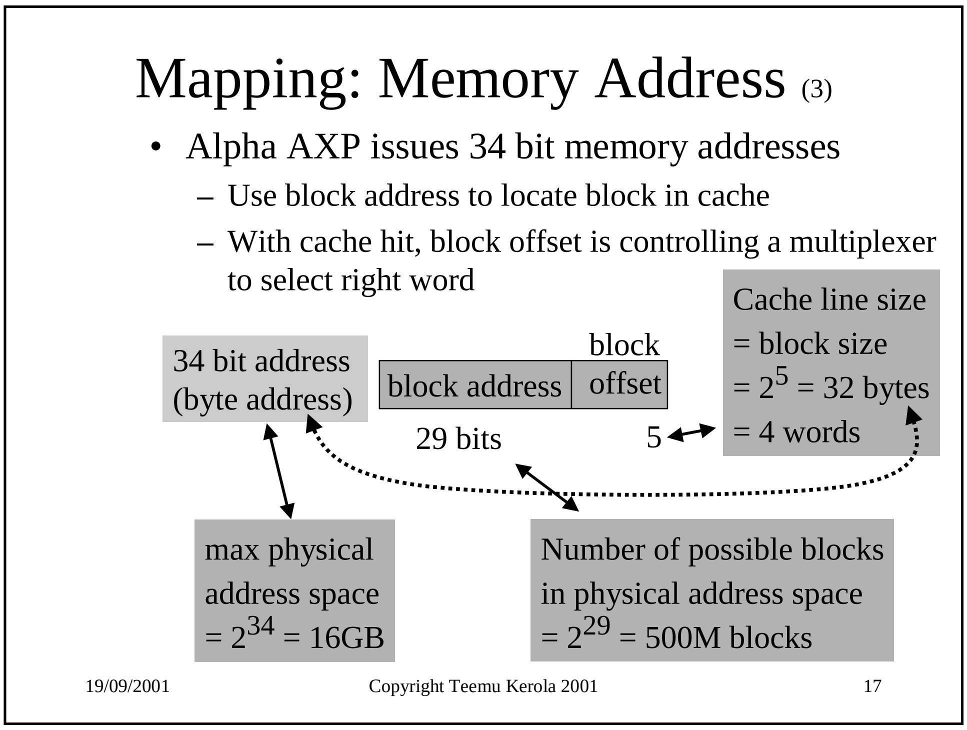
- Size
- Mapping function (kuvausfunktio)
  - how to find data in cache?
- Replacement algorithm (poistoalgoritmi)
  - which block to remove to make room for a new block?
- Write policy (kirjoituspolitiikka)
  - how to handle writes?
- Line size (block size)? (rivin tai lohkon koko)
- Number of caches?

Cache Size

- Bigger is better in general
- Bigger may be slower
  - lots of gates, cumulative gate delay?
- Too big might be too slow!
  - Help: 2- or 3-level caches

1KW (4 KB),
512KW (2 MB)?





Mapping: Memory Address (3)

- Alpha AXP issues 34 bit memory addresses
  - Use block address to locate block in cache
  - With cache hit, block offset is controlling a multiplexer to select right word

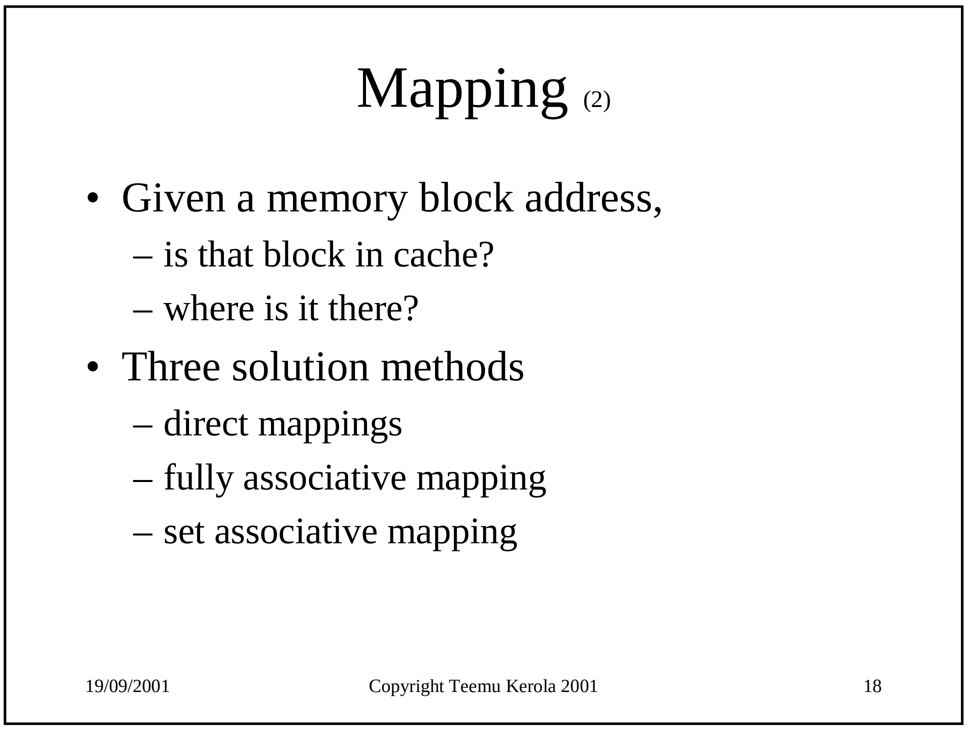
34 bit address (byte address)

