Cours d'architecture logicielle

Tactiques de conception

Philippe Lalanda

Philippe.lalanda@imag.fr

Rappel

- Approches pour la conception
  - Identification des composants fonctionnels
    - dérivation à partir des exigences (langage naturel)
    - dérivation à partir de modèles d'analyse (objet, fonctionnel)
  - Organisation et ajout de composants plus techniques
    - styles d'architecture généraux
    - styles d'architecture généraux par domaine
    - tactiques de conception
  - mesure de la cohésion et du couplage

Objectif de ce cours

- Définir la notion d'exigences non fonctionnelles
- Présenter des tactiques de conception répondant aux besoins non fonctionnels
- Développer des exemples

Plan

- **Exigences non fonctionnelles**
- Tactiques de conception
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Architecture et propriétés non fonct.

- L'architecture est critique pour l'atteinte de nombreuses propriétés non fonctionnelles
  - prises en compte lors de la conception
  - évaluées au niveau architectural
- L'architecture, par elle même, ne peut assurer des propriétés non fonctionnelles
  - elle fournit les fondations
  - inutiles si la conception détaillée ne prend pas le relai

Dépendances

- Pour les systèmes complexes, les propriétés ne peuvent jamais être réalisées en isolation
  - la réalisation d'une propriété a des impacts sur d'autres
    - parfois positifs
    - parfois négatifs !
  - Exemples
    - sécurité et robustesse : les systèmes les plus robustes ont le plus de failles
    - la plupart des propriétés ont des incidences négatives sur la performance

Les qualités du logiciel

- Qualités attendues d'un logiciel
  - Correction et robustesse
  - Performance
  - Disponibilité
  - Utilisabilité
  - Sécurité

Utilisateur

  - Modifiabilité
  - Testabilité
  - Réutilisabilité

Ingénieur

- Il faut faire les bons compromis et assurer les niveaux de qualité choisis (but du Génie Logiciel)

Correction et robustesse

- Un logiciel est correct lorsque son fonctionnement est consistent avec ses spécifications
  - Assume que le logiciel a été spécifié !
  - Assume que l'on peut vérifier la correction d'un logiciel !
  - La correction est une qualité nécessaire
- La robustesse traite de la capacité d'un logiciel à fonctionner lorsque l'on sort de ses spécifications
  - C'est une qualité nécessaire pour la plupart des systèmes non informatiques – pas toujours en logiciel

Efficacité et performance

- L'efficacité est une qualité interne qui traite de la façon dont le logiciel utilise ses ressources (CPU, mémoire, …).
- La performance est une qualité externe basée sur les exigences des utilisateurs (affecte l'utilisabilité)
- La performance se mesure souvent par le temps requis pour répondre à un événement. Elle dépend
  - du nombre de sources d'événements
  - du nombre de patterns d'arrivée des événements
- Considérer la performance attendue très tôt et l'évaluer (complexité des algos, simulations, …)

Disponibilité

- Un logiciel est disponible lorsqu'il est en état de fonctionner de façon correcte
- La disponibilité d'un système est la probabilité qu'il fonctionne correctement quand il est sollicité

$$\text{Disponibilité} = \frac{\text{Temps moyen de fonct.}}{\text{Tps moyen de fonct. + tps moyen de réparation}}$$

- Le niveau de disponibilité attendu doit être spécifié

Utilisabilité

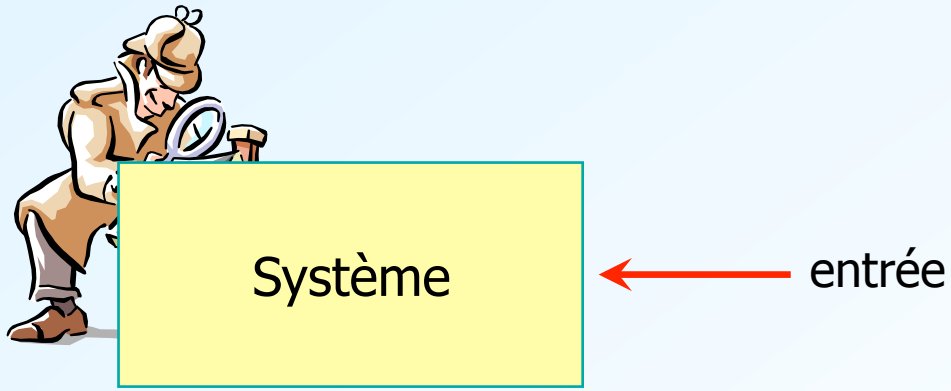
- L'utilisabilité est le niveau de facilité pour l'utilisateur d'effectuer une tâche et la qualité du support apporté
- On considère les points suivants
  - apprentissage des fonctions
  - utilisation efficace du système
  - Consistance et prédictibilité du système
  - capacité à minimiser l'impact des erreurs
  - adaptation aux besoins de l'utilisateurs
  - amélioration de la confiance et de la satisfaction de l'utilisateur
- Interfaces de plus en plus standards (Web)

Sécurité

- Un système est sécurisé lorsqu'il est capable de résister à des utilisations non autorisées tout en fonctionnant nominalement
- La sécurité est une mesure de la capacité d'un système à repousser des attaques
- La sécurité est caractérisée par rapport à six points
  - La non répudiation
  - La confidentialité
  - L'intégrité
  - L'assurance
  - La disponibilité !
  - L'audit

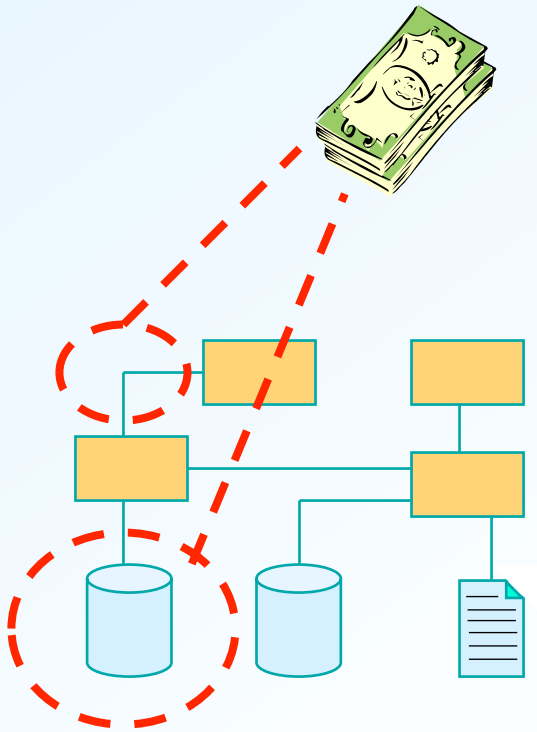
Testabilité

- La testabilité est la facilité avec laquelle on peut trouver les fautes d'un système
- Pour tester un système, on doit pouvoir
  - contrôler les inputs d'un composant et son état interne
  - observer les outputs



Maintenabilité (évolutions et réparations)

- La maintenabilité fait référence au coût des changements
- Un changement peut concerner tout aspect
  - les fonctionnalités
  - la plate-forme d'exécution (portabilité)
  - l'environnement d'interactions
  - les protocoles de communication
  - les propriétés non fonctionnelles, ...
- 60% du coût d'un logiciel
  - Maintenance corrective
  - Maintenance adaptative



Réutilisabilité

- Cette propriété fait référence à la possibilité de réutiliser des parties du système pour en construire d'autres
- La réutilisation peut aider à
  - réduire le time-to-market
  - réduire le coût
  - améliorer la qualité
- Vers un marché des COTS (grand défi d'aujourd'hui)

