Systèmes répartis

Samuel Tardieu

Samuel.Tardieu@enst.fr

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

Qu'est-ce qu'un système réparti?

- Un système réparti (ou distribué, de «distributed system»), est:
  - composé de plusieurs systèmes calculatoires autonomes (sinon, non réparti)
  - sans mémoire physique commune (sinon c'est un système parallèle, cas dégénéré)
  - qui communiquent par l'intermédiaire d'un réseau (quelconque)

Exemples de systèmes répartis

- On rencontre des systèmes répartis dans la vie de tous les jours:
  - WWW, FTP, Mail
  - Guichet de banque, agence de voyage
  - Téléphones portables (et bornes)
  - Télévision interactive
  - Agents intelligents
  - Robots footballeurs

Pourquoi utiliser un système réparti?

- Les systèmes répartis sont populaires pour plusieurs raisons:
  - Accès distant: un même service peut être utilisé par plusieurs acteurs, situés à des endroits différents
  - Redondance: des systèmes redondants permettent de pallier une faute matérielle, ou de choisir le service équivalent avec le temps de réponse le plus court

Pourquoi ... ? (suite)

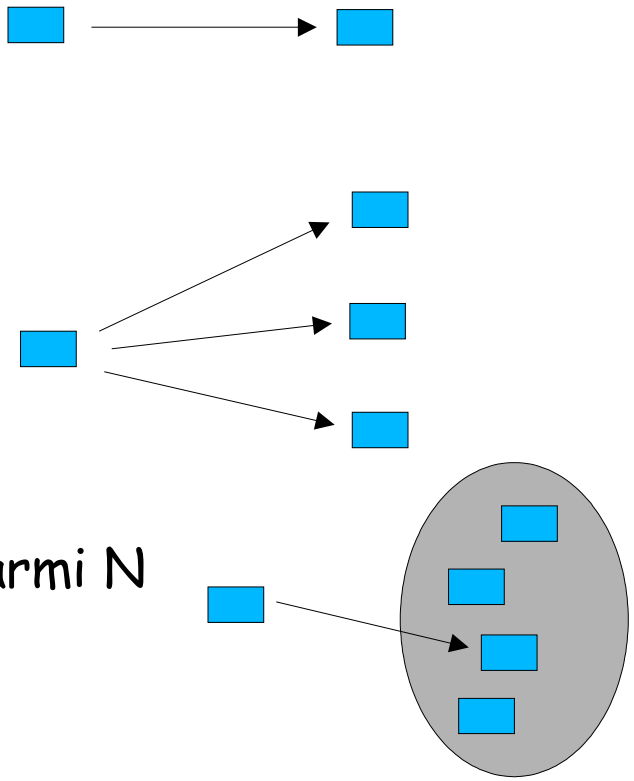
- Performance: la mise en commun de plusieurs unités de calcul permet d'effectuer des calculs parallélisables en des temps plus courts
- Confidentialité: les données brutes ne sont pas disponibles partout au même moment, seules certaines vues sont exportées

Comment communiquer? Les protocoles

- Un protocole est un « langage » de communication utilisé par deux ordinateurs pour communiquer entre eux.
- Des protocoles de bas niveau sont utilisés sur les réseaux:
  - Réseaux physiques (liaison spécialisée)
  - Réseaux virtuels de bout en bout (X25, ATM, modem/téléphone)
  - Systèmes basés sur les paquets (IP/Ethernet, RFC 1149, téléphone cellulaire)

Différents types de communications

- Unicast
  - Un émetteur
  - Un récepteur
- Multicast
  - Un émetteur
  - N récepteurs
- Anycast
  - Un émetteur
  - Un récepteur parmi N



Exemples de protocoles

- Protocoles non fiables: IP, UDP
  - Réémission et réordonnancement à la charge de l'utilisateur
  - Meilleure utilisation d'un réseau fiable et disponible
- Protocoles fiables: TCP, ATM
  - Réémission et réordonnancement automatiques
  - Surcoût systématique au niveau du noyau

Plan

- Généralités
- Classes de systèmes répartis
  - Envoi de messages
  - Appel de sous-programme à distance
  - Objets répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

Problèmes théoriques (non abordés ici)

- Le problème du temps
  - Ordonnancement des événements
- Le vote ou le consensus
  - Election d'un leader
  - Détection de fautes, normales ou byzantines
- Le partitionnement
  - Détection du partitionnement
  - Gestion de la réconciliation

Communication par envoi de messages

- Système réparti primitif: des messages indépendants unidirectionnels sont envoyés sur un réseau, fiable ou non fiable.
- Exemple de tels systèmes:
  - IRC, ICQ, FreeNet
  - X/Window (doublement asynchrone, le client et le serveur peuvent spontanément transmettre des informations, notion de boucle d'événements)

Localisation des services

- Problème: comment localiser un service sur la machine locale?
- Solution: utiliser le fichier /etc/services pour TCP et UDP

```
#Nom        Port        Alias    Commentaire
ftp         21/tcp
fsp         21/udp      fspd
Ssh         22/tcp                # Secure shell
```

Systèmes hétérogènes

- En présence d'un système hétérogène, il faut:
  - Définir un format d'échange commun pour les types de base (little-endian ou big-endian, 32 bits ou 64 bits, etc.)
  - Utiliser des dictionnaires pour les types complexes (ordre des composants dans une structure, etc.)
- Ces conventions doivent être connues a priori par les deux systèmes
- L'oubli de ces traductions fonctionnera néanmoins dans un système homogène; détection de problèmes difficile.

Exemple de dictionnaire: XDR

- XDR (eXternal Data Representation) est un format d'échange défini par Sun, qui normalise:
  - le format (poids fort d'abord) et la taille des entiers
  - le format des nombres en virgule flottante
  - le format des structures complexes

D'autres formats existent (CDR dans CORBA par exemple)

Données non transportables

- Certaines données ont une sémantique locale
  - Pointeurs
  - Objets actifs (tâches, fichiers)
- Ces objets sont difficilement transportables
- On peut parfois contourner ces limitations
  - Aplatissement de liste (transformation en tableau)
  - Encapsulation des données d'une tâche

Observation sur les dialogues utilisant l'envoi de messages

- Un dialogue classique est
  - Du côté de l'appelant
    - Envoi d'une requête et de ses arguments
    - Attente des résultats
  - Du côté de l'appelé
    - Attente d'une requête et de ses arguments
    - Calcul
    - Envoi des résultats
- Cela ressemble à un schéma connu...

