2016-2017

Multitâche & objets

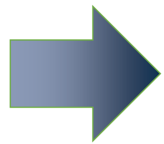
Modélisation objet des paradigmes multitâches

© Shebli Anvar — Multitâches & objets

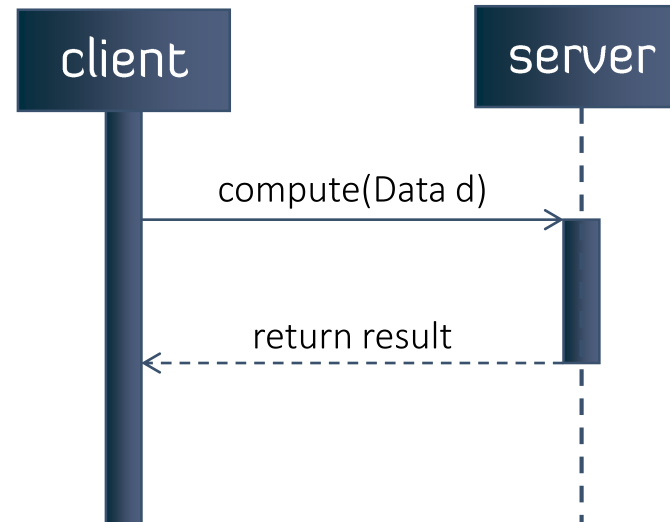
© Shebli Anvar
CEA — Institut de recherches sur les lois fondamentales de l'Univers
Centre de Saclay — 91191 Gif-sur-Yvette — France
shebli.anvar@cea.fr

Communication inter-objets

- Un objet est une instance de classe
- La classe encapsule sa structure interne
- La classe spécifie une interface à base d'opérations



Communication inter-objets
- appels d'opérations
- données échangées = paramètres



© Shebli Anvar — Multitâches & objets

Collaboration entre instances

© Shebli Anvar — Multitâches & objets

Abstraction des classes

Client

* client

1 manager

Manager

+compute()

#chooseServer ()

manager 1

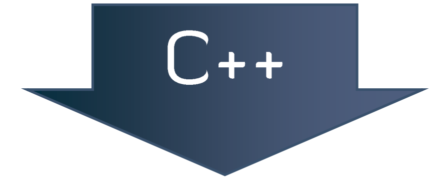
server *

Server

+batch ()

Rôles

Les messages sont traduits en opérations




```cpp
class Client
{
private:
    Manager* manager;
};
```

```cpp
class Manager
{
private:
    std::vector<Server*> server;
public:
    double compute(Data input);
protected:
    int chooseServer();
};
```

```cpp
class Server
{
public:
    double batch(Data input);
};
```

