Computer Electronics

Lecture 24: Cache

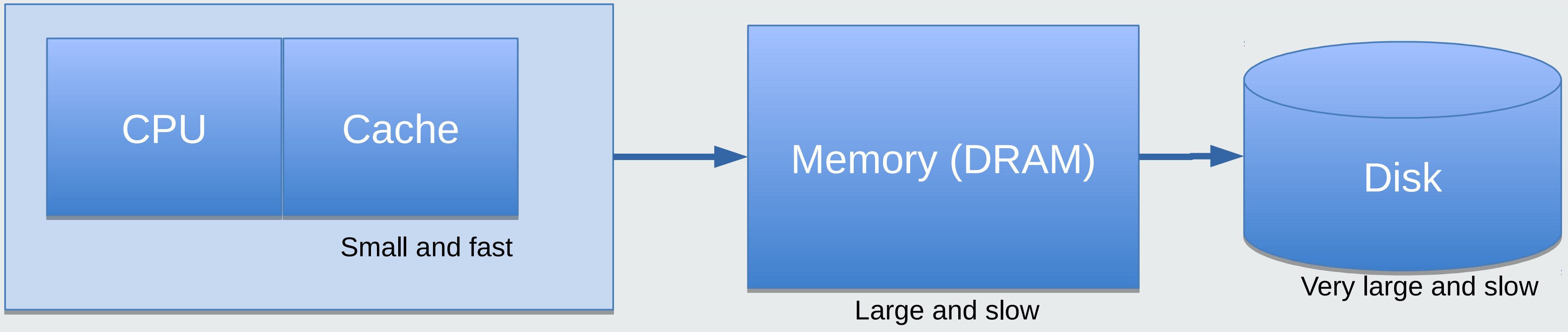
Outline

- Motivation
- The cache concept
- Memory address structure for using cache
- Directly mapped cache
- Fully associative cache
- K-way associative cache
- Replacement policy
- Write policies

Motivation

- Accessing main memory (off-chip DRAM) is slow due to latency of memory controller and memory itself
- The cache is a smaller SRAM on-chip memory that can be accessed in 1 clock cycle preferably
- Typical cache sizes are in the order of tens of kB
- Instructions or data word (32 or 64 bits) accessed using a parallel bus
- Look at iob-cache

The Cache Concept



Cache may have multiple levels successively slower but larger: L1, L2, L3… 2 or 3 levels common

Principle of Locality

- In space: data in near addresses are more likely to be accessed
- In time: data recently accessed is more likely to be accessed again

<u>Definitions</u>

p: cache hit rate
1-p: cache miss rate
Tc: cache access time
Tm: memory access time (miss penalty)
Ta: average access time

$$\boxed{Ta = pTc+(1-p)Tm}$$

Memory address structure for using cache

Main memory is divided into N address sets

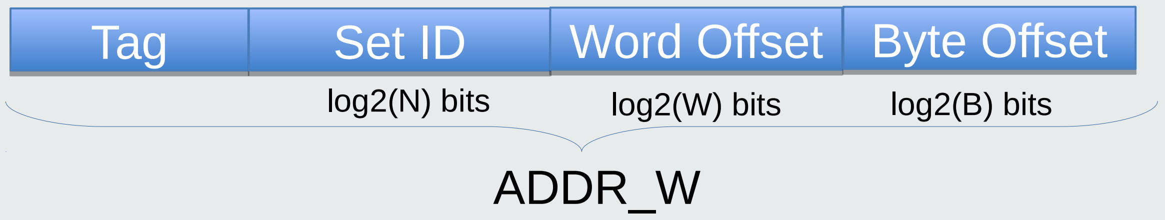
Each set element stores Y ways of L data lines

Each data line has W words

Each word has B bytes

Byte offset : select byte in word

Structure of a main memory address (char *addr)



