BLOCKCHAINS

ARCHITECTURE, DESIGN AND USE CASES

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

PRAVEEN JAYACHANDRAN
IBM RESEARCH,
INDIA

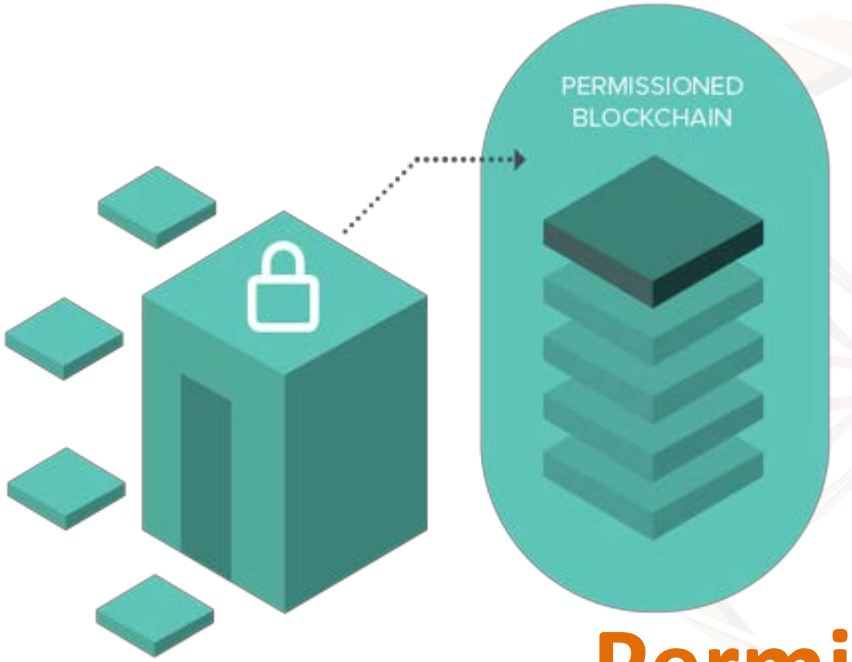


Image Source: https://nem.io/enterprise/

Permissioned Blockchain – I Basics

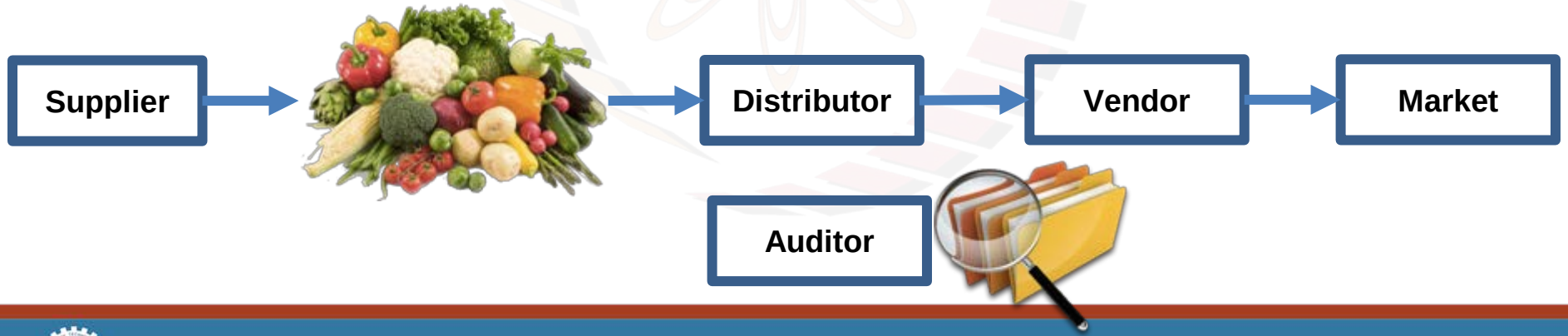
Permissioned Model

- 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

Use Cases

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

- Example: Provenance tracking of assets



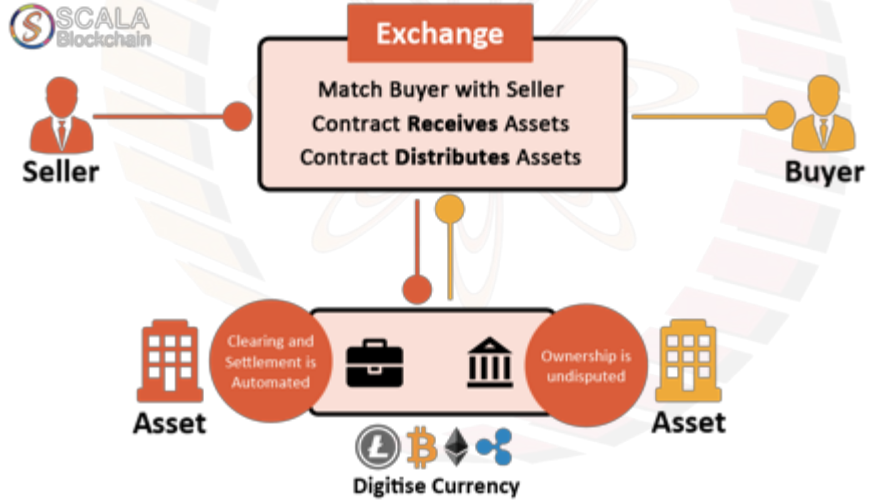
Smart Contracts

- "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/

- Remember the **bitcoin scripts** – you can change the script to control how the money that you are transferring to someone can be spend further
  - Your friend can use that money immediately
  - Your friend can use that money after 2 months

Smart Contracts

- You can extend the script to ensure smart contract execution
  - Execute a transaction only when certain condition is satisfied



Source: http://www.scalablockchain.com/smartcontract.html

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
  - Consider *golang* – iteration over a map may produce a different order in two executions

```
m := map[string]string{ "key1":"val1", "key2":"val2" };
for k, v := range m {
        fmt.Printf("key[%s] value[%s]\n", k, v)
}
```

