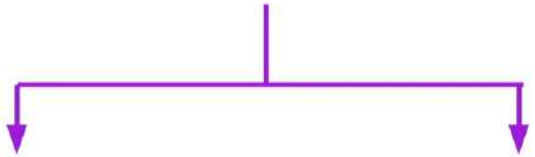
Shared Memory Multiprocessors

Coherence protocols
Ex: Dragon

Snooping protocols



Write invalidate

Write update

Write-update protocol

- When shared item is written, update all cached copies
- It must broadcast the write to all
- This uses considerable bandwidth, therefore most recent systems use write-invalidate protocols
- +ve avoids misses on later references
- -ve multiple useless updates

4-state (Dragon) write-back, Update protocol

- E=Exclusive
  - Only one cache (this one) has copy and it is not modified, i.e. Memory is up-to-date
- M=Modified
  - Exclusive ownership; Modified dirty; Present in this cache only; Memory is stale
- Sc=Shared-Clean
  - Potentially two or more caches (including this one) have the block
  - Memory may/may-not be up-to-date
  - When in Sc, there could be block in other cache in Sc or Sm state. Therefore Memory may not be up-to-date
  - If no other Sm then Memory is up-to-date
- Sm=Shared-Modified

4-state (Dragon) write-back, Update protocol

- E=Exclusive
- M=Modified
- Sc=Shared-Clean
- Sm=Shared-Modified
  - Potentially two or more caches (including this one) have the block
  - Memory is NOT up-to-date
  - Memory updated by this cache on replacement
  - Block in Sm state in only-one cache at a time
  - Possible that one Sm and some Sc states
  - Possible that no Sm and some Sc states

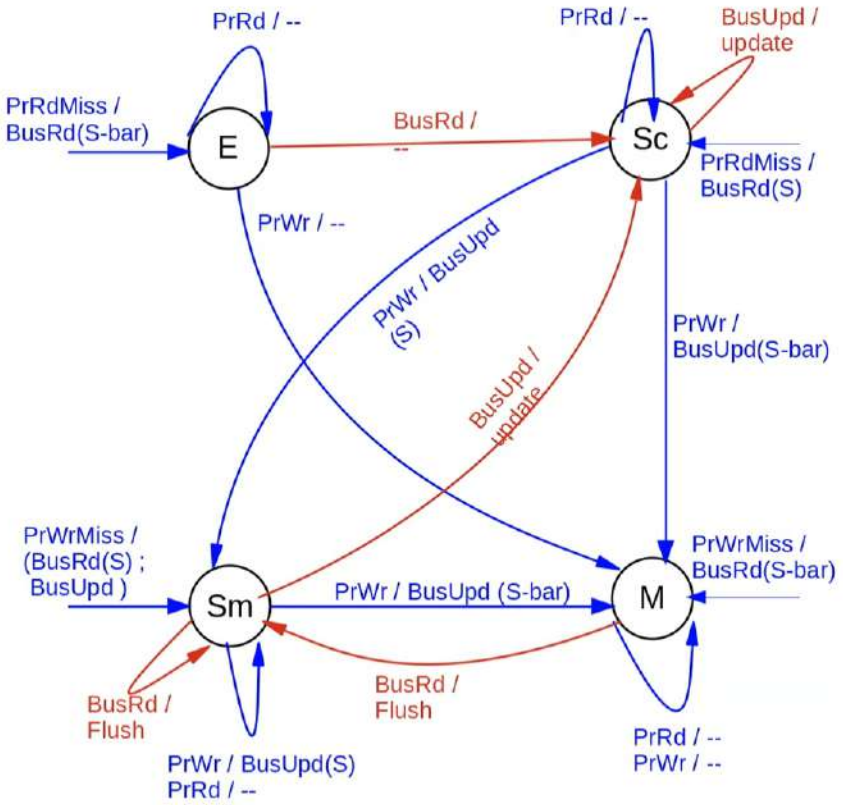
4-state (Dragon) write-back, Update protocol

- E=Exclusive
- M=Modified
- Sc=Shared-Clean
- Sm=Shared-Modified
- I = ?
  - There is no explicit 'I' Invalid state
  - Since it is update protocol, cache can keep copy until it gets replaced
  - If tag matches, copy gets updated
  - However, if block not in cache, then it is special invalid state
  - Initially load-cache (Read) then normal protocol

