Implementation of Small-Scale Shared Memory Machines (Continued)

Fast Processor =>

- Multi-level Caches
  - **correct coherence semantics**
  - **deadlock avoidance**
- Multiple Outstanding Requests

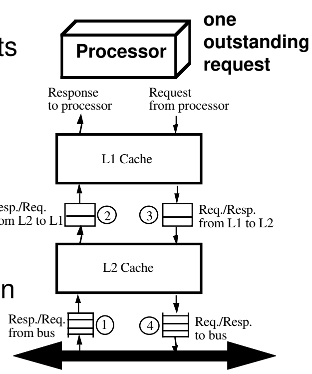
- Case Study
  - **SGI Challenge**

Internal Queue Structure of Two-Level Hierarchy

1. Entering bus requests and returning processor responses
2. Processor response or invalidation requests due to BusUpg or BusRdX

3/4. Processor requests and L1 responses

- Protocol requests percolate down till hit $ with block in proper state
  - **S or D for read**
  - **D for write or read-Exclusive**
- Shared responses are loaded into each $ in Shared state
- Read-Ex response are loaded into all but innermost level in dirty-but-stale, D in L1
- External Flush, CopyBack, Invalidate requests percolate up (how far?) changing state (to what?), response flow back down



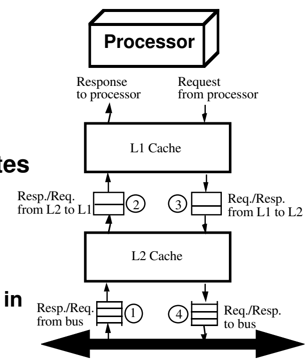
Correctness

- Ordering between transactions (request and response) as observed on the bus is maintained as transactions flow up and down the hierarchy
- Bad Example:
  - **suppose BusRdX slips in after BusRd transaction has completed and invalidates L1 before BusRd reaches L1?**
    - » **invalidate finds no block in L1 (maybe it was replaced while Inv was one its way up).**
    - » **Rd updates block, producing stale copy in S state**

  => use single queue for requests and responses

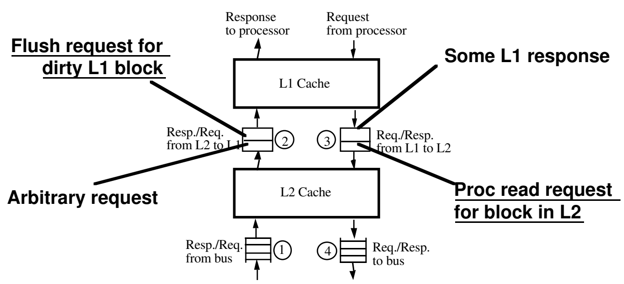
  => L2 processes transactions in bus order

  => processor must complete load before invalidation takes place



Deadlock

- Problem:

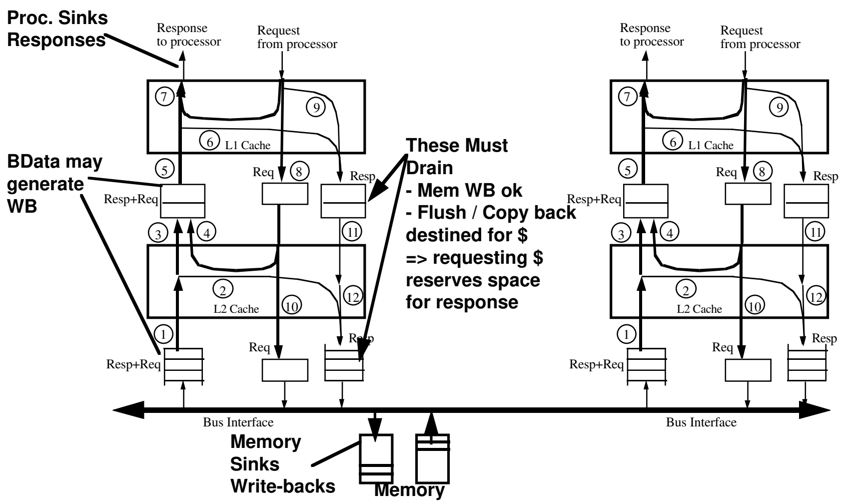


  - **L1 cannot process flush request because Q3 is full**
  - **L2 cannot process read request because Q2 is full**
- Solutions
  - **provide enough buffering for max outstanding requests and responses (1 + 8)**
    - » **responses are big**
  - **classify transactions to avoid the cycle**
    - » **separate queue for request and response**
    - » **response guaranteed to make progress**

Request / Response Classification

- Request: generates no request, zero or more responses
- Response: generates no request, no response
- Proc down to Bus:
  - **PRdMiss** **Req**
  - **PWrMiss** **Req**
  - **PwrBk** **Resp**
- Bus up toward proc:
  - **BData (data replies to Pr R/W miss) ?***
    - » **if space reserved in L1 => resp**
    - » **if not, may replace dirty block => req**
      - **don't want invalidate, flush, or copy back to bypass**
  - **Invalidations (due to BusUpG) ? (req)**
  - **Flush/Copyback on Dirty blocks Req**
    - » **(due to BusRd,BusRdX)**