block address	block offset
29 bits	5

Cache line size = block size = 2^5 = 32 bytes = 4 words

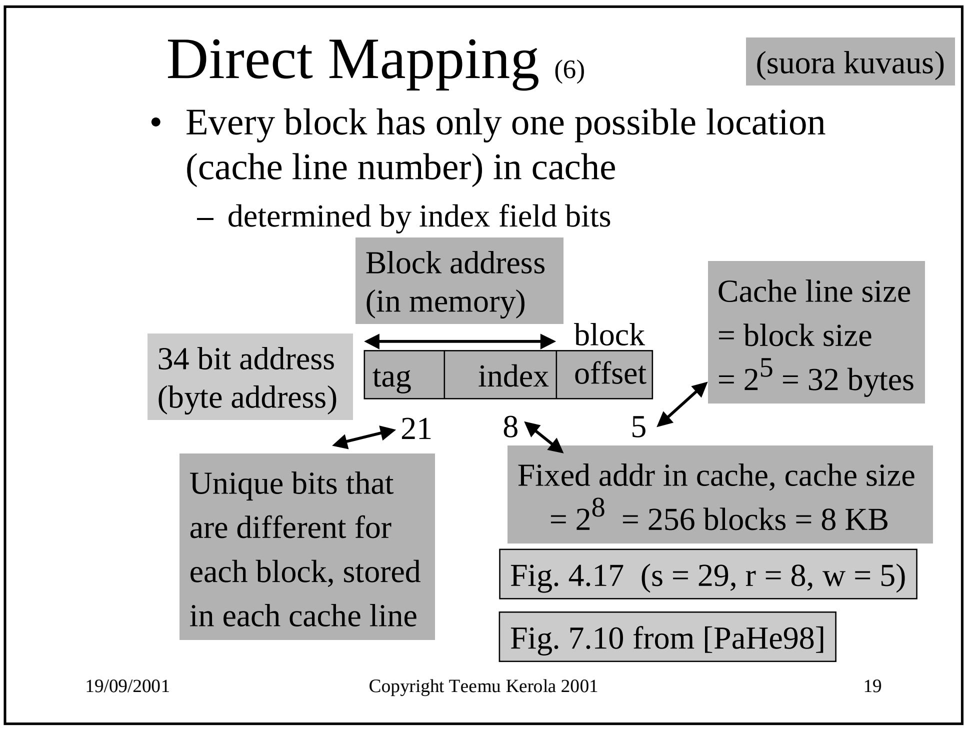
max physical address space = 2^{34} = 16GB

Number of possible blocks in physical address space = 2^{29} = 500M blocks



Mapping (2)

- Given a memory block address,
  - is that block in cache?
  - where is it there?
- Three solution methods
  - direct mappings
  - fully associative mapping
  - set associative mapping



Direct Mapping (6)

(suora kuvaus)

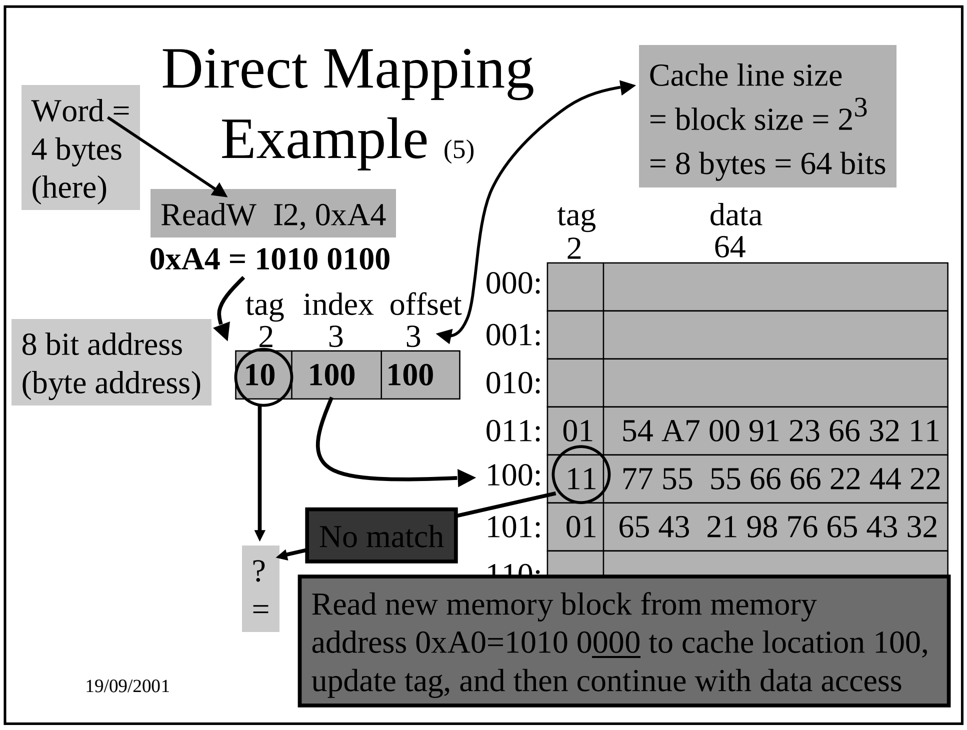
- Every block has only one possible location (cache line number) in cache
  - determined by index field bits

