**Whitepaper** Foundation

# Ethereum
# Beacon Chain Overview

2018. 7. 19

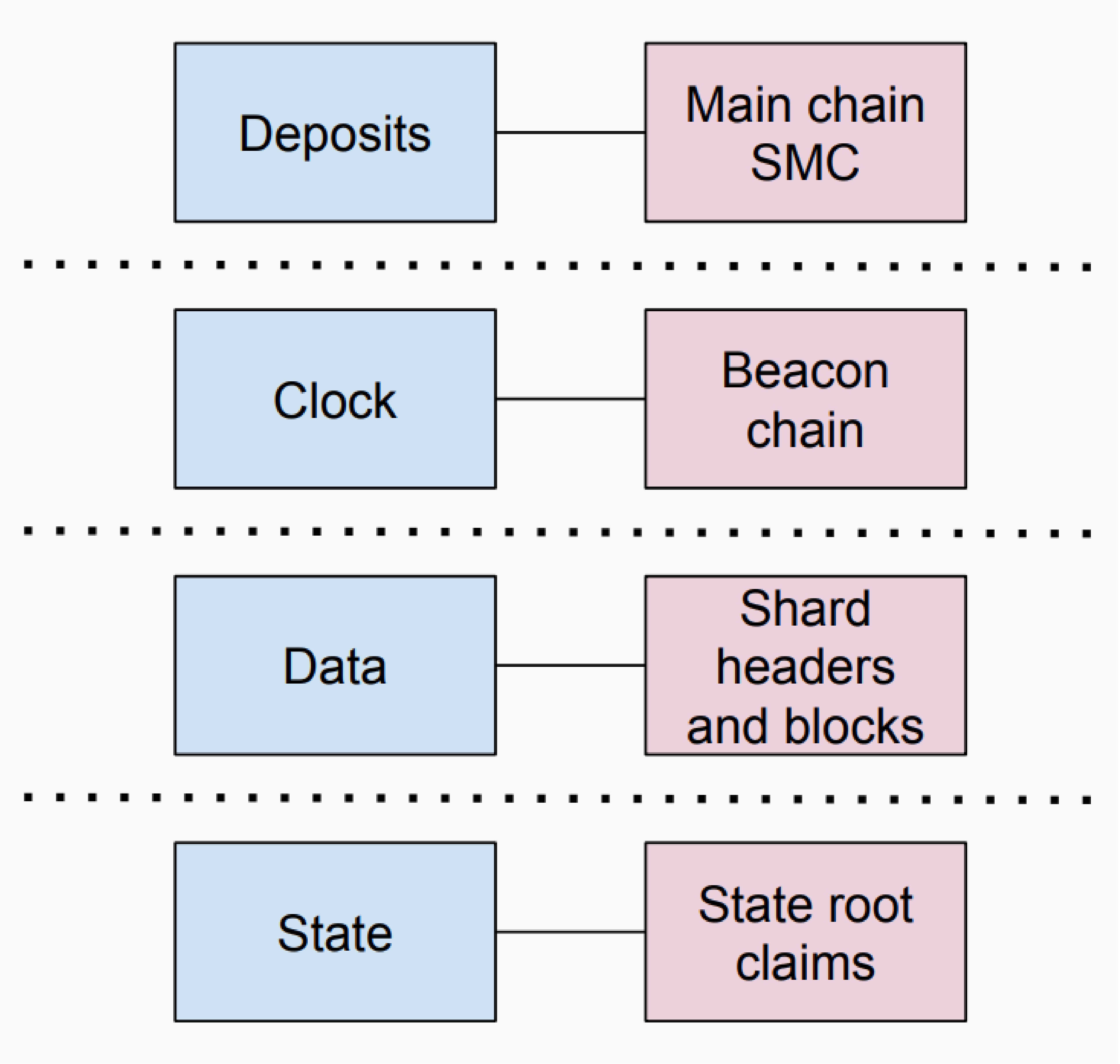
**Geore Han**
<kr8534@gmail.com>

# Outline

- Relations Between Validators (Recap)
- Cross-shard Communication
- Epoch Transition

# Beacon Chain & Validators

References: Edcon (2018.5.)

