Computer Science 425 Distributed Systems (Fall 2009)

Lecture 23

Replication Control

Reading: Section 15.1-15.4.1

Klara Nahrstedt

Acknowledgement

- The slides during this semester are based on ideas and material from the following sources:
  - Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.
  - Slides from Professor S. Gosh’s course at University o Iowa.

Administrative

- **MP3 posted**
  - **Deadline December 7 (Monday) – pre-competition**
    - » **Top five groups will be selected for final demonstration on Tuuesday, December 8**
  - **Demonstration Signup Sheets for Monday, 12/7, will be made available**
  - **Main Demonstration in front of the Qualcom Representative will be on Tuesday, December 8 afternoon - details will be announced.**
- **HW4 posted November 10, 2009**
  - **Deadline December 1, 2009 (Tuesday)**

Plan for Today

- Replication
- Review of View Concept and Group Communication
- Passive Replication
- Active Replication
- Gossiping Architecture

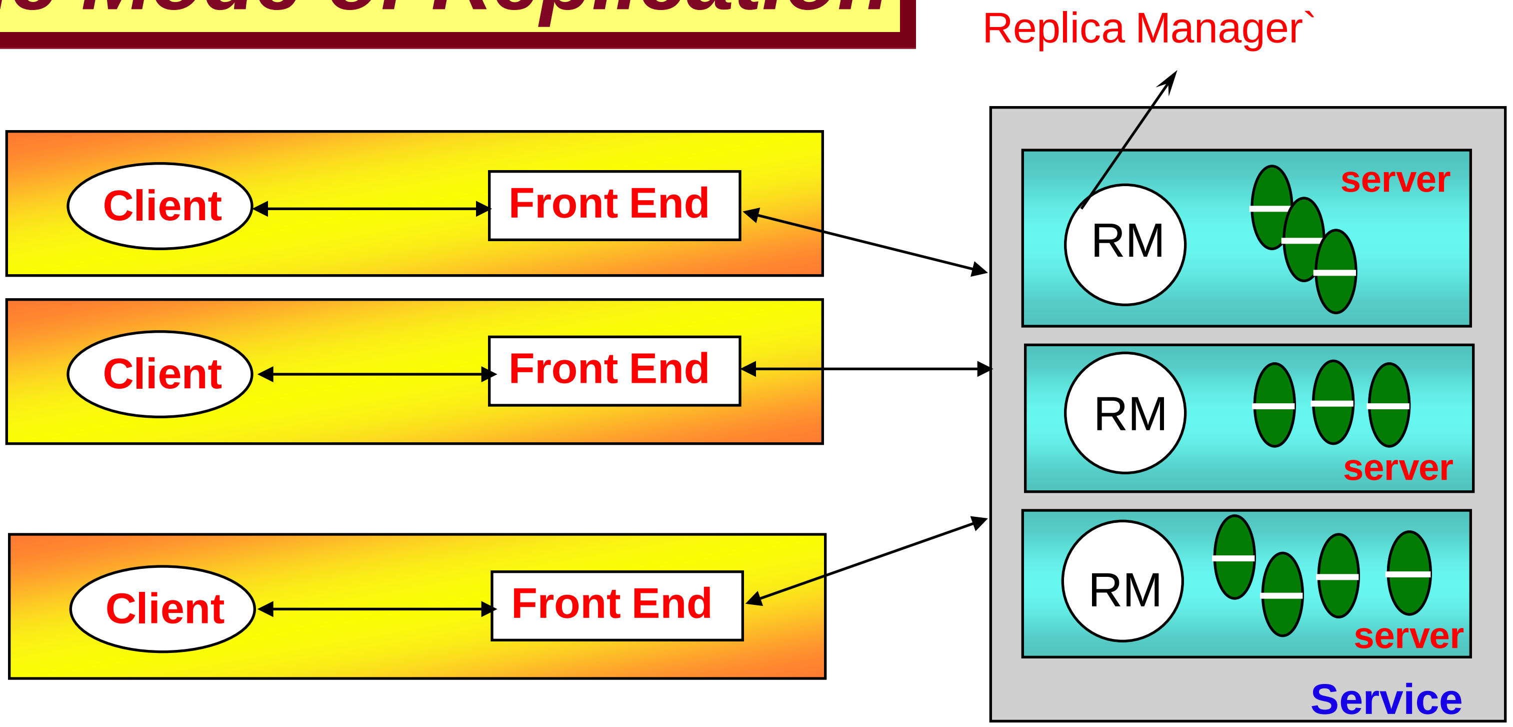
Replication

- **Enhancing Services by replicating data**
  - **Load Balancing**
    - **Example: Workload is shared between the servers by binding all the server IP addresses to the service's DNS name. A DNS lookup of the site results in one of the servers' IP addresses being returned, in a round-robin fashion.**
  - **Fault Tolerance**
    - **Under the fail-stop model, if up to *f* of *f+1* servers crash, at least one remains to supply the service.**
  - **Increased Availability**
    - **Service may not be available when servers fail or when the network is partitioned.**

P: probability that one server fails= 1 – P= availability of service. e.g. P = 5% => service is available 95% of the time.

P^n: probability that n servers fail= 1 – P^n= availability of service. e.g. P = 5%, n = 3 => service available 99.875% of the time

Basic Mode of Replication



- **Replication Transparency**

  User/client need not know that multiple physical copies of data exist.

- **Replication Consistency**

  Data is consistent on all of the replicas (or is in the process of becoming consistent)

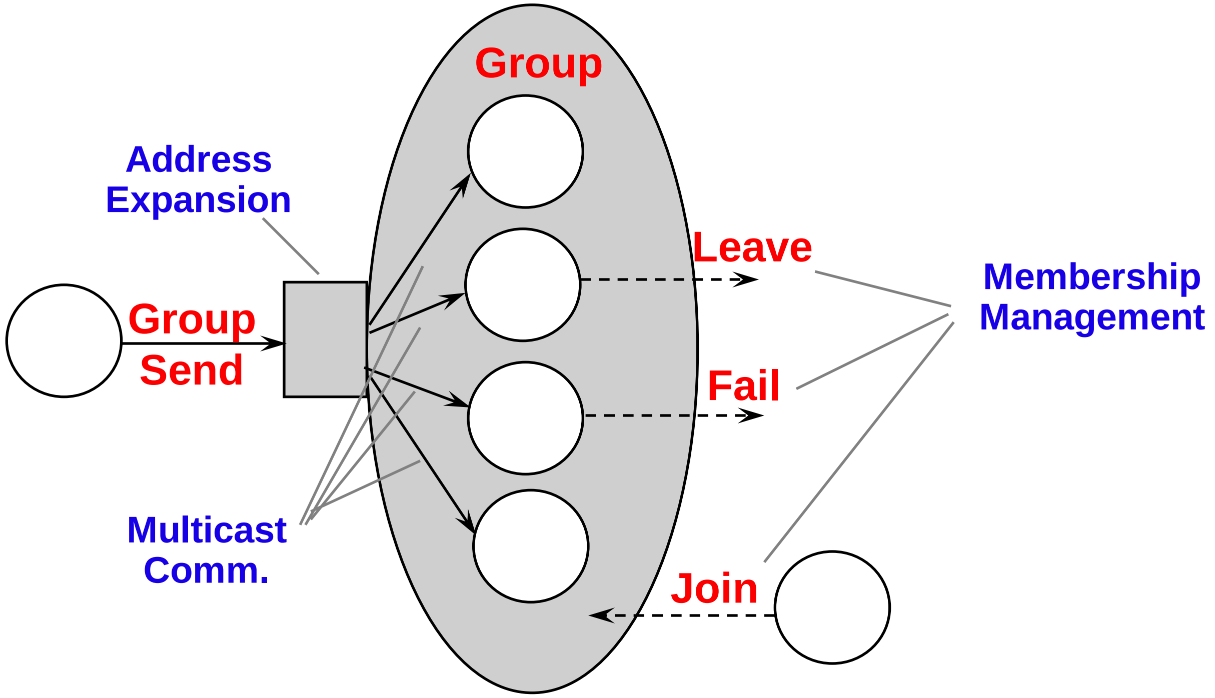
Replication Management (5 Steps)

