# Documentation :

## 1-pseudocode

1-Writer process:
1.Writer requests the entry to critical section.
2.If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3.It exits the critical section.

- The structure of a writer's process

```
while (true) {
wait(rw_mutex); // any writers or readers?
...
/* writing is performed */
...
signal(rw_mutex); // enable others
}
```

Reader process:
1.Reader requests the entry to critical section.
2.If allowed:
.it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.
.It then, signals mutex as any other reader is allowed to enter while others are already reading.
.After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore "wrt" as now, writer can enter the critical section.
3.If not allowed, it keeps on waiting.

- The structure of a reader process

```
do {
wait(mutex);
read_count++; // ensure mutual exclusion
if (read_count == 1) // another reader
wait(rw_mutex); // block writers
signal(mutex); // release lock for other readers
...
/* reading is performed */
...
wait(mutex); // ensure mutual exclusion
read count--; // reader done
if (read_count == 0)
signal(rw_mutex); // enable writers
signal(mutex); // release lock for other readers
} while (true);
```

## 2- Examples of Deadlock

1- if two or more writers want use a resource at the same time.

2- If writer want use a resource (before)and then reader(frist reader) want use a resource

## 3- How did solve deadlock

1- Only one writer acess a resource

2- The reader must wait the writer

**Deadlock prevention:** The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock.

**Deadlock avoidance:** Operating system runs an algorithm on requests to check for a safe state. Any request that may result in a deadlock is not granted.

**Deadlock detection & recovery:** OS detects deadlock by regularly checking the system state, and recovers to a safe state using recovery techniques.

## 4- Examples of starvation

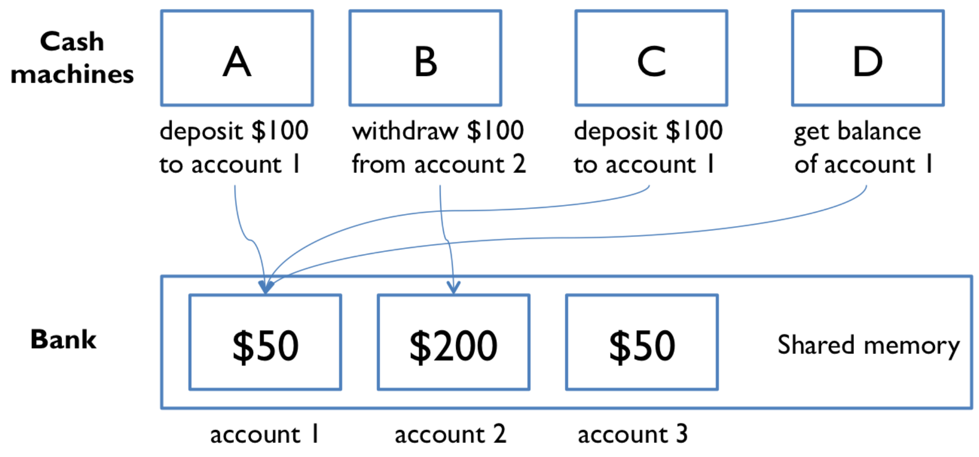
1- 1-if there is a group of readers using the resource then writer want use a resource then reader request use a resource . The writer does not allow this reader (its come after him)to work except when he starts and finishes his work

2- 2- if there is a group of writers using the resource then reader want use a resource then writer request use a resource . The reader does not allow this writer (its come after him)to work except when he starts and finishes his work

## 5-How did solve starvation

# Real World ex(Reader_Writer)

## Bank account example



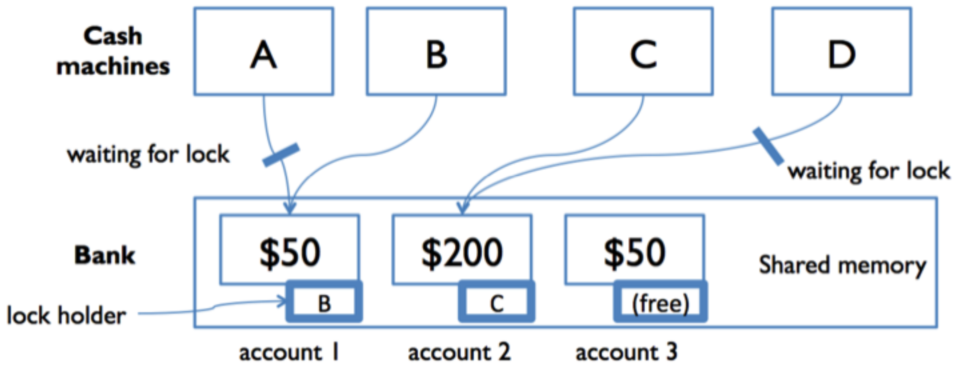
Our example of shared memory concurrency was a bank with cash machines. The diagram from that example is on the right.

