Deadlocks in Distributed Systems

Deadlocks in Distributed Systems

Deadlocks in distributed systems are similar to deadlocks in single-processor systems but much harder to handle.

This is because the information needed to detect, prevent, or fix deadlocks is spread across many machines.

Types of Deadlocks:

Some experts classify distributed deadlocks into two types:

1. **Communication Deadlocks:**
    - Example: Process A tries to send a message to process B, B tries to send to process C, and C tries to send back to A. If no message buffers are available, they all get stuck waiting for each other.
    - These are rare, especially in client-server systems where this kind of circular waiting is unlikely.
2. **Resource Deadlocks:**
    - Example: Processes are waiting for exclusive access to resources like files, I/O devices, or locks.

In most cases, the distinction isn't very important because communication channels, buffers, and other such resources can also be modeled as resources. The main focus is on resource deadlocks because they are more common.

Handling Deadlocks in Distributed Systems

Four major strategies are used to manage deadlocks in distributed systems. Here's a brief overview:

1. **The Ostrich Algorithm (Ignore the problem):**
    - Just like ignoring the problem in single-processor systems, this approach is also popular in distributed systems. Many systems don't have a system-wide deadlock mechanism, but individual applications (like databases) may handle deadlocks themselves.
2. **Deadlock Detection and Recovery:**
    - This approach allows deadlocks to happen and then attempts to detect and recover from them.
    - It's a popular method in distributed systems because preventing or avoiding deadlocks can be difficult.

3. **Deadlock Prevention:**
    - This method attempts to make deadlocks impossible by changing how resources are allocated.
    - It's harder to implement in distributed systems compared to single-processor systems, but in the case of atomic transactions (where operations are grouped together), some new methods can be used.
4. **Deadlock Avoidance:**
    - Deadlock avoidance requires knowing in advance how many resources each process will need, which is almost impossible in real-world systems.
    - This technique is not used in distributed systems (and rarely even in single-processor systems).

Distributed Deadlock Detection

Why is Deadlock Detection Important?

Preventing or avoiding deadlocks in distributed systems is very difficult, so most researchers focus on detecting deadlocks rather than trying to stop them from happening.

In **distributed systems with atomic transactions**, deadlock detection becomes easier to manage, because there's a good way to handle deadlocks when they occur.

Handling Deadlocks in Traditional Systems vs. Systems with Atomic Transactions

1. In Traditional Systems:
    - When a deadlock is detected, the solution is to kill one or more processes involved in the deadlock.
    - This can make users unhappy because their work is interrupted, and processes might be lost.

2. **In Systems with Atomic Transactions:**
    - When a deadlock is detected, instead of killing processes, the system **aborts** one or more transactions.
    - Transactions are designed to handle being aborted, so this is much less harmful than killing a process.
    - The system simply rolls back to the state it was in before the transaction began, and the transaction can start again.
    - If everything goes well, the transaction will succeed on the second attempt.

Centralized Deadlock Detection

As a first attempt, we use a centralized deadlock detection algorithm and then try to imitate the non-distributed algorithm. Here,

1. Each machine maintains the resource graph for its own processes and resources, and a central coordinator maintains the resource graph for the entire system (the union of all the resource graphs)

1. When the coordinator detects a cycle, it kills off one process to break the deadlock.

Contd ..

