LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Service de diagnostic en ligne pour les applications à base de composants logiciels

Thi-Quynh BUI
Groupe CTSYS - Laboratoire LCIS

Encadrée par
Michel DANG
Oum-El-Kheir AKTOUF

LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Contexte

- **Programmation par composants**
  - Réutilisation des composants existants
  - Simplification de la conception
  - Optimisation de la maintenance
- **Applications distribuées**
- **Environnements dynamiques**

→ **Tolérance aux fautes dans les applications à base de composants logiciels**

LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Objectif

- **Développement d'un service de diagnostic pour les applications à base de composants logiciels**
  - Propriété non-fonctionnelle
  - Flexible
  - Générique

LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Plan de présentation

- Cadre de travail
- Approche de diagnostic
- Service de diagnostic DISCO
- Expérimentation et cas d'étude
- Conclusion et perspectives

LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Plan de présentation

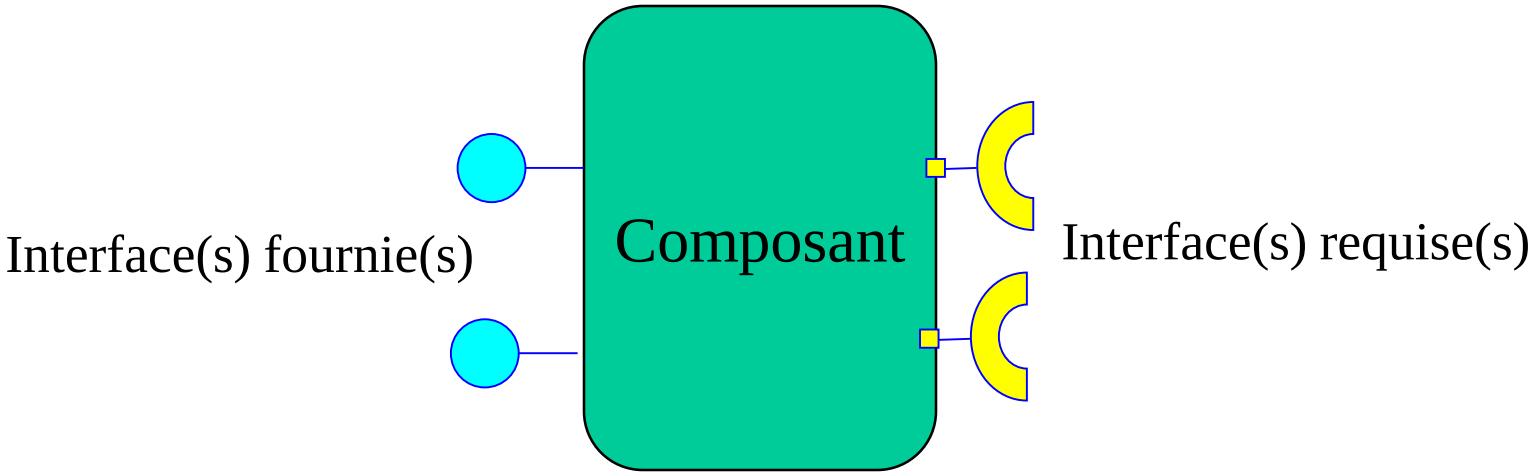
- **Cadre de travail**
  - Modèles de composants logiciels
  - Approches de tolérance aux fautes pour les applications à base de composants logiciels
  - Test des composants logiciels
- **Approche de diagnostic**
- **Service de diagnostic**
- **Expérimentation et cas d'étude**
- **Conclusion et perspectives**

Modèles de composants logiciels

- Technologie de composant logiciel
  - Modèle de composant
    - Définition de composant
    - Interfaces de composant
    - Composition de composant
  - Cadre du composant

Modèles de composants logiciels

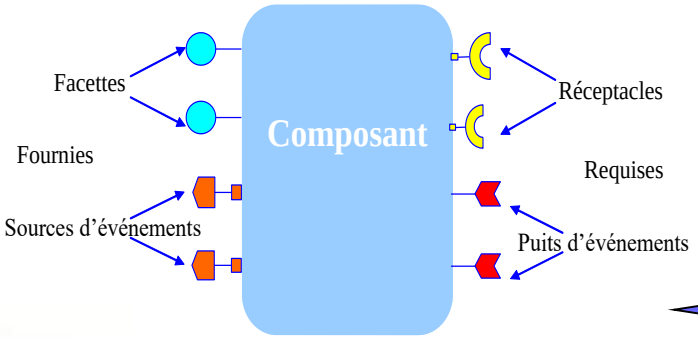
- **Modèle général de composant logiciel**

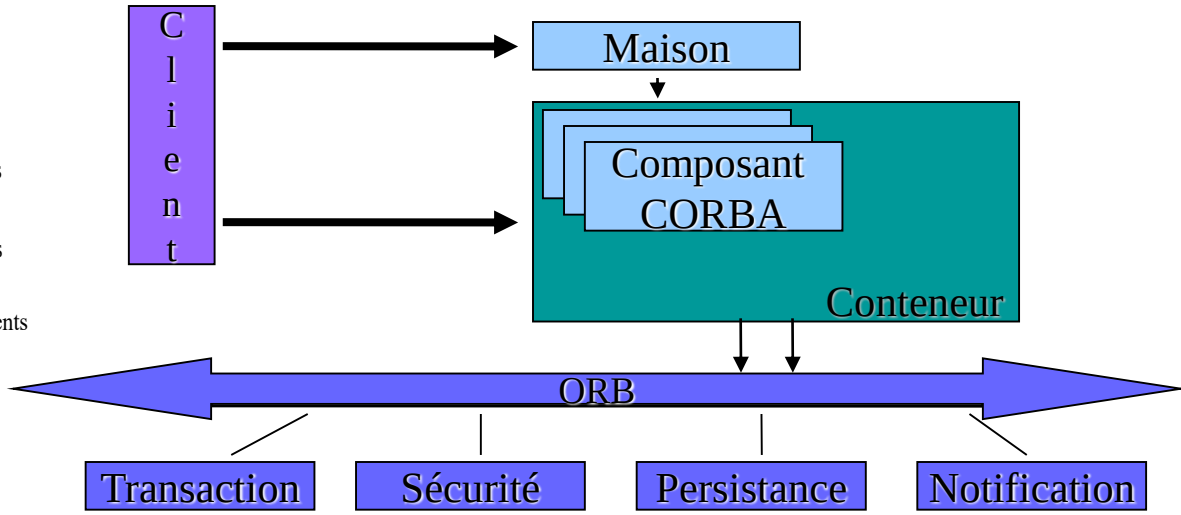


- **.NET, EJB, CCM, Fractal, OSGi**

Modèles de composants logiciels

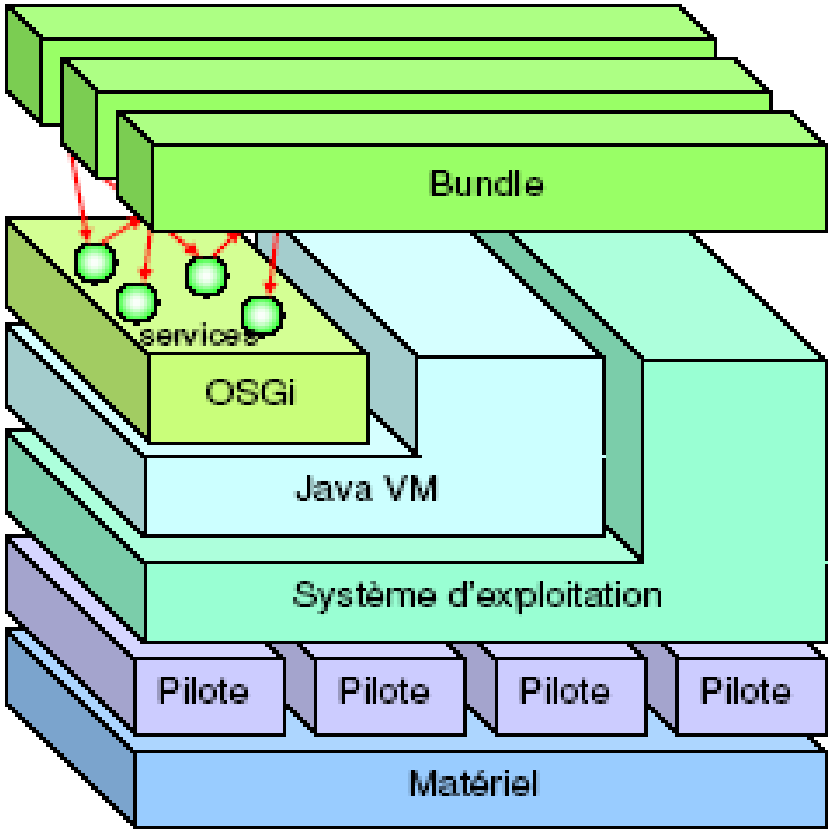
- **CCM**
  - Applications réparties
  - Implémentation de composants dans différents langages





Modèles de composants logiciels

- **OSGi**
  - Réseaux résidentiels
  - *Bundle* est une unité de déploiement
  - Un composant ou un service
  - Tenant compte des évolutions dynamiques des applications



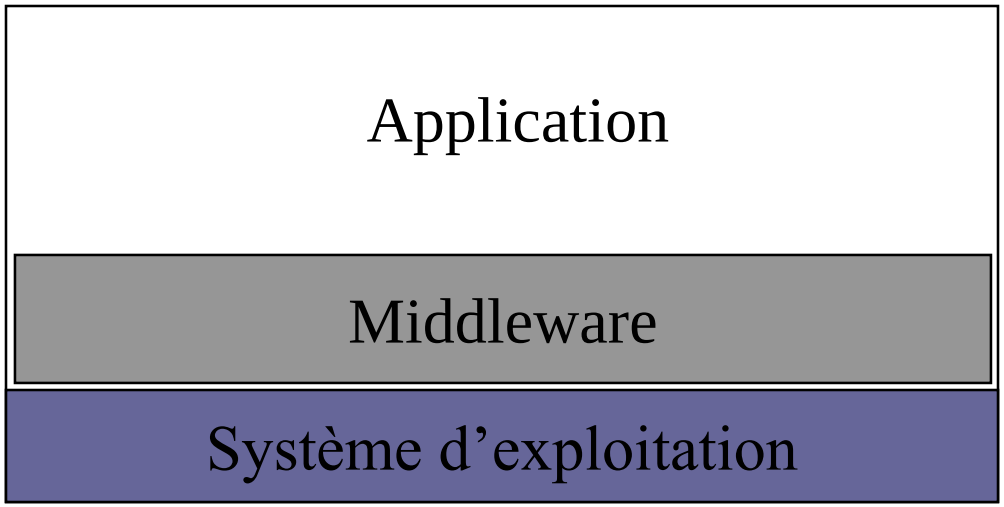
Modèles de composants logiciels

- **Les composants sont validés**
- **Composants défaillants durant l'exécution**
  - Interaction entre les composants
  - Environnement d'exécution
  - Validation incomplète ou insuffisante