Block address (in memory)
34 bit address (byte address)
tag
index
block offset
21
8
5
Cache line size = block size = 2^5 = 32 bytes
Unique bits that are different for each block, stored in each cache line
Fixed addr in cache, cache size = 2^8 = 256 blocks = 8 KB

Fig. 4.17 (s = 29, r = 8, w = 5)

Fig. 7.10 from [PaHe98]

19/09/2001 Copyright Teemu Kerola 2001 19



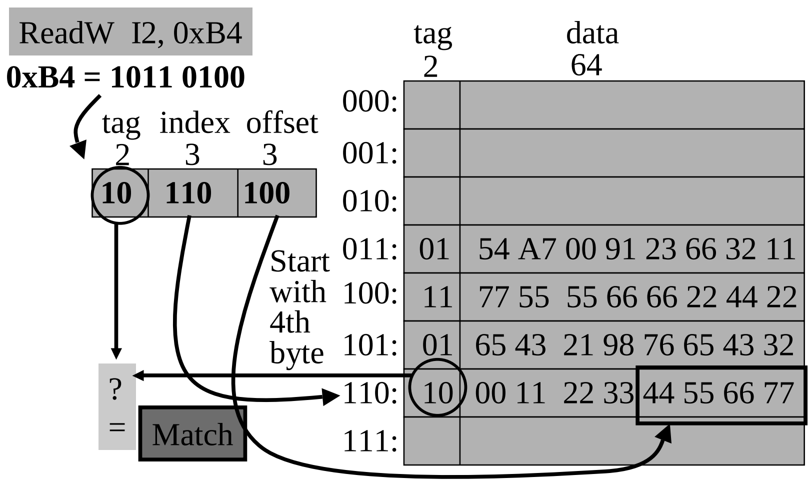
Direct Mapping Example (5)

	tag 2	data 64
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:		

Read new memory block from memory address 0xA0=1010 0<u>000</u> to cache location 100, update tag, and then continue with data access

19/09/2001

Direct Mapping Example 2 (5)



Fully Associative Mapping (5)

(täysin assosia-tiivinen kuvaus)

- Every block can be in any cache line
  - tag must be complete block address

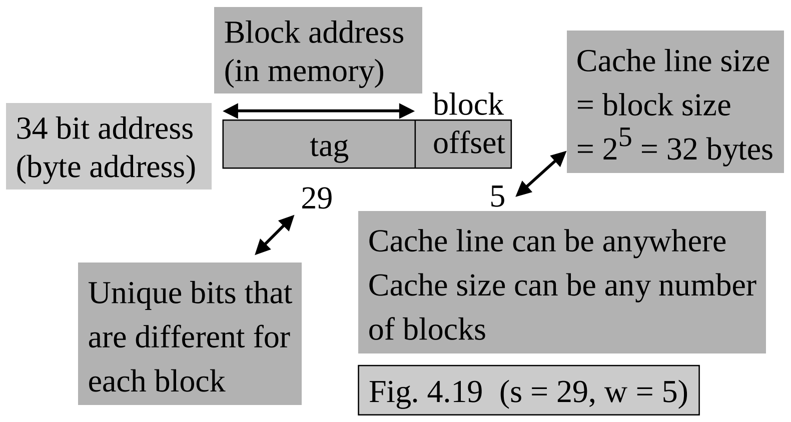

Block address (in memory)
34 bit address (byte address)
tag
block offset
29
5
Cache line size = block size = 2^5 = 32 bytes
Unique bits that are different for each block
Cache line can be anywhere
Cache size can be any number of blocks

Fig. 4.19 (s = 29, w = 5)

Fully Associative Mapping

- Lots of circuits
  - tag fields are long - wasted space!
  - each cache line tag must be compared simultaneously with the memory address tag
    - lots of wires
    - lots of comparison circuits

Large surface area on chip

- Final comparison “or” has large gate delay
  - did any of these 64 comparisons match?