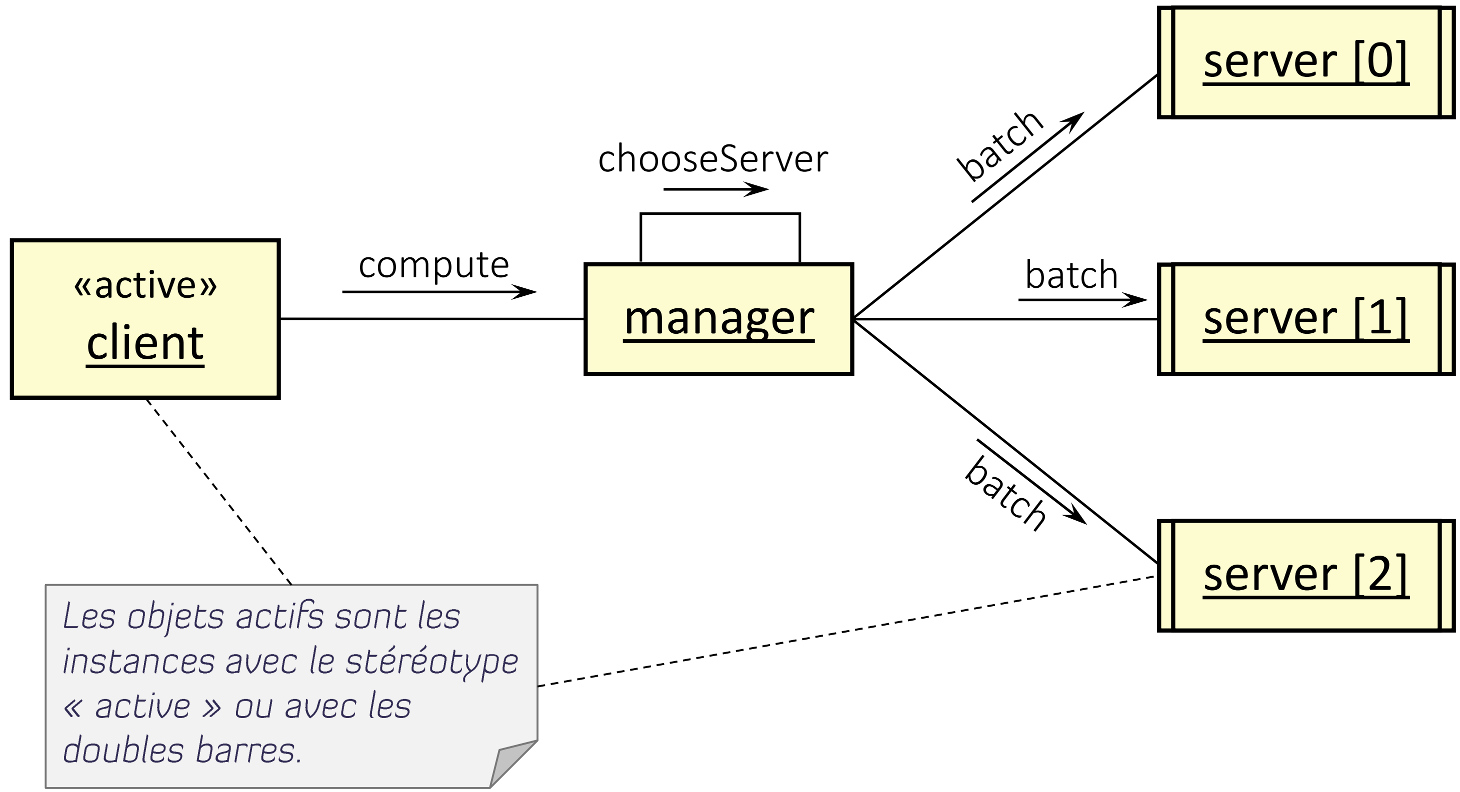
© Shebli Anvar — Multitâches & objets

Objets actifs

2016-2017

© Shebli Anvar — Multitâches & objets

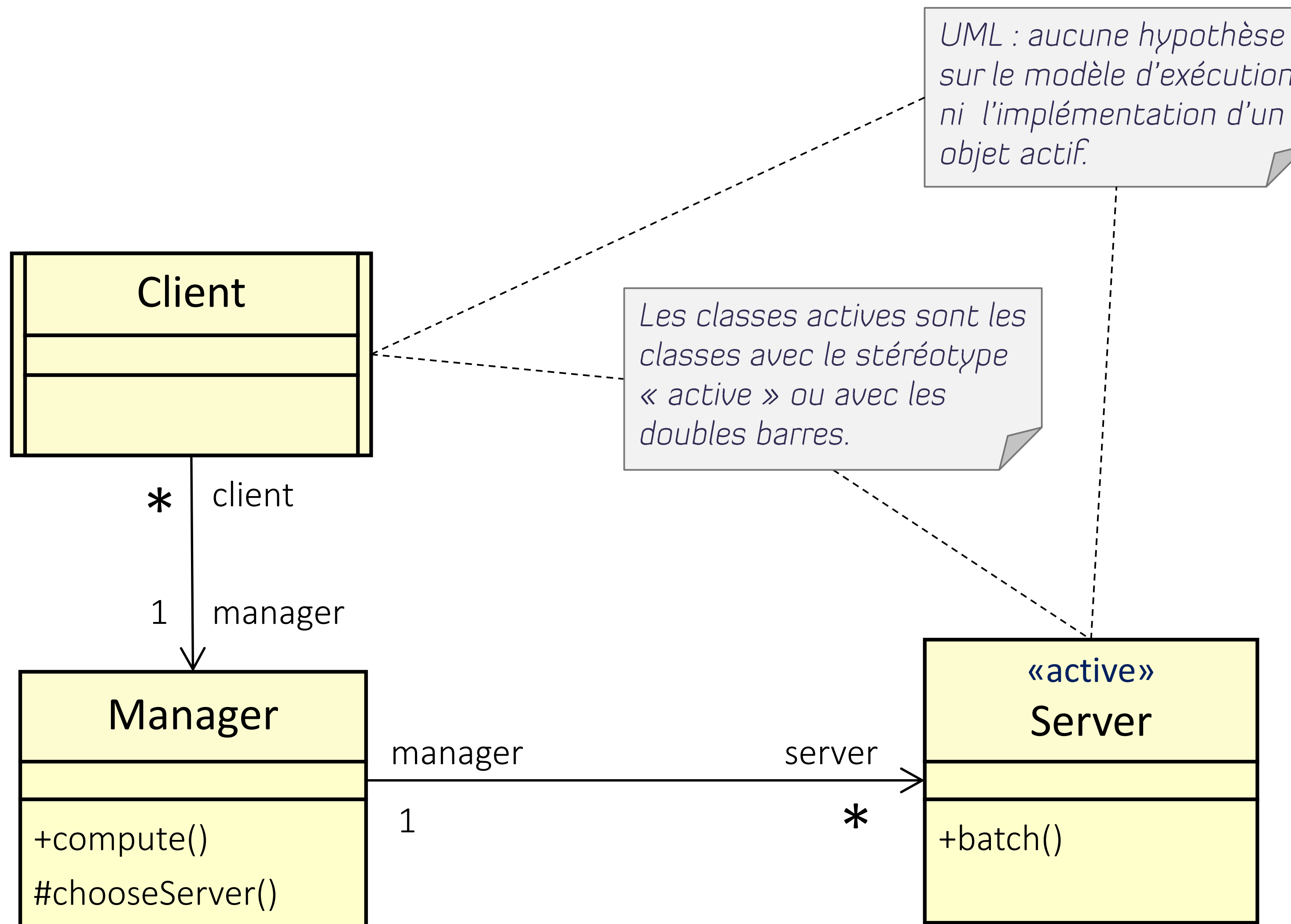
Un « objet actif » est un objet qui encapsule son (ses) propre(s) fil(s) d'exécution



