

Tolérance aux pannes

Définition

- Service

- Ensemble de fonctions défini par *une interface*, "contrat" entre le fournisseur et l'utilisateur du service
- **Un système est en faute** s'il ne se comporte pas de manière consistante avec ses spécifications
 - Renvoi d'une valeur erronée
 - Absence de réponse

Notion de panne catastrophique ➔ vie humaine en jeu

Quelques termes

- **Fiabilité (reliability)**

Probabilité pour qu'un système soit continûment en fonctionnement sur une période donnée

- **Disponibilité (availability)**

Probabilité pour qu'un système soit en fonctionnement à un instant t donné

- **Maintenabilité**

Probabilité pour qu'un système en panne à l'instant 0 soit réparé à l'instant t

- **Sécurité (safety)**

Probabilité pour qu'un système soit continûment en fonctionnement non catastrophique sur une période (entre 0 et t)

On ne parle pas de sécurité pour la sécurité des accès et des données (sécurité \rightarrow security), c'est-à-dire la confidentialité et l'intégrité des informations.

Modèle utilisé (1)

- Pour le processeur physique
 - Unité de traitement
 - Horloge locale
 - Mémoire locale
- Services de communication
 - Réception des données
 - Envoie des données
- **Un composant est correcte s'il respecte sa spécification**
 - Mais évaluation en boîte noire (généralement)
 - Observation par les interfaces


Modèle utilisé (2)

- Spécification d'un composant
 - Un ensemble possible d'événements entrants
 - Les traitements à réaliser
 - Les ensembles possibles de messages produits en résultat
 - Les contraintes temporelles (si nécessaire)

Vérification pour les composants :

- temps de traitement, instructions respectées,
- temps de transit entre message borné,
- granularité des horloges, ...

Les pannes (1)

- Panne franche, failstop, crash
 - Une fois en panne, le composant cesse immédiatement de répondre à toute sollicitation venant de l'extérieur
 - C'est une panne permanente (obligation de réparation)
- Panne transitoire ou intermittente, omission
 - En réponse à un événement, un composant ne délivre pas la réponse attendue (perte du service habituel)
 - Il répond ensuite de manière correcte, sans déviation par rapport à la spécification
 - 
 - Panne transitoire : apparaît une seule fois
 - Panne intermittente : apparaît de temps en temps

Les pannes (2)

- Panne temporelle
 - Une sortie correcte associée à un événement entrant se manifeste trop tôt ou trop tard (déviation de la spécification que sur le temps)
- Panne quelconque ou byzantine
 - Comportant quelconque s'écartant de la spécification
 - Comportement byzantin naturelle
 - Comportement byzantin malveillant (virus)
 - Panne difficile à détecter...

Génération d'une panne (1)

- Apparition d'une panne
 - **Une erreur** : état du système susceptible de provoquer une défaillance du système, et donc de sortir de la spécification
 - Ex : connexion physique coupé, instanciation mal faite
 - Une Faute : toute cause (événement, action,...) pouvant provoquer une erreur
 - Ex : erreur de programmation , malveillance, ...
 - **Une erreur est susceptible de provoquer une panne, mais pas obligatoirement immédiatement**
 - Du fait de la prise en compte de l'erreur par le système
 - Car la partie erronée n'est pas tout de suite utilisée

Génération d'une panne (2)

- Une erreur est latente tant qu'elle n'a pas provoqué de panne
- Le temps entre l'apparition de l'état d'erreur et la défaillance est le délai de latence
 - Plus le délai est long, plus la recherche de la panne est dure (on peut avoir la propagation de la panne entre composants)

Les classes de pannes sont imbriquées :

panne
franche \subset panne
transitoire \subset panne
temporelle \subset panne
byzantine

La sûreté de fonctionnement

- Evitement des fautes
 - Par la prévention
 - Analyser les causes possibles
 - Par l'évaluation
 - Simuler les services pour voir le fonctionnement
 - Par la vérification
 - Utilisation de test
- Tolérance aux fautes
 - Préserver le fonctionnement du système malgré les fautes

