ECOM - Gestion de Stock (concurrence)

Sébastien Chassande-Barrioz
sebastien.chassande-barrioz@cgi.com

CGI

Concurrence de modification du stock : Le problème

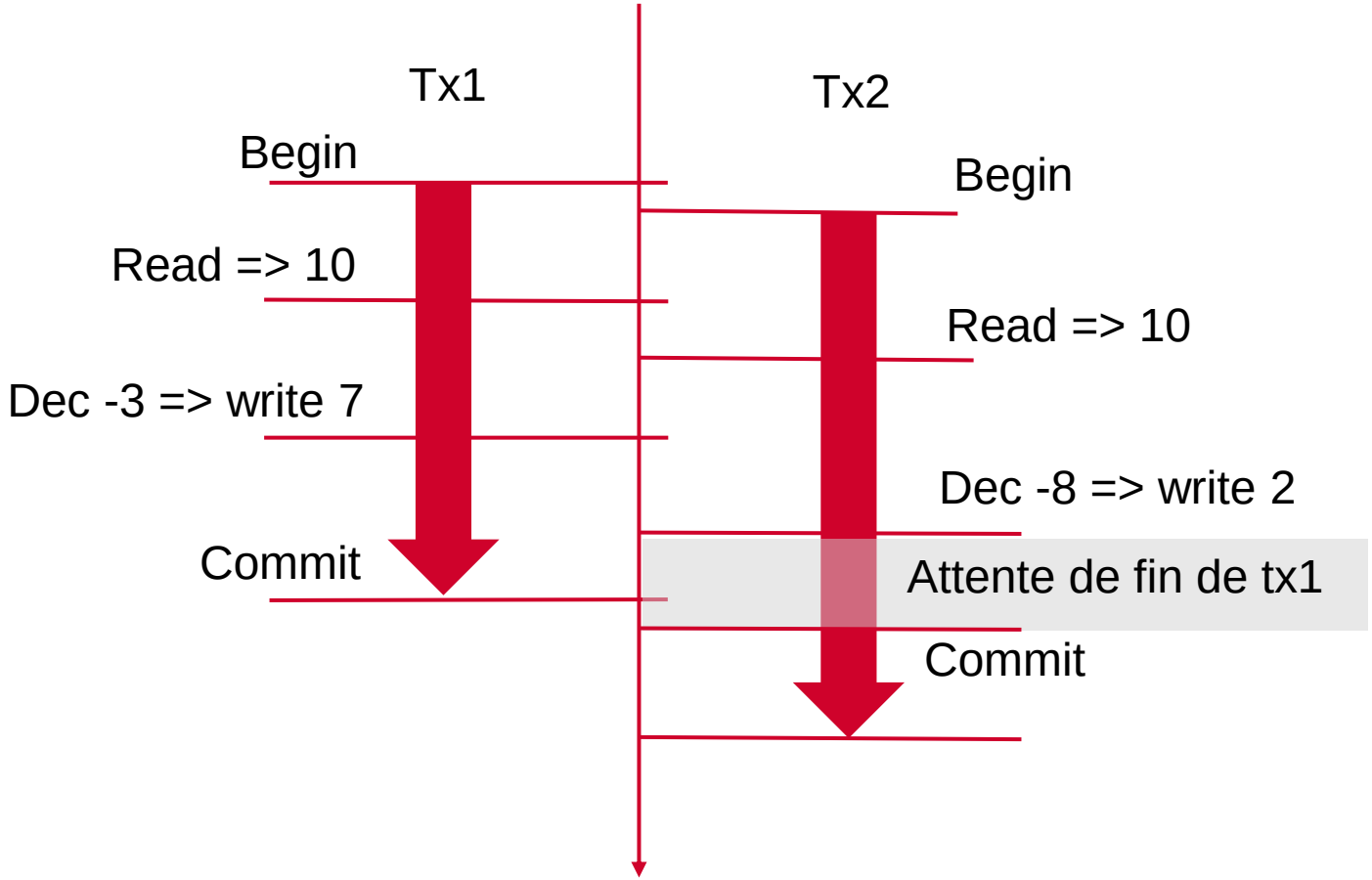
Stock initial à 10, toujours >0

2 transactions concurrentes :

