Cothority/ByzCoin

Elements from implementing a scalable blockchain Linus.Gasser@epfl.ch / DEDIS

Overview

- Presentation Cothority / Conodes
- Our ByzCoin implementation
- Hands-on

Users of our Code



Industry /
Commercial

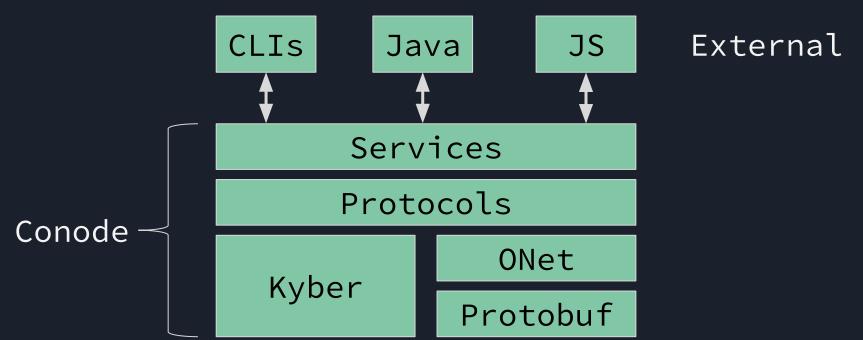
PhD / Researchers Semester students

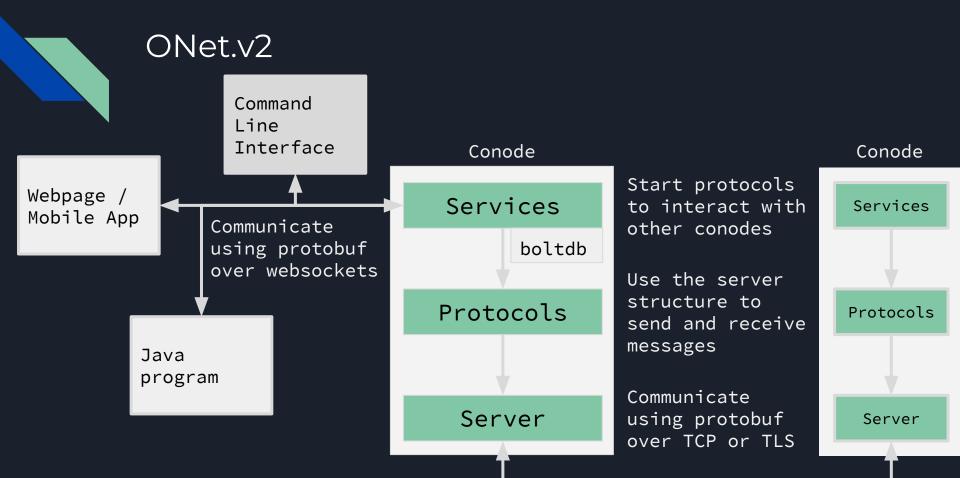




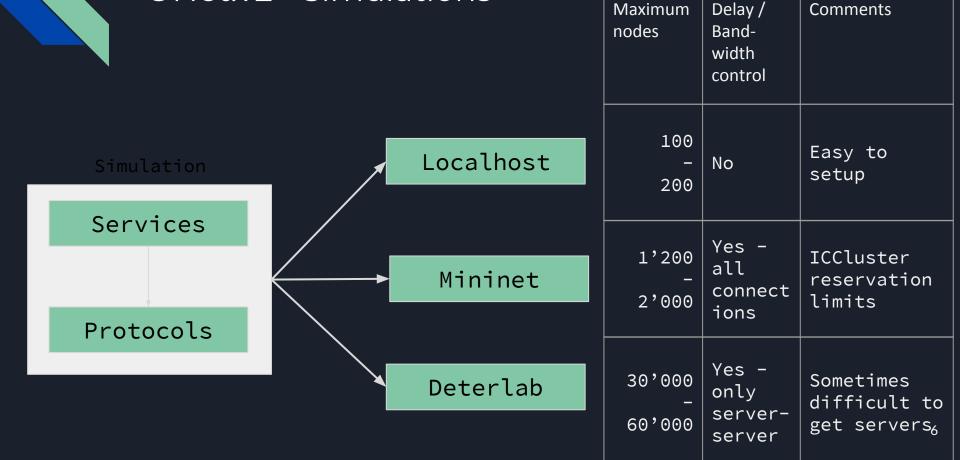
General Public

Library-overview - V2

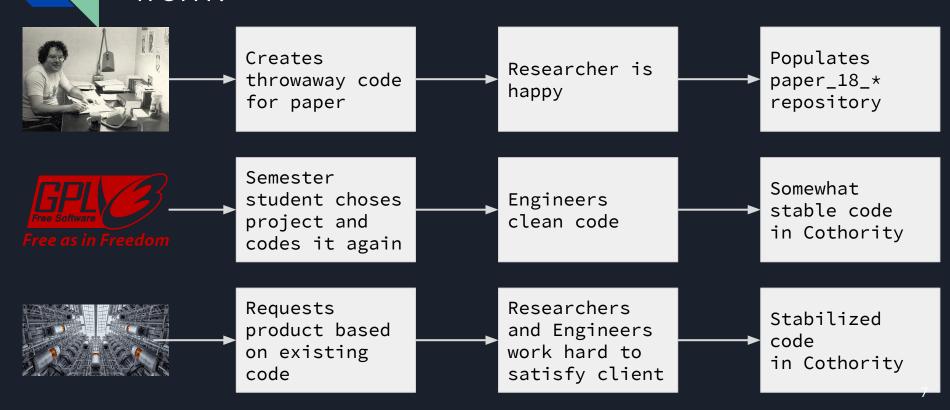




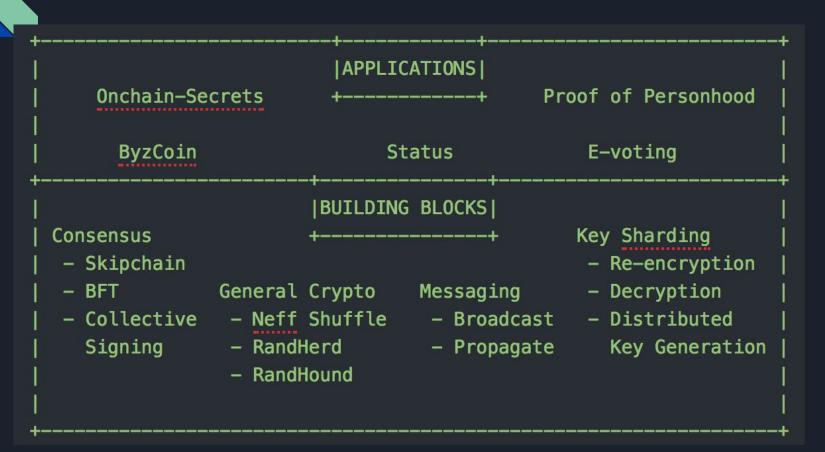
ONet.v2 - Simulations



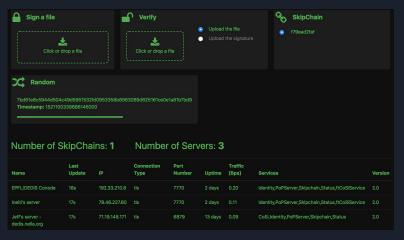
Cothority.v2 - Where Does the Code Come from?



Cothority.v2 - Overview



Cothority.v2 - GUIs

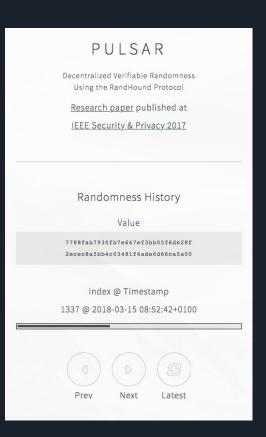






http://status.dedis.ch

http://personhood.online



https://pulsar.dedis.ch

Cothority.v3 - Roadmap

- Implement sharded ByzCoin: Omniledger
- More testing in real-world deployments
- Add semi-permissioned onboarding for nodes: proof-of-personhood
- Add EVM contract to run Ethereum-compatible smart contracts
- Calypso encrypted storage on the blockchain

