

Communication par messages dans les systèmes multi-agents

Jacques Ferber

**LIRMM - Université Montpellier II
161 rue Ada
34292 Montpellier Cedex 5**

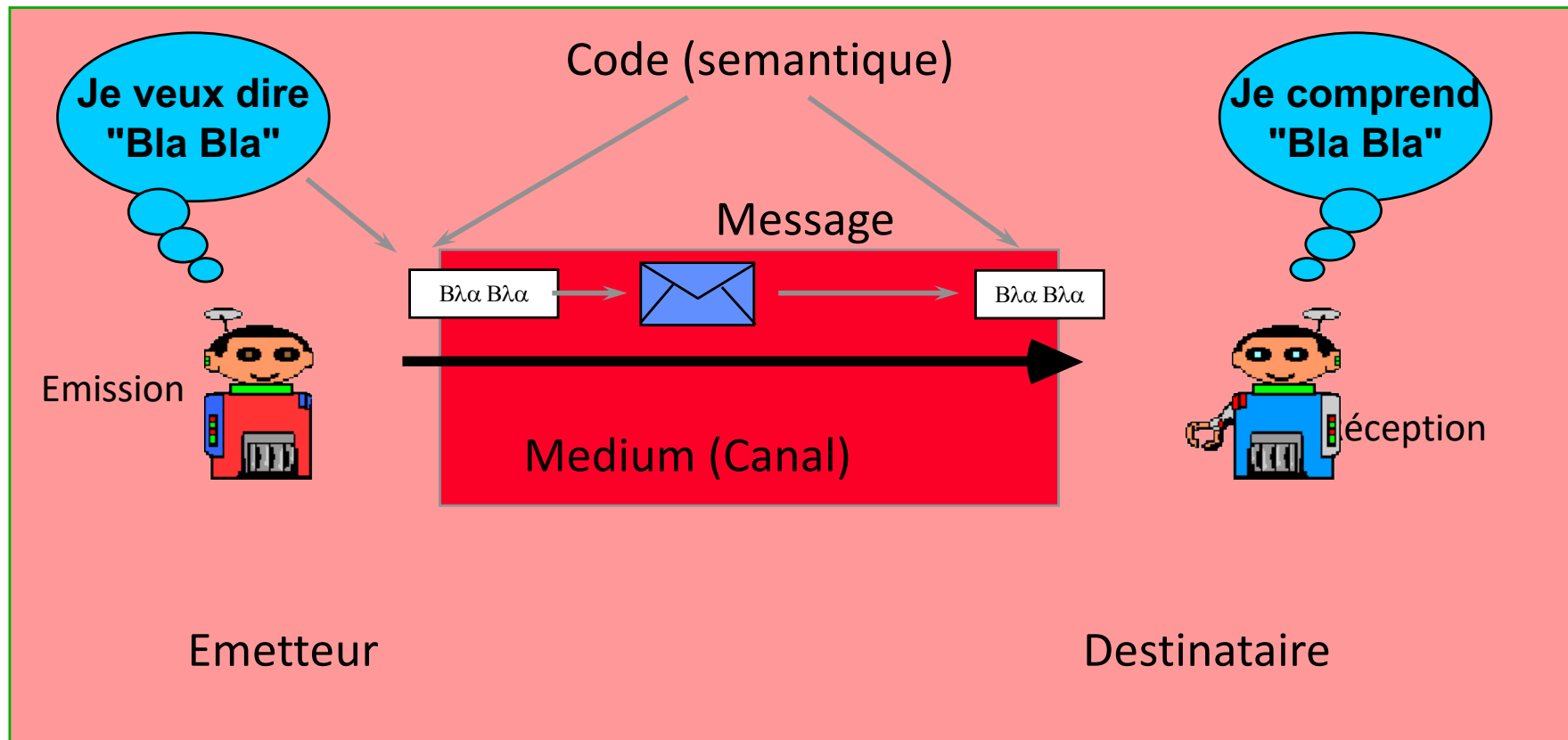
**Email: ferber@lirmm.fr
Home page: www.lirmm.fr/~ferber**

*Version 3.7
2014-2019*

Communication

- ◆ **Eléments de communication**
- ◆ **La théorie des actes de langage**
- ◆ **Conversation et protocoles**

Modèle classique de la communication



Les différentes manières de voir la communication

◆ **Point à point**

- Ex: email, sms

◆ **Groupes**

- Emails de groupes, messages envoyés à tout un groupe

◆ **A tous les agents à une certaine distance**

- Distance physique
- Distance sociale (ex: amis et amis d'amis)

◆ **Sur support partagé**

- Forums, murs d'un groupe, etc...

◆ **Message à un agent connu directement:**

- send-message [destinataire performatif contenu]
- Le contenu peut être n'importe quoi, et notamment une liste...
 - ☞ Attention: le receveur doit savoir quel type de donnée vous avez envoyé

◆ **Broadcast à un ensemble d'agents**

- broadcast-message [agentSet(destinataires) performatif contenu]
- Les destinataires naturels:
 - ☞ get-bases : renvoie l'agentset des bases amies.
 - ☞ get-rocket-launchers : renvoie l'agentset des rocket-launchers amis.
 - ☞ get-explorers : renvoie la liste des explorateurs amis.

Exemple de code d'envoi

- ◆ 1) Envoie à une base amie
- ◆ 2) Envoie à tous les rocket launchers.

```
let enemy-base one-of percepts with [is-base? self and is-my-enemy? myself]
let my-base one-of get-bases
if enemy-base != nobody and my-base != nobody [
  send-message my-base "seen-base" []
  broadcast-message get-rocket-launchers "attack" []
]
```

Réception des messages

- ◆ Les messages sont placés dans une boîte aux lettres: **incoming-queue**
- ◆ On peut les lire par les primitives **get-message** et **read-message**
 - **read-message** retourne le premier message de **incoming-queue** mais ne le supprime pas
 - **remove-msg** supprime le premier message de **incoming-queue**
 - **get-message** : lit et supprime le premier message de **incoming-queue**

Ex. de réception de messages

- ◆ On retourne "ok" à tous les émetteurs de messages

```
while [not empty? incoming-queue] [  
    let msg get-message  
  
    reply msg "ok" [] ;; permet de retourner un message à l'émetteur  
]
```

Les attributs des messages

- ◆ **Les messages contiennent tout un ensemble d'informations dont la distance et la direction vers l'émetteur**

let msg get-message

get-sender msg ;; l'émetteur du message

get-performative msg ;; le performatif, le "type" de message

get-content msg ;; le contenu du message

get-heading-to-sender msg ;; la direction vers l'émetteur

get-distance-to-sender msg ;; la distance à l'émetteur

- ◆ **Exemple:**

Let msg get-message

Let dir get-heading msg

set-heading dir

Signification des messages

◆ Signification fixe => communications intentionnelles

- La sémantique de la communication est partagée par l'émetteur et le receveur
 - ☞ Suppose un langage de communication commun
 - ☞ Pbs de la définition de standards..
- L'émetteur a **l'intention d'émettre** un message avec cette signification!

◆ Signification relative => communication liée à la situation

- Généralement lié à des traces
- La sémantique de la communication est donnée par le receveur
 - ☞ Notion de signe, d'indice
- L'émetteur communique parce qu'il ne peut pas le faire autrement

◆ Il peut y avoir un mix des deux:

- Vous voulez communiquer à X mais vous êtes repéré par Y

Les actes de langage (Speech acts)

◆ **Concepts développés initialement dans le contexte de la philosophie du langage (Austin, Searle, Vanderveken, ..)**

◆ **Communiquer c'est agir**

- Les phrases ne sont pas seulement vraies ou fausses. Elles servent à accomplir des actions.
- Notion de la "pragmatique" du langage (sens opérationnel d'une communication)
 - ☞ ex: demander de faire quelque chose (une manière d'accomplir un but)

Actes de langages (cont.)

◆ Catégorisation des actes de langages

- Inform, ask, request, warn, promise, ...

◆ Décomposer une phrase en un performatif et son contenu: F(P)

- Ask(*la lumière est allumée*) *la lumière est-elle allumée?*
- Inform(*la lumière est allumée*) *la lumière est allumée!*
- Request(*la lumière est allumée*) *allumez la lumière, SVP*

Performative

content

◆ Intérêt: la sémantique pragmatique (ou opérationnelle) du message est définie par le performatif

Aspects des actes de langage

◆ Un acte de langage comprend trois éléments d'action:

● élément locutoire

☞ l'acte de communiquer quelque chose à quelqu'un

● élément illocutoire

☞ le type d'action qui est accompli en communiquant.

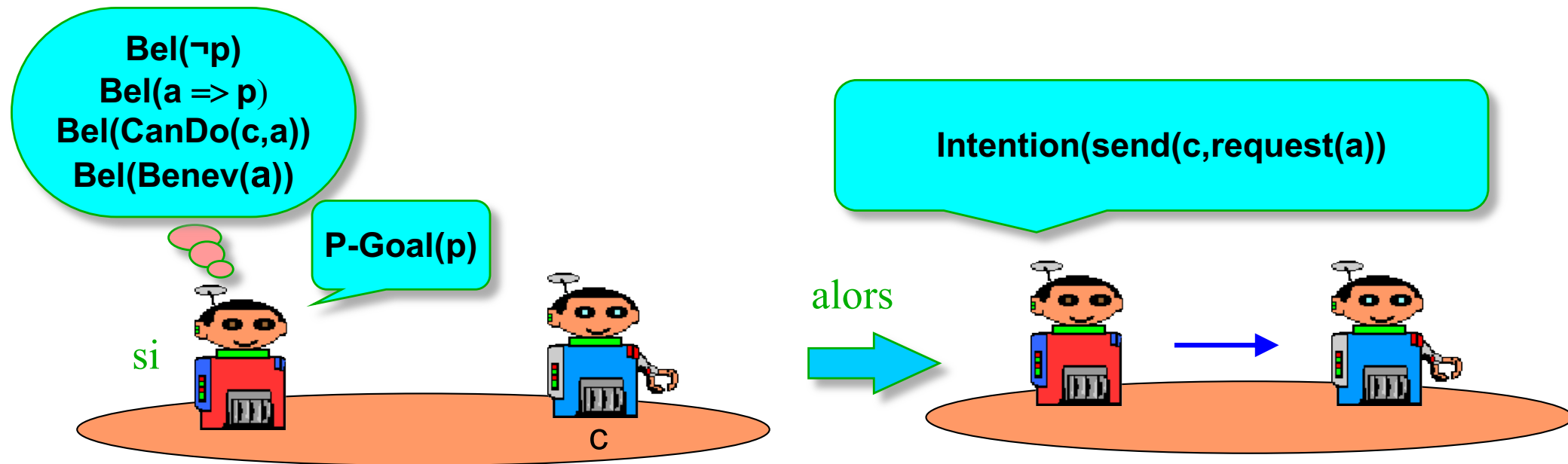
☞ e.g. demander, informer, promettre

● élément perlocutoire

☞ Conséquences des actes illocutoires qui sont en fait désiré par l'acte de langage

☞ ex: Request(e,r,a) → a est accompli

Comportements dirigés par les buts



Définition d'un langage de communication

◆ Langage de communication (ACL - Agent Communication Language)

- Les agents doivent avoir des capacités à manipuler un langage commun

◆ Aspects du langage

- **Syntaxe:** manière dont les symboles sont structurés.
- **Sémantique:** ce que le symbole signifie
- **Pragmatique:** manière dont les symboles sont interprétés pour conduire à l'action

Echec d'un acte de langage

- *Dans la transmission*

- ☞ ex: un message n'arrive pas

- *Dans l'interprétation*

- ☞ le message est mal compris par le receveur (pour des raisons de terminologie par exemple)

- *Dans l'accomplissement d'un acte illocutoire*

- ☞ Echec du succès, si la condition pour l'acte illocutoire n'est pas réalisée.

- ☞ **A: B, Request(P)** est effectif si **B croît que A veut que B accomplisse P**

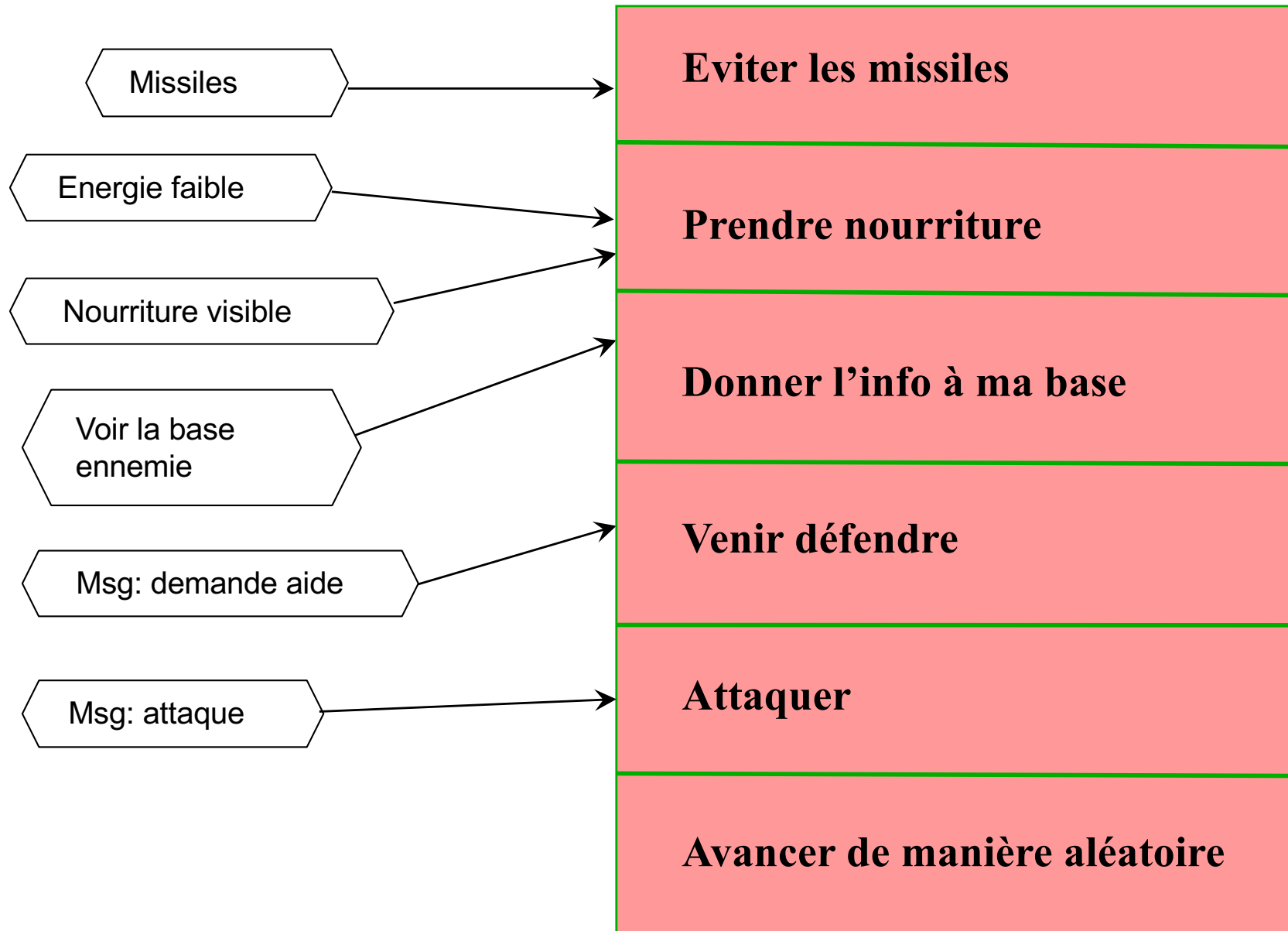
- *Dans l'accomplissement de l'acte perlocutoire*

