fuseelab.github.io

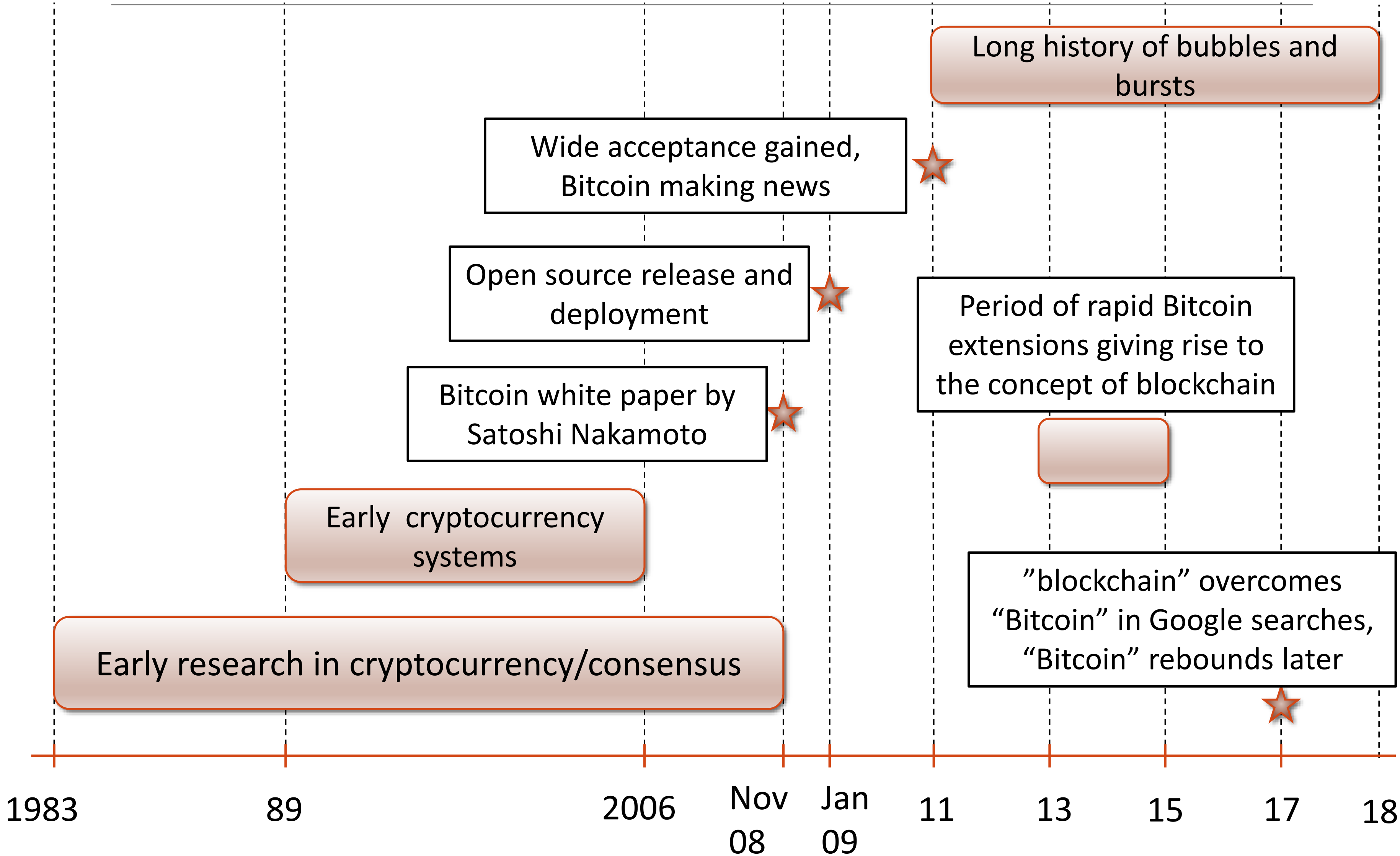
Blockchain Landscape and AI Renaissance
The Bright Path Forward

Link to our companion papers:
http://msrg.org/papers/bcbi-tr
http://heim.ifi.uio.no/~romanvi/debunking-bc-myths.html

BY KAIWEN ZHANG,
ROMAN VITENBERG,
HANS-ARNO JACOBSEN
MOHAMAD SADOGHI
MOHAMMAD TABATABAEI

Understanding Blockchains

Historical perspective



Status today: the Blockchain hype

Bitcoin gold rush

15 percent of top global banks rolled out full-scale commercial blockchain products in 2017

- Goldman Sachs alone investing half a billion USD

Blockchain became national storage technology in Estonia

Blockchain storage strategy and regulations in Netherlands

Microsoft declares “blockchain” as a “must win” technology for the Azure platform and business

IBM unveils new blockchain-oriented strategy; opens a new department

Dedicated labs and education programs in blockchain engineering around the globe

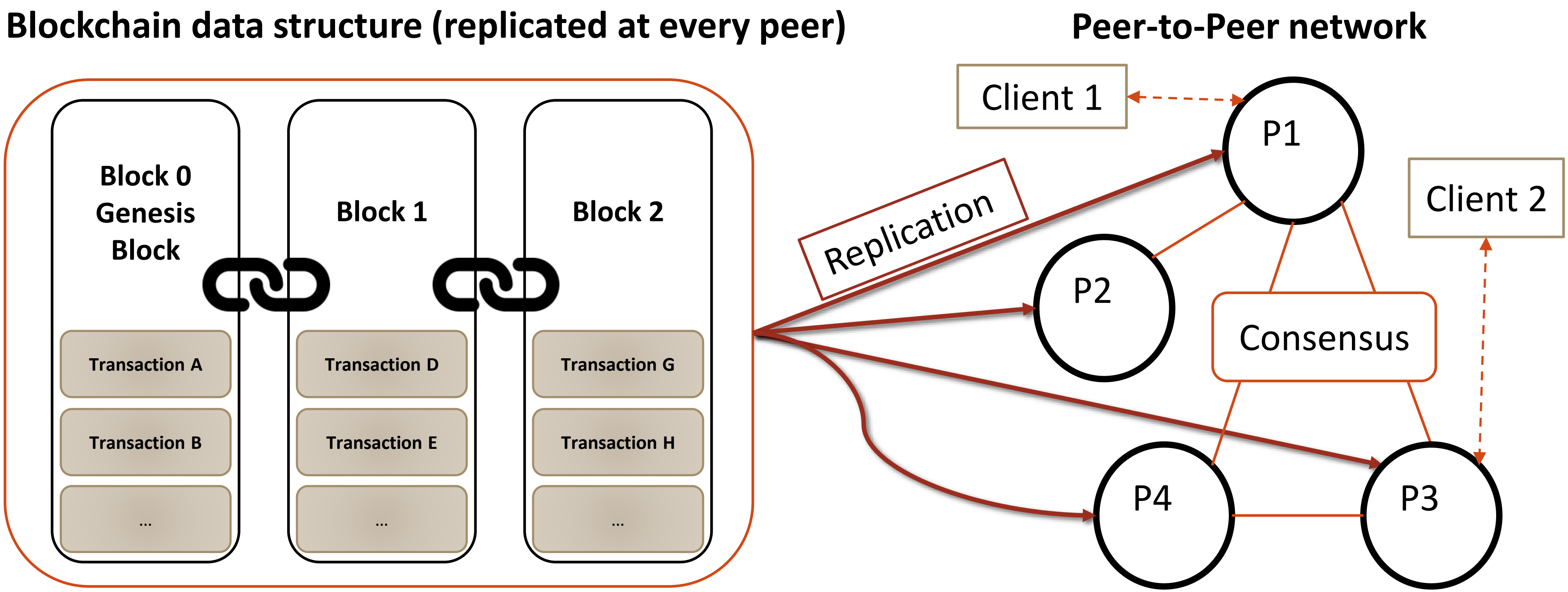
- A master program in blockchain engineering at the University of Delft
- A new course at the University of Oslo, TUM, Cornell, and many others

Hottest topic at many societal, industrial, and academic conferences

Blockchain 101

Distributed Ledger Technology (DLT)

BLOCKCHAIN



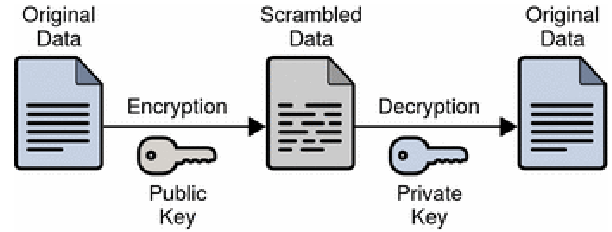
Cryptography is used to…

…encrypt data, prevent modification, insert new blocks, execute transactions, and query…

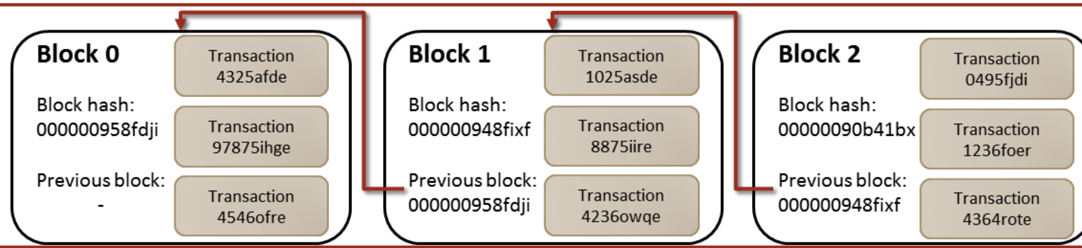
the distributed ledger

Cryptography and security in blockchains

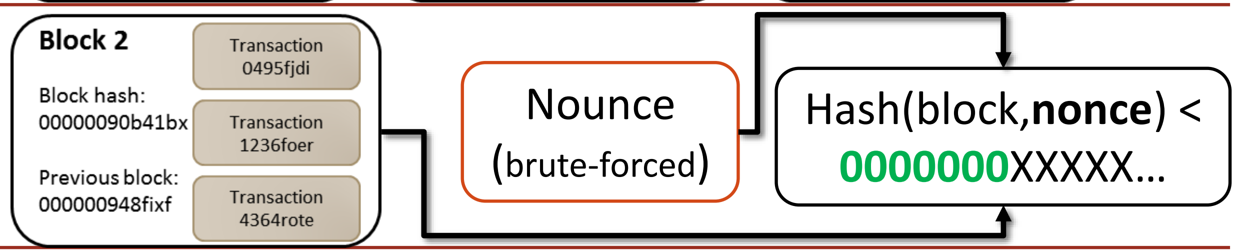
Encrypt data:
Public Key Encryption



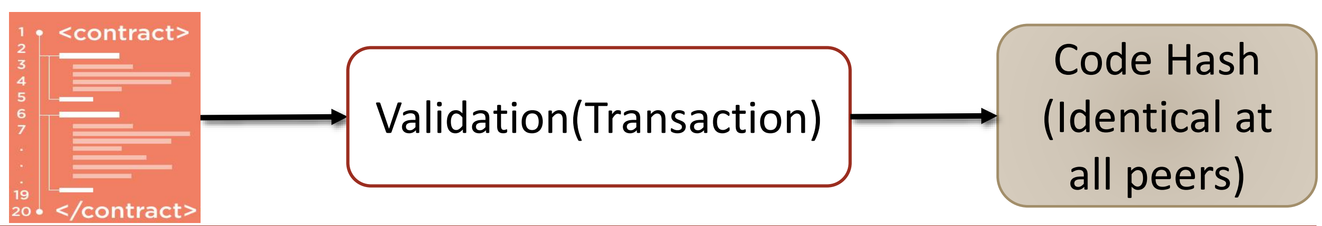
Prevent modification:
Hashed Linked List



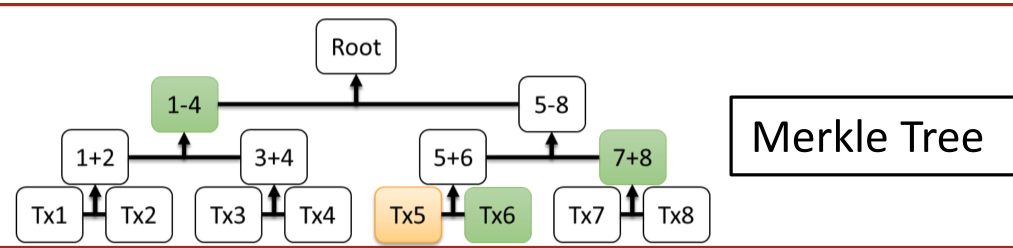
Insert new blocks:
Proof-of-Work



Execute transactions:
Smart Contracts



Query the blockchain:
Simple Payment Verification

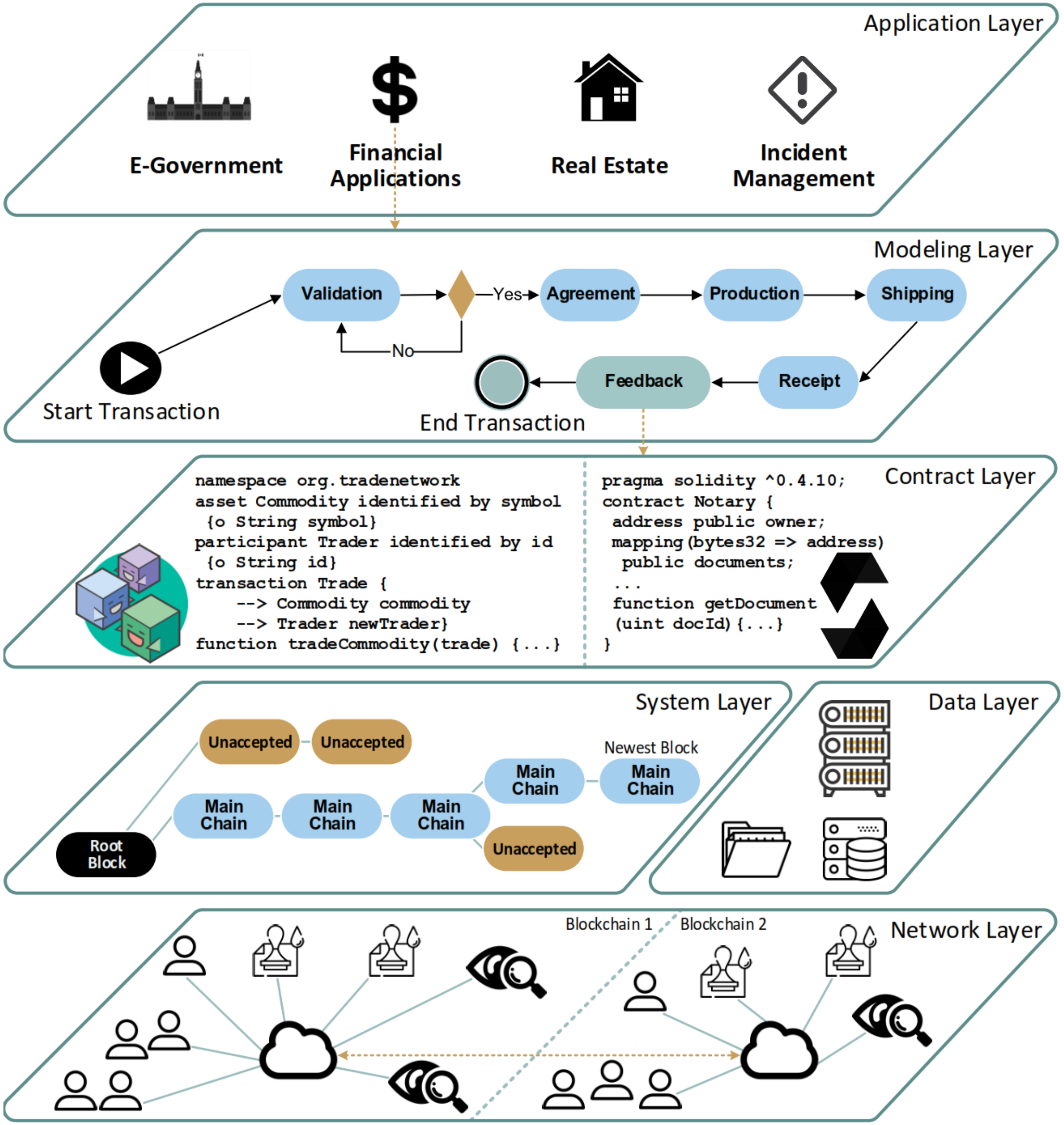


Blockchain Reference Architecture

This vision diagram encompasses all aspects related to blockchain technologies.

Upper layers capture application semantics and their implementation.

Lower layers are concerned with technical system details.



ZHANG, VITENBERG, JACOBSEN, SADOGHI, TABATABAEI © 2018

Blockchain vs. Distributed DB

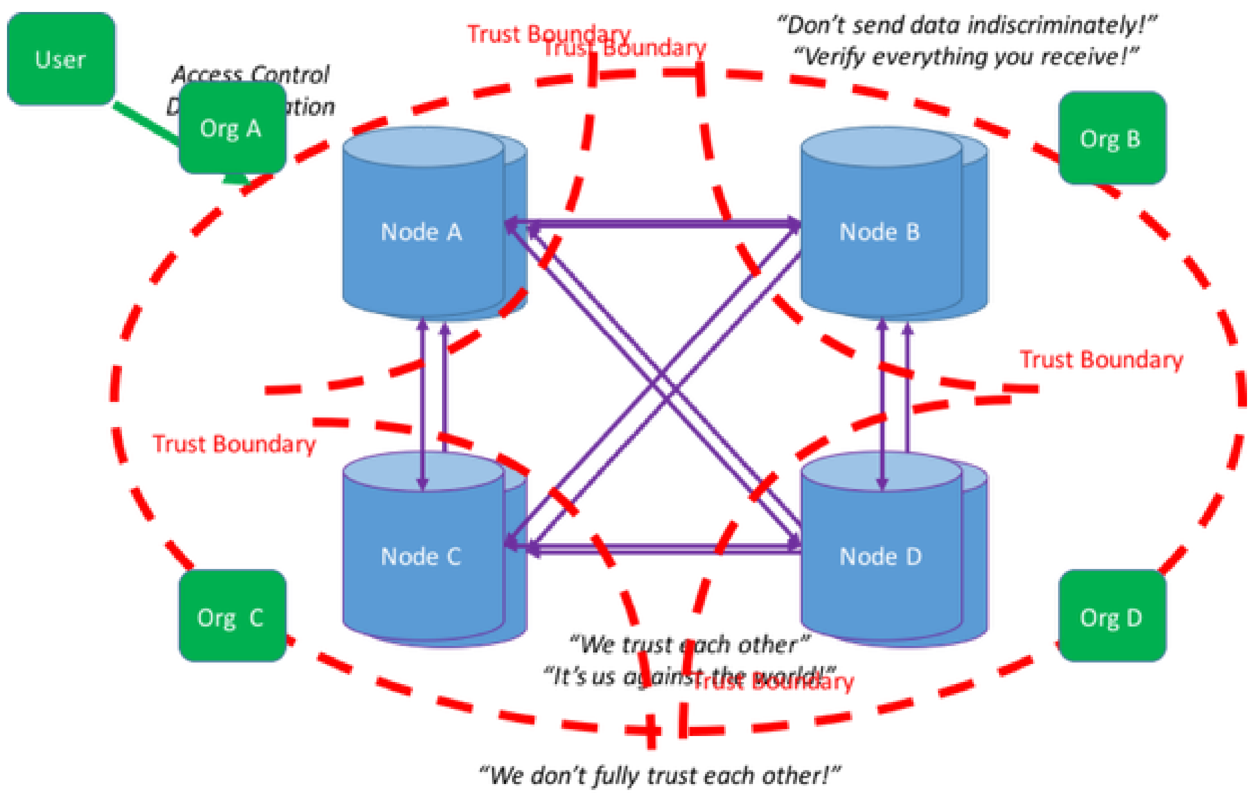
Blockchains maintain a log (aka a ledger) of all transactions since the start of deployment

- e.g. in Bitcoin, there is no direct record of the current state

The trust model is fundamentally different

Blockchain / distributed ledger

Distributed database



Outline

Session 1: Foundations

- Concepts: Byzantine Consensus, Mining, Proof-of-Work, Smart Contracts
- Original system: Bitcoin

Session 2: Beyond Bitcoin

- Smart contracts
- Platforms: Ethereum, Hyperledger

Session 3: Research

- System insights
- Research directions, integration with AI

Session 4: Hands-on tutorial on Ethereum

- Smart contract development and deployment
- Tools for deploying and managing Ethereum

Blockchain Concepts

DEFINITIONS

BITCOIN OVERVIEW

Bitcoin vs. Blockchain

Bitcoin is a specific system

- Design
- Open-source implementation
- Deployment
- There are alternative cryptocurrency systems (some of which are spawn-offs) but they are not Bitcoin

Blockchain is ambiguous: can be the data structure used in Bitcoin or a separate concept

A guiding design principle/paradigm

- Not even a standard
- Generalization of Bitcoin (In what direction?)
- Hundreds of implementations
- Ethereum alone has hundreds of proprietary deployments in addition to the main public deployment

What is a blockchain-based distributed ledger?

- ✓ *An append-only log* storing transactions
- ✓ Comprised of *immutable* blocks of data
- ✓ *Deterministically* verifiable (using the *blockchain* data structure)
- ✓ Able to execute transactions *programmatically* (e.g., Bitcoin transactions and smart contracts)
- ✓ *Fully replicated* across a large number of peers (called miners in Bitcoin)
- ✓ *A priori decentralized*, does not rely on a third party for trust

Blockchain and the land of ambiguities

Definition 1: a system that uses the blockchain structure of Bitcoin but extends the functionality

- Extended business logic
- Different consensus protocol

Definition 2: a system that maintains a chain of blocks

- Could be a structure other than that of Bitcoin

Definition 3: a system that maintains a ledger with all transactions

- Not necessarily stored as a chain of blocks
- Aka distributed ledger systems

Definition 4: a system with distributed non-trusting parties collaborating without a trusted intermediary

Definition 5: a system that uses smart contracts

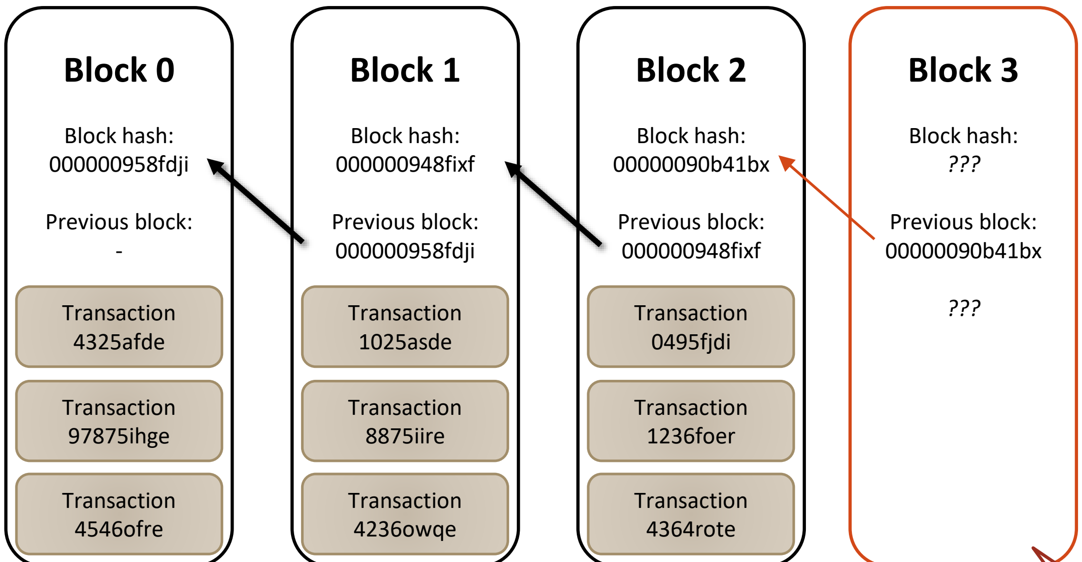
Main benefits of DLTs

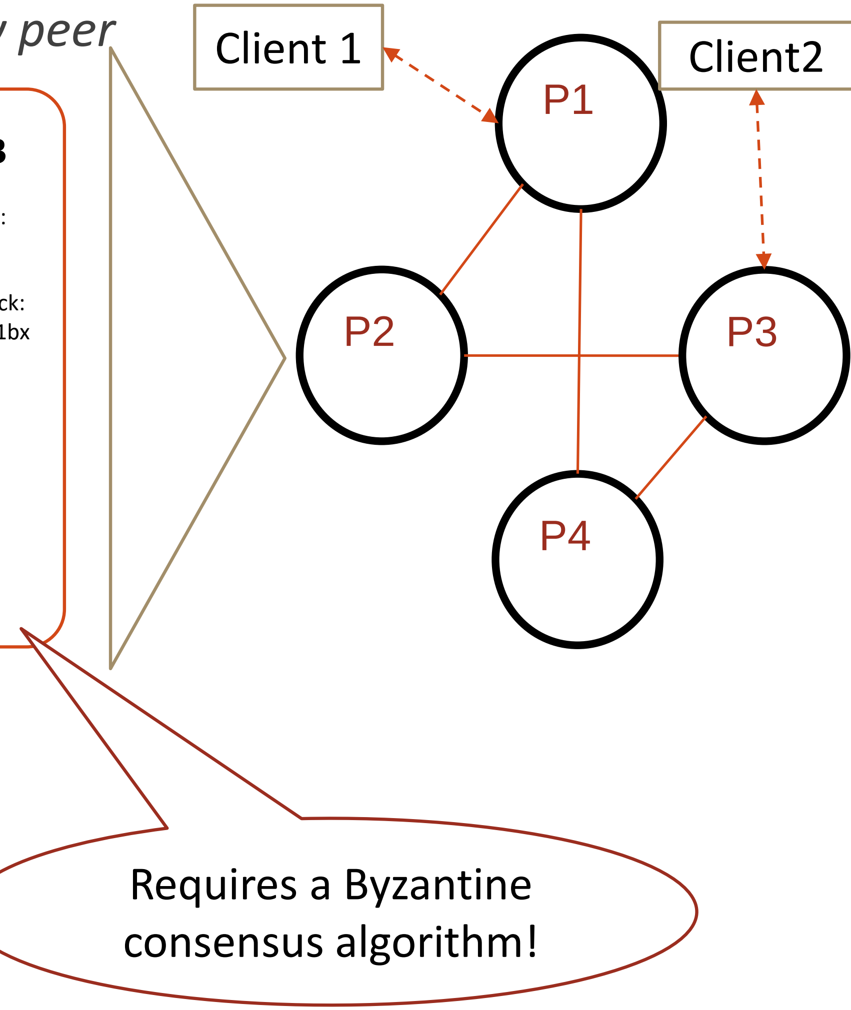
Enable parties who do not fully trust each other to form and maintain consensus about the existence, status and evolution of a set of shared facts

