Propagating coherence transactions in the hierarchy

- Intra hierarchy protocol

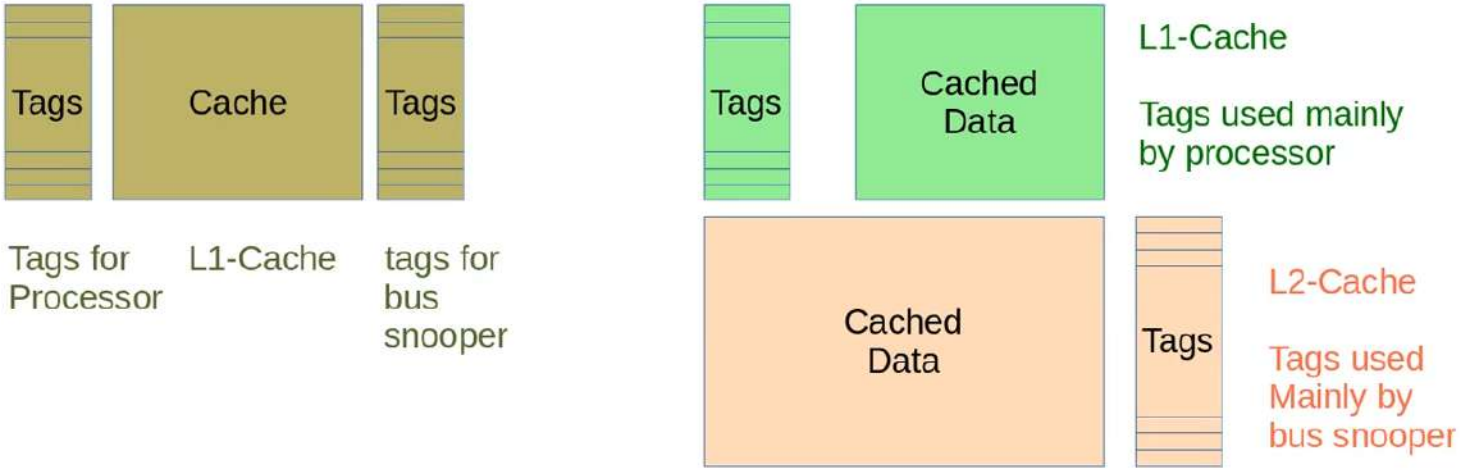
(1) Processor --> out

- Processor requests percolate downwards until they reach a cache in proper state or reach the bus
- Response to proc-req goes in up-direction to processor
- PrRead: send block to all levels
- PrWrite: send block. Nearest (L1) has update copy. Remaining levels dirty-but-invalid

(2) Bus --> In

- Bus-req percolate upwards and modify cache states on the way
- For flush or copy-back req, they travel till a block having update value is found
- Inv have to reach all levels
  - **Optimisation**: cache close to bus commits invalidation and releases bus while the other caches are invalidated in the background
  - Optimisation is OK for single transaction bus !

Propagating transactions ..



- Dual tags are not required as L2 filter reduces contention for L1 tags

Correctness

- Major correctness issues do not change much in multi-level caches as long as inclusion is maintained
- The necessary transactions are propagated up and down the hierarchy
- Bus transactions may be held up until the necessary propagation occurs
- Of course there is performance penalty for holding up the bus until the response is obtained !
  - We will relieve this constraint later ...

What we will see in this topic?