- ☞ Echec de la satisfaction si **B effectivement accomplit P**

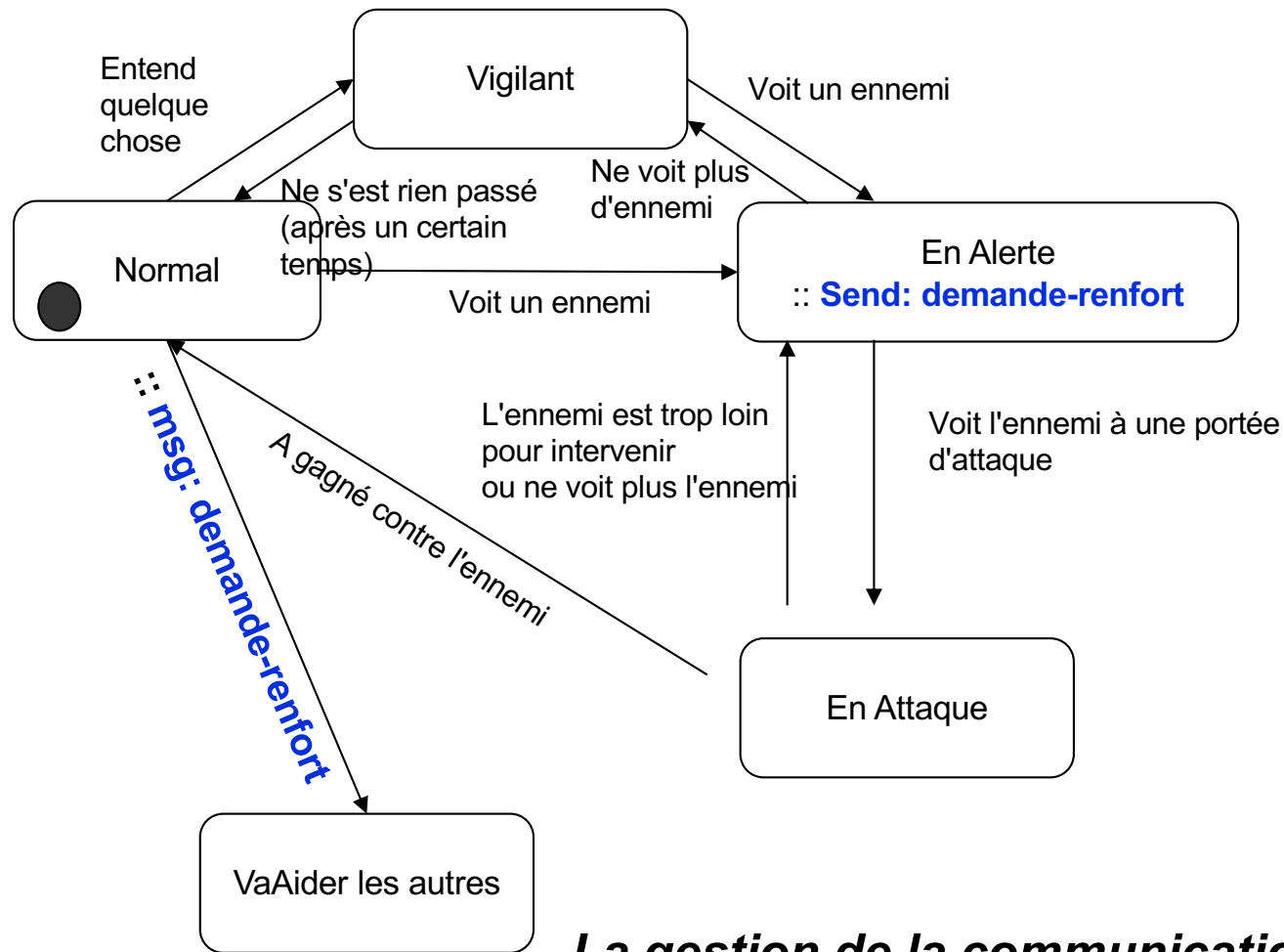
Comment intégrer les communications

- ◆ **Considérer que les messages sont des événements comme des percepts**
 - Architecture à base d'action située (ex: subsomption ou règles d'actions situées)
 - Architecture à base d'automates à états finis

Architecture de subsomption avec messages

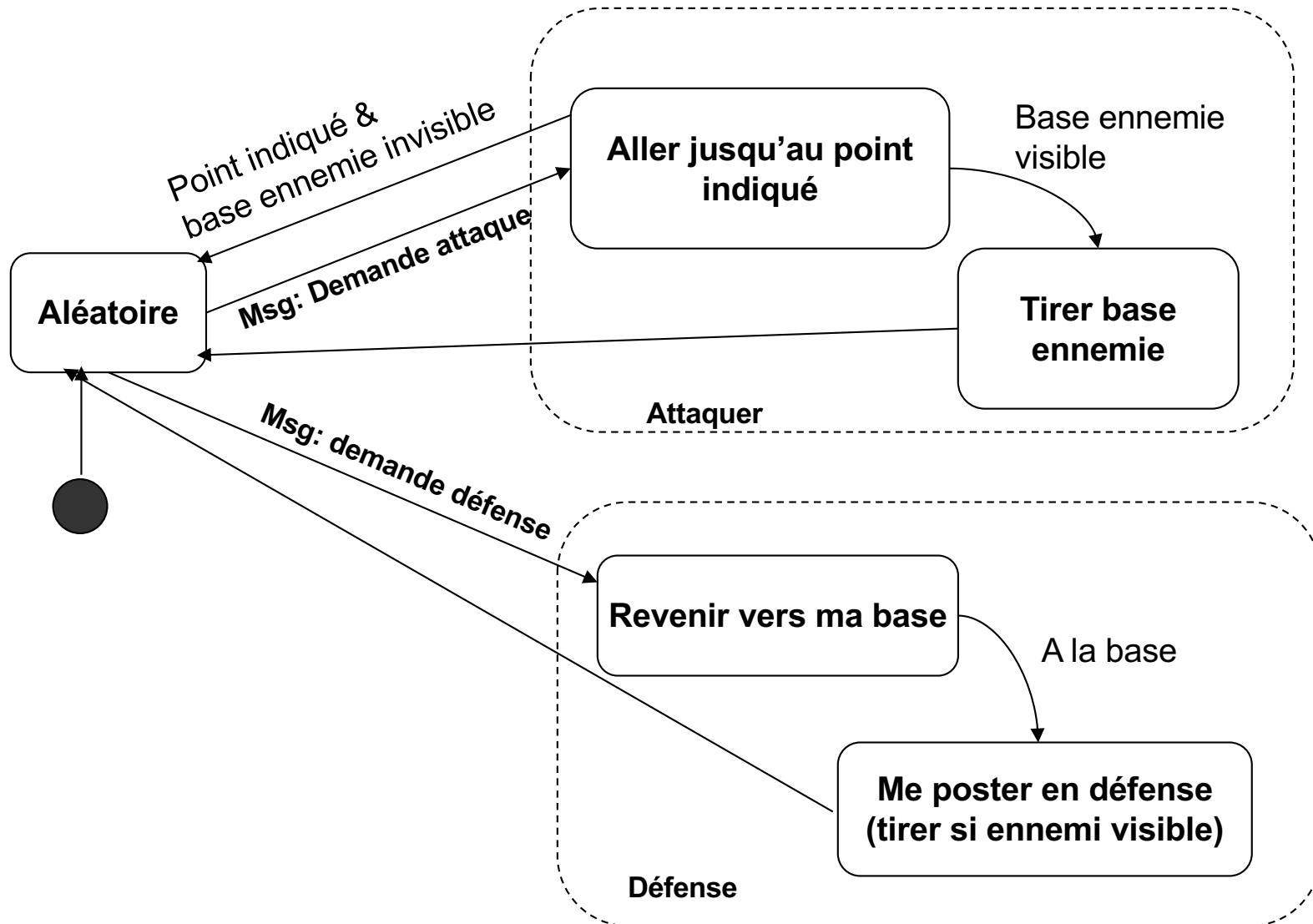


Intégrer les messages dans une FSM

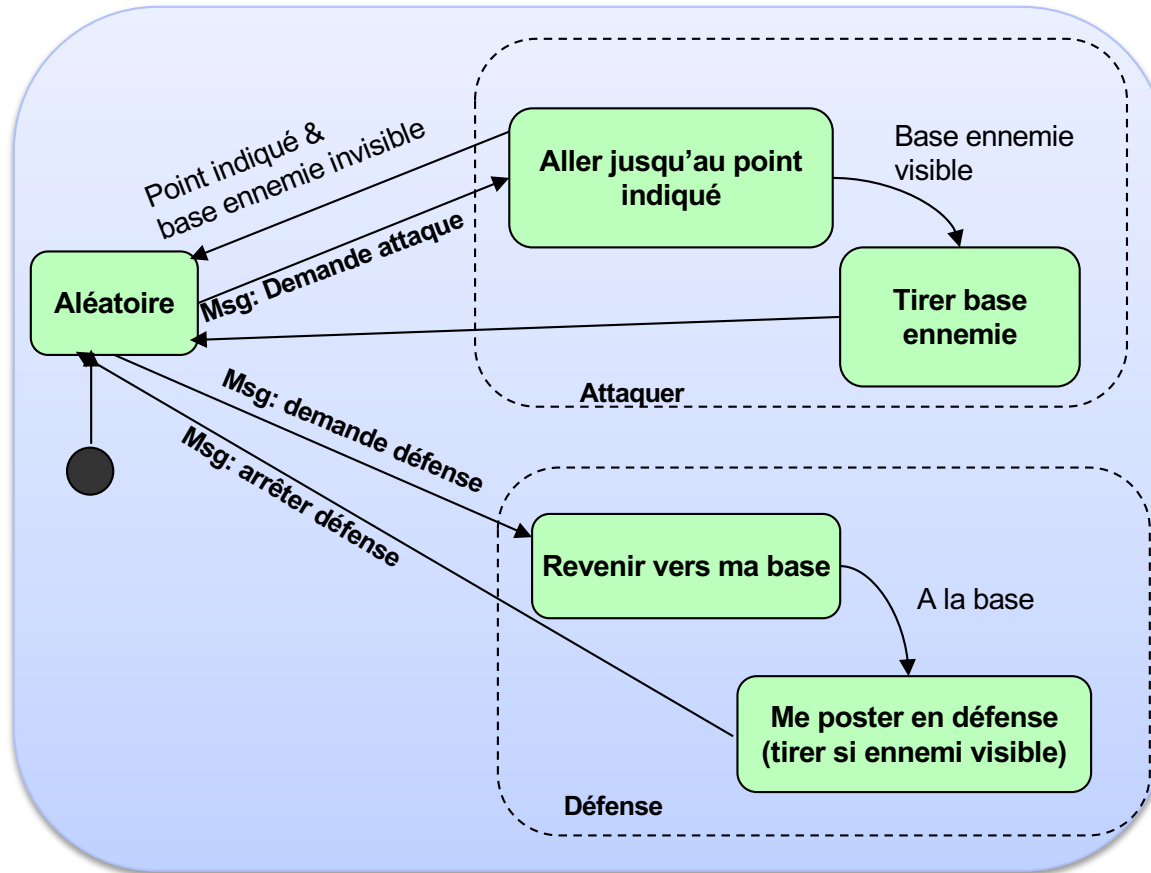


La gestion de la communication se situe dans les états

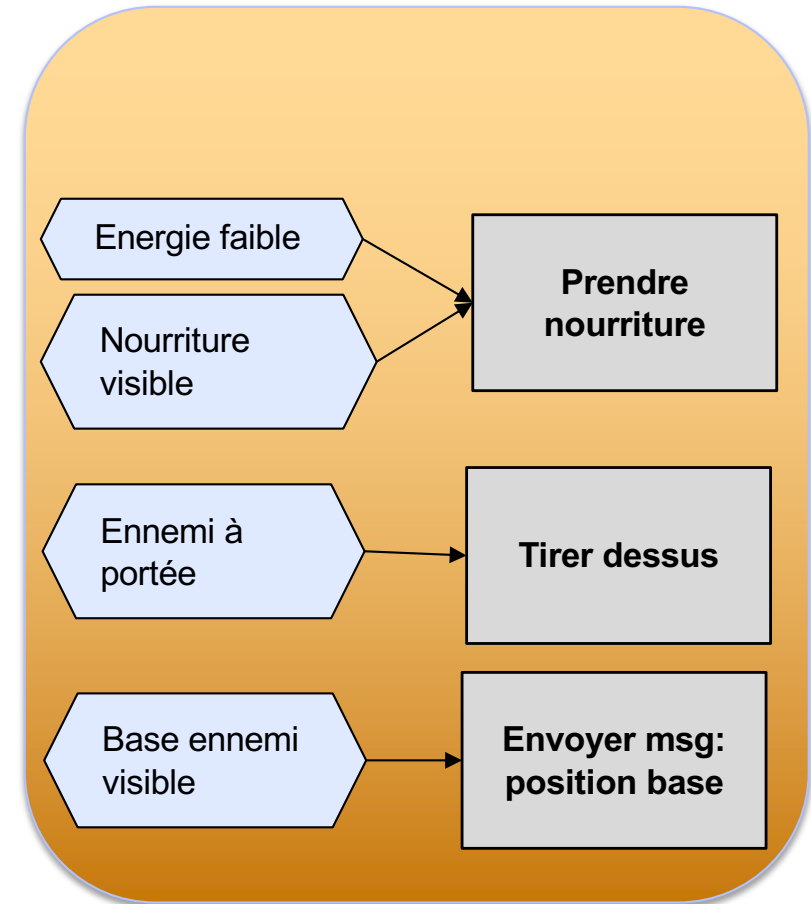
Architecture à base de FSM hiérarchique



FSM + réflexes



FSM



Réflexes

Limite des actes des communications simples

Les communications ne sont pas accomplies comme des ensembles de messages isolés

- ◆ Communication sont structurées en **dialogues** ou **conversations**, qui sont des séquences stylisées de messages (**protocoles**).
- ◆ Ex de protocoles
 - Demander quelque chose à un agent
 - Recruter des agents pour accomplir une tâche
 - S'abonner à une source d'information / informer et mettre à jour

Protocole

◆ Protocole = séquence attendue d'une suite de messages

- Ex: demander de faire

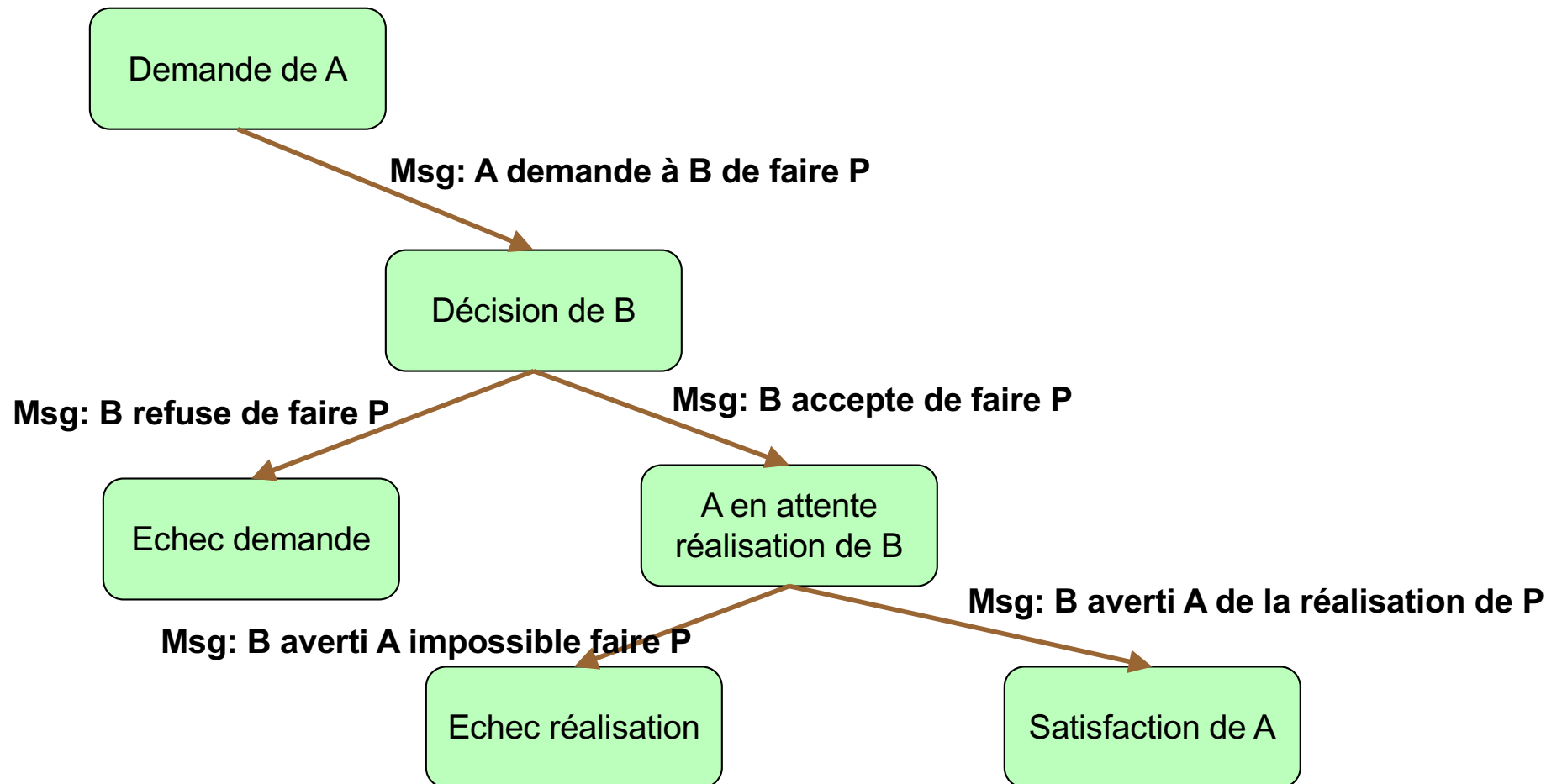
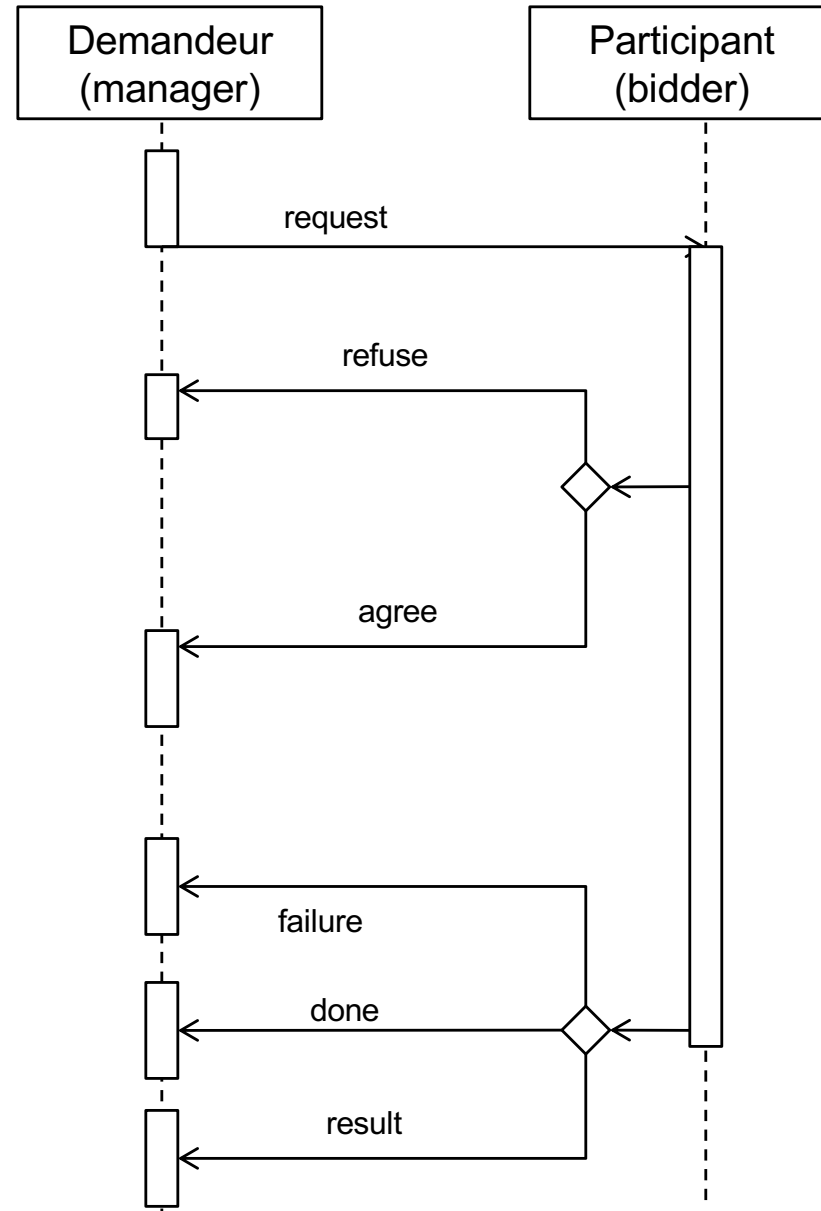