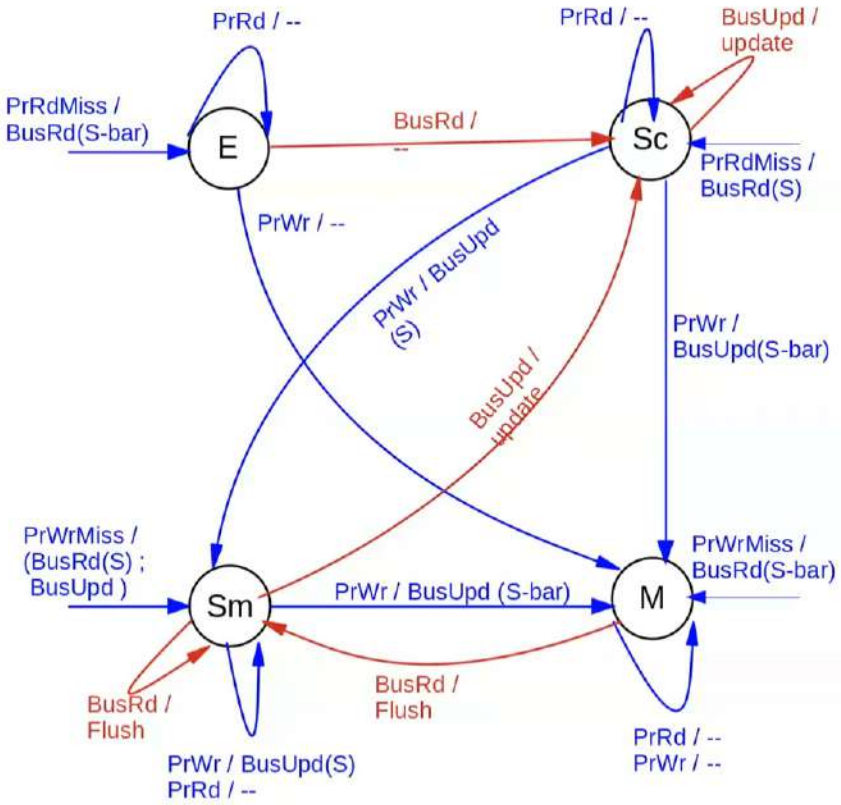
4-state Dragon : Transitions

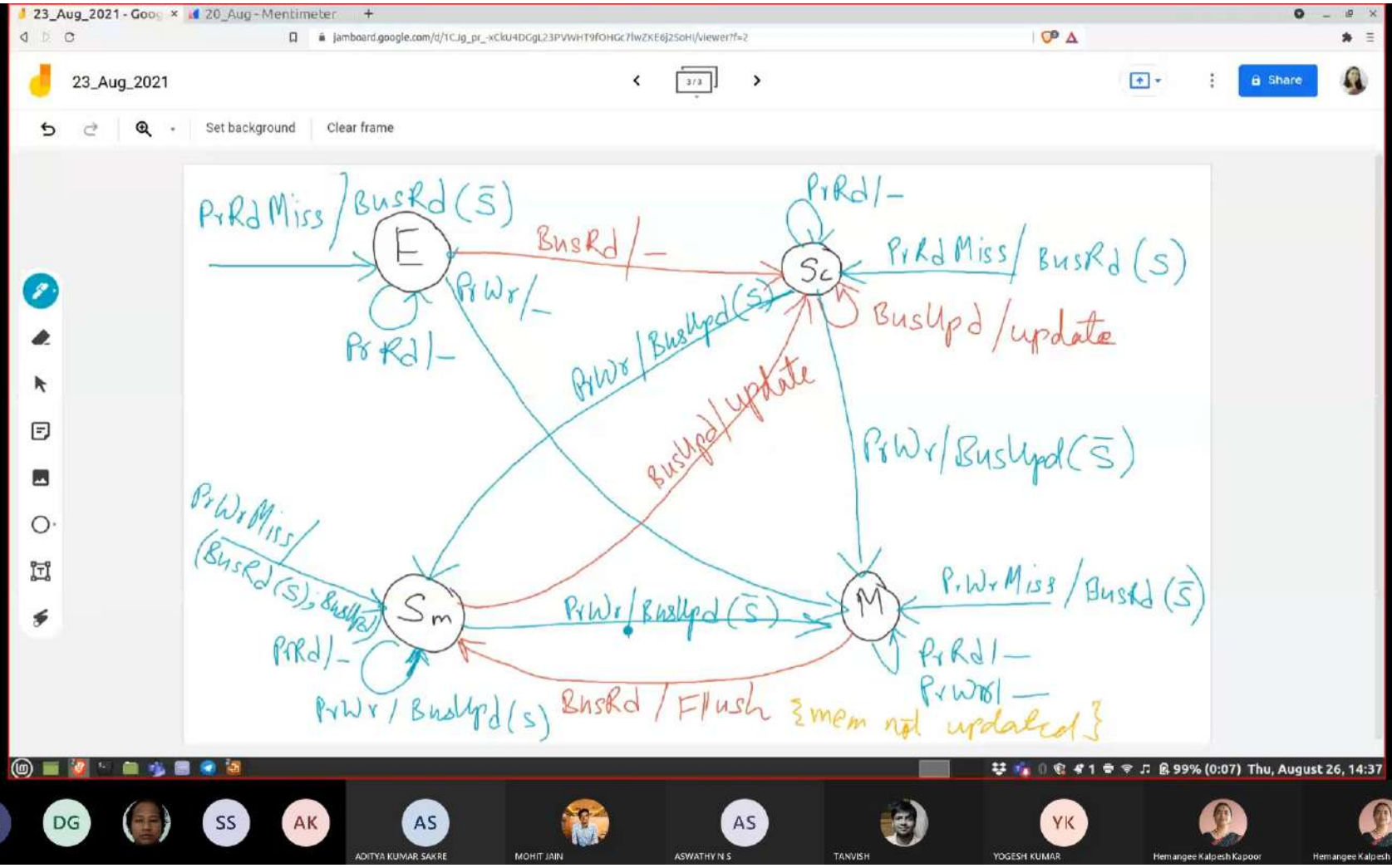
- Processor request + Bus Transactions
  - PrRd
  - PrWr
  - PrRdMiss {as no 'I' state specify actions on tag mis-match}
  - PrWrMiss {as no 'I' state specify actions on tag mis-match}
  - BusRd
  - BusWr
  - BusUpd: takes the specific word (or bytes) written by the processor and broadcasts it on the bus, so that other caches get updated. We hope that this limited data transfer saves bandwidth
- Similar to MESI, a shared signal (S) is available to the cache controller (wired-OR)
- New capability for cache controller to update contents when BusUpd is seen on the bus

4-state Dragon - FSM



4-state Dragon - FSM




23_Aug_2021 - Goog
20_Aug - Mentimeter
jamboard.google.com/d/1CJg_pr_-xCkU4DGgL23PVWHT9fOHGc7lwZKE6j2SoHI/viewer?f=2
23_Aug_2021
3 / 3
Share
Set background
Clear frame
PrRd Miss / BusRd (S̄)
