

Corosol : une JVM modulaire paramétrable à la volée

Christophe Deleray,
Nicolas Bedon, Gilles Roussel,
Etienne Duris

Institut Gaspard Monge
Université de Marne-la-Vallée



LMO'04



Motivations

Une JVM facilement modifiable
en particulier par l'application

Exemple

- **Persistance**

- Ajouter/modifier une instruction ou un type primitif
- Mapper le tas dans un fichier

- **Quand et par qui ?**

- au démarrage par l'utilisateur
- dynamiquement par l'application

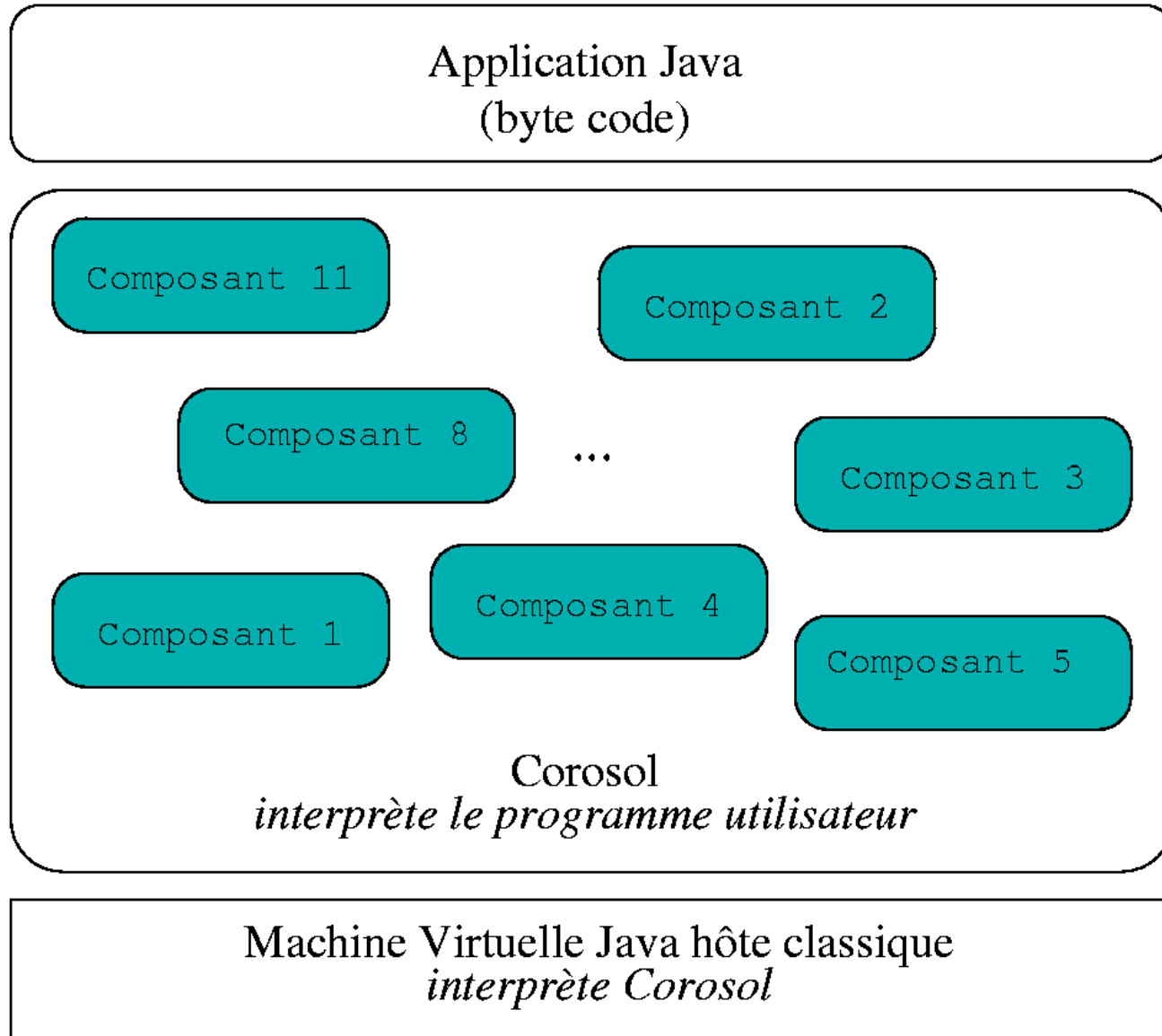
Corosol : Objectifs

- Fournir une **API de réflexion** de la JVM aux applications exécutées
- **Facilité** de configuration de la JVM:
 - ◆ **avant** l'exécution
 - ◆ **pendant** l'exécution
- **Portabilité** (100% pure Java)
- Architecture modulaire

Quelques travaux existants

- **Jupiter** : pour multi-processeurs
- **KissMe, PEVM** : pour persistance
- **Jikes RVM** : JVM ouverte en Java
- **VVM** : VM reconfigurable à la volée
- **Problèmes** :
 - ◆ Solutions ad-hoc et difficiles à combiner

