Tolérance aux fautes - 1
Introduction, techniques de base

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

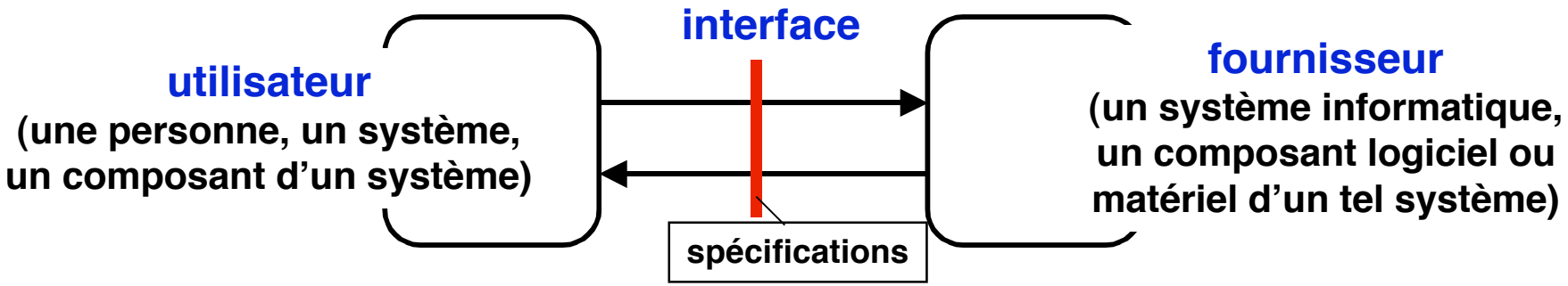
Plan

- Objectifs, définition, problèmes de la tolérance aux fautes
- Principales méthodes
  - Recouvrement
  - Compensation
- Exemples d'application : machines à haute disponibilité
- Réalisation de serveurs à haute disponibilité
  - Serveur primaire - serveur de secours
  - Redondance active
- Tolérance aux fautes pour les données

Définitions de base

Service

- **Ensemble de fonctions défini par une interface, “contrat” entre le fournisseur et l’utilisateur du service.**



Propriétés attendues d’un service

Propriétés “fonctionnelles” (définies dans les spécifications d’interface)

validité (“*correctness*”) : le système est conforme à ses spécifications

propriétés de sûreté et de vivacité

Propriétés “non fonctionnelles” (les autres)

performances ; sûreté de fonctionnement

La distinction n’est pas toujours évidente

Sûreté de fonctionnement (*dependability*)

- **Propriété d'un système informatique permettant à ses utilisateurs de placer une confiance justifiée dans le service que délivre le système**

Différents aspects

domaine de la tolérance aux fautes

Fiabilité (*reliability*) : le système est en état (continu) de rendre le service

Mesure : probabilité (fonction du temps t) que le système ne soit pas défaillant entre le temps 0 et le temps t

Disponibilité (*availability*) : le service est disponible en permanence

Mesure : fraction du temps (sur une période déterminée) durant laquelle le système fournit le service

Sécurité (au sens *safety*) : un mauvais fonctionnement du système n'a pas d'incidence catastrophique sur son environnement

Il faut définir "catastrophique" et "environnement"

Sécurité (au sens *security*) : le système assure la confidentialité et l'intégrité des informations ainsi que le contrôle de l'accès au service

le service est effectivement accessible aux utilisateurs autorisés, non aux autres

Sûreté de fonctionnement

- **Il n'y a pas de critère absolu**
  - L'importance relative des critères dépend
    - de la nature de l'application
    - des exigences des utilisateurs
    - des conditions d'utilisation (environnement, etc.)
- **Exemples**
  - Système embarqué : fiabilité, disponibilité
  - Système de communication (ex : commutateur téléphonique) : disponibilité
  - Service de fichiers, base de données : disponibilité, sécurité (*security*)
  - Système de transport (exemple : navigation, guidage, freinage) : sécurité (*safety*), disponibilité

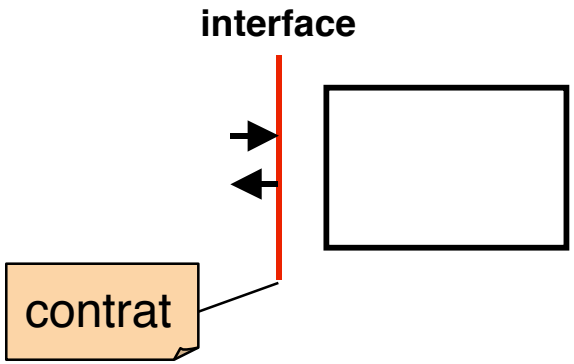
Défaillances

Définition

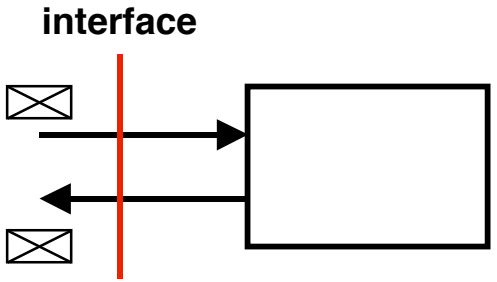
- Un système (ou composant) est sujet à une **défaillance** (***failure***) lorsque son comportement n'est pas conforme à sa spécification
  - Synonyme de défaillance : panne

Remarques

- Le système ou composant est considéré comme une "boîte noire" ; on ne regarde que son comportement global, observé à l'interface
- On peut définir différents "degrés" de gravité de défaillance en fonction de leur impact sur la sûreté de fonctionnement (cf plus loin)



Degré de gravité des défaillances (1)



Modèle

- boîte noire, messages entrants et sortants

Panne franche

- Dit aussi : arrêt sur défaillance (*fail stop*)
  - ou bien le système fonctionne, et donne un résultat correct
  - ou bien il est en panne (défaillant), et ne fait rien
- C'est le cas le plus simple, et on essaie de s'y ramener (au besoin en forçant l'arrêt d'un composant dès qu'une erreur y a été détectée : technique *fail fast*)

Panne par omission

- Le système perd des messages entrants (omission en réception), sortants (omission en émission), ou les deux. Il n'y a pas d'autres déviations par rapport aux spécifications
- Ce modèle peut servir à représenter des défaillances du réseau
- Plus difficile à traiter que la panne franche

Degré de gravité des défaillances (2)

- **Pannes de temporisation**
  - Les déviations par rapport aux spécifications concernent uniquement le temps (par exemple temps de réaction à un événement)
- **Pannes arbitraires (ou “byzantines”)**
  - Le système peut faire “n’importe quoi” (y compris avoir un comportement malveillant)
  - Intérêt théorique : conditions les plus défavorables
  - Hypothèse parfois nécessaire pour des systèmes à très haute fiabilité dans un environnement hostile (nucléaire, spatial)
  - Traitable, mais nécessite une redondance élevée (typiquement $3k+1$ exemplaires d’un composant pour résister à k défaillances)

Mesures de fiabilité et disponibilité (1)

Mesure de la fiabilité

- Probabilité R(t) que le système ne soit pas défaillant entre 0 et t
- Temps moyen jusqu'à la prochaine panne : E(R(t)) = MTTF (*Mean Time To Failure*)
  E = espérance mathématique (moyenne pondérée par la probabilité)

Mesure de la disponibilité

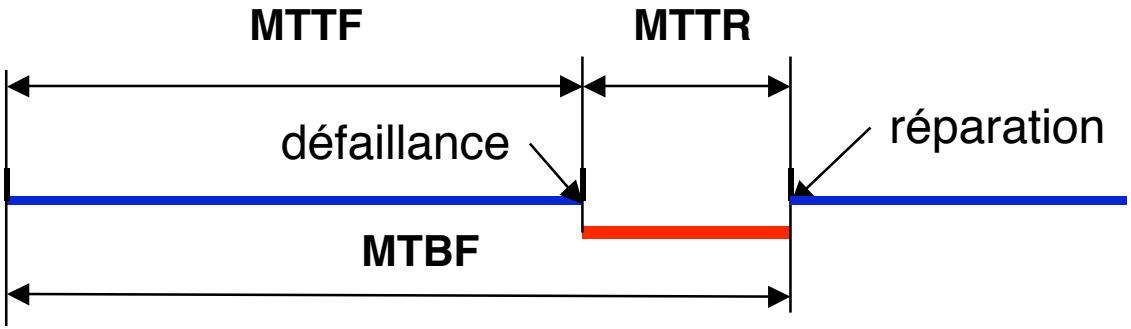
- Disponibilité instantanée : Probabilité A(t) que le système soit disponible (fournisse un service correct) à l'instant t
- Disponibilité moyenne a = E(A(t)) : fraction moyenne du temps où le système est disponible (sur une période donnée)

Réparation

- Réparer un système en panne : le remettre en état de rendre un service correct
- Mesure : temps moyen de réparation : MTTR (*Mean Time To Repair*)

Mesures de fiabilité et disponibilité (2)

On utilise aussi MTBF (*Mean Time Between Failures)*



Mesure de la disponibilité

Dans le cas de pannes franches : **disponibilité = a = MTTF/(MTTF + MTTR)**
Mesurée en “nombre de 9” :