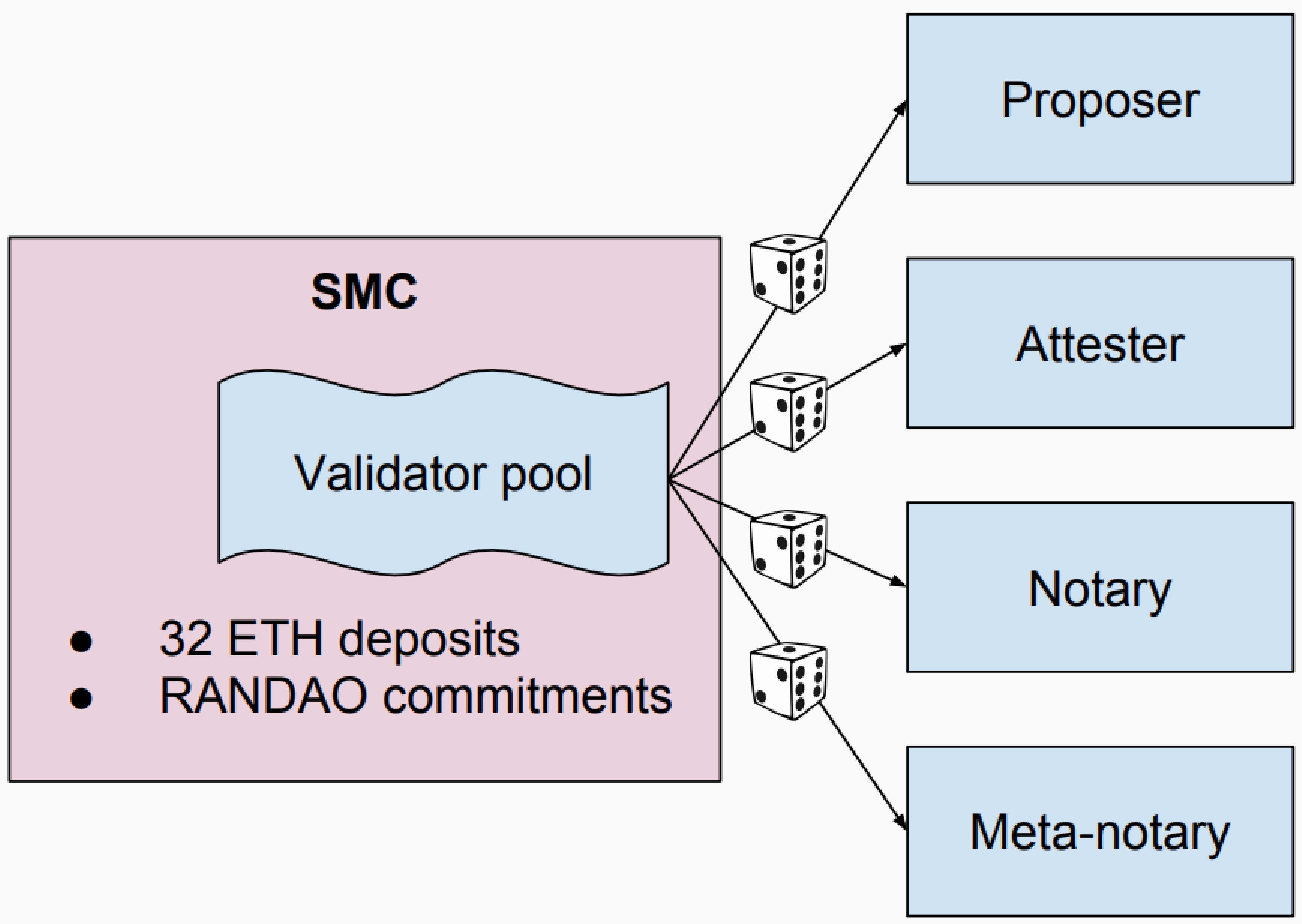


Deposits — Main chain SMC

Clock — Beacon chain

Data — Shard headers and blocks

State — State root claims



SMC

Validator pool

- 32 ETH deposits
- RANDAO commitments

Proposer

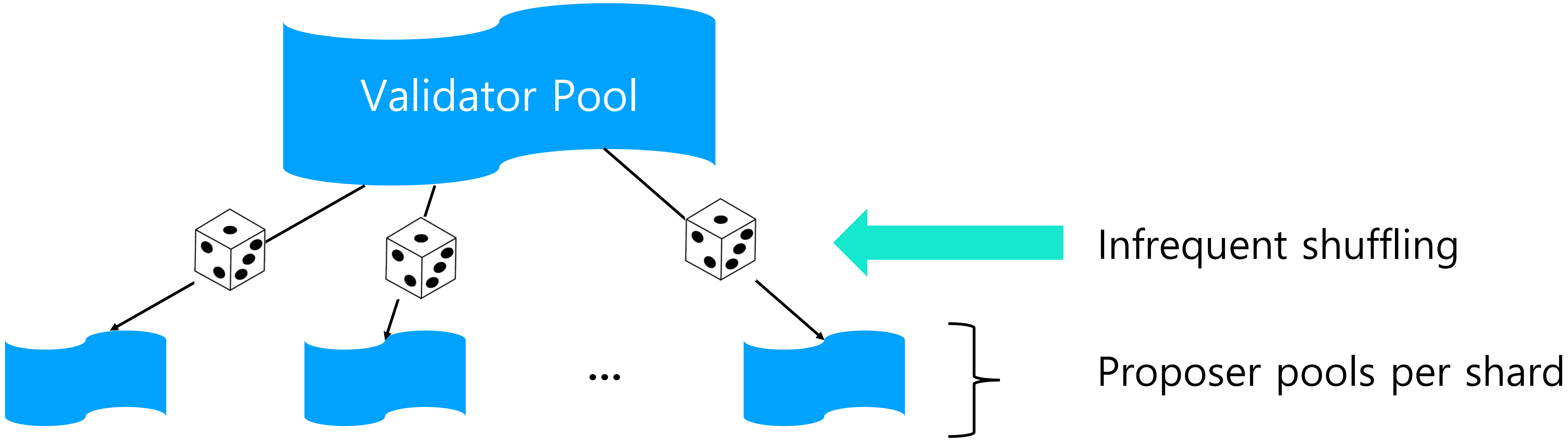
Attester

Notary

Meta-notary

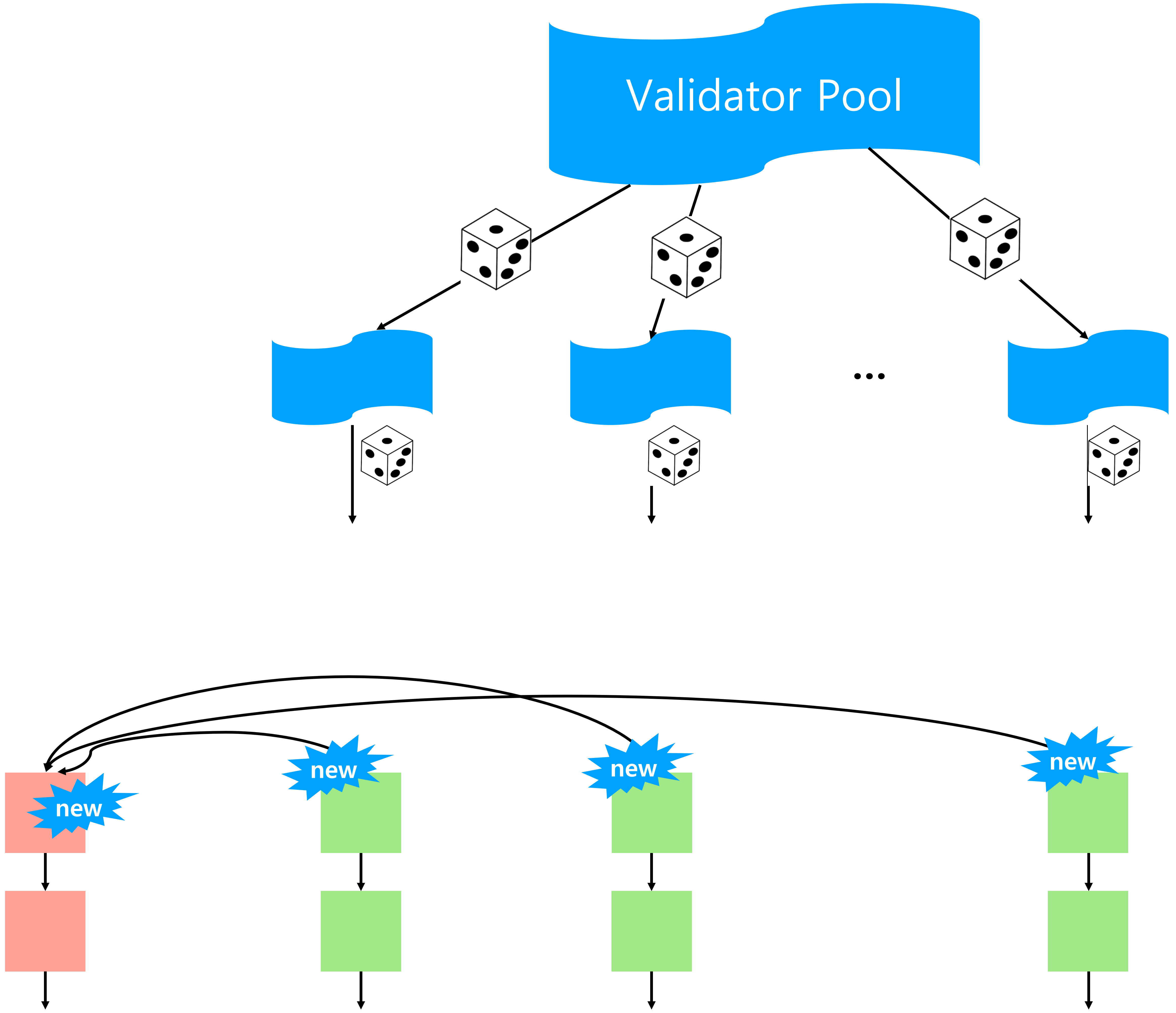
# Relation Between Validators

References: Edcon (2018.5.)



# Relation Between Validators

References: Edcon (2018.5.)



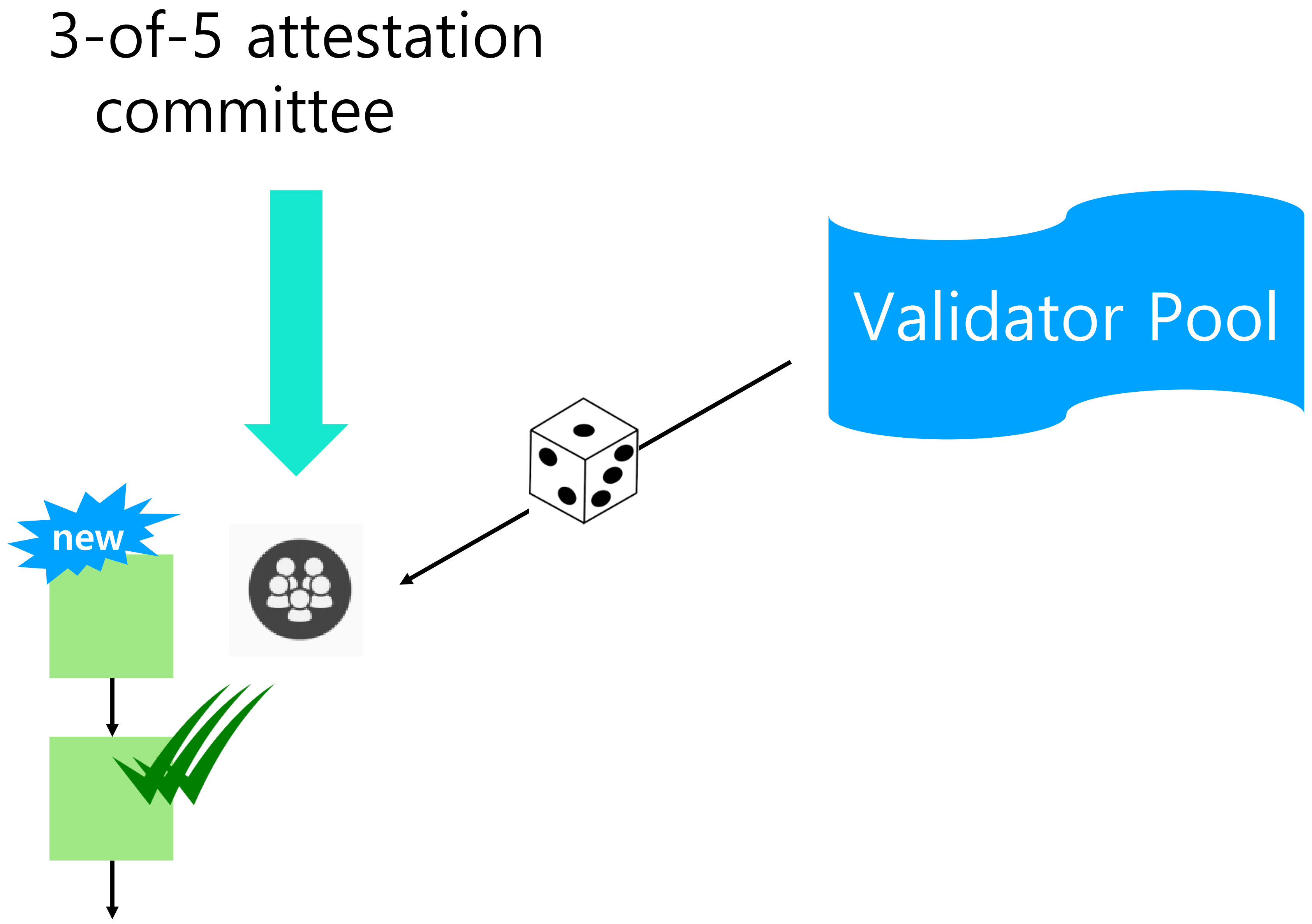
Sample proposers
per heartbeat

# Relation Between Validators - Attestation

References: Edcon (2018.5.)