Architecture de Corosol



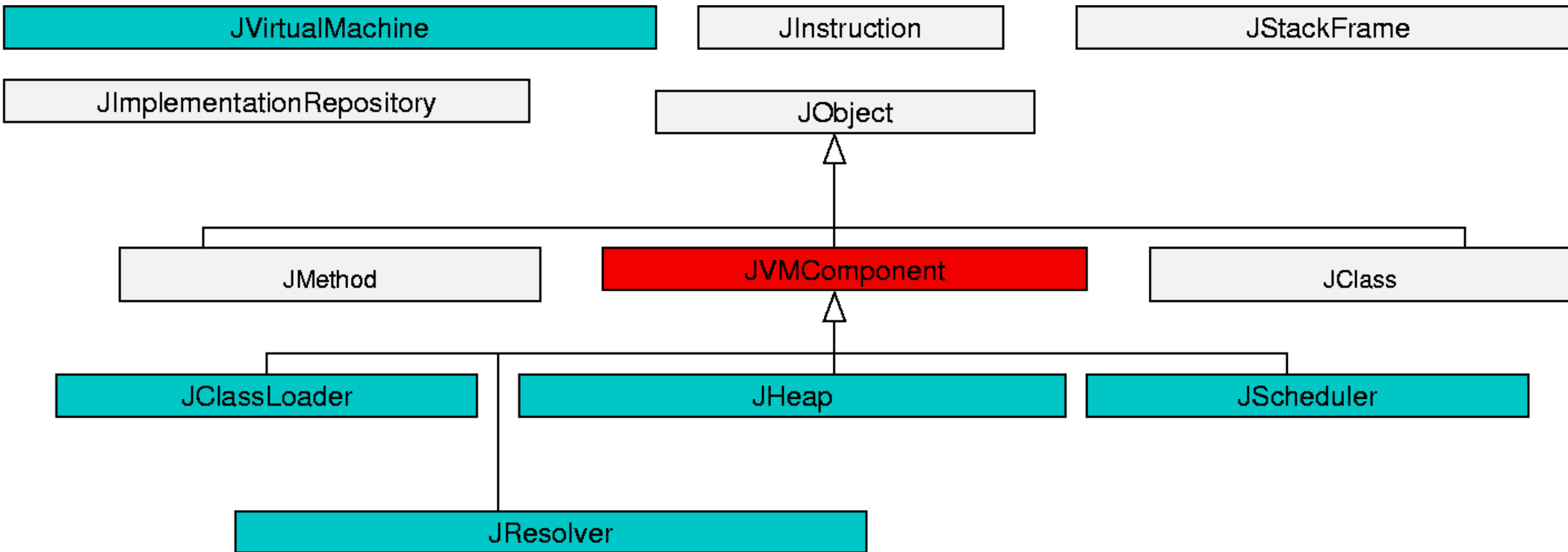
Composants de Corosol

- Unité fonctionnelle ou de stockage

En pratique:

- Groupe d'objets réalisant 2 interfaces :
 - ◆ une commune à tous les composants (**JVMComponent**)
 - ◆ l'autre décrivant ses **fonctionnalités** et son **type**

Architecture Java de Corosol



— Composant visible
par une application

Paramétrage avant démarrage

Fichier de propriétés

Au démarrage :

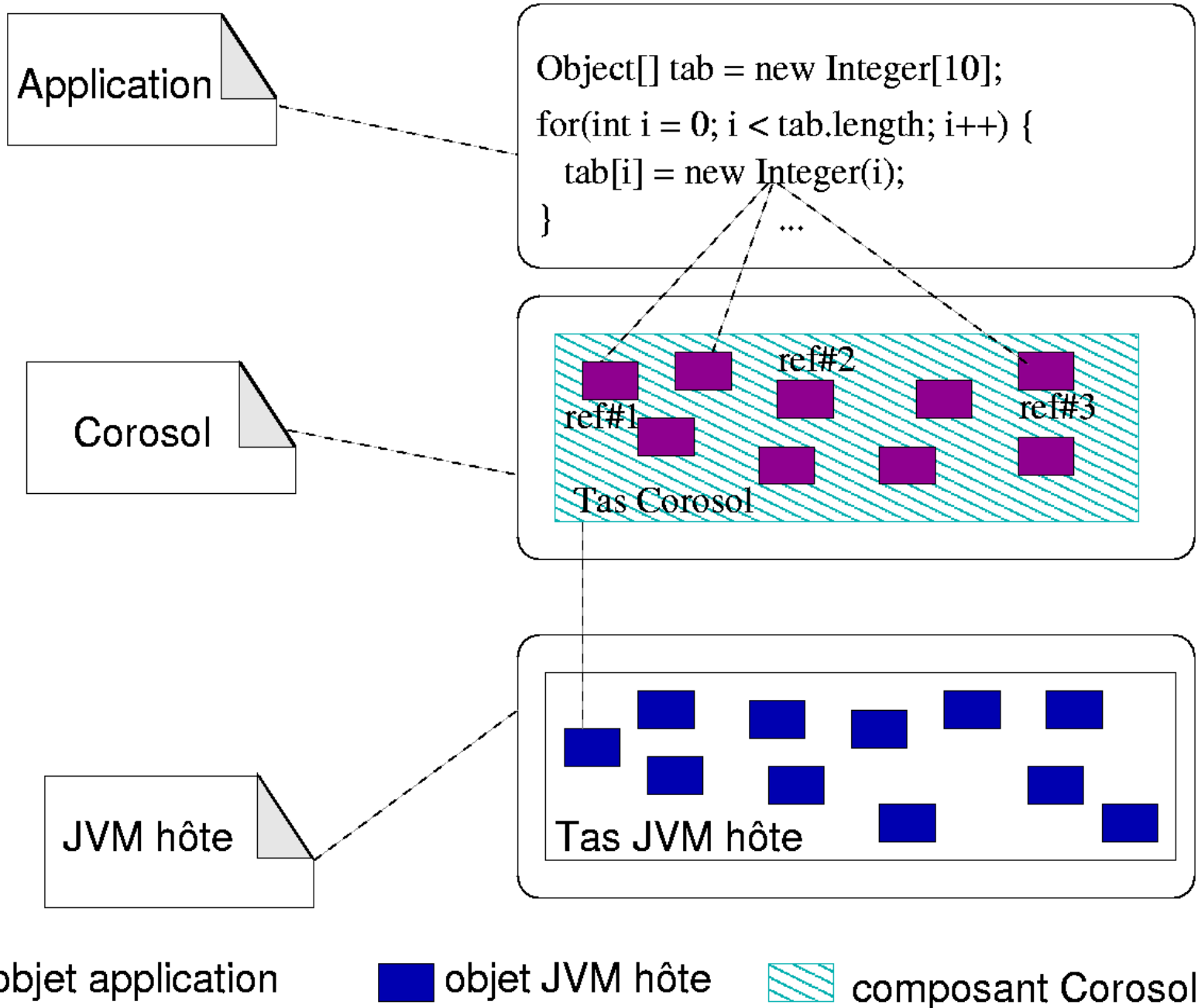
- 1) Lecture du fichier de propriétés
- 2) Création des composants
- 3) Configuration des composants

