Tolérance aux Fautes des Systèmes Informatiques

Thomas ROBERT

Plan du cours

- Tolérance aux fautes sur les données
- Tolérance aux fautes «logicielle »
- Tolérance aux fautes matérielle

Rappel Moyen SdF

- Tolérance aux fautes =

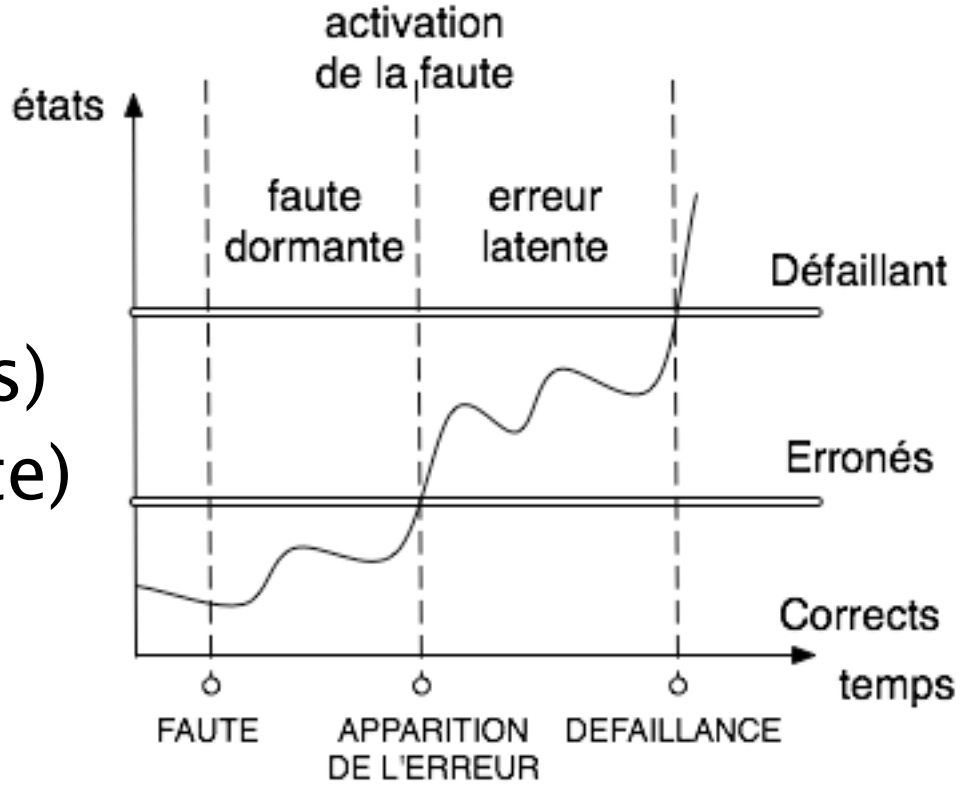
Prévenir l'occurrence des défaillances ou d'en limiter la sévérité (modes dégradés)

- Concepts clés:
  - processus de propagation,
  - modèles de fautes,
  - redondance,
  - diversification et isolation

La chaîne faute/erreur/ Défaillance

- Evénements :
  - Activation
  - Détection
  - Défaillance
- Etats :
  - Fautifs (non activés)
  - Corrects (sans faute)
  - Erronés (?)
  - Défaillants (KO)

Pas de détection == erreur dormante jusqu'à la défaillance …



Les défaillances et le besoin d'abstraction

- Une infinité de manières de défaillir
- Une classification pour comparer (typage))

Idée définir des attributs caractérisant de grandes familles

⇒4 attributs pour les définir

  - Domaine (nature de la défaillance)
  - Observabilité (conscience de la défaillance)
  - Homogénéité (perception de la défaillance)
  - Classification des Conséquences

- Approche alternative : établir une « taxonomie » basée sur l'expérience

La notion de modèle de faute

- Une faute == cause adjugée ou effective d'une erreur et donc d'une défaillance
- Quelques exemples :
  - **Corruption de la mémoire physique**
    -> les valeurs peuvent être modifiées aléatoirement
  - **Faute de développement**, pointeur mal initialisé
    -> donnée incorrecte / boucle infinies
  - **Défaillance du matériel ou sous-système**
    -> arrêt complet du système, comportement aléatoire

environnement

Production

Interaction

Modèles de fautes

- Une classification des fautes :
  - Phase de création ou occurrence
  - Positionnement (logicielle/matérielle)
  - Source (humaine / « naturelle »)

  (si source humaine)
    - Intention
    - « Compétence » (Délibéré/Accidentel/incompétence)
  - Persistance (permanente / transitoire)
- Les modèles permettent de clarifier les conditions de propagation et de traitement

=> Permet de **déterminer la stratégie adaptée**

Le principe de la TaF

- Empêcher les défaillances :
  1. Éviter l'activation des fautes (élimination)
  2. Éviter qu'une erreur n'entraîne une défaillance (tolérance aux fautes)

- 1) = Traitement des fautes,
  2) =Traitement des erreurs
- Comment faire 2) ?
  - Détection & Rétablissement
  - Masquage (ou compensation)
- Cela va avoir coût

Principes fondamentaux

- Zone de confinement => avoir des composants aux défaillances maitrisées
- Mise en œuvre :
  - Connaître l'état du système
  - Savoir traiter / maitriser un état erroné
  - Comprendre l'origine des erreurs pour la maintenance

Zone de confinement des erreurs

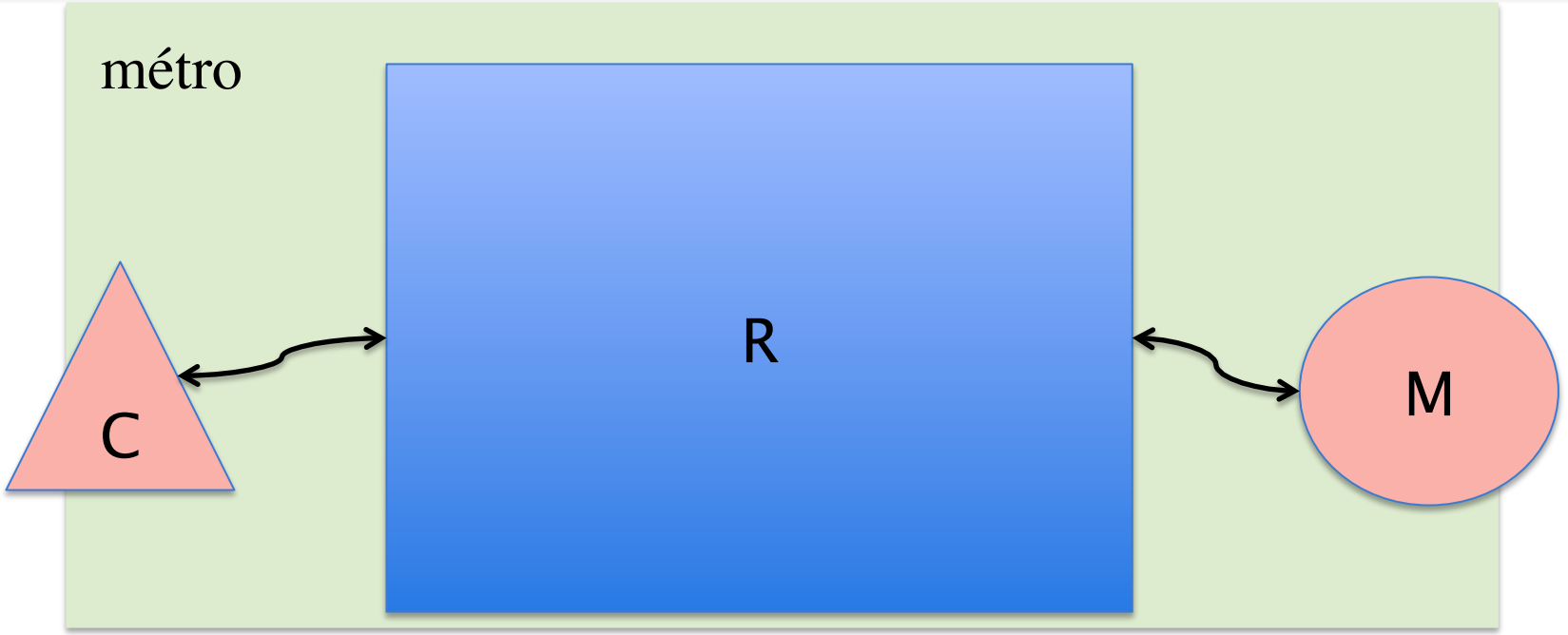
- **Définition**
  périmètre/interface d'un système muni de mécanismes de protection empêchant une erreur de se propager sans être au moins détecté et signalée
  (ie de contaminer l'environnement du système)
- En pratique :
  - Description architecturale définissant la décomposition du système en sous-systèmes dépendant les uns des autres
  - Description des défaillances possibles associées au système et chaque sous-système.
  - modèle de fautes (apparition des erreurs) et de leur propagation
- Mise en œuvre :contrôle aux interfaces, et modification architecturales / comportementales internes

