Computer Organization and Architecture

8 Cache

Tongwei Ren

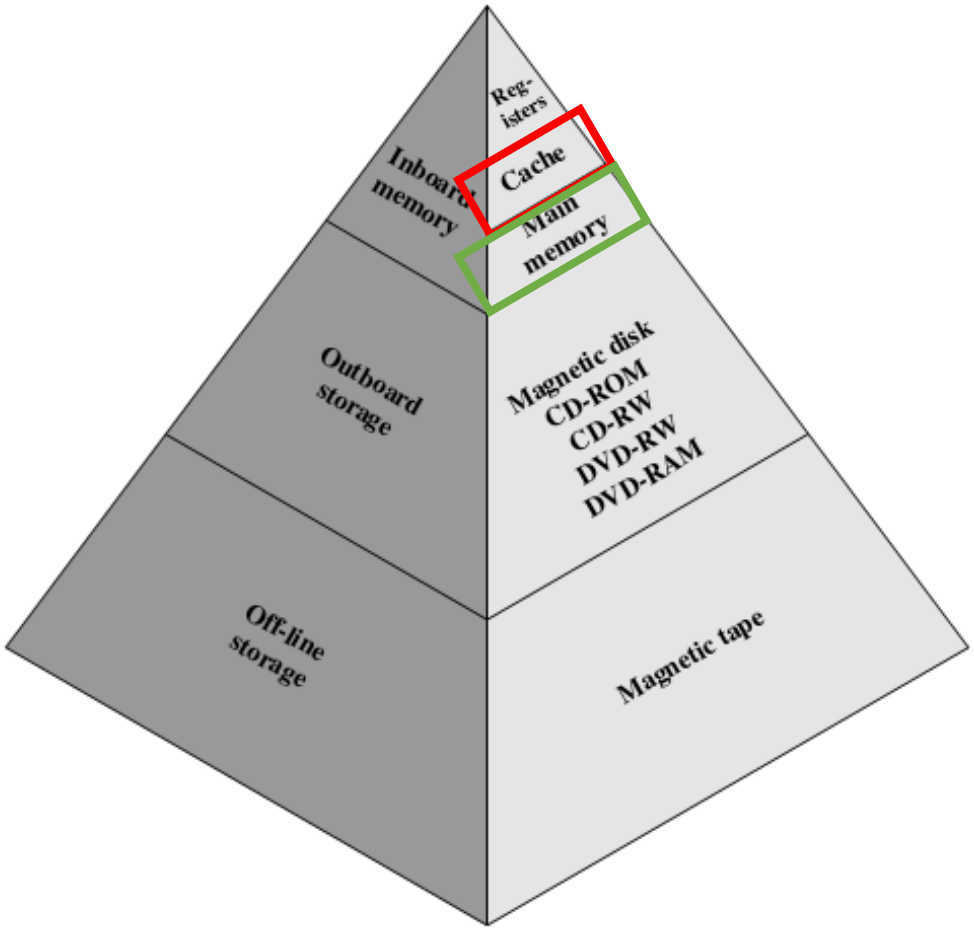
Oct. 10, 2019

南京大學
NANJING UNIVERSITY

Review

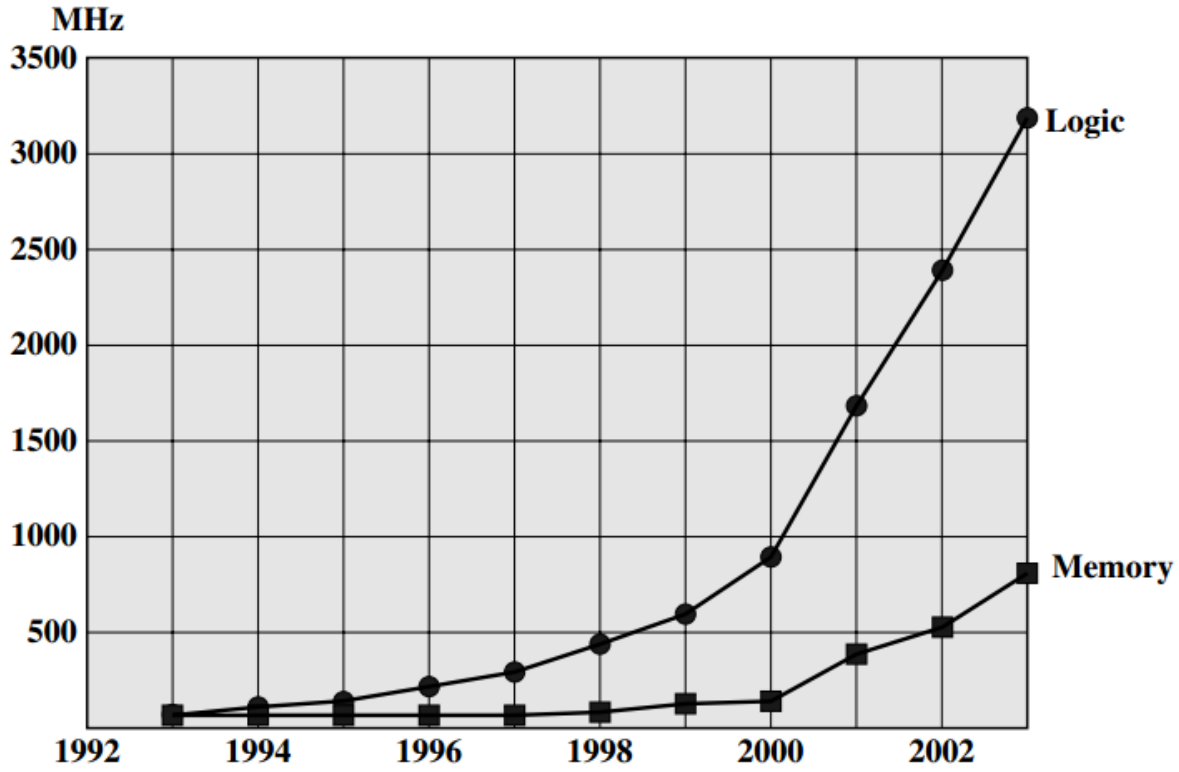
- Semiconductor memory
  - RAM: DRAM, SRAM, SDRAM, DDR
  - ROM, PROM
  - EPROM, EEPROM, flash memory
- From cell to main memory
  - Addressable unit, memory array, chip, module, main memory

