Exam and Term Project

- Exam
  - Exam, Monday, April 8, 2024
  - Time: 3:05pm – 5:05pm; Room XXxxx
  - Close book, Close note, no phone, calculator, or computer or any internet access

- Term Project
  - Presentation, Monday and Tuesday, April 22
  - Term project report, April 25, 2024
  - Progress report March 20

Chap. 7: Consistency and Replication

- Data-centric Consistency Models
- Client-centric Consistency Model
- Replica Management
- Consistency Protocols

Introduction: Replication

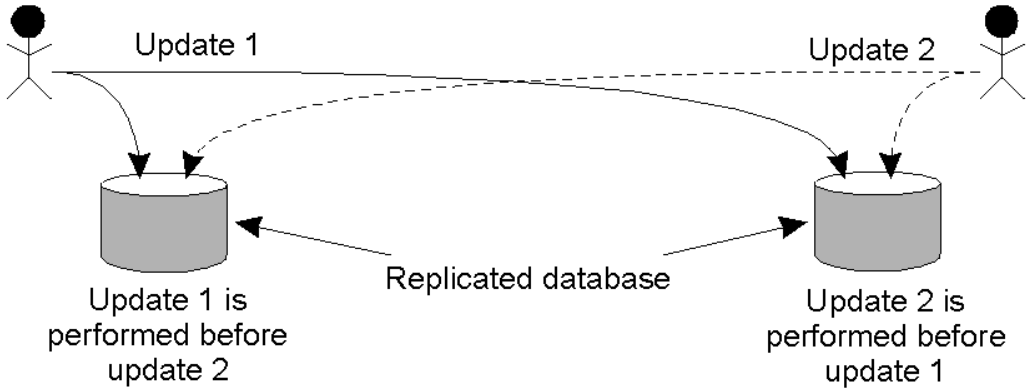
- What replicate?
  - Data, File, process, object, service, etc
- Why replicate?
  - Enhance performance:
    - Caching data in clients and servers to reduce latency
    - Server farm to share workload
  - Increase availability
    - Reduce downtime due to individual server failure through server replication
    - Support network partitions and disconnected operations through caching (e.g. email cache in POP3)
  - Fault tolerance
    - Highly available data is not necessarily current data. Fault-tolerant service should guarantee correct behavior despite a certain number and type of faults.
    - Correctness: freshness, timeliness
    - If f servers can exhibit byzantine failures, at least 2f+1 servers needed

Challenges

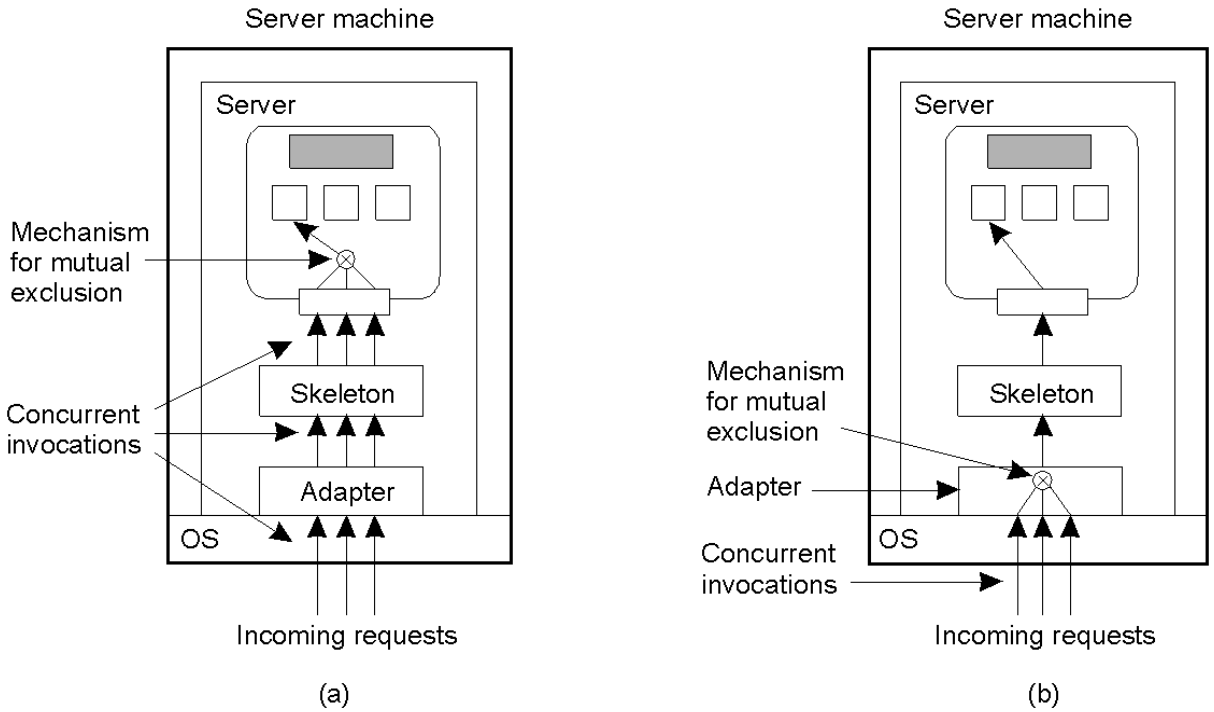
- Replication Transparency
  - Clients should not be aware of the presence of multiple physical copies; data are organized as logical objects.
  - Exactly one set of return
- Consistency
  - Whether the operations on a collection of replicated objects produce results that meet the specification of correctness for those objects
  - A read should return the value of last write operation on the data
  - Global synchronization for atomic update operations (transactions), but they are too costly
  - Relax the atomic update requirement so as to avoid instantaneous global synchronizations
    - To what extent consistency can be loosened
    - Without a global clock, how to define the "last"? Overhead?
- Overhead for keeping copies up to date

