CS209 Computer Architecture

Cache Memory-III

Somanath Tripathy
IIT Patna

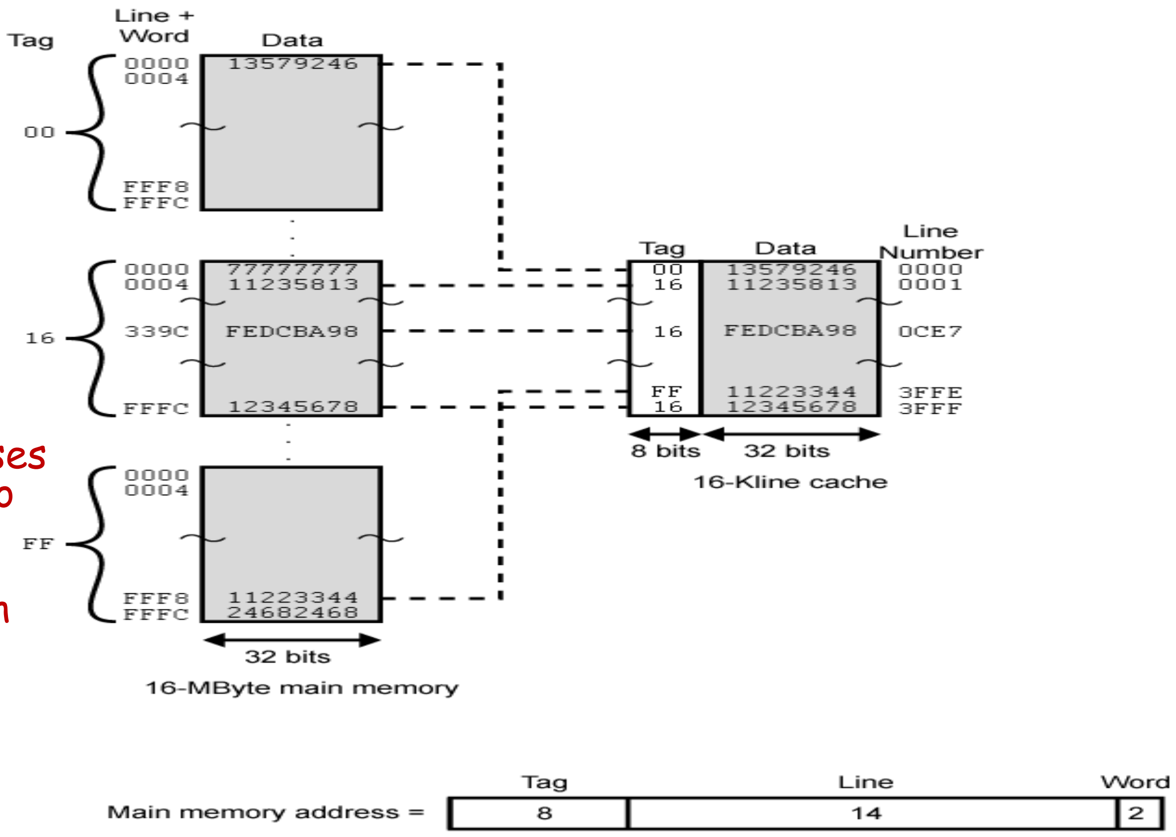
Mapping Function

- How to Map (How the cache is organized)
  - Direct
  - Associative
  - Set associative

- Assume:
  - Cache of 64kByte
  - Cache block of 4 bytes
    - i.e. cache has 16k (2^{14}) lines of 4 bytes
  - 16MBytes main memory
    - 24 bit address