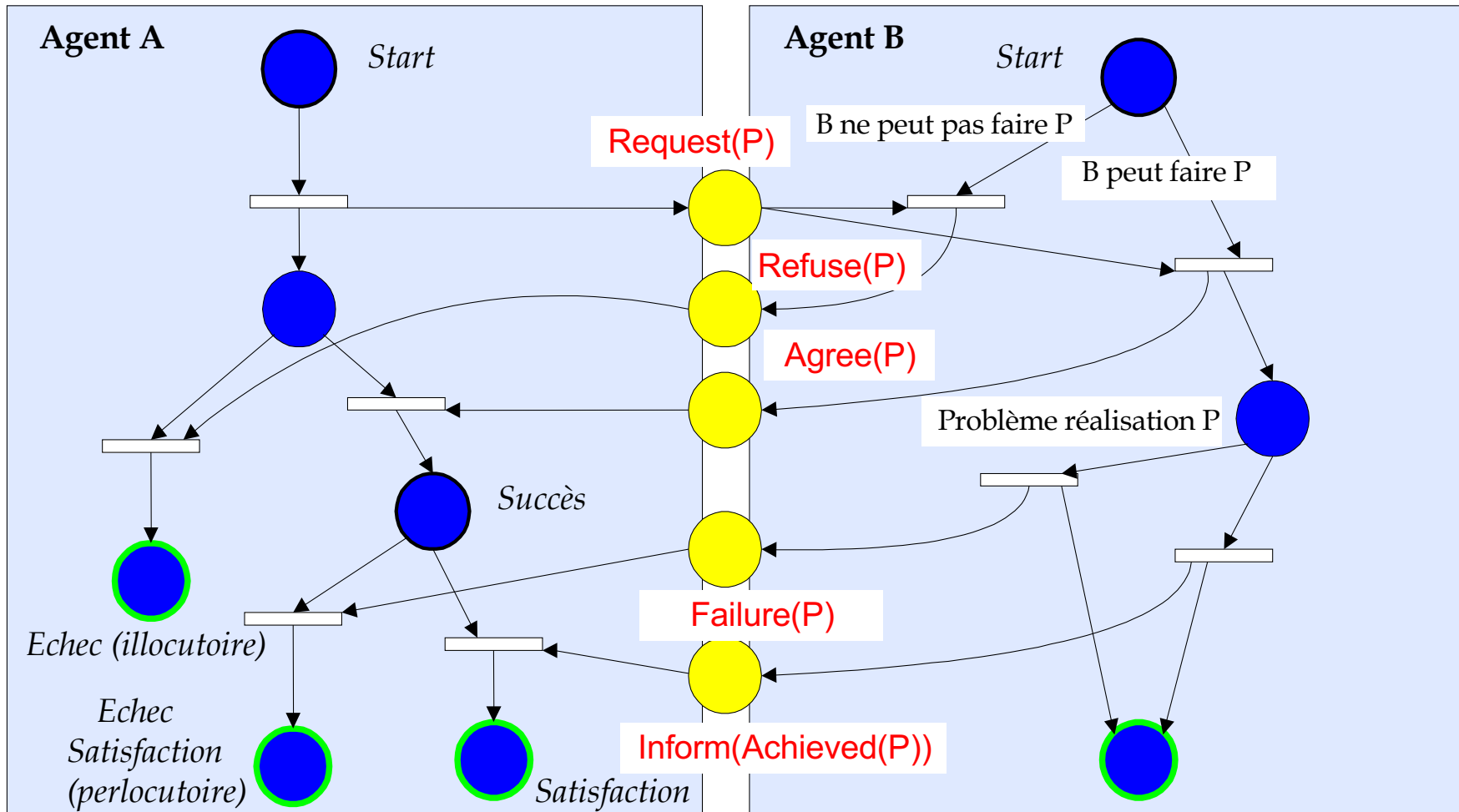
Diagramme de séquences Protocole de requête



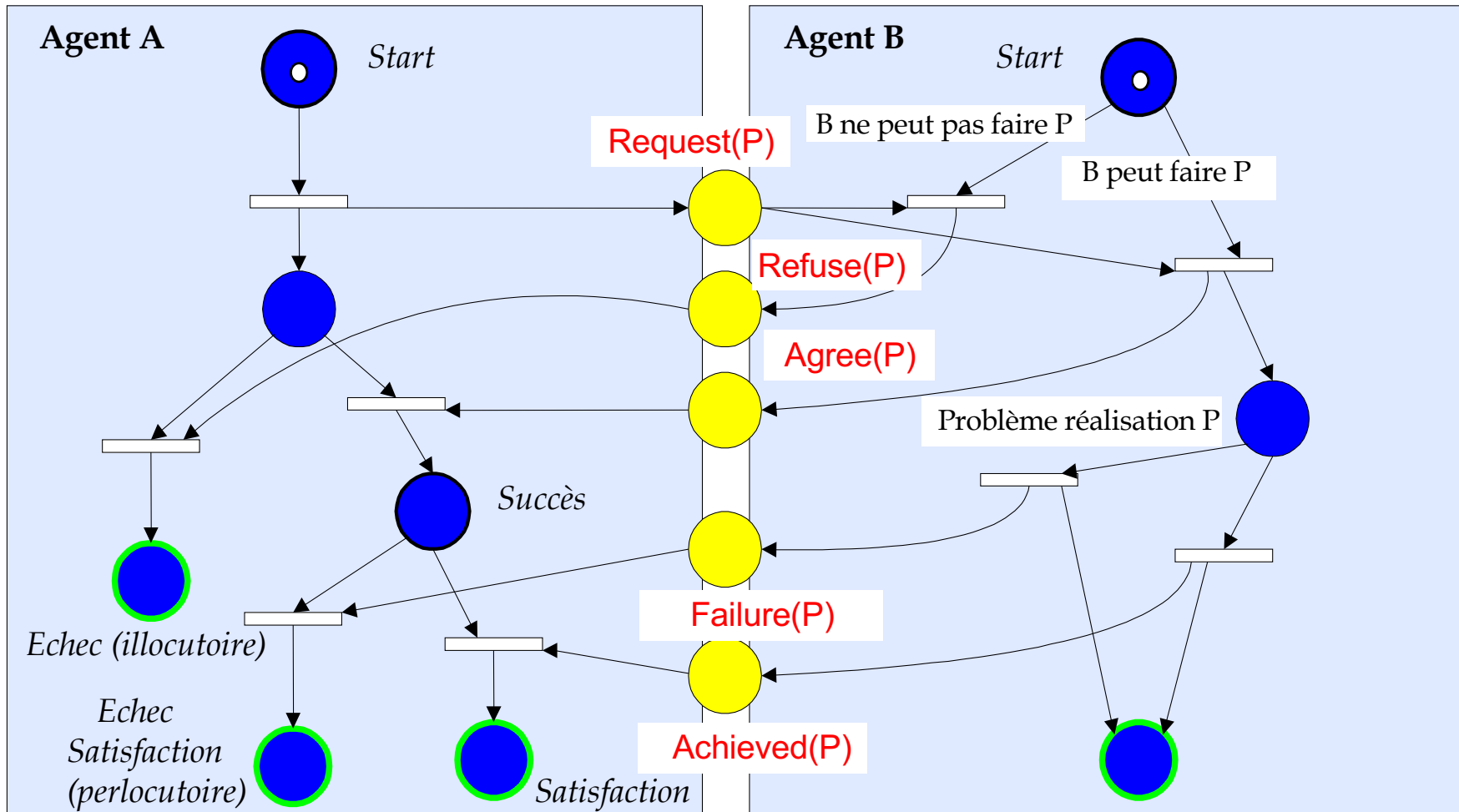
Quelques performatifs possibles

- **Accept Proposal**
- **Agree**
- **Cancel**
- **Call for Proposal (cfp)**
- **Confirm**
- **Disconfirm**
- **Failure**
- **Inform**
- **NotUnderstood**
- **Propose**
- **Query**
- **Refuse**
- **Reject**
- **Request**
- **Subscribe**
- **Unsubscribe**
- **Done**
- **Result**

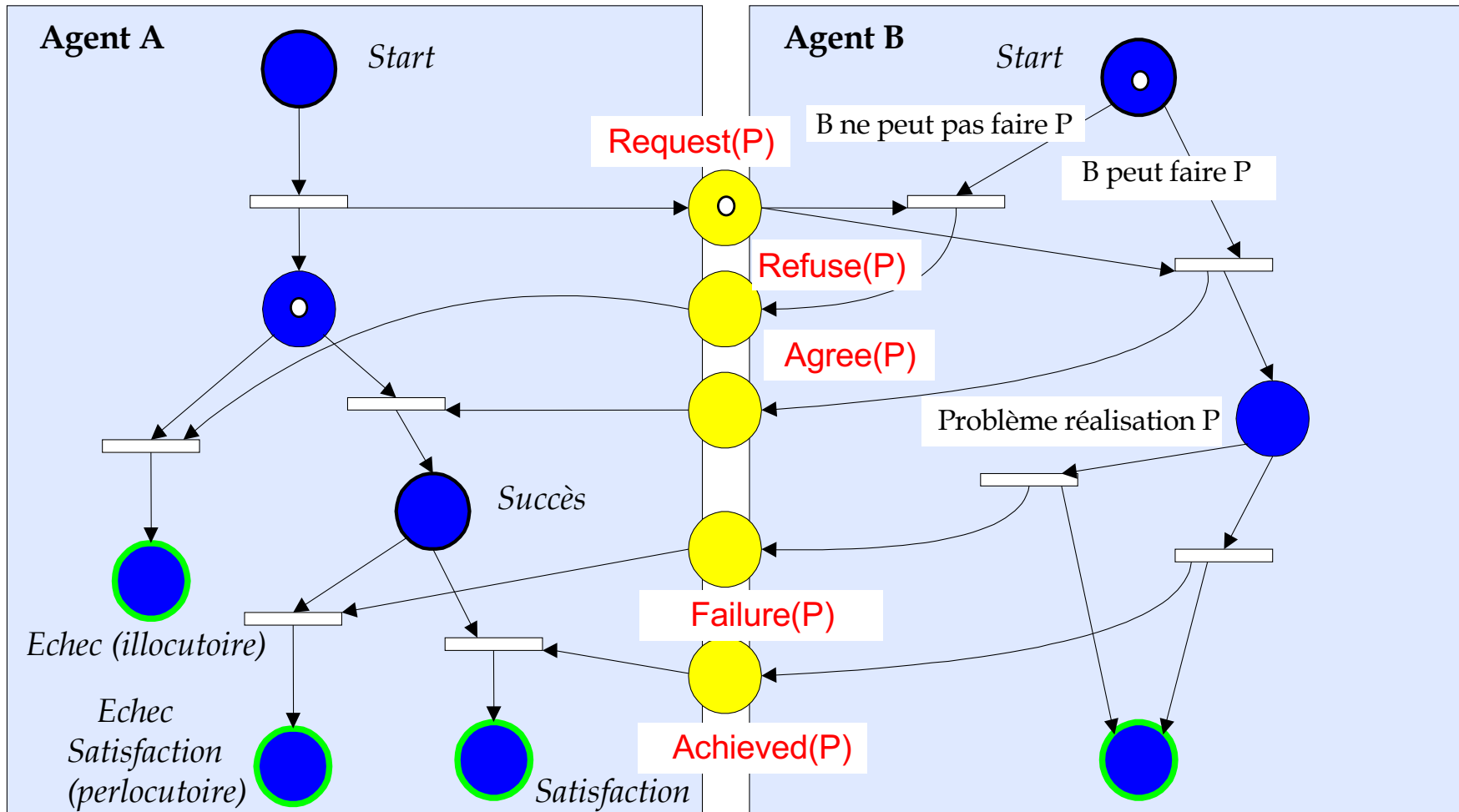
Autre modélisation: les réseaux de Petri



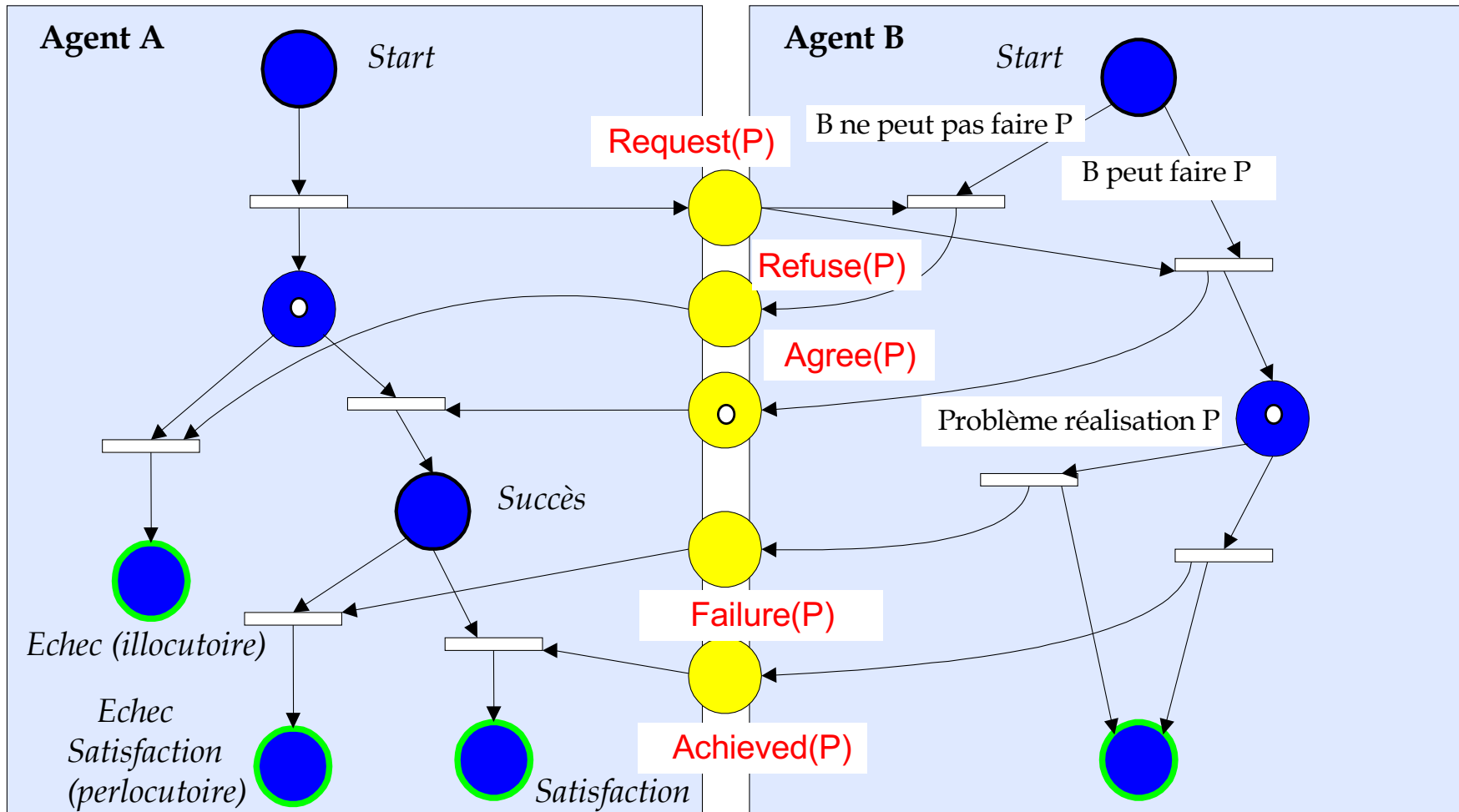
Autre modélisation: les réseaux de Petri



