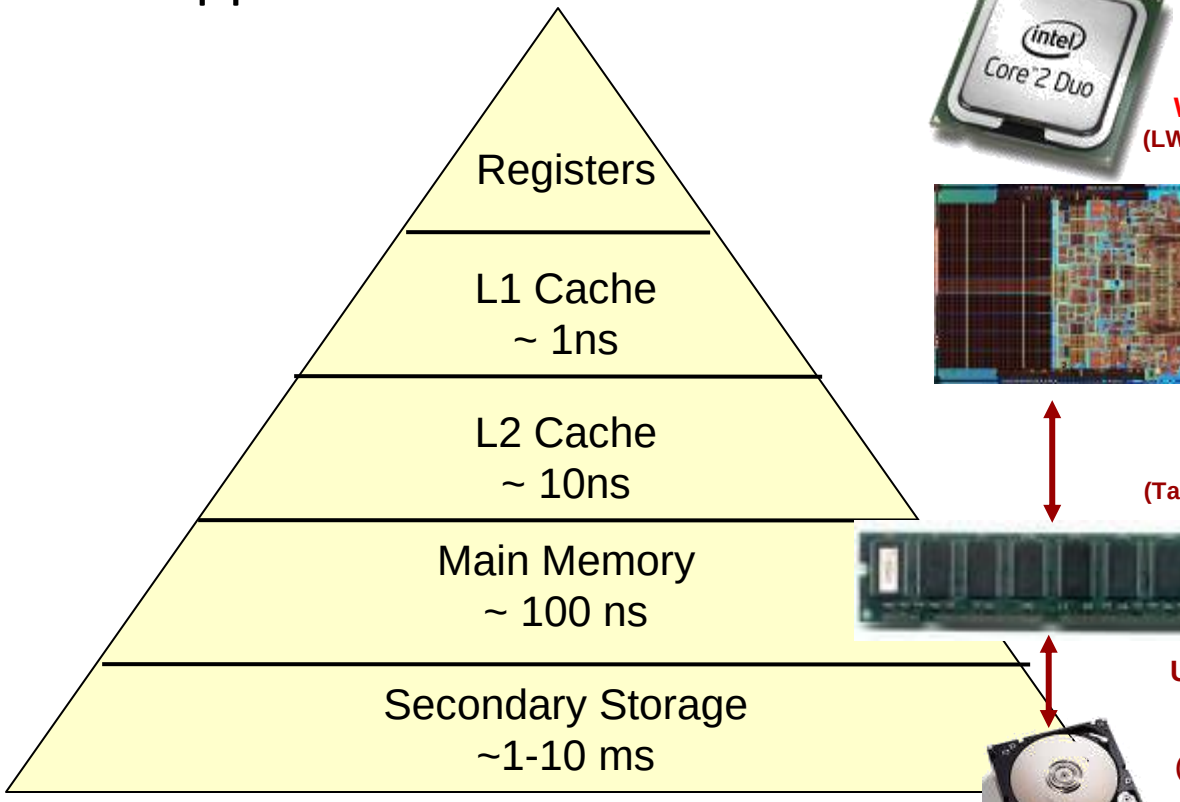
EE 457 Unit 7a

Cache and Memory Hierarchy

Memory Hierarchy & Caching

- Use several levels of faster and faster memory to hide delay of upper levels



Unit of Transfer:
Word, Half, or Byte
(LW, LH, LB or SW, SH, SB)

Unit of Transfer:
Cache block/line
1-8 words
(Take advantage of spatial locality)

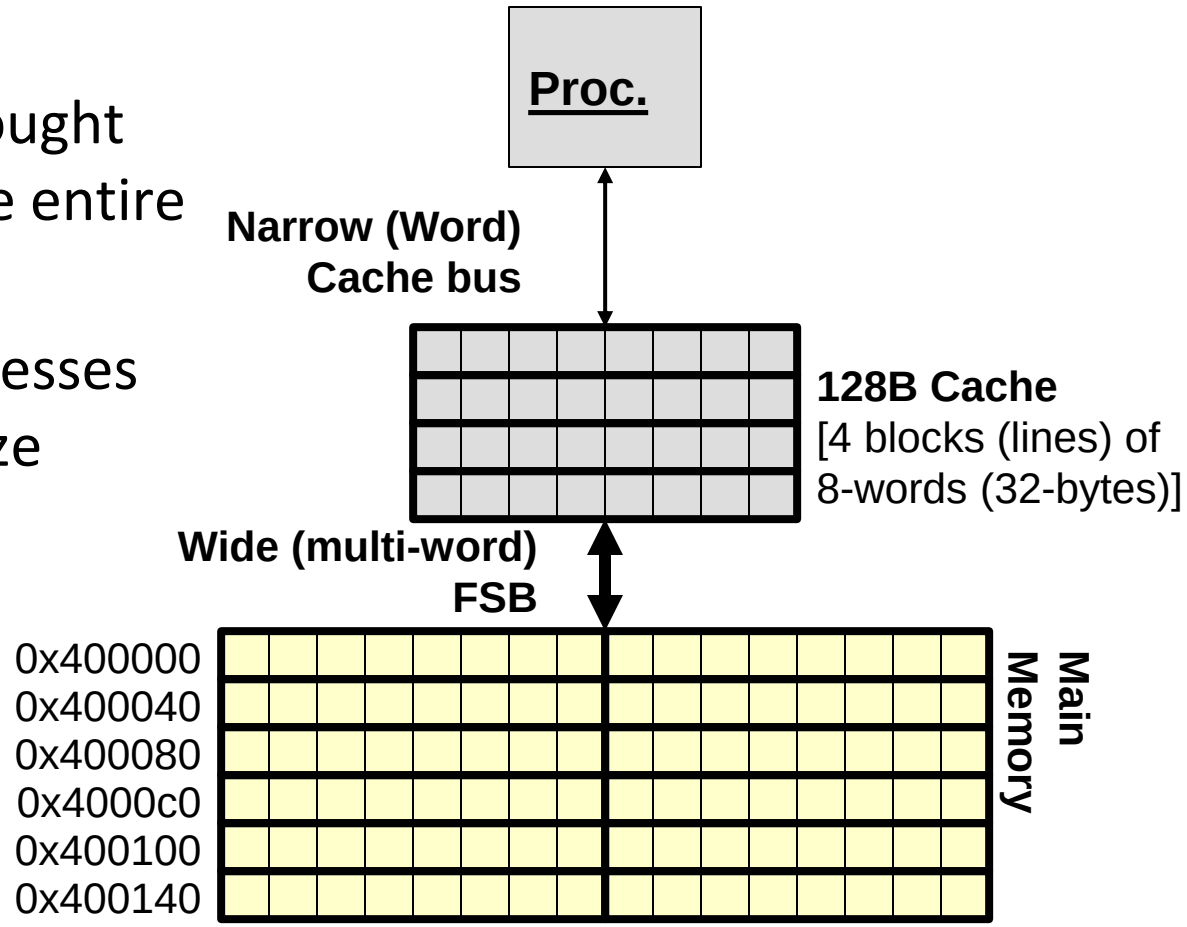
Unit of Transfer:
Page
4KB-64KB words
(Take advantage of spatial locality)

Smaller — **More Expensive** — **Faster**

Larger — **Less Expensive** — **Slower**

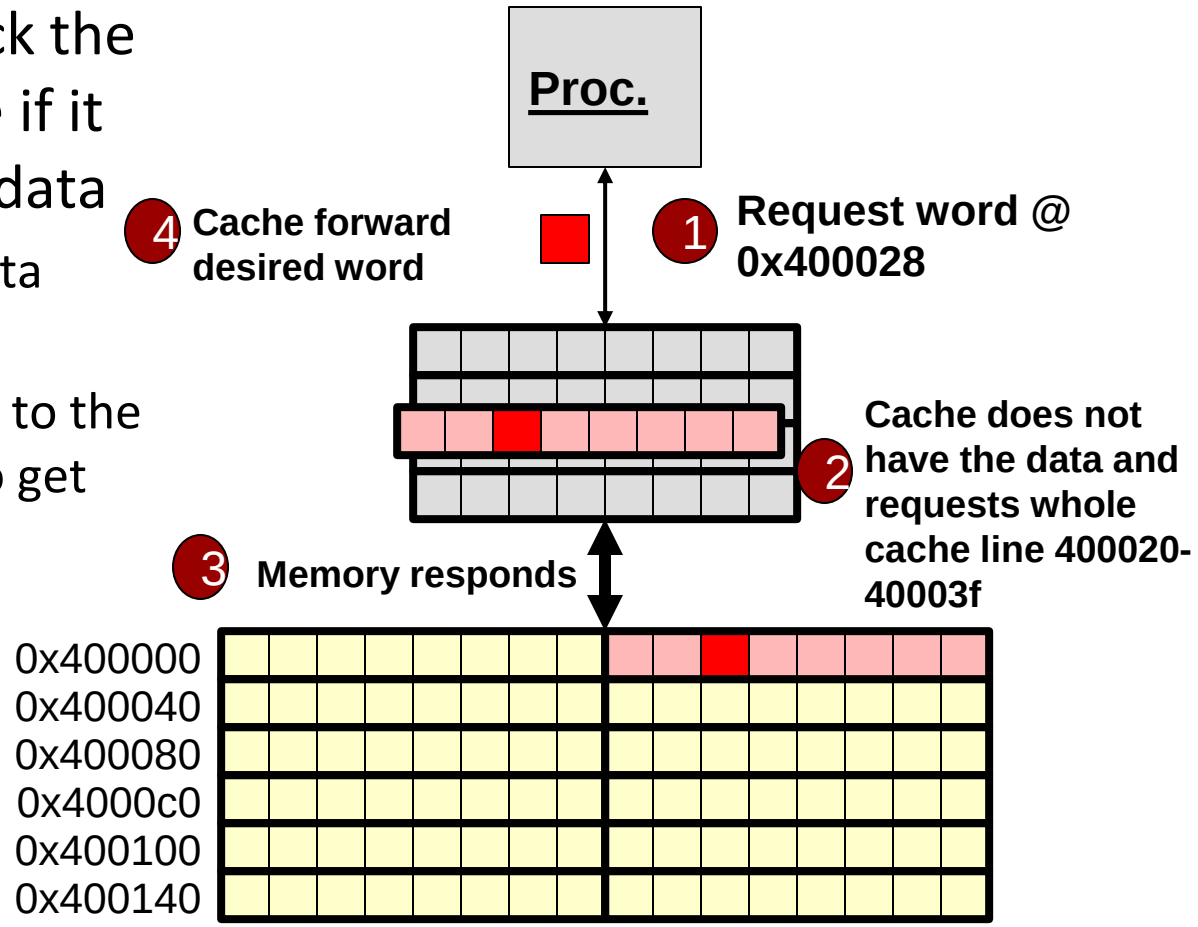
Cache Blocks/Lines

- Cache is broken into "blocks" or "lines"
  - Any time data is brought in, it will bring in the entire block of data
  - Blocks start on addresses multiples of their size



Cache Blocks/Lines

- Whenever the processor generates a read or a write, it will first check the cache memory to see if it contains the desired data
  - If so, it can get the data quickly from cache
  - Otherwise, it must go to the slow main memory to get the data



