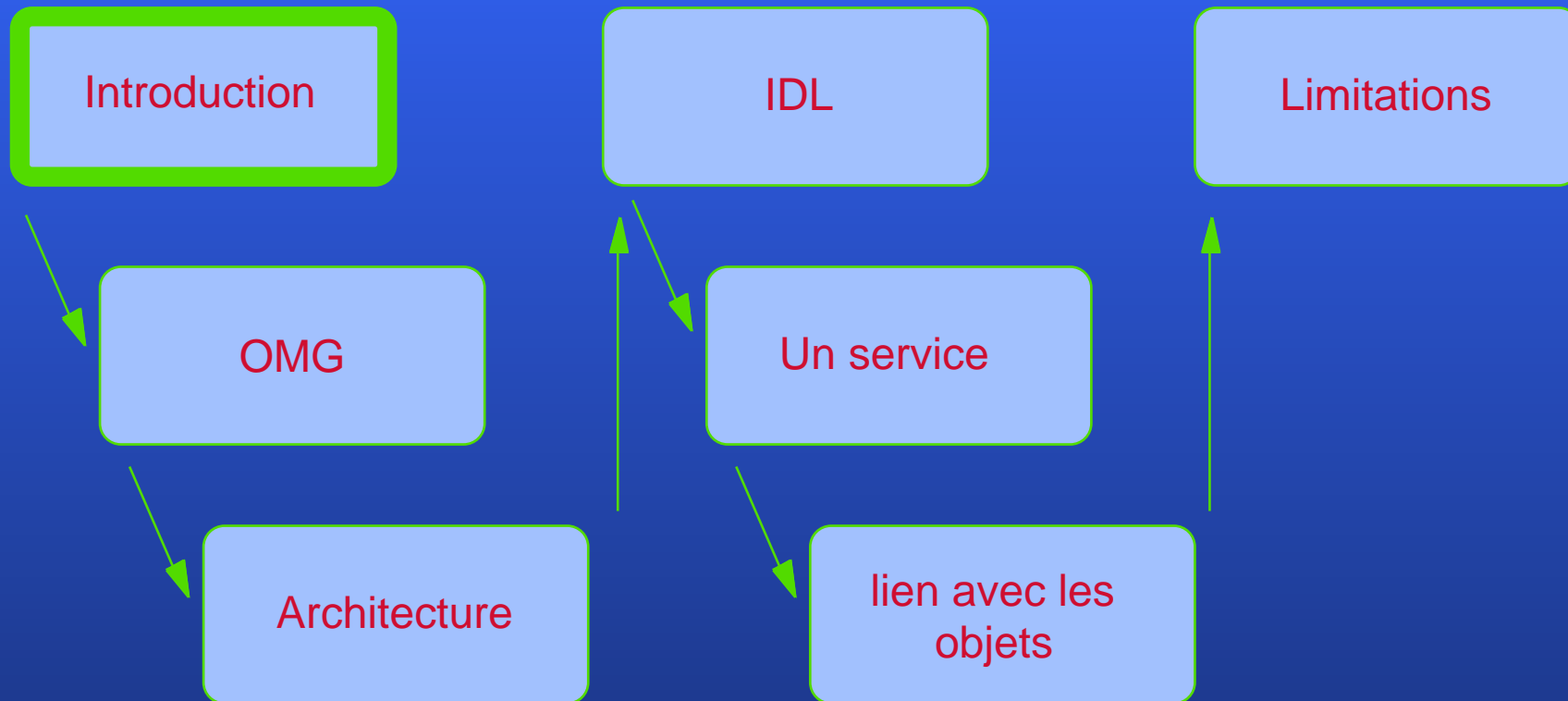


# Corba

- Architecture
- Broker
- Request
- Object
- Common



# Plan

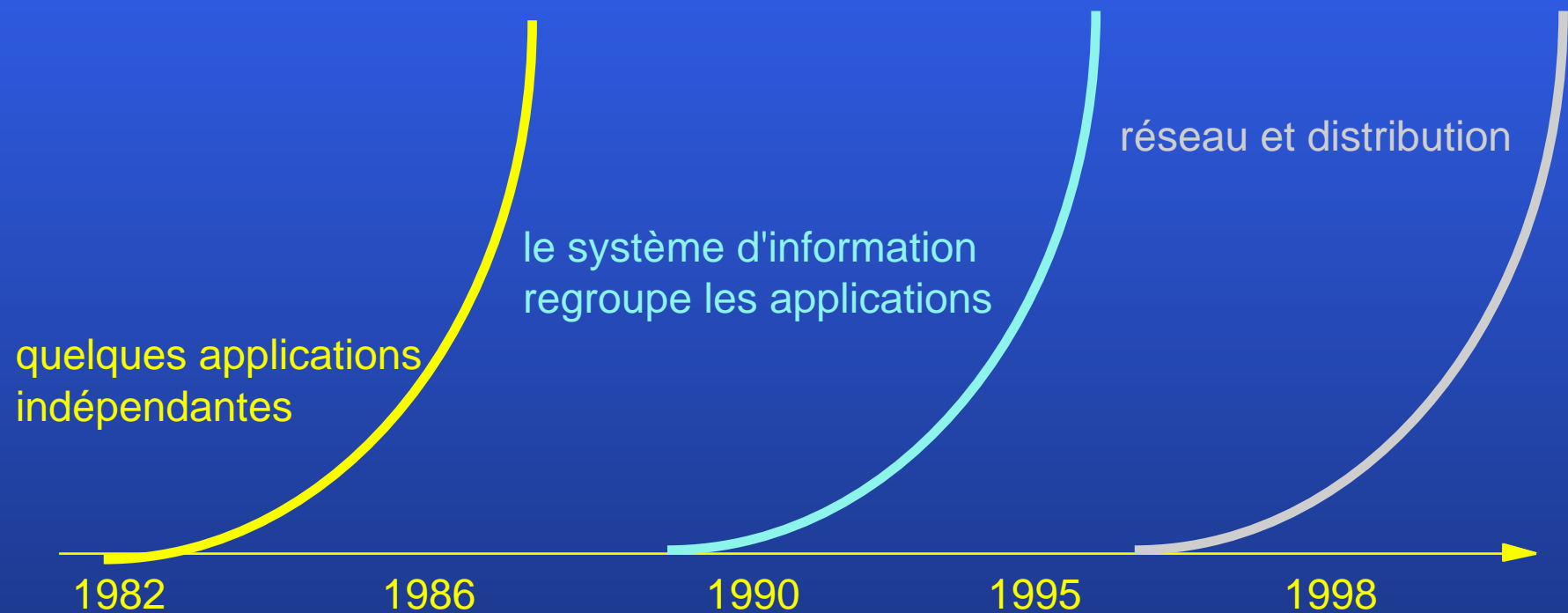




# *Introduction*

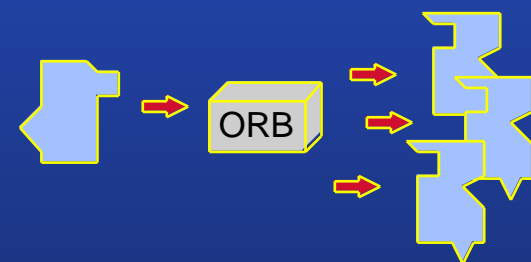
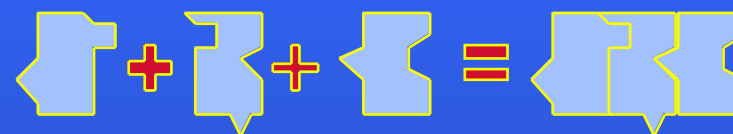
- **Quels besoins pour les applications ?**
- **Des solutions possibles**
- **Corba**

# Evolution des applications

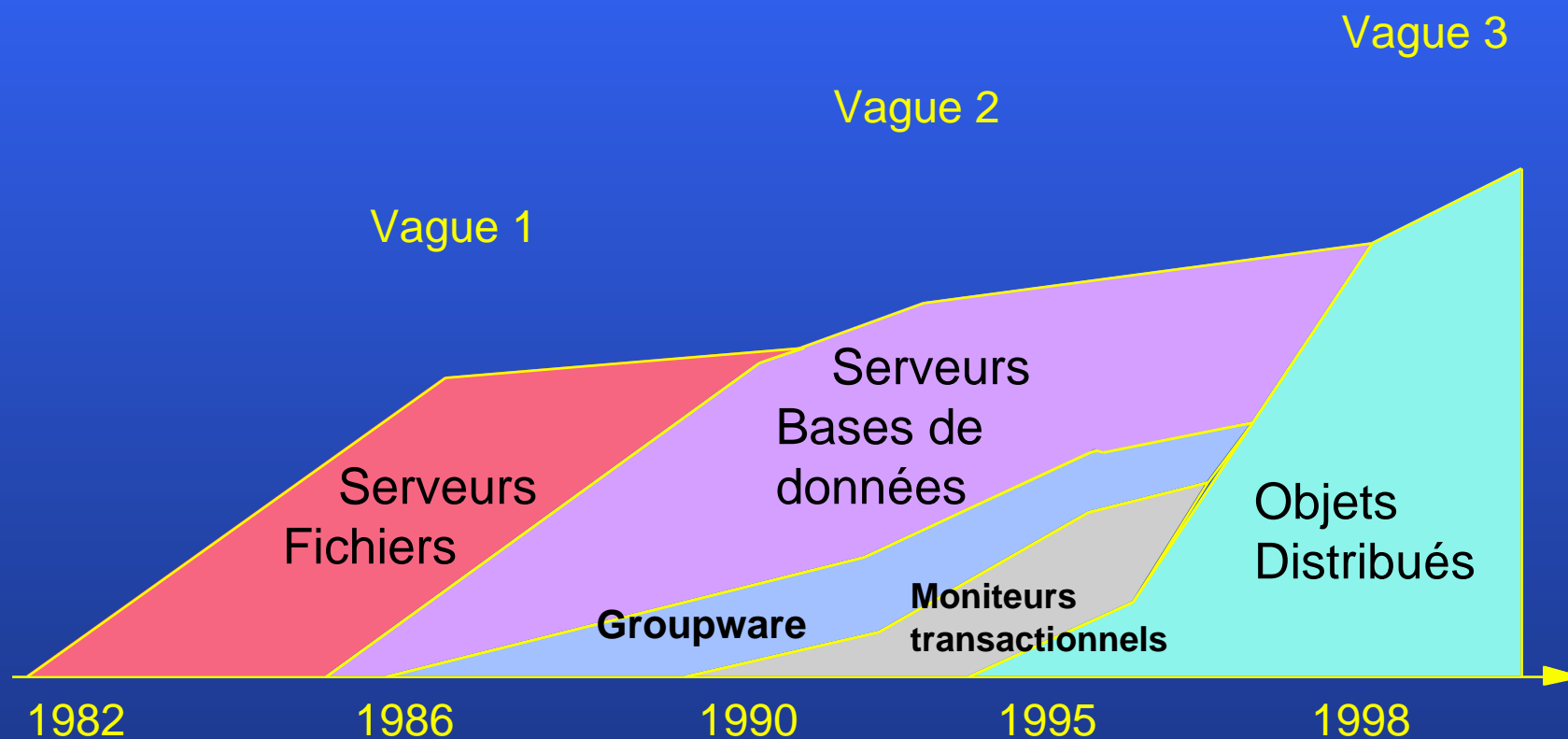


## Quels besoins pour les applications ?

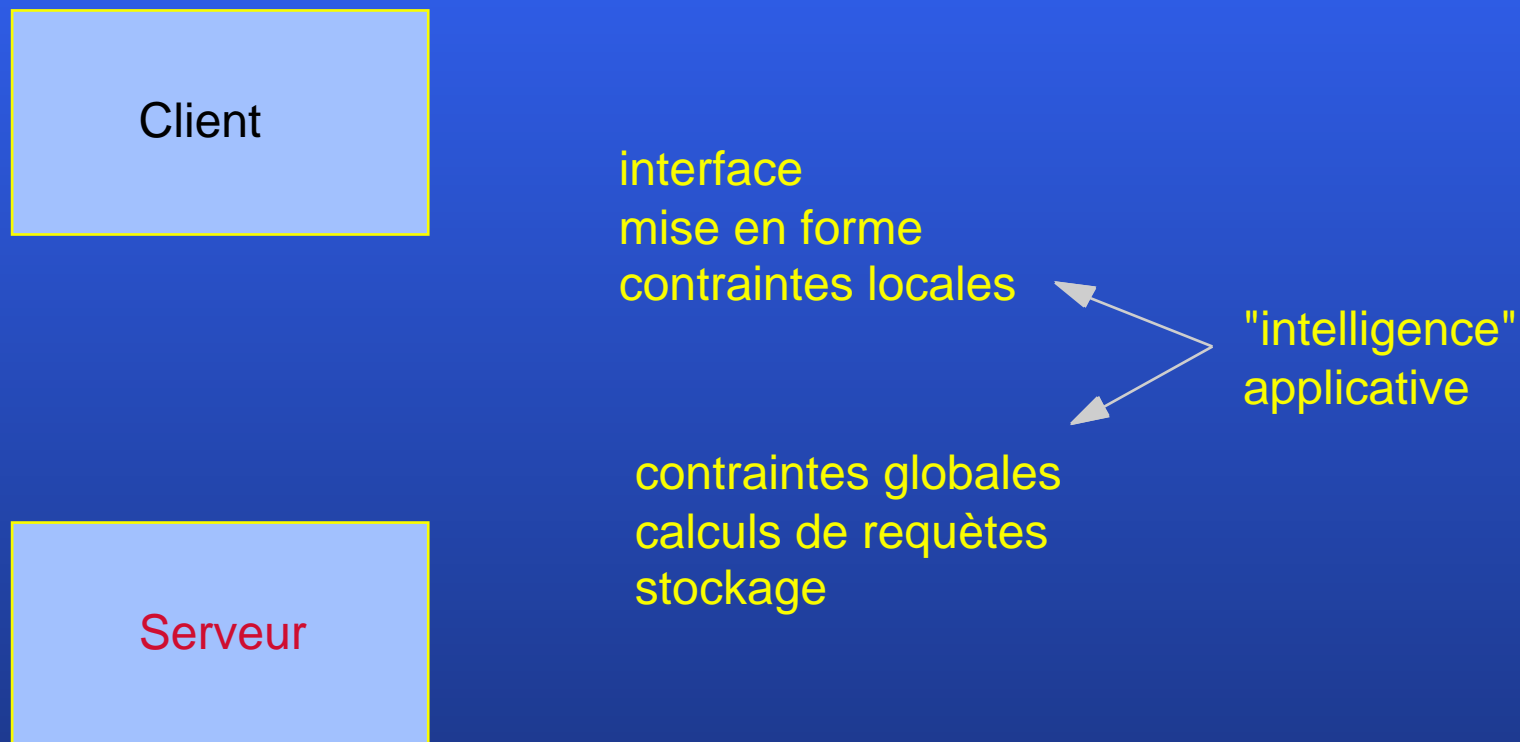
- Plug-and-Play
- Interopérable
- Portable
- Coexistence
- Transparence à la localité



