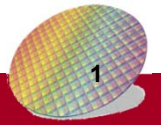
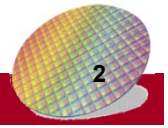
Memory Hierarchy Review



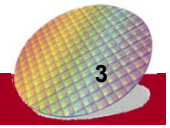
Recall: Terminology

- Cache
- Virtual Memory
- Memory Stall Cycles
- Direct mapped
- Valid Bit
- Block Address
- Write-through
- Instruction Cache
- Average Memory Access Time
- Cache Hit
- Page
- Miss Penalty
- Fully associative
- Dirty bit
- Block offset
- Write-back
- Data cache
- Hit time
- Cache miss
- Page fault
- Miss rate
- N-way set associative
- Least recently used
- Tag field
- Write allocate
- Unified cache
- Misses per instruction
- Block
- Locality
- Address trace
- Set
- Random replacement
- Index field
- No-write allocate
- Write buffer
- Write stall



Typical levels in memory hierarchy

Level	1	2	3	4
Name	Resisters	Cache	Main Memory	Disk storage
Typical Size	<1KB	32KB-8MB	<512GB	>1TB
Implementation Technology	Custom memory with multiple ports, CMOS	On-Chip CMOS SRAM	CMOS DRAM	Magnetic Disk
Access time (ns)	0.15~0.30	0.5~15	30~200	5000000
Bandwidth (MB/sec)	100,000-1,000,000	10,000-40,000	5,000-20,000	50-500
Managed by	Compiler	Hardware	OS	OS/operator
Backed by	Cache	Main memory	Disk	Other disks and DVD



Review: Cache Performance

- Assume CPU cycles includes the time to handle a cache hit and CPU is stalled during a cache miss:

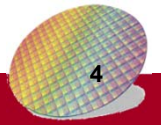
$$\text{CPU time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$= \text{Mem. access} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \text{IC} \times \frac{\text{Mem. access}}{\text{IC}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$\text{Miss Rate} = \frac{\text{\# of Misses}}{\text{\# of Mem. Accesses}}$$



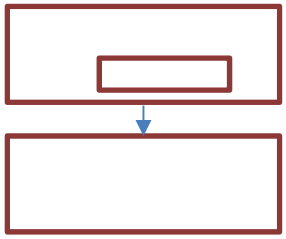
- Example: A computer has CPI=1.0 where no cache miss. The memory access are loads and store, and 50% of instructions are load and store. The miss penalty is 25 clock cycles and miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

CPU time if all instructions were cache hit

$$= (\text{CPU cycles}) \times \text{Cycle Time} \Longleftarrow \text{No Memory stall cycles}$$

$$= (\text{IC} \times \text{CPI}) \times \text{Clockcycle} = (\text{IC} \times 1) \times \text{Cycle Time}$$

$$= \text{IC} \times \text{Cycle Time}$$



CPU time with cache miss

$$= (\text{CPU cycles} + \text{Memory stall cycles}) \times \text{Cycle Time}$$

$$= \left(\text{IC} \times \text{CPI} + \text{IC} \times \frac{\text{Memory access}}{\text{IC}} \times \text{Miss rate} \times \text{Memory Penalty}\right) \times \text{Cycle Tim}$$

$$= (\text{IC} \times 1 + \text{IC} \times (1 + 0.5) \times 0.02 \times 25) \times \text{Clock cycle} \quad = 1.75 \times \text{IC} \times \text{Cycle Time}$$

(1+0.5): one instruction memory access and 0.5 data memory access per instruction

CPU with cache miss is 1.75 times slower

Miss Rate vs. Miss per instruction

- Miss rate: Miss per memory access

$$\text{Miss Rate} = \frac{\text{Misses}}{\text{Memory Access}}$$

- Miss per instruction

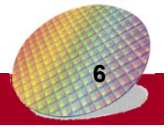
$$\text{Miss Per inst.} = \frac{\text{Memory AccessxMiss Rate}}{\text{Instructions}} = \frac{\text{Memory Access}}{\text{Instructions}} \times \text{Miss Rate}$$

Q: The memory access are loads and store, and 50% of instructions are loadand store. The miss rate is 2%. What is the misses per instruction?

A:

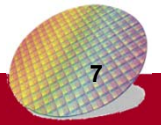
$$Miss\ per\ Inst = \frac{\text{Memory Access}}{\text{Instructions}} \times \text{Miss Rate}_{MA}$$

$$= (1 + 0.5) \times 0.02 = 0.03$$



Four Memory Hierarchy Questions

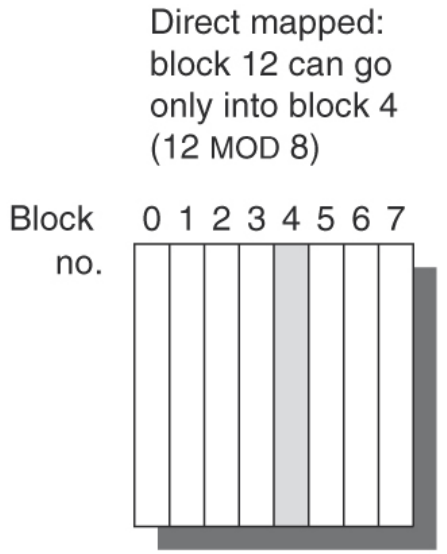
- Q1: Where can a block be placed in the upper level?
  - block placement
- Q2: How is a block found if it is in the upper level?
  - block identification
- Q3: Which block should be replaced on a miss?
  - block replacement
- Q4: What happens on a write?
  - write strategy



Q1: Where can a block be placed in the upper level?

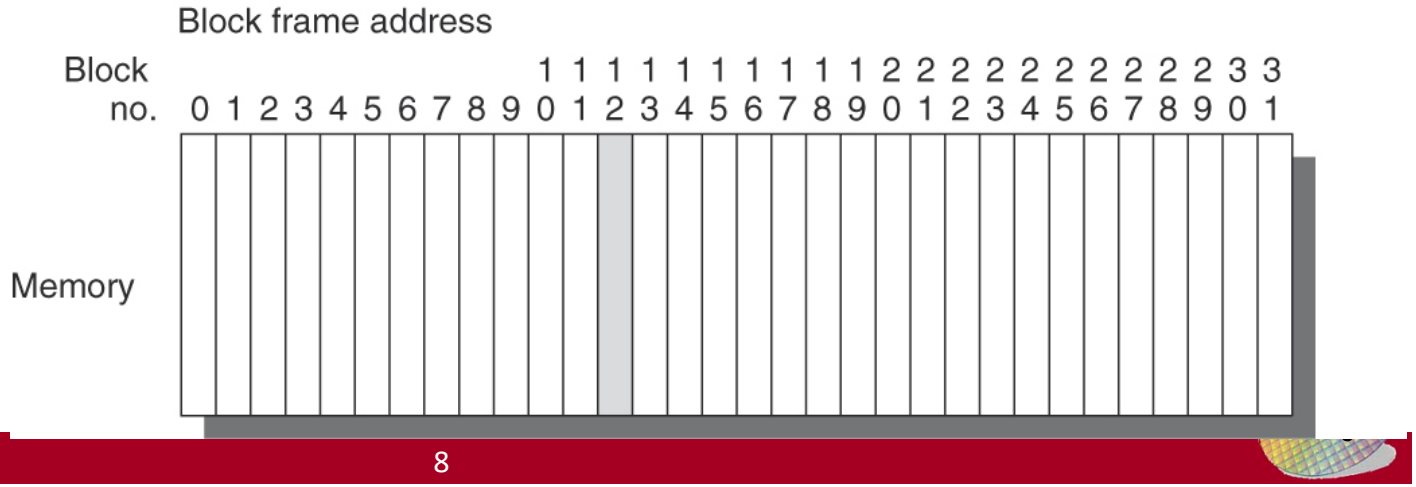
- Direct mapped

(Block address) mod (Number of blocks in cache)



Example: There are 32 blocks in the memory and 8 blocks in the cache. Assume direct mapped is used, where should the block with address 12 be placed?

Ans: (12 modulo 8) = 4

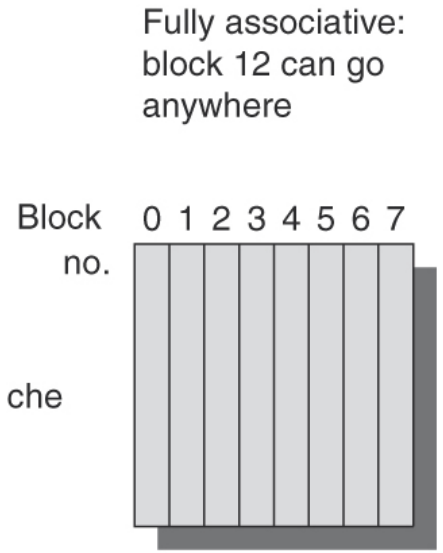


Q1: Where can a block be placed in the upper level?

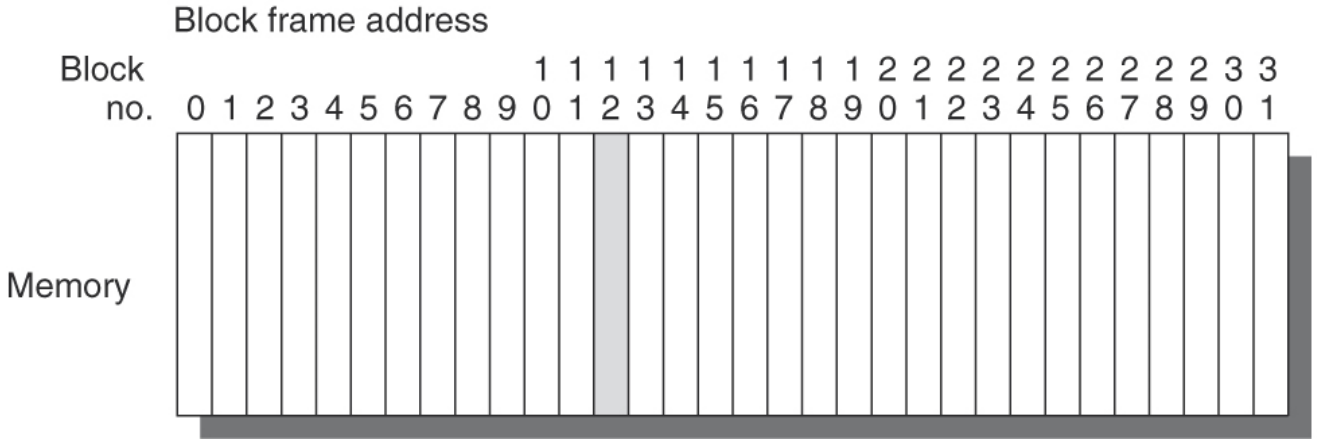
- Fully associative

  Any places.