- Décrémenter de 3
- Décrémenter de 8

➔ Ça ne devrait pas être possible

➔ La lecture dans tx2 devrait se faire après le commit de tx1

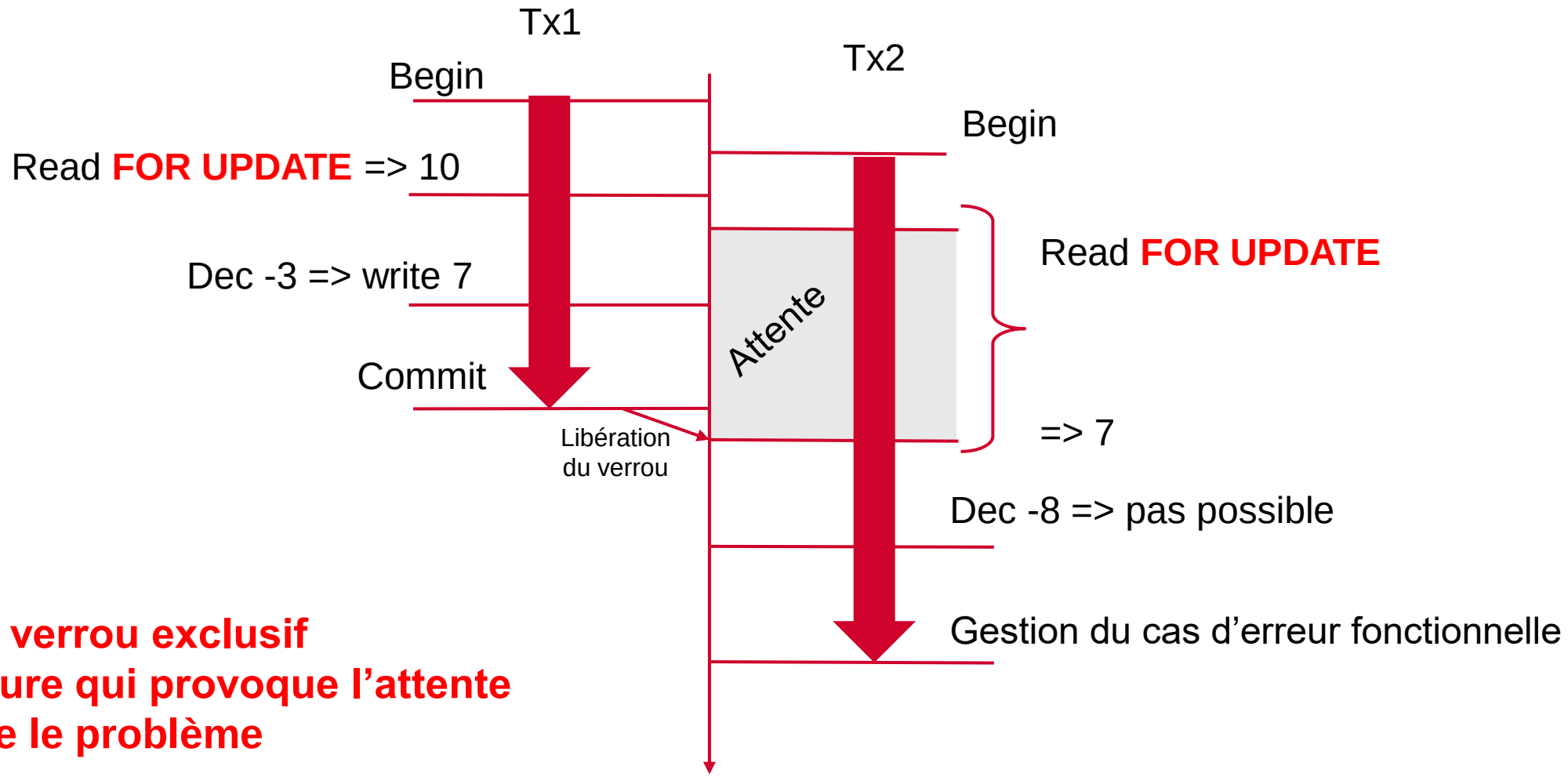


/!\ Stock final à 2

…

© 20XX CGI inc.