Cache & Virtual Memory

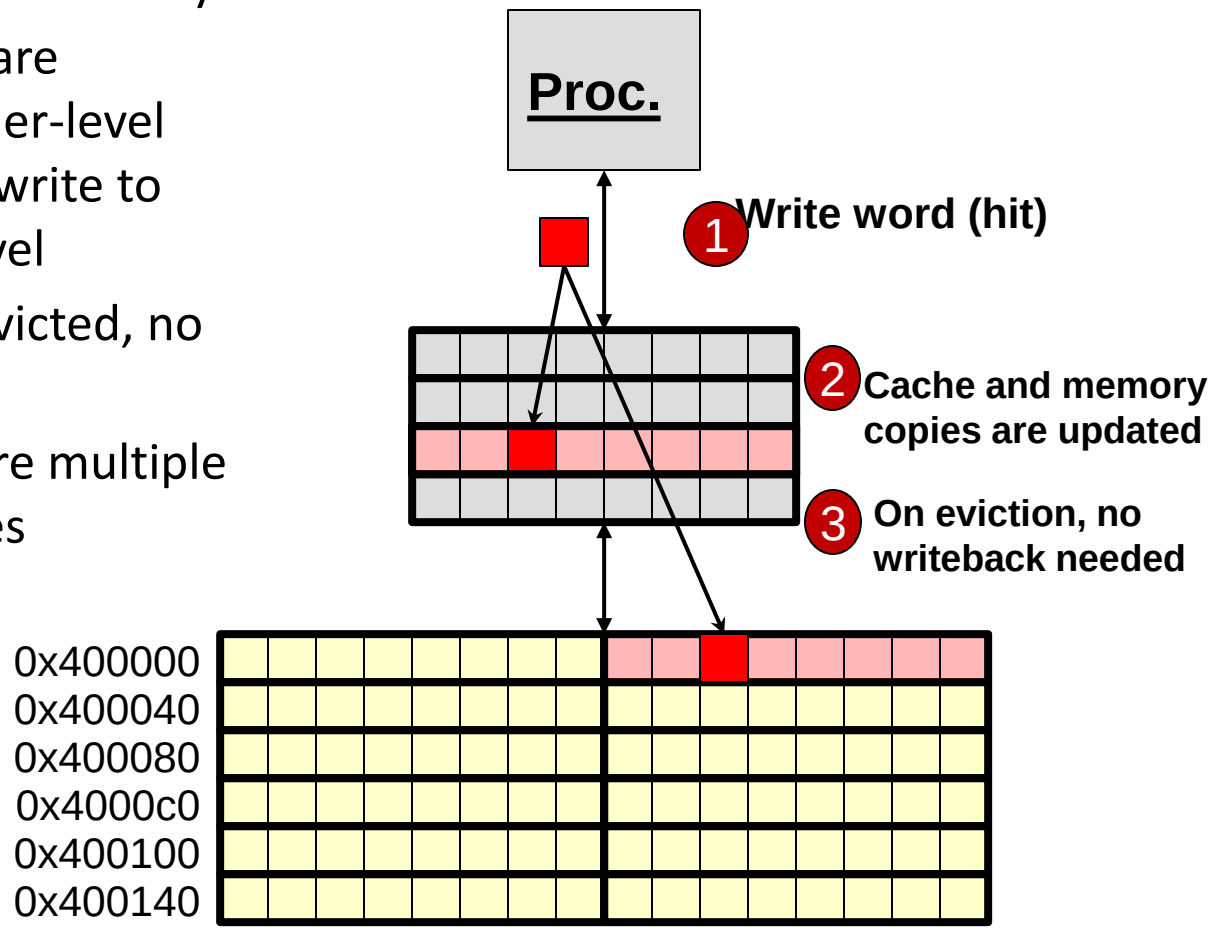
- Exploits the Principle of Locality
  - Allows us to implement a hierarchy of memories: cache, MM, second storage
  - Temporal Locality: If an item is reference it will tend to be referenced again soon
    - Examples: Loops, repeatedly called subroutines, setting a variable and then reusing it many times
  - Spatial Locality: If an item is referenced items whose addresses are nearby will tend to be referenced soon
    - Examples: Arrays and program code

Cache Definitions

- **Cache Hit** = Desired data is in cache
- **Cache Miss** = Desired data is not present in cache
- When a cache miss occurs, a new block is brought from MM into cache
  - Load Through: First load the word requested by the CPU and forward it to the CPU, while continuing to bring in the remainder of the block
  - No-Load Through: First load entire block into cache, then forward requested word to CPU
- On a Write-Miss we may choose to not bring in the MM block since writes exhibit less locality of reference compared to reads
- When CPU writes to cache, we may use one of two policies:
  - **Write Through (Store Through):** Every write updates both cache and MM copies to keep them in sync. (i.e. coherent)
  - **Write Back**: Let the CPU keep writing to cache at fast rate, not updating MM. Only copy the block back to MM when it needs to be replaced or flushed

Write Through Cache

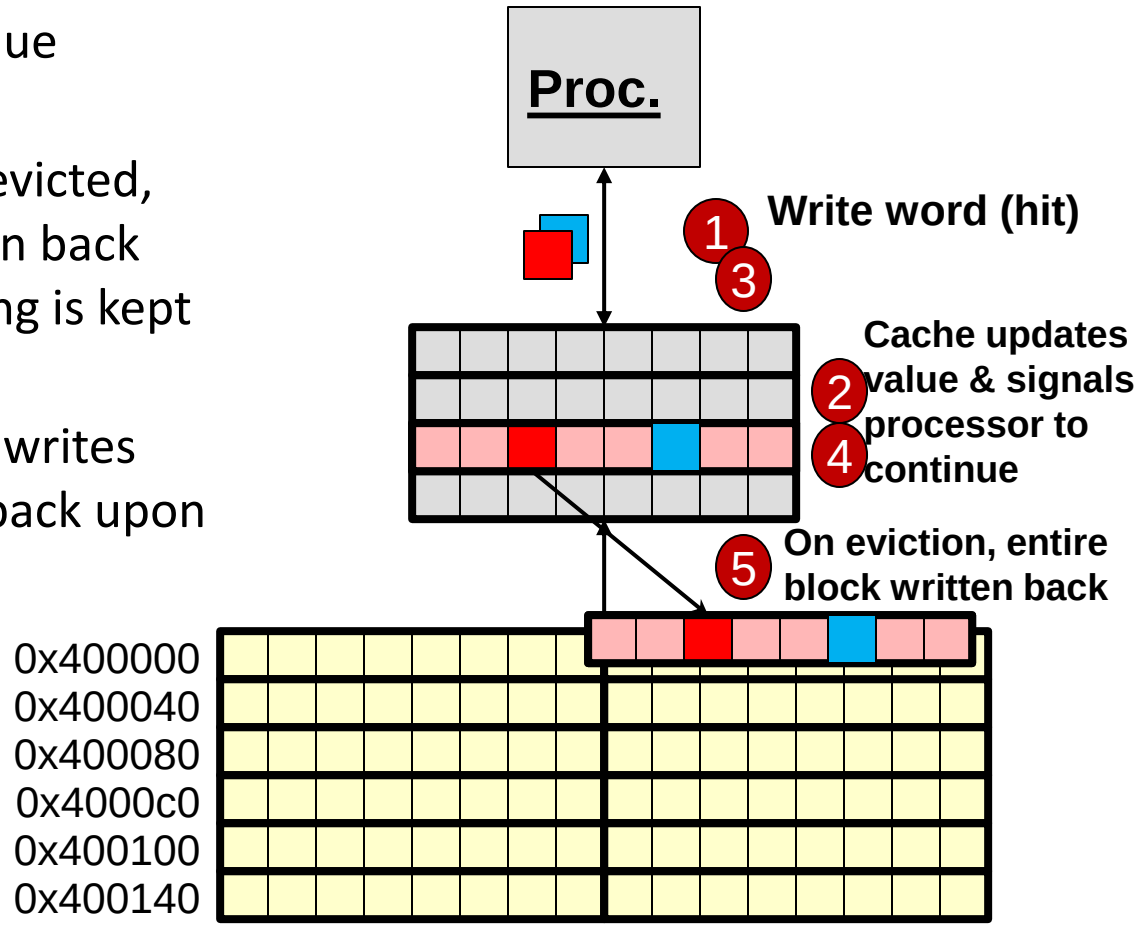
- Write-through option:
  - Update both levels of hierarchy
  - Depending on hardware implementation, higher-level may have to wait for write to complete to lower level
  - Later when block is evicted, no writeback is needed
  - Multiple writes require multiple main memory updates



Key Idea: Communicate **EVERY** change to main memory as they happen (keeps both copies in sync)

Write Back Cache

- Write-back option:
  - Update only cached copy
  - Processor can continue quickly
  - Later when block is evicted, entire block is written back (because bookkeeping is kept on a per block basis)
  - Notice that multiple writes only require 1 writeback upon eviction



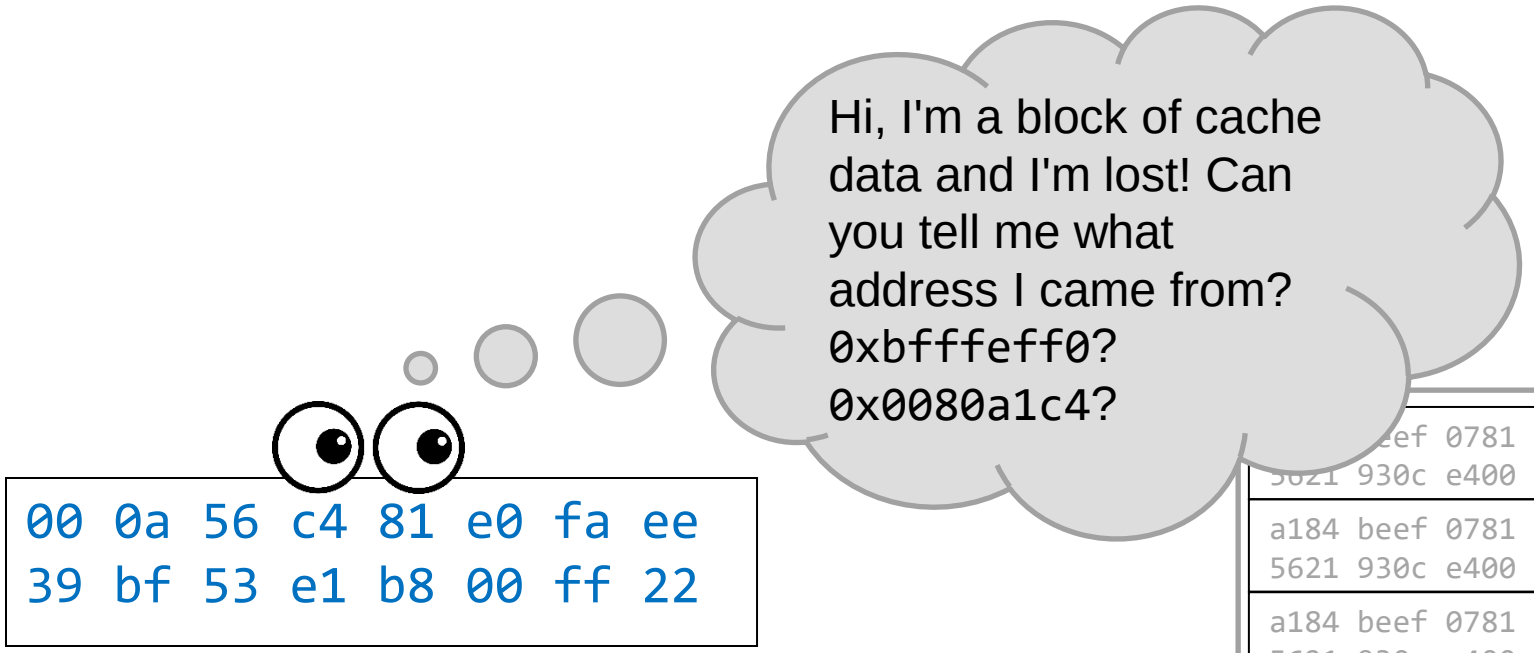
Key Idea: Communicate **ONLY the FINAL version** of a block to main memory (when the block is evicted)

Mapping and Replacement

- **Mapping Function**: The correspondence between MM blocks and cache block frames is specified by means of a mapping function
  - **Fully Associative** (increases hit rate, but large, slow hardware)
  - **Direct Mapping** (decreased hit rate, but fast, simple hardware)
  - **Set Associative** (tunable compromise of the above two methods)
- **Replacement Algorithm**: How do we decide which of the current cache blocks is removed to create space for a new block
  - Random
  - Least Recently Used (LRU)

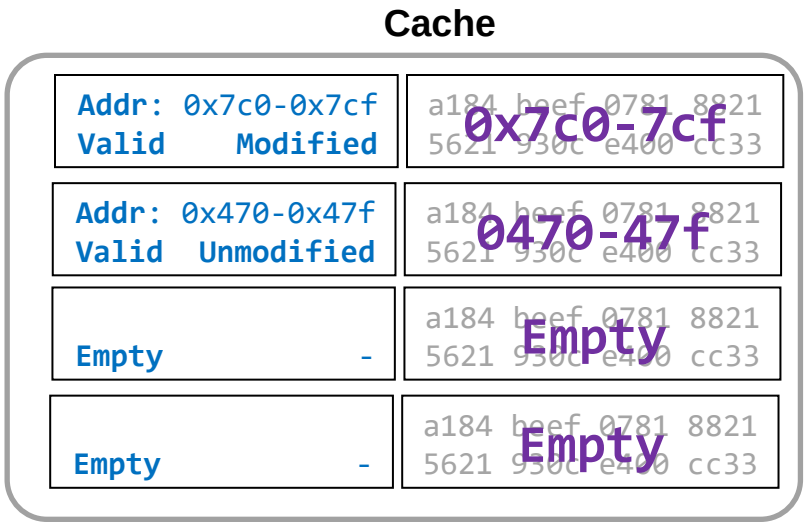
CACHE MAPPINGS OVERVIEW

Cache Question



Cache Implementation

- Assume a cache of 4 blocks of 16-bytes each
- Must store more than just data!
- What other bookkeeping and identification info is needed?
  - Is the block **empty** or **full**?
  - Has the block been **modified**?
  - Where did the block come from? **Address range** of the block data?



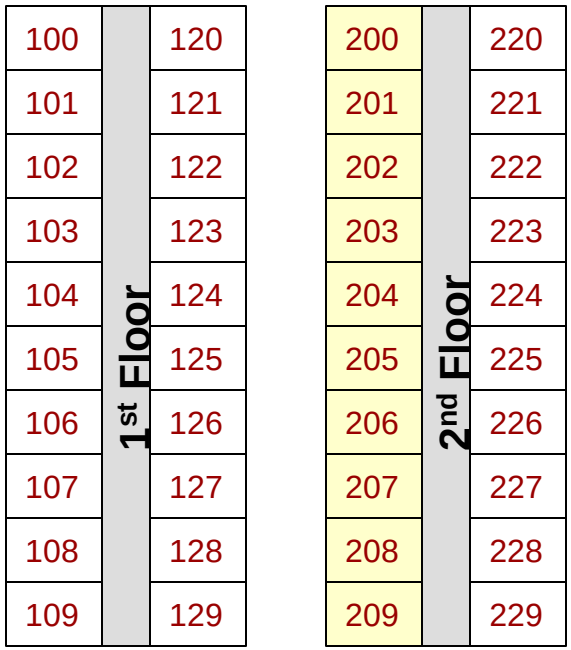
Implementation Terminology

What bookkeeping values must be stored with the cache in addition to the block data?

