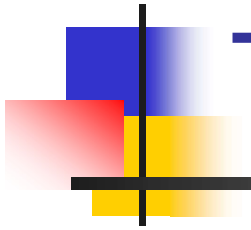
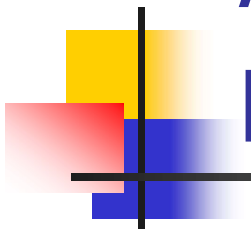


Cours d'architecture logicielle
Tactiques de conception

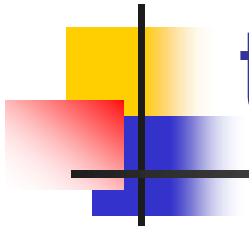
Lydie du Bousquet
Philippe Lalanda
Université Grenoble-Alpes



Architecture et propriétés non fonctionnelles

- 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
  - inutile si la conception détaillée ne prend pas le relais

Notion de tactique de conception

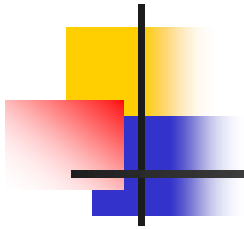


- une décision architecturale
- pour gérer un attribut de qualité

- une stratégie locale
- pour structurer un sous ensemble de composants

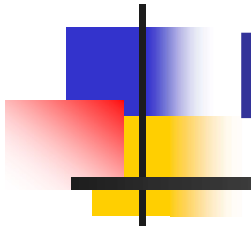
- Influence l'organisation logique, dynamique et/ou physique
- Besoin de traçabilité

  => tactiques explicitées dans les différentes vues



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



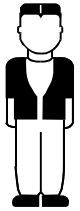
Cours d'architecture logicielle
Propriétés non fonctionnelles

