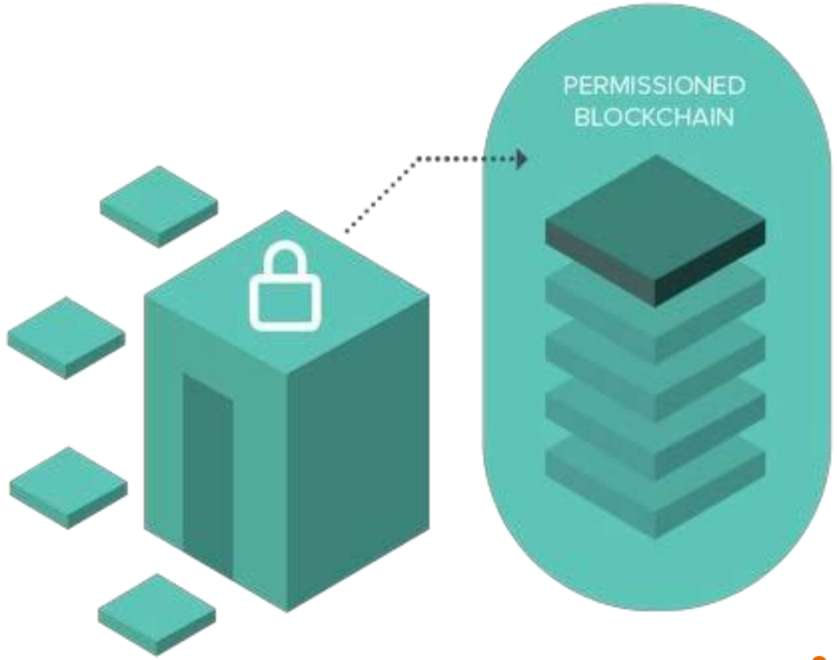


Image Source: https://nem.io/enterprise/

Permissioned Blockchain – I Basics

Outline

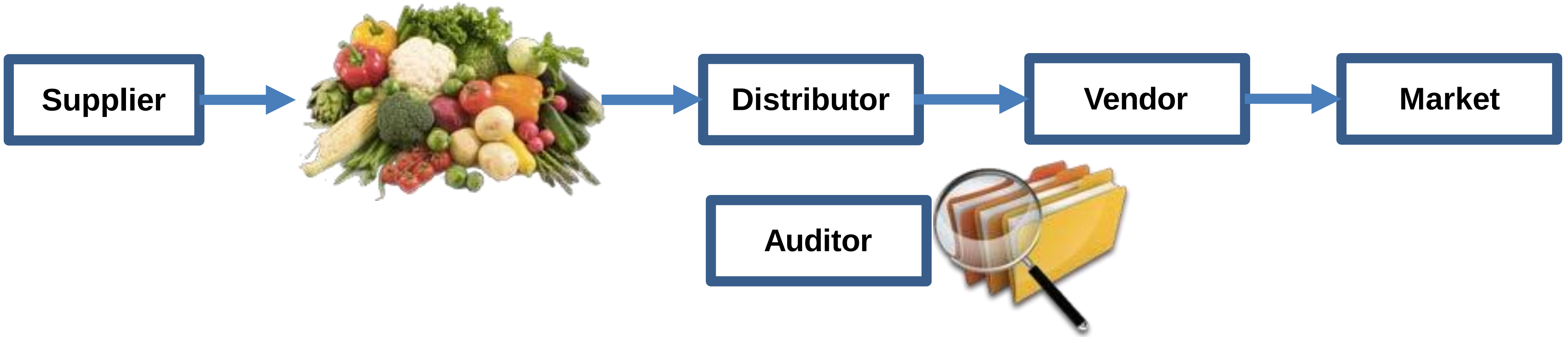
- Till date, we have discussed about the permissionless (open) BC settings where we have discussed about Architecture, System Internals of a bitcoin-based protocol.
- Means how a bitcoin-based system works, its design internals.
- Bitcoin network works on permisonless model, where any one can join the network without any authentication procedure.
- Another version of BC is the closed environment or permissioned environment, Means all **neighbors know each other but they don’t have trust on each other.**
- Means any node **can not join the network directly** with out any **pre authentication procedure**.

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.
- It may happen in the BC network that certain users got pre authentication to use the system but they have started behaving maliciously. **For example**, In any business environment, any user can behave maliciously any time after the authentication mechanism.
- So the final goal here is **to 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**, where one particular asset is moving from supplier to distributor, vendor, and market then at every stage **log** is maintained. Here every one is making entry to the centralized data base maintained by agency itself. But if **multiple authorities** like many countries involved in it then issue of trust comes in to picture.



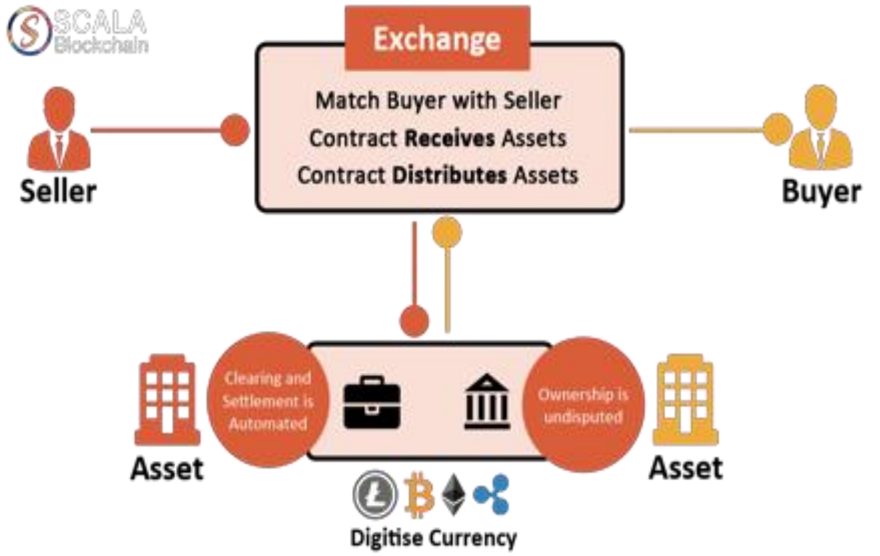
Smart Contracts

- “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 of Permissioned environment

- **Sequential Execution**
  - **Execute transactions sequentially** based on consensus, means if certain TXs get verified/committed will be executed first and so on.
