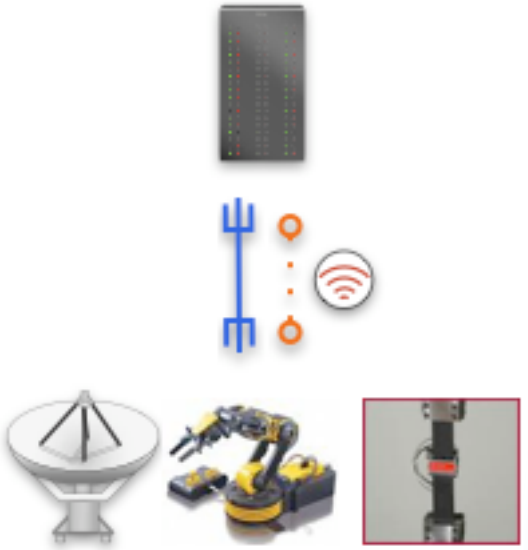
Tolérance aux Fautes des Systèmes Informatiques

Thomas ROBERT

SE301 – M2 SAR

Motivations (1/3)

- 2 points de vus d'un système embarqué :
  - Un système informatique
    matériel + logiciel
  - Un système physique
    capteurs/actuateurs
    support de communication



Et si le système fonctionne mal ?

Motivations (2/3)

- La conception et la réalisation dépendent de :
  - La nature de l'environnement considéré
  - La description du service à rendre
- Zéro défaut ≠ sans problème : l'environnement, l'usage entravent le fonctionnement du système
  - Environnement imprévisible
  - Le facteur humain à l'usage et à la conception
- Quelle confiance peut on vouloir associer à un système ?

Motivations (3/3)

Sciences et techniques de l'ingénierie

-> Sureté de fonctionnement

Vous en faîtes sûrement un peu tous les jours,

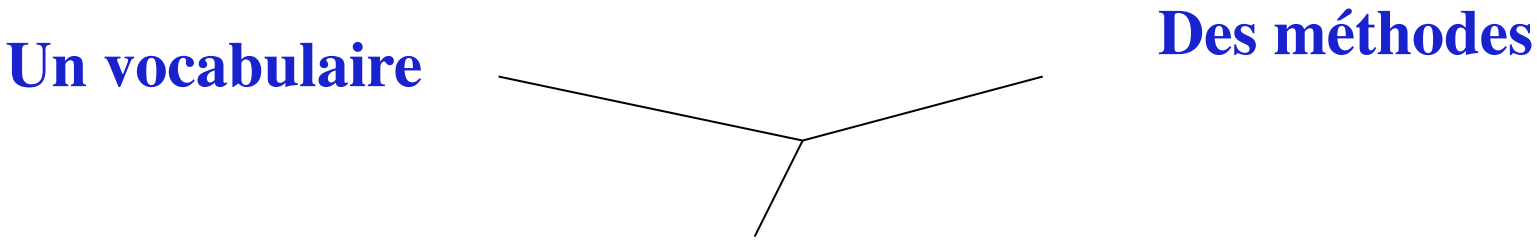
Le but est de rationaliser cette pratique

Plan du cours

- Concepts, objectifs et verrous
- Savoir faire élémentaire en TaF
- TaF dans le cadre temps réel
- Introduction au TP
- Bilan

La sûreté de fonctionnement en informatique…

- [Avizienis et al'04]
  "*l'**étude** et **mise en œuvre** de « **services** » en lesquels on puisse **placer une confiance justifiée***"



La vision système (def)

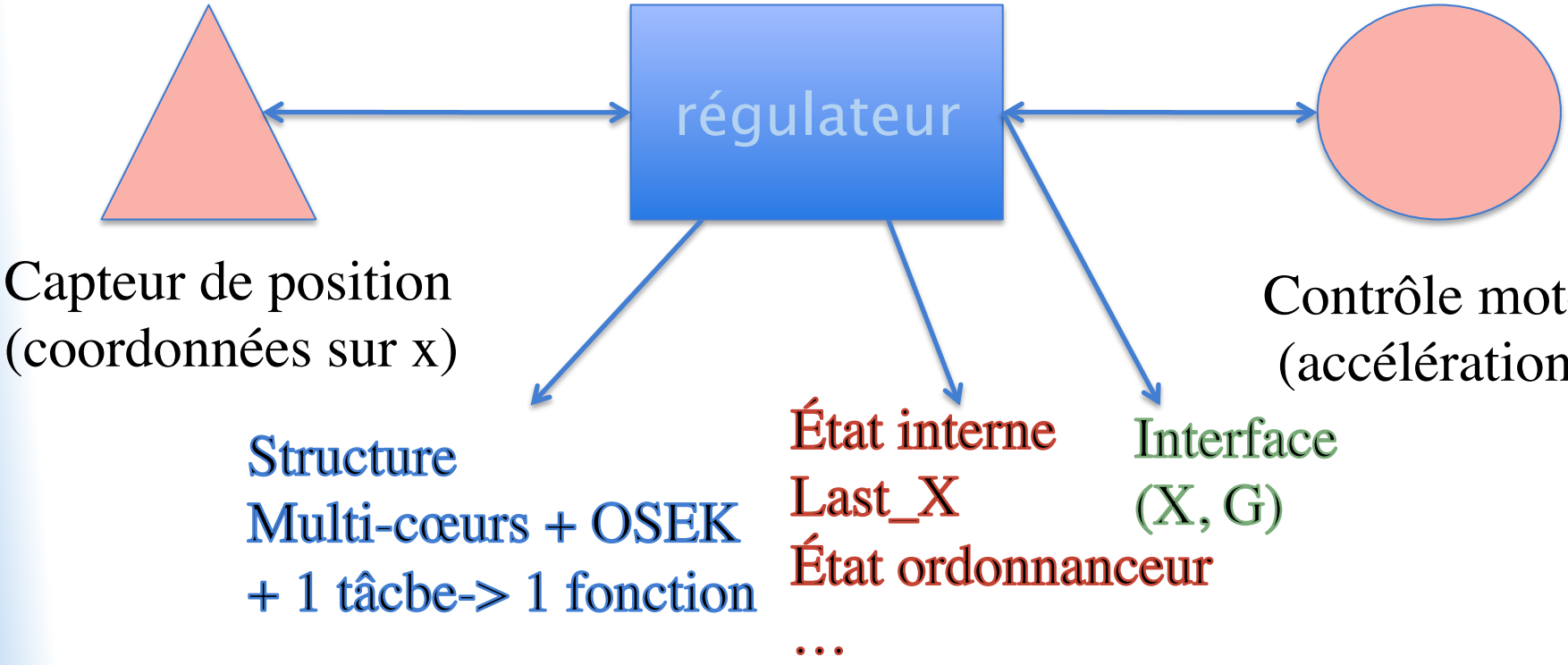
système: unité de description permettant de distinguer l'objet d'étude de son environnement

- **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 : fraction de l'état du système partagée avec son environnement (E/S)

La vision système par l'exemple

- La vision système ::
  structure + comportement + …

Exemple : Le régulateur de vitesse du métro



Identifier le risque et le danger

- Risque :
  - 1 événement redouté
  - 1 coût (qualitatif ou quantitatif)
  - 1 vraisemblance (probabilité, explication des causes)
- Danger: abstraction des causes du risque