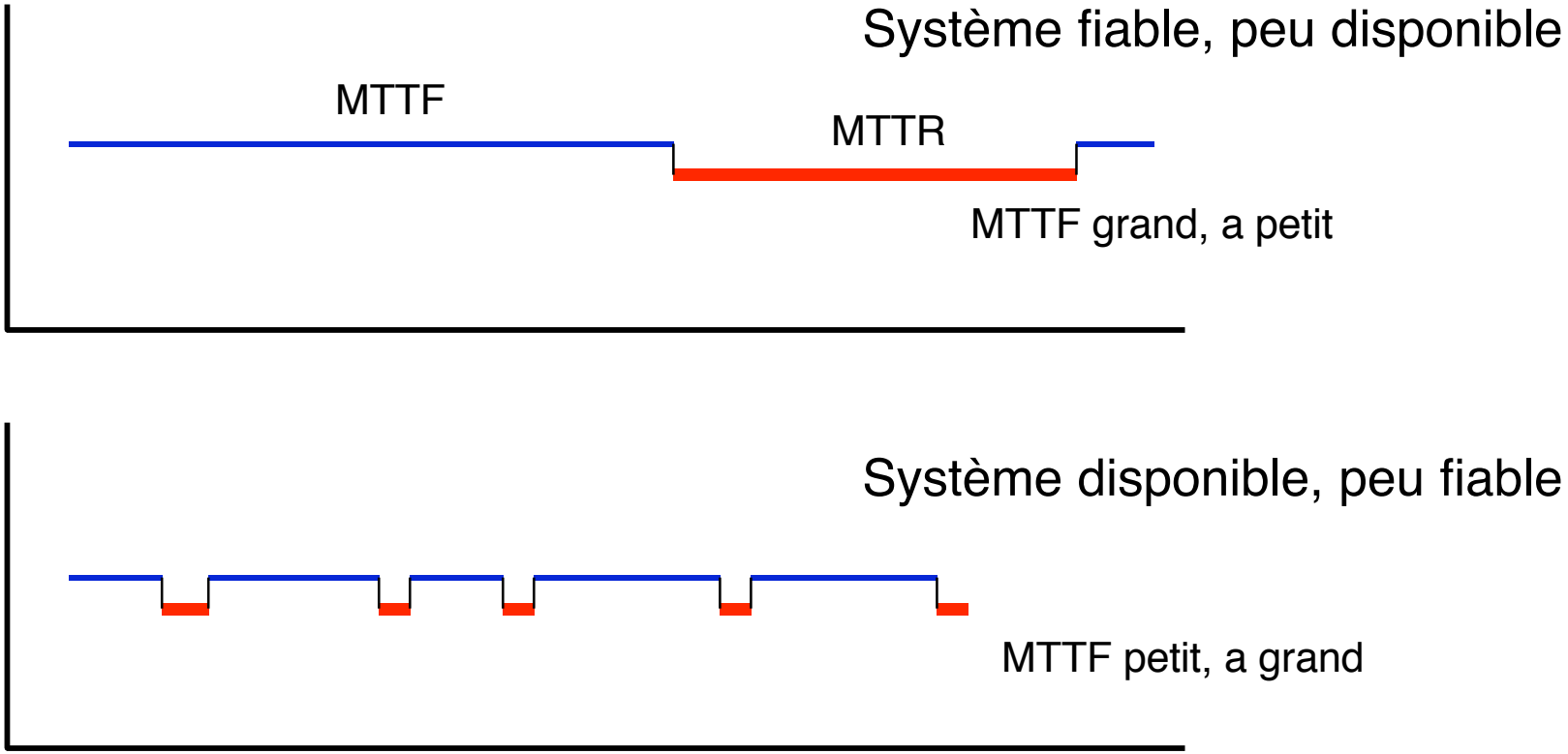
99.99%	(4 nines)	indisponible 50 minutes/an
99.999%	(5 nines)	indisponible 5 minutes/an

Indisponibilité = 1 — disponibilité = MTTR/(MTTF + MTTR) ≅ MTTR/MTTF, car en général MTTR << MTTF

Donc diviser MTTR par 2 a le même effet sur la disponibilité que doubler MTTF

Mesures de fiabilité et disponibilité (3)

Bien comprendre la différence entre fiabilité (mesurée par MTTF) et disponibilité (mesurée par a = MTTF / (MTTF + MTTR))



Analyse des défaillances (1)

- **Principe**
  - On s'intéresse à l'origine d'une défaillance (donc on doit ouvrir la "boîte noire" que constitue le système)
- **Définitions**
  - Erreur : état (ou partie de l'état) du système susceptible de provoquer une défaillance (partie de l'état interne dont les propriétés courantes ne sont pas conformes aux spécifications)
    - exemple logiciel : la valeur d'une table ne vérifie pas un invariant spécifié
    - exemple matériel : une connexion est coupée entre deux points qui devraient être reliés entre eux
  - Faute : toute cause (événement, action, circonstance) pouvant provoquer une erreur
    - exemple : faute de programmation, malveillance, catastrophe naturelle, etc.

Analyse des défaillances (2)

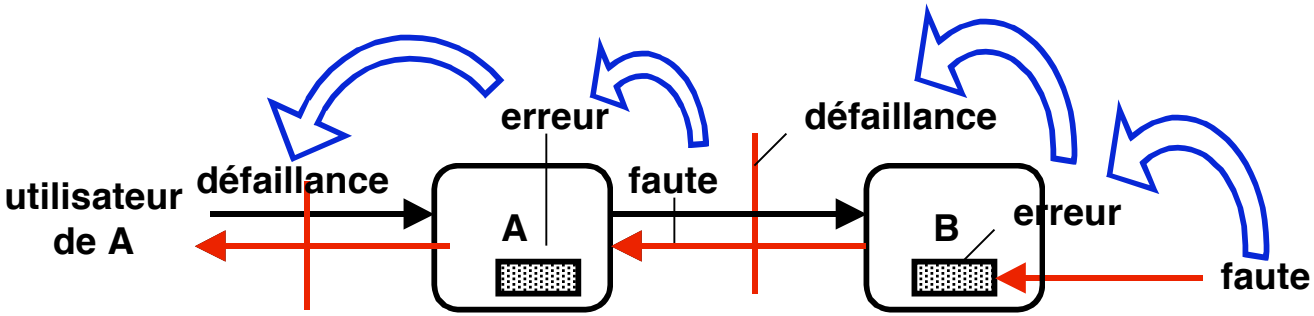
De l'erreur à la défaillance

- Une erreur est susceptible de provoquer une défaillance, mais ne la provoque pas nécessairement (ou pas immédiatement)
  - parcequ'il y a une redondance interne suffisante pour que le système continue de fournir le service
  - parceque la partie erronée de l'état n'est pas utilisée pour telle ou telle fonction
- Une erreur est latente tant qu'elle n'a pas provoqué de défaillance
- Le temps entre l'apparition de l'état d'erreur et la défaillance est le délai de latence
  - plus le délai de latence est long, plus la recherche des causes d'une défaillance est difficile

Analyse des défaillances (3)

Propagation entre composants (ou sous-systèmes)

- Un composant A **utilise** un composant B si le bon fonctionnement de A dépend de celui de B



Pour A, la **défaillance** de B (service incorrect) constitue une **faute**, qui peut à son tour provoquer une **erreur** interne à A, puis une défaillance de A

faute → erreur → défaillance → faute → erreur → …

Degrés de permanence des défaillances

- **Les défaillances n'ont pas un comportement uniforme dans le temps**
  - **Défaillance transitoire**
    - **se produit de manière isolée**
  - **Défaillance intermittente**
    - **se reproduit sporadiquement**
  - **Défaillance permanente**
    - **persiste indéfiniment (jusqu'à réparation) après son occurrence**
  - **Les défaillances non permanentes sont difficiles à modéliser et à traiter**

Comment assurer la sûreté de fonctionnement

Évitement des fautes : vise à empêcher l'occurrence de fautes

- **Par la prévention**
  - Analyser les causes potentielles de fautes
  - Prendre des mesures pour les éliminer (pas toujours possible) ou réduire leur probabilité
- **Par l'évaluation**
  - Prévoir les fautes (et les mesures pour y faire face)
  - Prévision souvent statistique
- **Par la vérification**
  - Avant mise en route du système : examiner les fautes restantes, et éliminer celles que l'on peut éliminer

Tolérance aux fautes : vise à préserver le service malgré l'occurrence de fautes

- **Par la redondance**
  - du matériel, des traitements, des données

Tolérance aux fautes - motivations

Constatation de base

- Quelles que soient les précautions prises, l'occurrence de fautes est inévitable (erreur humaine, malveillance, vieillissement du matériel, catastrophe naturelle, etc.). Cela ne veut pas dire qu'il ne faut pas essayer de prévenir ou d'éliminer les fautes, mais les mesures prises peuvent seulement réduire la probabilité de leur occurrence
- Il faut donc concevoir les systèmes de manière à ce qu'ils continuent à rendre le service attendu (éventuellement un service dégradé) même en présence de fautes

Remarque très importante …

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

Tolérance aux fautes - étapes

Plusieurs phases successives, non obligatoirement toutes présentes

Détection

Découvrir l'existence d'une erreur (état incorrect) ou d'une défaillance (comportement incorrect)

Localisation

Identifier le point précis (dans l'espace et le temps) où l'erreur (ou la défaillance) est apparue