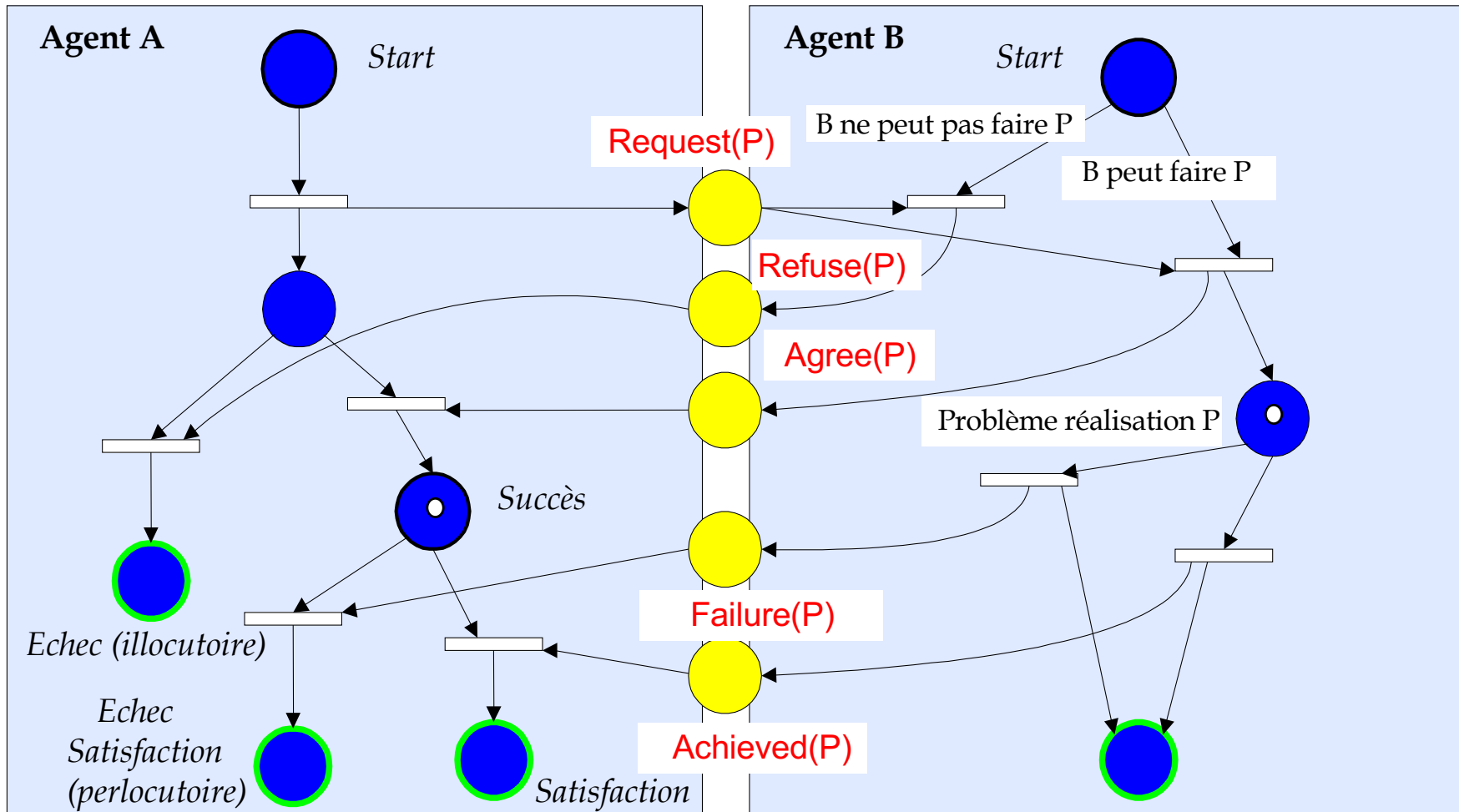
Autre modélisation: les réseaux de Petri



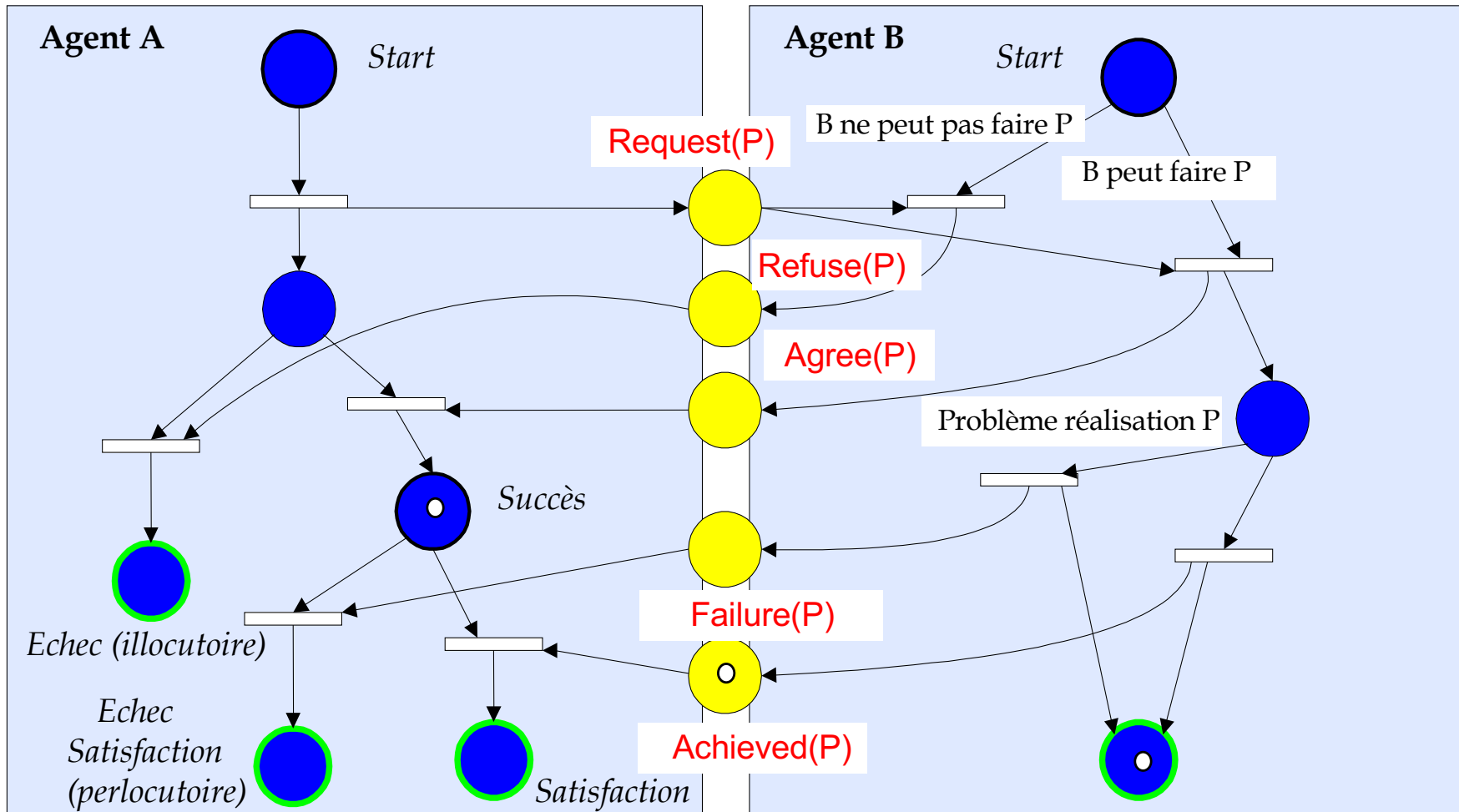
Autre modélisation: les réseaux de Petri



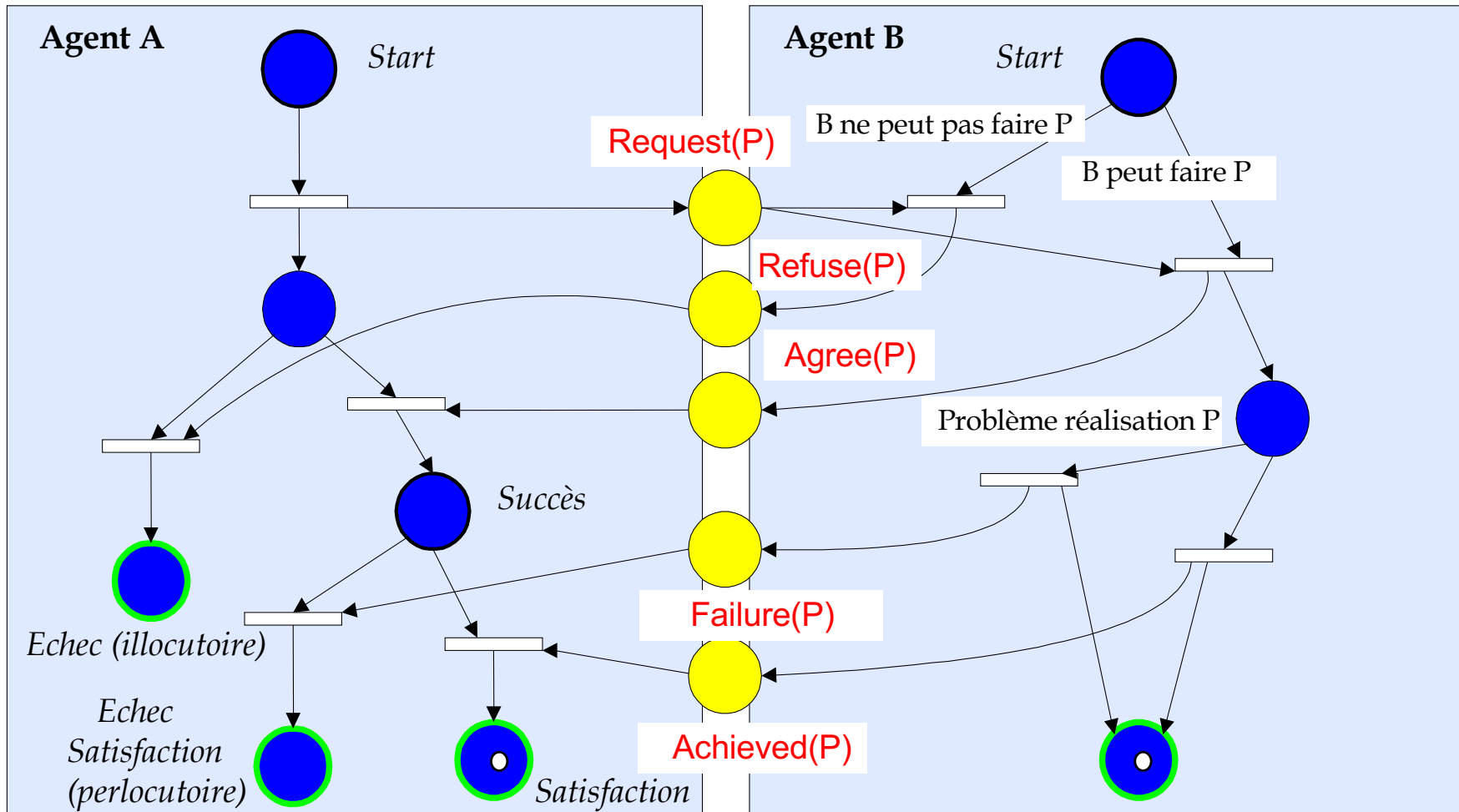
Autre modélisation: les réseaux de Petri



Autre modélisation: les réseaux de Petri



Autre modélisation: les réseaux de Petri



Appel collectif

◆ Appel collectif

- On veut qu'une tâche soit faite par un ensemble d'agents, quel que soit le nombre d'agents
- Pas de choix du « meilleur » agent
- Demande de type **request**

◆ L'initiateur appelle un ensemble d'agents

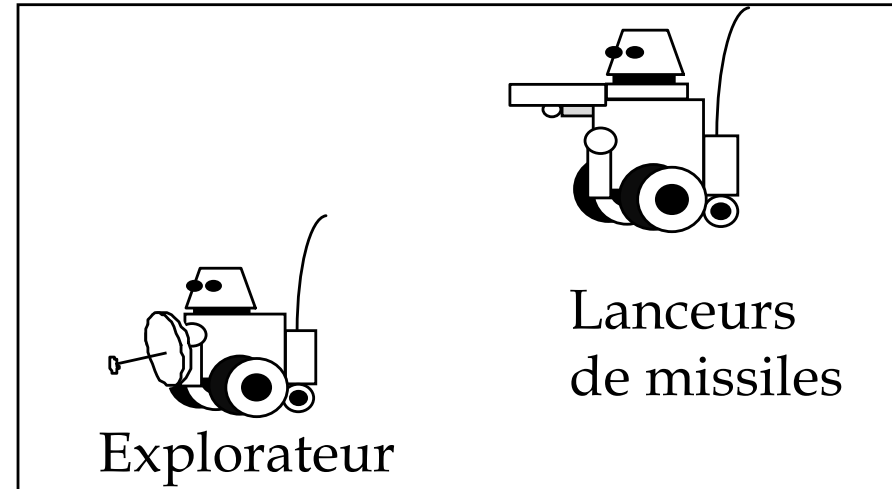
- Les agents répondent **accept** ou **refuse** et agissent immédiatement
- Si l'initiateur ne reçoit aucune acceptation (réception que de refuse après un certain délai), alors **Echec tâche T**

◆ Engagement des offrants: ne font qu'une seule tâche à la fois

Réseau contractuel dans Warbot

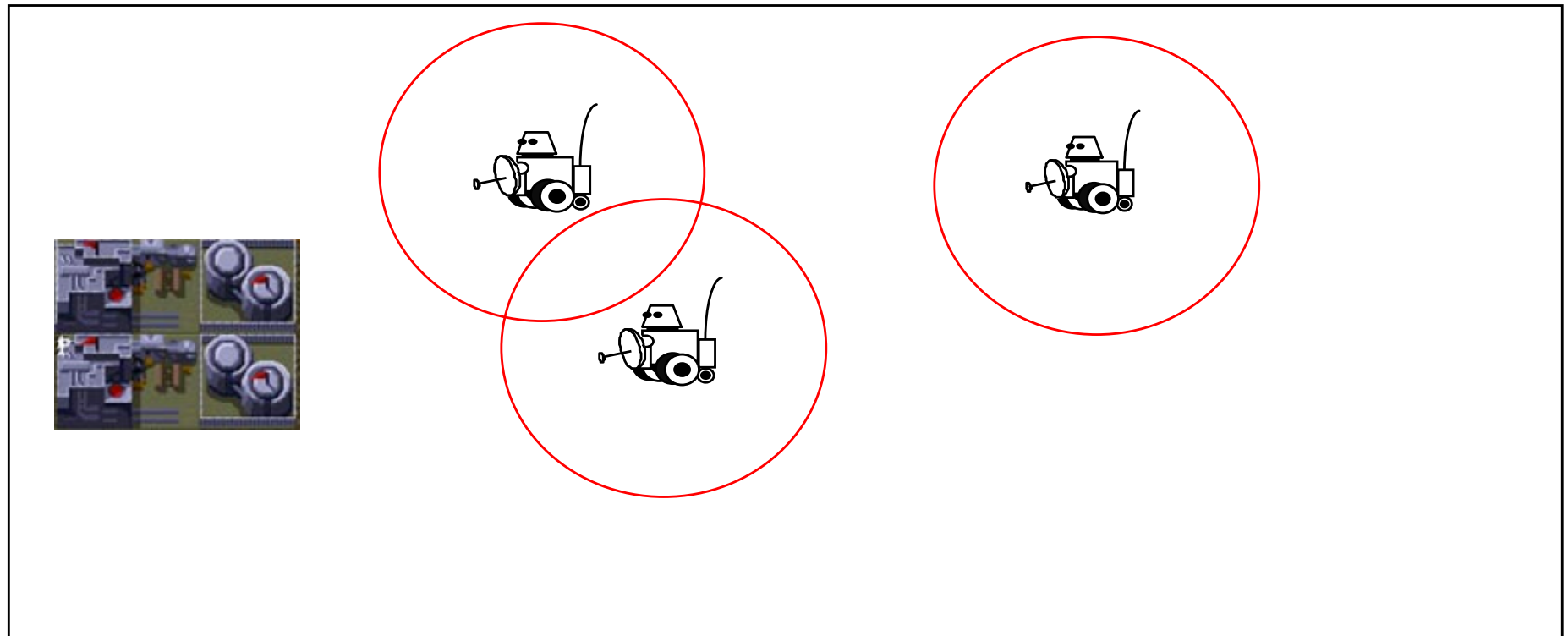
◆ **3 sortes d'agents**
(définit aussi 3 capacités différentes)