(1) Correctness requirements

(2) Single-level cache + Single transaction Atomic Bus

(3) Multi-level cache + Single transaction Atomic Bus

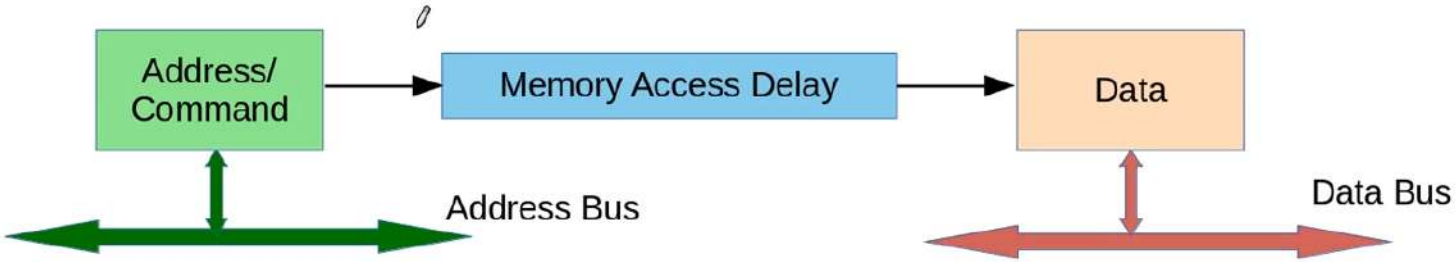
(4) Single-level cache + Split transaction Bus

(5) Multi-level cache + Split transaction Bus

Split transaction bus
+ single level cache

Split Transaction Bus

- Split bus transaction into **request** and **response** sub-transactions
  - Separate arbitration for each phase
- Other transactions may intervene
  - Improves bandwidth dramatically
  - Respone is matched to request
  - Buffering between bus and cache controllers
- Reduce serialisation down to the actual bus arbitration



Request-Response pairs

- BusRd = request ; data = response
- BusUpgr = request ; response = none
  - But it needs acknowledgement indicating that the transaction is committed and hence been serialised
  - To ensure that ACK does not appear on the bus as a separate transaction; the ACK is sent to the requesting processor by its own bus controller when the bus is granted to the BusUpgr request
- BusRdX = request ; response = data + ack of commitment (of inv by others)
  - The ack by others and data can be combined into one response transaction
- Write back = request ; response = none

Complications

(1) New request can appear on bus before previous one is serviced

- Even before snoop result is obtained
- Conflicting operations (minimum 1 write operation) to same block may be outstanding on the bus
- e.g. P1 and P2 both write to block 'A'
  - Both get bus before either snoop result, so both think they won --> may result in 2 processors writing to blk-A !