Isolation

Confiner l'erreur pour éviter sa propagation à d'autres parties du système

Réparation

Remettre du système en état de fournir un service correct

Tolérance aux fautes - techniques

- **Deux classes de techniques**
  - Traiter les erreurs
    - détecter l'existence d'un état incorrect (erreur)
    - remplacer l'état incorrect par un état correct (conforme aux spécifications)
  - Traiter les fautes
    - prévoir des composants multiples, pour réduire la probabilité qu'une faute conduise à une défaillance
    - réaliser des traitements multiples
      - (par exemple : réaliser la même opération par des algorithmes différents pour tenter d'éviter les fautes de conception)
- **Dans tous les cas, un principe unique : la redondance**
  - redondance d'information (détection d'erreur)
  - redondance temporelle (traitements multiples)
  - redondance matérielle (composants dupliqués)

Traitement des erreurs

Deux techniques de base

- **Recouvrement (*error recovery*)**
  - Remplacer l'état d'erreur par un état correct
  - Nécessite détection de l'erreur (identification de la partie incorrecte de l'état), au moyen d'un test de vraisemblance
    - explicite (exprimer et vérifier des propriétés spécifiques de l'état)
    - implicite (via des conséquences visibles : violation d'un délai de garde, violation de protection de mémoire, etc.)
  - Deux techniques de recouvrement : reprise et poursuite
- **Compensation (*error masking*)**
  - L'état posède une redondance interne suffisante pour détecter et corriger l'erreur (les tests externes de vraisemblance sont inutiles)

Détection d'erreur (1)

Objectifs

- prévenir (si possible) l'occurrence d'une défaillance provoquée par l'erreur
- éviter la propagation de l'erreur à d'autres composants
- faciliter l'identification ultérieure de la faute (en vue de son élimination ou de sa prévention

Paramètres

- latence : délai entre production et détection de l'erreur
- taux de couverture : pourcentage d'erreurs détectées

Détection d'erreur (2)

- **Techniques**
  - Comparaison des résultats de composants dupliqués
    - coût élevé
    - latence faible
    - taux de couverture élevé
    - nécessité d'indépendance des conditions de création et d'activation des fautes (cf Ariane 501)
  - Contrôle de vraisemblance
    - coût modéré
    - latence variable selon la technique
    - taux de couverture souvent faible

Illustration des techniques de traitement des erreurs

Recouvrement

- Détection explicite d'erreur ; remplacement de l'état d'erreur par un état correct
  - Reprise (à partir d'un état ancien enregistré)
  - Poursuite (reconstitution d'un l'état courant correct)
- Exemples
  - Tandem Non-Stop
  - Points de reprise dans un système réparti

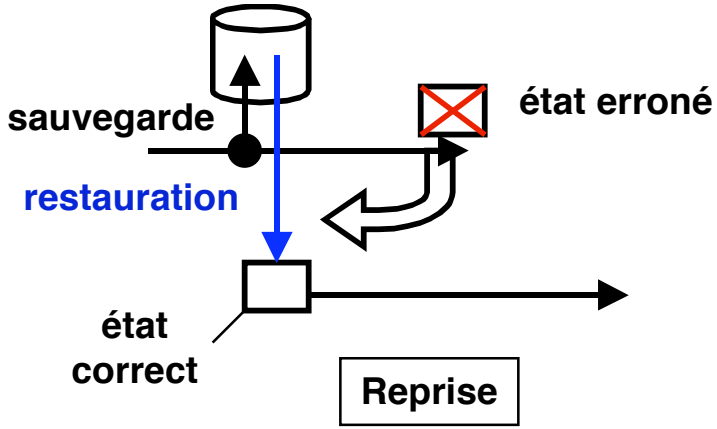
Compensation

- Redondance suffisante pour masquer (rendre invisibles aux utilisateurs) les conséquences d'une erreur
- Exemples
  - Vote majoritaire
  - Tandem Integrity
  - Stratus S/32

Recouvrement (1)

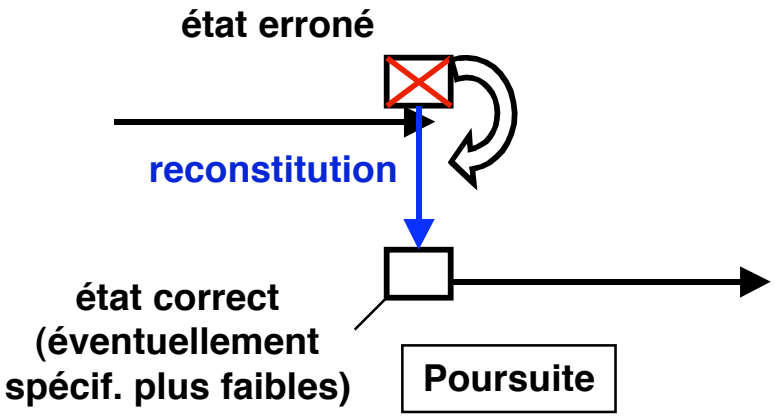
Reprise *(backward recovery)*

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



Poursuite *(forward recovery)*

- Tentative de reconstitution d'un état correct, sans retour arrière
- La reconstitution est souvent seulement partielle, d'où service dégradé
- Technique spécifique, la reconstitution dépend de l'application, et nécessite une analyse préalable des types d'erreurs possibles



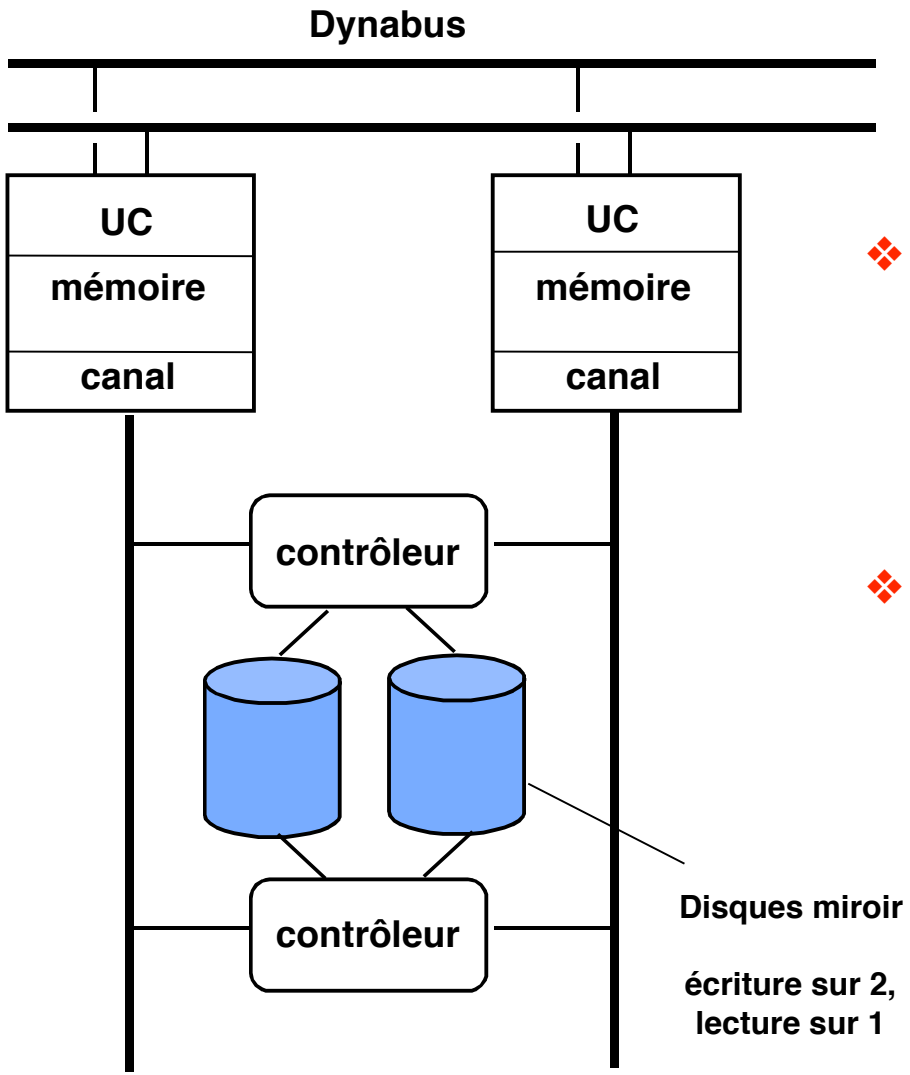
Recouvrement (2)

- **Un exemple**
  - Communication par paquets : comment réagir à la perte d'un paquet ?
- **Solution 1 : reprise**
  - L'émetteur conserve une copie des paquets qu'il a envoyés, jusqu'à réception de l'acquittement
  - En cas de détection de perte (par délai de garde), le récepteur demande à l'émetteur de renvoyer le paquet manquant
- **Solution 1 : poursuite**
  - Pour un type particulier d'application, l'émetteur peut reconstituer (totalement ou partiellement) le contenu d'un paquet manquant en utilisant les paquets voisins (suivants, précédents)
  - Le récepteur peut alors poursuivre sans interroger l'émetteur
- **Comparaison**
  - Reprise : technique générique, peut être coûteuse en temps et place ; c'est la plus utilisée
  - Poursuite : technique spécifique, peut être efficace, mais d'application limitée
  - Dans la suite, nous ne traitons que de la reprise