Tag bits = ADDR_W-log2(N*W*B) bits

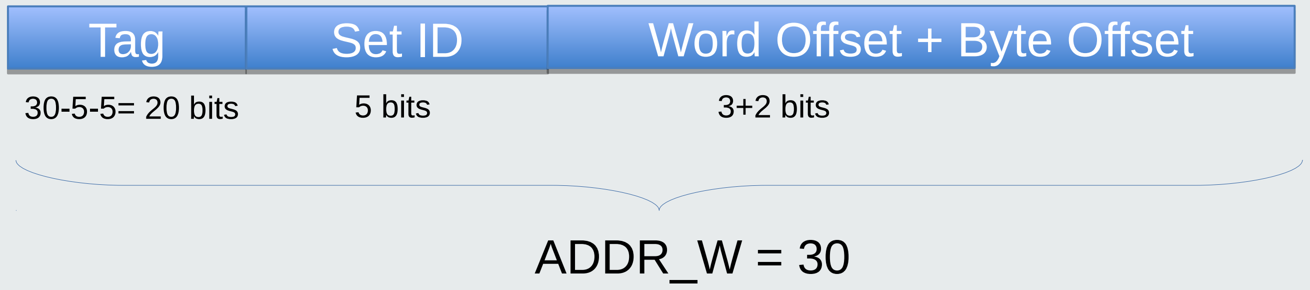
Memory address example

1GB memory: 32 sets, 4 ways, 4B words, 8 data words/line

1GB → 30 address bits (log2(2^{30}))

32 sets → 5 set ID bits (log2(32))

Cache size = 32 sets * 4 ways * 8 words/line * 4 bytes/word = exp2(5+2+3+2) = 4kB



Number of bits in the address

- #byte offset bits = log2(#bytes per word)
- #word offset bits = log2(#words per line)
- #set index bits = log2(#sets)
- #tag bits = #address bits – #byte offset bits - #word offset bits - #set index bits

Set visualization

1GB memory, 4 sets, 4 byte data words

1GB → 30 address bits ($\log2(2^{30})$)

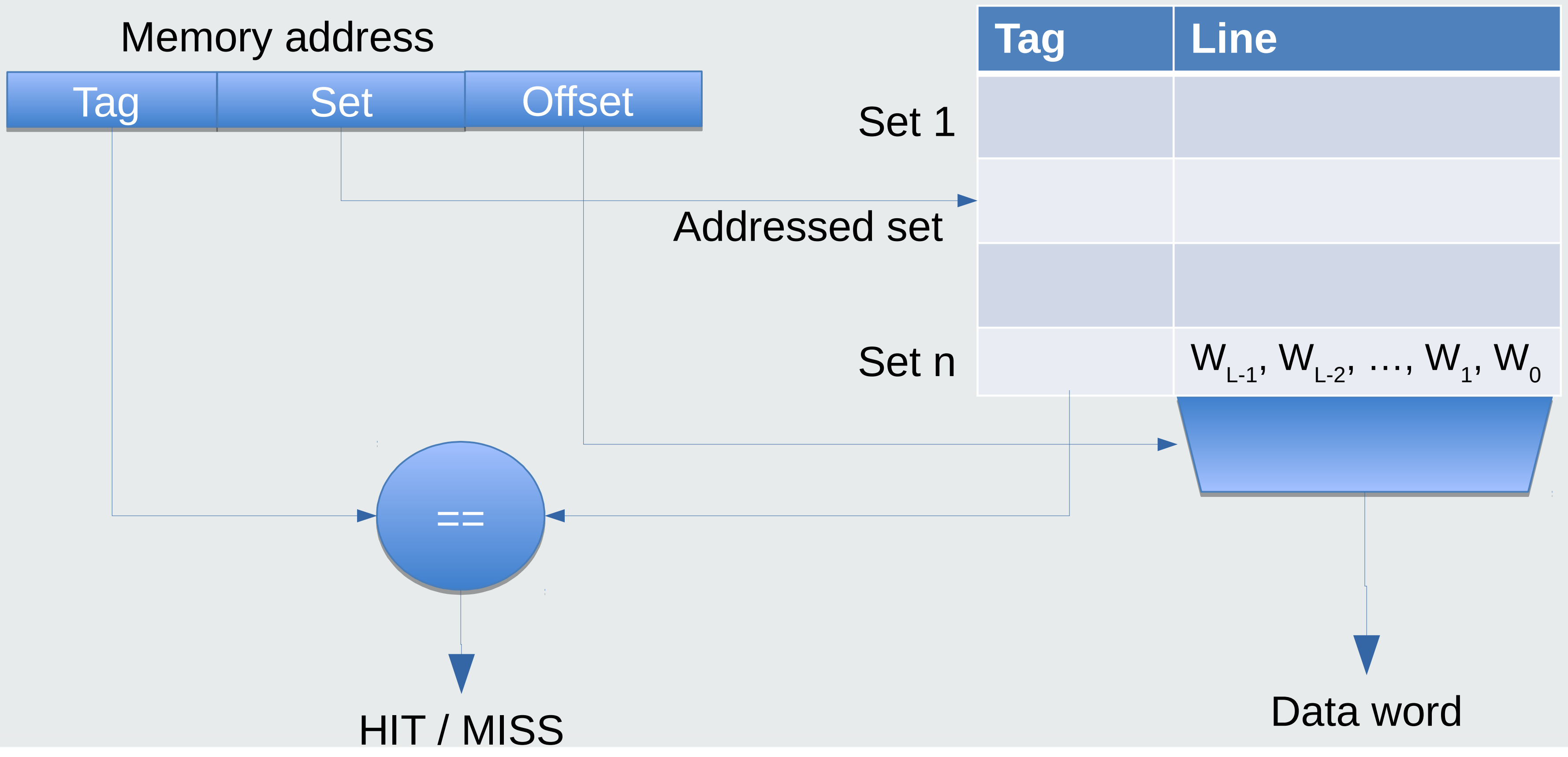
4 sets → 2 set ID bits (log2(4))

