Services de temps et de coordination

Module 7

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal
Département de génie informatique et génie logiciel

Sommaire

Services de temps et de coordination

Le temps

- L'heure change d'un fuseau horaire à l'autre, d'une saison à l'autre et d'un pays à l'autre en raison de l'heure avancée.
- La durée des jours change à cause du frottement des marées et des courants du noyau terrestre.
- Le temps universel (UTC): une seconde est 9 192 631 770 périodes du césium 133, nombre de secondes par jour ajustable, point de référence GMT (Greenwich Mean Time).
- L'horloge des ordinateurs dérive avec le temps comme les montres.
- En cas de retard, avancer d'un coup ou lentement.
- En avance, ralentir pour laisser le temps nous ratrapper. . . remonter dans le temps est dangereux.

Problématique du temps dans les systèmes répartis

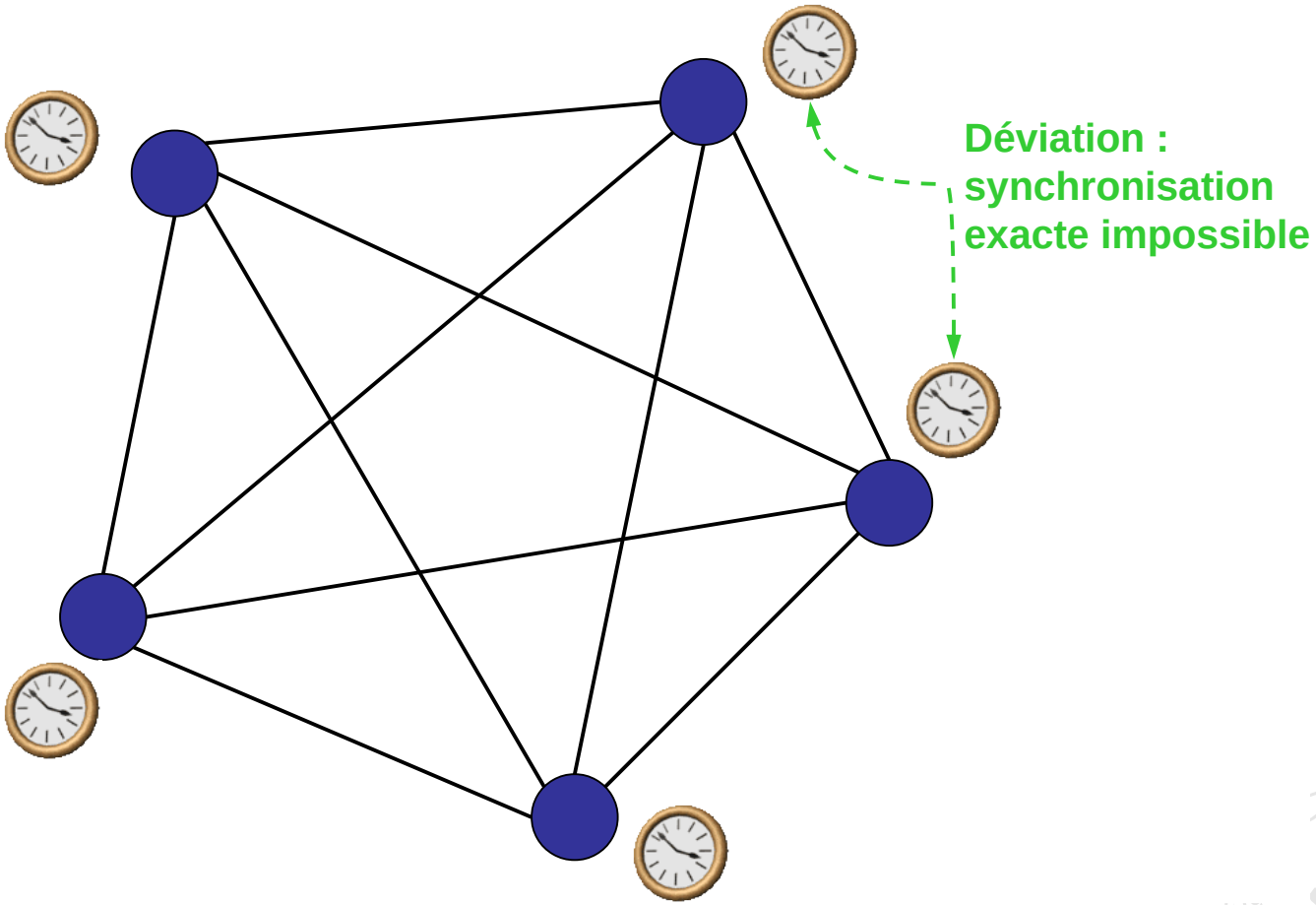
- Le temps est très important dans les systèmes répartis. On veut savoir avec précision quand un évènement est arrivé.
- Il n'y a pas d'horloge globale pour mesurer le temps à travers le système réparti, étant donné le délai pour envoyer par réseau une lecture de temps.
- Algorithmes de synchronisation servent à maintenir la cohérence des données réparties, éliminer les mises à jour redondantes, vérifier l'authenticité des requêtes envoyées au serveur, coordonner certaines opérations, tracer l'exécution du système...

Les horloge physiques

- Horloge d'ordinateur: cristal au quartz qui vibre à une fréquence précise (e.g. à plus ou moins 10^{-6}), avec une interruption après un certain nombre d'oscillations, ce qui génère un tic.
- L'horloge est obtenue par un calcul d'un temps initial (heure lue du circuit RTC mémorisé avec une pile): temps lu au démarrage + nombre de tics depuis ce temps initial x durée d'un tic.
- Les cristaux diffèrent d'un ordinateur à l'autre, ce qui cause une désynchronisation entre les systèmes multi-ordinateurs, appelée dérive des horloges (clock skew).

Systèmes asynchrones

- On ne peut synchroniser parfaitement des systèmes asynchrones.

