Projet gestion d'objets dupliqués

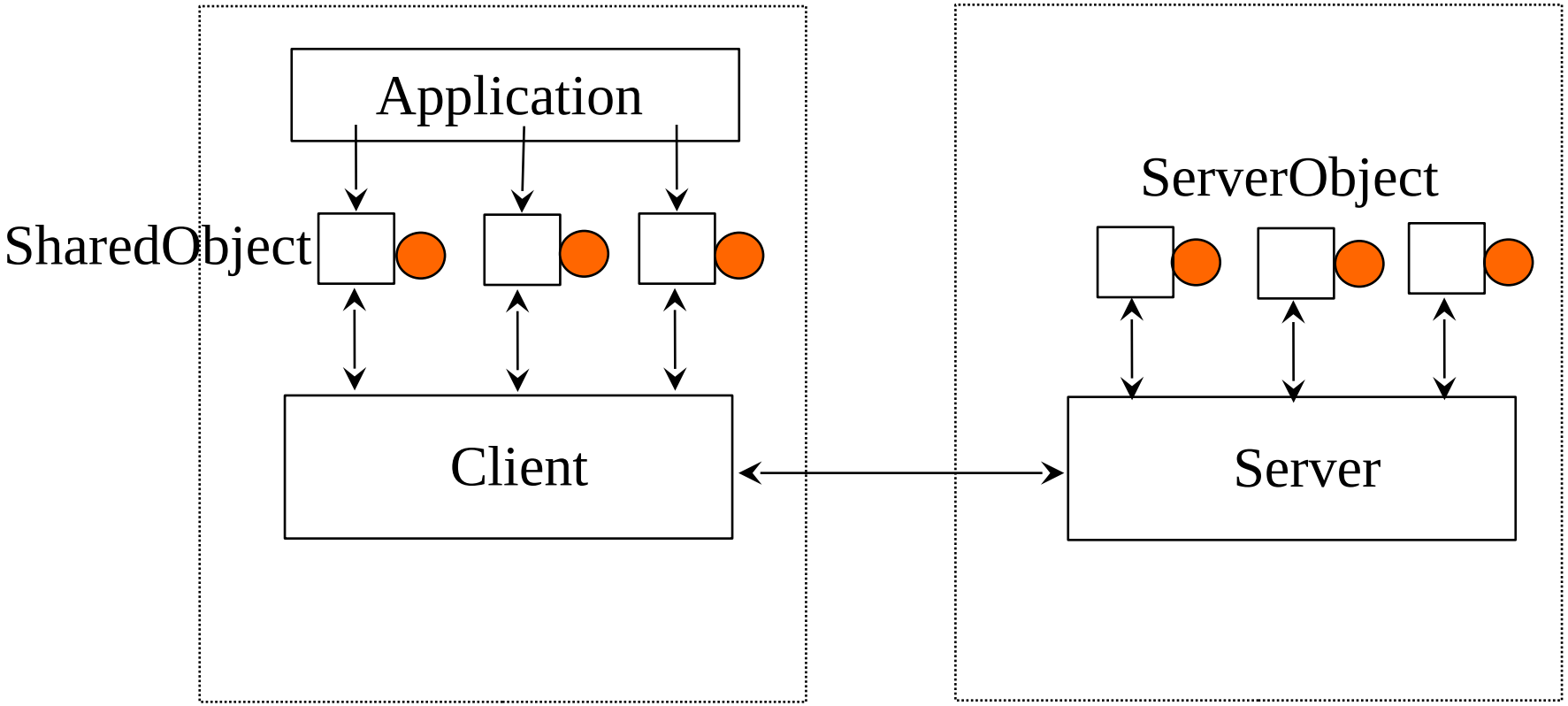
Daniel Hagimont

Daniel.Hagimont@enseeiht.fr

Projet

- Service de gestion d'objets dupliqués
- Mise en cohérence lors de la prise d'un verrou sur un objet
- Pas de verrous imbriqués
- Protocole à invalidation (write-invalidate)
- N lecteurs, 1 écrivain

Architecture



Verrous non retirables + cache

- Un client obtient un verrou en lecture ou écriture : il le conserve jusqu'à ce qu'il décide explicitement de le relâcher
- Verrou lecture/écriture : soit un écrivain seul, soit plusieurs lecteurs sans écrivain
- Verrou en cache : quand un client relâche un verrou :
  - Il n'informe pas le serveur
  - Il pourra (éventuellement) reprendre le verrou sans demander au serveur

Principe

- Quand un client demande un verrou qu'il n'a pas en cache : il le demande au serveur
- Le serveur sait quel(s) client(s) a(ont) le verrou et dans quel mode (lecture ou écriture)
- Le serveur peut réclamer les verrous auprès des clients (callback)
- Le serveur ne préempte pas les verrous d'un client : c'est le client qui choisit quand il relâche le verrou
- Quand le serveur sait que le verrou est libre, il peut l'attribuer au demandeur

SharedObject

- SharedObject : descripteur d'objet
  - une référence à un objet partagé pointe sur un SharedObject
  - champ *obj* pointe sur l'objet effectif
  - contient un *id* unique par rapport au serveur
  - primitives de verrouillage : lock_read(), lock_write() et unlock()
  - utilisation (SharedObject S) :
    - S.lock_read();
    - S.obj.meth(); // avec casting adéquat...
    - S.unlock();

SharedObject

- SharedObject : descripteur d'objet
  - Un attribut lock indique l'état du verrouillage de l'objet sur le site :
    - NL : no local lock
    - RLC : read lock cached (not taken)
    - WLC : write lock cached (not taken)
    - RLT : read lock taken
    - WLT : write lock taken
    - RLT_WLC : read lock taken and write lock cached
  - Si besoin d'interagir avec le serveur, on utilise les fonctions de la couche Client
    - Méthodes statiques
    - Appelées depuis SharedObject

Couche Client

- Méthodes statiques
- Appelées depuis SharedObject
- Appels locaux
  - **static Object lock_read (int id)** : demande d'un verrou en lecture au serveur, en lui passant l'identifiant unique de l'objet (le SharedObject devait être en NL). Retourne l'état de l'objet.
  - **static Object lock_write (int id)** : demande d'un verrou en écriture au serveur, en lui passant l'identifiant unique de l'objet (le SharedObject devait être en NL ou RLC). Retourne l'état de l'objet si le SharedObject était en NL.
- Redirigent les appels vers le serveur
  - Ajoute une référence au client pour rappel

Le serveur

- Reçoit les appels des clients (couche Client)
- Appels a distance (RMI)
  - public Object lock_read(int id, Client_itf c);
  - public Object lock_write(int id, Client_itf c);
- Redirige chaque appel vers le ServerObject associé

ServerObject

- Reçoit les appels du serveur
- Appels locaux
  - public Object lock_read(Client_itf c);
  - public Object lock_write(Client_itf c);

ServerObject

- Mémorise l'état de l'objet partagé
  - Id de l'objet
  - Mode de verrouillage (read/write)
  - Le ou les sites clients (pour les rappeler …)
- Il peut rappeler un site client (interface Client_itf)

Client

- Reçoit les rappels du serveur
- Appels à distance (RMI)
  - void invalidate_reader(int id)
  - Object invalidate_writer(int id)
  - Object reduce_lock(int id)
- Redirige chaque appel vers le SharedObject associé