Example: There are 32 blocks in the memory and 8 blocks in the cache. Assume fully associative mapping is used, where should the block with address 12 be placed?



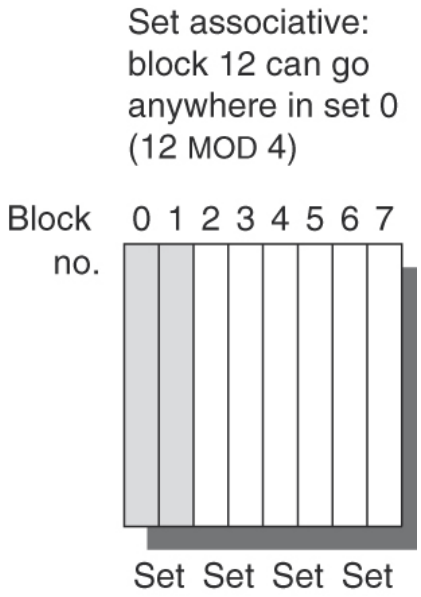
Ans: Any places



Q1: Where can a block be placed in the upper level?

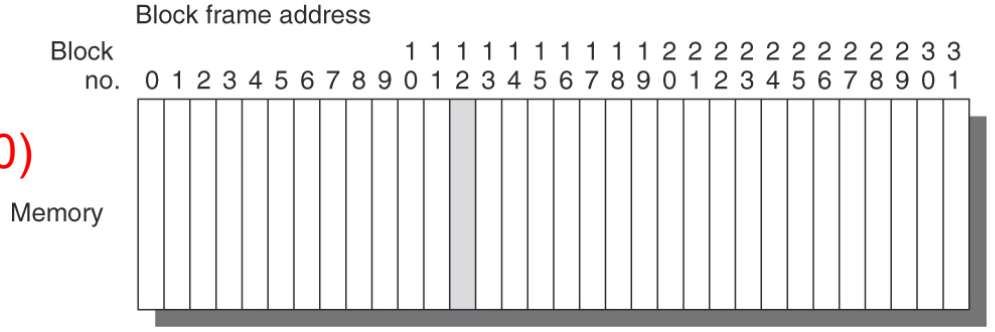
- Set associative:
  - A set is a group of blocks (can be 2, 4, 8, etc.)
  - A block can be placed in any places in a set.

(Block address) mod (Number of sets in cache)



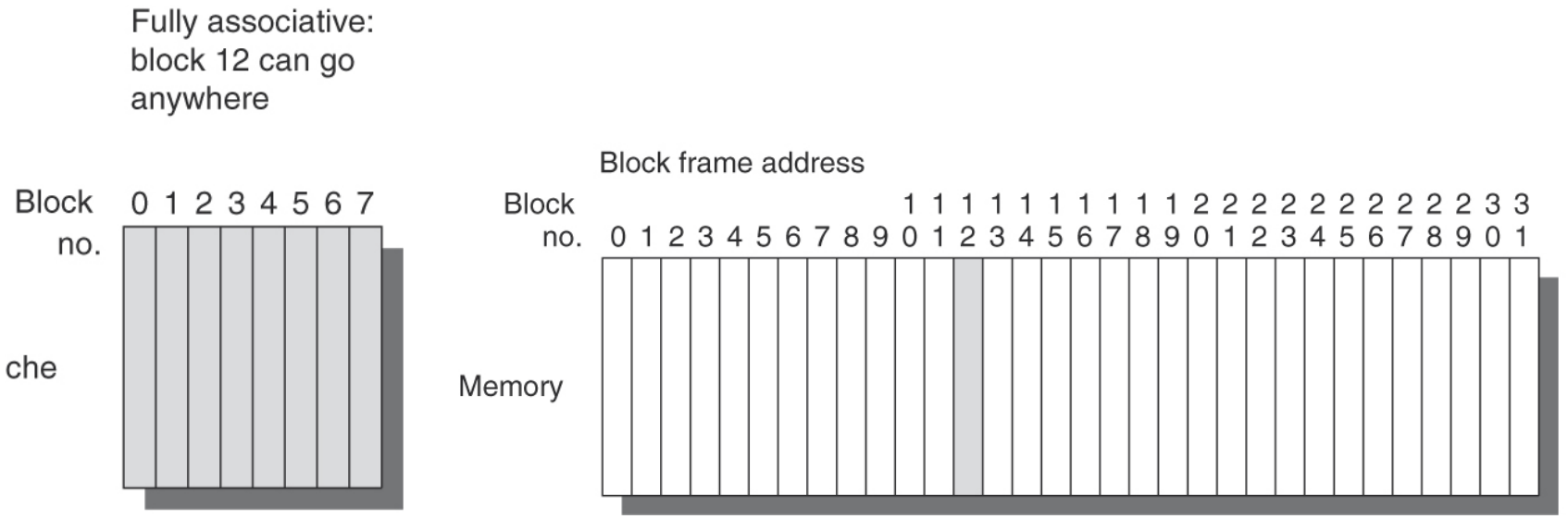
Example: There are 32 blocks in the memory and 8 blocks in the cache. Assume two-way set associative is used, where should the block with address 12 be placed?

Ans: 12 modulo (8 /2) = 0. Block 12 is placed in set 0 (any places in set 0)



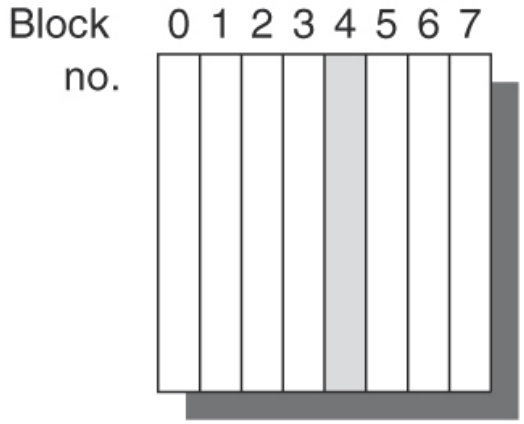
Q2: How is a block found if it is in the upper level?

- Fully associate
  - –All blocks are searched in parallel



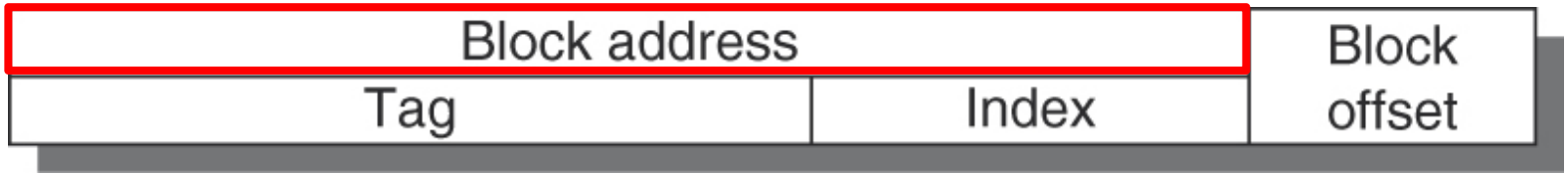
Q2: How is a block found if it is in the upper level?

- Direct mapped
  - Only one block is searched
  - Address is divided into block address, and block offset
  - Block address is used to identify the block



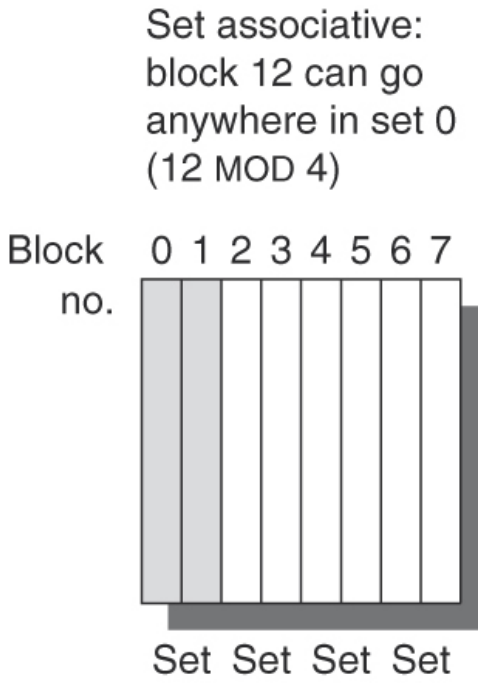
Block 12 can go only into block 4 (12 Mod 8), therefore, only block 4 is checked

Direct Mapped

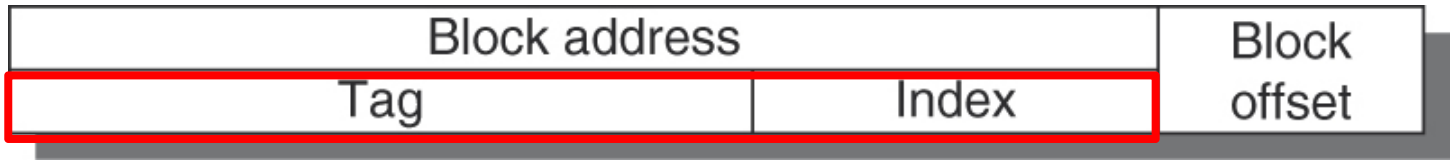


Q2: How is a block found if it is in the upper level?

