
Sûreté de Fonctionnement (SdF)

- 1^{ère} partie : généralités -

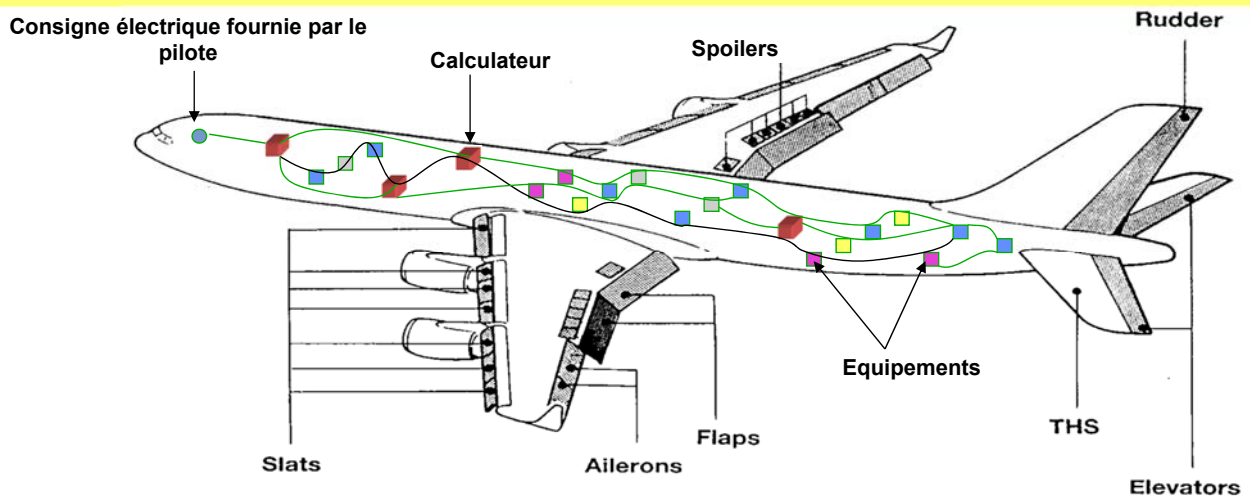
1 – Introduction

2 – Premières définitions de la SdF

3 – SdF : Concepts de base et terminologie

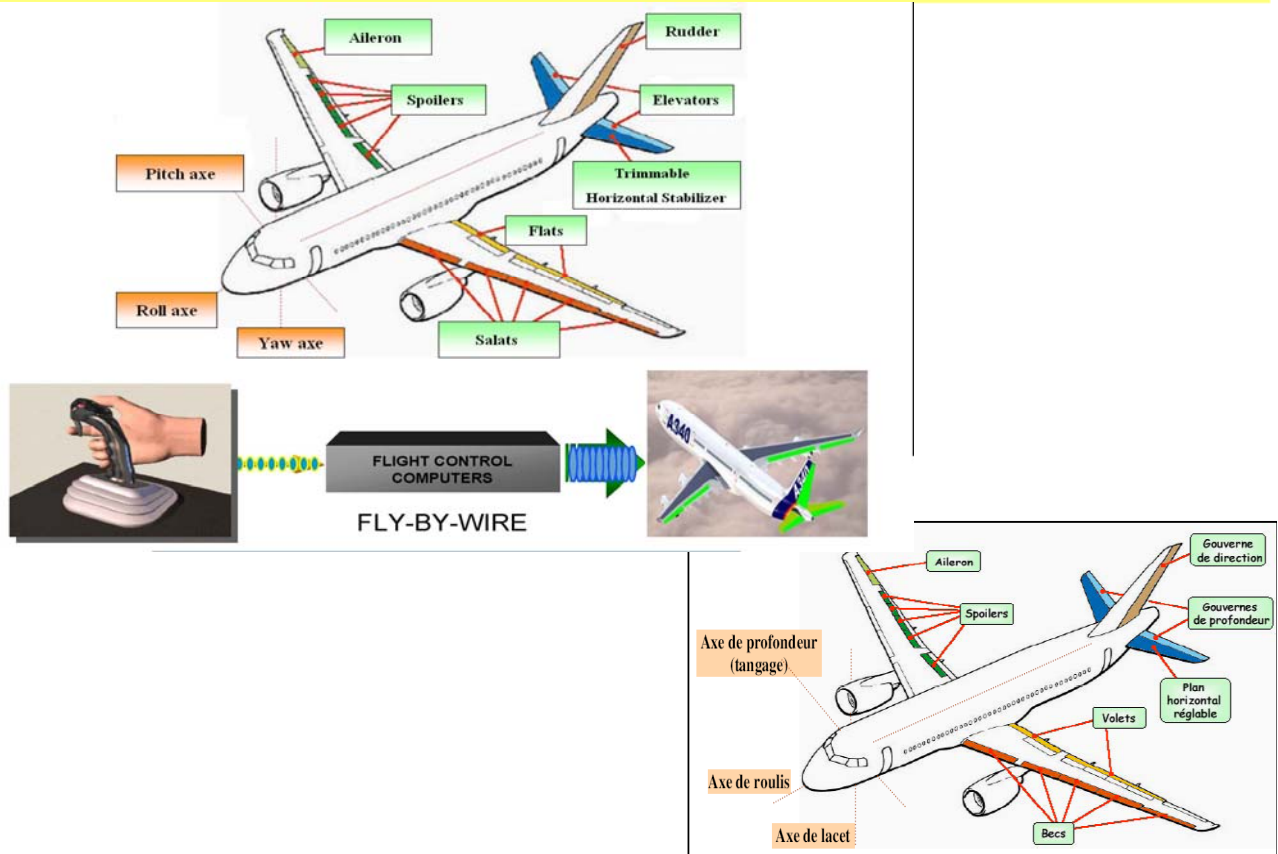
1 – Introduction : exemple de système à hautes exigences de sûreté de fonctionnement : CDVE

Système de Commandes De Vol Électrique (CDVE) : contrôle la trajectoire avion en appliquant les consignes du pilote (mécanique, électrique) aux surfaces de contrôle

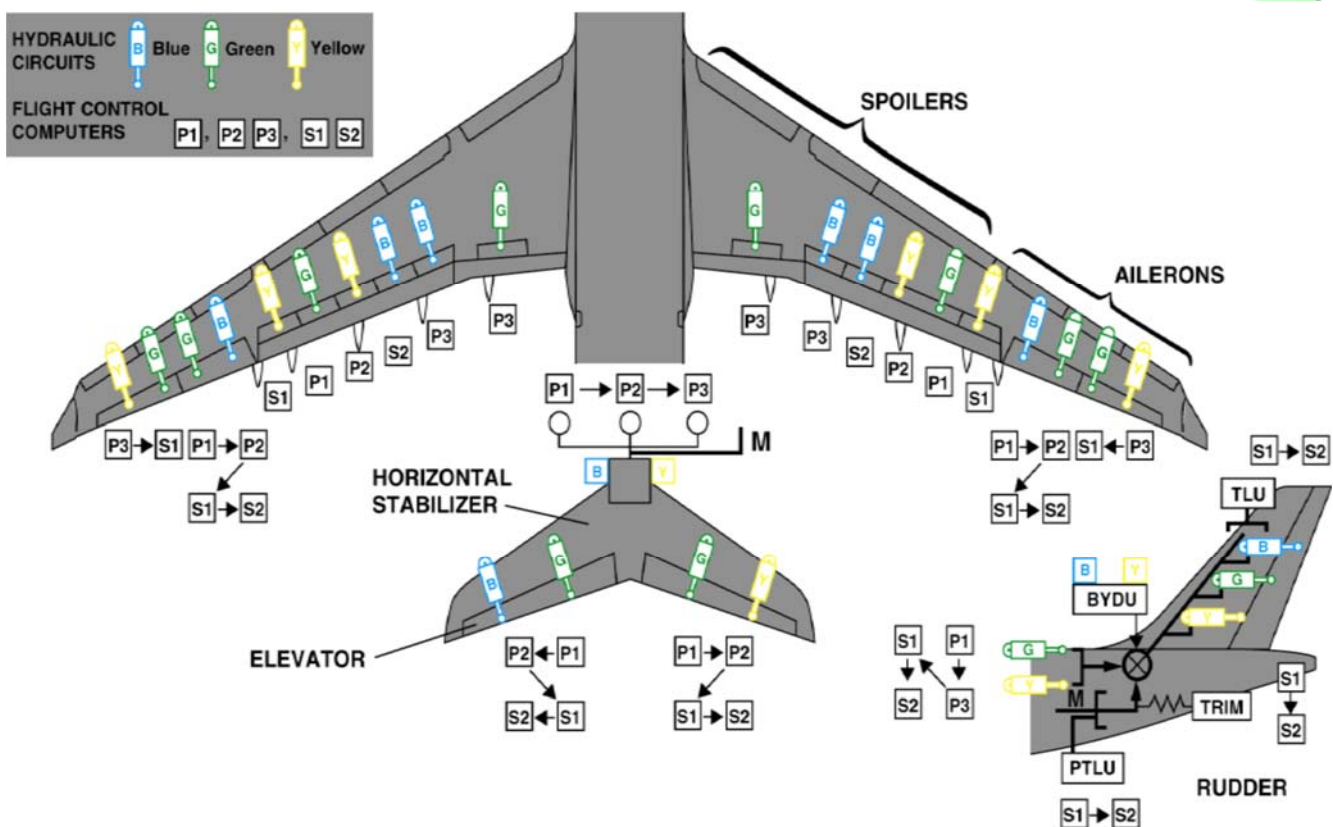


CDV : système de commande-contrôle, embarqué, réparti, temps réel, et critique → démonstration de la sûreté de fonctionnement régie par de sévères normes de certification

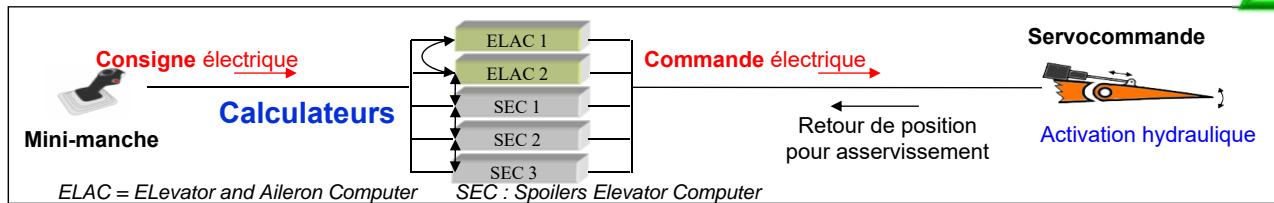
Surfaces de contrôle



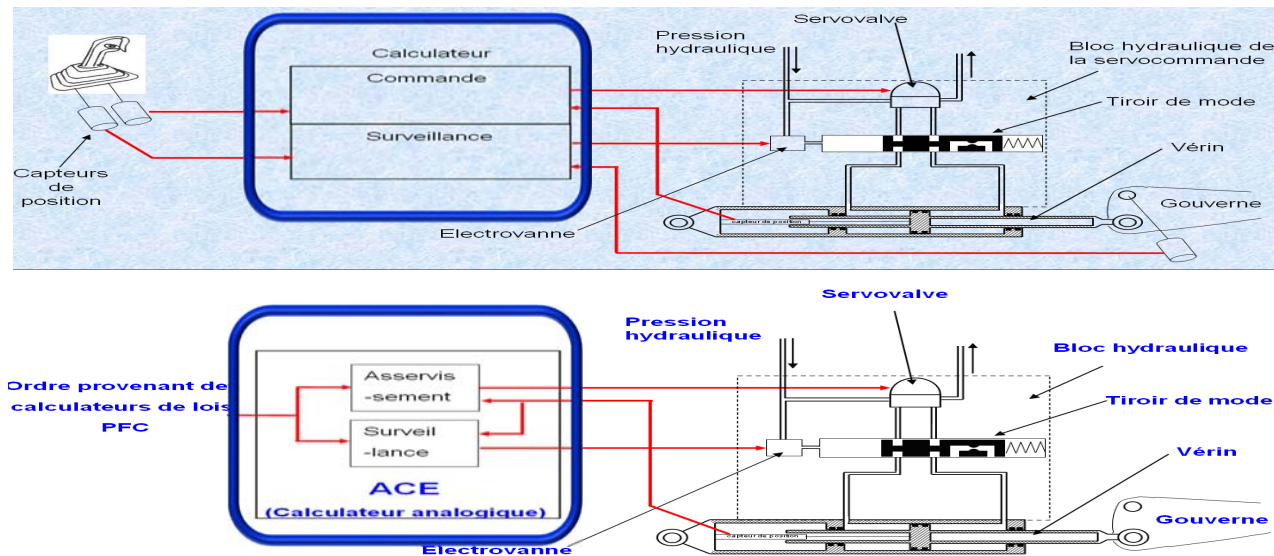
Surfaces de contrôle : nombre de « vérins » et calculateurs



Détails de la "chaîne" de commande-contrôle

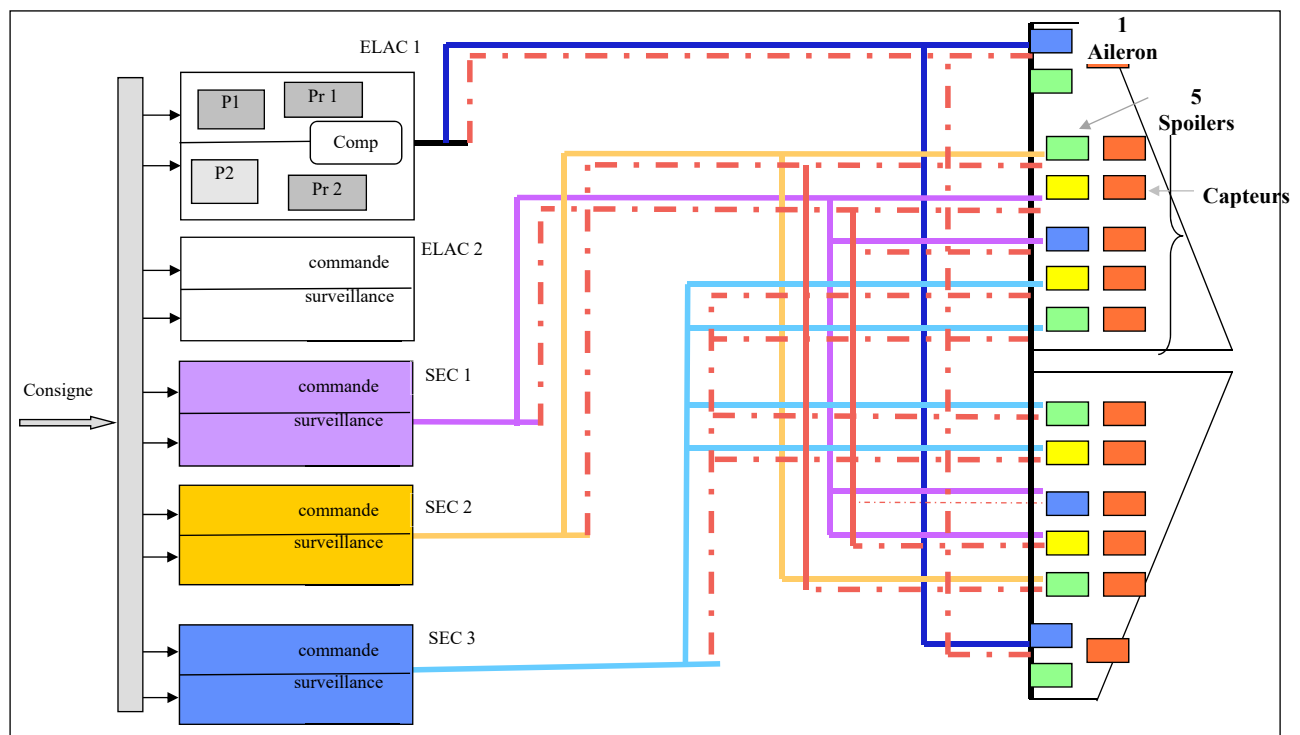


Détails "commande" et "actionneurs" Airbus et Boeing



Exemple de l'A320 : 24 asservissements au total

Les différentes couleurs traduisent la Ségrégation des moyens de commande et d'activation des surfaces

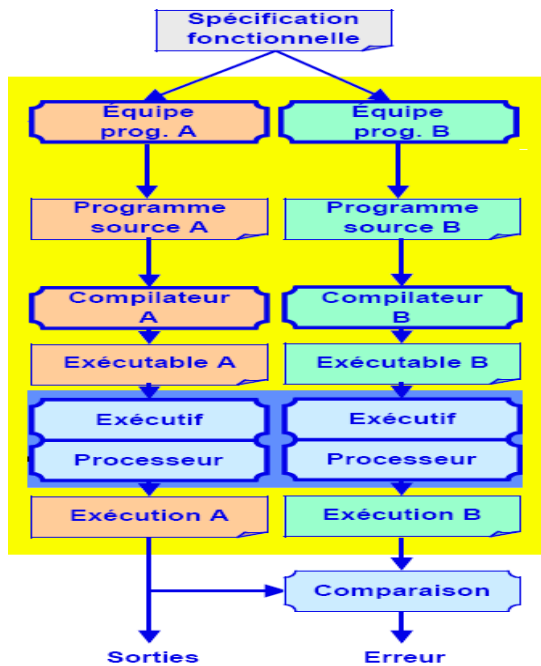


Sûreté de fonctionnement dans les CDVE : Techniques de tolérance aux fautes : Diversification, redondance



