UPDATE 10 April

: This draft spec is now retired and won't be maintained. The research team is working on a better design

More details here.

This document is a draft spec of sharding phase 1. It is a pre-release intended for researchers and implementers. Some details will likely change.

The old phase 1 sharding doc is now deprecated. Notice some of the nomenclature has changed, e.g. we no longer have a "VMC" or "validators". Please familiarise yourself with the glossary below. Credits to @hwwhww for preparing an accompanying infographic.

Questions and feedback are most welcome, and will help us evolve this spec. Go wild, tear it apart! In a few weeks this document will be migrated to GitHub.

Phase 1 is the first non-prototype implementation in a multi-phase sharding roadmap. It is a minimal version of sharding—without enshrined execution—intended as a foundation for the first production release. We encourage independent developers worldwide to build their own clients and join the phase 1 testnet.

Proposers, collators and executors

The current roadmap distinguishes three types of sharding participants:

- Proposers

: A proposer is responsible for preparing the data that goes in a given shard. It prioritises so-called blobs (think, transactions) and assembles them into collations (think, blocks). Collations are proposed to collators for inclusion in the collation tree via an open auction amongst proposers. To prioritise blobs and maximise revenue proposers are expected to run executing nodes.

- Collators

: A collator is responsible for extending the head of the canonical chain of shards. It adjudicates on the availability of collation bodies and selects the highest-paying available collations from proposers. Collators are not required to run executing nodes. This allows for fast shuffling of the collator pool across shards, a key part of phase 1 sharding's security.

- Executors

: An executor is responsible for running the enshrined execution engine on the canonical chain of a given shard, and posting cryptoeconomic claims on state roots. This allows light-clients to guess the state root of a shard without running executing nodes. Executors are planned for phase 3 after an EVM state transition function is specified in phase 2.

Glossary

Shards

- shard

: A chain of available collations, also known as a "child shard".

- shard ID

: A 32-byte identifier for shards made of a 1-byte network ID, 30 reserved bytes set to 0x00

, and a 1-byte shard number.

- network ID

: The most significant byte of the shard ID, with most significant bit 0

for mainnet and 1

for testnet. Provisionally NETWORK_ID := 0b1000_0001

for the phase 1 testnet.

- shard number

: The least significant byte of the shard ID, ranging from 0 to SHARD_COUNT - 1

.

- shard count

: The total number of shards within a network. Provisionally SHARD_COUNT := 100

for the phase 1 testnet.

- non-transactional shard

: A shard without enshrined execution. Sharding phase 1 is a “blob shard”, i.e. a non-transactional shard with non-executed blobs (as opposed to executed transactions).

Collations

- collation

: A collation header plus a corresponding collation body. Collations are the data unit by which shards can advance every period.

- collation header

: A concise header containing collation metadata, in particular pointers to a shard, to a collation body, and to a parent collation header.

- collation body

: A file of size COLLATION_SIZE

bytes (right-padding with zero bytes if necessary) composed of 32-byte chunks. All collation bodies are valid for inclusion in the canonical chain.

- collation size

: The fixed size, denominated in bytes, of a collation body. Provisionally COLLATION_SIZE := 2 ** 20

.

- chunk

: A 32-byte-aligned collation chunk of size CHUNK_SIZE := 32

bytes.

- chunk tree

: The full binary Merkle tree with leaves the chunks of a collation body.

- chunk root

: The root of the chunk tree which identifies a collation body.

- available collation

: A collation for which the collation body identified by the chunk root is available.

- blob

: A byte-aligned piece of data delimited and ordered within a collation. Blob serialisation is TBD

, and execution will be defined in phase 2.

Collation trees

- collation tree

: The connected and directed tree of collations for every shard, as recorded by the SMC. Headers are vertices pointing to parent headers to form edges.

- genesis collation

: The first collation of a shard, fixed as a protocol checkpoint.

- canonical chain

: The longest available chain in a collation tree with respect to the genesis collation. Tie breaking of equal-length chains is done by applying the fork choice rule.

- fork choice rule

: Precedence is given to the chain with the oldest tip.

- confirmed collation

: A collation included in a canonical chain.

- orphaned collation

: A collation in the collation tree not in a canonical chain.

- head

: The tip of the canonical chain.

- header hash

: The hash of a collation header acting as its identifier.

- parent header

: A header referenced by its hash from a subsequent header. The genesis collation has no parent header.

- height

: The height of a collation in a shard’s collation tree relative to the genesis collation.

Collators

- collator

: A sharding participant responsible for collation availability and chaining.

- collator pool

: The set of all collators collectively participating in the security of all shards.

- collator subsidy

: A fixed protocol subsidy, denominated in vETH, awarded to the collators of collations included in the canonical chain of a shard. Provisionally COLLATOR_SUBSIDY := 0.001

.

