Chapter 5
Large and Fast : Exploiting Memory Hierarchy

Computer Architecture and Organization

School of CSEE

Memory System and System Performance

- **Because most Programs spend much of their time accessing memory, the memory system is necessarily the major factor in determining system performance!**

1. **Memory hierarchy general**
   - Large vs. Fast
   - Principle of locality
2. **Basics of cache**
3. **Improving cache performance**
4. **Virtual memory**

Introduction

- **We will discuss how memory hierarchy is constructed in memory system.**
- **We will also discuss why memory hierarchy can enhance the performance of the computer.**

Large vs. Fast memory

- There is a conflict having large and fast memory.
- Large memory is slow and fast memory is small.
- Recall that ‘smaller is faster’.
- But, users wants large and fast memory.

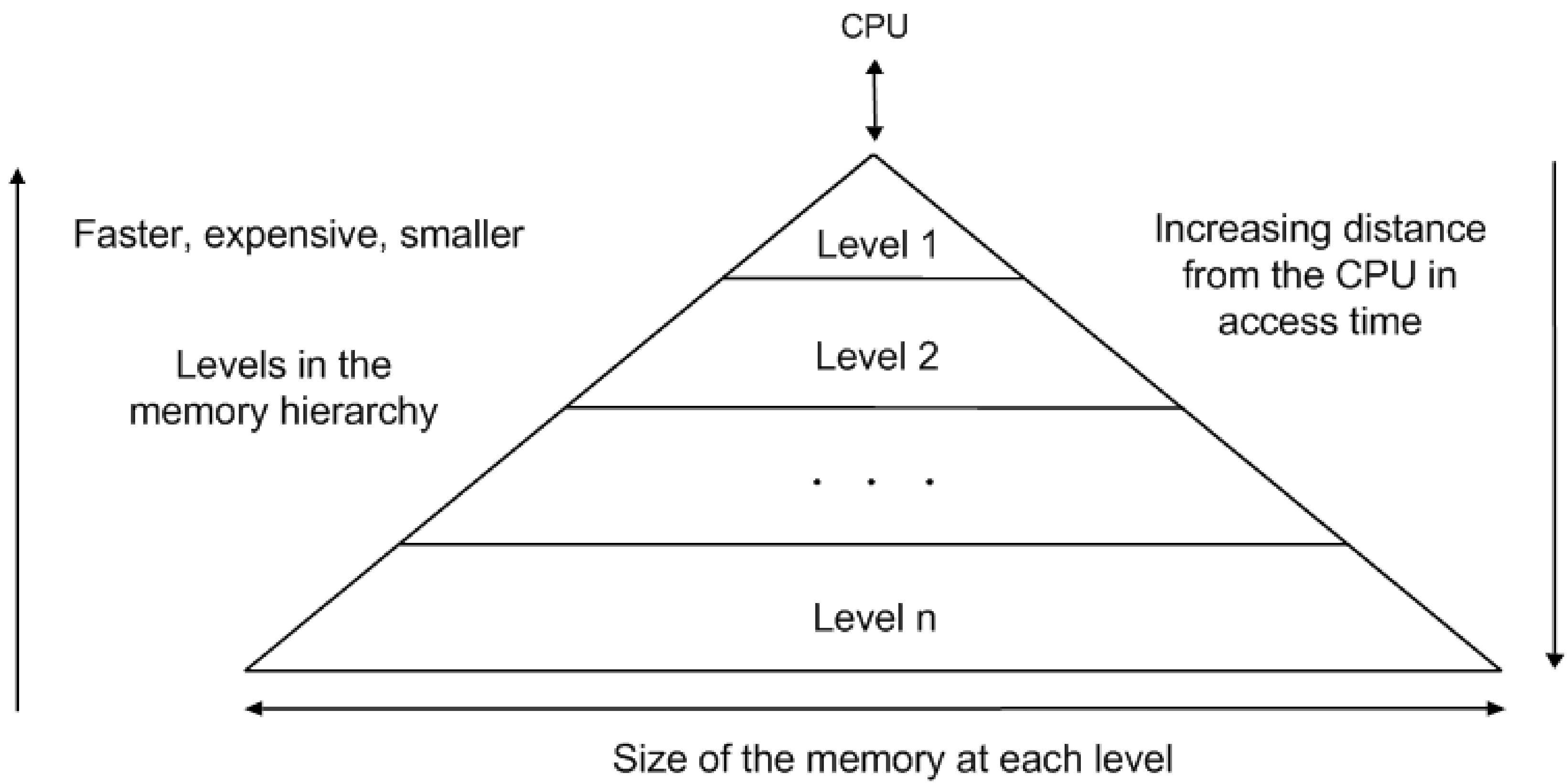
SRAM access times are 0.5ns – 2.5ns at cost of $2000 – $5000 per GB

DRAM access times are 50ns – 70ns at cost of $20 – $75 per GB

Disk access times are 5ms – 20ms at cost of $0.20 – $2 per GB

- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Solution) Build a memory Hierarchy



Q) Does memory hierarchy work?

Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  temporal locality: the same item will tend to be referenced again soon

  ex) a = b + c;

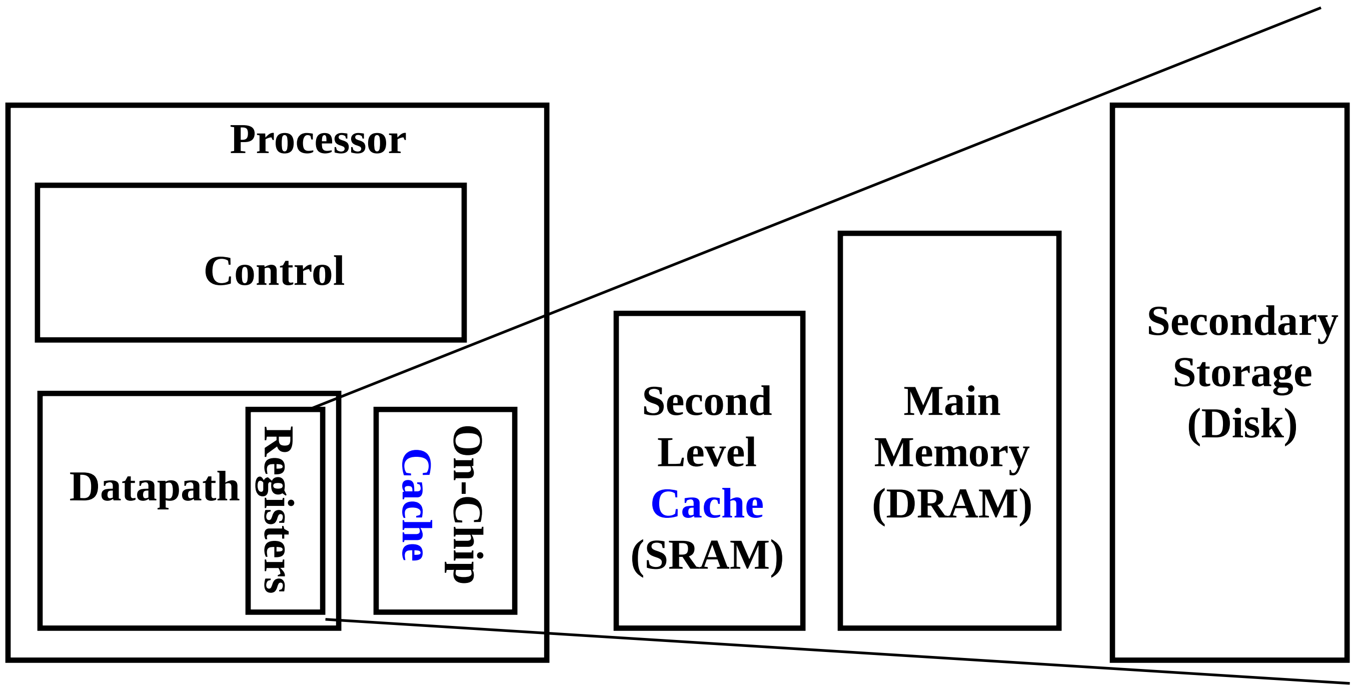
  d = 2*a + 1;

  spatial locality: nearby items will tend to be referenced soon.

  ex) for (i=0; i<10; i++) sum = sum + a[i];

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology

Memory Hierarchy of a Modern Computer System



Taking Advantage of Locality

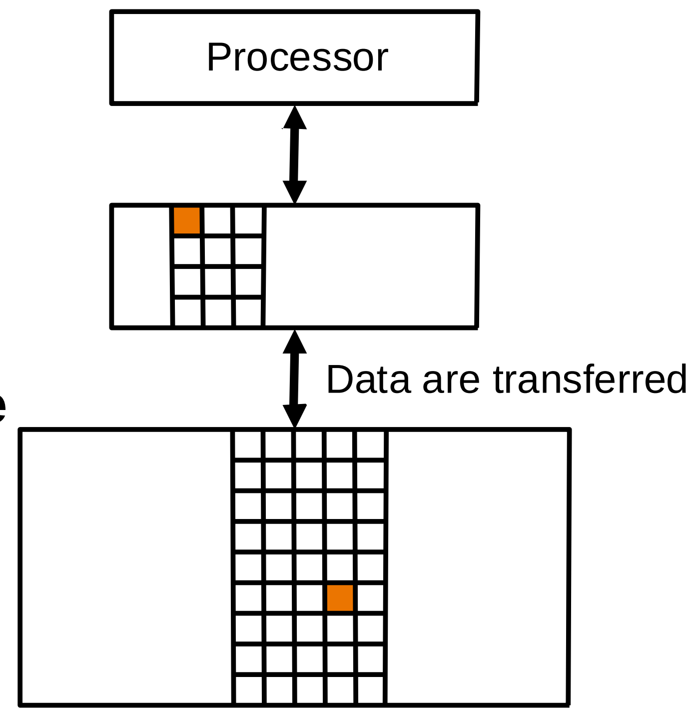
- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Basic Structure of a Memory Hierarchy

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data (several words in cache memory)
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

    In case of miss, the block that contains the requested data should be loaded into upper memory. ➔ CPU stalls ➔ very time consuming process ➔ performance degrade

    : Principle of locality justifies memory hierarchy
  - each pair of levels in the memory hierarchy can be thought of as having an upper and lower level



Summary

- To achieve ‘large’ and ‘fast’ memory for CPU, modern computer system has memory hierarchy.
- Smaller, faster, and expensive memory is located nearer to CPU.
- The reason why memory hierarchy works is that there is locality – spatial and temporal – in the program. This is called principle of locality.

