Institut
Mines-Télécom

Tolérance aux fautes

Thomas Robert
v 2020
thomas.robert@telecom-paris.fr

Motivations
Incident, Danger et responsabilité

Exemple motivant

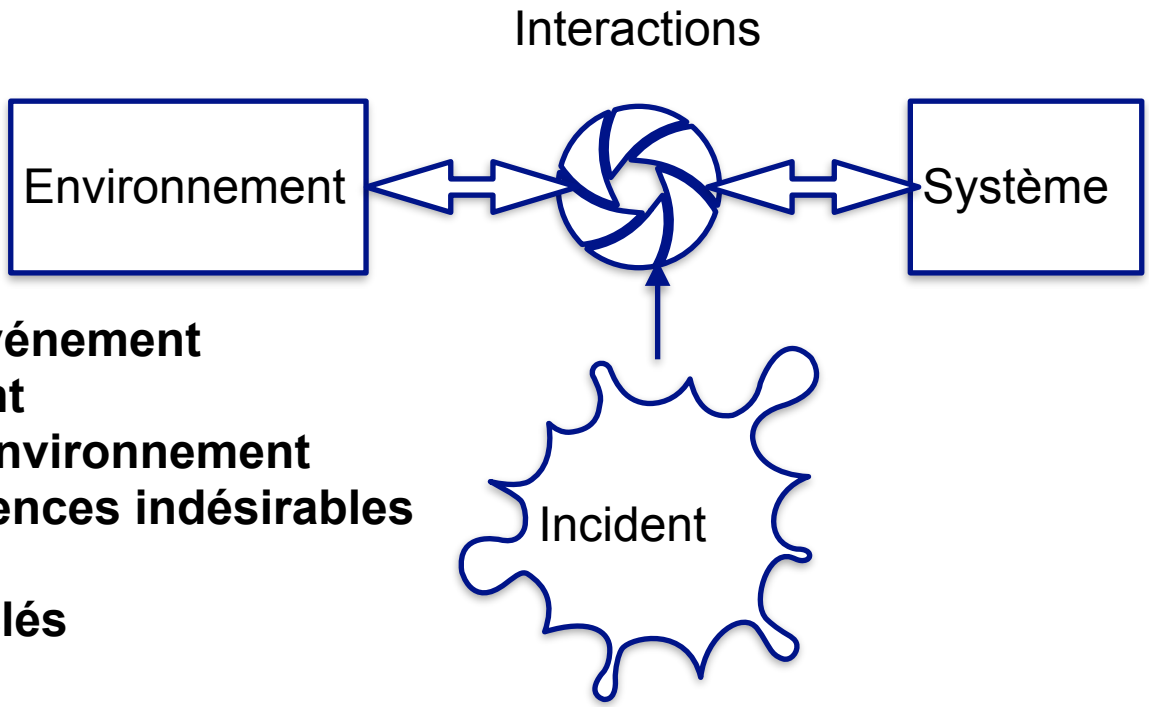
- **Veuillez commencer par consulter cette page « http:://…..com »:**

ERROR 404 ; page not found

- **Ceci est un incident … A partir de là tout dépend de**
  - **ce que contenait la page….**
  - **De l'impact de cette situation pour mon cours**
  - **De l'impact de cette situation pour vous (peut être indépendant de la ligne d'au dessus, ex la page contenait la correction de l'examen :))**
  - **De la possibilité de « réparer l'incident »**

- **Hypothèse 1 : c'était juste une image rigolote**
- **Hypothèse 2 : il y avait une recommandation pour l'examen qui devait s'afficher sur la page… moins drôle.**

Incident / Danger



Incident : état ou événement inattendu impliquant le système et son environnement ayant des conséquences indésirables

3 caractéristiques clés
- **Impact**
- **Vraisemblance**
- **Responsabilité**

Les incidents classiques par type de systèmes

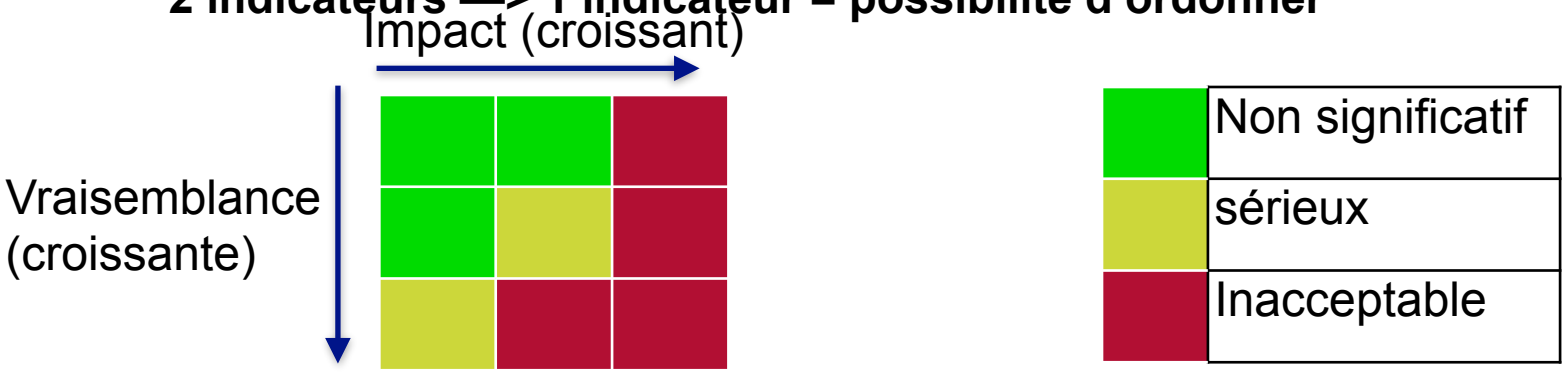
- Fusée : crash 1er lancement Ariane V: https://www.bugsnag.com/blog/bug-day-ariane-5-disaster
- Web services : many example with non trivial financial loss or impact on customer and employee https://www.cs.cmu.edu/~priya/PDL-CMU-05-109.pdf
- Radiographie : https://www.bugsnag.com/blog/bug-day-race-condition-therac-25

- Incidents : perte du contrôle de la direction de la fusée, mauvaise facturation d'un produit grand public, mauvais mode de fonctionnement d'un appareil de radio thérapie.
- Impacts : perte du système, perte financière / capital image, décès
- Vraisemblance : uniquement défini pour des événements à venir
- Responsabilité : il faut trouver la cause !!!! (Ariane multiples défaillance processus d'ingénierie

Notion de risque

- **Incident => coût => trouver un responsable**
- **Risque : une manière de combiner**
  - **La vraisemblance des incidents (fréquences, probabilité d'occurrence, temps moyen avant occurrence)**
  - **L'impact des incidents (coût financier / degré de gravité)**

- **Avantage on passe de**
  2 indicateurs —> 1 indicateur = possibilité d'ordonner



Gestion du risque

- 4 alternatives :
  - Accepter la situation
  - Refuser la situation (pas de produit)
  - Transférer le risque (sous-traiter)
  - Réduire le risque (réduire l'impact ou la vraisemblance)

- Conséquence pour l'ingénieur en informatique
  - Objectifs spécifiques à la maîtrise du risque
  - Mécanismes / architectures / méthodes de développement spécifiques à la réduction du risque

Ce qu'il faut retenir

- **Définition incident (tr 4)**
- **Différence Impact / Vraisemblance (ex. tr 6)**
- **Matrice de risque et différentes démarche de gestion**