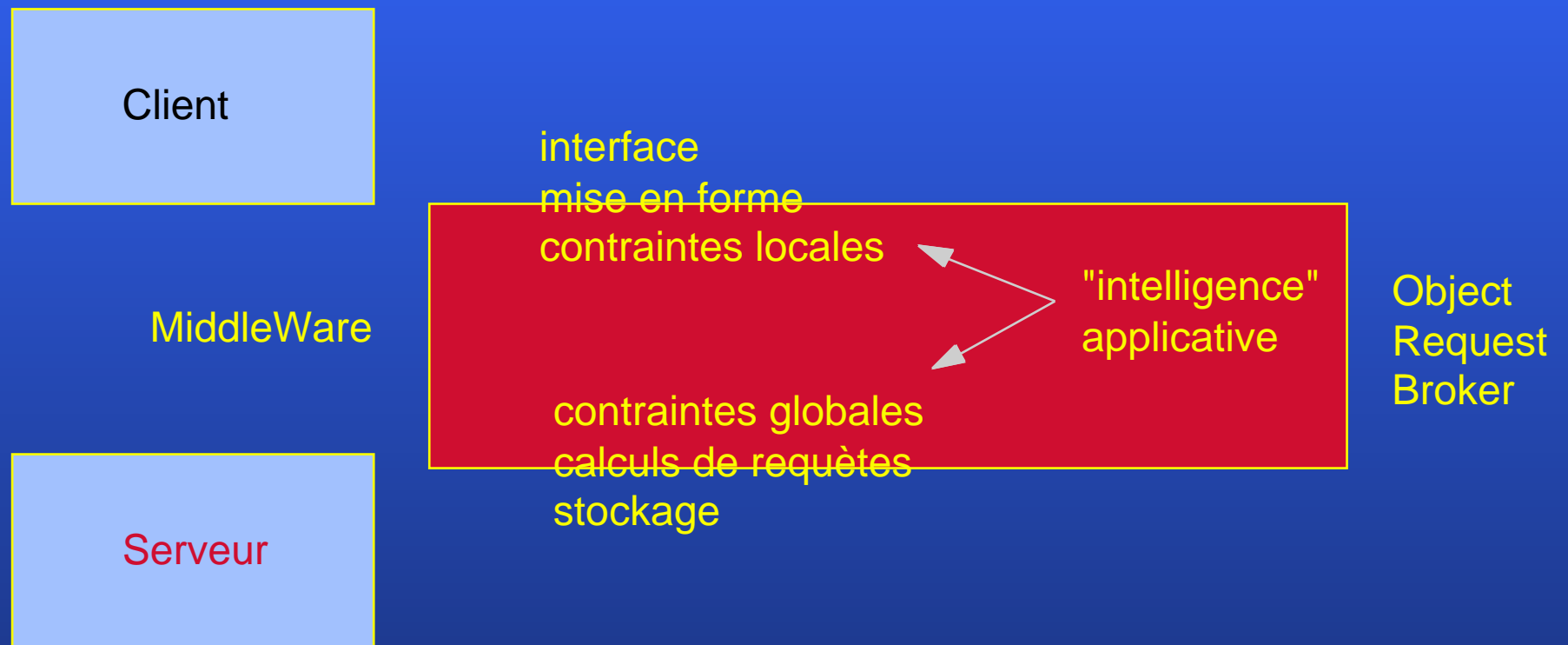
# Des solutions possibles



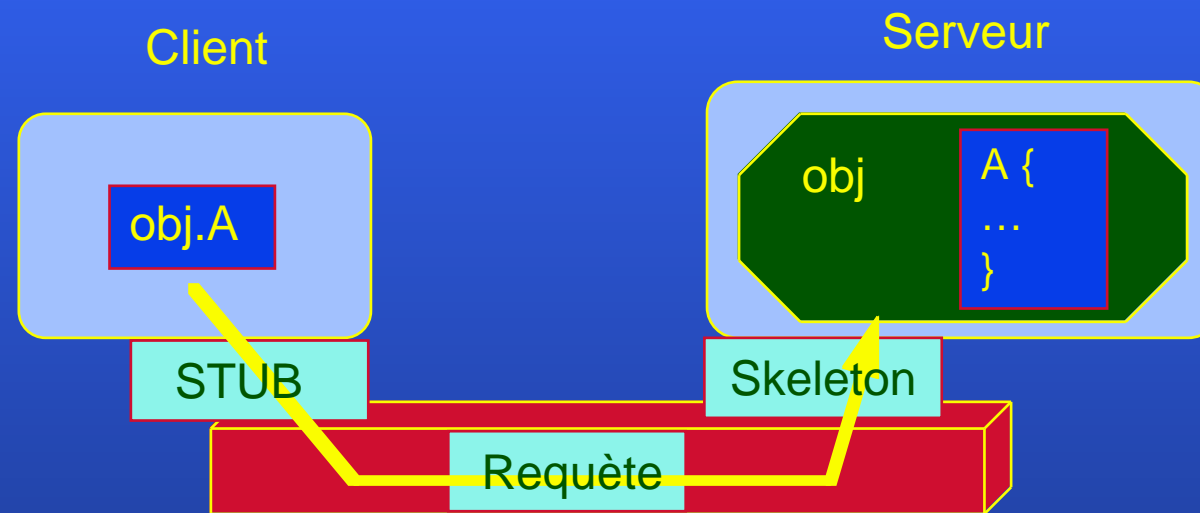
## Modèle 2 couches (two-tier)



## Modèle 3 couches (three-tier)

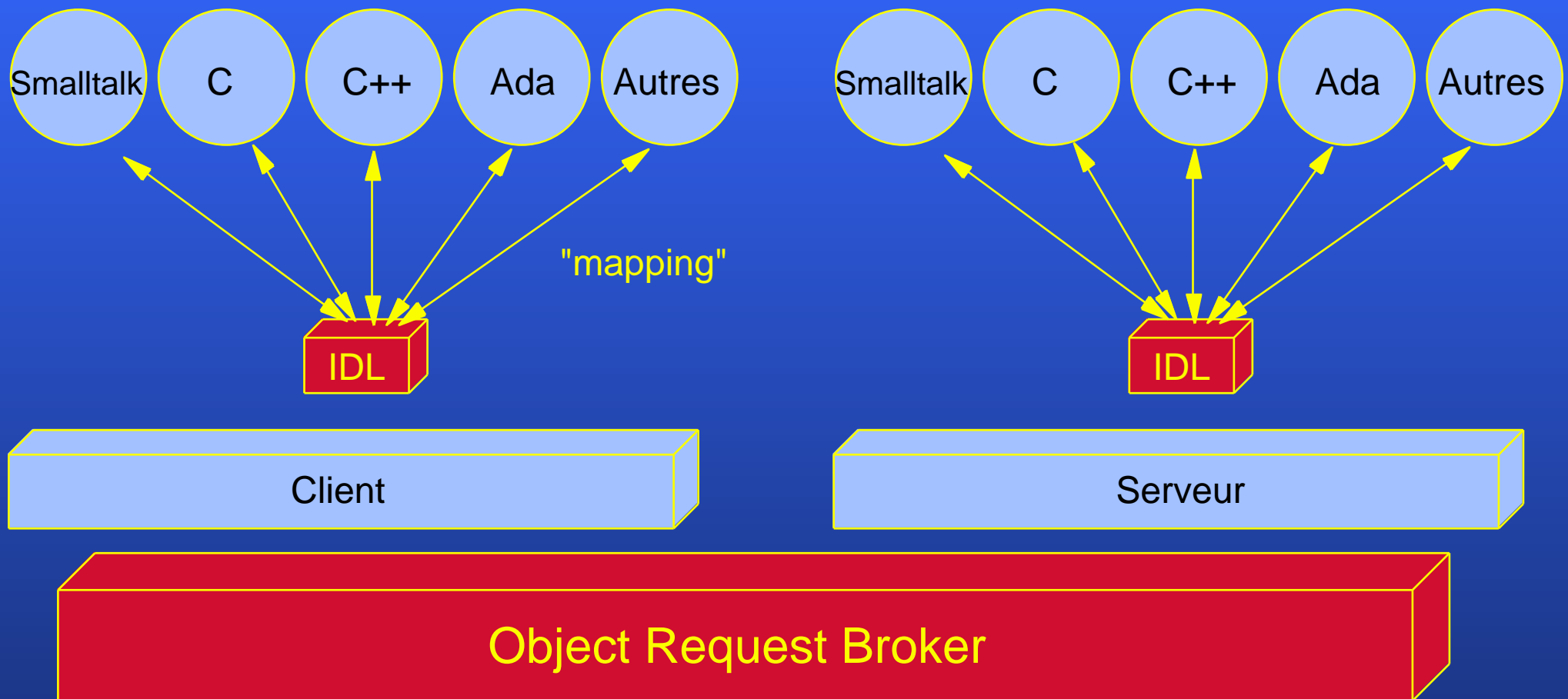


# Le principe d'un ORB

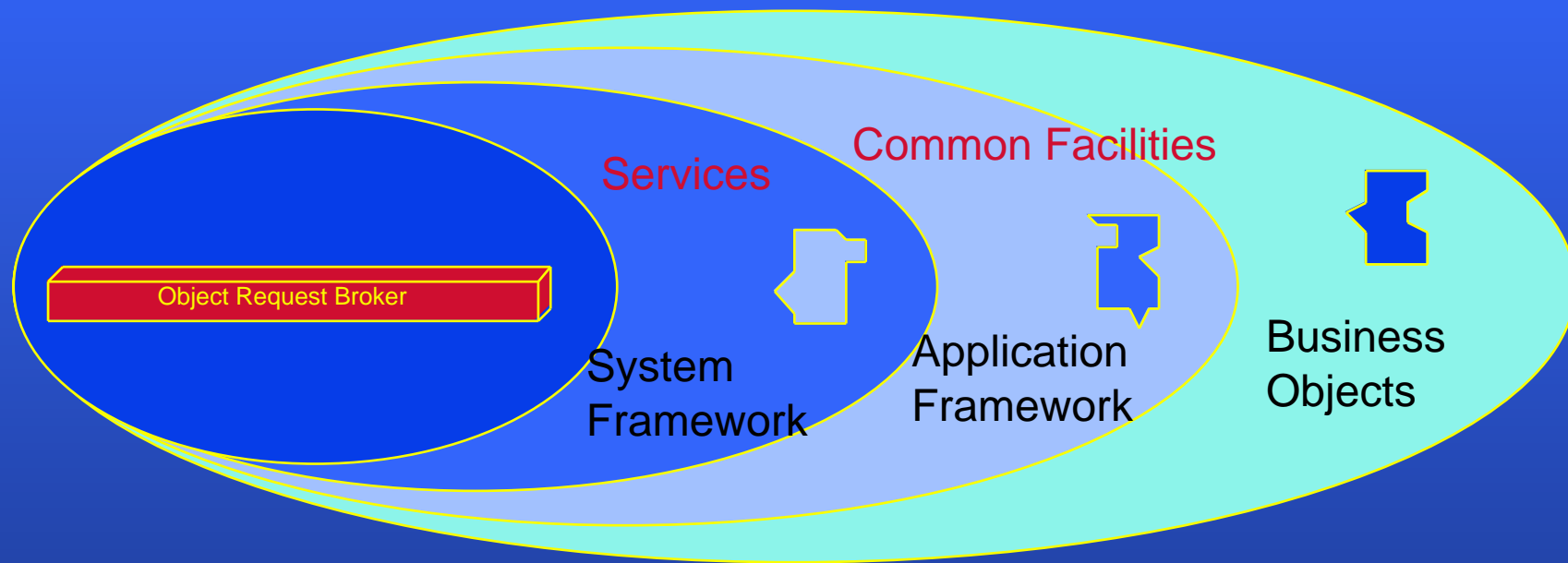


Symétrie Client-Serveur ; un objet peut être à la fois client d'objets et serveur d'autres