Les objets actifs sont les instances avec le stéréotype « active » ou avec les doubles barres.

Abstraction des classes objets actifs

© Shebli Anvar – Multitâches & objets



Notions à transposer

- Tâche (Thread)
  - Création, lancement
  - Endormissement, suspension
  - Arrêt, destruction
  - Attente d'arrêt (join)
- Mutex
  - Création, destruction
  - Types (simple, récursif…)
  - Prise et rendu de jeton
  - Rendu automatique
- Condition
  - Association avec Mutex
  - Attente et notification
  - Timeout
- Sémaphore
  - Binaire, à compte
  - Conditions initiales
  - Prise et rendu de jetons
  - Timeout
- Communication
  - Asynchrone (file d'attente)
  - Synchronisation différée
  - À distance
  - Broadcast
- Encapsulation
  - Objets thread-safe
  - Objets actifs

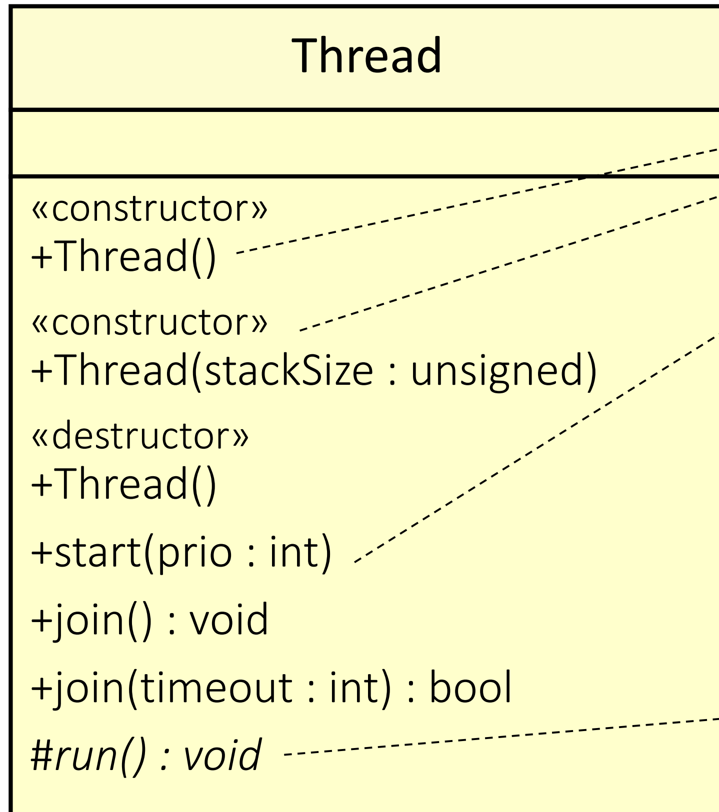
© Shebli Anvar – Multitâches & objets

La notion d'objet actif

- Fusion entre objet et tâche
  - Modèle d'exécution
    - monotâche
    - multitâche (une par opération)
  - Appel d'opération asynchrone
  - Appel d'opération à distance
- En pratique:
  - La classe dérive d'une classe « Thread »
    (ou implémente une interface ad hoc comme l'interface Runnable en Java)
  - Met en œuvre une file d'attente de requêtes d'exécution

© Shebli Anvar — Multitâches & objets

Exemple de classe Thread



On distingue la création du Thread (constructeur) de son exécution (start).