Type de composant	Implantation
"JVMVirtualMachine"	MyJVMDefaultImpl
"JHeap"	DefaultHeapImpl
"JThread"	DefaultThreadImpl
...	...

■ objet JVM hôte

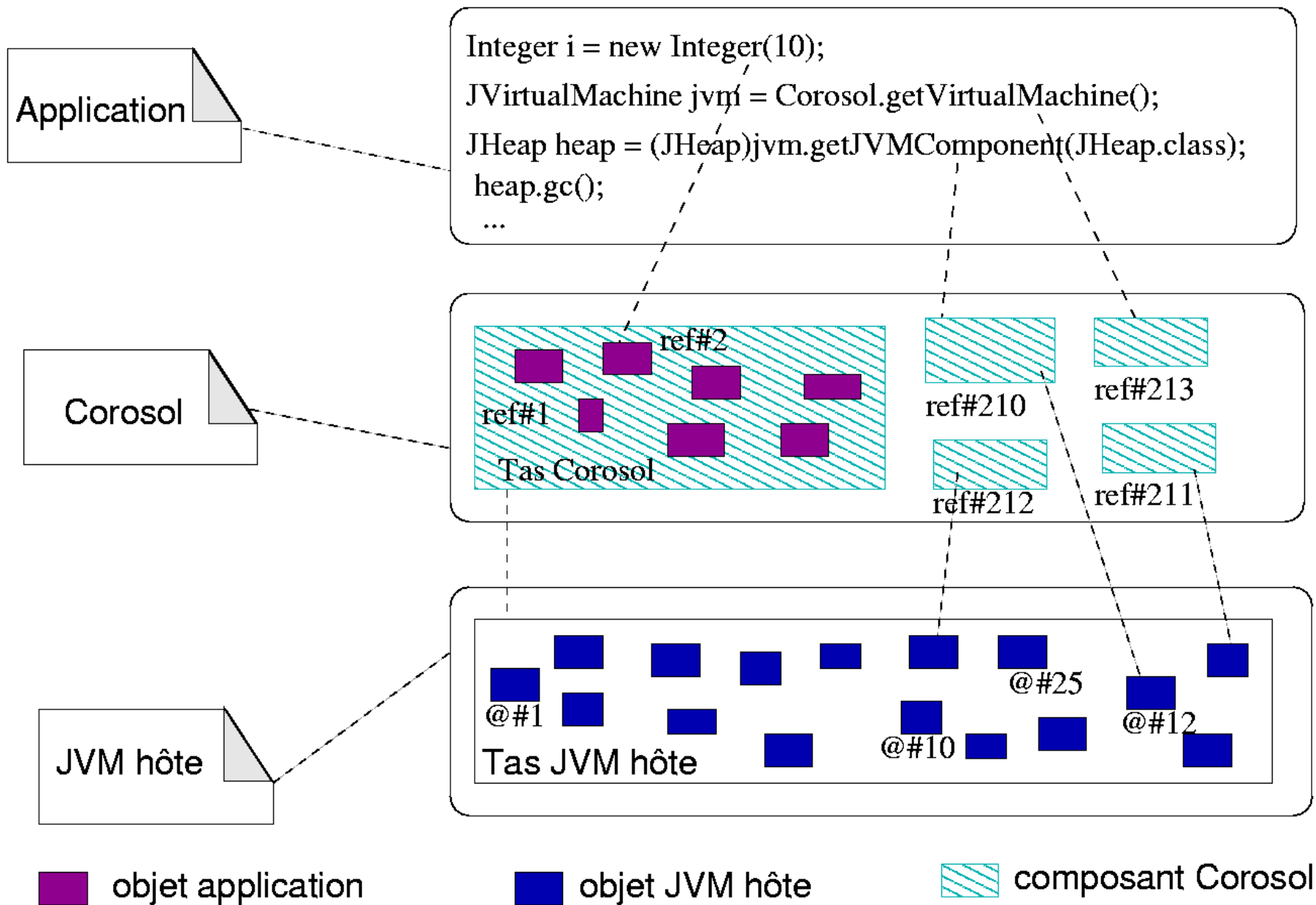
JVM hôte

Tas JVM hôte

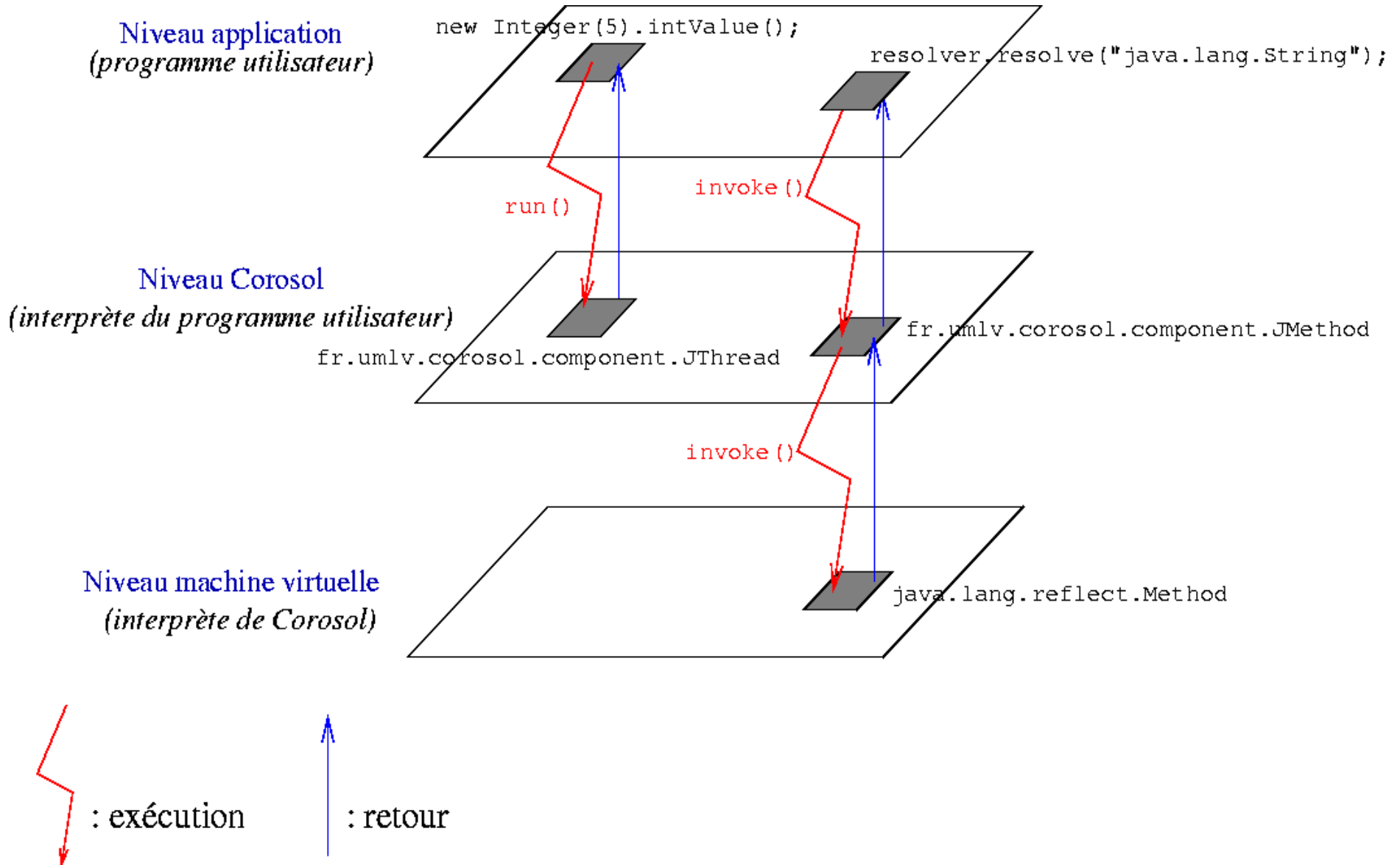


Consultation des composants

- Possibilité pour l'application de **consulter** les composants de Corosol
- **Comment ?**
 - ◆ Attribution d'une **référence** pour un composant au niveau Corosol
 - ◆ Manipulation de ces références sur la **pile** lors d'un appel de méthode sur un composant