Structure hiérarchique et systèmes de systèmes

- La structure d'un système :
  - Hiérarchie
  - Dépendances matériel / logiciel
  - Description des interactions aux interfaces
- Principe de propagation :

  La défaillance d'une partie d'un système peut devenir une faute pour le reste du système
- La définition d'une architecture aide à raisonner (un des intérêts des ADL)

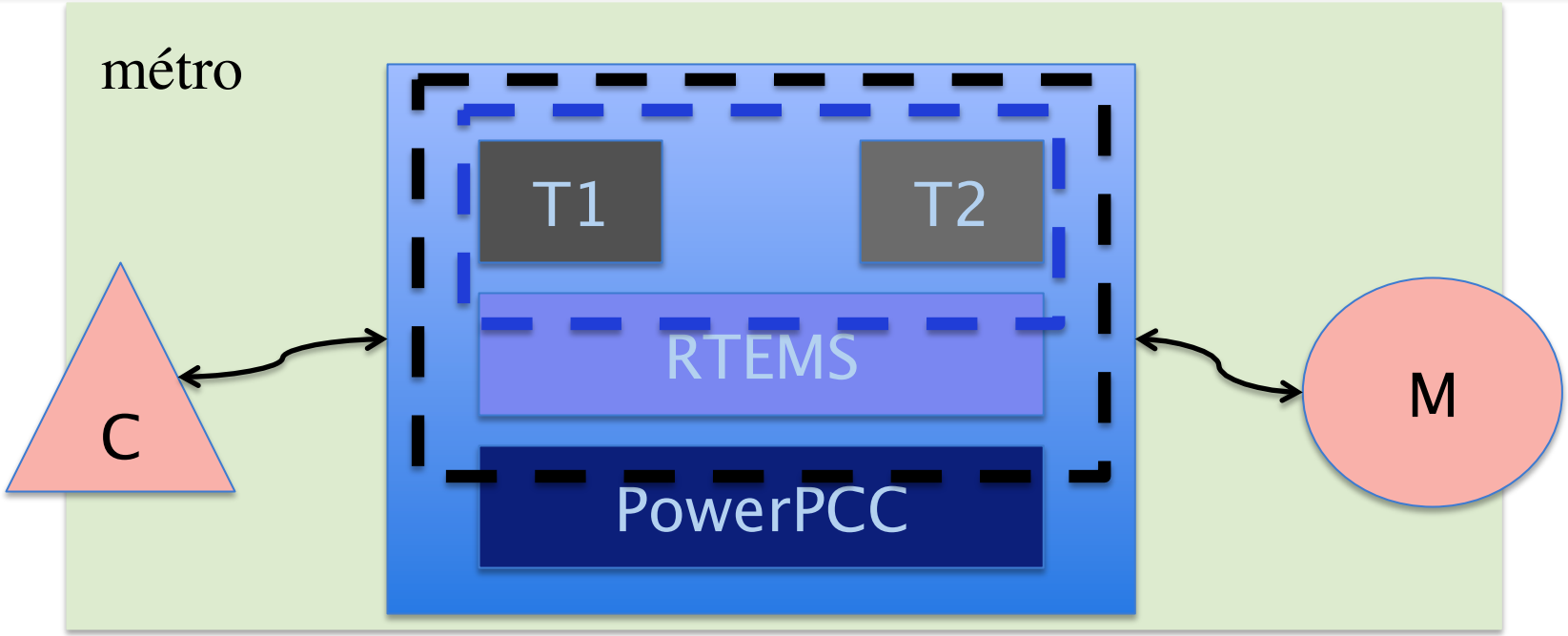
Défaillances, fautes et propagation des erreurs



- Fautes : interactions (propagées à travers l'interface)
- Propagation des erreurs : capteur (C) -> régulateur (R) -> moteur (M)
- Erreur dans C -> défaillance de C -> faute pour R -> erreur dans R

On peut « entrer » dans R…

Propagations Logiciel/Matériel



•Décomposition du boitier en éléments relativement indépendants

•La défaillance d'une tâche peut altérer le fonctionnement de RTEMS déclenchant une altération du matériel (effacement du bios

⇒Propagation interne au boitier du logiciel vers le matériel

C'est très compliquer, il faut raisonner par type de fautes, pas à pas…

Traitement des Fautes

- **Détection** : déterminer la cause première d'une défaillance.
- **Isolation** : relier la plus petite partie du système à la faute cause de la défaillance
  Détermination d'une zone de confinement
- **Reconfiguration** :
  altération de la structure et de la logique du système pour inhiber la faute

Traitement des Erreurs

- **Détection**
  - Test de vraisemblance : erreur si Observé ≠ Attendu
  - Exécution multiple + Comparaison, erreur si V1≠V2
- **Recouvrement :** Démarche de correction de l'erreur par
  - redéfinition de l'état ou compensation
  - Compensation de l'état erroné (cf redondance)
- **Utilisation de la détection**
  - Signalement/journalisation (exception Java)
  - Synchronisation d'action de Recouvrement

Tolérance aux fautes sur les données

Principe de détection

- Codage (binaire) : représentation sous forme de séquence de bits d'une information
- Soit V un ensemble de valeurs distinctes de taille 2^k => k bits au minimum pour son codage
- Modèle de faute : altération du support de stockage => défaillance = altération de la représentation stockée

Codage pour la détection

- Principe : Soit un ensemble de valeur V

de taille 2^k

Représentation sur r > k bits

Les représentations de V sont un sous ensemble de $\{0,1\}^r$

- Propriété souhaitée n°1 :

Toute modification d'au plus u bits d'une représentation d'un élément de V ne représente aucun élément de V

Codage pour la détection

- Principe : Soit un ensemble de valeurs V

de taille 2^k

Représentation sur r > k bits

Les représentations de V sont un sous ensemble de $\{0,1\}^r$

- Propriété souhaitée n°2 :

Toute modification d'au plus u bits d'une représentation d'un élément de V ne peut correspondre qu'à au plus un élément de V.

Vocabulaire

- Code correcteur d'erreur :

Principe de représentation de l'information satisfaisant les deux propriétés

- Distance de Hamming : opérateur de comparaison entre vecteurs binaires de même taille : d(v1,v2) décompte le nombre de bits distincts 2 à 2

Alternatives
prise en compte de la localité

- Idée : les bits altérés sont peut être groupés
- Codage de x dans V : x1, …. , xn
- Créer K vecteur de taille l ≤ n tels que V_i contient un sous ensemble des coordonnées de x
- Propriété : il suffit de disposer de suffisamment de copie intègres pour retrouver x.

TaF logicielle

Génie logicielle et TaF

- Les activités:
  - Identification / classification des défaillances
  - Conception / mise en œuvre des zones de confinement
  - Vérification des comportements/performances
- Les moyens
  - Des modèles et des méthodes
  - Des designs patterns
  - Des modèles de performances (chaînes de markov)

TaF logicielle ≠ logiciel pour la TaF

- TaF logicielle => zone de confinement au niveau LOGICIEL (ce doit être justifié)
- **Pré-requis** :
  connaître les modes de défaillance du logiciel et leur propagation au matériel
- **Exemple au tableau sur une application Java hébergée sous Unix**
  - **OutOfBoundException**
  - **NullPointerException** …

Confinement logiciel via le support d'exécution

- Défaillances concernées :
  tout ce qui ne compromet pas le matériel mais altère l'exécution
  - Valeurs incorrecte sur les interfaces des fonctions
  - Comportement aberrant de l'ordonnancement
  - Corruption de l'intégrités des donnée de programme
- Comportement aberrant: process Unix & sigsev
  - 1 process unix
  - 1 accès illicite à la mémoire du noyau (pointeur null)
  - Détection par la MMU (hw), et signalement au processus
  - Le processus se termine en indiquant au système d'exploitation (env du process) la cause de l'arrêt (SIGSEV)