Replication of Shared Objects

- Replicating a shared remote object without correctly handling of concurrent invocations lead to consistency problem.
  - W(x) → R(x) in one site, but R(x)→ W(x) in another site
- Replicas need additional sync. to ensure that concurrent invocations are performed in a correct order at each replica.
  - Need support for concurrent access to shared objects
  - Need support for coordination among local and remote accesses



Concurrent Access to（single) Object



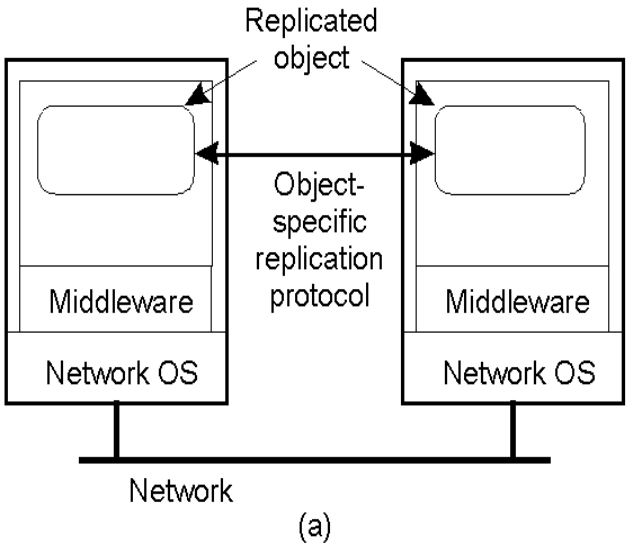
(a) A remote object capable of handling concurrent invocations on its own. (e.g. Java synchronized object)

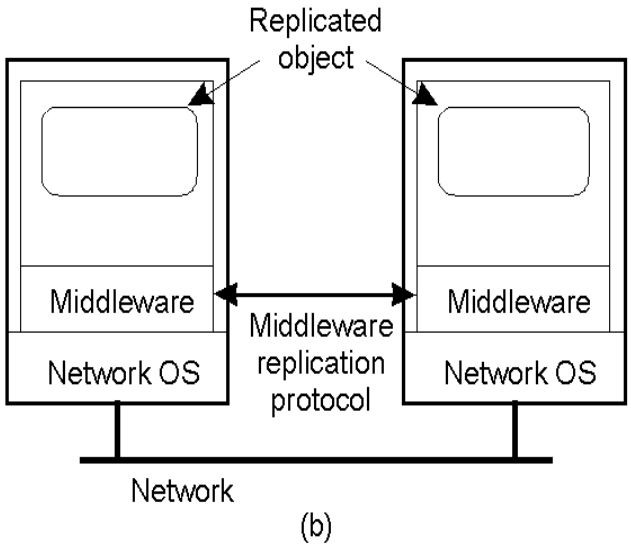
(b) A remote object for which an object adapter is required to handle concurrent invocations

Object Replication

- Approach 1: application is responsible for replication
  - Application needs to handle consistency issues
  - Pro: can adopt object-specific replication strategies
- Approach 2: system (middleware) handles replication
  - Consistency issues are handled by the middleware
  - Pro: simplicity for application developer

Corn?

