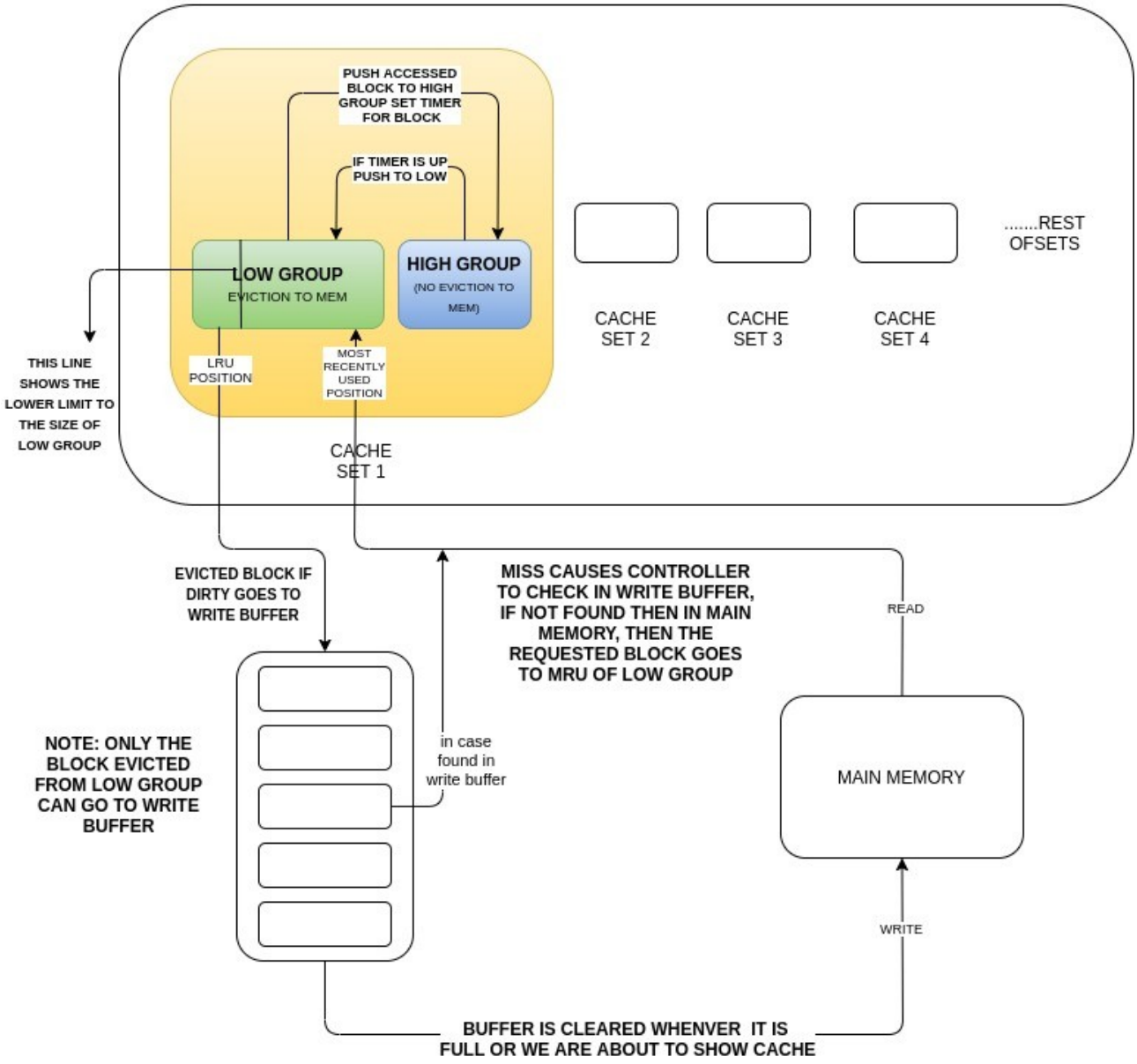
CACHE SIMULATOR

SAGAR SHARMA 2018CS10378

COL216 MAJOR

SIMULATED CACHE STRUCTURE



NOTE: Lower limit of low group is set to be 1/4 of set size. Sizes are dynamic.