→ Besoin de mécanismes de tolérance aux fautes

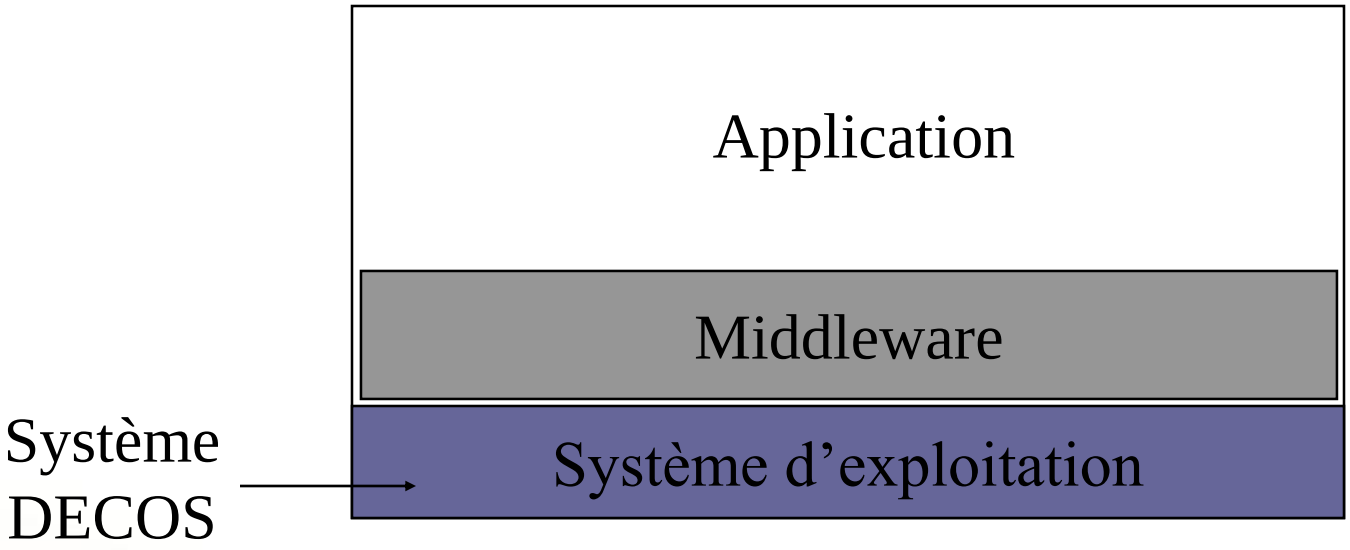
Tolérance aux fautes dans les applications à base de composants logiciels

- Différentes couches d'intégration de la tolérance aux fautes



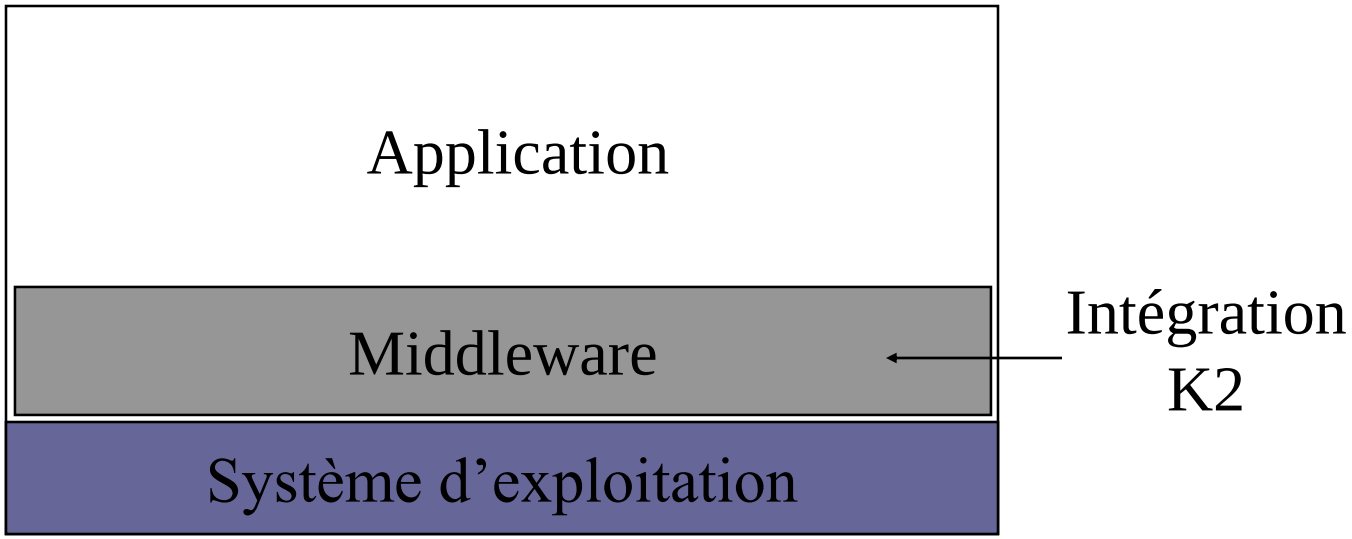
Tolérance aux fautes dans les applications à base de composants logiciels

- Différentes couches d'intégration de la tolérance aux fautes



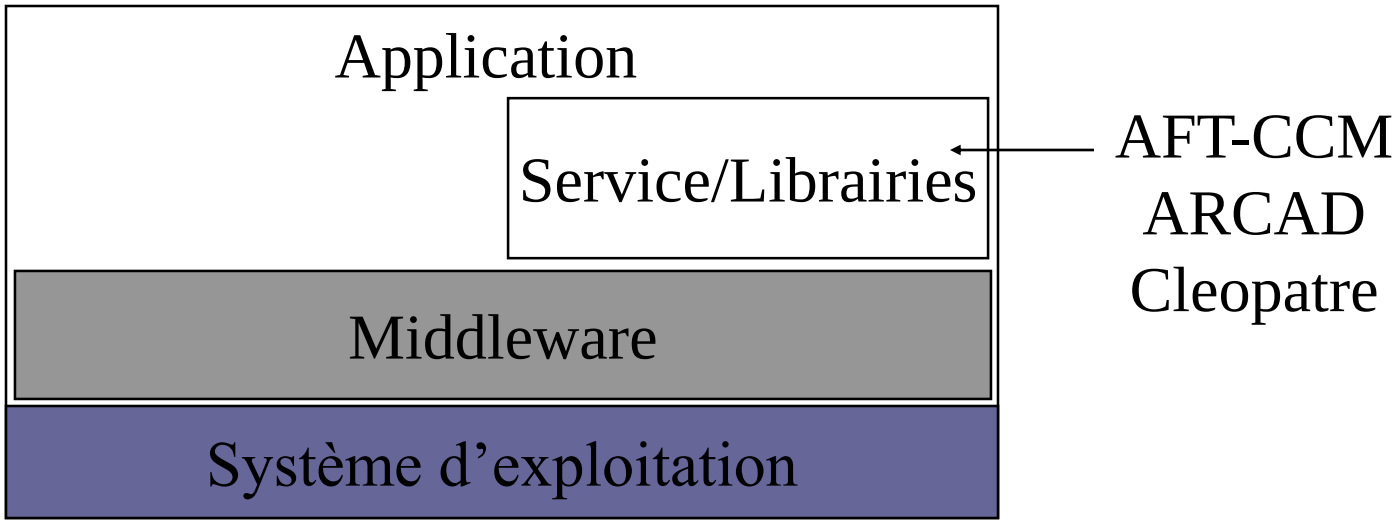
Tolérance aux fautes dans les applications à base de composants logiciels

- Différentes couches d'intégration de la tolérance aux fautes



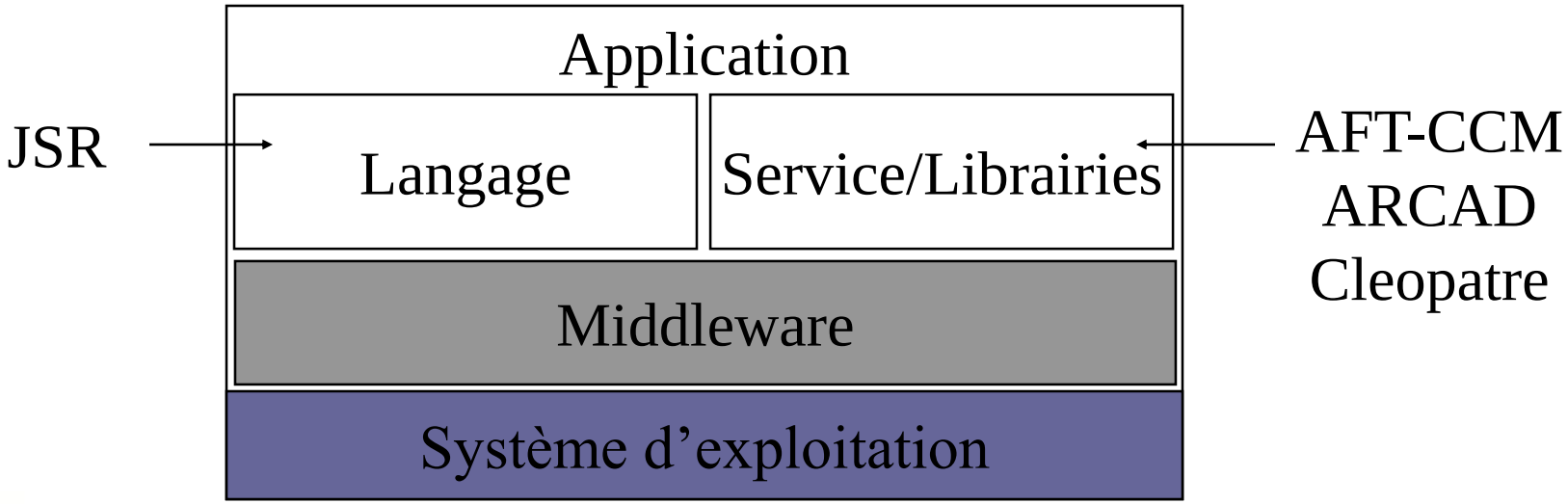
Tolérance aux fautes dans les applications à base de composants logiciels

- Différentes couches d'intégration de la tolérance aux fautes



Tolérance aux fautes dans les applications à base de composants logiciels

- Différentes couches d'intégration de la tolérance aux fautes



Tolérance aux fautes dans les applications à base de composants logiciels

- Comparaison des différentes approches

Niveau	Critère / Approche	Transparence	Séparation	Visibilité	Réutilisabilité	Portabilité
Système	Système	+	+	+	-	-
Middleware	Intégration	-	-	+	-	-
Application	Librairies	-	-	+	-	+
	Service	+	+	+	+	+
	Langage	-	+	+	+	+

Tolérance aux fautes dans les applications à base de composants logiciels

- Tolérance aux fautes dans les applications à base de composants logiciels : nouveau domaine de recherche
- Principales approches : basées sur la réplication de composants pour le masquage de fautes
- Utilisation de la réplication
  - Coûteuse : réplication des ressources

→ **Une solution mieux adaptée : l'utilisation des techniques de diagnostic**

Test des composants logiciels

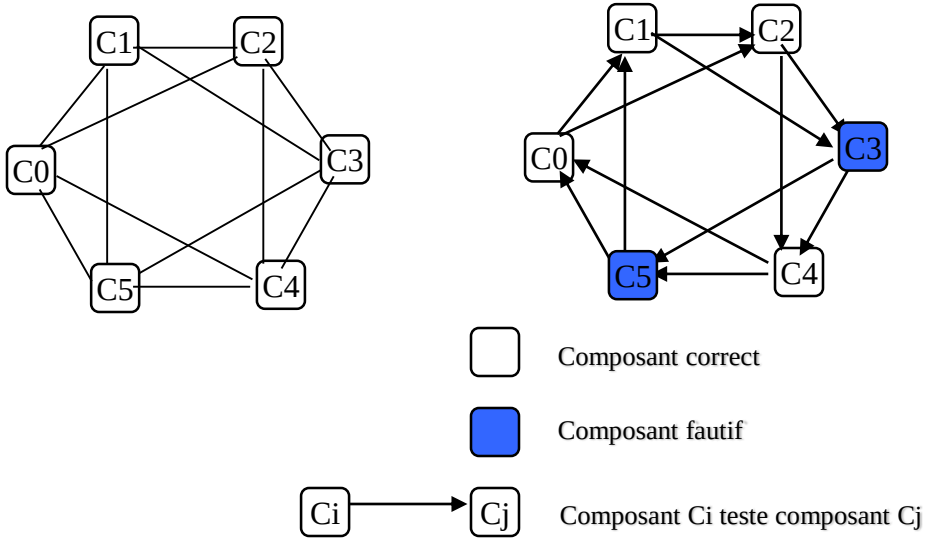
- Approches de *test intégré* (*Built-in test*) ou *autotest* (*Self-test*)
- Eléments de code non fonctionnel
- Interface de test
- Projets : Component +, STECC, *etc.*
- Principal objectif est de tester un composant dans son nouvel environnement d'exécution

