Distributed Systems
Group Communication

Guest Lecturer: David Lillis (david.lillis@ucd.ie)

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 minimize 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 & the 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 confederate 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 process’s node.

Basic Multicast

- A primitive multicast protocol that guarantees the delivery of messages to all processes in a (static) group.
  - **B-multicast(g,m)** sends 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 an **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 process’s 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 ∈ 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 **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 that 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.

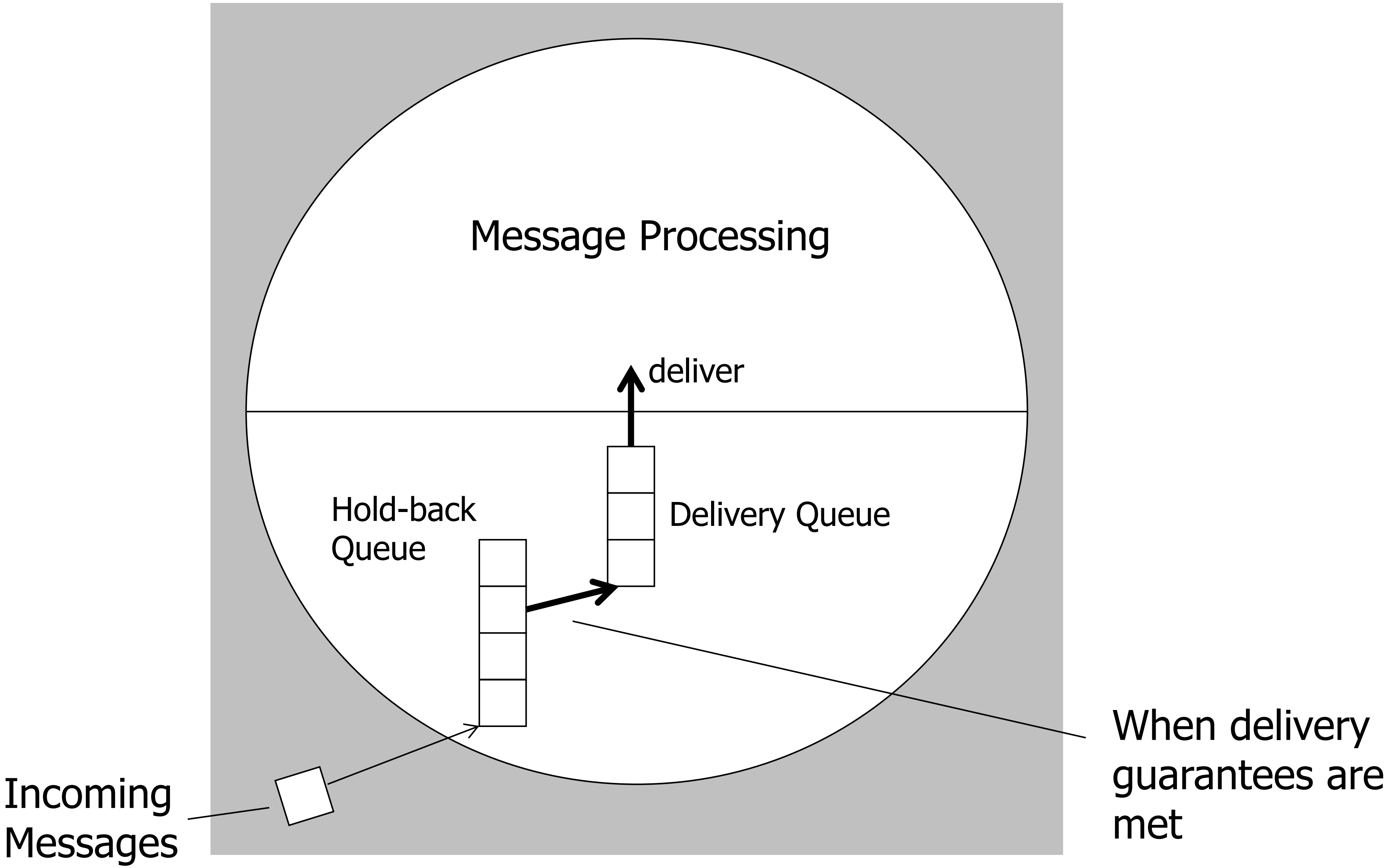
- To achieve this (for closed groups):
  - Each process **p** maintains a sequence number S^p_g (initially zero) for each group **g** that it belongs to. This records how many messages **p** has sent to **g**.
  - Each process also records S^q_g the sequence number of the latest message from process **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 $\langle q, R^q_g \rangle$
    - 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 $\mathbf{S} = \mathbf{R^p_g} + \mathbf{1}$.
        - It increments $\mathbf{R^p_g}$ by one immediately after delivering the message.
  - If an arriving message has $\mathbf{S} \leq \mathbf{R^p_g}$, then it discards the message because it has already delivered it
  - If $\mathbf{S} > \mathbf{R^p_g} + 1$ or $\mathbf{R} > \mathbf{R^q_g}$ 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

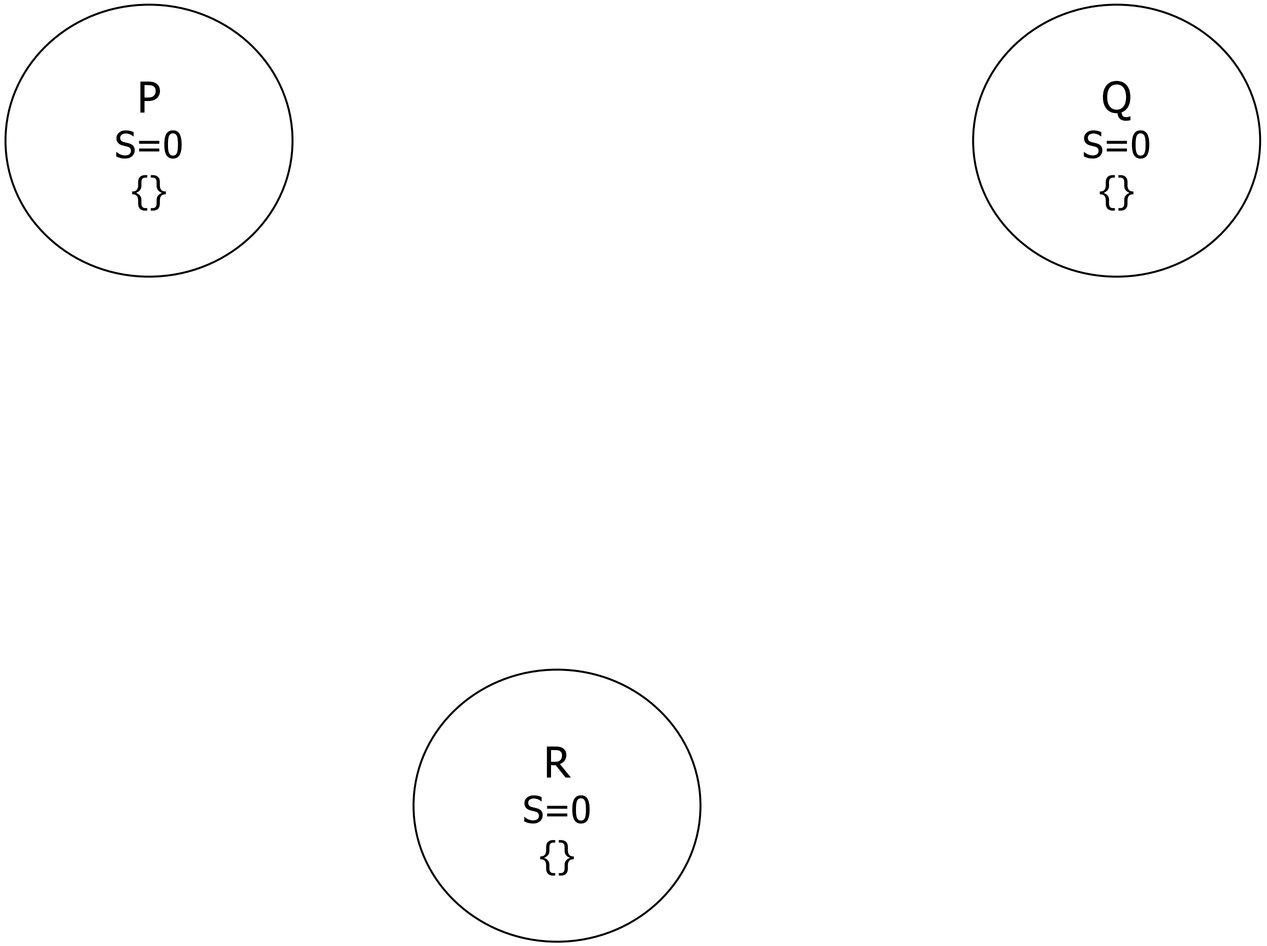
- Messages for which $\mathbf{S} > \mathbf{R^p_g} + 1$ are kept by the process in a **hold-back queue**.
  - Missing messages are then requested by sending NAKs to:
        - the original sender, or
        - to another process, **q** from which it has received the acknowledgement <**q**, $\mathbf{R^q_g}$>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