  Etat conjoint du système et de son environnement ayant le potentiel d'entrainer l'événement redouté

Gestion du risque

Gestion du risque =

identifier / caractériser / traiter

Traitement du risque =

1. Refus
2. Transfert
3. Limitation
4. Acceptation

Limiter le risque
Sureté fonctionnelle

- Limiter le risque =
  - Limiter/empêcher l'occurrence
  - Altérer l'impact
- Exigence fonctionnelle de sureté ==
  - Décrire une fonction/processus
  - Décrire une caractéristique structurelle
  - Décrire des conditions d'usage

  Limitant le risque

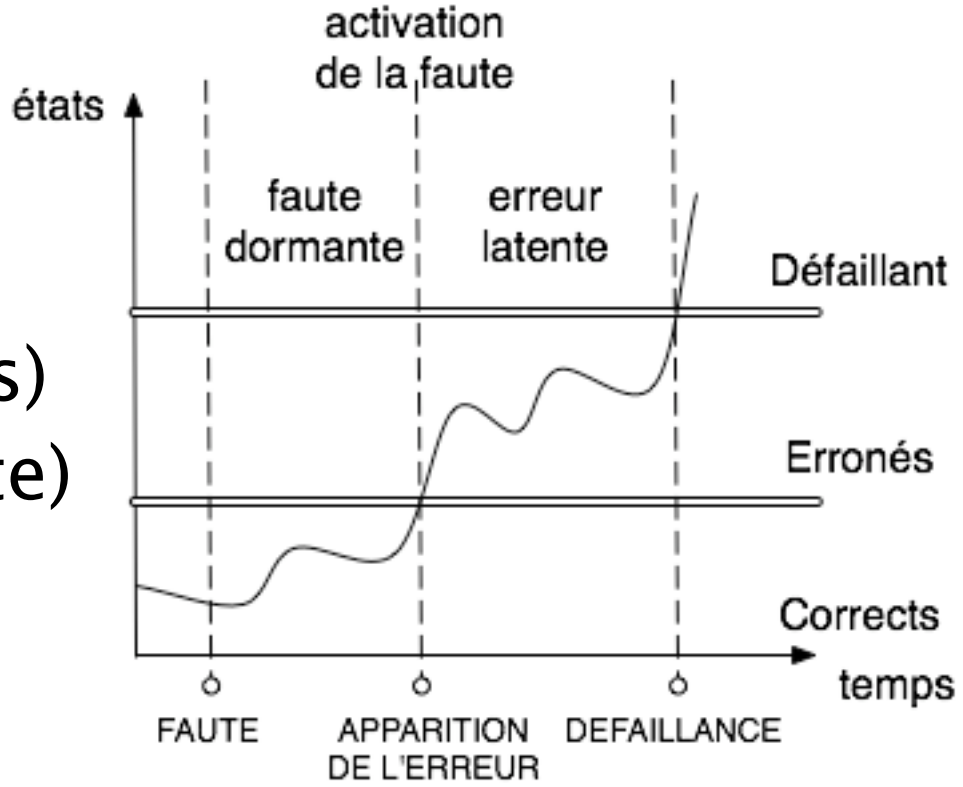
Les entraves (def)

- Un vocabulaire pour comprendre :
  - **Défaillance** :écart **observable** entre le service attendu et le service rendu
  - **Erreur** :
    Tout ou partie de **l'état interne** du système pouvant causer sa défaillance
  - **Faute** :
    Cause de l'apparition d'une erreur (origine structurelle ou liée à l'état)
- Quel est le lien entre ces concepts?

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)
- Distinguer les défaillances pour faciliter leur analyse / traitement
- Ce qui peut être pris en compte :
  - Nature de la déviation
  - Perception de la déviation (locale / globale / incertaine)
  - Amplitude (durée dans le temps / distance à l'attendu)

Les attributs

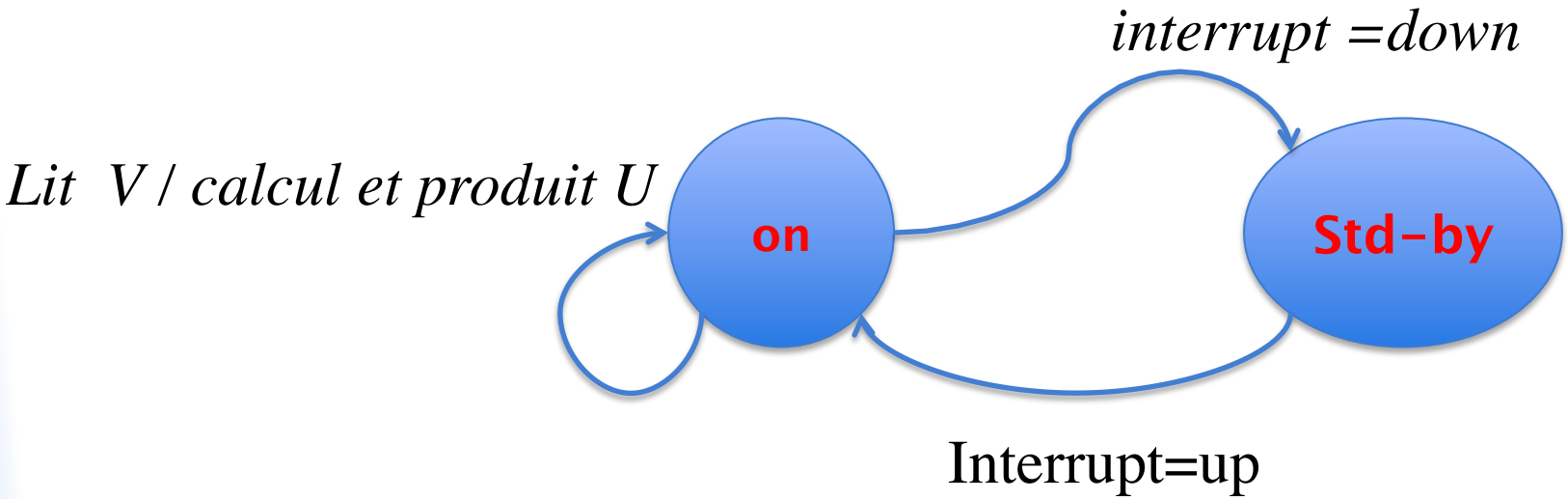
- Par où commencer ? => Les attributs
  - Disponibilité (availability)
  - Fiabilité (reliability)
  - Sécurité-innocuité (safety)
  - Intégrité (integrity)
  - Confidentialité (confidentiality)
  - Maintenabilité (maintainability)

…. En détail cela donne quoi …

Attributs en pratique (1)

