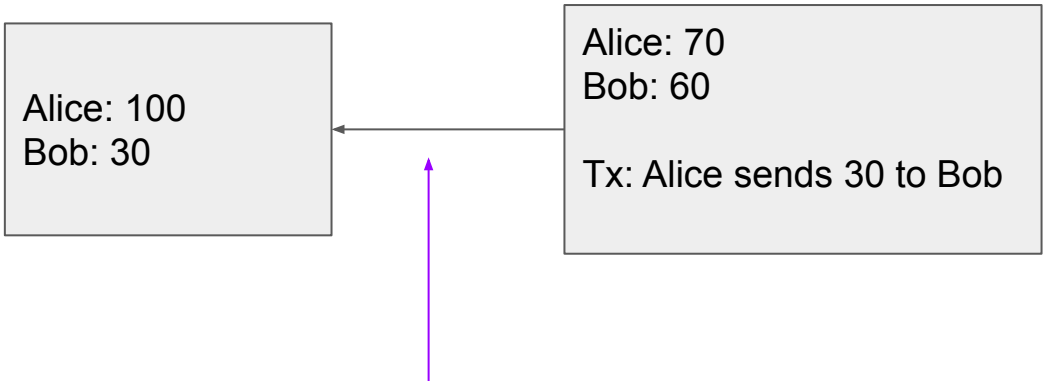
Inductive Consensus Tree Protocol (ICTP)

A scalable blockchain algorithm

Presentation by Jessica Taylor
Algorithm by Jessica Taylor and Jack Gallagher
Website & paper: ictp.io

Basic blockchain



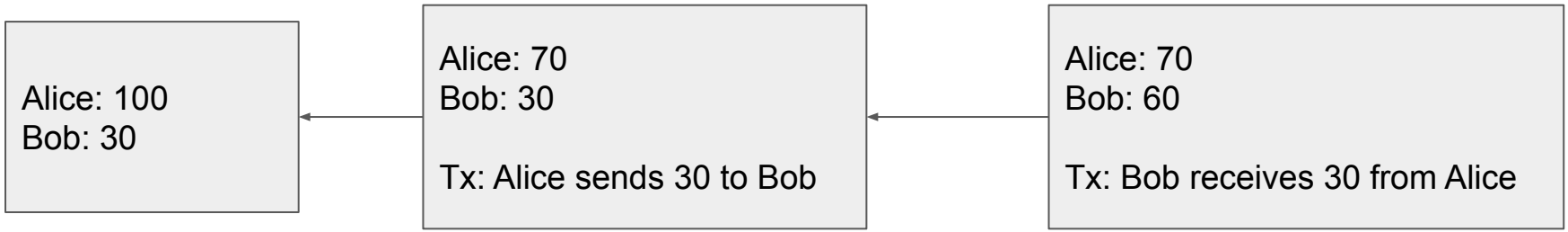
(arrow indicates inclusion of hash code of first block)

- New accounts can be created by being sent money
- Transactions signed by sender
- Multiple transactions per block
- POW or POS

