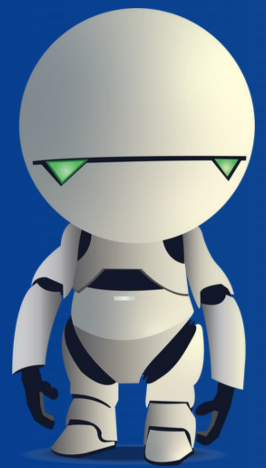
Shared Memory and Consistency Models

CS323 - Parallel and Distributed Computing

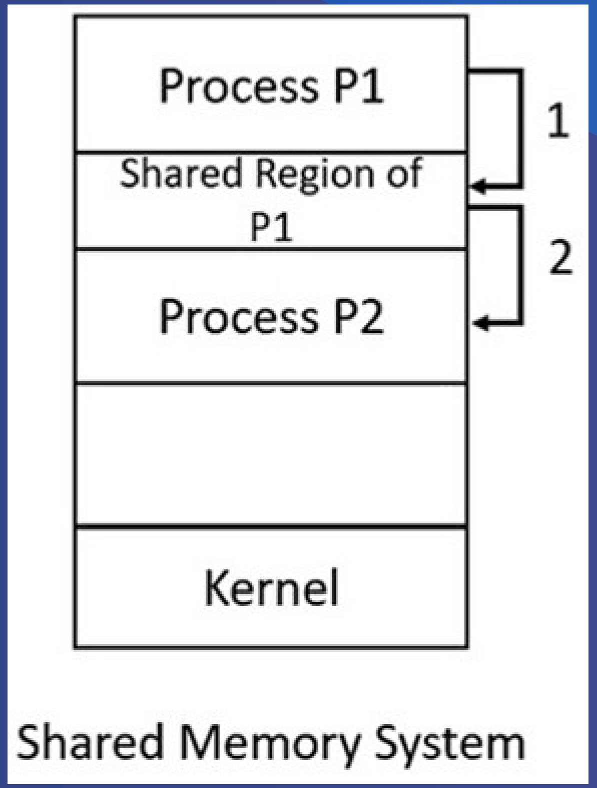
Intended Learning Outcomes

1. *to explain the concepts of shared memory and consistency models, understanding how these models help manage data accessed by multiple processes in parallel systems.*
2. *to use Python's multiprocessing module to set up shared memory between processes, demonstrating how processes can access and modify shared data.*
3. *to recognize common challenges related to data consistency in shared memory systems and apply basic synchronization techniques to ensure reliable data access.*

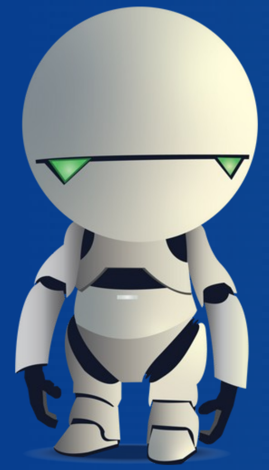


Shared Memory

- ***Shared Memory*** *refers to a memory space that can be accessed by multiple processes concurrently.*
- *It provides a way for processes to communicate by reading from and writing to the same region of memory.*
- *This approach is commonly used in parallel and distributed systems where multiple processes or threads need to share data without the overhead of message-passing mechanisms*
- *Types of Shared Memory Systems:*
  - ***Centralized Shared memory*** *– all processes access the same physical memory*
  - ***Distributed Shared memory*** *– each node in a distributed system has its own local memory, but they are logically treated as a single shared memory space*



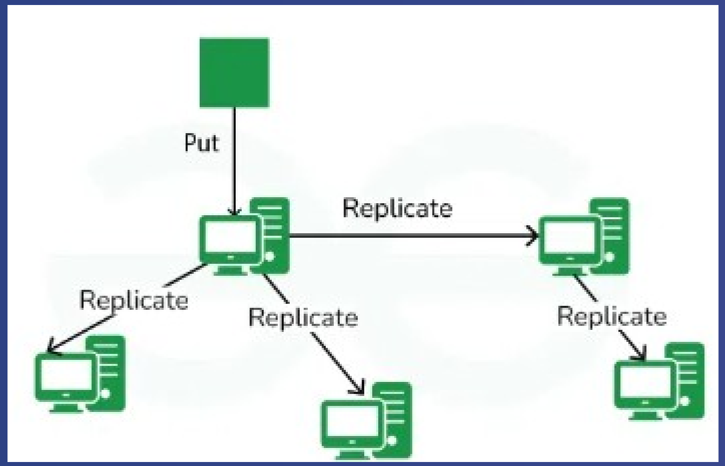
Shared Memory System

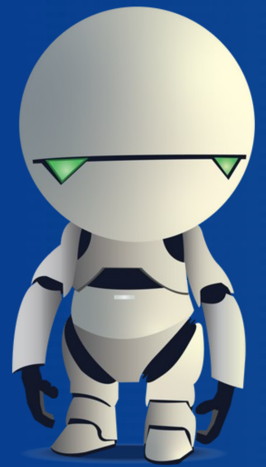


Consistency Models

- *A* ***consistency model*** *defines the rules for how updates to shared memory are seen by different processes.*
- *There are various consistency models, each with different guarantees on how changes made by one process are visible to others*