Design Limitations

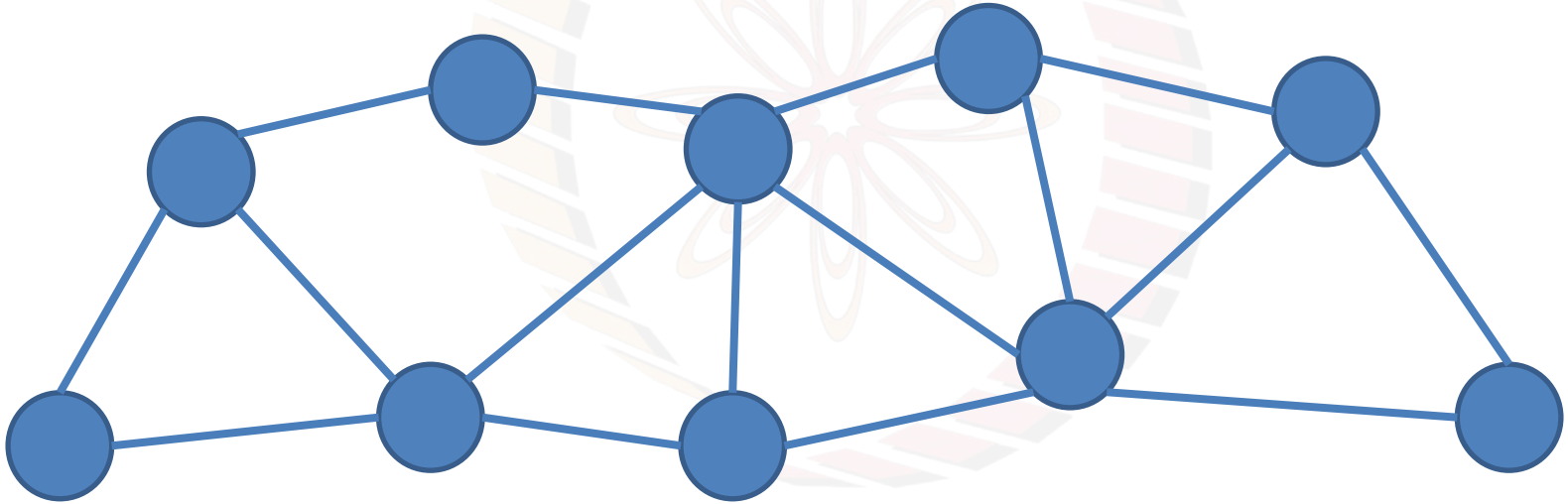
- Non-deterministic Execution
  - Smart-contract execution should always needs to be deterministic; otherwise the system may lead to inconsistent states (many fork in the blockchain)
  - Solution: Domain specific language (DSL) for smart contract

Design Limitations

- Execution on all nodes
  - Generally, execute smart contracts at all nodes, and propagate the state to others – try to reach consensus
  - Consensus: Propagate *same state* to all nodes, verify that the states match
  - Do you have sufficient numbers of trusted nodes to validate the execution of smart contracts?

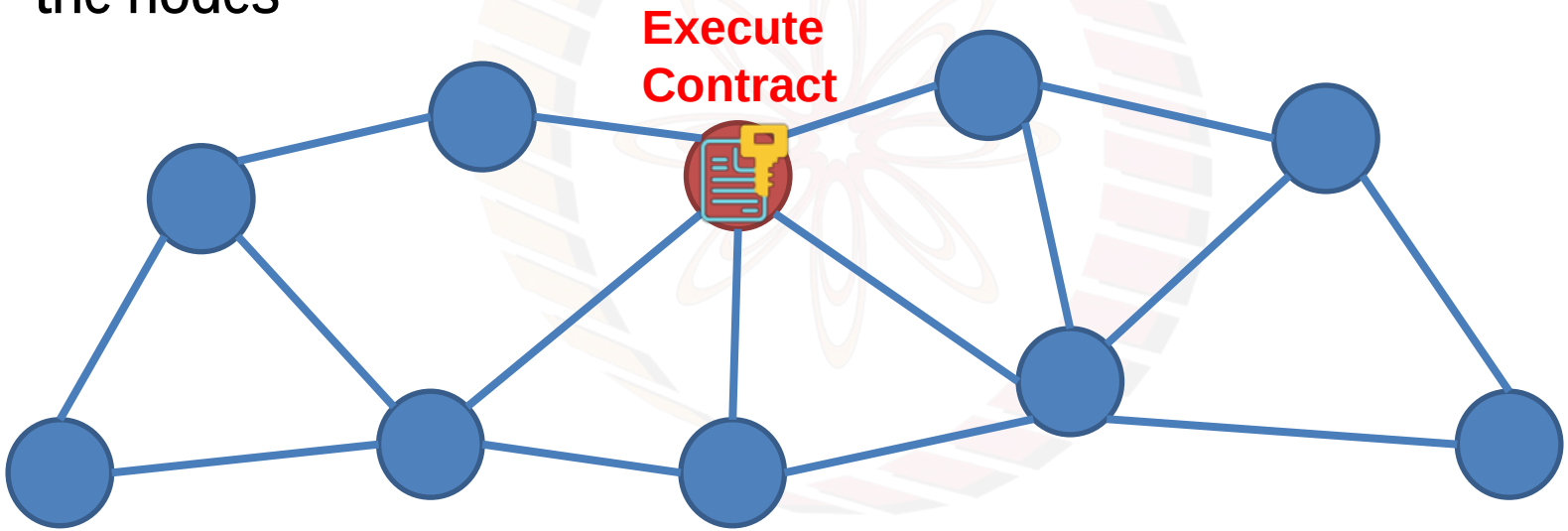
Do We Really Need to Execute Contracts at Each Node