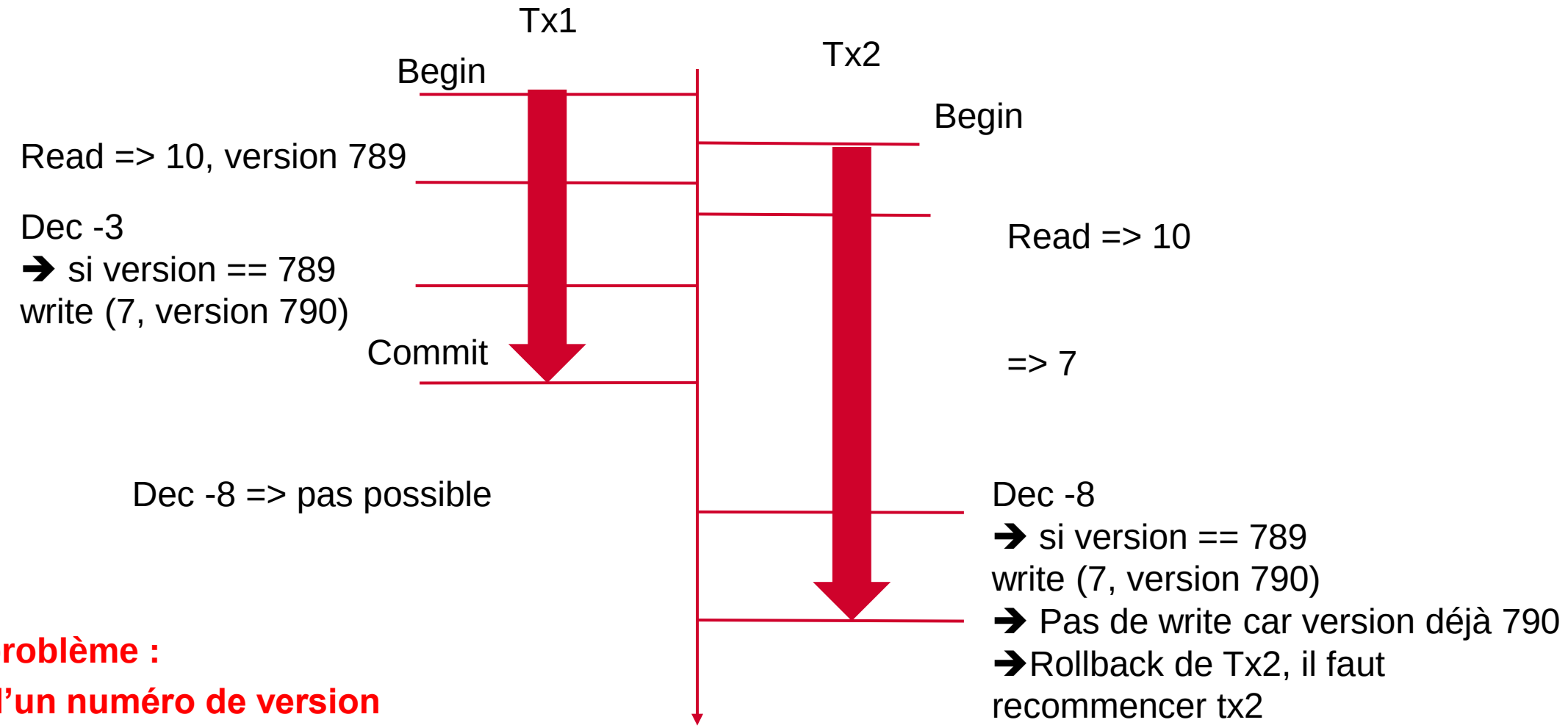
Concurrence de modification du stock : Verrouillage / Pessimiste



Prise d'un verrou exclusif
dès la lecture qui provoque l'attente
➔ On évite le problème

© 20XX CGI inc.

