Time, Clocks and the Ordering of Events in a Distributed System

Lamport (1978)

http://h-deb.clg.qc.ca/Sujets/Orthogonal/Individus-importants.html#leslie_lamport

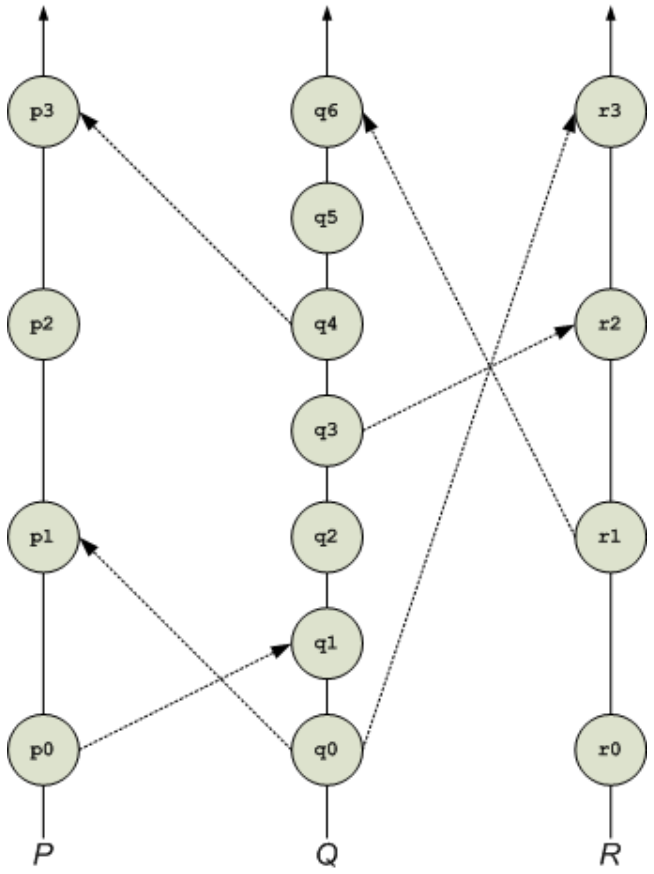
Happens-Before

- Que signifie «A s'est produit avant B»?
  - A s'est produit à 8:29 alors que B s'est produit à 8:30?
  - Suppose une horloge commune
  - Suppose une consultation instantanée
    - Tolérable à l'échelle humaine, soit

Dans un système réparti

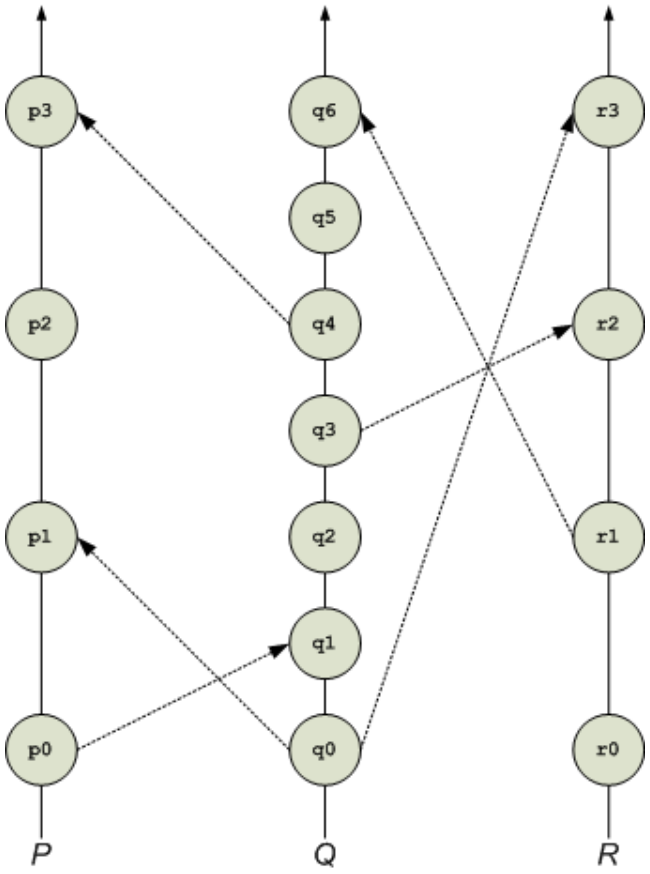
- Suppose des composants sur des supports physiques distincts
- Communication par messages
  - Non-instantanée
  - Déterminée par «*qui* envoie *quand*» et par «*qui* reçoit *quand*»
  - De prime abord, le *quand* est un *quand-selon* (quand *selon* l'émetteur, quand *selon* le récepteur)

