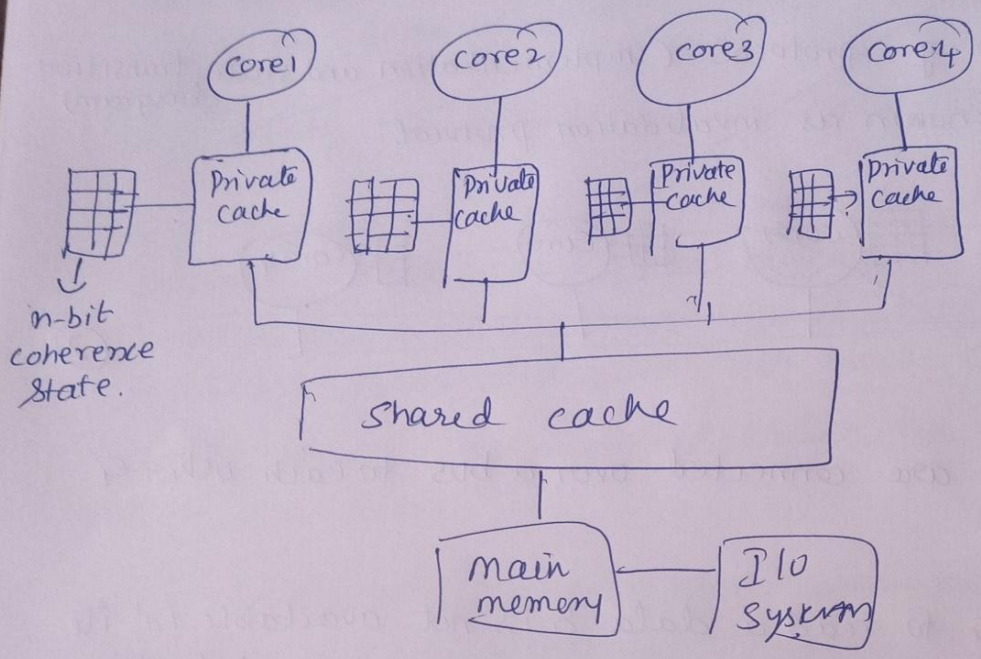
Bus Snooping protocol Implementation



Basics:-

1) In the diagram, a processor with multicore is considered A core is a micro cpu, an execution unit, executes an instruction

2) Initially data will be in main memory and cpu/core searches data for processing in its local cache, if not available, then searches in the shared cache

3) If the data is present in the shared cache, it supplies to the core requesting for it.

4) If in the shared cache, data is not there then search is made in main memory, brings the missing data back to shared cache and then the private cache of the corresponding requested core.
This consumes lot of cpu cycles

5) All the cores are connected over a bus to each other & with memory.

6) Any core wants to read a data & is not available in its cache, then it is going to place a transaction on the bus for Eg core 4 wants to access a data and is a miss, will place a transaction (indicating a miss) over the bus and all the other cache controller associated with multicore System will snoop on the bus and they take this transaction by checking the address and checks in the private caches / shared cache.

7) If any core has found that data, it responds by sharing/ the data based on the protocol.
Supplying

8) The state information about each cache block is maintained in the table like

| v | d | s | data | → cache block.

V :- valid bit, =1 if specifies that data stored in the block is valid i.e both cache and memory ~~for that~~ has the same value

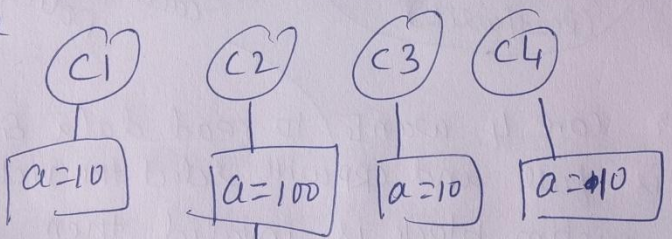
d :- dirty bit =1 indicated that data is modified/updated

S :- Shared bit, ~~the~~ =1 means, the data is shared across all the cores (Global data)
based on this a block can have different states

9) A state associated with each block can be

i) Shared :- As mentioned earlier, the data is shared across the cores

ii) modified :- The data block ~~which to be~~ updated by the core. Two cores can't update the same data at the same time so out of all cores for a particular data only one core can have modified state in it's private cache



only C2's cache can have modified state for a.