  - **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 (here is the number of TXs committed per sec or per unit time) is inversely proportional to the commitment latency.
  - **Can be a possible attack** on the smart contract platform – introduce contract which will take long time to execute.

Design Limitations

Second problem is implementation of the smart contract

Means you need to select any language to write the smart contract, which will give you more power as compared to bitcoin scripts (It does not support constructs like loop)

- **Non-deterministic Execution**
  - Consider ***golang*** – iteration over a map (data structure) 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)
}
```

At different run the order of pair of values are different.

Design Limitations

- **Non-deterministic Execution**
  - **Why 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
  - **For example,** Etherium is a platform where you can implement smart contract with the constrct that they have supported.

Design Limitations

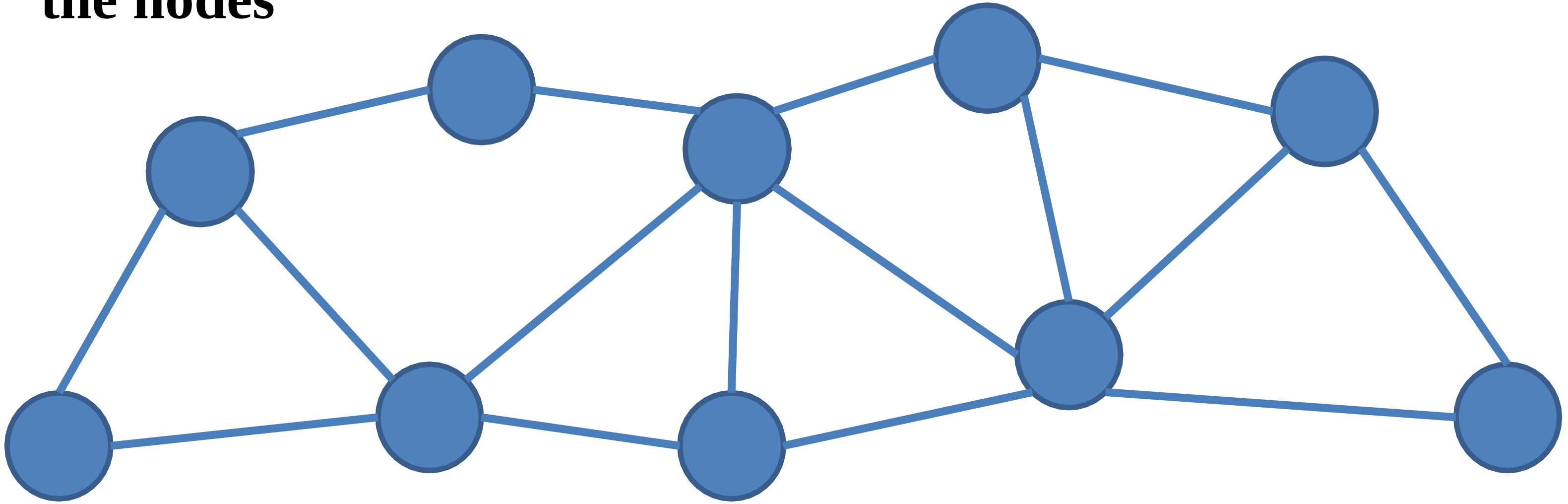
- **Third design limitation is the “Execution on all nodes"**
  - Generally, execute smart contracts at all nodes, and propagate the state to others – try to reach consensus
  - Consensus: Propagate *same state* to all nodes, verify that the states match
  - But to ensure consensus, **do you have sufficient numbers of trusted nodes to validate the execution of smart contracts?** What happened if the number of trusted nodes are less than the malicious nodes? Then, they may get the control over the entire network.
  - But you can prevent it by using permission less settings by PoW based consensus mechanism.
  - But some times here contract may take long time to execute and all the contracts are getting backlogged.

Motivation for permissioned environment

- So in case of permission settings, we moved from challenge response based method to traditional distributed method for consensus mechanism.
- But to do it, you need to have sufficient number of trusted nodes ?

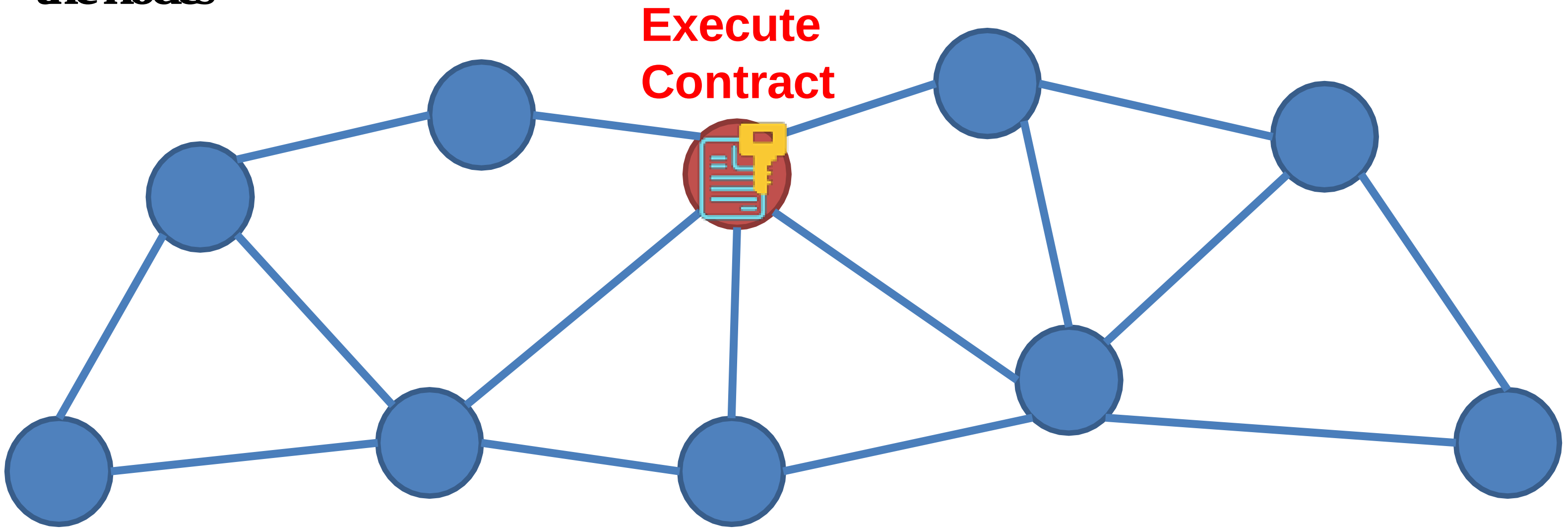
Do We Really Need to Execute Contracts at Each Node?

