ÉCOLE POLYTECHNIQUE MONTRÉAL

Module 8:

Réplication

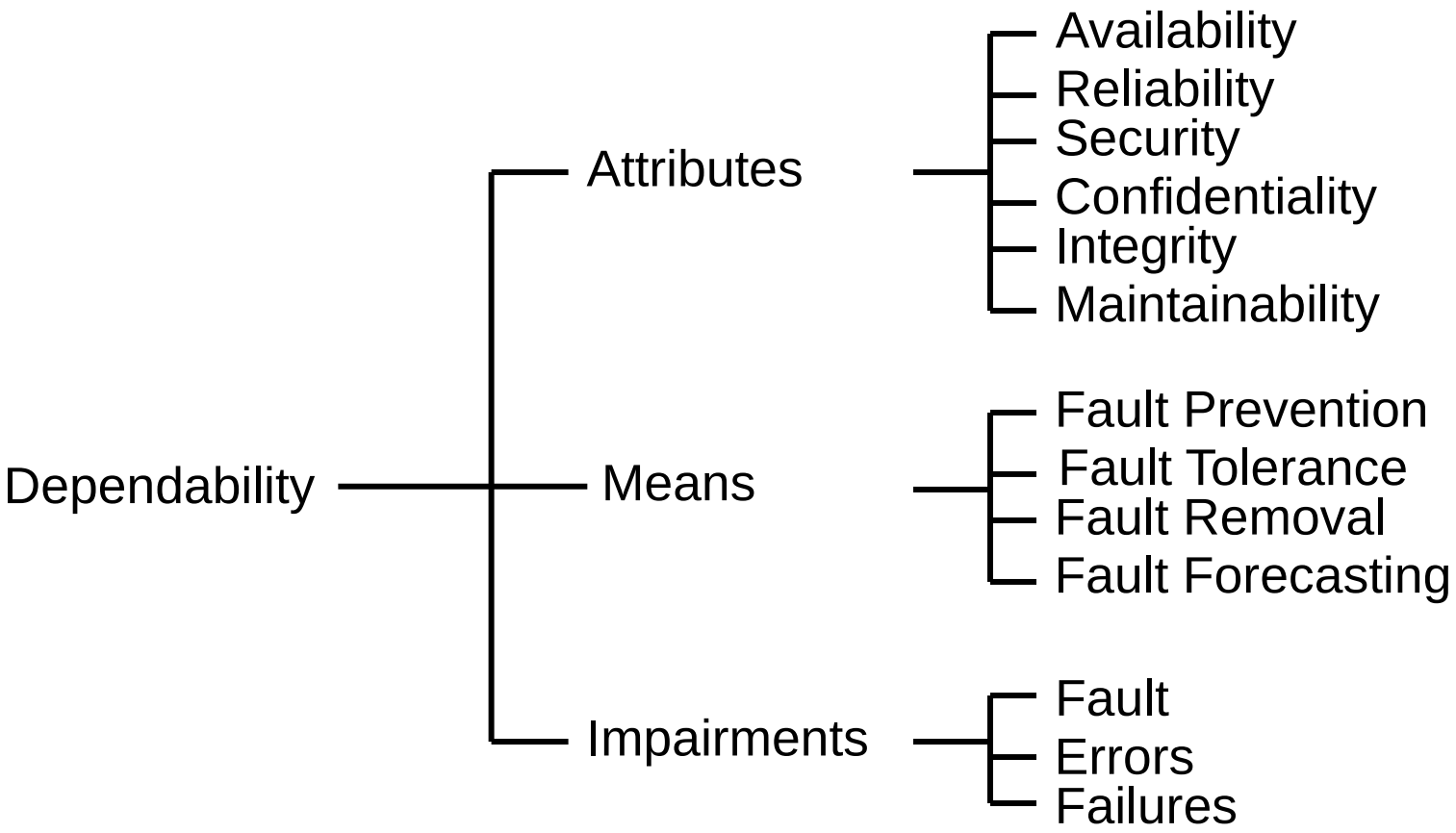
Introduction

- Augmentation de la performance (e.g. plusieurs serveurs Web qui se partagent la tâche).
- Haute disponibilité (serveur de rechange à jour qui prend le relais très rapidement).
- Tolérance aux pannes (disques RAID).

Systèmes fiables

- Disponibilité: être prêt à l'utilisation.
- Fiabilité: continuité de service.
- Sûreté: pas de conséquences catastrophiques pour l'environnement.
- Sécurité: prévention d'accès non autorisés et/ou de la manipulation de l'information (confidentialité, intégrité, disponibilité).
- Maintenabilité: réfère à la facilité avec laquelle un système défaillant peut être réparé.

