IN.5022 — Concurrent and Distributed Computing

Multicast Communication

Prof. P. Felber

pascal.felber@unine.ch

Agenda

- Multicast communication
  - Basic multicast
  - Reliable multicast
  - FIFO multicast
  - Causal multicast
  - Atomic multicast

What is multicast communication?

- Set of processes part of a group communicate *all together*
  - Each process receives the messages sent to the group
  - Often with *delivery guarantees*, e.g., agree on the set of messages received or on delivery ordering
- A process can multicast a message by the use of a single operation instead of one send per member
  - For efficiency and operation atomicity
- Groups can be closed (only members can send to group) or open, and static (membership is fixed) or dynamic
  - By default, we assume closed and static

Example: IP multicast

- An implementation of multicast communication *on top of IP*
  - Can transmit a single IP packet to a set of computers that form a multicast group (a *class D internet address* with first 4 bits 1110)
  - Dynamic membership, open groups
  - To multicast: send *UDP datagram* to a multicast address
  - To join: make a socket *join* a multicast group
- Failure model
  - Omission failures ⇒ some but not all members may receive a message
  - No order ⇒ IP packets may not arrive in sender order and group members can receive messages in different orders

System model

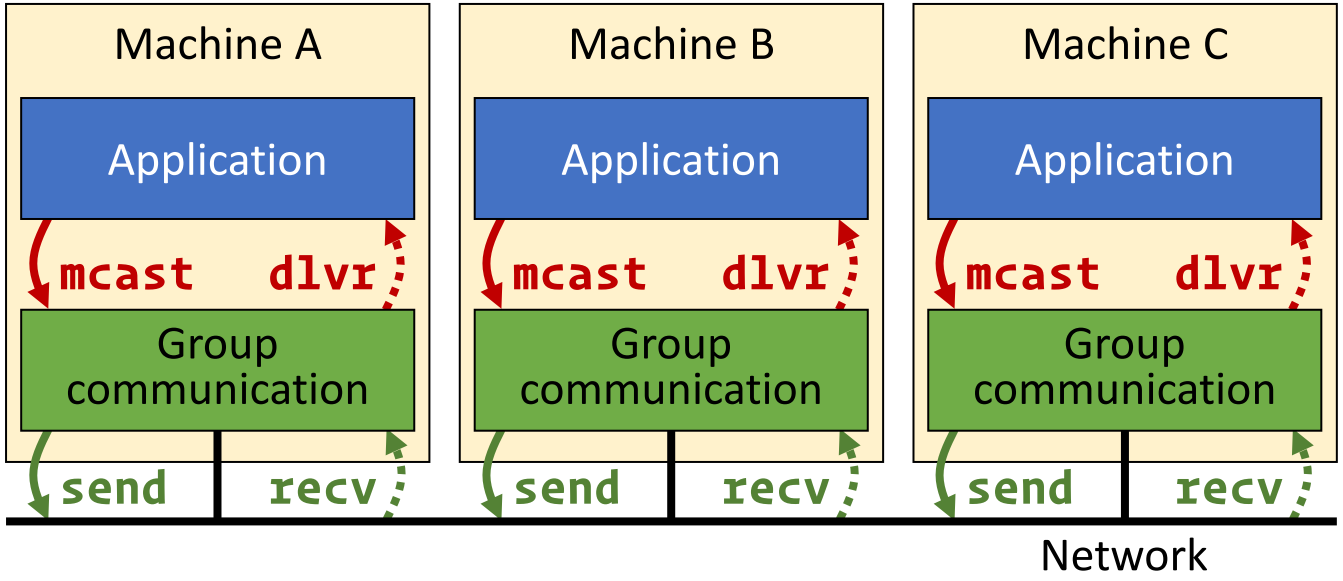
- The system consists of a collection of processes P
  - Processes communicate via reliable point-to-point channels
  - Processes fail only by crashing (no arbitrary failures)
  - No falsification of messages
- Processes are members of groups
  - In general, a process can belong to more than one group
- Two operations to communicate

  multicast(G, m) to send a message m to all members of G

  deliver(m) to get a message ready for delivery (≠ receive)
- Message m carries the identifier of the sending process and the target group: sender(m) and group(m)

Architecture

- The group communication layer implements primitives for multicast/deliver using underlying send/receive primitives
  - Multicast communication protocol with various guarantees
    - E.g., FIFO/causal/total/order, reliable delivery