Reliable Multicast



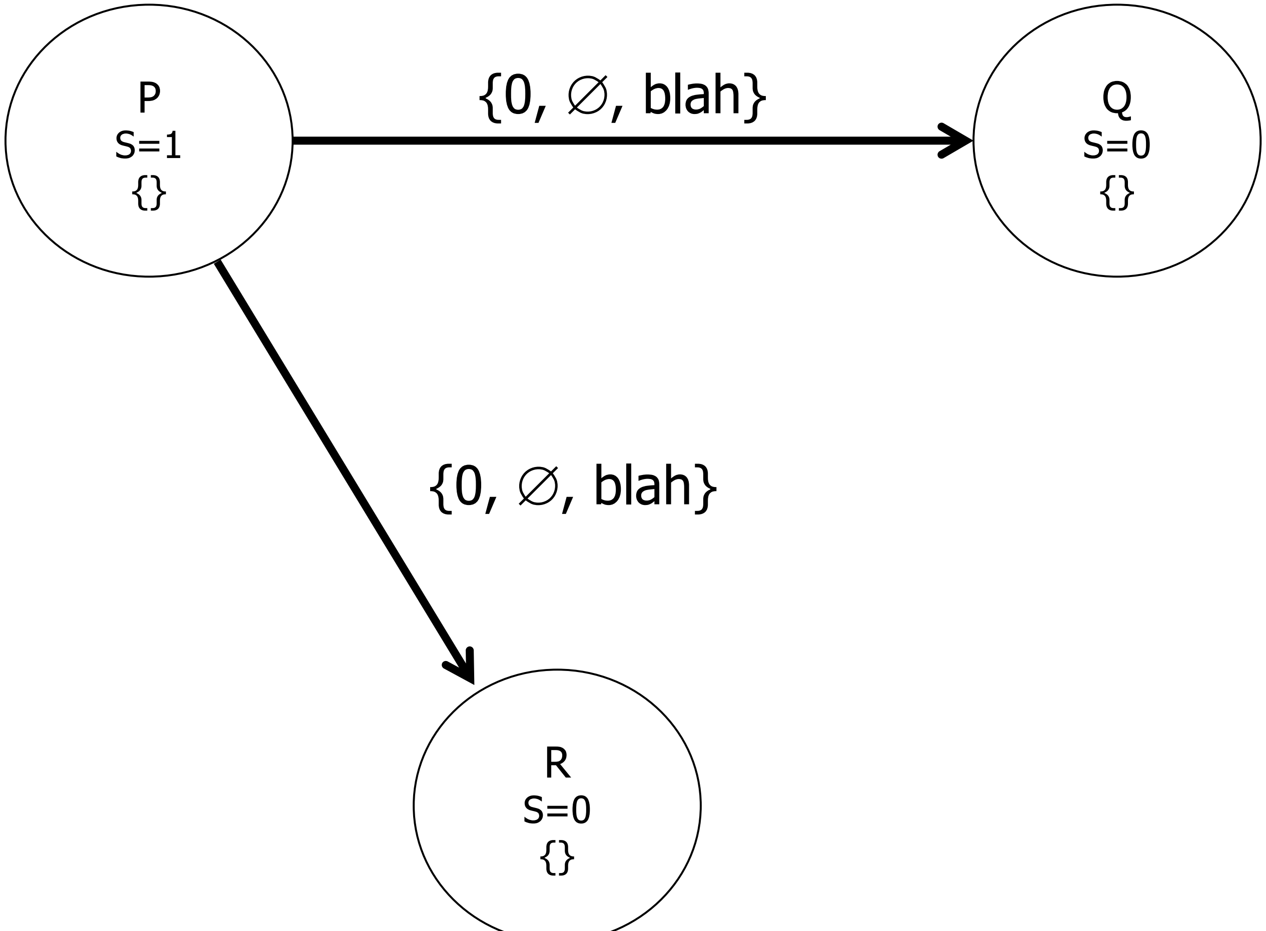
Reliable Delivery

- P, Q, R form a Multicast Group (in this example we assume there is only one group)