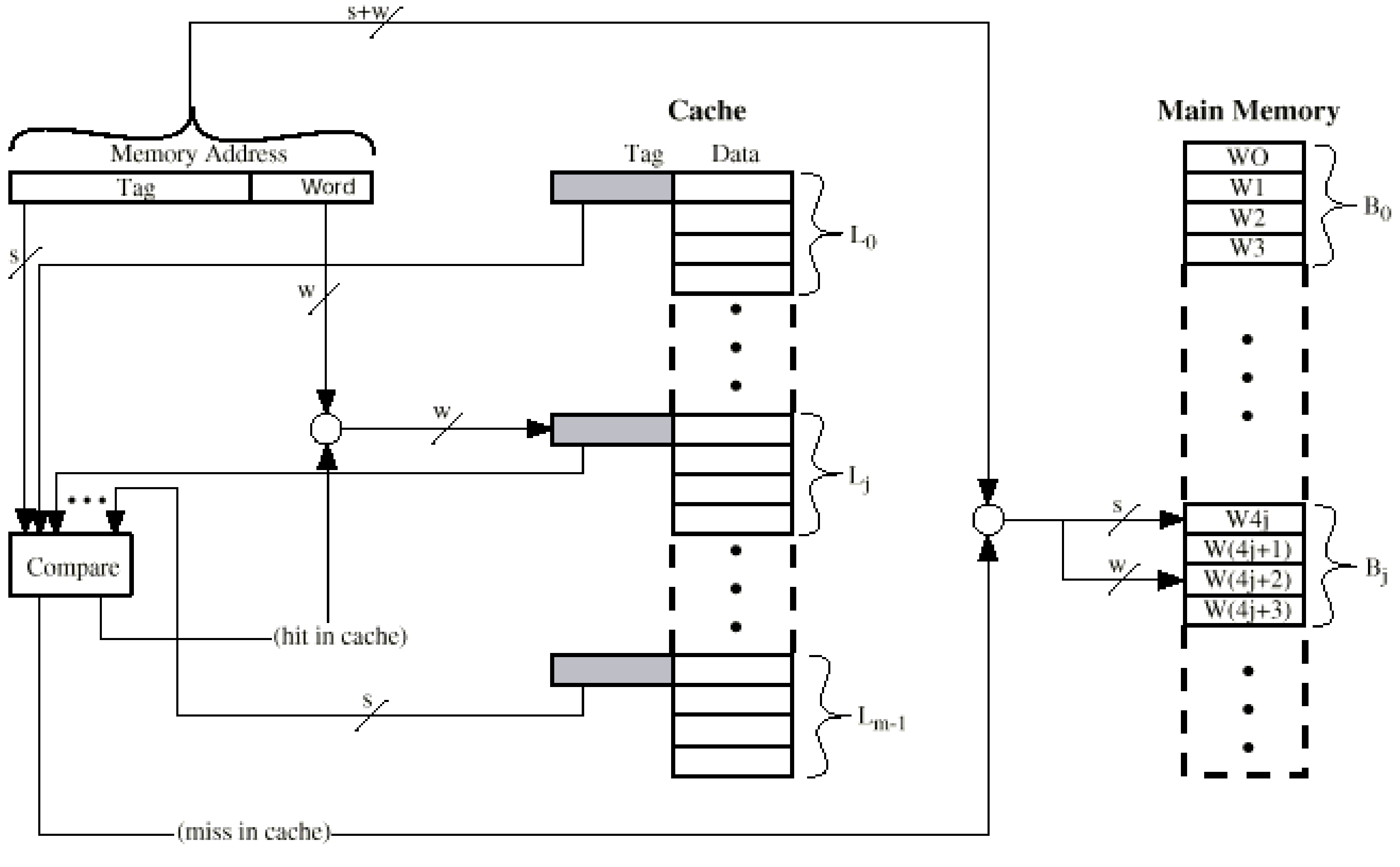
Direct Mapping Example

- Pros
  - Simple
  - Inexpensive
- Cons
  - Fixed location for given block
    - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high
      - Thrashing