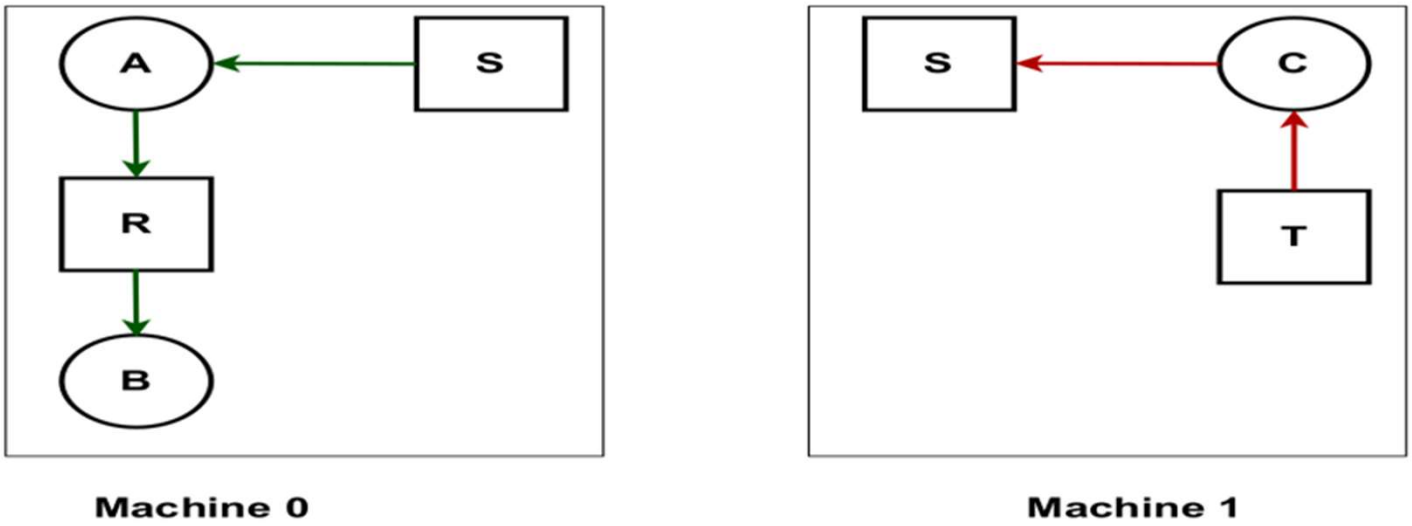
Unlike the centralized system (single processor), in a distributing environment, all the information has to be sent to the right place explicitly. Here, Each machine maintains a graph for its own processes and resources.

Possible situations to get the informations to the right place:

1. Whenever, an arc is added or deleted from the resource graph, a message can be sent to the coordinator providing the update.
2. Periodically, every message can send a list of arcs added or deleted since the previou update.
3. The coordinator can ask for informations when it needs it.

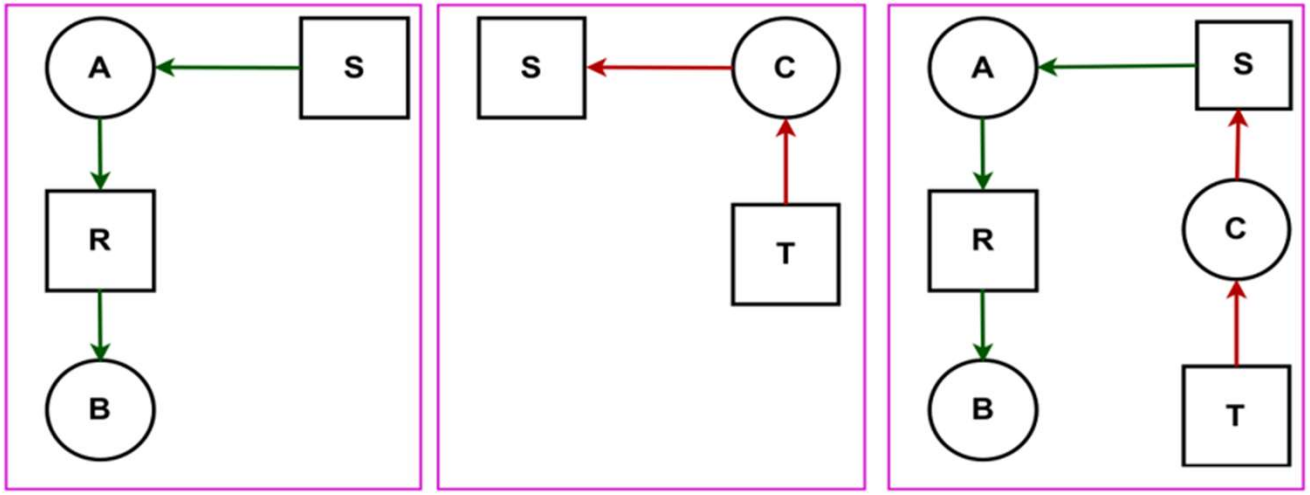
Unfortunately, none of these methods work well. Why ? Explain in the next slides

