Blockchain Course

Tendermint

fausto.spoto@univr.it

https://github.com/spoto/blockchain-course

Proof of. . .

Who decides the next block?

Proof-of-work [PoW] is expensive and leads to forks

- proof-of-stake [PoS] (who owns the most)
- proof-of-space (who consumed more memory)
- proof-of-authority (who has more authority)
- . . .

PoS is a variant of Practical Byzantine Fault Tolerance (BFT)

Miguel Castro and Barbara Liskov. *Practical Byzantine Fault Tolerance and Proactive Recovery*. ACM Trans. Comput. Syst., 20(4):398–461, November 2002

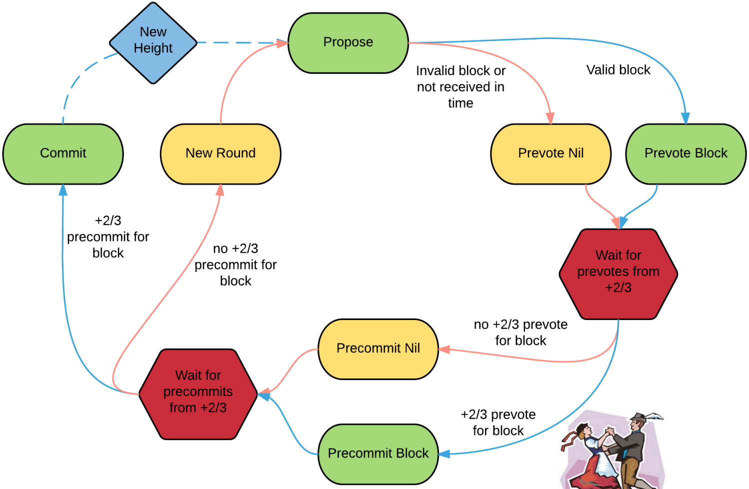
Tendermint

Jae Kwon. *Tendermint: Consensus without Mining*, 2014.
`https://tendermint.com/static/docs/tendermint.pdf`

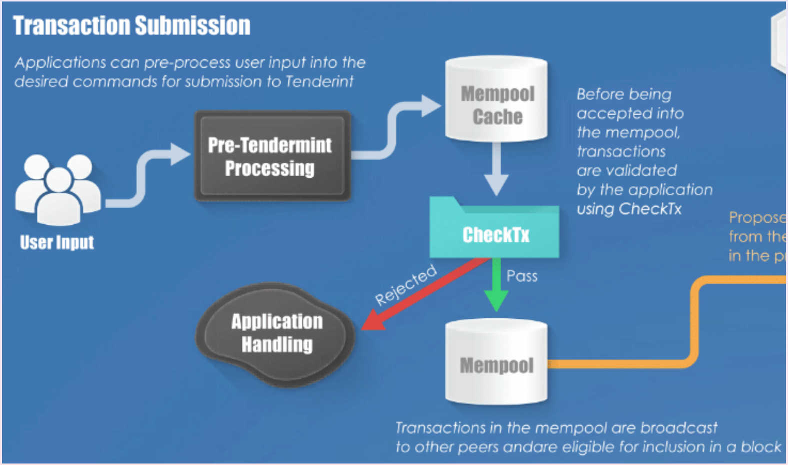
- a dynamic set V of validators decides the next block
- V might be different for each block
  - but deterministically computed from the previous history
- at each height H, each validator $v \in V$:
  1. identifies (deterministically) a validator $p \in V$ that is expected to aggregate some transactions and propose a next block b
  2. if it considers b valid, it pre-votes b
  3. counts how many validators pre-vote b
  4. if it counted at least $\frac{2}{3}$ pre-votes, it pre-commits b
  5. counts how many validators pre-commit b
  6. if it counted at least $\frac{2}{3}$ pre-commits, it commits b and increases H
  7. goes back to step 1

