CS 222 Computer Organization & Architecture

Lecture 26 [02.04.2019]

Block Replacement & Write Strategy

John Jose

Assistant Professor

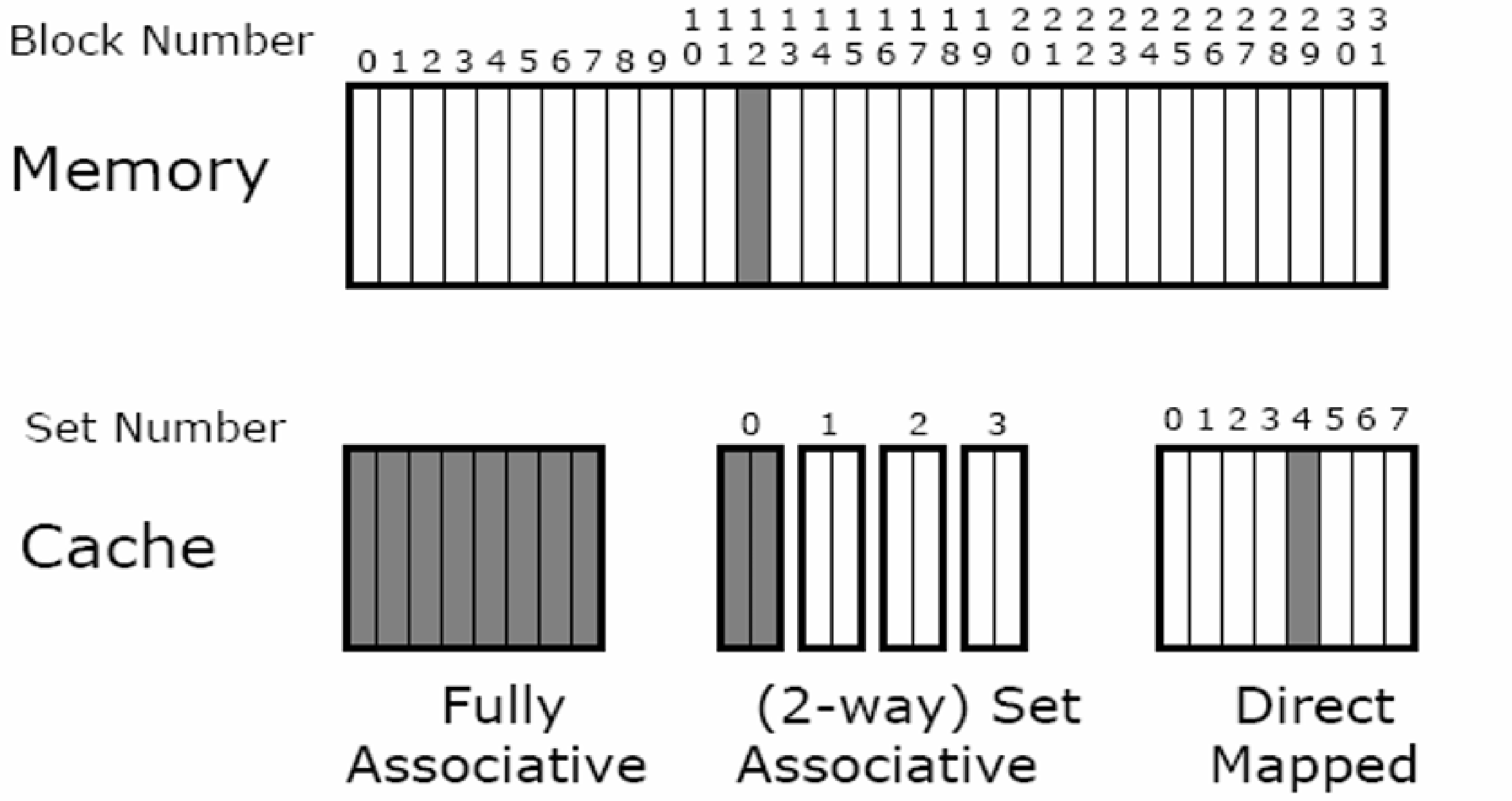
Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Four cache memory design choices

- **Where can a block be placed in the cache?**
  - **Block Placement [Mapping]**
- **How is a block found if it is in the upper level?**
  - **Block Identification**
- **Which block should be replaced on a miss?**
  - **Block Replacement**
- **What happens on a write?**
  - **Write Strategy**