The bank has several cash machines, all of which can read and write the same account objects in memory.

Of course, without any coordination between concurrent reads and writes to the account balances, things went horribly wrong (Race condition).

## The solution:

To solve this problem with locks, we can add a lock that protects each bank account. Now, before they can access or update an account balance, cash machines must first acquire the lock on that account.



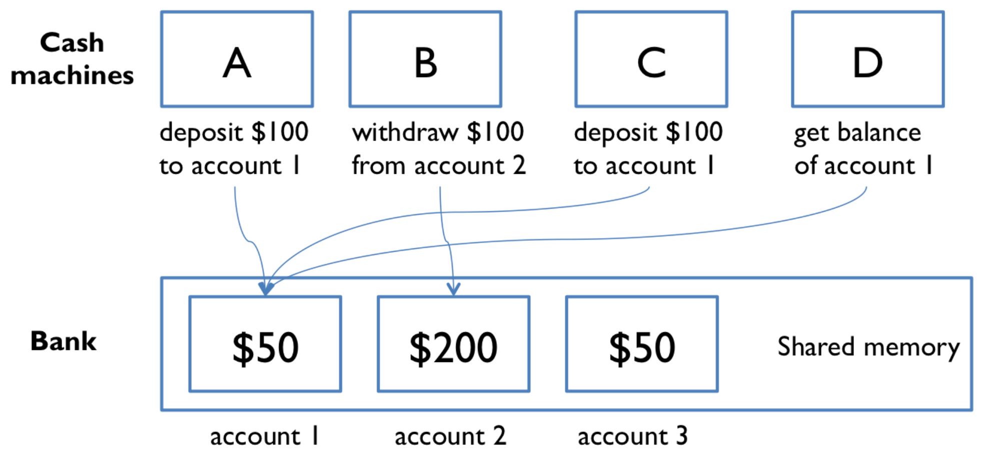
In the diagram to the right, both A and B are trying to access account 1. Suppose B acquires the lock first. Then A must wait to read and write the balance until B finishes and releases the lock. This ensures that A and B are synchronized, but another cash machine C is able to run independently on a different account (because that account is protected by a different lock).

**Solution to Starvation:** Aging
Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time

## 6-Example of real world And how did apply the problem

# Real World ex(Reader_Writer)

### Bank account example



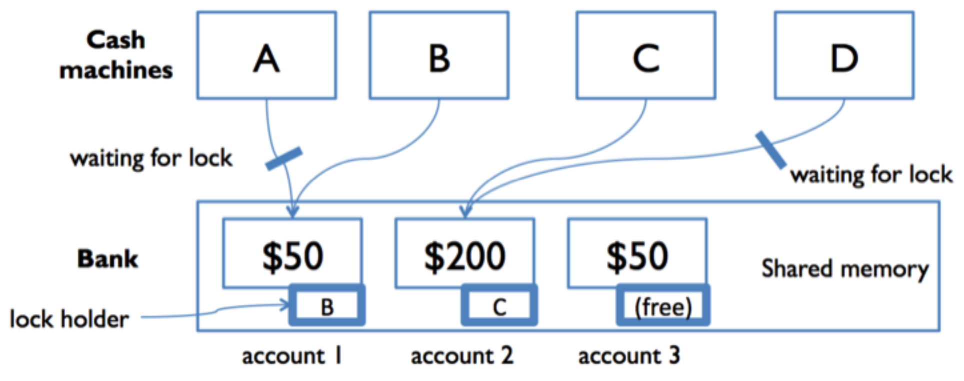
Our example of shared memory concurrency was a bank with cash machines. The diagram from that example is on the right.

The bank has several cash machines, all of which can read and write the same account objects in memory.

Of course, without any coordination between concurrent reads and writes to the account balances, things went horribly wrong (Race condition).

# The solution:

To solve this problem with locks, we can add a lock that protects each bank account. Now, before they can access or update an account balance, cash machines must first acquire the lock on that account.



In the diagram to the right, both A and B are trying to access account 1. Suppose B acquires the lock first. Then A must wait to read and write the balance until B finishes and releases the lock. This ensures that A and B are synchronized, but another cash machine C is able to run independently on a different account (because that account is protected by a different lock).