Plan

- Généralités
- Classes de systèmes répartis
  - Envoi de messages
  - Appel de sous-programme à distance
  - Objets répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

Les RPC (appels de procédures à distance)

- Idée: remplacer les appels réseaux par des appels de sous-programmes
- Solution: les RPC
  - Dialogue question/réponse géré de manière transparente par le système
  - Les paramètres sont empilés sur le canal de communication plutôt que dans la mémoire

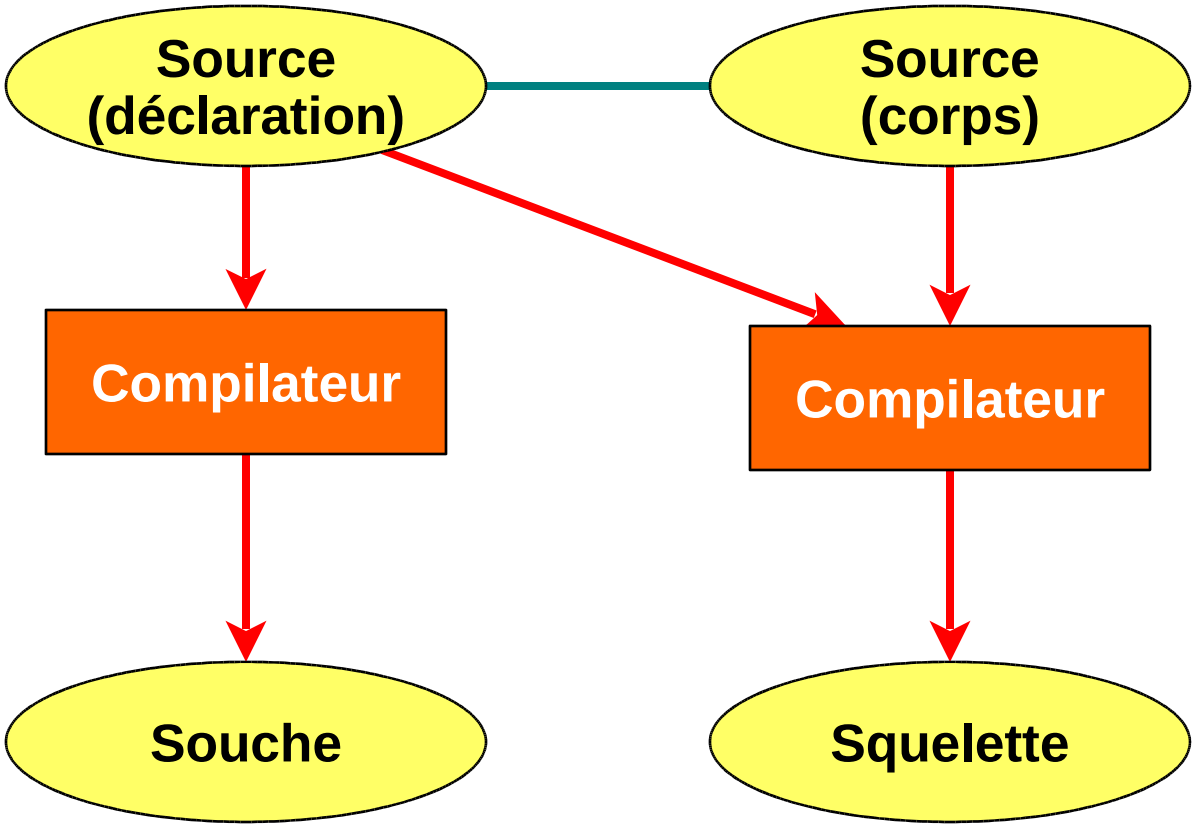
Pourquoi utiliser les RPC?

- Facilité de programmation (plus besoin de coder une boucle d'attente du résultat)
- Sujet bien maîtrisé de nos jours
- Facilité d'adaptation d'un code existant
- Possibilité de générer le code de l'appelant et de l'appelé automatiquement