Définition de bases sureté de fonctionnement

Entraves / Objectives / Moyens

Définition Sureté de fonctionnement à retenir absolument

Objectif:

Obtenir une **confiance justifiée** dans **la capacité du système** à **rendre le service attendu dans des conditions d'usage définies**

Conséquences

- **Définir les conditions d'usages sous lesquelles le système doit rendre le service attendu et/ou avoir un impact maitrisé sur**
  - **Son environnement (utilisateur(s) inclus)**
  - **Sur lui même**
- **Savoir ce que le système doit faire, lier le système ou ses parties à une fonction/responsabilité**
- **Définir des objectifs de garanties sur le respect du comportement attendu (qu'il soit sur le service attendu ou sur l'impact du système)**

Identifier le périmètre

- Système: unité de description permettant de distinguer l'objet d'étude de son environnement
- Structure : description des éléments à priori immuables du système (architecture)
- État : information variable au cours de la vie opérationnelle du système (effectivement représentée en mémoire pour du logiciel)
  - État interne: fraction non observable de l'état du système
  - Interface Système/Environnement : fraction de l'état du système partagée avec son environnement (E/S)

Les entraves (def)

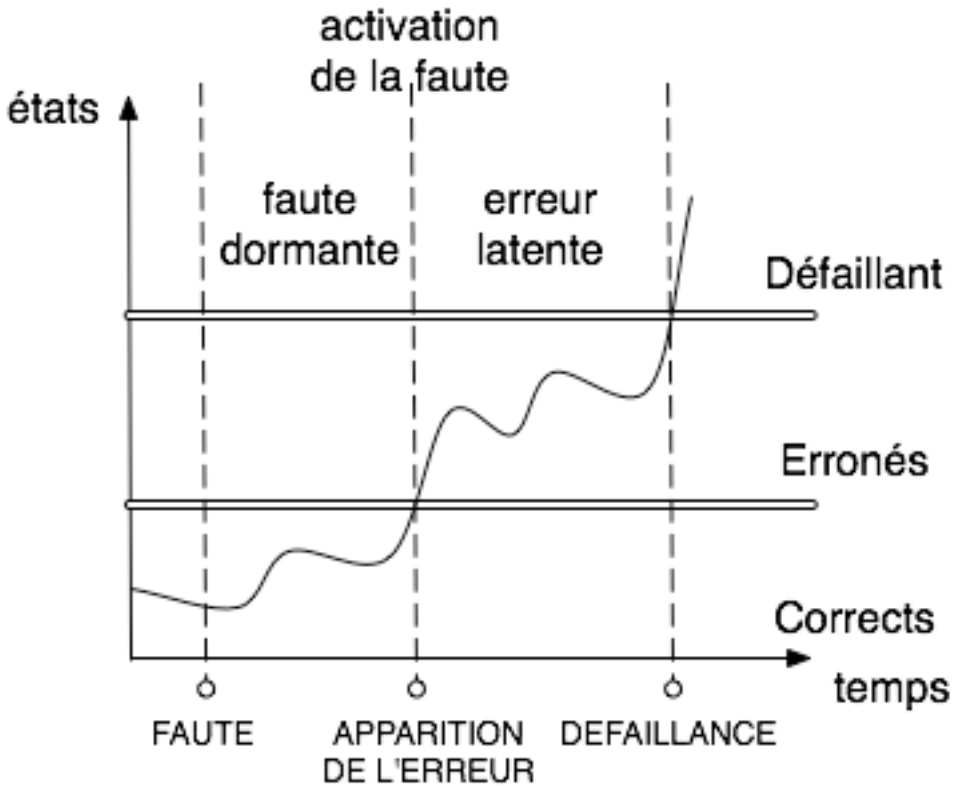
- **Défaillance :**
  écart observable entre le service attendu et le service rendu sur l'interface du système

- **Erreur :**
  Tout ou partie de l'état interne du système pouvant causer sa défaillance

- **Faute :**
  Cause de l'apparition d'une erreur (caractéristique structurelle ou changement d'état de l'environnement non prévu)

De la faute à la défaillance (cas simple)

- Evénements :
  - Apparition de la faute
  - Activation d'une faute
  - Défaillance
- Etats :
  - Correct OK
  - Erronés (?)
  - Défaillants (KO)



De la faute à la défaillance (cas complexe)

Exemple de l'application web pour la création de liste d'émargements / edition de convocation aux examens

- **2 machines : serveur web, base de donnée**
- **Sur chaque machine : un système d'exploitation, les applications (resp. apache, gestionnaire de base de données)**
- **Le serveur de base de donnée une fonctionnalité mal implémentée (la fonctionnalité permettant d'obtenir la liste des clés primaire d'une table) si cette fonction est appelée le serveur délivre une liste tronquée**
- **Le site web permet via l'exécution de cette requête l'édition de listes d'émargement + convocation aux examens, car les clés primaires sont les INEs ….**
- **Identifiez un incident, une défaillance, une faute, un état erroné dans un scénario d'exécution de ce système menant à l'édition de liste d'émargement et à l'envoi de convocations.**

Objectifs de tolérance au fautes

- **Format des objectifs : description d'un comportement ou respect d'une propriété exprimée sur une métrique**

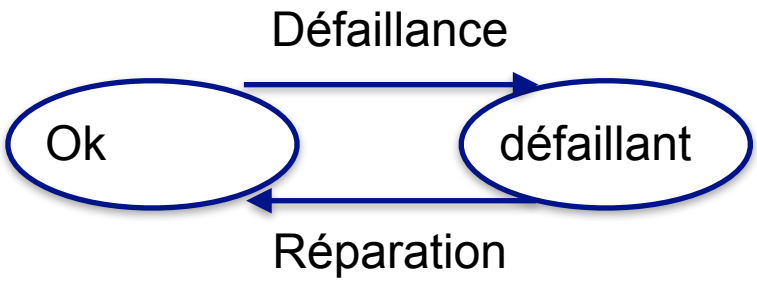
- **Exemple de description de comportement**
  - **La porte d'un ascenseur ne doit pouvoir s'ouvrir que lorsque l'ascenseur est à l'arrêt à un pallier et que les portes du pallier sont ouvertes.**
- **Exemple de description de garantie sur une métrique**
  - **Le temps moyen pendant lequel l'application web est incapable de répondre doit être inférieur à 10 minutes (en l'absence de problèmes de communication).**

Attributs de la tolérances aux fautes le jeu des 5 familles…. De métriques

- Indicateurs de disponibilité
- Indicateurs de fiabilité
- Indicateurs de l'intégrité d'un système (capacité à résister à des alteration de sa structure ou de son état)
- Indicateurs de maintenabilité
- Indicateurs de sécurité innocuité

Et sinon ça veut dire quoi …

- **1 systèmes : état abstrait = OK / défaillant, 2 transitions défaillance, réparation**



- **Td: temps jusqu'au franchissement de « Défaillance » — fiabilité**
- **R : proportion du temps passé dans l'état OK sur un période de temps — disponibilité**
- **Tr : temps jusqu'au franchissement de la transition réparation à partir de l'arrivée dans défaillant. — maintenabilité (surtout si cela implique un facteur humain, sinon participe à l'analyse de disponibilité**
- **Borne sur le coût financier de rester dans « défaillant » sécurité innocuité**
- **Impossibilité pour le système d'altérer sa structure physique (intégrité)**