Concurrence de modification du stock : Optimiste



On détecte le problème :

- **Utilisation d'un numéro de version**
- **Requête update avec where version = X**

Il faut recommencer la transaction si problème

Optimiste ou pessimiste, lequel choisir ?

Critère	Optimiste	Pessimiste
Performance	Ne coute rien de plus quand tout va bien	Coute du verrouillage sur la BD tout le temps
Simplicité de mise en oeuvre	Impose de traiter l'exception OptimisticLockException => code un peu plus complexe pour rejouer la transaction	Juste préciser le verrouillage lors de l'accès
Cas d'usage favorable	Modifications concurrentes **occasionnelles**	Modifications concurrentes **fréquentes**
Cas d'usage défavorable	Modifications concurrentes **fréquentes**	Modifications concurrentes **occasionnelles**

➔ Pas de solution parfaite. Il connaître son cas d'utilisation et choisir

© 20XX CGI inc.

Verrouillage en JPA

Le principe :

- Lors de la récupération de l'objet ou avant la lecture ou l'écriture des données, il faut indiquer le verrouillage ***LockModeType.PESSIMISTIC_WRITE***
- Optionnel : traiter l'exception ***PessimisticLockException***

```
try {
  // Récupération du produit avec un accès en écriture
  Product product = entityManager.find(Product.class, productId, LockModeType.PESSIMISTIC_WRITE);

  // Lecture et mise à jour du stock
  product.setStock(product.getStock() - nbProduct);

} catch (PessimisticLockException | LockTimeoutException e) {
  // Gestion de l'echec du verrouillage causé par un dead lock
  // ... Sortir de la fonction et recommencer la transaction par exemple
}
```

© 20XX CGI inc.

https://www.baeldung.com/jpa-pessimistic-locking

Concurrence Optimiste en JPA

1 - Dans la classe persistante, déclarer un champ « version » de type Long, avec l'annotation **@Version**

```
@Entity
public class Product {

  @Version
  @Setter
  @Getter
  Long version;

  ...
}
```

2 - Préciser l'**accès optimiste en écriture** lors de l'acquisition de l'objet

3 - Traiter les exceptions **OptimisticLockException** et recommencer la tx

```
@Transactional(REQUIRED)
public void decStockUnderTx(Long productId, int nbProduct) {
  // Récupération du produit avec un accès en écriture
  Product product = entityManager.find(Product.class, productId, LockModeType.WRITE);

  // Lecture et mise à jour du stock
  product.setStock(product.getStock() - nbProduct);
}


@Transactional(NOT_SUPPORTED)
public boolean decStock(Long productId, int nbProduct) {
  int attempt = 0;
  while(attempt<3)
  try {
    decStockUnderTx(productId, nbProduct);
    return true
  } catch (OptimisticLockException e) {
    // On recommence la transaction
    attempt++;
  }
  // Trop d'essai on abandonne
  return false;
}
```

© 20XX CGI inc.

https://www.baeldung.com/jpa-optimistic-locking

En python

Pour démarquer une transaction :

```
with transaction.atomic()
```

Pour verrouiller un produit :

```
select_for_update(of='self', skip_locked=True)
```

Un peu de lecture :

https://lukeplant.me.uk/blog/posts/double-checked-locking-with-django-orm/?ref=sangkon.com