1. **Memory hierarchy general**

2. **Basics of cache**

   - Direct mapped cache operation
   - Handling cache misses
   - Write policy

3. **Improving cache performance**

4. **Virtual memory**

Introduction

- In this section we will see how cache memory operate in ‘read’ and ‘write’ operation.
- Also we will discuss two different write policies.

Cache

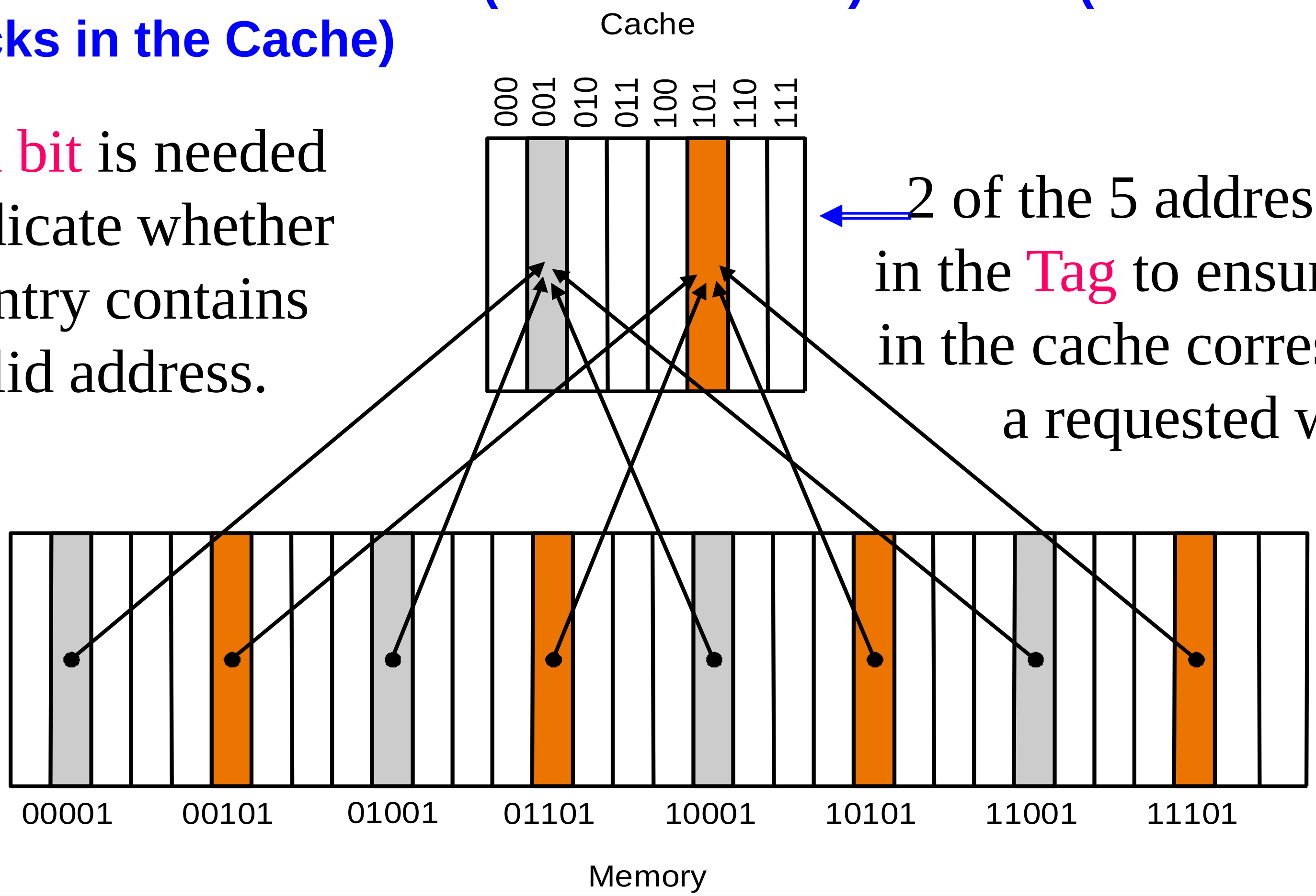
- **Two issues:**
  - **How do we know if a data item is in the cache?**
  - **If it is, how do we find it?**
  - **“How do we know if David is in the library?” & “ If David is in the library, how do we find him?”**
- **Our first example:**
  - **block size is one word of data**
  - **"direct mapped“ : For each item of data at the lower level, there is exactly one location in the cache where it might be. e.g., lots of items at the lower level share locations in the upper level**

Direct Mapped Cache

- **Mapping: address is modulo the number of blocks in the cache**

Location in the Cache = (Block address) modulo (Number of Cache Blocks in the Cache)

Valid bit is needed to indicate whether an entry contains valid address.



2 of the 5 address bits are in the Tag to ensure the data in the cache corresponds to a requested word.

An Example

1. 10110
2. 11010
3. 10000
4. 00011
5. 10010

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

An Example

1. 10110
2. 11010
3. 10000
4. 00011
5. 10010

Index	V	Tag	Data
0 0 0			
0 0 1			
0 1 0			
0 1 1			
1 0 0			
1 0 1			
1 1 0			
1 1 1			

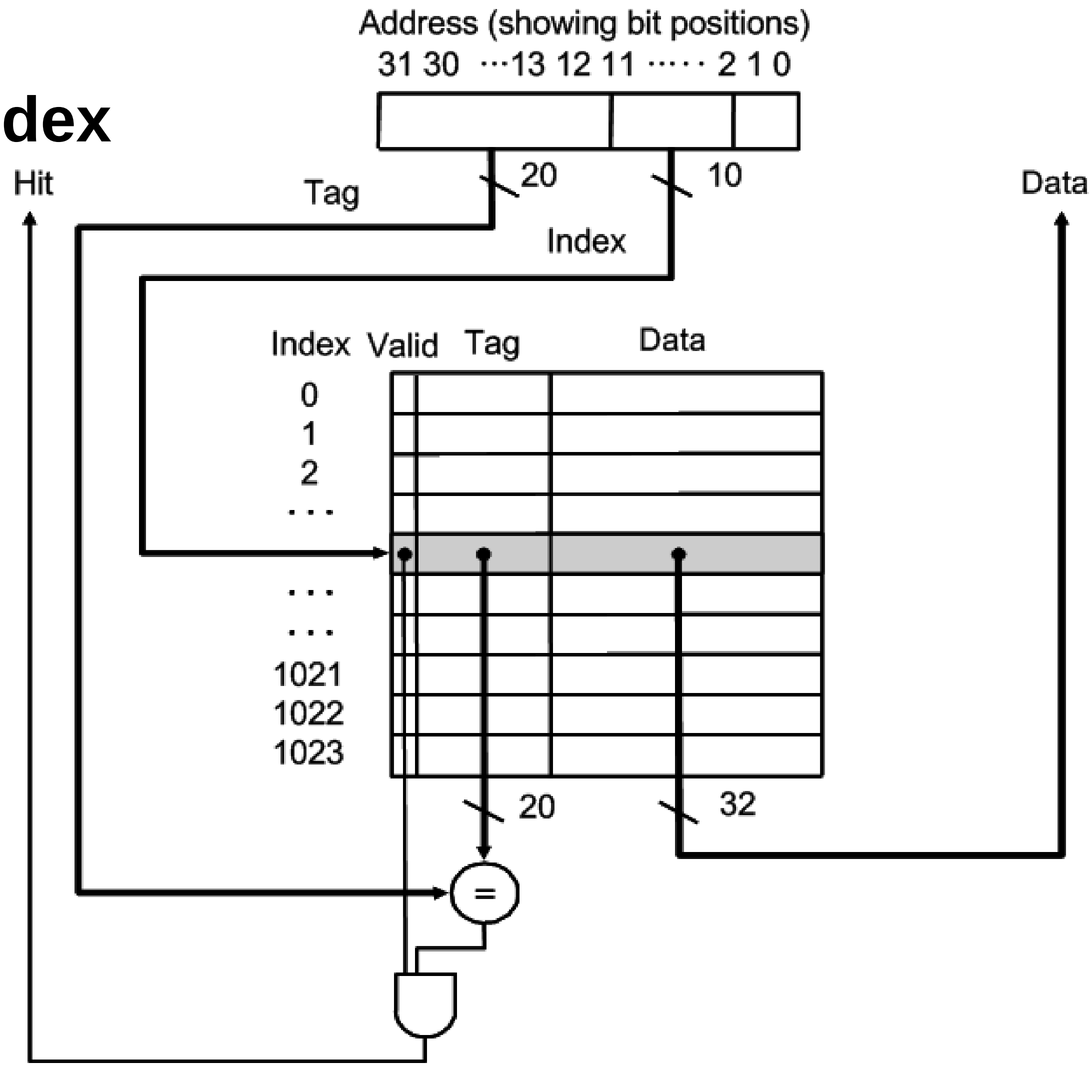
Direct Mapped Cache Example (MIPS)

