

## DISTRIBUTED SYSTEMS (COMP9243)

### Lecture 3a: Replication & Consistency

Slide 1

- ① Replication
- ② Consistency
  - Models
  - Protocols
- ③ Update propagation
- ④ Replica placement

## REPLICATION

Make copies of services on multiple machines.

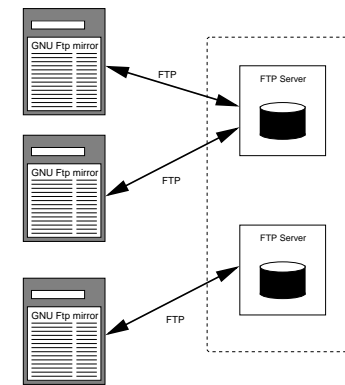
Why?:

- Slide 2
- Reliability
    - Redundancy
  - Performance
    - Increase processing capacity
    - Reduce communication
  - Scalability (prevent centralisation)
    - Prevent overloading of single server (*size scalability*)
    - Avoid communication latencies (*geographic scalability*)

## DATA VS CONTROL REPLICATION

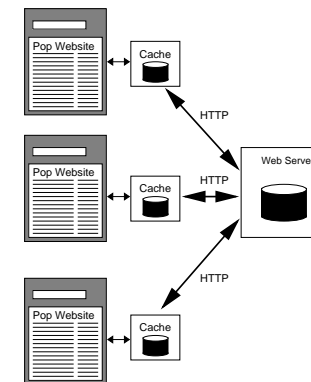
Data Replication (Server Replication/Mirroring):

Slide 3



Data Replication (Caching):

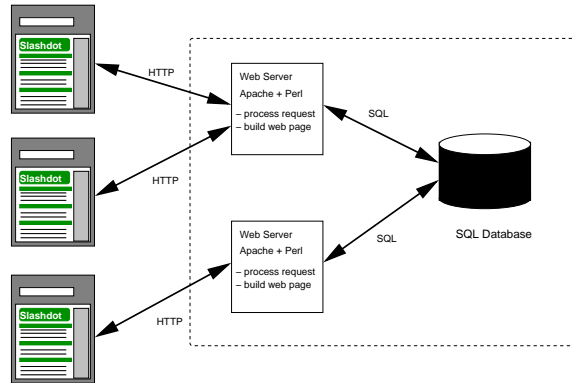
Slide 4



What's the difference between mirroring and caching?

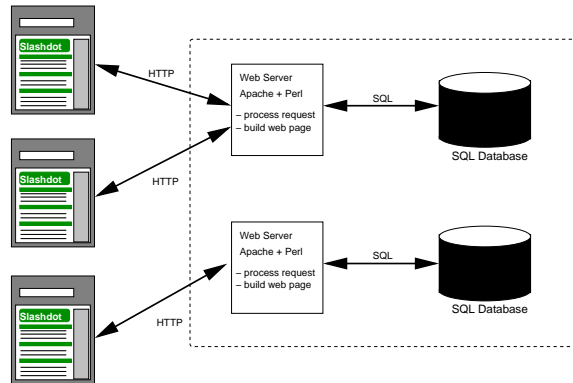
Control Replication:

Slide 5



Data and Control Replication:

Slide 6



Will be looking primarily at data replication (including combined data and control replication).

REPLICATION ISSUES

Updates

- Consistency (how to deal with updated data)
- Update propagation

Slide 7

Replica placement

- How many replicas?
- Where to put them?

Redirection/Routing

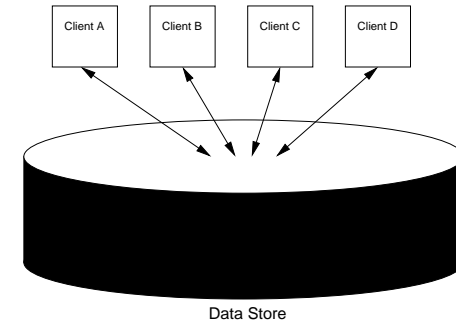
- Which replica should clients use?

