Sûreté de Fonctionnement des systèmes informatiques

Walter SCHÖN

Sûreté de Fonctionnement des systèmes informatiques

Méthodes de conception de systèmes informatiques sûrs de fonctionnement (Tolérance aux fautes)

Conception de systèmes informatiques sûrs de fonctionnement

La conception d'un système informatique sûr de fonctionnement doit couvrir deux aspects :

- Absence de fautes de conception ou de fabrication susceptibles de nuire à la sûreté de fonctionnement : objet de l'évitement (prévention et élimination) des fautes.
- Conserver un comportement sûr de fonctionnement si une faute (défaillance matérielle, perturbation, voire faute de conception restée dormante) venait malgré tout à se produire : objet de la tolérance aux fautes.

Prévention et élimination des fautes

- Absence de fautes de conception : prévention des fautes => rigueur du processus de développement, méthodes formelles de développement
- Absence de fautes de conception : élimination des fautes =>
  - Vérification
    - Analyse statiques (revues…)
    - Preuves mathématiques
    - Analyse de comportement (model checking)
    - Exécution symbolique
    - Tests (structurels, fonctionnels…)
  - Corrections, puis vérifications de non régression

⇒ Sont les deux autres sujets de l'UV

Tolérance aux fautes

- Comportement sûr en présence de fautes résiduelles : tolérance aux fautes. L'usage distingue =>
  - Traitement d'erreurs : détection, diagnostic et recouvrement d'erreurs (avant qu'elles ne conduisent à une défaillance)
  - Traitement de fautes : détection, diagnostic des fautes causes des erreurs, passivation des fautes, reconfiguration éventuelle du système
- Sujet du présent cours, la tolérance aux fautes est essentiellement basée sur des redondances et des contrôles qui permettent de détecter (et parfois de corriger) les erreurs.

Tolérance aux fautes

Les techniques de base de la tolérance aux fautes reposent donc sur :

- Détection des fautes (autotests, diagnostic des erreurs détectées…) ou des erreurs consécutives à ces fautes (redondances…).
- Recouvrement des fautes ou des erreurs (corrections, mise en œuvre d'un comportement dégradé….)

Détaillées ci-après dans l'ordre inverse de leur mise en œuvre.

Techniques de recouvrement des fautes

- But : après détection (autotests) ou diagnostic de fautes éviter leur propagation en isolant au besoin les parties fautives. Les techniques utilisées sont :
  - L'isolation de faute : elle consiste à retirer les composants considérés comme fautifs du processus d'exécution ultérieur.
  - La reconfiguration : si le système ne peut plus délivrer le même service, elle consiste à modifier la structure du système de telle sorte que les composants non-défaillants puissent délivrer un service dégradé acceptable.

Toutes deux supposent naturellement l'existence d'un certain niveau de redondance dans le système permettant la poursuite de ce service éventuellement dégradé.

Techniques de recouvrement des erreurs

- But : transformer l'état erroné en un état permettant le maintien d'un service éventuellement dégradé. Les techniques utilisées sont :
  - Le recouvrement d'erreurs par reprise (l'état du système est sauvegardé périodiquement et permet un retour en arrière en cas d'erreur),
  - Le recouvrement d'erreurs par poursuite (en cas d'erreur on recherche un nouvel état acceptable mais en général dégradé), cas en particulier des systèmes à défaillances contrôlées.
  - La compensation d'erreurs (le niveau de redondance est suffisant pour permettre de transformer l'état erroné en état sans erreur) : attention dans ce cas à la compensation automatique et systématique (masquage d'erreurs) qui nuit à l'observabilité du système => Signalement nécessaire

Recouvrement par reprise (backward revovery)

Parfois appelée « reprise arrière » : Consiste à restaurer le système à un état précédent (et général à exécuter une section alternative du programme de fonctionnalité équivalente mais utilisant d'autres algorithmes) :

- Point de recouvrement : point auquel un processus est restauré
- Checkpointing : établissement d'un point de recouvrement (en sauvegardant l'état approprié du système)
- Avantage : l'état erroné est supprimé. Ne repose pas sur la recherche de l'endroit de la faute ou de sa cause
- Le recouvrement d'erreurs en arrière peut être utilisé pour recouvrir de fautes imprévues, incluant (théoriquement) les erreurs de conception (sous réserve de mode commun de faute avec les sections alternatives).

Recouvrement par reprise (backward revovery)

- Technique relativement lourde : écriture d'un point de reprise (sauvegarde de l'état du système jumelé à des historiques des messages envoyés ou reçus) généralement coûteux.
- Une reprise implique de lire le point de reprise le plus récent et l'historique des messages envoyés (ou reçus) depuis le dernier point de reprise.
- Pour effectuer une reprise, l'information nécessaire à cette reprise doit avoir été écrite sur un média sûr :
  - Exclut la mémoire vive : effacée si le matériel plante et même le simple disque dur : peut être affecté en cas de plantage pendant les phases d'écriture.
  - Utilisation fréquente de redondances disques dur : RAID (Redundant Array of Inexpensive Disks).

Recouvrement par reprise (backward revovery)

- Un système utilisant le recouvrement d'erreur par reprise classe la mémoire en trois catégories :
  - La mémoire volatile: le contenu est perdu si il y a une défaillance (arrêt) du processeur. En cas de redémarrage le système est alors amnésique.
  - La mémoire non-volatile: le contenu peut survivre à une défaillance du processeur. La défaillance peut provoquer une altération des données. En cas de redémarrage, le système doit disposer d'un mécanisme de vérification de la correction des données.
  - La mémoire stable: le contenu survit toujours à une défaillance du processeur. En cas de redémarrage, le processeur peut utiliser sans crainte les données pour sa reconfiguration.

Recouvrement par poursuite (forward recovery)

- Parfois appelée « reprise avant » : Consiste après détection de l'état erroné à chercher un état acceptable (vis-à-vis des objectifs en général sécurité / disponibilité) et à poursuivre un service (en général dégradé) à partir de cet état.
- Cas extrême : utilisé dans de nombreux domaines d'applications critiques où il existe un état de sécurité identifié et accessible (relativement) rapidement (transport terrestre, production d'énergie…) : système conçu pour aller vers l'état de sécurité en cas d'erreur : Système à défaillance contrôlées.

Systèmes à défaillances contrôlées

- Les applications type transport terrestre ou énergie utilisent souvent une conception dite en sécurité intrinsèque (fail safe).
- Selon ce principe « toute défaillance doit conduire le système dans un état de sécurité au moins égale à celui où il était avant la défaillance »
- Le respect de ce critère est, pour les systèmes analogiques, souvent réalisé par conception en sécurité positive :
  - ✓ Un état sécuritaire, « restrictif » (souvent l'arrêt de l'installation) est identifié et conçu pour correspondre à l'état de plus basse énergie
  - ✓ Atteindre ou maintenir tout autre état (plus permissif) nécessite de fournir de l'énergie
  - ✓ Aucune défaillance ou combinaison vraisemblable ne conduit à fournir intempestivement de l'énergie

Systèmes à défaillances contrôlées

- Pour les systèmes informatiques une conception semblable amène aux systèmes à arrêt sur défaillance (fail stop),
- L'informatique doit évidemment être intégrée dans un système global pour lequel l'arrêt est sécuritaire : voir l'impact des E/S, messages réseau…
- Un calculateur sécuritaire fail stop recouvre ses erreurs en s'arrêtant (comportement souvent appelé mise en repli du calculateur qui peut être vu comme cas extrême de recouvrement par poursuite).
- Pour améliorer la disponibilité (et non la sécurité qui pourrait se satisfaire d'un seul calculateur), on utilise parfois une redondance de plusieurs calculateurs fail stop

Principales techniques de détection des fautes

- Les systèmes basés sur le traitement de fautes doivent remonter jusqu'au composant (matériel ou logiciel) fautif (afin en général de l'isoler : passivation des fautes). Les deux techniques utilisées sont :
  - Le diagnostic d'erreur : après détection des erreurs, par recoupement des « symptômes »
  - Les autotests : intégrés au cycle applicatif (écritures / relectures en mémoire, déroulement d'opérations à résultat connu et stockés en PROM…) . Problèmes : alourdissement du logiciel embarqué (coût sur le temps de cycle…) et question (jamais close…) du taux de couverture…

Principales techniques de détection des erreurs

Dans les techniques usuelles de détection d'erreurs, on distinguera :

- Les redondances d'exécution (duales temporelles),
- Les redondances matérielles (multiplex, architectures p/n),
- Les redondances Informationnelles (codes détecteurs et correcteurs d'erreurs, codes arithmétiques),
- Les contrôles temporels (« watchdogs »),
- Les contrôles d'exécution (contrôles de vraisemblance, contrôles basés sur codes arithmétiques…).

- Lorsqu'une erreur est détectée, elle est recouverte en utilisant l'une des techniques évoquée plus haut. En complément la partie matérielle (E/S…) peut utiliser la conception à défaillances contrôlées : les défaillances matérielles conduisent à un état où la sécurité est maîtrisée.
- Ces techniques, au cœur de la tolérance aux fautes sont largement développées ci-après.

Redondances d'exécutions (duales temporelles)

- Principe : une même application est exécutée deux fois par le même processeur en utilisant des ressources (PROM, RAM) différentes.
- Les résultats sont comparés en général par un dispositif externe au processeur, toute discordance provoquant la mise en repli du calculateur (comportement fail stop).