Lydie du Bousquet
Philippe Lalanda
Université Grenoble-Alpes

Les qualités du logiciel

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

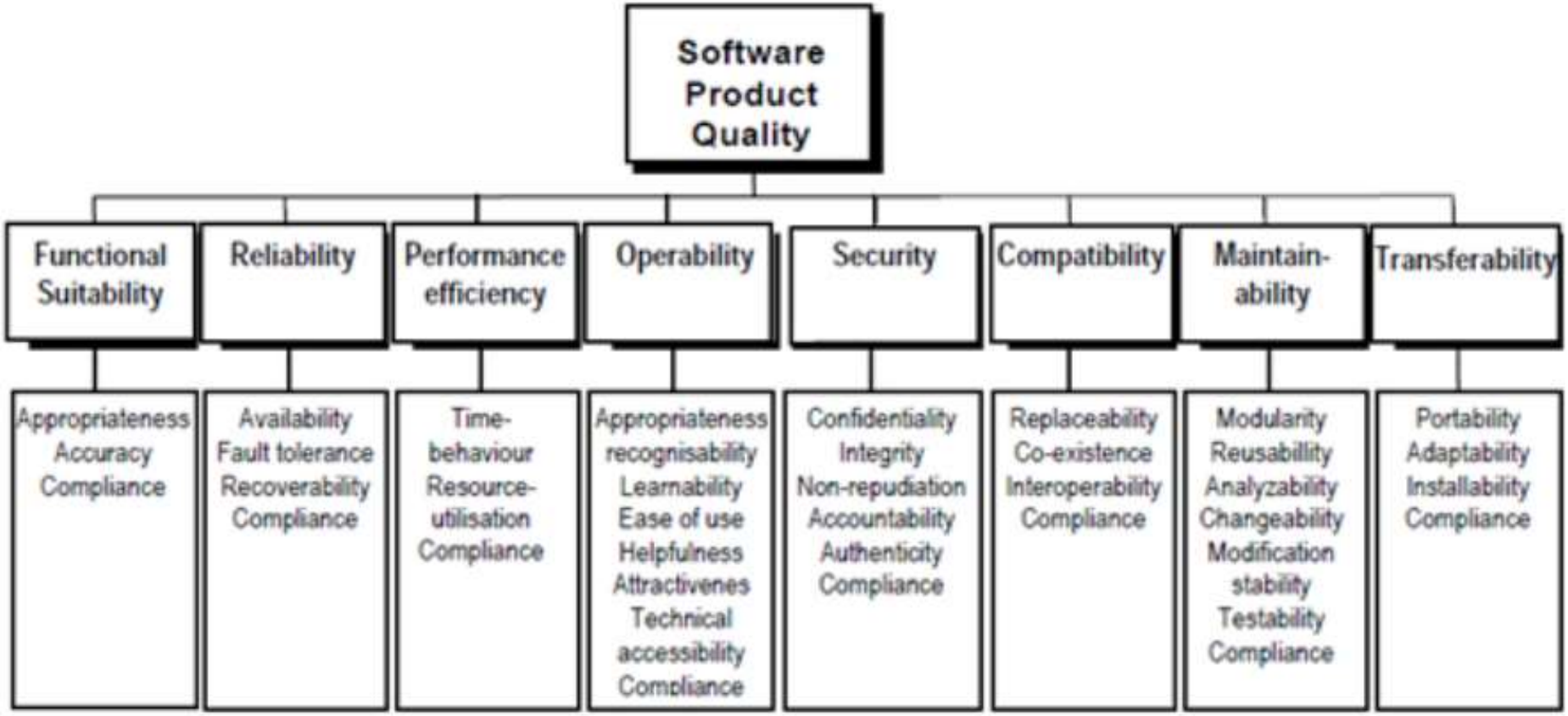
Utilisateur



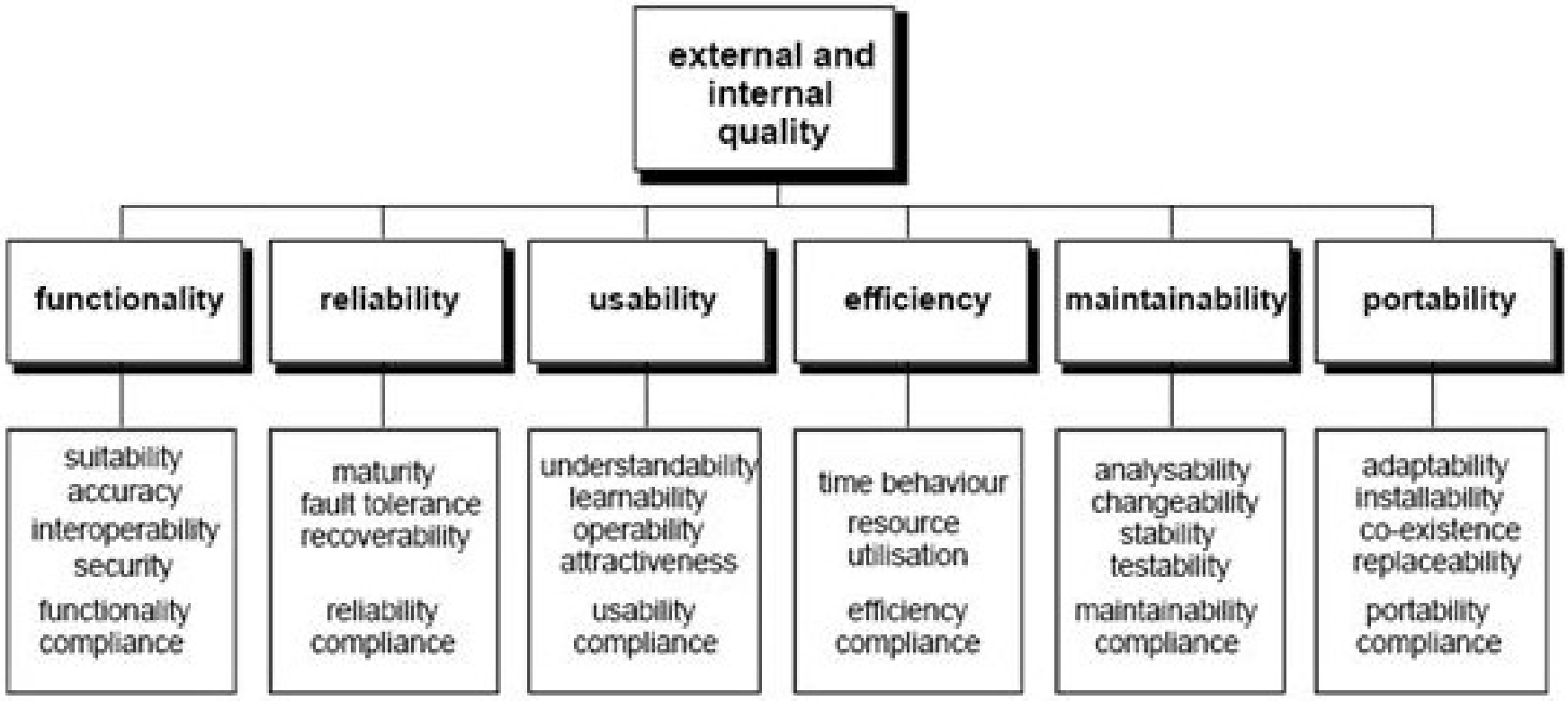
Ingénieur

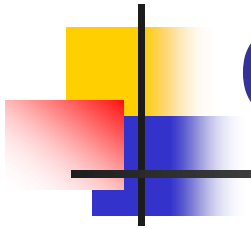
- Il faut faire les bons compromis et assurer les niveaux de qualité choisis (but du Génie Logiciel)

ISO 25000 - Square



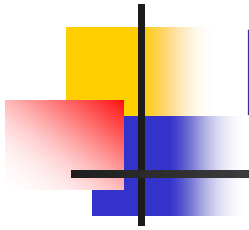
ISO 25000 - Square





Correction et robustesse

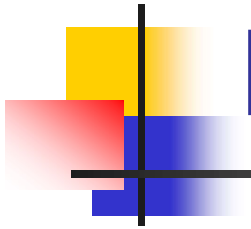
- Un logiciel est correct lorsque son fonctionnement est consistent avec ses spécifications
  - Assume que le logiciel a été spécifié !
  - Assume que l'on peut vérifier la correction d'un logiciel !
  - La correction est une qualité nécessaire
- La robustesse traite de la capacité d'un logiciel à fonctionner 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)
  - 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

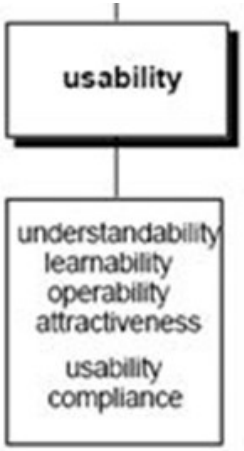
- Un logiciel est disponible lorsqu'il est en état de fonctionner de façon correcte
  - Une faute (ou combinaison de fautes) est la cause potentielle de défaillance(s)
  - Une défaillance apparaît quand le système délivre une réponse qui n'est pas consistent avec la spécification
- La disponibilité d'un système est la probabilité qu'il fonctionne correctement quand il est sollicité

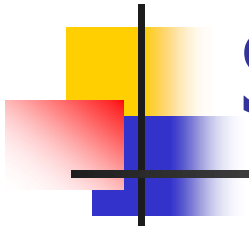
$$\text{Disponibilité} = \frac{\text{Temps moyen de fonctionnement sans défaillance}}{\left(\text{Tps moyen de fonct. sans défaillance}\right) + \text{tps moyen de réparation}}$$

Le niveau de disponibilité attendu doit être spécifié

Utilisabilité

- 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)



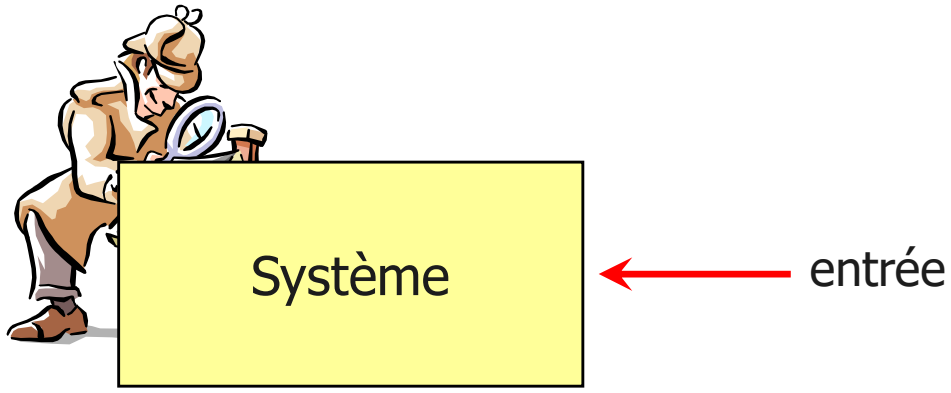


Sécurité

- 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 selon différentes dimensions
  - La confidentialité
  - L’authentification
  - L'intégrité
  - La disponibilité
  - La traçabilité & la non-répudiation