- Set associative
  - –A set is selected and all blocks in the set are searched
  - –Address is divided into tag, index, and block offset
  - –Index is used to identify the set, and all blocks in the set are searched in parallel



Set associative



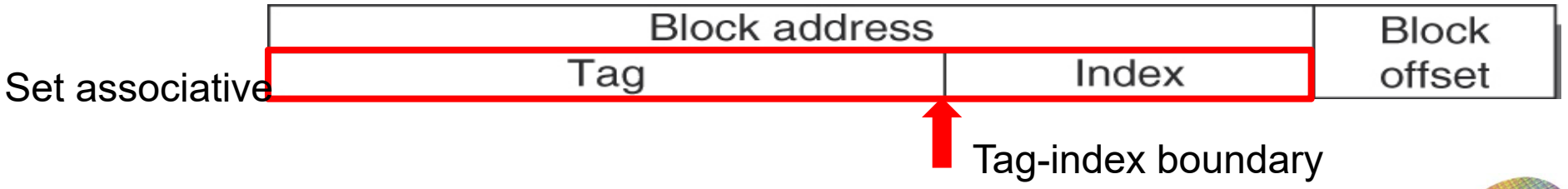
Q2: How is a block found if it is in the upper level?

- No need to compare block offset since all data in a block will either exist or not exist in the cache.
- No need to check index since index is used to select the set
- Increase associativity

  ⇒increase #block in a set

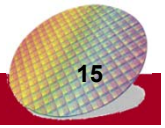
  ⇒Decrease the size of index

  ⇒Tag-index boundary moved to right



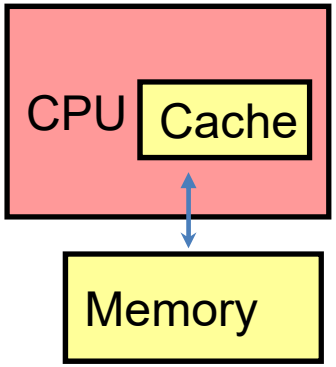
Q3: Which block should be replaced on a miss?

- Direct mapped
  - Only one block is checked for a hit, and only that block can be replaced
- Set associative and fully associative
  - Many blocks to choose from when a miss occurs
- Strategies to select which block to replace
  - Random or pseudorandom by using pseudorandom number generator.
  - First in, first out (FIFO)
  - Least Recently Used (LRU): based on temporal locality, block unused for the longest time is chosen
    - Pseudo LRU for less hardware cost and complexity: additional bit for a block. Bit is set when the block is accessed. Block is replaced if bit is not set



Q4: What happens on a write?

- Read dominate processor cache accesses
  - Read only on instruction cache. Only store and load instruction modify caches
- Read is faster than Write
  - Block read begins as soon as the block address is available
    - If hit, data are passed to CPU
    - If miss, data are discarded
- Write is longer than read
  - A block cannot be modified until its tag is checked to see there is a hit
- Handling Writes: two policies
  - Write-through: Info is written to the block in the cache and lower-level mem.
  - Write-back: Info is written to the block in the cache only. The modified block is written to mem only when it is replaced

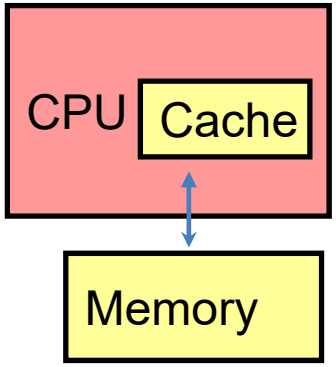


Q4: What happens on a write-- Write through

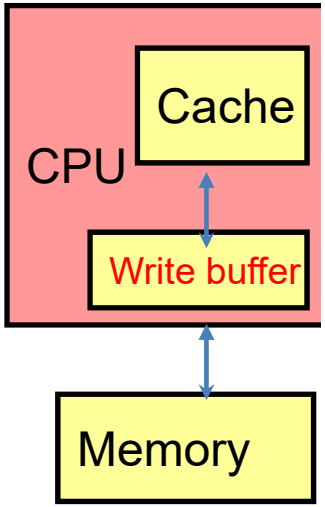
- Write through: Info is written to the block in the cache and lower-level mem. Writes take longer because needs to wait for MEM operation

  Q: if base CPI = 1, write to memory takes 100 cycles, miss rate is 10%, find new CPI

  A: New CPI= 1+10%x 100 = 11

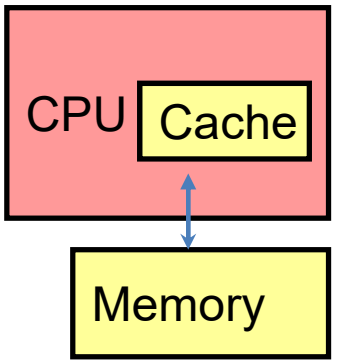


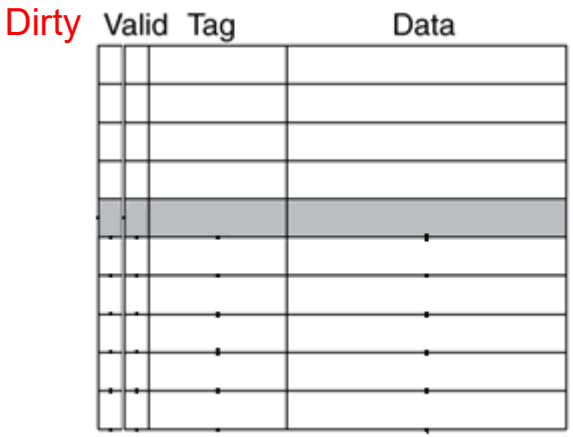
- Write through is slower : CPU must wait for write until data are written to lower level memory
- Improve the performance of write through using write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full



Q4: What happens on a write-- Write-Back

- Write back: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty (need a dirty bit)
  - When Dirty is 1, mean it is already modified (cache and memory data are different)
  - Multiple write to the same block result in only one write to lower level mem.
  - Use less memory bandwidth
  - More attractive for multiprocessors
  - Consume less power





Q4: What happens on a write? (write through vs. write back)

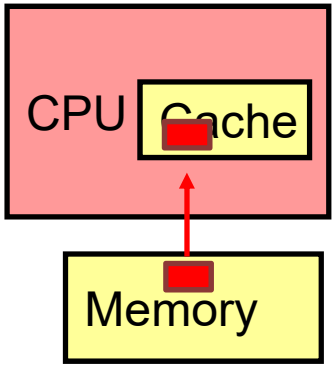
	Write Through	Write Back

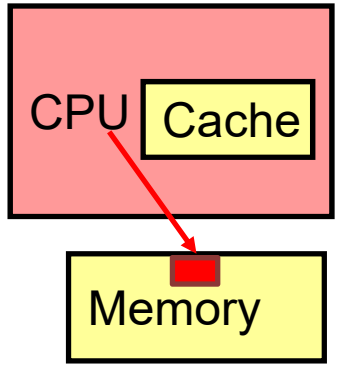
	• Easier to implement:
	  • cache is always clean. Write back policy will incur writes to lower level if a dirty block is replaced
	• No data coherency issues: lower level memory has the most current data, simplifying data coherency (especially for multiprocessor systems)
		• Faster: Write occur at the speed of the cache memory
		• Multiple write to the same block result in only one write to low level mem.
		  • Use less memory bandwidth
		  • More attractive for multiprocessors
		  • Consume less power

	Slower write: CPU must wait for write until data are written to lower level memory	More complicated to implement Data coherency issues

Two options on write miss

- What should happen on a write miss (write a block that is not in cache)?
  - Write allocate and No-write allocate
- Write allocate
  - Allocate block in cache (both cache and memory has the block), and write cache and memory
  - Write miss is like read miss
  - Simple
- No-write allocate
  - Do not allocate block in cache. Write block in memory only
  - Block stay in memory until it is read (Only reads are cached)
  - More cache space for read (reduce miss rate)
  - Useful for initialization (initialize data before use it)

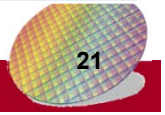




Write back or through vs. write allocate or no-write allocate

- Both write-through and write-back policies can use either of these write-miss policies, but usually they are paired in this way

	Write back	Write through
Write allocate	Subsequent writes (or even reads) to the same location, which is now cached.	
No write allocate		Subsequent writes still need to be written to the lower mem. => no advantage But cache space is saved for read



Cache Performance

- Average memory access time (AMAT): A better way to measure memory hierarchy performance

Average memory access time=Hit time + Miss rate x Miss Penalty

hit time: the time to hit in the cache

- Average memory access time can be measured in either absolute time (0.25 or 1.0 ns) or number of cycles (e.g. 50 cycles)

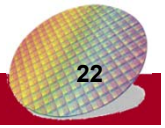
Example

CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%, find AMAT

AMAT = 1 + 0.05 × 20 = 2ns

AMAT is 2 cycles per instruction

But normally measured in cycles



Cache Performance Example

Assume cache miss penalty is 200 clock cycles, and all instructions normally take 1.0 clock cycles (ignoring memory stalls). Assume the average miss rate is 2%, and there is an average of 1.5 memory reference per instruction, and the average number of cache misses per 1000 instructions is 30. What is the impact on performance when the behavior of the cache is included?