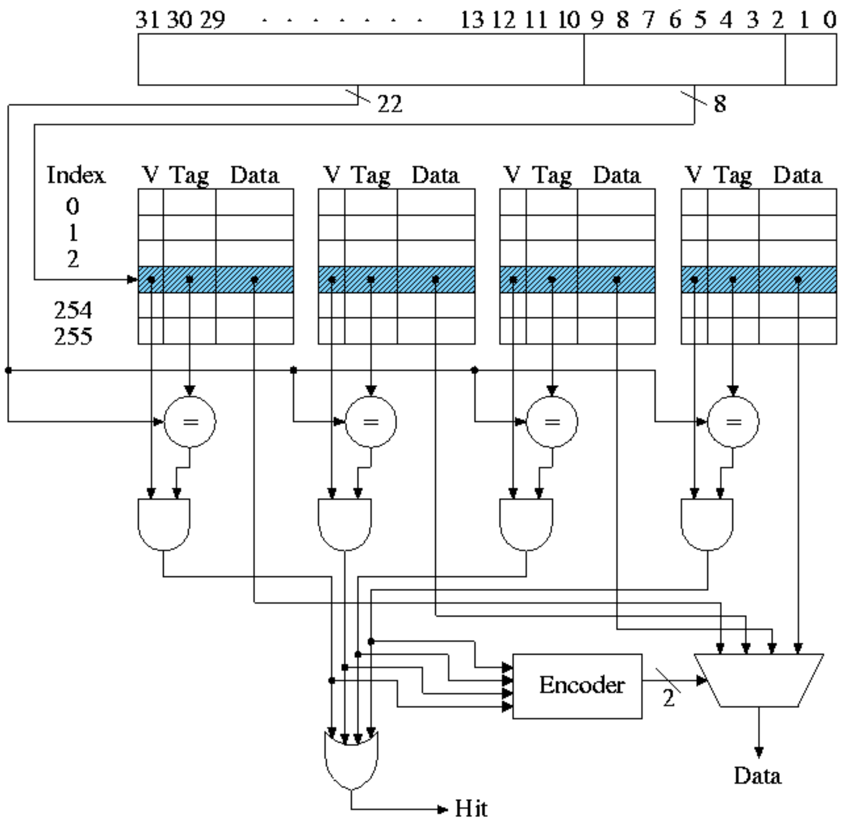
Block Placement



block 12 can be placed	anywhere	anywhere in set 0 *(12 mod 4)*	only into block 4 *(12 mod 8)*

Block Replacement

- Cache has finite size. What do we do when it is full?
- Direct Mapped is Easy
- Which block to be relaced for a set associative cache?



Block Replacement Algorithms

- ❖Random
- ❖ First In First Out (FIFO)
- ❖ Last In First Out (LIFO)
- ❖ Least Recently Used (LRU)
- ❖ Pseudo-LRU (PLRU)
- ❖ Not Recently Used (NRU)
- ❖ Least Frequently Used (LFU)
- ❖ Re-Reference Interval Prediction (RRIP)
- ❖ Optimal