Questions
Pourquoi la méthode Thread::run() doit être :
– protected ?
– virtuelle pure ?

© Shebli Anvar – Multitâches & objets 2016-2017

Les classes Mutex et Lock

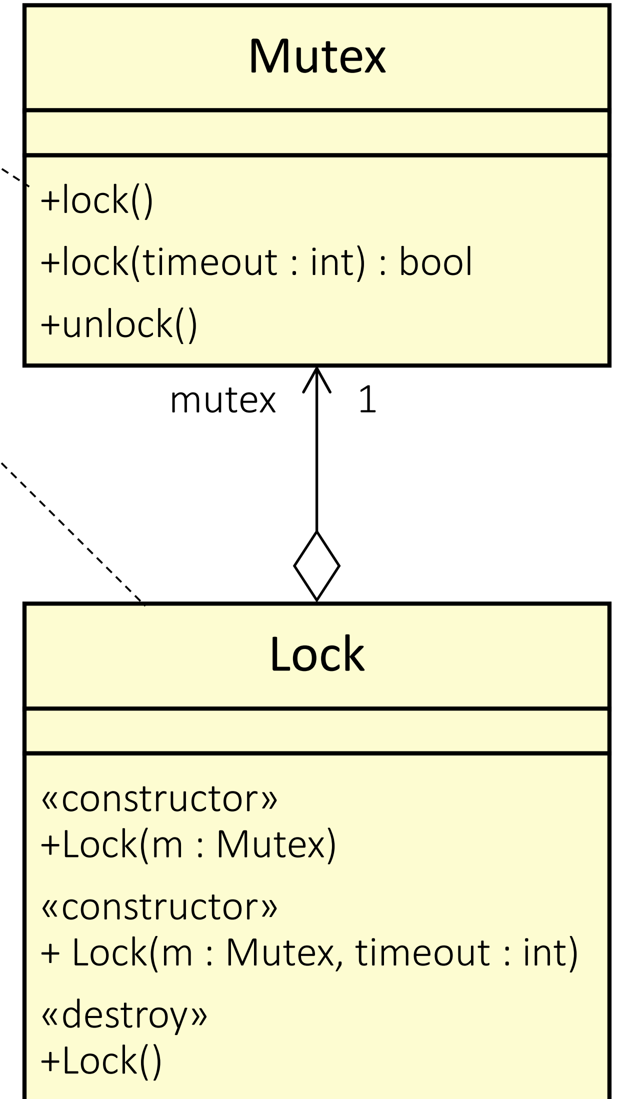
2016-2017

Le problème du rendu de mutex lors des exceptions

```
try
{
      ......
      mutex.lock();
      ......      <-- survenue de l'exception
      mutex.unlock();
      ......
}
catch(std::exception& e)
{
      ......      <-- mutex non rendu
}
```