DISTRIBUTED DATA STORE

→ data-store stores data items

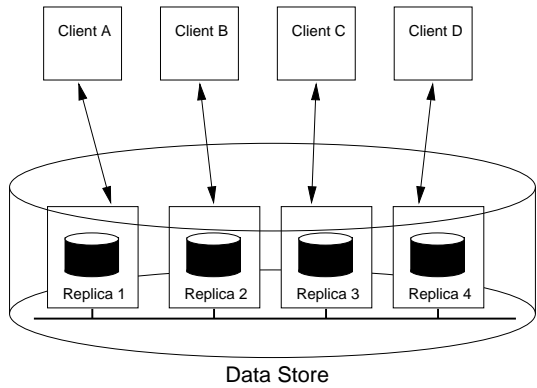
Client's Point of View:

Slide 8



Slide 9

Distributed Data-Store's Point of View:



Data Model:

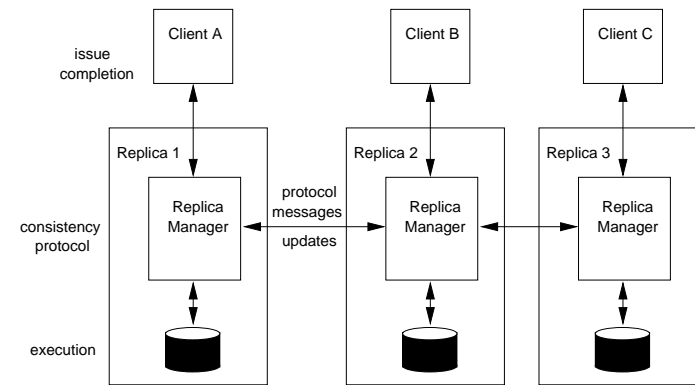
- data store: collection of data items
- data item: simple variable
- data item values: explicit (0, 1), abstract (a, b)

Operations on a Data Store:

- Read.  $R_i(x)$  Client  $i$  performs a read for data item  $x$  and it returns  $b$
- Write.  $W_i(x)a$  Client  $i$  performs write on data item  $x$  setting it to  $a$
- Operations not instantaneous
  - Time of issue (when request is sent by client)
  - Time of execution (when request is executed at a replica)
  - Time of completion (when reply is received by client)
- Coordination among replicas

Slide 10

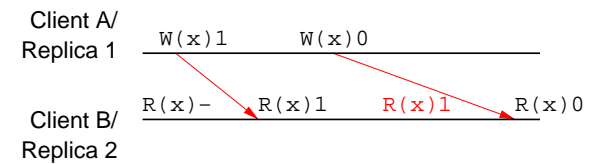
Replica Managers:



Slide 11

Timeline:

Slide 12





Slide 17

Data Coherence vs Data Consistency:

- Data Coherence** ordering of operations for single data item
  - e.g. a read of x will return the most recently written value of x
- Data Consistency** ordering of operations for whole data store
  - implies data coherence
  - includes ordering of operations on other data items too

DATA-CENTRIC CONSISTENCY MODEL

A contract, between a distributed data store and clients, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.

Slide 18

- Described consistency is experienced by all clients
- Multiple clients accessing the same data store
- Client A, Client B, Client C see same kinds of orderings
- Non-mobile clients (replica used doesn't change)

STRONG ORDERING VS WEAK ORDERING

Strong Ordering (tight):

- All writes must be performed in the order that they are invoked
- Example: all clients must see:  $W(x)_a W(x)_b W(x)_c$
- Strict (Linearisable) Sequential, Causal, FIFO (PRAM)

Slide 19

Weak Ordering (loose):

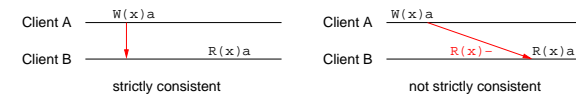
- Ordering of *groups* of writes, rather than individual writes
- Series of writes are grouped on a single replica
- Only results of grouped writes propagated.
- Example:  $\{W(x)_a W(x)_b W(x)_c\} == \{W(x)_b W(x)_a W(x)_c\}$
- Weak, Release, Entry

STRICT CONSISTENCY

Any read on a data item x returns a value corresponding to the result of the most recent write on x

Absolute time ordering of all shared accesses

Slide 20



What is *most recent* in a distributed system?