- **Valid bit**: An additional bit is maintained to indicate that whether the TAG is valid (meaning it contains the TAG of an actual block)
  - Initially when you turn power on the cache is empty and all valid bits are turned to '0' (invalid)
- **Dirty Bit**: This bit associated with the TAG indicates when the block was modified (got dirtied) during its stay in the cache and thus needs to written back to MM
  - Used only with the write-back cache policy
- **Tag** – *Portion* of the block's address range used to identify the MM block residing in the cache from other MM blocks

Identifying Blocks via Address Range

- Possible methods
  - Store start and end address (requires multiple comparisons)
  - Ensure block ranges sit on binary boundaries (upper address bits identify the block with a single value)
    - Analogy: Hotel room layout/addressing



Analogy: Hotel Rooms

1st Digit = Floor
2nd Digit = Aisle
3rd Digit = Room w/in aisle

To refer to the range of rooms on the **second floor, left aisle** we would just say **rooms 20x**

4 word (16-byte) blocks:

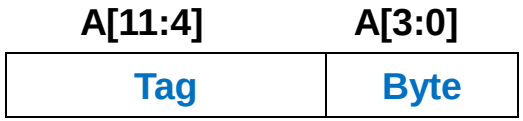
Addr. Range	Binary		
000-00f	0000	0000	0000..1111
010-01f	0000	0001	0000..1111

8 word (32-byte) blocks:

Addr. Range	Binary		
000-01f	0000	000	00000..11111
020-03f	0000	001	00000..11111

Cache Implementation

- Assume 12-bit addresses and 16-byte blocks
- Block offset will range from xx0 to xxF
  - Address can be broken down as follows
  - A[11:4] = Tag = Identifies block range (i.e., xx0-xxF)
  - A[3:0] = Byte offset within the cache block



Addr. = 0x124

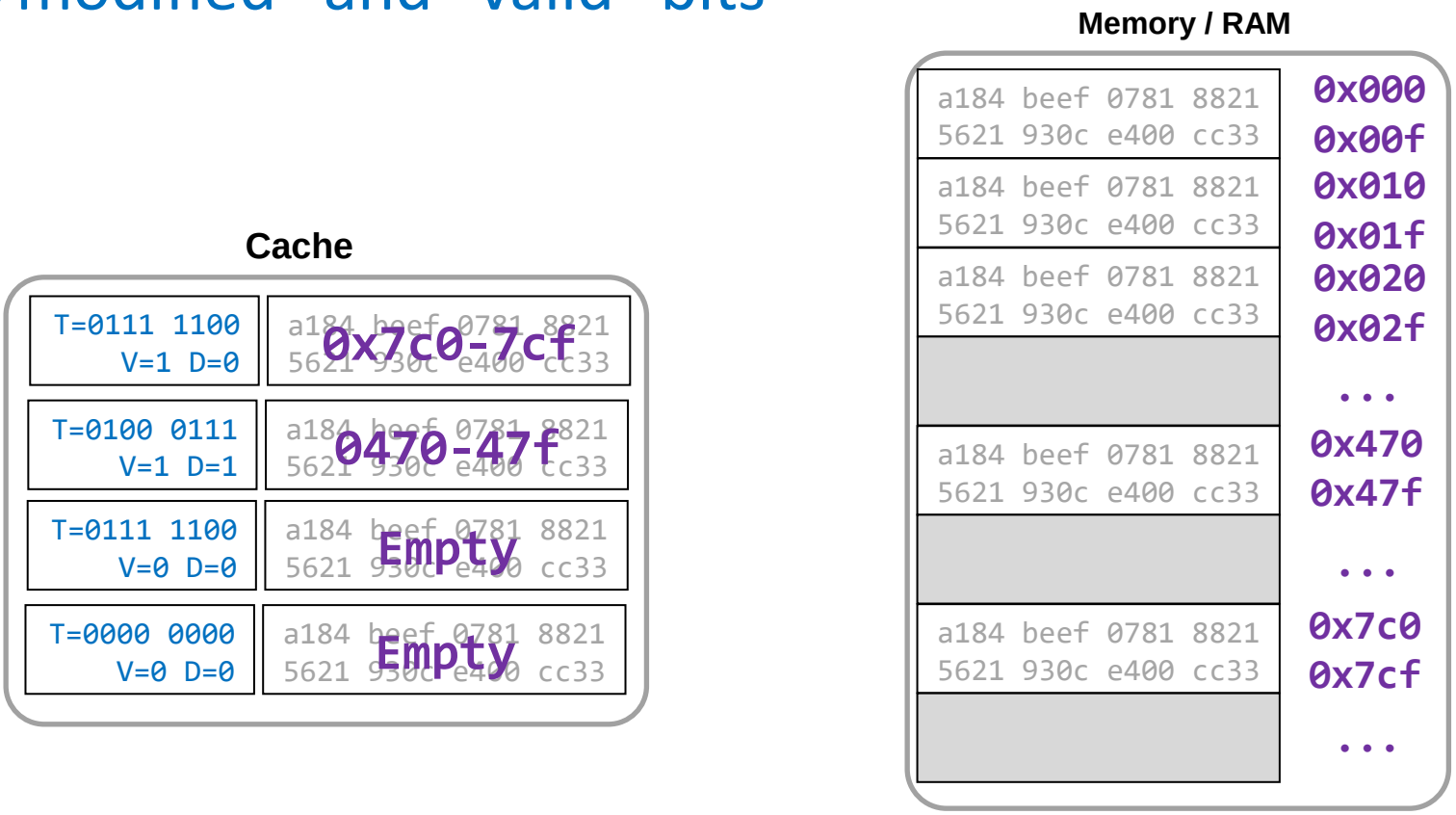
Byte 4 w/in block 120-12F

Addr. = 0xACC

Byte 12 w/in block AC0-ACF

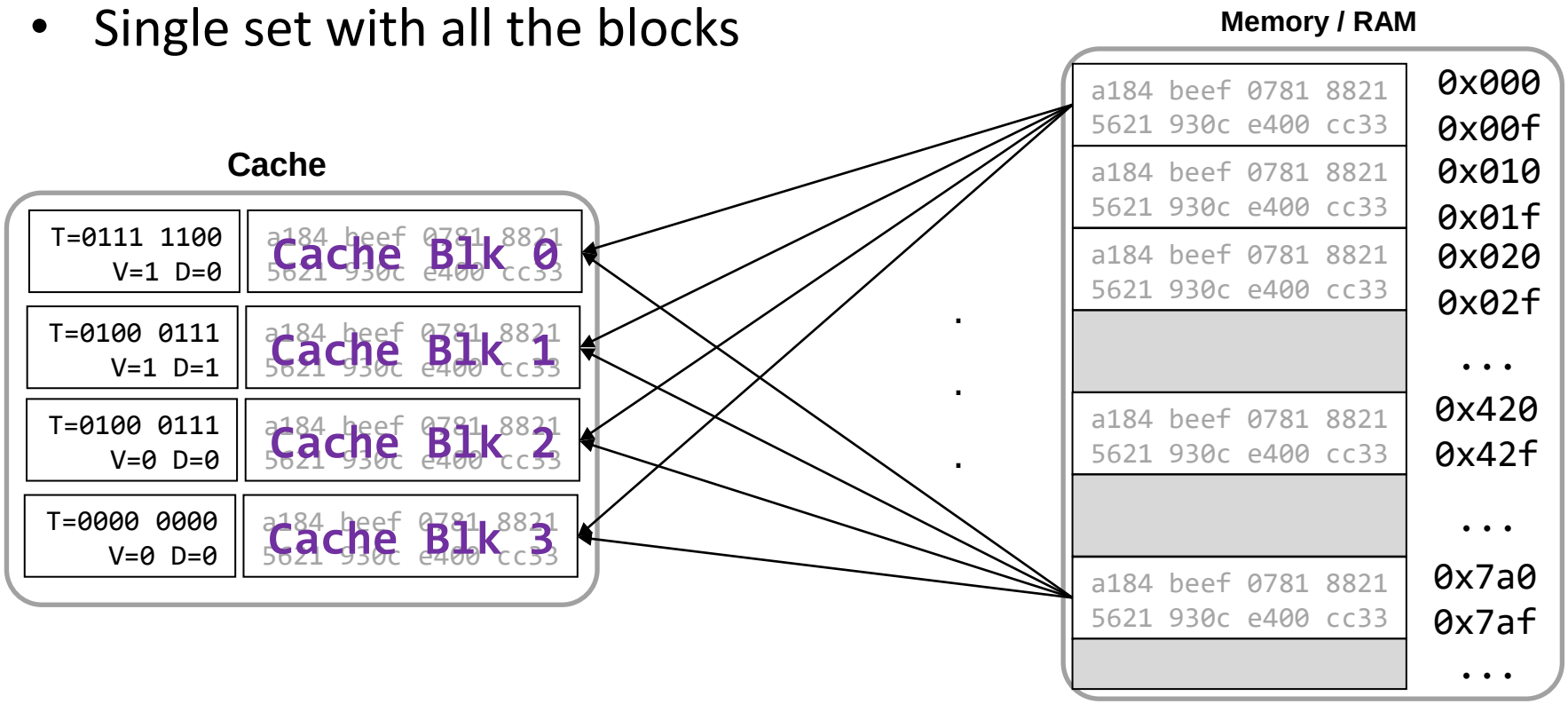
Cache Implementation

- To identify which MM block resides in each cache block, the tags need to be stored along with the "dirty/modified" and "valid" bits



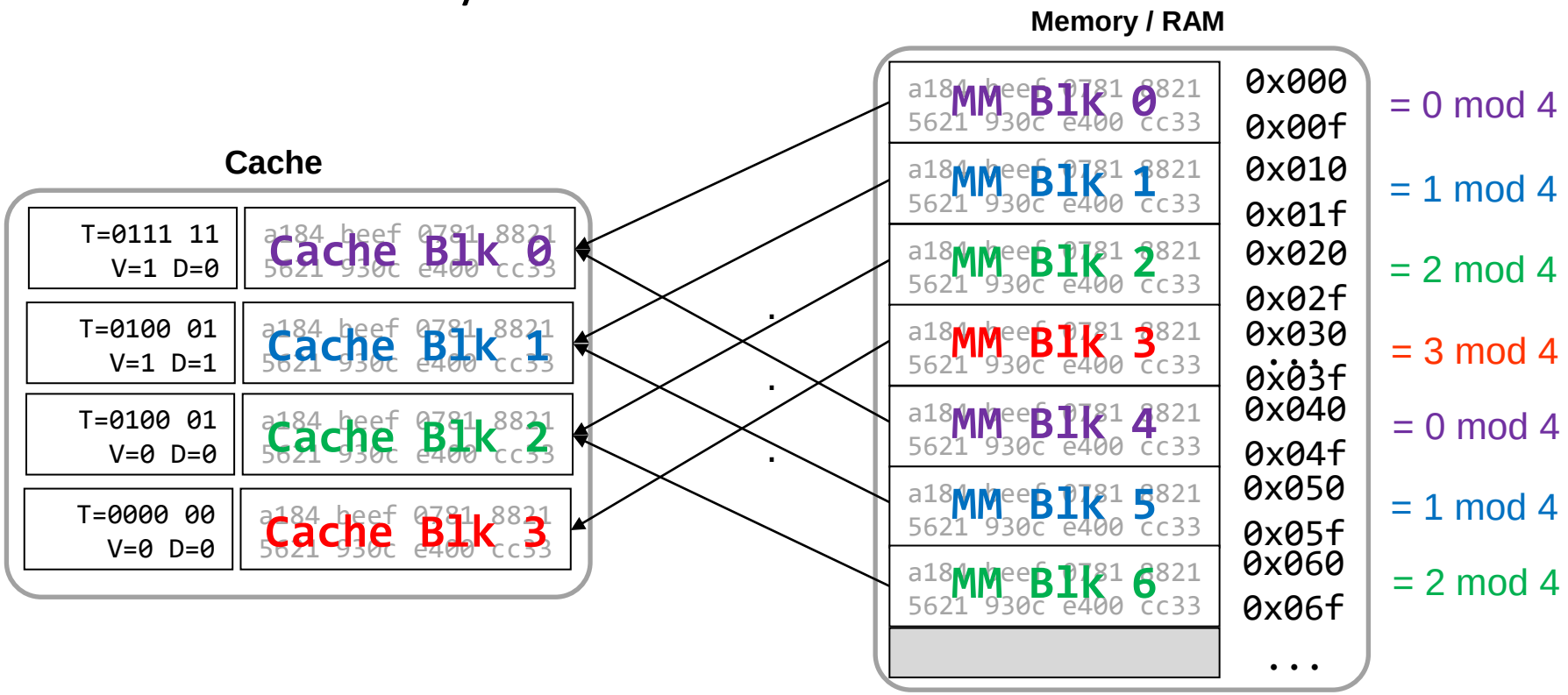
Fully Associative Mapping (S=1, K=N)