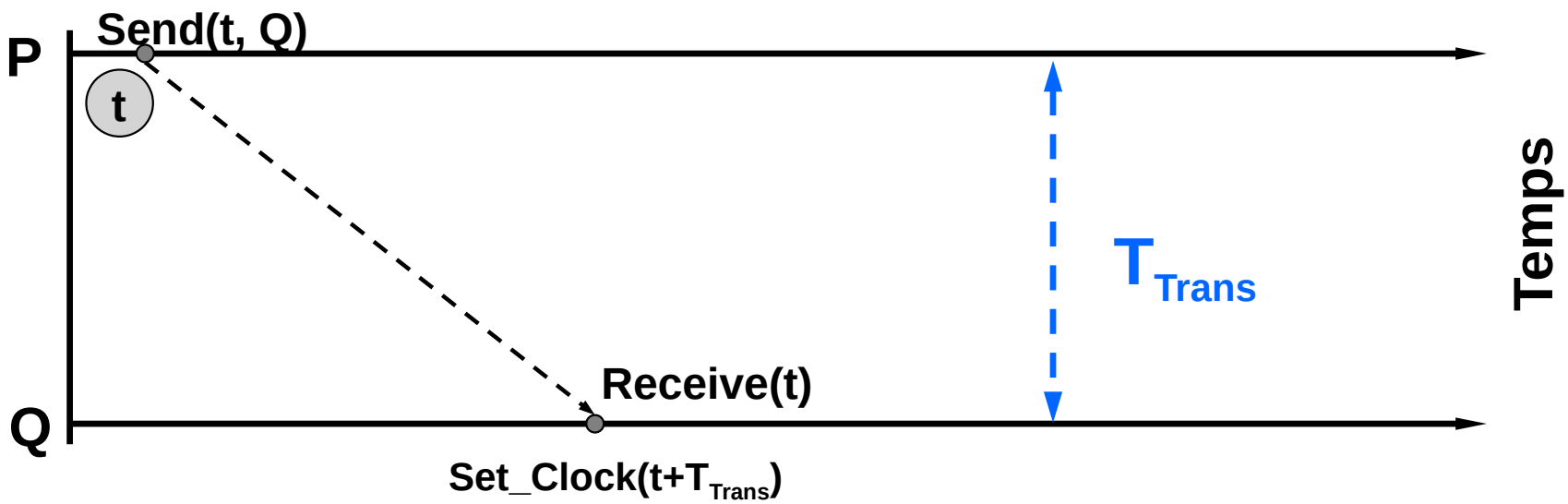


Synchronisation

- Le temps de propagation est limité par la vitesse de la lumière.
- Délais liés à la préemption et au traitement des interruptions pour recevoir un message de synchronisation.
- Le GPS (Global Positionning System) donne l'heure juste à 1us près à travers le monde. La différence d'heure reçue entre les satellites est causée par la distance et donne la position par triangulation.
- Plusieurs ordinateurs (20 à 30) sont connectés à des horloges atomiques ou GPS et agissent comme serveurs de temps primaires. On compte quelques milliers de serveurs de temps secondaires qui prennent l'heure des serveurs primaires via l'Internet. La précision est typiquement de 30ms ou mieux 99% du temps.
- Avec une horloge calibrable, il est possible d'obtenir une précision de 1ms sur plusieurs heures.

Synchronisation des systèmes synchrones

- Cas le plus simple:
  - Deux processus P et Q veulent synchroniser leurs horloges.
  - Le système est synchrone (délai fixe de transmission connu).



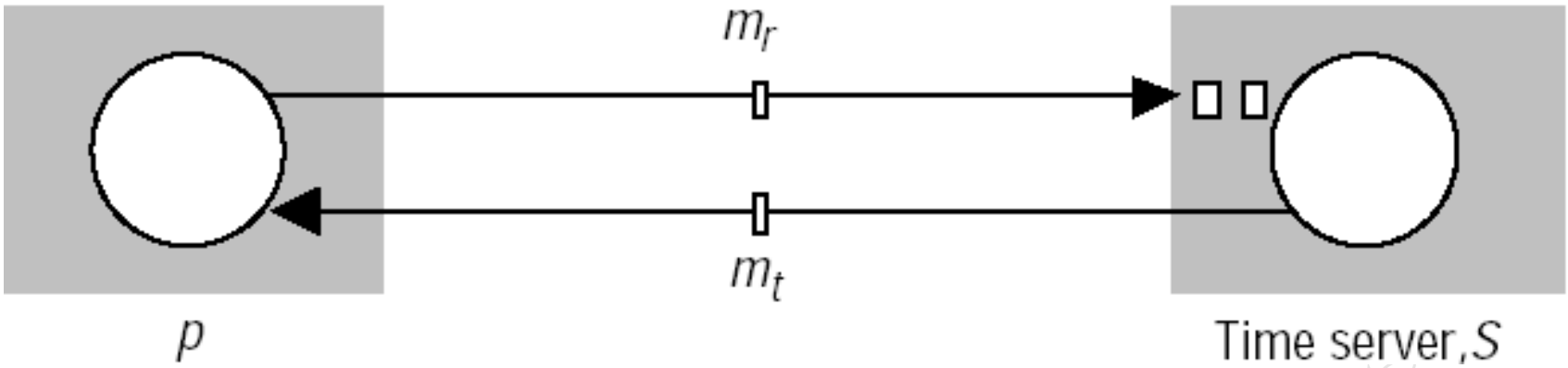
Synchronisation des systèmes asynchrones

- Si le temps de transmission peut varier dans un intervalle U, utiliser une valeur à l'intérieur de U = [MIN, MAX].

scénario	valeur choisie	déviation maximale
1	t + MIN	U
2	t + MAX	U
3	t + (MAX + MIN) / 2	U / 2

Synchronisation par la méthode de Cristian

- Méthode basée sur la synchronisation externe ;
- Elle utilise un serveur central de temps (Time server) qui est connecté à un périphérique recevant des signaux d'une source UTC ;
- Le processus P peut mesurer le temps de réponse à sa requête, T_r, et affecte à son horloge $t + \frac{T_r}{2}$.



Synchronisation par l'algorithme de Berkeley

- Un coordonnateur (primaire) interroge les autres ordinateurs (secondaires) dont les horloges sont à synchroniser.
- Les secondaires envoient la valeur de leur horloge au primaire.
- Le primaire estime leur temps local en observant le délai d'un aller-retour.
- Il calcule la moyenne des valeurs reçues et de sa propre valeur.
- Le primaire renvoie les ajustements nécessaires aux secondaires.
- Si le primaire tombe en panne, un autre est élu comme primaire.

