CS61065: Theory And Applications of Blockchain

Consensus in Permissioned Settings

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Permissioned Model

- A blockchain architecture where users are authenticated a priori
  - A Membership Service Provider (MSP) helps to obtain the chain membership
- Users know each other
  - However, users may not trust each other – Security and consensus are still required.
- Run blockchain among known and identified participants

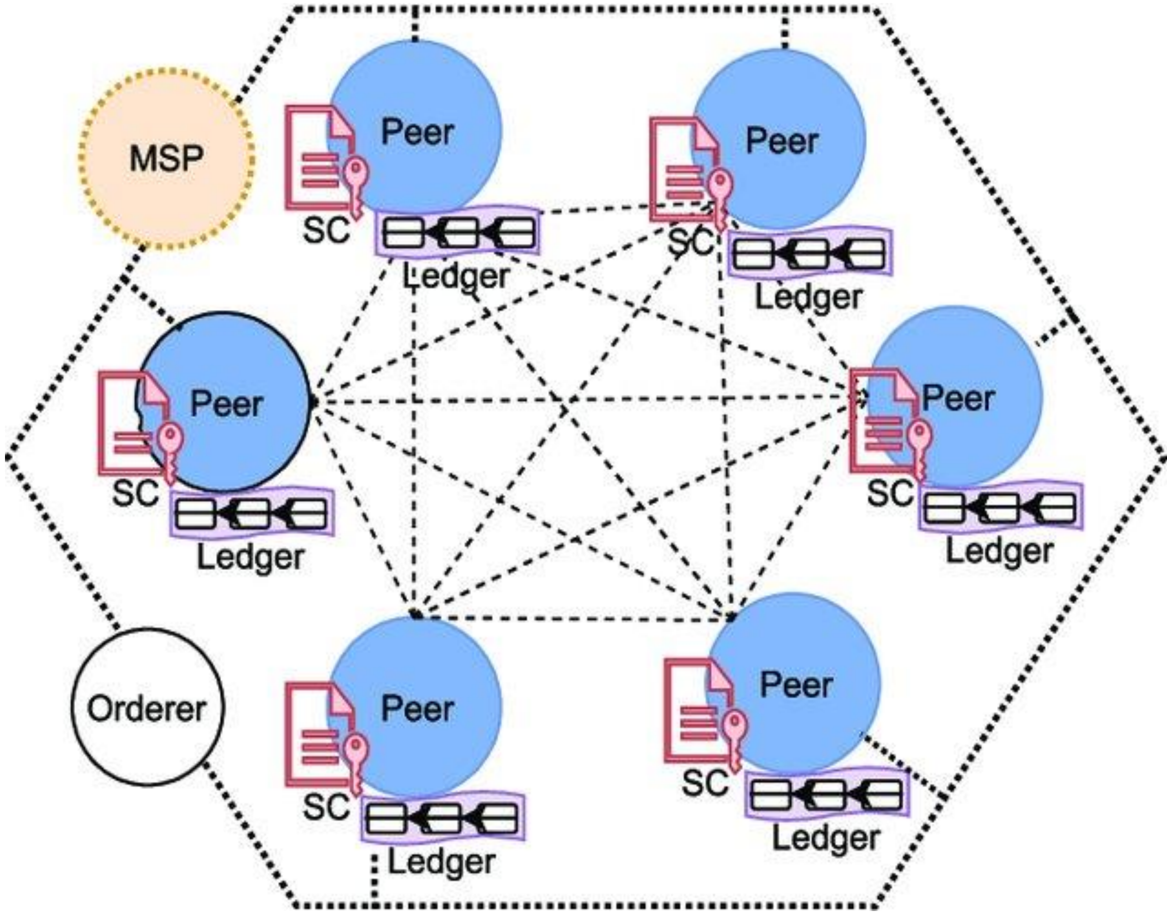


Image Source: https://ieeexplore.ieee.org/document/8481466

Permissioned Model – Use Cases

- Particularly interesting for business applications – execute contracts among a closed set of participants

- Example: Provenance tracking of assets in a supply chain

Executing Contracts over a Closed Network

- **Smart Contracts:** “A self-executing contract in which the terms of the agreement between the buyer and the seller is directly written into the lines of code” - http://www.scalablockchain.com/

- **Agreement on a Smart Contract Execution:**
  - Store the contract on a blockchain
  - Once an event is triggered, execute the codes locally on each peer
  - Generate transactions as the output of the contract execution
  - The peers of the blockchain network validates the transaction, and the output is committed in the blockchain – may trigger the next event to execute the code further

Executing Contracts over a Closed Network

- **Smart Contracts:** “A self-executing contract in which the terms of the agreement between the buyer and the seller is directly written into the lines of code” - http://www.scalablockchain.com/

- **Agreement on a Smart Contract Execution:**
  - Store the contract on a blockchain
  - Once an event is triggered, execute the codes locally on each peer
  - Generate transactions as the output of the contract execution
  - The peers of the blockchain network validates the transaction, and the output is committed in the blockchain – m [illegible] ode further

Do we really need to execute the code on each peer?

When does each peer execute the code?

Smart Contract Agreement as a State Machine Replication

- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "*proof*" that a peer has agreed on the "*state of execution*"

Smart Contract Agreement as a State Machine Replication

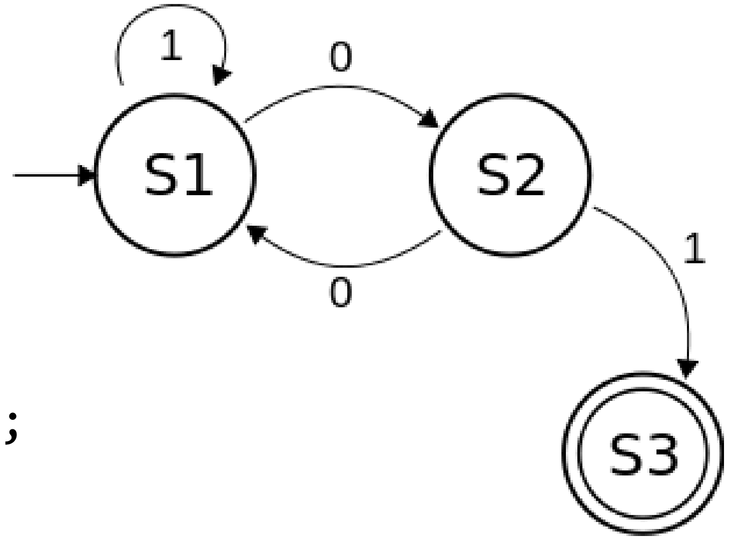
- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "*proof*" that a peer has agreed on the "*state of execution*"

How will we generate the proof?

Smart Contract Agreement as a State Machine Replication

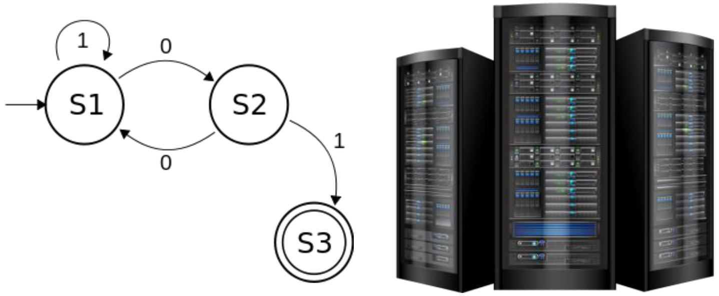
- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "*proof*" that a peer has agreed on the "*state of execution*"
- **State Machine Replication:**
  - Represent the smart contract as a state machine – Remember, any deterministically executable code can be represented as a state machine

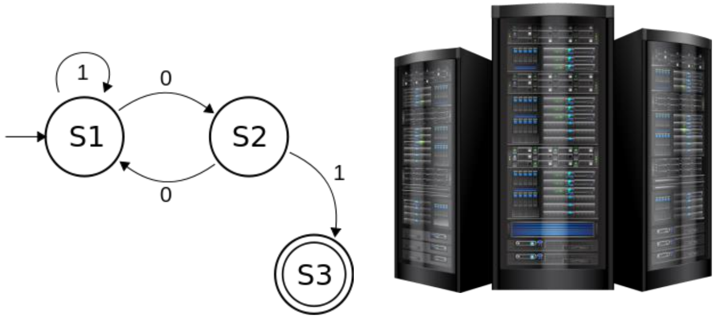
```
S1:
while (moreGoods == 1)
   DeliverGoods();
S2:
if (allOrderComplete == 0) goto S1;
else {
   S3:
   printf("Goods transfer complete");
}
```