Recouvrement : techniques de reprise

- **Reprise**
  - le système est ramené à un état qu'il occupait avant occurrence de l'erreur (retour arrière)
  - cet état doit avoir préalablement été sauvegardé (points de reprise)
- **Problèmes de la reprise**
  - sauvegarde et restauration doivent être atomiques (pas d'interférences)
  - l'état des points de reprise doit lui-même être protégé contre les fautes
  - dans un système réparti :
    - les points de reprise sont créés pour chaque processus
    - l'ensemble de ces points de reprise doit constituer un état global cohérent du système

Exemple de recouvrement par reprise:
Tandem Non-Stop



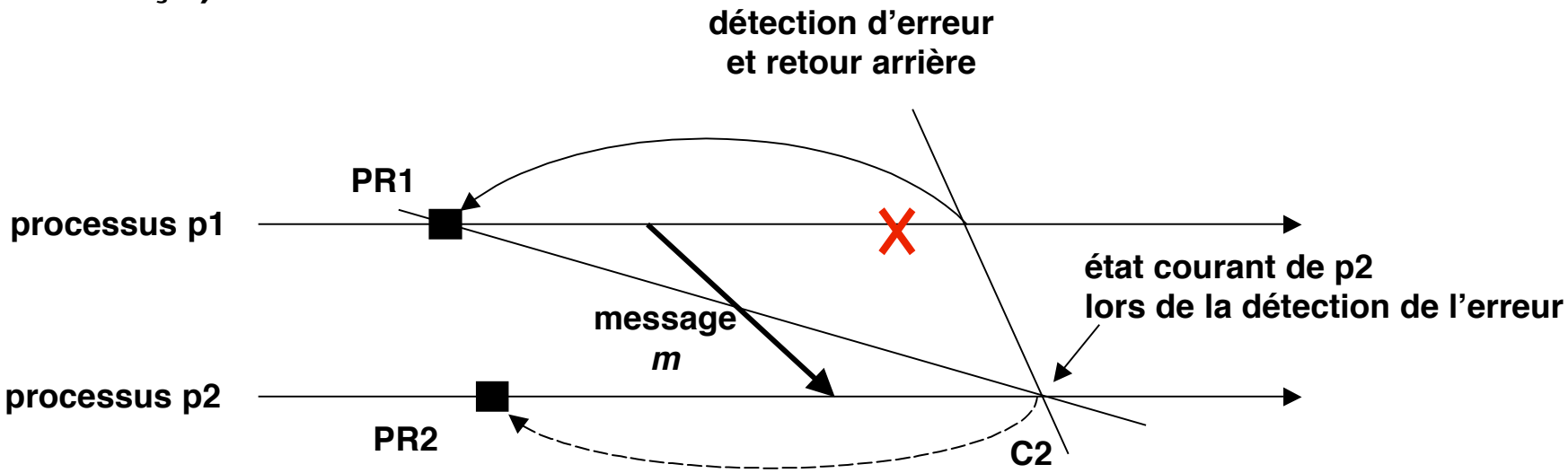
- **Matériel**
  - l'alimentation électrique est également redondante (deux sources indépendantes)
  - composants à détection d'erreur par contrôle de vraisemblance
  - une détection d'erreur bloque immédiatement le composant (*fail fast*)
- **Logiciel**
  - Paires de processus (processus actif - processus de secours, cf plus loin)
  - Nécessite système d'exploitation spécifique (Unix modifié pour gestion des paires de processus)

Tolère une faute unique d'un composant (UC, bus, mémoire, contrôleur, disque)

idem pour autres périphériques (bandes, terminaux)

Points de reprise d'un système réparti:
cohérence d'un état global

- La cohérence de l'état global est liée aux communications
- La causalité doit être respectée (un message est émis avant d'être reçu)



L'état global (PR1, C2) est incohérent (*m* noté comme reçu et non émis)
Le processus p2 (non défaillant) est "orphelin" et doit revenir en PR2
Avec un mauvais choix des points de reprise, le retour en arrière peut se propager jusqu'à l'état initial (effet domino)

Reprise :
comment constituer un état de reprise cohérent ?

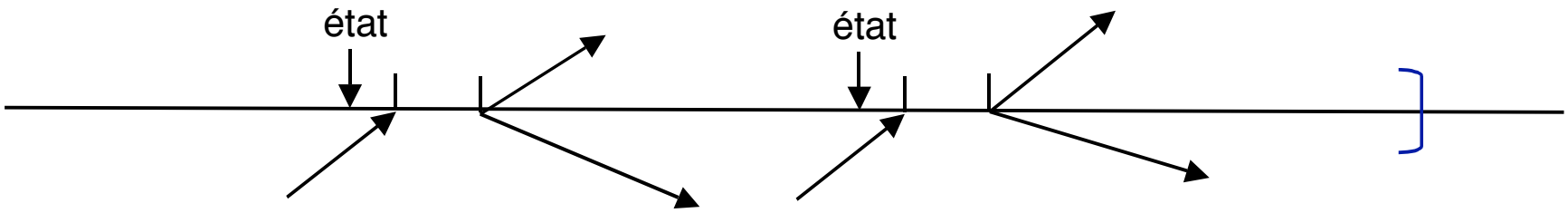
- ***A priori* (approche planifiée)**
  - **les sauvegardes des différents processus sont coordonnées pour constituer un état global cohérent (pas d'effet domino)**
  - **une synchronisation explicite doit donc être assurée entre les processus (coût lors de la sauvegarde)**
  - **il suffit de conserver, pour chaque processus, le dernier point de reprise**
  - **Exemples : Chandy-Lamport, Koo-Toueg**
- ***A posteriori* (approche non planifiée)**
  - **les sauvegardes des processus sont indépendantes**
  - **en cas de reprise, on essaie de reconstituer un état cohérent (coût lors de la restauration)**
  - **il faut conserver plusieurs points de reprise par processus, ainsi que les traces des communications**
  - **le risque d'effet domino n'est pas éliminé**
  - **Example : Juang-Venkatesan**

K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Transactions on Computer Systems, 3, 1, Feb. 1985, pp. 63-75
R. Koo, S. Toueg, Checkpointing and rollback-recovery in distributed systems, IEEE Trans. on Software Engineering, SE-13, 1, jan. 1987, pp. 23-31

Exemple d'enregistrement asynchrone (1)

Principe : Détecter les processus "orphelins" en comptant les messages envoyés et reçus
Hypothèses :
canaux fiables FIFO, asynchrones
application pilotée par les événements : réception, traitement, émission

Algorithme optimiste : pour chaque processus, on enregistre pour chaque événement, en mémoire volatile : (état, message reçu, messages émis). Périodiquement, ces enregistrements sont copiés en mémoire permanente.



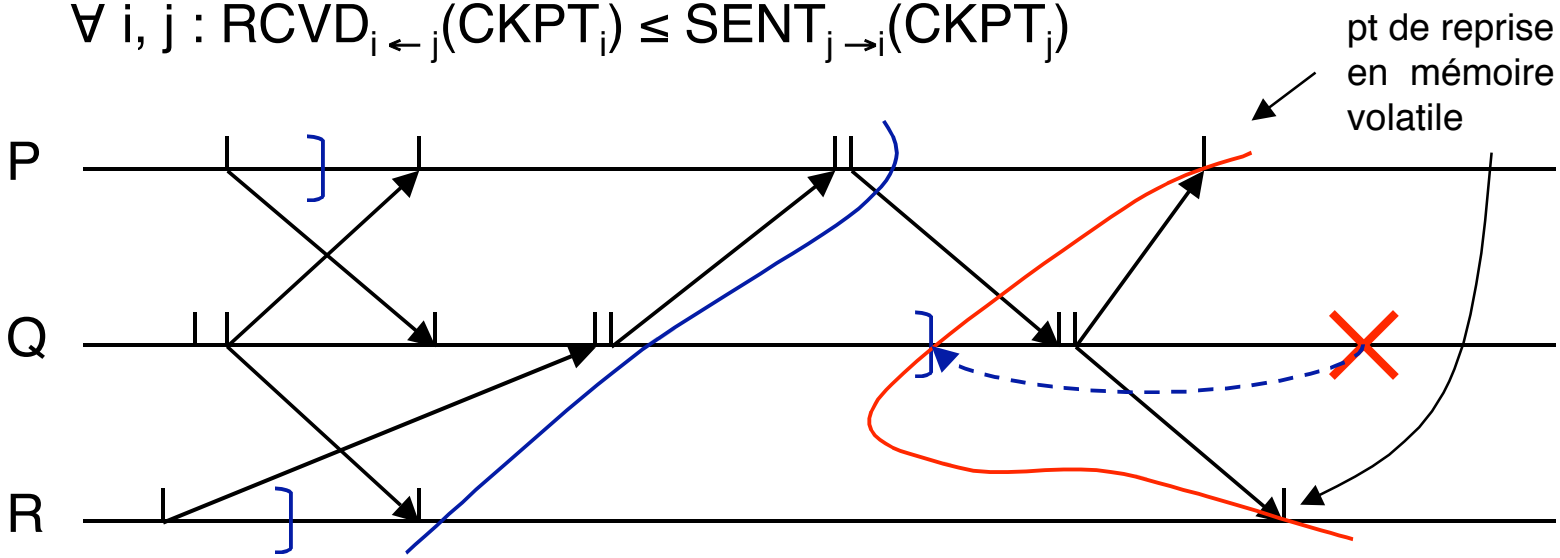
T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (2)

