Systèmes et algorithmes répartis

Tolérance aux fautes

Philippe Quéinnec

Département Informatique et Mathématiques Appliquées
ENSEEIHT

10 juillet 2018

plan

Plan

Sûreté de fonctionnement

- *Availability*, disponibilité : capacité à être prêt à délivrer un service
- *Reliability*, fiabilité : continuité de service
- *Safety*, sécurité/sûreté : absence d'effets catastrophiques sur l'environnement
- *Security*, sécurité : préservation de la confidentialité et de l'intégrité des informations
- *Repairability*, maintenabilité : capacité à restaurer un service correct après une défaillance

Mesures de fiabilité et disponibilité

Fiabilité

- MTTF (*mean time to failure*) : temps moyen jusqu'à la prochaine panne
- MTTR (*mean time to repair*) : temps moyen de réparation
- MTBF (*mean time between failure*) = MTTF + MTTR

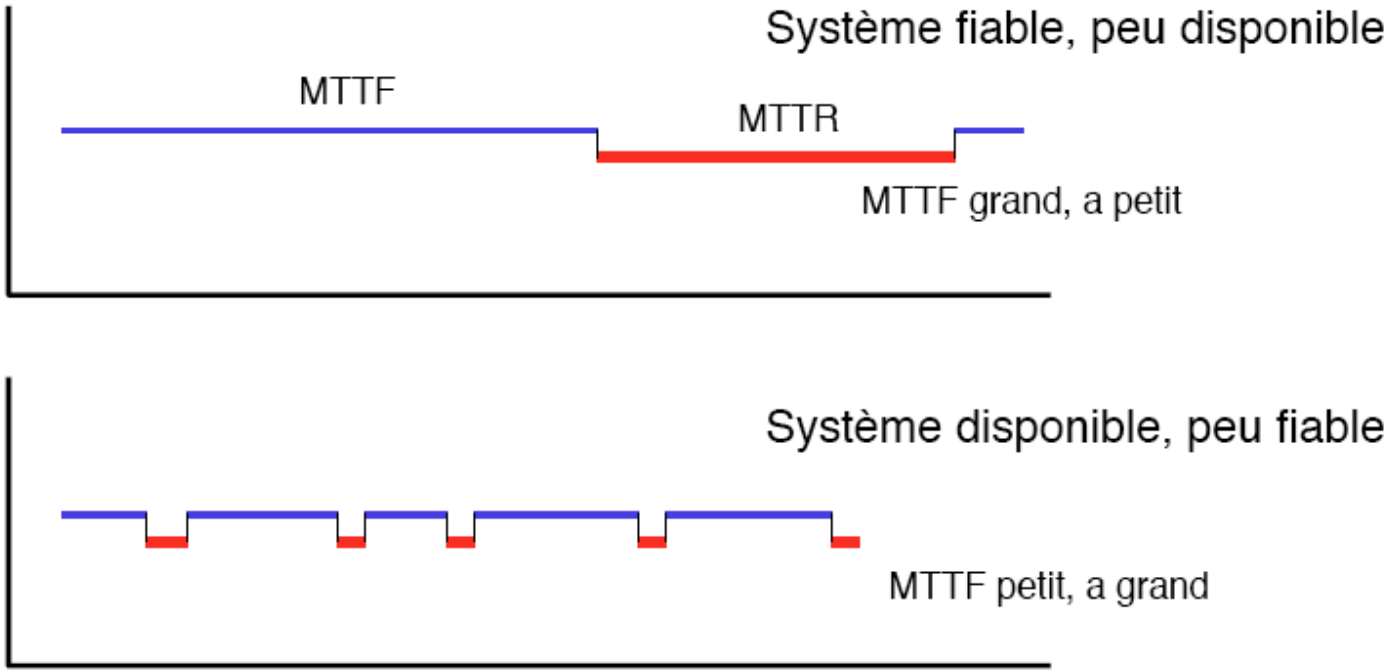
Disponibilité

- Disponibilité moyenne = fraction du temps où le système est disponible
- Pannes franches : dispo moyenne = MTTF/(MTTF+MTTR)

99.99%	(4 nines)	indisponible 50 min / an
99.999%	(5 nines)	indisponible 5 min / an

Différence entre fiabilité et disponibilité

- Indisponibilité = 1 - disponibilité = $MTTR/(MTTF + MTTR)$
  $\approx MTTR/MTTF$ (si $MTTR \ll MTTF$)
  $\Rightarrow$ diviser MTTR par 2 = doubler MTTF
- Différence entre fiabilité (MTTF) et disponibilité (a)



(dessin : S. Krakowiak)

Le vocabulaire

Défaillance (*failure*)

Déviation par rapport à la spécification.
Manifestation d'une erreur vis-à-vis du service.

Erreur (*error*)

Partie de l'état du système entraînant la défaillance.
Manifestation d'une faute vis-à-vis du système.

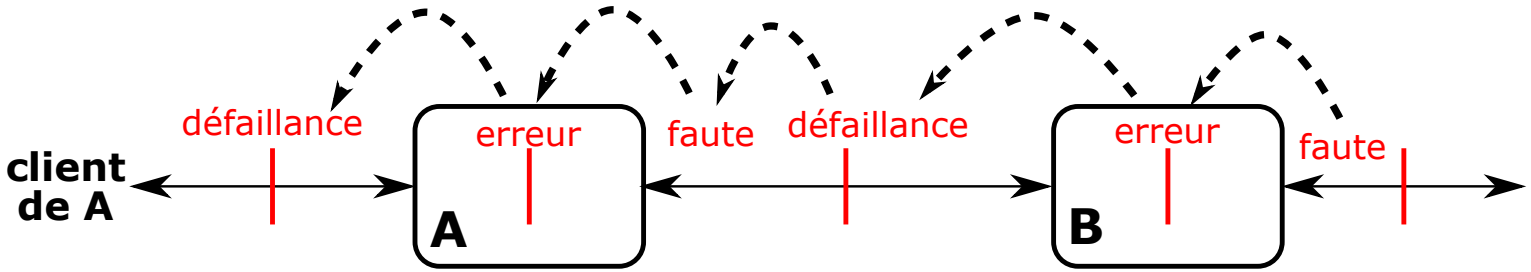
Note : une erreur est susceptible de causer une défaillance mais pas nécessairement (erreur latente, erreur compensée. . .)

Faute (*fault*)

Cause de l'erreur.

Ex : faute de programmation $\Rightarrow$ division par 0 $\Rightarrow$ arrêt du service
virus de la grippe $\Rightarrow$ malade $\Rightarrow$ absent en cours

Cascade/Propagation



- Le bon fonctionnement de A dépend du bon fonctionnement de B
- Une défaillance de B constitue une faute pour A
- Cette faute peut à son tour provoquer une erreur interne de A, causant une défaillance de A

Classification des défaillances

Un serveur doit rendre un service correctement, i.e. conformément à une spécification (valeur, transition, vitesse...).

- défaillance d'omission : le service ignore une requête
- défaillance temporelle : la réponse est fonctionnellement correcte mais pas au bon moment
  - défaillance temporelle d'avance : réponse trop tôt
  - défaillance temporelle de retard : réponse trop tard (défaillance de performance)
- défaillance de réponse : réaction inadaptée à la requête
  - défaillance de valeur : valeur renvoyée erronée
  - défaillance de transition : transition interne erronée
- défaillance arbitraire ou byzantine

Défaillance d'arrêt

Si, suite à une première omission, le serveur ignore toutes les requêtes jusqu'à ce qu'il soit redémarré, on parle de panne franche, défaillance d'arrêt, *crash failure* ou *fail stop*.