Centralized Deadlock Detection Explanation



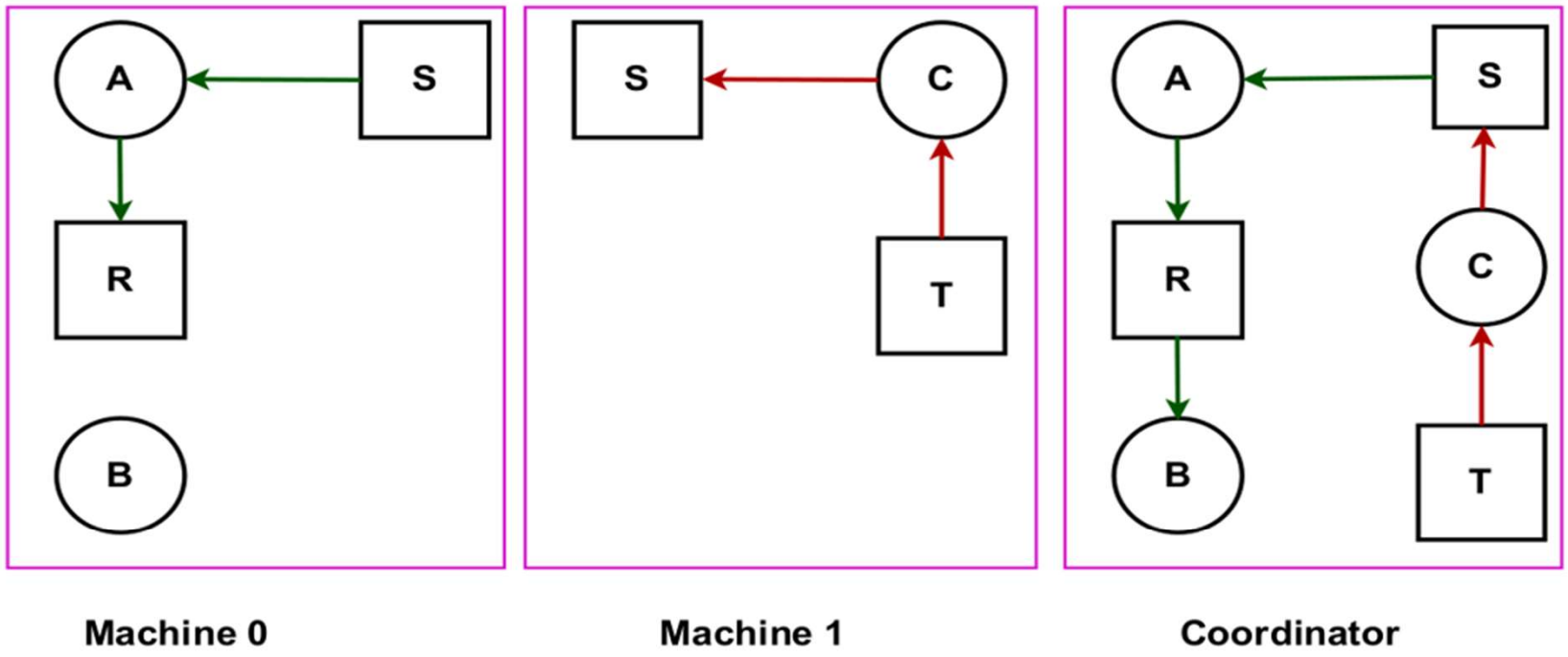
1. Process A and Process B are running in Machine 0, and Process C is running in Machine 1.
2. Three resources exist: R, S, and T
3. A holds S but wants R, which it cannot have because B is using it; C has T wants S.