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



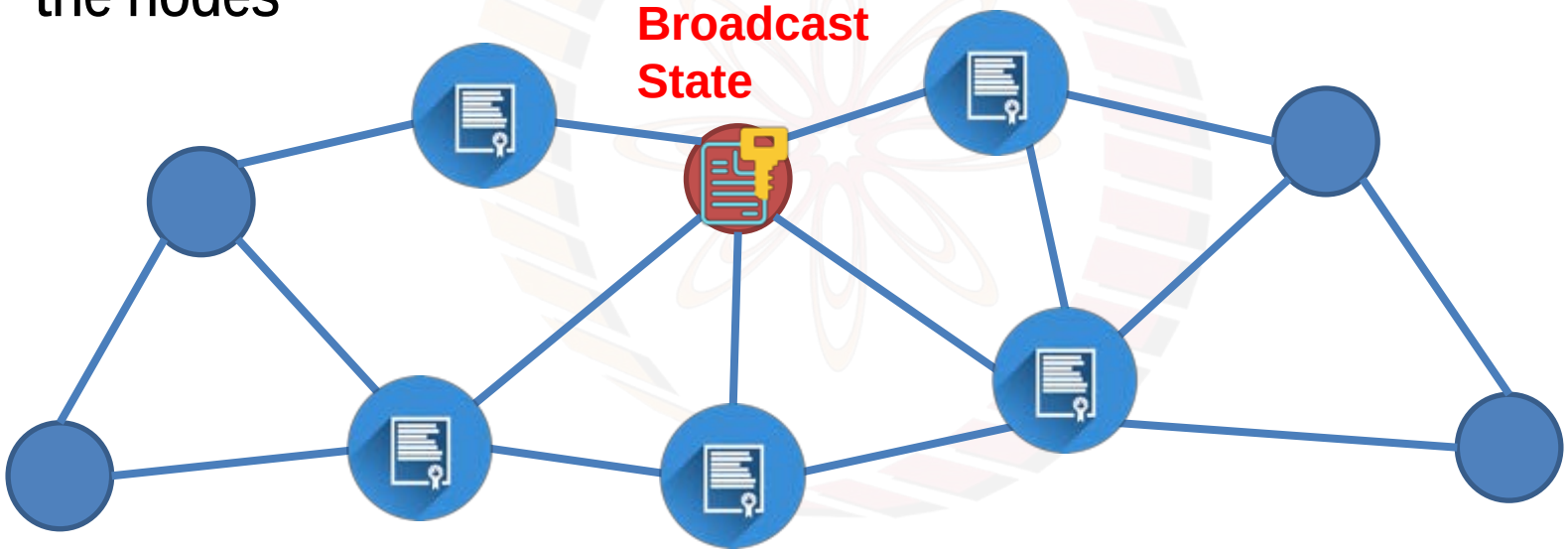
Do We Really Need to Execute Contracts at Each Node

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



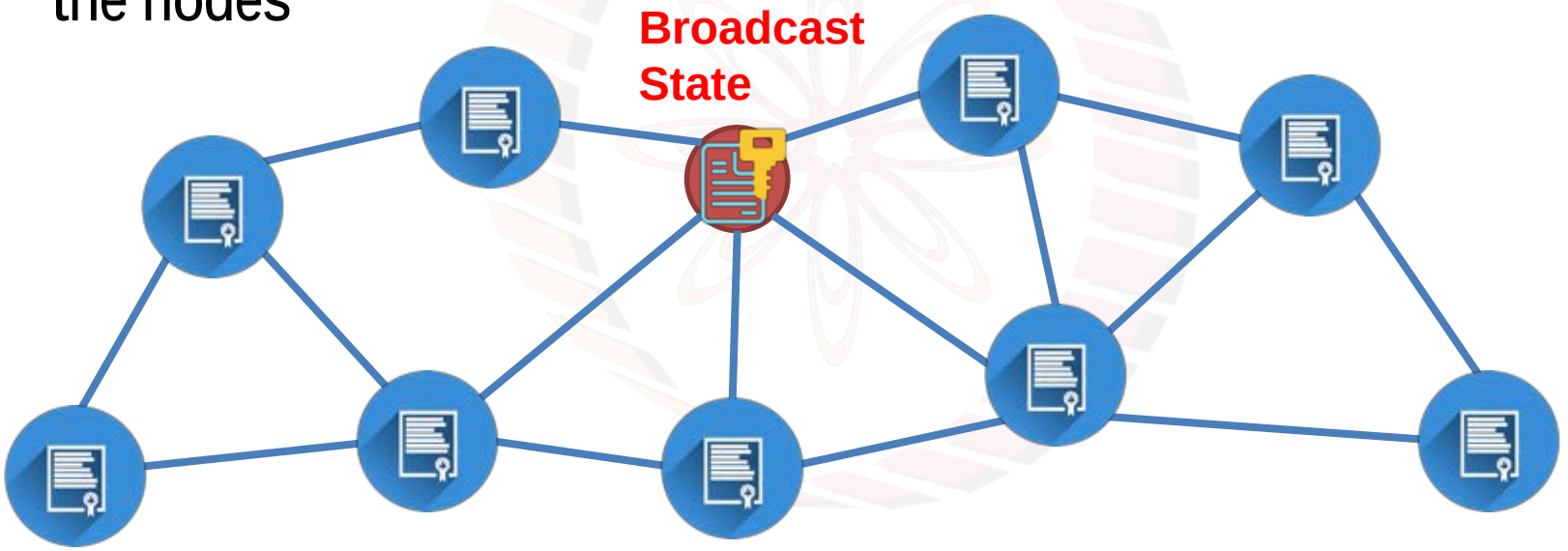
Do We Really Need to Execute Contracts at Each Node