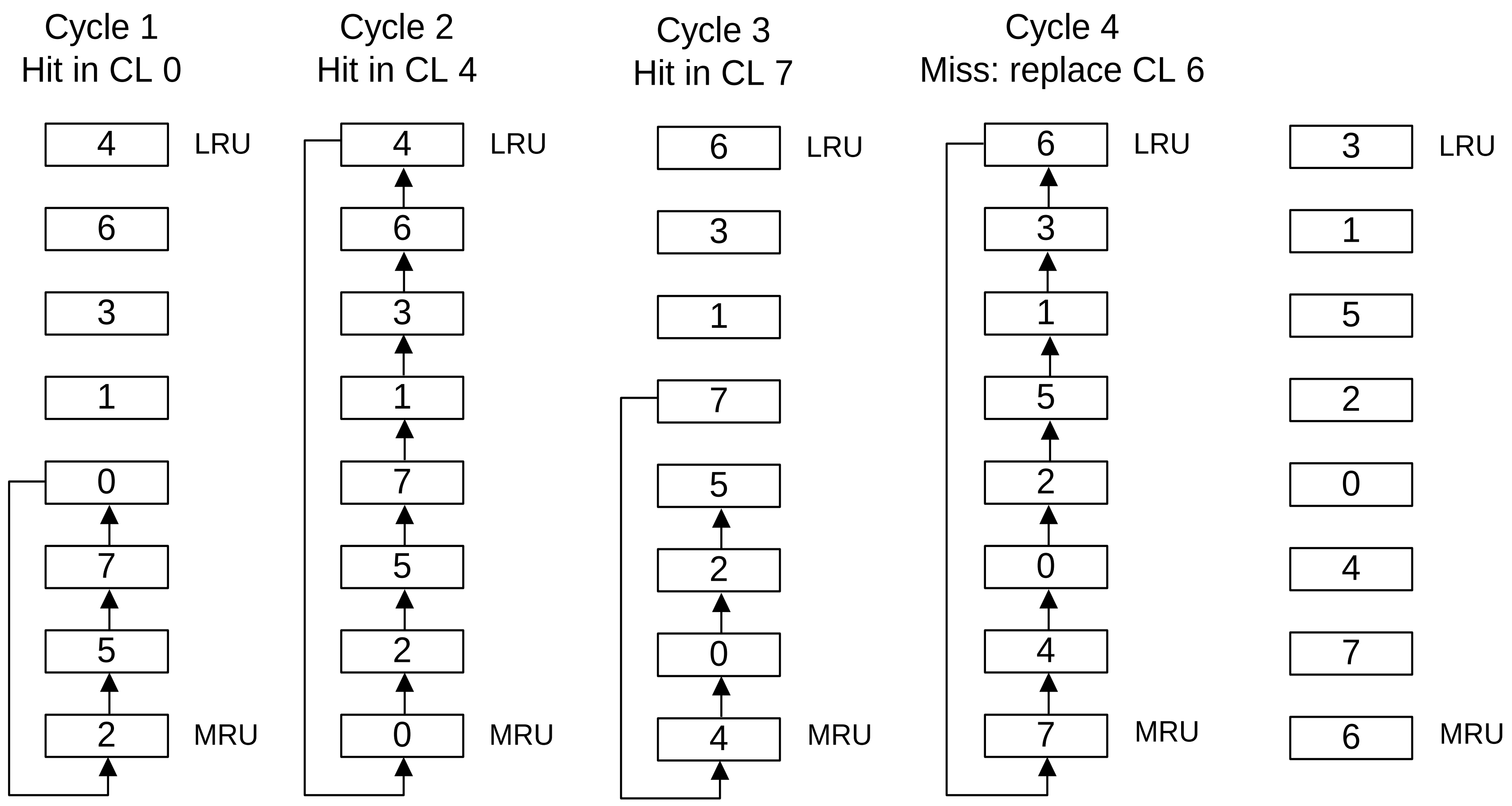
Random & FIFO Replacement Policy

- ❖ **Random policy** needs a pseudo-random number generator
- ❖ Makes no attempt to take advantage of any temporal or spatial localities
- ❖ **First-in, First-out(FIFO) policy** evict the block that has been in the cache the longest
- ❖ It requires a queue Q to store references
- ❖ Blocks are enqueued in Q, dequeue operation on Q to determine which block to evict.

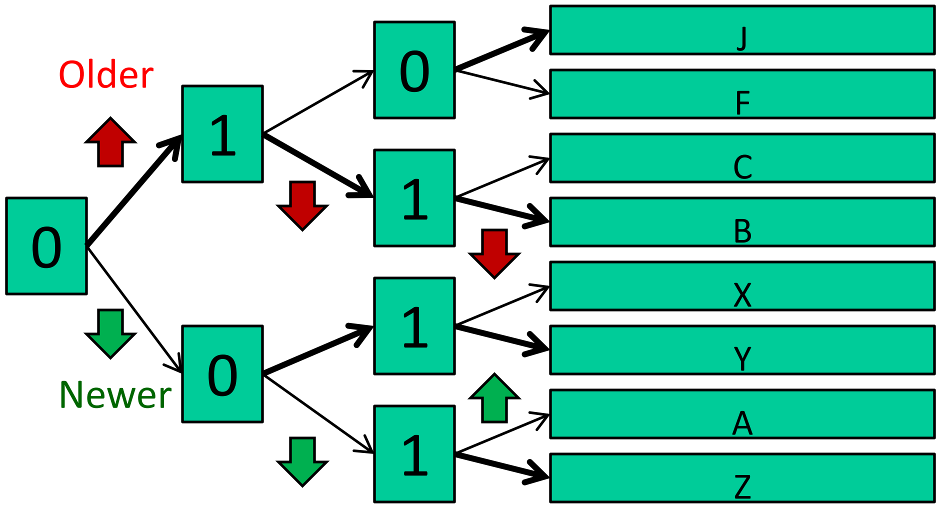
Least-Recently Used

- For associativity =2, LRU is equivalent to NMRU
  - Single bit per line indicates LRU/MRU
  - Set/clear on each access
- For a>2, LRU is difficult/expensive
  - Record Timestamps? How many bits?
  - Must find min timestamp on each eviction
  - Sorted list? Re-sort on every access?
  - Shift register implementation

