Paradigmes de la concurrence

Application concurrente

Architecture logique

- Une application est découpée en un ensemble de processus séquentiels, s'exécutant en parallèle et capables de communiquer entre eux.
- Le comportement opérationnel de l'application est exprimé par le comportement concurrentiel des processus

Application concurrente

Objectifs

- Exprimer un comportement correct des processus en utilisant des mécanismes de **synchronisation**

- Présenter un ensemble de problèmes de base et leurs solutions

Modèle de processus

- Processus **séquentiel**

Programme en exécution sur un processeur virtuel (multiprocesseurs, partage d'un processeur)

- Processus **concurrents**

Exécution en parallèle

Processus **indépendant** si son exécution n'interagit pas avec l'exécution d'autres processus du système

Processus **coopérant** si son exécution dépend de l'exécution d'autres processus (partage de données)

Modèle de processus

• Processus **coopérants**

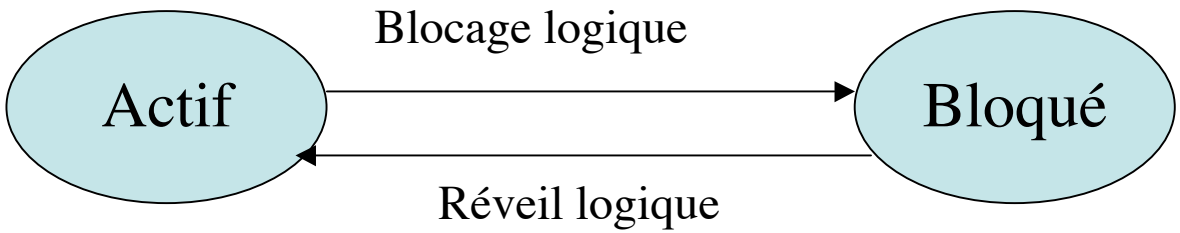
Unités de découpage des activités d'une application

Intérêt :

- Partage d'information
- Augmentation de la vitesse d'exécution (si multiprocesseurs)
- Modularité

Modèle de processus

États de synchronisation d'un processus séquentiel



Événements programmés au niveau de l'architecture logique

Modèle de processus

Propriétés

- Le comportement des processus est asynchrone
- Les processus peuvent partager des données
- Un processus est vu comme une thread

Paradigmes de la concurrence

- Modélisation de classes de problèmes fréquemment rencontrés

- Solution à un paradigme constitue un composant dans la conception de système ou d'application concurrente

Paradigmes de la concurrence

Principaux paradigmes

- **l'exclusion mutuelle** accès cohérent à des ressources partagées
- **la cohorte** coexistence d'un groupe de taille maximale
- **le passage de témoin** coopération par découpage de tâches entre les processus
- **les producteurs consommateurs** communication par un canal fiable
- **les lecteurs-rédacteurs** compétition cohérente
- **le repas des philosophes**

allocation de plusieurs ressources et interblocage

Exclusion mutuelle

Exemple 1 Partage d'une imprimante par 2 processus P et Q qui produisent des suites de lignes

```
processus_P
imprimer('je suis');//P1
imprimer('le processus P');//P2
```

```
processus_Q
imprimer('je suis');//Q1
imprimer('le processus Q');//Q2
```

[P1P2Q1Q2]	[P1Q1P2Q2]
je suis	je suis
le processus P	je suis
je suis	le processus P
le processus Q	le processus Q
OK	Incohérent

Exemple 2 partage d'une variable

```
entier compte=0 //variable partagée
```

```
processus_P
   entier x = 0;
//  variable locale
   pour j de 1 à 16 faire
        x = compte;   //P1
        x = x +1 ;    //P2
        compte = x ;  //P3
   fait
```

```
processus_Q
   entier x = 0;
//  variable locale
   pour j de 1 à 16 faire
        x = compte;   //Q1
        x = x +1 ;    //Q2
        compte = x ;  //Q3
   fait
```

Résultat attendu compte=32

{compte=0} [P1 P2 P3]^{16} [Q1 Q2 Q3]^{16} {compte=32}

{compte=0} [P1 P2 P3Q1 Q2 Q3]^{16} {compte=32}

{compte=0} [P1Q1P2P3Q2Q3]^{16} {compte=16} !