Confinement par les API (1/2)

La surcharge des valeurs de retour :

- Pré-requis: taxonomie des erreurs
- Composants : fonctions
- Mise en œuvre : encodage de l'état fonctionnel du composant dans la valeur de retour des fonctions
- Remarque : le type de retour doit posséder des valeurs « libres »
- Exemple : code retour en C

Confinement via les API (2/2)

- Les exceptions :
  - Pré-requis :
    - encapsulation des traitements séquentiel dans des « blocs » (fonctions, accolades),
    - arrêt et déroutement de l'exécution d'un bloc sur réception d'un événement
    - taxonomie des erreurs + support pour le déroutement d'exécution
  - Composants : méthodes, fonctions, boucles
  - Mise en œuvre :
    - Utilisation du typage : 1 type d'exception=1 classe d'erreur
    - Définition d'un politique de propagation des signalements en l'absence de traitement explicite

Exemple du mutex POSIX

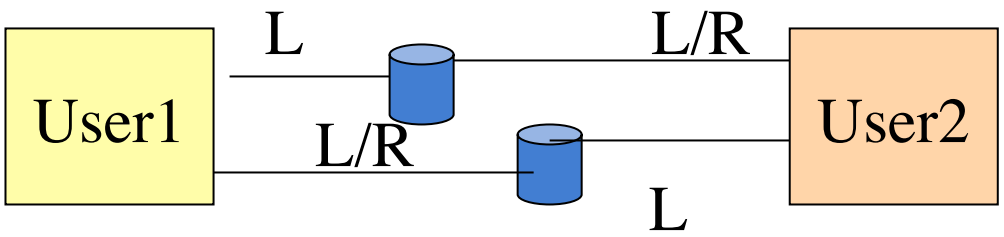
xterm

```
NAME
     pthread_mutex_lock -- lock a mutex

SYNOPSIS
     #include <pthread.h>

     int
     pthread_mutex_lock(pthread_mutex_t *mutex);

DESCRIPTION
     The pthread_mutex_lock() function locks mutex.  If the mutex is already
     locked, the calling thread will block until the mutex becomes available.

RETURN VALUES
     If successful, pthread_mutex_lock() will return zero, otherwise an error
     number will be returned to indicate the error.

ERRORS
     pthread_mutex_lock() will fail if:

     [EINVAL]           The value specified by mutex is invalid.

     [EDEADLK]          A deadlock would occur if the thread blocked waiting
                        for mutex.
:
```

Mode de défaillance non trivial

Le programme et son comportement

- 2 objets partagés en mode Rédacteur / lecteur par deux « User »



- On ne doit pas lire pendant l'écriture
- User1 lit fait une lecture sur O2, puis sur O1 puis une écriture sur O2
- User2 de manière similaire lit O1, puis O2 puis écrit O1
- Si User1 et User2 sont en situation d'interblocage
- pthread_mutex_lock défaille et génère un code d'erreur EDEADLK -> défaillance en valeur potentielle

Traitement (1)

- Si l'erreur n'est pas capturée :

```
// le mutex m1 protege le couple de variables x1, y1  (entier)
pthread_mutex_lock(m1);
tmp = x1*y1;
pthread_mutex_lock(m2);
tmp2= tmp - x2*y2;
pthread_mutex_unlock(m2);
x=sqrt(tmp2);
pthread_mutex_unlock(m1);
//
```

Contrôle du code de retour manquant

- Lectures et écritures concurrentes sans blocage -> pas de détection a priori
- **PB : Que fait on après la détection ?**

Confinement par moniteur externe

- Watchdog et interruption logicielle:
  - Défaillance constaté par l'absence de progrès dans l'exécution d'un programme
  - Causes possibles : boucle infinie, blocage sur un verrou pris et jamais restitué, récurrence trop longue …
  - Mise en œuvre : alarme, plus mécanisme de raz à des points critique du code.

Confinement par moniteur externe

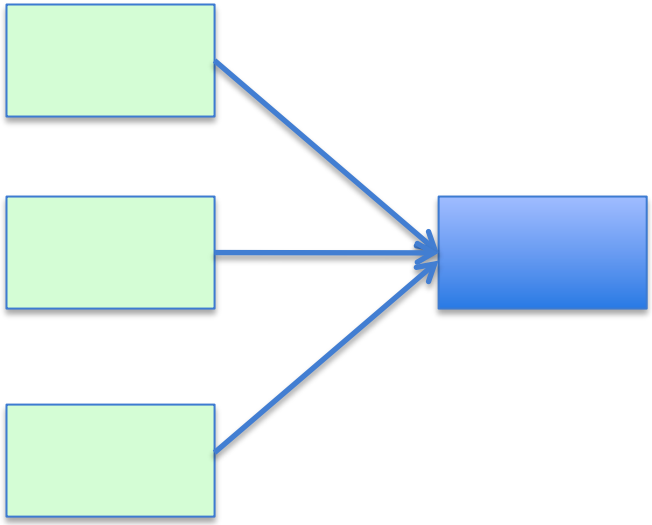
- Intérêt
  - Performances relativement indépendantes de la cause de la défaillance
  - Faible complexité d'implémentation « à court terme »
- Intégration du service dans de nombreux OS (ex Windows)

Recouvrement des erreurs 2 visions complémentaires

Corriger avant de poursuivre (réparable)

Recouvrement

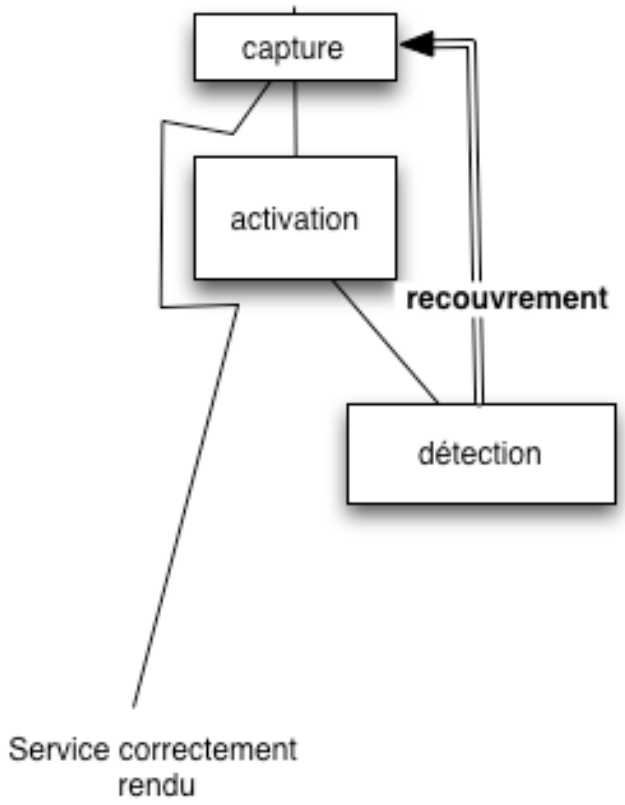
Choisir pour poursuivre (pas réparable)

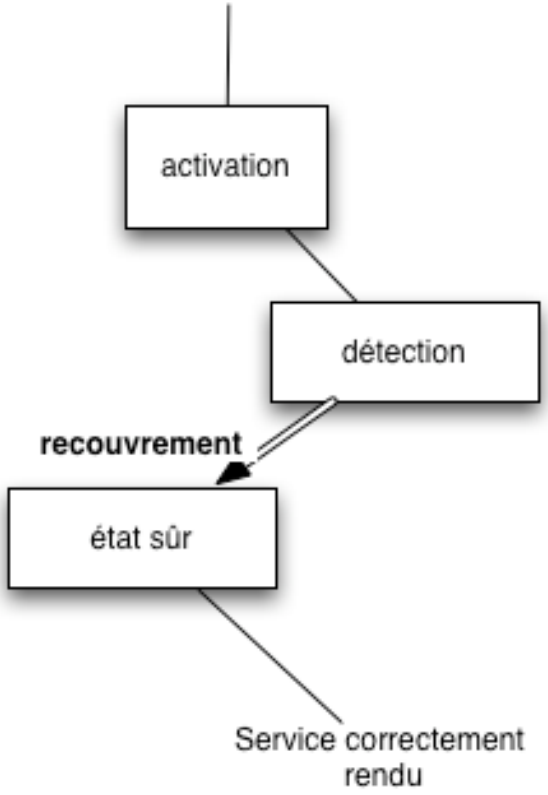


Masquage

Détection et recouvrement

- L'exécution du système est une séquence d'états
- **Détecter** la 1ere transition vers un état erroné
- **Reprendre** l'exécution depuis un état « correct »