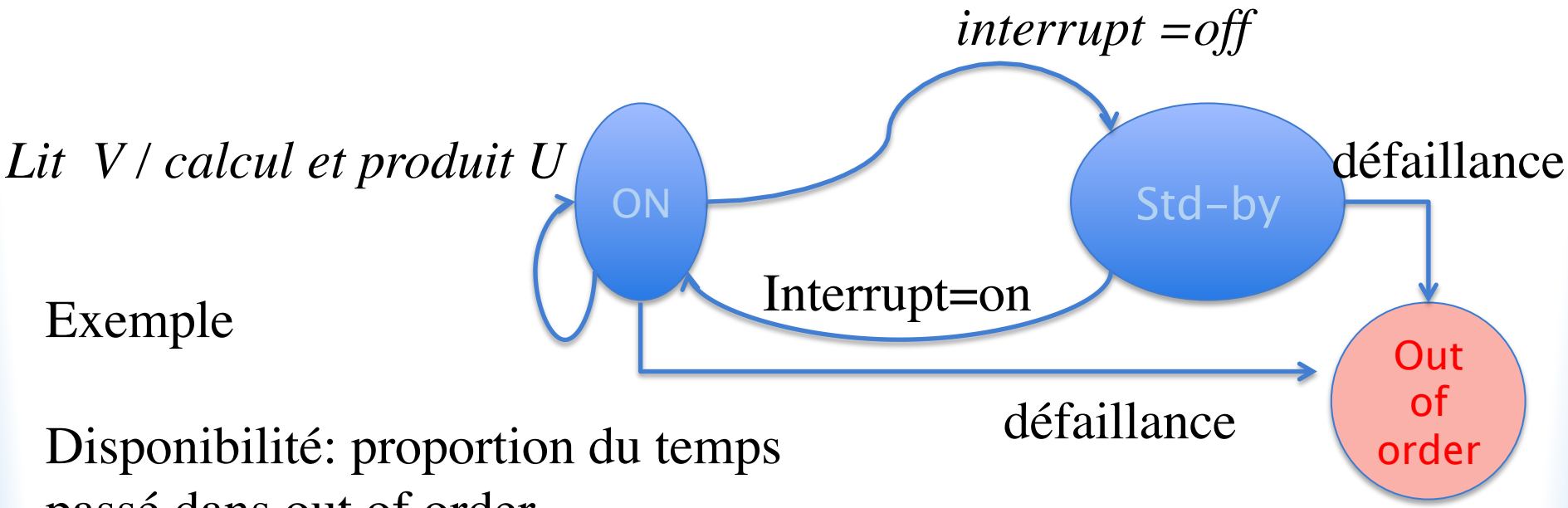
- Système : le boitier de contrôle de vitesse sur un chariot automatique
  - 2 états « opérationnels »: on, stand-by
  - 2 entrées : la consigne de vitesse (via une molette), et l'interrupteur (up/down) permettant d'activer la régulation
  - Une sortie : la tension à appliquer au moteur en fonction (relation de type quadratique U = ß*V^2
  - Observables : la vitesse effective du chariot, l'état opérationnel du boîtier
  - composants internes :
    - Alimentation (non réparable, non remplaçable)
    - Micro-contrôleur (remplaçable)

Attributs en pratique (2)



- V peut être modifiée dans n'importe lequel des deux états
- Lorsque l'interrupteur passe de up à down, le chariot décélère lentement animé d'une accélération nulle (frottement == arrêt)
- Défaillances :
  - La vitesse n'est pas régulée correctement
  - La vitesse ne peut être modifiée ou sa modification n'est pas prise en compte….

Attributs en pratique (2)



Exemple

Disponibilité: proportion du temps passé dans out of order

Fiabilité: probabilité instantanée de fournir la bonne tension en «on»

Intégrité: Identification des conditions permettant d'altérer la mémoire contenant la valeur U

Sécurité innocuité: Identifier la valeur de V représentant un danger

Maintenabilité: temps de remplacement du micro-contrôleur.

Définitions attributs

- Disponibilté : capacité du système à offrir tout ou partie du service (n'est pas « visiblement » défaillant)
- Fiabilité : caractérise la capacité du système à effectivement délivrer un service correct (lorsqu'il est disponible)
- Sécurité-innocuité (safety ou sûreté) caractérise la maitrise/connaissance des conséquences d'une défaillance

Définitions attributs

- Intégrité : caractérise la capacité du système à détecter ou empêcher toute altération non autorisée de sa structure ou de son état
- Maintenabilité : caractérise la capacité du système à faire évoluer sa structure ou son état pour faciliter le traitement des fautes ou le retour à un état fonctionnel depuis un état défaillant

Les moyens pour obtenir un niveau voulu de SdF

- 4 Approches complémentaires
  - **Prévention des fautes :**
    Méthodes destinées à empêcher l'occurrence même des fautes
  - **Elimination des fautes :**
    Méthodes de recherche et de suppression des fautes de conception et développement
  - **Prévision des fautes :**
    Analyse de la fréquence ou du nombre de fautes et de la gravité de leurs conséquences
  - **Tolérance aux fautes :**
    Mécanismes au sein du système destinés à assurer les propriétés de SdF voulues en présence des fautes

Plan du cours

- Intro Sûreté de fonctionnement (fait)
- Savoir faire élémentaire en TaF
- TaF dans le cadre temps réel
- Introduction au TP
- Bilan

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)

- PB 1 localisation de l'erreur / faute
- PB 2 traitement de la faute/erreur

Zone de confinement des erreurs

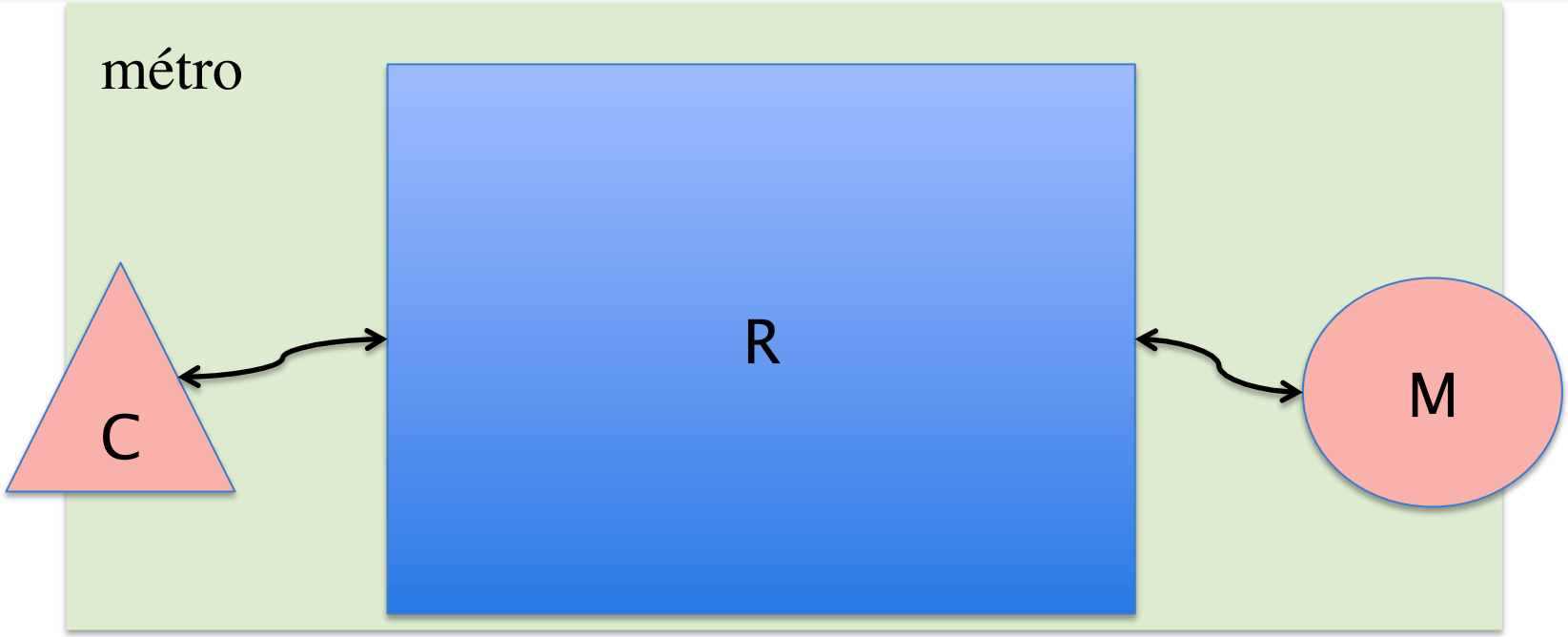
- **Définition**
  périmètre/interface d'un système muni de mécanismes de protection empêchant une erreur de se propager sans être au moins détecté et signalée
  (ie de contaminer l'environnement du système)
- En pratique :
  - Description architecturale définissant la décomposition du système en sous-systèmes 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)

