Lecture 11: Consistency Models

- Topics: sequential consistency, requirements to implement sequential consistency

Sequential Consistency

- A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program

- Atomicity: each processor sees operations complete instantaneously in the same order

- Program order is preserved within each processor

Example Programs

Initially, Flag1 = Flag2 = 0

P1	P2
Flag1 = 1	Flag2 = 1
if (Flag2 == 0)	if (Flag1 == 0)
critical section	critical section

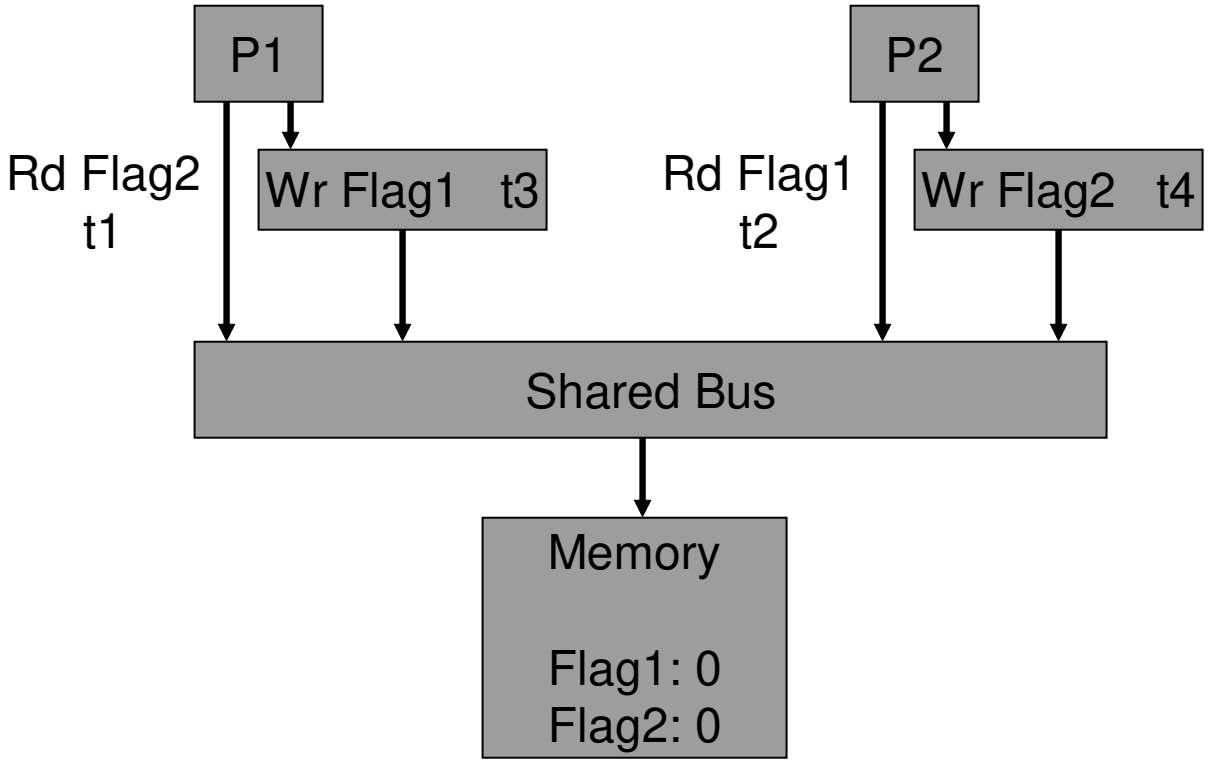
Initially, A = B = 0

P1	P2	P3
A = 1		
	if (A == 1)	
	B = 1	
		if (B == 1)
		register = A

Write Buffers with Bypassing

- Assume an architecture without caches
- Writes by a processor are inserted in the write buffer and the processor proceeds without waiting for the write to complete – subsequent reads have priority for mem-access
- This can result in both processors entering the critical section in example-1
- Illustrates the importance of program order (wr → rd dependence)