Un peu de terminologie



Définitions

- **Accident**: événement (ou une série d'événements) non planifié qui peut entraîner des conséquences fâcheuses.
- **Fiabilité**: La mesure dans laquelle un système est en état de rendre son service (état opérationnel)
  - Probabilité qu'un système informatique fonctionne sans tomber en panne pendant un certain temps, dans des conditions d'utilisation bien définies
- **Sûreté:** la probabilité que les conditions qui peuvent causer des accidents ne surviennent pas

Définitions (suite)

- **Défaillance (panne):** lorsque le comportement d'un système viole sa spécification de service
  - Les défaillances résultent de problèmes inattendus internes au système qui se manifestent éventuellement dans le comportement externe du système.

- Ces problèmes sont appelés **erreurs** et leur causes mécaniques ou algorithmiques sont dites **fautes**

- Quatre sources de fautes peuvent causer la défaillance d'un système:
  - Spécification inadéquate.
  - Erreurs de conception dans le logiciel.
  - Défaillance matérielle.
  - Interférence sur le sous-système de communication

Classification des fautes

- **Transitoire:** se produit de manière isolée (une fois et disparaît); rayonnement alpha qui perturbe un bit de mémoire.
- **Intermittente:** se reproduit sporadiquement (transitoire et survient à plusieurs reprises); problème de bruit, de couplage entre signaux, de chaleur...
- **Permanente:** persiste indéfiniment (jusqu'à réparation) après son occurrence; disque brisé, circuit brûlé...

Classification des défaillances

- **Type de défaillances:**
  - **plantage:** serveur s'arrête, mais il fonctionnait correctement jusqu'alors.
  - **Panne d'omission:** serveur ne répond pas à une requête; bonne réponse ou pas de réponse (disque avec parité sur les blocs).
  - **Panne de temporisation:** la réponse survient en dehors de l'intervalle de temps réel spécifié; délai excessif (commande de centrale nucléaire, commande de vol, transactions à haute fréquence).
  - **Panne de réponse:** réponse est simplement incorrecte; capteur bruité, mémoire corrompue.
  - **Panne arbitraire (Byzantine):** réponses arbitraires voire malicieuses (testeur qui essaie de déjouer le système, attaque de sécurité).

Approches pour atteindre la Fiabilité

Prévention de fautes: comment prévenir l'occurrence ou l'introduction de fautes

Élimination de fautes: comment réduire la présence (nombre et sévérité) de fautes

Évitement de Fautes

Tolérance aux fautes: comment fournir un service en dépit des fautes

Prévision de fautes: comment estimer la présence, la création et les conséquences des fautes

Acceptation de Fautes

Prévention de Fautes par évitement

- Utilisation des composants les plus fiables en respectant les coûts et contraintes de performance.
- Choix des meilleures techniques d'assemblage et d'interconnexion des composants.
- Langages de programmation avec abstraction de donnée, modularité, sécurité.
- Environnement de développement et méthodologie structurés aidant à gérer la complexité et ne rien oublier.
- Examiner les formes d’interférences.
- Spécification rigoureuse des besoins, si pas formelle.

Elimination de Fautes

- La prévention ne peut éliminer la possibilité de fautes. Il faut détecter et éliminer les fautes. Prévoir un jeu de test le plus complet possible, dans des conditions de charge réalistes.
- Un test peut seulement être utilisé pour démontrer la présence de fautes, pas leur absence complète.
- Il est parfois impossible de tester sous des conditions réelles.
- La plupart des tests sont faits dans un mode de simulation, et il est difficile de garantir que la simulation est exacte.
- Les erreurs introduites durant la spécification des besoins peuvent ne pas se manifester jusqu'à ce que le système tombe en panne.

Tolérance aux fautes

- **Récupération de faute**: si un délai de récupération est acceptable.
- Il suffit de pouvoir reprendre où on avait laissé, après avoir redémarré (et possiblement réparé) le système.
- Requiert un stockage permanent fiable pour ne perdre aucune information. Copies de sauvegarde, cliché à intervalle régulier...
- Faire attention de bien tout sauver avant d'accepter une transaction.

Tolérance aux fautes

- Masquer la présence de fautes en utilisant la **redondance**
  - **Redondance matérielle**: composants matériels ajoutés pour supporter la tolérance aux fautes à tous les niveaux (alimentation électrique, processeurs, disques, réseau...).
  - **Redondance logicielle** : inclut tous les programmes et instructions utilisés pour tolérer les fautes.
  - **Redondance temporelle**: temps extra pour exécuter les tâches (exécuter les instructions plusieurs fois) pour la tolérance aux fautes.