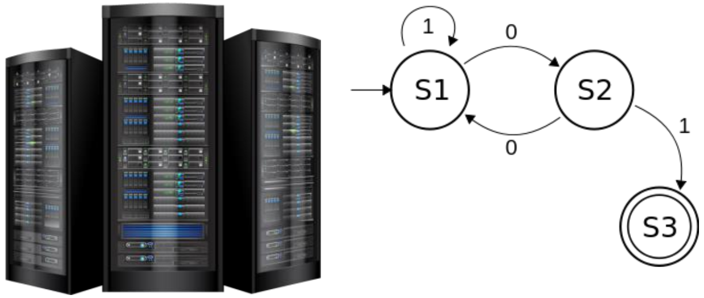


State Machine Replication

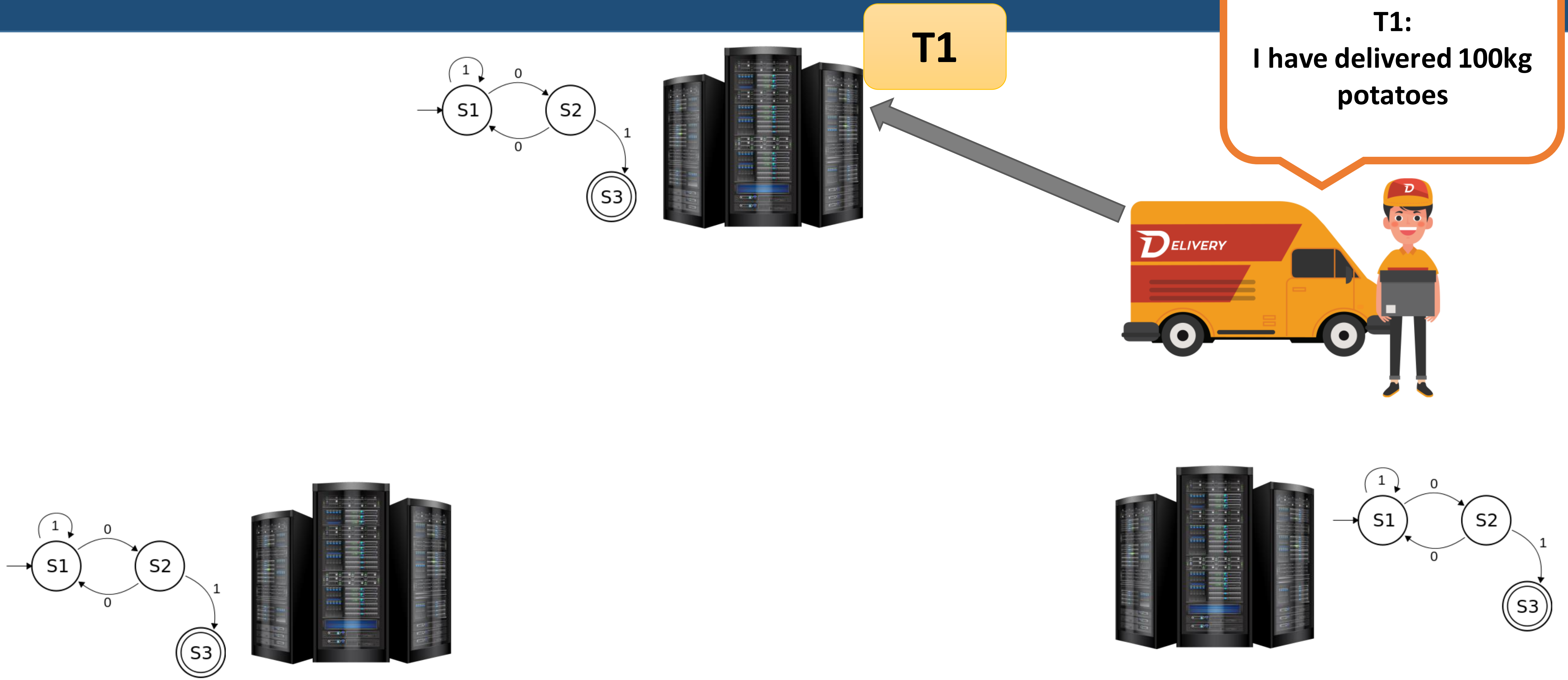
Replicate the state machine on multiple independent servers





