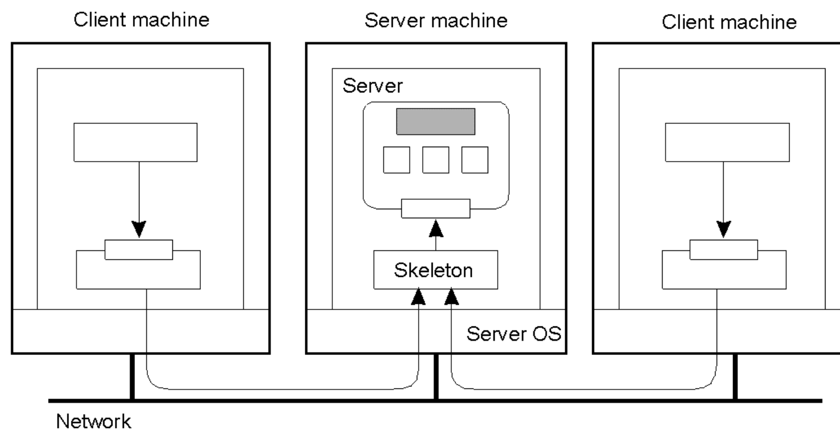
Replicas: why and why not?

- Reliability
  - Tolerance to component failures
  - Tolerance to corrupted data
- Performance
  - Benefits scalability
  - Allows for concurrent access to resources (data/objects/processors)
  - Reduces the delay for geographically dispersed resources
- Disadvantages
  - Cost of replications is high
  - Maintaining consistency of multiple copies is tough
  - Implementation is harder (e.g., different users have different needs of number of replicas and more or less accurate consistency models)

Object Replication

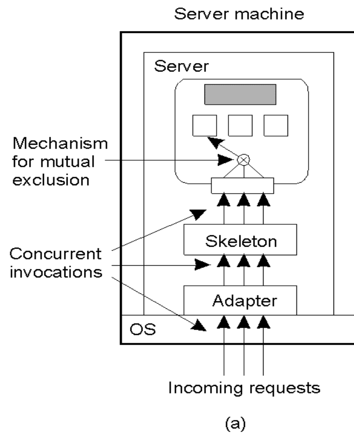
Problem: If objects (or data) are shared, we need to do something about concurrent accesses to guarantee state consistency.

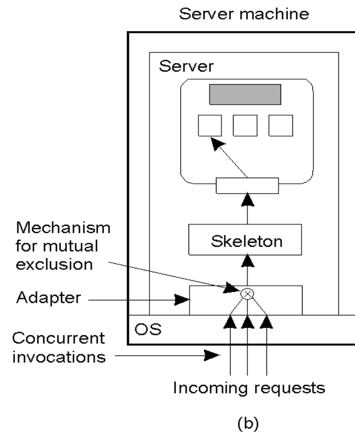


Object Replication solutions

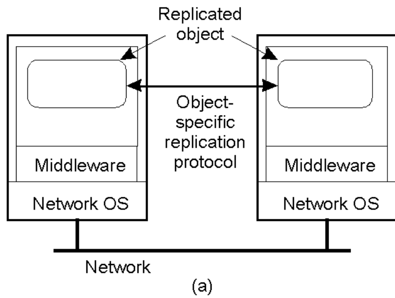
A remote object capable of handling concurrent invocations on its own.

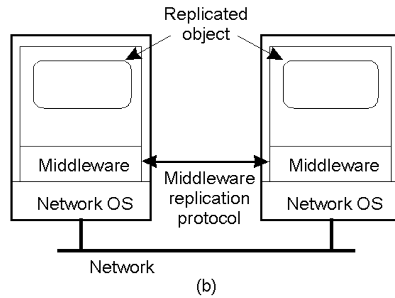
A remote object for which an object adapter is required to handle concurrent invocations





Where in the world is… Object Replication





A distributed system for replication-aware distributed objects is more customizable

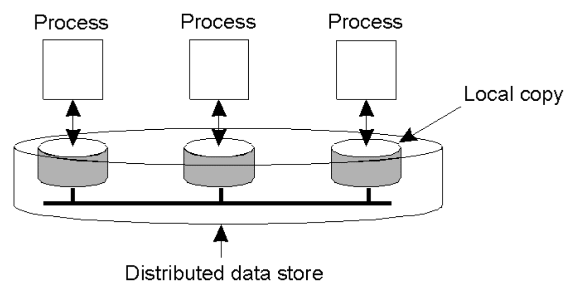
A distributed system responsible for replica management is more general and easier to implement

Performance, replication, scalability

- **Main issue:** To keep replicas consistent, we generally need to ensure that all *conflicting* operations are done in the the same order everywhere
- **Conflicting operations (e**xample from transactions):
  - Read–write conflict: a read operation and a write operation act concurrently
  - Write–write conflicts: two concurrent write operations
- Tradeoff: guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability
- **Solution:** weaken consistency requirements so that hopefully global synchronization can be avoided
- This solution only lends itself to some applications, not all.