Redémarrage

- *amnesia-crash* : état initial indépendant de l'état avant crash
- *pause-crash* : état identique à l'état avant crash
- *halting-crash* : pas de redémarrage (crash définitif)

Défaillance d'arrêt = cas le plus simple, auquel on essaie de se ramener (arrêt forcé sur tout type de défaillance)

Note : un client ne peut pas distinguer entre un serveur « planté », un serveur lent ou une défaillance du réseau.

Les moyens de la sûreté de fonctionnement

Construction

- Prévention des fautes : comment empêcher par construction, l'occurrence ou l'introduction de fautes
- Tolérance aux fautes : comment fournir un service conforme à la spécification, en dépit de fautes

Validation

- Élimination des fautes : comment réduire, par vérification et tests, la présence de fautes
- Prévision des fautes : comment estimer, par évaluation, la présence et l'apparition de fautes

Exemples

- Écrire un programme sans (trop de) bugs = prévention (méthodologie de développement) + élimination (vérification de types, preuves, tests...)
- S'adapter à un environnement hostile = prévision (à quoi faut-il s'attendre ?) + tolérance (que faut-il faire ?)
- Empêcher le « piratage » = prévision (quelles attaques ?) + prévention (cryptographie...)

Principes de la tolérance aux fautes

Traiter les erreurs

- Détecter l'existence d'un état incorrect (erreur)
- Remplacer l'état incorrect par un état correct

Traiter les fautes

- Composants multiples, pour réduire la probabilité qu'une faute dans l'un d'entre eux conduise à une défaillance de l'ensemble
- Traitements multiples

Principe : la redondance

- Redondance d'information (détection d'erreur)
- Redondance temporelle (traitements multiples)
- Redondance architecturale (composants multiples)

Méfiance

Nombreux pièges :

- Mauvaise analyse des fautes possibles
- Réaction inappropriée due à des hypothèses mal formulées : le remède aggrave la situation
- Redondance mal conçue :
  - Éléments redondants ayant des modes de défaillance identiques
  - Faux sentiment de sécurité

Il n'existe pas de méthodes de tolérance aux fautes valables dans l'absolu, seulement des méthodes adaptées à des hypothèses particulières d'occurrence de fautes. Ces hypothèses doivent donc être explicitement formulées, après analyse soigneuse. S.Krakowiak

Traitement des erreurs

Compensation (*error masking*)

- L'état possède une redondance interne suffisante pour détecter et corriger l'erreur
- Totalement transparent pour l'utilisateur
- Exemple : code correcteur, vote majoritaire interne

Recouvrement (*error recovery*)

- Remplacer l'état incorrect par un état correct
- Détecter l'erreur
  - explicitement (vérifier des propriétés spécifiques à l'état)
  - implicitement (observation d'anomalies : délai de garde, protection mémoire...)
- Deux techniques de recouvrement :
  - Poursuite (*forward recovery*)
  - Reprise (*backward recovery*)

Recouvrement

Poursuite (*forward recovery*)

- Reconstruction d'un état correct, sans retour en arrière
- Reconstruction souvent partielle, service dégradé
- Technique spécifique à l'application et aux types d'erreurs possibles

Reprise (*backward recovery*)

- Retour en arrière vers un état antérieur dont on sait qu'il est correct
- Nécessite la capture et la sauvegarde de l'état
- Technique générale
- Coûteuse en temps et place

Reprise

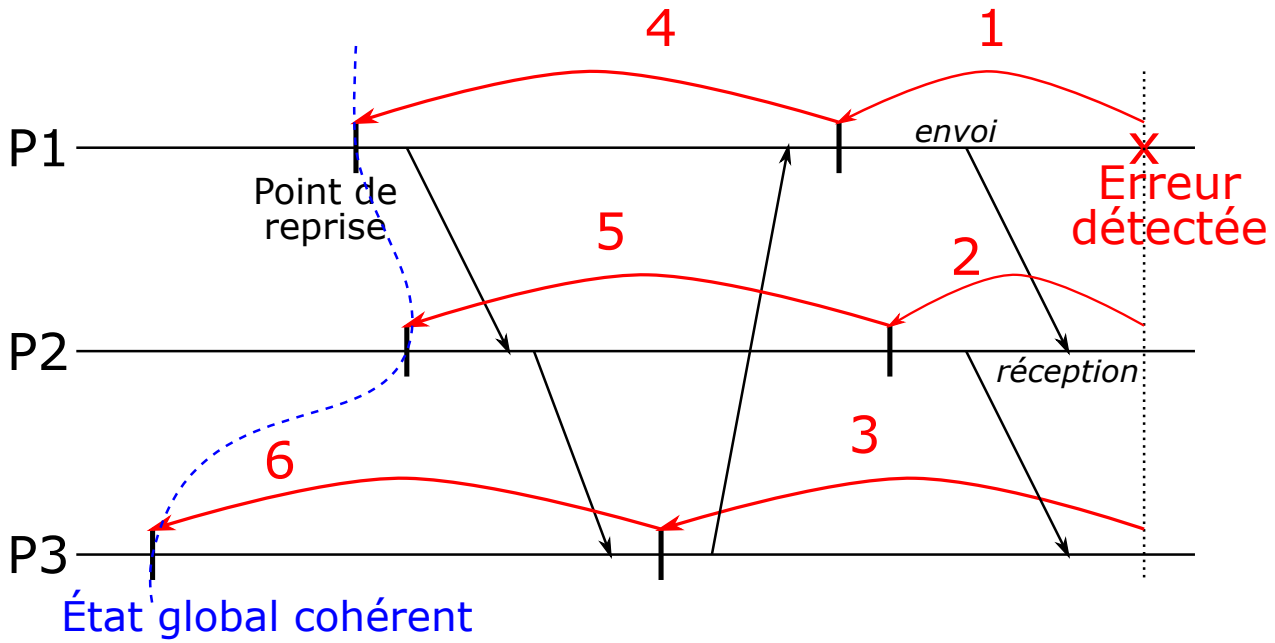
Reprise

- Le système est remis dans un état précédant l'occurrence de l'erreur
- $\Rightarrow$ points de reprise pour sauvegarder l'état (périodiquement, ou sur transition importante)

- La sauvegarde et la restauration doivent être atomique (pas d'interférences)
- L'état des points de reprise doit lui-même être protégé contre les fautes