•Cache has 1024 words
→10 bits are used as index (location in the cache)

•each block is
1 word (4 bytes)

•Tag Size :
32 – 10 – 2 = 20 bits

•1 bit of valid bit



Direct Mapped Cache Example (MIPS)

- **Cache has 1024 words**

➔ **10 bits are used as index (location in the cache)**

- **each block is 1 word (4 bytes)**
- **Tag Size : 32 – 10 – 2 = 20 bits**
- **1 bit of valid bit**
- **total number of bits in cache = 1024 * (32 + 20 + 1)**

or

$2^n * (32 + (32 - n - 2) + 1)$ for cache with *n* bits of index and block size of 1 word (4 bytes)

Hits vs. Misses

- **Read Hits:**
  - this is what we want!
- **Read Misses:**
  - stall the CPU, fetch block from memory, deliver to cache, restart

**** The Basic Approach to Cache Miss ****

- **Stall** the CPU, freezing the contents of all the registers while **waiting for memory**
- **A separate controller** handles the cache miss, **fetching the data** into cache **from memory**
- Once the data is present, **Restart** the execution

Handling Cache Misses

The instruction memory and data memory in chapter 4 may be replaced by instruction cache and data cache

- **Closer Look at the Read Miss**

[for instruction cache]

1. **Send PC-4 to memory**
2. **Read block from memory, CPU stalls.**
3. **Write the cache entry (data + tag) and set valid bit on**
4. **Restart the instruction execution (refetch instruction)**

[for data cache]

Step 2, 3, and 4

Q : What about the old block which was in cache?

Do I store it in memory? OR Do I throw them away?

What Happens on Write ?

- ***Write through*** **: the information is written to both the block in the cache and to the block in the main memory.**

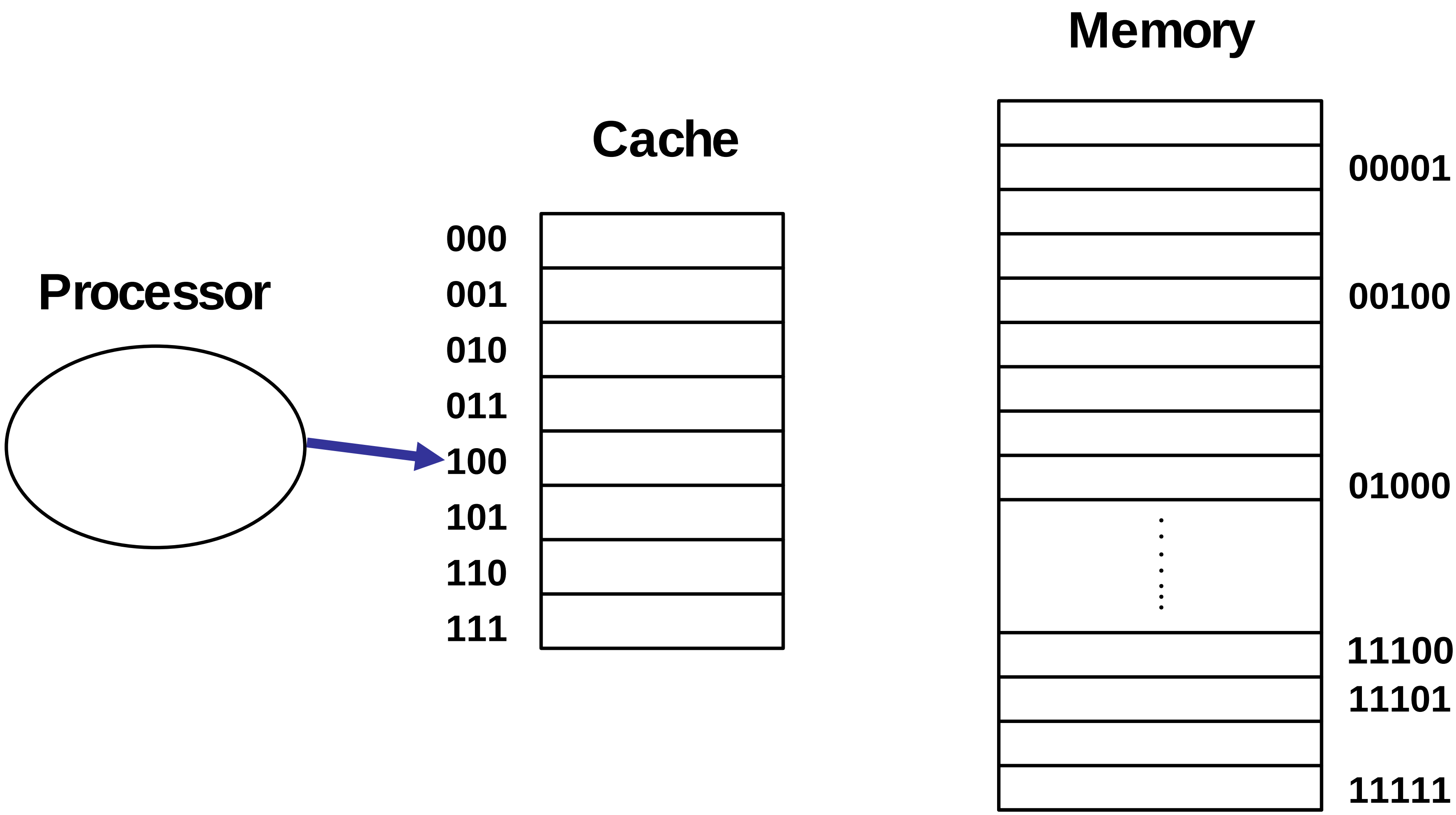
In case of 'miss', the old block does not need to be saved.

*** Faster processing in case of 'miss'.**

*** Whenever data cache is updated, there should a memory write.**

➔ More memory access

Write through



Write back

- ***Write back*** **: the information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.**
  - **Is block clean or dirty?**

Dirty : Dirty bit = 1

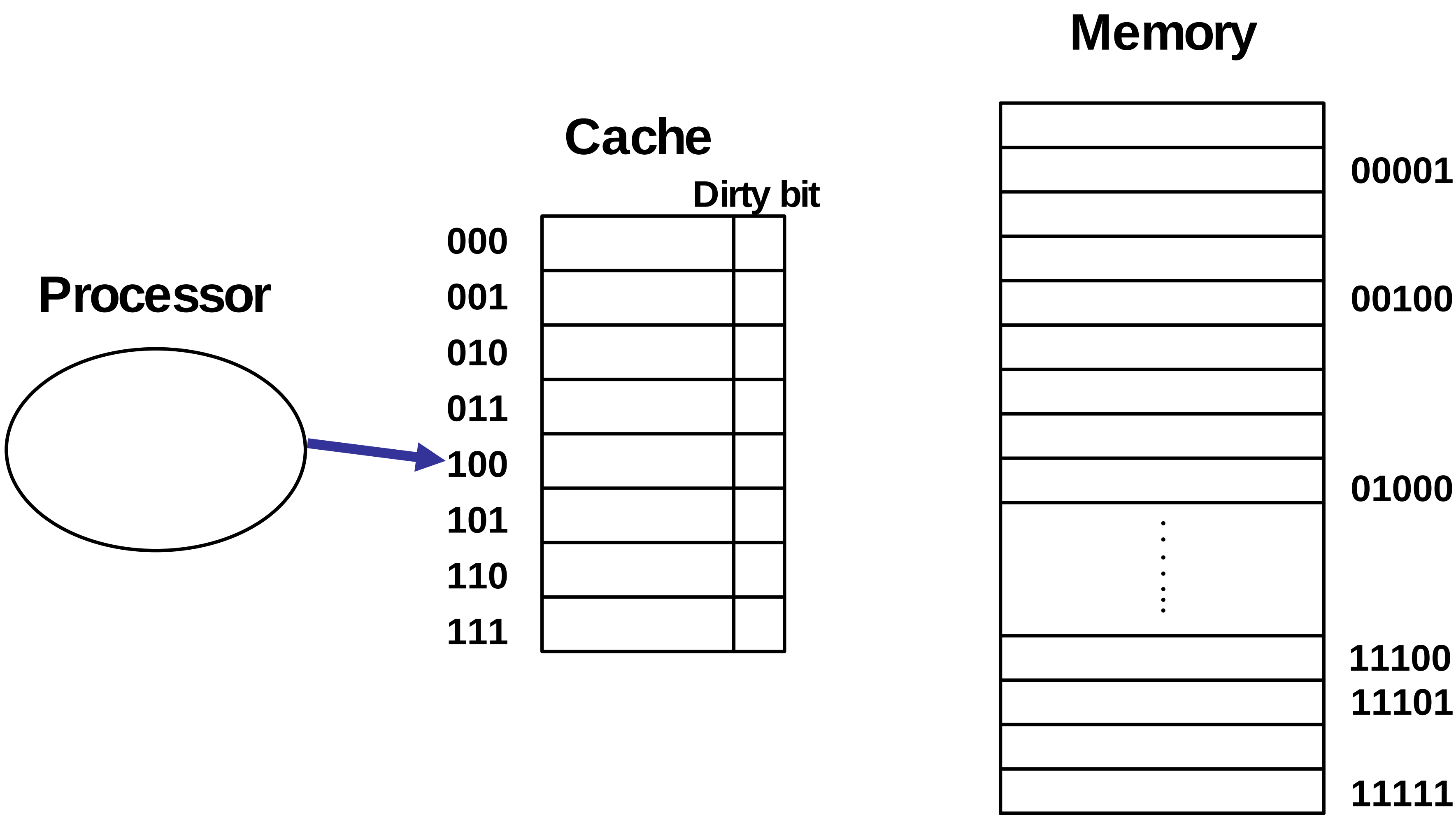
Cache block was updated after it is loaded into cache.