The ecosystem of smart contracts

Immutability using Hashing

Blockchain data structure maintained at every peer

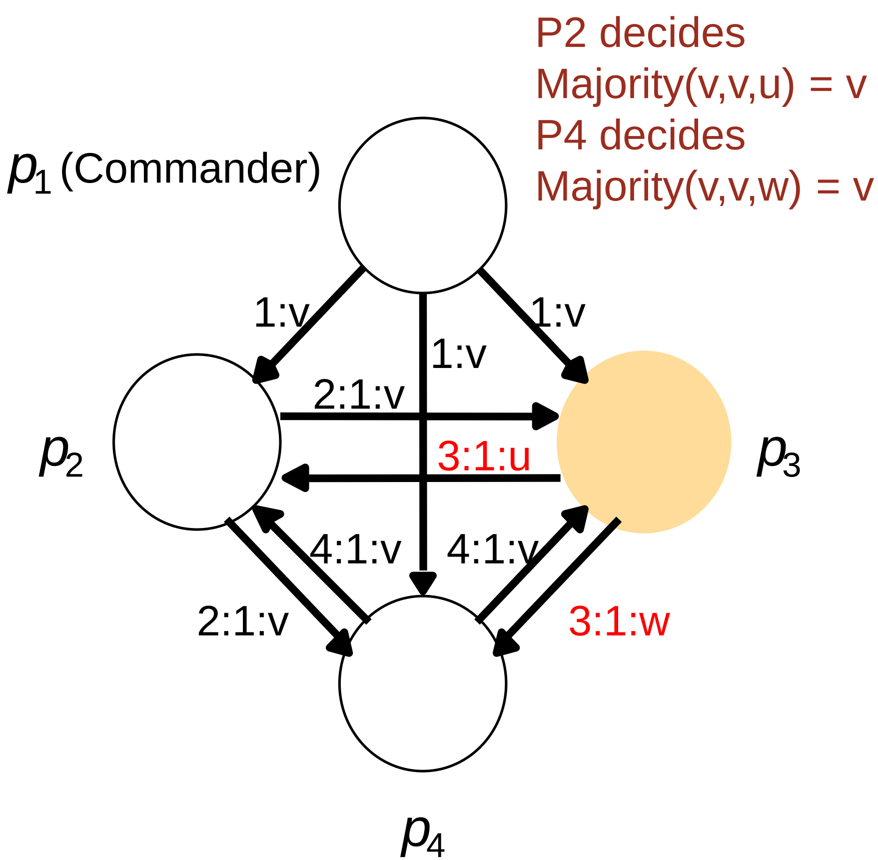


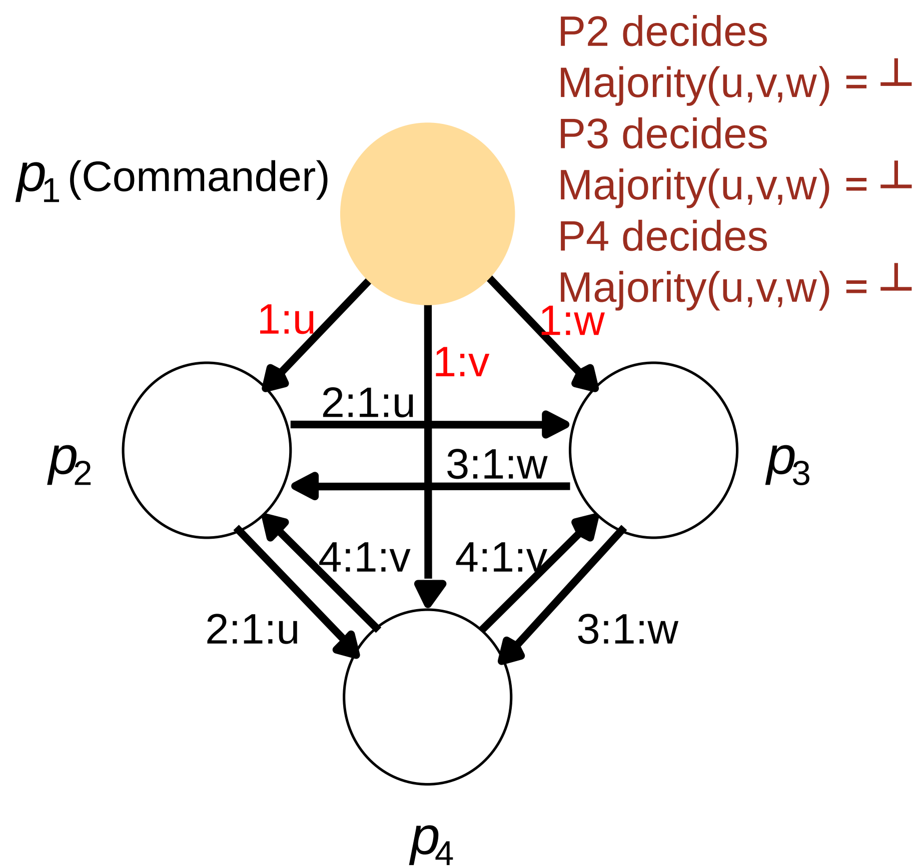


Origin: Byzantine Generals

- Devised by Lamport, 1982
- A *distinguished process* (the ***commander***) proposes initial value (e.g., “attack”, “retreat”)
- Other processes, the ***lieutenants***, *communicate the commander’s value*
- *Malicious processes* can lie about the value (i.e., *are faulty*)
- *Correct processes* report the truth (i.e., are *correct*)
- Commander or lieutenants may be faulty
- **Consensus** means
  - If the commander is correct, then correct processes should agree on commander’s proposed value
  - If the commander is faulty, then all correct processes agree on a value *(any value, could be the faulty commander’s value!)*

3f+1 Condition (1 failure, 4 nodes)

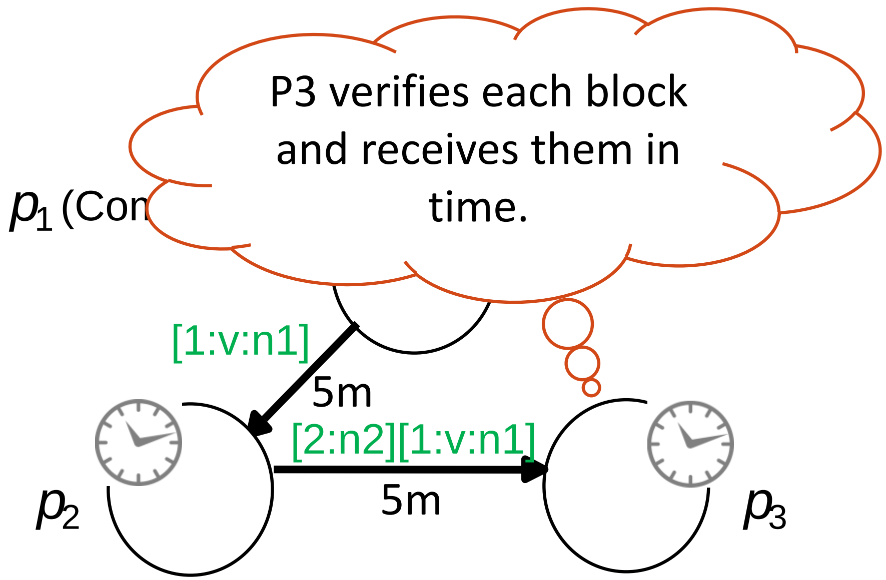


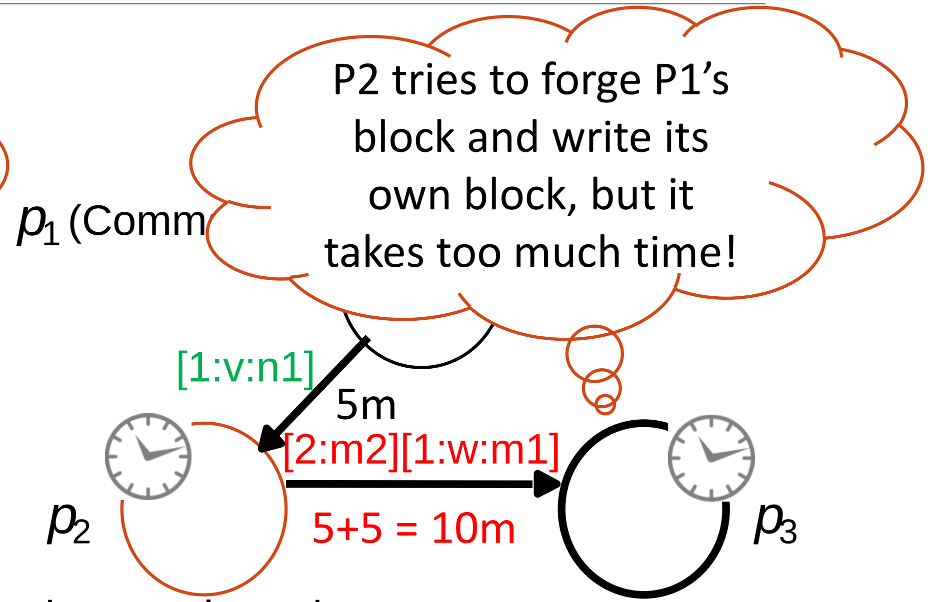


Source: Tanenbaum, Steen.

Faulty processes are shown coloured

With Blockchains (Proof-of-Work)





Faulty processes are shown coloured

<u>Idea #1</u>:
Each message takes exactly 5 minutes to create by any process. ("*Magic Block*")

<u>Idea #2</u>:
Each process can accurately measure the amount of time taken by a process to create a message. ("*Magic Watch*")

Blockchain “Cryptopuzzles”

verify(nonce, data) meets some “requirements”

Use of “trapdoor functions” (hash functions)

- Cannot reverse the function to find the input
- Therefore, keep trying random values (called nonce) until you find a solution
- Like trying random combinations to a lock…
- *The more computing power* you have, **the** *faster* you can solve the cryptopuzzle.
- “*Magic blocks*” are blocks with cryptopuzzles, where everyone has the same power. 

Proof-of-Work Example

E.g., the challenge is:

- sha256sum("data:nonce") starts with an "0"
- Normally more complicated than that! (e.g., 18 zeroes)

➢ P1 wants to send "1:v" to P2

```
kzhang@grey:~$ echo "1:v:118" | sha256sum
9479038ca7543ece09f48e8c77fcea147d7561cac14058199afea18c2f323b8b
kzhang@grey:~$ echo "1:v:119" | sha256sum
79ae2bbac929112a349c2fe7f50210355f4a24683b2dd1ea8f059c9beeed7fd6
kzhang@grey:~$ echo "1:v:120" | sha256sum
002ce3a3b7092d960abf1795a89f70eb0f9ef960036e7d4620cbd3d26d34ffc8
```

➢ Send "1:v:120" to p2

Proof-of-Work Example

- P2 verifies "1:v:120" is correct (very quick!)
- P2 wants to send "2:1:v:120" to P3

```
kzhang@grey:~$  echo "2:1:v:120:119" | sha256sum
911ab1edf1f331ff423a45fe4c382db30a3f1cf802bb2211df53c80d5798c7ba
kzhang@grey:~$  echo "2:1:v:120:120" | sha256sum
5344a3561673b1481b9cf69493368ca408b1edef67e3f96819c5d1b36cea53ce
kzhang@grey:~$  echo "2:1:v:120:121" | sha256sum
0a908c651e9ec5374976dc8f49a3342a4a78966011551da8871a6cc123c5b57
```

- P2 sends "2:1:v:120:121"
- P3 verifies "1:v:120" *AND* "2:1:v:120:121" are correct
- If P2 wants to send "2:1:w" and fool P3, it needs to find n1 for "1:w:n1" & n2 for "2:1:w:n1:n2"
- If P3 has a way to *detect* that P2 is *doing too much work*, it can detect fraud.

Bitcoin

LAYER BY LAYER

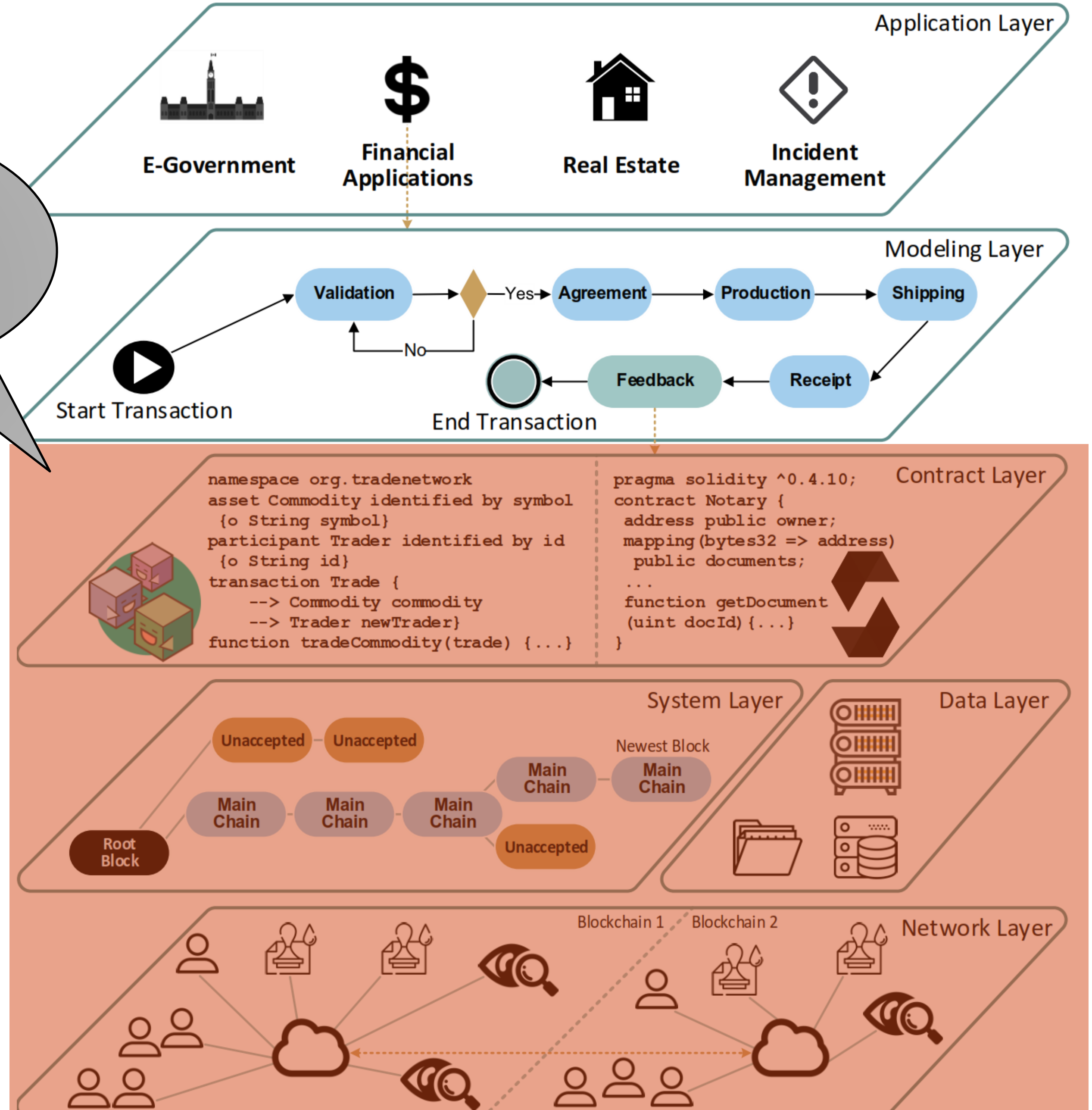
Bitcoin layers

Block Reference Architecture

This vision diagram encompasses all aspects related to blockchain technologies.

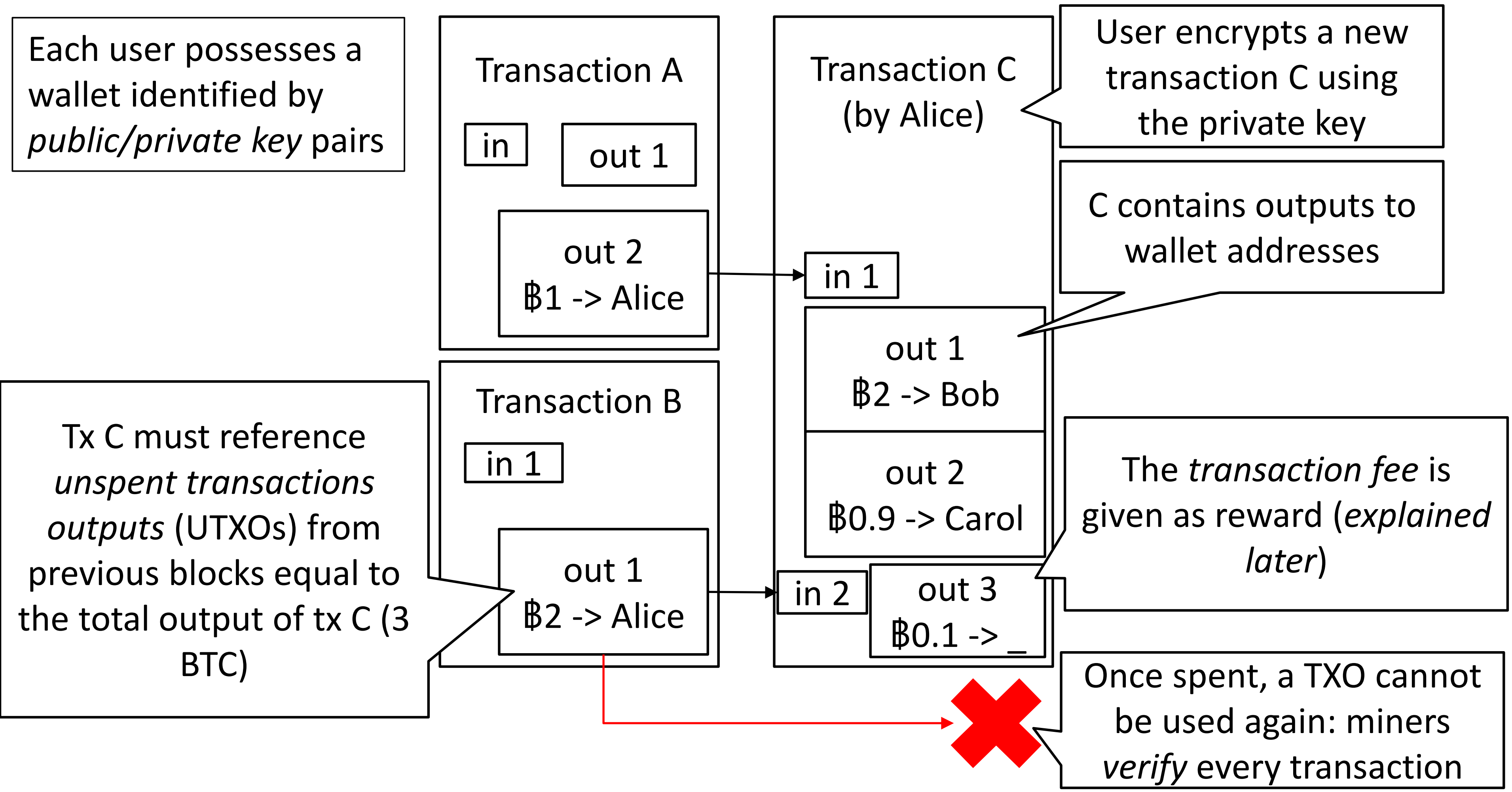
Upper layers capture application semantics and their implementation.

Lower layers are concerned with technical system details.



ZHANG, VITENBERG, JACOBSEN, SADOGHI, TABATABAEI © 2018

Bitcoin Transactions



Wallets and addresses

Users require a wallet to store money

- This includes any user, **including but not limited to** miners

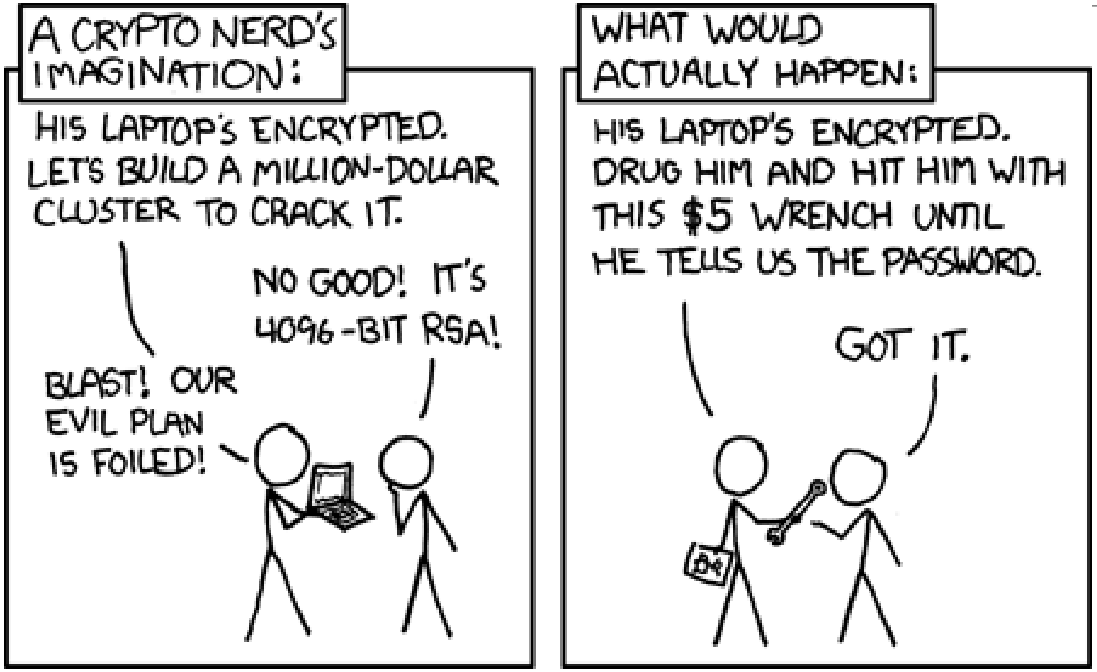
Wallet is authenticated and identified by public/private key pairs

- Generated using ECDSA (Elliptic curve cryptography)
- HD wallets contains a master seed to generate more private keys

Redeeming transactions:

- Each TXO address is a hash of the public key of the receiver, who signs proof with the private key
- Transactions do not have a “from” address, so it is impossible to prove you are the sender
- Each address is designed to be single use: wallet programs will automatically generate new addresses

Wallet security



Losing your private key:

- Loss of private key means the wallet and its funds are permanently locked, as it is no longer possible to sign proofs redeeming existing TXOs.
- This money is essentially lost, thereby reducing the total amount of currency in Bitcoin
- Trusting an online service to store your key is also risky, since there is no way to prove that you are the rightful owner if the key is stolen or misused
- The most reliable solution is to store your private keys on tamper-proof hardware wallets

Communication in Bitcoin

Broadcast to all the network

Two primary uses
- Users broadcast their transactions
- Miners broadcasts updates to the blockchain (new blocks)

Implemented via gossiping protocol in a P2P network
- Not terribly efficient but has not been a bottleneck so far

Works because financial transactions are very short and their rate in Bitcoin is far below that of credit cards

Needs to be fairly reliable for the system to work but 100 percent reliability in message delivery is not required
- Users and miners need to detect message loss and retransmit messages if needed

Message propagation should be reasonably fast
- Slower network quantifiably increases the risk of attacks

Transaction Flow

Alice
(Sender)

Bob
(Receiver)

1. Bob generates and send a public key address.
2. Alice creates a transaction using this address.
3. Alice sends the new transaction to the network.
4. The transaction is broadcast using gossiping.
5. The transaction is included in a block.
6. Bob can verify the transaction is in the blockchain.
7. Bob can now sign new transactions which redeem this address.

Transaction A

in 1

out 1
฿1 ->
Bob.Address1

Transaction B

in 1
Bob.Address1

out 1

Consensus in Bitcoin

The network needs to agree on

- Which recently broadcast transactions go into the blockchain
- In what order

The general anatomy of consensus:

Make a proposal

↓

Reach a consensus

↓

Announce the decision

Tough problem

- Especially in P2P
- Dozens of impossibility results since 1983
- Does not scale beyond ~30 participants
- Takes long time to converge

Challenge 1: who proposes and when?