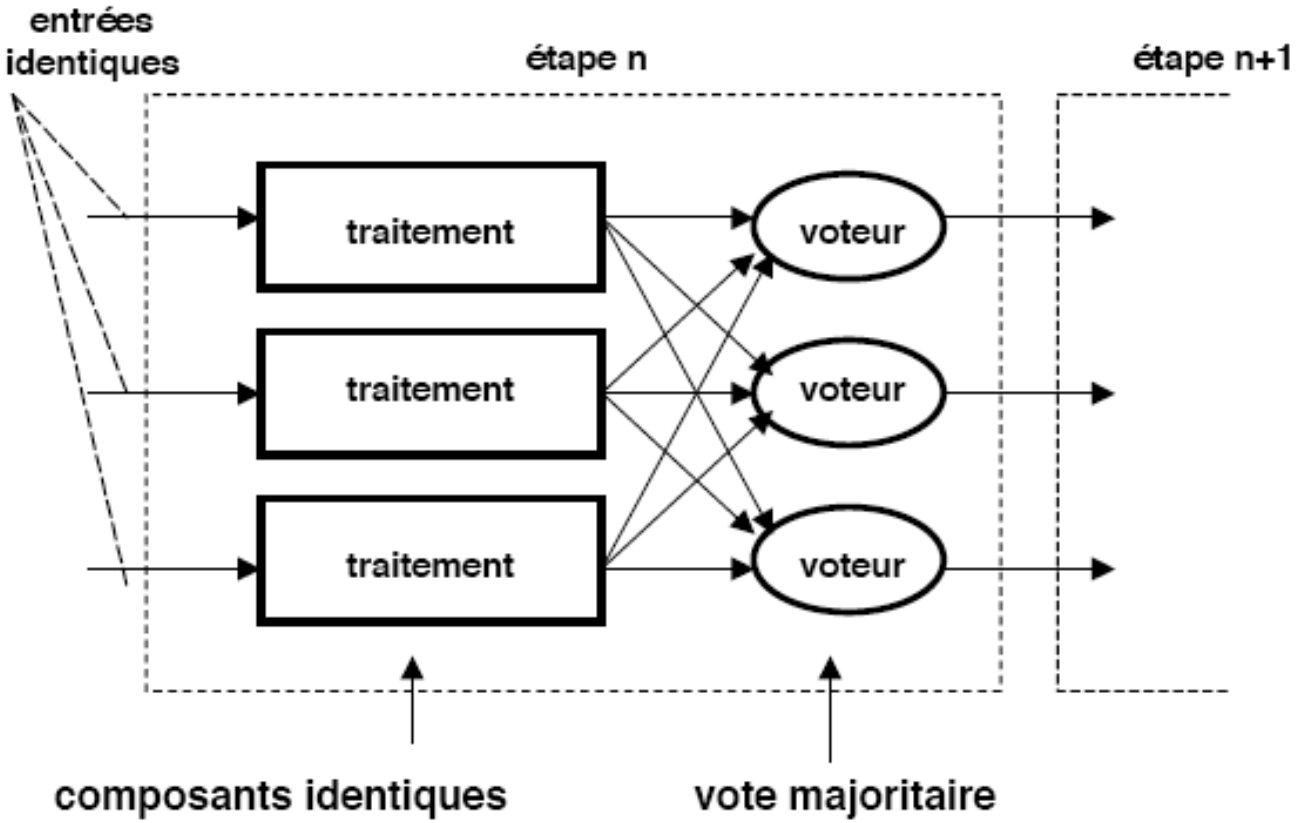
Points de reprise en réparti

- Soit prise de clichés ($\Rightarrow$ état cohérent possible)
- Soit des points de reprise individuels + construction d'un état global cohérent
  $\Rightarrow$ risque d'effet domino



Compensation

Exemple : vote majoritaire



Résiste à la défaillance d'un composant de traitement et d'un voteur par étape

(dessin : S. Krakowiak)

Plan

Serveur tolérant aux fautes

Redondance : N serveurs pour résister $N-1$ pannes franches

Redondance froide (reprise)

- Points de reprise
- Serveurs de secours prêts à démarrer

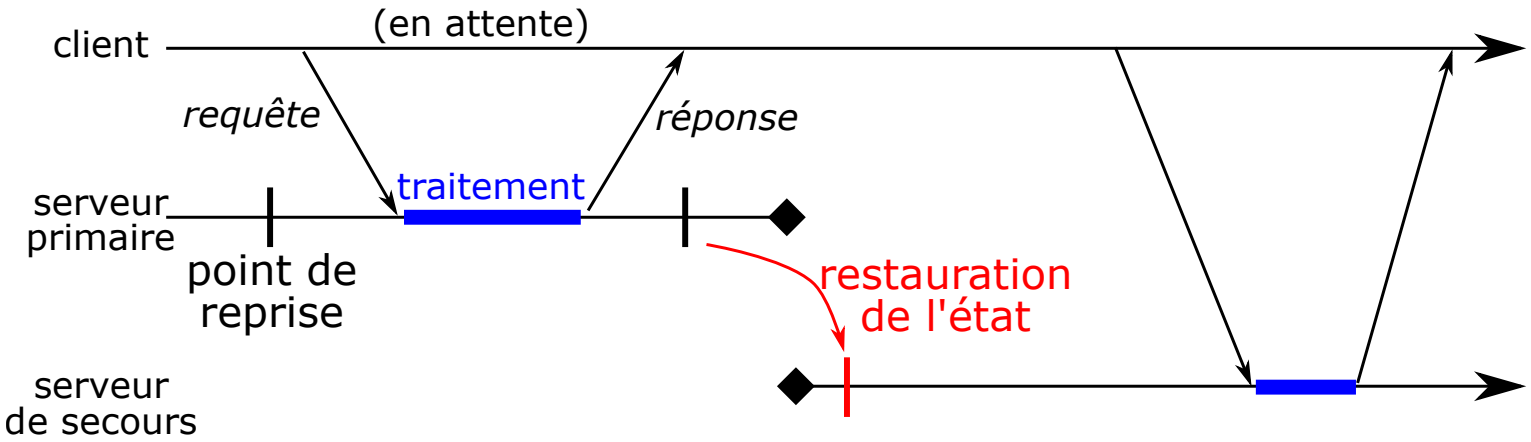
Redondance passive (poursuite)

- Serveur de primaire – serveurs de secours
- Un seul serveur (le primaire) exécute les requêtes des clients
- La panne du primaire est visible des clients

Redondance active (compensation)

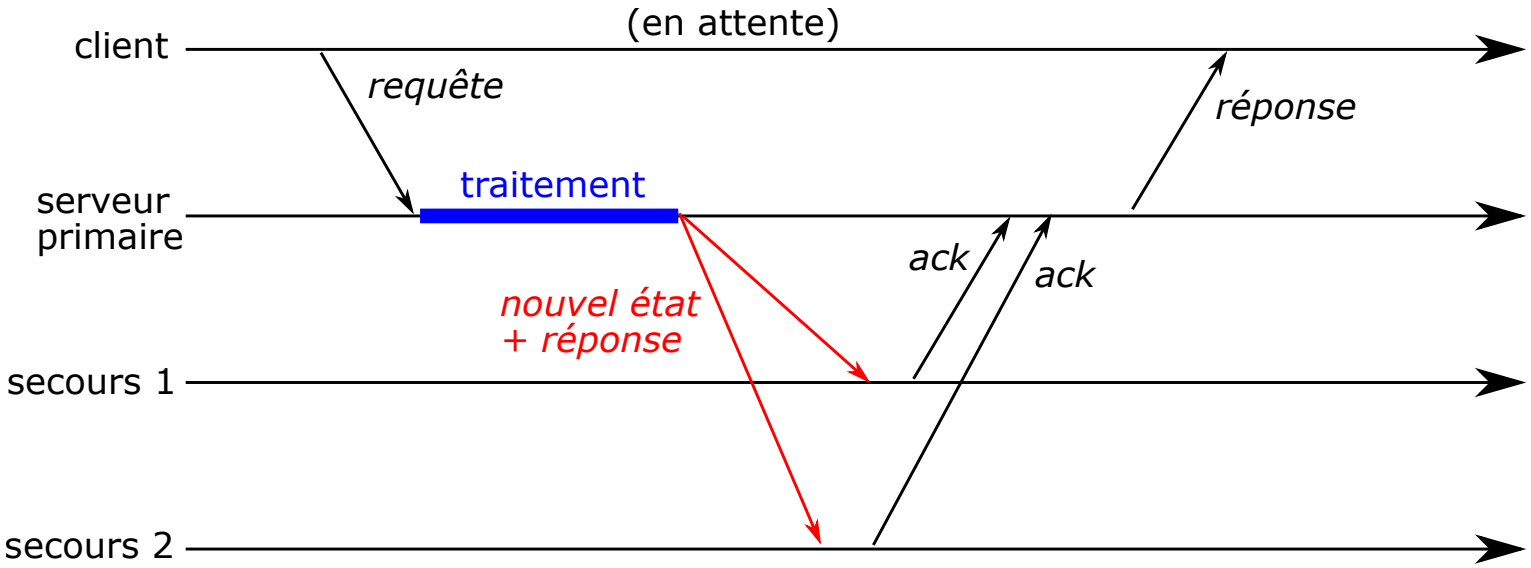
- N serveurs symétriques exécutant toutes les requêtes
- La panne d'un serveur est invisible aux clients (tant qu'il reste un serveur !)