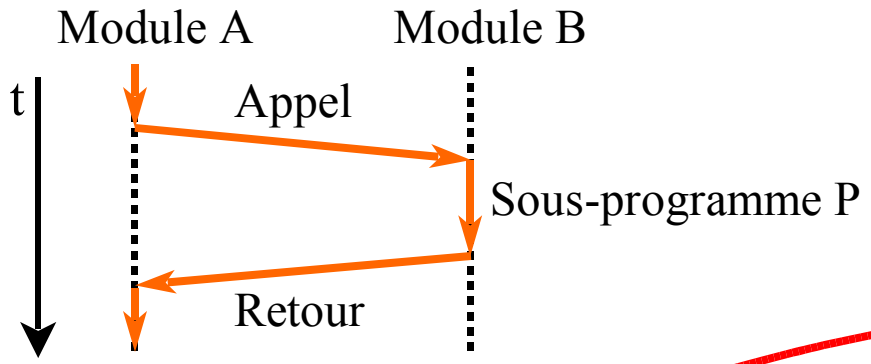
Souches et squelettes

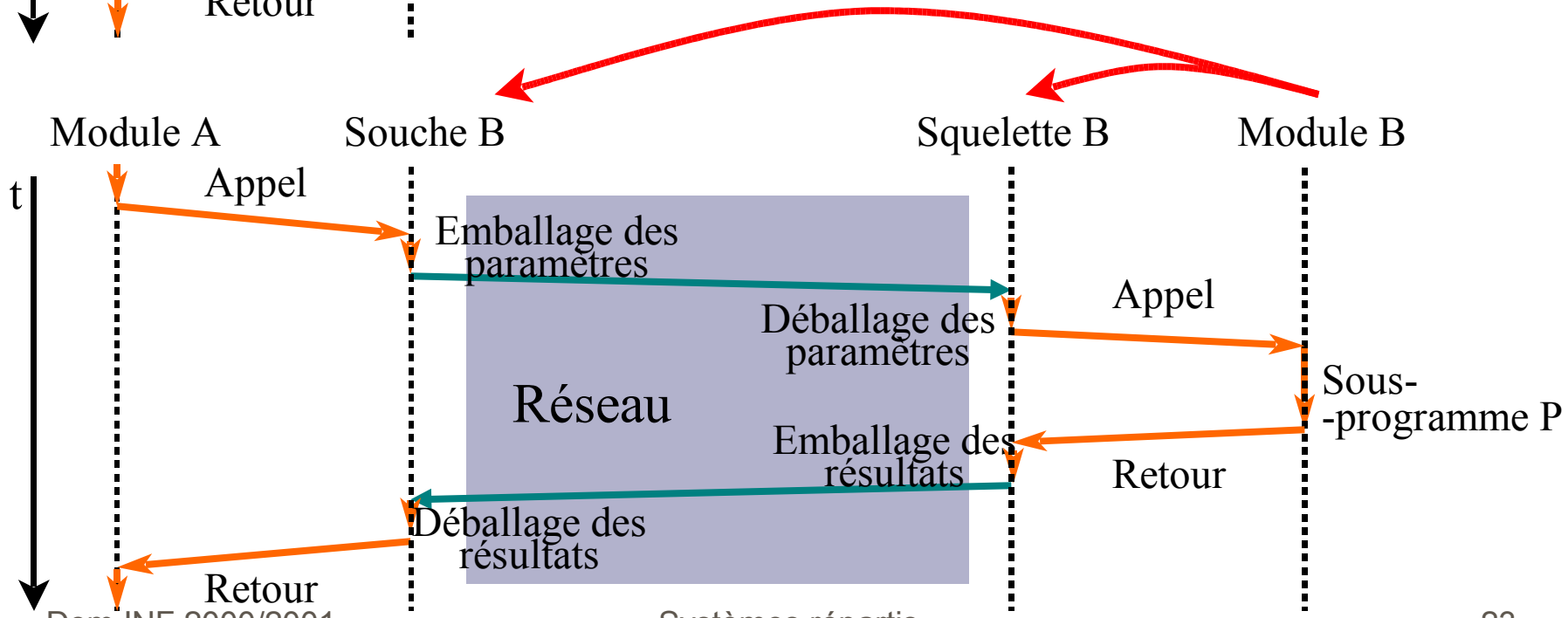
- A partir d'une spécification d'interface, on peut générer une souche
  - Emballage des paramètres et envoi de la requête
  - Attente et déballage du résultat
- On peut également générer un squelette
  - Réception de la requête et déballage des arguments
  - Appel du sous-programme réel
  - Emballage et renvoi du résultat

Souches et squelette (suite)



Schématique d'un RPC





Exemple
Les RPC de Sun Microsystems

- Utilisation de XDR pour échanger les données entre systèmes hétérogènes
- Programme **rpcgen** pour construire les souches et squelettes à partir d'une description de l'interface
- Un service indispensable: NFS
  - Montage de répertoires en mode sans état
  - Fonctionne au dessus d'UDP ou de TCP

Adresses dynamiques et adresses notoires

- Les RPC de Sun proposent un service de nommage plus souple que `/etc/services`
  - Tout nouveau service vient s'enregistrer avec son nom auprès du portmapper
  - Toute requête à un service se fait en interrogeant d'abord le portmapper afin de connaître le numéro de port utilisé
- Avantage: plus de liste de services à partager
- Contrainte: le portmapper est sur un port fixe

RPC évolués

- Il est possible d'augmenter les facilités offertes par les RPC
  - **Exceptions**: transmission des exceptions de l'appelé vers l'appelant
  - **Procédures asynchrones**: aucune nécessité d'attendre le résultat d'un sous-programme (procédures unidirectionnelles)

RPC évolués, suite

- On peut aussi utiliser
  - **Pointeurs sur sous-programmes distants**: au lieu de générer un appel statique vers un sous-programme distant, on utilise un pointeur dont on ne sait pas a priori où il pointe
  - **Contrôle de version**: vérification de la cohérence entre la souche et le squelette

Services associés aux RPC

- Possibilité d'ajouter des services externes
  - Service de nommage
    - Enregistre les coordonnées des services distants
    - Est appelé par le biais de RPC
    - Doit être localisé d'une autre manière (adresse notoire)
  - Service de sécurité
    - S'assure de l'identité des différents acteurs
    - Fournit des jetons d'authentification et de chiffrement

RPC en environnement concurrent

- Un client peut faire plusieurs requêtes à un serveur (environnement concurrent)
- Il faut soit
  - Utiliser un canal de communication par tâche du client
  - Utiliser un numéro de séquence pour différencier les requêtes
  - Le serveur renvoie le numéro de séquence avec la réponse