Appel bloquant si mutex vide

Prise du jeton dans constructeur, rendu dans destructeur



© Shebli Anvar — Multitâches & objets

Les classes Mutex et Lock

© Shebli Anvar – Multitâches & objets 2016-2017

Appel bloquant si mutex vide

Le problème du rendu de jeton lors des exceptions

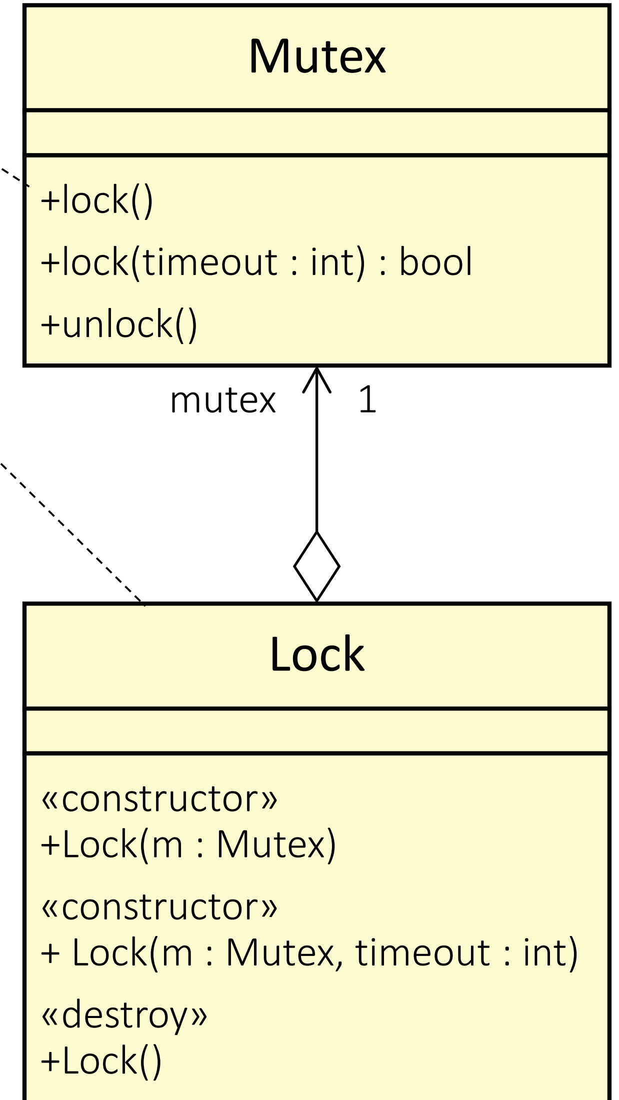
```
try
{
      ......
      Lock lock(mutex);
      ......
      ......
}
catch(std::exception& e)
{
      ......
}
```

survenue de l'exception

rendu exception du mutex ?

rendu normal du mutex ?

Prise du jeton dans constructeur, rendu dans destructeur



Classe Condition

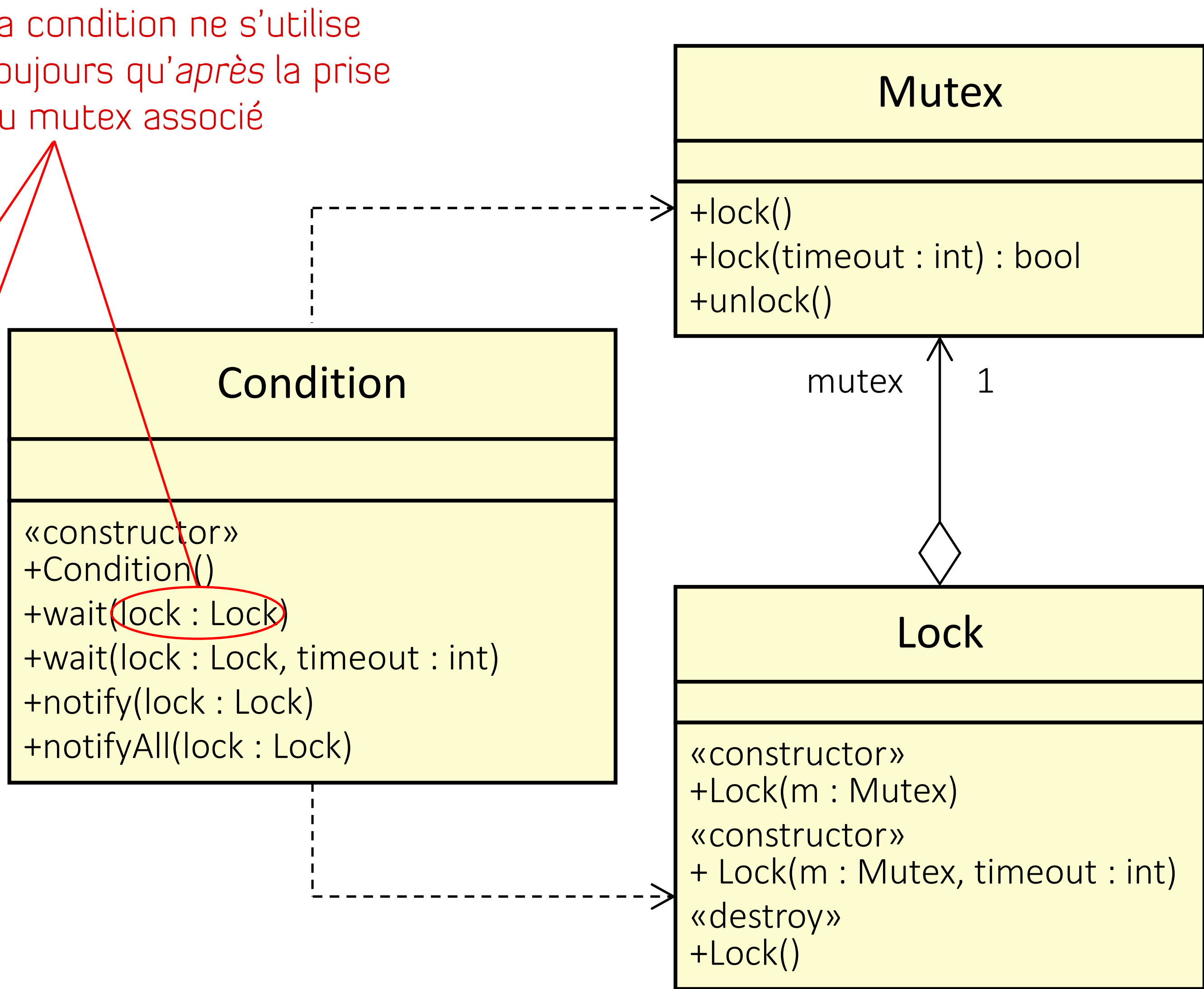
tâche A

```
void waitStop (
     volatile int* pCommand,
     Mutex* mtx,
     Condition* cnd
)
{
        Lock lock(mutex);
        while (*pCommand != STOP)
        {
                cnd->wait(&lock);
        }
}
```

