Ethereum
Beacon Chain Code Review

2018. 7. 12

Geore Han
<kr8534@gmail.com>

Outline

- Code Overview
- Relations Between Validators
- Minor Implementation details

Overview

ethereum/beacon_chain

beacon_chain

State

- config
- block
- active_state
- crystallized_state
- aggregate_vote
- crosslink_record
- partitial_crosslink_record
- …

Util

- Signature Scheme (BLS, Blake)
- Serialization Util
- Bit Field

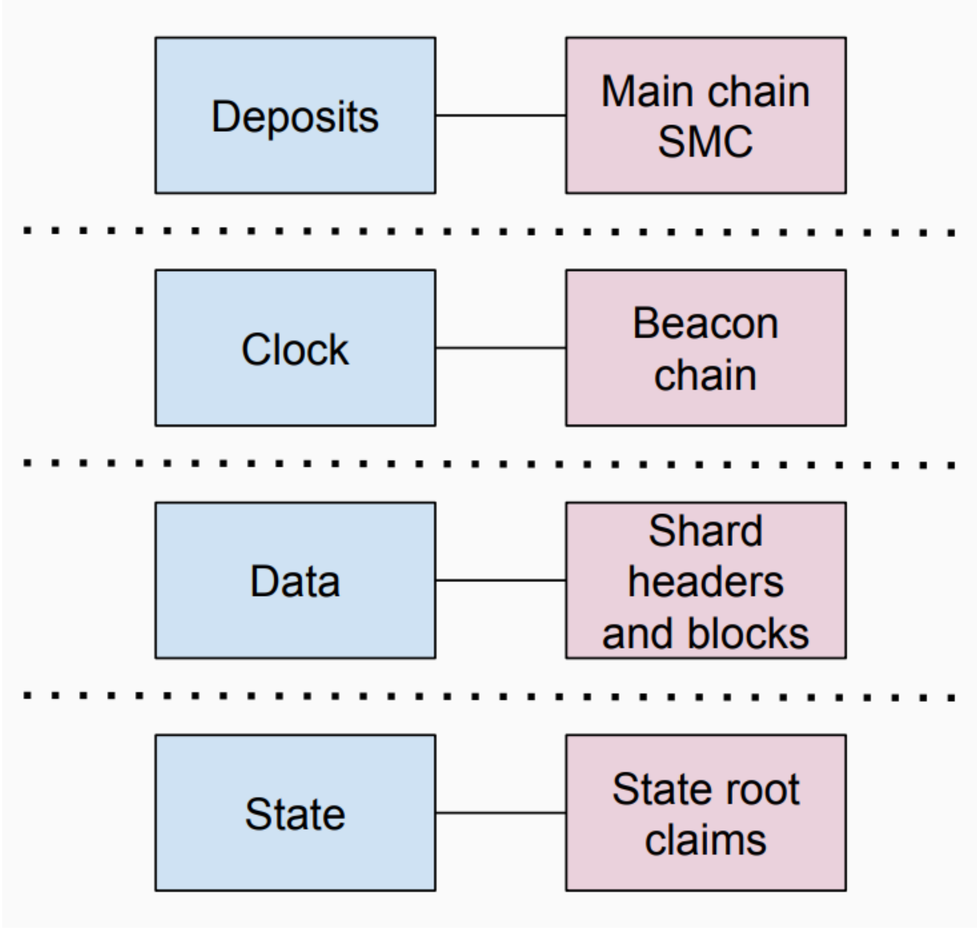
contracts

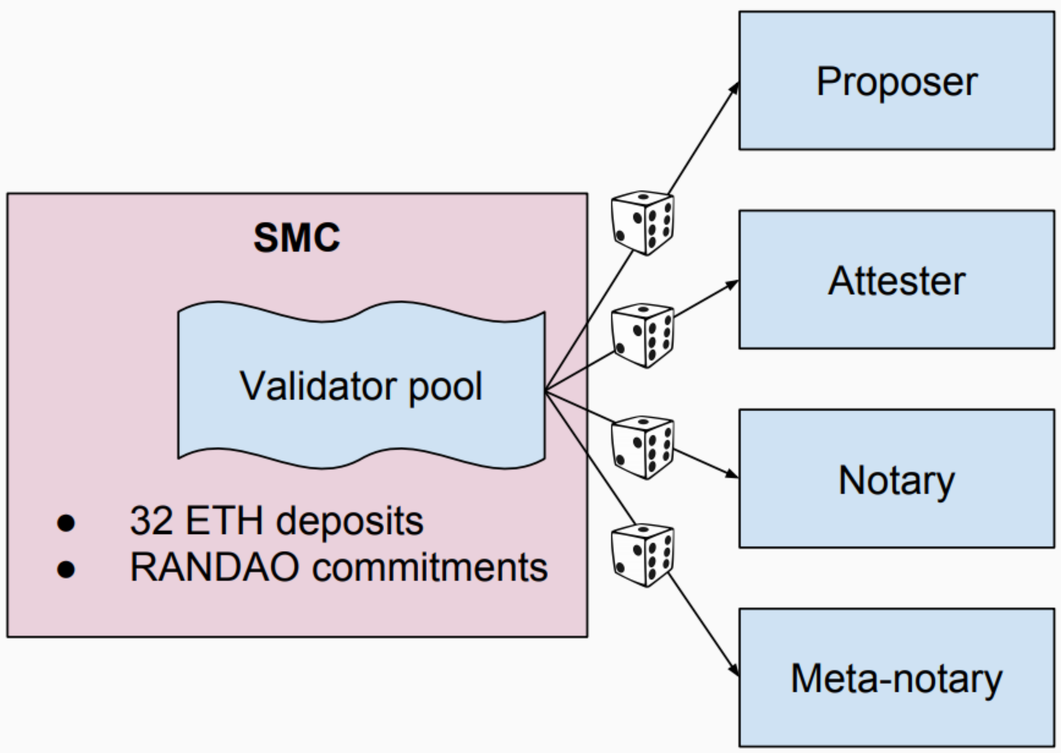
- Keep looking deposit event so that register validator

- Written in python 3.6
- Repo includes unit tests for all python files in beacon_chain and contracts
  (12 test files of 81 items)
- Main file doesn't exist yet
  ➔ hard to see the flowchart!

Beacon Chain & Validators

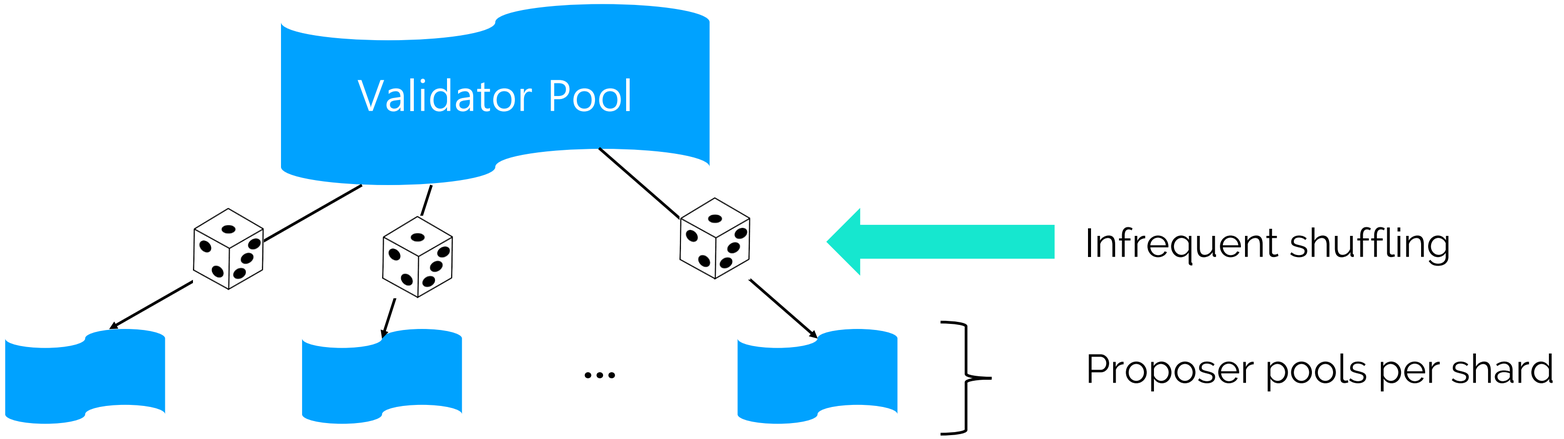
References: Edcon (2018.5.)