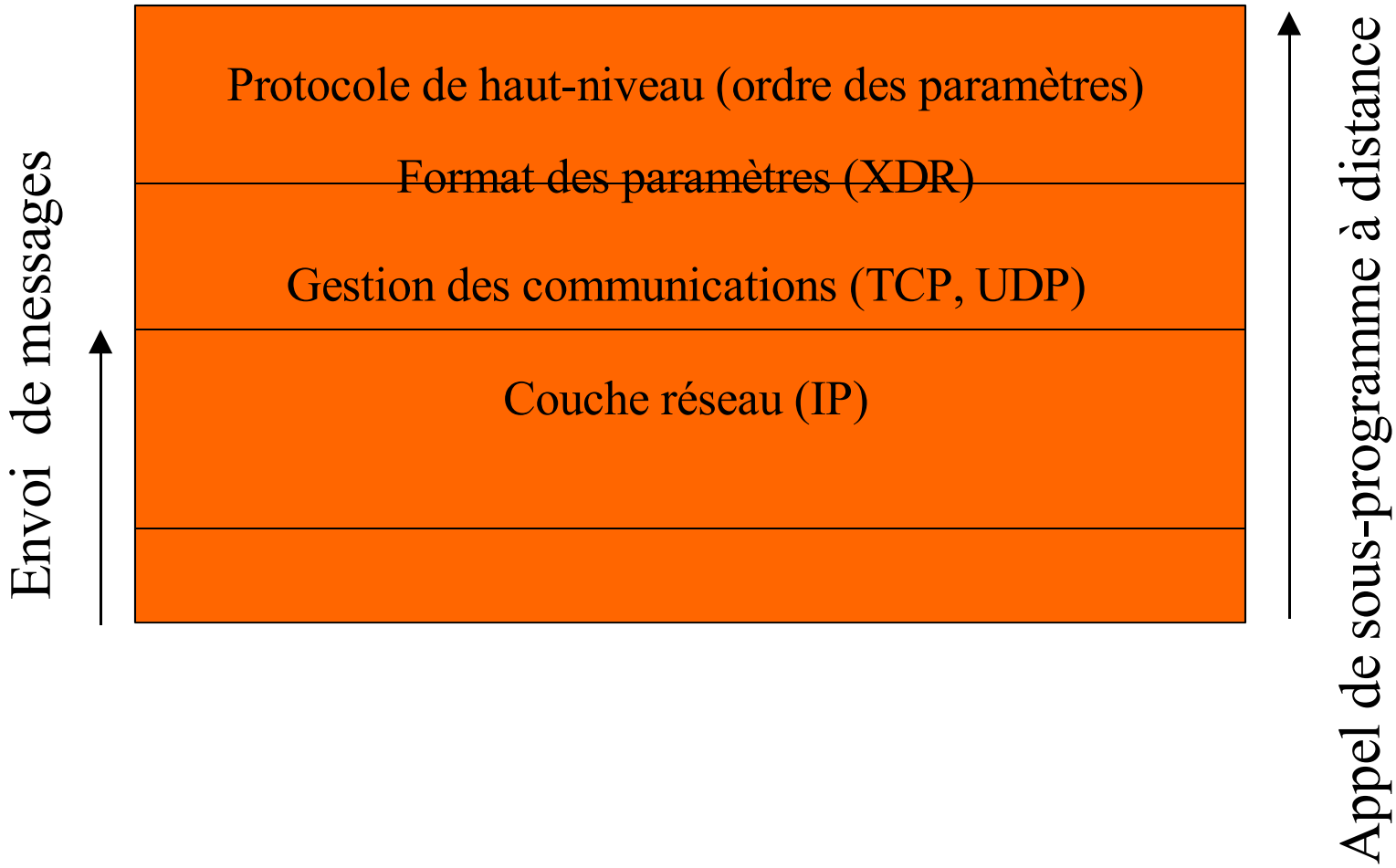
Et du côté du serveur?

- Plusieurs possibilités pour traiter des requêtes entrantes concurrentes
  - Lancer un serveur esclave à chaque connexion (fork(), comme les serveurs WWW)
  - Lancer une tâche à chaque requête (NFS sous Solaris)
  - Exécuter les requêtes les unes après les autres (sérialisation)

Envoi de messages vs. RPC

- Utiliser les RPC permet
  - De réduire les erreurs dues à la programmation des boucles d'attente
  - De laisser les programmeurs se concentrer sur leurs domaines d'activités
  - De tester la cohérence des versions
- Envoyer des messages permet d'optimiser certains systèmes spécifiques (homogènes)

Apport de chaque méthode



Plan

- Généralités
- Classes de systèmes répartis
  - Envoi de messages
  - Appel de sous-programme à distance
  - Objets répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

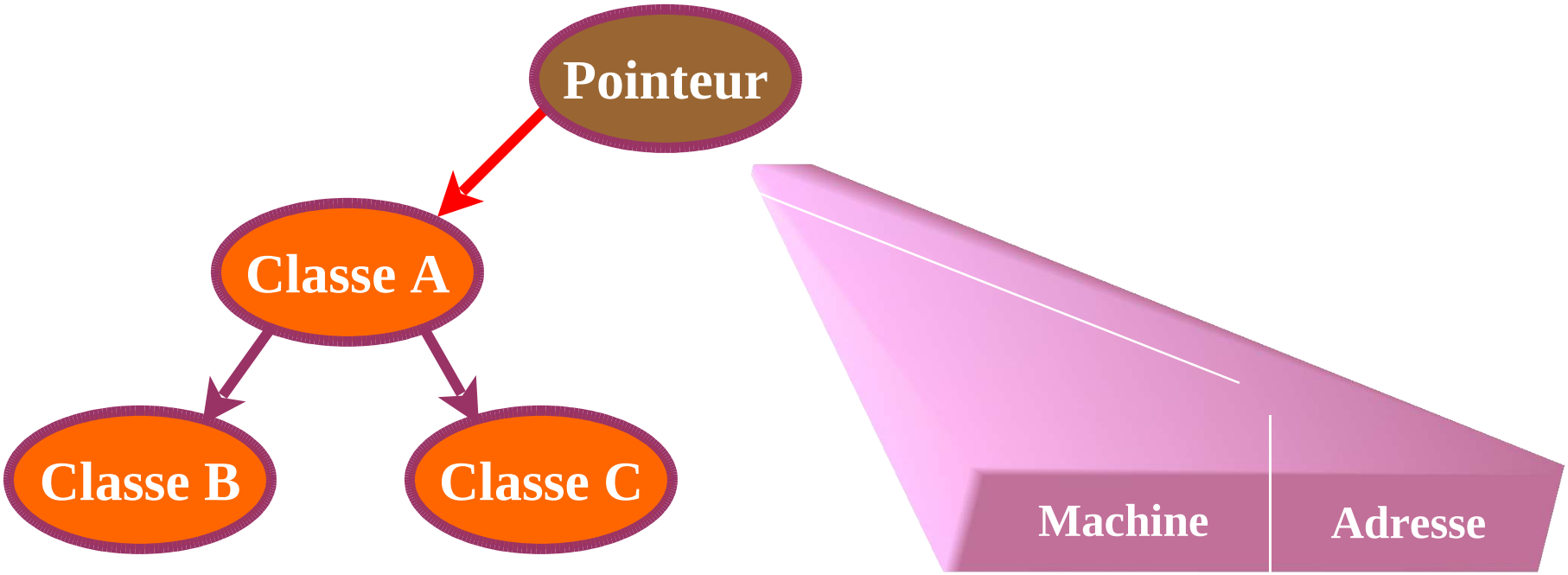
Les objets répartis

- La programmation orientée objet enrichit la programmation structurée
- Les objets répartis enrichissent les RPC
  - Conception orientée acteur plutôt que fonctionnalité
  - Programmation par extensions: il est possible d'étendre un service progressivement
- Aucune différence de concept entre les modes réparti et non-réparti (monolithique)