- virtual ETH

: The currency unit with symbol vETH in which collators are rewarded. Virtual ETH will become transferable within a shard in phase 2, fungible across shards in phase 4, and fungible with ETH in phase 5.

- eligible collator

: The collator with the right to extend the collation tree of a given shard in the current period.

- honest majority assumption

: The fundamental security assumption for phase 1 sharding that more than half the collators are honest.

- honest collator

: A collator that faithfully follows protocol rules irrespective of financial incentives such as bribes.

- collator shuffling

: The pseudo-randomised process of selecting an eligible collator from the collator pool for every shard and at every period.

Proposers

- proposer

: A sharding participant responsible for proposing collations to eligible collators.

- proposer pool

: The set of proposers for all shards collectively prioritising blobs for inclusion in collations.

- proposal

: A proposed collation header extending the collation tree presented by a proposer to an eligible collator.

- proposer address

: The proposer’s address in a proposal against which the proposer’s signature is checked.

- proposer bid

: The amount of ETH that is burned from the proposer balance, and the equivalent of vETH that is minted in the corresponding shard and awarded to the collator.

- proposer balance

: The balance associated with a given proposer and shard from which proposer bids are deducted.

- minimum proposer balance

: The minimum balance, denominated in ETH, required for participation as a proposer on a given shard. Provisionally MIN_PROPOSER_BALANCE := 0.1

.

- proposer signature

: The proposer’s signature as part of a proposal.

- self-proposal

: A proposal for which the proposer is the eligible collator. In a self-proposal the proposer address, bid and signature are omitted.

- available proposal

: A proposal for which the corresponding collation body has been made available by the proposer.

- selected proposal

: The proposal selected by an eligible collator. At most one proposal can be selected per period and per shard.

Proposer withholding

- proposer withholding

: The situation where a proposer does not make a proposal available. Proposer withholding of selected proposals is mitigated with a proposal commitment scheme.

- proposal commitment

: A cryptographic commitment by a collator for a set of proposals, allowing proposers to make available their proposals without risk of proposal plagiarism.

- proposal plagiarism

: The situation where a collator reuses the blobs of a proposal without compensating the proposer.

- proposal commitment slashing

: The slashing condition enforcing proposal commitments.

- availability challenge

: A challenge to reveal a chunk of a collation added to a collation tree. The challenge is targeted at a collator within the windback of the challenged collation.

- availability response

: A response from the challenged collator to an availability challenge.

- availability slashing

: The slashing condition enforcing the collation availability challenge-response scheme.

Sharding Manager Contract (SMC)

- SMC

: The contract on the main chain at address SMC_ADDRESS

that manages collators, proposers and collation trees.

- main chain

: The main Ethereum blockchain hosting the SMC, also known as the "main shard", "root chain" or "root shard".

- period

: The period of time, denominated in main chain block times, during which a collation tree can be extended by one collation. Provisionally PERIOD_LENGTH := 5

, approximately 75 seconds.

- lookahead

: The advance notice eligible collators get before their assigned period.

- lookahead length

: The lookahead time, denominated in periods, for eligible collators to perform windback and select proposals. Provisionally LOOKAHEAD_LENGTH := 4

, approximately 5 minutes.

- windback

: The process of attempting to determine the head by winding back recent collation headers, checking availability of collation bodies, and applying the fork choice rule. Also known as "head fetching" or "head guessing".

- windback length

: The depth, denominated in collations, to which collators need to wind back as part of the windback process. Provisionally WINDBACK_LENGTH := 25

.

Registries

- registry

: A data structure in the SMC maintaining a set of sharding participants. Phase 1 has a collator registry and a proposer registry.

- registration

: The act of adding a registry entry by posting a deposit to become a sharding participant.

- deposit

: The fixed-size deposit, denominated in ETH, required for registration. Provisionally COLLATOR_DEPOSIT := 1000

and PROPOSER_DEPOSIT := 1

.

- deregistration

: The act of requesting the removal of a registry entry and cease being a sharding participant. The deposit is subject to a temporary lockup after deregistration.

- lockup length

: The amount of time, denominated in periods, a deposit is locked up from the time of deregistration. Provisionally COLLATOR_LOCKUP_LENGTH := 16128

, approximately two weeks, and PROPOSER_LOCKUP_LENGTH := 48

, approximately one hour.