LRU Implementation

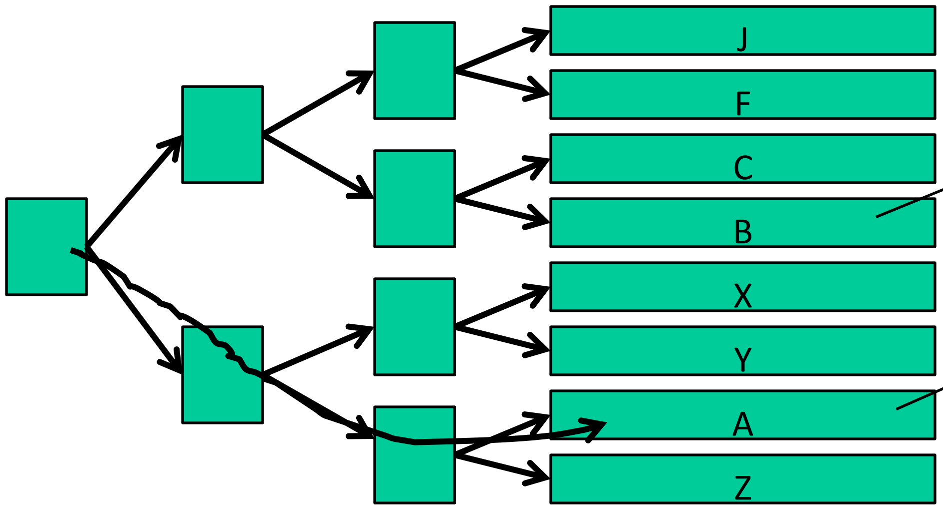


Practical Pseudo-LRU



- Rather than true LRU, use binary tree
- Each node records which half is older/newer
- Update nodes on each reference
- Follow older pointers to find LRU victim

Practical Pseudo-LRU



J Y X Z B C F A

011: PLRU
Block B is here

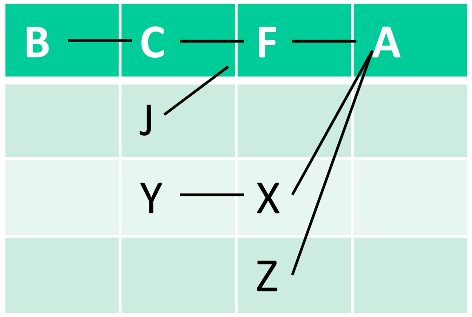
110: MRU
block is here

Partial Order Encoded in Tree:

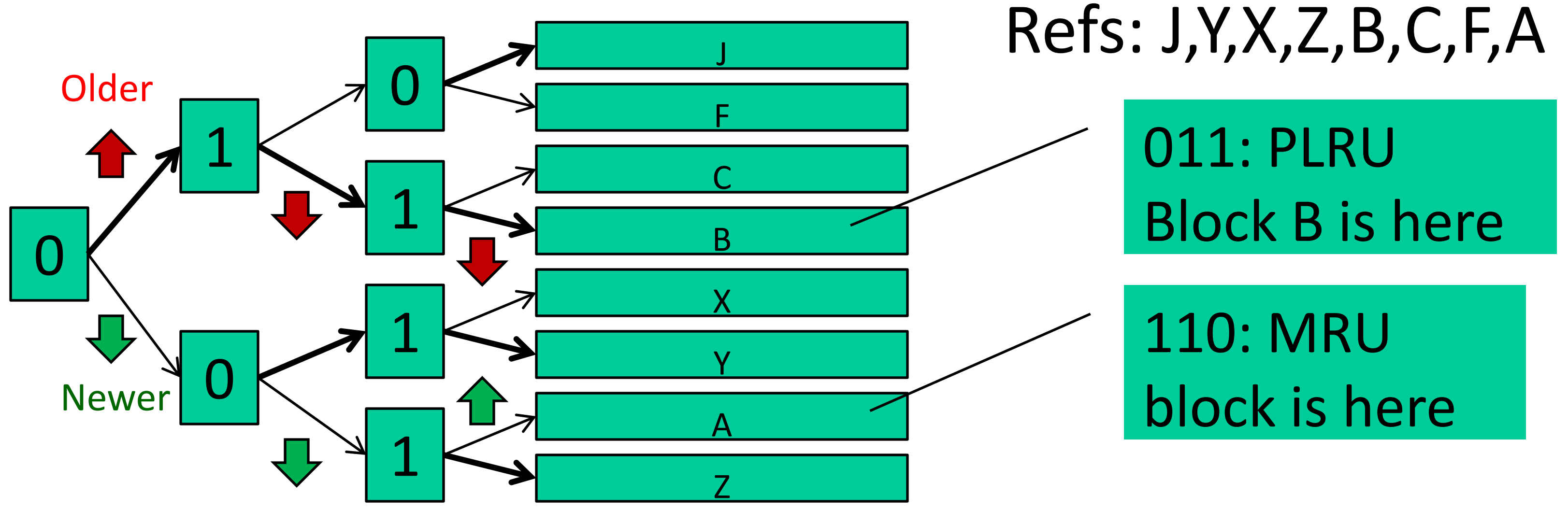
Z < A | Y < X | B < C | J < F

A > X | C < F

A > F



Practical Pseudo-LRU



- Binary tree encodes PLRU partial order
- At each level point to LRU half of subtree
- Each access: flip nodes along path to block
- Eviction: follow LRU path
- Overhead: *(a-1)/a* bits per block

Not Recently Used (NRU)

- Keep NRU state in 1 bit/block
    - Bit is reset to 0 when installed / re referenced
    - Bit is set to 1 when it is not referenced and other block in the same set is referenced
    - Evictions favor NRU=1 blocks
    - If all blocks are NRU=0 / 1 then pick by random
- Provides some scan and thrash resistance
- Randomizing evictions rather than strict LRU order

Least Frequently Used

- ❖ Counter per block, incremented on reference
- ❖ Evictions choose lowest count
- ❖ Comparison and sorting logic needed

Re-reference Interval Prediction

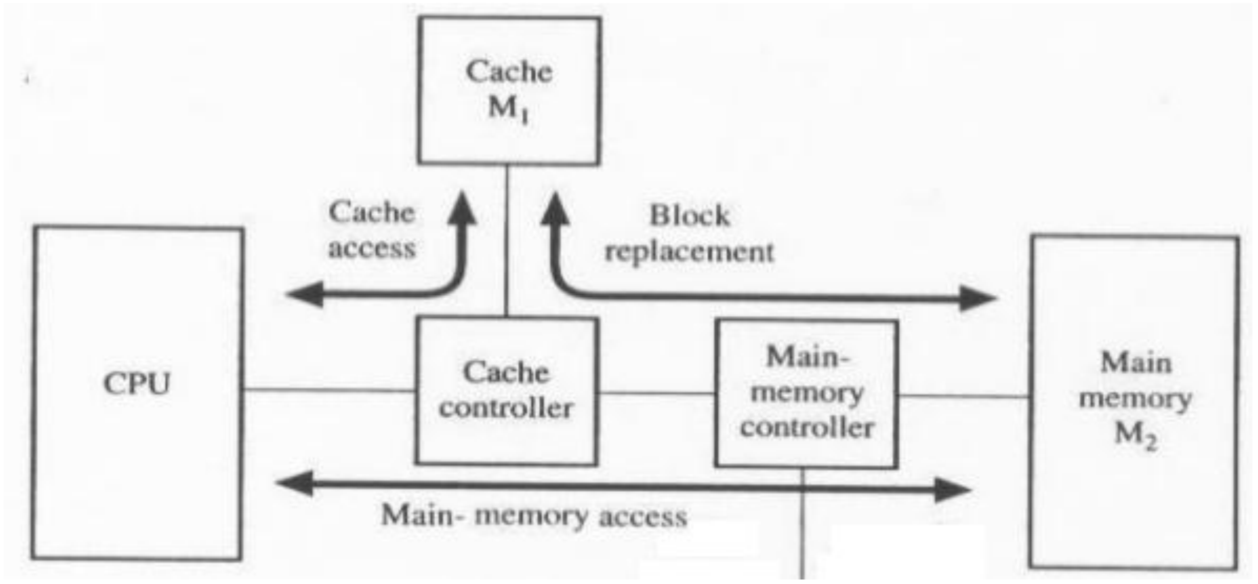
- RRIP
- Extends NRU to multiple bits
  - Start in the middle
  - promote on hit
  - demote over time
- Can predict near-immediate, intermediate, and distant re-reference