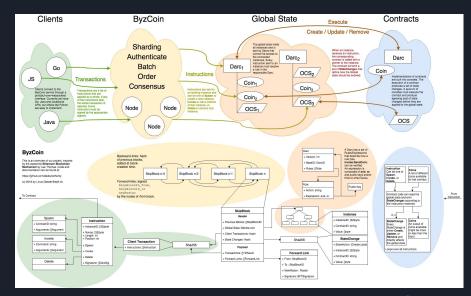
Overview

- Presentation Cothority / Conodes
- Our ByzCoin implementation
- Hands-on

PhD paper -> usable software

```
// verifyBlock is a simulation of a
real block verification algorithm
// FIXME merge with Nicolas' code (public method in byzcoin)
func verifyBlock(block *blockchain.TrBlock, lastBlock, lastKeyBlock string)
       //We measure the average block verification delays is 174ms for an
average
       //block of 500kB.
       //To simulate the verification cost of bigger blocks we multiply
174ms
       //times the size/500*1024
       b, := json.Marshal(block)
       s := len(b)
       var n time.Duration
       n = time.Duration(s / (500 * 1024))
       time.Sleep(150 *
time.Millisecond * n) //verification
of 174ms per 500KB simulated
       // verification of the header
       verified := block.Header.Parent == lastBlock &&
block.Header.ParentKey == lastKeyBlock
       verified = verified && block.Header.MerkleRoot ==
blockchain. HashRootTransactions (block. TransactionList)
```

