CIS 505: Software Systems Lecture Note on Consistency and Replication

Instructor: Insup Lee
Department of Computer and Information Science
University of Pennsylvania

CIS 505, Spring 2007

Replication of Data

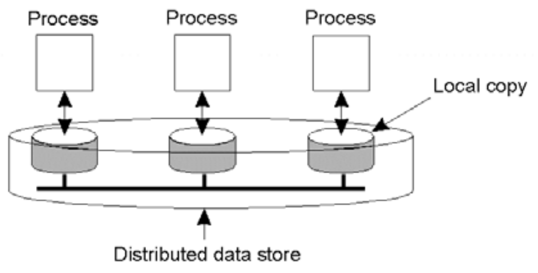
- Data are replicated
  - To enhance reliability
  - To improve performance
- Replication for performance
  - Scaling in numbers, scaling in geographic areas
  - Caveat - Cost of increased bandwidth for marinating replication
- Problems and issues: keeping them consistent
  - The notion of correctness: consistency models
    - Data-centric consistency
    - Client-centric consistency
  - Implementation Issues
    - Distribution of replicas
      - Placement of replicas
      - How to update them
    - How to keep replicas consistent
      - when to update them (immediate, lazy, pull/push, etc.)
      - Scalability (e.g., Web servers)

Reliability vs. Availability

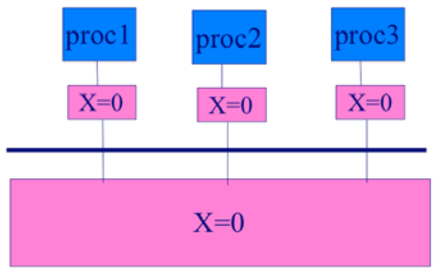
- Reliability: measured by time between system failures
  - Distributed system should be more reliable than single system
  - 3 machines with .95 probability of being up. 1 - .05**3 probability of being up.
- Availability: fraction of time the system is usable.
  - Redundancy improves it
  - Need to maintain consistency
  - Need to be secure
  - Need to tolerate failures: mask failures, recover from errors.
- Consistency tradeoff between improved reliability and performance vs. overheads in keeping them consistent
  - Weakening consistency can improve performance and availability

Data-Centric Consistency Models

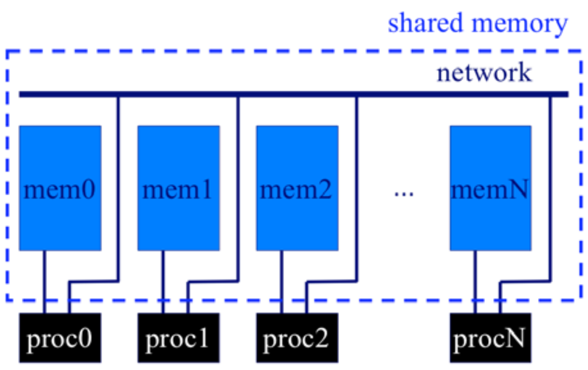
- The general organization of a logical data store, physically distributed and replicated across multiple processes.



Symmetric Multiprocessor (SMP)



Distributed Shared Memory (DSM)



Cache/Memory Coherence and consistency

- Replicated shared data and concurrent access by many processors
- Coherence: every cache/processor must have a coherent view of memory
  - Informally, memory is coherent if the value returned by a read op is always the value that the programmer expects, e.g., read mostly recently written value
  - If P writes X to A, then reads A, if no other proc writes A, then P reads X
  - If P1 writes X to A, and no other processor writes to A, then P2 will eventually read X from A.
  - If P1 writes X to A, and P2 writes Y to A, then *every* processor will either read X then Y, or Y then X, but all will see the writes in the same order.
- Consistency: memory consistency model tells us when writes to *different* locations will be seen by readers.

Consistency models

- Strict consistency
- Linearizability and sequential consistency
- Causal consistency
- FIFO consistency

Other weaker notions:

- Weak consistency
- Release consistency
- Entry consistency

Strict Consistency

```
P1:   W(x)a                          P1:   W(x)a
P2:                 R(x)a            P2:            R(x)NIL   R(x)a
            (a)                                  (b)
```

- Any read to a memory location X returns the value stored by the most recent write to X
  - o Total ordering on operations
  - o Inefficent
- Behavior of two processes, operating on the same data item.

