Agents Mobiles/Code Mobile

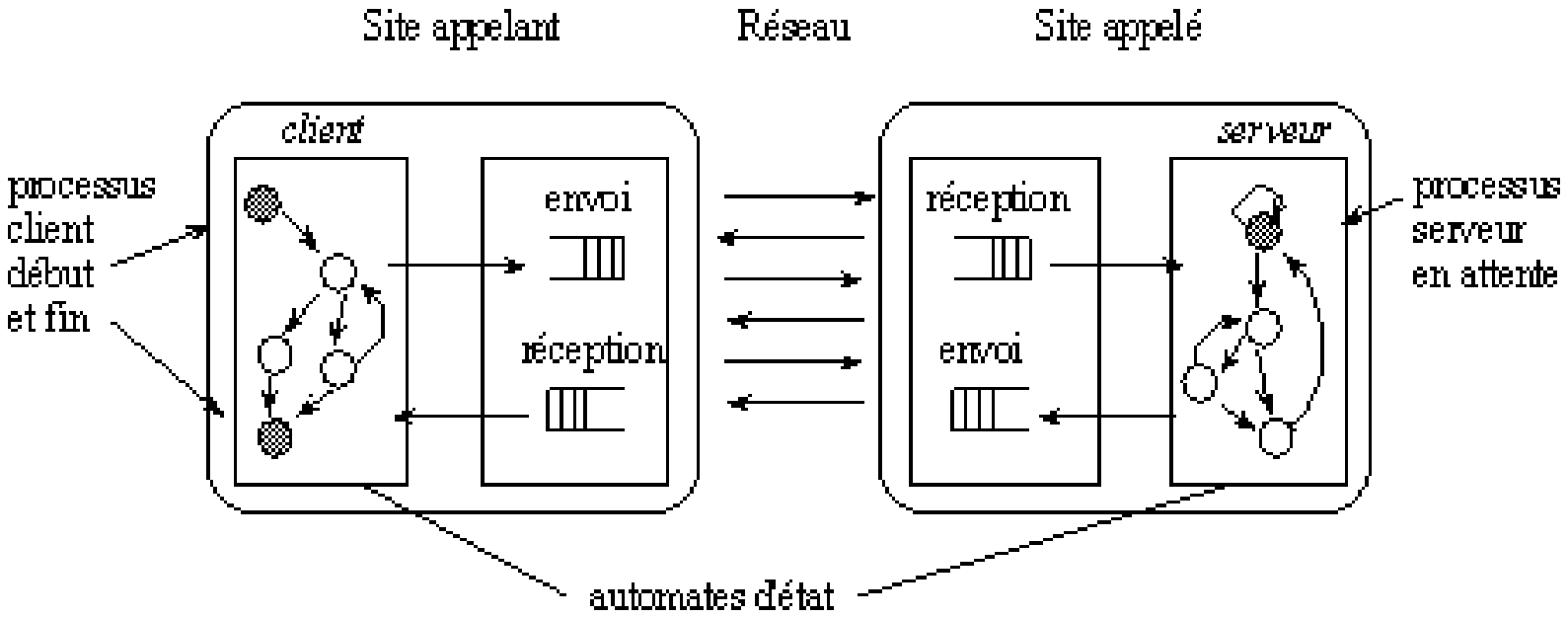
- Modèles d'exécution répartie et agents mobiles
- Infrastructure pour les agents mobiles
- Expérimentations
- Conclusion

Modèles d’exécution répartie

- Échange de messages
- Code référençable
- Évaluation distante
- Code à la demande

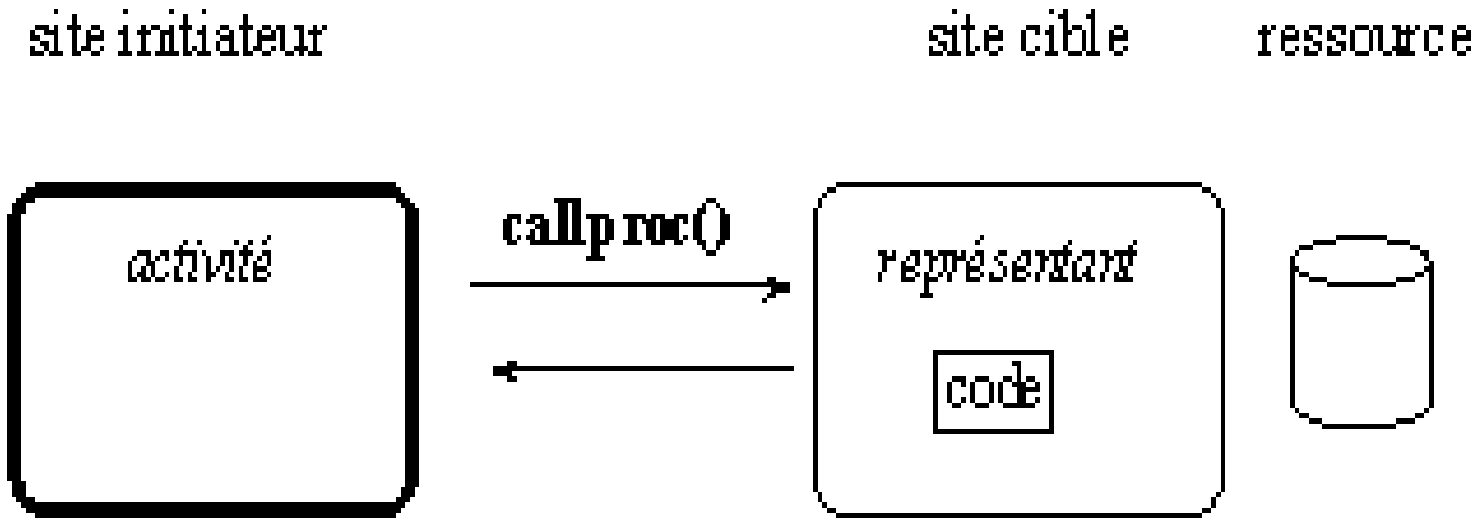
Échange de messages

- Mécanisme puissant et souple
- Le programmeur doit prendre en compte les problèmes de communication et de synchronisation
- Mécanisme de base dans les noyaux Chorus et Mach
- Très utilisé pour les applications de calcul parallèle (MPI)



