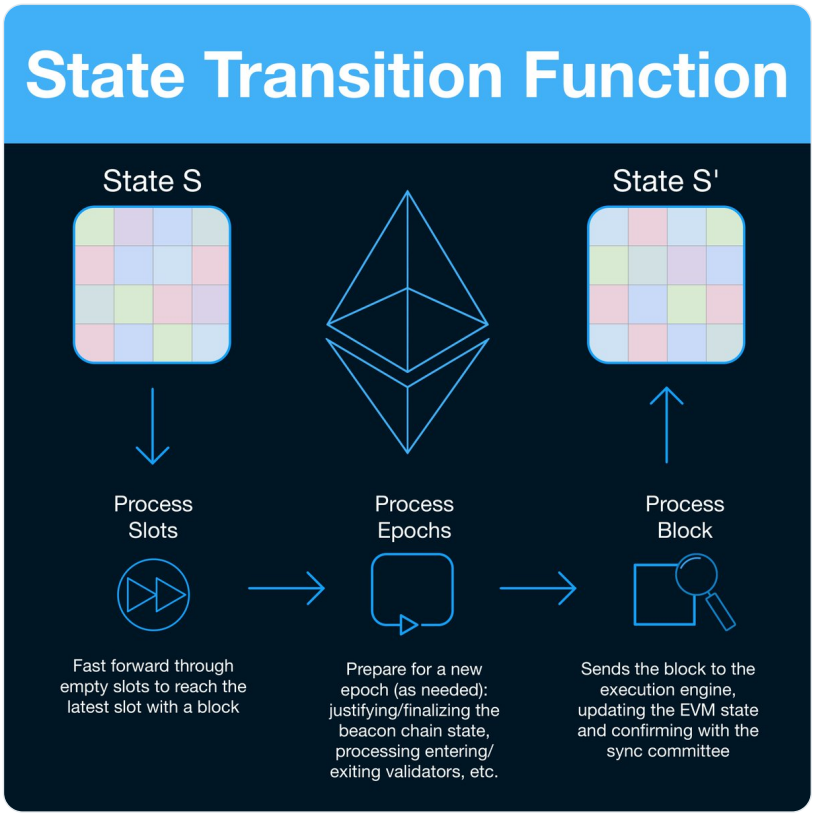
Haym @SalomonCrypto

Oct 5 · 18 tweets · SalomonCrypto/status/1577751304696274944

(1/17) @ethereum Fundamentals: The (Post-Merge) State Transition Function

The World Computer is a decentralized state machine, let's walk through the state transition function

Sound like nonsense? This thread will explain what happens every time a validator receives a new block