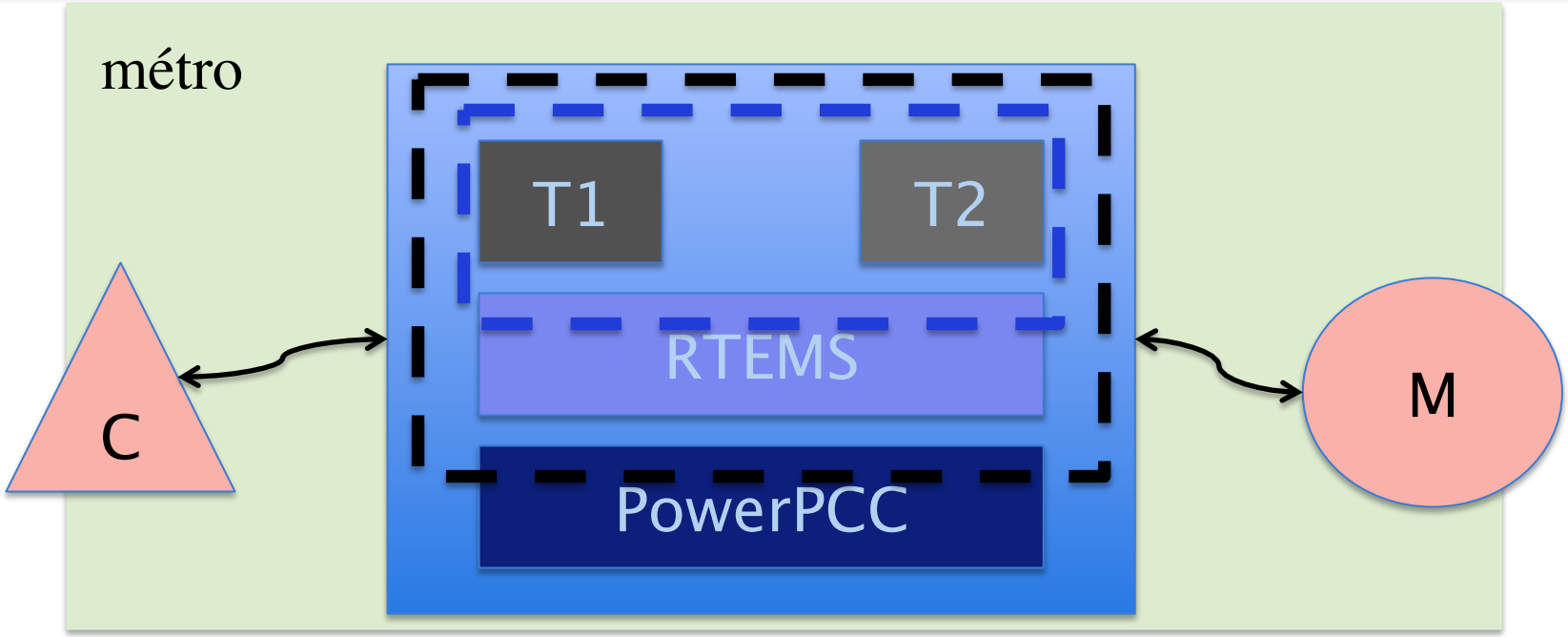
Défaillances, fautes et propagation des erreurs



- 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

- **Diagnostic**: 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

TaF logicielle

Génie logicielle et TaF

- Les activités:
  - Identification / classification des défaillances
  - Conception / mise en œuvre des zones de confinement
  - Vérification des comportement/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
- **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

- Défaillances concernées :
  tout ce qui ne compromet pas le matériel
  - Valeurs incorrecte sur les interfaces des fonctions
  - Comportement aberrant dans le « run-time »
  - Corruption de l'intégrités des donnée
- 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

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, (cf TP)

Confinement via les langages

- 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

Confinement par moniteur externe

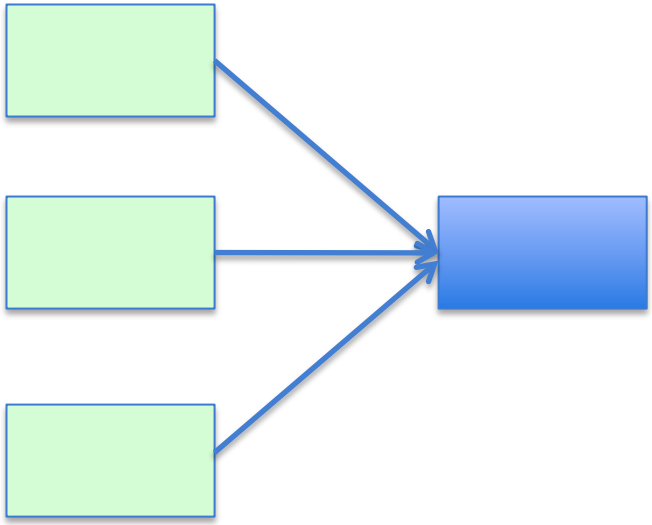
- Watchdog et interruption logicielle:
  - Défaillance constaté par l'absence de progrès dans l'exécution d'un programme
  - Causes possibles : boucle infinie, blocage sur un verrou pris et jamais restitué, récurrence trop longue …
  - Mise en œuvre : un contexte intègre pour réagir à l'expiration, une alarme, plus mécanisme de raz de l'alarme (cf cours suivant)

Recouvrement des erreurs 2 visions complémentaires

Corriger avant de poursuivre (réparable)