## Corba : le noyau



# Corba : vision globale



# Corba : les composants "service" 1/3

- Life Cycle
  - création, copie, déplacement et destruction d'objets
- Relationship
  - définition de relation entre objet et de techniques de parcours
- Naming
  - comment nommer, enregistrer et localiser un objet
- Persistence
  - interface pour stocker des objets sur SGBD, SGBDR, SGBDO,  
...
- Externalization
  - comment faire entrer ou sortir des objets sous forme de flux

## *Corba : les composants "service" 2/3*

- **Event**
  - signalisation d'événements entre objets via des canaux
- **Query**
  - surensemble de SQL (SQL3) et de OQL
- **Properties**
  - association d'informations aux objets (titre, date de création, ...)
- **Transaction**
  - validation deux-phases de transactions (commit, rollback)
- **Concurrency Control**
  - gestion de verrou pour la synchronisation

## *Corba : les composants "service" 3/3*

- Licensing
  - comment contrôler l'usage des objets (par session, par instance créée, par site, ...)
- Et d'autres à venir...
  - Trader
  - Security
  - Time
  - Collections
  - Change Management

# Corba : les composants

- Common facilities
  - horizontal
    - » User Interface
    - » Information Management
    - » System Management
    - » Task Management
  - vertical
    - » health, telcos, finance
- Business Object and Application

*Document  
Composition  
Edition  
Stockage*

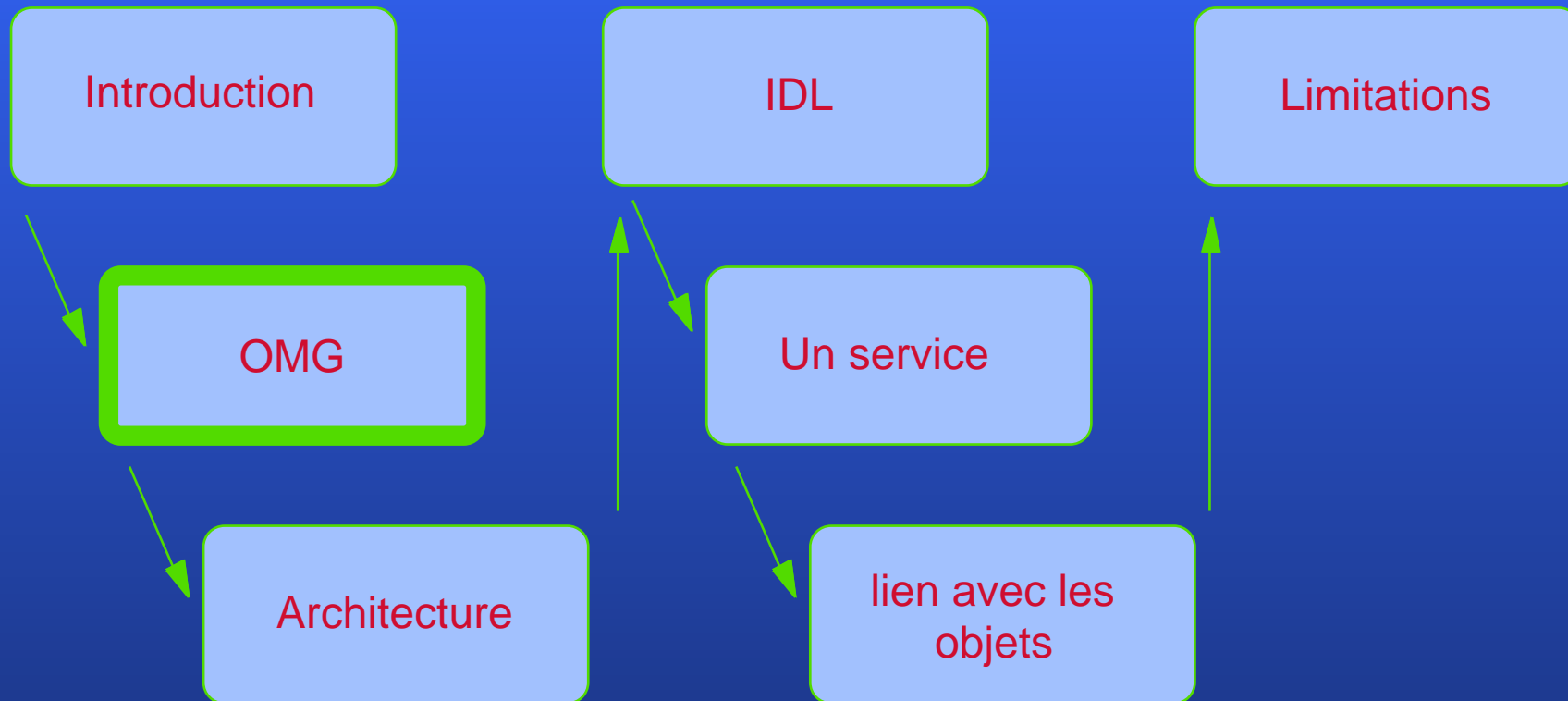
...  
*Configuration  
Gestion*

...  
*Workflow*

...



# Plan





# *Organisation*

- **Histoire**
- **Road map**

## *Histoire*

- Créée en 1989 par Sun, HP, Dec, NCR Hyperkesk, ODI
- Association type "loi 1901" basée aux USA
- Actuellement, plus de 600 membres
  - le plus grand groupe d'utilisateurs
  - 15 employés à plein temps
- Dédiée à la spécification et à la diffusion de standards objets
- Ne standardise que des technologies prouvées

# Road Map

RFP 1 : CF1  
OpenDoc

RFP 3 : COSS3  
Security  
Time

RFP 1 : COSS1  
Life Cycle  
Naming  
Persistence  
Event Notification

RFP 2 : COSS2  
Transactions  
Concurrency  
Relationships  
Externalization

RFP 4 : COSS4  
Query  
Licensing  
Properties

RFP 5 : COSS5  
Trader  
Collections  
Change Management

1993

1994

1995

1996

Corba 1.1

Corba 2.0

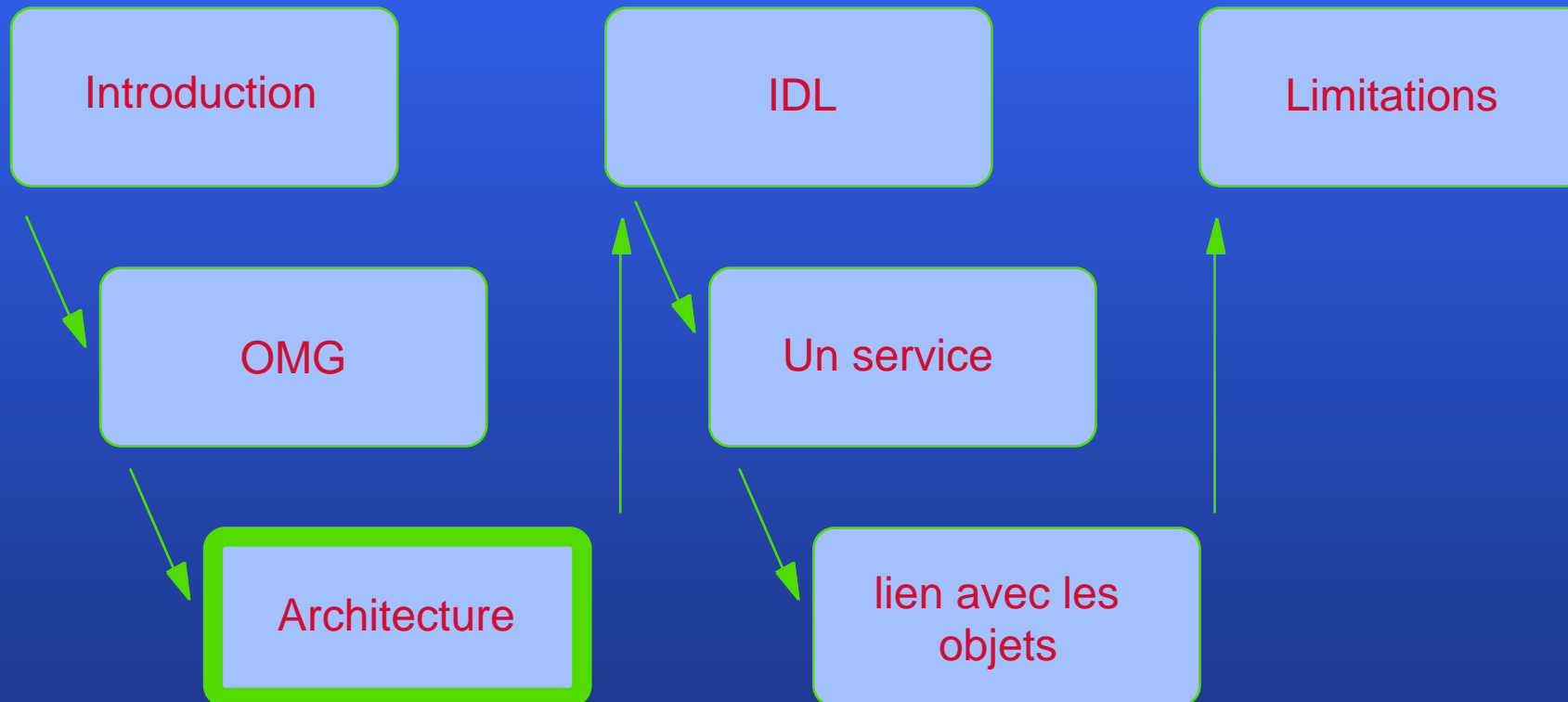
12/94

interopérabilité mono-ORB

interopérabilité multi-ORB



# Plan





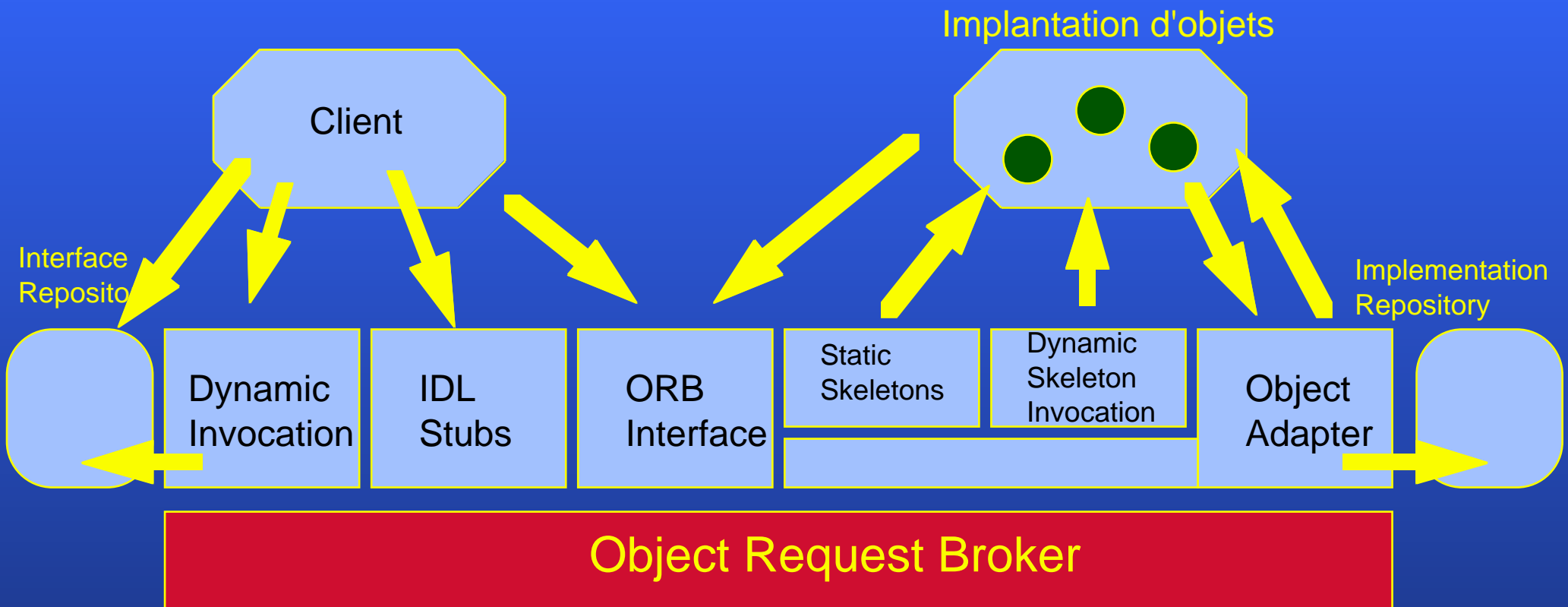
## *Corba 2.0 : le bus objet*

- 
- **Architecture**
  - **Utilisation**
  - **IIOP**

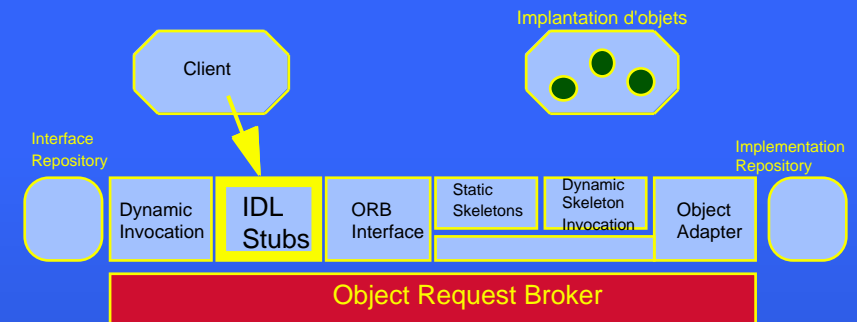
## *Architecture : le rôle de l'ORB*

- Administration d'objets : enregistrement d'objets, de services, descriptions d'objets (méta-données)
- Communication objet : Un objet client invoque une méthode sur un autre objet (distant, autre langage, autre plate-forme, etc)
  - transparence
  - symétrie
- 2 APIs : invocation statique ou dynamique