tâche B

```
void doStop (
    volatile int* pCommand,
    Mutex* mtx,
    Condition* cnd
)
{
        Lock lock(mutex);
        *pCommand = STOP;
        cnd->notify((&lock);
}
```

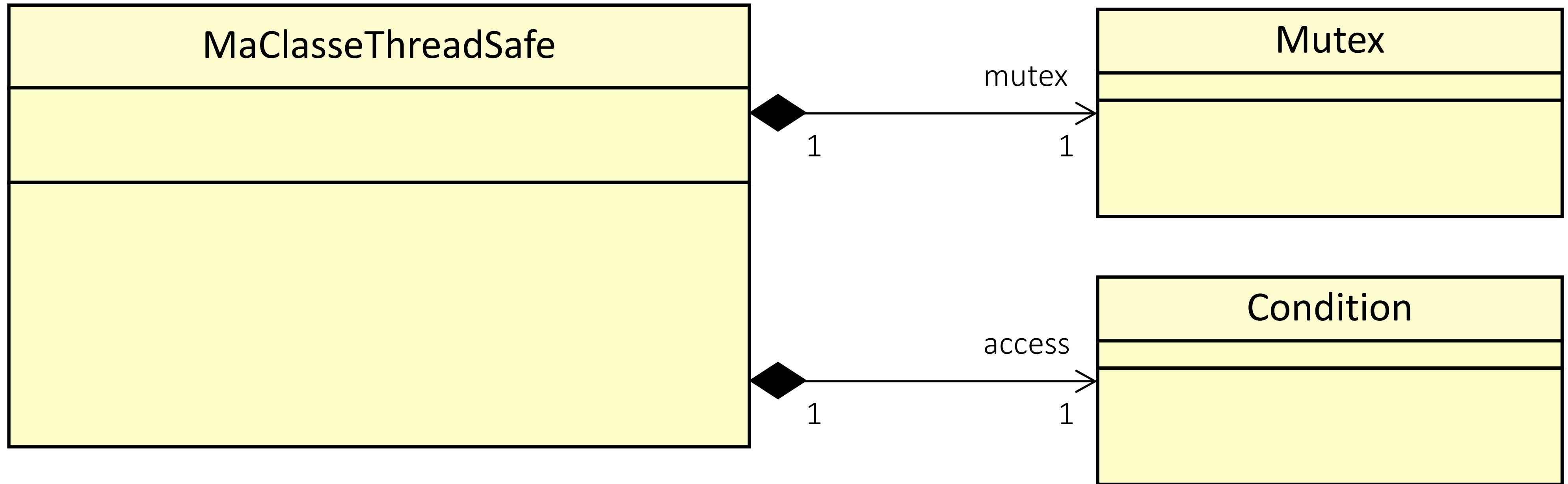
La condition ne s'utilise toujours qu'*après* la prise du mutex associé



