Comunicación a grupos

Sistemas Distribuidos

Alberto Lafuente, Mikel Larrea

Dpto. ATC, UPV/EHU

Comunicación a grupos

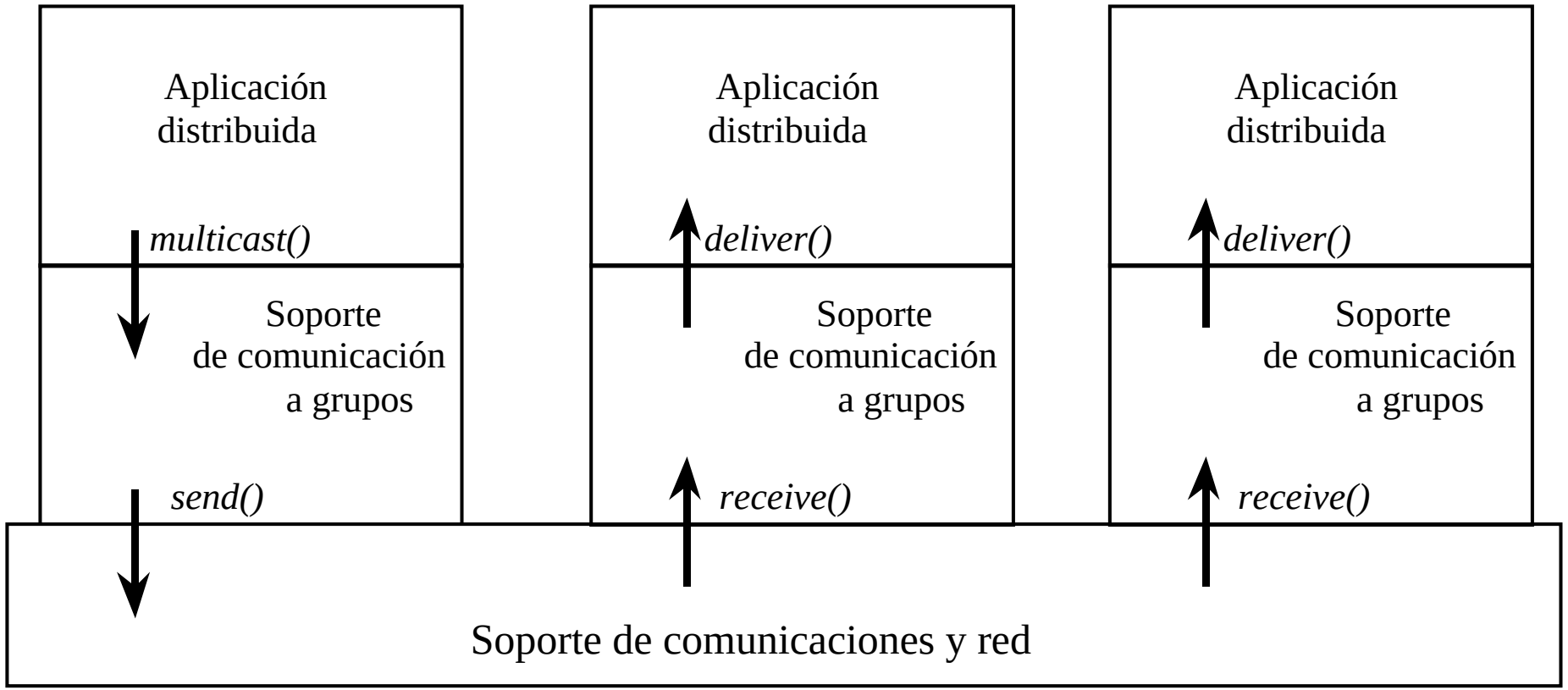
- Modelo:
  - grupos de procesos
  - radiado (comunicación 1:N), distintas semánticas
  - *broadcast* y *multicast*
- Aplicaciones:
  - notificación de eventos
  - descubrimiento de servicios remotos
  - replicación de servicios, para tolerancia a fallos, disponibilidad, rendimiento…
- Ejemplos: *Isis*, *Horus*, *Ensemble*, *JGroups*, *Transis*, *Totem*, *Cactus*, *Phoenix*, *Spread*, *Appia*…

Comunicación a grupos (2)