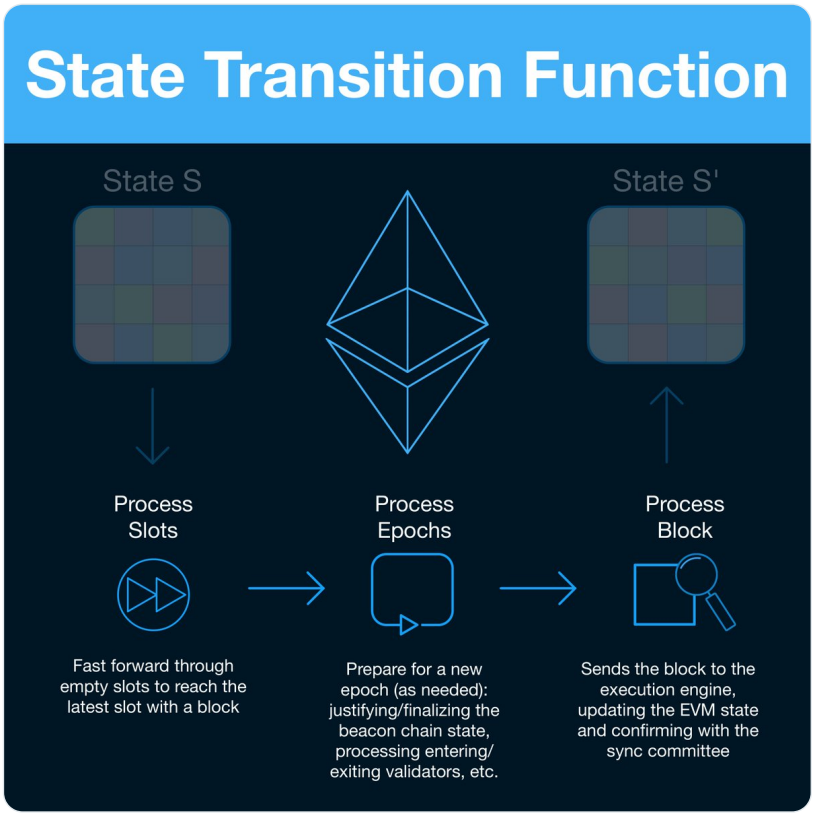


(2/17) @ethereum is (probably) most accurately described as a distributed state machine - a model of computation with 2 major components:

- the state, the configuration of the system at a moment in time
- a transaction, the set of instructions to change the state

(3/17) A state is a static snapshot in time, the transactions are the instructions to change the state and the state transition function is the mechanism that applies the transactions to the state.

Inputs: current state, block
Output: new state



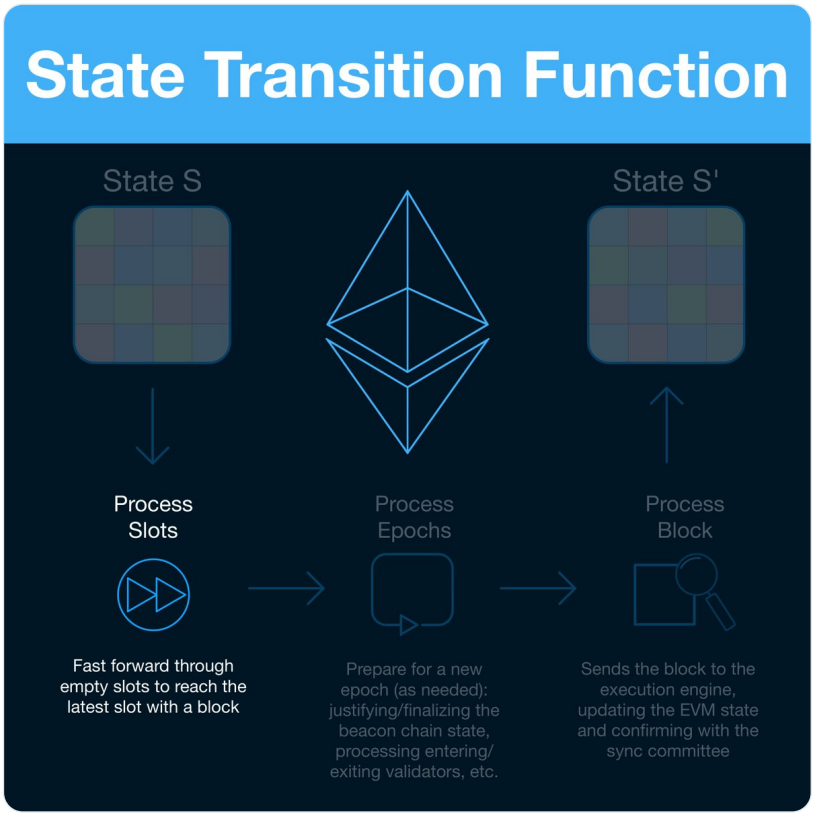
(4/17) Consensus Spec: state_transition

First, process all the empty slots that do not have a block (process_slots). Check the signature of the block (verify_block_signature) and if it's valid, process the block by executing the underlying transactions (process_block).

```
def state_transition(state: BeaconState, signed_block: SignedBeaconBlock, validate_result: bool=True) -> None:
    block = signed_block.message
    # Process slots (including those with no blocks) since block
    process_slots(state, block.slot)
    # Verify signature
    if validate_result:
        assert verify_block_signature(state, signed_block)
    # Process block
    process_block(state, block)
    # Verify state root
    if validate_result:
        assert block.state_root == hash_tree_root(state)
```

(5/17) Sometimes a slot passes without a block being proposed; maybe the proposer was offline or the network dropped the block. The state transition function moves through empty slots and triggers a change of epoch, if needed.