© Shebli Anvar – Multitâches & objets

Objet thread-safe

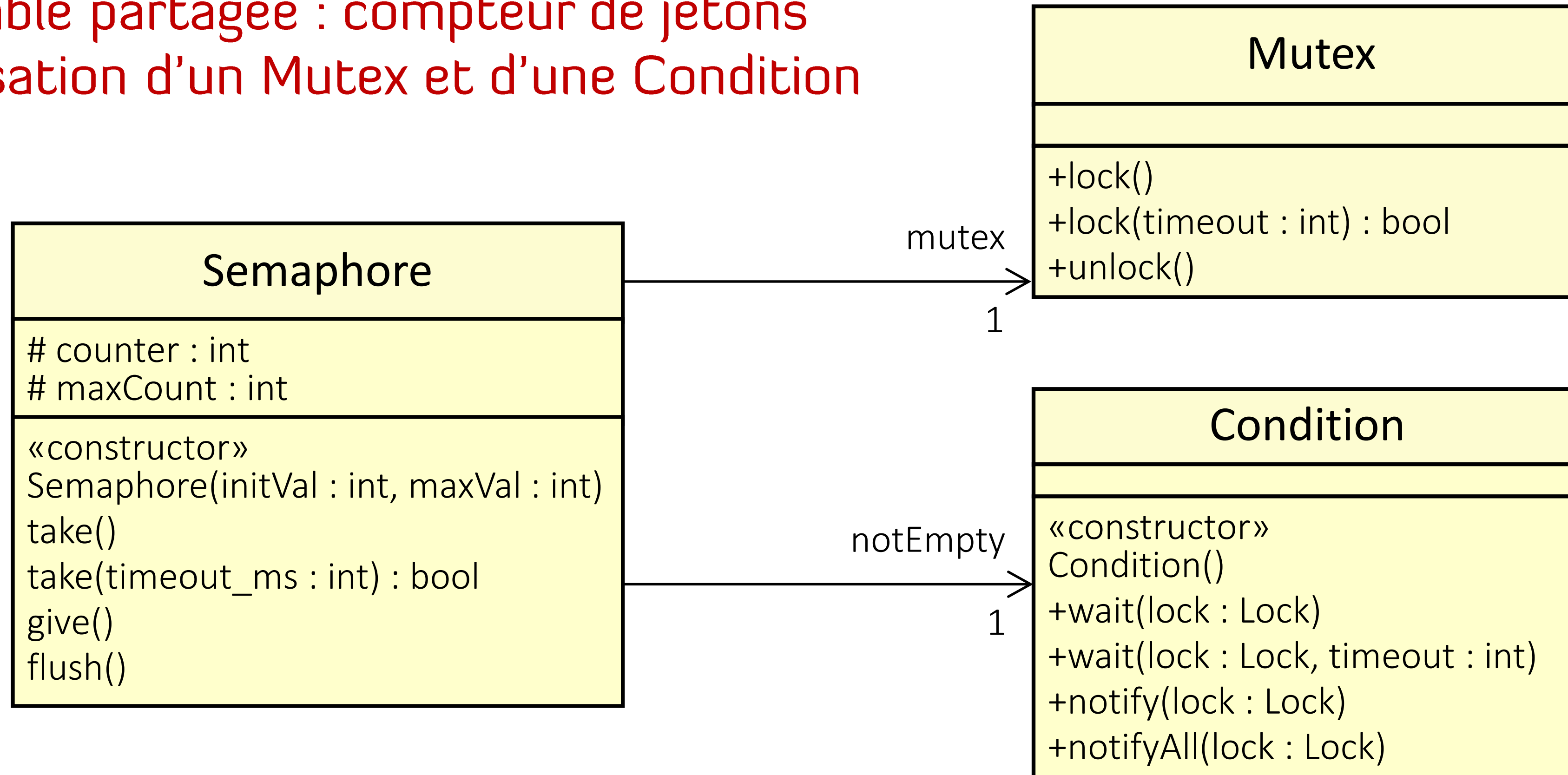
- Toutes les opérations d'une classe thread-safe doivent accéder ou modifier l'état de l'objet en garantissant les appels concurrents.



