# Permissioned Model

- A blockchain architecture where users are authenticated apriory
- Users know each other
- However, users may not trust each other – Security and consensus are still required.
- Run blockchain among known and identified participants

# Design Limitations

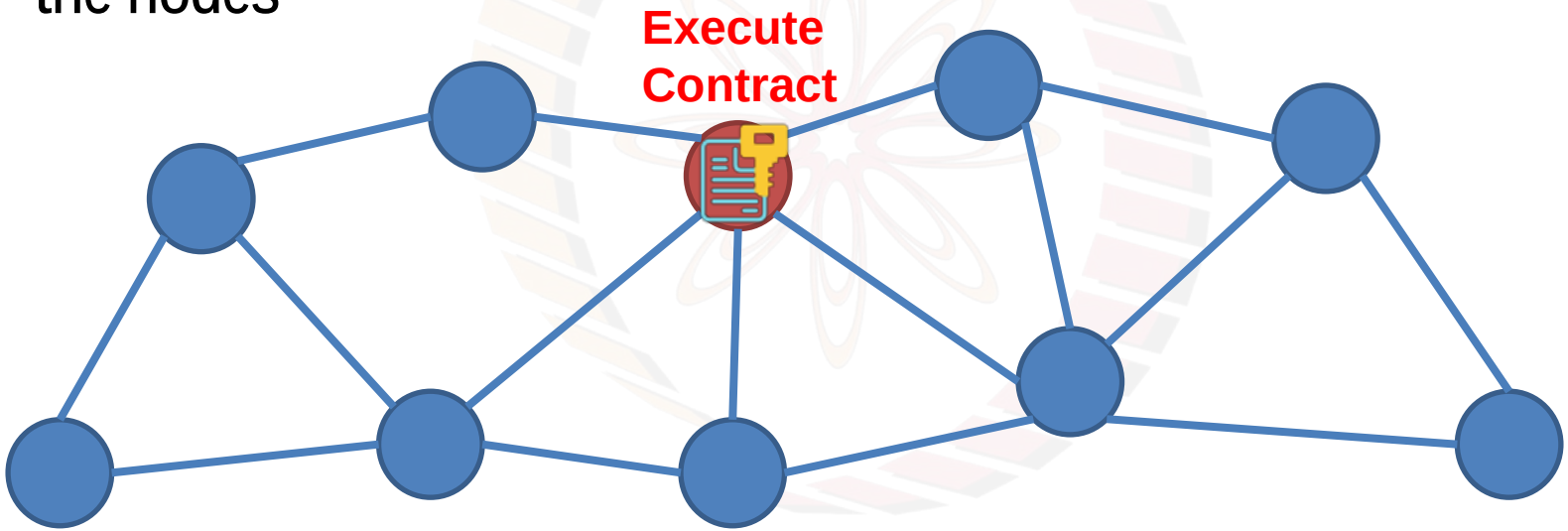
- Sequential Execution
  - Execute transactions sequentially based on consensus
  - Requests to the application (smart contract) are ordered by the consensus, and executed in the same order
  - This give a bound on the effective throughput – throughput is inversely proportional
  - Can be a possible attack on the smart contract platform – introduce contract which will take long time to execute

# Design Limitations

- Non-deterministic Execution
  - Smart-contract execution should always needs to be deterministic; otherwise the system may lead to inconsistent states (many fork in the blockchain)
  - Solution: Domain specific language (DSL) for smart contract