Plan de présentation

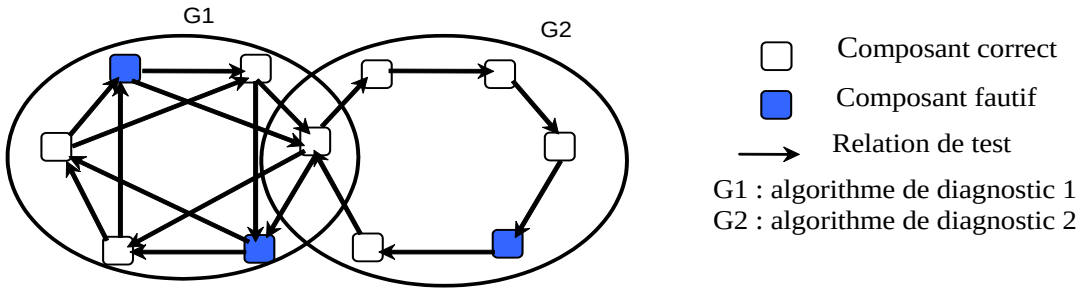
- Cadre de travail
- **Approche de diagnostic**
  - Définitions de base
  - Méthode de diagnostic
  - Tests inter-composants en ligne
- **Service de diagnostic DISCO**
- **Expérimentation et cas d'étude**
- **Conclusion et perspectives**

Définitions de base

- Graphe système
- Graphe de test



- Groupe de diagnostic
- Etat d'un groupe



Méthodes de diagnostic

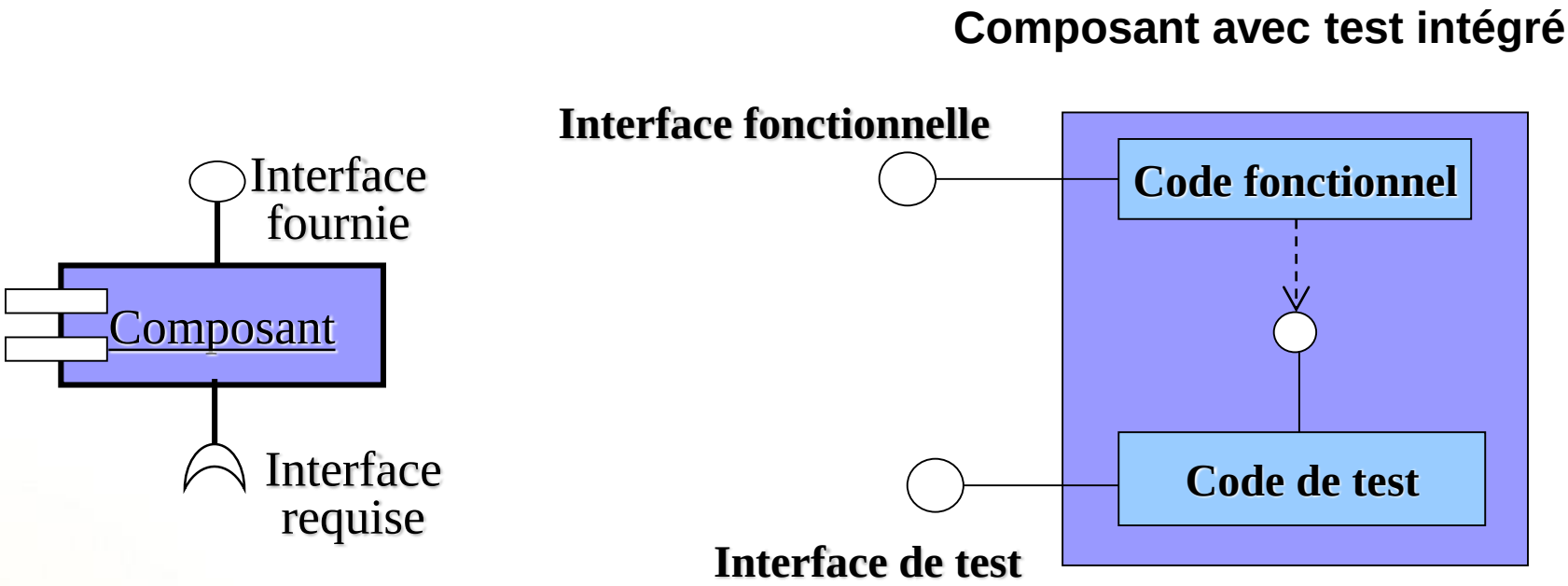
- **Méthodes de diagnostic**
  - Centralisé
    - Composant central fiable
    - Contrôle l'exécution des tests inter-composants
    - Détermine l'état du système à partir des résultats de tests
  - Distribué
    - Tout composant peut déterminer l'état du système
- **Stratégies de diagnostic**
  - Diagnostic statique
  - Diagnostic dynamique

Tests inter-composants en ligne

- **Implémentation des tests**
  - Basée sur le principe de l'approche de test intégré du projet Component+
  - Test fonctionnel
  - Modèle d'états du composant
- **Contexte en ligne**
  - Tenir compte des contraintes de concurrence du test et des fonctionnalités des composants
  - Mettre en place des mécanismes qui perturbent le moins possible le fonctionnement des composants

Tests inter-composants en ligne

- Composant avec test intégré (Component+)



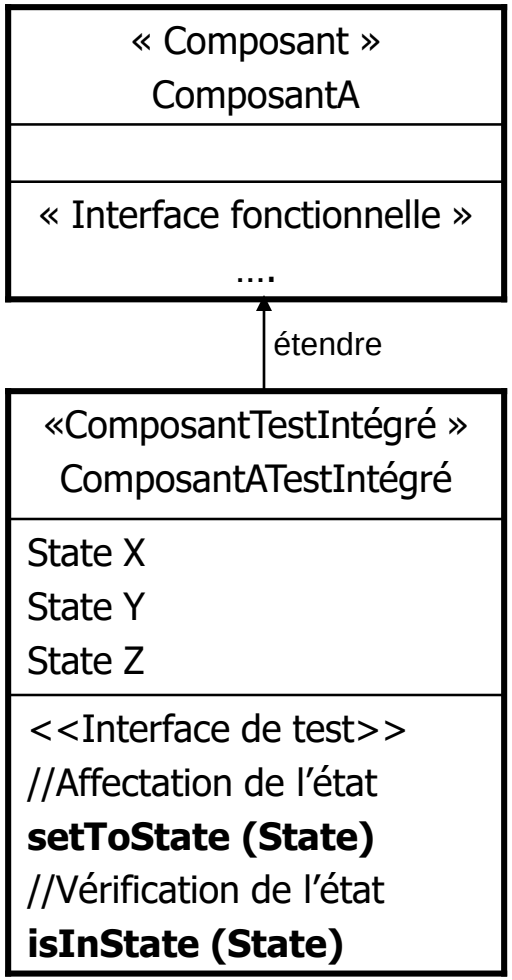
Tests inter-composants en ligne

Exécution d'un test

- État initial
- Sorties voulues
- État final voulu

Les opérations spéciales dans l'interface de test intégré

- isInState(state)
- setToState(state)

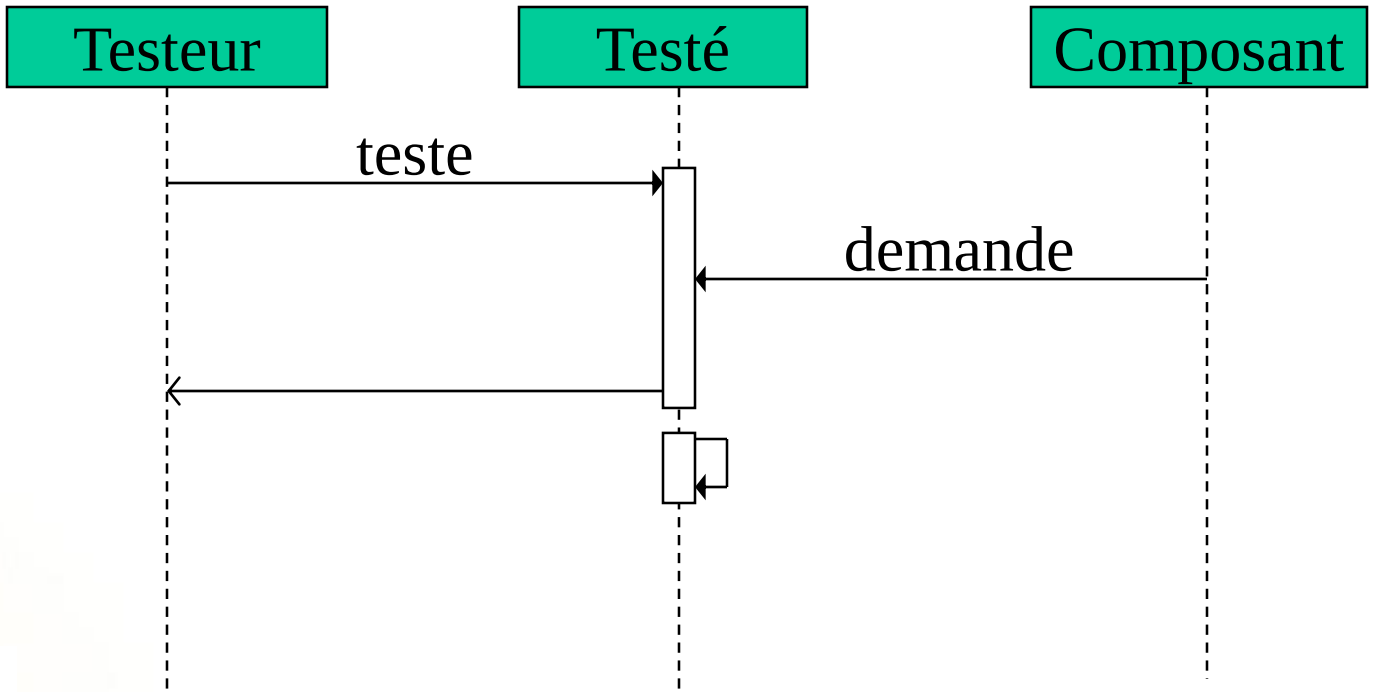


Tests inter-composants en ligne

- **Contexte en ligne**
  - Interférences entre les tests et les fonctionnalités normales du composant
    - Pendant le processus de test, un composant reçoit une demande de service fonctionnel d'autres composants
    - Le test manipule et change des données fonctionnelles du composant

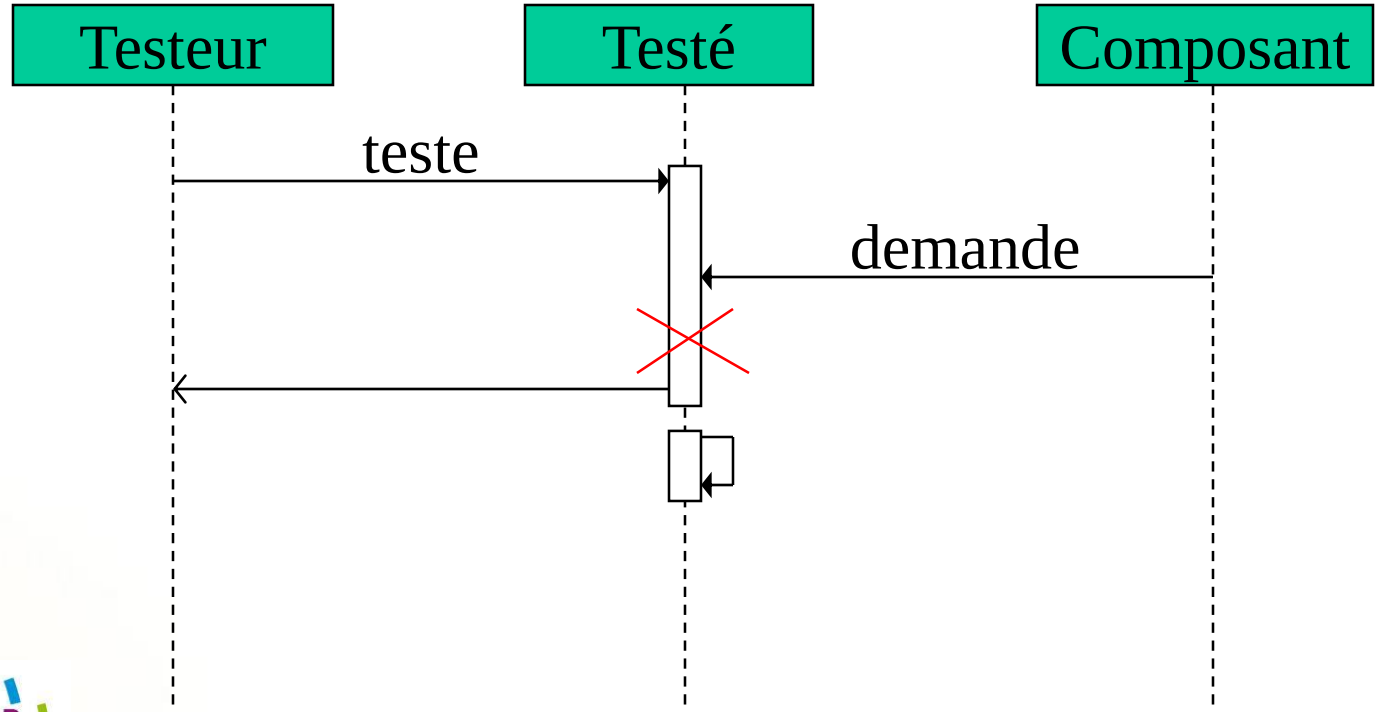
Tests inter-composants en ligne

- **Blocage du composant :** le composant est bloqué pendant le processus de test