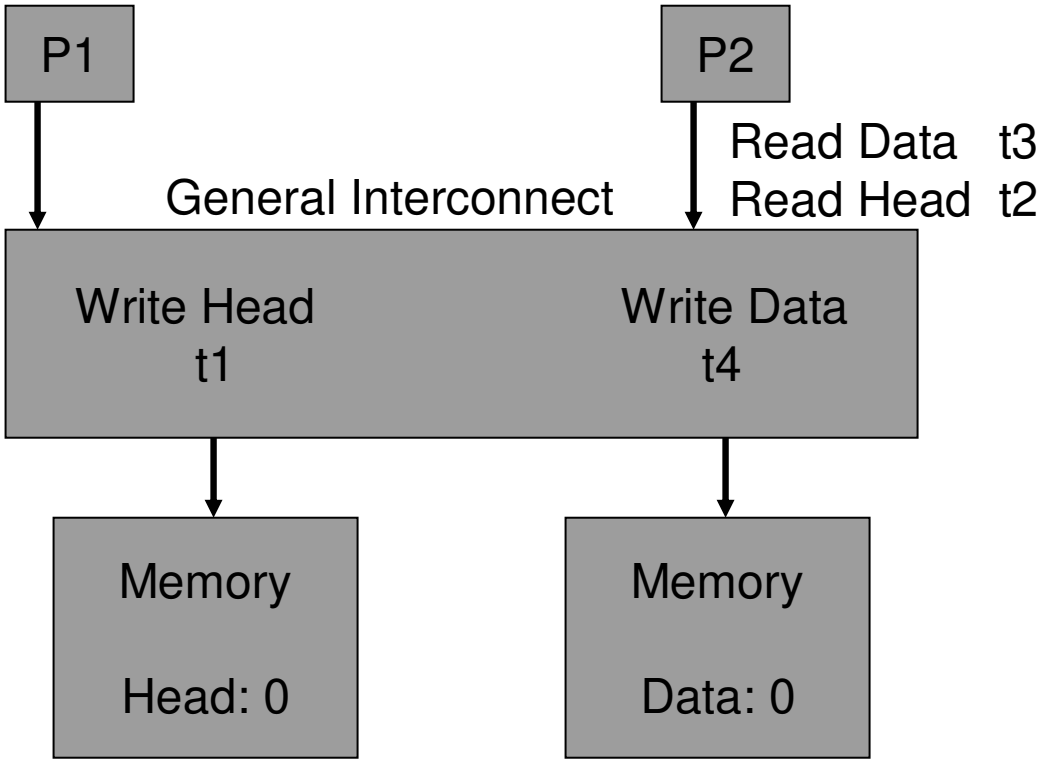
Write Buffer



Overlapping Writes

- Architecture without caches, multiple memory modules, general interconnect (non-bus), writes are issued and the processor continues without waiting for them to finish
- Again, sequential consistency is violated in next example
- Illustrates the importance of program order (wr → wr dependence)
- To enforce ordering, processors must wait for write acknowledgments before proceeding

