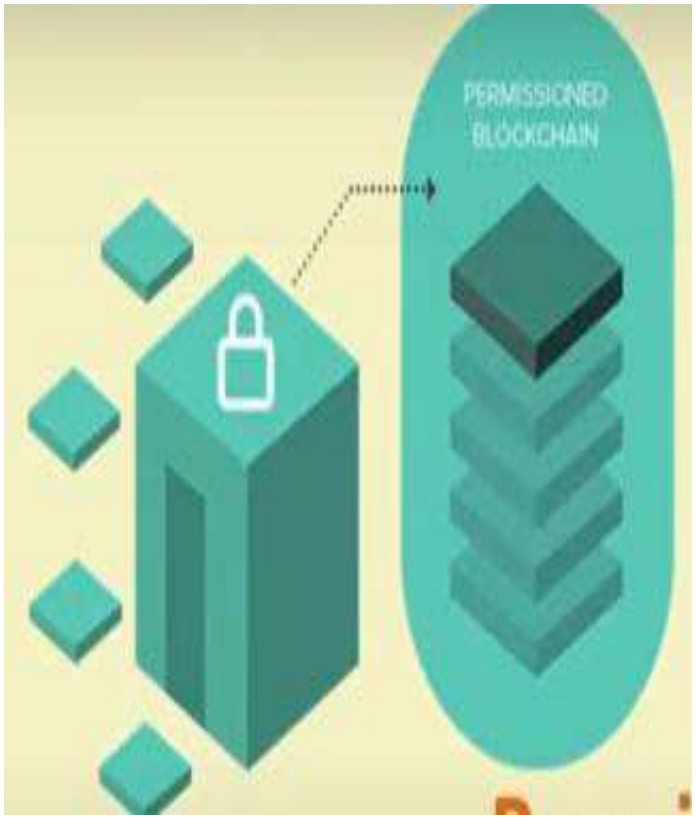
- **[Unit 3]**
- Permissioned Blockchain Consensus

Unit 3 Permissioned Block chain Consensus	**6 Hrs.**
Permissioned model, Design Limitations for Permissioned block chains, Execution of smart contracts, Smart Contract State Machine – Crowd-Funding, Distributed State Machine Replication, Permissioned Block chain & State Machine Replication, Different Consensus Algorithms in Permissioned Block chain : Paxos, RAFT Consensus, Byzantine general problem, Byzantine fault tolerant system, Lamport- Shostak-Pease BFT Algorithm or Agreement Protocol.	

- Permissioned Blockchain
- These are the closed network only a set of groups are allowed to validate transactions or data in a given blockchain network.
- These are used in the network where high privacy and security are required.
- Characteristics:
- A major feature is a transparency based on the objective of the organization.
- Another feature is the lack of anatomy as only a limited number of users are allowed.
- It does not have a central authority.
- Developed by private authority.



Permissioned Blockchain

- There is a central authority responsible for issuing licenses to participate in the network.
- This authority may or may not grant equal rights to participating nodes, but in any case only authorized entities will have access to the records contained in the ledger.
- As permissioned blockchains usually have a restricted number of participants, and as the consensus mechanism is more centralized, they are able to process a larger number of transactions in less time compared to a public blockchain.
- However, since records cannot be independently verified, their integrity will depend solely on the credibility of the participants — meaning that they need to trust each other.

Design Limitations:

- Let us look into, many of the design limitations which are there in permissioned environment.
- So, if you look into the blockchain concept which actually comes from the typical permission less environment like a bitcoin type of example, There is one issue that we execute the transaction sequentially based on the consensus.
- So, it is like that if certain transaction gets verified or if it gets committed that will first executed and the transaction that is committed latter on, that will executed next.
- So, the request to the application, application here of the smart contracts, they are ordered by the order of the consensus in which the individual application of the individual contracts get a consensus and they are executed in that particular order.

Design Limitations:

- So, these kind of sequential order actually gives a bound on the effective throughput because you want to ensure that certain consensus or certain ordering of transactions are made.
- We apply that proof of work base techniques in case of permission less model where the network or the system gives the challenge to the individual users.
- Every user tries to solve that particular challenge individually and the nature of the challenge is such that it is difficult to find out the solution for that challenge, but once a solution is found everyone can verify it very easily.
- So, with that particular challenge response based method the nodes try to come to the consensus, but as we have seen like the challenge is very hard.

- Design Limitations:
- As the challenge is taking certain time to get a solution and that is the reason that if you want this kind of serializable order of transactions execution you get a bound on the effective throughput
- So, ideally the throughput is inversely proportional to the commitment latency. So it is like that if your commitment latency gets increased, your throughput will get then decreased.
- So it is like that in a permission less environment in a bitcoin type of environment if you increase the difficulty of your challenge or if you increase the complexity of your challenge the effective throughput that you will get that (throughput here in terms of no. of transactions that can be committed per second per unit time), So the effective throughput that you will get it will drop inversely proportional to the commitment latency so this can be a possible attack on the smart contract platform.

- Design Limitations:
- When you introduced contract, it will and which will take long time to execute and that is why if certain contract take huge time to execute, the other contracts will not able to execute in further, because once the consensus for the previous contract has been reached then only you will be execute the contracts which has been submitted later on.
- So you maintain a kind of serializability order of the transaction which is preventing to execute later contract until the previous contract gets executed.
- If you introduced a malicious contract in the system which will take huge amount of time for execution you will be able to launch a denial of service attack on your consensus algorithm, this is the kind of 1st problem.

- Design Limitations:
- 2nd problem is the implementation of smart contract.
- To implement a smart contract need to go to some language which will give more power compared to bitcoin script.
- Bitcoin script is not turing complete language but because it is not turing complete language, it dose not support all the contracts which can be there.
- For ex. It dose not support loops or it has certain limitation in execution
- But to implement smart contract, because you want to increase a power of that particular script, you can write any general purpose contract in the form of a code by using different type of language for that.
- So the developers are develop different types of languages. So one interesting language is that Golang which widely used in this kind of contract execution platform.