Tolérance aux fautes (suite)

- **Niveaux de tolérance aux fautes :**
  - **Tolérance aux fautes complète:** le système continue à fonctionner en présence de fautes, sans perte significative de fonctionnalité ou de performance.
  - **Dégradation graduelle:** le système continue à fonctionner en présence de fautes, acceptant une dégradation partielle des fonctionnalités ou de performance durant le recouvrement ou la réparation.
  - **Arrêt sécuritaire:** le système maintient son intégrité tout en acceptant un arrêt temporaire de son fonctionnement.

- Niveau de tolérance aux fautes nécessaire dépend de l'application.
- La plupart des systèmes critiques nécessitent une tolérance complète, mais plusieurs se contentent d'une dégradation graduelle.

Phases de la tolérance aux fautes

- **Détection d'erreurs:** la présence des fautes est déduite en détectant une erreur.

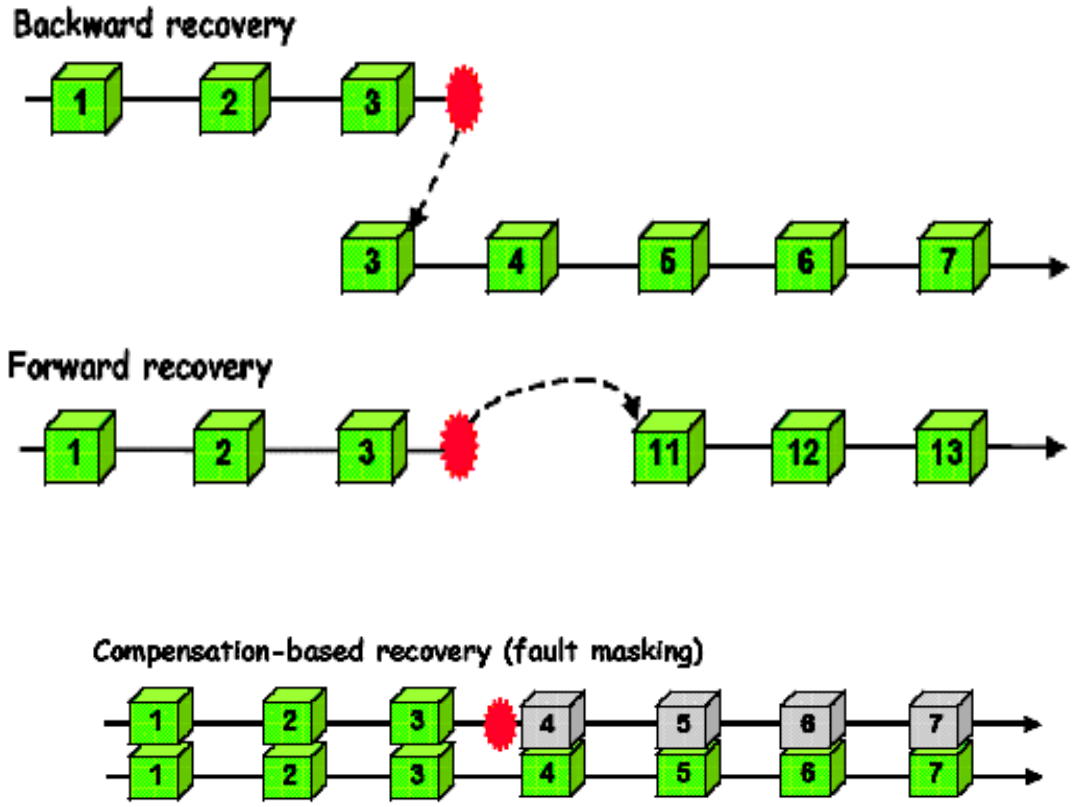
- **Recouvrement d'erreurs:** élimine les erreurs de telle façon qu'elles ne se propagent pas par des actions futures.

Phases de la tolérance aux fautes (suite)

- **Recouvrement d'erreurs en aval:** continuer à partir de l'état erroné en faisant des corrections sélectives à l'état du système. (Remplacer un disque défectueux dans une unité RAID)

- **Recouvrement d'erreurs en amont:** consiste à restaurer le système à un état précédent sûr et en exécutant une section alternative du programme. (Remplacer un disque défectueux et l'initialiser avec une copie de sauvegarde)
  - **Point de recouvrement:** point auquel un processus est restauré
  - **Point de contrôle (checkpoint):** établissement d'un point de recouvrement (en sauvegardant l'état approprié du système)

Méthodes de recouvrement d'erreurs



Phases de la tolérance aux fautes (suite)