Contd ..



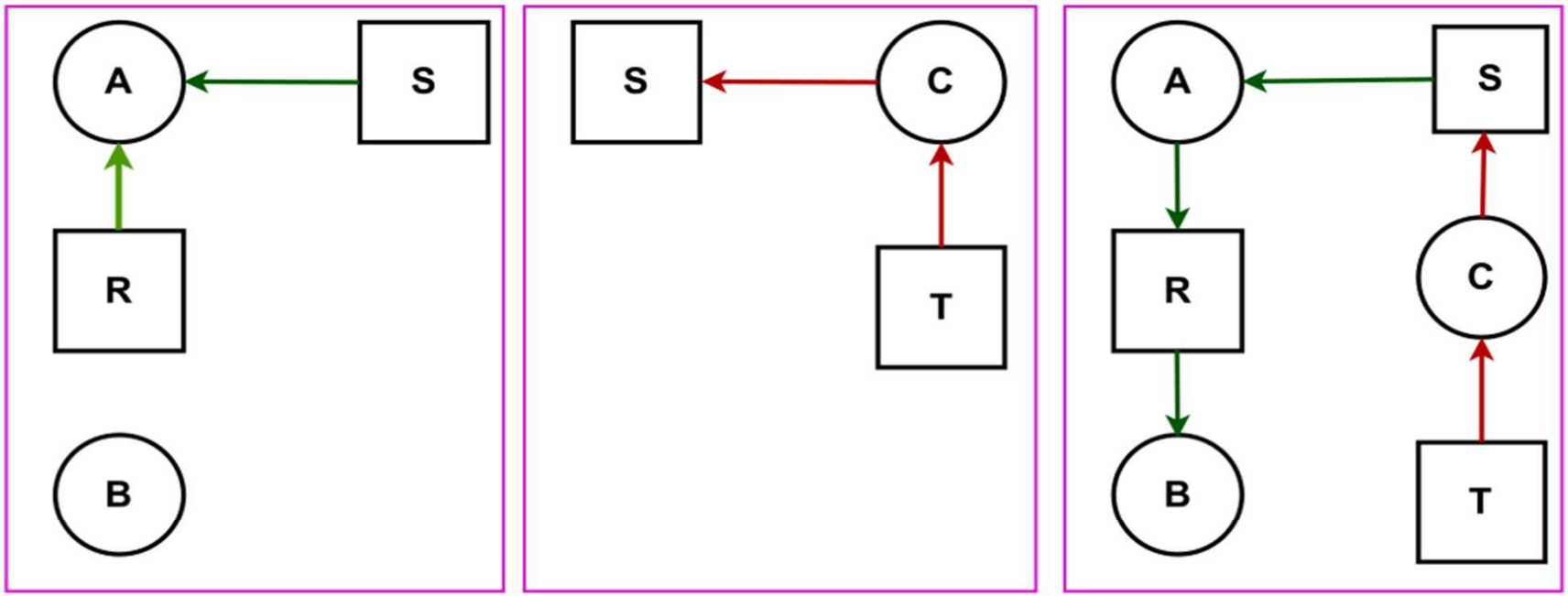
We can see the coordinator view of Machine 0 and Machine 1

Contd..