Nomenclature cheat sheet

Ethereum 1.0

sharding phase 1

blockchain

shard

block

collation

block time

period

gas limit

collation size

transaction

blob

transaction root

chunk root

miner

collator

PoW sampling

collator shuffling

Parameters

// Shards SMC_ADDRESS := (TBD) NETWORK_ID := 0b1000_0001 SHARD_COUNT := 100 // shards PERIOD_LENGTH := 5 // block times LOOKAHEAD_LENGTH := 4 // periods WINDBACK_LENGTH := 25 // collations

// Collations COLLATION_SIZE := 2 ** 20 // bytes CHUNK_SIZE := 32 // bytes COLLATOR_SUBSIDY := 0.001 // vETH

// Registries COLLATOR_DEPOSIT := 1000 // ETH PROPOSER_DEPOSIT := 1 // ETH MIN_PROPOSER_BALANCE := 0.1 // ETH COLLATOR_LOCKUP_LENGTH := 16128 // periods PROPOSER_LOCKUP_LENGTH := 48 // periods

Collation header fields

Sharding participants have light-client access to collation headers via the HeaderAdded

logs produced by the addHeader

method. The header fields are:

shard_id uint256 // pointer to shard parent_hash bytes32 // pointer to parent header chunk_root bytes32 // pointer to collation body period int128 height int128 proposer_address address proposer_bid uint256 proposer_signature bytes

SMC storage

The SMC has the following data structures held in storage:

- Collator pool
- collator_pool: address[int128]

—array of active collator addresses

- collator_pool_len: int128

—size of the collator pool

- empty_slots_stack: int128[int128]

—stack of empty collator slot indices

- empty_slots_stack_top: int128

—top index of the stack

- collator_pool: address[int128]

—array of active collator addresses

- collator_pool_len: int128

—size of the collator pool

- empty_slots_stack: int128[int128]

—stack of empty collator slot indices

- empty_slots_stack_top: int128

—top index of the stack

- Collator registry
- collator_registry: {deregistered: int128, pool_index: int128}[address]

—collator registry (deregistered

is 0 for not yet deregistered collators)

- collator_registry: {deregistered: int128, pool_index: int128}[address]

—collator registry (deregistered

is 0 for not yet deregistered collators)

- Proposer registry
- proposer_registry: {deregistered: int128, balances: wei_value[uint256]}[address]

—proposer registry

- proposer_registry: {deregistered: int128, balances: wei_value[uint256]}[address]

—proposer registry

- Collation trees
- collation_trees: bytes32[bytes32][uint256]

—collation trees (the collation tree of a shard maps collation hashes to previous collation hashes truncated to 24 bytes packed into a bytes32

with the collation height in the last 8 bytes)

- last_update_periods: int128[uint256]

—period of last update for each shard

- collation_trees: bytes32[bytes32][uint256]

—collation trees (the collation tree of a shard maps collation hashes to previous collation hashes truncated to 24 bytes packed into a bytes32

with the collation height in the last 8 bytes)

- last_update_periods: int128[uint256]

—period of last update for each shard

- Availability challenges

- availability_challenges:

TBD

—availability challenges

- availability_challenges_len: int128

—availability challenges counter

- availability_challenges:

TBD

—availability challenges

- availability_challenges_len: int128

—availability challenges counter

SMC methods

Most of the methods detailed below issue logs. For brevity we don't emphasise them except for the HeaderAdded

log.

Registries

- register_collator() returns bool

: Adds an entry to collator_registry

, updates the collator pool (collator_pool

, collator_pool_len

, etc.), locks a deposit of size COLLATOR_DEPOSIT

, and returns True

on success. Checks:

- Deposit size

: msg.value >= COLLATOR_DEPOSIT

- Uniqueness

: collator_registry[msg.sender]

does not exist

- Deposit size

: msg.value >= COLLATOR_DEPOSIT

- Uniqueness

: collator_registry[msg.sender]

does not exist

- deregister_collator() returns bool

: Sets the deregistered

period in the collator_registry

entry, updates the collator pool (collator_pool

, collator_pool_len

, etc.), and returns True

on success. Checks:

- Authentication

: collator_registry[msg.sender]

exists

- Authentication

: collator_registry[msg.sender]

exists

- release_collator() returns bool

: Removes an entry from collator_registry

, releases the collator deposit, and returns True

on success. Checks:

- Authentication

: collator_registry[msg.sender]

exists

- Deregistered

: collator_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > collator_registry[msg.sender].deregistered + COLLATOR_LOCKUP_LENGTH

- Authentication

: collator_registry[msg.sender]

exists

- Deregistered

: collator_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > collator_registry[msg.sender].deregistered + COLLATOR_LOCKUP_LENGTH

- register_proposer() returns bool

: Equivalent of register_collator()

, without the collator pool updates.

- deregister_proposer() returns bool

: Equivalent of deregister_collator()

, without the collator pool updates.

- release_proposer() returns bool

: Equivalent of release_collator()

. WARNING

: The proposer balances need to be emptied before calling this method.