E
BusRd / —
PrRd / —
Sc
PrRd Miss / BusRd (S)
PrWr / —
PrRd / —
BusUpd / update
PrWr / BusUpd (S)
BusUpd / update
PrWr / BusUpd (S̄)
PrWr Miss / (BusRd (S); BusUpd)
Sm
PrWr / BusUpd (S̄)
M
PrWr Miss / BusRd (S̄)
PrRd / —
PrWr / —
PrRd / —
PrWr / BusUpd (S)
BusRd / Flush {mem not updated}
99% (0:07) Thu, August 26, 14:37
+28
DG
SS
AK
AS
ADITYA KUMAR SAKRE
MOHIT JAIN
AS
ASWATHY N S
TANVISH
YK
YOGESH KUMAR
Hemangee Kalpesh Kapoor
Hemangee Kalpesh Kapoor

State Transitions

- Read Miss
- Write
- Write Miss
- Replacement

- Read Miss
  - Generate BusRd then goto ‘E’ or ‘Sc’
  - If S-bar: load in E; Memory gives data (as no sharer)
  - If S: load in state Sc (as other sharer present)
  - (1) IF other cache has block in M or Sm
    - It provides data
    - This cache becomes Sc. Other cache M --> Sm
  - (2) IF other cache is Sc
    - Memory provides data
    - This cache Sc. Other cache remains Sc