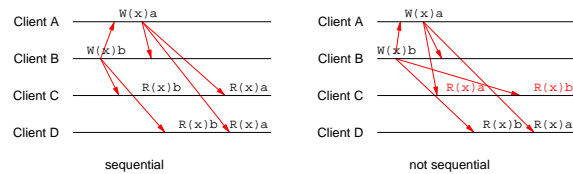
- Assumes an absolute global time
- Assumes instant communication (atomic operation)
- Normal on a uniprocessor
- ✗ Impossible in a distributed system

## SEQUENTIAL CONSISTENCY

All operations are performed in some sequential order

- More than one correct sequential order
- All clients see the same order
- Program order of each client maintained
- Not ordered according to time

Slide 21



Performance:

read time + write time  $\geq$  minimal packet transfer time

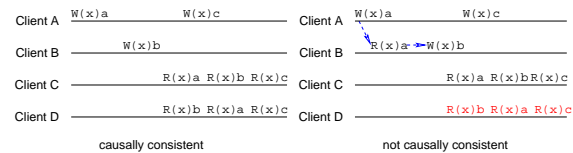
## CAUSAL CONSISTENCY

Potentially causally related writes are executed in the same order everywhere

Causally Related Operations:

Slide 22

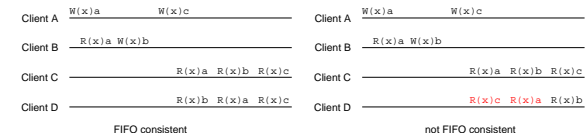
- Read followed by a write (in same client)
- $W(x)$  followed by  $R(x)$  (in same or different clients)



## FIFO (PRAM) CONSISTENCY

Only partial orderings of writes maintained

Slide 23



## WEAK CONSISTENCY

Shared data can be counted on to be consistent only after a synchronisation is done

Enforces consistency on a group of operations, rather than single operations

Slide 24

- Synchronisation variable (s)
- Synchronise operation ( $synchronise(S)$ )
- Define 'critical section' with synchronise operations

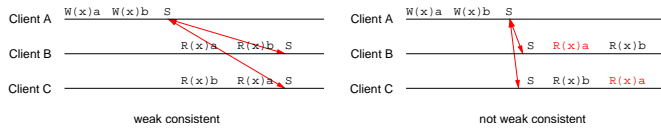
Properties:

- Order of synchronise operations sequentially consistent
- Synchronise operation cannot be performed until all previous writes have completed everywhere
- Read or Write operations cannot be performed until all previous synchronise operations have completed

Slide 25

Example:

- synchronise(S) W(x)a W(y)b W(x)c synchronise(S)
- Writes performed locally
- Updates propagated only upon synchronisation
- Only W(y)b and W(x)c have to be propagated



RELEASE CONSISTENCY

Explicit separation of synchronisation tasks

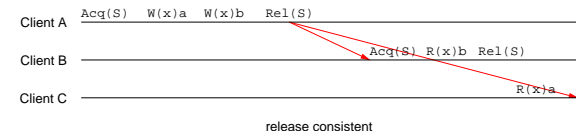
- acquire(S) - bring local state up to date
- release(S) - propagate all local updates
- acquire-release pair defines 'critical region'

Slide 26

Properties:

- Order of synchronisation operations are FIFO consistent
- Release cannot be performed until all previous reads and writes done by the client have completed
- Read or Write operations cannot be performed until all previous acquires done by the client have completed

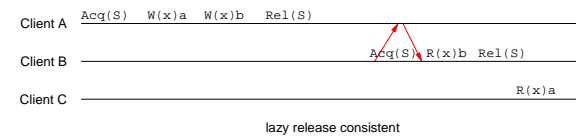
Slide 27



Lazy Release Consistency:

- Don't send updates on release
- Acquire causes client to get newest state
- Added efficiency if acquire-release performed by same client (e.g., in a loop)

Slide 28



## ENTRY CONSISTENCY

Synchronisation variable associated with specific shared data item (guarded data item)

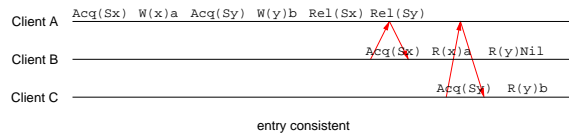
- Each shared data item has own synchronisation variable
- `acquire()`
  - Provides ownership of synchronisation variable
  - Exclusive and nonexclusive access modes
  - Synchronises data
  - Requires communication with current owner
- `release()`
  - Relinquishes exclusive access (but not ownership)

