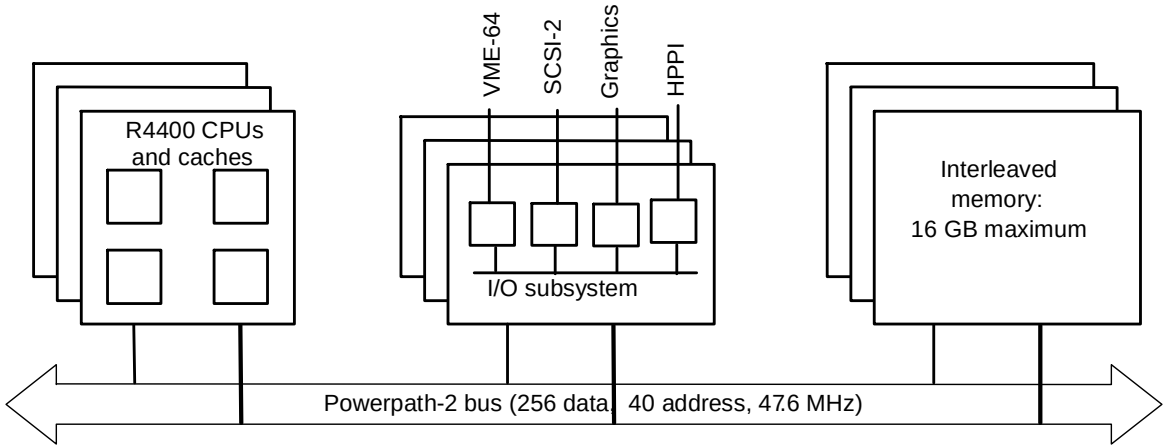
Old Example: SGI Challenge Overview

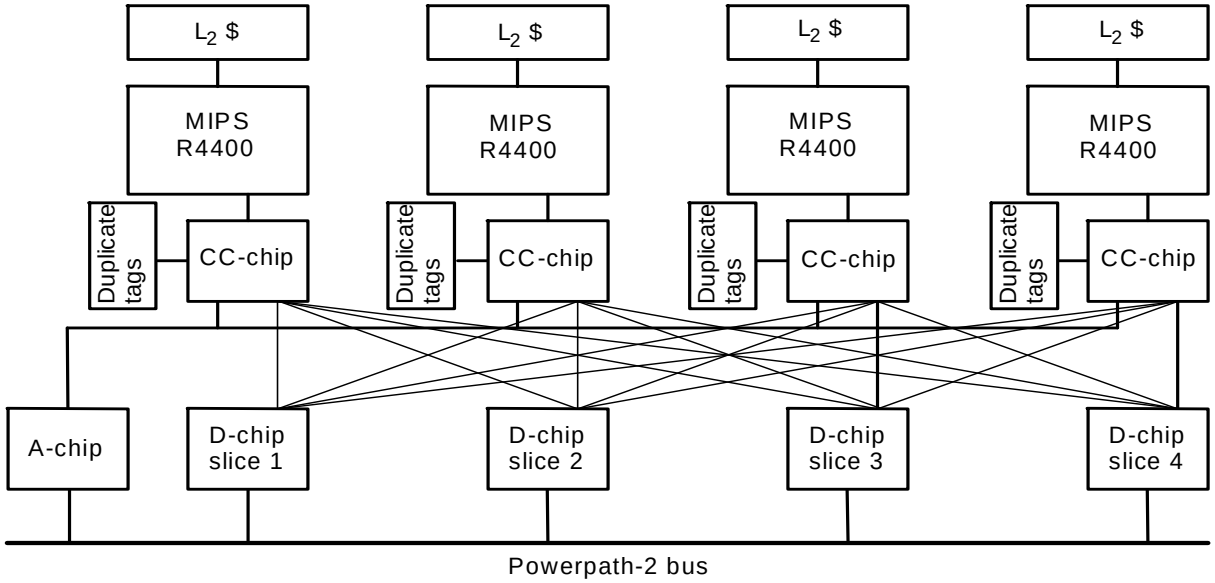
(a) A four-processor board



(b) Machine organization

- **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)**
- **1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals**
- **128-Byte lines (1 + 4 cycles)**
- **Split-transaction with up to 8 outstanding reads**
  - **All transactions take five cycles**
- **Miss latency nearly 1 us (mostly on CPU board, not bus…)**

