

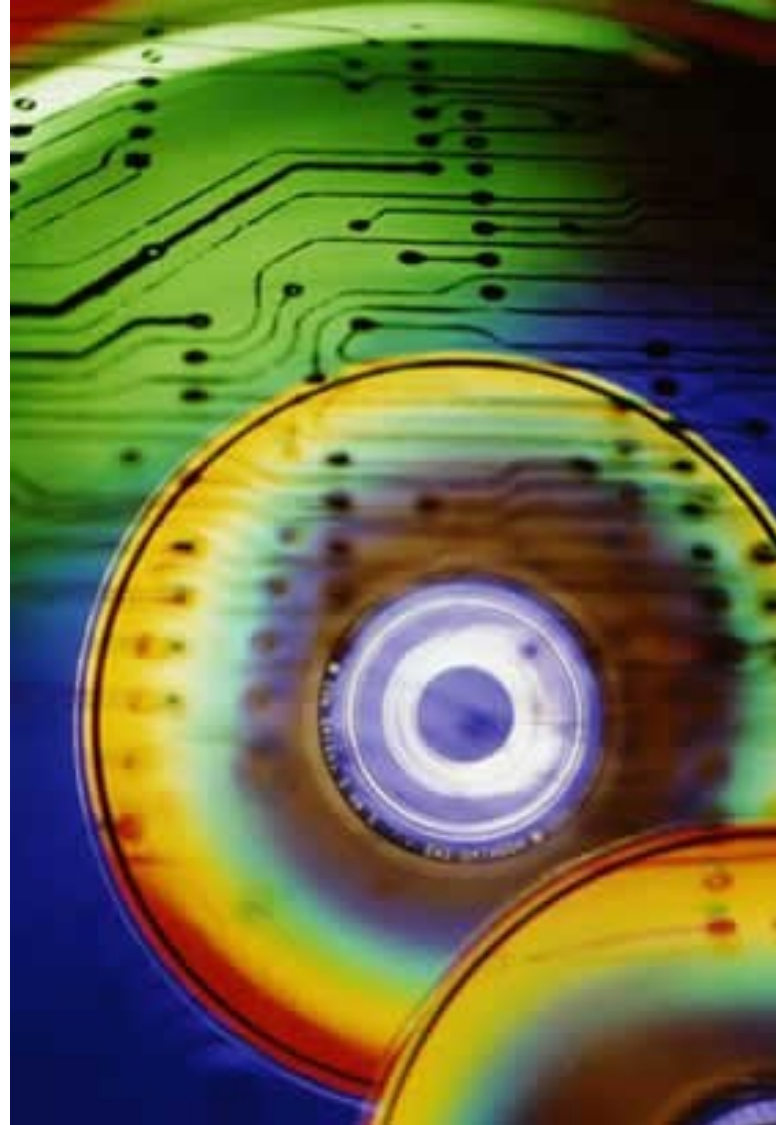
# ICS 362 Distributed Systems

***Distributed Systems: Part 14***

***Lecturer: Toby Daniel***

# Consistency and Replication

- Data Centric
- Client Centric
- Protocols



# Replication

- What is it?
- Why do it?