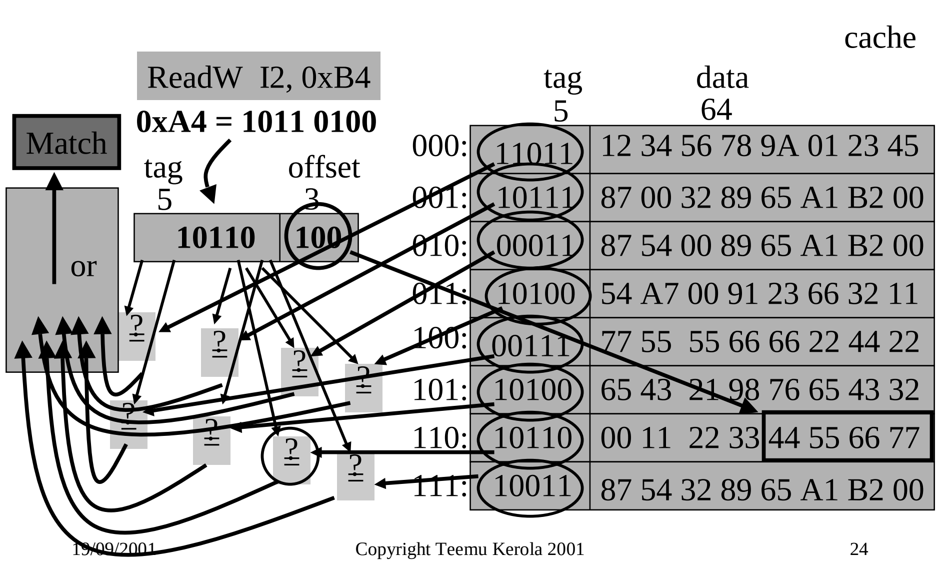
$^2\log(64) = 8$ levels of binary gates

  - how about 262144 comparisons?

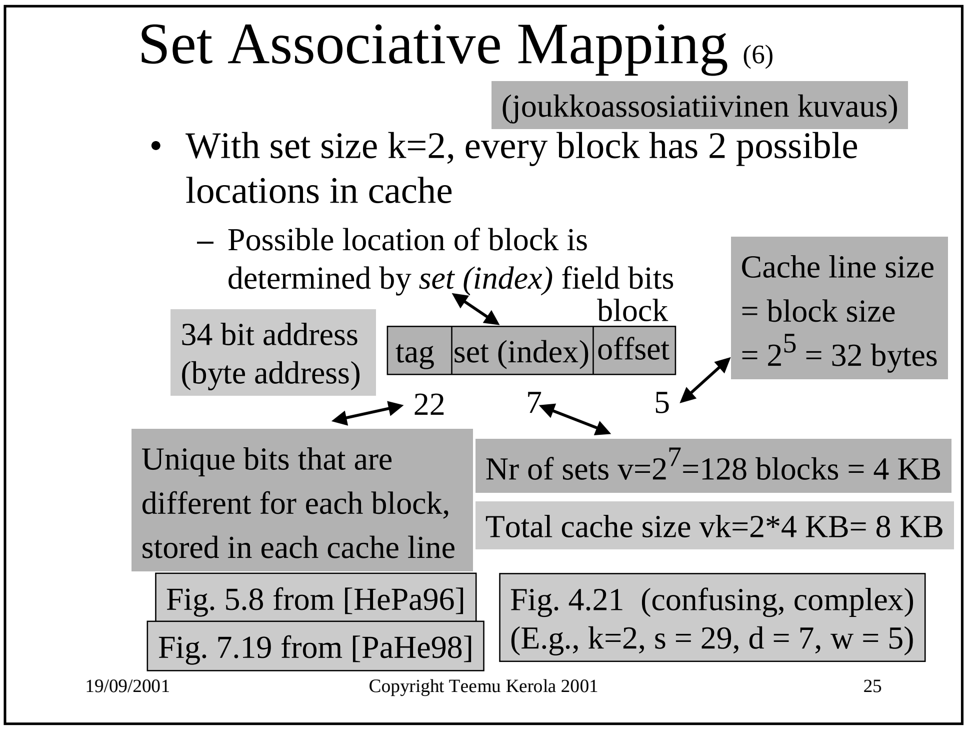
18 levels?

- $\Rightarrow$ Can use it only for small caches

Fully Associative Example (5)



	tag	data
000:	11011	12 34 56 78 9A 01 23 45
001:	10111	87 00 32 89 65 A1 B2 00
010:	00011	87 54 00 89 65 A1 B2 00
011:	10100	54 A7 00 91 23 66 32 11
100:	00111	77 55 55 66 66 22 44 22
101:	10100	65 43 21 98 76 65 43 32
110:	10110	00 11 22 33 44 55 66 77
111:	10011	87 54 32 89 65 A1 B2 00



Set Associative Mapping (6)

(joukkoassosiatiivinen kuvaus)

- With set size k=2, every block has 2 possible locations in cache
  - Possible location of block is determined by *set (index)* field bits

tag	set (index)	block offset
22	7	5

34 bit address (byte address)