Soit $RCVD_{i \leftarrow j}(CKPT_i)$ le nb de messages reçus par i depuis j, enregistrés au point de reprise $CKPT_i$
Soit $SENT_{i \rightarrow j}(CKPT_i)$ le nb de messages envoyé par i vers j, enregistrés au point de reprise $CKPT_i$

Alors pour qu'un ensemble de points de reprise $CKPT_i$ soit valide, il faut que :

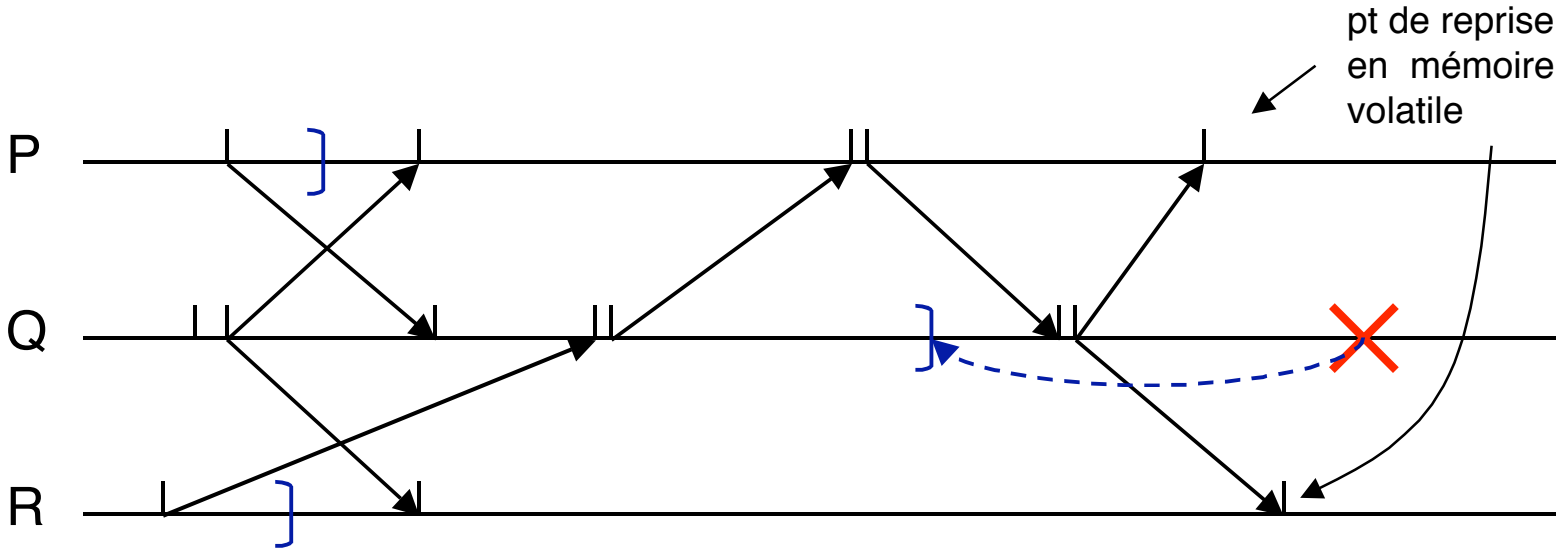
$$\forall\ i, j : RCVD_{i \leftarrow j}(CKPT_i) \leq SENT_{j \rightarrow i}(CKPT_j)$$



T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (3)

Algorithme : au moment de la reprise, le processus qui redémarre diffuse un message de reprise à tous. Chacun vérifie que son dernier point de reprise (en mémoire volatile pour les processus non en panne) vérifie la condition de validité. Sinon, il doit remonter au dernier point vérifiant la condition, et réitérer (à cause des réactions en chaîne)



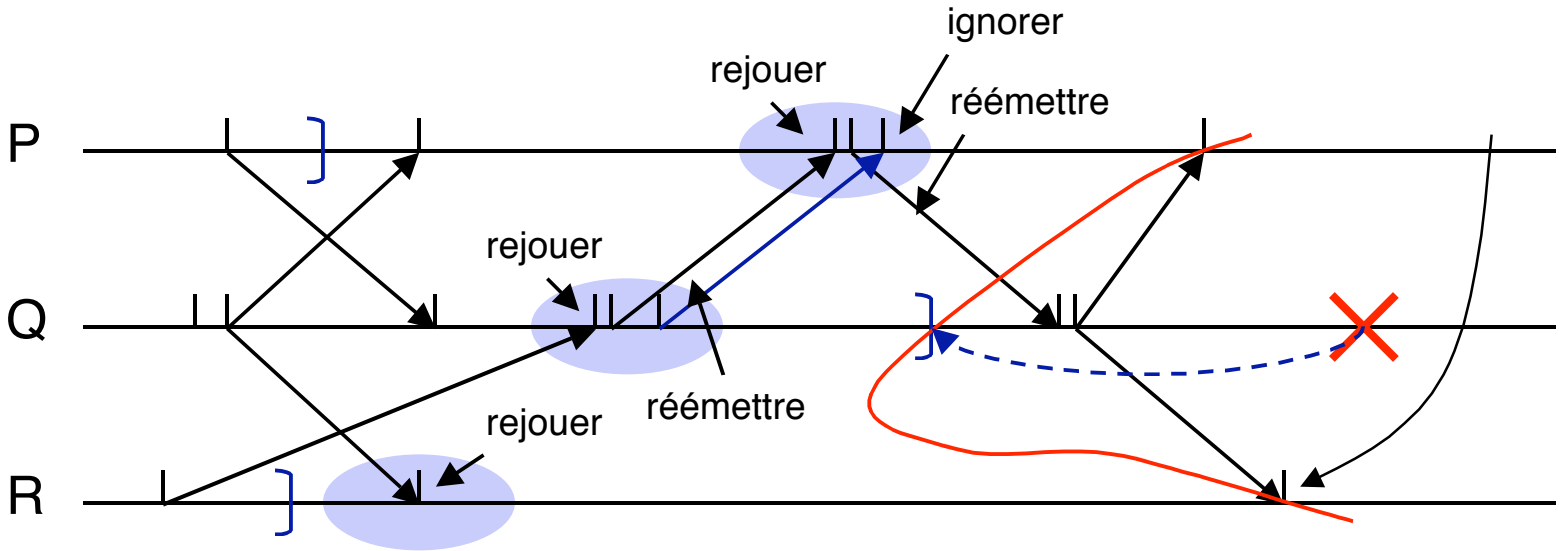
T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (4)

Lors de la reprise : utiliser les informations enregistrées au point de reprise (état, message reçu, message émis)

rejouer le dernier message reçu (qui a été enregistré)
rejouer le traitement
réémettre les messages émis