- Traitement de fautes et continuité de service :
  - Le recouvrement d'erreur retourne le système à un état exempt d'erreur; l'erreur peut survenir à nouveau. La phase finale de la tolérance aux fautes est d'éradiquer la faute du système.
  - Une erreur est un symptôme de faute; corriger l'erreur ne répare pas la faute.
  - Le composant défaillant (ou la panne) doit être identifié.
  - Le traitement automatique des pannes est difficile et spécifique au système.
  - Localisation de la panne et réparation du système
  - Les techniques de détection d'erreur peuvent aider à retracer la panne
    - Faute matérielle : le composant peut être remplacé
    - Faute logicielle : peut être éliminée dans une nouvelle version du code
    - Dans les applications qui ne doivent pas s'arrêter il est nécessaire de modifier le programme alors qu'il s'exécute!

Masquage hiérarchique des fautes

- Un niveau détecte, reprend et masque les erreurs au niveau plus bas:
  - Le client demande l'adresse IP d'un ordinateur.
  - Le serveur de nom fait une requête récursive à d'autres serveurs et prend les serveurs alternatifs lorsqu'un serveur ne répond pas.
  - Un serveur interrogé découvre une erreur sur un disque et lit plutôt de son second disque en miroir.
  - Le client a la réponse et ne sait rien des défaillances.

Masquage par groupe des fautes

- Le client demande à tous les serveurs, prend le premier qui répond. Peut tolérer (n-1)/n pannes par omission.

- Le client demande à tous les serveurs et prend un vote sur les réponses. Peut tolérer (n-2)/n pannes si les chances d'avoir deux mauvaises réponses identiques sont presque nulles, ou n/(2n+1) pannes même si tous les ordinateurs défaillants tentent vicieusement de s'entendre sur une mauvaise réponse.

- **Groupe synchronisé:** tous les serveurs doivent recevoir les mêmes mises à jour dans le même ordre.

- **Groupe non synchronisé:** les serveurs de secours enregistrent les mises à jour dans un journal mais sans les traiter. Au besoin, ils utilisent ce journal pour se mettre à jour et prendre le relais avec un certain délai.

Tolérance aux fautes matérielle

- Deux types : redondance statique et dynamique
  - **Statique :** composants redondants utilisés pour cacher les effets des fautes
    - **Exemple :** Triple Modular Redundancy (TMR): 3 composants identiques et un circuit de vote majoritaire; les résultats sont comparés et si un diffère des deux autres il sera masqué.
    - Assume que la faute n'est pas commune (telle qu'une erreur de conception) mais elle peut être transitoire ou causée par la détérioration du composant.
    - Pour masquer les fautes de plus d'un composant il faut NMR (N-tuple Modular Redundancy).
  - **Dynamique :** redondance utilisée à l'intérieur d'un composant indiquant si le résultat est erroné.
    - Fournit une technique de détection d'erreurs; le recouvrement doit être fait par un autre composant

Tolérance aux fautes logicielle

- Utilisée pour détecter les erreurs de conception
- **Statique :** programmation N-Versions
- **Dynamique :**
  - Détection et recouvrement
  - Blocs de recouvrement : recouvrement d'erreurs en amont
  - Exceptions : recouvrement d'erreurs en aval

- **Réplication de données**
  - Maintien de copies d’un objet sur plusieurs systèmes.
  - Peut améliorer les performances du système (2x plus de bande passante de lecture pour deux disques en miroir, même performance qu'un seul disque en écriture).
  - Augmente la disponibilité en masquant les fautes et donc minimisant l’impact des pannes.

Effets positifs de la réplication

- **Performance:**
  - Mise en cache des données accédées fréquemment ou la réplication des données près du point d'accès pour réduire le temps d'accès.
  - Répartition des données sur plusieurs serveurs pour répartir la charge de travail.
  - La réplication d'objets statiques est triviale tandis que la réplication d'objets dynamique requiert des mécanismes de contrôle de la concurrence et de maintient de la cohérence (ajoute de la surcharge de traitement et de communication).

- **Disponibilité :**
  - Offrir une disponibilité du service le plus près possible de 100%
  - 2 types de pannes:
    - Serveur: réplication de données sur des serveurs qui tombent en pannes indépendamment des autres.
    - Partition du réseau ou opérations déconnectées: les utilisateurs dans leurs déplacements se connectent/déconnectent aléatoirement au réseau.

