Recap: Blockchain scalability

• Two types of scaling problems
  - Transaction throughput (txs/sec)
  - Blockchain size (state storage required to validate txs)

• Last lecture:
  - Off-chain transactions (“channels”)

• This lecture:
  - Sharding (distributing the verification work)
  - State commitments (load balancing storage)
Recap: Payment channels

• Bitcoin processes 3 txs / second. This would never be able to replace Visa for payments!

• Off-chain transactions (via channels) could make this a realistic possibility...

• In channels, transactions only hit the blockchain twice:
  - Once to open the channel and deposit collateral
  - Once to close the channel and net settle
• Lightning is an overlay network of channels
• Payment from Alice to Bob can be routed over any multi-hop path of channels with sufficient collateral

Send 3 over red path
Send 2 over blue path
State channels

• Much simpler to build a payment channel with stateful smart contract (e.g. in Ethereum)

  - Init
  - CLOSE (A:5, B:5), 1
  - Pending
  - +1 day
  - Closed
  -CLOSE* (A:3, B:8), 1

  - Pays 3 to A and 8 to B

• But stateful contracts require more blockchain storage and more complex verification
Channels lower user costs

- Micropayments (tx fees too high)
- High gas costs for Ethereum smart contracts
Today’s topic: load balancing

• Sharding (load balancing transaction verification)
• State-commitments and authenticated data-structures (load balancing state storage)
Sharding Strawman #1

• Split blockchain into N independent blockchains, call each a “blockchain shard”
  - Shards have independent states (e.g. different types of coins, independent smart contracts)
  - Shards have different sets of validators
  - Shards run consensus independently

• Problems:
  - Dividing validator set isn’t great for consensus security
  - Only handles state that can be truly partitioned
Sharding Strawman #2

- Attacker of Strawman #1 needs to compromise 1/N fewer validators to cause forks on one of the blockchains.
- Idea: keep a root blockchain with all consensus validators that resolves forks, but doesn’t verify transaction validity.

Root ledger

Txs contains “heads” of the N shard chains
Sharding Strawman #2

- Problem: adversary compromises validators on one shard, approves invalid transactions. **Can we leverage root ledger to resolve?**
- Solution: state commitments and fraud proofs

Root ledger

Txs contains “heads” of the N shard chains
State commitment

• Bitcoin state is the UTXO set (the set of currently spendable coins) --- i.e. valid records of ownership

• Verifying a transaction consists of:
  - Checking inputs are valid, i.e. in the current UTXO set
  - Checking local information (running script on transaction data), e.g. sum inputs > sum outputs

• Idea: Blockchain stores Merkle tree root of UTXO set
  - Root commits to the state, updatable
  - Validity of coin/txo proved with Merkle path
Plasma (simplified)

- Root blockchain logs state commitments for N shards, and commitment to txs in each update
- Shard state commitment could be a UTXO set (Merkle tree), or more complex (Sparse Merkle tree, Patricia tree)
- If shard x sends state update that included bogus tx, then users may challenge the update:
  - Whistleblower posts txid, claims fraud
  - Validator(s) of shard x produce tx payload, validity proofs (e.g. Merkle) for the tx and all its inputs
  - Whistleblower may perhaps collect collateral reward
Plasma (simplified)

More complex state transition proofs possible!

Fraud (defensive) proof gives Merkle proofs to show challenged \( tx \in Txs_{i+1} \) and all inputs are in \( UTXOSet_i \), all outputs in \( UTXOSet_{i+1} \).

Validators of shard x lose posted collateral for failed fraud proof.
Cross-shard transaction

• Can we pay from account on shard x to account on shard y?
• With state commitments, validators on shard y can verify existence of transaction processed on shard x given validity proofs
• **Receipt tx:**
  - Tx1 removes z coins from account A on x, creates special receipt output *rct1*
  - Tx2 adds z coins to account B on y, pre-conditioned on validity proof for *rct1*
Random committee shards

- Rather than fraud proofs, can we guarantee that w.h.p. enough honest validators are in charge of each shard?

- **Recall beacons/VRF sortition:** Elect random committees of validators (e.g. from set of weighted public-keys) to govern each shard

- Can re-elect committees periodically
  - Requires reshuffling shard data storage
“Stateless” validator

- Validator only stores sequence of state commitments
- Attached to each tx are validity proofs (e.g., Merkle proofs) for all inputs
- Validators update state commitments after each new block of txs (how? Think how this could be done with Merkle trees...)
- With Merkle trees a bit impractical...attaching proofs to every tx makes this large (would increase Bitcoin tx size from 250bytes to >1KB)
- Merkle tree replacements based on classic RSA accumulators can batch/aggregate proofs (new result from last year!)
“Stateless” validator

“Stateless” validators enable:

- Validator separation from data/storage providers (availability providers)
- Frequent random shuffling of validators among shards without moving data
- In extreme case, users store their own UTXOs/account data (nobody stores the whole blockchain)

Utreexo – by T. Dryja
Batching Techniques for Accumulators w/ Applications to Stateless Blockchains by D. Boneh, B. Bünz, B. Fisch
https://ethresear.ch/t/the-stateless-client-concept/172 – V. Buterin
Summary

• **Payment/state channels** drastically reduce txs that need to hit the blockchain

• **Sharding methods** load balance both validation of txs and state storage

• **State commitments** are a key sharding tool:
  - Enable flexible roles (nodes contributing to data availability, consensus, validation)
  - Extreme case: data spread over users