Tendermint is BFT. If step 1 is based on stakes, then it is PoS

Tendermint's polka

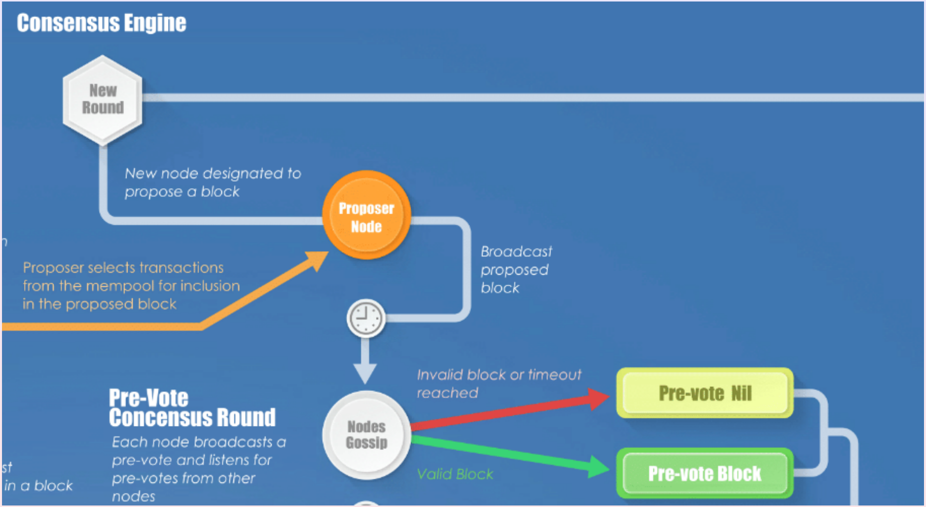

New Height
Propose
Invalid block or not received in time
Valid block
Commit
New Round
Prevote Nil
Prevote Block
+2/3 precommit for block
no +2/3 precommit for block
Wait for prevotes from +2/3
no +2/3 prevote for block
Precommit Nil
Wait for precommits from +2/3
+2/3 prevote for block
Precommit Block

The mempool


Transaction Submission
Applications can pre-process user input into the desired commands for submission to Tenderint
User Input
Pre-Tendermint Processing
Mempool Cache
Before being accepted into the mempool, transactions are validated by the application using CheckTx
CheckTx
Rejected
Pass
Application Handling
Mempool
Propose
from the
in the p
Transactions in the mempool are broadcast to other peers andare eligible for inclusion in a block

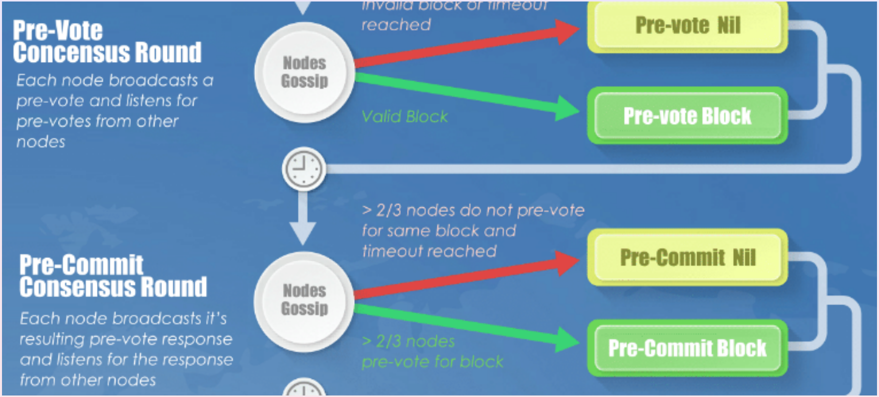
`checkTx(tx)` checks if the transaction `tx` is valid