Without any miss

$$\text{CPU time} = \text{IC x1 x Cycle Time}$$

With Cache -Method 1

$$\text{CPU time} = (\text{CPU cycles} + \text{Memory stall cycles}) \times \text{Cycle Time}$$

$$= (\text{IC} \times CPI + \text{Memory stall cycles}) \times \text{Cycle Time}$$

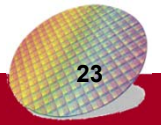
$$= \left(\text{IC} \times \text{CPI} + \text{IC} \times \frac{Memory\ access}{Instruction} \times \text{Miss rate} \times \text{Memory Penalty}\right) \times \text{Cycle Time}$$

$$= (\text{IC} \times 1.0 + \text{IC} \times 1.5 \times 2\% \times 200\ ) \times \text{Clock cycle} \qquad = 7 \times IC \times \text{ cycle Time}$$

With Cache - Method 2

$$\text{CPU time} = \left(\text{IC} \times \text{CPI} + \text{IC} \times \frac{Misses}{Instruction} \times \text{Memory Penalty}\right) \times \text{Cycle Time}$$

$$= (\text{IC} \times 1.0 + \text{IC} \times 0.03 \times 200\ ) \times \text{Clock cycle} \qquad = 7 \times IC \times \text{Clock cycle}$$

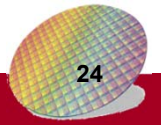


Remark on the previous example

- Cache behavior can have enormous impact on performance
  - CPU time is changed from ICx1xCycle Time to ICx7.0xCycle Time
- The lower the $CPI_{execution}$, the higher the relative impact of a fixed cache miss penalty

$$\text{CPU time} = \text{IC} \times \left(\text{CPI} + \frac{Memory\ access}{Instruction} \times \text{Miss rate} \times \text{Memory Penalty} \right) \times \text{Cycle Time}$$

- When calculating CPI, the cache miss penalty is measured in processor clock cycles for a miss. Even if memory hierarchies for two computer are identical, the processor with the higher clock rate has a larger number of clock cycles per miss and hence a higher memory portion of CPI

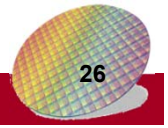


Six Basic Cache Optimization

Average memory access time=Hit time + Miss rate x Miss Penalty

Six basic cache optimizations are organized into

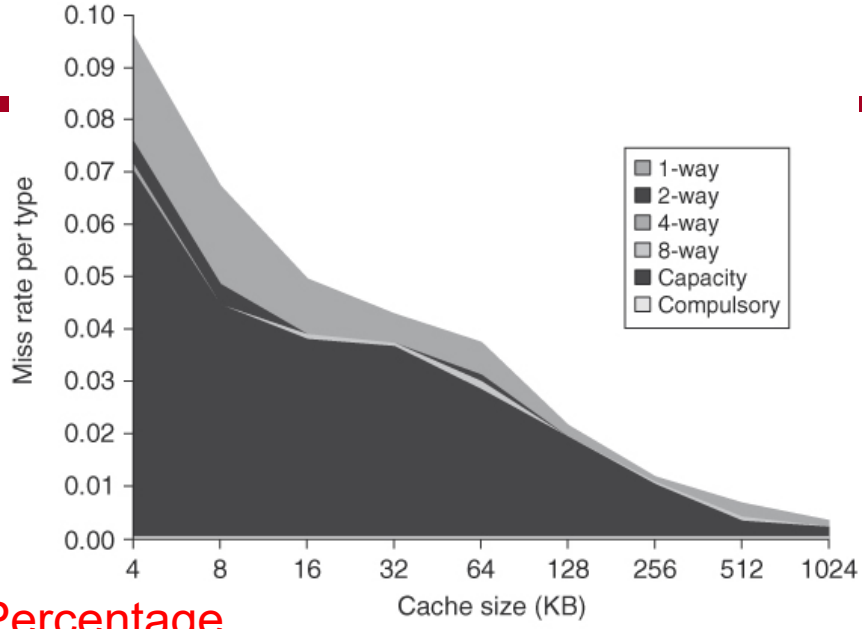
- Reducing the miss rate
  - a larger block size, large cache size, and higher associativity
- Reducing the miss penalty
  - Multilevel caches and giving reads priority over writes
- Reducing the time to hit in the cache
  - Avoiding address translation when indexing the cache



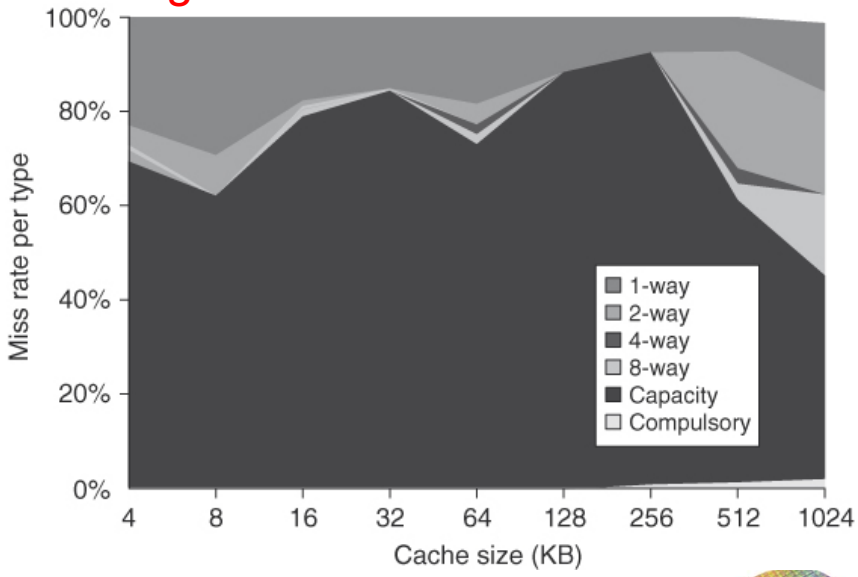
Types of Cache misses

- Compulsory: the very first access to a block cannot be in the cache. (normally miss rate is low)
  - Cold-start miss or first-reference misses
- Capacity: if cache cannot contain all blocks, capacity misses will occur because of blocks being discarded and later retrieved.
- Conflict : a block may be discarded and later retrieved if too many blocks map to a set (for set associative or direct mapped cache)
  - Collision misses