Overlapped Writes

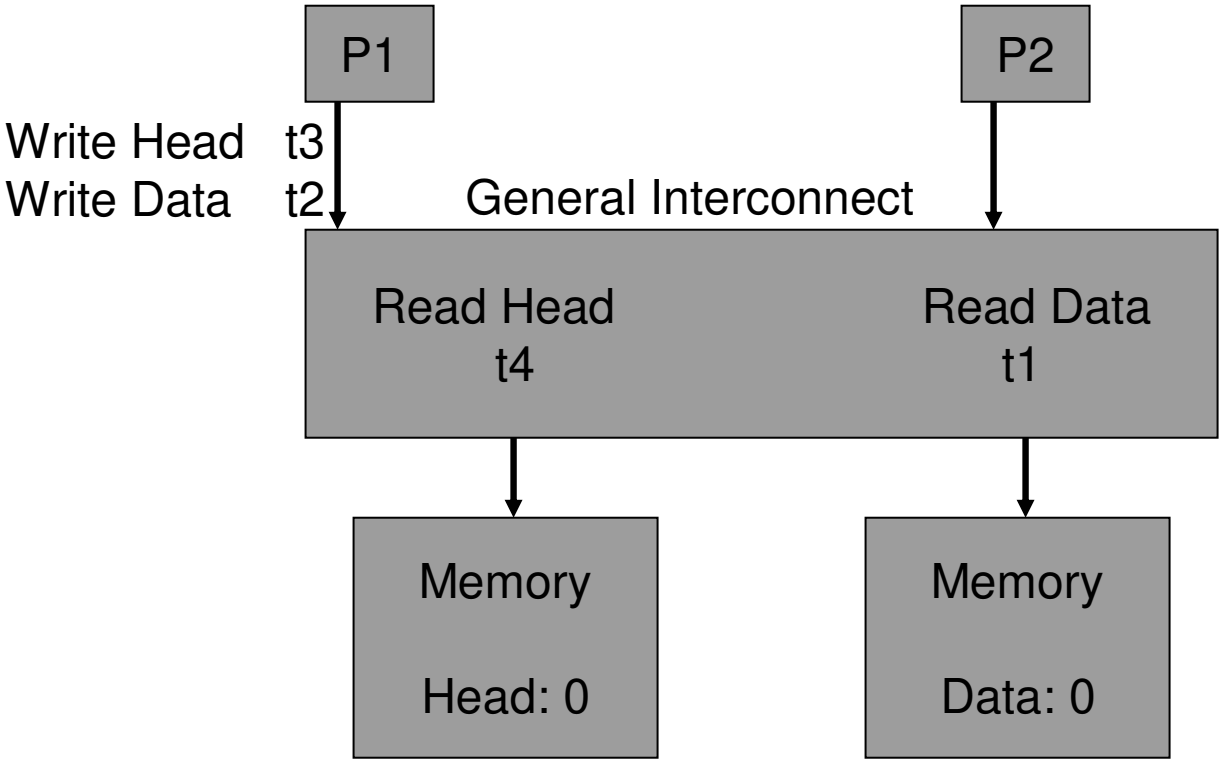


P1	P2
Data = 2000	while (Head == 0) { }
Head = 1	… = Data

Non-Blocking Reads

- Assume writes complete atomically and in program order
- If reads issue (or complete) out of order, sequential consistency is violated
- Illustrates the importance of rd→rd program order

Non-Blocking Reads



P1	P2
Data = 2000	while (Head == 0) { }
Head = 1	… = Data

Architectures with Caches

- The earlier examples only violated program order – writes were still atomic and seen by all processors in the same order
- The latter condition can be easily violated if each processor has a cache (in spite of cache coherence)
- Recall that cache coherence simply guarantees write propagation and write serialization to the same memory location – it does not guarantee that writes to different locations are seen in the same order

Maintaining Atomicity

- To preserve program order, we will not allow a processor to proceed unless it receives the write acknowledgment (all other processors have seen invalidates or updates)

- Two conditions can ensure the appearance of write atomicity:
  - write serialization to each location
  - stalling reads until all processors have seen the last update to that location

Write Serialization Example

P1	P2	P3	P4
A = 1	A = 2	while (B != 1) { }	while (B != 1) { }
B = 1	C = 1	while (C != 1) { }	while (C != 1) { }
		register1 = A	register2 = A

- register1 and register2 having different values is a violation of sequential consistency – possible if updates to A appear in different orders
- Cache coherence guarantees write serialization to a single memory location

Non-Atomic Write Updates

Initially, A = B = 0

P1	P2	P3
A = 1		
	if (A == 1)	
	B = 1	
		if (B == 1)
		register = A

- P2 reads new A before update reaches P3
- Update of B reaches P3 before update of A
- P3 reads B and then A before update of A arrives

- Assume each processor executes operations in program order (waiting for acks) and we have write serialization to the same memory location

Implementing Atomic Updates

- The above problem can be eliminated by not allowing a read to proceed unless all processors have seen the last update to that location

- Easy in an invalidate-based system: memory will not service the request unless it has received acks from all processors

- In an update-based system: a second set of messages is sent to all processors informing them that all acks have been received; reads cannot be serviced until the processor gets the second message

Summary

- To preserve sequential consistency:
  - hardware must preserve program order for all memory operations (including waiting for acks)
  - writes to a location must be serialized
  - the value of a write cannot be read unless all have seen the write (it is ok if writes to different locations are not seen in the same order as long as conflicting reads do not happen)

Performance Optimizations

- Program order is a major constraint – the following try to get around this constraint without violating seq. consistency
  - ➢ if a write has been stalled, prefetch the block in exclusive state to reduce traffic when the write happens
  - ➢ allow out-of-order reads with the facility to rollback if the ROB detects a violation

- Get rid of sequential consistency in the common case and employ relaxed consistency models – if one really needs sequential consistency in key areas, insert fence instructions between memory operations

- Next class: consistency models by relaxing constraints

Title

- Bullet