Memory Hierarchy



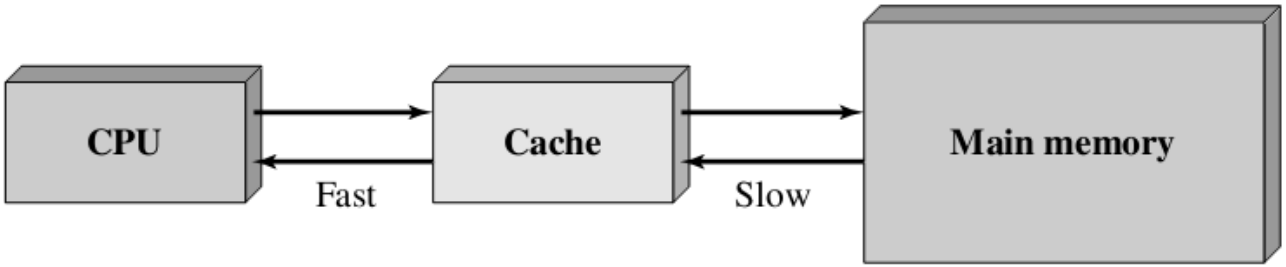
Why cache?

- The speed of CPU is faster than memory's



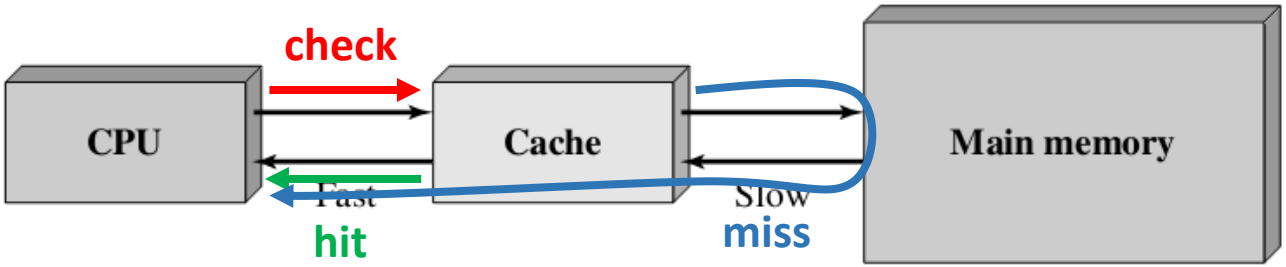
Basic Idea

- Use a smaller, faster cache memory block together with a relatively large and slow main block
- The cache contains a copy of portions of main memory
- Located between CPU and memory, and may be integrated inside CPU or as a module on motherboard



How Cache Works

- **Check**: when the processor attempts to **read a word** of memory, a check is made to determine if the word is in the cache
- **Hit**: if so, the word is delivered to the processor
- **Miss**: if not, **a block** of main memory, consisting of some fixed number of words, is **read into the cache** and then the word is **delivered to the processor**



Some Questions

- How to determine hit and miss?
- Why not directly move the word from memory to CPU if miss?
- Why read a block but not a word from memory if miss?
- Why cache can save time since more operations are required?

Hit and Miss Determination

- The von Neumann machine design
  - The contents of this memory are addressable by **location**, without regard to the type of data contained there
- Cache includes **tags** to identify its content's corresponding **locations** in main memory

Some Questions (cont.)

- How to determine hit and miss?
- Why not directly move the word from memory to CPU if miss?
- Why read a block but not a word from memory if miss?
- Why cache can save time since more operations are required?