# Why Replicate?

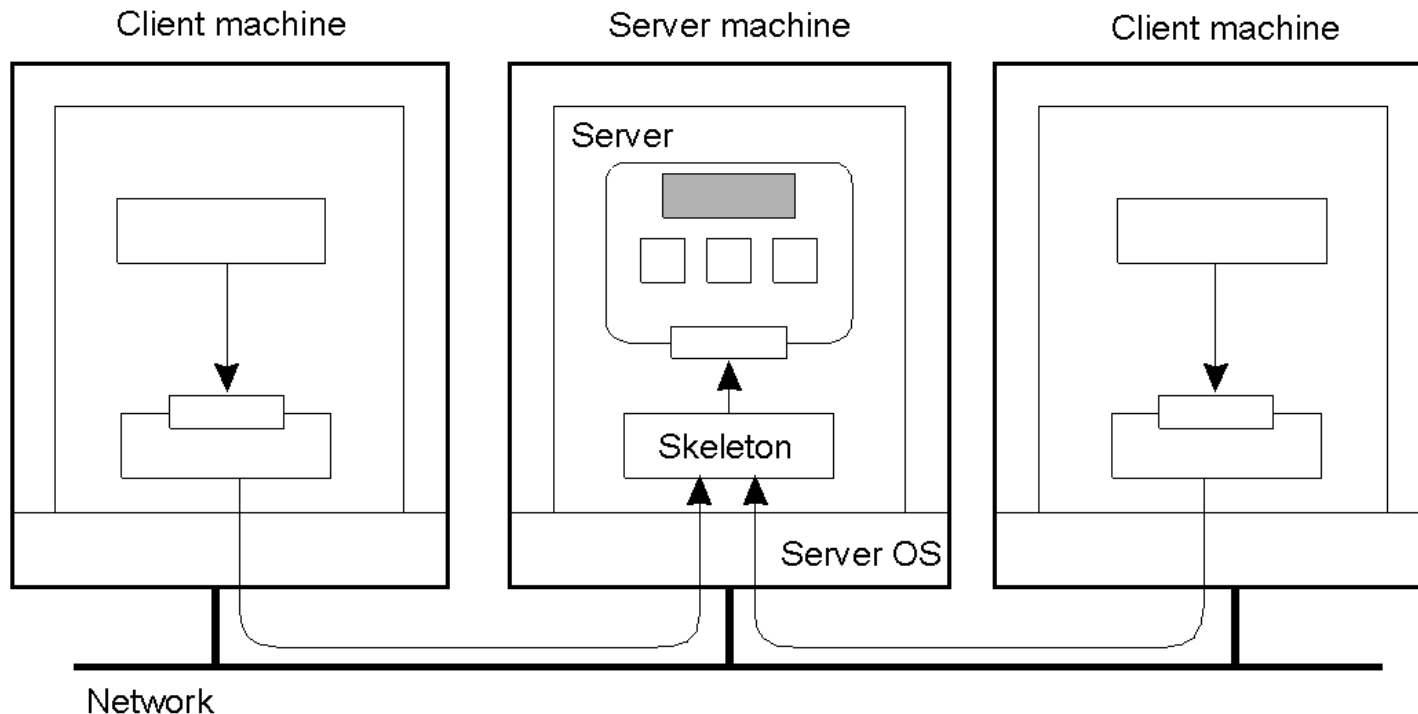
- **Enhance reliability.**
- **Improve performance.**
- Replicas allows remote sites to continue working in the event of local failures.
- It is also possible to protect against data corruption.
- Replicas allow data to reside close to where it is used.
- This directly supports the distributed systems
- goal of enhanced *scalability*.
- Even a large number of replicated “local” systems can improve performance.

# Problems?!

- What problems are associated with replication?



# Object Replication

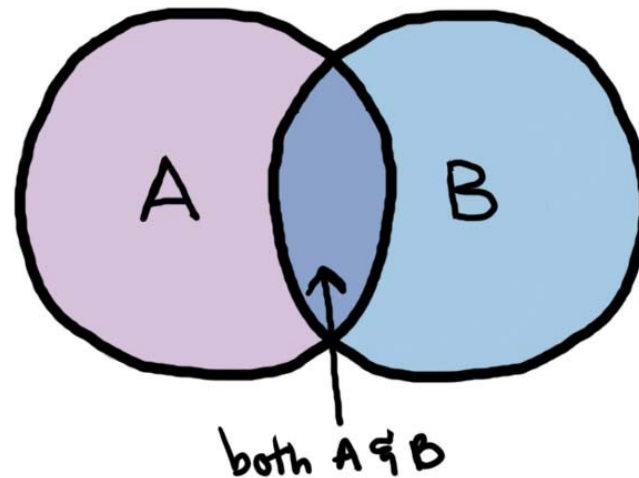


- Consider a remote object shared by multiple clients, how can we protect the object against concurrent access?