  (a) A strictly consistent store.

  (b) A store that is not strictly consistent.

Sequential Consistency

```
P1:      W(x)a
P2:                      R(x)NIL   R(x)a
```

- Figure 7-4. Behavior of two processes operating on the same data item. The horizontal axis is time.
- Should this be allowed?

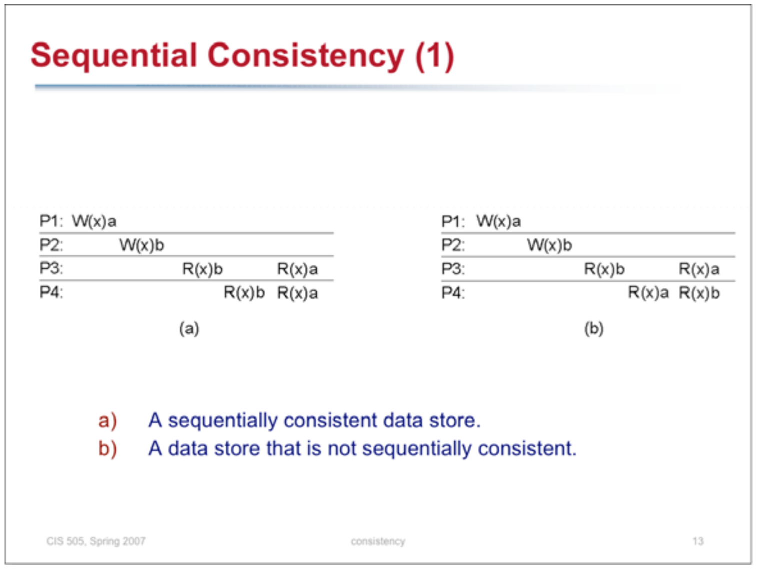
Sequential Consistency

- The result of any execution is the same as if
  - o the (read and write) operations by all processes were executed in some sequential order and
  - o the operations of each individual process appear in this sequence in the order specified by its program.

Linearizable & Squential Consistency Models

Consider a sequence σ of (possibly concurrent) Reads, Writes & local operations:

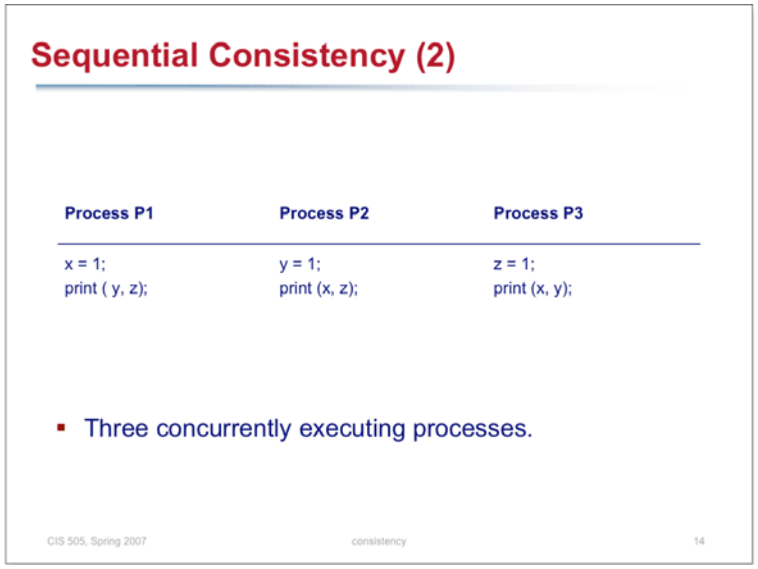
- Sequentially consistent: if consistent with a permutation of σ that preserves order of writes *within* each process.
- Linearizable: if sequentially consistent and also if ts of Op1(x) is before ts of Op2(y), then Op1(x) should precede Op2(y).