Slide 29

### Properties:

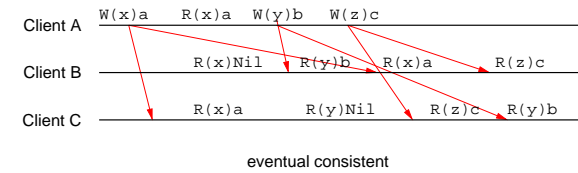
- Acquire does not complete until all guarded data is brought up to date locally
- If a client has exclusive access to a synchronisation variable, no other client can have any kind of access to it
- When acquiring nonexclusive access, a client must first get the updated values from the synchronisation variable's current owner

Slide 30



## EVENTUAL CONSISTENCY

If no updates take place for a long time, all replicas will gradually become consistent



Slide 31

### Requirements:

- Few read-write conflicts ( $R \gg W$ )
- Few write-write conflicts
- Clients accept inconsistency (i.e., old data)

### Examples:

- DNS:
  - no write-write conflicts
  - updates slowly (1-2 days) propagate to all caches
- WWW:
  - few write-write conflicts
  - mirrors eventually updated
  - cached copies (browser or proxy) eventually replaced

Slide 32

## CAP THEORY

C: Consistency: Linearisability  
A: Availability: Timely response  
P: Partition-Tolerance: Functions in the face of a partition



Slide 33

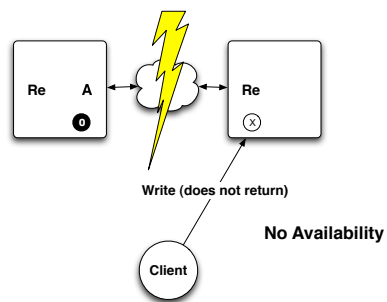
## CAP CONSEQUENCES

For wide-area systems:

- must choose: Consistency or Availability
- choosing Availability
  - Eventual consistency
- choosing Consistency
  - delayed (and potentially failing) operations

Slide 35

CAP Impossibility Proof:



Slide 34

## CLIENT-CENTRIC CONSISTENCY MODELS

*Provides guarantees about ordering of operations for a single client*

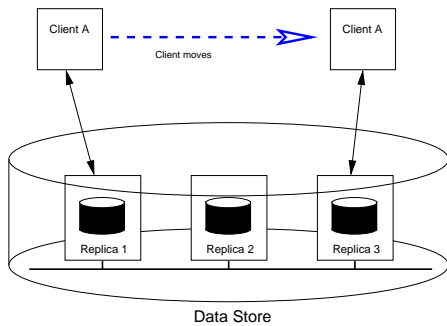
- Single client accessing data store
- Client accesses different replicas (modified data store model)
- Data isn't shared by clients
- Client A, Client B, Client C may see different kinds of orderings

In other words:

- The effect of an operation depends on the client performing it
- Effect also depends on the history of operations that client has performed.

Slide 36

Data-Store Model for Client-Centric Consistency:



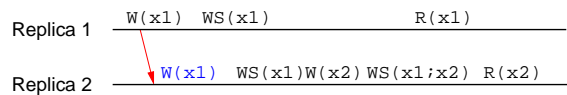
Slide 37

- Data-items have an owner
- No write-write conflicts

Notation and Timeline for Client-Centric Consistency:

- $x_i[t]$ : version of  $x$  at replica  $i$  at time  $t$
- Write Set:  $WS(x_i[t])$ : set of writes at replica  $i$  that led to  $x_i(t)$
- $WS(x_i[t_1]; x_j[t_2])$ :  $WS(x_j(t_2))$  contains same operations as  $WS(x_i(t_1))$
- $WS(!x_i[t_1]; x_j[t_2])$ :  $WS(x_j(t_2))$  does not contain the same operations as  $WS(x_i(t_1))$
- $R(x_i[t])$ : a read of  $x$  returns  $x_i(t)$

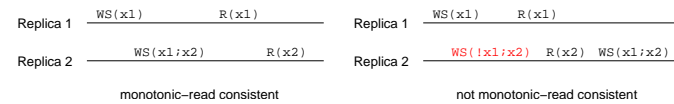
Slide 38



MONOTONIC READS

*If a client has seen a value of  $x$  at a time  $t$ , it will never see an older version of  $x$  at a later time*

Slide 39



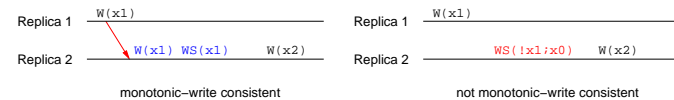
When is Monotonic Reads sufficient?