- **Request Communication**
  - **Requests can be made to a single Replication Manager (RM) or to multiple RMs**
- **Coordination: The RMs decide**
  - **whether the request is to be applied**
  - **the order of requests**
    - **FIFO ordering: If a FE issues *r* then *r'*, then any correct RM handles *r* and then *r'*.**
    - **Causal ordering: If the issue of *r* "happened before" the issue of *r'*, then any correct RM handles *r* and then *r'*.**
    - **Total ordering: If a correct RM handles *r* and then *r'*, then any correct RM handles *r* and then *r'*.**
- **Execution: The RMs execute the request tentatively.**

Replication Management

- **Agreement: The RMs attempt to reach consensus on the effect of the request.**
  - **E.g., Two phase commit through a coordinator**
  - **If this succeeds, effect of request is made permanent**
- **Response**
  - **One or more RMs responds to the front end.**
  - **In the case of fail-stop model, the Front End (FE) returns the first response to arrive.**

Group Communication - Review



- "Member"= process (e.g., RM)
- **Static Groups**: group membership is pre-defined
- **Dynamic Groups**: Members may join and leave, as necessary

Group Views - Review

- A ***group membership service*** maintains group ***views***, which are lists of current group members.
  - This is NOT a list maintained by one member, but…
  - ***Each member maintains its own local view***
- A view $V_p(g)$ is process *p*'s understanding of its group (list of members)
  - Example: $V_{p.0}(g) = \{p\}$, $V_{p.1}(g) = \{p, q\}$, $V_{p.2}(g) = \{p, q, r\}$, $V_{p.3}(g) = \{p, r\}$
- A new group view is disseminated, throughout the group, whenever a member joins or leaves.
  - Member detecting failure of another member reliable multicasts a "view change" message (requires causal-total ordering for multicasts)

Group Views - Review

- An event is said to occur in a view $V_{p,i}(g)$ *if the event occurs at p, and at the time of event occurrence, p has delivered $V_{p,i}(g)$ but has not yet delivered $V_{p,i+1}(g)$.*
- Messages sent out in a view *i* need to be delivered in that view at ***all*** members in the group ("What happens in the View, stays in the View")
- ***Requirements for view delivery***
  - Order: If *p* delivers $V_i(g)$ and then $V_{i+1}(g)$, then no other process *q* delivers $V_{i+1}(g)$ before $V_i(g)$.
  - Integrity: If *p* delivers $V_i(g)$, then *p* is in $V_i(g)$.
  - Non-triviality: if process *q* joins a group and becomes reachable from process *p*, then eventually, *q* will always be present in the views that are delivered at *p*.

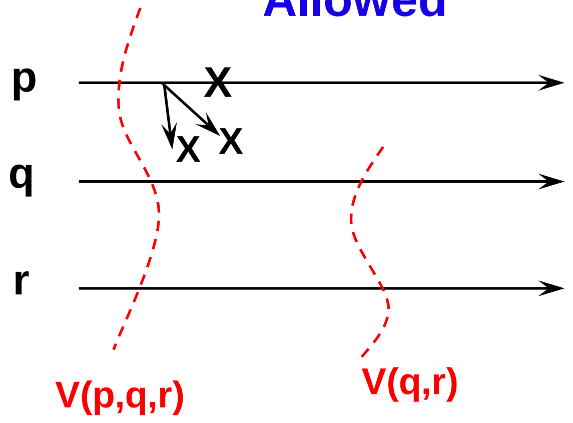
View Synchronous Communication - Review

- **View Synchronous Communication = Group Membership Service + Reliable multicast**
- **The following guarantees are provided for multicast messages:**
  - **Integrity: If *p* delivered message *m*, *p* will not deliver *m* again. Also *p* $\in$ *group (m).***
  - **Validity: Correct processes always deliver all messages. That is, if *p* delivers message *m* in view V*(g)*, and some process q $\in$ *V(g)* does not deliver *m* in view *V(g),* then the next view *V'(g)* delivered at *p* will not include *q.***
  - **Agreement: Correct processes deliver the same set of messages in any view.**

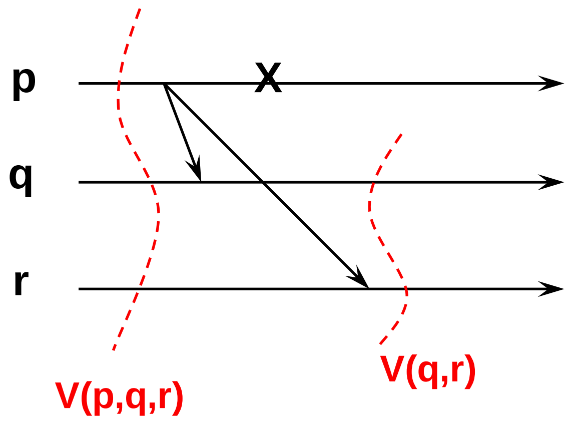
    if *p* delivers *m* in *V*, and then delivers *V'*, then all processes in *V* $\cap$ *V'* deliver *m* in view *V*
  - **All View Delivery conditions (Order, Integrity and Non-triviality conditions, from last slide) are satisfied**
- **"What happens in the View, stays in the View"**