A modular approach

- Multicast protocols form a hierarchy of specifications and corresponding algorithms
- Methodology: modular protocol design
  - Solve simpler problems first
  - Use weaker primitives as a black box to obtain stronger variations
  - Transformations: algorithms that transform weaker primitives into stronger primitives
  - Generic transformations: algorithms that do not depend on actual implementations

Reminder: reliable channels

- **Liveness:** if p sends m to q and q *does not fail*, then q *eventually receives* m from p
  - Implemented by use of sequence numbers, acknowledgements and retries
- **Safety:** process q receives message m from p *at most once* (no duplication) and only if p has *previously sent* m to q (no spurious message)
  - Implemented by use of checksums and discarding duplicates (retries), and possibly security techniques to deal with malicious users (e.g., signed messages)

1. Basic multicast

- A correct process will eventually deliver the message if the sender does not crash
  - IP multicast does not provide this guarantee!
  - Primitives called B-multicast and B-deliver
- Trivial protocol with reliable channels
- More efficient implementations on top of IP multicast
  - Sender piggy-backs monotonously increasing sequence numbers
  - Recipient detects and requests missed messages

```
program B-multicast

B-multicast(G, m)
  ∀p ∈ G :
    send(m) to p

% To B-deliver(m)
do
» receive(m) →
    B-deliver(m)
od
```

Multicast: motivating example

- Bulletin board application that multicasts messages
  - One multicast group per topic (e.g., *comp.os.linux*)
  - Requires reliable multicast so that all members receive messages

Bulletin board: comp.os.linux

Post #	From	Subject
23 **Total (atomic)**	A. Smith **FIFO**	Kernel 2.4.1 released!
24	J. Doe	Compile problem **Causal**
25	A. Smith	Re: Compile problem
26	R. Brown	Help needed
27	A. Smith	Kernel 2.4.2 released!

2. Reliable multicast

Agreement (liveness)

If a *correct process* delivers a message m, then eventually *all correct processes* in group(m) deliver m

Validity (liveness)

If a *correct process* multicasts a message m, then it will eventually deliver m

Integrity (safety)

For any message m, a correct process p ∈ group(m) delivers m *at most once* only if sender(m) *has previously multicast* m

The same (perhaps infinite) set of messages is delivered by all correct processes
That set includes all messages multicast by correct processes
No spurious messages are part of that set

Reliable multicast algorithm

- “Transformation” algorithm
  - Implements reliable multicast based on any implementation of basic multicast
- Correctness
  - Validity and integrity follow from B-multicast (and reliable channels)
  - Agreement follows from message re-multicasting: it is not possible for only some correct processes to deliver a message

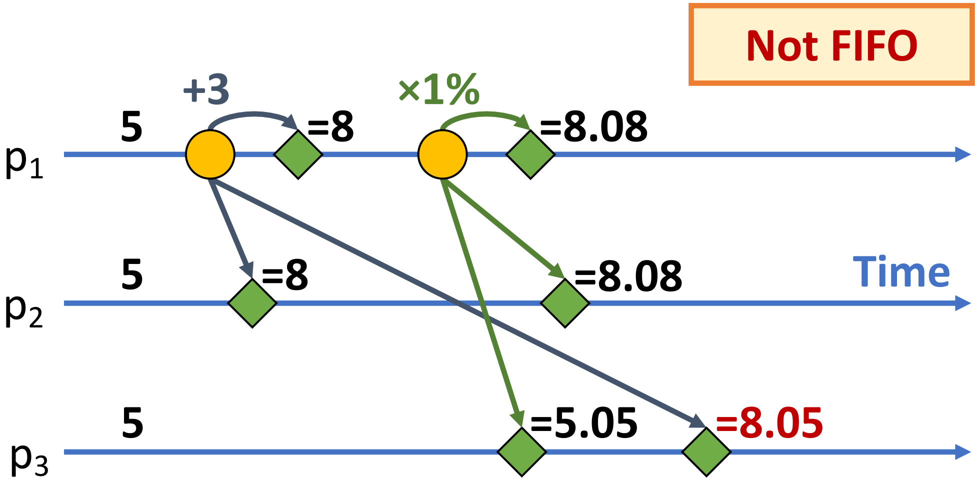
```
program R-multicast
define R: set of message identifiers
initially R = ∅

R-multicast(G, m)
  B-multicast(G, m)

% To R-deliver(m) at p ∈ group(m)
do
» B-deliver(m) →
    if id(m) ∉ R →
      R := R ∪ {id(m)}
      if p ≠ sender(m) →
        B-multicast(group(m), m)
      fi
      R-deliver(m)
    fi
od
```