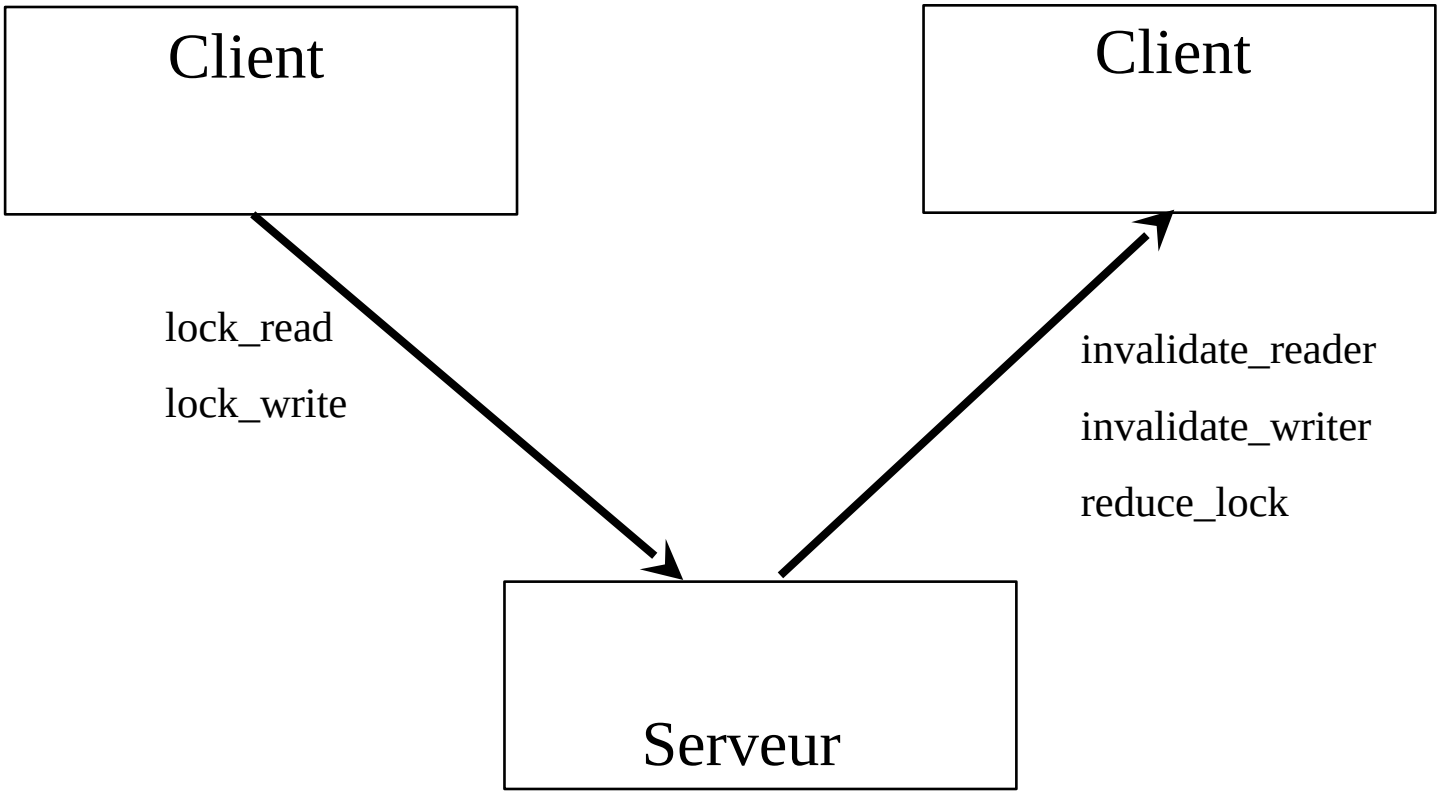
SharedObject

- Reçoit les appels du client
- Appels locaux
  - void invalidate_reader()
  - Object invalidate_writer()
  - Object reduce_lock()

Service de nommage

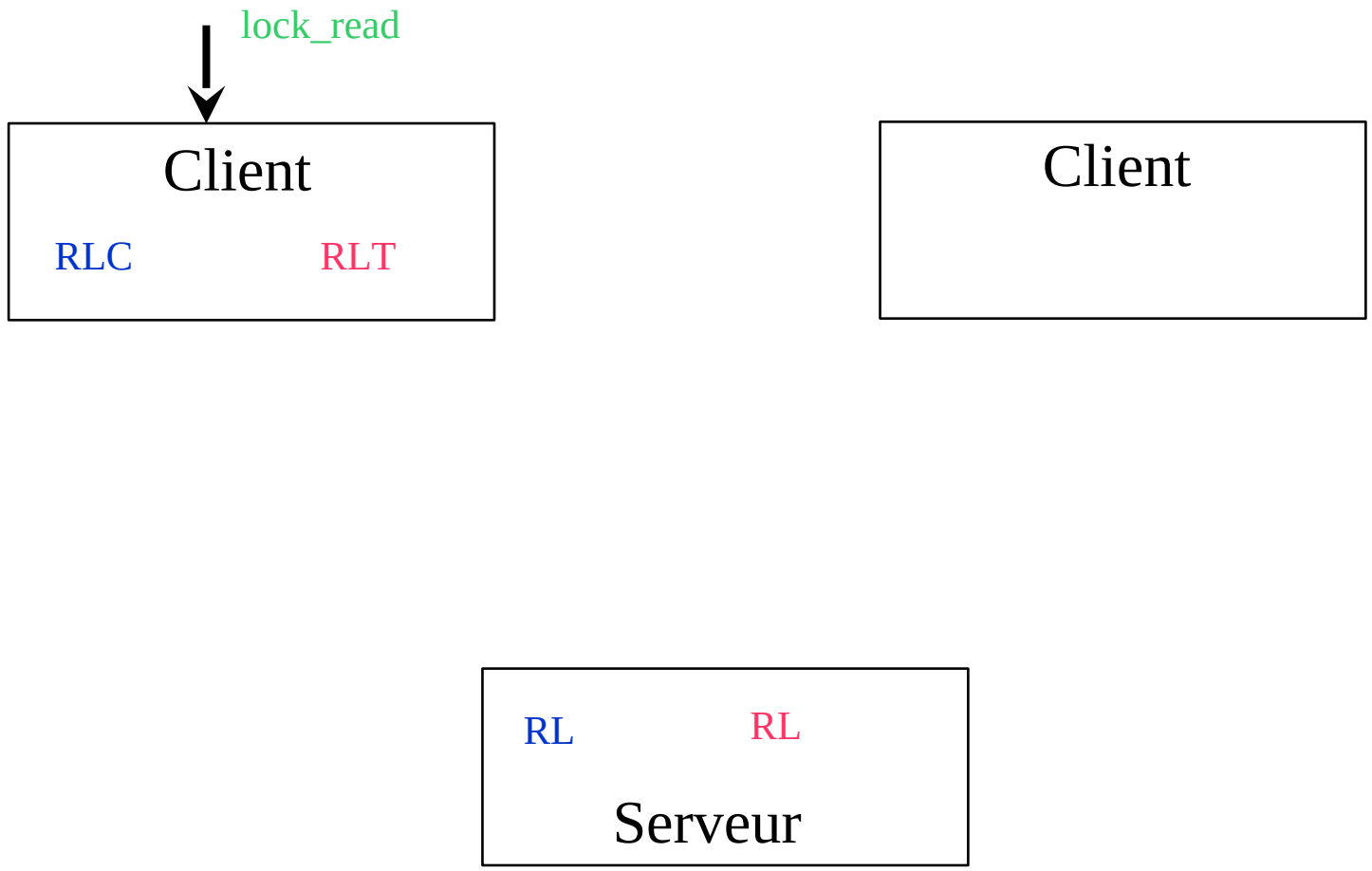
- Appelable à travers la couche Client
- Méthodes statiques
  - **static void init()** : initialise la couche cliente, à appeler au début de l'application.
  - **static SharedObject create (Object o)** : permet de créer un objet partagé (en fait un descripteur) initialisé avec l'objet o. Le descripteur de l'objet partagé est retourné. A la création, l'objet n'est pas verrouillé.
  - **static SharedObject lookup (String n)** : consulte le serveur de nom et retourne l'objet partagé enregistré.
  - **static void register (String n, SharedObject so)** : enregistre un objet partagé dans le serveur de noms.
- Doit être implanté coté serveur pour être partagé