Exécution



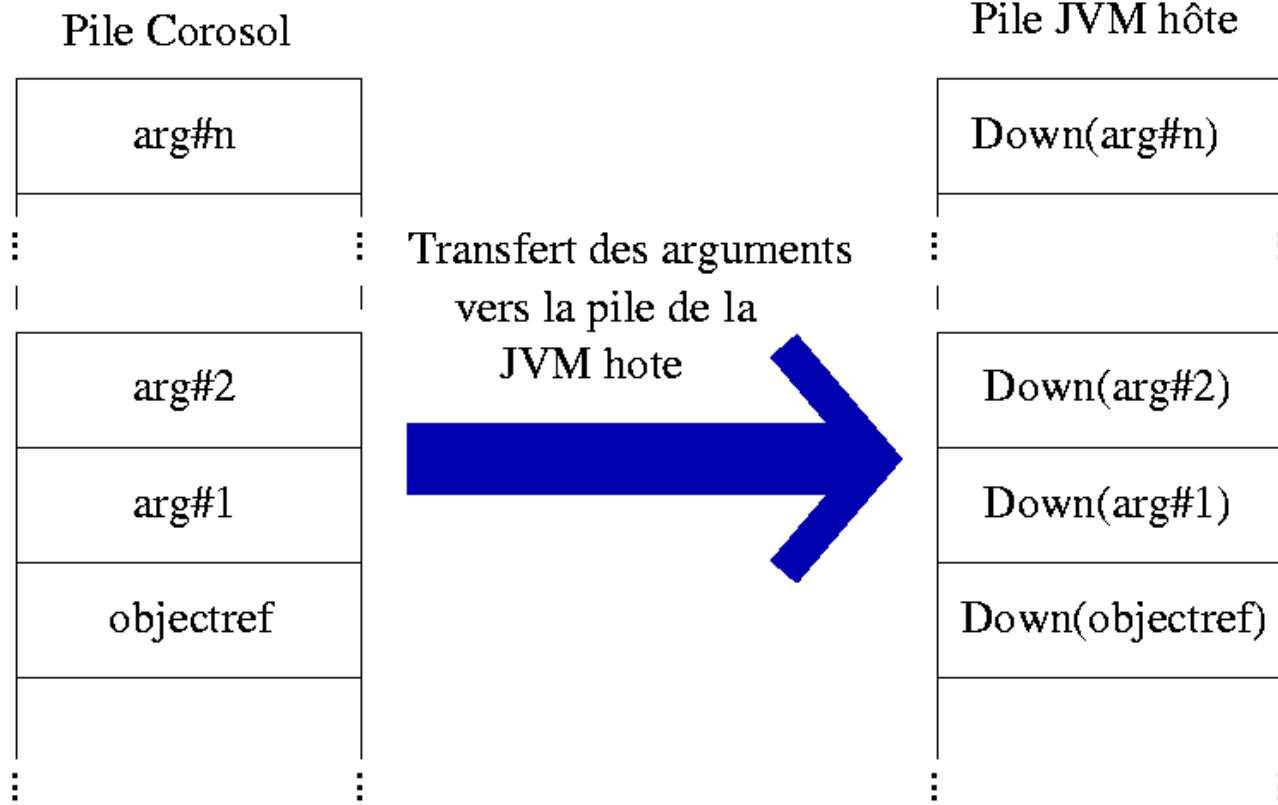
Symbiose (1)

- Utiliser des objets de la machine hôte dans Corosol
 - ◆ Si `value` : *type primitif*
 - `Up(value) = value`
 - ◆ Sinon
 - `Up(value) = ReferenceOf(value)`

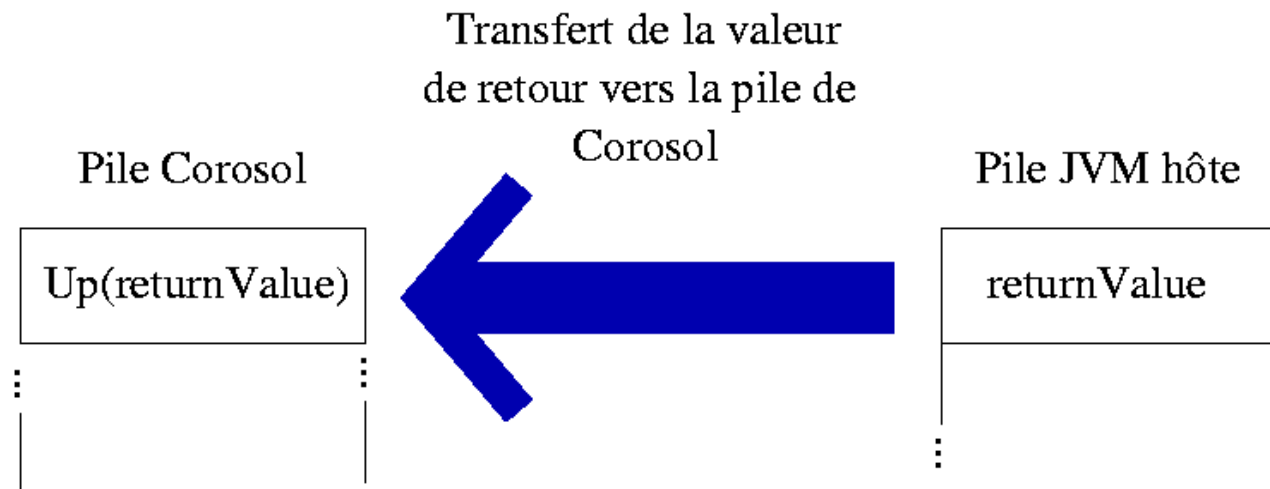
Symbiose (2)

- Utiliser des objets de Corosol dans la machine hôte
 - ◆ Si **value** : *type primitif*
 - Alors **Down(value)** = **value**
 - ◆ Sinon si **value** : *référence d'un composant*
 - **Down(value)** = **ReferentOf(value)**
 - ◆ Sinon
 - **Down(value)** = **Proxy(value)**