Moyens pour garantir la sûreté de fonctionnement

- Prévention des fautes : s'assurer que l'on n'introduise pas de fautes
  - Pas besoin de les chercher
  - Pas d'incident découlant des défaillances qu'elle peuvent causer
- Elimination des fautes : le système contient des fautes mais on va
  - Les identifier
  - Les retirer
- Tolérance aux fautes : le système contient des fautes, elles vont d'activer
  - Maitriser la défaillance
  - Empêcher qu'une erreur se transforme en défaillance

Moyens pour garantir la sûreté de fonctionnement

- **Evaluation / prévision : évaluer la vraisemblance des activations de fautes ou des défaillance**
  - **Statistiques / probabilités / autre théories de l'incertitude**
  - **Modèle de système orientés prévision du comportement**
  - **Modèle formel du comportement pour valider les méthodes cité auparavant (élimination ou prévention).**

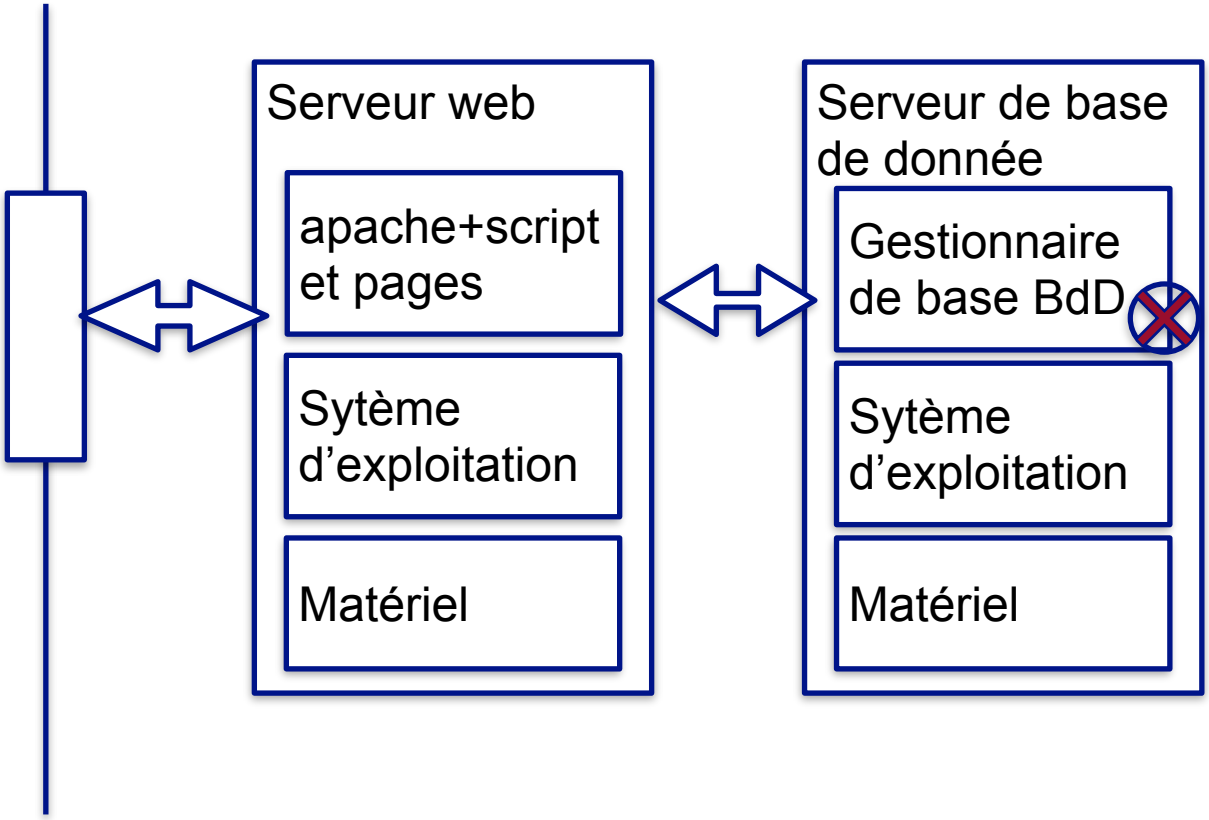
Vue d’ensemble de la tolérance aux fautes

Le principe de la TaF

- **Empêcher les défaillances :**
  1. **Éviter l'activation des fautes (différent de l'élimination au développement) => Traitement des fautes,**
  2. **Éviter qu'une erreur n'entraîne une défaillance (tolérance aux fautes) => Traitement des erreurs**

- **2 problèmes différents :**
  - **1. Difficulté = trouver la fautes à partir de l'erreur**
  - **2. Difficulté = manipuler l'état du système est complexe**

Rôle de la vision architecturale



Peut-on empêcher que l'erreur dans le code du gestionnaire de BdD n'entraîne systématiquement l'arrêt des scripts PHP coté serveur web ?

Il faut pouvoir avoir
1) conscience des problèmes
2) Un moyen de « vivre avec »

Zone de confinement d'erreur

- **Définition : périmètre/interface d'un système muni de mécanismes de protection :**
  - **empêchant une erreur de se propager sous forme d'une défaillance**
  - **sans être au moins détecté et signalée, ou confinée**
- **En pratique :**
  - **Description des défaillances associées à la zone de confinement** (i.e. description comportement attendu…)
  - **Possibilité de « contrôler/manipuler » les défaillances :** transformation d'une valeur erronée en absence de valeur
  - **Décomposition de systèmes en plusieurs zone de confinement**

Traitement des Fautes

- **Détection & diagnostic** : détecter **l'activation de fautes** (i.e. apparition d'erreur), si multiples erreurs **identification de la ou les fautes initiales** (propagation)

- **Isolation** : relier la plus petite partie de la **structure du système reconfigurable** à la(les) faute(s) initiale(s) (souvent une zone de confinement)

- **Reconfiguration** :
altération de la structure et de la logique du système pour inhiber la faute (i.e. plus d'activation)

Traitement des Erreurs

- **Détection** : observer l'état au cours de l'exécution
  - Test de vraisemblance : erreur si Observé ≠ Attendu
  - Exécution multiple + Comparaison, erreur si V1≠V2

- **Recouvrement** : manipuler l'état pour éviter la défaillance
  - Redéfinition ou reconstruction de l'état
  - Compensation de l'état erroné (cf redondance=moyen)

- **Utilisation de la détection**
  - Signalement/journalisation (exception Java)
  - Déclencheur des actions de recouvrement

Tolérance aux fautes logicielle

1. Enjeux
2. **Cas de la fonction / bloc**
3. Redondance et diversification

Confinement par fonction (enveloppe)

- Idée : une fonction d'une bibliothèque = ZCE
  - Pb : comment signaler / confiner les erreurs.
  - Pré-requis: taxonomie des erreurs
- Mise en œuvre :
  - encodage de l'état fonctionnel du composant dans la valeur de retour des fonctions
  - altération de la signature
- Pb : surcharge du type de retour ou altération de sa signature
- Exemple : code retour en C

Confinement par blocs

- Les exceptions :
  événement pouvant être déclenché de manière synchrone à l'exécution d'un code déclenchant un « déroutement » de l'exécution
  - encapsulation des traitements séquentiel dans des « blocs » (,accolades) pour définir un déroutement local,
  - arrêt et déroutement de l'exécution d'un bloc sur réception d'un événement
- Composants : blocs → méthodes, fonctions, boucles
- Propriétés supplémentaires : typage des exception pour définir un traitement par type, héritage / type abstrait pour langages à objets.
- Mise en œuvre :
  - Utilisation du typage pour définir le déroutement: 1 type d'exception=1 classe d'erreur
  - Possibilité d'avoir une exception propagée à travers la ZCE de la fonction
- Pb : souvent très mal utilisé malgré la puissance.