# Remember...

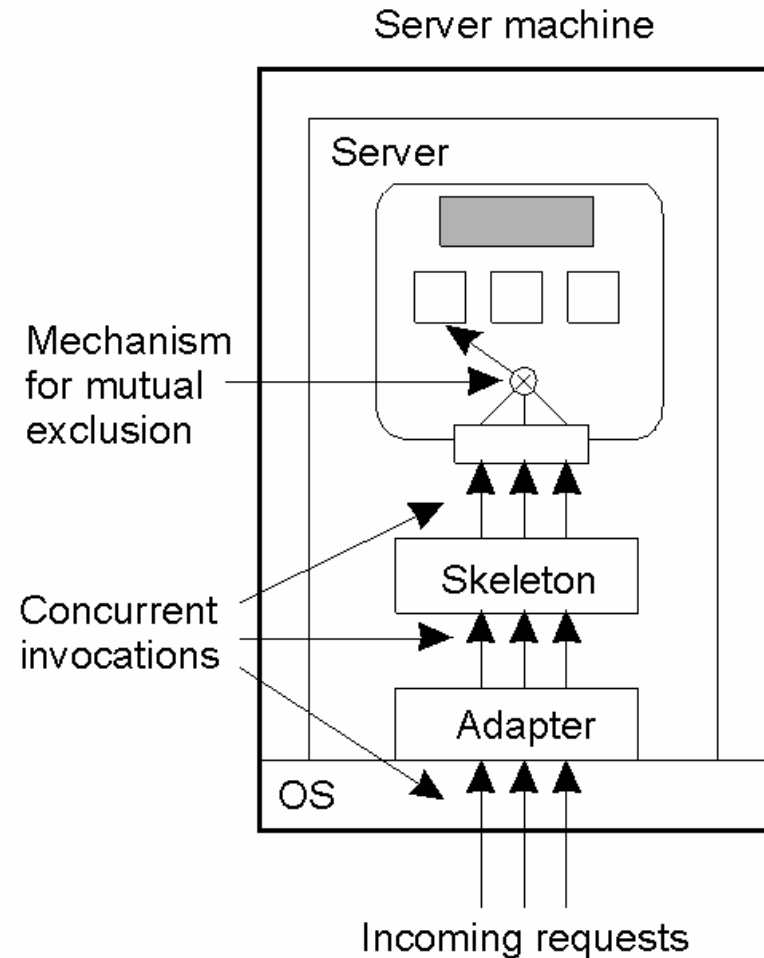
- Mutual Exclusion!

VENN DIAGRAM!