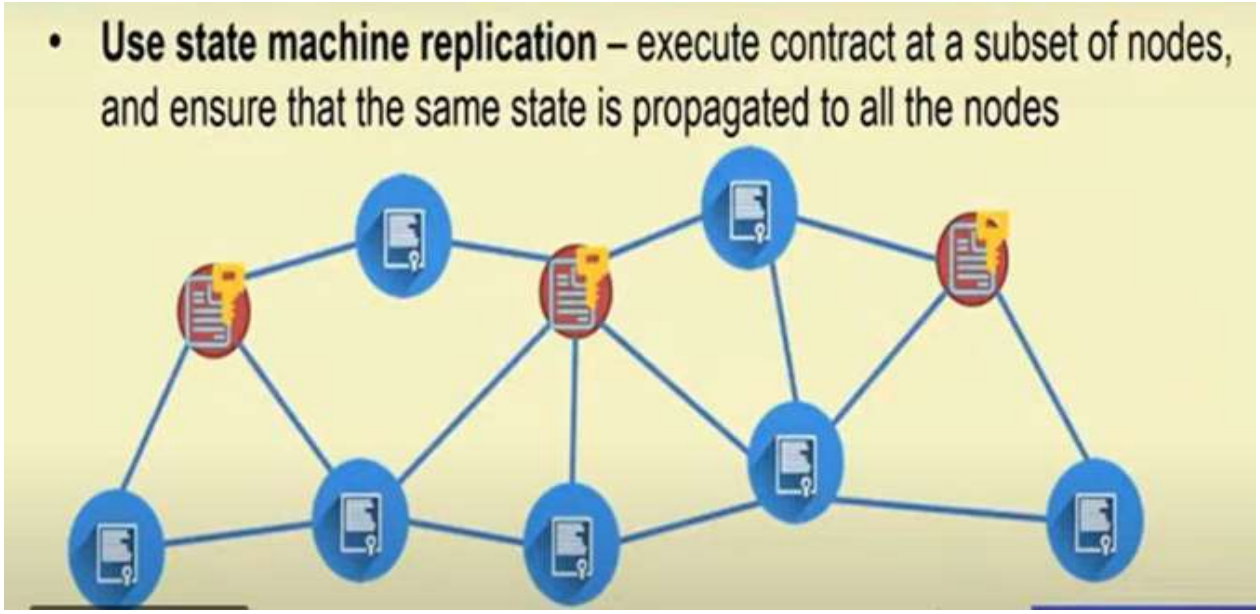
Execute Contracts:

- In case of permissioned model you have to move from challenge response based method of permissioned less to traditional distributed system based consensus algorithm.
- But in that case you have to ensure that you have sufficient no. of trusted nodes in the system.
- So one interesting question that comes for consensus of permissioned blockchain model that do you always need to execute the contract at each node.

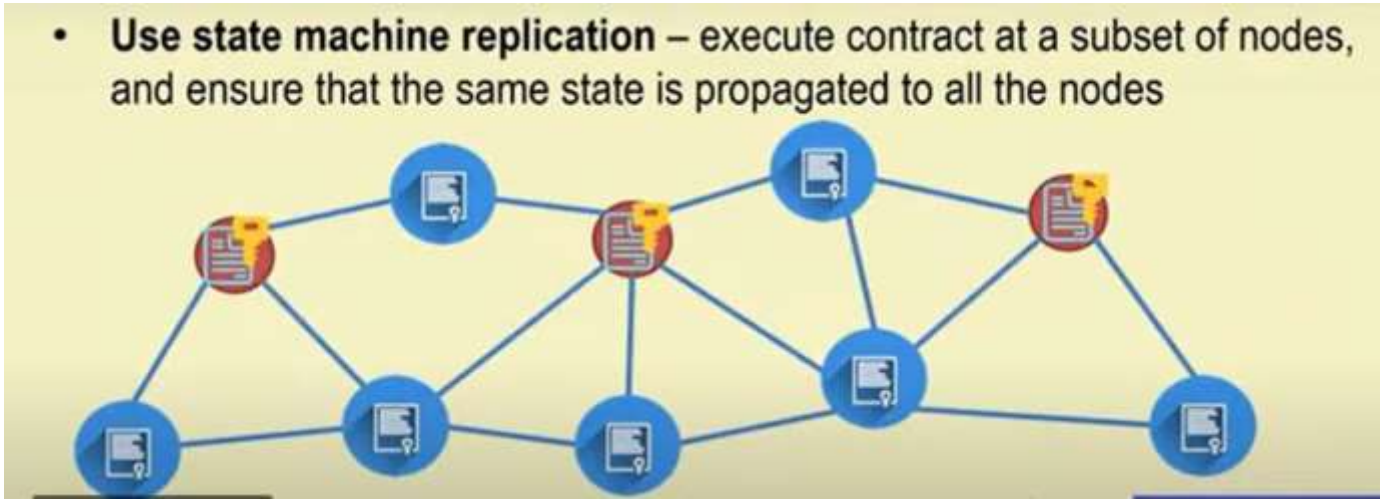
- Execute Contracts:
- It is not necessary you just need state synchronization across all the nodes.
- So it is like that you execute the contract in one node then after executing the contract in one node, you propagate the state of the contract to your neighboring node and that like state get further propagated so, that way every nodes in the system, it get same states of the contract and they can be on the same page of your smart contract.

Distributed State Machine Replication :

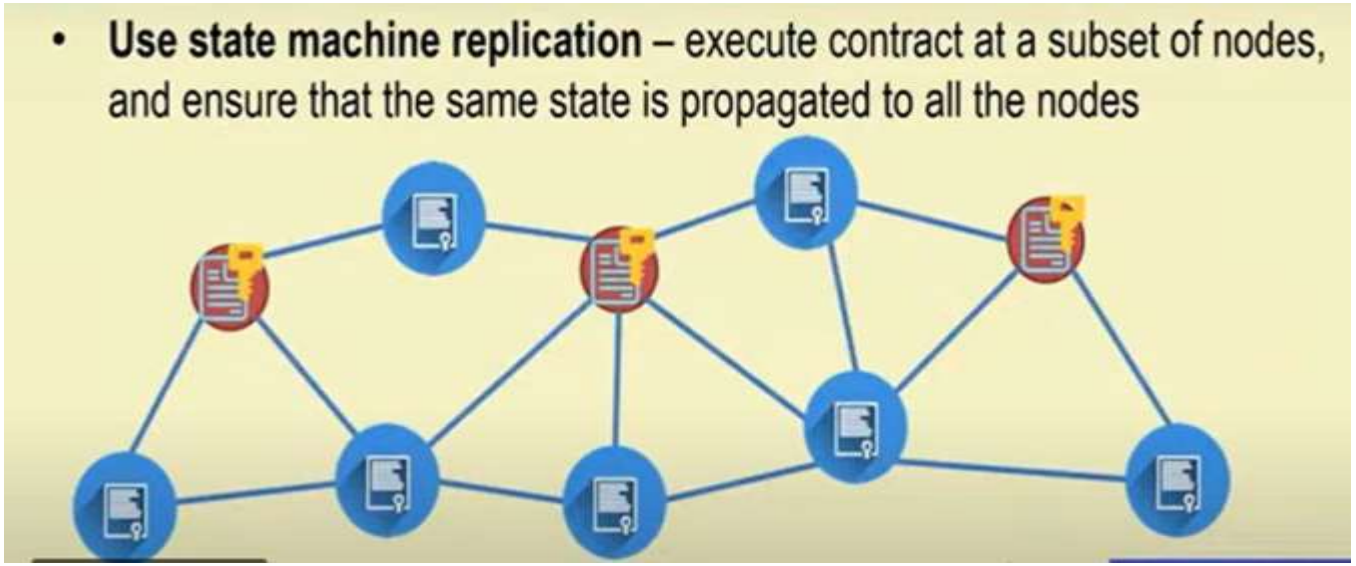
- But a typical question comes that what if the node that execute the contract it becomes faulty. So if this particular nodes becomes faulty then system gets down and the system will not be able to make any progress further. So in this particular scenario the idea is that you used the concept of state machine replication so you execute a contract at a subset of node rather than a single node.
- So you have multiple nodes which can be selected to run the particular contracts.