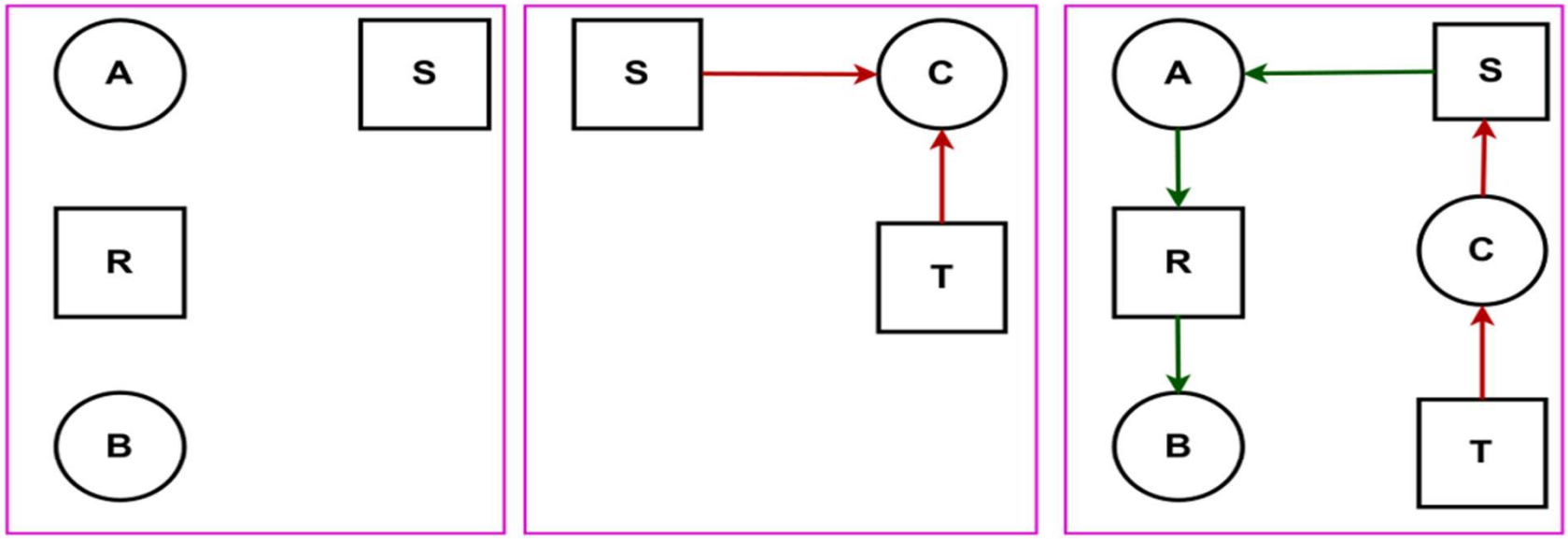
As soon as B finishes, It releases R

Contd ..



