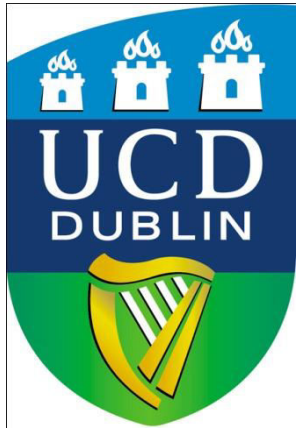
Distributed Systems:
Replication Systems

Anca Jurcut
E-mail: anca.jurcut@ucd.ie

School of Computer Science and Informatics
University College Dublin
Ireland



From Previous Lecture…

- **Case Study 1: Pastry**
  - **Pastry GUIDs**
  - **Pastry Routing Overlay: The Simple Version Algorithm and The Full Routing Algorithm**
  - **Analysis of Pastry**
  - **Pastry and PAST**
- **Case Study 2: BitTorrent**
  - **Peer to Peer File Sharing Protocol used for distributing large amounts of data**
  - **File Sharing – how it works**
  - **Basic Components**
  - **Overview – System Components**
  - **Tracker Protocol**
  - **Goals:** Efficiency and Reliability
  - **Distributed Copies** (the number of distributed copies is the number of copies of the rarest piece)
  - **Rarest First** (piece picking algorithm used in BitTorrent)
- **Case Study 3: Plaxton Mesh**

From Previous Lecture…

- **Replication Systems**
  - **Replication:**"the maintenance of multiple copies of data at multiple computers"
  - Key issue in the design of **effective** distributed systems
  - Provide: Enhanced Performance, High Availability, Fault Tolerance
  - **Key Features:** Replication Transparency, Consistency
  - **Replication System** is the sub-system (service) of a distributed system that is concerned with replication of data
  - Implemented as a set of **Replica Managers (RM)**
  - **5 STEPS** in Handling a request to perform an operation on a logical object
    - **Request:** FE issues the request to one or more RMs
    - **Coordination**: RMs coordinate to execute the request consistently
    - **Execution**: The RMs execute the request
    - **Agreement**: The RMs reach consensus on the effect of the request
    - **Response**: One or more RMs respond to the FE

From Previous Lecture…

- **Replication Systems**
  - **Coordination Techniques**
  - **Recoverability**
  - **State-Based Replication**
  - **Static versus Dynamic Systems**
  - **Example: Diary System**

Distributed Systems: Group Communication

Group Communication

- **Multicast communication**
  - A mechanism for sending a single message to a group of processes (almost) simultaneously.

- Multicast Implementations must account for:
  - Static vs Dynamic Groups
  - Multicasting vs Unicasting

- Benefits:
  - **Efficiency:** can minimise use of bandwidth by sending one message to n processes instead of n messages to n processes.
  - **Delivery Guarantees:** If a Unicast based approach is used, then failure can lead to only some of the processes receiving the messages & relative ordering of messages is undefined.

Multicasting

- A **multicast message** is sent from *one process* to the members of some *group of processes*.
- Range of possible behaviours:
  - **Unreliable multicast:** No guarantee of delivery or ordering – message only transmitted once.
  - **Reliable multicast:** Best effort is made to deliver to all members of the receiving group.
  - **Atomic multicast:** Message is either received by all processes in the receiving group or none of them.

IP Multicast

- Extension of IP that supports transmission of messages to groups of processes.
  - Employs IP addresses in the range: 224.x.x.x to 239.x.x.x
  - Supports dynamic group membership.
  - Uses UDP transport message format: Message delivery not guaranteed.

- **Multicast Routers** are used to route messages both locally and over the Internet.
  - **Mbone** (The Multicast Backbone): A loose confederation of IP routers that support routing of IP Multicast packets over the Internet.
  - Can use Time To Live (TTL) value to limit propagation distance

- IP Multicast is mainly used within clusters, server farms, etc.

Multicast System Model

- Assume a collection of processes that can communicate reliably over one-to-one channels.
  - Organise processes into groups
  - Membership of multiple groups is permitted
  - Each group has a globally unique identifier.

- Operations:
  - **multicast(g, m)** sends the message **m** to all members of group **g**.
  - **deliver(m)*** delivers the message **m**, sent by multicast, to the calling process.
  - **sender(m)** returns the sender of message **m**.
  - **group(m)** returns the unique destination group identifier of **m**.

NOTE: * The term deliver is used instead of receive because multicast messages are not always handed to the application layer inside when they are received at the processes node.

Basic Multicast

- A primitive multicast protocol that guarantees the delivery of messages to all processes in a (static) group.
  - **B-multicast(g, m)**: send message **m** to group **g**.
  - **B-deliver(m)**: delivers message **m**, sent by multicast, to the calling process.

- Implementation builds on an assumed unicast protocol that guarantees message delivery (e.g. TCP).
  - **send(p, m)** sends message **m** to process **p**
  - **receive(m)** the application layer process receives message **m**

- Multicast implemented as follows:
  - On **B-multicast(g, m)**: for each process **p** $\in$ Group **g**, **send(p, m)**
  - On **receive(m)** at **p**: **B-deliver(m)** at **p**

Basic Multicast

- Multi-threading can be used to send multiple unicast messages simultaneously.

- This approach suffers from “ACK-implosion”:

  - For reliable unicast, each **receive(m)** must send a **ACK**nowledgement back to the sender.

  - As the number of processes in a group increases, the number of ACKs also increases:

    - If they are returned at (about) the same time then the multicasting processes buffer will fill, causing ACKs to be dropped.
    - This will cause the multicasting process to retransmit the message to the failed destinations, causing more ACKs to be returned…

- As an alternative, we can build a more practical and reliable Multicast service using IP Multicasting.

Reliable Multicast

- Reliable Multicast is an extension of the Basic Multicast service:
  - Guaranteed Delivery (Basic Multicast)
  - Integrity:
    - A correct process **p** delivers a message **m** at most once
    - Furthermore, **p** $\in$ **group(m)** and **m** was supplied to a multicast operation by **sender(m)**.
  - Validity:
    - If a correct process multicasts message **m** then it will eventually deliver **m**.
  - Agreement:
    - If a correct process delivers message **m**, then all other correct processes in **group(m)** will eventually deliver **m**.

- Based on this, we introduce two new primitive operations:
  - **R-multicast(g, m)**: send message **m** reliably to group **g**.
  - **R-delivery(m)**: the application process reliably receives message **m**.

Reliable Multicast

- R-multicast can be implemented using a combination of:
  - IP Multicast: Assumption = IP multicast communication is often successful
  - Piggy-backed acknowledgements (ACKs): sent as part of Multicast messages
  - Negative acknowledgements (NAKs): sent when a process detects that it has missed a message

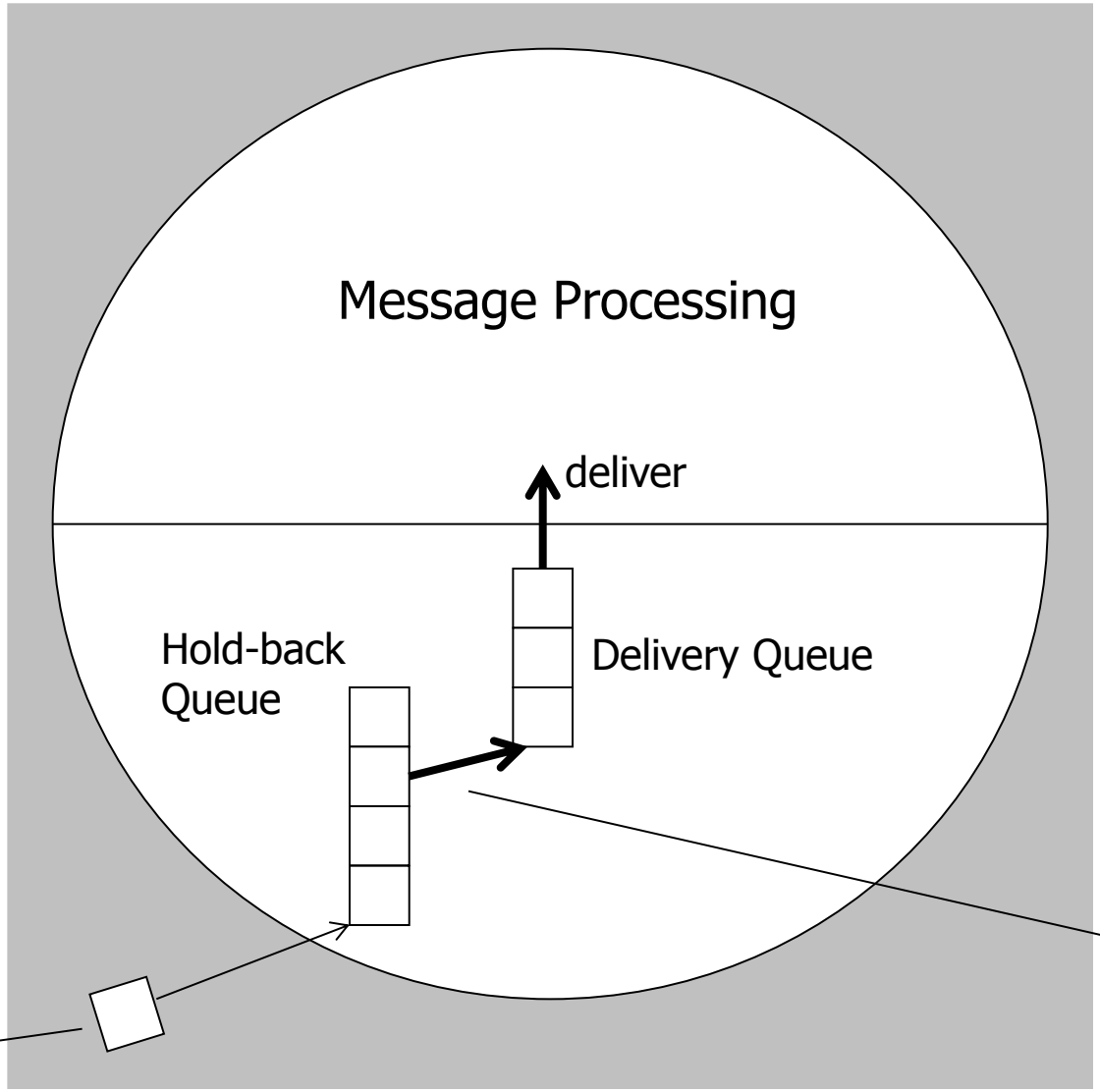
- To achieve this (for closed groups):
  - Each process p maintains a sequence number S^p_g (initially zero) for each group g that it belongs to. This records how many messages p has sent to g.
  - Each process also records S^q_g the sequence number of the latest message from process q in group g.
  - Whenever p multicasts to group g, it piggy backs the current value of S^p_g and acknowledgements of the form <q, R^q_g>
    - After sending the message, p increments S^p_g by one

Reliable Multicast

- Piggy backing allows the recipients to learn about messages that they have not received.
  - A process R-delivers a message destined for g bearing the sequence number S from p if and only if $S = R^p_g + 1$.
    - It increments R^p_g by one immediately after delivering the message.
  - If an arriving message has $S \leq R^p_g$, then it discards the message because it has already delivered it
  - If $S > R^p_g + 1$ or $R > R^q_g$ for an enclosed acknowledgement <q, R> then there are one or more messages that it has not yet received.
    - And which are likely to have been dropped

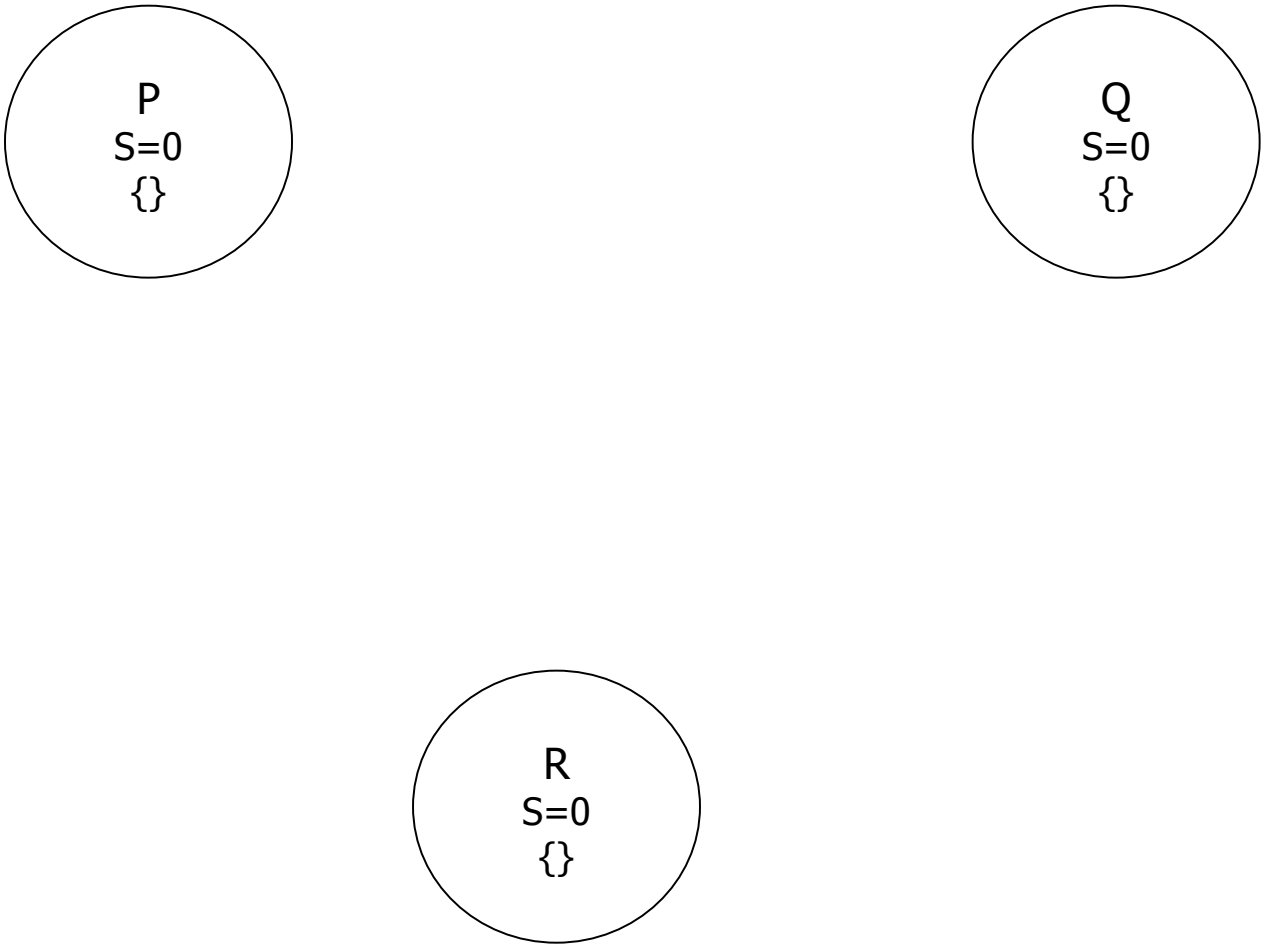
- Messages for which $S > R^p_g + 1$ are kept by the process in a **hold-back queue**.
  - Missing messages are then requested by sending NAKs to:
    - the original sender, or
    - to another process, q from which it has received the acknowledgement <q, R^q_g> where R^q_g is no less than the required sequence number.

