# COUPLAGE de Modèles

Méthodes d'implémentation

# Plan

- ☑ Couplage quand?
- ☑ Couplage comment?
  - ✓ Aspects à privilégier
  - ✓ Utilisation d'un protocol de communication
  - ✓ Avec ou sans coupleur
  - ✓ Coupleur statique/dynamique (OASIS/PALM)
- ☑ Interopérabilité (CORBA)

# Couplage quand?

**Obtenir une meilleure représentation et une intéraction réelle entre les systèmes**

- ✓ Problèmes multi-disciplinaires (océan-atmosphère, fluide-structure…)
- ✓ Problèmes multi-échelles
- ✓ Problèmes multi-physiques
- ✓ …

# Le couplage est-il nécessaire?

- ☑ **Alternatives**: Développer une application intégrant l'ensemble des connaissances nécessaires
  - ✓ Problème de mémoire, d'intégration, faible flexibilité

  ➔**Dans le cas d'applications simples**
- ☑ **OUI:**
  - ✓ Lorsque les phénomènes se complexifient et requièrent un haut degré d'expertise dans chacun des composants du couplage
  - ✓ Dans le cas des codes patrimoines à coupler
  - ✓ Pour apporter **plus de flexibilité dans l'introduction de nouveaux algorithmes de couplage**, éviter une restructuration du code à chaque nouvelle méthode et travailler sur l'interface des composants
  - ✓ Couplage de composants hétérogènes sur des calculateurs distants
  - ✓ …

# Aspects à privilégier?

- **Portabilité:** utilisation des standards les plus répandus
  - Installation sur des plates-formes hétérogènes
  - Pérennité maximale
- **Modularité**
  - Chaque composant est vu comme un module ayant une fonctionnalité spécifique (adapté à l'aspect pluridisciplinaire)
  - Permet de remplacer un module par un autre module ayant les mêmes fonctionnalités sans conséquences sur le reste du code.
  - Séparer l'implémentation de l'algorithme de couplage de celle des composants
- **Extensibilité**: possibilité d'ajouter des composants sans déstructurer l'architecture globale
- **Interopérabilité**: déploiement des modules de l'architecture couplée sur une grille hétérogène dans le but:
  - D'optimiser l'utilisation des ressources
  - De coupler des codes développer dans des contextes différents (machine, réseau, langage,…)

# Plusieurs approches

- ☑ **Inclure les fonctionnalités de couplage** « sans coupleur » en tant que services intégrés, dans les codes des modèles à coupler: **MPI**, **CORBA**…
- ☑ **Développer un code spécialisé (Coupleur)** qui va apporter la flexibilité dans l'implémentation des algorithmes de couplage, se charger d'établir la cohérence entre les entrées et les sorties des codes à coupler (**OASIS**, **PALM**…)
- ☑ **Inclure les fonctionnalités de couplage** en répondant aux besoins **d'intéropérabilité: CORBA**,…

# Bibliothèque Message passing (**MPI**)

- Les composants du couplage communiquent entre eux par échange de messages
  - Introduire dans le code des appels à la bibliothèque (standard MPI)
- **Méthode trop intrusive** n'apporte pas les critères de modularité, d'extensibilité et de réutilisabilité
  - Non générique, restructuration du code à chaque nouvel algorithme
  - Complexe à implémenter, à déboguer et à optimiser
- **Mélange les deux niveaux de parallélisme**, le parallélisme du couplage et celui des composants

# OASIS et PALM

- **OASIS** et **PALM** partagent:
  - Gestion externe de l'algorithme de couplage
  - Modularité et synchronisation des échanges des champs de couplage et des opérations sur ces champs
- **OASIS**: coupleur « **statique** »
- **PALM:** coupleur « **dynamique** »
  - Exécution dynamique des composants d'un système couplé et possibilité de les combiner selon un algorithme de couplage sophistiqué
  - Interface permettant de définir les fonctionnalités du couplage: **Prepalm**
  - …

# OASIS

- Développé dans le cadre du projet PRISM
  - **OASIS 4** : hautement parallèle permet de traiter des gros volumes de données
- Définition d'une **interface statique** entre les composants du système couplé
- **Composants**:
  - **Driver**: gère de façon synchrone les composants du couplage
  - **Transformer**: effectue les interpolations
  - **Bibliothèque** qui permet aux composants de communiquer avec le reste du système couplé

# Contexte de développement de PALM

- ☑ Implémentation modulaire **de chaînes d'assimilation de données** au sein du projet MERCATOR
  - ✓ **Les algorithmes d'assimilation** :
    - Peuvent être vus comme des **séquences** d'exécution **différentes des mêmes opérateurs de base**
    - Utilisent des **calculs algébriques lourds**
  - ➔ **Double contraintes**:
  - ✓ Recherche de méthodes **: flexibilité, modularité** ➔ réutilisabilité, extensibilité
  - ✓ Exploitation opérationnelle: **Performance**

Les **concepts de PALM** peuvent être appliquées à des applications plus générales de **couplage dynamique de modèles**

# Outil de couplage (PALM): objectifs

- ☑ **Flexibilité – Modularité** (unité PALM)
  - ✓ Unités indépendantes de l'algorithme de couplage
  - ✓ Dans l'enchaînements des unités et les communications entre ces unités (interface **Prepalm**)
- ☑ **Performance**
  - ✓ Performance des **accès mémoire**
  - ✓ Performance et flexibilité des **communications**
  - ✓ **Séparation des niveaux de parallélisme**
- ☑ **Prend en charge**
  - ✓ Gestions des communications entre les unités parallèles ou non
  - ✓ Transformation et distribution des données transférées entre composants
  - ✓ La gestion et la synchronisation de la ou des séquences de composants élémentaires définie dans Prepalm
  - ✓ …
- ☑ **Portabilité**