Exclusion mutuelle

Terminologie

- **ressource critique** ressource partagée dont l'utilisation n'est faite que par un processus à la fois
- **exclusion mutuelle** condition de fonctionnement garantissant l'accès exclusif à la ressource
- **section critique** phase d'utilisation d'une ressource critique par un processus

Une section critique par ressource critique

Exclusion mutuelle

Accès exclusif à une ressource

Hypothèses de fonctionnement

H1 Les vitesses relatives des processus sont quelconques
H2 Les priorités ou les droits des processus sont quelconques
H3 tout processus sort au bout d'un temps fini de sa section critique

Exclusion mutuelle

Spécification de comportement

- C1 Un processus au plus en section critique
- C2 Pas d'interblocage actif ou passif

Si aucun processus n'est en section critique et que plusieurs demandent à y entrer, alors un demandeur doit entrer en section critique au bout d'un temps fini

- C3 Un processus bloqué hors de sa section critique ne doit pas empêcher un autre d'y entrer
- C4 La solution doit être symétrique

Exclusion mutuelle

Autres propriétés souhaitables

P1 Équité pas de famine de processus

P2 Respect des priorités des processus, des échéances

P1 et P2 contradictoires!

Remarque : l'exécution d'une section critique peut être vue comme une action atomique soit entièrement, soit pas du tout

Exclusion mutuelle

Schéma d'exécution des processus

```
Processus
tant que vrai faire
     hors-SC
     entrée-SC  // contrôle de l'accès de P
     SC         // utilisation
     sortie-SC  // P libère l'accès
fait
```

Remarque : on considère pour plus de généralité des processus cycliques

La cohorte

Coopération d'un groupe de processus

•Exemples

Un service multiprogrammé avec N serveurs au plus

Partage d'une ressource à N points d'accès

Partage de N ressources banalisées : mémoire paginée

•Hypothèse

tout processus quitte au bout d'un temps fini la cohorte

Remarque si N=1 on retrouve l'exclusion mutuelle

La cohorte

- Hypothèse complémentaire

tout processus quitte au bout d'un temps fini la cohorte

Remarque si N=1 on retrouve l'exclusion mutuelle

- Spécification de comportement

N processus au plus, N fixé coopèrent de manière asynchrone pour

-se répartir une tâche ou un service

-partager N ressources banalisées

La cohorte

Schéma d'exécution

```
 processus P
tant que vrai faire
  hors-groupe
  entrée-groupe//contrôle l'accès au groupe
  groupe //participation au groupe
  sortie-groupe  // libère un accès
fait
```

Passage de témoin

Coopération par division du travail entre les processus : envoi d'un signal, fin d'une action

Schéma d'exécution(exemple)

```
Processus
 tant que vrai faire
  attente-signal// attend le droit d'accès
  suite d'actions
  envoi-signal// envoie le droit d'accès
 fait
```

Producteurs-Consommateurs

Coopération entre deux types de processus les producteurs et les consommateurs via un tampon ou une voie de communication avec un nombre maximal de messages

Producteurs-Consommateurs

Spécification

Contexte commun `tampon de N cases`

```
producteur                    consommateur
 tant que vrai faire           tant que vrai faire
 produire un message;          retirer un message;
 déposer un message;           consommer le message;
 fait                          fait
```

Objectif

Asservir la vitesse moyenne de production à la vitesse moyenne de consommation en ralentissant le moins possible le producteur

