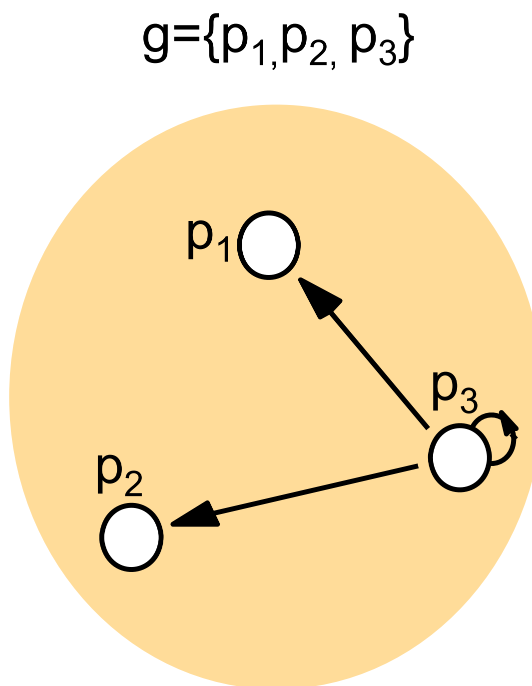
Multicast Communication
(aka. group communication)

Brian Nielsen

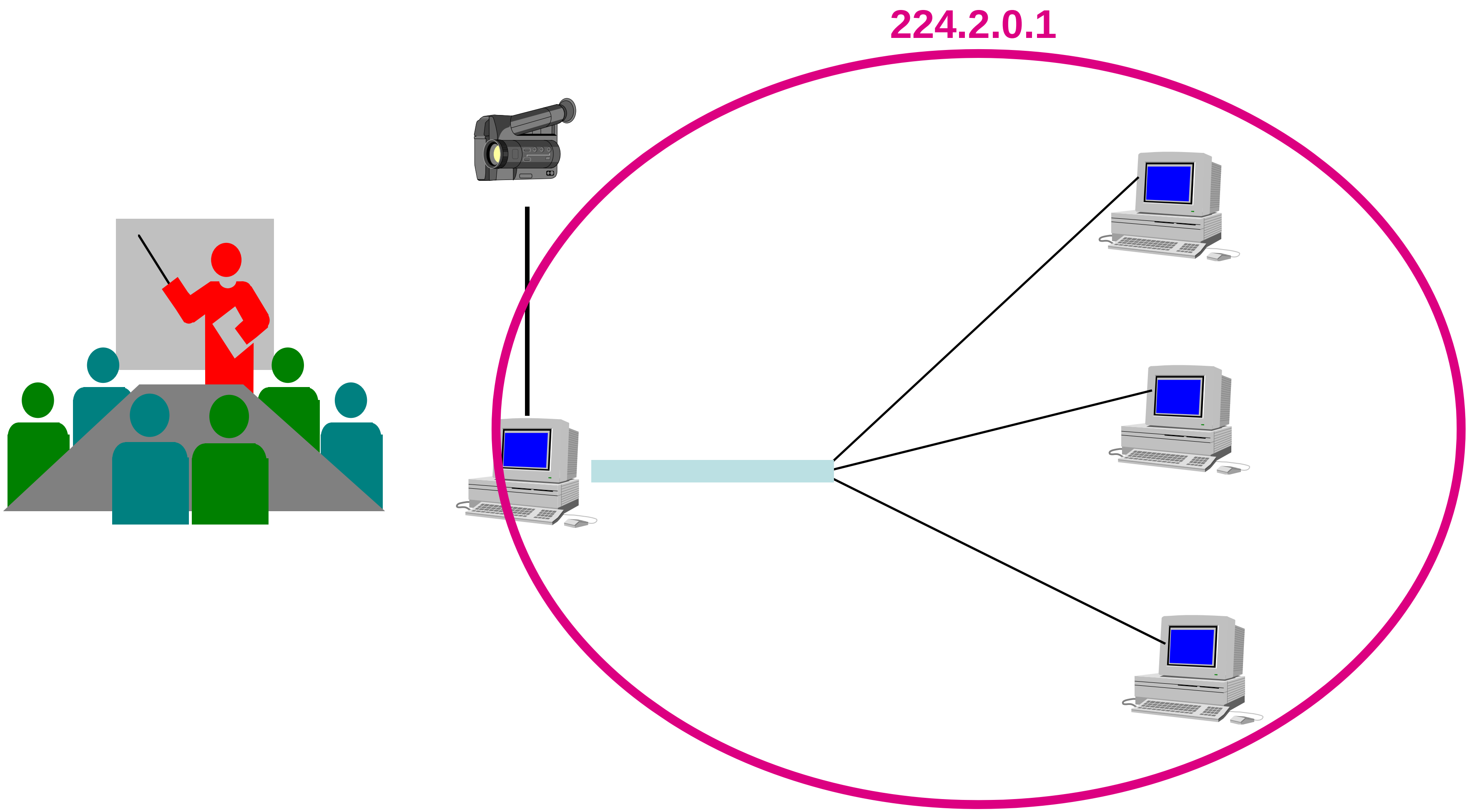
bnielsen@cs.aau.dk

Communication modes in DS

- **Uni-cast**
  - Messages are sent from exactly one process to one process
- **Broad-cast**
  - Messages are sent from exactly one process to all processes on the network.
- **Multi-cast**
  - Messages are sent from exactly one process to several processes on the network ***(named group)***.
- **Any-cast**
  - Message is sent to one (eg "best" or "nearest") of a set of possible receivers
- **Geo-cast:**
  - Message sent to geographically close neighbors



Example: video-conferencing



Multicast address group 224.2.0.1

from UREC, http://www.urec.fr


NYSE
NYSE
NYSE
NYSE
NYSE

Reliable Multicast

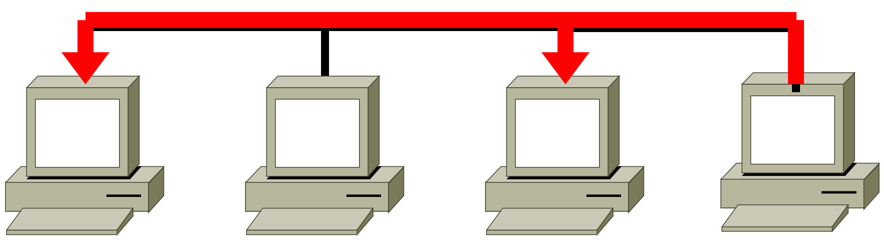
- Bulk Data
  - Corporate data, server cluster (eg. replication), software distribution
  - Files, large memory segments
  - Static
  - Full reliability, no real-time, one sender
- Streaming Data
  - Stock quotes, news, video, audio
  - Messages, a/v formats
  - Dynamic
  - Full-to-none reliability reqs, varying real-time reqs, one/few sender(s)
- Collaborative
  - Whiteboard interaction, multimedia conference, gaming
  - Short messages, a/v formats
  - Dynamic and/or static
  - Full-to-moderate reliability reqs, moderate real-time reqs, many senders

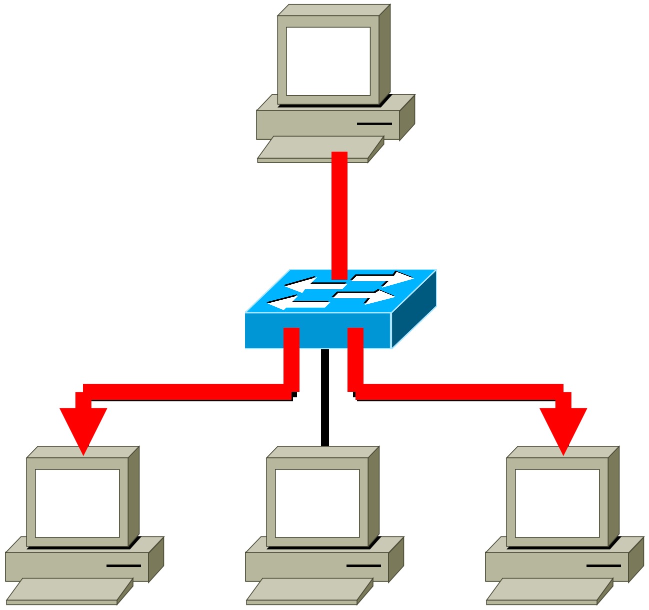
Middleware Systems

- JavaGroups : Reliable, ordered group communication for Java.
- The jGCS library provides a **generic interface for Group Communication**.
- PGM (for MSMQ), Pragmatic General Multicast. RFC 3208
- GROF# : **G**roup **O**riented **F**ramework for C#.
- The Group Communication Toolkit (GCT) is a .NET version of JavaGroups)
- Enterprise "Middleware"
  - Tibco:
    - Rendezvous "reliable broadcast" or multicast
    - 60-second limit, probably Nack mechanism
    - Routing daemons: subnet and wide-area
  - CorbaEvent services (?)
  - DCS