Reliable Multicast



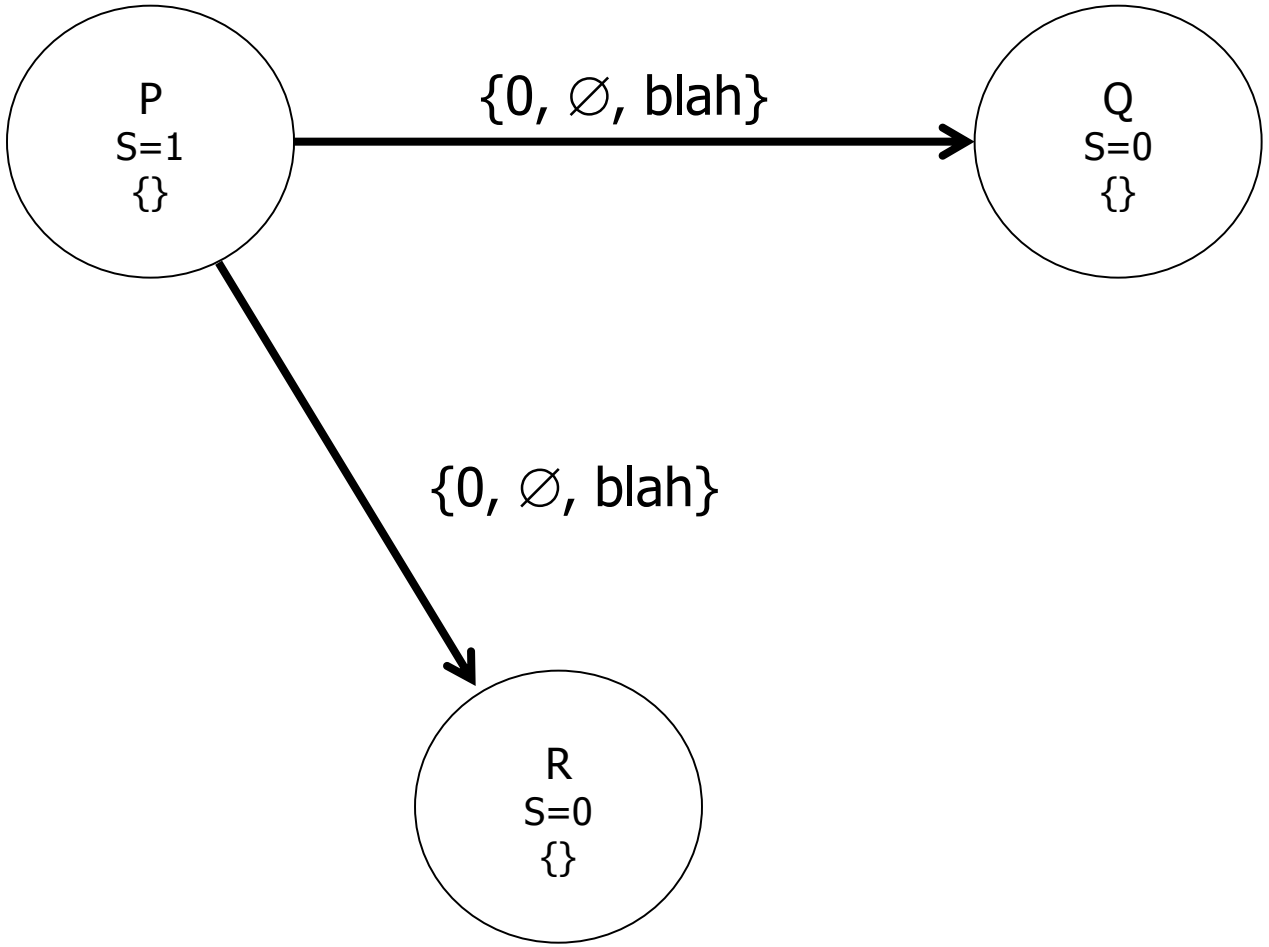
Reliable Delivery

- P, Q, R form a Multicast Group



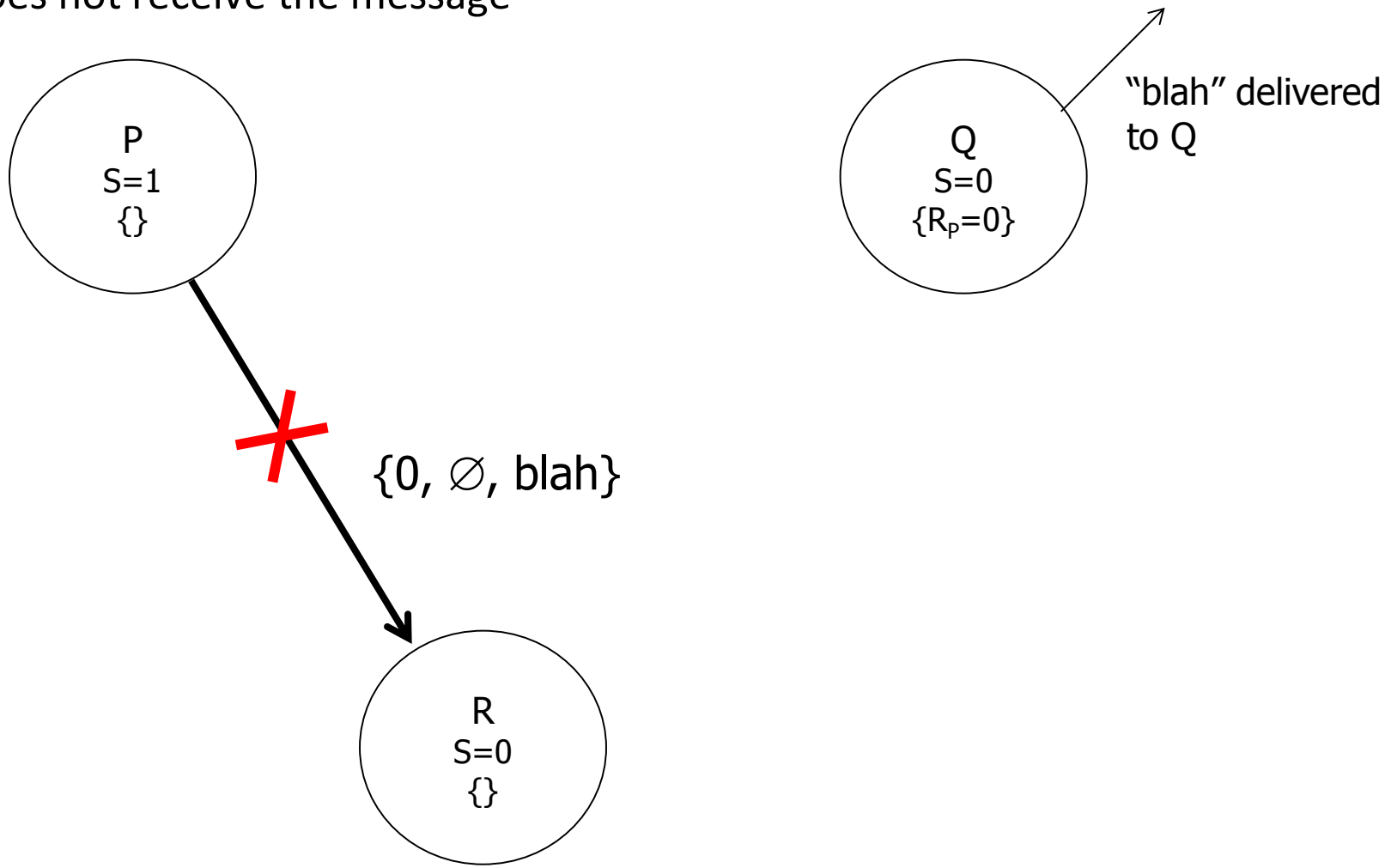
Reliable Delivery(2)

- P sends a message to the group



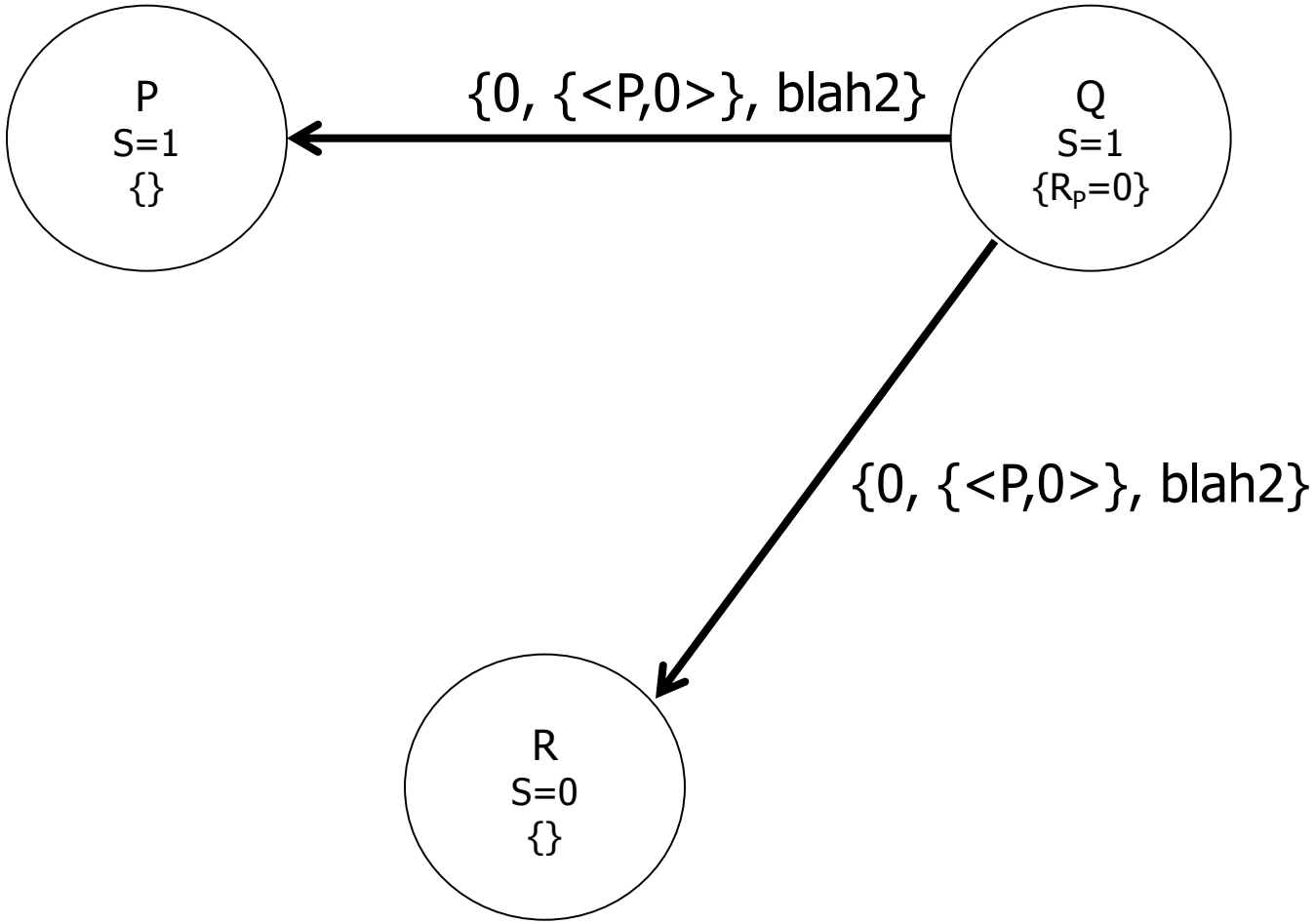
Reliable Delivery(3)

