CS 550:
Advanced Operating Systems

Consistency

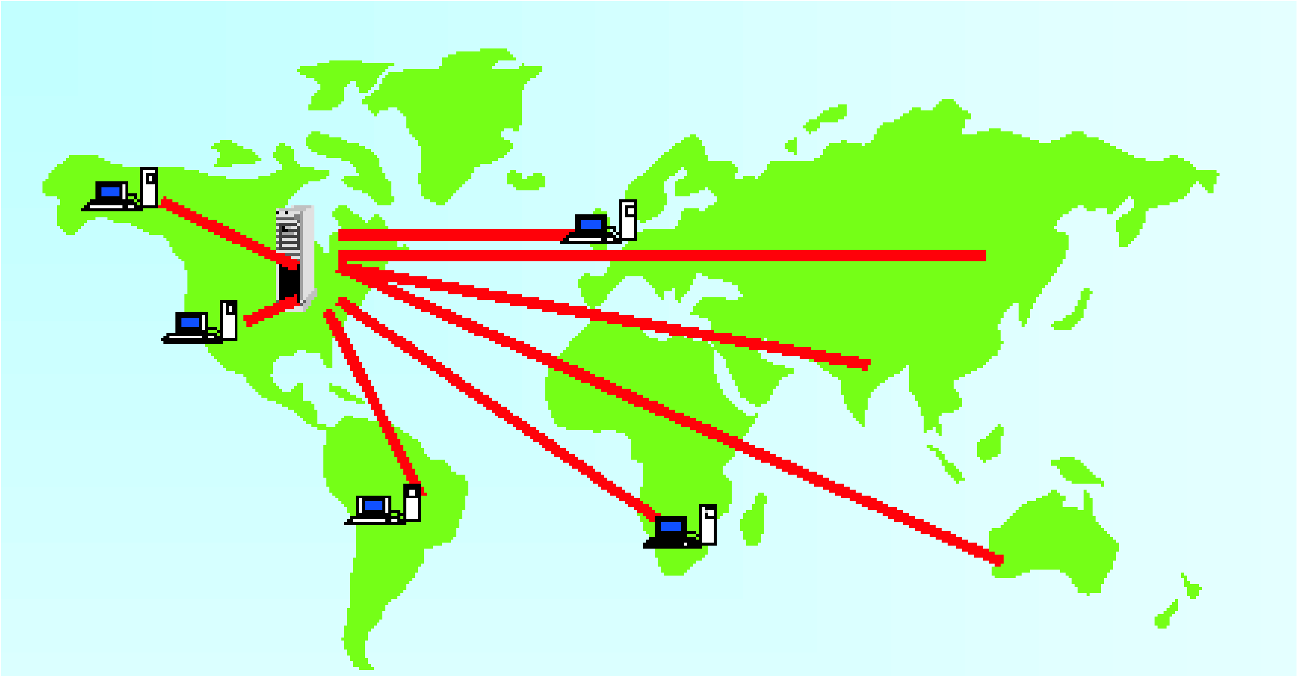
Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
March 3rd, 2011