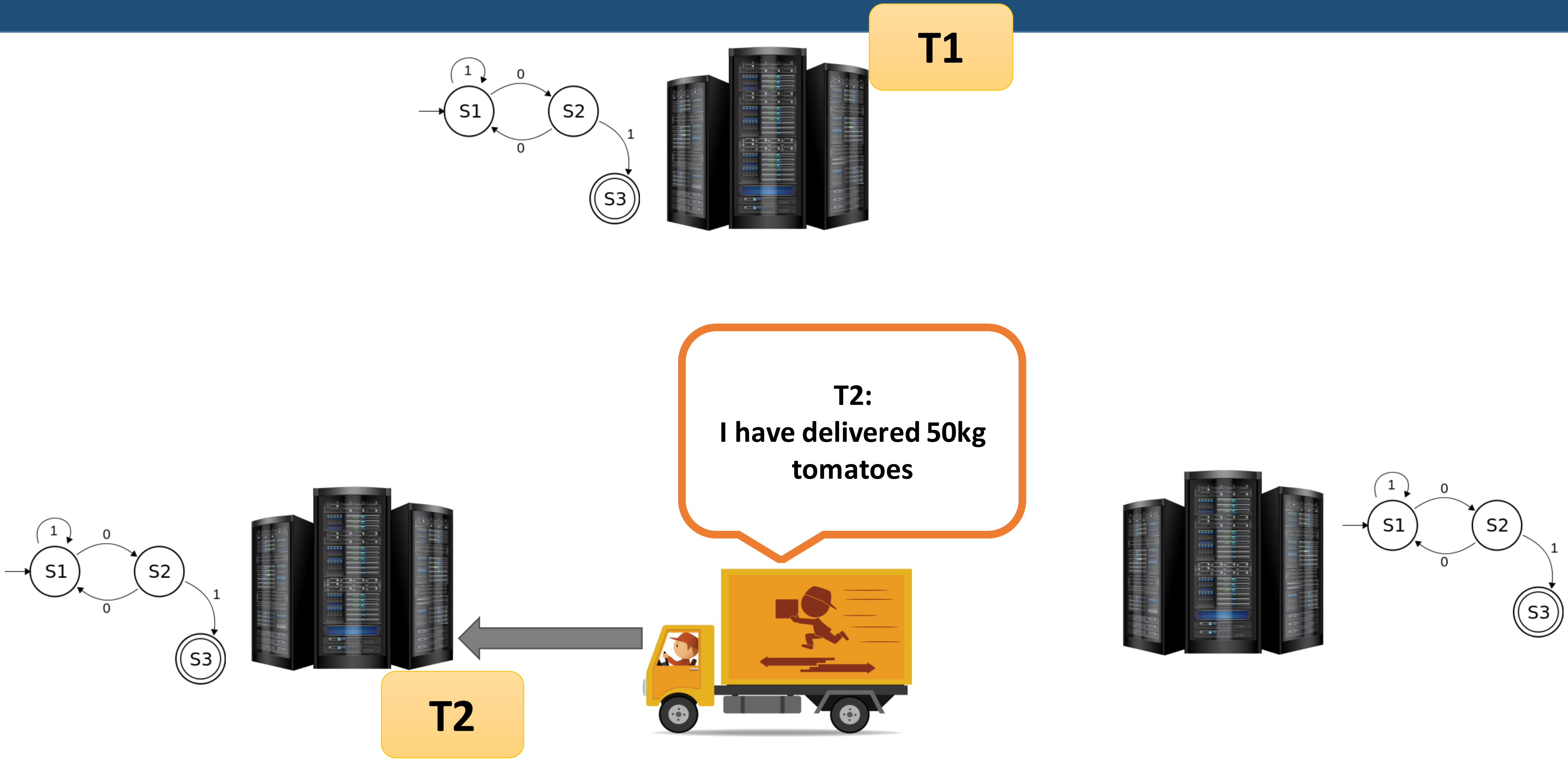


State Machine Replication



T1:
I have delivered 100kg potatoes

State Machine Replication

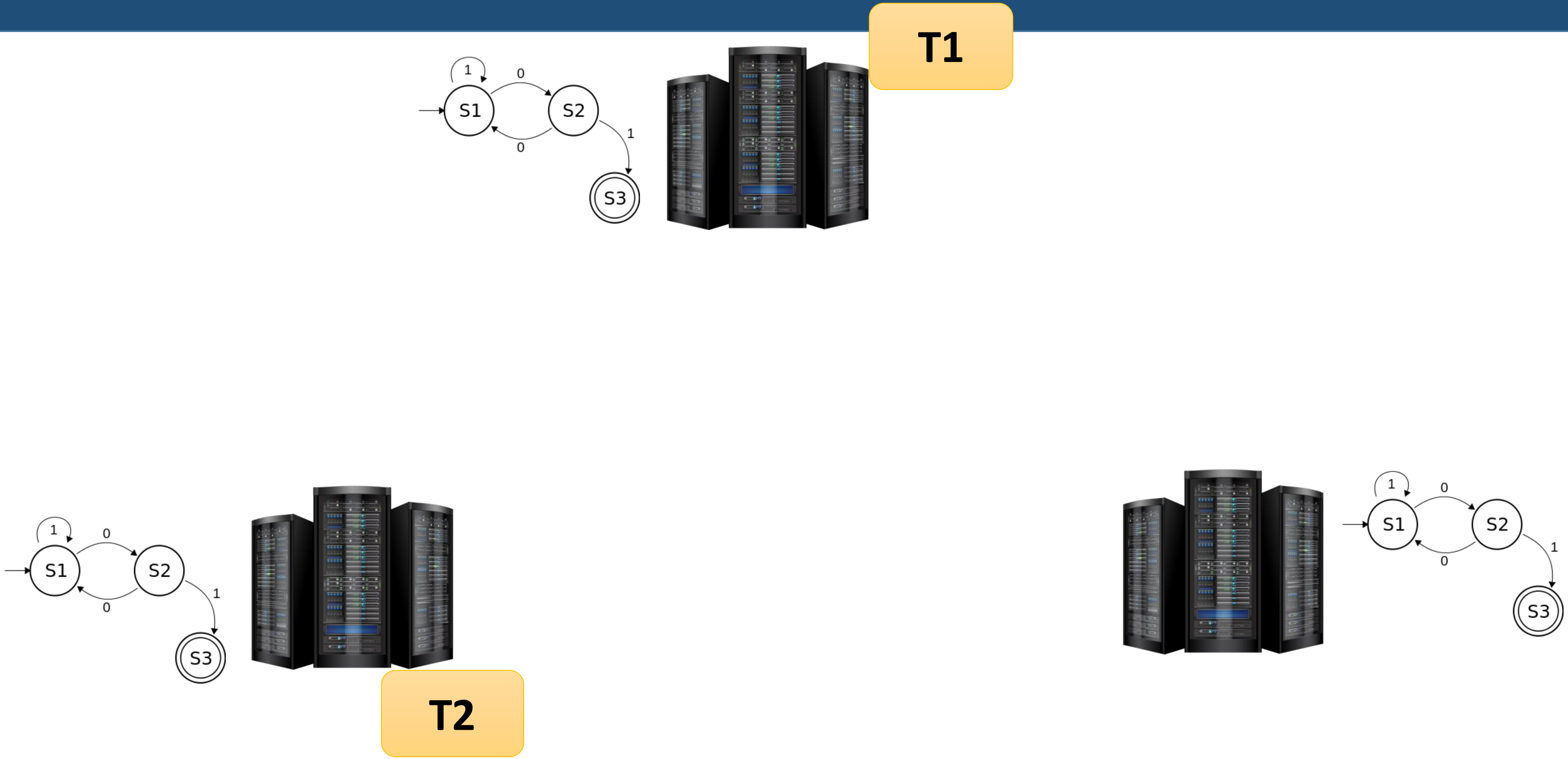


T1

T2:
I have delivered 50kg tomatoes

T2

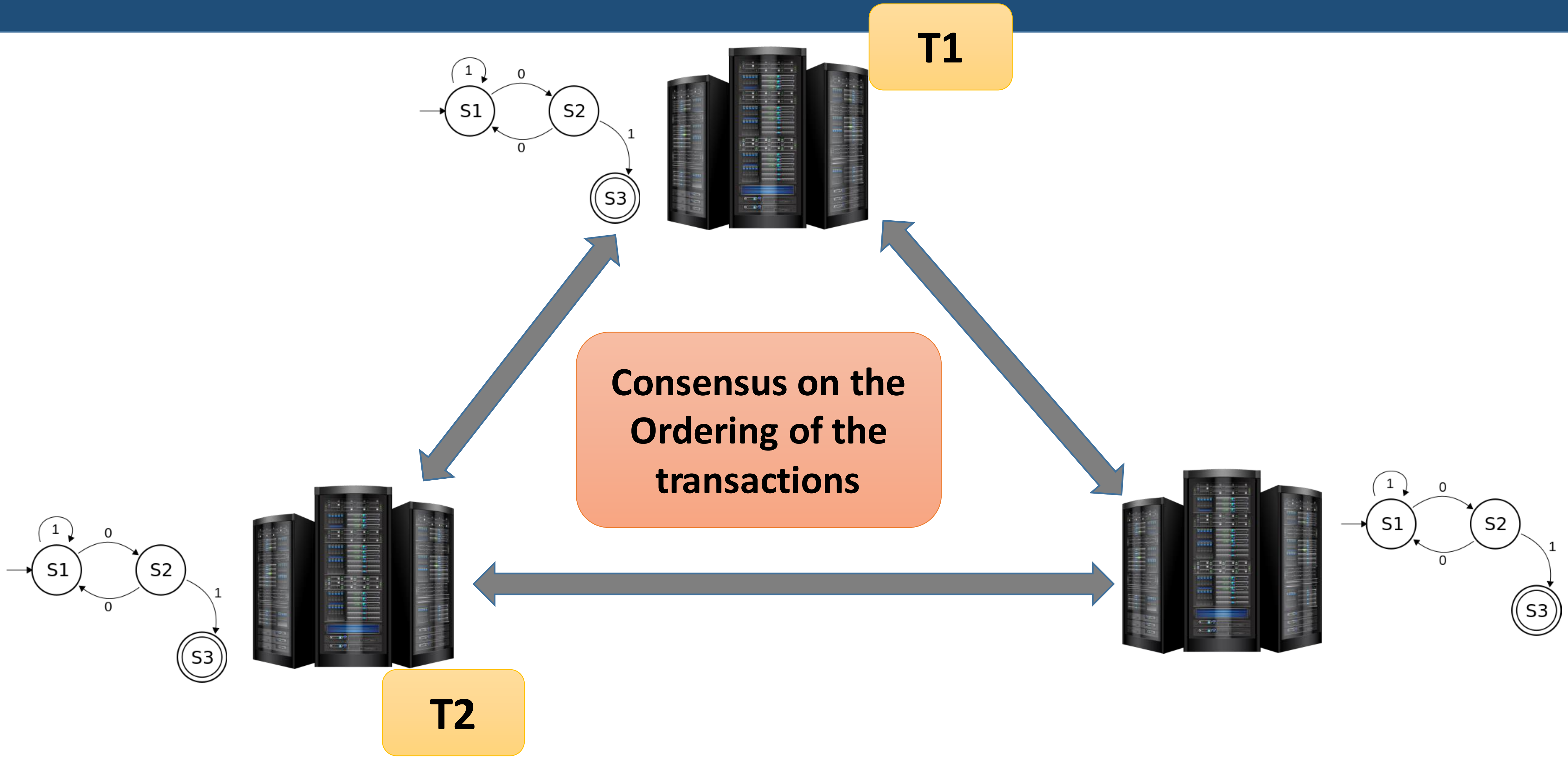
State Machine Replication



T1

T2

State Machine Replication

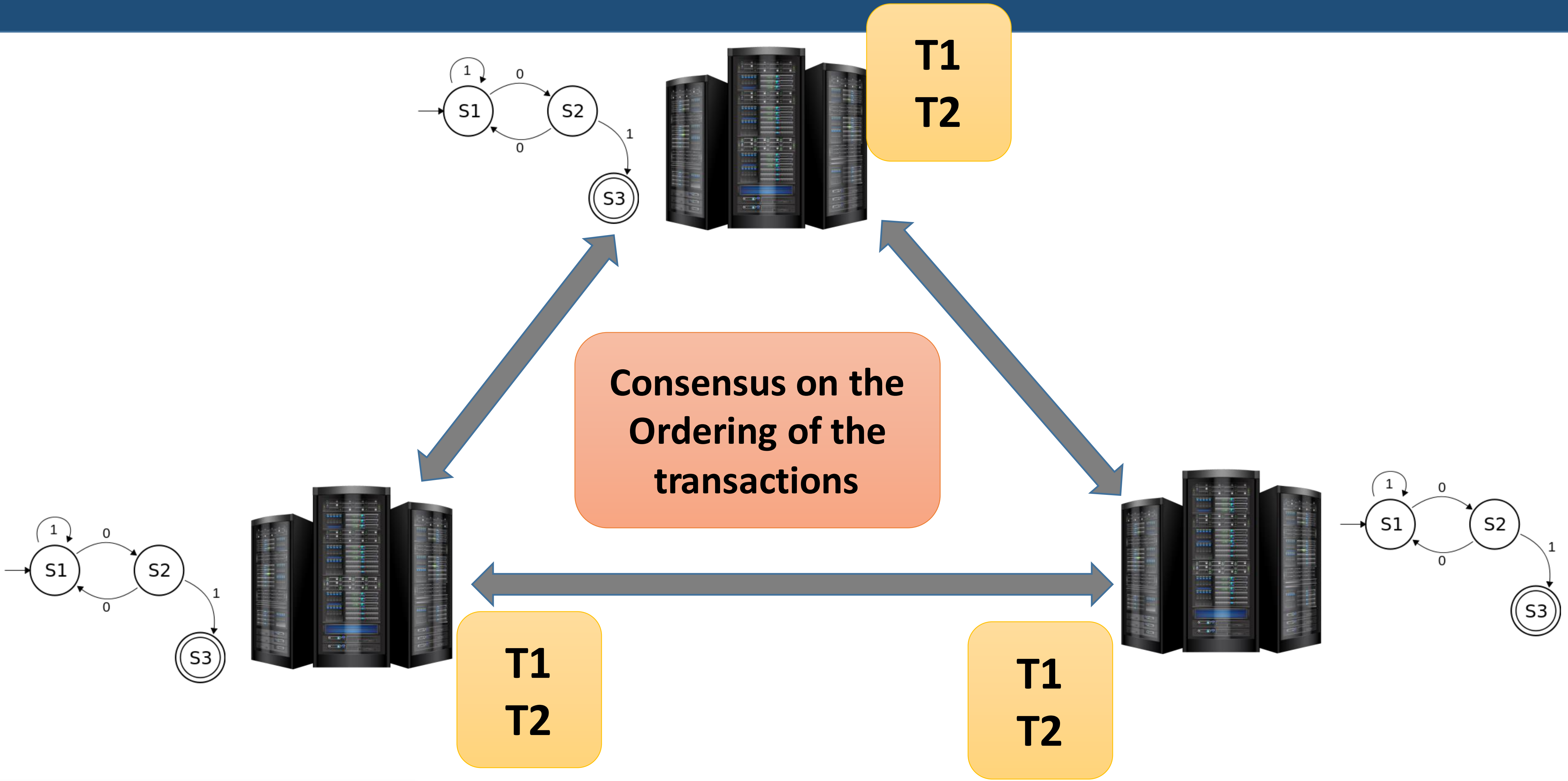


T1

Consensus on the Ordering of the transactions

T2