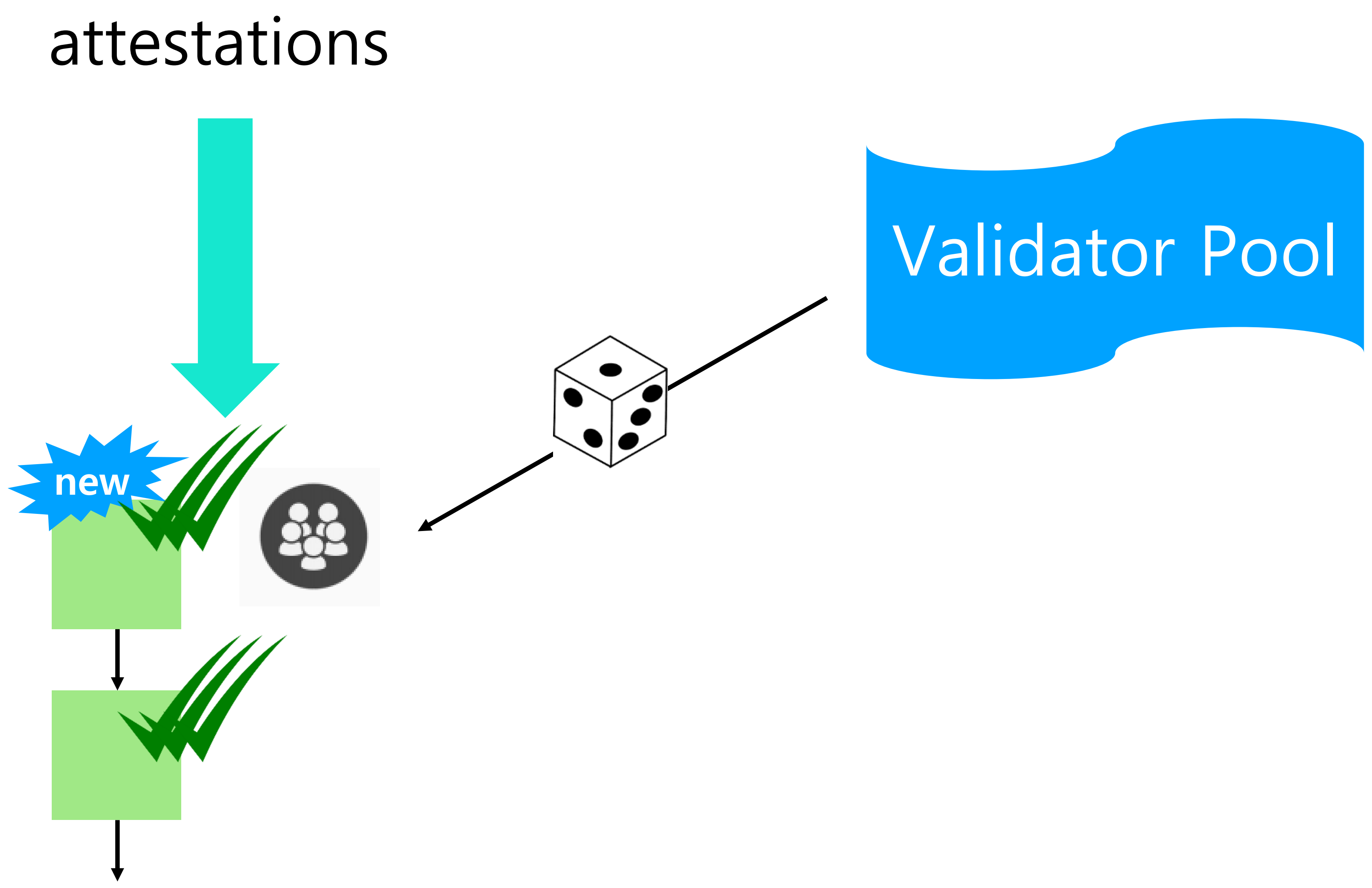
- Their sigs can be aggregated via BLS aggregation, STARK aggregation, etc

- Security model: honest majority ➔ can be secure against a semi-adaptive or adaptive adversary



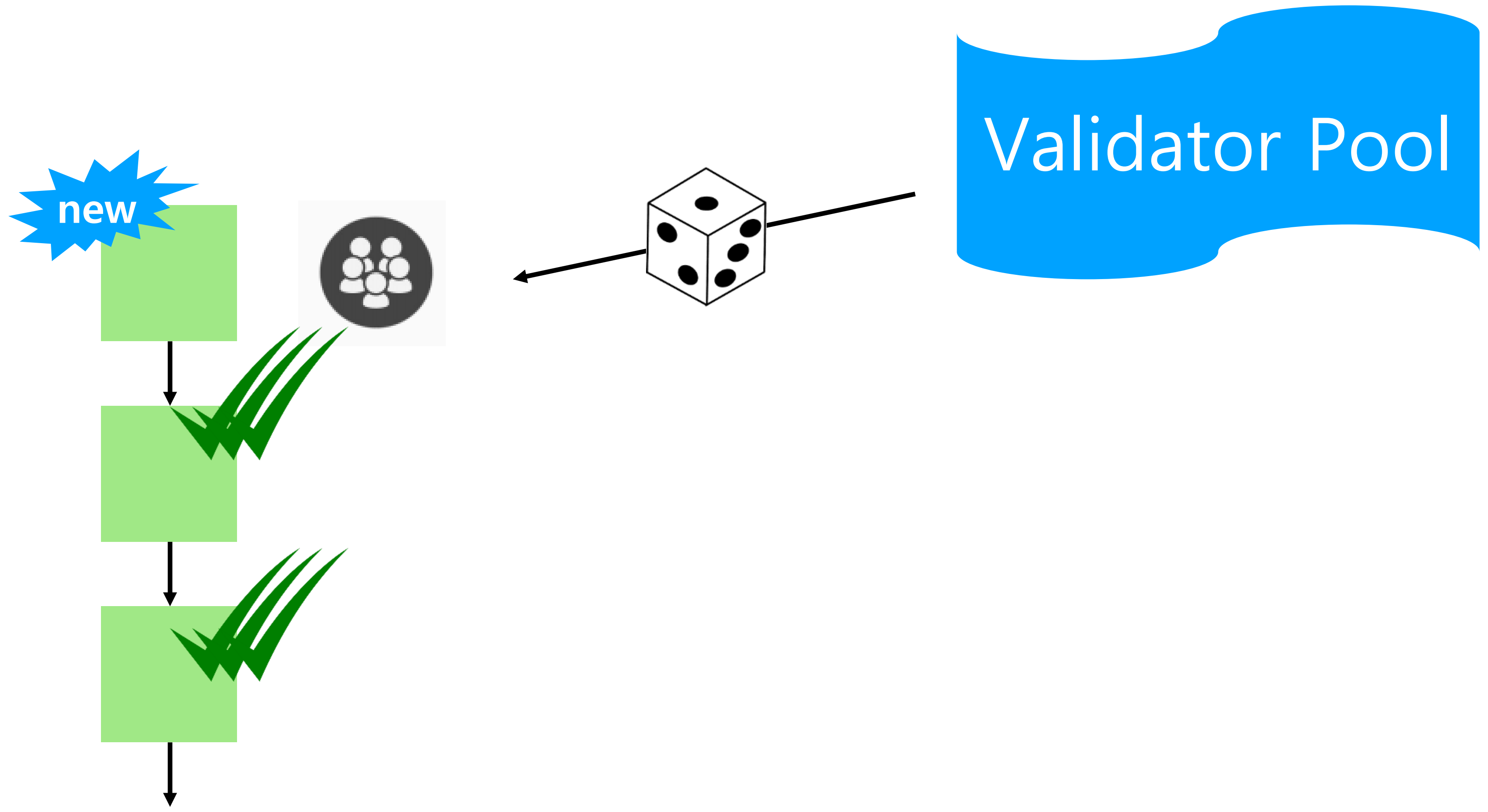
# Relation Between Validators - Attestation

References: Edcon (2018.5.)



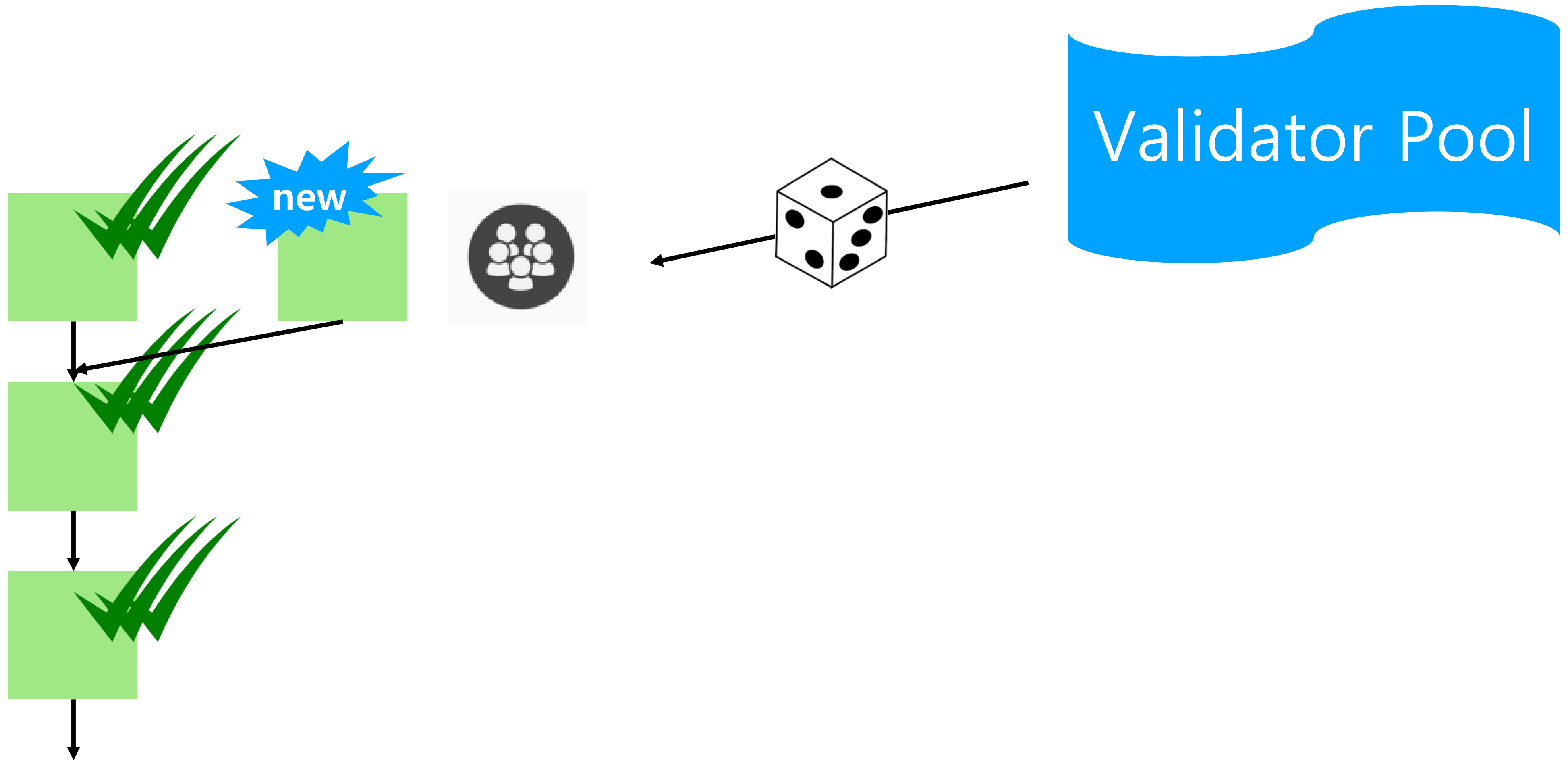
# Relation Between Validators - Attestation

References: Edcon (2018.5.)