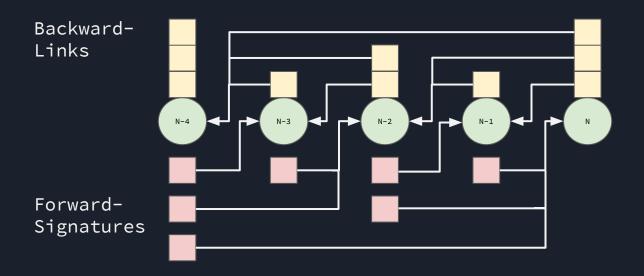
verified = verified && block.HeaderHash ==



https://raw.githubusercontent.com/dedis/cothority/master/byzcoin/ByzCoin.png

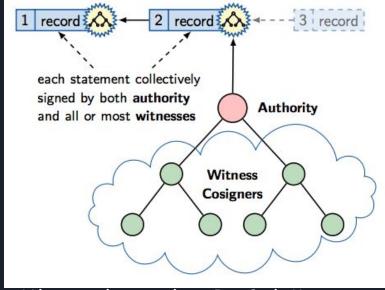
blockchain. HashHeader (block. Header)

Base for ByzCoin - 1: skipchains



https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-nikitin.pdf

Base for ByzCoin - 2: ByzCoinX



Witnessing using ByzCoinX

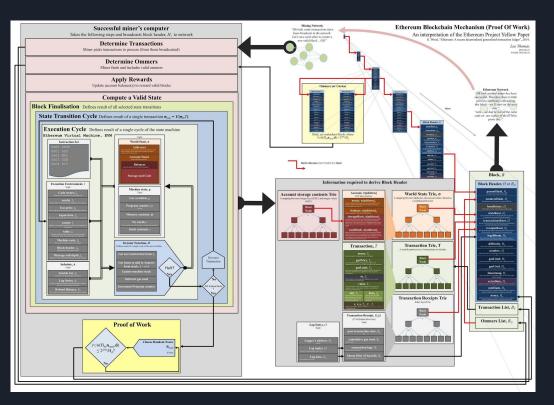
Two-round cosigning:

- prepare verification of new block
- commit creating final signature

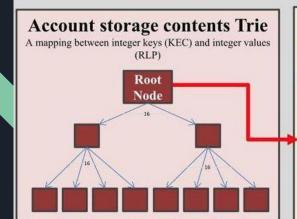
If subleaders fail:

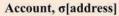
- rotate that sub-group, fast If leader fails:
 - all nodes must reach consensus on failure
 - propose next node in list
 - change is slow

Base for ByzCoin - 3: Ethereum



https://github.com/ethereum/go-ethereum





RLP data structure

nonce, σ[address],

scalar, the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

balance, σ[address],

storageRoot, σ[address],

56-bit hash of the root node of a Merkle Patricia tree that encodes integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit

codeHash, o[address]

Hash of the EVM code of this account—the code that gets executed should this address receive a message call, it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their nding hashes for later retrieval.

Transaction, T

nonce, T.,

Scalar, the number of transactions sent by the sender

gasPrice, Tp scalar, the number of Wei to be paid per unit of gas for all computation costs incurred as a result of execution of this transaction

gasLimit, T

scalar, the max amount of gas that should be used in executing this unassection. Paid before any computation is done, may not be uncreased later.

to, T_{i}

160-bit address of the message call's recipient or, for a contract creation transaction, O (zero bytes).

value, 7

to the newly created account

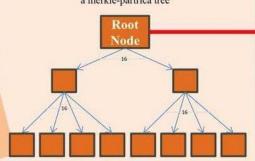
init, 7

data, T.