Code référençable

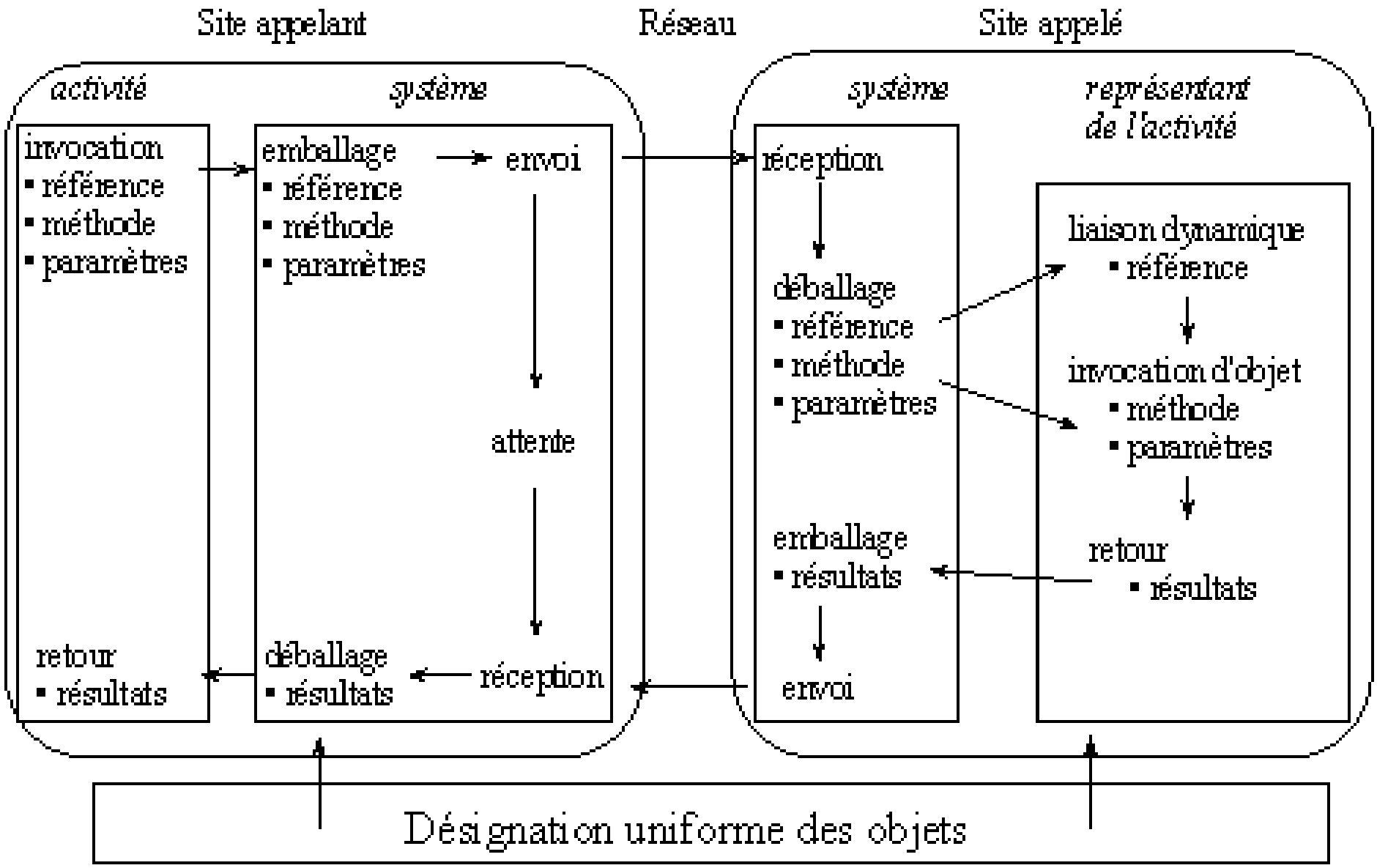
- RPC (NFS), RMI, HTTP/CGI
- Synchrone
- Grain fin
- Réseau local