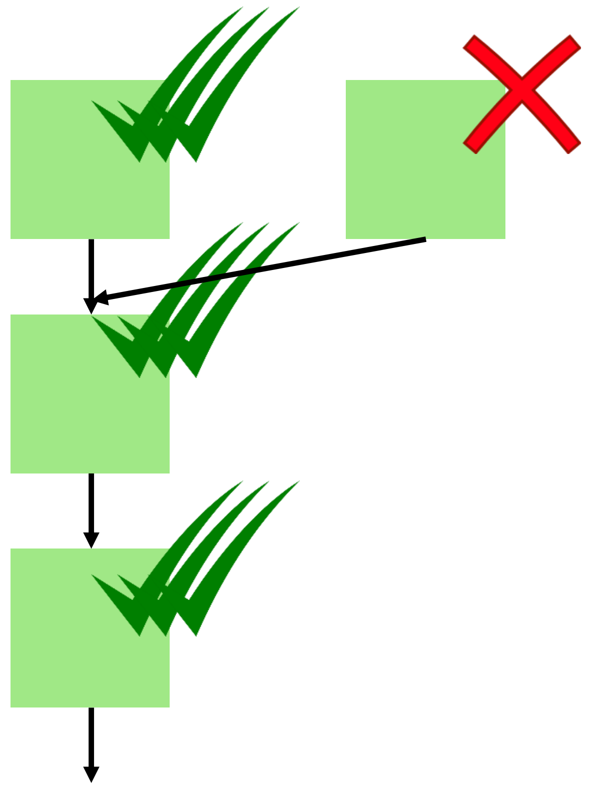
# Relation Between Validators - Attestation

References: Edcon (2018.5.)



# Relation Between Validators - Attestation

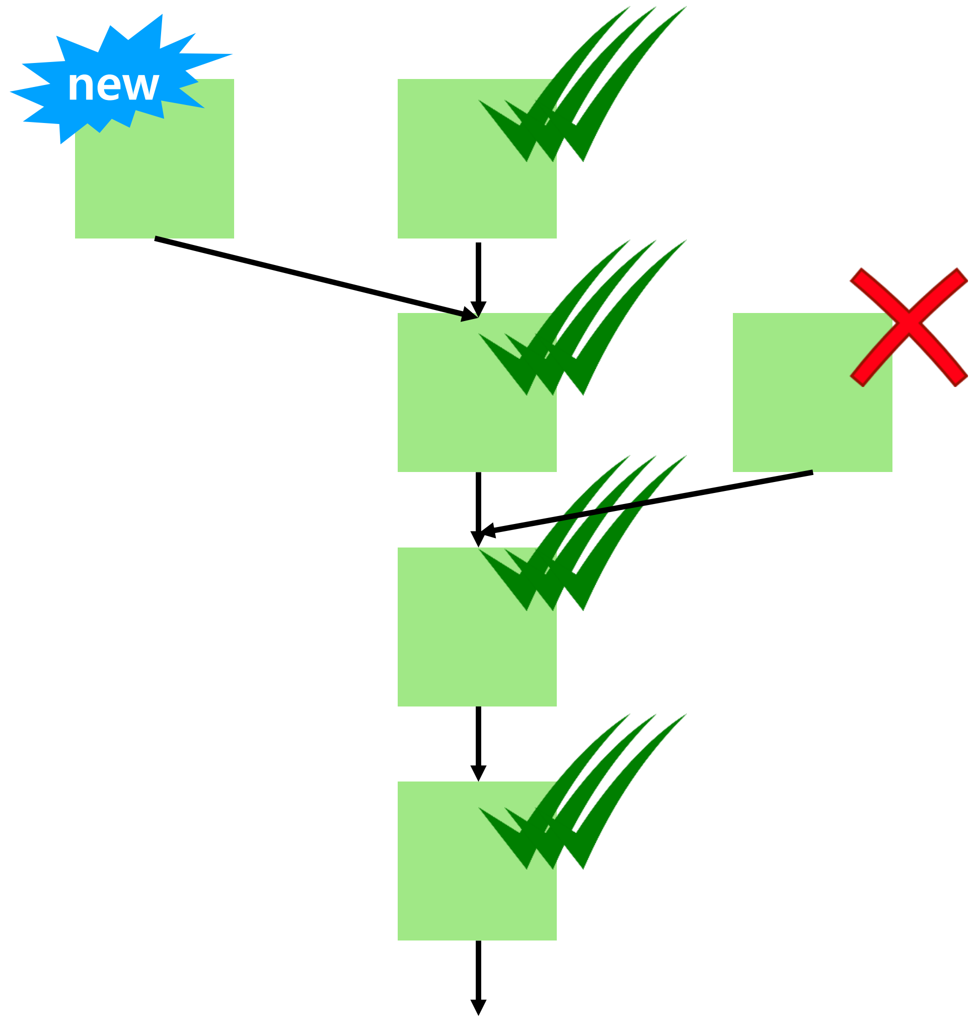
References: Edcon (2018.5.)



Whitepaper
Foundation

# Relation Between Validators - Attestation

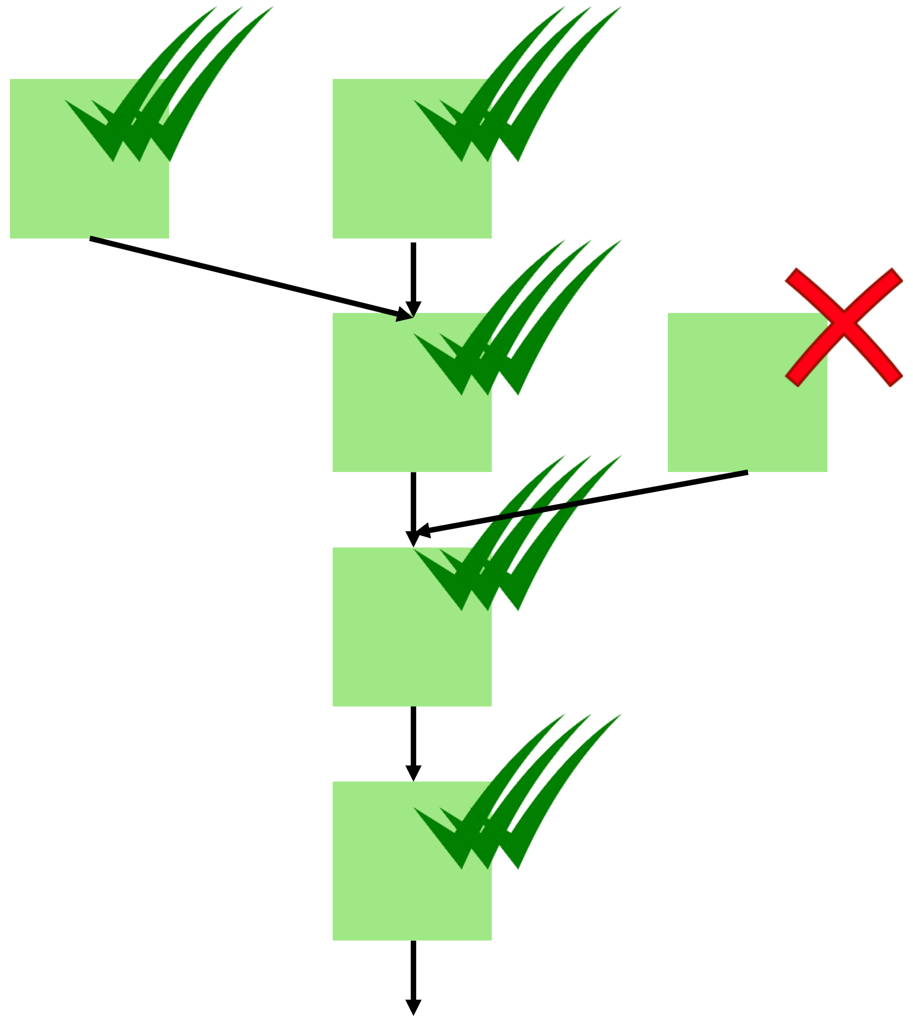
References: Edcon (2018.5.)



# Relation Between Validators - Attestation

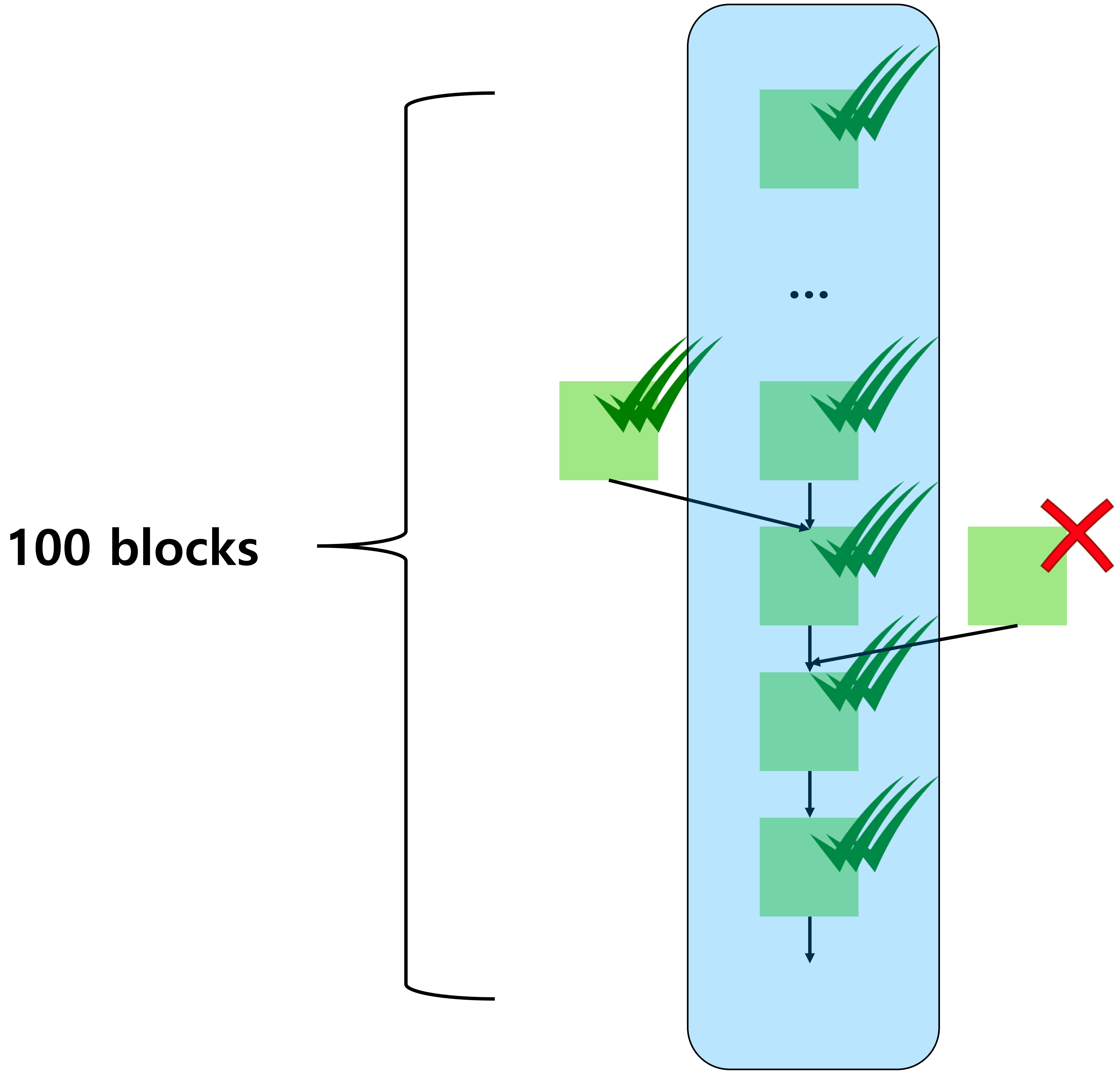
References: Edcon (2018.5.)