Traitement

- Différentes phases

1. Détection

Découvrir l'existence de l'erreur

2. Localisation

Identifier l'endroit de l'erreur

3. Isolation

Confinement de l'erreur

4. Réparation

Remettre le système en état et éviter que cela se reproduise

La redondance

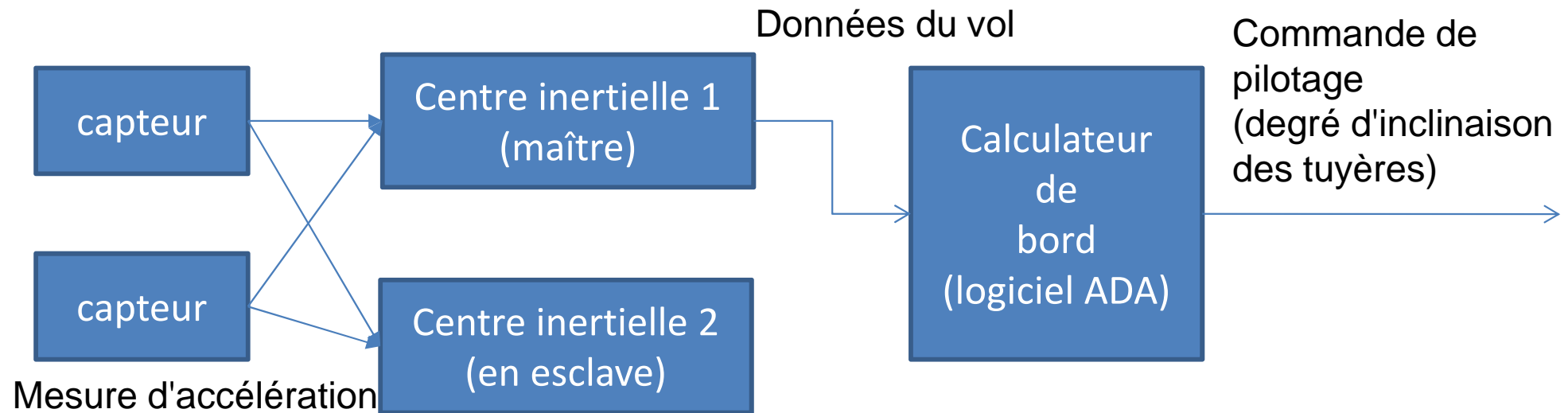
- Matérielle
- Des traitements
- Des données
- Temporelles
- Spatiales
 - Passive
 - Active

Techniques de traitement

- Deux techniques :
 1. Détection explicite de l'erreur :
 - Reprise (on repart d'un état OK)
 - » nécessite une sauvegarde des états
 - Poursuite (on essaie de reconstituer un état correct)
 - Pas simple à mettre en œuvre, souvent service dégradé ensuite
 2. Détection par compensation
 - Redondance permettant de masquer l'erreur
 - » utilisation d'un vote majoritaire

Exemple 1

- Ariane 5
 - Date 4 juin 1996
 - Crash en plein vol après 39 secondes
 - Problème de redondance !!