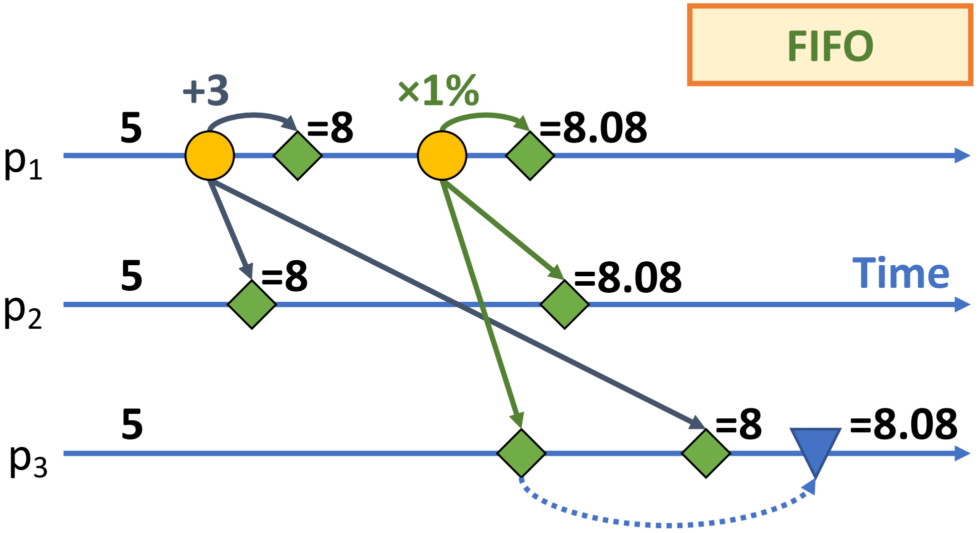
3. FIFO multicast (ordered)

Reliable multicast +

FIFO order (safety)

> If a process multicasts a message m *before* multicasting a message m', then *no correct process* delivers m' *unless it has previously delivered* m (sender order is preserved)