The network cannot sustain each and every user or peer making a proposal whenever she wishes

Made worse by the proliferation of identities (Sybil attack)

Need to moderate the number of proposers and rate of concurrent proposals

- While keeping them sufficiently high

Several principal solutions

- Proof-of-work: need to do heavy computation and show the proof of it
- Proof-of-stake: need to possess a sufficient amount of coins

Important optimization: propose new transactions in batches

- A block in Bitcoin is structured as a tree of proposed transactions
- With nice cryptographic properties; called a Merkle tree

Cryptopuzzles in Bitcoin

The proposer has to find ***nonce***, such that *hash(**nonce** | H | Tr_1 | … | Tr_n)* < ***target***

Effectively has to scan the entire ***nonce*** space

target is a fraction of the hash space

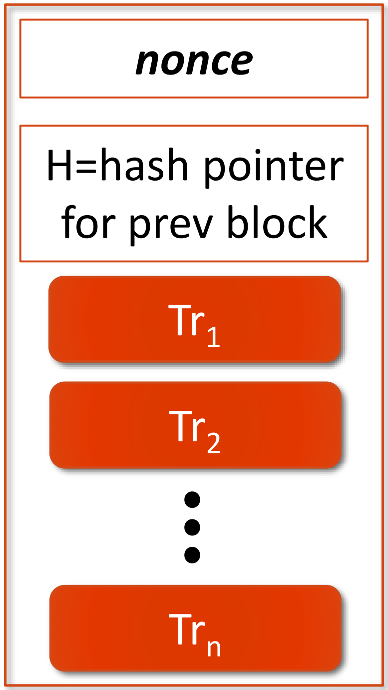
- Every node recomputes ***target*** every 2016 blocks
- Such that the average time for the whole network to solve a cryptopuzzle is 10 min

For proposer *p*,

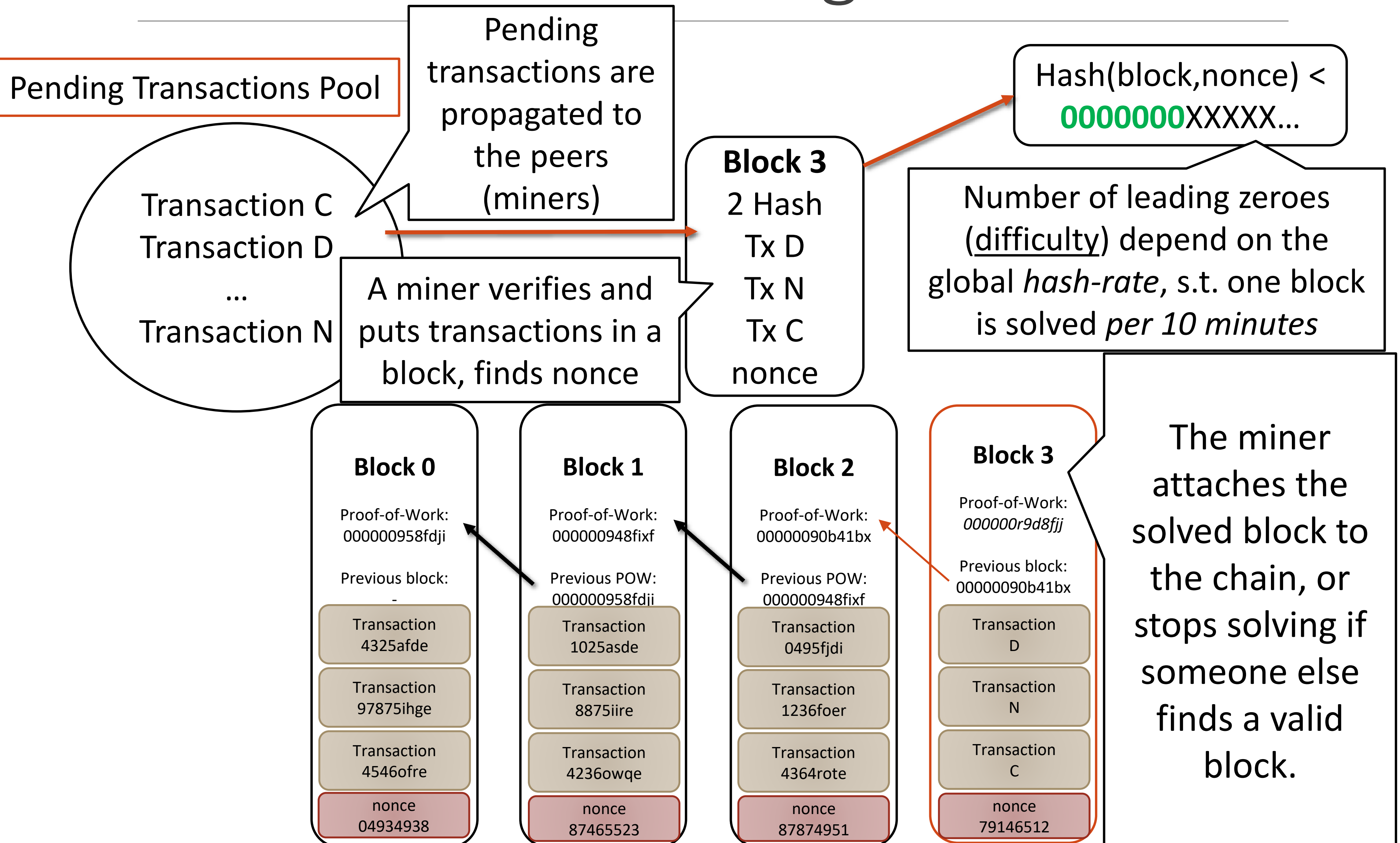
$$mean\ time\ to\ next\ block = \frac{10\ minutes}{fraction\ of\ p's\ computing\ power}$$

The solution is fast to verify

A block in Bitcoin



Proof-of-Work Mining in Bitcoin



Challenge 2: Why propose non-empty blocks?

Two incentive mechanisms in Bitcoin

- Block creation reward: a block proposal creates a number of new bitcoins and transfers them to the proposer
  - Included as a separate transaction in the block
  - Ensures that each proposer solves a different cryptopuzzle
  - The only way to create new bitcoins
  - The amount is predefined and gets halved every 210,000 blocks
  - Predicted to go down to zero before year 2140
  - The geometric progression totals to 21 million bitcoins
  - The rules may change in the future
- Transaction inclusion fee: Alice can decide to pay a small fee to the block creator as part of her transaction
  - Voluntarily, there is no predefined amount

Cryptoeconomy of Mining

Incentives give rise to the mining industry in Bitcoin

- Miners: cracking cryptopuzzles and listening to transaction broadcasts

Expenses: *mining rig + operating costs (electricity, cooling, repairs)*

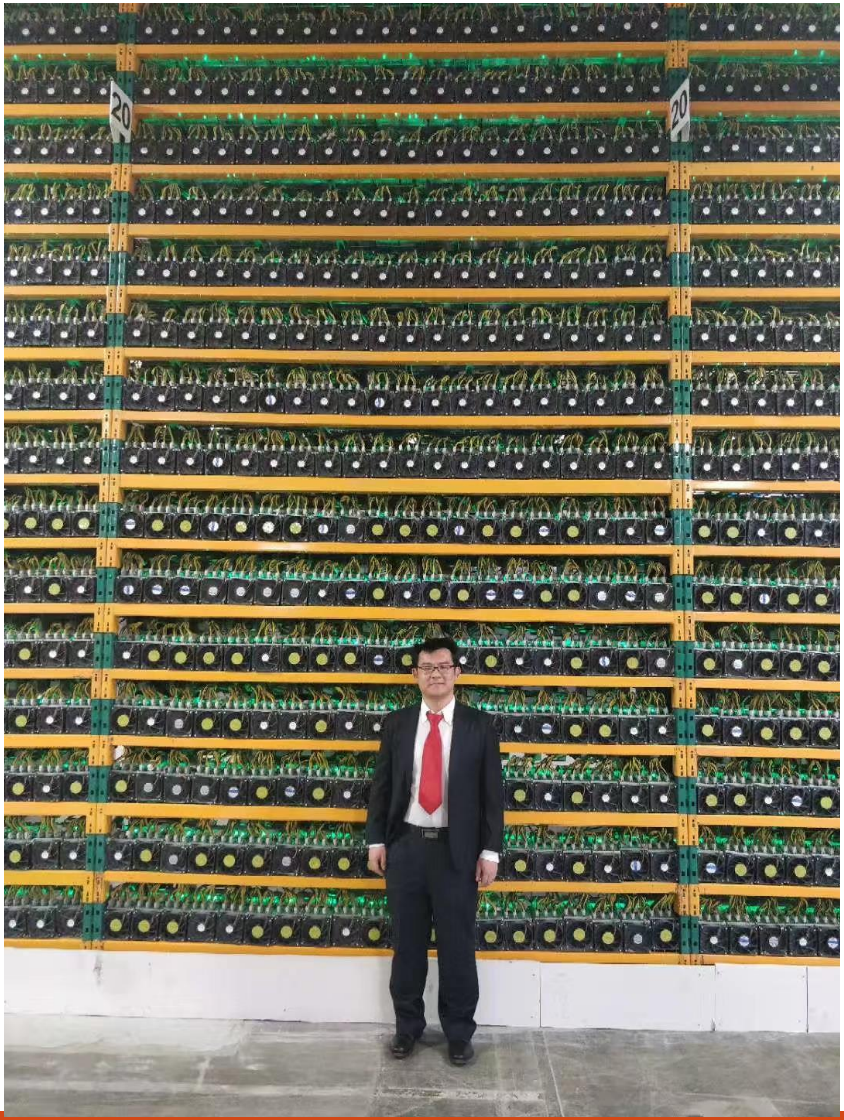
- Paid in real currency
- Operating costs are variable

Profits: *block reward + transaction fee * # of transactions in a block*

- Paid in Bitcoins
- The fee and rate of transactions are unpredictable
- The mean time to next block is easy to compute
  - However, the per-miner sample is small while variations are huge

Mining pools: groups of cooperating miners

Large-scale mining farms



Bitfarms™

Largest Bitcoin mining farm in NA
27.5 MWs, 200 PH/s (0.5% total)
AntMiner S9 (Bitmain): 14 TH/s, $500
Cheap hydroelectricity: 4 cents/kWh but…*

Another picture (different site)

Front side

Reaching consensus in Bitcoin

A miner broadcasts the proposed block

- The block includes a hash to the latest block known to the miner

When a peer receives a proposed block

- Check that the proof of cryptopuzzle solution is valid
- Check that each transaction is valid (business logic)
- If the hash pointer is valid, append the new block to the local copy of the blockchain
- Conflict resolution: if the proposed chain is longer than the current local copy, replace the local copy

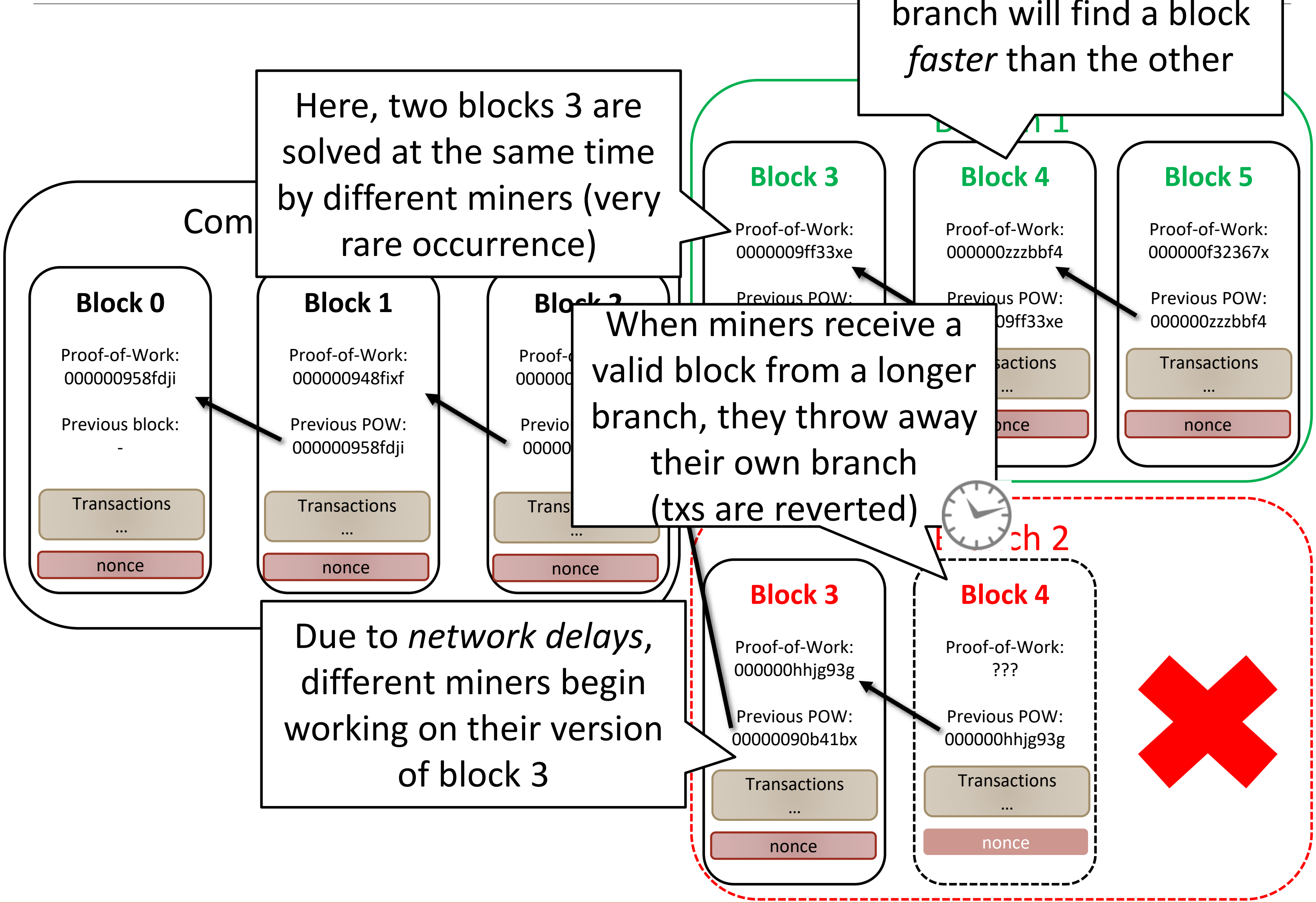
Local copies may diverge!

- Lost messages and concurrent blocks arriving in reverse order
- The probability depends on the network

Probabilistic convergence over time is proven when using the longest chain for conflict resolution

- The probability of a block being non-final decreases exponentially with the number of later blocks stored in the chain
- The standard client sends a confirmation after six later blocks stored in the chain
- Takes an order of one hour in practice

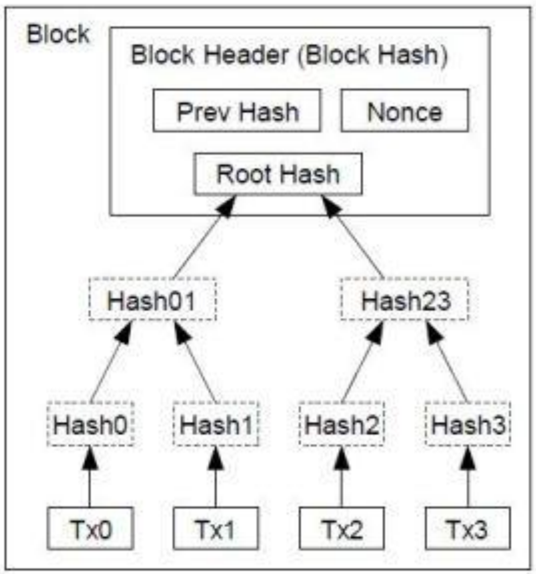
Branching



Due to variance, one branch will find a block *faster* than the other
Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Block 0
Proof-of-Work:
000000958fdji
Previous block:
-
Transactions
…
nonce
Block 1
Proof-of-Work:
000000948fixf
Previous POW:
000000958fdji
Transactions
…
nonce
nonce
Block 3
Proof-of-Work:
0000009ff33xe
Block 4
Proof-of-Work:
000000zzzbbf4
Block 5
Proof-of-Work:
000000f32367x
Previous POW:
000000zzzbbf4
Transactions
…
nonce
When miners receive a valid block from a longer branch, they throw away their own branch (txs are reverted)
Block 3
Proof-of-Work:
000000hhjg93g
Previous POW:
00000090b41bx
Transactions
…
nonce
Block 4
Proof-of-Work:
???
Previous POW:
000000hhjg93g
Transactions
…
nonce
Due to *network delays*, different miners begin working on their version of block 3

Data Structure within a Block

Merkle Tree



- To avoid hashing the entire block data when computing PoW, only the *root hash* of the Merkle tree is included.
- For users without a full copy of the blockchain, *simple payment verification (SPV)* is used to verify if a specific transaction exists.
  - A *Merkle proof* only requires the transaction itself, block root hash, and all of the hashes going up along the path from the transaction to the root, e.g., Hash01, Hash2 (for Tx3).
- Spent transactions can be *pruned* in the local copy, leaving only the necessary intermediate nodes to save space.
  - E.g., if both Tx0 and Tx1 are spent, we can prune everything under Hash01

Data manipulation and queries

Reading the ledger and verifying its correctness is straightforward but time-consuming

- Publicly available, no access control whatsoever
- A copy is held by many users (over 10,000 today)
- Users are encouraged to download and run a verification

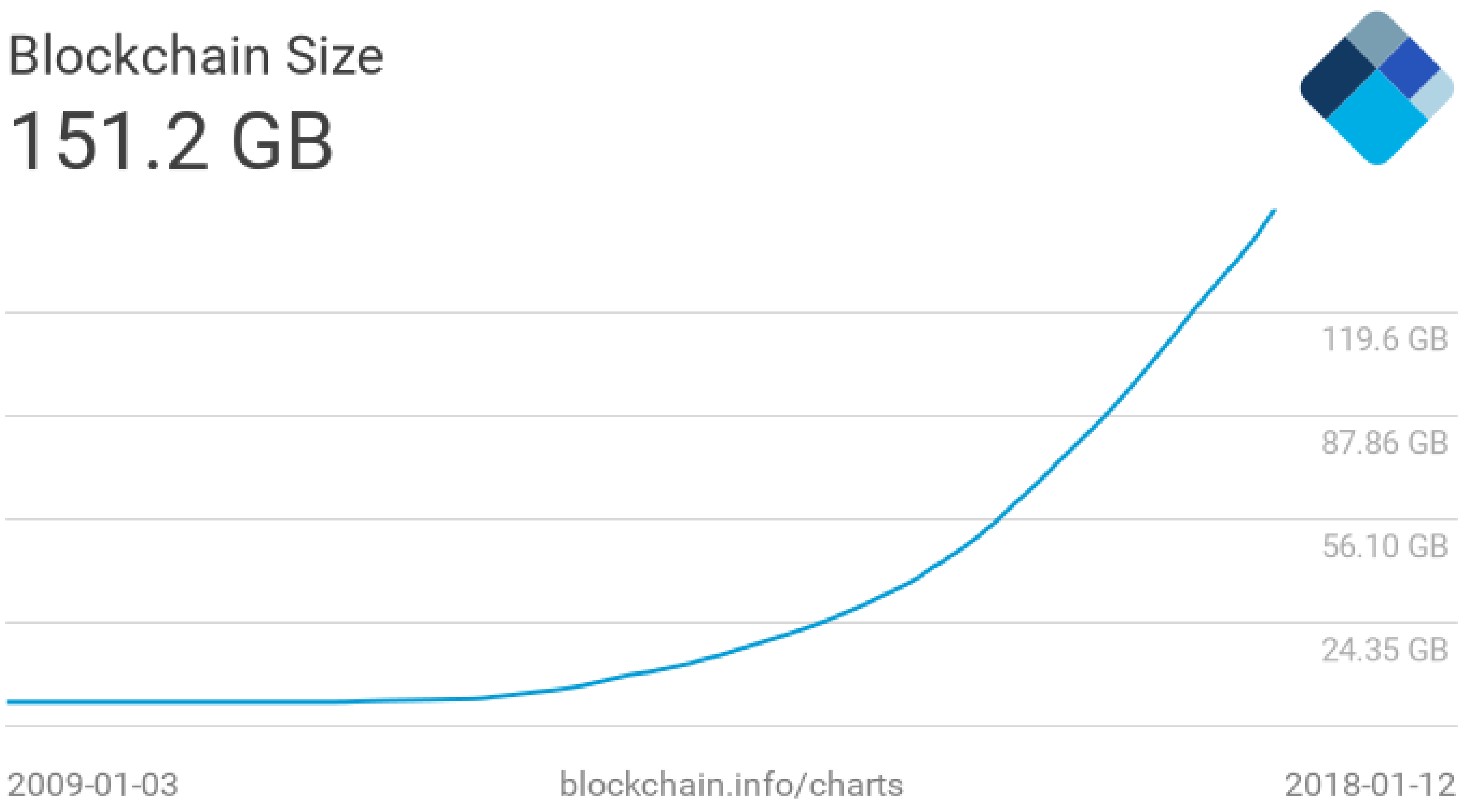
Transparency is a boon for data integrity but a bane for privacy

- Public keys are used as user identities
- A key can serve as a pseudonym, difficult to link to real identity
- A user can use a different pseudonym for each transaction
- The main threat comes from analyzing the history of transactions and linking them together

Temper-resistance is mostly a blessing

- But also a curse: difficult to compact or prune the history

Size of ledger



Business logic in Bitcoin

The output additionally includes a verification script

- representing the conditions under which the output can be redeemed, i.e., included as an input in a later transaction
- A typical script: “can be redeemed by a public key that hashes to X, along with a signature from the key owner”

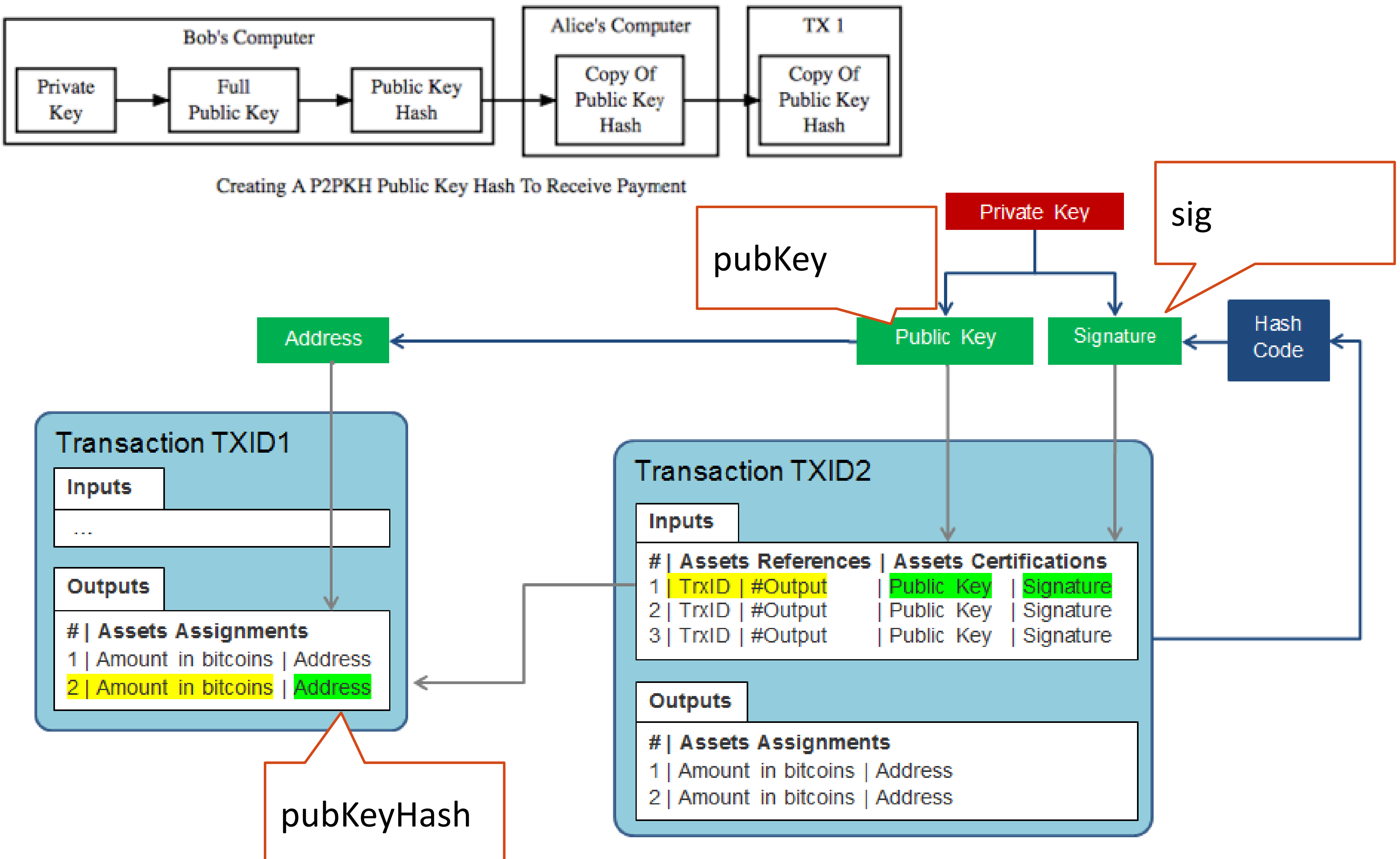
There is also a redeeming script attached to the input

Both scripts are executed by whoever verifies the redeeming transaction, such as a proposer

A script language with an order of 200 commands

- Support for cryptographic primitives
- Rather ad-hoc

Redeem a UTXO (P2PKH)

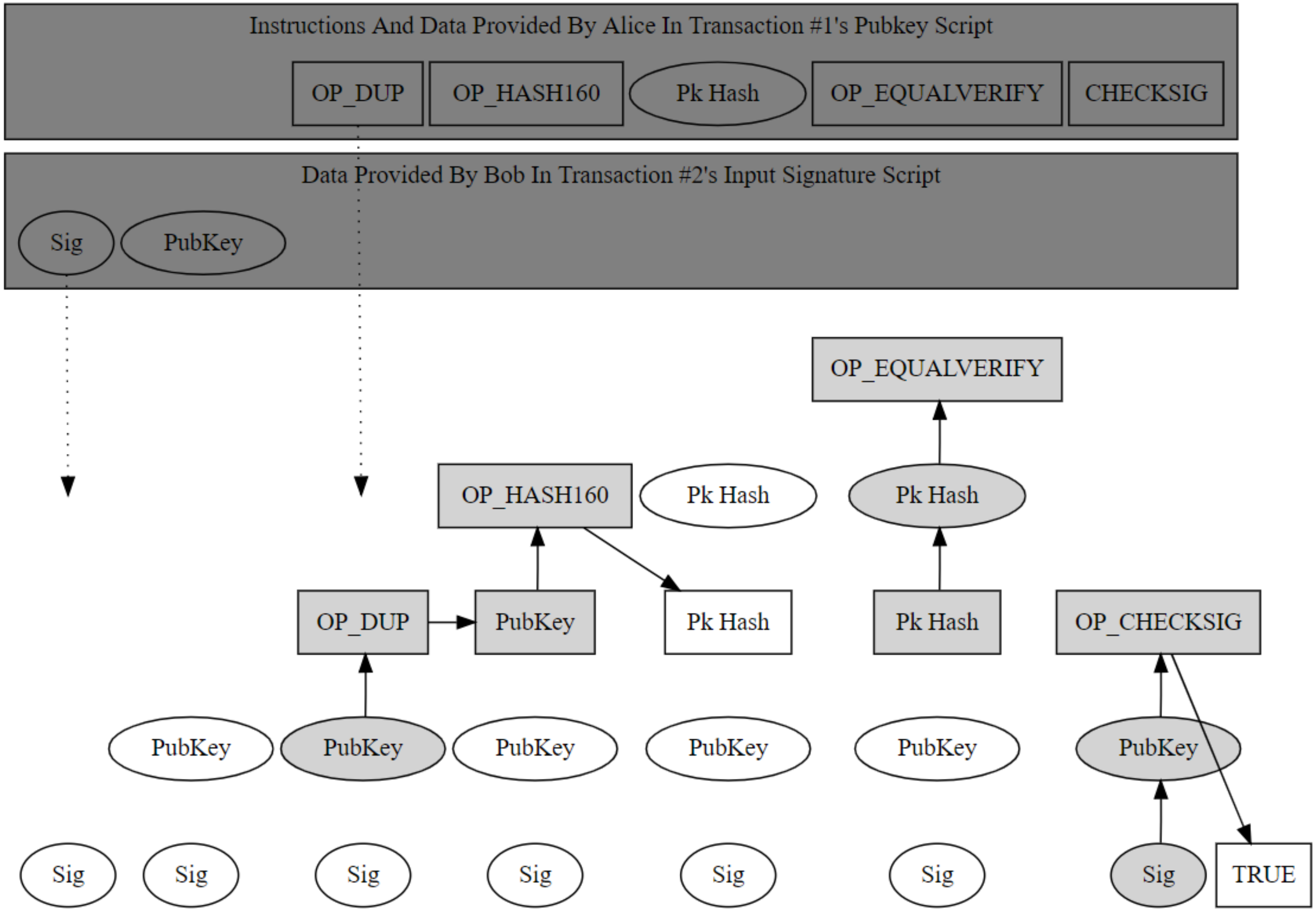


Bitcoin Script and P2PKH example

scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG (Script of output)
scriptSig: <sig> <pubKey> (Script of input)

Stack (top: to the right)	Script	Description
Empty.	<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Script = scriptSig.append(scriptPubKey)
<sig> <pubKey>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Add sig and pubKey to the stack
<sig> <pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Copy top element of the stack
<sig> <pubKey> <pubHashA>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Hash the top element
<sig> <pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG	Add pubKeyHash to the stack
<sig> <pubKey>	OP_CHECKSIG	Verify both elements are equal (using ECDSA)
true	Empty.	Verify first element is a signature of second element