Down



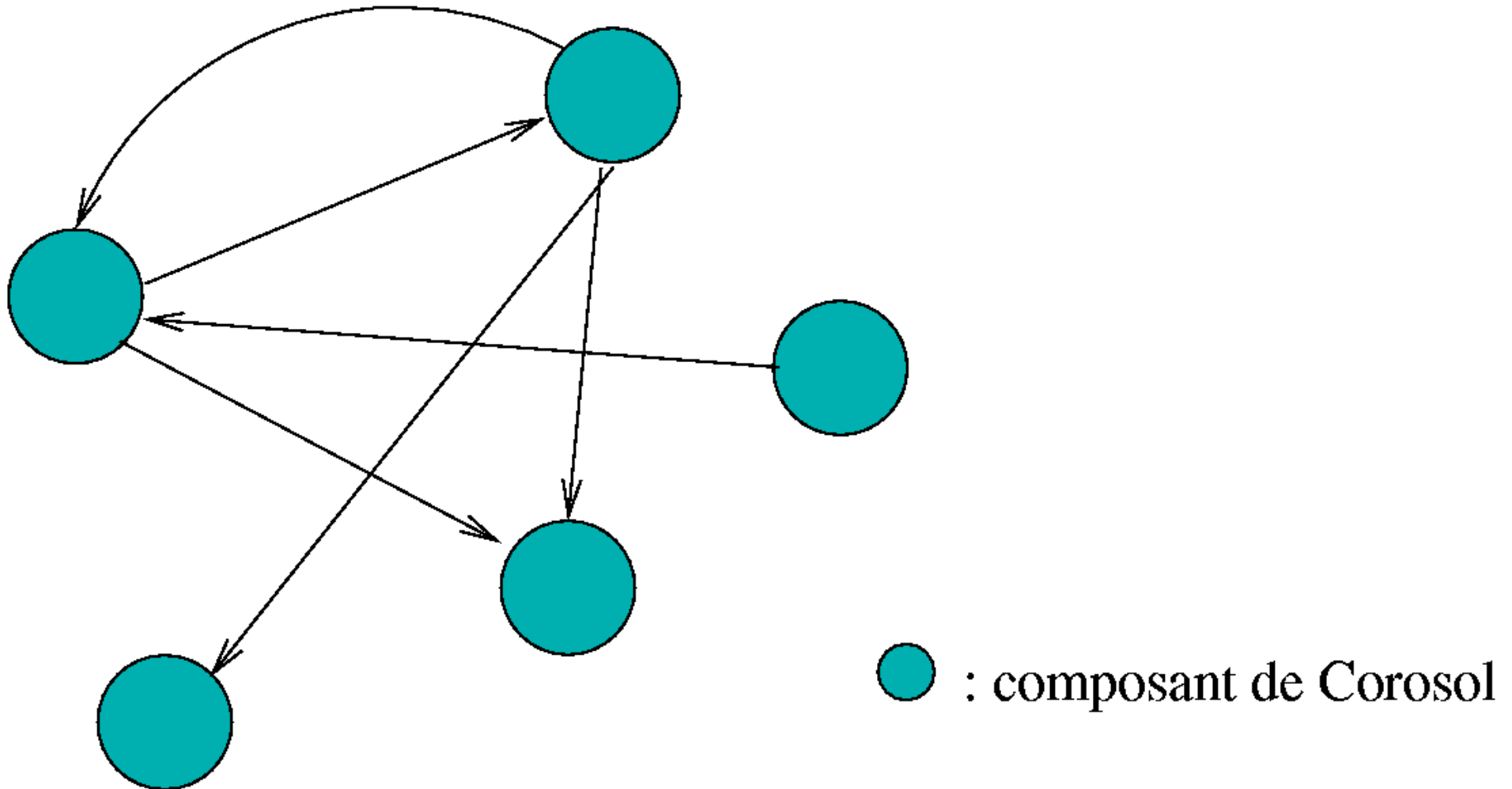
Up



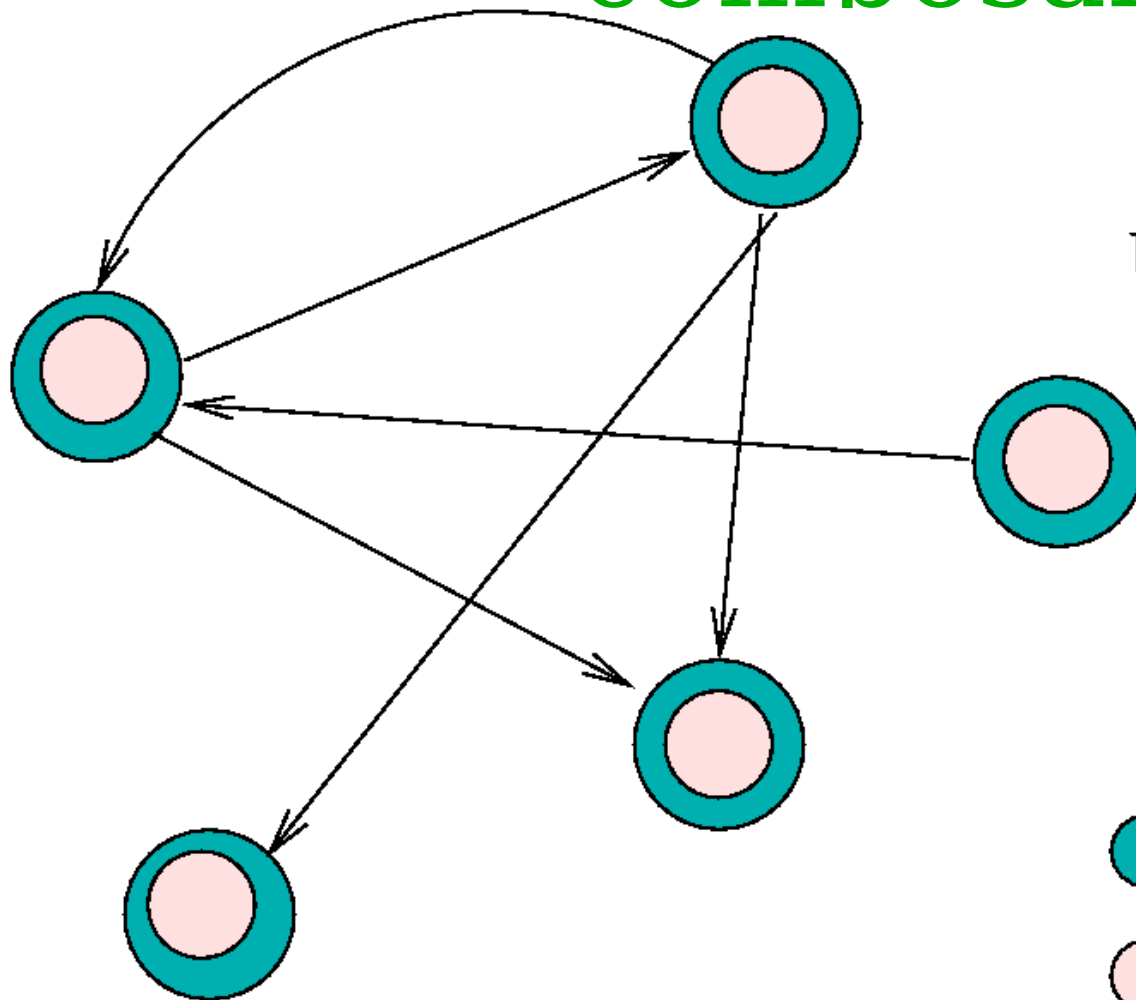
Remplacement de composant

- Changement de composant à la volée