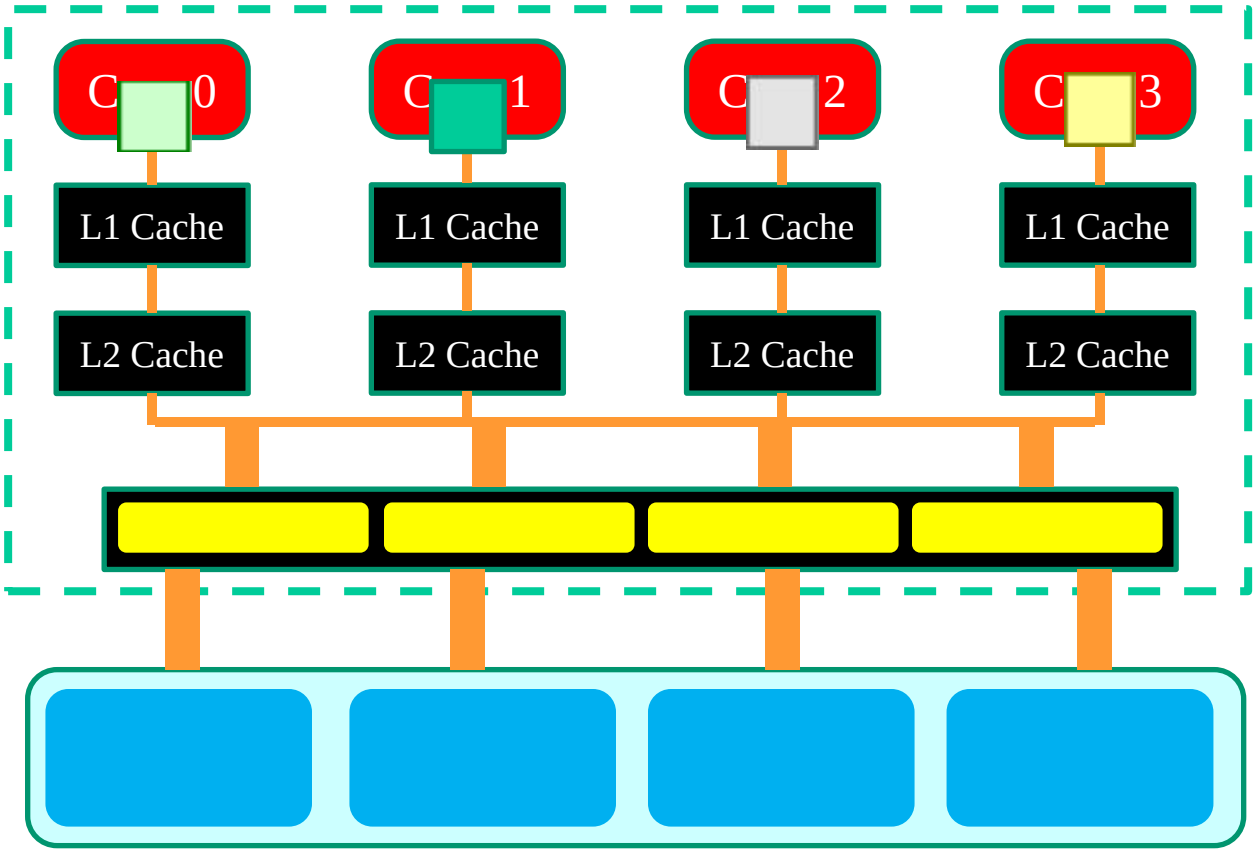




Replication and Scaling

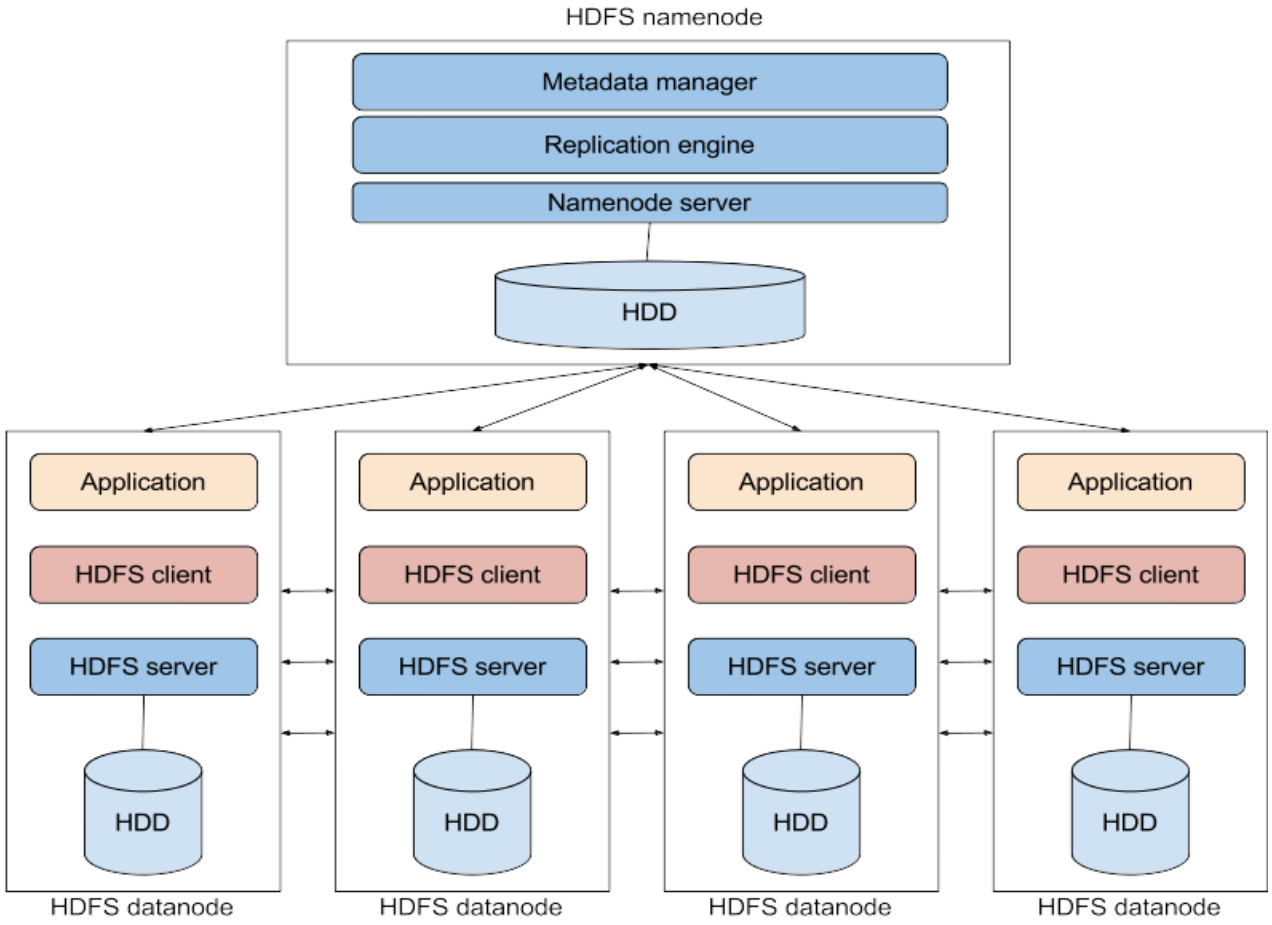
- Replication and caching used for system scalability
- Multiple copies:
  - Improves performance by reducing access latency
  - But higher network overheads of maintaining consistency
  - Example: object is replicated *N* times
    - Read frequency *R*, write frequency *W*
    - If *R*<<*W*, high consistency overhead and wasted messages
    - Consistency maintenance is itself an issue
      - What semantics to provide?
      - Tight consistency requires globally synchronized clocks!
- Solution: loosen consistency requirements
  - Variety of consistency semantics possible