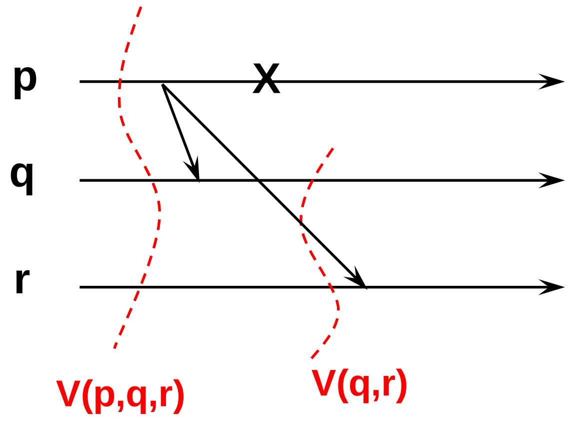
Example: View Synchronous Communication

Allowed

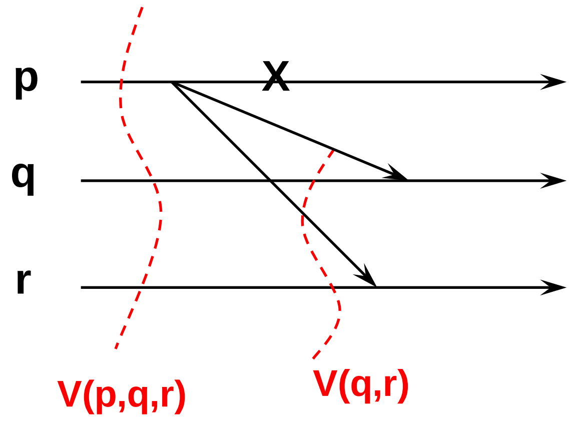


Allowed





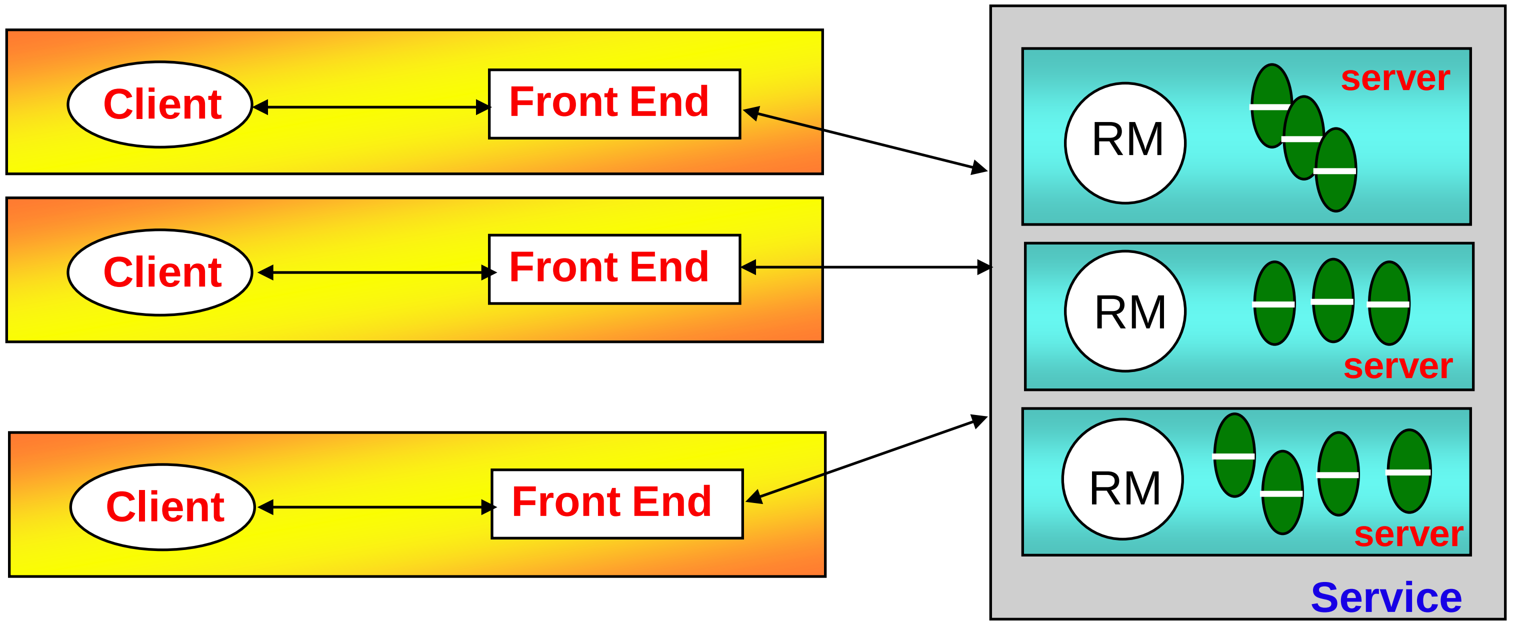
Not Allowed



Not Allowed

FAULT-TOLERANT SERVICES

Back to Replication



Need <u>consistent</u> updates to all copies of an object

- •Linearizability
- •Sequential Consistency

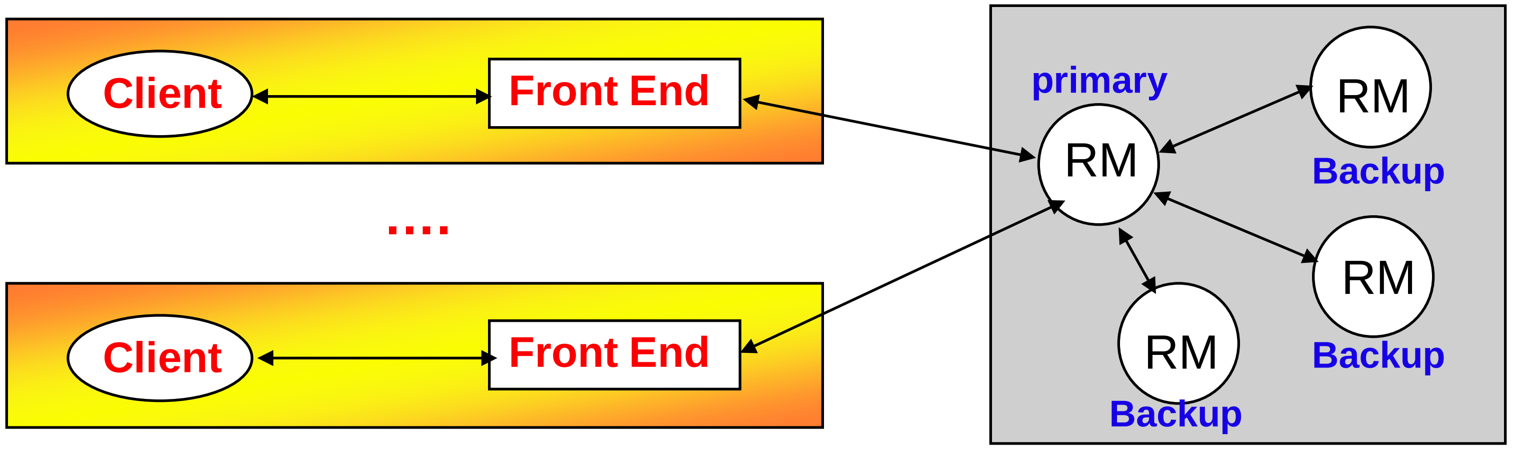
Linearizability

- Let the sequence of read and update operations that client *i* performs in some execution be o_{i1}, o_{i2},….
    - "Program order" for the client
- A replicated shared object service is linearizable if for any execution (real), there is some interleaving of operations (virtual) issued by all clients that:
    - meets the specification of a single correct copy of objects
    - is consistent with the ***real times*** at which each operation occurred during the execution
- **Main goal: any client will see (at any point of time) a copy of the object that is correct and consistent**

Sequential Consistency

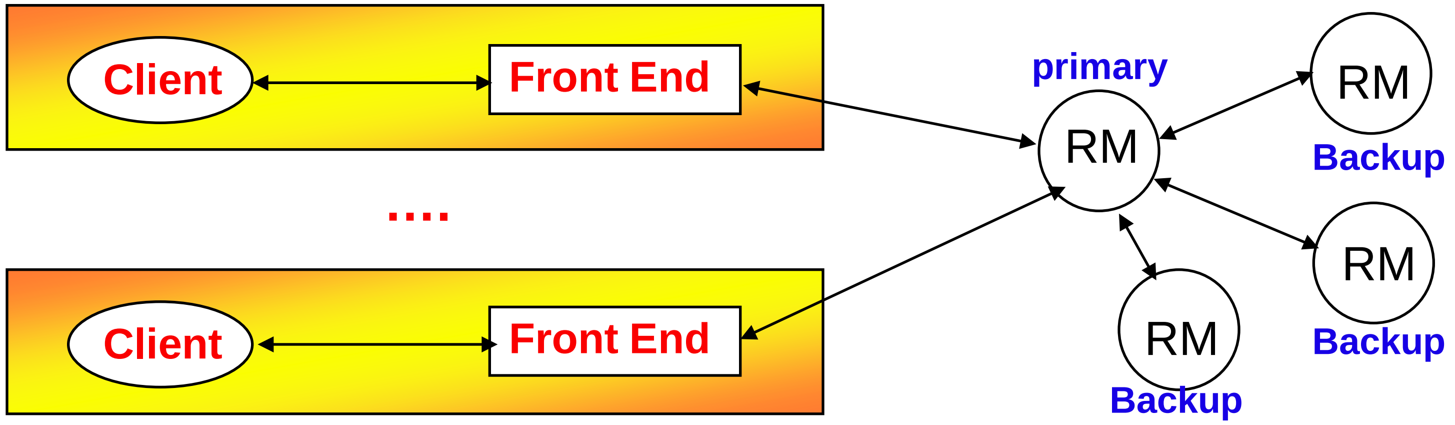
- The real-time requirement of linearizability is hard, if not impossible, to achieve in real systems (why?)
- A less strict criterion is sequential consistency: A replicated shared object service is sequentially consistent if for any execution (real), there is some interleaving of clients' operations (virtual) that:
  - meets the specification of a single correct copy of objects
  - is consistent with the program order in which each individual client executes those operations.
- This approach does not require absolute time or total order. Only that for each client the order in the sequence be consistent with that client's program order (~ FIFO).
- Linearilizability implies sequential consistency. Not vice-versa!
- Challenge to guarantee sequential consistency
  - Ensuring that all replicas of an object are consistent.