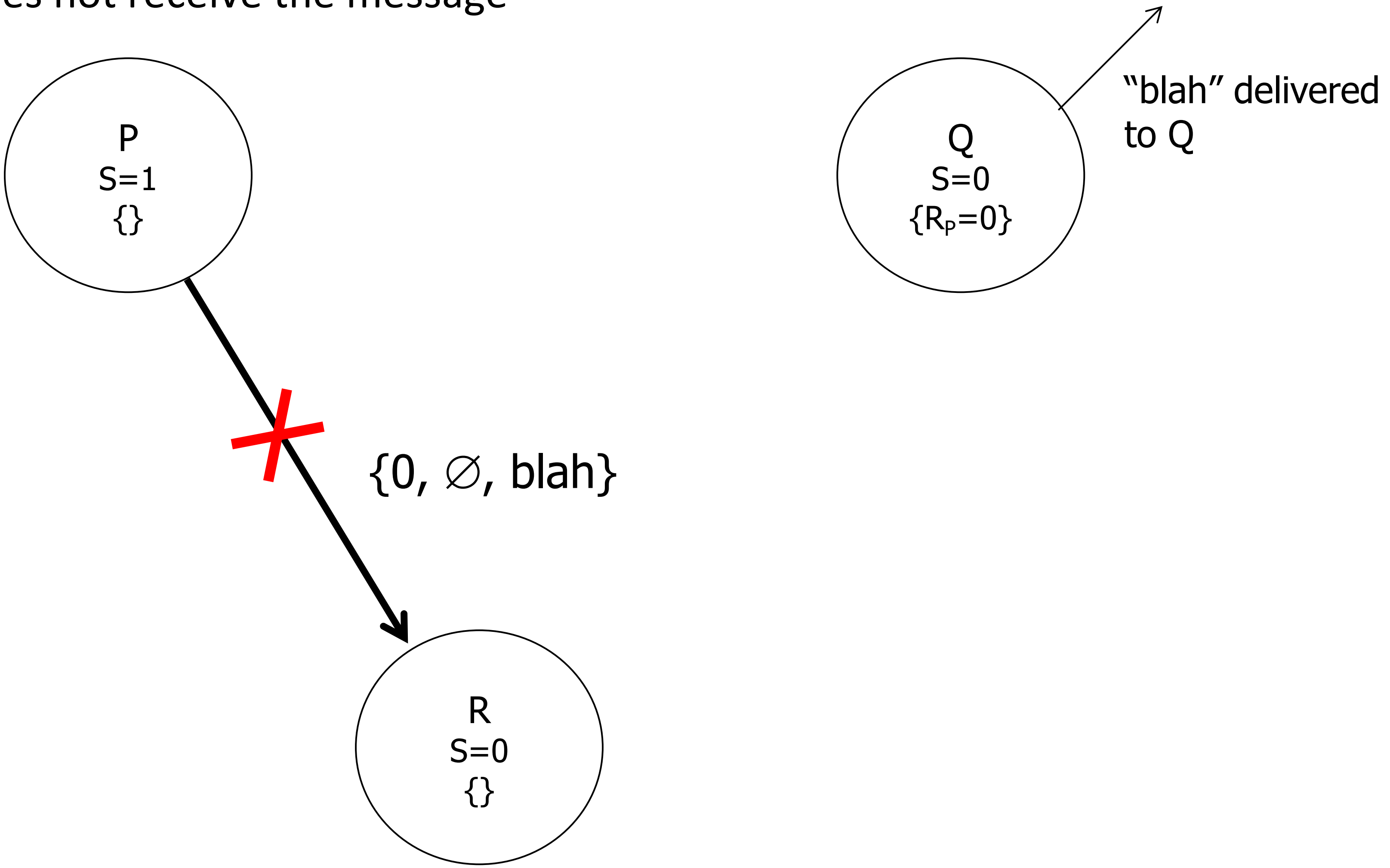
Reliable Delivery

- P sends a message to the group



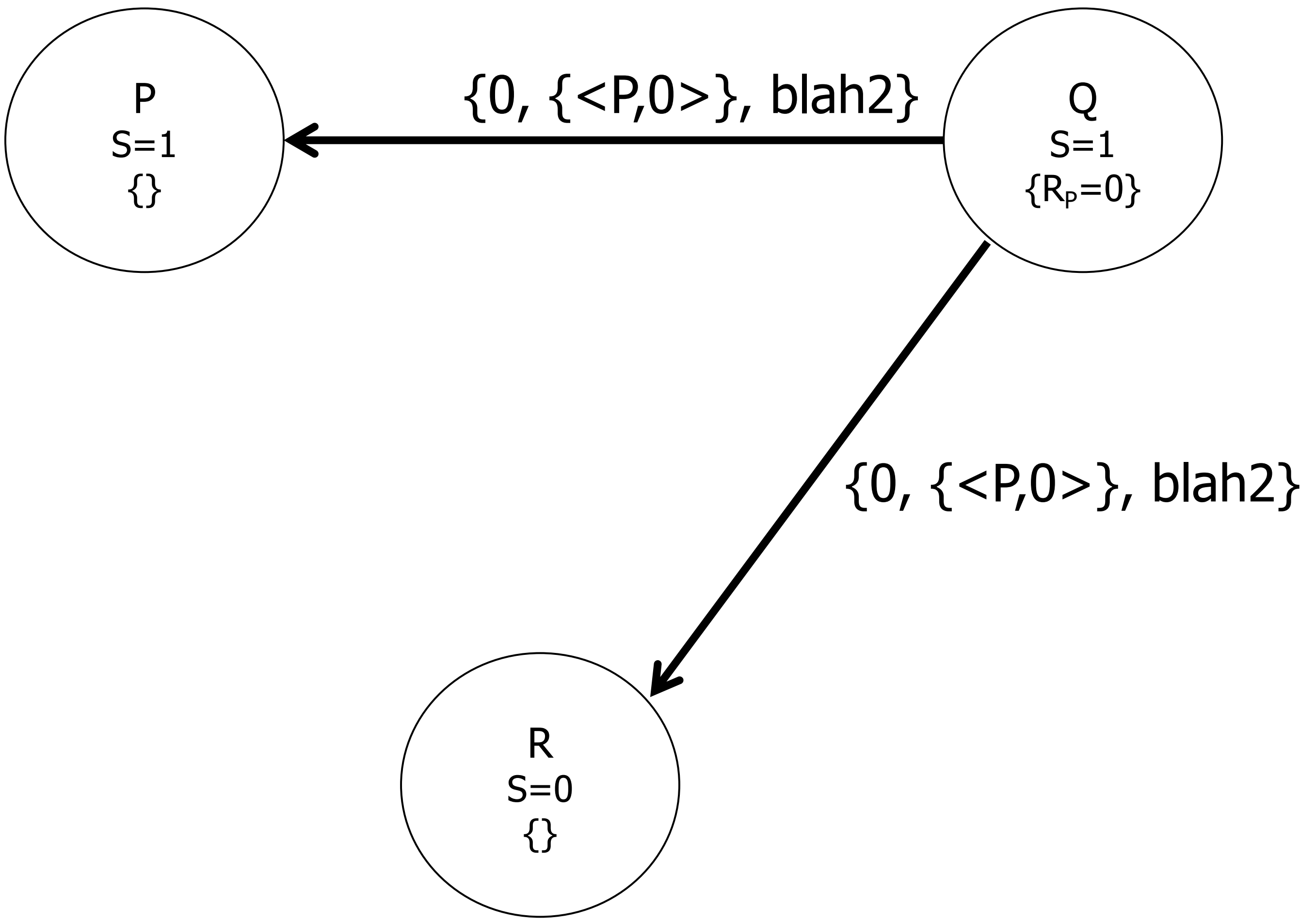
Reliable Delivery

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



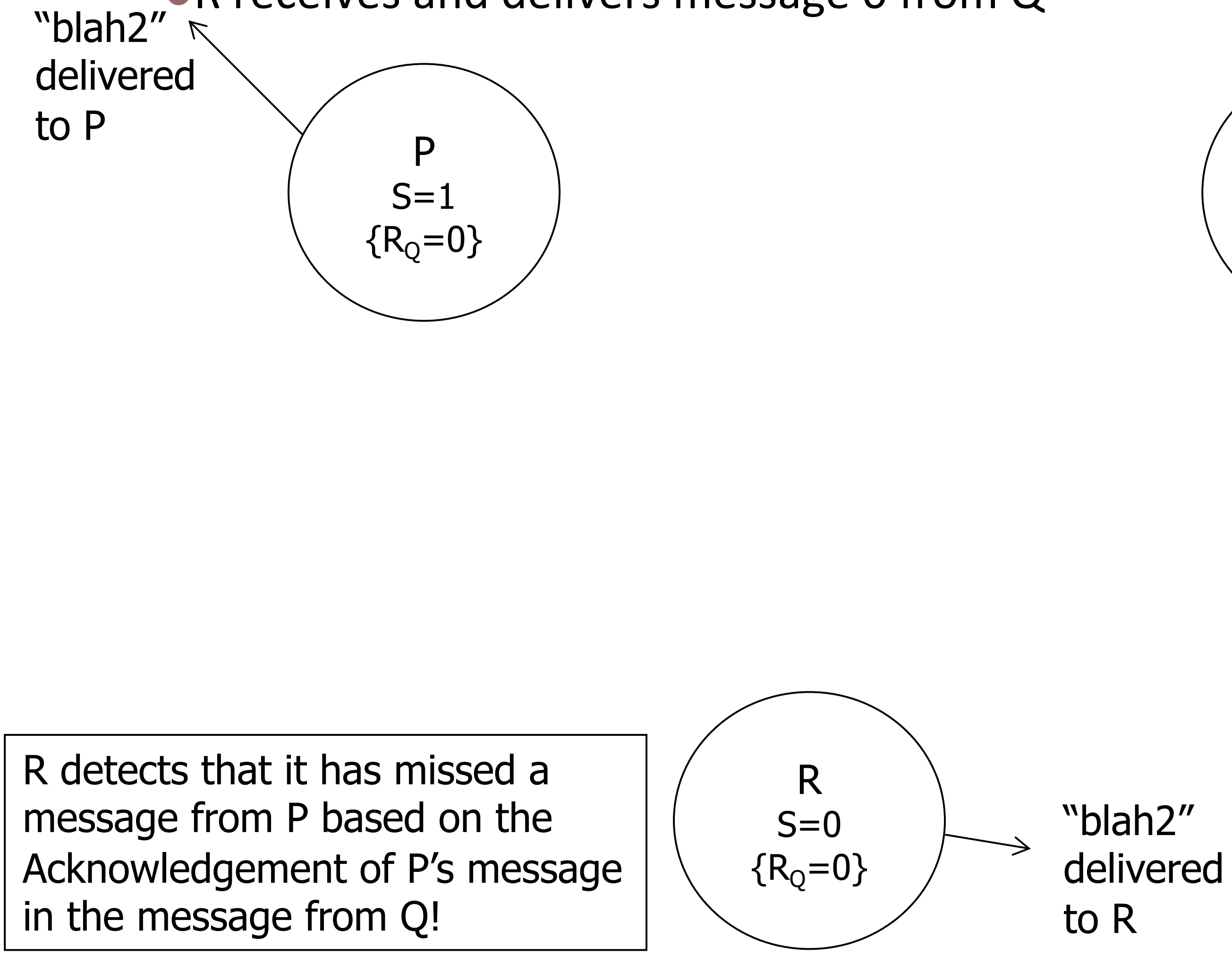
Reliable Delivery

- Q sends a message to the group



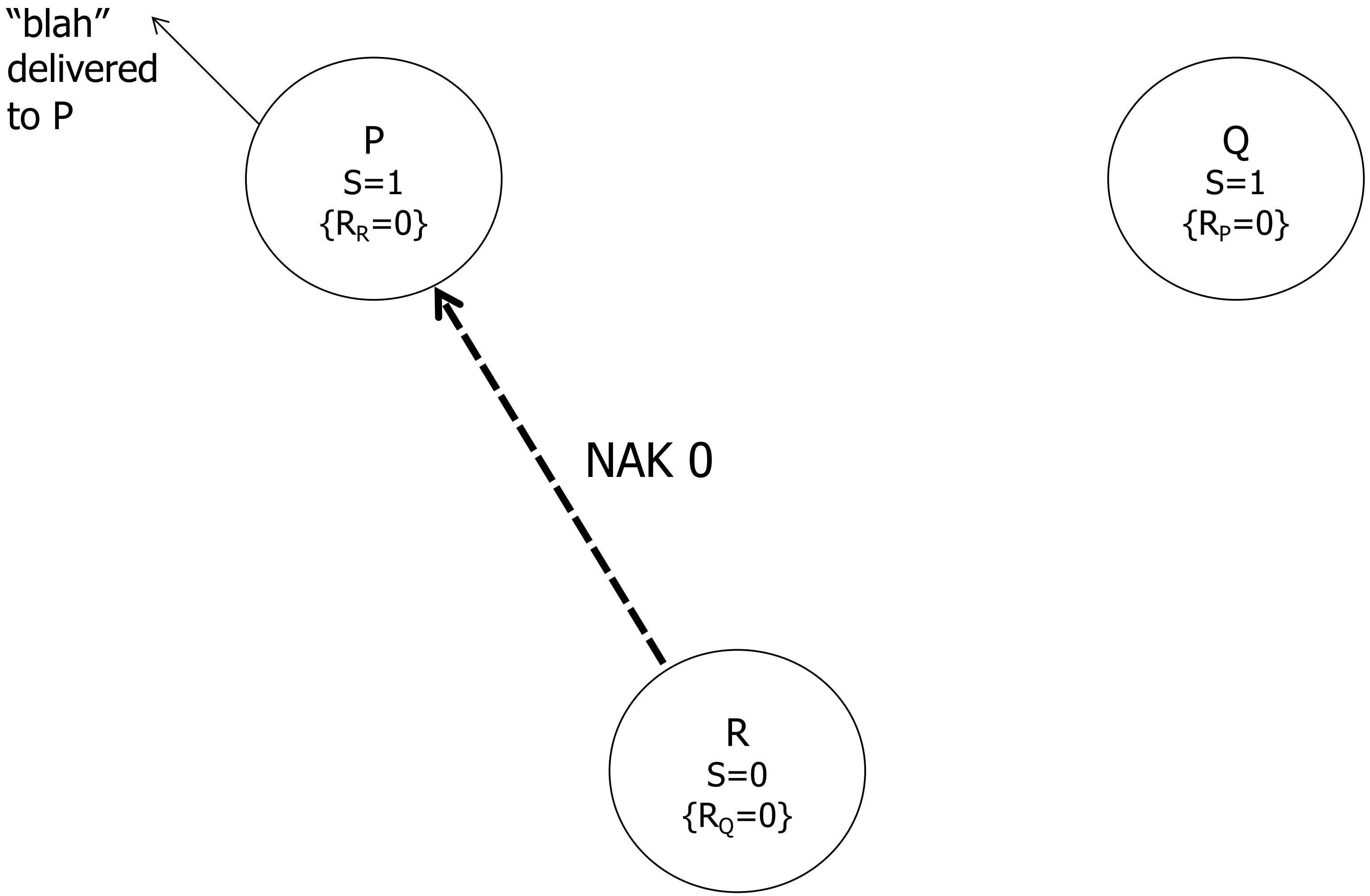
Reliable Delivery

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