- Distributed State Machine Replication :
- So here this 3 nodes are executing the contract and result of the 3 nodes are propagate to all the nodes in the network.
- So that way it can ensure a kind of state machine replication through which every node that is the part of the smart contract they are on the same page and knows that upto this much of the contract got executed and remaining part need to be executed.
- State machine replication which is a powerful tool to ensure consensus in permissioned blockchain environment.

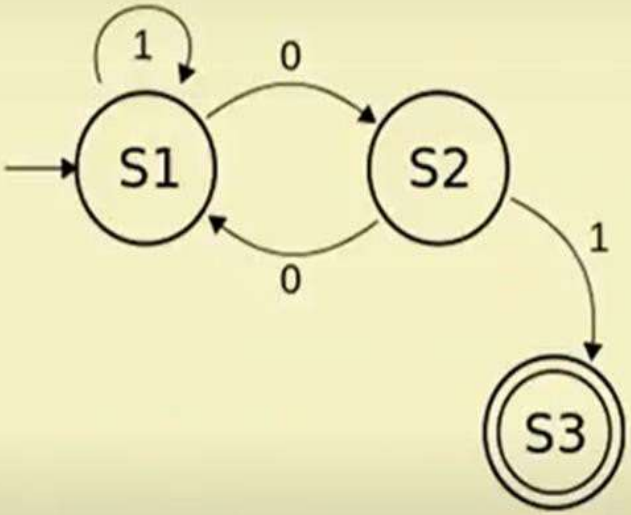


- Distributed State Machine Replication :
- The concept of State machine replication (do not need to execute a smart contract to all the nodes rather can execute it on subset of nodes and then ensure that the state of the contract is getting propagated to all the nodes in the network and there are certain consensus mechanism which will ensure that the state which have been propagated by multiple state machine or the contract execute that are on the same page or are indeed correct) help to achieve the consensus in a permissioned model so it is a powerful tool to ensure consensus in permissioned blockchain environment.

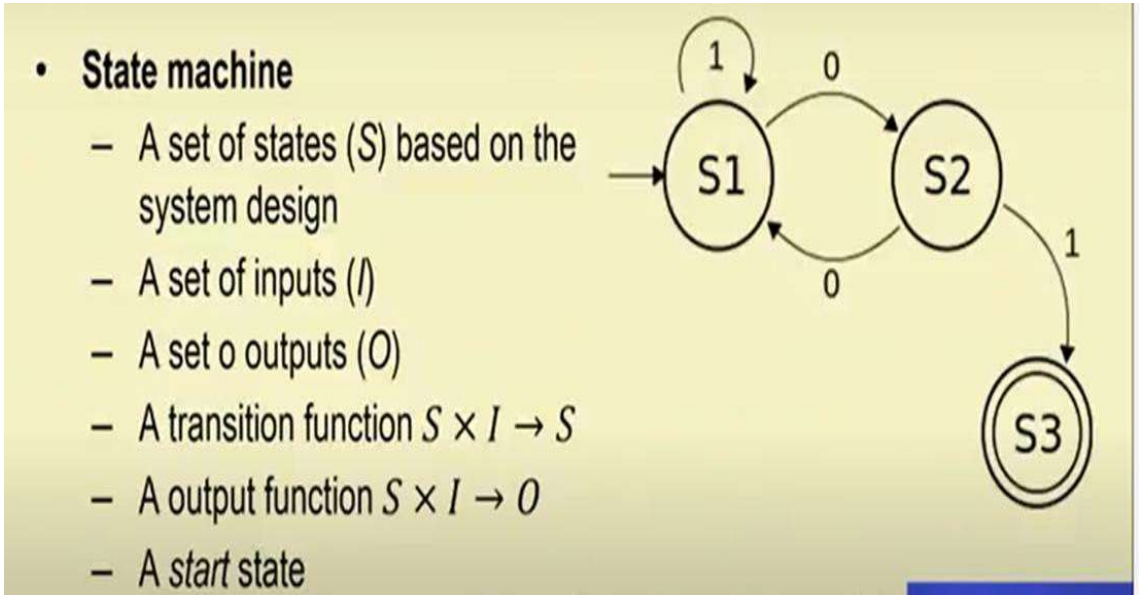


- Distributed State Machine Replication :
- So by applying this kind of distributed State machine replication technology ensure the consensus in a permissioned blockchain environment.
- State machine replication can be characterized by set of parameters. So the set of parameters are set of states based on the system design. So here in this particular example you have 3 states S1, S2 and S3. Then you have set of inputs which will tell you about how the system will behave.
-