Main memory (each color is a set)
addresses 0:3 (set 0, tag 0)
addresses 7:4 (set 1, tag 0)
addresses 11:8 (set 2, tag 0)
addresses 15:12 (set 3, tag 0)
addresses 19:16 (set 0, tag 1)
addresses 23:20 (set 1, tag 1)
addresses 27:24 (set 2, tag 1)
addresses 31:28 (set 3, tag 1)
Etc.

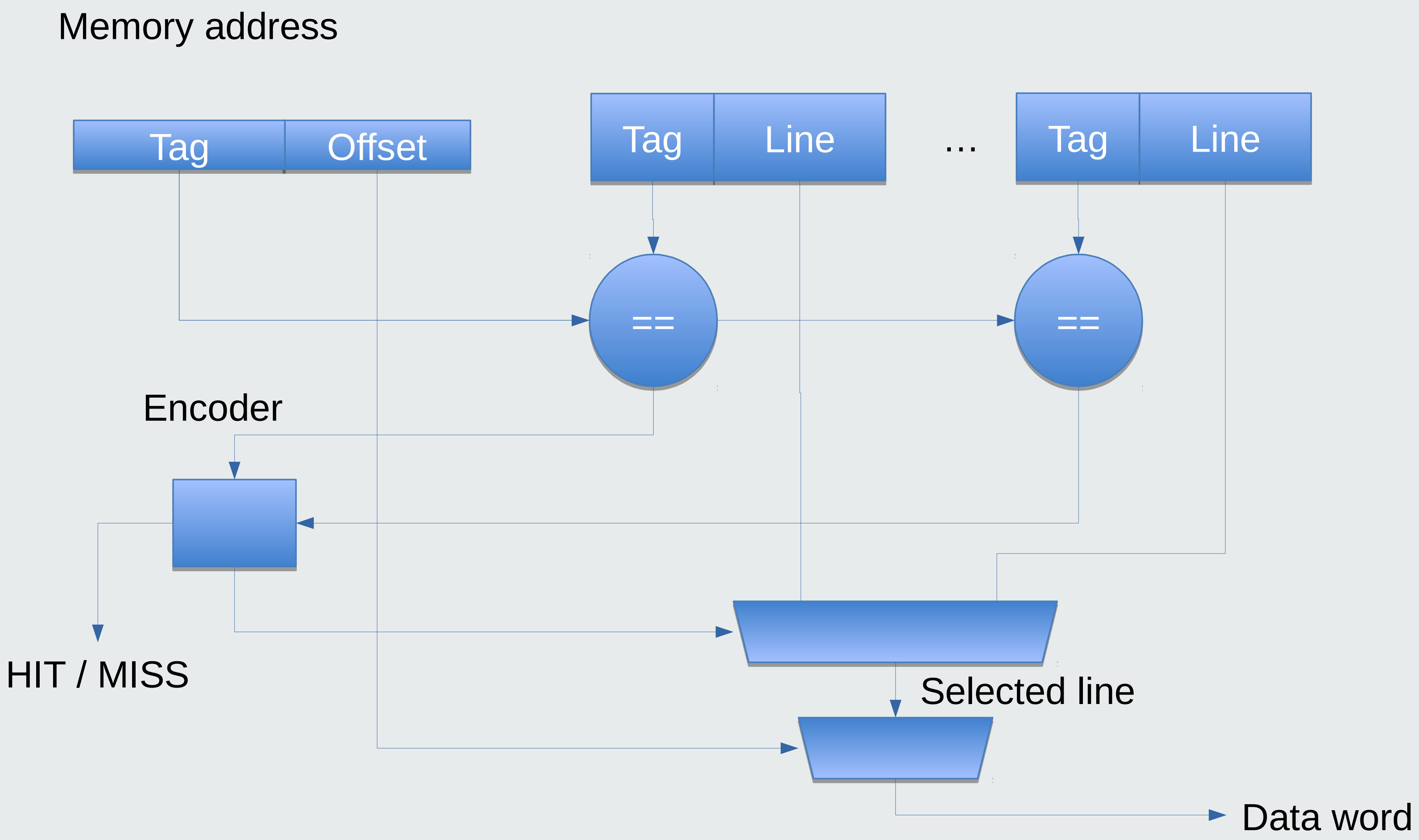
Directly Mapped Cache

1-way cache

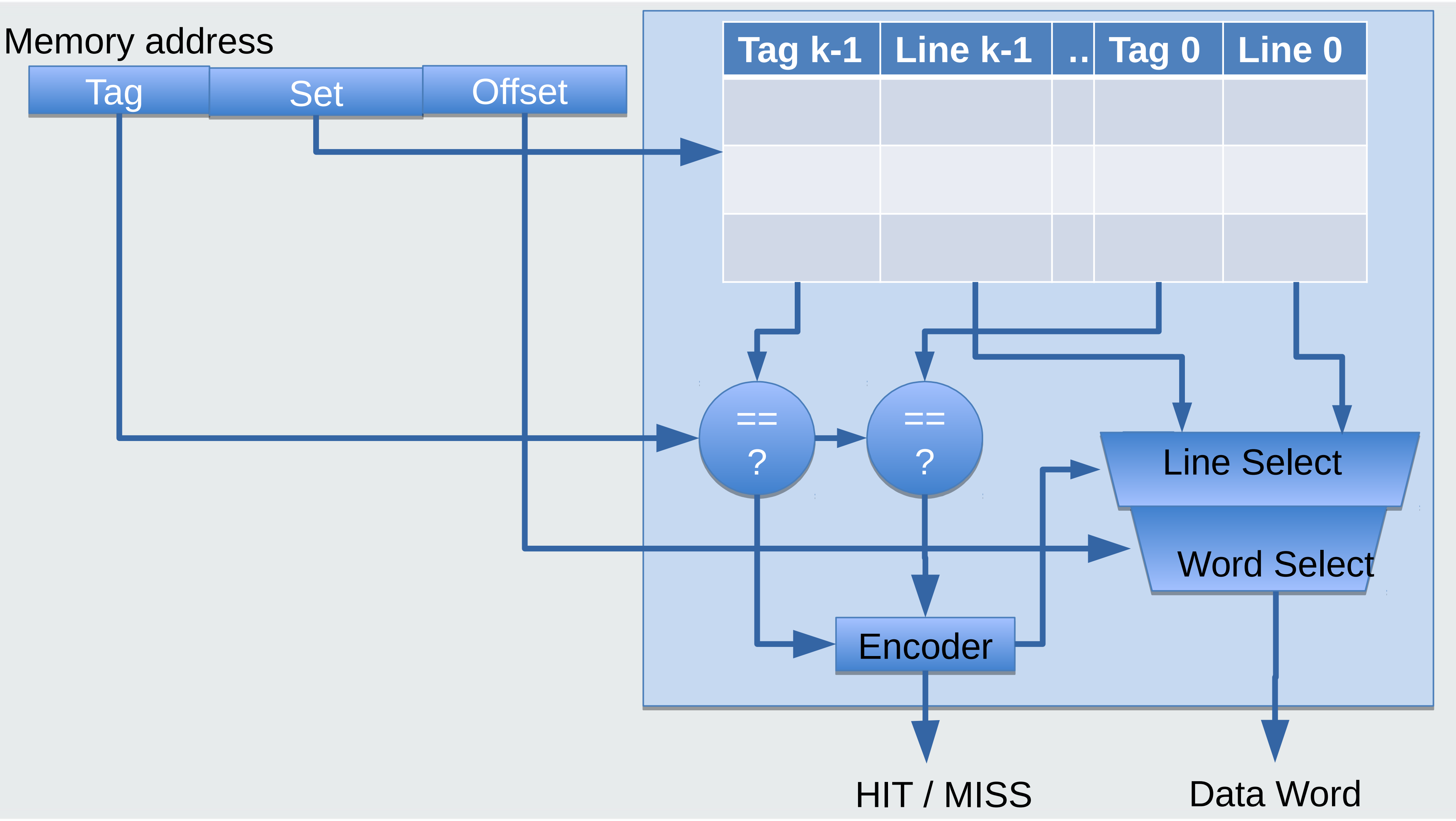
Each set only has 1 line in cache
Each line has L data words