Actual miss rate

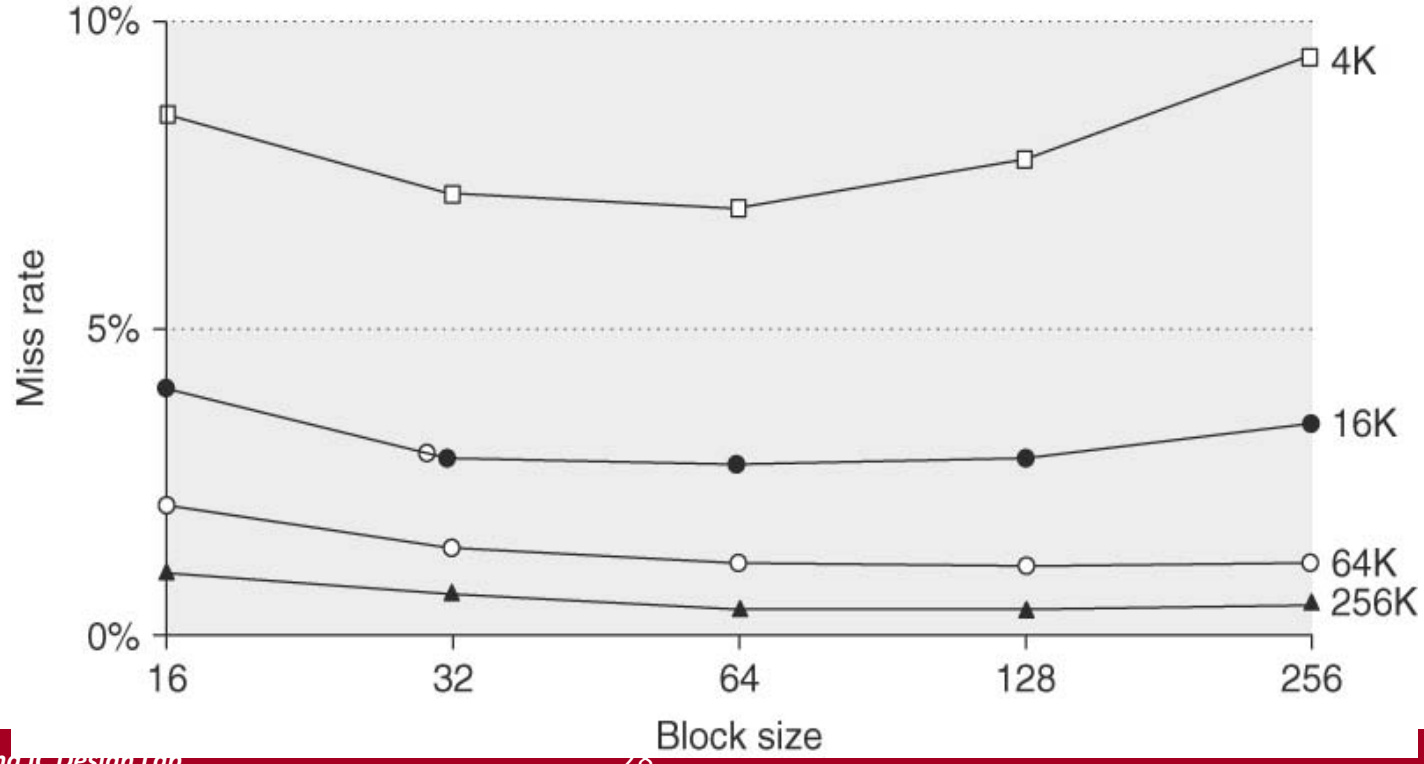


Percentage



First Optimization: Larger Block Size to Reduce Miss Rate

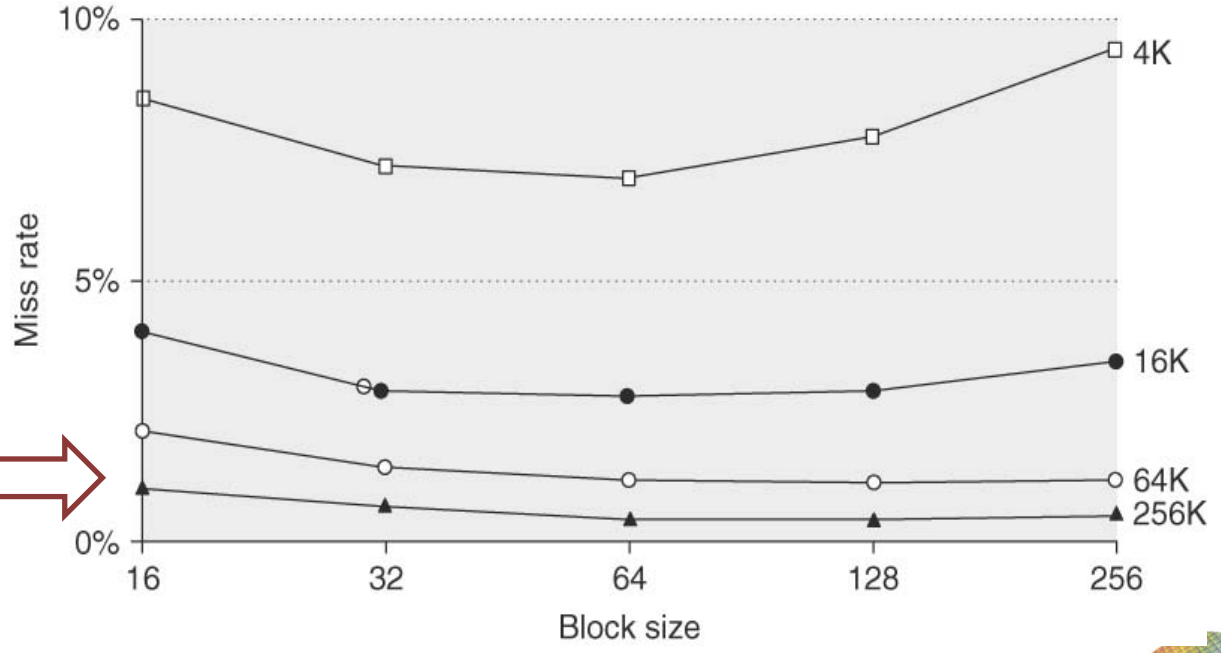
- Larger block size reduce miss due to spatial locality
- But Larger block increase miss penalty
- Larger block may increase conflict miss since the number of blocks is reduced
- So miss rate may go up if block size is too large relative to cache size



Second Optimization: Larger Caches to Reduce Miss Rate

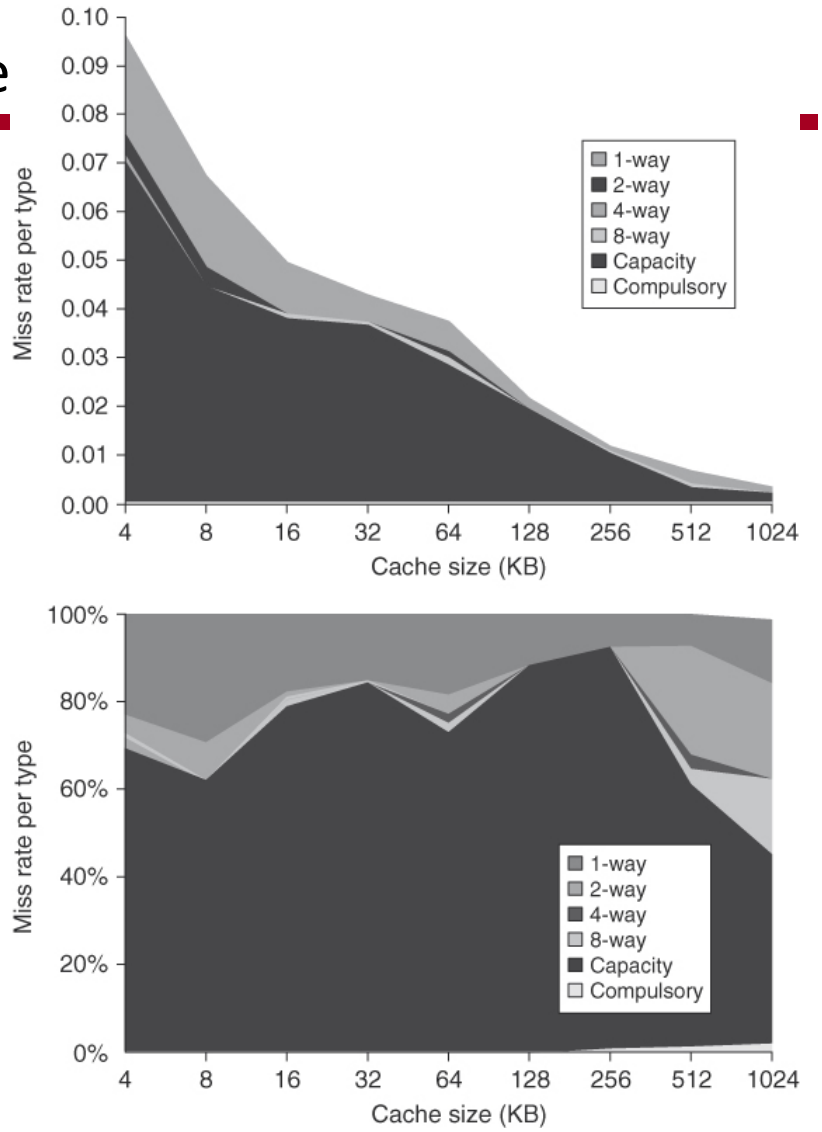
- Increase cache size can reduce miss rate
  - More blocks in cache
- Drawback:
  - Potentially longer hit time
  - Higher cost
  - Power

Larger Cache Size



Third Optimization: Higher Associativity to Reduce Miss Rate

- Higher associativity
  - ⇒more blocks in a set
  - ⇒reduce miss rate
- Practical Observation
  - 8-way set associative is almost as effective as fully associative (8-way vs. capacity)
  - 2:1 cache rule of thumb: a direct mapped cache of size N has about the same miss rate as a 2-way set associate of size N/2

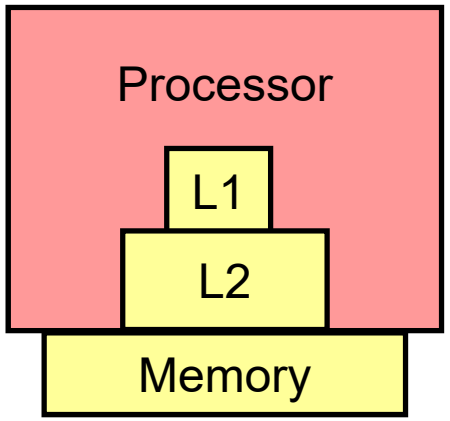


$$\text{Miss Rate}_{\text{Directed Mapped}}(\text{size N}) = \text{Miss Rate}_{2-\text{way}}(\text{size } \frac{\text{N}}{2})$$

Fourth Optimization: Multilevel Caches to Reduce Miss Penalty

- Use Multilevel Caches to reduce miss penalty
- Primary cache (Level-1) attached to CPU
- L2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- L2 cache reduces the write on mem

  => reduce miss penalty
- Some high-end systems include L-3 cache



$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{LI}$$

and $$\text{Miss penalty}_{LI} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Multilevel cache- Local Miss Rate vs. Global Miss Rate

- Local Miss Rate: the number of misses in a cache divided by the total number memory accesses to this cache. The miss rate in the above equation are both local miss rate.

$$\text{Local Miss Rate} = \frac{\#\ \text{Miss in a cache}}{\text{total number of memory access in this cache}}$$

- Global Miss Rate：the number of misses in the cache divided by the total number of memory access generated by the processor.

$$\text{Global Miss Rate} = \frac{\#\ \text{Miss in a cache}}{\text{total number of memory access}}$$

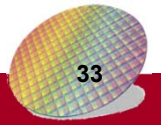
$\text{Global Miss Rate}_{L1} = \text{Miss Rate}_{L1}$

$\text{Global Miss Rate}_{L2} = \text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$

Local Miss rate

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Local Miss rate



Example

- Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache. What are the local and global miss rate of L1 and L2?

Global Miss Rate$_{L1}$ = Local Miss Rate$_{L1}$ =40/1000=4%

Local Miss Rate$_{L2}$ = 20/40=50%

Global Miss Rate$_{L2}$ = 4%x 50%= 2%

Or Global Miss Rate$_{L2}$ = 20/1000= 2%

- Follow the left question. Assume L1 hit time is 1 clock cycle, and L2 hit time is 10 clock cycles and L2 miss penalty is 200 clock cycles, and there are 1.5 memory references per instruction. What is the average memory access time ? Ignore the impact of writes.

$$\text{AMAT} = 1 + 0.04 \times (10 + 0.5 \times 200)$$
$$= 5.4 \text{ cycles}$$

$$\text{AMAT} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Fourth Optimization: Multilevel Caches to Reduce Miss Penalty

- Assume CPU CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

  find effective CPI if only L1 cache

Miss penalty = 100ns/0.25ns
= 400 cycles

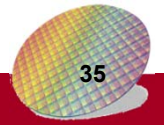
Effective CPI = 1 + 0.02 × 400 = 9

Now with L2 cache with access time= 5ns

- L2 cache
  - Global miss rate to main memory = 0.5% (Local miss rate = 25%)
  - L2 Miss penalty = 100ns/0.25ns = 400 cycles
- L-1 miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles

  find new CPI

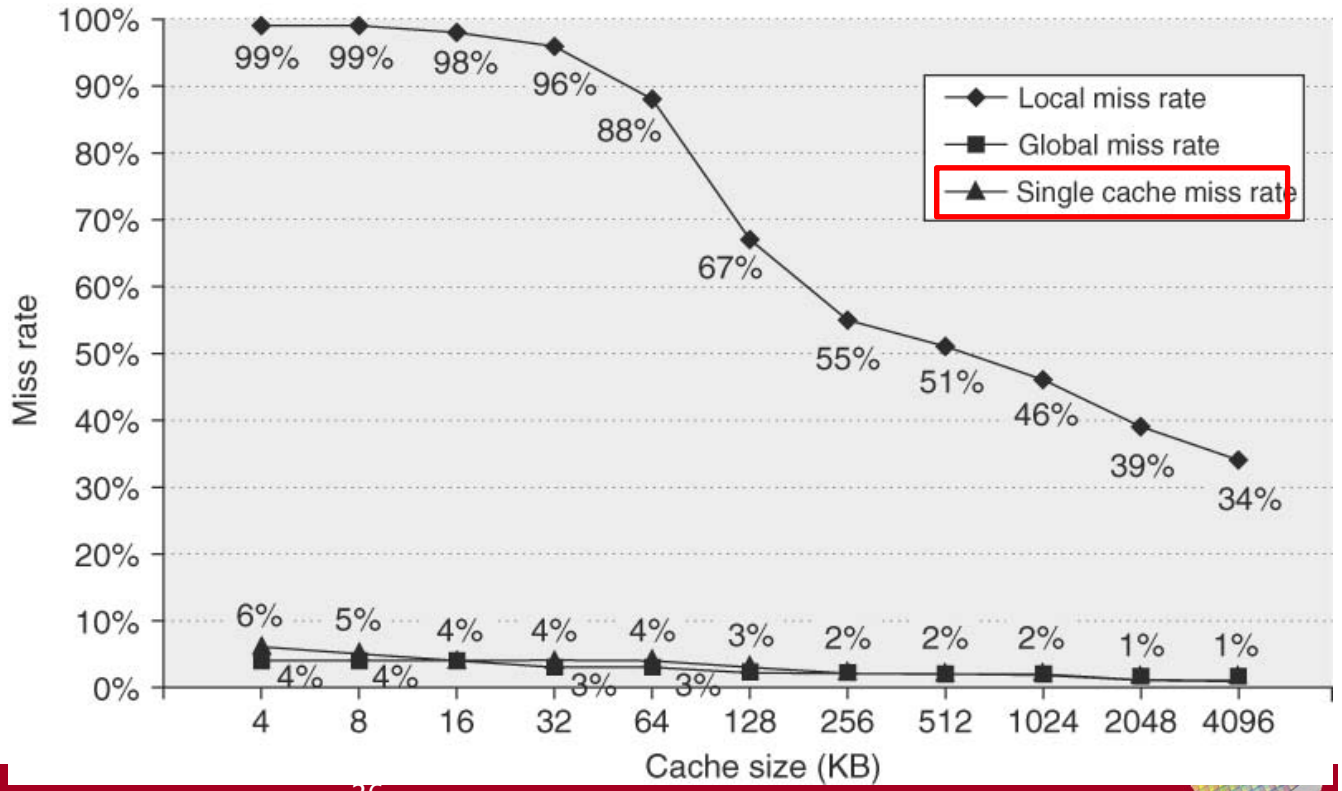
CPI = 1 + 0.02 (20 + 25% × 400)
= 3.4



Remark to Fourth Optimization: Multilevel Caches to Reduce Miss Penalty

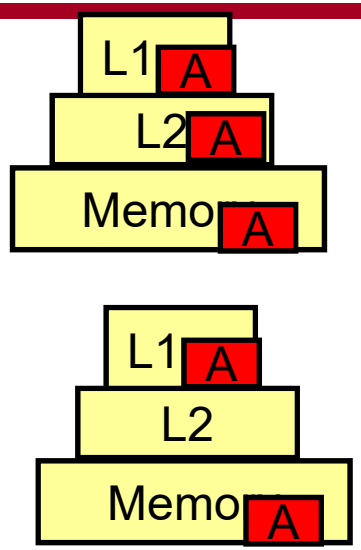
- Previous example uses write back first level cache and combined read and writes
  - Write-through first-level cache send all writes to the second level, not just the misses.
  - Write buffer may be used
- Miss rate in L2 vs. cache size
  - Global M.R. of L2 is similar to single cache miss rate
  - Local miss rate is NOT a good measure of secondary cache miss rate

- Normally, L1 Cache focus on minimal hit time, and L2 cache focus on low miss rate to avoid main memory access
- To reduce local miss rate in L2
  - L2 should be much larger than L1 cache



Remark: Fourth Optimization: Multilevel Caches to Reduce Miss Penalty

- Multilevel inclusion: Data in the first-level cache are in L2 cache
  - Inclusion is desired because consistency between I/O and caches can be determined just by checking the L2 cache
- Multilevel exclusion: Data in the first-level cache are not in L2 cache
  - Cache miss result in a swap of blocks between L1 and L2 instead of a replacement of an L1 block with an L2 block



	M. Inclusion	M. exclusion
Natural?	More natural	Less natural
L2 block replacement	• Lower hit rate in L1 cache • Replacement of L2 blocks must invalidate all block in L1	Higher L1 hit rate
Cache miss in L1	Cache miss in replacement of an L1 block with an L2 block	Cache miss in L1 result in swap L1 and L2

Fifth Optimization: Giving Priority to Read Misses over Writes to Reduce Miss Penalty

-- for write through cache with write buffer

- Write buffer is an critical improvement in write through cache
  - Reduce memory latecy
  - But latest data exist in write buffer => an issue for read miss

Assume write-through, and write buffer is used

SW R3, 512(R0)

(R3 is removed from cache)

……

LW R2, 512(R0)

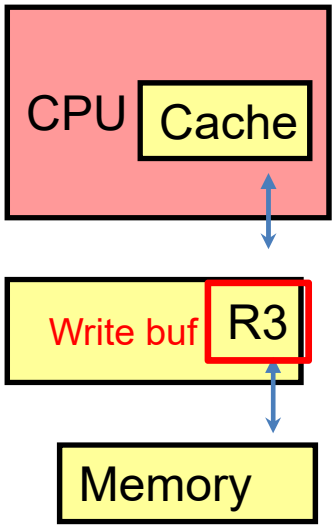
What is the problem?

If data R3 is still in the write buffer, and

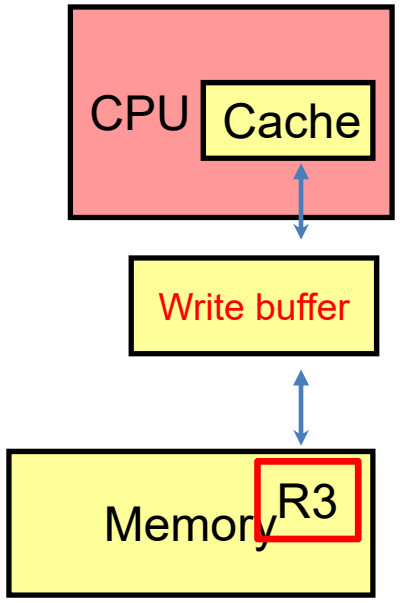
LW R2, 512(R0)

has a read miss

How to handle “Read Miss”?



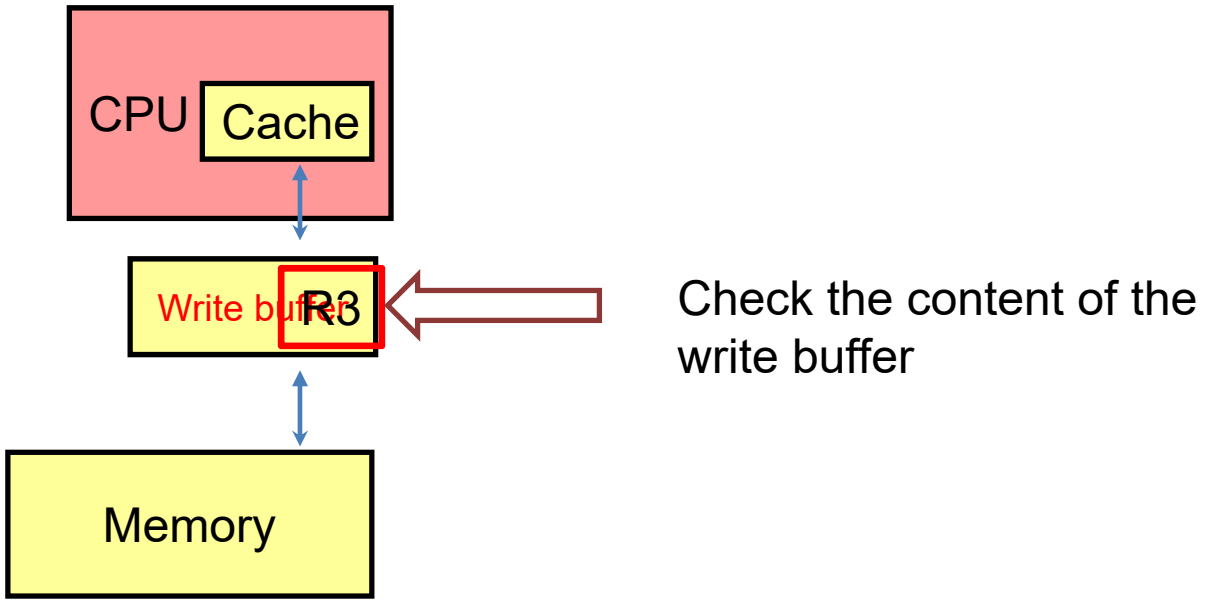
- Method 1: read miss to wait until write buffer is empty (read after R3 is written to memory)



Fifth Optimization: Giving Priority to Read Misses over Writes to Reduce Miss Penalty

-- for write through cache with write buffer

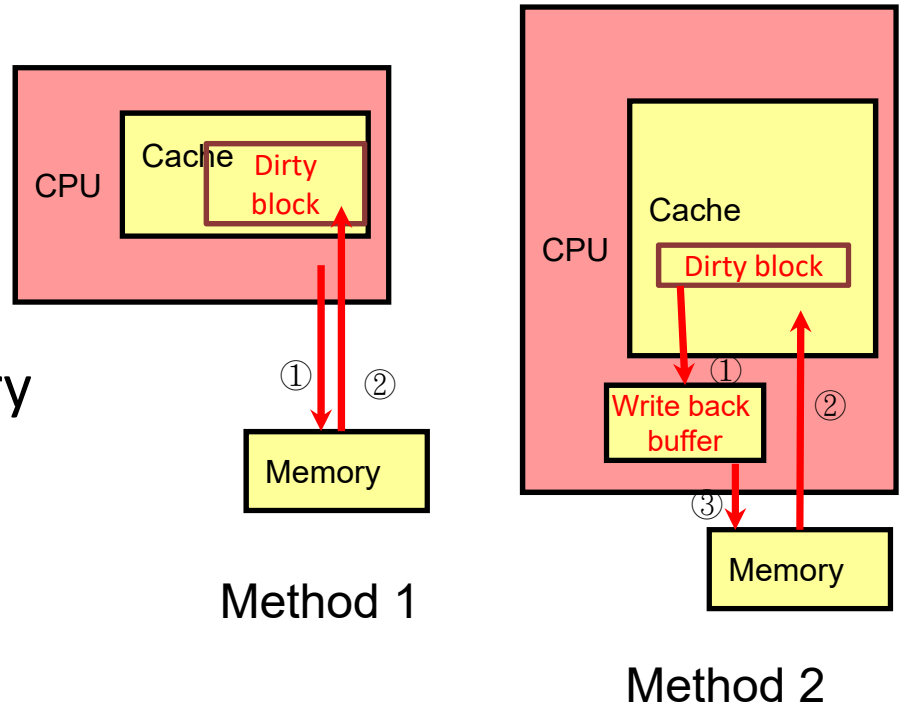
- Method 2: Check the content of the write buffer on a read miss, and if there are no conflict, read miss continues
  - Data in the write buffer are not used by the following instruction
  - Avoid Read after write hazard