Inputs: current state, slot
No output



(6/17) Consensus Spec: process_slots

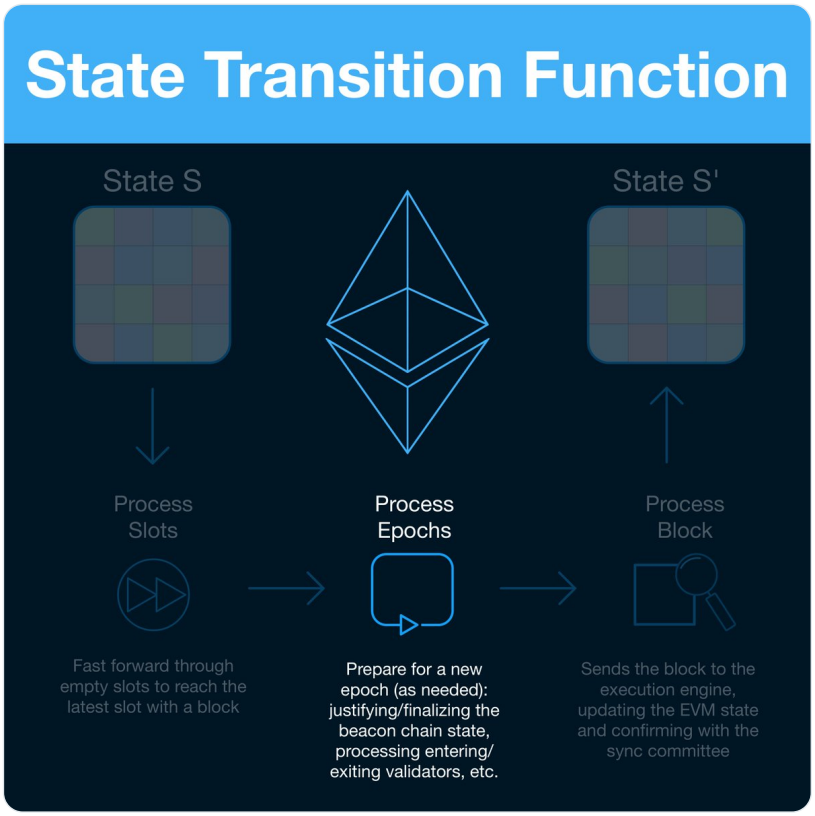
While the current slot is less than the intended slot, progress forward (process_slot). If the next slot is going to be a new epoch, execute the processes associated with consensus and prepare for the next epoch (process_epoch)

```
def process_slots(state: BeaconState, slot: Slot) -> None:
    assert state.slot < slot
    while state.slot < slot:
        process_slot(state)
        # Process epoch on the start slot of the next epoch
        if (state.slot + 1) % SLOTS_PER_EPOCH == 0:
            process_epoch(state)
        state.slot = Slot(state.slot + 1)
```

```
def process_slot(state: BeaconState) -> None:
    # Cache state root
    previous_state_root = hash_tree_root(state)
    state.state_roots[state.slot % SLOTS_PER_HISTORICAL_ROOT] = previous_state_root
    # Cache latest block header state root
    if state.latest_block_header.state_root == Bytes32():
        state.latest_block_header.state_root = previous_state_root
    # Cache block root
    previous_block_root = hash_tree_root(state.latest_block_header)
    state.block_roots[state.slot % SLOTS_PER_HISTORICAL_ROOT] = previous_block_root
```

(7/17) An epoch is 32 slots/blocks. When process_slots moves through this boundary, the validator must attend to many consensus-based and housekeeping duties.

Input: current state
No output



(8/17) Consensus Spec: process_epoch

Below is the consensus spec, as you can see it's very dense. The next tweet will provide a summary of each step.