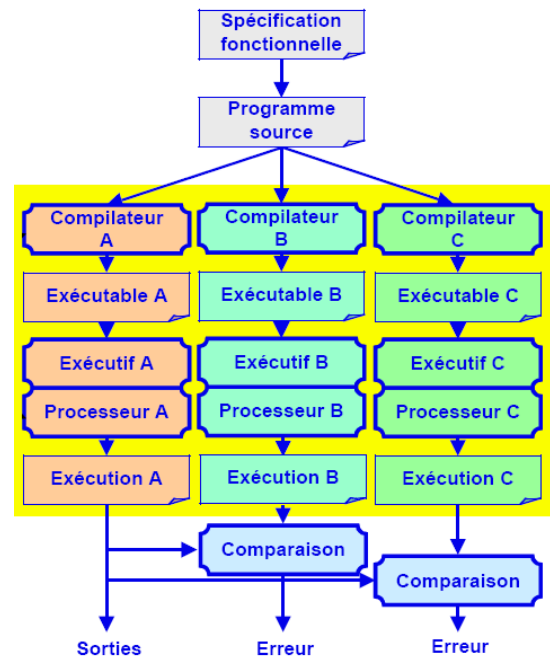
✓ La diversification → 2 grands principes de développement

Développement N-autotestables



Sûreté de fonctionnement

Développement N-Versions



Chapitre 1 : Introduction

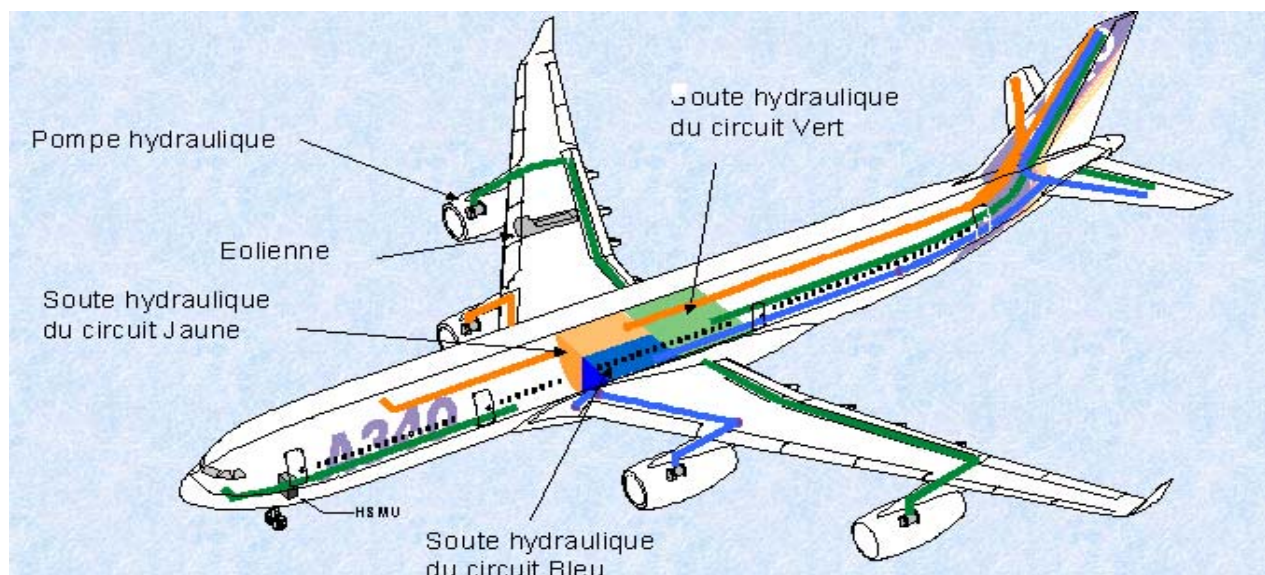
11

Sûreté de fonctionnement dans les CDVE : Techniques de tolérance aux fautes

✓ La redondance → plusieurs exemplaires d'un même équipement (S)



✓ La ségrégation → répartition géométrique de l'ensemble des ressources redondantes (les "câblages" ne passent pas au même endroit : couleurs ≠



Sûreté de fonctionnement

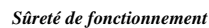
Chapitre 1 : Introduction

12

Pratiques industrielles chez Boeing : Briques de base

-
- The diagram illustrates the PFC (Power/Fuel Control) architecture, which is divided into three lanes: LANE 1, LANE 2, and LANE 3. Each lane contains a Power Supply, a Micro-Processor, and an ARINC 629 Interface. The Micro-Processors are connected to the ARINC 629 Interfaces via bidirectional arrows. The ARINC 629 Interfaces are connected to a common bus labeled L, C, and R. The bus lines are shown as horizontal dashed lines with small square connectors at the end of each lane.
- LANE 1 (Pink border):** Contains a POWER SUPPLY, MICRO-PROCESSOR AMD 29050, and ARINC 629 INTERFACES.
 - LANE 2 (Green border):** Contains a POWER SUPPLY, MICRO-PROCESSOR MOTOROLA 68040, and ARINC 629 INTERFACES.
 - LANE 3 (Blue border):** Contains a POWER SUPPLY, MICRO-PROCESSOR INTEL 80486, and ARINC 629 INTERFACES.
- Horizontal arrows connect the Micro-Processors across lanes. Bidirectional arrows connect each Micro-Processor to its ARINC 629 Interface. The ARINC 629 Interfaces are connected to a common bus labeled L, C, and R at the bottom of the diagram.

Pratiques industrielles chez Boeing : Assemblage des Briques de base

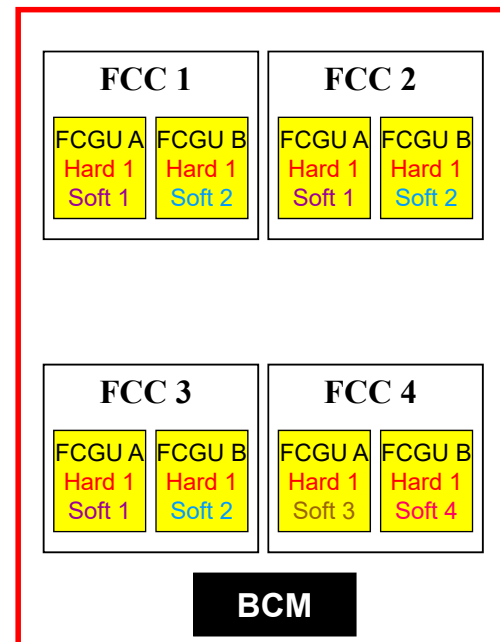
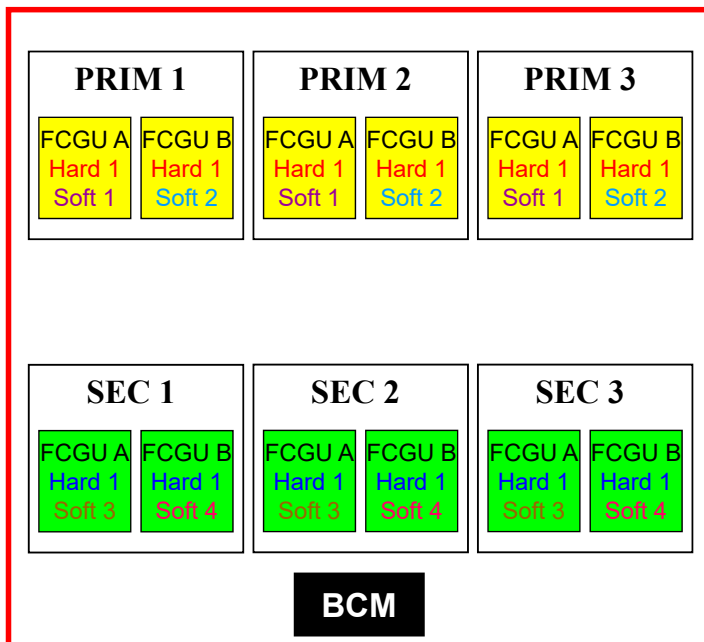


Pratiques industrielles chez Airbus : briques de base

-
- Le diagramme illustre l'architecture d'un système de commande et de monitoring, divisé en deux sections principales : COMMANDE (Commande) et MONITEUR (Monitoring), séparées par un cloisonnement.
- Section COMMANDE (Commande) :**
- Reçoit une alimentation de 28 VDC.
 - Comprend un **Processeur**, de la **RAM ROM**, et des **Entrées/Sorties**.
 - Un **Chien de garde** (Watchdog) est connecté au processeur et aux entrées/sorties.
 - Les entrées/sorties sont connectées à un **relais**.
- Section MONITEUR (Monitoring) :**
- Reçoit également une alimentation de 28 VDC.
 - Comprend un **Processeur**, de la **RAM ROM**, et des **Entrées/Sorties**.
 - Un **Chien de garde** (Watchdog) est connecté au processeur et aux entrées/sorties.
 - Les entrées/sorties sont connectées à un **relais**.
- Les deux sections sont connectées à un bus commun, et les relais de la section MONITEUR envoient des **Servocommandes** à la section COMMANDE.

A380

M400



Bilan Architecture systèmes CDVE Airbus et Boeing

	Airbus	Boeing
Type de redondance	COM/MON + remplacement	Vote majoritaire
Logiciel fonctionnel	4 (2 PRIM + 2 SEC)	3 PFC
Matériel	<ul style="list-style-type: none"> • 12 unités NUMERIQUES, 2 types • 1 BCM 	<ul style="list-style-type: none"> • 9 unités NUMERIQUES, 3 types • 4 unités ANA (ASIC), • 1 Ultime secours

Glossaire

- ACE : Actuators Control Electronics
- FCC : Flight Control Computer
- FCCs : Flight Control Computer secondary
- FCGU : Flight Control and Guidance Unit
- FCPC : Flight Control Primary Computer
- FCSC : Flight Control Secondary Computer
- PA : Pilot Automatic
- PRIM : PRIMary System
- SEC : SECondary Computer
- JAR : Joint Aviation Requirements
- FAR : Federal Aviation Regulations
- IMA : Integrated Modular Avionics
- SAO : Spécification Assistée par Ordinateur. L'atelier SAO est utilisé par l'avionneur pour notamment concevoir les spécifications fonctionnelles de calculateurs.
- Scade : même fonction que l'atelier SAO, et est aujourd'hui utilisé uniquement sur le FCSC A340-500/600
- SSA : System Safety Assessment : A systematic comprehensive evaluation of an implemented system to show that the relevant safety requirements are met.



■ 1. Définition générique

Partant du constat que depuis toujours beaucoup de systèmes présentent un jour ou l'autre des dysfonctionnements, plus ou moins "importants" ...

"La **sûreté de fonctionnement** d'un **système** est la **propriété** (de ce système) qui permet à ses **utilisateurs** de placer une **confiance justifiée** dans le **service** qu'il leur délivre (Laprie 96)".

- ⇒ **Service** = comportement du système tel qu'il est **perçu** par les utilisateurs
 - **Fonction** = ce à quoi le système est destiné
 - **Comportement** = ce que fait le système pour accomplir sa fonction
- ⇒ **Utilisateur** = un autre système (physique ou humain) avec lequel il interagit
- ⇒ **Système** = tout type de système (physique, logiciel, ..., mixte)
- ⇒ **Propriété** : comment la formuler, prouver, vérifier, évaluer, garantir ???
- ⇒ **Confiance justifiée** : pourquoi ne peut-on pas naturellement avoir confiance, et comment définir et apprécier le "justifié" ???

■ 2. Les Systèmes actuels



2.1. Complexité ↗ et criticité ↗

COMPLEXITÉ (nombre et types de composants et de fonctions à assurer) et
CRITICITÉ ("coûts" d'une défaillance : économiques, humains, confort, ...)

- ⇒ **Interdisciplinarité** : nécessité d'utilisation conjointe de théories, modèles et technologies issues de plusieurs domaines
 - Ex. : téléphones mobiles (électronique, informatique), engins spatiaux (informatique, hydraulique, mécanique, automatique), biopuces (informatique, électronique, biologie, chimie)
- ⇒ **Répartition** : des différentes "fonctions" constituant un système ou une application, sur plusieurs "éléments" matériels/logiciels
- ⇒ **Réactivité** : interactions des systèmes avec leurs environnements, pour certains, très **inconnus** et/ou **imprévisibles** (ex. : sondes spatiales), et/ou nécessitant des **temps de réaction** de plus en plus courts
- ⇒ **Criticité** : généralement accrue pour les systèmes complexes temps réels
- Impossible de tenir compte de toutes les situations possibles ⇒ **besoins de "bonnes" méthodes et outils, d'organisation et de coordination entre tous les acteurs**
- Ex. sonde Mars Polar Lander (1999). A la descente, communications coupées comme prévu, mais jamais reprises
 - hyp. : déploiement des pieds ⇒ signaux parasites interprétés comme détection du sol ⇒ arrêt des moteurs !
 - Parasite normal, mais dont l'interprétation n'avait pas été prévue **pour cette étape (de descente)**, car lors des tests, les capteurs de sol avaient été mal branchés, et le problème n'était pas apparu
 - ⇒ mauvaise conception, suite à une mauvaise configuration de test.



2.2. La part du logiciel dans les systèmes

a) Nature du Logiciel

- ⇒ **Immatériel** : pas de coût de production, uniquement de développement
- ⇒ **Très sensible aux erreurs : comportement discontinu du logiciel** → un bit à 0 ou 1 (différence minime) peut faire la différence entre succès et catastrophe → pas de zone de tolérance (comme par exemple en mécanique), pas d'interpolation possible.
- ⇒ **Pas d'usure : un logiciel ne s'use pas au sens courant** → pas de fatigue, pas de signe avant coureur (genre pneu qui s'use, précision qui diminue, ...) → les fautes sont là **dès le début et ne sont donc dues qu'à la conception**.
- ⇒ **Impossible à tester complètement** car combinatoire trop élevée.
- ⇒ **Logiciel basé sur du langage** → **pas de super-structure naturelle** (genre avion : ailes clairement séparées, "par nature", du cockpit, du fait de sa fonction) → la structure d'un logiciel doit y être mise volontairement, par le concepteur.

b) Logiciel = première source de défaillance des systèmes informatiques

■ 3. Sûreté de Fonctionnement



❑ **Besoins NON fonctionnels** : besoins qui ne sont pas directement liés à l'accomplissement de la mission ou de la fonction du système

- Exemples : lisibilité (d'un code, d'une documentation), maintenabilité, extensibilité, confidentialité, ...

❑ **Activité transversale** aux différents domaines et étapes de la vie d'un système

❑ **Nombreux points de vues !**

- ⇒ La SdF du logiciel, du matériel, des deux ?
- ⇒ A quelle « aptitude » du système s'intéresse-t-on davantage :
 - être prêt à délivrer un service, ou à assurer la continuité du service ?
 - ne pas engendrer d'événements catastrophiques ?
 - ne pas permettre de modification ou de divulgation non autorisée d'information ?