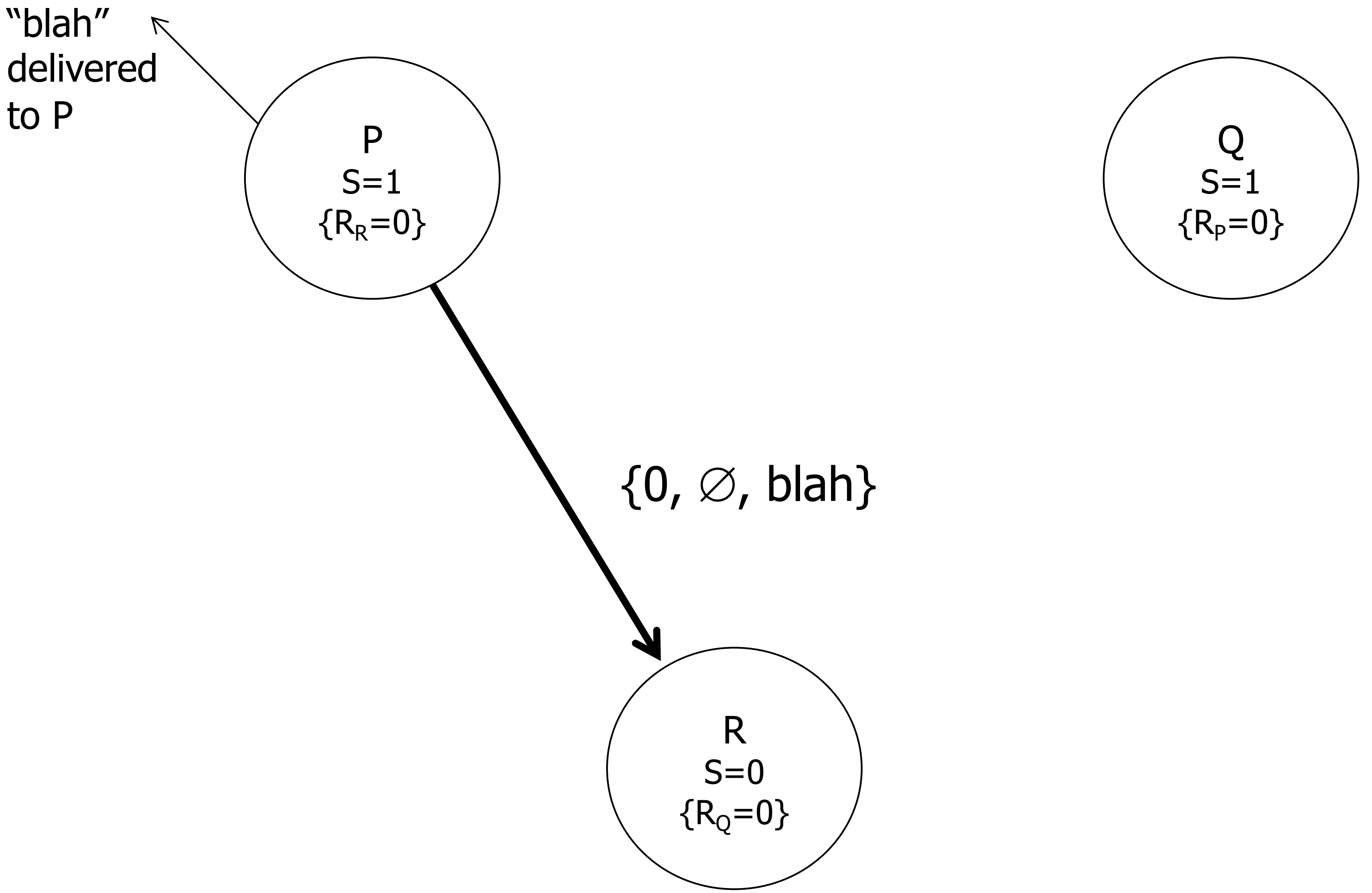
Reliable Delivery

- R sends a NAK to P



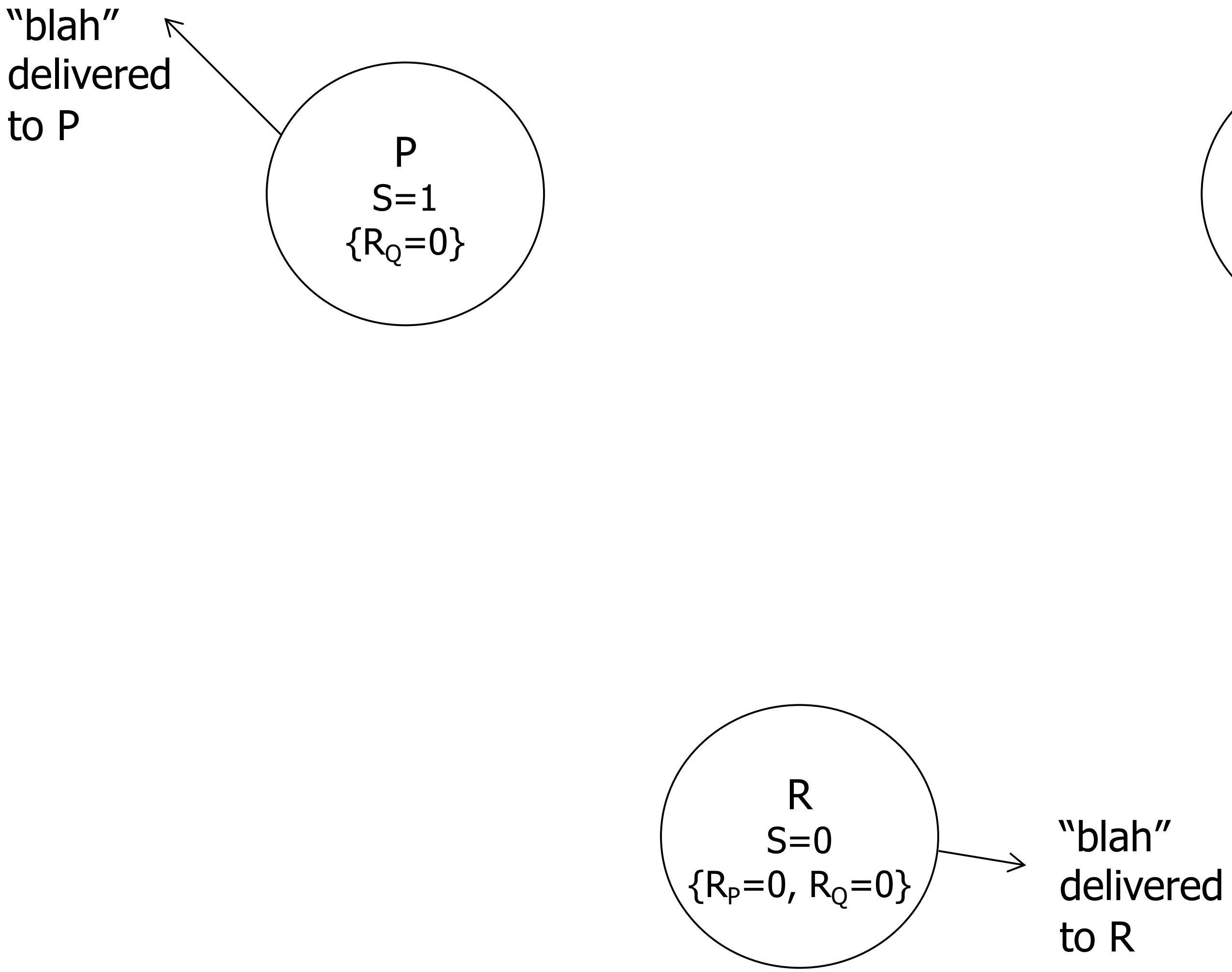
Reliable Delivery

- P retransmits the missing message to R



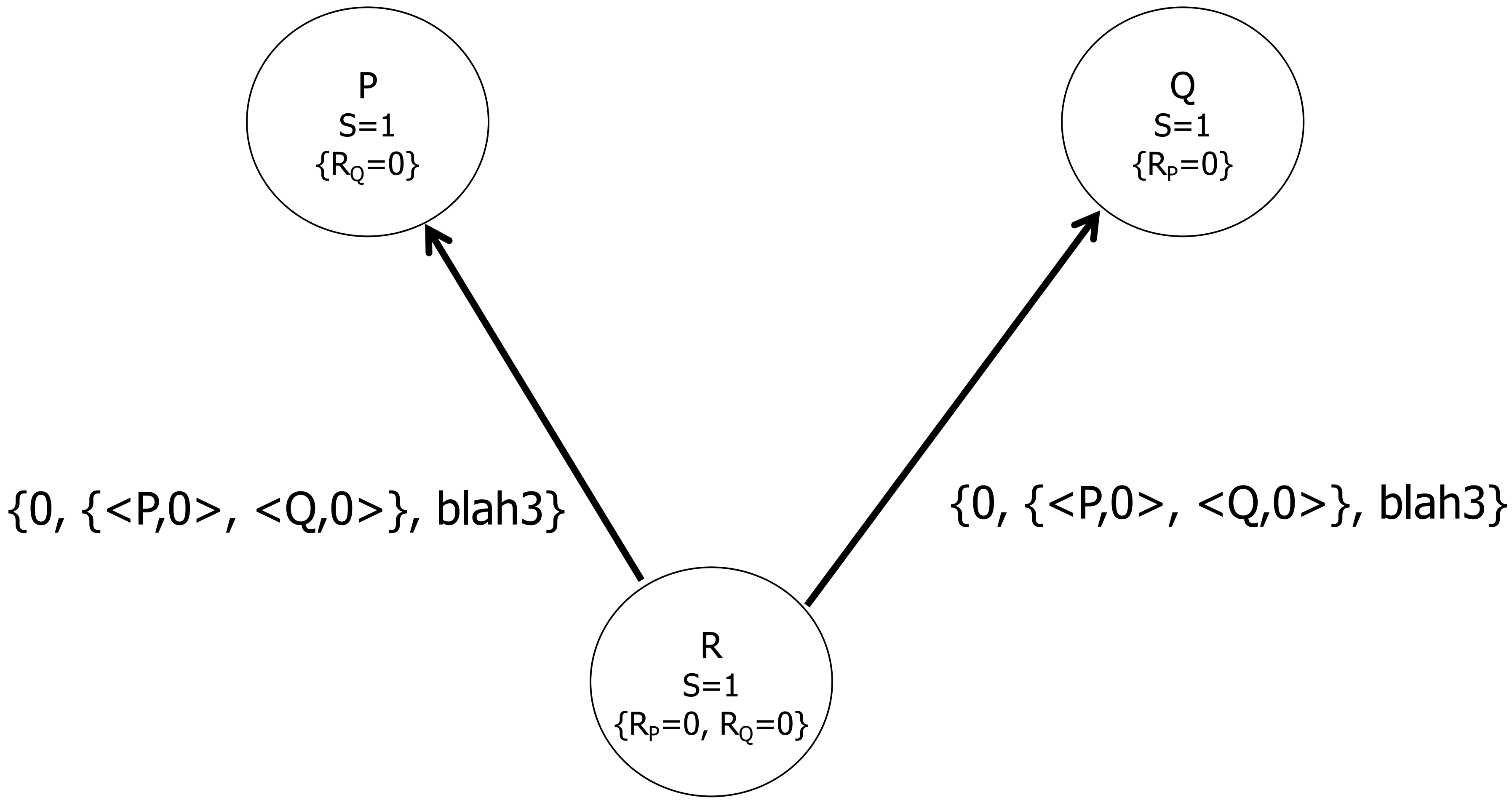
Reliable Delivery

- R receives and delivers the message



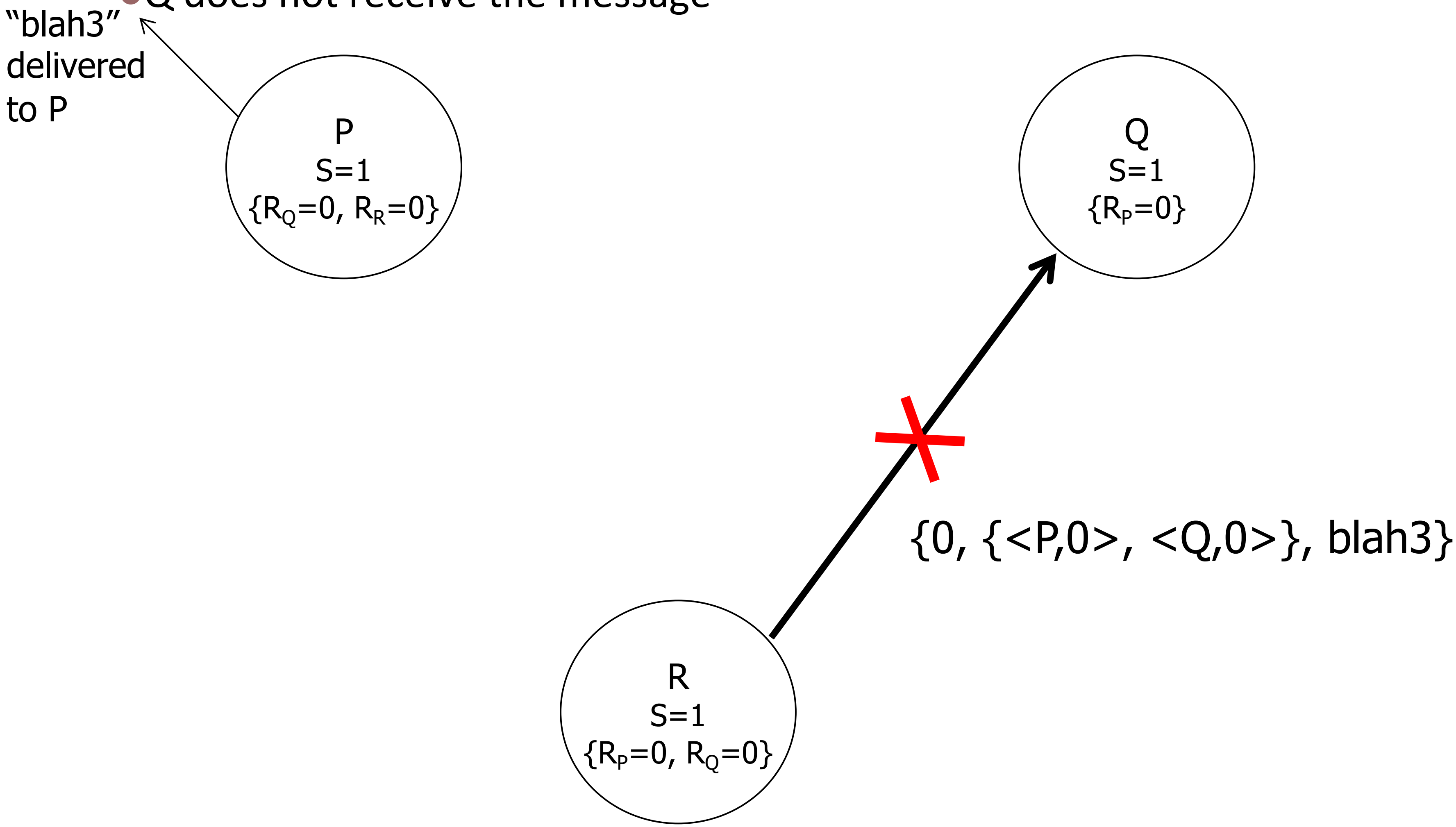
Reliable Delivery

- R multicasts a message to the group



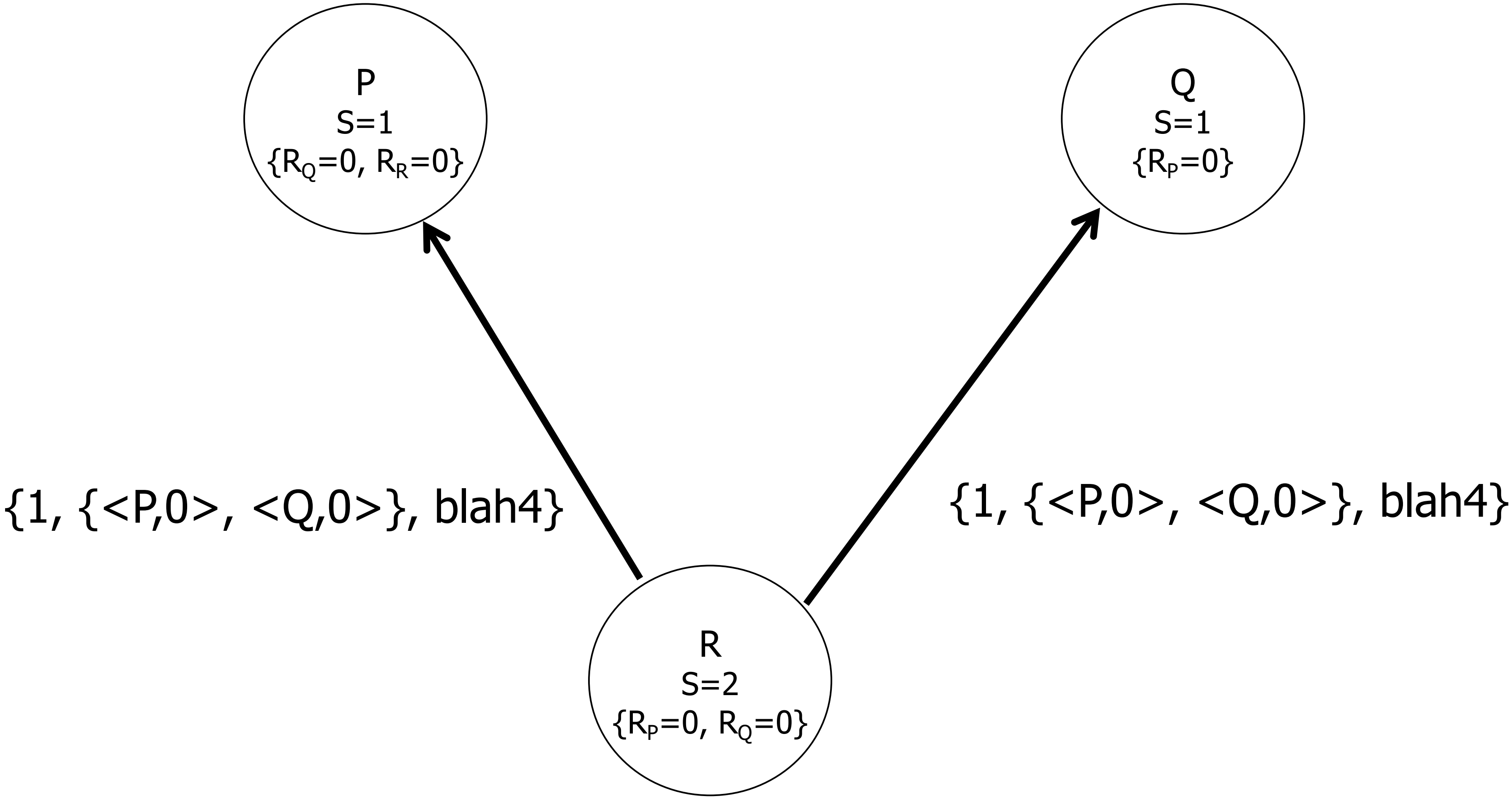
Reliable Delivery