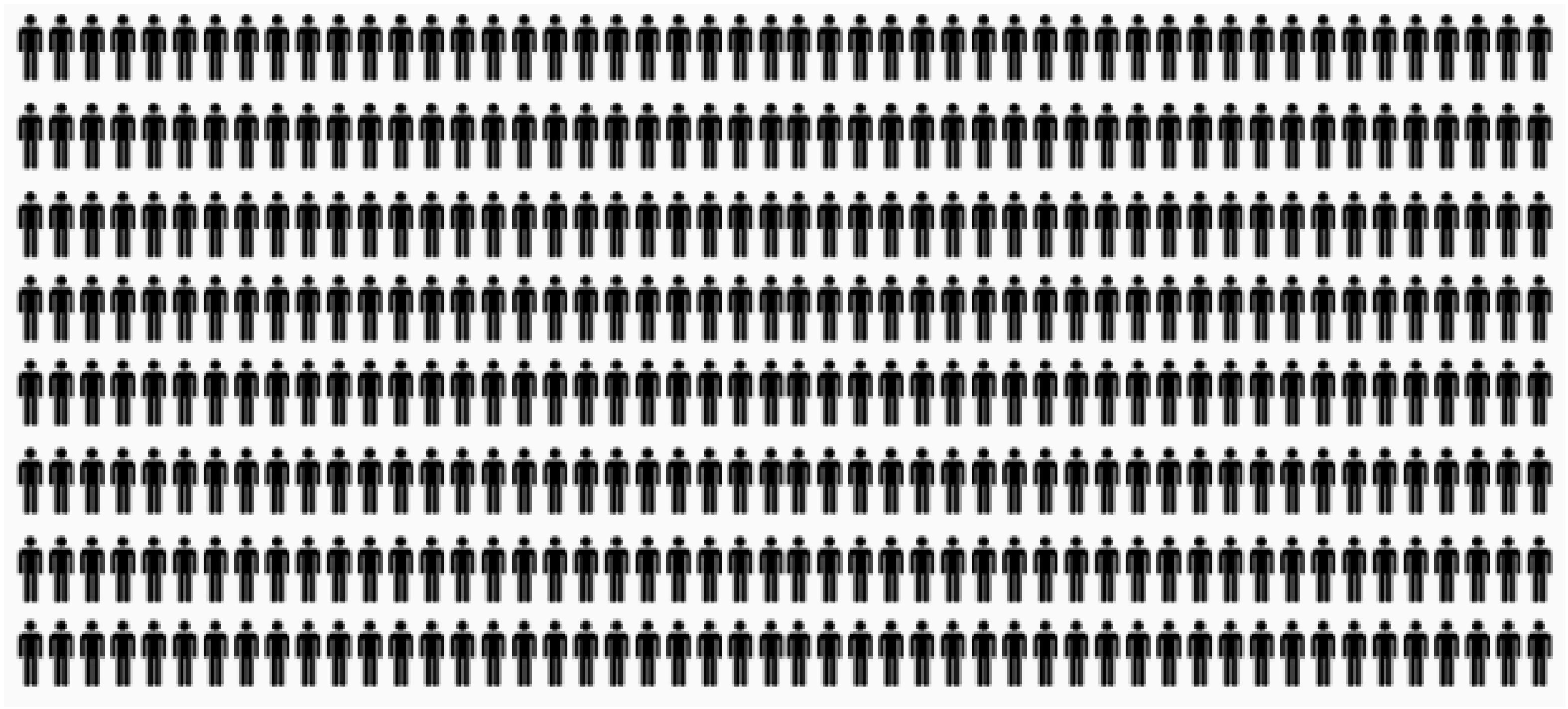
References: Edcon (2018.5.)



# Relation Between Validators - Notarisation

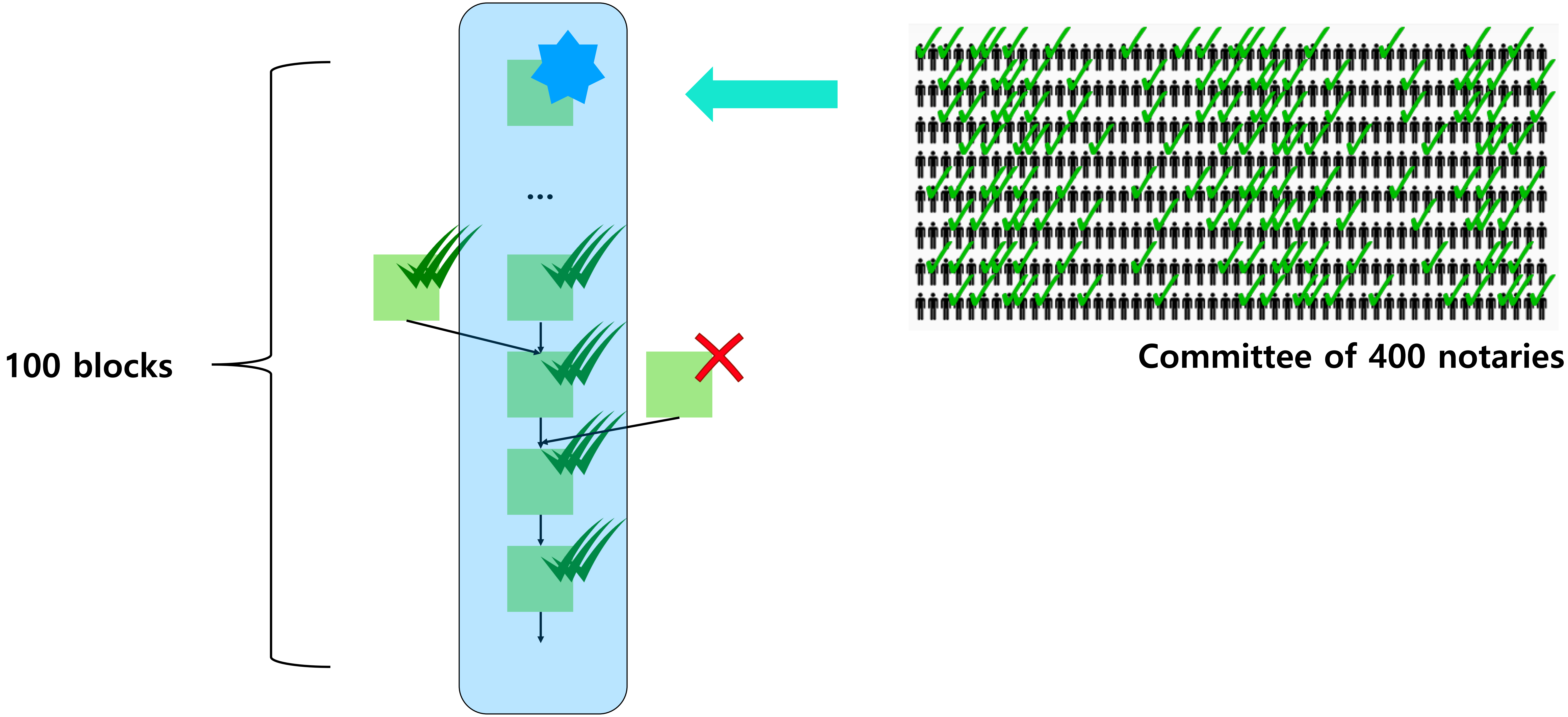
References: Edcon (2018.5.)