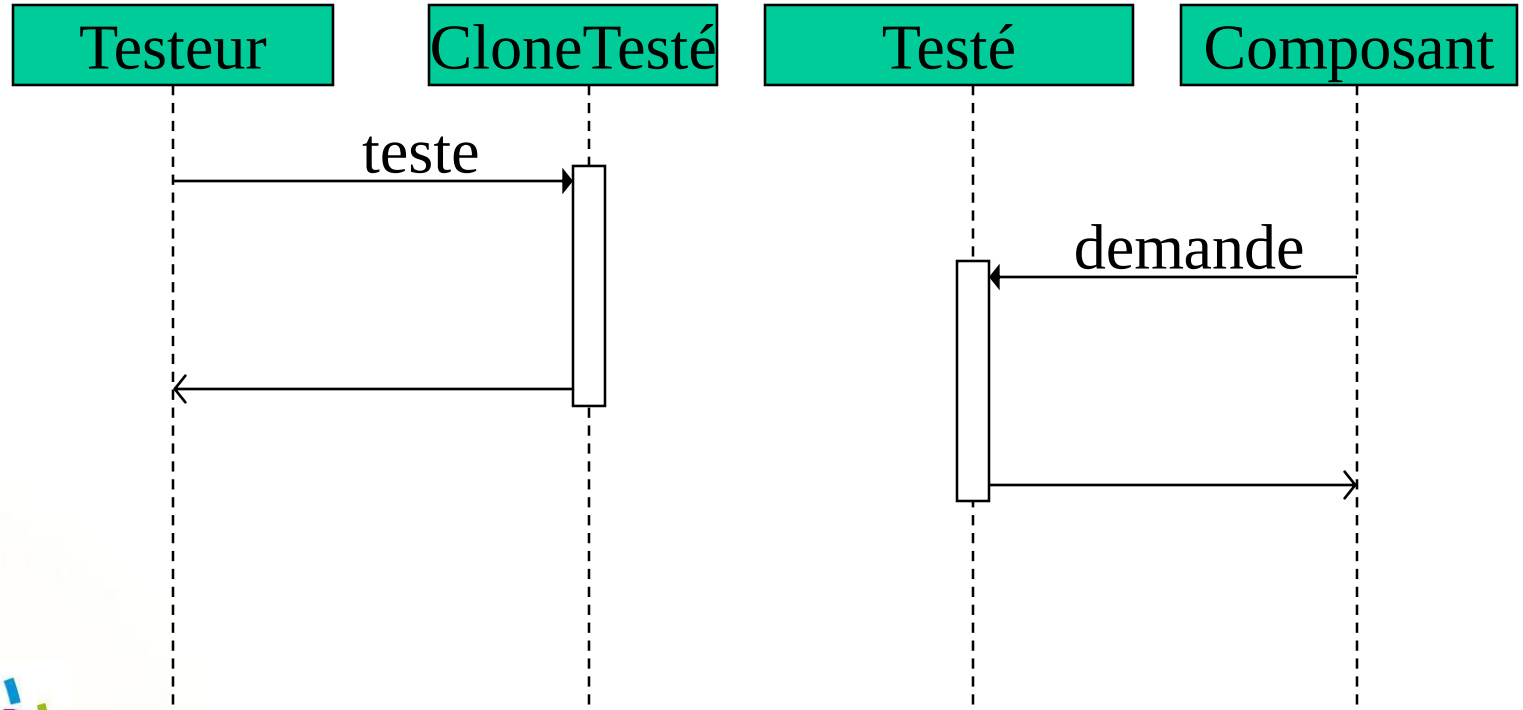
Tests inter-composants en ligne

- **Abandon du processus de test** : le composant abandonne le processus de test



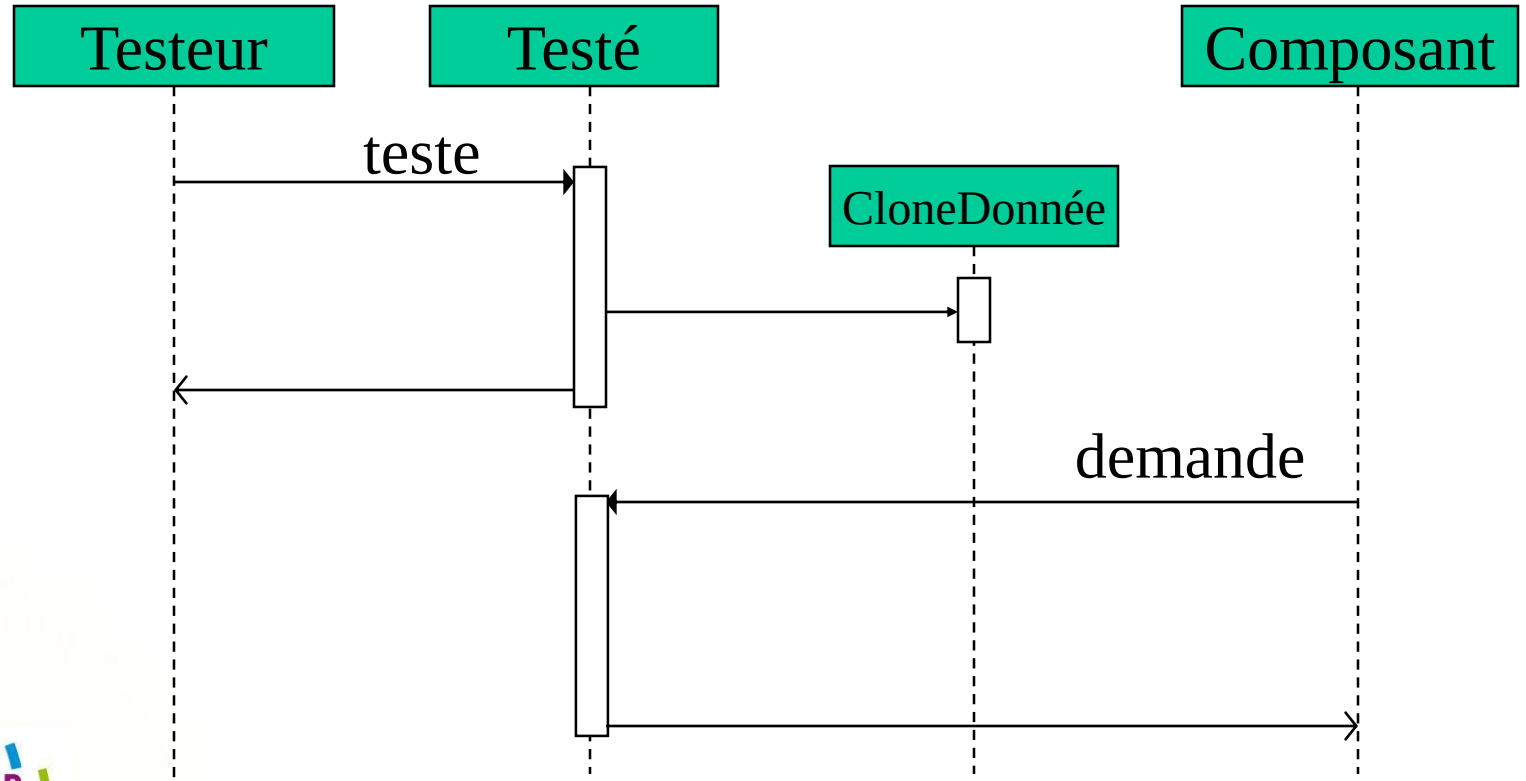
Tests inter-composants en ligne

- **Clonage du composant** : le composant est cloné par l'infrastructure d'exécution avant le démarrage du test



Tests inter-composants en ligne

- **Session de test** : l'interface de test du composant fournit des méthodes qui permettent des sessions de test



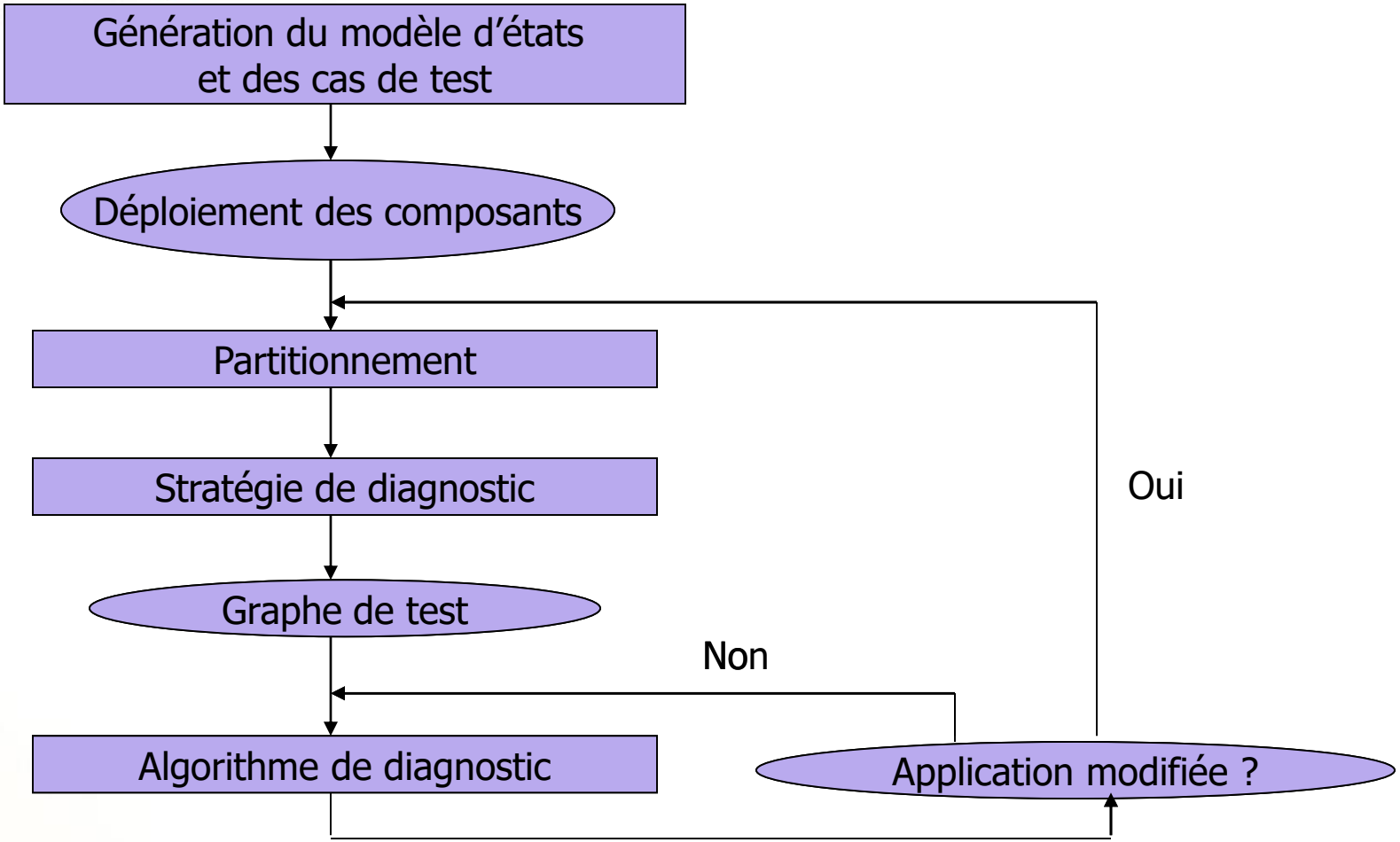
Tests inter-composants en ligne

- **Le choix d'une solution plutôt qu'une autre dépend du contexte considéré :**
  - Applications temps réel ou critiques, les demandes de test sont abandonnées au profit des appels fonctionnels
  - Applications non critiques, il est envisageable d'interrompre momentanément un composant en vue de le tester :
    - Les fonctions du composant sous test sont de type lecture seule, l'opération est bloquée jusqu'à la fin du test
    - Les opérations sous test modifient des données, on crée des sessions de test