# Architecture générale

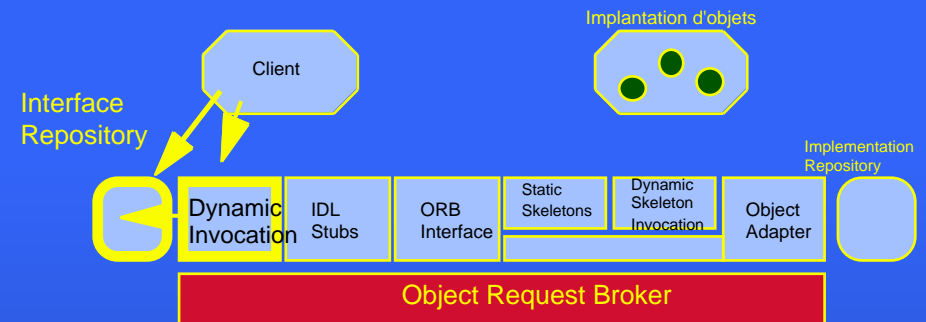


# IDL Stubs (talons)



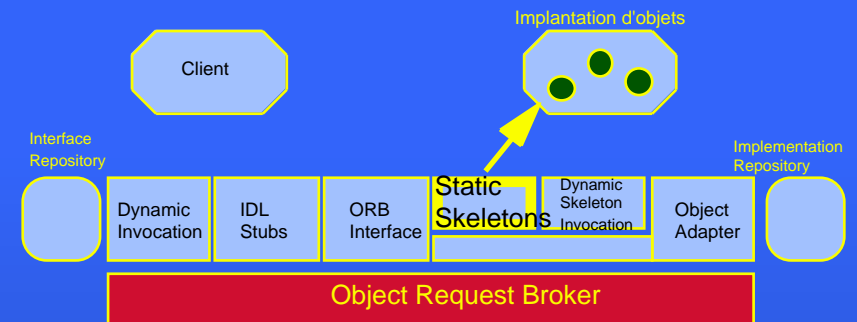
- Précompilés, ils définissent comment le client peut invoquer un service d'un objet serveur.
- Généré à partir de la description IDL de l'objet serveur.
- Agit comme un proxy.
- Inclus le code de marshalling (mise à plat standard) des messages envoyés au serveur.

# Dynamic Invocation Interface et Interface Repository



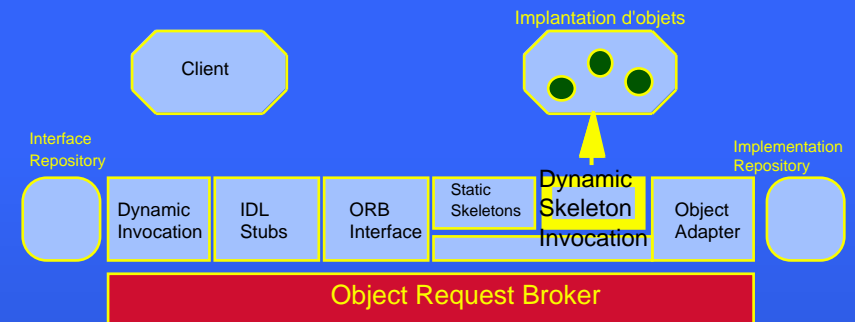
- La DII est utilisée par le client pour accéder aux descriptions des services du serveur (méta-données) afin de choisir à l'exécution l'opération, les paramètres et la nature du résultat
- L'IR enregistre l'ensemble des signatures des opérations accessibles. C'est la base de données des méta-données de l'ORB.

# IDL Skeletons



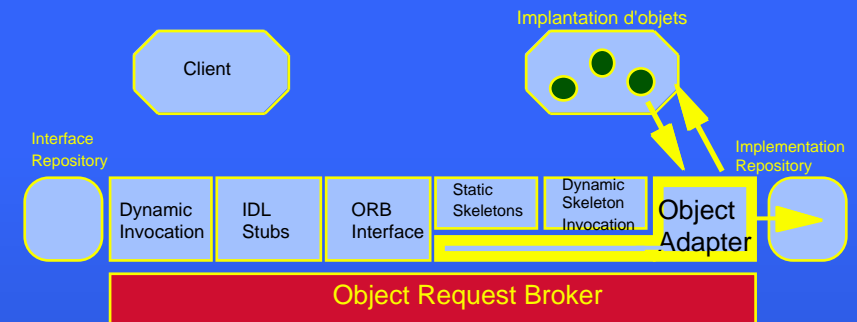
- Fournissent une interface avec les services offerts par le serveur.
- Générés à partir de l' IDL.
- Inclus le code de unmarshalling des messages envoyés par le client.

# Dynamic Skeleton Invocation



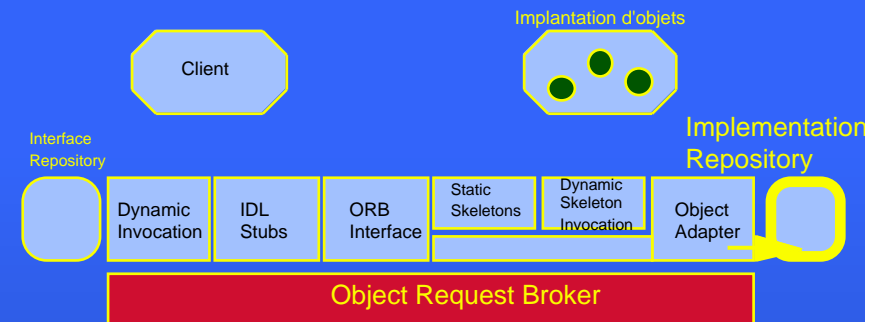
- Nouveauté Corba 2.0
- Pour accéder à des objets qui n'auraient pas de description statique (squelette) coté serveur.
- Permet de générer à l'exécution des implantations d'objets. (utile pour les interpréteurs ou les langages de script)
- Peut traiter des requêtes statiques ou dynamiques.

# Object Adapter



- Fait le lien entre les requêtes émanant des clients et les objets via l'Implementation Repository.
  - instancie les objets, si nécessaire
  - transmet les requêtes

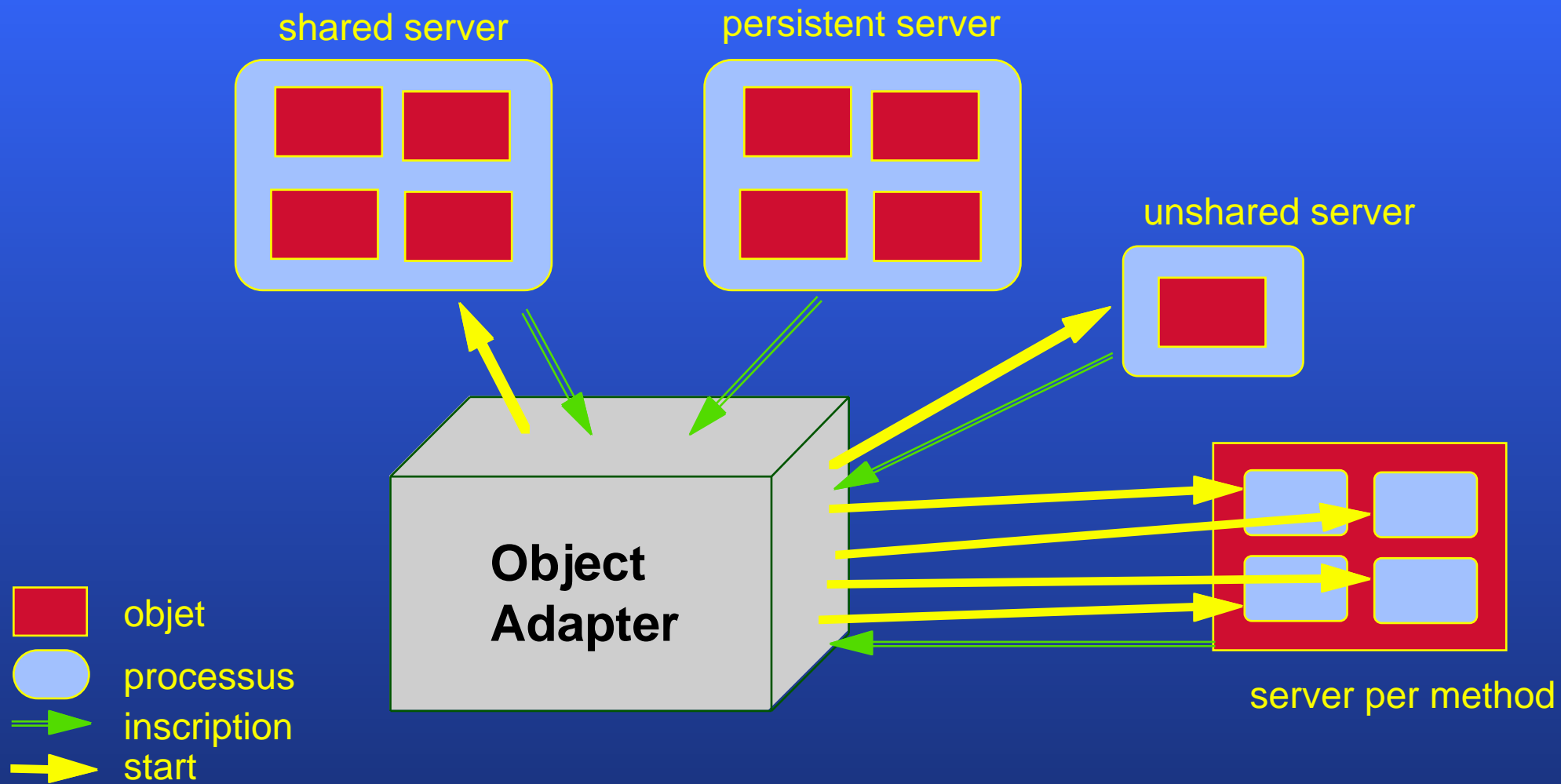
# Implementation Repository



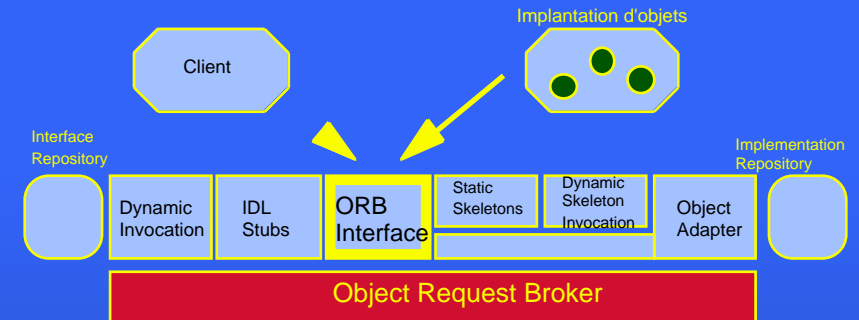
- Base de données concernant les classes offertes, les objets instanciés et leurs IDs.
- Permet d'enregistrer diverses informations comme la sécurité, les accès, et autres données administratives.