Multicore Data Access



Concurrency and Duplication

Industrial Solution: Distributed I/O Systems



Architecture of a typical Hadoop File System

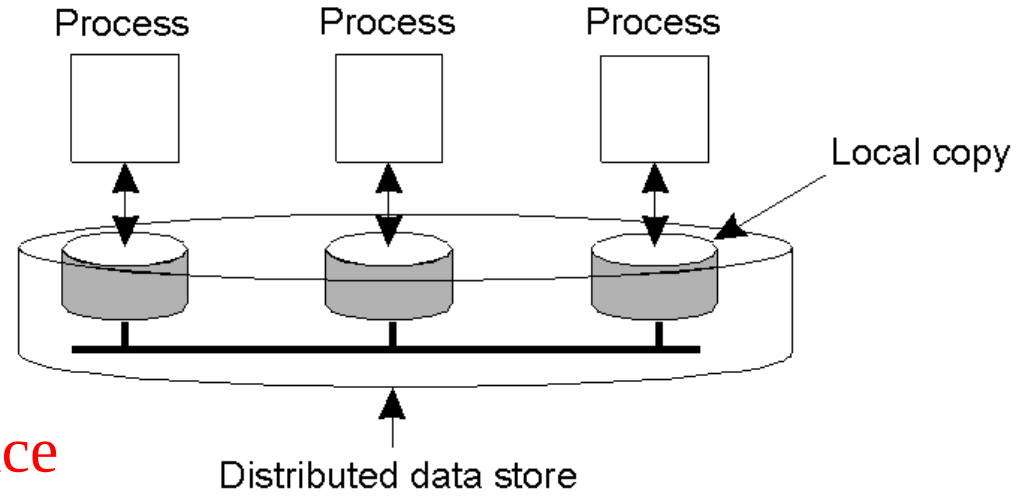
Locality with Duplication

Chapter 7 (Model)

- Data Consistency Model
- Client-centric models
- Eventual consistency and Epidemic protocols
- Distribution protocols
  - Invalidate versus updates, Push versus Pull, Cooperation between replicas
- Implementation issues (consistency protocols)
  - Primary-based, Replicated-write, Cache-coherence
- Putting it all together
  - Final thoughts
- Replica placement

Consistency Models

- A consistency model is a contract between processes and the data store. If processes agree to obey certain rules, the store promises to work correctly.
- A consistency model specifies the consistency guarantees that the data store makes about the values that processes read from objects, given that they actually access a replica of each object and that multiple processes may update the objects.



The right consistence

Overview of Consistency Models

- Strict Consistency
- Linearizable Consistency
- Sequential Consistency
- Causal Consistency
- FIFO Consistency
- Weak Consistency
- Release Consistency
- Entry Consistency

Ease of Programming
Less Efficiency

Hard to Programming
More Efficient

All attempt to return the results of the last write

Differ in how “last” write is determined/defined

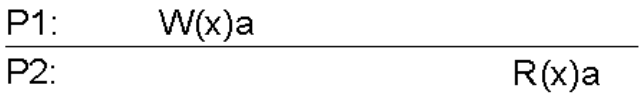
Strict Consistency

- Read of data item x returns value of the most recent write on x.
  - Most recent in the sense of absolute global time
  - E.g x=1; x=2; print(x) in the same process P1 ?
  - E.g x=1 (at P1 @5:00pm); x=2 (at P2 @ 1 ns after 5:00pm); print(x) (at P1)?
- A strictly consistent store requires all writes are instantaneously visible to all processes.

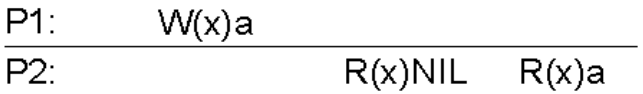
(a) Strictly consistent; (b) Non-strictly consistent

Strict Consistency

- Any read always returns the result of the most recent write. It implicitly assumes
  - Relies on absolute global time
  - Impossible in distributed systems
  - Time intervals (one action per interval), timestamps
  - What kind of behavior is acceptable?
  - Sometime, impossible



(a)



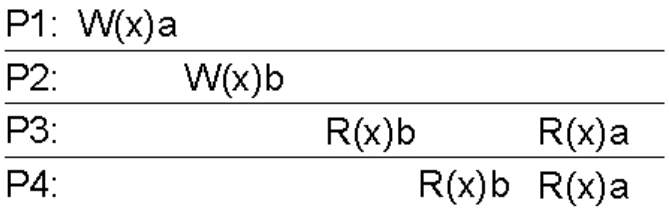
(b)