Outputs from EDSCA signature of $KEC(T_a, T_a, T_a, T_a, T_c, \ell T_c)$ or T_a

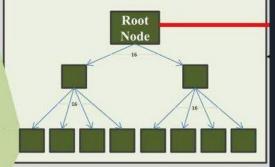
World State Trie, o

A mapping between addresses and account states. Stored as a merkle-partrica tree



Transaction Trie, T

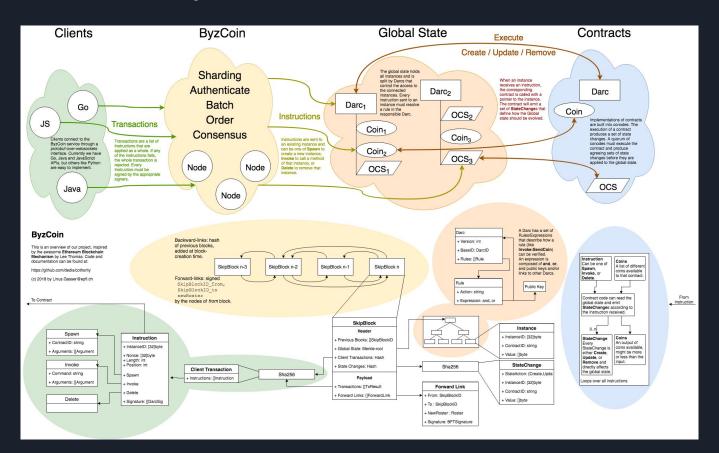
A merkle-partrica tree of transactions to include



Transaction Receipts Trie

Index keyed trie

Base for ByzCoin - 4: Overview



Overview

- Presentation Cothority / Conodes
- Our ByzCoin implementation
- Hands-on

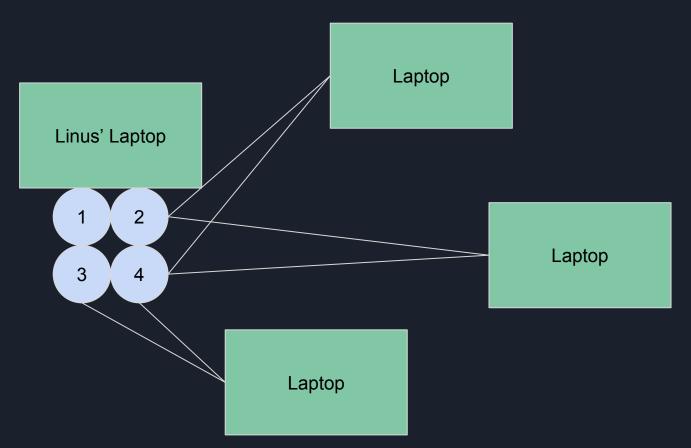
0 - Introduction

- There is a status page with more explanations here:
 - o http://192.168.0.164:8000
- And a blockchain explorer:
 - o http://192.168.0.164:8000/student 18 explorer/dist
- The following google-doc is used to exchange data:
 - https://docs.google.com/document/d/11WseidLl2pIE96aNzm-s dxANaMvB
 DO0Xe2b73JygMY/edit?usp=sharing

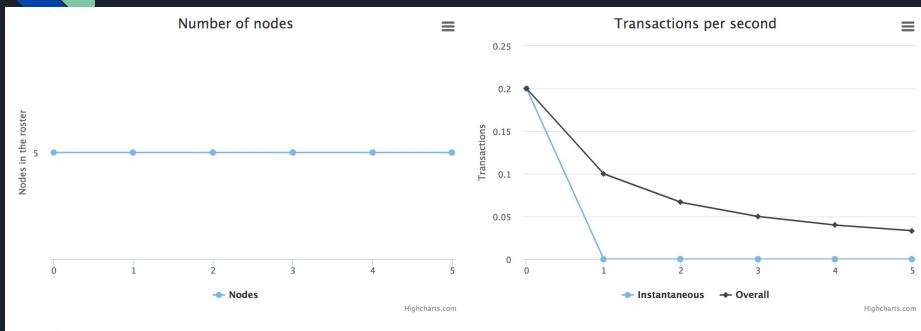
0 - Introduction to Docker Images

- You need a docker-installation from https://docker.com
- You'll be running two docker images:
 - dedis/mba-cothority for the commands
 - dedis/mba-conode for running your own server
- The commands for mba-conode will create a directory in ~/data
 - This directory will hold the private data of your conode
- Please copy your log-file from ~/data to the following directory at the end:
 - https://drive.google.com/drive/folders/1GWSvXOKOZSnTmWZ4G6Rbsrvl 3zOy9r0j?usp=sharing

Network Overvice



http://192.168.0.164:8000



Here is a list of links:

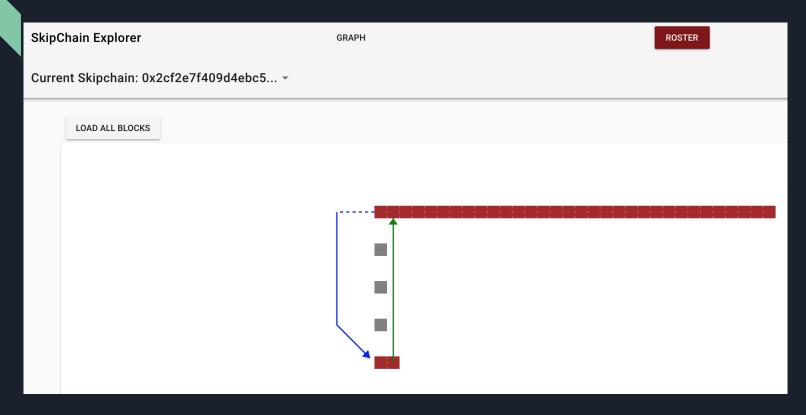
- MBA Worksheet
- Presentation
- Directory for logs

Roster is: ok - tls://192.168.0.42:7772 - Conode_1 ok - tls://192.168.0.42:7774 - Conode 2

ok - tls://192.168.0.42:7776 - Conode_3

ok - tls://192.168.0.42:7778 - Conode_4 FAIL - tls://192.168.0.42:7770 - # blocks: 2
backlinks: 1
block-size: 812

Blockchain Explorer



Google-Doc

MBA working document

Rules

Be nice - where nice is defined as making the demo as successful as possible with regard to showing off what the DEDIS/cothority can do.

If we have more than 10 participants, I want to have 90% passing.

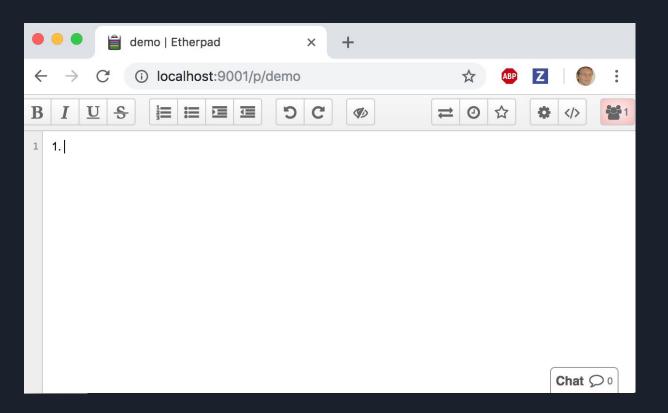
Please help others. Two rules: 1. ask if they need help first, 2. don't touch their keyboards Don't laugh when it breaks.

Participants

Please enter your name or pseudo here. For every exercise you completed, put an **X**. If you also made the bonus, put a **XX** in it.

#	Name / Pseudo	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7
1						Ţ		
2								
3								
4								

Etherpad doc



Keys for the Terminal

```
<ctrl+a> - start of the line
<ctrl+e> - end of the line
<ctrl+w> - delete to the left
<ctrl+k> - delete to end of line
<ctrl+r> - search in previous commands
<ctrl+c> - abort command or delete current command
<alt+right> - jump to next word
<alt+left> - jump to previous word
ZZ - save and quit vim
:q! - quit vim without saving
```

General Rules

- Be nice where nice is defined as making the demo as successful as possible with regard to showing off what the DEDIS/cothority can do.
- If we have more than 10 participants, I want to have 80% passing (Pareto principle).
- Please help others. Two rules:
 - 1. ask if they need help first
 - 2. don't touch their keyboards
- Don't laugh when it breaks, please.

1 - Contact a Conode

- Sign up: http://192.168.0.164:8000
- Download docker image from here:

https://hub.docker.com/r/dedis/

with the following command:

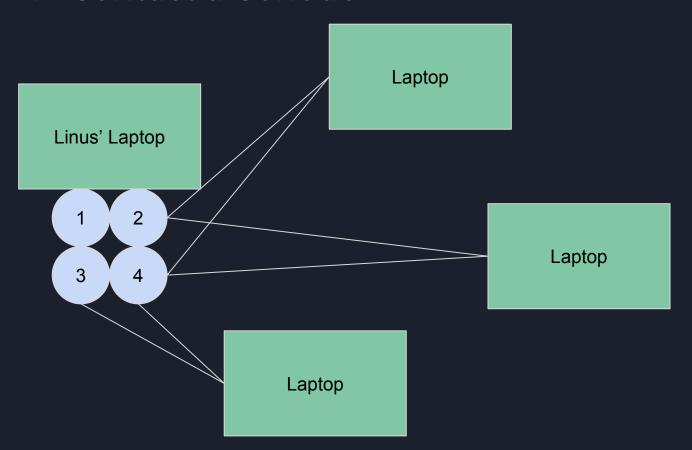
docker run -ti -v ~/data:/conode data dedis/mba-cothority