Note

- Il est intéressant de noter que ce sont des propriétés éminemment système
  - ne peuvent pas être considérées de façon uniquement locale
  - d'où la difficulté de les exprimer, de les traiter, de les satisfaire ensemble, …

Plan

- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Notion de tactique

- Une tactique est une stratégie locale qui a pour but de structurer un sous ensemble de composants
  - elle sous entend une organisation logique et une organisation dynamique
- Une tactique ne structure pas l'ensemble du système
  - notion de style architectural

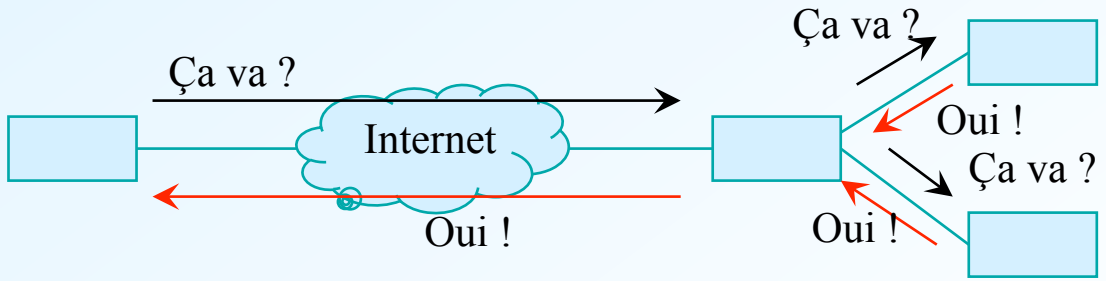
Tactiques pour la disponibilité

- Rappelons que le but est d'assurer un fonctionnement nominal du système lorsqu'il est sollicité

- Utilisation de tactiques pour
  - la détection de fautes
    - se rendre compte qu'il y a une faute
  - le traitement des fautes
    - réparer la faute : retrouver un fonctionnement nominal
  - la prévention des fautes
    - éviter les fautes

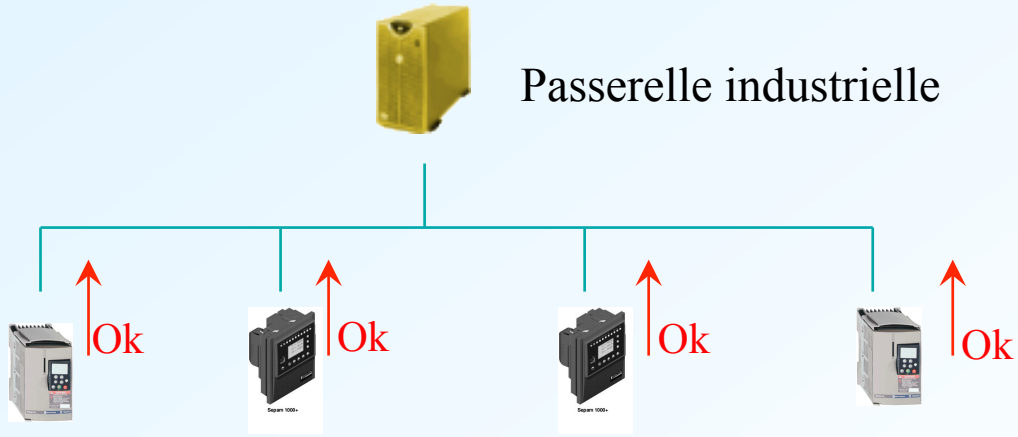
Tactique de détection des fautes - 1

- Ping : un composant envoie un "ping" et attend un écho en retour avant une certaine échéance
  - utilisé au sein d'un ensemble de composants collaborant pour exécuter une tâche donnée
  - utilisé dans des architectures distribuées (client / serveur)
  - les détecteurs peuvent être organisés en hiérarchies
    - les détecteurs de haut niveau font des "pings" sur des détecteurs de plus bas niveau
    - moins coûteux en communication qu'un détecteur distant qui "ping" tous les processus



Tactique de détection des fautes - 2

- Heartbeat : un composant émet périodiquement un battement écouté par d'autres composants
  - "jusque là tout va bien …"
  - si on manque un battement, on fait l'hypothèse que le composant émetteur est en faute
  - un battement peut également contenir des données

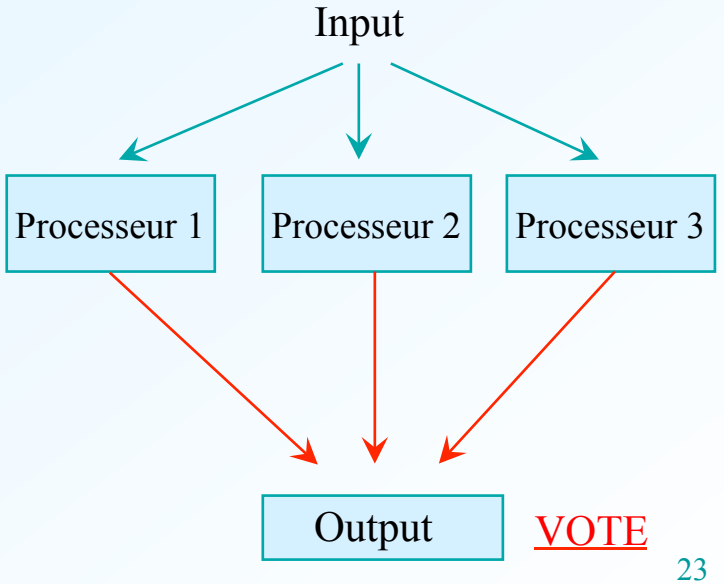


Tactique de détection des fautes - 3

- Programmation d'exceptions
  - le principe est de répertorier les fautes possibles et de déclencher une exception lorsqu'une de ces fautes est rencontrées
  - note : une exception correspond à du code

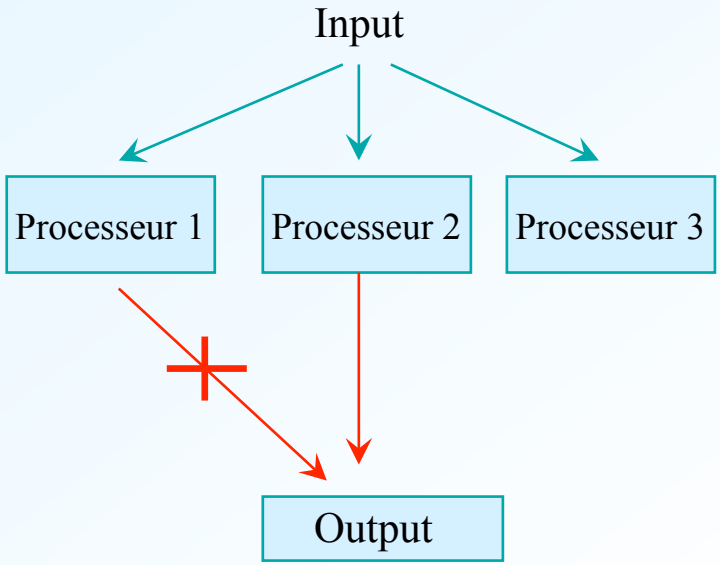
Tactique de traitement des fautes - 1

- Redondance avec vote
  - plusieurs processeurs redondants reçoivent les mêmes inputs et envoient leurs outputs à un tiers.
    Si les résultats diffèrent : le tiers "vote" puis met en échec les processeurs "faux"
  - on peut voter à la majorité ou privilégier des composants
- Remarques
  - les processus n'exécutent pas forcément les mêmes algos
  - pour les systèmes sensibles, on utilise des algorithmes différents sur plate-formes différentes