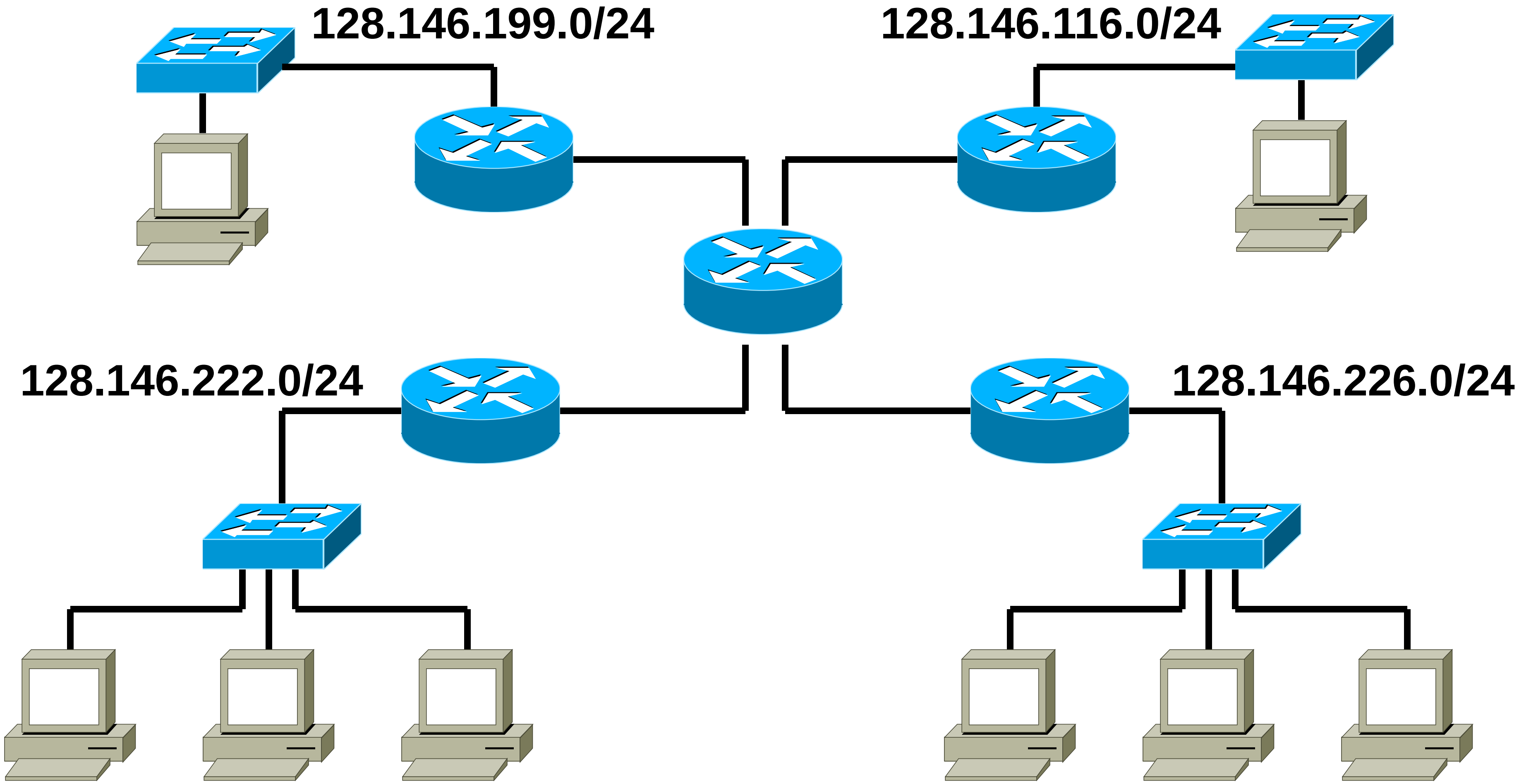
LAN IP Multicast

- Class D IP address
- Hardware support = 1 message is sent

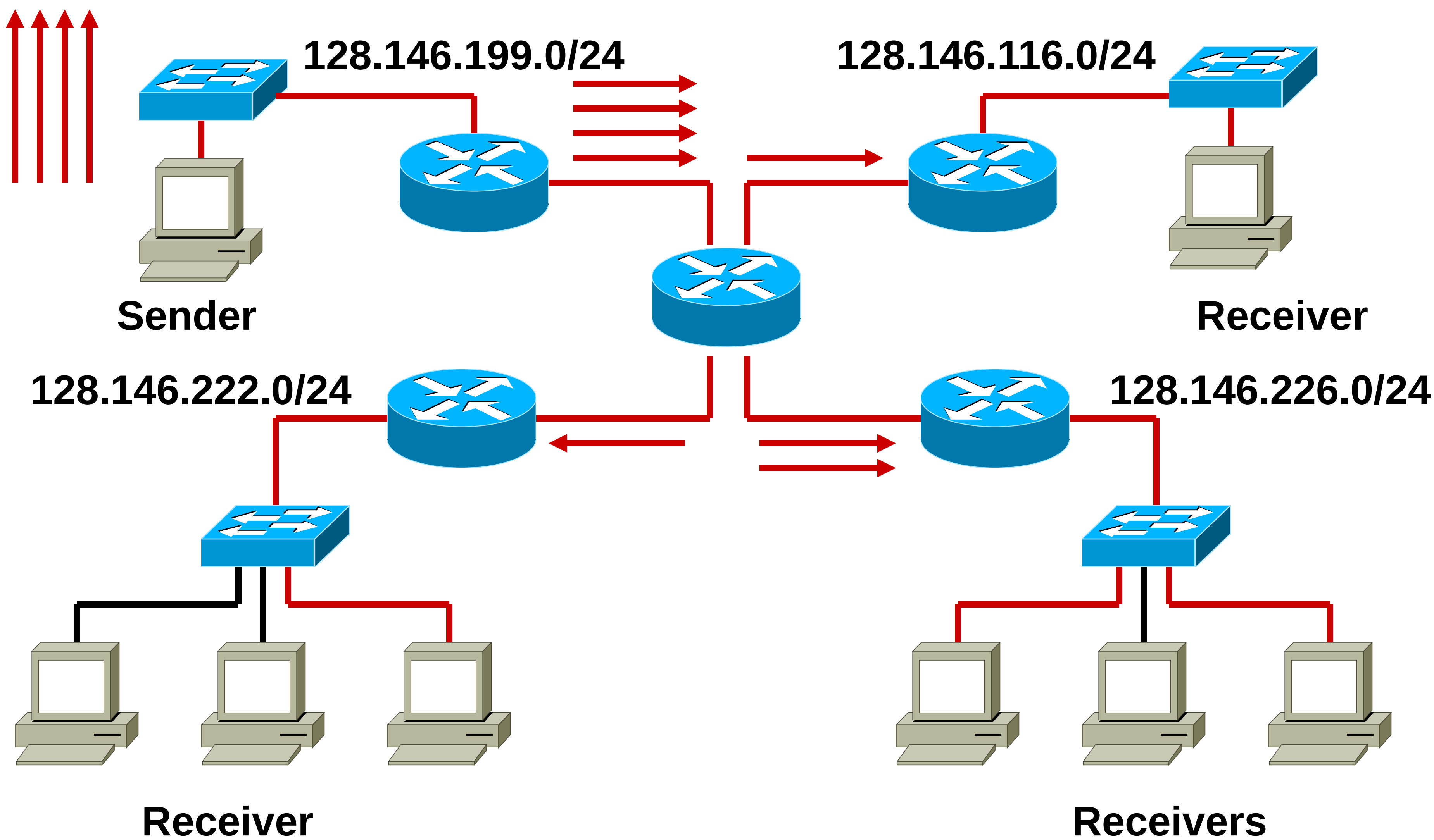




WAN IP-Multicast



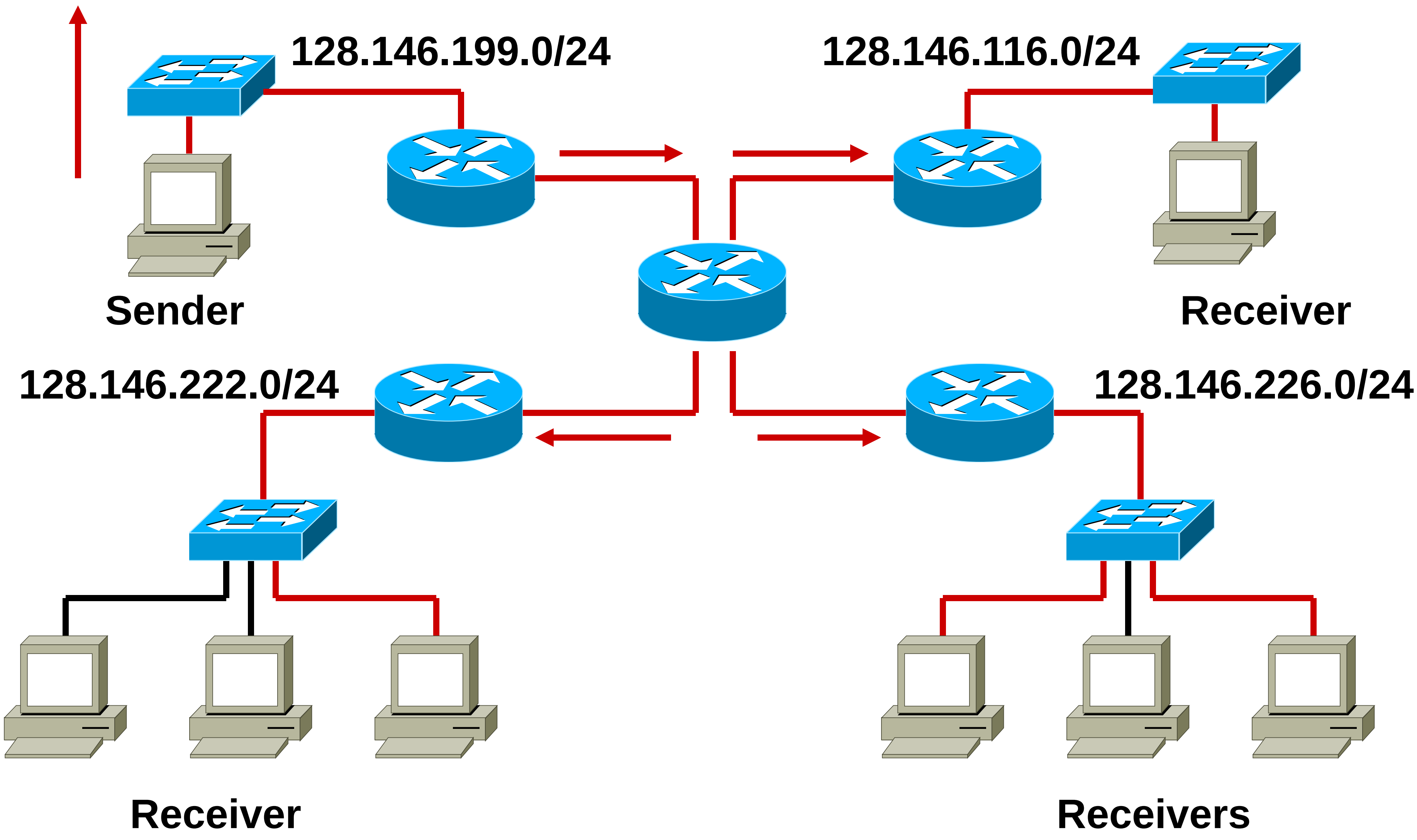
Unicast to multiple receivers



Unicast

- With 4 receivers, sender must replicate the stream 4 times.
- Consider good quality audio/video streams are about 1.5Mb/s (a T1)
- Each additional receiver requires another 1.5Mb/s of capacity on the sender network
- Multiple duplicate streams over expensive WAN links