- **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

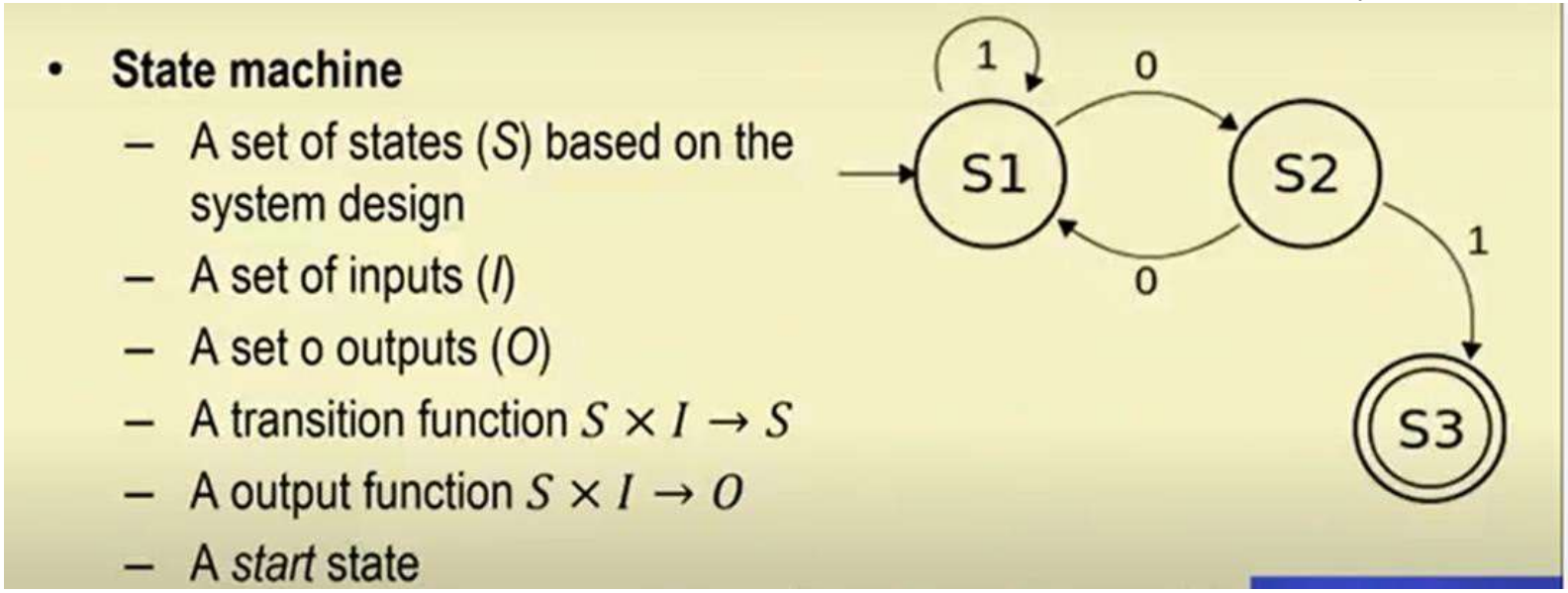


- Distributed State Machine Replication :
- So here there are 2 inputs. 0 can be input to the system and 1 can be another input to the system. And then you have set of outputs. Here S3 is final output of the system that is represented by the state machine.
- A transition function which will take a state and input, it will produce a set as the output.
- So here from S1 if you take 0 as an input it produces state S2 as output.
- (S1,0)-> S2 then output function to produce output of the system.
- And S1 as designated as
- Start state.



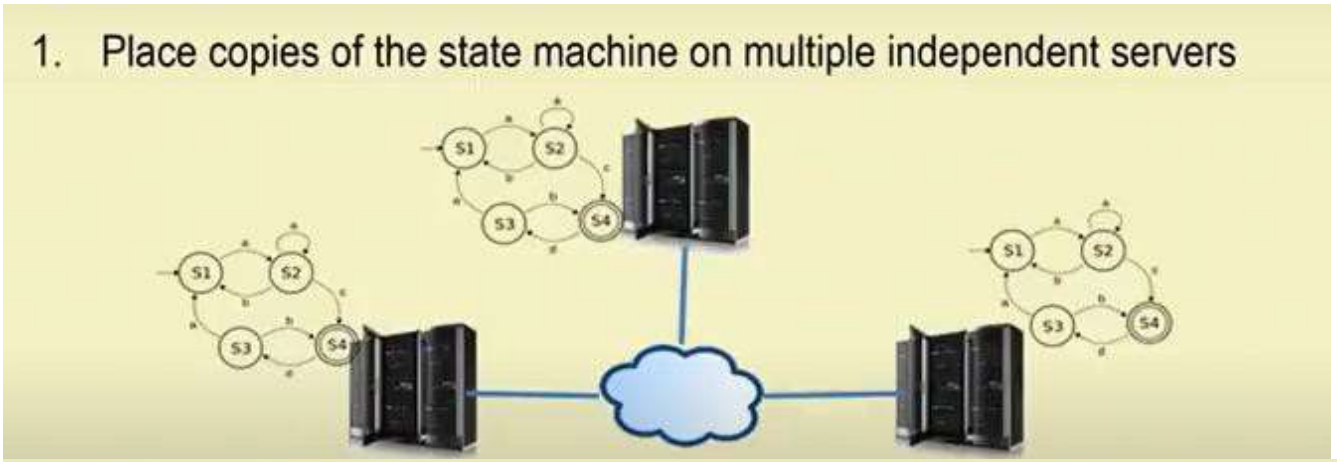
Distributed State Machine Replication :

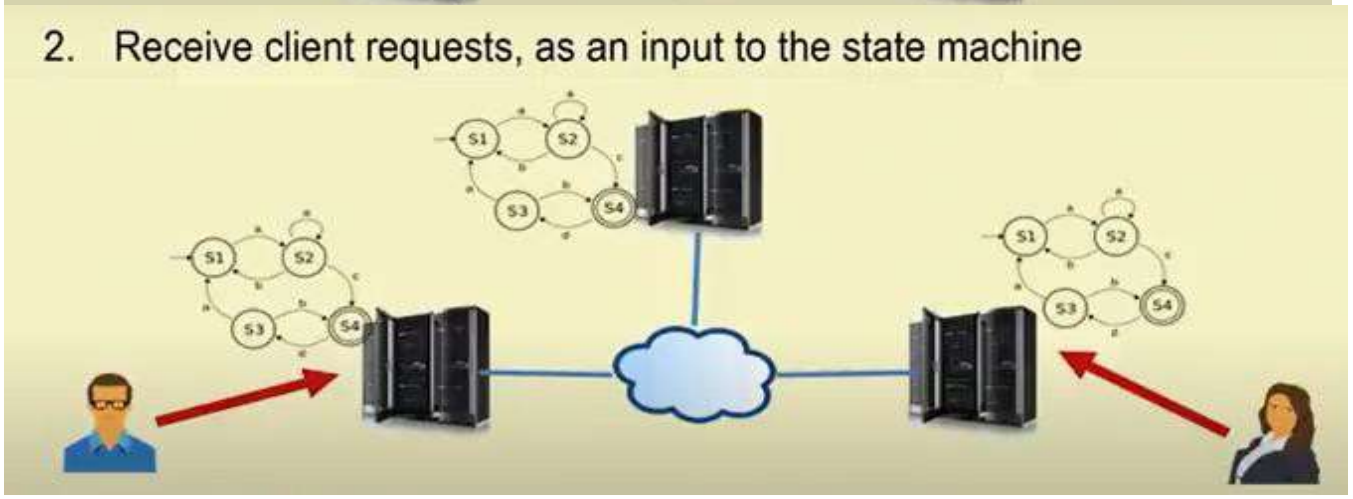
- So here there are 2 inputs. 0 can be input to the system and 1 can be another input to the system. And then you have set of outputs. Here S3 is final output of the system that is represented by the state machine.
- A transition function which will take a state and input, it will produce a set as the output.
- So here from S1 if you take 0 as an input it produces state S2 as output.
- (S1,0)-> S2 then output function to produce output of the system.