Principle of Locality

- Also known as: locality of reference
- a phenomenon describing the same value, or related storage locations, being frequently accessed (Wikipedia)
- Types
  - **Temporal locality**: the reuse of specific data, and/or resources, within a relatively small time duration
  - **Spatial locality**: the use of data elements within relatively close storage locations
    - **Sequential locality**: a special case of spatial locality, occurs when data elements are arranged and accessed linearly, such as, traversing the elements in a one-dimensional array

Principle of Locality (Example)

- Temporal locality

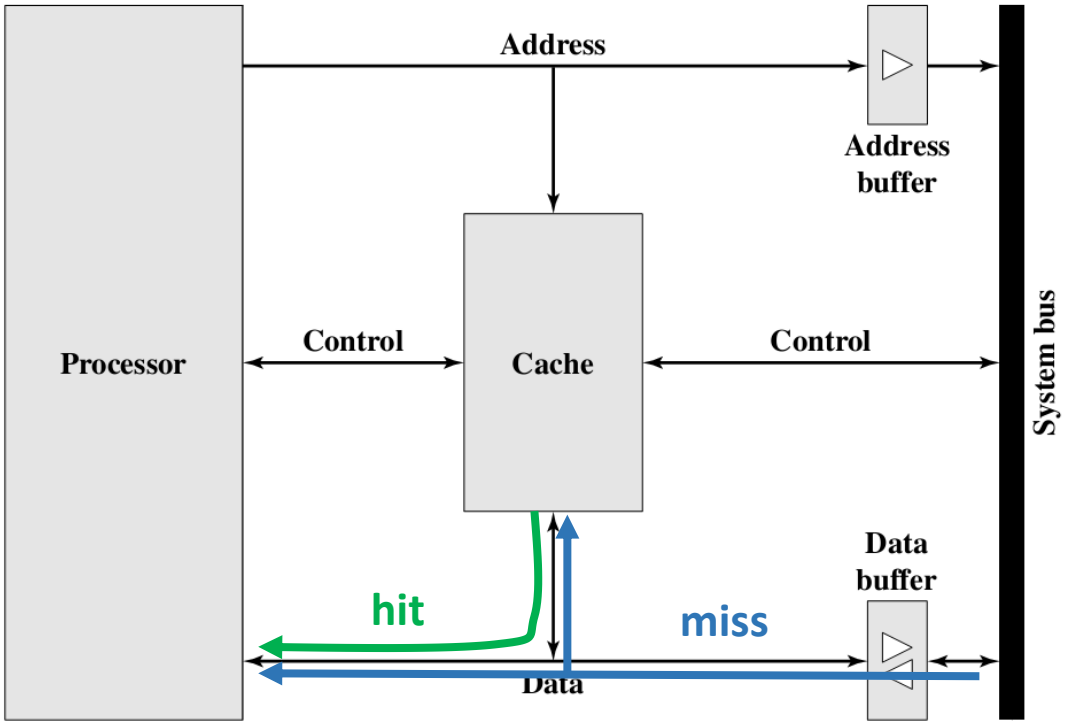
```
int factorial = 1;
for (int i = 2; i <= n; i++) {
    factorial = factorial * i;
}
```

- Spatial locality

```
for (int i = 0; i < num; i++) {
    score[i] = final[i] * 0.4 + midterm[i] * 0.3 + assign[i] * 0.2 + activity[i] * 0.1;
}
```