The old block should be written into memory when it is replaced.

Clean : Dirty Bit = 0

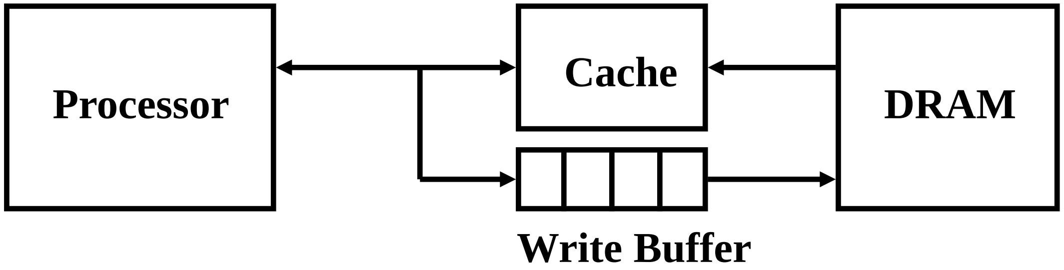
Cache block was not updated after it is loaded into cache.

Write back


Memory
Cache
Dirty bit
Processor
000
001
010
011
100
101
110
111
00001
00100
01000
11100
11101
11111

What Happens on Write ?

- Pros (and Cons) of each?
  - W-T: Read misses does not result in writes to memory
  - W-B: No repeated writes to memory
- W-T is always combined with write buffers so that we don’t wait for lower level memory write.
- A Write Buffer is located between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory



Hits vs. Misses

- **Read Hits:**
  - this is what we want!
- **Read Misses:**
  - stall the CPU, fetch block from memory, deliver to cache, restart
- **Write Hits: (only to data cache)**
  - Write the data both in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- **Write Misses: (only to data cache)**
  - Same as read miss

Conclusion

- We discussed how cache memory works in direct mapped cache system.
- We considered how the cache memory system operates in four cases – read hit, read miss, write hit, and write miss.
- We also discussed two write policies – write through and write back. Each has its own advantage and disadvantge.

1. Memory hierarchy general

2. Basics of cache

3. Improving cache performance

- Block size
- interleaving
- Three placement policies
- replacement alg.
- multi-level cache

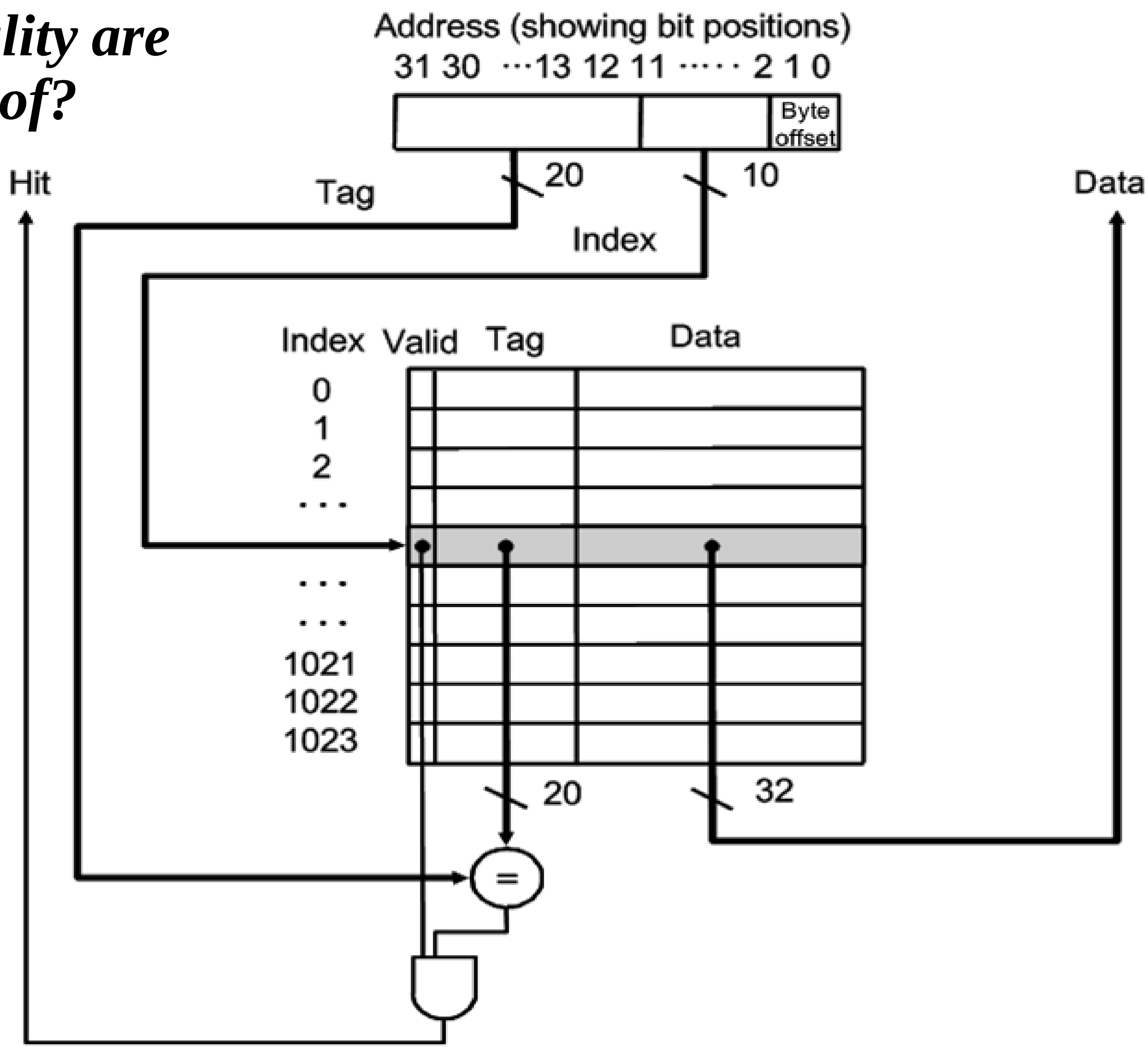
4. Virtual memory

Introduction

- **In this section we will discuss various ways to improve cache memory system.**
- **We will also discuss cache placement policy and cache replacement policy.**