- Any block from memory can be put in any cache block (i.e., no restriction)
  - We have to search everywhere to determine hit or miss
- Single set with all the blocks



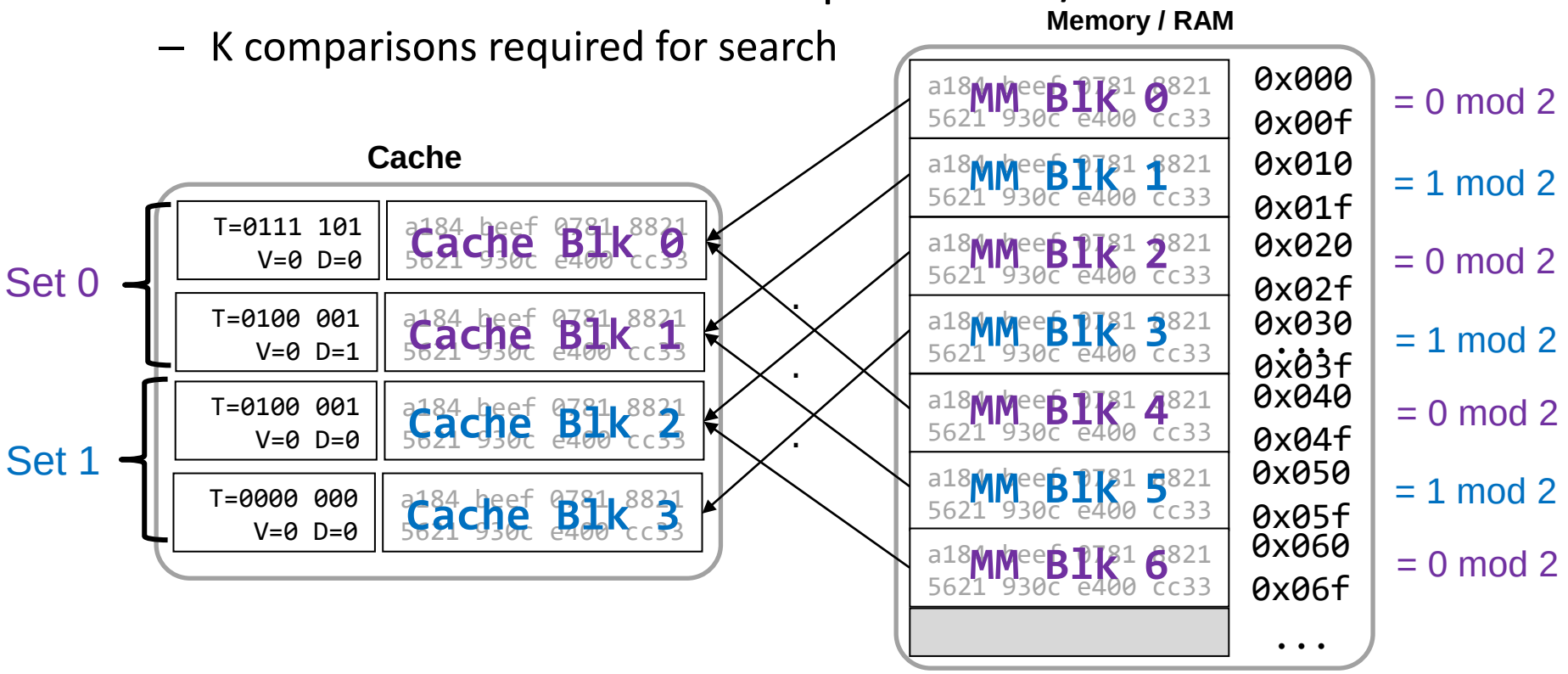
Direct Mapping (S=N, K=1)

- Each block from memory can only be put in one location
- Given *n* cache blocks,
  MM block ***i*** maps to cache block "***i* mod *n***"
- Each set has only 1 block



K-way Set-Associative Mapping

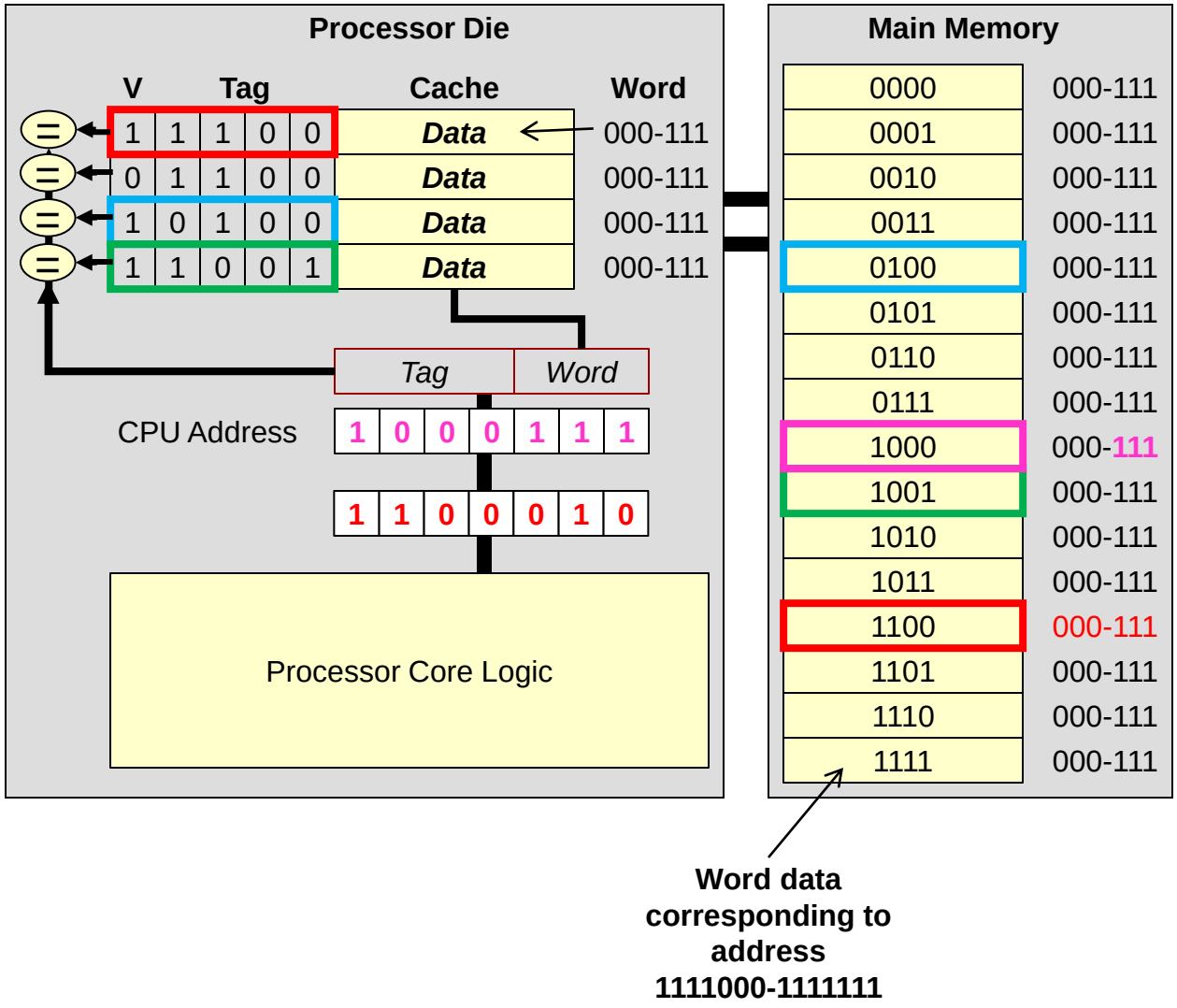
- Given *S* sets, block ***i*** of MM maps to 1 set: "***i* mod *S***"
- Within the set, block can be put anywhere
- Given N = total cache blocks,
  let K = number of cache blocks per set = N/S
  - K comparisons required for search



CACHE MAPPING IMPLEMENTATION

Fully Associative Cache Example

- Cache Mapping Example:
  - Fully Associative
  - MM = 128 words
  - Cache Size = 32 words
  - Block Size = 8 words
- Fully Associative mapping allows a MM block to be placed in (associate with) any cache block
- To determine hit/miss we have to search everywhere



Fully Associative Hit Logic

- Cache Mapping Example:
  - Fully Associative, MM = 128 words (2^7), Cache Size = 32 (2^5) words, Block Size = (2^3) words
- Number of blocks in MM = 2^7 / 2^3 = 2^4
- Block ID = 4 bits
- Number of Cache Block Frames = 2^5 / 2^3 = 2^2 = 4
  - Store 4 Tags of 4-bits + 1 valid bit
  - Need 4 Comparators each of 5 bits
- CAM (Content Addressable Memory) is a special memory structure to store the tag+valid bits that takes the place of these comparators but is too expensive

Fully Associative Does Not Scale

- If 80386 used Fully Associative Cache Mapping :
  - Fully Associative, MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words
- Number of blocks in MM = 2^{32} / 2^4 = 2^{28}
- Block ID = 28 bits
- Number of Cache Block Frames = 2^{16} / 2^4 = 2^{12} = 4096
  - Store 4096 Tags of 28-bits + 1 valid bit
  - Need 4096 Comparators each of 29 bits

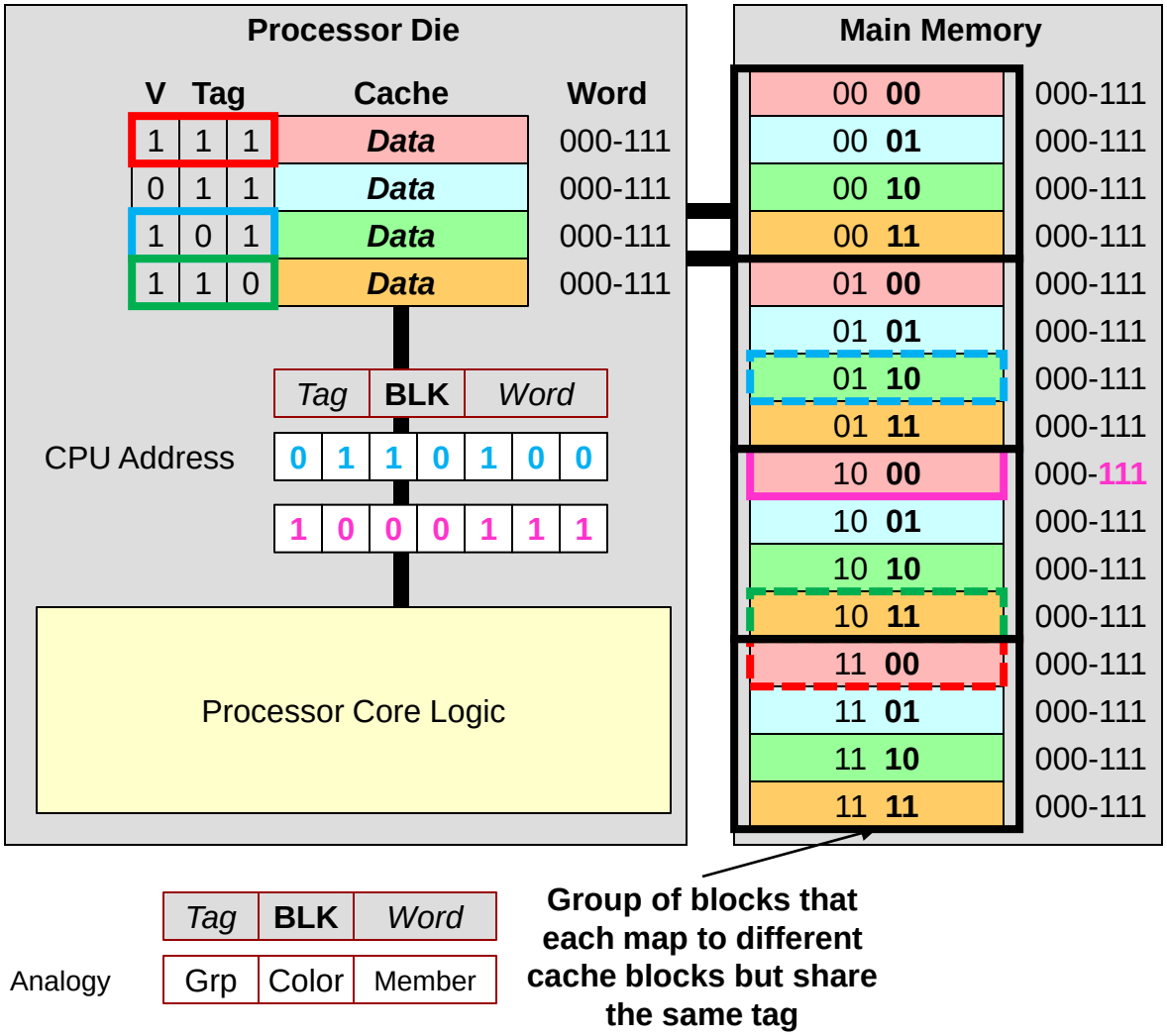
Prohibitively Expensive!!