Redondances d'exécutions (duales temporelles)

Problèmes : certaines défaillances des dispositifs matériels «partagés» (voteur et processeur) ne sont pas détectées et restent donc latentes :

- Vote toujours positif quelle que soit l'entrée
- Pannes subtiles de la partie ALU du processeur (exemple A-B au lieu de A+B systématique)

Redondances d'exécutions (duales temporelles)

Traitement : Le plus souvent par autotests intégrés dans chaque cycle applicatif :

- Vote négatif sur données volontairement discordantes (sans aller jusqu'au repli…)
- Test fonctionnel très complet du processeur

➡ Coût en performance, et problème (toujours ouvert) de la couverture des tests mis en place.

Redondances d'exécutions (duales temporelles)

Traitement : Parfois complété par :

- Dissymétrie volontaire de codage des deux applicatifs : exemple : A+B vs -(-A-B)
  (mais vérifier que le compilateur n'effectue pas d'optimisation intempestive !)
- Détermination expérimentale du taux de fautes non détectées par injection volontaire de fautes sur banc de test adéquat

Redondances matérielles (multiplex)

- Principe : plusieurs calculateurs déroulent la même application et comparent leurs résultats (unanimité ou vote majoritaire)
- La redondance est donc bien là cette fois pour la sécurité et/ou la disponibilité
- Un calculateur mis plusieurs fois en minorité peut être isolé matériellement par les autres (dispositif genre «fusible»)

➡ Très bon compromis (cas des votes majoritaires) sécurité / disponibilité

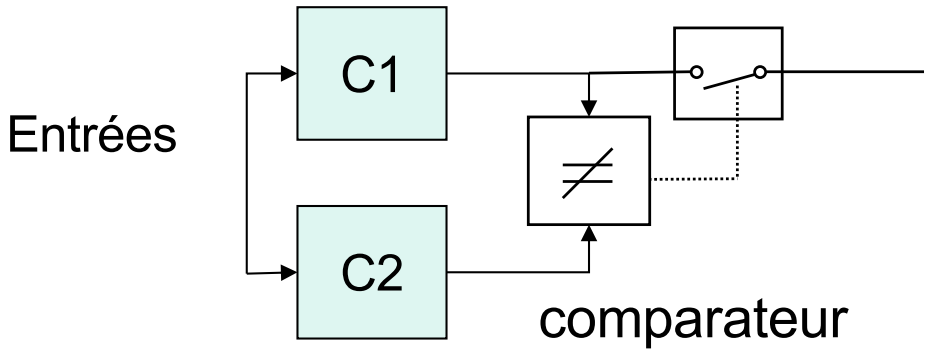
Redondances matérielles (multiplex)

Problèmes :

- Multiplicité du matériel => coûts de matériel et de maintenance accrus
- « Arbitrage » entre les résultats souvent réalisé par un voteur physique : mode commun
- Traitements asynchrones (conception la plus fréquente) => nécessite des applications robustes aux retards (pour éviter les votes en échec par effets de bords temporels
- Synchronisation par horloge commune (moins fréquent) => l'horloge est un mode commun
- Si matériels de même technologie : modes communs liés à des fautes de conception ? (Ex. les bugs des premiers Pentium…).

Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

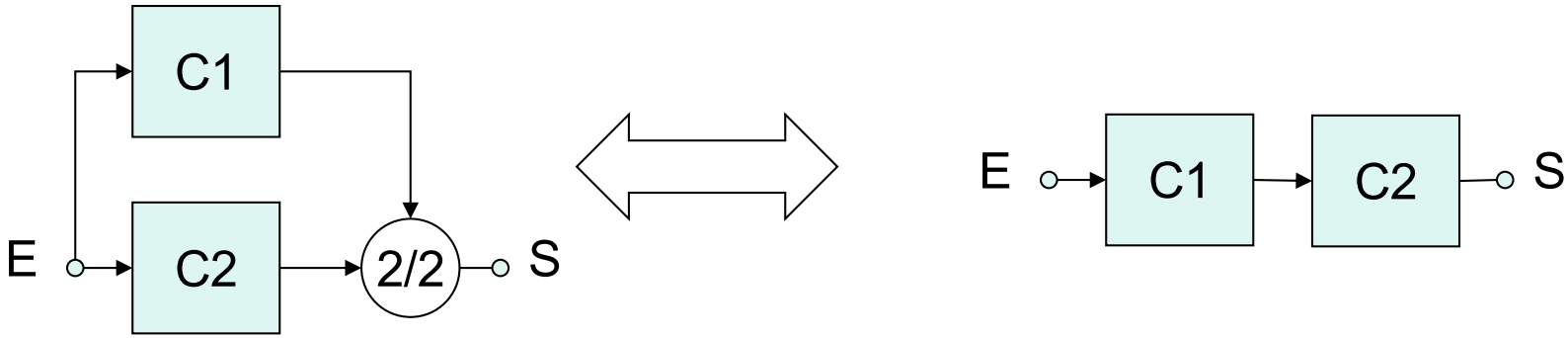
- Deux équipements identiques (redondance homogène) ou non (redondance hétérogène) qui ne sont pas fail stop (safe) individuellement =>Sécurité basée sur la redondance.
- Redondance sécurité : toute discordance provoque la mise en repli de l'ensemble (sorties restrictives : suppose un domaine où la notion existe) : système à défaillances contrôlées (Une sortie à 0V doit correspondre à l'état de sécurité).
- Généralisable N/N dans un souci de sécurité maximal



Sortie de C1 (par exemple) validée si non discordance, 0 sinon

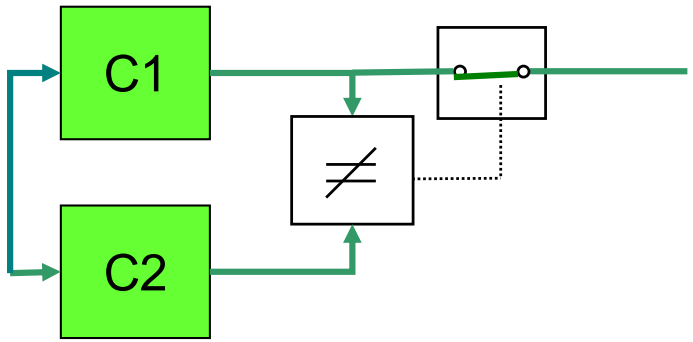
Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- Vu d'un point de vue fiabilité, il s'agit donc d'un système série (Voteur 2/2 ou 2oo2 : « 2 Out Of 2 » acronyme d'usage pour ce type d'architecture)



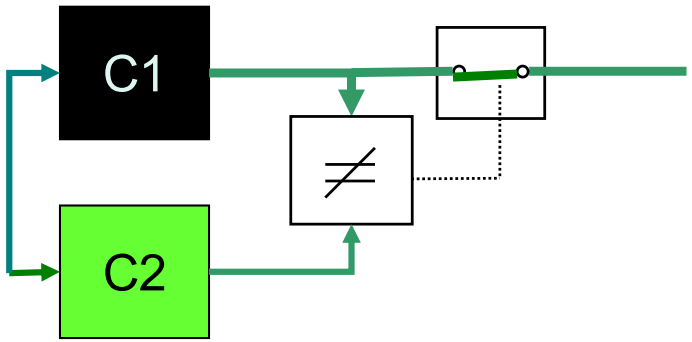
Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- En effet la défaillance de l'un quelconque des calculateurs aboutit en général à une discordance et donc à une mise en repli (sorties à 0 qui correspond à l'état de sécurité mais qui interrompt le service).
- La première défaillance peut toutefois rester masquée un certain temps selon son instant d'occurrence et le fonctionnel nominal du système : Les deux calculateurs fonctionnent initialement correctement durant une phase ou la sortie est nominalement permissive (« feu vert ») :



Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

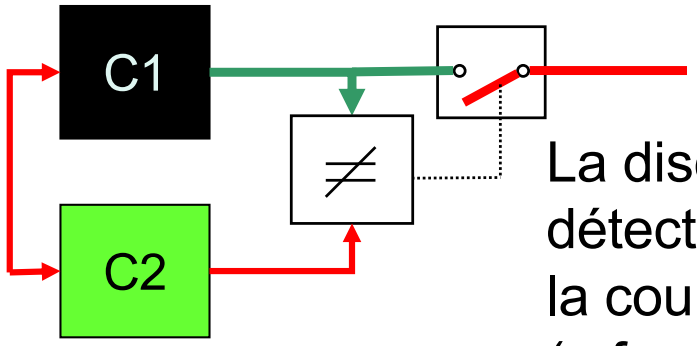
- Durant cette phase où la sortie est nominalement permissive, l'un des deux calculateurs connaît une défaillance qui a pour effet de bloquer sa sortie dans l'état permissif (sortie toujours égale à « feu vert ») : défaillance contraire à la sécurité (individuellement chaque calculateur n'est pas fail stop (safe).
- La défaillance est indétectable durant toute la phase où la sortie est nominalement permissive (« panne latente »).



Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- Le fonctionnel système nominal impliquant généralement un passage régulier par l'état restrictif (si tel n'était pas le cas il faudrait le prévoir…), la défaillance est rapidement détectée :
- Du point de vue de la fiabilité et en considérant donc (hypothèse pessimiste pour la fiabilité) que la première défaillance est immédiatement détectée, le système est donc série : $\lambda_{\text{Fiabilité}} \cong \lambda_1 + \lambda_2$

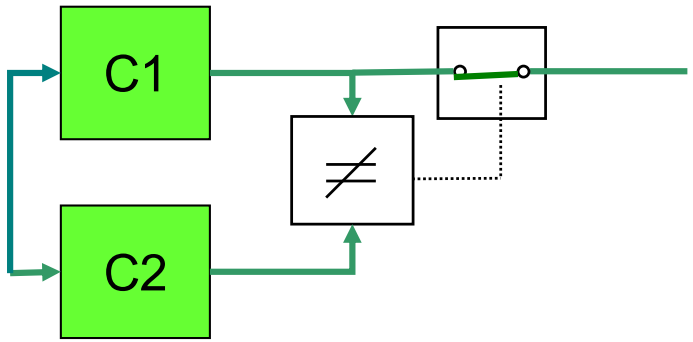
Le calculateur défaillant reste bloqué sur permissif mais l'autre passe restrictif



La discordance est détectée et provoque la coupure de la sortie (« feu rouge »)

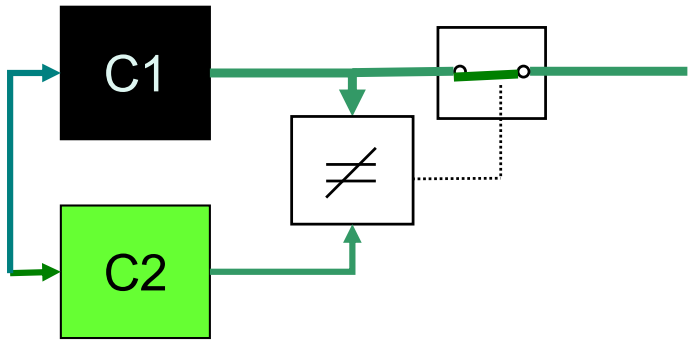
Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- Cette même architecture vue du point de vue sécurité peut être vue comme un système parallèle (deux points de vue différents pour un même système peuvent conduire à des diagrammes différents).
- Scénario redouté : Les deux calculateurs fonctionnent correctement durant une phase ou la sortie est nominalement permissive (« feu vert »)



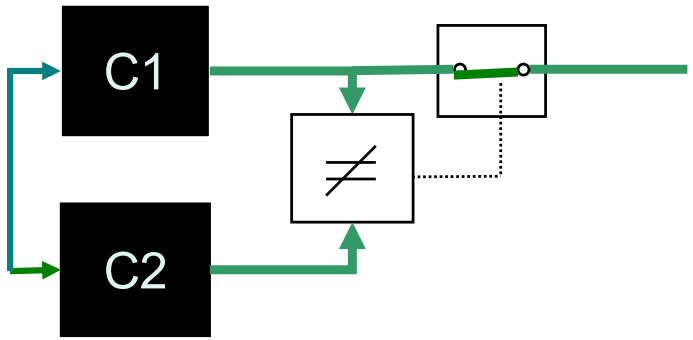
Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- Durant cette phase où la sortie est nominalement permissive, l'un des deux calculateurs connaît une défaillance qui a pour effet de bloquer sa sortie dans l'état permissif (sortie toujours égale à « feu vert »).
- La défaillance est indétectable durant toute la phase où la sortie est nominalement permissive (« panne latente »).



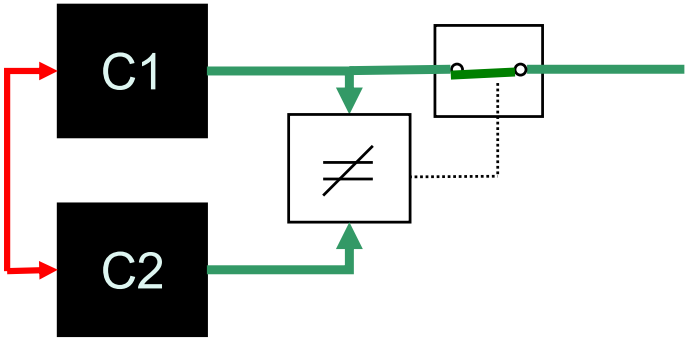
Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

- Le second calculateur connaît le même mode de défaillance durant cette durée de latence (on est bien dans un scénario à double défaillance, le système est bien, vu sous cet angle sécurité, parallèle)…
- La sortie est donc après comparateur maintenant bloquée à permissif (vert), l'accident est proche…



Redondances matérielles : Architectures les plus fréquentes : Duplex 2/2

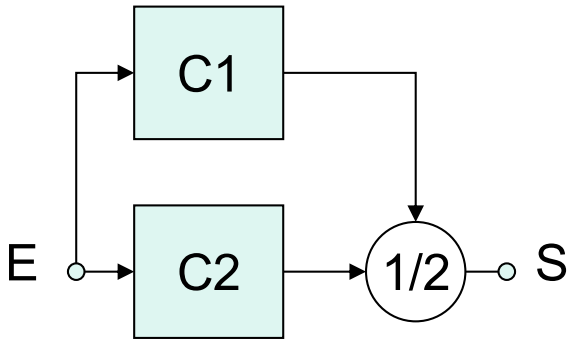
- Enfin le fonctionnel nominal demande un passage à l'état restrictif (feu rouge), qui n'est pas pris en compte suite à une non discordance due à la double défaillance….
- Le scénario nécessite l'occurrence des deux défaillances contraires à la sécurité (sortie collée à permissif) entre deux passages nominaux du fonctionnel système à l'état restrictif. Du point de vue sécurité, le système est comparable à un parallèle avec contrôle périodique de période T :



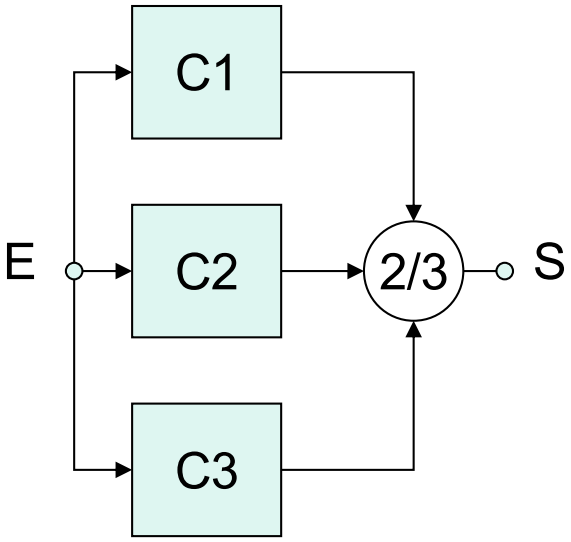
$$\lambda_{Secu} \cong \lambda_{s1}.\lambda_{s2}.T$$

Redondances matérielles : Architectures les plus fréquentes : Duplex 1/2

- Redondance disponibilité : en cas de désaccord soit on connaît l'équipement fautif (autotest) et on continue sur l'autre, soit on choisit l'un des résultats (en général le plus permissif: celui qui permet de poursuivre la mission) : souvent utilisé dans un souci de disponibilité avec des calculateurs individuellement fail stop.
- Il s'agit donc d'un système parallèle (Voteur 1/2 ou 1oo2)
- Généralisable 1/N dans un souci de disponibilité maximal



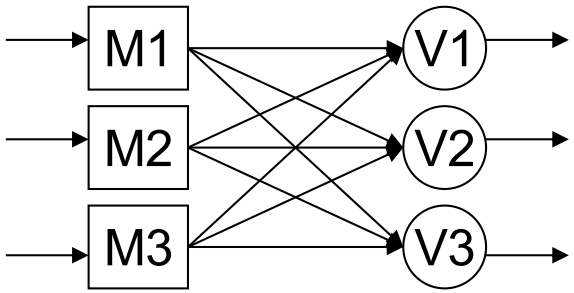
Redondances matérielles : Architectures les plus fréquentes : Triplex 2/3



- Trois équipements
- Voteur : valide résultat donné par au moins deux équipements
- Calculateur fautif peut être retiré définitivement (le système devient un 2/2)
- Il s'agit d'un 2/3 ou 2oo3 : très bon compromis disponibilité (tolère une faute) sécurité (nécessite le consensus d'au moins 2)

Redondance Triple Modulaire : TMR

- Il s'agit d'une variante du 2/3 où le voteur est lui même en trois exemplaires => Tolère une faute de voteur
- Utilisé souvent au niveau de modules élémentaires, la sortie de chaque module étant l'entrée du suivant :



Types de redondances

- Active ou chaude : tous équipements actifs, la sortie globale est élaborée par le voteur.
- Passive ou froide : un équipement (principal) seul actif. Le où les autres démarrent sur détection de faute du principal (suppose mécanisme de détection type autotest).
- Semi-active ou tiède : tous équipements actifs mais l'un est désigné comme principal et fournit les sorties d'ensemble. L'équipement principal change s'il se détecte fautif (autotest) ou s'il est mis en minorité par ses partenaires.

Types de redondance

- Homogène : Tous équipements de même technologie :
  - Avantage : Unicité des équipements, outils de développement etc.
  - Inconvénient : Mode commun lié à une faute de conception lors du développement ?
- Hétérogène : Équipements de technologies diversifiées :
  - Avantage : Faute de développement identique peu probable
  - Inconvénient : multiplication des outils, compilateurs etc.