# Exécution d'une application avec PALM

- **Deux versions de PALM**: **PALM_RESEARCH** et **PALM_MP**
- PALM_RESEARCH(1ère version):
  - modèle SPMD (Single Program Multiple Data) construit sur MPI-1
  - Simule un coupleur MPMD
- PALM_MP (3ème version):
  - modèle MPMD (Multiple Program Multiple Data) construit sur MPI-2
  - Un processus driver qui est capable de générer de nouveaux processus (spawn)
  - Fonctionnalités supplémentaires (objets dynamiques…)

# Couplage Dynamique

- **Définition d'un algorithme de couplage**:
  - Assemblage de composants élémentaires indépendants dans Prepalm
  - Lancer des composants en cours de simulation
  - Répéter en boucle une partie de l'application (FOR,WHILE)
  - Exécution conditionnelle d'une partie de l'application (IF,SELECT,…)
  - Utiliser des primitives de communications de PALM
  - Changer la distribution parallèles des données transmises dans PALM
  - …

# Modularité totale de l'application

- **Pour assurer l'indépendance entre l'algorithme de couplage et les unités PALM**
  - ✓ Pas de référence à l'algorithme dans une unité
  - ✓ La description de l'algorithme et du schéma de communication est faite dans une interface graphique **Prepalm**
  - ✓ La clé est le schéma de communication « end-point »

➔

  - ✓ Une même structure peut être utilisée pour différentes applications
  - ✓ Les mêmes unités peuvent être utilisées dans différents algorithmes sans modification

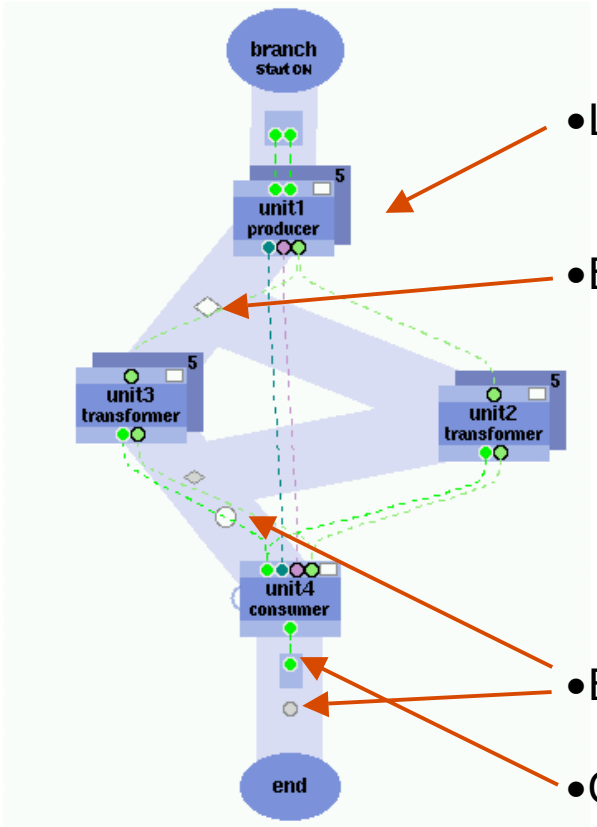
# Les concepts principaux



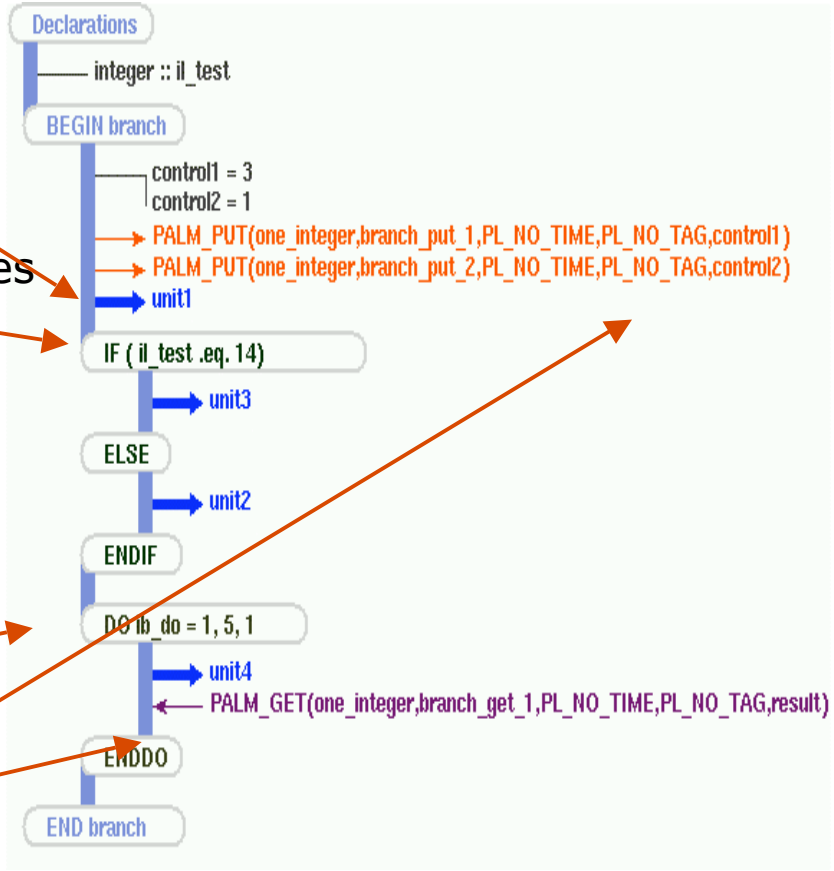
La description de l'algorithme et du schéma de communication utilise les concepts suivants :