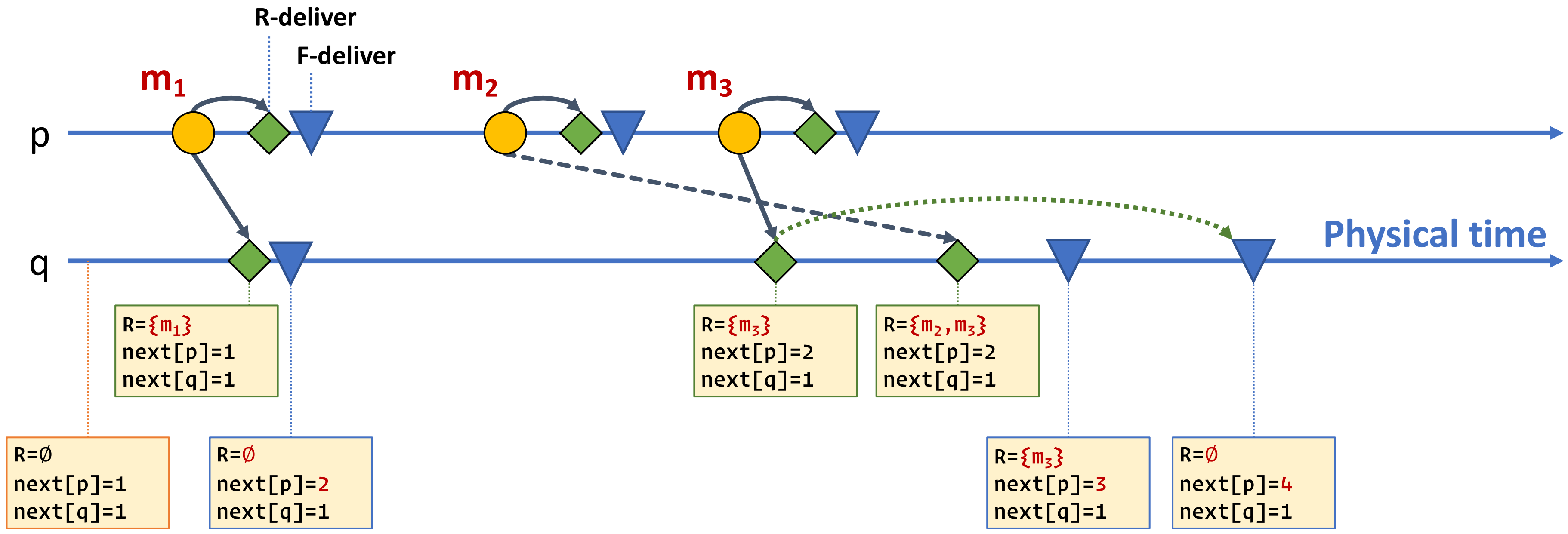


FIFO multicast algorithm

- Use sequence numbers
  - Similar to reliable FIFO channels, with one counter maintained for each group member
  - Keep a buffer of messages that have been R-delivered but not yet F-delivered
  - Deliver messages in sending order

```
program F-multicast
define
  c: integer        % sequence number (message counter)
  next[]: integer array  % next messages to F-deliver
  R: set of messages   % messages not yet F-delivered
initially c = 1, R = ∅, next = [1,…,1]

F-multicast(G, m)
  seq#(m) := c         % tag message with next number
  R-multicast(G, m)         % p also receives a copy!
  c := c + 1

% To F-deliver(m) at p ∈ group(m)
do
» R-deliver(m) from s →
    R := R ∪ {m}
    do
    » ∃r ∈ R : sender(r) = s AND seq#(r) = next[s] →
        F-deliver(r)
        next[s] := next[s] + 1
        R := R \ {r}
    od
od
```

FIFO multicast: sample run



R={m1}
next[p]=1
next[q]=1

R={m3}
next[p]=2
next[q]=1

R={m2,m3}
next[p]=2
next[q]=1

R=∅
next[p]=1
next[q]=1

R=∅
next[p]=2
next[q]=1

R={m3}
next[p]=3
next[q]=1

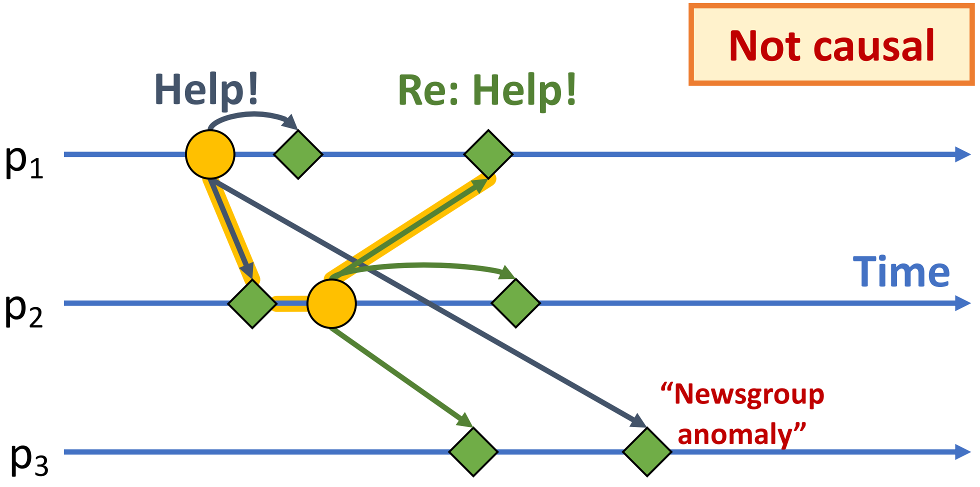
R=∅
next[p]=4
next[q]=1

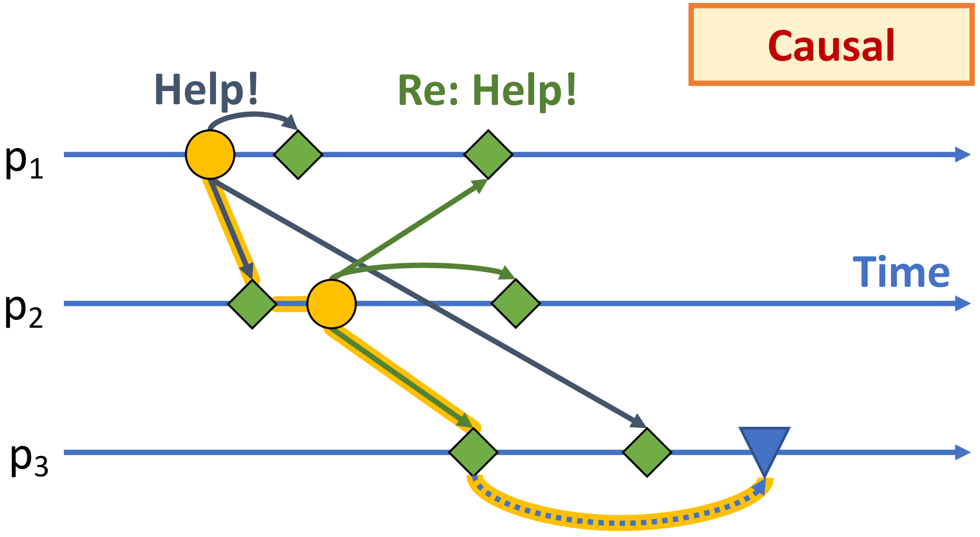
3. Causal multicast (ordered)

Reliable multicast +

Causal order (safety)

> If the multicast of a message m *causally precedes* the multicast of a message m', then *no correct process* delivers m' *unless it has previously delivered* m





Causal multicast on top of FIFO multicast?