Pre-vote

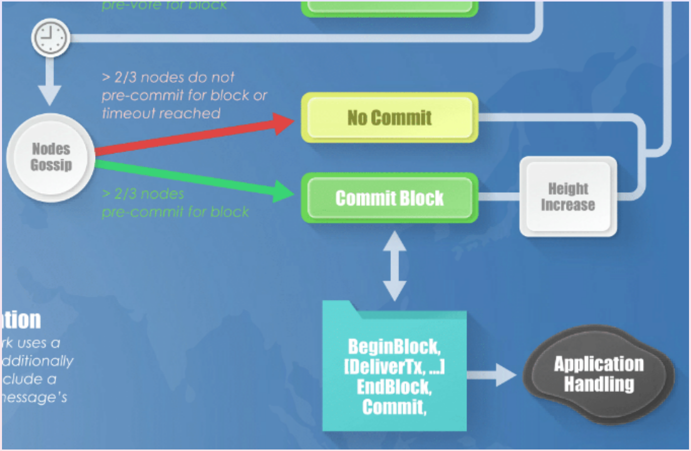

Consensus Engine
New Round
New node designated to propose a block
Proposer Node
Proposer selects transactions from the mempool for inclusion in the proposed block
Broadcast proposed block
Invalid block or timeout reached
Pre-vote Nil
Pre-Vote Concensus Round
Each node broadcasts a pre-vote and listens for pre-votes from other nodes
Nodes Gossip
Valid Block
Pre-vote Block
st
in a block

`checkTx(tx)` again to check if the transactions in the block are valid

Pre-commit

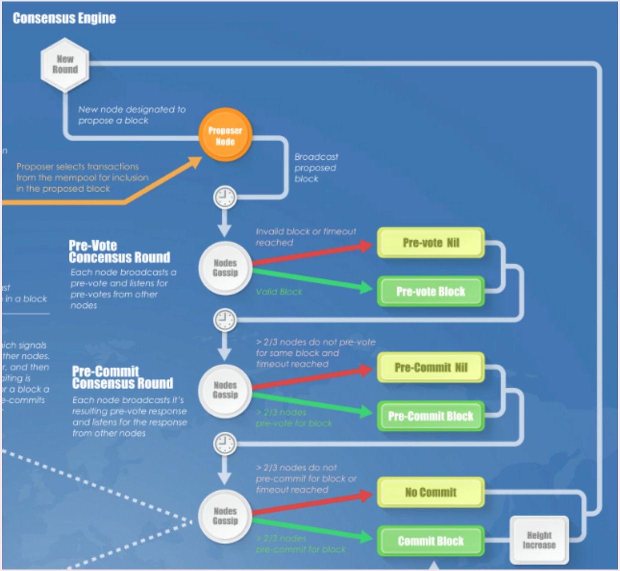

Pre-Vote Concensus Round
Each node broadcasts a pre-vote and listens for pre-votes from other nodes
Nodes Gossip
invalid block or timeout reached
Pre-vote Nil
Valid Block
Pre-vote Block
Pre-Commit Consensus Round
Each node broadcasts it's resulting pre-vote response and listens for the response from other nodes
Nodes Gossip
> 2/3 nodes do not pre-vote for same block and timeout reached
Pre-Commit Nil
> 2/3 nodes pre-vote for block
Pre-Commit Block

Commit

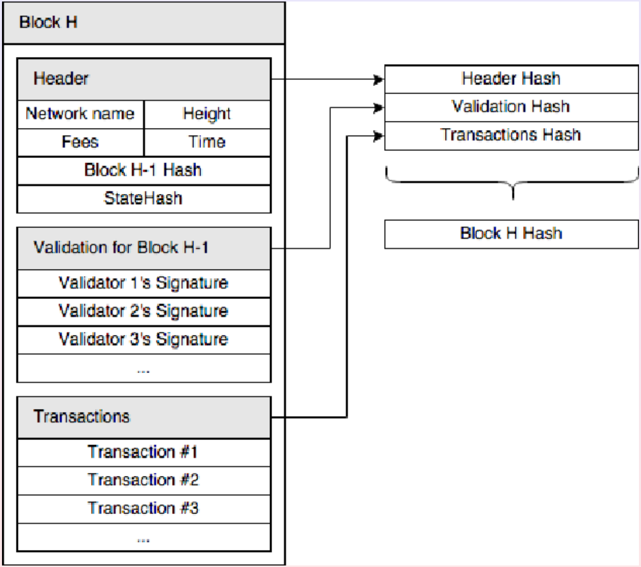

pre-vote for block
> 2/3 nodes do not pre-commit for block or timeout reached
No Commit
Nodes Gossip
> 2/3 nodes pre-commit for block
Commit Block
Height Increase
tion
rk uses a
dditionally
clude a
essage's
BeginBlock, [DeliverTx, ...] EndBlock, Commit,
Application Handling