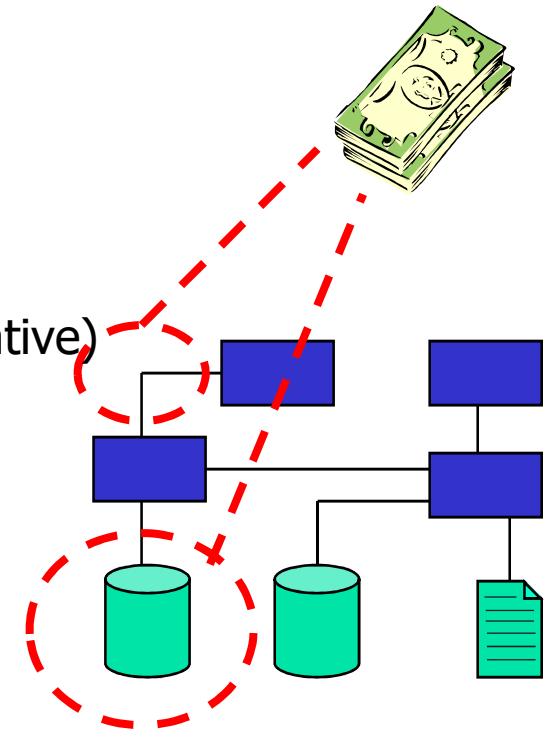
Testabilité

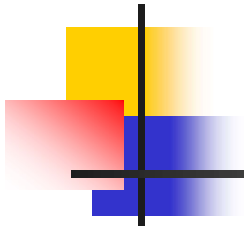
- La testabilité est la facilité avec laquelle on peut trouver les fautes d'un système
- Pour tester un système, on doit pouvoir
  - contrôler les entrées d'un composant et son état interne
  - observer les sorties



Maintenabilité (évolutions et réparations)

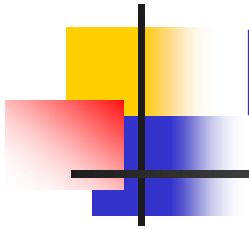
- Fait référence au coût des changements
- Un changement peut concerner tout aspect
  - les fonctionnalités
  - la plate-forme d'exécution (portabilité)
  - l'environnement d'interactions
  - les protocoles de communication
  - les propriétés non fonctionnelles, …
- 60% du coût d'un logiciel
  - Maintenance corrective (palliative, curative)
  - Maintenance adaptative
    - systématique, périodique, programmée
    - conditionnelle
    - prévisionnelle
  - Maintenance évolutive





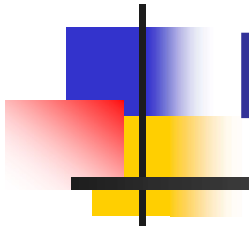
Réutilisabilité

- 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)



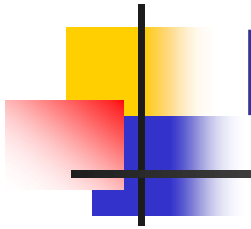
Note

- Ce sont des propriétés éminemment système
  - ne peuvent pas être considérées de façon uniquement locale
  - donc difficulté de les exprimer, de les traiter, de les satisfaire ensemble, …



Cours d'architecture logicielle
Tactiques de conception
Disponibilité

Lydie du Bousquet
Philippe Lalanda
Université Grenoble-Alpes

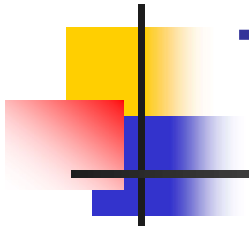


Disponibilité

- Un logiciel est disponible lorsqu'il est en état de fonctionner de façon correcte
  - Une faute (ou combinaison de fautes) est la cause potentielle de défaillance(s)
  - Une défaillance apparaît quand le système délivre une réponse qui n'est pas consistente avec la spécification
- La disponibilité d'un système est la probabilité qu'il fonctionne correctement quand il est sollicité

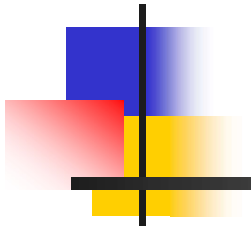
$$\text{Disponibilité} = \frac{\text{Temps moyen de fonctionnement sans défaillance}}{\left(\text{Tps moyen de fonct. sans défaillance}\right) + \text{tps moyen de réparation}}$$

Le niveau de disponibilité attendu doit être spécifié



Tactiques pour la disponibilité

- Pour 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

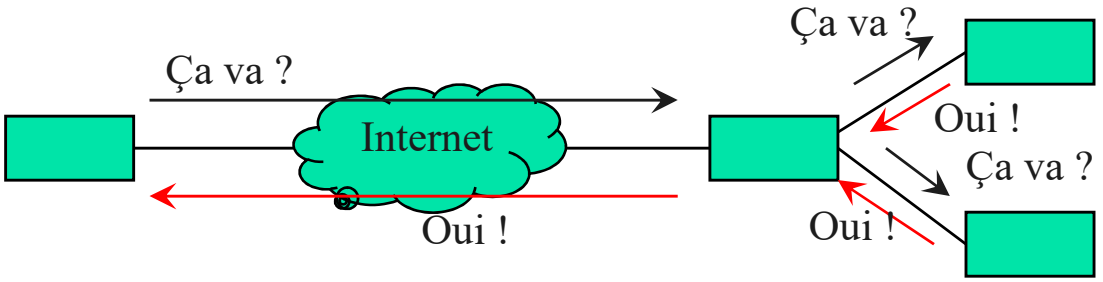


Disponibilité

Détecter les fautes

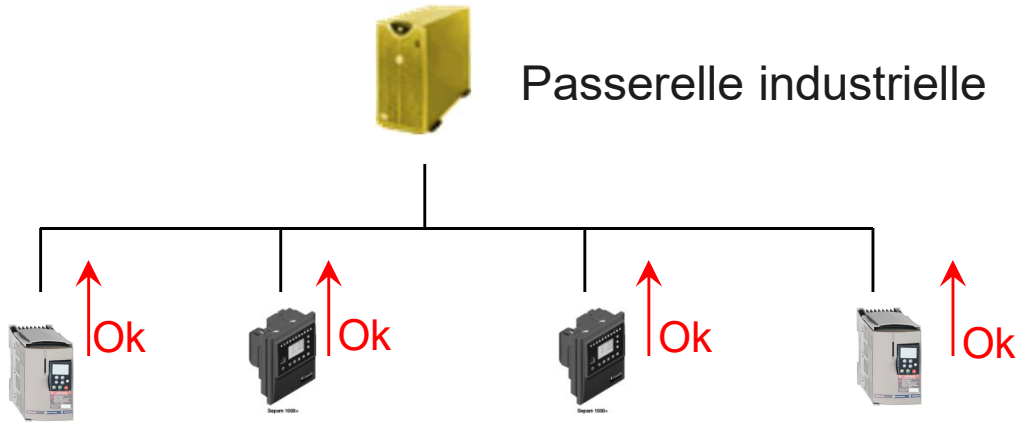
Tactique de détection des fautes – 1 « ping/echo »

