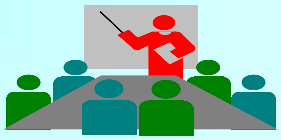
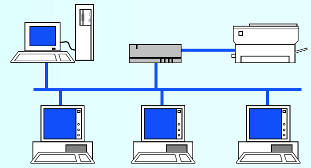




Distributed Systems
600.417
Replication

Department of Computer Science
The Johns Hopkins University

Replication

Lecture 7

Further readings:

- *Distributed Systems (second edition) Sape Mullender, chapters 7,8 (Addison-Wesley) 1994.*
- *Concurrency Control and Recovery in Distributed Database Systems Bernstein, Hadzilacos and Goodman (Addison Wesley) 1987.*
- *From Total Order to Database Replication ICDCS 2002* (www.dsn.jhu.edu)
- *Paxos Made Simple*, Leslie Lamport ACM Sigact News 2001
- *Paxos for System Builders: An Overview* LADIS 2008 (www.dsn.jhu.edu)
- Raft: In Search of an Understandable Consensus Algorithm USENIX 2014 https://raft.github.io/

Replication

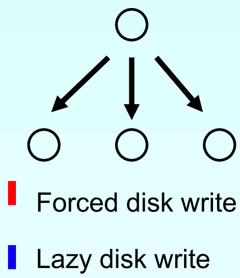
- Benefits of replication:
  - **High Availability.**
  - **High Performance.**
- Costs of replication:
  - **Synchronization.**
- Requirements from a generic solution:
  - Strict consistency – one copy serializability.
  - Sometimes too expensive so requirements are tailored to applications.

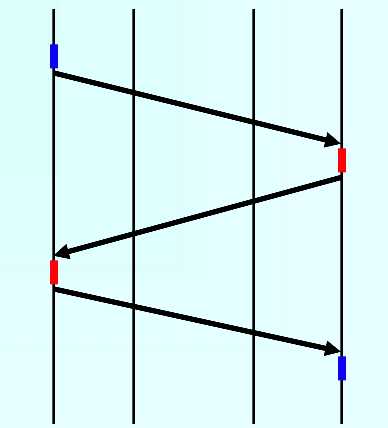
Replication Methods

- Two phase commit, three phase commit
- Primary and backups
- Weak consistency (weaker update semantics)
- Quorum (primary component) methods with state machine replication
  - Congruity: Virtual Synchrony based replication
  - Paxos: Leader based replication
  - Raft: Leader based replication with better understandability
- Analysis and summary

Two Phase Commit

Send decision





Two Phase Commit

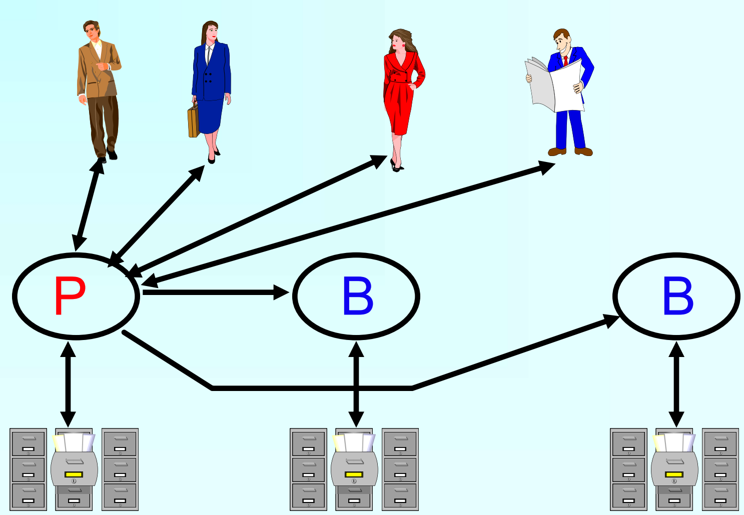
- Built for updating distributed databases
- Can be used for the special case of replication
- Consistent with a generic update model
- In contrast to the distributed transaction case, we do not need all replicas to agree (and hence to participate) in committing each update (each participant has the same state) – a quorum is sufficient, making this method not as good of a fit

Primary and Backups

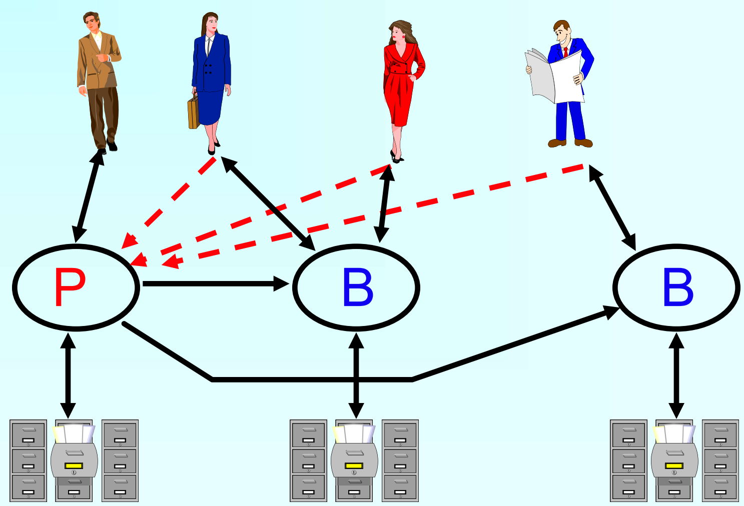
Possible options:

- Backups are maintained for availability only
- Backups can improve performance for reads, updates are sent to the primary by the user
  - **What is the query semantics? How can one copy serializability be achieved?**
- The user interacts with one copy, and if it is a backup, the updates are sent to the primary
  - **What is the query semantics with regards to our own updates?**

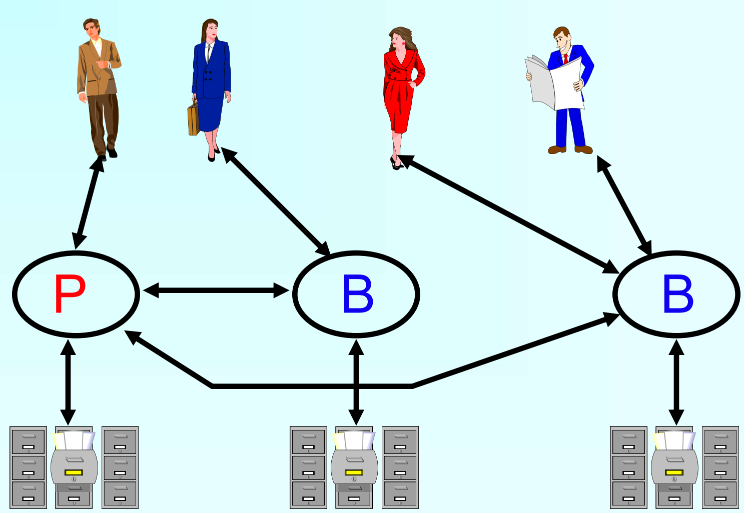
Primary and Backups (1)



Primary and Backups (2)



Primary and Backups (3)



Replication Methods

- Two phase commit, three phase commit
- Primary and backups
- Weak consistency (weaker update semantics)
- Quorum (primary component) methods with state machine replication
  - Congruity: Virtual Synchrony based replication.
  - Paxos: Leader based replication
  - Raft: Leader based replication with better understandability
- Analysis and summary

Weak Consistency (weaker update semantics)

The Anti-Entropy method

- State kept by the replication servers can be weakly consistent i.e. copies are allowed to diverge temporarily. They will eventually come to agreement assuming commutative update semantics (for applications where updates can be executed in any order to reach the same state)
- From time to time, a server picks another server and these two servers exchange updates and converge to the same state
- The same method can be used to support strong semantics if total order is obtained by getting one message from every server (e.g. by using Lamport time stamps to order messages) but that would not be live if the network partitions

The Anti-Entropy method

Knowledge at Server A

A	1	3	5	12
B	2			
C	2	3	4	

Knowledge at Server B

A	1	3			
B	2	5	6	9	11
C	2				

Summary A

12
2
4

Summary B

3
11
2

Numbers refer to Lamport time stamps.

The Anti-Entropy Method (cont.)

Knowledge at Server A

A	1	3	5	12
B	2			
C	2	3	4	

Knowledge at Server B

A	1	3			
B	2	5	6	9	11
C	2				

Summary A

12
2
4

Summary B

3
11
2

Summary After merge

12
11
4

The Anti-Entropy Method (cont.)