Sequential Consistency (1)

```
P1: W(x)a                              P1: W(x)a
P2:      W(x)b                         P2:      W(x)b
P3:            R(x)b    R(x)a          P3:            R(x)b    R(x)a
P4:                R(x)b R(x)a         P4:                R(x)a R(x)b
            (a)                                    (b)
```

a) A sequentially consistent data store.
b) A data store that is not sequentially consistent.

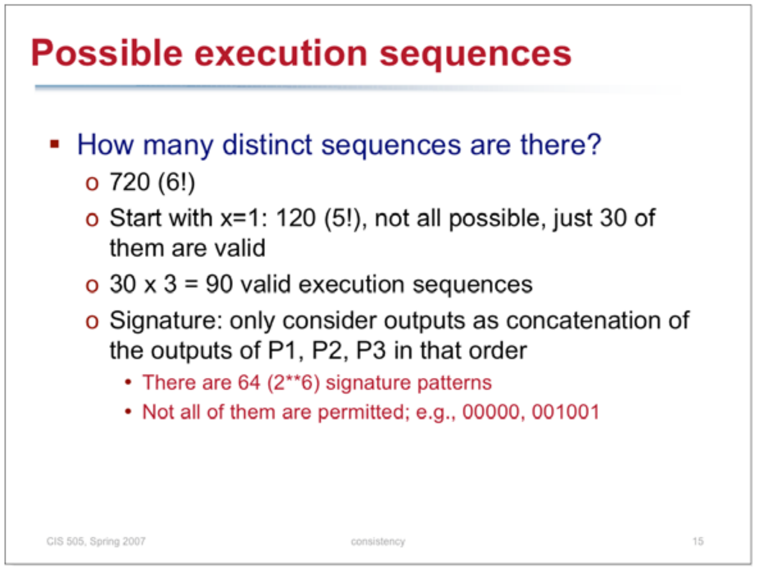


Sequential Consistency (2)

Process P1	Process P2	Process P3

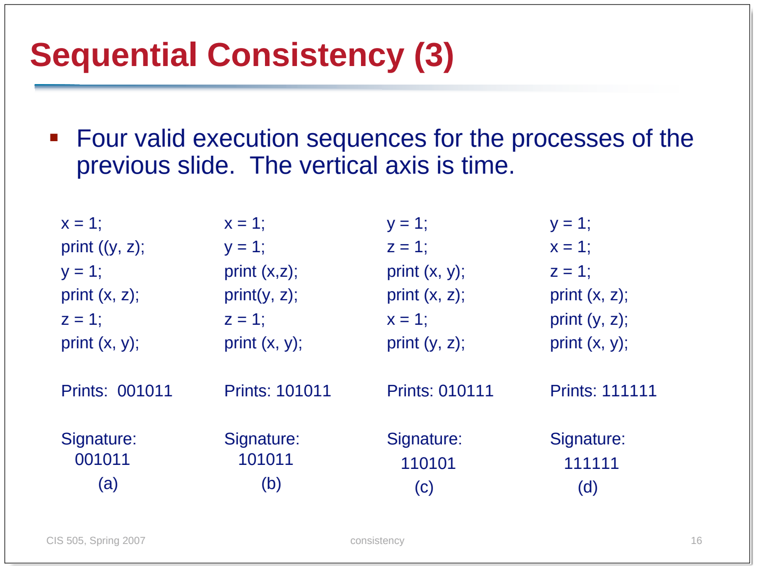
x = 1;
print (y, z);
	y = 1;
	print (x, z);
		z = 1;
		print (x, y);

- Three concurrently executing processes.