Détection et recouvrement

Problème où trouver l'état correct ? :

- Dans le passé :
  - Hyp : il existe un mécanisme de capture d'états qui mémorise de manière répétée un état correct « récent »
  - Restauration sur détection d'erreur du dernier état sauvé
- Dans une liste d'états prédéfinis :
  - Hyp : Identification, a priori, d'états de poursuite d'exécution sûrs en fonction de l'erreur
  - Forçage d'une transition vers l'état de poursuite d'exécution correspondant à l'erreur détectée

Réflexions sur l'usage du recouvrement

- Dans l'approche « arrière », échec si :
  - la **faute est toujours active** et cause systématiquement l'erreur
  - la **latence de détection** est si grande que **l'état sauvegardé contient déjà une faute dormante ou que l'erreur s'est déjà propagé à l'extérieur du composant**
- Dans l'approche « avant » echec si
  - Les états de poursuite sûr sont erronés (mauvaise évaluation du lien erreur – état sûr).
  - La faute sous-jacente et toujours active et cause à nouveau l'erreur
- Comparaison sur une faute causée par un bug :
  - Le recouvrement arrière a de fortes de chances de rater si le bug est persistent
  - L'activation des états sûrs permet d'appeler un autre code…

En parallèle

- 2 visions
  - Vrai et faux parallélisme
    - Faux parallélisme == extension du backward recovery
    - Vrai parallélisme == masquage

Intérêt de la redondance

- Pb: comment s'assurer après recouvrement que l'erreur ne revienne pas …
- Raisonnement alternatif « l'indépendance » :

  Il est peu très peu probable qu'une même faute s'active sur deux exécutions indépendantes d'une même fonction

- La redondance ~ création de données, de composants, de « résultats indépendants »

Intérêt de la redondance (bis)

- La redondance permet de masquer les erreurs
  - Vote-élection / reconstruction / moyenne
    - Consensus
    - Code correcteurs d'erreur
    - Capteur de pression consolidés
- Problème : comment caractérise-t-on l'indépendance ?
  - Physique : réplication et séparation (COM-MON)
  - Processus : développement diversifié (NVP

Design pattern

- Architecture flexible pour la SdF
- Approches :
  - Programmatiques (syntaxe et enchainement)
  - Architecturales (modèles de flux et indépendance)
- On peut souvent combiner les deux…

N-version programming (process)

- L'idée est la même :
  avoir N versions indépendantes d'un même système :
  - 1 seule spécification
  - N équipes indépendantes de développement (lieu, formation, hiérarchie …)

  ⇒Une faute a peu de chances de s'activer de la même manière dans 2 versions distinctes

La transformée de fourrier discrète :

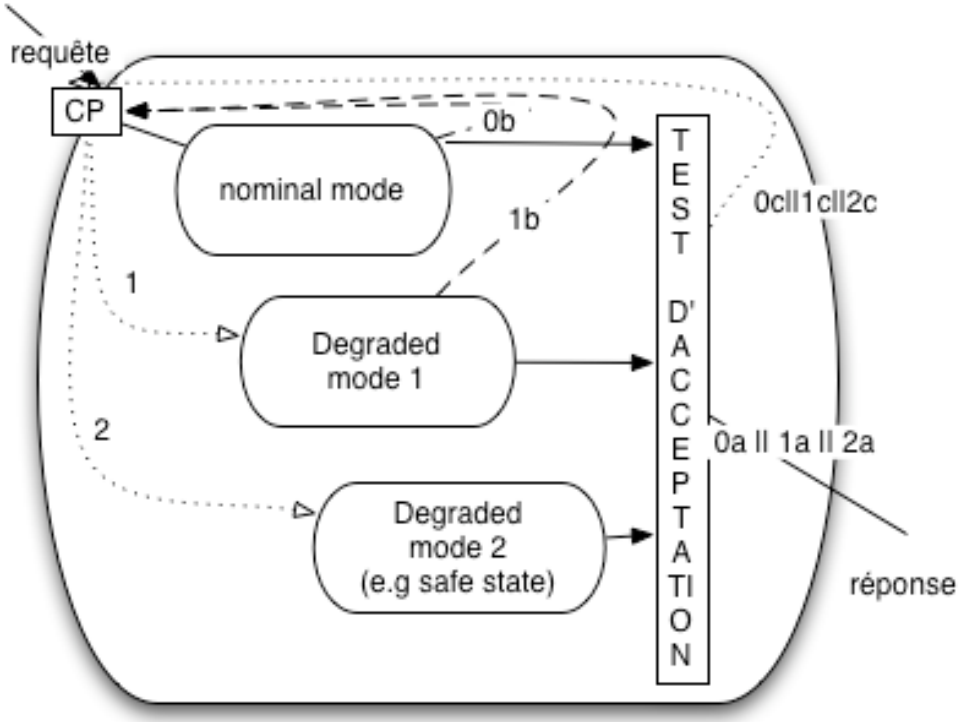
⇒ 2 algorithmes pour la calculer avec une sensibilité différente aux erreurs numériques

Recovery Blocks [Randall'95]

- Exemple d'intégration de NVP dans une application
- Un RB est un composant implémentant un service à la demande (client/server)
- La fonction est implémentée de N manières distinctes A1…AN
- Chaque version peut elle-même être un RB
- Il existe un test d'acceptation que doit pouvoir passer chaque alternative en l'absence d'erreur

Un exemple d'intégration de NVP : les Recovery Blocks

- A l'exécution :
  - Chaque requête génère la capture d'un point de reprise
  - La requête est transmise au premier alternat.
  - Si erreur ou échec du test, alors rétablir l'état du système et passer au bloc suivant
  - Dimension temporelle (watchdogs, timeouts)



CP : checkpoint; 1a,1b,1c chemins d'exécution possibles

Capture de contexte … et en vrai ?

- Principe :
  - Déterminer l'information représentative de l'état d'un système (variable, pile, ports de communication …)
  - Déterminer une méthode pour capturer un état cohérent de ces données
  - L'information enregistrée doit être suffisante pour recommencer l'exécution du système depuis cet état
- Obstacles
  - Usage de ressources systèmes et **effets de bords** (réservations de ressources, communications en cours)
  - **Implémentations concurrentes** du système
  - Quand doit-on capturer l'état ?

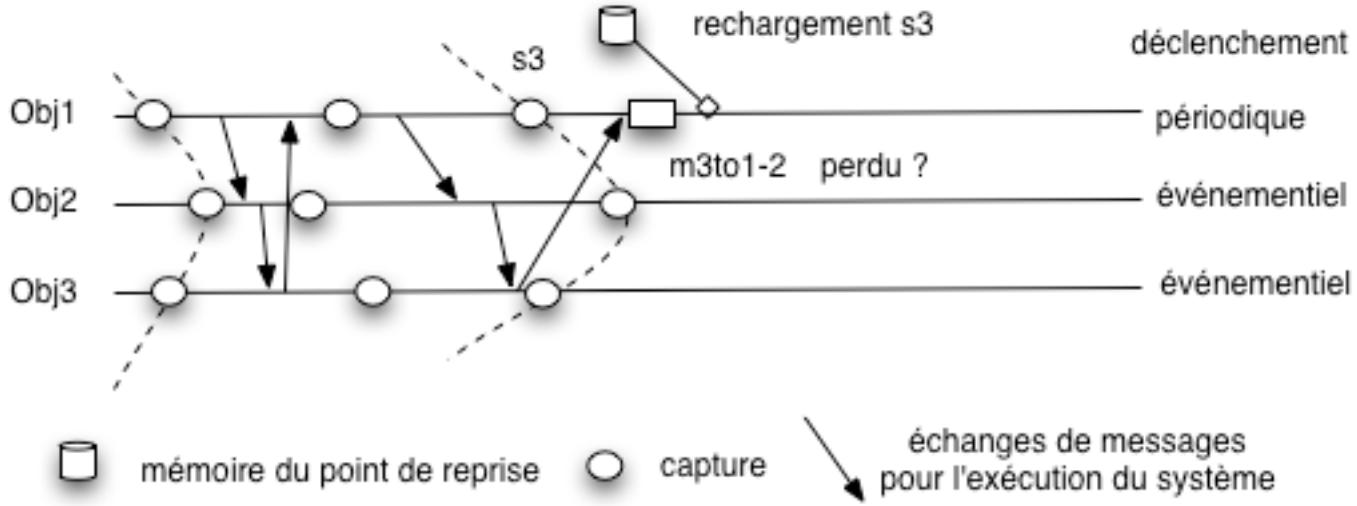
La vision centralisée

- Capture d'état ==
  capturer l'état d'une machine, d'un processus
- L'approche totale : recopier l'état de l'application et de sa plateforme d'exécution
  - Connaître l'OS (process / E-S / verrous)
  - Connaître le matériel (configuration des périphériques …)
- L'approche sémantique : identifier des états dans lesquels l'information utile est très faible
  - Connaître l'application à 100%