(2) Buffers are small, so may need flow-control

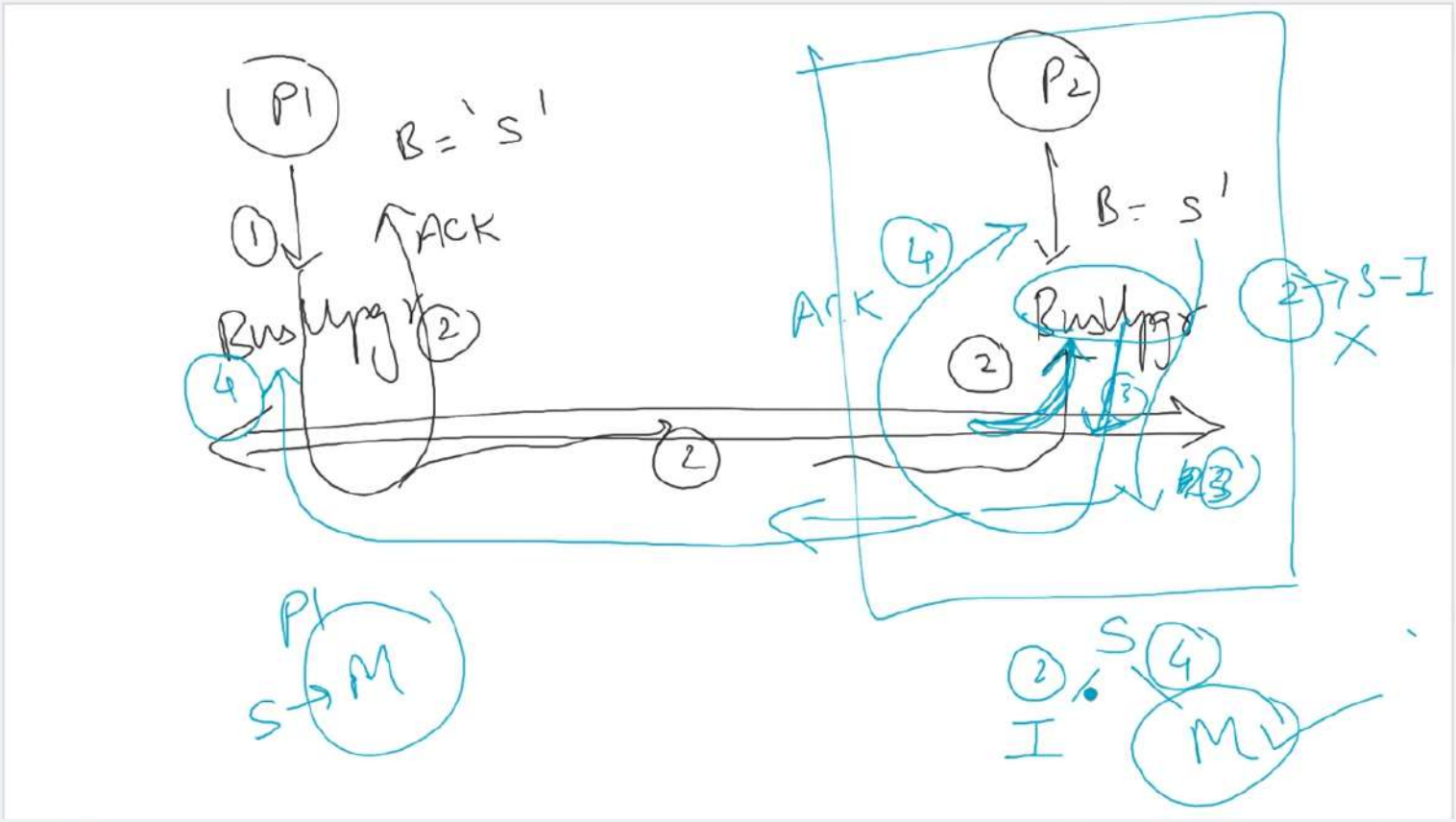
(3) Buffering implies re-visiting snoop issues