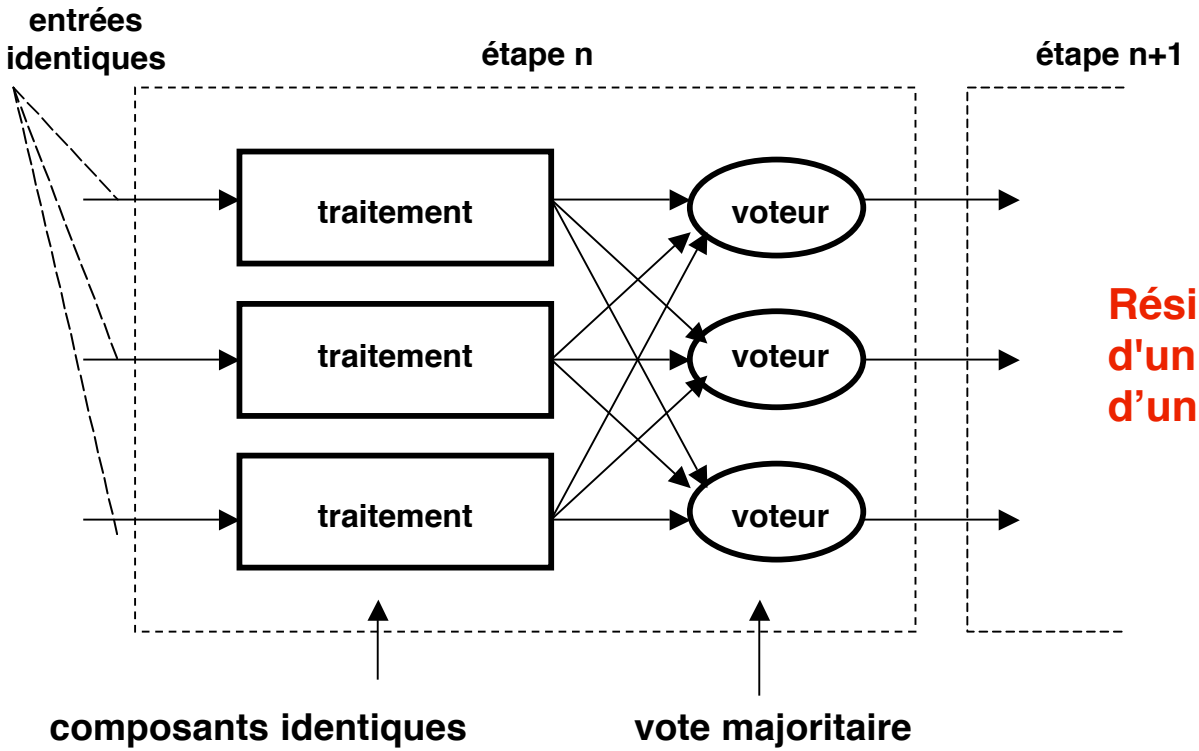
ignorer les messages dupliqués



T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

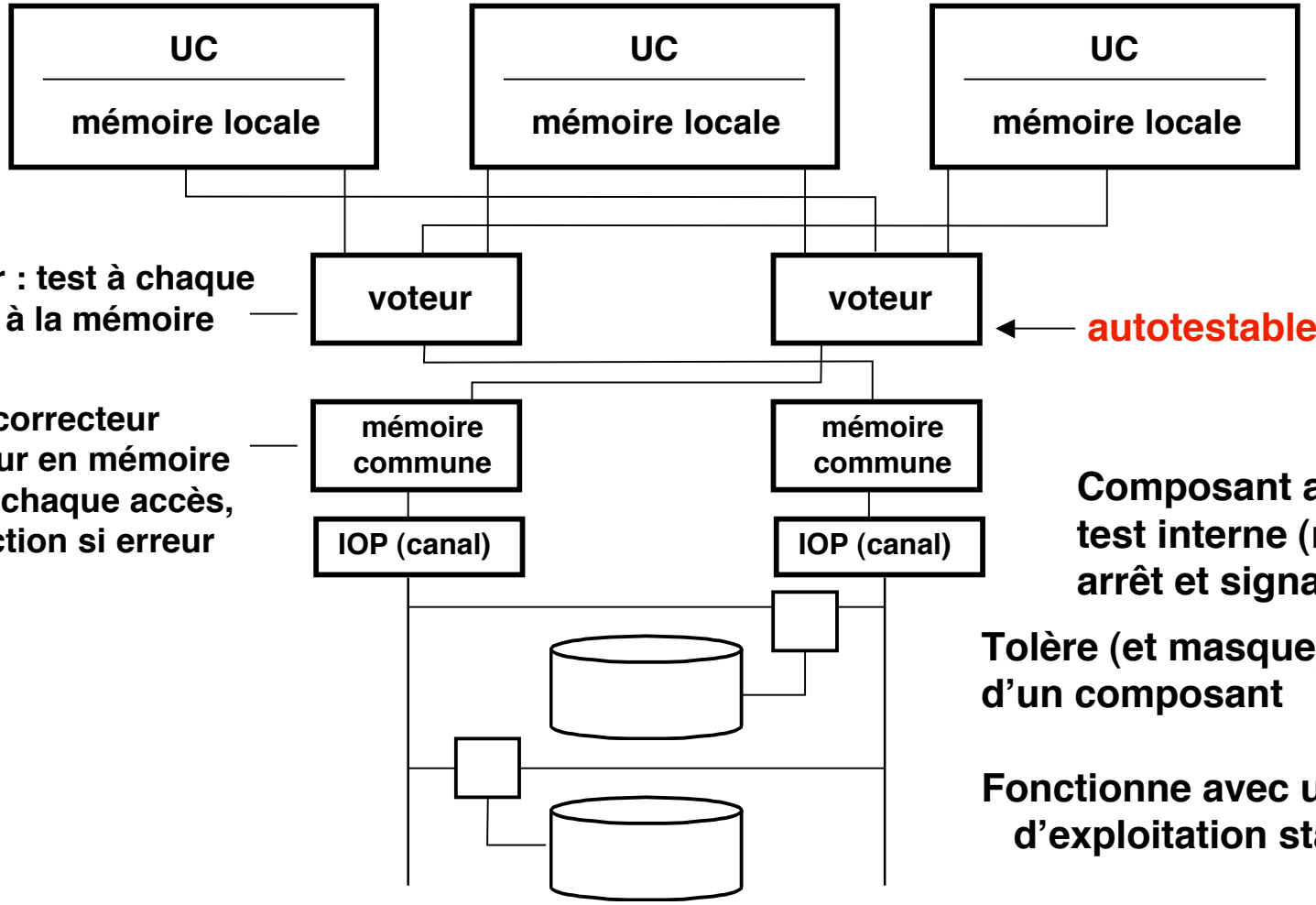
Exemple de compensation : vote majoritaire

Système TMR (*Triple Modular Redundancy*)



Résiste (en général) à la défaillance d'un composant de traitement et d'un voteur par étape

Exemple de compensation : Tandem Integrity

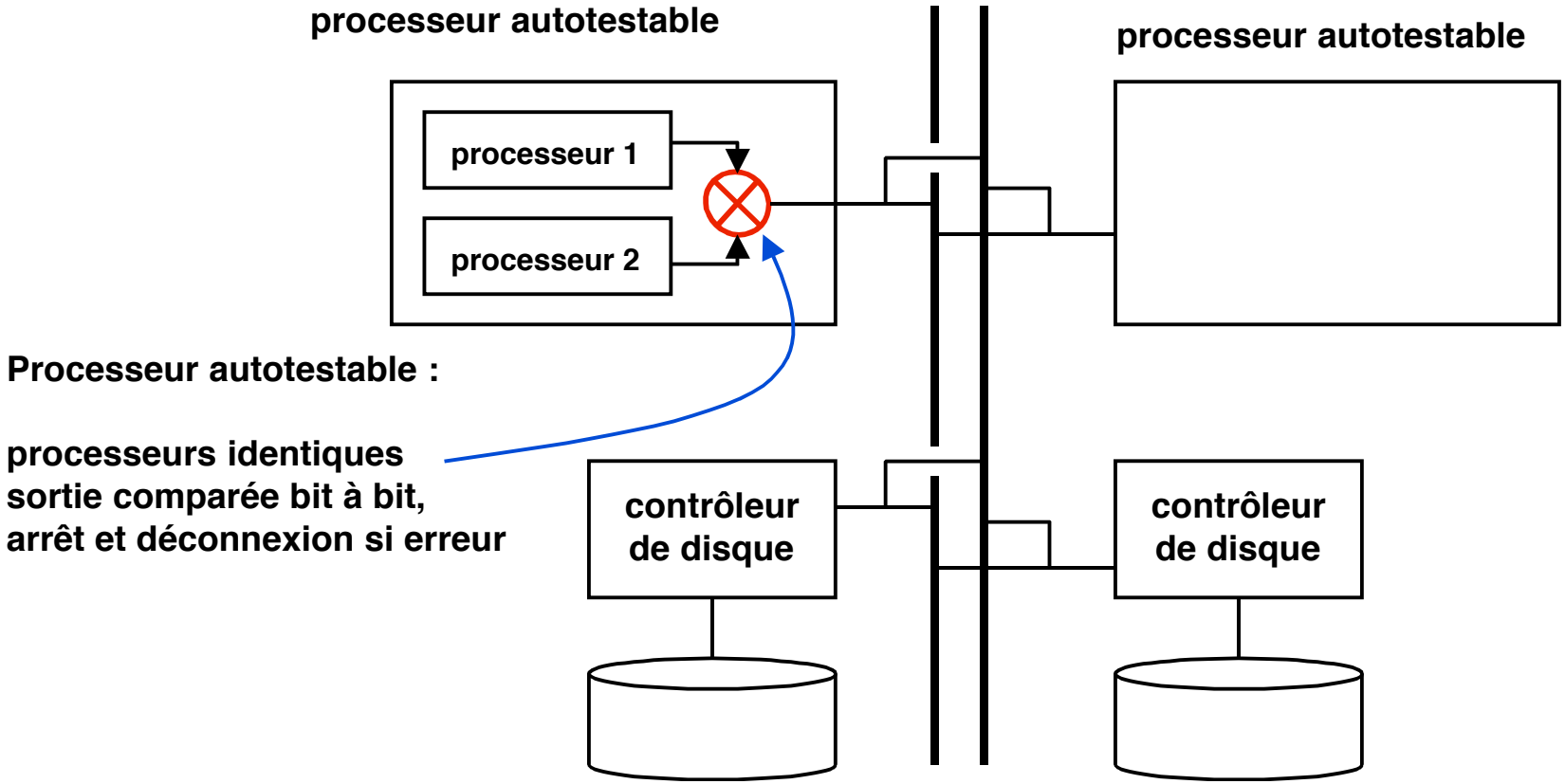


Composant autotestable : test interne (redondance), arrêt et signal si erreur

Tolère (et masque) la faute d'un composant

Fonctionne avec un système d'exploitation standard

Exemple de compensation : Stratus S/32



Compensation : résumé

- **Avantages**
  - Traitement d'erreur rapide
  - Masquage : défaillances internes invisibles aux composants utilisateurs, à condition que
    - les hypothèses de fautes soient respectées
    - les fautes soient indépendantes sur les composants dupliqués
- **Inconvénients**
  - Redondance élevée, donc coût élevé
    - Tandem Integrity : X 3
    - Stratus : X 4

Difficulté de la tolérance aux fautes

- **Les pièges sont nombreux …**
  - Mauvaise analyse des fautes possibles
    - ne pas prévoir des cas possibles de faute
    - laisser passer des fautes sans réaction
  - Réaction inappropriée due à des hypothèses mal formulées
    - le “remède” peut aggraver la situation
  - Mise en œuvre incorrecte de la redondance
    - les éléments redondants doivent avoir des modes de défaillance différents
    - une mauvaise redondance donne un faux sentiment de sécurité
- **Deux exemples d’école**
  - Ariane 501

    http://www.cnes.fr/espace_pro/communiques/cp96/rapport_501/rapport_501_2.html
  - Therac 25

    http://sunnyday.mit.edu/therac-25.html

Ariane 501

Le 4 juin 1996, le premier tir de la fusée Ariane 5 se termina par l'explosion de l'engin environ 40 secondes après le décollage.