Caractéristique des objets répartis

- Un objet réparti
  - Ne se déplace pas physiquement (seules les références sont échangées)
  - N'est accédé qu'au travers de ses méthodes
  - Peut être désigné de plusieurs manières (pointeur long par exemple)
  - N'est utilisé qu'à travers un pointeur

Pointeur sur objet distant



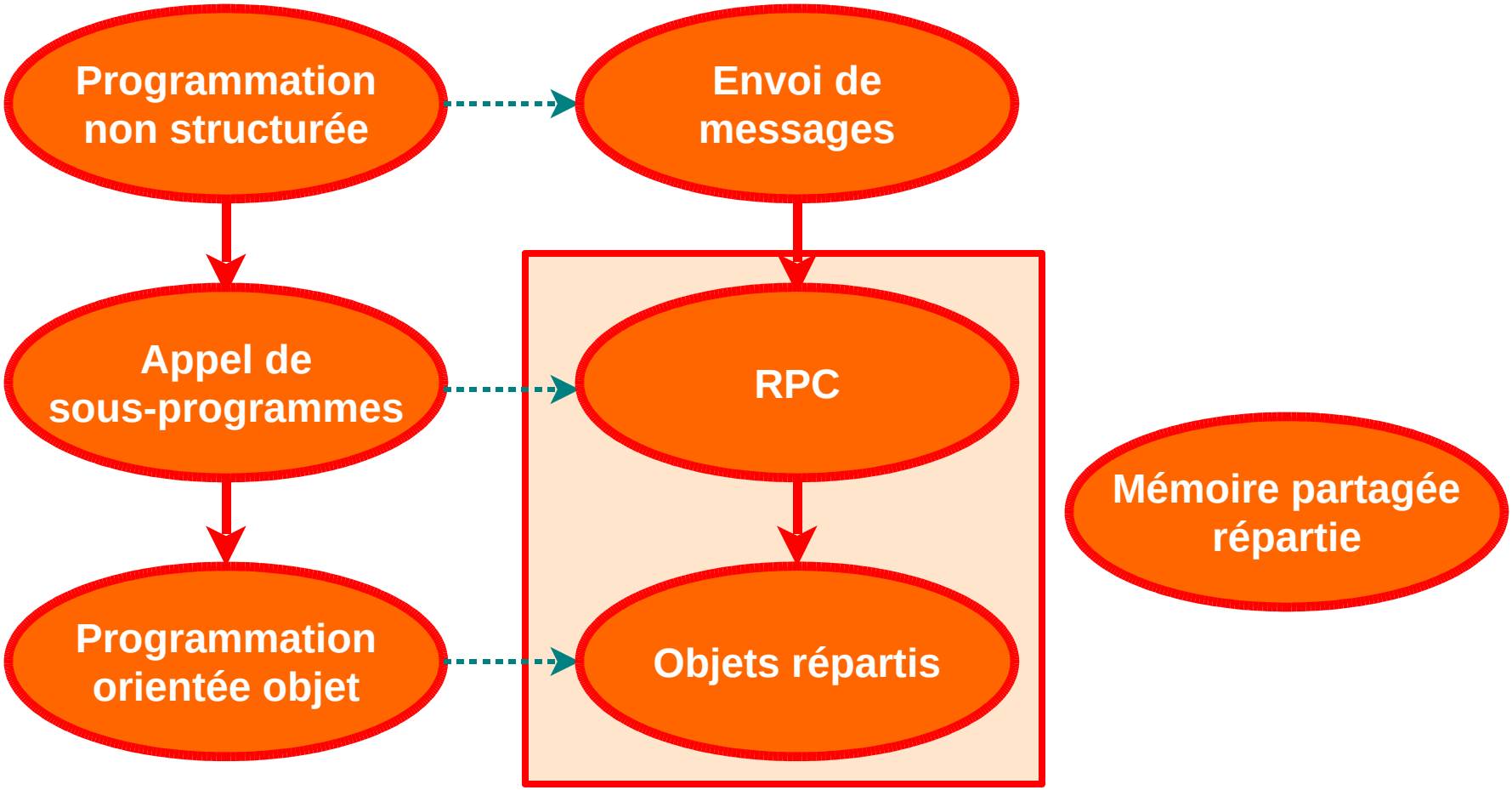
On n'insistera jamais trop...

- Les objets répartis sont en fait des pointeurs sur objets distants
  - Pas de mouvement d'objets (champs et méthodes sont fixes)
  - Possibilité d'échanger des références
  - Aucune possibilité de garder quoi que ce soit de plus que les références

Qui utilise les objets répartis?

- CORBA
  - Basé uniquement sur les objets répartis
  - Supporte les programmes multi-langages
- Ada 95
  - Supporte les objets répartis et les RPC
  - Permet de définir des objets partagés (mémoire partagée virtuelle)
  - Est limité au langage Ada

Les différentes techniques



Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
  - CORBA
  - Ada 95
- Autres types de systèmes répartis
- Conclusion

Les objets dans CORBA

- Les objets CORBA sont originaux
  - Les interfaces sont décrites dans le langage IDL (Interface Description Language)
  - Chaque interface est traduite dans un
    - Langage hôte pour le client (souche)
    - Langage hôte pour le serveur (squelette)
  - Le programmeur complète le squelette et utilise la souche
  - Le programmeur ne livre que l'IDL (contrat)

L'architecture CORBA

- L'IDL suffit à invoquer une méthode sur un objet (notion d'interface-contrat)
- Les appels de méthodes transitent par des ORB (Object Request Broker)
- Les ORB constituent un bus logiciel
- Les ORB communiquent avec un ensemble de protocoles standardisés (IIOP, GIOP)