- Q receives and delivers message 0 from P
- R does not receive the message



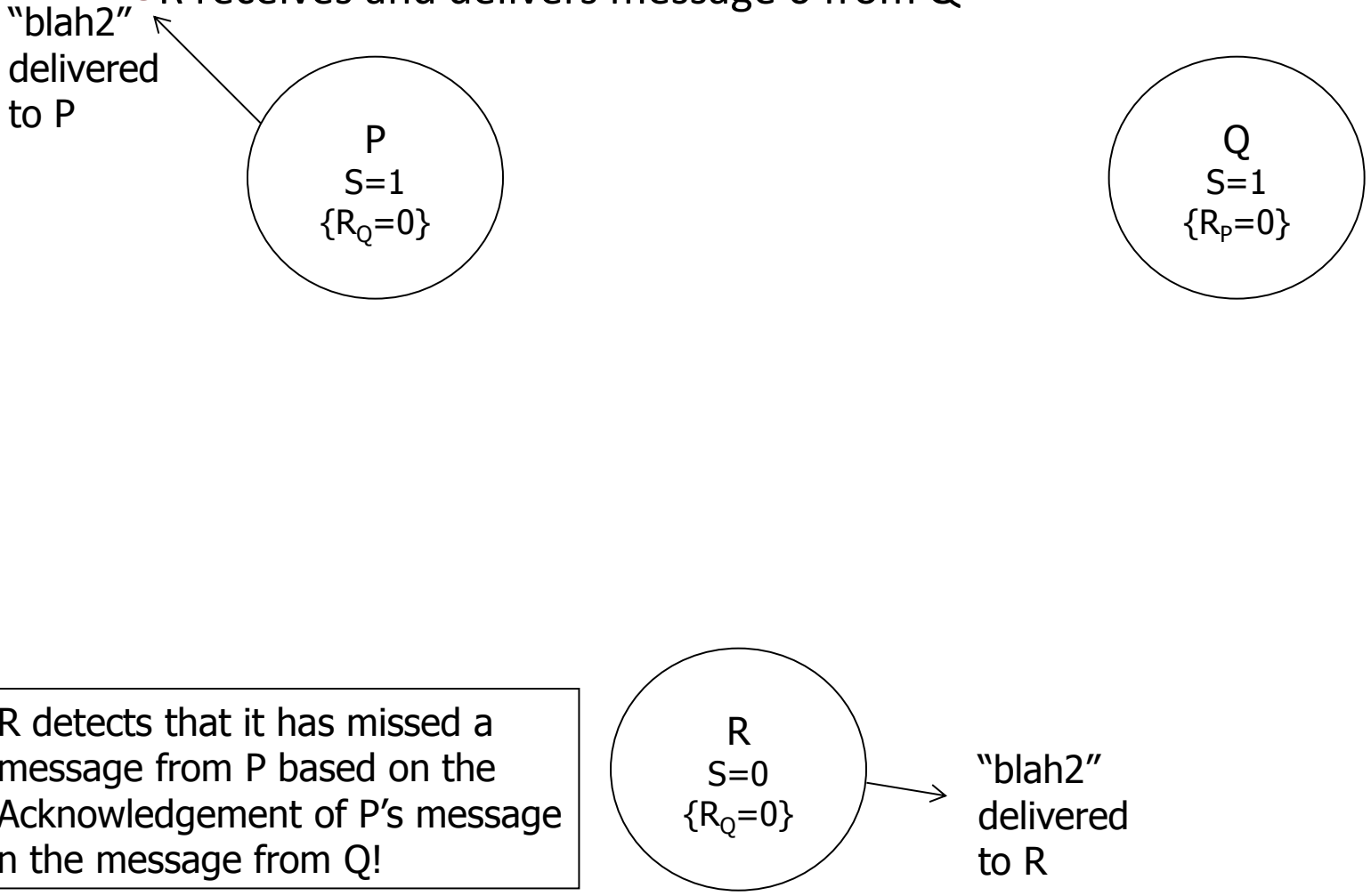
Reliable Delivery(4)

- Q sends a message to the group



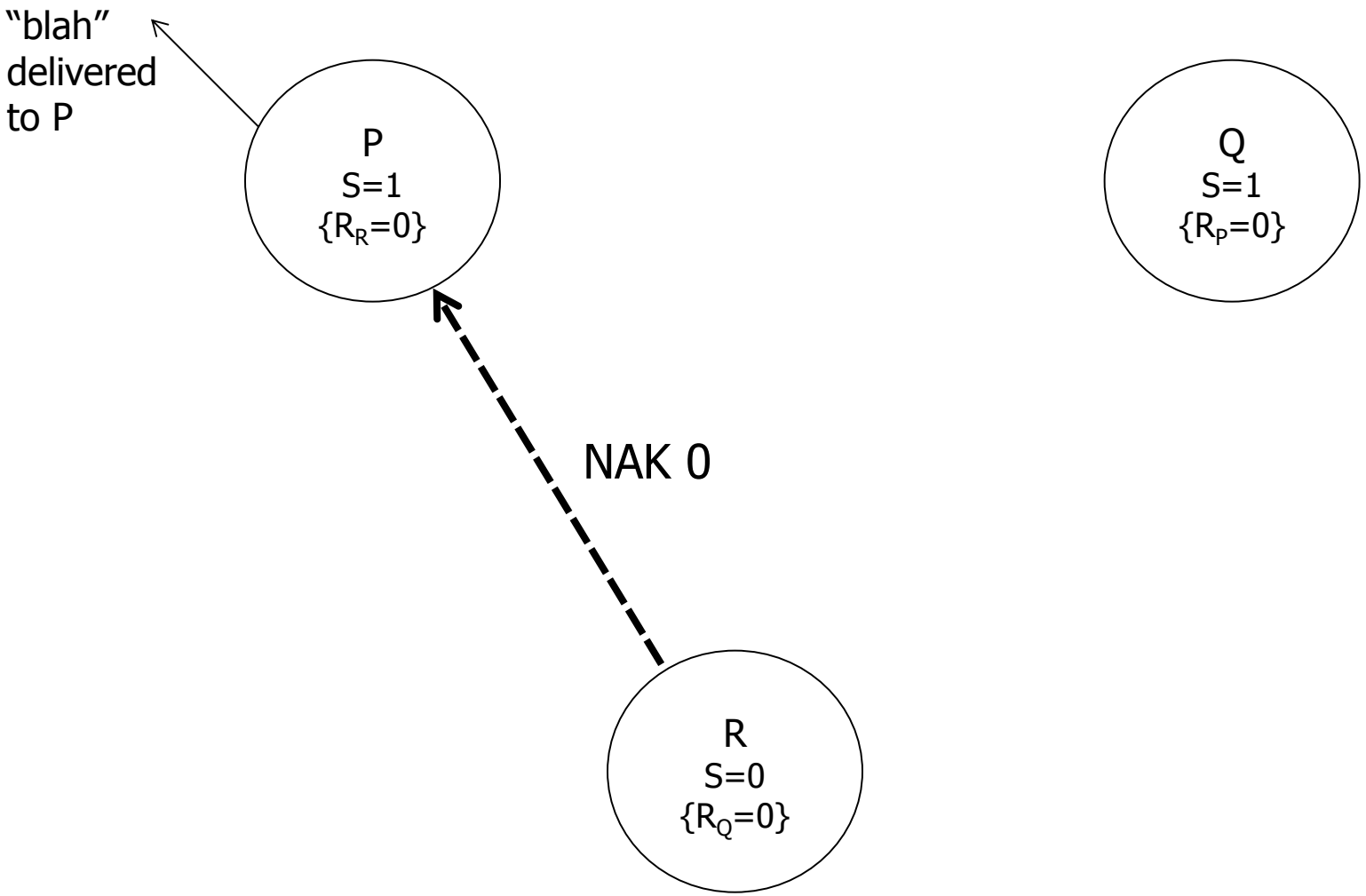
Reliable Delivery(5)

- P receives and delivers message 0 from Q
- R receives and delivers message 0 from Q



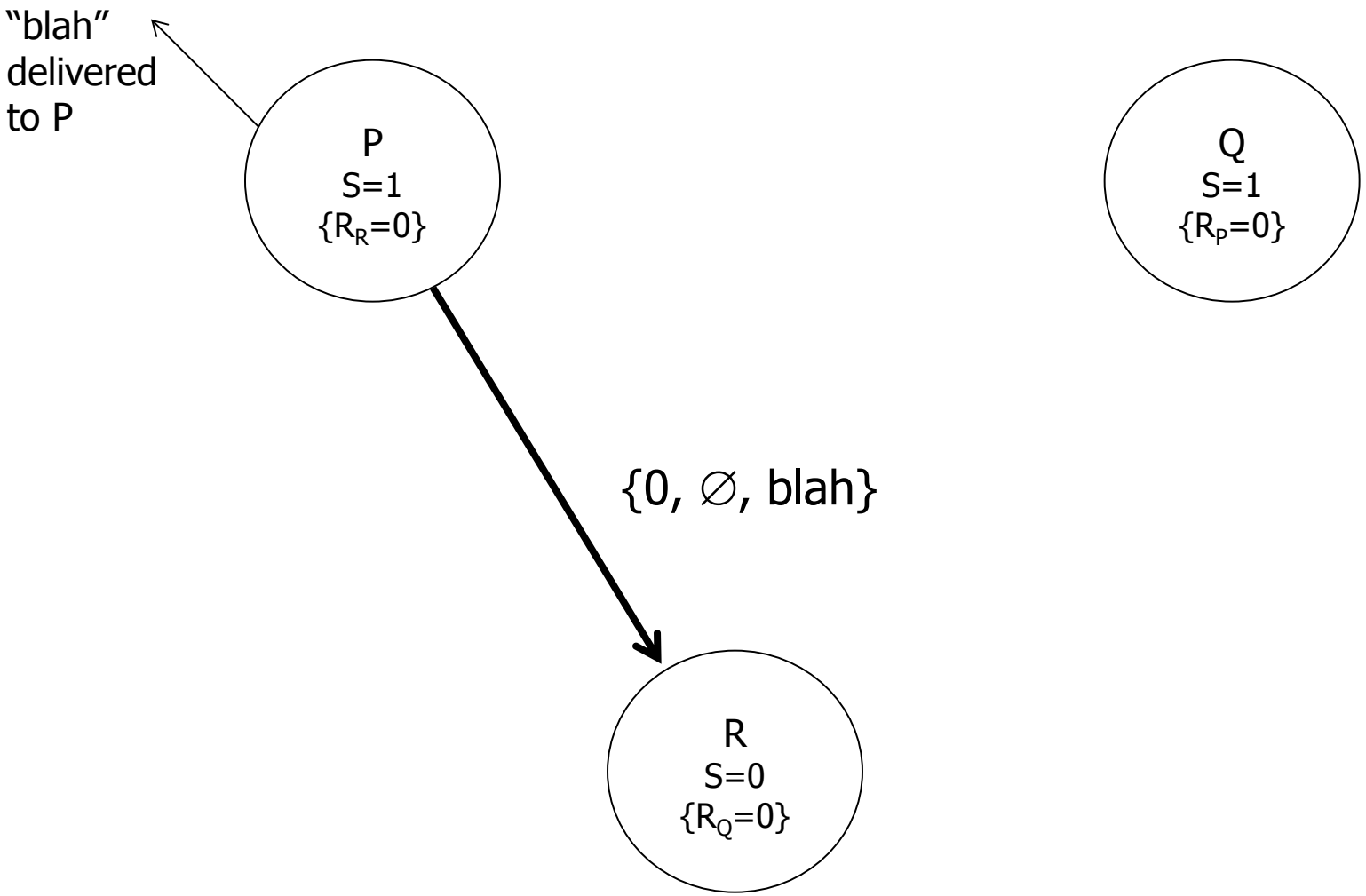
Reliable Delivery(6)

- R sends a NAK to P



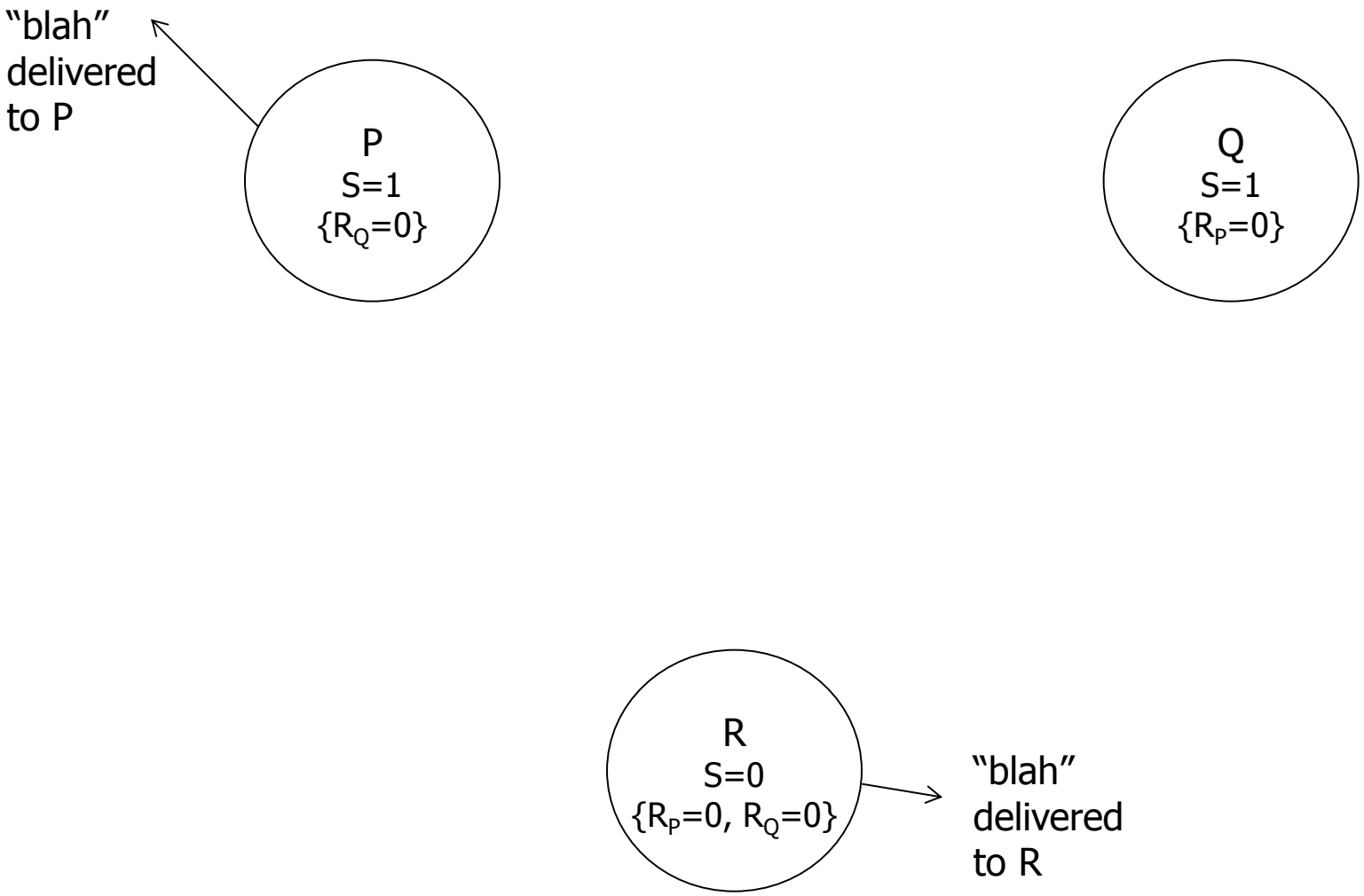
Reliable Delivery(7)