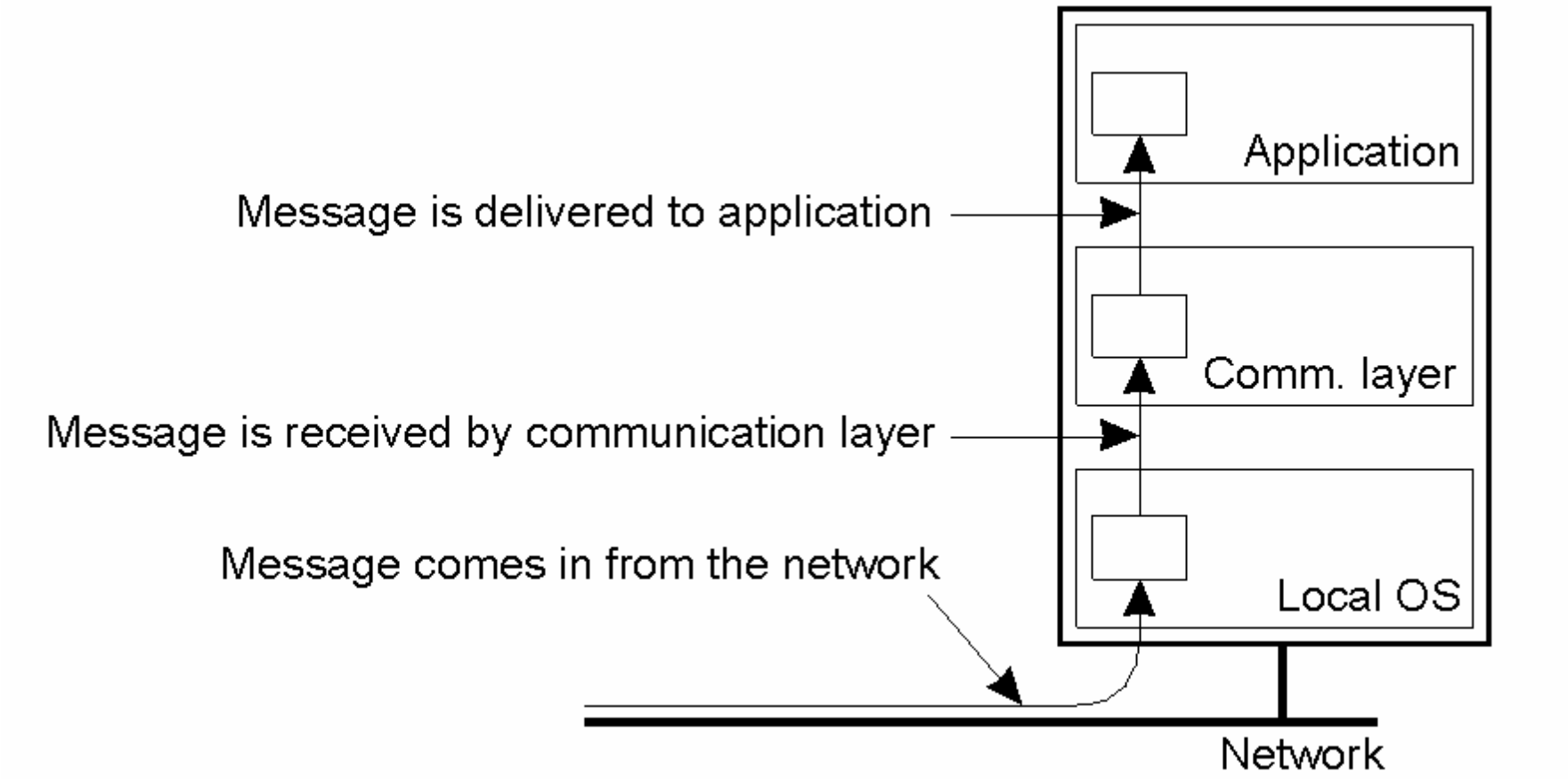
- Infraestructura de soporte (*middleware)*:
  - identificación de los grupos
  - Entrega de mensajes a los miembros del grupo, respetando la semántica correspondiente
  - Gestión de la pertenencia al grupo (entradas, salidas, fallos…) de manera consistente: vistas “síncronas”
- Modelo de comunicación:
  - ***multicast****(G, m)*. Difusión del mensaje *m* al grupo *G*. Suposición: el emisor es miembro del grupo
  - ***deliver****(m)*. Entrega del mensaje *m* al proceso que ejecuta dicha llamada

Comunicación a grupos (3)

- Modelo de comunicación:

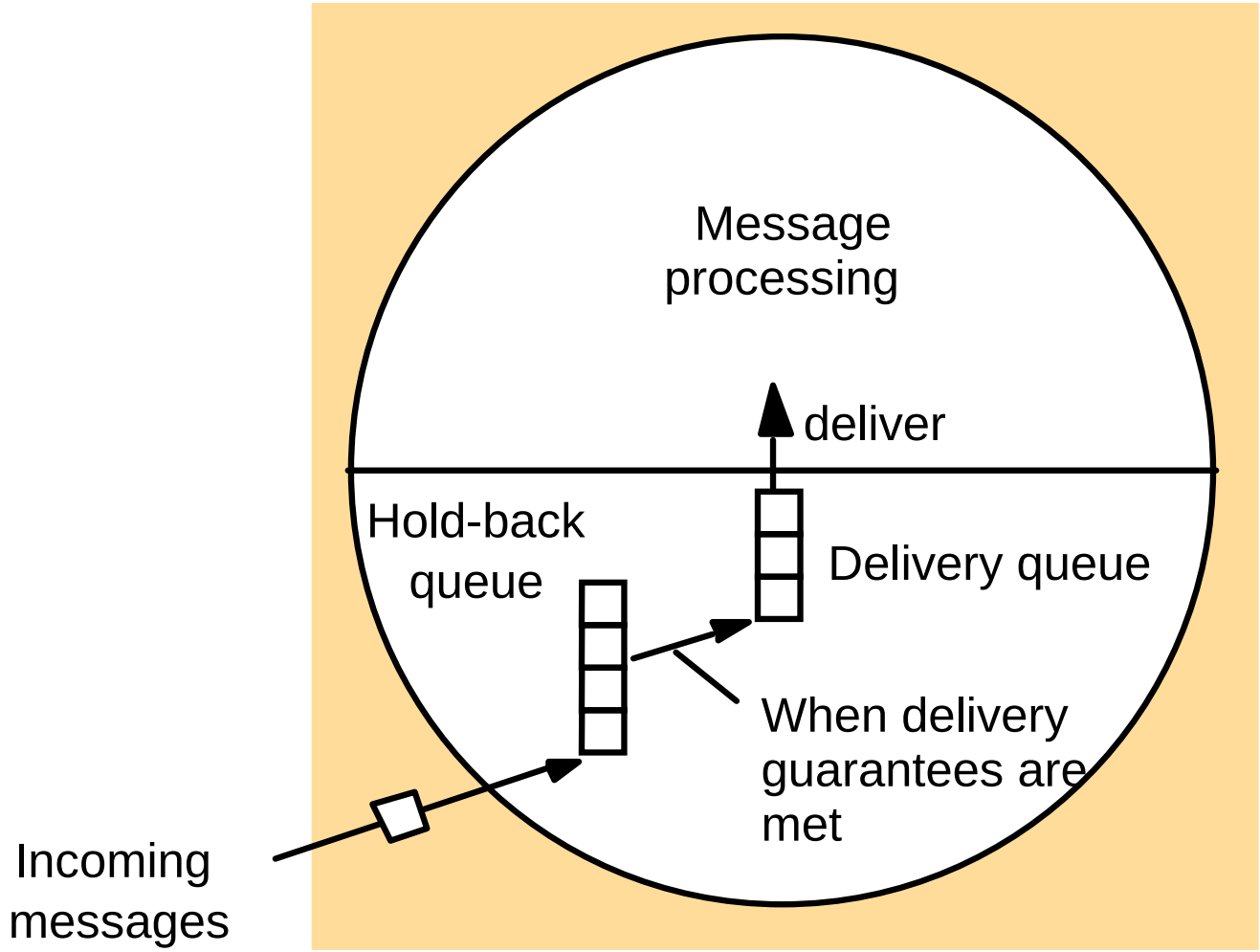


Message Receipt and Delivery



The logical organization of a distributed system to distinguish between message receipt and message delivery

The hold-back queue for arriving multicast messages



Comunicación a grupos (4)

- Los procesos pueden fallar
  - suponemos fallos de parada permanente (*crash*)
  - dos tipos de procesos: correctos e incorrectos
- Los canales de comunicación son casi-fiables
  - todos los mensajes enviados entre procesos correctos terminan por ser recibidos
  - Los canales fiables garantizan la entrega de los mensajes enviados por procesos incorrectos a procesos correctos (por supuesto, antes de fallar)
    - se emplean para demostrar resultados teóricos, en la práctica se utilizan canales casi-fiables o incluso más débiles (no fiables)

Comunicación a grupos (5)

- Semánticas de difusión. Aspectos:
  - fiabilidad
  - orden de entrega
- Implementación “aproximada” (no fiable):
  - *multicast(G, m)*: $\forall p \in G$, *send(p, m)*
  - al ejecutar *receive(m)*: ejecutar *deliver(m)*
    - difusión no fiable: algunos procesos correctos entregan el mensaje, mientras que otros procesos correctos no lo entregan (sucede si el emisor falla durante *multicast*)

Comunicación a grupos (6)

- Difusión fiable (“todos o ninguno”):
  - validez: si un proceso correcto difunde un mensaje *m*, entonces dicho proceso entrega el mensaje *m*
  - acuerdo: si un proceso correcto entrega un mensaje *m*, entonces todos los procesos correctos entregan el mensaje *m*
    - el algoritmo anterior no cumple esta propiedad
  - integridad: para cualquier mensaje, los procesos correctos lo entregarán como mucho una vez, y únicamente si ha sido previamente difundido
  - Implementación (*R-multicast*, *R-deliver*):
    - al ejecutar *R-multicast(G, m)*: ejecutar *multicast(G, m)*
    - al ejecutar *receive(m)* por primera vez:
      1. si no se es el emisor original de *m*: ejecutar *multicast(G, m)*
      2. ejecutar *R-deliver(m)*

Reliable multicast algorithm

On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(*g*, *m*); // $p \in g$ is included as a destination