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



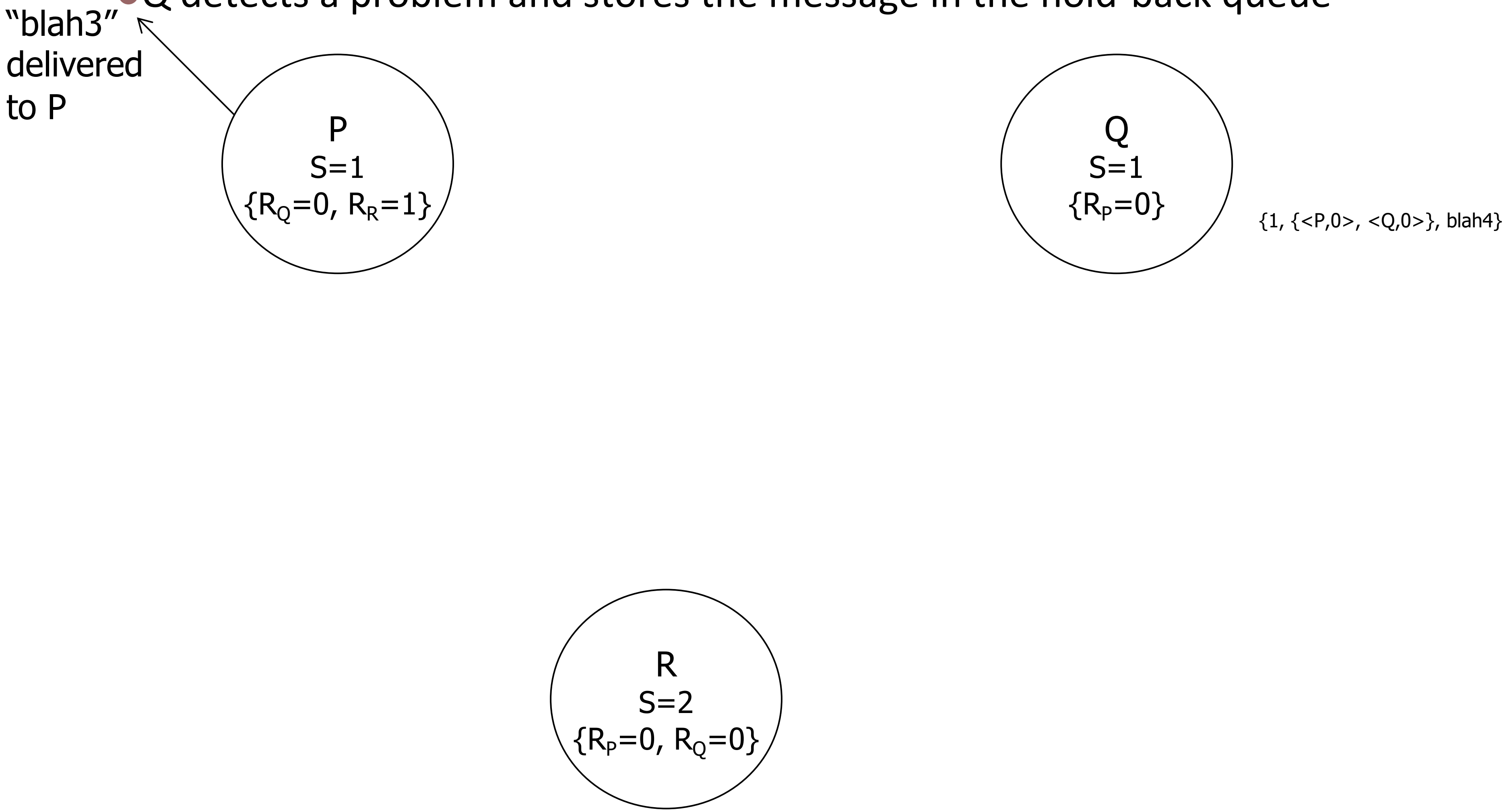
Reliable Delivery

- R multicasts another message to the group



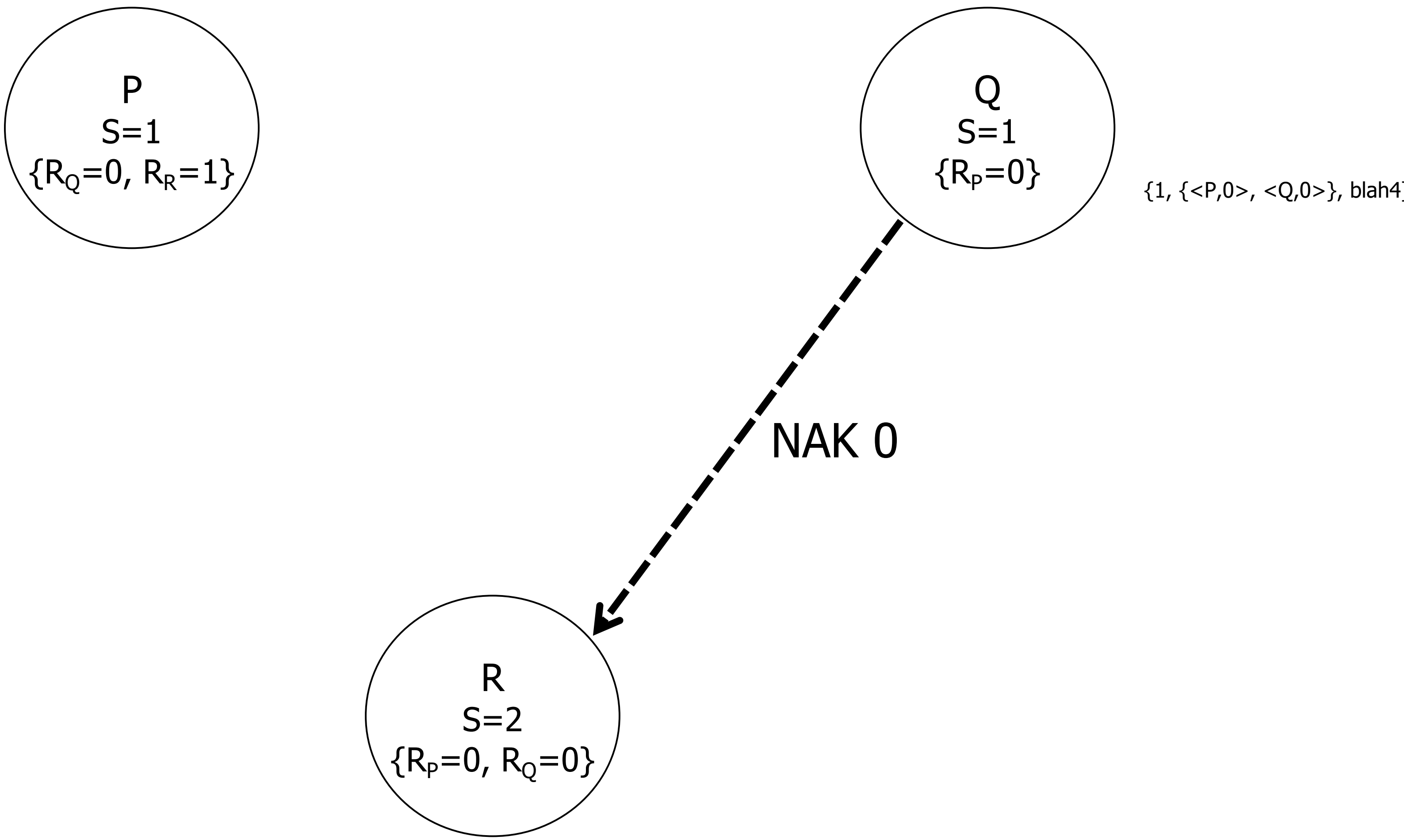
Reliable Delivery

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



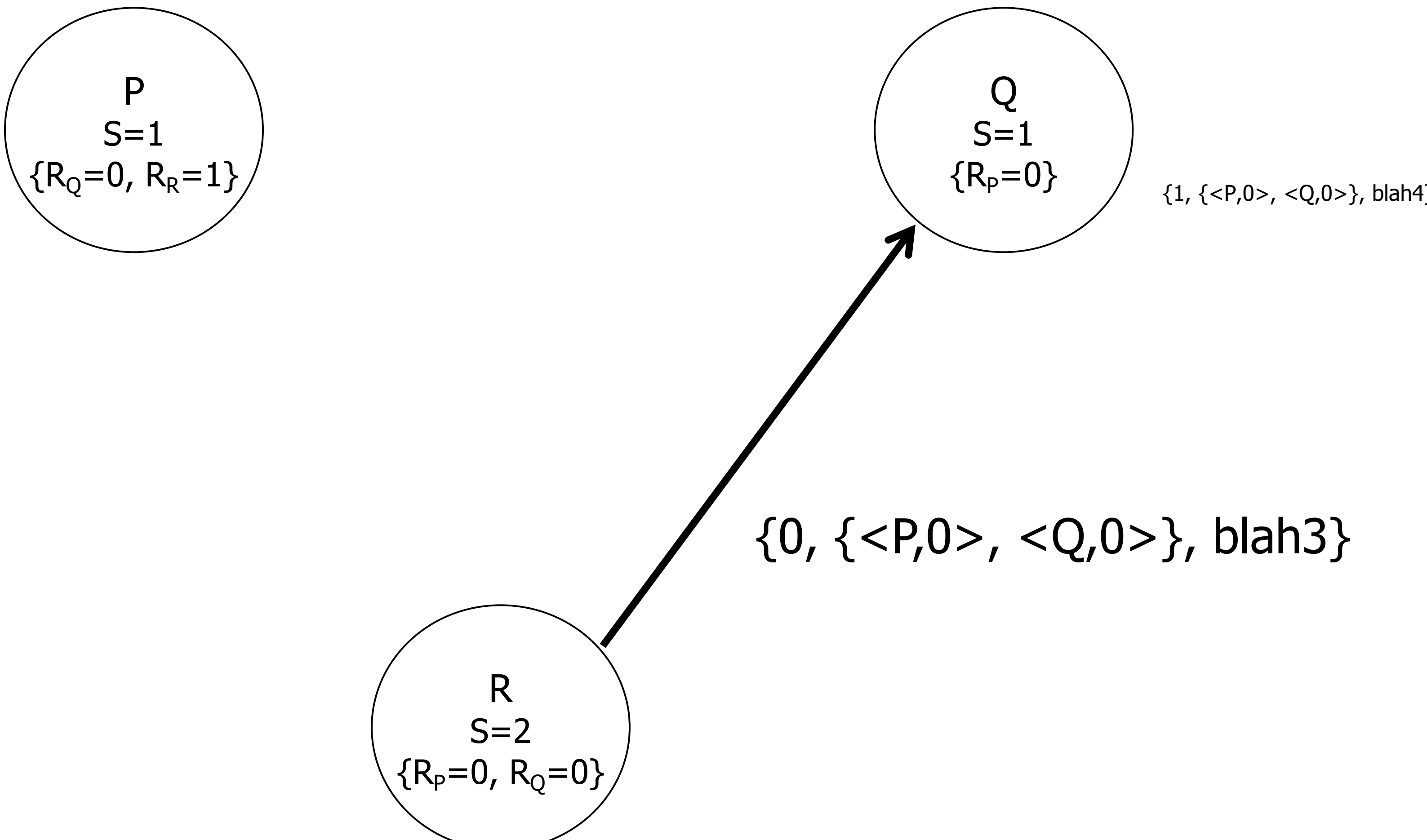
Reliable Delivery

- Q sends a NAK to R



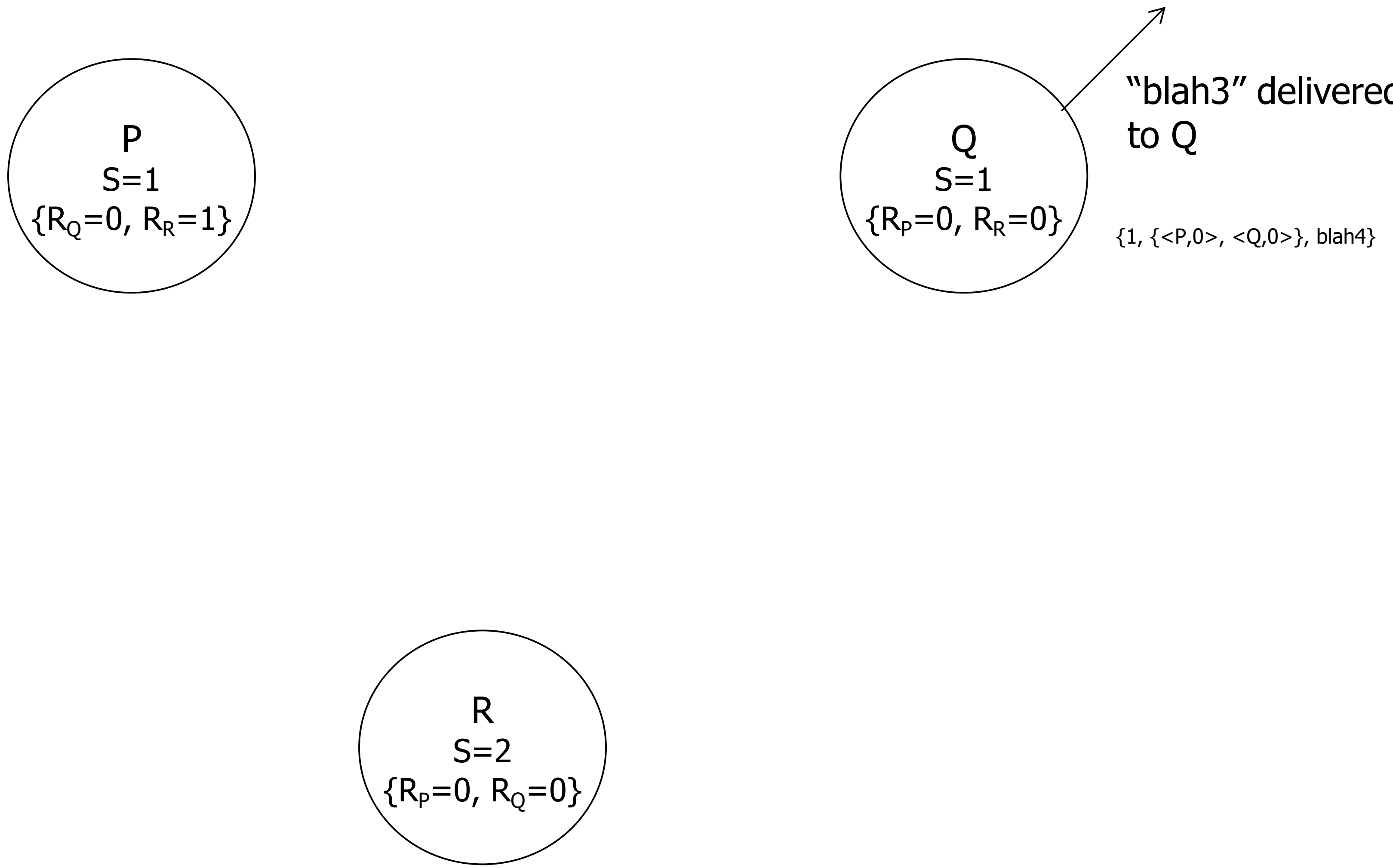
Reliable Delivery