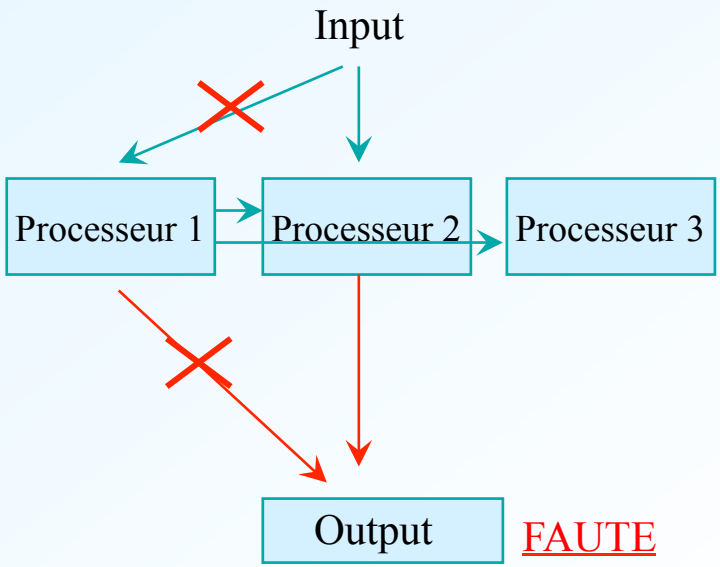
Tactique de traitement des fautes - 2

- Redondance active
  - plusieurs processeurs redondants réagissent aux mêmes inputs en parallèle. On utilise une seule réponse, définie statiquement (souvent la première)
  - quand une faute survient, on change de composant
- Remarques
  - cette solution est très rapide
  - elle ne résout pas le problème de la détection d'une faute
  - dans certains systèmes distribués, on applique la redondance à la communication (chemins parallèles)



Tactique de traitement des fautes - 3

- Redondance passive
  - un seul composant réagit aux inputs et informent les composants redondants du nouvel état courant
  - quand une faute survient, on change de composant
- Remarques
  - on peut changer de primaire de façon périodique pour accroître la disponibilité
  - le primaire est en charge de la synchronisation des redondants



Tactique de traitement des fautes - 4

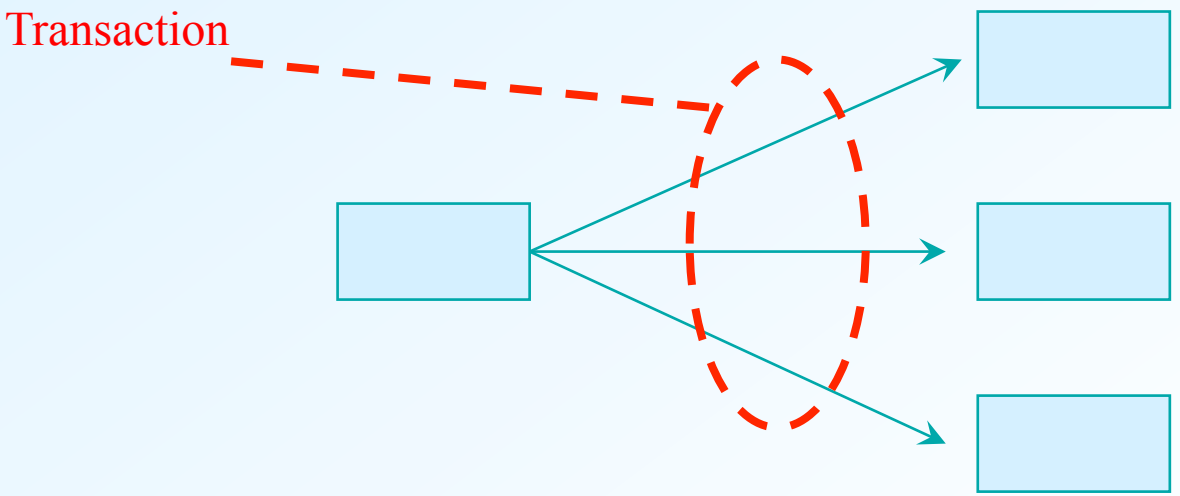
- Plate-forme de secours
  - une plate-forme complète est configurée pour remplacer plusieurs composants en faute
  - le système primaire sauvegarde ses données et ses états régulièrement sur une base persistante
- Remarques
  - quand une faute survient, il faut rebooter la plate-forme et initialiser l'état
  - le changement dure quelques minutes

Tactique de prévention des fautes - 1

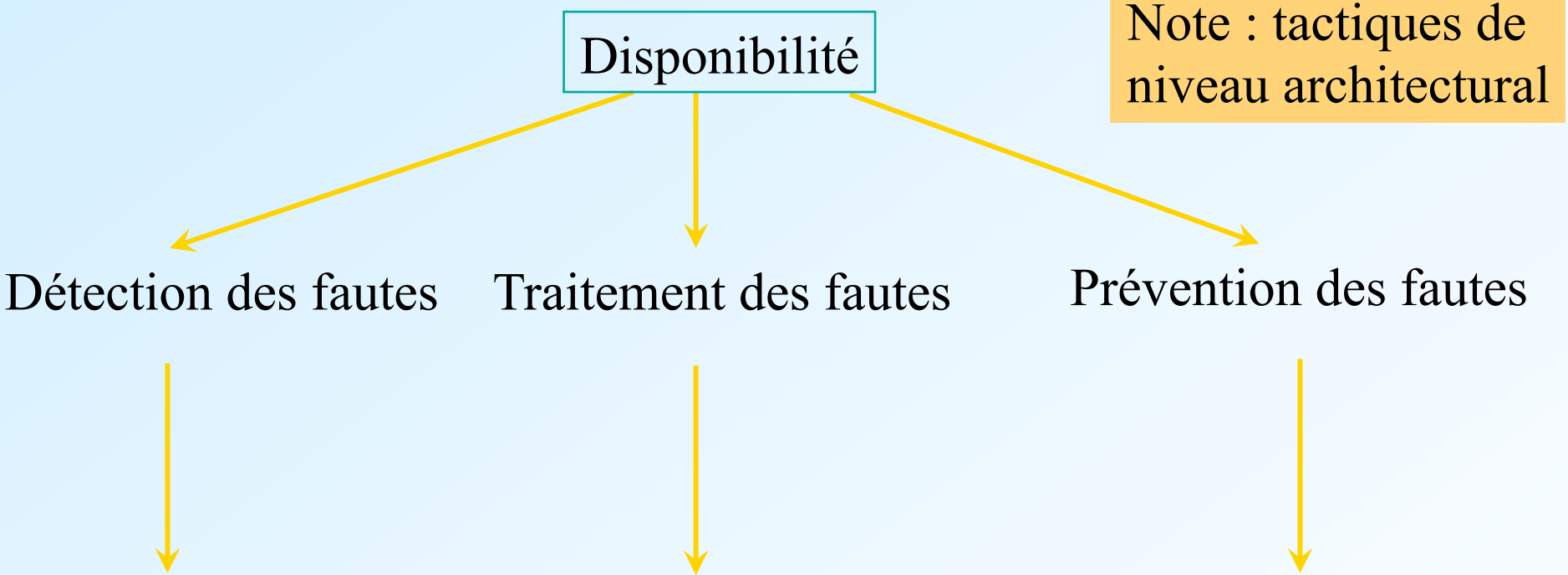
- Retrait d'un composant
  - on retire un composant pour effectuer des opérations de maintenance prédictive
  - le retrait peut être automatique et être géré au niveau architectural
  - si le retrait est manuel, le système doit être conçu pour le supporter
- Exemple
  - reboot périodique d'un composant pour éviter des fuites de mémoire