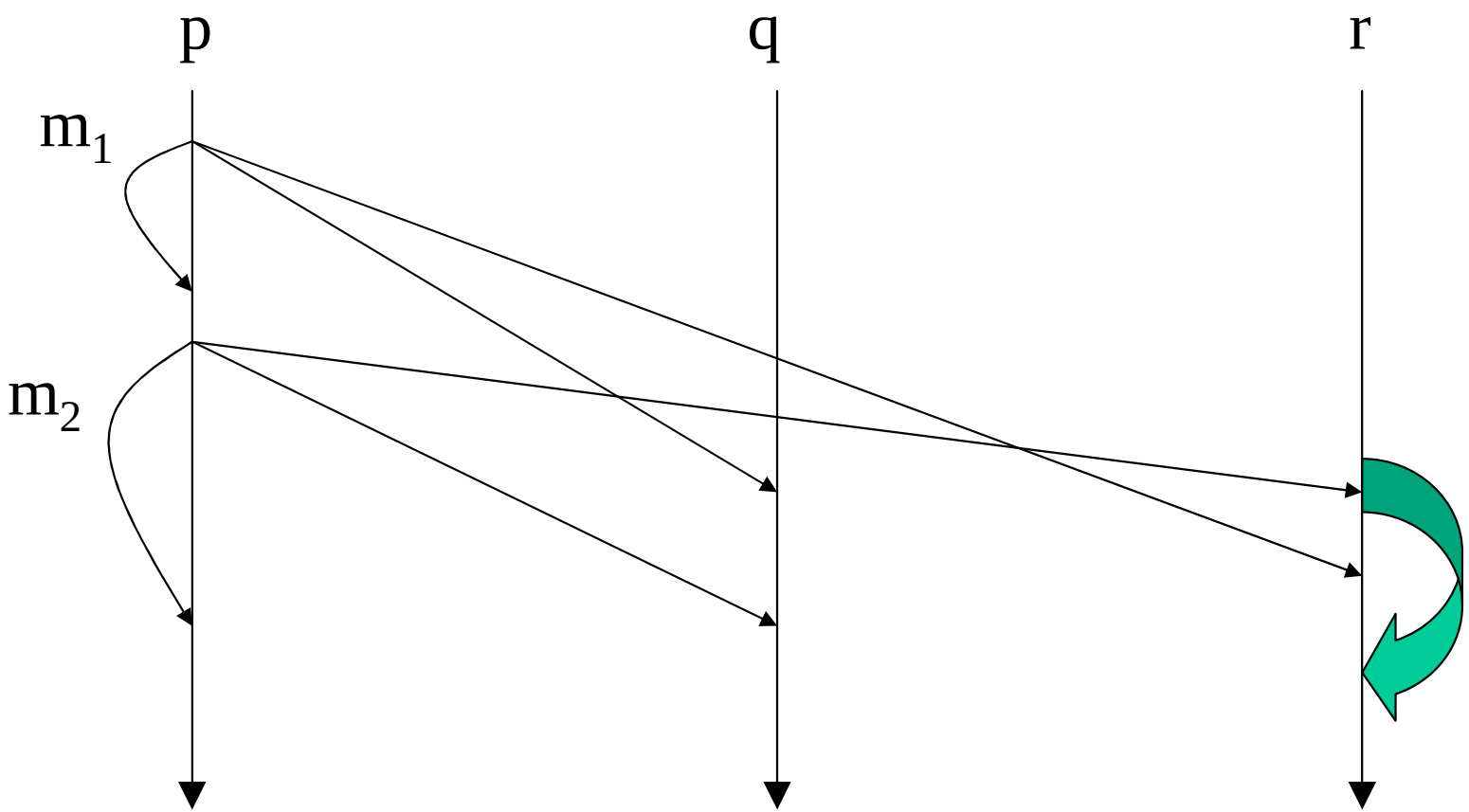
On B-deliver(*m*) *at process q with g = group*(*m*)
    if ($m \notin$ *Received*)
    then
        Received := *Received* $\cup$ {*m*};
        if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*
        R-deliver m;
    end if

Comunicación a grupos (7)

- Difusiones (fiables) ordenadas:
  - orden *FIFO*: si un proceso difunde un mensaje m_1 antes que otro mensaje m_2, entonces ningún proceso correcto entregará m_2 sin haber previamente entregado m_1
  - orden causal: si la difusión de un mensaje m_1 precede causalmente la de otro mensaje m_2, entonces ningún proceso correcto entregará m_2 sin haber previamente entregado m_1
    - el orden causal implica orden *FIFO*
  - orden total: si dos procesos correctos P_i y P_j entregan dos mensajes m_1 y m_2, entonces P_i entregará m_1 antes que m_2 si y sólo si P_j también entrega m_1 antes que m_2
    - a la difusión con orden total también se le conoce como difusión atómica. Se suele utilizar como base para la consistencia en sistemas replicados

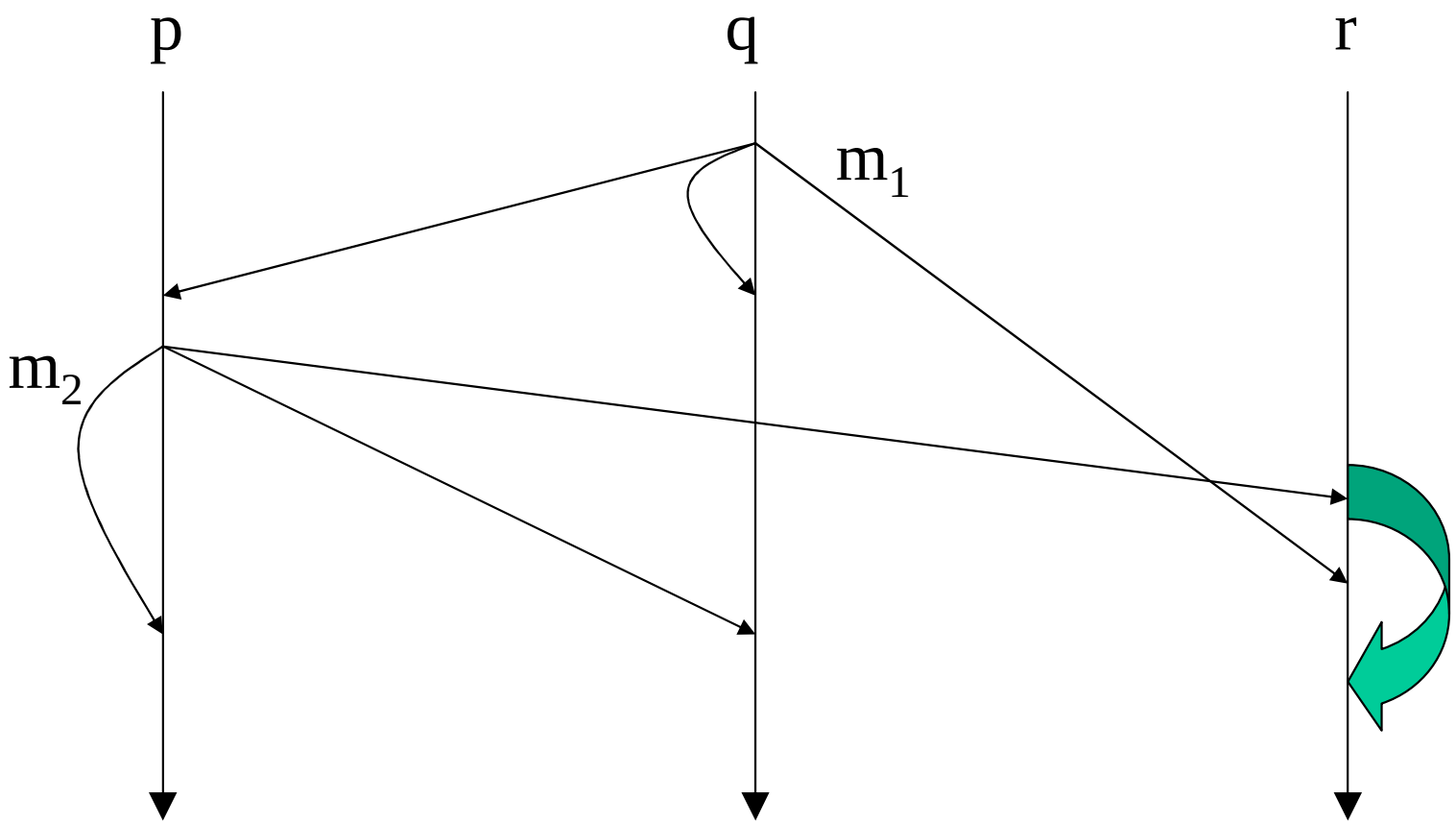
Comunicación a grupos (8)