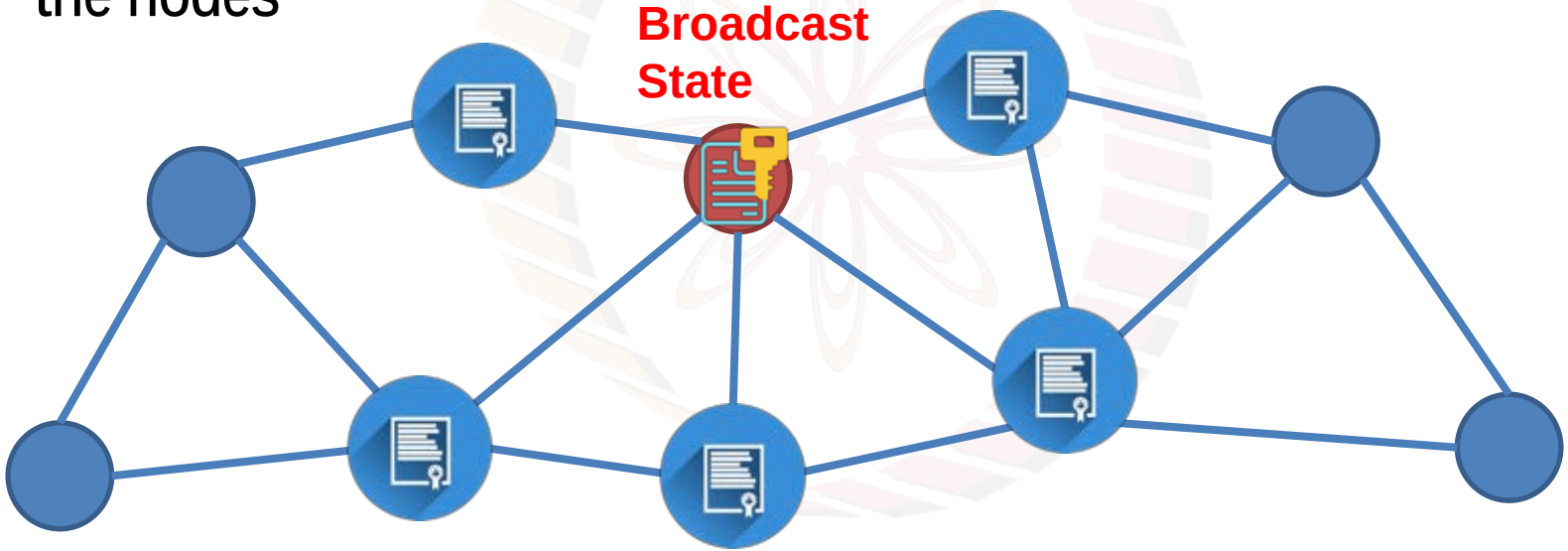
# Do We Really Need to Execute Contracts at Each Node

- Not necessary always, we just need state synchronization across all the nodes



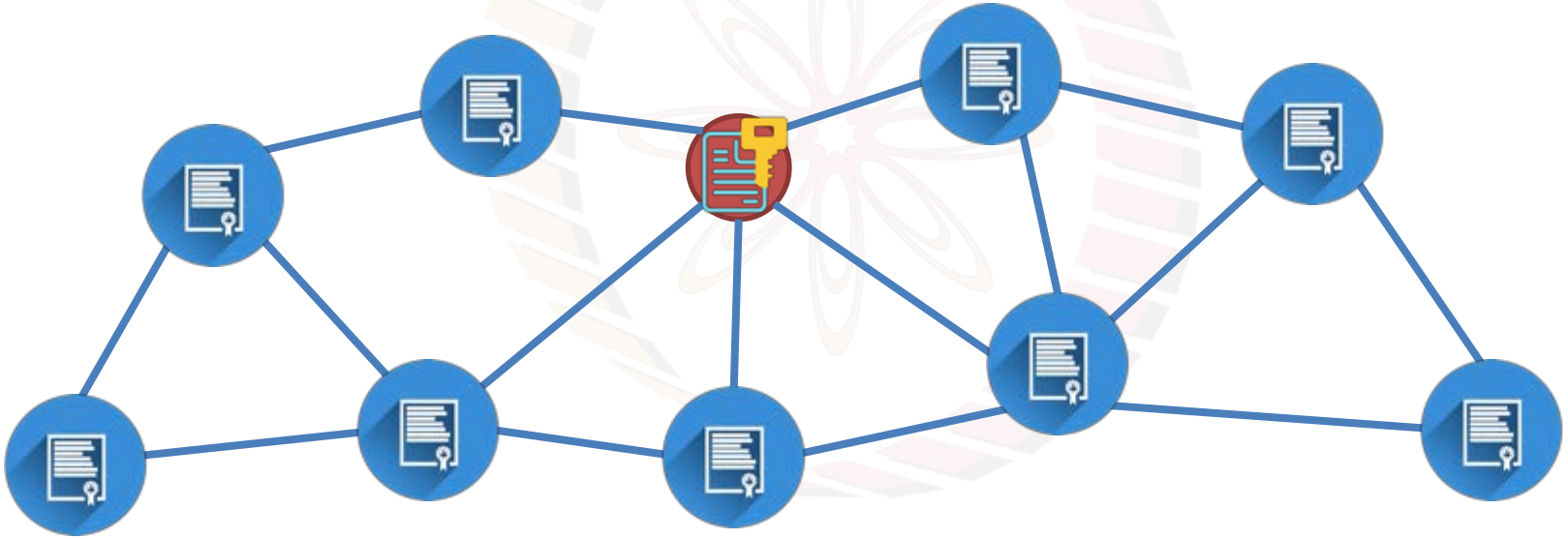
# Do We Really Need to Execute Contracts at Each Node