Vue globale

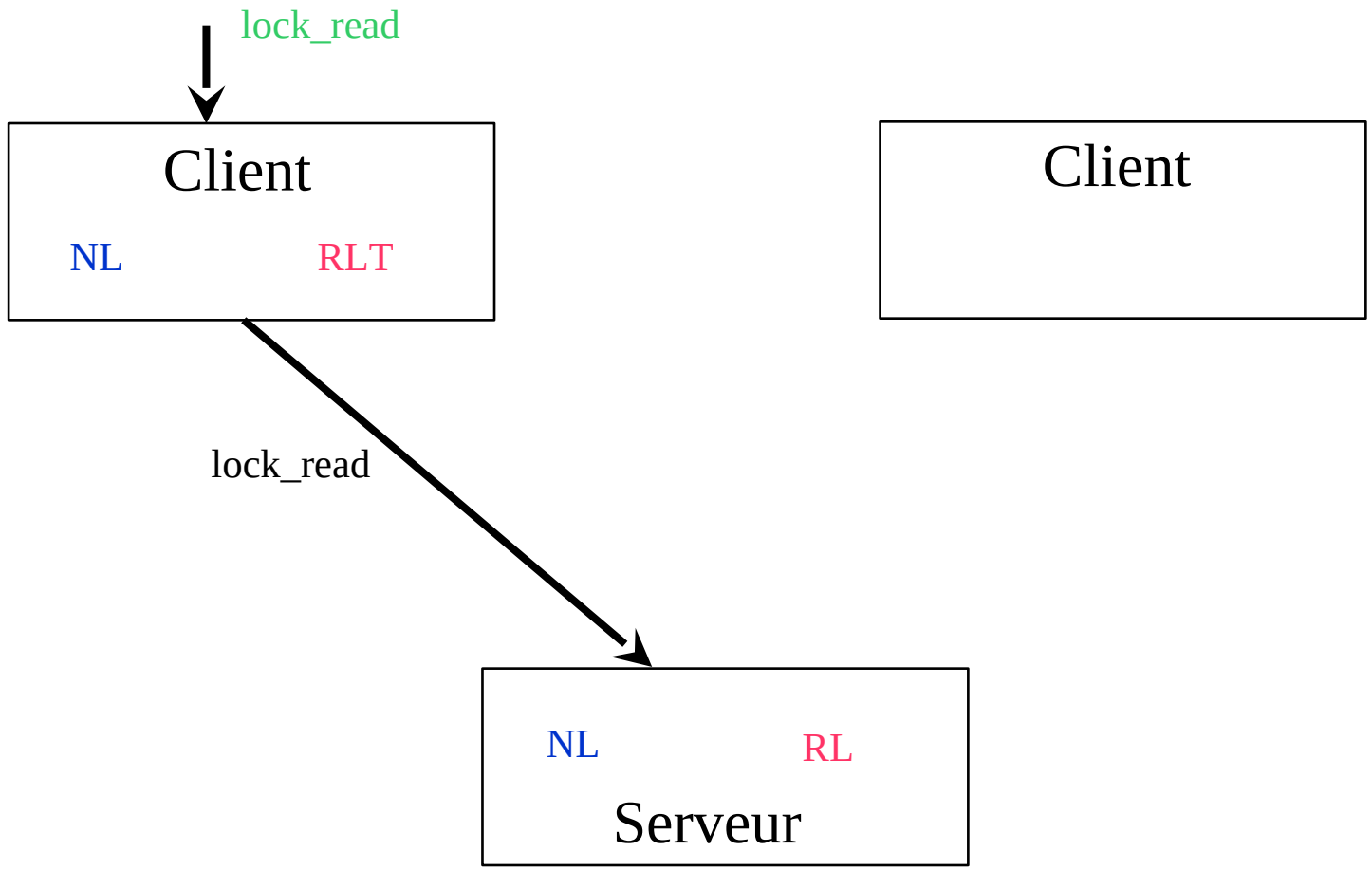

Client
Client
lock_read
lock_write
invalidate_reader
invalidate_writer
reduce_lock
Serveur