Optimal Replacement Policy

- Evict block with longest reuse distance
  - i.e. next reference to block is farthest in future
  - Requires knowledge of the future!
- Can’t build it, but can model it with trace
- Useful, since it reveals opportunity
- Optimal better than LRU
  - (X,A,B,C,D,X): LRU 4-way SA cache, 2nd X will miss

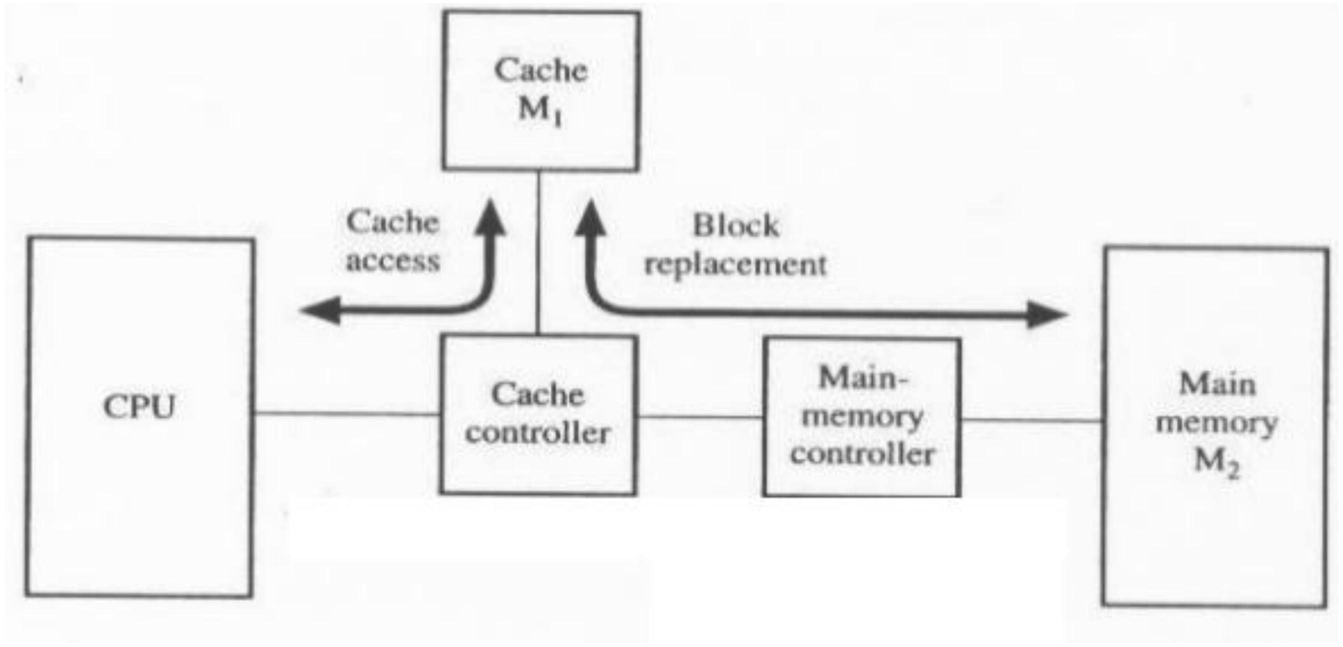
Look-aside vs Look through caches

- **Look-aside cache:** Request from processor goes to cache and main memory in parallel
- Cache and main memory both see the bus cycle
- On cache hit→ processor loaded from cache, bus cycle terminates; On cache miss: processor & cache loaded from memory in parallel



Look-aside vs Look through caches

- ❖ **Look-through cache:** Cache checked first when processor requests data from memory
- ❖ On hit→ data loaded from cache: On miss→ cache loaded from memory, then processor loaded from cache