Knowledge at Server A

A	1	3	5	12	
B	2	5	6	9	11
C	2	3	4		

Knowledge at Server B

A	1	3	5	12	
B	2	5	6	9	11
C	2	3	4		

Summary A

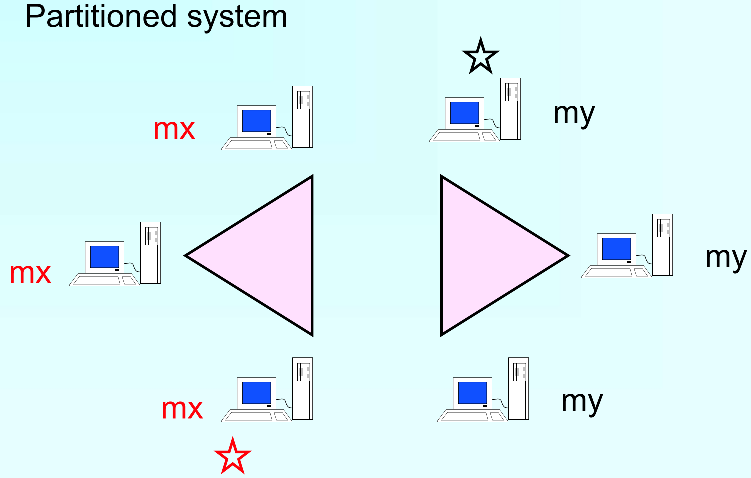
12
11
4

Summary B

12
11
4

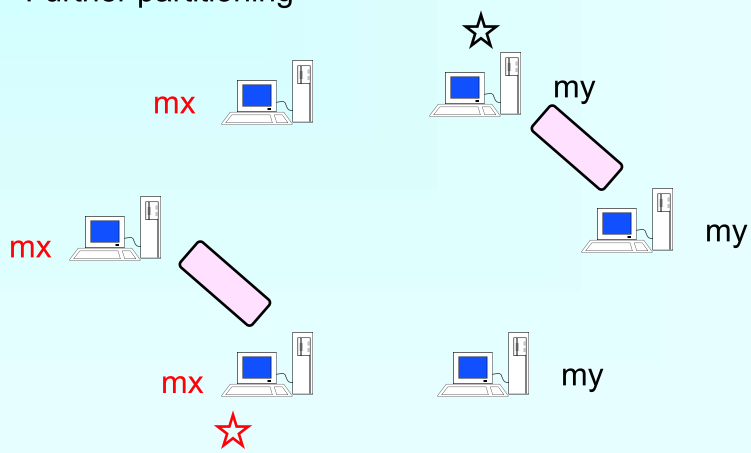
Eventual Path Propagation

Partitioned system



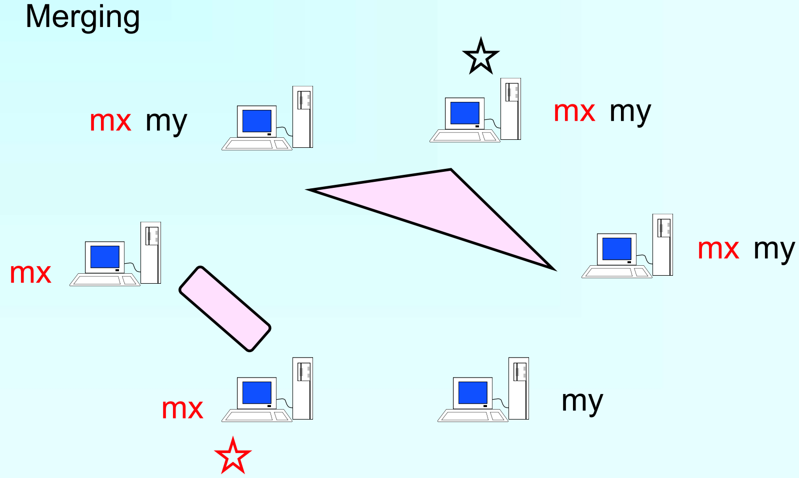
Eventual Path Propagation (cont.)

Further partitioning



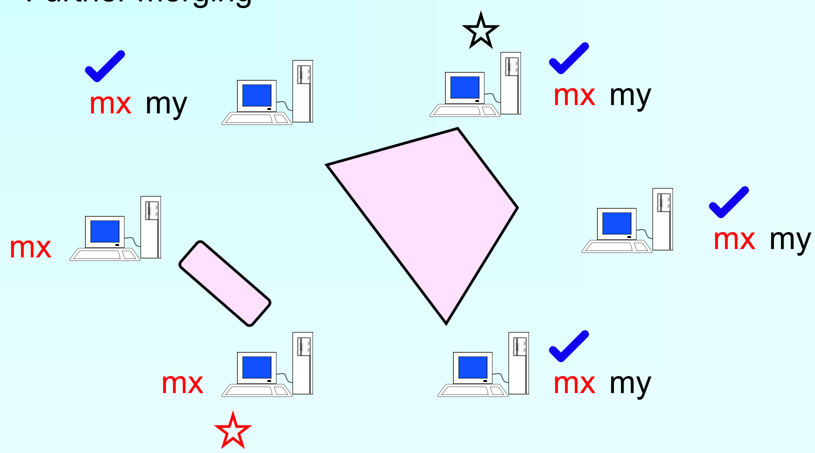
Eventual Path Propagation (cont.)

Merging



Eventual Path Propagation (cont.)

Further merging



Replication Methods

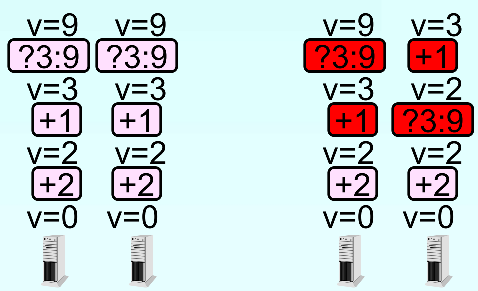
- Two phase commit, three phase commit
- Primary and backups
- Weak consistency (weaker update semantics)
- Quorum (primary component) methods with state machine replication
  - Congruity: Virtual Synchrony based replication.
  - Paxos: Leader based replication
  - Raft: Leader based replication with better understandability
- Analysis and summary

State Machine Replication

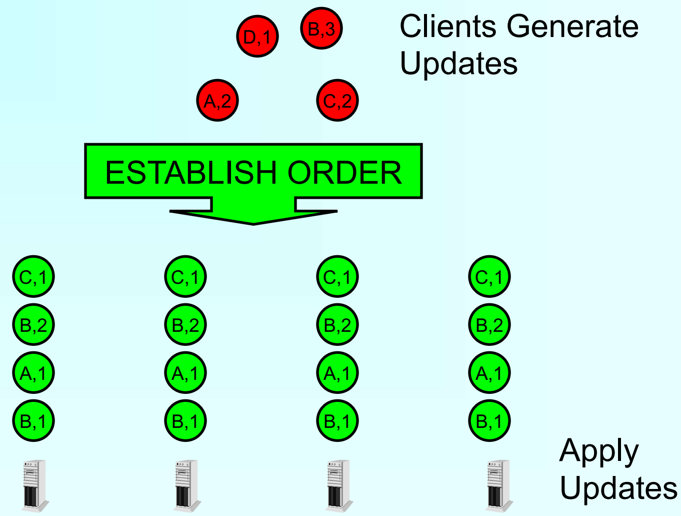
- Servers **start in the same state**
- Servers change their state only when they execute an update
- State changes are deterministic. **Two servers in the same state will move to identical states, if they execute the same update**
- If servers **execute updates in the same order**, they will progress through exactly the same states. **State Machine Replication!**