Le cas réparti et l'état inconsistant

- Principe : cas de figure désavantageux pour la capture d'états concurrents

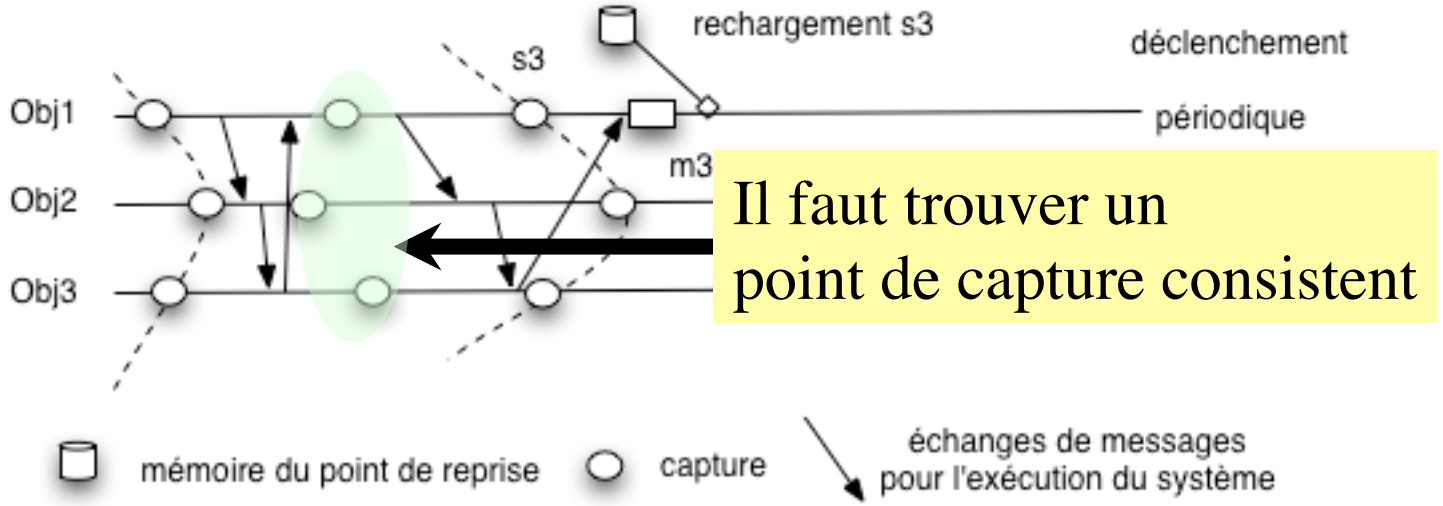
  Le délai de transmission des messages fait qu'un site croit avoir transmis une donnée alors qu'un autre l'a ignorée à la suite d'une détection / reprise



Le cas réparti et la cause de l'effet domino

- Principe : cas de figure désavantageux pour la capture d'états concurrents

  Le délai de transmission des messages fait qu'un site croit avoir transmis une donnée alors qu'un autre l'a ignorée à la suite d'une détection / reprise

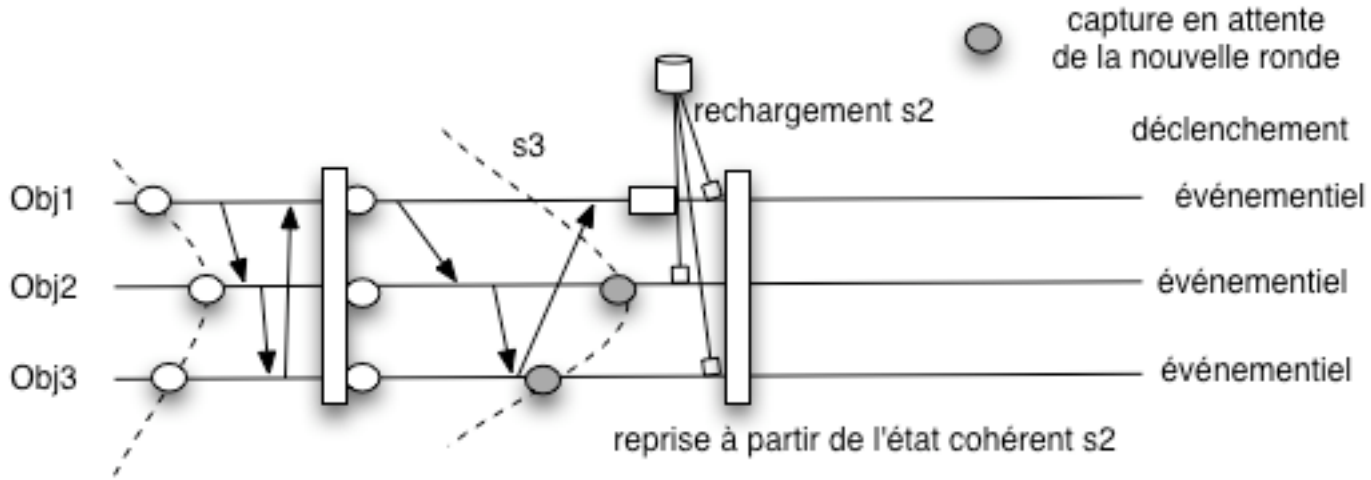


Les solutions pour la capture d'états concurrents

- La question : s'autorise-t-on à modifier en profondeur l'exécution du système ?
- Non – Utiliser une mémoire et un algorithme de reconstruction d'état global
  - Chaque site crée les points de capture qu'il désire.
  - Chaque point de capture est accompagné d'un historique des messages reçus (estampilles)
  - Sur défaillance, un état global consistent S est recherché dans l'historique des points de capture
  - => synchronisation + consensus/risque d'effet domino absolu (S=S0 état initial)
- Des solutions hybrides (pour information) :
  - Vase communicant temps de rétablissement / coût en mode nominal
  - Tout va dépendre de la taille des messages à échanger (taille de l'état) et du nombre de sites à mettre d'accord
  - Le graal : perturber le moins possible l'exécution du système et éviter la recherche potentiellement infructueuse de points de reprise
    - Algorithme de Koo & Toueg sur la notion de Capture en deux temps [Koo&Toueg 1986]

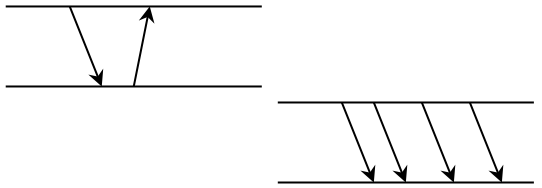
Les solutions pour la capture d'états concurrents

- La question : s'autorise-t-on à modifier en profondeur l'exécution du système ?
- Oui – Mise en place de rondes (ou conversations)
  - Aucun message ne peut être échangé entre deux rondes distinctes
  - Une nouvelle ronde ne peut commencer que si la précédente est terminée sur les autres sites



Un pied dans la théorie de la programmation distribuée

- La **mise en œuvre** de la **réplication** ou de la **capture d'état** nécessite différents **algorithmes distribués**
- Détection des répliques défaillantes vs communications lentes
  - Ping (aller – retour)
  - Heart-beat (message périodique)



- Les problèmes de détection et d'agrément distribués
  - Comment se mettre d'accord sur une valeur commune *consensus sur l'identité des sites défaillants*
  - Que se passe-t-il si en fonction du modèles des répliques (appelées souvent sites), du réseau ?
- Les protocoles de groupes : pratiques et nécessaires pour obtenir des preuves de correction des systèmes

TaF matérielle

Intérêt de la redondance matérielle

- La redondance logicielle sans redondance matérielle suppose que le matériel est isolé du logiciel -> cela reste difficile à prouver
- La redondance matérielle permet de rendre un système informatique tolérant aux défaillances du logiciel sans isolation particulière
- Le système est modélisé comme système distribué : 1 ensemble de calculateurs communicant par message