4_Sep_2021 - Google × 20_Aug - Mentimeter +

jamboard.google.com/d/1B_Zd7MYAoIegUAXuk5xSs3AviPYtXSrw80FmzLHO5P4/viewer

4_Sep_2021 1/1 Share

Set background Clear frame



82% (2:36) Mon, September 6, 15:27

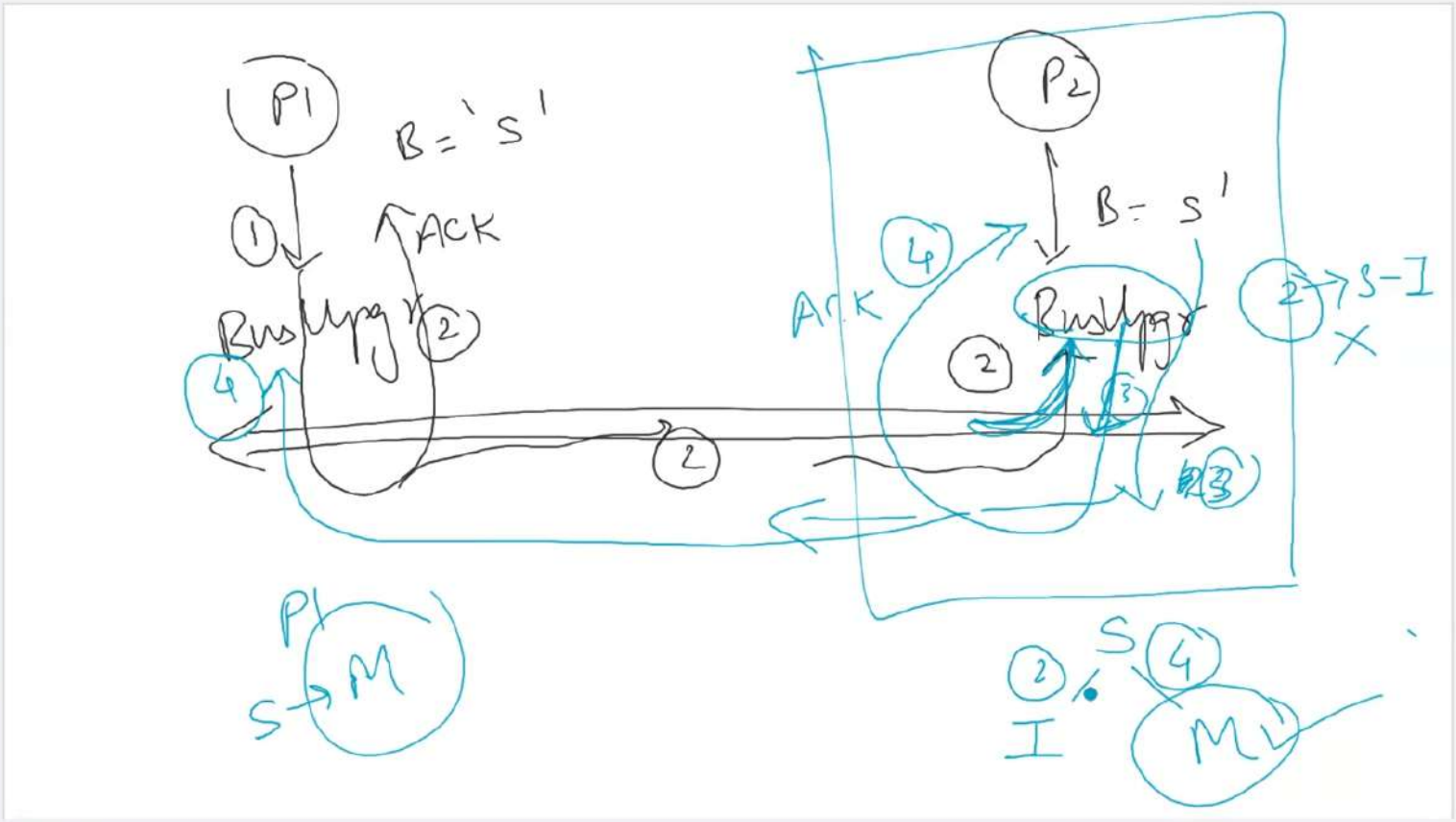
+28 D1 DG TANVISH SUVARTHI SARKAR AK ABHISHEK KUMAR SP SARASWATULA PHANI SAI PRANAV BHASKER GOEL IMLIJUNGLA LONGCHAR Hemangee Kalpesh Kapoor