Décryptons une page de man

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 principe de l'enveloppe

- Soit T1 func(Tp1,...Tpn) une fonction pouvant défaillir à cause de fautes d'interaction (mauvais paramètres).
  - Pb : func ne signale pas son état de fonctionnement….
- On crée ft_func
  - Utilisation d'une référence (en valeur) pour la valeur de sortie de func
  - Code ajouté avant appel à func: Test des paramètres
  - Code ajouté après appel à func : Test de la valeur retournée par func
  - Valeur de retour status de fonctionnement

L'enveloppe par l'exemple : calcul d'une racine carrée

- Fonction PGCD (plus grand commun diviseur)
- Que se passe t il si x ou y négatifs ? « Recursion infinie » ça crash

```
int find_gcd(int x, int y)
{
    if(x > y)
        return find_gcd(x-y, y);

    else if(y > x)
        return find_gcd(x, y-x);
    else
    return x;
}
```

- Idem si x ou y trop grands (stack overflow)
- => pb on ne peut empêcher la propagation au niveau fonction si activation…

Confinement par moniteur externe

- Watchdog et interruption logicielle:
  - Défaillance constaté due à 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 + mécanisme de recouvrement de l'erreur
- Quelles solution pour le recouvrement ? (manipulation de l'état)

Tolérance aux fautes logicielle

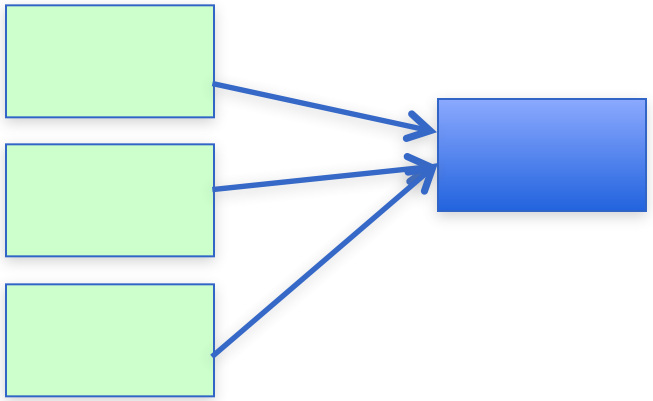
1. Enjeux
2. Le cas de la fonction / bloc
3. Redondance et diversification

Recouvrement des erreurs
2 visions complémentaires

Corriger avant de poursuivre (on modifie l'état afin de supprimer l'erreur)

Recouvrement

Choisir pour poursuivre (correction par filtrage/reconstruction — cf TaF données)



Masquage

Solutions génériques de recouvrement

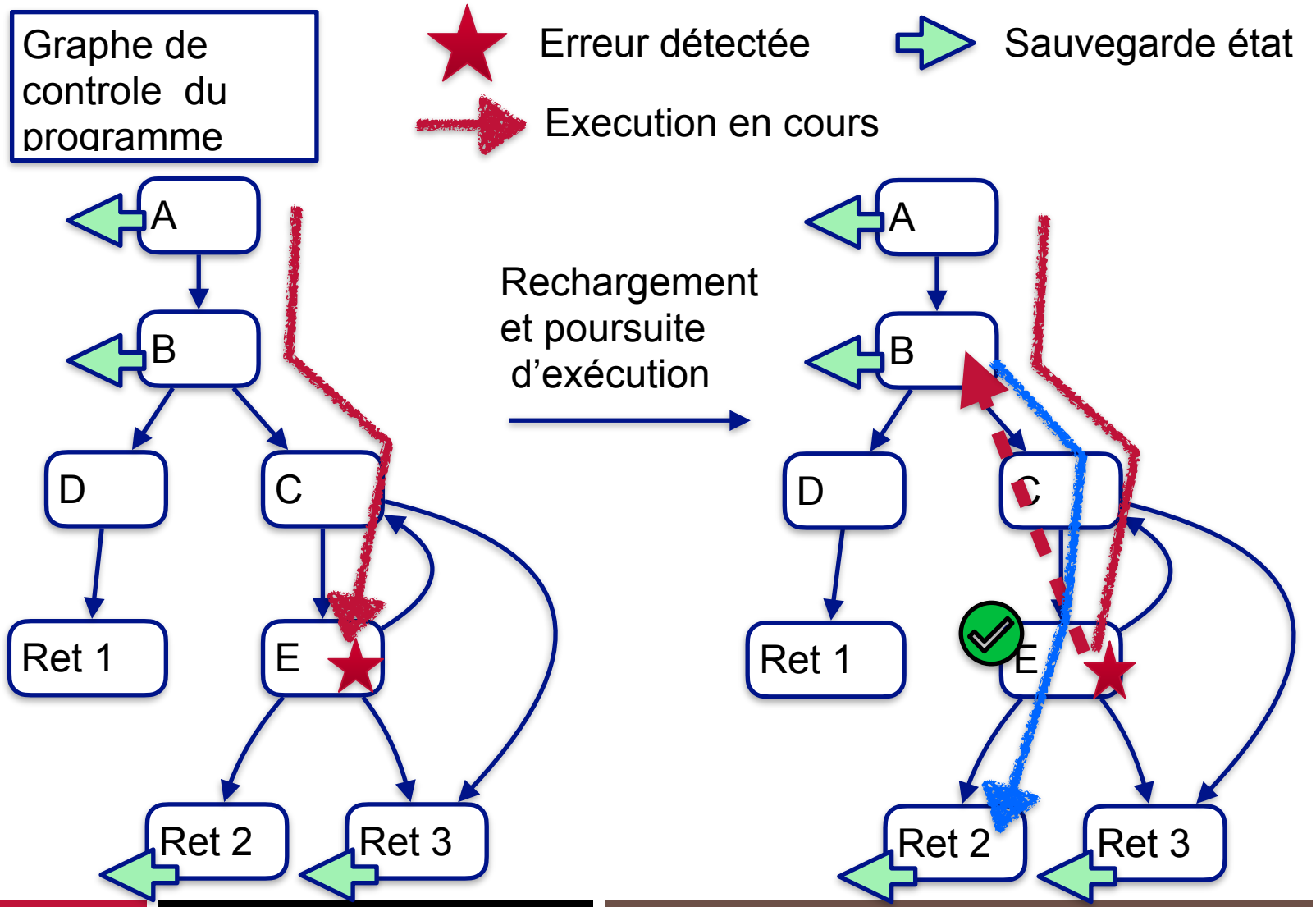
- 2 a priori sur l'activation de la faute
  - Hyp 1 : la faute s'active de manière aléatoire (avec une forte variabilité)
    => si on revient à un état passé, vraisemblablement aucune activation avant la fin
  - Hyp 2 : la faute s'active de manière systématique, la séquence d'exécution ne peut se poursuivre en l'état, il faut une alternative
    => on « saute » jusque'à un état par défaut pour exécuter un traitement alternatif pour lequel la faute n'a aucun effet

Principe du recouvrement arrière sur un composant logiciel.