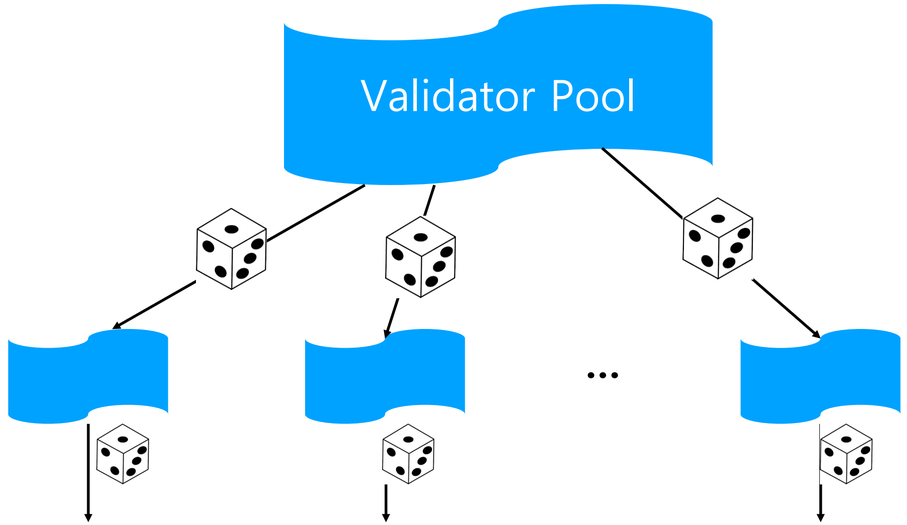
Relation Between Validators

References: Edcon (2018.5.)

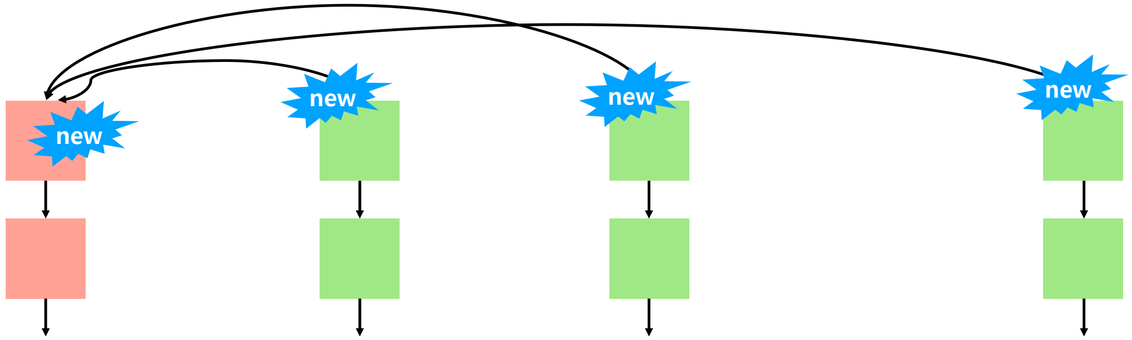


Relation Between Validators

References: Edcon (2018.5.)