- We have

  Causal order ⇒ FIFO order

- But…

  FIFO order ⇏ causal order

- How can we implement causal on top of FIFO?

  Causal order = FIFO order + ?

Causal order = FIFO order + local order (safety)

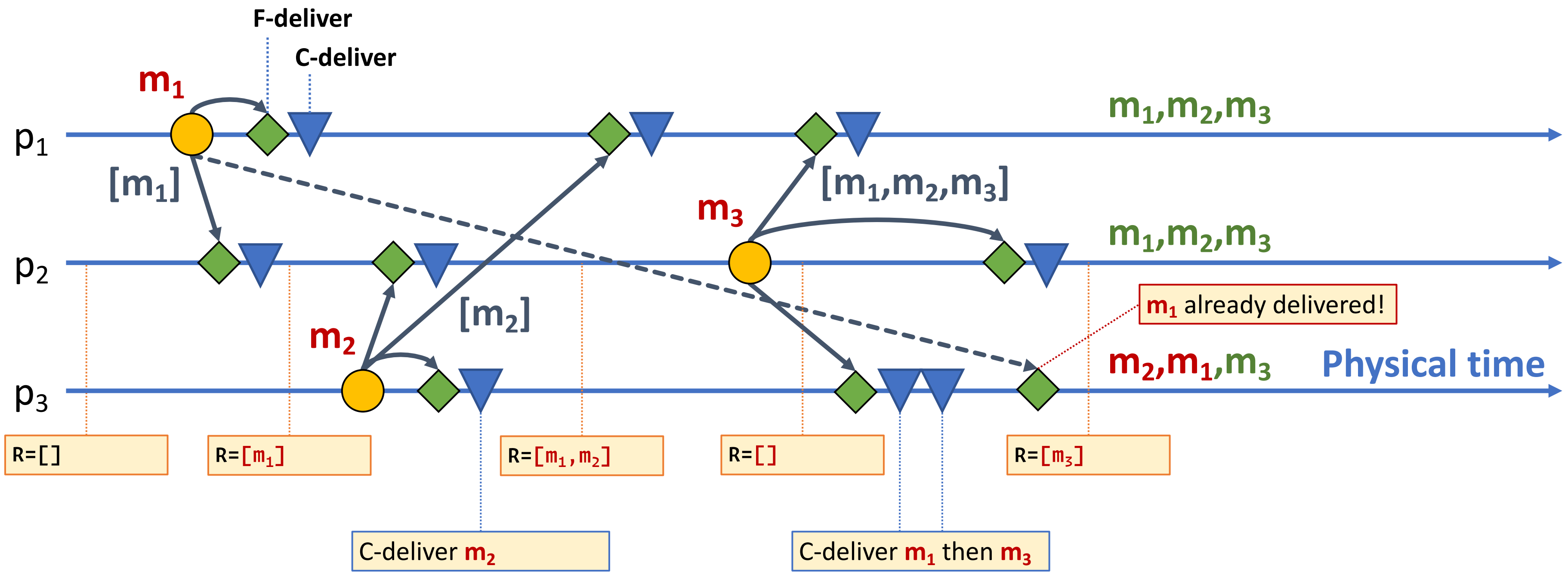
If a process delivers a message m *before* multicasting a message m’, then *no correct process* delivers m’ *unless it has previously delivered* m

Causal multicast algorithm

- Use message lists
  - Keep a list of messages that have been F-delivered but not yet C-delivered
  - Multicast ordered list together with new message
  - Deliver messages in order and empty list
  - Remember previously C-delivered messages (no duplicates)

```
program C-multicast
define
  C: set of message identifiers % already C-delivered
  R: list of messages    % newly C-delivered messages
initially C = ∅, R = []

C-multicast(G, m)
  F-multicast(G, R++[m]) % append m to list, send it…
  R := []                            % …and empty list

% To C-deliver(m) at p ∈ group(m)
do
» F-deliver(M) →
    ∀m ∈ M :   % traverse list in order (important!)
      if id(m) ∉ C →
        C := C ∪ {id(m)}
        C-deliver(m)
        R := R++[m]
      fi
od
```