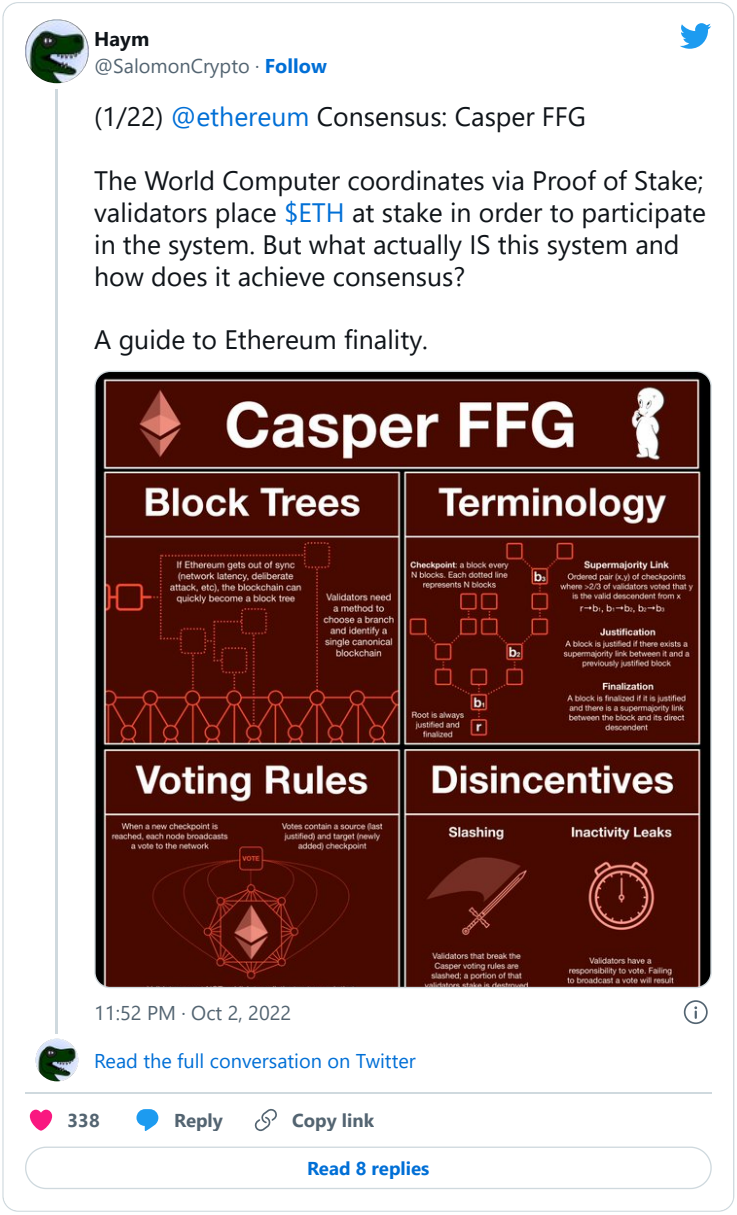
```
def process_epoch(state: BeaconState) -> None:
    process_justification_and_finalization(state)  # [Modified in Altair]
    process_inactivity_updates(state)  # [New in Altair]
    process_rewards_and_penalties(state)  # [Modified in Altair]
    process_registry_updates(state)
    process_slashings(state)  # [Modified in Altair]
    process_eth1_data_reset(state)
    process_effective_balance_updates(state)
    process_slashings_reset(state)
    process_randao_mixes_reset(state)
    process_historical_roots_update(state)
    process_participation_flag_updates(state)  # [New in Altair]
    process_sync_committee_updates(state)  # [New in Altair]
```

(9/17) Even at this level, things are very dense. I'll add a few resources for further deep dives.