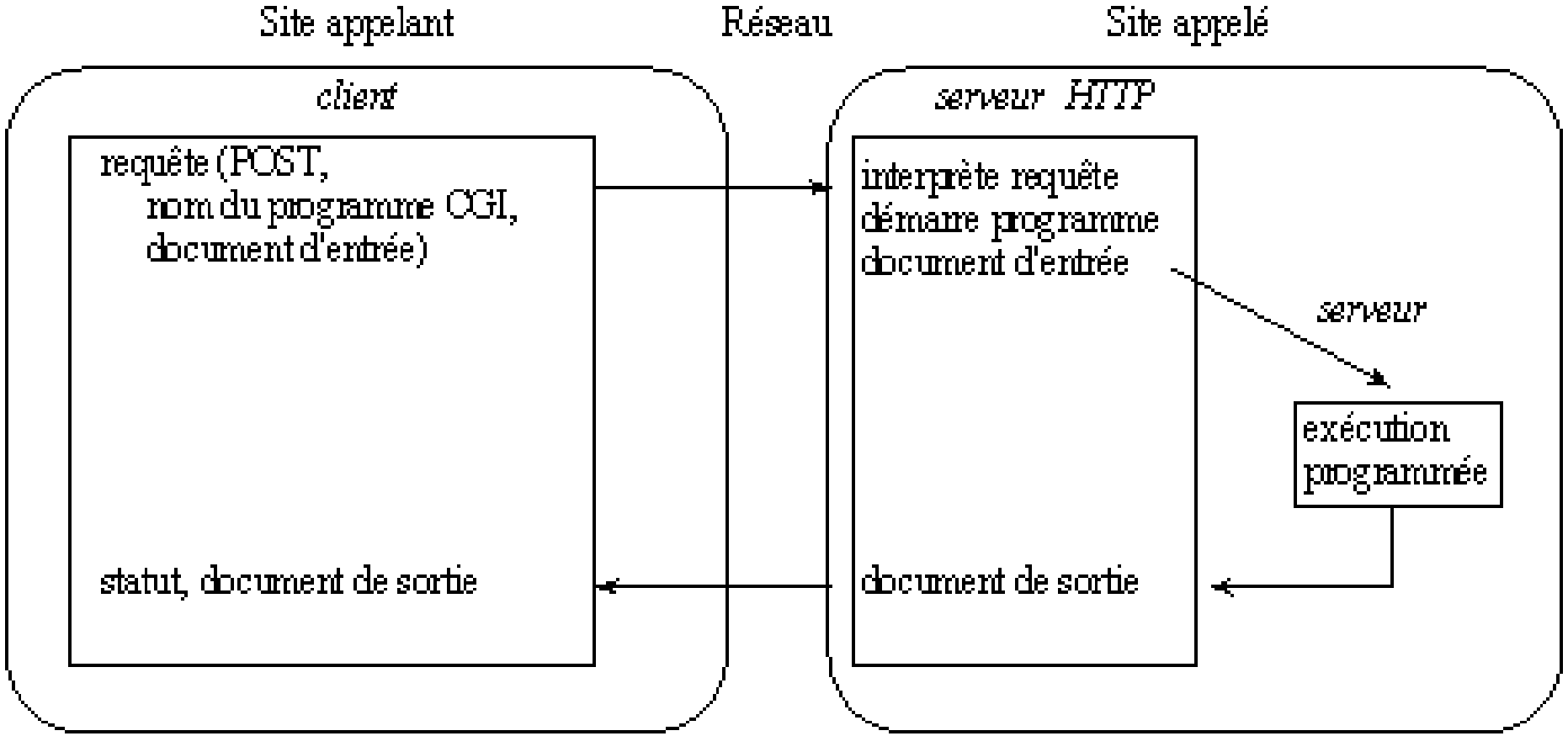
Appel de procédure à distance

- mécanisme de structuration des programmes permettant le transfert du contrôle et des données entre les différentes fonctions d'une application
- très utilisé pour la construction d'applications réparties
- ne permet pas de passer des paramètres par référence
- exemple : l'architecture de SR "DCE" de l'OSF utilise un mécanisme de RPC Sun-like