- Explorateurs
- Lanceurs de missiles
- Base



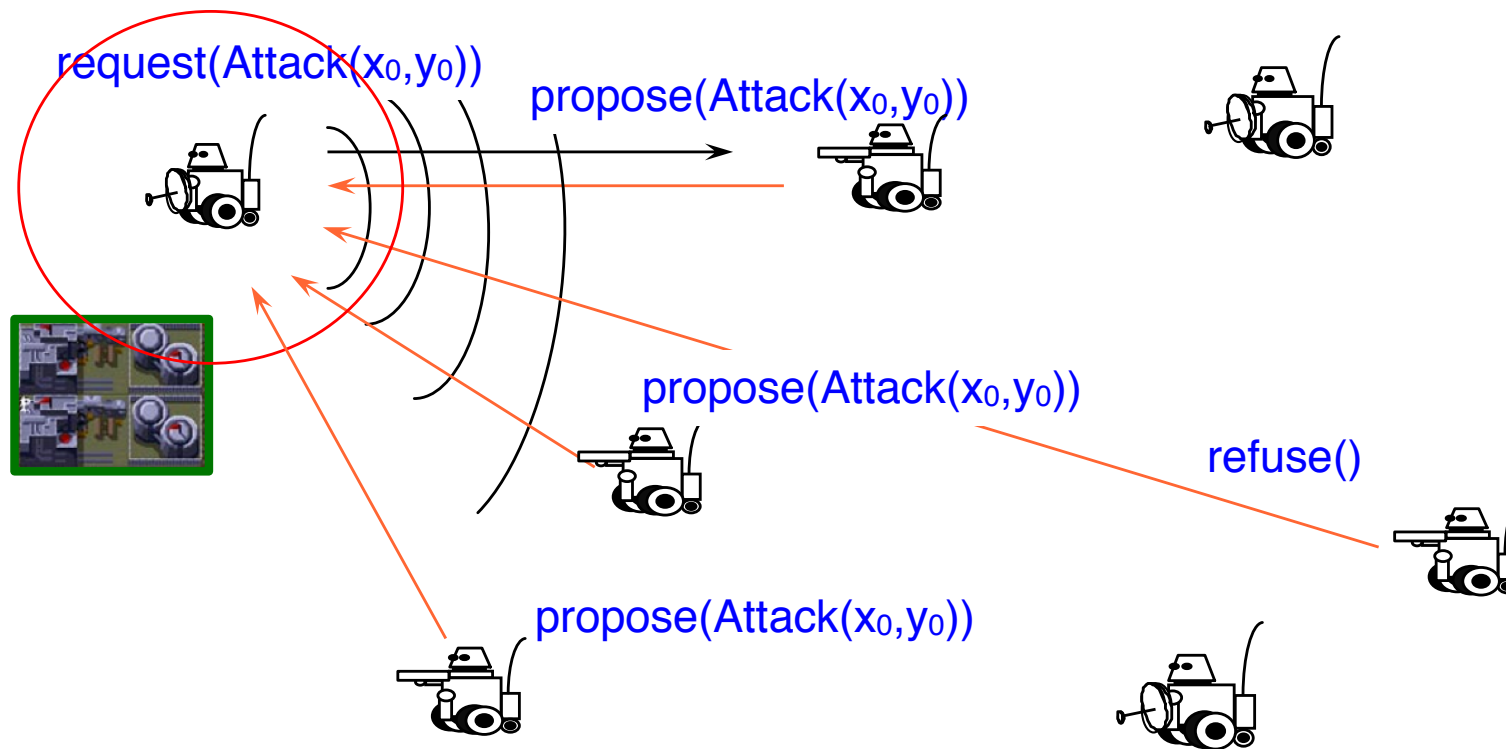
Phase 0: Exploration

- ◆ **Les explorateurs cherchent la base ennemie**



Phase 1,2: Appel pour attaquer

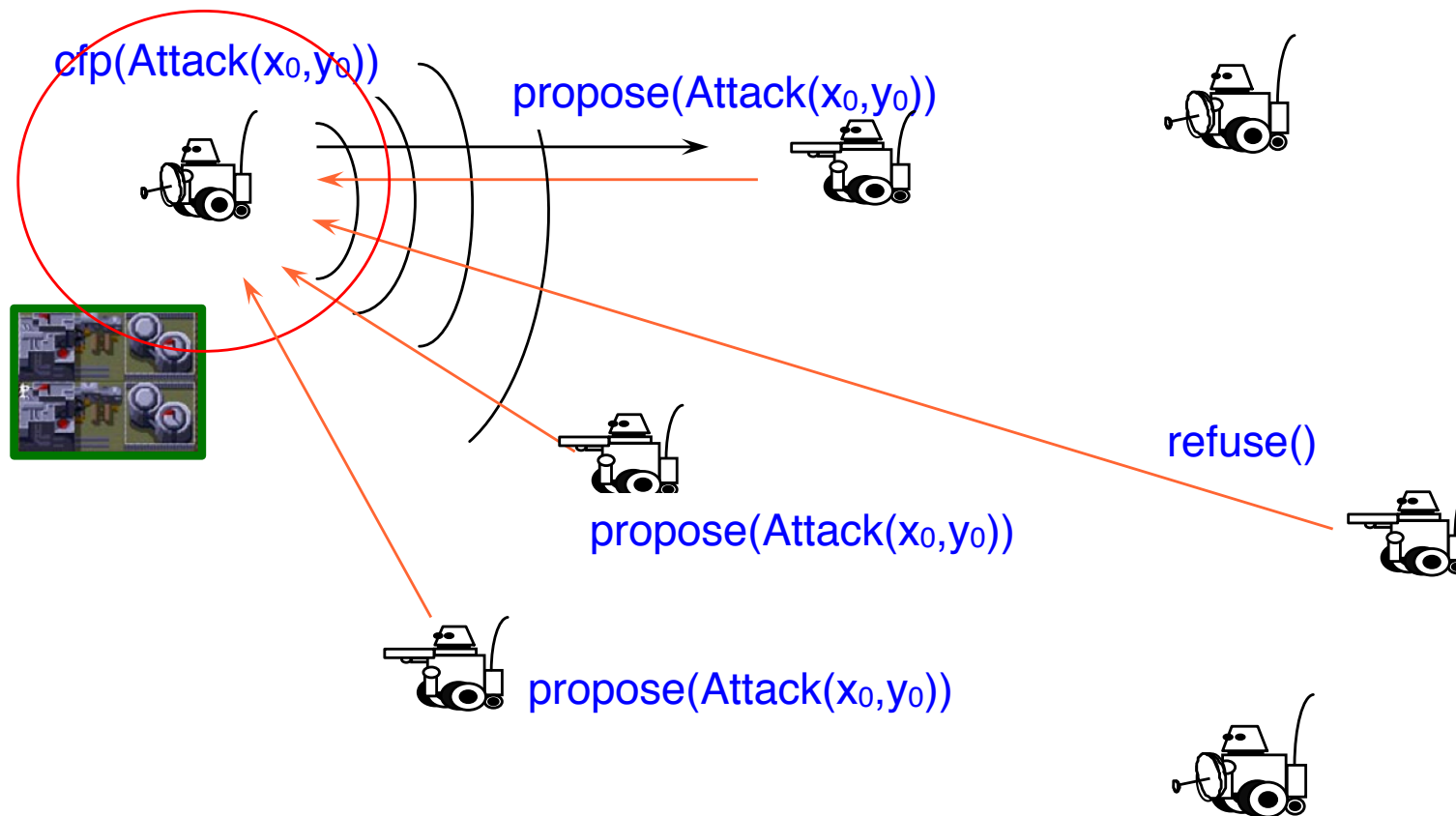
- ◆ Les explorateurs envoient des appels d'offres aux lanceurs de missile



Problème: tout le monde vient. A partir de quand attaque-t-on?