ALGORITHM:

CASE:1 Initially all block are invalid

READ

1. If it is read, then it would definitely lead to a miss.
2. Search in buffer will lead to a miss
3. Search in main memory will return a new block valid, not dirty but with arbitrary data.
4. Returned block is push into low group of its corresponding set
5. Now there is a valid block at MRU position of low group

WRITE:

1. As all are invalid, it leads to a write miss
2. Block is searched in buffer which leads to miss
3. Then it is searched in memory which returns a new block which is valid and not dirty but with random data.
4. This block is inserted into MRU of low group and data is over written. The block is marked dirty. The evicted block from LRU of low is invalid hence discarded i.e not written in buffer.

CASE:2 INTERMEDIATE WORKING
SITUATION 1: WHEN LOW GROUP IS NOT WORKING AT LOWER LIMIT

READ:

1. If read block is in low group, its timer is set and it is transfered to MRU of high group.
2. If read block is in high group, its timer is reset and it is transfered to MRU of high group.
3. If it is a miss, block is searched in buffer.
4. If found it is pushed to MRU of low group, its dirty bit turned off. The evicted block from LRU position is then checked for dirty and written into buffer.
5. If not found then search in main memory.
6. If it was already there then it is pushed to MRU of low group with dirty bit off. Else a new block is returned. The evicted block from LRU of low group is checked for dirty and inserted into buffer.

WRITE:

1. If the block is in low group then it is over written and inserted into higher group with timer T.
2. If the block is in high group, its time is reset is pushed to MRU position.
3. If a miss occurs then the block is searched in buffer.
4. If found then the block is sent to low group MRU position and LRU block is evicted. Data is over written to the new block

and dirty is turned on. The evicted block if dirty goes to buffer.
5. If the block is not found in buffer, then searched in memory.
6. If found then the block is return with valid on, dirty off. It is sent to MRU position of low group and LRU block is evicted , checked for dirty and sent to buffer. The new block data is overwritten and marked dirty.

SITUATION 2: WHEN LOW GROUP IS AT LOWER LIMIT
NOW SIZE OF LOW GROUP CAN'T BE REDUCED.

READ:
1. If in lower group, send to high group but now as size of low can't reduce, send LRU block of high to MRU position of low.
2. If in high group simply promote it to MRU.
3. If a miss occur, find in buffer, push it MRU of low, evicted block is checked for dirty and pushed into buffer.
4. If not found in buffer look into memory do same as above.

WRITE:
1. If block found in low, overwrite and make it dirty. Push this block to high group, set timer and evict the LRU of high to MRU of low. This evicts LRU of low which is checked for dirty and send to buffer.
2. If block found in high simply overwrite, make it dirty and push to MRU position.
3. If not found, get it from write buffer and insert it into low group, over write make it dirty. The evicted LRU block of low is checked for dirty and inserted to buffer.
4. If not found in buffer, get from memory and do the same

IN SHORT MAINTAIN SIZE OF LOW GROUP IF IT IS GOING TO DECREASE.

NOTE: IN ALL THESE STEPS WHENEVER A READ OR WRITE OCCURS, MAINTAIN THE HIGH GROUP BY CHECKING ALL TIMERS AND SENDING THE STALE BLOCKS BACK TO LOW GROUP

HIGH AND LOW GROUPS DESIGN

My cache design is simple, no block is evicted from high group to buffer. Only blocks from low are sent to buffer. Thus if a block is in high group, it means it is accessed more and stays in cache for long.

The dynamic design helps in better runtime decision, accomodating more high priority blocks. The lower limit on low priority is necessary for easy working and correctness.

The size of high can thus vary between 0-(Size-low).

My buffer design is for write back scheme. Searching in buffer takes less time than in main memory. Collecting all write requests untill buffer becomes full takes more advantage of temporal locality. If a block is accessed just after it is written, we will be at advantage using a buffer.

ASSUMPTIONS:

1. When block is sent to buffer, neither valid nor dirty status change.
2. When block is written after being brought from memory or buffer, it is made dirty and valid.
3. Read block is only considered whenever it is valid.
4. After T access to cache all block in high group in all sets whose timer has run out, is pushed to low group.