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



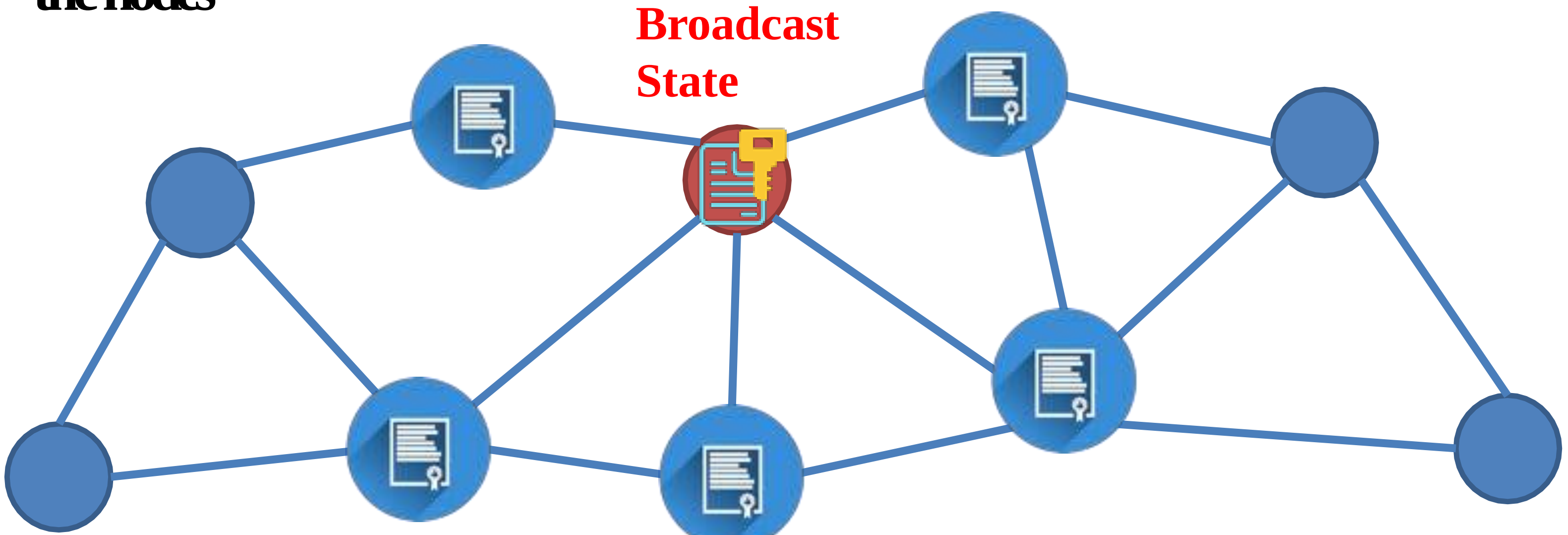
Do We Really Need to Execute Contracts at Each Node

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



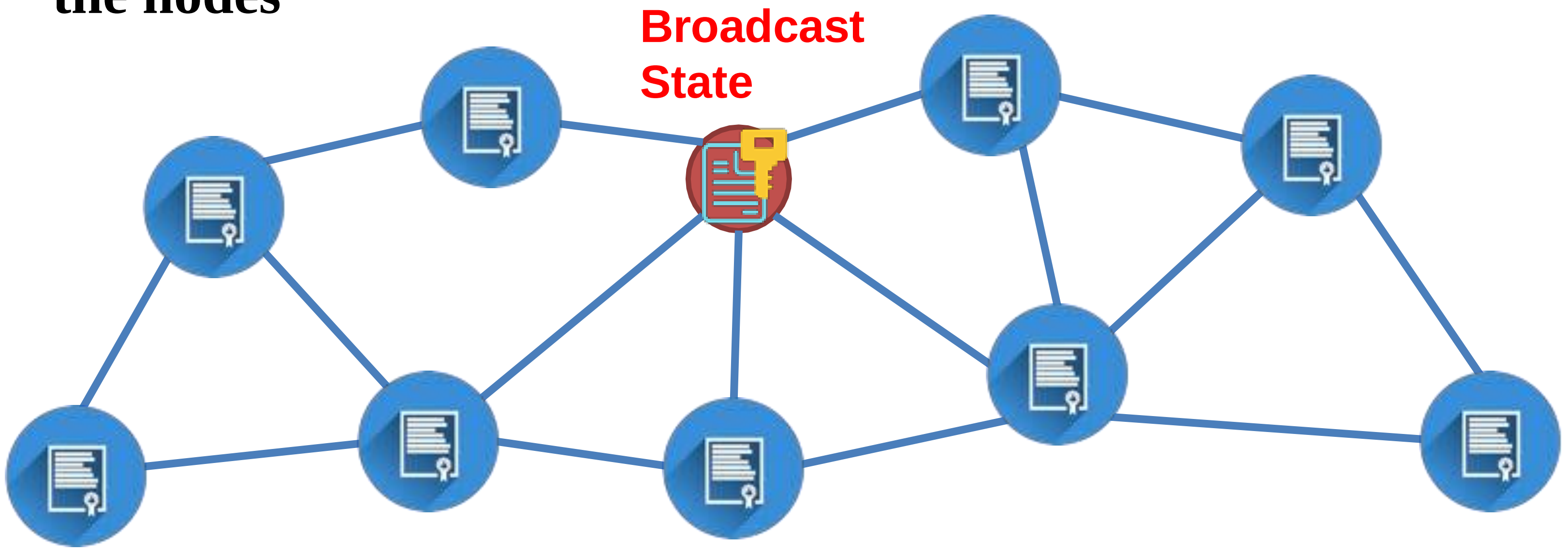
Do We Really Need to Execute Contracts at Each Node

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



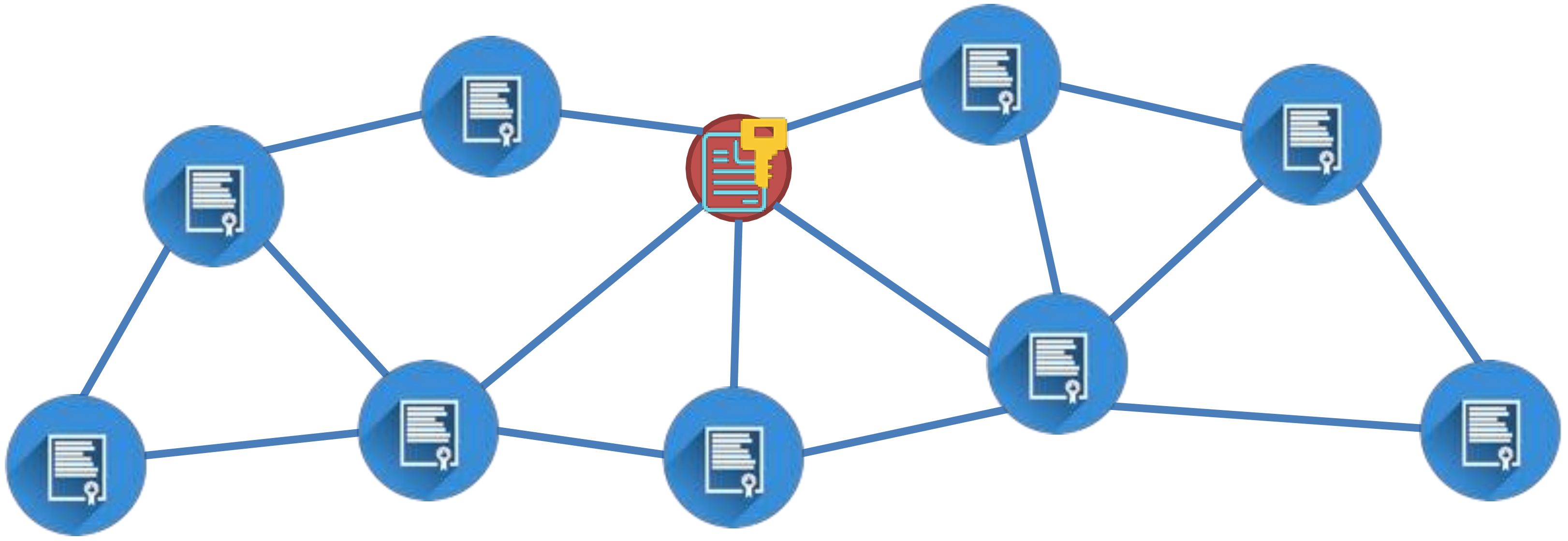
Do We Really Need to Execute Contracts at Each Node

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



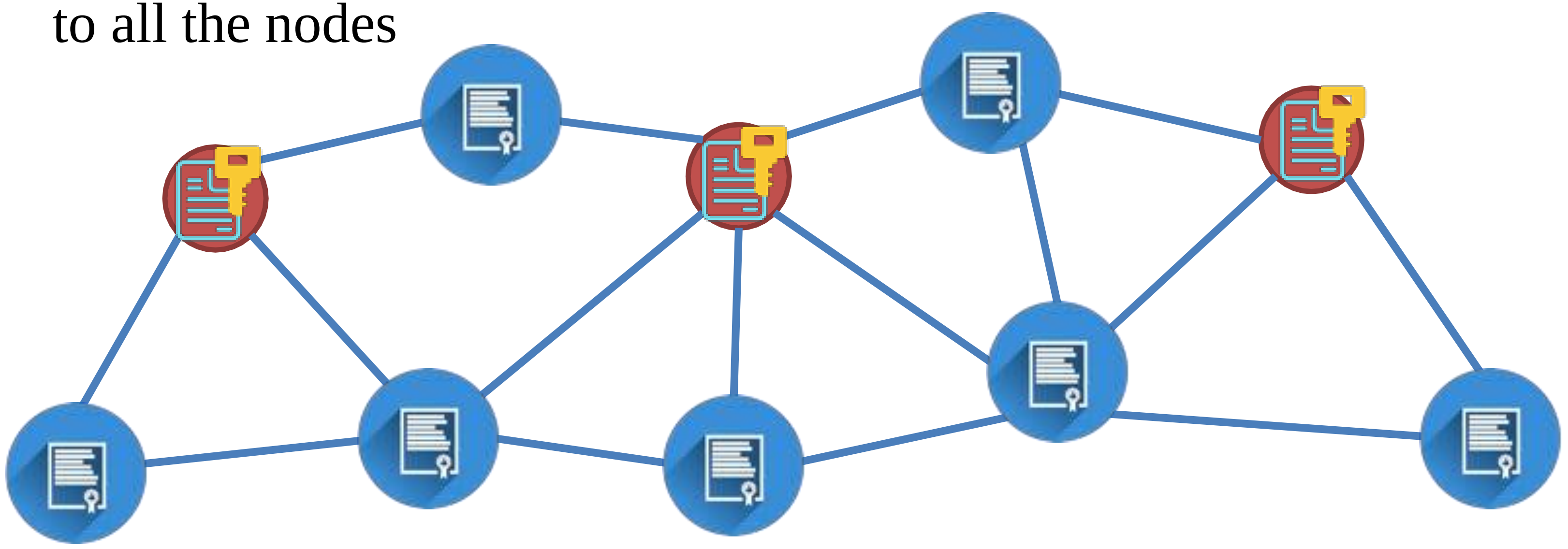
Do We Really Need to Execute Contracts at Each Node

- **What if the node that executes the contract is faulty?**