Committee of 400 notaries

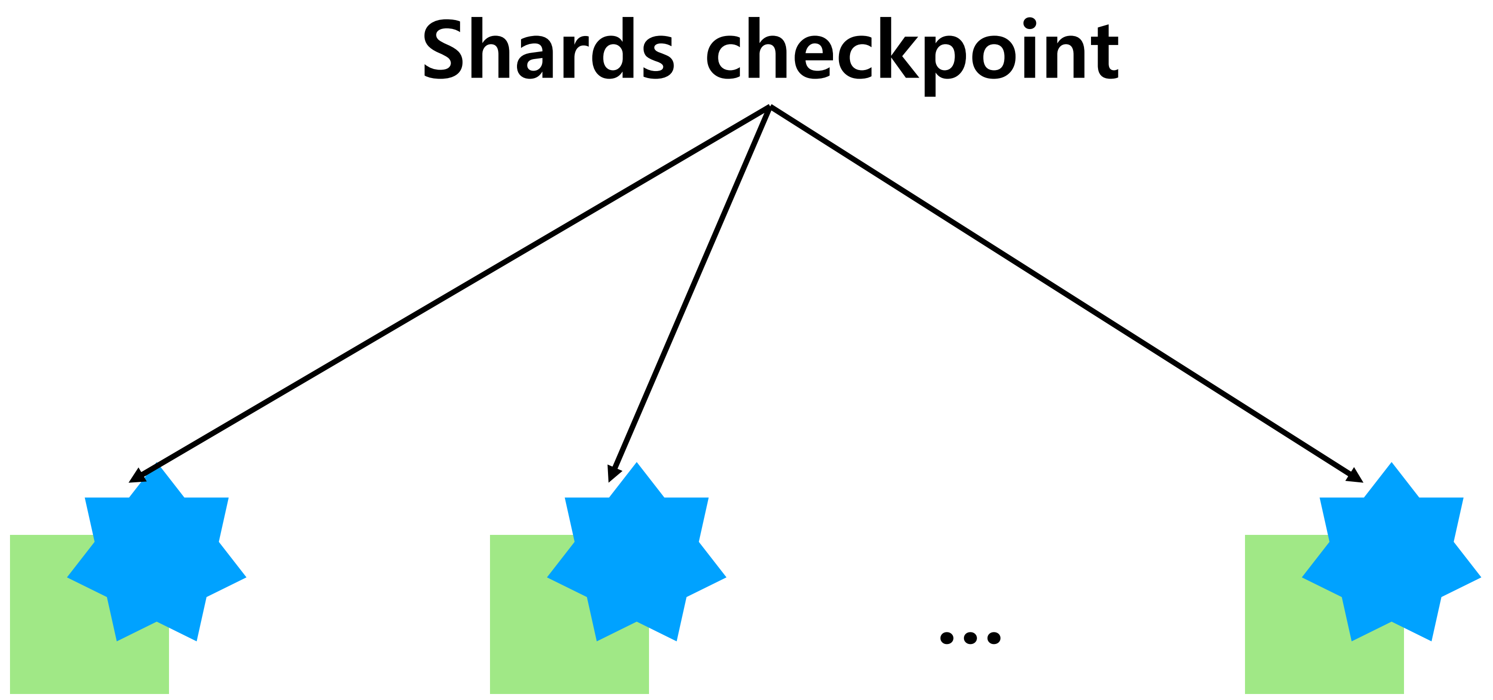
# Relation Between Validators - Notarisation



Whitepaper
Foundation

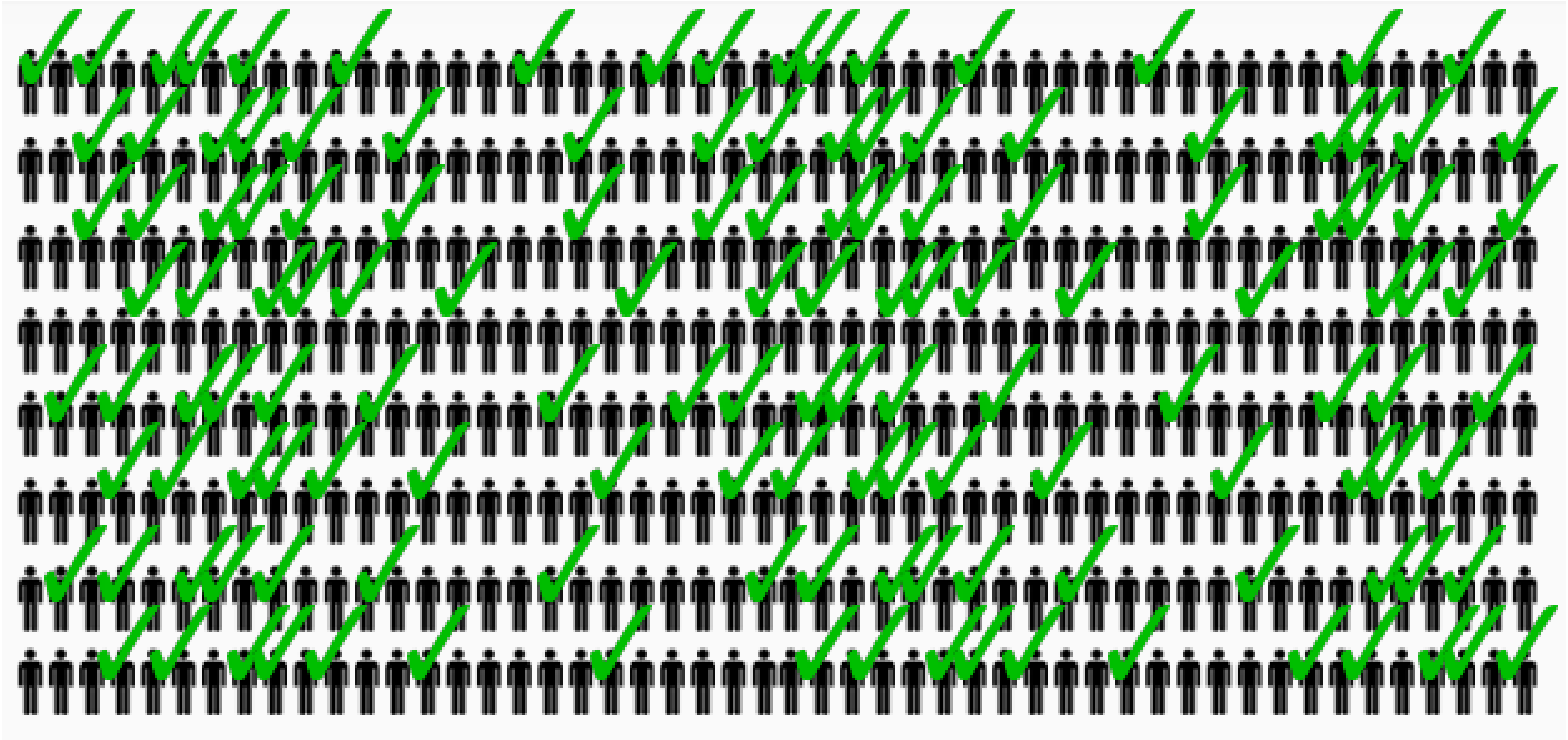
# Relation Between Validators – Meta-Notarisation

References: Edcon (2018.5.)