- Difusión *FIFO*:



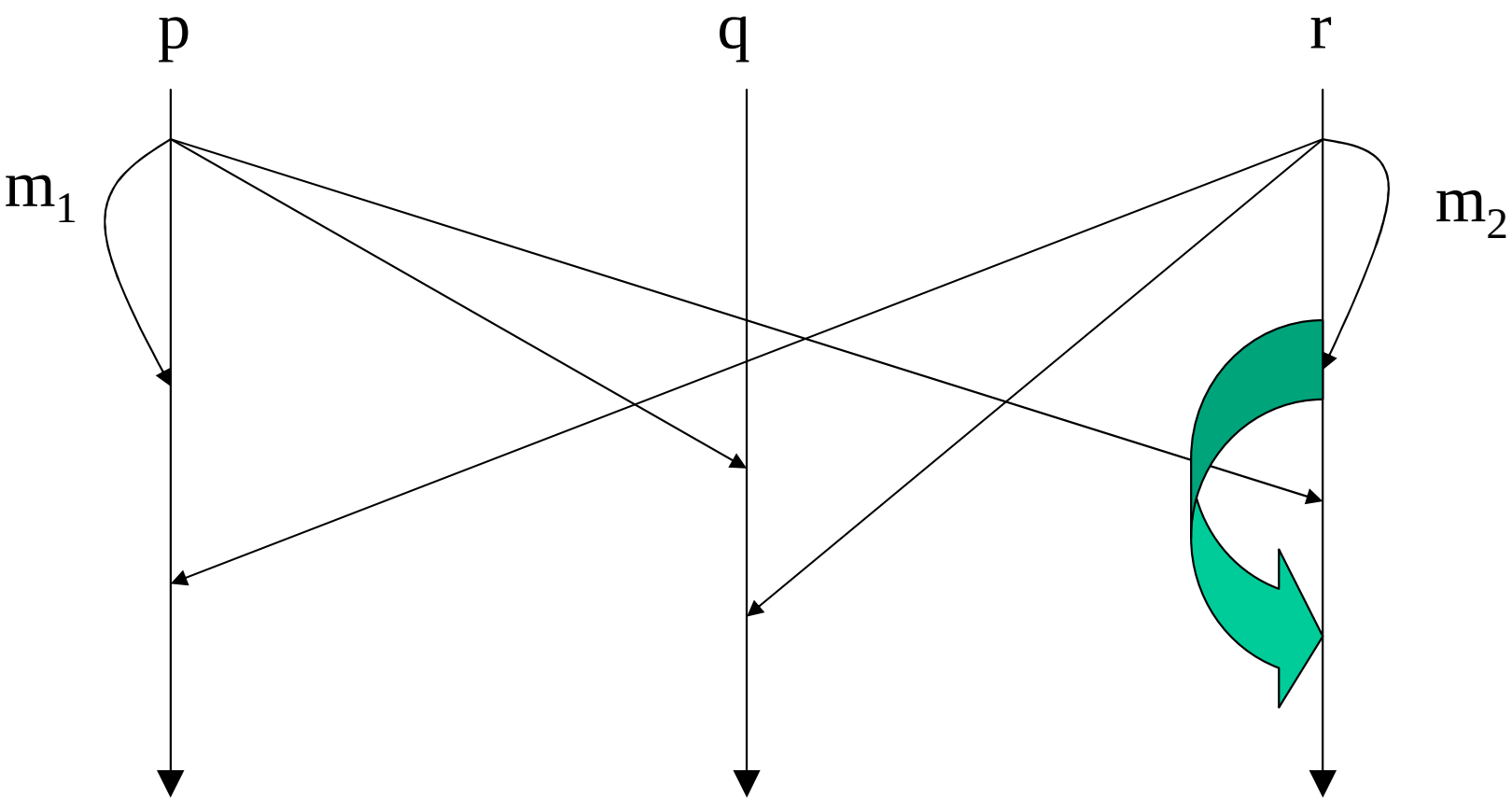
Comunicación a grupos (9)