State Machine Replication Example

- Our State: one variable
- Operations (cause state changes)
  - Op 1) + n : Add n to our variable
  - Op 2) ?v:n : If variable = v, then set it to n
- Start: All servers have variable = 0
- If we apply the above operations in the same order, then the servers will remain consistent



State Machine Replication



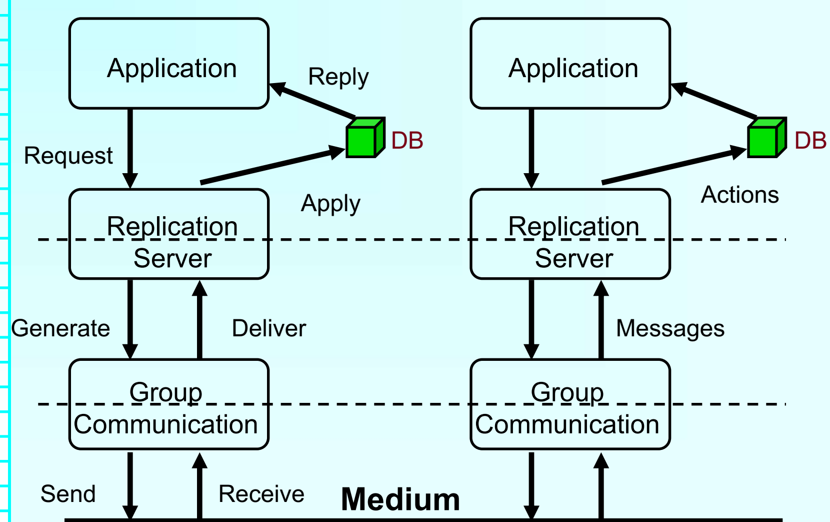
Quorum

- A quorum can proceed with updates.
  - **Remember that for distributed transactions, every DM had to agree**
  - **But in the more specific problem of replication, a quorum can continue (not all DM have to agree)**
- When the network connectivity changes, if there is a quorum, the members can continue with updates
- Dynamic methods will allow the next quorum to be formed based on the current quorum
  - **Dynamic Linear Voting: the next quorum is a majority of the current quorum**
  - **Useful to put a minimum cap on the size of a viable quorum to avoid relying on too few specific remaining replicas, which can lead to potential vulnerability**

Group Communication "Tools"

- Efficient message delivery
  - Group multicast
- Message delivery and ordering guarantees
  - Reliable delivery
  - FIFO and Causal orders
  - Agreed order
  - Safe delivery
- Partitionable Group Membership
- Strong semantics (what is actually needed?)

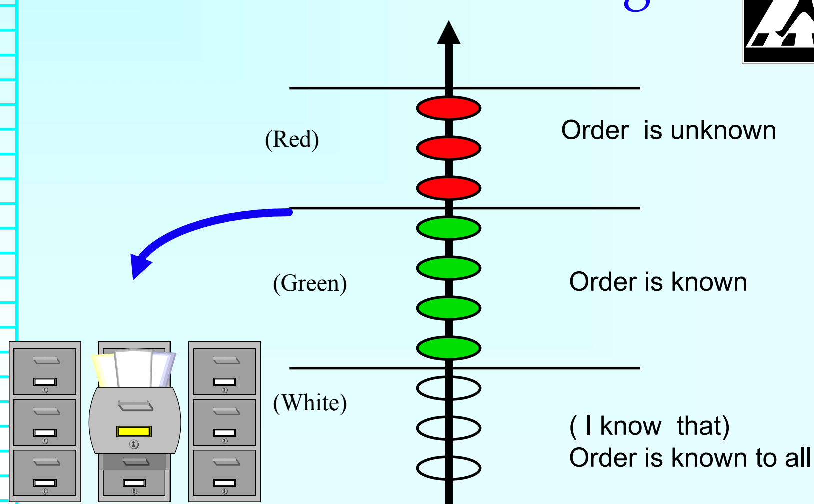
Congruity: Virtual-Synchrony based replication



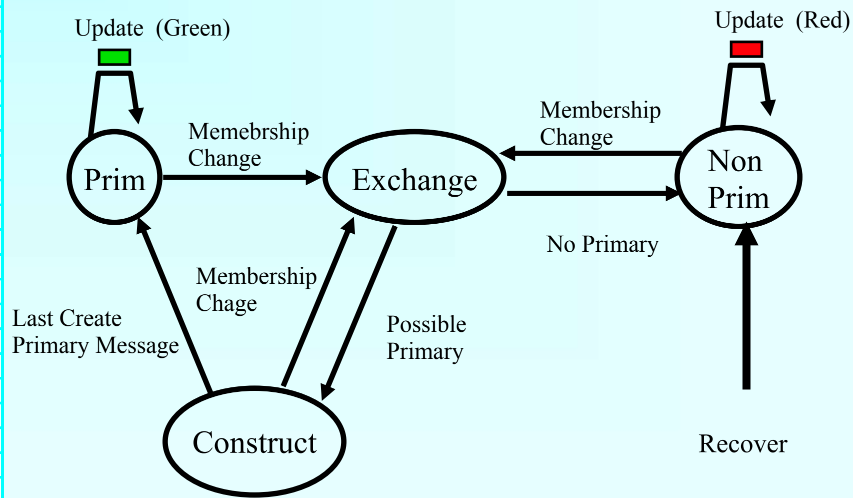
Congruity: The Basic Idea

- Reduce database replication to **G**lobal **C**onsistent **P**ersistent **O**rder
  – Use group communication ordering to establish the Global Consistent Persistent Order on the updates.
  – deterministic + serialized = consistent
- Group Communication membership + quorum = **primary** component.
  – Only replicas in the **primary** component can commit updates.
  – Updates ordered in a primary component are marked **green** and applied. Updates ordered in a non-primary component are marked red and will be delayed.