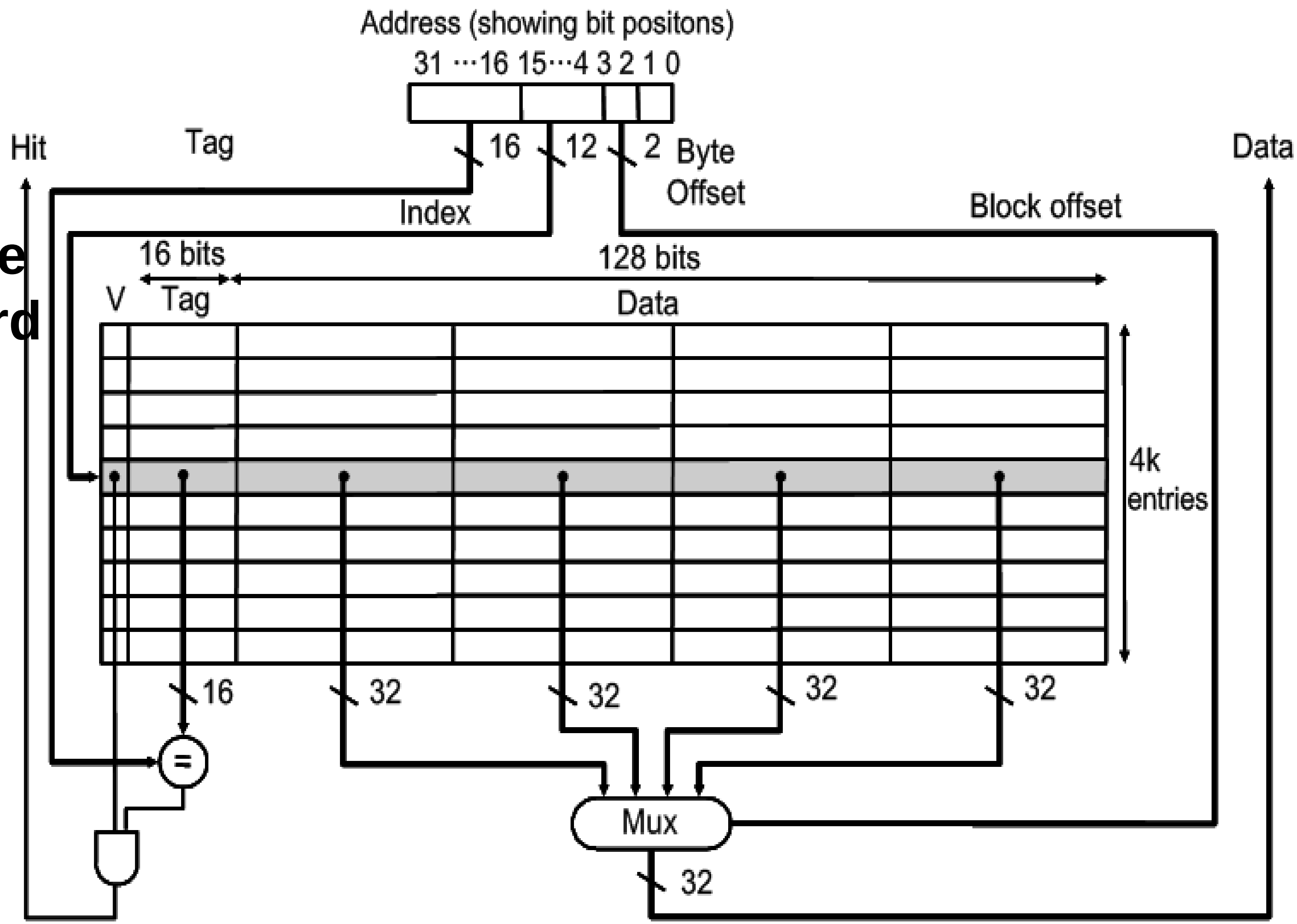
Direct Mapped Cache Example (MIPS)

- **Q)** ***What kind of locality are we taking advantage of?***



Multiple Word Direct Mapped Cache

64KB cache using 4-word blocks



Multiple Word Direct Mapped Cache

- Taking advantage of spatial locality also

  ➔ have several words in one block

  ➔ each words in same block share Tag & Valid

- In case of miss : brings entire block

Performance

- **Simplified model:**

  execution time

  = (execution clock cycles + stall clock cycles) $*$ cycle time

  stall cycles = # of instructions $*$ miss ratio $*$ miss penalty

- **miss penalty : the time to replace a block in cache with the corresponding block from the Memory + the time to deliver this block (or word) to the processor**
- **Note that : hit time << miss penalty**
- **Two ways of improving performance:**
  - **decreasing the miss ratio (== increasing the hit ratio)**
  - **decreasing the miss penalty**

What happens if we increase block size? : miss ratio, miss penalty

Performance

- **Increasing the Block Size tends to decrease Miss Ratio:**
- **But, the miss ratio goes up if the block size becomes a significant fraction of the cache size**

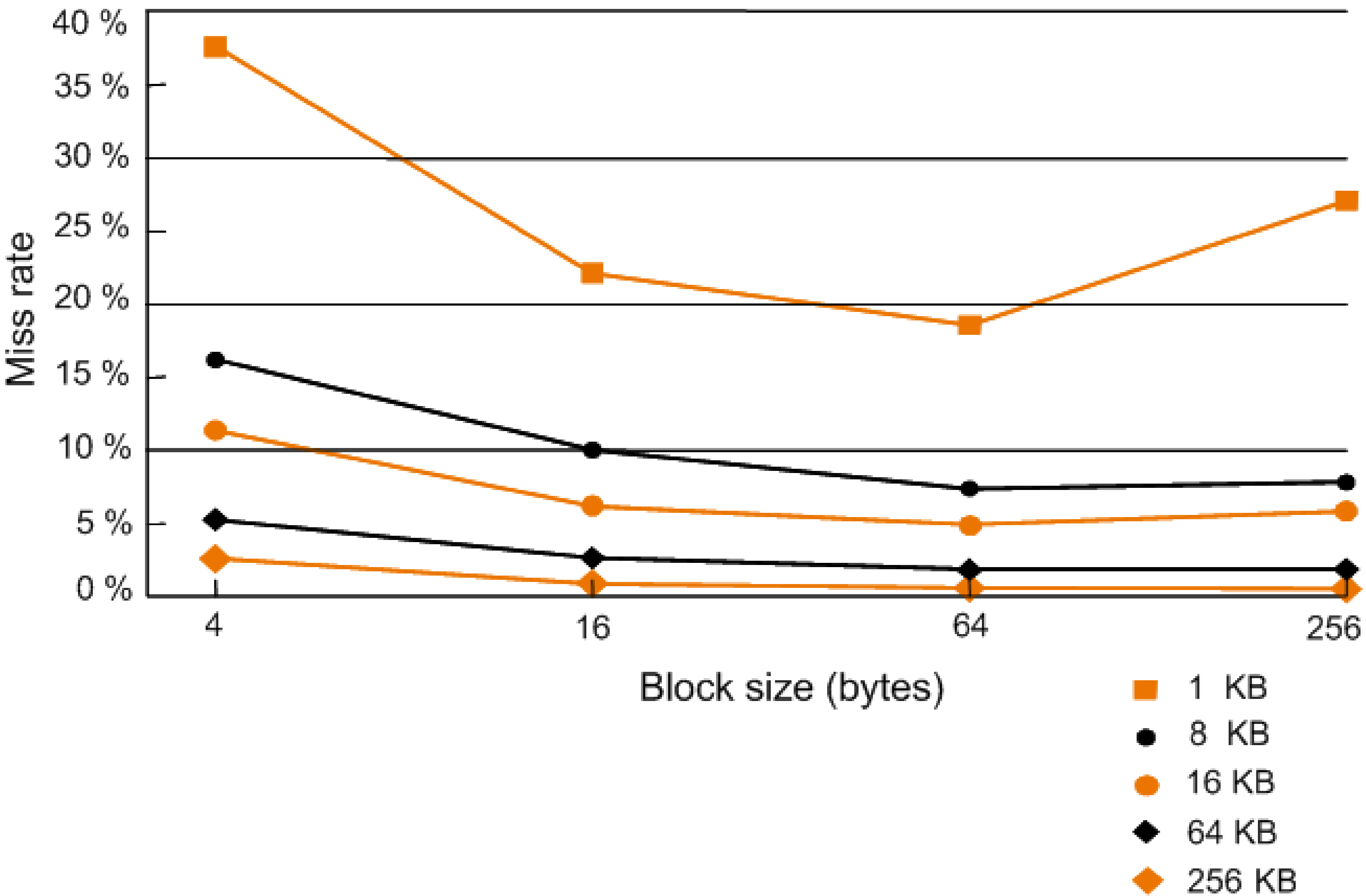
Why?:

1. Number of blocks in the cache become small.

2. Spatial locality among the words in a block decrease.

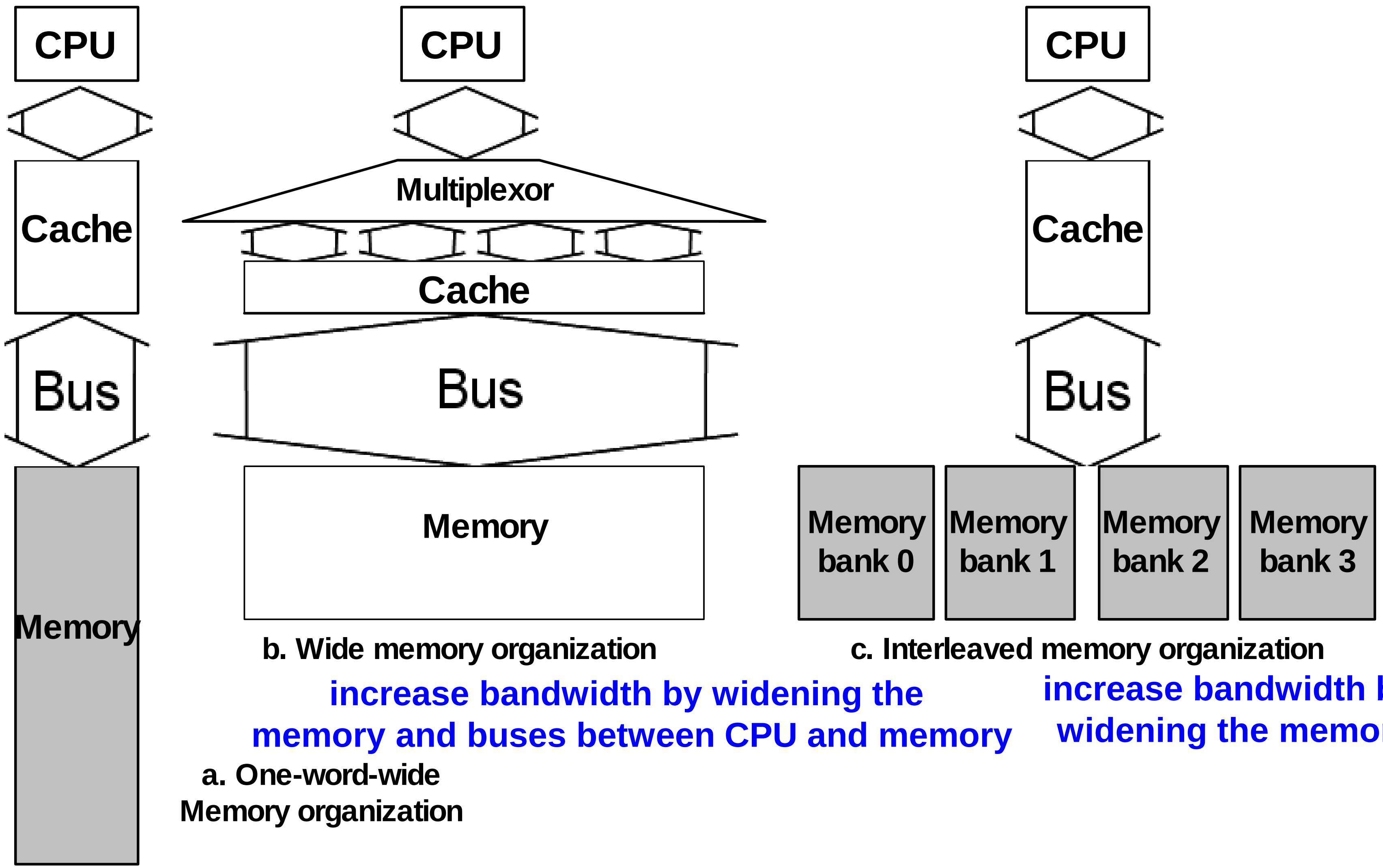
3. The cost (miss penalty) increases because the data transfer time increases.