Scaling of basic blockchain

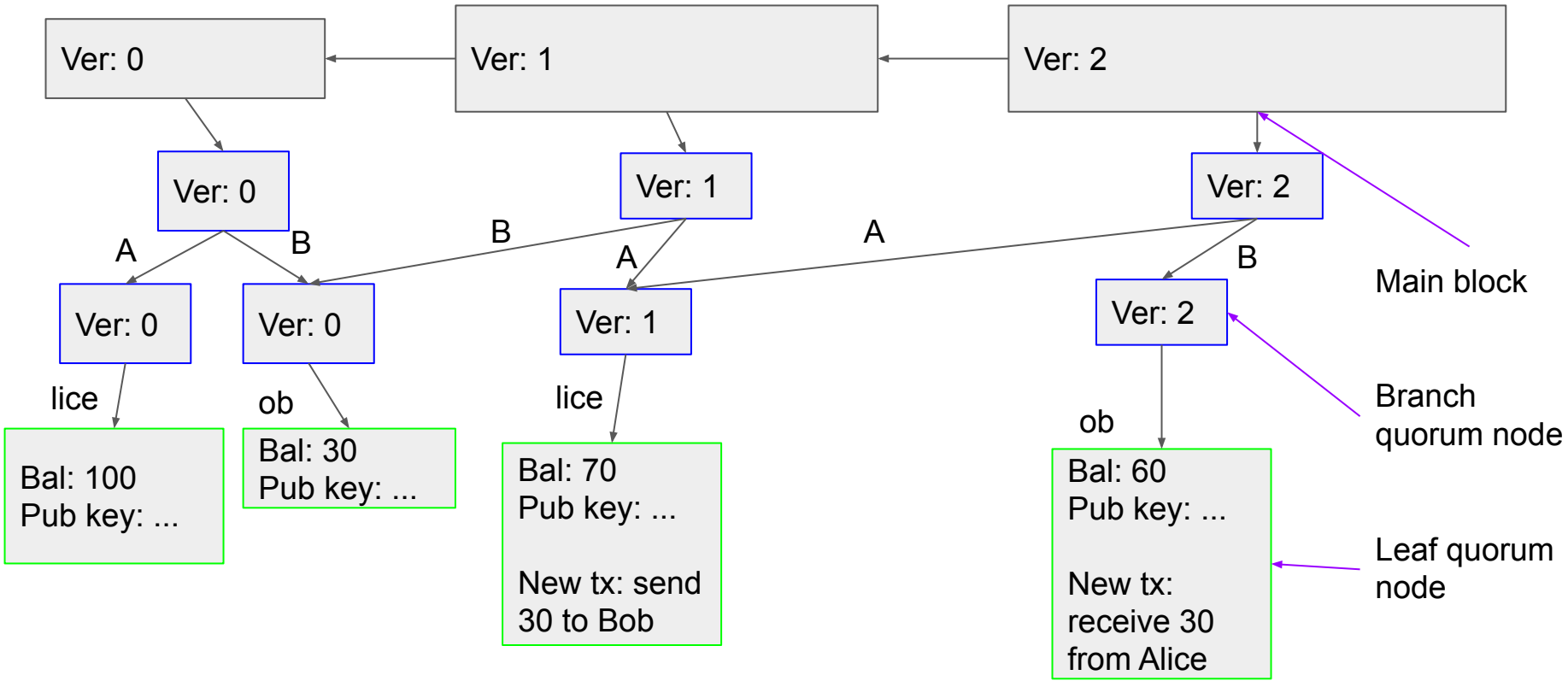
- Balances for non-genesis blocks don’t have to be stored, they can be derived from the transaction history.
- Every transaction has to be processed by every account.
- Let m = number of accounts, n = number of transactions. Total work is ~O(mn).
- Each account needs to store O(n) data.

Separating sends and receives



- This means each transaction only modifies a single account
- Similar to actor model (Erlang): message-passing between state-carrying actors, with time delay between send and receive
- Will make data pipelining easier when expanding to a tree