- P retransmits the missing message to R



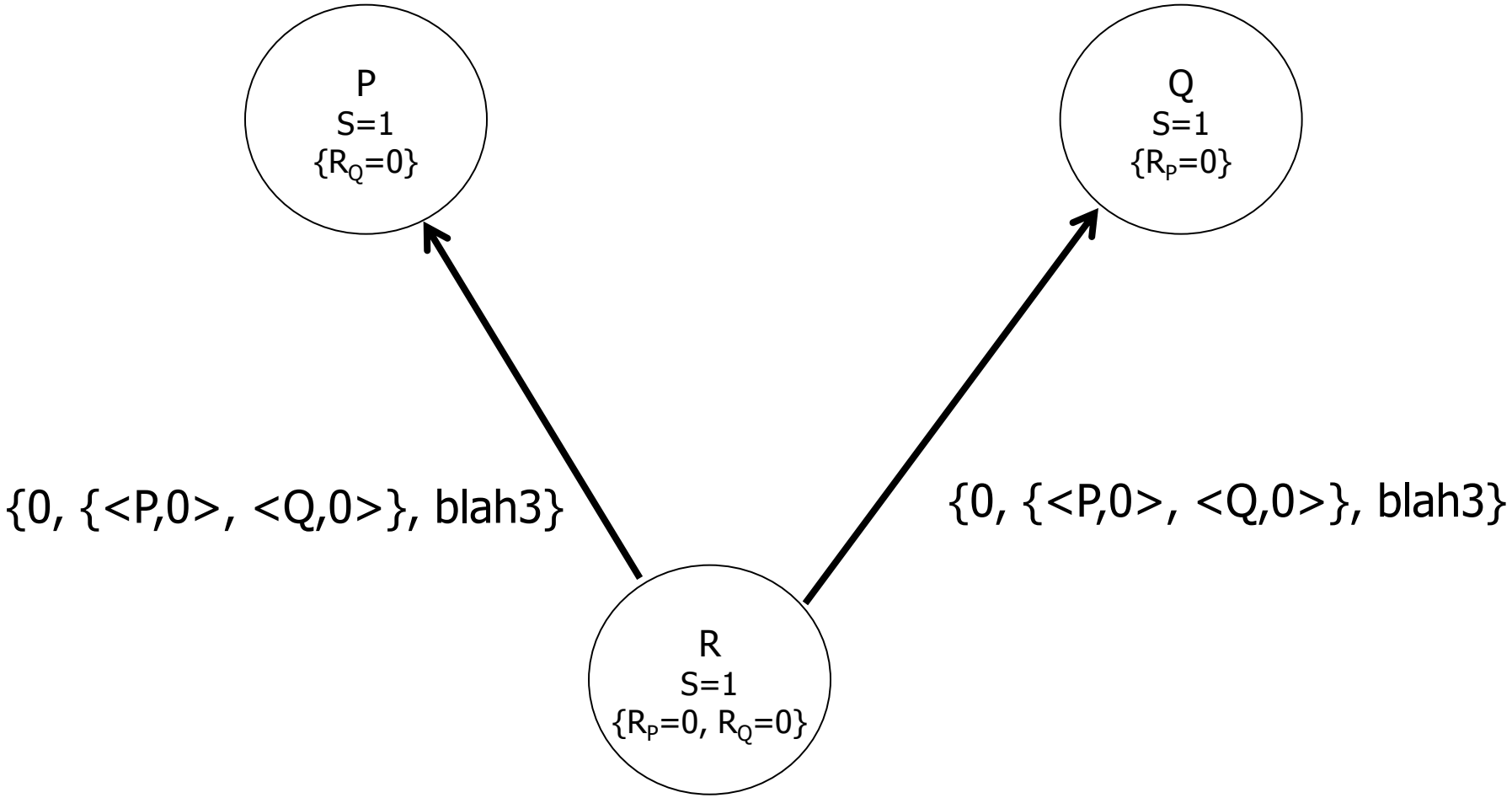
Reliable Delivery(8)

- R receives and delivers the message



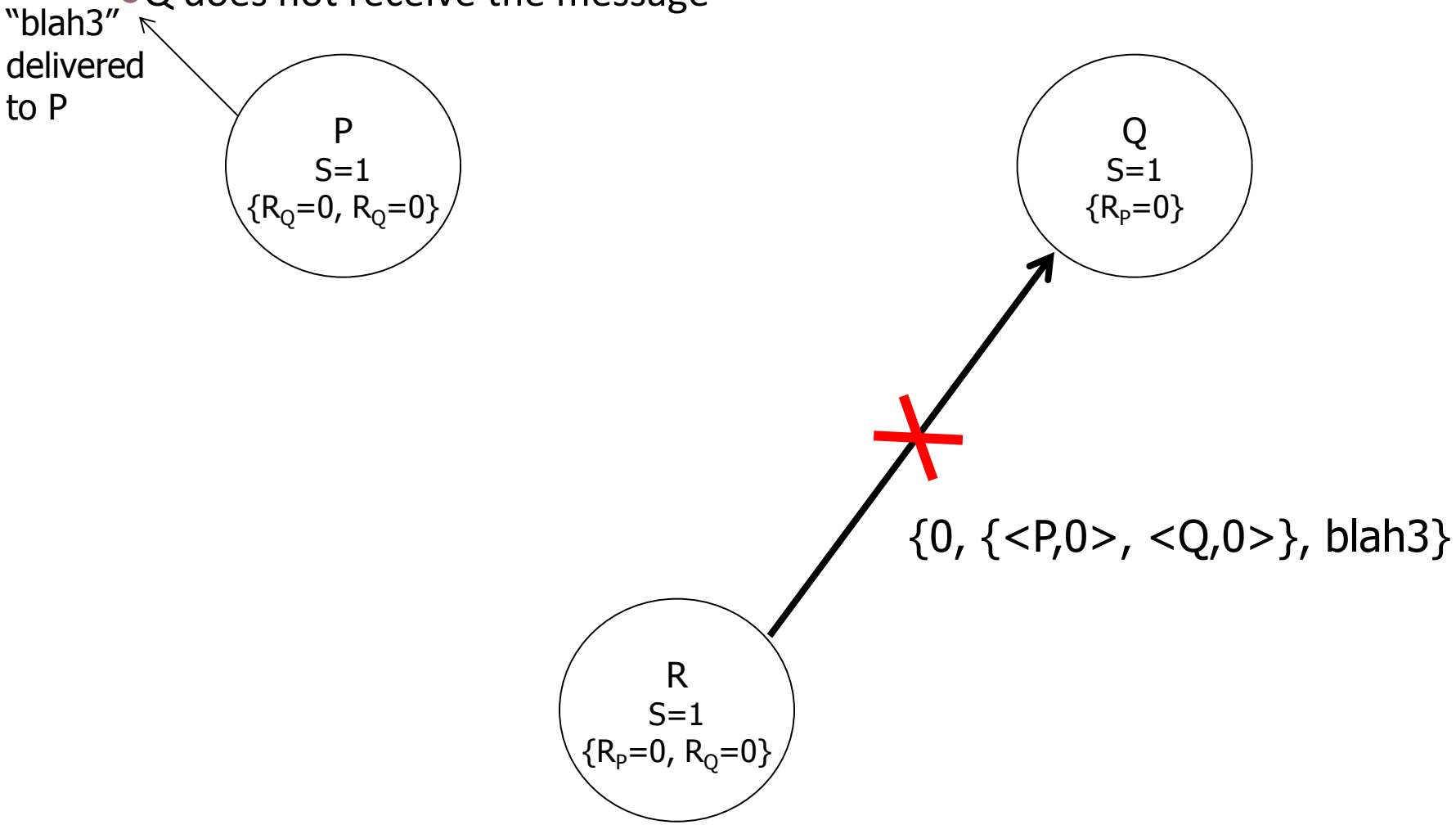
Reliable Delivery(9)

- R multicasts a message to the group



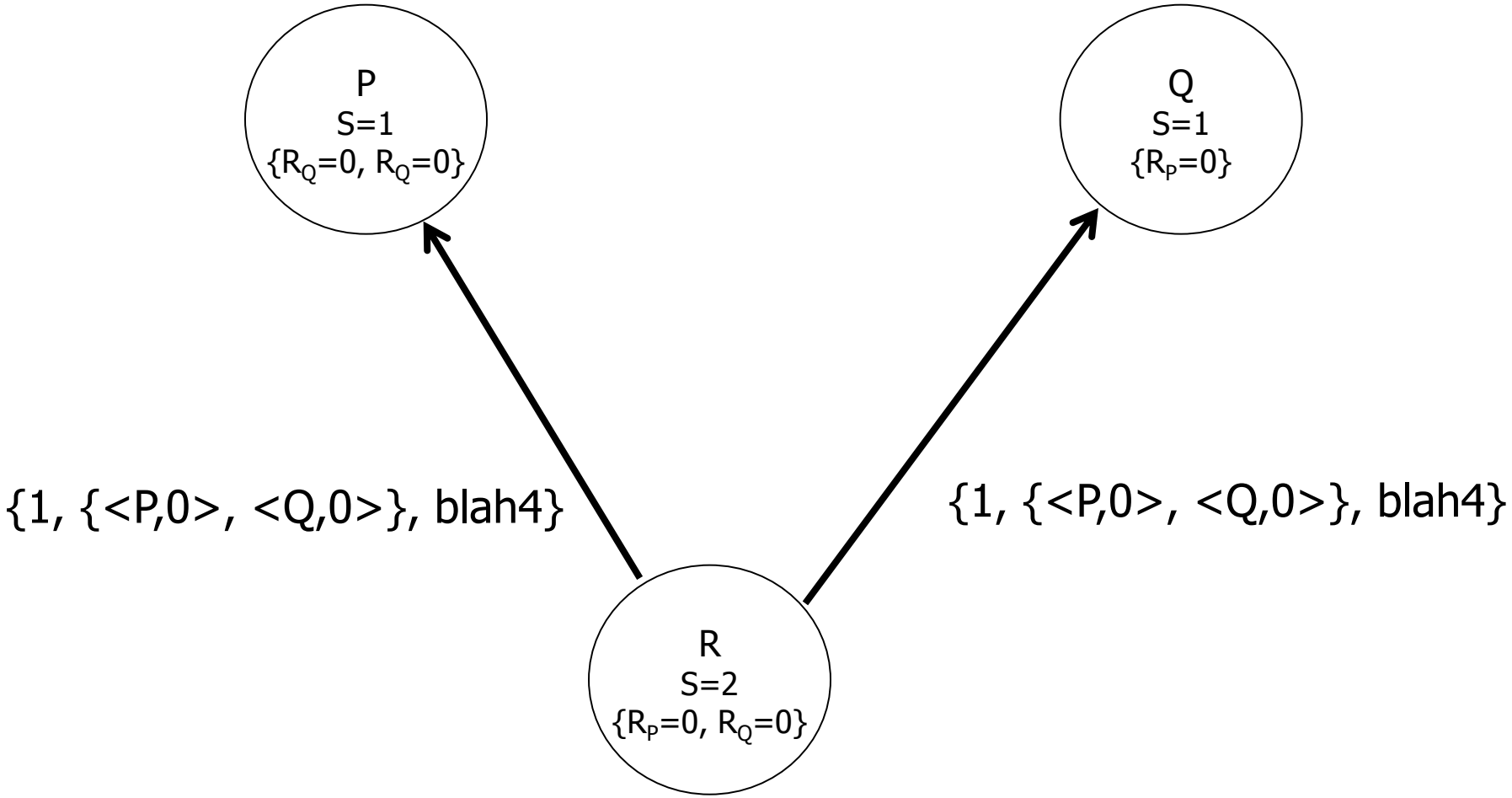
Reliable Delivery(10)