Causal multicast: sample run



Non-blocking transformation: C-delivery is never postponed, but the size of messages can grow large ⇒ not very practical (would be blocking if forwarding only message identifiers)

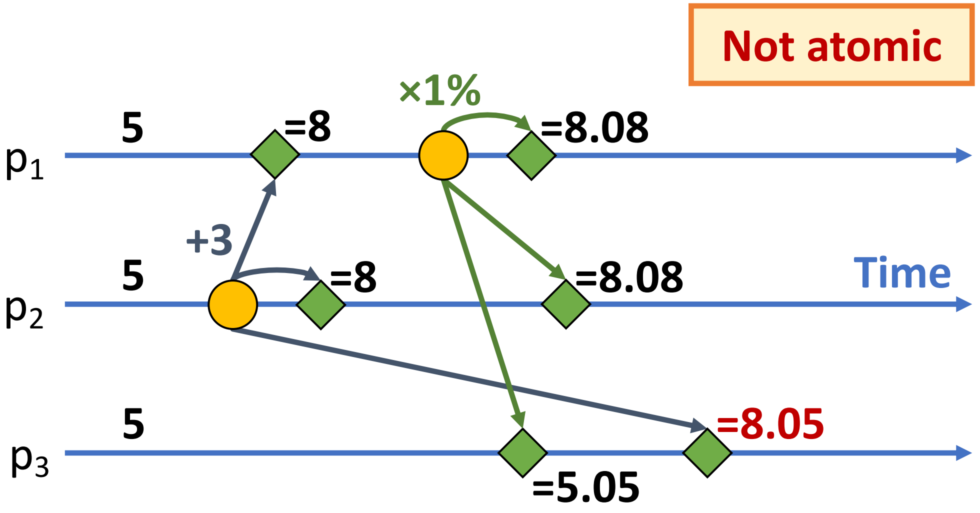
4. Atomic multicast (ordered)

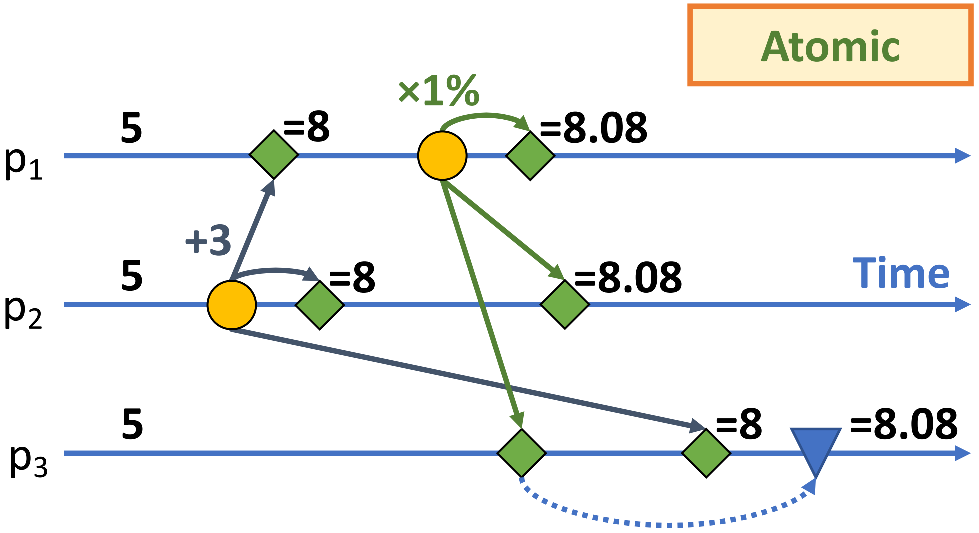
Is atomic multicast also FIFO?
Is atomic multicast also causal?

Reliable multicast +

Total order (safety)

If *correct processes* p and q *both* deliver messages m and m’, then p delivers m before m’ *if and only if* q delivers m before m’