L'IDL en quelques points

- Syntaxe similaire à celle du C++
- Sous-ensemble commun à tous les langages hôtes, donc pas très expressif (pas de sous-types par exemple)
- Le programmeur doit connaître les détails de la traduction entre l'IDL et le langage hôte choisi pour implémenter le squelette

Exemples d'IDL

```
interface Echo {
   string echoString (in string mesg);
};
```

```
Interface Buffer {
   exception Empty;
   void put (in string content);
   string get() raises (Empty);
};
```

Traduction Ada du service Echo

```
with Corba.Object;
package Echo is
   type Ref is new Corba.Object.Ref with null record;
   function To_Echo (Self : in Corba.Object.Ref'Class)
      return Ref'Class;
   function To_Ref (From : in Corba.Any) return Ref;
   function To_Any (From : in Ref) return Corba.Any;
   function echoString (Self : in Ref;
                        msg  : in Corba.String)
                       return Corba.String;
   Null_Ref : constant Ref := (Corba.Object.Null_Ref
                               with null record);
   Echo_R_Id : constant Corba.RepositoryId :=
      Corba.To_Unbounded_String ("IDL:Echo:1.0");
end Echo;
```

Les services CORBA

- Le noyau CORBA
  - Est tout petit
  - Est enrichi de nombreux services externes
- Les services
  - Service de nommage hiérarchique réparti
  - Service de persistance
  - Service de transactions
  - Service de sécurité

Le service de nommage CosNaming

- Adresse de la racine localisable autrement
- Nommage hiérarchique
  - Une entrée pointe sur un objet CORBA
  - Un objet peut être une liste de nœuds (répertoire)
  - Possibilité d'avoir plusieurs arbres indépendants
- Service normalisé par l'OMG
- Possibilité de stocker n'importe quel type de données

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
  - CORBA
  - Ada 95
    - Concepts
    - RPC et RPC évolués
    - Objets répartis
    - Résume des concepts
- Autres types de systèmes répartis
- Conclusion

Une approche langage Ada 95

- Ada 95 permet de construire des systèmes répartis sans sortir du cadre du langage
- Répartition introduite en même temps que l'orienté objet
- Remplace toutes les solutions propriétaires existant depuis des années
- Ne permet pas de coopérer avec d'autres systèmes

Langage de description pour Ada

- A la différence d'IDL
  - Le langage de description est un sous-ensemble du langage de l'application, Ada 95
  - Les règles de typage, de visibilité et de sécurité sont préservées
  - Le langage de description est beaucoup plus riche
  - La sémantique est déjà connue et maîtrisée
- L'unité de répartition est le paquetage
- Les paquetages sont regroupés en partitions

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
  - CORBA
  - Ada 95
    - Concepts
    - RPC et RPC évolués
    - Objets répartis
    - Résume des concepts
- Autres types de systèmes répartis
- Conclusion

Catégorisation des paquetages

- Des directives de compilation permettent de catégoriser certains paquetages
  - **`pragma Remote_Call_Interface`** désigne un paquetage dont les sous-programmes peuvent être appelés à distance
  - **`pragma Remote_Types`** désigne un paquetage dont les types peuvent être transportés
  - **`pragma Shared_Passive`** désigne un paquetage contenant des variables partagées

Note sur le partitionnement

- Le partitionnement est un processus post compilatoire
  - Il est possible de passer de monolithique en réparti et vice-versa sans recompilation
  - Il est possible de repartitionner une application sans recompilation
  - Il est facile de debugger une application non répartie en la rendant monolithique

Comparaison: le service Echo

```
package EchoSvc is
   pragma Remote Call Interface;
   --  les sous-programmes déclarés
   --  ici seront appelables à distance
   function Echo String (S : String)
      return String;
end EchoSvc;
```

Le service Echo analysé

- En présence de la directive Remote_Call_Interface, le compilateur
  - Vérifie que tous les types utilisés sont transportables (tous les types scalaires et agrégats de types transportables)
  - Génère deux codes différents, un pour la souche, un pour le squelette
- Cette directive ne change pas la sémantique du paquetage à laquelle il s'applique

Utilisation du service Echo

```
with Ada.Text_IO; use Ada.Text_IO;
with EchoSvc;     use EchoSvc;
procedure CallEcho is
begin
   Put_Line (Echo_String («abcde»));
   -- Génère un appel à distance
end CallEcho;
```

Traitement des exceptions

```
with Ada.Text_IO; use Ada.Text_IO;
with EchoSvc;     use EchoSvc;
procedure CallEcho is
begin
   Put_Line (Echo_String («abcde»));
   -- Génère un appel à distance
exception
   when others =>
      Put_Line («Error when calling Echo»);
end CallEcho;
```

Sémantique des exceptions

- Sémantique préservée entre appel local et appel distant
- Possibilité de rattraper une exception inconnue par une clause «when others»
- Possibilité de relever une exception inconnue, puis de la rattraper par son nom
- Exception prédéfinie: `Communication_Error`

Appels asynchrones

- Permet de ne pas attendre de résultat
- Toute exception est perdue
- Création implicite de parallélisme
- Ne s'applique que sur certaines procédures

```
package Log is
   pragma Remote_Call_Interface;
   procedure Log_(Event : in String);
   pragma Asynchronous (Log);
end Log;
```

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
  - CORBA
  - Ada 95
    - Concepts
    - RPC et RPC évolués
    - Objets répartis
    - Résume des concepts
- Autres types de systèmes répartis
- Conclusion