process_epoch	
process_justification_and_finalization	apply the Casper FFG protocol to the blockchain (checkpoints = epochs = N = 32 blocks)
process_inactivity_updates	increase the inactivity_scores[index] for all inactive validators, decrease it for all active validators)
process_rewards_and_penalties	increase/decrease the balance of each validator based on rewards[index]/penalties[index]
process_registry_updates	updates the registry, which stores validator records. This function moves validators through the activation queue
process_slashings	second stage of slashing penalties, scaled based on the number of validators slashed during the previous period
process_eth1_data_reset	resets the view of the staking deposit contract, allowing a new round of voting for new deposits to begin
process_effective_balance_updates	effective balances are a tool uses to dampen the amount of change that happens to a validators balance. This value is used to calculate rewards and changing it is costly; effective balance uses a hysteresis scheme to only change effective balance when the actual balance change is large
process_slashings_reset	the second stage of slashing applies increasing penalties based on the number of validators slashed. This resets the data on a rolling basis (pushing out the last slashed epoch set and resetting it to make room for the next round. This is all done based on EPOCHS_PER_SLASHINGS_VECTOR)
process_randao_mixes_reset	randao_mixes is a circular list that gets updated with every block (process_randao). At the end of every epoch, the final value is copied over to become the starting value for the next epoch.
process_historical_roots_update	the eth2 chain contains a record of its own historical blocks, first caching them and then accumulating them to archival logs. If the epoch matches SLOTS_PER_HISTORICAL_ROOT // SLOTS_PER_EPOCH, then the cache is committed to the archives
process_participation_flag_updates	two epochs' worth of validator participation flags (that record validators' attestation activity) are stored. At the end of every epoch the current becomes the previous, and a new empty list becomes current.
process_sync_committee_updates	sync committees are rotated every EPOCHS_PER_SYNC_COMMITTEE_PERIOD. The next sync committee is ready and waiting so that validators can prepare in advance by subscribing to the necessary subnets. That becomes the current sync committee, and the next is calculated.

(10/17) Deep dive:

- process_justification_and_finalization
- process_inactivity_updates
- process_rewards_and_penalties
- process_slashings
- process_slashings_reset
- process_participation_flag_updates

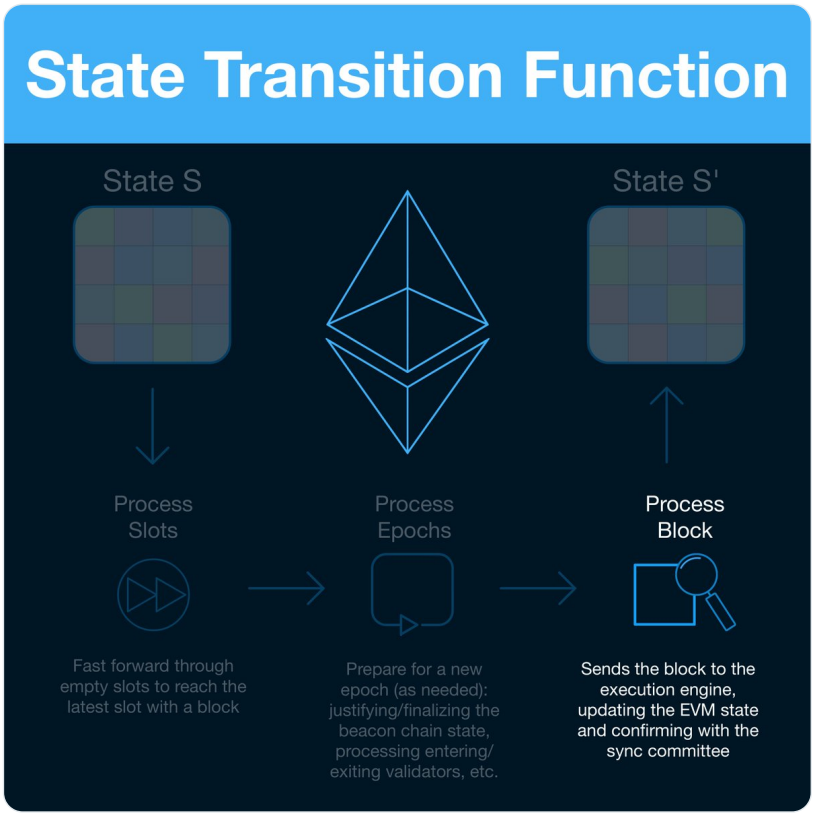


(11/17) Deep dive:

- process_randao_mixes_reset

(12/17) Finally, we get to the good part; it's time to process the block! Checking the header one more time, the validator will feed the block (and the transactions it carries) into the EVM. It ends with some consensus housekeeping.