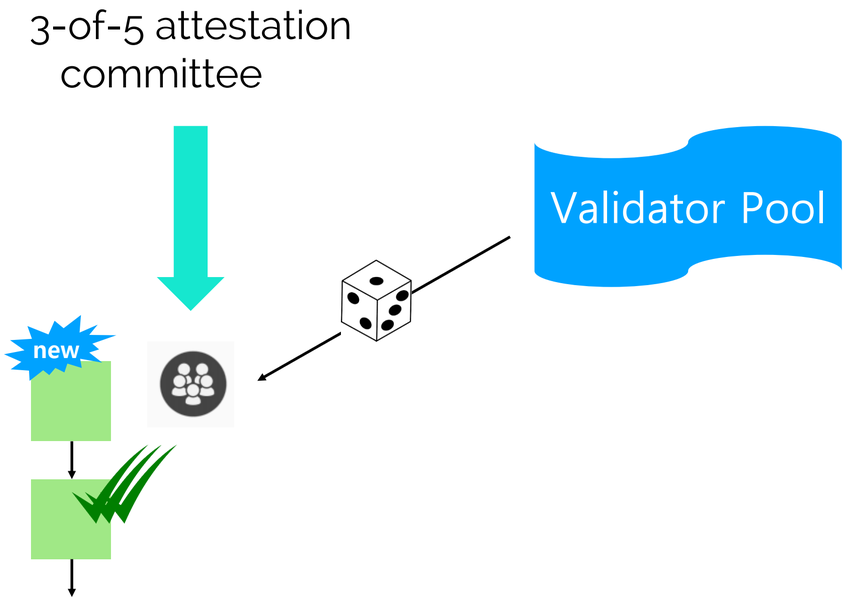


Sample proposers
per heartbeat



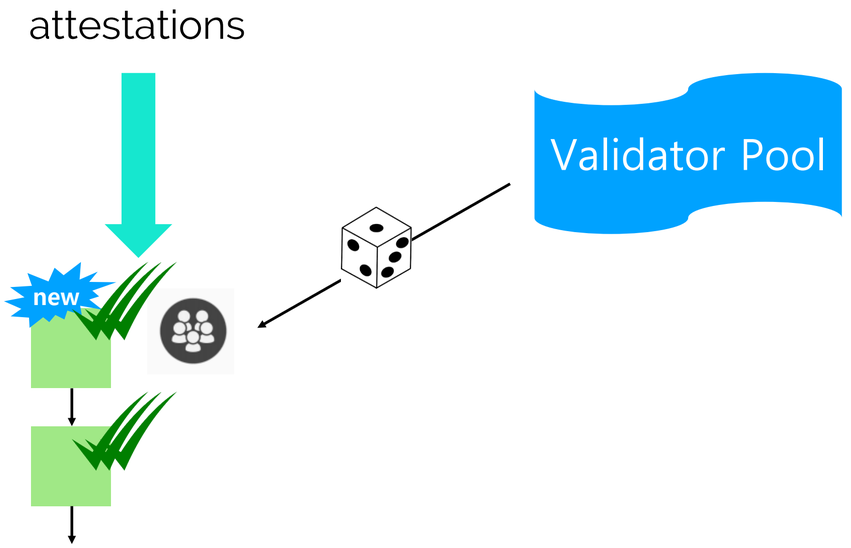
Relation Between Validators - Attestation

References: Edcon (2018.5.)



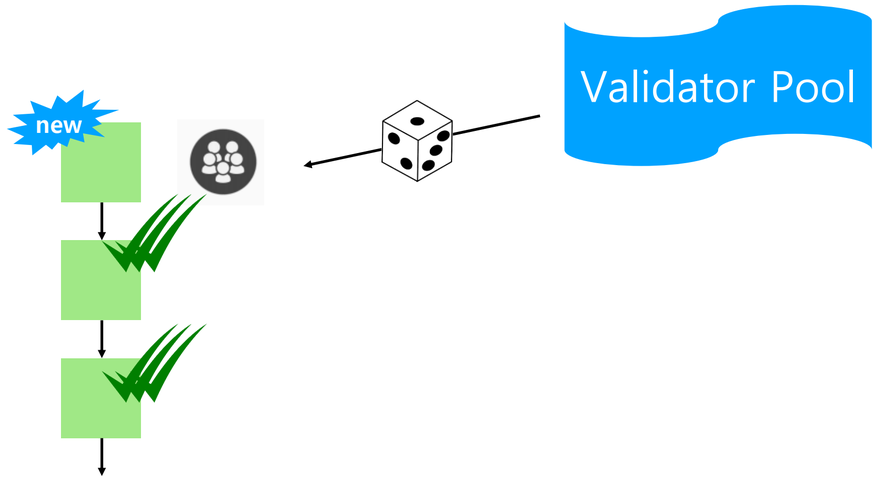
Relation Between Validators - Attestation

References: Edcon (2018.5.)