Fifth Optimization: Giving Priority to Read Misses over Writes to Reduce Miss Penalty

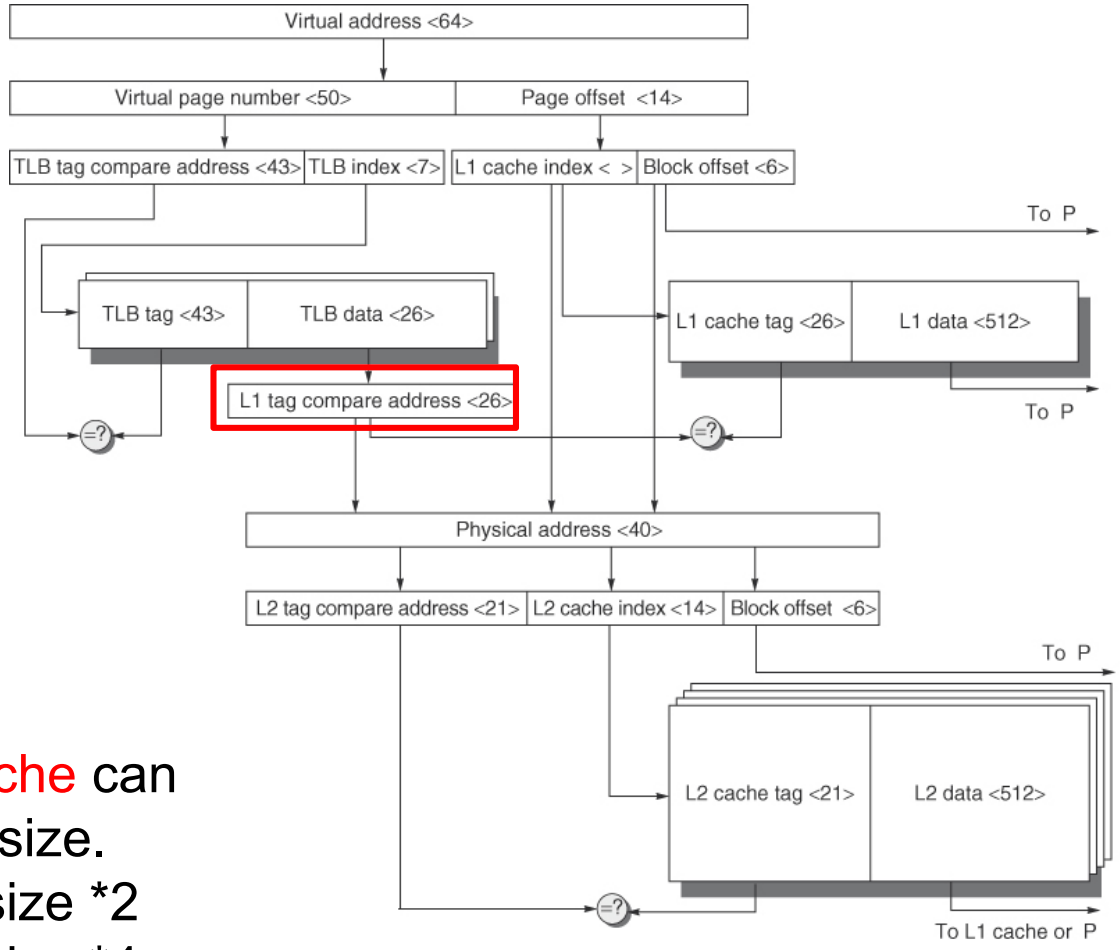
-- for write back cache when read miss needs to replace a dirty block

- Method 1: write the dirty block into memory, and the read memory
- Method 2: copy the dirty block into a buffer, then read the memory, then write the memory
  - No need to wait for write to memory



Reduce cost of writes in write back cache

- Virtually indexed, physically tagged cache
  - Allows the cache read to begin immediately, but the tag comparison is still with physical address
  - Virtual address translation and cache read are done simultaneously

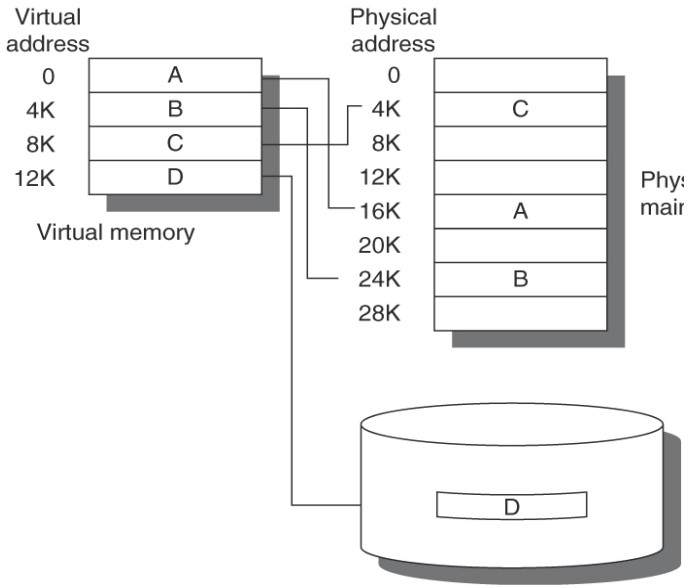


Cache size limitation:

a direct mapped (1-way) cache can be no bigger than the page size.
2-way cache size <= page size *2
4-way cache size <= page size *4

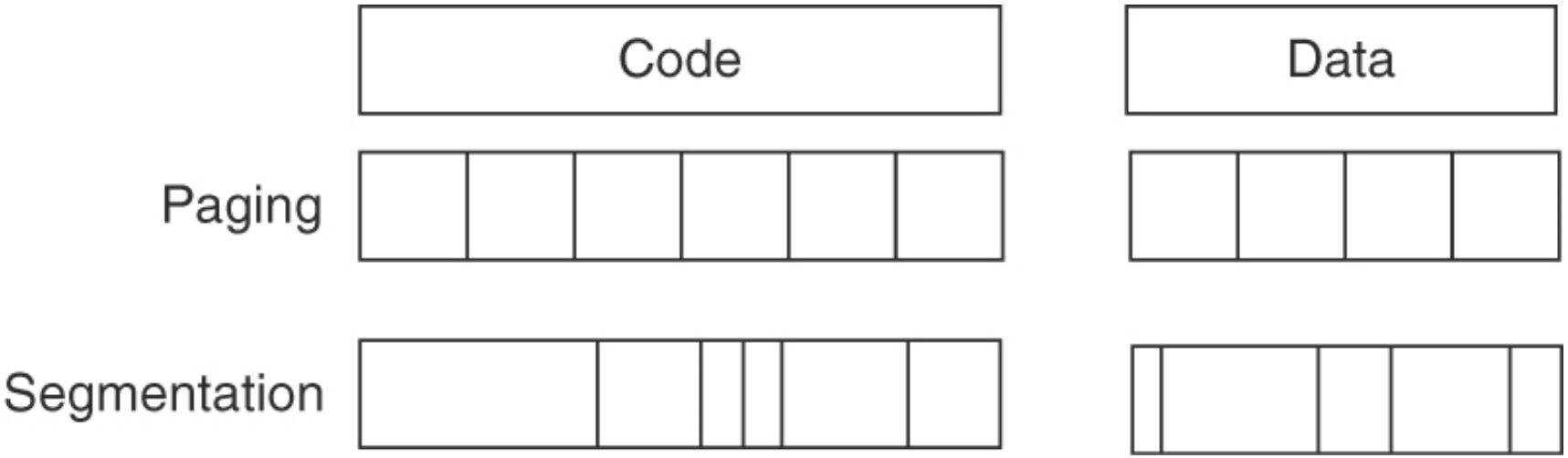
Virtual Memory

- Virtual Memory automatically manages the two levels of the memory hierarchy represented by main memory and secondary storage.
  - divide physical memory into blocks and load page form storage to main memory when requested.
- Benefit of VM
  - Automatically manage the memory hierarchy (No need for programmer)
  - Save program start time (not all program needed to be in physical memory before it can begin)
  - Protection: Blocks are allocated to different processes. Protection scheme is necessary to restrict a process can only its blocks
  - Simplified program loading for execution via *relocation*



Variable or Fixed page size

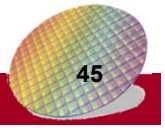
- Virtual memory block size can be fixed or variable
- Fixed block size: page
- Variable block size: segment
- Hybrid approach: paged segment where a segment contains a integral number of pages



- A hybrid approach – paged segmented
  - Segments are composed of pages, so replacing a block is easy,

Cache vs. VM

Cache	Virtual Memory
Blocks (cache line)	Page or segment
Cache miss	Page fault or address fault
Memory Address is used to identify a block in cache	Address translation or memory mapping
Replacement on cache miss controlled by hardware	VM replacement is mainly controlled by OS
Cache size is independent of processor address size	Size of processor address determine the size of virtual memory
Lower cache is backing of upper level	Storage is lower-level backing of main mem, but it is also used by file system



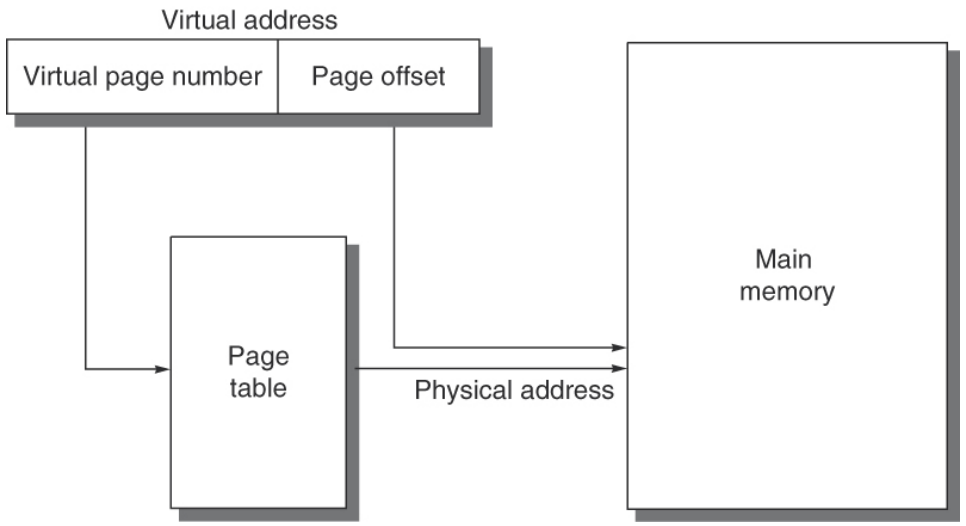
Four Memory Hierarchy Question Revisited

- Q1: Where can a block be placed in main memory?