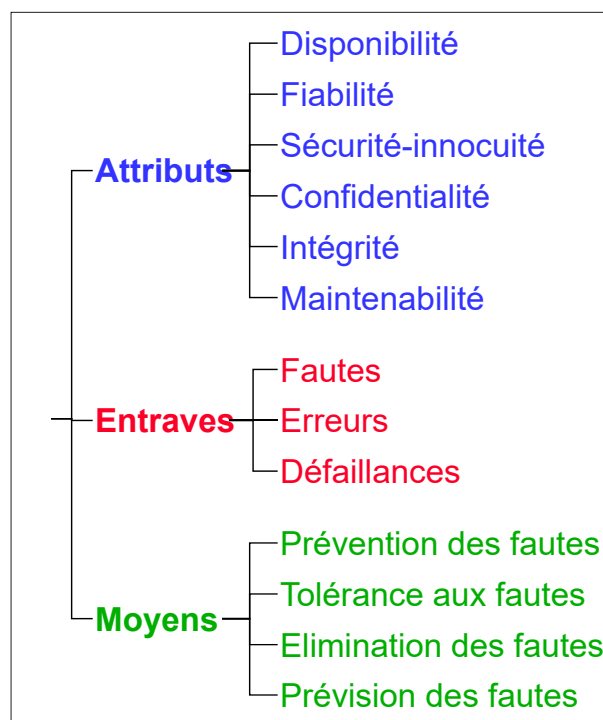
⇒ **Besoin d'une taxonomie** → **"orientations" des définitions présentées ici** :

- ⇒ point de vue de l'utilisateur et du service qui lui est rendu
- ⇒ plutôt SdF "informatique"



Premier niveau de la taxonomie de la SdF → 3 concepts :

- 1) Les **ATTRIBUTS** de la SdF : les capacités (aptitudes) auxquelles on s'intéresse
→ **Fiabilité, disponibilité, ...**
- 2) Les **ENTRAVES** de la SdF : ce qui entrave, nuit, s'oppose, gêne, ..., la SdF
→ **Fautes, erreurs, défaillances**
- 3) Les **MOYENS** de la SdF : les différentes méthodes pour améliorer la SdF, et leurs "points" d'application dans le cycle de vie d'un système
→ **Prévention des fautes, ..., tolérances aux fautes**



■ 1. ATTRIBUTS de la sûreté de fonctionnement (*dependability*)



Selon les points de vue (ou domaines d'application), on s'intéresse à la CAPACITÉ, ou APTITUDE, du système à :

- | | | |
|---------------------------------------------------------------------------------|-----------------------------|----------------------------|
| ⇒ Être prêt à délivrer le service | ⇒ disponibilité | (<i>availability</i>) |
| ⇒ Assurer la continuité du service | ⇒ fiabilité | (<i>reliability</i>) |
| ⇒ Pouvoir être réparé et évoluer | ⇒ maintenabilité | (<i>maintainability</i>) |
| ⇒ Ne pas provoquer de catastrophe | ⇒ sécurité-innocuité | (SAFETY) |
| ⇒ Éviter les divulgations illicites d'informations | ⇒ confidentialité | (<i>confidentiality</i>) |
| ⇒ Éviter les altérations illicites d'informations | ⇒ intégrité | (<i>integrity</i>) |
| ⇒ Confidentialité + Intégrité + Disponibilité (vis-à-vis des actions autorisée) | ⇒ sécurité-immunité | (SECURITY) |
| ⇒ SdF par rapport aux fautes externes | ⇒ robustesse | (<i>robustness</i>) |

□ Remarques :

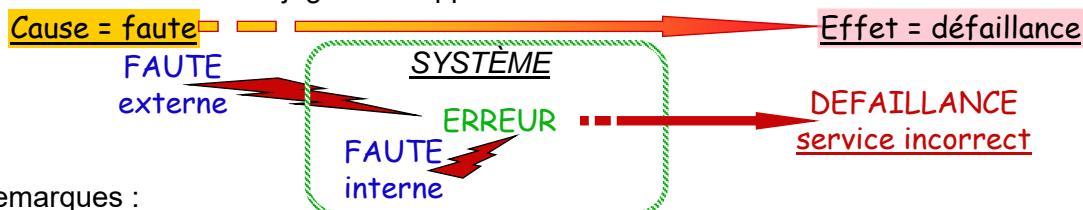
- Fiabilité : alors que le service est rendu, capacité à le rendre sans discontinuité "illicite" = non autorisé (sans précision sur "autorisé")
- Attributs : primaires, composites, secondaires
- Attributs : complémentaires et antagonistes → choisir un compromis
- Problème de vocabulaire entre les mots anglais et français de certaines notions

■ 2. ENTRAVES de la sûreté de fonctionnement



2.1. Premières définitions : classification en 3 types corrélés

- ① **DÉFAILLANCE** : le service délivré par le système dévie de l'accomplissement de la fonction du système
 - "fonction" = ce à quoi le système est destiné, pas nécessairement ce qui a été spécifié
 - "déviation" constatée (perçue) par l'utilisateur \equiv simplement un "symptôme", insuffisant en soit pour nécessairement identifier le problème, et encore moins pour le résoudre.
- ② **ERREUR** : partie de l'état du système susceptible d'entraîner une défaillance
- ③ **FAUTE** : cause adjudgée ou supposée d'une erreur



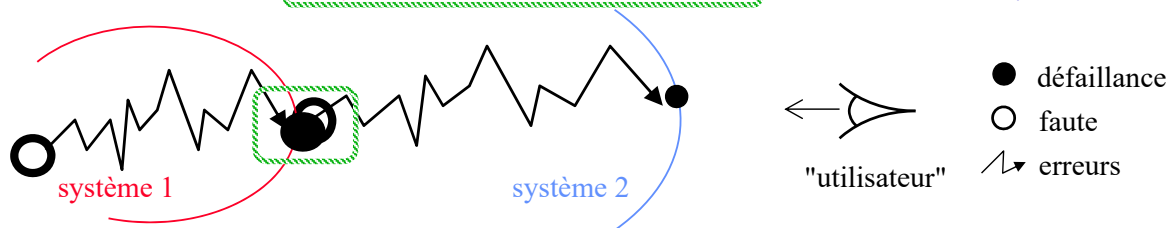
Remarques :

- Si on décompose le systèmes en sous-systèmes (composants), la **défaillance** d'un composant est une **faute interne** pour le système et une **faute externe** pour les composants avec lesquels il interagit.
- Concept de "Faute" proche de celui de "panne". Mais "panne" est très orienté matériel, et on ne parle pas de "panne de conception", ou "panne de manipulation"

2.2. Définitions récursives : relativité aux "délimitations" considérées



... Faute 1 → Erreur 1 → Défaillance 1 ... devient ... Faute 2 → Erreur 2 → Défaillance 2 ...



- Un système peut être "l'utilisateur" d'un autre système :
 - tant qu'une erreur n'atteint pas l'interface avec "l'utilisateur", il n'y a pas de défaillance ... du point de vue de l'utilisateur
 - quand le système 2, utilisateur du service délivré par le système 1, perçoit une défaillance de ce service, alors cette défaillance du système 1, devient une faute dans le système 2, etc.
- Faute = "**là où tout commence**" → dépend jusqu'à quel point on **veut/peut** remonter
 - Ex. **d'Ariane 5** : on peut dire Faute = *overflow* d'une variable. Or un *overflow* ne se produit pas par hasard, mais parce que le programme a mal fonctionné → donc Faute = mauvais programme. Or le programme était parfait pour ce qu'il était censé faire (Système Référence Inertiel **Ariane 4**), donc problème de mauvaise réutilisation (faute de conception).

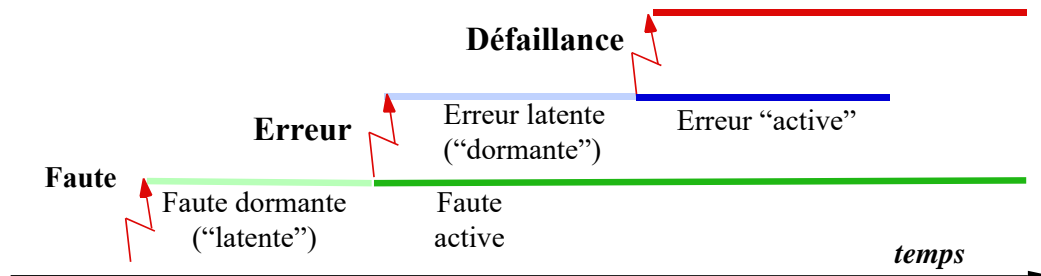
Remonter jusqu'au point où l'on identifie un phénomène INDÉSIRABLE (chose pas faite comme elle aurait dû l'être) qu'on peut ÉVITER ou CORRIGER.

(c'est vague, mais il est important de ne jamais se laisser enfermer dans un modèle trop restrictif)

2.3. Autres concepts et terminologie associés aux entraves



- ⇒ **Dormance de FAUTE :**
 - une faute est **dormante** tant qu'elle ne produit pas d'erreur
 - une faute est **active** dès qu'elle produit une erreur
- ⇒ **Latence d'ERREUR :** une erreur est **latente** (on pourrait aussi dire « dormante ») tant qu'elle n'est pas détectée et qu'elle n'affecte pas le service.



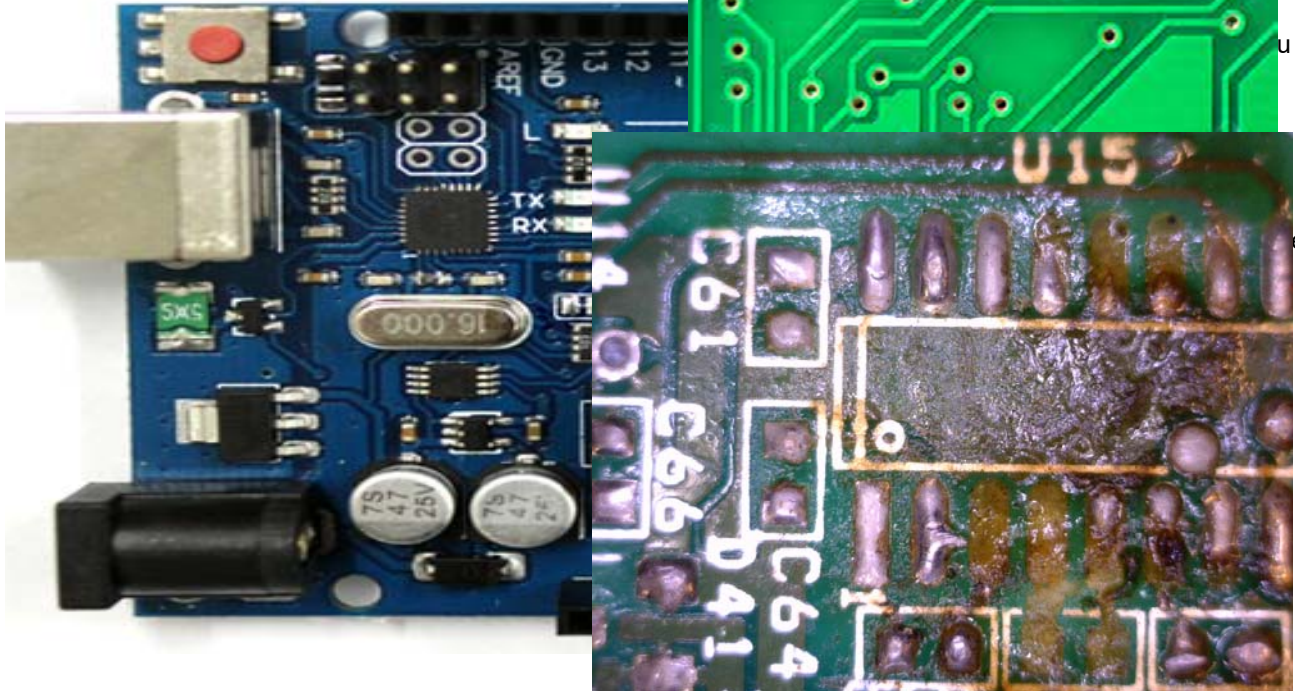
- ⇒ Erreur = "Partie de l'état du système **susceptible** de causer une défaillance du système " → Toute Erreur ne débouche pas sur une défaillance. Cela dépend de :
 - ① l'existence et utilisation de redondance(s) (intentionnelle ou non)
 - ② de l'activité du Système → une Erreur peut être écrasée, ou jamais "activée"
 - ③ la sensibilité des utilisateurs, qui, entre 2 utilisateurs, varie en termes :
 - d'attentes (exigences) de Qualité de Service (QoS) : pour l'un, ce sera une nuisance supportable, pour l'autre, ce sera défaillance
 - de Sensibilité Temporelle : Granularité Temporelle
 - de Phénomène possible d'Accumulation (exemple : capteur + erreurs temporelles)

⇒ Important de définir des seuils acceptables de dégradation du service dans la spécification

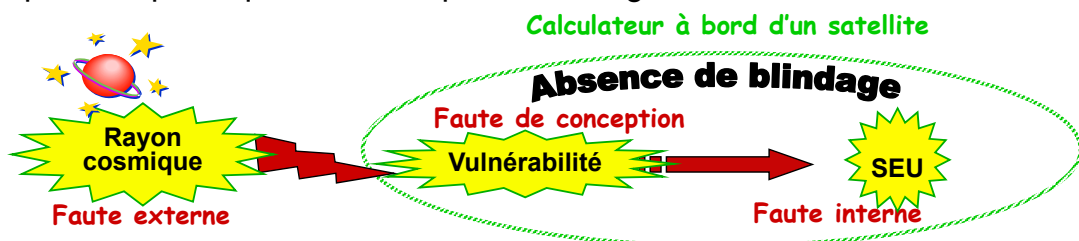
2.4. Quelques exemples de principes

① Faute physique : un court-circuit

- dans un circuit intégré, la migration d'un métal peut provoquer une fonction logique → **défaillance** du circuit

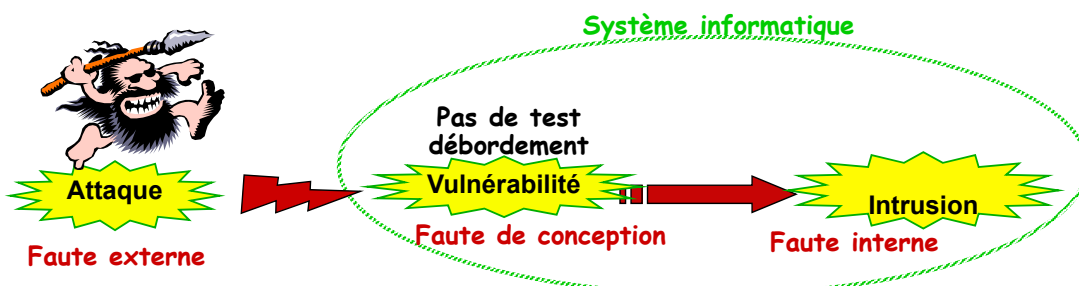


⑤ Exemple de SEU (Single Event Upset) : rayons cosmiques et ions lourds peuvent provoquer des bit-flips, des collages ou des modifications de circuits



⑥ INTRUSION

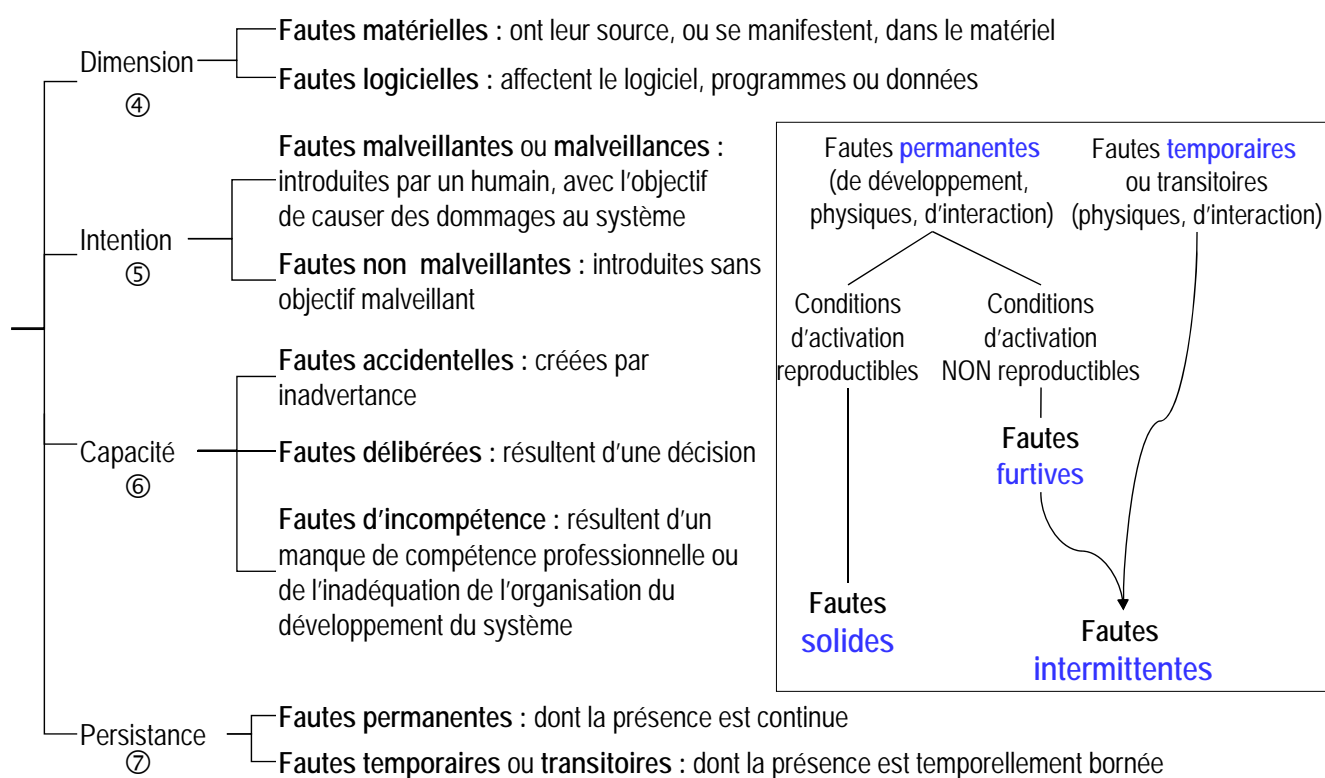
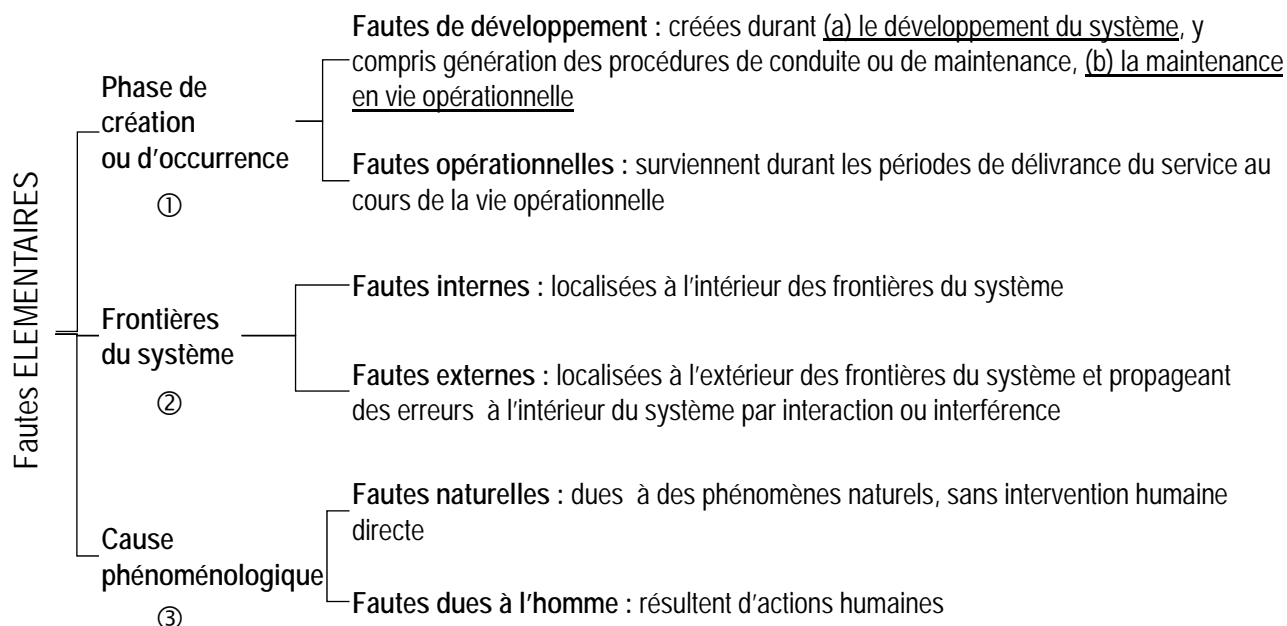
- **Attaque** = faute d'interaction délibérée
- **Intrusion** = faute interne résultant d'une attaque