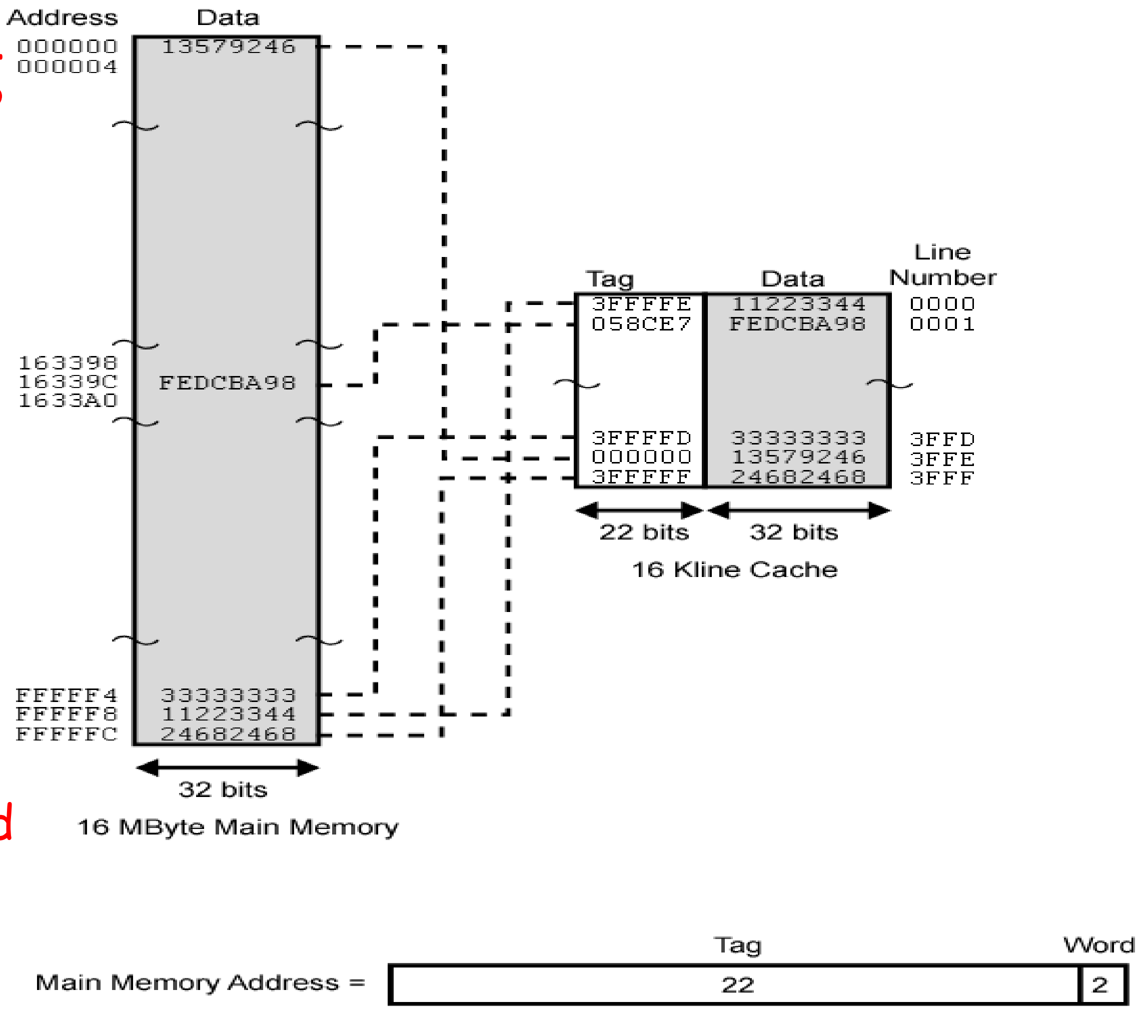
e.g: Address	Tag	Data	Set number	
FFFFF8	FF	11223344	3FFE	
16FFF8	?	?	?	
	16		3FFE	Miss

Fully Associative Cache Organization



Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive
- Complex circuitry required to examine the tags of all the cache lines in parallel



e.g. Address	Tag	Data	Cache line
FFFFFC	3FFFFF	24682468	3FFF
16339C	?	?	?
	058CE7		0001