Example illustration of P2PKH



Evaluation Stack Over Time During Succesful P2PKH Script Validation

Other examples

```
scriptPubKey: <expiry time> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP
              OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

Freezing funds for a period of time

```
scriptPubKey: (empty)
scriptSig: OP_TRUE
```

UTXO free to claim

```
scriptPubKey: OP_HASH256 6fe21b3….0000000000 OP_EQUAL
scriptSig : X, such that hash256(X) = 6fe…00000
```

Transaction Puzzle

```
scriptPubKey: OP_ADD OP_8 OP_EQUAL
scriptSig: OP_3 OP_5 (or… )
```

Challenge

```
scriptPubKey: OP_4 OP_5 OP_EQUAL
scriptSig: Impossible!
```

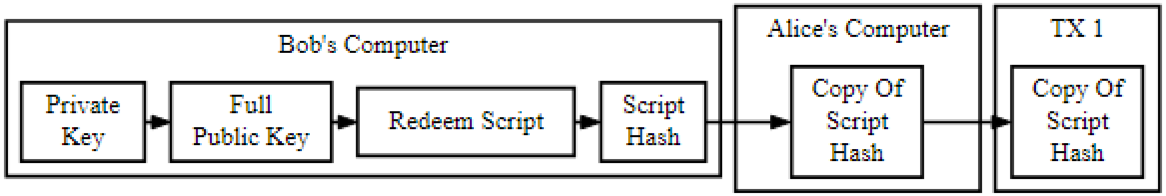
Proof-of-Burn

```
scriptPubKey: OP_RETURN PUSHDATA(N) <Data>
scriptSig: Impossible!
```

Data storage

http://learnmeabitcoin.com/glossary/script

P2SH Addresses



Address generation P2SH

```
redeemScript: <OP_2> <A pubkey> <B pubkey> <C pubkey> <OP_3> OP_CHECKMULTISIG

scriptPubKey: OP_HASH160 <Hash160(redeemScript)> OP_EQUAL
(scriptSig is added to scriptPubKey in serialized form)