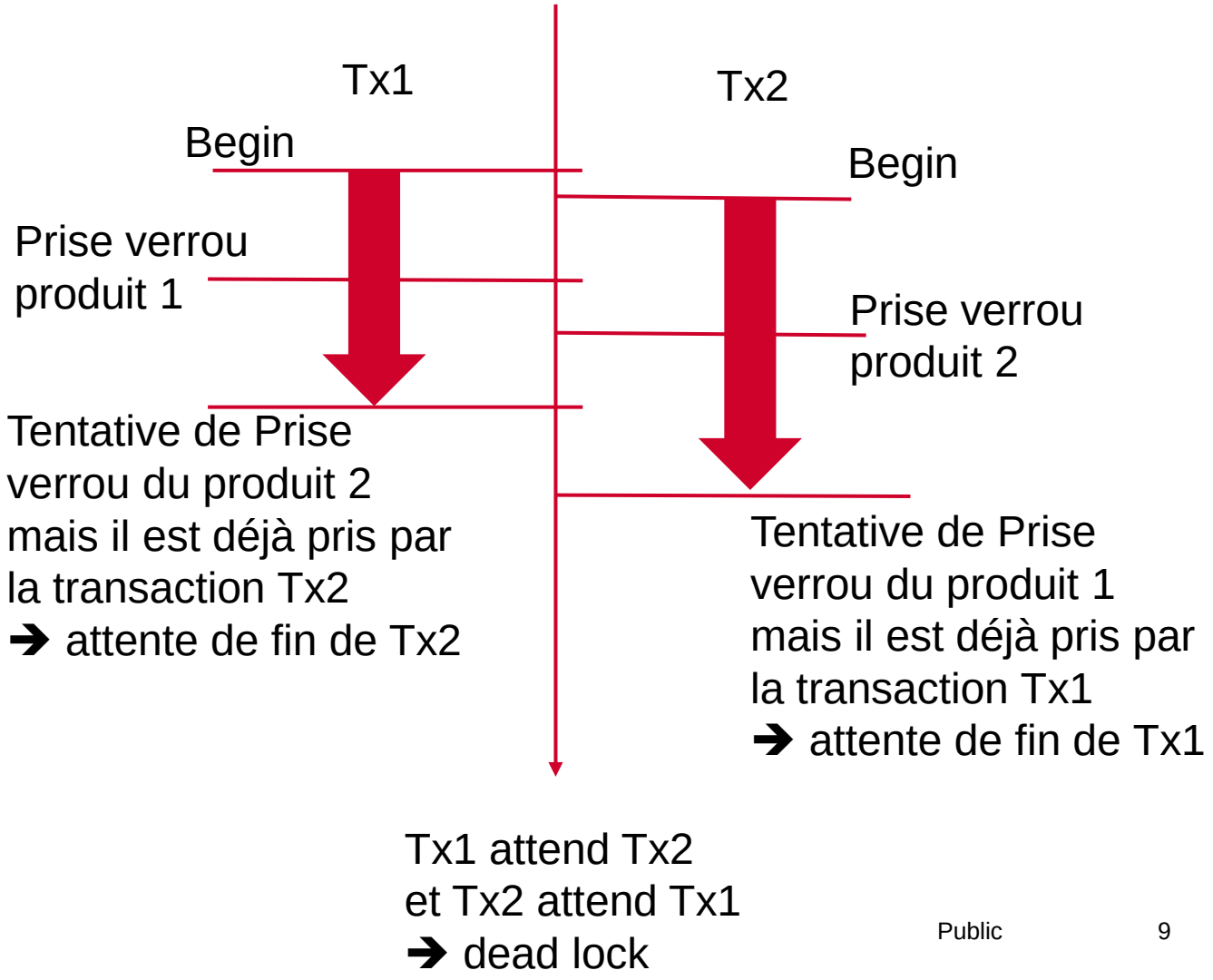
© 20XX CGI inc.

Le problème du dead Lock

Objectif : prendre un verrou sur plusieurs produits

Problématique : Si l'ordre de prise de verrou est mauvais alors il peut apparaitre un dead lock

Solution : trier les données pour prendre toujours les verrous dans le même ordre



© 20XX CGI inc.