4_Sep_2021 - Google × 20_Aug - Mentimeter +

jamboard.google.com/d/1B_Zd7MYAoIegUAXuk5xSs3AviPYtXSrw80FmzLHO5P4/viewer

4_Sep_2021 1/1 Share

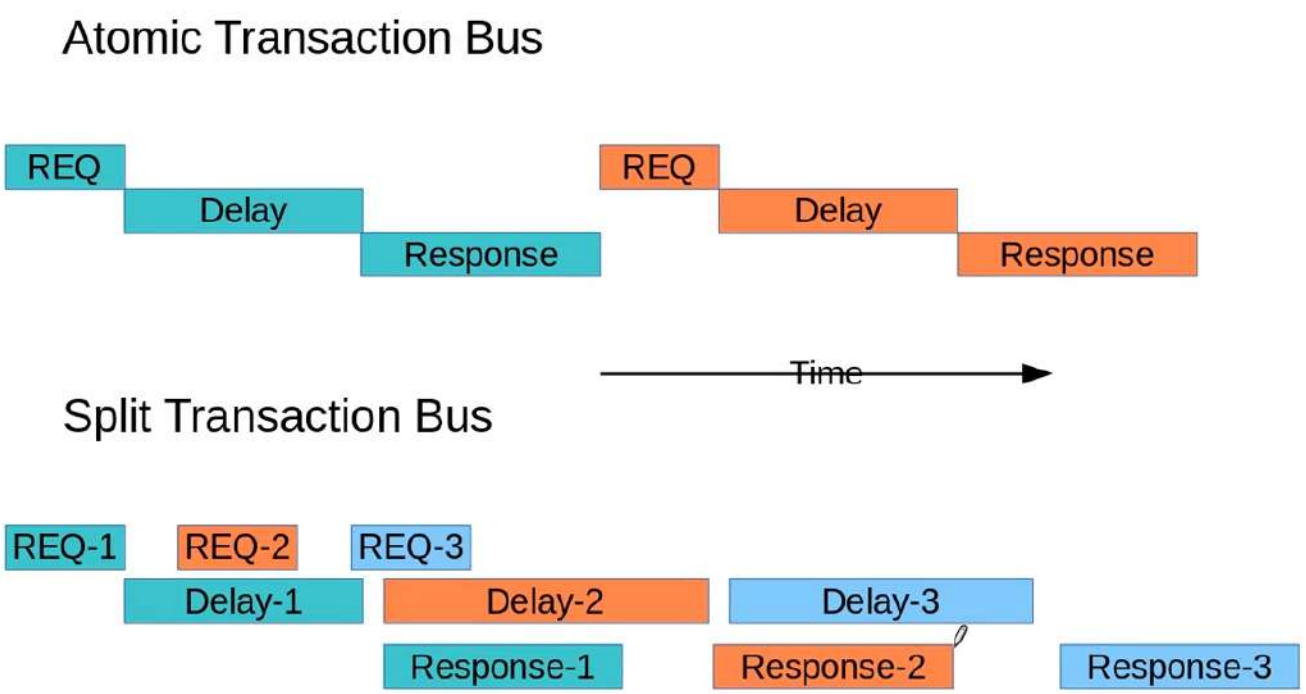
Set background Clear frame



81% (3:17) Mon, September 6, 15:28

+28 D1 AK TANVISH SUVARTHI SARKAR YK YOGESH KUMAR SP SARASWATULA PHANI SAI PRANAV BHASKER GOEL IMLIJUNGLA LONGCHAR Hemangee Kalpesh Kapoor

Atomic vs Split



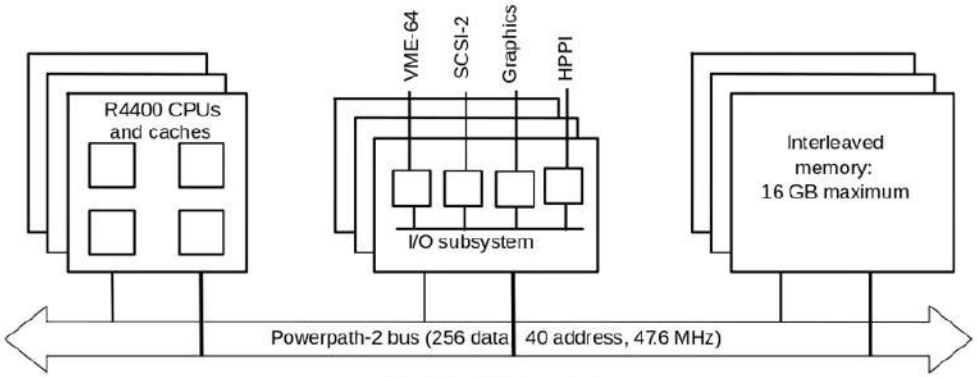
Atomic vs Split

- Design space too large
- When and how snoop results and data response provided
  - Are snoop results part of request phase or response phase?
- How many out-standing requests to allow
  - More requests => more buffering => complicated flow control
  - More requests => better bandwidth utilisation
- Are data response sent in the order of requests?
  - In-order: Intel Pentium Pro, DEC TurboLaser
  - Out-of-order: SGI Challenge, Sun Enterprise
- We will do a detailed split-bus example based on SGI Challenge Design

SGI Challenge overview

- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (up to 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with up to 8 outstanding reads
- all transactions take five cycles
- OS is variant of SVR4 UNIX called IRIX