- 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

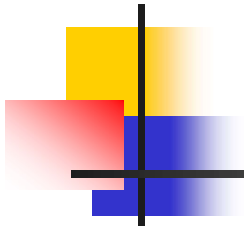


Tactique de détection des fautes – 2 « heartbeat » (dead man timer)

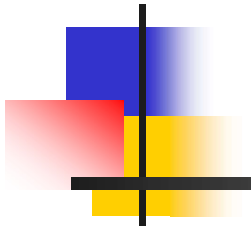
- Heartbeat : un composant émet périodiquement un battement écouté par d'autres composants
  - "jusque là tout va bien ..."
  - si on manque un battement, on fait l'hypothèse que le composant émetteur est en faute
  - un battement peut également contenir des données



Tactique de détection des fautes – 3 « Exception »



- Programmation d'exceptions
  - Répertorier les fautes possibles
  - Déclencher une exception lorsqu'une de ces fautes est rencontrée
  - Un mécanisme d’exception s’utilise au sein du même process

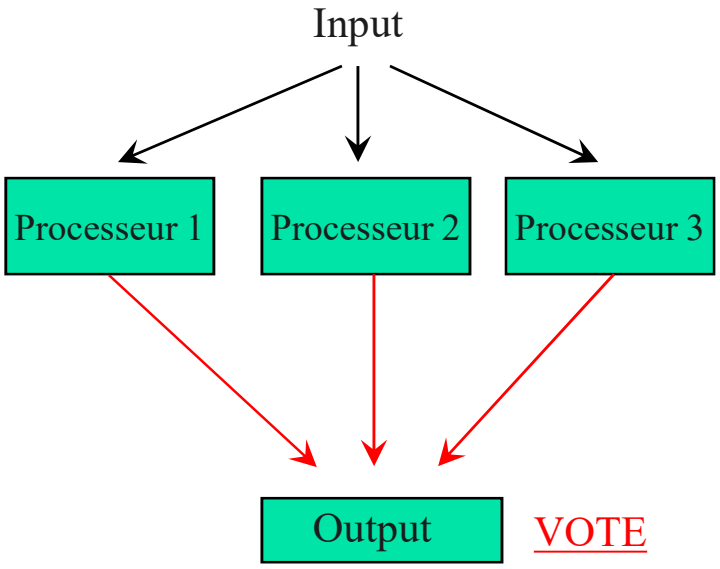


Disponibilité

Traiter les fautes

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"
- vote à la majorité ou privilégier des composants



Remarques

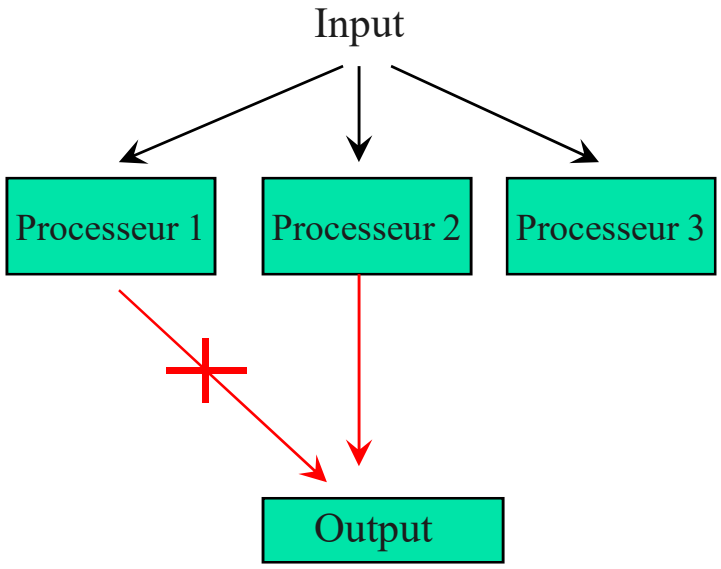
- détection des fautes processeurs si mêmes algos
- pas forcément les mêmes algos : pour les systèmes sensibles, algo différents sur des plate-formes différentes
  - très coûteux donc rare
  - ex : contrôleur de vol

Tactique de traitement des fautes – 2 « redondance active »

- Plusieurs processeurs redondants réagissent aux mêmes inputs en parallèle (tous les processeurs ont le même état)
- On utilise une seule réponse, définie statiquement (svt la 1ère)
- quand une faute survient, on change de composant

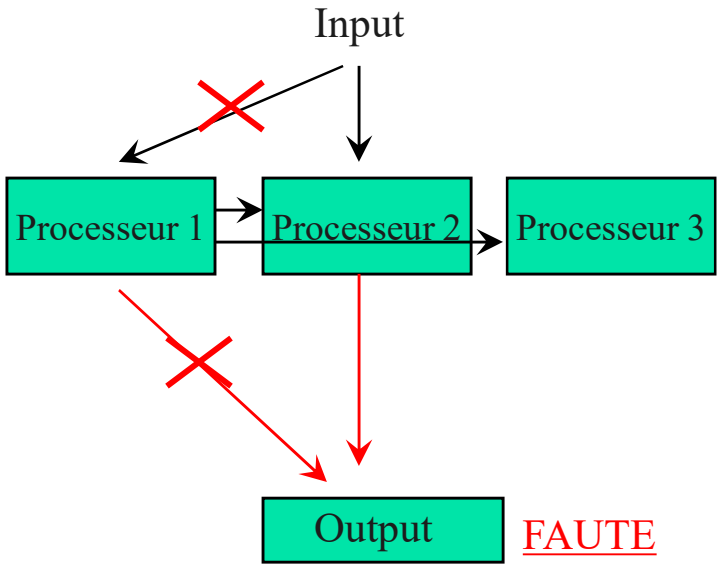
Remarques

- changement de processeur très rapide
- elle ne résout pas le problème de la détection d'une faute
- adaptée architectures client/serveur
- Pour des systèmes distribués, on applique la redondance à la communication (chemins parallèles)