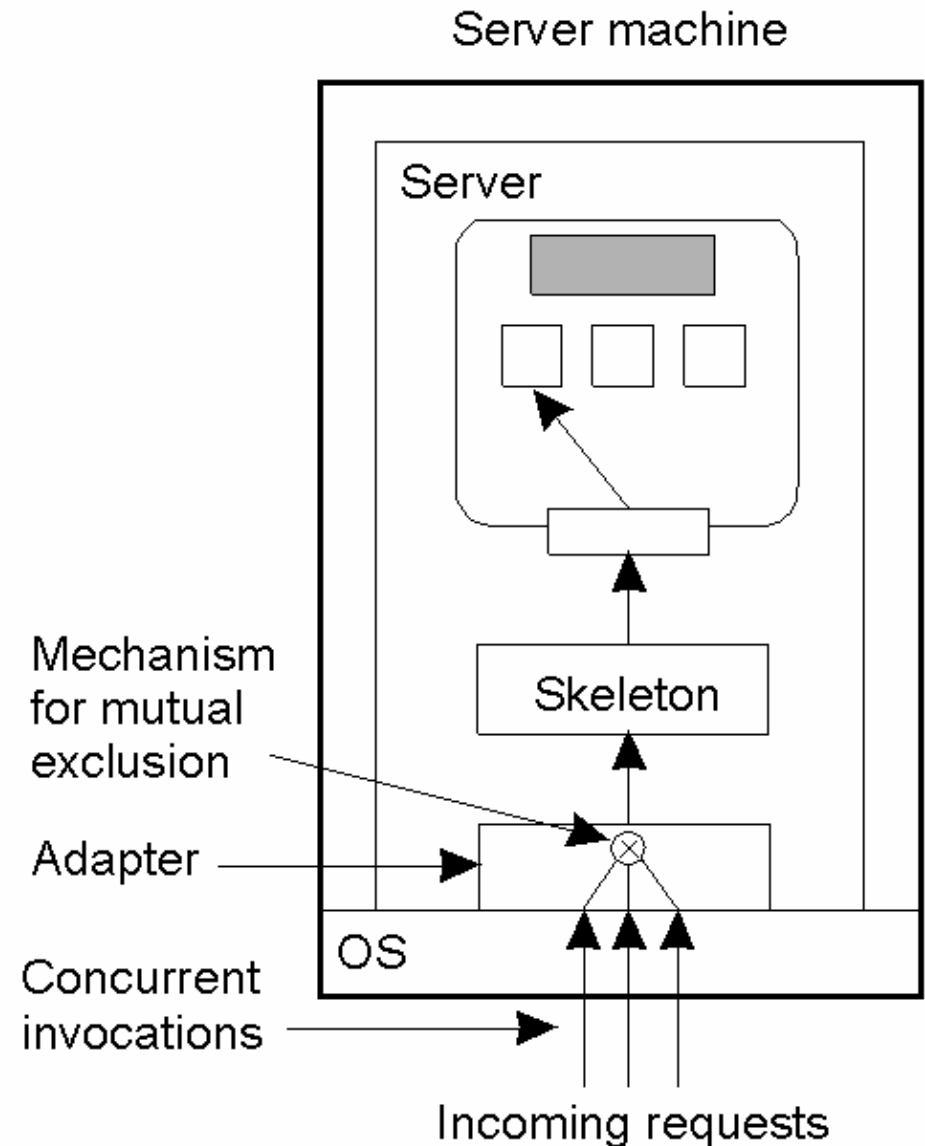
# Option 1

- Allow the object itself to handle concurrent invocations, such as in Java. While more than one thread might exist concurrently, one is blocked while another completes.



# Option 2

- Make the server where the object resides responsible for protecting the object.
- An object adapter can serialise accesses, managing concurrency.

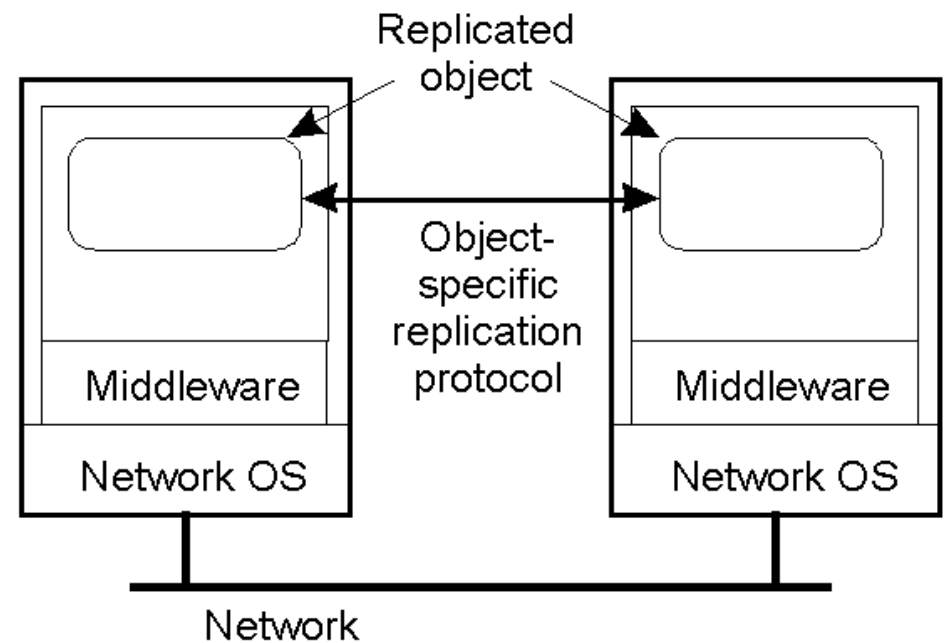


# Object Replication

- Next lets consider if we replicated the object across several servers (to gain reliability and performance advantages).
  - How can we protect multiple replicas against concurrent access?
  - How can we ensure consistency?