- Difusión causal (causal → *FIFO*):



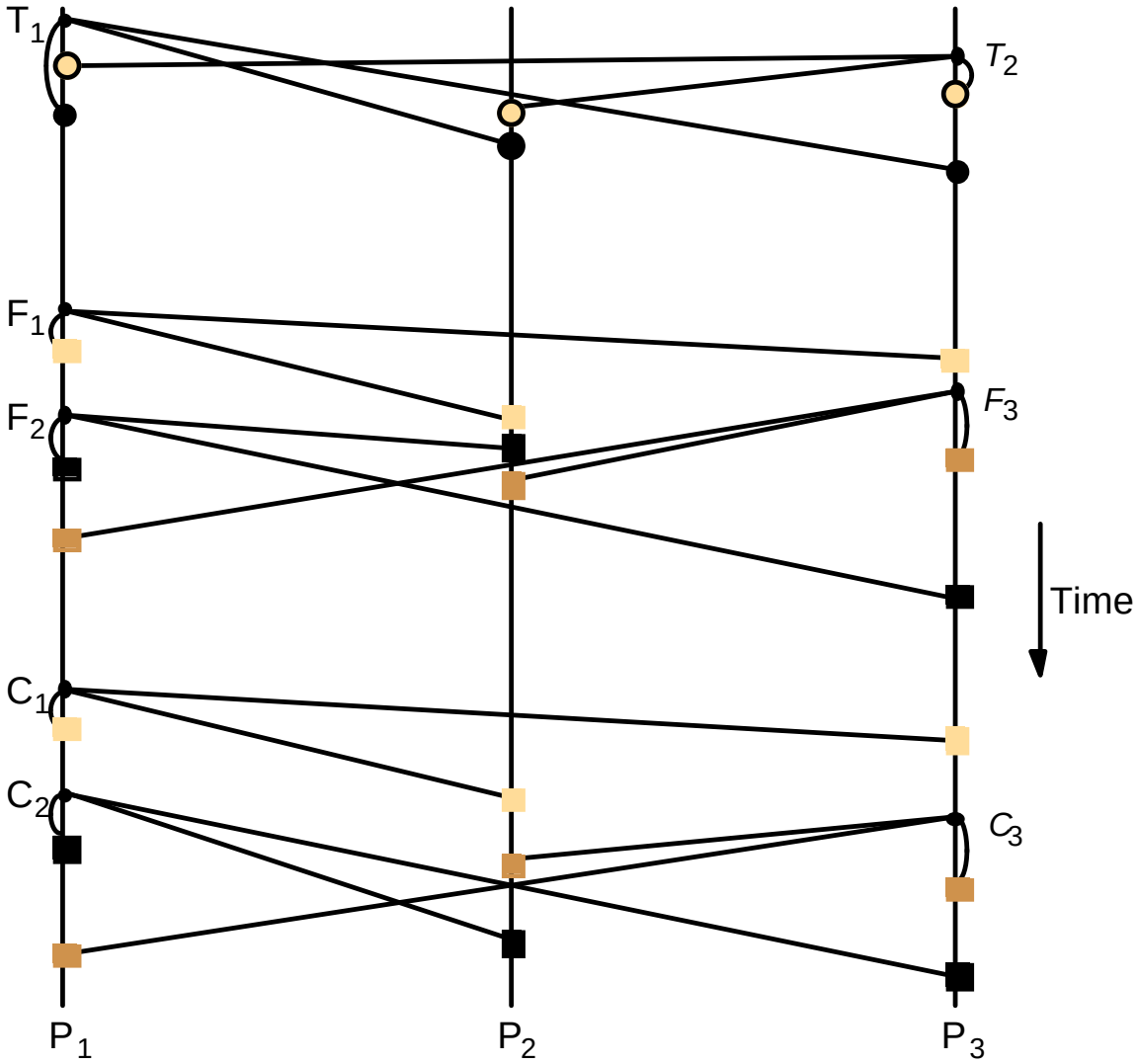
Comunicación a grupos (10)

- Difusión con orden total (“atómica”):



Total, FIFO and causal ordering of multicast messages

Notice the consistent ordering of totally ordered messages T_1 and T_2,
the FIFO-related messages F_1 and F_2 and the causally related messages C_1 and C_3
– and the otherwise arbitrary delivery ordering of messages.