State Transitions

- Read Miss
- Write
- Write Miss
- Replacement

- Read Miss
  - Generate BusRd then goto ‘E’ or ‘Sc’
  - If S-bar: load in E; Memory gives data (as no sharer)
  - If S: load in state Sc (as other sharer present)
  - (1) IF other cache has block in M or Sm
    - It provides data
    - This cache becomes Sc. Other cache M --> Sm
  - (2) IF other cache is Sc
    - Memory provides data
    - This cache Sc. Other cache remains Sc

State Transitions

- Read Miss
- Write
- Write Miss
- Replacement

- Write
  - Local state M => no action
  - Local state E => next state M ; No further action
  - Local state Sc or Sm => generate BusUpd => next state Sm or M
    - (1) IF other cache having block they update data and become Sc (if necessary). Local next state Sm
    - (2) IF other cache do NOT have copy => Local next state M
  - Memory not updated

State Transitions

- Read Miss
- Write
- Write Miss
- Replacement

- Write Miss
  - On a write, if block not present in the cache, it is treated as read-miss followed by write
  - Thus first BusRd is generated
  - Then if block is found in other caches, a BusUpd is generated; Block loaded locally in Sm state
  - If block not in other cache, load locally in M state

State Transitions

- Read Miss
- Write
- Write Miss
- Replacement

- Replacement
  - On a replacement (arcs not shown in figure) the block is written back to memory if it is in M or Sm state
  - If block in Sc
    - Some other cache has in Sm => that cache will update memory
    - No cache has block => memory is already up-to-date

Lower level design choices

- Can shared modified state (**Sm**) be eliminated? (done by DEC Firefly multiprocessor)

- Should Replacement of **Sc** be broadcast?

Lower level design choices

- Can shared modified state (**Sm**) be eliminated? (done by DEC Firefly multiprocessor)
  - YES, if we update other caches and Memory on BusUpd
  - Underlying assumptions for
    - Keeping Sm = SRAM fast, DRAM slow
    - Not keeping Sm = DRAM is fast enough
- Should Replacement of **Sc** be broadcast?
  - This will allow the last copy to go to 'E' or 'M' and save generating update transactions
  - Also good, as this replacement transaction is NOT on the critical-path of a memory operation, whereas the later bus transactions that it saves might be on the critical-path (i.e. each BusUpd is on critical-path)

Proofs: coherence + consistency

- Proof same as write-through case
- All writes appear on the bus in update protocol
- Therefore, write-serialisation, write-completion detection and write-atomicity are straight forward with an atomic bus

Example: Dragon

Processor Action	State in P1	State in P2	State in P3	Bus Action	Data Supplied by
1) P1 reads u					
2) P3 reads u					
3) P3 writes u					
4) P1 reads u					
5) P2 reads u					

Fill the Table

Example: Dragon

Processor Action	State in P1	State in P2	State in P3	Bus Action	Data Supplied by
1) P1 reads u	E	--	--	BusRd	Memory
2) P3 reads u	Sc	--	Sc	BusRd	Memory
3) P3 writes u	Sc	--	Sm	BusUpd	P3 cache
4) P1 reads u	Sc	--	Sm	null	--
5) P2 reads u	Sc	Sc	Sm	BusRd	P3 cache