Passive (Primary-Backup) Replication

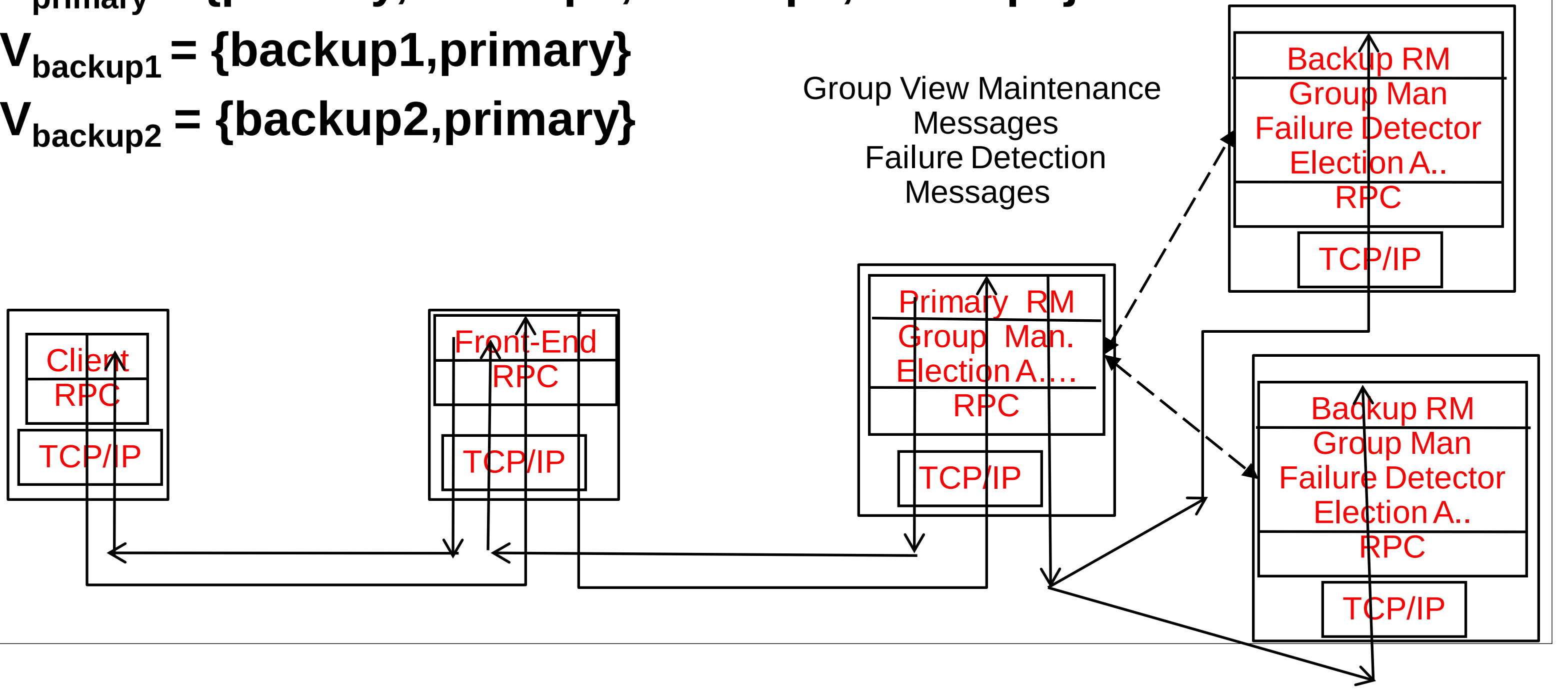


- **Request Communication:** the request is issued to the primary RM and carries a unique request id.
- **Coordination:** Primary takes requests atomically, in order, checks id (resends response if not new id.)
- **Execution:** Primary executes & stores the response
- **Agreement:** If update, primary sends updated state/result, req-id and response to all backup RMs (1-phase commit enough).
- **Response:** primary sends to the front end

View Synchrony - Example



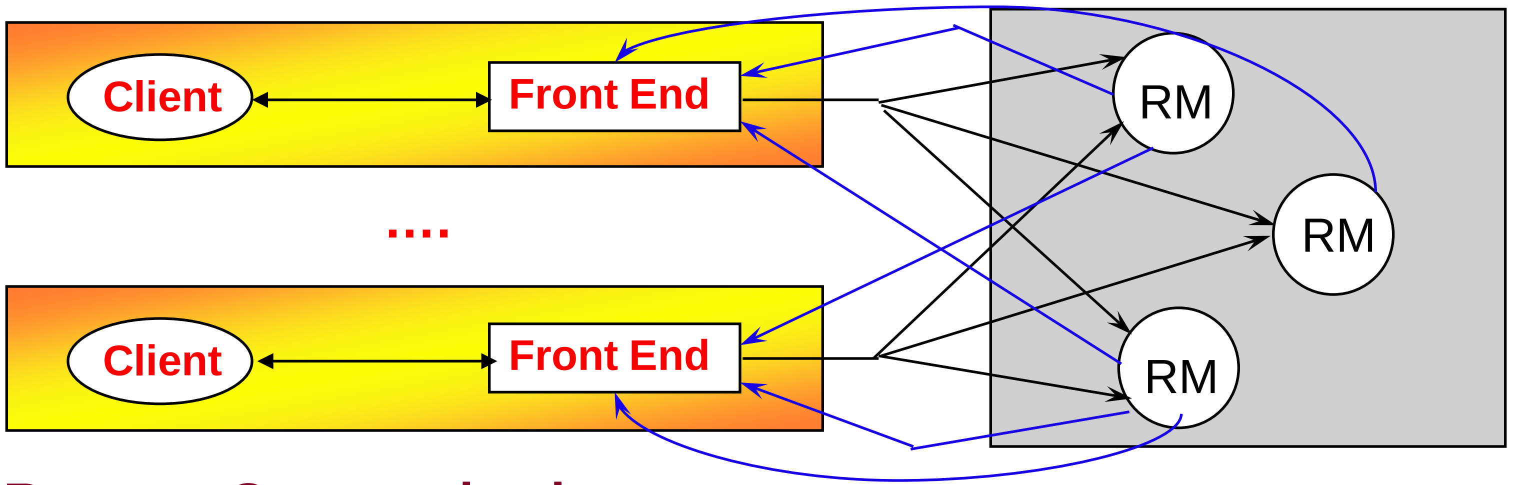
- $V_{primary}$ = {primary, backup1, backup2, backup3}
- $V_{backup1}$ = {backup1,primary}
- $V_{backup2}$ = {backup2,primary}



Fault Tolerance in Passive Replication

- The system implements linearizability, since the primary sequences operations in order.
- If the primary fails, a backup becomes primary by leader election, and the replica managers that survive agree on which operations had been performed at the point when the new primary takes over.
  - **The above requirement is met if the replica managers (primary and backups) are organized as a group and if the primary uses view-synchronous group communication to send updates to backups.**
- The system remains linearizable after the primary crashes

Active Replication



- **Request Communication:** The request contains a unique identifier and is multicast to all by a reliable totally-ordered multicast.
- **Coordination:** Group communication ensures that requests are delivered to each RM in the same order (but may be at different physical times!).
- **Execution:** Each replica executes the request. (Correct replicas return same result since they are running the same program, i.e., they are *replicated protocols* or *replicated state machines*)
- **Agreement:** No agreement phase is needed, because of multicast delivery semantics of requests
- **Response:** Each replica sends response directly to FE