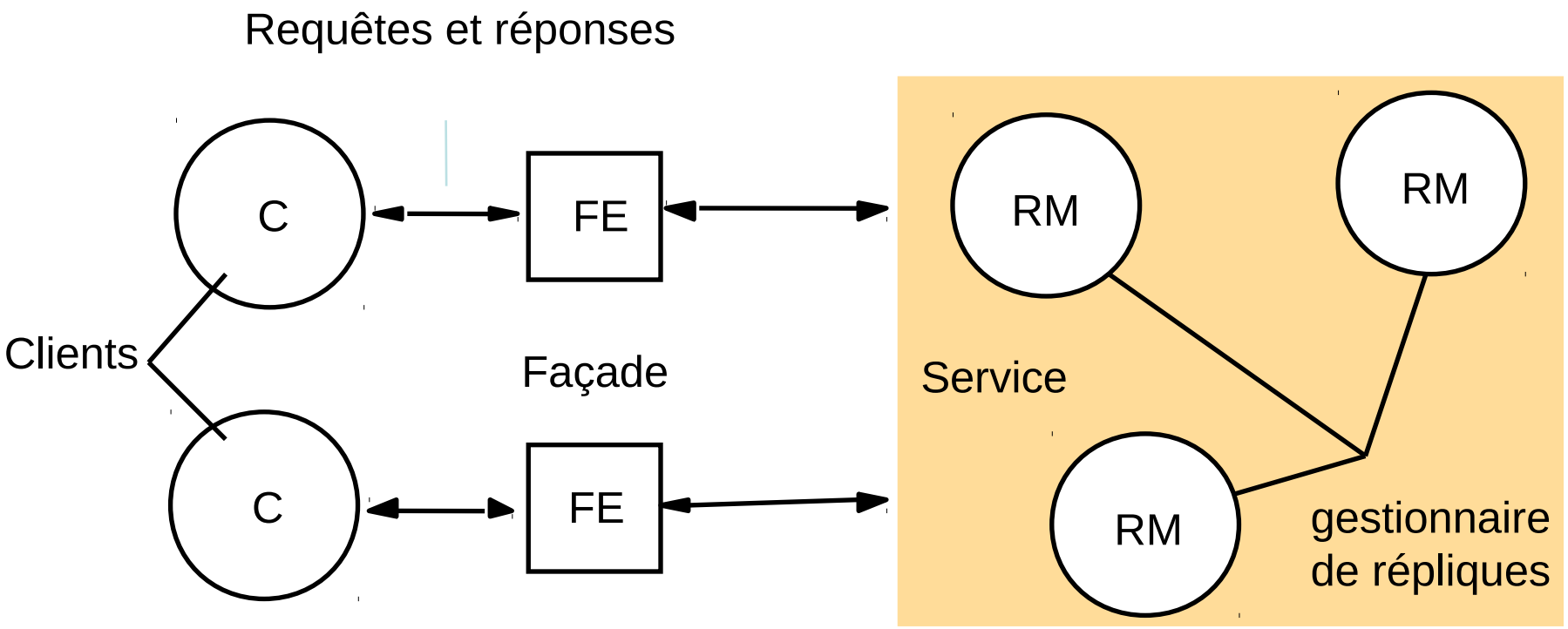
Requis généraux pour la réplication de données

- **Transparence de la réplication:**
  - Les utilisateurs voient des objets logiques, ils n'ont aucune connaissance des multiples copies physiques qui existent
  - Ils accèdent une instance de l'objet, peu importe laquelle, et reçoivent un résultat unique

- **Cohérence:**
  - Étant donné plusieurs copies physiques de l'objet et des accès concurrents, les copies doivent être maintenues dans un état cohérent entre elles

- **Modèle du système**
  - Les données dans le système sont considérées comme un ensemble d'objets logiques.
  - Chaque objet logique est implémenté par un ensemble d'objets physiques appelés répliques.
  - La cohérence de chacune des répliques dépend des opérations appliquées localement par le système sur lequel elles résident.
  - Les répliques peuvent être incohérentes entre elles à un point donné dans le temps.
  - Le système en soi est asynchrone et peut tomber en panne.

- **Modèle architectural pour la réplication**



- **Modèle architectural de base pour la réplication**
  - Le système se compose de trois éléments :
    - Les clients effectuent les requêtes sur des objets répliqués à travers les FE.
    - Les FE agissent comme intermédiaire pour les clients et s'occupent de contacter les RMs. Ils assurent la transparence pour les utilisateurs
    - Les RMs maintiennent les répliques d'objets qui sont sous sa responsabilité
      - Les objets peuvent se trouver répliqués sur tout ou sur un sous-ensemble des RMs
      - Les RMs offrent un service d'accès aux objets répliqués au client (accès en lecture ou en mise à jour)