Types of Cache Misses

Cache Miss Types

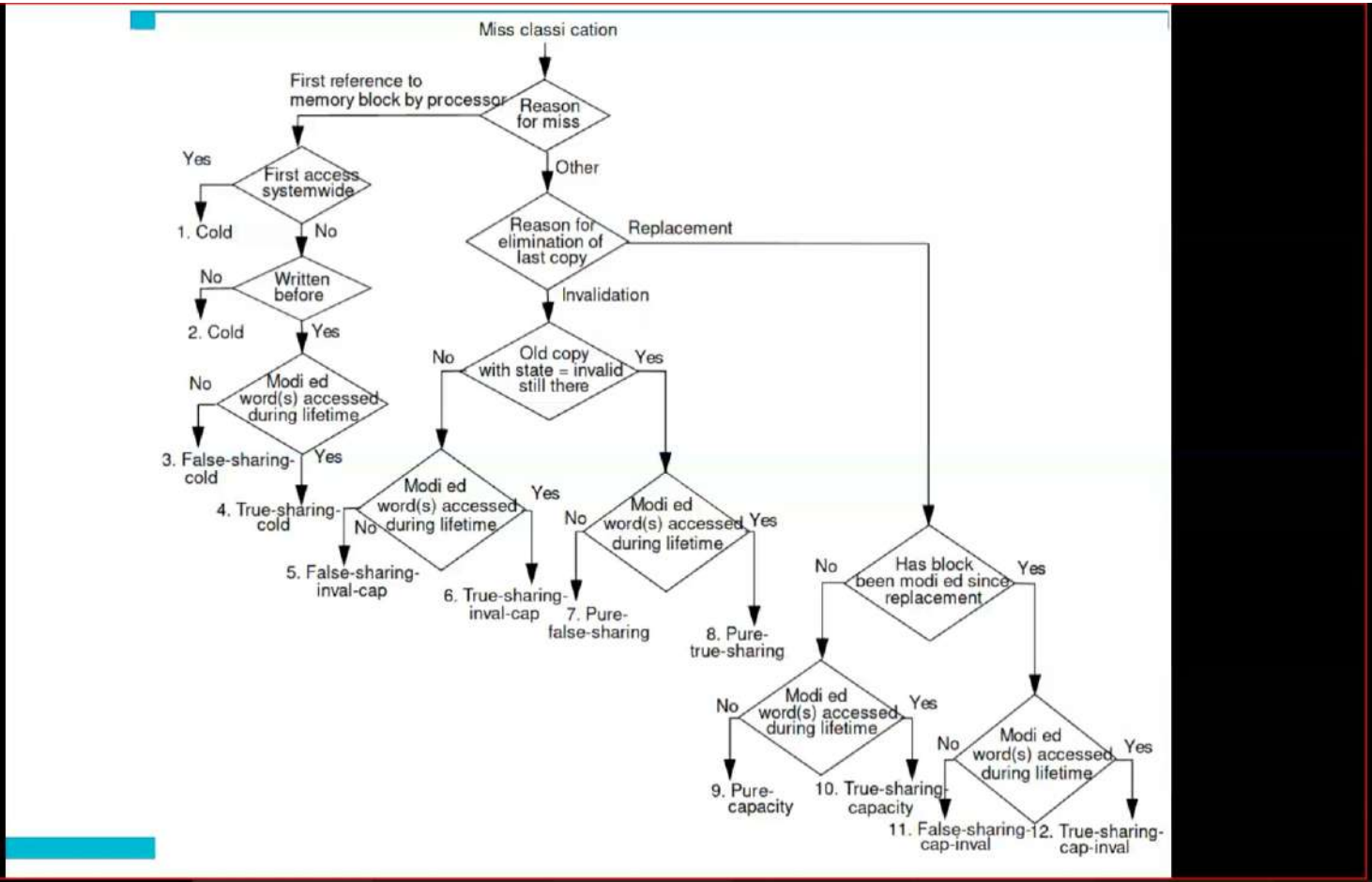
- Cold miss --- occurs on the first reference to a memory block by a processor. (compulsory miss)
- Capacity miss --- occurs when all the blocks that are referenced during the execution of a program do not fit in the cache
- Conflict miss --- occurs in caches with less than full associativity, i.e., the referenced block does not fit in the set. (Collision miss)
- Coherence miss --- occurs when blocks of data are shared among multiple processors
  - **True sharing**: a word in a cache block produced by one processor is used by another processor
  - **False sharing**: words accessed by different processors happen to be placed in the same block

Ex: Sharing misses

- Cache block is the granularity of data fetch from memory as well as for coherence
- True Sharing Miss
  - One processor writes some words in a cache block
  - The same block in other processors is invalidated
  - The second processor reads one of the modified words (read miss)
  - Called “true” because the miss truly communicates newly defined data values that are used by the second processor => these are essential for program correctness
- False Sharing Miss
  - One processor writes some words in a cache block
  - The same block in other processors is invalidated
  - The second processor reads a different word in the same cache block

How to reduce sharing misses?

- True Sharing Miss
  - Inherent to the parallel decomposition and assignment
  - Similar to cold misses, true sharing miss can be reduced by increasing the block size and the spatial locality of the communicated data
  - Bigger the block, more shared items fit in them and hence in one invalidation many shared data items can be handled
- False Sharing Miss
  - Increases as the cache block size increases
  - Would not occur if the cache block size is one word
  - Current trend is enlarging the cache block size, which potentially increases false sharing misses



Miss classification