Data-Centric Consistency Models (1)

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



- **Consistency model:** a contract between a (distributed) data store and processes, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency. Processes agree or don't use it.

Data-Centric Consistency Models (2)

- **Strong consistency models:** Operations on shared data are synchronized:
  - Strict consistency (related to “absolute global” time)
  - Sequential consistency (what we are used to)
  - Causal consistency (maintains only causal relations)
  - FIFO consistency (maintains only individual ordering)
- **Weak consistency models:** Synchronization occurs only when shared data is locked and unlocked:
  - General weak consistency
  - Release consistency
  - Entry consistency
- **Observation:** The weaker the consistency model, the easier it is to build a scalable solution.

Strict Consistency

Any read to a shared data item X returns the value stored by the most recent write operation on X.

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

A strictly consistent store. A store that is not strictly consistent.

- It may be expensive to maintain strict consistency
- Does everyone need it? Who does?
- How can it be better/easier/less costly?
- **Note:** Strict consistency is what you get in the “normal” uniprocessor, sequential case, where your program does not interfere with any other program.

Sequential Consistency

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

A strictly consistent store. A store that is not strictly consistent.

- This is for *interleaved* executions: there is ONE total ordering for all operations

Linearizability

Sequential consistency + operations are ordered according to a global time.

- This may be more practical, since it assumes loosely synchronized clocks (Lamport clocks? NTP?)
- Since the definition of "global time" is loose, so is the consistency model.
- Therefore, linearizability is weaker than strict consistency, but stronger than sequential consistency. Happy medium.

Casual Consistency (1)

- Events *a* and *b* are causally related if *a* causes or influences *b*.
- Events that are not causally-related are *concurrent.*
- *Causal consistency*: 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.

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 consistency (what writes are concurrent?)

Casual Consistency (2)

A violation of a casually-consistent store. WHY?

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)

A correct sequence of events in a casually-consistent store. WHY?

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

(b)

FIFO Consistency

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.

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

A valid sequence of events of FIFO consistency

Is it valid for causal consistency?

What about sequential consistency?

FIFO Consistency

- Implementation is simple:
  - Attach a PID+sequence# to each event
  - Perform writes according to the this ordering

Process P1	**Process P2**
x = 1;	y = 1;
if (y == 0) kill (P2);	if (x == 0) kill (P1);

Two concurrent processes.

What's the beef?

Summary of Consistency Models

not using 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

Weak Consistency (1)

Properties:

- 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 been 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.

Implementation: use a synchronization phase

Basic idea: You don’t care that reads and writes of a *series* of operations are immediately known to other processes. You just want the *effect* of the series itself to be known.

Weak Consistency (2)

Observation: Weak consistency implies that we **need to** lock and unlock data (implicitly or not).

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 valid sequence of events for weak consistency.

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

An invalid sequence for weak consistency.

Release Consistency (1)

Idea: Divide access to a synchronization variable into two parts
- **Acquire phase:** forces a requester to wait until the shared data can be accessed
- **Release phase:** sends requestor's local value to other servers in data store.

P1:	Acq(L)	W(x)a	W(x)b	Rel(L)				
P2:					Acq(L)	R(x)b	Rel(L)	
P3:								R(x)a

Release Consistency (2)

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).

Entry Consistency (1)

Conditions:

- 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.
- 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.
- 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)

- With release consistency, all local updates are propagated to other copies/servers during release of shared data.
- With entry consistency, each shared data item is associated with a synchronization variable.
- When acquiring the synchronization variable, the most recent values of its associated shared data item are fetched.

- **Note:** Where release consistency affects *all* shared data, entry consistency affects only those shared data associated with a synchronization variable.

- **Question:** What would be a convenient way of making entry consistency more or less transparent to programmers?

Entry Consistency (3)

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	

A valid event sequence for entry consistency.

Summary of Consistency Models

with synchronization operations

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.

Summary of Consistency Models

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

Consistency models not using synchronization operations

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.

Models with synchronization operations.