Producteurs-Consommateurs

Hypothèses

1- vitesses relatives quelconques
2- tampon de taille fixe, 1 case=1message, vide initialement
3- tout message est déposé et retiré une fois et une seule
4- le dépôt et le retrait se font en un temps fini

Producteurs-Consommateurs

Propriétés de la solution

1- exclusion mutuelle d'accès aux messages
2- le producteur attend si le tampon est plein, il est réveillé dès que le tampon n'est plus plein
3- le consommateur attend si le tampon est vide, il est réveillé dès que le tampon n'est plus vide
4- pas d'interblocage

Producteurs-Consommateurs

Schéma d'exécution

Contexte commun `tampon de N cases vide;`

```
entier nbmess=0;
//nombre de messages dans tampon
producteur
  tant que vrai faire
   produire un message;
   si nbmess==N alors attendre;
   déposer un message;
   nbmess++;
   réveil éventuel d'un consommateur;
  fait
```

Producteurs-Consommateurs

Schéma d'exécution

```
consommateur
tantque vrai faire
   si nbmess==0 alors attendre;
   retirer un message;
   nbmess--;réveil éventuel du producteur;
   consommer le message;
fait
```

Lecteurs-Rédacteurs

Compétition d'accès à un ensemble de données par un ensemble de processus

- lecteurs accès seulement en lecture
- rédacteurs accès en lecture et écriture

Objectif

Garantir la cohérence des données

Spécification

- plusieurs lectures simultanément
- les écritures sont en exclusion mutuelle

Lecteurs-Rédacteurs

Hypothèse complémentaire

tout processus termine sa lecture ou son écriture au bout d'un temps fini

Spécification de comportement

- Les lectures sont concurrentes et cohérentes, les écritures sont en exclusion mutuelle
- Priorité aux lecteurs, équité entre lecteurs et rédacteurs, service à l'ancienneté

Lecteurs-Rédacteurs

Schéma d'exécution

```
lecteur
  tant que vrai faire
   début-lecture;//contrôle l'accès en lecture
    lecture;
   fin-lecture;//libère un accès en lecture ou
                écriture
  fait;
rédacteur
  tant que vrai faire
   début-écriture;//contrôle accès écriture
   écriture;
    fin-écriture;//libère accès en lecture ou
                  écriture
   fait;
```

Le repas des philosophes

Partage de plusieurs classes de ressources entre des processus concurrents asynchrones

Hypothèses de fonctionnement

- Méthodes d'allocation : globale, à la demande, une à une
- tout processus libère ses ressources au bout d'un temps fini
- Interblocage possible
- Famine possible

Le repas des philosophes

Hypothèses

• chaque assiette (philosophe) a une place fixe

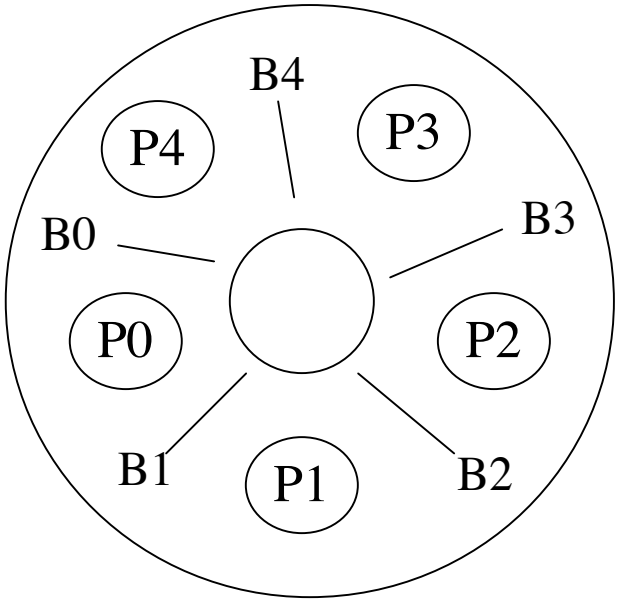
• chaque baguette a une place fixe

• accès à chaque baguette en exclusion mutuelle

Propriétés

• pas d'interblocage

• pas de famine



Le repas des philosophes

Une première solution prendre les baguettes une par une, la sienne puis celle de droite

```
Philosophe i // i identifiant 0..n-1
   tant que vrai faire
   penser;
   prendre la baguette i;
   prendre  la baguette(i+1)%n;
   manger;
   rendre la baguette i;
   rendre la baguette(i+1)%n;
   fait;              }
```