# Stratégies des OA



# L'Interface ORB



- Contient quelques services primitifs Corba qui peuvent être utiles :
  - object\_to\_string
  - string\_to\_object

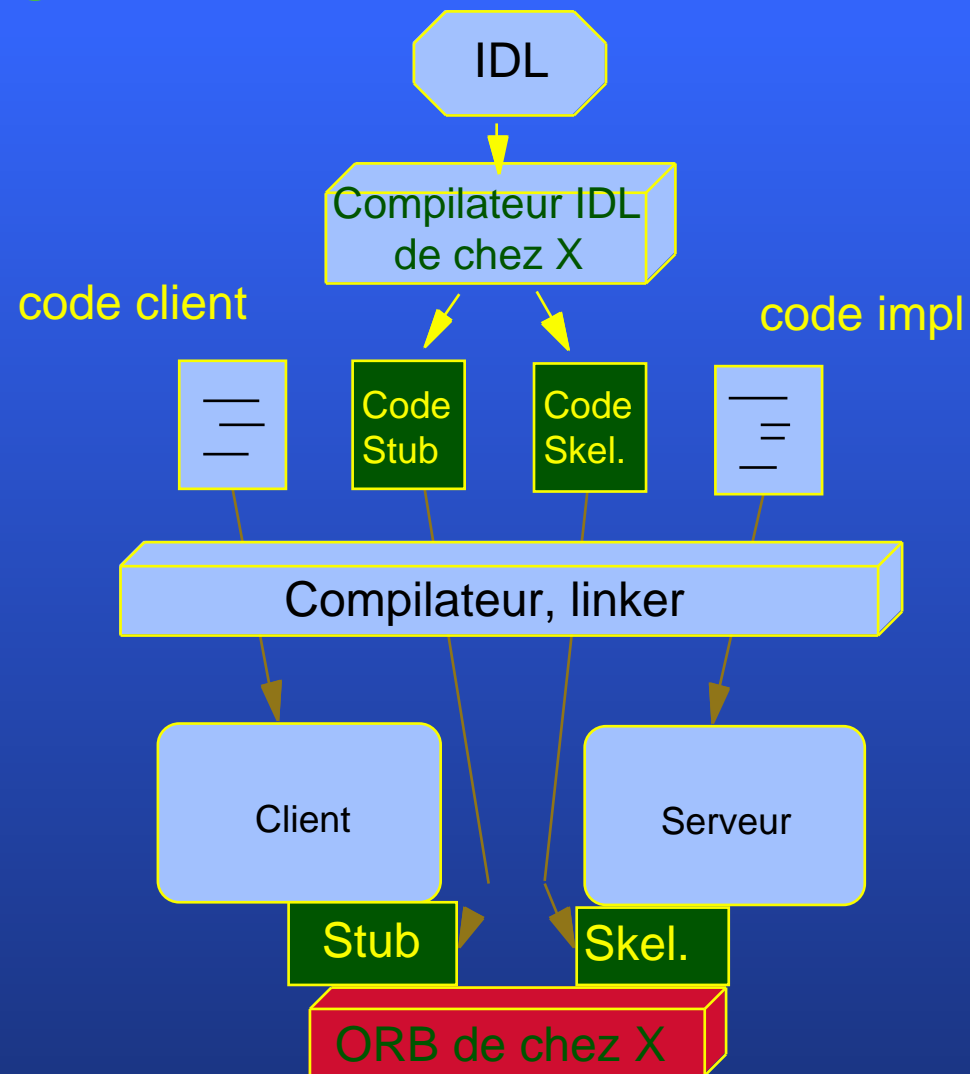
ObjectReference  $\longleftrightarrow$  string



# ObjectReference

- Identificateur unique d'objet
  - dépend de l'implantation de Corba
  - Corba 2.0 spécifie *Interoperable Object References* (IOR)
- Ou trouver des Objects References ?
  - via des opérations envoyées à des objets connus
  - via *string\_to\_object* et *object\_to\_string*
- L'ORB implance pour une Reference
  - *get\_interface* qui retourne les meta-données de cet objet
  - *get\_implementation*

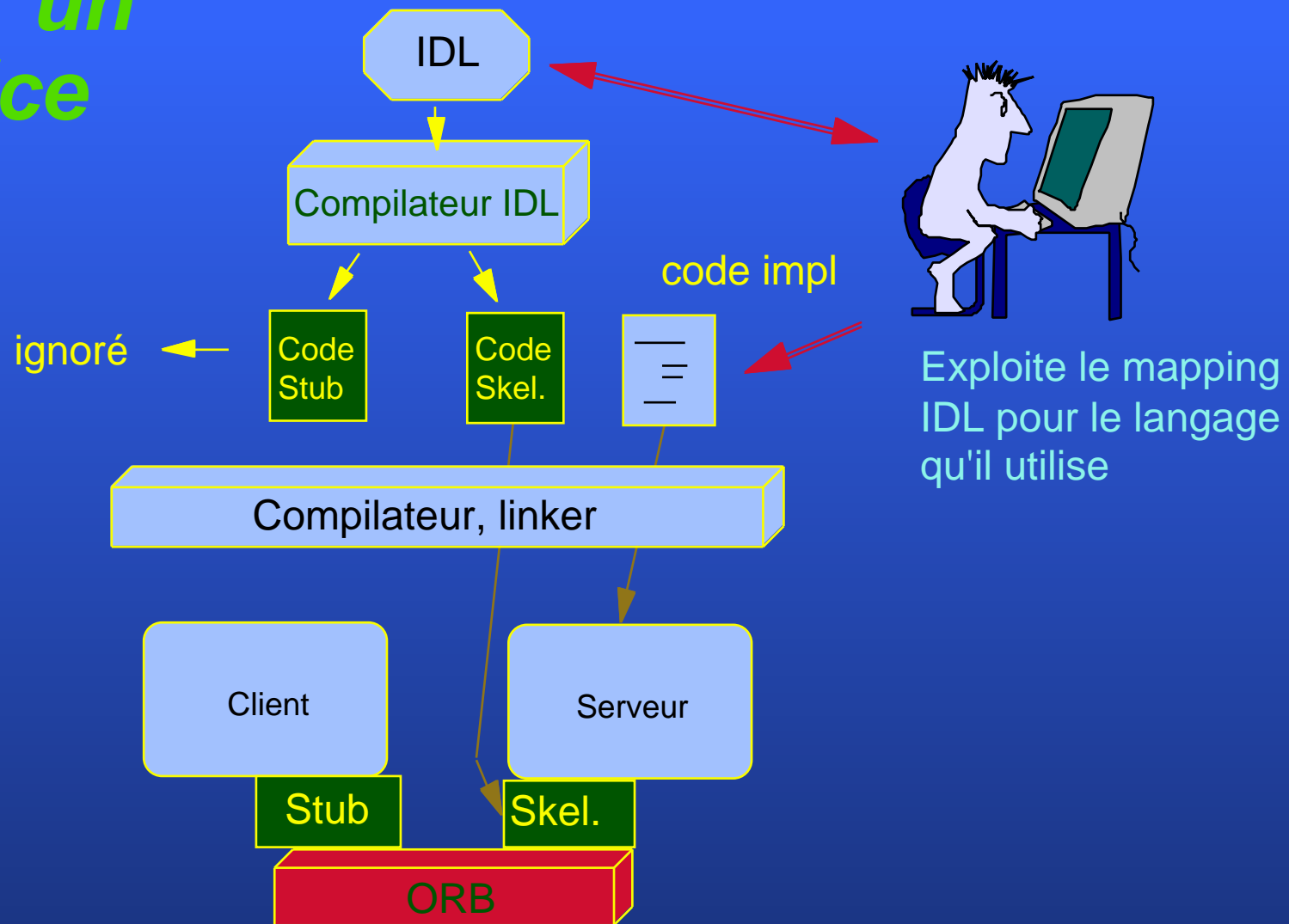
# Connecter via Corba



## *Utilisation : cas statique*

- Définir l'interface en IDL
- Compiler l'IDL
  - génération des talons
  - génération des squelettes
- Ajouter l'implantation aux squelettes
- Compiler l'ensemble
- Enregistrer dans l'IR
- Instancier les objets sur le serveur
- Enregistrer les instances dans l'Implementation Repository

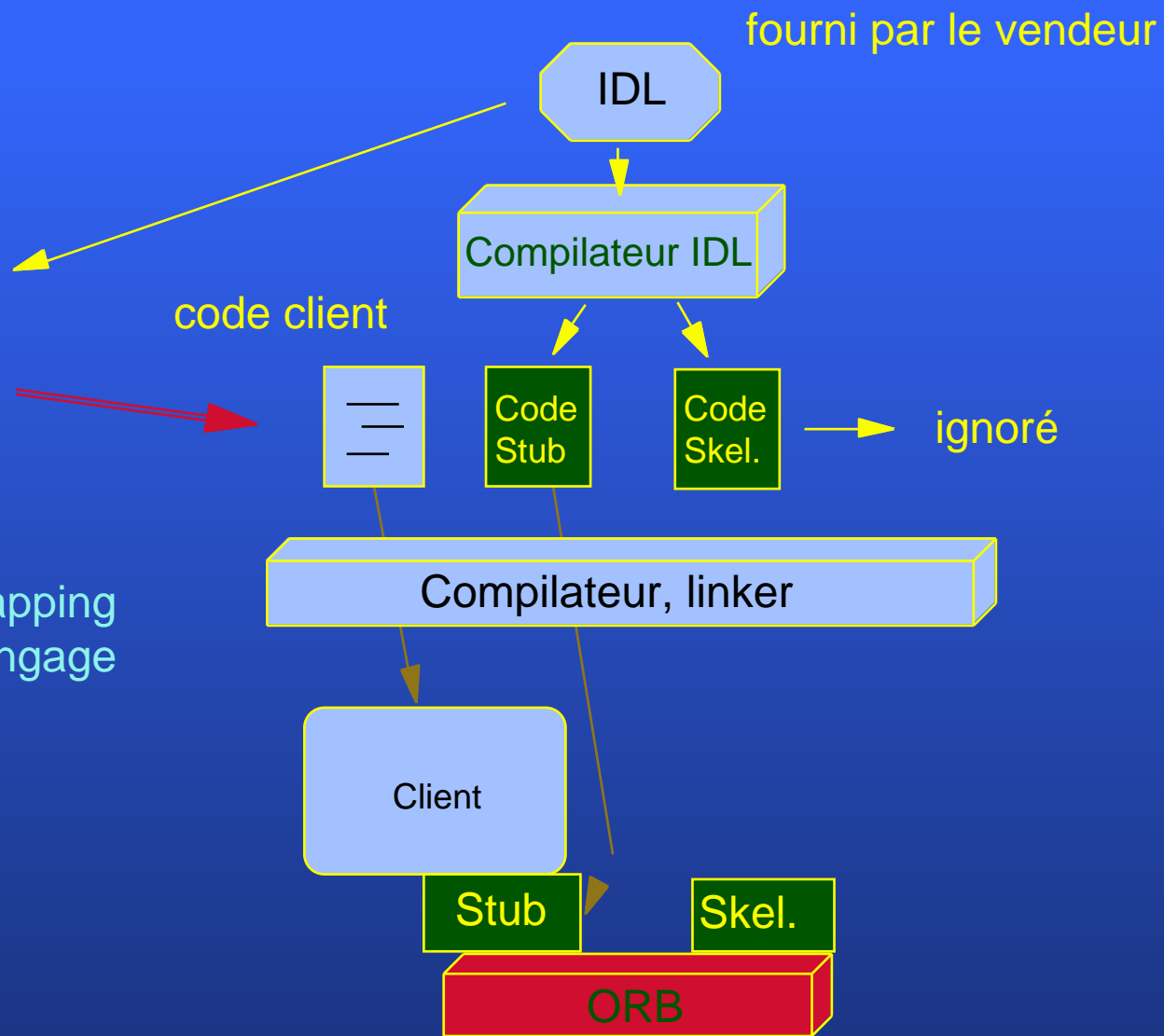
# Offrir un service



# Exploiter un service



Exploite le mapping IDL pour le langage qu'il utilise



## *Trouver le destinataire d'une requête (coté client)*

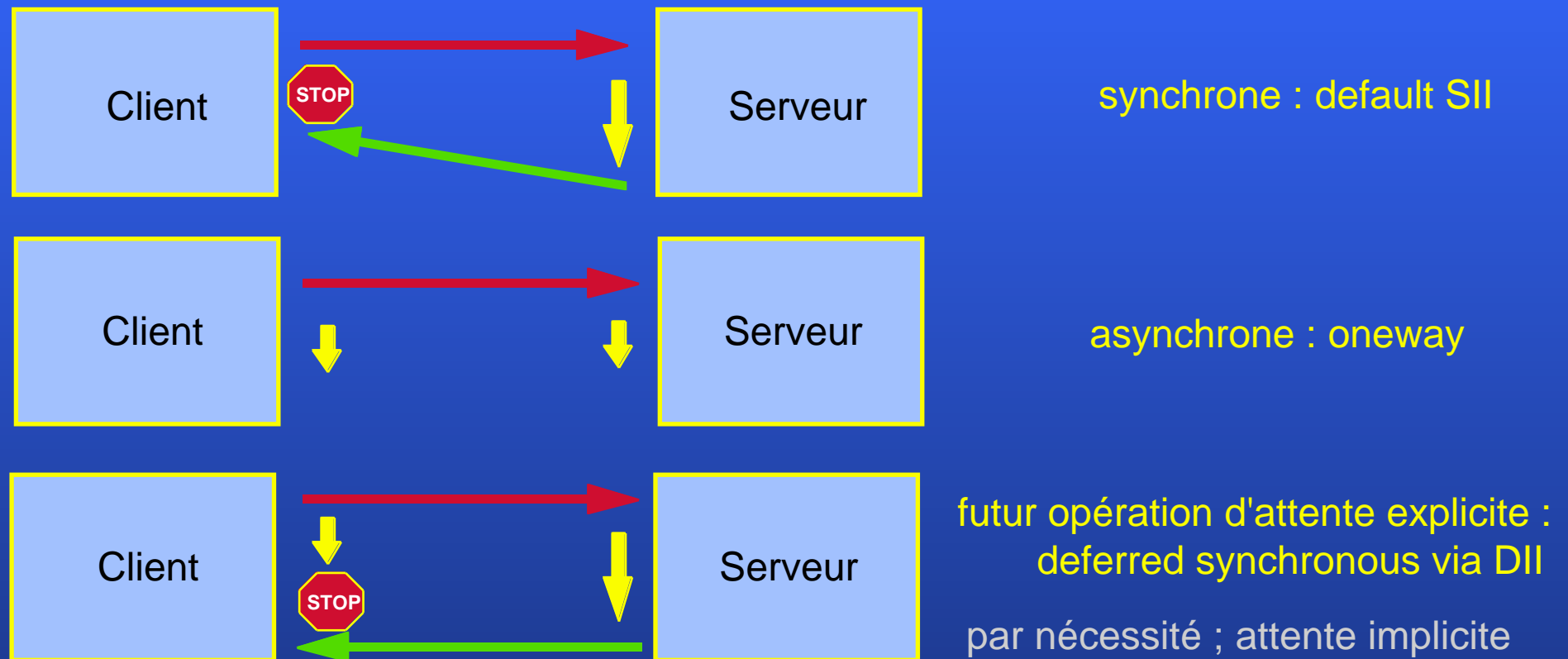
- On connaît son Object Reference
  - via la réponse à une précédente requête
  - via le service de l'ORB `string_to_object`
- On s'adresse à un service de nommage
  - un nom peut-être associé à chaque objet