© Shebli Anvar – Multitâches & objets

Classe Semaphore

- Un sémaphore est un compteur de jetons
  Lorsqu'il est vide, la demande de jeton bloque la tâche
  Déblocage : une autre tâche fournit un jeton
- Mécanisme de blocage déblocage ?
  - Variable partagée : compteur de jetons
  - Utilisation d'un Mutex et d'une Condition



© Shebli Anvar – Multitâches & objets

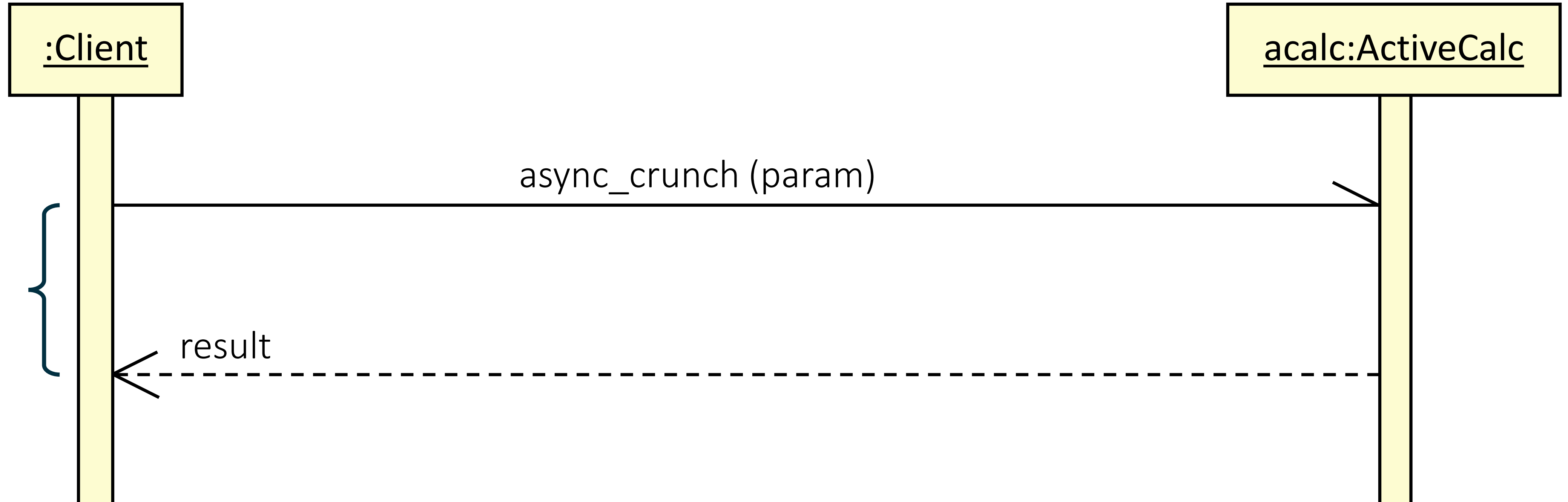
Objet passif : appel d'opération synchrone


```
Client::run(Calculator* calc)
{
        int param = 10;


        double result = calc->crunch(param);
}
```

© Shebli Anvar — Multitâches & objets

Objet actif : appel d'opération asynchrone




```
Client::run(Calculator* calc)
{
        int param = 10;
        Request* req = acalc->async_crunch(param);
        // ... Autres instructions
        double result = req->waitReturn();
}
```

© Shebli Anvar — Multitâches & objets

Objet actif : décomposition de l’appel asynchrone

© Shebli Anvar — Multitâches & objets

Objet actif : décomposition de l'appel asynchrone


```
void Client::main(ActiveCalculator* acalc)
{
    CrunchReq* req = acalc->async_crunch(10); // requête
    // ...................................... // Autres instructions
    double result = req->waitReturn();        // Attente result
}
```

```
CrunchReq* ActiveCalc::async_crunch(double param)
{
    CrunchReq* req = new CrunchReq(param);    // Création de la requête d'exécution
    reqFifo.push(req);                        // Envoi de la requête
    return req;                               // Transmission au client
}
```

```
void ActiveCalc::run()
{
    while(true)
    {
        CrunchReq* req = reqFifo.pop();       // Réception de la requête
        req->execute();                       // Exécution de la requête
    }
}
```

```
void CrunchReq::execute()
{
    result = calc->crunch(param);             // exécution effective du calcul
    returnSema.give();                        // notification du sémaphore « fin de calcul »
}
```


© Shebli Anvar — Multitâches & objets

Objet actif : classes principales

Thread
- *#run() : void*

ActiveObject
- #run() : void

RequestFifo
- +push(Request req)
- +pop() : Request

Semaphore
- +give()
- +take()

Request
- *+execute()*
- +waitReturn()

Calculator
- +crunch(param : double) : double

ActiveCalc
- +async_crunch(param : double) : CrunchReq

CrunchReq
- param : double
- result : double
- +execute()
- +waitReturn() : double

1 reqFifo

* reqs

1 returnSema

1 calc

« create »