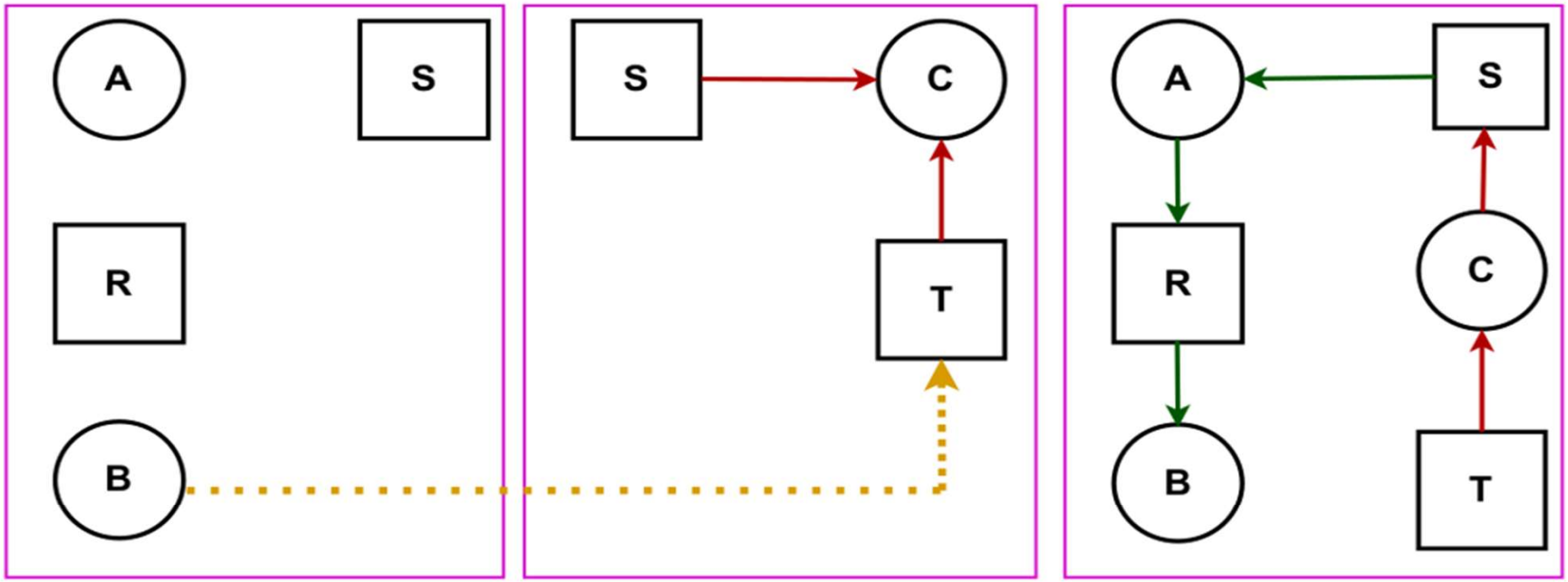
A gets R and finishes its task, releasing S for C

Contd..



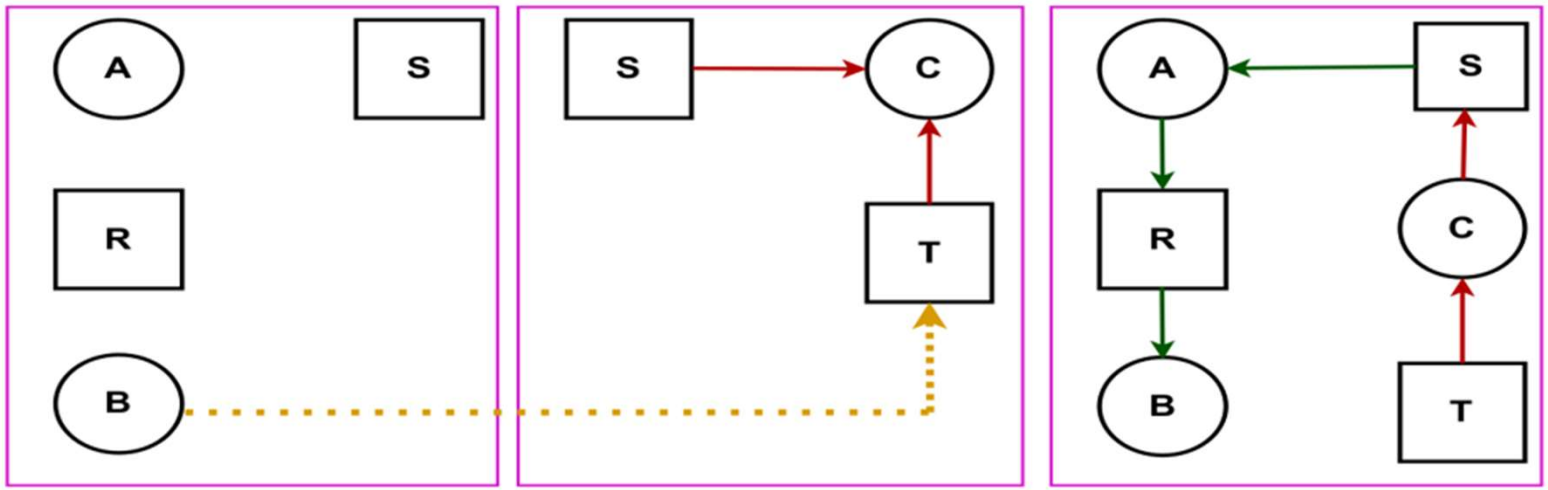
Now C has both S & T, now C can execute its task

Contd..