State Machine Replication



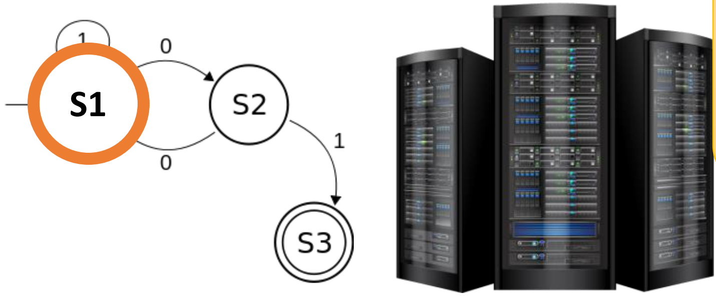
T1
T2

Consensus on the Ordering of the transactions

T1
T2

T1
T2

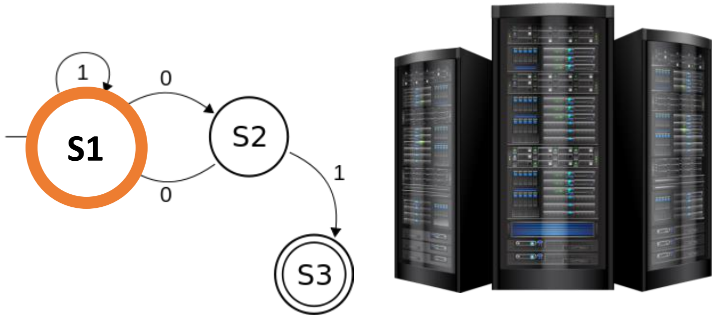
State Machine Replication



T1

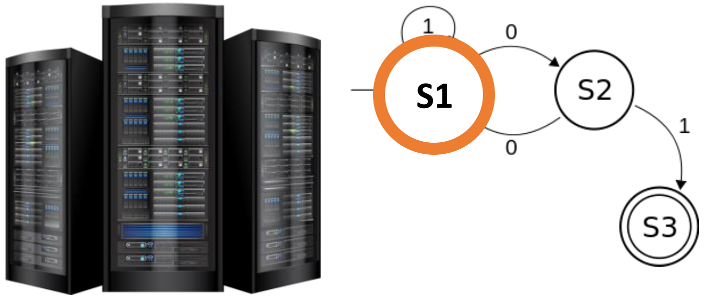
T2

Independently execute the transactions



T1

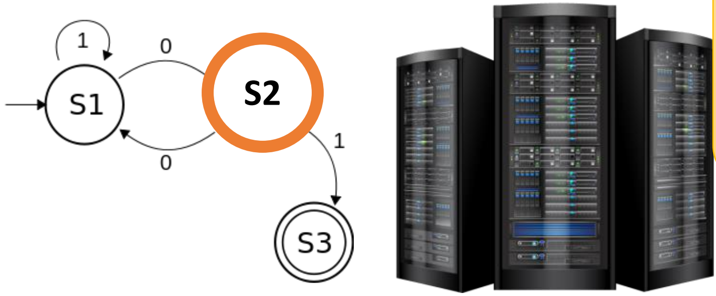
T2



T1

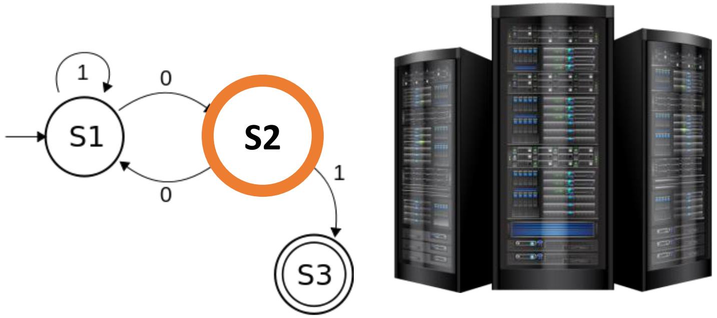
T2

State Machine Replication