Les étapes de base pour l'accès aux objets répliqués

1) Requête

- Le client effectue une requête sur un objet répliqué
- La requête est prise en charge par le FE qui s'occupent de la relayer à un RM ou a plusieurs RM par diffusion sélective

2) Coordination

- Les RMs décident d'accepter ou non la requête, lequel d'entre les RMs va desservir la requête et dans quel ordre par rapport aux autres requêtes (FIFO, causal, total)

3) Exécution

- Le ou les RMs concernés réalisent la requête (peut-être partiellement pour permettre le recouvrement de l'ancien état de l'objet)

4) Accord

- Les RMs se mettent d'accord sur les effets de la requête sur les différentes instances de l'objet répliqué (approche immédiate ou paresseuse)

5) Réponse

- Un ou plusieurs RMs renvoient la réponse au FE ayant effectué la requête, qui lui-même la donne au client concerné

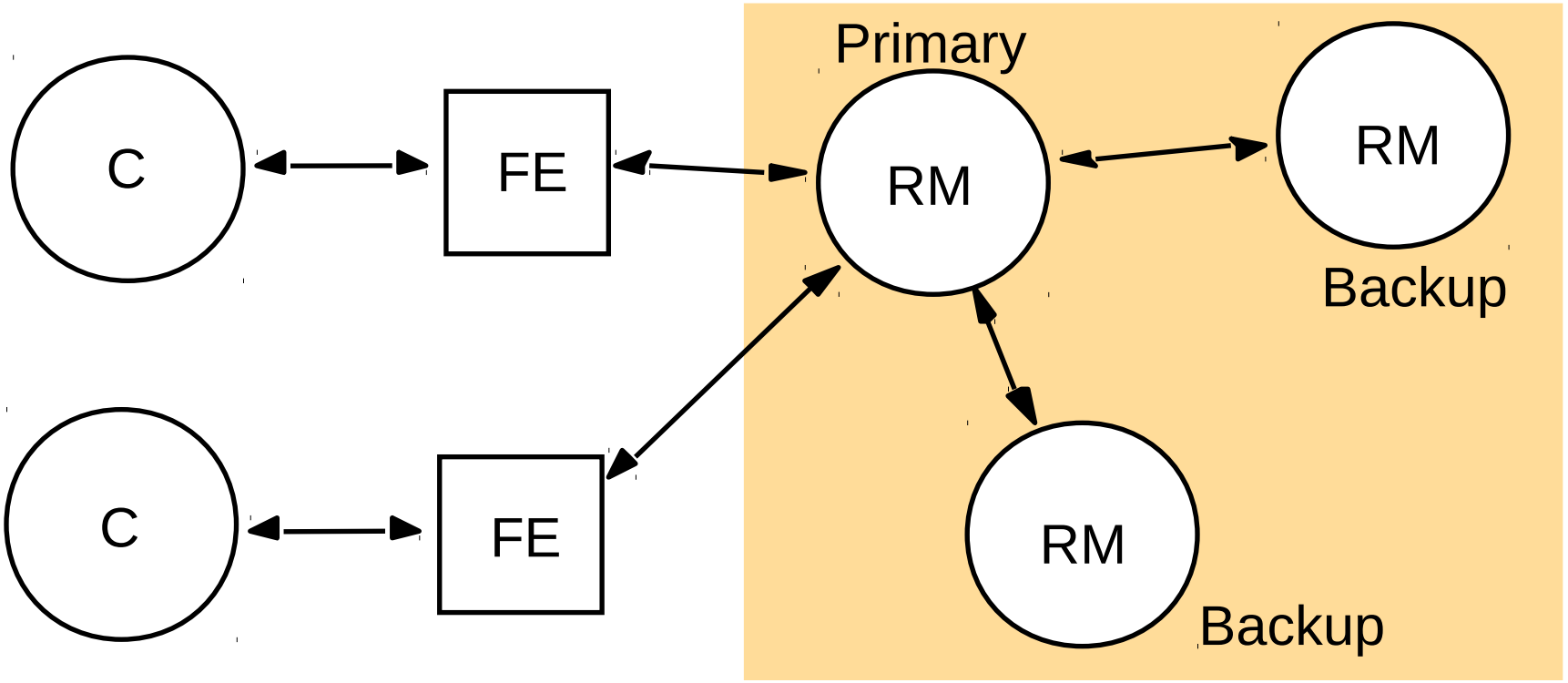
Service tolérant aux fautes

- **Un service a un comportement dit correct s'il fonctionne selon les spécifications même en présence de pannes**
  - Les clients ne peuvent voir la différence entre un service obtenu à partir de données répliquées et celui obtenu à partir d'une copie unique
  - On doit s'assurer que l'ensemble des répliques produit le même comportement qu'une seule copie