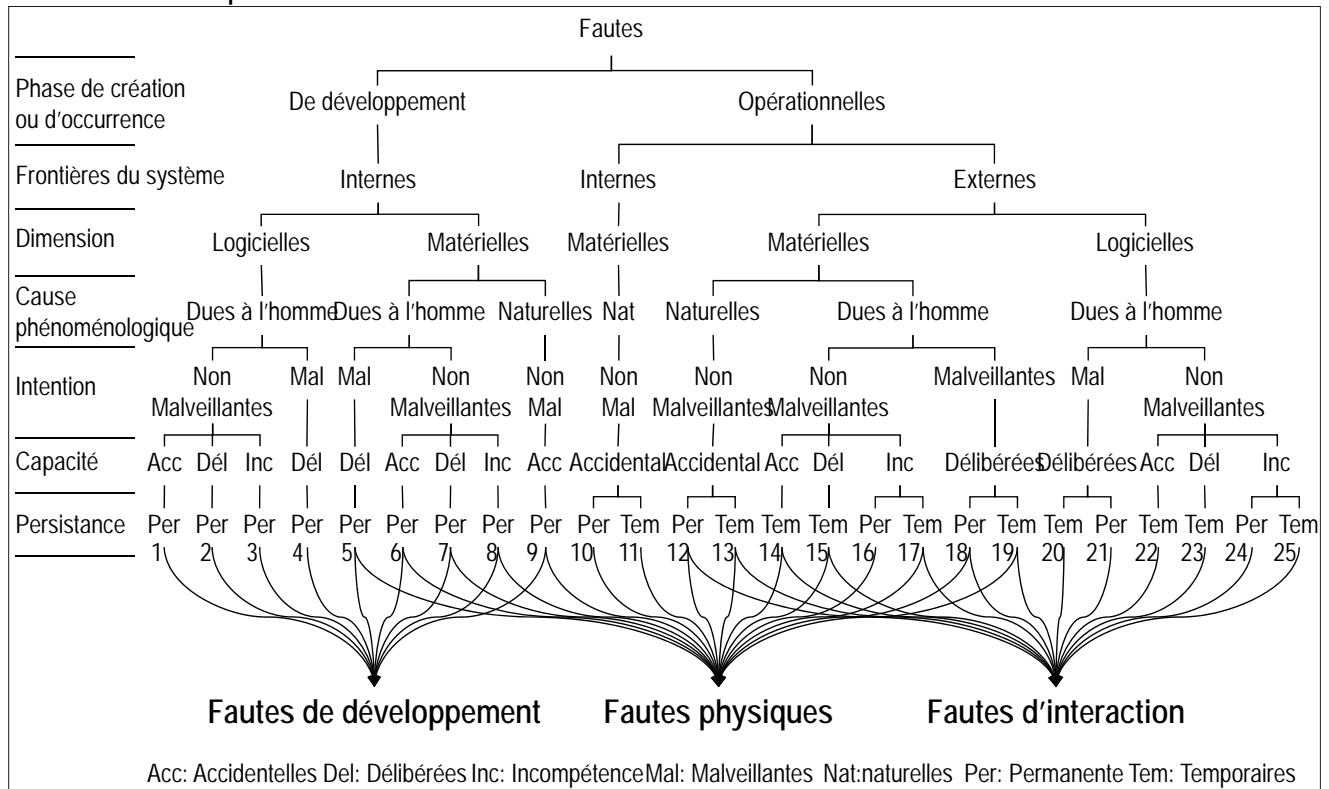
2.5. Classification des FAUTES



- ❑ **Fautes ELEMENTAIRES : classification ...** selon 7 points de vue : l'origine, l'intention, la phase de création, la localisation, la persistance, ...



❑ **Fautes COMBINEES : classification ... plusieurs dizaines de classes : un exemple avec 25 classes ...**



2.6. Classification des DEFAILLANCES





- ⇒ Défaillance : définie par rapport aux utilisateurs et non à la spécification du système
- ⇒ Un système peut défaillir de différentes façons ⇒ **modes de défaillance**

① Domaines de défaillance

- **en valeurs** : valeur incorrecte d'un résultat (hors du domaine fixé pour les valeurs "admissibles")
- **temporel** : service non délivré aux dates (ou délais) prévus → soit en avance (trop tôt), soit en retard (trop tard)
- **"silence sur défaillance"** = cas particulier de défaillance par arrêt (figement ou silence)

② Sévérité des défaillances

- défaillances **bénignes** : écart minime entre « coût » des conséquences et « bénéfices » du service
- défaillances **catastrophiques** : écart ... « énorme »
- **Criticité** = liée au mode de défaillance le plus grave

③ Cohérence des défaillances

- Défaillances **Cohérentes**
- Défaillances **Incohérentes (Byzantine)** (les plus dures à détecter !) → Les différents utilisateurs perçoivent des comportements différents → décisions incohérentes
 - Aussi appelées (généralement à tort) : défaillances (ou fautes) arbitraires ou malicieuses

2.7. Types d'exemples concrets

- 06/80 : Fausses alertes au NORAD (North American Air Defense)
- 06/81 : Doses excessives de radiothérapie (Therac-25)
- 01/90 : Téléphone interurbain des USA indisponible pendant 9 h
- 11/92 : Écroulement du système de communication des ambulances de Londres
- 06/93 : Blocage des autorisations de cartes bancaires en France pendant un week-end
- 06/96 : Échec du vol 501 d'Ariane 5
- 07/97 : Blocage du service de nommage (DNS) d'internet
- 02/00 : Attaques concertées de Yahoo, Amazon, CNN, E-Bay, ...
- 03/01 : Extorsion de fonds par des groupes de hackers ukrainiens et russes auprès de 40 compagnies US de vente sur le web
- ...
- ...
- ...
-



2.8. Précisions sur les notions de criticité et de sévérité

- Un système est dit **CRITIQUE** si une de ses **défaillances peut conduire** à un **événement jugé CATASTROPHIQUE** sur le plan humain, économique ou environnemental.
- « **Défaillance catastrophique** » : *“défaillance dont les conséquences sont incommensurablement différentes du bénéfice procuré par le service délivré en l’absence de défaillance”* [Laprie et al. 1996].

Intérêt de cette définition : elle couvre la diversité de points de vue sur le caractère *catastrophique*, en ramenant son appréciation à l’appréciation de la “**distance/écart**” entre le **service nominal** et **celui rendu en cas de défaillance**.

Problème de cette définition : abstraite, inutilisable directement sans être précisée par rapport au système auquel on veut l’appliquer.

- D’abord définir la notion de **SÉVÉRITÉ** ou **GRAVITÉ** des défaillances : déterminée à partir du **classement** des conséquences des défaillances sur l’environnement du système. Les modes de défaillance sont *ordonnés en niveaux de sévérité, auxquels sont généralement associés des probabilités maximales d’occurrence admissible*.
- Nombre, dénomination et définition des niveaux de **SÉVÉRITÉS** des défaillances et des probabilités associées → variables selon le domaine d’application :
 - **4 dans l’aéronautique civile** : mineure, majeure, dangereuse, catastrophique,
 - **3 dans la production d’énergie nucléaire** : incidents mineurs, incidents, accidents graves,
 - **4 pour les lanceurs spatiaux** : significatif, majeur, grave, catastrophique,
 - **4 dans l’automobile** : mineure, majeure, grave, catastrophique,
 - **4 dans le ferroviaire** : pas de dénominations à notre connaissance.



- Extrait du guide de la sûreté des systèmes de l’administration fédérale de l’aviation américaine [FAA 2000], décrit le cas de l’aviation civile.

Dénomination des défaillances	Conséquences des défaillances	Probabilité d’occurrence par heure	
Mineure	Réduction non significative de la sécurité de l’avion. Peut inclure : légère réduction des marges de sécurité ou des fonctions, léger accroissement de la charge de travail, quelques inconvénients pour les passagers.	Probable	$> 10^{-5}$
Majeure	Réduction significative des marges de sécurité ou des fonctions, ou augmentation significative de la charge de travail de l’équipage, ou inconfort des occupants avec possibilité de blessures.	Rare	comprise entre 10^{-5} et 10^{-7}
Dangereuse	Grande réduction des marges de sécurité ou des fonctions, ou détresse physique ou grande surcharge de travail, ou blessure fatale ou sérieuse pour un relativement petit nombre de passagers.	Extrêmement rare	comprise entre 10^{-7} et 10^{-9}
Catastrophique	Empêche la continuité de la sécurité à l’atterrissage ou en vol: l’avion, les passagers et l’équipage sont perdus.	Extrêmement improbable	$< 10^{-9}$

- Le niveau de **CRITICITÉ** d’un système : se déduit simplement du plus fort niveau de sévérité de ses modes défaillances.
 - Là aussi, dénominations et nombre des niveaux de criticité dépendent du domaine
 - Ex. : ils sont notés, par ordre décroissant : de A à C dans le nucléaire, de A à D dans l’aéronautique civile, et de SIL 4 à SIL 1 dans le ferroviaire.

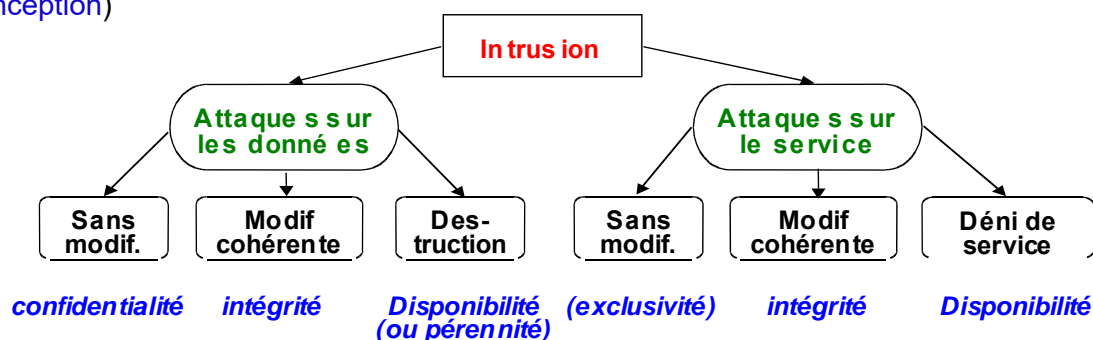


■ NORMES DE CERTIFICATIONS !!!!!

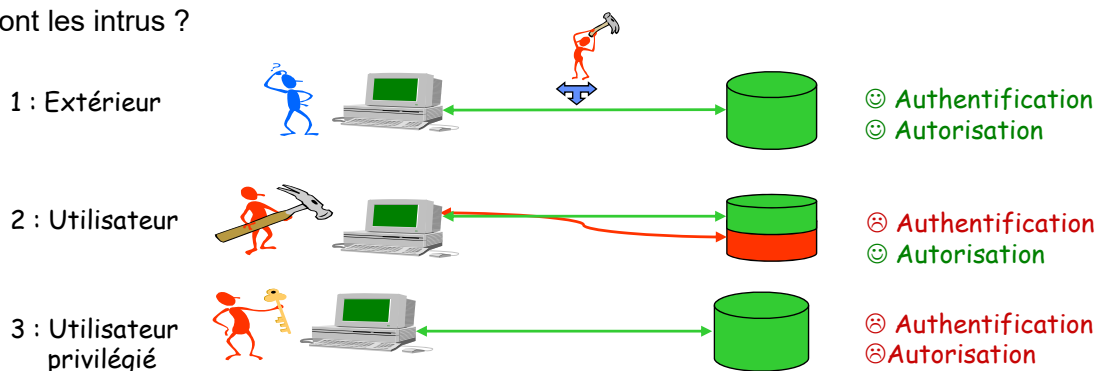
- Classes et dénominations : établies par des **règlements de certification** normalisés, qui sont un processus complexe, induisant un surcoût et ralentissant l'utilisation d'innovations, mais, sont une garantie pour les utilisateurs et l'environnement des systèmes critiques.
 - **Ex. de normes de certification : DO-178B pour l'aéronautique, NF EN 50128 pour le ferroviaire, et ECSS pour le spatial**
 - En effet, le caractère critique induit d'importantes contraintes supplémentaires en termes de SdF lors de la conception et du développement de tels systèmes.
 - La définition et le respect de ces contraintes ne sont pas laissés sous la seule responsabilité du concepteur. Un organisme tiers établit des règlements pour traduire les exigences des autorités de tutelle du domaine du système en question. Puis cet organisme vérifie et valide (ou invalide), d'une part, l'application des règlements et d'autre part, que le niveau imposé de SdF soit atteint pour le système.
- **Au final, pour un système critique, l'objectif principal est d'assurer que le taux d'occurrence d'un événement catastrophique reste inférieur au seuil fixé pour ce système.**

2.9. Précisions sur les Intrusions

⇒ Intrusion = **faute interne** résultant d'une attaque (**faute externe**) et d'une vulnérabilité (**faute de conception**)



⇒ Qui sont les intrus ?



■ 3. MOYENS de la sûreté de fonctionnement



MOYENS = Méthodes, outils et solutions pour :

"Spécifier, concevoir, réaliser et exploiter des systèmes où la faute est naturelle, prévue et tolérable." [Laprie et al.]

3.1 Première taxonomie : 4 classes de moyens regroupées en 2 catégories

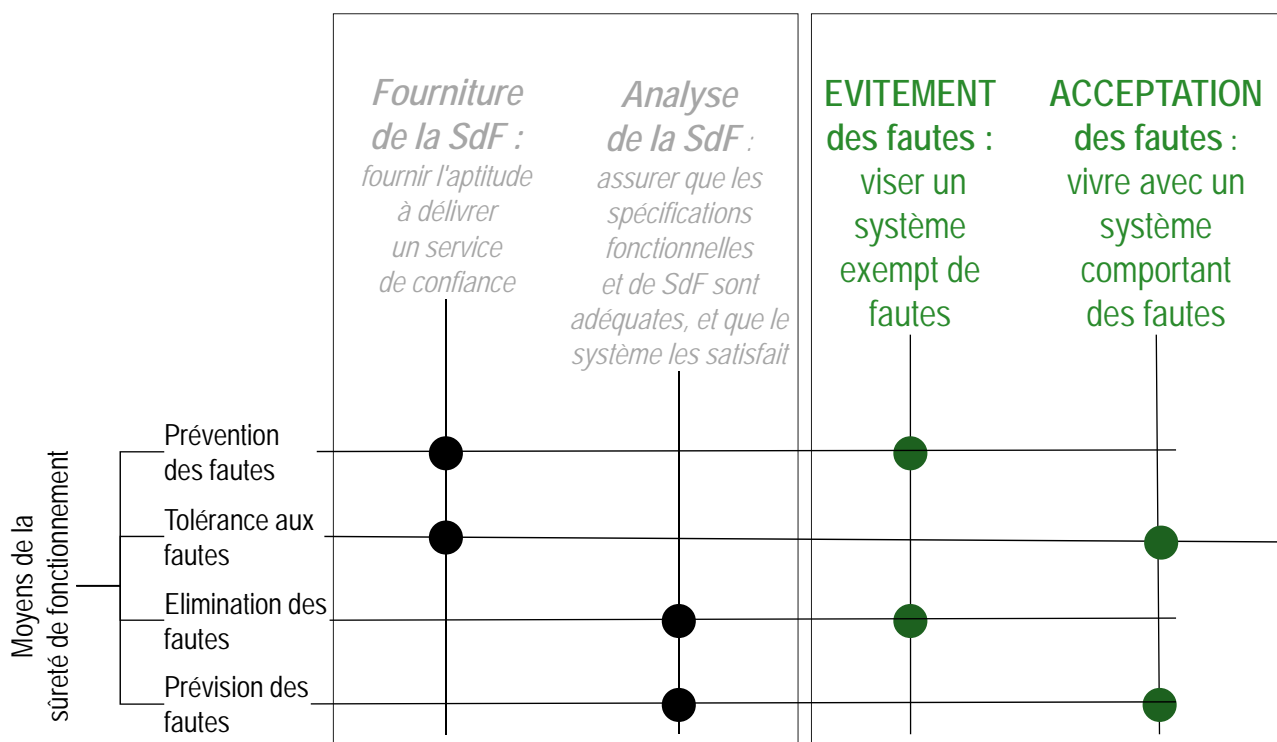
1. **FOURNIR** au système l'aptitude à délivrer un service conforme à l'accomplissement de sa fonction

- ① **Prévention** des fautes : empêcher, par **construction**, l'occurrence ou l'introduction de fautes
- ② **Tolérance** aux fautes : fournir, par **redondance**, un service conforme à l'accomplissement de la fonction, en dépit des fautes

2. **VALIDER** le système, pour donner confiance dans cette aptitude

- ③ **Élimination** des fautes : réduire, par **vérification**, la présence (nombre, sévérité) de fautes : preuve, test
- ④ **Prévision** des fautes : estimer, par **évaluation**, la présence, la création et les conséquences des fautes

3.2. Autres regroupements des moyens, selon les objectifs visés



3.3. Un exemple de Moyen : la tolérance aux fautes



Objectifs : **Détection de l'Erreur** puis « **Recouvrement** » de l'Erreur

a) La Détection d'Erreur

⇒ **Détection : Basée sur la REDONDANCE & La DIVERSIFICATION (dissemblance)**

- De l'information (fautes de transmission, de stockage)
- Du logiciel (fautes de conception)
- Du matériel (fautes physiques)

⇒ **Principe : "répliquer" l'entité dont on veut tolérer les défaillances**

- Modes de Défaillances non Corrélés (Diversité)
- Si une réplique défaille, l'autre continue de délivrer service correct.
- Permet de repérer l'erreur, et éventuellement de la masquer.