Appel de procédure à distance

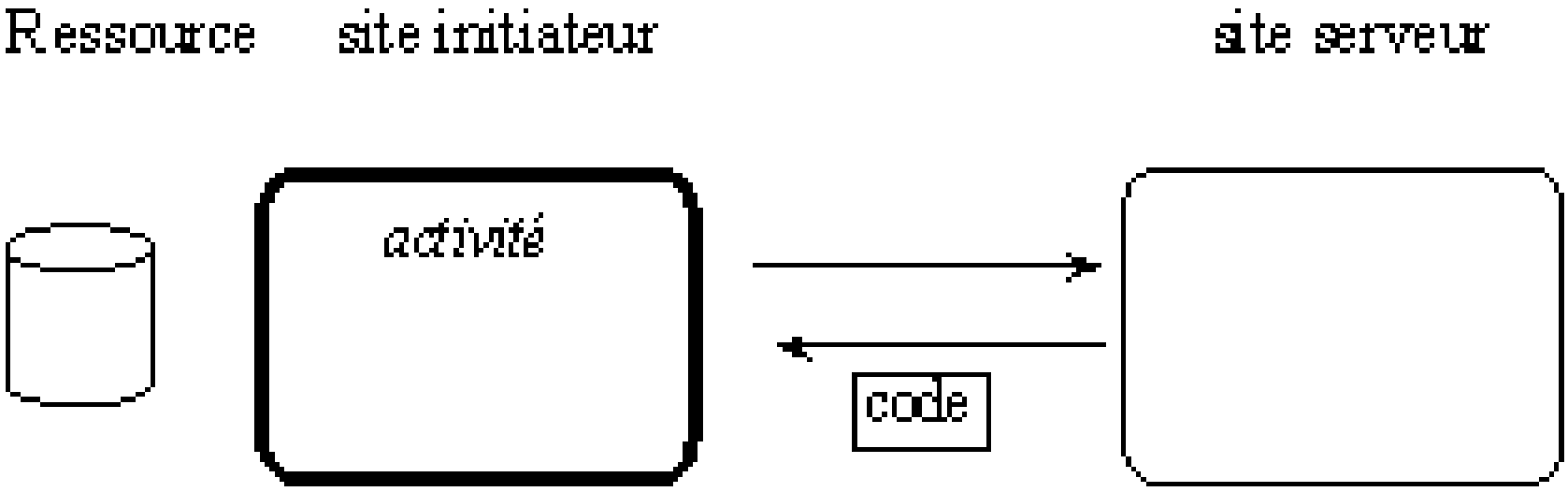


HTTP et programmes CGI

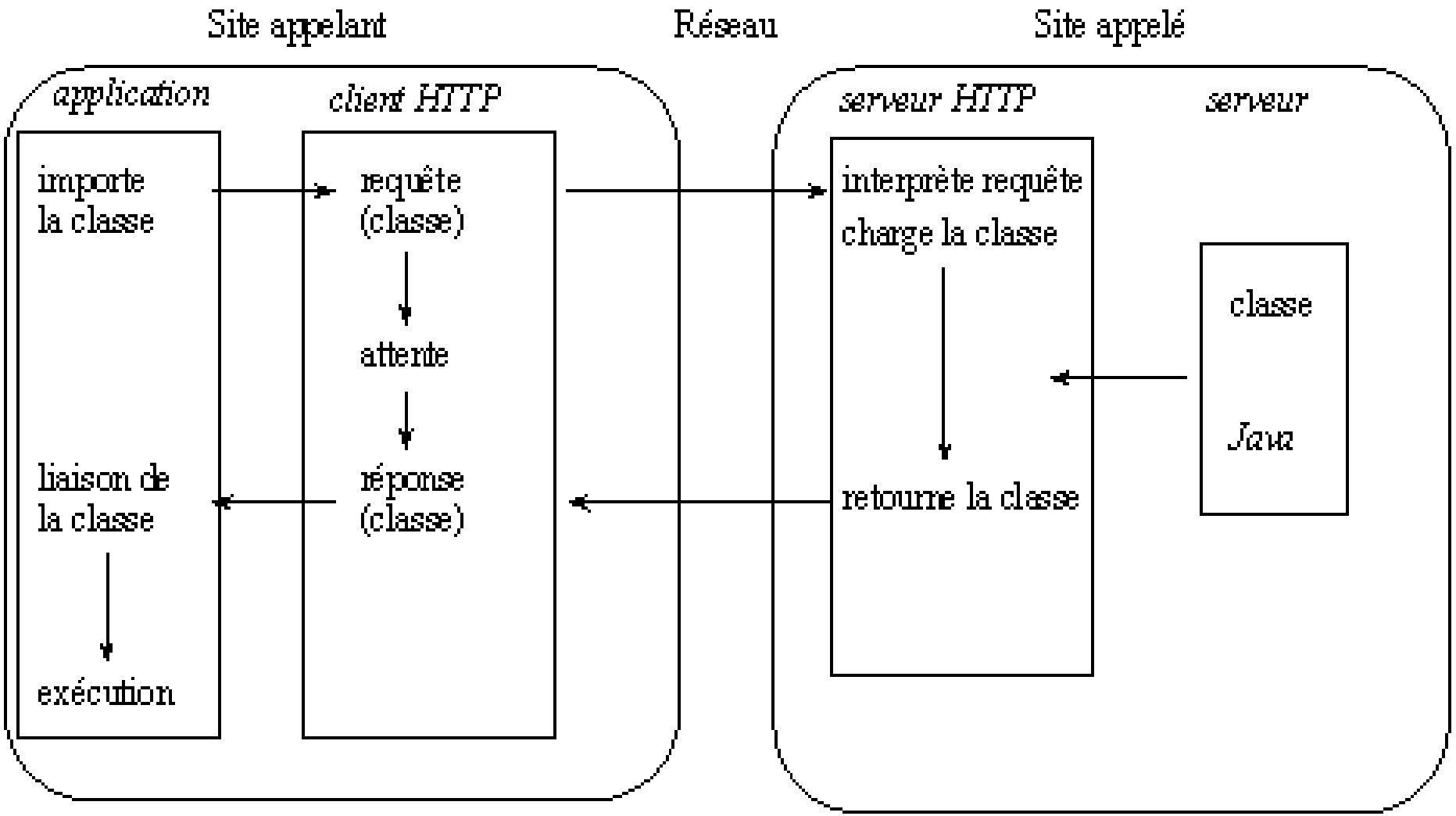


Modèles d'exécution répartie : Code à la demande

- applets
- asynchrone
- gros grain, code et interface définis par le serveur
- approprié aux WANs

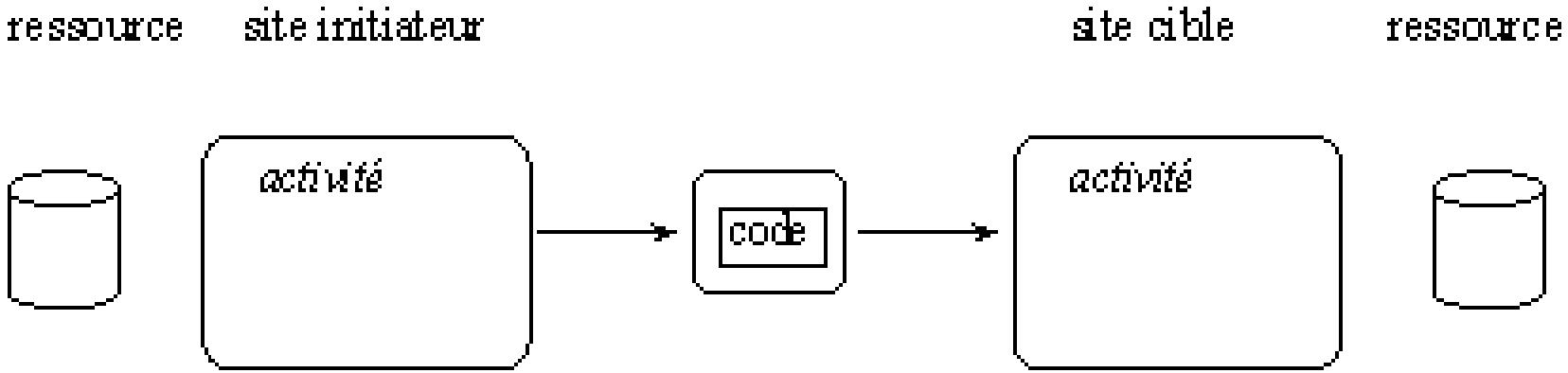


Applets JAVA



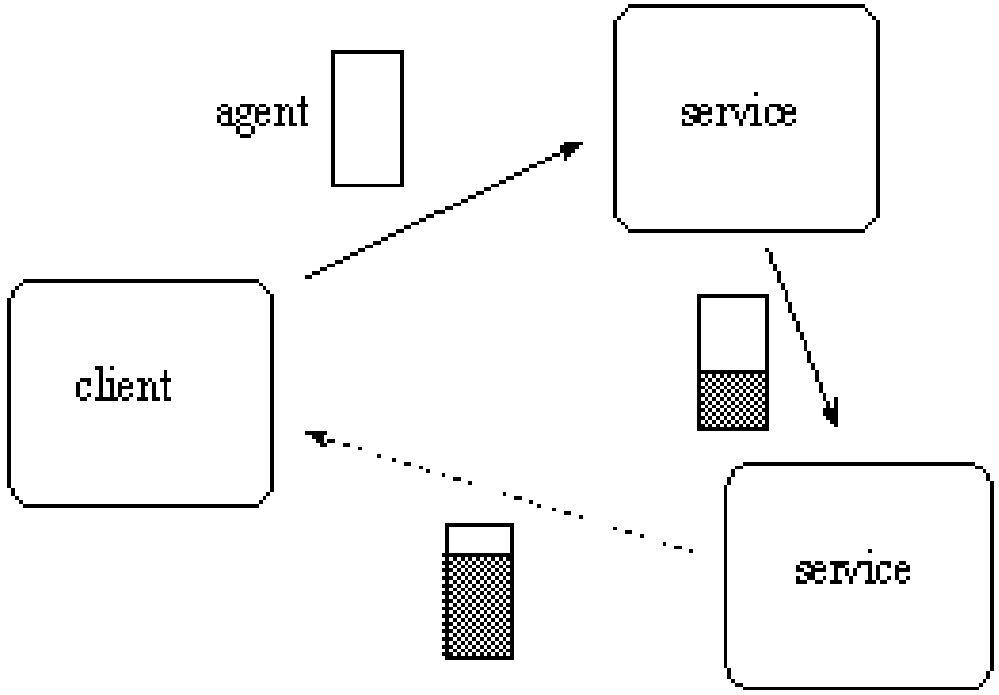
Paradigme

- l'activité démarre sur un site très utilisé pour la construction d'applications réparties
- migration du code et des données sur un site distant
- reprise de l'exécution



Agents mobiles

- connexion seulement pendant la phase de migration
- activation
- exécution de la mission
- récupération des résultats



Caractéristiques des agents mobiles

- opérations à gros grain, code défini par l'utilisateur
- approprié aux WANS
  - migration à l'endroit où se trouvent les données
  - s'accommode d'une grande latence
  - réduction du trafic des données
- Asynchronisme
  - le site d'activation peut se déconnecter
- localisation explicite
  - migration proactive : l'agent décide où aller, et quand – autonomie
  - migration réactive : réponse à une requête externe (du propriétaire notamment)

Agents mobiles et migration de processus

Migration d ’activité dans les systèmes répartis :

- mouvement d'une entité en cours d'exécution sur une machine distante
- utilisé pour la répartition de charge, la tolérance aux pannes, le partage d'information
- migration de processus (Locus, Condor) ou d'objets (COOL, Emerald)
- mécanisme puissant mais complexe à mettre en œuvre (pb de l'hétérogénéité hard / soft, capture de l'état difficile, ...)