scriptSig: OP_0 <A sig> <C sig> redeemScript (must evaluate to true on its own)
```

Multi-Signature 2-of-3 (P2SH)

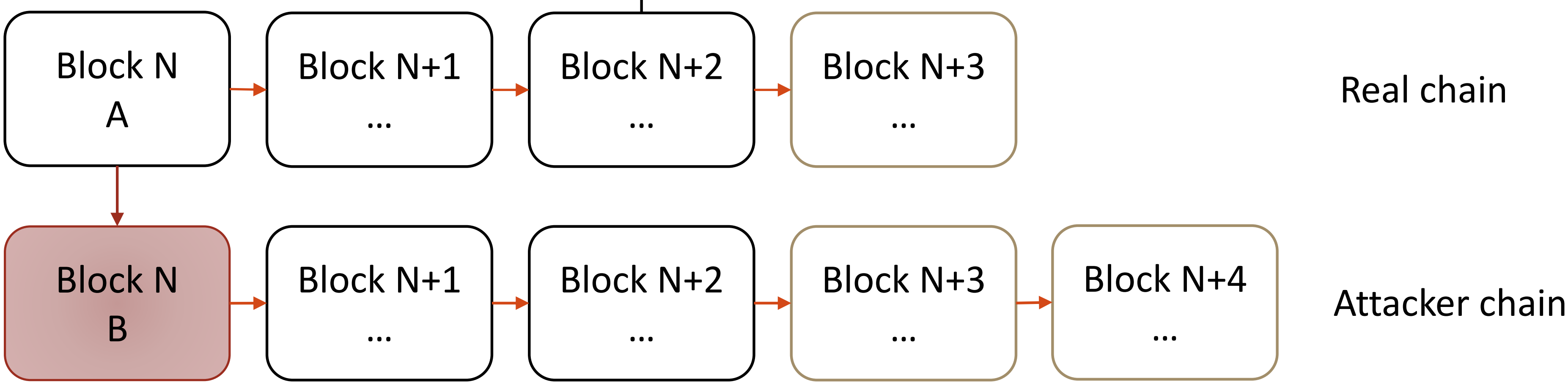
Preventing double spending

(51% Attack)

Transaction A
฿1 ->
Merchant 1

Transaction B
฿1 ->
Merchant 2

A malicious attacker creates two transactions using the same money (*double-spending*)



- The "Magic Watch" is the *continuous generation* of blocks in the main chain which *limits the amount of time* an attacker has to create its own chain.
- If the attacker owns *>51% of the power* in the network, the "Magic Watch" gives *enough time* to the attacker to *tamper the data*!

It must replace A with B in N, and solve the modified puzzles for the blocks faster than the real chain grows so that it can become longer

Other attacks (cursory)

Stealing bitcoins is hard because of digital signatures

- If, however, someone accumulates a lot of bitcoins, it becomes a prime target

Denial-of-service on the entire Bitcoin network is hard because of proof-of-work

- Still possible to bombard the network with invalid transactions

Starving a specific user: does not work if there is a sufficient number of honest miners

- Possibility to blackmail users with high tx fees if miners are “rational”
- cf. feather forking attacks

Economic attacks: selfish mining

- Attempts to maintain private branches longer than the public branch
- Releasing a longer private branch causes honest miners to lose revenue, “stolen” by the attacker
- 25% attack with “rational” miners

Limitations of Bitcoin

Limited expressiveness

- Cryptocurrency only
- Each app requires new platform (e.g. NameCoin, PrimeCoin, CureCoin)

Slow block time (10 mins)

- Also slow confirmation time (1+ hour for 6 confirmations)

Hard/Soft forks

- Updates to the code cause forks
- Hard forks are not compatible
- Duplicated money
- Bitcoin: Cash, Classic, Gold

Slow transaction rate

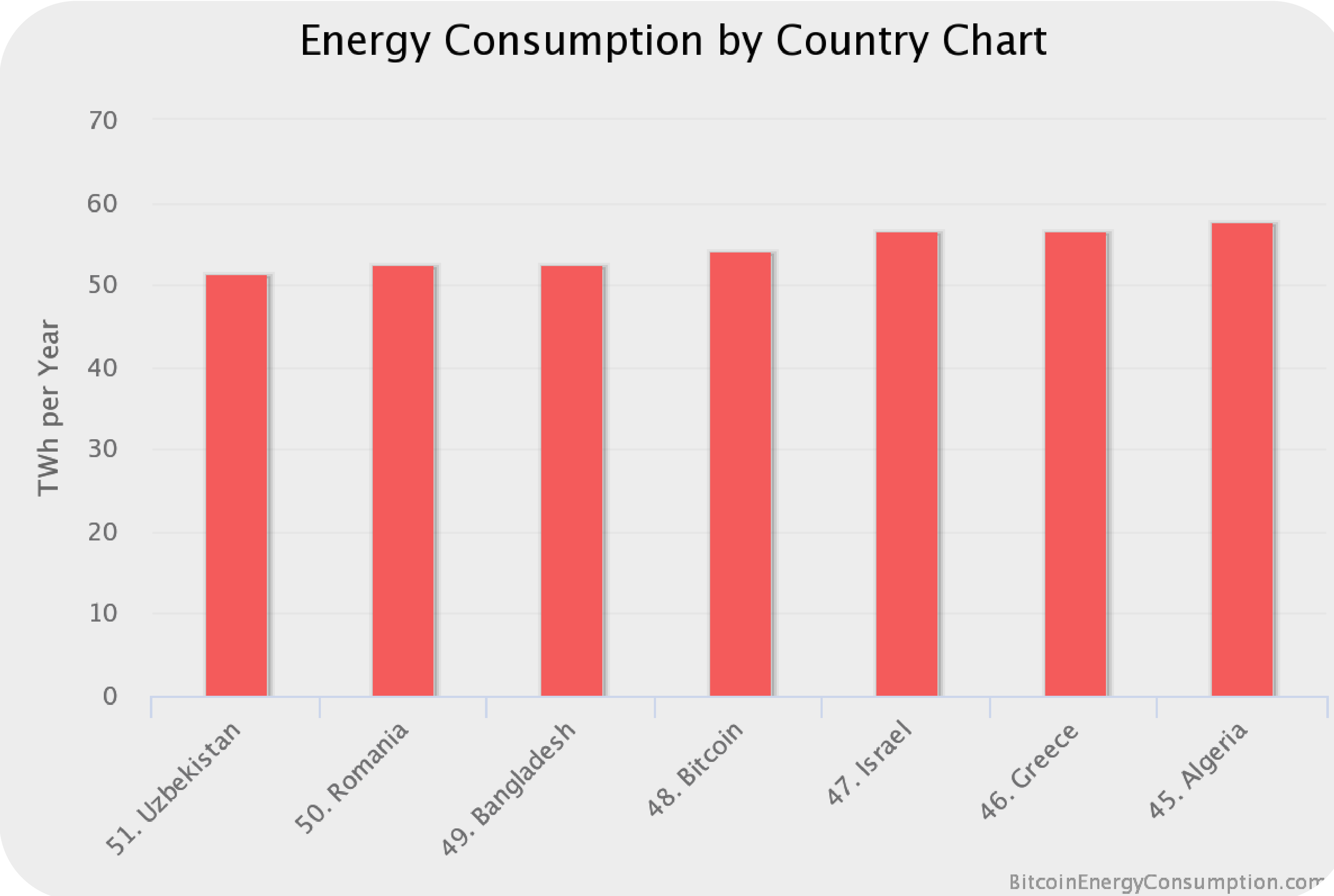
- 7 transactions/second
- VISA Network: 2000 tps (average)
- Limited block size (Segwit2x: 1MB -> 2MB)

Weaknesses of proof-of-work

- Environmental impact: ~1000x more energy than credit card
- Currently 43th in energy consumption (comparable to Switzerland)

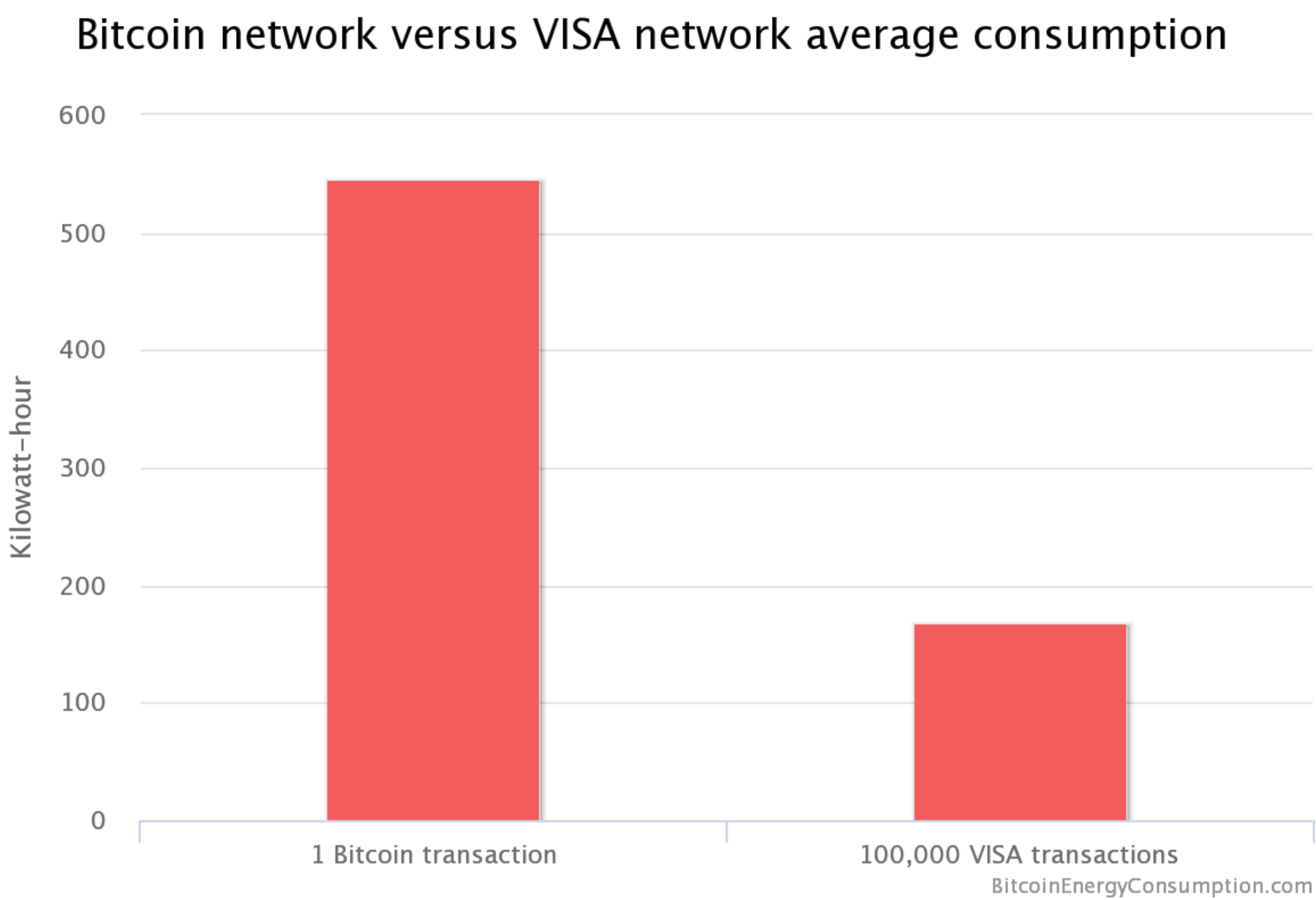
Long bootstrap time for a miner

- Full ledger: 164 GB (2018/04)
- CPU/IO cost to verify each transaction/block
- Takes hours/days



Energy Consumption by Country Chart
TWh per Year
70
60
50
40
30
20
10
0
51. Uzbekistan
50. Romania
49. Bangladesh
48. Bitcoin
47. Israel
46. Greece
45. Algeria
BitcoinEnergyConsumption.com

Bitcoin vs. VISA



blockexplorer.com

Transactions

f55bfe8fecd8d0634ed39efb7c5494f3a05037a9b076848c657633ec00895021

Input			Output	
1FGBxBpz9ccXQAFdHrXJVEFP4X2JG62pyU	23.91853748 BTC	›	1DsuGQc1yFn5WUT8Fy7yts8diRBHHRbygn	0.09 BTC (U)
			13BNhgrmJe9Sa1NweFsU6855REeRAAvaqi	23.82852033 BTC (U)

FEE: 0.00001715 BTC

UNCONFIRMED TRANSACTION! 23.91852033 BTC

db08215073e03a2a01bb6c26b3a1b3b1623fc4885681f4f462ddfabf0121dc0b mined Jul 18, 2018 9:19:47 PM

Input			Output	
19vZsqwAanT85MJALj3ymTxazeubyezpGi	24.41119681 BTC	›	15KJDwFsEQrCjRUbxfGXyg7iNktEYoGoFr	0.49263165 BTC (U)
			1FGBxBpz9ccXQAFdHrXJVEFP4X2JG62pyU	23.91853748 BTC (S)

FEE: 0.00002768 BTC

9 CONFIRMATIONS 24.41116913 BTC

Blockchain Systems

ETHEREUM

HYPERLEDGER

ETHEREUM

Managing entity: Ethereum Foundation

- Major players: Deloitte, Toyota, Microsoft, …

Focus: Open-source, flexible, platform

- Cryptocurrency: 1 Ether = 1e18 Wei (502 USD, 2018/04)
- Smart contracts: Solidity, Remix (Web IDE), Truffle (Dev./Test), *Vyper*
- Ethereum Virtual Machine (EVM), Ethereum Web Assembly (eWASM)
- Permisionless (public) ledger: Proof-of-Work, *Proof-of-Stake (Casper)*

Notes

- DOA Event: $150 million lost, hard forked into Eth. Classic
- GHOST Protocol: Merging of branches
- Ethash: Memory-hard hashing protocol which is ASIC-resistant
- *Scalability: L1 Sharding and L2 Plasma*

Evolution in business logic

Proliferation of Bitcoin spawn-offs

- Digital currency is not the only electronic object of value
  - Documents: authorizations, legal, diploma, design, various deliverables
  - Software
- Support for extended financial applications such as crowdfunding
- Support for multi-party escrow transactions

Ethereum envisioned that a single platform supporting the above is better than hundreds of specialized systems

- Provided a verifiable Turing-complete script language
- With script templates
- Scripts can be stateful, with a state stored on the chain

Benefits of smart contracts

Compared to a human intermediary

- Cheaper
- Open and transparent program that fulfils the contract and does nothing else
  - Does not peek into your data
- Highly resistance to attacks

Compared to distributed databases

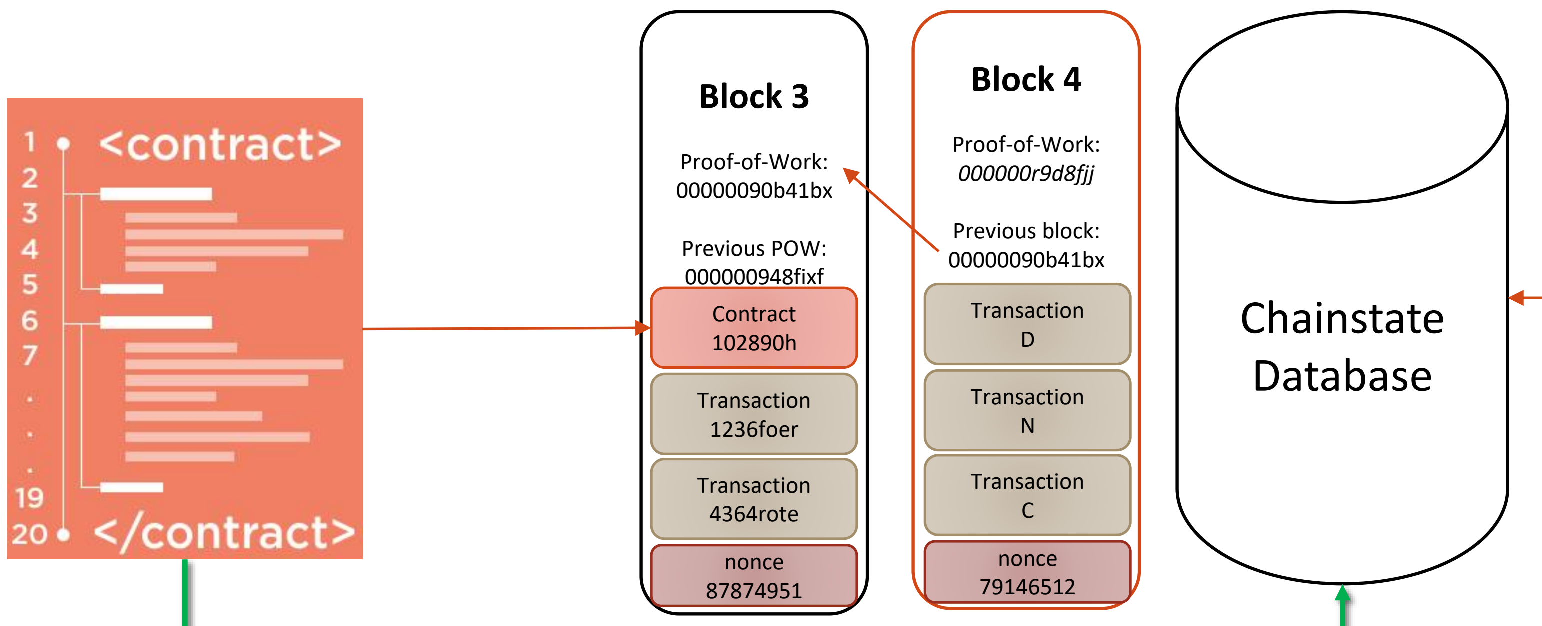
- Rule-based rather than data-based
- Rich language and (relative) easy of development
- The collection of rules is transparent and reusable
- May initiate and play an active role in the communication
- May integrate and fuse data from multiple sources

Smart Contracts

- Contracts contain *executable bytecode*
- Created with a blockchain tx
- Contracts have internal storage

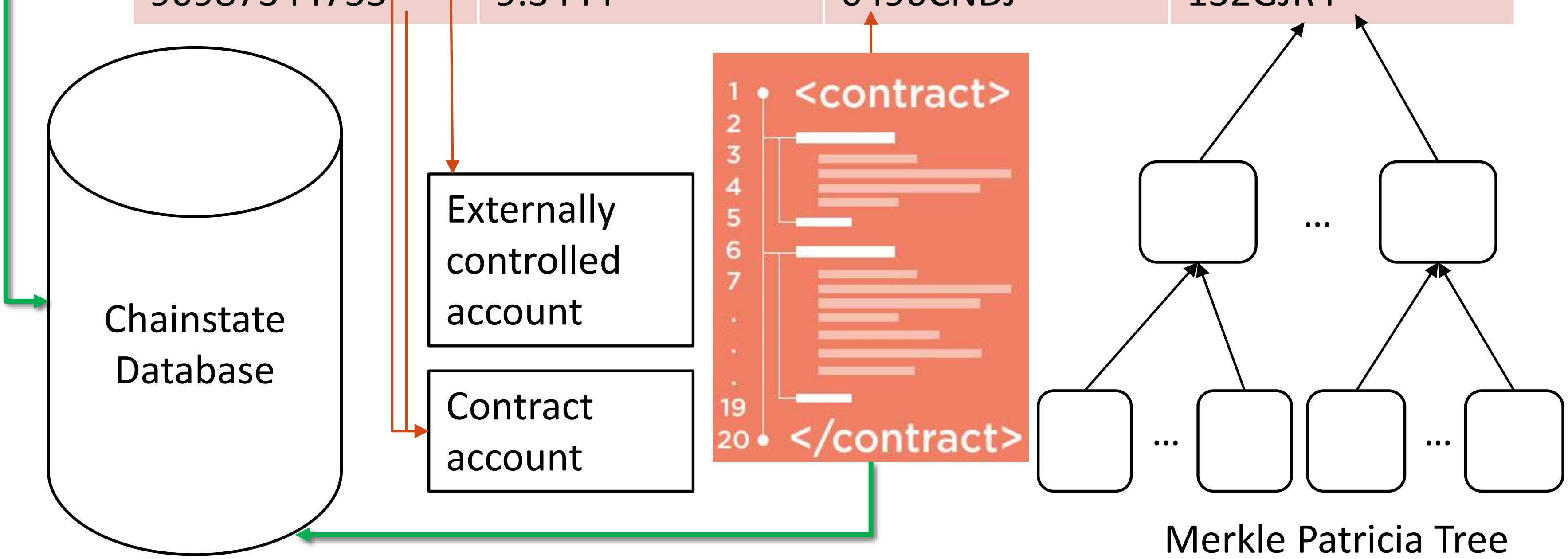
Contracts execute when triggered by a transaction (or by another contract)
Execution time is limited by *gas*
Example: Land registry

Wallet ID	Held Titles
99823428347	34356,324324
98217981623	677343,4444
90987344755	994,38842,439

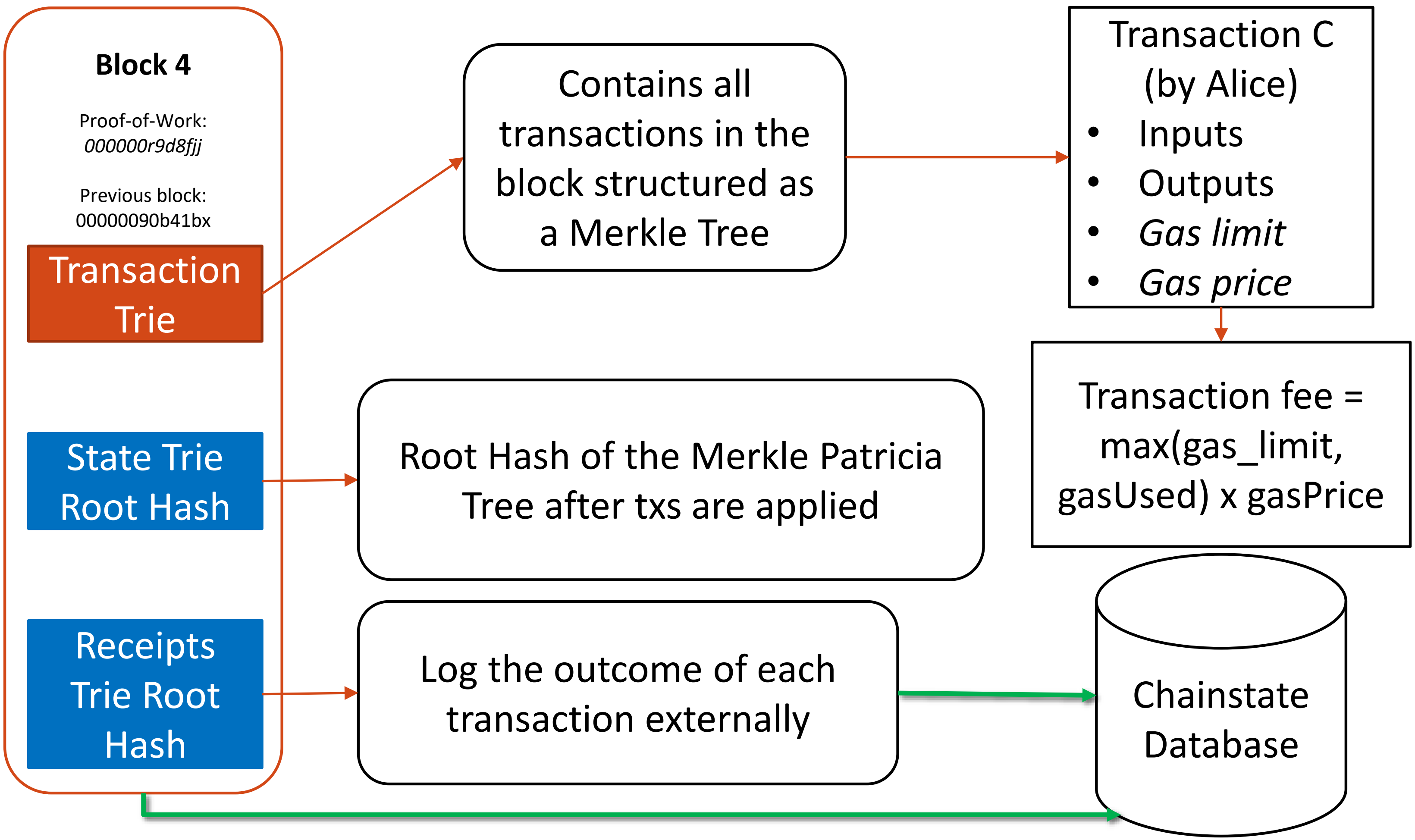


Account State ("World State")

Wallet ID	Balance	Code Hash	Internal State
99823428347	45.12	-	99554HGJ
98217981623	1123.332	9ERU12T4	3453ADFG
90987344755	9.3444	0490CNDJ	132GJR4

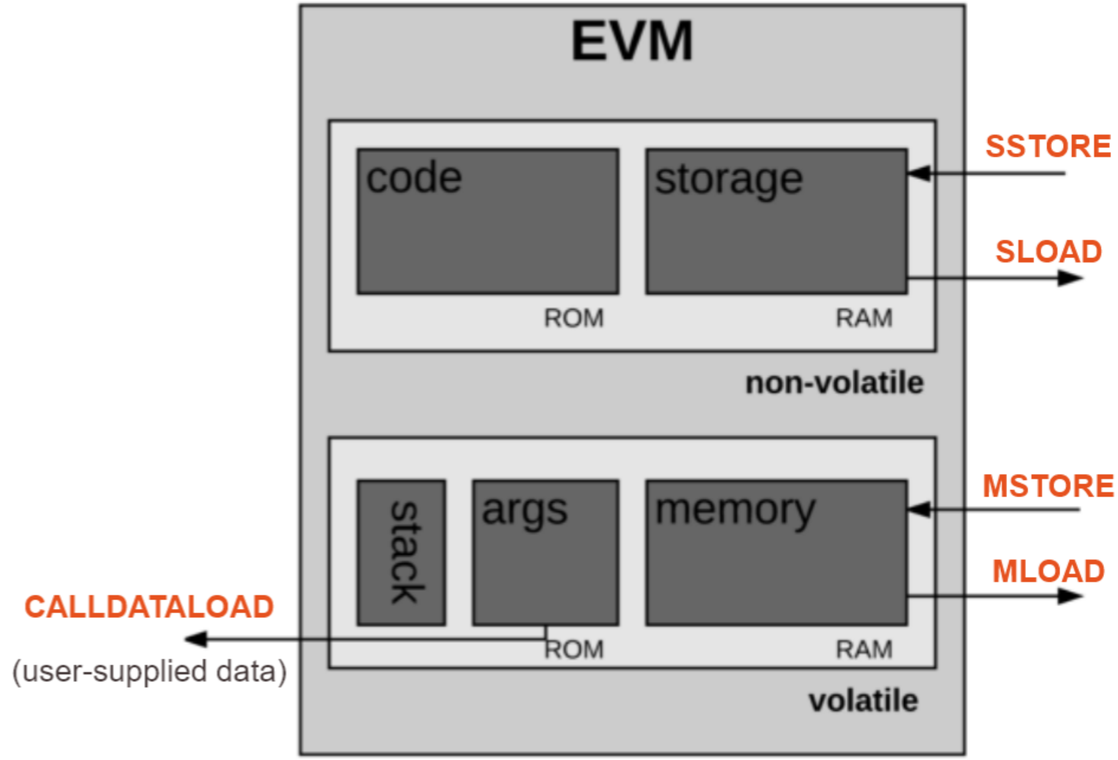


Execution and Mining



Ethereum Virtual Machine

Architecture		
Stack machine		
Turing complete		
Instruction set	~180 Opcodes	
Memory type		
Stack	volatile	byte-array (list [])
Memory	volatile	byte-array (list [])
Storage	persistent	key-value database (dictionary {})



Gas calculation

https://github.com/djrtwo/evm-opcode-gas-costs/blob/master/

Each OPCODE costs a different amount

The usage of each type of storage is measured

- Persistant storage is extremely expensive (SSTORE): 20K gas = 256 bits
- Memory is volatile (MSTORE)
- Stack is almost free, but very limited (cf. Bitcoin)

Compiler optimizes the bytecode based on the Solidity code written

- Important to use the right keywords to allow the compiler to optimize properly! (c.f. last session)

Comparison with Bitcoin

	Bitcoin	**Ethereum**
Transactions	Transfer of bitcoins	*Contract creation*, transfer of ether, *contract calls, internal transactions*
Accounts	User wallets	Externally owned accounts, *contract accounts*
Transaction fees	Amount specified by sender	Gas calculated using sender's values
Block content	Transactions trie	Transactions, *State Root Hash, Receipts Root Hash*
Chainstate Database	World state: UTXOs for wallets	World state, *receipts, bytecodes for contracts*
Querying	Simple Payment Verification	Merkle proofs for *events*, transactions, *balance*, etc.

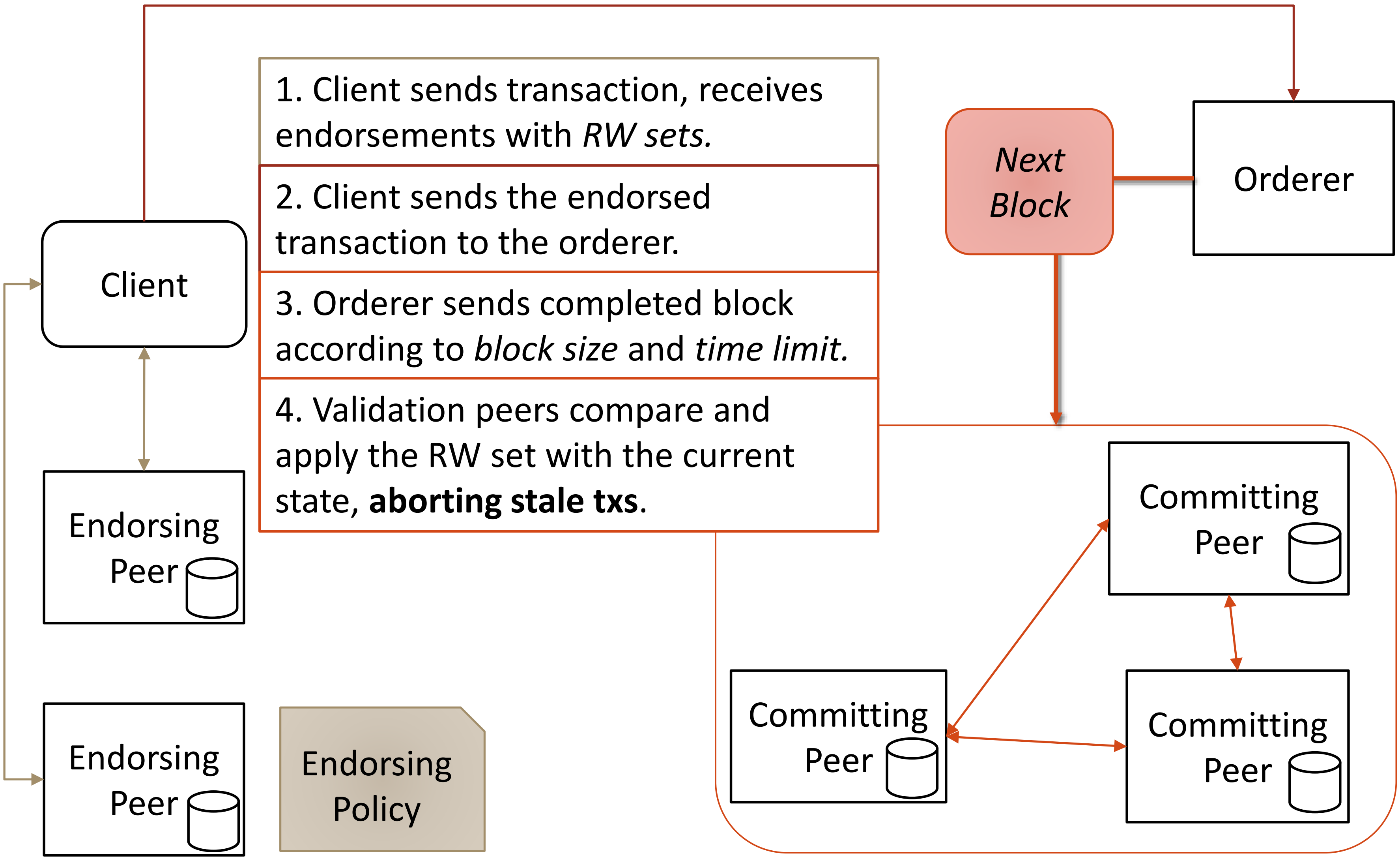
HYPERLEDGER

Managing entity: Hyperledger Consortium
- Major players: IBM, NEC, Intel, R3, …

Focus: Enterprise blockchains
- Permissioned ledger (private/consortium network)
- Smart contracts
- Open-source
- World state on CouchDB/LevelDB, event listener

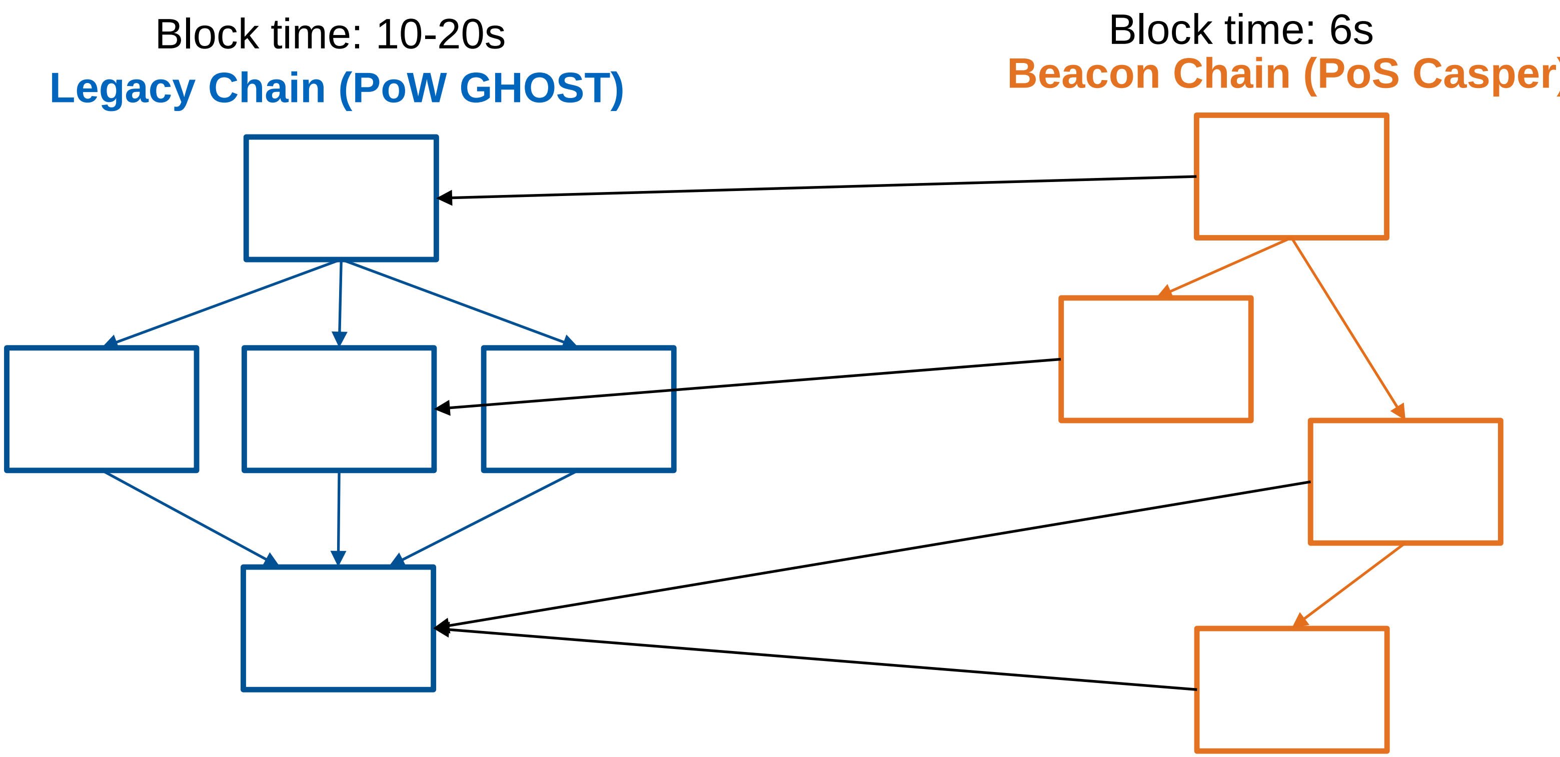
Projects
- Fabric: Execute-Order-Validate transaction processing
- Sawtooth: Proof-of-Elapsed-Time (using Intel SGX)
- Composer: Smart contract language and development tool
- Cello: Blockchain-as-a-Service framework
- R3 Corda: Financial applications

Fabric: Transaction processing flow



Ethereum 2.0

Integration of Casper Consensus with Legacy Chain



Details on Casper Consensus

The beacon chain is maintained by *validators* registered in a validator set

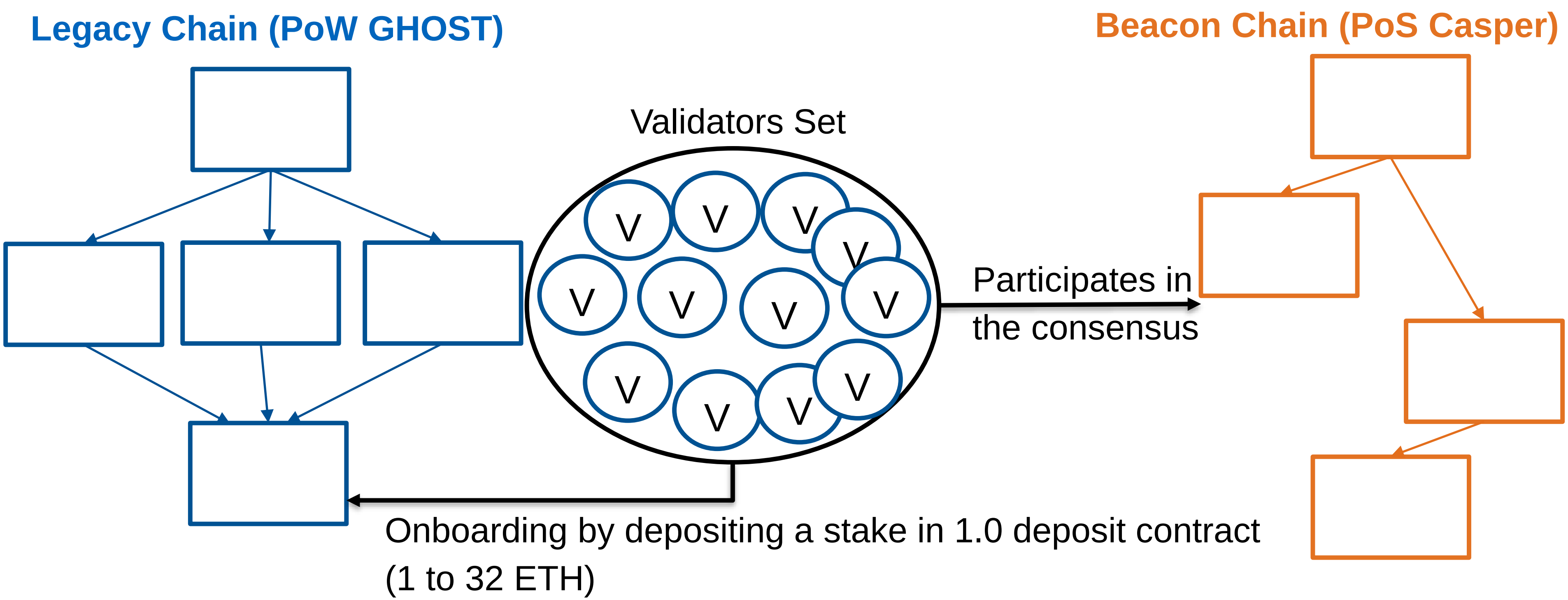
- **Stake deposit** of 1-32 ETH into a contract on the 1.0 legacy chain

“Finality Gadget” allows for legacy chain blocks to be finalized: cannot be reverted

- A stronger form of the “confirmation wait” mechanism used in 1.0 or Bitcoin
- Allows for data to be pruned beyond the latest finalized block
- **Point of no return** assuming 2/3 honest validators

fuseelab.github.io

Validator Registration



Epoch State Transition

Beacon chain progresses through epochs

- Each epoch has 64 slots, each slot last 6 seconds
- Model assumes validators clocks are synchronized within 6s

During an epoch boundary, execute a deterministic state transition function:

- A random number generator seed is chosen (using RANDAO as randomness source)
- The validator set is randomly shuffled into committees and proposers, and assigned to slots
- Each committee size may vary but generally aims to have 256 attestators

Slots and Proposers

Each slot:

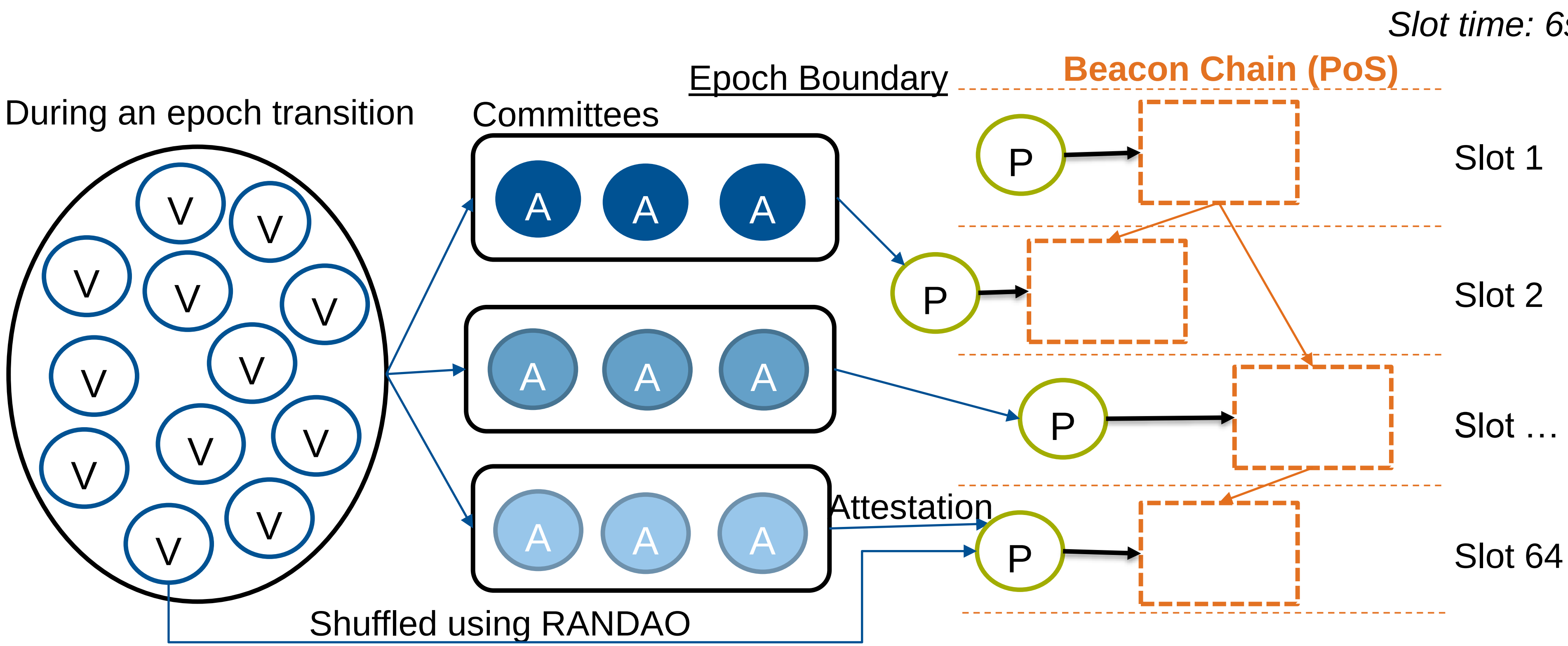
- has at least 1 committee, up to 16
- has a proposer chosen from the validator set

Each proposer:

- Will try to propose one beacon chain block, attached to its known head of the beacon chain
- Will collect attestations from each committee assigned to previous slots

fuseelab.github.io

Beacon Chain Details



Attestations

Each attestator publishes an attestation when during its slot time:

- The attestation records the latest block(s) perceived by the attestator at the time
- Contains a history up to the last known finalized block

Attestations are collected by proposers in **future slots**

- Minimum delay to respect is 4 slots (24 seconds)

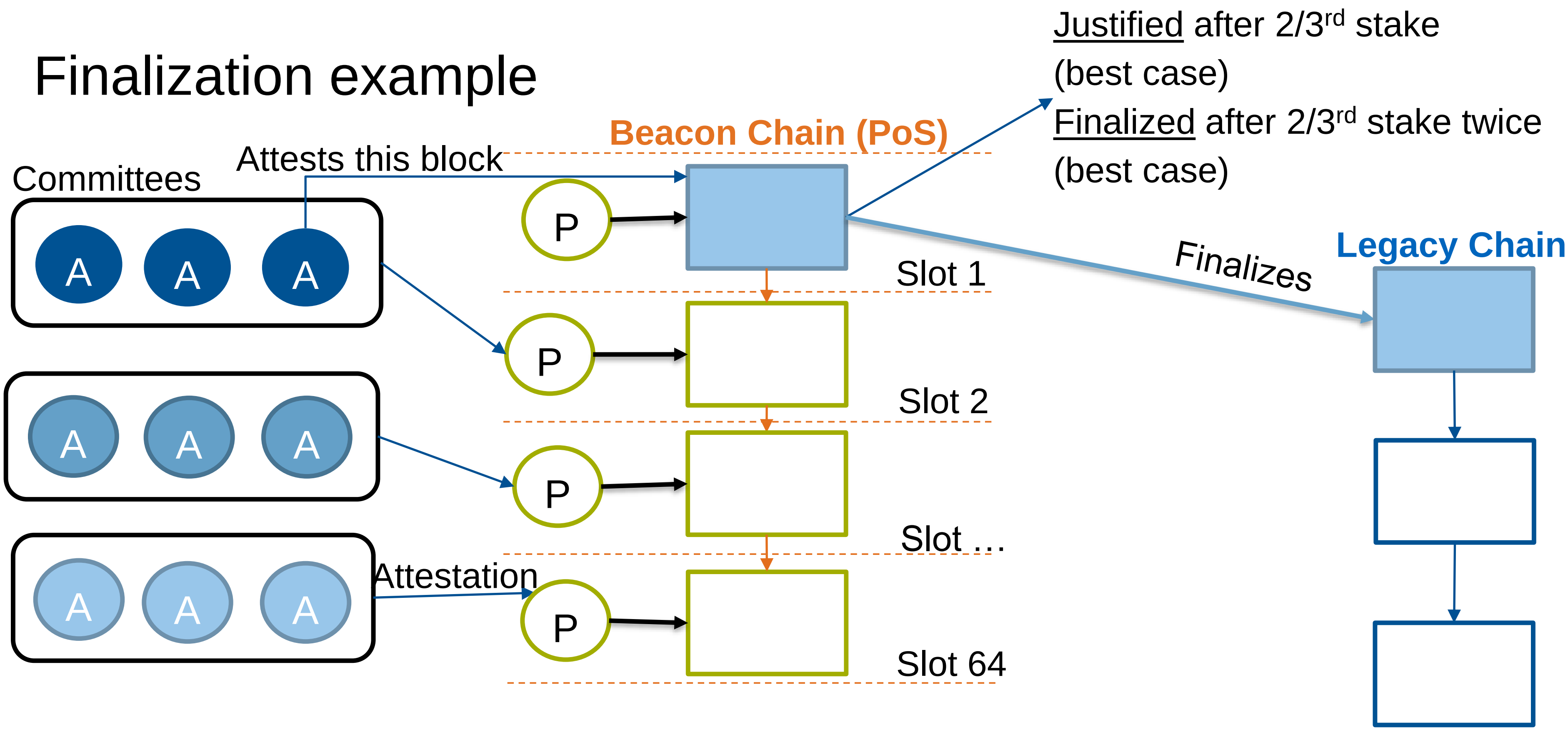
Proposer aggregates received attestations and put them in its beacon chain block

Justifications and Finality

Block properties (evaluated during epoch transitions):

- A block is justified if it has 2/3rd stake of attestations of the **entire validating set**
- A block is finalized if it has 64 justified children (64 consecutive justified blocks)
- Once finalized, the **attached legacy block is also finalized**
- The data contained in that legacy block **cannot be reverted**

Finalization example



Backup Slide

Consensus: Fork Choice Rule

Due to network partitions, latency, crash and byzantine failures, etc…:

- Possible that a slot does not have a block
- Possible that several beacon blocks reference the same parent, causing a fork

Validator use IMD-GHOST to choose the correct fork:

- Immediate Message Driven Greedy Heaviest Subtree *(IMD-GHOST)*
- Measures closest proximity to justification for each subtree

Adoption by honest participants with 2/3 stake to provide **probable liveness**

- Blocks will continue to be justified and validated

Rewards for Participating in Casper FFG

Reward function applied during epoch state transition:

- Target: 12% to 15% interest rate on your deposit per year (for optimal case)
- Non participation (did not publish attestation or did not propose block) punished by slashing stake
- Payout is based on participation rate for that cycle (% of validators who were active)
- Gain rewards for attesting, justifying, and finalizing blocks

This incentive structure is resilient to denial attacks

- Proposers will not omit attestations as it decreases its own payout
- Attestators will not withhold attestations as they may get slashed

Layer 1 Scalability: Sharding with Casper FFG

Goal: Replace legacy chain with multiple shard chains, in order add parallelism for transaction processing

Partition the world state into disjoint shards:

- Each shard chain processes transactions independently for its share of the state
- Execute smart contracts for that partition
- Allow us to obtain $\mathcal{O}(n) > \mathcal{O}(c)$ global capacity
- Limited cross-shard communication possible (future work to improve it)

How to leverage Casper FFG and the Beacon Chain to finalize shard blocks?

Crosslinks + Casper Consensus

Each committee is assigned to a shard during an epoch