Serveur primaire – secours froids



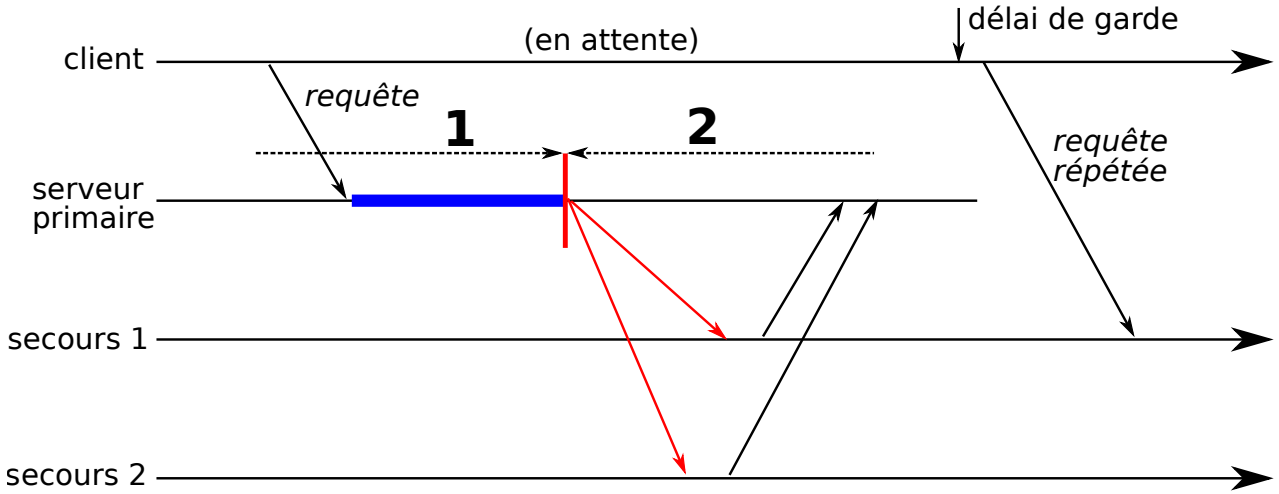
- Panne visible par le client
- Points de reprise : périodiquement, ou à chaque transition importante
- Simple mais latence de reprise
- Retour en arrière ⇒ requêtes traitées mais oubliées

Serveur primaire – serveurs de secours



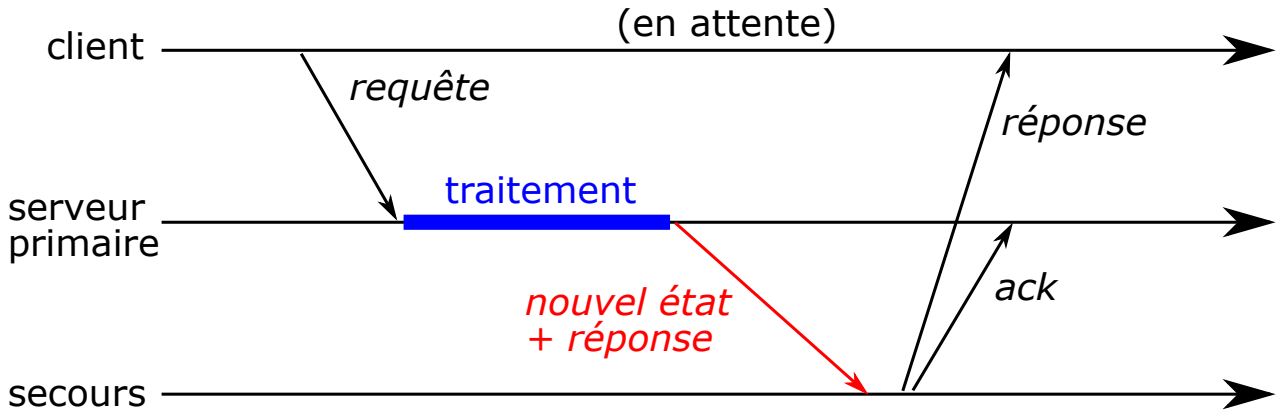
- Diffusion fiable (pas de diffusion partielle)
- Cohérence : quand le primaire répond, tous les serveurs non en panne sont dans le même état
- Poursuite : tout serveur de secours peut remplacer le primaire en cas de panne

Défaillance du serveur primaire



- Détection de la défaillance par le client et par les secours
- Le client renvoie sa requête au secours devenu primaire (élection ou ordre défini à l'avance)
- Diffusion fiable ⇒ tous les serveurs sont à jour, ou aucun :
  - Si la panne est arrivée en 1 (aucun serveur à jour) ⇒ comme une nouvelle requête
  - Si la panne est arrivée en 2 (tous les serveurs à jour) ⇒ renvoie de la réponse déjà calculée
  - Les requêtes ont un identifiant unique, pour distinguer 1 et 2.

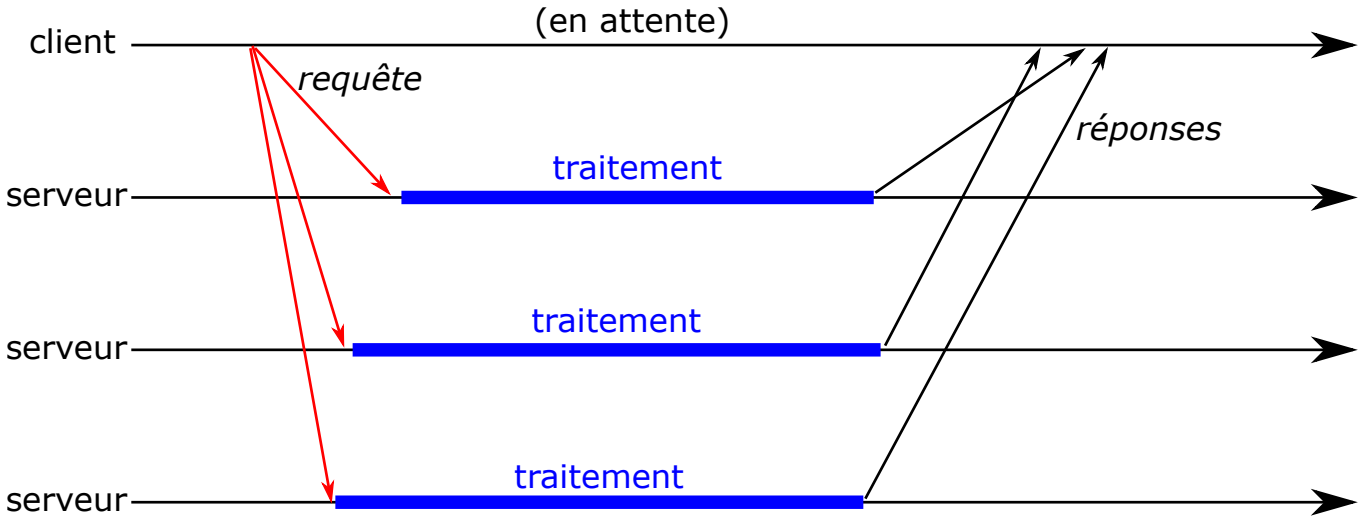
Cas particulier : 1 seul secours



- Protocole plus simple (pas besoin de diffusion fiable)
- Réponse envoyée par le secours