Cache line size = block size = 2^5 = 32 bytes

Unique bits that are different for each block, stored in each cache line

Nr of sets v=2^7=128 blocks = 4 KB

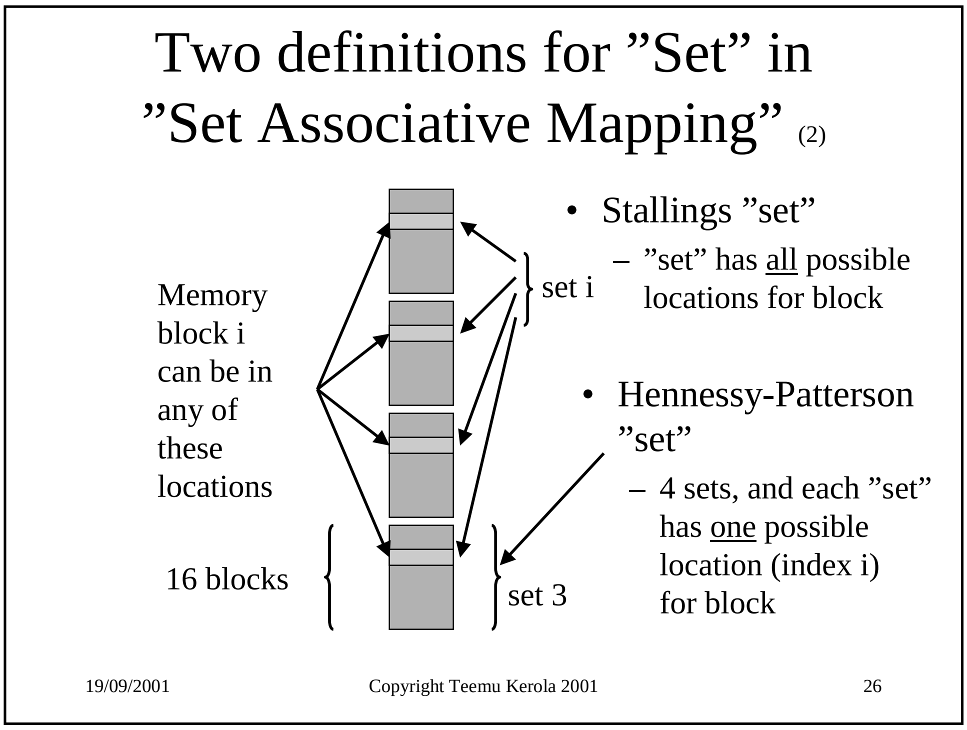
Total cache size vk=2*4 KB= 8 KB

Fig. 5.8 from [HePa96]

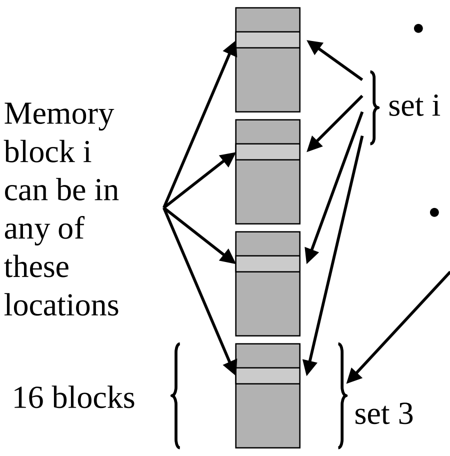
Fig. 7.19 from [PaHe98]

Fig. 4.21 (confusing, complex) (E.g., k=2, s = 29, d = 7, w = 5)

19/09/2001 Copyright Teemu Kerola 2001 25



Two definitions for ”Set” in ”Set Associative Mapping” (2)



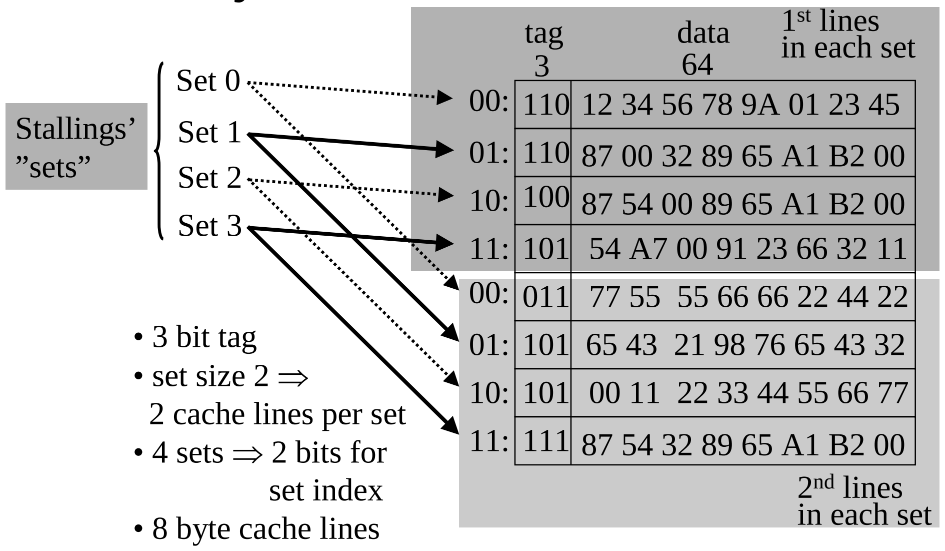
- Stallings ”set”
  - ”set” has <u>all</u> possible locations for block
- Hennessy-Patterson ”set”
  - 4 sets, and each ”set” has <u>one</u> possible location (index i) for block