⇒ **Redondance de l'information : Reprendre toute ou partie de l'information**

- ex. : Codage Binaire → du plus simple « Bit de Parité » ... à l'autre Extrême → Duplication

⇒ **Redondance Logiciel (fautes de conception)**

- Dans le même programme
- Avec des programmes différents : Diversification Fonctionnelle → Plusieurs équipes développent séparément la même fonctionnalité

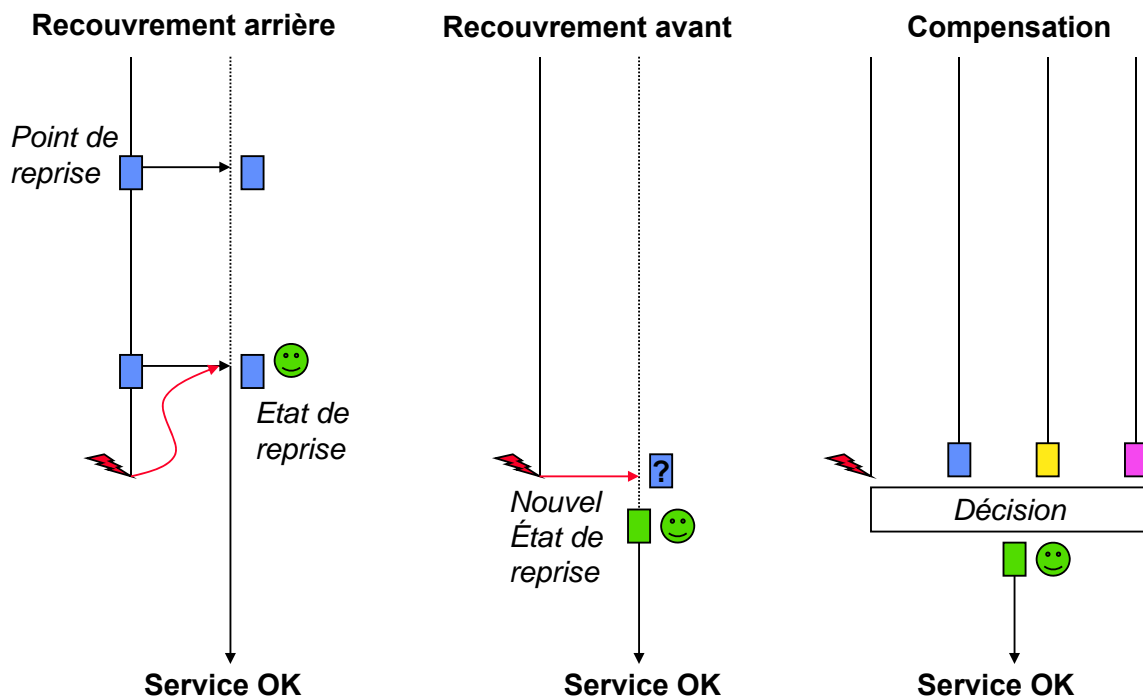
⇒ **Redondance Matériel**

- Pour détecter / tolérer des Fautes Physiques
- Sans aucun effet sur les fautes de Conception (Ariane 5)

b) Le Recouvrement d'Erreur (Error-Recovery)



⇒ **3 Grandes Approches de recouvrement : par reprise, par poursuite, par compensation**





⇒ **Recouvrement par reprise :**

- Sauvegarde régulière de l'état du système → "points de reprise"
- Quand Détection d'Erreur : retour en arrière au dernier état sauvé
- Reprise à partir de l'état restauré

⇒ **Recouvrement par poursuite :**

- But : après détection, rechercher nouvel état acceptable
- Très dépendant de l'application
- A l'extrême : tout arrêter de manière contrôlée ("gracefully") : Possible si l'état "arrêté" est sûr
- Nouvel état souvent en modes dégradés (modes de survie)
- Exemple : Réinitialisation / Relecture de tous les capteurs

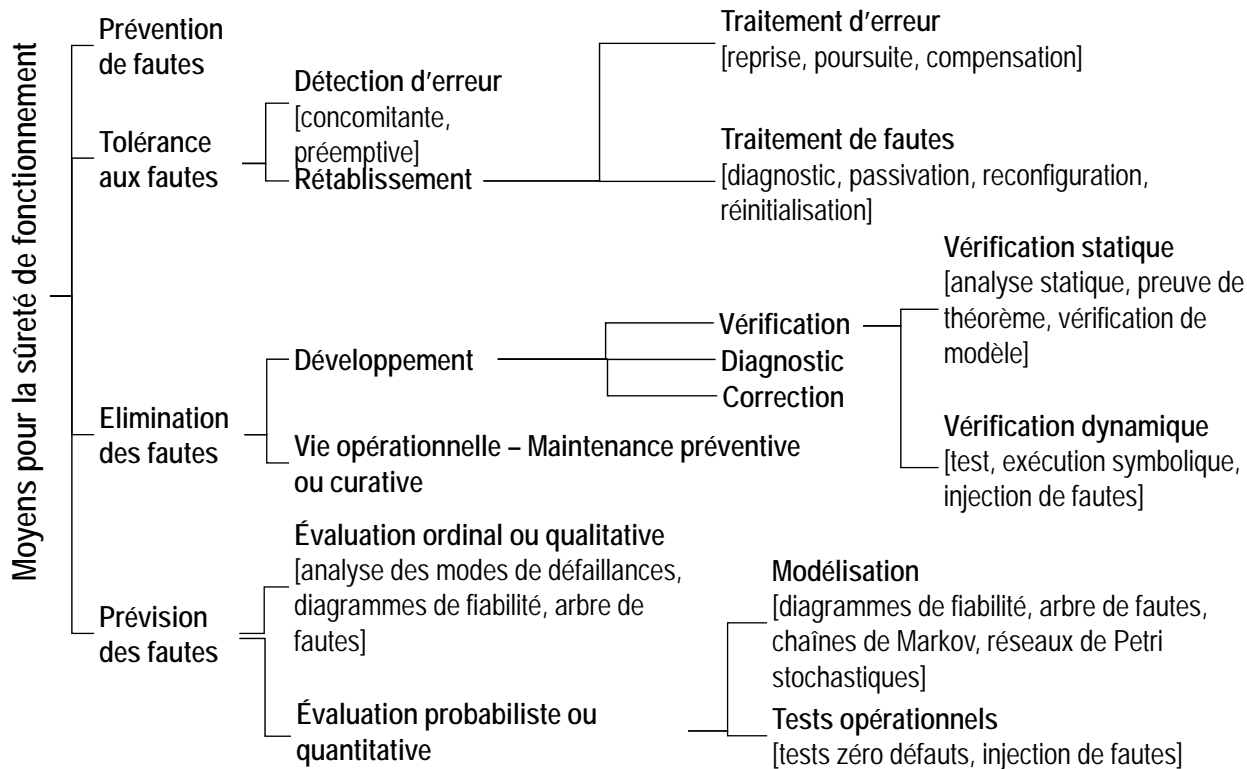
⇒ **Recouvrement par Compensation / Masquage**

- Uniquement possible avec suffisamment redondance
- Défaillance d'un composant compensée par 1 ou plusieurs répliques : soit après détection de l'erreur, soit automatiquement (vote)

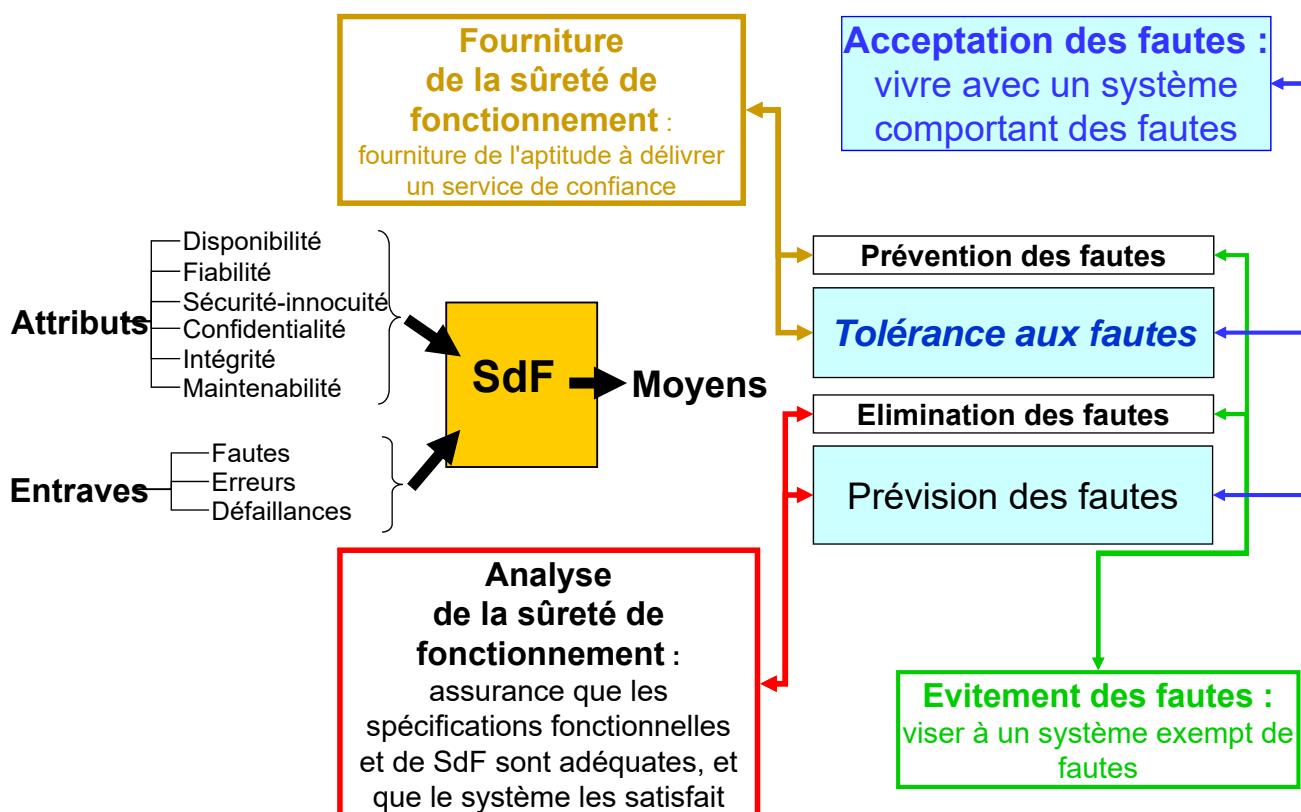
⇒ **Détails sur masquage par vote majoritaire**

- $2N+1$ répliques permette de masquer N défaillances.
- Suppose les traitements déterministes
- Si les répliques sont à silence sur défaillance : problème grandement simplifié (avec M répliques, on peut tolérer $M-1$ défaillances)

3.4. L'arbre des Moyens



■ 4 . Points de vue synthétiques sur la SdF



Sûreté de Fonctionnement (SdF)

- 2^{ème} partie : les MOYENS -

- 4 – PRÉVENTION des FAUTES
- 5 – PRÉVISION des FAUTES
- 6 – ÉLIMINATION des FAUTES
- 7 – TOLÉRANCE des FAUTES

4 – MOYENS de la SdF : La PRÉVENTION des fautes



- Objectif de la PRÉVENTION : réduire ou empêcher l'introduction ou l'occurrence de fautes
- Comment, pour quelles types de fautes ???
 - ⇒ fautes de conception : méthodes de développement qui minimisent le risque d'introduction de fautes (ex: génie logiciel)
 - ⇒ fautes de fabrication : qualité, vérification
 - ⇒ fautes d'interaction (installation, opération, maintenance) : ergonomie des interfaces, protection, qualité de la documentation, formation, etc.
 - ⇒ fautes externes : isolation, blindage, etc.
 - ⇒ fautes physiques internes : surdimensionnement, qualité, protection, etc.
- Note : pour du logiciel, les fautes sont toutes des fautes de « conception »




■ 1. Objectifs de la PRÉVISION

- ⇒ Évaluer présence, création, et conséquences des fautes ou erreurs sur la sûreté d'un système (construit ou à construire)
- ⇒ Guider la Conception d'un Système à Construire
- ⇒ Valider / Certifier un Système Construit

■ 2. Concepts et vocabulaire

- ⇒ Le Mot "risque" est relativement vague ⇒ distinguer :
 - l'événement dangereux (en terme de sa probabilité d'occurrence)
 - des dégâts conséquents (le coût de l'occurrence),
 - de la combinaison des deux.
- ⇒ Exemple : probabilité de mourir par impacte de météorite \equiv à celle de mourir dans un accident d'avion (selon certains chercheurs ...).

MAIS, en fait la probabilité qu'une météorite tombe est bien plus faible que celle d'un crash d'un avion. Cependant, ce dernier fait beaucoup plus de victimes.

- ⇒ Probabilité acceptable pour une Défaillance lié à la Gravité des conséquences de l'occurrence de la défaillance (\neq Coût!) 
- ⇒ Gravité de la défaillance : humaine, économique, environnementale
- ⇒ Criticité = Couple (Gravité, Probabilité Acceptable)
 - une criticité par Mode de Défaillance
 - niveau de Criticité du Système = LA plus haute des Criticités
- ⇒ Risque
 - "Danger éventuel, plus ou moins prévisible, inhérent à une situation ou à une activité"
- ⇒ Danger
 - "Situation où une personne (ou un pays) est menacé dans sa sécurité, ou, le plus souvent, dans son existence."
 - "Ce qui constitue une menace pour la tranquillité ou l'existence même d'une personne (ou d'un pays)."
 - "Être **EN** Danger" \neq "Être **UN** Danger"

■ 3. Les méthodes de la PRÉVISION



a) 2 familles de méthodes

- ⇒ **Évaluations Ordinales (dites Qualitatives) : 2 approches**
 - **Inductives** : de causes particulières → vers la défaillance générale
 - **Déductive** : d'une défaillance générale → vers des causes particulières
- ⇒ **Évaluations Probabilistes (dites Quantitatives)**

b) Différence entre Matériel et Logiciel

- ⇒ **Prévision des Fautes Matérielles** → **Fautes** Basée sur des Phénomènes Physiques
 - Théorie bien comprise, largement appliquée en industrie.
 - Mesures "stabilisées" : composants identiques lors de la réparation.
- ⇒ **Prévision des Fautes Logicielles (de conception)** → **Fautes** basée sur des phénomènes humains
 - Techniques fortement basées sur les Tests ("mesures" spécifiques à chaque logiciel)
 - Mesures de "tendance" : car "Réparation" du Logiciel = Nouveau Logiciel.
- ⇒ **Logiciel Embarqué/Enfoui** intimement lié au Matériel → Défaillance Matériel ⇒ Besoins Logiciels

c) Évaluations Ordinales (Qualitatives)



- ⇒ **AMDE(C) (ou FME(C)A)** → Analyse des **M**odes de **D**éfaillance, de leur **E**ffets, (+Criticité)
 - Approche **inductive** : analyse des modes de défaillance de chaque composant → causes, effets, détection, protection → tout ça sous forme de tableaux
 - **Standardisée** (CEI : Commission Électrotechnique Internationale)
 - Limitation : pas de Défaillances Multiples

	Défaut sur produit			Prévu/existant	Cotation			
MD	Défauts potentiels	Effets défauts	Causes défauts	Plan de validation	V	O	S	C
	Four pas assez chaud	Temps de cuisson plus long Consommation électrique plus importantes Mécontentement du client Aliment non cuit ... etc	Résistance pas assez puissante	Tester la cuisson d'un rôti de porc (ou élément dans la cuisson nécessite un temps de cuisson long et une haute température)				

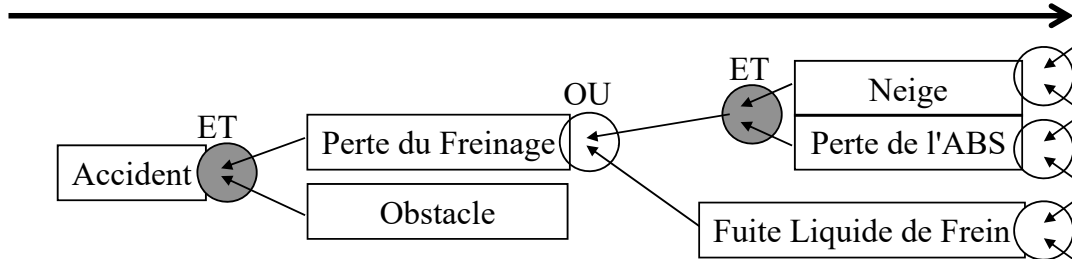
Le but est de déterminer l'effet final. Il faut donc suivre l'enchaînement afin d'obtenir l'effet final.