Network Time Protocol (NTP)

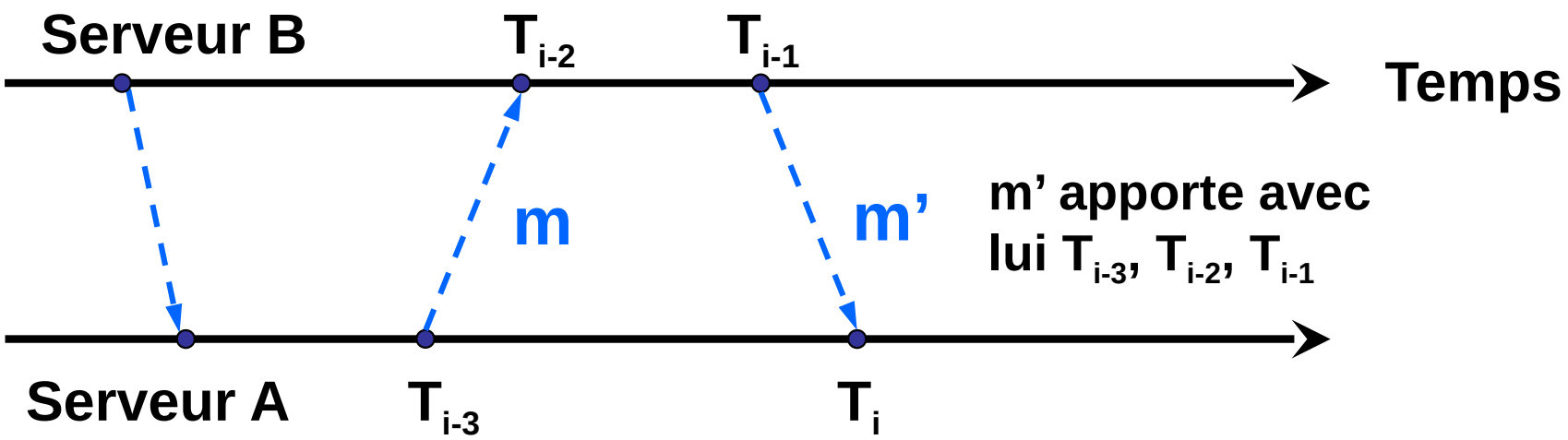
- Défini en 1995 dans le RFC 958.
- Le service NTP est fourni par un réseau de serveurs localisés à travers l'Internet.
- Serveurs primaires connectés directement à une source de temps UTC.
- Serveurs secondaires synchronisés avec les serveurs primaires.

Modes de synchronisation des serveurs

- Diffusion sélective - multicast:
  - Utilisé dans des LANs à haut débit.
  - Un ou plusieurs serveurs diffusent le temps aux démons s'exécutant sur d'autres ordinateurs connectés au LAN.
  - Les démons qui reçoivent les messages affectent à leur horloge les valeurs reçues en considérant un faible délai.
- Appel de procédure:
  - Un serveur accepte les requêtes des autres ordinateurs ;
  - Renvoie une réponse contenant une estampille (valeur de l'horloge locale).
  - Offre une meilleure précision.

Calcul du décalage

- Un processeur demande à l'autre le temps par envoi de messages.
- Les temps d'envoi et de réception sont notés pour en tenir compte dans le calcul de décalage.



Calcul du décalage

- Soit le temps de transmission t de A vers B et t' de B vers A.
- $T_{i-2} = T_{i-3} + t + d$ et $T_i = T_{i-1} + t' - d$
- Soit $a = T_{i-2} - T_{i-3}$ et $b = T_{i-1} - T_i$
- $imprecision = t + t' = a - b$ avec $t > 0$ et $t' > 0$
- $ajustement = \frac{a+b}{2}$
- $d = ajustement + \frac{t'-t}{2}$
- $\frac{a+b}{2} - \frac{a-b}{2} \leq decalage \leq \frac{a+b}{2} + \frac{a-b}{2}$
- $decalage = \frac{a+b}{2} \pm \frac{a-b}{2}$
- On peut envoyer plusieurs messages et retenir ceux dont a et b sont les plus petits. Normalement seuls t et t' varient sur une courte prériode, en les minimisant on réduit l'imprécision.

Services de temps et de coordination

Etat des processus et systèmes répartis

- Collection de N processus Pi où i = 1, 2, 3, ... N.
  - Les processus sont indépendants;
  - Il n'y a pas de mémoire partagée entre les processus.
- Si: état du processus Pi.
  - L'état = valeurs des variables, des ressources ou des objets locaux utilisés par le processus.
- La communication entre les processus ne se fait que par transmission de messages.
- Sur un processus Pi, le passage de l'état Si à l'état Si' est directement associé à l'ensemble des actions pouvant modifier l'état d'un processus (envoi et réception de message).

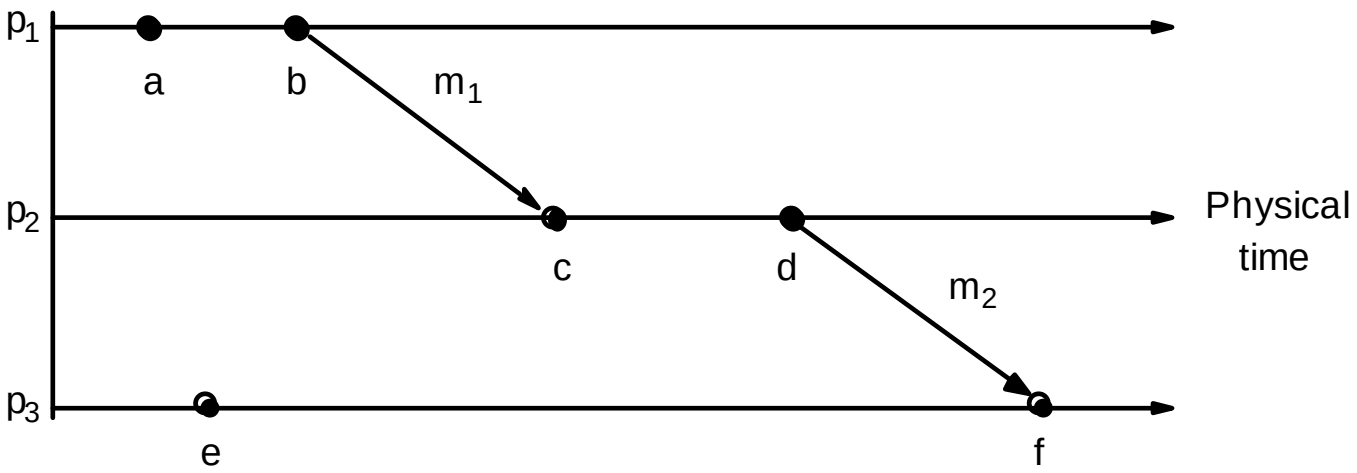
Evènements dans un système réparti

- Un évènement e est l'occurrence d'une seule action.
- Un évènement modifie l'état d'un processus.
  - L'ordre total unique des évènements est noté : $\rightarrow$
  - $e \rightarrow e'$ si l'événement e est survenu avant e' dans Pi
- L'historique de Pi est une série d'évènements survenus dans Pi et ordonnés par la relation $\rightarrow$
  - $History(P_i) = h_i =< e_i^0, e_i^1, e_i^2, ... >$

Horloges logiques

- Souvent, seul l'ordre entre les événements importe.
- Dans un même processus, les événements sont numérotés avec un compteur.
- Lors d'un envoi de message, la correspondance est faite entre le décompte de l'envoyeur et celui du receveur ($Te < Tr$).
- En l'absence de messages, on ne peut rien dire sur l'ordre réel entre deux événements dans des processus différents. On les considère concurrents.