- R retransmits the message



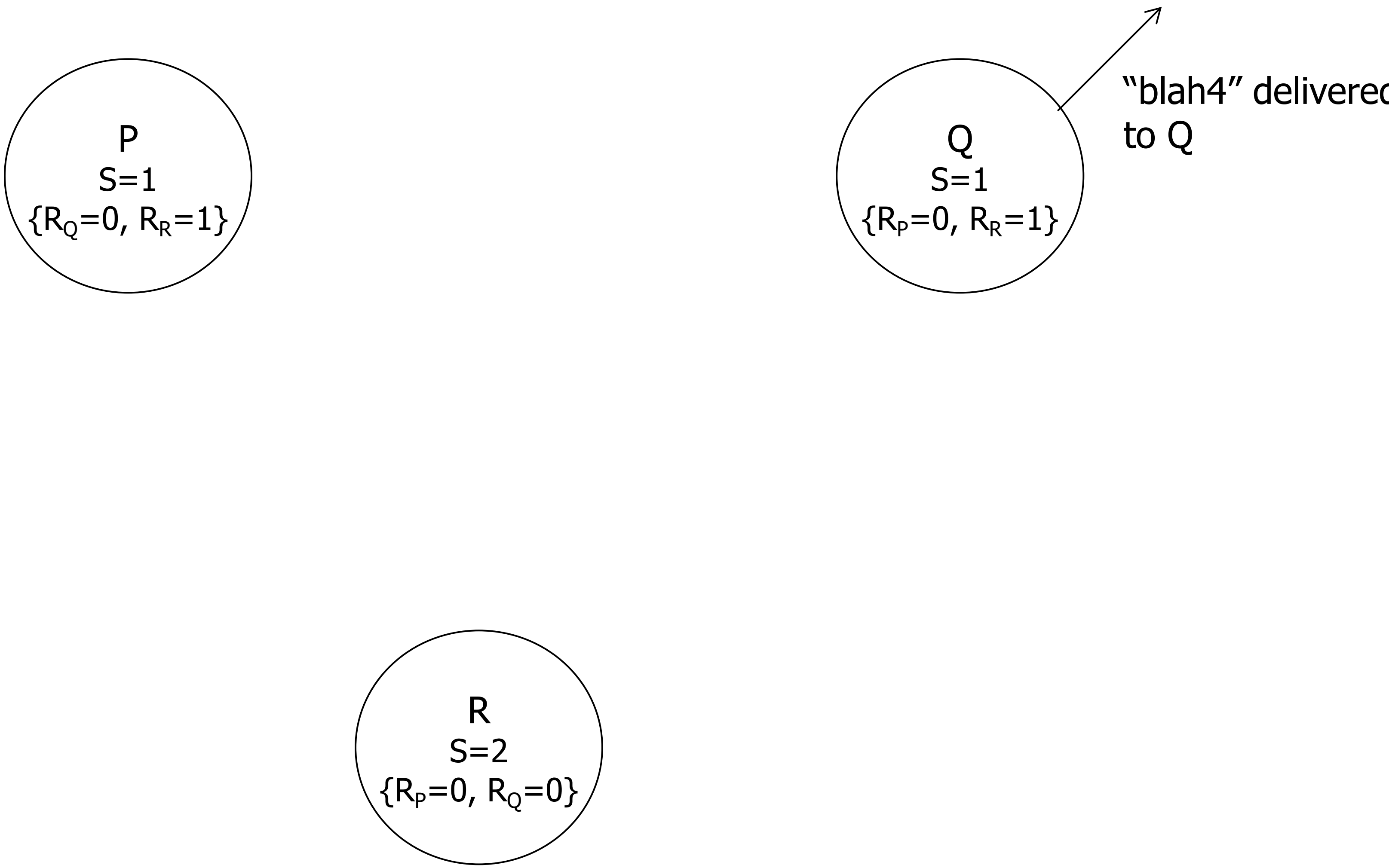
Reliable Delivery

- Q receives and delivers message 0 from R



Reliable Delivery

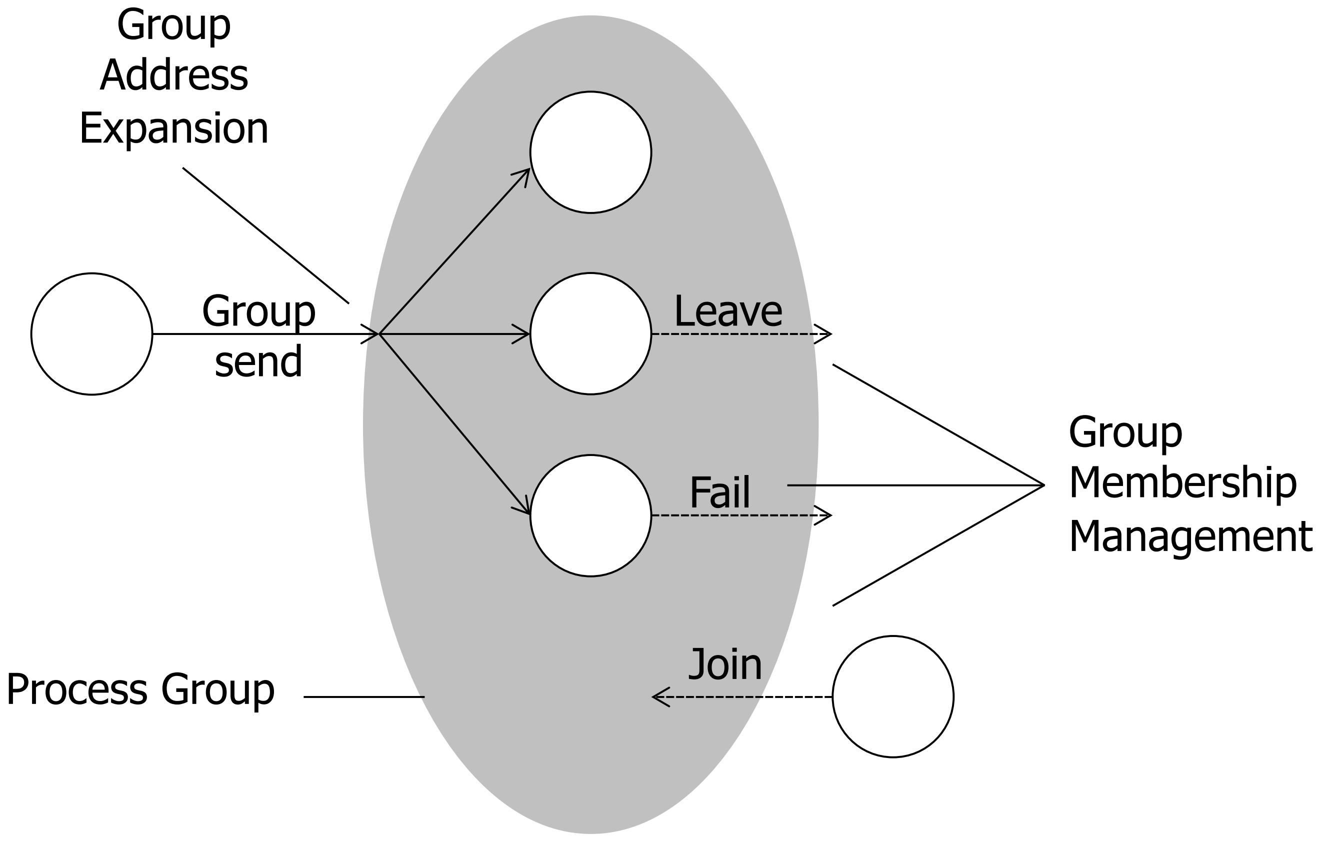
- Q removes message 1 from R from the hold-back queue and delivers it



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
    - Adding & removing processes to/from a group
  - Implementing a failure detector.