Redondance informationnelle

- Technique majeure de la tolérance aux fautes la redondance informationnelle consiste à manipuler une information additionnelle (dite de contrôle ou de redondance) permettant de détecter voire corriger les erreurs. On évoquera ici :
  - Contrôles de parité
  - Codes à redondance cyclique
  - Codes de Hamming
  - Codes arithmétiques (exemple du processeur sécuritaire codé)
- Les codes envisagés seront essentiellement séparables (bits de contrôle distincts des bits d'information principale)
- Les codes non séparables (ou l'extraction est plus complexe) seront brièvement évoqués

Contrôle de parité

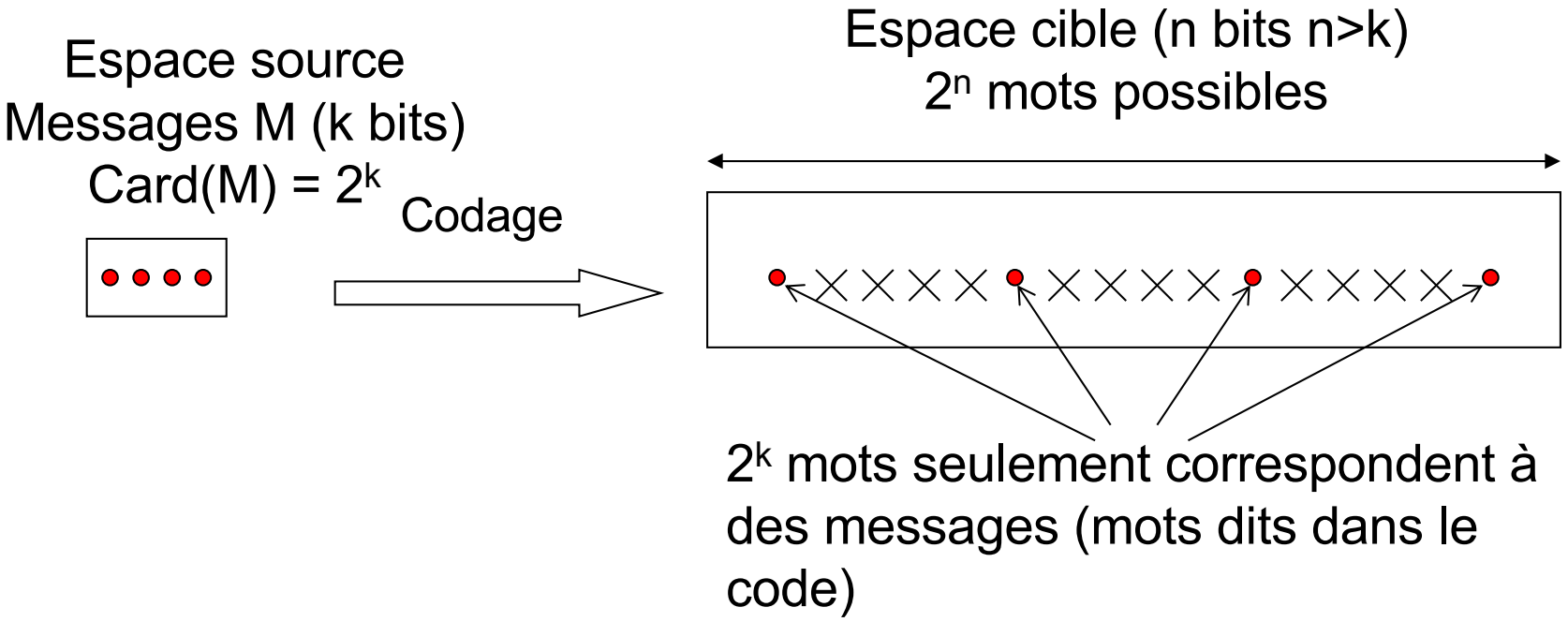
- La plus simple des techniques de détection des erreurs,
- Un bit de contrôle par mot
- Le bit additionnel est calculé de sorte que la somme des bits du mot (modulo 2) soit nulle (convention parité paire) ou égale à 1 (convention parité impaire).
- Permet de détecter une erreur (ou des erreurs en nombre impair).
- Ne détecte pas les erreurs doubles (ou en nombre pair)
- Très utilisé dans les transmissions par modem

Codes détecteurs et correcteurs d'erreurs

- Surtout utilisés pour les systèmes distribués (détection d'erreurs de transmission sur un réseau)
- Peuvent parfois être également pour détecter des erreurs internes à un calculateur (erreurs d'écriture en mémoire…)
- Principe : ajout d'information redondantes avec le message, permettant de détecter (et parfois de corriger) les erreurs
- On distingue les codes séparables (la partie redondance et la partie message sont des champs de bits distincts) et codes non séparables (séparation message / redondance est plus complexe)
- Certains codes particulièrement robustes (fonctions de hachage à sens unique) sont utilisés en cryptographie (détection de modifications malveillantes et non accidentelles).

Codes correcteurs d'erreurs : Codes de Hamming

- Principe : Injection de l'ensemble des messages (vocabulaire source sur k bits : en général constitué des 2^k mots possibles) dans un espace plus grand (n bits n>k), de sorte que les mots codés correspondant à des messages soient suffisamment différents.

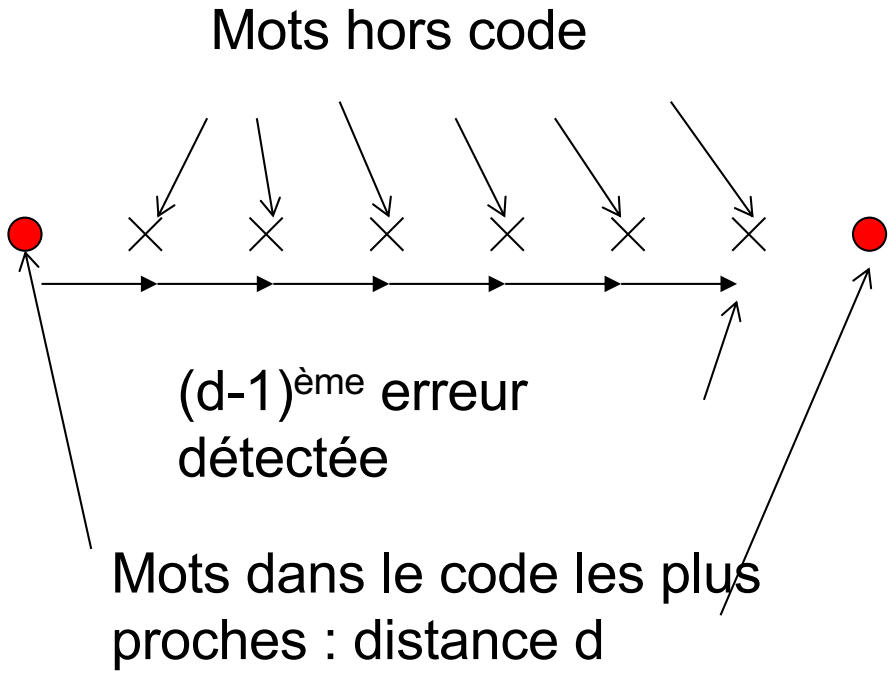


Distance de Hamming

- La différence entre deux messages codés (mots correspondant à des messages du vocabulaire source : mots dits dans le code) se mesure par la distance de Hamming qui est leur nombre de bits différents.
- La distance de Hamming d'un code pour un vocabulaire source donné est la valeur inférieure de la distance entre deux mots dans le code.
- Les mots ne correspondant pas à des messages codés sont dits hors code.
- Le code est caractérisé par le triplet [n,N,d], n : nombre de bits espace source, N : nombre de bits espace cible, d : distance de Hamming

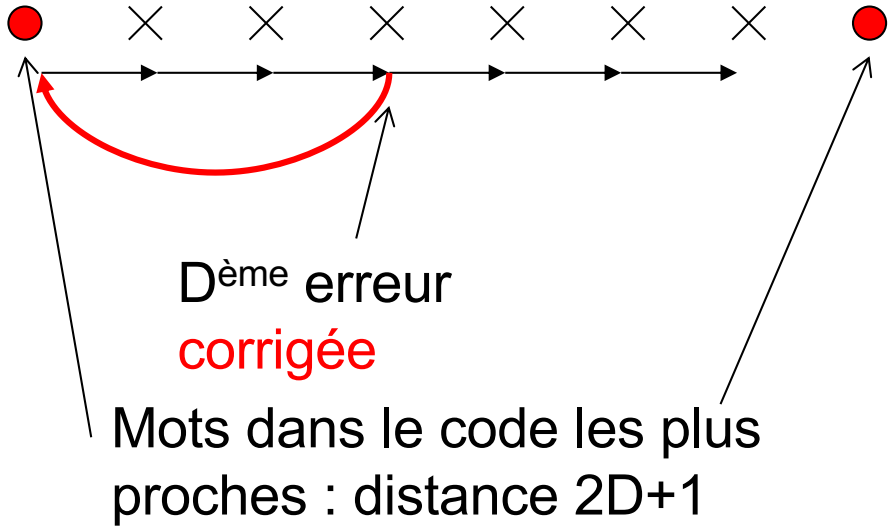
Détection d'erreurs

- Un code dont la distance de Hamming est d est capable de détecter au maximum d-1 erreurs



Correction d'erreurs

- Un code dont la distance de Hamming est 2D+1 est capable de corriger au maximum D erreurs, en ramenant le mot erroné (donc hors code) au mot dans le code le plus proche.