- P receives and delivers the message
- Q does not receive the message



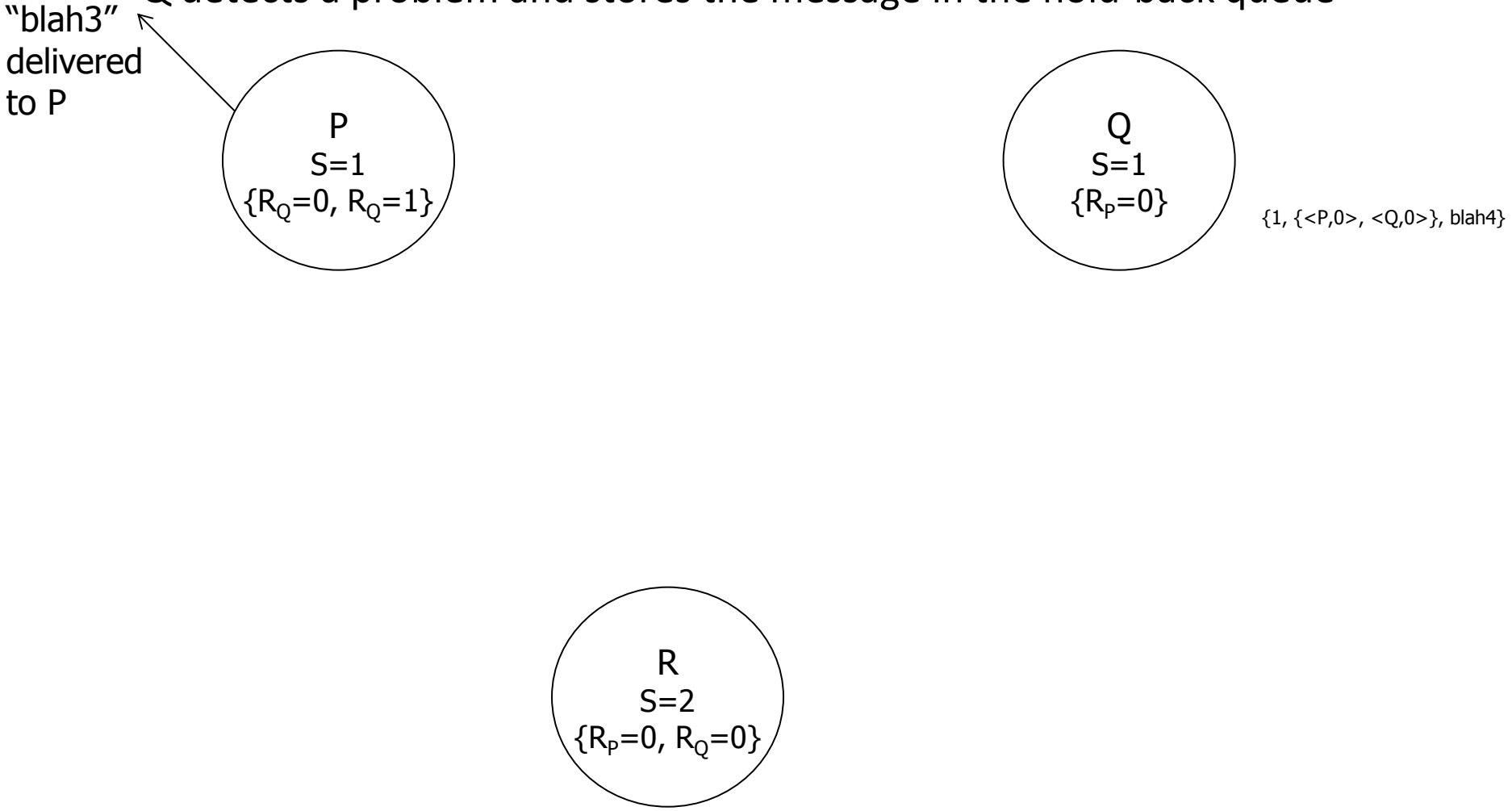
Reliable Delivery(11)

- R multicasts another message to the group



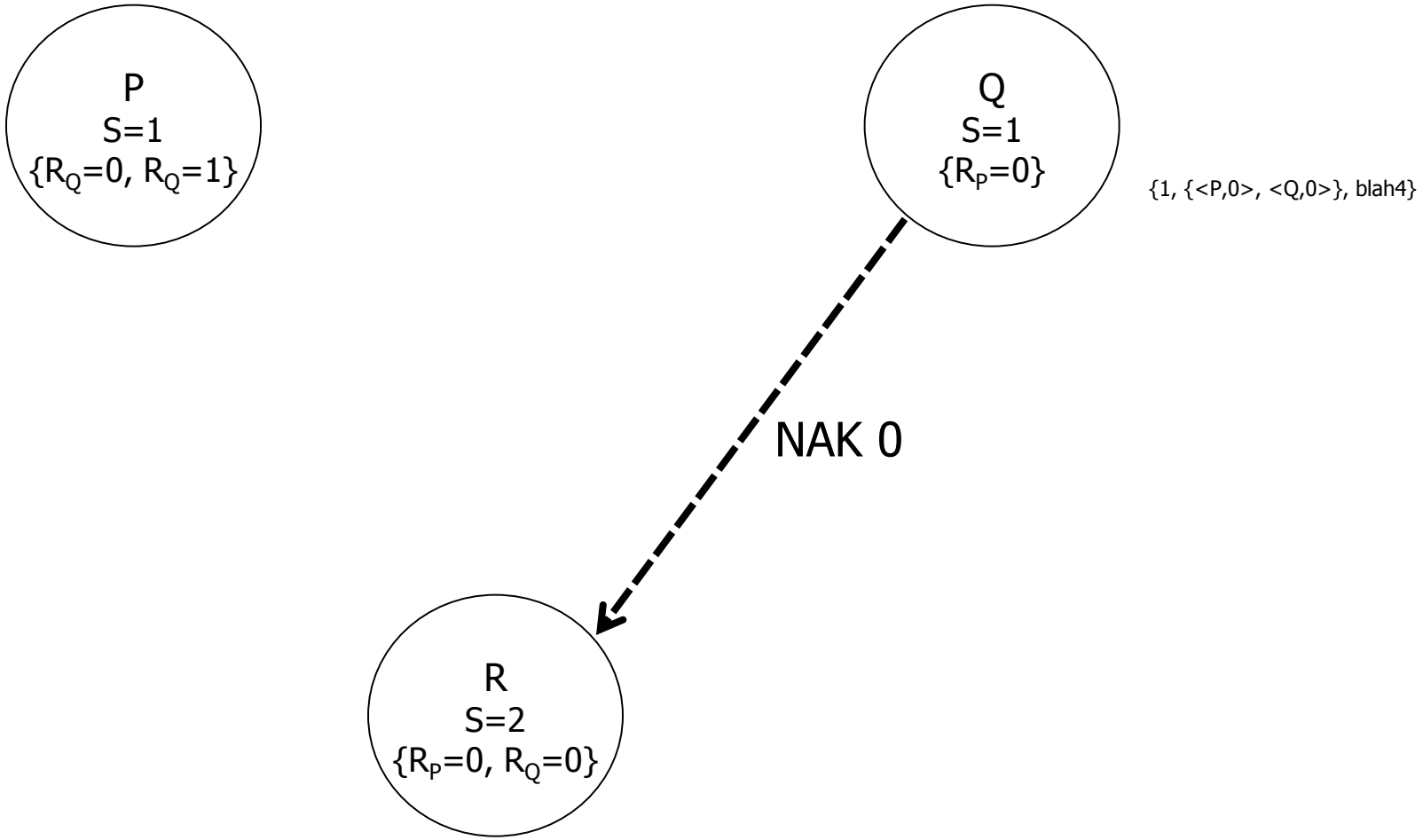
Reliable Delivery(12)

- P receives and delivers message 1 from R
- Q detects a problem and stores the message in the hold-back queue



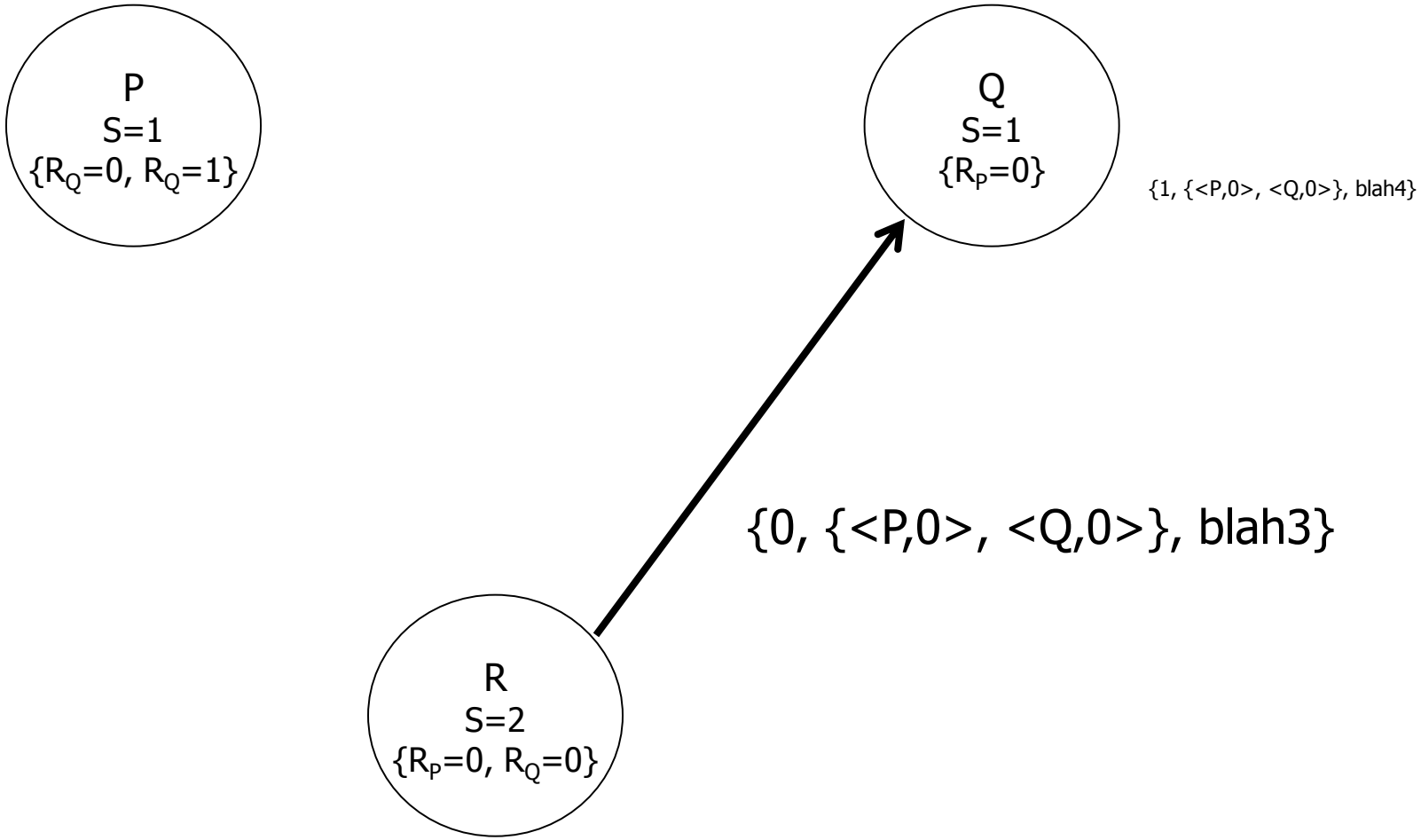
Reliable Delivery(13)

- Q sends a NAK to R



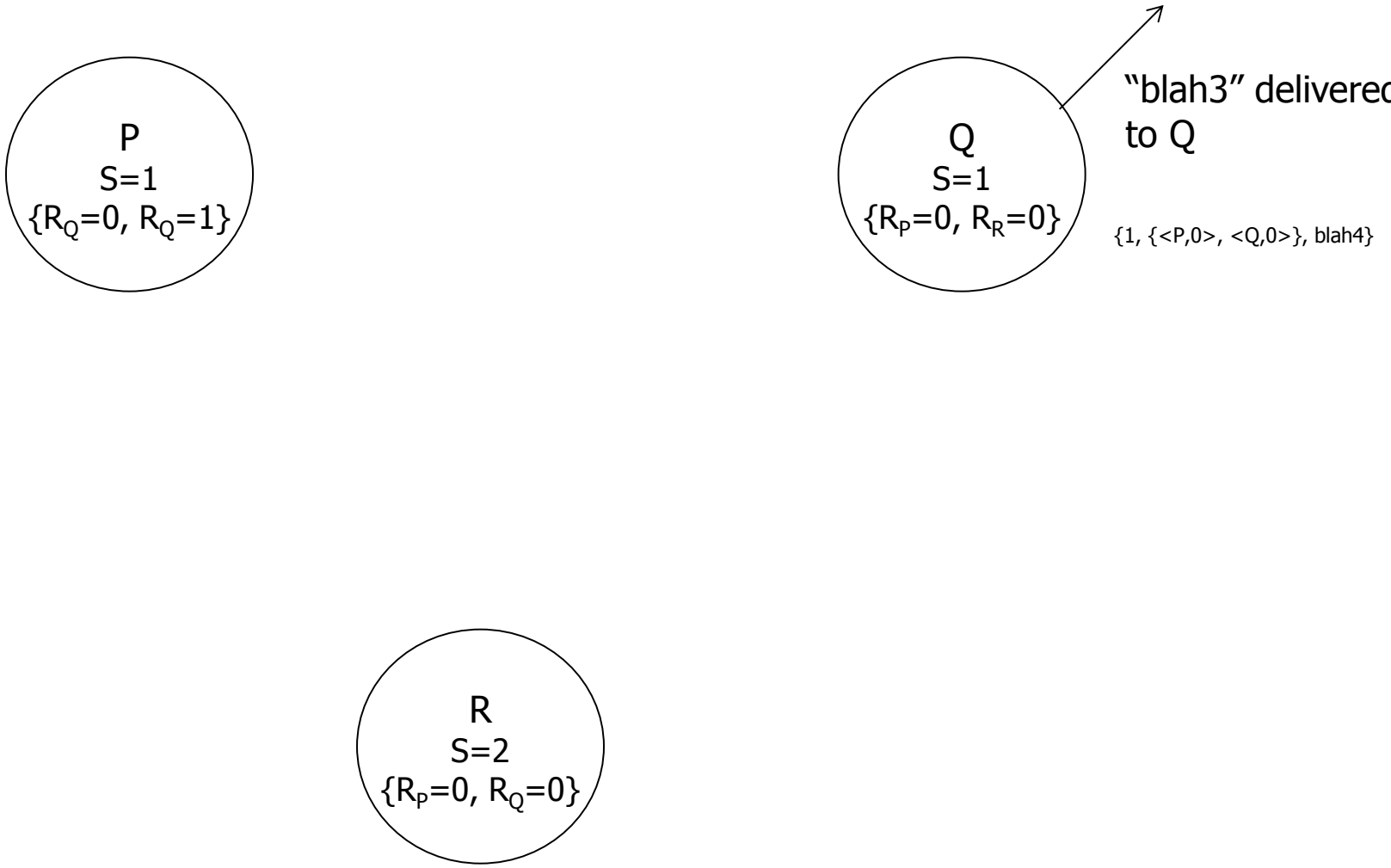
Reliable Delivery(14)