Tactique de prévention des fautes - 2

- Mise en place de transactions
  - une transaction permet de gérer de façon globale un ensemble d'actions (notamment en cas d'annulation)
- Avantages
  - assure la cohérence des données en cas d'échec d'une étape
  - permet d'éviter les collisions en cas d'activités parallèles



Résumé des tactiques de disponibilité



Plan

- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

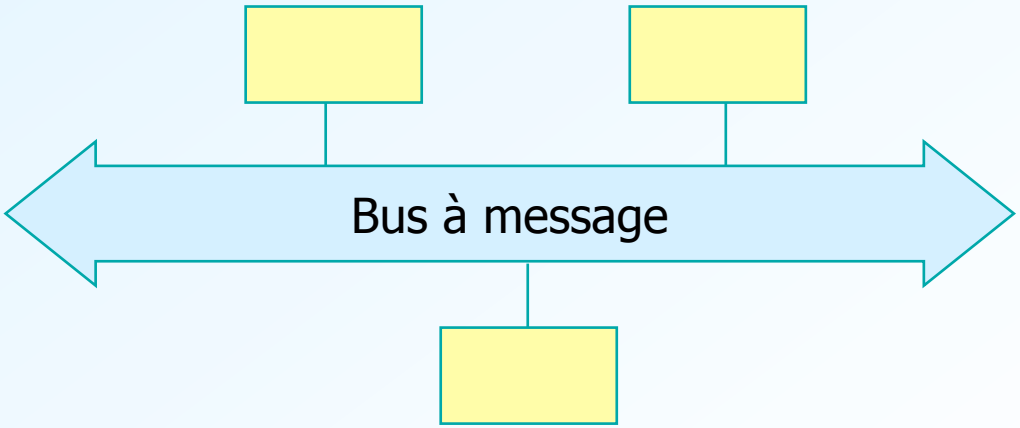
Modifiabilité

- Rappelons que le but est de contrôler le temps et le coût nécessaires pour réaliser, tester et déployer les modifications

- Utilisation de tactiques pour
  - localiser les modifications
  - éviter les effets de bords
  - différer les "bindings"

Tactique pour localiser les modifications - 1

- Forte cohésion des composants
  - les composants doivent regrouper des fonctionnalités sémantiquement proches
  - ils doivent être cohérents et complets (limitation des dépendances sémantiques)
  - une sous-tactique est d'abstraire les services communs
    - utilisation de middlewares
    - utilisation de frameworks



Tactique pour localiser les modifications - 2

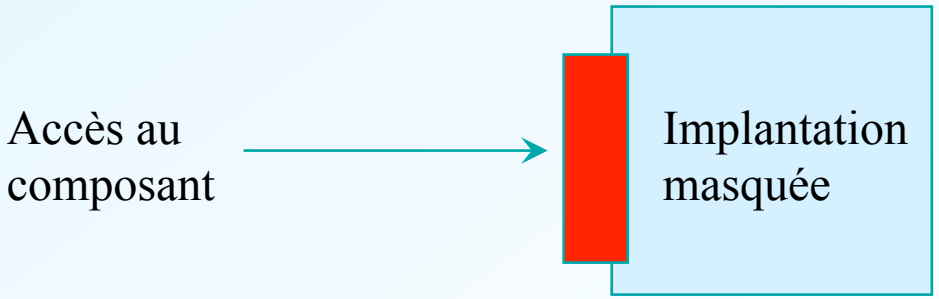
- Identification des évolutions potentielles
  - il s'agit de limiter le nombre de composants à modifier pour les évolutions les plus probables
- Remarques
  - cette tactique ne se préoccupe pas de sémantique
    - peut s’opposer à la cohésion !
  - elle est difficile à utiliser
    - bien anticiper est ardu (et coûteux en temps)
    - un écueil classique est de trop compliquer

Tactique pour localiser les modifications - 3

- Généralisation des composants
  - il s'agit ici de concevoir des composants très génériques assurant plus de fonctionnalités que nécessaire
  - quand un composant est très général, les changements concernent plutôt les paramètres d'entrée que l'intérieur du composant
- Remarques
  - on n'anticipe pas les changements, on les inclut
  - complexe et coûteux

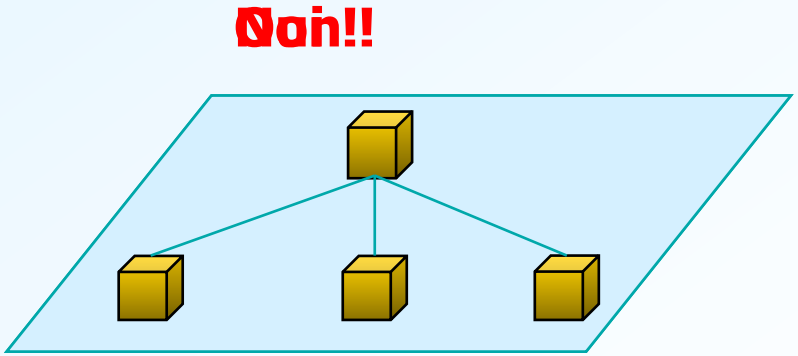
Tactique pour éviter les effets de bord - 1

- Masquer l'information
  - définition de données privées non accessibles via les ports de communication des composants
  - séparation des interfaces et des implantations
- Remarques
  - le but est d'isoler les modifications au niveau des données privées (pas de modifications d'interface qui entraîne de modifier d'autres composants)



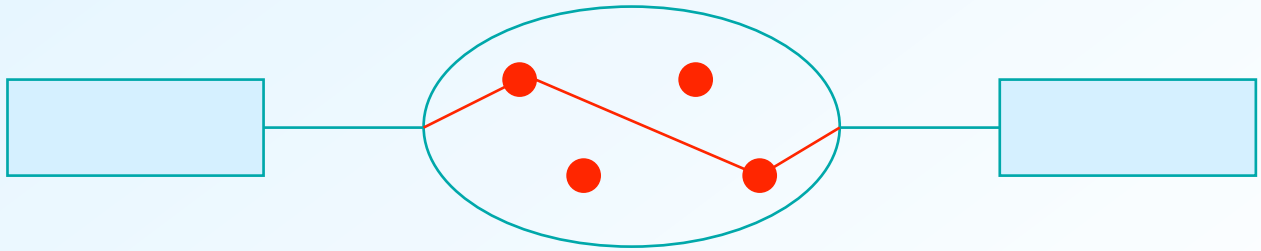
Tactique pour éviter les effets de bord - 2

- Faible couplage entre les composants
  - limiter le nombre de communication
  - privilégier des communications de données par valeur
- Remarque
  - cette tactique est liée à "cohérence et complétude sémantique"