Migration de processus

- but de la migration : partage de charge, survivance aux arrêts de machines
- initiative : système d’exploitation
- prise en compte de la répartition : implicite
- portée : LAN

Agents mobiles

- but de la migration : naviguer en fonction de l'application, supporter des opérations déconnectées
- initiative : application
- prise en compte de la répartition : explicite
- portée : WAN

Exemple de domaine d'application : l'admin. de réseaux

- SNMP (IETF) et CMIP (ISO) basés sur interactions Client (station d'admin.) - Serveurs (devices)
  - polling périodique
  - bas niveau (get et set de valeurs dans les MIBs)
  - tous calculs (stats, ...) sur station ==> forte charge de la station et du réseau
- améliorations...
  - SNMPv2 (RFC 1902, 1996) : décentralisation hiérarchique (1 "proxy agent" par pool de devices)
  - RMON (RFC 1757, 1995) : "probes" monitorant l'info globale traversant les lignes plutôt que les infos locales à un device
  - mais pas assez décentralisé pour les réseaux de grande taille

Exemple de domaine d'application : l'admin. de réseaux

- apport des agents mobiles
  - composants de monitoring localisés dans les devices, rapportant les événements représentant une évolution de l'état du device ==> performances
    - ex. : pool de devices sur LAN administrés à distance via liaison lente
  - possibilité de télécharger du code sur des sites distants
    - customisation
    - autonomie
    - tolérance aux pannes

Infrastructure système pour les agents mobiles

- introduction
- exécution
- migration
- communication
- désignation et localisation
- contrôle de l’utilisation des ressources
- sécurité

Infrastructure pour les agents mobiles : introduction

- des environnements locaux sont nécessaires pour accueillir des agents et leur permettre de réaliser leurs tâches : « agences »

- fonctionnalités de base des agences : fournir un support pour :
  - l'exécution (allocation de ressources) (par le runtime)
  - la communication agents-serveurs (locaux ou distants)
  - la migration (en environnement hétérogène - transfert de l'état)
  - la désignation et la localisation des agents
  - le contrôle de l’utilisation des ressources
  - prise en compte de la sécurité

Infrastructure pour les agents mobiles : exécution

- admission
  - (analogie : portier de l'hôtel)
  - réception de l'agent (code, état et attributs) et déballage des attributs liés à la sécurité
- allocation de ressources
  - (analogie : comptoir d'enregistrement)
  - vérification du code
    - compilation dynamique
    - + : vérif. commode, optimisations locales
    - - : durée, non confidentialité du code source, dépendances machines
    - Interprétation
      - directe (Tcl, Scheme, ...)
      - indirecte (code intermédiaire portable : Java)
  - 1 agent = 1 processus ou 1 thread dans une JVM
  - autres attributs :
    - besoins mémoire, limitations (date d'expiration), ...
- actions en cas d'erreurs : rejet, notification à une certaine adresse, routage vers une autre agence

Infrastructure pour les agents mobiles : migration (1)

- sérialisation du code, des données et de l’état d ’exécution - transfert – désérialisation
- migration "traditionnelle" (objets, processus)
  - difficulté : état (dépend en partie de la plate-forme : adresses sur la pile, registres)
  - le système décide quoi, quand, où ==> état a priori quelconque
- migration d'agents
  - l'agent décide quand et où
  - quand :
    - en général, pour faire autre chose ==> dans le meilleur des cas, l'état ne contient que les résultats accumulés (data) et le pointeur de programme
  - où :
    - liste statique des hôtes à visiter
    - choix dynamique (en fonction des résultats et/ou de la disponibilité des services)
  - actions à entreprendre par l'agence locale
    - le comptoir d'information supprime l'agent de la liste locale
    - le portier emballe l'agent (code, état, attributs) et l'envoie sur le réseau
    - le comptoir d'enregistrement peut réclamer les ressources utilisées

Infrastructure pour les agents mobiles : migration (2)

- mobilité forte
  - support langage / système spécifique
    - interpréteur modifié (ARA, D'Agents, MAP)
    - JVM modifiée (SIRAC, Sumatra)
    - ...

    instruction i ;

    go (« autre place ») ;

    instruction i + 1

    ...

Infrastructure pour les agents mobiles : migration (3)

- mobilité faible
  - la JVM ne permet pas de sauvegarder le contexte d'exécution du thread
  - simulation de la migration du thread :
    - migration du code et de données d'initialisation
    - reprise de l'exécution à une méthode spécifiée
    - callbacks pour manipuler les données nécessaires à la reprise
    - exemple : Aglets

```
run ()
...
dispatch()
    onDispatching() -> onArrival()
                                run ()
...
```

Infrastructure pour les agents mobiles : migration (4)

Ressources

- migration EU ==> modification des bindings aux ressources ou migration des ressources
- types de ressources
  - transférables : libres (ex : fichier) ou fixées (ex : base de données)
  - non transférables (ex : imprimantes)
- ressource = triplet [Identificateur, Valeur, Type]
- forme de binding EU-ressource ==> mécanismes de gestion de l'espace des données exploités par la migration
  - par I : pas de ressource équivalente (écran du propriétaire)
  - par V : EU intéressée par le contenu de la ressource
  - par T (librairies, driver réseau, écran local)
- ==> 2 problèmes :
  - relocalisation des ressources
  - reconfiguration des bindings
- mécanismes envisageables
  - perte du binding, déplacement, copie, référence réseau, re-binding

Infrastructure pour les agents mobiles : communication

- avec des services (locaux ou non), d'autres agents (locaux ou non), le propriétaire
- mécanismes standard (RPC, invocation d'objet) au minimum
- service de localisation
  - (analogie : comptoir d'information)
  - infos sur agents et services locaux, comm. avec agences distantes pour infos sur agents et services distants
  - naming services (DCE, CORBA) peuvent être utilisés
- communication avec le propriétaire (identifié dans les attributs)
  - généralement asynchrone (mailbox)