MONOTONIC WRITES

*A write operation on data item  $x$  is completed before any successive write on  $x$  by the same client*

All writes by a single client are sequentially ordered.

Slide 40



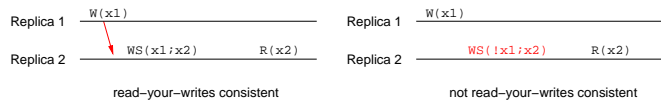
How is this different from FIFO consistency?

- Only applies to write operations of single client.
- Writes from clients not requiring monotonic writes may appear in different orders.

## READ YOUR WRITES

The effect of a write on  $x$  will always be seen by a successive read of  $x$  by the same client

Slide 41



## CHOOSING THE RIGHT MODEL

### Trade-offs

#### Consistency and Redundancy:

- All copies must be strongly consistent
- All copies must contain full state
- Reduced consistency → reduced reliability

Slide 43

#### Consistency and Performance:

- Consistency requires extra work
- Consistency requires extra communication
- ✗ Can result in loss of overall performance

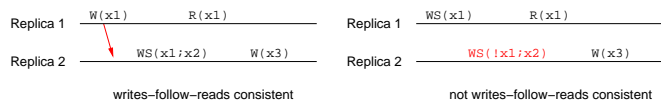
#### Consistency and Scalability:

- Implementation of consistency must be scalable
  - don't take a centralised approach
  - avoid too much extra communication

## WRITE FOLLOWS READS

A write operation on  $x$  will be performed on a copy of  $x$  that is up to date with the value most recently read by the same client

Slide 42



## CONSISTENCY PROTOCOLS

Consistency Protocol: implementation of a consistency model

Slide 44

#### Primary-Based Protocols:

- Remote-write protocols
- Local-write protocols

#### Replicated-Write Protocols:

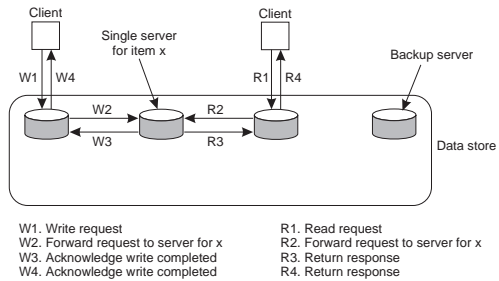
- Active Replication
- Quorum-Based Protocols

## REMOTE-WRITE PROTOCOLS

### Single Server:

- All writes and reads executed at single server
- No replication of data

Slide 45

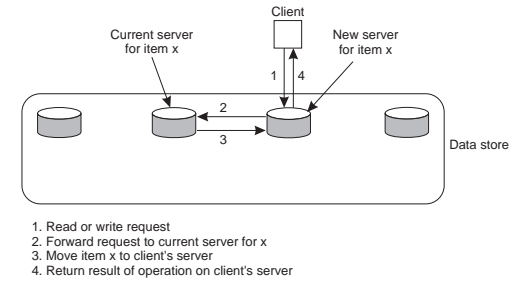


## LOCAL-WRITE PROTOCOLS

### Migration:

- Data item migrated to local server on access
- Distributed, non-replicated, data store

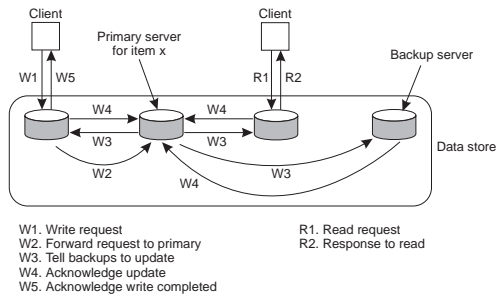
Slide 47



### Primary-Backup:

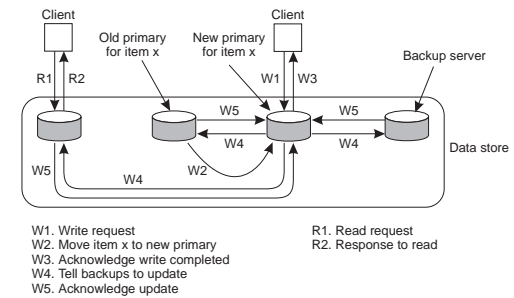
- All writes executed at single server, Reads are local
- Updates block until executed on all backups
- ✗ Performance