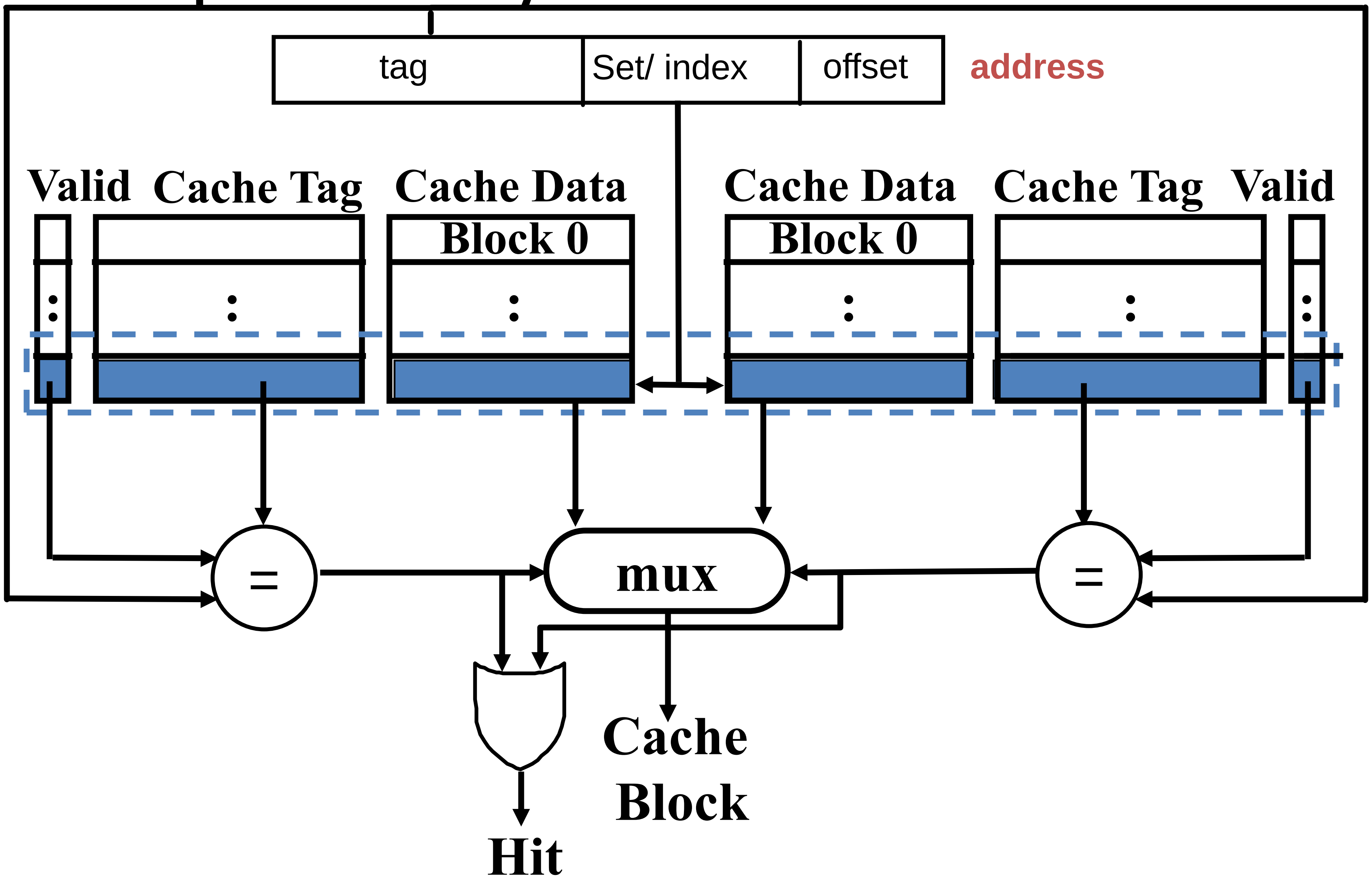
Set Associative Mapping

- To exhibit the strength of both direct and associative mapping while reducing their disadvantages

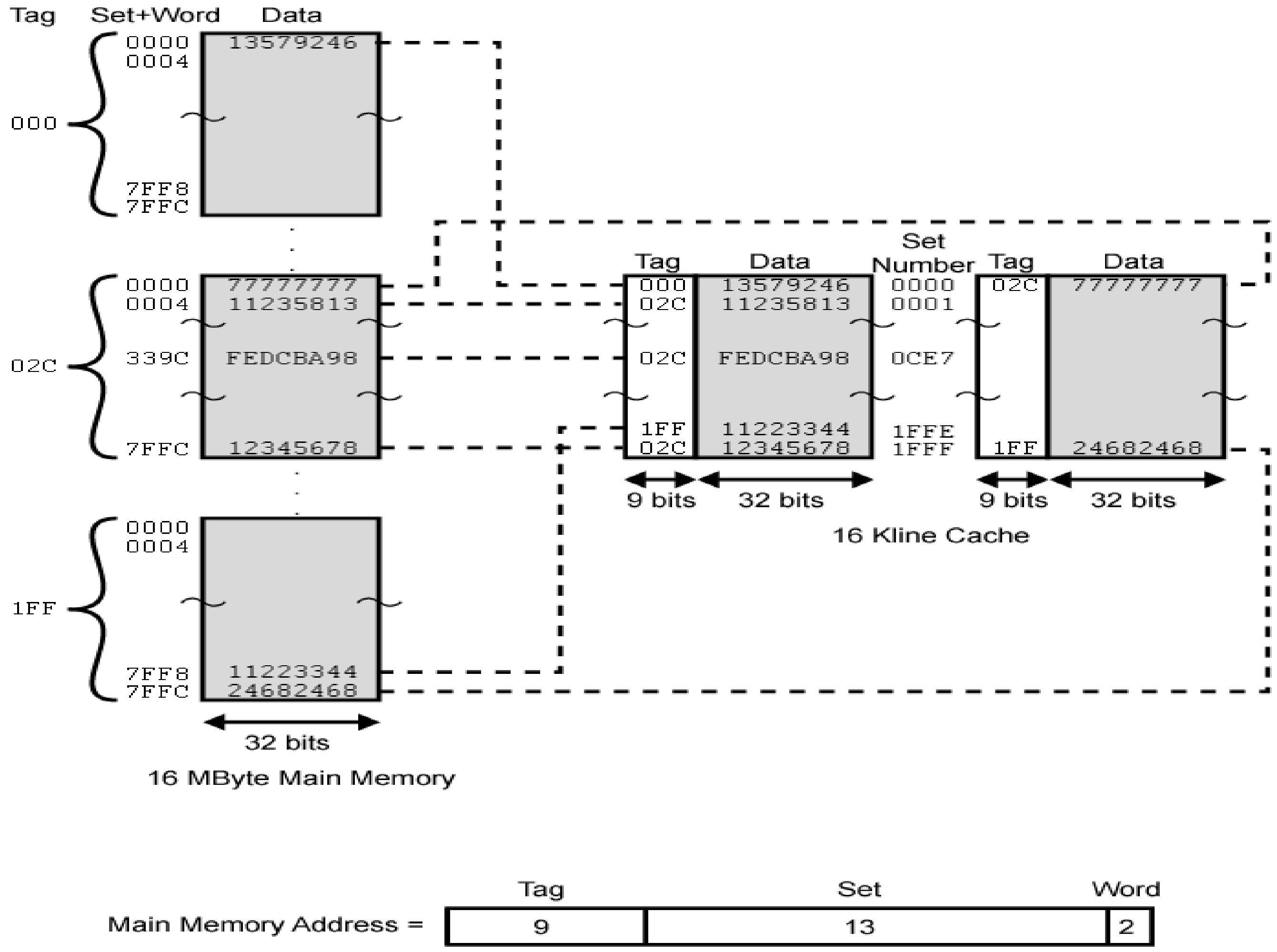
- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i

- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Example: 2-way Set Associative Cache



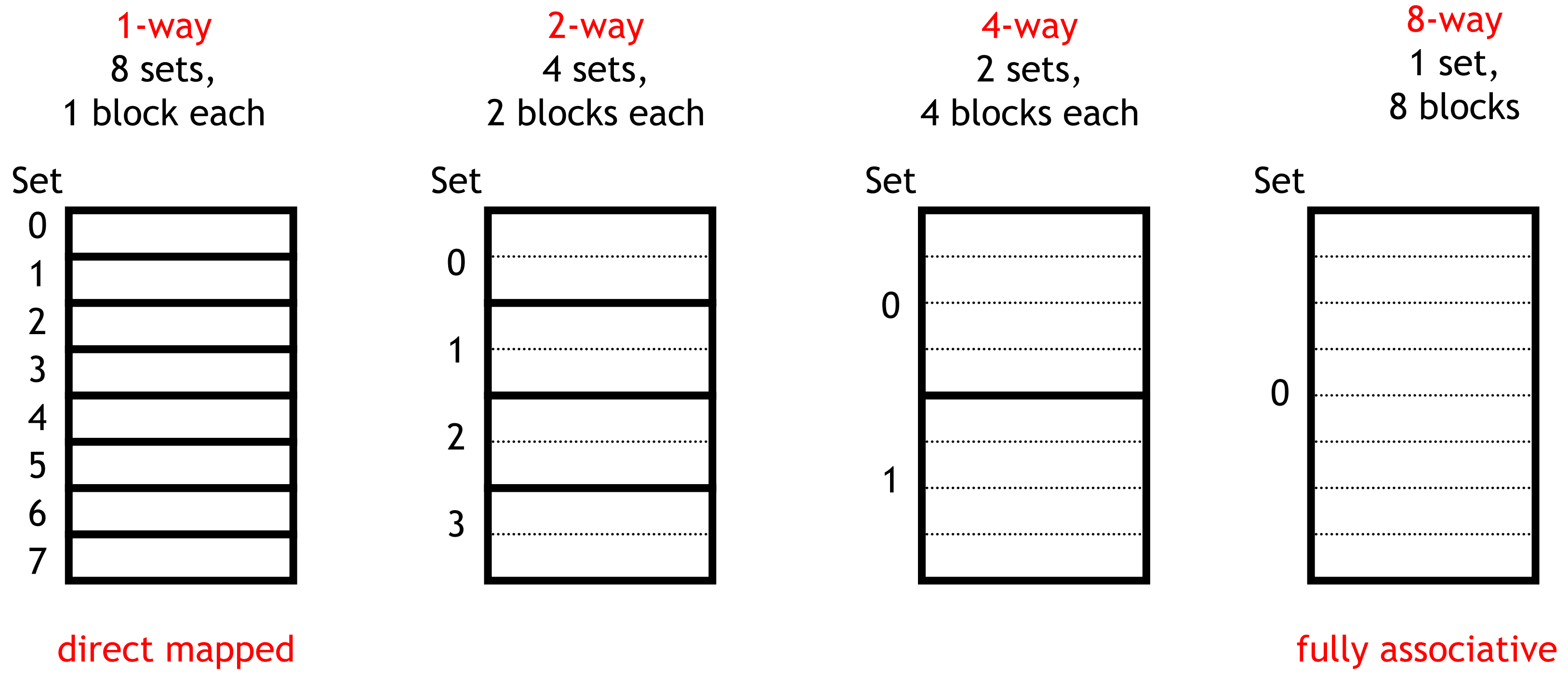
Two Way Set Associative Mapping Example



e.G Address	Tag	Data	Line number or set
FFFFF8	1FF	11223344	1FFE
FFFFFC			
	1FF	24662468	1FFF

Set associative caches general idea

- Notice that 1-way set associative cache is the same as a direct-mapped cache.
- Similarly, if a cache has 2^k blocks, a 2^k-way set associative cache would be the same as a fully-associative cache.