    - Monitors the group members for both crashes and common failures.
    - Detector marks services as **Suspected** or **Unsuspected.**
    - This is used 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.

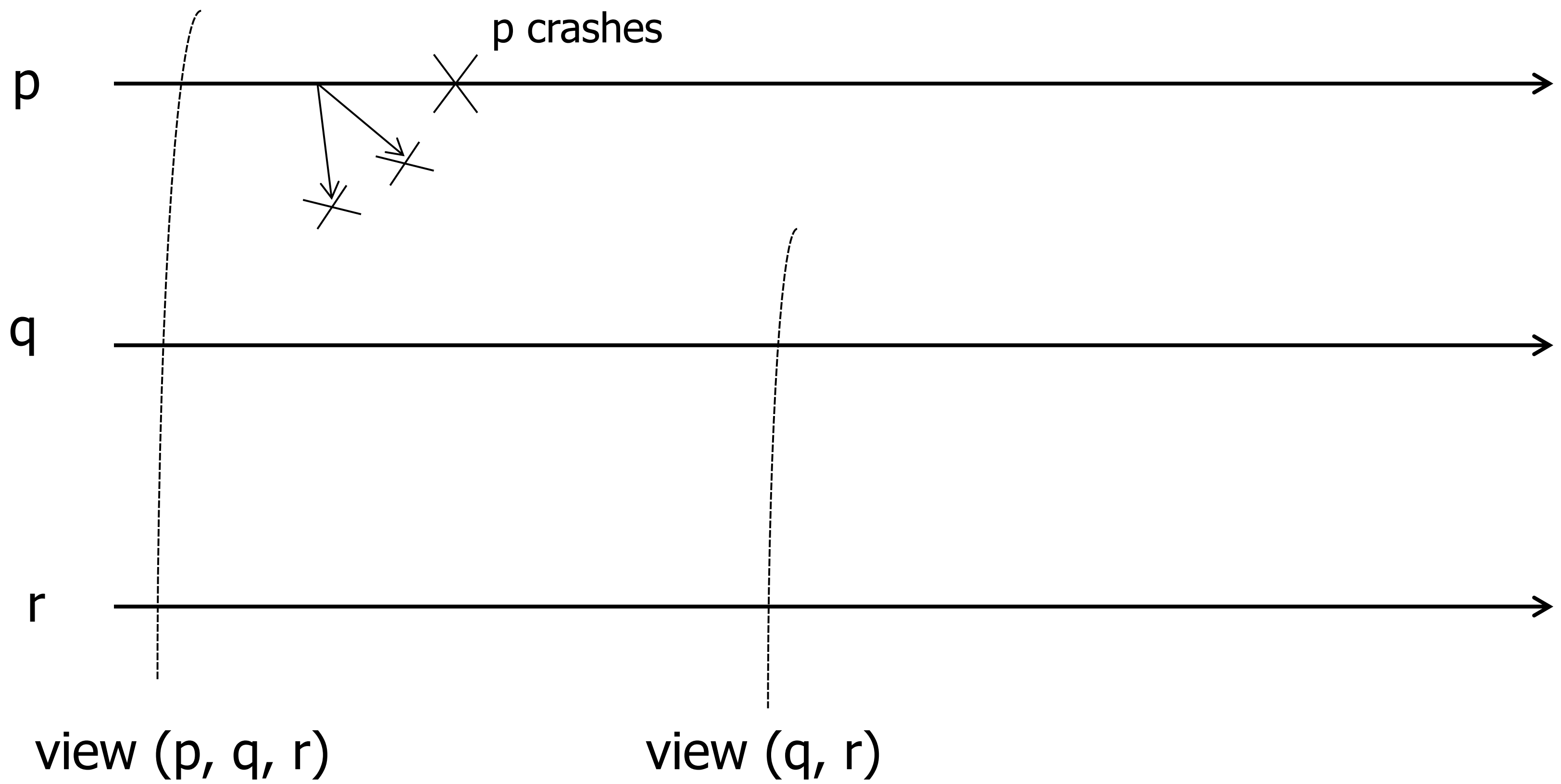
- Group communication can be further enhanced through the use of group views to constrain whether a received message can/should be delivered.

- 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.