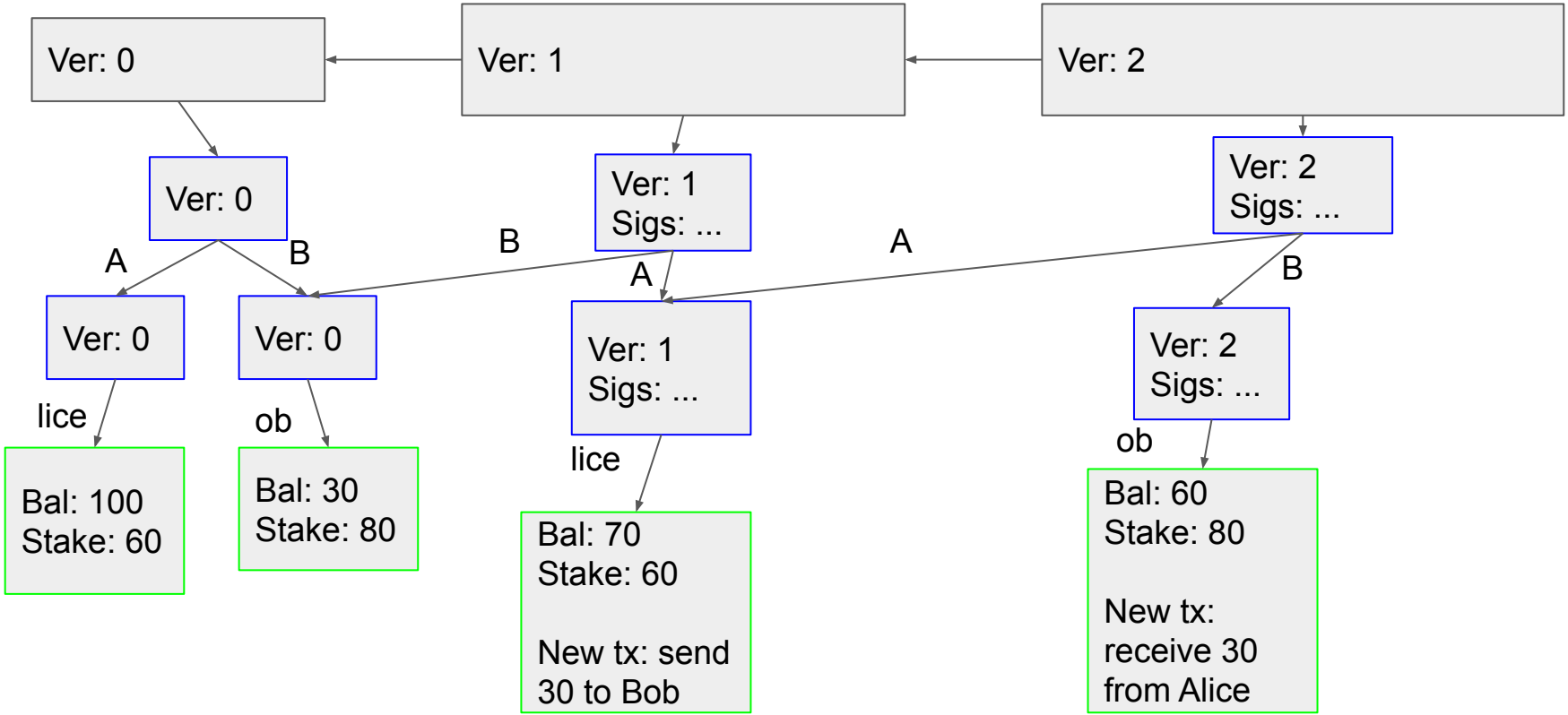
Balances and transactions stored in a Merkle Patricia Tree



Explanation

- Persistent data structure, at most O(log m) new nodes created per transaction
- Accounts are actually indexed by hexadecimal strings representing account public key hash codes, not English names (Alice/Bob)
- Transactions can be verified by checking against the previous Merkle-Patricia tree
- Store a set of received transaction hash codes, so the same send won’t be received twice
- For scalability to many fields, the account state can be stored as a Merkle Patricia tree
- Data can be stored in a DHT, but everyone needs to check it
- This requires O(m n log(m)) work for m accounts & n transactions, not an improvement!