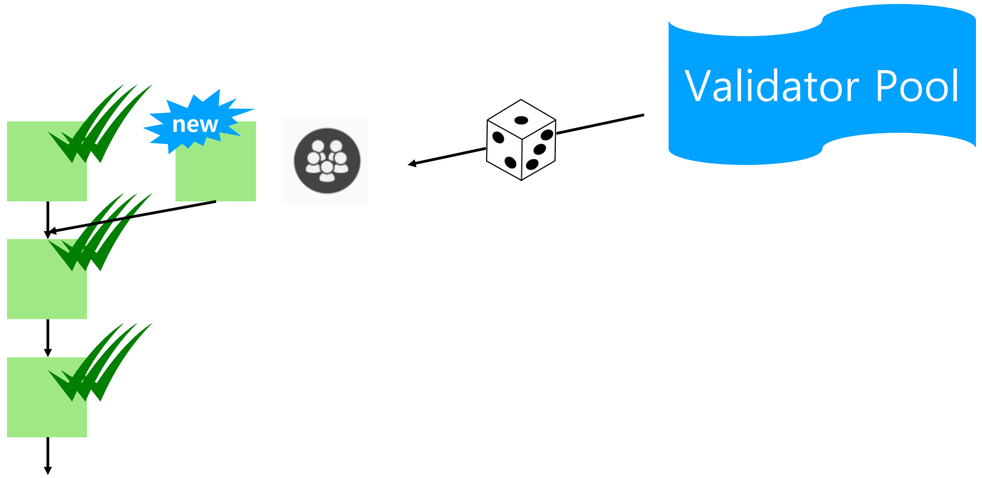
Relation Between Validators - Attestation

References: Edcon (2018.5.)



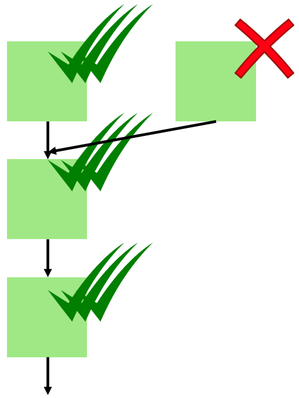
Relation Between Validators - Attestation

References: Edcon (2018.5.)



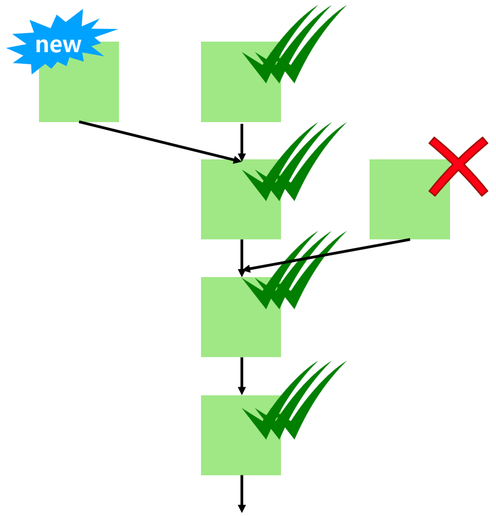
Relation Between Validators - Attestation

References: Edcon (2018.5.)