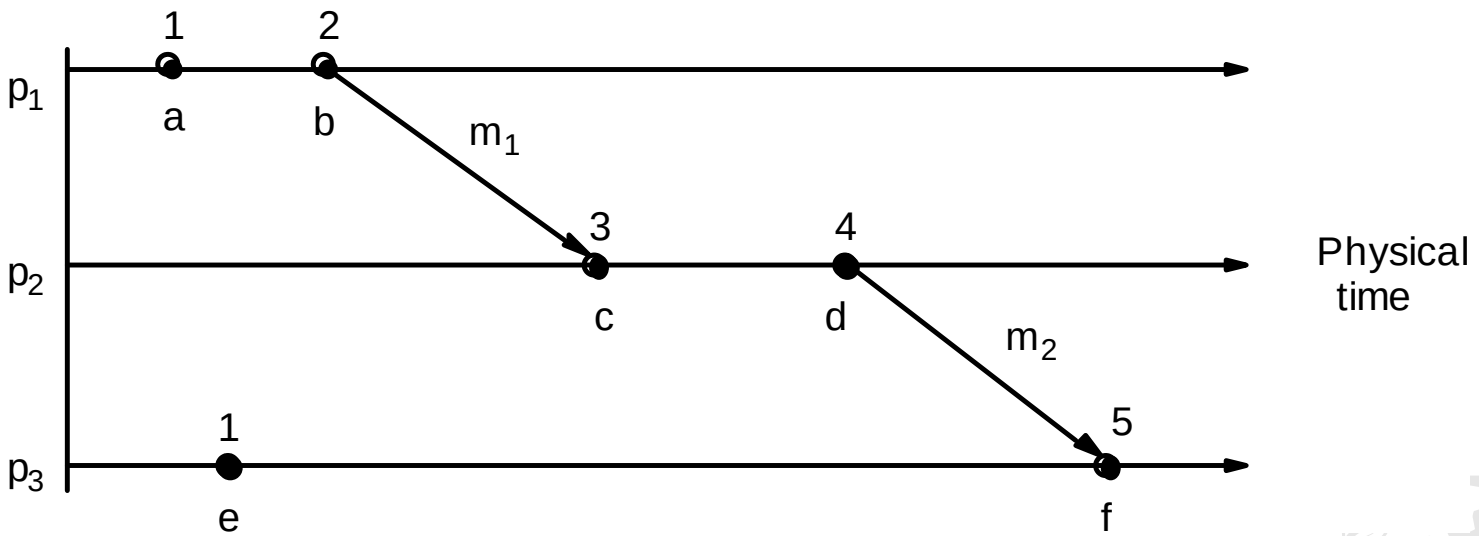
Exemple de relations entre évènements



- $a \rightarrow b$ (puisque les deux se suivent dans p1).
- $b \rightarrow c$ (puisque b = send(m1) et c = receive (m1))
- $c \rightarrow d$ (puisque les deux se suivent dans p2)
- $d \rightarrow f$ (puisque d = send(m2) et f = receive (m2))
- $a \rightarrow f$ (puisque $a \rightarrow b \rightarrow c \rightarrow d \rightarrow f$)
- e et a sont des évènements concurrents, sans ordre démontrable

Exemple d'horloge logiques

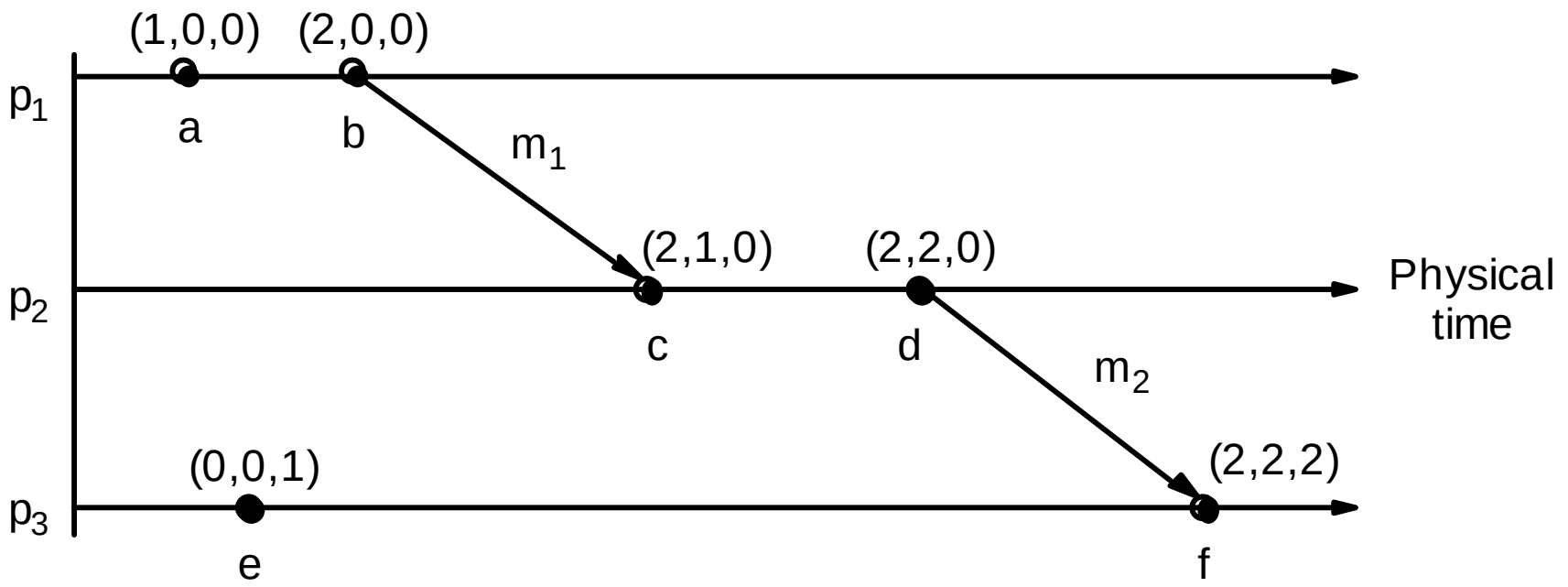
- L'horloge logique est incrémentée de 1 par rapport à la valeur précédente à chaque relation (séquence ou réception de message).
- Si un évènement est postérieur à un autre, sa valeur d'horloge logique est plus grande, l'inverse n'est pas nécessairement vrai (évènements concurrents).