Fully Associative Cache (single set, multiple lines)



K-Way Set Associative Cache (each set has k lines in cache)



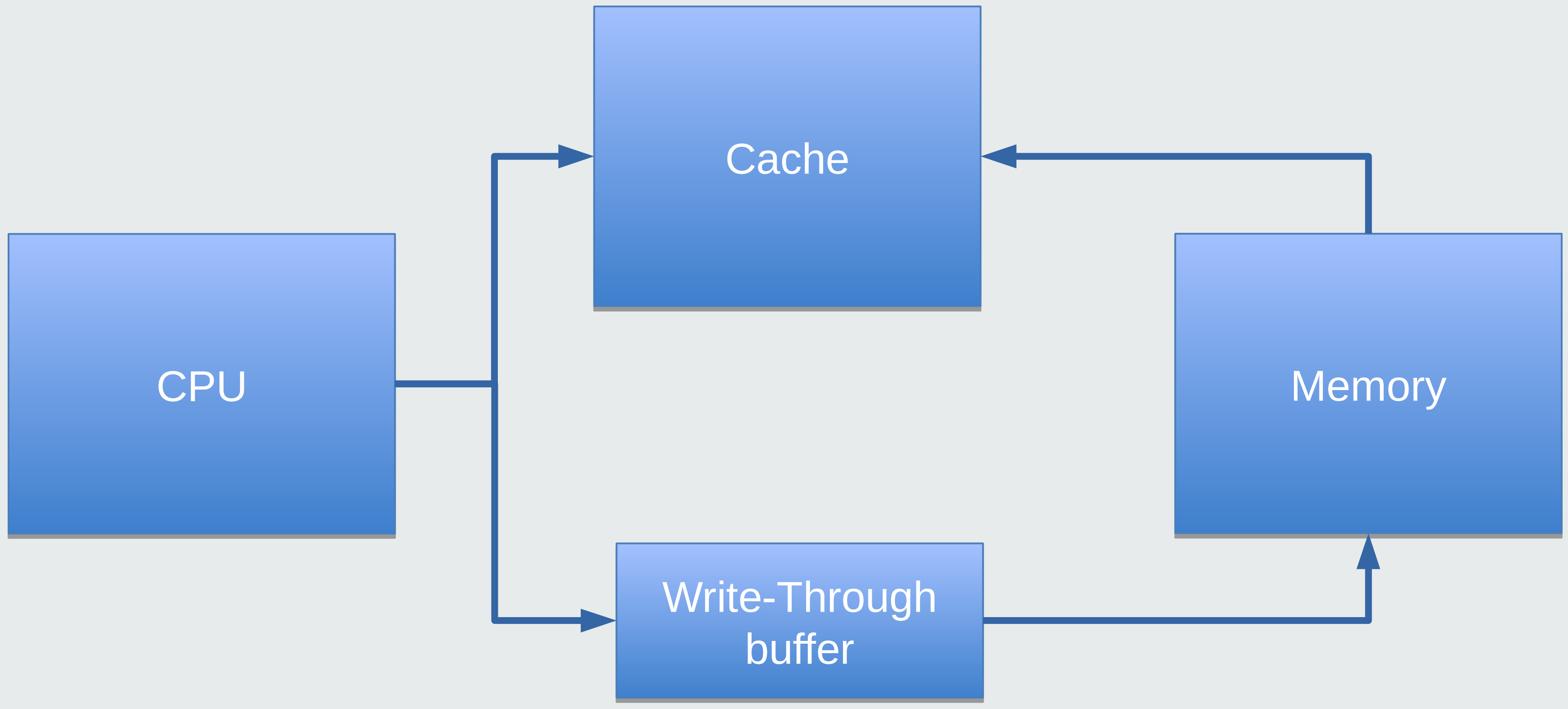
Cache line replacement policy

- Upon a cache miss, line must be replaced
- Directly mapped cache (aka 1-way cache)
  - Only one line to choose – no choice
- K-way cache replacement policy
  - Random replace (RR): needs random counter – easy
  - Least recently used (LRU): needs usage record – difficult
  - Pseudo LRU: needs 1 bit for the last used, replace adjacent – easy
  - Most recently used (MRU): needs 1 bit for the last used – easy
  - Least Frequently Used: needs frequency counter – difficult
  - …. 17 policies listed in Wikipedia…