Code de Hamming : Exemple

- Code [7,4,3] :
  - 4 bits de message $M_4M_3M_2M_1$,
  - 3 bits de contrôle $C_3C_2C_1$,
  - Distance de Hamming : 3
- Principe : code conçu tel que :
  - M_4 intervient dans le calcul des trois C_j
  - Chaque autre M_i intervient dans le calcul de deux C_j sur les trois (paires différentes pour chaque M_i)

Code de Hamming : Exemple

- A la réception du message, on recalcule les C_j avec les M_i reçus:
  - Si une seule équation fausse => le bit faux est le bit C_j de l'équation incriminée
  - Si deux équations fausses => le bit faux est le bit M_i commun aux deux équations
  - Si trois équations fausses => le bit faux est M_4
- La distance de Hamming est bien 3 car deux messages source qui différent d'un Mi se traduisent par deux messages codés dont la partie contrôle diffère d'au moins deux C_j

Code de Hamming : Exemple

- Implémentation : le message émis est :
  $E_7E_6E_5E_4E_3E_2E_1 = M_4M_3M_2C_3M_1C_2C_1$
  (les bits de contrôle sont placés en position 2^n pour n=0..2)
- Trois équations donnent les C_j en fonction des M_i. On détermine à quelles équations participe un bit donné émis comme E_k en écrivant k sur 3 bits :
  - M_4 correspond à E_7 soit k=111 : intervient dans C_1, C_2 et C_3
  - M_3 correspond à E_6 soit k=110 : intervient dans C_2 et C_3
  - M_2 correspond à E_5 soit k=101 : intervient dans C_1 et C_3
  - C_3 correspond à E_4 soit k=100 : n'intervient que dans C_3
  - M_1 correspond à E_3 soit k=011 : intervient dans C_1 et C_2

Code de Hamming : Exemple

- En d'autres termes l'expression de C_1 va contenir tous les bits E_k dont le numéro k a 1 comme bit de poids faible : $(001)_2$, $(011)_2$, $(101)_2$, $(111)_2$ donc les bits 1, 3, 5 et 7
- L'expression de C_2 va contenir les bits : $(010)_2$, $(011)_2$, $(110)_2$, $(111)_2$ donc les bits 2, 3, 6 et 7
- L'expression de C_3 va contenir les bits : $(100)_2$, $(101)_2$, $(110)_2$, $(111)_2$ donc les bits 4, 5, 6 et 7
- Les équations sont donc (additions modulo 2 donc XOR)
  - $C_1 = E_1 = E_3+E_5+E_7 = M_1+M_2+M_4$
  - $C_2 = E_2 = E_3+E_6+E_7 = M_1+M_3+M_4$
  - $C_3 = E_4 = E_5+E_6+E_7 = M_2+M_3+M_4$
- On émet E7E6E5E4E3E2E1 = M4M3M2C3M1C2C1

Code de Hamming : Exemple

- A l'arrivée on reçoit $r_7r_6r_5r_4r_3r_2r_1 = m_4m_3m_2c_3m_1c_2c_1$, les r_k étant éventuellement différents de E_k (donc les m_i des M_i et/ou c_j des C_j)
- On reprend alors les expressions précédentes pour calculer les bits d'erreur
  - n_1 (XOR des bits r_k de numéros k=1, 3, 5, 7)
  - n_2 (XOR des bits 2, 3, 6, 7)
  - n_3 (XOR des bits 4, 5, 6, 7)
  - $n_1=c_1+m_1+m_2+m_4=r_1+r_3+r_5+r_7$
  - $n_2=c_2+m_1+m_3+m_4=r_2+r_3+r_6+r_7$
  - $n_3=c_3+m_2+m_3+m_4=r_4+r_5+r_6+r_7$

Code de Hamming : Exemple

- Si n_1, n_2 et n_3 tous nuls la transmission s'est faite sans erreur
- Sinon, par construction $n_3n_2n_1$ donne en binaire le numéro de bit erroné dans $r_7r_6r_5r_4r_3r_2r_1$ sachant que :
  - Une double erreur est détectée mais corrigée de manière inappropriée, comme l'erreur simple équivalente sur le contrôle (générant donc une triple erreur non détectée)
  - Une triple erreur peut ne pas être détectée

Code de Hamming : Exemple

- Principe généralisable à des codes [2^k-1,2^k-k-1,3] en faisant en sorte qu'un bit de l'espace source intervienne dans au moins deux bits de contrôle (donc distance de Hamming = 3) :
  - 2^k-1 bits au total dans l'espace cible
  - dont k bits de contrôle
  - donc 2^k-k-1 bits dans l'espace source
- Le plus simple d'entre eux [1,1,3] est le code par répétition : $M_1 \rightarrow M_1M_1M_1$ qui permet bien (moyennant deux bits de code) de corriger une erreur sur un seul bit de message

Codes linéaires

- Les codes de Hamming sont des cas particulier des *codes linéaires* (les équations donnant les bits du message codé en fonction des bits du message source sont linéaires), formalisables en termes de matrices :
- Matrice génératrice G de taille $(2^k-1)x(2^k-k-1)$ permet de calculer le message codé à partir du message source : C=G*M (C et M vecteurs colonne de tailles respectives (2^k-1) et (2^k-k-1))
- Matrice de contrôle de parité H de taille $(k)x(2^k-1)$ qui permet de vérifier qu'un message codé est correct : H*C=0 (vecteur nul à k composantes) ou sinon donne le numéro n (codé sur k bits) du bit erroné H*C=n
- G et M sont liées et vérifient en particulier H*G=0 (matrice nulle de taille $(k)x(2^k-k-1)$

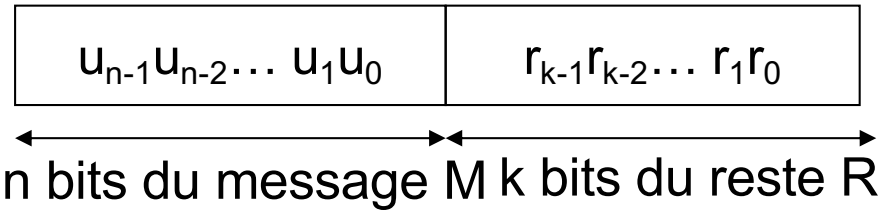
Codes linéaires

Pour le code [7,4,3] formalisé comme ci-avant (plusieurs formulations équivalentes sont bien sûr possibles) les matrices sont respectivement :

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Codes de contrôle à redondance cyclique

- Également appelés CRC (Cyclic Redundancy Check)
- Suite M de n bits à transmettre $u_{n-1}u_{n-2}\ldots u_1u_0$: considérée comme un polynôme $M(x)=u_{n-1}x^{n-1}+ u_{n-2}x^{n-2}+\ldots u_1x+u_0$
- M(x) : multiplié par x^k (donnant un polynôme de degré maximal n+k-1 dont les k bits de poids faible sont nuls)
- $x^k.M(x)$: divisé par un polynôme G(x) de degré k dit polynôme générateur du code : $x^k.M(x)=G(x)Q(x)+R(x)$ reste R(x) (de degré maximal k-1) : sur k bits
- On émet $x^k.M(x)+R(x)$: Il s'agit donc d'un code séparable :



Codes de contrôle à redondance cyclique

- La suite de bits émise est donc équivalente au polynôme :
  $E(x) = x^k.M(x)+R(x)$ que l'on peut également écrire
  $E(x) = x^k.M(x)-R(x)$: addition et soustraction équivalentes
- Mais comme $x^k.M(x)=G(x)Q(x)+R(x)$, on voit que E(x) est divisible par G(x)
- Si le polynôme reçu E'(x) est non divisible par G(x) : Erreur certaine
- Si E'(x) est divisible par G(x) : fortes chances (fonction du choix du polynôme G(x) qu'il n'y ait pas d'erreur)
- Un choix judicieux de G(x) (en général sur 12, 16 ou 32 bits) assure des taux d'erreurs non détectées inférieurs à 0,1% voire mieux

Contrôles temporels : Watchdogs

- Simple et peu coûteux, le chien de garde ou watchdog est utilisé pour détecter les erreurs (fréquentes et pas toujours tolérables) conduisant à l'inactivité (« plantage ») d'une unité centrale.
- Il s'agit d'un dispositif matériel, rafraîchi ou réarmé périodiquement par le processeur (par exemple en début de chaque cycle applicatif).
- Si le rafraîchissement n'a pas été effectué au bout d'un délai fixé, le watchdog génère un signal d'erreur (qui peut être utilisé par exemple pour effectuer un redémarrage : « reset » du processeur).

Contrôles d'exécution : Codes arithmétiques

Principe : remplacer les données par des "données sécuritaires" comprenant deux champs :

- La valeur de la donnée (partie dite "fonctionnelle")
- Une partie codée contenant une redondance d'information par rapport à la partie fonctionnelle dont la cohérence est préservée par les opérations arithmétiques

Le processeur sécuritaire codé

Une donnée (entier par exemple) est alors remplacée par un entier sécuritaire comportant deux champs :

- Partie fonctionnelle ou valeur
- Partie codée ou redondance

Naturellement les opérations arithmétiques doivent être remplacées par des opérations codées (Opérations Elémentaires : OPELs) opérant sur les deux champs

Exemple simple : La preuve par 9

Ancienne technique des écoliers pour vérifier les opérations. Principe :

Manipuler outre les opérandes leur valeur modulo 9 (redondance d'une partie de l'information) calculée facilement :

$$\rightarrow 10[9] = 1 \Rightarrow \sum_i A_i.10^i[9] = \sum_i A_i[9]$$

(le modulo 9 d'un nombre est égal au modulo 9 de la somme de ses chiffres, processus que l'on peut répéter autant que nécessaire…)