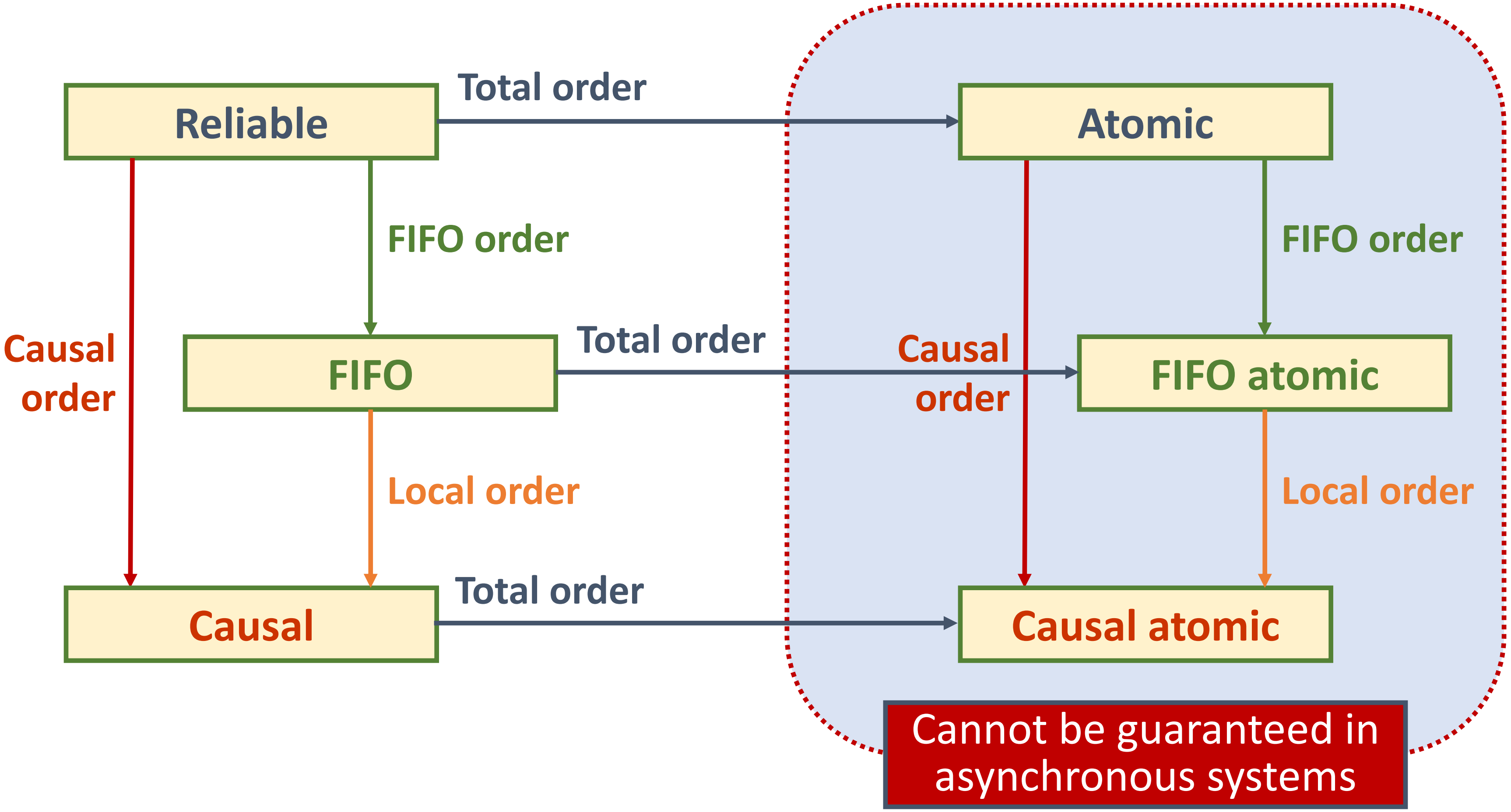

Not atomic


Atomic

Implementing atomic multicast

- Atomic multicast cannot be implemented in asynchronous systems
  - Even for *one* process failure!
- Why?
  - Consensus cannot be solved (i.e., “guaranteed”) in an asynchronous system [FLP85]
  - Atomic multicast is *equivalent* to consensus, i.e., there are transformations between consensus and atomic multicast
    1. Use consensus to *agree* on message ordering
    2. Use atomic multicast to *impose* proposal
  - If atomic multicast could be implemented, so would be consensus ⇒ contradiction

Relationships among multicasts



Cannot be guaranteed in asynchronous systems

Atomic multicast with sequencer (no failure)

- Two distinct roles
  - Group members buffer messages and wait for sequence numbers
  - Distinguished process (sequencer) orders messages

What about failures?

Group member?

B-multicast may fail…

Sequencer?