- Not necessary always, we just need state synchronization across all the nodes



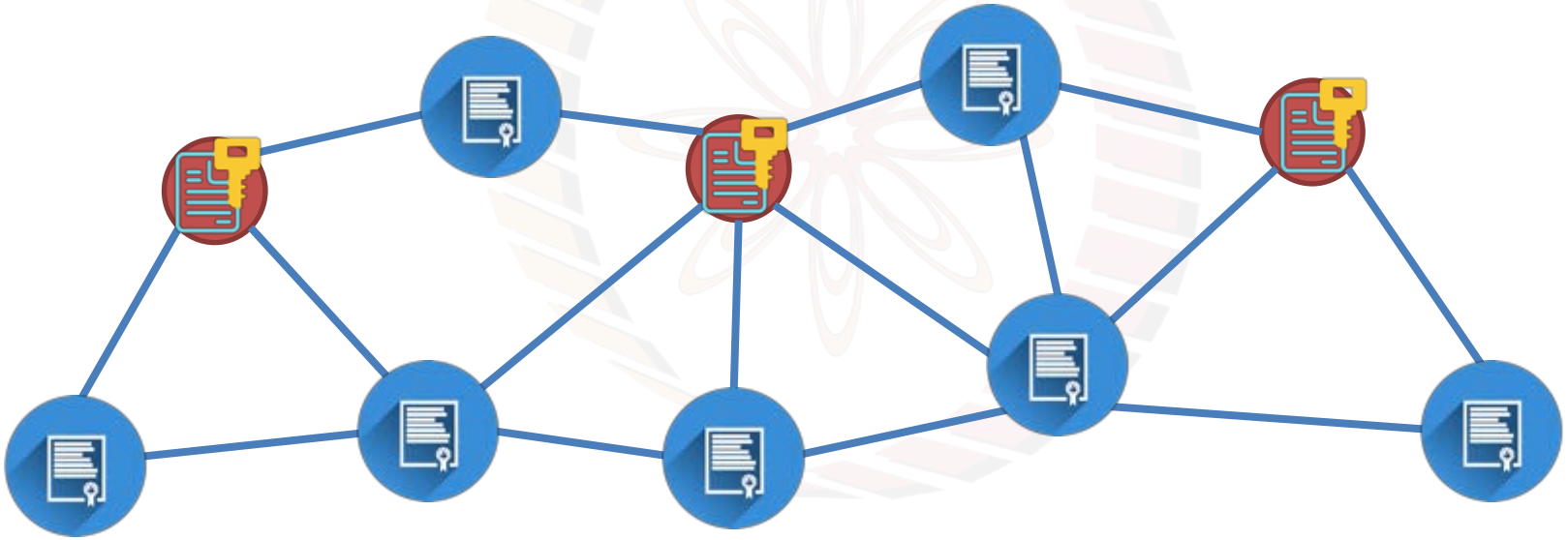
# Do We Really Need to Execute Contracts at Each Node

- What if the node that executes the contract is faulty?