Fully Associative Address Scheme

- A[1:0] unused => /BE3…/BE0
  - Word access only (LW and SW…no LB, SH, etc.)
- Word bits = $\log_2 B$ bits (B=Block Size)
- Tag = Remaining bits

Direct Mapping Cache Example

- Limit each MM block to one possible location in cache
- Cache Mapping Example:
  - Direct Mapping
  - MM = 128 words
  - Cache Size = 32 words
  - Block Size = 8 words
- Each MM block i maps to Cache frame i mod N
  - N = # of cache frames
  - Tag identifies which group that colored block belongs



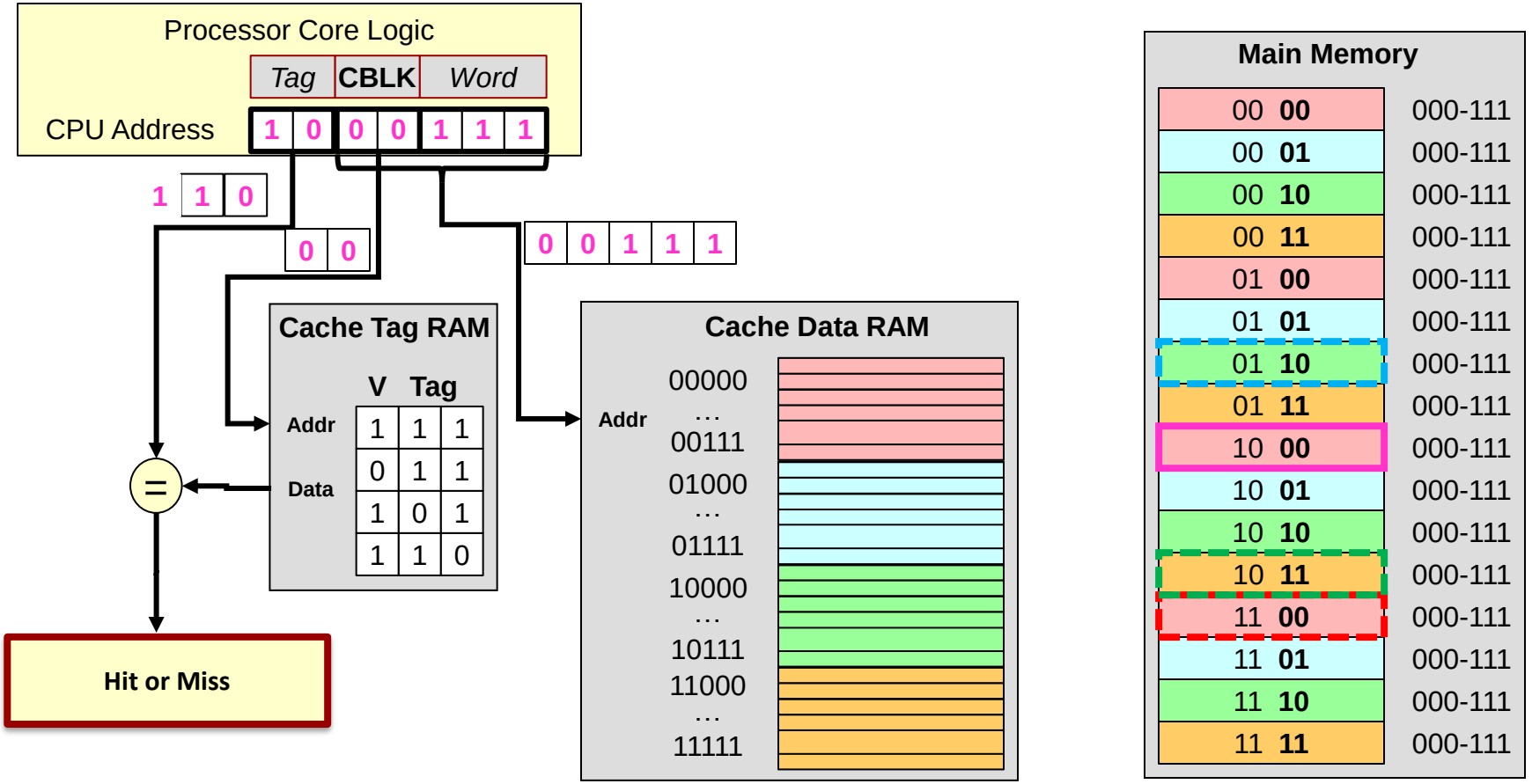
Direct Mapping Address Usage

- Cache Mapping Example:
  - Direct Mapping, MM = 128 words (2^7), Cache Size = 32 (2^5) words, Block Size = (2^3) words
- Number of blocks in MM = 2^7 / 2^3 = 2^4
- Block ID = 4 bits
- Number of Cache Block Frames = 2^5 / 2^3 = 2^2 = 4
  - Number of "colors“ => 2 Number of Block field Bits
- 2^4 / 2^2 = 2^2 = 4 Groups of blocks
  - 2 Tag Bits

Tag	**CBLK**	*Word*
2	2	3
Block ID=4		

Direct Mapping Hit Logic

- Direct Mapping Example:
  - MM = 128 words, Cache Size = 32 words, Block Size = 8 words
- Block field addresses tag RAM and compares stored tag with tag of desired address



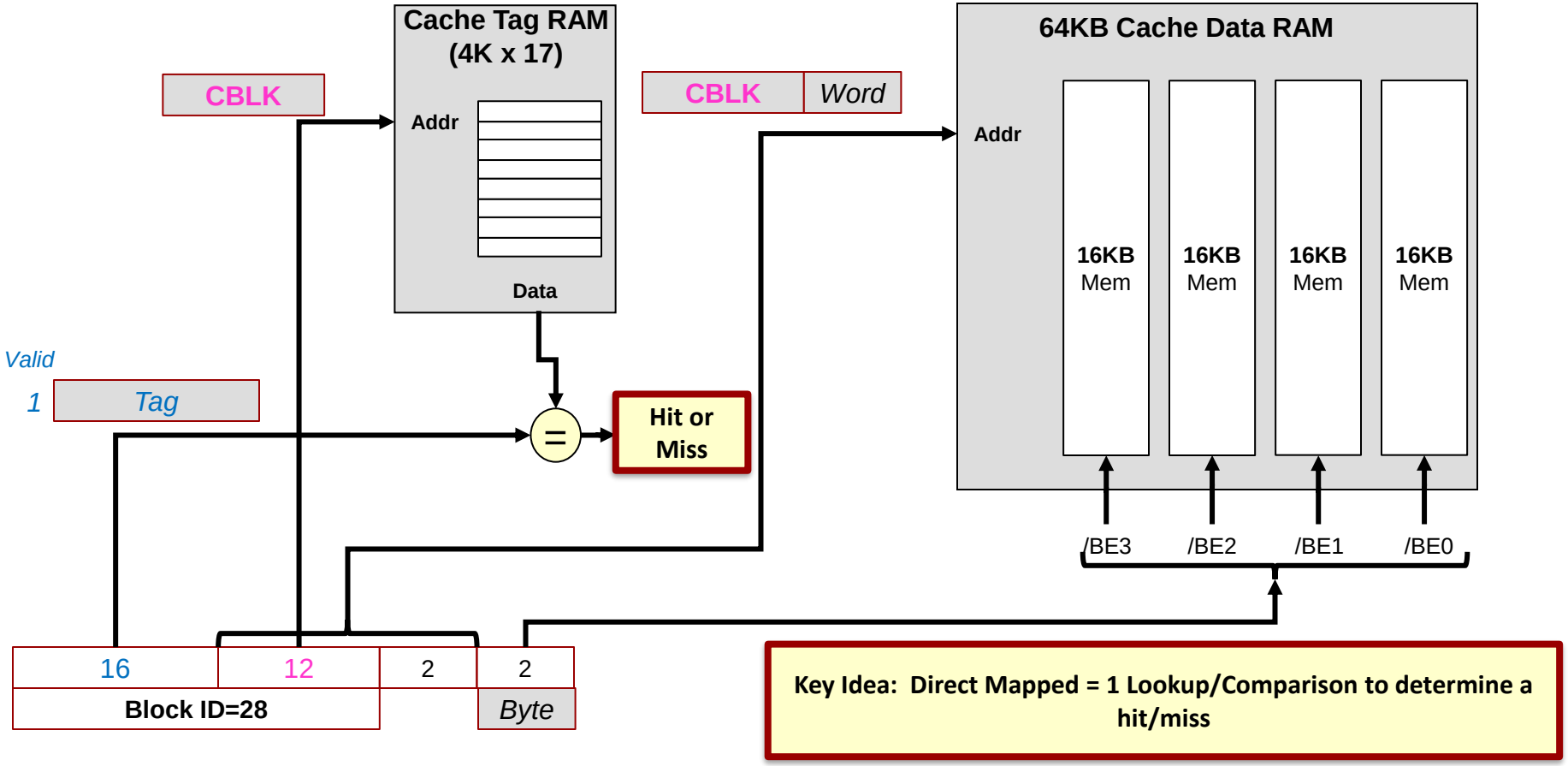
Direct Mapping Address Usage

- If 80386 used Direct Cache Mapping :
  - MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words
- Number of blocks in MM = 2^{32} / 2^4 = 2^{28}
- Number of Cache Block Frames = 2^{16} / 2^4 = 2^{12} = 4096
  - Number of "colors“ => 12 Block field bits
- 2^{28} / 2^{12} = 2^{16} = 64K Groups of blocks
  - 16 Tag Field Bits

Tag	**CBLK**	*Word*	*Byte*
16	12	2	2
Block ID=28			

Tag and Data RAM

- 80386 Direct Mapped Cache Organization



Direct Mapping Address Usage

- Divide MM and Cache into equal size blocks of B words
  - M main memory blocks, N cache blocks
  - $Log_2(B)$ word field bits
- A block in caches is often called a cache block/line frame since it can hold many possible MM blocks over time
- For direct mapping, if you have N cache frames, then define N "colors/patterns"
  - $Log_2(N)$ block field bits
- Tag = Remaining upper bits of the address ..or..
  - Repeatedly paint MM blocks with those N colors in round-robin fashion
  - M/N groups will form
    - $Log_2(M/N)$ tag field bits

Direct Mapping Datapath

- How many TAG RAM's?
  - Is that answer dependent on address field sizes?
- How many entries in the TAG RAM?

- How many bits wide is each entry in the TAG RAM?
- How many DATA RAM's?
  - What size is the address field?

Alternate Direct Mapping Scheme

- Can you "color" (i.e. map) the blocks of main memory in a different order?
- Use high-order bits as BLK field or low-order bits
- Which is more desirable or does it not really matter?

Cache

BLK 00
BLK 01
BLK 10
BLK 11

Main Memory Mapping A

0000	00	000 111
0001	01	000 111
0010	10	000 111
0011	11	000 111
0100	00	000 111
0101	01	000 111
0110	10	000 111
0111	11	000 111
1000	00	000 111
1001	01	000 111
1010	10	000 111
1011	11	000 111
1100	00	000 111
1101	01	000 111
1110	10	000 111
1111	11	000 111

Main Memory Mapping B

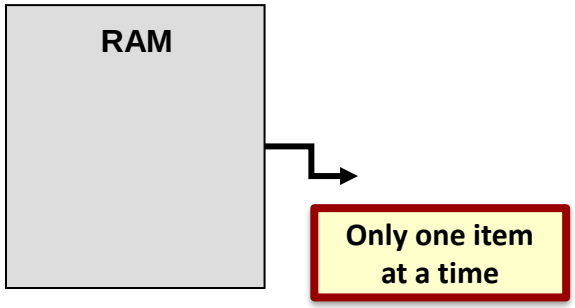
0000	00	000 111
0001	00	000 111
0010	00	000 111
0011	00	000 111
0100	01	000 111
0101	01	000 111
0110	01	000 111
0111	01	000 111
1000	10	000 111
1001	10	000 111
1010	10	000 111
1011	10	000 111
1100	11	000 111
1101	11	000 111
1110	11	000 111
1111	11	000 111

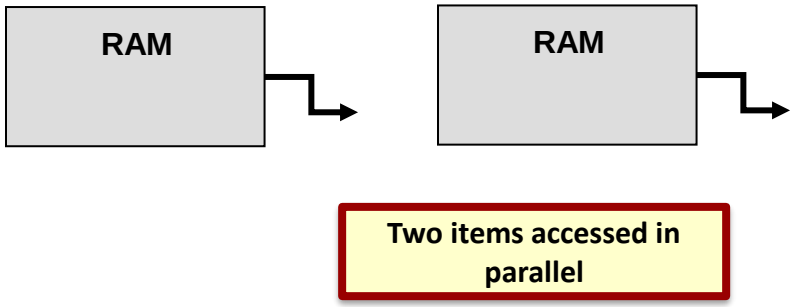
	Tag	**BLK**	*Word*
Analogy	Grp	Color	Member