- R retransmits the message



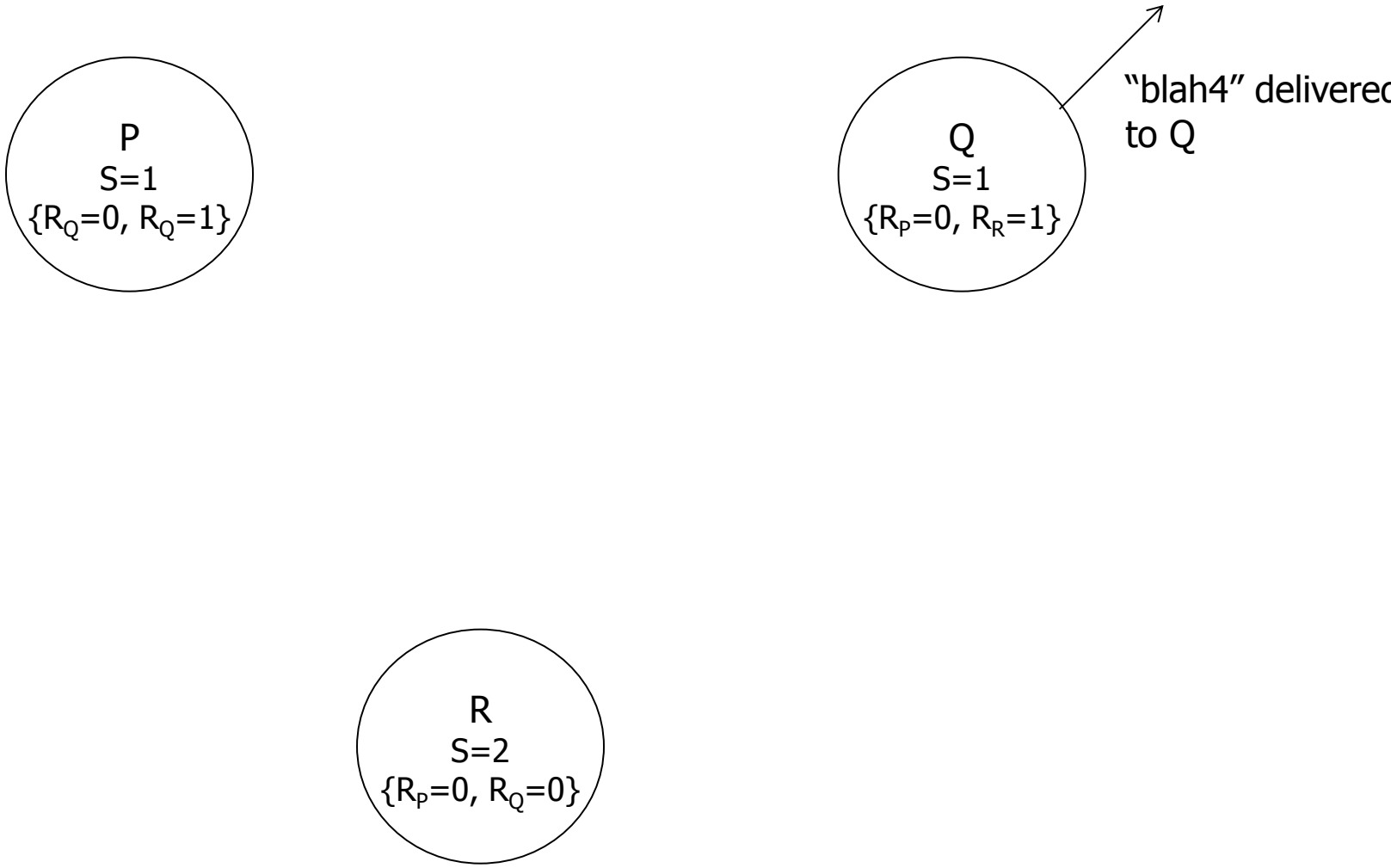
Reliable Delivery(15)

- Q receives and delivers message 0 from R



Reliable Delivery(16)

- Q removes message 1 from R from the hold-back queue and delivers it

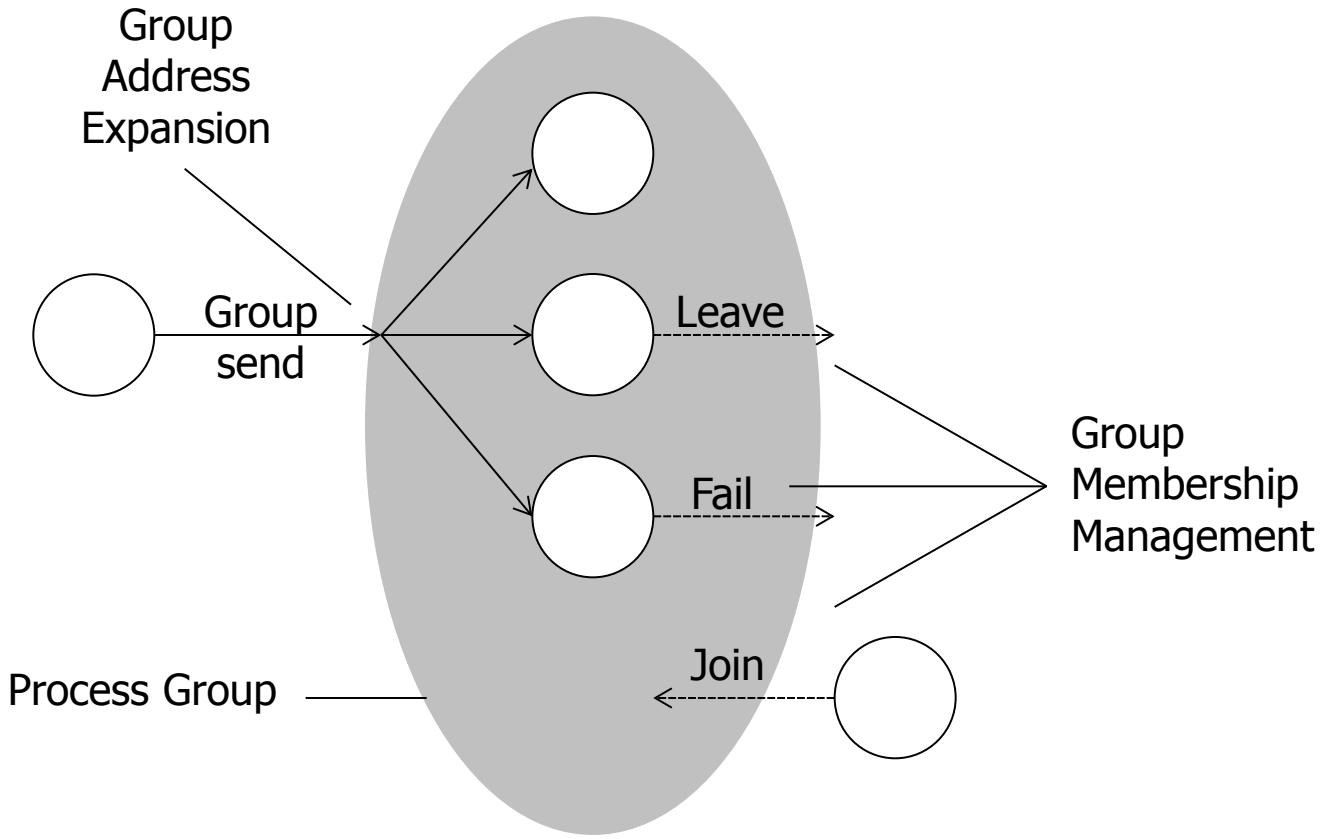


Group Membership Service

- A **Group Membership Service (GMS)** is a service that provides support for the (dynamic) management of both group membership and multicast communication.

- This service has four main objectives:
- Providing an interface for group Membership Change.
  - Creating/Destroying groups
- Implementing a Failure Detector.
  - Monitors the group members for both crashes & communication failures
  - Detector marks services as **Suspected** or **Unsuspected**.
  - This is used to by the service to determine who is in the group.
- Notifying members of changes in membership.
  - Informs members of the addition or exclusion of a group member
- Performing group Address Expansion.
  - Group identifier is used to associate the message with a set of group member addresses to whom incoming multicast messages must be sent.

Group Membership Service



Group Membership Service

- IP Multicasting implements a partial GMS:
  - Supports dynamic joining/leaving of groups
  - Supports address expansion
  - It does not support failure monitoring of notification of group changes.

- Support for failure monitoring and group changes is central to the implementation of fault tolerant systems.
  - They must be able to adapt how they operate to cater for changes in the make up of the community.
    - E.g. differing sets of capabilities, introduction of new capabilities, …
  - To achieve this, each member maintains a local view of the membership.
    - This view is known as a **group view**.

Group Views

- A **group view** is an ordered list of the current group members, identified by their unique process identifiers.
  - E.g. based on the order in which processes join the group
- Views are **delivered** whenever a membership change occurs.
  - E.g. a new member is added, a member leaves, …
- View delivery involves notifying the application of the new membership.
  - For example: consider an initially empty group, g
    - When a process, p, joins the group, a first group view, $v_0(g) = \{p\}$ is generated
    - Soon afterwards, a second process, p', joins the group, resulting in view $v_1(g) = \{p, p'\}$
    - Later, p' leaves the group, resulting in the view $v_2(g) = \{p\}$
  - These views are generated by the GMS as each change occurs, and are transmitted to all current group members.

View Delivery

- Views delivery is similar to multicast message delivery:
  - The GMS transmits views as membership changes occur.
  - Views are distinguished by a unique identifier that defines an ordering on them (i.e. integer value).
  - Each member keeps track of the next view to be displayed.
  - If a member receives a view that is later than the view it expected to receive then the view is stored in a hold-back queue until appropriate.

- View Delivery Implementations must satisfy three requirements:
  - **Order**:
    - If process p delivers view v(g) and then view v'(g), then no other process $q \neq p$ delivers v'(g) before v(g).
  - **Integrity**:
    - If process p delivers view v(g) then $p \in v(g)$.
  - **Non-triviality**:
    - If process q joins a group and is or becomes indefinitely reachable from process $p \neq q$, then eventually q is always in the views that p delivers.

View-Synchronous Group Com.

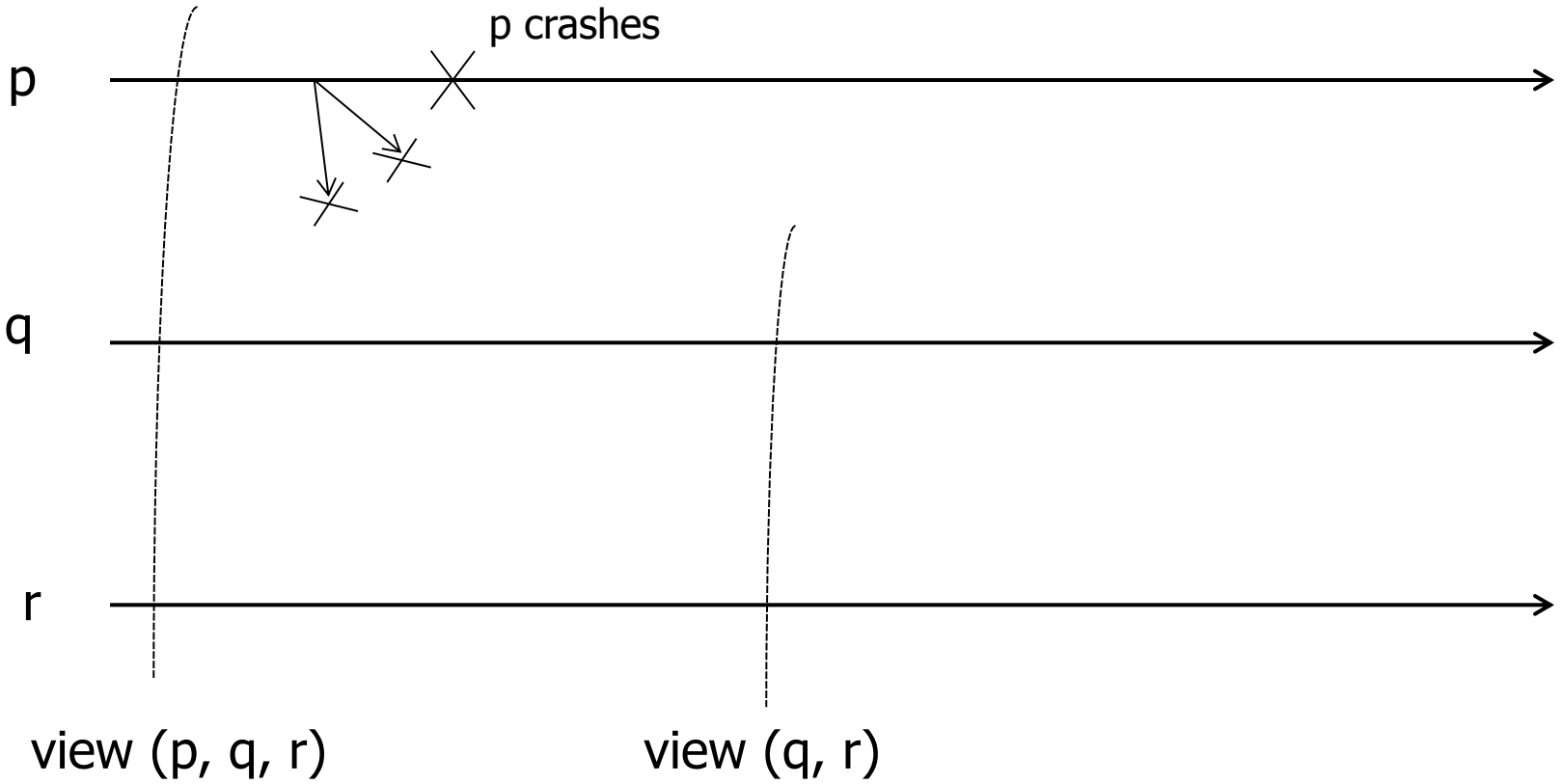
- Group communication can be further enhanced through the use of group views to constrain whether a received message can/should be delivered.