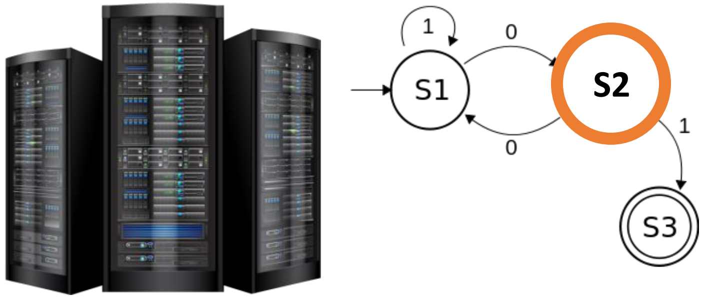


T1
T2

Independently execute the transactions



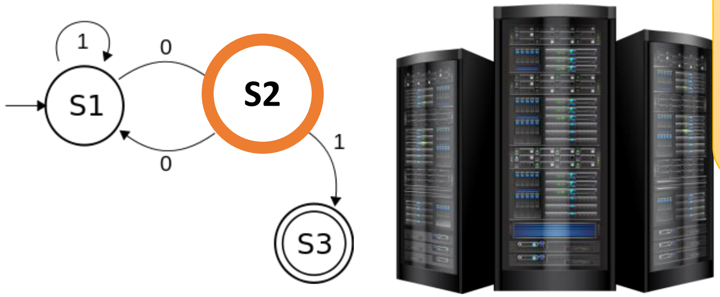
T1
T2



T1
T2

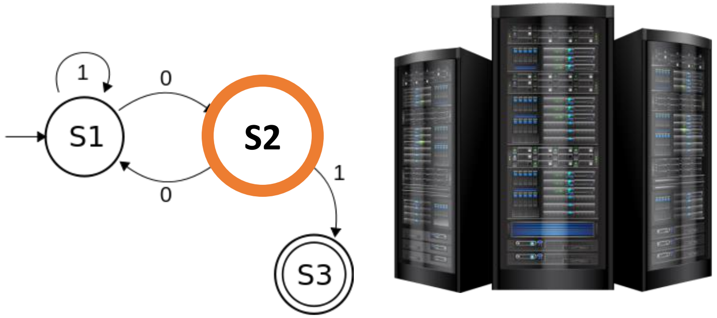
State Machine Replication

More orders?
Yes

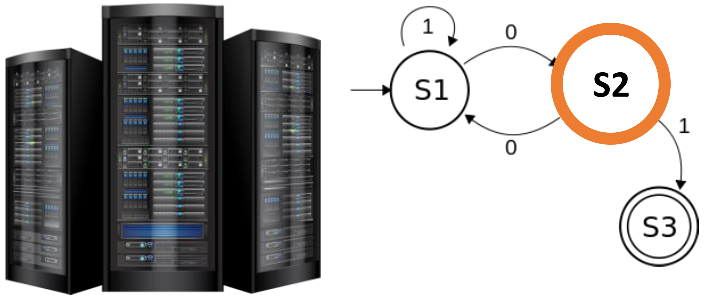


T1
T2

Independently execute the transactions



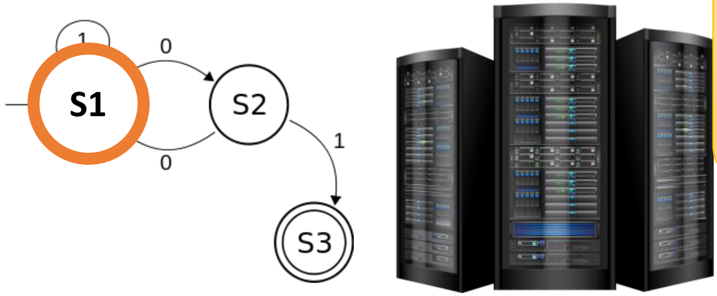
T1
T2



T1
T2

State Machine Replication

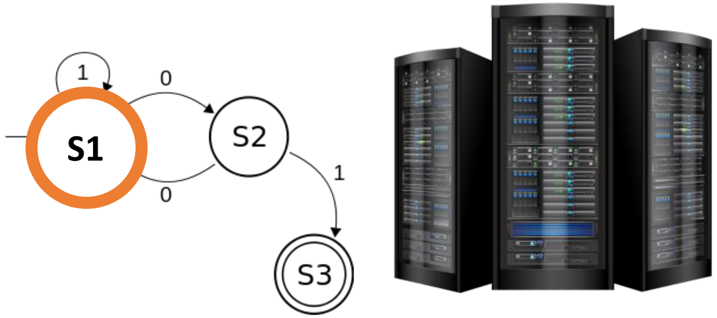