19/09/2001 Copyright Teemu Kerola 2001 26

Two definitions for ”Set” in ”Set Associative Mapping”

- Term ”set” is the set of all possible locations where referenced memory block can be
  - Field ”set” of memory address determines this set
  - [Stal99]
- Cache memory is split into multiple ”sets”, and the referenced memory block can be in only one location in each ”set”
  - Field ”index” of memory address determines possible location of referenced block in each ”set”
  - [HePa96], [PaHe98]

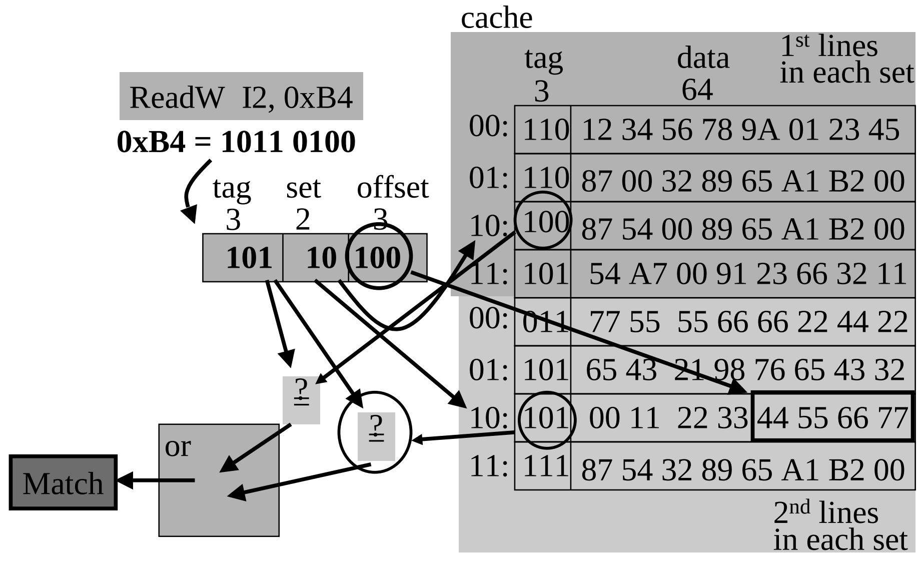
2-way Set Associative Cache



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

- 3 bit tag
- set size 2 ⇒ 2 cache lines per set
- 4 sets ⇒ 2 bits for set index
- 8 byte cache lines ⇒ 3 bits for byte address in cache line

Set Associative Example (6)



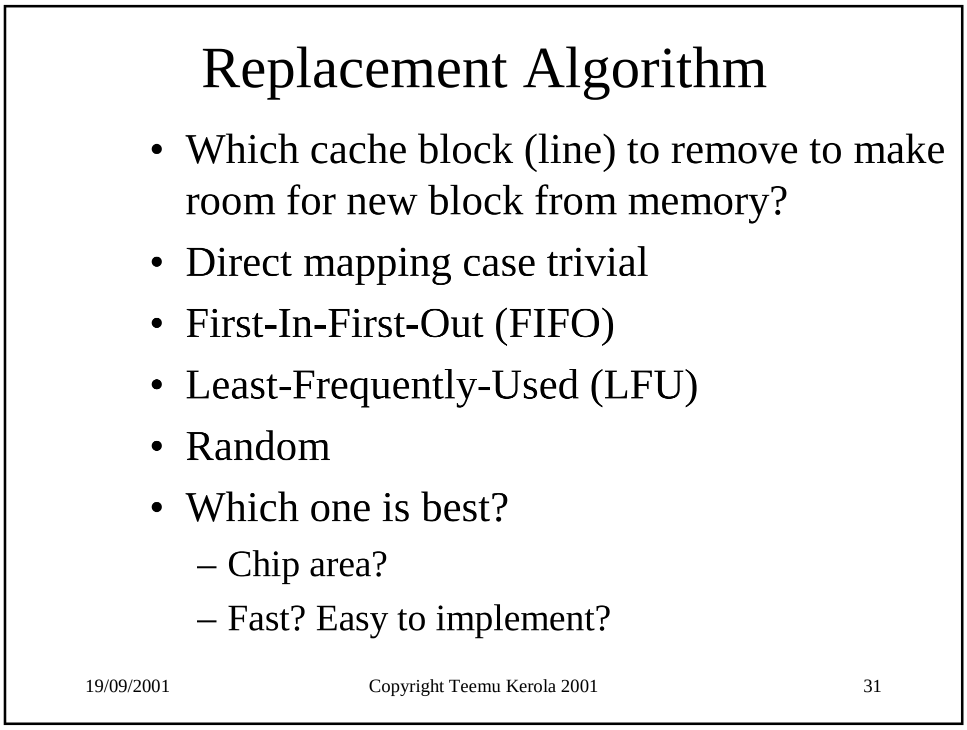
	tag 3	data 64
00:	110	12 34 56 78 9A 01 23 45
01:	110	87 00 32 89 65 A1 B2 00
10:	100	87 54 00 89 65 A1 B2 00
11:	101	54 A7 00 91 23 66 32 11
00:	011	77 55 55 66 66 22 44 22
01:	101	65 43 21 98 76 65 43 32
10:	101	00 11 22 33 44 55 66 77
11:	111	87 54 32 89 65 A1 B2 00