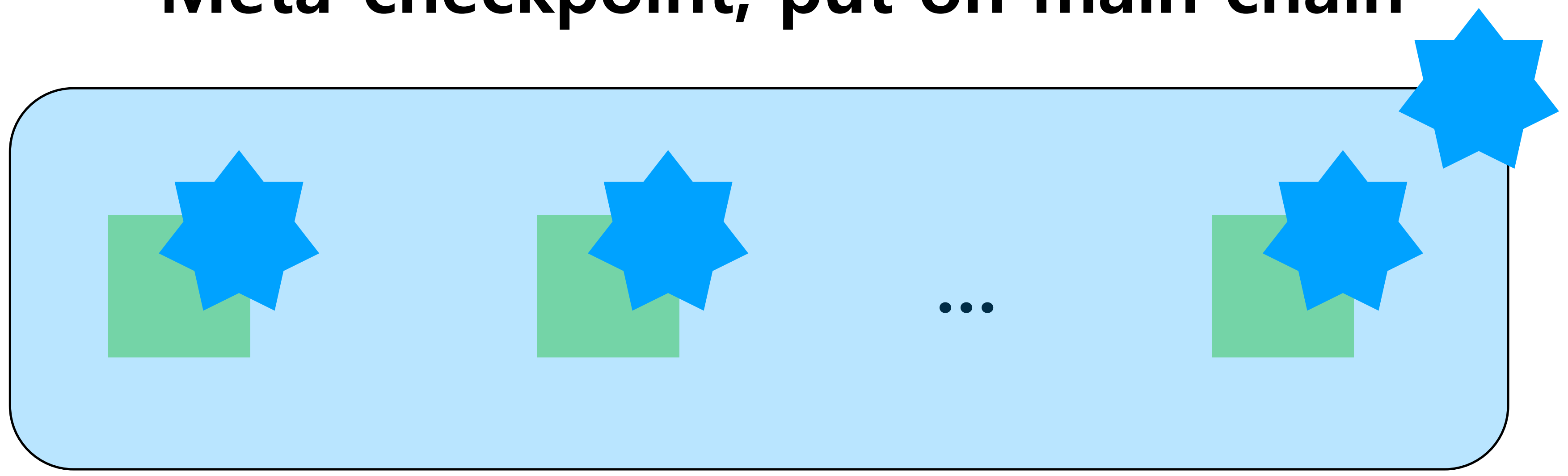


# Relation Between Validators – Meta-Notarisation

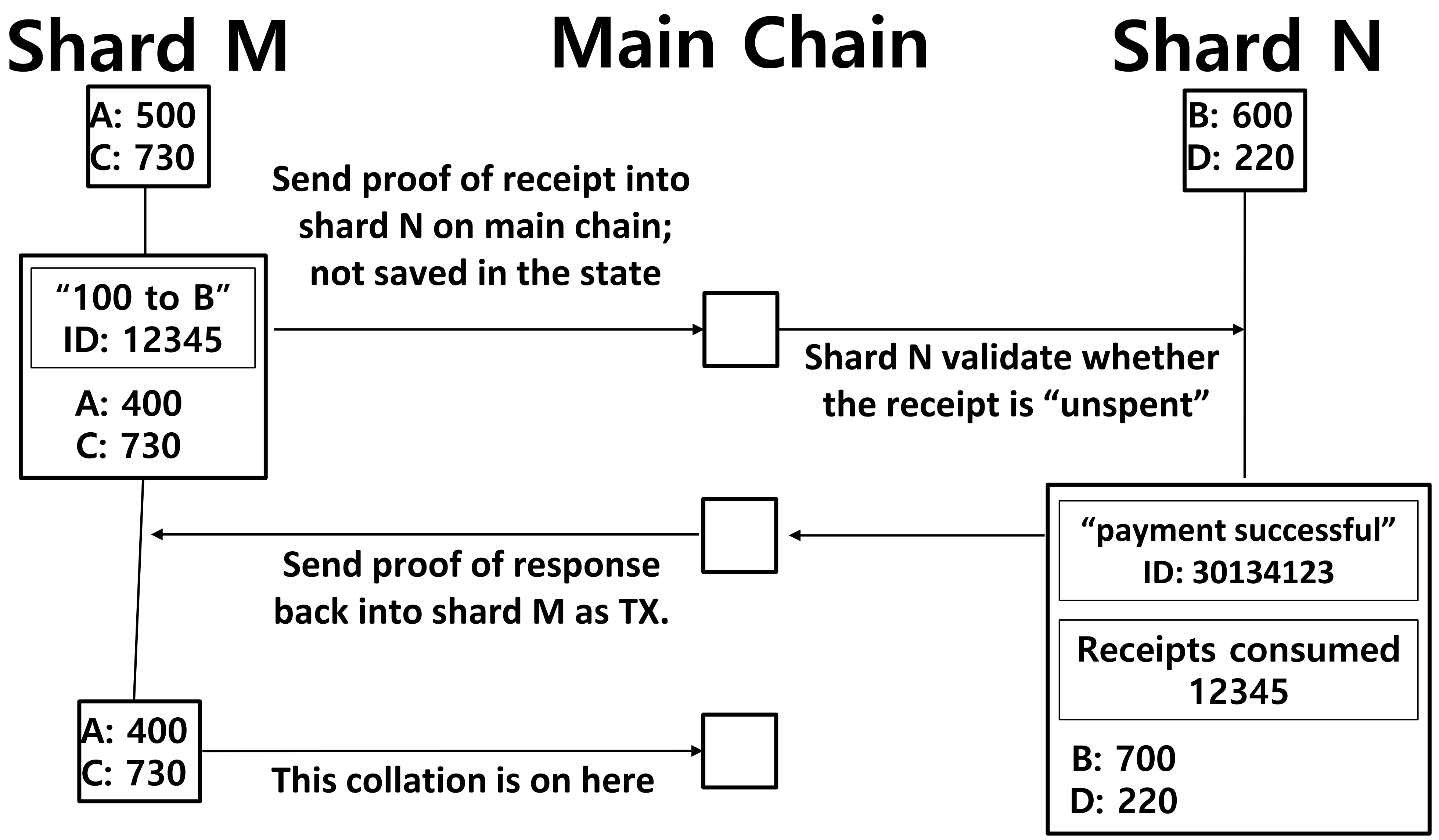
References: Edcon (2018.5.)



Meta-checkpoint, put on main chain



# Cross-shard Communication: Asyncronous


Shard M
Main Chain
Shard N
A: 500
C: 730
B: 600
D: 220
Send proof of receipt into shard N on main chain; not saved in the state
"100 to B"
ID: 12345
A: 400
C: 730
Shard N validate whether the receipt is "unspent"
"payment successful"
ID: 30134123
Receipts consumed 12345
B: 700
D: 220
Send proof of response back into shard M as TX.
A: 400
C: 730
This collation is on here

1. Operation on shard A creates a receipt on shard A
2. The receipt on shard A gets confirmed
3. An operation on shard B incorporates a proof that the receipt on shard A was confirmed, and perform execution based on this

- What if shard A has a large reorg?
  ➔ if B doesn't do reorg, consistency fail
  ➔ if B has a reorg, this could be a DoS attack point
  waiting shard A's finalization is non-sense!

- **Delayed stated execution!**
  ➔ Instead of reorg any tx on shard B after A's reorg, let the executors recalculate the state roots of shard B

# Train and Hotel Problem & Yanking

References: Ithaca (2018.07)

- Suppose you want to book a train ticket and a hotel room, but the transaction is worth it only if you book both

- Want to try to book both, but book neither if booking either one fails

- Suppose train and hotel smart contracts live on different shards

Yanking Scenario

- Step 1: Extract bookable seat into separate contract
- Step 2: Yank it into shard B
- Step 3: If the hotel is still available, atomically book both. Otherwise, give up
- Step 4: yank seat back into shard A, reinsert it into “main” train contract (if needed)

# Cross-shard Communication: Synchronous

References: Ithaca (2018.07)

- The consensus already gives us a total order on messages
- State execution is delayed until consensus on order settles
- Then, a separate process can compute state roots
- Problem: for a node with the state of only one shard, this should not require too many sequential rounds of network communication to fetch Merkle branches of “foreign” shards

  ➔ Synchronous communication scheme is still an open issue!

# Epoch Transition (1)

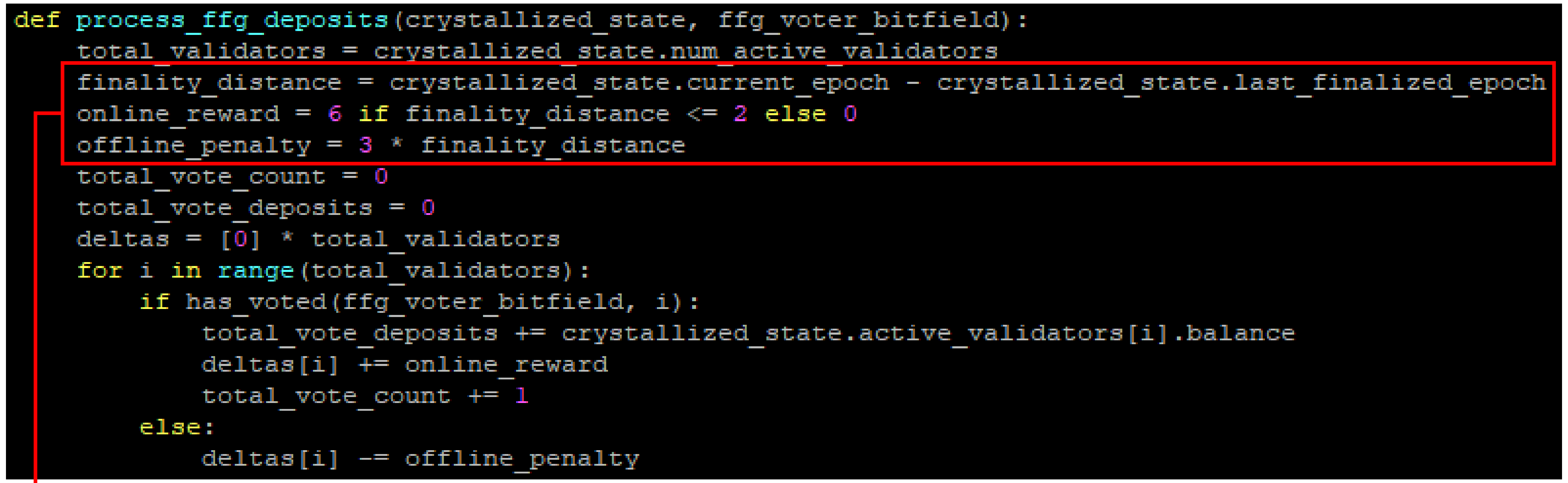
## Calculate rewards within epoch transition

**1) Calculate rewards for FFG votes**

1. Compute the total deposits of every validator who participated in the last epoch (ffg_voter_bitfield in active_state).
   ➔ If this value is >= 2/3 of the total deposits of all validators, update crystallized_state.justified_epoch
   ➔ If this happens, and the justified epoch was previously crystallized_state.current_epoch -1, update the finalized_epoch

2. Compute the online_reward and offline_penalty based on Casper FFG (not fixed yet)

3. Add the online_reward to every validator who participated in the last epoch, and subtract the offline_penalty from everyone who did not



```
def process_ffg_deposits(crystallized_state, ffg_voter_bitfield):
    total_validators = crystallized_state.num_active_validators
    finality_distance = crystallized_state.current_epoch - crystallized_state.last_finalized_epoch
    online_reward = 6 if finality_distance <= 2 else 0
    offline_penalty = 3 * finality_distance
    total_vote_count = 0
    total_vote_deposits = 0
    deltas = [0] * total_validators
    for i in range(total_validators):
        if has_voted(ffg_voter_bitfield, i):
            total_vote_deposits += crystallized_state.active_validators[i].balance
            deltas[i] += online_reward
            total_vote_count += 1
        else:
            deltas[i] -= offline_penalty
```

Force validators to be diligent

# Epoch Transition (2)

## Calculate rewards within epoch transition

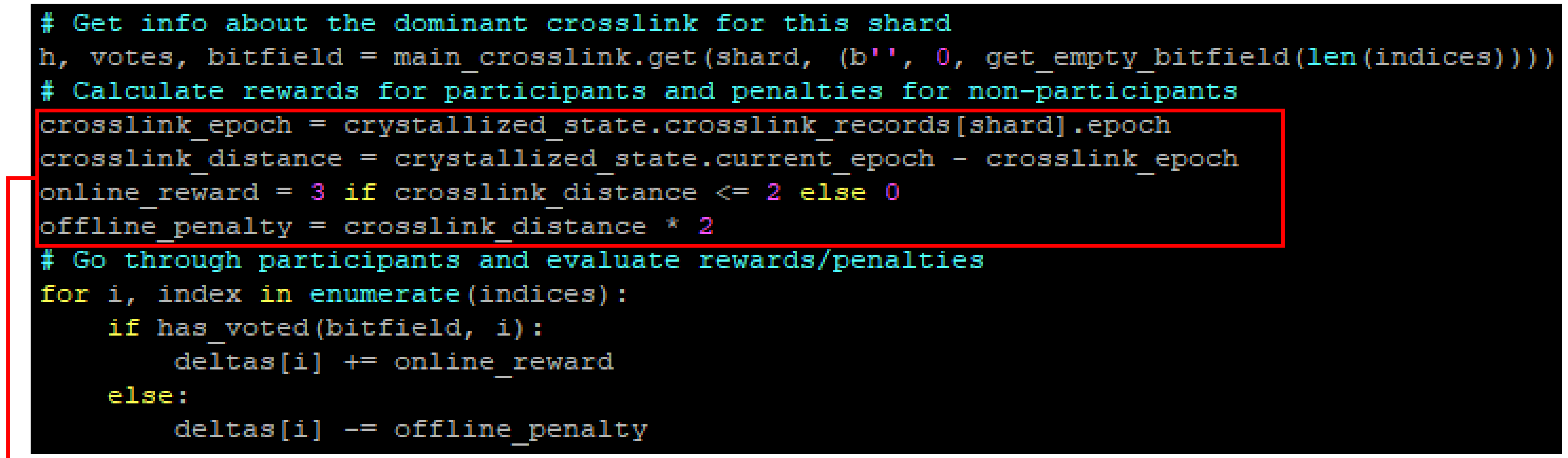
### 2) Calculate rewards for crosslinks

Repeat for every shard
(After finding the most popular crosslink in each shard)