IP - Multicast



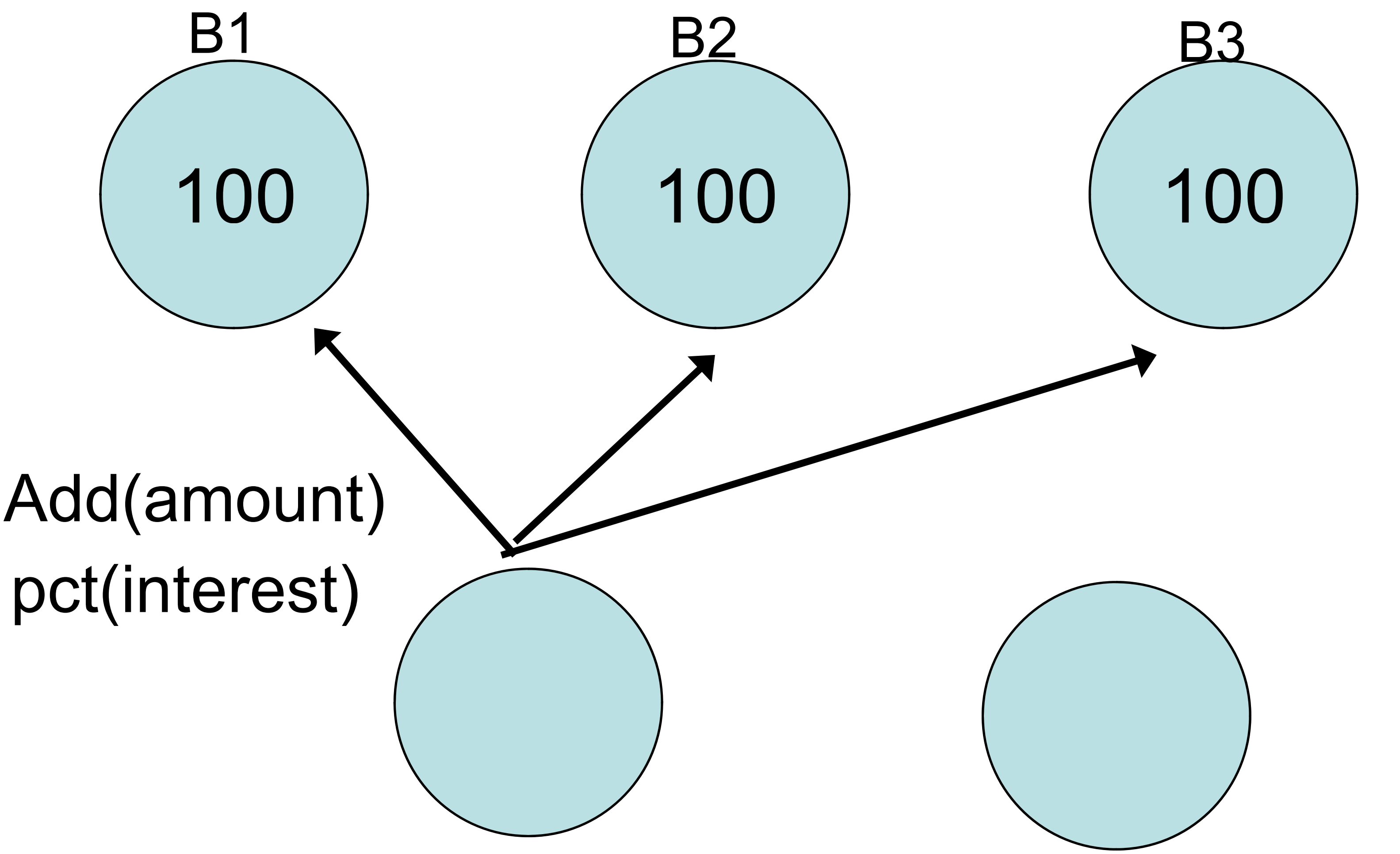
IP-Multicast Efficiency

- IP-multicast more **Efficient** than n sends!
  - Source transmits one stream of data for n receivers
  - Replication happens inside routers and switches
  - WAN links only need one copy of the data, not n copies.
- IP datagram multicast:
  - Hosts join/leave on a class D address
  - IGMP constructs and maintains multicast tree

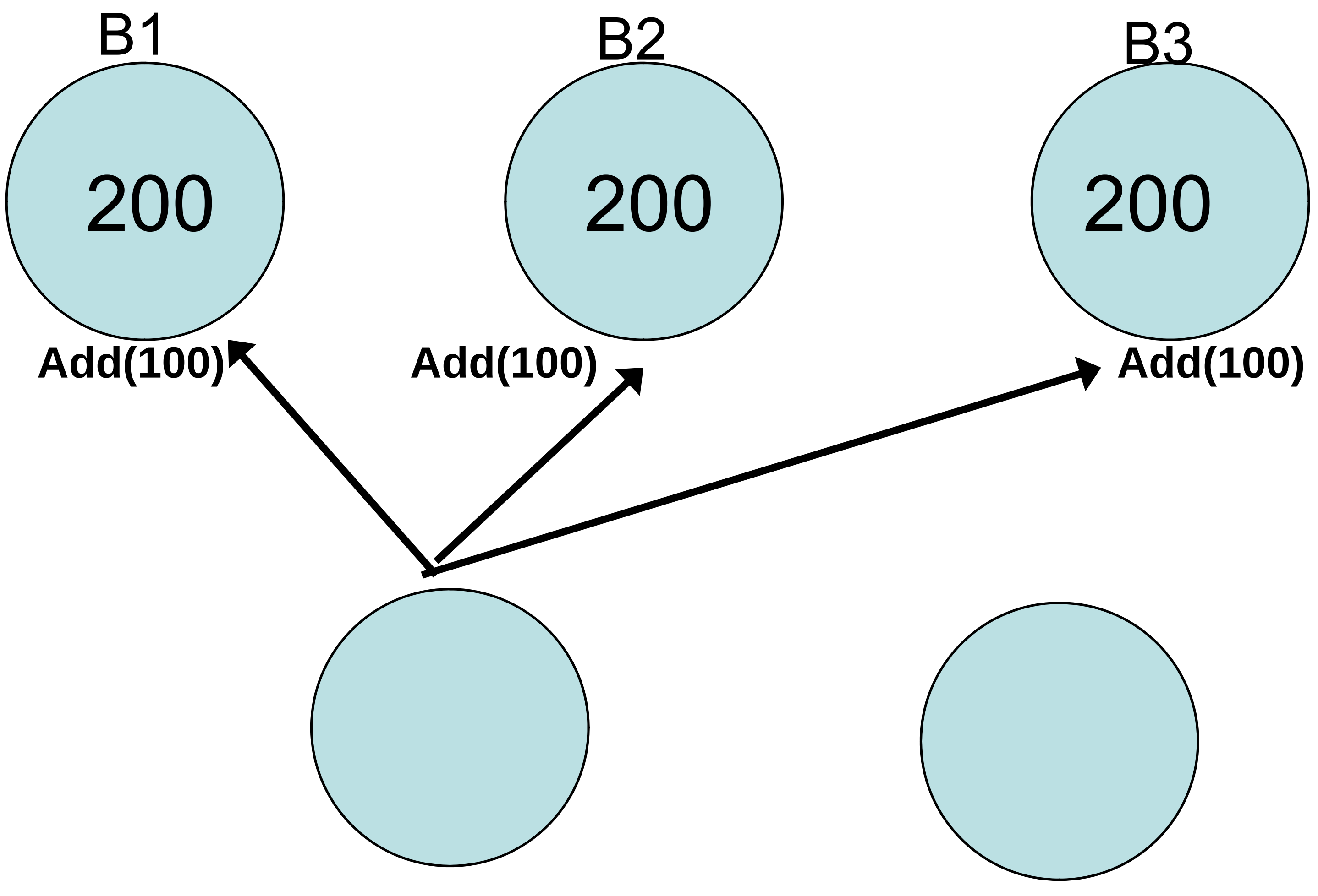
IP-Multicast Failures

- HW- and IP-multicast Failure model ~ UDP
  - Omission failures
    - Delivery to none
    - Delivery to some
  - No ordering guarentees
    - Consequetive multicasts may be received ín different order
      - At same receiving node
      - At different nodes
- **However, ordering and reliability are required by many applications**
- **Reliable & Ordered multicast requires “fancy” algorithms**

