Cours 6

La communication de groupe dans les systèmes réparties

Main references:

Ref 1: Communication de groupe, Abdelouahid Derhab Maitre de recherche A (CERIST) Email: aderhab@cerist.dz

Ref 2: Défago, Xavier & Schiper, André & Urbán, Péter. (2000). Totally Ordered Broadcast and Multicast Algorithms: A Comprehensive Survey.

Ref 3 : Distributed systems Lecture 12: Logical time, vector clocks, process groups, and ordered broadcast, Michaelmas 2019 Dr Martin Kleppmann;

Ref 4:, CS542 Topics in Distributed Systems. : Communication Modes in Distributed System. Diganta Goswami.

Communication de groupe (Adapté)

Abdelouahid Derhab

Maitre de recherche A (CERIST)

Email: aderhab@cerist.dz

Communication de Groupe (1)

- **But :** chaque membre d'un groupe doit recevoir une copie de tous les messages adressés au groupe

- **Communication de groupe nécessite :**

  - Coordination

  - Accord : sur l'ensemble des messages reçus et leur ordre

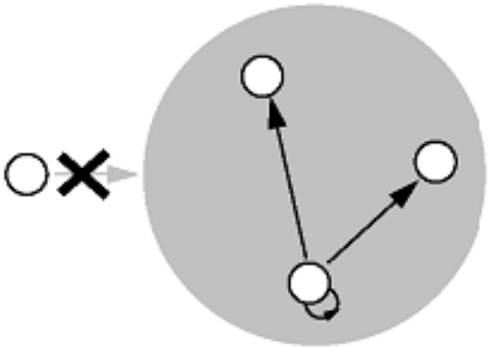
- On se restreint aux groupes de processus dont leur appartenance est connue (groupes statiques)

Communication de Groupe (2)

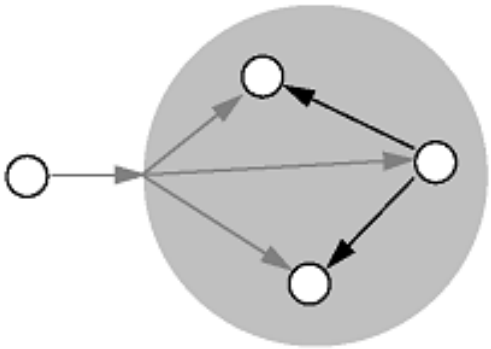
- **Système :** collection de processus pouvant se communiquer d'une façon **fiable** sur des liens de **communication point-à-point**

- **Processus :** membres de groupes, peuvent subir une panne totale

- **Groupes :**



Groupe fermé



Groupe ouvert

Communication de Groupe (3)

- **Primitives :**
  - ***multicast(g, m)*** **:** envoie le message *m* à tous les processus du groupe g
  - ***deliver(m)*** **:** livre le message *m* au processus

    (le message n'est pas consommé directement par le processus)
  - ***sender(m)*** **:** identificateur unique qui désigne l'émetteur du message *m*
  - ***group(m)*** **:** identificateur unique du groupe auquel est destiné le message *m*

Communication de Groupe (4)

- Communication de groupe de base
- Communication de groupe fiable
- Communication de groupe ordonnée

Communication de Groupe de Base

- **But :** Garantir qu'un processus non-défaillant livre le message reçu tant que l'émetteur n'est pas en panne.
- **Primitives :** B_multicast, B_deliver

B_multicast(g, m)

Pour chaque processus p ∈ g, send(p, m);

receive(m) par p

B_deliver(m) à p (Consommer le message m)

Communication de Groupe Fiable (1)

- **But :**La communication de groupe fiable garantit qu'un message multicasté par un processus correct (non-faulty) sera finalement délivré à tous les processus corrects du groupe.
  - **Intégrité :** un processus correct P délivre le message m au plus une fois
  - **Accord :** Si un processus correct livre le message m, alors tous les autres processus corrects dans group(m) livreront éventuellement m

Communication de Groupe Fiable Algorithme (2)

```
R_multicast(m):
    pour tout processus q ∈ group:
        send(p, m) → q
```

```
à la réception de (p, m) depuis q:
    si (p, m) ∉ delivered:
        deliver(p, m)     // première fois que r voit ce message
        pour tout processus s ∈ group, s ≠ r:
            send(r, (p, m)) → s   // RETRANSMISSION FIABLE
```

```
deliver(p, m):
    delivered = delivered ∪ {(p, m)}
    afficher(m) ou donner m à l'application
```