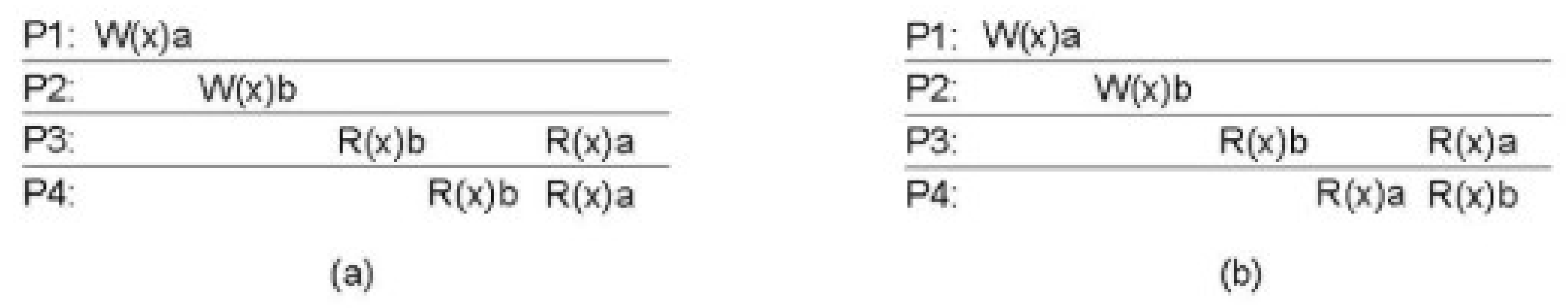
1. ***Strong Consistency Model*** *– in a strongly consistent system, all nodes in the system agree on the order in which operations occurred. Reads will always return the most recent version of the data (most recent write). This provides the highest level of consistency, but it can be slower and require more resources in a distributed environment since all must stay perfectly in sync.*



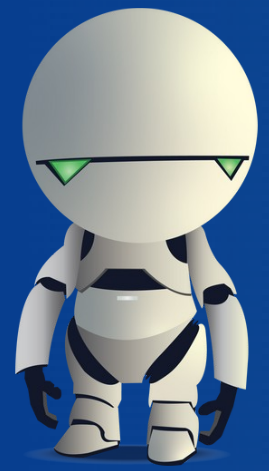


Consistency Models

*2. **Sequential Consistency Model**– It is a consistency model in distributed systems that ensures all operations across processes appear in a single, unified order. In this model, every read and write operation from any process appears to happen in sequence, regardless of where it occurs in the system. Importantly, all processes observe this same sequence of operations, maintaining a sense of consistency and order across the system.*