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



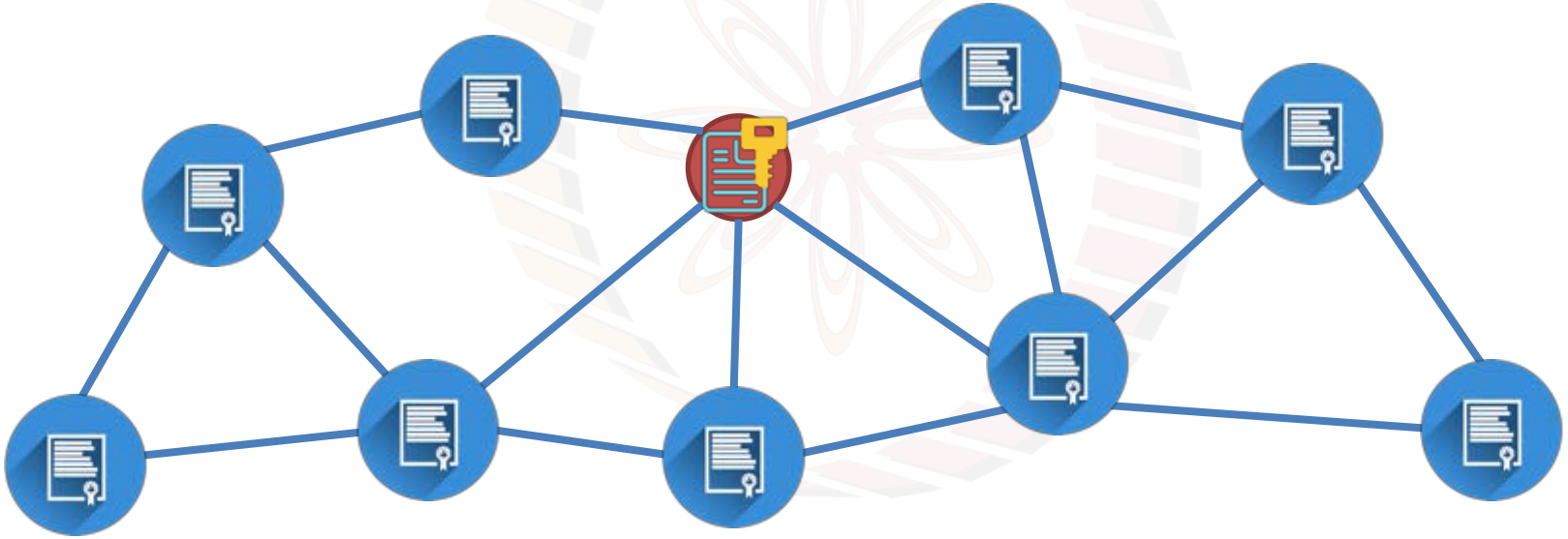
Do We Really Need to Execute Contracts at Each Node

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



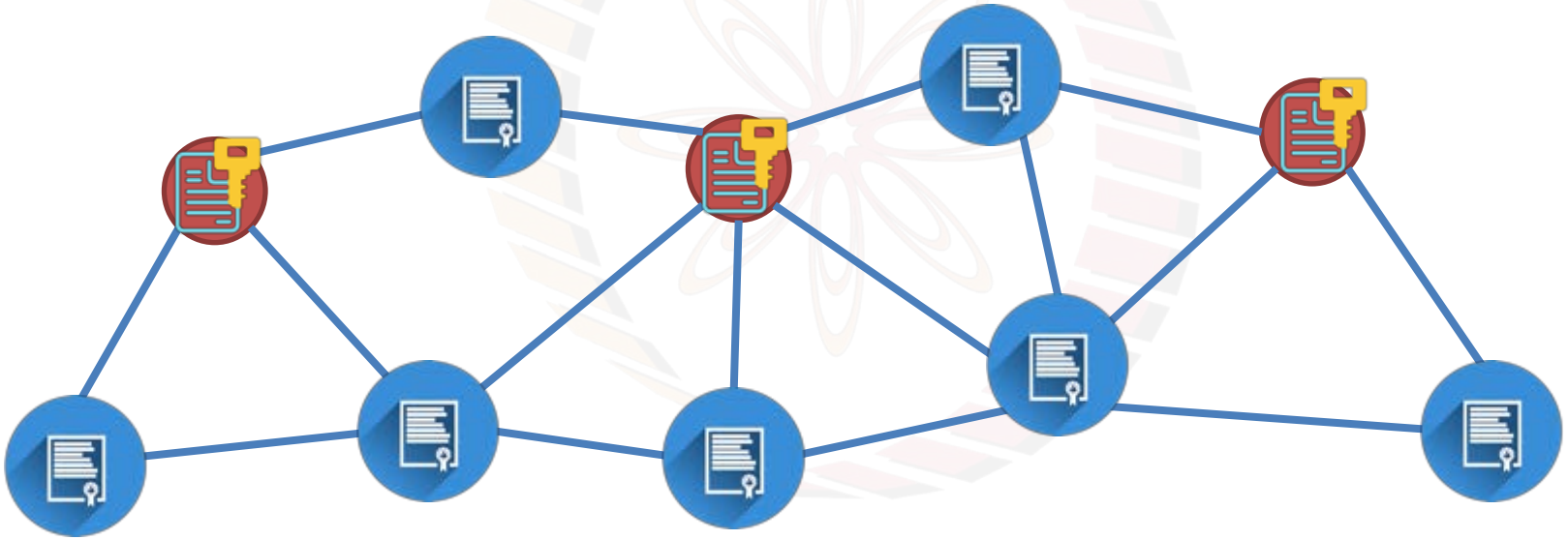
Do We Really Need to Execute Contracts at Each Node

- 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



State Machine Replication

- State machine
  - A set of states (*S*) based on the system design
  - A set of inputs (*I*)
  - A set o outputs (*O*)
  - A transition function $S \times I \rightarrow S$
  - A output function $S \times I \rightarrow O$
  - A *start* state