	BLK	*Tag*	*Word*
Analogy	Color	Grp	Member

Single or Parallel RAM's

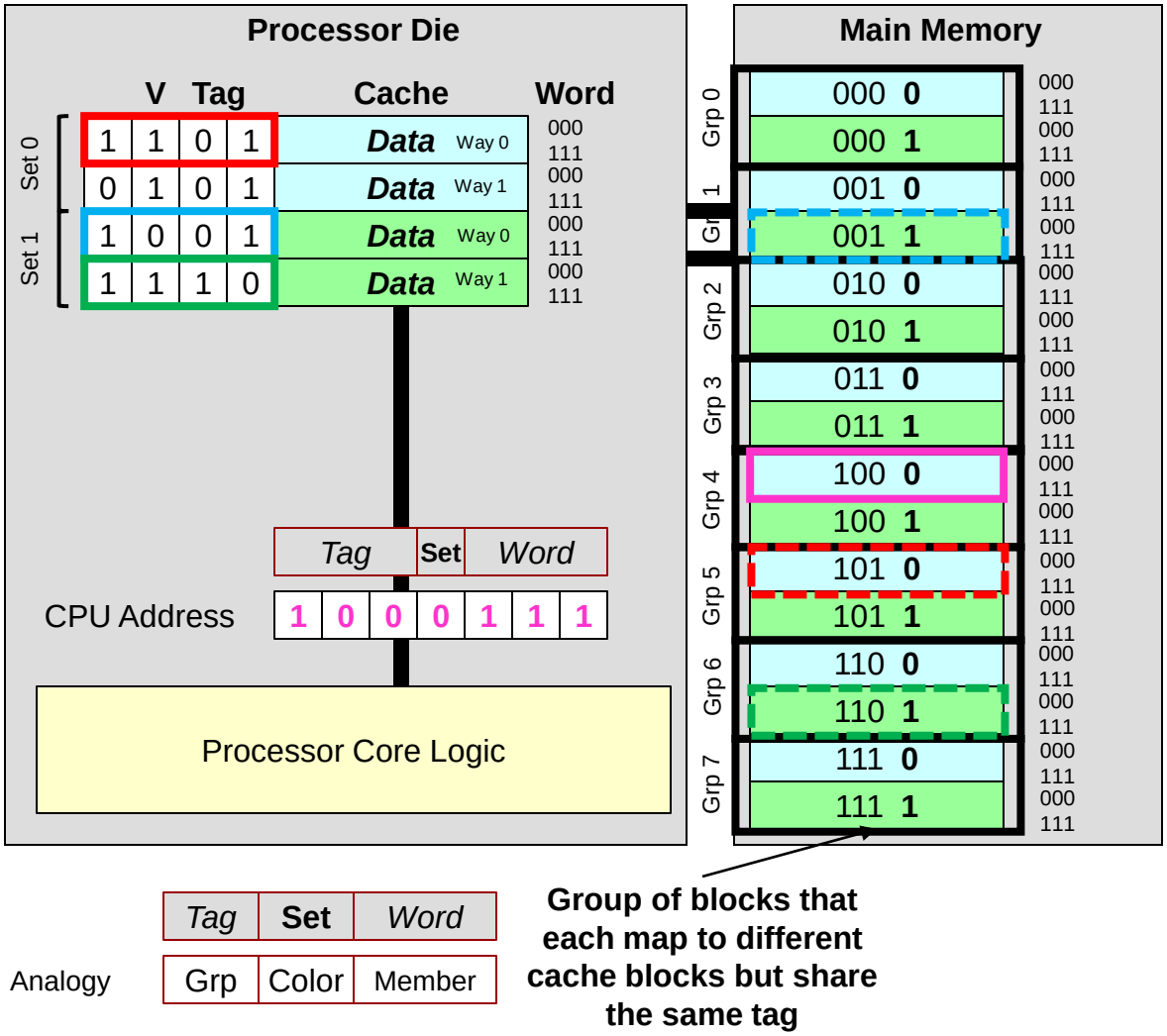
- Is it cheaper to have
  - (1) 2KB RAM
  - (2) 1KB RAM's
- Area wise a 2KB RAM occupies less area
- For tag and data RAMs it would be more economical to use fewer, big RAM's
- However, consider need for parallel access





Set-Associative Mapping Example

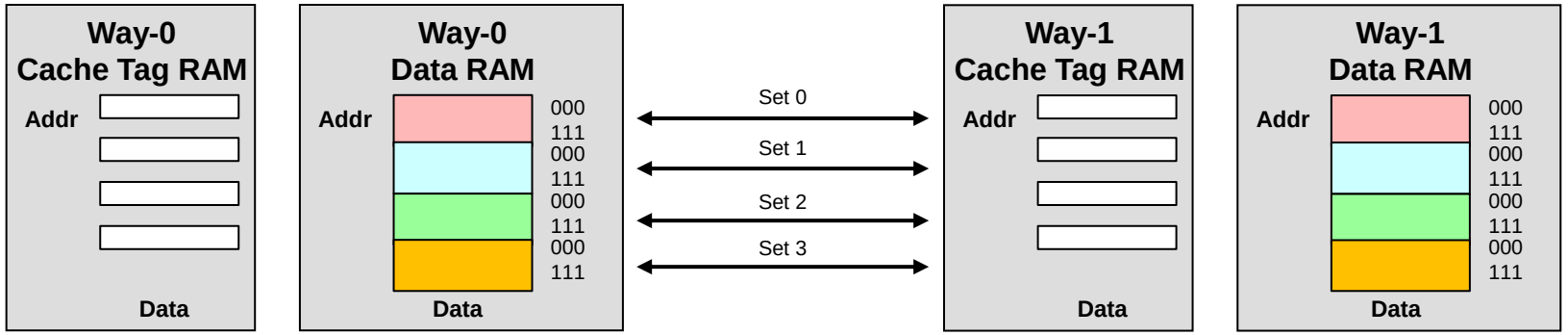
- Cache Mapping Example:
  - Direct Mapping
  - MM = 128 words
  - Cache Size = 32 words
  - Block Size = 8 words
- Each MM block i maps to Cache frame i mod S
  - S = # of sets (groups of cache frames)
  - Tag identifies which group that colored block belongs to



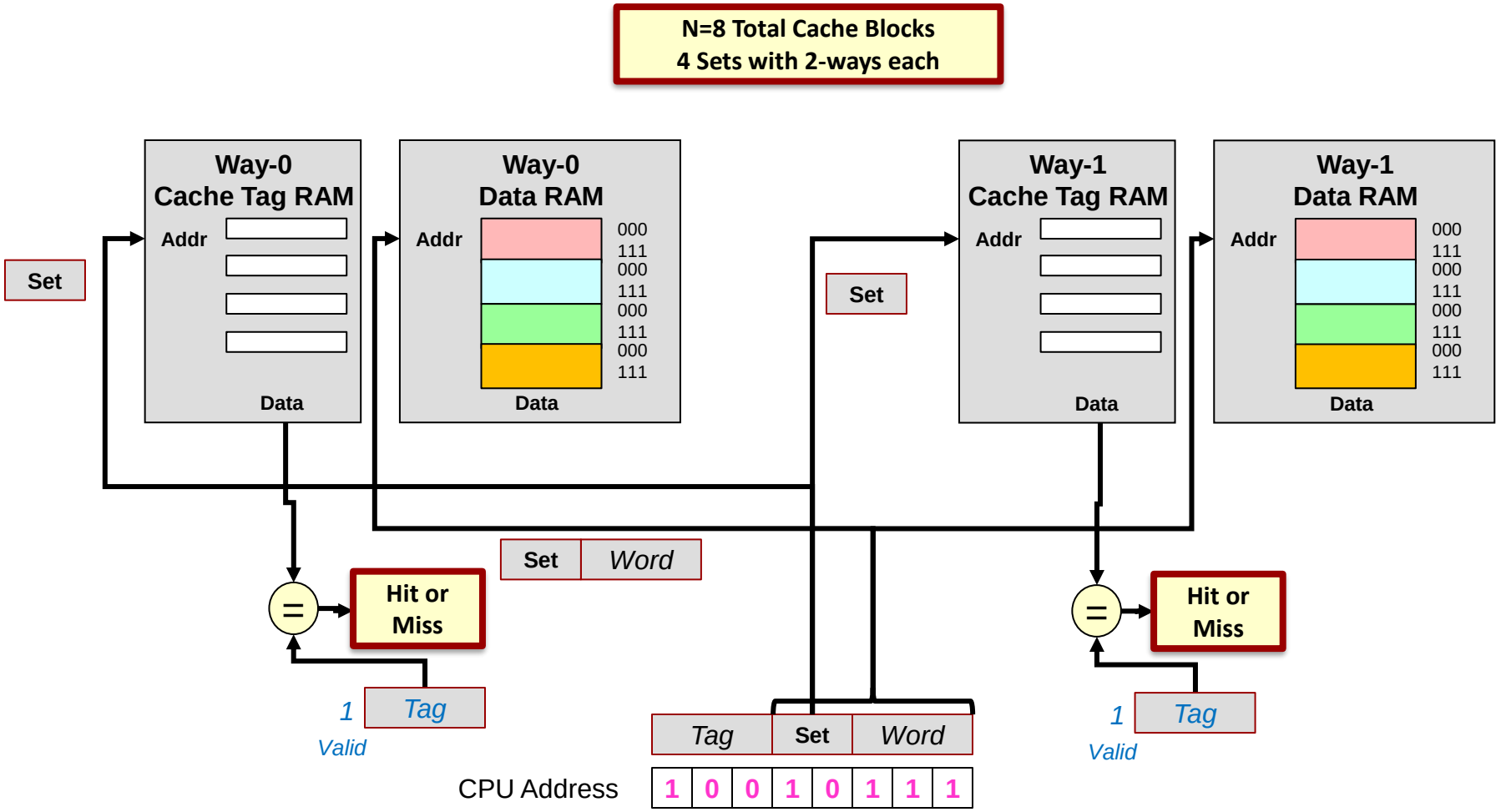
Set-Associative Datapath

N=8 Total Cache Blocks
4 Sets with 2-ways each

	V	Tag			Cache		Word
Set 0	1	1	0	1	*Data*	Way 0	000 111
	0	1	0	1	*Data*	Way 1	000 111
Set 1	1	0	0	1	*Data*	Way 0	000 111
	1	1	1	0	*Data*	Way 1	000 111
Set 2	1	1	1	1	*Data*	Way 0	000 111
	0	1	0	1	*Data*	Way 1	000 111
Set 3	0	1	0	1	*Data*	Way 0	000 111
	1	0	1	0	*Data*	Way 1	000 111



Set-Associative Datapath



Set-Associative Mapping Address Usage

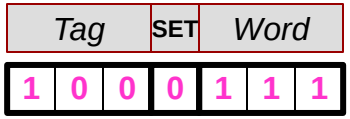
- Define K = Blocks/Set = “Ways”
- If you have N total cache frames, then define number of sets, S, = N total cache blocks / K Blocks per set
- Define S colors/patterns
  - $Log_2(S) = Log_2(N/K)$ set field bits
- Repeatedly paint MM blocks with those S colors in round-robin fashion
- M/S groups will form
  - $Log_2(M/S)$ tag field bits

Set-Associative Mapping Datapath

- How many TAG RAM's?
- How many entries in the TAG RAM?
- Place tags from different sets that belong to 'Way 0' in one tag ram, 'Way 1' in another, etc.
- How many DATA RAM's?
  - What size is the address field?