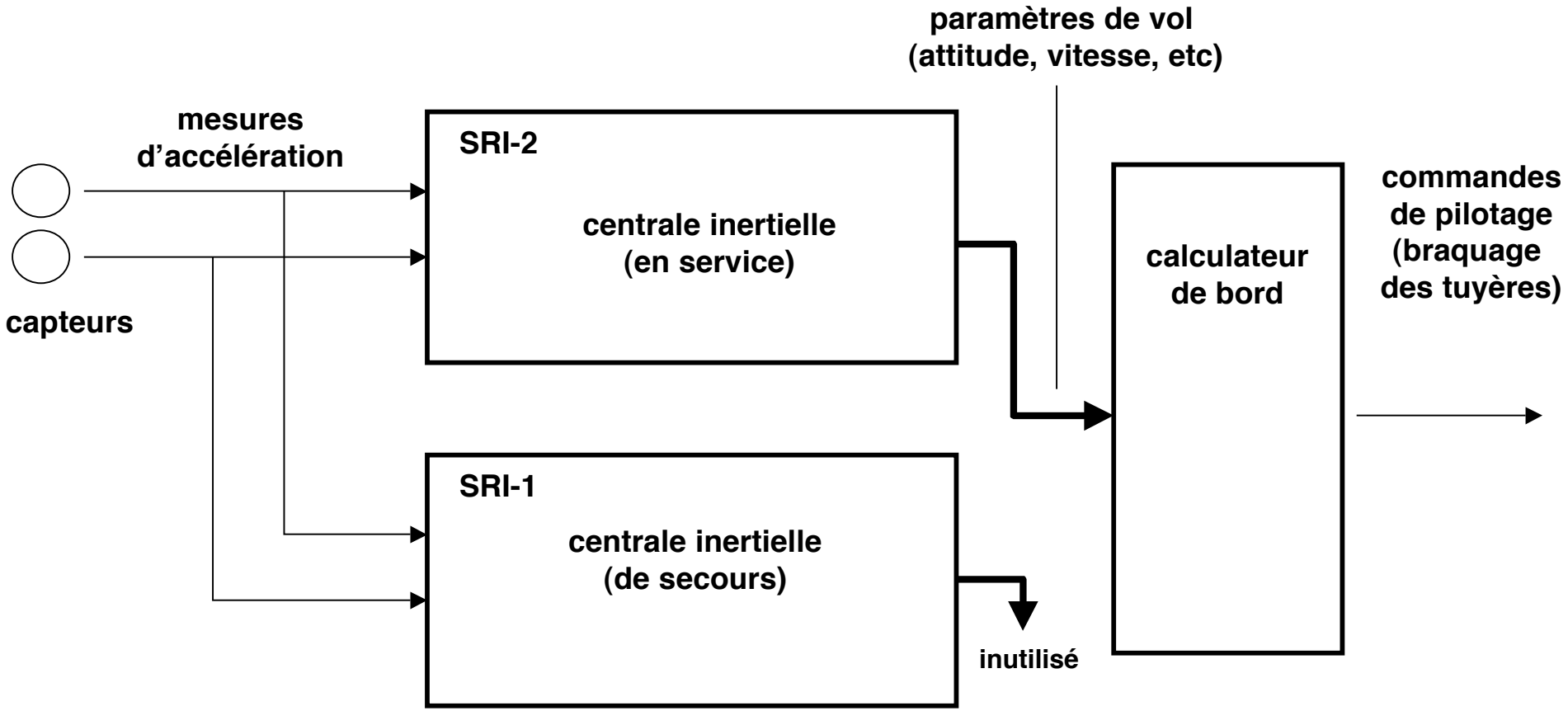
L'enquête permit de déterminer les circonstances précises de la catastrophe et mit en évidence plusieurs erreurs de conception du logiciel de commande, et notamment de son système de tolérance aux fautes, qui conduisirent au déclenchement d'une réaction inappropriée alors que les conditions de vol étaient normales.

voir rapport d'enquête en :

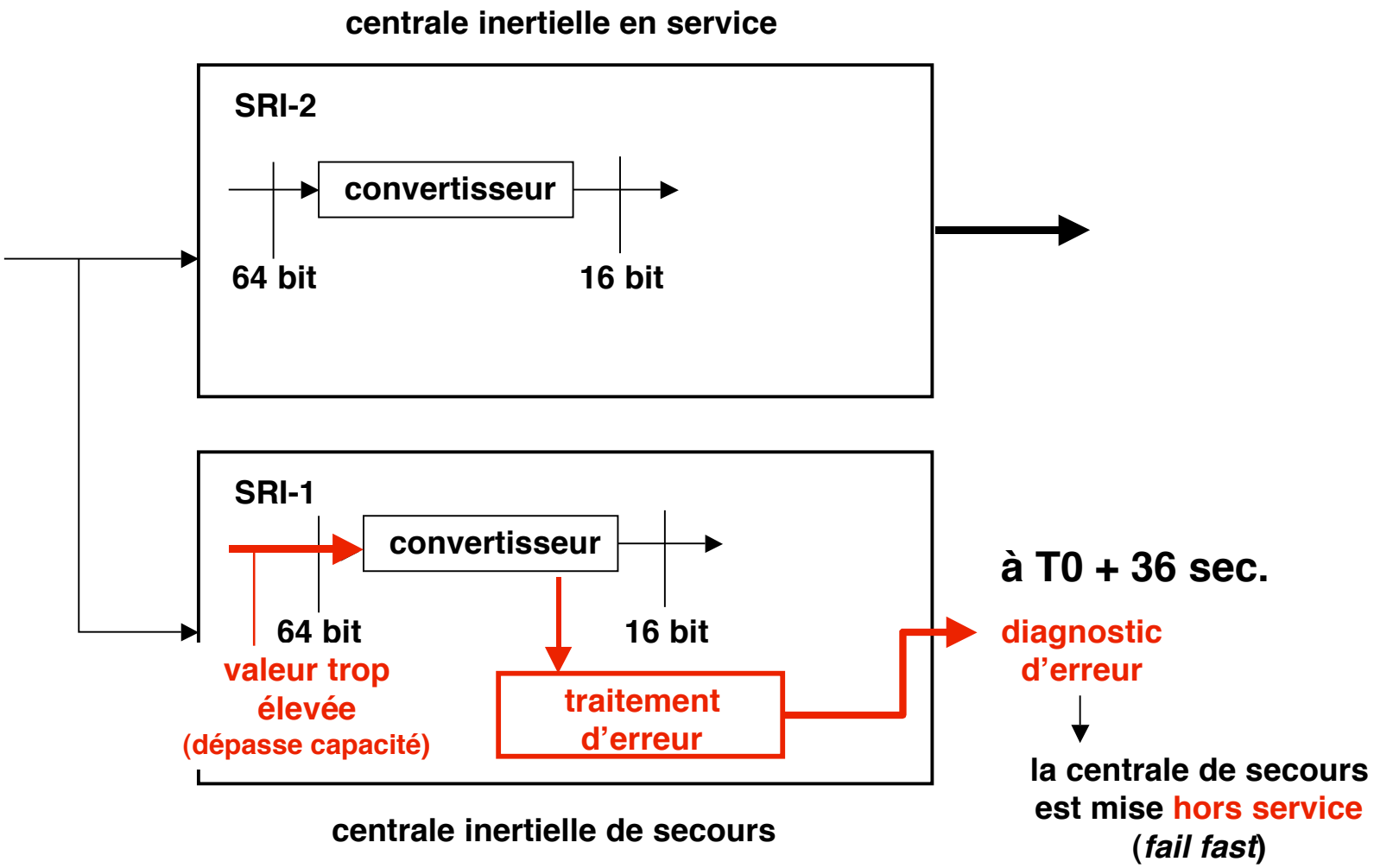
http://www.cnes.fr/espace_pro/communiques/cp96/rapport_501/rapport_501_2.html

Ariane 501: scénario (très) simplifié (1)

