Modèle Client/Serveur orienté objet

Modèle Client/Serveur orienté objet

Introduction

Modèle Client/Serveur orienté objet

Introduction
Nommage

Modèle Client/Serveur orienté objet

Introduction
Nommage
Sécurité d’accès

Modèle Client/Serveur orienté objet

Introduction
Nommage
Sécurité d’accès
Durée de vie

Modèle Client/Serveur orienté objet

Introduction
Nommage
Sécurité d’accès
Durée de vie
Objets Concurrents

Modèle Client/Serveur orienté objet

Introduction
Nommage
Sécurité d’accès
Durée de vie
Objets Concurrents
Synchronisation

Modèle Client/Serveur orienté objet

Introduction
Nommage
Sécurité d’accès
Durée de vie
Objets Concurrents
Synchronisation
Middleware orienté objet

Introduction

les applications sont souvent en mode client/serveur

les clients font appel à des services offerts par des serveurs distants

l’API socket est vraiment de bas niveau et demande :

- l’allocation manuelle des sockets et des threads
- la sérialisation manuelle des appels de service distants
- l’emballage et le déballage des données des messages
- la sérialisation manuelle des objets complexes
- un manque d’annuaire et d’autres services ‹‹ middleware ››

Introduction

Les applications sont de plus en plus conçues selon une approche orientée objet.

Les « bonnes » propriétés de l'objet
(encapsulation, modularité, réutilisation, polymorphisme, composition)
vont être utilisées pour les applications réparties.

- L'unité de désignation et de distribution sera l'objet.
- But : permettre la communication directe et transparente entre objets répartis
  par exemple : `objetDistant.methode(parametres)`
- La transparence est uniquement pour le programmeur !

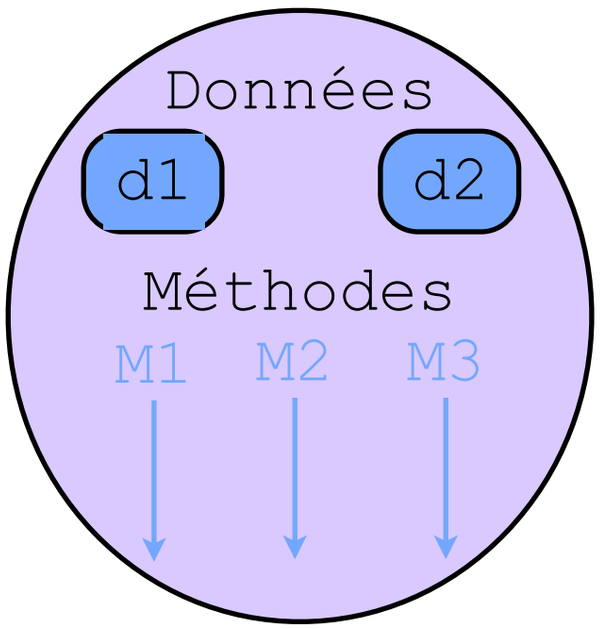
Introduction

Notion d'objet dans un environnement réparti

- Mieux structurer les applications
- Simplifier la conception, le développement, la maintenance
- Uniformiser l'accès au système au travers d'un langage concurrent et réparti

Objet = données + méthodes + référence

- données : variables
- méthodes : traitements
- référence : moyen d'accès à l'objet



Introduction

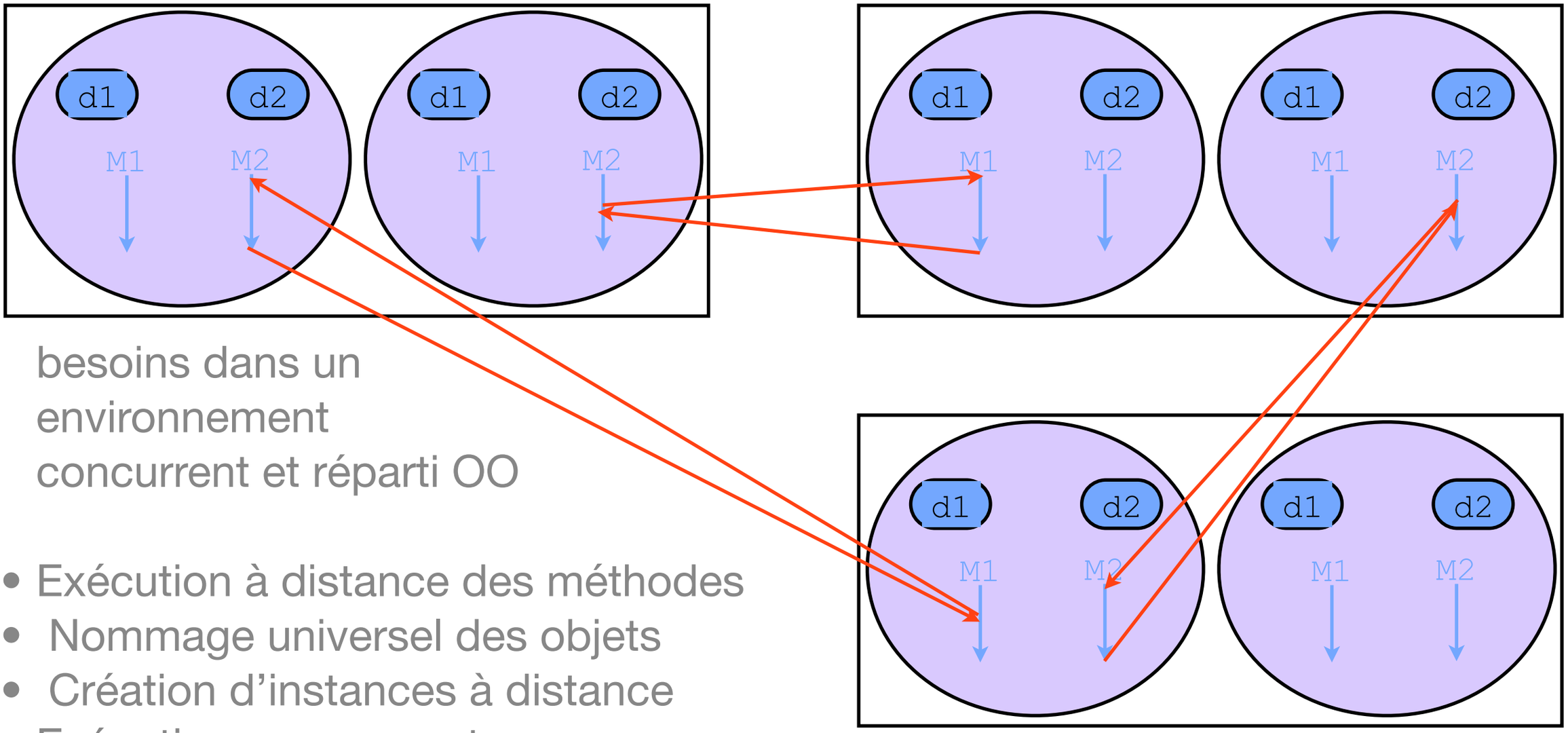
Quelques rappels

- Encapsulation : l'objet protège les données en ne les rendant visibles qu'au travers de méthodes
- Instanciation : les objets sont créés à partir de moules (les classes)
- Héritage : les classes peuvent être définies en dérivant des classes existantes

Classification (P. Wegner)

- Langage basé objet : ayant un mécanisme d'encapsulation (Ada)
- Langage basé classe : ayant en plus un mécanisme d'instancitation (Simula)
- Langage orienté objet : ayant en plus un mécanisme d'héritage (Smalltalk, C++, Java)

Introduction



besoins dans un environnement concurrent et réparti OO

- Exécution à distance des méthodes
- Nommage universel des objets
- Création d'instances à distance
- Exécutions concurrentes
- Synchronisation des exécutions
- Migration des instances

Nommage

Objets à désigner

- le site d’exécution
- le serveur
- la procédure

La désignation doit être globale et indépendante de la localisation
reconfiguration possible des services (pannes, régulation de charge,...)

Désignation

- Statique : la localisation du serveur est connue à la compilation
- Dynamique : la localisation n ’est pas connue à la compilation
  séparation des connaissances du nom de service et de son adresse
  grande indépendance au niveau de l’implantation

Nommage

Identifier les objets dans un environnement réparti

- deux objets différents sur le même site ou sur des sites différents ne doivent pas avoir la même identité (on parle de référence d'objet)
- la référence sert à « retrouver » l'objet pour pouvoir invoquer ses méthodes
- c'est une généralisation de la notion de pointeur à un environnement réparti

Deux techniques principales

- un ID sans rapport avec la localisation généré par une fonction mathématique
- un ID en deux parties : son site de création + un numéro local unique

Recherche d'un objet à partir de sa référence

- annuaires de localisation centralisés ou répartis, ou diffusion
- interrogation du site contenu dans la référence + liens de poursuite

Nommage

La localisation du serveur : le client doit localiser et se « lier » (bind) avec le serveur
La localisation se fait en trois étapes :

- enregistrer le serveur auprès du serveur de désignation
- le client interroge le serveur de désignation
- le client accède au service sur le serveur

Plusieurs possibilités

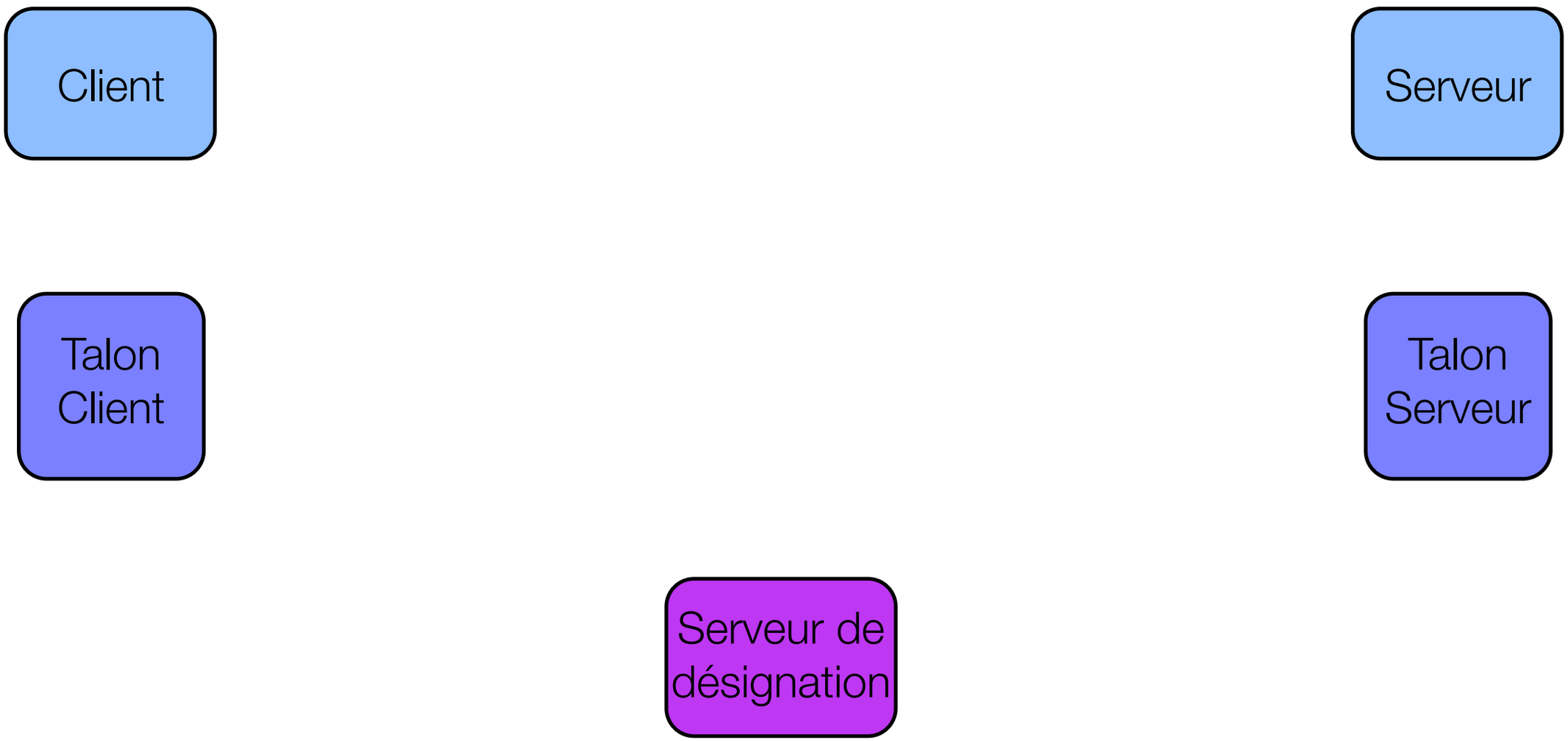
- liaison statique (pas d ’appel à un serveur de nom)
- liaison au premier appel (appel du serveur de nom lors du premier appel
- liaison à chaque appel (appel du serveur de nom à chaque appel)