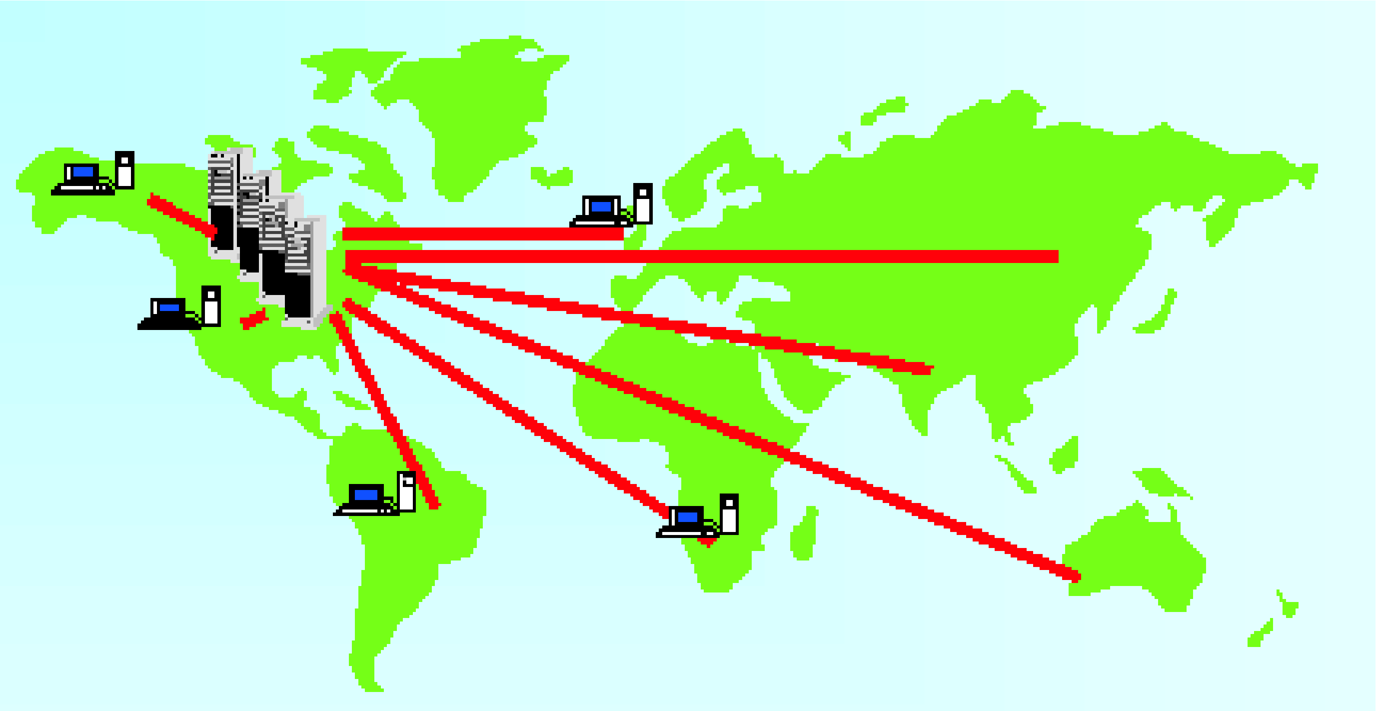
Outline

- Replication
  - Motivation
- Consistency models
  - Data/Client-centric consistency models
- Replica placement
- Distribution protocols
  - Invalidate versus updates
  - Push versus Pull
  - Cooperation between replicas
- Client-centric models
- Eventual consistency and Epidemic protocols
- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence

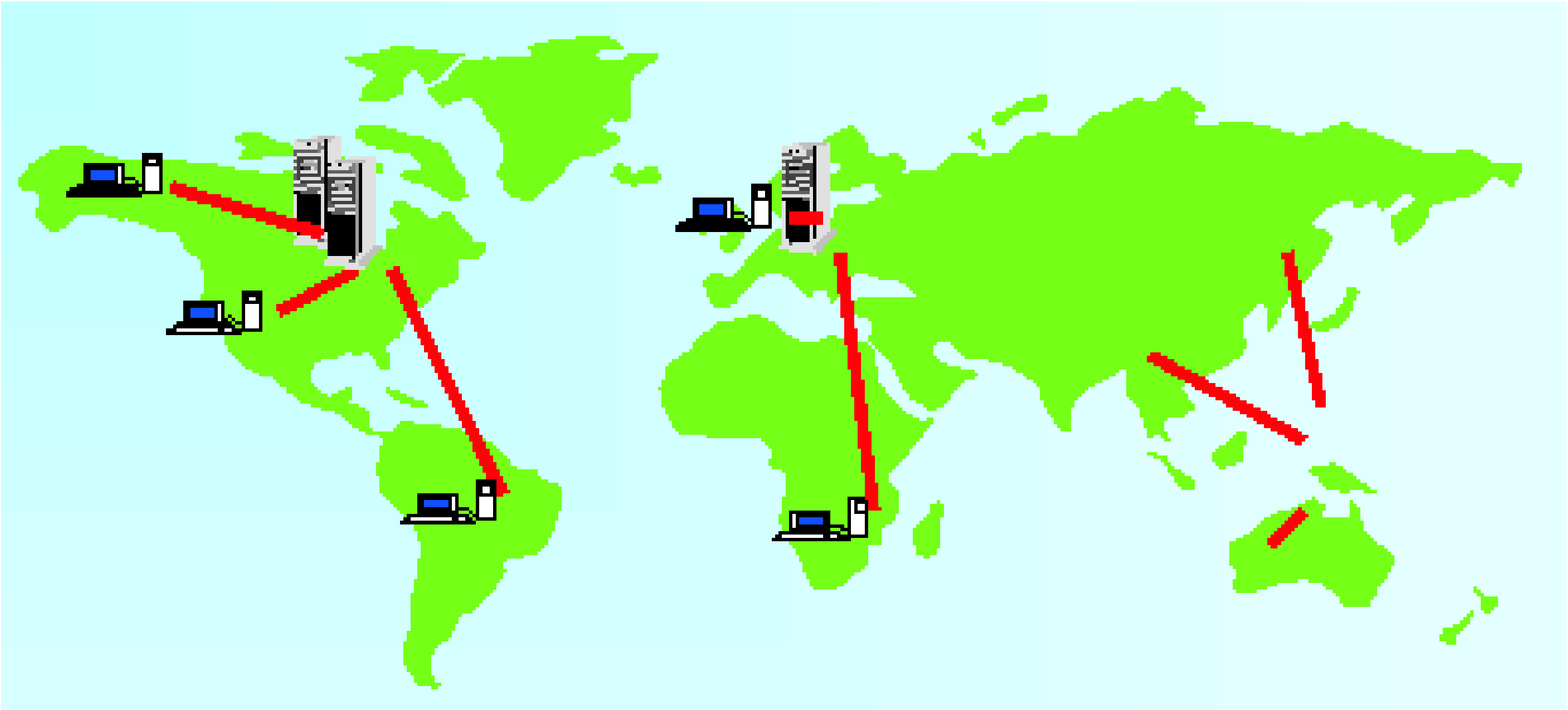
Example: Client-Sever Model



Example: Server Replication



Example: Wide Area Replication

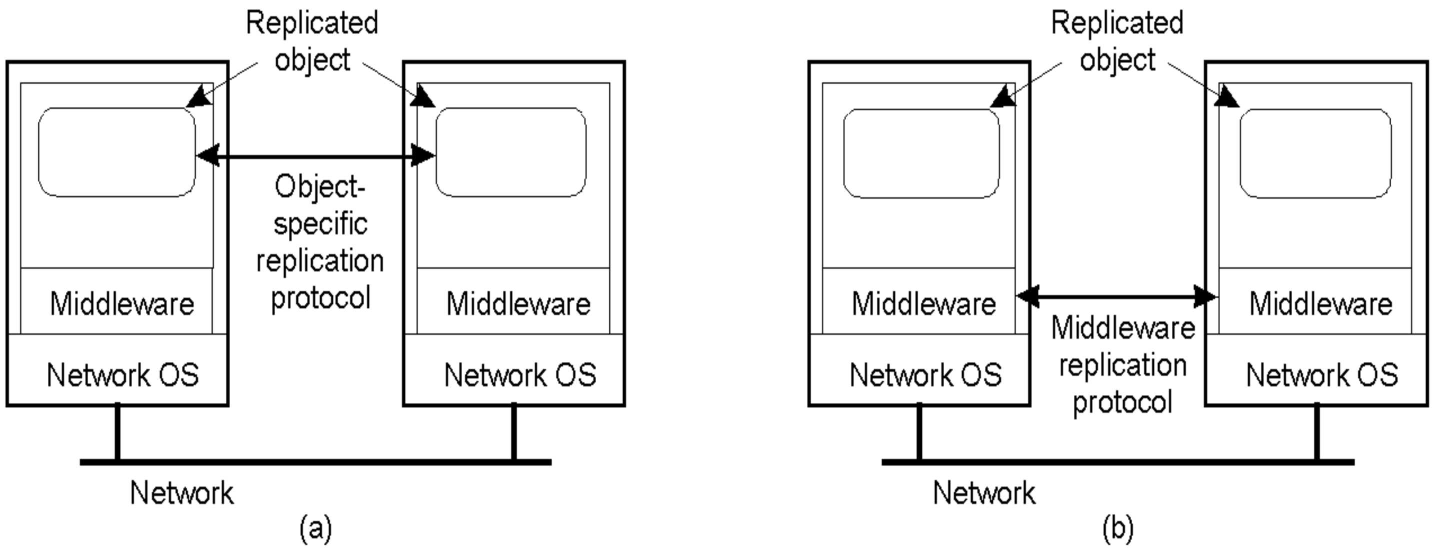


Why Replication?

- Benefits of replication:
  - ?
- Key issue: ?

Object Replication

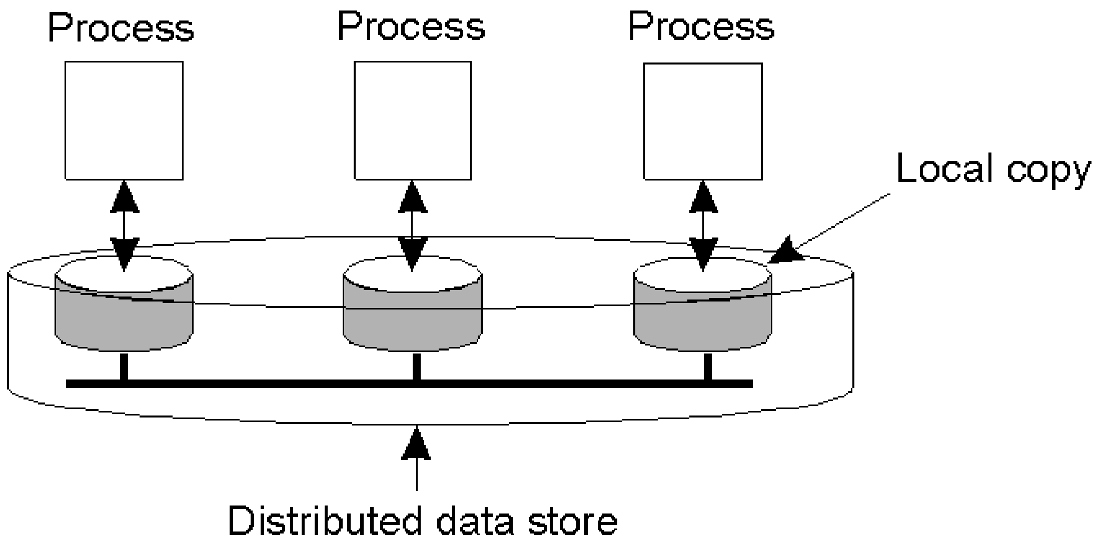
- Approach 1: application is responsible for replication
  - Application needs to handle consistency issues
  - Pro: ?
- Approach 2: system (middleware) handles replication
  - Consistency issues are handled by the middleware
  - Pro: ?



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!

Data-Centric Consistency Models



- Consistency model (aka *consistency semantics*)
  - Contract between processes and the data store
    - If processes obey certain rules, data store will work correctly
  - All models attempt to return the results of the last write for a read operation
    - Differ in how “last” write is determined/defined

Strict Consistency

- Any read always returns the result of the most recent write. It implicitly assumes
  - ?