Tactique pour éviter les effets de bord - 3

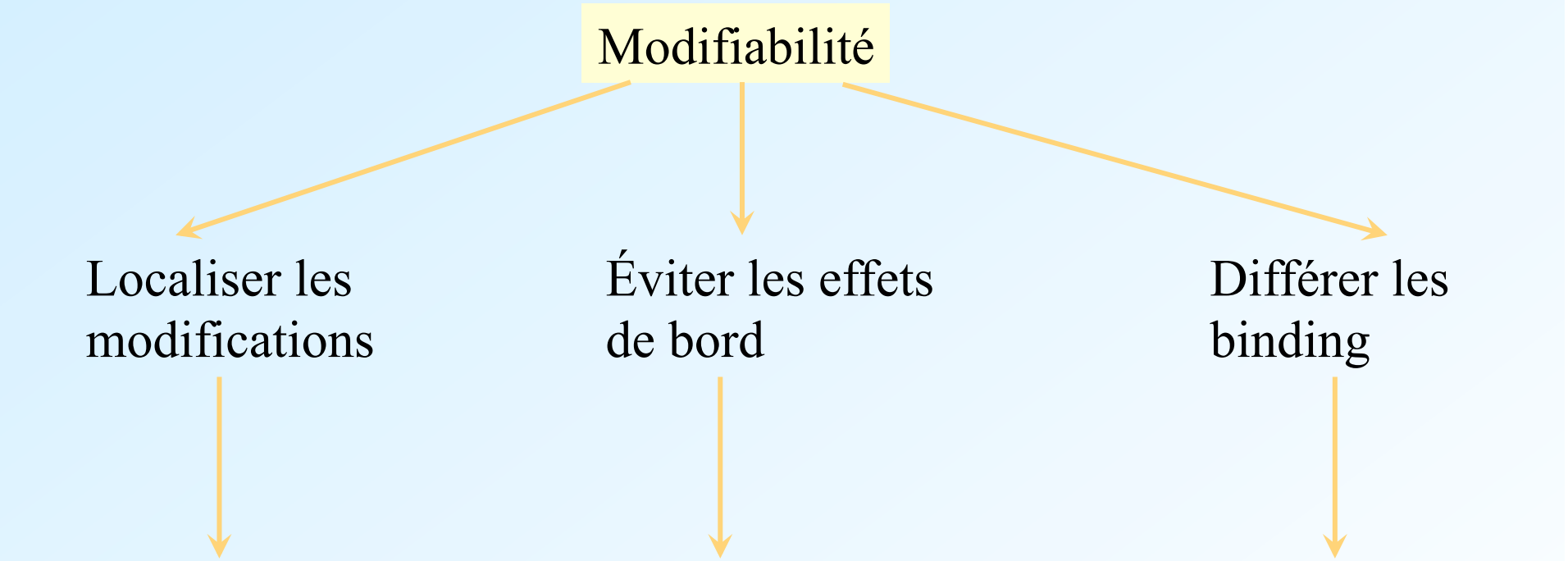
- Utilisation d'intermédiaires entre les composants
- Types d'intermédiaires
  - une façade, un pont, un proxy … peuvent convertir les syntaxes de services
  - un broker peut masquer les modifications d'identité
  - un serveur de nom permet de modifier les localisations
  - une factory peut créer des instances au besoin



Tactiques pour différer les bindings

- Tous mécanismes permettant de mettre les composants en relation de façon dynamique
- Exemples
  - mécanismes de plug&play (introduction de composants broker, registry, etc.)
  - introduction d'un composant de configuration dynamique (gestion d'un fichier)
  - introduction d'un composant de gestion du chargement dynamique

Résumé des tactiques de modifiabilité



Plan

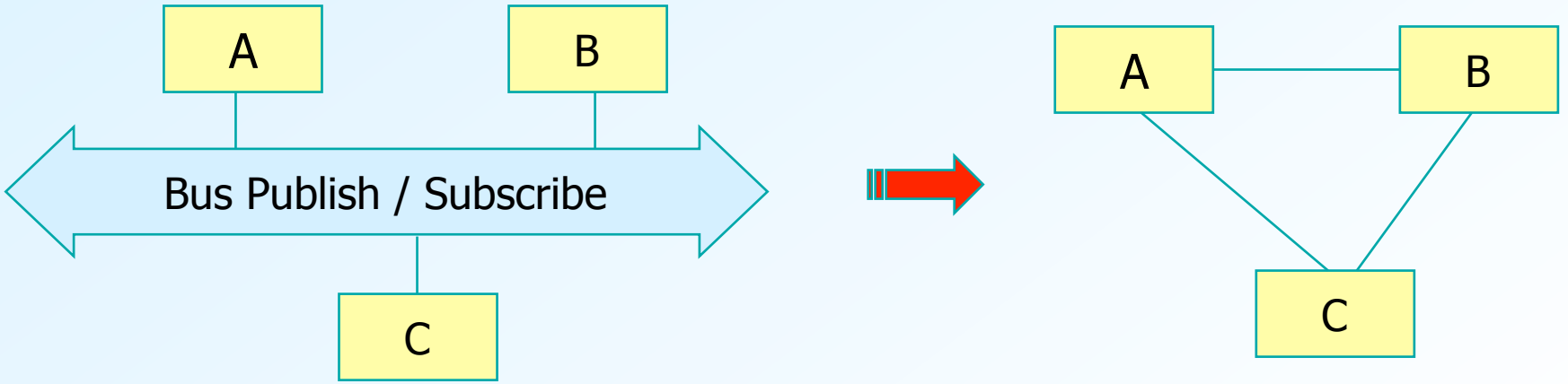
- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Performance

- Rappelons que le but est de générer une réponse à un événement avant une certaine échéance
- Utilisation de tactiques pour
  - optimiser les calcul
    - faire en sorte que les calculs soient le plus efficaces possible
  - augmenter les ressources
    - affecter plus de ressources aux calculs
  - gérer les ressources
    - contrôler finement l'utilisation des ressources

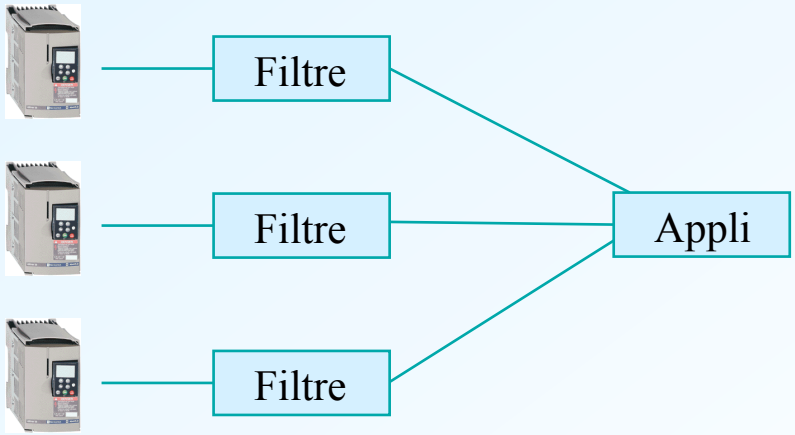
Tactique pour optimiser les calculs - 1

- Limiter le nombre de composants
  - identifier les composants non fonctionnels et les éliminer
  - il s'agit, par exemple, d'éliminer les intermédiaires (broker, proxy, …)
- Remarque
  - exemple typique où un compromis entre performance et modifiabilité doit être fait



Tactique pour optimiser les calculs - 2

- Réduire la fréquence de communication entre composants
  - il s'agit de réduire la fréquence d'échantillonnage des événements, notamment externes
- Remarque
  - certains systèmes ont des fréquence de traitement trop élevées à cause d'exigences excessives ou par commodités de programmation (synchronisation d'événements par exemple)