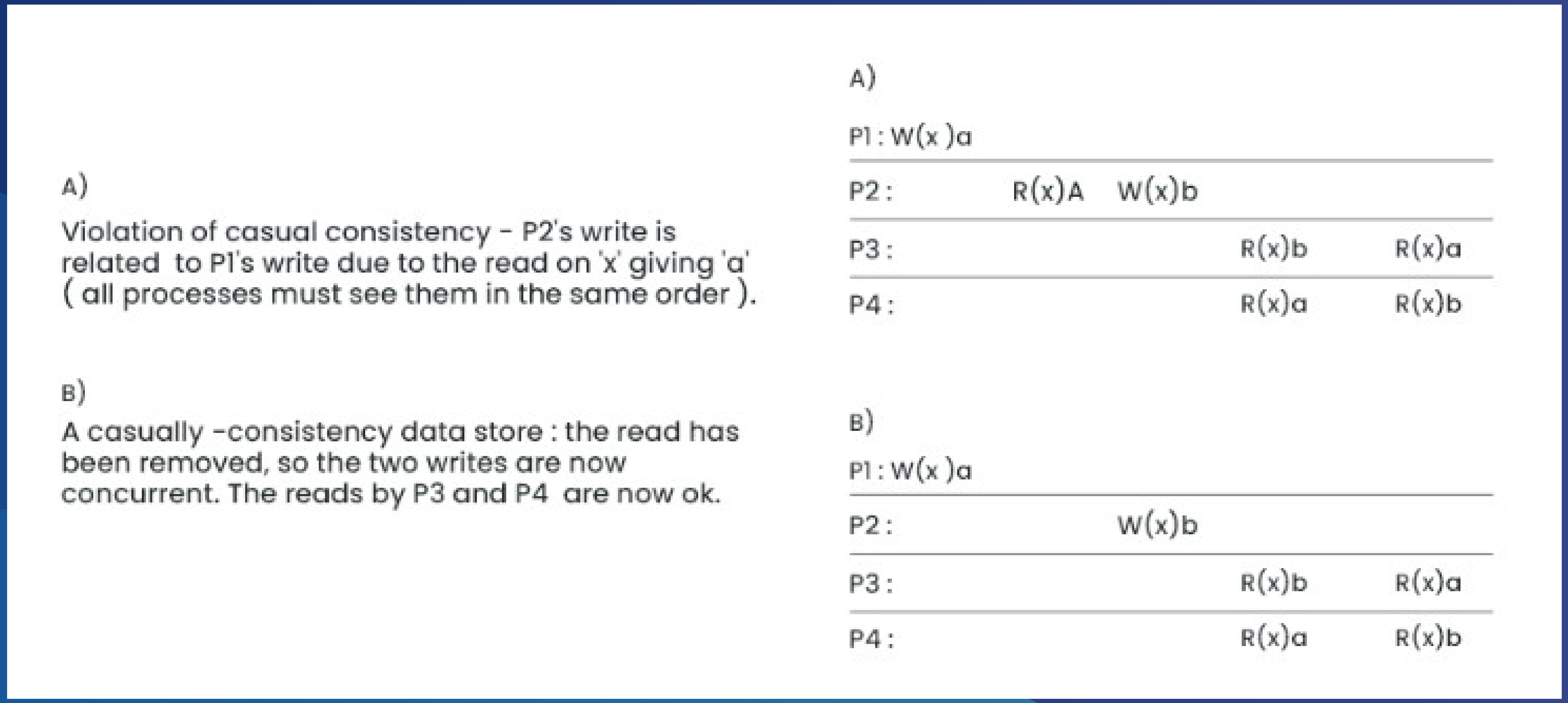


a) A sequentially consistent data store.
b) A data store that is not sequentially consistent.



Consistency Models

*3. **Causal Consistency Model**– is a type of consistency in distributed systems that ensures that related events happen in a logical order. In simpler terms, if two operations are causally related (like one action causing another), the system will make sure they are seen in that order by all users. However, if there's no clear relationship between two operations, the system doesn't enforce an order, meaning different users might see the operations in different sequences.*



A)

Violation of casual consistency - P2's write is related to P1's write due to the read on 'x' giving 'a' (all processes must see them in the same order).

B)

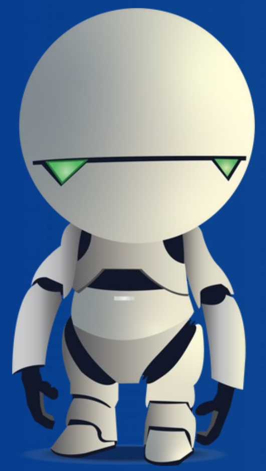
A casually -consistency data store : the read has been removed, so the two writes are now concurrent. The reads by P3 and P4 are now ok.

A)

P1 : W(x)a				
P2 :	R(x)A	W(x)b		
P3 :			R(x)b	R(x)a
P4 :			R(x)a	R(x)b

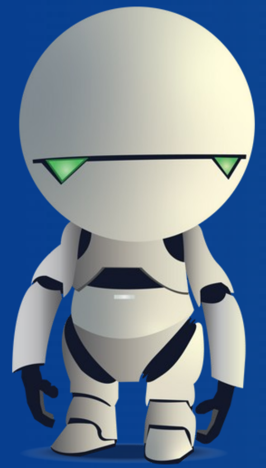
B)

P1 : W(x)a			
P2 :	W(x)b		
P3 :		R(x)b	R(x)a
P4 :		R(x)a	R(x)b



Synchronization techniques

- ***Process synchronization*** *is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources at the same time.*
- *To maintain consistency in shared memory, various synchronization techniques are used*
  - ***Locks (Mutexes)*** *– A mutex, or mutual exclusion object, is a locking mechanism that allows only one process can access a shared resources at a time. When a thread locks a mutex, other threads attempting to lock the same mutex will be blocked until the mutex is unlocked. This ensures that critical sections of code that access shared resources are executed by only one thread at a time.*
  - ***Semaphores*** *– allow controlled access to shared resources by multiple processes. Unlike mutexes, semaphores can allow multiple threads to access a resource simultaneously, up to a specified limit.*



Lab Activity 4

- *Use Python's multiprocessing module to set up shared memory between processes, demonstrating how processes can access and modify shared data.*
- *Screenshots the code (in parts) and all the output. Save all screenshots in one pdf and submit it to USTEP*