Performance



Hardware Issues to increase Bus Bandwidth

- Make reading multiple words easier by using **banks of memory**



b. Wide memory organization

c. Interleaved memory organization

increase bandwidth by widening the memory and buses between CPU and memory

increase bandwidth by widening the memory

a. One-word-wide Memory organization

Where can a block be placed?

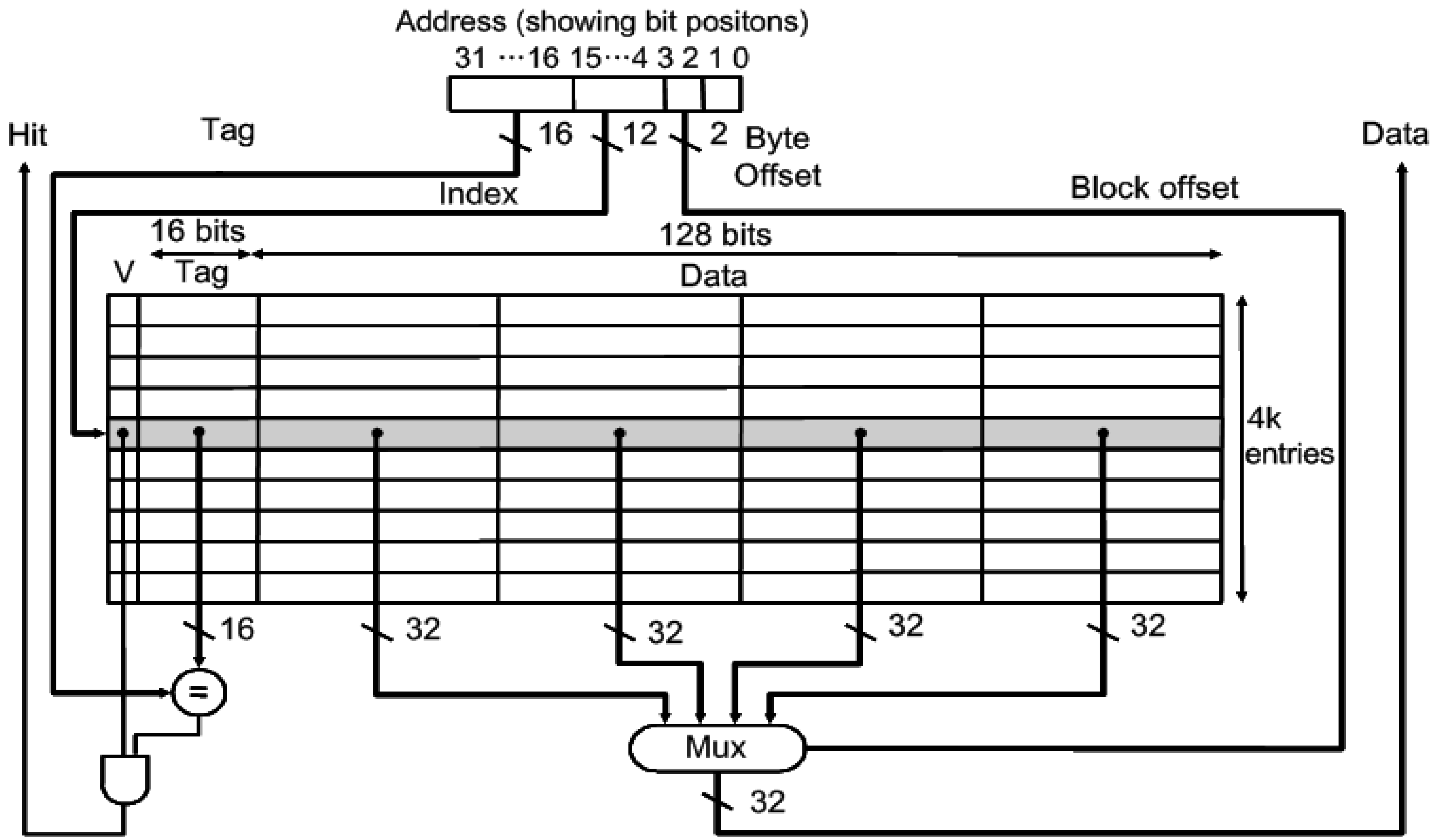
Three placement policies

- Direct mapped
- Fully associative
- Set associative

[1] Direct mapped

- **address is modulo the number of blocks in the cache**

31			0
Cache Tag	Block number (index)	Block offset	Byte Offset



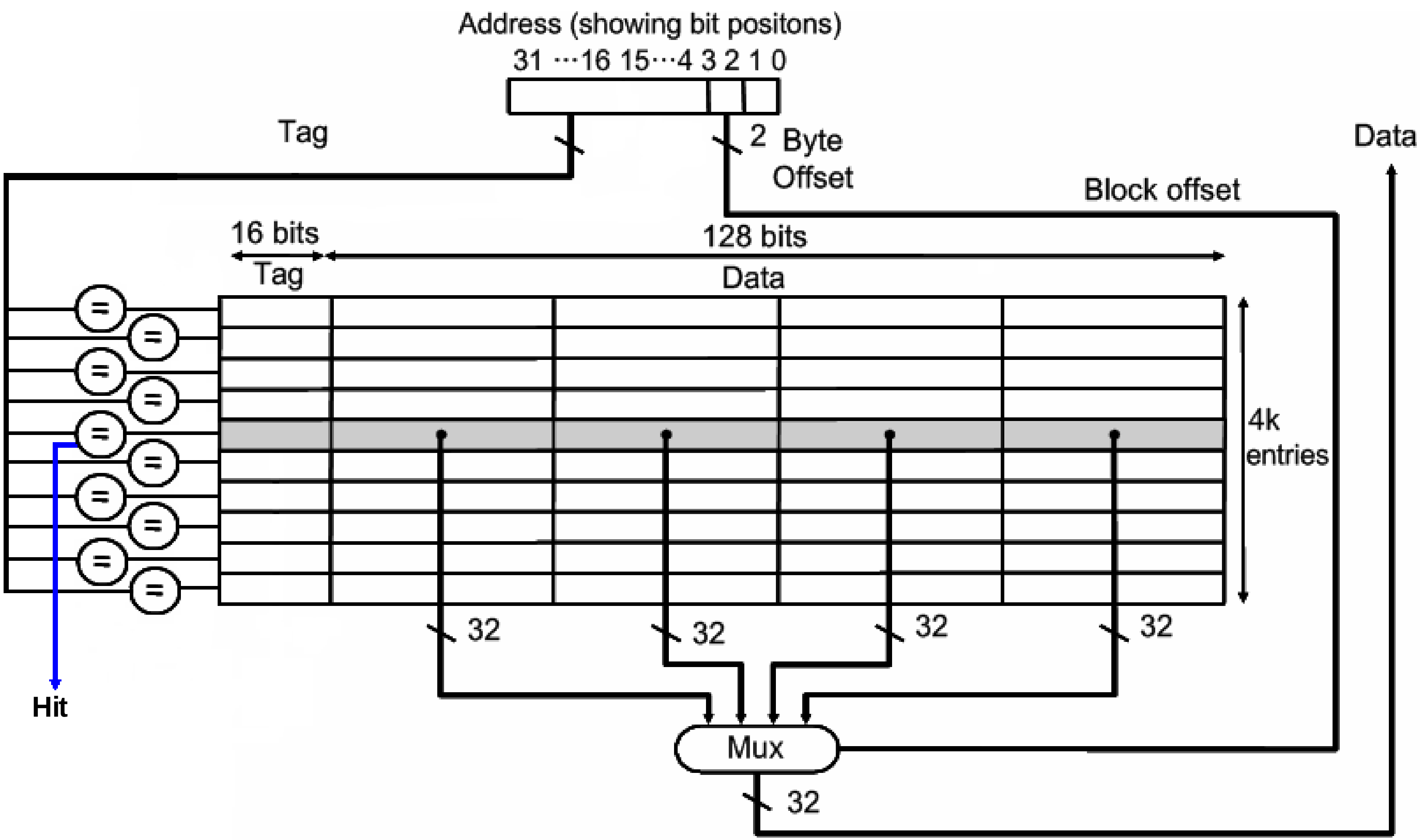
[2] Fully Associative

- **Fully Associative Cache : the other extreme**
  - Forget about the Cache Index
  - Block can be placed in any location in the cache
  - Compare the Cache Tags of all cache entries in parallel