- Determined randomly at the state transition (using RANDAO)

Current design: 1024 shards

- Ideally, 16 committees per slot, so that each shard is included in each epoch

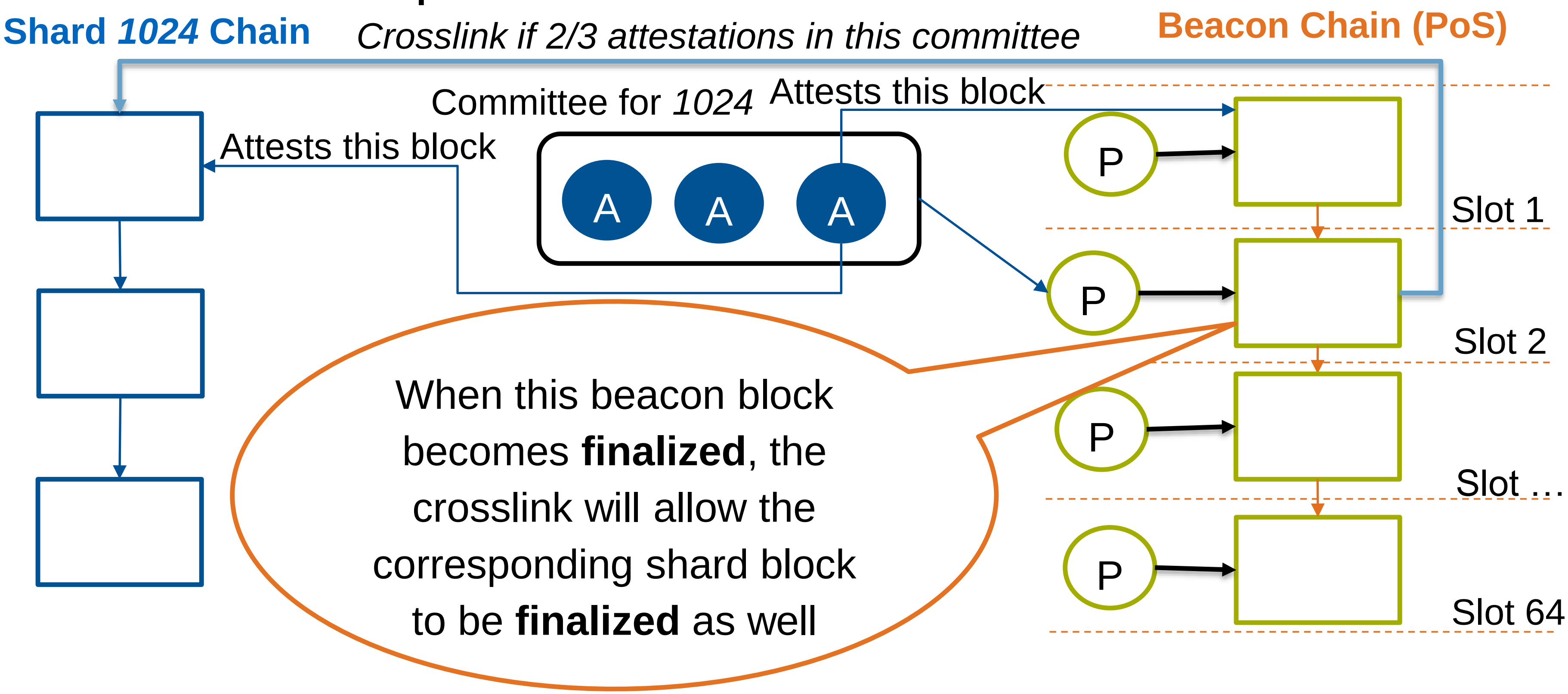
Crosslink between a shard block and a beacon block:

- Each attestation references a specific shard block in the committee shard
- 2/3 of committee attested to the same shard block: Cross-link created with beacon block

Shard block finality

- A beacon block is **finalized** which contains the **crosslink** to the shard block

Crosslink Example



Ethereum 2.0

Consensus: Shard Fork choice rule

- Shard fork-choice rule depends on Beacon chain
- IMD-GHOST starting from last finalized cross-link

Cross-shard communication

Each shard block references a beacon chain block to reference the RNG of the beacon chain

Cross-shard communication: asynchronously through cross-links

Could be realized through events emitted are perceived in another shard at the next CSC opportunity

More work is needed to make it faster

fuseelab.github.io

MIDDLEWARE SYSTEMS RESEARCH GROUP MSRG.ORG

Overview of Delayed State Execution

Unaddressed Questions in the Specifications

How to propose blocks in a shard?
How to choose the order of transactions?
How to execute the transactions and transition the state of the shard?

Our proposed solution: *Delayed state execution*

- Original idea by Vitalik Buterin
- We designed a working solution under the current 2.0 specifications
- **Solves the Data Availability Problem**

Delayed State Execution: Overview

Main idea: separate transaction execution (state storage) / block ordering / transaction storage

- Transactions within a block won't be fully processed until two blocks later!
- Each block pipelines information about transactions who are at various stages

Role of an *executor node*:

- Executors must be onboarded with a stake deposit
- Randomly assigned to a shard, but infrequently reshuffled (once every 3 days)
- This slow churn allows sufficient time to synchronize shard data and mitigate overhead

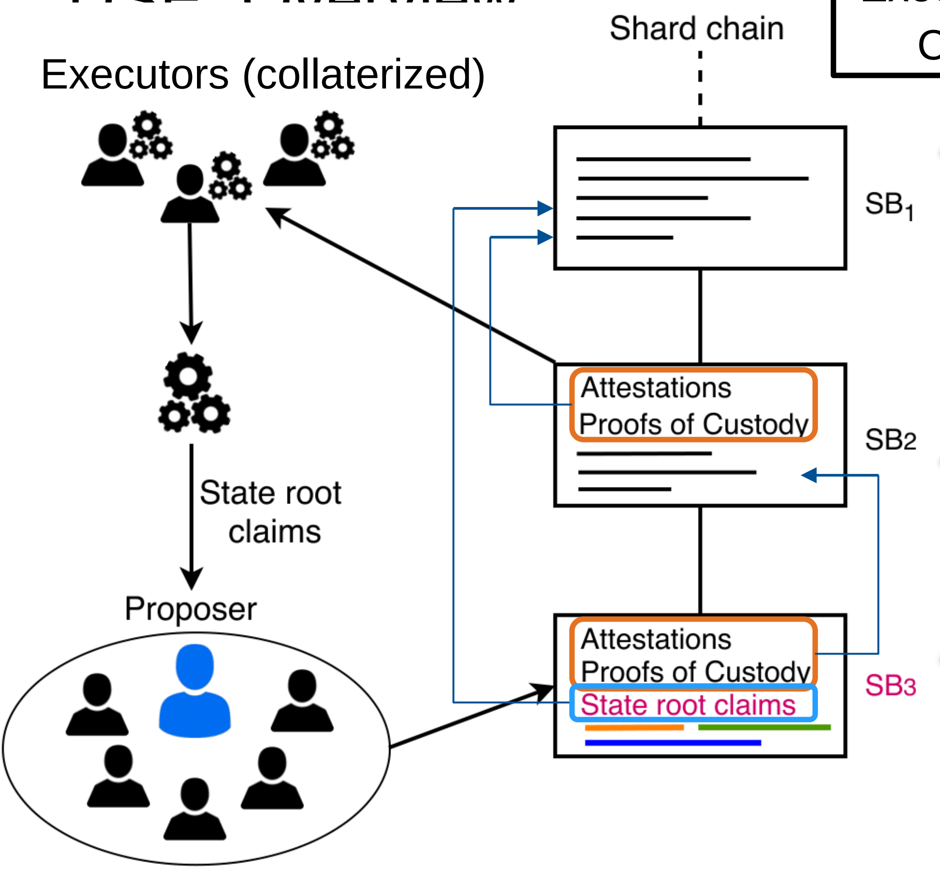
Validator committee for a shard:

- Chosen from validators **not** in a Casper committee at that slot
- Contains a proposer, who receives transactions and ordered lists from executors
- Contains attestators, who will provide Proof-of-Custody (invented by Justin Drake)

fuseelab.github.io

DSF: Overview

Executors are shuffled every 3 days
Committee shuffled every slot



Committee(Different from beacon committee!)

Contains final transaction order, aggregated from executors' suggestions

Attestations contain Proofs of Custody that committee will keep transaction data from SB1 for 1 month

State root claims contain final state root hash for SB1, in case of conflict, request witness data and slash offenders

Delayed State Execution: Overview (ii)

In the next block *n+1:*

- The rest of the validator committee attest the content of this block
- The attestation contains a Proof-of-Custody (invented by Justin Drake)
- The validator committee promises to keep the transactions in this block available for 1 month
- Can be challenged to demonstrate availability
- Attestations about block *n* are stored in block *n+1*

In the next next block *n+2*:

- Executors read the transaction order from block *n* and verify attestations from block *n+1*
- Executors calculate the final state of the trie by executing transactions in block *n* <u>with the order written</u>
- Executors send a **state root claim** to the proposer
- **State root claims** about block *n* are stored in block *n+2*

Conflicting claims:

- Validators request witness data and execute transactions → Malicious executor(s) slashed

Properties of our Proposed Approach

Economic Safety: Data Availability

Initially Unavailable Data

- Requires attacker to have 100% control over at least one attestation committee

→ Requires attacker to have almost 100% of entire validator set

→ *Breaks 2/3 honesty assumption and is more difficult than consensus failure*

Lost Data

- Attacker bribes entire executor set to delete state data

→ State data from last month can be reconstructed through validators

→ Only targeted attack vector against old and infrequently accessed state

→ *Requires victim to stay offline for a month or not store its own data (negligence)*

Ethereum 2.0

Supporting Technologies

- RANDAO
  - Distributed Random Number Generator (RNG)
  - Generates global entropy by combining local entropies
  - Hash Onions from every validator as local entropy sources
- BLS Signature Aggregation
  - Elliptic Curve Cryptography based on Gap Groups
  - Constant-size aggregated signatures for *n* signers
  - Efficient aggregation and verification of aggregate signatures

Backup Slide

Ethereum 2.0

Supporting Technologies: RANDAO

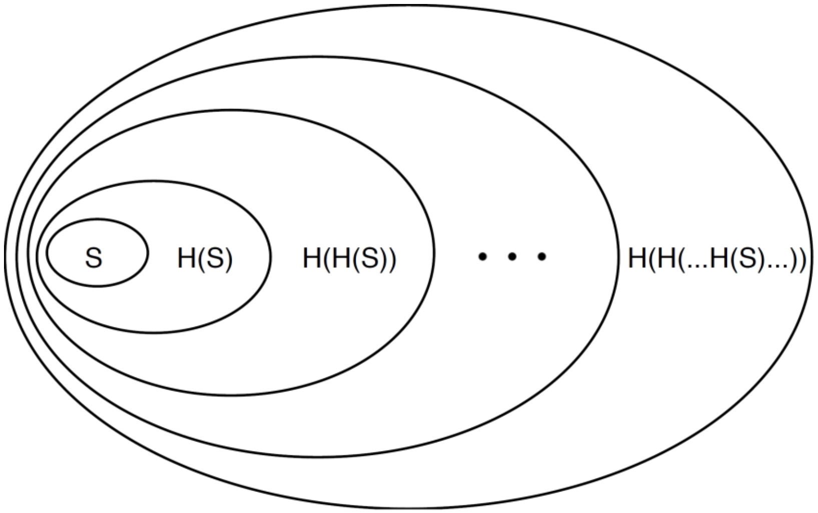


Figure: Hash Onion

Blockchain Insights

BENEFITS AND CHALLENGES

TAXONOMY OF BLOCKCHAINS

RESEARCH OPPORTUNITIES

New challenges introduced by DLTs

Compared to databases

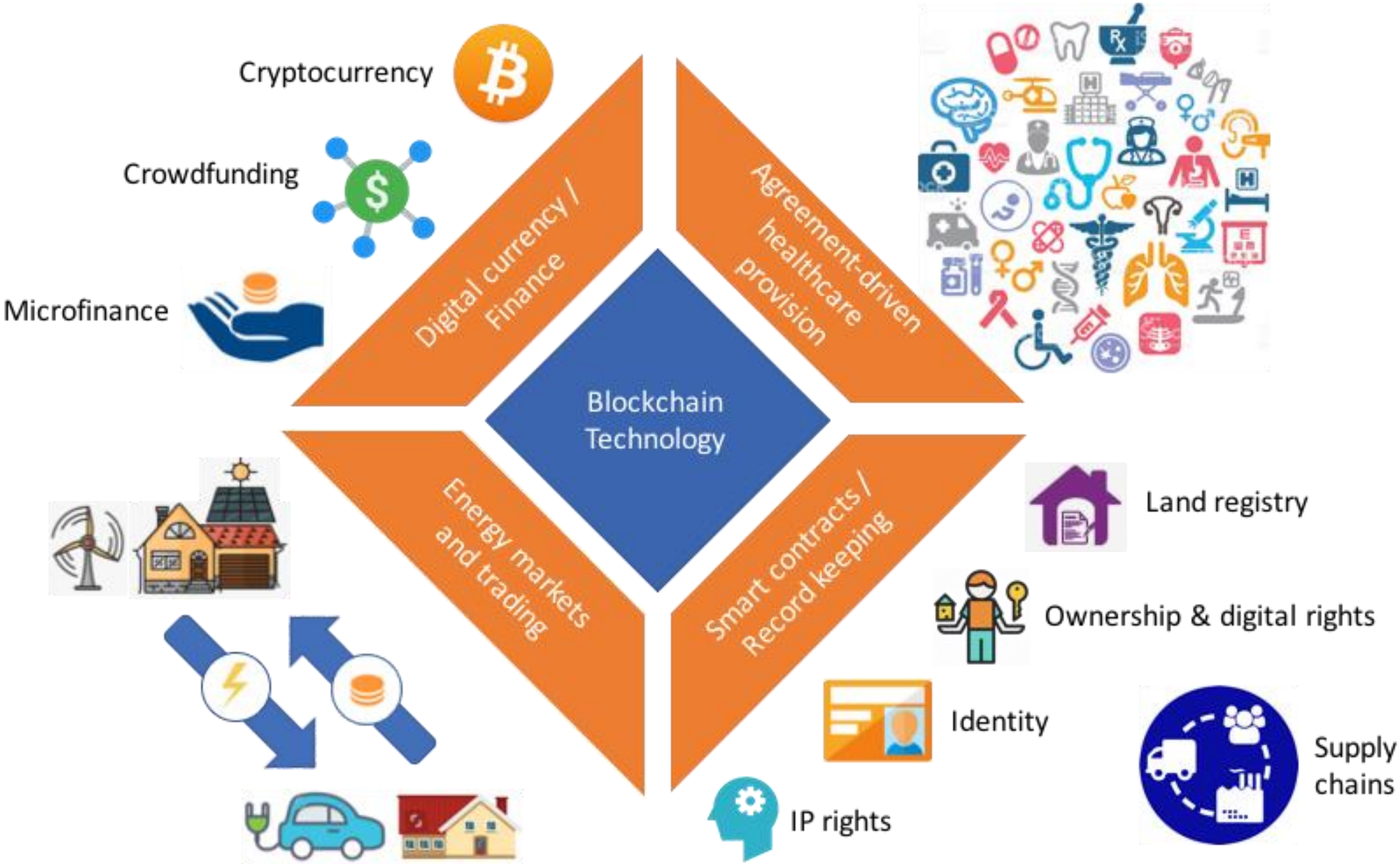
- Slower
- Lower rate of transactions
- Less compact storage

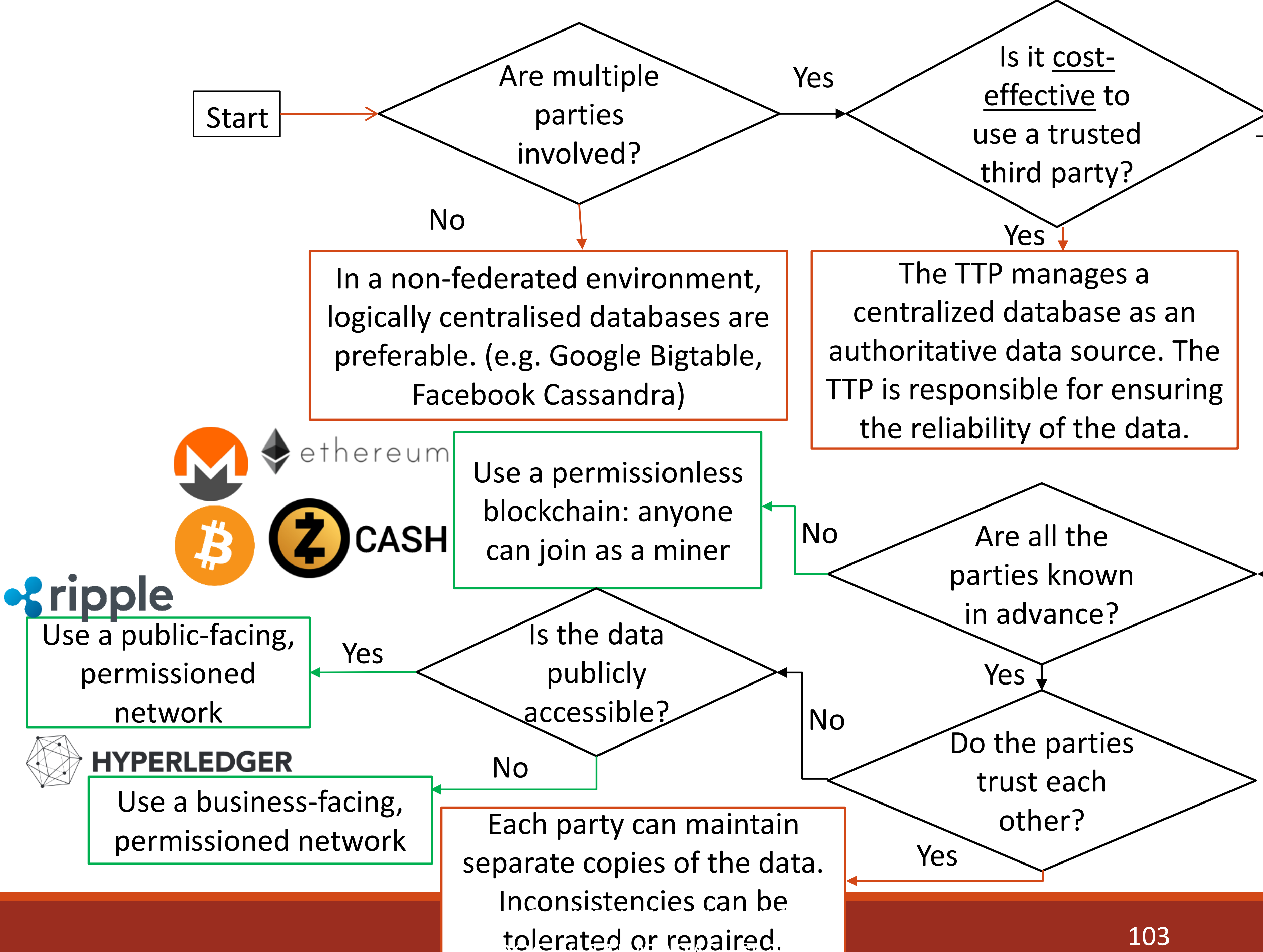
The technology and even standards (and even terminology) are still developing

Additional challenges related to smart contracts

- Bug prone, no established programming or verification practices
- State machine execution, with each contract replica performing every action
- If a contracts interacts with an external non-blockchain service, this service needs to be designed with this in mind

Versatility and potential




Start
Are multiple parties involved?
Yes
No
Is it cost-effective to use a trusted third party?
Yes
In a non-federated environment, logically centralised databases are preferable. (e.g. Google Bigtable, Facebook Cassandra)
The TTP manages a centralized database as an authoritative data source. The TTP is responsible for ensuring the reliability of the data.
ethereum
CASH
ripple
Use a permissionless blockchain: anyone can join as a miner
No
Are all the parties known in advance?
Yes
Use a public-facing, permissioned network
Yes
Is the data publicly accessible?
No
Do the parties trust each other?
HYPERLEDGER
No
Use a business-facing, permissioned network
Each party can maintain separate copies of the data. Inconsistencies can be tolerated or repaired.
Yes

Taxonomy

	Anyone can read	**Read access restricted**
Anyone can propose updates	Bitcoin, Ethereum	Ethereum (Smart Contracts)
Update access restricted	Ripple	Hyperledger, Corda

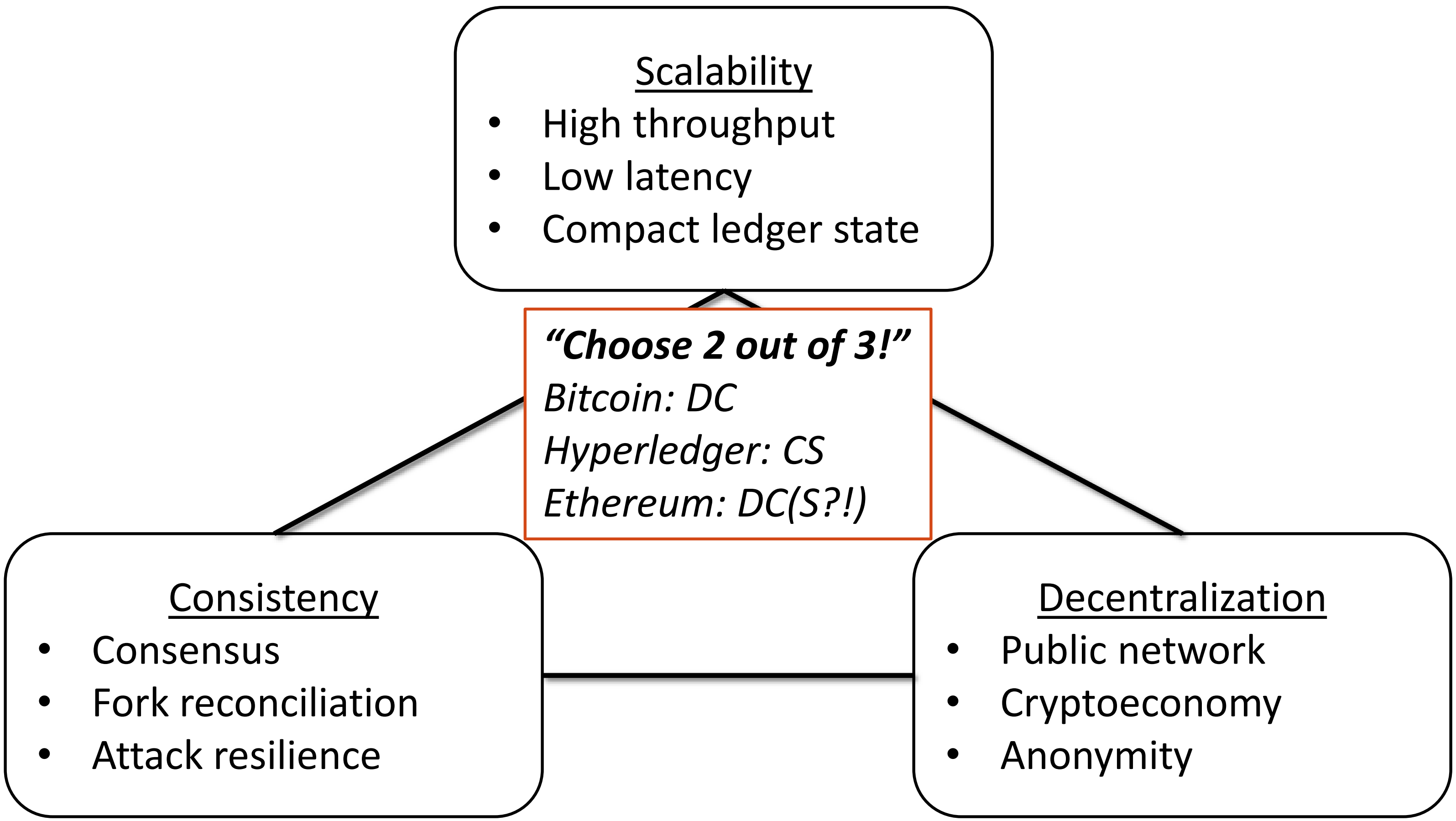
A related feature is if authentication is required

The above is well defined, but has no common terminology associated with it

Journalists use other terms instead: open/closed, permissioned/permissionless, public/private

Decentralization: centralized, large-scale decentralized, and consortium blockchains

"CAP Theorem" for DLTs



MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

DCS Conjecture

Safe and verifiable smart contracts
Attacker models: <51% attacks
Security of off-chain services (e.g. exchanges)
“Garbage in, garbage out”: IoT barrier

Incentives, mining rewards
Privacy: Anonymity, fungibility
Endorsement policies, governance
Selective replication: State channels

Consistency

Sharding, sidechains, tree-chains, …
Large-scale chainstate storage
Big Data analytics
Layer 2 Network: Lightning, Raiden
Proof-of-Stake, POET, PBFT, …

Decentralization

Scalability

“Choose” 2 out of 3!

Bitcoin: DC
Hyperledger: CS
Ethereum: DC*(S?!)*

Investigate **potential use cases**
Choose and **tune** the right platform
Develop **reusable middleware**

Applicability of blockchains

- DCS: May lead to fundamental research
- Applications: mostly 3.0, and some 2.0
- Layers: application, modeling, contract

Blockchain middleware

- Applications: 1.0 – off-chain exchanges and payment networks, 2.0 – reusable online services, 3.0 – data integration, analytics
- Layers: contract

Security and privacy

- DCS: +DC, -S
- Applications: 1.0 –transactions, 2.0 – smart contracts, 3.0 – data privacy
- Layers: contract, system, data, (network)

Scalable system innovations

- DCS: +S, -DC
- Applications: 1.0 – incremental, 2.0 – public smart contracts, 3.0 – clean slate designs
- Layers: system (consensus), data

Blockchain 1.0: Currency

Over 13700 public cryptocurrencies available!

Research for 1.0 Apps

Formally analyze the *security* model of Bitcoin

- 51% attack
- DoS attacks on: mining pools, currency exchanges, …

Conduct *performance modelling*

- Simulate various Bitcoin scenarios
- Understand impact of network topologies (e.g. partitions)

Develop *scalable* mechanisms with *legacy support* to maintain the *sustainability* of Bitcoin

- SegWit2x
- Bitcoin-NG (NSDI ‘16)
- Off-chain (Lightning network)
- Algorand (SOSP ‘17)

Blockchain 2.0: Decentralized Apps

ĐApps are applications built on blockchain platforms using smart contracts (e.g. Ethereum)

Ðapps

ETHEREUM

GNOSIS

Forecast market (e.g. betting, insurance)

EtherTweet

Decentralized Microblogging

Token Distribution

Crowdfunding

alice

Charity donation payment

Research for 2.0 Apps

Formal *verify* smart contracts, detect and repair security flaws

- Ethereum Viper

Develop *scalable consensus* mechanisms which support *smart contracts* in an *public* network (w/ *incentives*)

- Proof-of-Stake (Casper)
- Side-chain (Plasma)
- Sharding (ShardSpace)

Develop *efficient data storage* techniques to store *smart contracts* and the *chainstate*

- AVL+ (Tendermint)
- Merkle Patricia Trees (Ethereum)
- Zero-Knowledge Proofs: zk-SNARK

Blockchain 3.0: Pervasive Apps

everledger

Diamonds Provenance

Applications involve entire industries, **public sector**, and IoT.

FACTOM

Land Registry in Honduras

BlockchainHealth

Electronic Health Records

VOTEWATCHER

Transparent Voting System

Research for 3.0 Apps

Develop *"clean-slate"* scalable distributed ledgers:

- Permissioned ledgers (Hyperledger Fabric)
- Blockless DLTs (IOTA Tangles, R3 Corda Notaries, Hashgraph)

Develop *blockchain modelling tools and middleware*

- BPMN, Business Artifacts with Lifecycles, FSM
- Authentication, reputation, auction, voting, etc.

Support strict *governance, security, and privacy* requirements

- State channels
- Endorsement policies

Overcome the *cyber-physical barrier for data entry*:

- Object fingerprinting
- Secure hardware sensors

HyperPubSub: A Decentralized, Permissioned, Publish/Subscribe Service using Blockchains

ONGOING RESEARCH

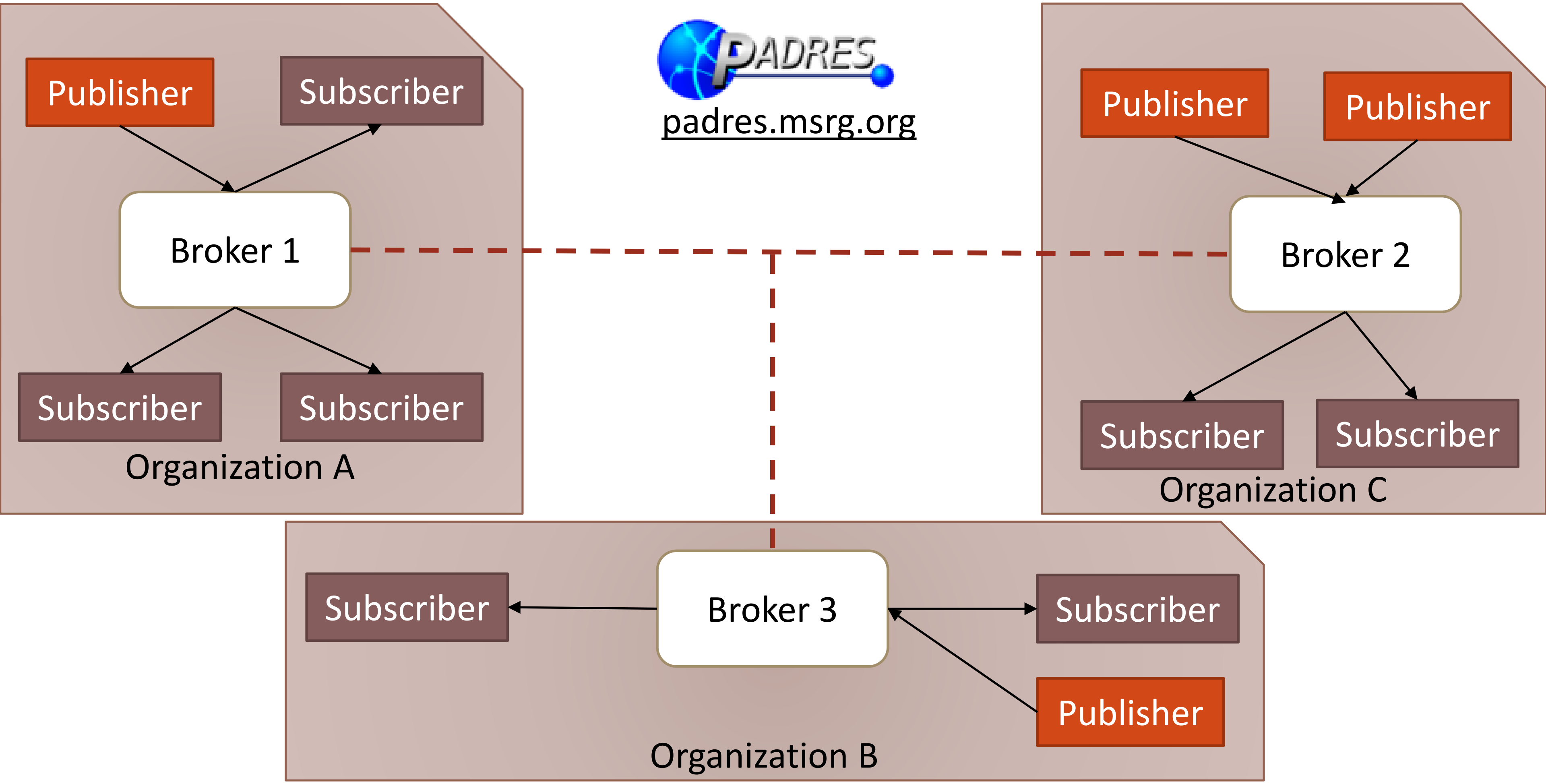
NEJC ZUPAN

KAIWEN ZHANG

HANS-ARNO JACOBSEN

Motivation: Federated Messaging

padres.msrg.org



Use cases

Trusted communication in federated systems*:* Allow for cross-organizational communication which tolerates Byzantine failures.

Client-driven auditing*:* Allow clients to obtain a trail of messages sent and received, to ensure complete publication delivery and data verification

Data marketplace*:* Publication delivery can be monetized; publishers can verify accurate payment for all deliveries, while subscribers can verify correct billing for received messages

HyperPubSub

Hyperledger Fabric (1.0)-based pub/sub system:

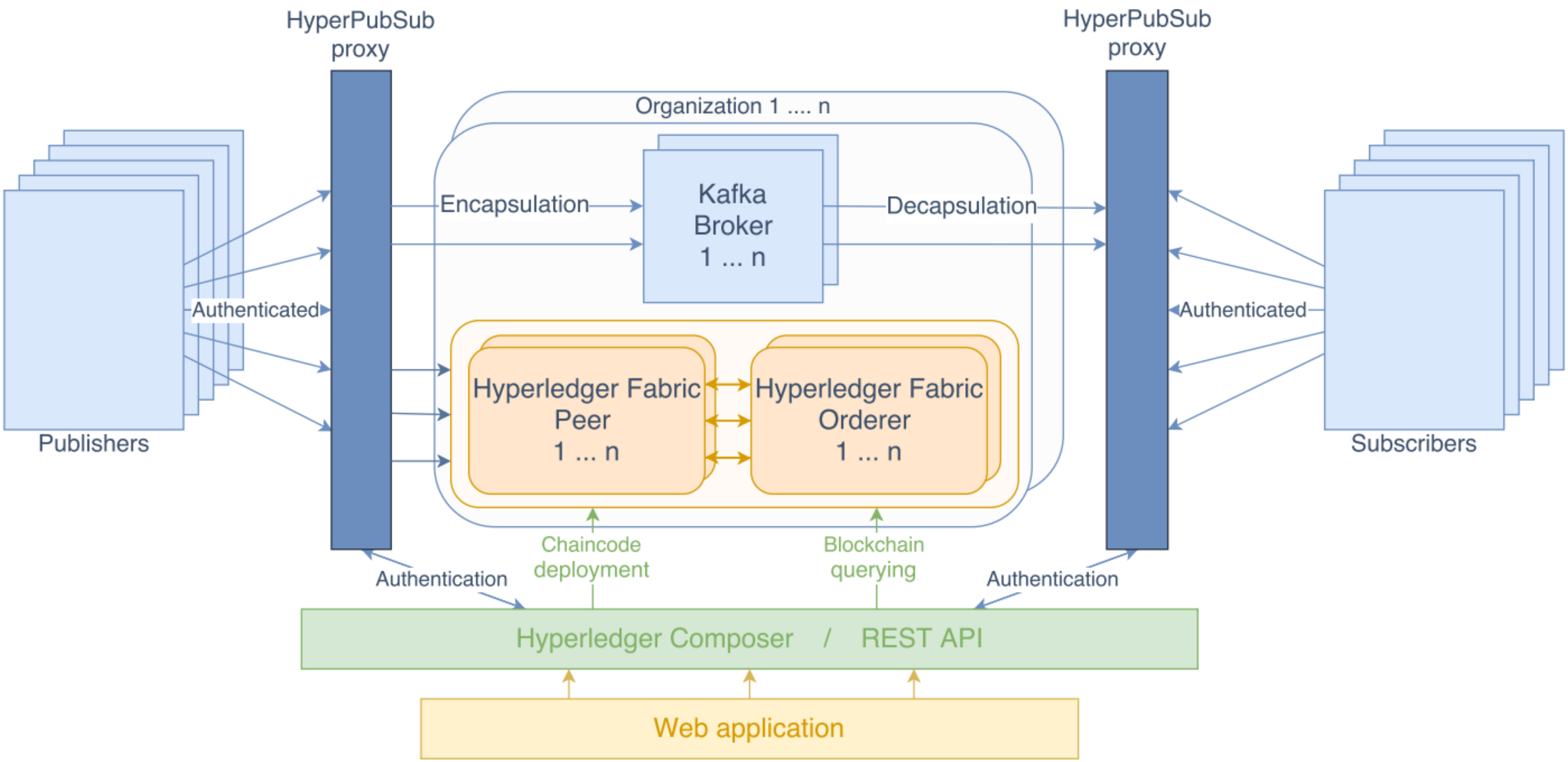
- ***Modular pub/sub component:*** currently Kafka (topic-based)
- ***Out-of-band matching logic:*** async. Composer chaincode to minimize overhead during online pub/sub operations
- ***Privacy-preserving pub/sub:*** Access control, authentification
- ***Asynchronous client API:*** Auditing past history

Web demo (using Playground) for publishers and subscribers to:

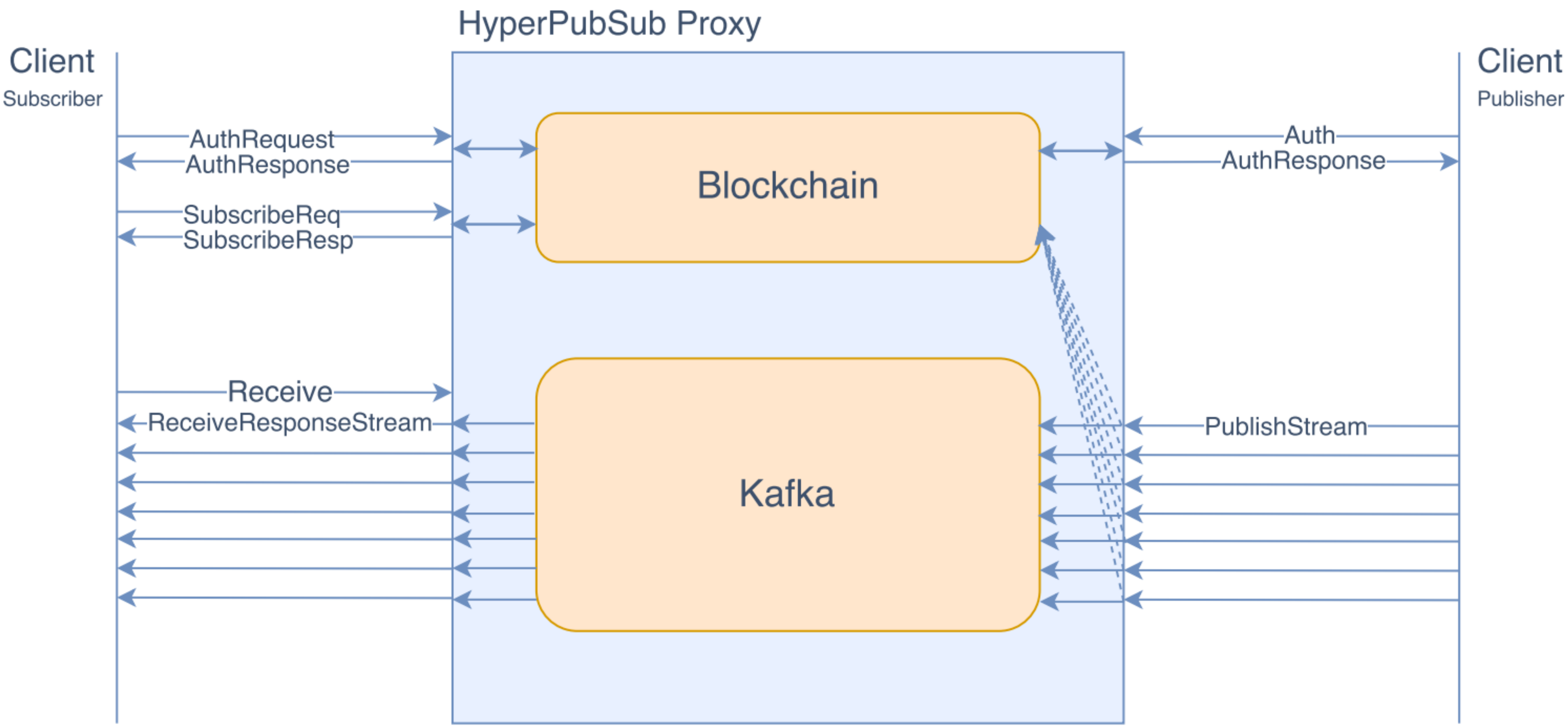
- Check for complete delivery
- Validate consumed data
- Verify system status

Diverse language support: gRPC (Protobuf) connectors

HyperPubSub Architecture



Pub/Sub Protocols



Participants and Assets

Participants:

- Publisher
- Subscriber

Transactions:

- Publish
- Subscribe
- Register