Le repas des philosophes

Problème

Tous les philosophes peuvent prendre leur baguette gauche au même moment et se bloquer lorsque la baguette droite n'est pas accessible.

Il y a alors **interblocage**

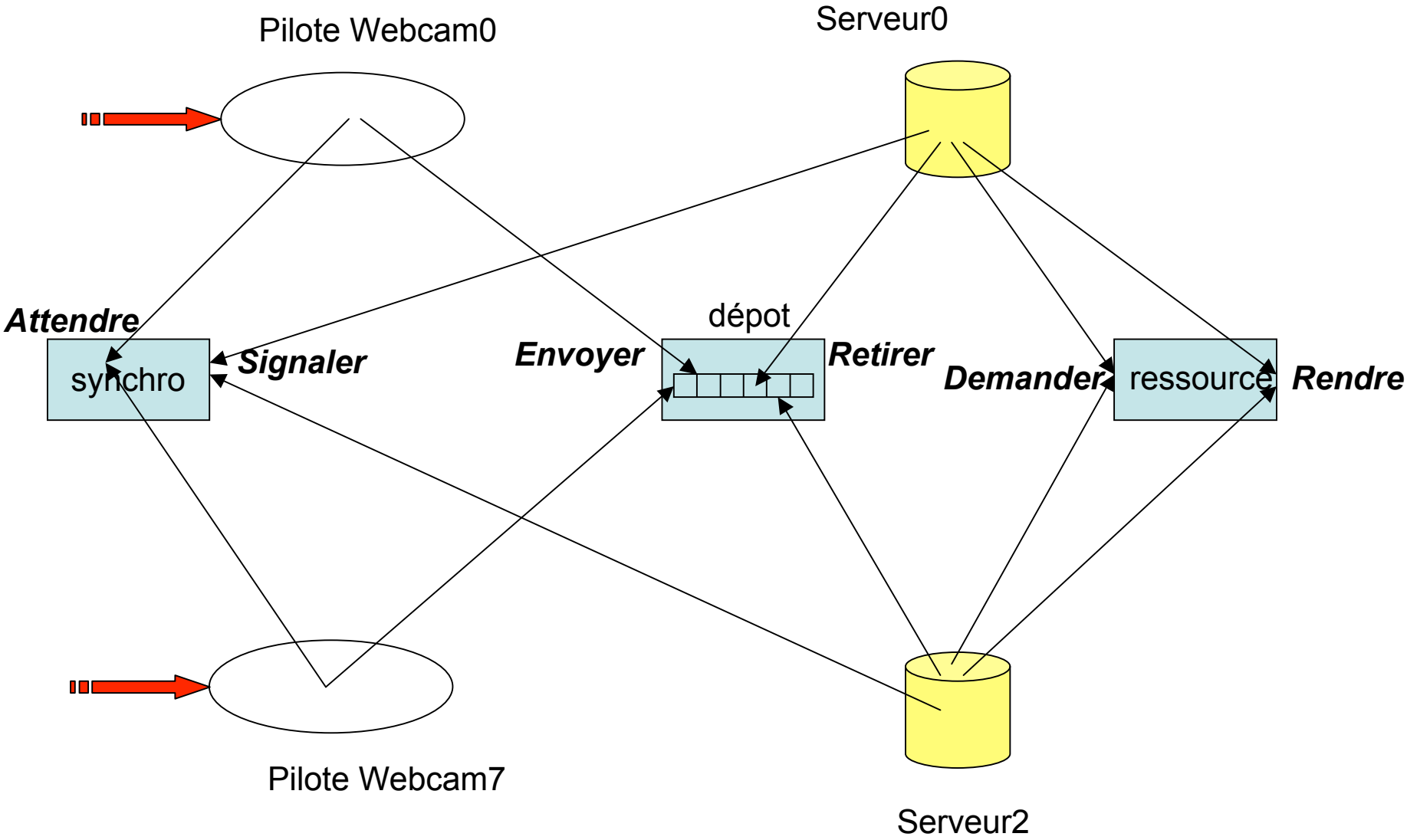