Increasing associativity shrinks index and expands tag

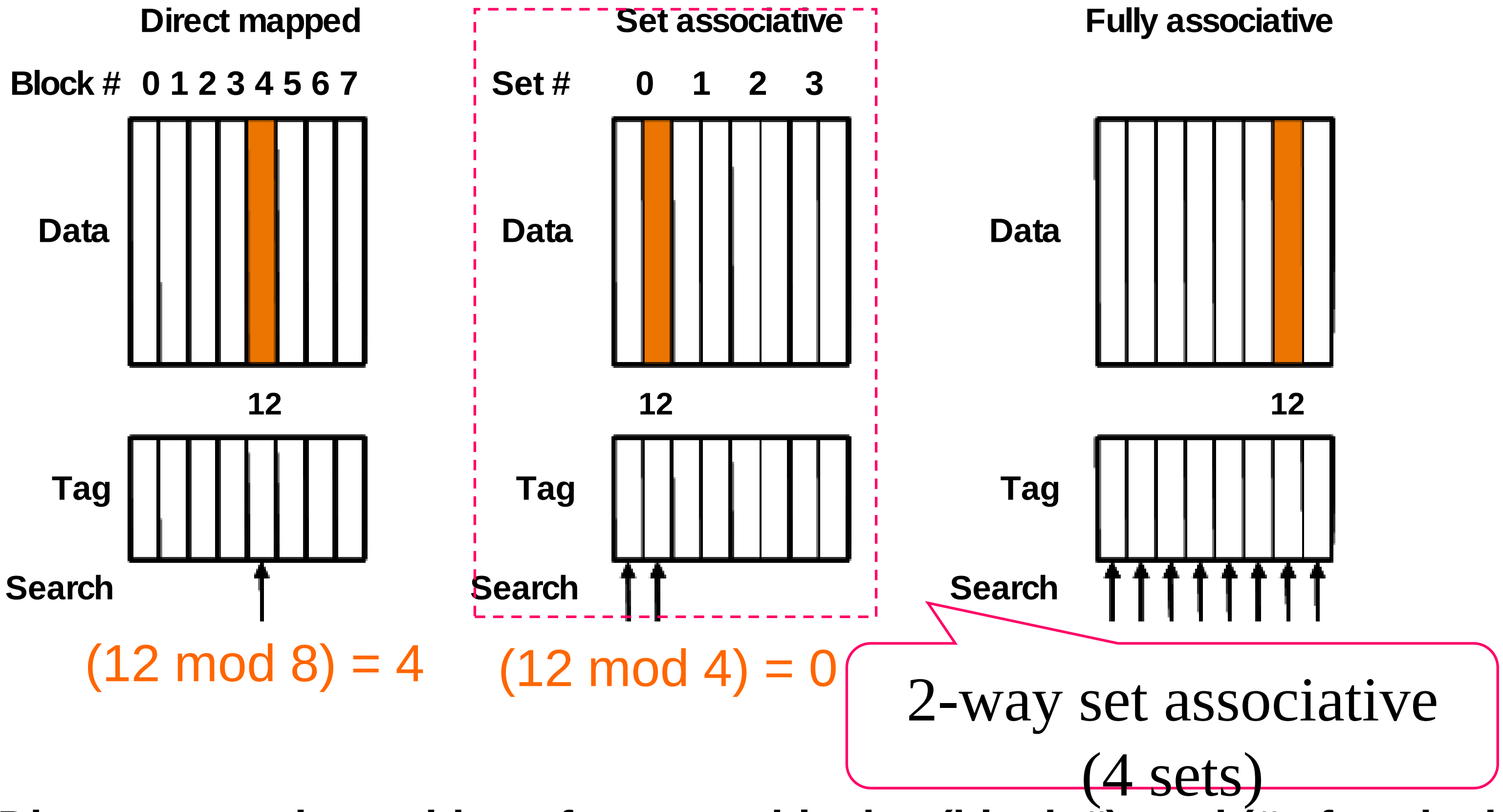
31		2 0
Cache Tag	Block offset	Byte offset

[2] Fully Associative



[3] Set Associative

Reducing Miss Ratio by More Flexible Placement



- **Direct mapped : position of memory block = (block #) mod (# of cache blocks)**
- **Set Associative : position of memory block = (block #) mod (# of sets in the cache)**
  - **block may be placed in any element in that set**

We can think Block Placement Strategies as Variations of Set-Associativity

Direct mapped (1-way set associative)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

4 sets (2-way set associative)

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

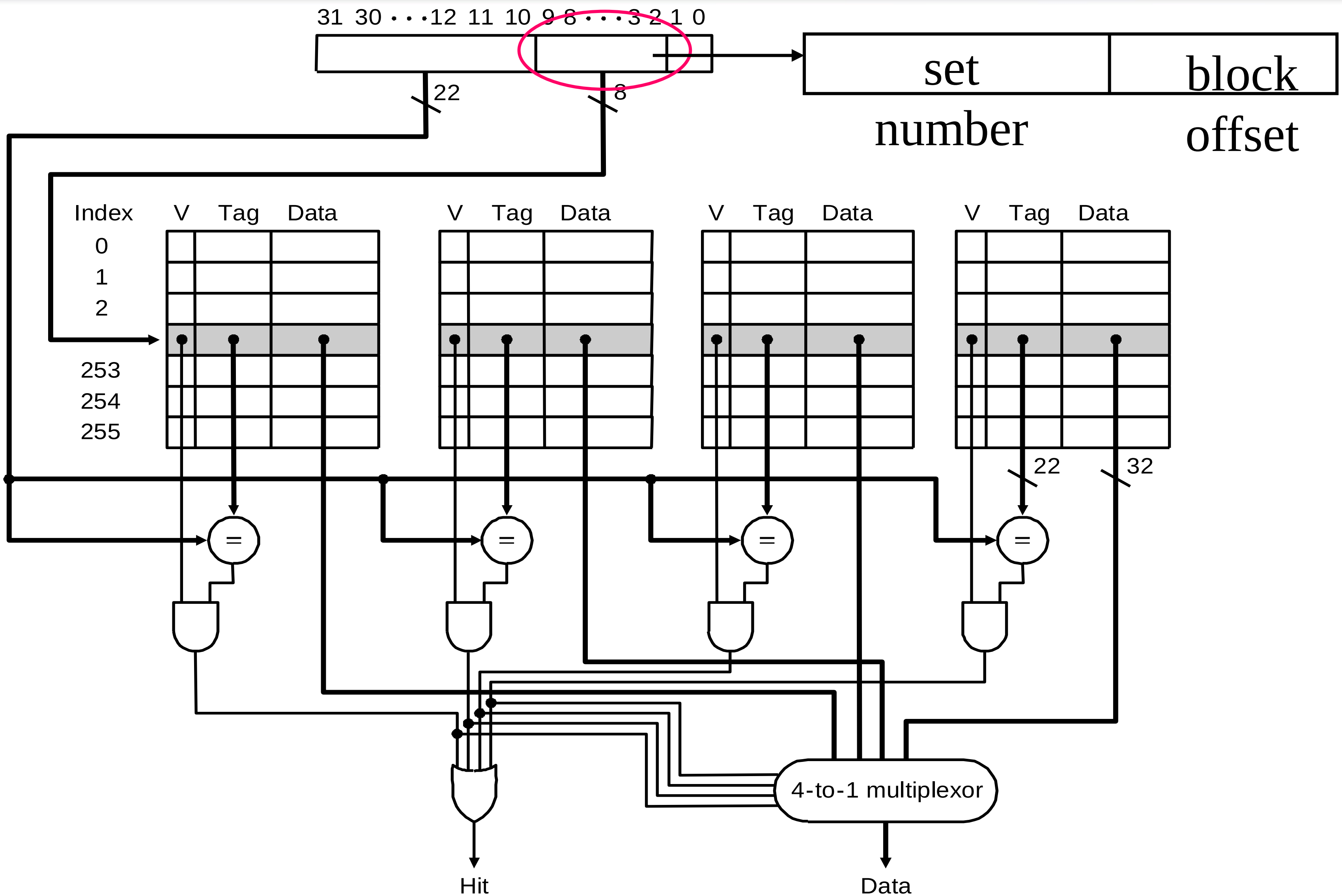
2 sets (4-way set associative)

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Fully associative (8-way set associative)