Tactique de traitement des fautes – 3 « redondance passive »

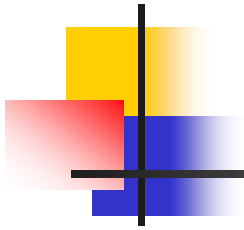
- un seul composant (primaire) réagit aux inputs et informent les composants redondants du nouvel état courant
- quand une faute survient,
  - on s'assure que le back-up est suffisant
  - on change de composant



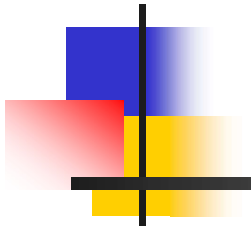
Remarques

- 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 »



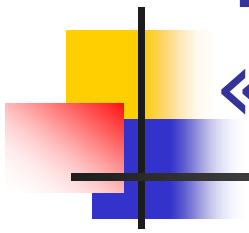
- Une plate-forme complète est configurée pour remplacer plusieurs composants en faute
- Quand une faute survient, il faut démarrer la plate-forme de secours et initialiser l'état
- Le système primaire doit sauvegarder ses données et ses états régulièrement sur une base persistante
- Remarque
  - le changement peut durer de quelques minutes à plusieurs heures selon le type de plate-forme de secours.



Disponibilité

Prévenir les fautes

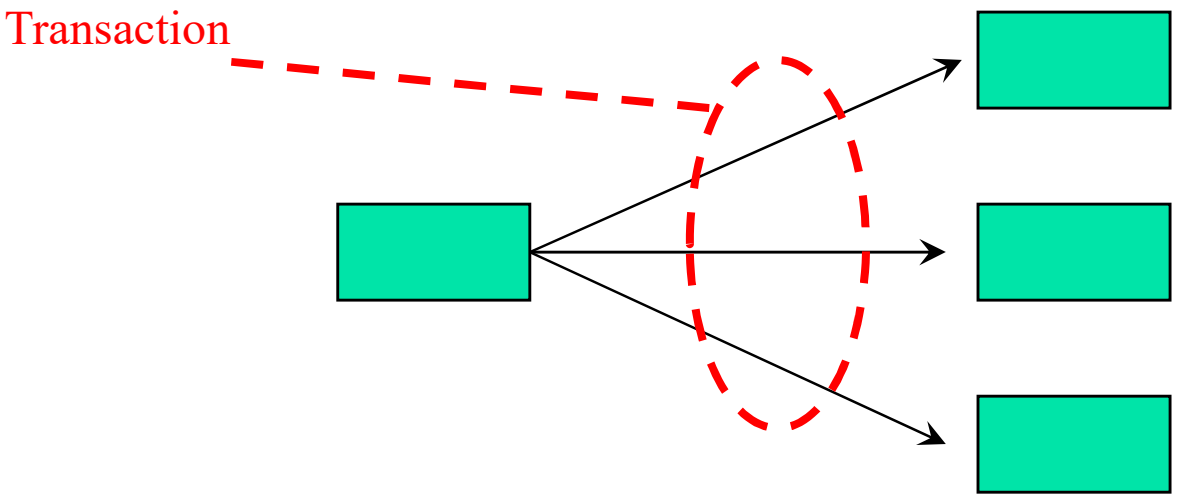
Tactique de prévention des fautes – 1 « retrait d'un composant »



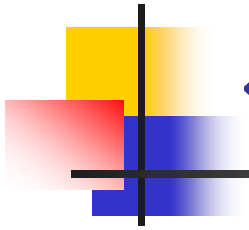
- On retire un composant pour effectuer des opérations de maintenance prédictive
- Le retrait
  - automatique => géré au niveau architectural
  - Manuel => le système doit le supporter
- Exemple
  - reboot périodique d'un composant pour éviter des fuites de mémoire

Tactique de prévention des fautes – 2 « Mise en place de transaction »

- Principes :
  - 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



Tactique de prévention des fautes – 3 « contrôleur de processus »