Finding a Block

Mapping method	**Location method**	**Tag comparisons**
Direct mapped	Index	1
k-way set associative	Set index, then search entries within the set	k
Fully associative	Search all entries	#entries

Sources of Misses

- if we try to write to an address that is not already contained in the cache; is called a write miss.
- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Design change	**Effect on miss rate**
Increase cache size	Decrease capacity misses
Increase associativity	Decrease conflict misses
Increase block size	Decrease compulsory misses

Memory and overall performance

- How do cache hits and misses affect overall system performance?
  - Assuming a hit time of one CPU clock cycle,
    - program execution will continue normally on a cache hit. (assume one clock cycle for an instruction fetch or data access.)
  - For cache misses,
    - the CPU must stall to wait for a load from main memory.
- The total number of stall cycles depends on the number of cache misses *and* the miss penalty.

Memory stall cycles = Memory accesses x miss rate x miss penalty

- To include stalls due to cache misses in CPU performance equations, we have to add them to the "base" number of execution cycles.

CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time

Performance example

- Assume that 33% of the instructions in a program are data accesses. The cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles. Find the Memory stall cycles

Memory stall cycles= Memory accesses x Miss rate x Miss penalty

= 0.33 I x 0.03 x 20 cycles
= 0.2 I cycles

- If I instructions are executed, then the number of wasted cycles will be 0.2 x I.
  - This code is 1.2 times slower than a program with a "perfect" CPI of 1!

Cache Types

Single level | Multi level

On-chip | Off-chip

- on-chip : fast but small
- off-chip : large but slow

Unified | Split

- Split allows specializing each part
- Unified allows best use of the capacity

Intel Cache evolution

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four way set associative organization
- Pentium (all versions) – two on chip L1 caches

  Data & instructions
- Pentium III – L3 cache added off chip

- Pentium 4
  - L1 caches
    - 8k bytes
    - 64 byte lines
    - four way set associative
  - L2 cache
    - Feeding both L1 caches
    - 256k
    - 128 byte lines
    - 8 way set associative
  - L3 cache on chip

Basic main memory design

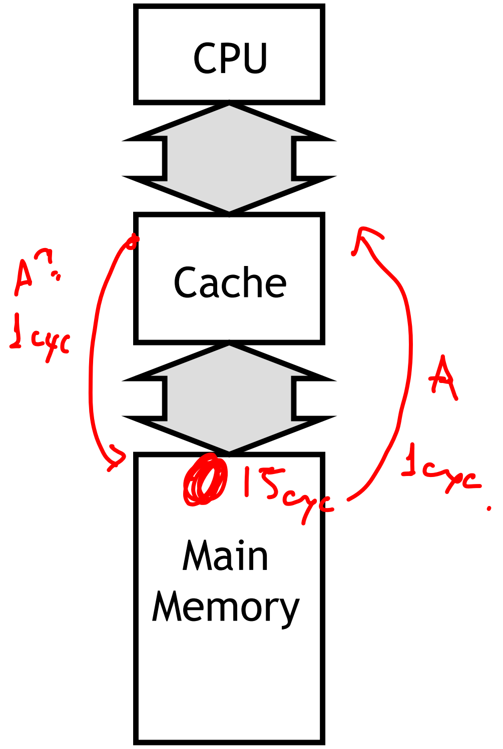
- There are some ways the main memory can be organized to reduce miss penalties and help with caching.

- Assume that following three steps are taken when a cache needs to load data from the main memory.
  1. It takes 1 cycle to send an address to the RAM. ✓
  2. There is a 15-cycle latency for each RAM access.
  3. It takes 1 cycle to return data from the RAM.

- In the setup shown here, the buses from the CPU to the cache and from the cache to RAM are all one word wide.
- If the cache has one-word blocks, then filling a block from RAM (*i.e.*, the miss penalty) would take 17 cycles.

1 + 15 + 1 = 17 clock cycles

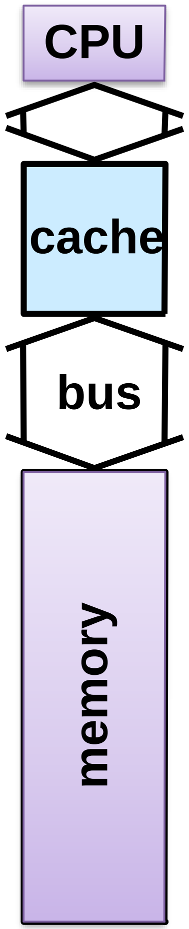
- The cache controller has to send the desired address to the RAM, wait and receive the data.