Homework

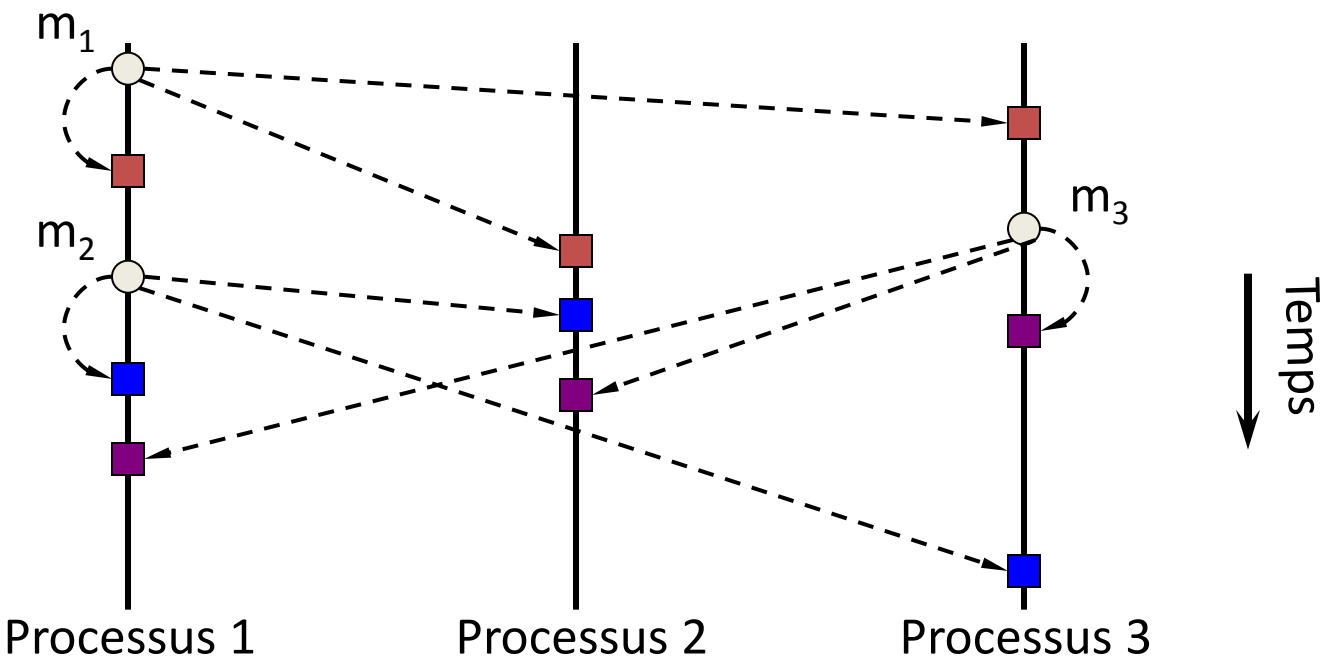
- Reliable group communication in distributed systems. Another optimized algorithms.

Communication de Groupe Ordonnée (1)

- **Catégories d'ordonnancement :**
  - Ordonnancement FIFO
  - Ordonnancement Total
  - Ordonnancement Causal
  - Ordonnancement hybride : Total-Causal, Total-FIFO

Ordonnancement FIFO (1)

- Si un processus correct exécute multicast(g, m_1), puis multicast(g, m_2), alors tout processus correct membre de g qui livre m_2 livrera m_1 avant m_2