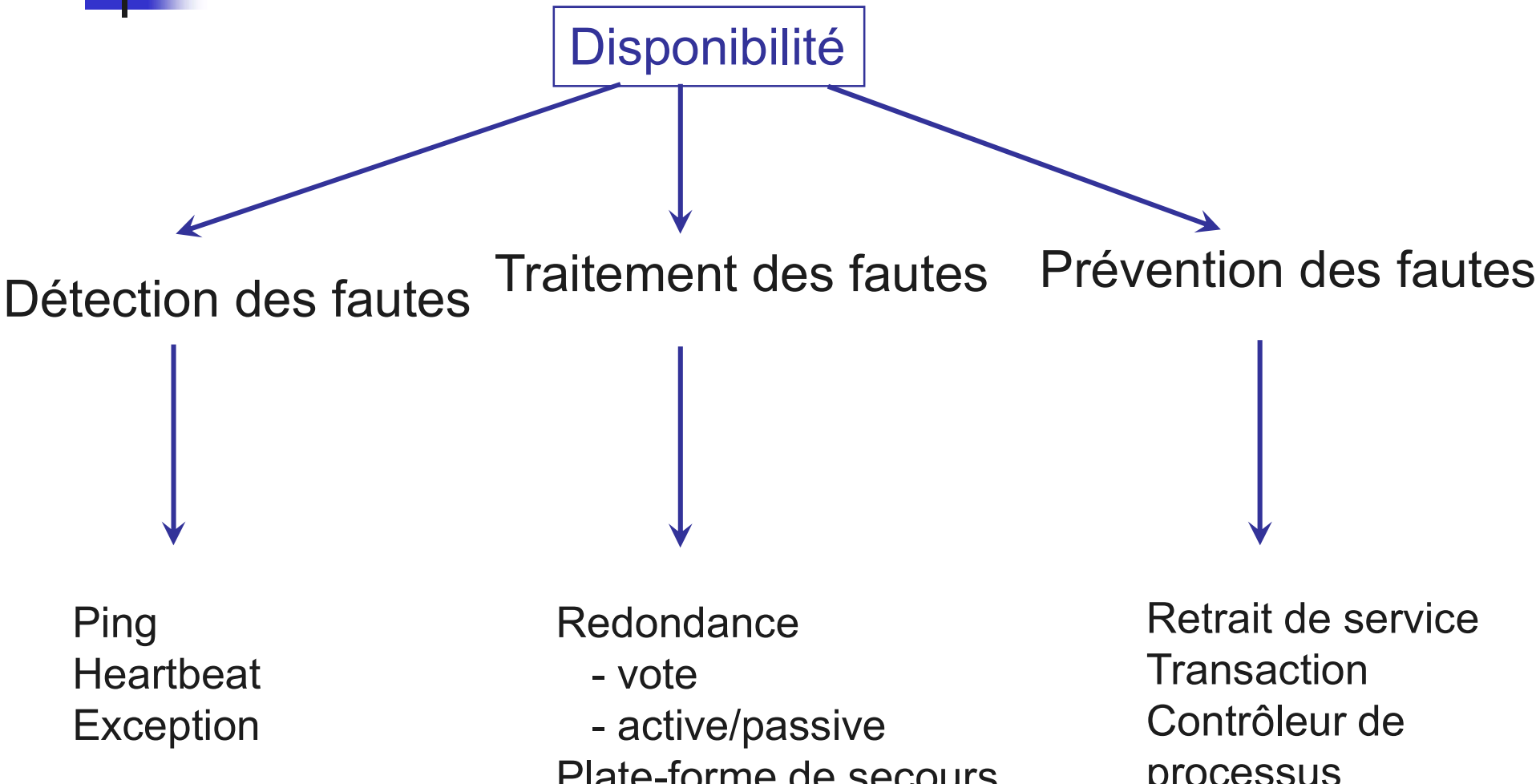


- Principes :
  - Lorsqu'une faute est détectée, un contrôleur élimine/efface le processus défaillant
  - Le contrôleur crée une nouvelle instance du processus
- Remarque
  - Le nouveau processus doit être initialisé dans l'état adéquat

Note : tactiques de niveau architectural

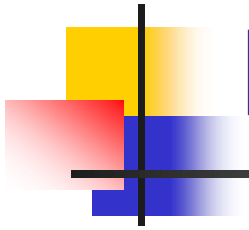
1. disponibilité

Résumé des tactiques



Cours d'architecture logicielle
Tactiques de conception
Performance

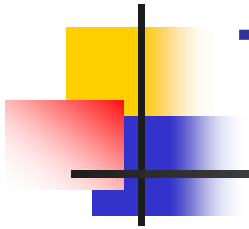
Lydie du Bousquet
Philippe Lalanda
Université Grenoble-Alpes



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)
  - 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, …)



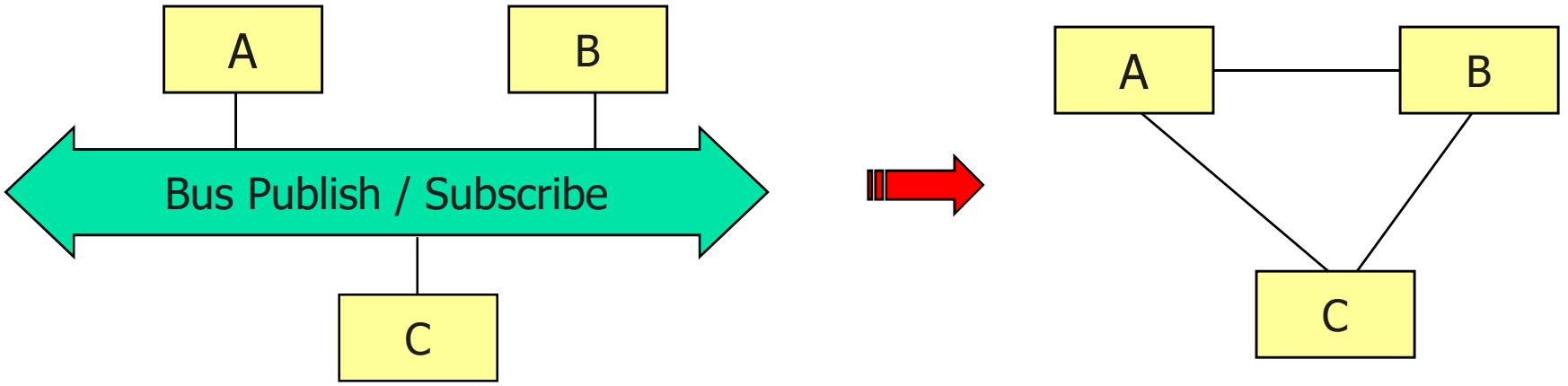
Tactiques pour la performance

- Objectif : générer une réponse à un événement avant une certaine échéance
- Utilisation de tactiques pour
  - optimiser les calculs
    - 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

3. performance

Tactique pour optimiser les calculs – 1 « limiter le nombre de composants »

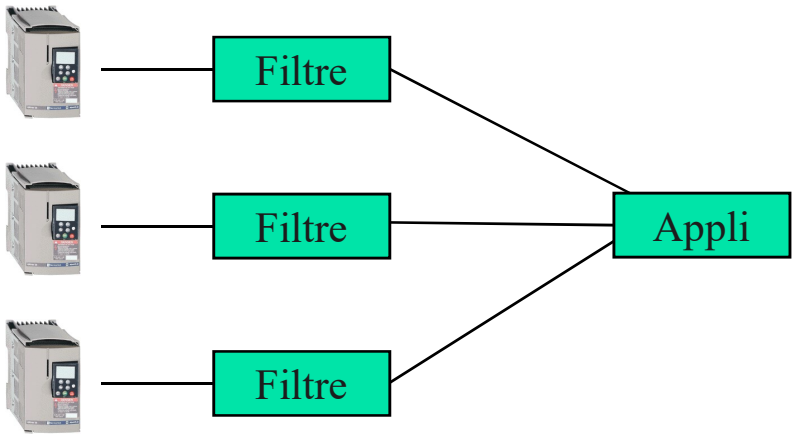
- Principes :
  - identifier les composants non fonctionnels et les éliminer
  - il s'agit, par exemple, d'éliminer les intermédiaires (broker, proxy, …)
- Remarque
  - exemple typique où un compromis entre performance et modifiabilité doit être fait



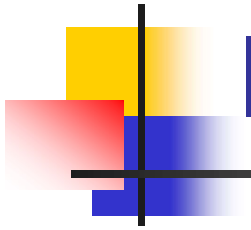
3. performance

Tactique pour optimiser les calculs – 2 « limiter les communications »