Action Ordering

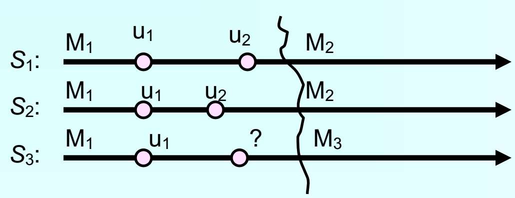


Congruity: Conceptual State Machine



Not so simple…

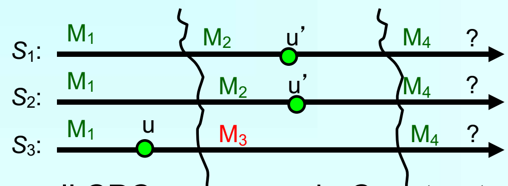
- **Virtual Synchrony**: If s_1 and s_2 move directly from membership M_1 to M_2, then they deliver the same ordered set of messages in M_1.
  - What about s_3 that was part of M_1 but is not part of M_2?



- Total (Agreed) Order **with no holes** is not guaranteed across partitions or server crashes/recoveries!

Delicate Points

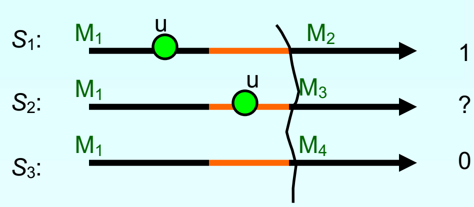
- s_3 receives update *u* in Prim and commits it right before a partition occurs, but s_1 and s_2 do not receive *u*. If s_1 and s_2 will form the next primary component, they will commit new updates, without knowledge of *u*!!



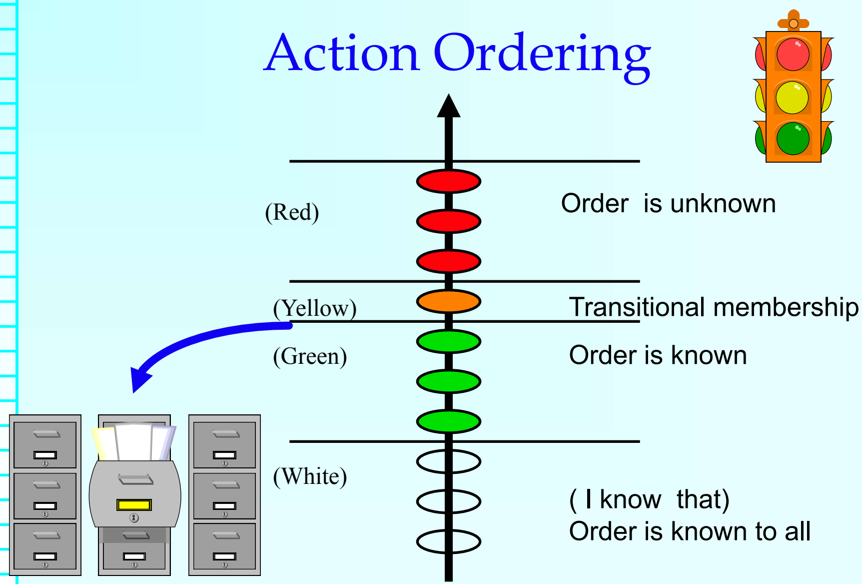
- s_1 receives all CPC messages in Construct, and moves to Prim, but one of the servers that were with s_1 in Construct does not receive the last CPC message. A new primary is created possibly without having the desired majority!!

Virtual Synchrony

- Regular and Transitional membership notifications
- Safe message = Agreed plus every server in the current membership will deliver the message unless it crashes.
- Safe delivery breaks the two-way uncertainty into 3 possible scenarios, the extremes being mutually exclusive!

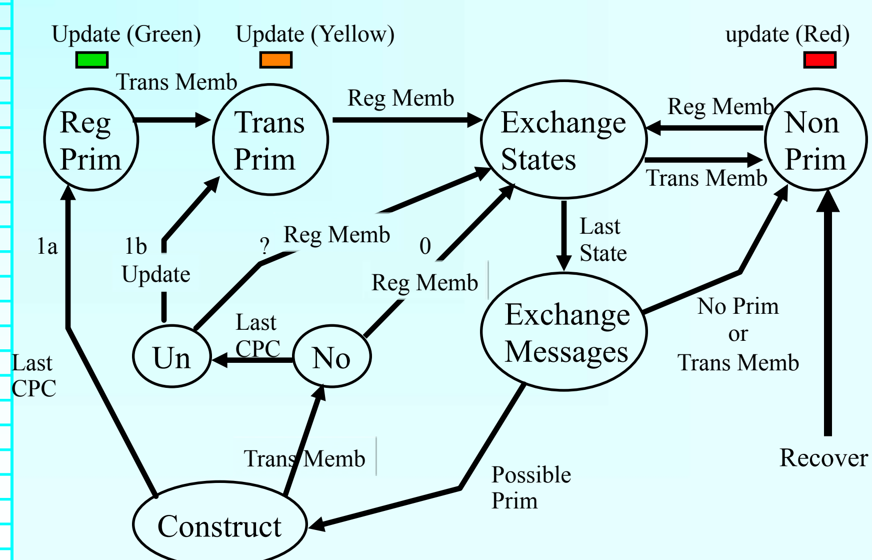


Action Ordering

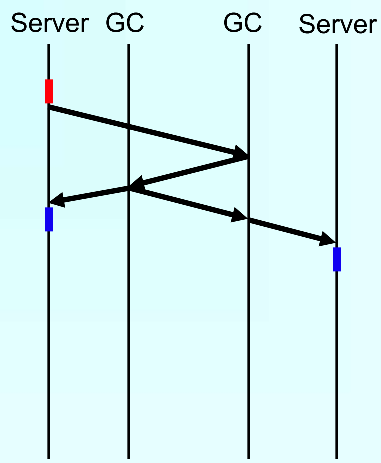


Congruity State Diagram

[ICDCS02]