Sequential Consistency

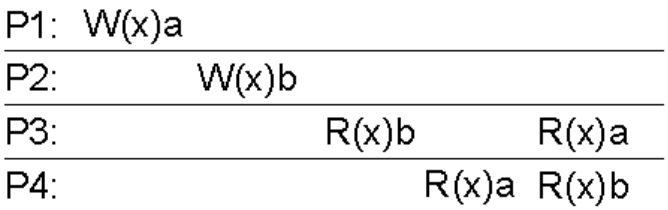
- A data store is sequentially consistent if

  i) The result of any execution is the same as if the ops by all processes on the data store were executed in some sequential order

  ii) Ops of each process appear in this sequence in the order specified by its program



(a)



(b)

a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

Possible Executions in Seq. Consistent Model

Assume initially x=y=z=0

Process P1	Process P2	Process P3
x = 1;	y = 1;	z = 1;
print (y, z);	print (x, z);	print (x, y);

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 (a)	Signature: 101011 (b)	Signature: 010111 (c)	Signature: 111111 (d)

Is signature 001001 valid by SC model ?

Remarks on the SC Model

- SC model was conceived by Lamport in 1979 for shared memory of multiprocessors.
  - Maintain program order of each thread
  - Coherence view of any data item
- Low efficiency in implementation
  - Each read/write op must be atomic.
  - Atomic read/write on distributed shared objects costs too much

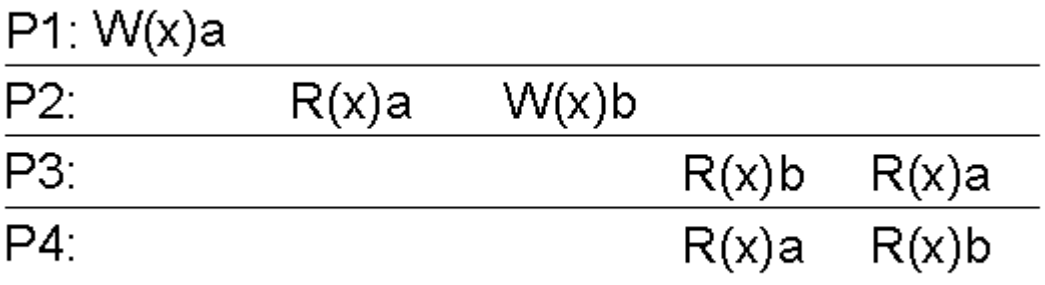
Causal Consistency

- In SC, two ops of w(x) and w(y) must provide consistent view to all processes.
- In CC, the requirement is relaxed if w(x) and w(y) have no causal relationship (Hutto and Ahamad, 90)
- That is, 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.

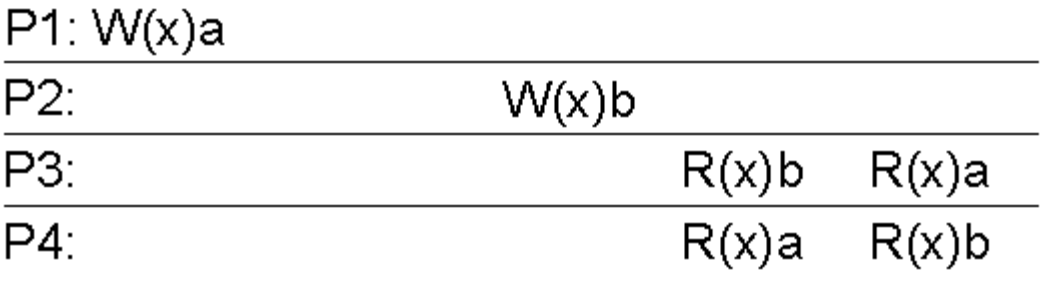
P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

Valid in CC, but invalid in SC

Causal Consistency (Cont)



(a)



(b)

a) A violation of a casually-consistent store.