Miss penalties for larger cache blocks

- If the cache has four-word blocks, then loading a single block would need four individual main memory accesses, and a miss penalty of 68 (65) cycles!

- 4 x (1 + 15 + 1) = 68 clock cycles
- 1 + 4 x 15 + 4 x 1 = 65 clock cycles

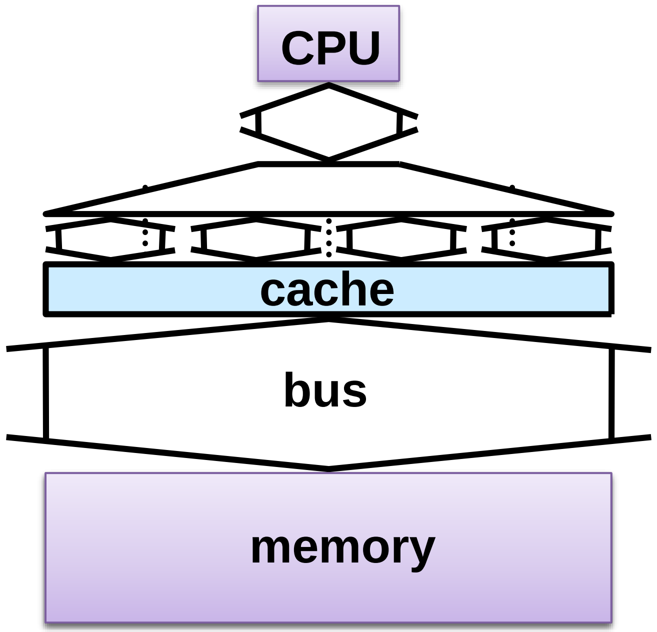


A wider memory

- A simple way to decrease the miss penalty is
  - to widen the memory and its interface to the cache, so we can read multiple words from RAM in one shot.
  - If we could read four words from the memory at once we would need just

    1 + 15 + 1 = 17 cycles

- The disadvantage is the cost of the wider buses—each additional bit of memory width requires another connection to the cache.

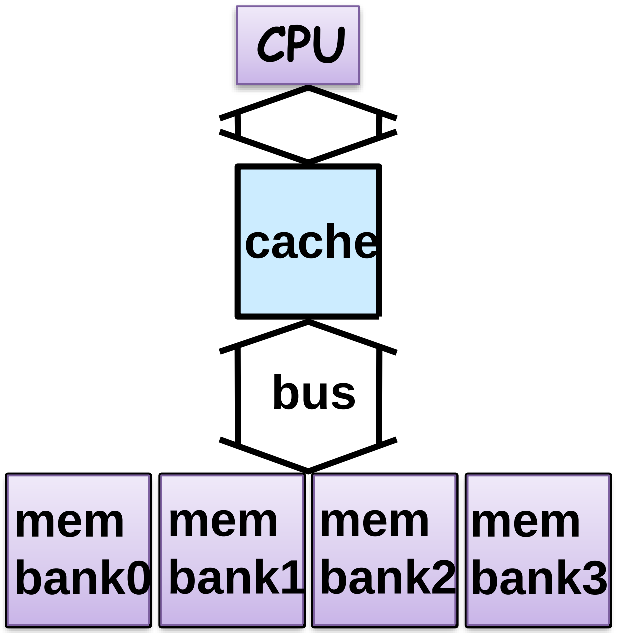


An interleaved memory

- Another approach is to interleave the memory, or split it into "banks" that can be accessed individually.
  - The main benefit is overlapping the latencies of accessing each word.
- For example, if our main memory has four banks, each one word wide, then we could load four words into a cache block in just 20 cycles.

  1 + 15 + (4 x 1) = 20 cycles

- Our buses are still one word wide here, so four cycles are needed to transfer data to the caches.
- This is cheaper than implementing a four-word bus, but not too much slower.