`deliverTx(tx)` executes the transaction `tx`, hence modifies the state

Next round, or next height


Consensus Engine
New Round
New node designated to propose a block
Proposer Node
Proposer selects transactions from the mempool for inclusion in the proposed block
Broadcast proposed block
Pre-Vote Concensus Round
Each node broadcasts a pre-vote and listens for pre-votes from other nodes
Nodes Gossip
Invalid block or timeout reached
Pre-vote Nil
Valid Block
Pre-vote Block
st
in a block
ich signals
ther nodes.
, and then
iting is
r a block a
-commits
Pre-Commit Consensus Round
Each node broadcasts it's resulting pre-vote response and listens for the response from other nodes
Nodes Gossip
> 2/3 nodes do not pre-vote for same block and timeout reached
Pre-Commit Nil
> 2/3 nodes pre-vote for block
Pre-Commit Block
Nodes Gossip
> 2/3 nodes do not pre-commit for block or timeout reached
No Commit
> 2/3 nodes pre-commit for block
Commit Block
Height Increase

Inside a Tendermint block



Block H

Header

Network name	Height
Fees	Time

Block H-1 Hash

StateHash

Validation for Block H-1

Validator 1's Signature

Validator 2's Signature

Validator 3's Signature

...

Transactions

Transaction #1

Transaction #2

Transaction #3

...

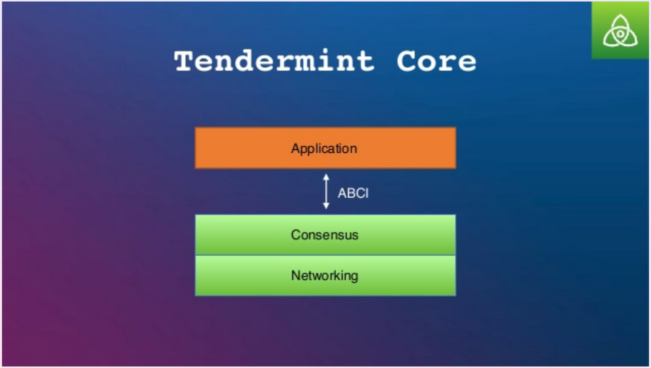
Header Hash

Validation Hash

Transactions Hash

Block H Hash

A layered implementation in Golang



Tendermint Core

Application

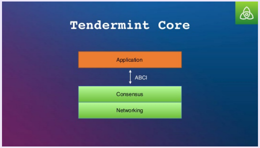
ABCI

Consensus

Networking

ABCI: Application BlockChain Interface

The application layer



Tendermint Core

Application

ABCI

Consensus

Networking

The application layer is not part of Tendermint Core

Programmers can write their own application layer

- connected to Tendermint Core via ABCI using sockets
- possibly on a different machine than Tendermint Core
- in any programming language

The application layer must be deterministic!

The ABCI

`https://docs.tendermint.com/master/spec/abci/abci.html`

`checkTx`: called before entering the mempool and to verify blocks

⇒ only transactions that satisfy `checkTx` are added in blocks

⊗ must not modify the state of the application

`beginBlock`: called at the beginning of a block; receives information about the validator set of the previous block and which of them signed the previous block

`deliverTx`: called for each transaction added to a block: it executes the transaction by modifying the state of the application

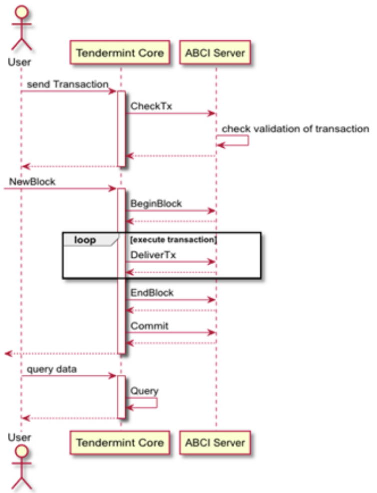
`endBlock`: called at the end of a block; provides information about the validator set for the next block

`commit`: called when a block is being committed; provides the hash of the state of the application