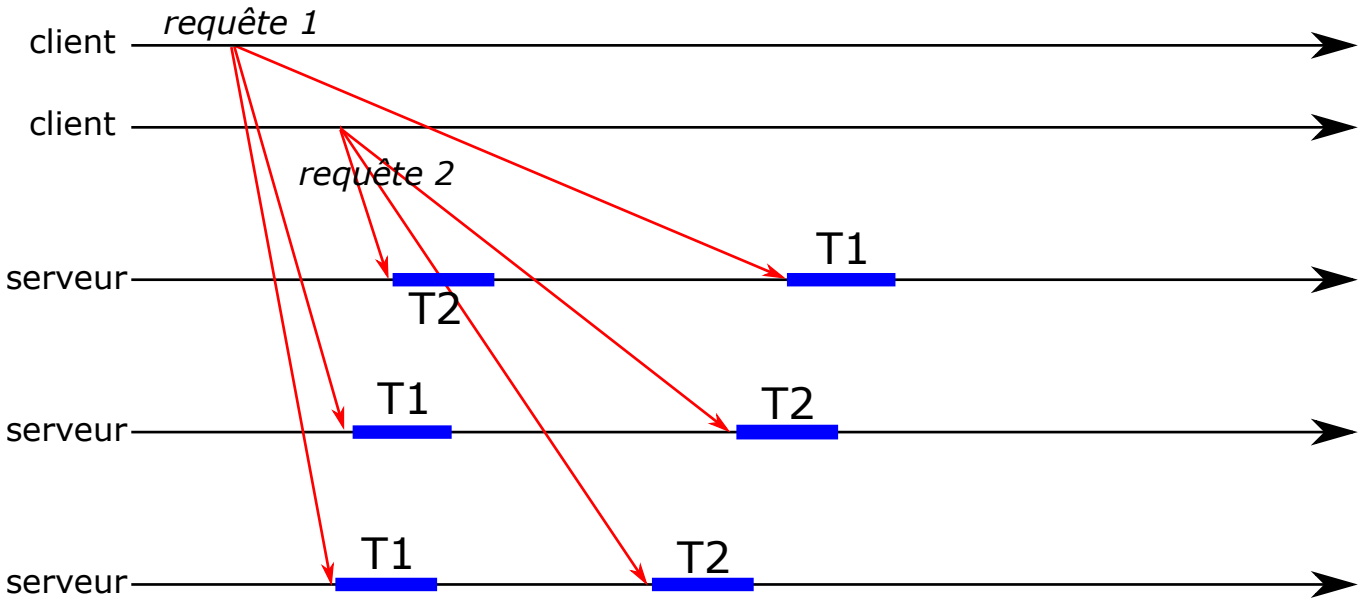
1. *A principle for resilient sharing of distributed resources*, Peter Alsberg and John Day. Proceedings of the 2nd international conference on Software engineering, October 1976.

Redondance active



- Tous les serveurs sont équivalents et exécutent le même traitement
- Le client choisit la première réponse (panne d'arrêt), ou la réponse majoritaire (panne arbitraire)
- Diffusion fiable et totalement ordonnée (atomique) :
  - Message délivré par tous les destinataires ou aucun
  - Deux messages différents sont délivrés dans le même ordre sur tous les destinataires communs

Nécessité de la diffusion atomique



Le serveur 1 traite la requête 2 avant la 1, alors que les serveurs 2 et 3 traitent 1 puis 2 $\Rightarrow$ état incohérent si les traitements modifient l'état des serveurs (traitements non commutatifs)

Modèle : réplication de machine à états (Lamport 1984)

Machine à états

$M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \mathcal{T}, \omega, s_0)$ où
$\mathcal{S}, \mathcal{I}, \mathcal{O}, s_0$ = ensemble d'états / d'entrée / de sortie / état initial
$\mathcal{T}$ = fonction de transition $\mathcal{S} \to \mathcal{I} \to \mathcal{S}$
ω = fonction de sortie $\mathcal{S} \to \mathcal{I} \to \mathcal{O}$
$\mathcal{T}$ et ω sont déterministes.

Principe

- Répliquer la machine à états décrivant le service
- $\mathcal{I}$ = requêtes des clients, $\mathcal{O}$ = réponses aux clients
- Évolution asynchrone des réplicas (chacun à son rythme)
- Même ordre de traitement des requêtes + déterminisme $\Rightarrow$ même séquence de sorties pour tous les réplicas
- Défaillance détectée par divergence des états ou des sorties

Comparaison redondance froide/passive/active

- Mécanismes nécessaires :
  - Diffusion fiable pour passive, fiable-atomique pour active
  - Coordination des départs/arrivées (notion de vue, à suivre)
- Ressources :
  - Serveur froid : éteint ; reprise longue
  - Serveur primaire : serveurs de secours non utilisés $\Rightarrow$ gratuits / utilisés pour autre chose ; reprise non immédiate
  - Redondance active : exécutions superflues ; reprise immédiate
- Transparence pour le client : uniquement en redondance active
- Sensibilité aux bugs logiciels : même état + même entrée $\Rightarrow$ même bug $\Rightarrow$ tous se plantent... (redondance froide moins sensible : retour en arrière)
- Système critique : combinaison des trois techniques compensation/poursuite/reprise : 2 serveurs actifs + 1 secours passif + 1 secours froid avec points de reprise

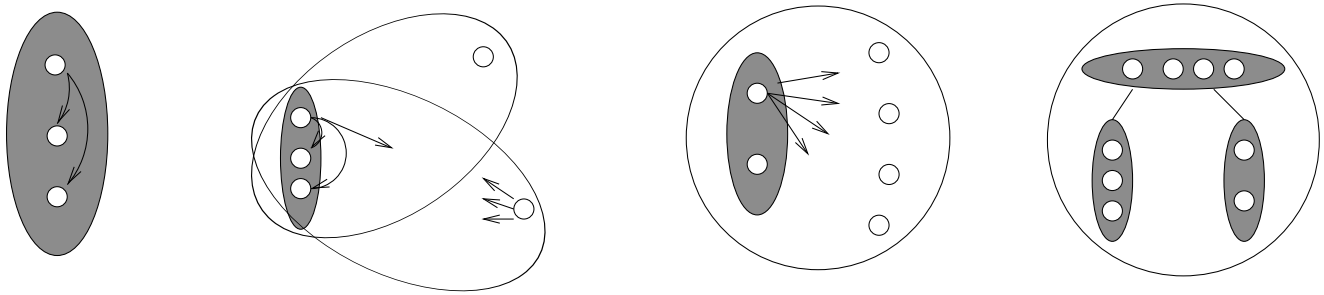
Groupes de processus

Protocoles de groupes

Gestion d'un ensemble de processus avec des opérations pour

- l'appartenance : évolution de la composition du groupe, connaissance de la composition du groupe
- la diffusion : communication vers un ensemble de processus

Motivations : ensemble de processus se comportant comme un processus unique, avec tolérance aux fautes (réplication)