Modèle de réplication passive

- Dans ce modèle, à tout moment il y a un RM actif (primaire) qui répond aux requêtes des clients et des RMs esclaves (secondaires) qui servent de RMs de secours en cas de panne.
- Le résultat d'une opération effectuée sur le RM primaire est communiqué à tous les esclaves (ce qui permet de maintenir la cohérence des données).
- Quand le RM primaire tombe en panne, un RM esclave prend la relève (après une élection, par exemple).
- Le système est linéarisable puisque la séquence des opérations est programmée et contrôlée par le RM primaire.
- Si le RM primaire tombe en panne, le système demeure linéarisable puisqu'un des RMs secondaires reprend exactement là où le RM primaire est tombé en panne.

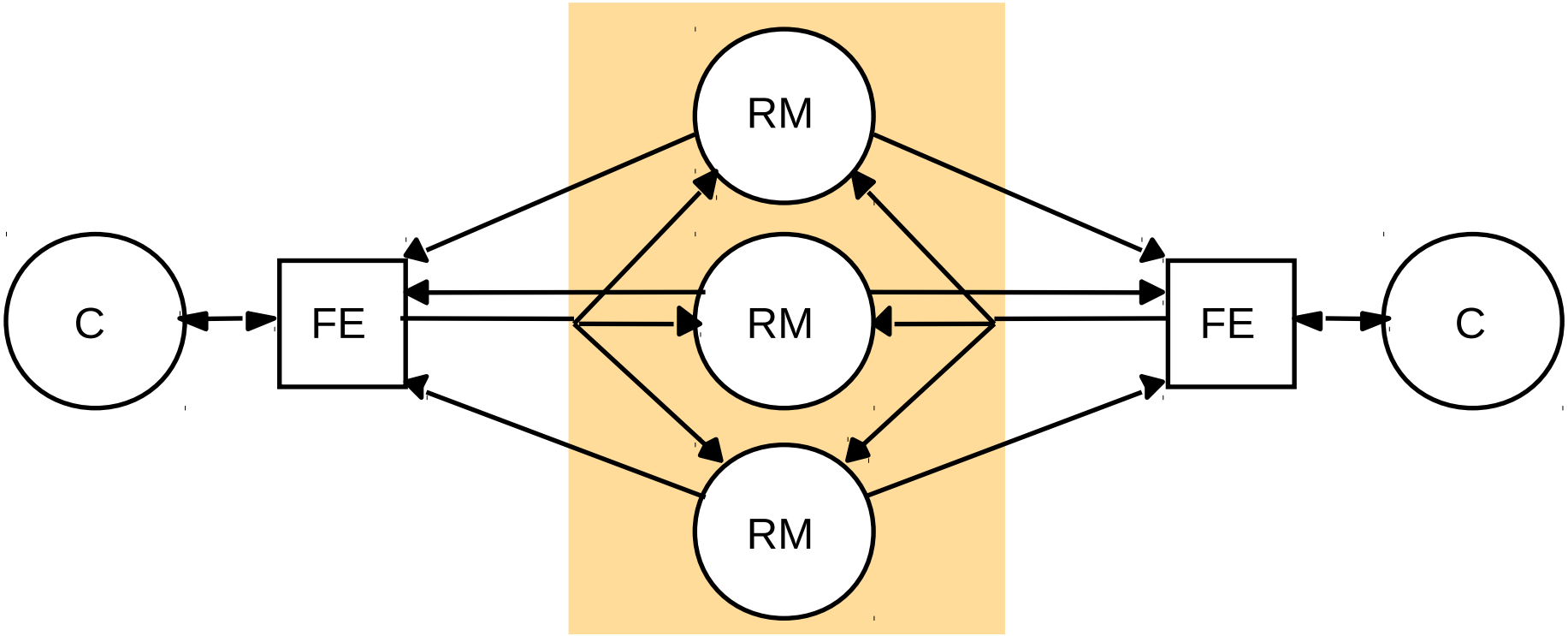
- **Modèle de réplication passive**



- **Exemples de réplication passive**
  - Serveur NIS: un serveur maître qui propage les modifications à plusieurs serveurs esclaves.
  - Linux virtual server: un serveur primaire a deux adresses IP, une pour la gestion et une pour le service. Un serveur de rechange reçoit les mises à jour et interroge souvent le serveur primaire. Si le serveur primaire ne répond plus, il le déconnecte, prend son adresse IP de service et commence à s'annoncer sous cette adresse et à servir les requêtes.
  - Deux serveurs DHCP peuvent coexister s'ils offrent des blocs d'adresses différents. Le second serveur répond s'il voit que le premier ne répond pas aux requêtes à tous pour une adresse dynamique.