- Unités utilisateurs et unités d'algèbre
- Branches
- Communications
- Objets
- Espaces
- Distributions et distributeurs
- localisation
- Times et tags
- Le buffer
- …

# Définir l'algorithme dans PrePALM




- Lancer des unités
- Exécution conditionnelles
- Boucles DO, While
- Communications



# Carte d'identité d'une unité

```
!PALM_UNIT -sub create\
!
!PALM_SPACE –name space1\
!          -shape (10,10)\
!          -element_size PL_DOUBLE_PRECISION\
!
!PALM_OBJECT -name obj1\
!            -space space1\
!            -intent OUT\
```

On fournira d'autres informations comme la **distributions d'un objet parallèle sur une unité**, la **localisation**, l'utilisation du paramètre temps,….

# Communications PALM

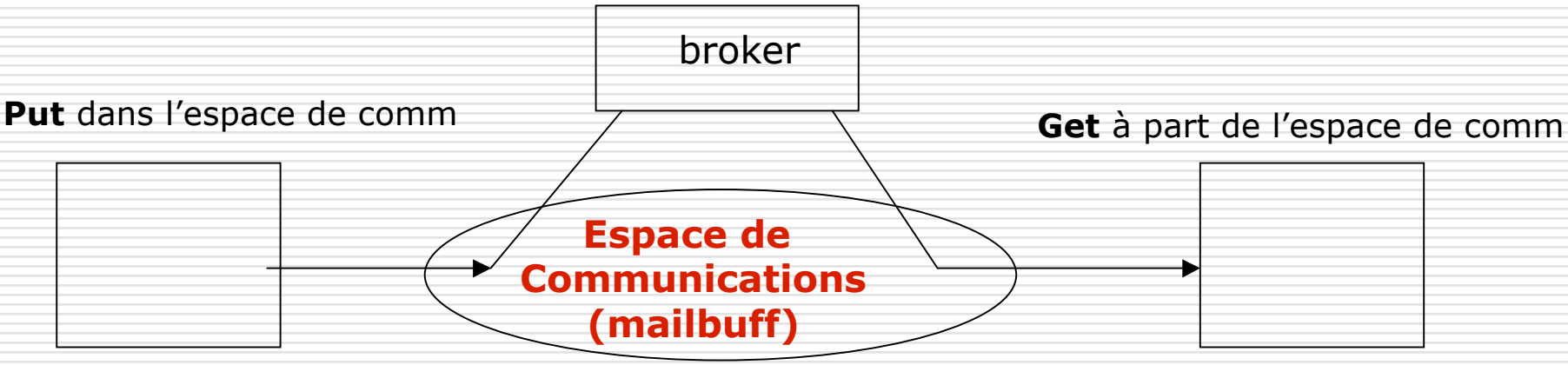
- ☑ Permet l'échange d'informations entre les unités
- ☑ Les données échangées sont des « **objets** »
- ☑ Une communication est un lien entre deux objets de deux unités
  - ✓ Les objets sont définis **dans les unités**
  - ✓ Les communications dans la configuration de l'application PALM (prepalm)

    ➔ assure la modularité
- ☑ C'est le driver palm qui assure la synchronisation des communications

# Schéma de communication end-point

**La modularité est assuré si le code des « unités » est indépendant du profil des communications de l'application**

**Modèle de communication « end-point »**




- La part d'information transmise est un « object »
- Le driver de PALM gère les requêtes de communication dans l'espace de communication
- **Le « put » ne connaît pas sa cible et le « get » ne connaît pas sa source ➔indépendance des unités et des communications**

# Date et « time stamp »

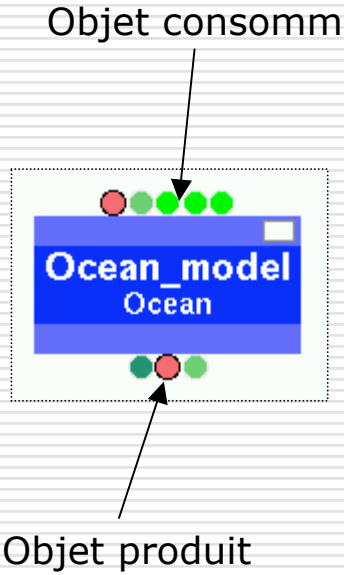
- ☑ Identificateur de temps associé à une même donnée à des temps différents
- ☑ Code utilisateur:entier identifiant un temps associé à l'objet transmit ou reçu
- ☑ L'objet aura l'attribut time dans sa description sur la carte d'identité de l'unité
- ☑ Communications: un temps ou un intervalle de temps peut être fourni dans une communication PALM
- ☑ Lorsque le temps de l'objet source est différent du temps de l'objet cible, PALM peut effectuer une interpolation temporelle

# Unités PALM

- ☑ Partie de code (sous-programme fortran ou fonctions C++) qui peut **consommer** ou **produire** des données (**PALM Object**) en appelant des primitives PALM
- ☑ **Instrumentation des unités**
  - ✓ Source « end-point »

call PALM_Put(‘space1’,’obj1’,time, tag, array,err)