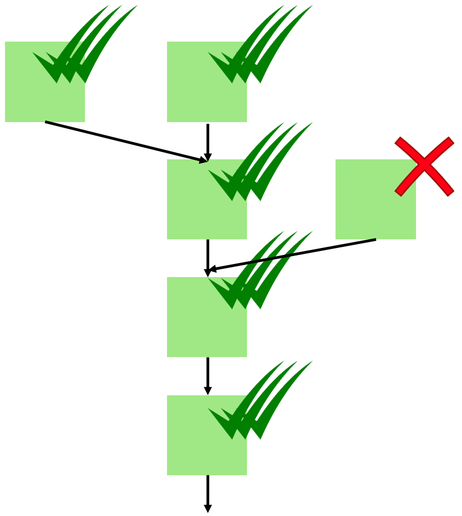
Relation Between Validators - Attestation

References: Edcon (2018.5.)



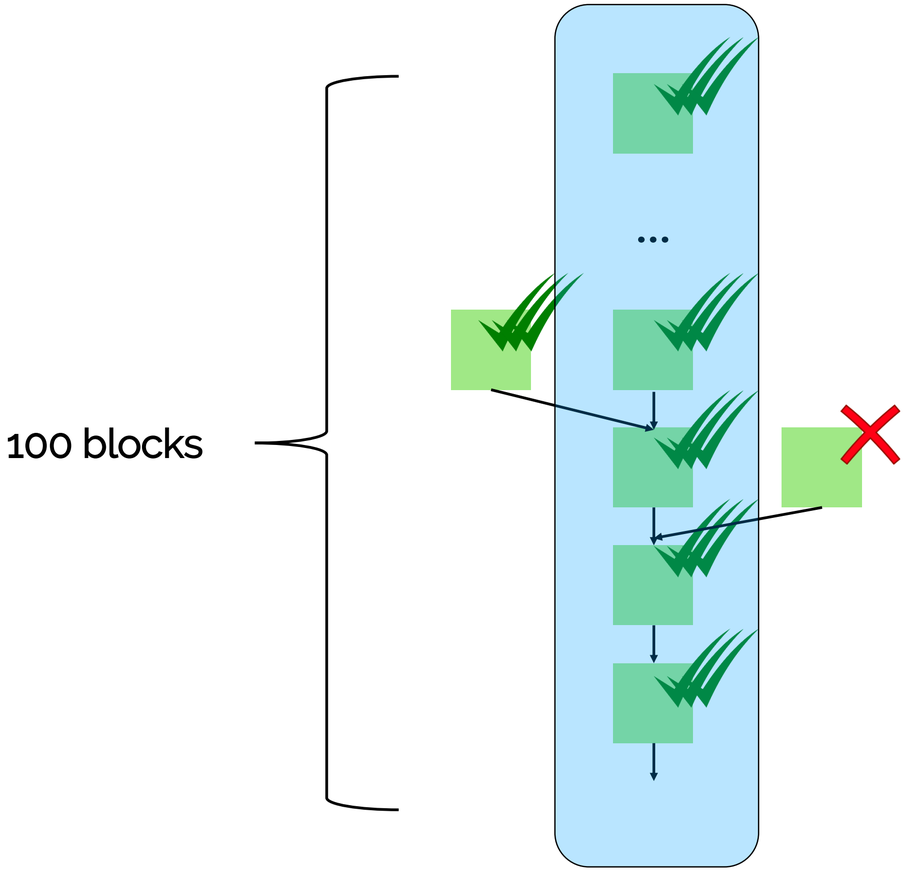
Relation Between Validators - Attestation

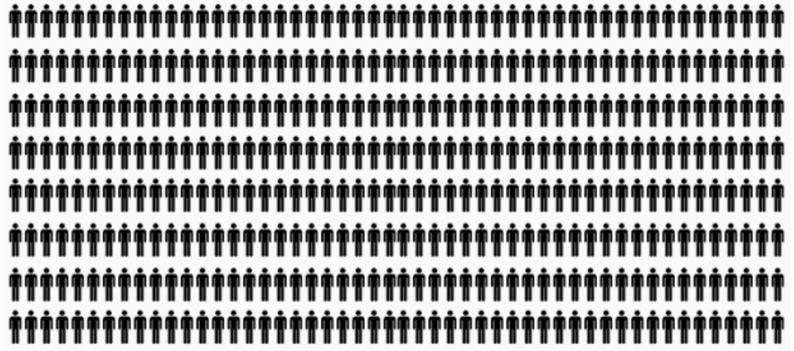
References: Edcon (2018.5.)