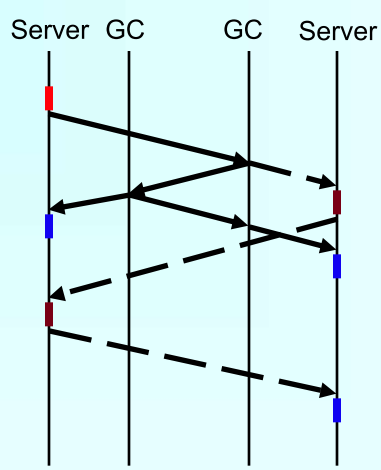
Latency Comparison



Forced disk write

Lazy disk write

Latency Comparison



Forced disk write

Lazy disk write

Congruity Recap

- Knowledge propagation
  - Eventual Path Propagation
- Amortizing end-to-end acknowledgments
  - Low level Ack derived from Safe Delivery of group communication
  - End-to-end Ack upon membership changes
- Primary component selection
  - Dynamic Linear Voting

Replication Methods

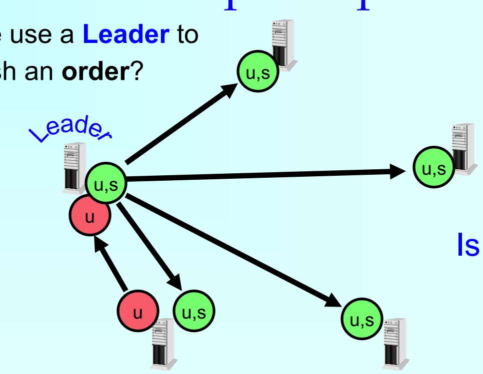
- Two phase commit, three phase commit
- Primary and backups
- Weak consistency (weaker update semantics)
- Quorum (primary component) methods with state machine replication
  - Congruity: Virtual Synchrony based replication
  - Paxos: Leader based replication
  - Raft: Leader based replication with better understandability
- Analysis and summary

What about Dynamic Networks?

- Group communication requires stable membership to work well
  - If membership is not stable, group communication based scheme will spend a lot of time synchronizing
- A more robust replication algorithm is needed for such environments – **Paxos**
  - Requires a stable-enough network to elect a leader that will stay stable for a while
  - Requires a (potentially changing) majority of members to support the leader (in order to make progress)

Simple Replication

Can we use a **Leader** to establish an **order**?



Is this resilient?

If leader fails, then the system is not live!

- **Server sends update, u, to Leader**
- **Leader assigns a sequence number, s, to u, and sends the update to the non-leader servers.**
- **Servers order update u with sequence number s.**

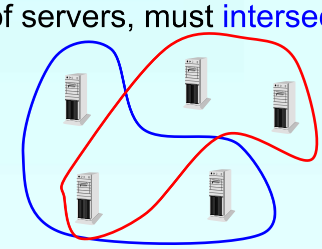
How can we improve resiliency?

Elect another leader.

Use more messages.

Assign a sequence number to each leader. (Views)

Use the fact that two sets, each having at least a majority of servers, must intersect!



First… We need to describe our system model and service properties.

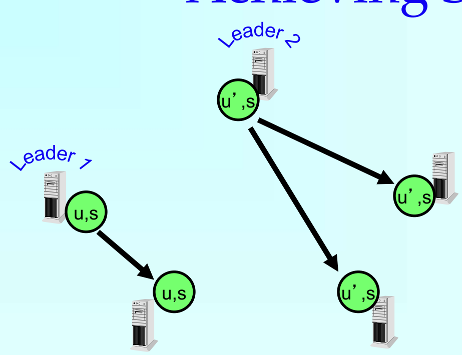
Paxos System Model

- N servers
  - Uniquely identified in {1…N}
- Asynchronous communication
  - Message loss, duplication, and delay
  - Network partitions
  - No message corruption
- Benign faults
  - Crash/recovery with stable storage
  - No Byzantine behavior

What is Safety?

- Safety: **If two servers execute the ith update, then these updates are the same – supporting state machine replication**
- Another way to look at safety:
  - If there exists an ordered update (u_i, s) at some server, then there cannot exist an ordered update (u'_i, s') at any other server, where $u_i \neq u'_i$
- We will now focus on achieving safety -- making sure that we don't execute updates in different orders on different servers.

Achieving Safety

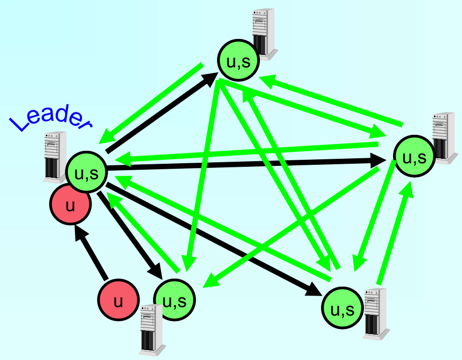


Is this safe?