Pool members sign valid new branch quorum nodes



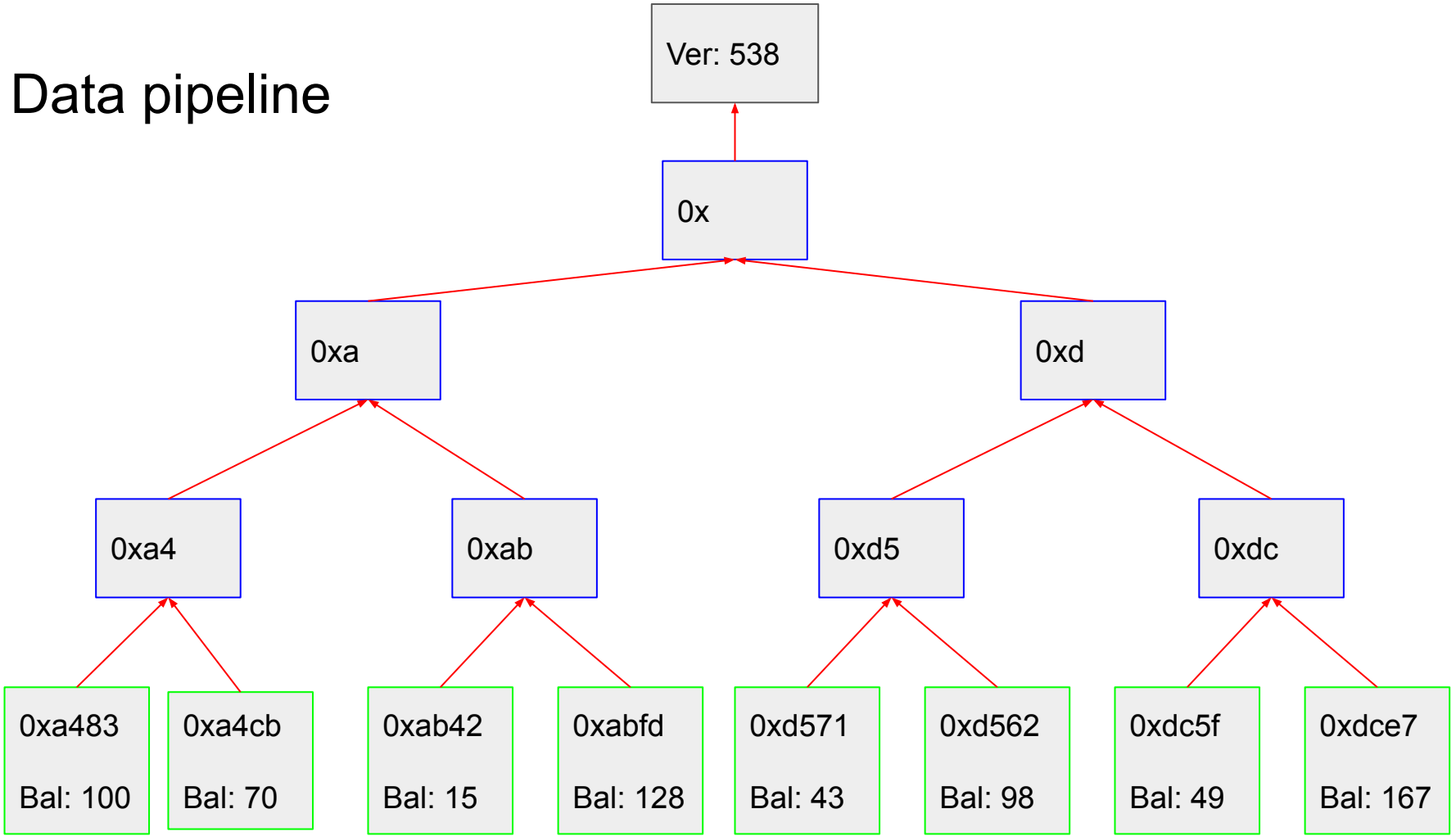
Pools and quorums

- Pool: a randomly-selected set of accounts
- Accounts with more stake are more likely to be selected
- A pool is chosen for each quorum node
- A branch quorum node *may* contain signatures by a threshold number of its corresponding pool; “quorum” refers to the threshold set of pool members
- Main idea: cooperative pool members only sign valid nodes. An attacker would need to own a substantial fraction of stake to attain a quorum in *any* quorum node with non-negligible probability. But if they have stake, they have a financial interest in the system continuing to work.
- As a result, checking whether there’s a quorum is *sufficient* for checking whether the node (and its descendents) are valid, almost certainly
- Pool members check if each descendent node has a quorum; if it does, then it’s considered validated; otherwise, they check the node recursively