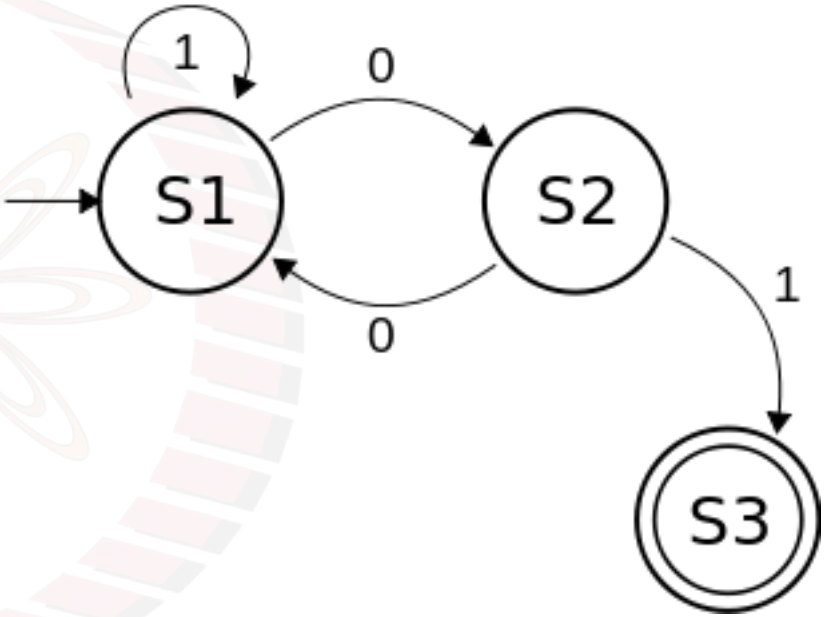
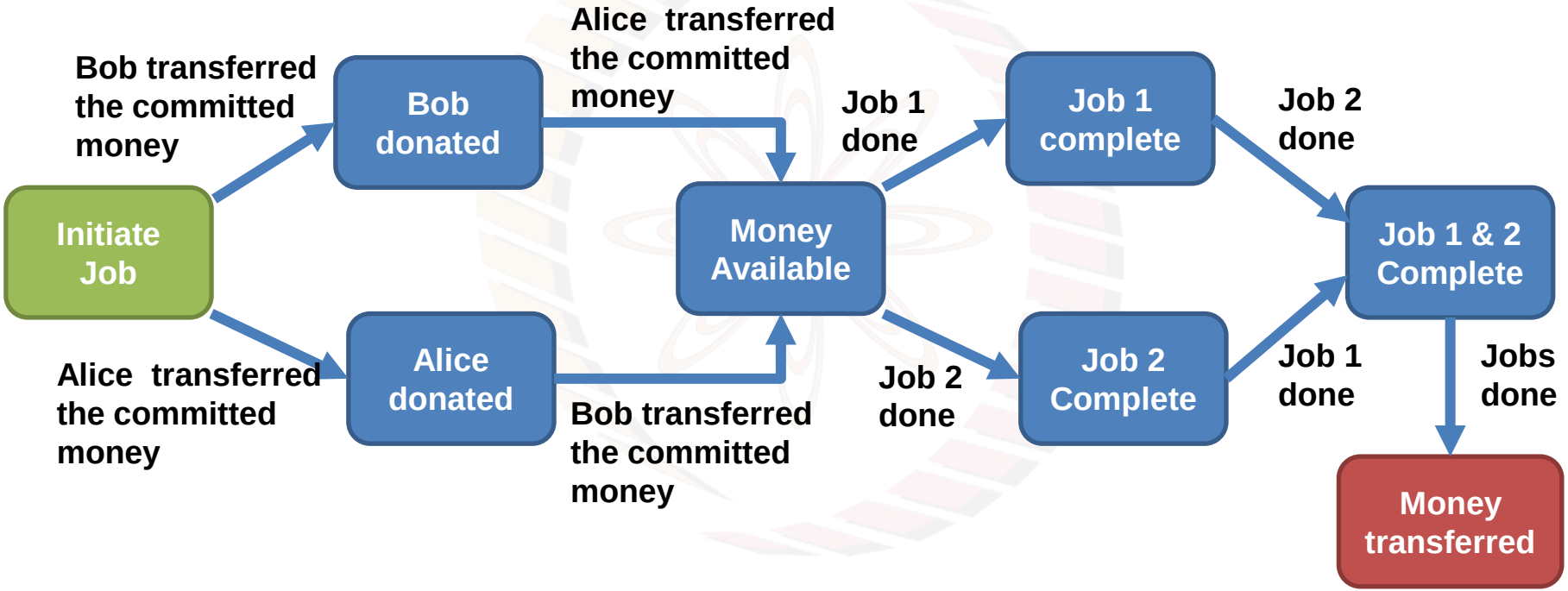