- **Then, the system becomes down and no progress further**



Do We Really Need to Execute Contracts at Each Node

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



State Machine Replication used to achieve consensus

State machine contains parameters such as

– A set of states (S) based on the system design

– A set of inputs (I): tell you how your system behaves

– A set o outputs (O): here S3 is the final O/P

– A transition function $S \times I \rightarrow S$; e.g.) $S1 \times 0 \rightarrow S2$

– A output function $S \times I \rightarrow O$; e.g.) $S1 \times 0 \rightarrow S2$

$\rightarrow 0$

– A *start* state; S1 is the start state

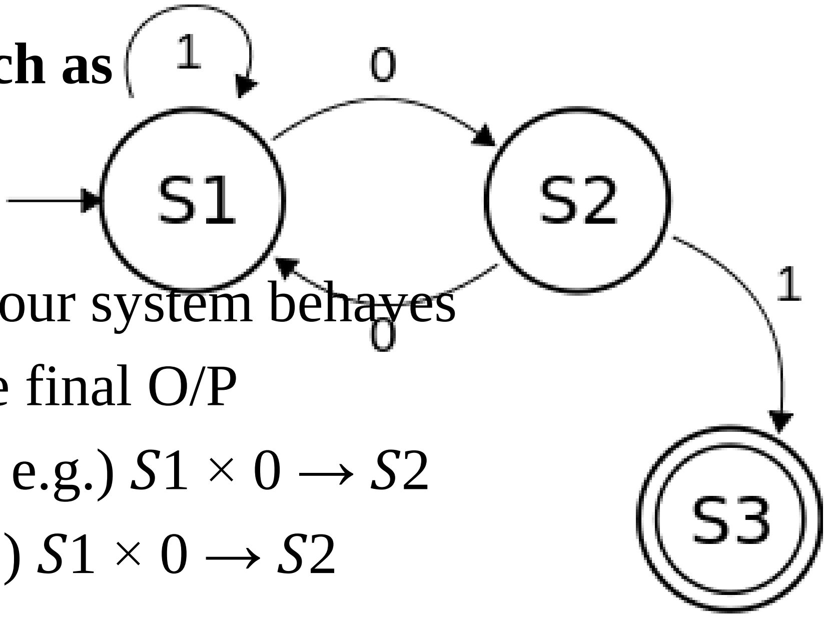
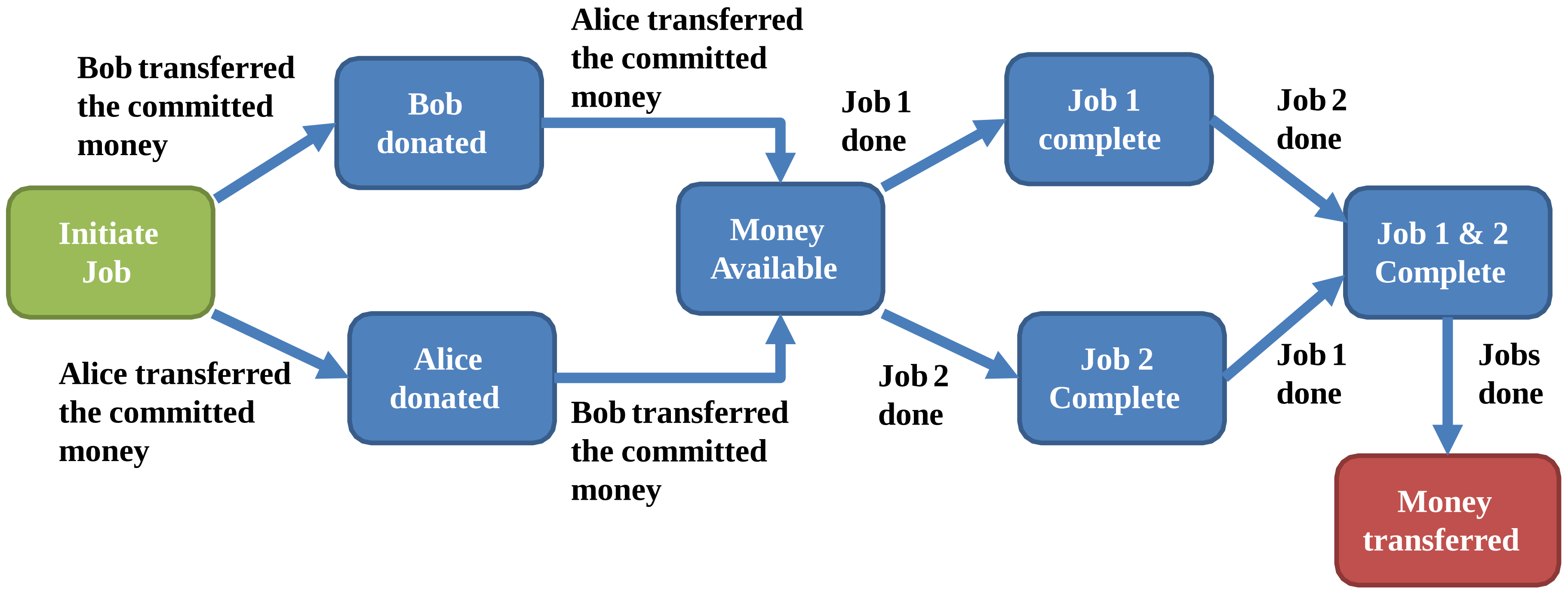


Image source: **commons.wikimedia.org**

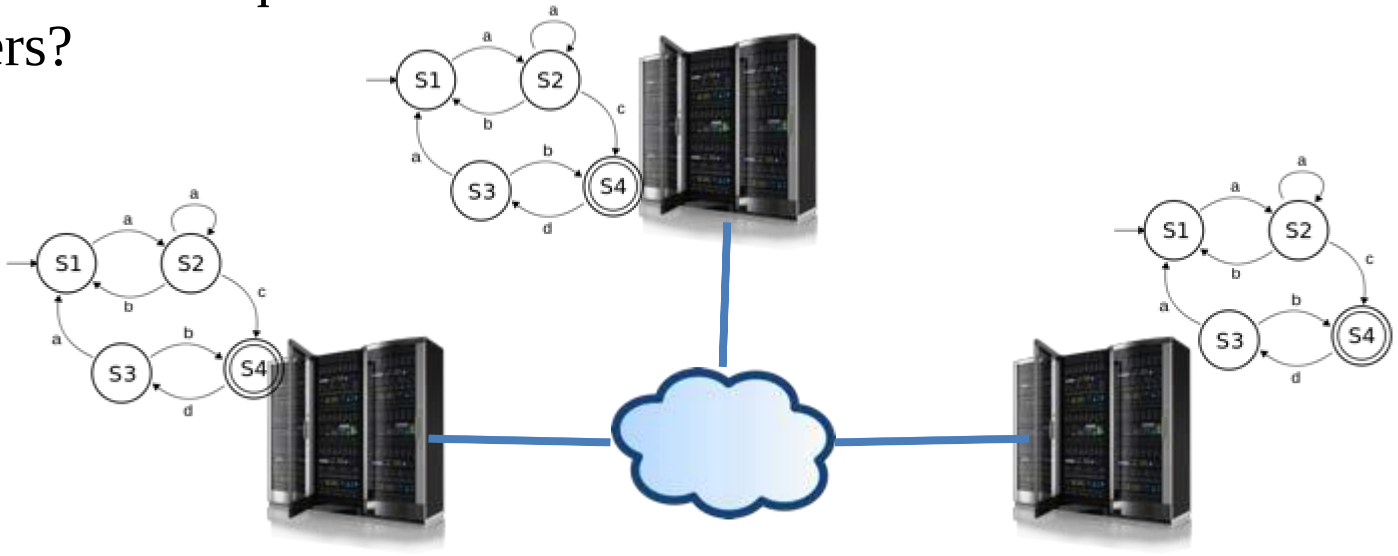
Representation of Smart Contract as a State Machine - **Crowd-Funding**



How Distributed State Machine Replication works

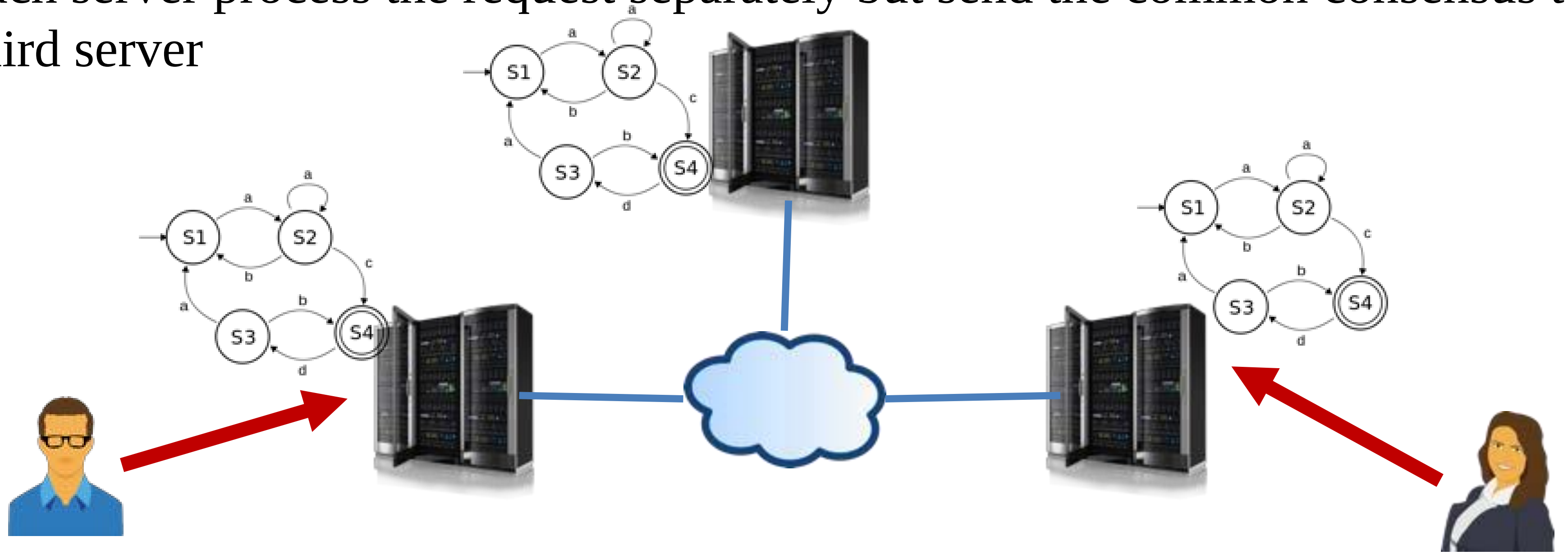
1. Place copies of the state machine on multiple independent servers.

- Why there is requirement to use distributed servers instead of Centralized servers?



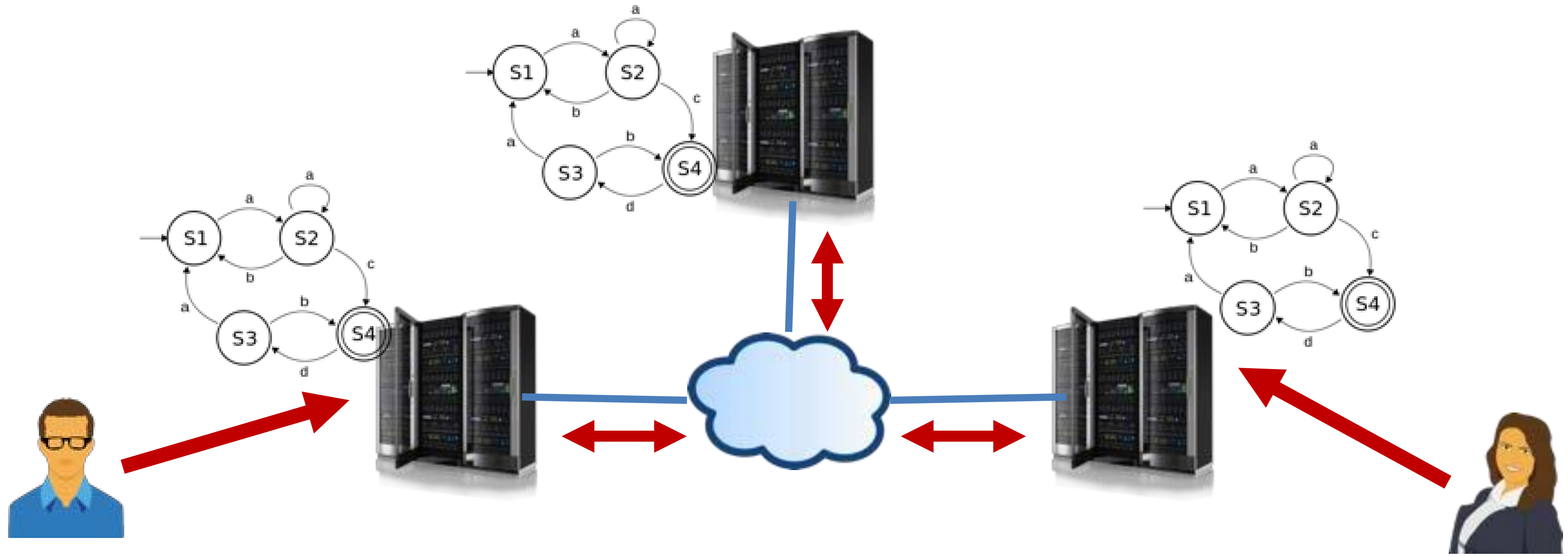
Distributed State Machine Replication