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

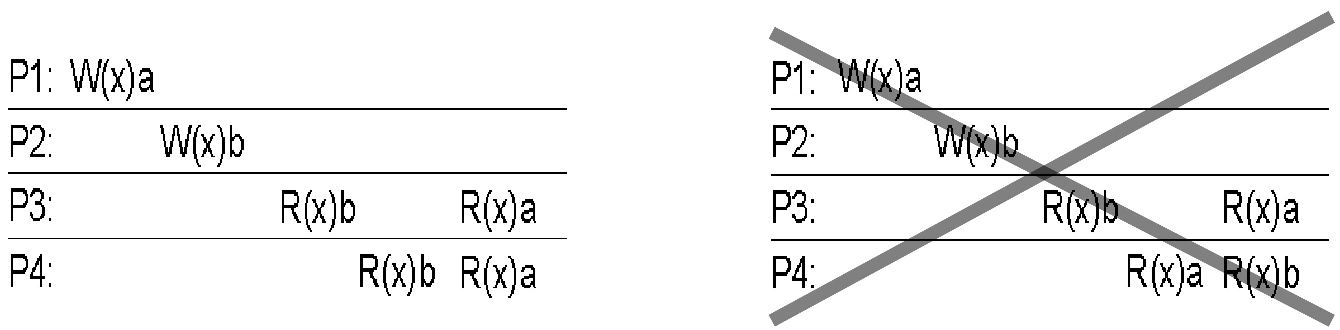
(a)

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

(b)

Sequential Consistency

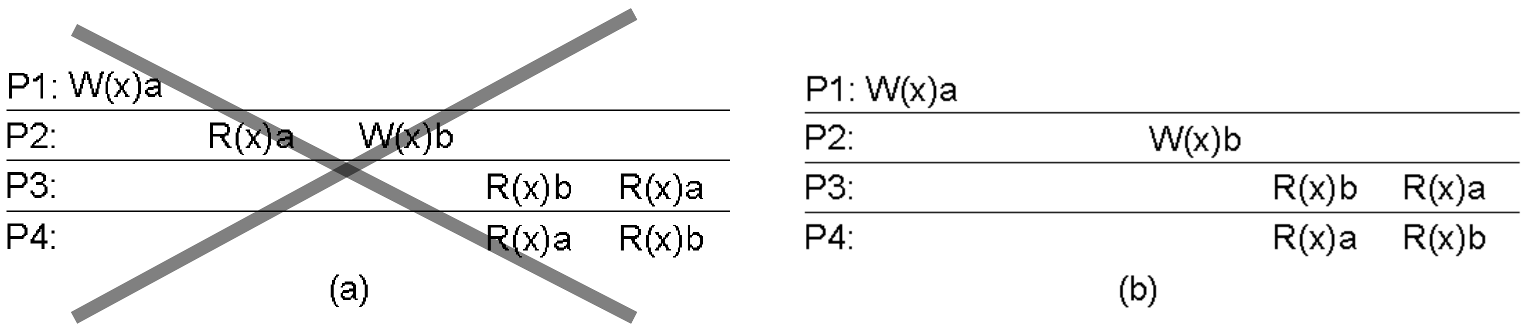
- Sequential consistency: weaker than strict consistency
  - Assumes all operations are executed in some sequential order and each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each proc preserves its program order
    - Nothing is said about “most recent write”



(a) (b)

Causal consistency

- Causally related writes must be seen by all processes in the same order
- Concurrent writes may be seen in different orders on different processes