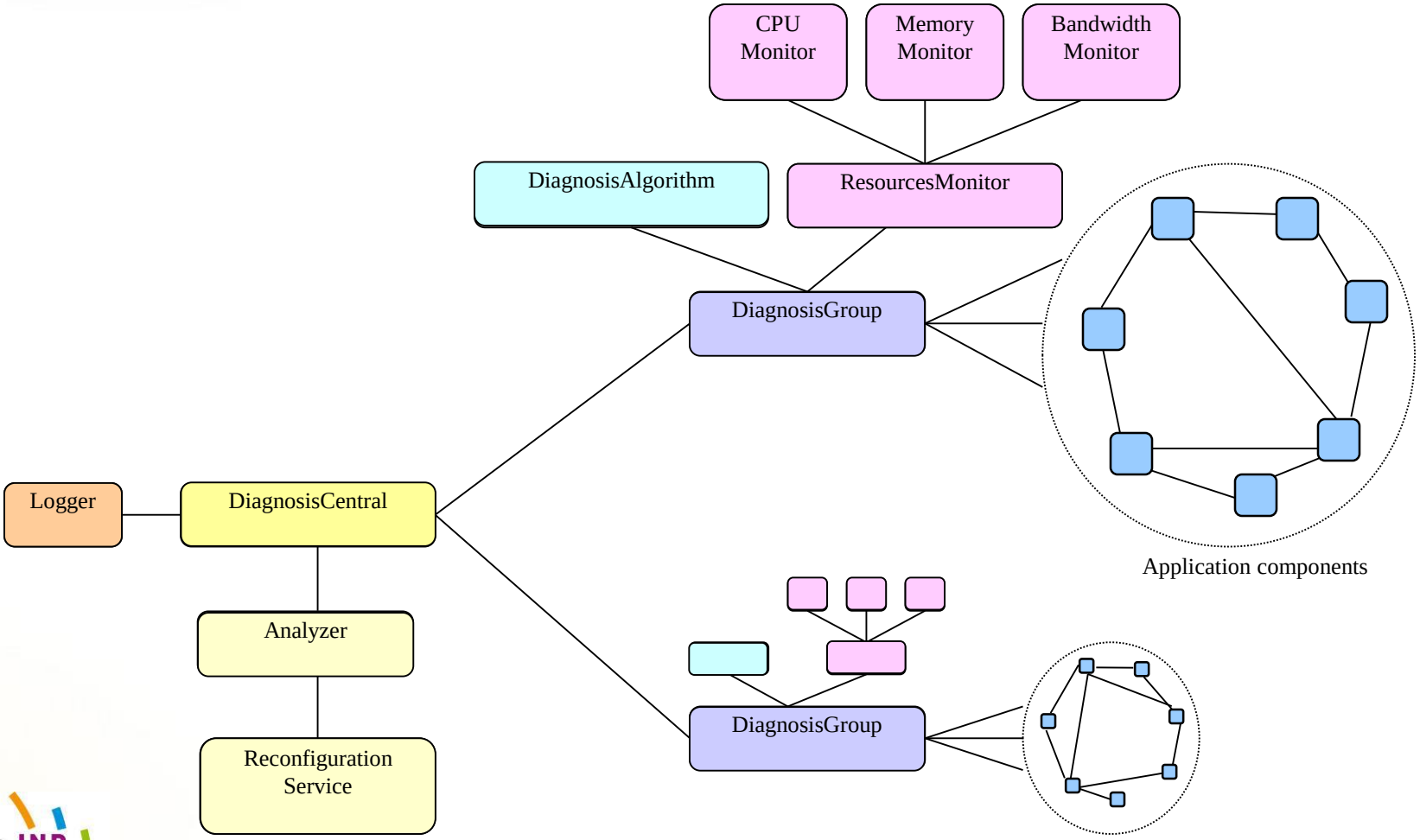
Plan de présentation

- Contexte
- Approche de diagnostic
- **Service de diagnostic DISCO**
  - Procédure de diagnostic
  - Architecture du service DISCO
  - Fonctionnement du service DISCO
- **Expérimentation et cas d'étude**
- **Conclusion et perspectives**