Execution of T1 completes



T1

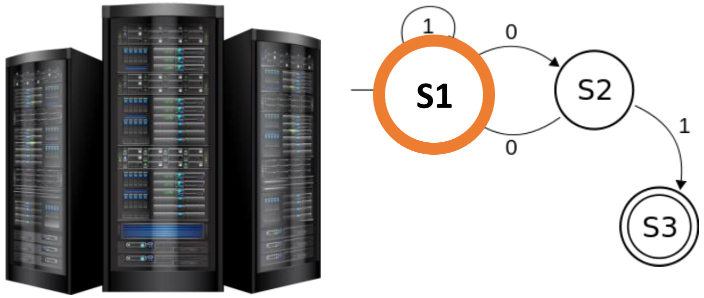
T2

Independently execute the transactions



T1

T2

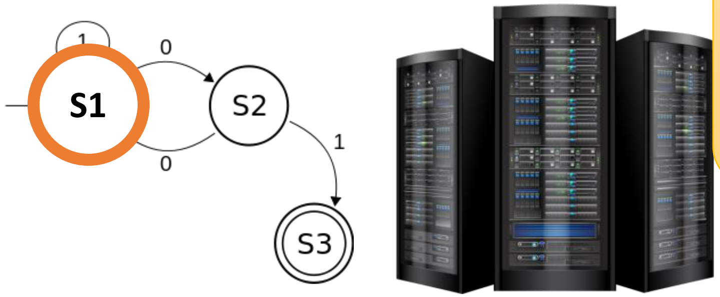


T1

T2

State Machine Replication

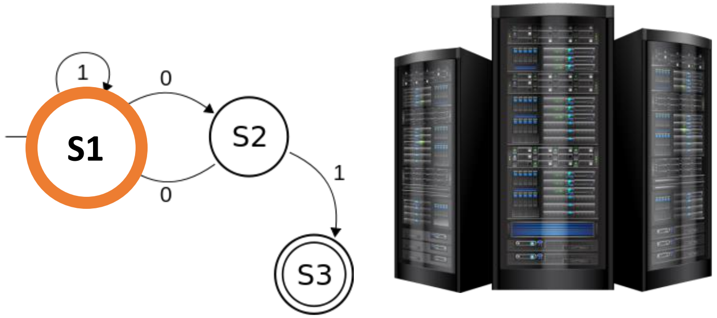
Start executing T2



T1

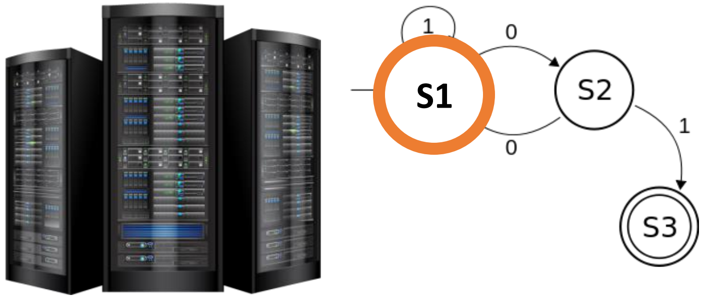
T2

Independently execute the transactions



T1

T2

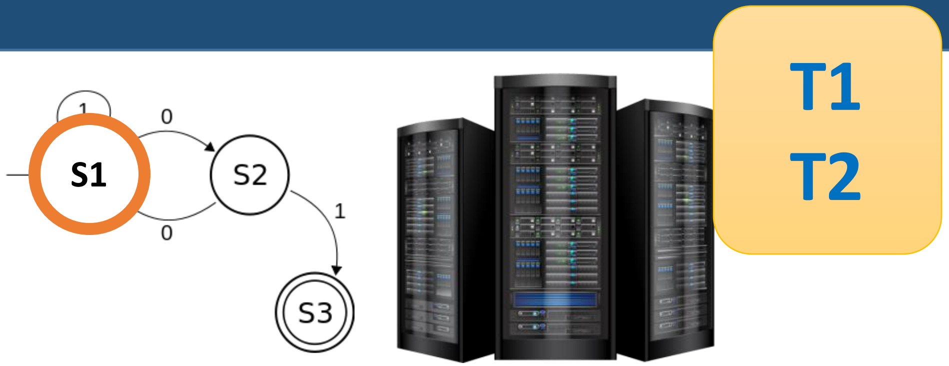


T1

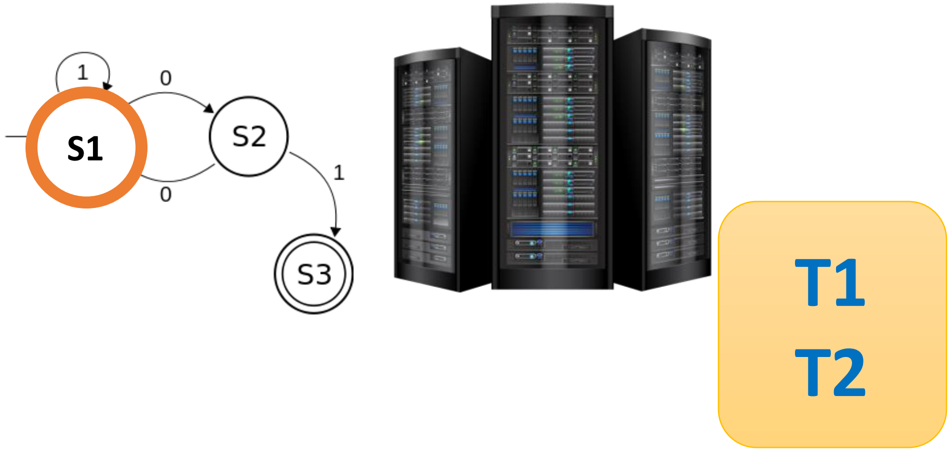