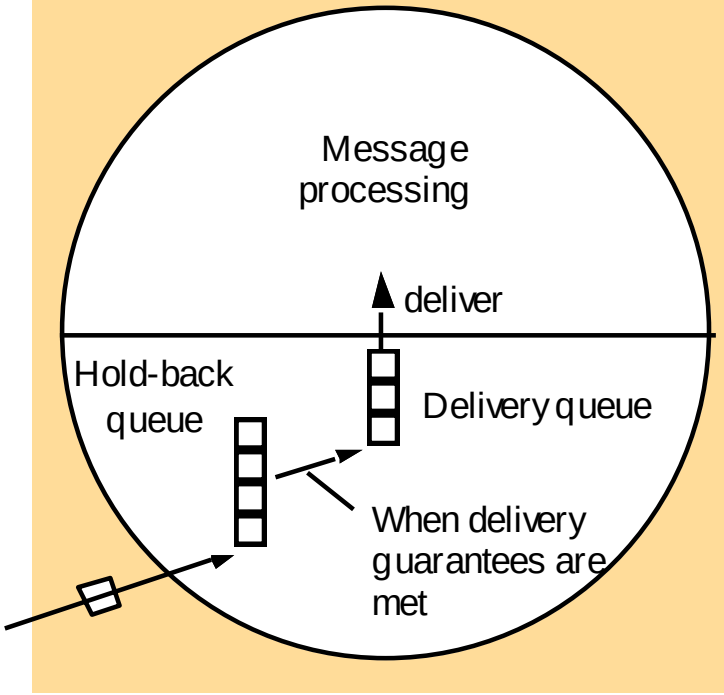
Ordonnancement FIFO (2)

- **Primitives :** FO_multicast, FO_deliver
- **Implémentation :** Utilisation de numéros de séquence
- Variables maintenues par chaque processus p :
  - S_g^p **:** Nombre de messages envoyés par p au groupe g
  - R_g^q **:** Numéro de séquence du dernier message envoyé au groupe g par q et que p a livré
- Algorithme
- Ordonnancement FIFO garantie seulement si les groupes ne se chevauchent pas

Ordonnancement FIFO (2)

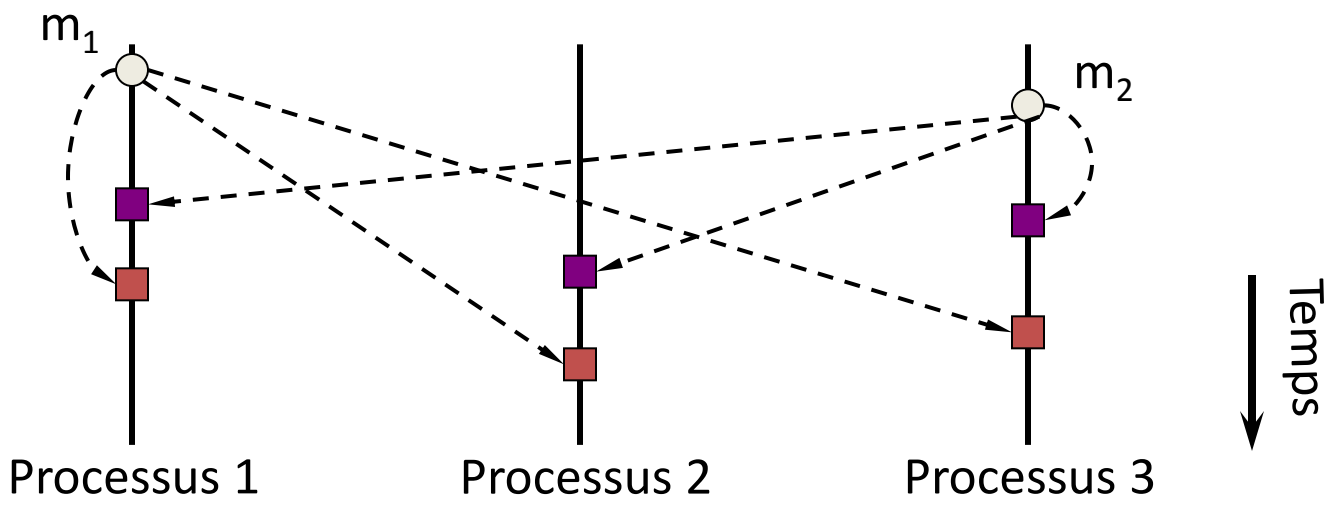
- **FO-multicast(m, g)**
  B-multicast(m, g, < S_g^p >
  increment S_g^p by 1

- **upon receipt of a message from q with sequence number S**
  if S = R_g^q +1, then this is the next message,
  therefore FO-deliver(m)
  R_g^q := S
  if S > R_g^q +1, then
  place message on hold-back queue until intervening messages have been delivered and S = R_g^q +1