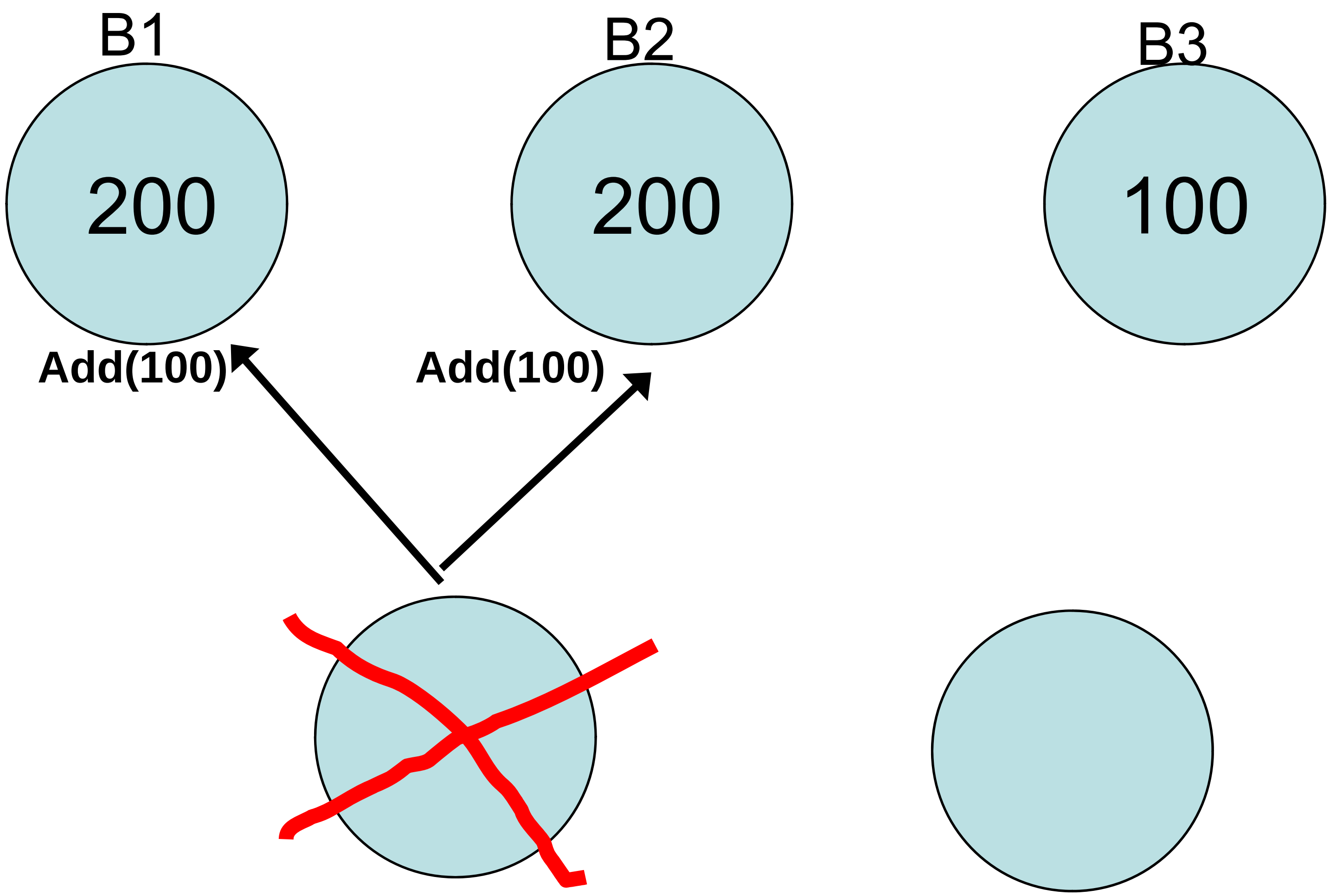
Replicated Bank Account



Replicated Bank Account

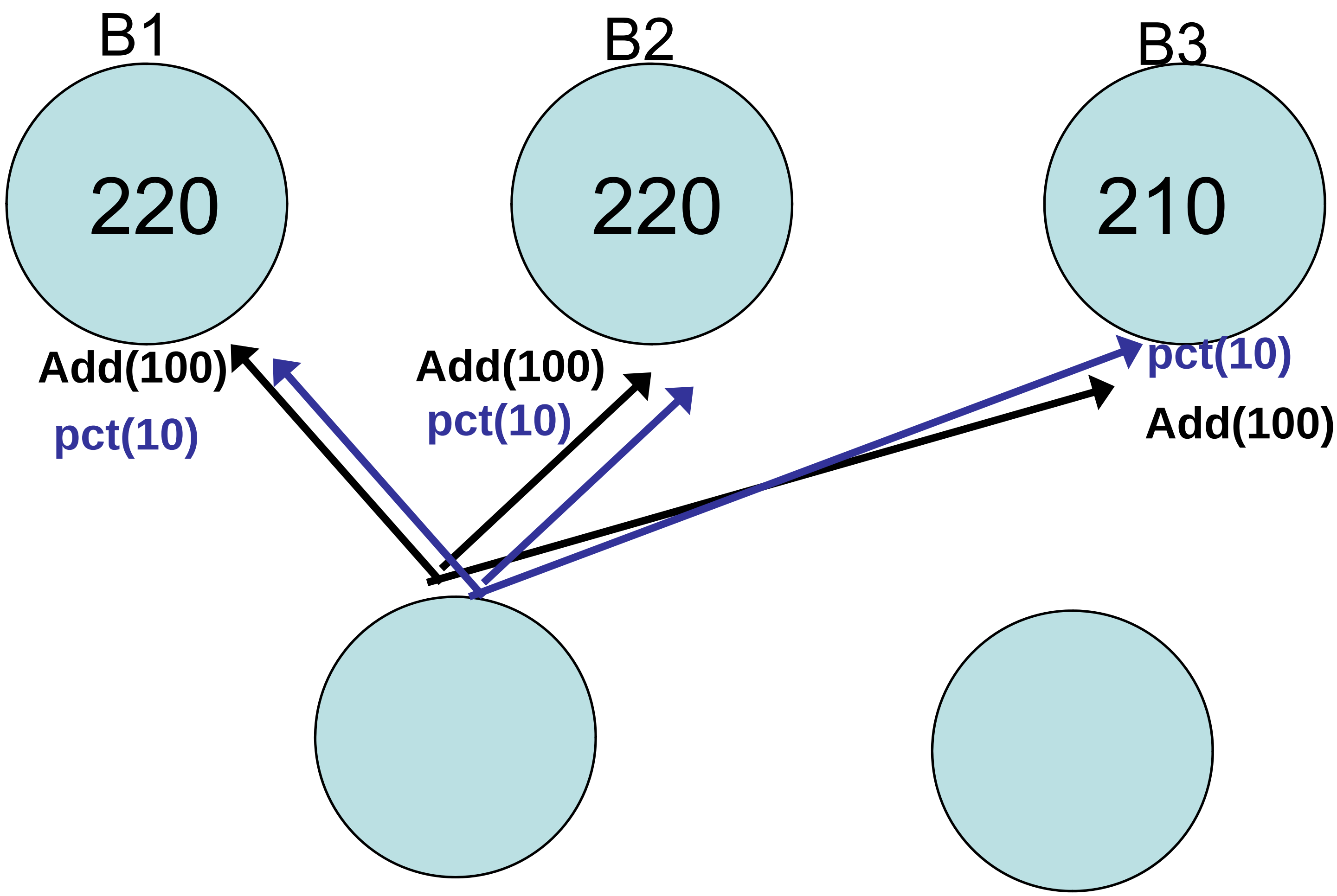


Replicated Bank Account



UNRELIABLE Multicast ⇒ INCONSISTENCY

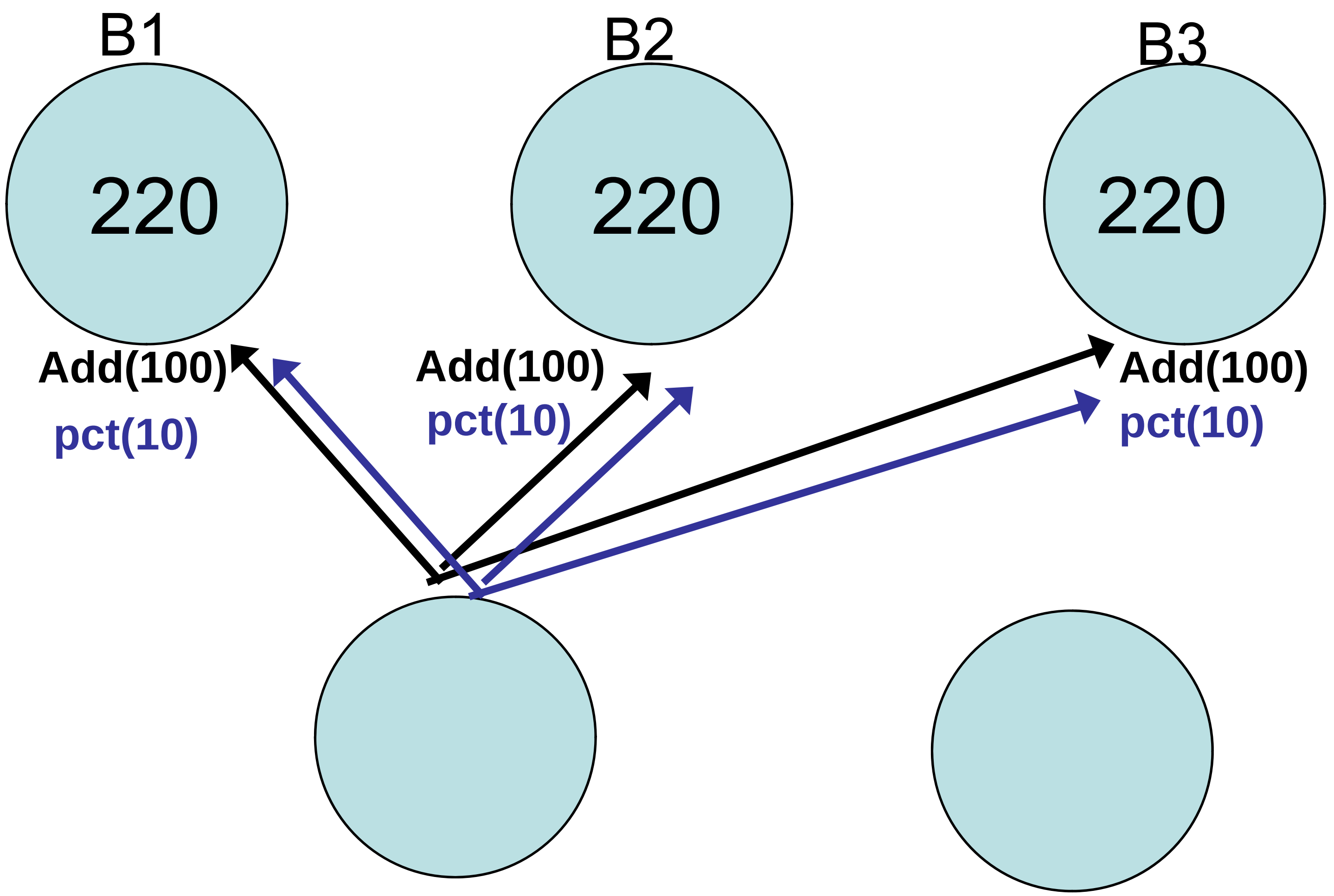
Replicated Bank Account



UNORDERED Multicast ⇒ INCONSISTENCY

Replicated Bank Account

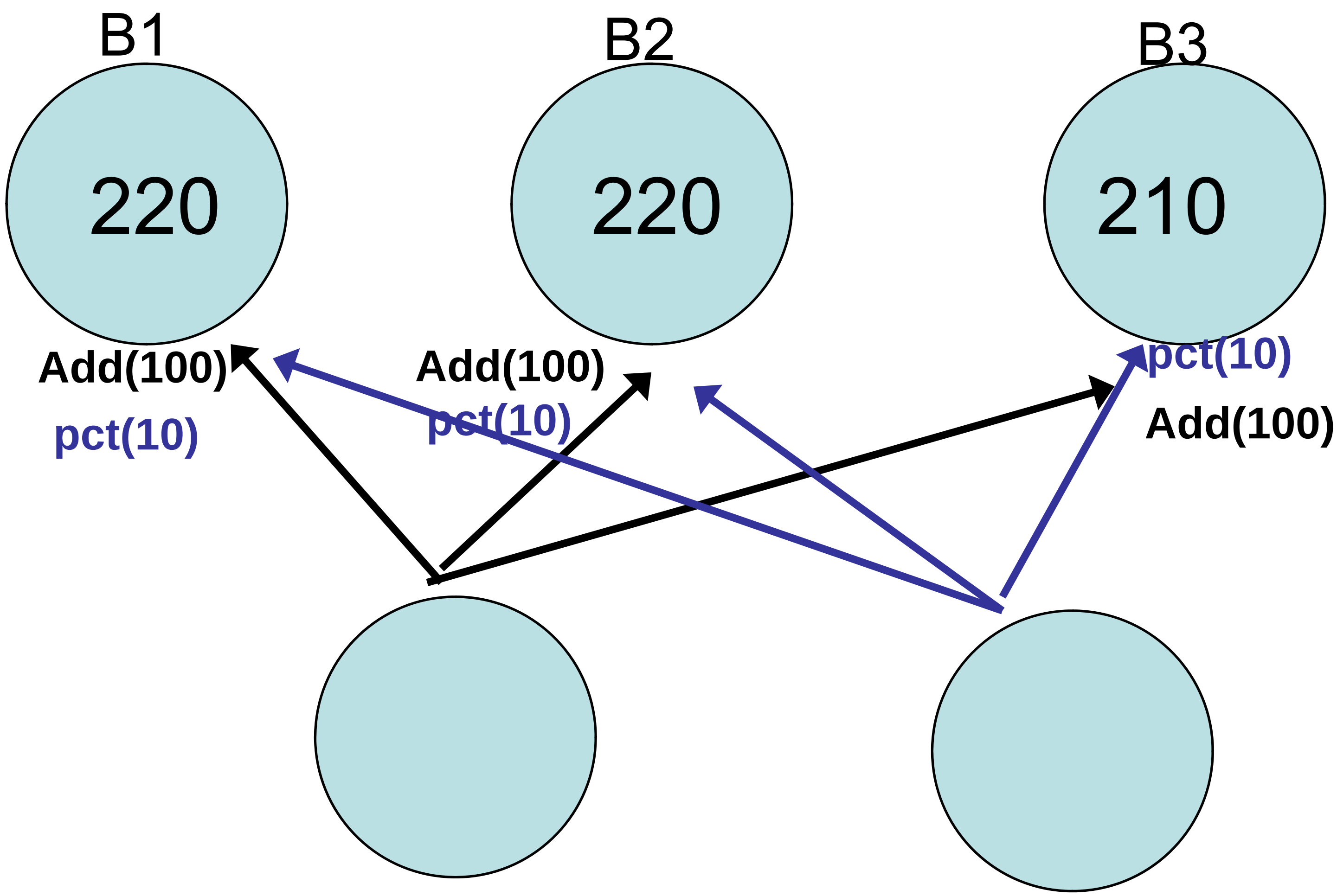
FIFO-ORDERING



FIFO Multicast ⇒ CONSISTENCY??