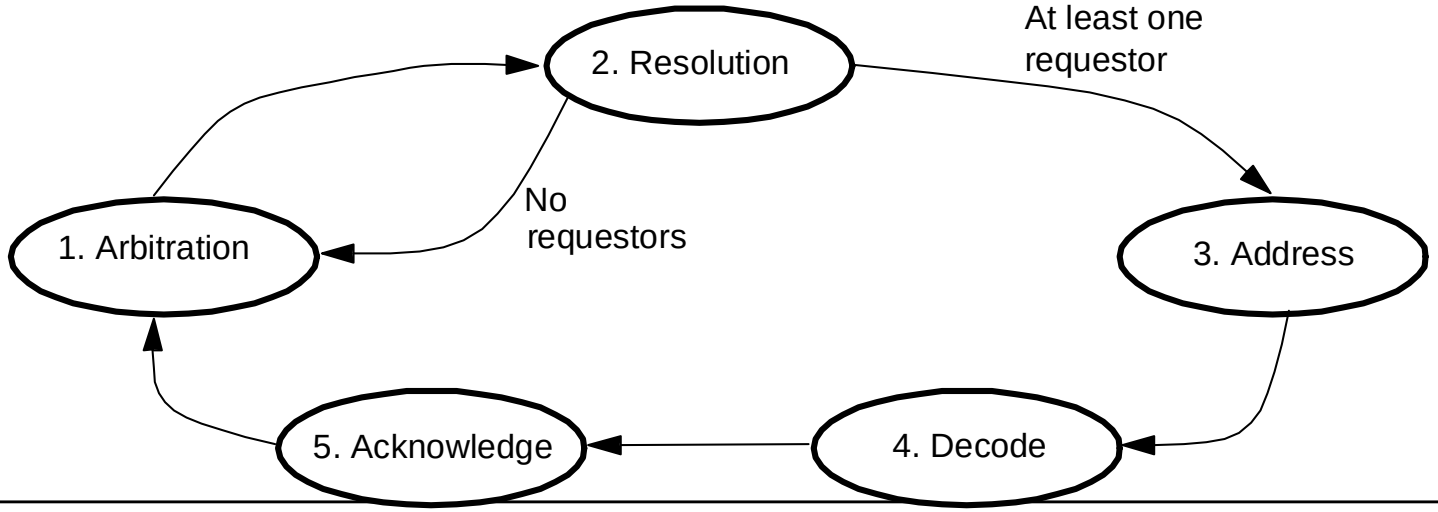
Processor and Memory Systems



- **4 MIPS R4400 processors per board share A / D chips**
- **A chip has address bus interface, request table, control logic**
- **CC chip per processor has duplicate set of tags**
- **Processor requests go from CC chip to A chip to bus**
- **4 bit-sliced D chips interface CC chip to bus**

SGI Powerpath-2 Bus

- **Non-multiplexed (i.e., separate A and D), 256-data/40-address, 47.6 MHz, 8 outstanding requests**
- **Wide → more interface chips so higher latency, but more bandwidth at slower clock**
- **Large block size also calls for wide bus**
- **Uses Illinois MESI protocol (cache-to-cache sharing)**