Fault Tolerance in Active Replication

- **RMs work as <u>replicated state machines</u>, playing equivalent roles. That is, each responds to a given series of requests in the same way. This is achieved by running the same program code at all RMs.**
- **If any RM crashes, state is maintained by other correct RMs.**
- **This system implements sequential consistency**
  - **The total order ensures that all correct replica managers process the same set of requests in the same order.**
  - **Each front end’s requests are served in FIFO order (because the front end awaits a response before making the next request).**
- **So, requests are FIFO-total ordered. But if clients are multi-threaded and communicate with one another while waiting for responses from the service, we may need to incorporate causal-total ordering**

Eager versus Lazy

- **Eager replication**
  - **Multicast request to all RMs immediately in active replication**
  - **Multicast results to all RMs immediately in passive replication**
- **Alternative: Lazy replication**
  - **Allow replicas to converge eventually and lazily**
  - **FEs need to wait for reply from only one RM**
  - **Allow other RMs to be disconnected/unavailable**
  - **Propagate updates and queries lazily**
  - **May provide weaker consistency than sequential consistency, but improves performance**
  - **Concepts also applicable in building disconnected file systems**

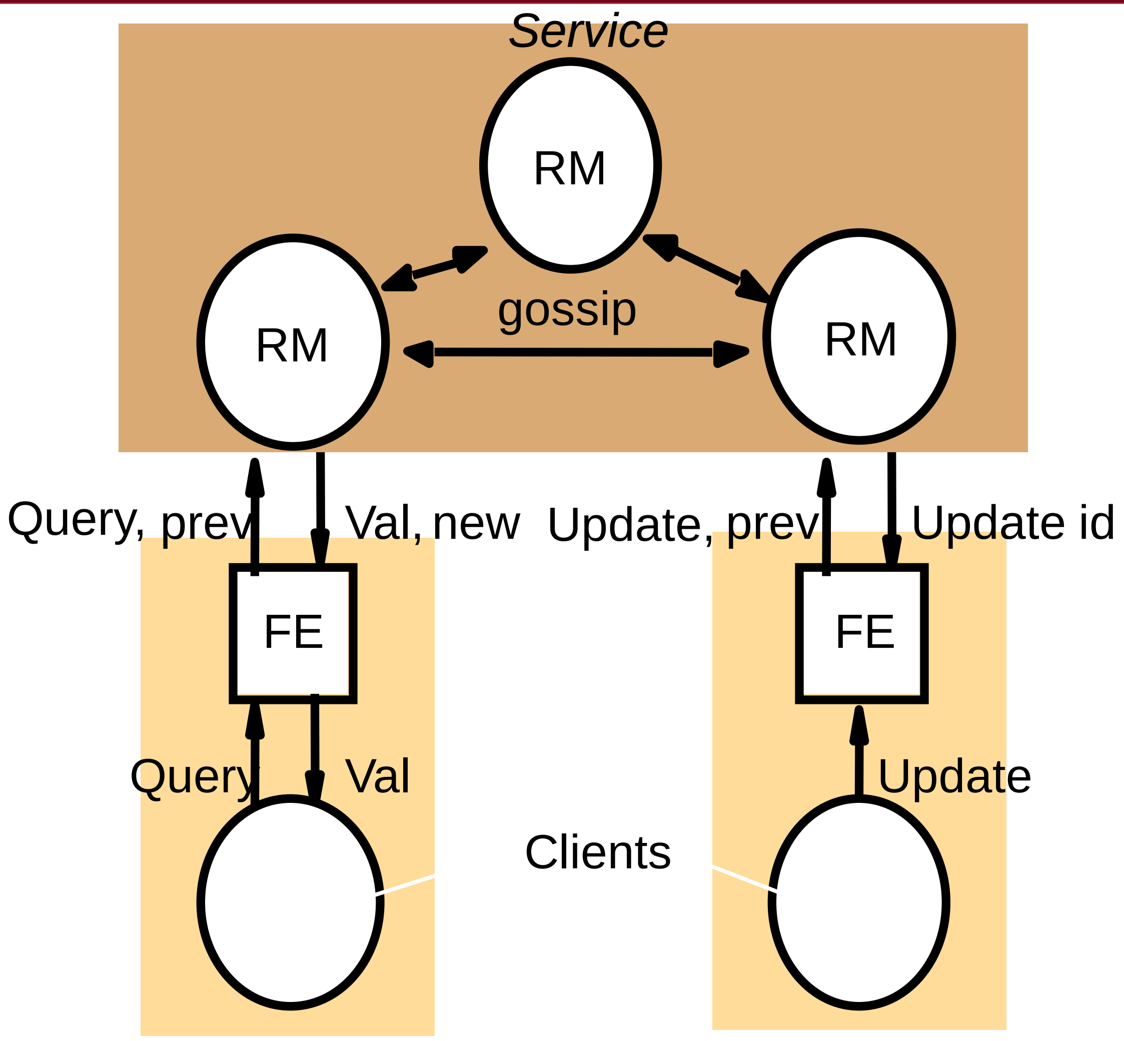
- **Lazy replication can be provided by using the gossip architecture**

Gossiping Architecture

- **The RMs exchange “gossip” messages**

  (1) periodically and (2) amongst each other. Gossip messages convey updates they have each received from clients, and serve to achieve anti-entropy (convergence of all RMs).
- **Objective: provisioning of highly available service. Guarantee:**
  - **Each client obtains a consistent service over time:** **in response to a query, an RM may have to wait until it receives “required” updates from other RMs. The RM then provides client with data that at least reflects the updates that the client has observed so far.**
  - **Relaxed consistency among replicas:** **RMs may be inconsistent at any given point of time. Yet all RMs eventually receive all updates and they apply updates with ordering guarantees. Can be used to provide sequential consistency.**

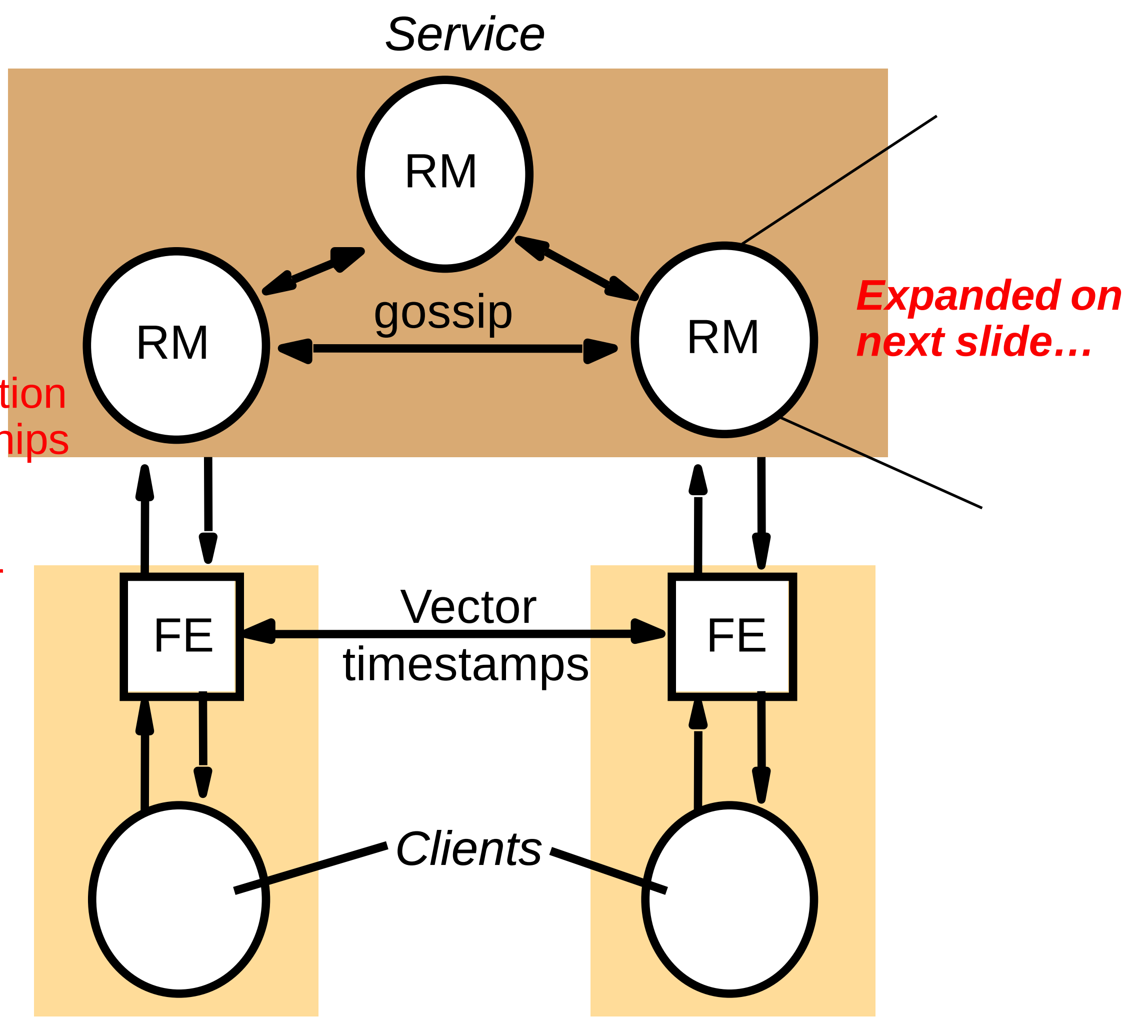
Query and Update Operations in a Gossip Service