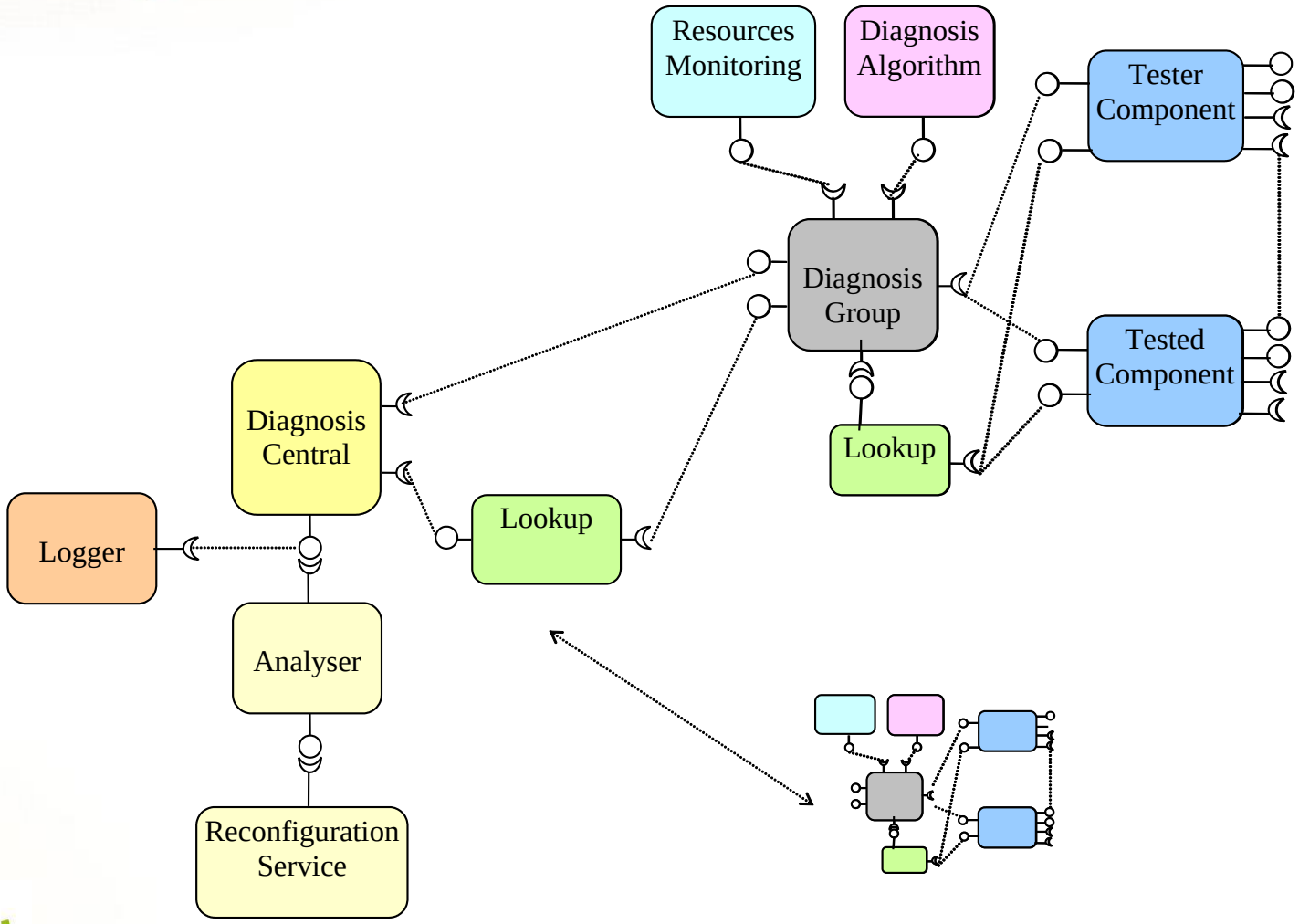
Procédure de diagnostic



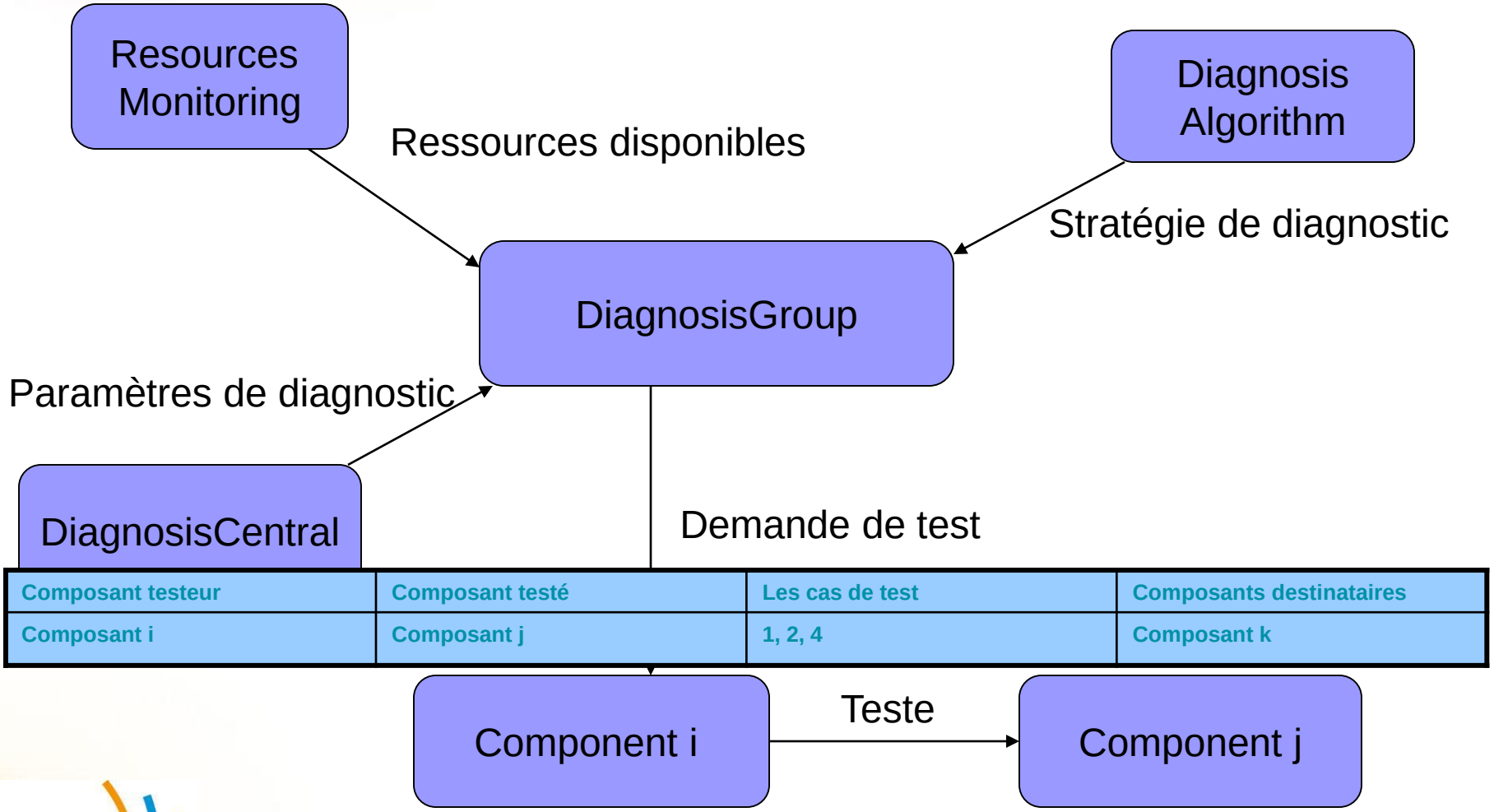
Architecture du service DISCO



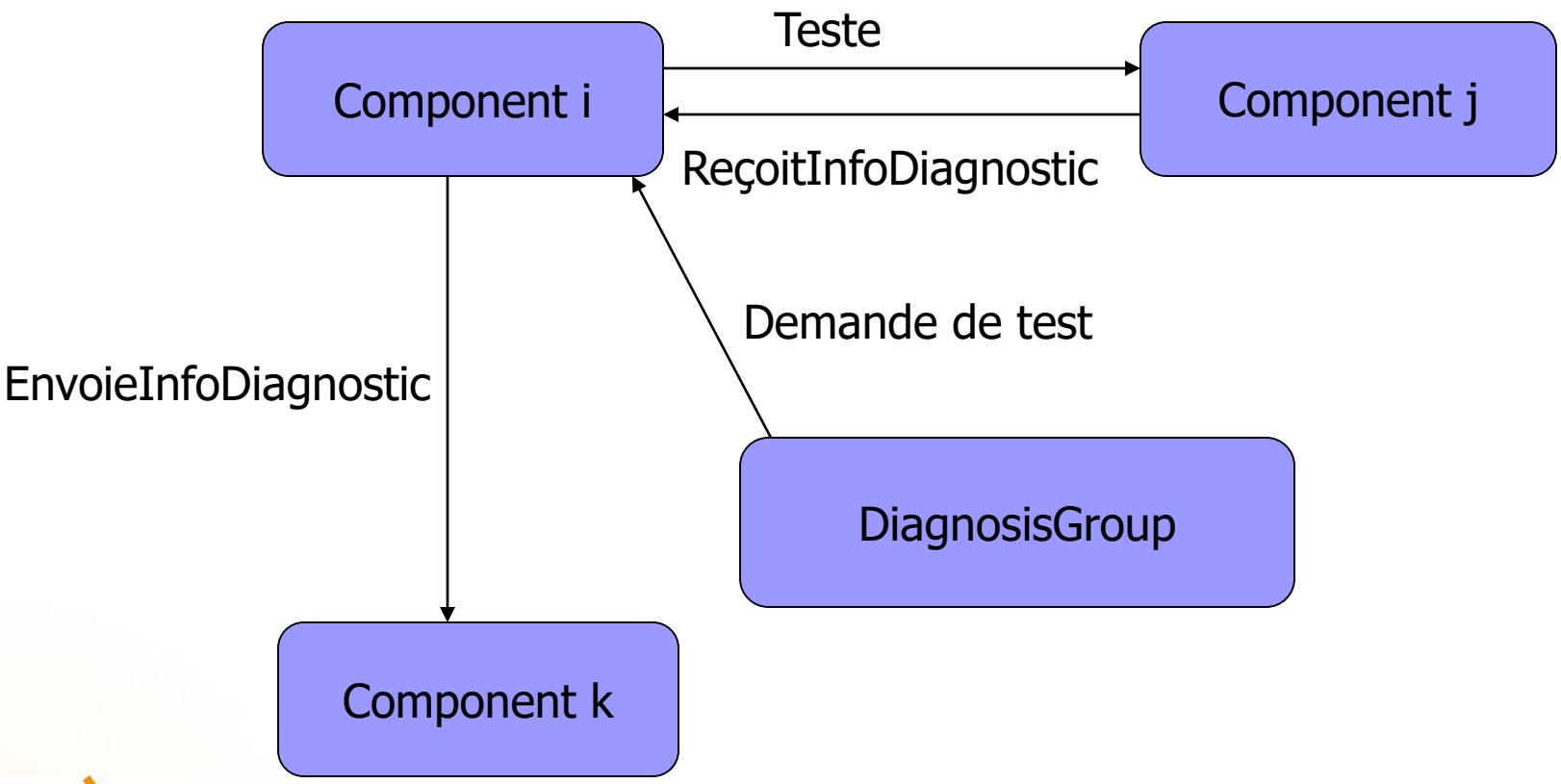
Architecture orientée composants



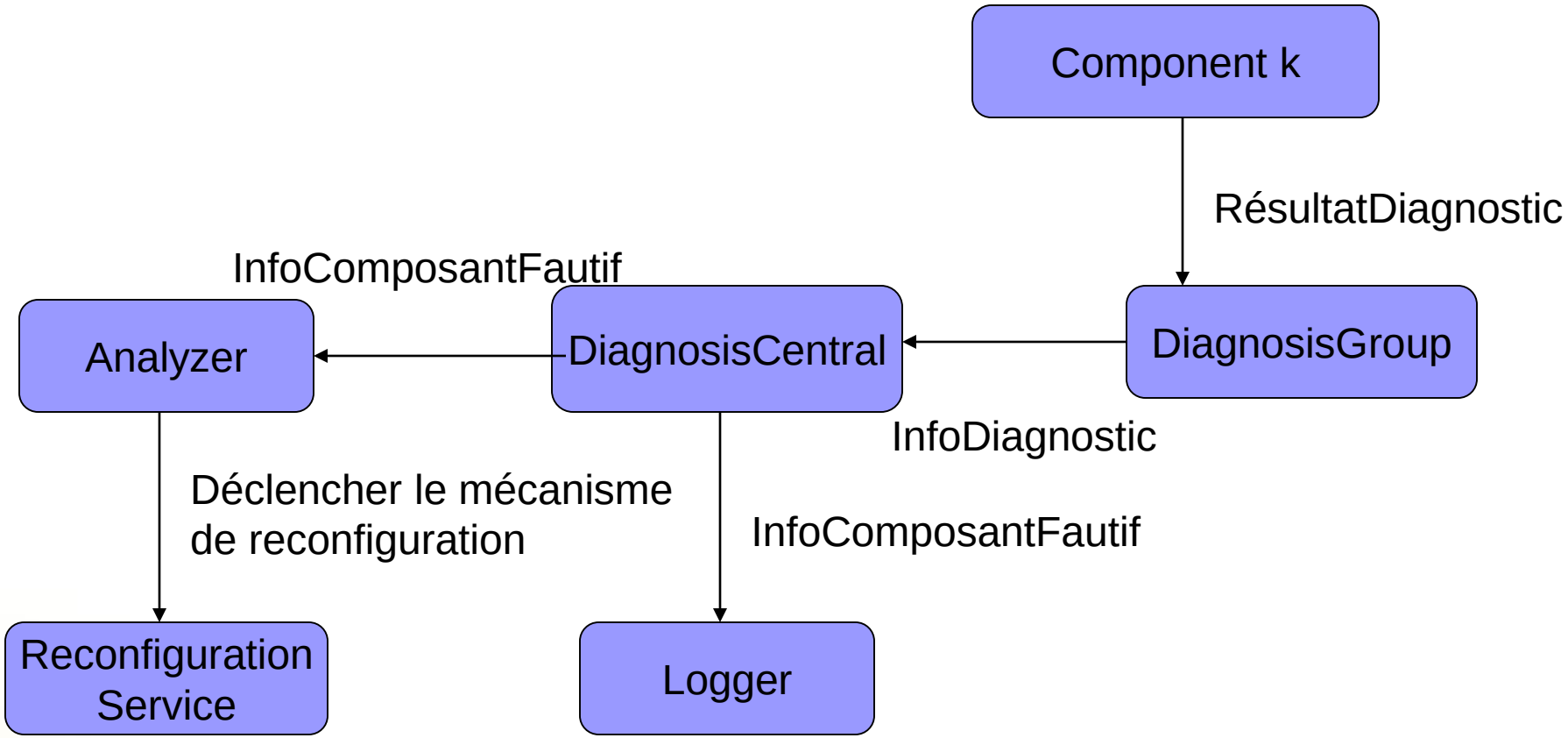
Fonctionnement du service DISCO



Fonctionnement du service DISCO



Fonctionnement du service DISCO



Plan de présentation

- Contexte
- Approche de diagnostic
- Service de diagnostic
- **Expérimentation et cas d'étude**
  - Cas d'étude
  - Démarche expérimentale
  - Résultats obtenus
- **Conclusion et perspectives**

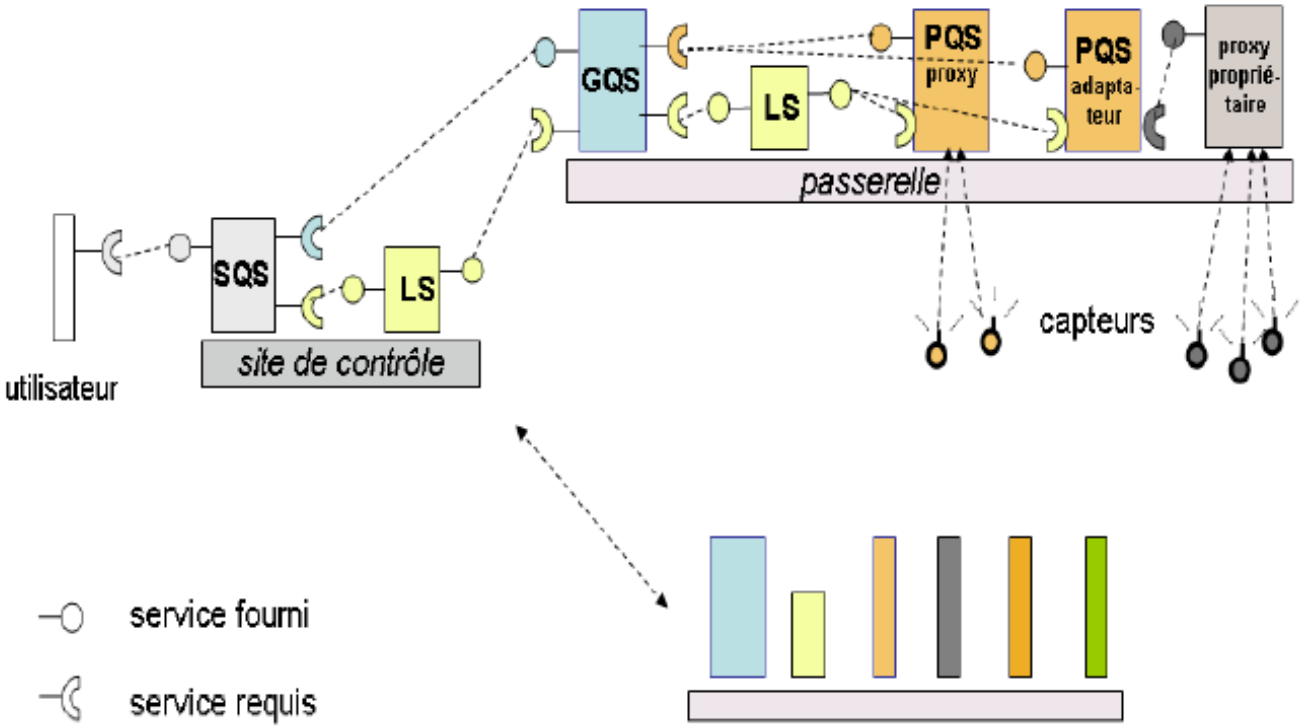
Expérimentation et cas d'étude

- **Objectifs**
  - Mesurer le surcoût de diagnostic
  - Analyser les performances relatives du diagnostic centralisé et du diagnostic distribué