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

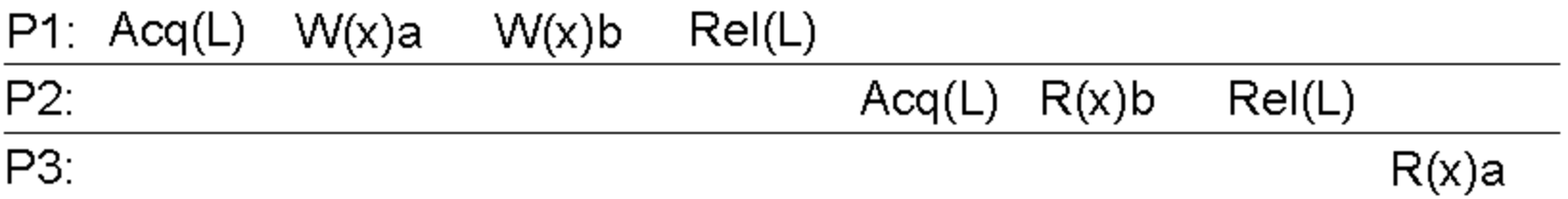
Remarks on Causal Consistency

- Causal Consistency Model allows concurrent writes to be seen in a different order on different machines. So
- Concurrent writes can be executed in parallel (overlapped) to improve efficiency
- It relies on a compiler/run-time support for constructing and maintaining a dependency graph that captures causality between the operations.

Release Consistency

- Sync. op is performed when a process wants to enter or exit from a critical section.
- Release consistency distinguishes between Acquire and Release sync operations (Gharachorloo et al. 1990)
  - When Acquire is performed, bring up all protected data to be consistent with remote ones
  - When Release is performed, propagate all updated protected data

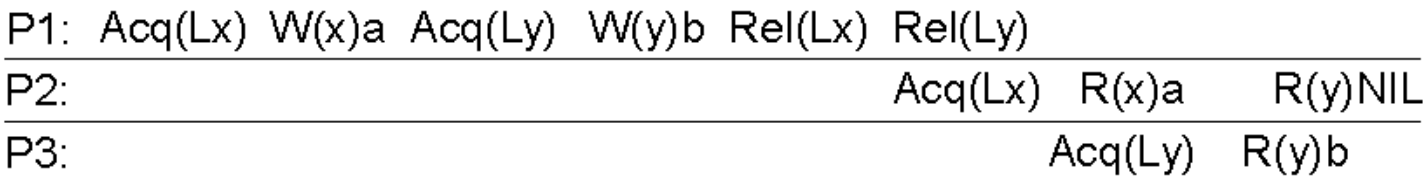
A valid event sequence for release consistency.



Two operations, Acquire and Release, instead of one, for efficiency

Entry Consistency

- Unlike RC where Acquire and Release are to protect a collection of shared items, EC requires each ordinary shared item to be associated with some sync. variables like lock or barrier.
  - Associating each shared data with a sync. variable reduces the overhead due to acquire and release because only a few need to be synchronized
  - Multiple critical sections involving disjoint shared data to be run concurrently
  - But, complex association and error-prone programming
- EC is fine-grained. Lazy RC needs to determine empirically, at acquire time, which variables it needs.



A valid event sequence for entry consistency

Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
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.

(a)

Consistency	Description
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)

(a) Consistency models not using synchronization operations.

(b) Models with synchronization operations.

Chapter 7 (continuation)

- Data Consistency Model
  - Client-centric models
  - Eventual consistency and Epidemic protocols
- Distribution protocols
  - Invalidate versus updates, Push versus Pull, Cooperation between replicas
- Implementation issues (consistency protocols)
  - Primary-based, Replicated-write, Cache-coherence
- Putting it all together
  - Final thoughts
- Replica placement

Eventually Consistent Distributed Data Store

- Characteristics:
  - Most ops involve reading data
  - Lack of simultaneous updates
  - A relatively high degree of inconsistency can be tolerated
- Examples
  - In most of database systems, only one or very few processes have privileges to update data items
  - In distributed web servers, web pages are usually updated by a single authority;
  - Web cache may not return the latest contents, but this inconsistency acceptable in some situations
- In such special distributed data stores, all replicas will gradually become consistent, if there are no updates for a long time.

Eventual Consistency

- Assume a replicated database with few updaters and many readers
- *Eventual consistency*:
  - Definition: if no more updates, all replicas will gradually become consistent
  - Only requirement: guaranteed propagate
  - Cheap to implement: ?
  - Things work fine as long as user accesses the same replica
  - What if they don't? Mobile users?