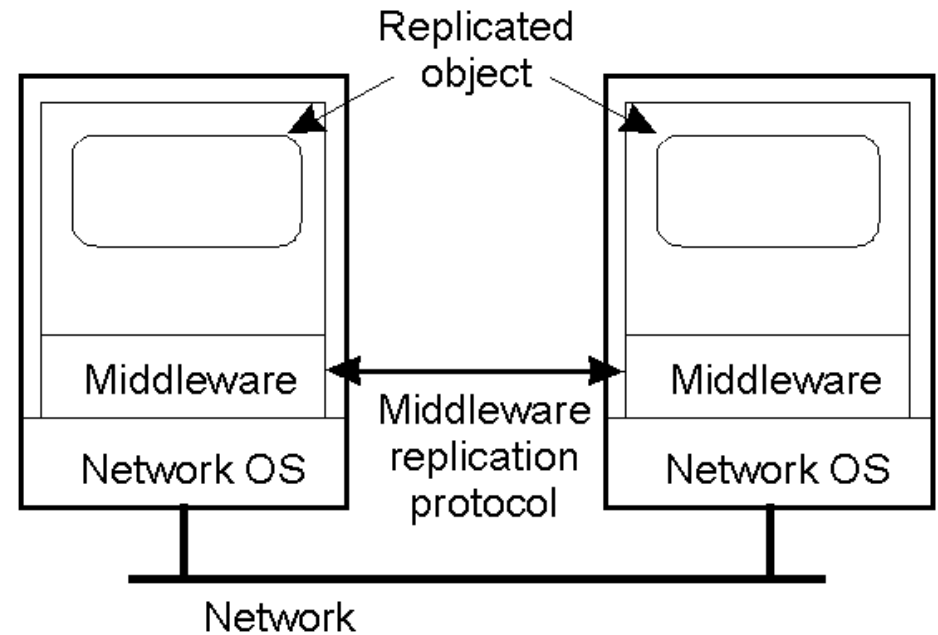
# Option 1

- Make the object aware that it can be replicated, and hence responsible for maintaining consistency across replicas.
- The key advantage of this option is that it allows object specific replication strategies.



# Option 2

- A commoner solution is to make the distributed system responsible for maintaining consistency.
- The system must ensure that requests are passed to replicas in the correct order.
- This makes life a lot easier for the application developer.



# Scalability

- Let us return to one of our key requirements... scalability
- What are some of the problems associated with scaling up in size and location?

# Scalability by Replication

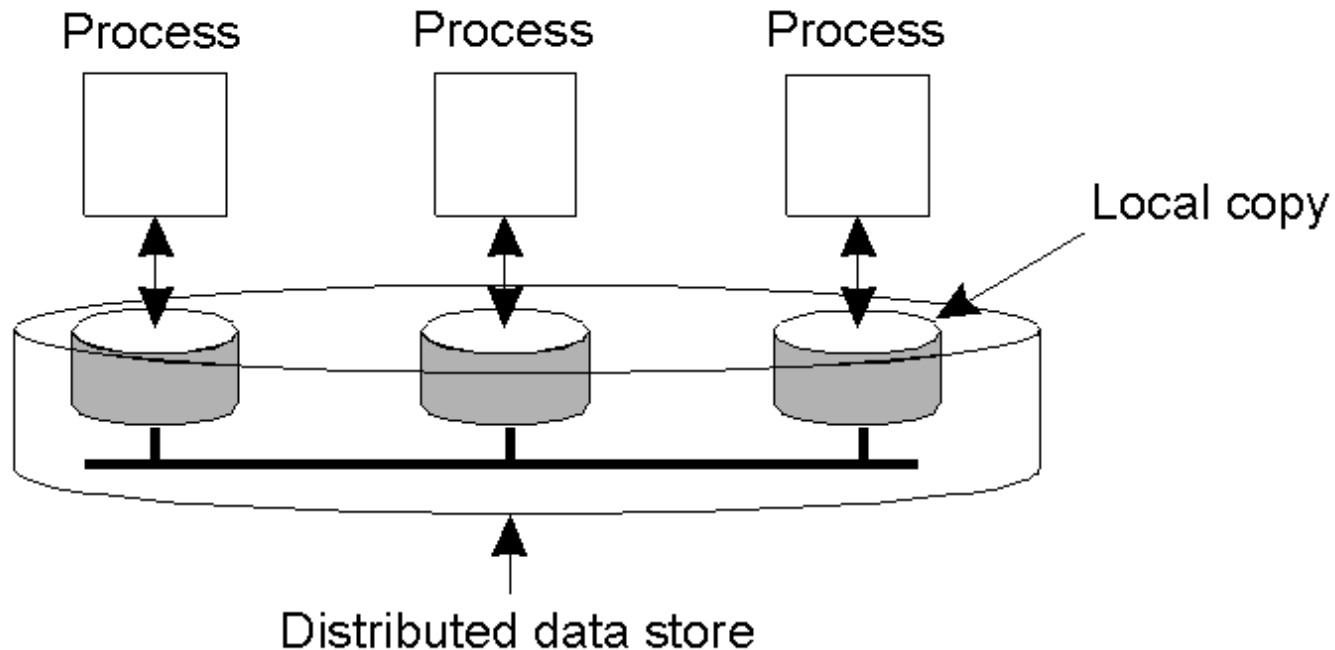
- When systems scale, the first problems to surface are those associated with performance – as the systems get bigger (e.g., more users), they get often slower.
- Replicating the data and moving it closer to where it is needed helps to solve this scalability problem.
- Just consider the web.