- **Plate-forme expérimentale**
- **Cas d'étude : système de gestion à grande échelle de données de capteurs hétérogènes**

Cas d'étude

- **Système de gestion à grande échelle de données de capteurs hétérogènes (LIG – France Telecom)**



– Expérimentation : 14 composants PQS

Démarche expérimentale

- **Evaluation du coût en mémoire du service DISCO**
  - Mémoire physique occupée par le service DISCO
  - Mémoire utilisée durant l'exécution du service DISCO
- **Algorithme de diagnostic centralisé et diagnostic distribué**
  - Nombre de tests exécutés
  - Evolution du temps de diagnostic en fonction du nombre de composants
  - Evolution du temps de diagnostic en fonction du nombre de composants fautifs

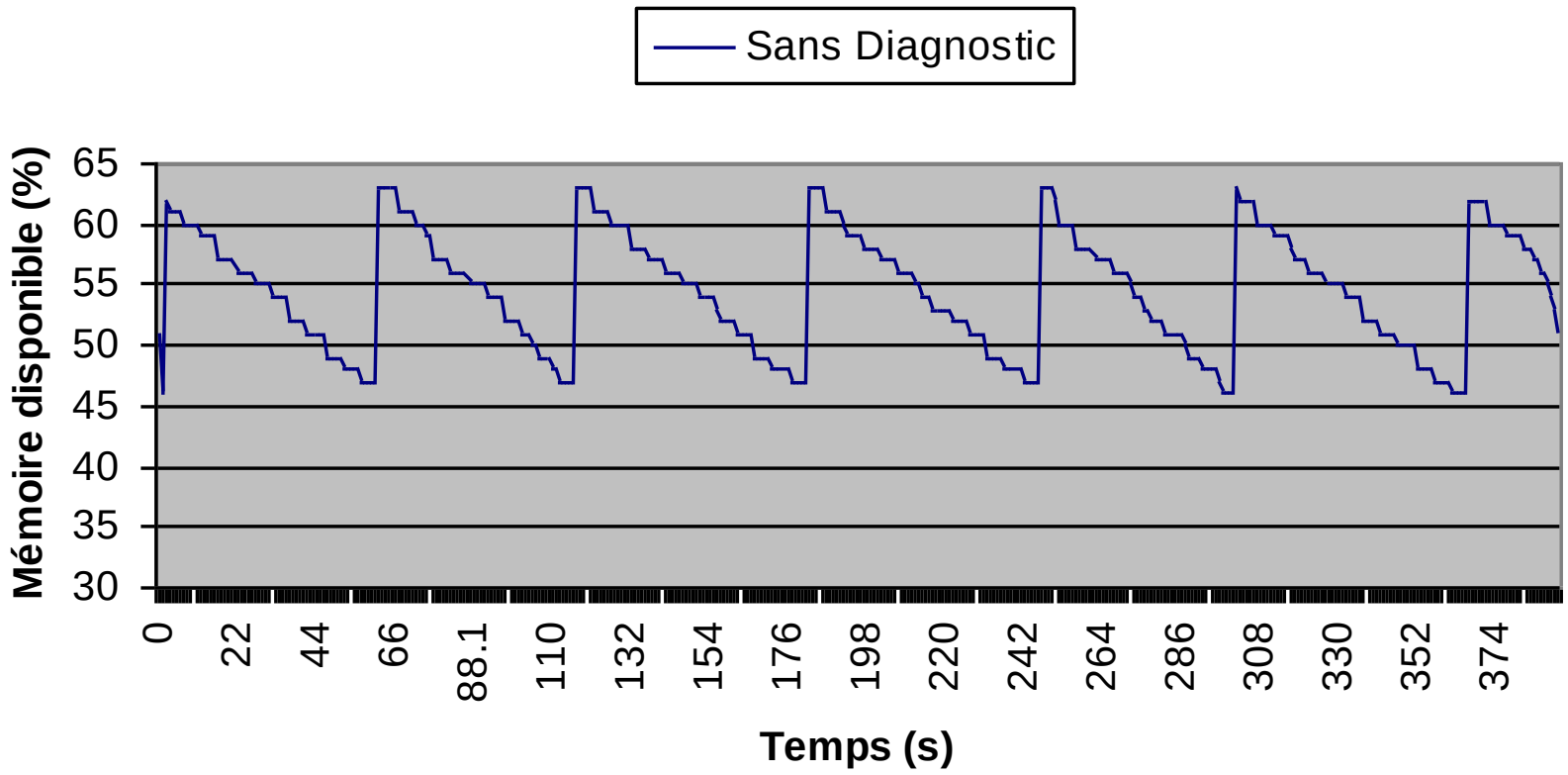
Résultats obtenus

- **Mémoire physique occupée par le service DISCO**

Critères	Surcoût de service de diagnostic	Surcoût dans chaque composant	Application	% comparé avec l'application
Taille des classes compilées	27.4 kB	9.2 kB	503.49 kB	~ 7.27 %
Taille des bundles	23.82 kB	4.51 kB	5.063 MB	~ 1.1%