⇒ A.d.F : Arbre de Fautes

- Approche **Déductive** : sous la forme d'un Arbre avec des Branches ET / OU
- Obtention de "Coupes Minimales" = Ensemble d'événements pouvant entraîner l'événement racine. Une coupe minimale n'en contient pas d'autres.
- Étude des coupes → faire apparaître les événements critiques par rapport aux événements indésirables

Sens de Construction



⇒ A.d.F. et AMDE(C) pour le Logiciel

- Déterminer Fonctions / Objets Critiques
- But : Mise en Place Prévention, mais aussi ... de la Tolérance



d) Évaluations Probabilistes (Quantitatives)

⇒ Questions ???

- Que veut-on calculer ? → exemples :
 - Fiabilité → Temps moyens jusqu'à la première défaillance (MTFF)
 - Disponibilité → Temps moyens entre défaillances (MTBF)
- Quel modèle construit-on ? Sachant que les modèles doivent être : Faciles, Proches de la Réalité et Utilisables pour les calculs.
- Comment calcule-t-on ?

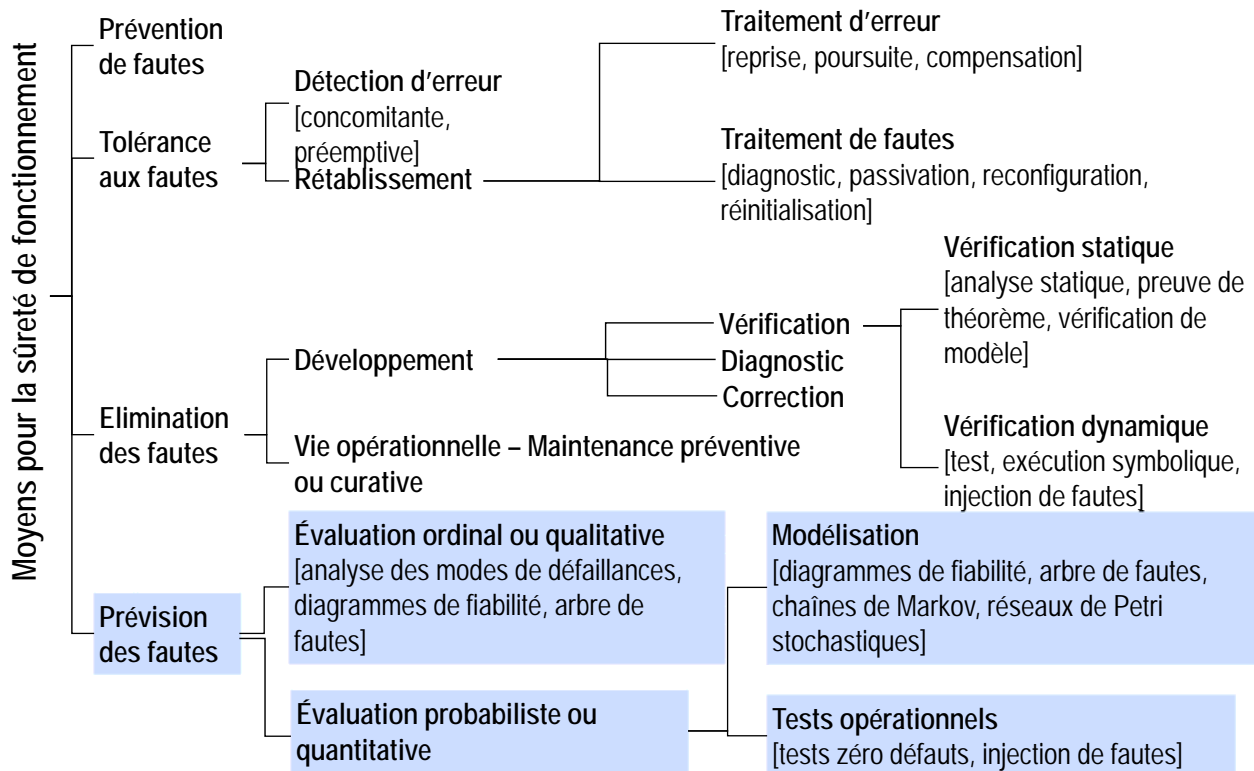
⇒ Modèles pour le Matériel

- Les plus courants : Réseaux de Petri Stochastiques + chaîne de Markov
- Problème général de la Prévision par Modèle :
 - Exactitude des Probabilités de départ difficile à obtenir
 - Explosion d'états ⇒ nécessité de faire des approximations

⇒ Pour le Logiciel

- Fautes de Conception → quasi impossible à modéliser quantitativement « a priori ».
- Utilisation des résultats de tests pour évaluer le comportement dans les conditions opérationnelles.
- Élimination des Fautes Trouvées : la qualité s'accroît → "Modèle de Croissance de Fiabilité." : Évaluer la quantité de fautes résiduelles.

e) Résumé dans l'arbre des Moyens





1. Objectifs de L'ÉLIMINATION de fautes

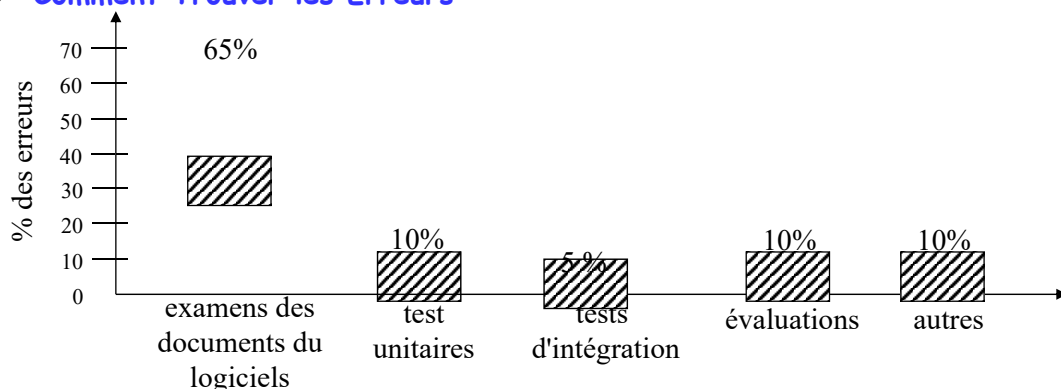
- ⇒ Réduire, par **vérification**, la présence (nombre, sévérité) de fautes : preuve, test
- ⇒ Donc **Éliminer les Fautes le plus tôt possible** → 3 classes de méthodes :
 - 1) **Vérification** (exemple d'application : **Projet Fil Rouge !!!!!**)
 - 2) Diagnostic
 - 3) Modification
- ⇒ RAPPEL : Fautes Logiciels = Fautes de Conception
 - Fautes présentes dès le début du fonctionnement.
 - Fautes introduites tout au long du processus de développement.
 - Élimination → Activités à toutes les étapes :
 - La spécification répond bien aux besoins
 - La conception générale répond bien aux spécifications
 - la conception détaillée répond bien à la conception générale
 - l'implémentation répond bien à la conception détaillée
- ⇒ **Point essentiel : Trouver la Faute avant de l'éliminer**

2. Les méthodes de L'ÉLIMINATION par vérification



a) Les 2 familles de méthodes

- ⇒ **Techniques Statiques (sans exécution)**
 - **Revue et Inspections : faciles, mais encore trop peu utilisées.**
 - Analyse Automatique : bien répandue dans les compilateurs.
 - Vérification formelle, preuve.
- ⇒ **Techniques Dynamiques (avec exécution)**
 - L'implémentation doit déjà avoir commencée.
 - Technique la plus répandue : **le Test**
- ⇒ **Comment Trouver les Erreurs**





b) Techniques Statiques

⇒ Revues et Inspections

- Examens "à la main" des documents produits
- Par plusieurs personnes, indépendantes des auteurs du document.
- Utilisation de "Check-Lists" ("Pense-Bêtes") généraux, ou spécifiques au domaine
- Très peu onéreux (ne coûte que le temps de la réunion) + Très efficaces + Très flexibles (dès le début du processus, sur tout type de doc).

⇒ Analyse Automatique

- Essentiellement sur du code source → donc seulement possible tardivement.
- Évolution : analyse Statique de Modèle dans les AGL (Ateliers de Génie Logiciel)
- Courant dans les compilateur modernes.
- Permet de localiser des fautes potentielles (Code Mort, variables non initialisées, Mesure de Complexité).

⇒ Preuve et Vérification Formelle

- Les programmes ne sont en général pas vérifiables.
- Soit Modèle Simplifiés du Programme (MEF, ≠ Algèbres, ...)
- Soit Restrictions Fortes de la Programmation (pour les logiciels critiques).

c) Techniques Dynamiques (avec exécution) : test / exécution symbolique

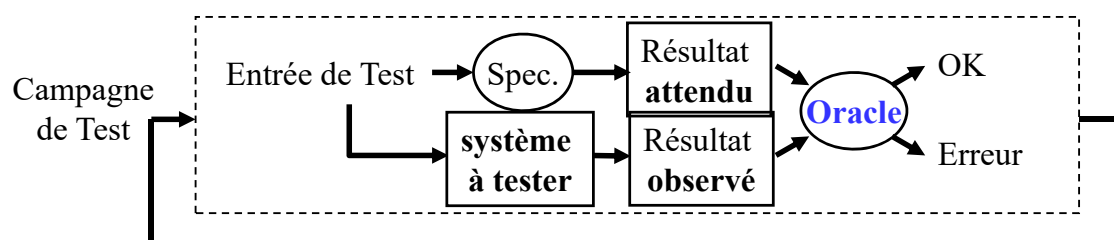
⇒ Pourquoi Tester ?

- Parce que l'Erreur est Humaine (et qu'il y a toujours un humain quelque part)
- Pour le logiciel, un des seuls moyens d'évaluer quantitativement la SdF

⇒ Pour révéler une faute, il faut :

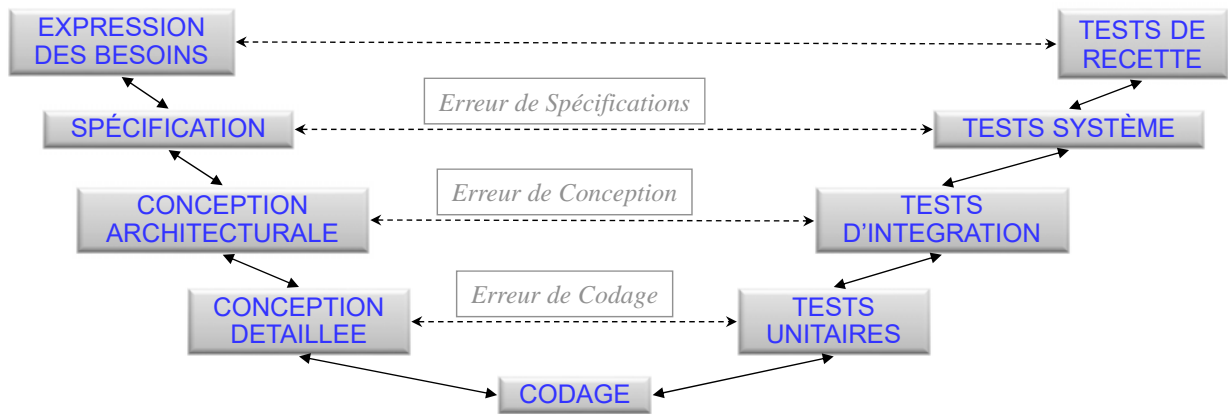
- Que la faute soit activée (donc provoque une erreur) par une des entrées de test
- Que l'erreur se propage pour affecter une sortie **observable**
- Qu'une condition de vérification soit transgressée

⇒ Schéma de principe



⇒ Relation entre domaines d'entrées et de sorties : notions de commandabilité et d'observabilité (lien avec l'automatique)

⇒ Où tester dans le cycle de vie ?



- Quel que soit le type de cycle de vie retenu, il induit les processus de test associés, en interaction avec le processus de « développement » du logiciel
 - Planification des étapes de test associées aux étapes de développement
 - Stratégie d'intégration progressive (ex : conception top-down, test bottom-up)
 - Règles d'indépendance testeur / développeur (selon étape et criticité du logiciel)
 - Règles guidant le choix des méthodes de test à utiliser (selon étape et criticité du logiciel)
 - Procédures pour coordonner les processus
- Des documents sont produits pour chaque étape de test : Méthodes de test utilisées + Jeux de test + oracle + Plate-forme de test : machine hôte, émulateur de la machine cible, machine cible, simulateur de l'environnement externe + Autres outils (compilateur, outils de test, moniteurs (*drivers*) et bouchons (*stubs*) spécifiquement développés) + Compte-rendu du déroulement des tests

⇒ Méthodes de génération des entrées ?

- PB : Exhaustivité impossible → Trouver un échantillon représentatif des entrées.
- représentatif par rapport à quoi ? → Nécessité d'un critère de représentativité pour sélectionner des entrées : critère structurel, critère fonctionnel, autre ?
- Méthodes : statistiques ou déterministes ?

⇒ Tests Déterministes

- Chaque entrée déterminée "à la main"
- Privilégie la qualité sur la quantité
- Difficilement automatisable
- Risque de biais cognitif

⇒ Tests Statistiques

- Méthode (pseudo-) aléatoire de génération des entrées
- Automatisable
- Pas de biais cognitif
- But : éviter des corrélations entre erreurs à révéler et entrées sélectionnées
- Inconvénient : plus long que le test déterministe



⇒ critères de sélection des entrées ?

⇒ Test structurel (boite transparente, "glass box")

- De préférence pour composants de petite taille
- Tous les chemins : impossible en pratique
- Critères de Couverture par rapport à la structure : toutes les instructions ? toutes les branches ? tous les chemins ?

⇒ Test fonctionnel (boite noire, "black box")

- Indépendant de la taille du composant
- Choisir les entrées de tests par rapport aux comportements attendus (les fonctions) du système, mais "fonction" du programme difficile à définir et à modéliser
- Critères de Couverture sans modélisation formelle :
 - activer toutes les fonctions du système au moins une fois,
 - activer toutes les différentes combinaisons d'entrées,
 - activer toutes les formes de sortie,
 - pousser le système dans ses limites de performance (teste de charge : activer les bornes limites des entrées autorisées, activer toutes les situations critiques prévues dans la spécification)
- Modélisation formelle du Comportement
 - beaucoup de possibilités, de formalismes (MEF, RdP, ...)
 - Critères de couverture défini selon le formalisme : dans une MEF, par exemple, tous les états, les transitions, les chemins, ...



⇒ Classification des méthodes de test

⌚ / ⌚	MODELE STRUCTUREL	MODELE FONCTIONNEL
CHOIX SELECTIF	structurel déterministe	fonctionnel déterministe
CHOIX ALEATOIRE	structurel statistique	fonctionnel statistique

⇒ Critère

- Le modèle synthétise l'information dont on dispose sur le programme à tester.
- Le critère indique comment exploiter cette information au cours du test : il définit un ensemble d'éléments du modèle à activer au cours du test.

⇒ procédé de génération des entrées

- Déterministe : choix sélectif d'entrées pour satisfaire (couvrir) le critère.
- Probabiliste : tirage aléatoire selon une distribution des probabilités d'entrée propre à satisfaire rapidement le critère retenu.



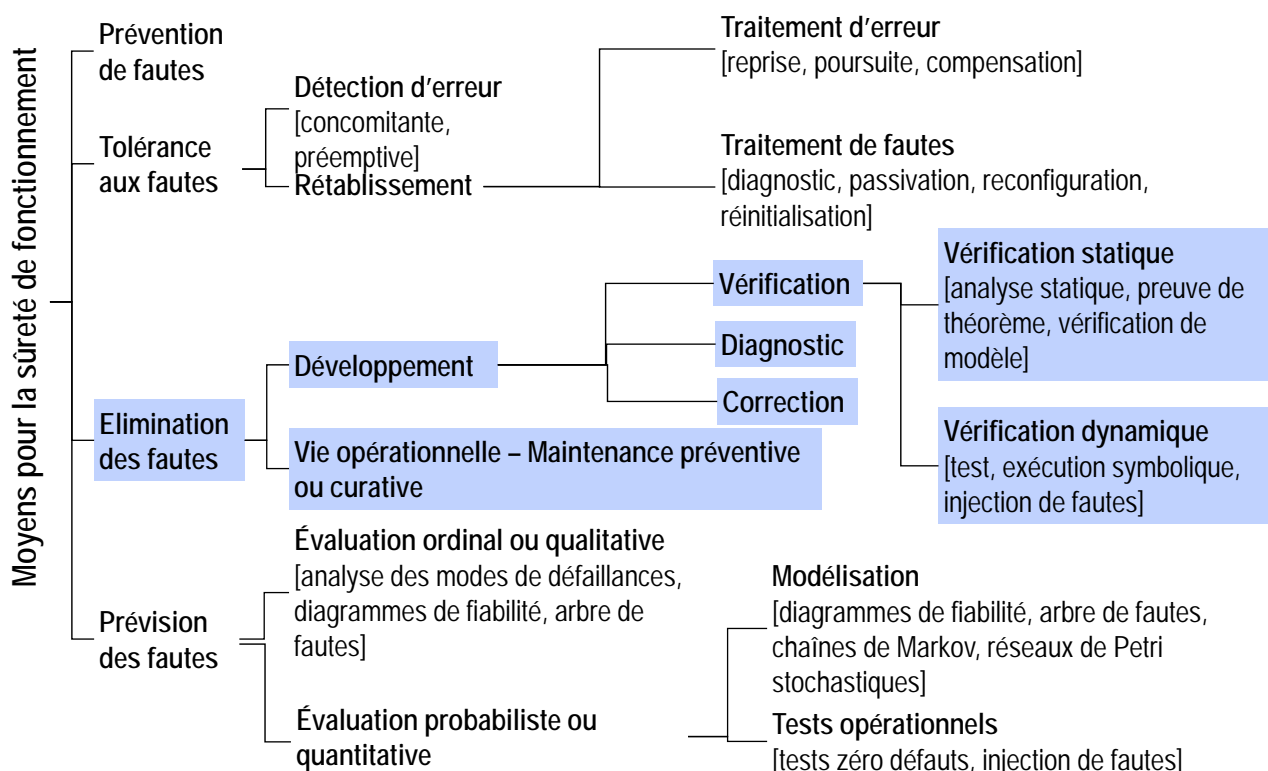
⇒ Comment définir l'Oracle ?