Replicated Bank Account

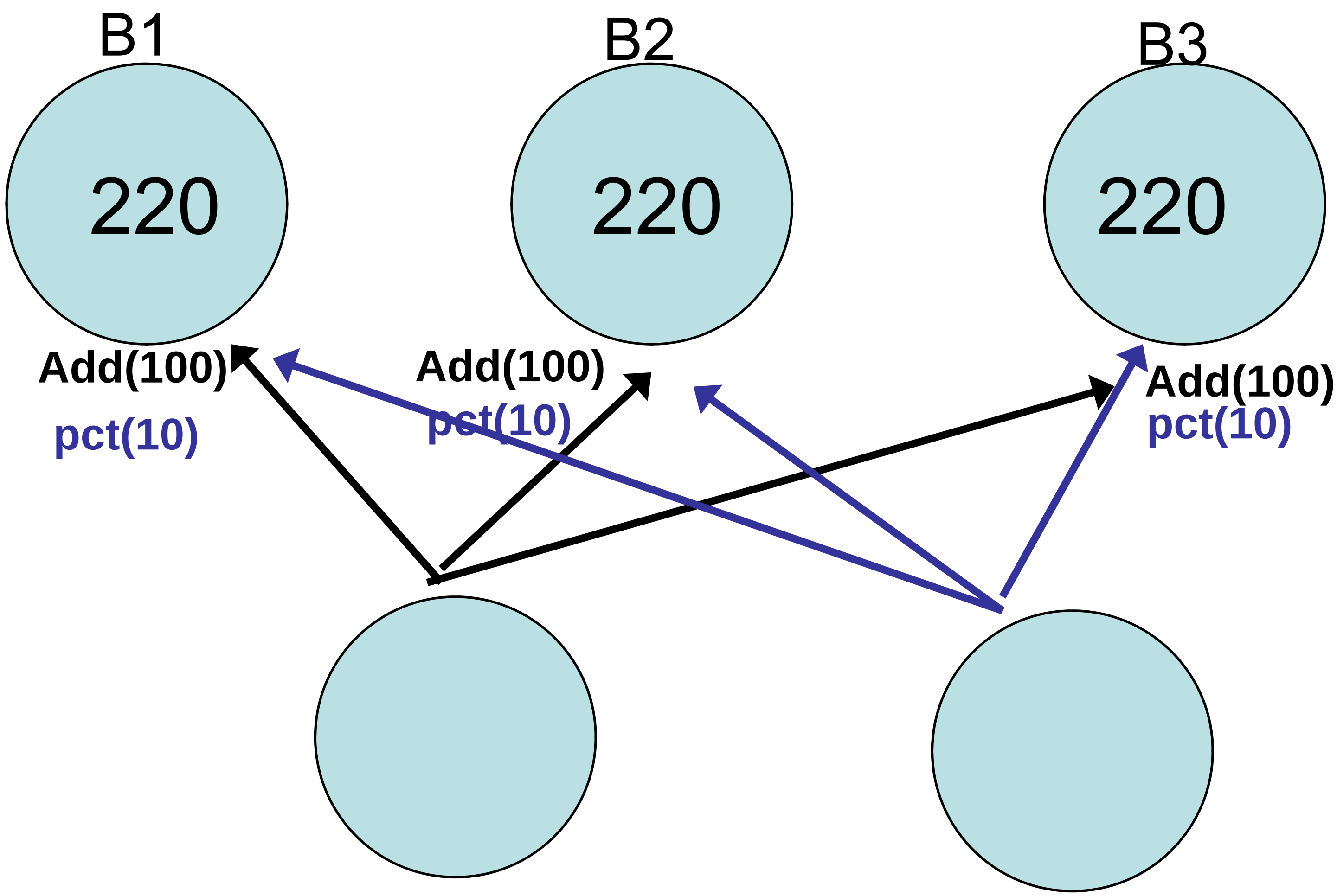
FIFO-ORDERING



FIFO Multicast ⇒ INCONSISTENCY

Replicated Bank Account

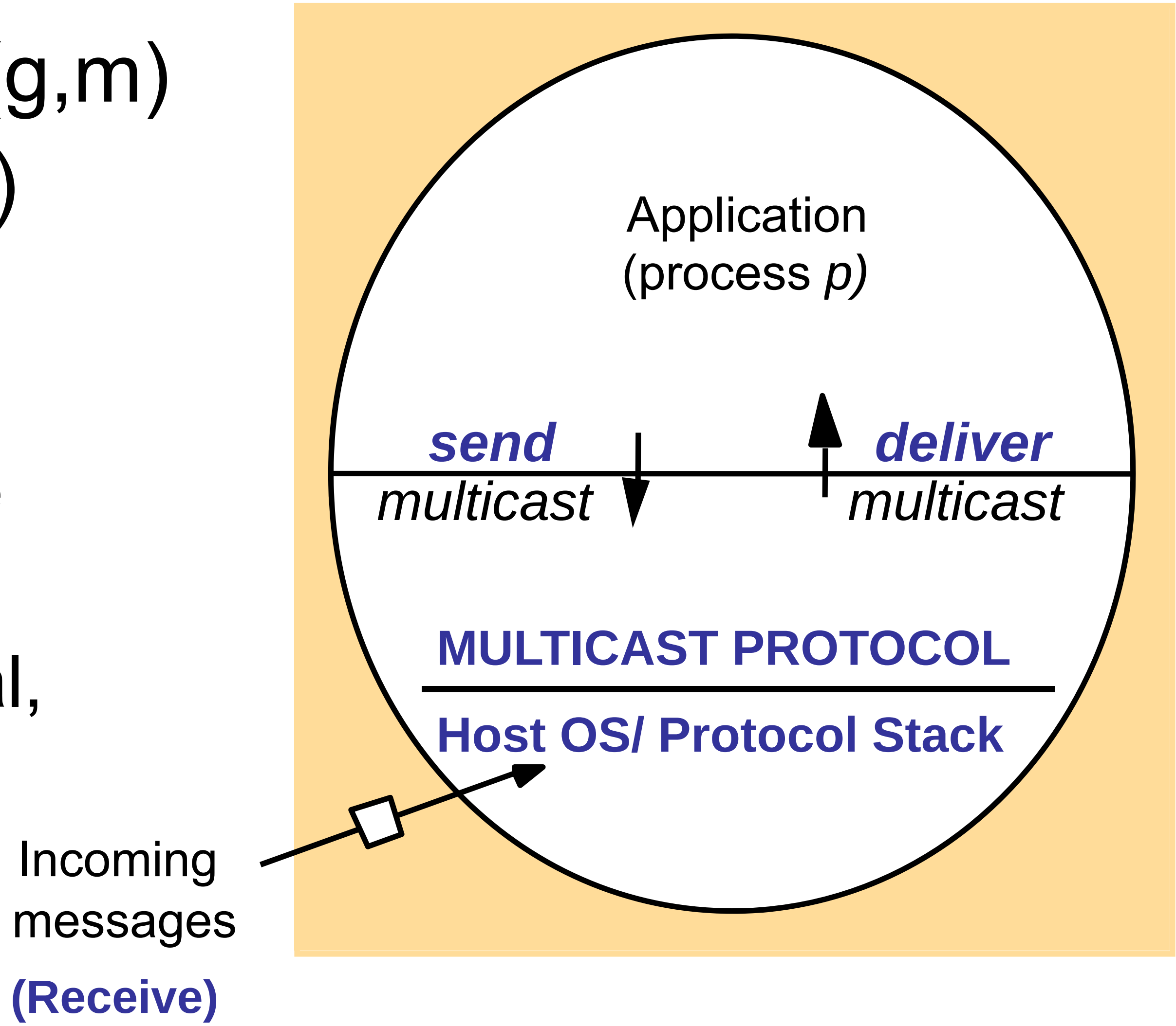
TOTAL ORDERING



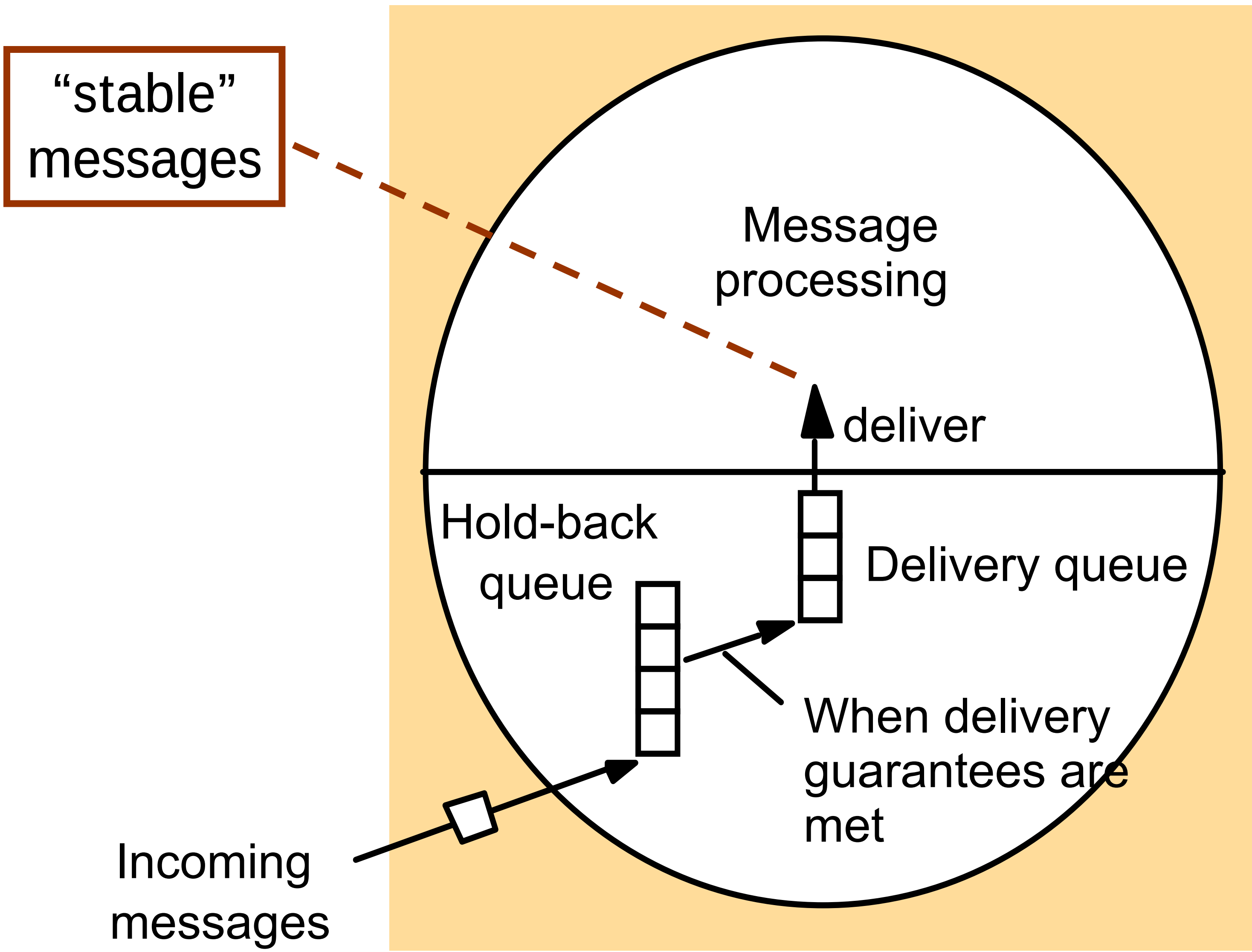
TOTAL Multicast ⇒ CONSISTENCY??

Multicast-API

- **X**-multicast(g,m)
- **X**-deliver(m)
- X is one of
  - B: Basic,
  - R: Reliable
  - FO: FIFO,
  - CO: Causal,
  - TO: Total
  - …



The Hold-back queue



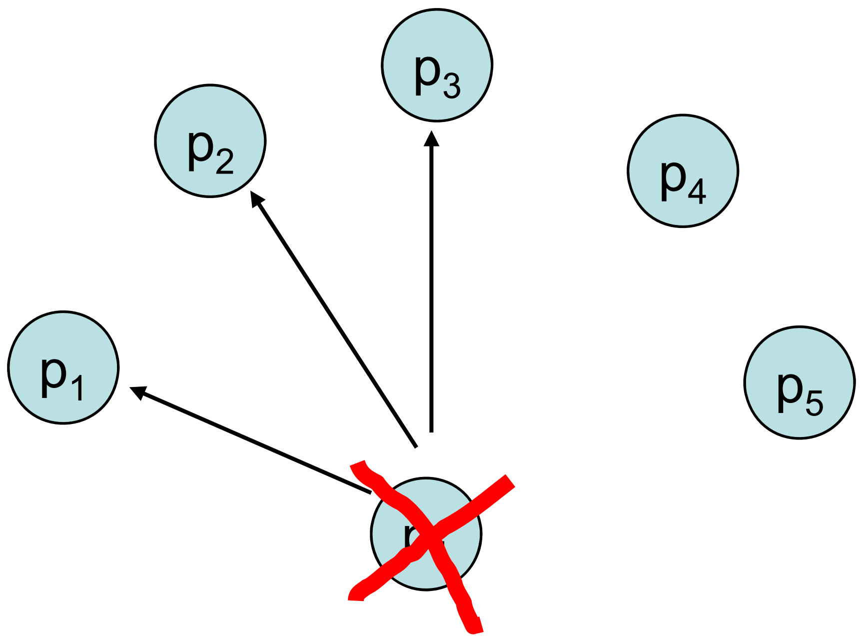
Basic Multicast

- A **basic multicast** primitive guarantees
  - ***All correct process eventually delivers the message, as long as the sender (multicasting process) does not crash***
  - A "**correct**" process = a process that exhibits no failures at any execution point under consideration
  - NB: ***NOT*** satisfied by HW (IP) multicast

- *A straightforward way to implement B-multicast is to use a reliable one-to-one send operation:*

  - *B-multicast(g,m):* for each process *p* in *g*, *send (p,m).*
  - receive(m) at p: *B-deliver(m)*.