T2

State Machine Replication

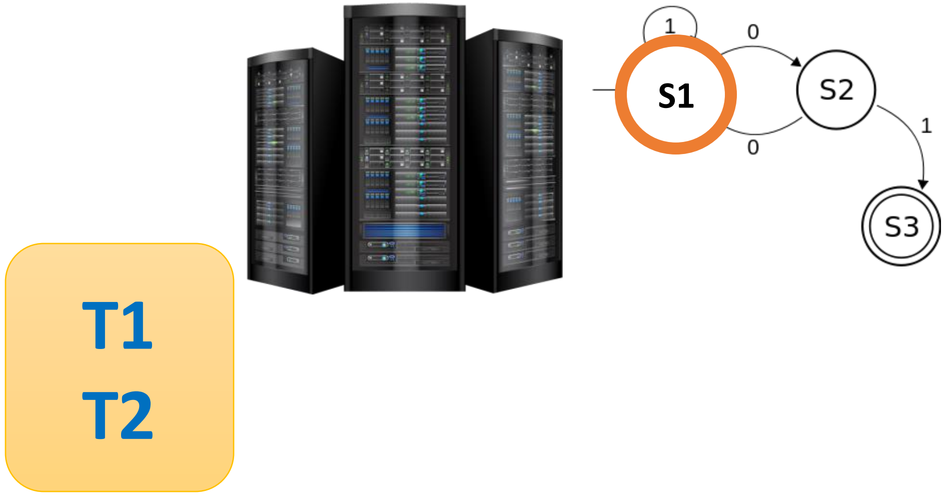
Complete execution



T1
T2

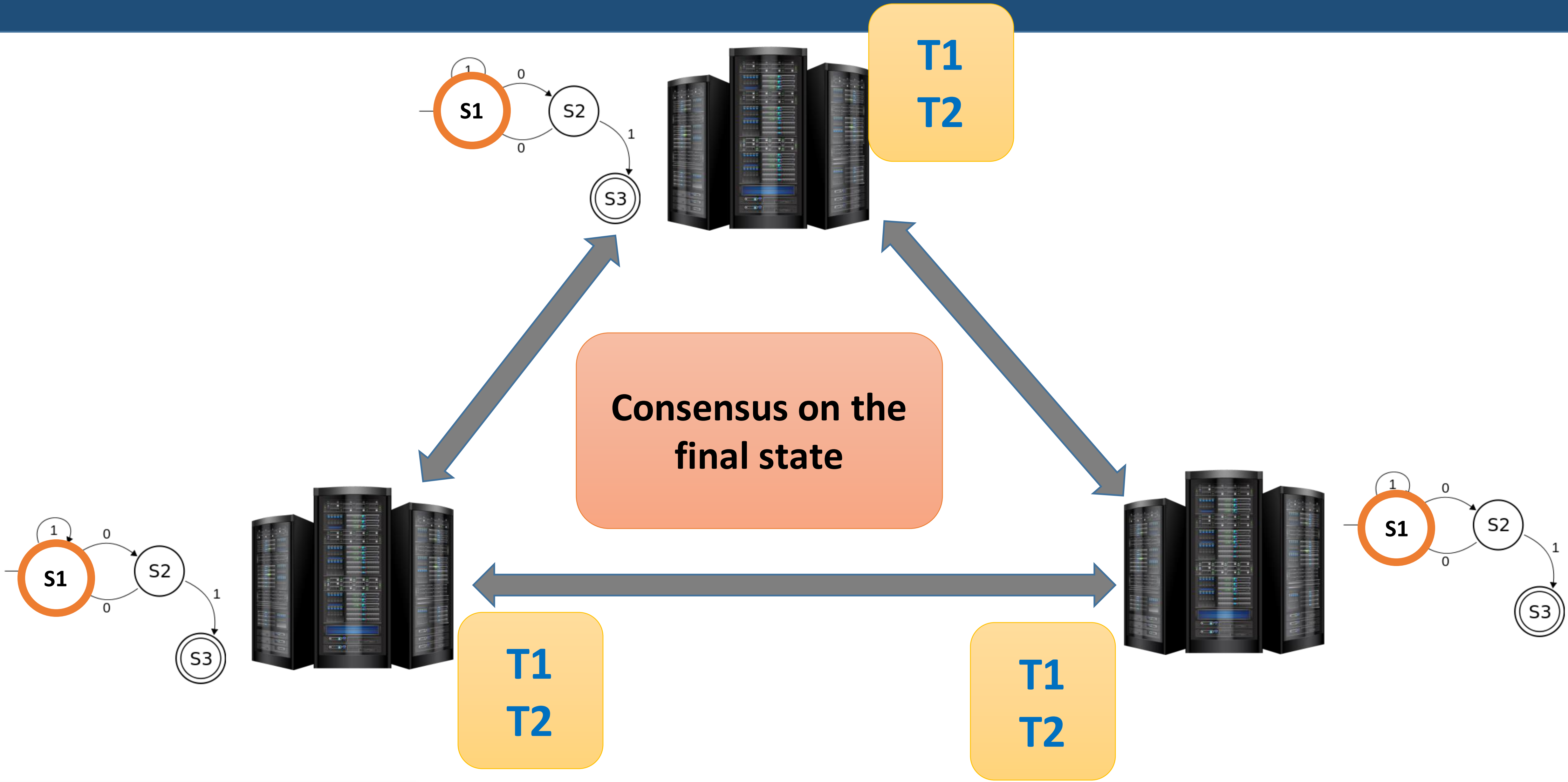


T1
T2

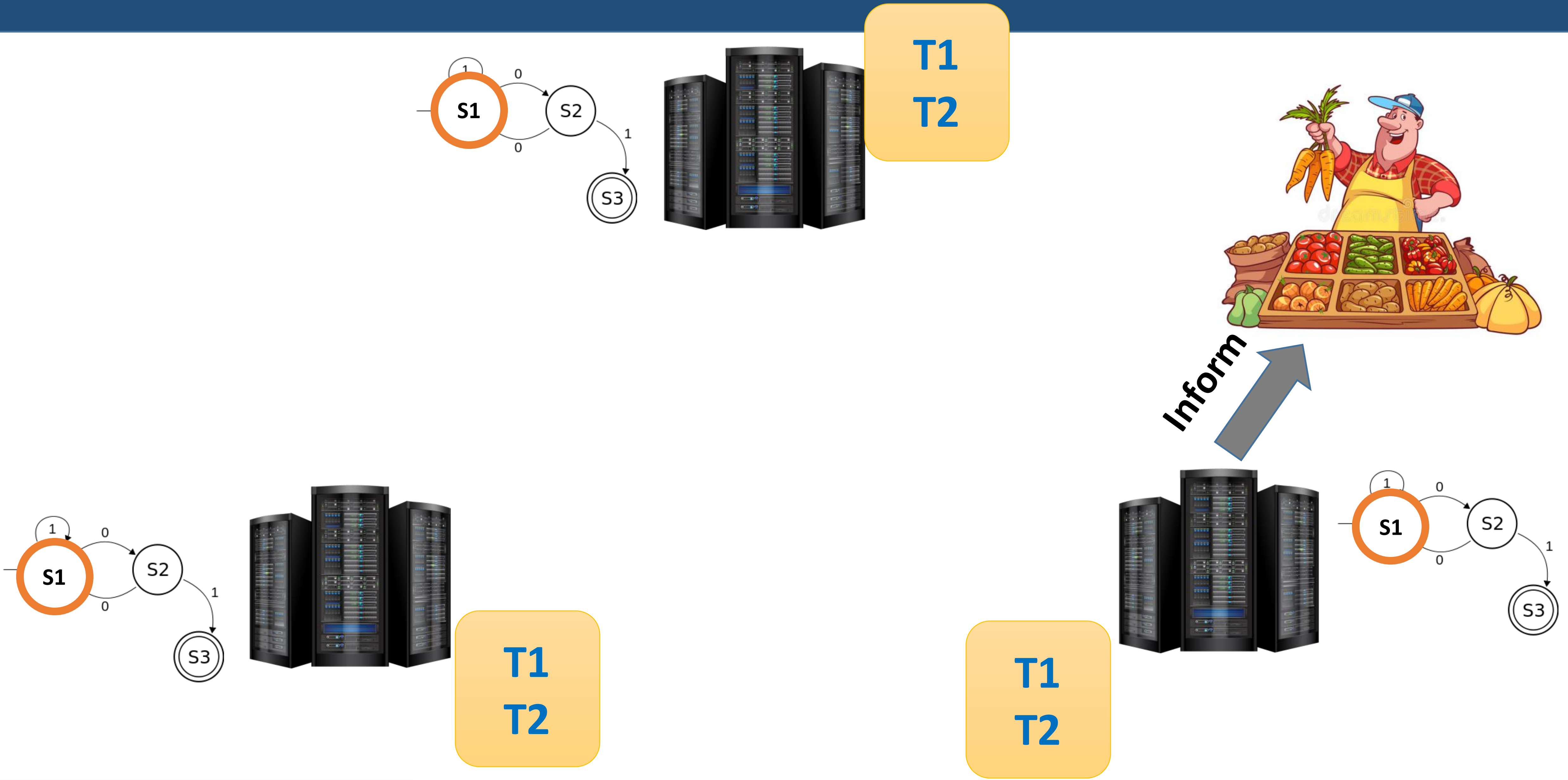


T1
T2

State Machine Replication



State Machine Replication

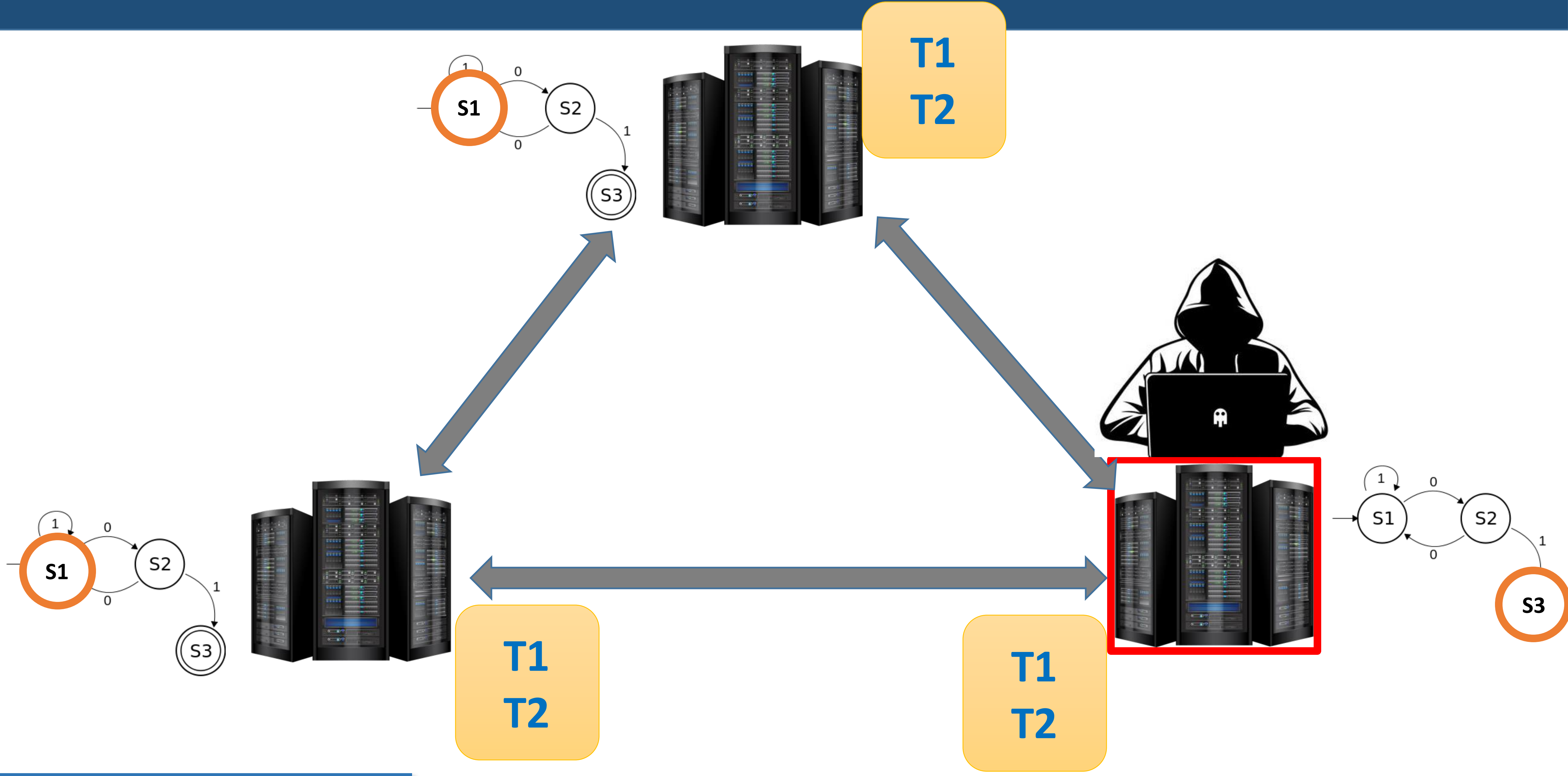


T1
T2

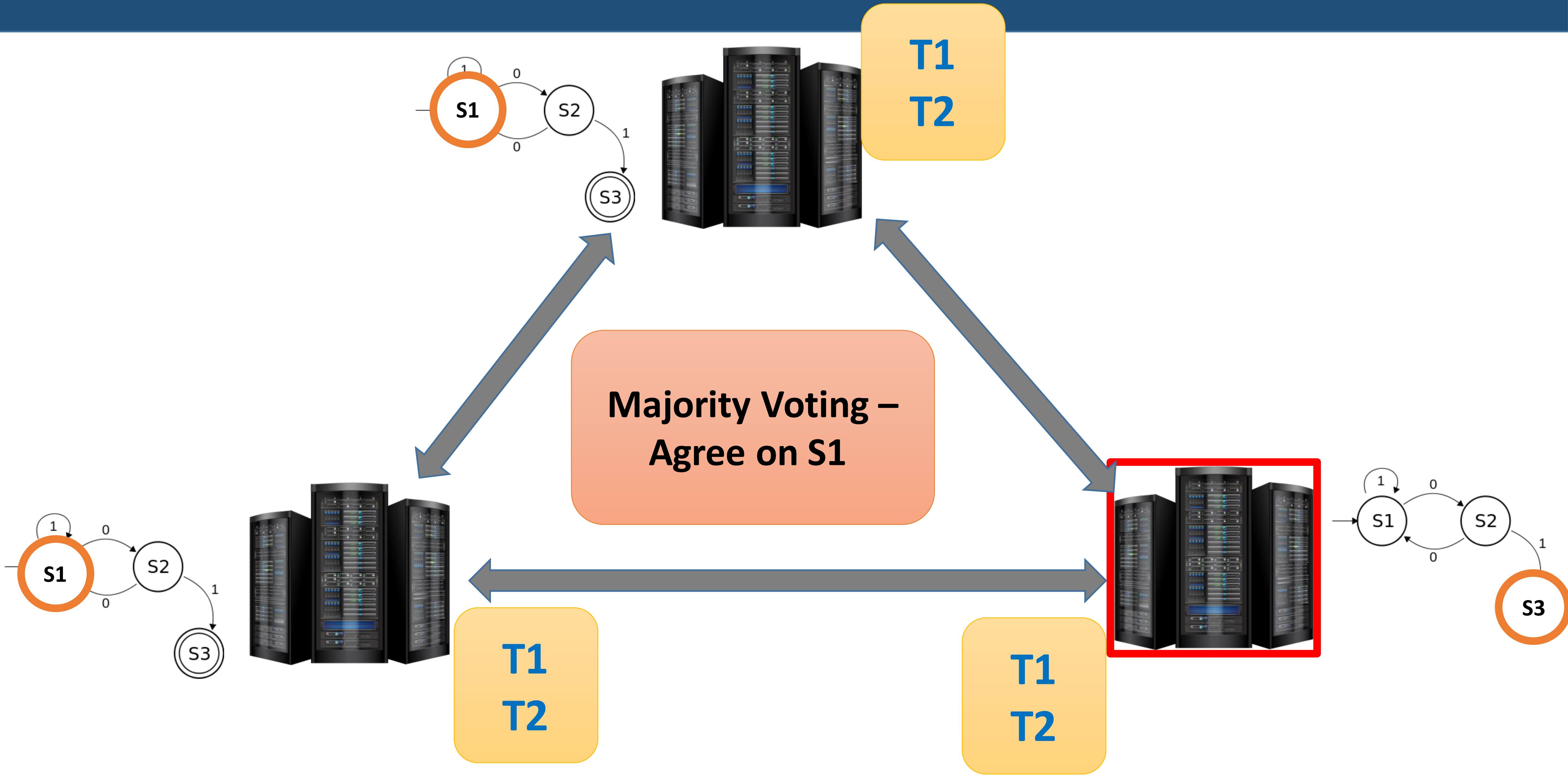
T1
T2

T1
T2

State Machine Replication – Why do we need Consensus?