Modèle de fautes dans un système distribué

- 4 gabarits classiques de fautes
  - Silence (perte définitive du service)
    Arrêt du matériel/blocage de l'OS
  - Omission (perte occasionnelle du service)
    lien réseau peu fiable ou timing très mauvais
  - Temporelle (mauvais timing)
    mauvaise estimation de WCET
  - Byzantine (service totalement incontrôlé)
    modélisation classique pour la sécurité
- Haut niveau d'abstraction

1 faute == défaillance d'un site de calcul

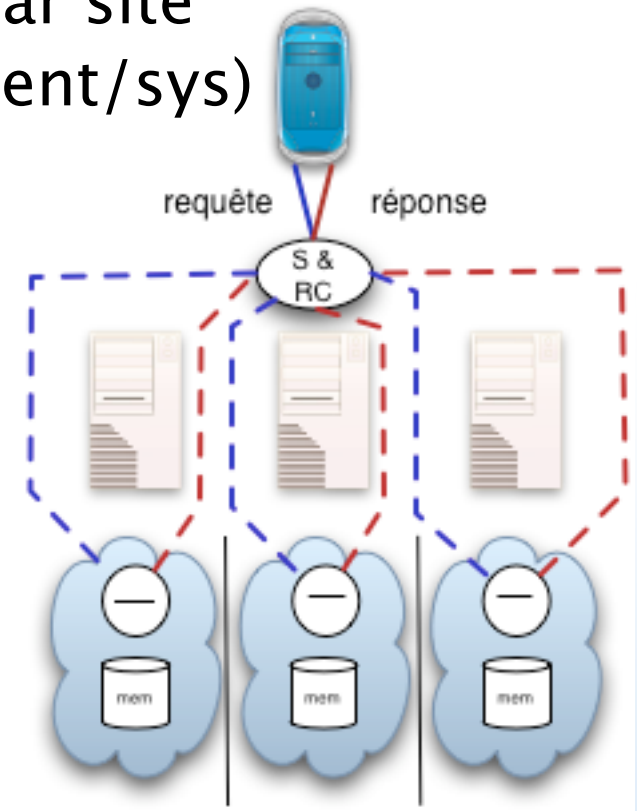
Stratégies de réplication

- Principe :
  - Déployer de manière concurrente plusieurs fois la même fonction
  - Utiliser un contrôleur pour
    - Piloter l'exécution de ces répliques
    - Assurer la transmission du résultat
- 3 gabarits (motifs de conception) :
  - Réplication active
  - Réplication passive
  - Réplication semi-active

Comment et avec quelles ressources ?

Réplication Active

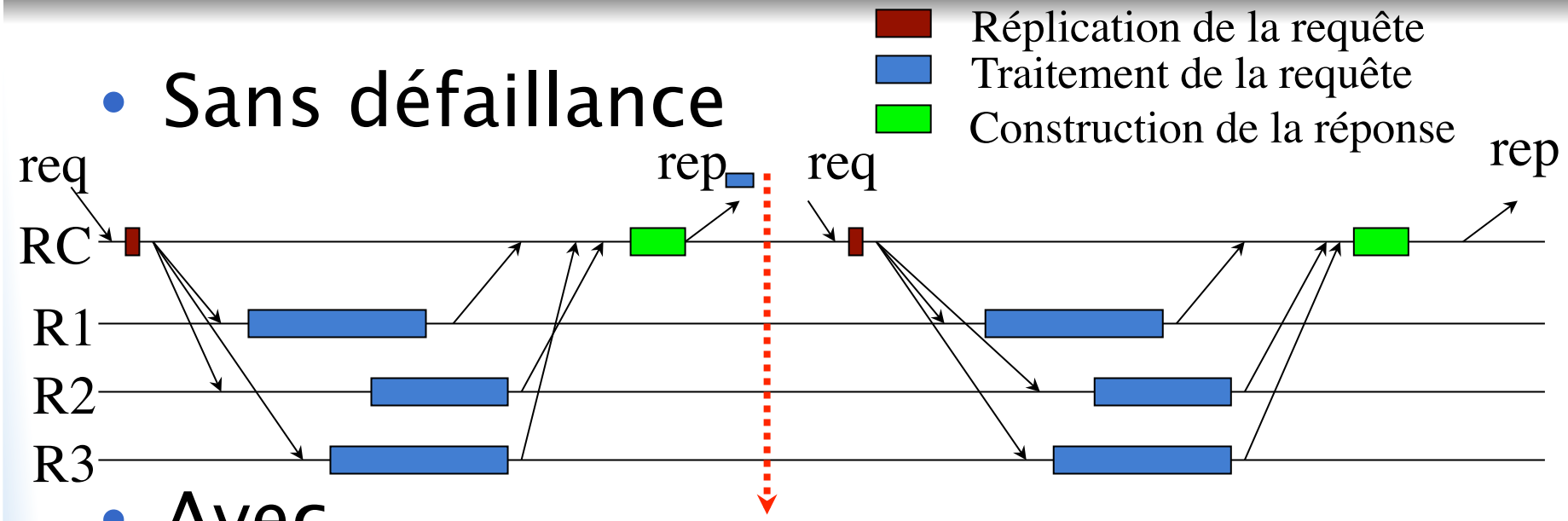
- N calculateur => 1 application par site
- 1 contrôleur pour interpréter (client/sys)
- Décision: vote
- Communications :
  diffusion des requête +
  agrément sur le résultat
- Déroulement
  1) Envoyer la requête à tous
  2) Chaque site exécute son service
  3) Construction de la réponse par vote majoritaire