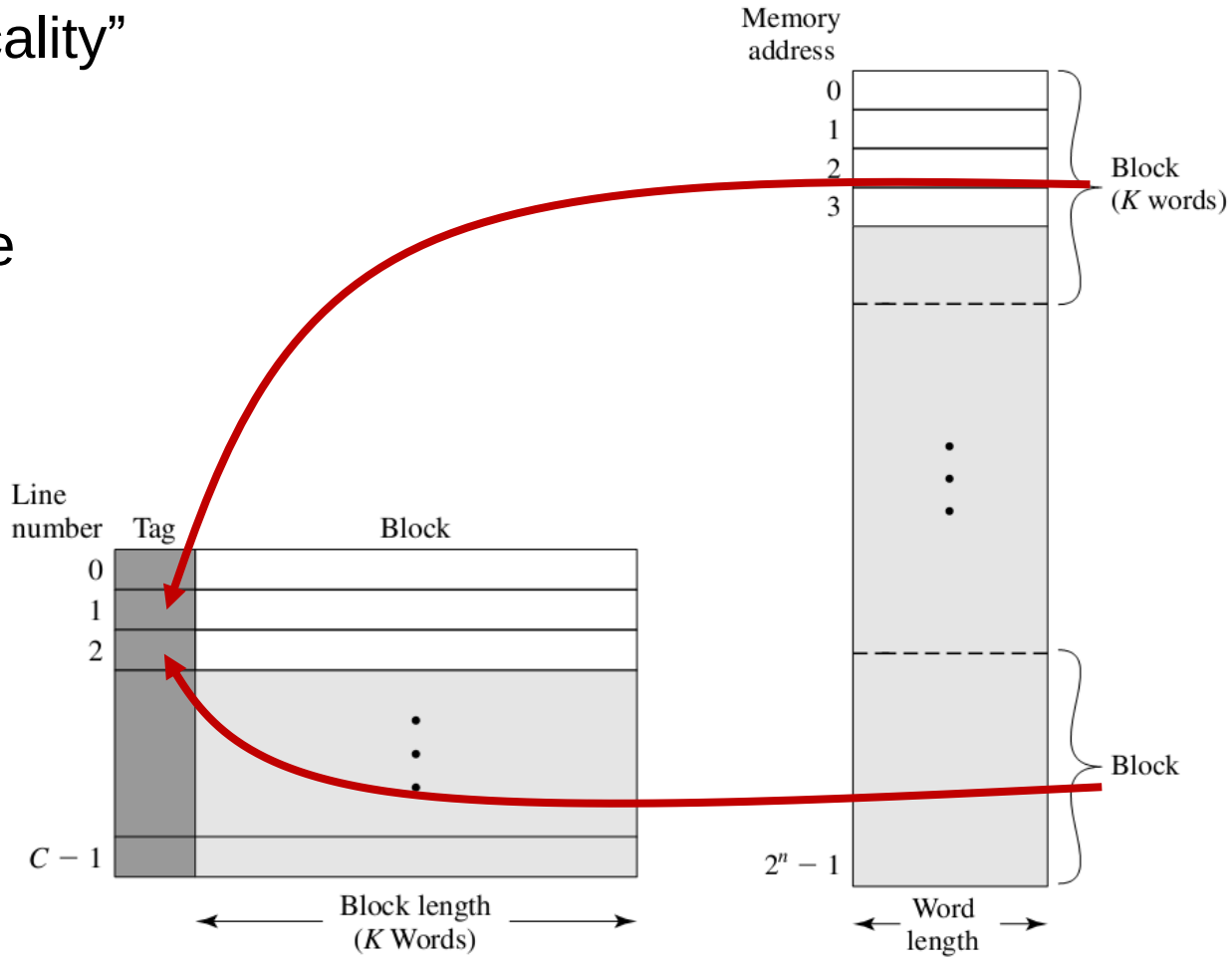
Move Content to Cache

- Use “temporal locality”
- Typical cache organization



Move Block Instead of Word

- Use “spatial locality”
- Cache structure



Some Questions (cont.)

- How to determine hit and miss?
- Why not directly move the word from memory to CPU if miss?
- Why read a block but not a word from memory if miss?
- Why cache can save time since more operations are required?

Average Access Time

- Assume p is **hit rate**, T_C is access time of cache, T_M is access time of main memory, the average access time when using cache is

$$T_A = p \times T_C + (1 - p) \times (T_C + T_M)$$

$$= T_C + (1 - p) \times T_M$$

- The larger p and smaller T_C are, the better performance is
- If we want $T_A < T_M$, it is required

$$p > T_C \,/\, T_M$$

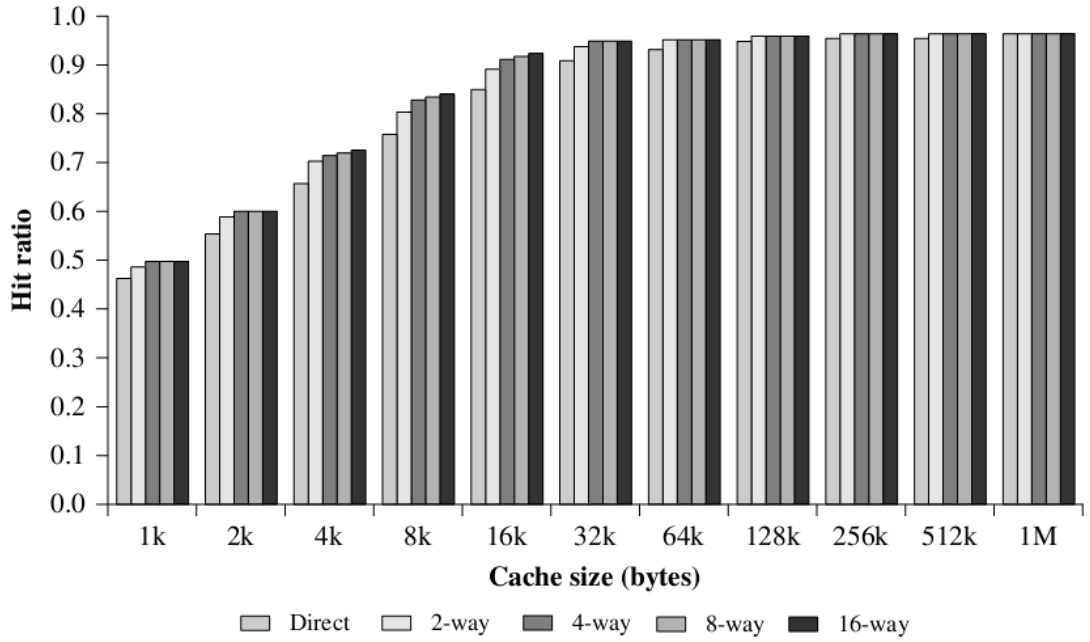
- Difficulty: the capacity of cache is much smaller than the capacity of memory

Elements of Cache Design

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Cache Size

- Increasing cache size will:
  - Increases hit rate p
  - Increases cost and access time of cache T_C



Cache Size (cont.)