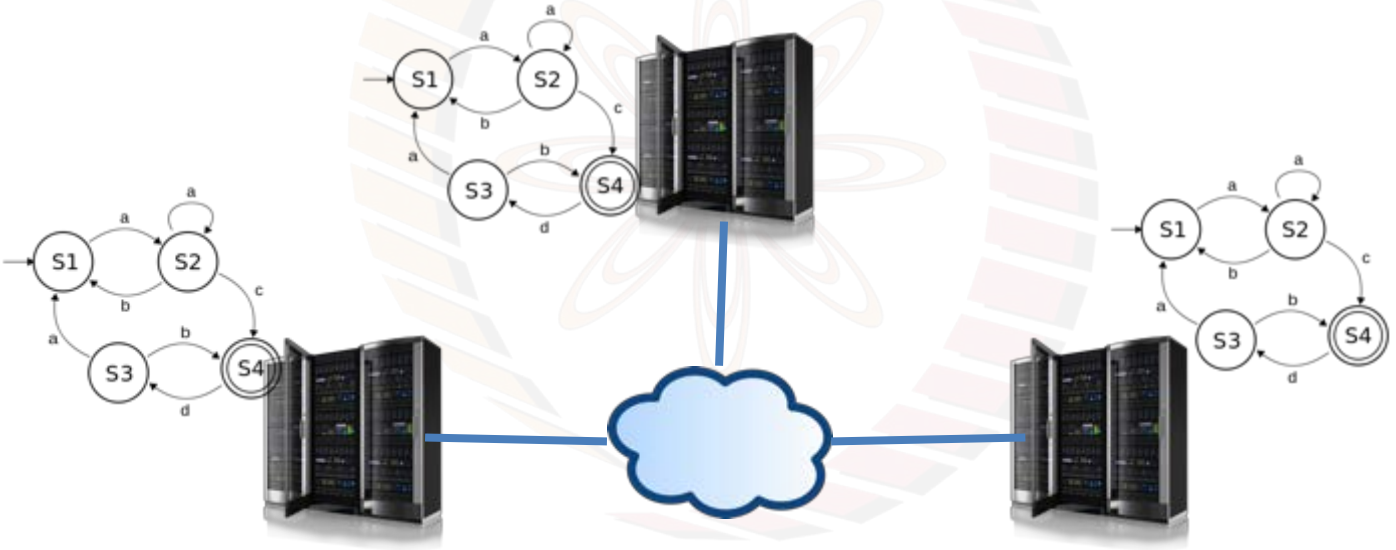
Image source: commons.wikimedia.org

Smart Contract State Machine - Crowd-Funding



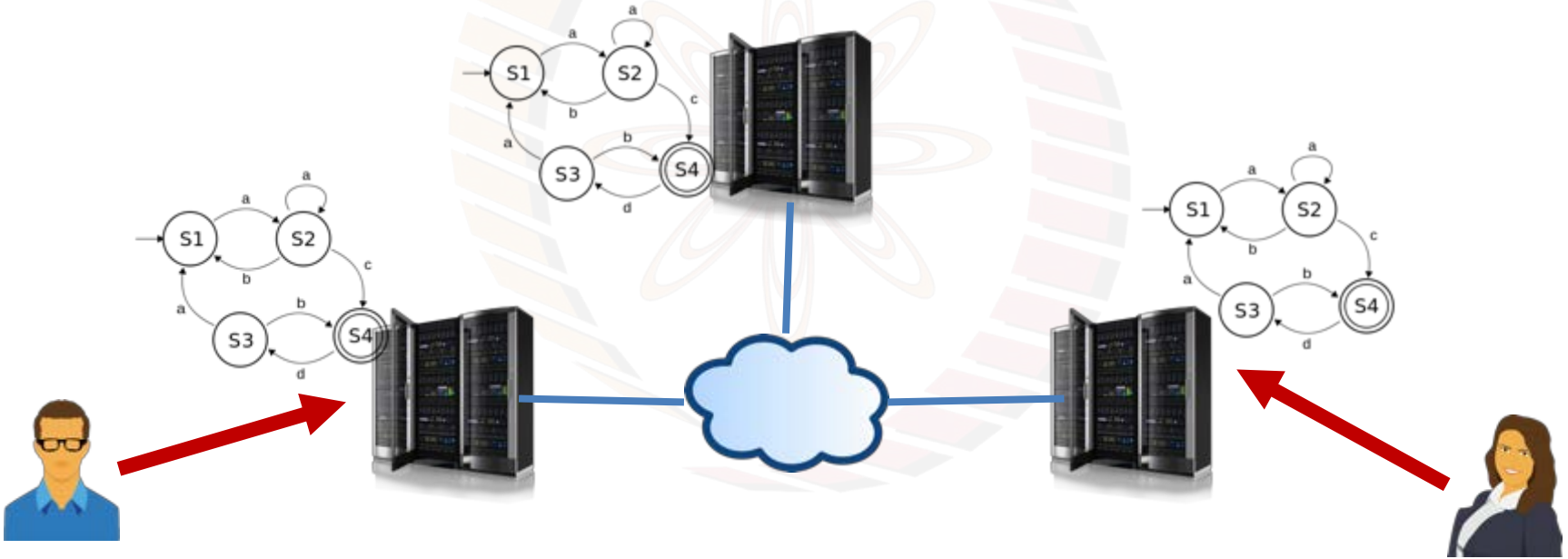
Distributed State Machine Replication

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



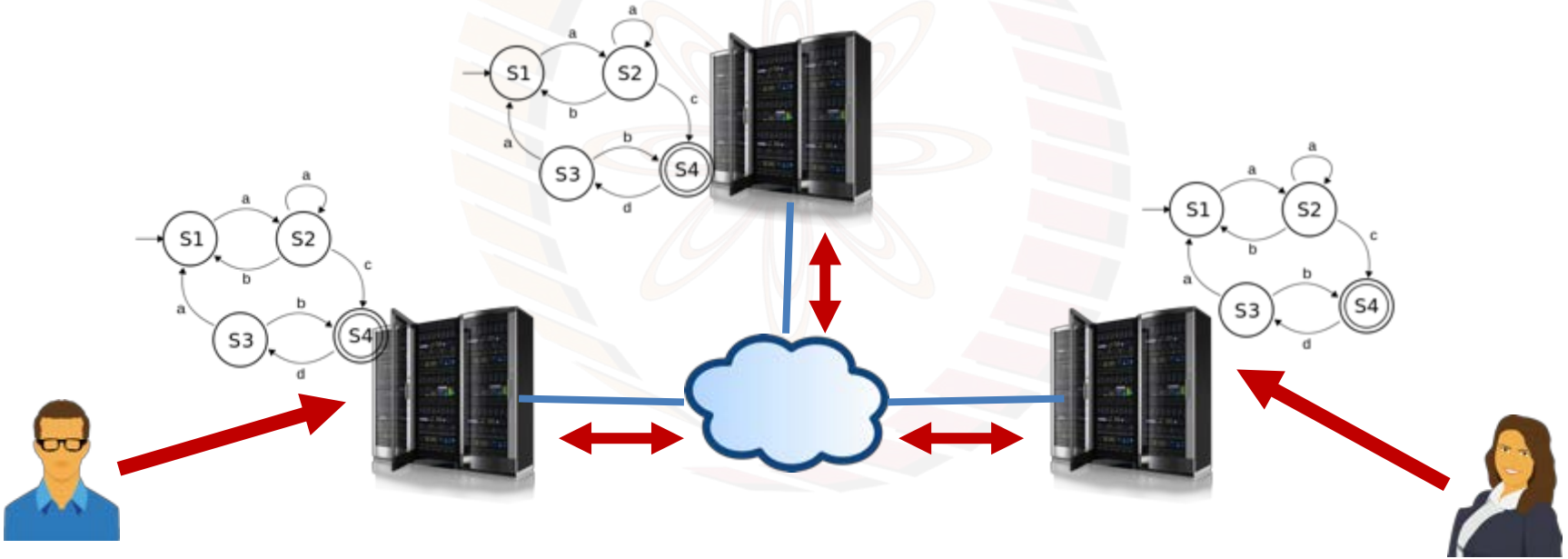
Distributed State Machine Replication