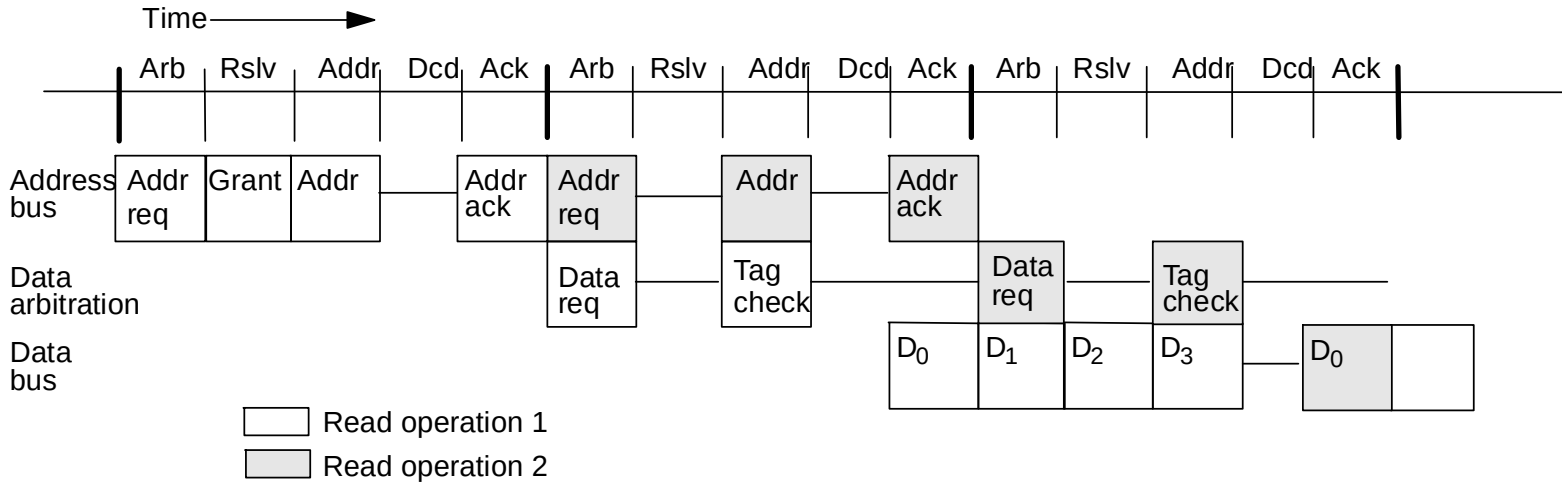


Bus Design and Request-Response Matching

- **Essentially two separate buses, arbitrated independently**
  - **"Request" bus for command and address**
  - **"Response" bus for data**
- **Out-of-order responses imply need for matching request with corresponding response**
  - **Request gets 3-bit tag when wins arbitration (8 outstanding max)**
  - **Response includes data as well as corresponding request tag**
  - **Tags allow response to not use address bus, leaving it free**
- **Separate bus lines for arbitration and for snoop results**

Bus Design (continued)

- **Each of request and response phase is 5 bus cycles**
  - **Response: 4 cycles for data (128 bytes, 256-bit bus), 1 turnaround**
  - **Request phase: arbitration, resolution, address, decode, ack**
  - **Request-response transaction takes 3 or more of these**



Cache tags looked up in decode; extend ack cycle if not possible

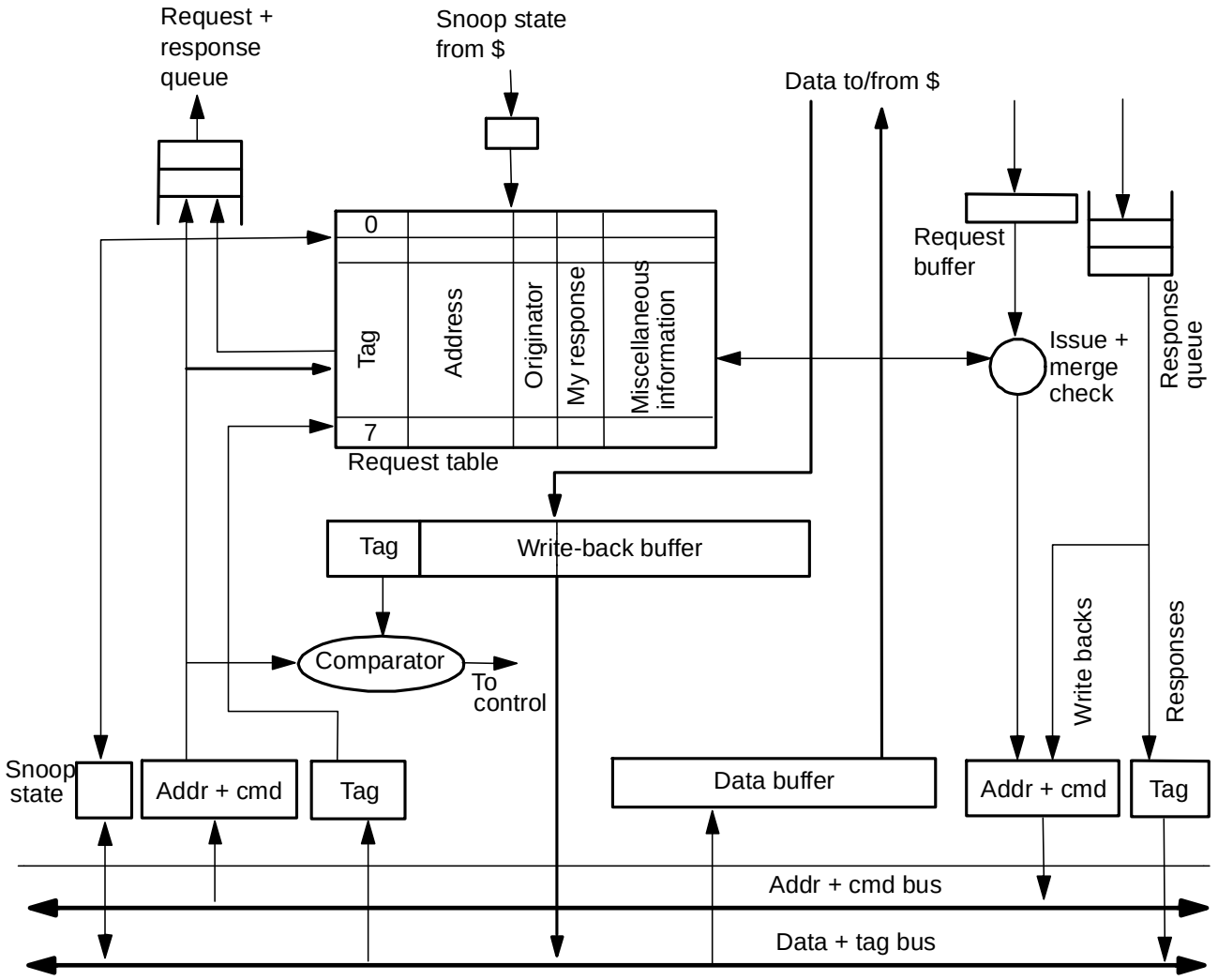
- Determine who will respond, if any
- Actual response comes later, with re-arbitration