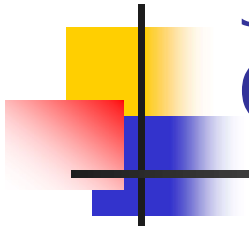
- Principes
  - Réduire la fréquence de communication entre composants
  - Exemple : réduire la fréquence d'échantillonnage des événements, notamment externes
- Remarque
  - certains systèmes ont des fréquences de traitement trop élevées à cause d'exigences excessives ou par commodités de programmation



Trois 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



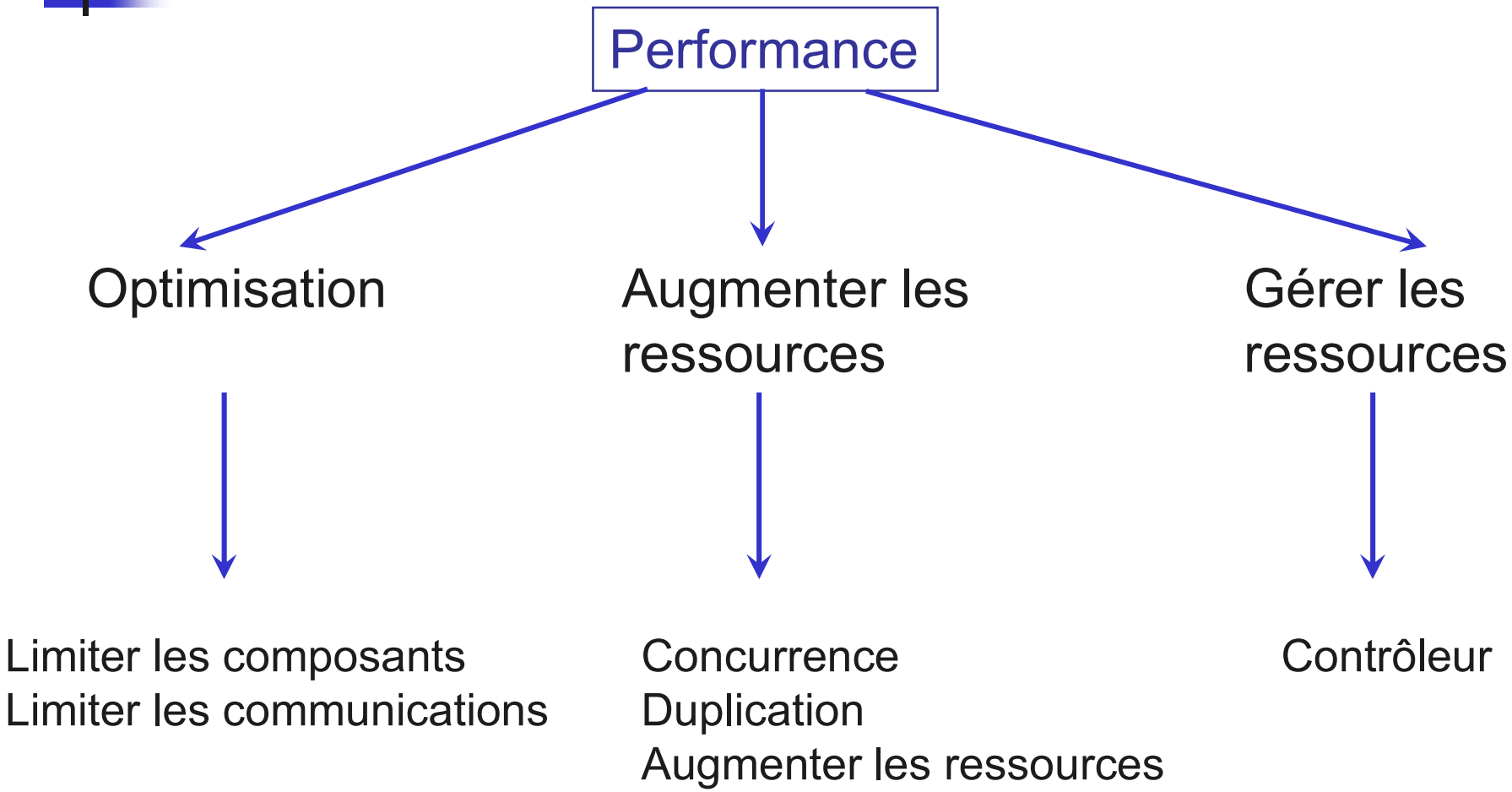
3. performance

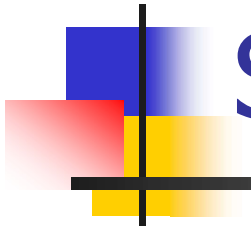
Gérer les ressources avec un « contrôleur »

- Principe :
  - 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

3. performance

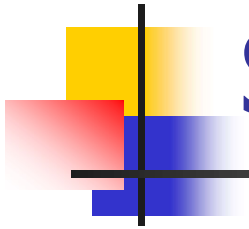
Résumé des tactiques





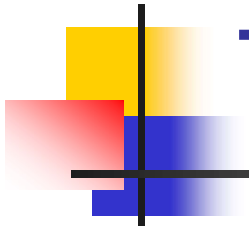
Cours d'architecture logicielle
Tactiques de conception
Sécurité

Lydie du Bousquet
Philippe Lalanda
Université Grenoble-Alpes



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 selon différentes dimensions
  - La confidentialité
  - L’authentification
  - L'intégrité
  - La disponibilité
  - La traçabilité & la non-répudiation