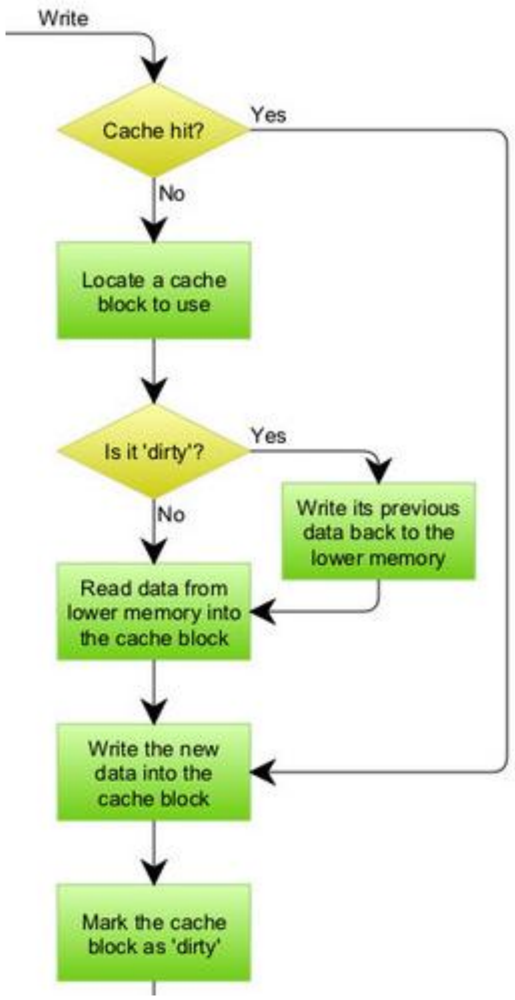


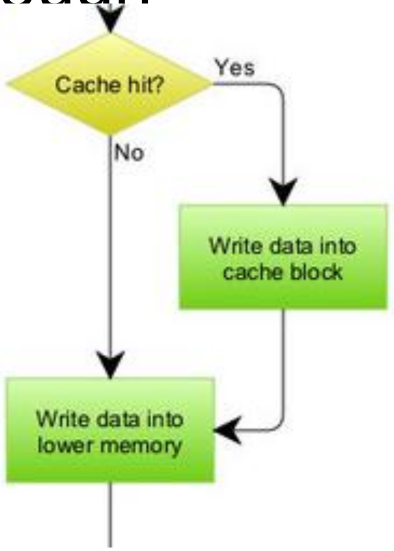
Write strategy

- **Write Hits → Write through vs Write back**
- **Write Miss→ Write allocate vs No-Write allocate**
- **Write through:** The information is written to both the block in the cache and to the main memory
- Read misses do not need to write back evicted line contents
- **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.
  - Have to maintain whether block clean or dirty.
  - No extra work on repeated writes; only the latest value on eviction gets updated in main memory.

Write strategy

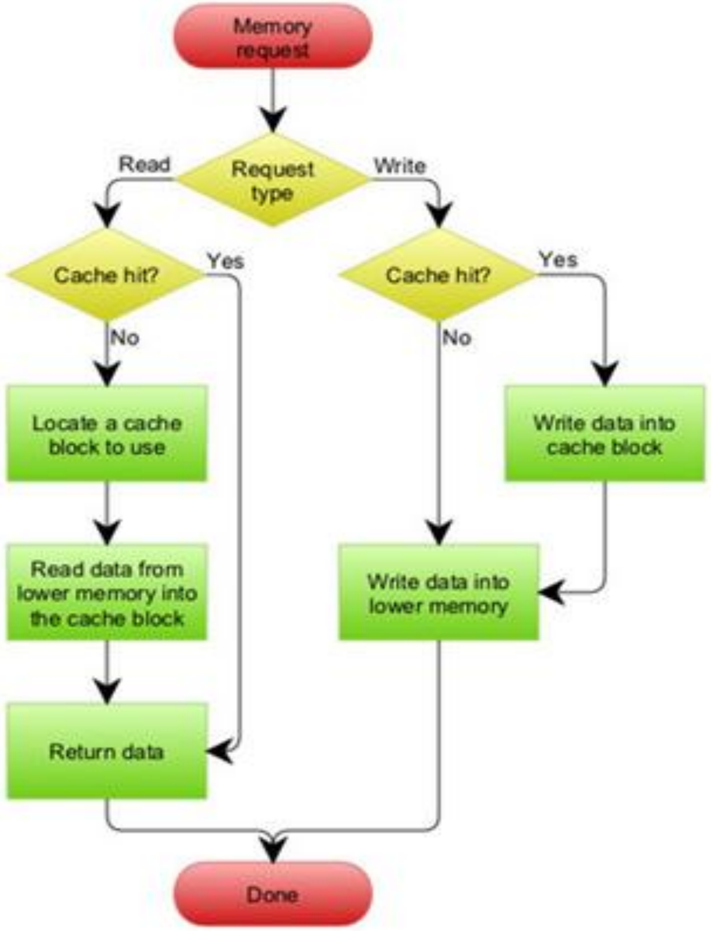
- **Write allocate:** The block is loaded into cache on a write miss.
- Used along with write back caches
- **No-Write allocate:** The block is modified in the main memory but not in cache
- Used along with write through caches


Write
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as 'dirty'