- Mise à jour des références
- Mécanisme sans vérification d'intégrité

Remplacement d'un composant (1)



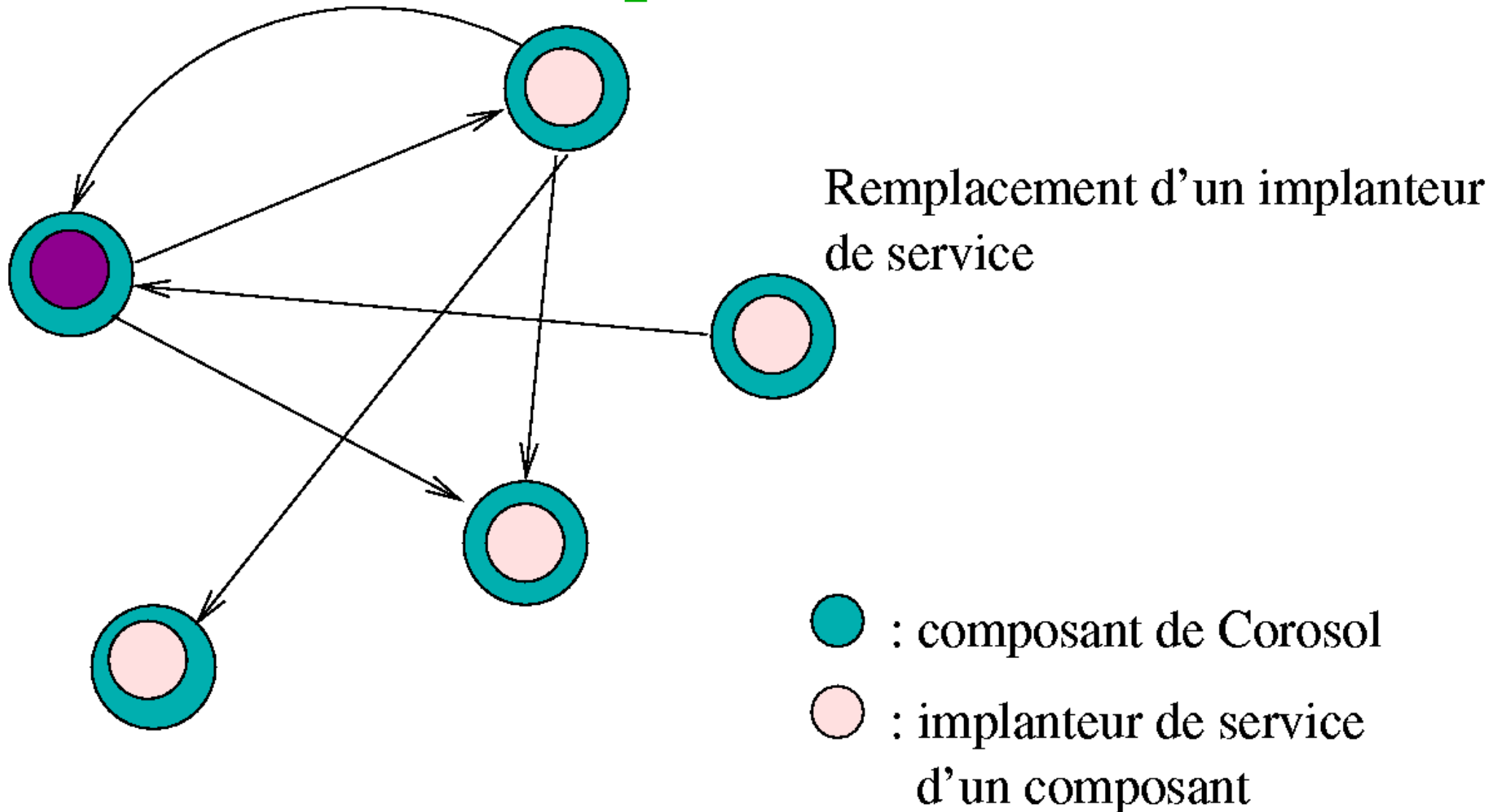
Remplacement d'un composant (2)