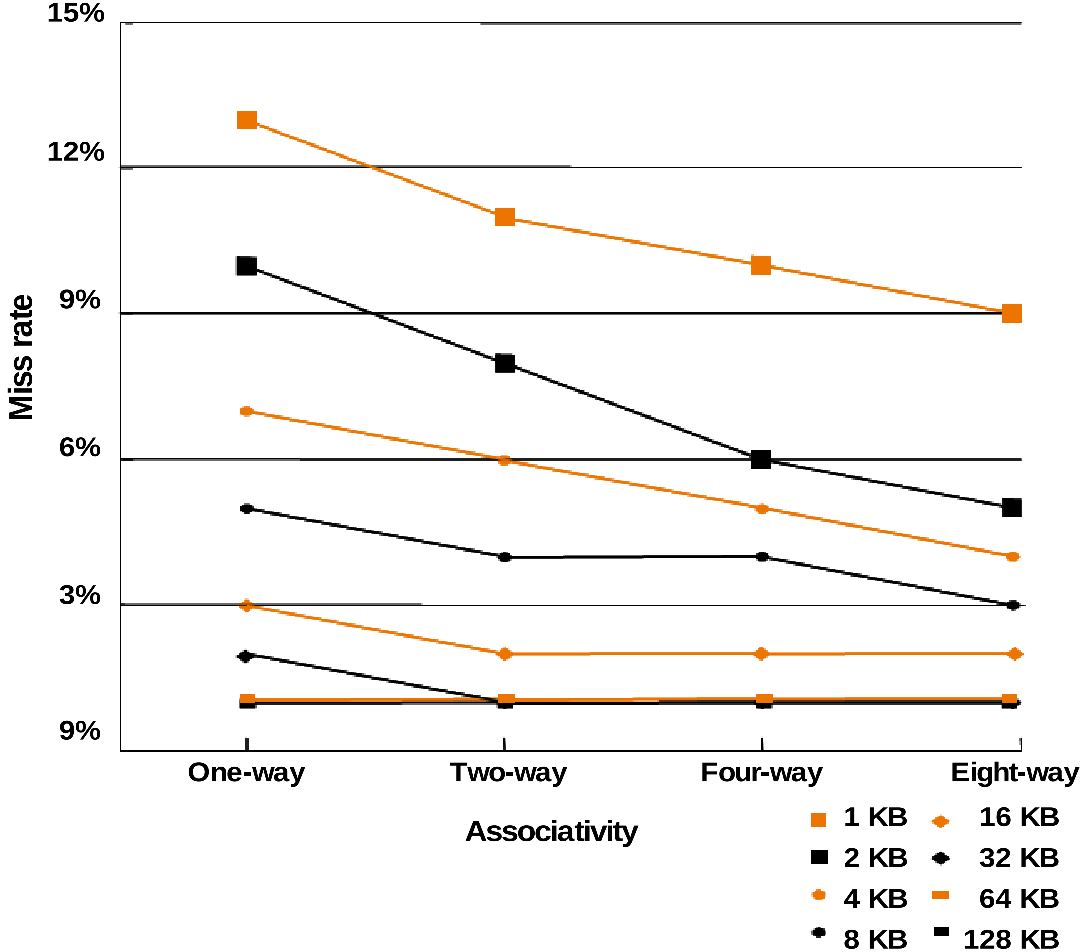
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

An Implementation of 4-way Set Associative Cache



Cache Performance with Associativity and Size

- **Increasing the Degree of Associativity**



– **Pros. : usually decrease miss ratio**

– **Cons. : increase the hit time (hardware complexity)**

Q: Which block should be replaced on a cache miss?

In case of fully or set associative cache only

[1] Random

- Replace the randomly selected block.

[2] FIFO

- Replace the block that has been in the cache longest.
- Need time stamp which records the time the block has been loaded.
- danger : might replace heavily used block

Q: Which block should be replaced on a cache miss?

[3] Least Recently Used (LRU)

- Replace the block that has been in the cache longest without reference.
- Need time stamp
  : Time stamp is updated every time the cache is referenced
- substantial overhead of updating the time stamp

[4] Least Frequently Used (LFU)

- Replace the block with fewest reference
- Need counter
- danger : the block that has been loaded into cache most recently might be replaced

Optimum replacement algorithm

What is the ideal (optimum) replacement algorithm?

: Replace the block that will not be used again for the furthest time into the future.

Associativity in Caches Example

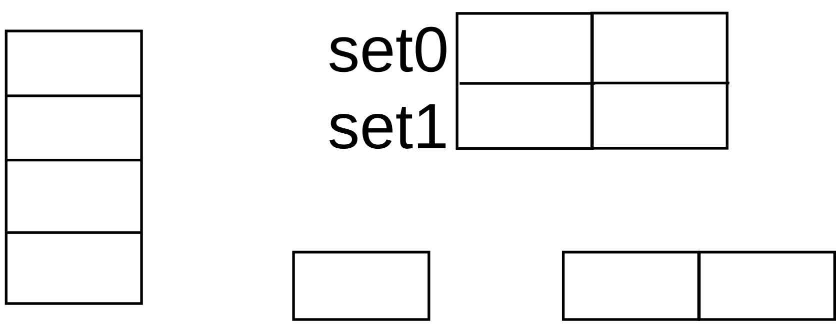
Usually, Higher Associativity ➔ Lower Miss Rate

- ***Q: assuming we use the least recently used replacement strategy***

  3 caches with 4 one-word block:

  direct mapped, 2-way set associative, fully associative

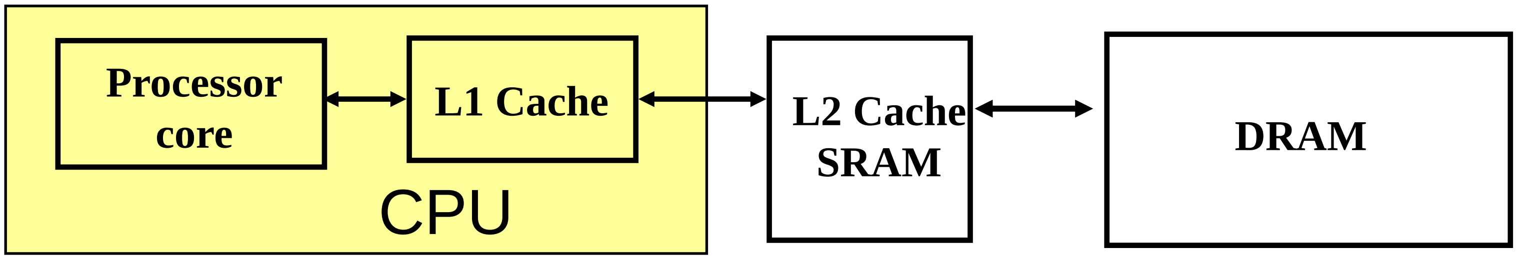
- **ex1) sequence of block addresses : 0, 8, 0, 6, 8**
  - direct-mapped cache block : 5 misses
  - 2-way set associative cache block : 4 misses
  - fully-associative : 3 misses → best



Decreasing miss penalty with Multilevel Caches

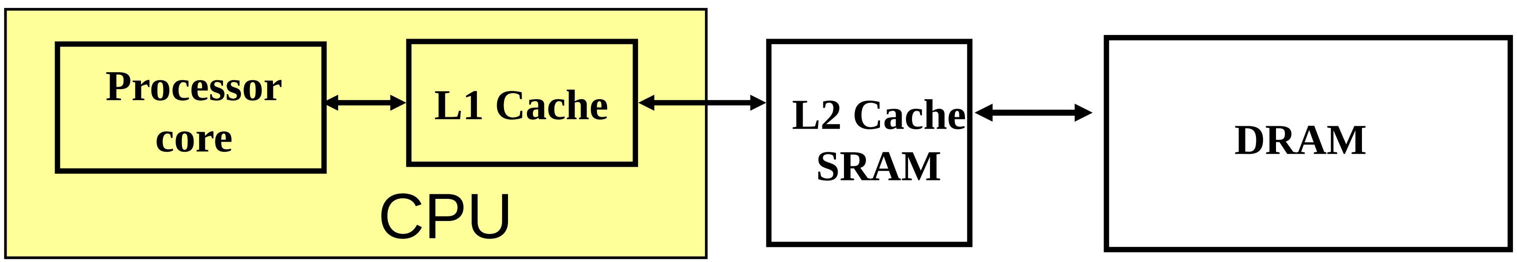
- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in the 2nd level cache

- **Example:**
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%



Decreasing miss penalty with Multilevel Caches

- Primary Cache is smaller and faster and uses a smaller block size
- Second Cache use larger block size compared to the single level cache



Conclusion

- **We can improve cache performance in the following ways.**

  - When the cache block size is too small or too large, cache shows poor performance.

  - There are three different placement policies : Direct mapped, Set Associative, and Fully Associative. Fully Associative cache the most complex, but usually shows best performance.

  - There are several replacement algorithms. LRU usually shows closest performance to optimum one.

  - Nowadays there are more than one level of cache to improve the performance.

  - Interleaved memory

Summary : Cache

- Principles of Locality with Memory Access
  - temporal locality : build memory hierarchy for large and fast
  - spatial locality : a cache must have a block size larger than 1 word
- Although larger block size decrease the miss ratio, it can increase miss penalty
- To avoid performance degrade from larger block size Increase memory bandwidth by wider bus and interleaving

Summary : Cache

- **Cache structure**
  - direct mapped : simplest
  - fully associative : low miss rate, but complex hw cost
  - set associative : compromised
- **Cache Write Policy**
  - write through : simple, but may cause processor stall frequently
  - write back : copy back to memory only when it is replaced, but needs complicated control

Today, CPU time is a f(ops, cache misses) vs. just f(ops)