Possible execution sequences

- How many distinct sequences are there?
  - 720 (6!)
  - Start with x=1: 120 (5!), not all possible, just 30 of them are valid
  - 30 x 3 = 90 valid execution sequences
  - Signature: only consider outputs as concatenation of the outputs of P1, P2, P3 in that order
    - There are 64 (2**6) signature patterns
    - Not all of them are permitted; e.g., 00000, 001001



Sequential Consistency (3)

- Four valid execution sequences for the processes of the previous slide. The vertical axis is time.

x = 1;	x = 1;	y = 1;	y = 1;
print ((y, z);	y = 1;	z = 1;	x = 1;
y = 1;	print (x,z);	print (x, y);	z = 1;
print (x, z);	print(y, z);	print (x, z);	print (x, z);
z = 1;	z = 1;	x = 1;	print (y, z);
print (x, y);	print (x, y);	print (y, z);	print (x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Casual Consistency (1)

- Sequential consistency too restrictive
  - o Need to distinguish between events that are potentially causally related and those that are not.
- Definition
  - o Writes that are potentially casually related must be seen by all processes in the same order.
  - o Concurrent writes may be seen in a different order on different machines.

Casual Consistency (2)

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

- This sequence is allowed with a casually-consistent store, but not with sequentially or strictly consistent store.

Casual Consistency (3)

P1:	W(x)a				
P2:		R(x)a	W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

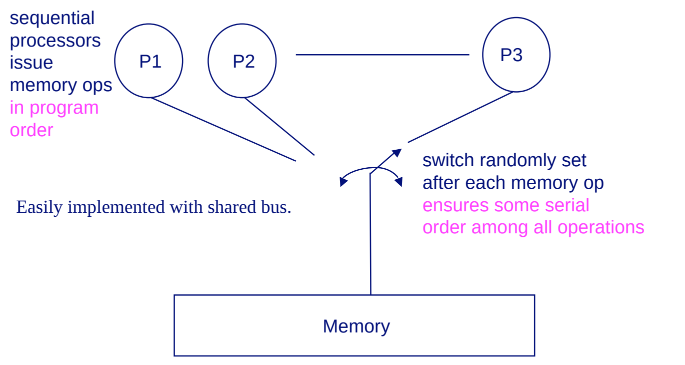
(a)

P1:	W(x)a				
P2:			W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

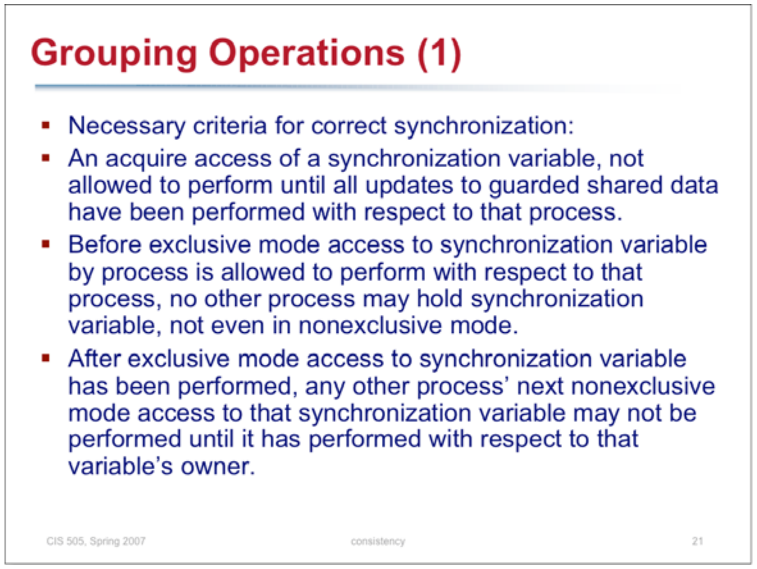
(b)

a) A violation of a casually-consistent store.
b) A correct sequence of events in a casually-consistent store.