Palm, j’ai produit l’objet nommé “obj1” qui appartient à l’espace “space1”, et est associé au temps time, au tag tag, dis-moi où l’envoyer (unité ou espace de stockage implicite)



# Unités PALM

✓ Target « end-point »

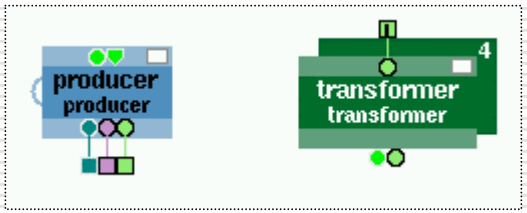
Call PALM_Get(‘space2’,’obj2’,time, tag,array,er)

Palm, j’ai besoin de l’objet nommé “obj2”, …, regarde si cet objet est dans ton espace de stockage implicite (**mailbuff**) ou si une unité peut m’envoyer directement cet objet.

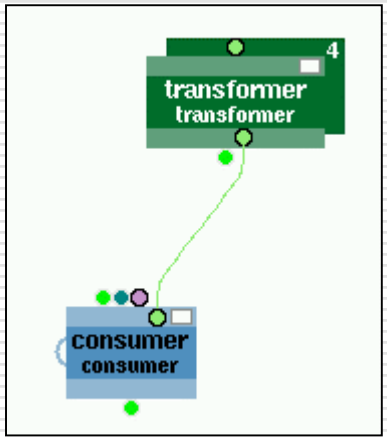
➔**Ces unités pourront être assemblées dans une séquence d’exécution dans PREPALM: PALM Branche**

# Communication effective

- ☑ Instrumentation du code: unité, code de branche



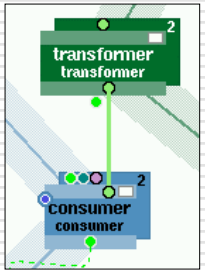

- ☑ Définition d'un « tube » dans Prepalm entre une source et une cible

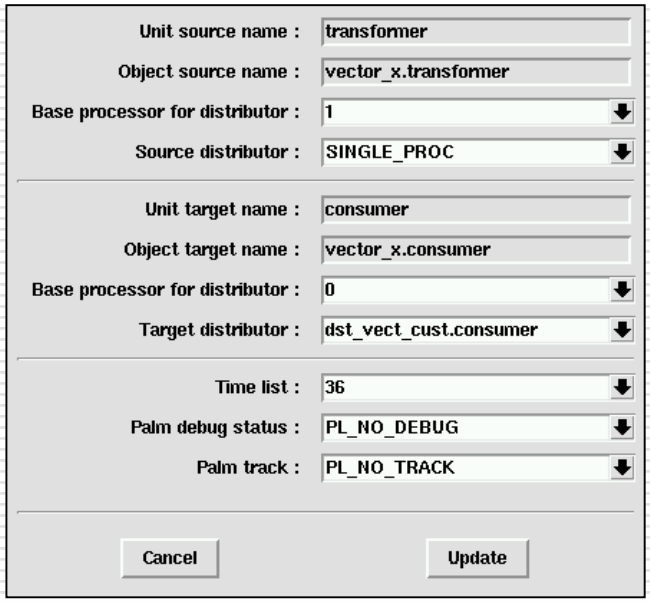


# Communications entre unités PALM

Echange d'objets entre unités

**PrePALM**




## Autres informations

- Distribution de la cible et de la source
- localisation des objets
- Paramètres temps
- Tag

# Communication par buffer

- ☑ Stockage explicite d'un objet dans un buffer PALM
- ☑ Définition dans Prepalm à la définition de la communication (tube). Aucune information dans la carte d'identité de l'objet ou dans la primitive de communication.

➔ préserve l'indépendance de l'unité avec l'algorithme de couplage

- ☑ Assemblage par combinaison linéaire
  - ✓ **Interpolation temporaire**
  - ✓ **Transformation des données transférées**

# Modularité

- Pour assurer la modularité, la description d'une communication est composée de trois parties
  - La « carte d'identité » de l'unité source qui décrit les objets qui **peuvent** être produits
  - La « carte d'identité » de l'unité cible qui décrit les objets qui **peuvent** être requis
  - Les informations qui permettent de relier la requête source et la requête cible seront définies dans **Prepalm**, elles dépendent de l'algorithme de couplage. Elles seront connues par le driver PALM,

# Unité algébrique

Permet de séparer la partie algébrique de la partie physique du problème

☑ **2 méthodes:**

- ✓ Utilisation du buffer.