Vecteurs de compteurs d'évènements

- Vecteur de N entiers initialisés à 0 dans chacun des N processus.
- A chaque évènement dans le processus i, Vi[i] est incrémenté.
- A chaque message envoyé par le processus i, Vi est inclus.
- Lorsque le processus i reçoit un vecteur dans un message, il prend pour chaque entrée de son vecteur le maximum entre l'entrée présente et celle reçue (fusion des vecteurs).
- Chaque entrée dans le vecteur du processus i indique le dernier événement dans un autre processus qui peut avoir influencé le processus i (lien de causalité).

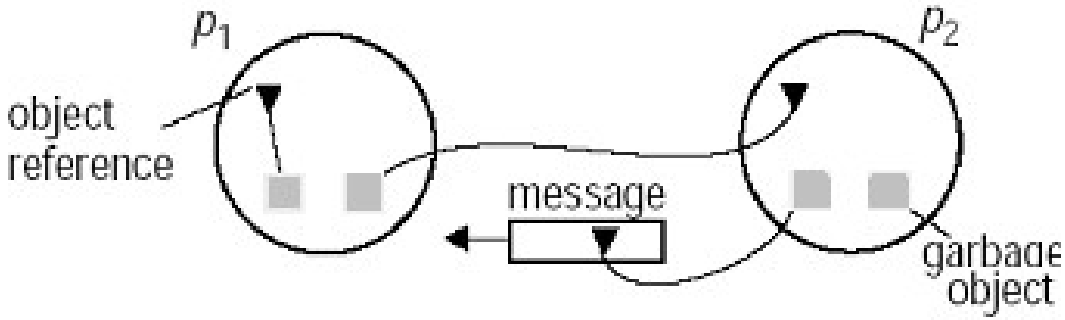
Exemple de vecteurs de compteurs d'évènements



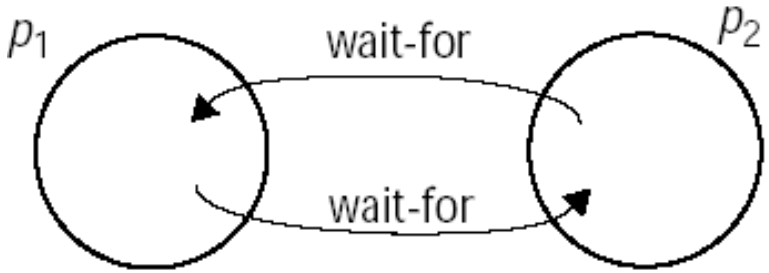
- $V = V'$ ssi $V[j] = V'[j], \forall j = 1...N$
- $V < V'$ ssi $V[j] \leq V'[j], \forall j = 1...N$ et $V \neq V'$
- $e \rightarrow e'$ ssi $V(e) < V(e')$

État global d'un système réparti:

- Pour vérifier si une des propriétés particulières est vérifiée ou non durant son exécution.
- Ramasse-miettes (garbage collection), un objet doit être supprimé s'il n'est plus rejoignable.



- Interblocage réparti s'il y a un cycle dans le graphe d'attente de plusieurs processus.



État global d'un système réparti:

- L'état global est important mais difficile à obtenir sans tout figer.
- Une propriété est stable si une fois atteinte elle demeure vraie (e.g. interblocage).
- Un système est sécuritaire pour une propriété si cette propriété reste vérifiée indépendamment de l'ordre dans lequel les évènements non reliés causalement surviennent (e.g. jamais d'interblocage).
- Un système est vivace pour une certaine propriété si elle devient éventuellement vraie, indépendamment de l'ordre dans lequel les évènements non reliés causalement surviennent (e.g. une demande de verrou doit être satisfaite).

Etat global d'un système réparti

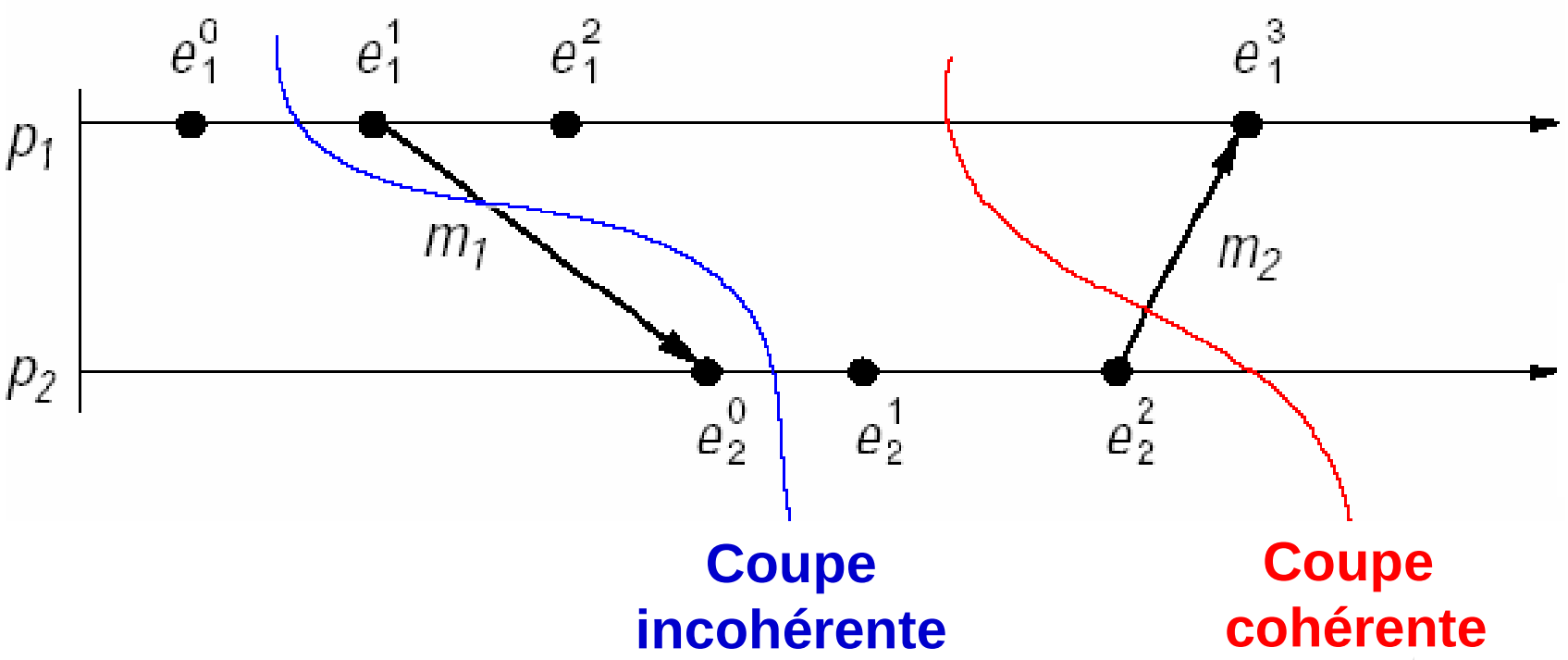
- Un système E est composé de N processus Pi où i = 1, 2, 3, ... N;
- L'historique d'un processus est:
  $History(Pi) = h_i =< e_i^0, e_i^1, e_2^i, ... >$
- L'historique fini est: $h_i^k =< e_i^0, e_i^1, e_i^2, ..., e_i^k >$
- L'historique global du système est: $H = h_1 \cup h_2 \cup ... \cup h_N$
- L'état global du système E est: $S = (S_1, S_2, ..., S_N)$