Data-centric 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

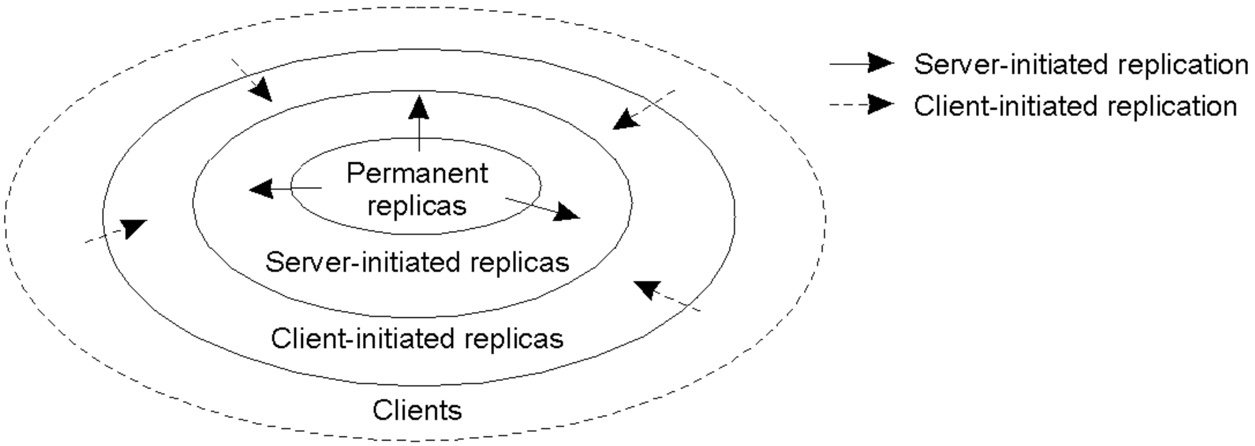
(a)

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)

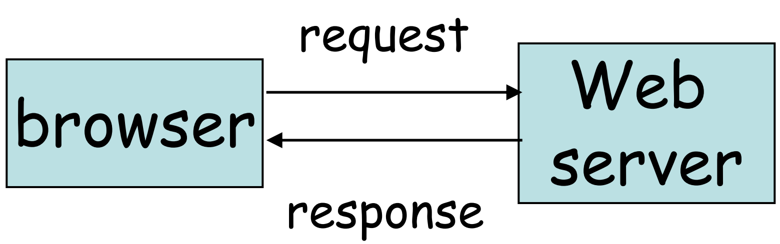
Replica Placement

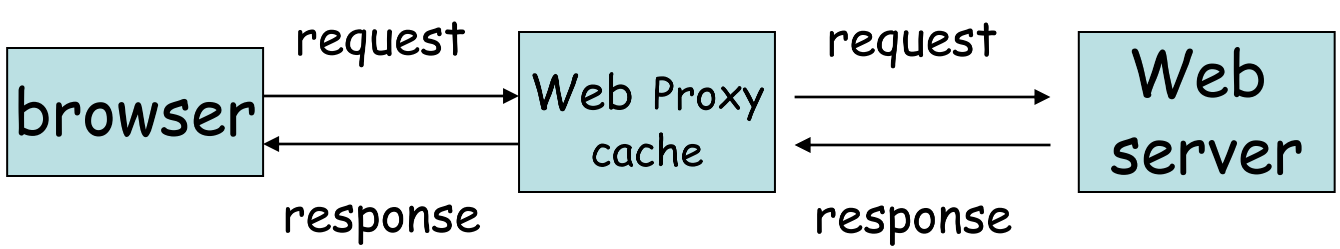
- Permanent replicas (mirroring)
- Server-initiated replicas (push caching)
- Client-initiated replicas (pull/client caching)



Web Caching

- Example of the web to illustrate caching and replication issues
  – Simpler model: clients are read-only, only server updates data