Communication par messages



- Les bulles sont des événements
  - Pour chaque processus, les événements perçus localement constituent une séquence chronologique
  - Pour R, R_i précède R_j si $i<j$
  - Envoyer un message est un événement
  - Recevoir un message est un événement
- Les traits pointillés sont des messages
  - P.ex.: un message quitte Q à q4 et atteint P à p3
  - Écrivons *M(Q:q4;P:p3)*

Communication par messages (suite)



- Pour chaque processus, il existe un ordre partiel des événements
  - Pour *Q*, q0 précède q1, qui précède q2, *etc*.
- Il est possible que des messages se croisent
  - P.ex.: *M(Q:q0;R:r3)* mais *M(Q:q3;R:r2)*
  - L'ordre des envois peut ne pas correspondre à l'ordre de réception

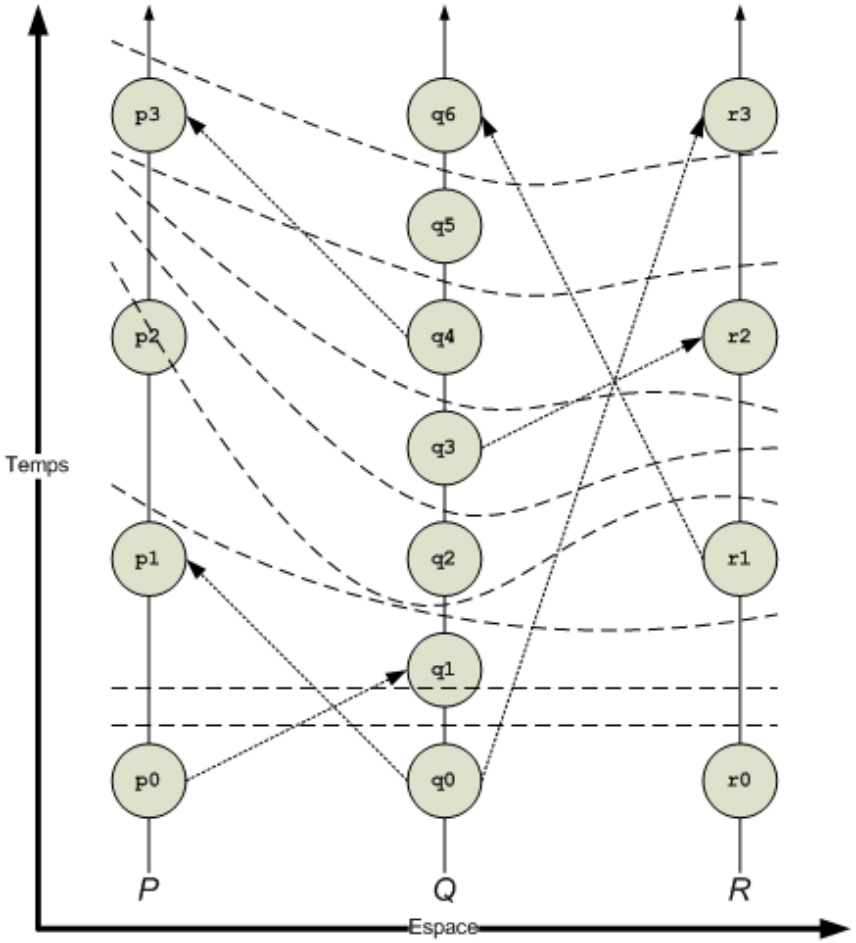
Peut-on déterminer un ordre global?

- Si oui, à quelles conditions?