Write-backs have request phase only: arbitrate both data+addr buses

Bus Design (continued)

- **Flow-control through *negative acknowledgement (NACK)***
- **No conflicting requests for same block allowed on bus**
  - **8 outstanding requests total, makes conflict detection tractable**
  - **Eight-entry "request table" in each cache controller**
  - **New request on bus added to all at same index, determined by tag**
  - **Entry holds address, request type, state in that cache (if determined already), ...**
  - **All entries checked on bus or processor accesses for match, so fully associative**
  - **Entry freed when response appears, so tag can be reassigned by bus**

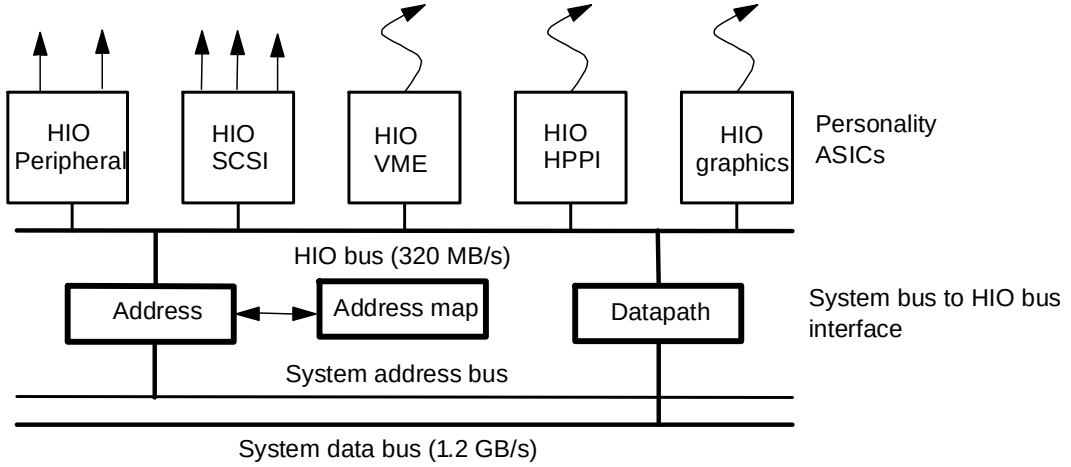
Bus Interface with Request Table



Memory Access Latency

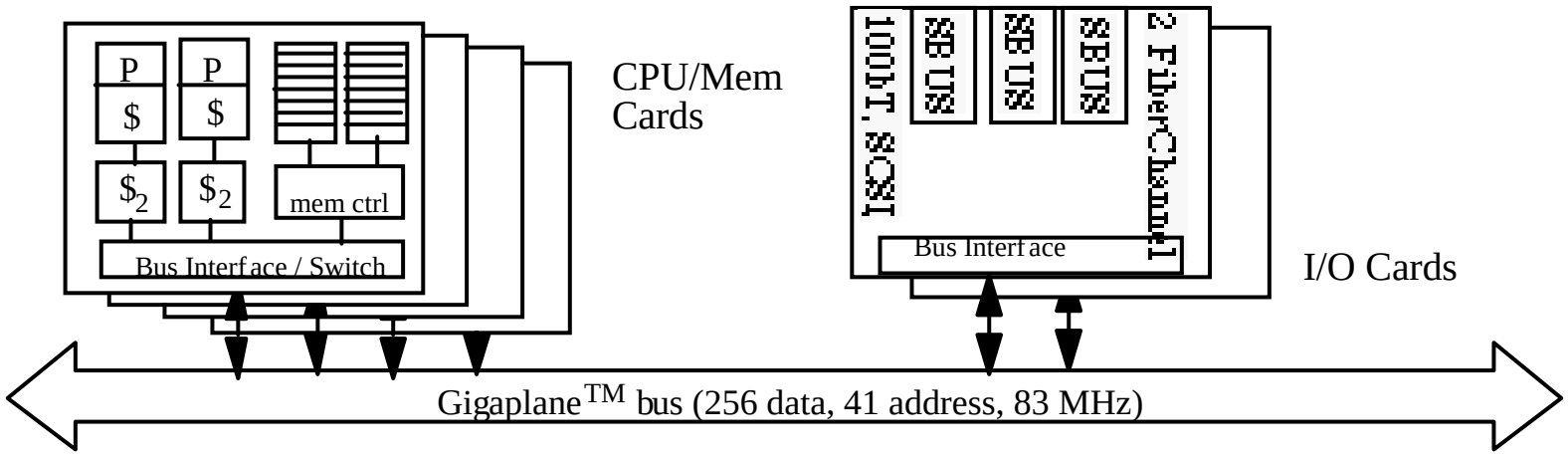
- **250ns access time from address on bus to data on bus**
- **But overall latency seen by processor is 1000ns!**
  - **300 ns for request to get from processor to bus**
    - » **Down through cache hierarchy, CC chip and A chip**
  - **400ns later, data gets to D chips**