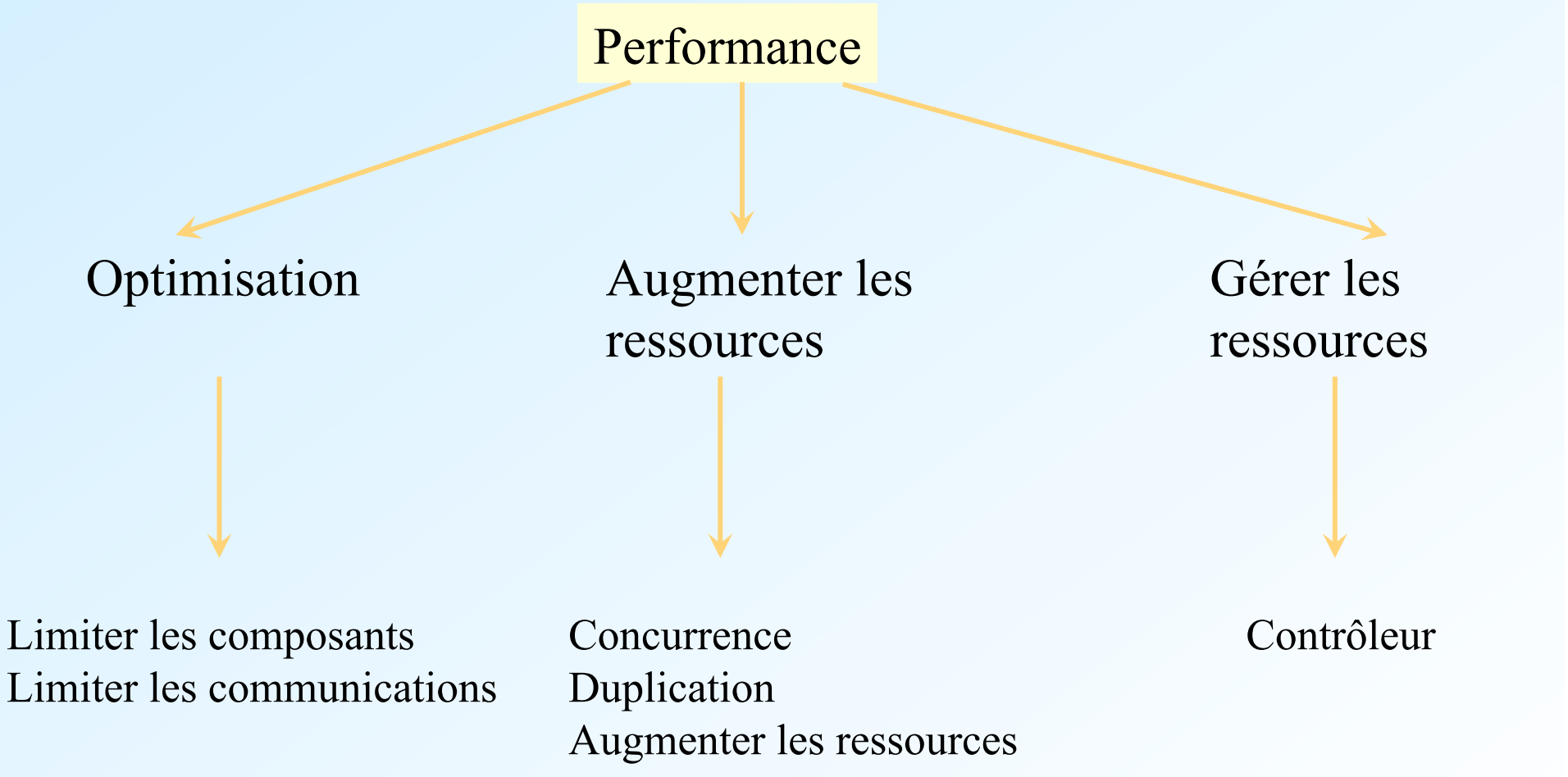
Tactiques pour augmenter les ressources

- Introduire la concurrence
  - traitement en parallèle des événements entrant
  - attention à la gestion du "load balancing" (projection des processus sur les ressources)
- Dupliquer les données ou les calculs
  - utilisation de caches et/ou de machines supplémentaires
  - attention à la synchronisation des dupliqués
- Augmenter les ressources physiques

Tactique pour gérer les ressources

- Utilisation d'un contrôleur de composants
  - pour limiter les durées d'exécution
    - besoins en algorithmes incrémentaux
  - pour gérer les activations
    - calcul de priorités statiques
    - calcul de priorités dynamiques
    - utilisation de mécanismes de préemption

Résumé des tactiques de performance



Plan

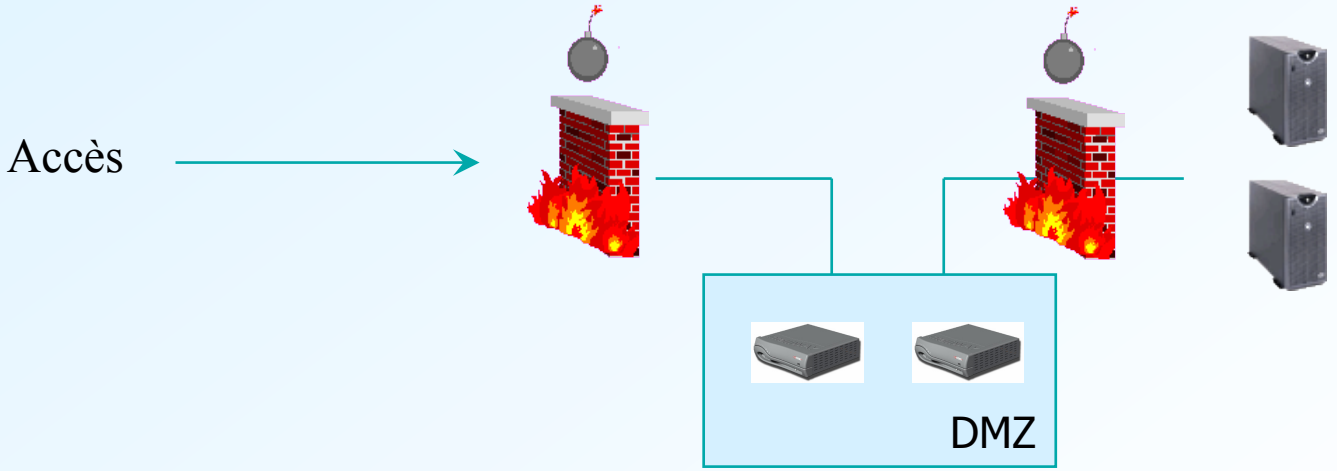
- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Sécurité

- Rappelons que le but est de garantir un fonctionnement nominal tout en résistant aux attaques
- Utilisation de tactiques pour
  - résister aux attaques
  - détecter les attaques
  - traiter les attaques

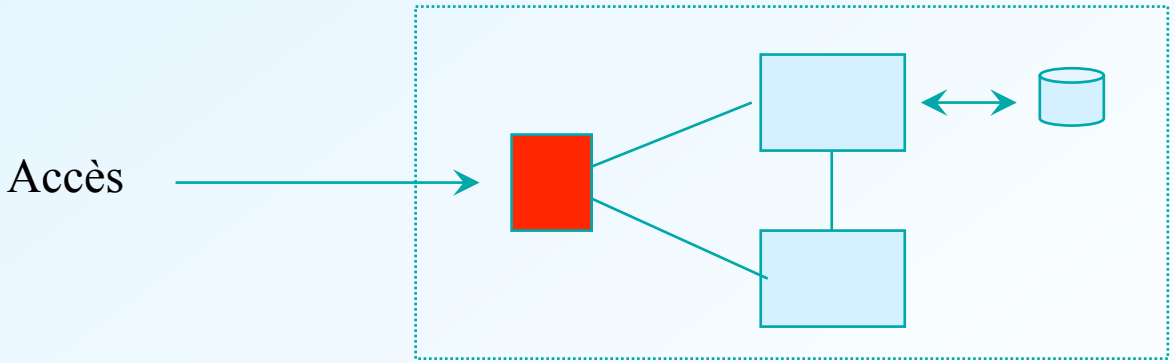
Tactique de résistance aux attaques - 1

- Limitation de l'exposition
  - répartition des services sur différents composants et différents "hosts"
  - utilisation d'un composant de type firewall
  - utilisation d'une DMZ (entre l'Internet et le firewall)
- Remarque
  - en opposition avec les tactiques de performance, voire de disponibilité