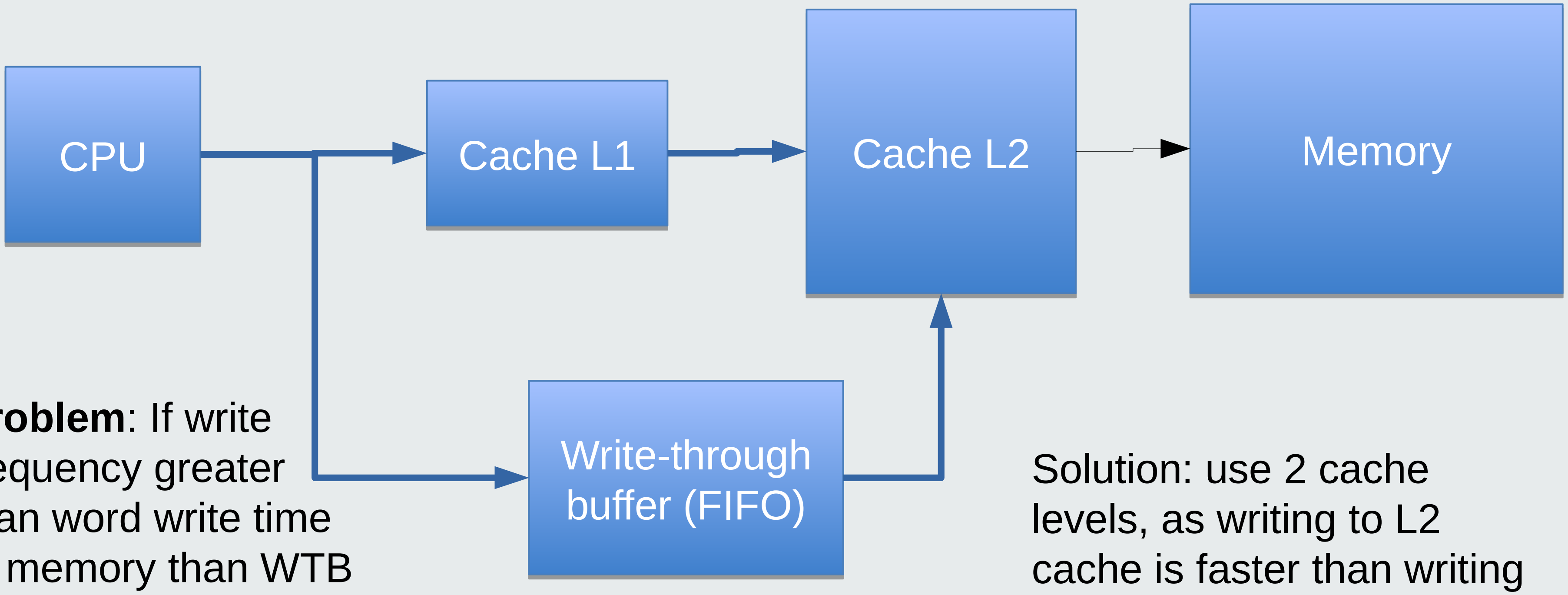
Cache write policy

- Write-through
  - data word (not block) is written to cache and main memory in case of a hit (miss case analyzed separately)
- Write-back
  - Data is written to cache only in case of a hit (miss case analyzed separately)
  - Data is written to memory only when line is replaced
  - Performance improvement with Dirty bit
    - Dirty bit is set when line is modified
    - Upon replacement, data is saved in memory only if Dirty bit is set

Write through hardware



Write-through buffer overflow



Problem: If write frequency greater than word write time to memory than WTB becomes full

Solution: use 2 cache levels, as writing to L2 cache is faster than writing to memory

Cache write miss policy

- What to do when line to write is not in cache
- Write Allocate policy
  - Transfer line to cache (allocate), then write to it
- Write Not-Allocate
  - Write to main memory only
- Usual policy combination
  - Write-Through + Write Not-Allocate
  - Write-back + Write Allocate