Cliché cohérent de l'état global

- Une coupe (cut) est un sous-ensemble de l'état global.
- Une coupe C est cohérente si pour chaque événement e qu'elle inclut, tous les événements qui sont survenus avant la coupe sont inclus
- $\forall e \in C, f \rightarrow e \Rightarrow f \in C$

Exemple de coupe dans l'historique d'état

- Un état global cohérent correspond à une coupe cohérente.



Algorithme pour l'obtention d'un cliché cohérent

- Un processus mémorise son état, envoie un message demandant la mémorisation aux autres processus connectés en sortie, et enregistre tous les messages reçus d'autres processus qui n'ont pas encore mémorisé leur état.
- Un processus qui reçoit l'ordre de mémorisation fait de même.
- A la fin, l'état global est l'état de chaque processus plus les messages mémorisés (déjà envoyés avant que le processus d'origine n'ait mémorisé son état mais reçus après que le processus courant ait mémorisé son état).

Calcul d'un cliché cohérent possible

- Il est possible d'enregistrer les événements (changements d'état) de tous les processus. Chaque processus envoie une trace à un moniteur central avec pour chaque événement un vecteur de compteurs d'événements.
- Un état global est défini par un numéro de dernier événement pour chaque processus impliqué. Cet état est cohérent si pour chaque dernier événement, chaque entrée des vecteurs de compteurs d'événements est inférieure ou égale à la même entrée pour le dernier événement inclus pour le processus correspondant. $V(s_i)[i] \geq V(s_j)[i] \quad i, j = 1, 2, ..., N$
- Dans un système synchrone, un état est cohérent si les différences entre les temps des événements sont inférieures à l'imprécision des horloges.

Propriétés d'un cliché cohérent possible

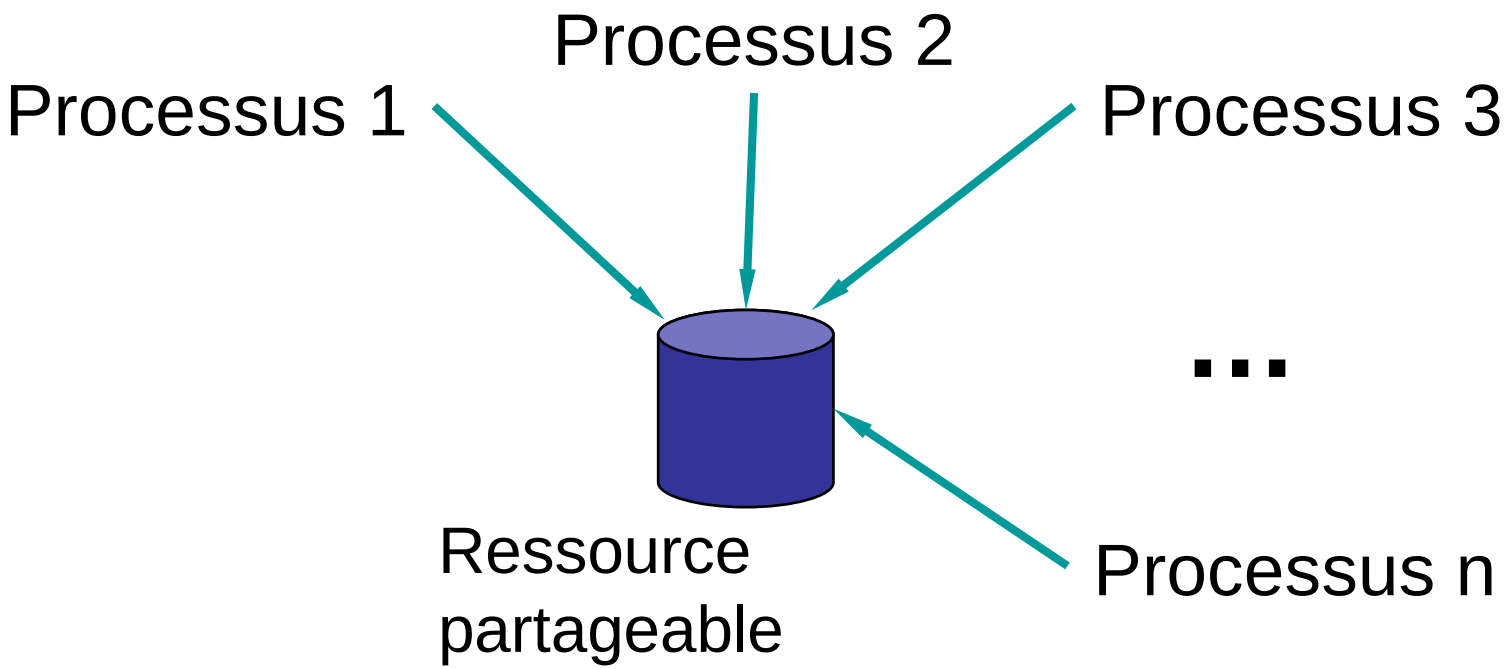
- Construire un treillis d'états globaux possibles (cohérents).
- Traverser le treillis un niveau à la fois.
- Si un des états globaux possibles rejoint a une propriété, cette propriété est possible.
- Si on arrête à chaque état global possible pour lequel la propriété est vraie et qu'on ne puisse ainsi arriver à l'état final, cette propriété est nécessairement vraie.

Services de temps et de coordination

Coordination et temps

- Un des problème fondamentaux des systèmes répartis : le consensus!
  - Comment faire pour mettre d'accord plusieurs processus indépendants?
  - Comment faire pour coordonner leurs actions?
- Comment faire pour résoudre ce problème?
  - Une solution naïve est d'implémenter le modèle primaire-secondaire.
- Est-ce que le problème du consensus est le même pour les systèmes synchrones et asynchrones?