---

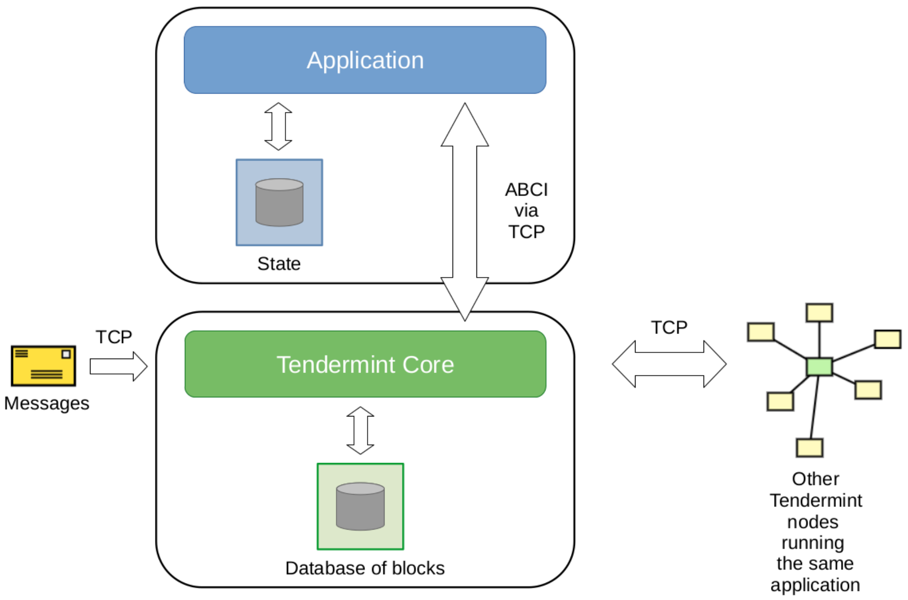
`query`: called when the user wants to read data from the blockchain

The ABCI



State updates between `beginBlock` and `commit` must be seen as a single atomic update of the application state

The database of blocks and the application state



The application state

It must have a function to compute its hash

Only that hash is reported in blockchain, for consensus

It must allow transactional, atomic updates

Between `beginBlock` and `commit`

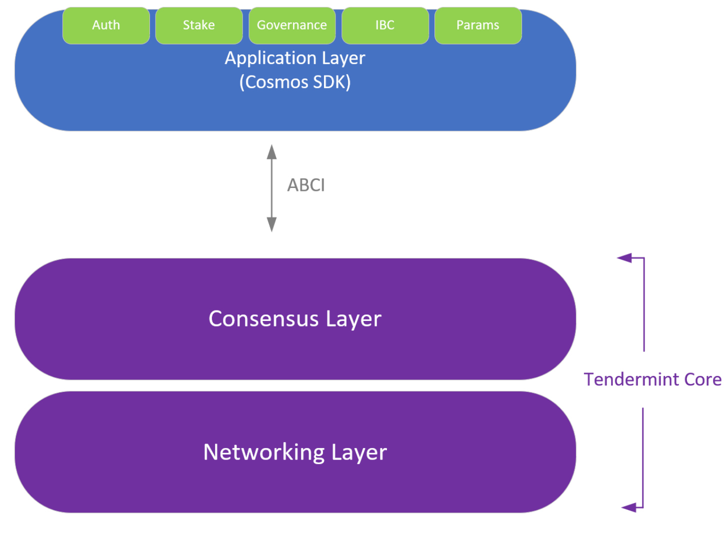
The API of the state

Tendermint enjoys finality: there are no forks

⇒ one never needs to come back in time to the state of a previous block

1. get data
2. put data
3. `h=get_hash()`
4. ~~`checkout(h)`~~ ⇒ big opportunity for garbage collection!

Cosmos: a Tendermint application in Golang



The application state of Cosmos

Cosmos keeps data inside keepers

They are maps *key* → *value*. Programmers must store persistent data inside a keeper: all other data is lost if the node is turned off and on again

Golang

Cosmos can be expanded with arbitrary Golang code, but:

- it must be deterministic
- it must store persistent data in a keeper
- it must count gas consumption explicitly

Smart contracts?

There are no smart contracts in Cosmos, really. The system as a whole is a big (fixed) smart contract

⇒ maintenance issue