Nommage

les différentes étapes de la liaison

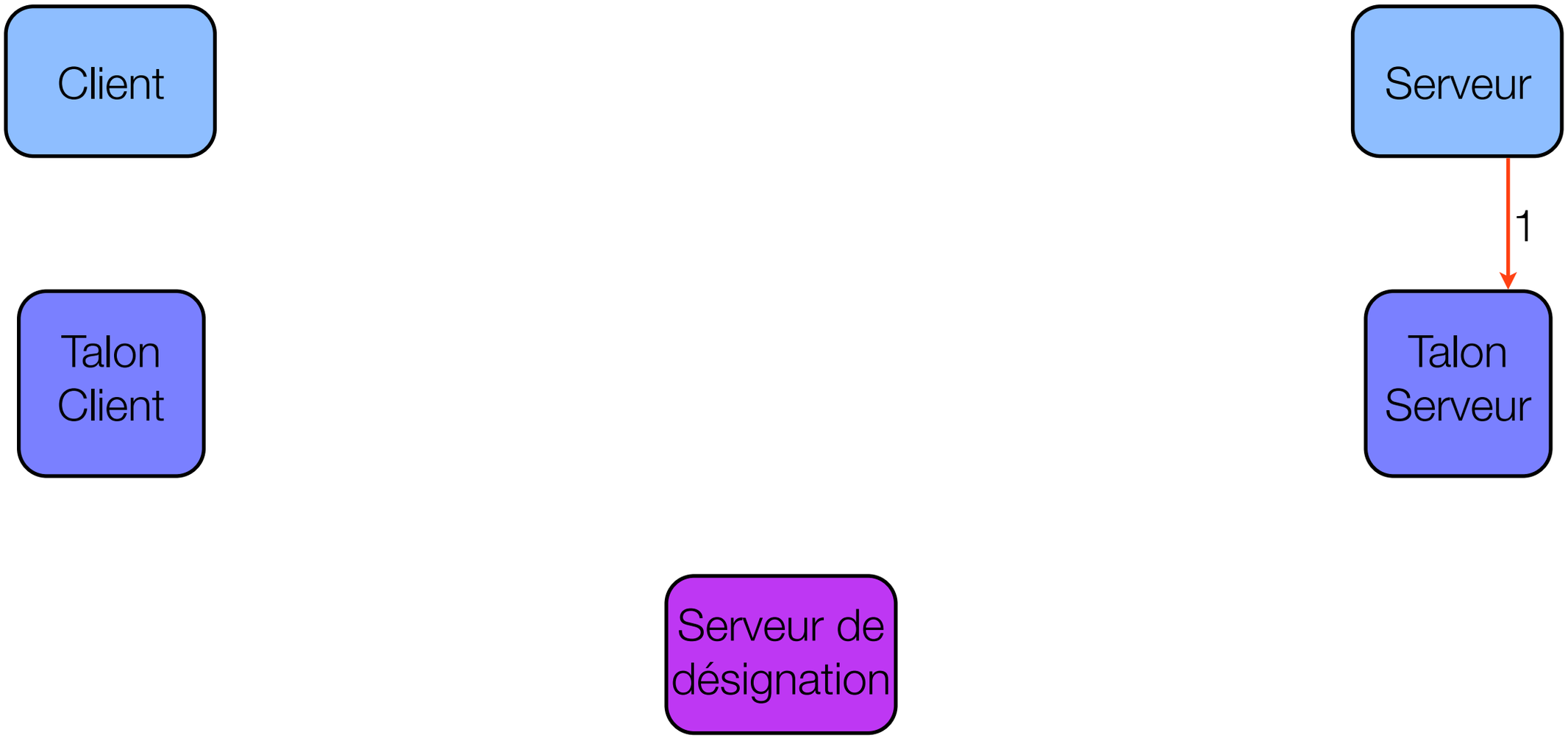
Nommage

les différentes étapes de la liaison



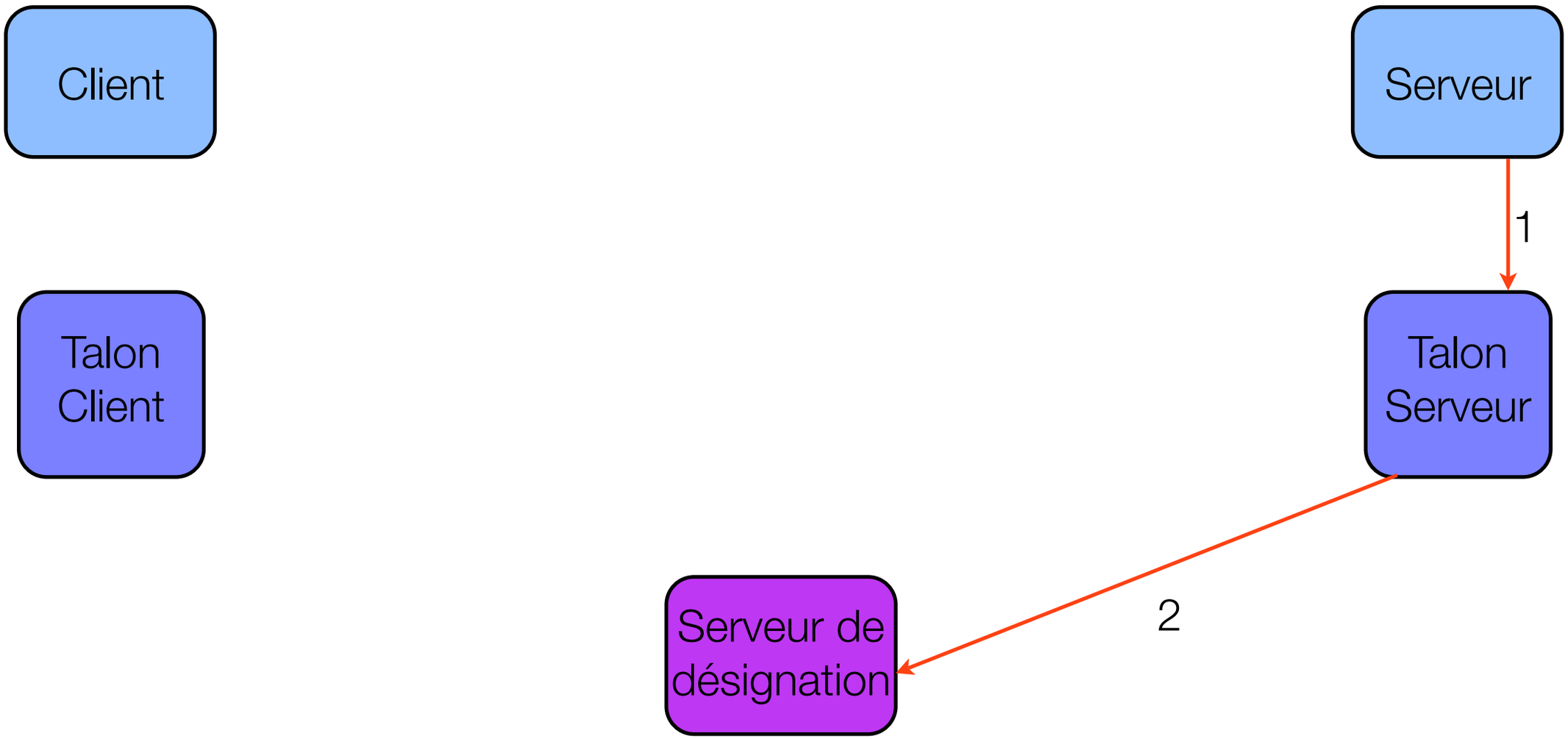
Nommage

les différentes étapes de la liaison



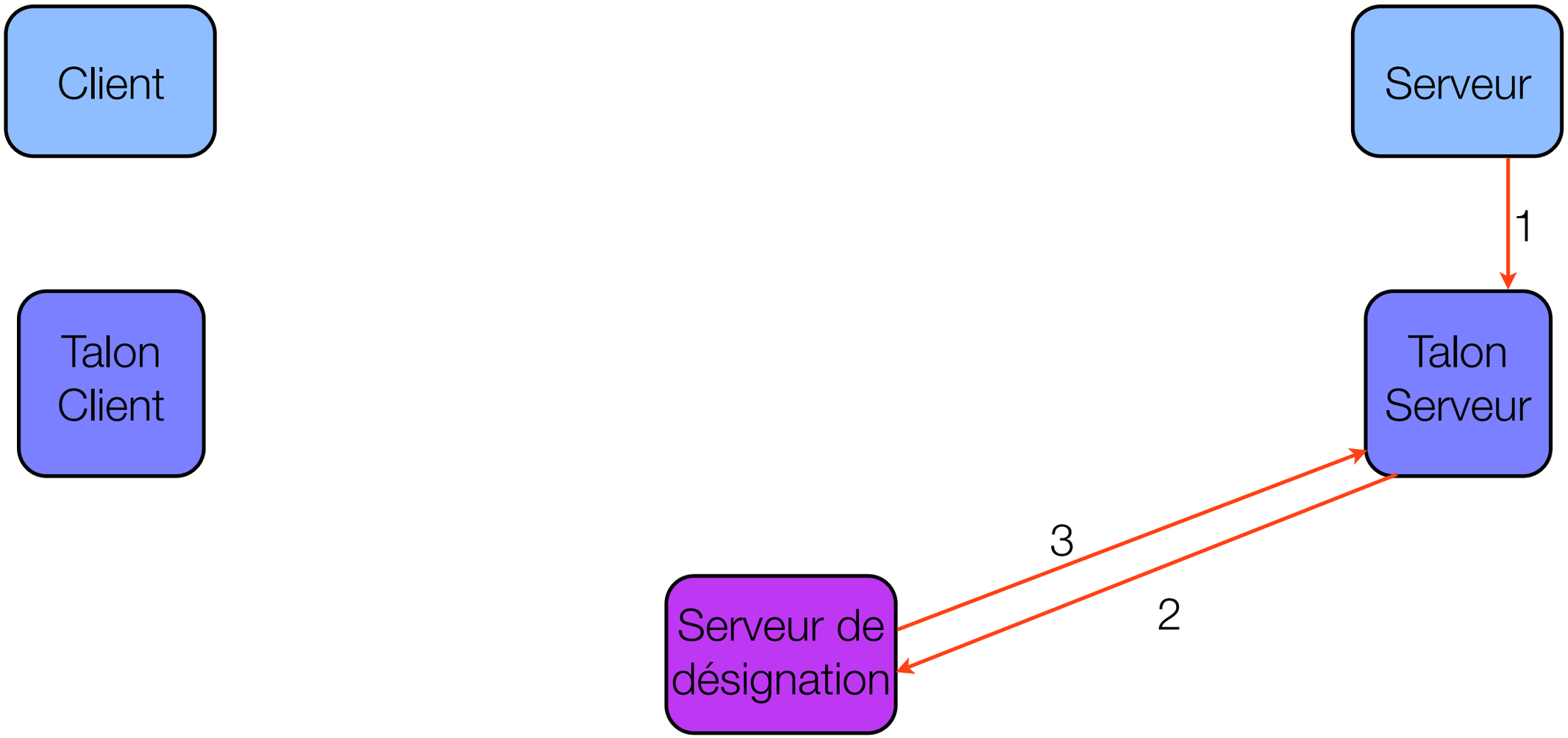
Nommage

les différentes étapes de la liaison



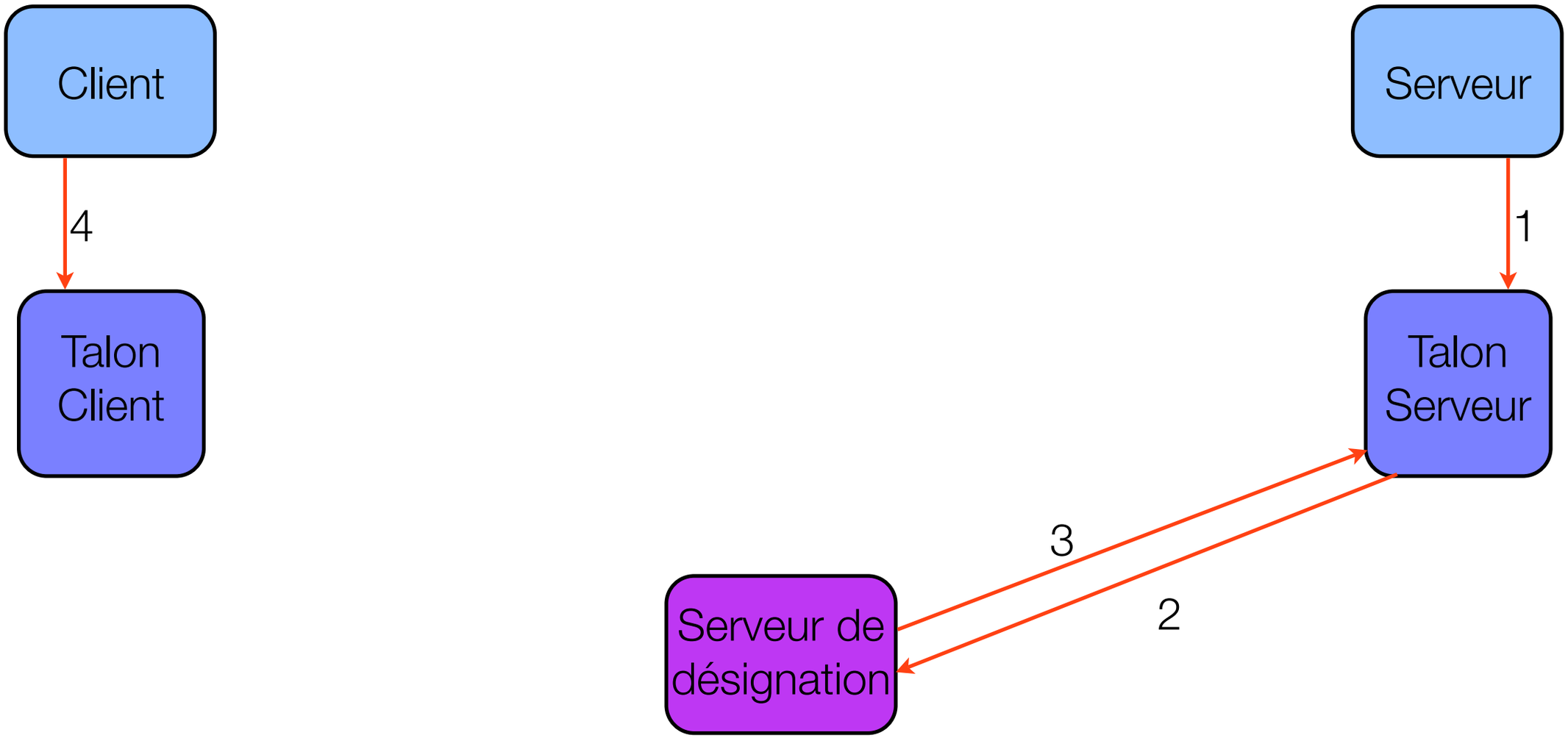
Nommage

les différentes étapes de la liaison



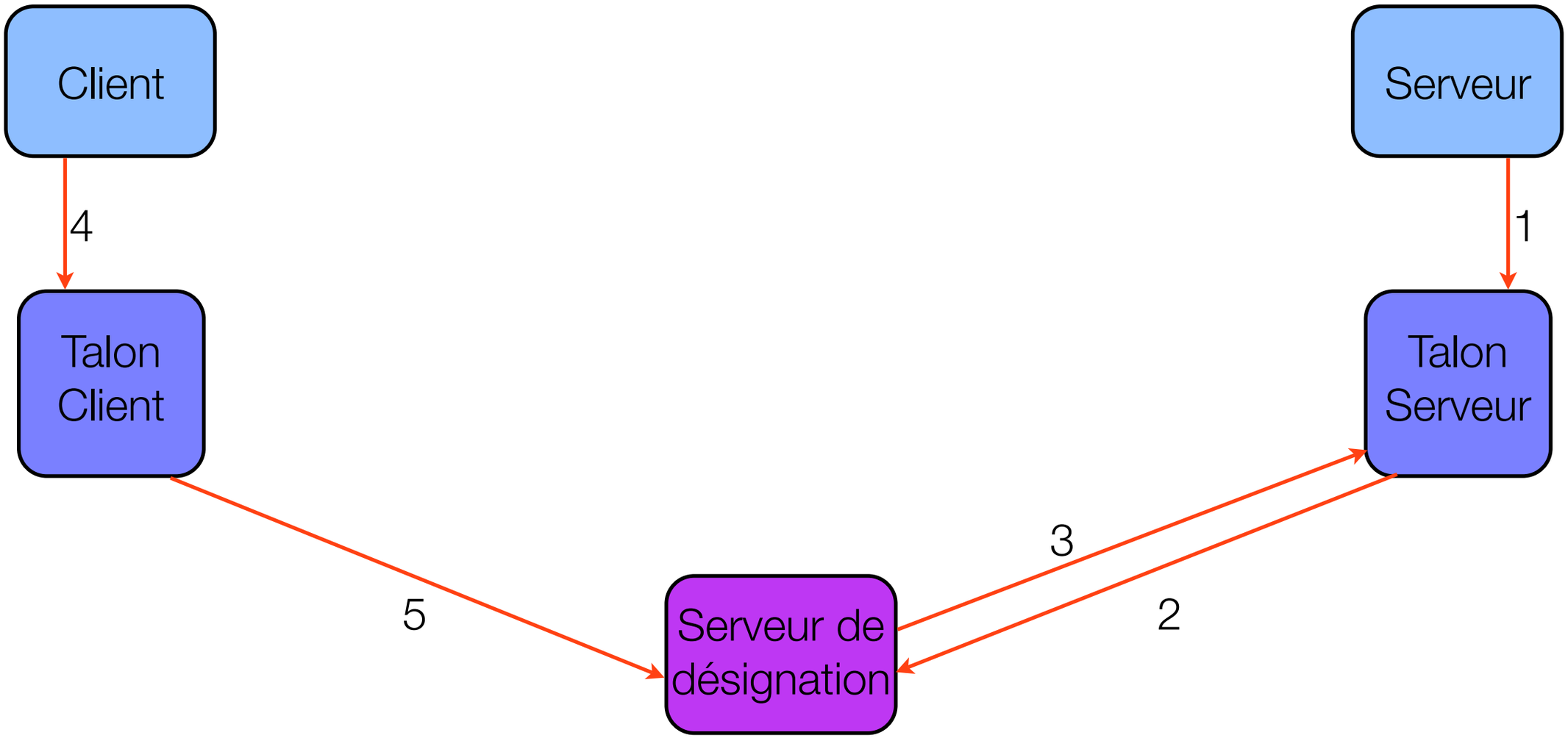
Nommage

les différentes étapes de la liaison



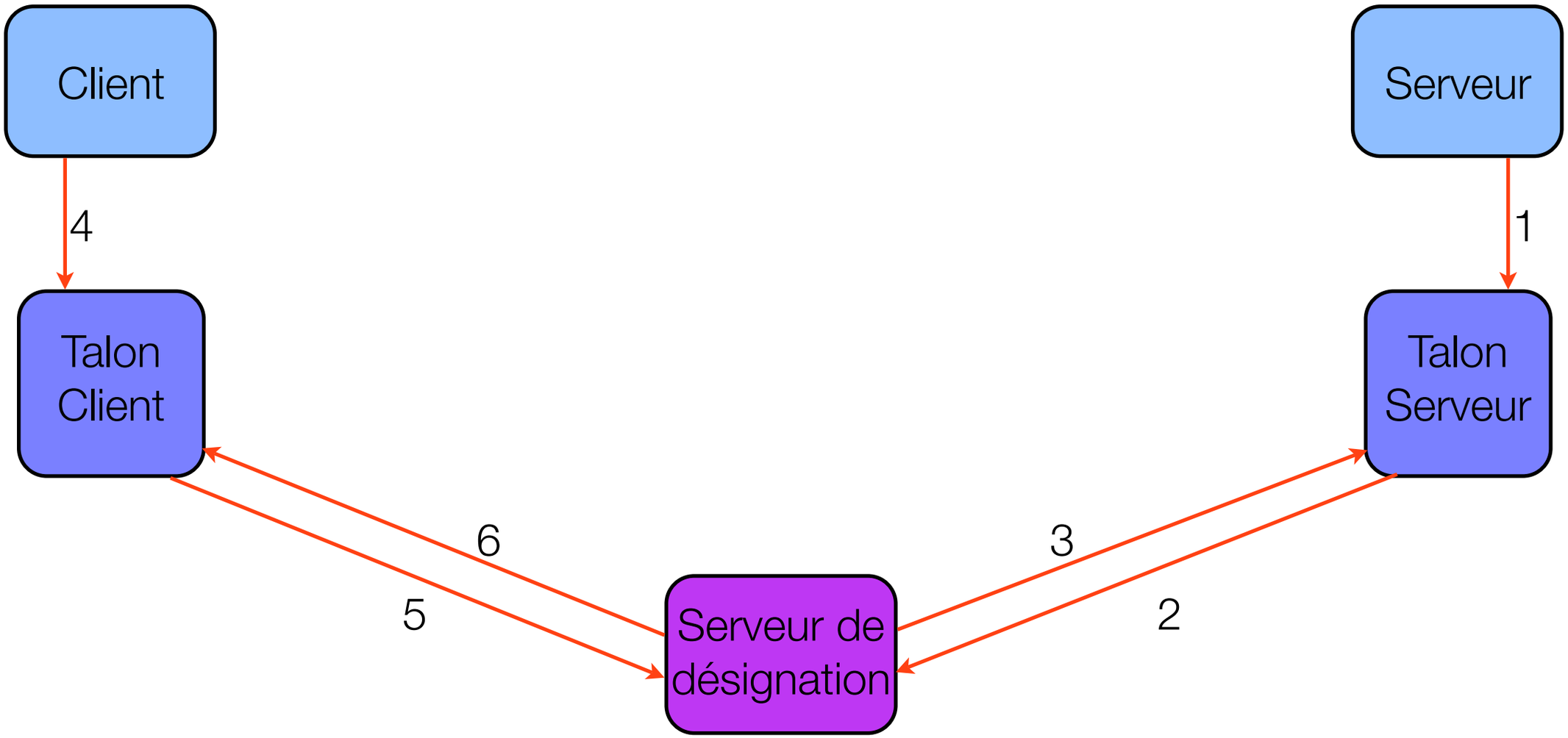
Nommage

les différentes étapes de la liaison



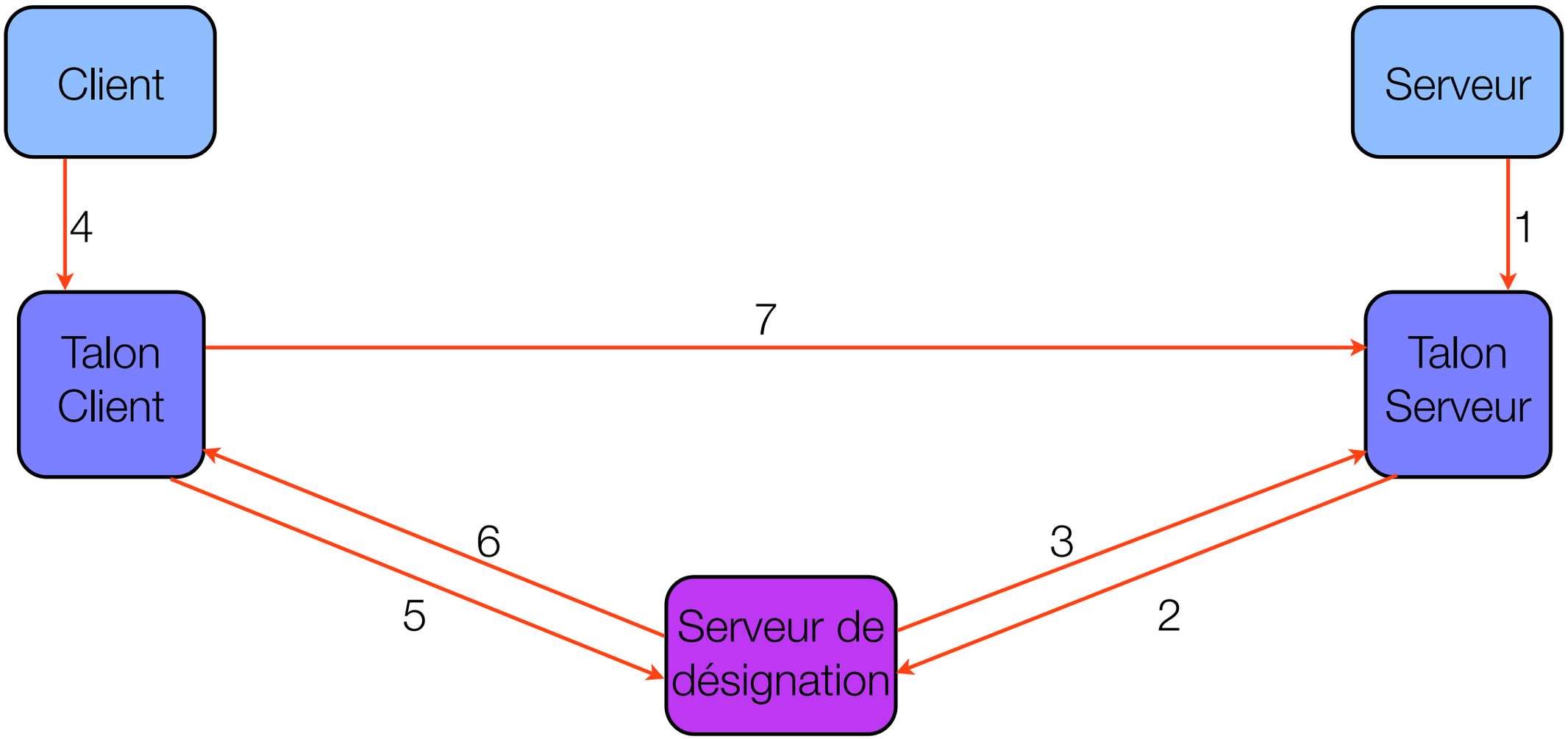
Nommage

les différentes étapes de la liaison



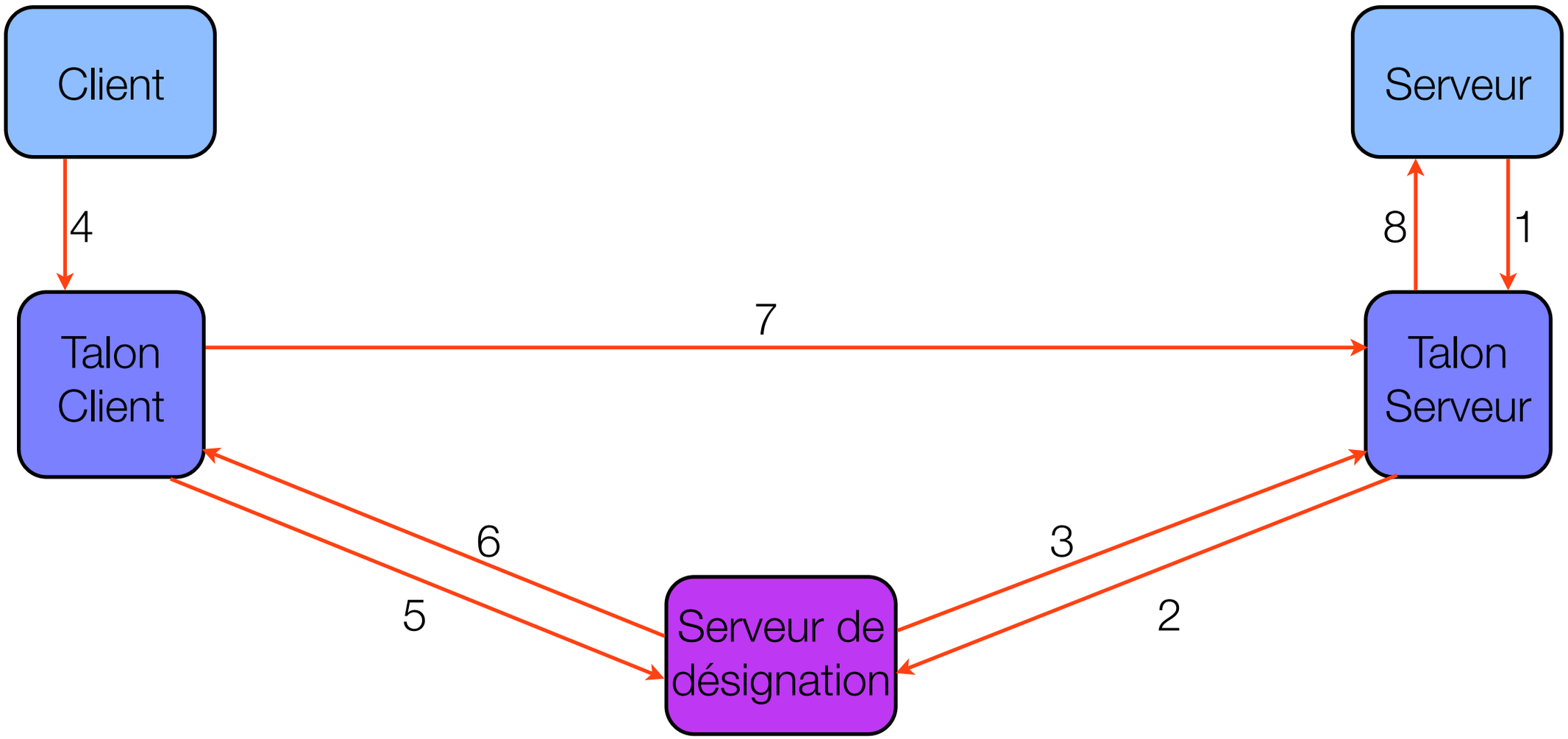
Nommage

les différentes étapes de la liaison



Nommage

les différentes étapes de la liaison



Sécurité

gérer le partage des objets dans un environnement réparti

Pour des raisons de sécurité, l'accès à certains objets peut être restreint

- en fonction de l'identité de l'objet appelant