- Cache size of some processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128 to 256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256 to 512 KB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 kB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 kB/32 kB	4 MB	—
Itanium 2	PC/server	2002	32 kB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB

Elements of Cache Design (cont.)

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Mapping Function

- An algorithm used to map main memory blocks into cache lines
- A method is needed for determining which main memory block currently occupies a cache line
- The choice of the mapping function dictates how the cache is organized
- Types
  - Direct mapping
  - Associative mapping
  - Set associative mapping

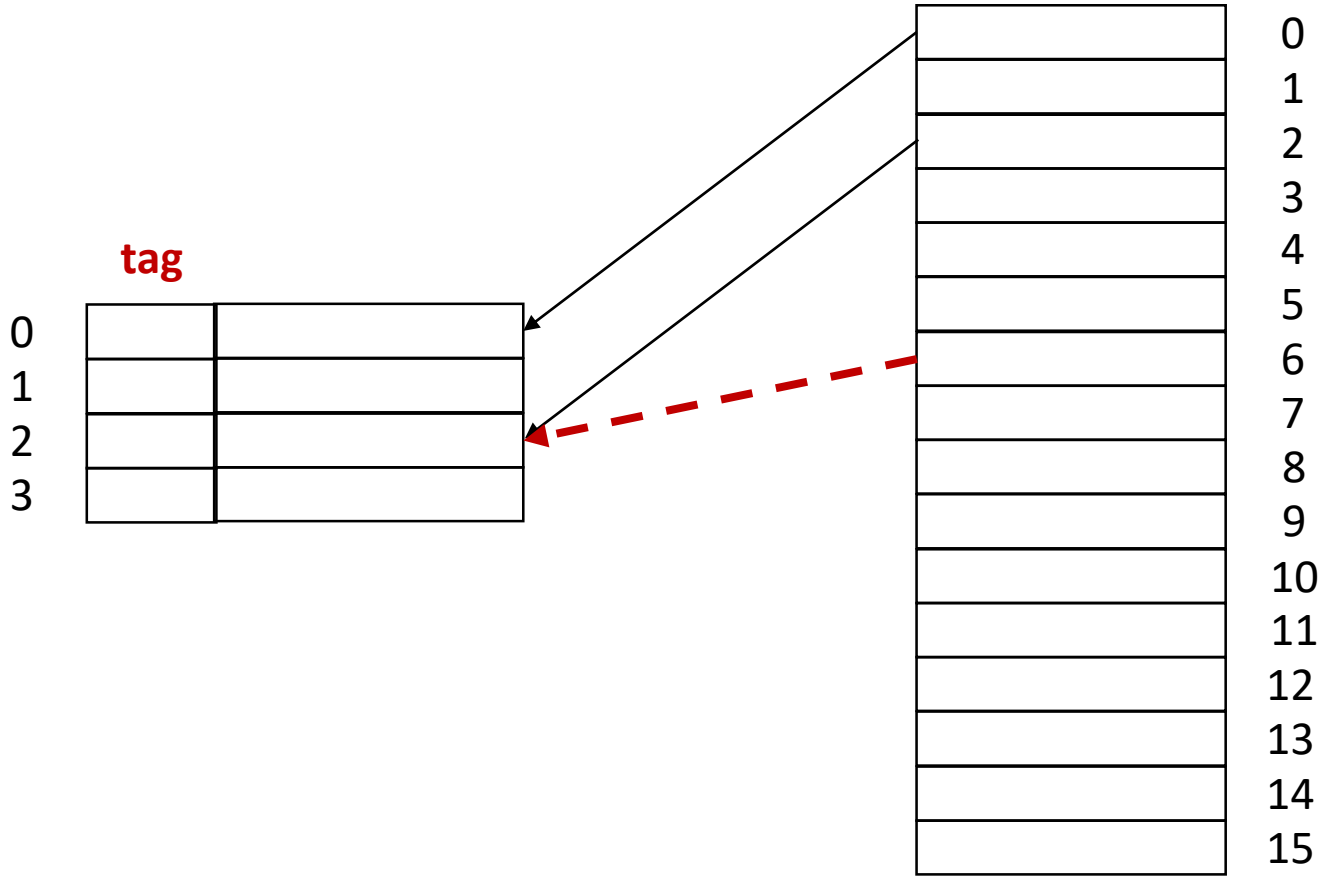
Direct Mapping

- Map each block of main memory into **only one possible** cache line
- Assume i is cache line number, j is main memory block number, C is number of lines in cache

$$i = j \ mod \ C$$

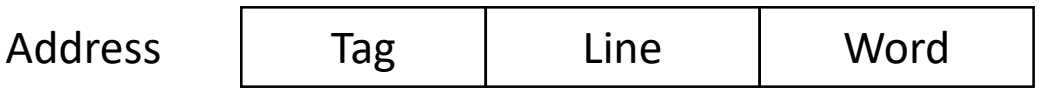
Direct Mapping (cont.)