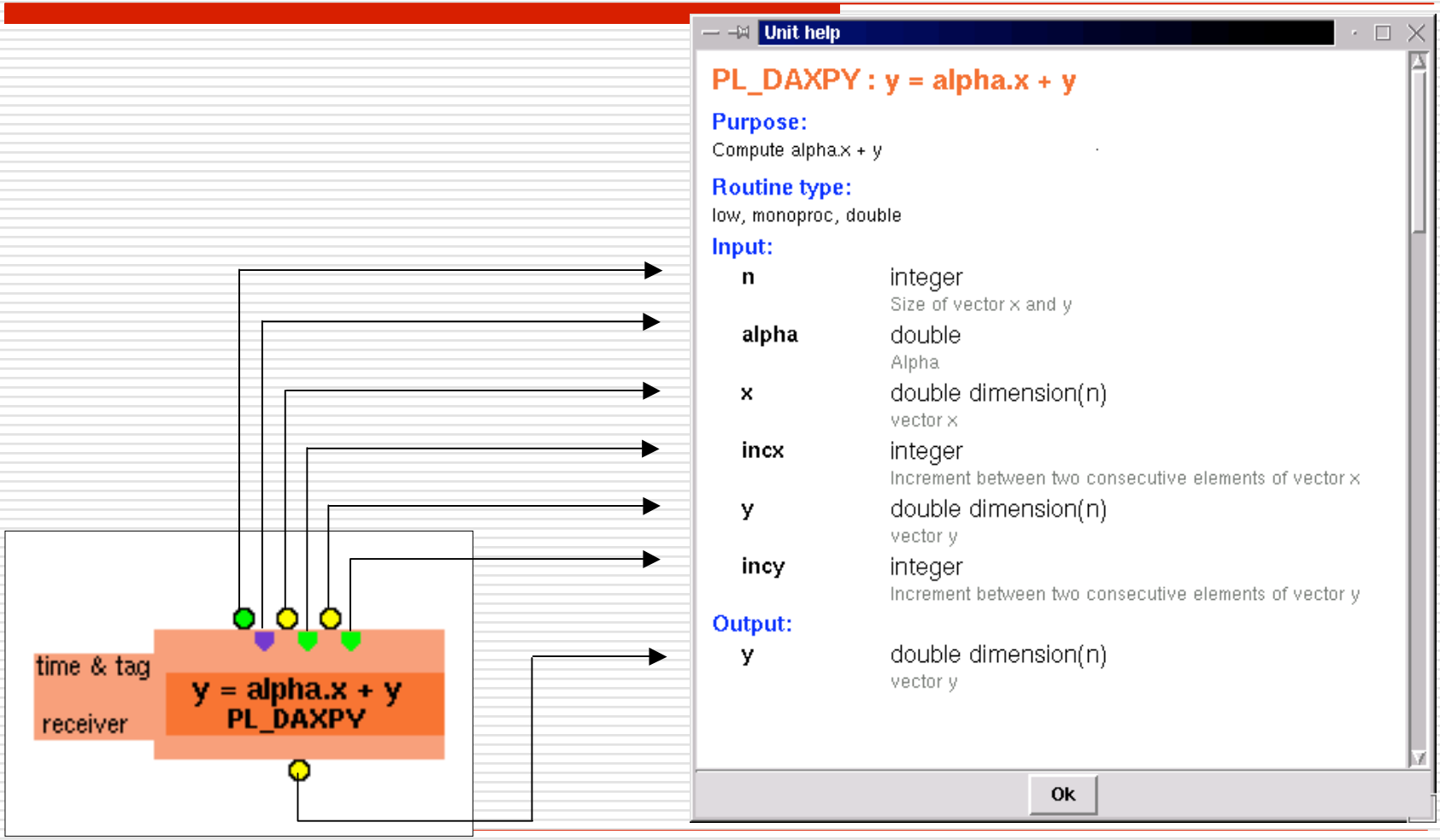
$$J=J1+J2+\ldots+Jn$$

Le contenu du buffer lorsque toutes les contributions auront été ajoutées

- ✓ Utilisation **d'unités algébriques prédéfinies** construite à partir de bibliothèques mathématiques, opérant sur des objets PALM