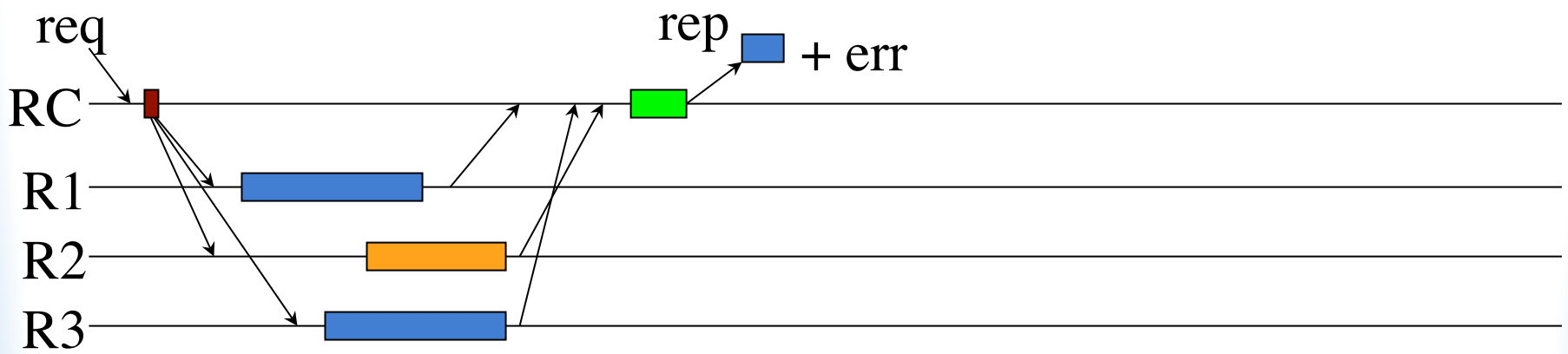
S & RC : vote et contrôle des répliques

Communication sans et avec défaillance

- Sans défaillance

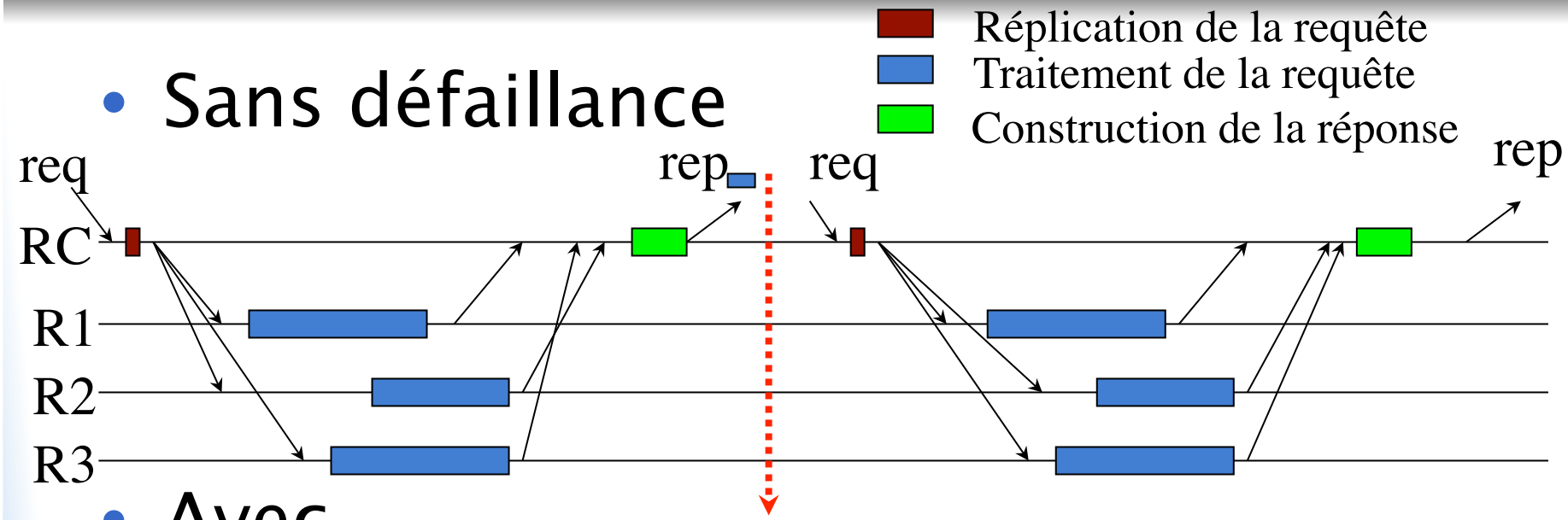


- Avec

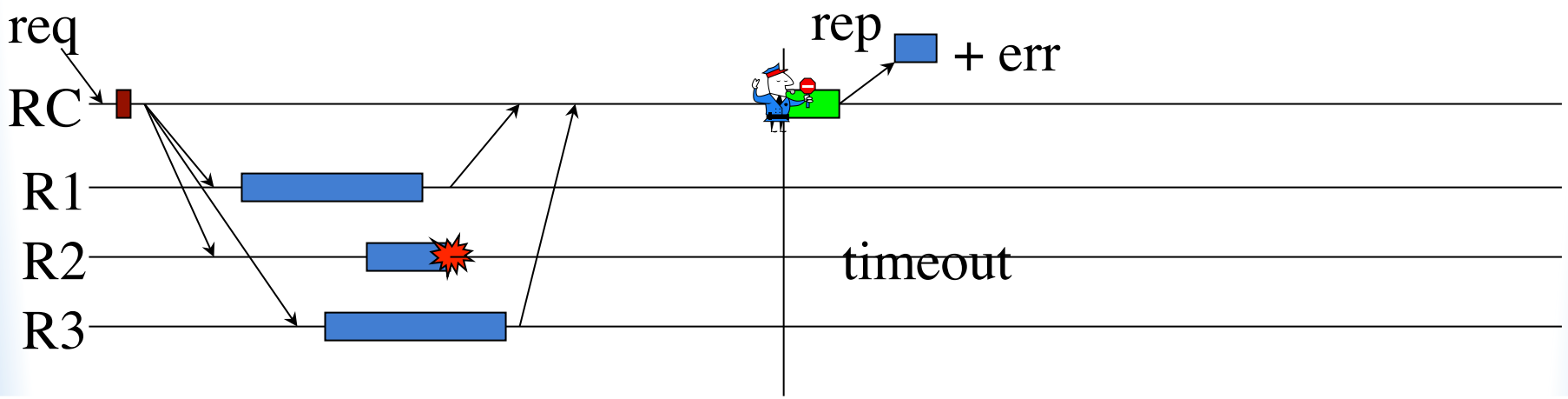


Communications sans et avec défaillance

- Sans défaillance



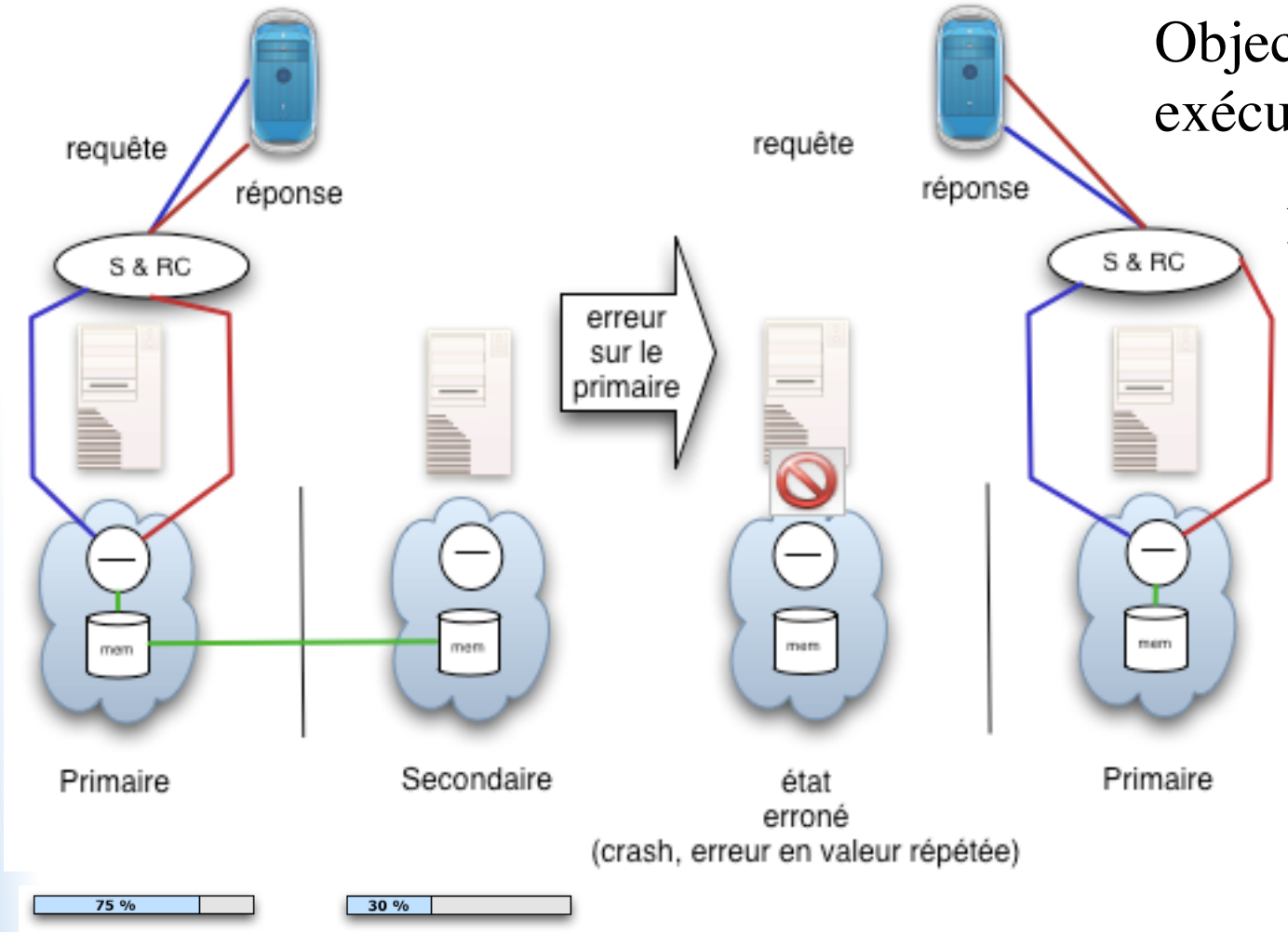
- Avec



Caractéristiques de la réplication active

- Modèle de faute toléré pour N répliques :
  - N/2-1 fautes Byzantines
- Détection réussie si au moins 1 correct
- Pour qu'une faute entraîne une erreur non détectée, elle doit s'activer sur :
  - le contrôleur des répliques,
  - Plus de N/2 -1
- Δt(régime normal / rétablissement) ~ nul