Définitions

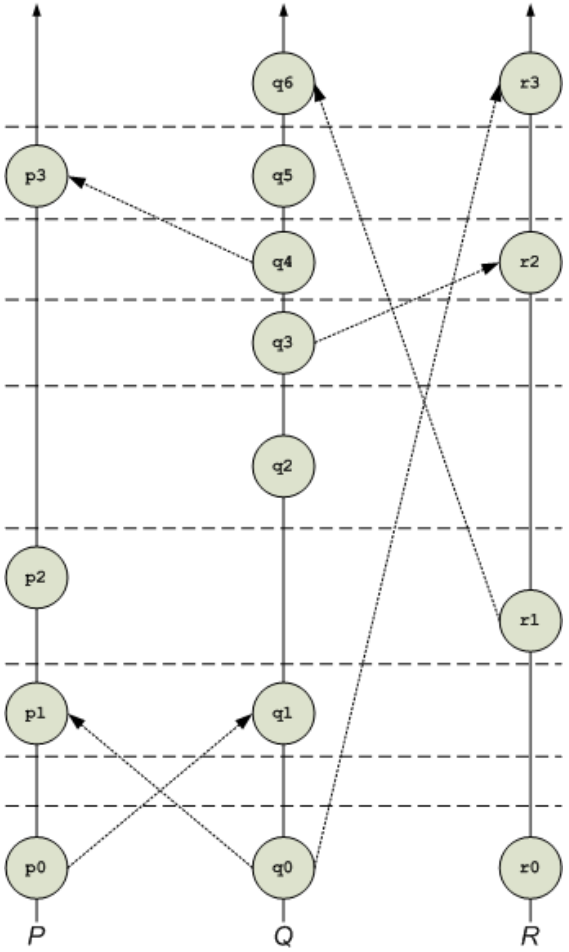
- Processus: séquence d'événements
  - Ordonnancement local total *a priori*
- A *happens-before* B: A→B
- «*The relation "→" on the set of events of a system is the smallest relation satisfying the following three conditions:*
  1. *If a and b are events in the same process, and a comes before b, then a → b.*
  2. *If a is the sending of a message by one process and b is the receipt of the same message by another process, then a → b.*
  3. *a → b && b → c ⇨a → c.*
- *Two distinct events a and b are said to be concurrent if !(a→b) && !(b→a)*»
  - On suppose !(a→a) pour tout a.
- La relation → définit un ordonnancement partiel *non-réflexif* sur l'ensemble des événements du système

Diagramme d'espace-temps



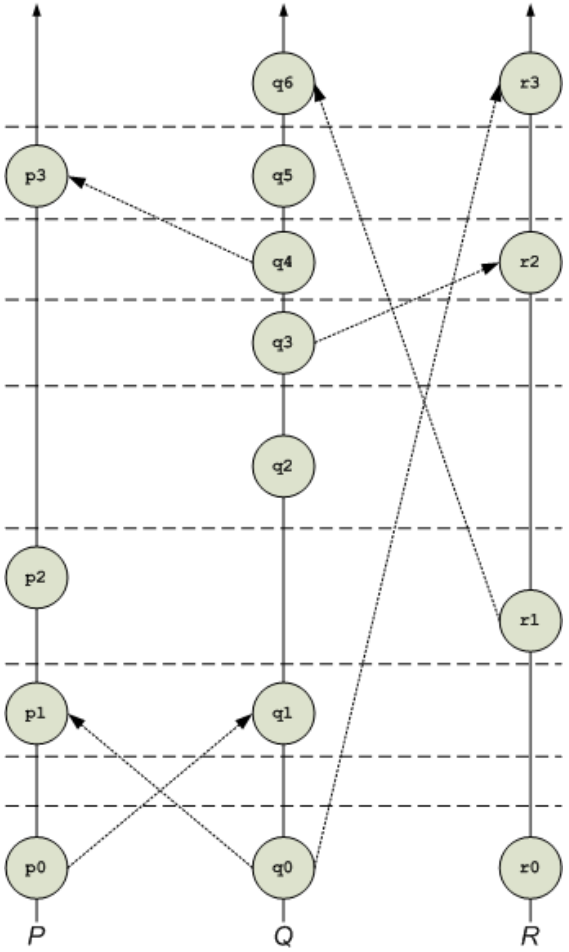
- Les lignes coupées montrent un ordonnancement global possible
  - Visiblement, ici, p0→r3
- *Happens-before* signifie aussi *Peut-influencer*
  - p0 peut influencer r3
  - p2 et q2 sont concurrents; l'un ne peut influencer l'autre
- Le temps logique n'est pas nécessairement connexe au temps physique
  - Que q2 précède physiquement p2 ne signifie pas que q2 précède logiquement p2
  - *Au mieux*, avant p3, P peut savoir ce que planifiait Q en vue de q2