Scalability

- Let p be the pool size (e.g. 1000).
- A new transaction will be checked by O(p) parties, forming a signed quorum node. (This node may amalgamate multiple transactions, call this number q)
- The signed quorum node has to be checked by O(p) parties, forming a parent node of this node.
- Checking the parents of the parents can be ignored, it’s a constant factor (exponential decrease).
- Total work is O(n(p + p^2/q)) = O(np(1 + p/q))
- p scales logarithmically with n, so this is O(n polylog(n))
- The constants matter (worked out in the paper), tx cost ~= $0.0002

Data pipeline



Data pipeline explanation

- Data is processed in parallel
- Nodes are processed, possibly signed, and sent to parent pools
- Parent pool members check child nodes, create a parent node, and sign the node if it’s valid (and contains enough total data for checking the signatures to be cheaper than checking the data)
- Top branch quorum node (0x) is sent to signers/miners for inclusion in next main block (proof of stake)
- Total time taken is linear in the depth of the tree, O(log m), approximately minutes per block
- Limit on number of transactions is based on the number of pool members and their computation/communication speed; if there are enough pool members, *everyone* can do a transaction in a single block!

Details

- Data stored in IPFS-like DHT, indexed by hash code
  - New quorum nodes are stored by members of the corresponding pool, ensuring data availability
  - Highly-accessed nodes must have their data replicated
  - Total storage is O(n) not O(mn) as in a standard blockchain
- Incentives
  - Transaction fees are collected and distributed to pool members, miners, signers, data storers
  - Punishments for pool members who sign invalid nodes, and normal PoS punishments
- Pseudorandom seed is randomized every so often using threshold cryptography (details in paper)
- Pools are occasionally reshuffled, to reflect changes in account stake (stake is escrowed for the duration of the period between pool shuffles)

Extra features

- Smart contracts
  - Account nodes may be represented by a contract, rather than a public key
  - Contracts send/receive messages as in the actor model, and can store persistent data
  - Receives are handled similar to in Ethereum, there are methods for receiving different messages
  - Accounts may have extra “contract data” attached
  - Contracts are encoded in WebAssembly, a VM that many languages can be compiled to
- Privacy
  - Separate account public state and private state
  - Use zk-SNARKs to prove validity of transformations to account states, as in Zcash
  - See paper for details
  - Note: this makes implementation a lot harder; will not be included in the first version

Development roadmap

- Start with auditable centralized system with no contracts or privacy
- 3 possible improvements
  - Make it distributed
  - Add contracts
  - Add privacy (this is the hardest)

Creation of new cryptocurrency

- Current name is “Mercatoria”
- Idea: implement a new cryptocurrency using this algorithm
- Existing currencies, e.g. Bitcoin and Ethereum, must implement scaling features to compete with Mercatoria
- So, the value of Mercatoria is based on the expectation that existing currencies *won’t* implement these features for a while
- Ideally, scaling features are eventually implemented in existing blockchains, and Mercatoria is usable as a scalable currency in the meantime
- Alternatively, if Mercatoria “wins”, existing currencies may be ported to Mercatoria as smart contracts (given Mercatoria’s generality and scalability)

Conclusion

- This is MUCH more scalable than any alternative proposal, e.g. Ethereum's current sharding proposal
- It easily scales to billions of transactions per day (multiple per person), in contrast to current cryptocurrencies which choke on low millions per day with huge transaction fees
- It can support micropayments, decentralized Internet, auditable government
- See our paper for technical details (ictp.io)