iii) when one core modifies the shared data, other cores which has the same copy, should change their state as invalidated in it's local copy

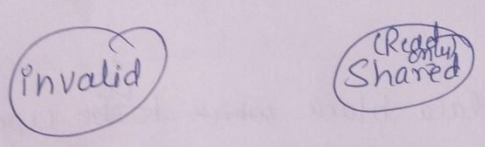
3 – State transition diagram for write back (4)

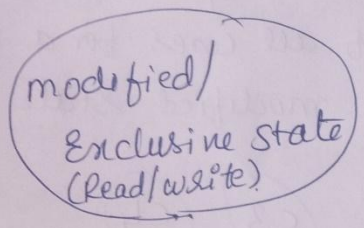
Remember : a) when cache is full, one of the victim block will be removed to keep new data.

b) write back cache policy is implemented.

Cache controller for each private cache updates the state of each block in response to processor and snoop events and updates the transaction.

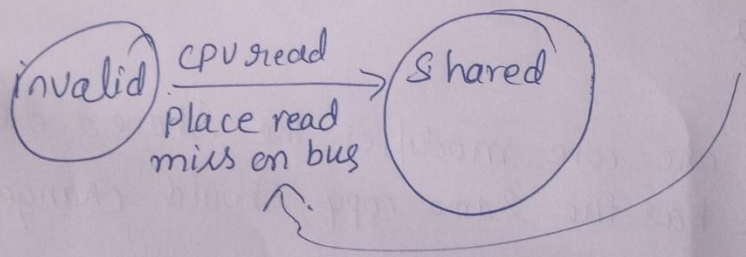
1) A cache can have 3 states



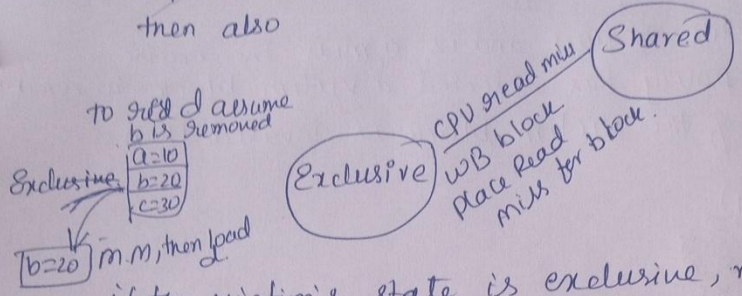


cache Replacement policy.

2) Consider core 4 wants to read data & it is a miss and cache is full and CRP will select the victim block & the state of victim block is invalid then state is changed ferom invalid to Shared & places the transaction read miss on the bus

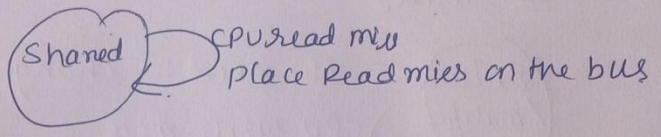


3) & CPU/core 4 read miss, victim's state is exclusive. then also



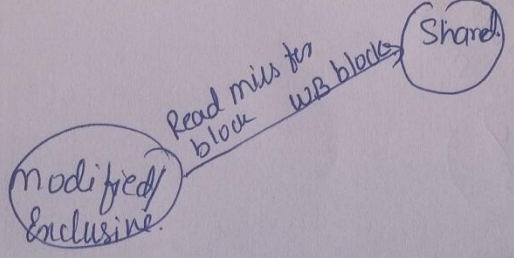
if the victim's state is exclusive, means d=1 it is updated therefore the updated data must be written back to the lower level cache.

4) CPU/core 4 Read miss & victim's block state is shared then



So for all the 3 states & for read miss, the state is changed to shared & transaction is placed on the bus.

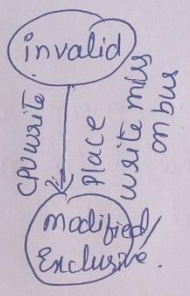
5) her core 4 request data & is miss and assume the cache of core 3 has that data & is in modified state then in ~~core~~ core 3 cache the state changes from



So now core 3 & core 4 cache has same data.