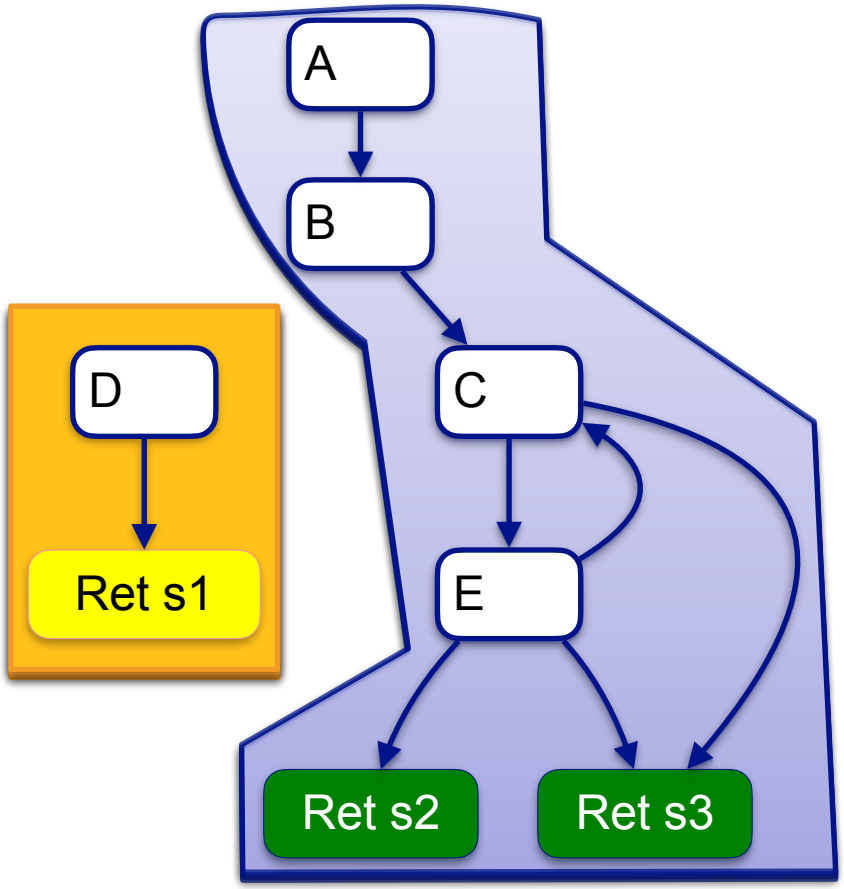
- Pré-requis : être capable de capturer un état == ensemble des informations pour une reprise de l'exécution à une étape définie

- Réaliser de manière régulière une capture d'état = sauvegarde de l'état courant du composant
  - De manière synchrone
  - De manière asynchrone (Attention aux data races)
- Lors d'une détection :
  1. Interrompre l'exécution en cours (pause)
  2. Déclencher la restauration de l'état sauvegardé
  3. Reprendre l'exécution à partir de l'état sauvegardé.

Représentation visuelle de la stratégie par retour arrière (rollback)

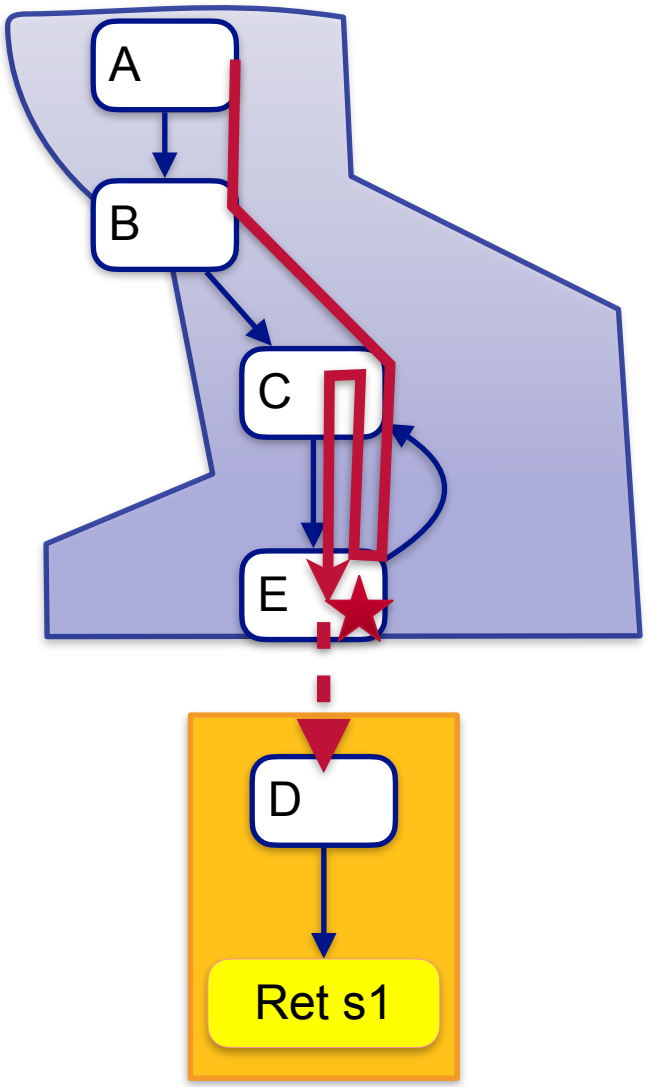


Recouvrement Avant (forward recovery)



- **Que faire si une activation systématique => reprendre la même exécution ne sert à rien …**
- **Hypothèse supplémentaire : il existe toujours une manière de rendre un service minimal**
  - **Graphe de contrôle « bleu » == service optimal**
  - **Graphe de contrôle « orange » == service dégradé « on fait au mieux »**

Logique d'exécution recouvrement avant forward recovery



- Sur détection de l'erreur, peu importe où la détection a lieu
  - Déroutement vers D
  - Poursuite de l'exécution selon niveau de service « orange »
- Exemples concrets
  - Application sauvegarde du contenu d'un disque vers un cloud
    => mode dégradé : affichage message d'alerte pb de connexion
  - Gestionnaire de connectivité internet
    Si pas de WIFI, Bluetooth
    … sinon affichage message « absence de connexion »

Tolérance aux fautes logicielle

1. Enjeux
2. Le cas de la fonction / bloc
3. Designs patterns
4. **Redondance et diversification (RB)**

Généralisons l'idée du recouvrement avant
Intérêt de la redondance

- **Rappel** : recouvrement avant
  == exécuter une solution différente => pas de propagation / pas d'erreur
- **Idée** : déployer N (N>1) composants implémentant la même fonction => **résultats comparables**
- **Avantages** :
  - solution de recouvrement : 1 correct parmi N
  - solution de détection : différents == défaillants
- **Désavantages** :
  - coût : CPU, développement, complexité
  - Correction difficile à prouver :
    ++ complexité => ++ fautes possibles
    et si tous rencontrent la même défaillance