B-Multicast



- If P_n crashes, message not delivered in p_4 and p_5
- Hence, Unreliable

Reliable Uni-cast

- ***Integrity:*** A correct process *p* delivers a message *m* at most once. Furthermore, *m* is unmodified and was destined for *p*.
- ***Validity:*** If *m* was sent and the receiver is correct, it eventually delivers *m*.

Reliable 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 *m*, then all other correct processes in *group(m)* will eventually deliver *m.*
- *Liveness*=Validity+agreement

Reliable multicast

Algorithm 1 with B-multicast

On initialization

$Received := \{\};$

For process p to R-multicast message m to group g

$B\text{-}multicast(g, m);$ // $p \in g$ is included as a destination

On B-deliver(m) at process q with g = group(m)

$if\ (m \notin Received)$

$then$

$Received := Received \cup \{m\};$

$if\ (q \neq p)\ then\ B\text{-}multicast(g, m);\ end\ if$

$R\text{-}deliver\ m;$

$end\ if$

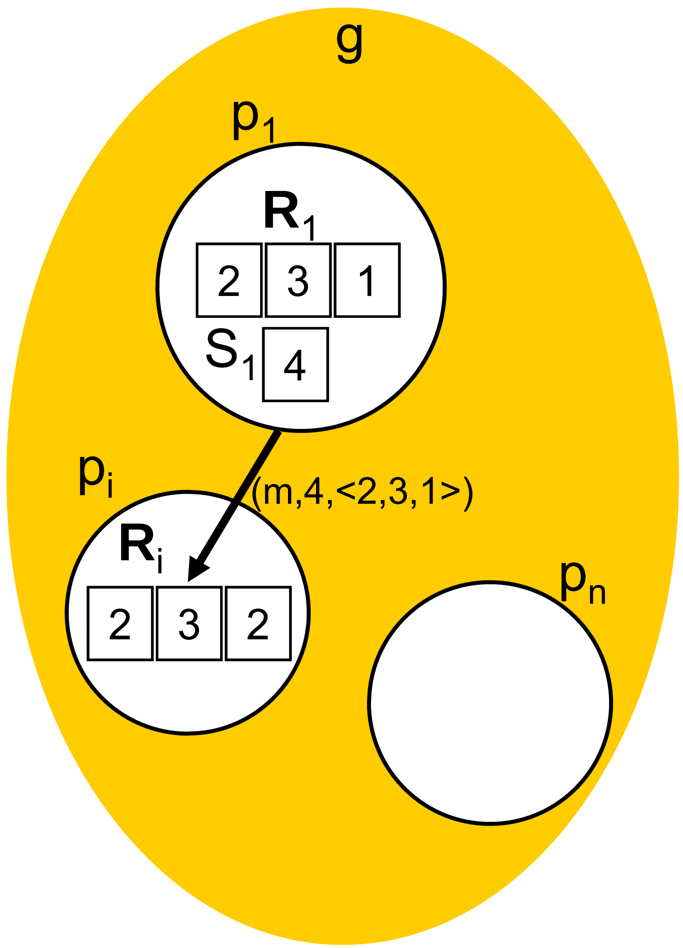
Each R-multicast message is sent |g| times, ie $O(N^2)$.

Reliable multicast

- Correct?
  - Integrity
  - Validity
  - Agreement
- Efficient?
  - NO: each message transmitted |g| times

R-multicast using IP multicast

- Each process maintains sequence numbers
  - S^p_g *next message to be sent*
  - R^q_g (for all $q \in g$) *latest message delivered from q*
- On ***R-multicast*** of *m* to group *g*, attach S^p_g and all pairs $<q, R^q_g>$
- ***R-deliver*** in process *q* happens iff $S_m = R^p_g + 1$
  - if $S_m < R^p_g + 1$, process *q* has seen the message before,
  - if $S_m > R^p_g + 1$ or if $R_m > R^p_g$ for some pair $<q, R_m>$ in message a message has been lost



R-multicast using IP multicast

Data structures at process p:

S_g^p : sending sequence number

R_g^q : sequence number of the latest msg p delivered from q (for each q)

On initialization:

$S_g^p = 0$, $\mathbf{R}_g^q = -1$, for all $q \in g$

For process p to R-multicast message m to group g

IP-multicast $(g, <m, S_g^p, <\mathbf{R}_g> >)$

S_g^p ++

On IP-deliver $(<m, S, <\mathbf{R}>>)$ at q from p

(continued)

R-multicast using IP multicast

On IP-deliver (<m, S, <**R**>>) at q from p

save m

if $S = R_g^p + 1$

then R-deliver (m)

R_g^p ++

check hold-back queue

else if $S > R_g^p + 1$

then store m in hold-back queue

request missing messages endif

endif

if $\exists p.\ r_g^p \in \mathbf{R}$ and $r_g^p > R_g^p$ then request missing messages endif

R-multicast using IP multicast

- 3 processes in group: P, Q, R
- State of process:
  - S: Next sequence number
  - R_q: Already delivered from Q
  - Set of Stored messages!
- Presentation:

P: 2
Q: 3 R: 5
< >

R-multicast using IP multicast

- Initial state:

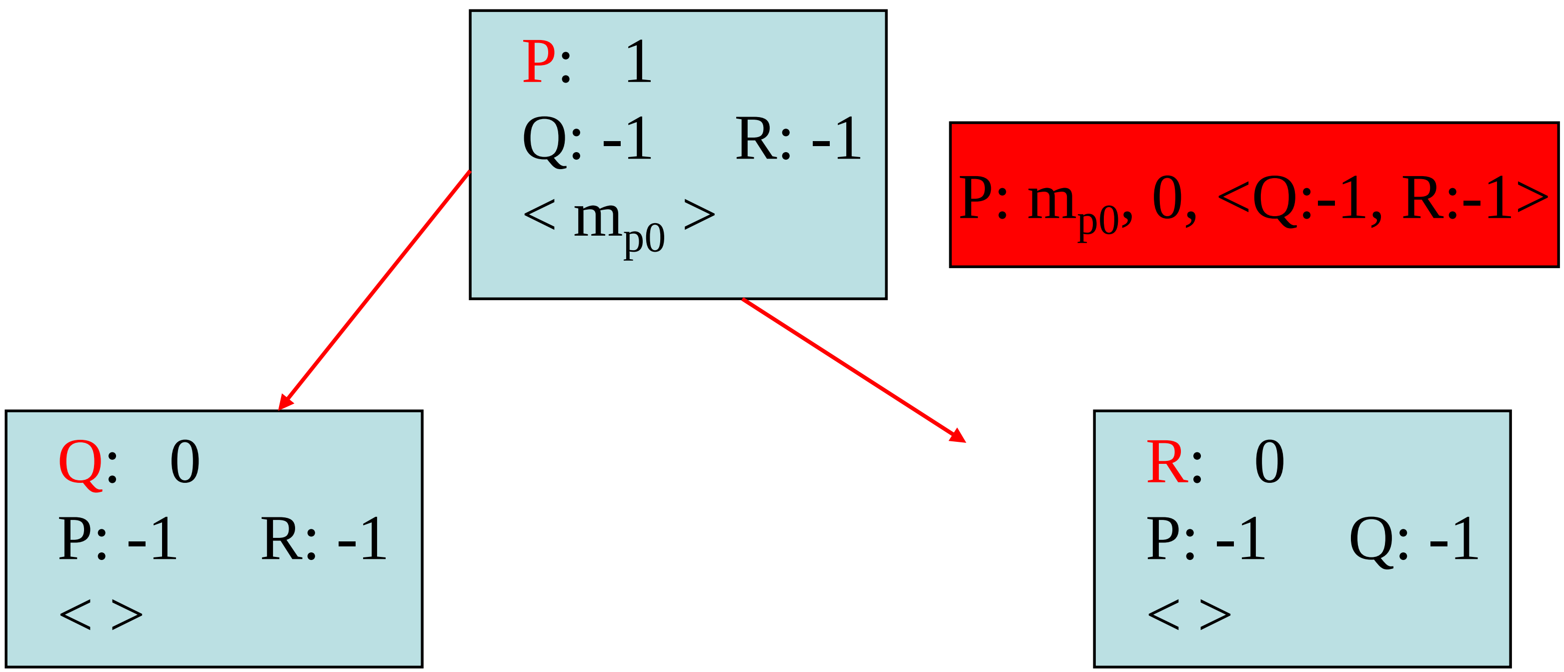
P: 0
Q: -1 R: -1
< >

Q: 0
P: -1 R: -1
< >

R: 0
P: -1 Q: -1
< >

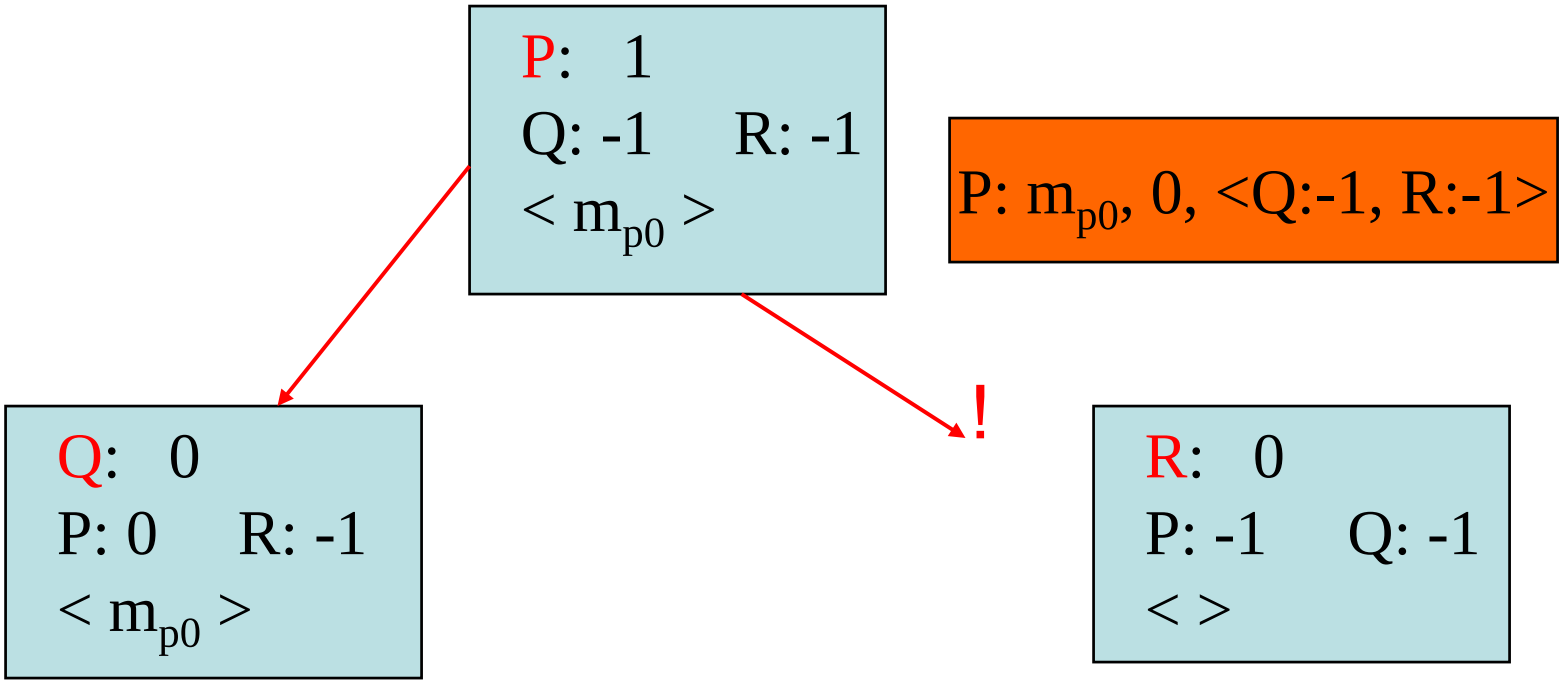
R-multicast using IP multicast

- First multicast by P:



R-multicast using IP multicast

- Arrival multicast by P at Q:



R-multicast using IP multicast

- New state:

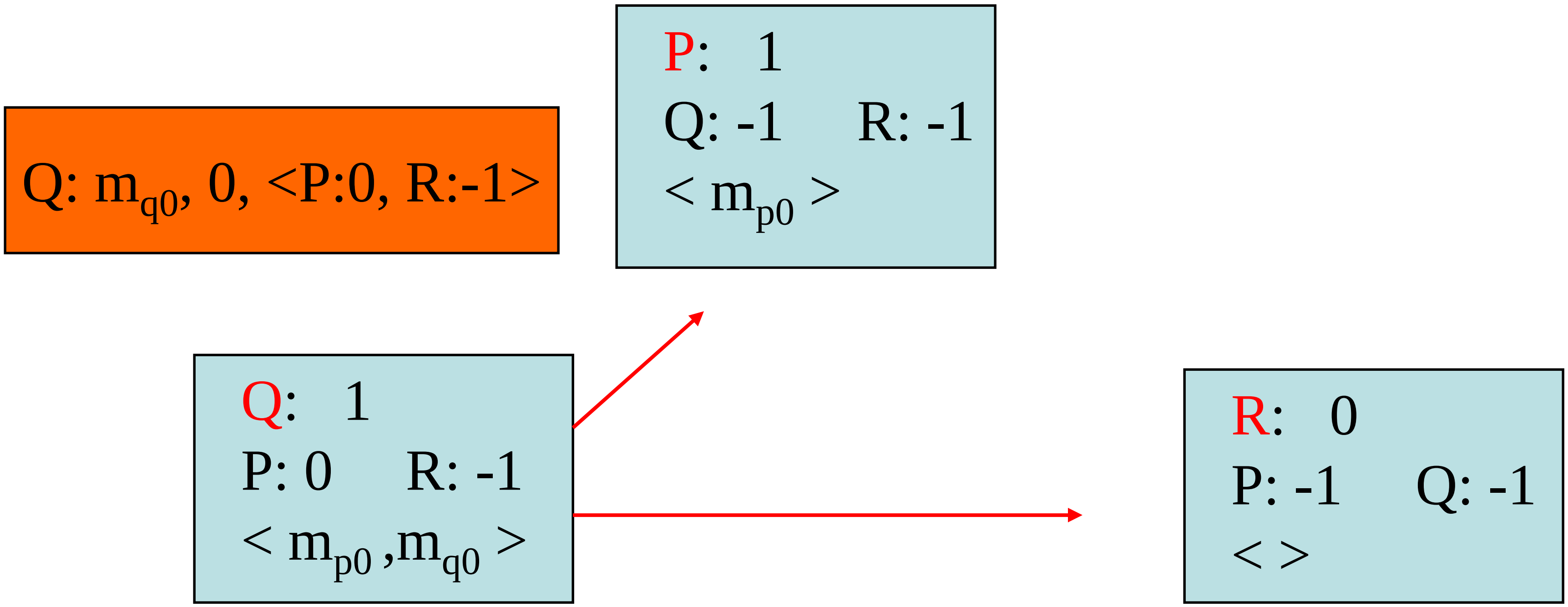
P: 1
Q: -1 R: -1
$< m_{p0} >$

Q: 0
P: 0 R: -1
$< m_{p0} >$

R: 0
P: -1 Q: -1
$< >$

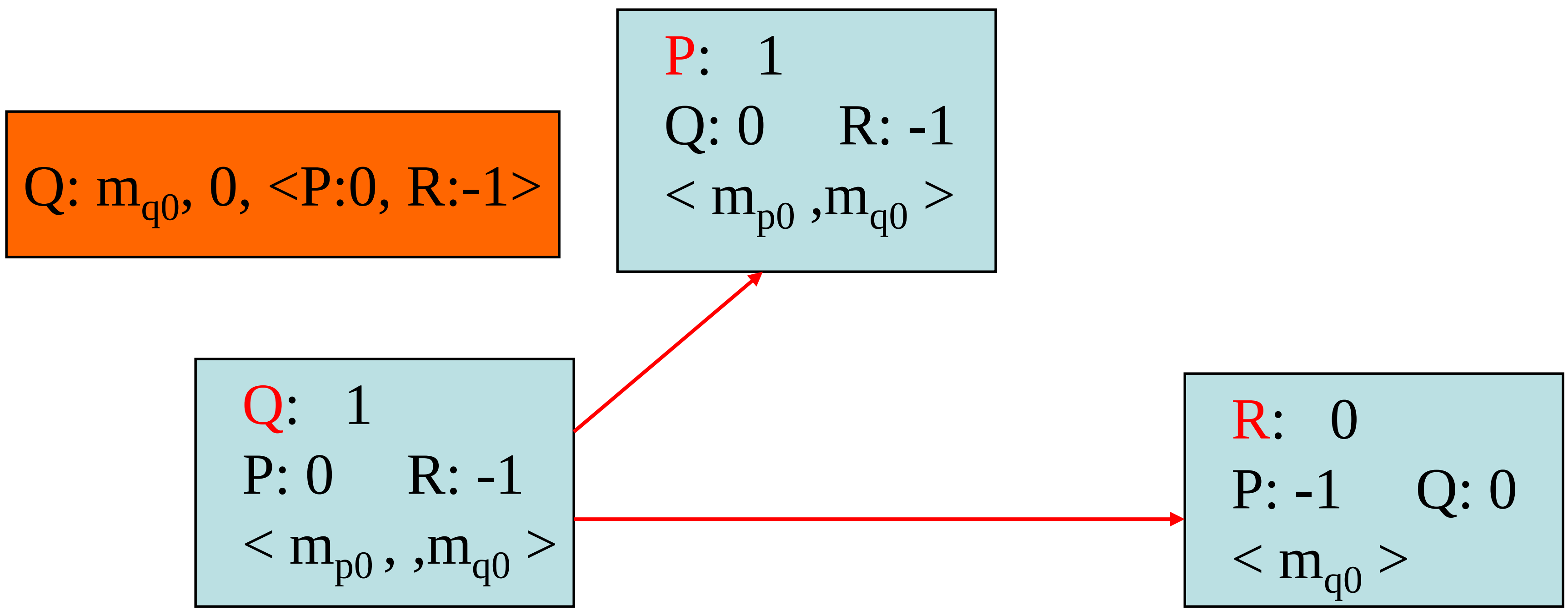
R-multicast using IP multicast

- Multicast by Q:



R-multicast using IP multicast

- Arrival of multicast by Q:



R-multicast using IP multicast

- R detects missing message!
- When to delete stored messages?