- The delivery of a new view draws a conceptual line across the system:
  - Every message that is delivered must be consistently delivered on one side or the other of that line.

- Consider a group g containing (at least) processes p and q
  - Process q sends a message m to the group and then crashes
  - When process p receives the message from process q, p will only deliver m if $q \in v(g)$
  - So, m is only delivered to p if the message is received by p before it receives and delivers a new view from the GMS.

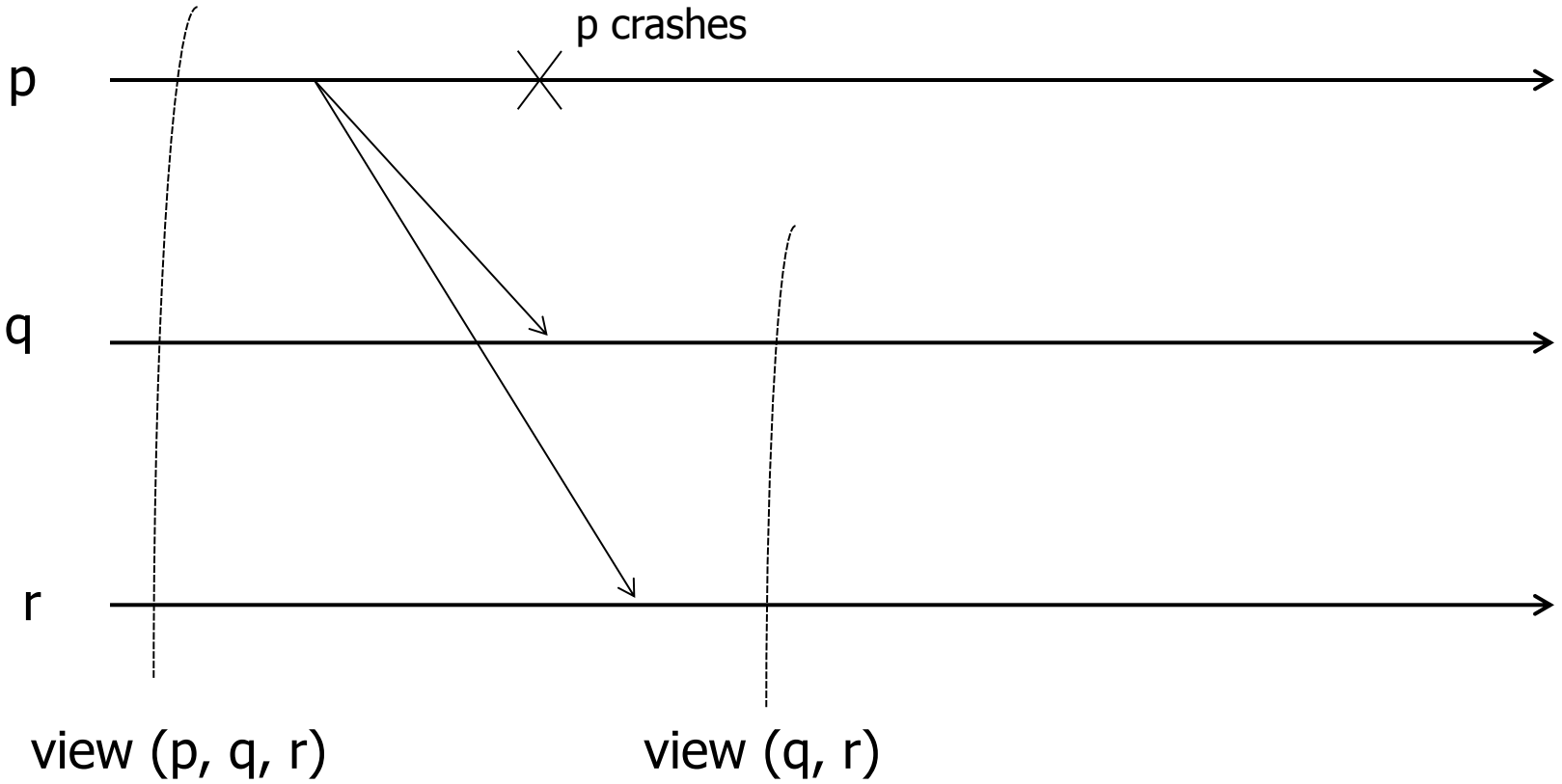
View-Synchronous Group Com.

- p sends a message m, but crashes soon after sending m
- Option A: m does not reach q or r (ALLOWED)



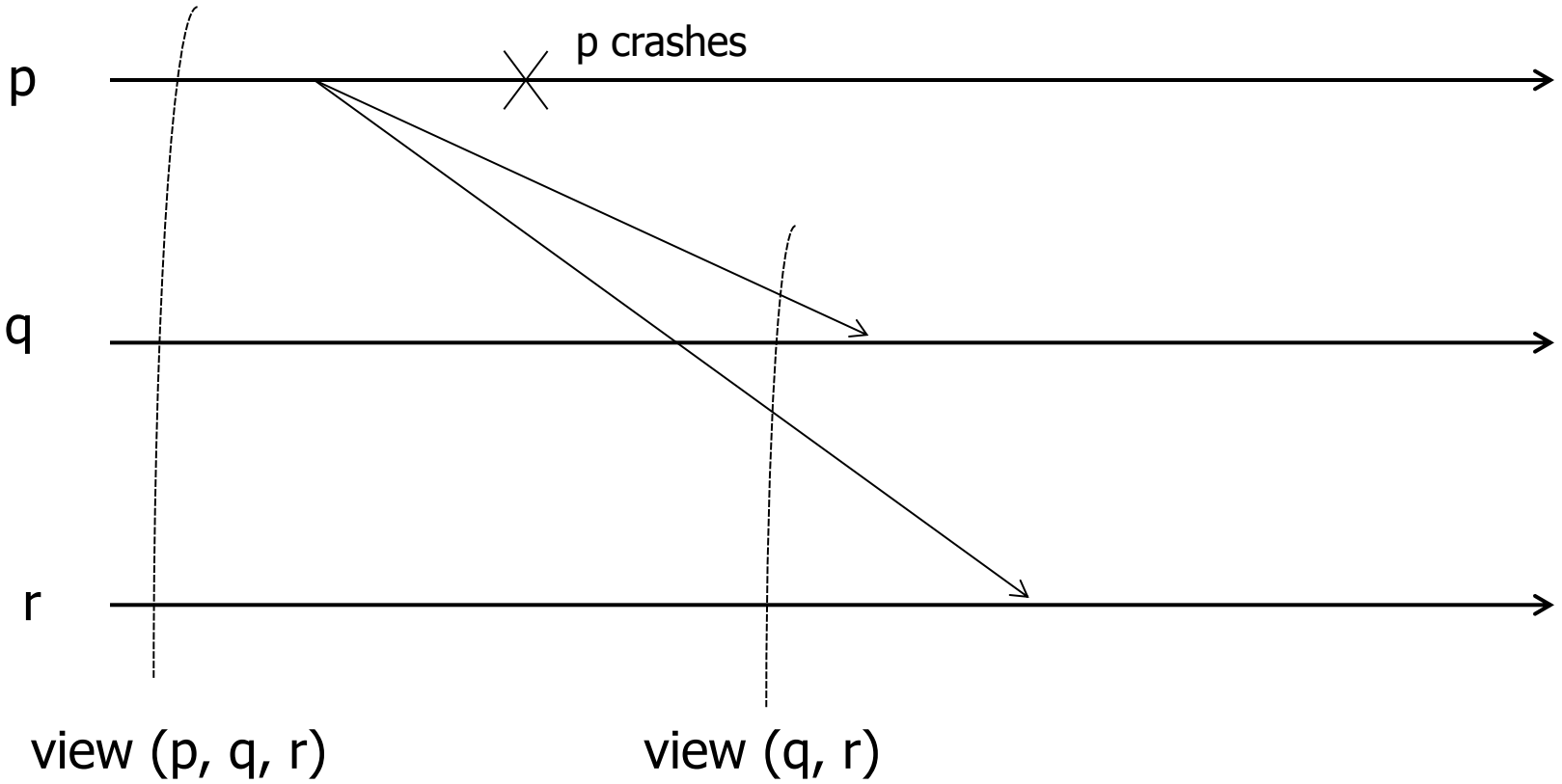
View-Synchronous Group Com.(2)

- p sends a message m, but crashes soon after sending m
- Option B: m reaches either q or r before p crashes (ALLOWED)



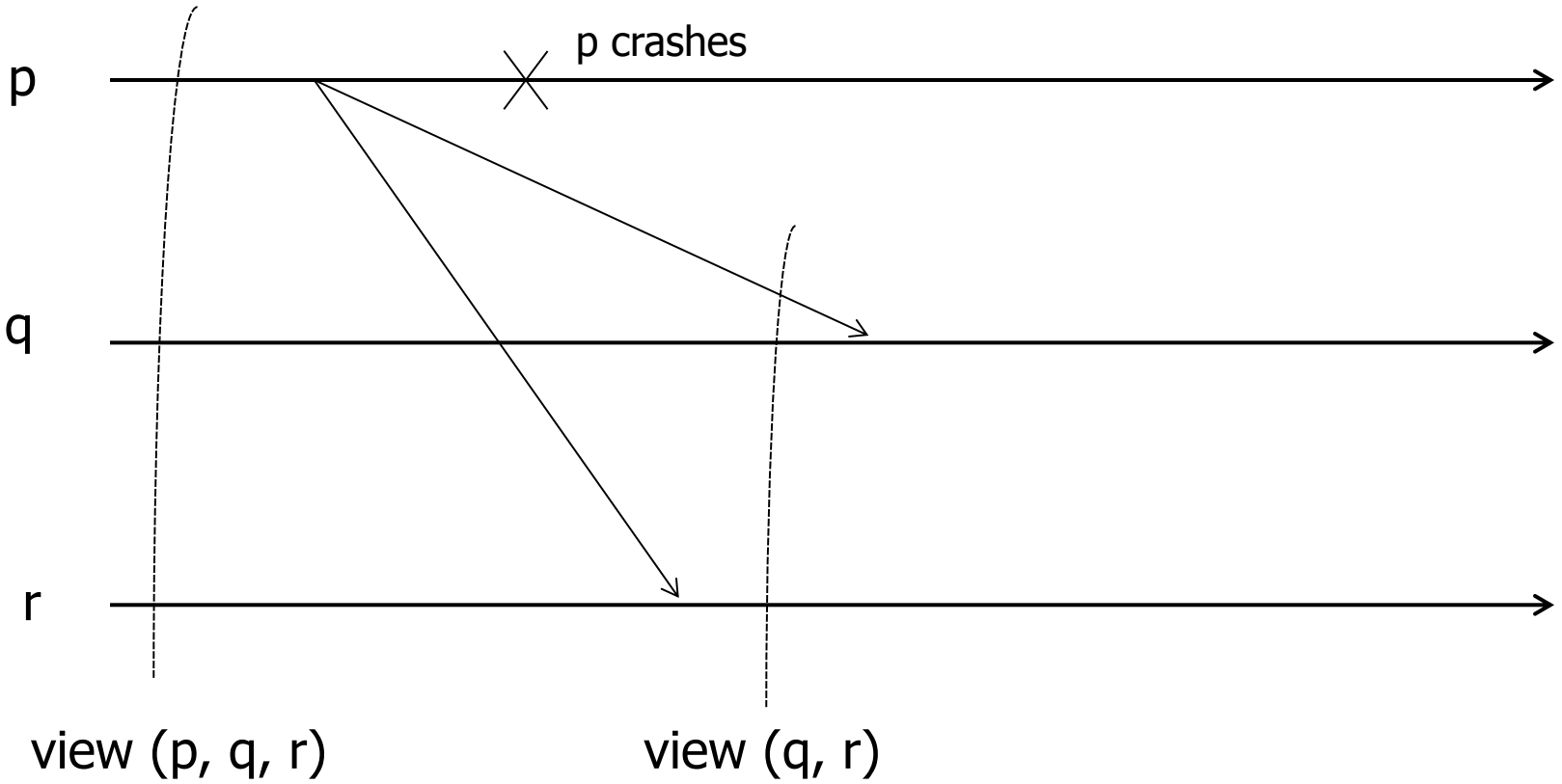
View-Synchronous Group Com.(3)

- p sends a message m, but crashes soon after sending m
- Option C: m reaches either q or r after p crashes and after a new group view is delivered to q and r (DISALLOWED)



View-Synchronous Group Com. (4)

- p sends a message m, but crashes soon after sending m
- Option D: m reaches r before the new view is delivered but reaches q after the new view is delivered (DISALLOWED)



View-Synchronous Group Com.