Key Idea: K-Ways => K comparators

(What is a 1-way Set Associative Mapping)



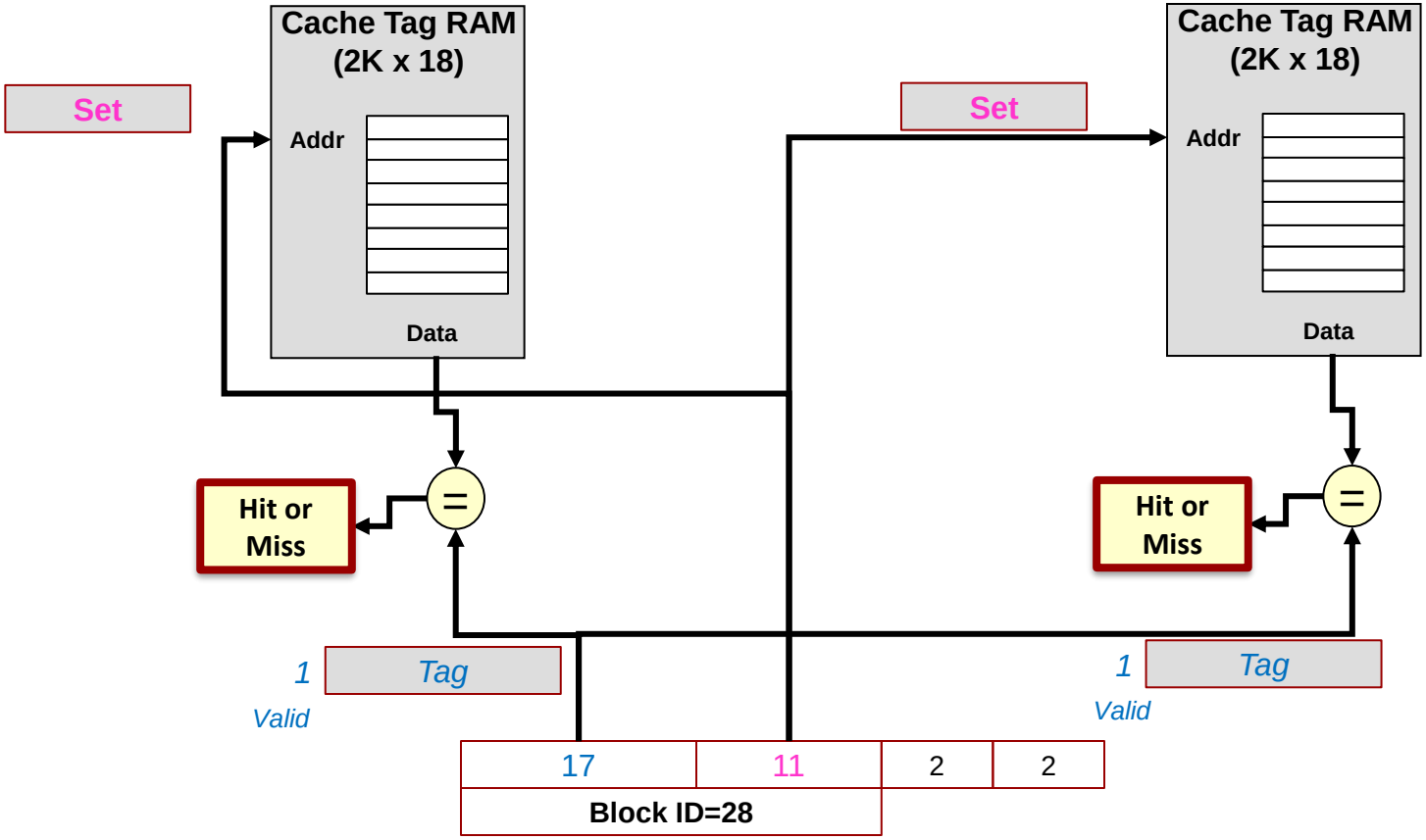
K-Way Set Associative Mapping

- If 80386 used K-Way Set-Associative Mapping:
  - MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words
- Number of blocks in MM = 2^{32} / 2^4 = 2^{28}
- Number of Cache Block Frames = 2^{16} / 2^4 = 2^{12} = 4096
- Set Associativity/Ways (K) = 2 Blocks/Set
  - Number of "colors“ => $2^{12}/2$ = 2^{11} Sets => 11 Set field bits
- 2^{28} / 2^{11} = 2^{17} = 128K Groups of blocks
  - 17 Tag Field Bits

Tag	**Set**	*Word*	*Byte*
17	11	2	2
Block ID=28			

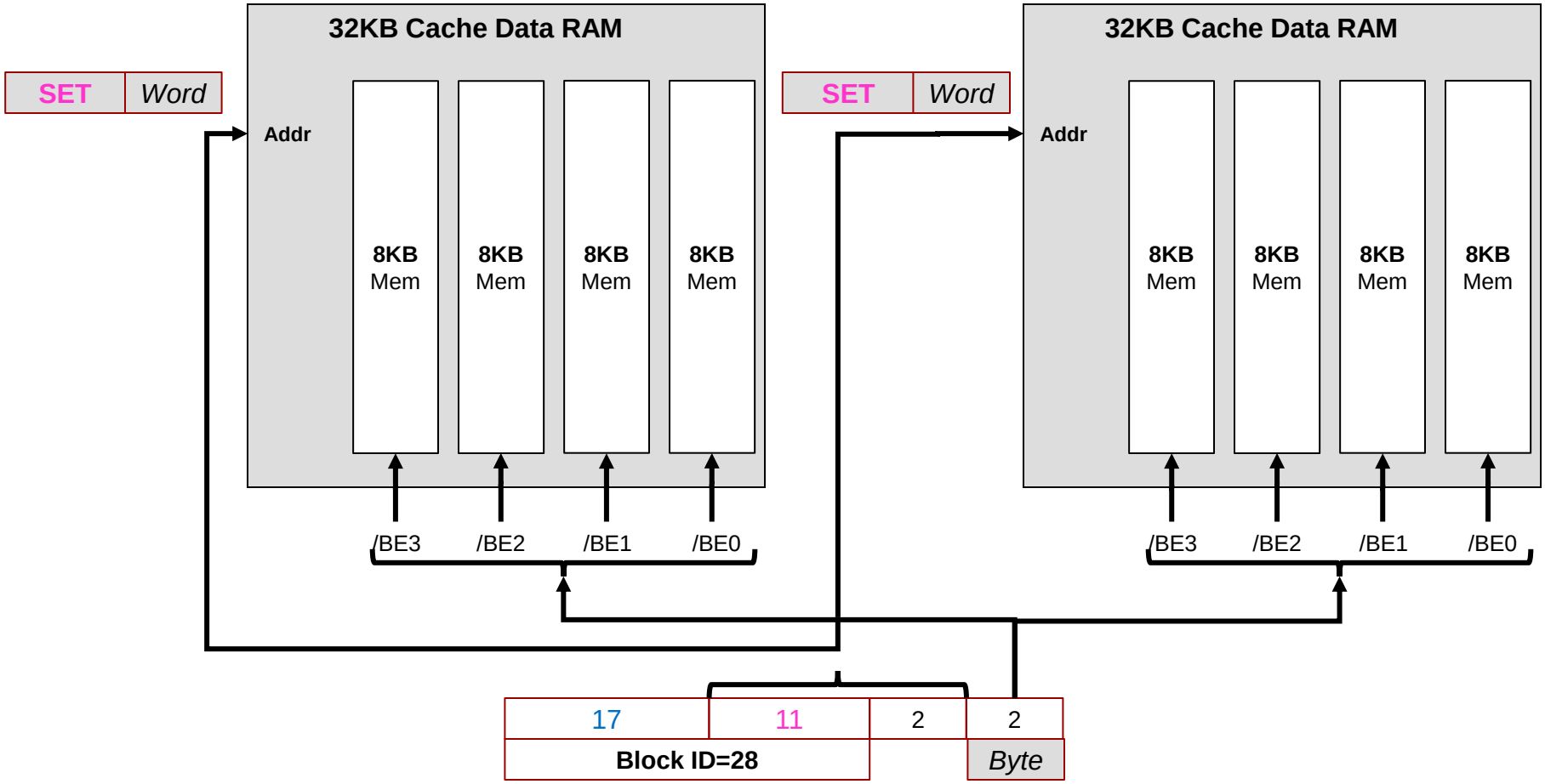
Tag RAM Organizations

- 80386 2-Way Set-Associative Cache Organization



Data RAM Organizations

- 80386 2-Way Set-Associative Cache Organization



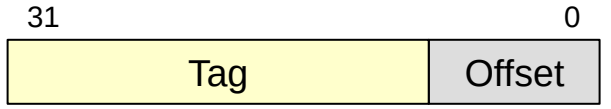
Set Associative Example

Tag	Set	Word	Byte
18	10	2	2

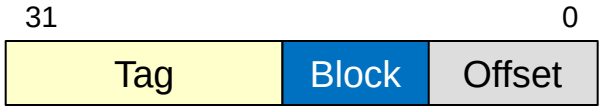
- Suppose the cache size is 2^{12} blocks
- What is the set size? 4=$2^{12}/2^{10}$ blocks/set
- If the set associativity can be changed,
  - What is the smallest set size? 1 block/set
    - Maximum # of sets = 2^{12}
    - Largest Set Field=12-bits, Smallest Tag=16-bits
    - Direct Mapping!
  - What is the largest set size? 2^{12} blocks/set
    - Minimum # of sets = 1
    - Smallest Set Field=0-bits, Largest Tag=28-bits
    - Fully Associative!

Summary of Mapping Schemes

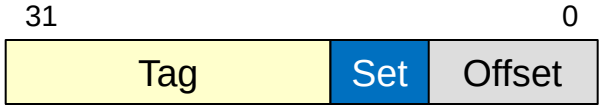
- Fully associative
  - Most flexible (less evictions)
  - Longest search time O(N)
- Direct-mapped cache
  - Least flexible (more evictions)
  - Shortest search time O(1)
  - 1 Tag RAM/comparator and 1 Data RAM
- K-way Set Associative mapping
  - Compromise
    - 1-way set associative = Direct
    - N-way set associative = Fully Assoc.
  - Work to search is O(K)
    - For small K, search in parallel: O(1)
    - K Tag RAMs/comparators and K Data RAMs



Fully Associative
No hashing…can be placed anywhere in cache. Must search N locations.



Direct Mapped Cache
h(a) = block field
Only search 1 location.



K-way Set Associative Mapping
h(a) = set field
Only search k locations

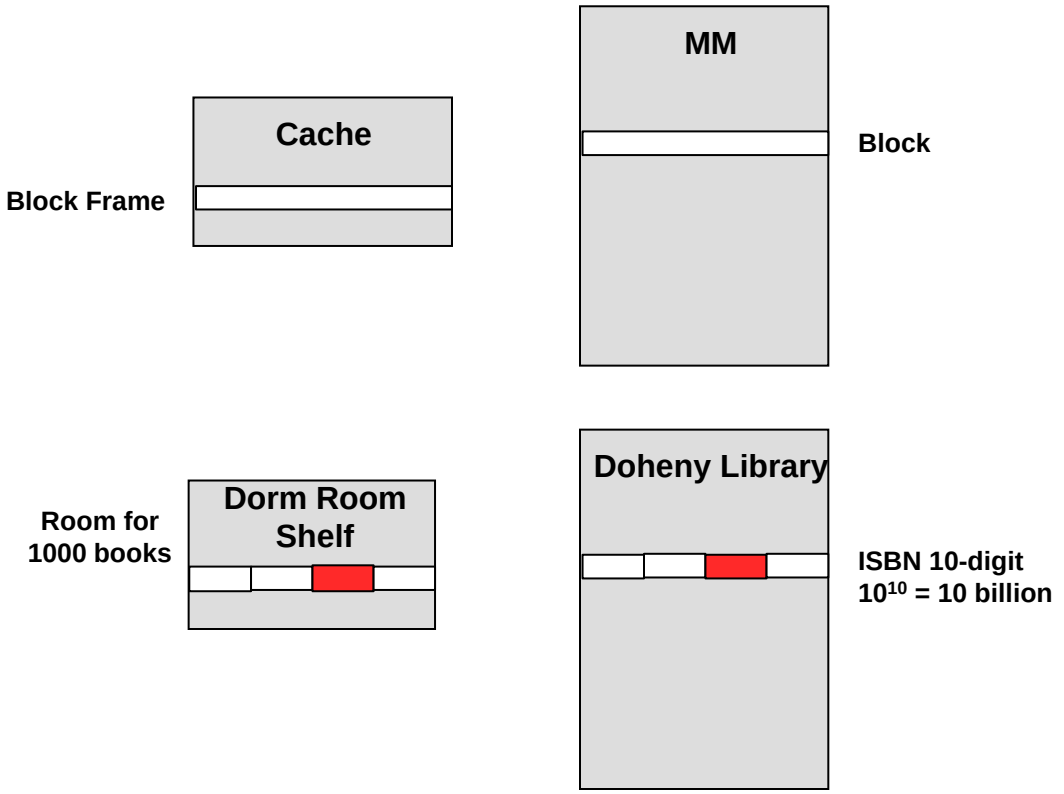
LIBRARY ANALOGY

Mapping Functions

- A mapping function determines the correspondence between MM blocks and cache block frames
- 3 Schemes
  - Fully Associative
  - Direct Mapping
  - Set-Associative
- Really just 1 scheme
  - Fully Associative = N-way Set Associative
  - Direct Mapping = 1-way Set Associative