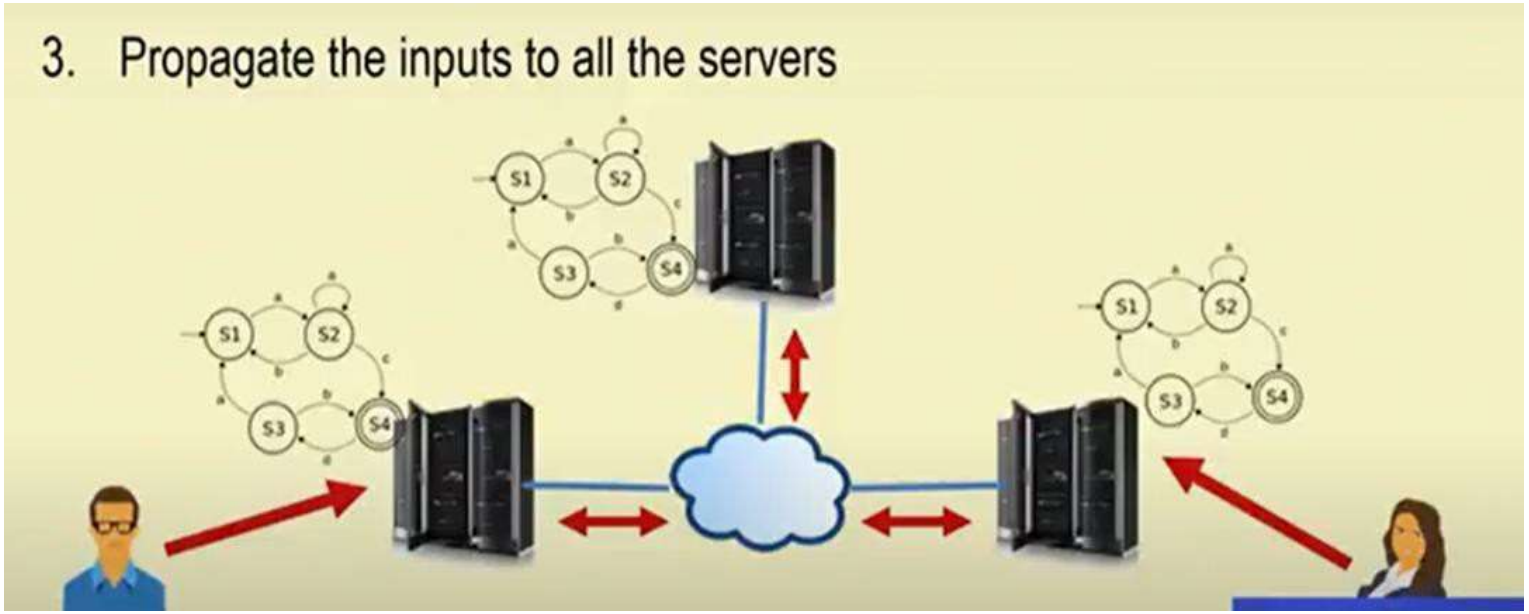
Distributed consensus in closed environment:

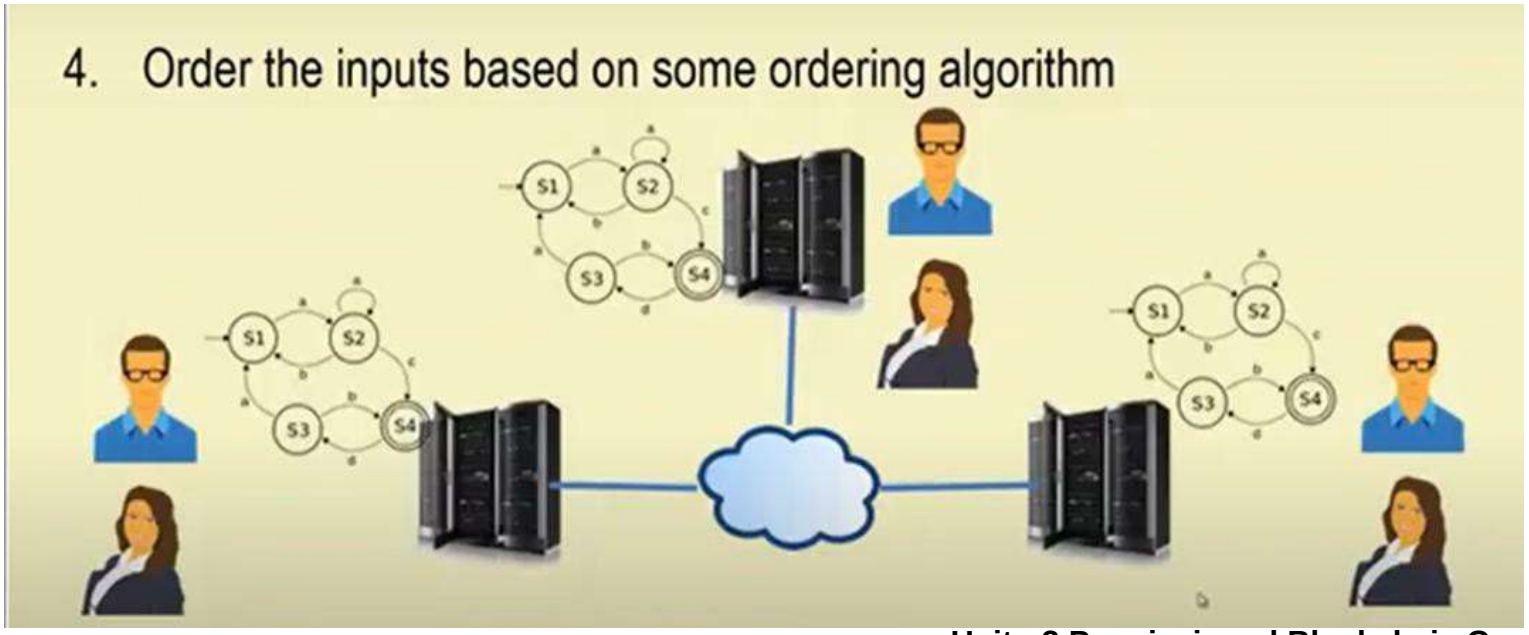
- In a typical distributed architecture, the distributed state machine replication mechanism works in following way. It have multiple servers which works in a distributed fashion.