No more leader…

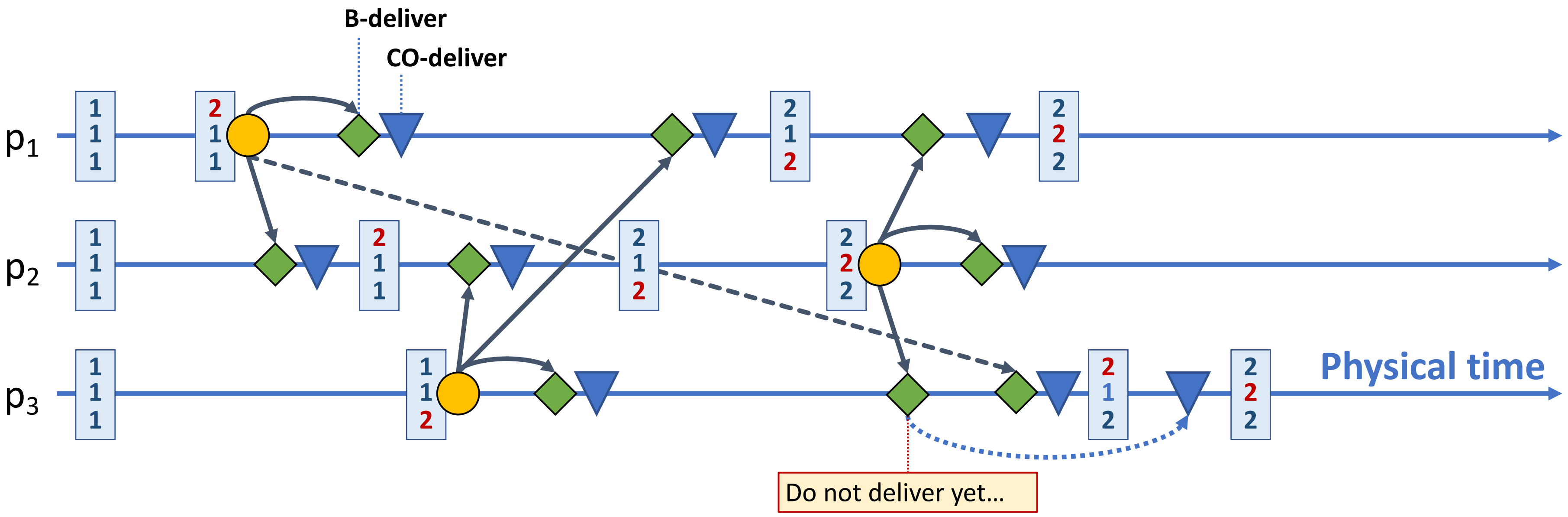
```
program A-multicast      % underlined ≡ sequencer only
define
  r, s: integer                       % sequence numbers
  R: set of messages   % messages not yet A-delivered
initially r = 1, s = 1, R = ∅

A-multicast(G, m)
  B-multicast(G, m)    % basic multicast (no failure)

% To A-deliver(m) at p ∈ group(m)
do
» B-deliver(m) →
    R := R ∪ {m}
    B-multicast(group(m), <ORDER, id(m), s>)  % only…
    s := s + 1                      % …done by the sequencer
» B-deliver(<ORDER, id(m), n>) AND m ∈ R AND r = n →
    A-deliver(m)
    R := R \ m
    r := r + 1
od
```

CO-multicast (causal order with vector clocks)

- Isis causal multicast algorithm
  - Processes maintain vector clocks and attach them to multicast messages
  - Before delivery, wait until all previous messages sent and delivered by sender have been delivered

```
program CO-multicast                                      % process $p_i$
define
  V[]: integer array                                 % vector timestamp
initially V = [1,…,1]

CO-multicast(G, m)
  V[i] := V[i] + 1                              % increment local clock
  B-multicast(G, <V, m>)                          % send with timestamp

% To CO-deliver(m) at $p_i \in group(m)$
do
» B-deliver(<W, m>) from $p_i$ →
    CO-deliver(m)
» B-deliver(<W, m>) from $p_j \neq p_i$   % after delivery of…
  AND W[j] = V[j] + 1                 % …all messages sent by $p_j$…
  AND W[k ≠ j] ≤ V[k] →              % …and those delivered by $p_j$
    CO-deliver(m)
    V[j] := V[j] + 1
od
```

CO-multicast: sample run



Protocol is deadlock-free because the ordering relation imposed by vector clocks is acyclic (and it becomes reliable when using R-multicast instead of B-multicast)

Summary

- Multicast is a powerful communication pattern for groups of processes
  - Processes can multicast messages by using a single operation instead of one send per member
  - Messages are delivered with reliability and ordering guarantees
- Algorithms can be implemented as transformations
  - Multicast protocols form a hierarchy of specifications and corresponding algorithms
  - Use weaker primitive to implement algorithms with stronger guarantees
  - Basic → reliable → FIFO → causal [→ atomic]