The Sequential Consistency Memory Model

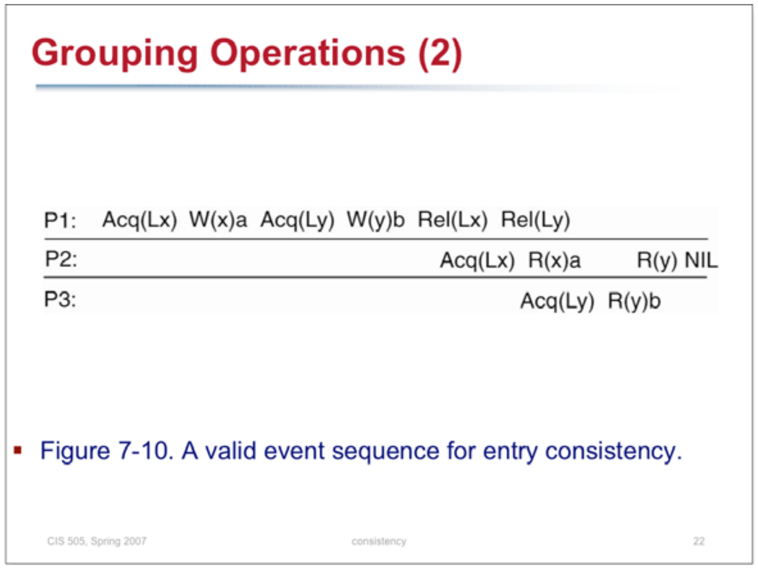

sequential processors issue memory ops in program order
P1
P2
P3
switch randomly set after each memory op ensures some serial order among all operations
Easily implemented with shared bus.
Memory

[Lebeck]



Grouping Operations (1)

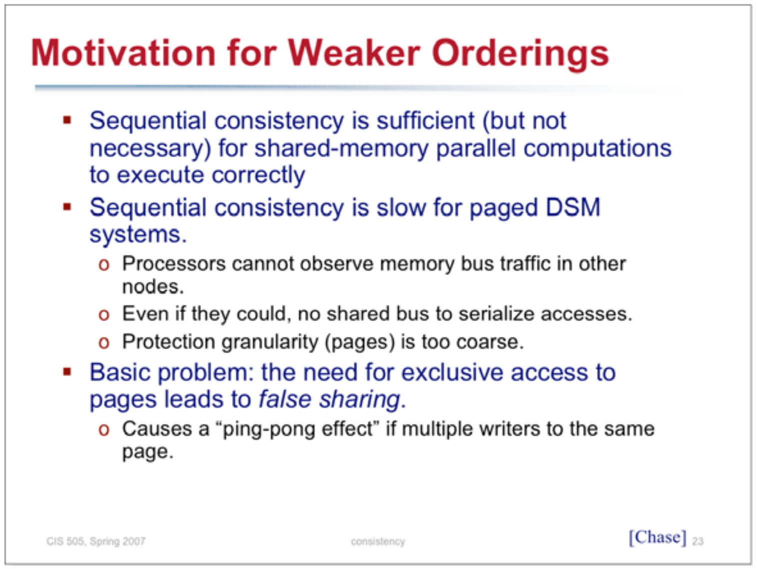
- Necessary criteria for correct synchronization:
- An acquire access of a synchronization variable, not allowed to perform until all updates to guarded shared data have been performed with respect to that process.
- Before exclusive mode access to synchronization variable by process is allowed to perform with respect to that process, no other process may hold synchronization variable, not even in nonexclusive mode.
- After exclusive mode access to synchronization variable has been performed, any other process' next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.



Grouping Operations (2)

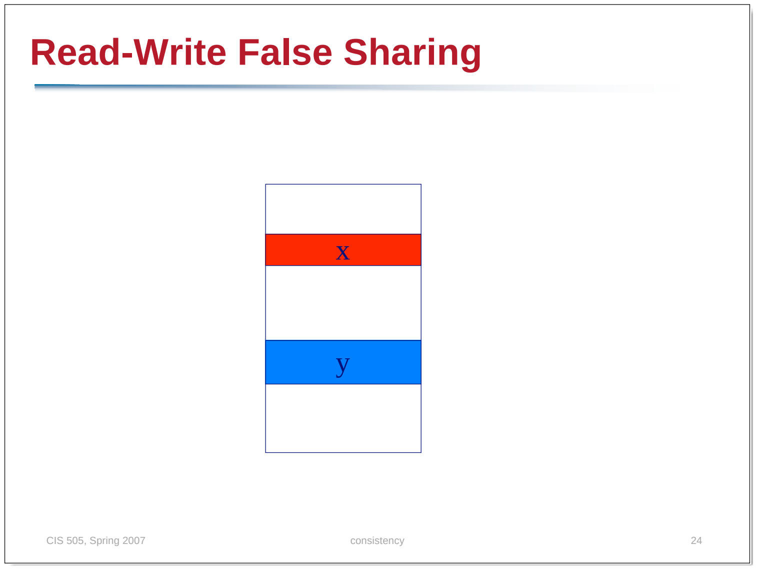
```
P1:  Acq(Lx)  W(x)a  Acq(Ly)  W(y)b  Rel(Lx)  Rel(Ly)
P2:                                    Acq(Lx)  R(x)a       R(y) NIL
P3:                                             Acq(Ly)  R(y)b
```