Inputs: current state, block
No output



(13/17) Consensus Spec: process_block

At the core, the validator performs final checks (process_block_header) and feeds the block into the EVM (process_execution_payload).

```
def process_block(state: BeaconState, block: BeaconBlock) -> None:
    process_block_header(state, block)
    if is_execution_enabled(state, block.body):
        process_execution_payload(state, block.body.execution_payload, EXECUTION_ENGINE)  # [New in Bellatrix]
    process_randao(state, block.body)
    process_eth1_data(state, block.body)
    process_operations(state, block.body)
    process_sync_aggregate(state, block.body.sync_aggregate)
```

(14/17) Consensus Spec: process_execution_payload

The first chunk of this function is just more checks. The moment the validator passes the transactions to the EVM is:

execution_engine.notify_new_payload(payload)

```
def process_execution_payload(state: BeaconState, payload: ExecutionPayload, execution_engine: ExecutionEngine) -> None:
    # Verify consistency of the parent hash with respect to the previous execution payload header
    if is_merge_transition_complete(state):
        assert payload.parent_hash == state.latest_execution_payload_header.block_hash
    # Verify prev_randao
    assert payload.prev_randao == get_randao_mix(state, get_current_epoch(state))
    # Verify timestamp
    assert payload.timestamp == compute_timestamp_at_slot(state, state.slot)
    # Verify the execution payload is valid
    assert execution_engine.notify_new_payload(payload)
    # Cache execution payload header
    state.latest_execution_payload_header = ExecutionPayloadHeader(
        parent_hash=payload.parent_hash,
        fee_recipient=payload.fee_recipient,
        state_root=payload.state_root,
        receipts_root=payload.receipts_root,
        logs_bloom=payload.logs_bloom,
        prev_randao=payload.prev_randao,
        block_number=payload.block_number,
        gas_limit=payload.gas_limit,
        gas_used=payload.gas_used,
        timestamp=payload.timestamp,
        extra_data=payload.extra_data,
        base_fee_per_gas=payload.base_fee_per_gas,
        block_hash=payload.block_hash,
        transactions_root=hash_tree_root(payload.transactions),
    )
```

(15/17) Finally, the validator returns to its consensus duties. It adds to RANDAO (process_randao), votes on its view of the deposit contract (process_eth1_data), manages slashings, attestations, etc (process_operations) confirms the sync committee (process_sync_aggregate).

process_block	
process_block_header	cache the current block as the new latest block, do some sanity checks
process_execution_payload	execute the transactions in the underlying block (assert execution_engine.notify_new_payload(payload))
process_randao	verify RANDAO reveal and mix in the RANDAO reveal into the RANDAO value
process_eth1_data	the proposer submits a copy of the deposit contract; every EPOCHS_PER_ETH1_VOTING_PERIOD epochs (6.8 hours) this is copied to the beacon chain and new deposits enter the validator set.
process_operations	processes core consensus mechanics (slashings, attestations, staking deposits/withdraws)
process_sync_aggregate	verify the signatures of the sync committee and distribute rewards

(16/17) And we're done!

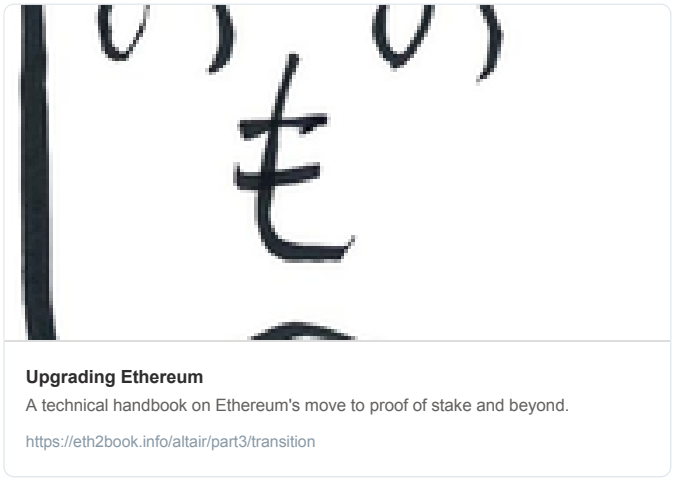
In summary:

def ethereum_state_transition:
process_slots
if new_block = new_epoch:
process_epoch
process_blocks

Repeat.