    - » **3 bus cycles to address phase of request transaction, 12 to access main memory, 5 to deliver data across bus to D chips**
  - **300ns more for data to get to processor chip**
    - » **Up through D chips, CC chip, and 64-bit wide interface to processor chip, load data into primary cache, restart pipeline**

Challenge I/O Subsystem



- **Multiple I/O cards on sys bus, each w/320MB/s HIO bus**
  - **Personality ASICs connect these to devices (standard and graphics)**
- **Proprietary HIO bus**
  - **64-bit multiplexed address/data, split read trans., up to 4 per device**
  - **Pipelined, but centralized arbitration, with several transaction lengths**
  - **Address translation via mapping RAM in system bus interface**
- **I/O board acts like a processor to memory system**

SUN Enterprise 6000 Overview



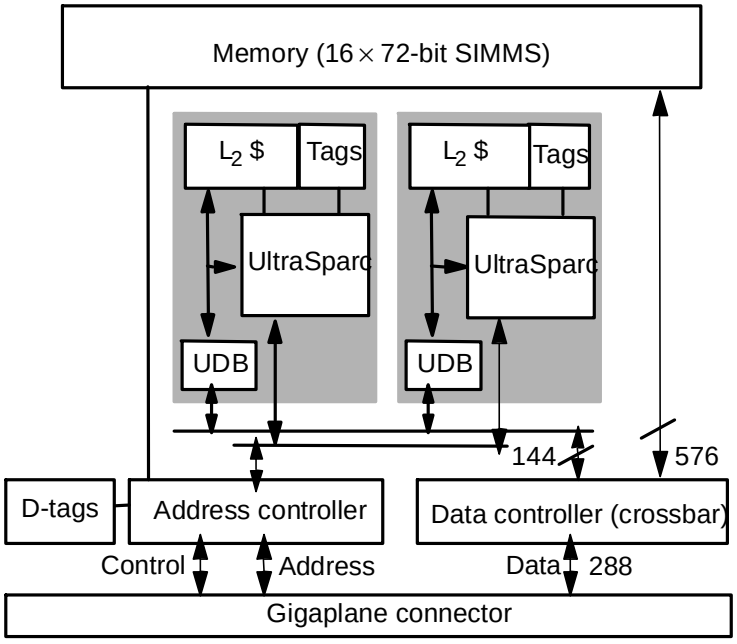
- **Up to 30 UltraSPARC processors, MOESI protocol**
- **Gigaplane™ bus has peak bw 2.67 GB/s, 300 ns latency**
- **Up to 112 outstanding transactions (max 7 per board)**
- **16 bus slots, for processing or I/O boards**
  - **2 CPUs and 1GB memory per board**
    - » **Memory distributed, but protocol treats as centralized (UMA)**