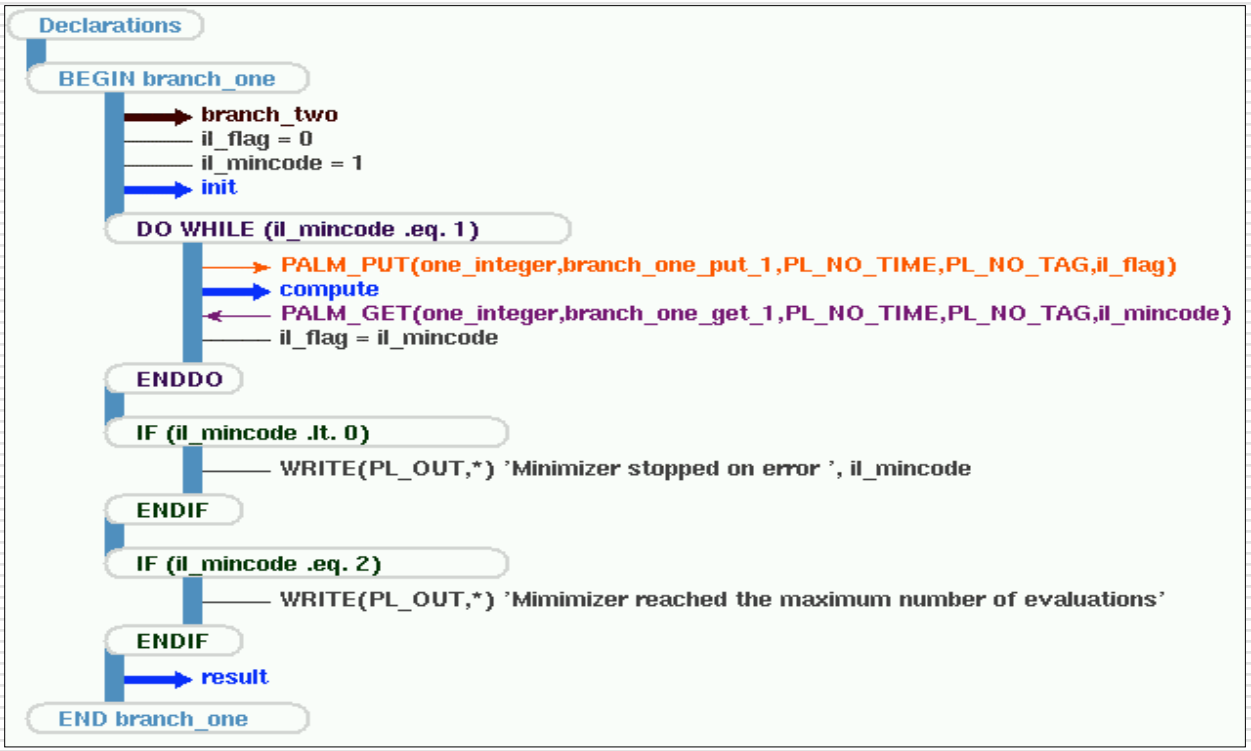
➔ **Algébra toolbox**

# Unité Algébrique prédéfinie



# Les branches

**Définition**: Suite d'unités et d'instructions fortran90, de scripts utilisant des structures de contrôle (if, select,loops), des primitives de communications (get et put d'objects scalaires)



# Optimisations des communications

- **Objectifs**:
  - Minimise le nombre de messages envoyés par MPI
  - Minimise l'espace mémoire utilisé
- **Optimisations**
  - get bloquant
  - put jamais bloquant:
  - get avant le put: transfert direct de la source vers la cible
  - put avant le get: transfert temporaire vers une espace « volatile » appelé mailbuff
  - Les communications entre unités s'exécutant en séquence sur les mêmes processeurs utilisent un mailbuff local
  - …

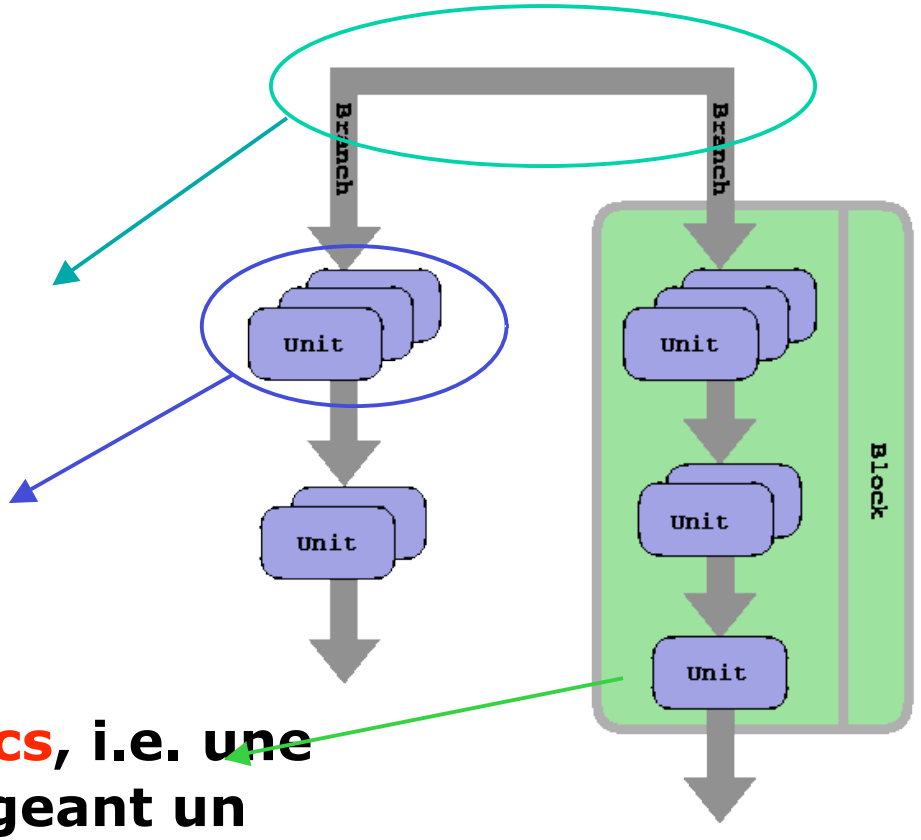
# Les caractéristiques de PALM

- **Flexibilité**
  - Dynamicité du couplage
  - Assure la modularité des applications
- **Performances**
  - **Parallélisme des applications**