Cache Policies

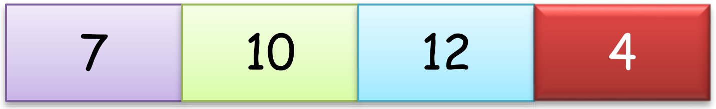
- Read:
  - Sequential / Concurrent
  - Simple / Forward
- Replacement: which one?
  - LRU / LFU / FIFO / Random
- Write:
  - Write back/ Write Through

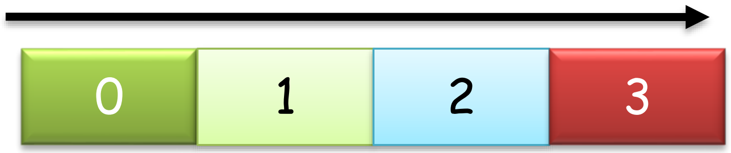
Read policies

- Sequential or concurrent
  - initiate memory access only after detecting a miss
  - initiate memory access along with cache access in anticipation of a miss
- With or without forwarding
  - give data to CPU after filling the missing block in cache
  - forward data to CPU as it gets filled in cache

Replacement Policies: Only for Associative cache

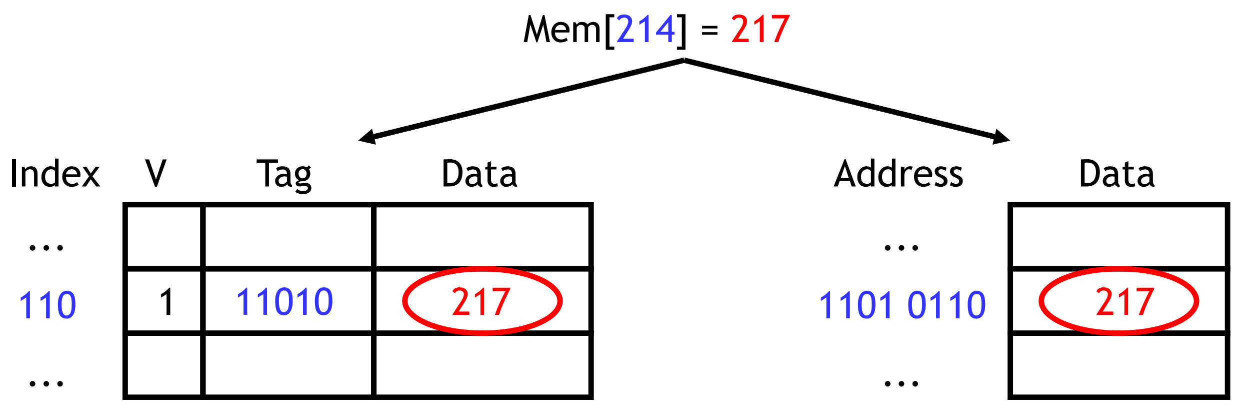
- Least Recently Used (LRU)
  - Add time stamp to each access
- Least Frequently Used (LFU)
  - Newly added line have frequency 0
  - Add Frequency Ctr for each line access
- First In First Out (FIFO)
  - Choose the in order
- Random
  - Randomly choose





Write-through caches

- A write-through cache forces all writes to update both the cache *and* the main memory.

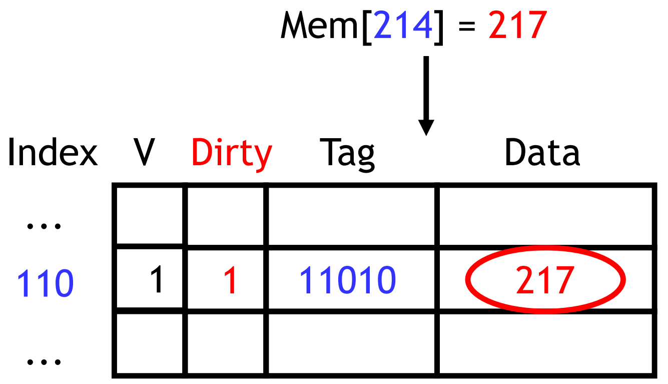


- This is simple to implement and keeps the cache and memory consistent.
- The bad thing is that forcing every write to go to main memory, we use up bandwidth between the cache and the memory.

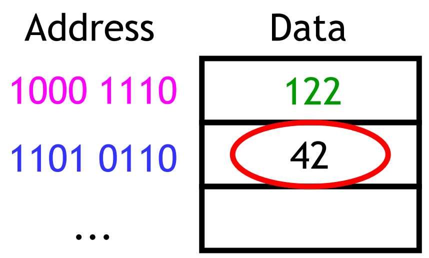
Write-back caches

- In a write-back cache, the memory is not updated until the cache block needs to be replaced (*e.g.*, when loading data into a full cache set).
- For example, we might write some data to the cache at first, leaving it inconsistent with the main memory as shown before.
  - The cache block is marked "dirty" to indicate this inconsistency

- Advantage: not all write operations need to access main memory, as with write-through caches.
  - If a single address is frequently written to, then it doesn't pay to keep writing that data through to main memory.
  - If several bytes within the same cache block are modified, they will only force one memory write operation at write-back time.



Index	V	Dirty	Tag	Data
...				
110	1	1	11010	217
...				



Address	Data
1000 1110	122
1101 0110	42
...	

ALL THE BEST FOR YOUR END-SEM EXAM

THANKS