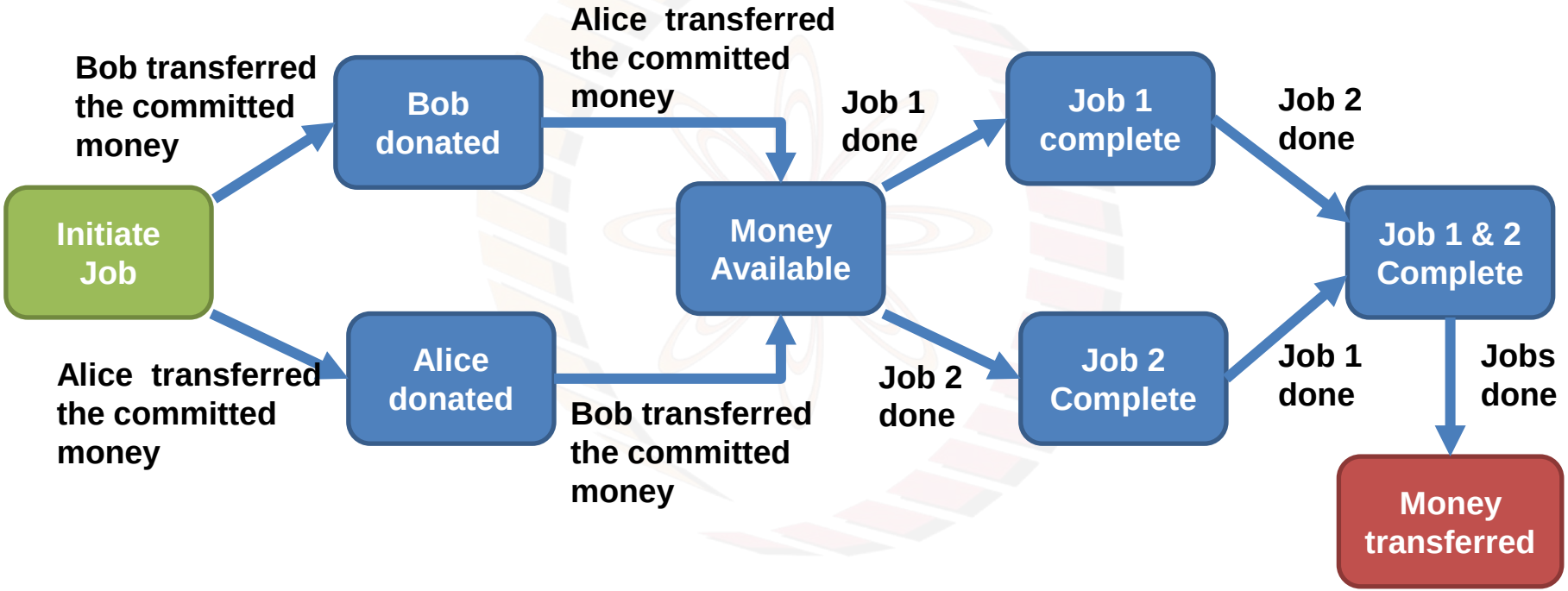


# Do We Really Need to Execute Contracts at Each Node

- Use state machine replication – execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes

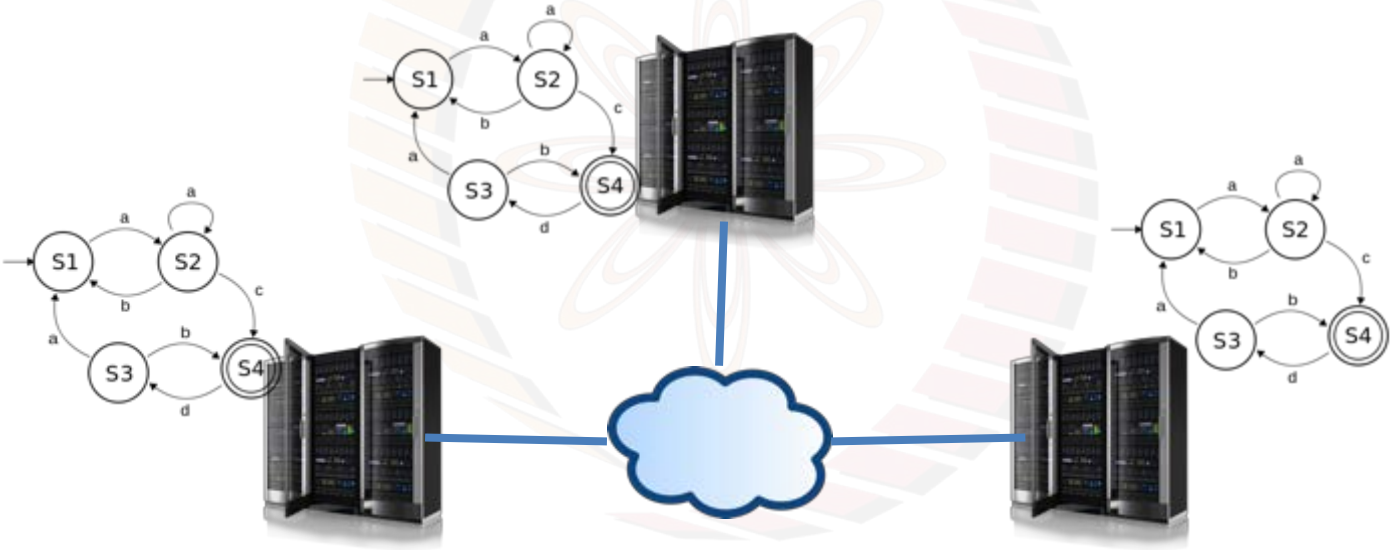


# Smart Contract State Machine - Crowd-Funding



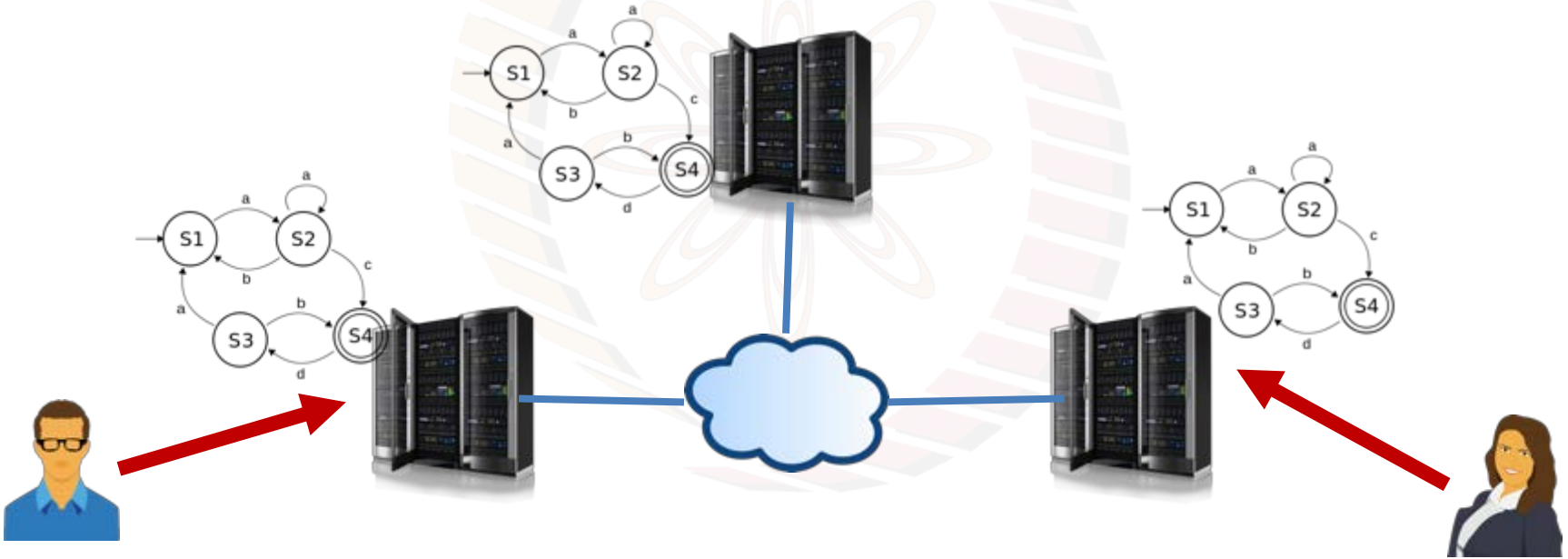
# Distributed State Machine Replication

1. Place copies of the state machine on multiple independent servers



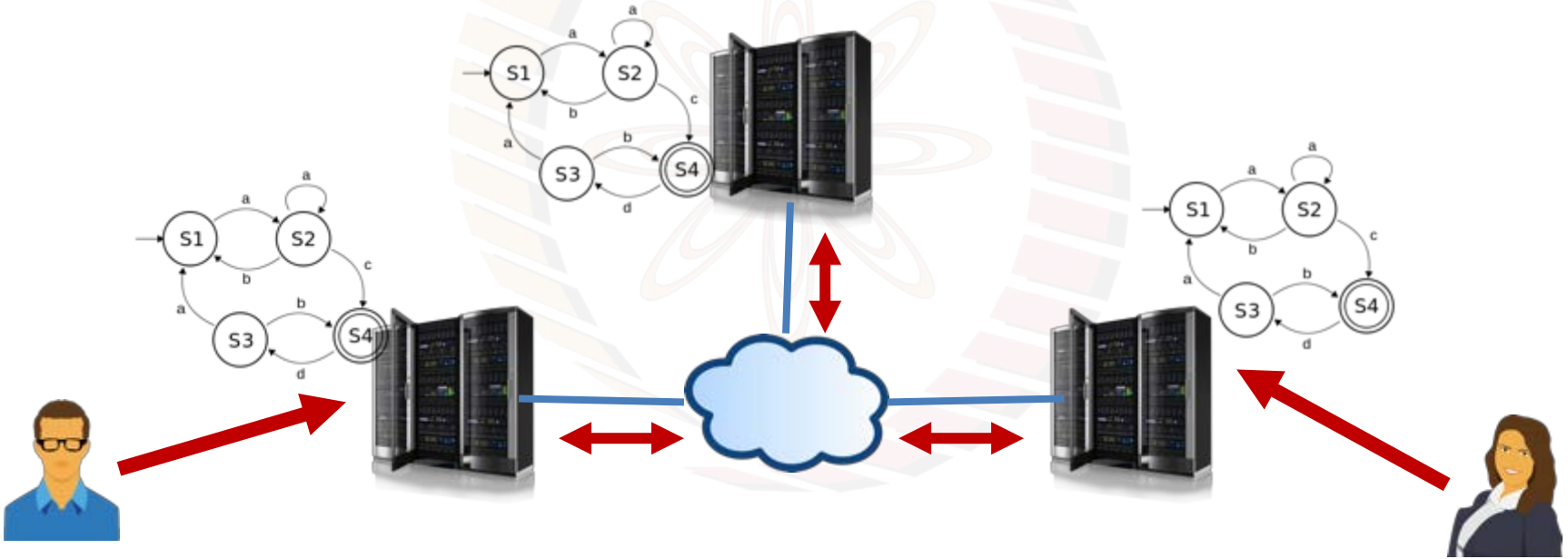