Meanwhile, B is requesting for T

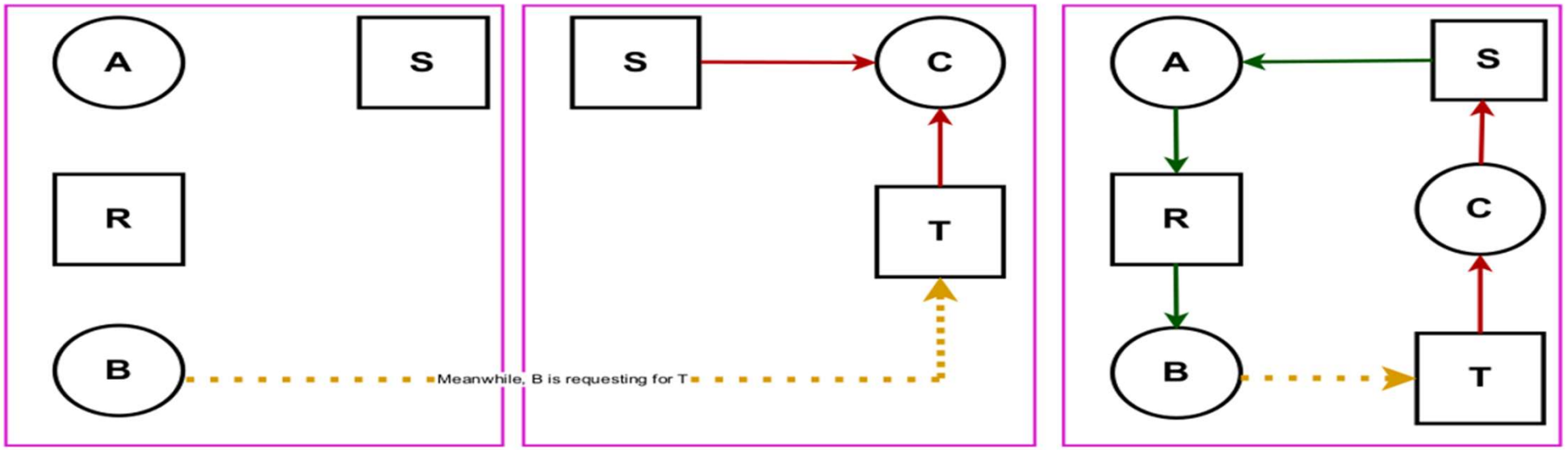
Contd..



Machine 0 sends a message to the coordinator announcing the release of R

Machine 1 sends a message to the coordinator announcing the fact that B is requesting its resource, T

Contd..



Unfortunately, the message from Machine 1 arrives first, leading the coordinator to construct the graph as shown above.

Coordinator incorrectly concludes that a deadlock exists and kills some process

Such a situation is called **False deadlock**

Contd ..

1. Many deadlock algorithms in distributed systems produce **False deadlocks** like this due to incomplete or delayed information.
2. One possible way out might be to use Lamport's algorithm to provide global time.

   i) Message from machine 1 to the coordinator is triggered by the request from machine 0, so the message from machine 1 will have later timestamp than the message from machine 0 to the coordinator.

   ii) When the coordinator gets a message from machine 1 that leads it to suspect deadlock, it could send a message to every machine to the system saying that: " I just received a message with timestamp T that leads to deadlock. If anyone has a message for me with earlier timestamp, send me immediately."

   iii) When every message will send positively or negatively, the coordinator will see that arc from R to B is vanished. Hence, the system is still safe.