P: 1
Q: 0 R: -1
$< m_{p0}, m_{q0} >$

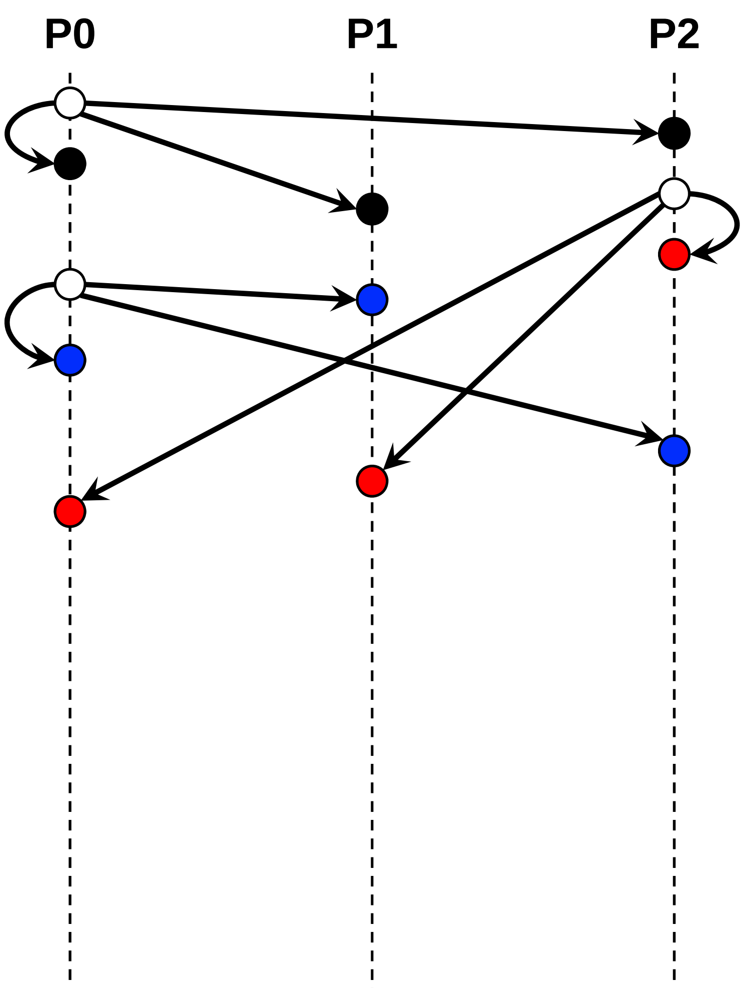
Q: 1
P: 0 R: -1
$< m_{p0}, , m_{q0} >$

R: 0
P: -1 Q: 0
$< m_{q0} >$

R-multicast using IP multicast

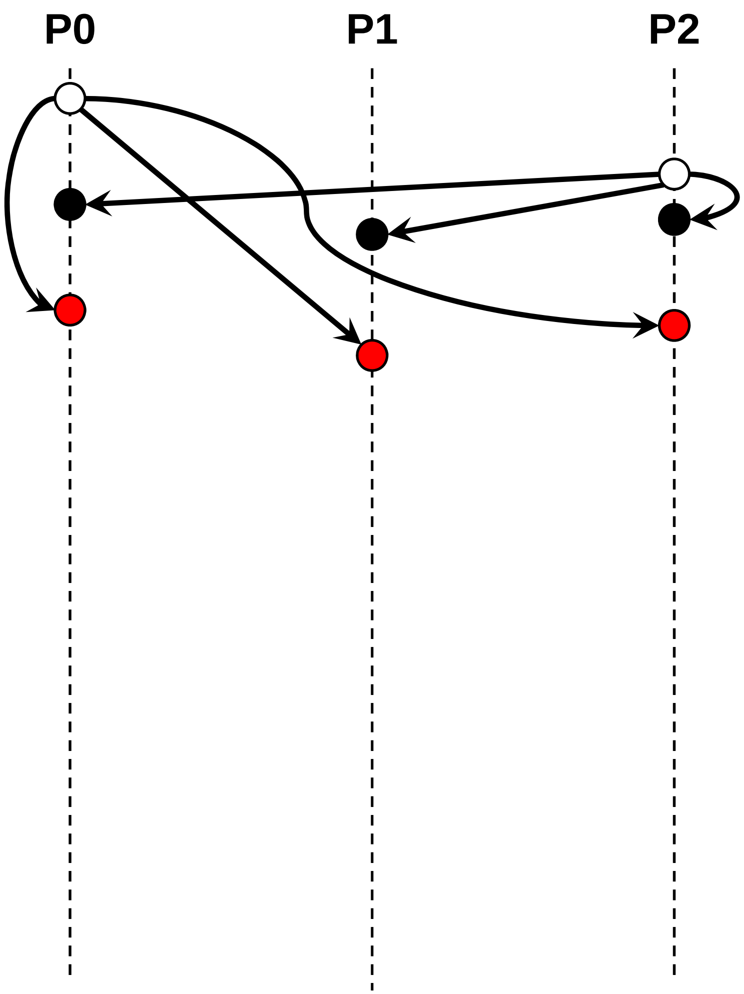
- Correct?
  - Integrity:
    - seq numbers (duplicate detection) + checksums in IP multicast
  - Validity:
    - Self delivery assumed for IP
  - Agreement:
    - if missing messages are detected
    - $\Rightarrow$ Correct processes multicasts indefinitely
    - if copy of message remains available
  - IMPROVE IT!

Ordered multicast



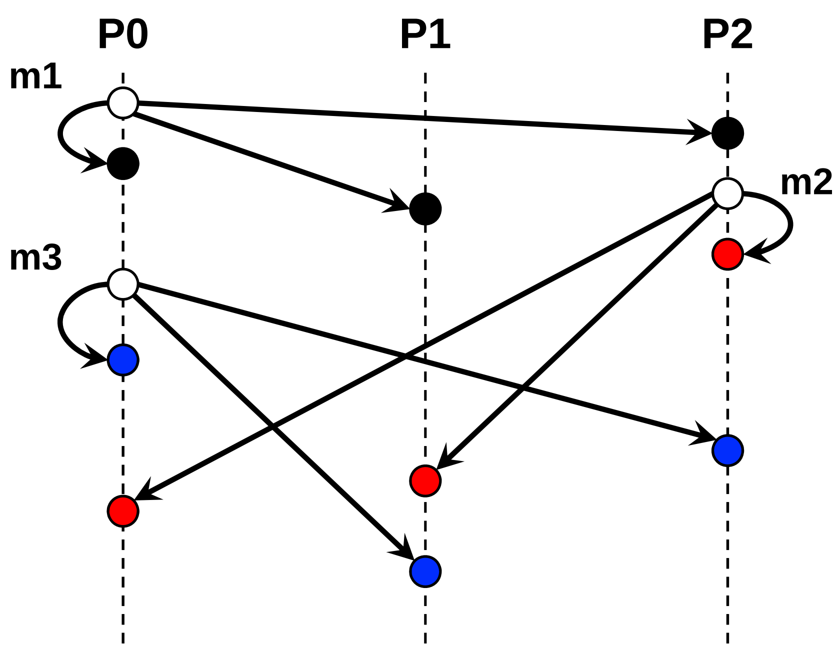
- **FIFO** ordering
  - If a process multicasts message m and subsequently multicasts message m', every process will deliver m before m'

Ordered multicast



- **Total** ordering
  - If a process delivers message m before it delivers m’, then any other process will also deliver m before m’

Ordered multicast



- **Causal** ordering

  If multicast(m) “happens-before” multicast(m’), all processes will deliver m before m’

The **happened before** relation ($\rightarrow$) causally relates two events.

m1 $\rightarrow$ m2	Process P2 multicast m2 after it received message m1.
m1 $\rightarrow$ m3	Process P0 multicast m3 after it multicast message m1.
m2 $\not\rightarrow$ m3	Process P0 multicast m3 **concurrently** with P2 multicasting m2.

FIFO multicast

- Analyse our algorithm for reliable multicast on top of IP-multicast.
- A process q delivers all messages from p in p sending order ($S^p{}_g$) by comparing to local expected sequence number $R^p{}_g$

(Unreliable) TO-multicast

- Basic approach as FIFO:
  - Uses globally unique IDs instead of per process unique IDs (as FIFO)
  - Receiver: deliver as for FIFO ordering

- Alg. 1: use a (single) sequencer process
- Alg. 2: participants collectively agree on the assignment of sequence numbers

TO-multicast: 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(<"order", i, S>) *with* $g = group(m)$

  wait until $<m, i>$ in hold-back queue and $S = r_g + 1$;

  TO-deliver m; // (after deleting it from the hold-back queue)

  $r_g = S$;

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>);

  $s_g := s_g + 1$;

r_g: seq nr of last delivered message

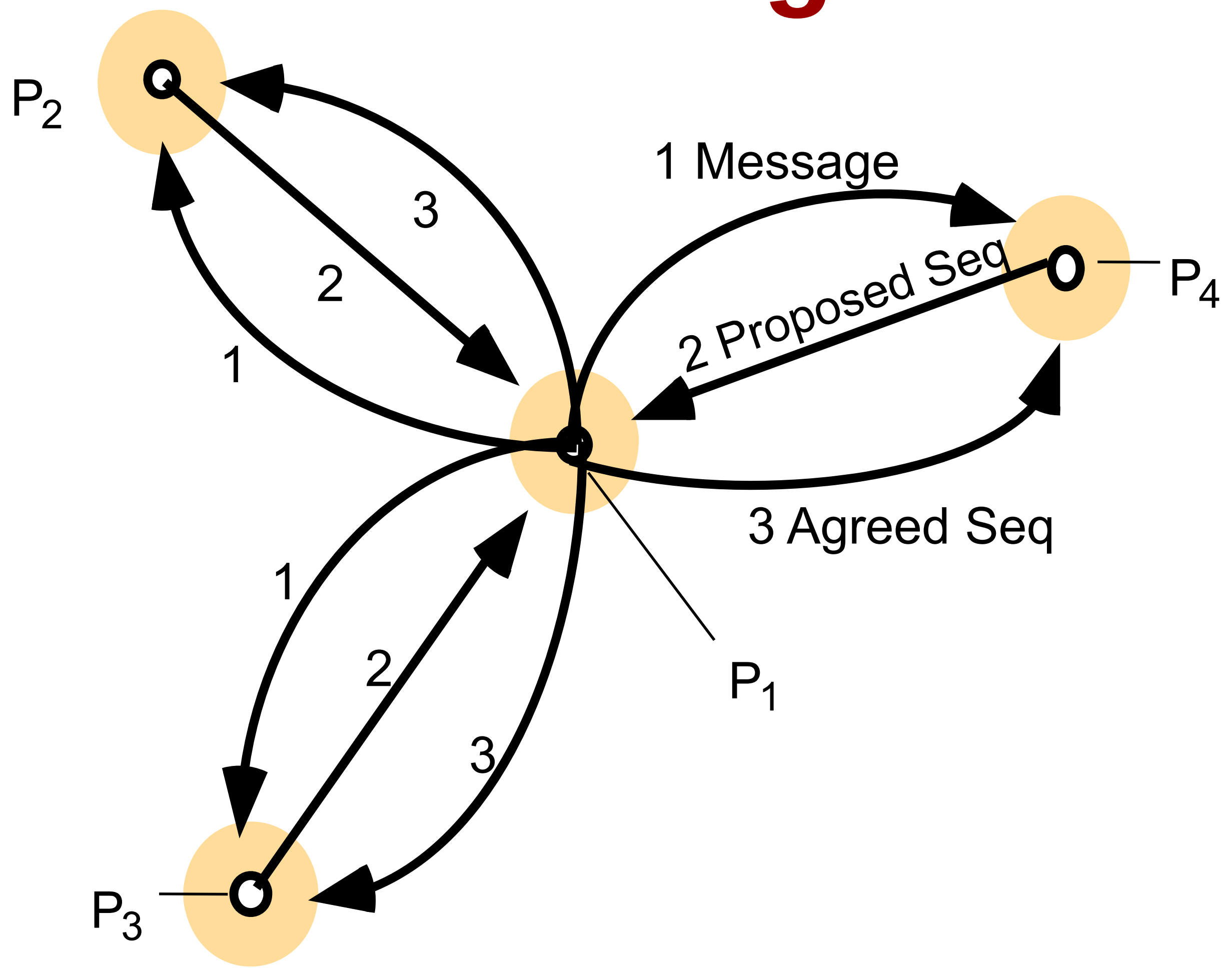
i: Unique message id

s_g: global unique seq nr

(Unreliable) TO-multicast: ISIS

- Approach:
  - Sender:
    - B-multicasts message
  - Receivers:
    - Propose sequence numbers to sender
  - Sender:
    - uses returned sequence numbers to generate agreed sequence number

The ISIS algorithm for total ordering



The ISIS algorithm

- Process q maintains sequence numbers
  - A^q_g the largest agreed seq nr q has observed for g
  - P^q_g q's own largest proposed sequence number q
- Process *p* performs *B-multicast(<m,i>,g)*, where *i* as a unique identifier for message *m*.
- Each process *q* replies *p* with a proposed sequence number $P^q_g := max(A^q_g, P^q_g) + 1$.
- Process *p* collects proposed sequence numbers and chooses the largest, let's call it *a*. Then *p* performs *B-multicast(<i,a>,g)*.
- Each process *q* in *g* sets $A^q_g := max(A^q_g, a)$ and attach sequence number *a* to message *m*

TO-multicast: ISIS alg.

- Correct?
  - Processes will agree on sequence number for a message
  - Sequence numbers are monotonically increasing
  - No process can prematurely deliver a message
- Performance
  - 3 serial messages!

CO-multicast

- Each process p_i maintains vector clock
  - $V_g^i[j]$ is the number of messages from each process P_j that happened-before next message to be multicast
- To ***CO-multicast(m):*** P_i increments $V_g^i[i]$ and *B-multicasts*(g,< V_g^i,m>)
- P_i ***CO-delivers(m)*** from P_j iff
  a) It has delivered any earlier message send by P_j
     $V_g^j[j] = V_g^i[j] + 1$, and
  b) It has delived any message that P_j had delivered at the time it multicast the message:
     $V_g^j[k] \le V_g^i[k] + 1, k \neq j$

E.g. message: V^2=[3,6,2] Receiver V^3=[2,5,2]

I.e p3 needs to deliver a message from p1 first

Summary

- So you thought multi-cast was simple??!!
- Applications have different semantic ordering, reliability and cost requirements
  - Unreliable / reliable multicast
  - FiFo, Causal, Causal-Fifo, Total, …
  - FiFo+Total (Exercise)
- Many algorithms available with different cost / ordering tradeoff
- Did you see an algorithm for totally ordered reliable multicasting ????

END