Recouvrement

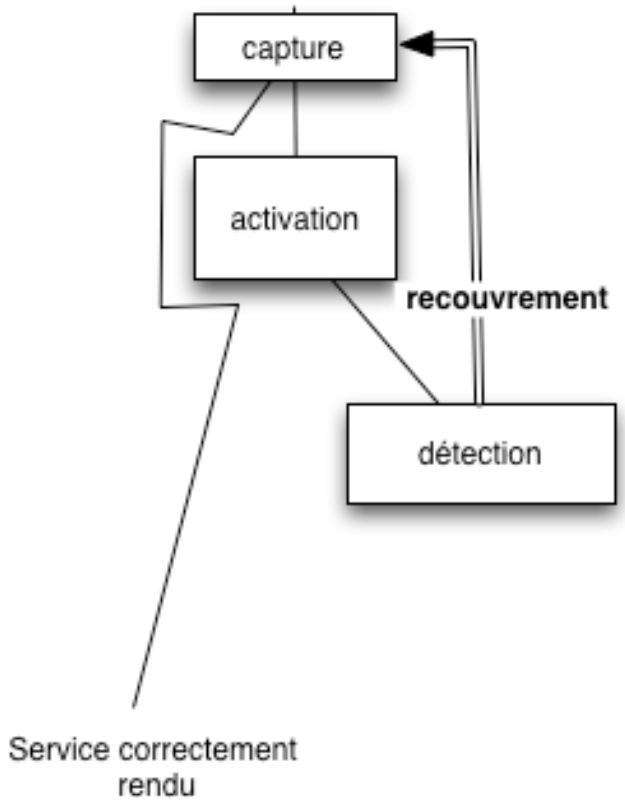
Choisir pour poursuivre (pas réparable)

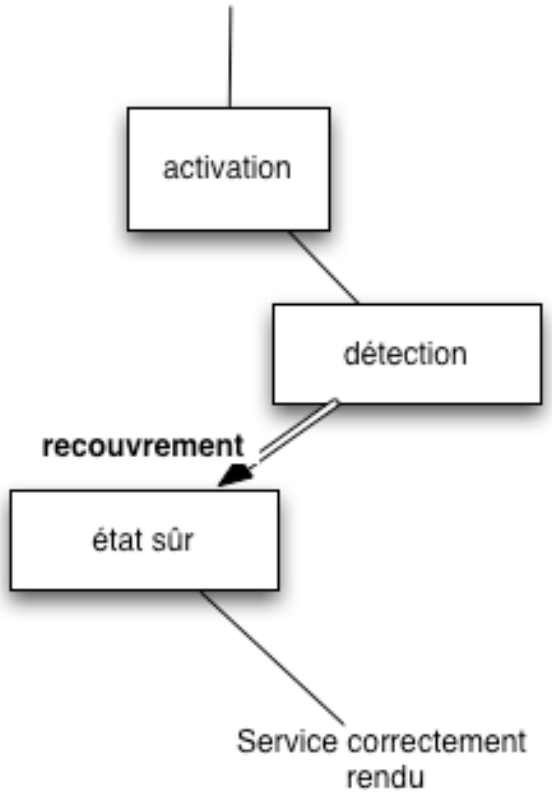


Masquage

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

- Le recouvrement ne résout pas les fautes activées de manière déterministe
- Les tests de vraisemblances sont parfois peu efficaces
- « l'indépendance » des activations :

  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 :

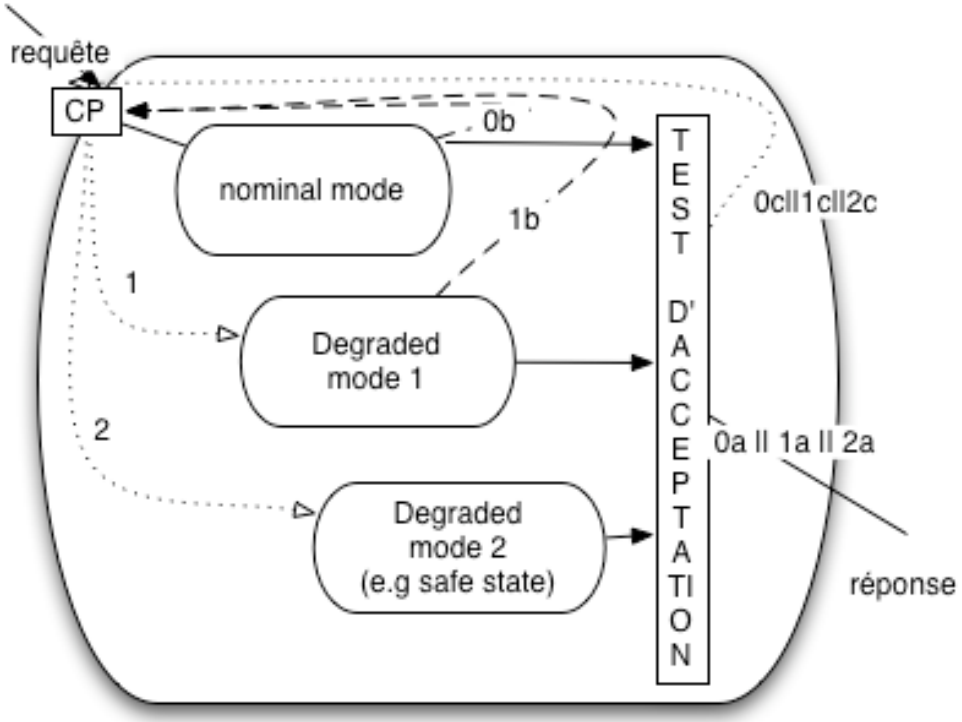
⇒ 2 algorithmes pour la calculer avec une sensibilité différente aux erreurs numériques

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

Principe de la capture d'état

- 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%

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 (attention au data race!!)
  - 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 ?

TaF matérielle

Intérêt de la redondance matérielle

- Une erreur dans le logiciel peut se propager au matériel puis à un autre logiciel
- Idée : donner à chaque unité « logicielle » indépendante son propre matériel
- Le système est 1 système distribué :
  - 1 ensemble de calculateurs communicant par messages
  - Granularité minimale : la machine

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

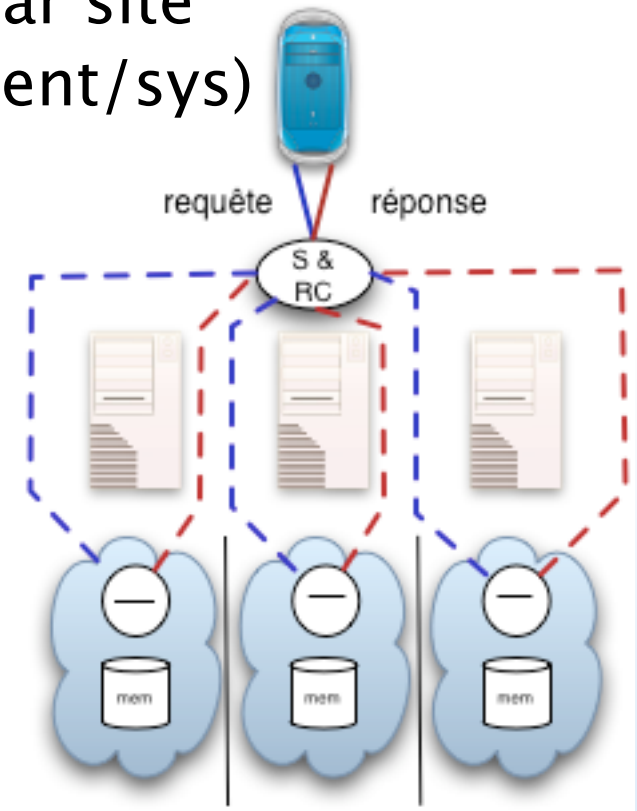
Stratégies de réplication mise en œuvre de la redondance

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

Comment et avec quelles ressources ?