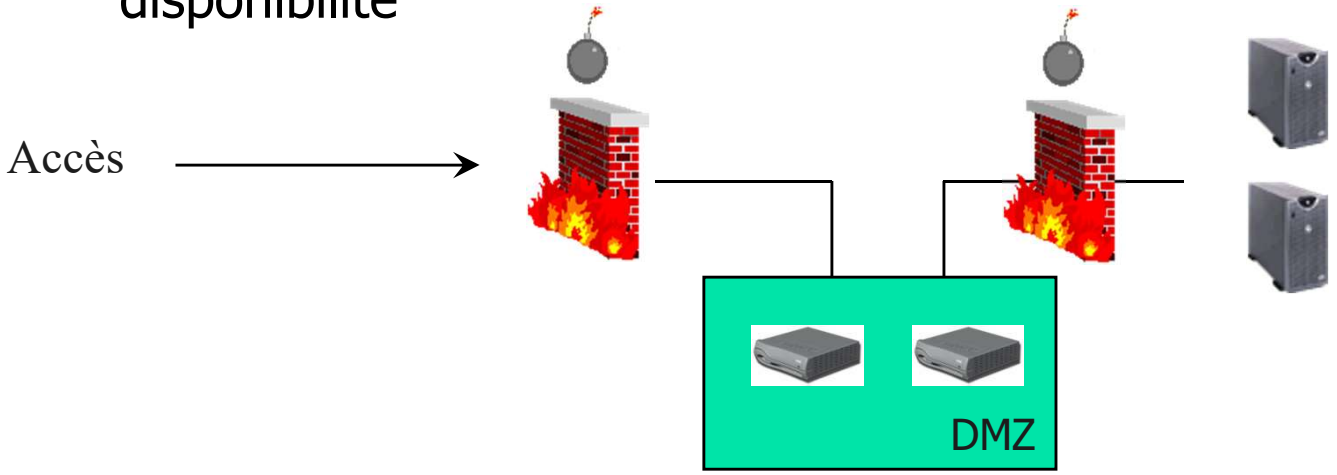
Tactiques pour la sécurité

- Pour garantir un fonctionnement nominal tout en résistant aux attaques
- Utilisation de tactiques pour
  - résister aux attaques
  - détecter les attaques
  - traiter les attaques

4. Sécurité

Tactique de résistance aux attaques – 1 « limiter l'exposition »

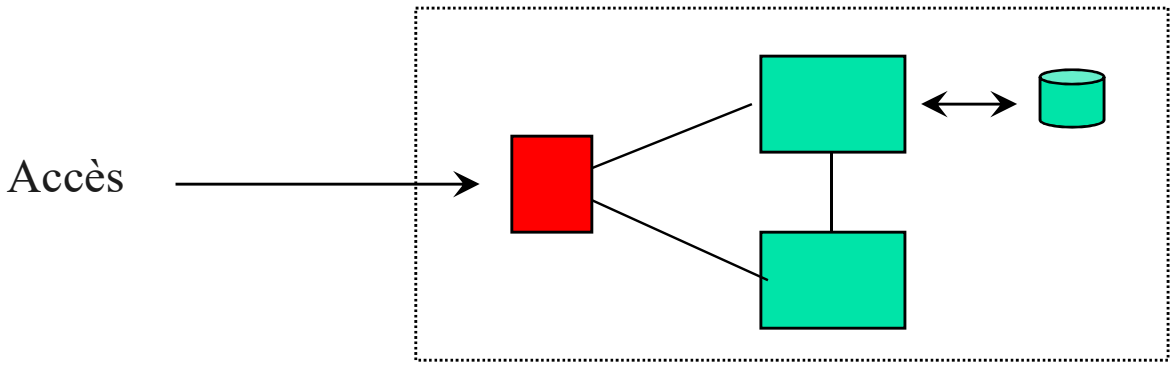
- Principe
  - répartition des services sur différents composants et différents "hosts"
  - utilisation d'un composant de type firewall
  - utilisation d'une DMZ (entre l'Internet et le firewall)
- Remarque
  - en opposition avec les tactiques de performance, voire de disponibilité



4. sécurité

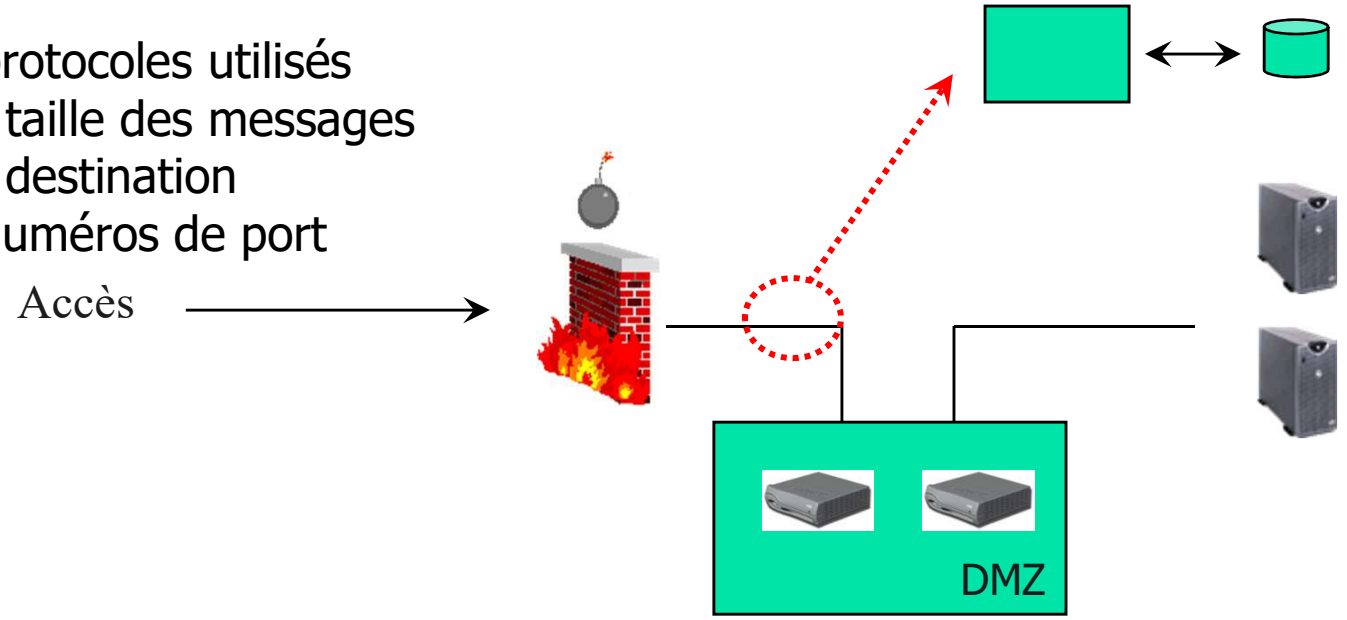
Tactique de résistance aux attaques – 2 « protéger les communications »

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

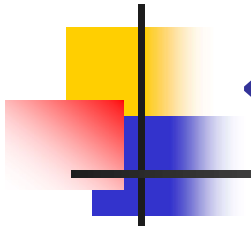


Tactique de détection des attaques « composant de détection »

- Principe : introduire un 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 »



- Principe : introduire un composant de Log pour
  - stocker des informations de communication
  - stocker 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