Comunicación a grupos (11)

- Tipos de difusión:
  - difusión no fiable (por ejemplo, *IP-multicast*)
  - difusión fiable
    - sin orden
    - orden *FIFO*
    - con orden causal (por tanto, *FIFO*)
    - con orden total
    - con orden total y FIFO
    - con orden total y causal (por tanto, *FIFO*)
- Ejemplo: *USENET News*
    - fiable (*TCP*), *FIFO* (no causal, no total)
    - causal en la práctica: enviar el mensaje original en la respuesta

Display from bulletin board program

Bulletin board: *os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

Alice

Bob

Bulletin board: *os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	T.L'Heureux	RPC performance
25	M.Walker	Re: Mach
26	A.Hanlon	Re: Microkernels
27	G.Joseph	Microkernels
end		

Comunicación a grupos (12)

- Implementación de difusiones ordenadas:
  - difusión *FIFO*: mediante contadores
  - difusión causal: mediante relojes vectoriales
  - difusión con orden total síncrona:
    - al ejecutar *A-multicast(G, m):* ejecutar *R-multicast(G, m)*, incluyendo en el mensaje *m* la marca de tiempo local *t(m)*
    - al ejecutar *R-deliver(m)*: planificar *A-deliver(m)* para el instante local *t(m)*+Δ
      - Δ: tiempo máximo de transmisión de mensajes en el sistema + desviación máxima de los relojes (se suponen sincronizados)
      - si *A-deliver* no se puede ejecutar en el instante planificado (porque el mensaje se ha entregado fiablemente demasiado tarde), significa que el sistema ha dejado de comportarse de manera síncrona

Comunicación a grupos (13)

- ISIS:
  - *middleware* para comunicación a grupos
  - desarrollado para *UNIX* (Cornell University, 1983)
  - grupos de procesos + gestión de vistas + difusiones
  - *FBCAST*: difusión *FIFO*
  - *CBCAST*: difusión causal
    - basada en relojes vectoriales
  - *ABCAST*: difusión atómica y causal
    - basada en un proceso secuenciador
  - *GBCAST*: gestión de vistas
    - protocolo *flush*

Causal ordering using vector timestamps (ISIS)

Algorithm for group member p_i ($i = 1, 2\ldots, N$)

On initialization

$V_i^g[j] := 0$ ($j = 1, 2\ldots, N$);

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1$;

B-multicast(g, $<V_i^g, m>$);

On B-deliver($<V_j^g, m>$) *from* p_j, *with* $g = group(m)$

place $<V_j^g, m>$ in hold-back queue;

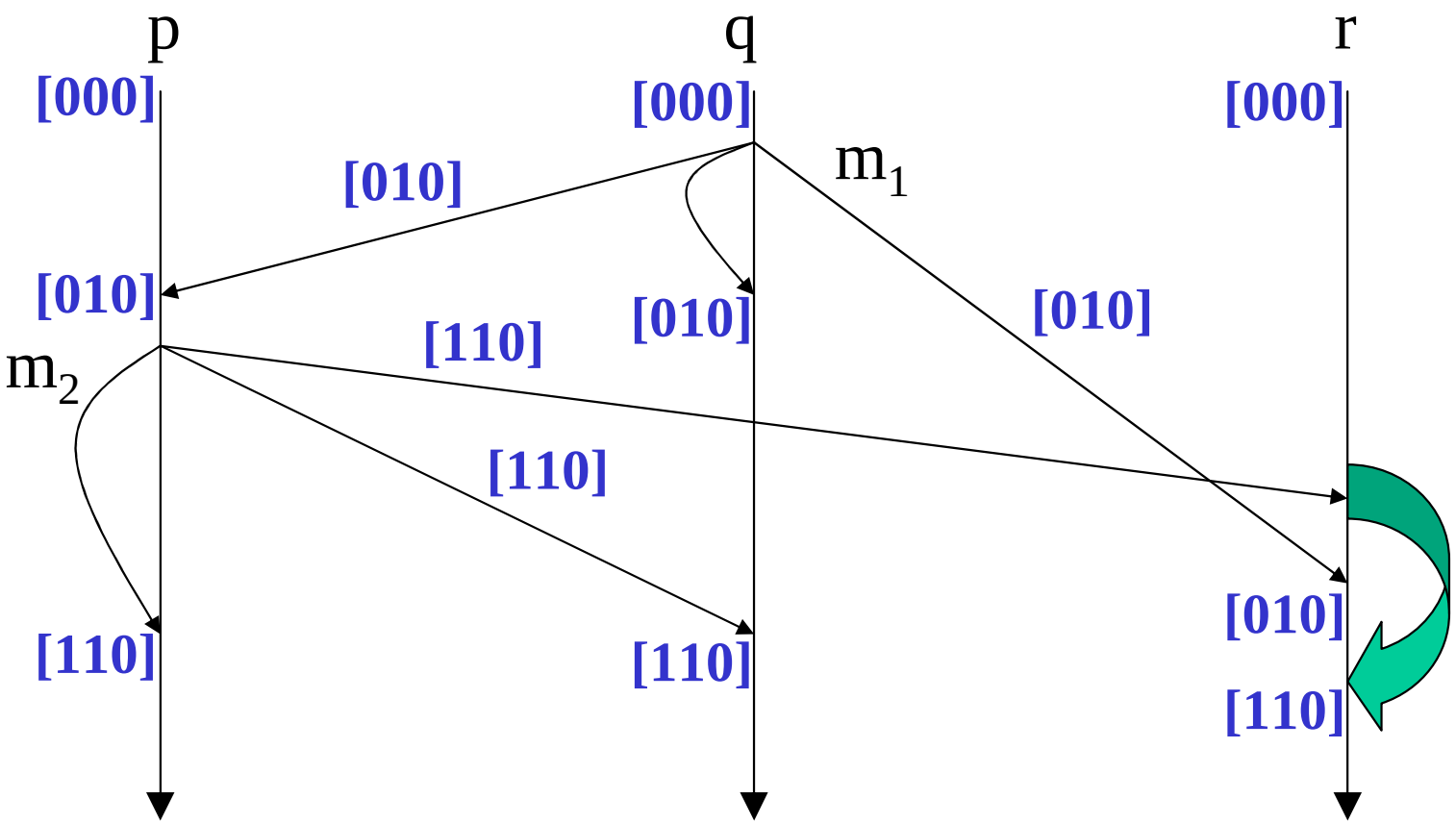
wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \leq V_i^g[k]$ ($k \neq j$);