  ex : seuls les accès en provenance de l'intranet sont autorisés
- à partir d'une liste de contrôle d'accès
  ex : mot de passe, mécanisme de clés de session

La restriction peut

- interdire complètement l'accès à l'objet
- fournir une vue dégradée
  ex : autoriser les méthodes de consultation et interdire celles qui modifient l'état de l'objet

De nombreuses informations sont à ajouter aux objets.

Durée de vie

Objets temporaires : leur durée de vie correspond à celle de leur créateur

Objets persistants : l'objet persiste tant qu'il n'est pas détruit,
la persistance peut être limitée par la durée de vie du système
ou s'étendre au delà des redémarrages

Problèmes à résoudre :

- sauvegarde d'informations multiples (type, classe, données)
- gérer les lieux de stockage (mémoire, disque, réseau)
- assurer que l'objet reste accessible quelque soit son lieu de stockage

Choix de conception :

- tous les objets sont ils persistants ?
- un objet est il créé persistant ?
- un objet temporaire peut il devenir persistant ? vice-versa ?
- un objet référencé par un objet persistant est il persistant ?

Objets Concurrents

Degrés de concurrence

possibilité d'exécuter simultanément plusieurs activités
rq: le terme parallélisme est lié aux machines multi-processeurs

4 degrés de concurrence (adapté de P. Wegner, J.P. Briot et al.) :

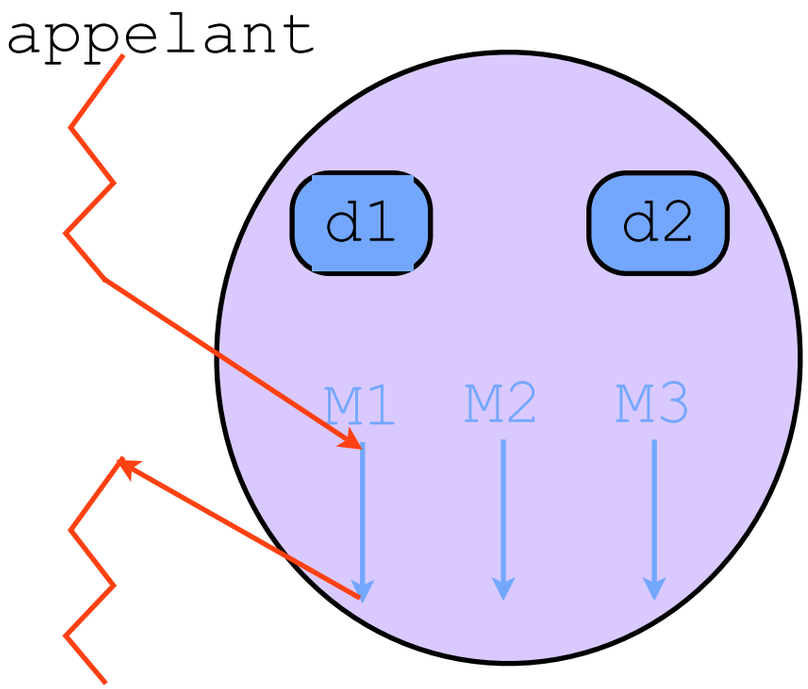
- séquentiel : une seule méthode est exécutée à la fois par l'objet
- quasi-concurrent : plusieurs activations de méthodes coexistent, mais au plus une seule n'est pas suspendue
- concurrent : plusieurs activités peuvent s'exécuter, mais le programmeur peut restreindre le degré de concurrence (synchronisation)
- complètement-concurrent : concurrence sans aucune restriction (objet sans états)

Objets Concurrents

deux façons d'associer un objet et une activité (thread)

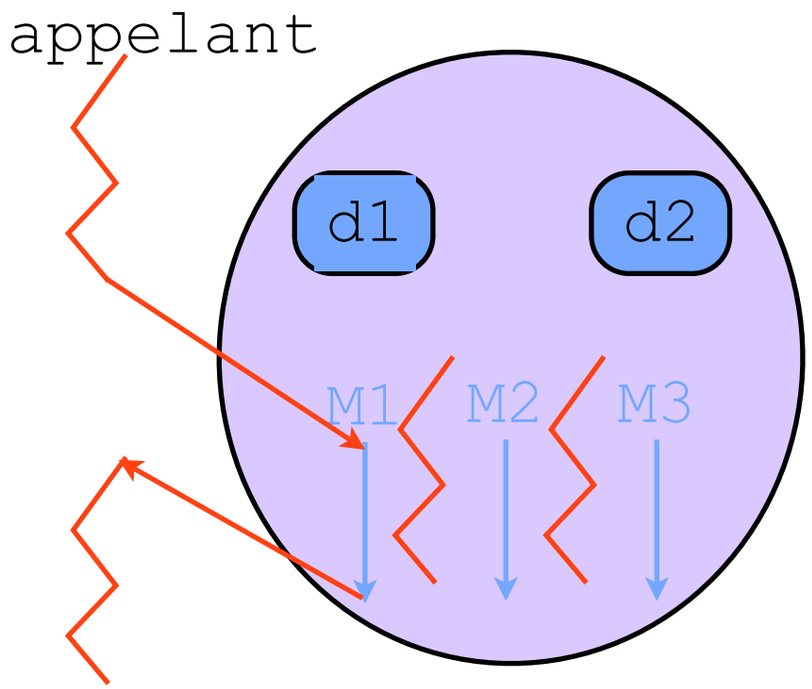
Objet passif

- manipulé de façon explicite
- orthogonal à l'objet
- se «plaque» sur les méthodes