State Machine Replication – Why do we need Consensus?



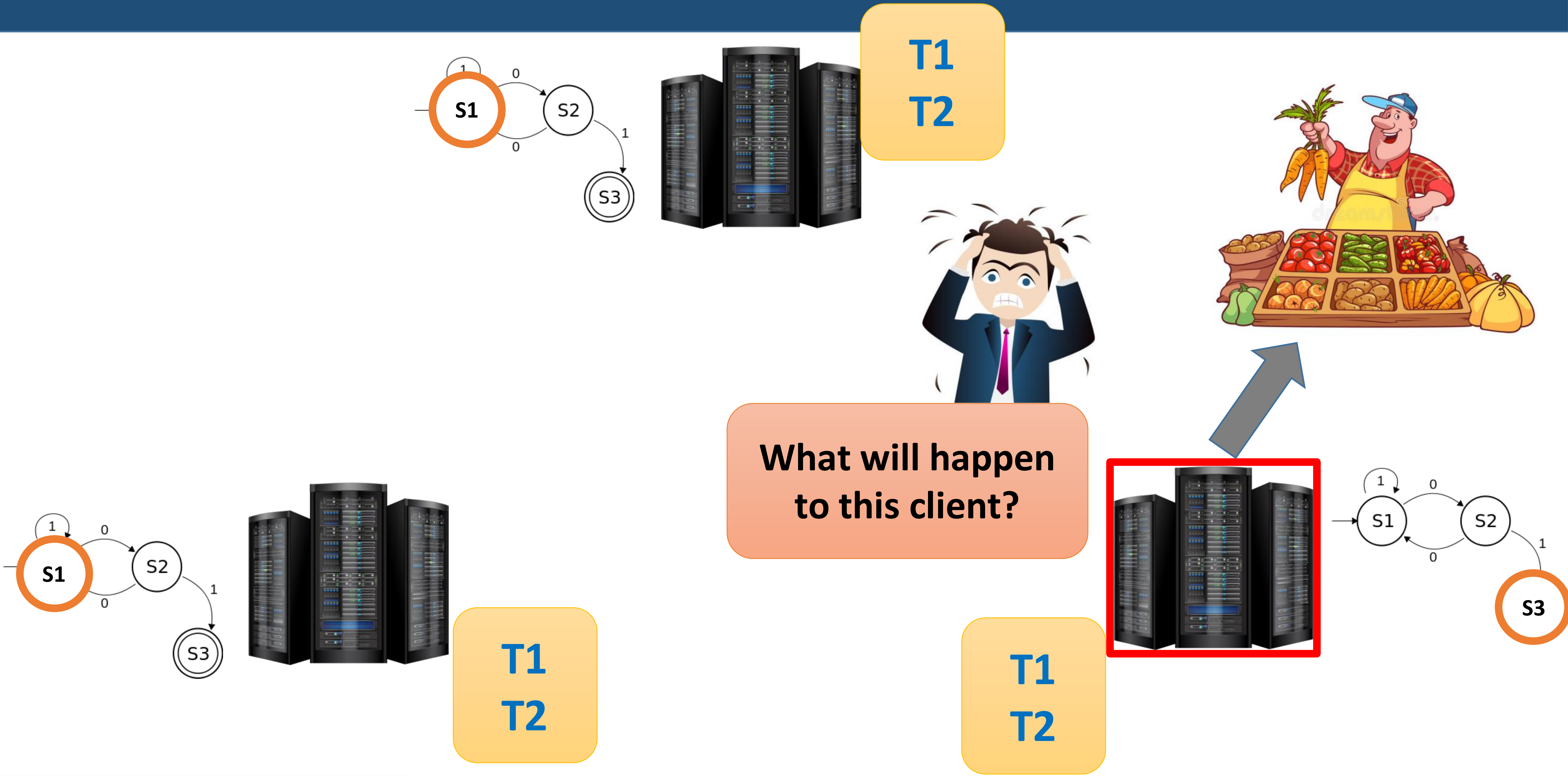
T1
T2

Majority Voting – Agree on S1

T1
T2

T1
T2

State Machine Replication – Why do we need Consensus?



T1

T2

T1

T2

What will happen to this client?

T1

T2

Permissioned Blockchain and State Machine Replication

- There is a natural reason to use state machine replication-based consensus over permissioned blockchains
  - The network is closed, the nodes know each other, so state replication is possible among the known nodes
  - Avoid the overhead of mining - do not need to spend anything (like power, time, bitcoin) other than message passing
  - However, consensus is still required - machines can be faulty or behave maliciously

Permissioned Blockchain and State Machine Replication

- There is a na replication-based consensus over permis
  - The netw , so state replication is possible among th
  - Avoid the nd anything (like power, time, bitcoin) ot
  - However, n be faulty or behave maliciously

But, we need a bit redesign !

Permissioned Blockchain and State Machine Replication

- There is a na e replication-based consensus over permis
  - The netw , so state replication is possible among th
  - Avoid the end anything (like power, time, bitcoin) ot
  - However, n be faulty or behave maliciously

But, we need a bit redesign !

Crypto is the saver

Crypto + Distributed Consensus = Consensus for Permissioned Blockchain

Permissioned Blockchain and State Machine Replication

- There is a natural reason to use state machine replication-based consensus over permissioned blockchains
  - The network is closed, the nodes know each other, so state replication is possible among the known nodes
  - Avoid the overhead of mining - do not need to spend anything (like power, time, bitcoin) other than message passing
  - However, consensus is still required - machines can be faulty or behave maliciously

- Classical Distributed Consensus Algorithms (Paxos, RAFT, Byzantine Agreement) are based on State Machine Replication
  - Let us (re)visit those algoithms

Faults in a Distributed System

- **Crash Faults**: The node stops operating – hardware or software faults
  - In an asynchronous system: You do not know whether messages have been delayed or the node is not responding
  - Rely on majority voting – progress as and when you have received the confirmation from the majority
  - Propagation of the consensus information – nodes on a slow network will receive it eventually

Faults in a Distributed System

- **Crash Faults:** The node stops operating – hardware or software faults
  - In an asynchronous system: You do not know whether messages have been delayed or the node is not responding
  - Rely on majority voting – progress as and when you have received the confirmation from the majority
  - Propagation of the consensus information – nodes on a slow netwok will receive it eventually

- **Byzantine Faults:** Nodes misbehave – send different information to different peers (partition the network)
  - More difficult to handle
  - More suitable for blockchains

Asynchronous Consensus with Crash Faults

- Remember the **FLP Impossibility**
  - Give priority to safety over liveness

- Guarantees the followings --
  - **Validity**: If all correct process proposes the same value v, then any correct process decides v
  - **Agreement:** No two correct processes decide differently
  - **Termination**: Every correct process eventually decides

Asynchronous Consensus with Crash Faults

- Remember the **FLP Impossibility**
  - Give priority to safety over liveness

- Guarantees the followings --
  - **Validity**: If all correct process proposes the same value v, then any correct process decides v **(Unlikely to happen in PoW)**
  - **Agreement:** No two correct processes decide differently **(Safety – Not in PoW)**
  - **Termination**: Every correct process eventually decides **(Liveness – Priority in PoW)**

Asynchronous Consensus with Crash Faults

- Remember the **FLP Impossibility**
  - Give priority to safety over liveness

- Guarantees the followings --
  - **Validity**: If all correct process proposes the same value v, then any correct process decides v
  - **Agreement:** No two correct processes decide differently
  - **Termination**: Every correct process eventually decides

- CFT Consensus
  - Paxos (Proposed by Lamport, the most fundamental CFT) -- used in DynamoDB
  - RAFT (Much simpler than Paxos) -- Used in **Fabric Transaction Ordering**

thank you!