CO-deliver m; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1$;

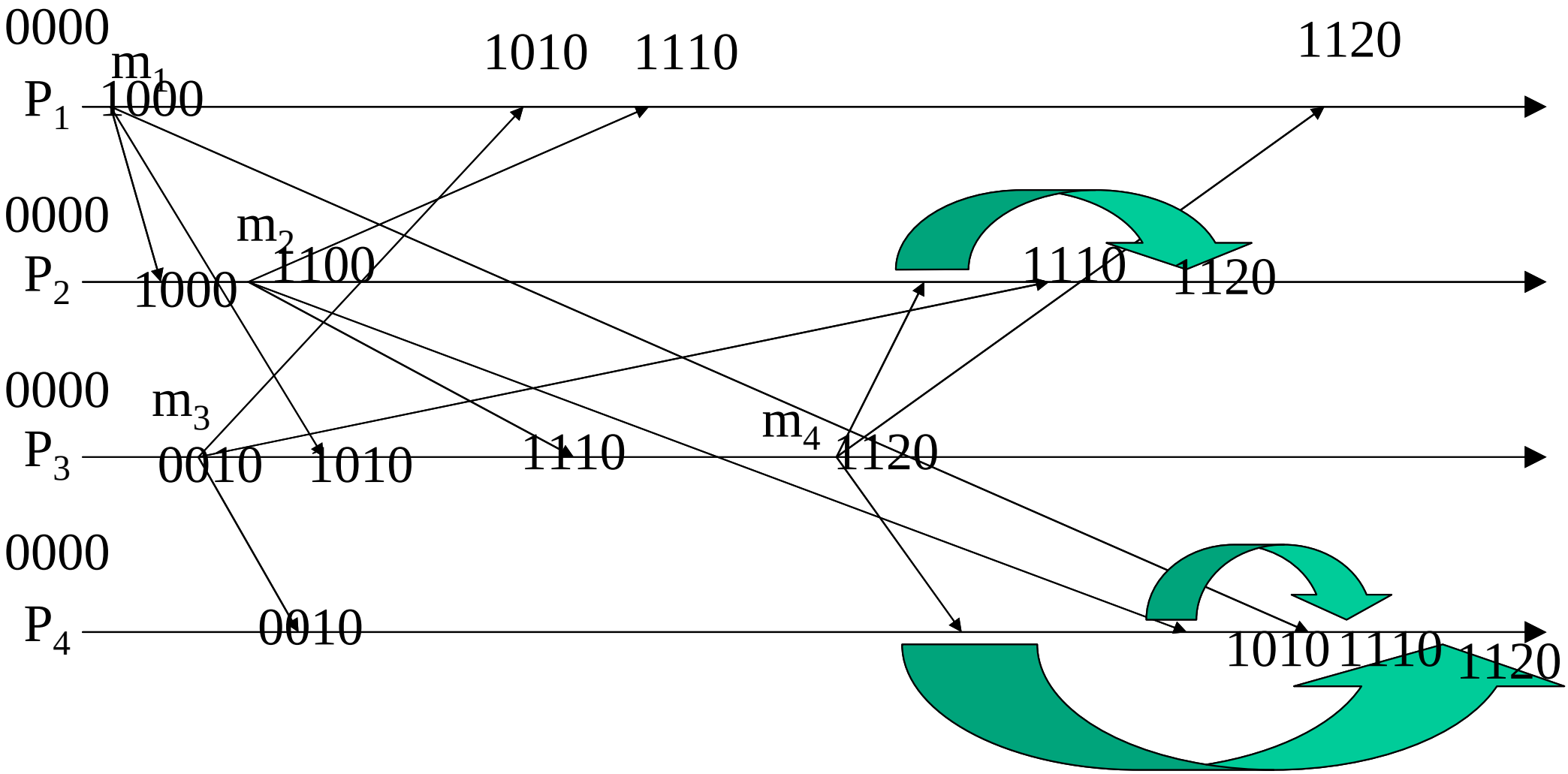
Comunicación a grupos (14)

- Ejemplo de difusión causal:



Comunicación a grupos (15)

- Ejercicio de difusión causal:



Total ordering using a sequencer

1. Algorithm for group member p

On initialization: $r_g := 0$;

To TO-multicast message m to group g
    B-multicast($g \cup \{sequencer(g)\}$, <*m, i*>);

On B-deliver(<*m, i*>) *with* $g = group(m)$
    Place <*m, i*> in hold-back queue;

On B-deliver(m_{order} = <"order", *i, S*>) *with* $g = group(m_{order})$
    wait until <*m, i*> in hold-back queue and $S = r_g$;
    TO-deliver m; // (after deleting it from the hold-back queue)
    $r_g = S + 1$;

2. Algorithm for sequencer of g

On initialization: $s_g := 0$;

On B-deliver(<*m, i*>) *with* $g = group(m)$
    B-multicast(*g*, <"order", *i*, s_g>);
    $s_g := s_g + 1$;