Objet actif

- une ou plusieurs activités dédiées à l'objet exécutent les méthodes



Synchronisation

Restreindre la concurrence d'accès à une ressource partagée afin d'en maintenir la cohérence
ex : exclusion mutuelle, lecteurs/écrivain, transaction

- Synchronisation Comportementale
  en fonction de l 'état des **données** de l 'objet
  ex : Pile avec empile et dépile
  - vide : empile
  - 1/2 : empile et dépile
  - plein : dépile

- Synchronisation Intra-objet
  en fonction de l 'état d '**exécution** de l 'objet
  ex : Fichier avec lire et écrire
  - soit + lire simultanément
  - soit 1 seul écrire

Synchronisation

- objets sémaphores : méthodes `P` et `V`
- objets moniteurs : méthodes `synchronized`, `wait` et `notify` (ex `Java`)
- expressions de chemin: avec les opérateurs `,` `*` `|` `//` on spécifie les séquences valides d'exécutions de méthodes
- gardes : chaque méthode est associée à une condition booléenne
- remplacement de comportement : chaque objet a plusieurs « comportements », chacun correspondant à l'exécution d'une méthode on spécifie avec la primitive `become` le comportement suivant
- approches à états : chaque objet est associé à un ensemble d'états chaque état est associé à un sous-ensemble de méthodes pouvant être exécutées

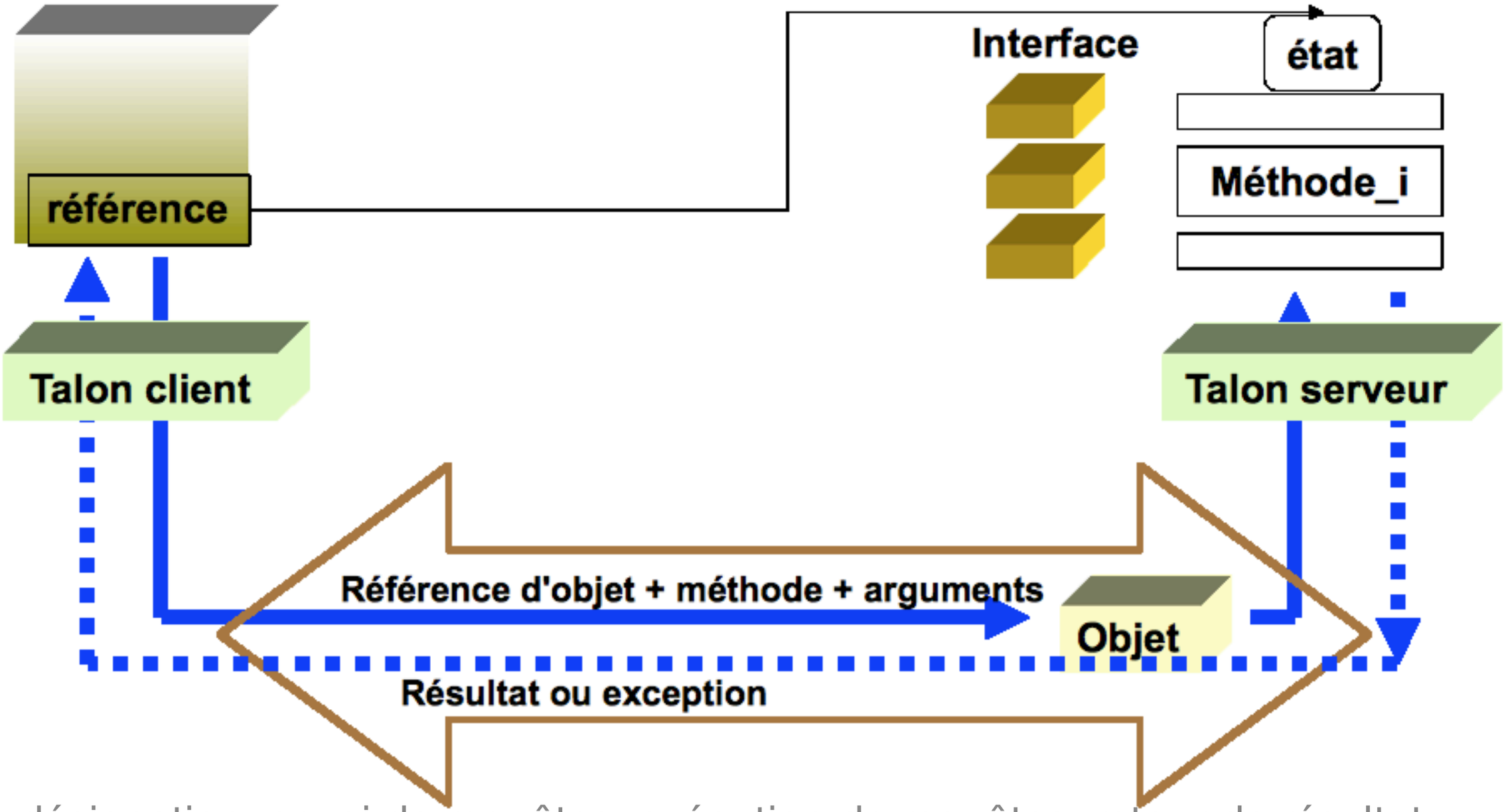
Synchronisation

- Synchronisation inter-objets
  dépendances entre objets demandent des mécanismes de synchronisation
  exemple : un transfert entre deux objets comptes bancaires

- Il faut assurer qu'un ensemble de 2 ou plusieurs invocations de méthodes compte1.depot(50); compte2.retrait(50); s'effectue complètement avec les propriétés ACID ou pas du tout

- problématique de la sérialisabilité et des moniteurs transactionnels
  intégration du moniteur dans le système réparti objet avec un :
  - protocole de validation (2PC ou 3PC)
  - mécanisme de verrouillage des ressources
  - mécanisme de détection des conflits
  - mécanisme de traitement des conflits

Middleware orienté objet : appel de procédure à distance



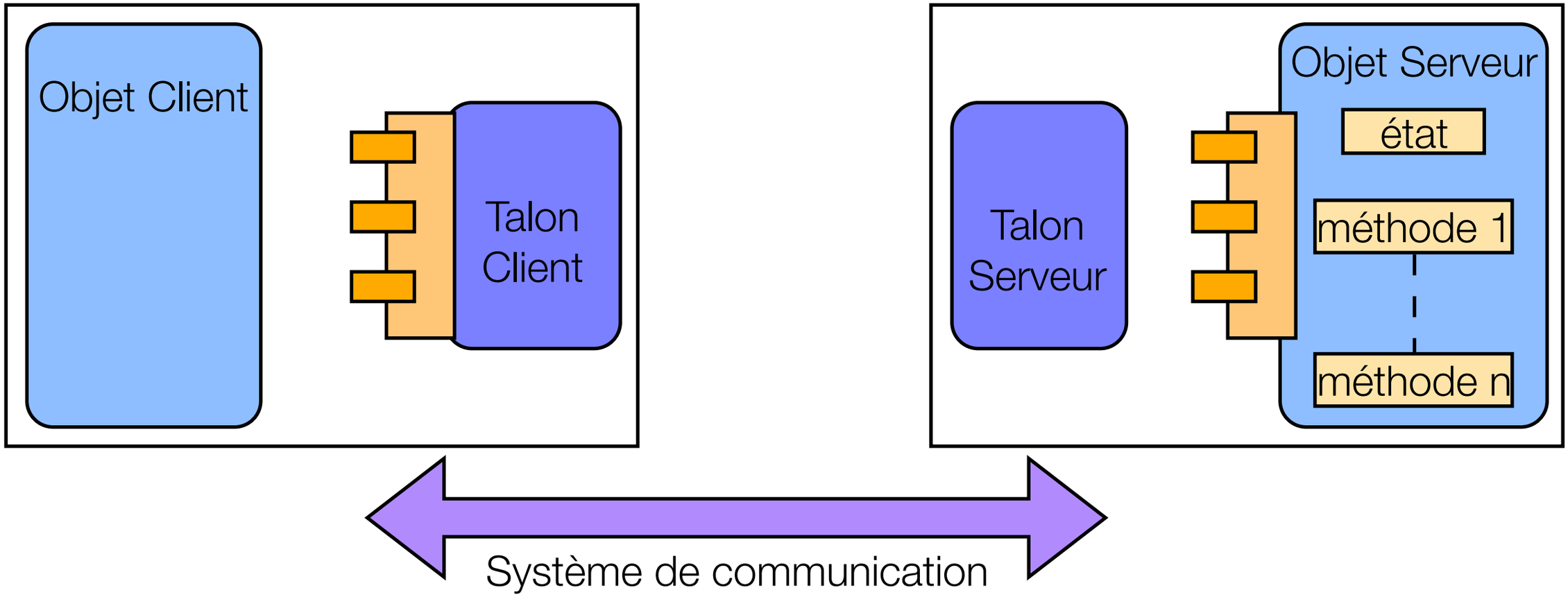
désignation, envoi de requêtes, exécution de requêtes, retour de résultat

Middleware orienté objet : appel de procédure à distance

les éléments d'une « invocation »

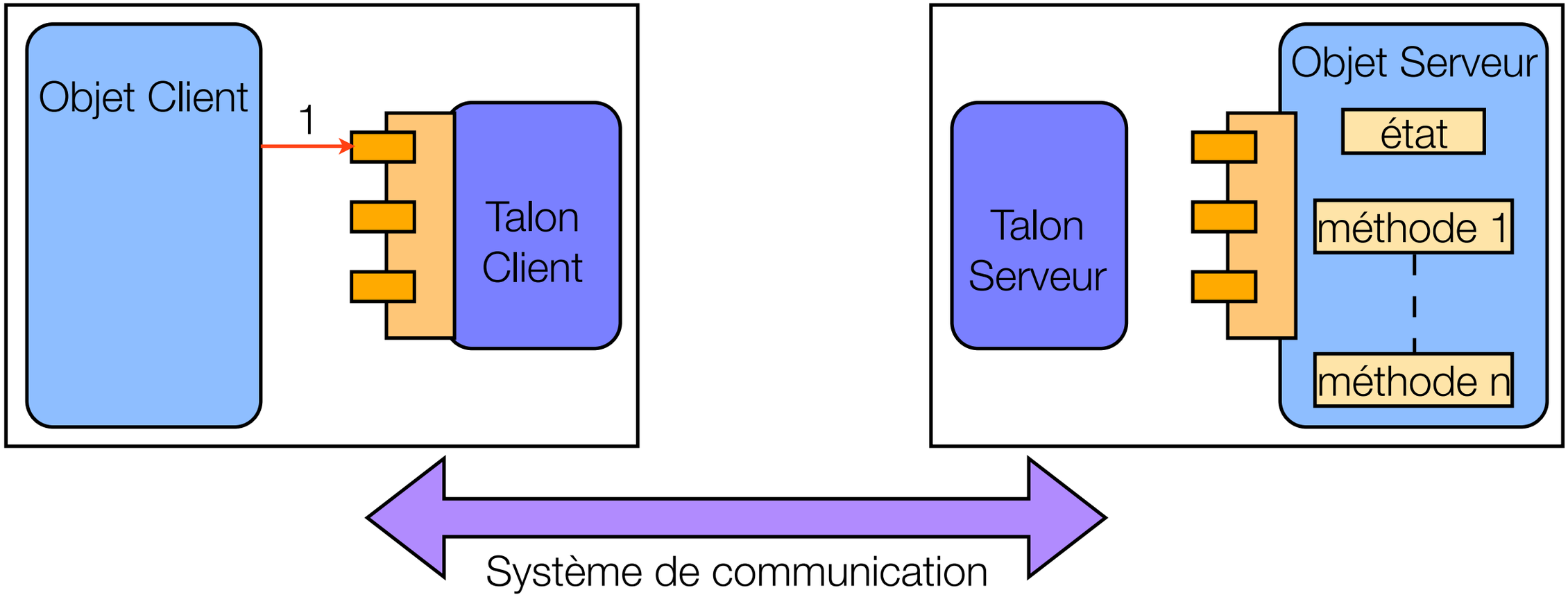
- référence d'objet (« pointeur » universel)
- identification d'une méthode
- paramètres d'appel et de retour
  - passage par valeur : types élémentaires et types construits
  - passage par référence
- possibles exceptions en retour
- objets « langage »
  représentation propre au langage : instance d'une classe
  exemple : Java RMI
- objets « système »
  représentation « arbitraire » définie par l'environnement d'exécution
  exemple : CORBA