- Observer les sorties et décider si elles satisfont les conditions de vérification → Calcul manuel des sorties attendues, spécifications exécutables, tests dos à dos de versions différentes, contrôle de plausibilité des sorties, ...

⇒ Comment décider de l'issu d'un test ? Critère d'arrêt du test et Notion de Qualité du Test.

- Le but du Test : trouver des fautes (« Erreur observée n'est pas faute trouvée »), mais surtout augmenter notre confiance dans la qualité du produit
- Si (plus) aucune faute trouvée, que conclure ???? programme très bon ou test très mauvais, ou quelque chose entre les deux ?
- Notion de Qualité d'un jeu de tests.
 - Arrêt du Test quand la Qualité souhaitée atteinte
 - Comment mesurer la Qualité ?
 - Par rapport à critère de représentativité: par exemple un degré de couverture.
 - Encore mieux : en évaluant la sensibilité du test, soit par Injection de Fautes "volontaires" dans le système, soit pour le logiciel, par mutation de programme pour générer des **mutants** (un bon jeu de test distingue les mutants de l'original)

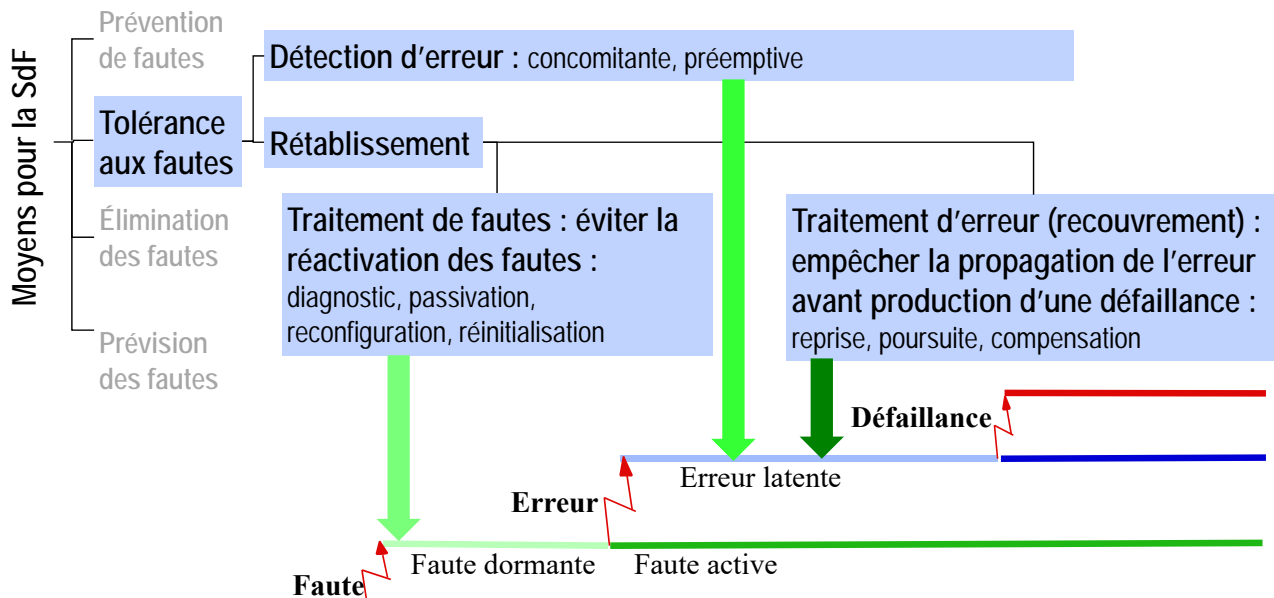
d) Résumé dans l'arbre des MOYENS





1. Objectifs et tâches de la Tolérance aux fautes :

- ⇒ But : fournir par **redondance** un service conforme aux buts de la fonction, en dépit de fautes
- ⇒ Tâches essentielles



2. Notions de **modèle** de fautes et **hypothèse** de fautes



- Quelles fautes cherche-t-on à traiter et quels sont leurs effets (erreurs) ?
- Exemples vis-à-vis de fautes physiques

- Arrêt sur défaillance

L'erreur conduit à l'arrêt du composant
→ basculement sur un composant identique correct

- Omissions

L'erreur conduit à la perte d'un message
→ protocole de répétition ou de diffusion fiable

- Erreurs en valeur

L'erreur conduit à l'altération d'une valeur en sortie
→ mécanisme de compensation d'erreur

- Exemples vis-à-vis de fautes du logiciel

- Erreur de conception

Faute de conception par rapport aux spécifications
→ diversification

- Erreur de codage

Faute de programmation
→ diversification

- Erreur de ressources

Défaut de ressource
→ ré-exécution sur autre support d'exécution



■ 3. Traitement de faute : éviter qu'une faute ne produise à nouveau des erreurs

⇒ Identifier la faute = **diagnostic** :

- Activer (de façon contrôlée) les fautes plausibles, et comparer les erreurs obtenues avec celles attendues pour les fautes connues → **programme de diagnostic**
- Le diagnostic peut échouer, par ex. :
 - fautes « douces » : assimilées à une faute transitoire
 - programme de diagnostic insuffisant : à améliorer

⇒ Puis, en cas de faute interne permanente ou de faute dure :

- éliminer le sous-système défaillant : **passivation**
- mettre le système en mode dégradé : **reconfiguration**

⇒ Et ensuite :

- réparer ou corriger le sous-système : **réparation**
- remettre le système en mode nominal : **reconfiguration**

⇒ Et enfin propager la correction à des systèmes analogues

Il faut parfois faire le traitement de faute avant de recouvrer les erreurs

■ 4. La Détection d'Erreur



⇒ **Détection : Basée sur la REDONDANCE & La DIVERSIFICATION (dissemblance)**

- De l'information (fautes de transmission, de stockage)
- Du logiciel (fautes de conception)
- Du matériel (fautes physiques)

⇒ **Principe : "répliquer" l'entité dont on veut tolérer les défaillances**

- Modes de Défaillances non Corrélés (Diversité)
- Si une réplique défaille, l'autre continue de délivrer service correct
- Permet de repérer l'erreur, et éventuellement de la masquer

⇒ **Redondance de l'information : Reprendre toute ou partie de l'information**

- ex. de codage Binaire : du plus simple « Bit de Parité » ... à l'autre Extrême → Duplication

⇒ **Redondance Logiciel (fautes de conception)**

- Dans le même programme
- Avec des programmes différents : Diversification Fonctionnelle → Plusieurs équipes développent séparément la même fonctionnalité

⇒ **Redondance Matériel**

- Pour détecter / tolérer des Fautes Physiques
- Sans aucun effet sur les fautes de Conception (Ariane 5)

⇒ Exemples de détection par contrôle de vraisemblance

- Contrôle par matériel
 - adresse inexistante ou interdite
 - instruction ou commande inexistante ou interdite
 - dépassement de durée (watchdog)
 - violation de code détecteur d'erreur
- Contrôle par logiciel
 - contrôle sur les valeurs (absolues, relatives, fourchettes) et le format des résultats
 - contrôle sur l'ordre et les instants des événements
- Programmes de test : périodiques, sur demande, ...

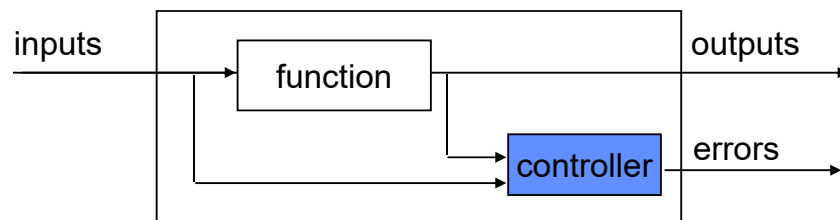
⇒ Exemple de détection par comparaison : comparer les résultats (dates, valeurs) de plusieurs exécutions choisies pour qu'une faute éventuelle produise des erreurs différentes

- faute physique interne : exemplaires **identiques** + comparaison bit-à-bit
- faute physique externe : exemplaires **similaires** + comparaison bit-à-bit
- faute de conception (matériel ou logiciel) ou d'interaction : exemplaires **diversifiés** + comparaison "intelligente"

⇒ Notion de composant autotestable



- L'ajout de capacités de détection d'erreur à un composant conduit à la notion de **composant autotestable** (*self-checking component*)
- Cette notion peut s'appliquer à des composants logiciels et matériels



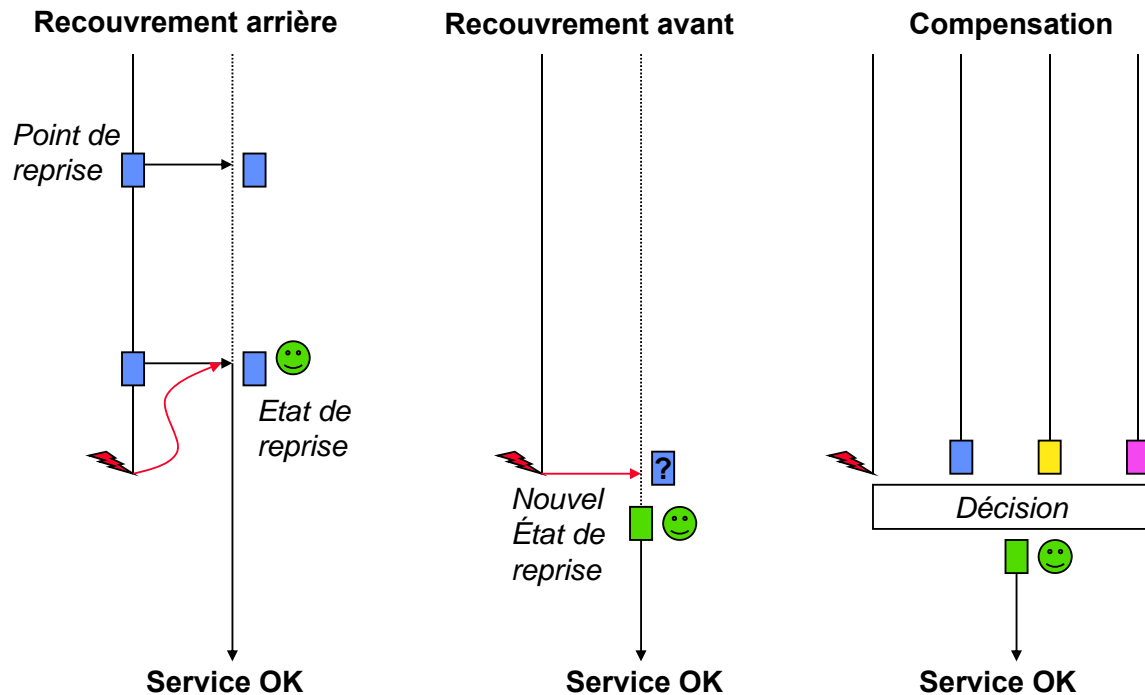
• Commentaires

- Avantage: limitation de la propagation d'erreur → **zone de confinement** (*error confinement area*)
- Composant autotestable + couverture 100% → **composant à silence sur défaillance** (*fail silent component*).
- Approche récursive: contrôleur = composant autotestable

■ 5. Le Recouvrement d'Erreur (Error-Recovery)



⇒ 3 Grandes Approches de recouvrement : par reprise, par poursuite, par compensation



⇒ Recouvrement par reprise :

- Sauvegarde régulière de l'état du système → "points de reprise"
- Quand Détection d'Erreur : Retour en arrière au dernier état sauvé
- Reprise à partir de l'état restauré

⇒ Recouvrement par poursuite :

- But : après détection, rechercher nouvel état acceptable.
- Très dépendant de l'application
- A l'extrême : tout arrêter de manière contrôlée ("gracefully") : possible si l'état "arrêté" est sûr
- Nouvel état souvent en modes dégradés (modes de survie)
- Exemple : Réinitialisation / Relecture de tous les capteurs.

⇒ Recouvrement par Compensation / Masquage

- Uniquement possible avec suffisamment redondance
- Défaillance d'un composant compensée par 1 ou plusieurs répliques : soit après détection de l'erreur, soit automatiquement (vote)

⇒ Détails sur masquage par vote majoritaire

- $2N+1$ répliques permette de masquer N défaillances.
- Suppose les traitements déterministes
- Si les répliques sont à silence sur défaillance : problème grandement simplifié (avec M répliques, on peut tolérer $M-1$ défaillances)



Sûreté de Fonctionnement (SdF)

- 3^{ème} partie : Exemples et compléments -

8 – Exemple de tolérance au fautes (TaF) pour réseau de communication des CDVE

9 – Exemples d'architectures et de stratégies de TaF

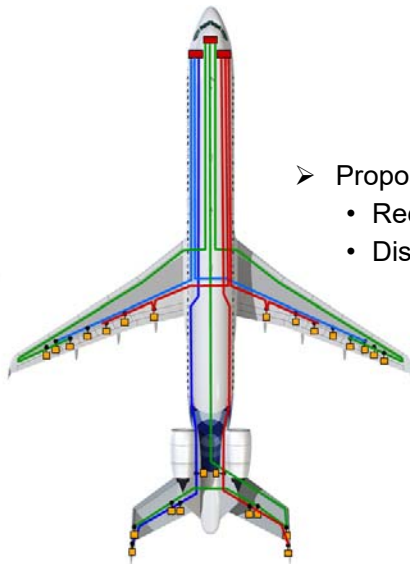
10 – MESURES de base : exemples

11 – INJECTIONS de FAUTES : moyen expérimental de validation de Taf

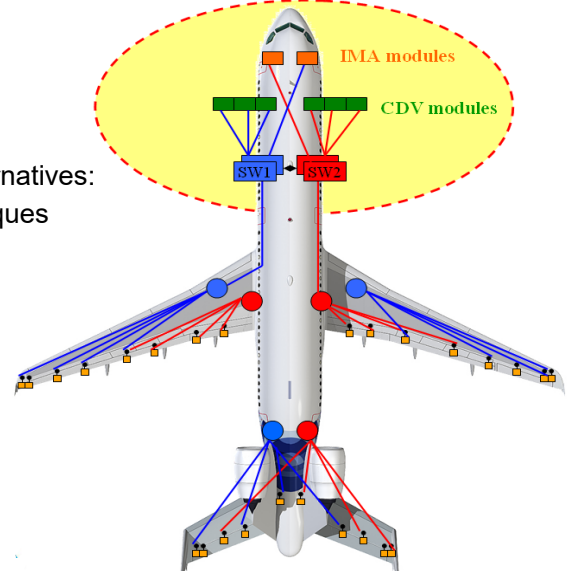
8 – Exemple de tolérance au fautes (TaF) pour réseau de communication des CDVE

Systèmes de Commandes De Vol Electrique (CDVE) : contrôler la trajectoire avion en appliquant les consignes du pilote aux surfaces de contrôle

CDVE = système de commande-contrôle, embarqué, réparti, temps réel, et critique → démontrer la sûreté de fonctionnement est régie par de sévères normes de certification.