Cas de figure 1

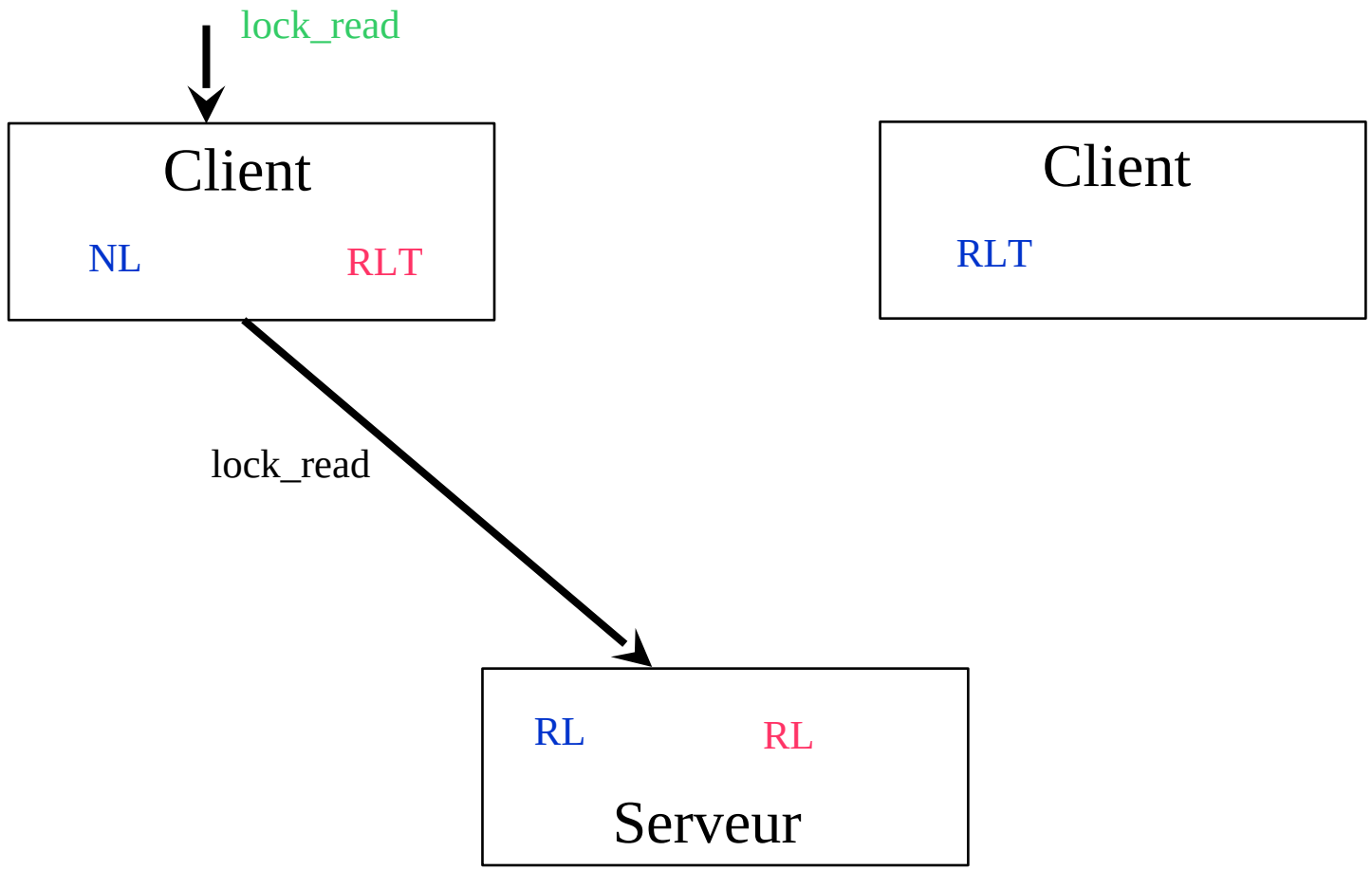
Etat avant	Etat après



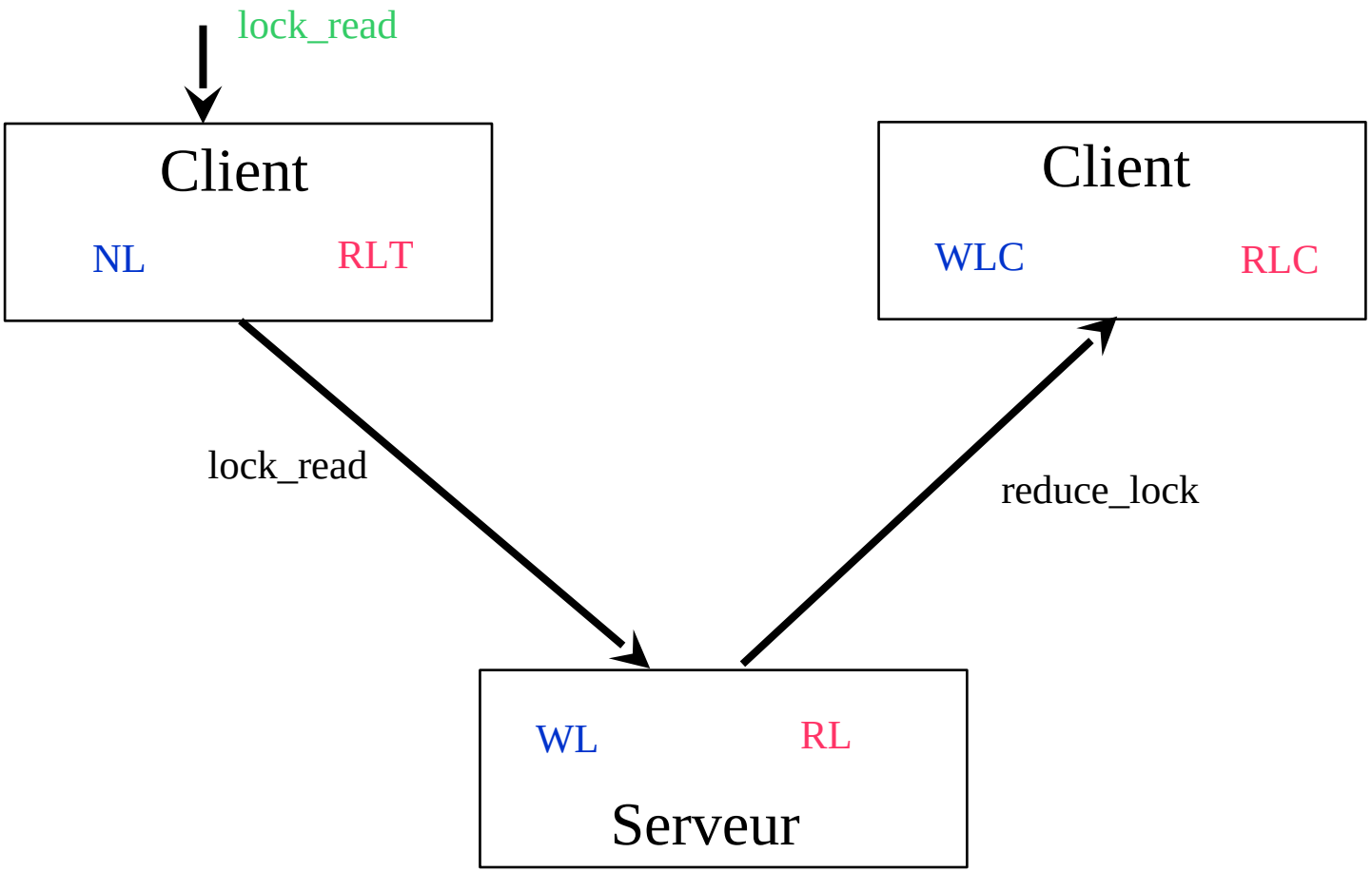
Cas de figure 2



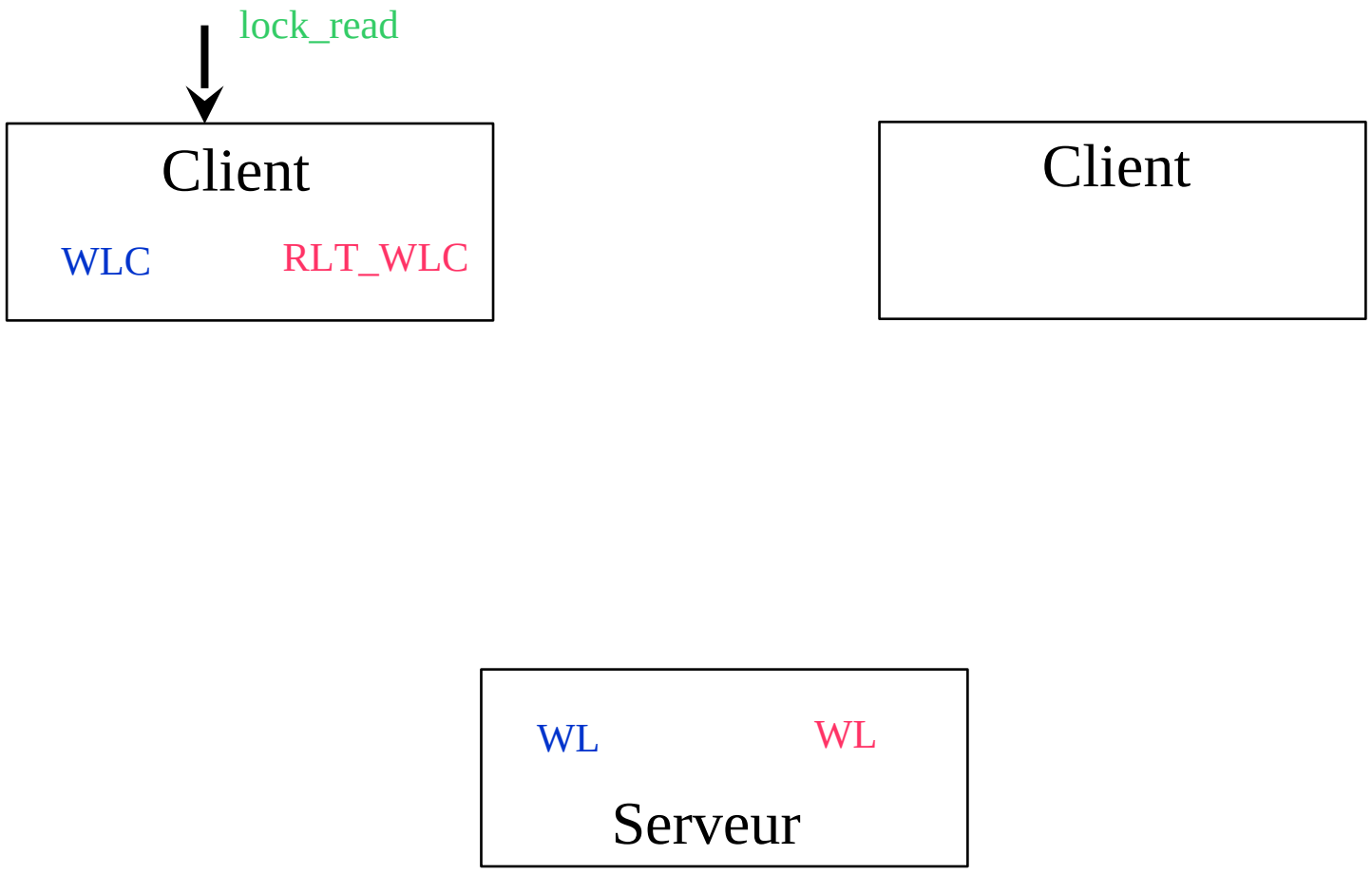
Cas de figure 3



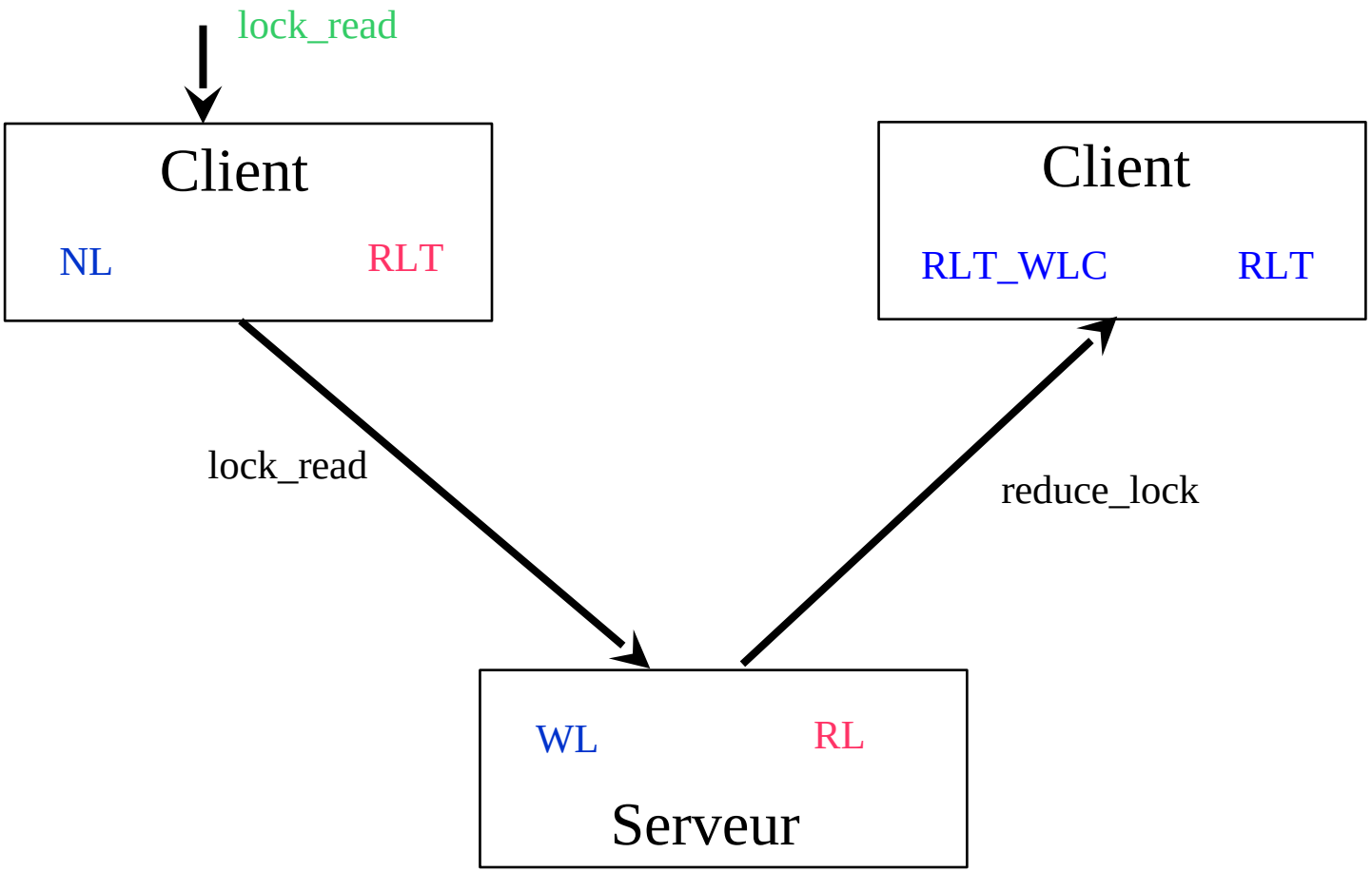
Cas de figure 4



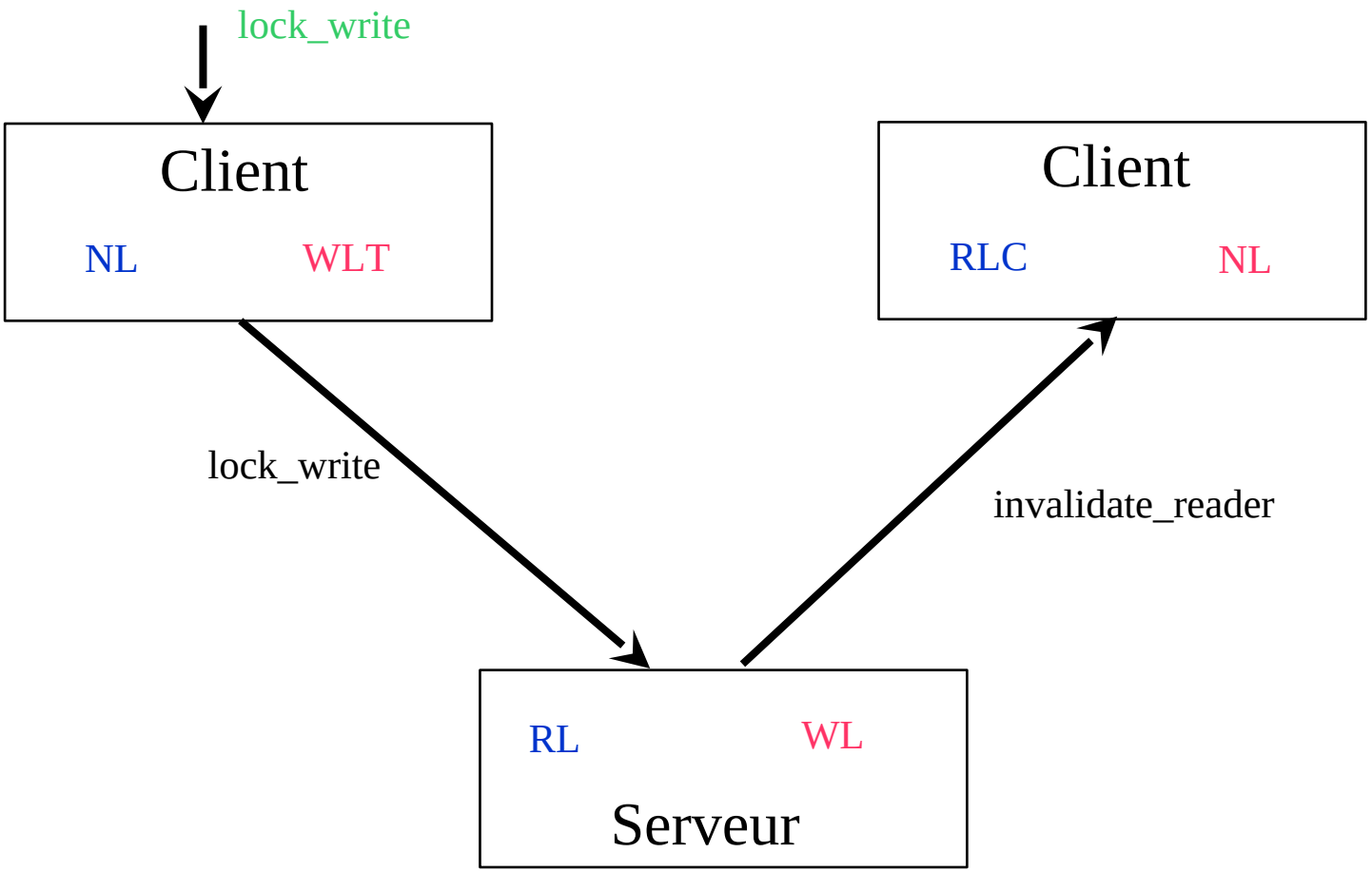
Cas de figure 5



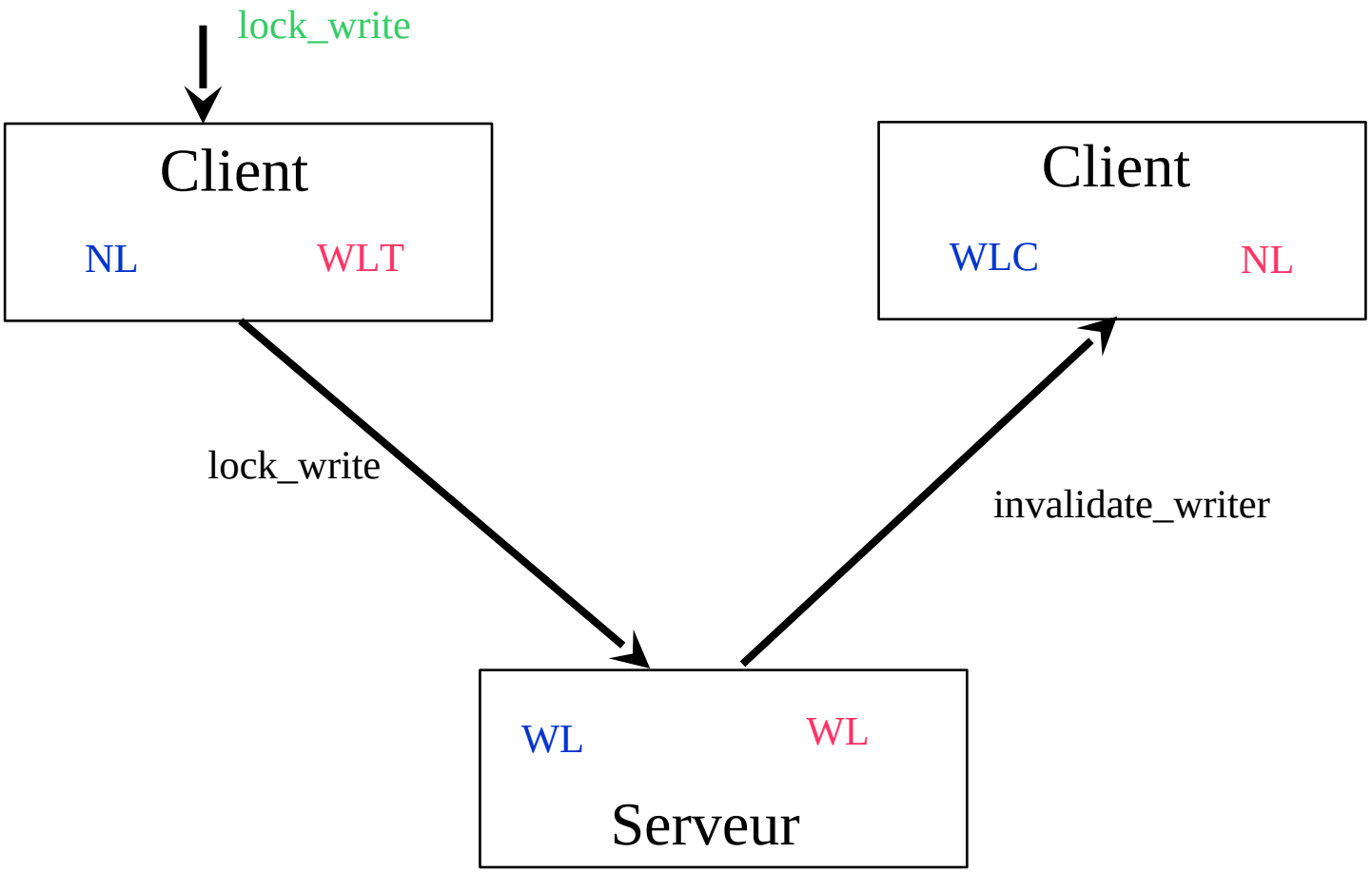
Cas de figure 6



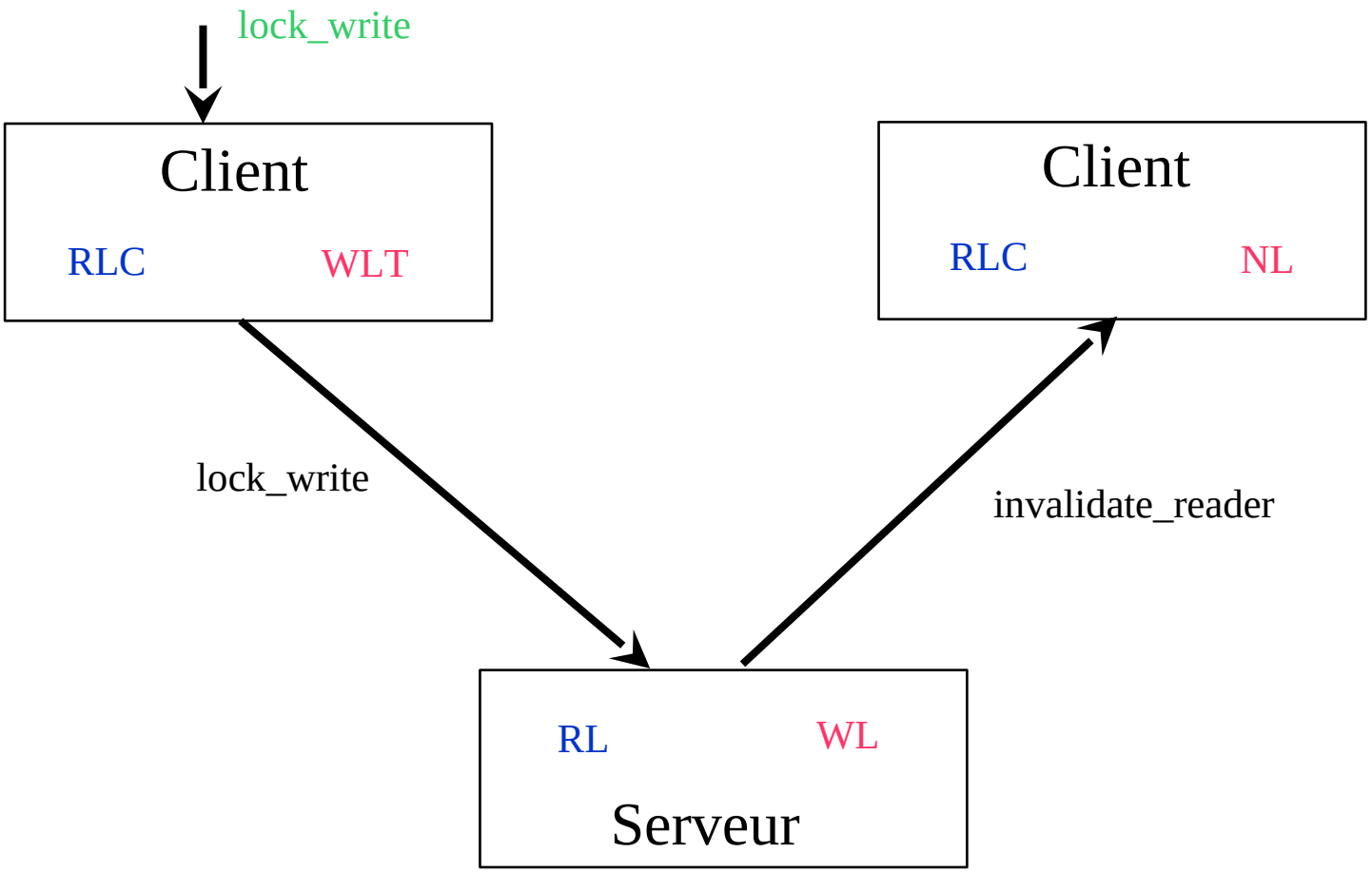
Cas de figure 7



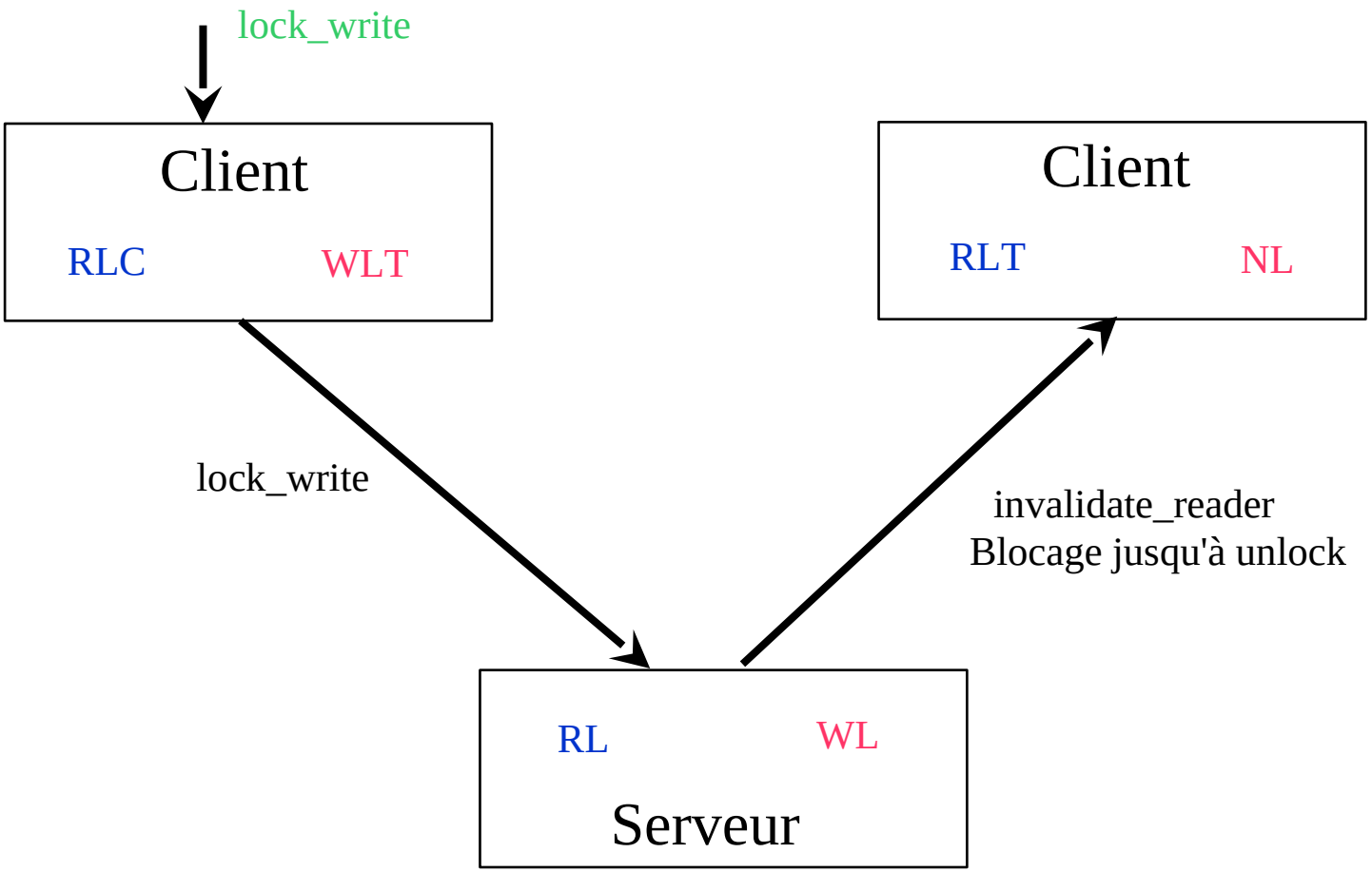
Cas de figure 8



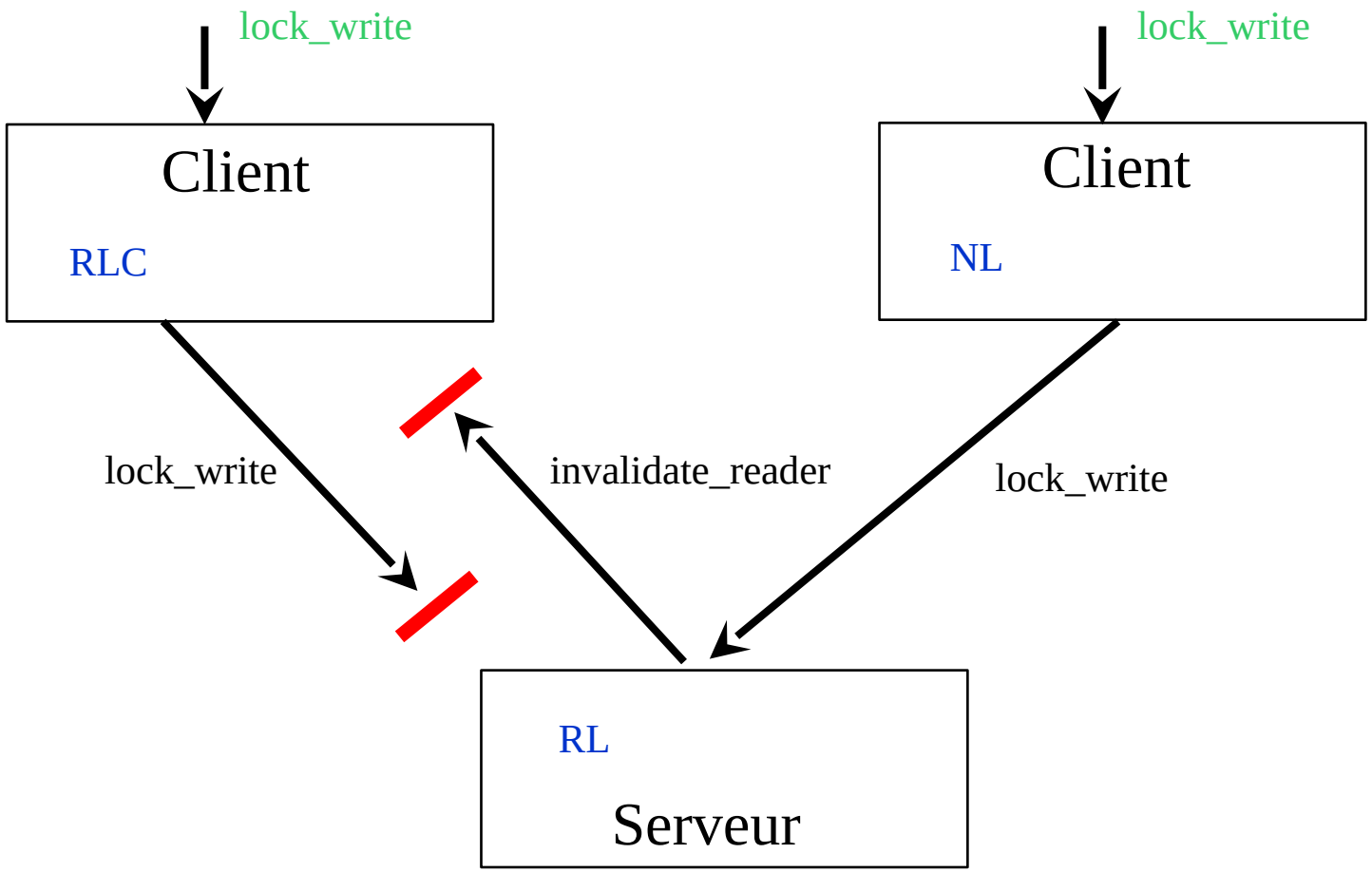
Cas de figure 9



Cas de figure 10

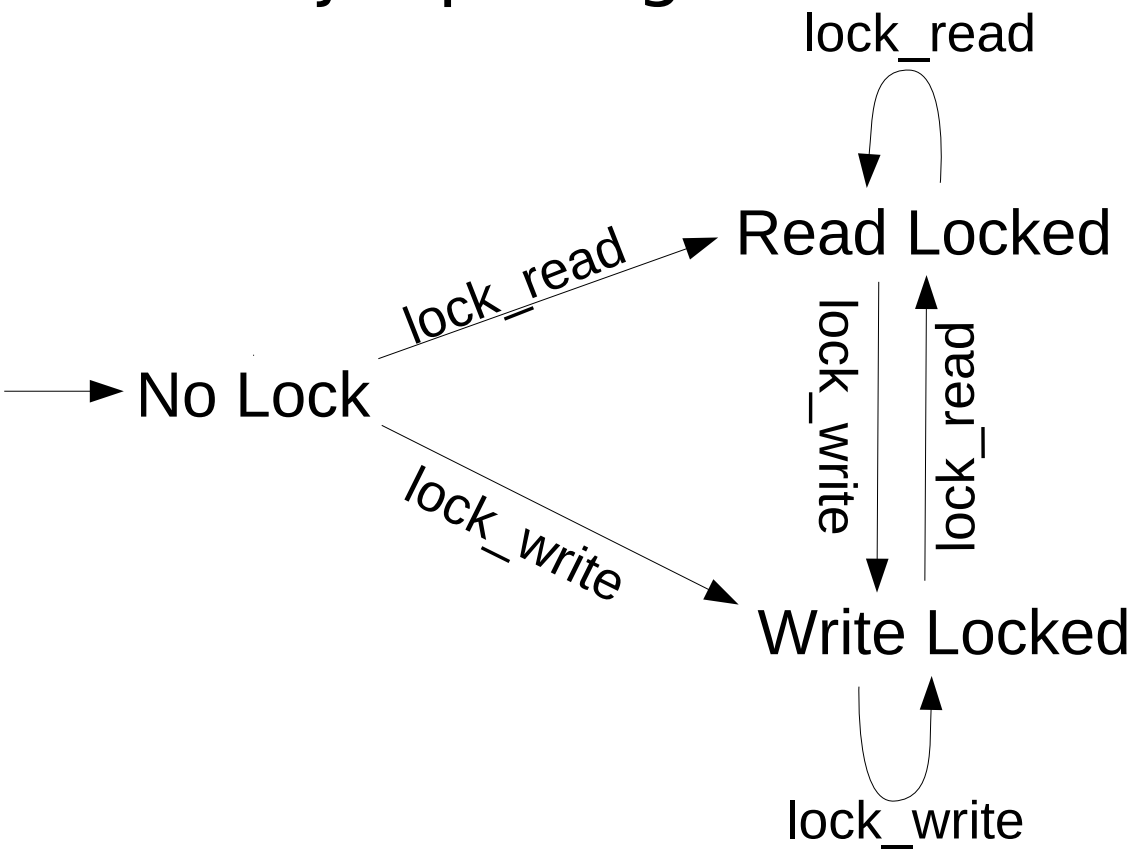


Cas de figure 11



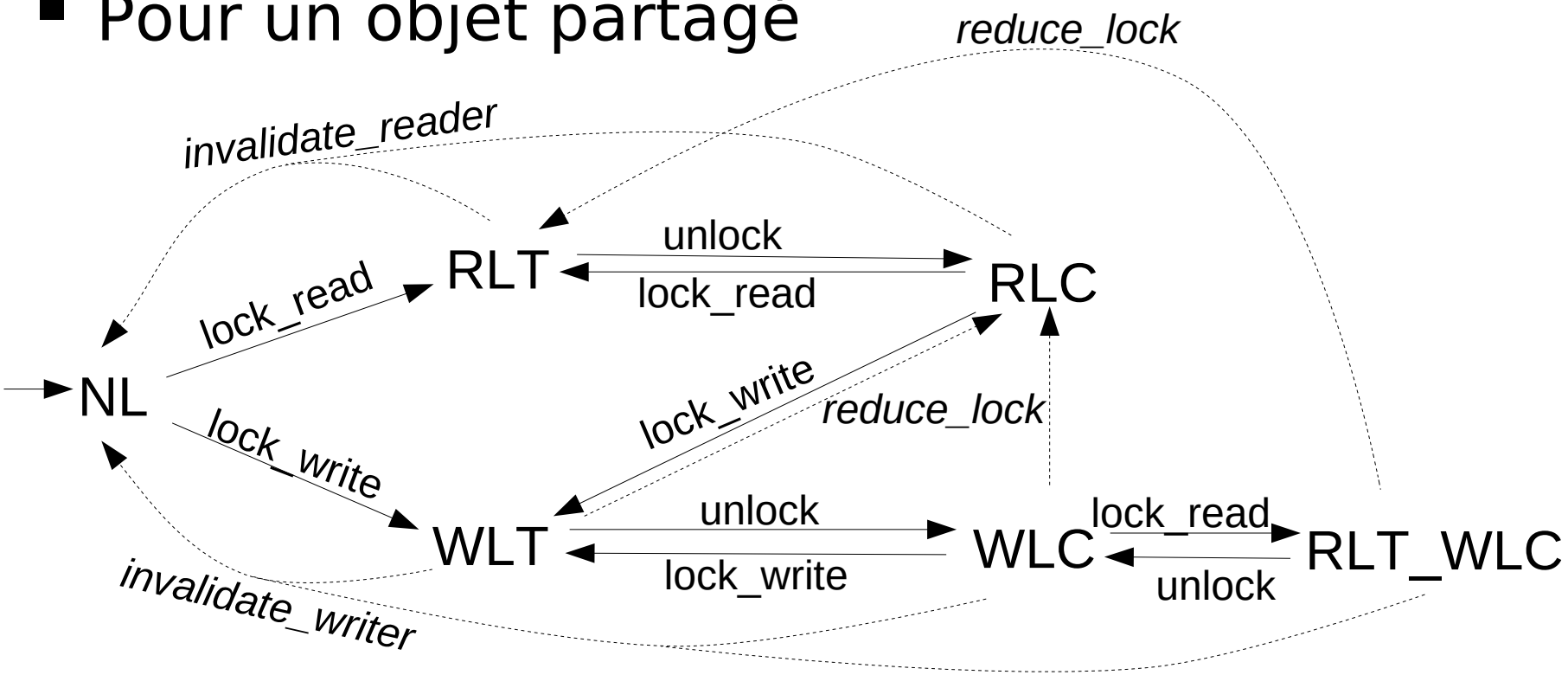
Point de vue serveur

- Pour un objet partagé



Point de vue client

- Pour un objet partagé



Appel par le code utilisateur

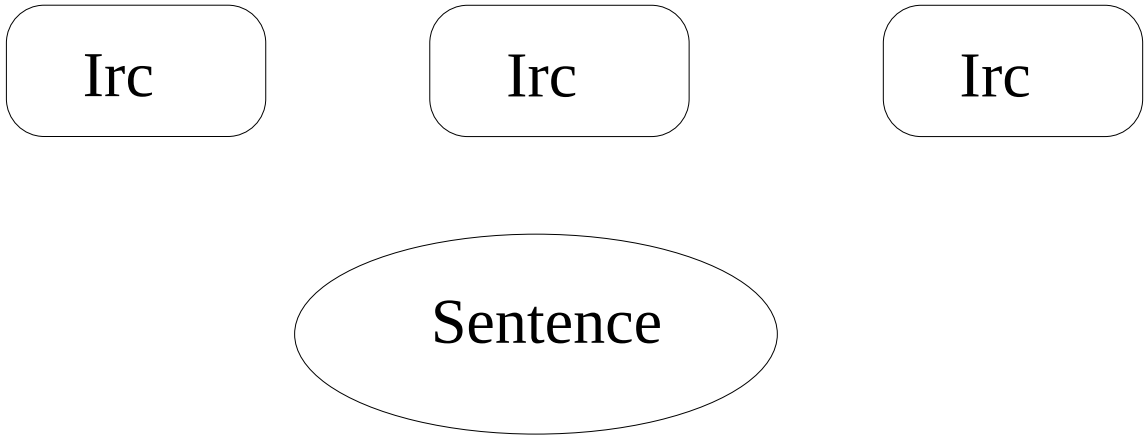
Appel par le serveur (callback)

Gestion des attentes de verrous

- Quand le serveur réclame un verrou au client (invalidate_reader/invalidate_writer/reduce_lock)