Relation Between Validators - Notarisation

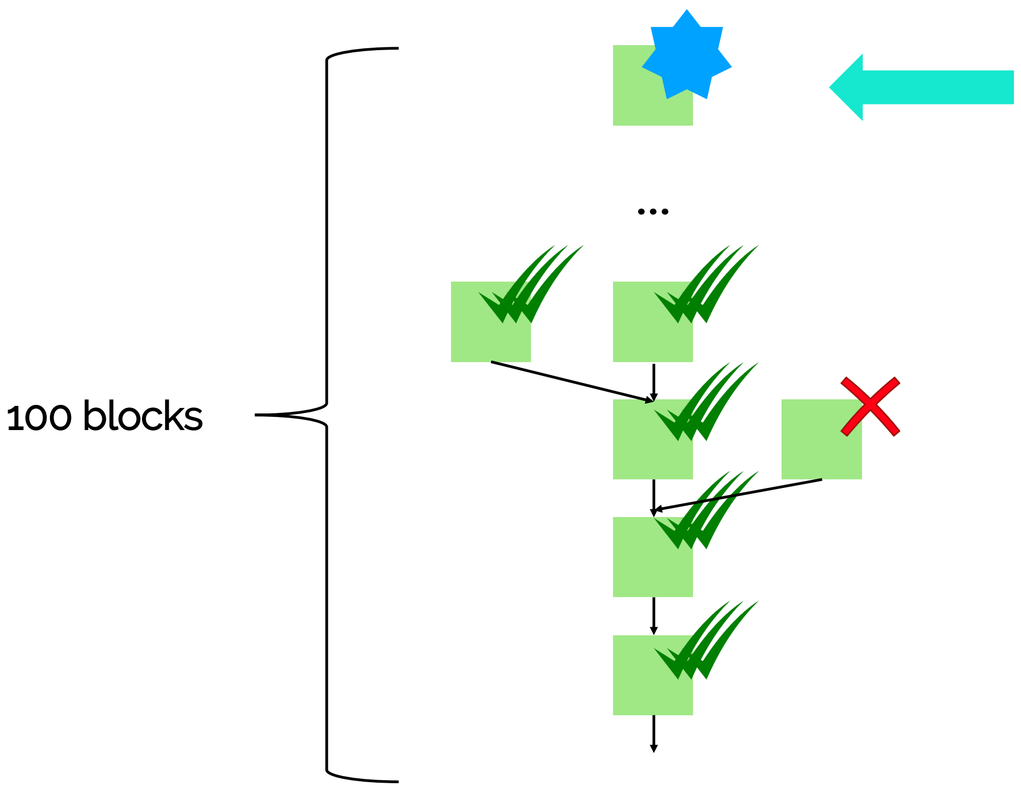
References: Edcon (2018.5.)





Committee of 400 notaries

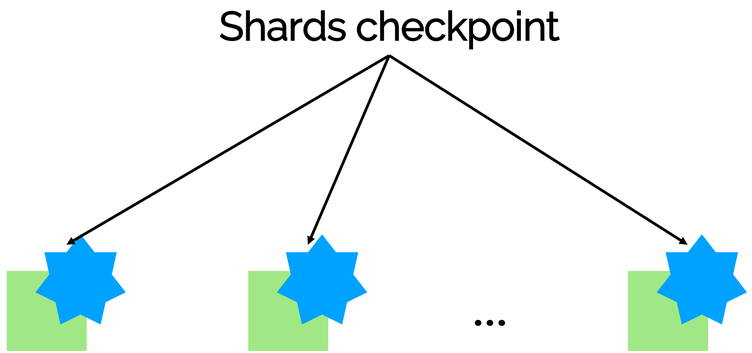
Relation Between Validators - Notarisation