# Scalability by Replication (2)

- But, while adding replicas improves scalability, but incurs the (oftentimes considerable) overhead of keeping the replicas up-to-date.
- The solution is often though relaxing consistency constraints.
- Just consider the web.

# Data Centric Consistency Models

- Distributed Data Store (could be shared memory, databases or file systems).
- Read operations can be made on local replica, while Write operations need to be propagated across *all* replicas.



# Strict Consistency

- The Rules:
  - *Any read on a data item 'x' returns a value corresponding to the result of the most recent write on 'x' (regardless of where the write occurred).*
- Consider this code;
  - `int a=1; a=2; cout << a;`
- If this displayed 1 (or anything other than 2) you'd get frustrated as a programmer.
- But that is because you are used to strict consistency!

# Strict Consistency

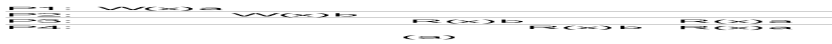
- For strict consistency to succeed we need a global time (remember discussion on synchronisation).



# Sequential Consistency

- The Rules:
  - *The result of any execution is the same as if the (read and write) operations by all processes on the data-store were executed in the same sequential order and the operations of each individual process appear in this sequence in the order specified by its program.*
- Essentially this means that all operations must see the same interleaving of operations; processes are aware of their own reads, but everyone's writes.

# Sequential Consistency (2)



- (a) demonstrates an acceptable interleaving of reads and writes, while (b) is unacceptable.
- Consider updating the football scores, in the correct order.

# Causal Consistency

- The Rules:
  - *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines (i.e., by different processes).*
- This model distinguishes between events that are “causally related” and those that are not.
- If event B is caused or influenced by an earlier event A, then causal consistency requires that every other process see event A, then event B.



# Causal Consistency (2)



- (a) is not valid as P2's write is related to P1's write due to the read on 'x' giving 'a' (all processes must see them in the same order).
- (b) is valid as now the two writes are concurrent



# FIFO Consistency

- The Rules;
  - *Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.*
- FIFO further weakens the rules by allowing causally related writes to be seen in a different order on a different machine.



# Weak Consistency Models

- Even the FIFO model might be too restrictive for some applications, so the rules are relaxed further with weak consistency models and implemented using a synchronisation variable;
  1. Accesses to synchronisation variables associated with a data-store are *sequentially consistent*.
  2. No operation on a synchronisation variable is allowed to be performed until all previous writes have been completed everywhere.
  3. No read or write operation on data items are allowed to be performed until all previous operations to synchronisation variables have been performed.

# Weak Consistency (2)

- By doing a sync., a process can *force* the just written value out to all the other replicas.
- Also, by doing a sync., a process can be *sure* it's getting the most recently written value before it reads.
- In essence, the weak consistency models enforce consistency on a *group of operations*, as opposed to individual reads and writes (as is the case with strict, sequential, causal and FIFO consistency).



# Syncs

- Clearly there is a difference between two different syncs;
  - A sync due to a write (to update all other replicas)
  - A sync due to a read (to ensure most up to date value)

# Release Consistency

- The Rules;
  - *When a process does an “acquire”, the data-store will ensure that all the local copies of the protected data are brought up to date to be consistent with the remote ones if needs be.*
  - *When a “release” is done, protected data that have been changed are propogated out to the local copies of the data-store.*
- Release consistency uses two different sync variables (‘acquire’ and ‘release’) depending on which type of sync is required.



# Entry Consistency

## 1. The Rules

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

# Entry Consistency (2)

- 
- Entry consistency locks individual data items to ensure that no other processes are accessing that data item at that time.
  - Note; P2's read on 'y' returns NIL as no lock has been requested.

# Summary of Consistency Models

<b>Consistency</b>	<b>Description</b>
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.

<b>Consistency</b>	<b>Description</b>
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.