- Example



Direct Mapping (cont.)

- Tag
  - Highest n bits in address, $n = log_2 M - log_2 C$



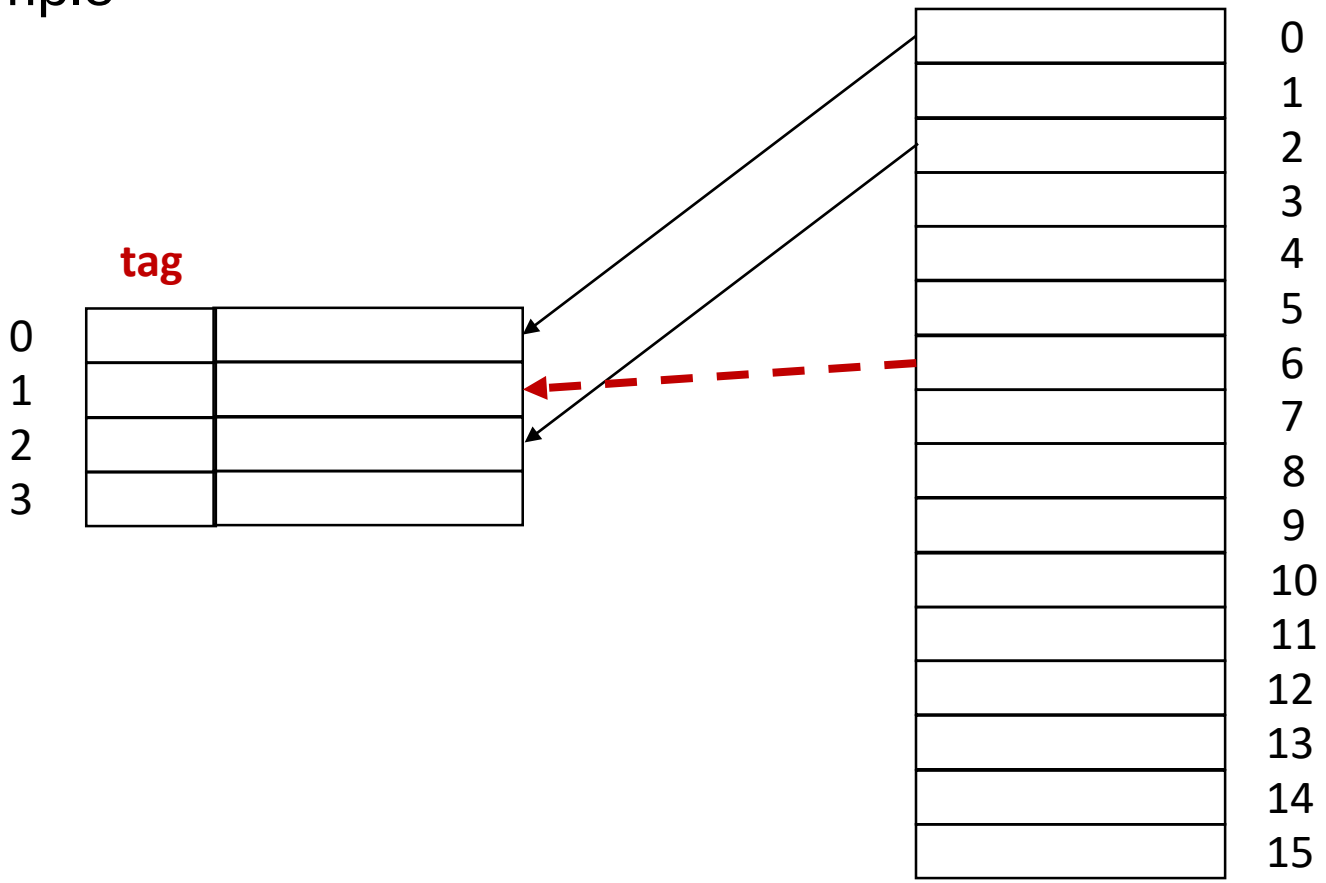
- Example
  - Assume cache has 4 lines, each line contains 8 words, and main memory contains 128 words. To access main memory, the length of address is 7 bits. The lowest 3 bits determines which word in the block, the middle 2 bits determines which line is possible, and the highest 2 bits determines which block occupies the cache

Direct Mapping (cont.)

- Advantages
  - Simple
  - Fast mapping
  - Fast checking
- Disadvantages
  - Thrashing: if a program accesses two blocks that map to the same line repeatedly, cache miss rate will be very high
- Good for cache with large capacities

Associative Mapping

- A memory block can load into **any line** of cache
- Example



Associative Mapping (cont.)

- Tag
  - Highest n bits in address, $n = log_2 M$

- Example
  - Assume cache has 4 lines, each line contains 8 words, and main memory contains 128 words. To access main memory, the length of address is 7 bits. The lowest 3 bits determines which word in the block, and the highest 4 bits determines which block occupies the cache

Associative Mapping (cont.)