Autre exemple : La preuve par 11

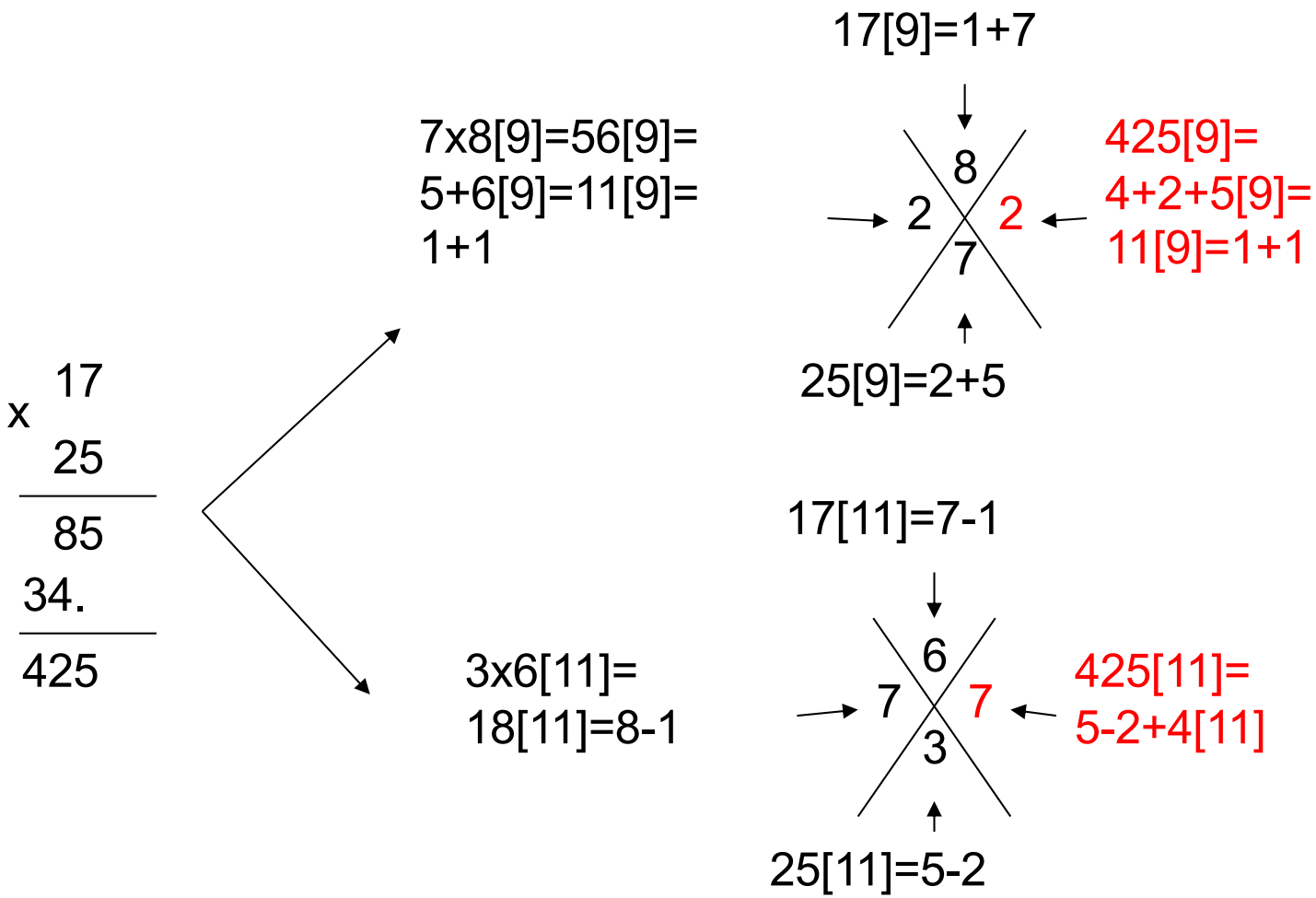
Principe analogue avec 11 :

$$\rightarrow 10[11] = -1 \Rightarrow \sum_i A_i.10^i[11] = \sum_i (-1)^i A_i[11]$$

(le modulo 11 d'un nombre est égal au modulo 11 de la différence de la somme de ses chiffres de rang pair et de la somme de ses chiffres de rang impair)

Le processeur sécuritaire codé utilisé dans certaines applications ferroviaires fonctionne à la base sur un principe analogue (preuve modulo un grand nombre premier A "clé du code")

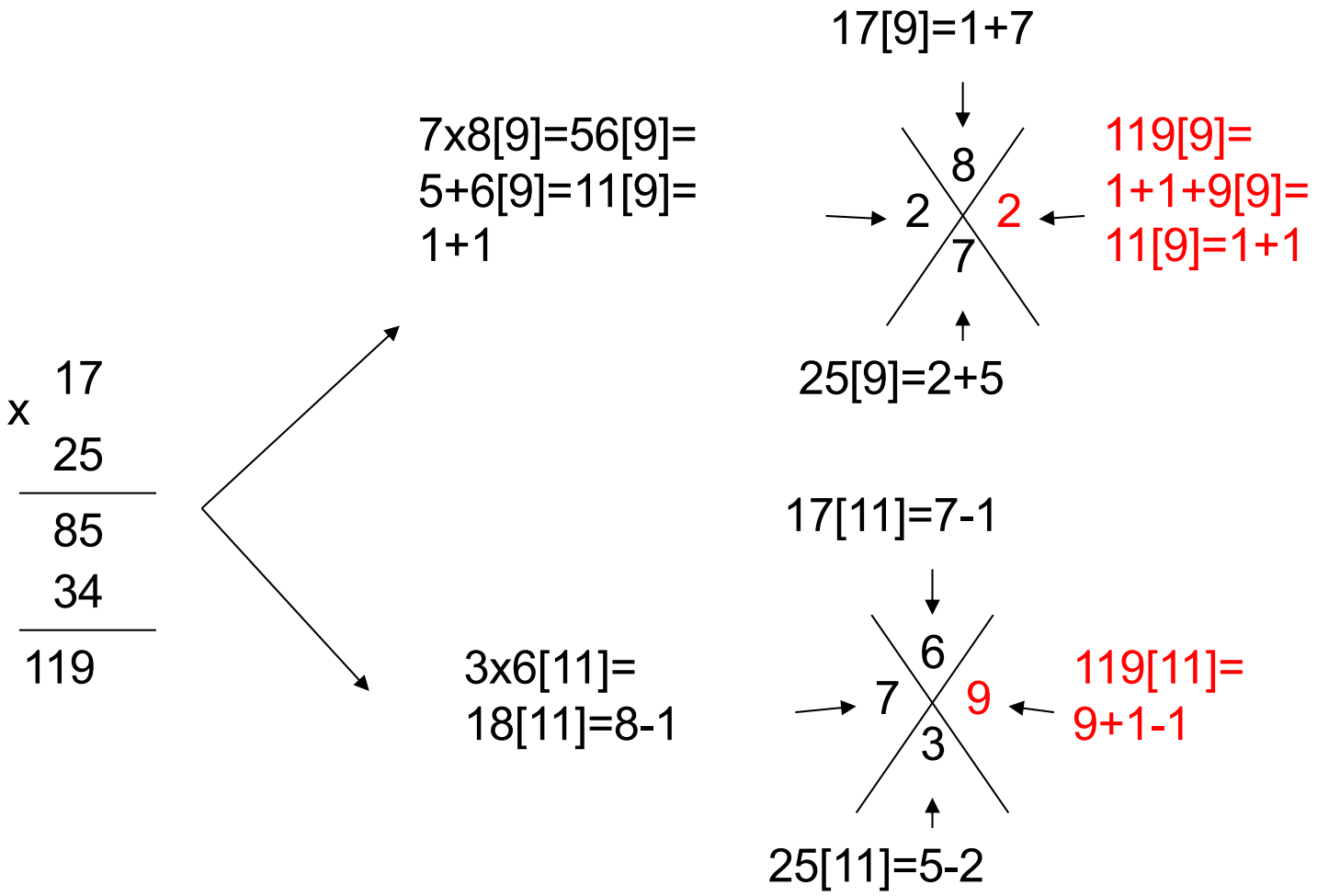
Preuves par 9 et 11 : exemple



Preuves par 9 et 11

- Si erreurs purement aléatoires (résultat tiré au hasard) on a :
  - 8 chances sur 9 de détecter les erreurs en preuve par 9
  - 10 chances sur 11 de détecter les erreurs en preuve par 11
  - 98 chances sur 99 de détecter les erreurs en combinant les deux

Preuves par 9 et 11 : erreurs non détectées

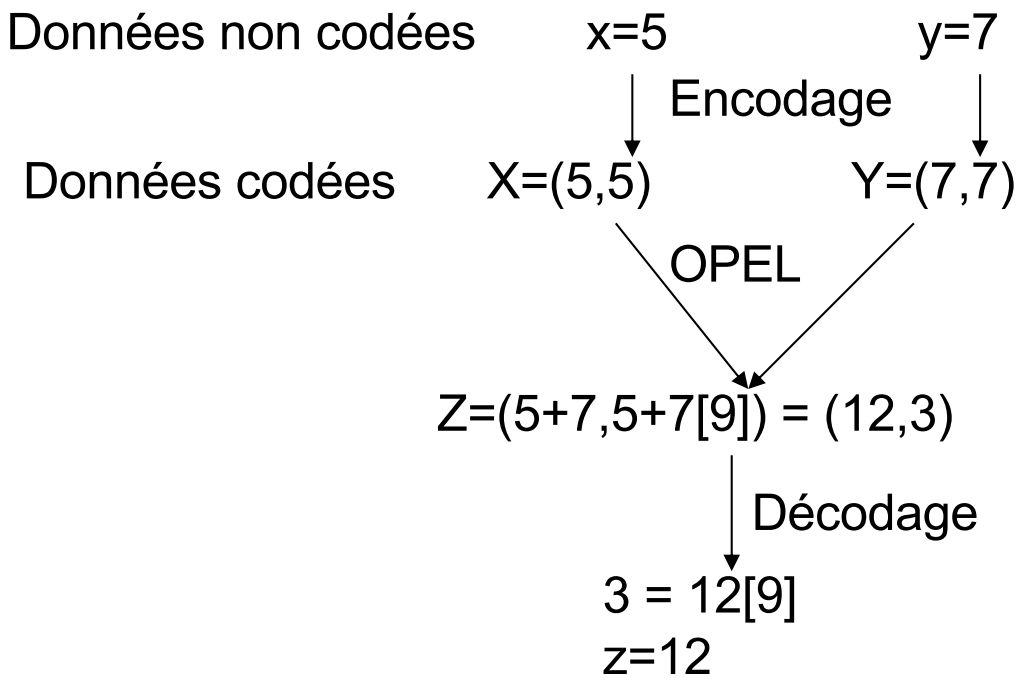


Erreurs non détectées

- En preuve par 9 : ajouter un multiple de 9, multiplier par une puissance de 10, omettre un décalage dans une opération à étages
- En preuve par 11 : ajouter un multiple de 11
- Par la combinaison des deux : multiplier par une puissance paire de 10, omettre un nombre pair de décalages dans une opération à étages