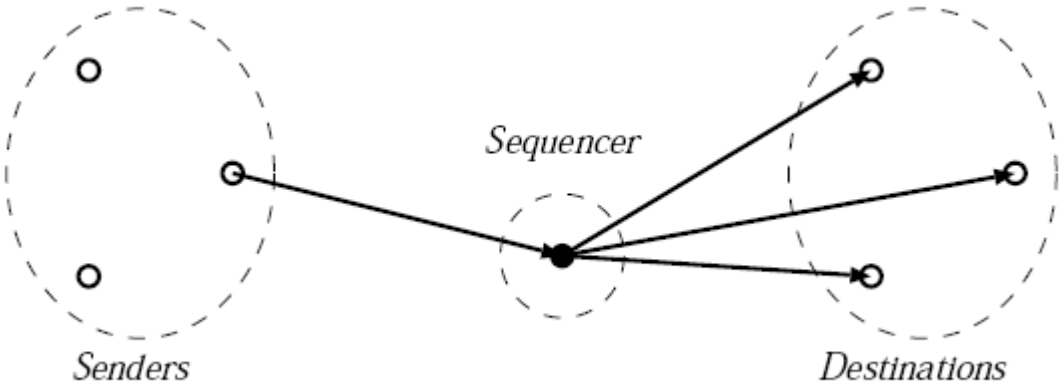


Ordonnancement Total (1)

- Si un processus correct livre un message m_2 avant de livrer un message m_1, alors tout autre processus correct qui livre m_1 livrera m_2 avant m_1



Fixed sequencer algorithms



Simple fixed sequencer algorithm

(code of process p)

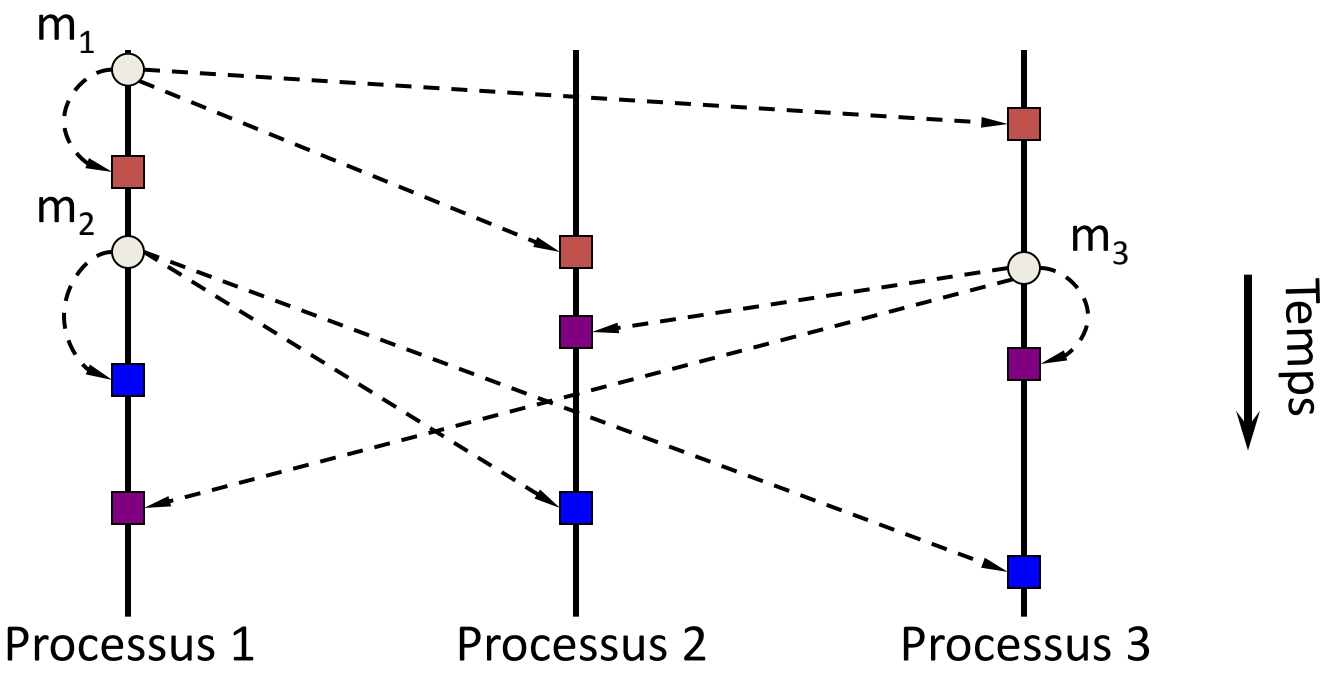
```
1:  Sender:
2:     procedure TO-multicast(m)
3:        send (m) to sequencer
4:     end

5:  Sequencer:
6:     Initialisation:
7:        seqnum ← 1
8:     when receive (m)
9:        sn(m) ← seqnum
10:       send (m, sn(m)) to all
11:       seqnum ← seqnum + 1
12:    end when

13: Destinations (code of process p_i):
14:    Initialisation:
15:       nextdeliver_{p_i} ← 1
16:       pending_{p_i} ← ∅
17:    when receive (m, seqnum)
18:       pending_{p_i} ← pending_{p_i} ∪ {(m, seqnum)}
19:       while ∃(m', seqnum') ∈ pending_{p_i} : seqnum' = nextdeliver_{p_i} do
20:          deliver (m')
21:          nextdeliver_{p_i} ← nextdeliver_{p_i} + 1
22:       end while
23:    end when
```

