

Memory Consistency

- Cache coherence ensures multiple processors see a consistent view of memory, but how consistent must this view be?
- The key question: **When** must a processor see values updated by another processor?
- This determines what properties must be enforced among reads and writes to different locations by different processors.

Challenges

```
P1:      A = 0;  
        .....  
        A = 1;  
L1:      if (B==0) ...
```

```
P2:      B = 0;  
        .....  
        B = 1;  
L2:      if (A==0) ...
```

- Although the question of how consistent memory must be seems simple, it is remarkably complicated
- If writes always take immediate effect and are immediately seen by other processors, it will be impossible for both if statements (labeled L1 and L2) to evaluate their conditions as true.
- But suppose the write invalidate is delayed, and the processor is allowed to continue during this delay. Then, it is possible that both P1 and P2 have not seen the invalidations for B and A (respectively) before they attempt to read the values
- The question now is should this behavior be allowed, and, if so, under what conditions?

Sequential Consistency

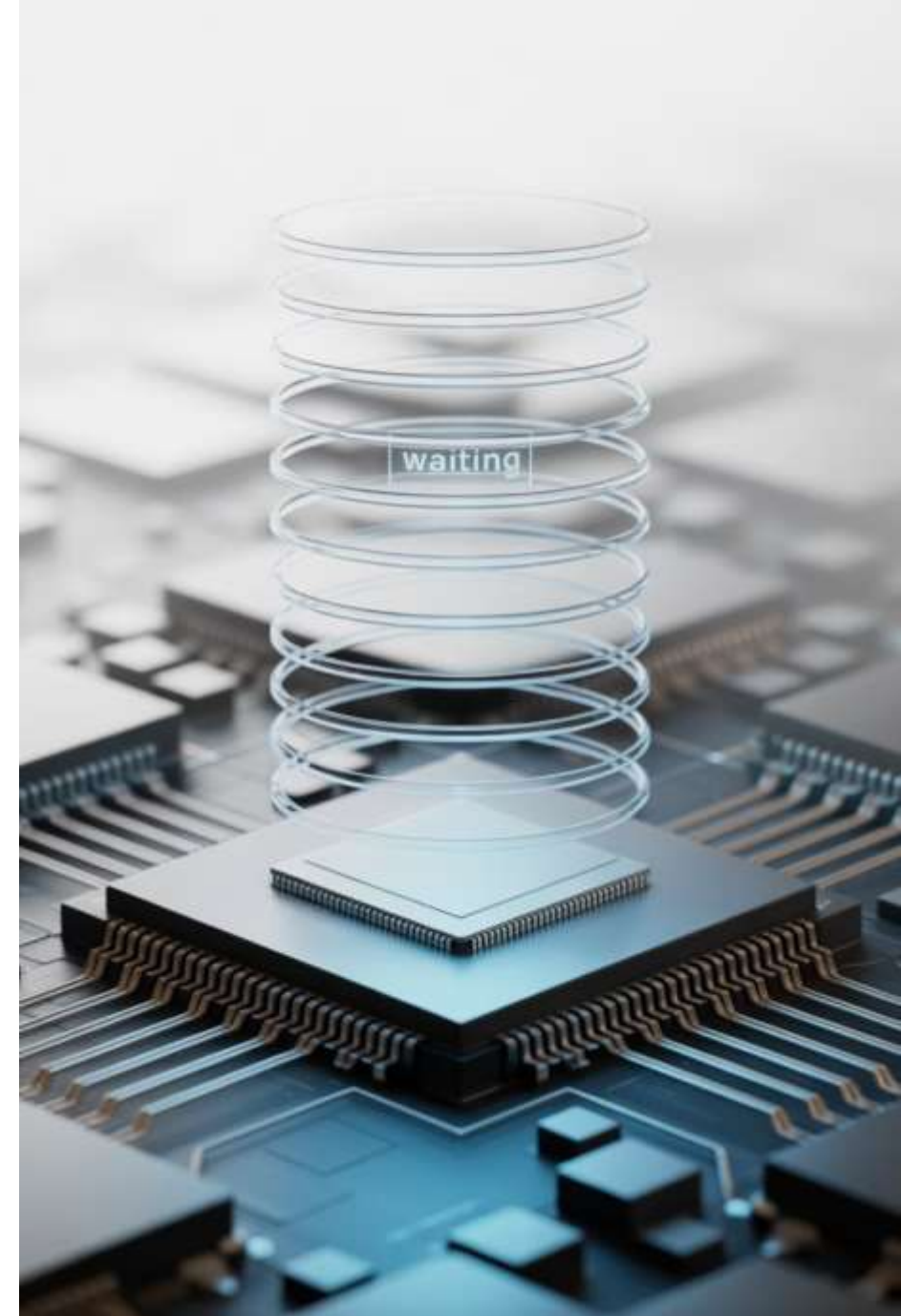
Sequential Consistency

Requires that execution results be the same as if memory accesses by each processor were kept in order and accesses among different processors were arbitrarily interleaved.

Implementation: A processor must delay completion of any memory access until all invalidations caused by that access are completed.

Performance Impact

Example: With 50 cycles for ownership, 10 cycles per invalidate, and 80 cycles for acknowledgment, a write miss with four sharing processors takes 170 cycles under sequential consistency.



Relaxed Consistency

Synchronized Programs

A program is synchronized if all accesses to shared data are ordered by synchronization operations.

Most programs are synchronized because unsynchronized programs have unpredictable behavior dependent on execution speed.

Programmers typically use standard synchronization libraries rather than creating their own mechanisms.

Relaxed Consistency Models

Allow reads and writes to complete out of order, but use synchronization to enforce ordering when needed.

Sequential consistency requires maintaining all four orderings: $R \rightarrow W$, $R \rightarrow R$, $W \rightarrow R$, and $W \rightarrow W$.

Relaxed models selectively relax these orderings:

- Total store ordering/processor consistency:
Relaxes $W \rightarrow R$
- Partial store order: Relaxes $W \rightarrow W$
- Weak ordering/release consistency:
Relaxes $R \rightarrow W$ and $R \rightarrow R$

Current Trends in Memory Consistency



Modern Implementations

Many current multiprocessors support some form of relaxed consistency model, from processor consistency to release consistency.



Programming Impact

Most programmers use standard synchronization libraries and write synchronized programs, making the choice of consistency model largely invisible.



Performance Considerations

Alternative viewpoint: With speculation, much of the performance advantage of relaxed consistency can be obtained with sequential or processor consistency.

The compiler's ability to optimize memory access to potentially shared variables remains a key consideration in the debate between consistency models.