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



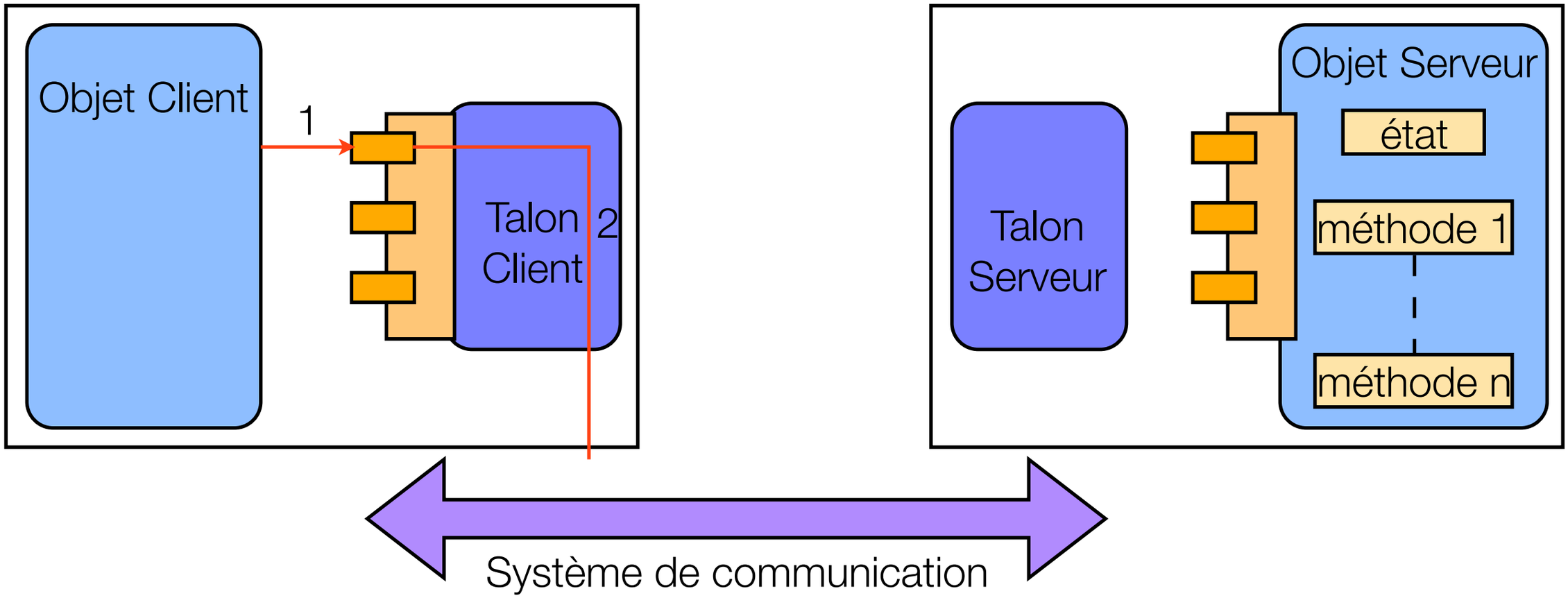
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



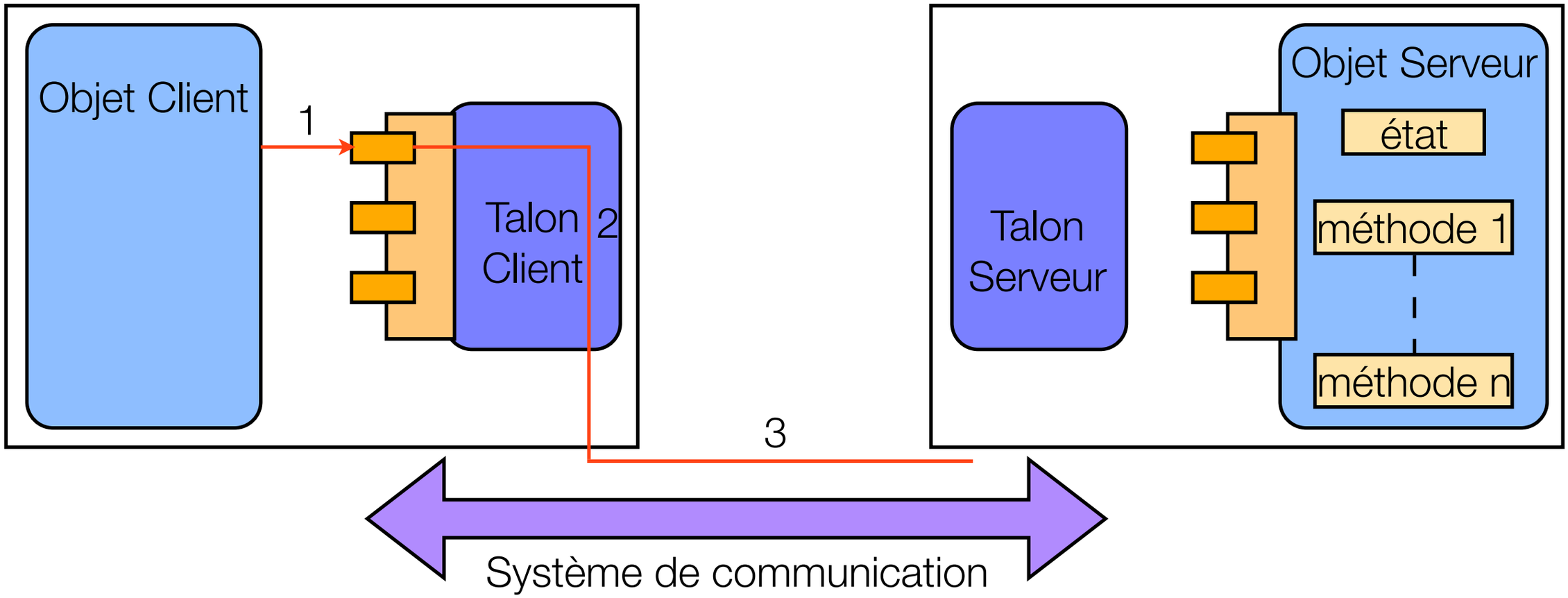
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



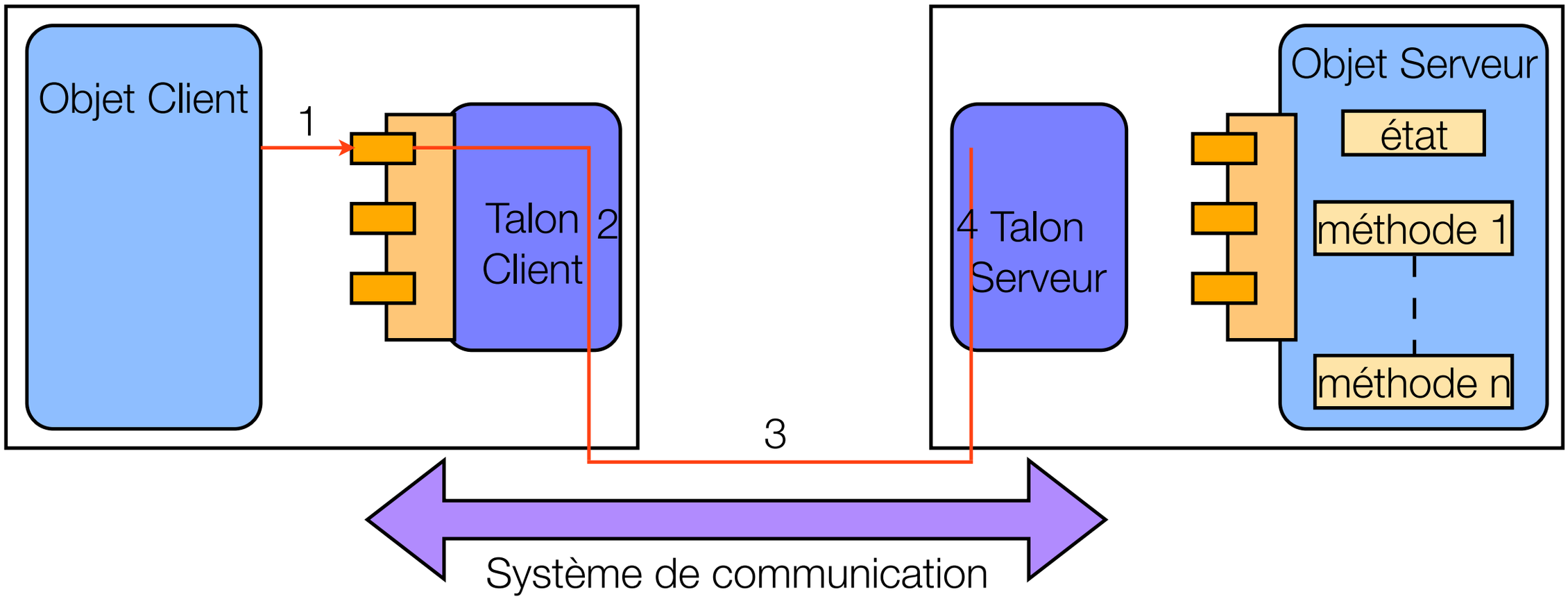
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



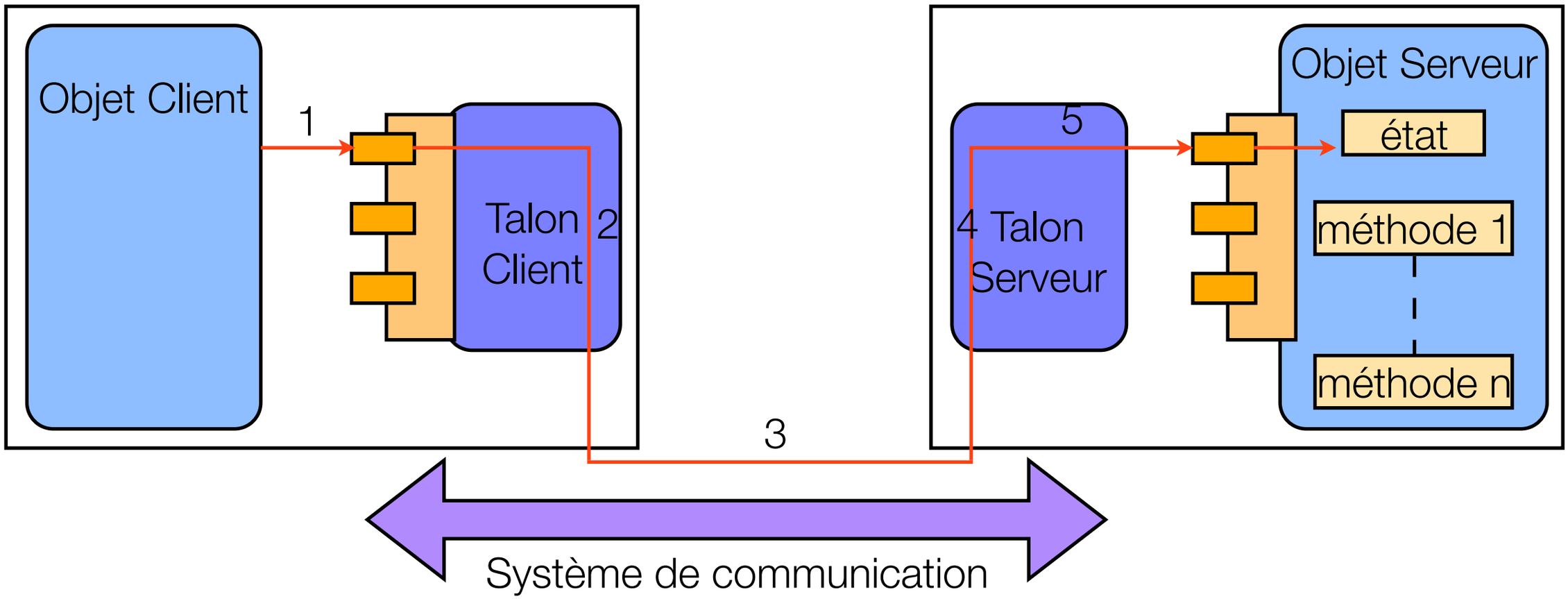
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