Forward Progress



Update Protocols

- Update write flows downward checking $ state till hits bus
- Memory updates
- Caches grab and propagate update upward as long as snoop is positive
- Rely on bus arbitration for serialization

Multiple Outstanding Misses (Lock-up Free Caches)

- Issues
  - **single read / multiple writes or multiple reads and writes ?**
  - **mechanism to keep track of outstanding requests**
    - » **req. address, type, resp. address, status**
  - **mechanism and policy for conflicts**
    - » **multiple outstanding writes to same mem block**
      - **same word vs different word within block**
      - **stall on w-w? r-w? r-r ? conflicts**
  - **multiple requests to same $ line, but different mem block**
- Approach
  - **add transient state to each cache block**
    - » **invalid pending (IP), shared pending (SP), exclusive pending (EP)**
    - » **cache tag is valid, awaiting data from memory system**
  - **add sub-block write bit (SWB) per word**
    - » **IP all SWBs are off**
    - » **SP on => proc writes since request, off => valid**
    - » **EP on => proc writes since request, off => invalid**
  - **pending read registers**

Operation (direct map)

- Processor Write: if we don't find the block already D, go into pending dirty state, initiate proper bus transaction, and use the SWB to indicate words written for subsequent merge
  - **if hit?**
    - » **if D or E, update and set state to D**
    - » **if S, (1) upgrade to SP and clear SWBs; (2) issue BusUpG; (3) write word and set SWB**
    - » **if EP or SP, (3) write word and set SWB**
    - » **if IP, (1')upgrade to EP; (2') issue BusRdX, (3) write word and set SWB**
  - **no tag match?**
    - » **if D, issue WB; upgrade to IP and clear SWB; issue BusRdX;**
    - » **(3) write word and set SWB**
- Processor read:
  - **hit, VE or S or SP or D => return data**
  - **hit, EP => return data if SWB otherwise register pending read**
  - **miss, write-back if dirty; go to IP; issue BusRd**
    - » **stall on pending**

Operation (cont)

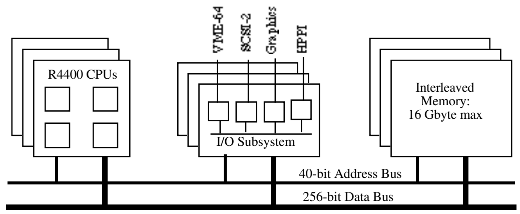
- Response
  - **update cache block**
    - » **SWB words take precedence**
  - **upgrade pending -> D**
  - **service pending reads**
- Race conditions
  - **ex: observe BusRdX or BusUpG for block in SP/EP state**
    - » **data written by local processor should not be *made visible* until exclusive ownership has been obtained**
    - » **bus <u>is</u> the serialization**

  busUpG or BusRdX while in IP/EP => no action

  busUpG or BusRdX while in SP => go to EP

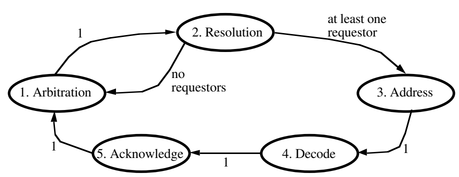
  busRd while in IP/SP/EP => no action

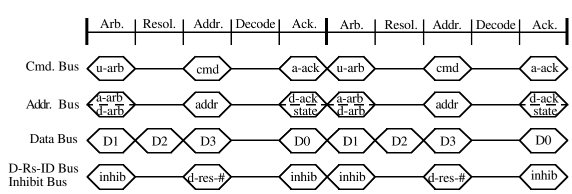
SGI Challenge



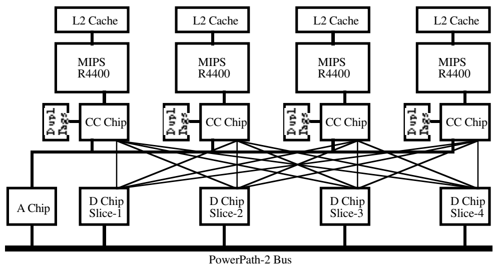
- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (upto 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with upto 8 outstanding reads
  - **all transactions take five cycles**
  - **parallel distributed arbitration**

Bus Timing

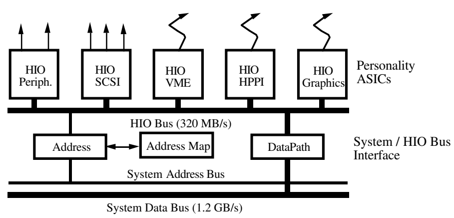




Board Design



I/O Subsystem



- HIO
  - **64-bit multiplexed address / data**
  - **split-read, upto 4 per device**
  - **pipelined, centralized arbitration**
  - **several transaction lengths**
  - **address remapping**
- I/O board acts like a processor to memory system