- Advantages
  - Avoid thrashing
- Disadvantages
  - Complicated implementation
  - Cache search is expensive, i.e. require to access each line of cache in checking
- Good for cache with smaller capacities

Set Associative Mapping

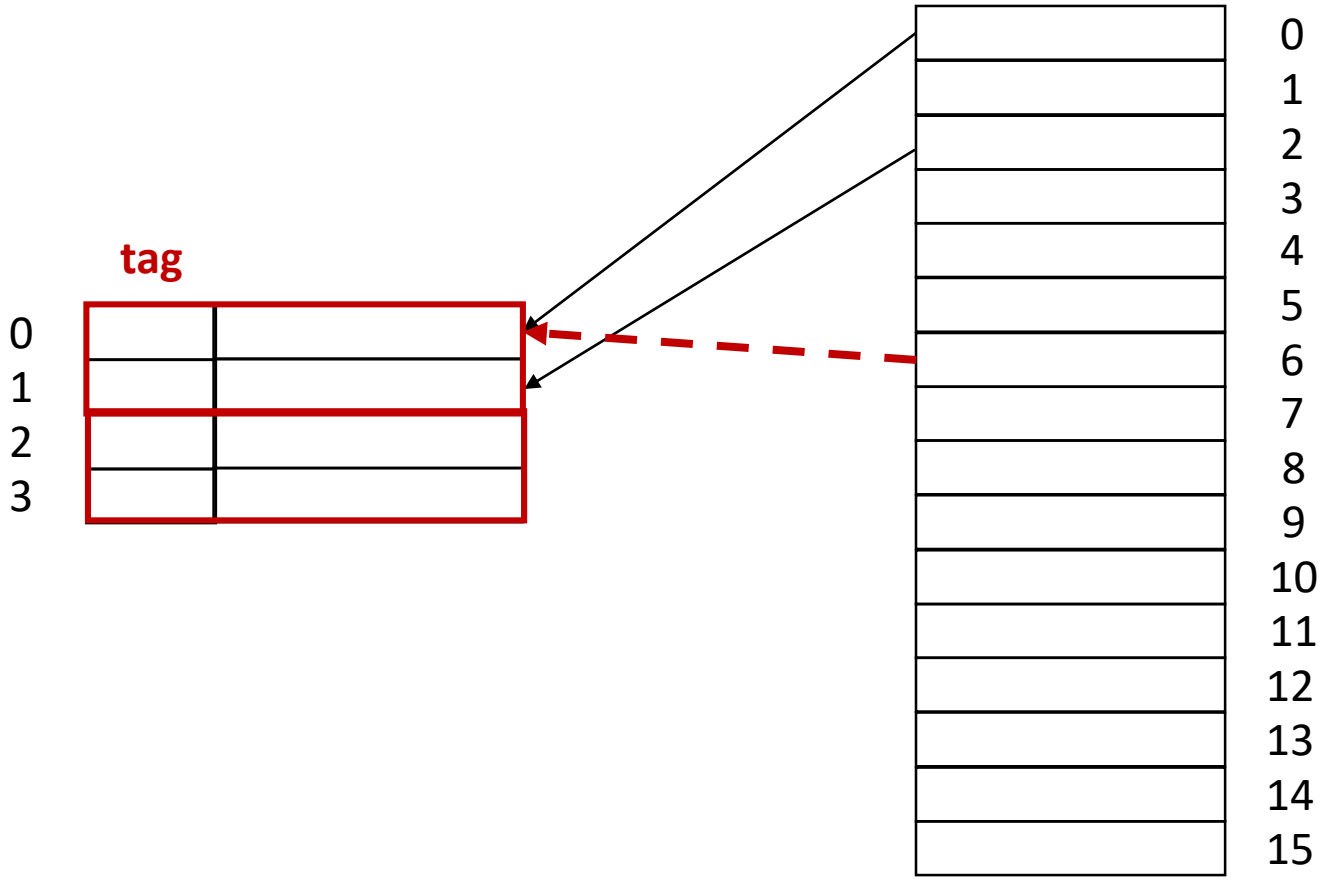
- Cache is divided into a number of sets, each set contains a number of lines, and a given block maps to **any line in a given set**
- Assume s is set number, j is main memory block number, S is number of sets in cache

$$s = j \ mod \ S$$

- K-way set $K = C/S$

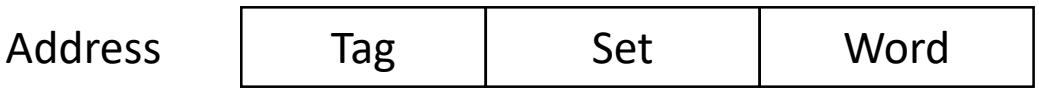
Set Associative Mapping (cont.)

- Example



Set Associative Mapping (cont.)