Now consider write operation

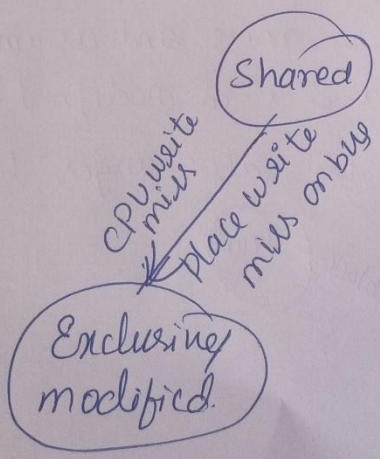
6) core 4 write request and is a miss the cache replacement policy has selected a victim block to remove and the state of the victim block is invalid then.



7) core 4 write request & is a miss & victim block's state is Exclusive

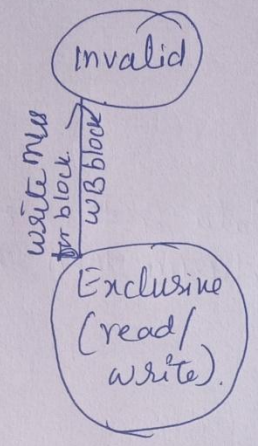
No need to change the state
CPU write miss
WB block: place write miss on bus

8) core 4 write request & is a miss & victim block's state is Shared

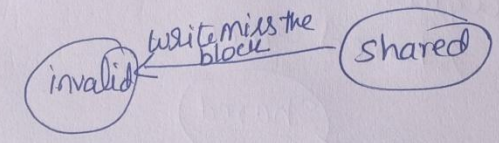


For all the write miss transaction the state changes to modified/Exclusive state

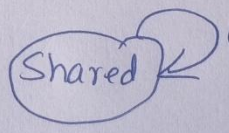
9) Now for a write miss, if any core has the data in an Exclusive ~~data~~ state Say for eg core 3 has then core 3 for that block changes the state as



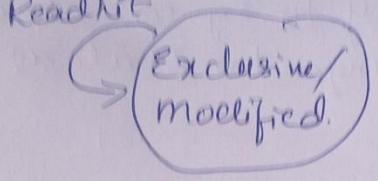
10) Assume only core 2 has a data which is in Shared state and other core has updated it, then core 2 cache state will change from



11) Another situation for Read hit.
Core 4 Read data & is a hit & state is shared then no need to change the state

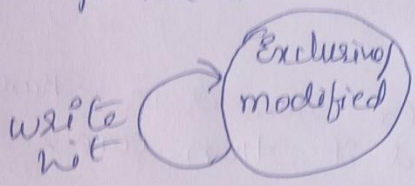


12) Core 4 read request for data & is a hit
& The state is Exclusive, then no need to change the state

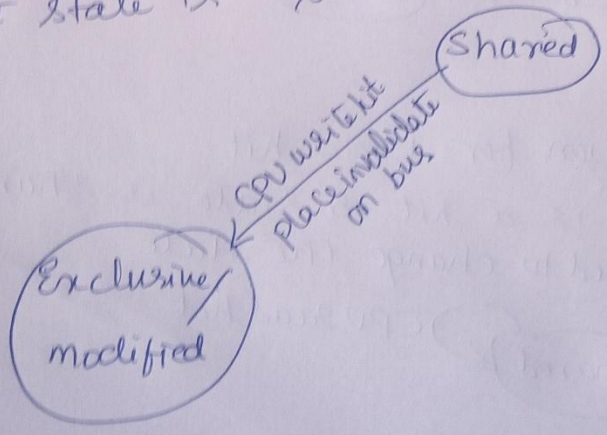


Now consider for write hit.

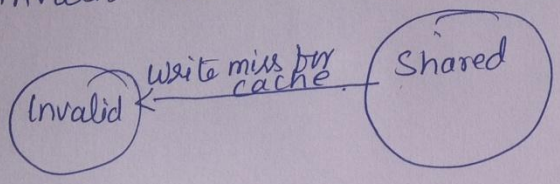
13) Core 4 request to write a data & is a hit in a blode whose state is Exclusive then no need to change the state



14) Core 4 request to write a data & is a hit but state is shared then



15) As a result of (14) if any core has that same copy of data then it has to move from shared to invalid state



This is about the 3-State write back invalidation protocol.

It ensures that coherence is maintained across all multicore system

Limitations :-

1) It is used for multicore s/m which has upto 8 cores
2) It is difficult to implement more than 8 cores