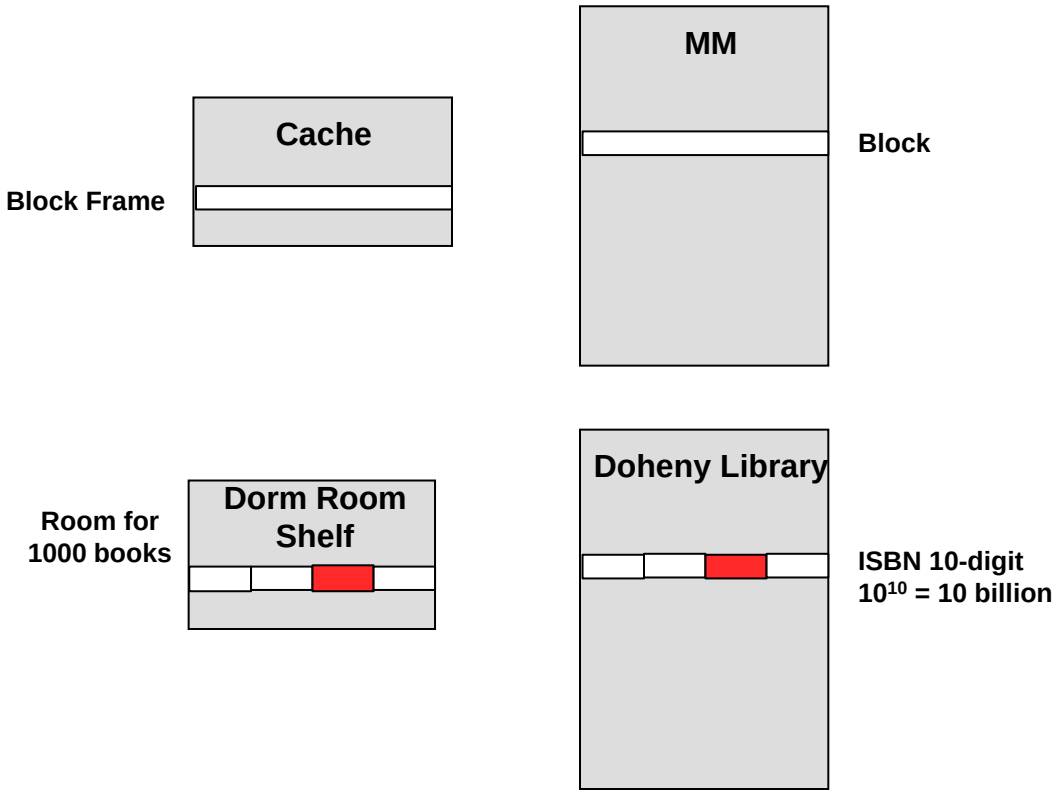
Library ⇔ Memory

- Compare MM to a large library
- Compare cache to your dorm room book shelf
- "Address" of a book = 10-digit ISBN number
- Assume library has a location on the shelf for all 10^{10} possible books



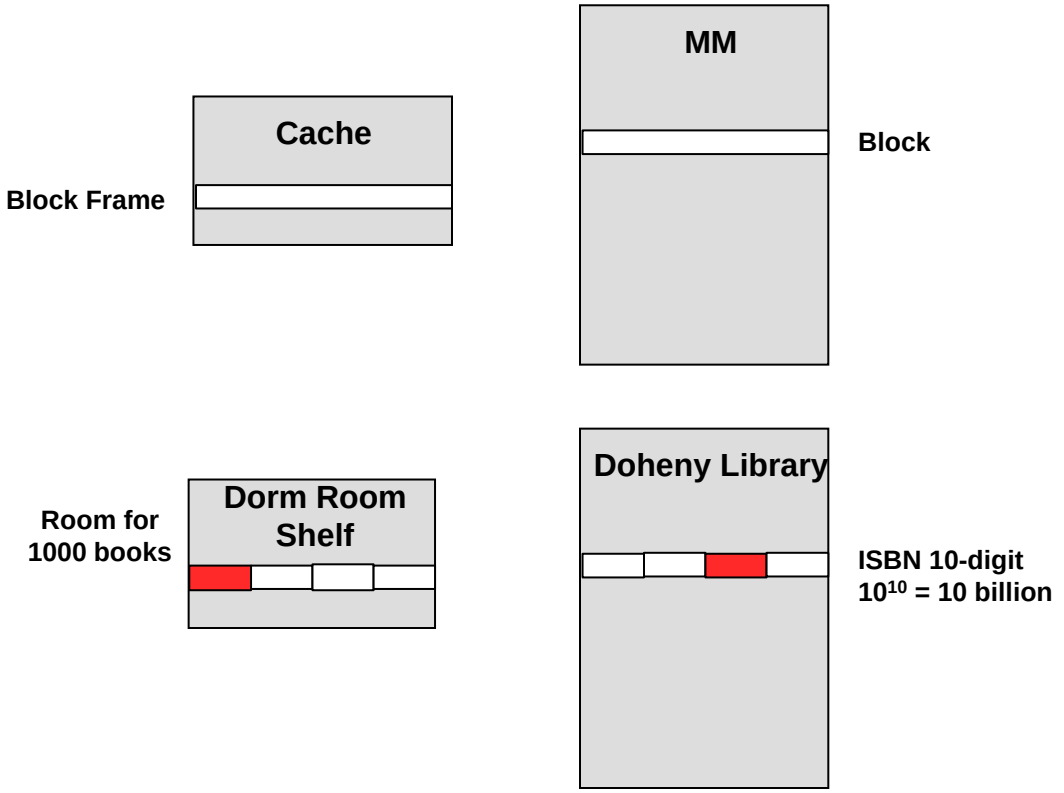
Book Addressing

- Addresses are not stored in memory (only data)
- Assume library has a location on the shelf for all 10^{10} possible books
- No need to print ISBN on the book if each book has a location (find a book by going to its slot using ISBN as index)



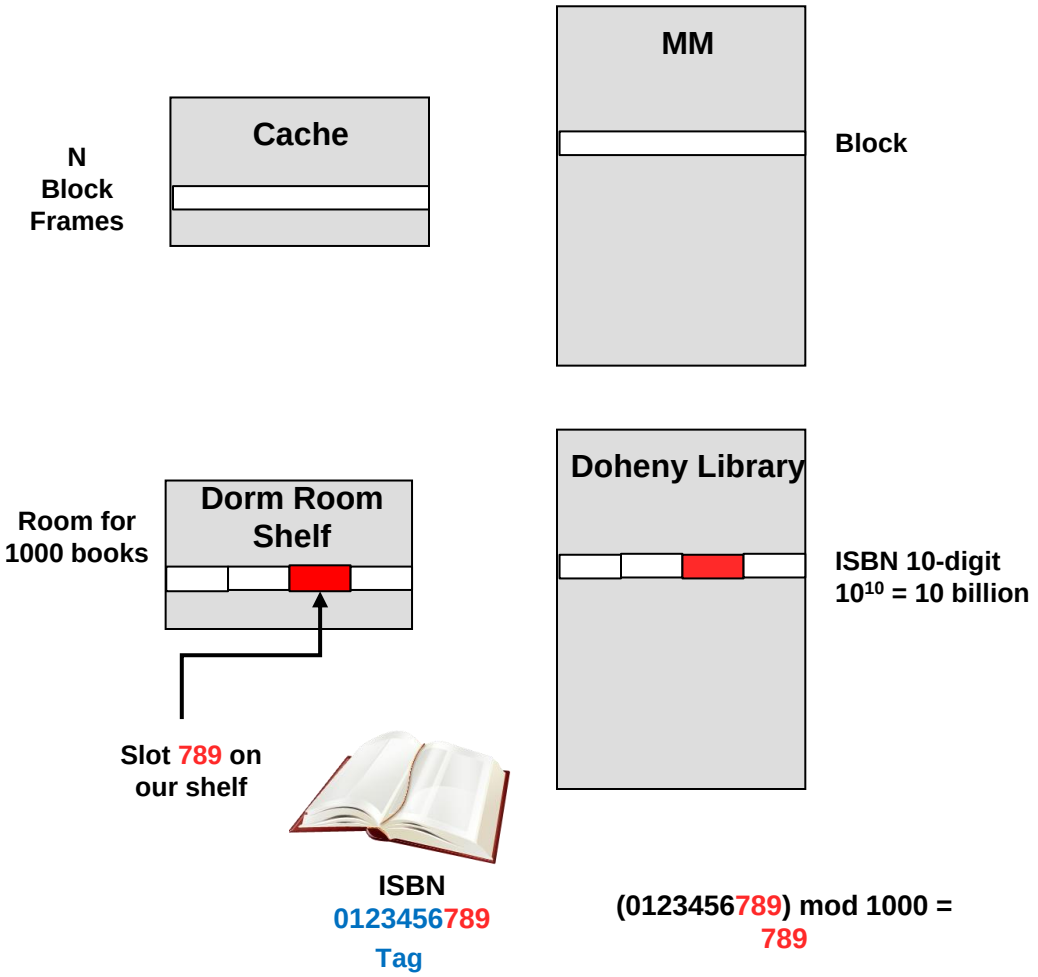
Fully Associative Analogy

- Cache stores full Block-ID as a TAG to identify that block
- When we check a book out and take it to our dorm room shelf…
  - Let's allow it to be put in any free slot on the shelf
  - We need to keep the entire ISBN number as a TAG
- To find a book with a given ISBN on our shelf, we must look through them all



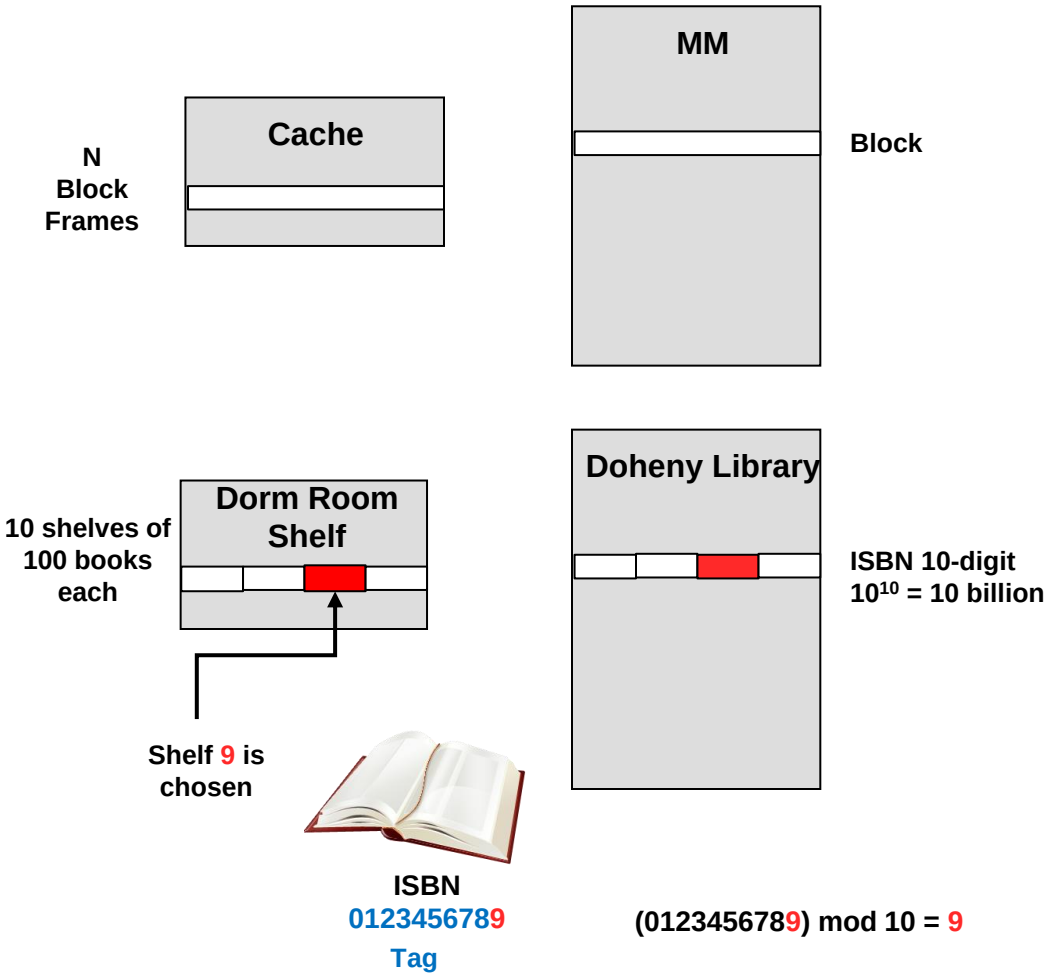
Direct Mapping Analogy

- Cache uses block field to identify the slot in the cache and then stores remainder as TAG to identify that block from others that also map to that slot
- Assume we number the slots on our book shelf from 0 to 999
- When we check a book out and take it to our dorm room shelf we can…
  - Use last 3-digits of ISBN to pick the slot to store it
  - If another book is their, take it back to Doheny library (evict it)
  - Store upper 7 digits to identify this book from others that end with the same 3-digits
- To find a book with a given ISBN on our shelf, we use the last 3-digits to choose which slot to look in and then compare the upper 7-digits



Set Associative Mapping Analogy

- Cache blocks are divided into groups known as sets. Each MM block is mapped to a particular set but can be anywhere in the set (i.e. all TAGS in the set must be compared)
- Assume our bookshelf is 10 shelves with room for 100 books each
- When we check a book out and take it to our dorm room shelf we can…
  - Use last 1-digit of ISBN to pick the shelf but store the book anywhere on the shelf where there is an empty slot
  - Only if the shelf is full do we have to pick a book to take back to Doheny library (evict it)
  - Store upper 9 digits to identify this book from others that end with the same 1-digit
- To find a book with a given ISBN on our shelf, we use the last
1-digits to choose which shelf to look in and then compare upper 9-digits with those of all the books on the shelf



Set Associative Mapping Analogy

- Can we confidently say,
  - We can bring in any (10/100/other) book(s)
  - We can bring in (10/100/other) consecutive book(s)
- Library analogy:
  - 10 sets each with 100 slots = 100-way set associative cache