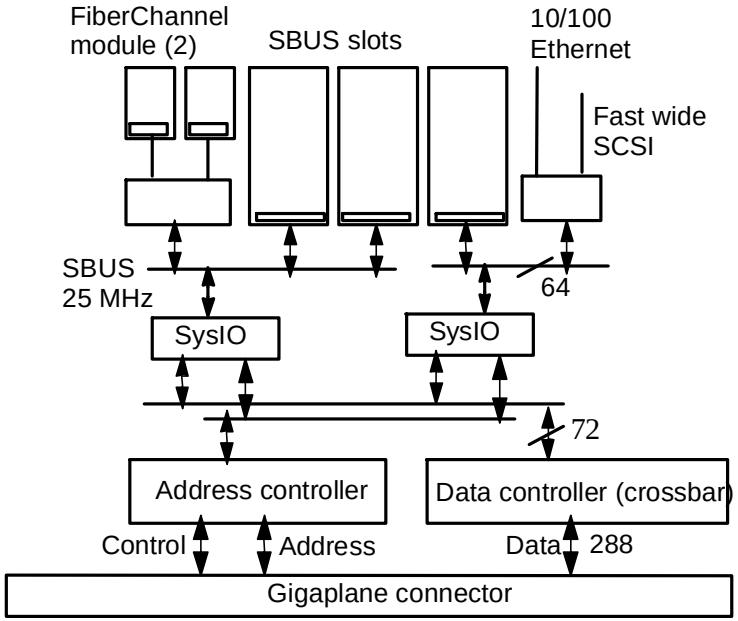
Sun Gigaplane Bus

- Non-multiplexed, split-transaction, 256-data/41-address, 83.5 MHz (plus 32 ECC lines, 7 tag, 18 arbitration, etc. Total 388)
- Cards plug in on both sides: 8 per side
- 112 outstanding transactions, up to 7 from each board
  - Designed for multiple outstanding transactions per processor
- Emphasis on reducing latency, unlike Challenge
  - Speculative arbitration if address bus not scheduled from prev. cycle
  - Else regular 1-cycle arbitration, and 7-bit tag assigned in next cycle
- Snoop result associated with request (5 cycles later)
- Main memory can stake claim to data bus 3 cycles into this, and start memory access speculatively
  - Two cycles later, asserts tag bus to inform others of coming transfer
- MOESI protocol

Enterprise Processor and Memory System

- **2 procs / board, ext. L2 caches, 2 mem banks w/ x-bar**
- **Data lines buffered through UDB to drive internal 1.3 GB/s UPA bus**
- **Wide path to memory so full 64-byte line in 2 bus cycles**





Enterprise I/O System

- I/O board has same bus interface ASICs as processor boards
- But internal bus half as wide, and no memory path
- Only cache block sized transactions, like processing boards
  - Uniformity simplifies design
  - ASICs implement single-block cache, follows coherence protocol
- Two independent 64-bit, 25 MHz Sbuses
  - One for two dedicated FiberChannel modules connected to disk
  - One for Ethernet and fast wide SCSI
  - Can also support three SBUS interface cards for arbitrary peripherals
- Performance and cost of I/O scale with # of I/O boards

Sun Enterprise 10000 (aka E10K or Starfire)

- How far can you go with snooping coherence?
- Quadruple request/snoop bandwidth using four "logical" address buses
  - Each handles 1/4 of physical address space
  - Impose *logical* ordering for consistency: for writes on same cycle, those on bus 0 occur "before" bus 1, etc.
- Get rid of data bandwidth problem: use a network
  - E10k uses 16x16 crossbar between CPU boards & memory boards
  - Each CPU board has up to 4 CPUs: max 64 CPUs total
- 10.7 GB/s max BW, 468 ns unloaded miss latency
- We will discuss a paper on E10K later