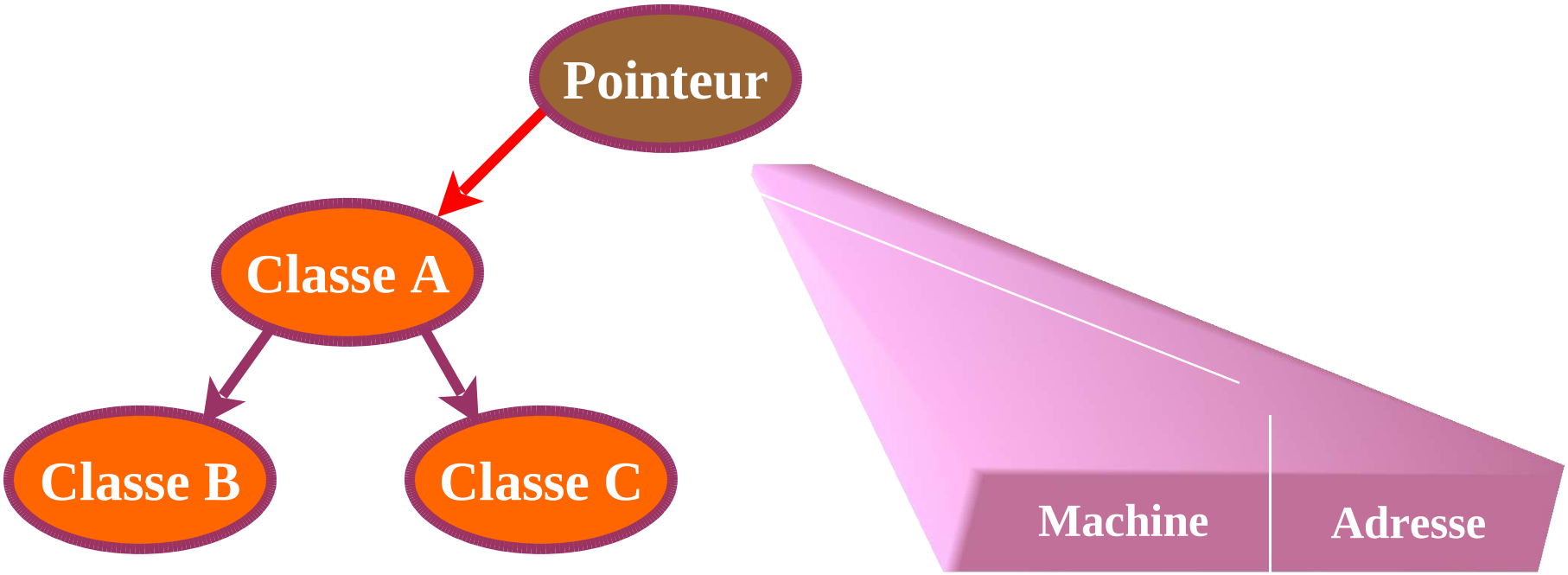
Objets répartis en Ada

- Pointeurs de types généraux
- Pointeurs déclarés dans un paquetage `Remote_Call_Interface` **ou** `Remote_Types`
- Type pointé
  - `tagged`: sommet d'une hiérarchie
  - `limited`: ne peut pas être copié ou comparé
  - `private`: les champs ne peuvent être modifiés qu'à travers les appels de méthodes

Déclaration de type «pointeur sur objet distant»

```
package Alerts is
   pragma Remote_Types;
   type Alert is abstract tagged limited private;
   procedure Handle (A : access Alert);
   procedure Log (A : access Alert)
      is abstract;
   type AlertPtr is access all Alert'Class;
   --  pointeur sur objet distant
   --  descendant de Alert
private
end Alerts;
```

Pointeur sur objet distant (rappel)



Utilisation d'un pointeur sur objet distant

- S'utilise comme un pointeur normal pour un appel de méthode (opération primitive)
- Ne peut pas être déréférencé hors d'un tel appel
- Subit un double aiguillage
  - Détermination de la partition sur laquelle se trouve l'objet distant
  - Détermination du sous-programme à appeler sur la partition

Exemple d'utilisation

```
with Alerts; use Alerts;
with Pools;  use Pools;    -- Pour Get_Alert
procedure Main_Loop is
begin
   loop
      Handle (Get_Alert);
      --  Get_Alert est défini comme
      --     function Get_Alert return AlertPtr;
   end loop;
end Main_Loop;
```

Extension d'un objet réparti

- Un objet ne sait pas qu'il est réparti, cela dépend si un pointeur sur objet distant existe
- Un objet réparti est étendu comme n'importe quel objet local
- L'extension doit elle aussi être privée
  - Pas de possibilité de modifier les champs sans passer par les méthodes
  - Elle offre néanmoins la même interface

Exemple d'extension

```
with Alerts; use Alerts;
with Types;  use Types;  --  Pour Person
package Medium_Alerts is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
private
   type Medium_Alert is new Alert with record
      Technician : Person;
   end record;
end Medium_Alerts;
```

Utilisation des objets répartis

- Un objet réparti
  - S'obtient à partir d'un objet local ou d'une autre référence
  - Cette référence peut être échangée par des appels de sous-programmes distants ou des appels de méthodes
  - Il faut donc au moins un appel de sous-programme distant
- Il faut donc au moins un paquetage Remote_Call_Interface dans une application répartie Ada

Exemple: une collection d'alarme

```
with Alerts; use Alerts;
package AlertPool is
   pragma Remote_Call_Interface;
   procedure Register (A : in AlertPtr);
   pragma Asynchronous (Register);
   function Get_Alert return AlertPtr;
end AlertPool;
```

Note sur le paquetage précédent

- On peut enregistrer une alarme, locale ou récupérée, de n'importe où
- Trois partitions sont potentiellement en jeu
  - Le producteur de l'alarme, qui l'enregistre auprès de Register
  - La partition gérant AlertPool
  - La partition appelant Get_Alert
- L'appel à Handle reviendra à la première partition