Fautes de mode commun et redondance

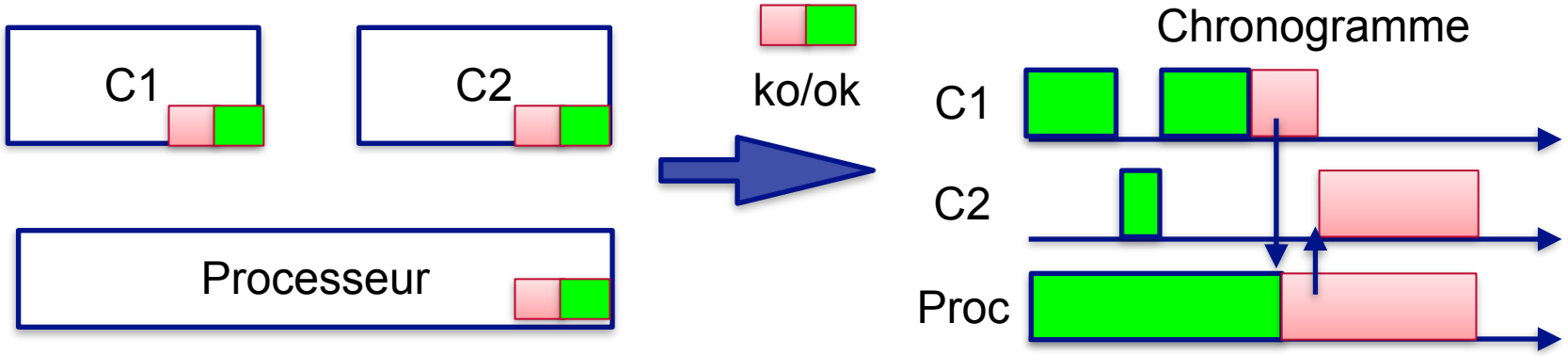
- **Définition : 1 faute s'active dans plusieurs composants => injecte des erreurs similaires dans ces composants**
- **Enjeux : peut remettre en cause des arguments de bon sens du type « les composants C1 et C2 ne défailleront simultanément qu'en de très rares circonstances »**
- **Exemple :**
  - **3 composants logiciel utilisant la même bibliothèque en interne**
  - **peuvent tous subir des erreurs identiques si une faute s'active de manière systématique dans cette bibliothèque …**
  - **Cause possible du phénomène : une implémentation incompatible avec un processeur**

N-version programming (processus de développement)

- Principe : disposer de N versions indépendantes d'un même composant logiciel
- Objectif : empêcher les fautes de mode commun logicielles (i.e. fautes de développement).
- Mise en oeuvre (cas optimal) :
  - 1 seule spécification
  - N équipes indépendantes de développement (lieu, formation, hiérarchie …)
  - Langages et outils de développement différents
  - Méthode de prévention / élimination de faute différentes
- Exemple : algorithme de calculs (transfo de fourrier, interpréter SQL ….)
- Difficultés cachées : fonction avec état pérenne, comportement aléatoire (générateur de nombres …), non déterminisme

Partage du matériel et propagation de fautes

- **Le modèle de partage de temps == ZCE temporelle**



- **Pb propagation d'une erreur dans le processeur qui se transfère de C1 à C2 sans qu'il n'y ait d'interactions explicites entre C1 et C2 ….**

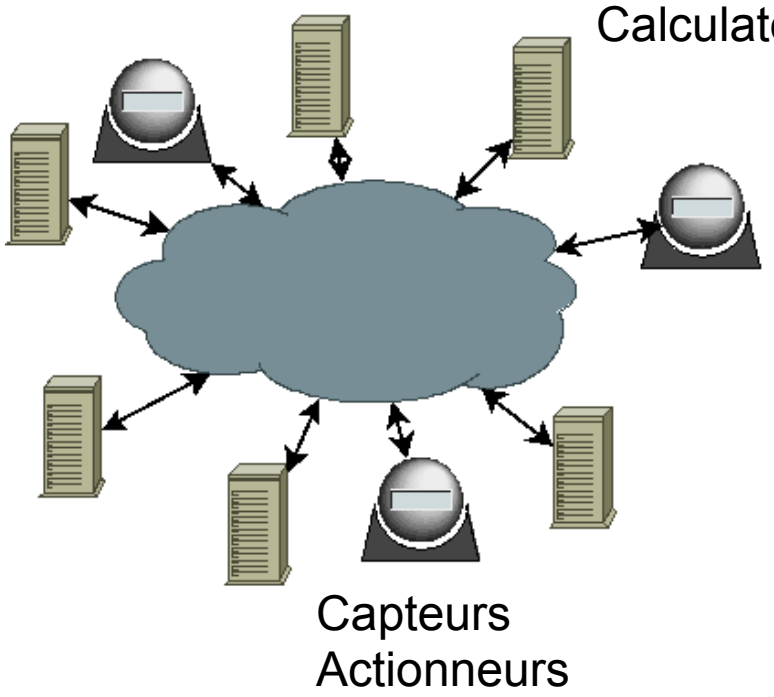
- **Solution : déployer C1 et C2 sur 2 matériels différents !**

Solution : stratégie de réplication

- Réplication = redondance + déploiement sur différents calculateurs
- Permet d'obtenir une plus grande confiance dans l'absence de faute de mode commun
- Un système == une collection de machines s'exécutant en parallèle …
  - Interactions : échanges de messages
  - Mode de défaillances = observables « à distance »

Modèle de système

- 1 réseau parfait
- Des machines qui défaillent = faute (système)
- Pas de défaillance du réseau
- Modèle de composant
  de type filtre
  - Flux E/S
  - 1 point d'entrée (PE)
  - 1 point de réception (PR)
  - Hypothèse simplificatrice
    (PE=PR)
- 1 stratégie =
  - 1 architecture
  - 1 comportement
    Sortie = Comportement (E)

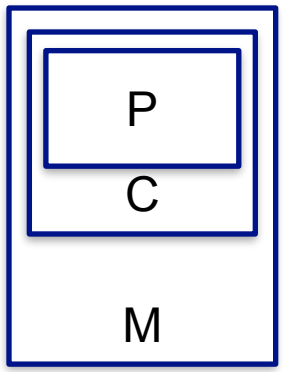


Modèle de fautes dans un système distribué
4 gabarits classiques de fautes

- Silence ou crash : plus de message émis
  effet permanent jusqu'à réparation
- Omission : certains messages ne sont pas émis (non détecté coté émetteur)
- Commission : certains messages sont émis plus d'une fois (cas attendu)
- Byzantin :
  - envoi de messages quelconques
  - sans capacité à se faire passer pour un autre
  - possibilité de synchronisation des réponses entre noeuds byzantins

Réplique d'un traitement P pour une stratégie S.

- P est déployé sur M dans le container C pour S.
  - P réalise le traitement si nécessaire
  - C contrôle les E/S (quoi, à qui, quand) selon S + état réplique par rapport architecture + contrôle d'exécution de P
  - M fournit les ressources / intégration réseau
- C n'est pas nécessairement implémenté en N version programming
- 2 répliques peuvent avoir un comportement très différent (à cause de C).

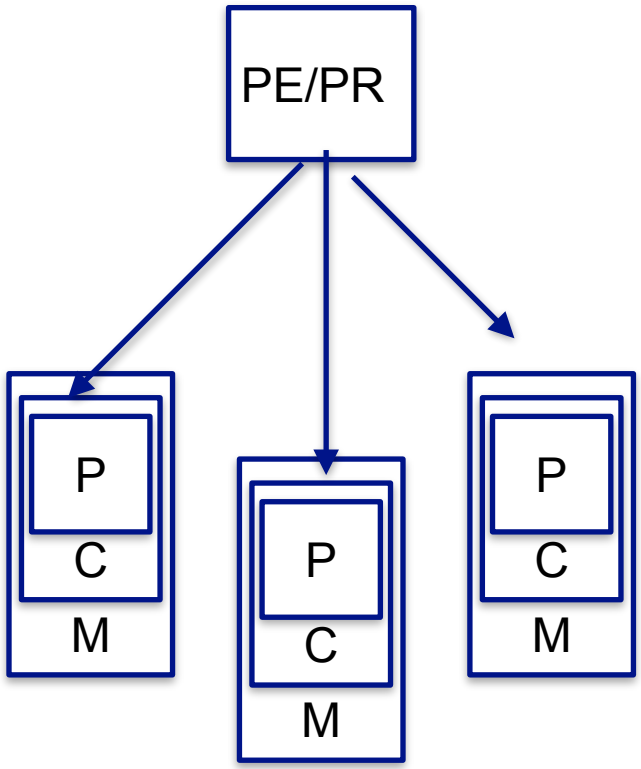


Réplique(P,S)