## Deux niveaux de parallélisme

**Séquences : branches**

**Composants distribués : unités**

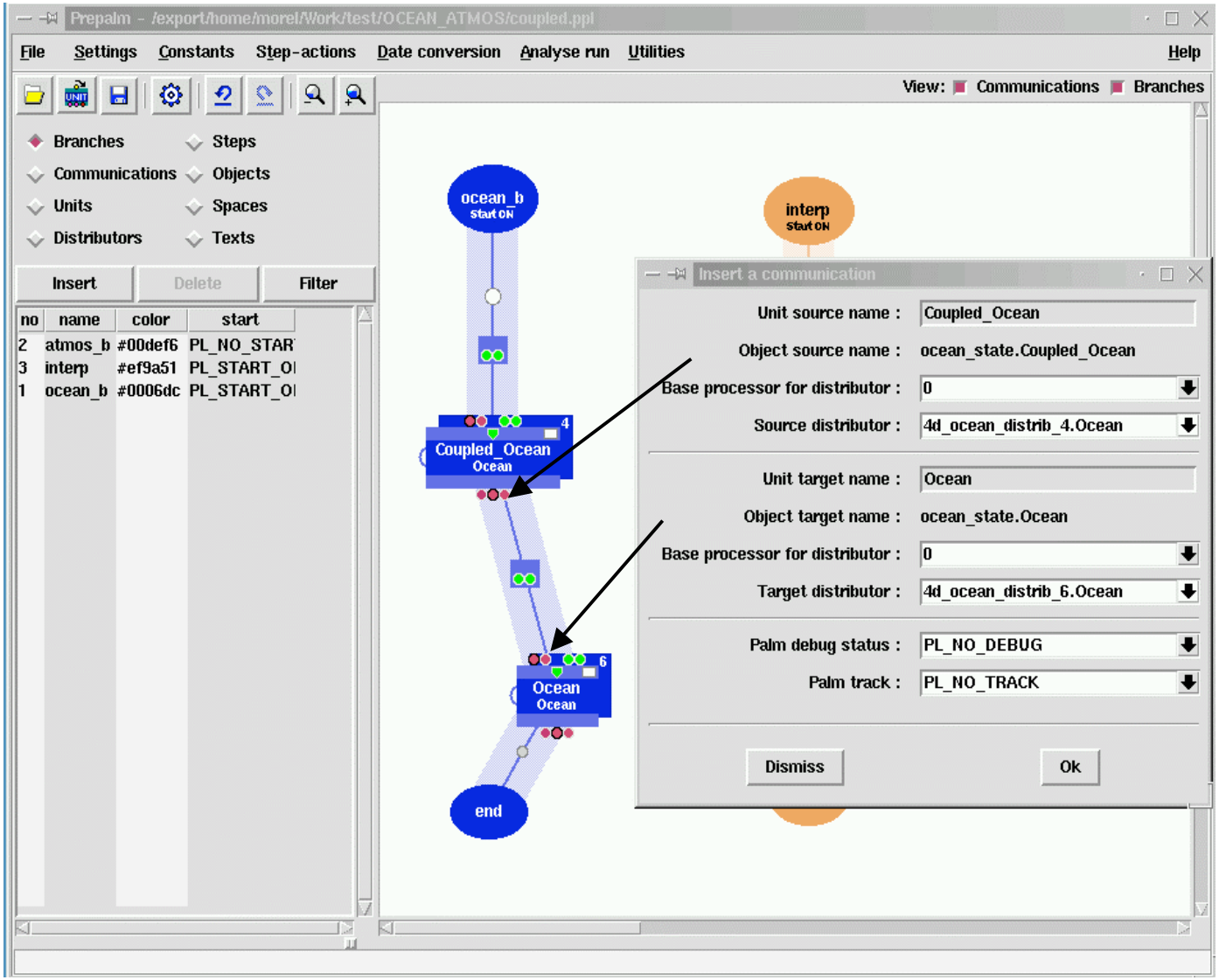



**Optimisation : les blocs, i.e. une séquence d'unités partageant un espace mémoire commun dans une enveloppe SPMD.**

# Les caractéristiques de PALM

- **Flexibilité**
  - Dynamicité du couplage
  - Assure la modularité des applications
- **Performances**
  - Parallélisme des applications
  - **Communications Hautes performances**