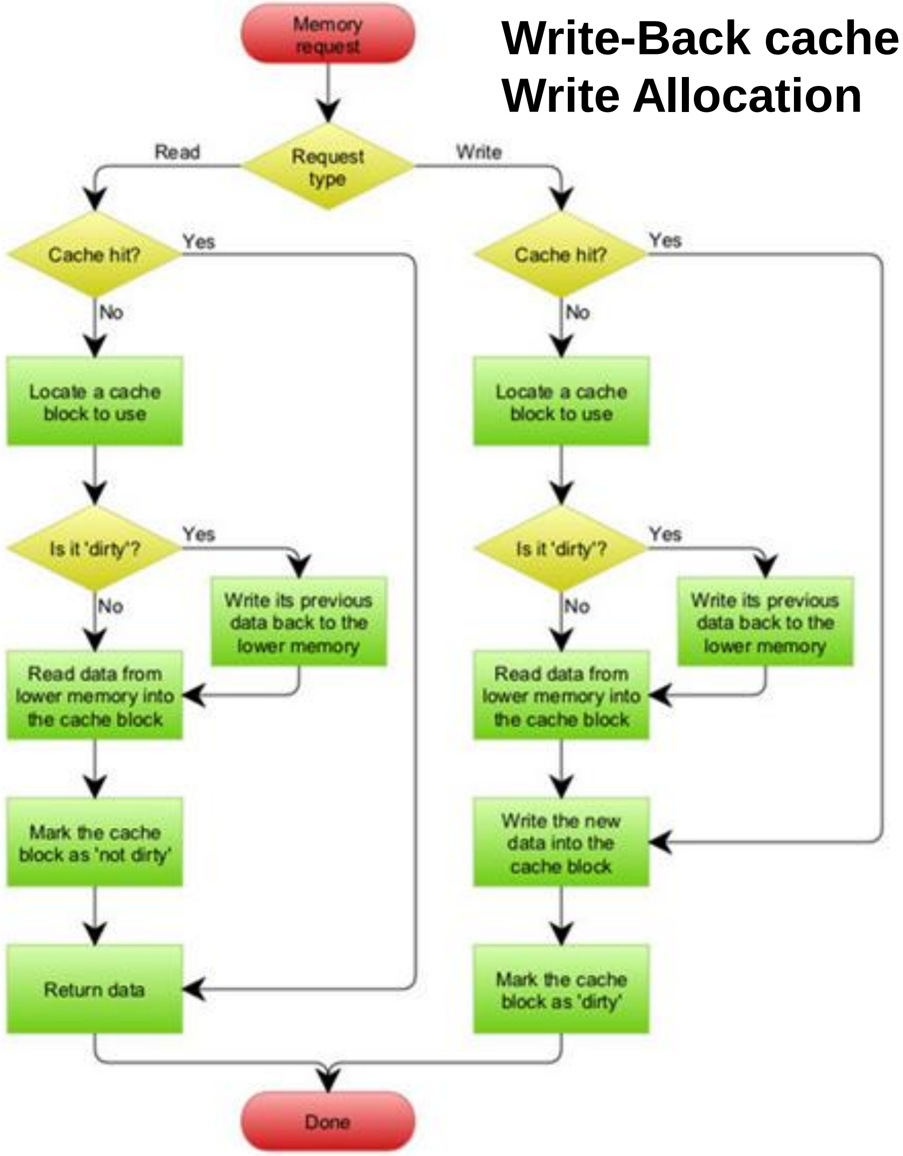
Write strategy

Write-Through cache with No-Write Allocation



Memory request
Request type
Read
Write
Cache hit?
Yes
No
Cache hit?
Yes
No
Locate a cache block to use
Write data into cache block
Read data from lower memory into the cache block
Write data into lower memory
Return data
Done

Write-Back cache with Write Allocation



Memory request
Request type
Read
Write
Cache hit?
Yes
No
Cache hit?
Yes
No
Locate a cache block to use
Locate a cache block to use
Is it 'dirty'?
Yes
No
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Read data from lower memory into the cache block
Mark the cache block as 'not dirty'
Write the new data into the cache block
Return data
Mark the cache block as 'dirty'
Done

Types of Cache Misses

- **Compulsory**
  - Very first access to a block
  - Will occur even in an infinite cache
- **Capacity**
  - If cache cannot contain all the blocks needed
  - Misses in fully associative cache (due to the capacity)
- **Conflict**
  - If too many blocks map to the same set
  - Occurs in associative or direct mapped cache

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/