  Fully associative: access storage incurs high miss penalty, to reduce miss rate, fully associative is used

- Q2: How is a block found if it is in main memory

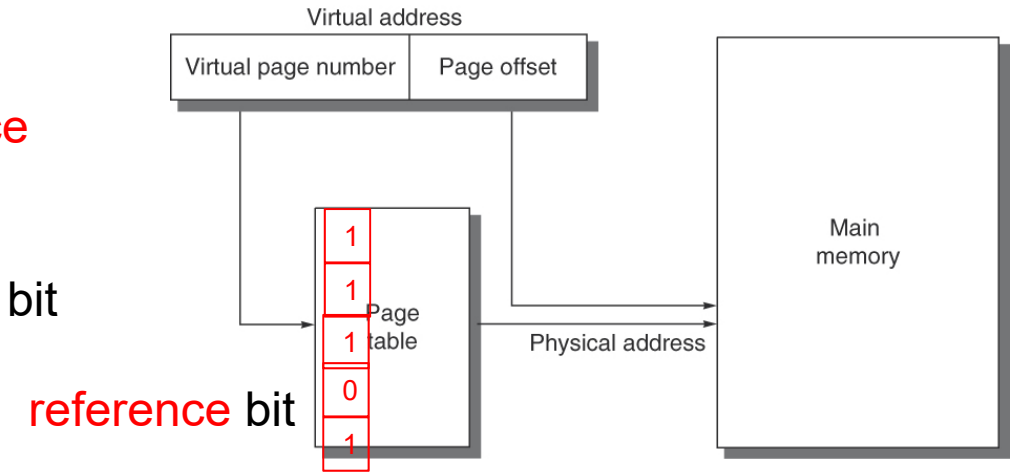
Page table contains the physical address of a page, and is indexed by the virtual page number.



Four Memory Hierarchy Question Revisited

- Q3: Which Block should be replaced on a Virtual Memory miss

- Almost all OS used LRU
- To help OS estimate LRU, reference bit (use bit) is set when a page is access
- OS periodically clear the reference bit



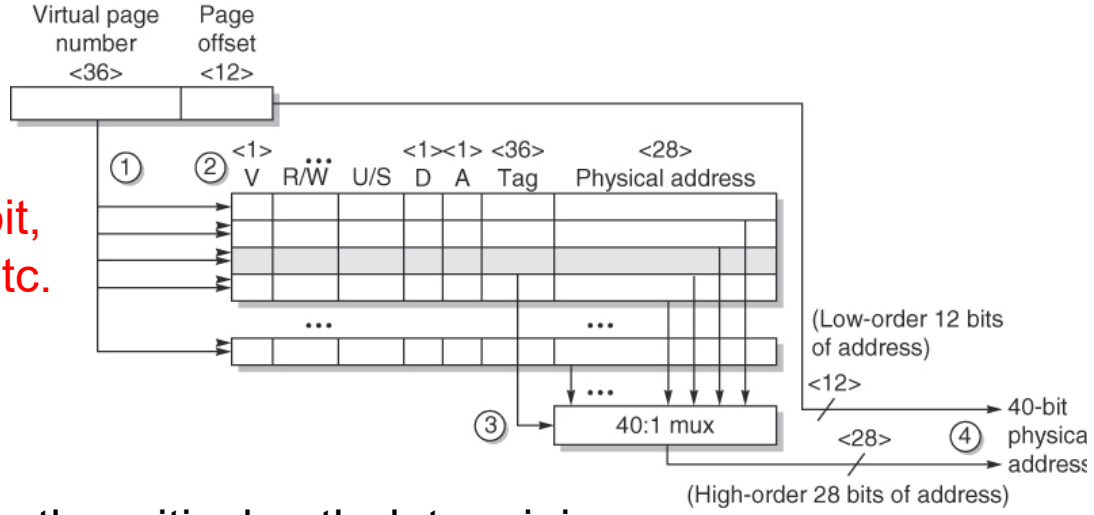
- Q4: What happened on a write

Write back. Writing to disk incur significant penalty

Technique for fast address translation

- Page table is stored on main memory.
- Paging requires additional memory accesses for each memory access
  - One for accessing page table
  - One for accessing memory
- Translation look aside buffer (TLB): A special cache that keep the address translation

In a TLB entry
Tag: virtual address
Data include physical address, valid bit,
Protection field, use bit, Dirty bit, etc.
(More details in OS textbook)



Address translation can easily be on the critical path determining the clock cycle of the process, so the Opteron uses virtually addressed, physically tagged L1 caches

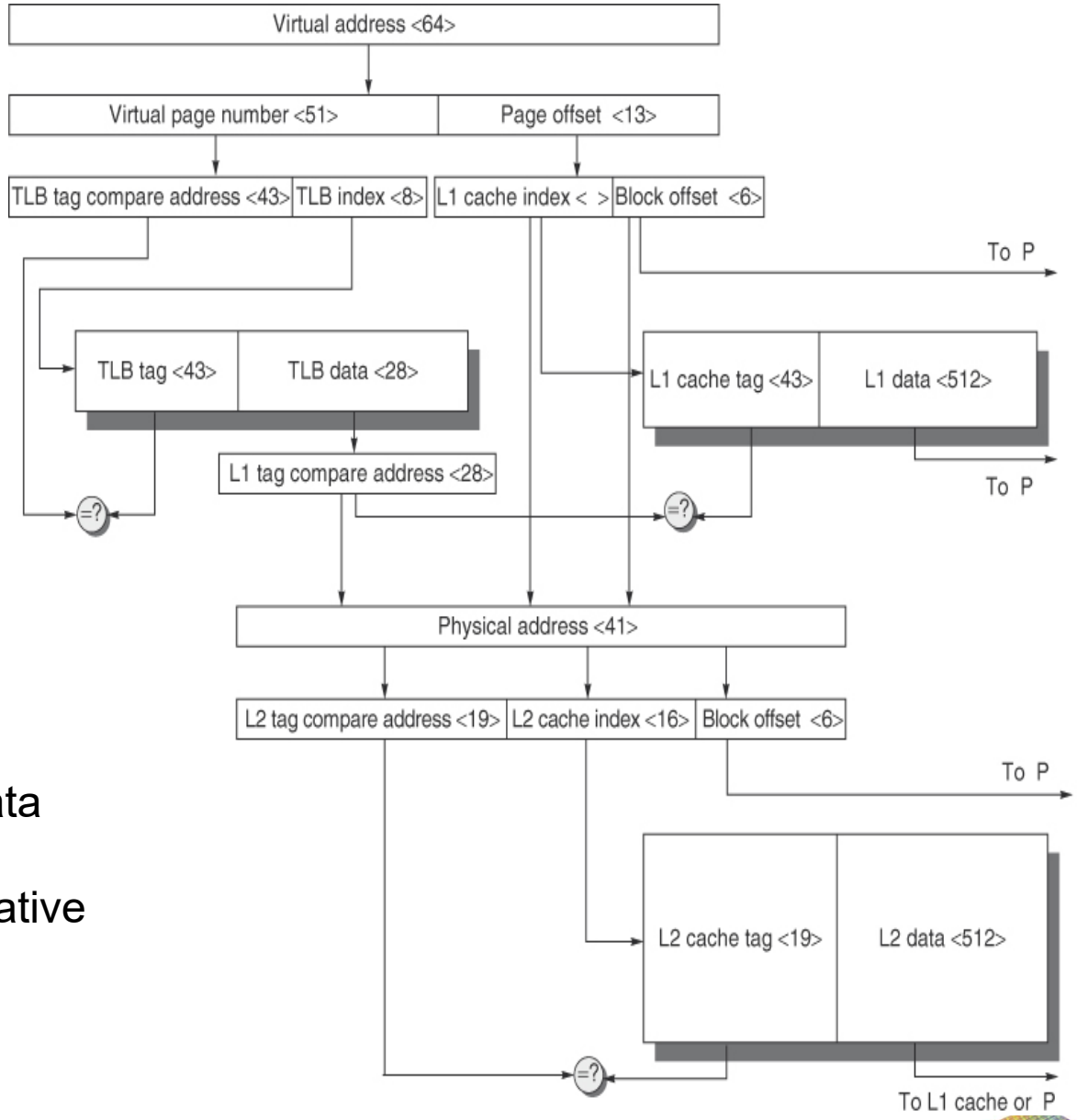
Example

Virtual/Physical address :64/41 bits

L1: virtually indexed, physically tagged cache

Cache size and page size are both 8K

L2 cache: 4M
Block size for L1/L2: 64 bytes

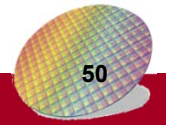


In the real world,

1. There are instruction and data caches
2. Cache can be n-way associative
3. TLB can be fully associative

Summary

- Read Appendix B1-B4



Backup slides