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



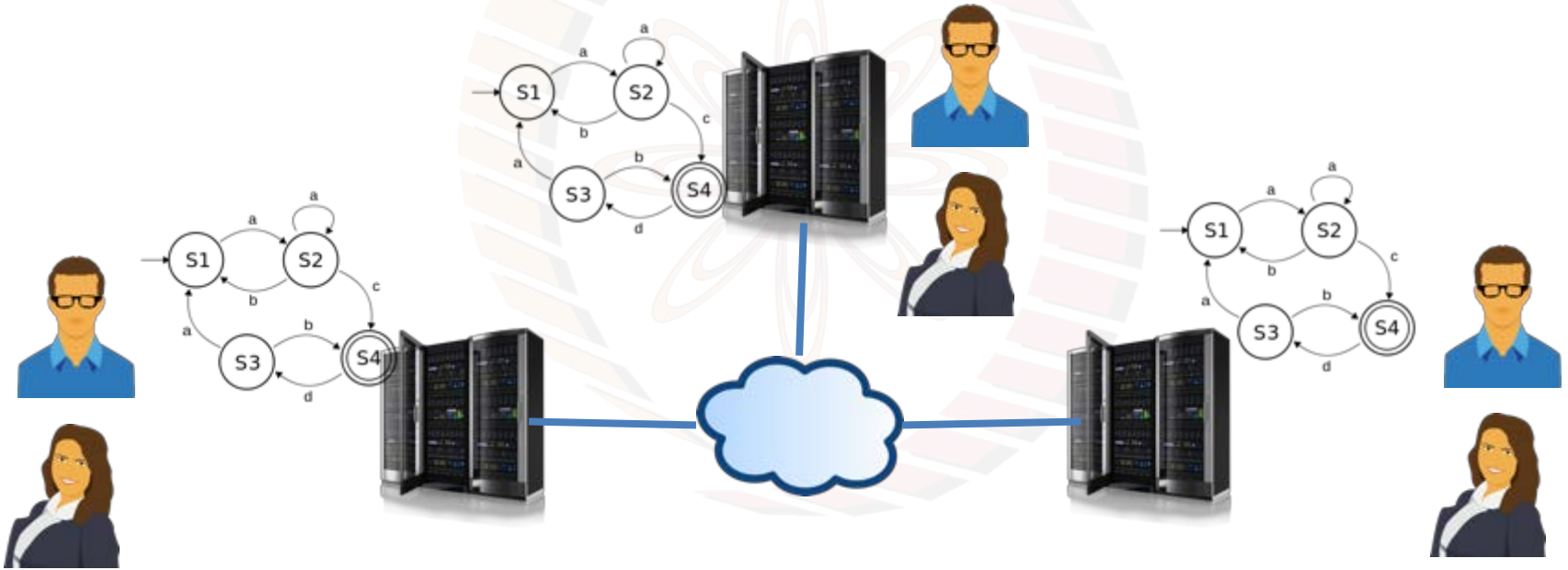
Distributed State Machine Replication

3. Propagate the inputs to all the servers



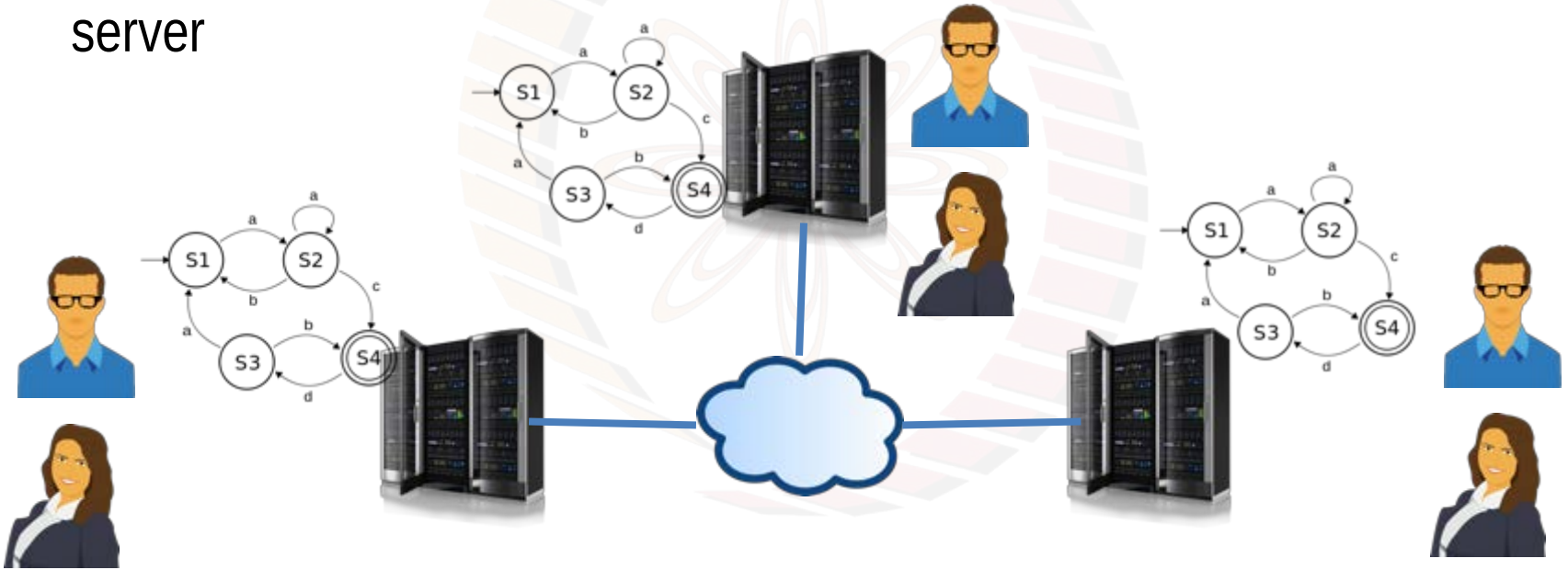
Distributed State Machine Replication

4. Order the inputs based on some ordering algorithm