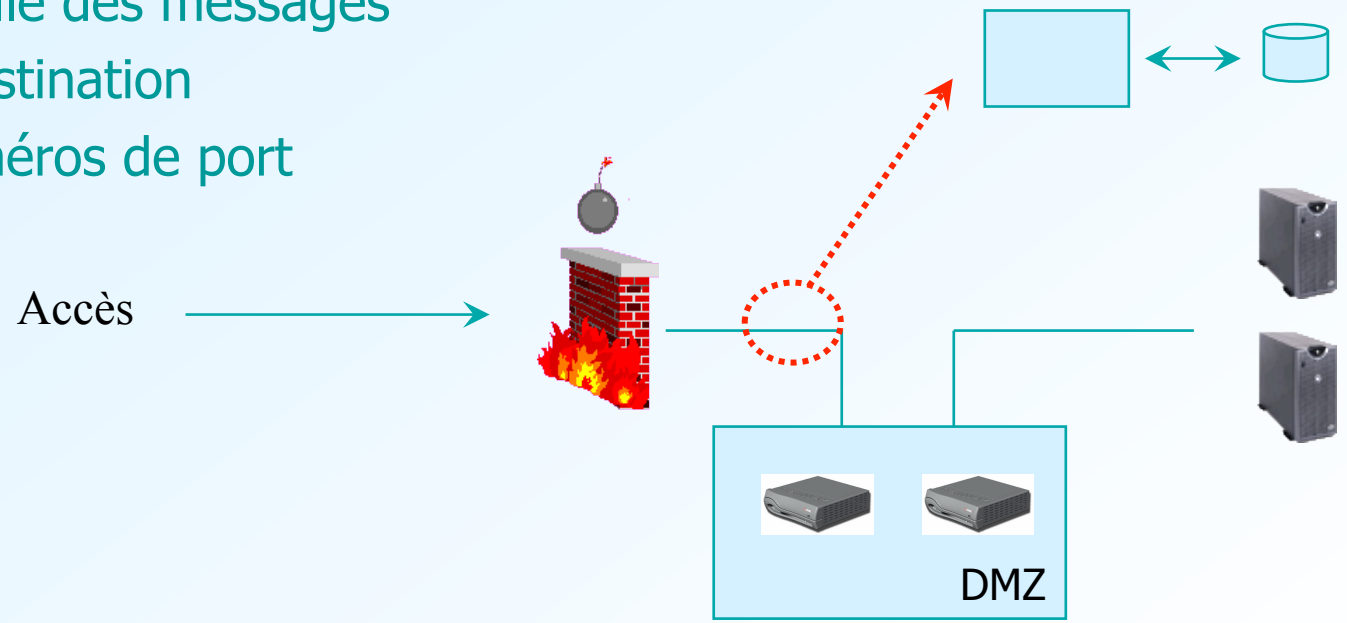
Tactique de résistance aux attaques - 2

- Protection des communications
  - techniques de cryptographie (avec des clés symétriques ou asymétriques)
  - création d'un composant global assurant la sécurisation des communications
    - Cryptographie
    - Établissement de profils d'utilisateurs, de mots de passe, …
    - Logging



Tactique de détection des attaques

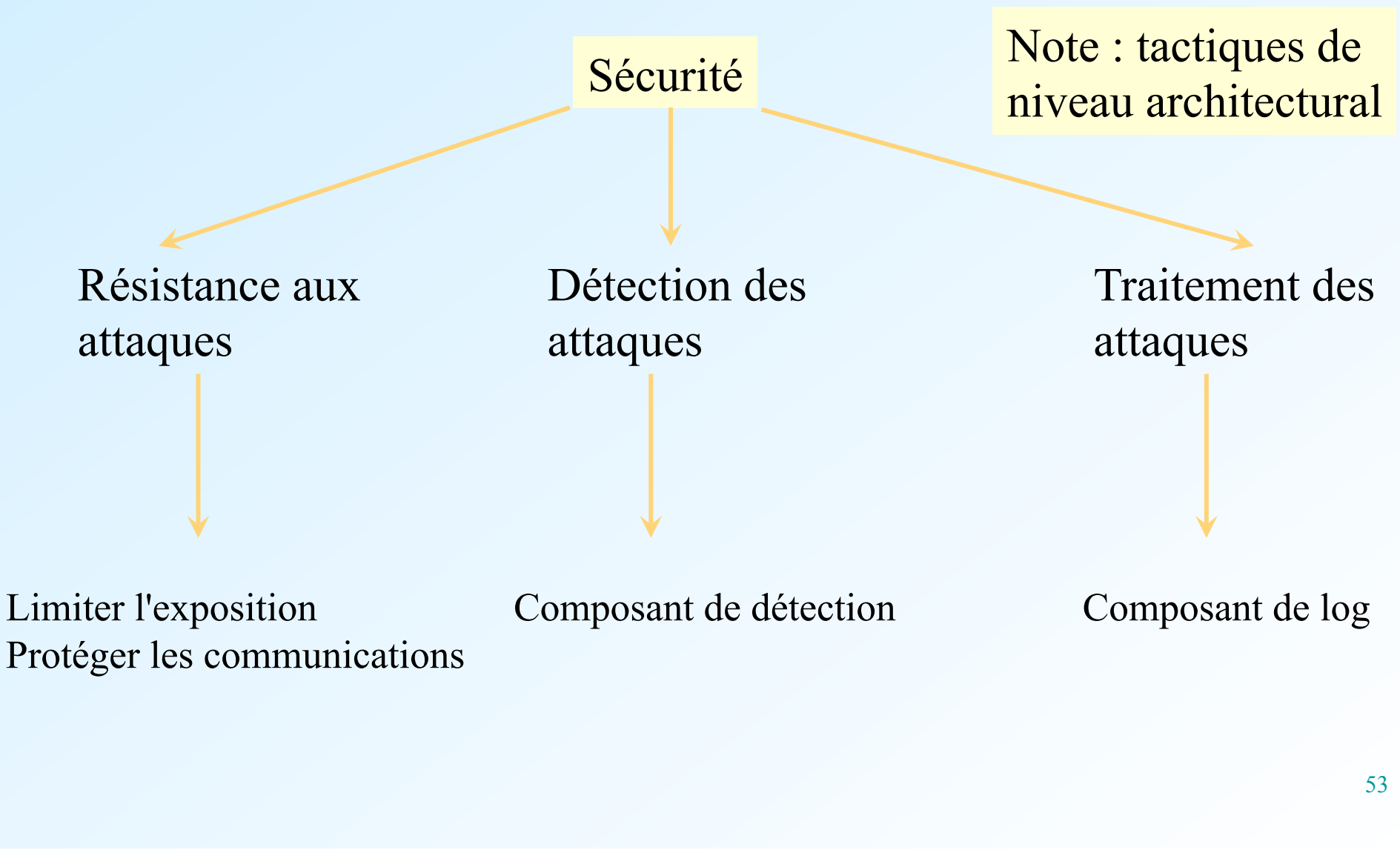
- Composant de détection d'intrusion
  - comparaison des patterns de communication avec ceux d'une base de données
  - en cas de soupçon, comparaison avec des patterns d'attaques connues
  - pour faire ces comparaisons, certains messages sont filtrés sur la base
    - des protocoles utilisés
    - de la taille des messages
    - de la destination
    - des numéros de port



Tactique de traitement des attaques

- Composant de Log
  - stockage des informations de communication
  - stockage de l'état courant avec une attention particulière aux informations administratives (mots de passe, liste d'utilisateurs, noms de domaine, etc.)
- Remarque
  - le composant de log est souvent l'objet d'attaques et doit être protégé (conception détaillée)
  - tactique proche des tactiques de redondance pour la disponibilité