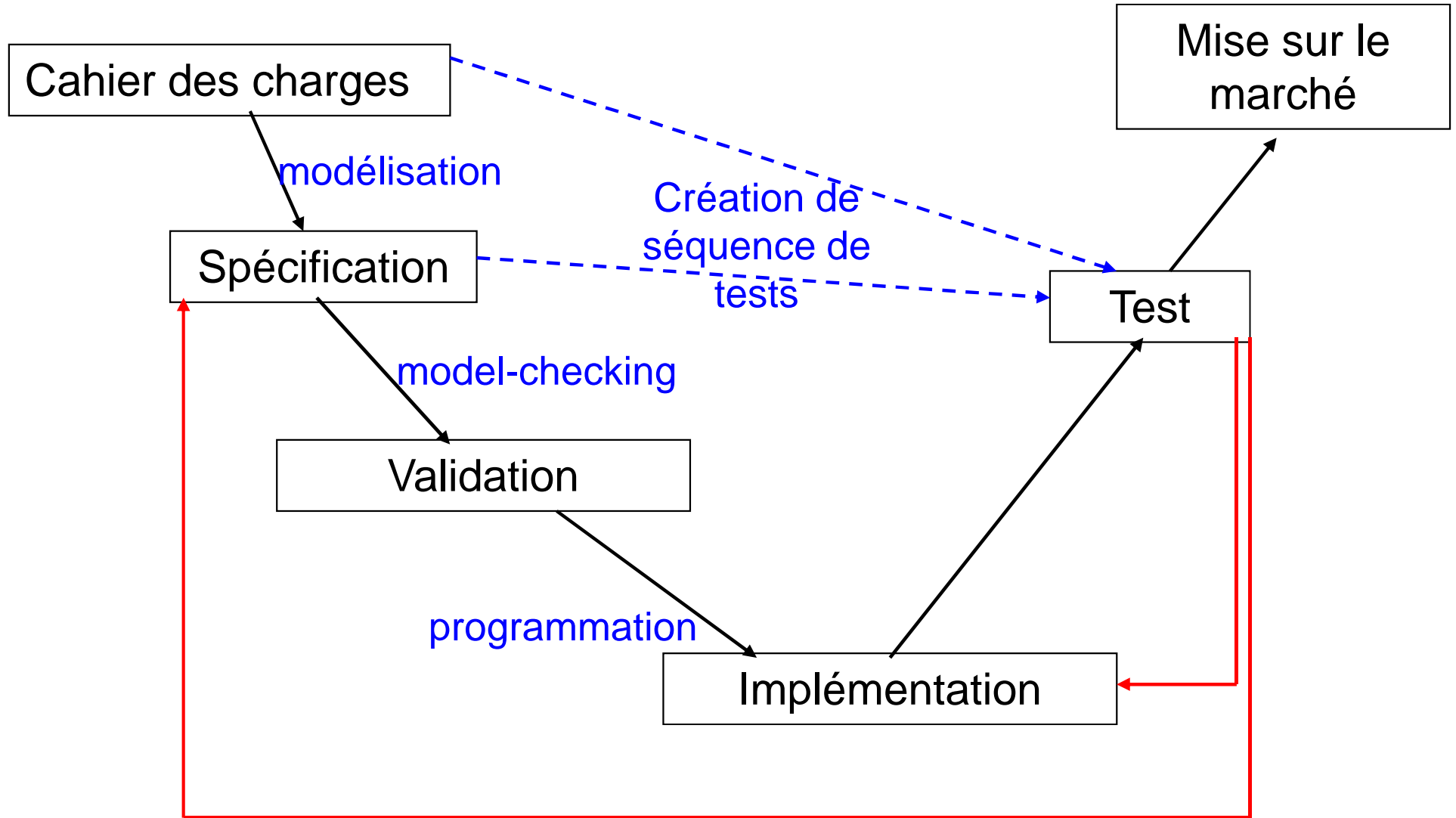


Exemple 2

- La Haute Disponibilité (High Availability)
 - Exemple : heartbeat sous linux
 - les machines virtuelles
 - Les clusters...
- Pour les données : RAID, DRDB, oracle data guard