- Proposer des architectures alternatives:
- Recours aux médias numériques
 - Distribution de l'intelligence



Sûreté de fonctionnement

Chapitre 8 : Exemple de TaF pour réseaux de CDVE

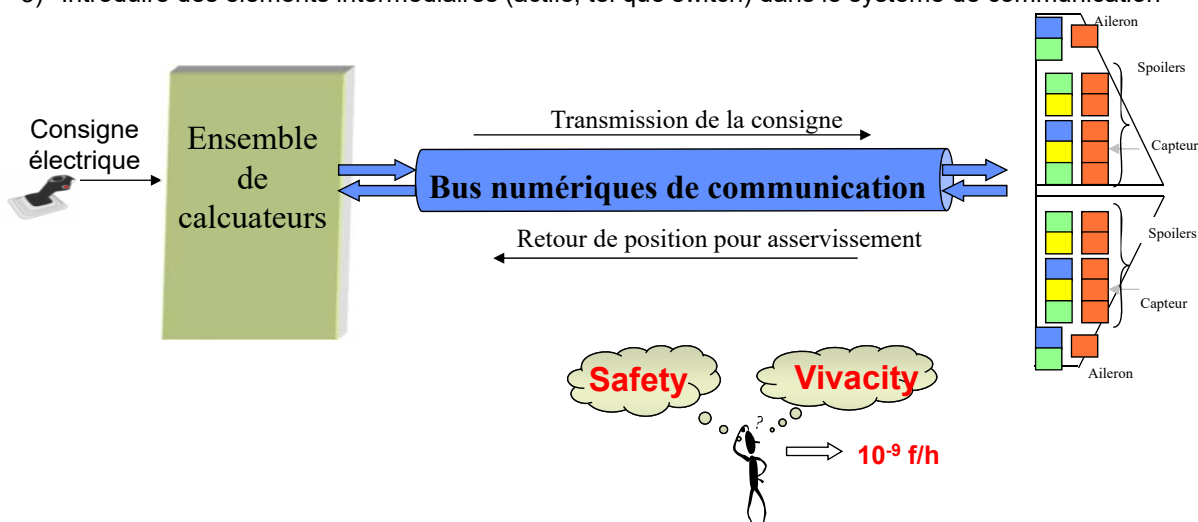
77

1. Problématique et expression des BESOINS

- 1) Spécifier une architecture de communication avec des bus numériques
- 2) Concevoir un protocole de communication qui assure :
 - L'intégrité des valeurs des données échangées entre calculateurs, capteurs, actionneurs : *Safety*
 - L'intégrité temporelle des données échangées : *Vivacity*

Satisfaire les contraintes de criticité imposées par les réglementations avioniques → objectif d'intégrité de haut niveau = **Taux d'occurrence de l'événement indésirable < $10^{-9}/h$**

- 3) Introduire des éléments intermédiaires (actifs, tel que switch) dans le système de communication



Sûreté de fonctionnement

Chapitre 8 : Exemple de TaF pour réseaux de CDVE / 1 - Problématique

78

■ 2. Expression des Contraintes

- **Contraintes d'intégrité**
 - Eviter l'embarquement
 - Eviter le non rafraîchissement
- **Contraintes de débit et de taux de rafraîchissement des données**
 - Chaque servocommande doit être rafraîchie toutes les 10 ms
 - Débit nominal du bus de communication = 1 Mbits/s
 - Trame de données \sim 100 bits
- **Politiques adoptées par le protocole**
 - Réutilisation de la donnée précédente en cas de non-rafraîchissement (exp. Protocole ARINC)
 - Déclaration du bus de communication en panne en cas de 3 non rafraîchissements consécutifs

d'un nombre **X** de surfaces durant un temps **T**

Objectif d'intégrité de haut niveau (dans le cas de $X = 1$)

Taux de non détection de 3 trames erronées par une servocommande $< 10^{-9}$ /h

Technique classique de détection des erreurs de transmission : **CRC ???**

■ 3. Analyse des types de risques à couvrir (par le protocole)

■ Risques associés au bruit

- Bruit intrinsèque des composants -> en général, une inversion de bit
- Le taux d'occurrence associé est le BER

$2,6 \cdot 10^{-80}$ /h

■ Risques associés aux défaillances du câblage

- Perturbations électromagnétiques -> erreurs en rafale
- Rupture ou court-circuit d'un connecteur ou d'un câble -> risque plus important -> plusieurs trames successives peuvent être erronées

$2,4 \cdot 10^{-18}$ /h

■ Risques associés aux défaillances des éléments intermédiaires

- Eléments passifs : */ Amplification d'un signal avant retransmission
*/ Stockage temporairement les données reçues

Risques équivalent à un défaut de câblage

$3,4 \cdot 10^{-36}$ /h

- Eléments actifs : Traitement local des données

- Risques : **Non couvert par le CRC -> Toute défaillance peut se traduire par la transmission de données erronées non détectées -> Nécessité de compléments de codage**

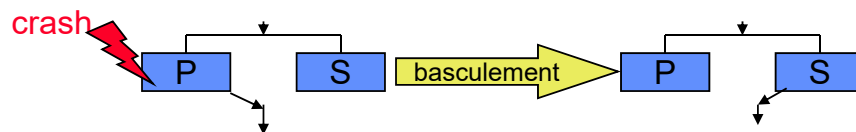
Compléments de codage efficaces

Vivacité du système de CDV

10^{-9} f/h

9 – TaF : exemples d'architectures et de stratégies

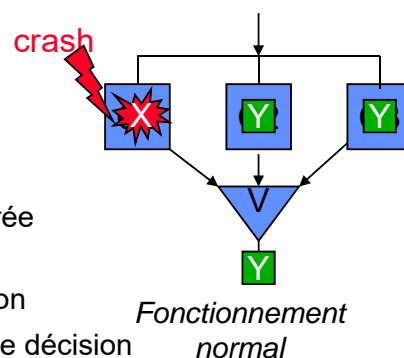
1. Architecture duplex



- Sorties du primaire P : considérées correctes tant qu'un **crash** n'a pas été détecté
- Si un **crash** est détecté, on considère que le composant P est à silence sur défaillance, c'est-à-dire sans propagation d'erreur. Ceci impose :
 - que toute erreur interne est confinée dans P (idem S)
 - un protocole de surveillance signalant à S le crash de P

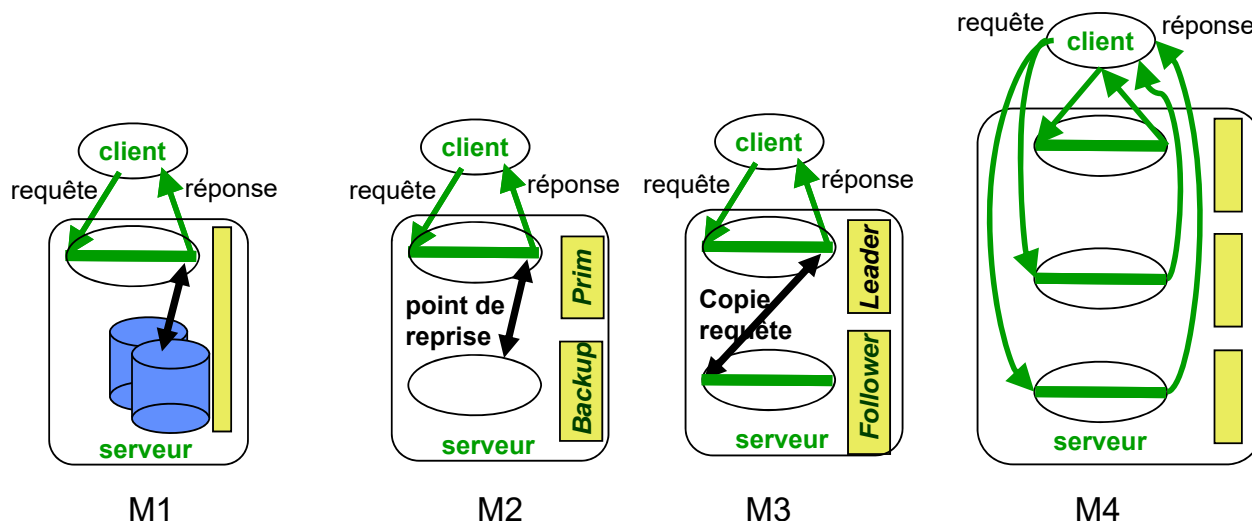
2. Architecture TMR (Triple Modular Redundancy)

- Les entrées sont identiques : valeurs et séquence d'entrée
- Le crash est considéré comme une absence de réponse
- Parmi les réponses fournies, la valeur majoritaire est considérée comme le résultat correct
- Si les composants sont identiques, le vote est une comparaison
- Si les composants sont diversifiés, le vote est un algorithme de décision



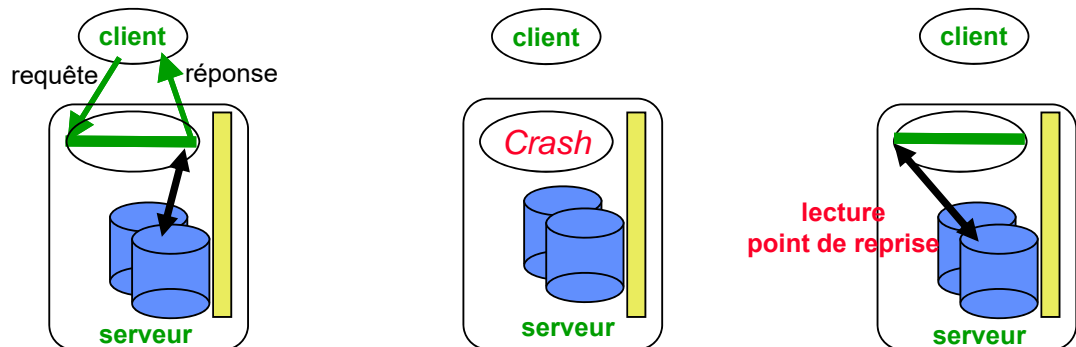
3. Stratégie/mécanismes de tolérance aux fautes PHYSIQUES, dans un principe de Client - Serveur

- M1: disques miroir
- M2: réplication passive
- M3: réplication semi-active
- M4: réplication active



3.1. M1 - Disques Miroir :

- Principe : les processus d'application traitent des requêtes des clients et sauvegardent l'état courant des calculs (points de reprise) sur des disques miroirs :
 - les points de reprise sont écrits sur les deux disques simultanément
 - lors de la remise en service, le processus relit son contexte sur disque

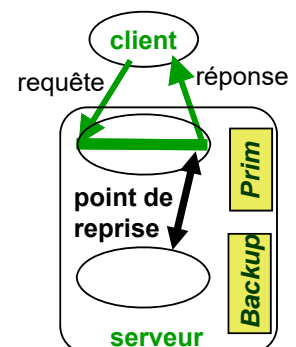


Remarques

- Si UNE seule unité de traitement est utilisée : la réparation doit être finie avant de redémarrer le service opérationnel → interruption de service pendant la durée de réparation
- Si PLUSIEURS unités ont accès au support de mémoire stable : redémarrage d'un service opérationnel avant terminaison de la réparation

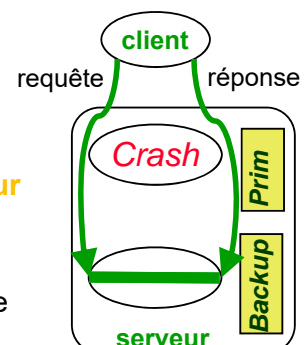
3.2. M2 - Réplication passive : Principes

- Principe :
 - Chaque processus d'application (primaire) possède un processus de secours (secondaire) sur un **site distinct de celui du primaire**
 - Le processus primaire effectue le traitement des requêtes des clients et **transmet périodiquement une image de son état au secondaire**
 - Le secondaire met à jour son état (contenu de ses variables) sur réception des points de reprise
 - Seul le primaire retourne une réponse au client



Commentaires

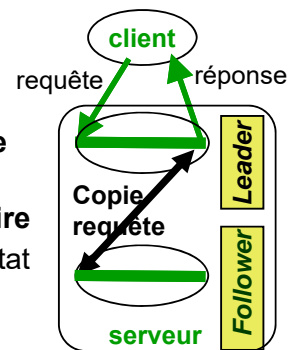
- Hypothèse de base : toute réponse à une requête est correcte.
- Notion de silence sur défaillance → **composant autotestable + couverture de détection de 100%**
- Un point de reprise implique la capture d'un état complet pour la reprise après basculement (processus + exécutif)**
- La détection correspond à une surveillance mutuelle
- Durée d'interruption de service : due à la restauration d'un service actif



3.3. M3 - Réplication semi-active

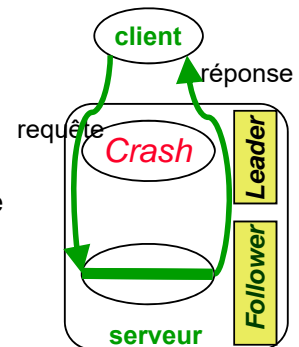
• Principe :

- Chaque processus d'application (primaire) possède un processus de secours (secondaire) sur **un site distinct de celui du primaire**
- Le processus primaire effectue le traitement des requêtes de ses clients et **transmet la requête qu'il vient de traiter, au secondaire**
- Le secondaire traite à son tour la requête pour mettre à jour son état
- Seul le primaire retourne une réponse au client



• Remarques :

- Hypothèse de base : toute réponse à une requête est correcte.
- Notion de silence sur défaillance → composant autotestable + couverture de détection de 100%
- Il n'y a pas de point de reprise : l'état est reconstruit par le traitement de la requête
- La détection correspond à une surveillance mutuelle
- Durée d'interruption de service : due au seul changement de mode



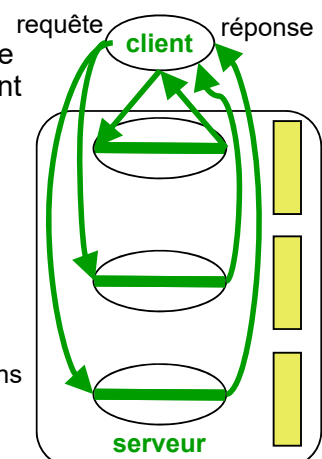
3.4. M4 - Réplication active et vote majoritaire : Principes

• Principe :

- Chaque processus d'application est répliqué en 3 exemplaires **sur des sites distincts**.
- Les 3 répliques reçoivent des requêtes au travers d'un système de communication de groupe et effectuent le traitement correspondant
- Chaque réplique transmet une réponse au client qui effectue un vote majoritaire :
 - la défaillance d'un processus est détectée par l'absence de réponse
 - une erreur de valeur sur l'une des réponses est détectée par vote majoritaire

• Remarques :

- Diffusion des requêtes d'entrée :
 - Toutes les requêtes, quelle que soit leur source, doivent être reçues dans le même ordre
 - Une copie de chaque requête est reçue par toutes les répliques, ou aucune n'en reçoit
- Les copies doivent être déterministes : les mêmes entrées produisent les mêmes sorties
- La décision:
 - Répliques identiques : vote bit-à-bit
 - Répliques similaires : algorithme de décision



10 – MESURES de base : exemples

■ 1. Quelques attributs, symboles et estimateurs :

- Fiabilité (Reliability) : mesure de la délivrance continue d'un service correct, donc du temps jusqu'à défaillance
- Disponibilité (Availability) : mesure de la délivrance d'un service correct par rapport à l'alternance « service correct – service incorrect », donc de la succession des défaillances et des restaurations
- Des exemples d'acronymes / symboles
 - **MTFF : Mean Time to First Failure, MTTF : Mean Time To Failure**
 - **MTBF : Mean Time Between Failures**
 - **MTTR : Mean Time to Repair**
- Des estimateurs statistiques (moyennes arithmétiques)
 - $N(0)$: nb de systèmes de l'échantillon
 - $N(t)$: nb de systèmes non défaillants à l'instant t
 - $N_1(t)$: nb de systèmes n'ayant pas défailli depuis l'instant initial

■ 2. Mesures / estimateurs statistiques

- Fiabilité (Reliability):

$R(t)$ = probabilité {S non défaillant sur $[0, t]$ }

Fonction non croissante sur $[0, \infty[$

Estimateur statistique de $R(t) = N_1(t) / N(0)$

Nb de systèmes n'ayant pas défailli depuis t_0

Nb total de systèmes

- Calcul des MTTF et MTTR :

$$MTTF = \int_0^{\infty} R(t) dt \quad MTTR = \int_0^{\infty} M(t) dt$$

■ 4. Notion de COUVERTURE : efficacité de la tolérance et fautes

Notion de couverture $C = \text{Prob}\{\text{traitement correct} \mid \text{erreur}\}$

- Sachant qu'il y a eu une erreur, quelle est la probabilité d'obtenir un traitement correct ?
- Supposons qu'il y ait 10 erreurs dans un système parfaitement observable, ou bien que l'on ait injecté 10 fautes qui ont été activées et ont conduit à 10 erreurs :
 - si on détecte 10/10 alors $C=1$ (100% de résultats corrects)
 - si on détecte 9/10 alors $C=0.9$ (90% de résultats corrects)

11 – Injections de fautes : moyen expérimental de validation de Taf

Les mécanismes de tolérance aux fautes sont à la fois

- Faillibles (fautes de conception et/ou de réalisation)
- Conçus pour traiter des entrées spécifiques : les fautes !
→ leur vérification est essentielle !

■ 1. Injection de fautes

- Simulation des fautes, ou plutôt de leurs effets, les erreurs
 - physiques : bit-flip
 - du logiciel : mutation d'instruction
- Cibles : un composant logiciel équipé de son mécanisme de TAF
 - Boite blanche : visibilité structurelle (*comment il est fait*) et comportementale (*comment il agit/réagit*)
 - Boite noire : aucune connaissance, seules les interfaces sont visibles
- Types d'injection :
 - niveau physique : *stuck-at*
 - modèle (ex. : VHDL)
 - par logiciel (SWIFI)
- Problèmes :
 - Représentativité des fautes
 - Observation et analyse des résultats

■ 3. Les techniques d'injection de fautes

Injection au niveau :

- système
- niveau RTL
- porte logique
- circuit

		SYSTÈME CIBLE	
		Modèle	Prototype
M O Y E N	Logique & Information	<i>injection en simulation</i>	<i>injection en logiciel</i>
	Physique		<i>injection physique</i>

Injection au niveau :

- Nœuds
- tâche
- instructions
- mémoires
- Registres

Injection au par :

- ions lourds : non « répétable »
- coupure des métallisations par laser : non « répétable » et **destructif**
- perturbations EM : non « répétable »
- altération des niveaux logiques (au niveau broches des CI)
- altération des niveaux d'alimentation : non « répétable » et **destructif**