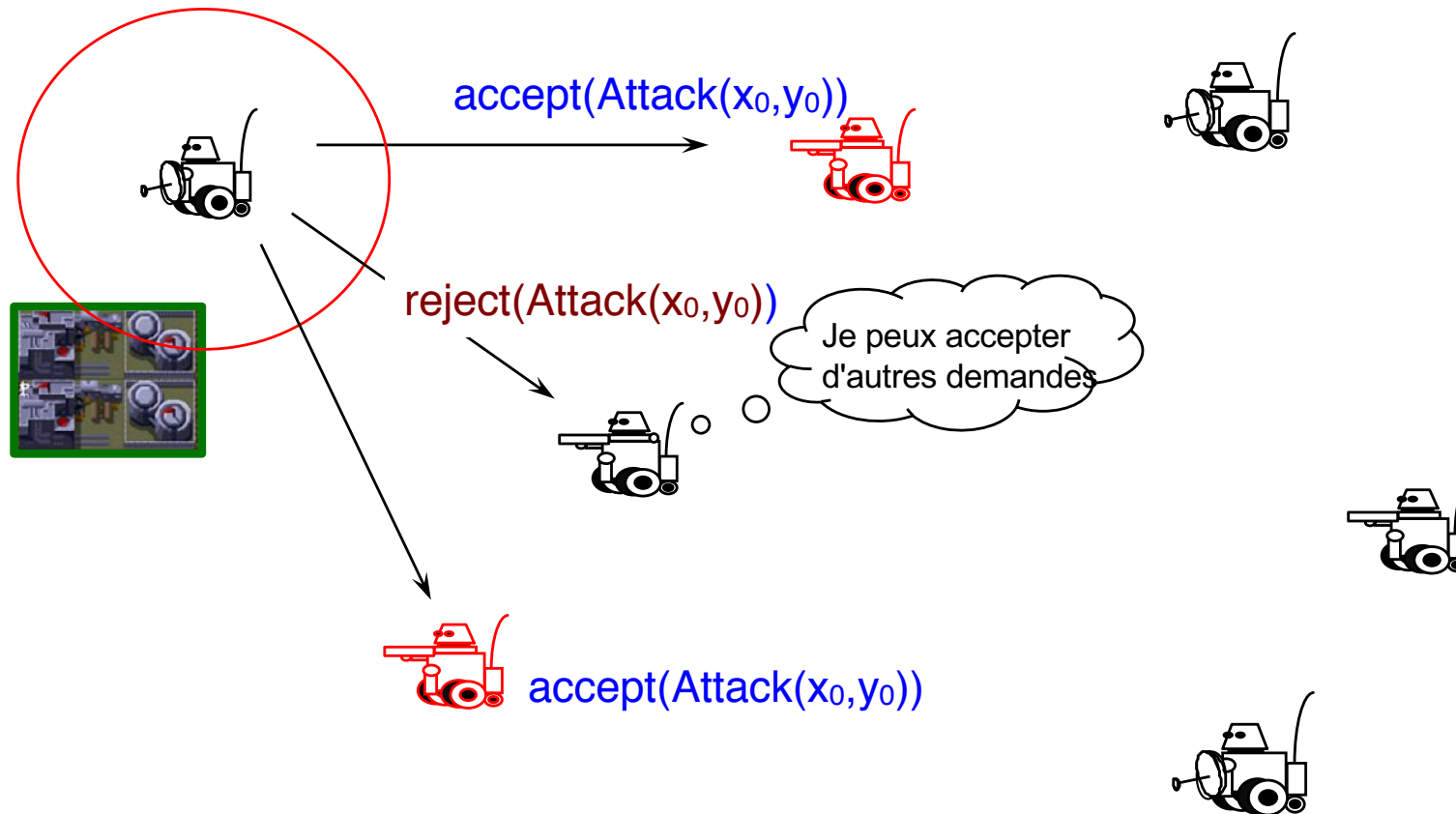
Phase 1, 2: Appel avec nombre défini

◆ Exemple: réseau contractuel



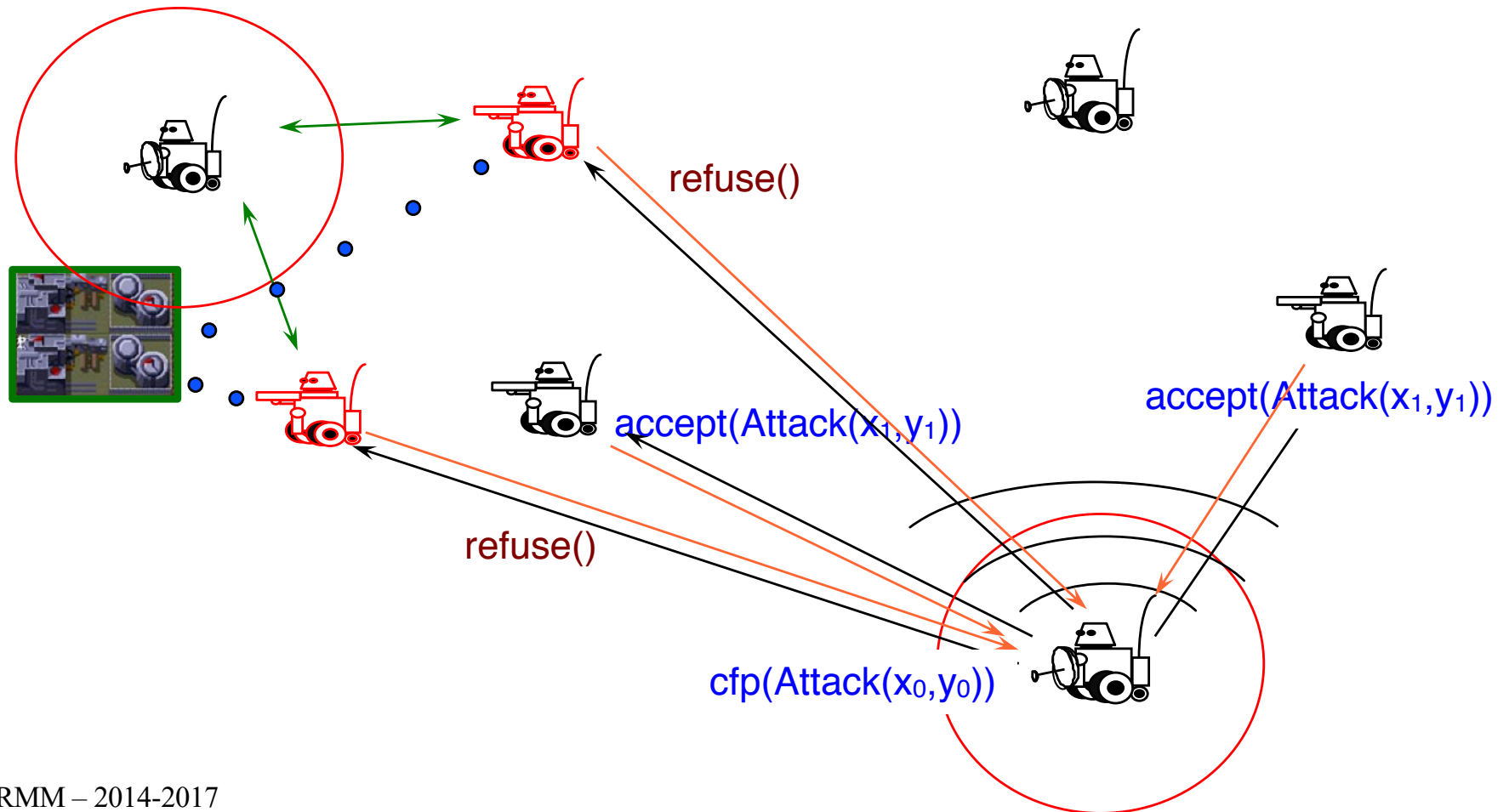
Phase 3 : Appel avec nombre défini

- ◆ Exemple: attente de 2 agents. Rejette les autres



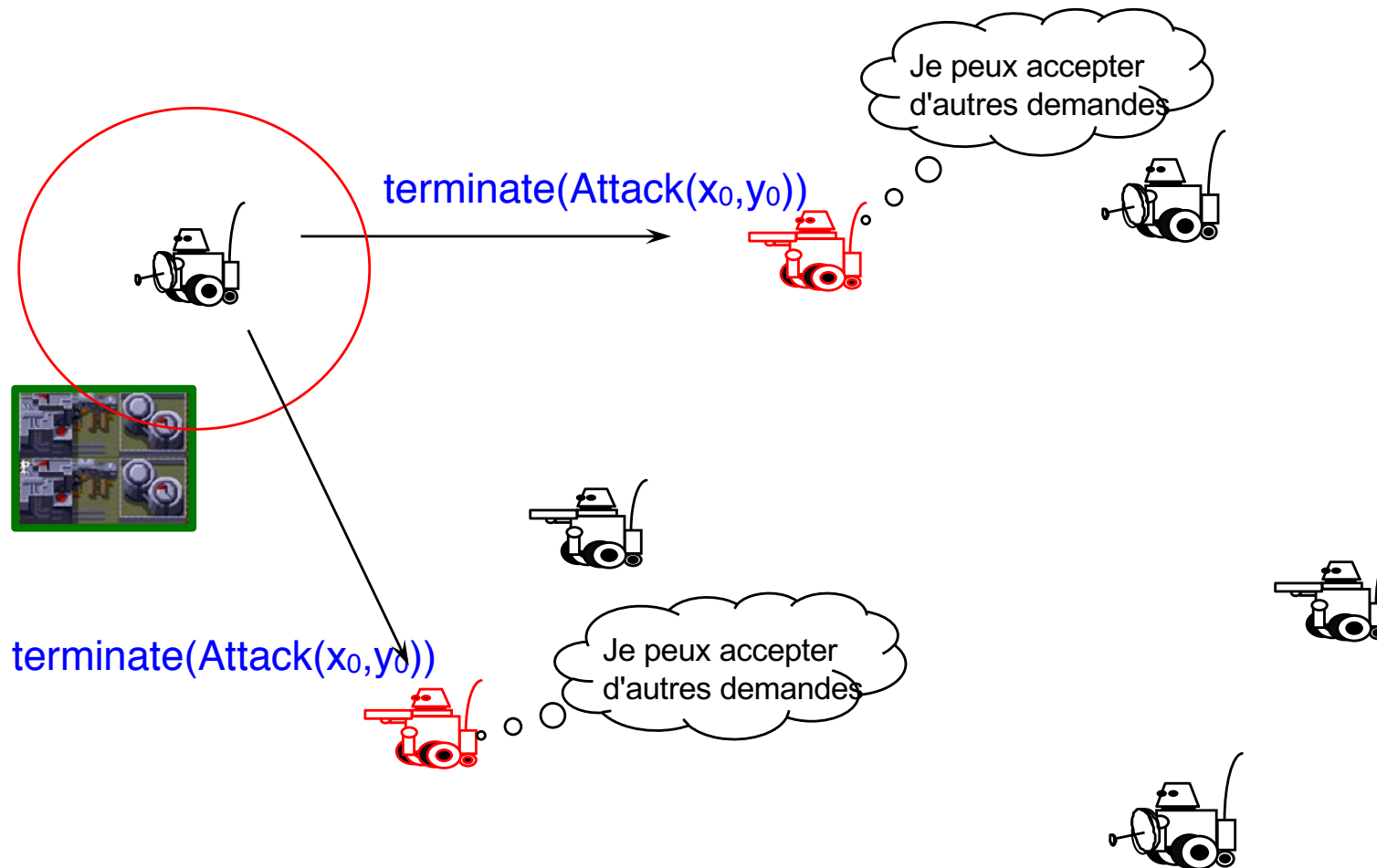
Phase 4: réalisation du contrat:

- ◆ Les contractants ne peuvent plus accepter d'autres propositions



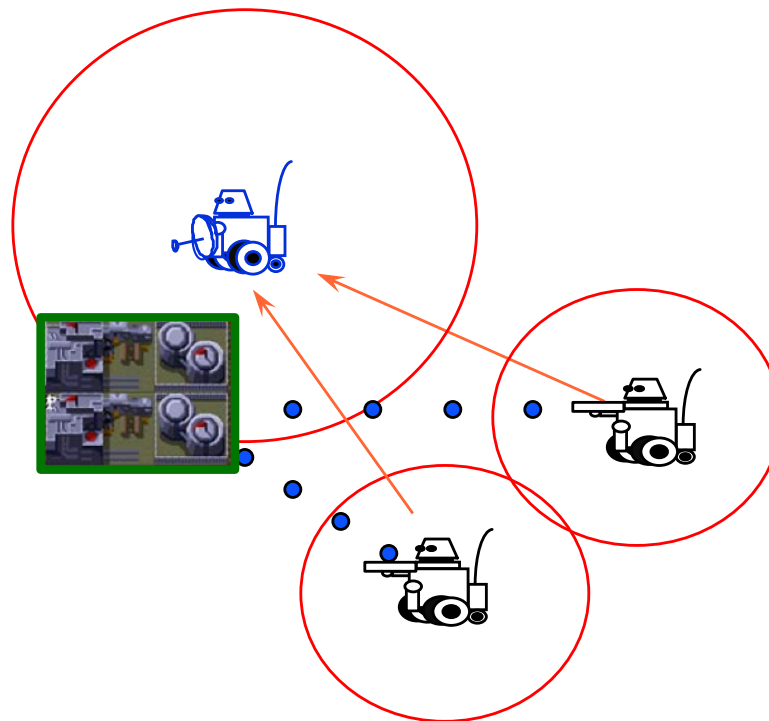
Phase 4 bis : fin du contrat

- ◆ Ici c'est le manager qui termine l'engagement



L'aveugle et le paralytique

- ◆ Une attaque dirigée par un explorateur: les lanceurs de missile ne perçoivent pas la cible

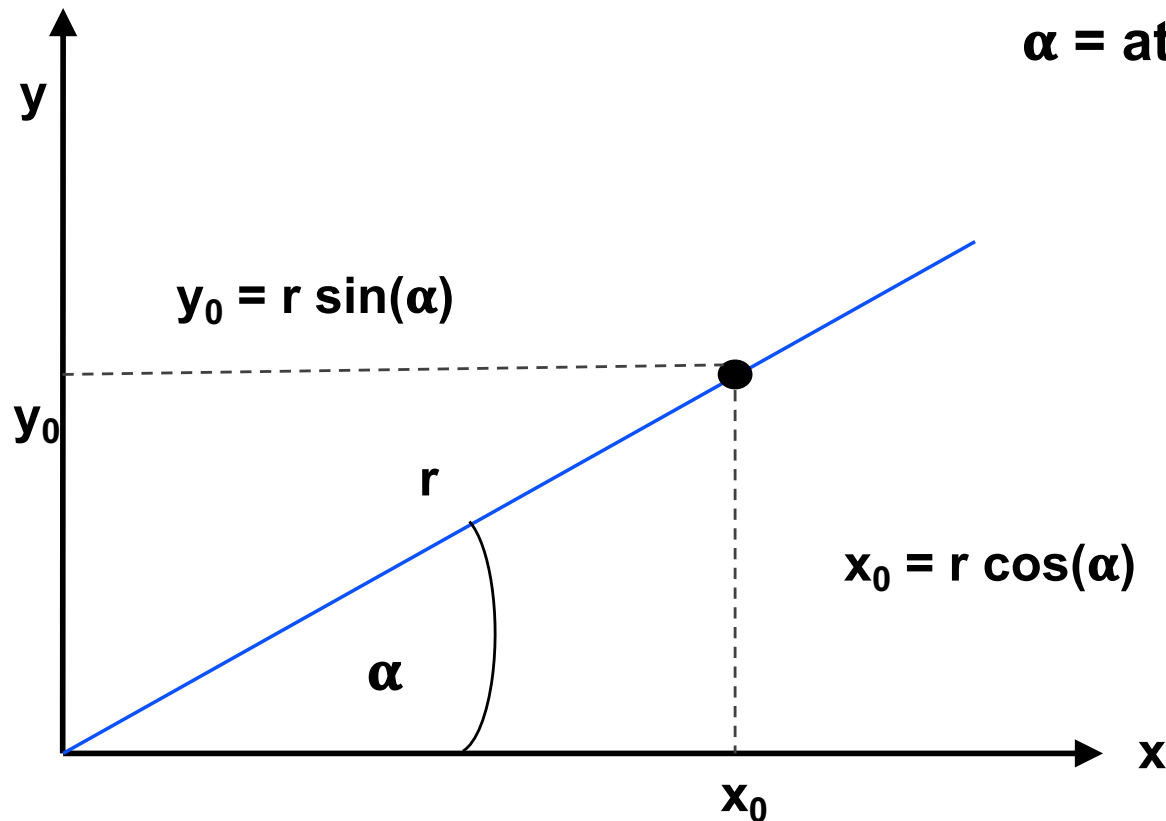


Rappel de trigonométrie

- ◆ Passage de coordonnées polaires \leftrightarrow cartésiennes

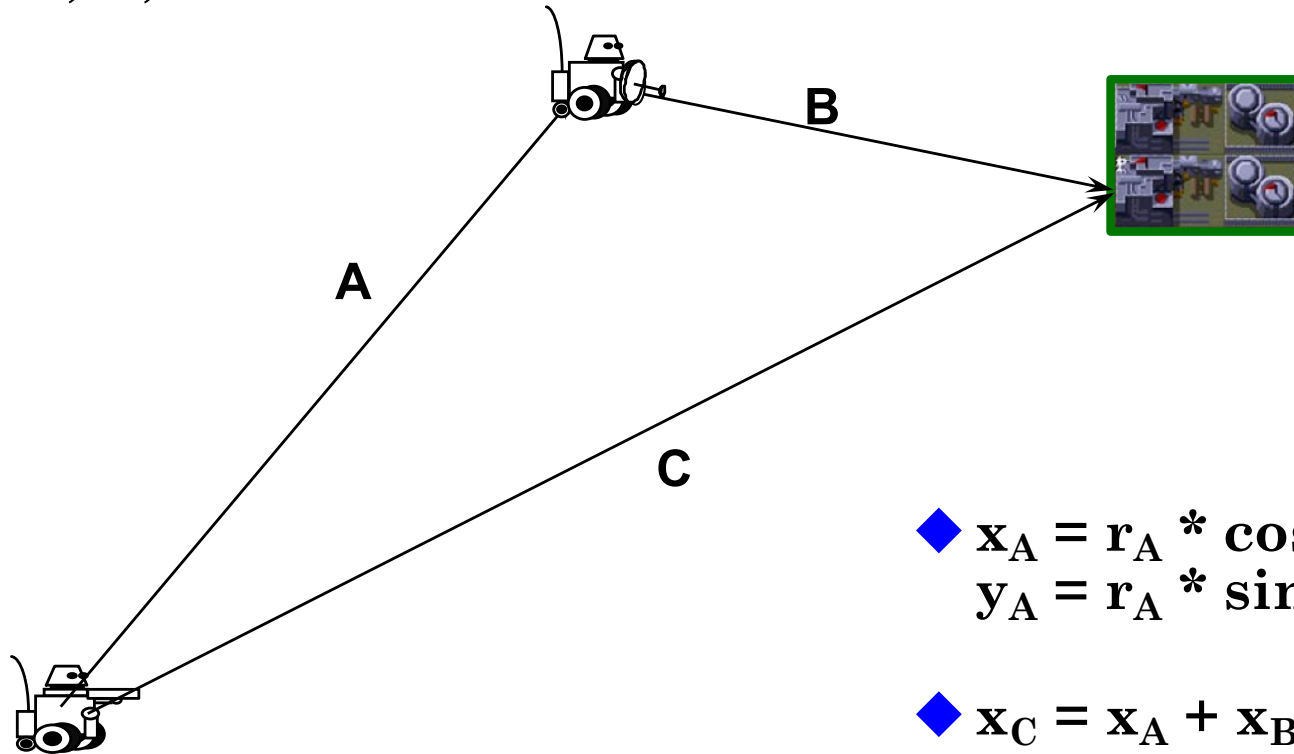
$$r = \sqrt{x_0^2 + y_0^2}$$

$$\alpha = \text{atan}(y_0/x_0)$$



Récupérer en polaire les infos pour aller vers ou tirer la base

- ◆ Si A, B, C, des vecteurs: $C = A + B$



- ◆ B est donné par la perception de l'explorateur
- ◆ A est donné par le message entre explorateur et combattant

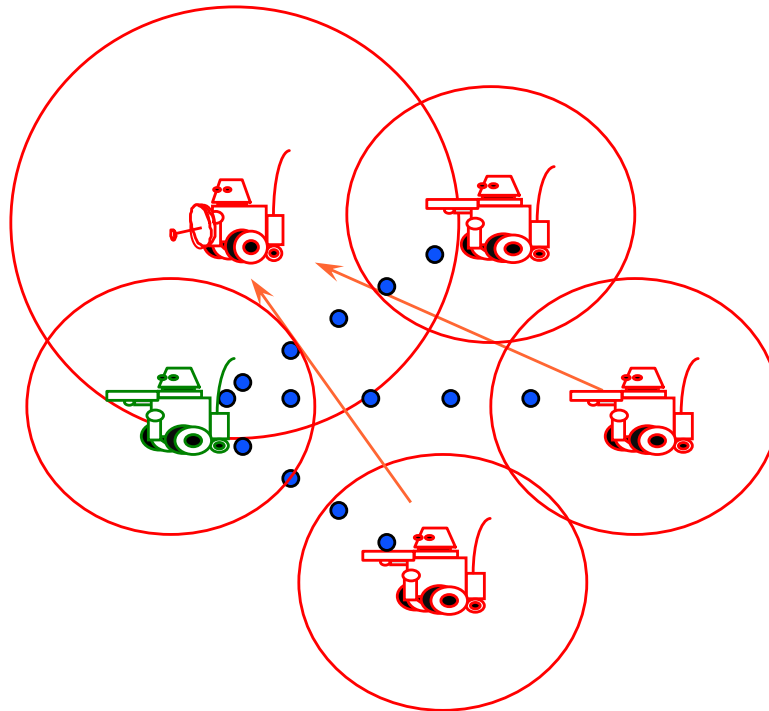
- ◆ $x_A = r_A * \cos(\theta_A)$
 $y_A = r_A * \sin(\theta_A)$

- ◆ $x_C = x_A + x_B$
 $y_C = y_A + y_B$

- ◆ $r_C = \text{Sqrt}(x_C^2 + y_C^2)$
 $\theta_C = \text{atan}(y_C / x_C)$

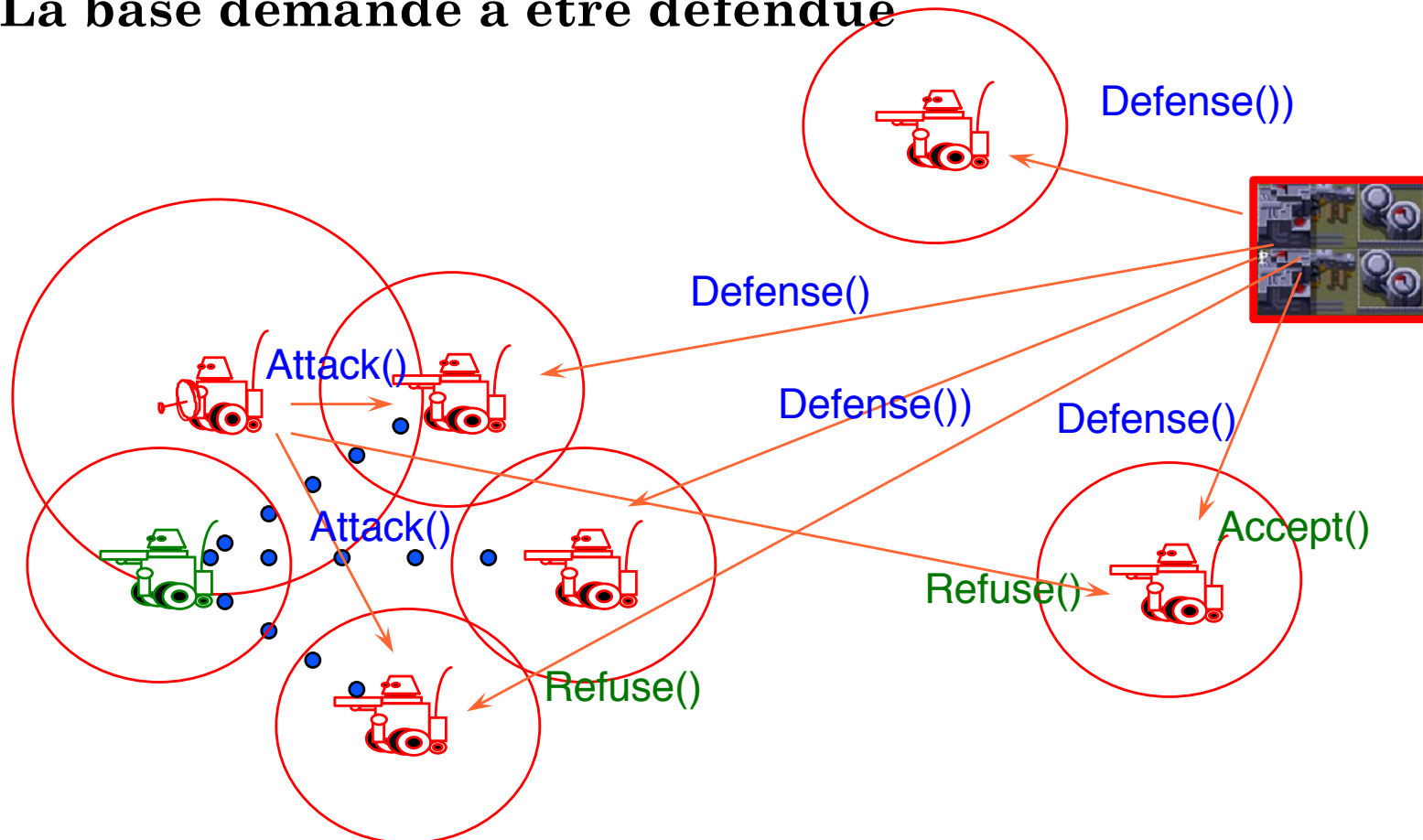
Phase 2: Attaquer à plusieurs sans être vu