Slide 46



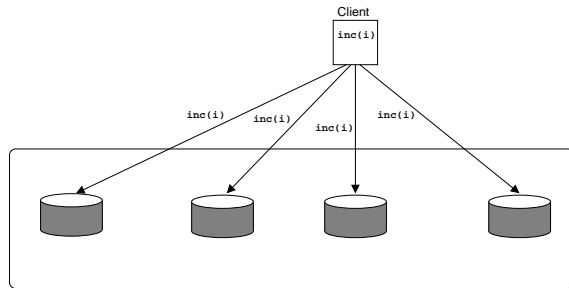
### Migrating Primary (multiple reader/single writer):

Slide 48



## ACTIVE REPLICATION

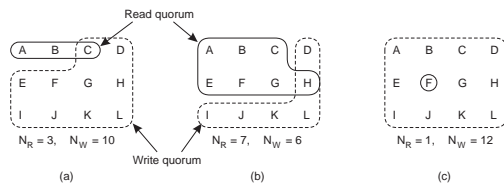
- Updates (write operation) sent to all replicas
- Need totally-ordered multicast
- e.g. sequencer/coordinator to add sequence numbers



Slide 49

## QUORUM-BASED PROTOCOLS

- Voting
- Versioned data
- Read Quorum:  $N_r$
- Write Quorum:  $N_w$
- $N_r + N_w > N$  Why?
- $N_w > N/2$  Why?



Slide 50

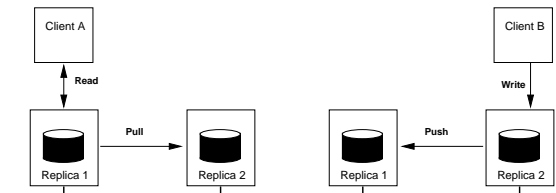
## UPDATE PROPAGATION

What to propagate?

- Data
  - R/W high
- Update operation
  - low bandwidth costs
- Notification/Invalidation
  - R/W low

Slide 51

## PUSH VS PULL



Slide 52

Pull:

- Updates propagated only on request
- Also called *client-based*
- R/W low
- Polling delay

Push:

- Push updates to replicas
- Also called *server-based*
- When low staleness required
- $R \gg W$
- ✗ Have to keep track of all replicas

Leases:

Server promises to push updates until lease expires

Lease length depends on:

**Slide 53** **age:** Last time item was modified

**renewal-frequency:** How often replica needs to be updated

**state-space overhead:** lower expiration time to reduce bookkeeping when many clients

Permanent Replicas:

- Initial set of replicas
- Created and maintained by data-store owner(s)
- Allow writes

Server-Initiated Replicas:

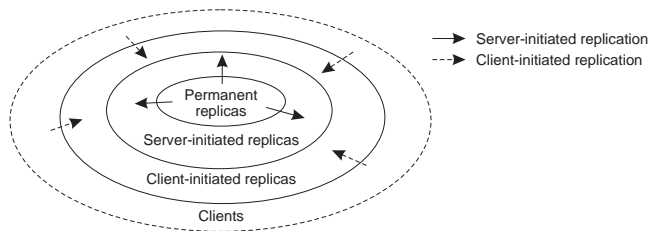
- Enhance performance
- Not maintained by owner
- Placed close to groups of clients
  - Manually
  - Dynamically

**Slide 55**

Client-Initiated Replicas:

- Client caches
- Temporary
- Owner not aware of replica
- Placed close to client
- Maintained by host (often client)

## REPLICA PLACEMENT



**Slide 54**

## DYNAMIC REPLICATION

Situation changes over time

- Number of users, Amount of data
- Flash crowds
- R/W ratio

**Slide 56**

Dynamic Replica Placement:

- Network of replica servers
- Keep track of data item requests at each replica
- Deletion threshold
- Replication threshold
- Migration threshold
- Clients always send requests to nearest server

---

## MISCELLANEOUS IMPLEMENTATION AND DESIGN ISSUES

End-to-End argument:

- Where to implement replication mechanisms?
- Application? Middleware? OS?

Policy vs Mechanism:

- Consistency models built into middleware?
- One-size-fits-all?

Slide 57

Determining Policy:

- Who determines the consistency model used?
    - Application
    - Middleware
    - Client
    - Server
- 
- 

## READING LIST

Slide 58

**Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services** An overview of the CAP theorem and its proof.

**Eventual Consistency** An overview of eventual consistency and client-centric consistency models.

---