Committee of 400 notaries

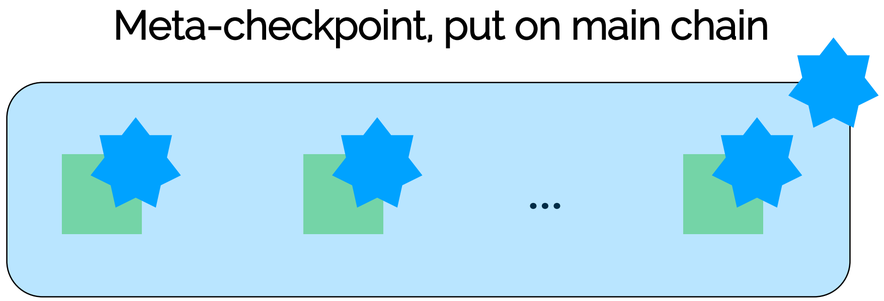
Relation Between Validators – Meta-Notarisation

References: Edcon (2018.5.)



Relation Between Validators – Meta-Notarisation

References: Edcon (2018.5.)



Unit Test (w/ AggregateVote)

- A software testing method by which individual units of source code

```python
class AggregateVote():
    fields = {
        'shard_id': 'int16',
        'shard_block_hash': 'hash32',
        'notary_bitfield': 'bytes',
        'aggregate_sig': ['int256']
    }
    defaults = {
        'shard_id': 0,
        'shard_block_hash': b'\x00'*32,
        'notary_bitfield': b'',
        'aggregate_sig': [0, 0],
    }

    def __init__(self, **kwargs):
        for k in self.fields.keys():
            assert k in kwargs or k in self.defaults
            setattr(self, k, kwargs.get(k, self.defaults.get(k)))

    @property
    def num_aggregate_sig(self):
        return len(self.aggregate_sig)
```

aggregate_vote.py

```python
def test_num_properties():
    aggregate_vote = AggregateVote(
        aggregate_sig=list(range(3))
    )

    assert aggregate_vote.num_aggregate_sig == 3
```

test_aggregate_vote.py

- Check whether *num_aggregate_sig* works well or not

- This includes a bug
  ➔ Because a signature includes a pair, it shouldn't be just len(list)

Aggregate Signature

References: Edcon (2018.5.)

Signature claims bitfield

```
01001110101010101010111001010101011101010101001011
10101010111001010101011101010101001011010011101010
01000101010101111101010101001011100101010101110101
01011101101010101110010101010101001011010011101010
01001110101010101010111001010101011101010101
10101010111001010101011101010101001011010011101010
01000101010101111101010101001011100101010101110101
01011101101010101110010101010101001011010011101010
```

Bad claim! Challenge!

bitfield

- attestation_bitfield is byte (in python, b'')
- So it refers to bitmask who participated in the block notarization
- Bitfield.py includes related util functions

```python
def has_voted(bitfield, index):
    return bool(bitfield[index // 8] & (128 >> (index % 8)))


def set_voted(bitfield, index):
    byte_index = index // 8
    bit_index = index % 8
    new_byte_value = bitfield[byte_index] | (128 >> bit_index)
    return bitfield[:byte_index] + bytes([new_byte_value]) + bitfield[byte_index + 1:]


def get_bitfield_length(bit_count):
    """Return the length of the bitfield for a given number of attesters in bytes."""
    return (bit_count + 7) // 8


def get_empty_bitfield(bit_count):
    return b"\x00" * get_bitfield_length(bit_count)

def get_vote_count(bitfield):
    votes = 0
    for index in range(len(bitfield) * 8):
        if has_voted(bitfield, index):
            votes += 1
    return votes
```

bitfield.py

Questions Last Time

- What's attestation_bitfield & shard_aggregate_vote?
  Who is the owner of aggregate_sig?

- What's notary's role? Who makes crosslinks ? ➔ block's proposer. Notaries sign on it
  Difference between crosslink & partial_crosslink?

- What does the state_hash in beacon block means?
  Crystalized_state & active_state?

- The way how to determine block's proposer & attester

- Balance-delta?

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