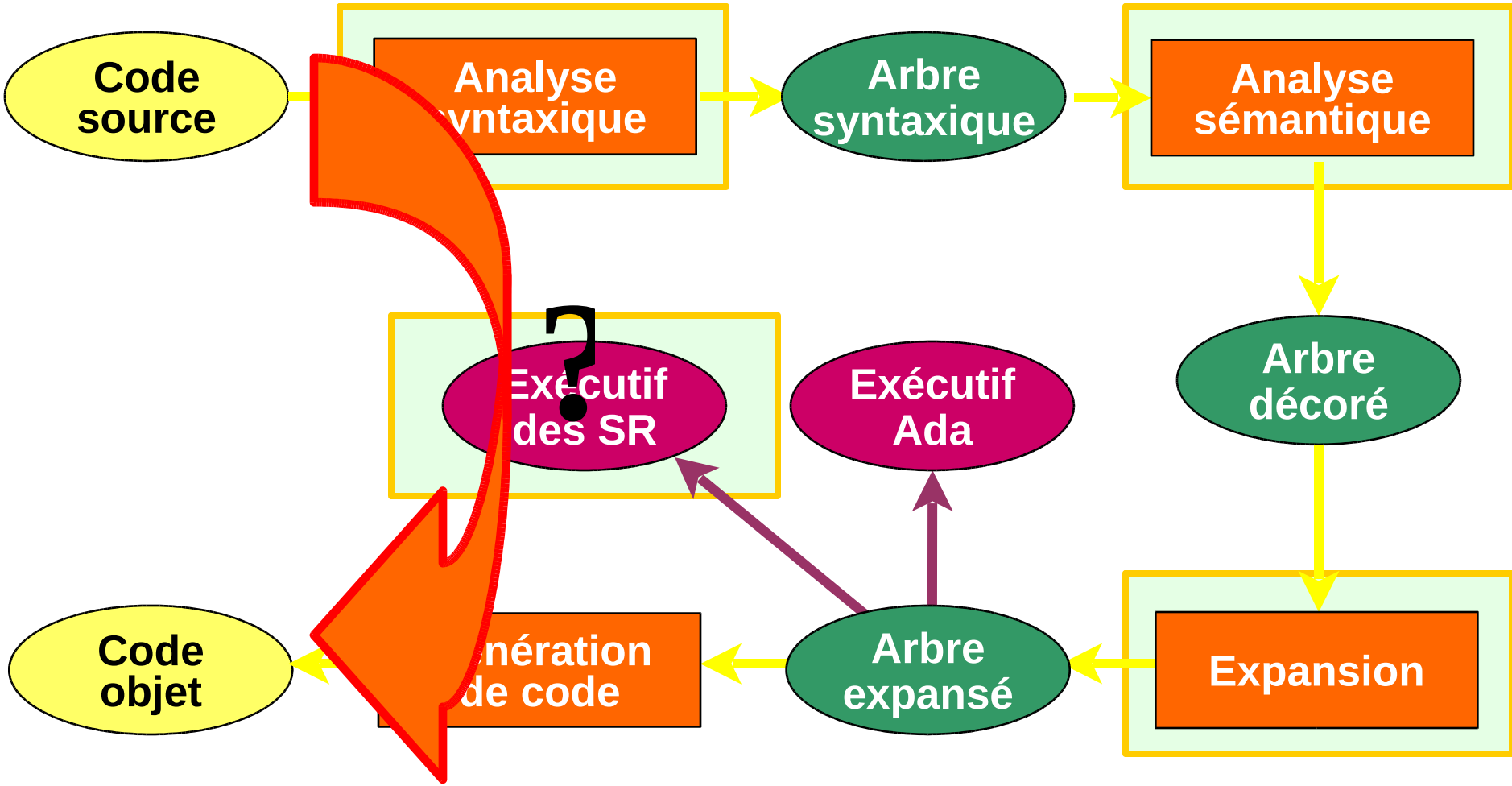
Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
  - CORBA
  - Ada 95
    - Concepts
    - RPC et RPC évolués
    - Objets répartis
    - Résume des concepts
- Autres types de systèmes répartis
- Conclusion

Résumé des mécanismes en jeu

- Paquetages Remote_Call_Interface
  - Présents à un seul exemplaire dans une application
  - Localisation automatique
  - Indispensables pour faire communiquer des acteurs
- Objets répartis
  - Double aiguillage lors d'un appel de méthode
  - Peut créer des liens entre partitions ne se connaissant pas

Mais comment ça marche?



Comparaison des mécanismes de nommage

- Le nommage est
  - Manuel pour les systèmes basés sur l'envoi de messages (au mieux, fichier décrivant les services locaux)
  - Assisté pour les RPC de base
  - Largement assisté pour le système CORBA
  - Totalement automatique pour Ada

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

Autres systèmes répartis

- PVM
  - Système populaire, prévu pour le calcul scientifique
  - Basé sur l'envoi de message
  - Supporte les messages de groupe
  - Nécessite des appels explicites aux routines d'emballage et de déballage (pas de contrôle de type)

Autres systèmes répartis (2)

- DCOM
  - Objets répartis de Microsoft
  - Concurrent direct de CORBA
- RMI
  - Objets répartis de Java
  - Supporte la migration de code
  - Doit devenir compatible avec CORBA

Autres systèmes répartis (3)

- Erlang
  - Langage développé par Ericsson pour développer des autocommutateurs
  - Basé sur l'envoi de messages typés entre processus, sans distinction de localité
  - Transforme tout événement administratif (mort d'un processus, etc.) en message
  - Supporte la migration de code

Autres systèmes répartis (4)

- XML-RPC
  - Utilise des technologies normalisées:
    - XML
    - HTTP
  - Bénéficie d'un niveau de sécurité normalisé
    - HTTPS (SSL+X509)
- SOAP
  - Basé sur XML-RPC
  - Développé par Microsoft

Autres systèmes répartis (5)

- Les réseaux actifs
  - Ne se contentent plus d'exécuter du code dans les nœuds finaux
  - Transportent le code sur les routeurs qui en ont besoin (nouveaux protocoles)
  - Permettent de localiser des services équivalents par anycast (imprimante sur un réseau par exemple)

Coopération entre systèmes répartis

- Des points existent pour faire coopérer des types de systèmes répartis incompatibles
  - CIAO: exporter des services Ada 95 vers CORBA
  - DROOPI: intergiciel générique (CORBA, Ada 95, ...)
- Les systèmes répartis peuvent également converger
  - RMI va adopter IIOP (de CORBA)
  - GLADE (une implémentation de l'annexe des systèmes répartis d'Ada 95) va utiliser IIOP à travers DROOPI

Plan

- Généralités
- Classes de systèmes répartis
- Exemples de systèmes répartis
- Autres types de systèmes répartis
- Conclusion

Conclusion

- Les systèmes répartis sont à la mode
- Il existe de nombreuses manières de développer une application répartie
- Il faut considérer les différents facteurs
  - Le système doit-il être sûr?
  - Doit-il supporter plusieurs langages?
- Il est difficile de comparer deux concepts de systèmes répartis complexes