- proposer_add_balance(uint256 shard_id) returns bool

: Adds msg.value

to the balance of the proposer on shard_id

, and returns True

on success. Checks:

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Authentication

: proposer_registry[msg.sender]

exists

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Authentication

: proposer_registry[msg.sender]

exists

- proposer_withdraw_balance(uint256 shard_id) returns bool

: Withdraws the balance of a proposer on shard_id

, and returns True

on success. Checks:

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Authentication

: proposer_registry[msg.sender]

exists

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Authentication

: proposer_registry[msg.sender]

exists

Collation trees

- get_eligible_collator(uint256 shard_id, uint256 period) returns address

: Uses the blockhash at block number (period - LOOKAHEAD_LENGTH) * PERIOD_LENGTH)

and shard_id

to pseudo-randomly select an eligible collator from the collator pool, and returns the address of the eligible collator. Checks: * Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Period

: period == floor(block.number / PERIOD_LENGTH)

- Non-empty pool

: collator_pool_len > 0

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Period

: period == floor(block.number / PERIOD_LENGTH)

- Non-empty pool

: collator_pool_len > 0

- compute_header_hash(uint256 shard_id, bytes32 parent_hash, bytes32 chunk_root, uint256 period, address proposer_address, uint256 proposer_bid) returns bytes32

: Returns the header hash.

- add_header(uint256 shard_id, bytes32 parent_hash, bytes32 chunk_root, uint256 period, address proposer_address, uint256 proposer_bid, bytes proposer_signature) returns bool

: Calls compute_header_hash(...)

, extends the collation tree of shard_id

, burns the proposer_bid

from the proposer’s balance at shard_id

, issues a HeaderAdded

log, and returns True

on success. Checks: * Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Collator eligibility

: msg.sender == get_eligible_collator(shard_id, period)

- Parent exists

: collation_trees[shard_id][compute_header_hash(...)]

exists

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- Unique update

: period != last_update_periods[shard_id]

- Proposer balance

: proposer_registry[proposer_address].balances[shard_id] >= max(proposer_bid, MIN_PROPOSER_BALANCE)

- Proposer signature

: proposer_signature

matches compute_header_hash(...)

and proposer_address

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Collator eligibility

: msg.sender == get_eligible_collator(shard_id, period)

- Parent exists

: collation_trees[shard_id][compute_header_hash(...)]

exists

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- Unique update

: period != last_update_periods[shard_id]

- Proposer balance

: proposer_registry[proposer_address].balances[shard_id] >= max(proposer_bid, MIN_PROPOSER_BALANCE)

- Proposer signature

: proposer_signature

matches compute_header_hash(...)

and proposer_address

Slashing

- proposal_commitment_slashing(uint256 shard_id, bytes32 collation_hash, uint256 height, uint256 left_hash, uint256 right_hash, bytes signature) returns bool

: Slashes a collator that called add_header

with a non-committed proposal. Checks: * Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Collation tree

: collation_trees[shard_id][collation_hash]

exists

- Height

: collation_trees[shard_id][collation_hash]

matches height

- Signature

: signature

matches height

, left_hash

and right_hash

- Slashing condition

: left_hash < collation_hash

and collation_hash < right_hash

- Shard

: shard_id

against NETWORK_ID

and SHARD_COUNT

- Collation tree

: collation_trees[shard_id][collation_hash]

exists

- Height

: collation_trees[shard_id][collation_hash]

matches height

- Signature

: signature

matches height

, left_hash

and right_hash

- Slashing condition

: left_hash < collation_hash

and collation_hash < right_hash

- availability_challenge()

: TBD

- availability_response()

: TBD

- availability_slashing()

: TBD

Roadmap

The roadmap is an active area of research. The outline below is only intended to provide flavour.

- Phase 1

: Basic sharding without EVM * Blob shard without transactions

- Proposers
- Proposal commitments
- Collation availability challenges
- Blob shard without transactions
- Proposers
- Proposal commitments
- Collation availability challenges
- Phase 2

: EVM state transition function * Full nodes only

- Asynchronous cross-contract calls only
- Account abstraction
- eWASM
- Archive accumulators
- Storage rent
- Full nodes only
- Asynchronous cross-contract calls only
- Account abstraction
- eWASM
- Archive accumulators
- Storage rent
- Phase 3

: Light client state protocol * Executors

- Stateless clients
- Executors

- Stateless clients
- Phase 4

: Cross-shard transactions * Internally-synchronous zones

- Internally-synchronous zones
- Phase 5

: Tight coupling with main chain security * Data availability proofs

- Casper integration
- Internally fork-free sharding
- Manager shard
- Data availability proofs
- Casper integration
- Internally fork-free sharding
- Manager shard
- Phase 6

: Super-quadratic sharding * Load balancing

- Load balancing