- ◆ Les explorateurs dirigent l'attaque contre un lanceur de missile qui ne voit pas ses assaillants



Attaquer et défendre

- ◆ Les explorateurs dirigent l'attaque contre un lanceur de missile qui ne voit pas ses assaillants
- ◆ La base demande à être défendue



Le comportement des robots tireurs

De tireur // sans engagement

Si je reçois un message `request(attack,p0)`

et si distance avec $p0 < \text{dist-max}$, alors lancer-missile direction vers $x0, y0$

// pas d'engagement

De tireur // avec engagement

Si je suis **disponible** ou état **en-attaque** et je reçois un message `request(attack,p0)`

et si distance avec $p0 < \text{dist-max}$, alors lancer-missile direction vers $x0, y0$

et `etat <- en-attaque`

Si je suis en-defense et je reçois un message `request(attack,p0)`

alors ne rien changer

Si je suis en-attaque et je n'ai pas reçu de messages d'attaque depuis n tours

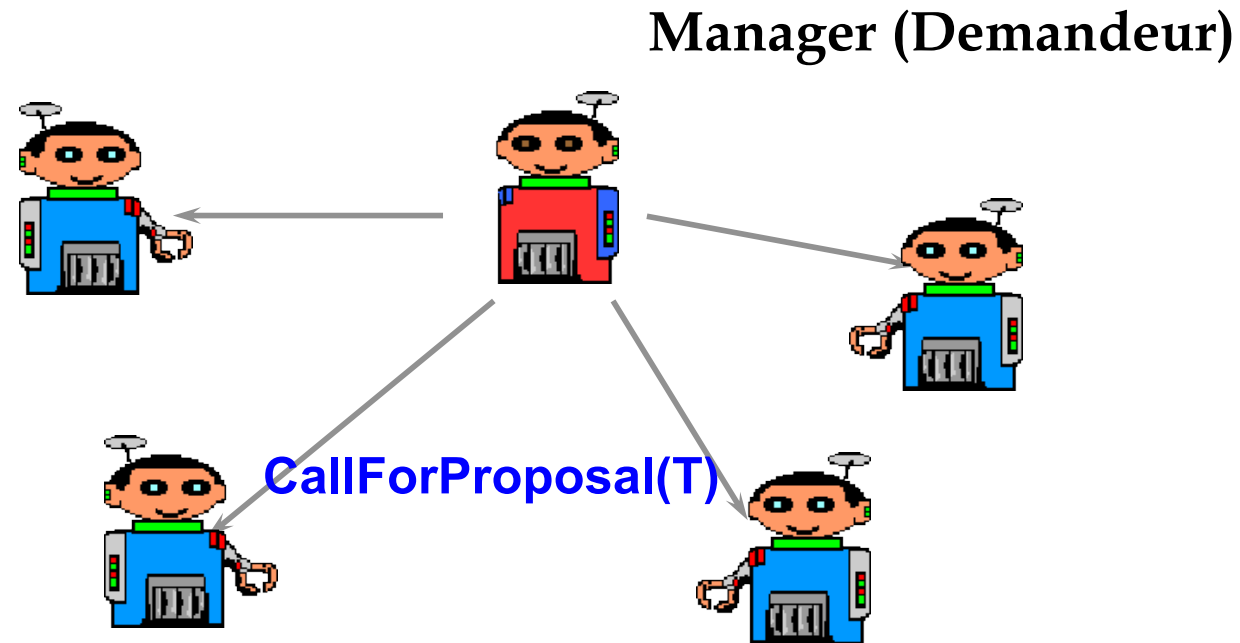
alors `etat <- disponible`

Un modèle classique de conversation pour l'allocation de tâche: le réseau contractuel

- ◆ **Le plus connu des modèles de conversation pour la coordination par allocation de tâches.**
- ◆ **Utilise le protocole des appels d'offres des marchés d'états.**
- ◆ **Deux types d'agents (deux rôles)**
 - Manager et Bidder (Offrant)
- ◆ **Est accompli en 4 phases**
 - Appel d'offre
 - Offre (bids)
 - Attribution du contrat
 - Signature du contrat et travail

Réseau contractuel: phase 1

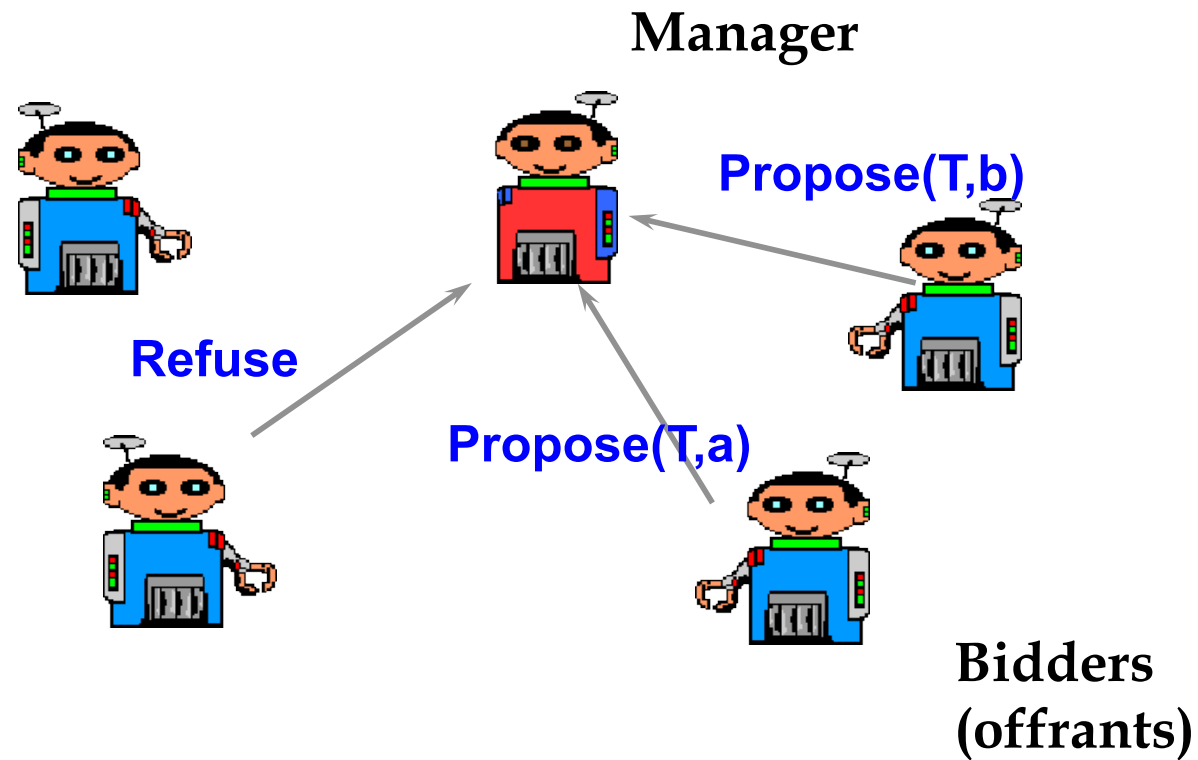
◆ Appel d'offre (call for proposal)



Offrants potentiels (bidders, offrants)

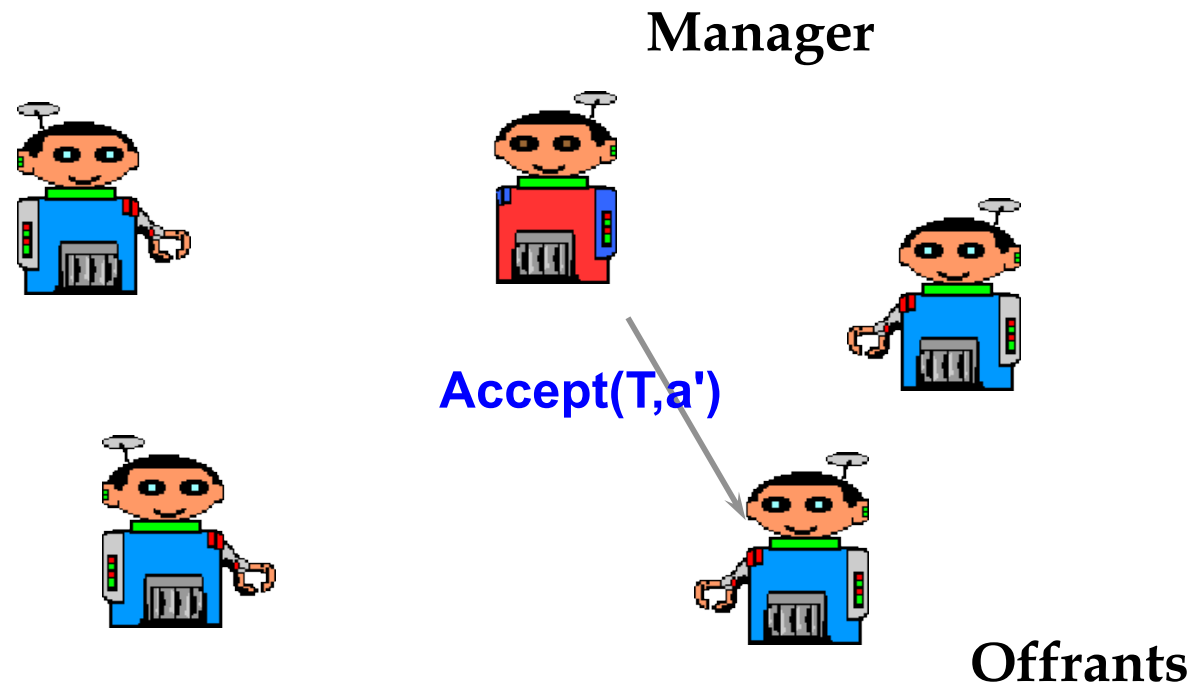
Réseau contractuel: phase 2

◆ Offres (Propose)



Réseau contractuel: phase 3

- ◆ Sélection: attribution du contrat (attribution « accept-proposal »)



Réseau contractuel: phase 4

◆ Travail et résultat

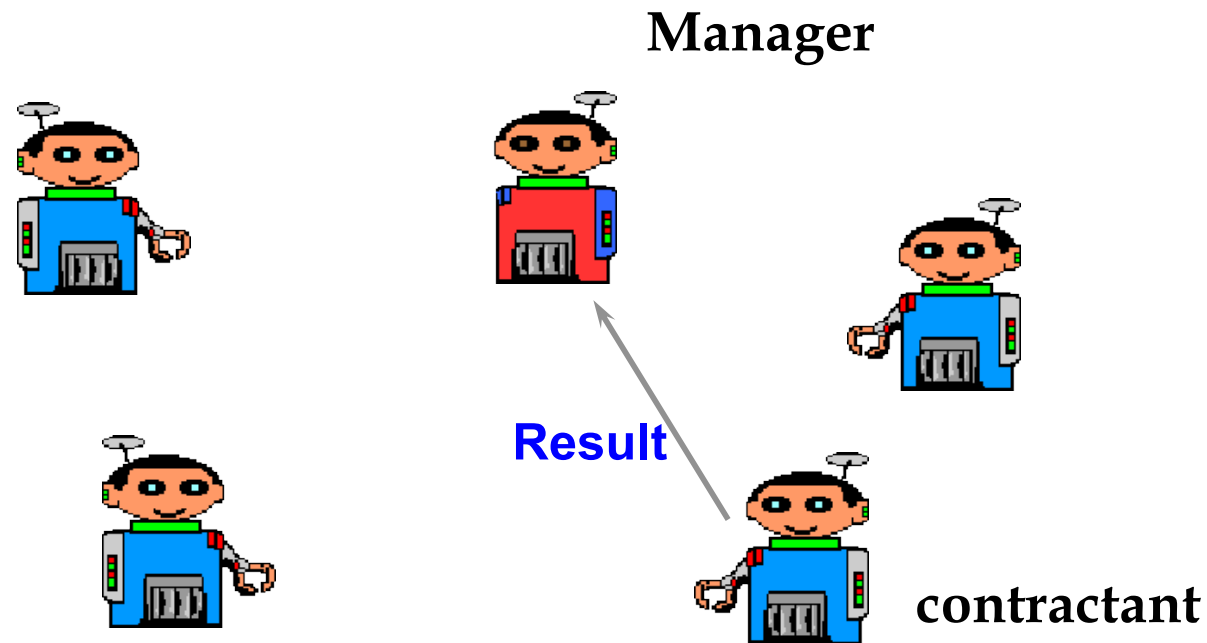
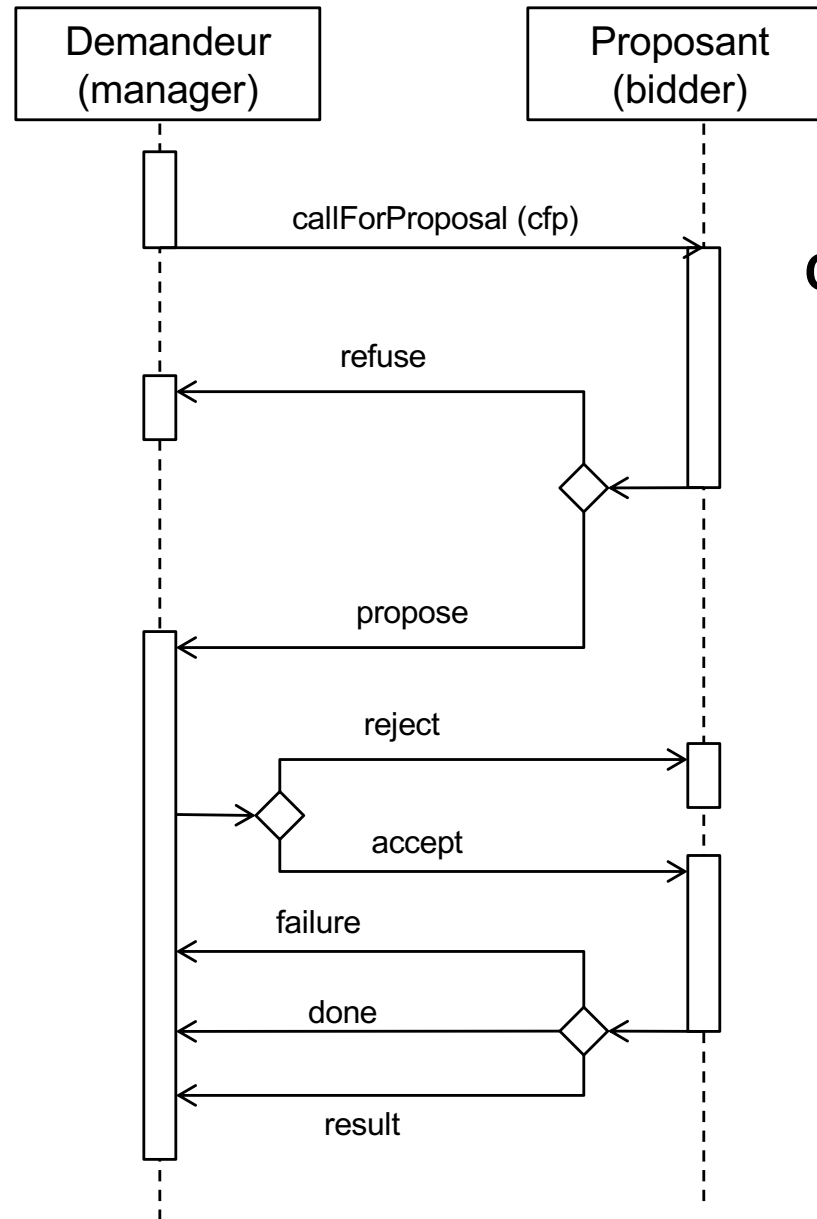
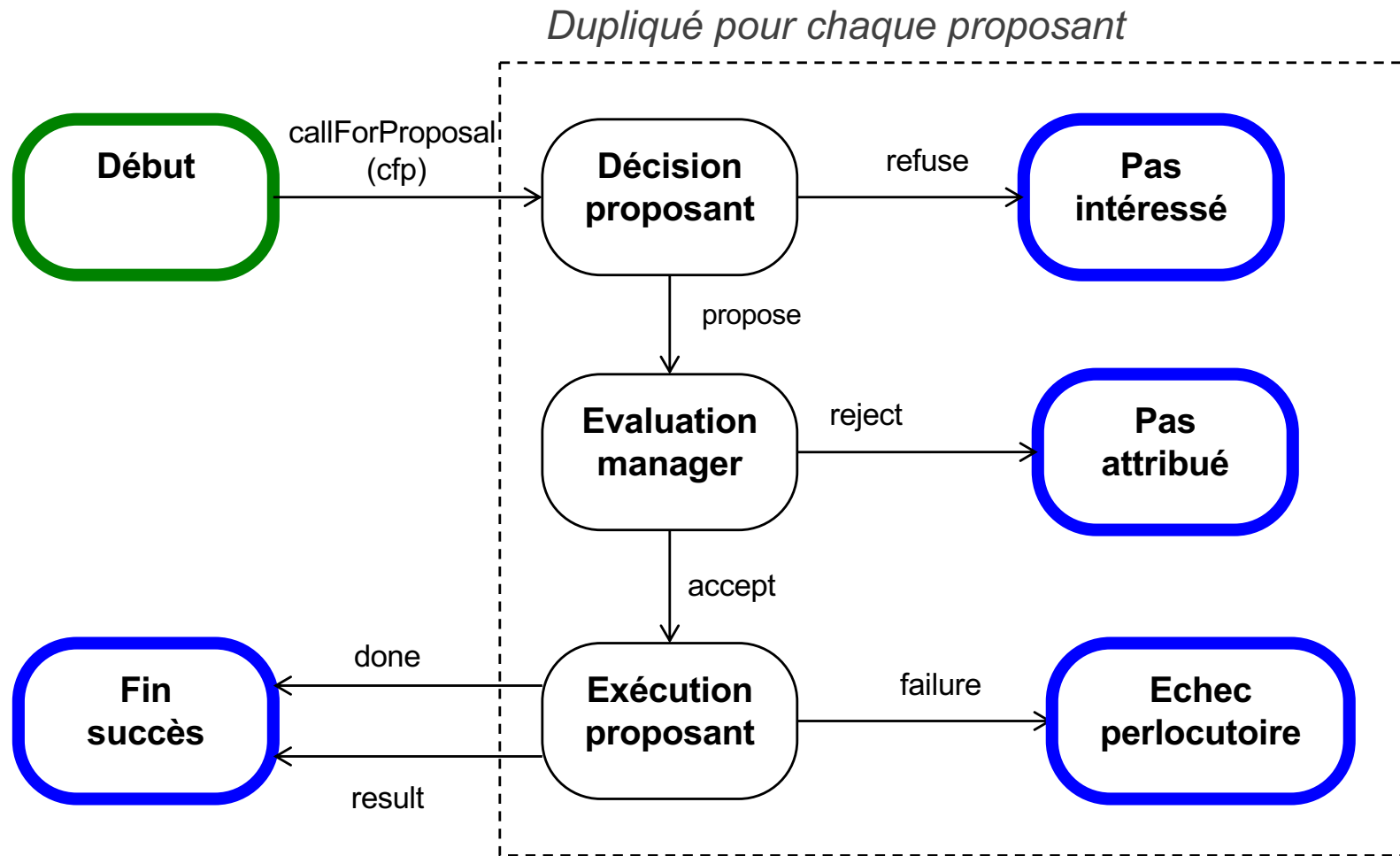