Réplication Active

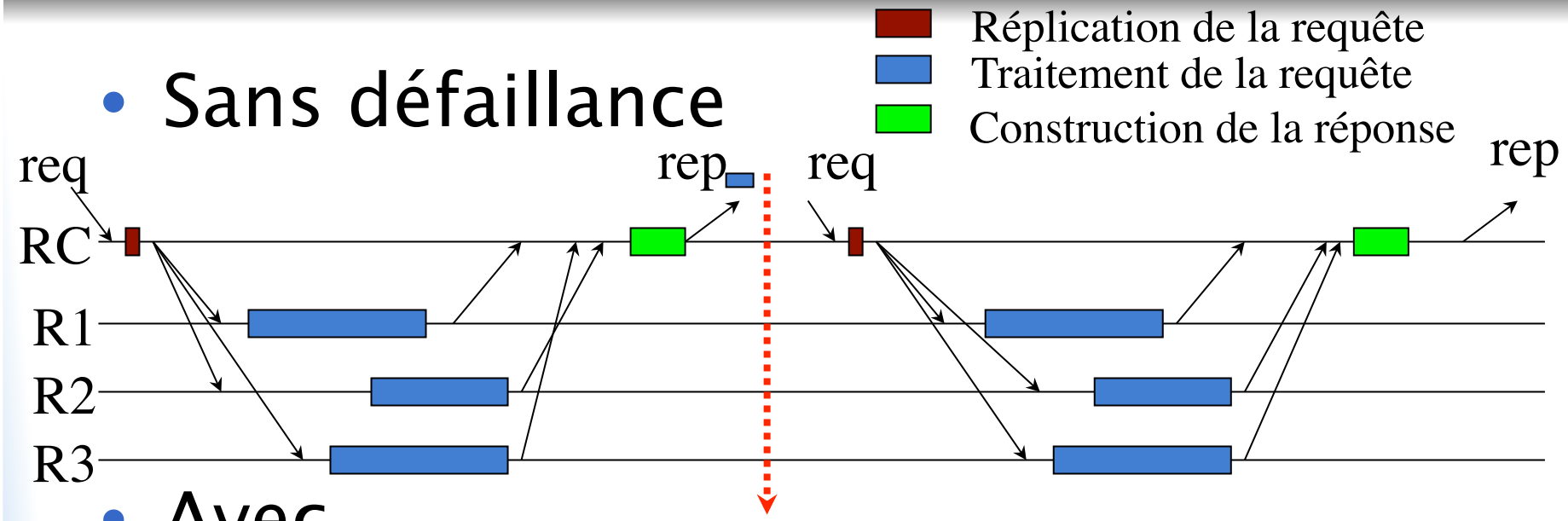
- N calculateur => 1 application par site
- 1 contrôleur pour interpréter (client/sys)
- Décision: vote
- Communications :
  diffusion des requête +
  agrément sur le résultat
- Déroulement
  1) Envoyer la requête à tous
  2) Chaque site exécute son service
  3) Construction de la réponse par vote majoritaire