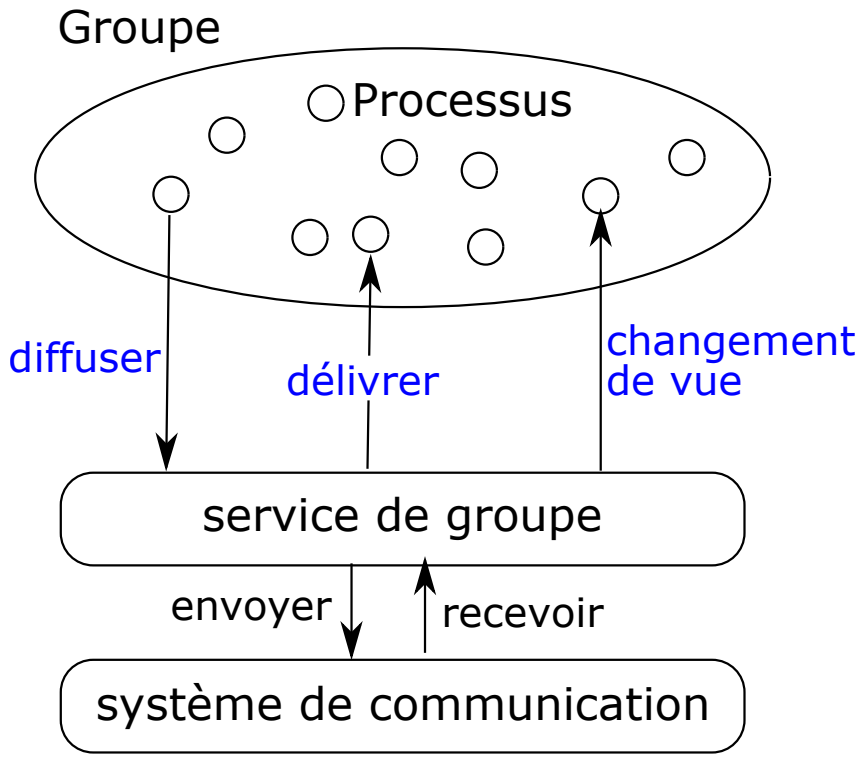
Groupe de pairs | Groupe client-serveur | groupe de diffusion | groupe hiérarchique

1. *Reliable Distributed Computing with the Isis Toolkit*, Kenneth P. Birman and Robbert van Renesse. IEEE Computer Society Press, 1994.

Service de groupes : interface

- diffuser(p,m) : le processus p diffuse le message m au groupe
- délivrer(p,m) : le message m est délivré à p
- changement-de-vue(p,id,V) : une nouvelle vue V (= ensemble de processus) identifiée par id est délivrée au processus p

La diffusion se fait toujours dans la vue courante



Protocoles de groupe : l'appartenance

Le service d'appartenance (*membership*) vise à fournir à tout instant à chaque membre du groupe une vue de la composition courante du groupe.

- Partition primaire : la suite des vues fournies aux membres est totalement ordonnée. Il n'y a pas de partition, ou de manière équivalente, un unique primaire.
- Partition multiple : la suite des vues fournies aux membres est partiellement ordonnées. Il peut y avoir simultanément plusieurs vues disjointes.

Diffusion

Envoi d'un message à un ensemble de destinataires

Diffusion générale (*broadcast*)

Les destinataires sont tous les processus d'un ensemble défini implicitement (p.e. tous les processus du système). L'émetteur est généralement aussi destinataire.

`broadcast(émetteur, message)`

Diffusion de groupe (*multicast*)

Les destinataires sont un ou plusieurs groupes spécifiés. L'émetteur peut ne pas appartenir aux groupes destinataires. Les groupes peuvent ne pas être disjoints.

`multicast(émetteur, message, {groupe1, groupe2...})`

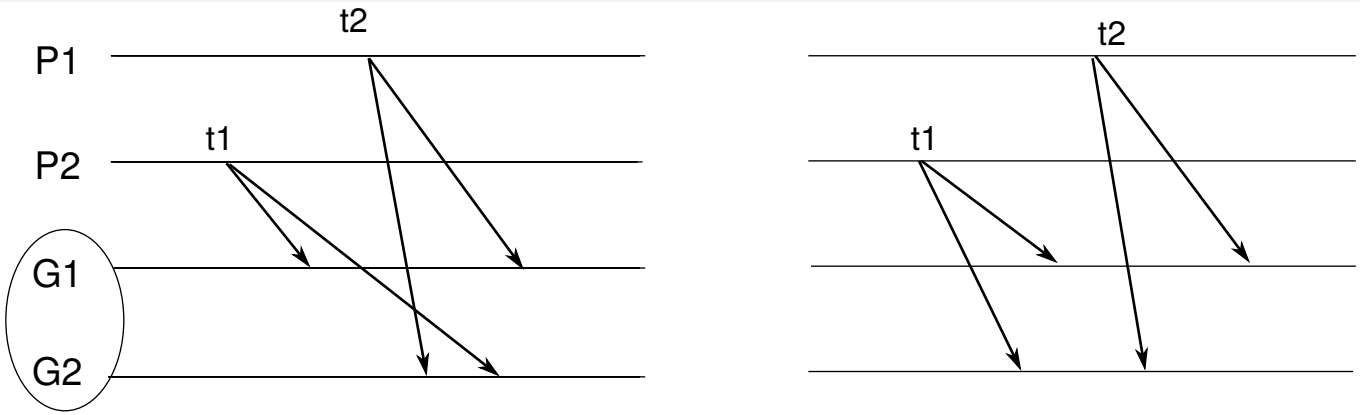
Propriétés indépendantes de l'ordre d'émission

Diffusion fiable

Un message est délivré à tous les destinataires corrects ou à aucun.

Diffusion totalement ordonnée (dite atomique)

La diffusion est fiable et les messages sont délivrés dans le même ordre sur tous leurs destinataires.



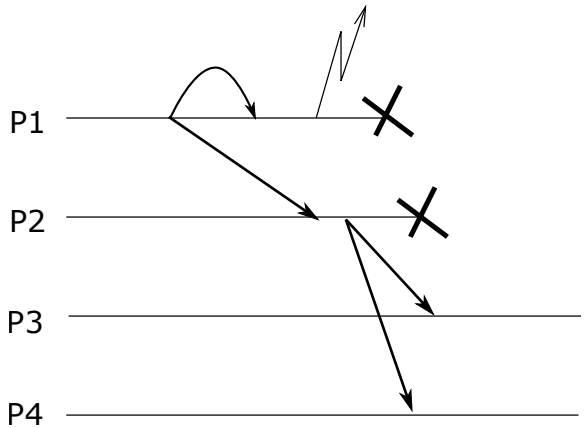
L'ordre total peut (ou non) être compatible avec la causalité

Uniformité

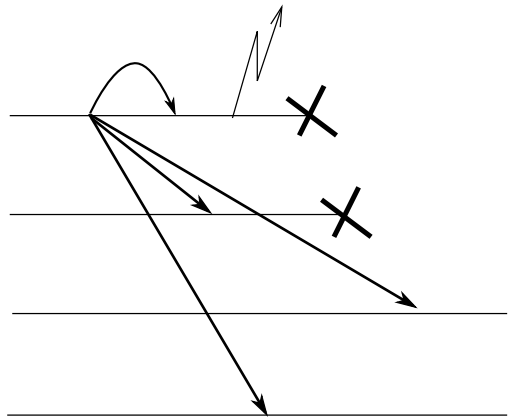
Diffusion fiable uniforme

Si un message est délivré à un processus (correct ou défaillant), il sera délivré à tous les processus corrects

$\neq$ avec diffusion fiable : délivrance à tous les processus corrects ou aucun, on ne se préoccupe pas des processus défaillants.



Non uniforme (mais fiable)

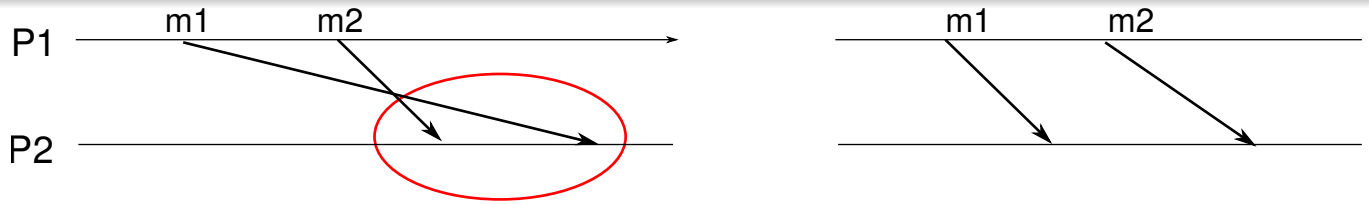

Uniforme

Nécessaire si un processus défaillant peut (avant sa défaillance) exécuter des actions irréversibles, notamment vis-à-vis de l'extérieur.