1. Calculate the online_reward and offline_penalty for that crosslink

2. Rewards any validator that participated in that partial crosslink; penalize any validator who did not

3. If any crosslink reaches >= 2/3 of its sample, weighted by total deposits, save it as the most recent crosslink
   (NOT INCLUDING any balance deltas that are part of this epoch transition)



```
# Get info about the dominant crosslink for this shard
h, votes, bitfield = main_crosslink.get(shard, (b'', 0, get_empty_bitfield(len(indices))))
# Calculate rewards for participants and penalties for non-participants
crosslink_epoch = crystallized_state.crosslink_records[shard].epoch
crosslink_distance = crystallized_state.current_epoch - crosslink_epoch
online_reward = 3 if crosslink_distance <= 2 else 0
offline_penalty = crosslink_distance * 2
# Go through participants and evaluate rewards/penalties
for i, index in enumerate(indices):
    if has_voted(bitfield, i):
        deltas[i] += online_reward
    else:
        deltas[i] -= offline_penalty
```

Force validators to be diligent

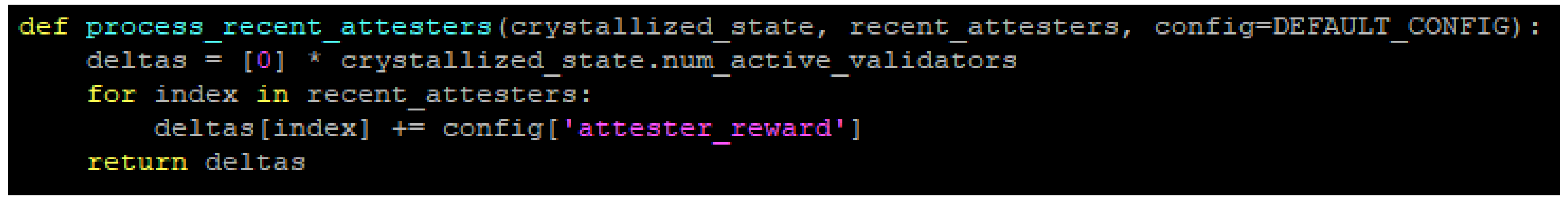
# Epoch Transition (3)

## Calculate rewards within epoch transition

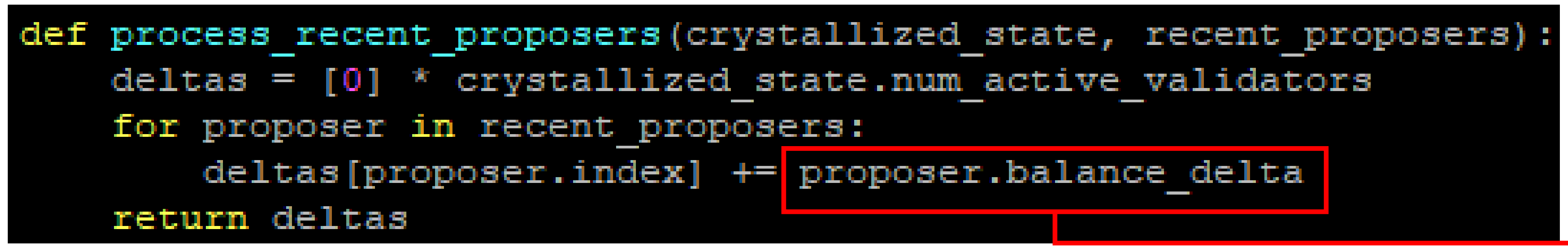
### 3) Process balance deltas (attesters & proposers)

1. Increase the balance of everyone in **recent_attesters** by 1

2. Increase the balance of everyone in **recent_proposers** by the balance_delta in the RecentProposerRecord object (giving the block proposer a reward of n if there are n total voters that have not yet voted)



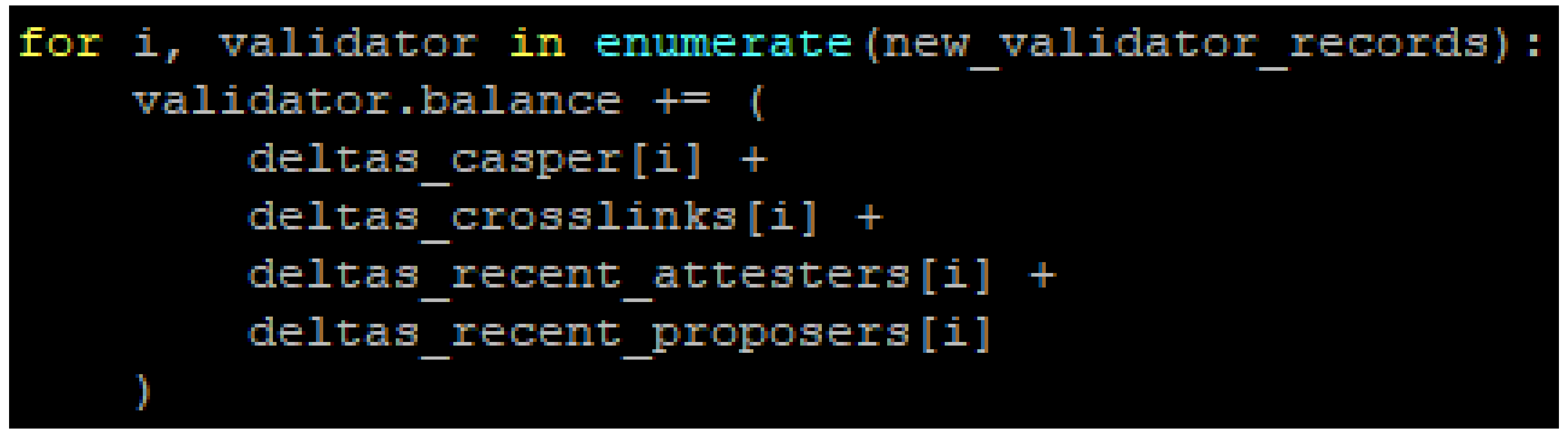
```
def process_recent_attesters(crystallized_state, recent_attesters, config=DEFAULT_CONFIG):
    deltas = [0] * crystallized_state.num_active_validators
    for index in recent_attesters:
        deltas[index] += config['attester_reward']
    return deltas
```



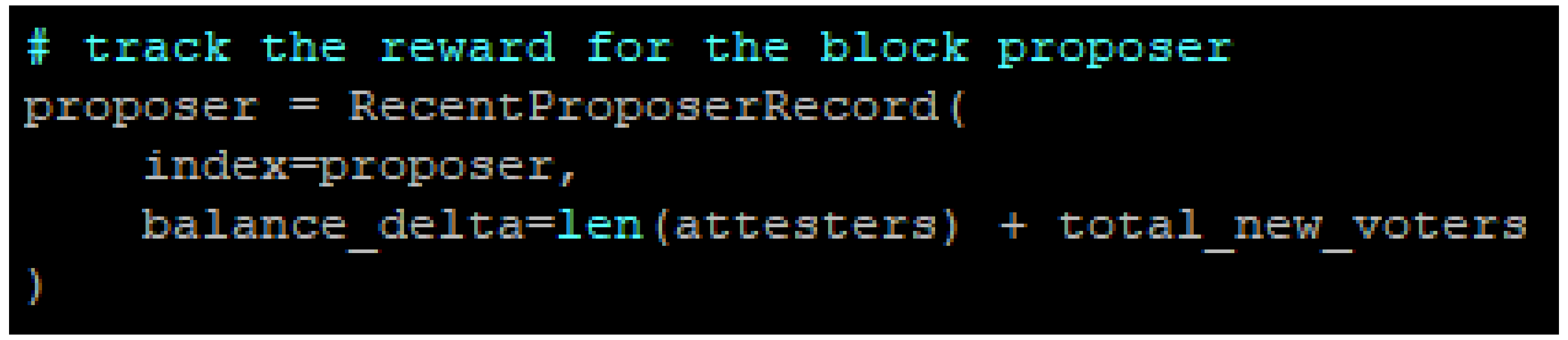
```
def process_recent_proposers(crystallized_state, recent_proposers):
    deltas = [0] * crystallized_state.num_active_validators
    for proposer in recent_proposers:
        deltas[proposer.index] += proposer.balance_delta
    return deltas
```

The way how to calc balance_delta has a bug! (20180719)

### 4) Add calculated balance on validators



```
for i, validator in enumerate(new_validator_records):
    validator.balance += (
        deltas_casper[i] +
        deltas_crosslinks[i] +
        deltas_recent_attesters[i] +
        deltas_recent_proposers[i]
    )
```



```
# track the reward for the block proposer
proposer = RecentProposerRecord(
    index=proposer,
    balance_delta=len(attesters) + total_new_voters
)
```

# Epoch Transition (4)

## Prepare the next epoch

**5) Crosslink seed-related calculation**

**6) Dynasty transition**

1. Check whether this is justified
2. Check whether this is finalized
3. Reset crystalized_state & active_state
4. etc etc (increase the current epoch,

# Still Have to Cover

- Random Process in Beacon Chain
- Delayed State Execution

# References

- https://medium.com/@icebearhww/ethereum-sharding-and-finality-65248951f649
- https://github.com/ethereum/sharding/blob/develop/docs/doc.md
- https://github.com/ethereum/wiki/wiki/Sharding-FAQ
- https://github.com/ethereum/wiki/wiki/chain-fibers-redux
- https://github.com/ethereum/sharding/tree/develop/sharding
- https://medium.com/l4-media/making-sense-of-ethereums-layer-2-scaling-solutions-state-channels-plasma-and-truebit-22cb40dcc2f4