• Get status from nodes on my computer:

```
status --group public.toml
```

Bonus: get status of node at tls://192.168.0.164:7780

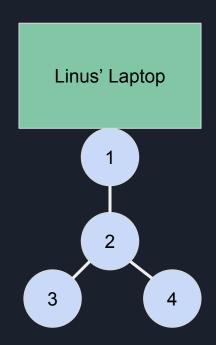
1 - Contact a Conode



2 - Fault Tolerant Collective Signatures (ftCoSi)

- Still in dedis/mba-cothority docker container, run ftcosi sign msg.txt > signature.json
- Now this file is collectively signed by all the nodes in public.toml
- To verify it, use
 ftcosi verify msg.txt < signature.json
- Bonus: include conode at tls://192.168.0.164:7780 and sign
- Bonus: verify file from other participant

ftCoSi Signature



3 - Run a Conode

Run the dedis/mba-conode docker image

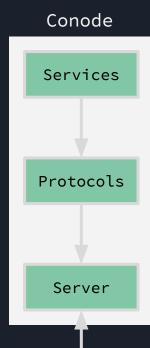
```
docker run -ti -v ~/data:/conode_data -p
7770-7771:7770-7771 dedis/mba-conode
```

- Make sure to enter your local IP address: 192.168.x.y:7770
- Test status from your dedis/mba-cothority container:

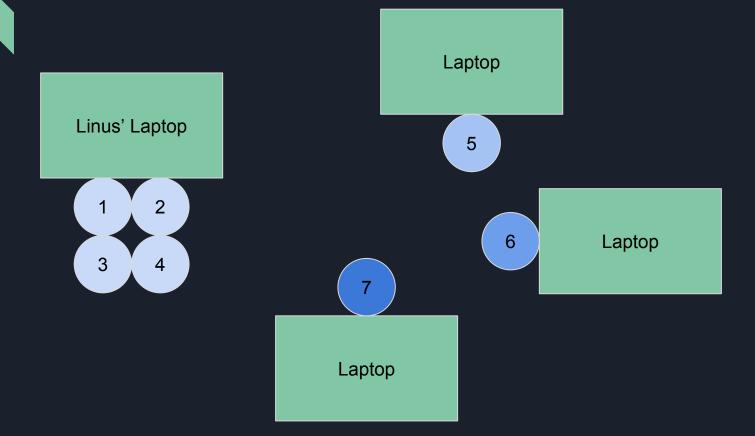
```
status --host <your_local_ip>:7770
```

• Bonus: test the status of your neighbour's node

ONet.v2 Command Line Interface Conode Webpage / Services Mobile App Communicate using protobuf over websockets Protocols Java program Server



3 - Run a Conode



4 - Join ByzCoin Ledger

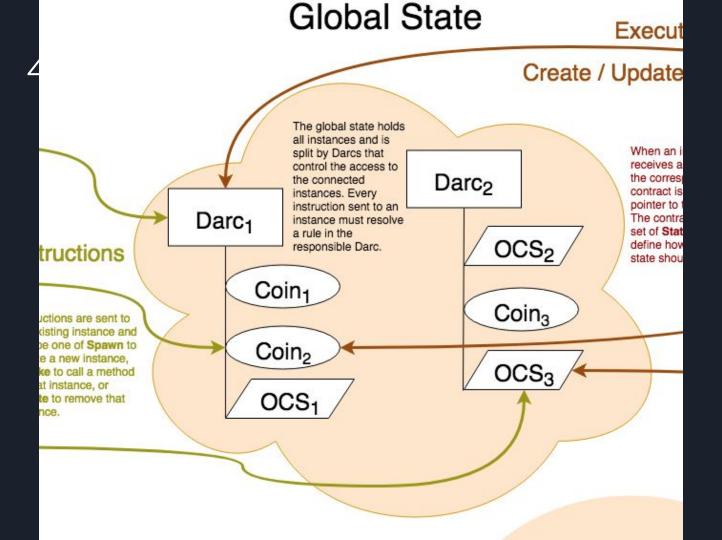
• Start by checking out the latest block from the local byzcoin ledger:

bcadmin show bc*cfg

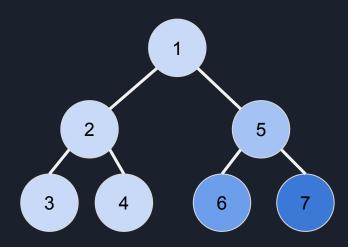
Put the content of your

```
/conode_data/public.toml -or- ~/data/public.toml on the wiki and verify periodically if we included it: bcadmin show bc*cfg
```