Processeur codé minimal

Réalise une preuve modulo A (grand nombre premier).
Équivalent modulo 9 :

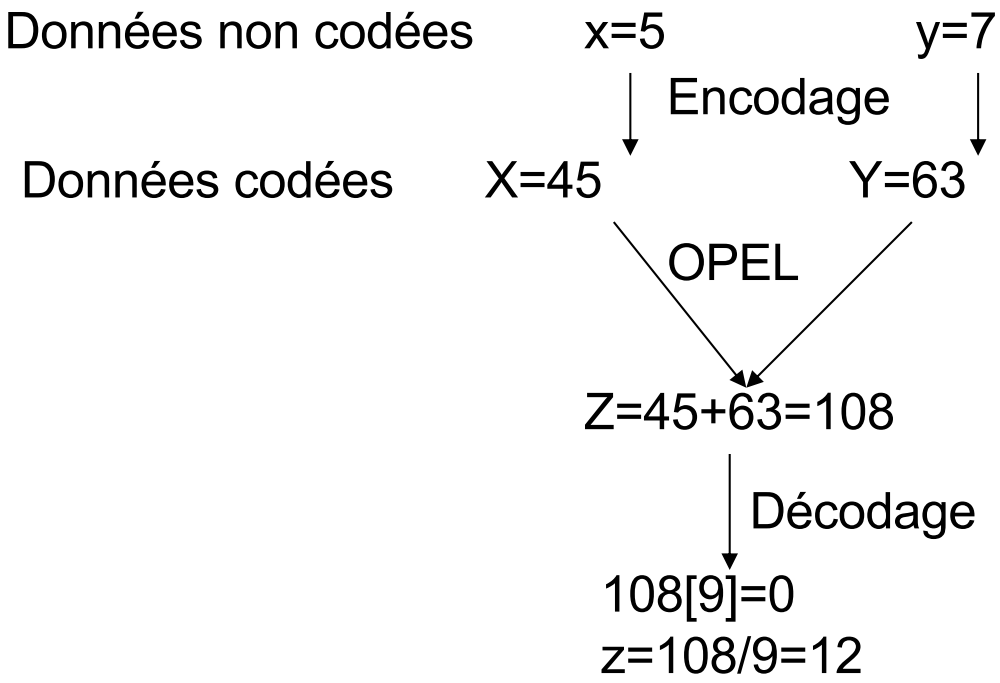


Processeur codé minimal : limitations

- Hypothèse : la clé A du code n'est pas connue du processeur (oracle externe)
- Le codage ne doit pouvoir être mis en défaut (erreur non détectée) qu'en tombant "par hasard" sur la clé (faible probabilité)
- Pour le processeur codé minimal, on met en défaut le codage en additionnant par exemple une même constante aux deux champs => insuffisant

Processeur codé : autre solution (code non séparable)

Codage : multiplication par la clé du code :



Processeur codé autre solution : limitations

- Toutes les données codées sont multiples de A (mais le processeur n'a pas l'intelligence nécessaire pour le détecter)
- Beaucoup plus grave, il faut connaître A pour extraire la partie fonctionnelle => il n'est pas possible que le processeur ignore la clé du code.
- S'il la connaît, il peut l'additionner n'importe où...
- => Il faut éviter à tout prix que la clé du code soit stockée en clair à un endroit accessible par le processeur

Processeur codé : solution adoptée

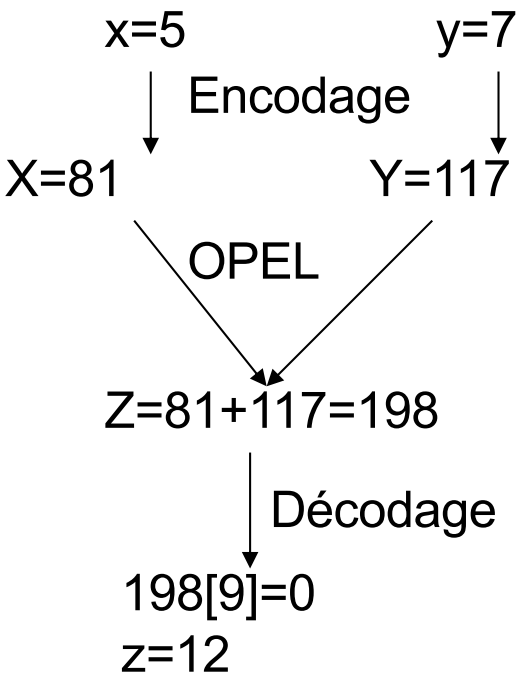
Entier sécuritaire : champ de bits en 2 parties

- Partie code : valeur<A rangée dans les k bits de poids faible (2^k>A)
- Partie fonctionnelle : valeur x de la variable rangée dans les bits de poids >k (équivaut à ranger 2^k.x dans le champ considéré globalement comme un entier)
- La partie code ne contient pas x[A] mais -2^k.x[A] (qui est bien un entier positif <A)

Processeur codé, solution adoptée Exemple

Avec A=9 et k=4 : 2^k=16>9 et -2^k[9]=-16[9]=2

valeur				code			
16x5=80				2x5[9]=1			
0	1	0	1	0	0	0	1
Ensemble=81							



valeur				code			
16x7=112				2x7[9]=5			
0	1	1	1	0	1	0	1
Ensemble=117							

Ensemble=198							
z=12							
1	1	0	0	0	1	1	0

Processeur codé : OPEL de recalage

- Selon le principe précédent les seuls mots respectant le code dans l'exemple précédent (mots dits « dans le code ») sont en notation Hexa :

Octet	Valeur	Octet	Valeur	Octet	Valeur	Octet	Valeur
00	0	48	72	87	135	C6	198
12	18	51	81	90	144	D8	216
24	36	63	99	A2	162	E1	225
36	54	75	117	B4	180	F3	243

Les autres valeurs sont erronées et sont dites « hors code »

- Problème : il faut effectuer dans le champ code l'addition modulo A. Mais il ne faut pas que A soit accessible en clair par le processeur :

Comment faire ???

Processeur codé : OPEL de recalage

- La solution consiste à détecter les débordements du champ code : si le résultat d'un calcul dans le champ code « déborde » la valeur du champ code est amputée de son bit de poids fort et contient donc $-2^k.X - 2^k$ [A] qui n'est plus cohérente avec la partie fonctionnelle.
- On rétablit un mot dans le code en rajoutant 2^k[A] qui peut être stocké quelque part sans inconvénient (ne permet pas de fabriquer des faux mots dans le code).
- Dans notre exemple : soit à ajouter 48+48
- L'addition des deux champs code déborde et laisse les 4 bits à 0. Le débordement est détecté, il faut rajouter 2^4[9]=16[9]=7
- L'octet correct est donc 87 qui est dans le code (valeur décimale 135)

Processeur codé : signatures des variables

- Le principe précédent protège contre toute corruption de valeur des données avec une probabilité de 1/A
- Il ne protège par contre pas contre les permutations d'opérandes ou d'opérateurs (effectuer X+X ou X-Y au lieu de X+Y)
- Ces erreurs étant possibles (erreurs d'adresse ou de codes d'opérandes) il faut trouver moyen de les couvrir

Processeur codé : signatures des variables

- Il faut donc rajouter à toute variable codée X (toute instance de X dans le code) une signature notée B_X.
- Les signatures des entrées sont des constantes (entiers modulo A) prédeterminées de manière pseudo-aléatoire
- Les signatures des variables calculées peuvent être prédeterminées connaissant les signatures des opérandes

Processeur codé : signatures des variables : exemple

- $X=(5;-2^k.5+3[A])$ $B_X=3$
- $Y=(7;-2^k.7+5[A])$ $B_Y=5$
- $Z:=X+Y=(12;-2^k.12+8[A])$ $B_Z=B_X+B_Y$
- $X:=Z$ $B_X=B_Z=8$

⇨ Chaque instance de variable dans le programme a une signature constante, indépendante de la valeur de la variable, et donc prédéterminable hors ligne.

Processeur codé : signatures des variables

- Les signatures prédeterminées sont stockées dans une PROM
- L'exécution du programme avec différentes valeurs de variables en entrée conduit toujours à la même évolution des signatures
- La vérification en ligne consiste à s'assurer qu'une donnée sécuritaire reste congrue à sa signature modulo A

Processeur codé : signatures des variables

- La permutation d'opérandes est maintenant détectée (sauf si les deux opérandes ont la même signature, ce qui a la probabilité 1/A de se produire)
- La permutation d'opérateurs l'est également car l'évolution des signatures en dépend :
- $B_{X+Y}=B_X+B_Y$ $B_{X-Y}=B_X-B_Y$
- Pour les autres opérateurs, c'est plus complexe !