# Définir le schéma de communication



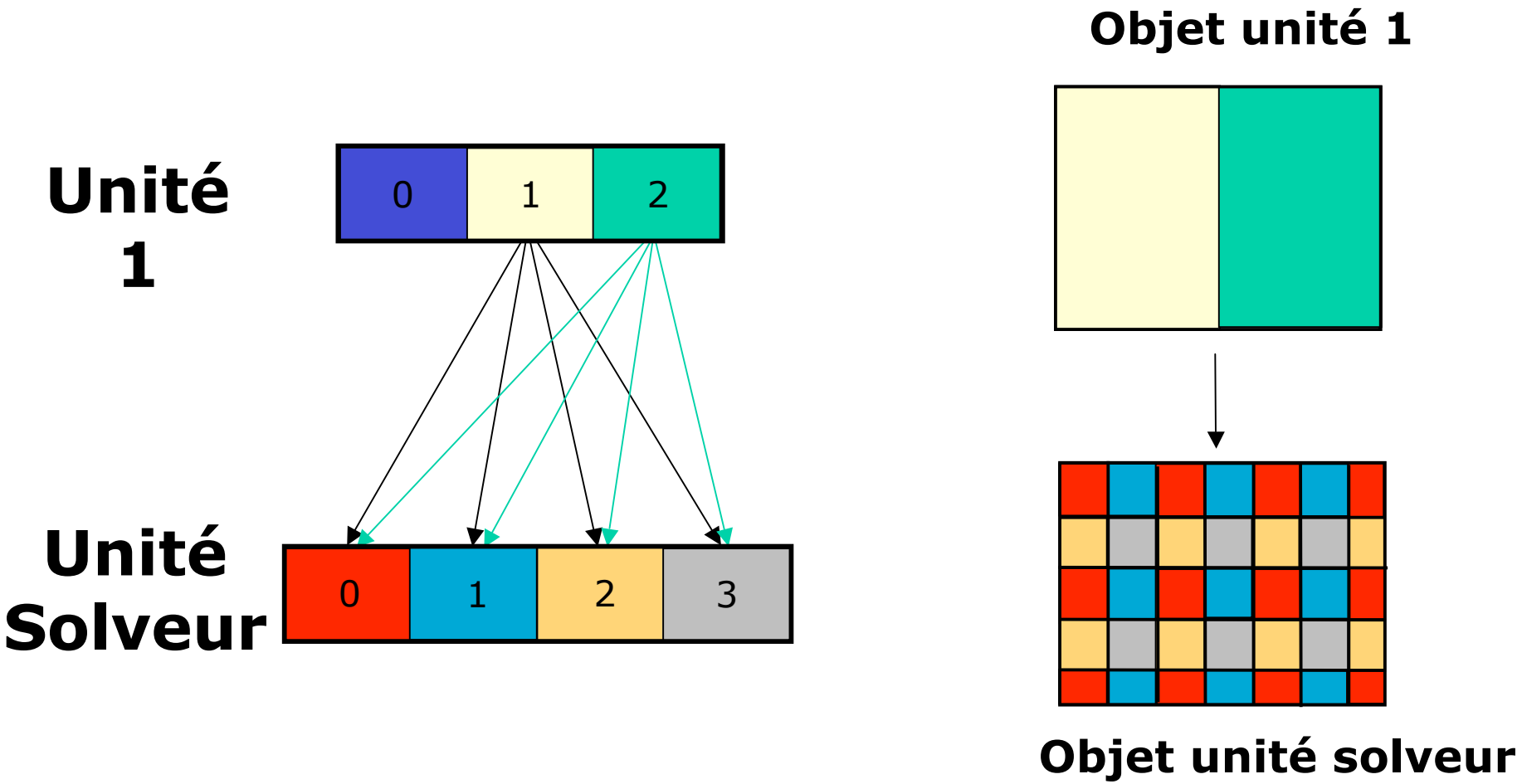
# Distributor PALM

- Description **statique** ou **dynamique** de la **distribution d'un objet global** sur un ensemble de processus
- Chaque processus possède la partie locale de l'objet distribué
- **Description statique**: 4 modes de description fournis par PALM
  - Single_proc,regular,regular avec « ghosts points », Custom
- **Description dynamique**: fonction de distribution fournie par l'utilisateur
- La distribution de l'objet source peut être différente de celle de l'objet cible ➔ **remapping**

# Localisation d'un objet PALM

- ☑ Permet de décrire l'ensemble des processus sur lesquels est **distribué** un objet
- ☑ Permet de gérer des **objets répliqués**
- ☑ Définie dans la carte d'identité de l'unité
  - ✓ **Statique**: trois localisations par défaut fournies par PALM
    - DISTRIBUTED_ON_ALL_PROCS
    - REPLICATED_ON_ALL_PROC
    - SINGLE_ON_FIRST_PROC
  - ✓ **Dynamique**: appel à une fonction

# Le remapping



# Actions  dirigées par des événements

- Synchronisation et déclanchement d'actions sur les données d'un buffer
- Primitives PALM permettent de déclarer ses événements dans Prepalm uniquement. Appel de la primitive dans une branche de code.
- Définition d'actions:
  - Action prédéfinie: nullify, delete, ready, unready
  - Utilisation du language Steplang
  - …

# La norme CORBA

- La **norme CORBA** offre une infrastructure globale pour répondre à l'ensemble des besoins **d'interopérabilté**
- Les modules logiciels utilisés pour le couplage sont:
  - Intégrés dans le code des modèles à coupler
  - Connectables dynamiquement à une ossature appelée « bus logiciel », en les encapsulant dans des **interfaces standardisées** qui permettent la communication avec les différents modèles par le bus logiciel

# CORBA quand?

**Applications réparties** composée de modules pouvant être écrits dans **différents langages** et pouvant fonctionner **sur des plates-formes hétérogènes**

➔Technologies permettant les communications et le fonctionnement des organes de couplages

# CORBA offre

- ☑ Infrastructure globale pour répondre aux besoins d'**intéropérabilité**, de **modularité** et d'**extensibilité**
- ☑ Les modules logiciels utilisés pour le couplage sont **connectables dynamiquement** à une ossature – **bus logiciel** – en les encapsulant dans des **interfaces standards**
- ☑ Les interfaces standards sont réalisés en utilisant le **langage IDL** (Interface Definition Langage)

➔les modèles **échanges des données** à travers les interfaces standardisées

➔ **Avantage**: faire fonctionner de manière uniforme des composants hétérogènes sur des matériels hétérogènes

# Points clés

- ☑ Interopérabilité
- ☑ Approche objet
- ☑ Architecture client/serveur

# Interopérabilité

**Indépendance** vis-à-vis:

- ✓ Des **plates-formes**
- ✓ Des **couches réseau**
- ✓ Des **systèmes d'exploitation**
- ✓ Du **langage de programmation**
- ✓ Des **formats de données**

# Approche objet

Réduit les phases de développement et de maintenance des logiciels

- **Modularité**: un **objet** comporte des attributs et des méthodes. Facilite le regroupement de caractéristiques communes par classes
- **Encapsulation**: utilisation de l'objet sans en connaître la structure interne. Les attributs de l'objet seront protégés et ne seront accessibles que par les méthodes publiques
- **Réutilisation et extensibilité** :la modularité et l'encapsulation permettent la réutilisation d'objets, facilité par les mécanismes d'héritage et de polymorphisme

# Client/serveur

**Architecture client/serveur**: un objet (objet serveur) offre des services à des applications clientes distantes ou à d'autre objets clients.

- ✓ L'objet serveur possède l'intégralité des caractéristiques de l'objet
- ✓ Le client accède à ces services grâce aux méthodes de l'objet serveur

➔ **Vers un bus logiciel** où différents modules interagissent via un bus et des interfaces (modularite, extensibilité)