- T he additional guarantees that View-Synchronous Group Communication provides are:
- **Agreement**:
  - Correct processes deliver the same sequence of views, and the same set of messages in any given view.
- **Integrity**:
  - If a correct process p delivers message m, then it will not deliver m again.
  - Furthermore, p $\in$ group(m) and the process that sent m is in the view in which p delivers m.
- **Validity**:
  - Correct processes always deliver the messages that they send.
  - If the system fails to deliver a message to any process q, then it notifies the surviving processes by delivering a new view with q excluded.
  - This view is delivered immediately after the view in which any of them delivered the message.

Summary

- Group/Multicast Communication is concerned with sending messages to groups of processes.
  - Static and Dynamic Groups
  - Improved Efficiency

- Techniques:
  - Basic Multicast
    - Requirements: Guaranteed Delivery of Messages
    - Unicast Implementation
    - ACK-implosion
  - Reliable Multicast
    - Requirements: Integrity, Validity, Agreement
    - Multicast Implementation
    - Piggy-backing
    - Hold-back Queues

Summary

- Group Membership Service (Dynamic Groups):
  - Requirements:
    - Provide an interface for group membership change
    - Implement a failure detector
    - Notifying members of changes in membership
    - Performing group address expansion
- Group Views & View Delivery
  - View-Synchronous Group Communication

Distributed Systems:
Active and Passive Replication

Consistency

- In replication systems, consistency refers to the correctness of the replicas within the system.

- Inconsistencies between replicas can cause errors in the operation of the system.
  - Delayed updates can cause the system to use incorrect data
    - E.g. when we deposit money into a bank account, we expect that the balance on our account will be updated globally.

- Strict consistency requires that:
  - Any read on a data item ‘x’ returns a value corresponding to the result of the most recent write on ‘x’ (regardless of where the write occurred).
  - This is often termed **linearisation**.

Linearisability

- A **linearisable system** is a system in which all the operations appear to have been performed in some sequential and non-overlapping order.
  - That is, the set of operations that are performed by the system can be written down sequentially (even if carried our by more than one process)
    - E.g. boil the water, get the cup, add the coffee, pour the water, …

- A replication system is said to be linearisable if, for any execution of the system, there is some interleaving of the series of operations issued by all the clients that satisfies that following criteria:
  - The interleaved sequence of operations meets the specification of a (single) correct copy of the objects.
  - The order of the operations in the interleaving is consistent with the real time at which the operations occurred in the actual execution.

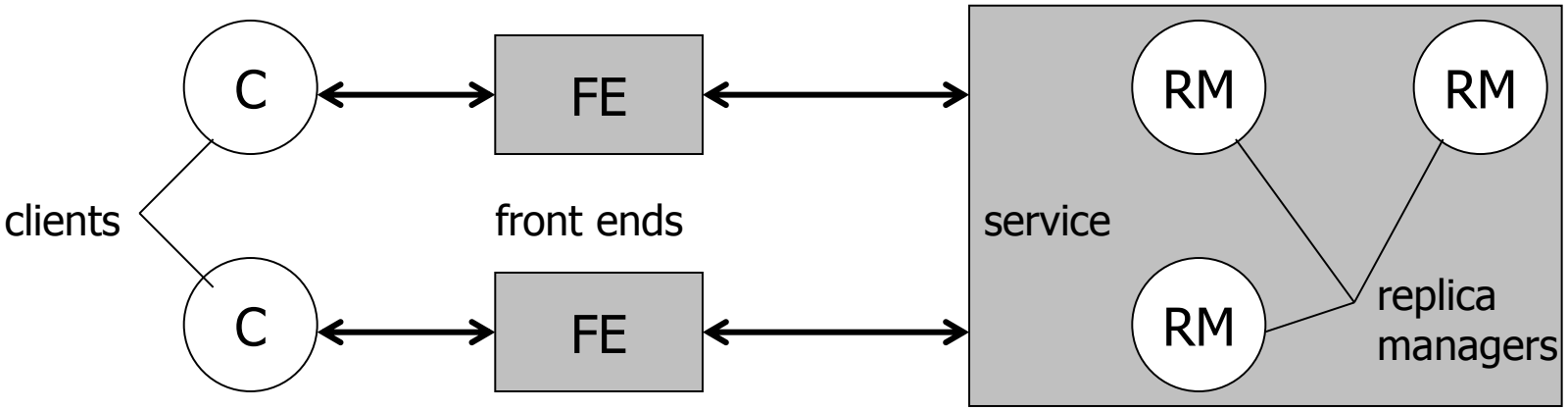
Sequential Consistency

- A weaker correctness condition for Replication Systems is **sequential consistency**:
  - The interleaved sequence of operations meets the specification of a (single) correct copy of the objects.
  - The order of operations in the interleaving is consistent with the program order in which each individual client executed them.

- The difference between Sequential Consistency and Linearisability is that:
  - Absolute time and total ordering of operations is not required.
  - Instead, ordering is relative to the order of events at each separate client.

- As a result, Linearisation requires that the ordering of events conform to the real-world, while Sequential Consistency requires only that the operations be ordered relative to each process.

Recap: Replication Systems



Recap: Replication Systems

- Handling a request to perform an operation on a logical object normally involves 5 steps:
  - **Request**: FE issues the request to one or more RMs
    - Single message or Multicast
  - **Coordination**: RMs coordinate to execute the request consistently.
    - They agree on whether or not the request is to be applied
    - They decide on the ordering of the request relative to others
    - FIFO, Causal, Total, …
  - **Execution**: The RMs execute the request
    - This may be done tentatively (i.e. it can be undone later)
  - **Agreement**: The RMs reach consensus on the effect of the request
  - **Response**: One or more RMs respond to the FE
    - Single/Multiple Responses
    - Response Acceptance/Synthesis (where necessary)

Passive Replication

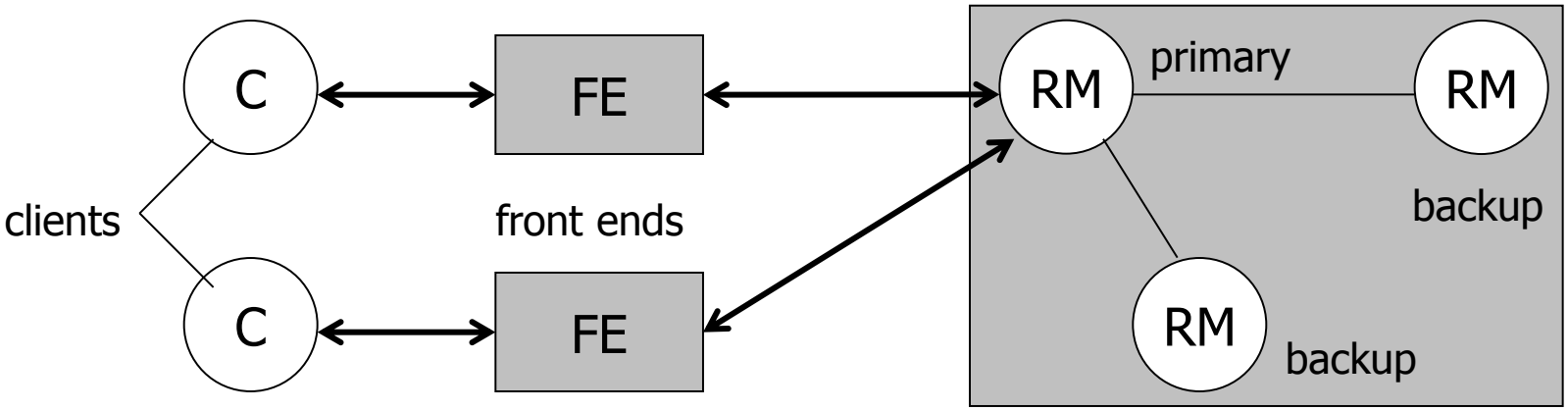
- In the passive or primary-backup model of replication, there is, at any one time:
  - a single primary replica manager, and
  - one or more secondary (backup) replica managers

- In its simplest form, the FEs communicate only with the primary RM to obtain the service.
  - The primary RM executes the operations and sends copies of the updated data to the backups.

- If the primary fails, then one of the backups is promoted to act as the primary
  - i.e. some election algorithm is employed…

- Because the primary RM sequences all operations upon the shared objects, passive replication systems are linearizable!

Passive Replication



Passive Replication

- When a FE issues a request, the following steps are executed:
- **Request**: The FE issues a request, containing a unique identifier, to the primary RM.
- **Coordination**: The primary takes each request atomically, in the order in which it receives it.
  - It checks the unique identifier, in case it has already executed the request and if so, it simply resends the response.
- **Execution**: The primary executes the request and stores the response.
- **Agreement**: If the request is an update then the primary sends the updated state, the response, and the unique identifier to the backups.
  - The backups send an ACKnowledgement.
- **Response**: The primary responds to the FE, which hands the response back to the client.

Passive Replication

- The primary and backups are organized as a group, and the primary uses view-synchronization group communication to send updates to the backups.
- When a primary fails, it is replaced by a unique backup.
- The GMS will eventually deliver a new group view to one of the backups.
  - This can be used to kick off an election algorithm.
- The RMs that survive agree on which operations had been performed at the point where the replacement primary takes over.
  - View-synchronization group communication ensures that either all the backups or none of them will deliver any given update before delivering the new view.

Passive Replication

- To survive up to f process crashes, a passive replication system requires f+1 replica managers.

- The front end requires little functionality:
  - It needs only to be able to locate the new primary when the current primary does not respond.

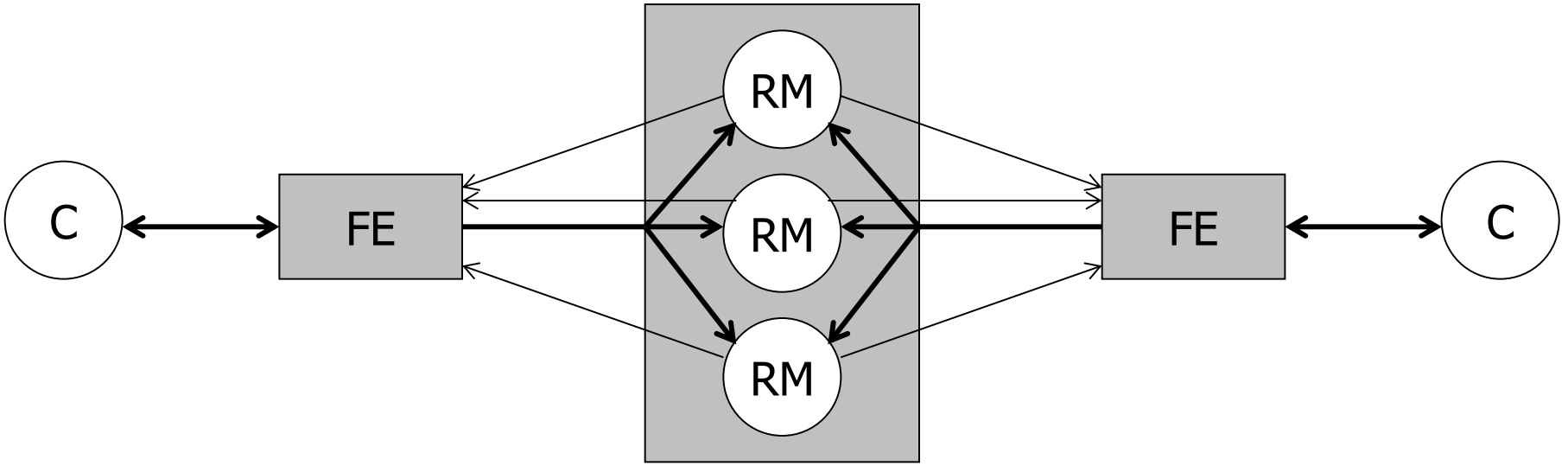
- Large overheads due to view-synchronization group communication

- Variant implementations allow clients to submit reads to the backups:
  - This reduces the load on the primary.
  - But we lose linearizability, and instead get only sequential consistency.

Active Replication

- In the active replication model, replica managers are state machines that play equivalent roles and are organised as a group.

- Front Ends multicast their requests to the group and all the RMs process the requests independently, but identically, and reply.



Active Replication

- When a FE issues a request, the following steps are executed:
  - **Request**: The FE sends a multicast request to the replica manager group, containing a unique identifier.
    - The multicast is both totally-ordered and reliable.
    - The FE does not issue the next request until it has received a response.
  - **Coordination**: The group communication system delivers the request to every correct RM in the same (total) order.
  - **Execution**: The RM executes the request.
    - Since the RMs are state machines and all requests are delivered in the same order, correct RMs all process the request identically.
    - The response contains the clients unique request identifier.
  - **Agreement**: No agreement phase is needed because of the multicast delivery semantics.
  - **Response**: Each RM sends its response to the FE.
    - The number of replies that the FE collects before sending a response depends upon both the failure assumptions and the multicast algorithm.

Active Replication

- The system is sequentially consistent:
  - All correct RMs process the same sequence of requests.
  - The reliability of multicast ensures that every correct RM processes the same set of requests and the total order ensures that they process them in the same order.
  - The FE requests are served in FIFO order:
    - The FE waits for the response to the previous request before issuing the next request.

- When a process fails there is no impact on the performance of the service:
  - The remaining RMs continue to respond in the normal way.
  - The FE collects and compares the responses it gets before sending a response to the client.
    - This can be first response, majority response, …

Summary

- Consistency: How to keep replicas consistent.
  - Linearisability:
    - The sequence of interleaved operations that is totally ordered based on real time.
  - Sequential Consistency:
    - Any sequence of interleaved operations that satisfies the local ordering of operations carried out by the processes.

- Two types of replication:
  - **Passive Replication**:
    - Primary RM handles all requests and sends updates to the backup RMs.
    - Linearisable
  - **Active Replication**:
    - All RMs handle all requests concurrently.
    - Sequentially Consistent.

Next Lecture…
Case Study: The Gossip Architecture