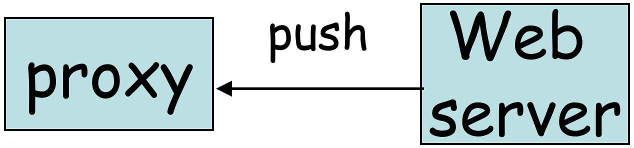


Consistency Issues

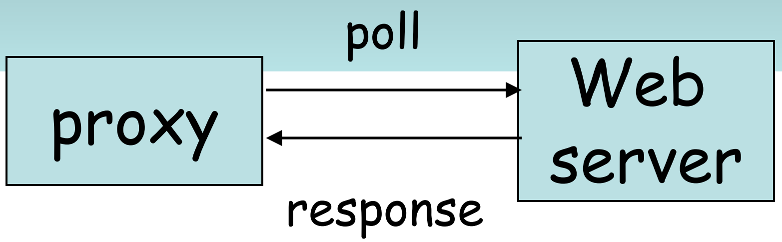
- Web pages tend to be updated over time
  - Some objects are static, others are dynamic
  - Different update frequencies (few minutes to few weeks)
- How can a proxy cache maintain consistency of cached data?
  - Send invalidate or update
  - Push versus pull

Push-based Approach

- Server tracks all proxies that have requested objects
- If a web page is modified, notify each proxy
- Notification types
  - (1) invalidate
  - (2) update
- How to decide between invalidate and updates?
  - Pros and cons?
  - Alternative approach: ?



Pull-based Approaches



- Proxy is entirely responsible for maintaining consistency
- Proxy periodically polls the server to see if object has changed
  – Use if-modified-since HTTP messages
- Key question: when should a proxy poll?

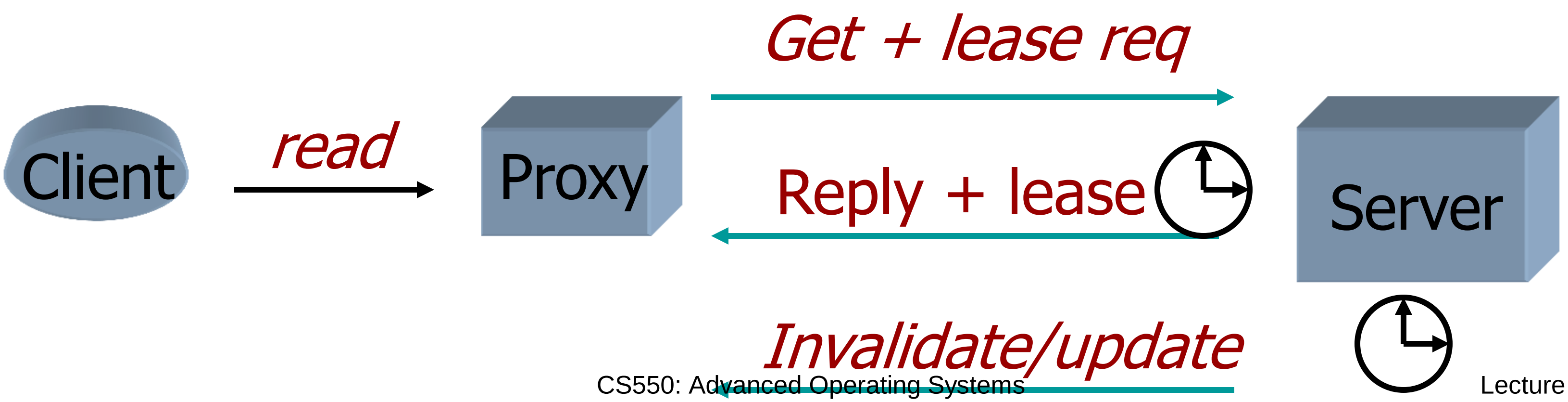
Pull-based Approach

- Method 1: use a server assigned time-to-live (TTL) values

- Method 2: proxy can dynamically determine the refresh interval

A Hybrid Approach: Leases

- Lease: duration of time for which server agrees to notify proxy of modification
- Server issues lease on first request and sends notification until expiry
  - Need to renew lease upon expiry
- Efficiency depends on the *lease duration*
  - Zero duration => ?
  - Infinite leases => ?



Questions

?