Méthodologie de test pour les systèmes répartis

Cycle de développement



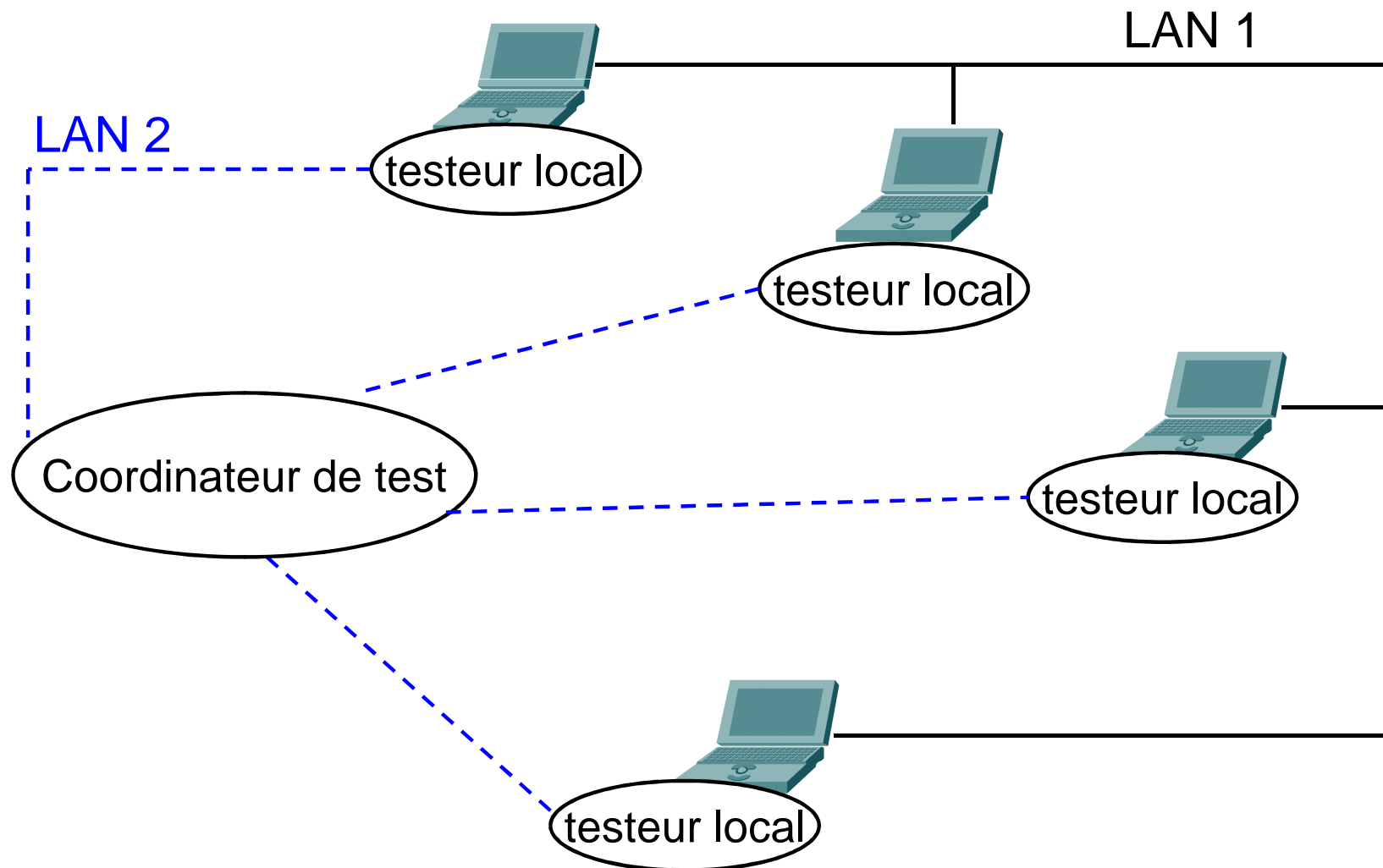
Tests

- Différents tests
 - test de conformité
 - test de robustesse
 - test de performance
 - test d'interopérabilité
- Différentes conditions
 - test boîte blanche → orienté logiciel
 - test boîte noir → orienté communication

A une Implémentation Sous Test (IST) est toujours associé un testeur :

- PCO (Point de Control et d'Observation)
- PO (Point d'Observation)

Architecture Test



Méthodologie (1)

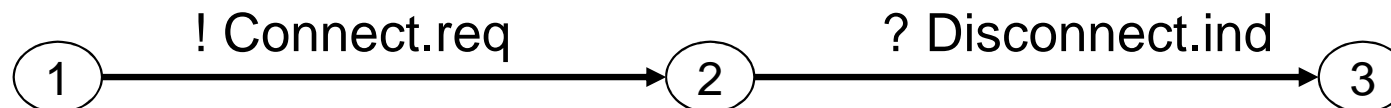
- But : obtenir une séquence de test pour chaque testeur local
→ nécessité d'une synchronisation entre eux



Différentes étapes:

1. Modéliser le système avec un langage formel
→ exemple : LOTOS, DSL, UML, IOSM

Chaque composant doit avoir son propre alphabet afin d'éviter des interactions.



Méthodologie (2)

2 : Création des séquences de test

- en utilisant une méthode exhaustive -> séquence de test très longue
- en utilisant une méthode non exhaustive
 - > méthode **orienté Objectif de Test**
 - Sélection d'une propriété à tester
 - Génération de la séquence de test permettant de tester cette propriété

3 : Génération de tests dédiés pour les testeurs locaux

- division de la séquence initial en "tests dédiés" pour chaque testeur
- introduction d'une synchronisation entre les testeurs avec l'utilisation de drapeaux ($-\text{sync}_j^P$, $+\text{sync}_j^P$)

4 : Utilisation de ces "tests dédiés" pour obtenir un verdict

Exemple sur le WAP

☀ Objectif de test : **S_connect.req** puis **S_methodeInvoke.req**