Distributed consensus in closed environment:

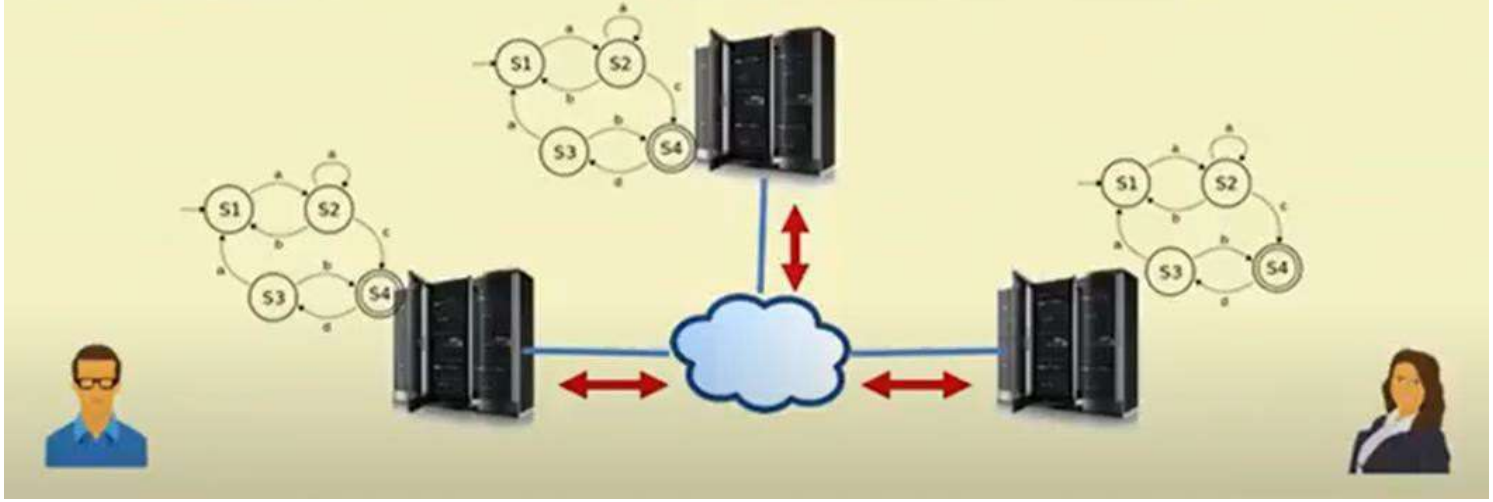




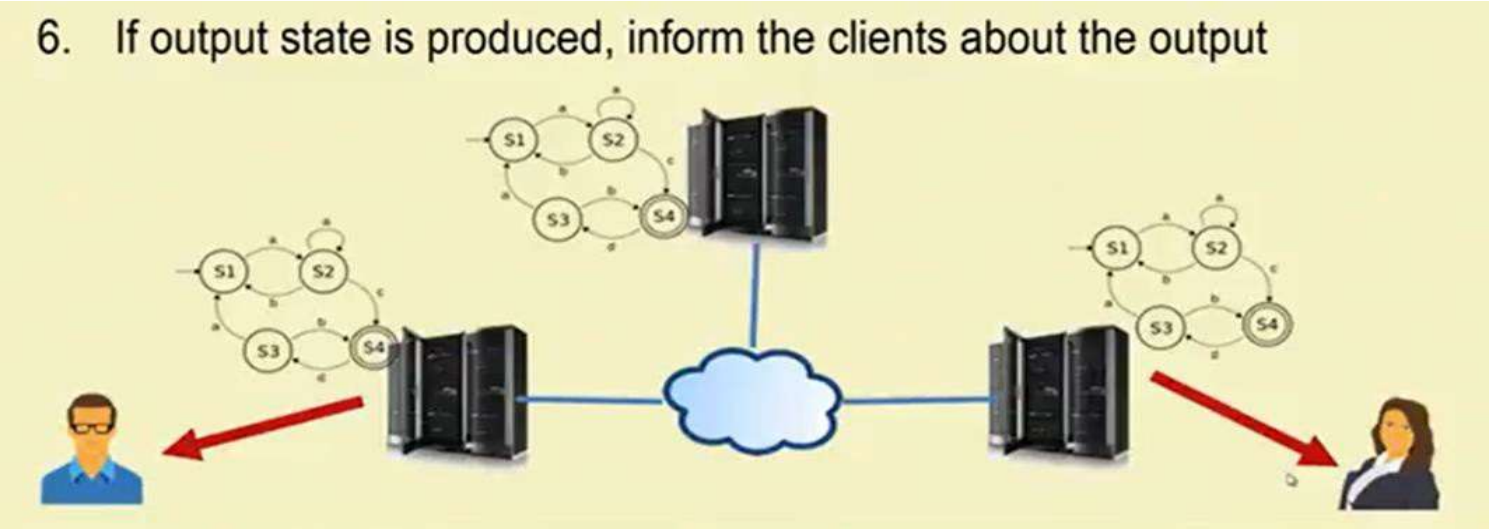
Distributed consensus in closed environment:

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

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

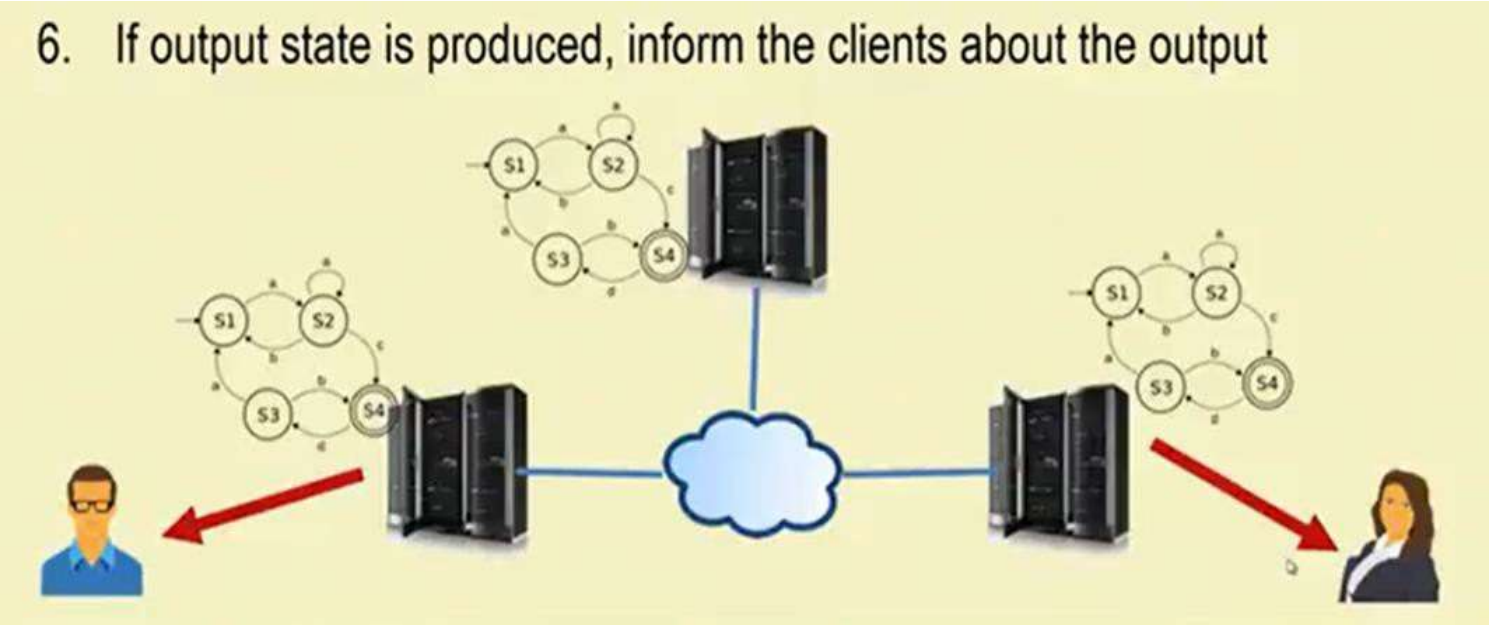


- Distributed consensus in closed environment:



- In this entire algorithm it have certain outputs that can be produced.
- Output is when both the share of the money has been transferred between Bob and Alice. You send a output or you notified the client that well the job is ready to get executed.
- So that particular output is send from server to the individual users.
- Now in this entire procedure there are 2 glitches. The 1st one is that you need to maintain order in service.

- Distributed consensus in closed environment:



- 2nd is that in presence of failure you need to ensure that all the individual servers are on the same page, everyone knows each other well.
- Both the transactions from Bob and Alice they got executed and the entire money has been transferred.

Why Distributed consensus:

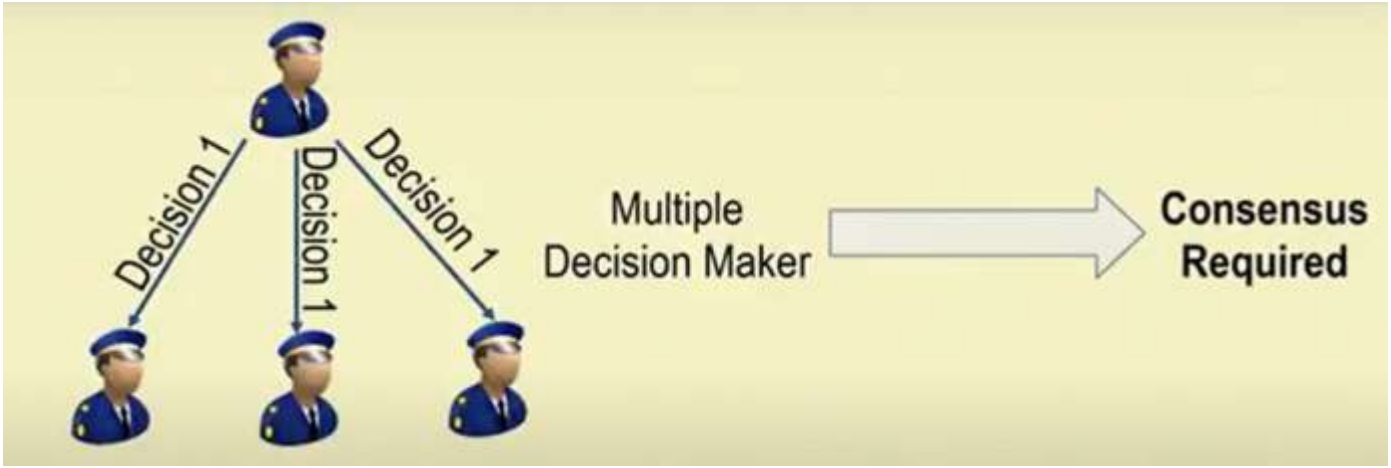
- Why do we apply for state machine replication based consensus algorithm in permissioned model in contrast to challenge response based consensus model apply for permissioned less settings?

- 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

- So if you know that who are yours peers you can always replicate your state machine with the current state to your peers.

- Why Distributed consensus:
- In case of consensus algorithm if there is only single decision maker you don't require any consensus.

- Whenever there are multiple decision makers and in a collective way they want to come to certain decision then you require the consensus.

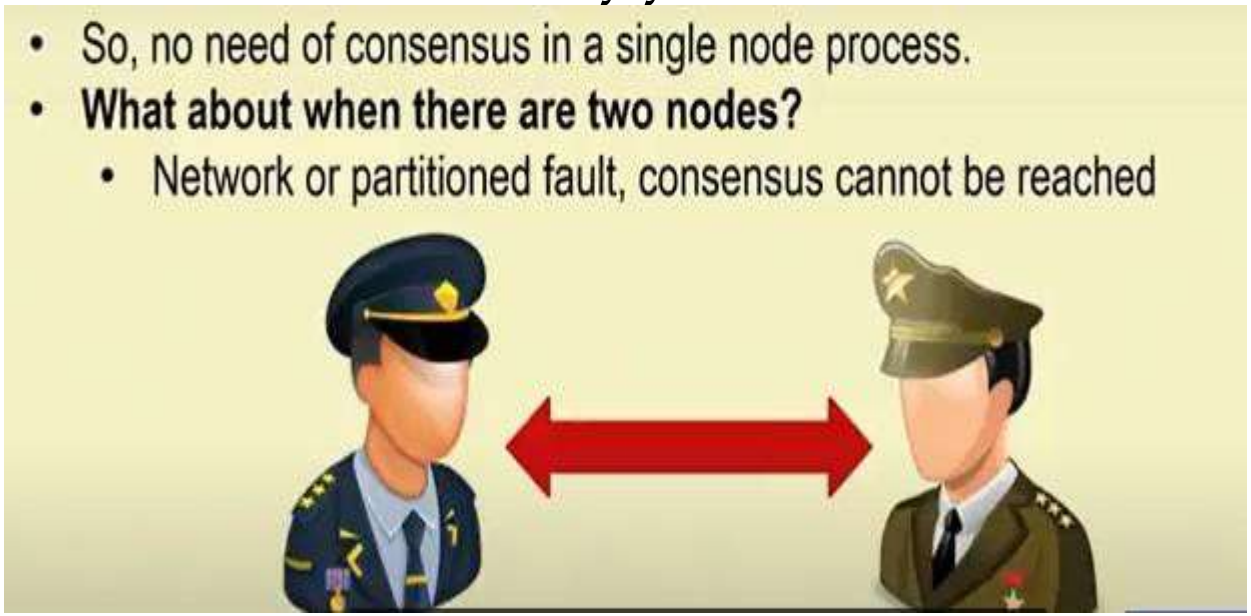


- Why Distributed consensus:
- So the distributed consensus it helps you to reach in a agreement in case of distributed computing. So in terms of state machine replication concept you replicate the common states.
- So that all the have the same view of the state, they can understand.