# Distributed State Machine Replication

2. Receive client requests, as an input to the state machine



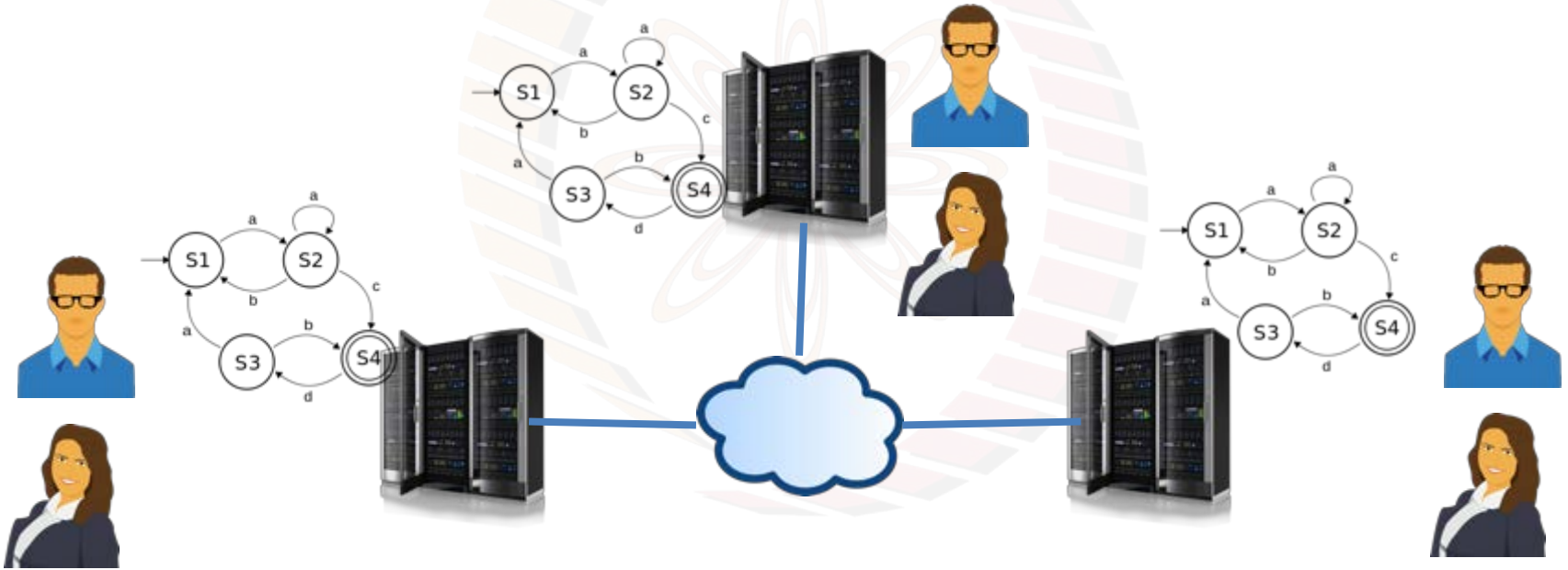
# Distributed State Machine Replication

3. Propagate the inputs to all the servers



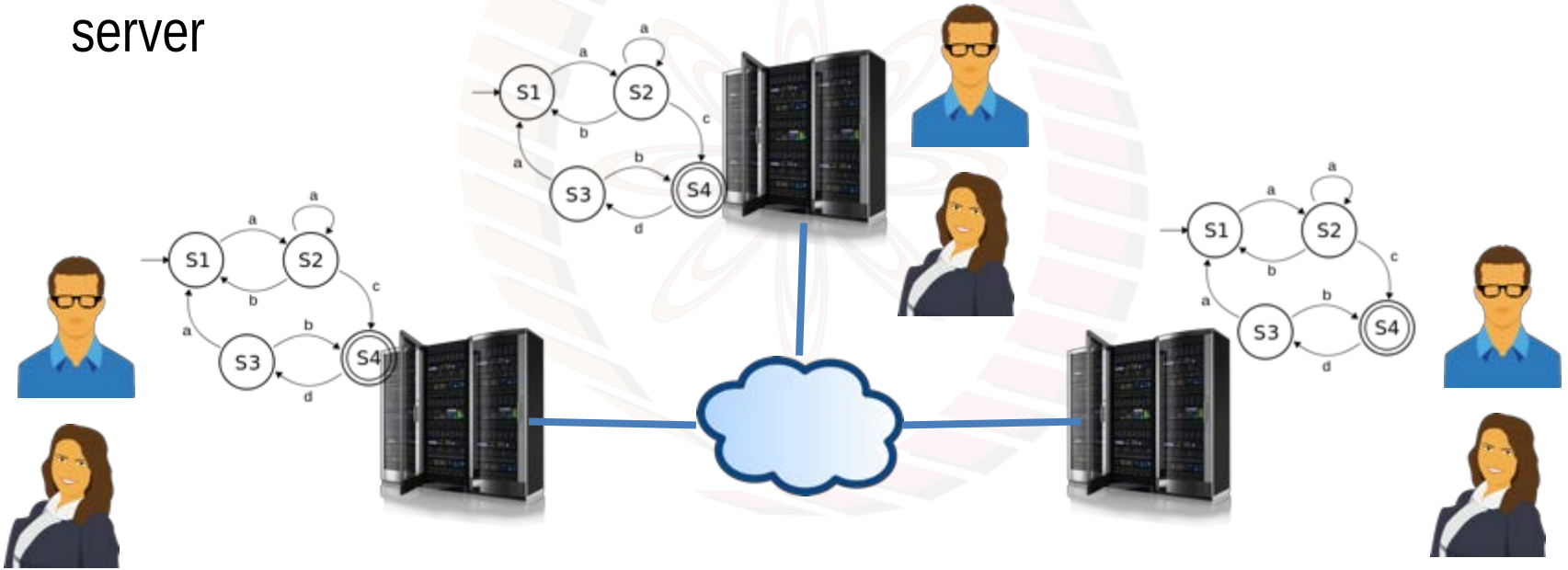