2. Server receives the client requests, as an input to the state machine and each server process the request separately but send the common consensus to third server



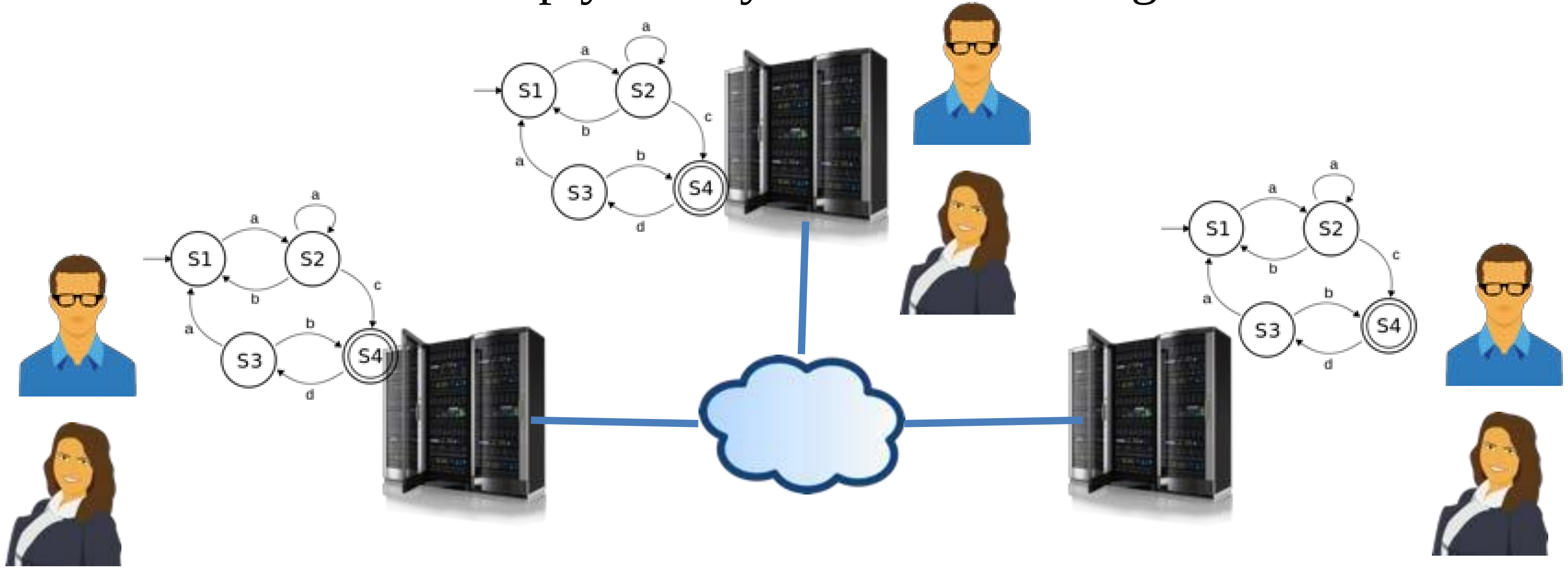
Distributed State Machine Replication

3. To achieve common consensus-Propagate the inputs to all the servers



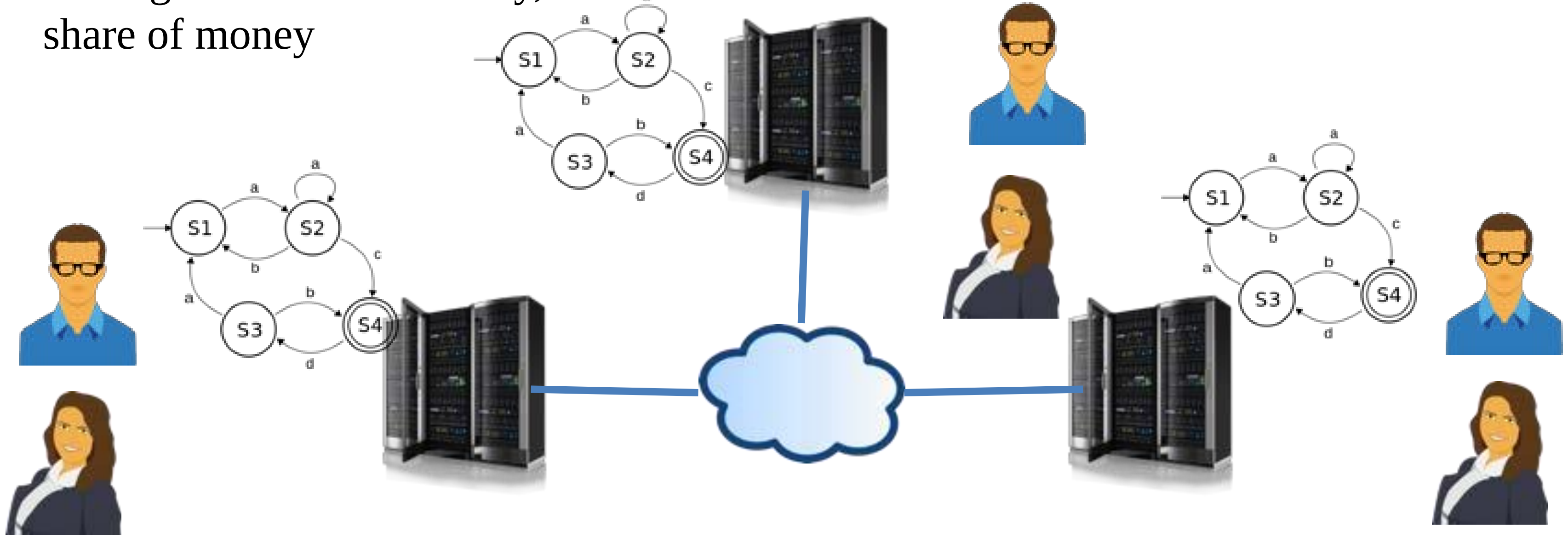
Distributed State Machine Replication

- 4. Order the inputs based on some ordering algorithm with time-stamp. Based on the timestamp you may have the ordering of Txs



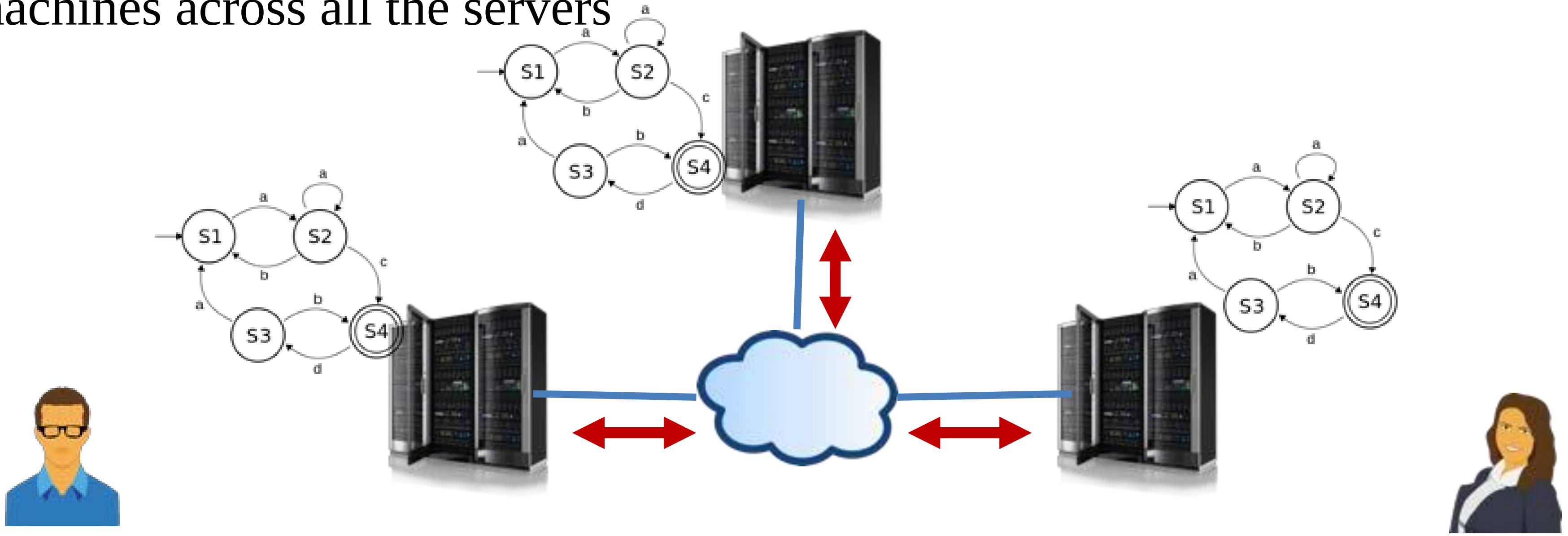
Distributed State Machine Replication