Résumé des tactiques de sécurité



Plan

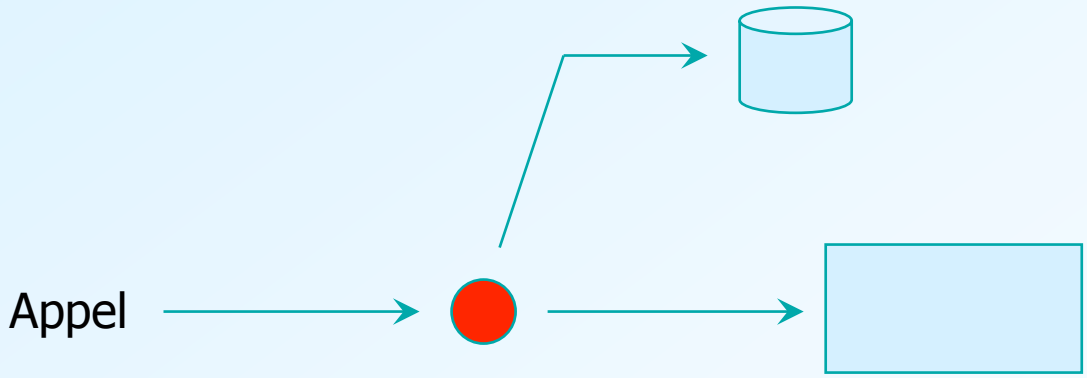
- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Testabilité

- Rappelons que le but est de faciliter la réalisation de tests lors d'un incrément logiciel
- Utilisation de tactiques pour
  - générer des tests
  - orienter le système

Tactique de génération de tests

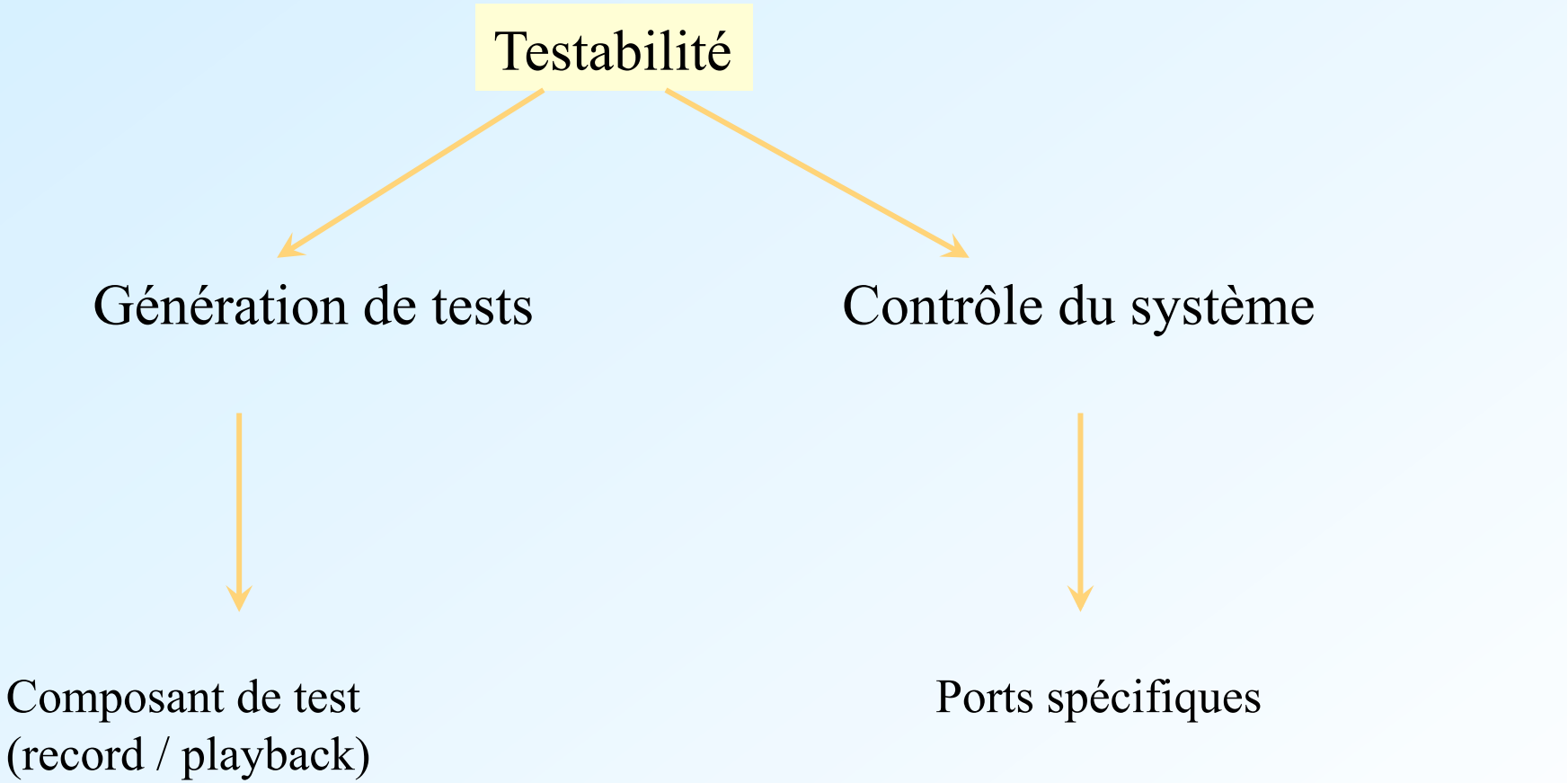
- Composant de test
  - le principe est de capturer et de mémoriser les informations (entrantes et sortantes) traversant une interface durant une exécution normale
  - permet la génération de tests



Tactique de contrôle du système

- Ports de communication spécifiques pour le test
  - mise à disposition de méta-données permettant d'orienter le fonctionnement d'un composant
  - ajout d'une interface de contrôle permettant de connaître l'état du système (charge, capacité, utilisateurs, etc.) et couplage à un système de test qui enregistre les valeurs nominales
- Remarque
  - coûteux à mettre en place mais extrêmement efficace

Résumé des tactiques de testabilité



Plan

- Exigences non fonctionnelles
- **Tactiques de conception**
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- Conclusion

Utilisabilité

- Rappelons que le but est de faciliter le travail de l'utilisateur et de lui apporter un support constant

- Tactiques
  - construction de modèles utilisés à l'exécution

Construction de modèles

- Composant "modèle de tâche"
  - un modèle de la tâche en cours est utilisé par le système pour aider l'utilisateur (la correction automatique de fautes possible lorsque le système connaît le contexte)
- Composant "modèle de l'utilisateur"
  - un modèle de l'utilisateur détermine le niveau d'expertise de l'utilisateur, ses comportements, etc.
- Composant "modèle du système"
  - un modèle du système permet de prédire les réactions du système et de mieux présenter les informations

Plan

- Exigences non fonctionnelles
- Tactiques de conception
  - Tactiques pour la disponibilité
  - Tactiques pour la modifiabilité
  - Tactiques pour la performance
  - Tactiques pour la sécurité
  - Tactiques pour la testabilité
  - Tactiques pour la utilisabilité
- **Conclusion**

Résumé

- Nous avons présenté un ensemble de tactiques favorisant la réalisation des propriétés non fonctionnelles
  - il y en a d'autres
  - il faut prendre en compte le contexte pour les appliquer
- La principale difficulté est de les assembler
  - besoin de compromis