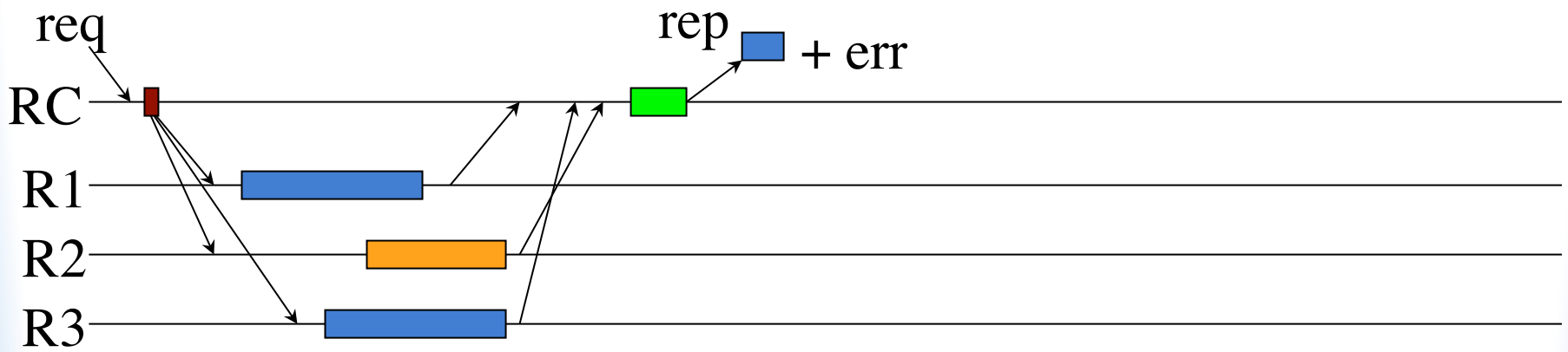
S & RC : vote et contrôle des répliques

Communication sans et avec défaillance

- Sans défaillance

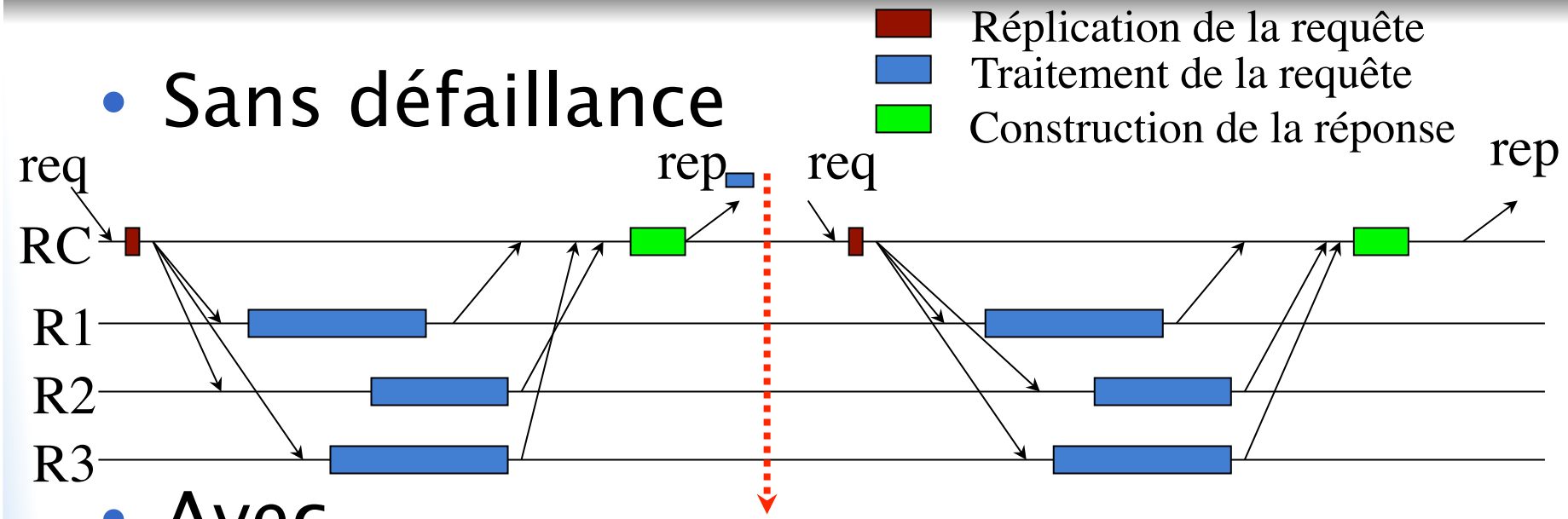


- Avec

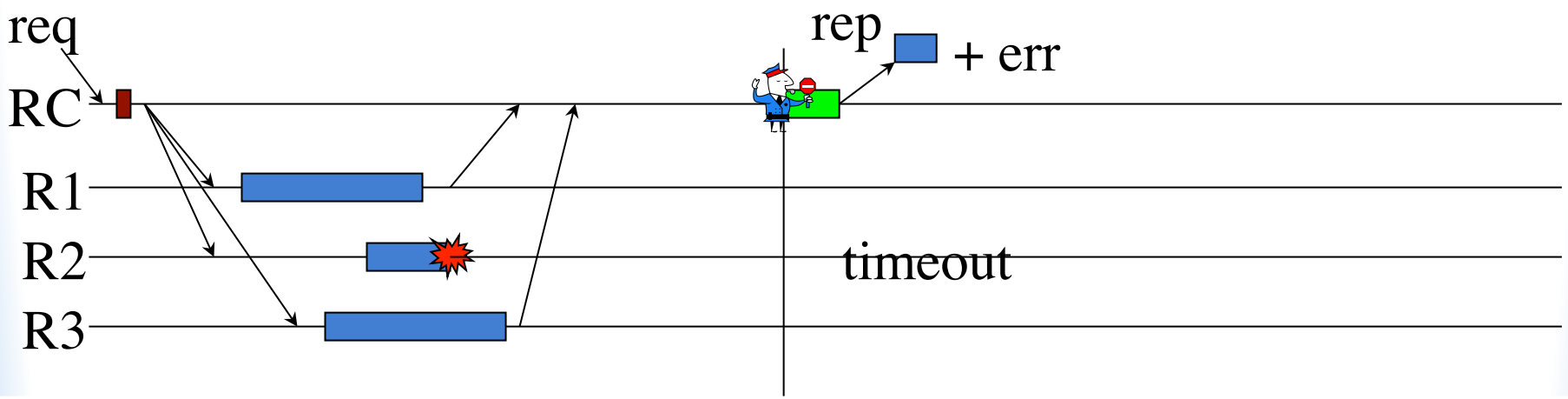


Communications sans et avec défaillance

- Sans défaillance



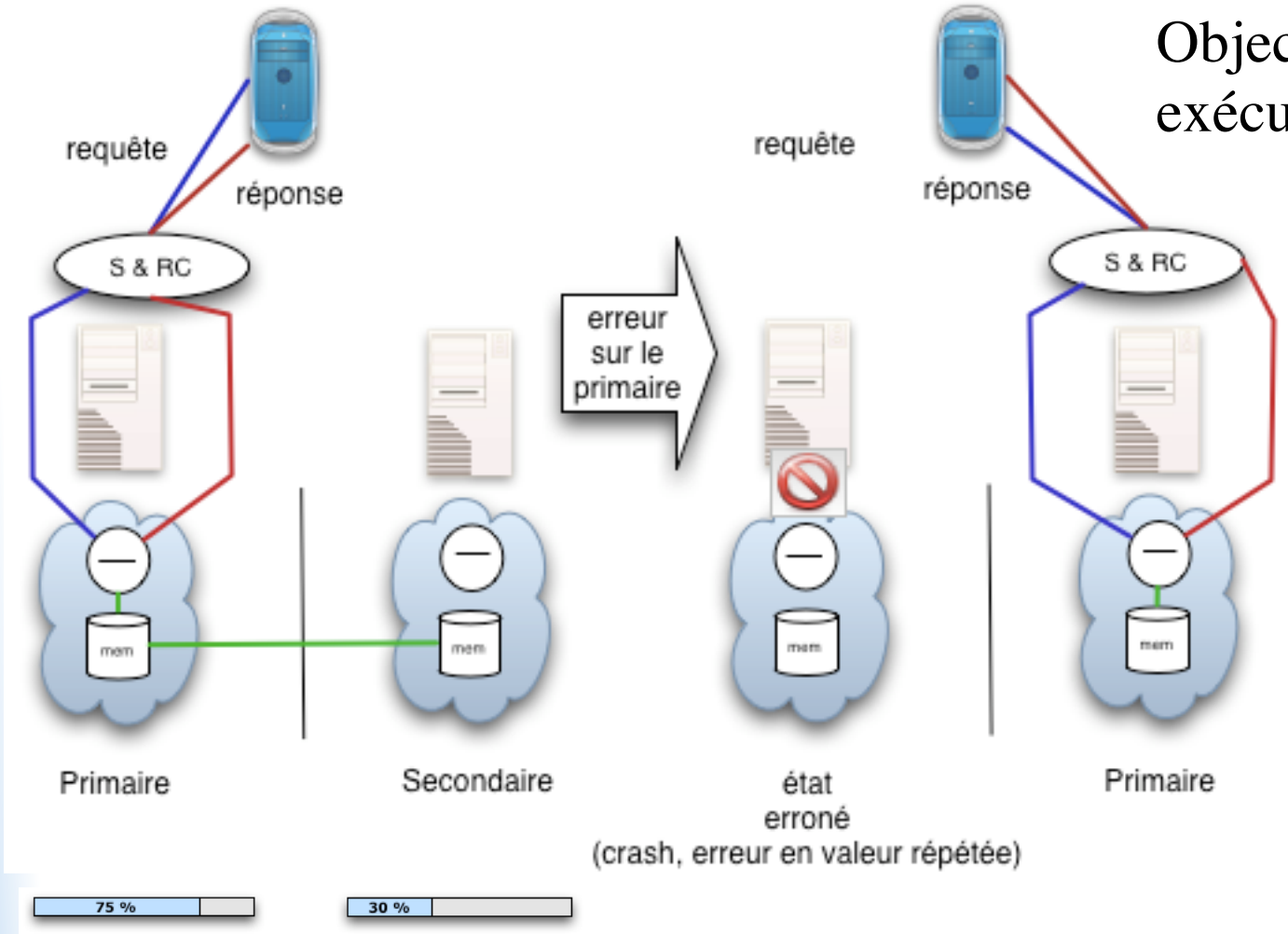
- Avec



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

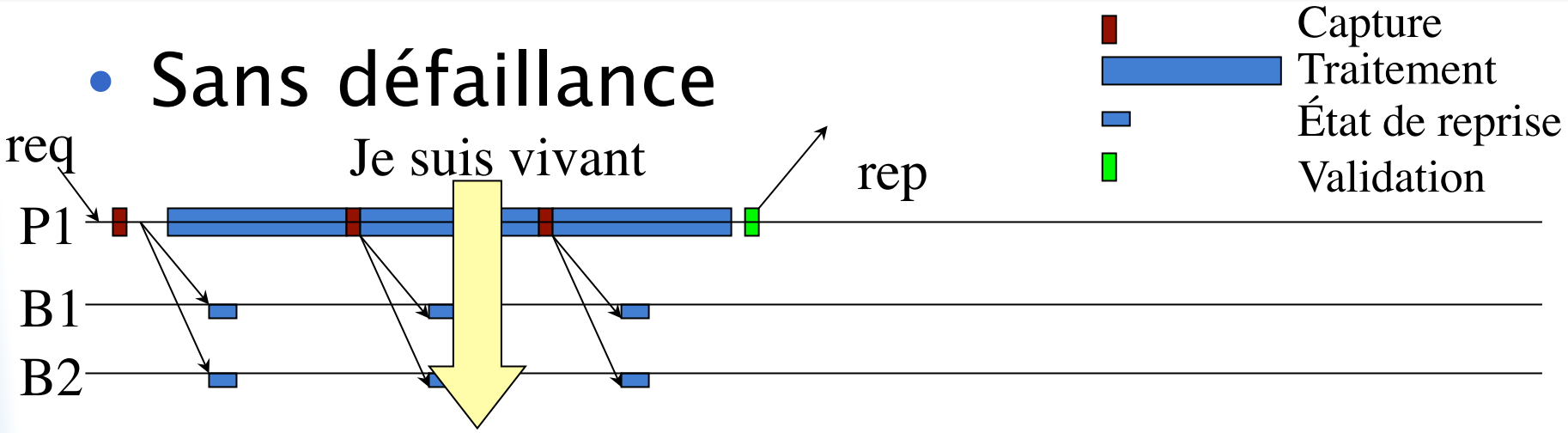
1 seule exécution à la fois

Communication : Transfert d'un état

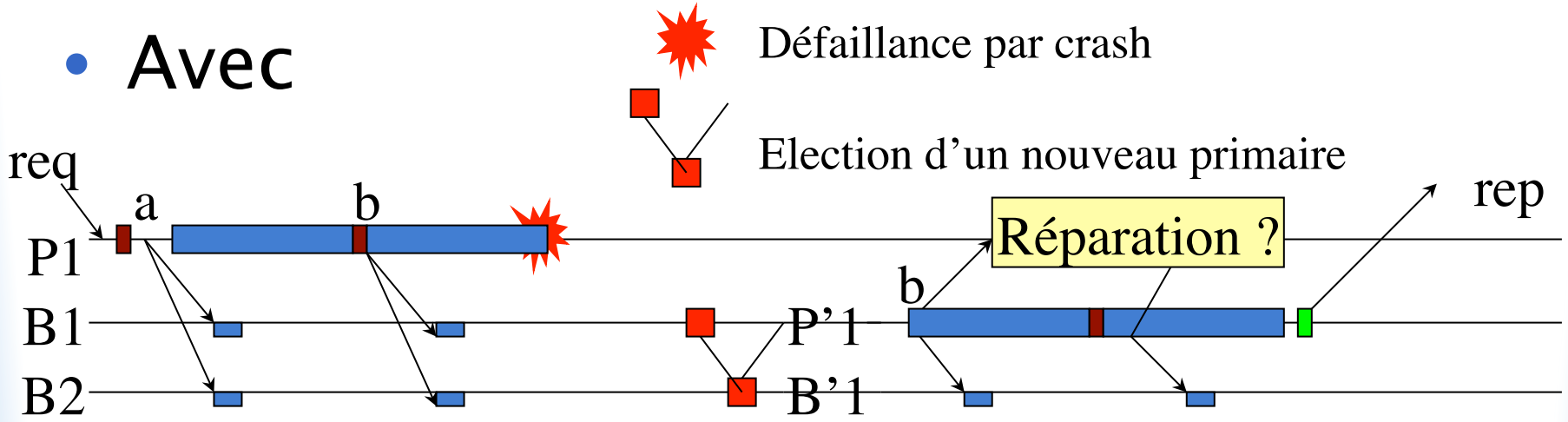
Le primaire doit capturer son état pour l'envoyer au secondaire

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.

Et pour le temps réel ?

- Les méthodes vues jusqu'à présent :
  « asynchrones » ou « ultra synchrones »

⇒Synchronisation forte sur des événement ≠ T

- PB : RT => synchronisation
  - faible par rapport aux événements,
  - forte par rapport au temps
- Le dimensionnement du système est critique

(dur de maintenir une bas de temps précise à grande échelle)

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

Permet de définir des compromis!!!

Application N-version programming au temps réel

- Détection d'erreur = détection de dépassement de budget
- Les N versions == programme de plus en plus simples (WCET de plus en plus petit et sur !!!)
- Question : fait on du temps réel dur **?**

OUI

Tolérance aux fautes & temps réel

- Le temps réel possède un modèle fortement contraint : le lot de tâches
- Prise en compte de l'exceptionnel

  le pire cas devient plus riche -> WCET erroné

  + tous les cas classiques vus jusqu'à présent
- Détection de comportements temporels erronés +
  - Architectures de masquage sans délai (TMR) si ordonnançable
  - Déclenchement d'un mode de fonctionnement dégradé (délestage ou reconfiguration fonctionnelle).

Surdimensionnement et modes dégradés

- Pour assurer les bornes temporelles :
  - surdimensionnement (élimination des fautes) ***prévoir les reprises***
  - Mode dégradés temporellement prédictibles (tolérance par recouvrement)

  => Le couplage du surdimensionnement et des modes opérationnels laisse une marge pour passer d'un lot de tâche TR défaillant à un autre lot de tâches TR moins complet