• Bonus: set up your own byzcoin ledger with 3 other participants



4 - Join ByzCoin



4 - Joining ByzCoin - Ideally

According to the paper:

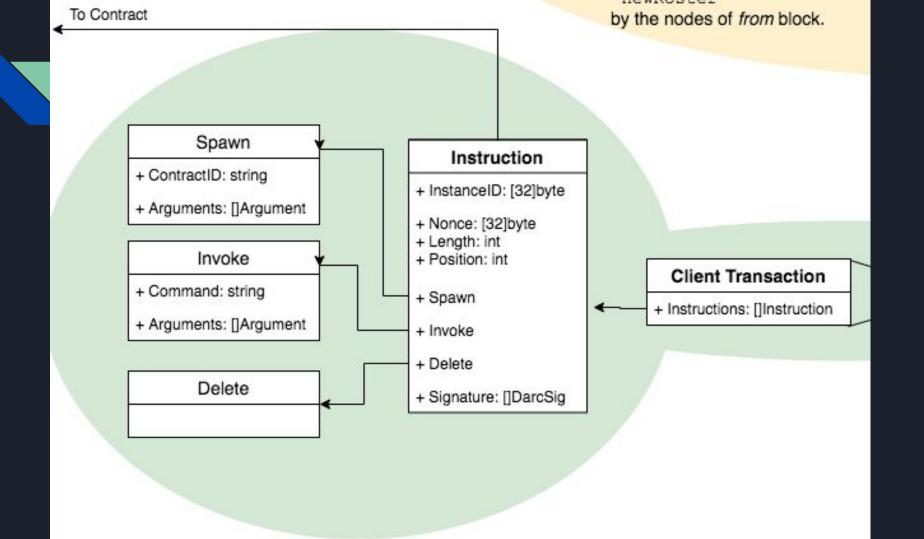
• Use a window of mined blocks to define the participants

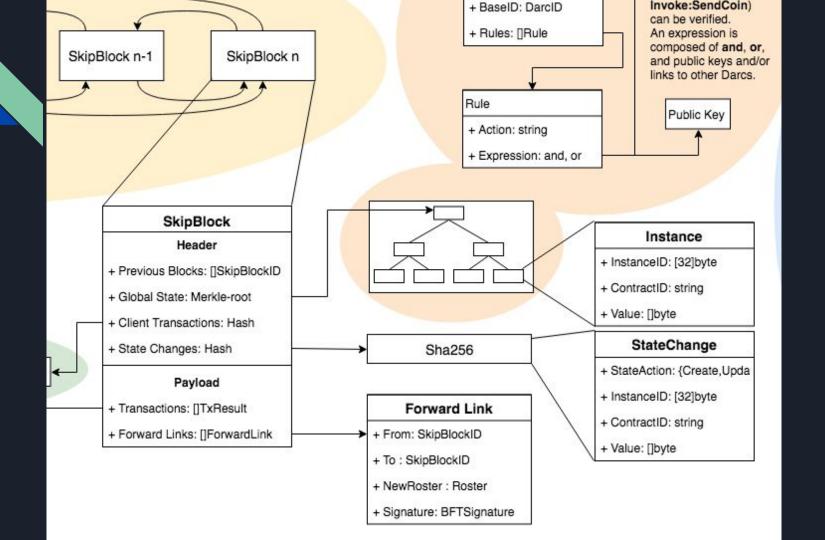
Latest idea:

- Use Proof-of-Personhood to allow nodes to enrol in the system
- Proof-of-Personhood allows us to restrict the number of nodes to one per person

5 - Open a Wallet and Transfer Coins

- First create your address:
 - ./wallet join bc-*.cfg
- Now enter your public key in the google doc to get some coins
- Verify the coins with
 - ./wallet show
- Transfer some coins to a public address -add-
 - ./wallet transfer 1000 -add-
- Bonus: transfer coins to your neighbour(s)