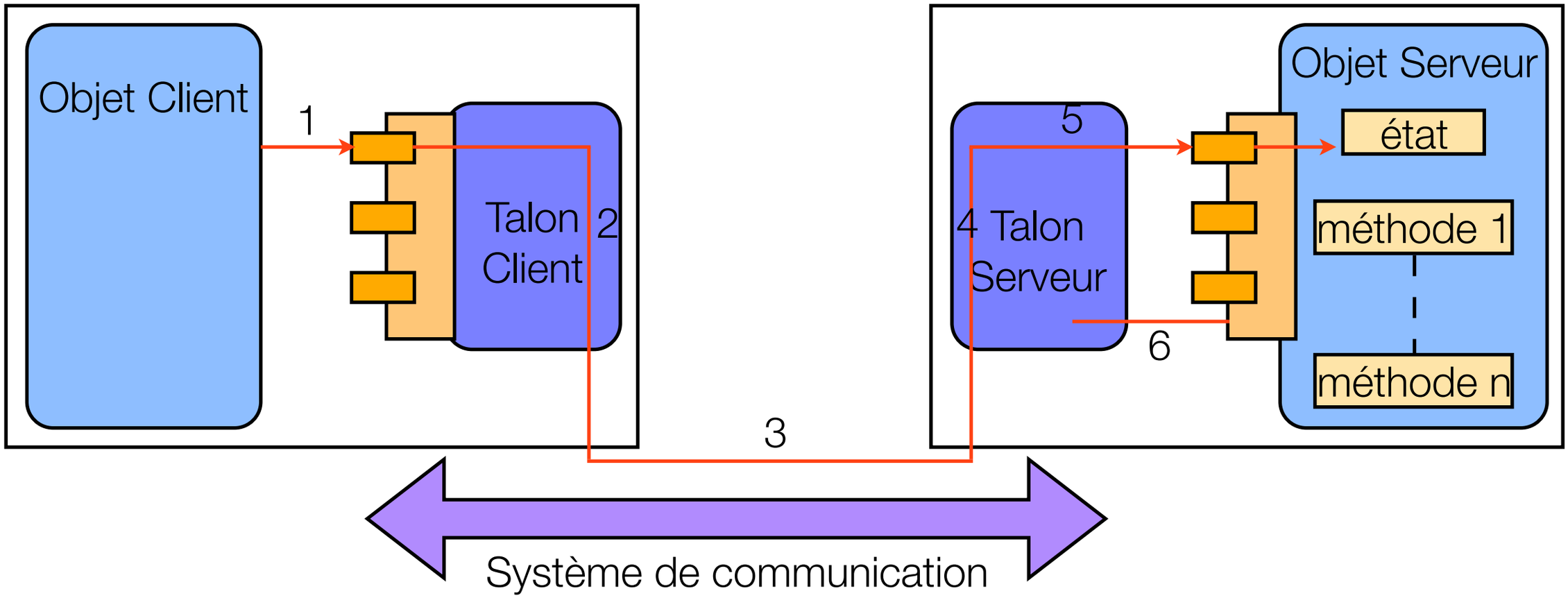
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



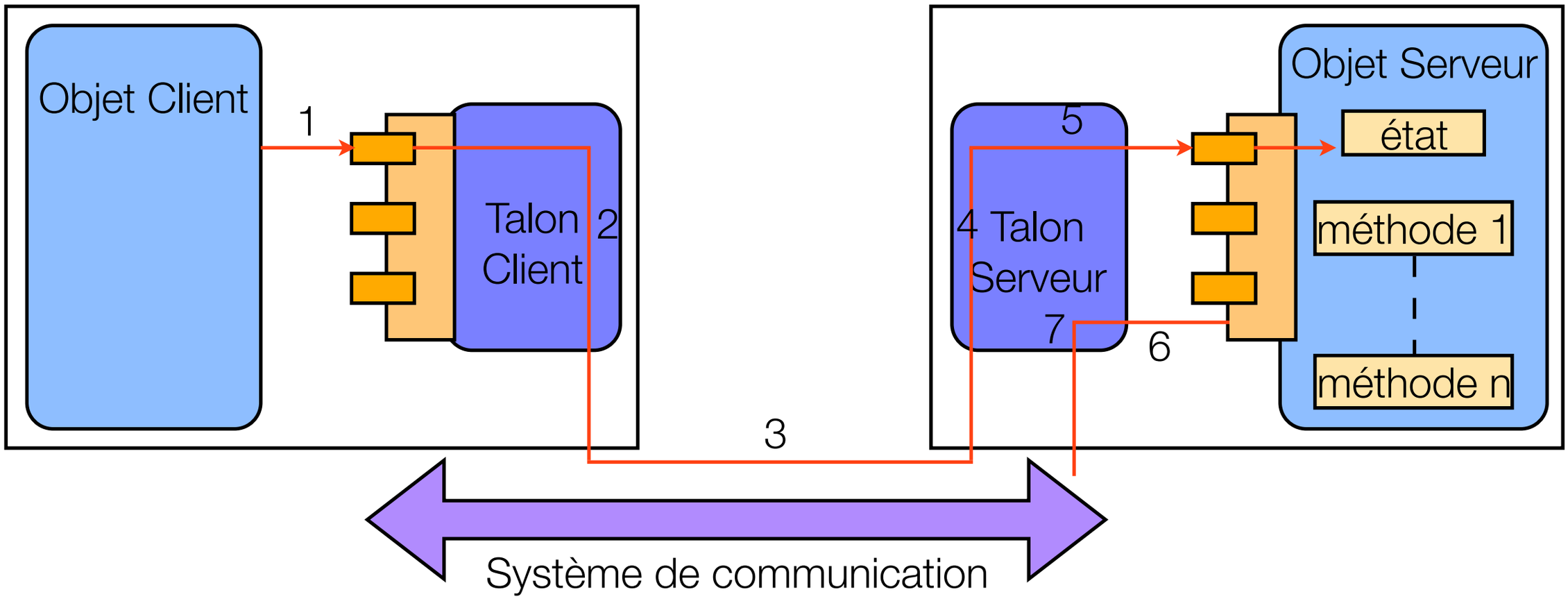
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



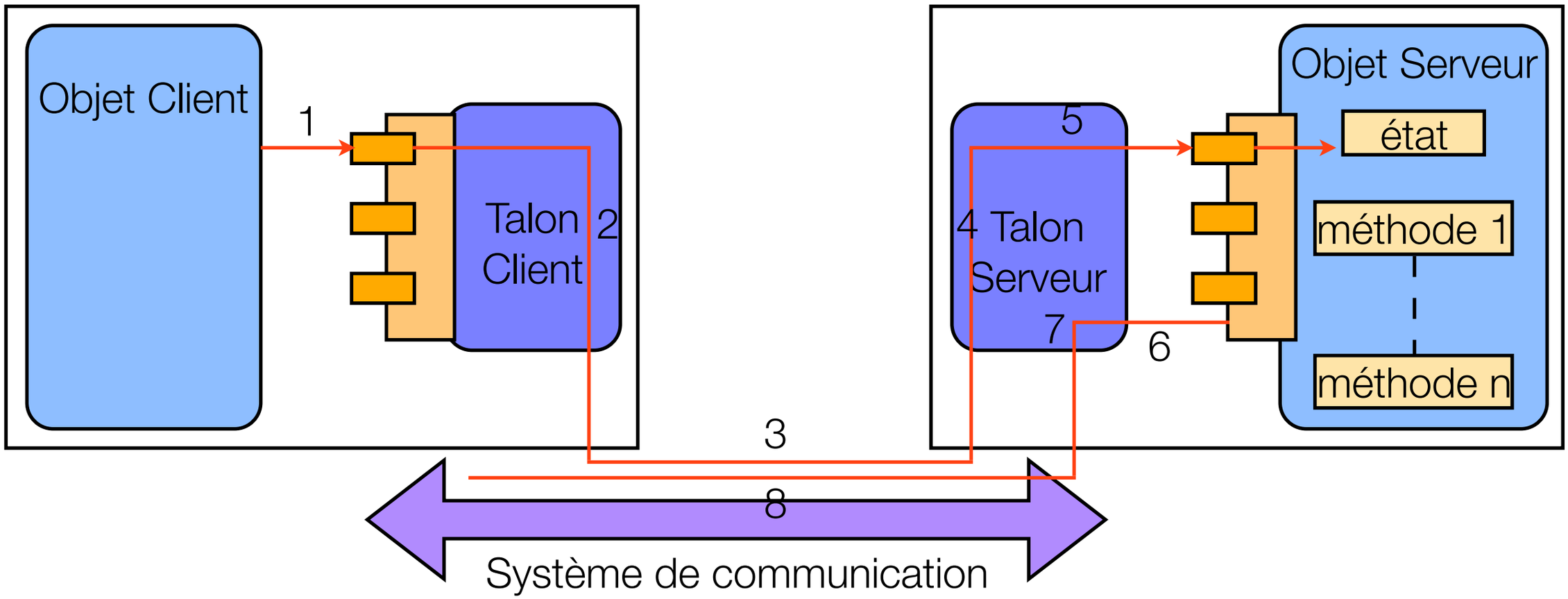
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



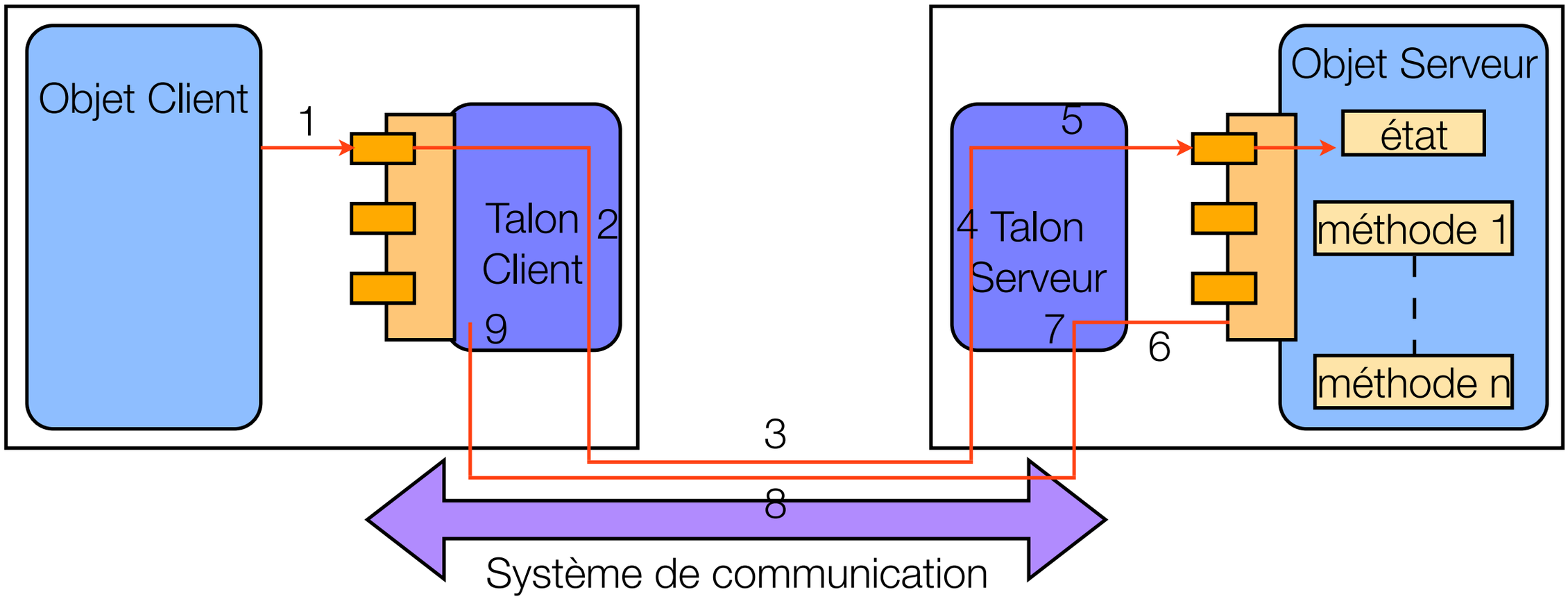
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