A new leader can violate safety!
Can we fix this?

- A new leader must not violate previously established ordering!
- The new leader must know about all updates that may have been ordered.

Achieving Safety



What does this give us?

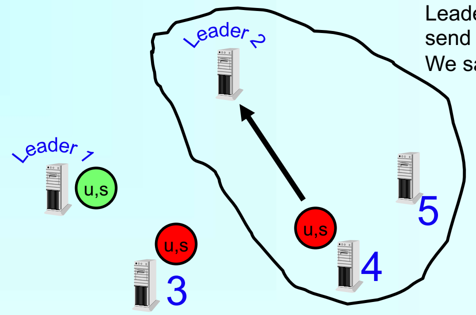
If a new leader gets information from **any** majority of servers, it can determine what **may** have been ordered!

- Leader sends Proposal(u,s) to all servers
- All servers send Accept(u,s) to all servers.
- Servers order (u,s) when they receive a majority of Proposal/Accept(u,s) messages

Changing Leaders

- Changing Leaders is commonly called a **View Change.**
- Servers use timeouts to detect failures.
- If the current leader fails, the servers **elect** a new leader.
- The new leader cannot propose updates until it collects information from a majority of servers!
  - Each server reports any Proposals that it knows about.
  - If any server ordered a Proposal(u,s), then at least one server in any majority will report a Proposal for that sequence number!
  - The new server will never violate prior ordering!!
  - **Now we have a safe protocol!!**

Changing Leaders Example



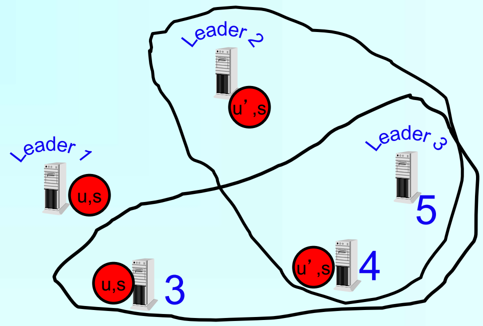
Leader 2 can send a Proposal(u,s). We say it **replays** (u,s)

- If any server orders (u,s), then at least majority of servers must have received Proposal(u,s).
- If a new server is elected leader, it will gather Proposals from a majority of servers.
- **The new leader will learn about the ordered update!!**

Is Our Protocol Live?

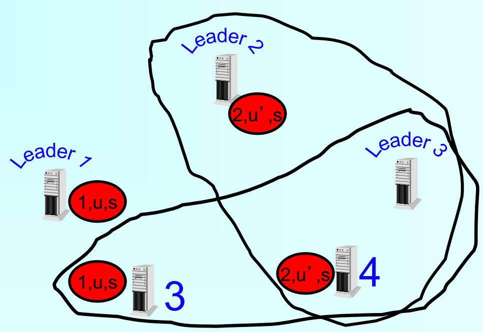
- Liveness: If there is a set, Q , consisting of majority of connected servers (quorum), then if any server in set Q has a new update, then this update will eventually be executed.
- Is there a problem with our protocol? It is safe, but is it live?

Liveness Example



- Leader 3 gets conflicting Proposal messages!
- **Which one should it choose?**
- **What should we add??**

Adding View Numbers



- We add view numbers to the Proposal(v,u,s)!
- Leader 3 gets conflicting Proposal messages!
- Which one should it choose?
- It chooses the one with the greatest view number!!

Normal Case

Assign-Sequence()

A1. u := NextUpdate()

A2. next_seq++

A3. SEND: Proposal(view, u,next_seq)

Upon receiving Proposal(v, u,s):

B1. if not leader and v == my_view

B2. SEND: Accept(v,u,s)

Upon receiving Proposal(v,u,s) and majority - 1 Accept(v,u,s):

C1. ORDER (u,s)

We use **view numbers** to determine which Proposal may have been ordered previously.

A server sends an Accept(v,u,s) message only for a view that it is currently in, and never for a lower view!

Leader Election

Elect Leader()

Upon Timer T Expire:

A1. my_view++

A2. SEND: New-Leader(my_view)

Upon receiving New-Leader(v):

B1. if Timer T expired

B2. if v > my_view, then my_view = v

B3. SEND: New-Leader(my_view)

Upon receiving majority New-Leader(v) where v == my_view:

C1. timeout *= 2; Timer T = timeout

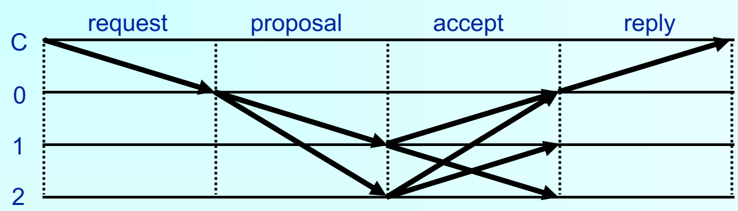
C2. Start Timer T

Let V_max be the highest view that any server has. Then, at least a majority of servers are in view V_max or V_max - 1.

Servers will stay in the maximum view for at least one full timeout period.