- Etude des lois d'occurrence des fautes requis pour analyse de disponibilité

Notion de système intégré TR et TaF

- Un support d'exécution logiciel &/ou matériel pour les deux aspects
- Idée : borner le temps du processus détection / rétablissement
- Exemples : MARS, Delta-4, ROAFTS …
- Acceptation des fautes => prévoir le pire amène un compromis ≠ temps réel souple
- Compromis == sélectionner des services moins critiques et sacrifiables.

Ordonnancement à criticité mixte

- Principe N modes de fonctionnement
  - Mode i : temps d'exécution = confiance de niveau i
  - Objectif d'ordonnancement = f(i)
- Enjeux et opportunités :
  - Permet de définir des objectifs (et donc un coût sur mesure / modes)
  - Optimiser disponibilité / prédictibilité / dimensionnement

Le modèle Discard

- 2 modes : LO / HI
- 2 types de taches CR (critique), NC non critique
- Objectifs :
  - Exécuter CR NC
  - Exécuter CR (NC non requises)
- Hypothèses :
  - Mode X Exec Time ≤ WCET(X)
  - WCET (LO) << WCET(HI)

Exemple en priorité fixe

- Tâches :

Nom	Période	Budget LO	Budget HI	Type
τ1	5	1	2	CR
τ2	20	2	5	CR
τ3	10	1	1	NC
τ4	20	6	2	NC

- U(LO) = 1/5 + 2/20 + 1/10+ 6/20=14/20
- U(HI) = 13/20
- U(max) > 100%

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 »
- Cohabitation TR / TaF
  - Tous les deux visent à contrôler le flux d'exécution
  - TaF a un impact sur l'ordonnancement

  => utiliser les modes de fonctionnement et les modes dégradés

Conclusion (suite)

- Cohabitation TR / TaF (suite)

  Fixer la probabilité des modes dégradés est strictement différents de fixer la probabilité du respect des échéances (les échéances sont dures)

- Toujours viser le plus simple

  le mécanisme peut lui même entraîner des défaillances parfois pire que celle que l'on chercher à tolérer

Acronymes

- SdF : sûreté de fonctionnement
- TaF : Tolérance aux Fautes
- TMR, NMR : triple/ N – moduar replication
- RB : recovery blocks ou Rollback…
- TR : temps-réel
- 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 : 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.