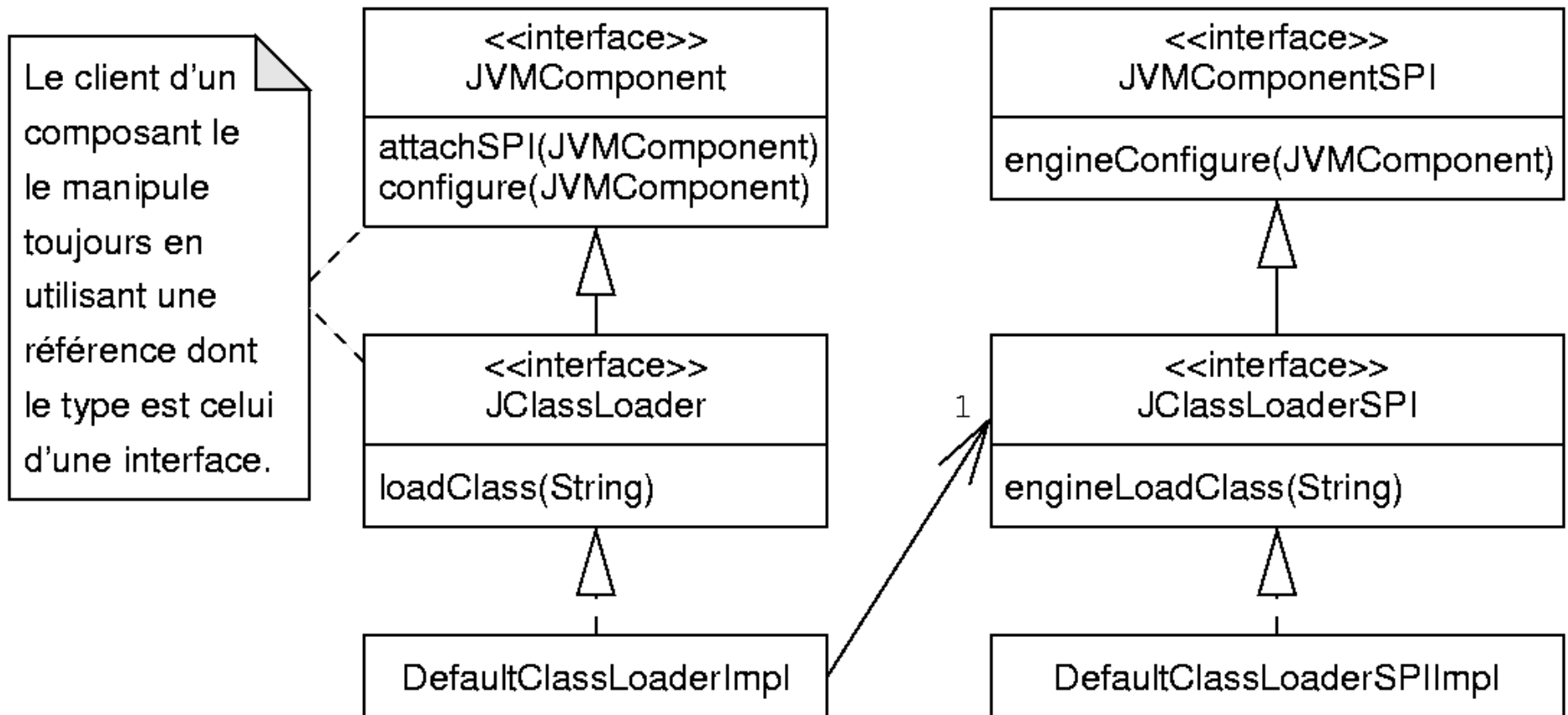
Utilisation du Bridge Pattern

- : composant de Corosol
- : implanteur de service d'un composant

Remplacement d'un composant (3)



Retour sur la structure d'un composant



Conclusion (1)

- Corosol, une JVM :
 - ◆ Facilement modifiable avant et durant l'exécution
 - modifications portables
 - ◆ Entièrement écrite en Java
 - 54 interfaces, 242 classes concrètes
 - ~30000 lignes
 - Libre, <http://www-igm.univ-mlv.fr/~cdeleray/>
 - ◆ 100 fois plus lente qu'une JVM classique
 - ◆ Étude pédagogique des JVM

Conclusion (2)

- Travaux en cours
 - ◆ Migration de l'état des composants
 - ◆ Transfert générique des arguments vers la JVM hôte
 - ◆ Finalisation
- Travaux futurs
 - ◆ Amélioration des performances
 - ◆ Utilisation des composants de la JVM sous-jacente s'ils ne sont pas modifiés
 - ◆ Ajouter un JIT