- Si le verrou est tenu .... Il faut attendre
  - Dans le client :
    - Méthodes synchronized
    - public void wait() : attente notification sur l'objet
    - public void notify() : notification sur l'objet
  - Un *unlock* appelle *notify()*
  - Une réclamation (dans SharedObject) appelle *wait()* si le verrou est tenu

Projet gestion d'objets dupliqués

- Une application de test (*Irc*)




```
s.lock_read()
((Sentence)(s.obj)).read();
s.unlock()
```

```
s.lock_write()
((Sentence)(s.obj)).write();
s.unlock()
```

Travail à réaliser

- Etape 1
  - Implanter le service à objets dupliqués
  - Utilisation explicite des SharedObject
  - Plusieurs applications peuvent accéder au service de façon concurrente : synchronisation
  - Pas stockage de référence à des SharedObject dans les SharedObject

  - Respecter strictement les interfaces externes !!!
  - Pas de packages

Travail à réaliser

- Etape 1
  - Client
    - Couche Client : les méthodes statiques, service de nommage, redirection vers le serveur
    - Objet réparti pour l'interface de rappel : redirection vers le SharedObject concerné
  - Serveur
    - Objet réparti :
      - redirection vers le ServerObject concerné
      - Serveur de nom
  - SharedObject et ServerObject : protocole de cohérence

Travail à réaliser

- Etape 2
  - Génération de stub pour fournir la transparence d'accès aux objets
  - Stub généré à partir d'une interface Java
    - Interface d'introspection de Java
  - Un stub fournit
    - La même interface que l'objet qu'il représente
    - Les primitives de verrouillage

```
Sentence_itf s = (Sentence_itf)Client.lookup ("sentence");
s.lock_read();
s.read();
s.unlock();
```

Travail à réaliser

- Etape 2
  - Sentence_itf hérite de SharedObject_itf
  - Sentence n'implémente pas Sentence_itf
  - Sentence_stub hérite de SharedObject
  - Sentence_stub implémente Sentence_itf

```
Sentence_itf s = (Sentence_itf)Client.lookup ("sentence");
s.lock_read();