Fonction *C*



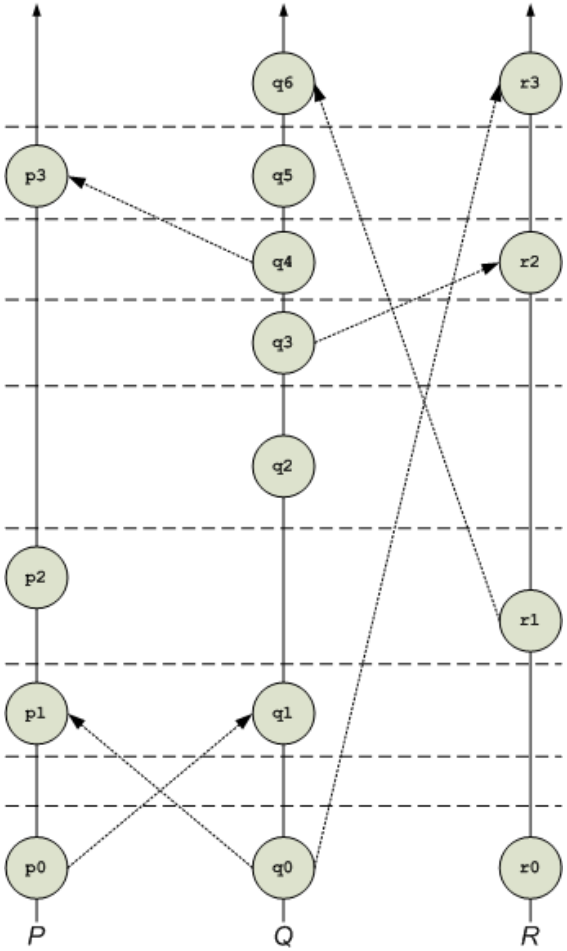
- *C*, pour *Clock*
  - $C(e)=C_j(e)$ si *e* est un événement de P_j
  - Pas de relation directe entre *C* et le temps physique
- Clock Condition:
  - Si a→b alors $C(a)<C(b)$
  - L'inverse n'est pas vrai (sinon, a et b concurrent signifierait a et b simultané, ce qui n'est ni vrai, ni nécessaire)

Fonction *C* (*suite*)



- Deux conditions
  1. Dans le processus P_i, a→b implique $C_i(a)<C_i(b)$
  2. $M(P_i:a;P_j:b)$ implique $C_i(a)<C_j(b)$
- Règle d'implémentation IR1:
  - Tout processus P_i incrémente C_i entre deux événements successifs pour lui
  - Pour y arriver, tout message *m* doit être estampillé du temps d'émission T_m

Fonction *C* (*suite*)



- Règle d'implémentation IR2:
  - Sur réception de *m* par P_j, C_j doit être avancé de manière à dépasser T_m
  - Si a est l'envoi de *m* par P_i, alors $T_m=C_i(a)$. Sur réception de T_m par P_j, P_j fait en sorte que $C_j>T_m$
  - Si $C_j>T_m$, alors on peut laisser C_j tel quel

Déterminer un ordre total

- Supposons une numérotation unique pour chaque processus P_i
- Si l'événement a se produit dans P_i et si l'événement b se produit dans P_j, alors
  - a➔b si // a→b est local, a➔b est total
    - $C_i(a)<C_j(b)$ ou
    - $C_i(a)=C_j(b)$ && $P_i<P_j$
- La relation ➔ n'est pas unique (entre autres, l'ordonnancement des P_i est arbitraire)
  - C'est un choix, pas un fait ou une vérité

Utilité d'un ordre total

- Supposons un système de processus $P_0..P_n$ partageant une ressource R
- À tout moment, un seul P_i, i:0..n peut utiliser R
- Supposons un algorithme de synchronisation devant satisfaire ces trois conditions:
  1. Si P_i possède R, P_i doit libérer R avant que P_j,$j \neq i$ n'y ait accès
  2. Les requêtes pour R doivent être desservies dans l'ordre où elles ont été faites (si elles sont concurrentes, alors l'ordre est sans importance)
  3. Si tout processus obtenant R relâche éventuellement R, alors toutes les requêtes seront éventuellement desservies
- Supposons qu'initialement, R soit possédé par un seul processus

- Suppositions simplificatrices:
  - les messages sont reçus dans l'ordre où ils sont émis
  - un message, une fois émis, sera éventuellement reçu
- Ces simplifications peuvent être éliminées en numérotant les messages et en implémentant un mécanisme de poignée de mains

Utilité d'un ordre total (*suite*)

- Algorithme:
  - Tout processus P_i possède une file Req_i de requêtes
  - Pour demander R, Pi envoie m={T_m:P_i veut R} à tout P_j, $j \neq i$, et place m dans Req_i
  - Sur réception de m, P_j envoie m'={$T_{m'}$:tigidou} à P_i et place m dans Req_j
  - Pour relâcher R, P_i retire m de Req_i et envoie m''={$T_{m''}$:P*i* relâche R} à tout P_j, $j \neq i$
  - Sur réception de m'', P_j envoie m'''={$T_{m'''}$:tigidou} à P_i et retire m de Req_j
- P_i obtient R quand:
  - dans Req_i, on trouve {T_m:P_i veut R} précédant tous les autres messages «veut R» au sens de ➔
  - dans Req_i, on trouve une réponse {$T_{m'}$:tigidou} de tous les P_j, $j \neq i$ tel que $T_{m'} > T_m$ pour tout $T_{m'}$