Various Timestamps

- **Virtual timestamps are used to control the order of operation processing. The timestamp contains an entry for each RM (i.e., it is a vector timestamp).**
- **Each front end keeps a vector timestamp, *prev*, that reflects the latest data values accessed by the front end. The FE sends it in every request to a RM.**
- **Replies to FE:**
  - **When an RM returns a value as a result of a query operation, it supplies a new timestamp, *new*.**
  - **An update operation returns a timestamp, *update id*.**
- **Each returned timestamp is *merged* with the FE's previous timestamp to record the data that has been observed by the client.**

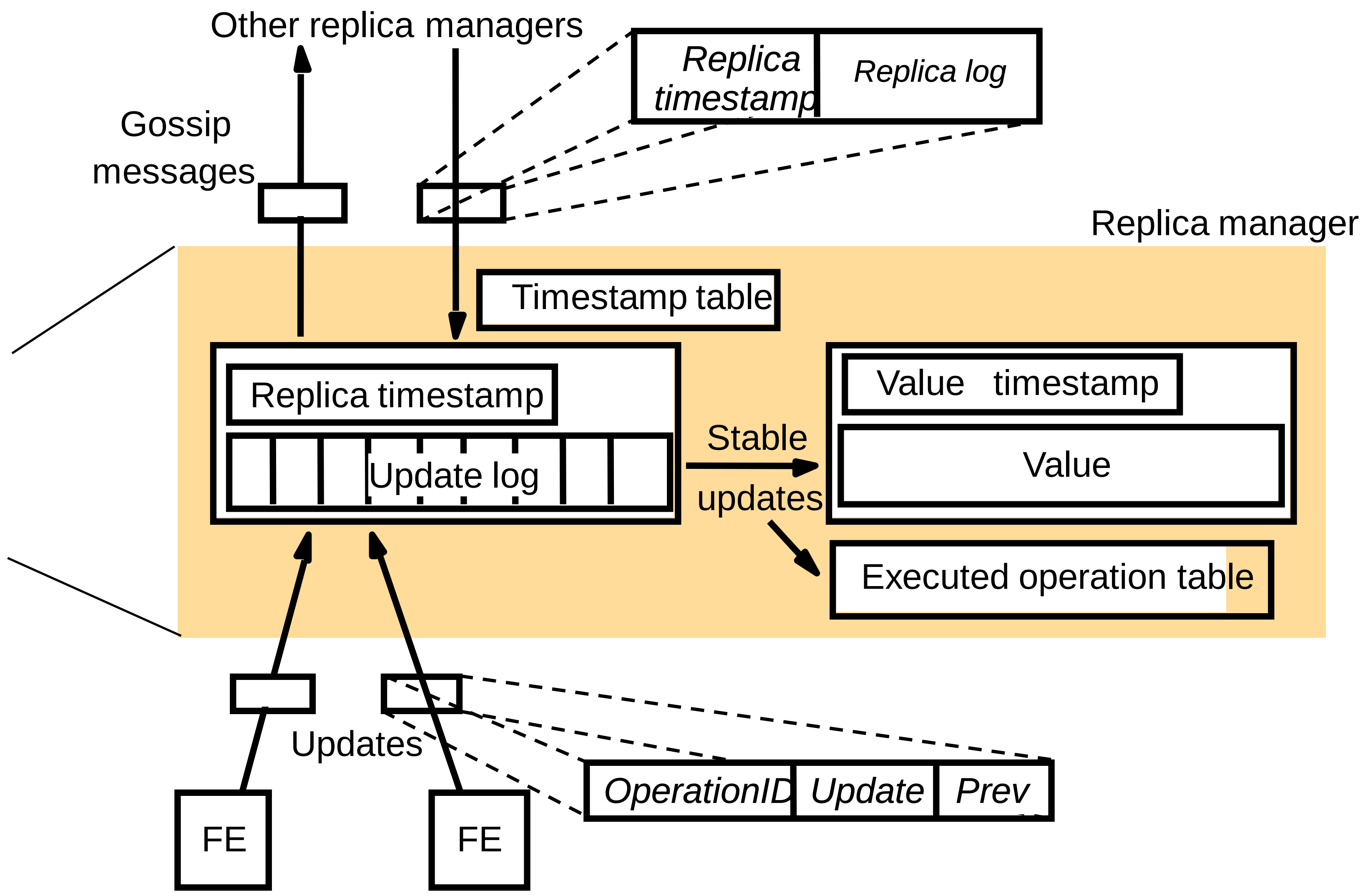
Front ends Propagate Their Timestamps

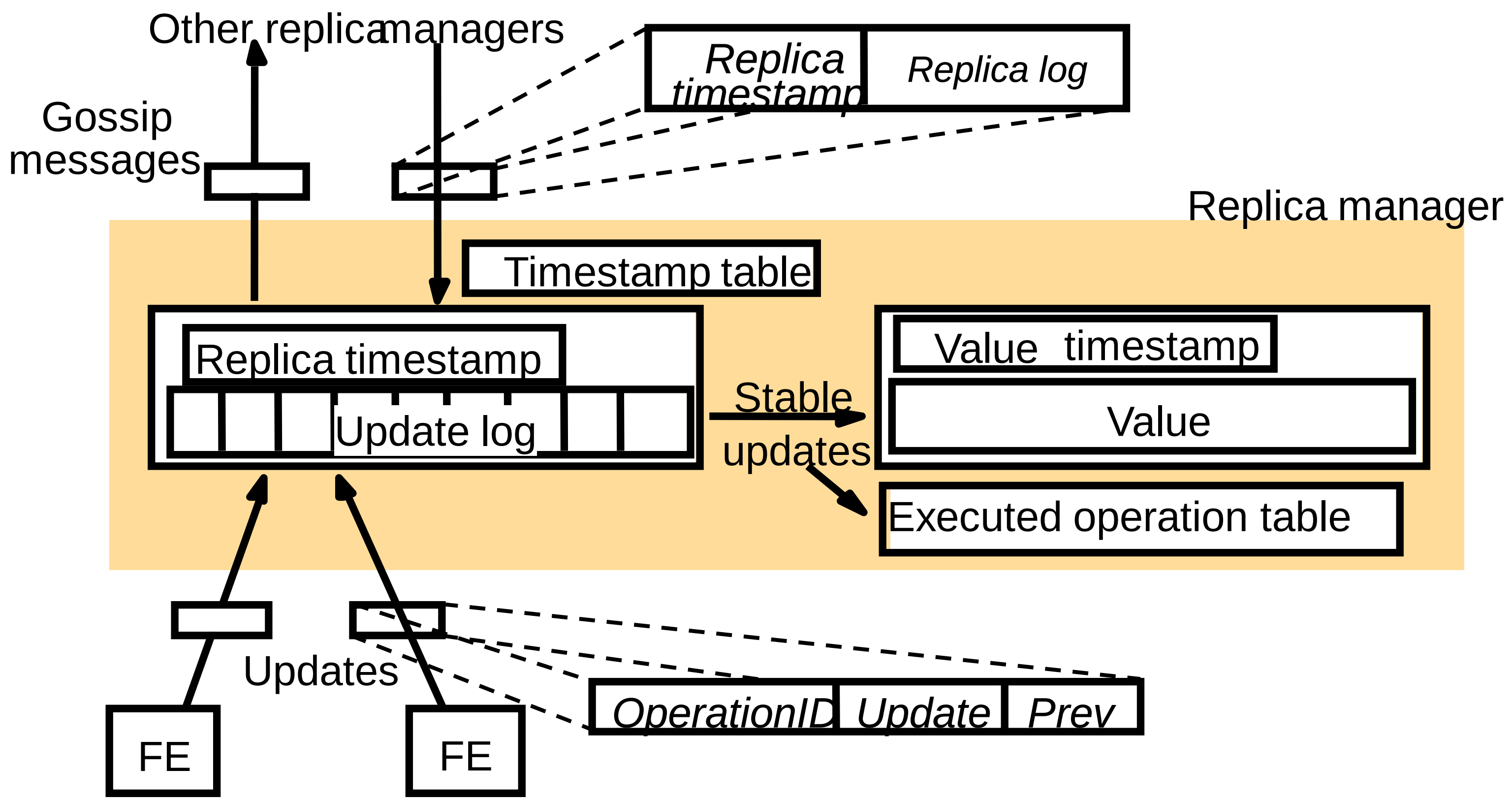


Since client-to-client communication can also lead to causal relationships between operations applied to services, the FE piggybacks its timestamp on messages to other clients.