s.read();
s.unlock();
```

Travail à réaliser

- Etape 2
  - Client
    - public static SharedObject lookup(String name);
    - public static void register(String name, SharedObject_itf so);
    - public static SharedObject create(Object o);
  - On peut même faire un système d'annotations :

```
public interface Sentence_itf {
        @write
        public void write(String text);
        @read
        public String read();
}
```

```
Sentence_itf s = (Sentence_itf)Client.lookup(...) ;
s.read();
```

Travail à réaliser

- Etape 3
  - Prendre en compte le stockage de référence à des SharedObject dans les SharedObject
  - Spécialiser les primitives de sérialisation
    - Sérialisation d'un stub
      - Ne pas copier l'objet référencé
    - Désérialisation d'un stub
      - Installer le stub sur la machine de façon cohérente (sans doublons)

Suivi

- Constitution des binômes : 23/11/2022 midi
- Suivi de l'avancement régulier de votre projet
  - séance du 29/11/2022 : validation compréhension et organisation du projet, identification des problèmes, architectures et solutions de principe (étape1)
  - séance du 8/12/2022 : validation des solutions retenues et implantées, tests permettant de valider (étape 1)
  - séance du 5/01/2023 : validation étape 1, conception des étapes 2 et 3
  - séance du 12/01/2023 : identification des difficultés restantes

Evaluation

- Rendu : 23/01/2023 midi
  - Petit rapport
  - Code complet
- Evaluation finale : 27/01/2023
  - Les enseignants ont évalué votre projet (code, rapport) et testé avec leurs propres tests
  - Séance de retour et questions pour affiner l'appréciation

Warnings

- Les séances de suivi sont obligatoires
- Les deadlines sont fermes
  - Constitution des binômes
  - Rendu du projet
- Respectez les interfaces : nous utilisons nos propres tests
- Développez vos propres tests pour valider la robustesse de votre implantation
- Ne copiez pas un projet, nous avons des outils de détection de plagiat