Propriétés liées à l'ordre d'émission

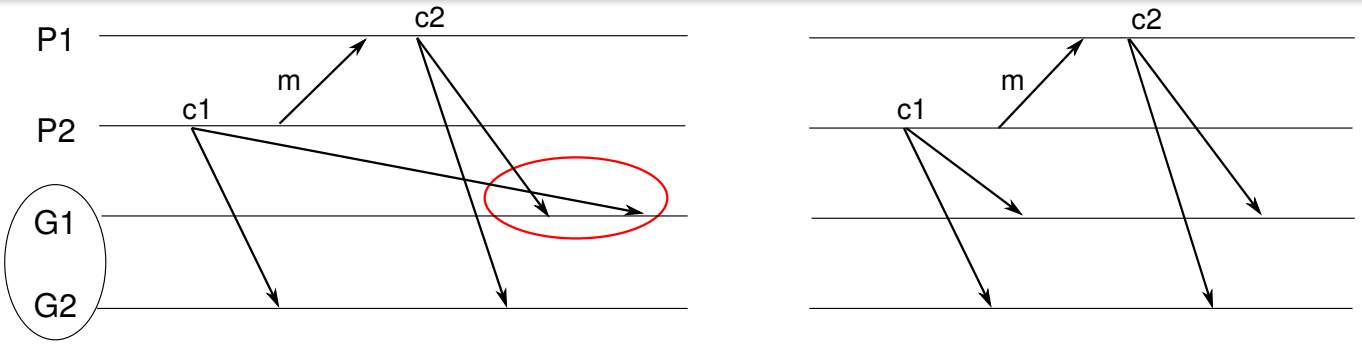
Diffusion FIFO

Deux messages d'un même émetteur sont délivrés dans l'ordre de leur émission.

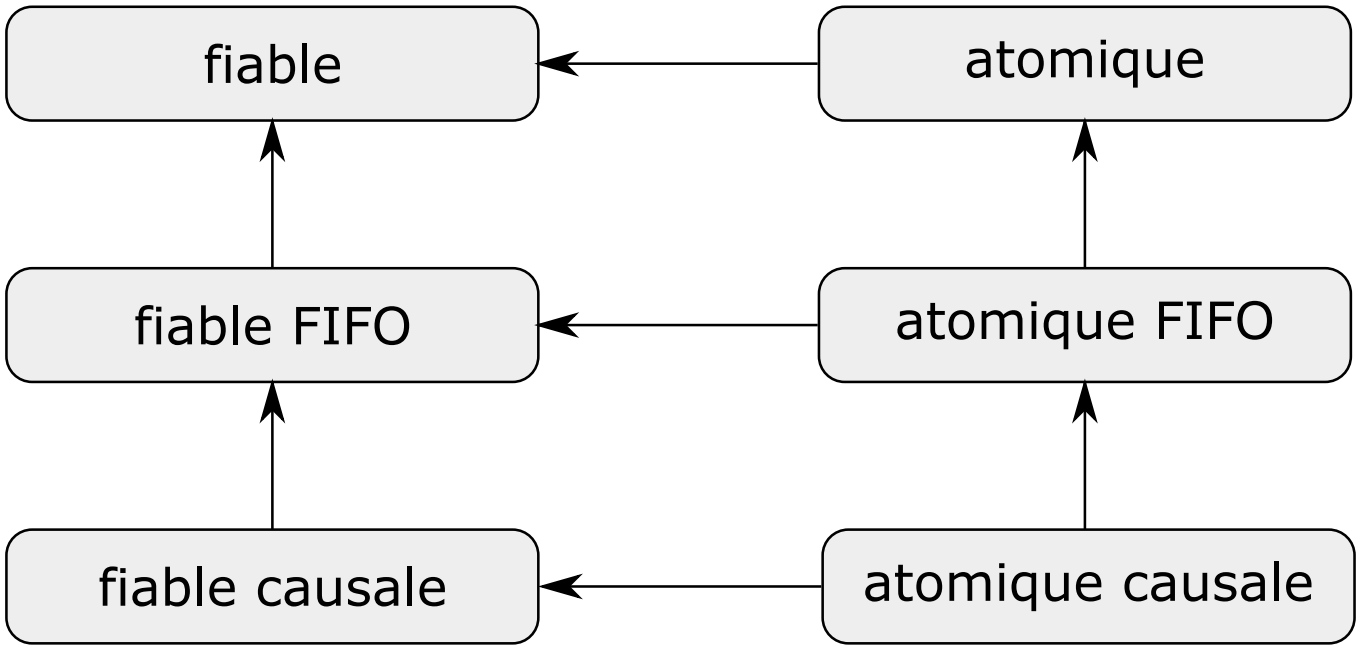


Diffusion causale

L'ordre de délivrance respecte la causalité de l'ordre (causal) d'émission.