Infrastructure pour les agents mobiles : communication

- communication inter-agents
  - locaux vs. Distants
  - synchrone vs. Asynchrone
  - multicast local ou distant
- communication avec un service ou agent distants
  - comm. distante ou migration sur le site distant ?
  - complexe : dépend des possibilités offertes par le site distant, compromis entre coûts de communication et de migration
  - décision "câblée", aidée par l'agence, ou prise par le propriétaire

Infrastructure : désignation et localisation

- désignation et localisation des unités d'exécution : URL
- désignation des agents :
  - UID immuable
  - noms symboliques (agents ou points de communication) : pas de contrôle global
- localisation des agents :
  - nécessaire au contrôle (migration réactive, terminaison forcée) et à la communication
  - pointeurs de poursuite ou notification à l'environnement de création

Infrastructure : utilisation des ressources

- pour un agent
  - durée de vie, nombre maximum de migrations, nombre maximum de clones, …
- pour une EU
  - nombre maximum d'agents présents simultanément, accès au système de fichiers, accès à l'écran, utilisation CPU ou mémoire, …
  - listes de contrôle d'accès pour
    - les sites de provenance (trusted et untrusted)
    - les propriétaires (idem)
- limitations de la JVM : ne permet pas un contrôle fin de l'accès aux ressources (mémoire, disque)

Infrastructure pour les agents mobiles : sécurité

- protection de l'hôte : assez bien résolu
  - JAVA : vérification du bytecode, chargement des classes, gestionnaire de sécurité
  - restrictions aux langages interprétés (Safe-Tcl)
- protection de l'agent : plus complexe
  - doit avoir l'assurance que le code / l'état n'est pas modifié par l'agence (commerce électronique)
  - tout ce qui est accessible à l'agent l'est aussi à l'hôte!!
  - ==> des moyens d'authentification mutuelle et d'autorisation sont essentiels
  - peuvent impliquer des tierces parties
  - 2 pistes :
    - encryptage du code
    - détection dès que possible qu'un agent a été « manipulé » (signatures digitales)
  - solutions techniques partielles - une partie de la solution sera toujours sociologique ou légale
- cryptage éventuel pour la protection des conversations locales ou distantes

Exemples

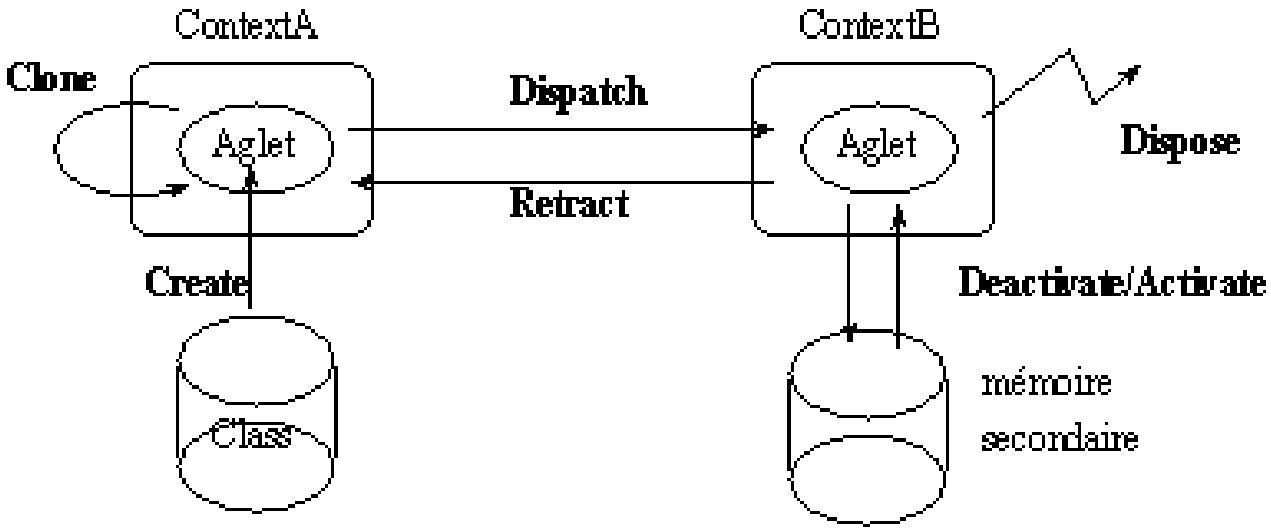
- Aglets
- Ara
- Agent Tcl
- Voyager

Aglets

- Aglet WorkBench IBM Japon, 1996
- environnement de programmation d'agents mobiles en Java
- aglet = objet Java comportant un flot d'exécution et méthodes pour la mobilité
- contexte = espace d'adressage sur une machine pouvant abriter un ensemble d'aglets
- proxy = capacité sur un aglet : protège de l'accès direct, gère la localisation
- contexte <==> machine et numéro de port
- lancement du système et interaction utilisateur graphique (Tahiti) agletsd -port 9000