- Tag
  - Highest n bits in address, $n = log_2 M - log_2 S$



- Example
  - Assume cache has 4 lines which are divided into 2-way set, each line contains 8 words, and main memory contains 128 words. To access main memory, the length of address is 7 bits. The lowest 3 bits determines which word in the block, the middle 1 bit determines which set in the cache, and the highest 3 bits determines which block occupies the cache

Set Associative Mapping (cont.)

- Advantages
  - Combine the advantages of direct mapping and associate mapping
- Disadvantages
  - Combine the disadvantages of direct mapping and associate mapping
- Make a good trade-off for arbitrary capacity cache

Comparison of Mapping Functions

- Relationship
  - If $K = 1$, set associate mapping is same to direct mapping
  - If $K = C$, set associate mapping is same to associate mapping

Comparison of Mapping Functions (cont.)

- Correlation: number of possible cache lines for each block in main memory
  - Direct mapping: 1
  - Associative mapping: C
  - Set associative mapping: K

Comparison of Mapping Functions (cont.)

- The lower correlation is, the lower hit rate is
  - Direct mapping is the lowest in hit rate, and the associative mapping is the highest
- The lower correlation is, the quicker checking is
  - Direct mapping has the least check time, and associative has the most check time
- The lower correlation is, the shorter tag is
  - Direct mapping is the shortest tag, and the associative mapping is the longest tag

Elements of Cache Design (cont.)

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Replacement Algorithm

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping, there is only one possible line for any particular block, and no choice is possible
- For the associative and set-associative mappings, a replacement algorithm is needed, which must be implemented in hardware

Replacement Algorithm (cont.)

- Most common replacement algorithms
  - Least Recently Used (LRU)
  - First In First Out (FIFO)
  - Least Frequently Used (LFU)
  - Random

Least Recently Used (LRU)

- Replace a set of blocks in cache having the longest time of having no references to it
- Assumption: more recently used memory locations are more likely to be referenced
- Implementation for two-way set associative mapping
  - Each line includes a USE bit
  - When a line is referenced, its USE bit is set to 1 and the USE bit of the other line in that set is set to 0
  - When a block is to be read into the set, the line whose USE bit is 0 is used

First In First Out (FIFO)

- Replace the set of blocks in cache that has been in the cache for the longest time
- Assumption: later used memory locations are more likely to be referenced
- Implementation: round-robin or circular buffer technique

Least Frequently Used (LFU)

- Replace the set of blocks having the least number of references
- Assumption: more frequently used memory locations are more likely to be referenced
- Implementation: associate a counter with each line

Random

- Replace that block in cache or the set randomly
- Assumption: each memory location is similar to be referenced
- Implementation: randomly replace
- random replacement provides only slightly inferior performance to an algorithm based on usage

Elements of Cache Design (cont.)

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Write Policy

- Consistency between memory and cache
  - When a block in cache is being replaced, it should consider whether the block has been altered
- Two cases
  - If not altered, it may be overwritten with a new block without any other operation
  - If altered, the block in main memory must be updated before replacement
- Policy
  - Write through
  - Write back

Write Through

- All write operations are made to main memory as soon as cache is modified
- Advantage
  - Ensures that main memory is always up to date
- Disadvantages
  - Generates substantial memory traffic and may create a bottleneck
  - Slows down write operation

Write Back

- Updates are made only in the cache, and when a block is replaced, it is written back to main memory if and only if it is altered
- Use a dirty bit, or use bit, to represent whether a block is altered
- Advantage
  - Minimizes memory writes
- Disadvantages
  - Portions of main memory are outdated, and hence accesses by I/O modules can be allowed only through the cache, which makes for complex circuitry and a potential bottleneck

Elements of Cache Design (cont.)

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Line Size

- As the line size increases from very small to larger
  - Hit ratio increases
    - More useful data can be brought into the cache
- As the line becomes even larger
  - Hit ratio decreases
    - Larger blocks reduce the number of lines in a cache, which leads to frequent replacement of lines
    - Each additional word is farther from the requested word and therefore less likely to be needed in the near future
- The relationship between block size and hit ratio is complex

Elements of Cache Design (cont.)

- Cache size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Numbers of Caches

- Single or two level
  - Single
    - Integrates a cache on the same chip as the processor
    - Reduces external bus activity and speeds up execution
  - Two level
    - Reduces the processor's access of DRAM or ROM memory across the bus when L1 misses
    - Uses a separate data path instead of system bus for transferring data between the L2 cache and the processor
    - Some processors incorporate L2 cache on processor chip

Numbers of Cache (cont.)

- Unified or split
  - Unified
    - Higher hit rate for automatic balancing of load between instruction and data
    - Only one cache is needed to be designed and implemented
  - Split
    - Eliminate contention for the cache between the instruction fetch/decode unit and the execution unit, which is important to the pipelining of instructions

Summary

- Cache
  - Motivation, basic idea, work procedure
- Some questions about cache's work procedure
- Elements of cache design
  - Cache size, mapping function, replacement algorithm, write policy, line size, number of caches

Thank You

rentw@nju.edu.en

南京大學
NANJING UNIVERSITY