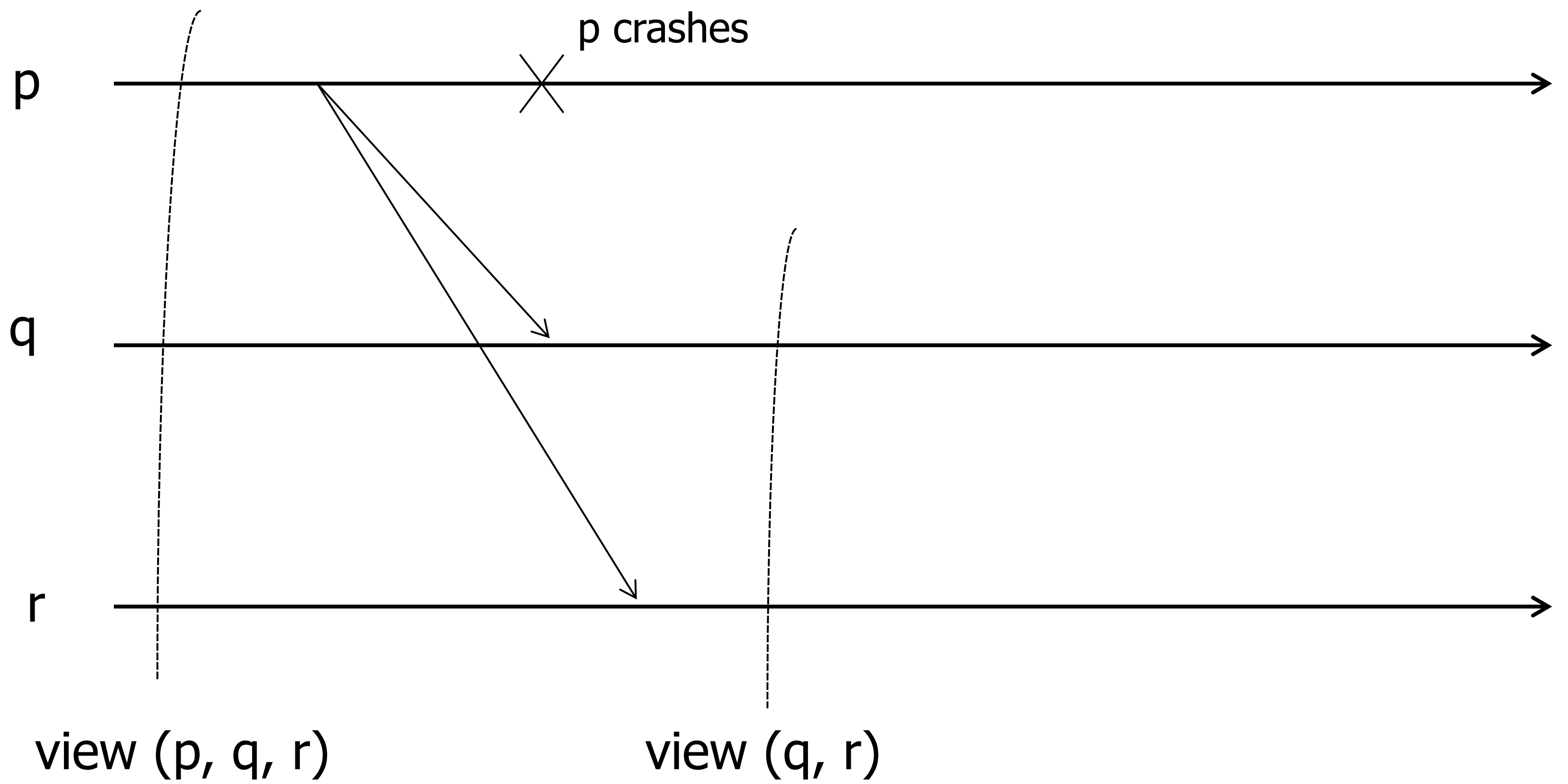
View-Synchronous Group Com.

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



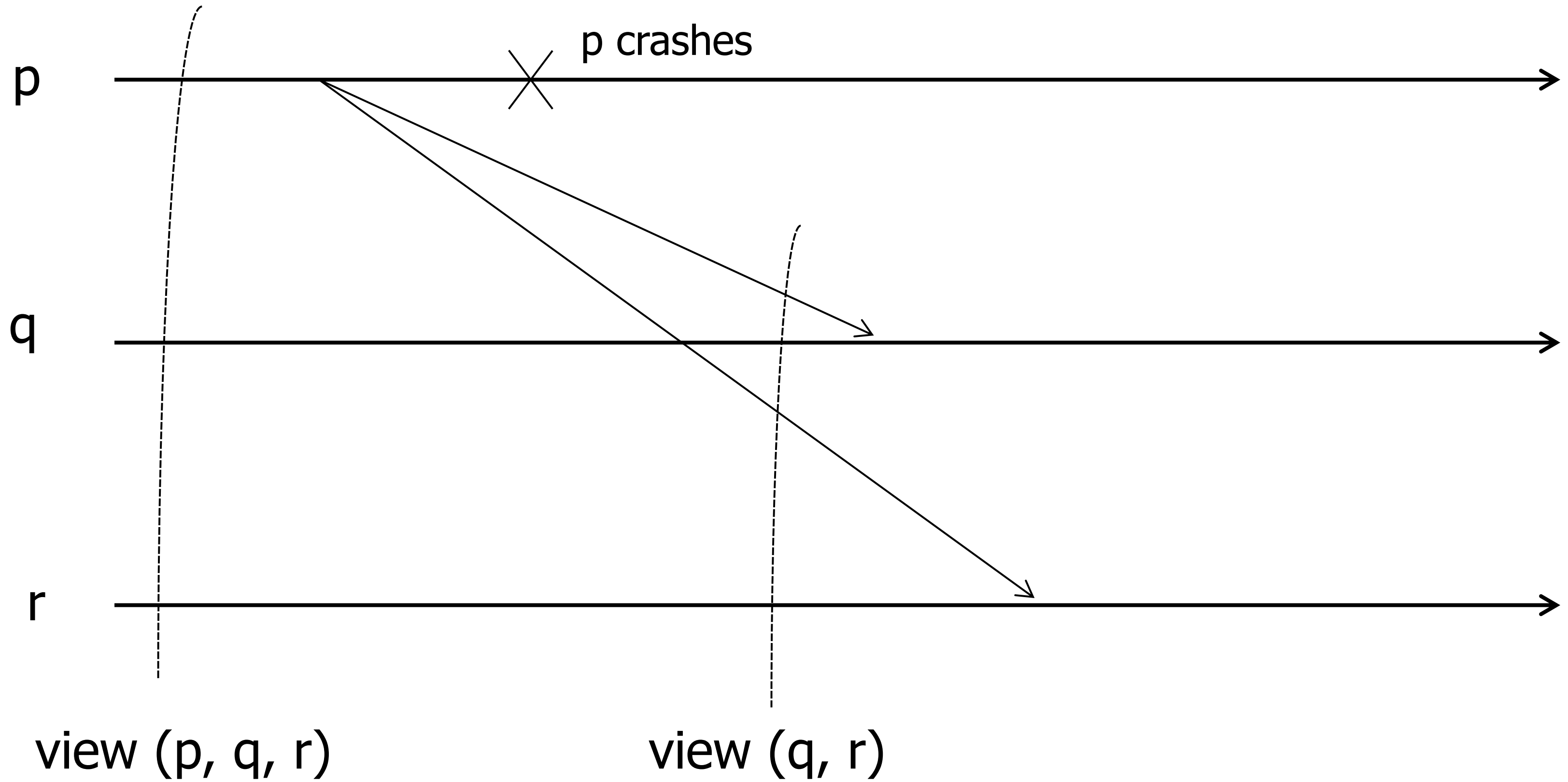
View-Synchronous Group Com.

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



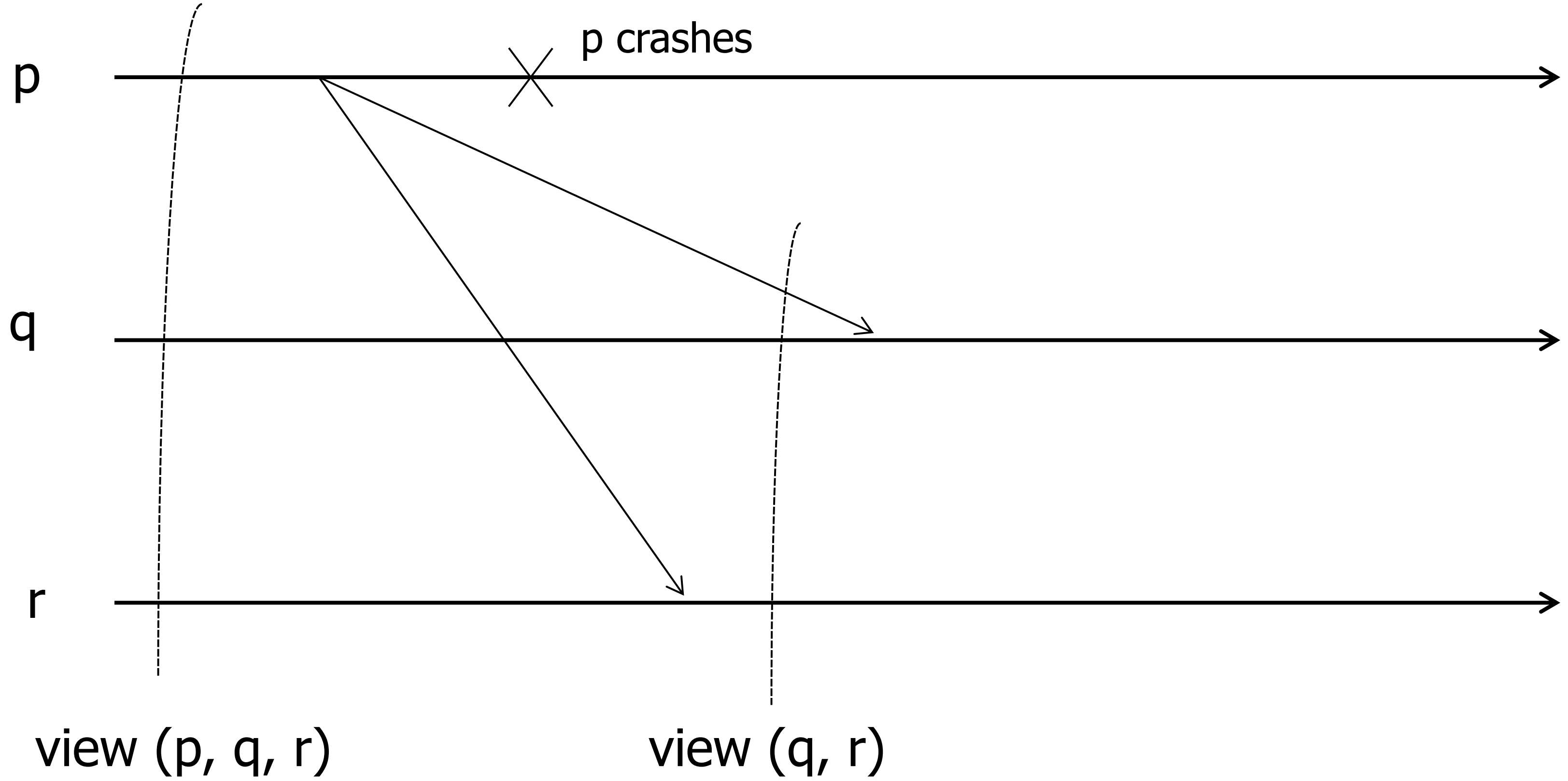
View-Synchronous Group Com.

- **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.

- **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.

- The 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

Thank you

For general enquries, contact:

Please contact the Head Teaching Assistant: Xingyu Pan (Star), Xingyu.Pan@ucdconnect.ie