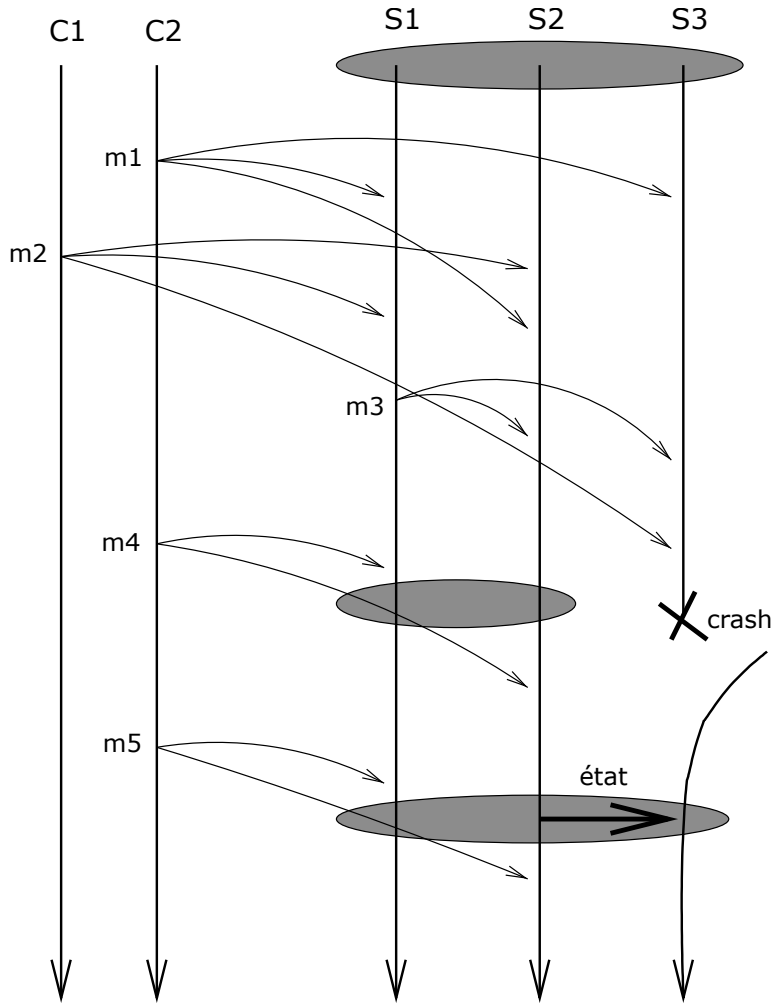


Classes de propriétés

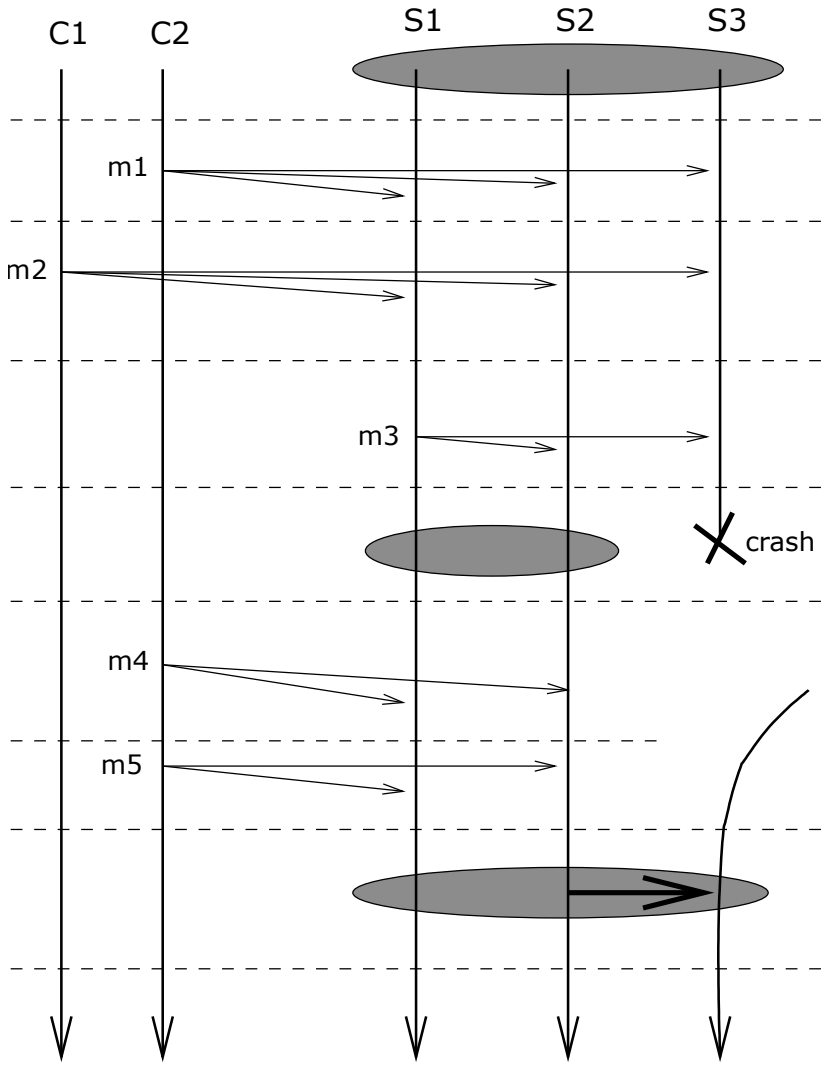


Diffusion anarchique

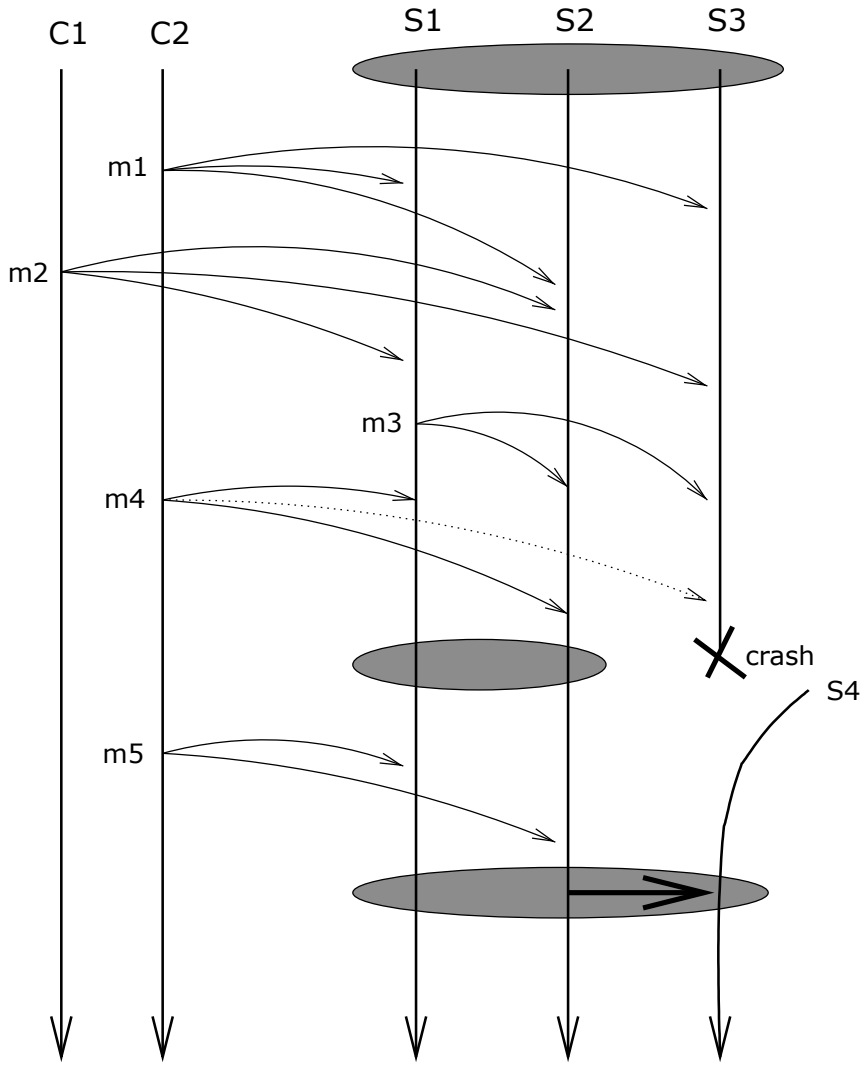
- Ordre causal ?
- Ordre total ?
- Atomicité ?
- Arrêt/Reprise ?



groupe avec synchronisme fort



groupe avec synchronisme virtuel



Le synchronisme virtuel

Une vue = état d'un groupe (ensemble des membres)

- Garantir qu'une diffusion est délivrée dans la vue où elle a été émise
- Garantir que, entre deux changements de vue, tous les membres reçoivent les mêmes messages
- Garantir que tous les membres voient la même séquence de changements de vue
- Progression limitée à la partition majoritaire

Impossibilités

Impossibilité du service d'appartenance

Un service d'appartenance minimal (spécifications peu contraignantes) est impossible dans un système asynchrone avec une seule défaillance d'arrêt.

Résultat similaire à Fischer, Lynch, Paterson (cf chapitre « consensus »). Le consensus est réductible à l'appartenance.

Impossibilité de la diffusion atomique

La diffusion atomique (= totalement ordonnée) est impossible dans un système asynchrone avec une seule défaillance d'arrêt.

Solutions (cf partie « consensus ») : supposer du synchronisme et/ou les détecteurs de fautes

Plan

RAID – *Redundant Array of Inexpensive/Independent Disks*

- Plusieurs disques sur lesquels on partitionne et duplique les données
- $\Rightarrow$ meilleure performance (accès parallèle)
- $\Rightarrow$ meilleure disponibilité (redondance)
- Multiples configurations selon besoins

(non développé : contrôle centralisé)

Techniques logicielles

Cf cours données réparties (réplication)