Diagramme de séquences Réseau contractuel



Contract net protocol

Réseau contractuel - automate



Nécessité de prendre en compte le dialogue pour chaque proposant

Implémentation d'un protocole

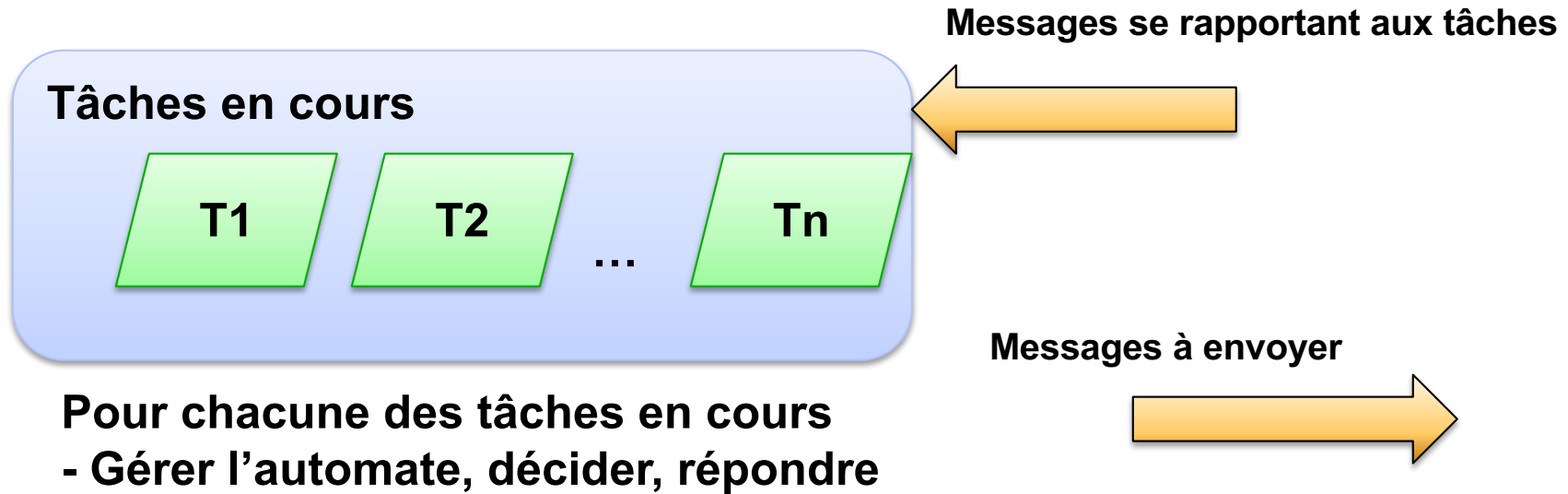
◆ Chaque agent (manager, offrant) doit gérer « son » dialogue avec l'autre

- Ajouter la notion de tâche en cours
 - ☞ A demande à B de faire P
 - ☞ Création d'un agenda des tâches en cours
 - 1 pour l'initiateur (manager), 1 pour les offrants (participants)

◆ Communication

- Tâches demandée
 - ☞ A qui,
 - ☞ Quand, délai? (coût?)
 - ☞ Faire quoi

Gestion des tâches



Implémenter le réseau contractuel? #1

◆ Du point de vue du manager (initiator)

Appel d'offre:

J'envoie une demande,

`cfp(demande, T)` à une liste d'agents L

 où T est le numéro de cfp

Je crée une tâche avec les infos (T, L, nom-demande, Prop)

où Prop est la liste des proposants (initialement vide)

Réponse appel d'offre

Si je reçois un message de type `refuse (T)`,

 je supprime l'émetteur de la liste L des possibles pour la tâche T

 si la liste L est vide et que Prop est vide, alors **Echec tâche T**

Si je reçois un message de type `propose (T)`,

 je supprime l'émetteur de la liste L,

 j'ajoute l'émetteur et sa proposition à la liste Prop

 si L est vide, passer à la **phase décision**

// note: on gère généralement un dead line (time out) pour éviter d'attendre des réponses qui ne viendront jamais...

Implémenter le réseau contractuel? #2

◆ Du point de vue du manager (initiator)

Décision:

Pour la tâche T, je choisis la proposition P qui me paraît la meilleure,
pour tout a = agents de Prop,

si a est le meilleur proposant de P je lui signale que e contrat lui est
attribué : **accept (T)**

sinon: envoyer à tous les autres agents de Prop que leurs propositions
ne sont pas acceptées: **reject(T)**

Fin de contrat:

Si je reçois le message **failure(T)** alors **Echec tâche T**

Si je reçois le message **done(T)** (ou **result(T,r)**)

alors **Succès tâche T**

Implémenter le réseau contractuel? #3

◆ Du point de vue de l'offrant (participant)

Réponse à CFP:

Dès que je reçois un message de type `cfp(demande, T)`

Si je peux (et veux faire T):

je réponds à l'émetteur `propose (T, args)`, et je mémorise un projet en cours avec `T(manager ← émetteur)`

Si ne ne peux pas (ou ne veux pas faire T):

je réponds à l'émetteur `refuse(T)`,

Réception de l'attribution du contrat :

Si je reçois le message `accept (T)` alors faire T

Si je reçois le message `reject (T)` alors supprimer T de la liste des projets

Faire T:

Accomplir T (FSM)

Si T est terminé,

envoyer message `done(T)` (ou `result(T,r)`)
à l'initiator de T (émetteur du CFP)

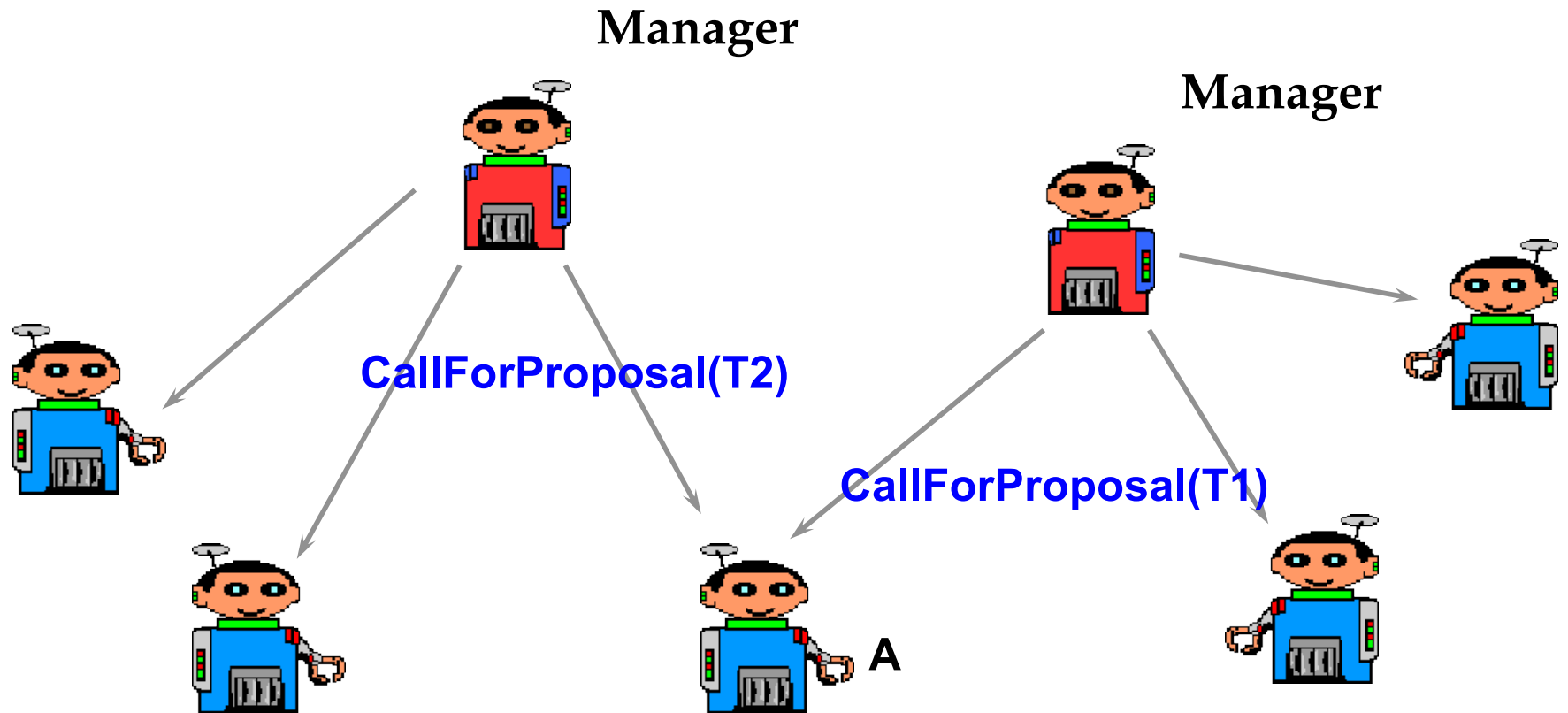
Si problème dans la réalisation de T,

envoyer message `failure(T)`

Problèmes avec plusieurs managers

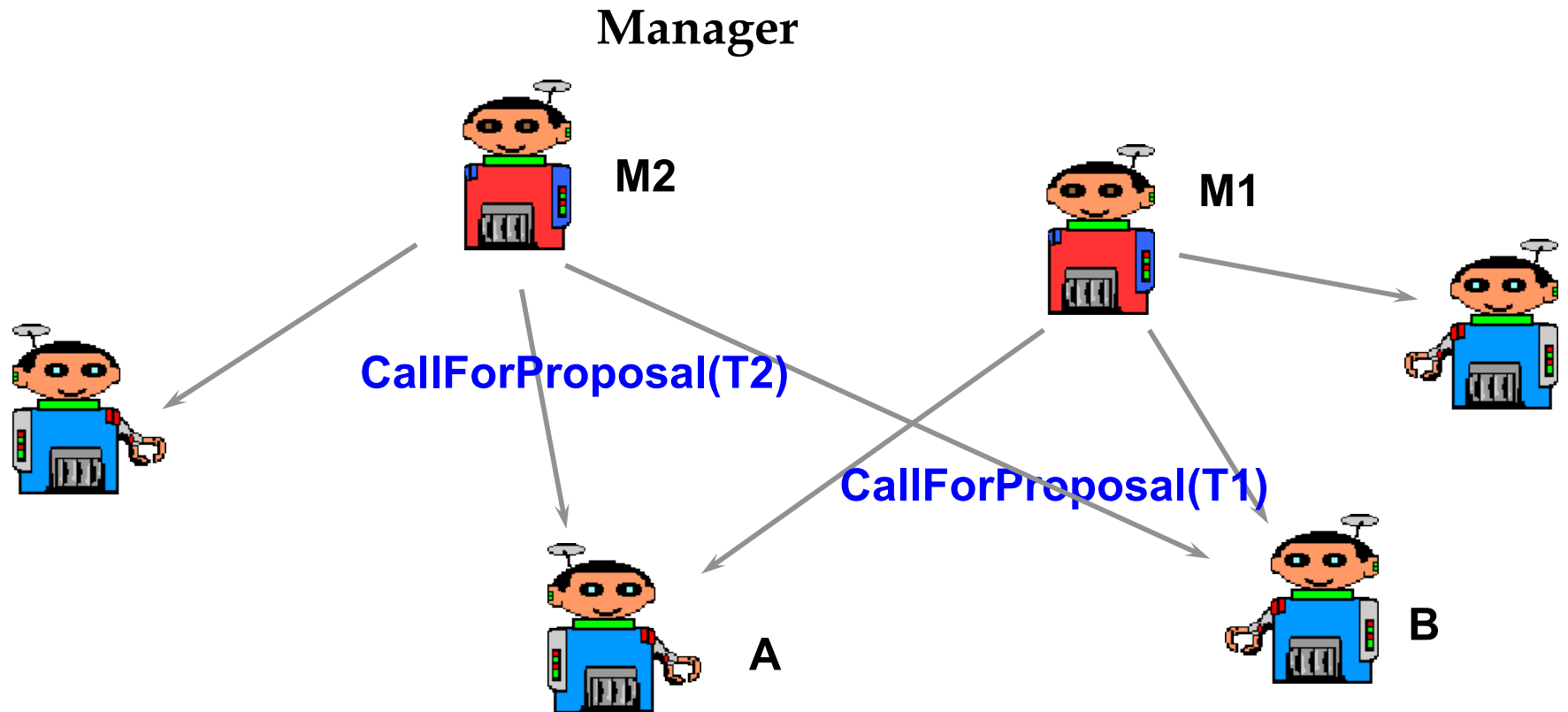
- ◆ **Que se passe-t-il quand plusieurs managers font des propositions?**
- ◆ **Au moment de l'appel d'offre (réception de plusieurs offres en parallèles**
- ◆ **Au moment de l'exécution, si le manager a besoin de plusieurs agents pour accomplir sa tâche.**

Réponses aux appels d'offre



L'agent A répond-il à T2 si T1 ou non?

Deadlock si le manager a besoin de plusieurs exécutants



Si A s'engage auprès de M1 et B auprès de M2, impossible d'accomplir La tâche...

Engagement des agents

◆ **Très fort engagement**

- Les agents ne répondent pas à un appel d'offre dès qu'ils ont fait une proposition.

◆ **Fort engagement (aussi appelé early commitment)**

- Les agents peuvent répondre à un autre cfp, s'ils ont fait une proposition.
- Mais ils ne peuvent pas répondre à un cfp s'ils sont déjà dans une tâche

◆ **Faible engagement**

- Tout est possible..