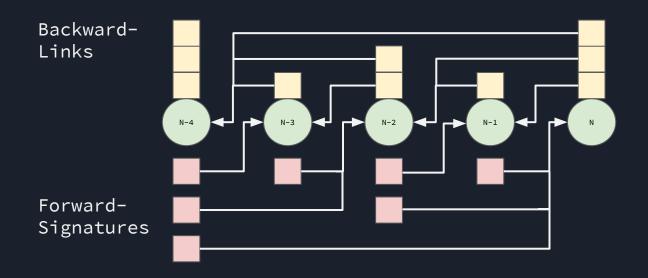
- On the status-website from http://localhost:8000/, open a javascript-console
 Mac OS X: <cmd><alt>j
- Type

```
proof
```

to see the proof of the configuration-contract. It contains:

links - skiplinks from the genesis block to the latest block

Base for ByzCoin - 1: skipchains

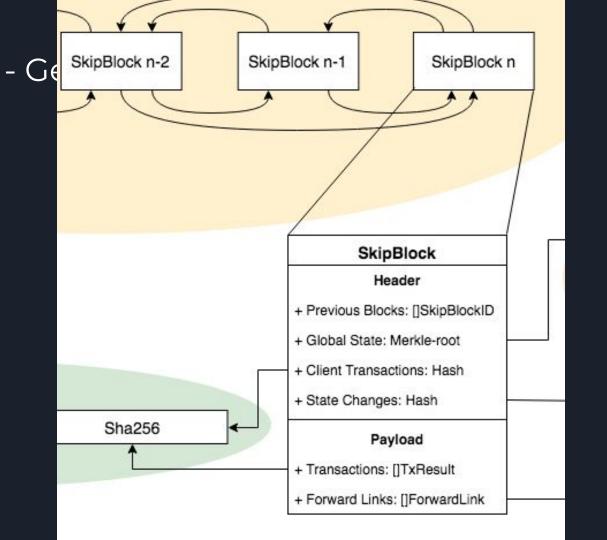


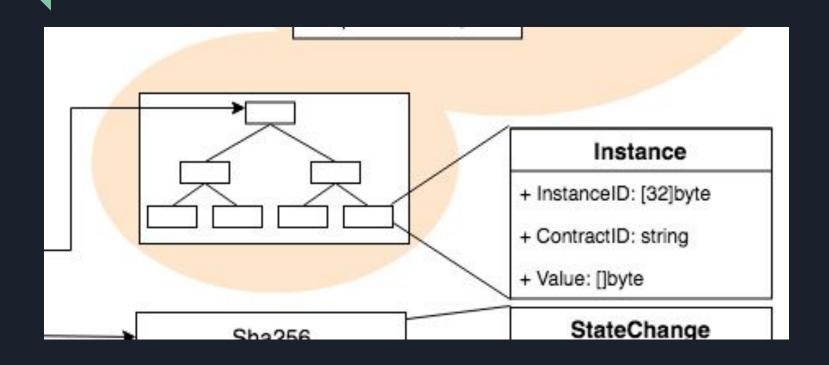
- On the status-website from http://localhost:8000/, open a javascript-console
 Mac OS X: <cmd><alt>j
- Type

proof

to see the proof of the configuration-contract. It contains:

- o links skiplinks from the genesis block to the latest block
- Latest block for the merkle-tree root hash of the state
- Inclusion-proof of the instance in the merkle-tree root





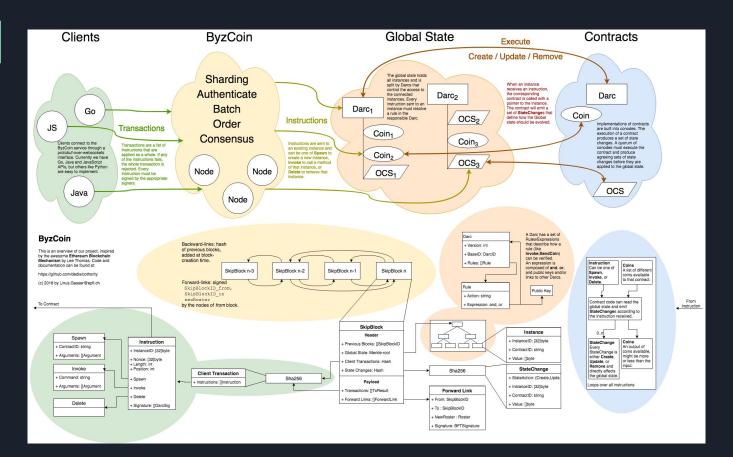
• First get the InstanceID of your coin:

```
./wallet show --address
```

• Get the proof of your coin-instance (for Firefox use *then*):

```
mycoin = await
bc.getProof(hex2iid("coinaddresshex"))
```

- Convert the value of the proof to the coin-data:
 coindata = value2coin (mycoin.instance.data)
- And check if the data is correct



7 - The End

- Copy of this demo at <u>https://github.com/dedis/demo 18 master</u>
- Write issues / bug reports to https://github.com/dedis/cothority
- Contact us at <u>dedis@epfl.ch</u>
- Keep an eye on
 https://personhood.online
 https://pop.dedis.ch