## *Utilisation : cas dynamique*

- Obtenir la description du service via l'IR
  - `get_interface()`
- Créer la liste d'argument idoine
  - `create_list() ; add_arg() ; ... ; add_arg()`
- Créer la requête
  - `create_request(ObjectReference, Method, ArgumentList)`
- Envoyer la requête
  - `invoke()`
  - `send() ; get_response()`
  - `send() "one way"`



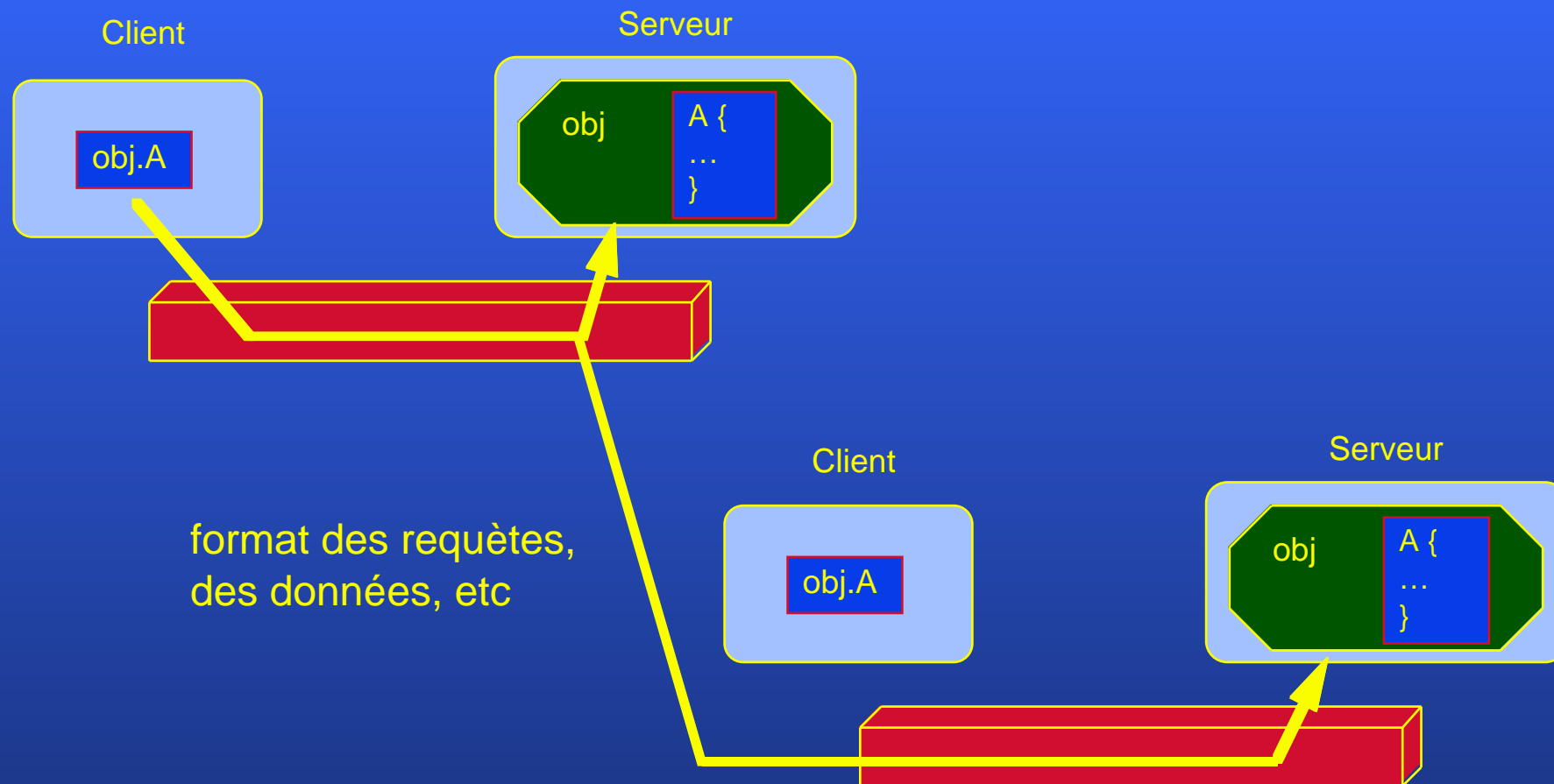
# Invocations



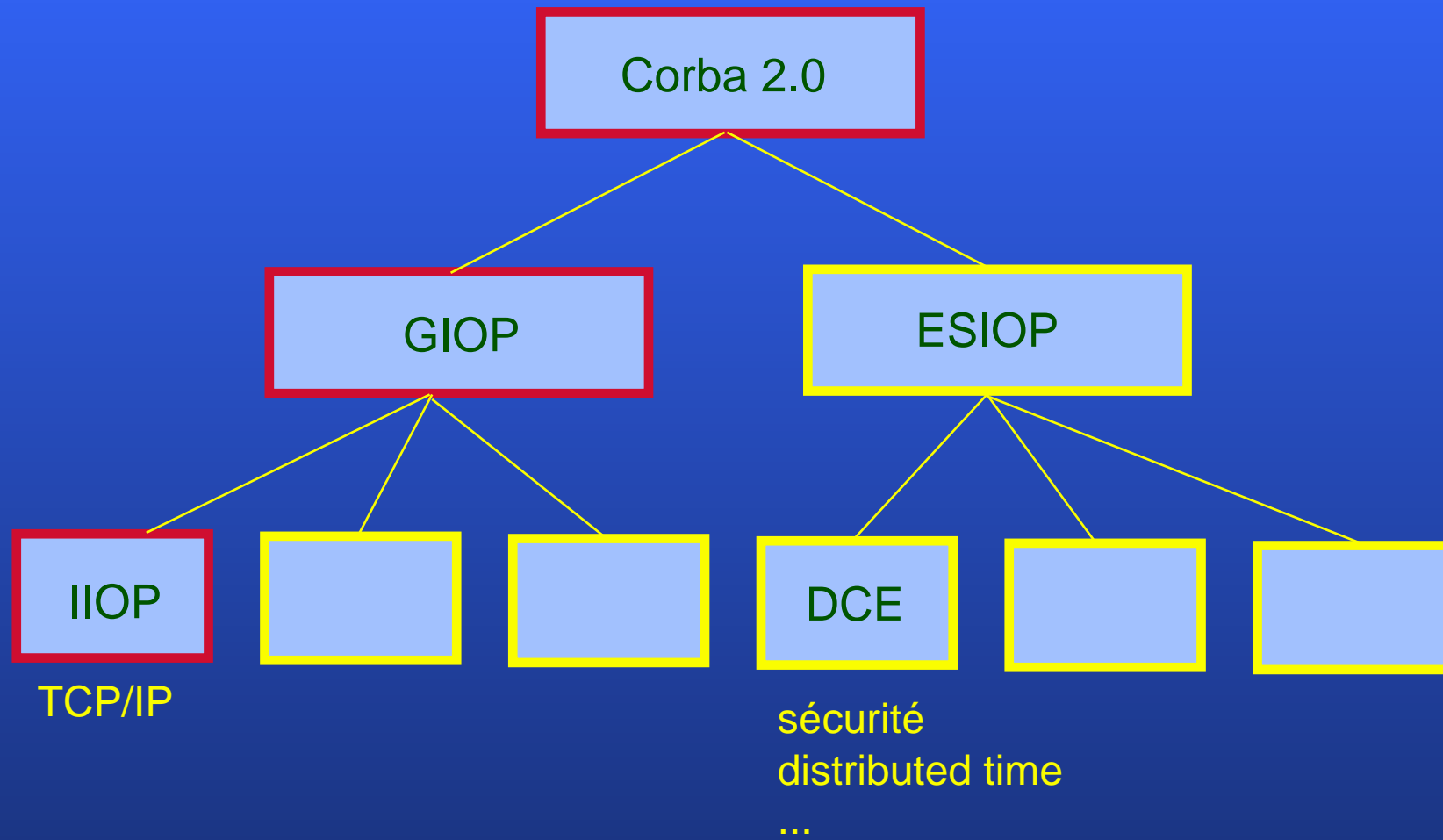
## *Trouver le destinataire d'une requête (coté serveur)*

- C'est le rôle de Object Adapter
- Recherche dans l'Implementation Repository
  - association (Object Reference objet)

# Communication inter-ORB

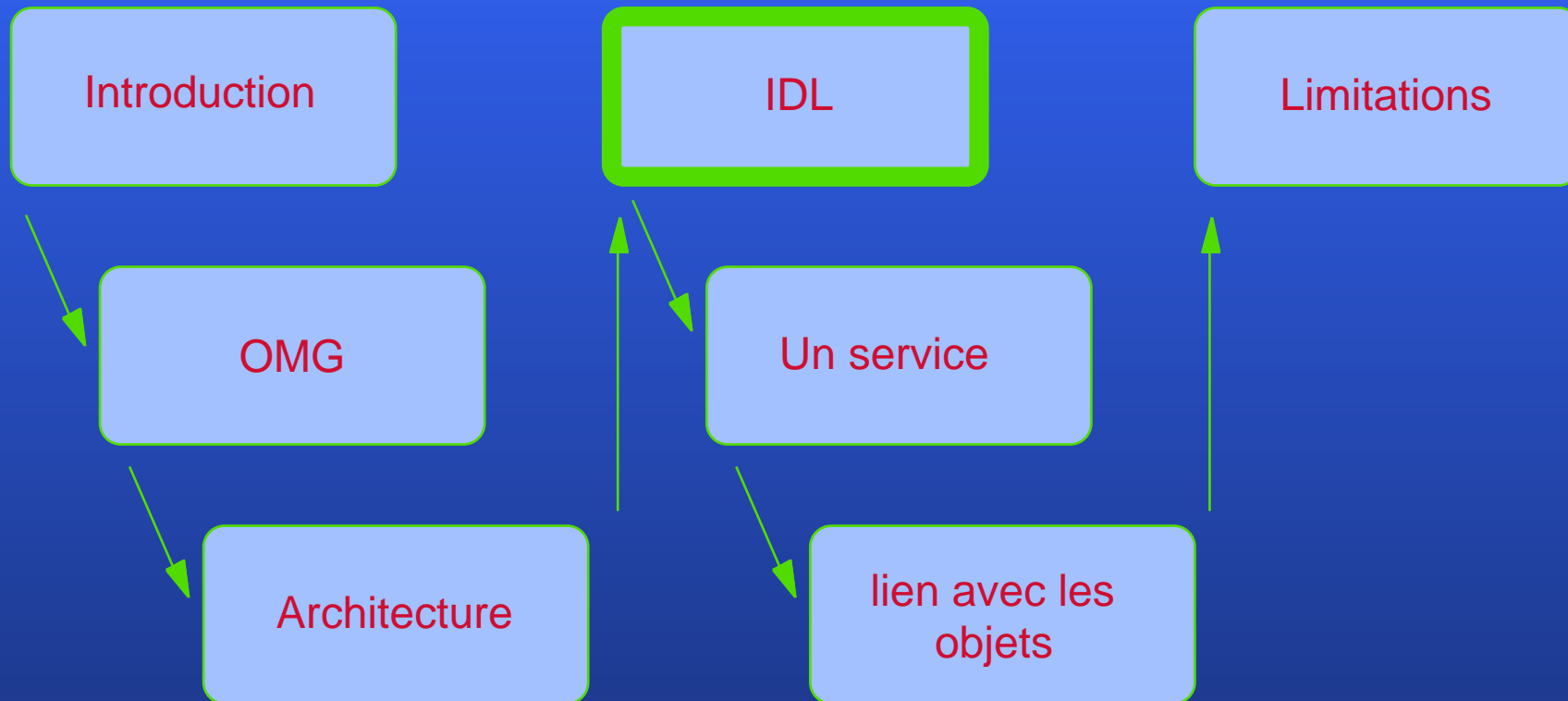


# IIOP





# Plan





# *Interface Repository et IDL*

- **Interface Definition Language**
- **Interface Repository**

## Un exemple

C/C++ flavors  
définitions seulement

```
//POS Object IDL example
module POS {
    typedef string Barcode;
    interface InputMedia {
        typedef string OperatorCmd;
        void barcode_input(in Barcode item);
        void keypad_input(in OperatorCmd cmd);
    };
    interface OutputMedia {
        boolean output_text(in string string_to_print);
    };
};
```

## *Modules et portée*

- Définissent des "paquets" de définitions
  - types
  - constantes
  - exceptions
  - interfaces
- Les définitions ne sont visibles que dans la portée de `{...}` englobant.
- Importation par l'opérateur `::`
  - exemple `POS::Barcode`