Idée : allocation globale

Un philosophe prend les deux baguettes si elles sont disponibles, sinon il attend

Pas d'interblocage, mais risque de **famine**

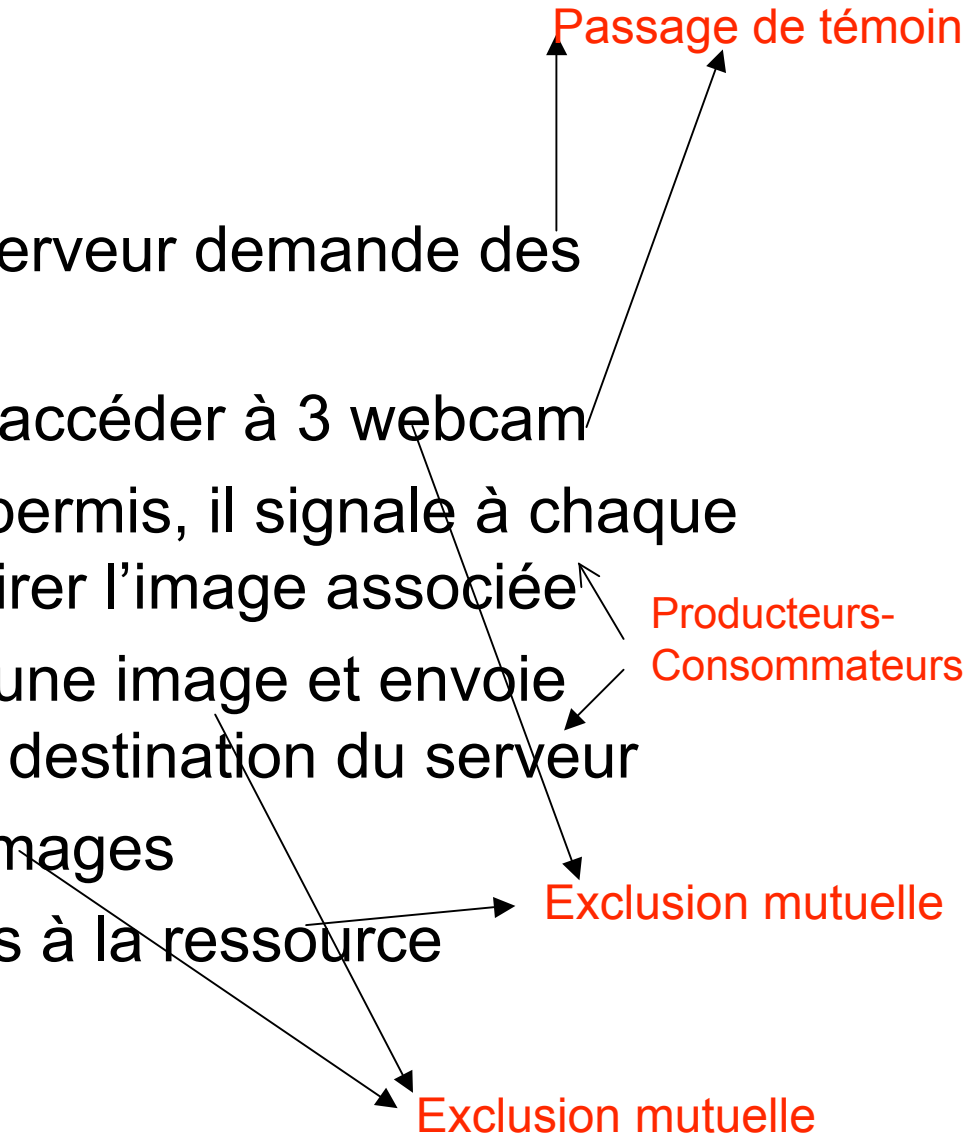
Etude de cas 1 : surveillance par Webcam