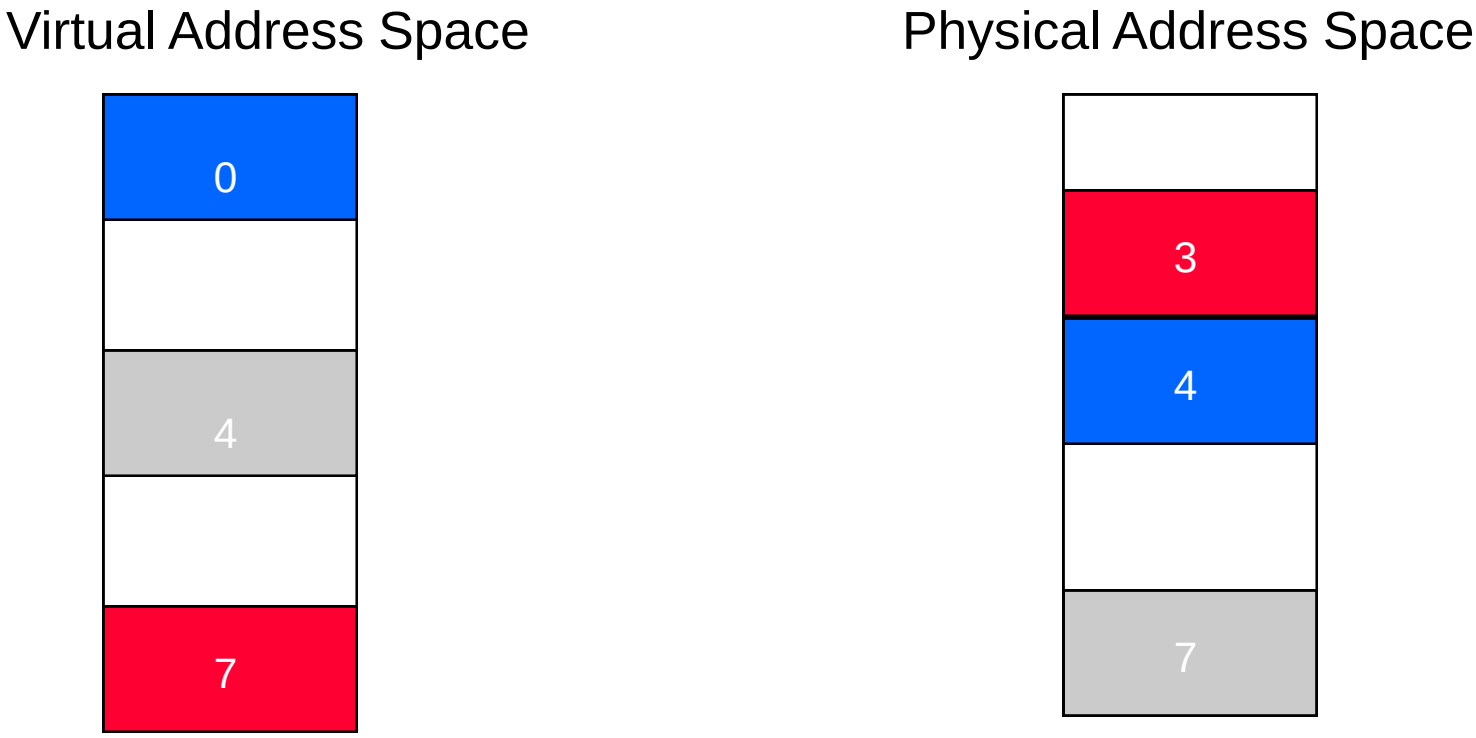
Outline for Implementing Snooping

- **Coherence Control Implementation**
- **Writebacks, Non-Atomicity, & Serialization/Order**
- **Hierarchical Cache**
- **Split Buses**
- **Deadlock, Livelock, & Starvation**
- **Three Case Studies**
- **TLB Coherence**
- **Virtual Cache Issues**

These two issues apply to any coherence protocol, not just snooping

Translation Lookaside Buffer

- **Cache of page table entries**
- **Page table maps virtual page to physical frame**



The TLB Coherence Problem

- **Since TLB is a cache, must be kept coherent**
- **Change of PTE on one processor must be seen by all processors**
- **Why might a PTE be cached in more than 1 TLB?**
  - **Actual sharing of data**
  - **Process migration**
- **Historical view:**
  - **Changes are infrequent → get OS to do it**
- **Current view (as of January 2010):**
  - **Coherence via OS is way too slow → use HW for TLB coherence**
  - **Optional paper by Romanescu, Lebeck, Sorin [HPCA 2010]**