```
asset Publication identified by publicationId {
  o String publicationId
  o DateTime time
}
```

Created during publish: visible to matching publisher and subscribers, contains publication hash

```
asset ExtendedPublication identified by id {
  o String id
  o Integer count default=0
  --> Publication publication
}
```

Created during publish: exposes number of matching subs (but not their identity), for monetization

Participants and Assets

```
asset Topic identified by id {
  o String id
  --> Subscriber[] subscribers
}
```

List of topics with subscribers, used by the smart contract for topic-based matching

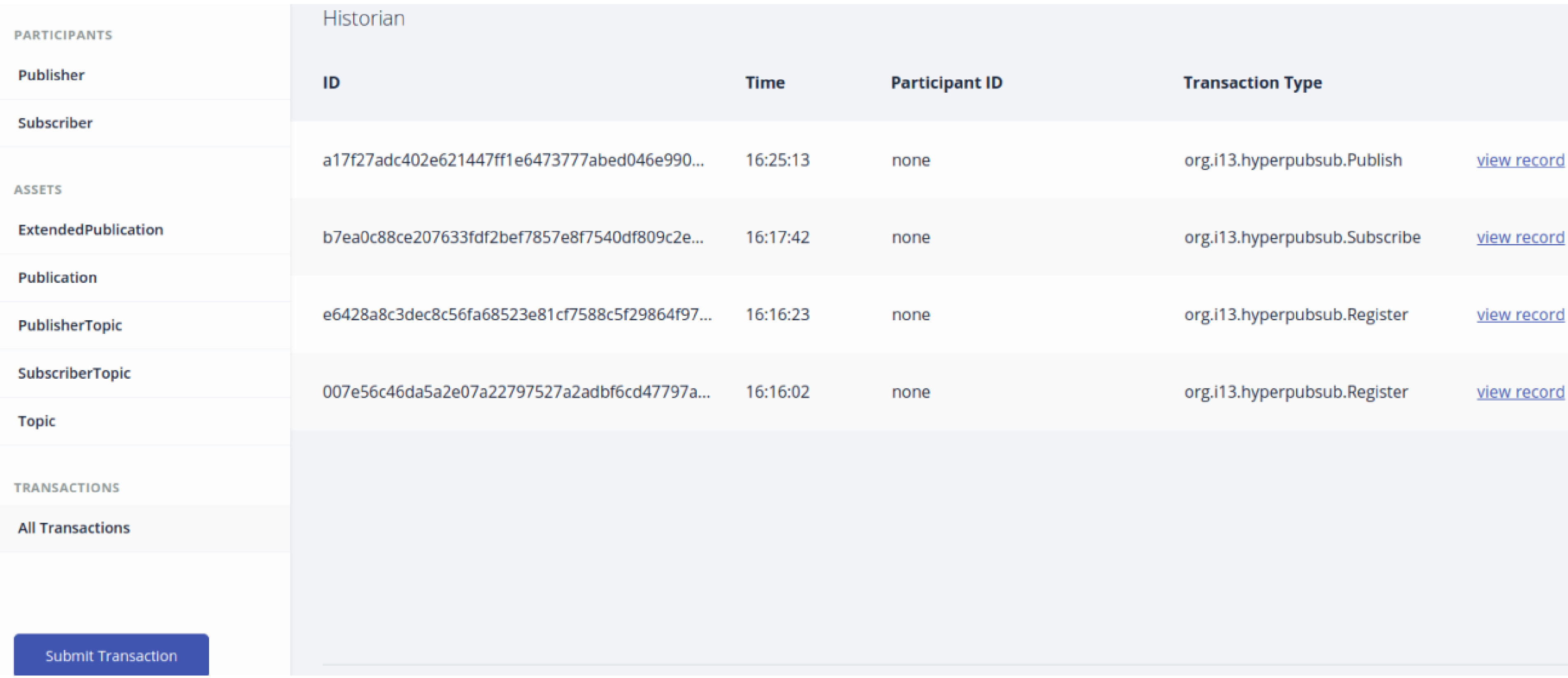
```
asset PublisherTopic identified by id {
  o String id
  --> Topic topic
  --> ExtendedPublication[] publications
}
```

Visible to the publisher, to audit publication history