19/09/2001 Copyright Teemu Kerola 2001 29

Set Associative Mapping

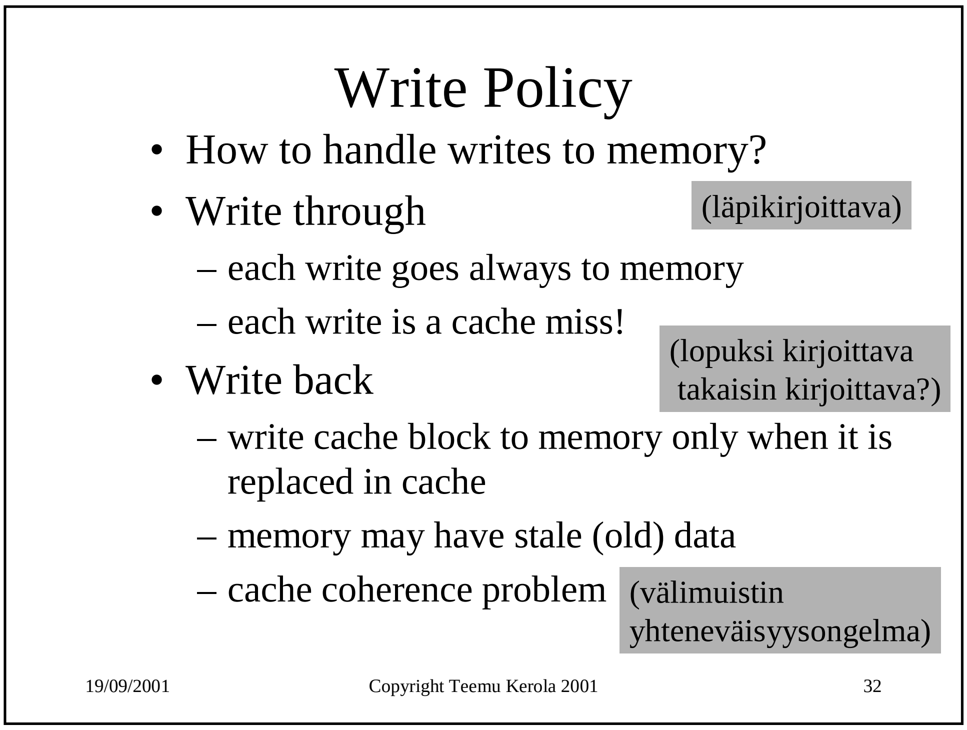
- Set associative cache with set size 2
  = 2-way cache
- Degree of associativity v? Usually 2
  - v large? Fig. 7.16 from [PaHe98]
    - More data items (v) in one set
    - less "collisions"
    - final comparison (matching tags?) gate delay?
  - v maximum (nr of cache lines)
    ⇒ fully associative mapping
  - v minimum (1) ⇒ direct mapping

19/09/2001 Copyright Teemu Kerola 2001 30



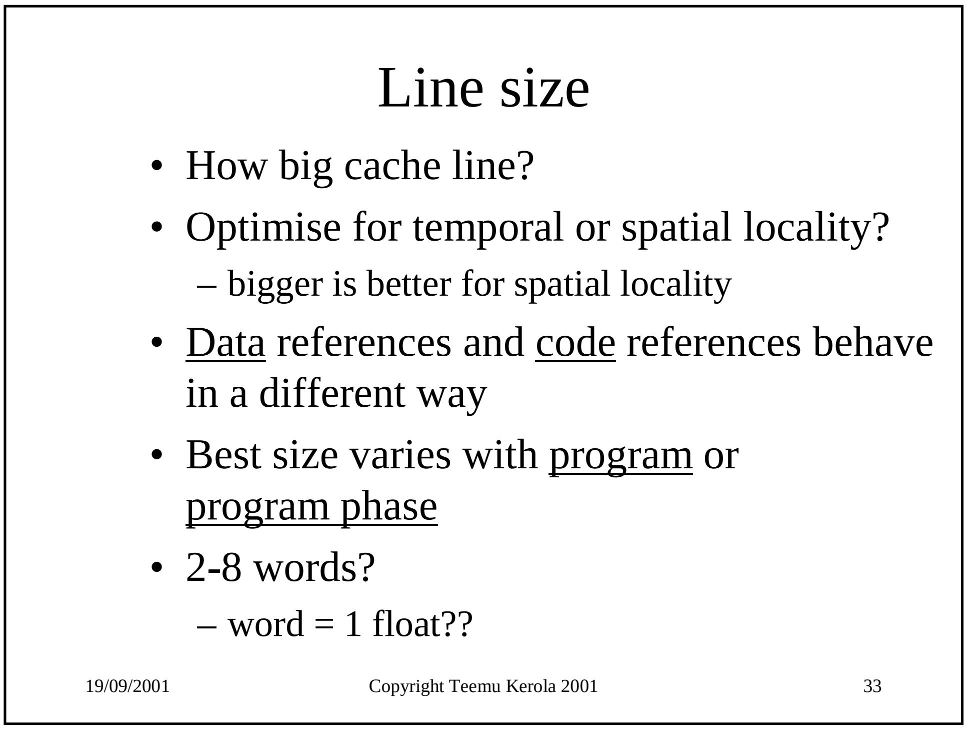
Replacement Algorithm

- Which cache block (line) to remove to make room for new block from memory?
- Direct mapping case trivial
- First-In-First-Out (FIFO)
- Least-Frequently-Used (LFU)
- Random
- Which one is best?
  - Chip area?
  - Fast? Easy to implement?