Ordonnancement Causal (1)

- Si multicast(g, m_1) $\rightarrow$ multicast(g, m_3), alors tout processus correct qui livre m_3 livrera m_1 avant m_3

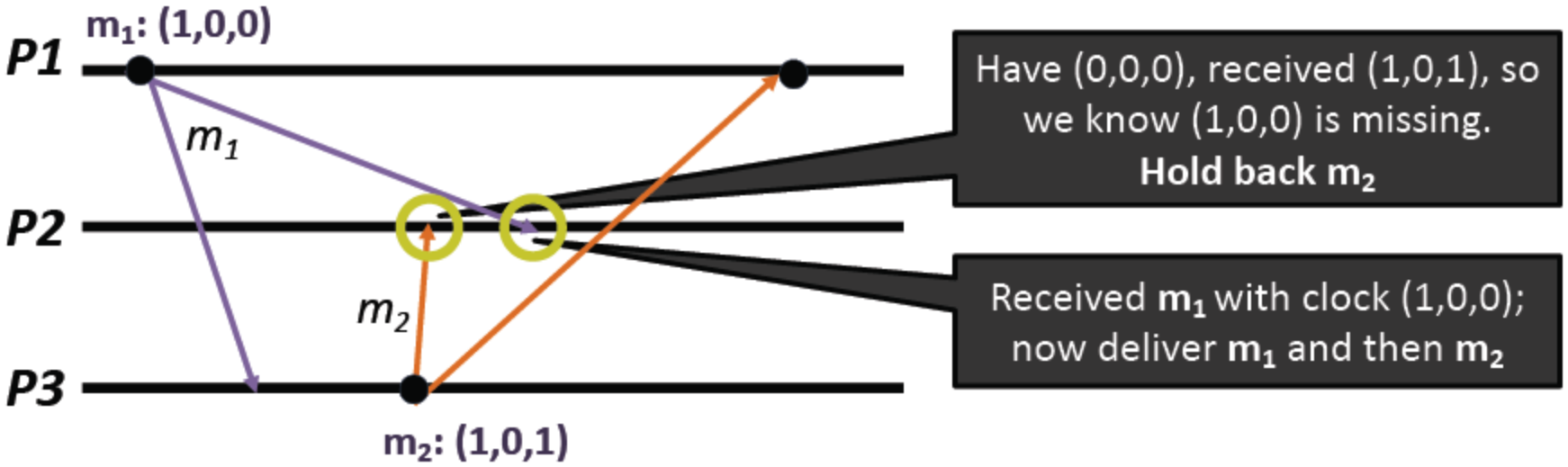


Causal order message delivery

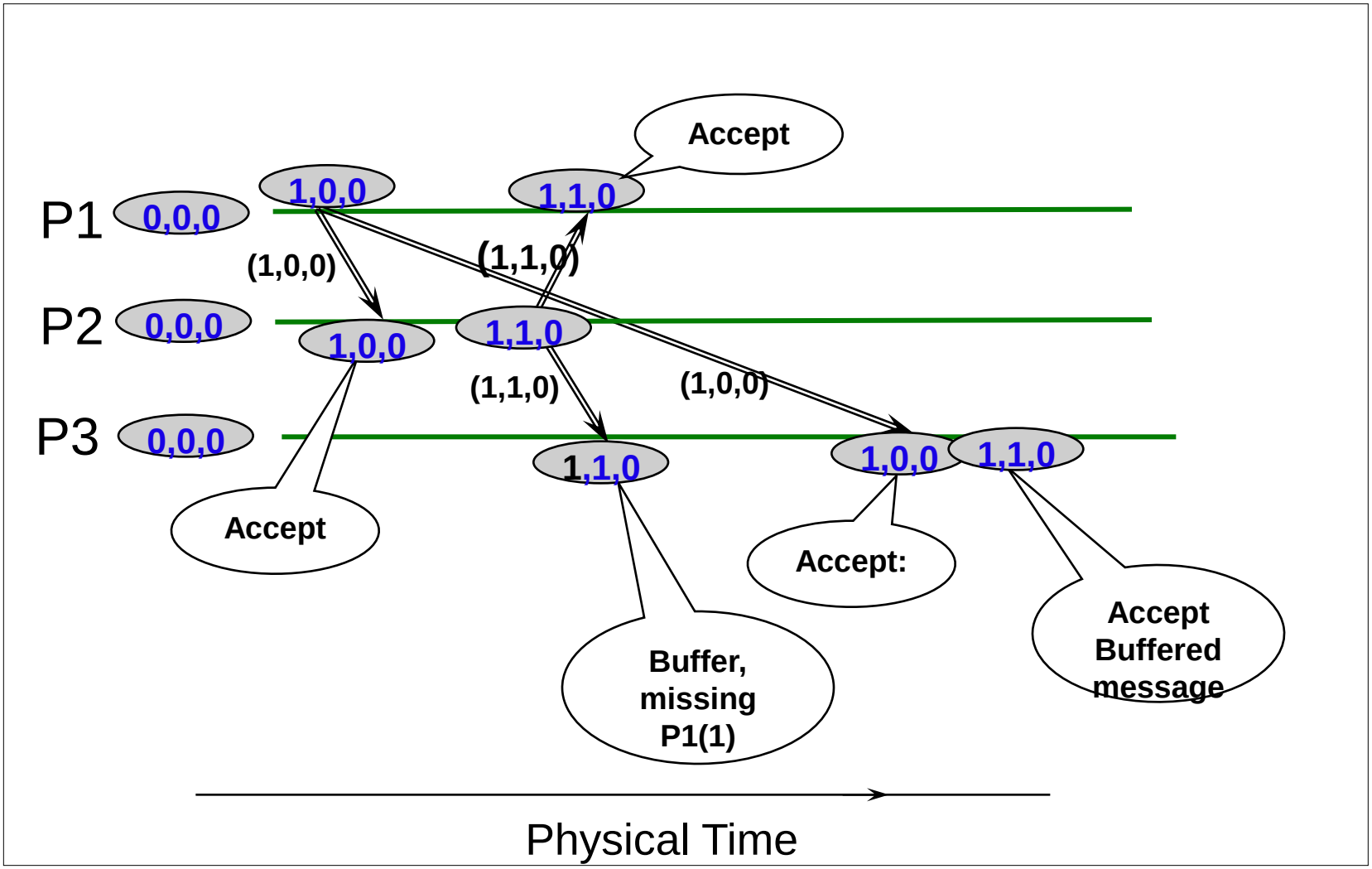
- When message ***m*** is received, need to decide:
  - Does a message ***m'*** exist that we have not yet received, such that ***m'*** $\rightarrow$ ***m***?
  - If yes, wait for ***m'*** to be received and deliver it first
  - If no, deliver ***m*** to the application now
- Solution: variant of vector clocks
  - Increment only on ***message send***, not on every event
  - Detects relative ordering of ***messages***, not events
  - Gap in number sequence $\Rightarrow$ wait for message

Implementing causal ordering

- Like FIFO multicast, but with vector clocks instead of sequence numbers



Example: Causal Ordering Multicast



Causal Ordering using vector timestamps

Homework : Write the algorithm

Abstract

- Total order multicast algorithms constitute an important class of problems in distributed systems, especially in the context of fault-tolerance. In short, the problem of total order multicast consists in sending messages to a set of processes, in such a way that all messages are delivered by all correct destinations in the same order. However, the huge amount of literature on the subject and the plethora of solutions proposed so far make it difficult for practitioners to select a solution adapted to their specific problem. As a result, naïve solutions are often used while better solutions are ignored (Réf. 2)