Processeur codé : problème des branchements

- Les branchements induisent une évolution non prédéterminable (évolution différente selon la branche prise)
- SI(test) Z:=X+Y => $B_Z=B_X+B_Y$
  SINON Z:=X-Y => $B_Z=B_X-B_Y$
  FIN SI
- Obligation d'ajouter à FIN SI une procédure rajoutant BZ(SI)-BZ(SINON) si on a pris la branche SINON (OPEL de convergence)

Processeur codé : problème des boucles

- La signature finale dépend du nombre d'itérations non forcément prédéterminable
- TANT QUE(test)

  X:=X+Y => B_Y ajouté à B_X à chaque itération

  FIN TANT QUE
- Nécessité de rajouter après chaque itération la quantité B_X(Initial)-B_X(Final)

Processeur codé : datation des données

- Avec la correction précédente, un nombre d'itérations incorrect n'est pas détecté !
- Solution : à chaque itération on rajoute B_X(Initial)-B_X(Final)+δD (δD=constante)
- A l'exécution du FIN TANT QUE on soustrait N.δD (N : nombre d'itérations effectuées)

Processeur codé : forme finale du champ code

$$-2^{k}.x + B_X + D$$

Redondance informationnelle

Signature statique

Signature dynamique ou date

Remarque : les OPEL doivent tenir compte de la présence de la date :
OPEL_Addition(X,Y) : X+Y-D

Méthodes de conception en sécurité

Pour un système complet (pouvant contenir de l'informatique mais aussi des sous-ensembles électromécaniques) les principales techniques sont :

- La sécurité intrinsèque (fail safe, ou fail stop pour les systèmes informatiques),
- La redondance informationnelle (exemple codes : peuvent être utilisés pour détecter des erreurs dont le recouvrement est ensuite un fail stop),
- La redondance matérielle (architectures multiplex pour les systèmes informatiques),
- La technique contrôle et validation (très fréquente pour surveiller des processus complexes : les watchdogs peuvent être vus comme un exemple simple de contrôle et validation),
- Le surdimensionnement : typique de la conception en sécurité de pièces mécaniques pour lesquelles on ne sait pas faire autrement.

Intègrent d'entrée de jeu le « comment ça marche mal » sans se borner au « comment ça marche ».

Sécurité intrinsèque

Rappel des principes :

- « Toute défaillance doit conduire le système dans un état de sécurité au moins égale à celui où il était avant la défaillance ».
- Souvent réalisé par conception en sécurité positive :
  - ✓ Un état « restrictif » (souvent l'arrêt de l'installation) est identifié et conçu pour correspondre à l'état de plus basse énergie
  - ✓ Atteindre ou maintenir tout autre état (plus permissif) nécessite de fournir de l'énergie
  - ✓ Aucune défaillance ou combinaison vraisemblable ne conduit à fournir intempestivement de l'énergie
- Nécessite par conséquent de connaître les technologies et les modes de défaillances possibles et impossibles (il en existe des catalogues).

Sécurité intrinsèque

- La sécurité intrinsèque repose donc sur des composants de technologie particulière dont les modes de défaillance sont connus et maîtrisés.
- Certaines technologies sont donc exclues de la sécurité intrinsèque car de modes de défaillance trop nombreux (condensateurs électrochimiques, résistances au Carbone aggloméré…).
- Pour les technologies retenues, certains modes de défaillance sont considérés comme impossibles (diminution de valeur de résistance, soudure de contact de relais…) à condition que les conditions d'utilisation spécifiées (intensité max…) soient respectées.

Sécurité intrinsèque

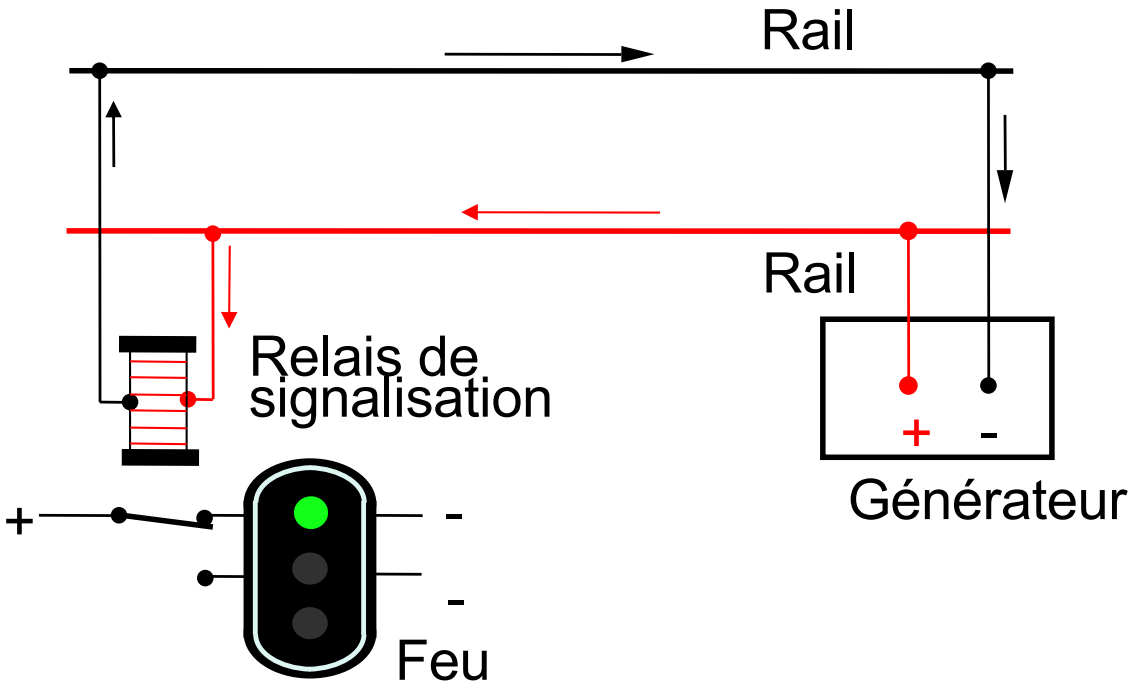
- Le respect du critère de sécurité intrinsèque se démontre par combinaison AMDEC / Arbres de causes.
- L'AMDEC doit montrer que les défaillances simples soit conduisent à l'état restrictif soit sont dormantes sur une certaine durée

Matériel à défaillances contrôlées

- Le principe de sécurité positive amène souvent à des dispositifs fonctionnant en logique inverse :
  - En frein automobile, on utilise une pression hydraulique pour serrer le frein,
  - En frein ferroviaire standard, on utilise une pression pneumatique pour desserrer le frein (la disparition de pression conduit à l'état de plus basse énergie : freinage).
- Pour cette raison, des composants spécifiques (dont certains modes de défaillance sont éliminés par conception) sont utilisés.

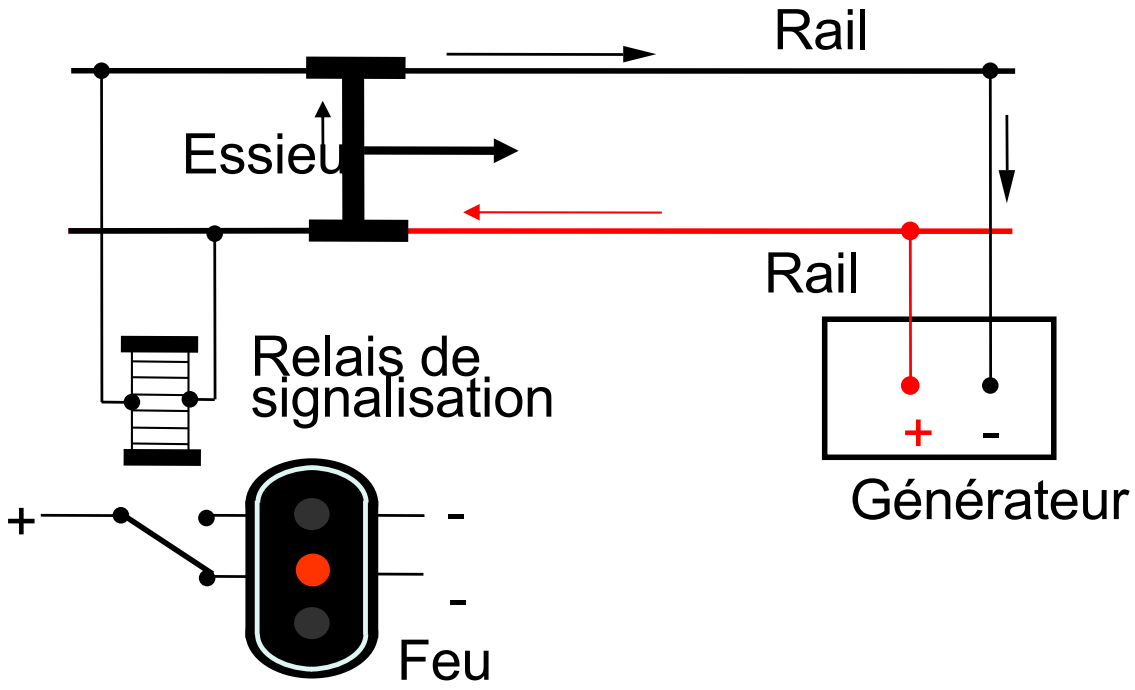
Circuits de voie

- Pour réaliser la logique de canton, la détection est effectuée par un dispositif dit « circuit de voie » (CdV) :



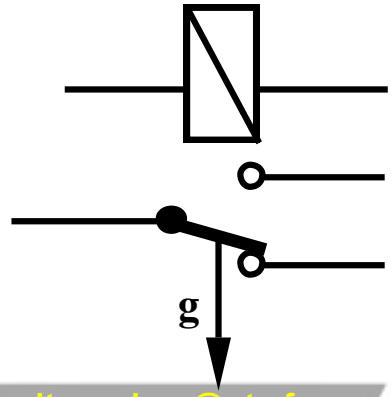
Circuits de voie

- Pour réaliser la logique de canton, la détection est effectuée par un dispositif dit « circuit de voie » (CdV) :



Circuits de voie

- La sécurité du CdV est basée sur la détection négative (absence signal = présence train).
- Toutes les défaillances (émetteur, récepteur, relais, rupture de rail) conduisent à absence signal (donc fausse information présence train, ce qui va dans le sens de la sécurité).
- Pour les relais utilisés (dits NS1) le mode de défaillance « collé en position de travail » est éliminé par conception (en particulier il est ramené à l'état repos par la pesanteur).



➢Le contact repos peut être utilisé pour allumer un feu rouge, mais la sécurité repose sur l'extinction du feu vert (un feu entièrement éteint doit être interprété comme rouge).