(a) A four-processor board

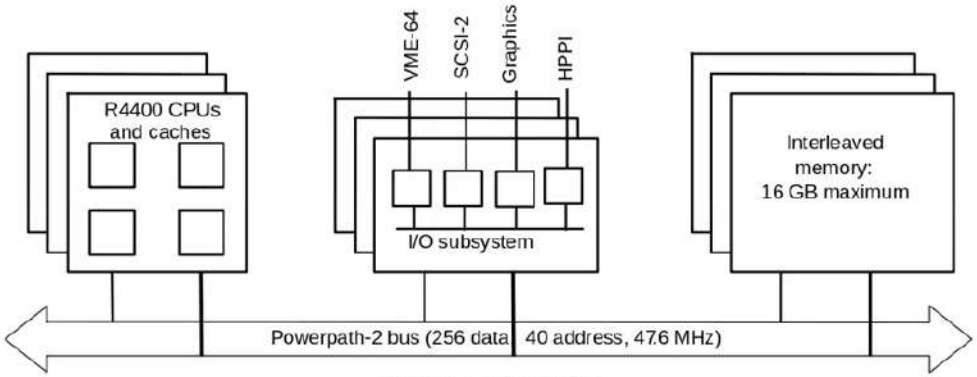


(b) Machine organization

SGI Challenge overview

- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (up to 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with up to 8 outstanding reads
- all transactions take five cycles
- OS is variant of SVR4 UNIX called IRIX

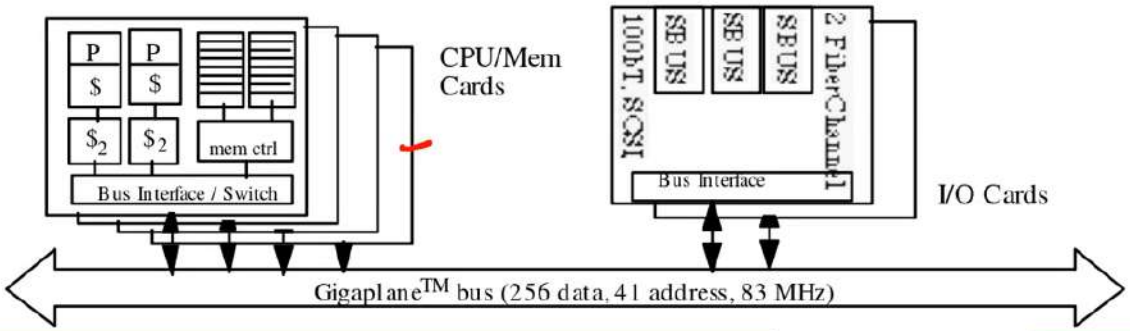
(a) A four-processor board



(b) Machine organization

SUN Enterprise overview

- Up to 30 UltraSPARC processors (peak 9 GFLOPs)
- GigaplaneTM bus has peak bw 2.67 GB/s; upto 30GB memory
- 16 bus slots, for processing or I/O boards
- 2 CPUs and 1GB memory per board
- memory distributed, unlike Challenge, but protocol treats as centralized
- Each I/O board has 2 64-bit 25Mhz SBUSes
- OS is Solaris UNIX



SGI Challenge design

- No conflicting requests for the same block allowed on bus
  - At most 8 outstanding requests total, makes conflict detection tractable
- Flow control through negative acknowledgement (NACK)
  - If buffer full, NACK request so that requestor retries
  - Need separate lines for: command (inc. NACK), addr+tag, Data
- Responses may be in different order than requests
  - Total order among transactions maintained by bus requests (phase)
  - Snoop results go as part of response phase
- Look at ===>

Bus Design | Snoop Results + Conflicting requests | Flow Control | Path of Request Through system

Bus design and Req-Response matching

- Two separate buses, arbitrated independently
- (1) Request bus: command and address
- (2) Response bus: data
- Req-response are out-of-order
- Therefore need a tag to match 8 outstanding requests => 3-bit tag on 3-bit wide lines
  - Tag lines used during response phase
  - Therefore address lines are free for other new requests
- Separate bus line for arbitration and snoop result