Réplication Active (N répliques)

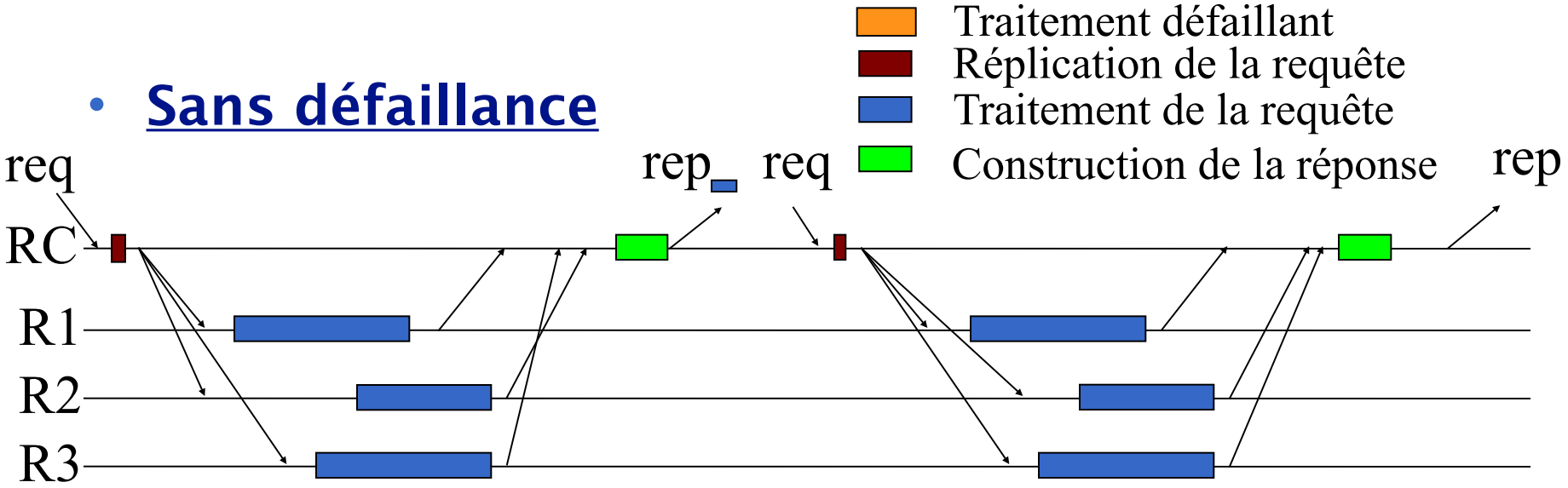
- 1 machine spéciale : le PE/PR
- Comportement d'une requête sur PE/PR
  - Diffusion à toutes les répliques de la requête
  - Attente réception résultats des répliques (bornée par un délai)
  - Identification et renvoi de la valeur majoritaire
- Comportement d'une réplique (≠ PE /PR)
  - Attente d'une requête du PE
  - Exécution de P sur la requête
  - Envoi du résultat au PR
- Le PE/PR est appelé voteur, et il n'a pas connaissance du « status » des répliques

- Le voteur infère l'état des répliques (comparaisons) => peut propager une défaillance)

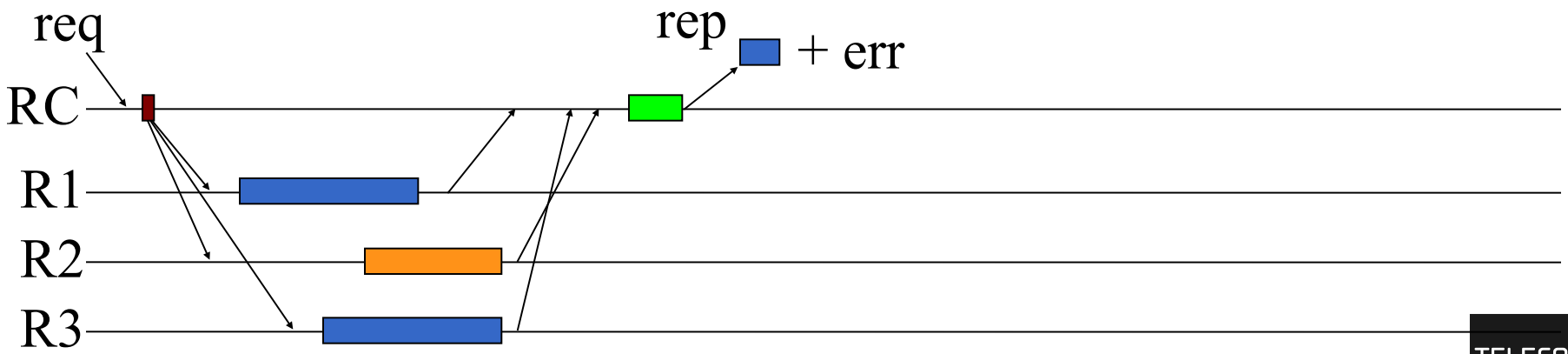


Communication sans puis avec défaillance

- **Sans défaillance**



- **Avec**



Caractéristiques de la réplication active

- Objectif : réponses correctes si majorité de répliques correctes
- Hypothèses :
  - Pas d'usurpation d'identité
  - Pas de faute d'interaction propagée dans le PE/PR (i.e. un byzantin ne peut corrompre les réponses correctes dans PE/PR)
  - Il faut une contrainte de convergence « si réplique correcte, alors temps de réponse borné »

Conditions de succès : réplication active

- Fiabilité : si N répliques tolère jusqu'à f fautes tel que 2f > N si hypothèse temps de réponse borné
- Disponibilité : dépend de comment le voteur garde une trace des noeuds qui défaillent
  - Cas simple : pas de mémoire, retourne la valeur la plus fréquente
    si f > N/2 => défaillance mais pas forcément de silence
  - Cas plus complexe : on met en quarantaine les défaillants
    => conditions de défaillances + strictes

Réplication Passive (N répliques)