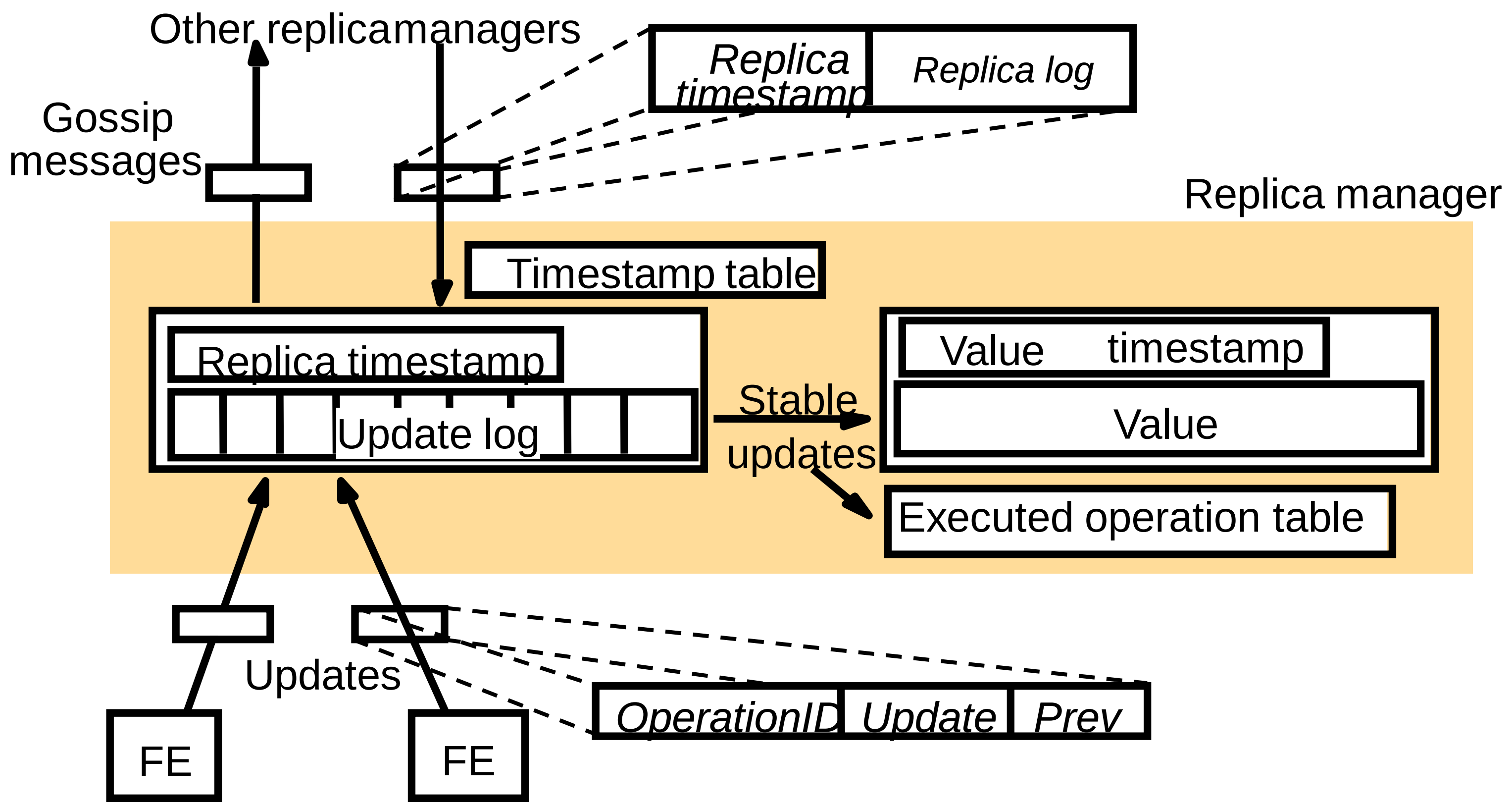
Expanded on next slide…

A Gossip Replica Manager

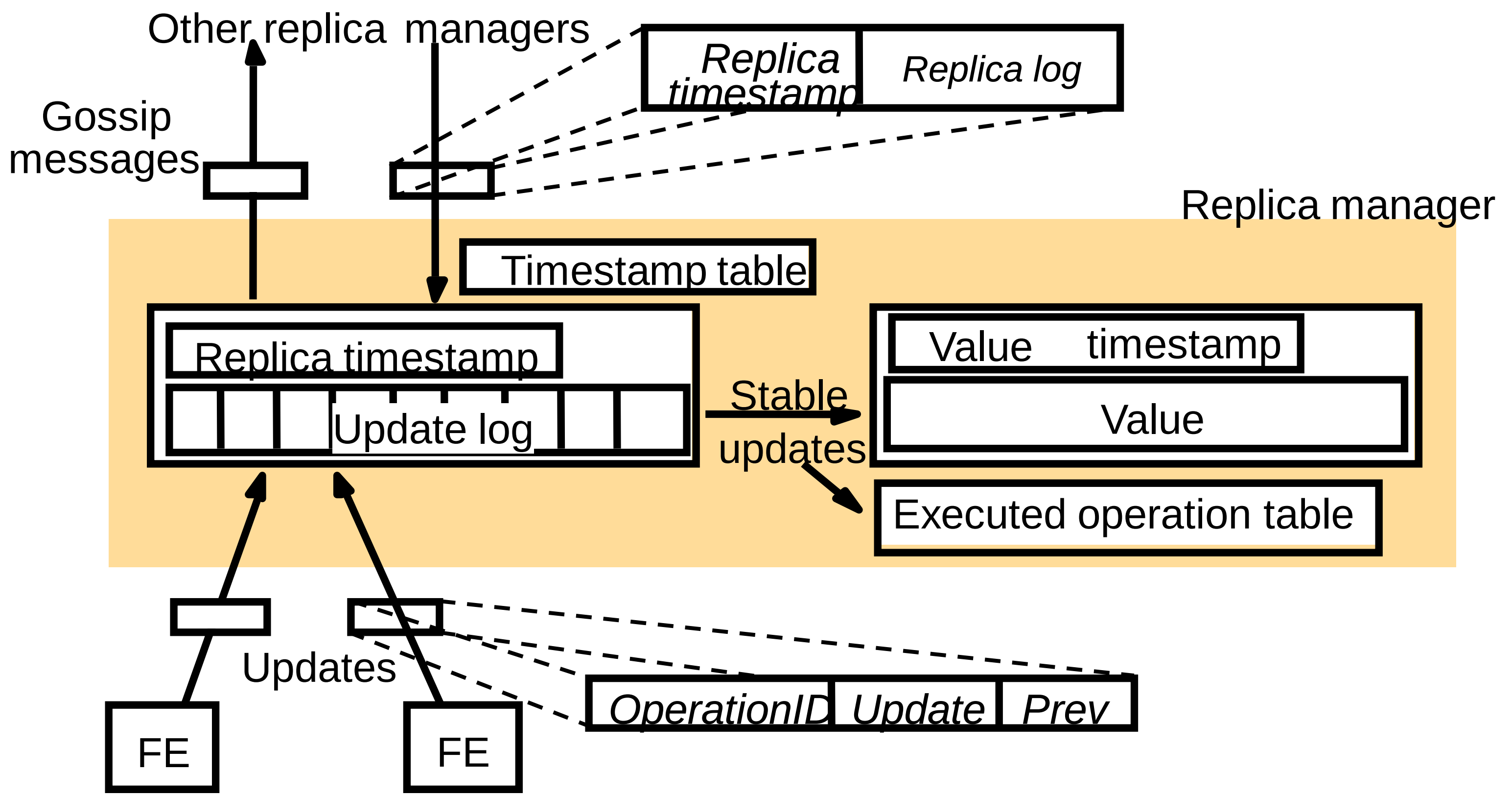




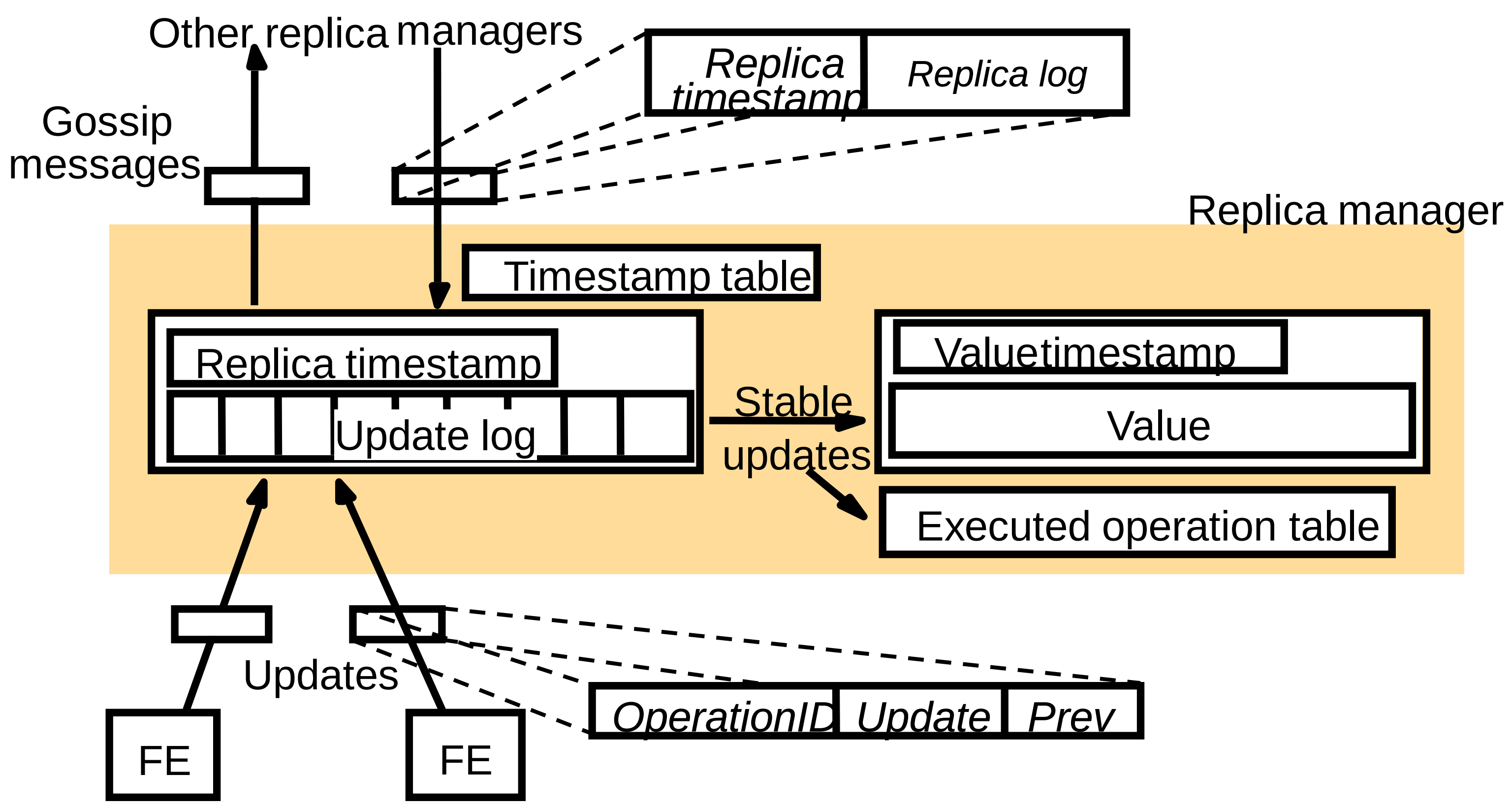
- **Value: value of the object maintained by the RM.**
- **Value timestamp: the timestamp that represents the updates reflected in the value. Updated whenever an update operation is applied.**



- **Update log: records all update operations as soon as they are received, until they are reflected in Value.**
  - **Keeps all the updates that are not stable, where a stable update is one that has been received by all other RMs and can be applied consistently with its ordering guarantees.**
  - **Keeps stable updates that have been applied, but cannot be purged yet, because no confirmation has been received from all other RMs.**
- **Replica timestamp: represents updates that have been accepted by the RM into the log.**



- **Executed operation table: contains the FE-supplied ids of updates (stable ones) that have been applied to the value.**
  - **Used to prevent an update being applied twice, as an update may arrive from a FE and in gossip messages from other RMs.**
- **Timestamp table: contains, for each other RM, the latest timestamp that has arrived in a gossip message from that other RM.**



- **The *ith* element of a vector timestamp held by *RM_i* corresponds to the number of updates received from FEs by *RM_i***
- **The *jth* element of a vector timestamp held by *RM_i* (*j* not equal to *i*) equals the number of updates received by *RM_j* and forwarded to *RM_i in gossip messages.***

Summary

- Replicating objects across servers improves performance, fault-tolerance, availability
- Raises problem of Replica Management
- View Synchronous communication service provides totally ordered delivery of views+multicasts
- RMs can be built over this service
- Passive and Active Replication
- Introduction to Gossiping Architecture

- Reading for next lecture: Section 15.4.1 and 15.5