Data response phase

- Data needs 5 cycles to transfer in our design
- Cache block = 128 bytes = 1024 bits
- Data bus = 256-bit wide
- For 1024 bits = 256 * 4 cycles on data bus
- One additional cycle is for turn-around time
- Therefore 4+1 = 5 cycles for data response phase
- The signalling technology requires this turn-around time between controllers driving the lines

3-Phases

(1) Address request: 5 cycles:

- Arbitration (for bus), Resolution (one wins, tag assigned), Address (send addr), Decode (cache-tag look-up, snoop-hit-check, make transaction visibile to processor: state-change-E-S,M-S..etc), Ack (continue tag-lookup if not complete and extend ACK-cycle until snoop results are ready
- At decode decide who (which cache) will respond (if any)
- Actual response will come later
- End of this it is known who will provide data: cache or memory

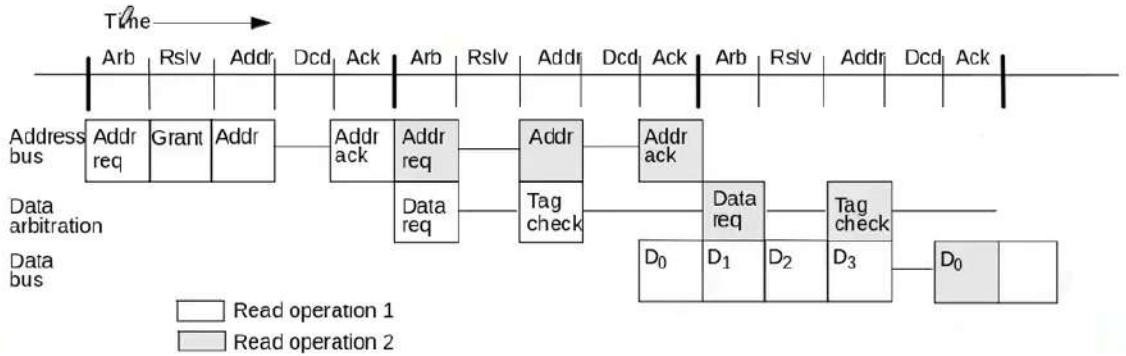
(2) Data request : 5 cycles

- Arbitration, Resolution, Address-cycle(=Tag-check), empty, [if target is ready then in ack cycle) Data-starts <D0,>

(3) Data response: 5 cycles (+ In parallel snoop results are conveyed so that states are correctly updated)

- Starts from ACK-stage of data -request phase
- Therefore D0 is sent in previous phase
- This phase sends <D1, D2, D3, empty,> <D0 of next request>
- The “empty” is for turn-around time

Ex: Read request that results in data transfer



- Scenario: cache has block in modified state
  - BusRd: send the block and state change E--> S
  - BusRdx or BusUpgr: send block and E --> I (inv.)

3-Phases

(1) Address request: 5 cycles:

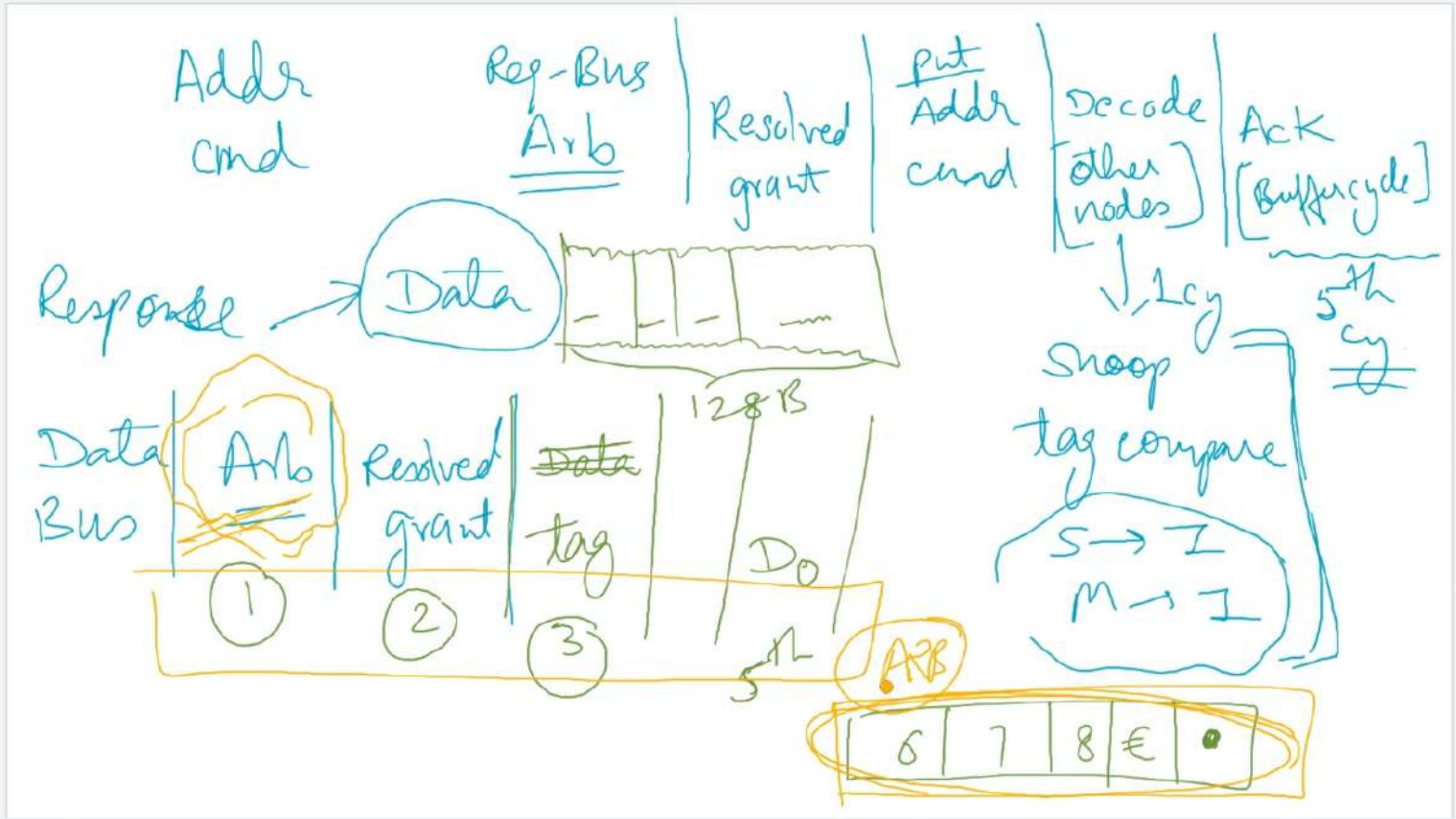
- Arbitration (for bus), Resolution (one wins, tag assigned), Address (send addr), Decode (cache-tag look-up, snoop-hit-check, make transaction visibile to processor: state-change-E-S,M-S..etc), Ack (continue tag-lookup if not complete and extend ACK-cycle until snoop results are ready
- At decode decide who (which cache) will respond (if any)
- Actual response will come later
- End of this it is known who will provide data: cache or memory

(2) Data request : 5 cycles

- Arbitration, Resolution, Address-cycle(=Tag-check), empty, [if target is ready then in ack cycle) Data-starts <D0,>

(3) Data response: 5 cycles (+ In parallel snoop results are conveyed so that states are correctly updated)

- Starts from ACK-stage of data -request phase
- Therefore D0 is sent in previous phase
- This phase sends <D1, D2, D3, empty,> <D0 of next request>
- The “empty” is for turn-around time


Addr cmd
Req-Bus Arb
Resolved grant
Put Addr cmd
Decode [other nodes]
Ack [Buffer cycle]
5th cy
1cy
Response
Data
128B
Snoop
tag compare
S → I
M → I
Data Bus
Arb
Resolved grant
Data tag
D0
1
2
3
5th
ARB
6
7
8
€