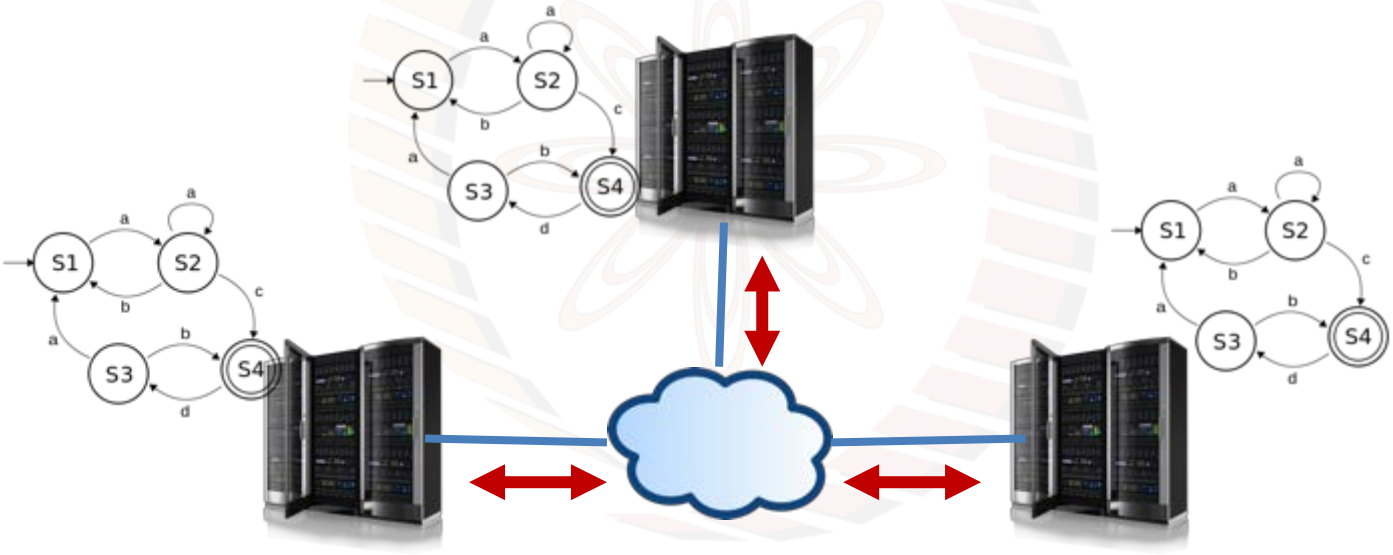
Distributed State Machine Replication

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



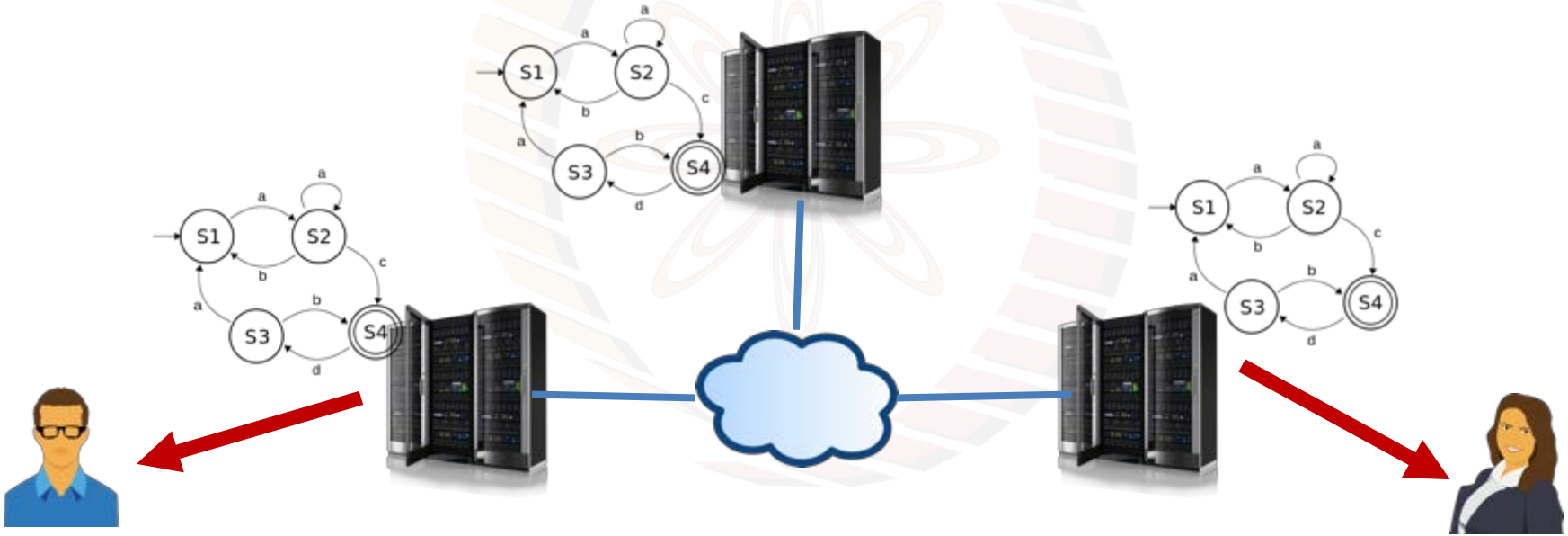
Distributed State Machine Replication

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



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


thank you!