- **Modèle de réplication active**
  - Les RMs sont des machines à état qui jouent exactement le même rôle et sont organisés en groupe.
  - Les RMs débutent dans le même état, et changent d'état concurremment, de telle manière que leurs états restent identiques.
  - Si un RM tombe en panne, cela n'a aucun effet sur les performances du système parce que les autres RMs peuvent prendre la relève de manière transparente.
  - Ce modèle requiert un mécanisme de communication à diffusion sélective (*multicast*) fiable et totalement ordonné (pour que les RMs effectuent les mêmes opérations dans le même ordre).
  - Supporte la cohérence séquentielle mais pas la linéarisation (car l'ordre total ne correspond pas nécessairement à l'ordre d'exécution temporel des opérations).

- **Modèle de réplication active**



- **Exemple de réplication active**
  - Serveurs NFS avec automount: la requête est envoyée à tous les serveurs et le premier à répondre est pris.
  - CODA: plusieurs serveurs offrent les mêmes fichiers et se propagent les modifications. Très flexible mais ne peut garantir l'absence de conflits (mises à jour concurrentes).

- **Transactions réparties avec réplication**
  - Le protocole à deux phases pour compléter une transaction doit s'étendre aux données répliquées qui doivent elles aussi changer de manière atomique avec le reste.
  - Le coordonnateur de transaction parle au gestionnaire de copies qui agit à son niveau comme coordonnateur. Les messages *prêt*, et dans une seconde phase *compléter* ou *annuler*, sont relayés à toutes les copies avant de produire une réponse au coordonnateur. C'est un peu comme si le coordonnateur de transaction devait contacter toutes les copies.
  - Lecture sur un, écriture sur tous: verrou en lecture sur un, ou verrou en écriture sur tous (bloqué si un verrou en lecture existe).
  - Copie primaire: tous les verrous sont pris sur la copie primaire, au moment d'accepter la transaction, toutes les mises à jour sont propagées aux copies.
  - Il faut faire attention si les copies disponibles pendant la transaction changent, que le tout demeure cohérent.
  - Lorsque le réseau est partitionné, on peut continuer dans chaque partition et résoudre les conflits plus tard (optimiste), ou ne continuer que dans la partition qui a le quorum (si une partition a plus que la moitié et est donc la seule ainsi), ou en être réduit à des lectures seulement puisqu'il n'est plus possible d'obtenir des verrous d'écriture sur tous.

- **Calcul de disponibilité**
  - Probabilité qu'un composant soit fonctionnel au temps t.
  - Peut être calculé comme temps moyen avant panne (MTTF) et temps moyen de réparation (MTTR): MTTF / (MTTF + MTTR).
  - Probabilité p que deux événements indépendants, de probabilité p1 et p2, se produisent en même temps: p = p1 * p2.
  - Les probabilités de cas disjoints peuvent s'additionner.
  - Probabilité p que k disques sur n soient fonctionnels (et n - k en panne), si la propabilité pour un disque est pd.
  - Sommation de k à n pour k ou plus fonctionnels.
  - Probabilité qu'un ou l'autre disque soit fonctionnel dans un miroir, 1 - probabilité que les deux soient en panne.

$$p = \binom{n}{k} \times p_d^k \times (1 - p_d)^{n-k}$$

- **Probabilités : attention!**
  - Probabilité que deux événements indépendants de probabilité p1 et p2 se produisent au même moment: p1 x p2. Par exemple, un ordinateur est fonctionnel si la carte mère (p1) et le disque (p2) sont fonctionnels. Il y a une panne sauf si les deux sont fonctionnels.
  - 1 - p1 x p2
  - Peut-on plutôt prendre la probabilité de panne de carte + la probabilité de panne de disque?
  - (1 - p1) + (1 - p2) ≠ 1 - p1 x p2
  - Quelle est la différence? La panne de disque se produit indépendamment de la carte qui peut être fonctionnelle ou non. On compte donc deux fois le cas où le disque et la carte sont défectueux. Les deux éléments sommés se recoupent alors qu'il faut bien tout séparer. Si p1 et p2 sont grands, la partie sommée en double est petite et le résultat est incorrect mais très proche. Si p1 et p2 sont petits, le résultat dépasse largement 1!
  - (1 - p1) x p2 + (1 - p2) x p1 + (1 - p1) x (1 - p2) = (1 - p1) + (1 - p2) - (1 - p1) x (1 - p2) = p2 - p1 x p2 + p1 - p1 x p2 + 1 - p2 - p1 + p1 x p2 = 1 - p1 x p2