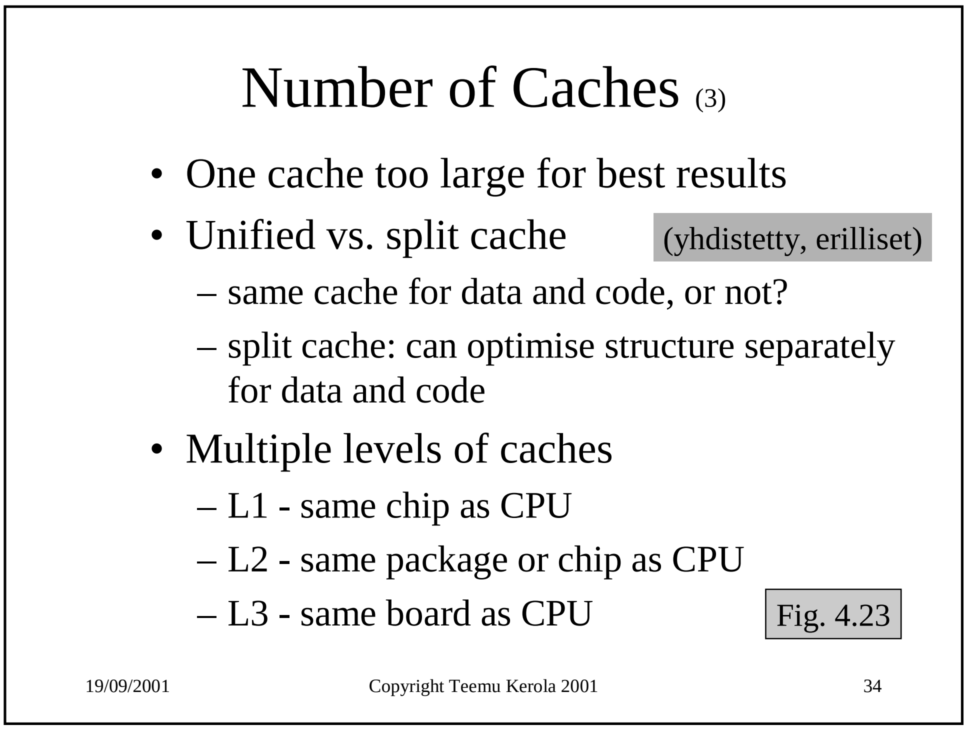
Write Policy

- How to handle writes to memory?
- Write through (läpikirjoittava)
  - each write goes always to memory
  - each write is a cache miss!
- Write back (lopuksi kirjoittava takaisin kirjoittava?)
  - write cache block to memory only when it is replaced in cache
  - memory may have stale (old) data
  - cache coherence problem (välimuistin yhteneväisyysongelma)



Line size

- How big cache line?
- Optimise for temporal or spatial locality?
  - bigger is better for spatial locality
- Data references and code references behave in a different way
- Best size varies with program or program phase
- 2-8 words?
  - word = 1 float??

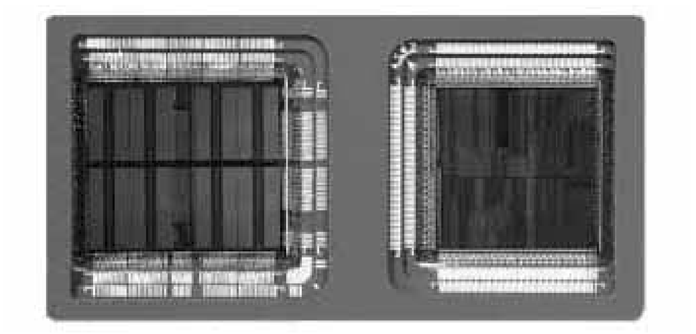


Number of Caches (3)

- One cache too large for best results
- Unified vs. split cache (yhdistetty, erilliset)
  - same cache for data and code, or not?
  - split cache: can optimise structure separately for data and code
- Multiple levels of caches
  - L1 - same chip as CPU
  - L2 - same package or chip as CPU
  - L3 - same board as CPU

Fig. 4.23

-- End of Ch. 4.3: Cache Memory --



http://www.intel.com/procs/servers/feature/cache/unique.htm

“The Pentium® Pro processor's unique multi-cavity chip package brings L2 cache memory closer to the CPU, delivering higher performance for business-critical computing needs.“