## *Types IDL*

- long , short , unsigned long and short
  - float, double (IEEE Standard for Binary FP Arithmetic)
  - char (ISO Latin-1), string
  - boolean
  - octet
  - any
  - array
  - struct, union
  - enumeration
  - sequence
- typedef  
struct  
union switch  
enum  
sequence

# Interface

- Définissent des opérations "naturellement" liées

- type de l'opération obligatoire

- nom de l'opération

Signature

- types et noms des paramètres (in out inout)

```
void keypad_input(in OperatorCmd cmd);
```

```
boolean output_text(in string string_to_print);
```

- Des exceptions

- quand elles sont levées l'ORB les retourne au client pour traitement

```
exception input_out_of_range {long dummy};
```

```
void op(in long arg) raises(input_out_of_range);
```

## *Sémantique des invocations*

- Les opérations retournent au moins un objet
  - type
  - out
  - inout

- sauf celles spécifiées `oneway`

- doivent retourner `void`
- ne pas avoir de `out` `inout`
- ne pas lever d'exception

```
oneway void output(in string string_to_print);
```

# Contexte

- Ensemble de propriétés
  - nom
  - string
- Définit un environnement (à la Unix)

```
context(ctx1, ctx2, ...);
```

# Attributs

- Définissent l'état des objets

```
interface exemple{  
    attribute float heigth;  
    //readonly attribute float heigth;  
};
```

- et implicitement des opérations d'accès

```
interface exemple{  
    float _get_heigth();  
    void _set_heigth(in float h); //absent en readonly  
};
```

# Héritage

- Relation entre interfaces introduite par :
- Redéfinition des éléments interdites

```
interface exemple1{
    void op1(in float h);
};

interface exemple2 : exemple1{
    void op2(in float h, out int v);
};

interface exemple3 : exemple2, exemple1{
    void op1(in float h); // interdit
};
```

héritage multiple

## *L'Interface Repository*

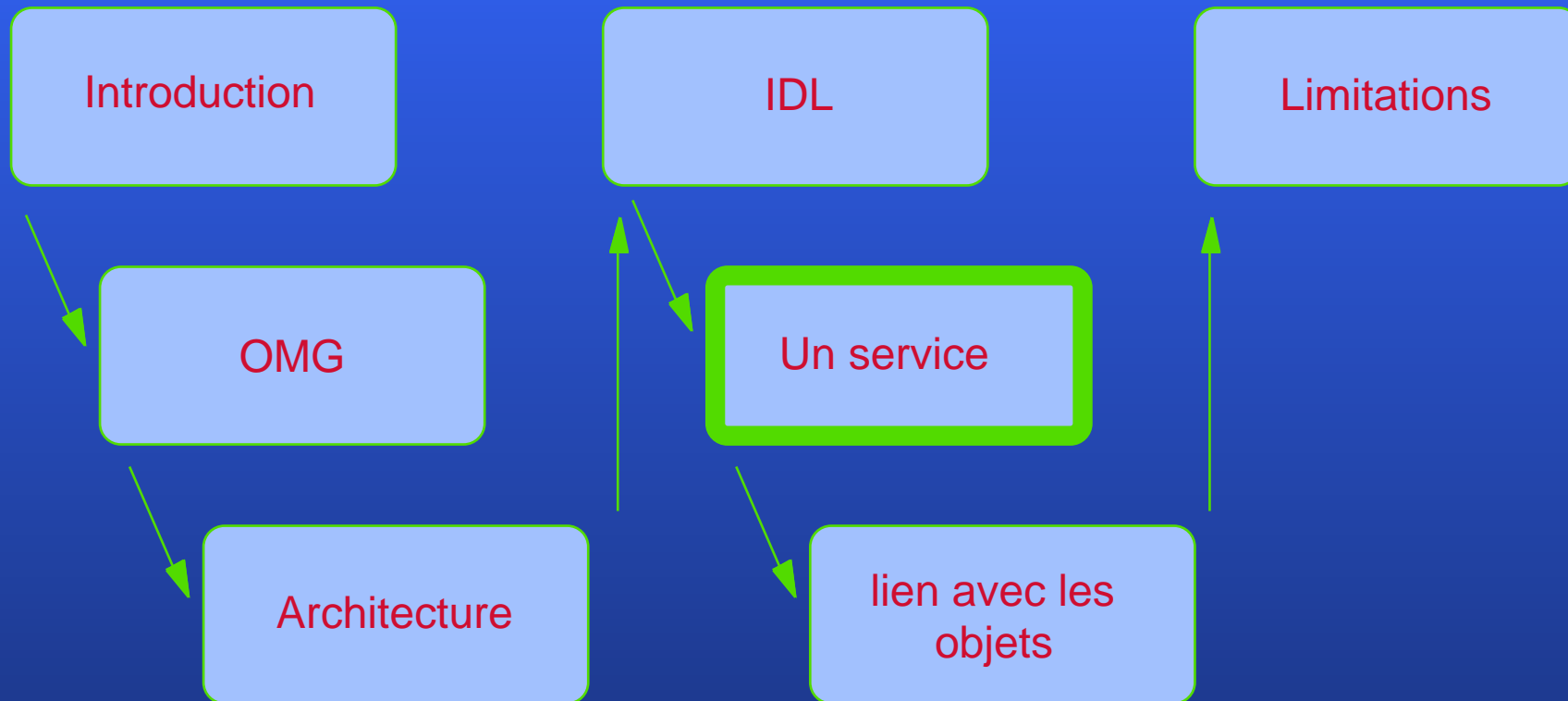
- Chaque ORB en possède un pour
  - enregistrer les définitions IDL
  - les mettre à jour
  - les fournir à la demande (DII)
- Et garantir :
  - l'interopérabilité des différents ORB
  - que les requêtes statiques ou dynamiques sont conformes aux signatures des opérations enregistrées
  - la correction du graphe d'héritage

## *Autres services de l'IR*

- Gérer la distribution et l'installation des interfaces sur un réseau
  - exemple : une IR partagée par plusieurs ORB
- En phase de développement, permet de naviguer et mettre au point les interfaces
- Compiler directement depuis l'IR plutôt que depuis les fichiers IDL

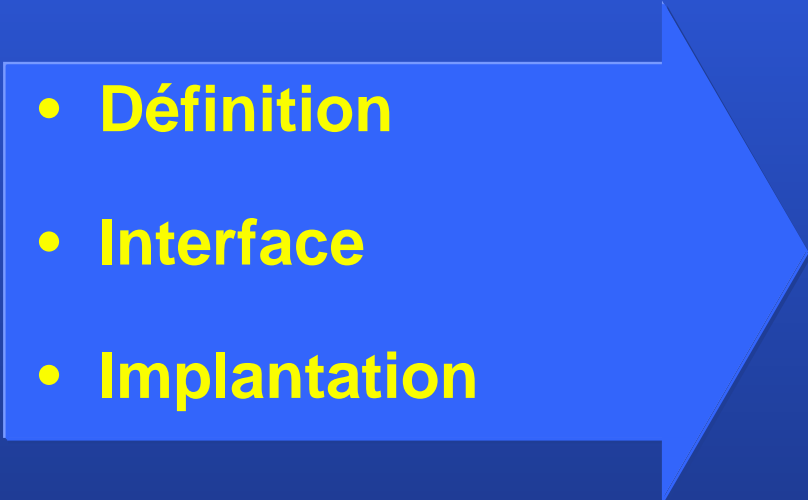


# Plan

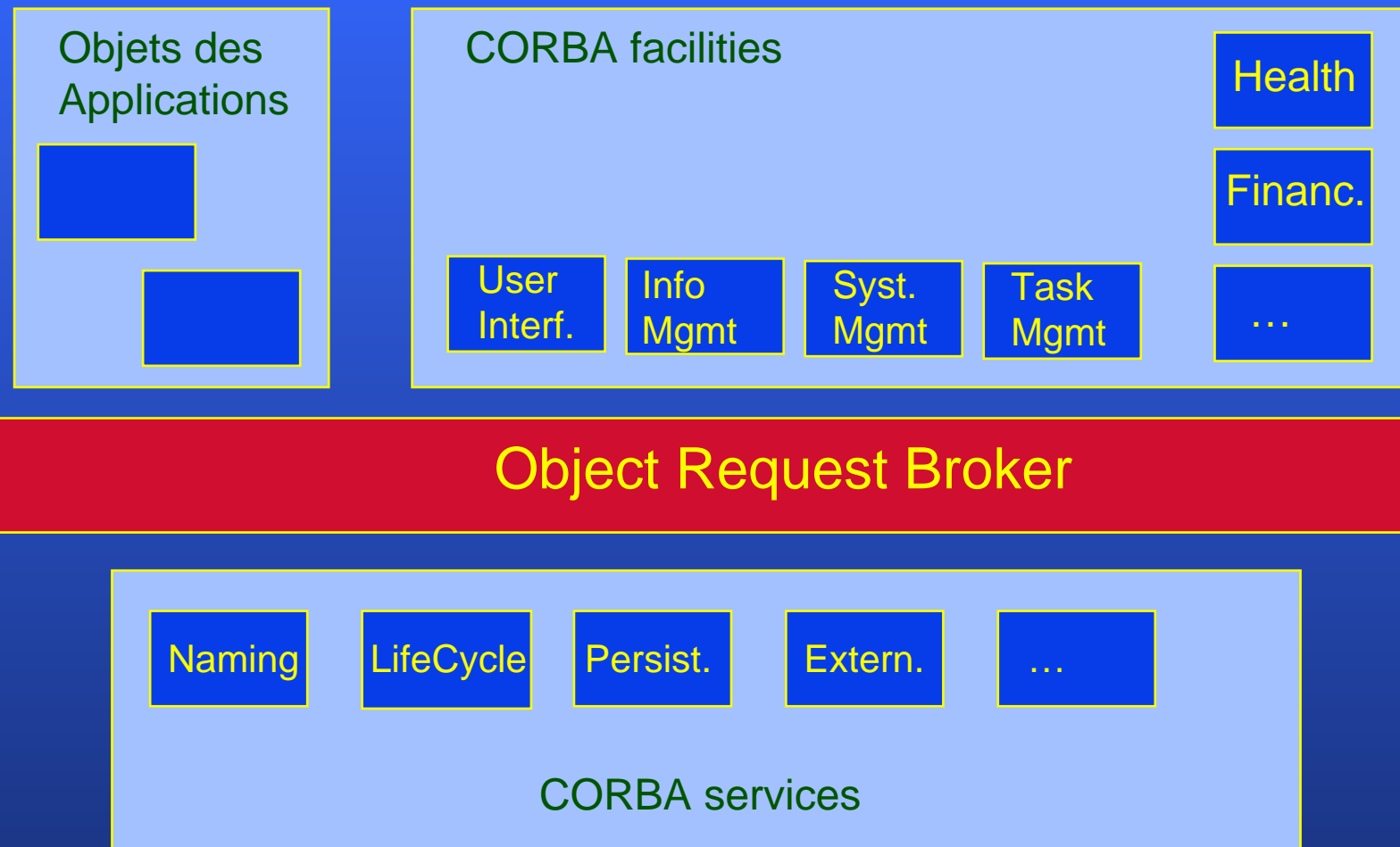




## *Un service en détail : Naming*

- 
- **Définition**
  - **Interface**
  - **Implantation**

# Object Management Architecture



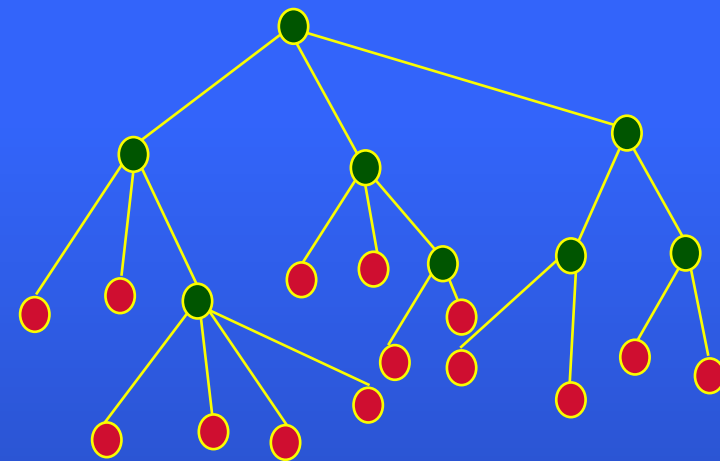
# *Principe*

- Lien entre un nom et un objet
- Le service nommage résoud (retrouve) un objet à partir de son nom
- Partage des services de nommage entre ORB