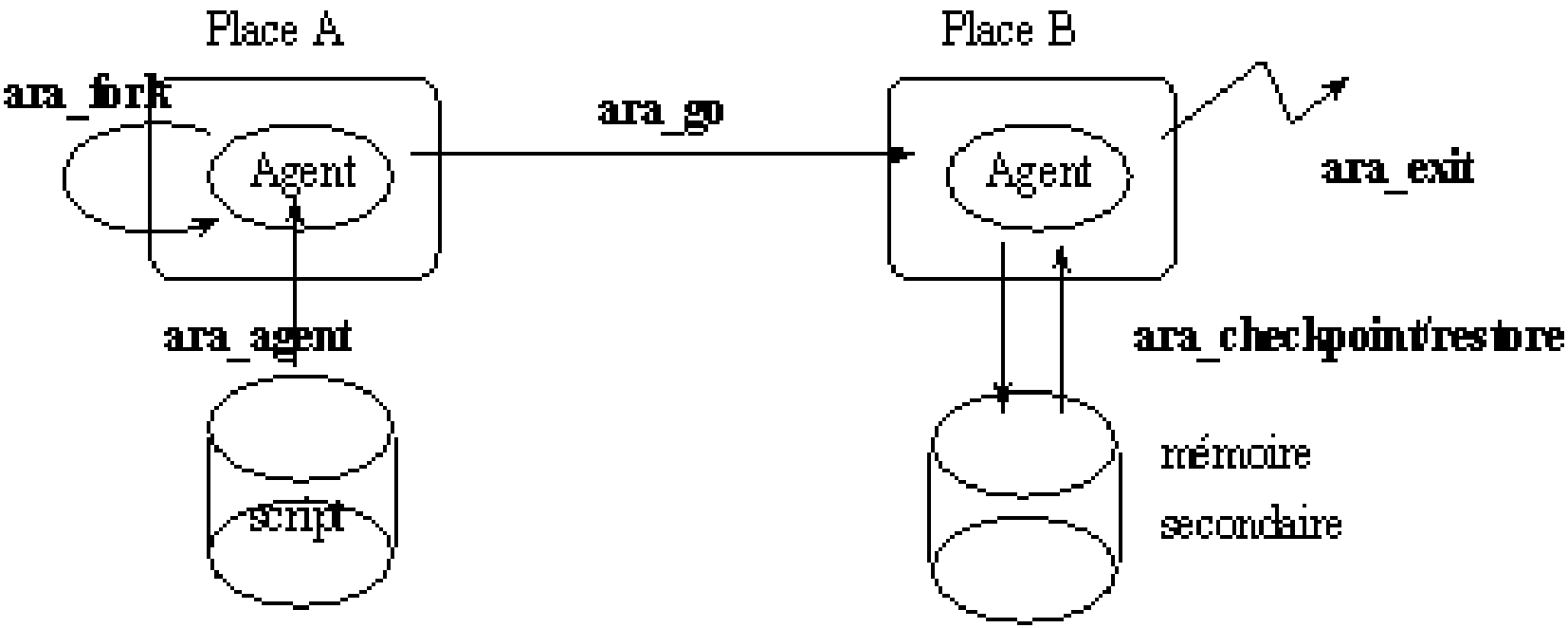
Aglets

- messages : communication synchrone et asynchrone
- migration proactive : méthode go(URL) - notion d'Itinéraire ; migration réactive (Retract)
  - weak mobility : sérialisation du code et des données, mais pas de l'état du thread : l'exécution reprend au début (méthode run()) - possibilité de différencier les comportements par des variables globales
  - EU = 1 thread dans un interpréteur Java ==> partage de ressources possible - migration par Copie
- sécurité : contrôle de l'accès en lecture et écriture au système de fichiers local
- cycle de vie



Ara

- Ara = "Agents for Remote Action", Univ. Kaiserslautern, 1997
- multi-langages (Tcl, C/C++, Java prévu)
- précompilation de C en bytecode interprété
- cycle de vie (primitives Tcl) :



Ara

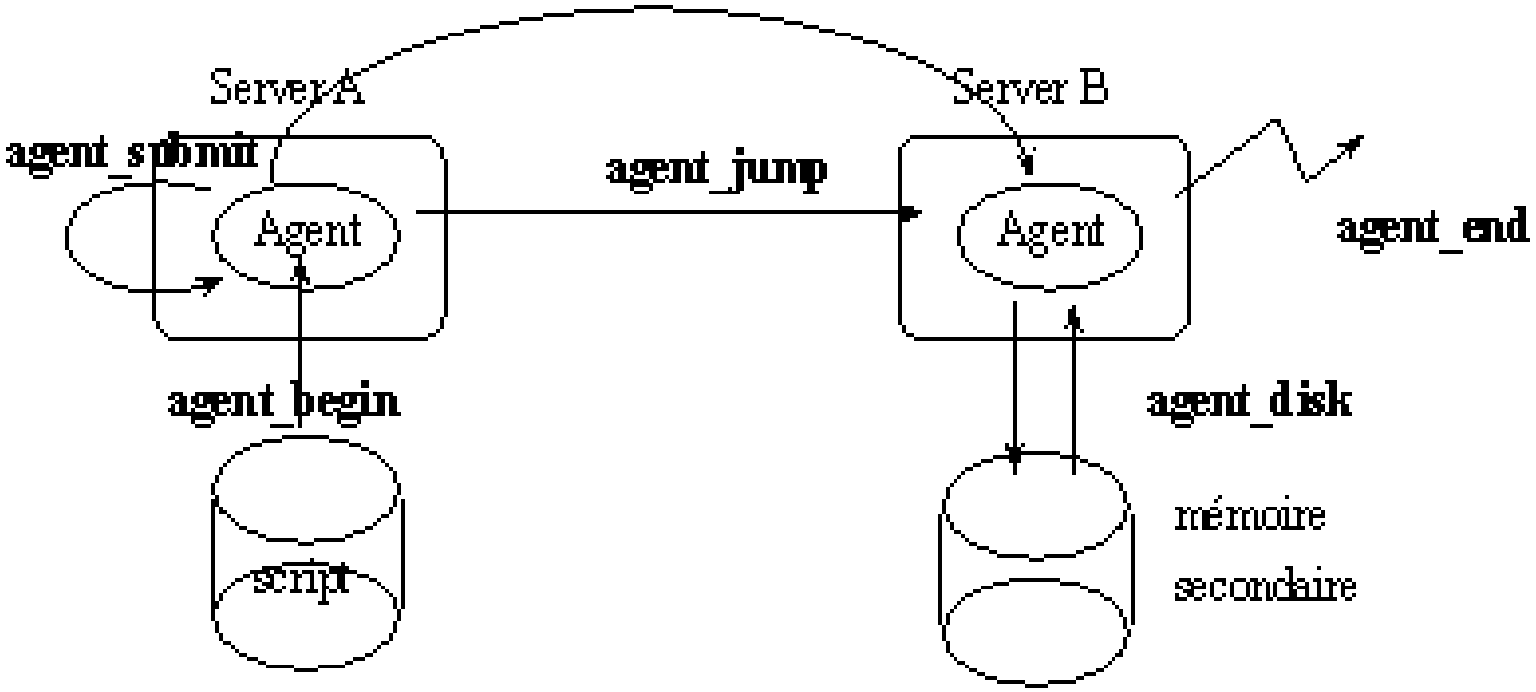
- messages : communication synchrone (uniquement entre agents locaux)
- migration proactive : méthode ara_go(machine:port)
  - strong mobility : sérialisation du code et des données, et de l'état du thread : l'exécution reprend immédiatement après la primitive ara_go()
- véritable checkpointing de l'état interne (implicite pour la migration ou explicite)
  - utilisable pour la tolérance aux pannes
  - 1 EU (processus UNIX) par agent ==> peu de partage de ressources ==> perte des bindings lors de la migration
  - pas de sauvegarde(ni de transfert) de l'état externe (fichiers ouverts, fenêtres, ...)
- sécurité : rien
- lancement du système et interaction utilisateur : interpréteur "arash"
- contrôle de l'utilisation des ressources : max CPU et MEM
- possibilité de compilation C (pour agents fixes principalement)