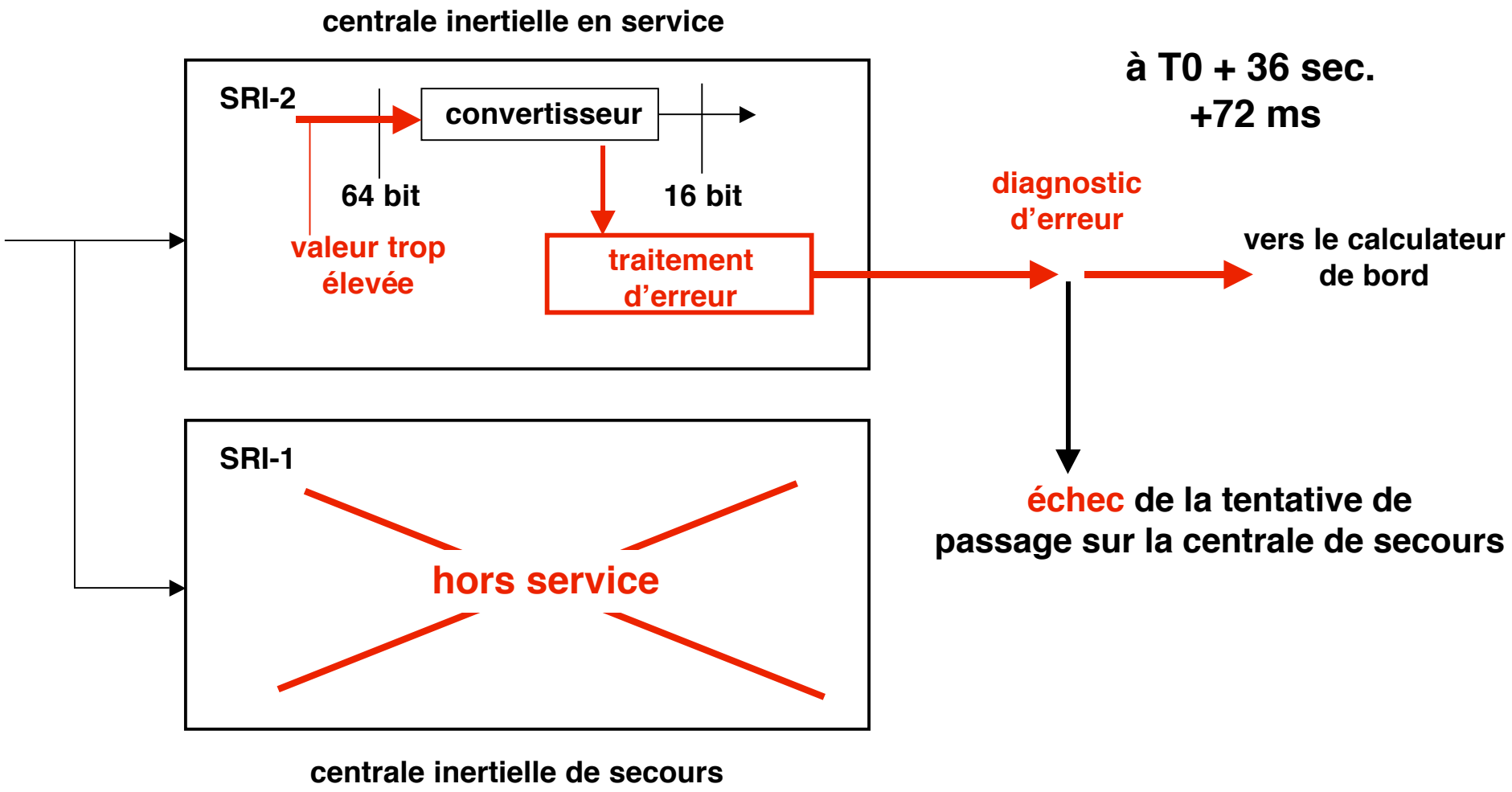
Le système de pilotage



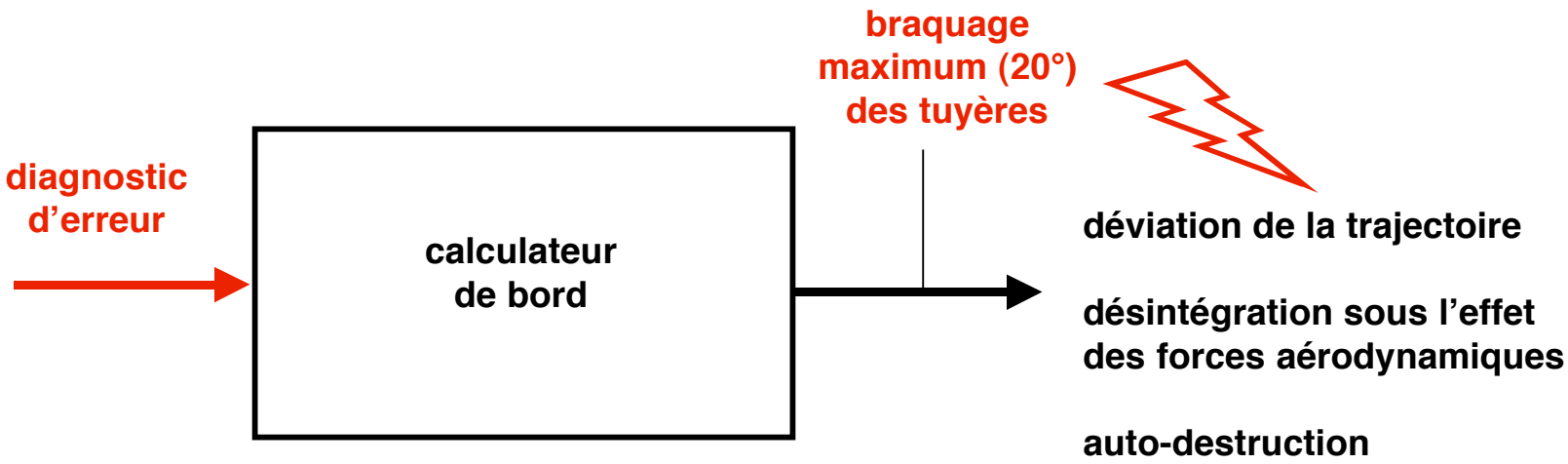
Ariane 501: scénario (très) simplifié (2)



Ariane 501: scénario (très) simplifié (3)



Ariane 501: scénario (très) simplifié (4)



Le calculateur de bord reçoit en entrée un diagnostic d'erreur, qu'il interprète comme des données de vol. Les valeurs étant hors limites, il réagit (normalement) par un ordre de braquage maximum

Ariane 501 : explications (1)

Pourquoi le convertisseur a-t-il reçu en entrée une valeur trop grande ?

1) L'entrée était la mesure d'une vitesse de déplacement horizontale, et servait à un système de calibrage des instruments utilisé avant le décollage. Ce système était inchangé depuis sa version Ariane 4 (on ne change pas un dispositif qui marche…).

2) Néanmoins, ce système était maintenu en service 40 secondes après le décollage, en vue de permettre l'arrêt puis la reprise du compte à rebours dans les dernières secondes avant le décollage (dans une limite de 40 secondes). Cette disposition (qui a servi une fois pour Ariane 4) était en fait inutile pour Ariane 5 en raison d'un changement des procédures.

3) Mais la trajectoire d'Ariane 5 était différente de celle d'Ariane 4, et en particulier sa vitesse de déplacement horizontale était beaucoup plus élevée

4) Donc la valeur de l'entrée, correspondant à la vitesse horizontale mesurée, a dépassé les capacités du convertisseur

Ariane 501 : explications (2)

Pourquoi le convertisseur a-t-il réagi en envoyant un diagnostic d'erreur sur le bus de données ?

1) **Les instructions de conversion de données (écrites en Ada) n'étaient pas protégées par un traitement d'exception (on n'avait pas prévu que la capacité d'entrée pouvait être dépassée).**

2) **C'est le dispositif standard de réaction aux erreurs qui a été activé par défaut. La centrale à inertie a déclaré son incapacité à fonctionner et a été mise hors service selon le principe *fail stop* (arrêt du processeur).**

3) **L'hypothèse sous-jacente (non explicite) était celle de la défaillance matérielle transitoire : défaillance de probabilité très faible, traitée par redondance (on passe sur la centrale de secours).**

4) **Mais la défaillance étant d'origine logicielle, les même causes ont produit des effets identiques, et la centrale de secours s'est trouvée hors service pour les mêmes raisons que la centrale active.**

5) **On se trouvait donc hors des hypothèses prévues pour le traitement des défaillances (panne simultanée des deux unités). Ce cas étant supposé ne jamais se présenter, rien n'était prévu pour le traiter et le diagnostic d'erreur a été envoyé sur le bus de données.**

Ariane 501 : la morale de l'histoire

Le scénario de la catastrophe révèle plusieurs erreurs de conception

1) Analyse incorrecte de la transition Ariane 4 → Ariane 5 (pour la centrale à inertie)
   - Un dispositif totalement inutile a été maintenu en fonction dans un équipement critique
   - Les conséquences du changement de conditions (vitesse différente) n'ont pas été analysées

2) Analyse incorrecte des hypothèses de défaillance
   - Le dépassement de capacité n'a pas été prévu dans le convertisseur (pas de traitement d'exception associé à l'opération de conversion).
   - L'hypothèse à la base de la redondance a été incorrectement formulée (la redondance ne vaut que si les conditions de défaillance sont indépendantes pour les deux unités redondantes).
   - L'éventualité d'erreurs logicielles (qui précisément ne sont pas indépendantes, surtout si un même algorithme sert dans les deux unités) n'a pas été prise en compte.

3) Prévision insuffisante des conditions de traitement de situations anormales
   - La réaction de mise hors service de la centrale après erreur (arrêt du processeur) prive le système d'une possibilité de reprise (cette erreur d'analyse est liée à l'idée que la défaillance est toujours traitée par la redondance des unités).
   - La possibilité d'envoyer des informations de diagnostic sur un bus de données aurait du être totalement exclue. Il valait mieux, en cas de situation anormale, envoyer des données "raisonnables", par exemple obtenues par extrapolation.