# Distributed State Machine Replication

4. Order the inputs based on some ordering algorithm



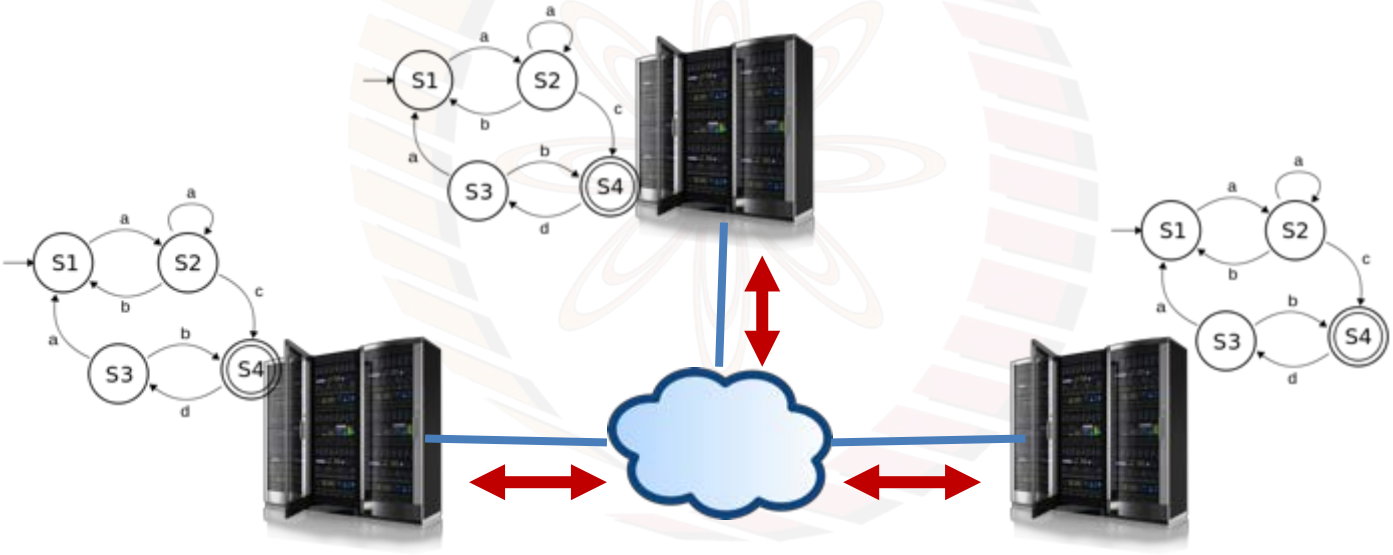
# Distributed State Machine Replication

4. Execute the inputs based on the order decided, individually at each server