- Les activités :
  - 8 Pilotes qui gèrent 8 webcam
  - 3 Serveurs qui récupèrent les images : chaque serveur gère 3 webcam.
- La communication :
  - Chaque pilote dépose une image dans un dépôt commun
  - Chaque serveur demande à être autorisé à retirer les 3 images du dépôt
  - Le serveur autorisé retire les images.



- Les pilotes sont des clients
- Les serveurs sont… des serveurs

- La synchronisation :
  - Le pilote attend qu'un serveur demande des images
  - Un serveur demande d'accéder à 3 webcam
  - Lorsque l'accès lui est permis, il signale à chaque pilote qu'il est prêt à retirer l'image associée
  - Chaque pilote acquiert une image et envoie l'image dans le dépôt à destination du serveur
  - Le serveur retire les 3 images
  - Le serveur libère l'accès à la ressource



Algorithme

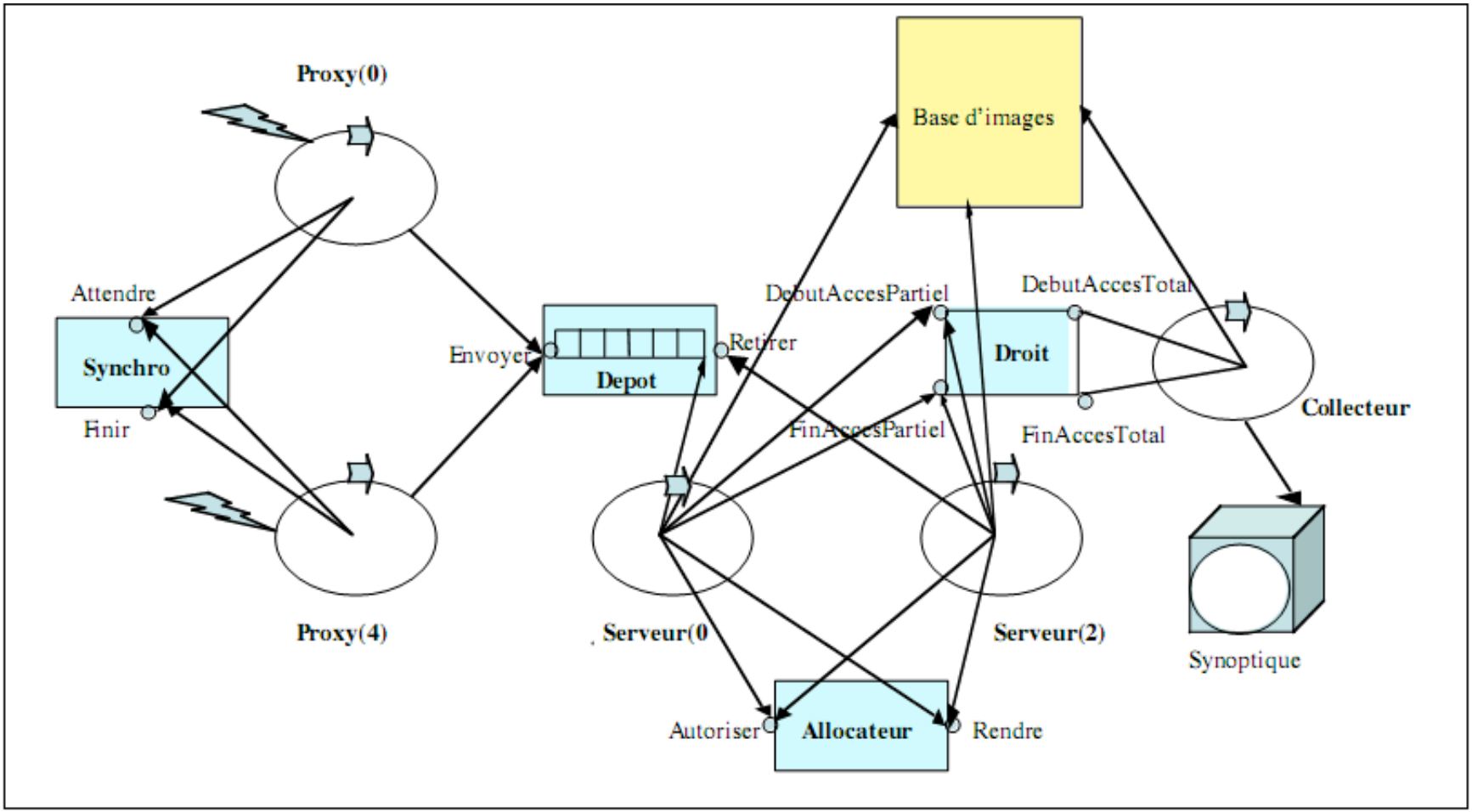
Un Client

```
 tant que vrai faire
  synchro.Attendre (X) ;
 // attente d'un signal envoyé à X par un serveur
    Lecture_de_la_Camera (Z) ;
 // acquisition de l'image bimap
    ….
    Préparation de la donnée avant rangement
     ….
    depot.Envoyer (X,Y)
 // envoi vers les serveurs de l'image Y prise par la
 Webcam X
 fait
```