4. Every system execute the inputs based on the order decided individually at each server. Then, we will know first the Tx corresponds to Bob gets executed then Tx corresponds to Alice gets executed. Finally, all servers reach to the state that both have transferred their share of money



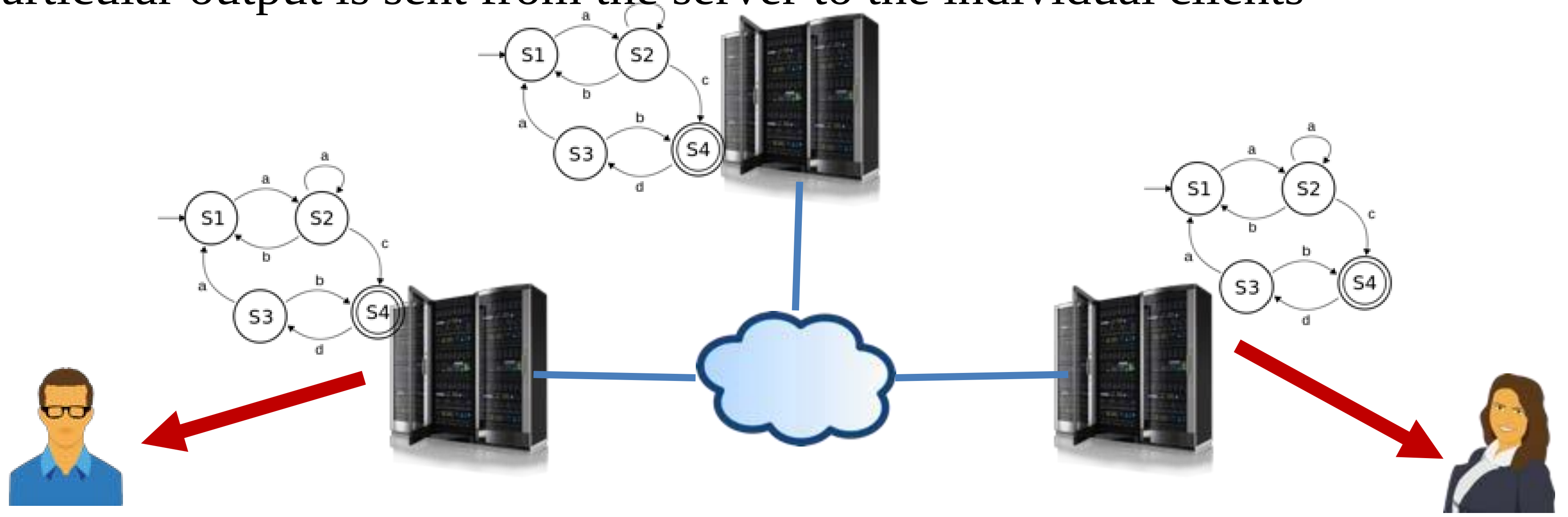
Distributed State Machine Replication

5. Sync the state machines across the servers, to avoid any failure. It may happen that during any Tx any server may fail so we need to sync state machines across all the servers



Distributed State Machine Replication

6. If output state is produced (means when money is transferred then inform the clients that work has been done), inform the clients about the output. That particular output is sent from the server to the individual clients



Problems in Distributed State Machine Replication

In this entire procedure, there are **<u>two problems:</u>**

1. You need to **maintain the ordering service.**
2. In the **presence of a failure**, you have to ensure that all the individual servers are in the same pace means all the servers must knows that both the Txs from Bob and Alice are committed and the entire money has been transferred.

Permissioned Blockchain and State Machine Replication

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