(17/17) Further Reading

@benjaminion_xyz's eth2 book is the best semi-technical manual on Ethereum I have come across. If you like my stuff, just skip to the good stuff:

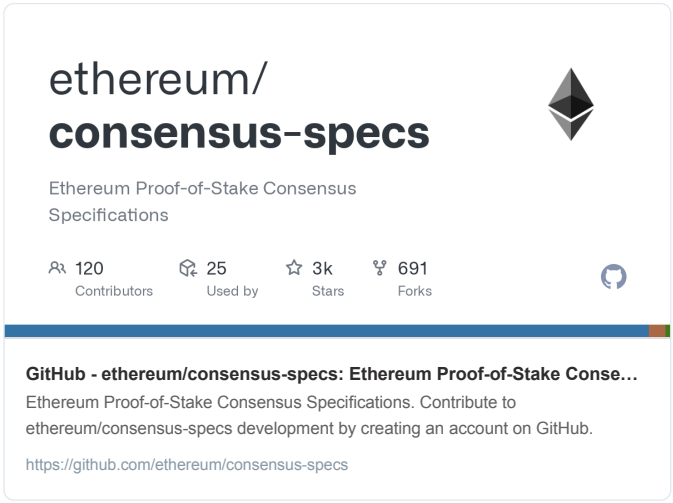


Upgrading Ethereum

A technical handbook on Ethereum's move to proof of stake and beyond.

https://eth2book.info/altair/part3/transition

All the code you be found on the Ethereum github:



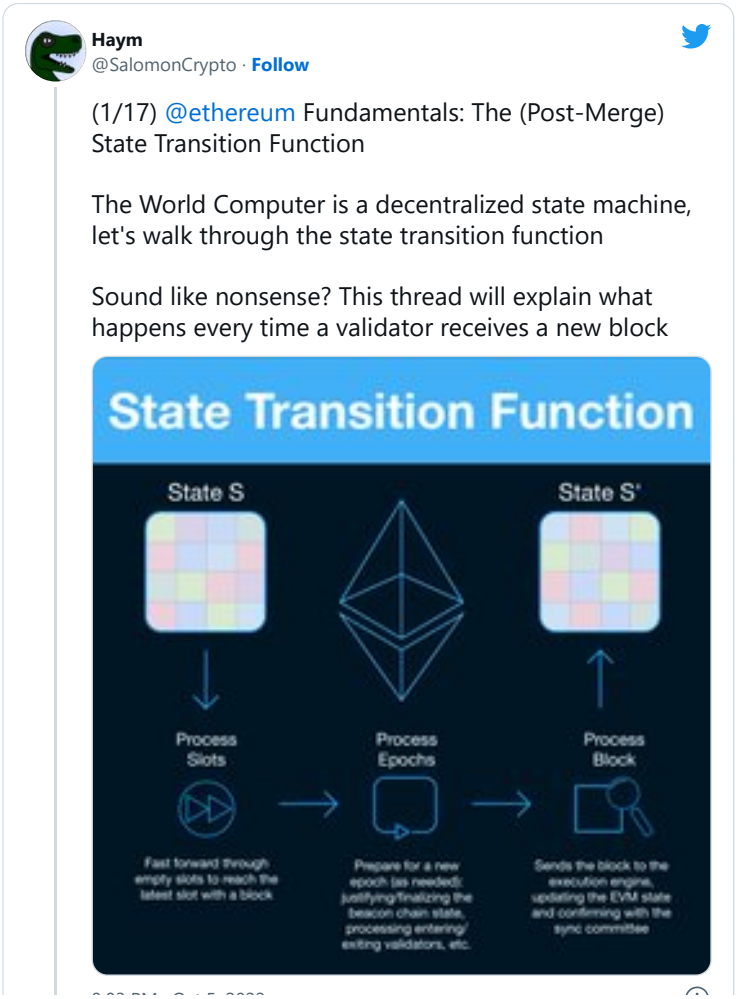
GitHub - ethereum/consensus-specs: Ethereum Proof-of-Stake Conse…

Ethereum Proof-of-Stake Consensus Specifications. Contribute to ethereum/consensus-specs development by creating an account on GitHub.

https://github.com/ethereum/consensus-specs

Like what you read? Help me spread the word by retweeting the thread (linked below).

Follow me for more explainers and as much alpha as I can possibly serve.



• • •