```
asset SubscriberTopic identified by id {
  o String id
  o SubscriptionType status
  --> Topic topic
  --> Publication[] publications
}
```

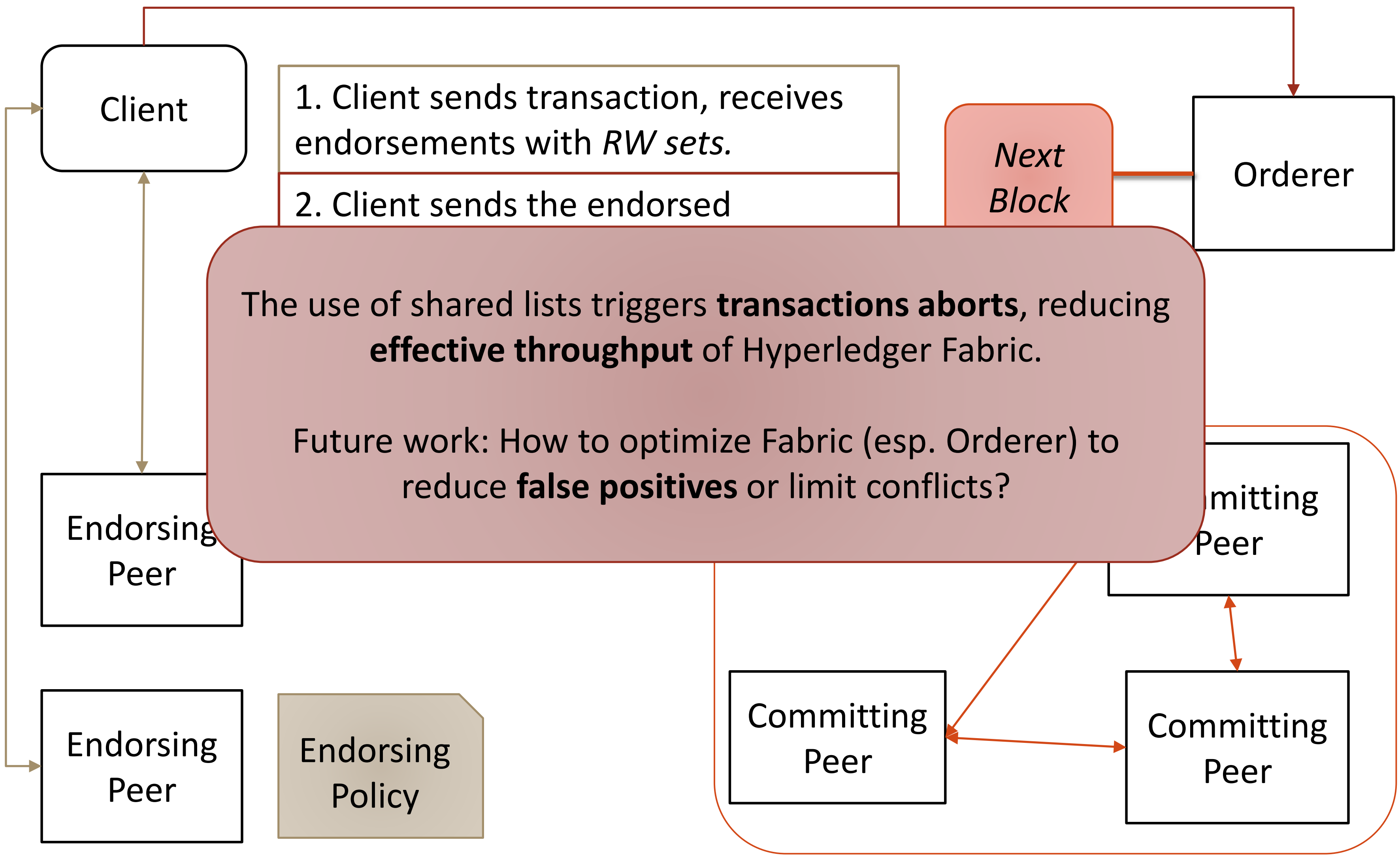
Visible to the subscriber, to audit publication history

Demo Interface (Playground)



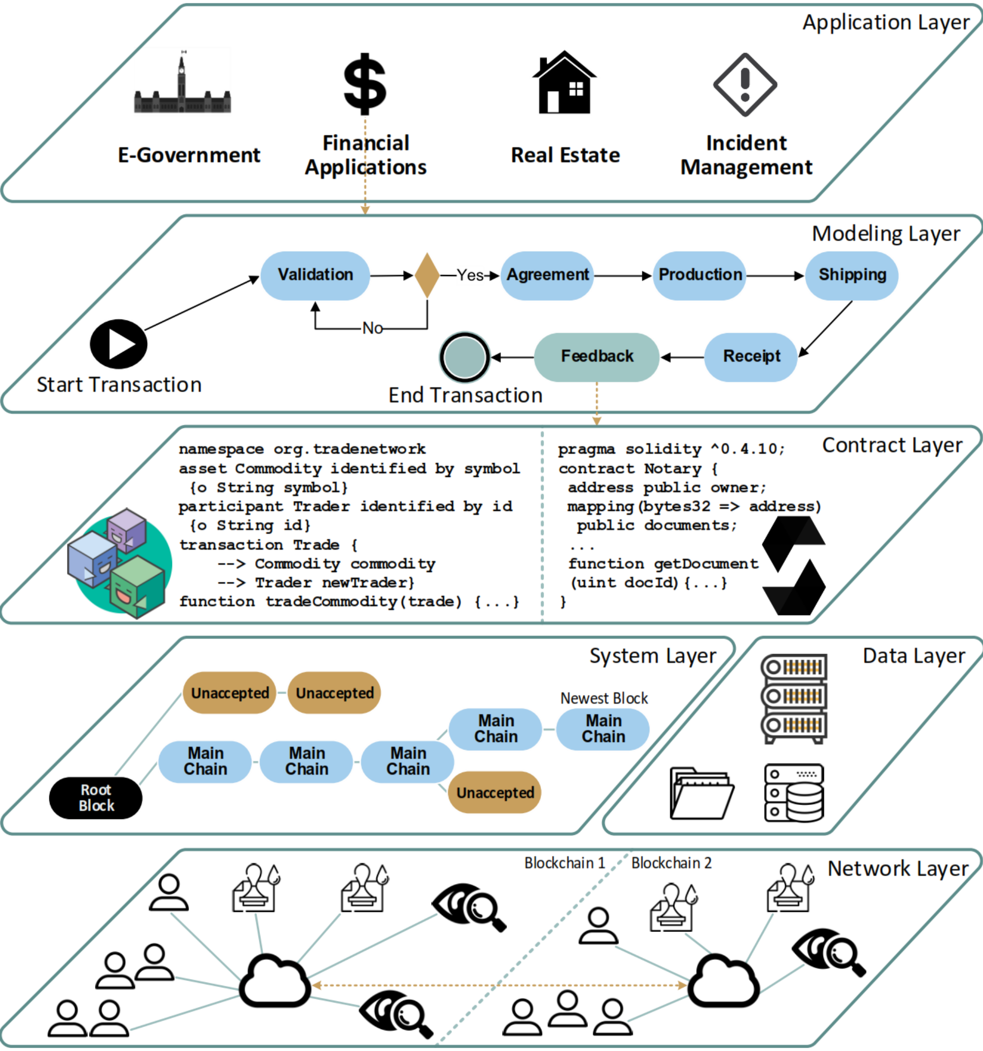
ID	Time	Participant ID	Transaction Type	
a17f27adc402e621447ff1e6473777abed046e990...	16:25:13	none	org.i13.hyperpubsub.Publish	view record
b7ea0c88ce207633fdf2bef7857e8f7540df809c2e...	16:17:42	none	org.i13.hyperpubsub.Subscribe	view record
e6428a8c3dec8c56fa68523e81cf7588c5f29864f97...	16:16:23	none	org.i13.hyperpubsub.Register	view record
007e56c46da5a2e07a22797527a2adbf6cd47797a...	16:16:02	none	org.i13.hyperpubsub.Register	view record

Multi-Version Concurrency Control



Link to our companion papers:
http://msrg.org/papers/bcbi-tr
http://heim.ifi.uio.no/~romanvi/debunking-bc-myths.pdf

Conclusions



- Blockchains provide *decentralized storage and code execution,* and can be used to combat fraud, avoid redundancy, and provide transparency.
- Blockchains rely on *cryptography* and massive replication using a robust consensus mechanism.
- Blockchains are useful for a wide variety of applications, ranging from cryptocurrency (*1.0*) to health (*3.0*).
- Research directions exist *across the six layers* for all kinds of applications (from 1.0 to 3.0), and involves different tradeoffs in the DCS spectrum: Decentralization, Consistency, Scalability.

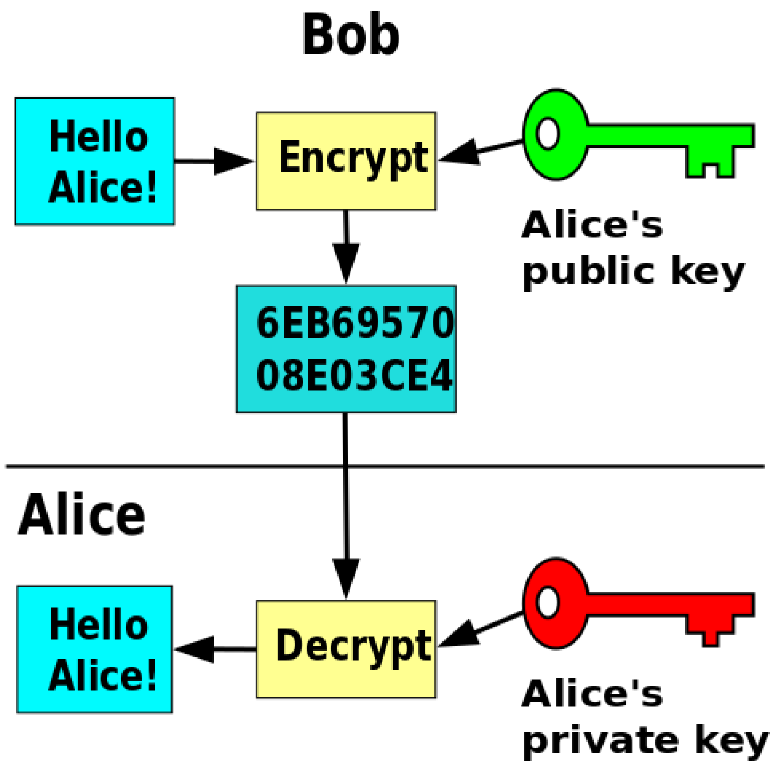
Bonus Material

APPENDIX

Public Key Cryptography

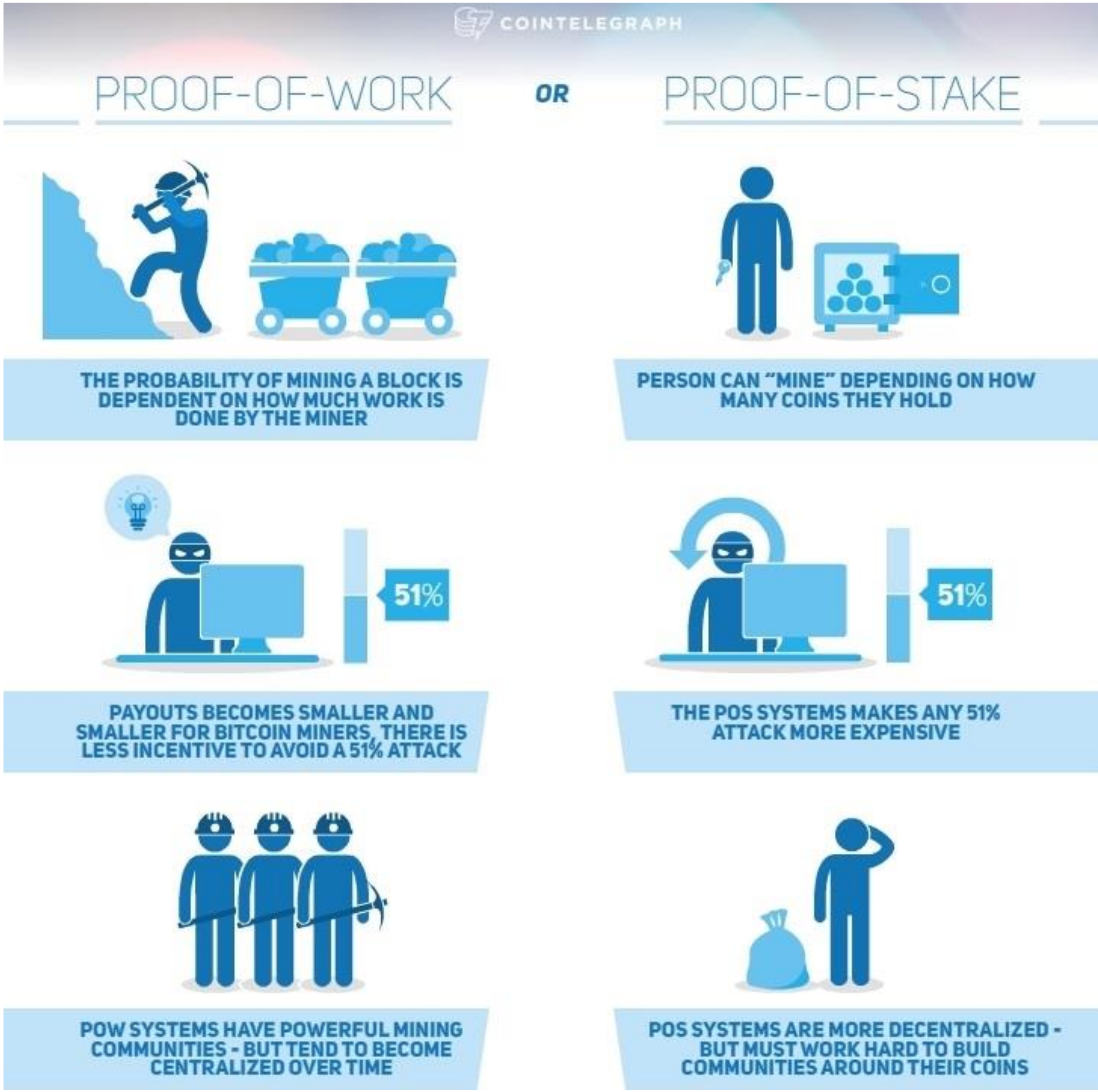
(Asymmetrical Cryptography)

- Recipient's public key is used to encrypt the plaintext to ciphertext
- Recipient's private key to decrypt the ciphertext to original plaintext
- No one can use the public key to decrypt the ciphertext to plaintext



Proof-of-Stake

- PeerCoin
- Nxt
- Ethereum (Future)
- “Nothing at stake” problem



Proof-of-Stake Details

verify() function in PoS:

- *sha256(PREVHASH + ADDRESS + TS) <= 2^256 * BALANCE / DIFFICULTY*
- ADDRESS of wallet of the miner, BALANCE is the recorded stake for the wallet
- TS is the timestamp in UNIX time (seconds)
- Thus, only one hash needed per second (per wallet)

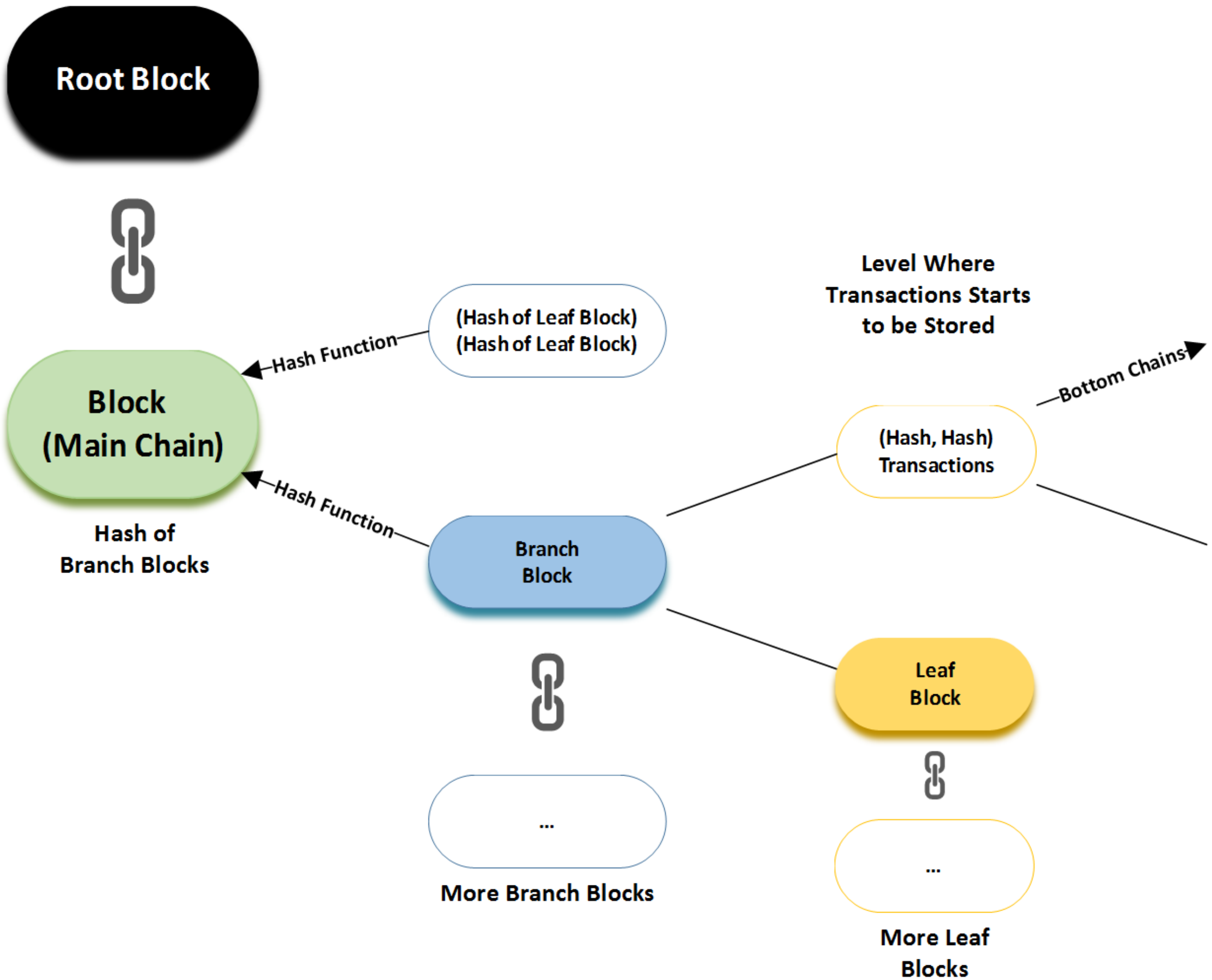
Branches can still exist in PoS:

- Due to propagation delays, multiple timestamps are valid for a block
- The puzzle function does not return an unique winner

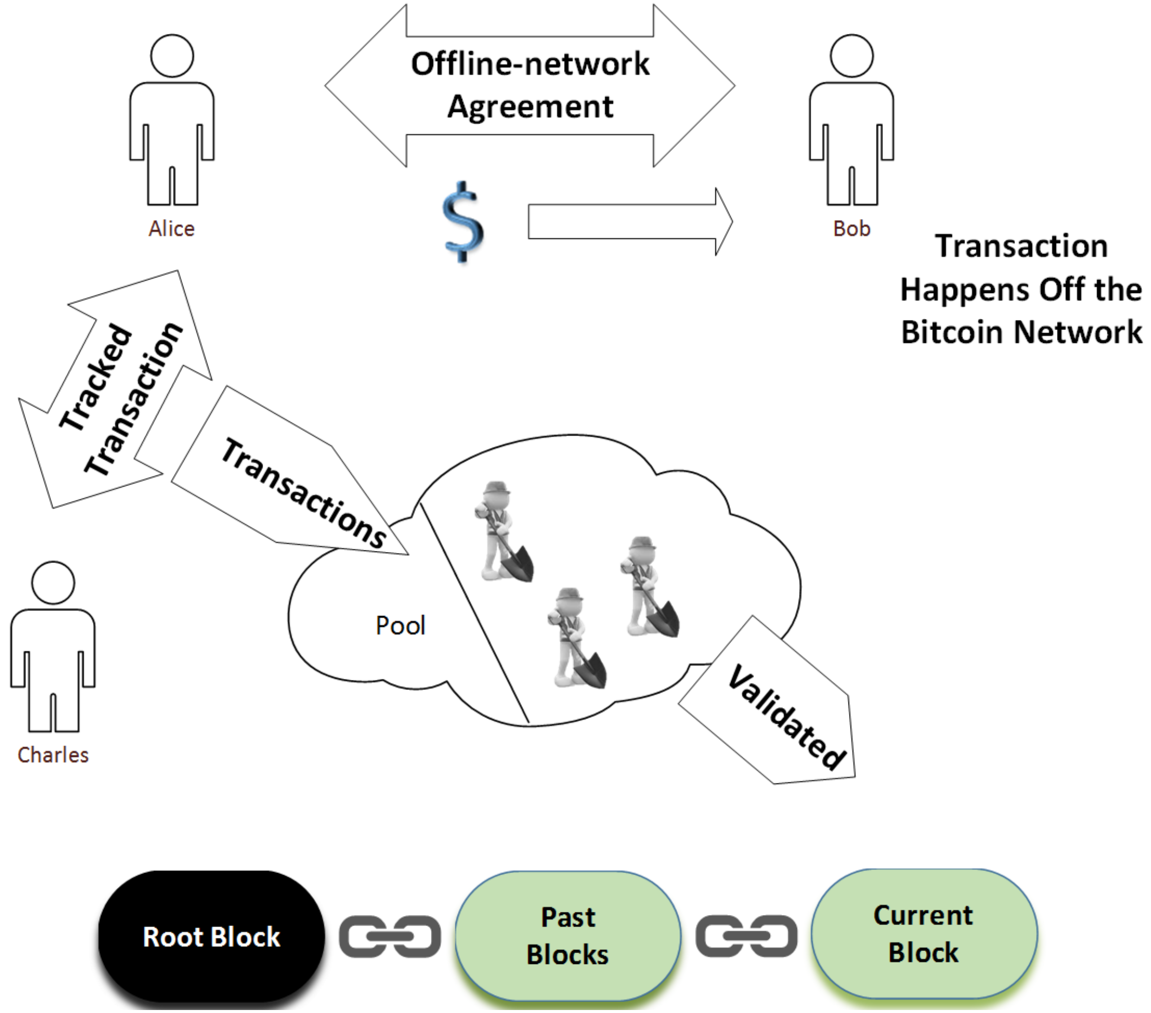
Nothing-at-Stake problem:

- PoW: cannot mine parallel branches since splitting resources is not effective
- PoS: mining parallel branches is easy since it only requires 1 hash/s
- Slasher algorithm: detection of parallel mining confiscates the stake

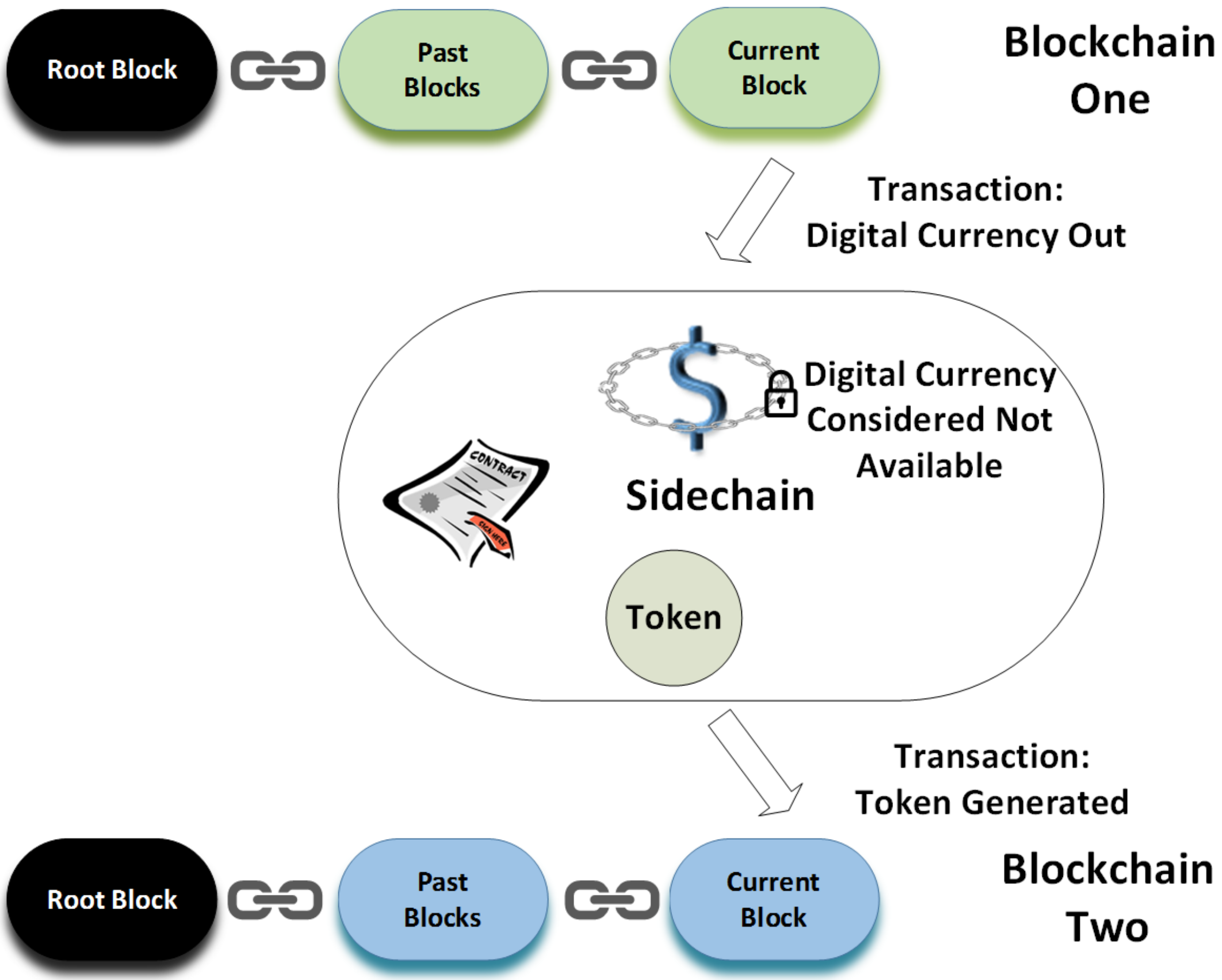
Scalability: Tree Chain - GHOST



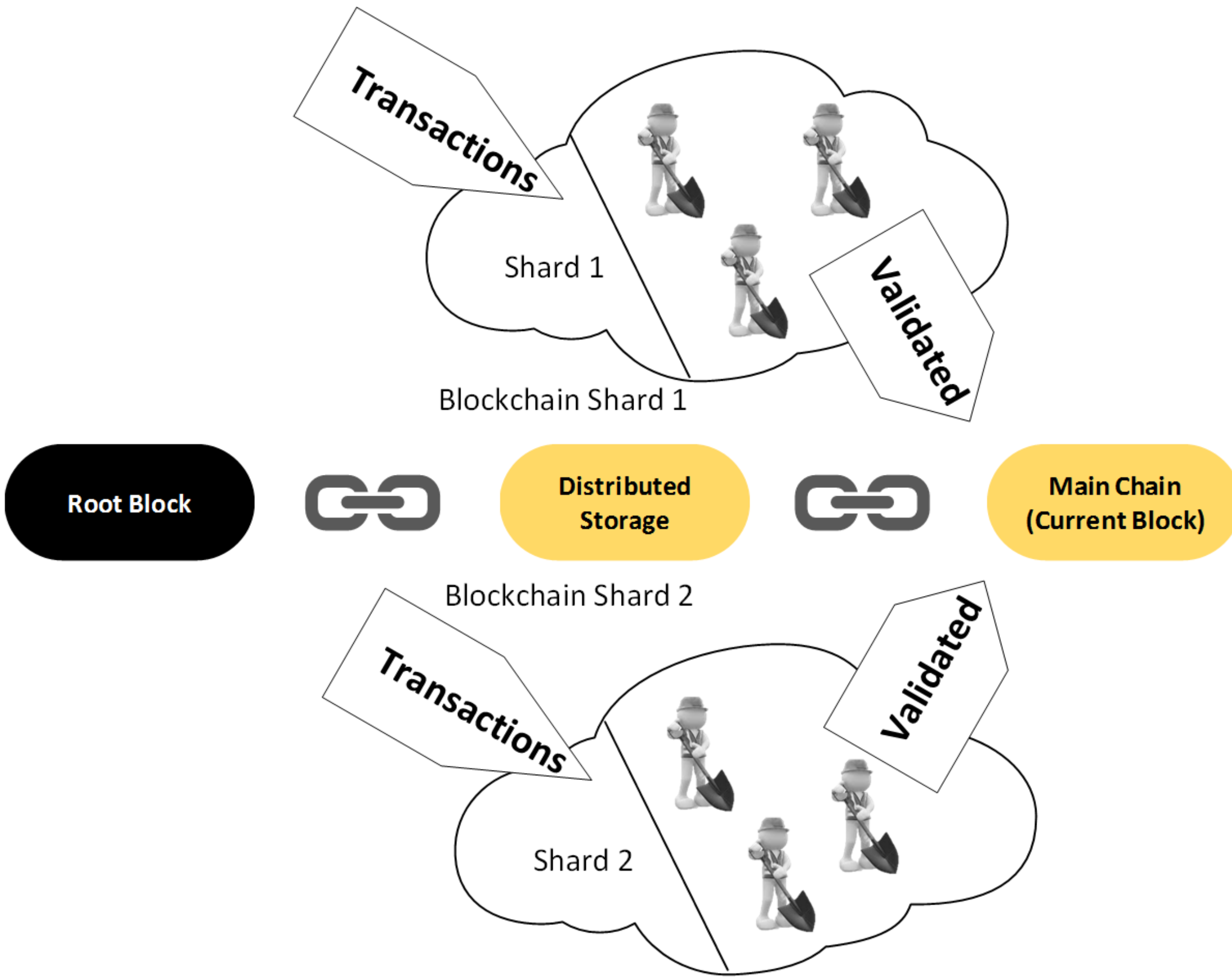
Scalability: Off-Chain



Scalability: Sidechain



Scalability: Sharding



Blockchain Platform

REFERENCE ARCHITECTURE

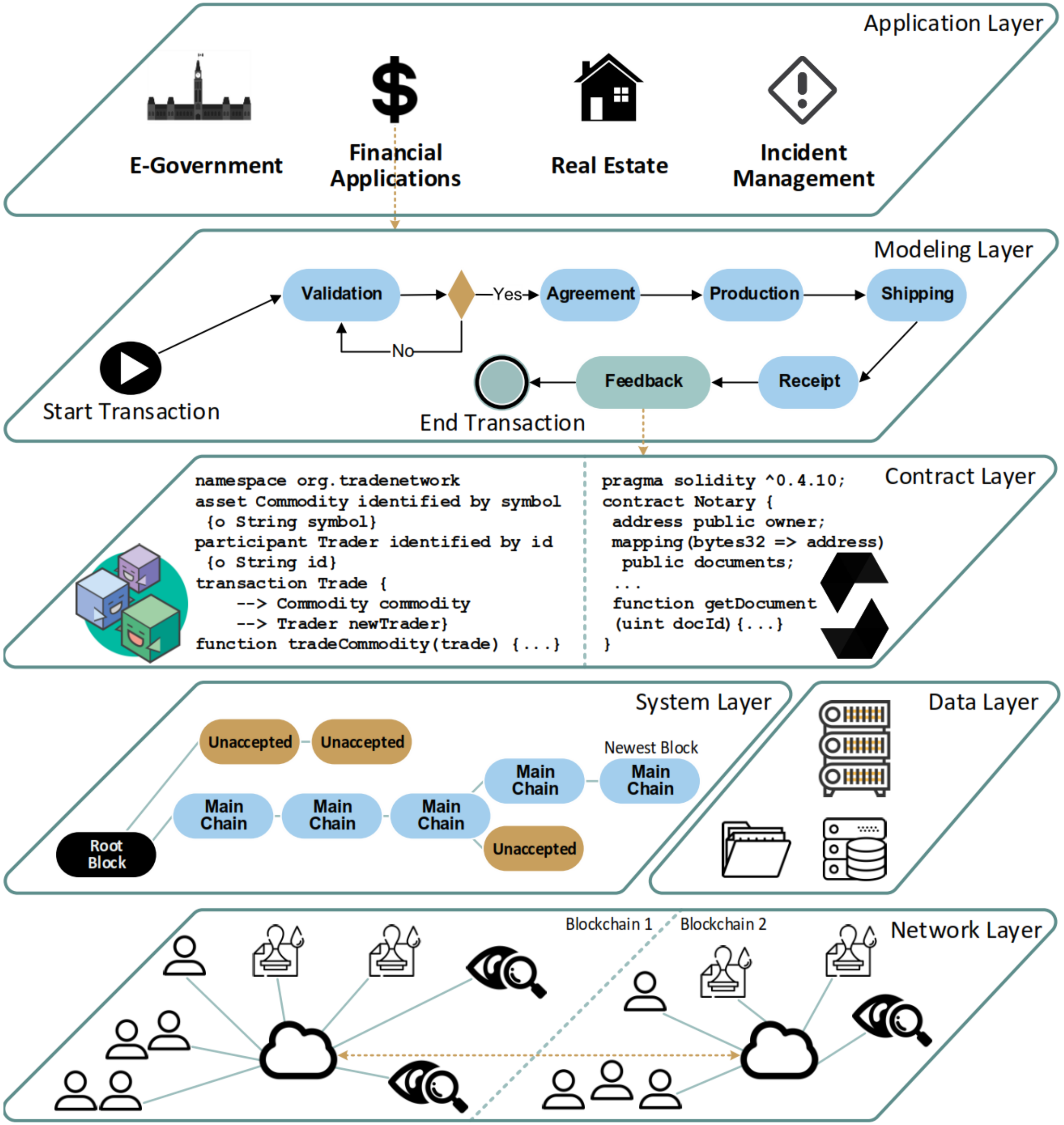
RESEARCH DIRECTIONS

Blockchain Reference Architecture

This vision diagram encompasses all aspects related to blockchain technologies.

Upper layers capture application semantics and their implementation.

Lower layers are concerned with technical system details.



ZHANG, VITENBERG, JACOBSEN, SADOGHI, TABATABAEI © 2018

Application Layer

Potential Research Directions

- Identify *application* and *service characteristics* that benefit from a Blockchain-based approach
  - cf. “Do you need a blockchain?” paper
- Develop a *methodology* to evaluate potential applications and select the appropriate *optimized* blockchain system:
  - Position applications as Blockchain 1.0, 2.0, or 3.0
- Create a *standard template* to describe and articulate use cases:
  - Describe actors, assets, transactions, queries, functional requirements, SLAs, etc.

Modelling Layer

Potential Research Directions

- Identify higher-level *modelling* and “programming” abstractions that are useable by business analysts, that are *verifiable*, that offer guarantees to end-users and map these abstractions into lower layers
  - BPMN, Petri-Nets, FSM, Business artifacts with lifecycles
- Identify *common services* and design blockchain *middleware* to support a variety of use cases
  - Identity management (authentication), reputation, risk analysis (spot checks), auditing, bidding, zero-knowledge proofs, document input etc.
- Extend modelling languages using blockchain semantics
  - FSM+: States can be described as *“on-chain”* or *“off-chain”*
  - Use of *Controlled English* which is portable to smart contracts

Smart Contract/Programming Layer

Potential Research Directions

- Design *mappings* for standard modelling languages (e.g., BPMN) into *smart contracts*
  - Create execution engines on blockchains for BPM
- Formally *verify* smart contracts for correctness
  - Use of formal verification tools (e.g., Why3, F*)
- Investigate the use of *domain-specific languages* for smart contracts
  - E.g., to circumvent the halting problem
- *Scalable execution* and storage of smart contracts
  - E.g., Sharding in Plasma, zk-SNARKS in Zcash

System Layer

Potential Research Directions

- Evaluate existing *consensus algorithms* and design new ones specifically tailored to application characteristics with varying tradeoff:
  - *Proof-of-Stake*, *Practical Byzantine Fault-Tolerance* (PBFT), …
- Develop mechanisms to increase the scalability of blockchains:
  - *Off-chain, side-chains, tree-chains (GHOST), sharding*
- Use of innovative hardware for achieving consensus
  - *Proof-of-Elapsed-Time* using *Intel SGX* (Hyperledger Sawtooth)
- Develop *quantum-resistant* mechanisms for securing Blockchain computations

Data Layer

Potential Research Directions

- Develop effective *data management abstractions* to enable efficient Blockchain computations and verification
  - AVL+ Trees, Merkle Patricia Trees
- Develop *compression techniques* to reduce the size of historical data and scale with the number of users
  - Ethereum Fast Sync
- Provide *off-chain storage (chain state)* which is securely and privately verifiable by the on-chain data, executable by the smart contracts
- Maintain *availability* of smart contracts and assets in the presence of space saving techniques

Network Layer

Potential Research Directions

- Develop effective *networking abstractions* to support *scalable* and *low-latency* blockchain operations
- Investigate effects of *networking characteristics* on Blockchain computations (e.g. network partitions)
- Integrate with *Software-Defined Networking* (SDN) and other technologies
- Tolerate *unreliability* in hardware components (IoT, Edge Computing)
- Support *cross-platform* communication (e.g. private & public networks)