Exclusion mutuelle répartie

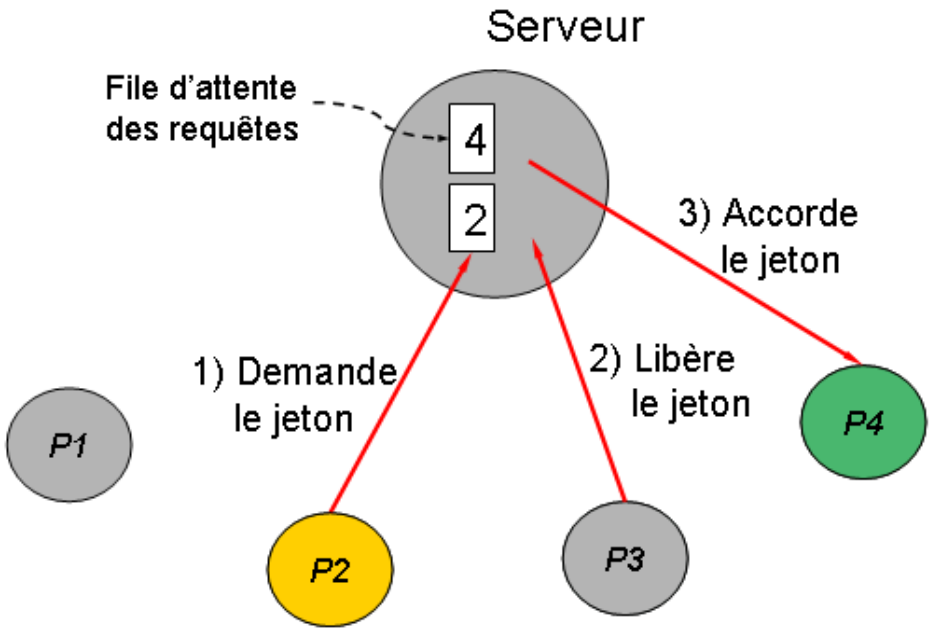


- Exclusion mutuelle est nécessaire pour :
  - Prévenir les interférences ;
  - Assurer la cohérence en cas d'accès simultanés aux ressources.

Synchronisation par exclusion mutuelle répartie

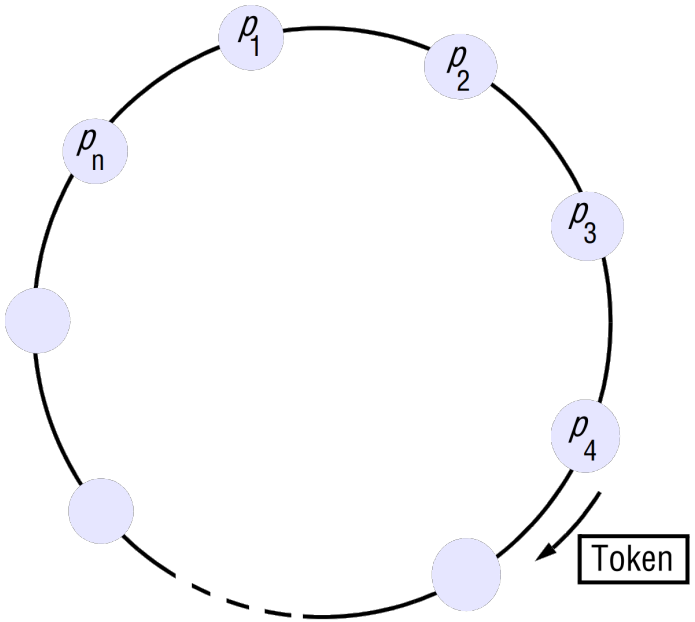
- Sûreté: un seul client à la fois obtient le verrou.
- Vivacité: personne n'est laissé pour compte, chaque client voit éventuellement sa requête satisfaite.
- Ordre: premier arrivé, premier servi, si un client A fait une demande, envoie un message à B, et B fait une demande, l'ordre logique dit que la demande de A arrive avant celle de B. Elle devrait être servie en premier.

Serveur central



- Un client envoie une demande de verrou, le serveur attend que le verrou soit libre, le donne au client, et le client le relâche éventuellement, ce qui permet au serveur de le donner à un autre client.
- Si le serveur est en panne, élection d'un nouveau serveur (majorité de clients), et vérification de tous les clients pour voir qui possède des verrous ou a soumis une requête (problème si clients non rejoignables en raison de division du réseau).
- Exemple: serveur `lockd` sous NFS.

Anneau à jeton



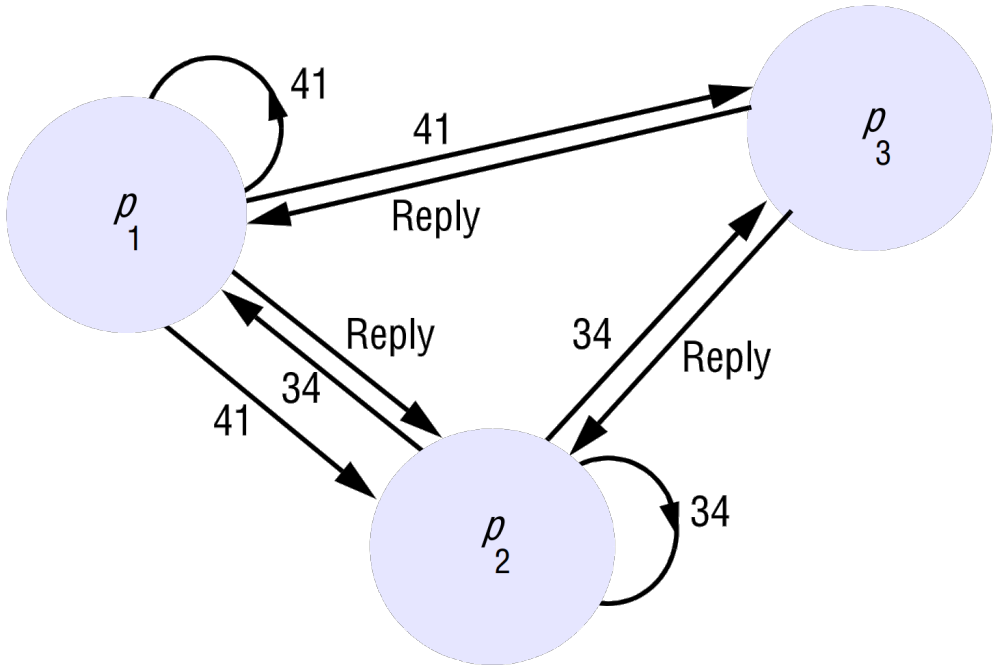
- Le verrou passe de processus en processus constamment.
- Inutile lorsque personne ne requiert le verrou.
- Problème dès qu'un client fait défaut.
- Ne respecte pas premier arrivé, premier servi.