## Définition

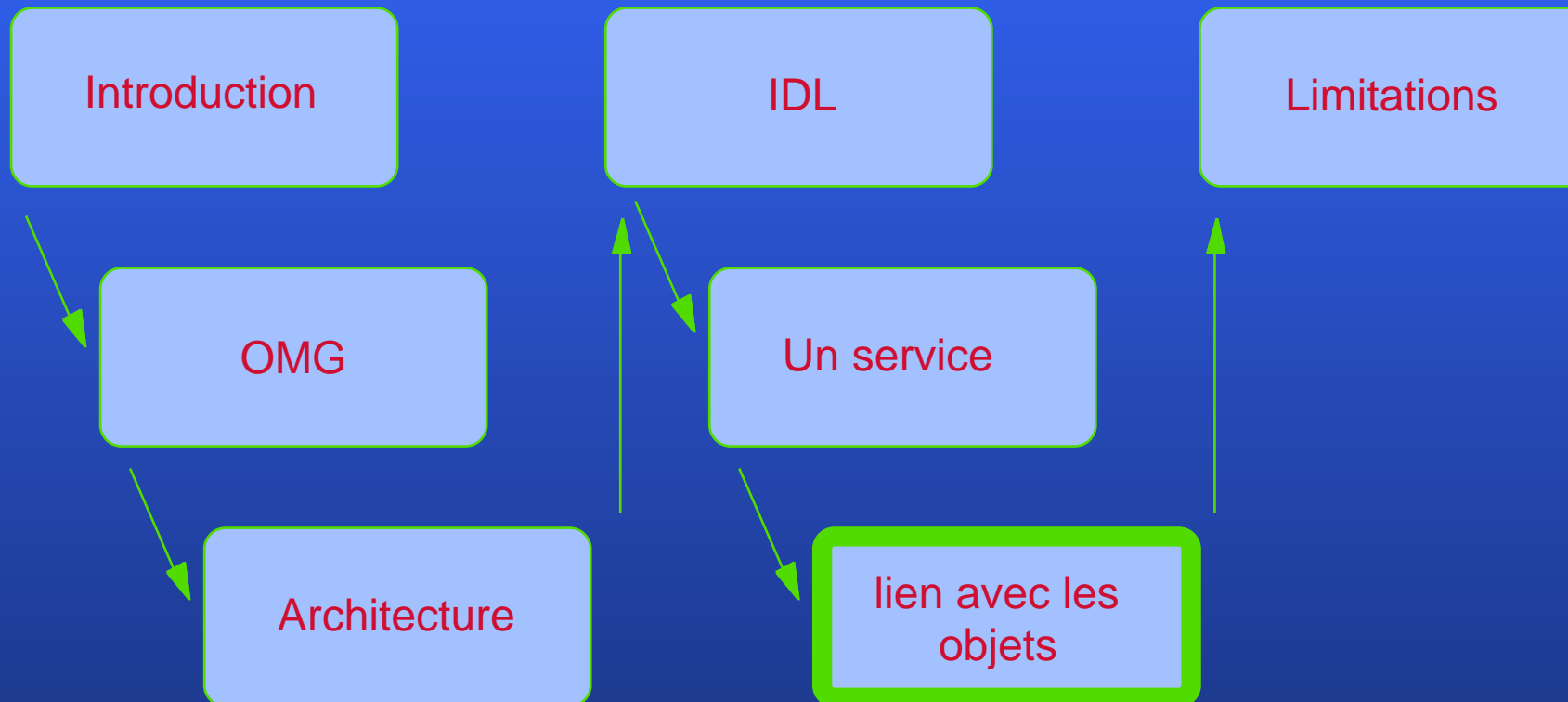
```
typedef string Istring
struct NameComponent {
    Istring id;
    Istring kind
};
```

```
typedef sequence<NameComponent> Name;
//séquence dont le dernier élément est le "name" ●
//et les prédécesseurs définissent le "naming context" ●
void bind(in Name n, in Object obj);
void bind_context(in Name n, in NamingContext nc);
Object resolve(in name n);
```





# Plan





## *Corba et les objets*

- Encapsulation "à la Corba"
- Héritage
- Polymorphisme

## *Encapsulation "à la Corba"*

- Séparation entre l'interface et l'implantation
- L'interface est un **contrat** entre le serveur et ses clients potentiels
- Transparence à la localisation ; un client s'adresse à l'ORB qui transmet...
- Et l'objet dans tout ça ?

# Héritage

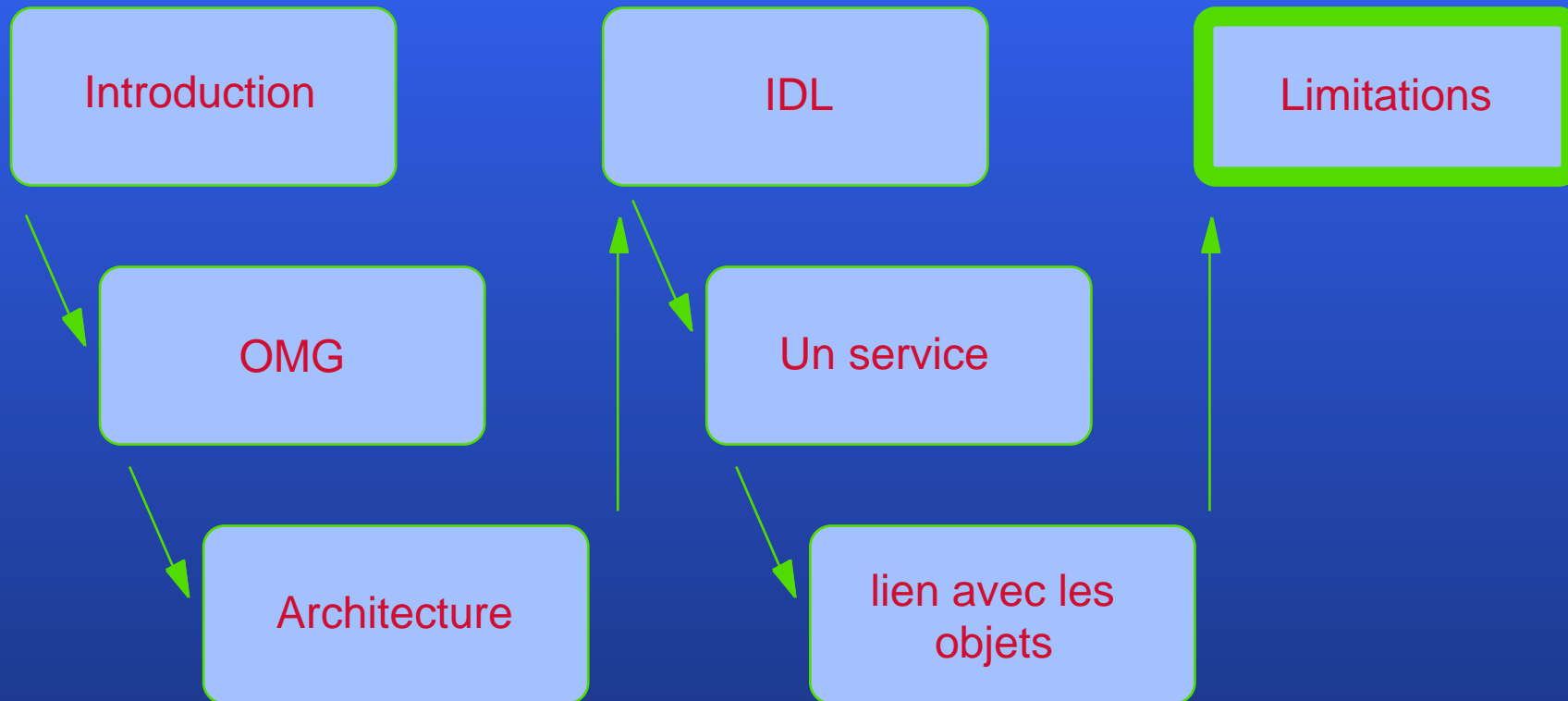
- Réutilisation des définitions des interfaces dont on hérite
- La surcharge est réalisée par l'implantation

# *Polymorphisme*

- Implicite dans l'IDL, c'est à l'implantation de l'assurer
  - Object Adapter
  - Mapping



# Plan





## *Limitations de Corba*

- **Spécification**
- **Sémantique**
- **Temps réel**
- **Gestion de la fiabilité avec IIOP**

## Spécification : contrat ?

- Une signature ne spécifie pas un service !
  - on garantit seulement que l'appel et les résultats sont correctement typés

```
montant déposer(in montant somme);
```

- Introduction de précondition et de postcondition

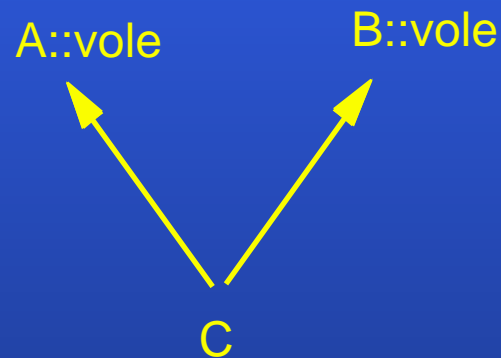
```
montant déposer(in montant somme)  
{require somme > 0}  
{ensure solde = old solde + somme};
```

## *Sémantique des opérations*

- Qui garantit le bon usage des ressources ?
- Invoquer une opération est-ce réaliser une opération atomique ?
- Faut-il faire des Transactions ?
- Faites confiance au service implanté !

## Sémantique de l'héritage

- Quelle sémantique pour l'héritage multiple ?



- Faites encore une fois confiance au serveur !

## *Temps réel*

- Aucune garantie de qualité de service
  - durée d'attente de la réponse
  - durée d'exécution du service
- Aucune connaissance de la stratégie d'ordonnancement du serveur

## *Gestion de la fiabilité avec IIOP*

- IIOP assure une connexion point à point
- C'est au programmeur client et serveur d'assurer la gestion des problèmes de communication.
  - connexion rompue
  - perte de messages
- ESIOP/DCE offre ce service au dépend d'un coup de communication plus élevé dû au protocole fiabilisant la communication

## Conclusion

- Standard d'interopérabilité incontournable
- Des manques liés au processus même de standardisation qui mène toujours à des solutions simples, minimaliste ?
- Retenu dans la norme Open Distributed Processing (ODP) comme modèle d'ingénierie (interopérabilité)
- De nombreux projets industriels commencent à s'appuyer sur Corba...

## *Bibliographie*

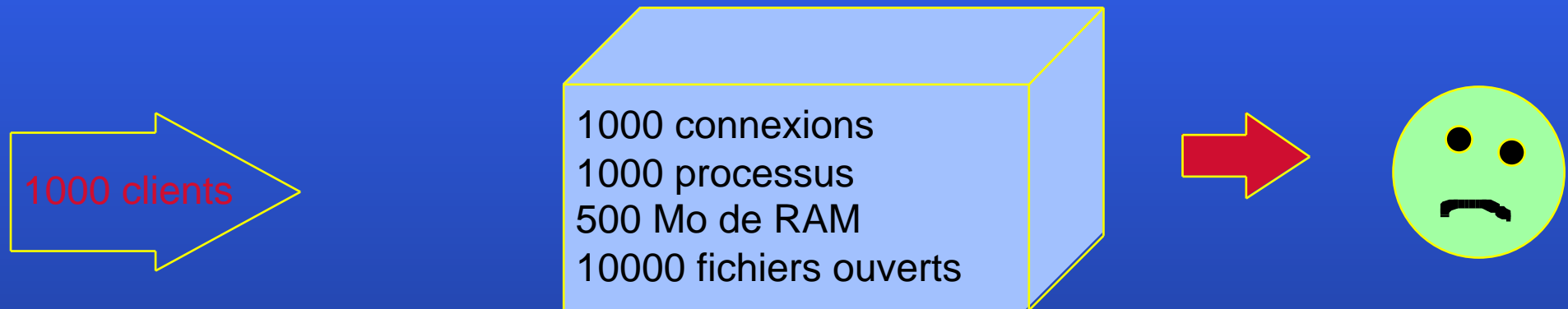
R. Orfali, D. Harkey, J. Edwards, *The Essential Distributed Objects Survival Guide*, Wiley & sons, Inc, 1996

Jon Siegel, *CORBA Fundamentals and Programming*, Wiley & sons, Inc, 1996



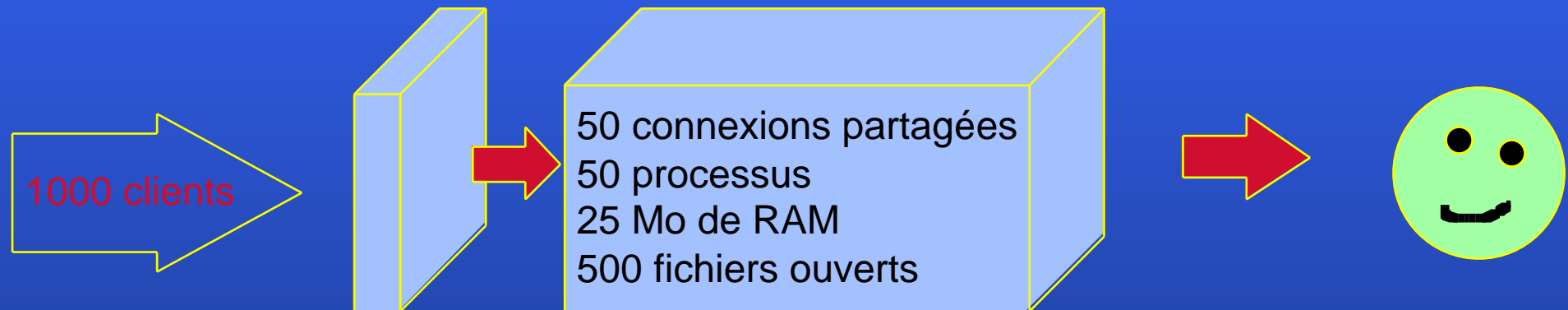


# Client Serveur





# Client Serveur avec moniteur



Moniteur transactionnel