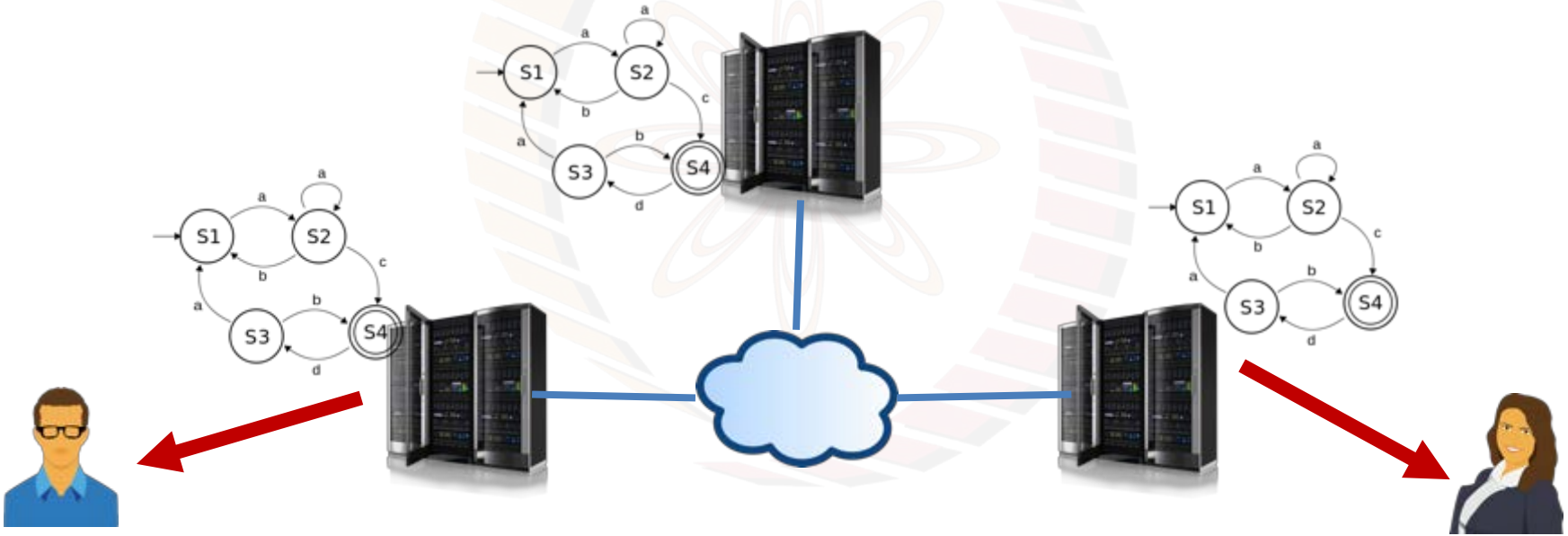
# Distributed State Machine Replication

5. Sync the state machines across the servers, to avoid any failure.



# Distributed State Machine Replication

6. If output state is produced, inform the clients about the output




But Consensus is still required.