Exclusion par envoi à tous

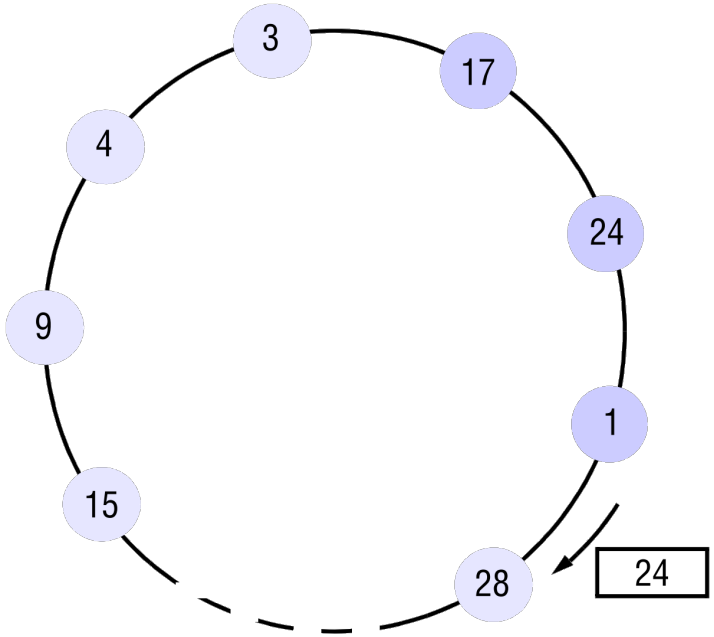
- Envoyer une requête à tous.
- Recevoir le O.K. de tous.
- Si on a le verrou, attendre d'en avoir terminé avant de répondre O.K.
- Si on veut le verrou et notre demande est antérieure (estampille de temps), attendre le verrou et d'en avoir fini avant de répondre O.K.
- Demande n messages (ou 2n - 2 sans message à tous).
- Tous les clients doivent être actifs.
- Moins bon que serveur central.

Exclusion par envoi à tous (suite)

- p_3 ne veut pas entrer dans la section critique
- p_1 et p_2 demandent accès à la section critique en même temps:
  - requête de p_1 à $T = 41$
  - requête de p_2 à $T = 34$
- Processus:
  - p_3 répond immédiatement aux requêtes
  - p_2 reçoit la requête de p_1, il voit que $T(p_2) < T(p_1)$ et ne répond pas (hold p_1)
  - p_1 reçoit la requête de p_2, il voit que $T(p_2) < T(p_1)$ et répond immédiatement
  - p_2 reçoit la réponse de p_1 et entre dans la section critique; quand il la quitte, il répond à p_1
  - p_1 reçoit la réponse de p_2 et entre dans la section critique



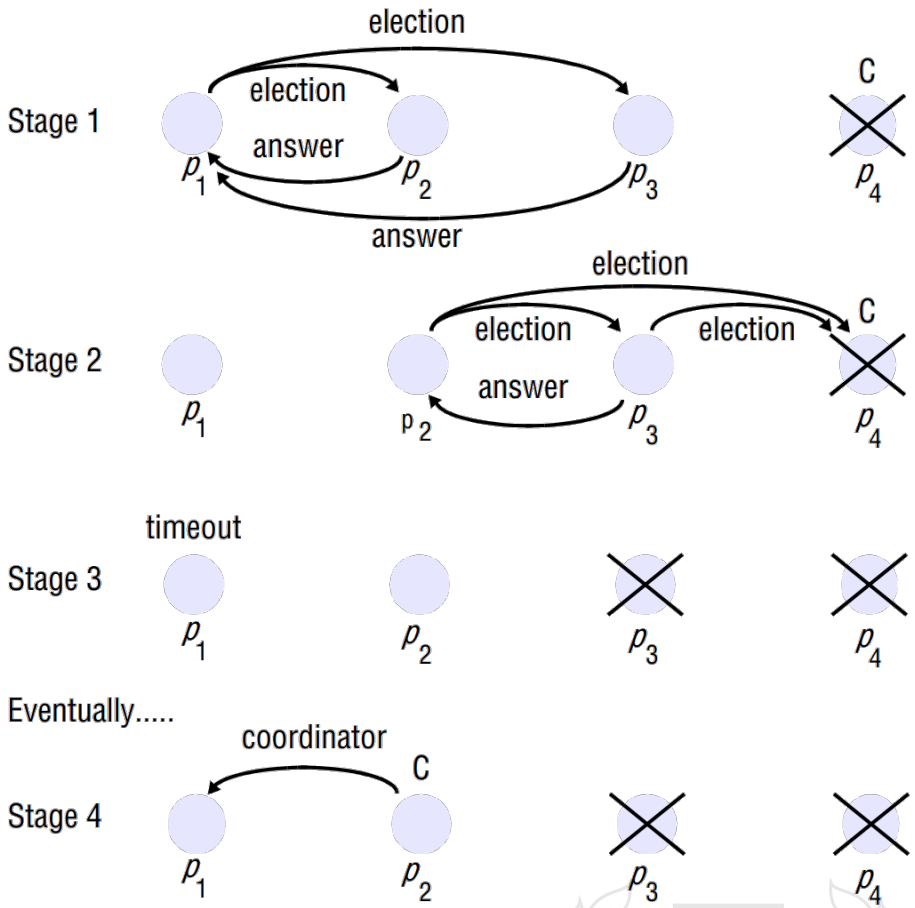
Election en anneau



- Un participant envoie son message d'élection avec son ID.
- Le prochain participant envoie max(ID reçu, ID) à son voisin.
- Si un participant reçoit un message avec son ID, il se déclare élu et propage la nouvelle.
- Ceci prend un maximum de 3n - 1 messages (ID maximum n - 1, se découvrir élu n, propager la nouvelle n).

Election hiérarchique (*Bully algorithm*)

- Lorsqu'un nouvel ordinateur est connecté, ou si le coordonnateur ne peut être rejoint, déclencher une élection.
- Envoyer un message d'élection à ceux de plus haute priorité.
- Pas de réponse, il se proclame coordonnateur et le signale à tous.
- Une réponse, il attend un message de proclamation qui devrait suivre.
- Demande $n - 1$ messages dans le meilleur des cas, $n \times n$ si tous les processus déclenchent une élection en même temps.



Services de temps et de coordination

Conclusion

- Chaque ordinateur a son horloge asynchrone et la synchronisation de temps est imparfaite.
- Pour plusieurs applications, un ordre relatif, une horloge logique, suffit.
- Exclusion mutuelle, élection ou consensus, les approches les plus distribuées ou les plus "démocratiques" ne sont pas nécessairement les plus efficaces.
- Attention aux pannes multiples, répétées, au partitionnement de réseau...

Résumé