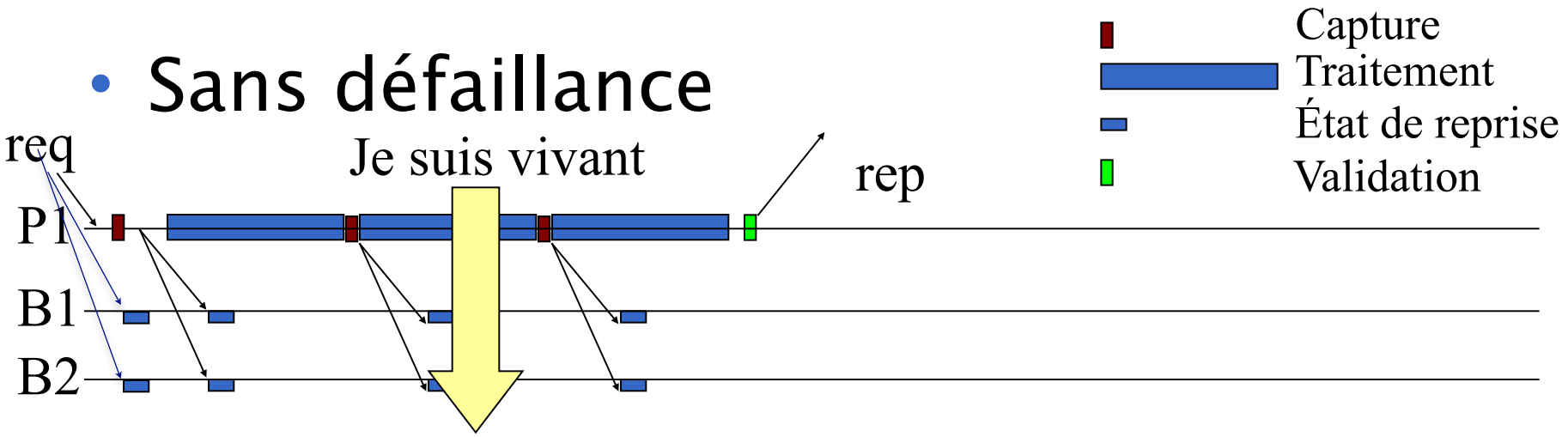
- 1 machine PE/PR
- Caractéristiques commune à toutes les répliques
  - état **role**: {primaire, secondaire}
  - État **capture** : contiendra la capture de l'état d'exécution (soit reçue, soit réalisée)
  - Toutes les répliques sauf 1 initialisées à « secondaire »
- Si réplique avec role = primaire
  - Si le changement de role vient d'avoir lieu, chargement de l'état à partir de **capture**
  - Capture de manière régulière son état d'exécution et le diffuse à toutes les autres répliques
  - Sur réception de le requête démarre le traitement

Réplication passive (N répliques)

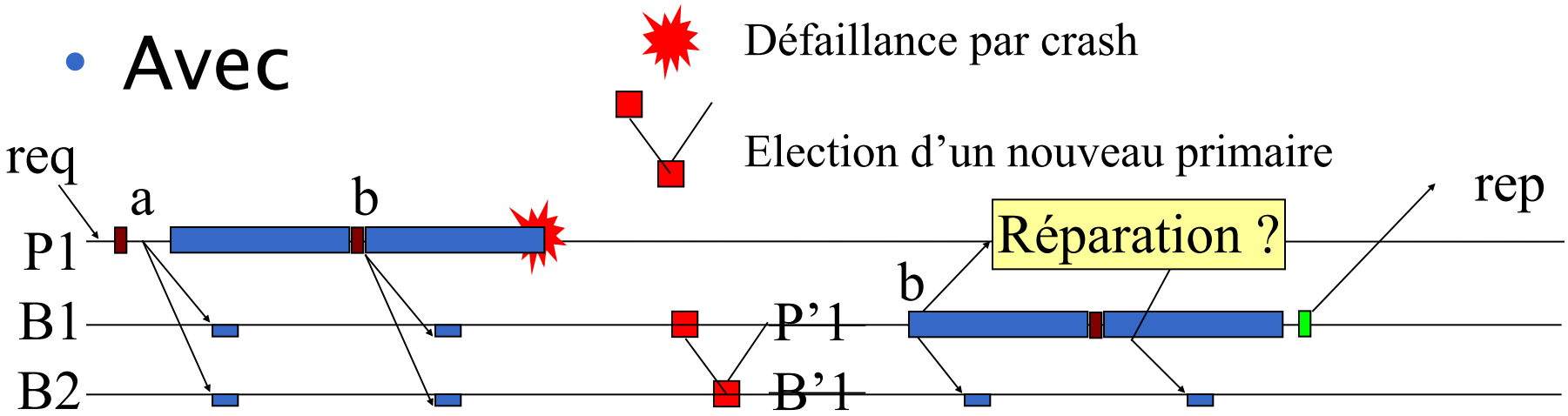
- Si réplique avec role = secondaire
  - Attend une sauvegarde d'état dans un délai borné
  - Si délai dépassé, synchronisation avec les autres secondaires pour élection d'un primaire
  - A la fin du protocole d'élection de primaire changement d'état pour le secondaire choisi, communication au PE/PR identité primaire
- Comportement PE/PR
  - Sur réception d'une requête transmission à toutes les répliques de la requête (i.e. initialisation de l'état capture)
  - Puis attente de la réponse, et sur réception transmission au « client »

Communications et opérations significatives

- Sans défaillance



- Avec



Caractéristiques de la réplication passive

- 1 seule réplique exécute le traitement à chaque instant (attention réseau parfait requis)
- Objectif : tolérer le crash/défaillance silencieuse
- Peut tolérer jusque'à N-1 défaillances
- Amélioration possible pour faire en sorte de tolérer omission / défaillance réseau par suppression de messages
- Remarque : on peut forcer une réplique à implémenter le mode crash dans certaines limites

Violation des hypothèses de fonctionnement et résilience

- Chaque réplique possède des modes de défaillance supposés
- 1 stratégie de réplication = 1 garantie de tolérance sous condition
  - Du respect du modèle de défaillance couvert pour chaque répliques
  - Du respect du nombre maximal de fautes pouvant être tolérées
    => l'état de ces hypothèse change
- Résilience : caractéristique d'un système pouvant maintenir ses objectifs de sureté de fonctionnement même en présence de changement concernant les modèles de fautes actifs
- Remarque 1: on peut chercher à vérifier que certaines combinaisons de modes de défaillance peuvent être tolérés en plus des cas standards (résilience passive)
- **Remarque 2: on peut reconfigurer la structure de la stratégie pour s'adapter à un nouveau mode de défaillance observé == maintient de la mitigation du risque= résilience active**

Distinguer disponibilité / fiabilité pour la réplication active

- On peut être disponible et non fiable … si N fautes mais il y a des cas plus pénibles
- Pb : le PE/PR ne sait pas si il est fiable à 100% mais il continue à produire des résultats (disponible)
  - Disponible = continue à produire des résultats
  - Fiable = produit un résultat correct

Lien avec l'algorithmique distribuée théorique

- Mot clés : consensus, systèmes asynchrone
- Idée de base : concevoir des algorithme avec des hypothèses très limitées sur le temps de réponse des répliques
- Pourquoi à votre avis :
  - Car le système n'est pas temps réel et ne peut jamais vraiment borné les temps de réponses dans ces conditions
  - Car le temps est un concept relatif => pas de borne sur le temps de réponse => pas de fréquence minimale de fonctionnement
- Tout algorithme dit « synchrone » a une fréquence maximale de fonctionnement, pas les algorithme asynchrones….

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
- Sûreté de fonctionnement difficile à obtenir
  - Des compromis à faire,
  - Notre objectif : vous donner les clés pour savoir lesquels
- Importance des zones de confinement
  - Savoir si les défaillances sont détectées/signalées,
  - Lister les modes de défaillances connus, indiquer la cause si externe
- Comment vérifie-t-on, valide-t-on que l'on a « bien travaillé » ? => évaluation (cours suivant)

Acronymes

- SdF : sûreté de fonctionnement
- TaF : Tolérance aux Fautes
- TMR, NMR : triple/ N – modular replication (réplication active)
- LF ou PB: leader follower synonyme de Primary – Backup (et donc de primaire secondaire)
- RB : recovery blocks ou Rollback…
- ND : non – déterminisme (ou non déterministe)
- SR : stratégies de réplication

Références
(pour aller plus loin — hors cours)

- 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 (pour aller plus loin — hors cours)

- 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.