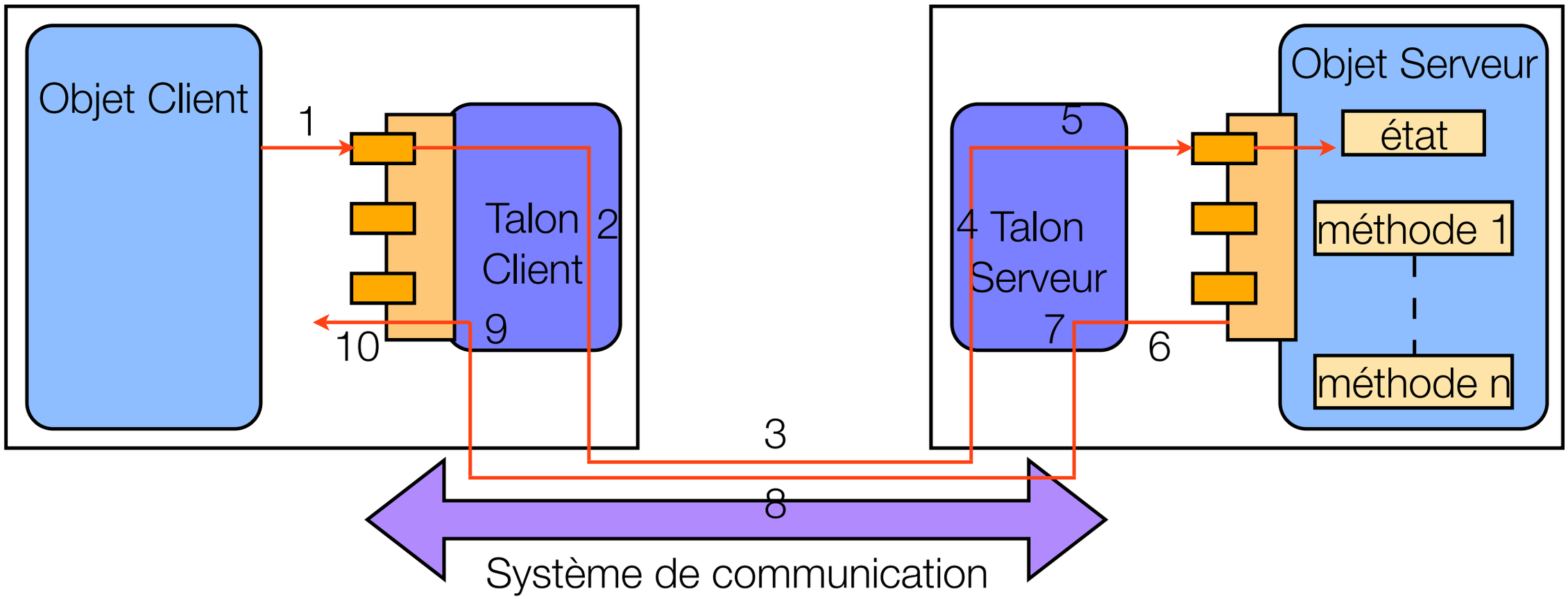
1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet : principe des mandataires objets (proxy, talon, souche)



1. invocation d'une méthode
2. emballage des paramètres
3. transport de l'invocation
4. déballage des paramètres
5. invocation de l'objet
6. retour de l'invocation locale
7. emballage des résultats
8. transport des résultats
9. déballage des résultats
10. retour de l'invocation distante

Middleware orienté objet

Rôle du talon client

- Représentant local de l'objet distant : référence sur l'objet distant
- Emballage des appels de méthodes locaux en messages à destination du talon serveur
- Déballage des résultats ou des exceptions contenus dans un message retourné par le talon serveur
- Contrôle de la bonne utilisation des signatures du talon à la compilation / l'exécution : contrôle des messages du protocole

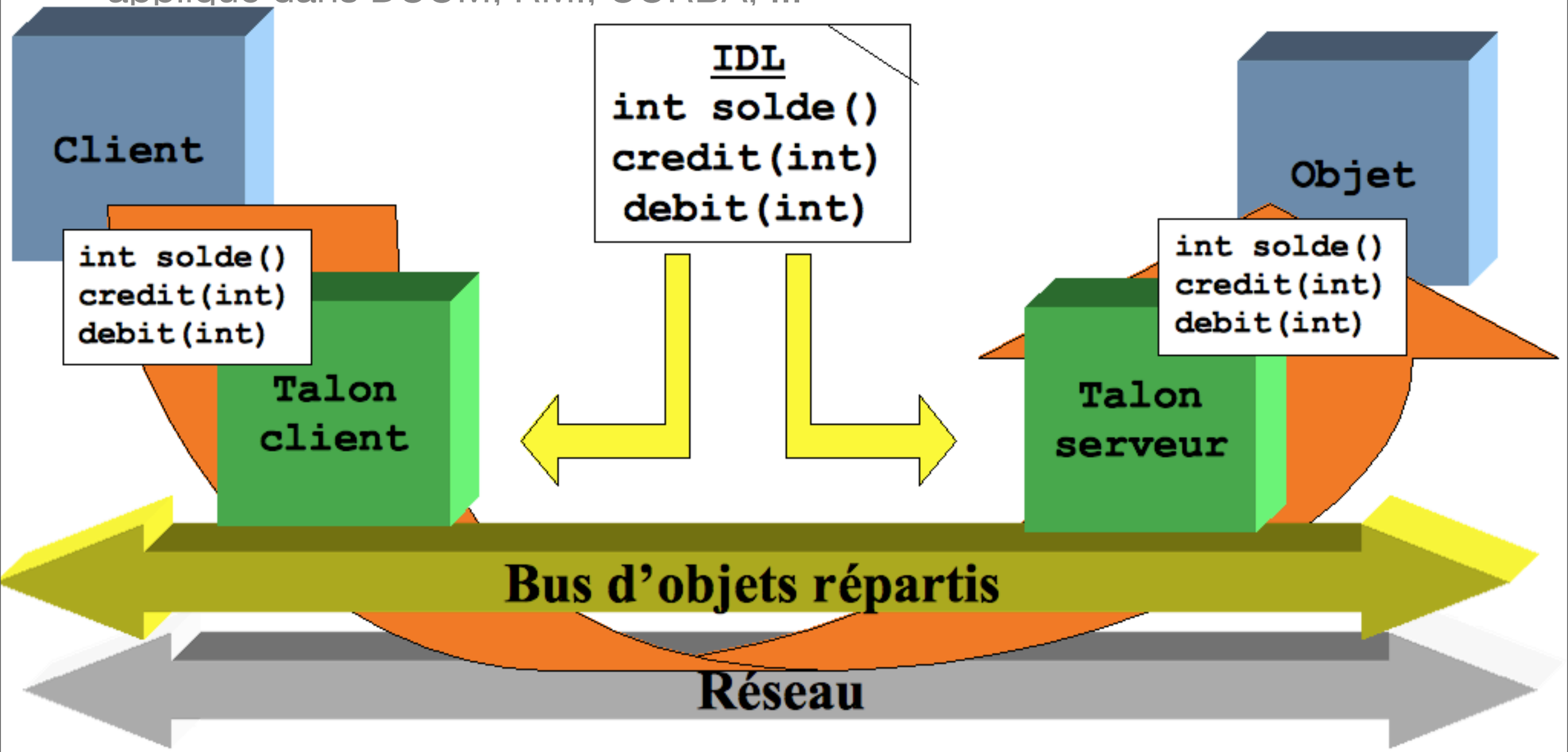
Middleware orienté objet

Rôle du talon serveur

- Passerelle entre le réseau et l'objet serveur : rend l'objet serveur accessible à distance
- Déballage des messages émis par les talons clients
- Appel local des méthodes de l'objet serveur
- Emballage des résultats ou exceptions dans un message à destination du talon client

Middleware orienté objet

Le bus d'objets réparti prend en charge de la fonction de communication appliqué dans DCOM, RMI, CORBA, ...



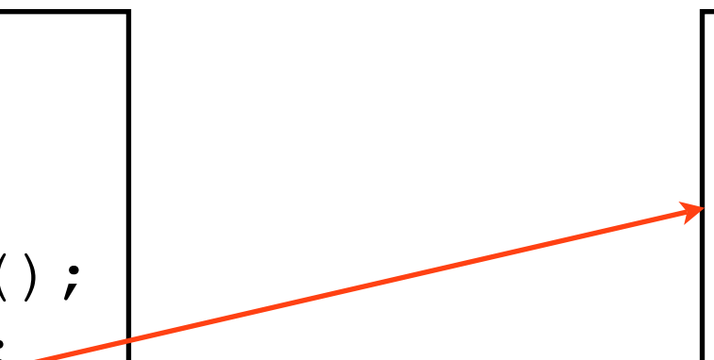
Middleware orienté objet : la prise en charge de la fonction de communication

Talon serveur

```
if(op == «credit»)
{
   int m = receive();
   objet.credit(m);
   return;
} else
. . .
```

Objet

```
credit(int montant)
{
   ...code métier...
   balance+=montant;
}
```



- Le développeur ne code pas la communication
  - gestion des sockets, du protocole applicatif, des threads
  - déballage des arguments et emballage des résultats
- Indépendance vis-à-vis de la couche de transport
  - TCP/IP, ATM, . . ., mémoire partagée, . . .
- Amélioration de la qualité logicielle par génération

Middleware orienté objet : le rôle du langage IDL

Interface Definition Language = langage de définition des interfaces des objets répartis

il permet de décrire la signature des méthodes publiques et accessibles à distance

- nom de la méthode
- type du résultat et des paramètres
- liste des exceptions

caractérisation des messages du protocole applicatif entre le client et le serveur

compilation de l'idl pour une production automatique des talons clients et serveurs

Middleware orienté objet : le rôle du bus d'objets répartis

- Implantation d'un protocole de transport des appels de méthode à distance
- Bufférisation des messages
- Allocation des canaux de communications
  - sockets mais aussi «pipes», mémoire partagée, . . .
  - des adresses physiques, e.g. des ports
  - gestion d'un pool de sockets, . . .
- Gestion de pools de threads serveurs
- Autres services . . .

Middleware orienté objet

Bus pour communication synchrone entre objets distants
mais peut aussi fournir d'autres fonctions

- annuaires de recherche
  - stockage des références d'objets distants
- communication asynchrone
- diffusion d'événements
- activation à la volée des objets : objets présents en mémoire seulement si invoqués
- Sécurité
- Transactions réparties
- . . .

Middleware orienté objet

- Les solutions industrielles

- Produits de Microsoft : système d'exploitation Windows partout !

- Java RMI pour Java Remote Method Invocation :
  produit de SUN Microsystems
  langage Java partout !

- CORBA pour Common Object Request Broker Architecture
  norme du consortium Object Management Group
  interopérabilité !