TLB Shootdown

- To modify TLB entry, modifying processor must
  - LOCK page table,
  - Flush TLB entries,
  - Queue TLB operations,
  - Send inter-processor interrupt,
  - Spin until other processors are done
  - UNLOCK page table
- SLOW!
- But most common solution today
  - Until Romanescu's HPCA 2010 paper convinces everyone of the folly of their ways

TLB Shootdown Improvements

- **Evolutionary Changes**
  - **Keep track of which processor even had the mapping & only shoot them down**
  - **Defer shootdowns on "upgrade" changes (e.g., page from read-only to read-write)**
  - **SGI Origin "poison" bit for also deferring downgrades**
- **Revolutionary Changes**
  - **"Invalidate TLB entry" instruction (e.g., PowerPC)**
  - **No TLB (e.g., Berkeley SPUR)**
    - » **Use virtual L1 caches so address translation only on miss**
    - » **On miss, walk PTE (which will often be cached normally)**
    - » **PTE changes kept coherent by normal cache coherence**

Virtual Caches & Synonyms

- **Problem**
  - **Synonyms: V0 & V1 map to P1**
  - **When doing coherence on block in P1, how do you find V0 & V1?**
- **Don't do virtual caches (most common today)**
- **Don't allow synonyms**
  - **Probably use a segmented global address space**
  - **E.g., Berkeley SPUR had process pick 4 of 256 1GB segments**
  - **Still requires reverse address translation**
- **Allow virtual cache & synonyms**
  - **How do we implement reverse address translation?**
  - **See Wang et al. next**

Wang et al. [ISCA89]

- **Basic Idea**
  - Extended Goodman one-level cache idea [ASPLOS87]
  - Virtual L1 and physical L2
  - Do coherence on physical addresses
  - Each L2 block maintains pointer to corresponding L1 block (if any) (requires log2 #L1_blocks - log2 (page_size / block_size)
  - Never allow block to be simultaneously cached under synonyms
- **Example where V0 & V1 map to P2**
  - Initially V1 in L1 and P2 in L2 points to V1
  - Processor references V0
  - L1 miss
  - L2 detects synonym in L1
  - Change L1 tag and L2 pointer so that L1 has V0 instead of V1
  - Resume

Virtual Caches & Homonyms

- **Homonym**
  - **“Pool” of water and “pool” the game**
  - **V0 of one process maps to P2, while V0 of other process maps to P3**
- **Flush cache on context switch**
  - **Simple but performs poorly**
- **Address-space IDs (ASIDs)**
  - **In architecture & part of context state**
- **Mapping-valid bit of Wang et al.**
  - **Add mapping-valid as a “second” valid bit on L1 cache block**
  - **On context switch do “flash clear” of mapping-valid bits**
  - **Interesting case is valid block with mapping invalid**
    - » **On processor access, re-validate mapping**
    - » **On replacement (i.e., writeback) treat as valid block**

Outline for Implementing Snooping

- **Coherence Control Implementation**
- **Writebacks, Non-Atomicity, & Serialization/Order**
- **Hierarchical Cache**
- **Split Buses**
- **Deadlock, Livelock, & Starvation**
- **Three Case Studies**
- **TLB Coherence**
- **Virtual Cache Issues**

Outline

- **Motivation for Cache-Coherent Shared Memory**
- **Snooping Cache Coherence**
- **Implementing Snooping Systems**
- **Advanced Snooping Systems**
  - **Sun UltraEnterprise 10000**
  - **Multicast Snooping (Wisconsin)**

Sun UltraEnterprise 10000 (Starfire)

- **Shared-wire bus is bottleneck in snooping systems**
  - **Tough to implement at high speed**
  - **Centralized shared resource**
- **Solution: multiple “logical buses”**
- **PRESENTATION**

Multicast Snooping

- **Bus is bottleneck in snooping systems**
  - **But why broadcast requests when we can multicast?**
- **PRESENTATION**

Timestamp Snooping

- Optional paper by Martin et al. [ASPLOS 2000]
- Insight: snooping doesn't actually require:
  - All coherence requests to arrive at every node at the exact same time (ok, so we already knew that)
  - All coherence requests to arrive at every node in the same order … in physical time
- Key idea: assign logical times to requests and let nodes process them in logical time order
- OK, so why is this interesting?
  - Can implement snooping on any network topology! Not just buses or trees

Outline

- **Motivation for Cache-Coherent Shared Memory**
- **Snooping Cache Coherence**
- **Implementing Snooping Systems**
- **Advanced Snooping Systems**