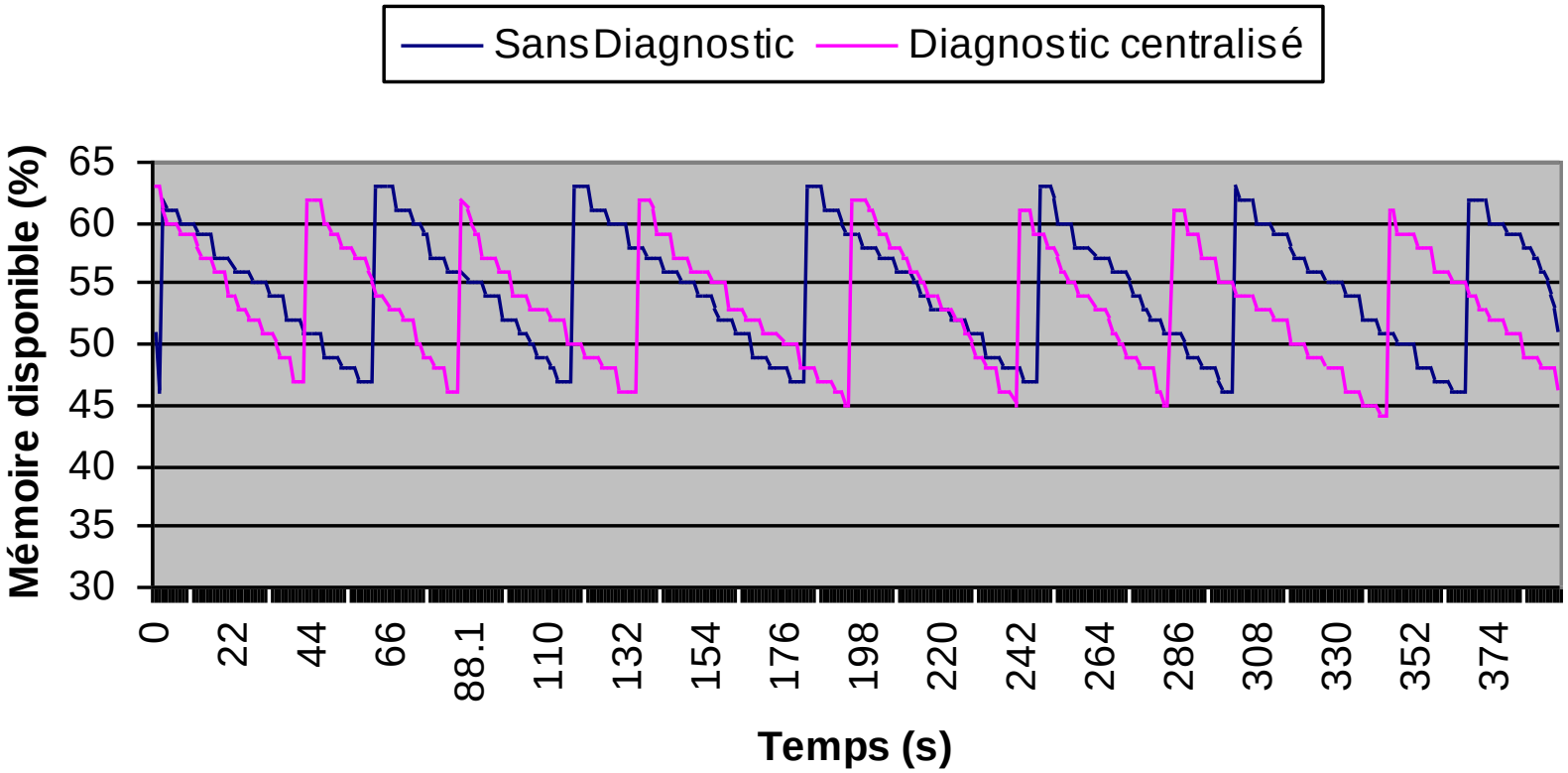
Résultats obtenus

- Mémoire utilisée durant l'exécution du service DISCO



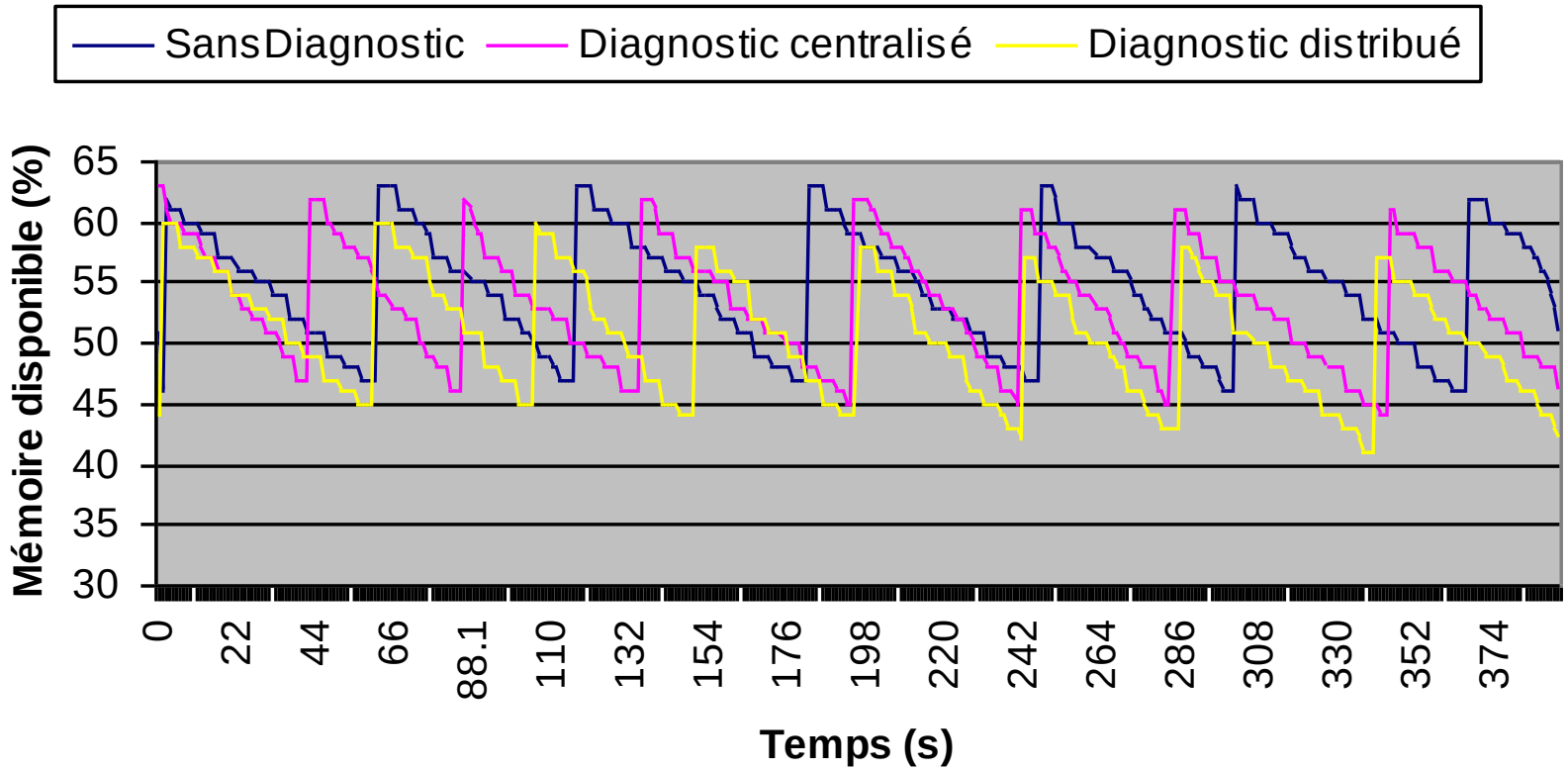
Résultats obtenus

- Mémoire utilisée durant l'exécution du service DISCO



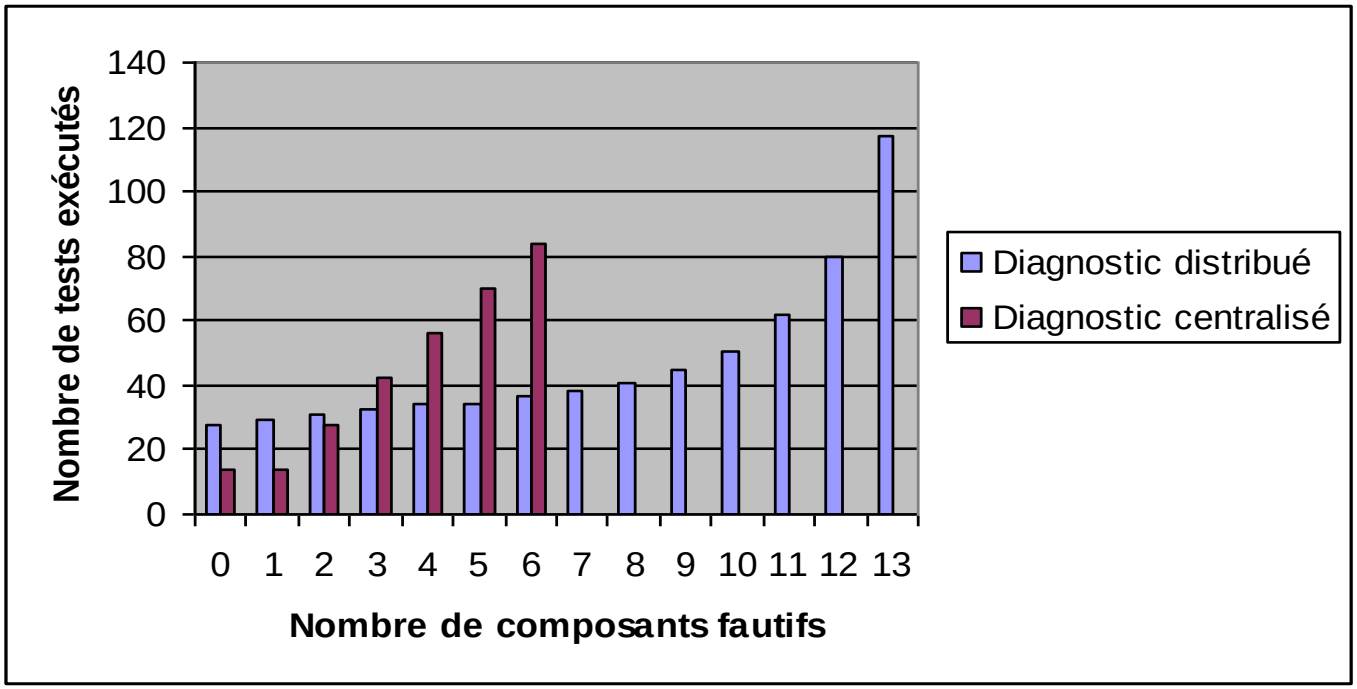
Résultats obtenus

- Mémoire utilisée durant l'exécution du service DISCO



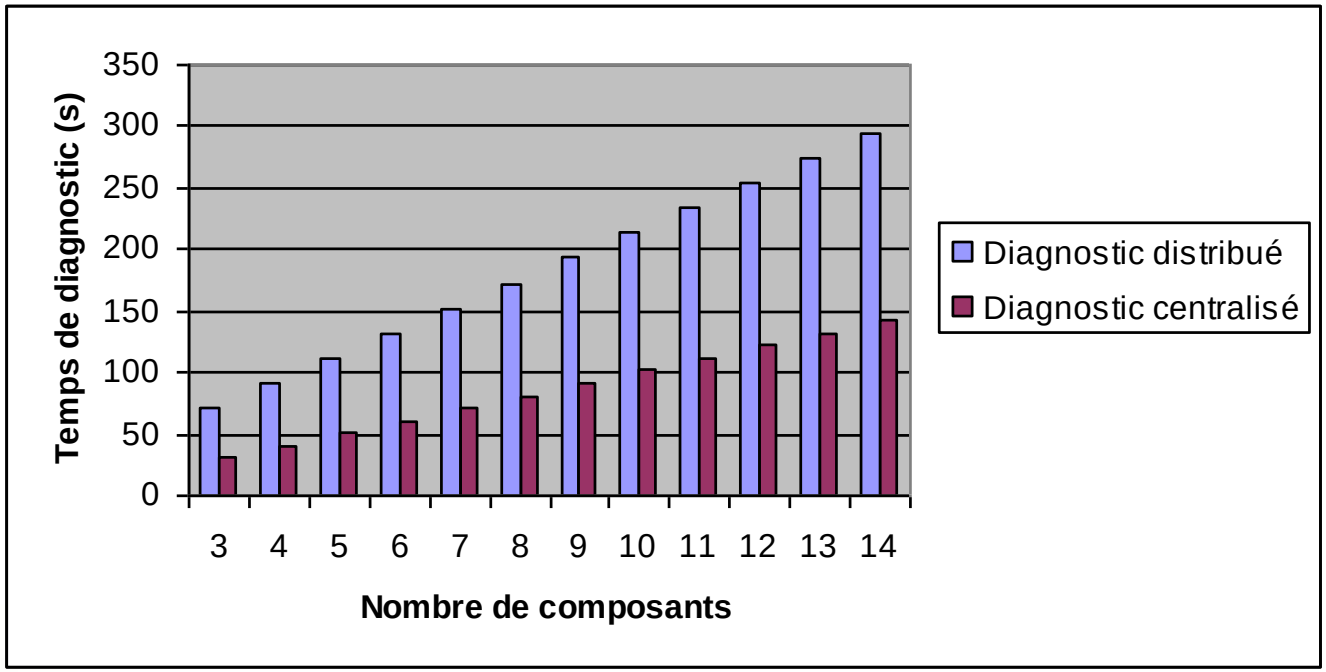
Résultats obtenus

- Nombre de tests exécutés



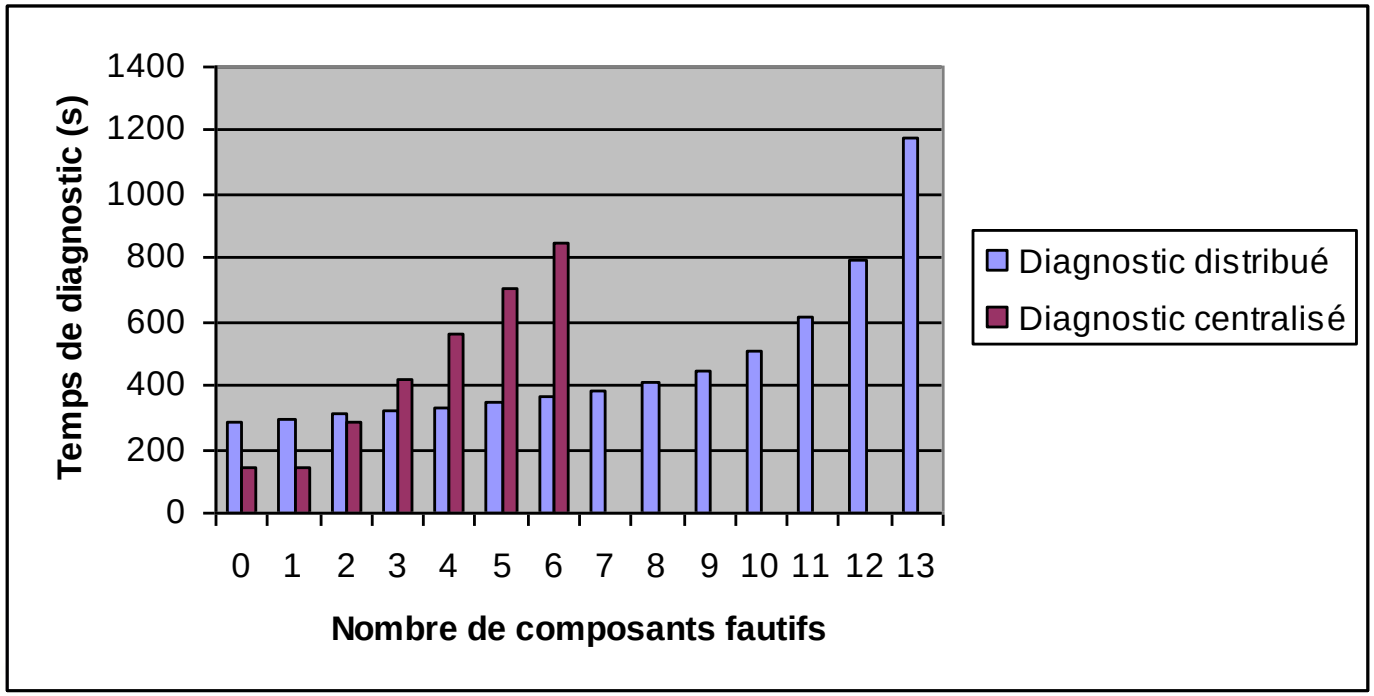
Résultats obtenus

- **Evolution du temps de diagnostic en fonction du nombre de composants**



Résultats obtenus

- **Evolution du temps de diagnostic en fonction du nombre de composants fautifs**



Expérimentation et cas d'étude

- **Conclusion**
  - Le surcoût dû au diagnostic est globalement acceptable
  - Les résultats expérimentaux que nous avons obtenus confortent les résultats théoriques
  - Le diagnostic centralisé et le diagnostic distribué présentent des caractéristiques différentes

Plan de présentation

- Contexte
- Approche de diagnostic
- Service de diagnostic
- Expérimentation et cas d'étude
- **Conclusion et perspectives**

Conclusion

- **Service de diagnostic DISCO**
  - Se baser sur les tests inter-composants en-ligne
  - Détecter et localiser les composants défaillants dans les applications à base de composants logiciels
  - Flexibilité
  - Architecture générale

Conclusion

- **Limitations**
  - Partitionnement : localisation géographique
  - Génération manuelle des cas de test et de l'interface de test
  - Ressources : mémoire

Perspectives

- **Partitionnement**
  - Fiabilité des composants
  - Nombre de tests à exécuter
  - Temps de communication
- **Enrichissement des tests et du diagnostic**
  - Modèle de fautes adapté au test en ligne des composants logiciels
  - Algorithmes de test et de diagnostic
- **Automatisation de l'intégration de l'interface de test dans les composants**
- **Interfacer le diagnostic avec un service de reconfiguration**
- **Test inter-composant et diagnostic dans les systèmes matériels/logiciels**

LABORATOIRE DE CONCEPTION ET D'INTEGRATION DES SYSTEMES

Merci de votre attention
Questions ?

Résultats obtenus

- **Surcoût du diagnostic sur le temps d'exécution de l'application**

	Sans diagnostic	Avec diagnostic centralisé	Avec diagnostic distribué
Temps d'exécution de l'application (ms)	21117.79	21121.85	21122.29

Résultats obtenus

- Evolution du temps de diagnostic en fonction de la durée du test