Ara

- messages : communication synchrone (uniquement entre agents locaux)
- migration proactive : méthode ara_go(machine:port)
  - strong mobility : sérialisation du code et des données, et de l'état du thread : l'exécution reprend immédiatement après la primitive ara_go()
- véritable checkpointing de l'état interne (implicite pour la migration ou explicite)
  - utilisable pour la tolérance aux pannes
  - 1 EU (processus UNIX) par agent ==> peu de partage de ressources ==> perte des bindings lors de la migration
  - pas de sauvegarde(ni de transfert) de l'état externe (fichiers ouverts, fenêtres, ...)
- sécurité : rien
- lancement du système et interaction utilisateur : interpréteur "arash"
- contrôle de l'utilisation des ressources : max CPU et MEM
- possibilité de compilation C (pour agents fixes principalement)

Agent Tcl

- Agent Tcl ("D'Agents"), Univ. Dartmouth, 1997
- basé sur Tcl 7.5
  - modifié pour fournir la capture de la pile et de l'état
  - étendu de commandes pour migration, communication, clonage
- cycle de vie :



Agent Tcl

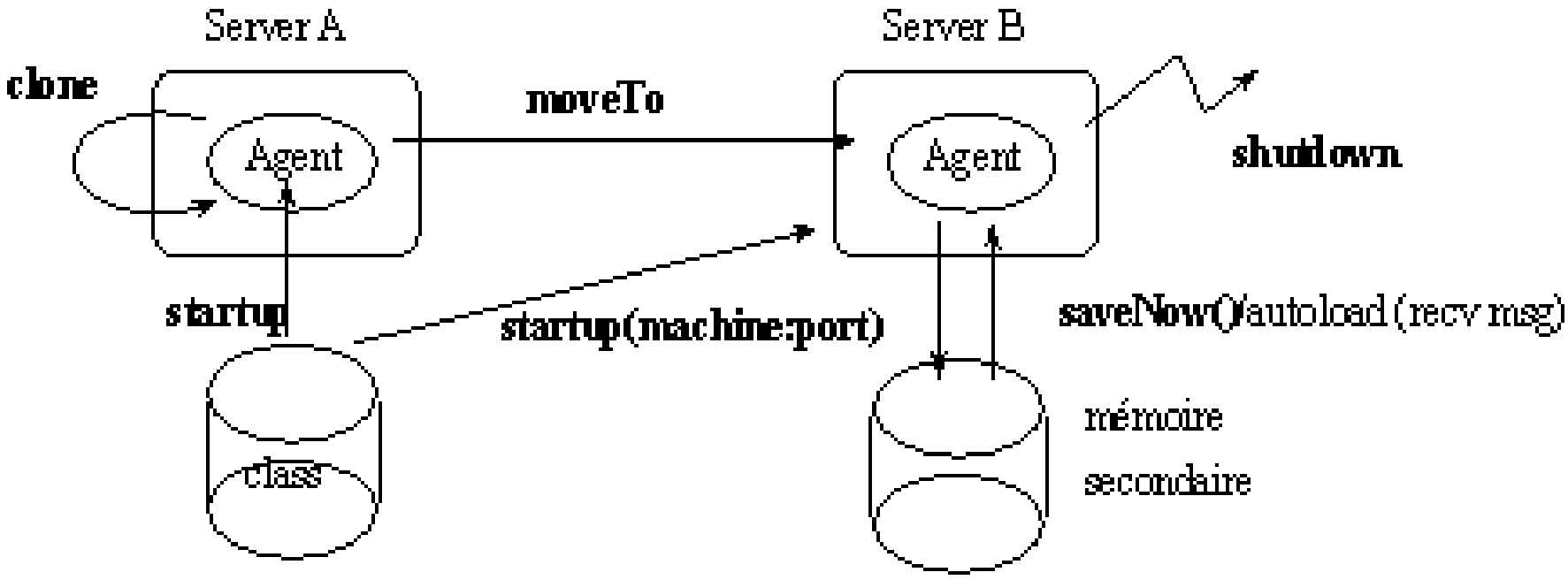
- messages : communication synchrone et asynchrone, locale ou distante
- migration proactive : méthode agent_jump(machine)
  - : sérialisation du code et des données, et de l'état du thread : l'exécution reprend immédiatement après la primitive agent_jump()

- véritable checkpointing de l'état interne
  - utilisable pour la tolérance aux pannes
  - 1 EU (processus UNIX) par agent ==> peu de partage de ressources ==> perte des bindings lors de la migration
  - pas de sauvegarde(ni de transfert) de l'état externe (fichiers ouverts, fenêtres, ...)

- sécurité : PGP
- contrôle de l'utilisation des ressources : max CPU, devices, file system, programmes externes
- lancement du système et interaction utilisateur : ligne de commande (interpréteur Tcl "agent") - interface graphique (interpréteur "agent-tk")

Voyager (1)

- Voyager, Objectspace, Version 1.0, 1997
- extensions de Java
- cycle de vie :



Voyager (2)

- messages : communication synchrone et asynchrone, entre agents locaux ou distants
- communication de groupe (objets groupés en "spaces" reliant des "subspaces" locaux)
- migration proactive : méthode moveTo(machine:procedure) - migration réactive
  - weak mobility : sérialisation du code et des données, mais pas de l'état du thread : possibilité de différencier les comportements par des variables globales - possibilité de spécifier le point d'entrée pour la reprise de l'exécution
  - 1 EU = 1 thread dans un interpréteur Java ==> partage de ressources possible - migration par Référence Réseau

Voyager (3)

- gestion de la persistance : activation sur réception de message
- sécurité, authentification : rien ?
- contrôle de l'utilisation des ressources ??
- directory service pour objets locaux ou distants
- lancement du système et interaction utilisateur :
  - serveur : voyager <N° de port> ; autres programmes : java <programme>
- ORB avec support pour les agents mobiles (dans la version 2.0 beta 1 et les futures)
  - Voyager peut être utilisé comme client ou serveur CORBA
  - les agents Voyager peuvent être vus comme des objets CORBA et inversement

Fin