- Reaching agreement in distributed computing
- Replication of common state so that all processes have same view
- Applications:
  - Flight control system: E.g. Boeing 777 and 787
  - Fund transferring system: Bitcoin and cryptocurrencies
  - Leader election/Mutual Exclusion

- Flight Control Systems- when there are multiple flights and they want to coordinate their position among themselves, you can apply this kind of state machine replication technology and distributed system to achieve consensus.

- Why Distributed consensus:
- For this kind of agreement protocols where the nodes collectively needs to come to certain agreement, we require this kind of distributed consensus algorithm.
- We have looked into that we do not need a consensus in single node process but when there are 2 nodes, so in case of 2 nodes, if there is kind of fault, if the network is fault or even if there is a kind of crash fault or even if the node behaves maliciously you cannot reach consensus.
- To reach consensus, you always require more than 2 nodes.

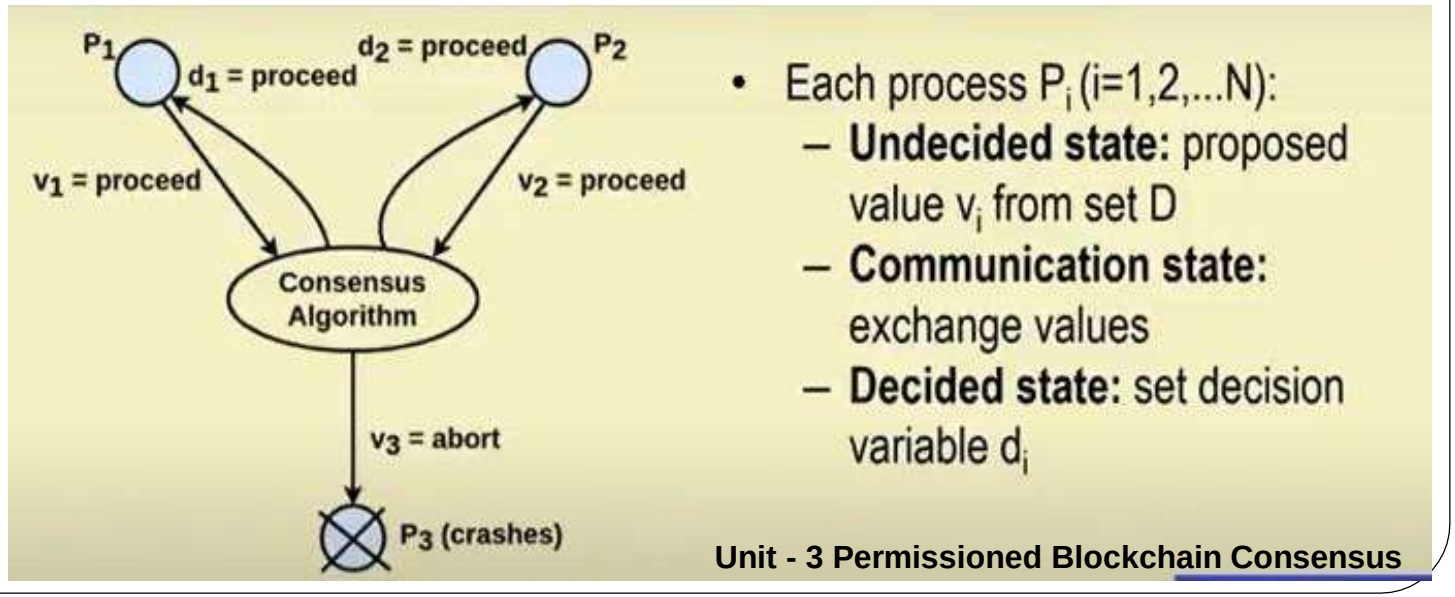


- Faults in Distributed consensus:
- In distributed environment you can have primarily 3 types of faults
- Crash Fault:
- Where a node crashes and not able to send message. It may recovered after some time and it may start sending the message again.
- So these call as fail stop behavior.
- Network or Partition Faults :
- Where because of network fault entire network gets partitioned and the message from one partition is not able to get propagated to another partition.

- Faults in Distributed consensus:
- Byzantine Faults :
- 3rd type of fault is difficult to handle in a distributed environment is Byzantine fault. So you have kind of malicious behavior among the nodes, which may come due to certain kind of hardware or software faults.
- So, this byzantine fault is that for certain views, the node is behaving in one way whereas, for certain other views the node is behaving in a different way.

- **Crash Fault**
- **Network or Partitioned Faults**
- **Byzantine Faults**
  - malicious behaviour in nodes
  - hardware fault
  - software error

- Consensus for 3 processes:
- To reach the consensus for 3 process, every process can have in 1 of 3 states, Undecided state : In undecided state the process or the node has proposed certain value from a set of feasible value D. so every node has proposed some value and they are in undecided state.
- Communication state : Then in communication state they are exchanging the value among themselves.
- Decided state : So by exchanging the value and then by applying the consensus algorithm then can reach to decided state where they set the decision that everyone in the network they agree under certain variable Di.



- Requirements of Consensus Algorithm:

- Termination: Eventually each correct process sets its decision variable. So it terminates after setting the decision variable.

- Agreement: The decision value of all correct processes is the same. Everyone should reach to the common agreement.

- Integrity: If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

- So if every correct process propose one value x. so at the decided state they should decide on the same value x.

- Different Algorithms:
- We have different kind of algorithms for ensuring consensus in atypical distributed system which we apply in permissioned blockchain.
- This algorithms based on the state machine replication principal.
- So the initial algorithms for distributed consensus are
- Paxos
- RAFT

- **Crash or Network Faults:**
  - PAXOS
  - RAFT
- **Byzantine Faults (including Crash or Network Failures):**
  - Byzantine fault tolerance (BFT)
  - Practical Byzantine Fault Tolerance (PBFT)

- Which support crash of network faults but they can not support byzantine fault so to support byzantine fault we have algorithms like