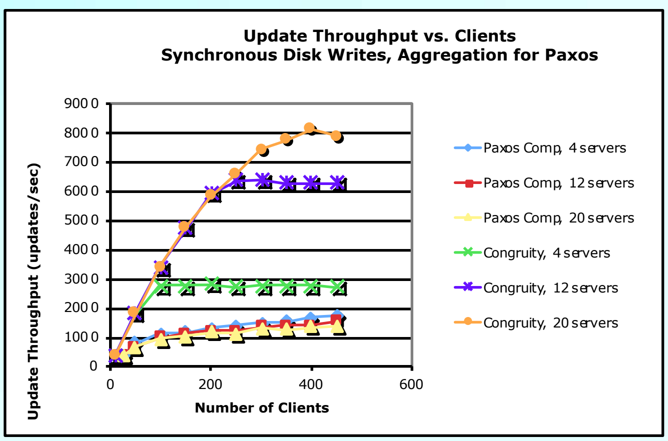
A server that becomes disconnected/connected repeatedly cannot disrupt the other servers.

We Have: Paxos



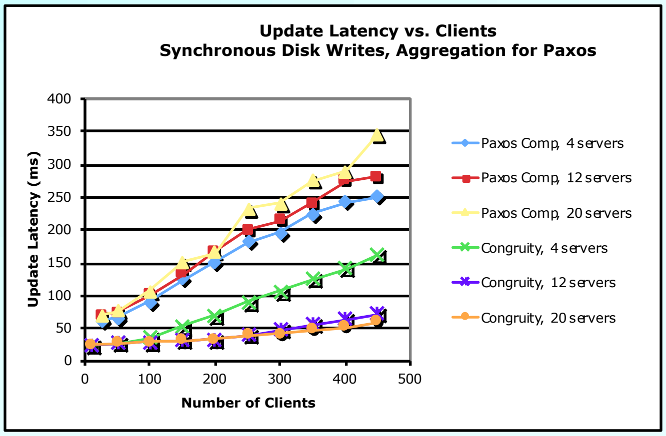
- The Part-Time Parliament [Lamport, 98]
- A very resilient protocol. Only a majority of participants are required to make progress.
- Works well on unstable networks.
- Note: Paxos is complex to understand, so I explained a variant based on Paxos for System Builders – Paxos-SB [KA 2008]

Performance Results (Paxos-SB)

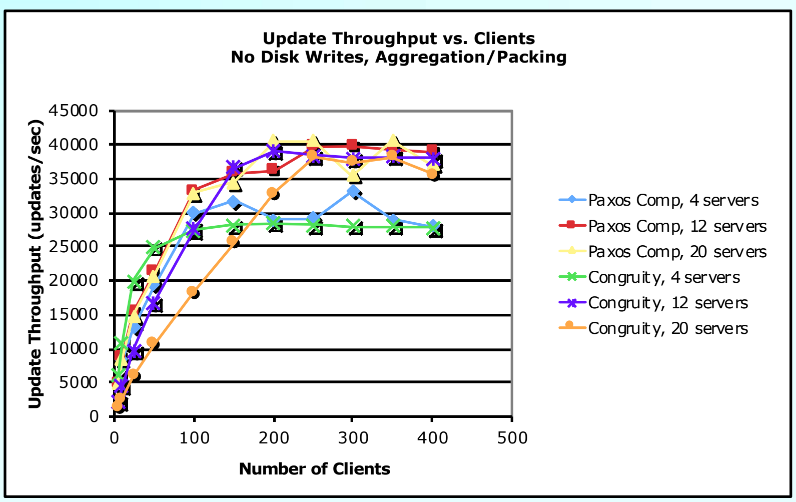


Local area network cluster.
Congruity: group communication-based replication.

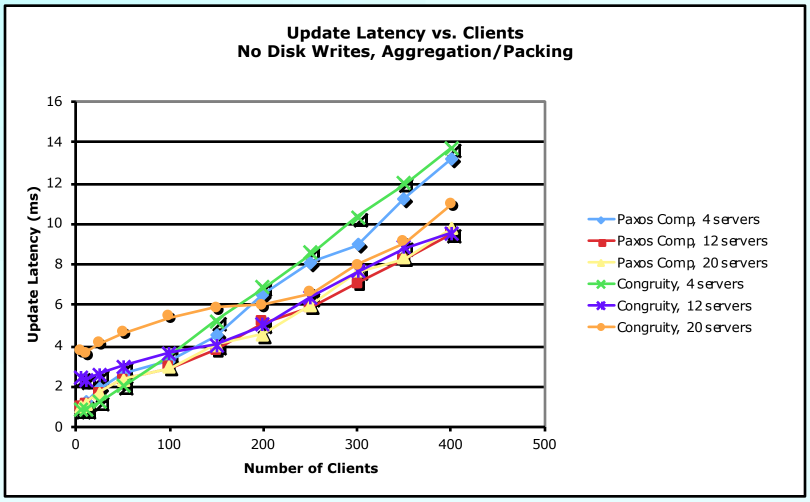
Performance Results (Paxos-SB)



Performance Results (Paxos-SB)



Performance Results (Paxos-SB)



Replication Methods

- Two phase commit, three phase commit
- Primary and backups
- Weak consistency (weaker update semantics)
- Quorum (primary component) methods with state machine replication
  - Congruity: Virtual Synchrony based replication.
  - Paxos: Leader based replication
  - Raft: Leader based replication with better understandability
    - A look at Raft based on Raft presentation
- Analysis and summary
  - From algorithms to deployment
    - Wide area latency analysis for Congruity and Paxos/Paxos-SB/Raft
    - Optimal global deployment considerations