- Figure 7-10. A valid event sequence for entry consistency.

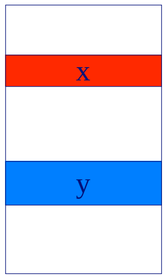


Motivation for Weaker Orderings

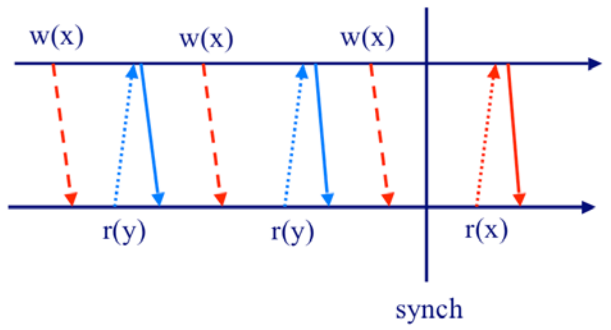
- Sequential consistency is sufficient (but not necessary) for shared-memory parallel computations to execute correctly
- Sequential consistency is slow for paged DSM systems.
  - o Processors cannot observe memory bus traffic in other nodes.
  - o Even if they could, no shared bus to serialize accesses.
  - o Protection granularity (pages) is too coarse.
- Basic problem: the need for exclusive access to pages leads to *false sharing*.
  - o Causes a "ping-pong effect" if multiple writers to the same page.



Read-Write False Sharing



Read-Write False Sharing (Cont.)



Weak Consistency Ordering

- Solution: allow *multiple writers* to a page if their writes are "nonconflicting".
- Careful access ordering only matters when data is shared.
  - Shared data should be synchronized.
- Classify memory operations as *data* or *synchronization*
- Can reorder data operations between synchronization operations
- Forces consistent view at all synchronization points
- Cannot let synch operation complete until previous operations complete (e.g., ACK all invalidations)

[Lebeck]

Weak consistency

- Accesses to synchronization variables associated with a data store are sequentially consistent.
- No operation on a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
- No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

Weak Consistency Example (1)

```
P1: W(x)a    W(x)b    S
P2:                       R(x)a   R(x)b    S
P3:                       R(x)b   R(x)a    S
```

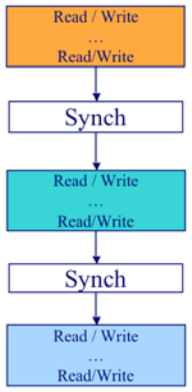
(a)

```
P1: W(x)a    W(x)b    S
P2:                              S   R(x)a
```

(b)

a) A valid sequence of events for weak consistency.
b) An invalid sequence for weak consistency.

Weak Consistency Example (2)



```
A                                B
                                 (x = y = 0;)
if (y > x)                       loop {
   panic("ouch");                   x = x + 1;
                                    y = y + 1;
                                 }
```

```
A                        B
acquire();               loop() {
if (y > x)                  acquire();
   panic("ouch");           x = x + 1;
release();                  y = y + 1;
                            release();
                         }
```

[Lebeck]

Release Consistency (RC)

- Data modifications are only propagated at the time of synchronization.
- Rules:
  - Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.
  - Before a release is allowed to be performed, all previous reads and writes by the process must have completed
  - Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).
- Works fine if program is properly synchronized through system primitives.
  - All programs should be …
  - No busy waiting!

Summary of Consistency Models

a) Consistency models not using synchronization operations.
b) Models with synchronization operations.

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(a)

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)