Un Serveur Algorithme

```
Tant que vrai faire
 // le client donne 3 noms de Webcam
negociation_avec_le_client (A,B,C) ;
 // réservation des Webcams
ressource.Demander(A) ; ressource.Demander(B) ;
ressource.Demander(C) ;
 // réveil des pilotes
synchro.Signaler(A) ; synchro.Signaler(B) ;
synchro.Signaler(C) ;
 // retrait de l'image de A et copie dans Y
depot.Retirer(A,Y) ;
 // retrait de l'image de B et copie dans Z
depot.Retirer(B,Z) ;
 // retrait de l'image de C et copie dans T
depot.Retirer(C,T) ;
 // libération des Webcams
ressource.Rendre(A) ; ressource.Rendre(B) ;
ressource.Rendre(C) ;
fait
```

Etude de cas N°2 : surveillance d'usines



Etude de cas N°2 : surveillance d'usines

- Les activités
  - 5 usines produisent un rapport d'activités
  - 3 serveurs récupèrent ces rapports
  - 1 collecteur les affiche
- La communication
  - Un rapport est transféré de chaque usine dans un serveur qui le transforme en une image stockée dans une base d'images
  - Périodiquement le collecteur affiche le contenu de la base d'images

- Un proxy associé à une usine produit des données
- Un serveur retire une donnée, construit une image et demande l'accès à la base à l'emplacement réservé pour l'usine
- Le collecteur demande le droit d'accéder à la base

  - La synchronisation

  - les proxys s'exécutent dans l'ordre 0,1… passage de témoin
    - Le retrait n'est possible que si l'envoi a été fait : le plus ancien rapport est retiré : chaque usine dépose son rapport à son tour
    - Un serveur récupère ce rapport et construit une image qui sera stockée dans la BD ; un seul serveur à la fois peut accéder à la BD
    - Le collecteur interdit l'accès aux serveurs pour lire la BD