Réplication Passive



Objectif : éviter les exécutions concurrentes

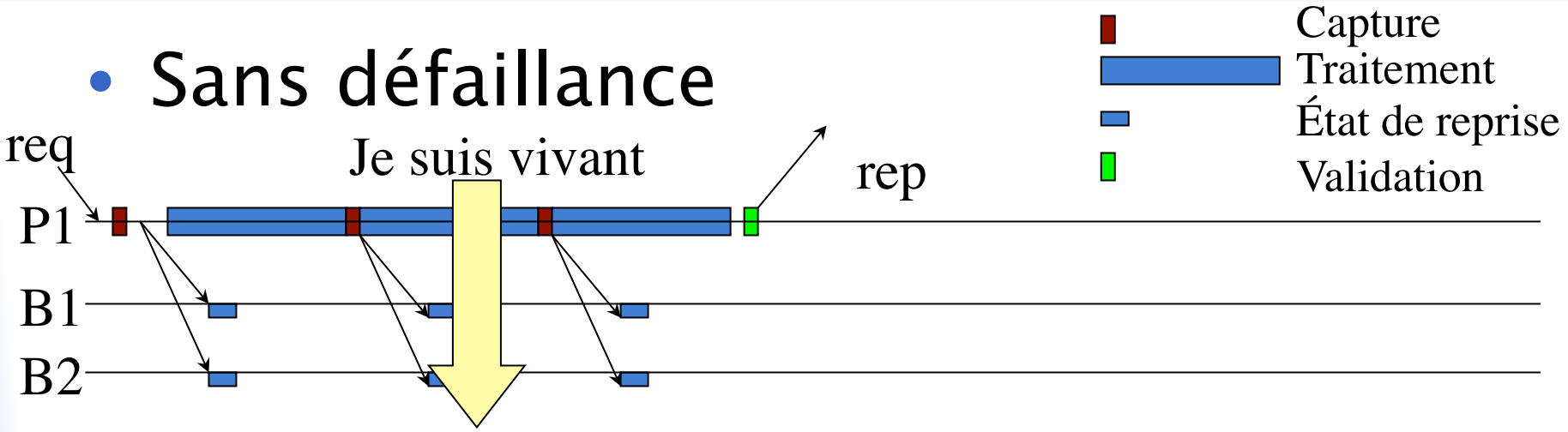
1 seule exécution à la fois

Communication : Transfert d'un état

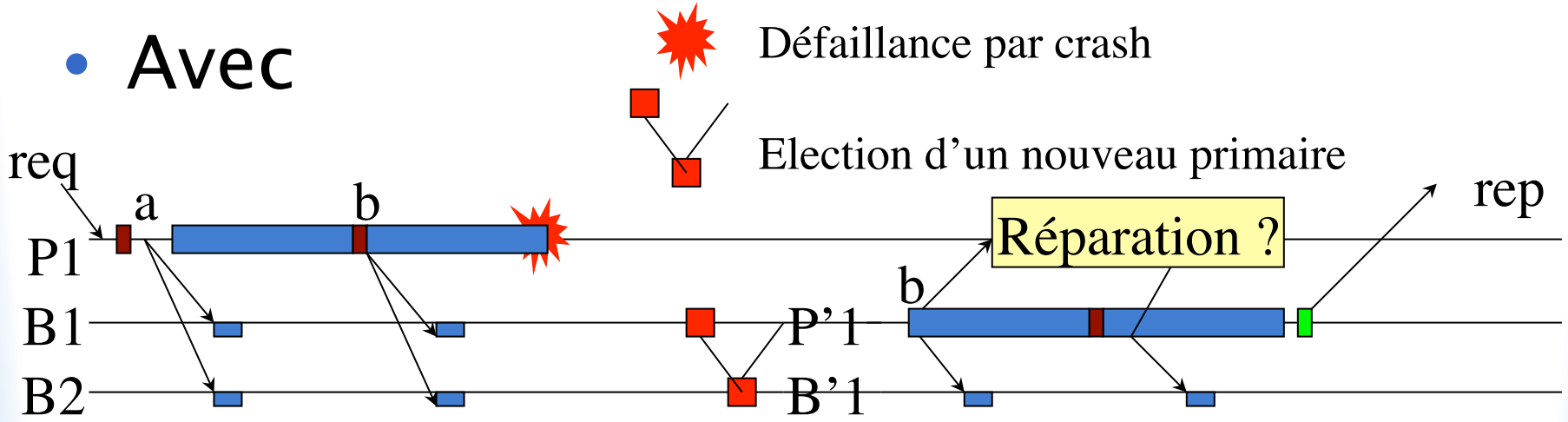
Le primaire doit capturer son état pour l'envoyer au secondaire

Communications et opérations significatives

- Sans défaillance



- Avec



Caractéristiques de la réplication passive

- 1 seul site exécute le service, jusqu'à la détection d'erreur (défaillance du primaire)
- Après détection d'erreur, basculement vers une réplique (nouveau primaire)
  - Identification du crash par « battements de cœur »
  - Élection d'un nouveau primaire
- Hypothèse forte : le primaire n'a qu'un seul mode de défaillance, le crash.
- Grand Δt pour un service avec et sans faute
- Le coût en communication dépend de la taille de l'état de reprise.

mode de fonctionnement dégradés

- Modes dégradés :
  État du système tel qu'une partie du système est dysfonctionnelle sans pour autant compromettre l'intégralité du service
- Mode fail-safe: mode dégradé n'assurant aucun service mais garantissant la sûreté des biens et personnes

Conclusion

- Tolérance aux fautes = domaine établi
  - Des motifs de conception utilisés mais à adapter à chaque application
  - Pas de dogme mais une prise de conscience
    - La Sûreté de fonctionnement est difficile à obtenir
    - Il y a nécessairement des compromis à faire mais « les bons »
- Importance des zones de confinement
  - Savoir si les défaillances sont détectées/ signalées
  - Lister les modes de défaillances connu, indiquer la cause si externe

Acronymes

- SdF : sûreté de fonctionnement
- TaF : Tolérance aux Fautes
- TMR, NMR : triple/ N – moduar replication
- RB : recovery blocks ou Rollback…
- ND : non – déterminisme (ou non déterministe)
- SR : stratégies de réplication

Références

Concepts de la SDF :

- PA Lee, T Anderson, JC Laprie, A Avizienis, … - 1990 - Springer-Verlag New York, Inc. Secaucus, NJ, USA
- A. Avizienis; J. Laprie; B. Randell & C.E. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing“, IEEE Transaction Dependable Sec. Computing 2004, 1, 11-33
- J.C. Laprie, “Guide de la sûreté de fonctionnement (2° Ed.)“, ed. Lavoisier, 330p

Techniques de mise en place de la TaF

- B. Randel and J. Xu, "The Evolution of the Recovery Block Concept," Software Fault Tolerance, M.R. Lyu, ed., John Wiley & Sons, New York, 1995, chapter 1
- Elnozahy, E. N., Alvisi, L., Wang, Y., and Johnson, D. B. 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 34, 3 (Sep. 2002), 375-408. DOI= http://doi.acm.org/10.1145/568522.568525
- [Cristian91] F. Cristian, "Understanding fault-tolerant distributed systems", Communications of the ACM, 34(2), February 1991
- [KD+89] Kopetz, Damm, Koza, Mulazzani, Schwabl, Senft, Zainlinger. "Distributed faulttolerant real-time systems: the Mars approach", IEEE Micro, pp. 25-40, February 1989

Références

- Xavier Défago and André Schiper, “Semi-passive replication and lazy consensus“, Journal of Parallel and Distributed Computing, 64(12):1380-1398, December 2004.
- Koo, R. and Toueg, S. Checkpointing and rollback-recovery for distributed systems. In Proceedings of 1986 ACM Fall Joint Computer Conference (Dallas, Texas, United States). IEEE Computer Society Press, Los Alamitos, CA, 1150-1158, 1986.
- Powell, D. 1994. “Distributed fault tolerance—lessons learnt from Delta-4“. In Papers of the Workshop on Hardware and Software Architectures For Fault Tolerance : Experiences and Perspectives: Experiences and Perspectives, M. Banâtre and P. A. Lee, Eds. Springer-Verlag, London, 199-217.

Algorithmique distribuée et prise de décision :

- Lamport, Leslie; Marshall Pease and Robert Shostak, "Reaching Agreement in the Presence of Faults". Journal of the ACM 27 (2): 228--234, April 1980
- Xavier Défago and André Schiper, “Semi-passive replication and lazy consensus“, Journal of Parallel and Distributed Computing, 64(12):1380-1398, December 2004.
- Chandra, T. D. and Toueg, S. 1996. Unreliable failure detectors for reliable distributed systems. J. ACM 43, 2 (Mar. 1996), 225-267.

Conception de stratégies de réplication :

- Schneider, F. B. 1990. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22, 4 (Dec. 1990), 299-319.