Ethereum Sharding General Introduction

Ethereum Research

Hsiao-Wei Wang and Karl Floersch

2018 March 19th

Sharding

Sharding

Scaling Solution!

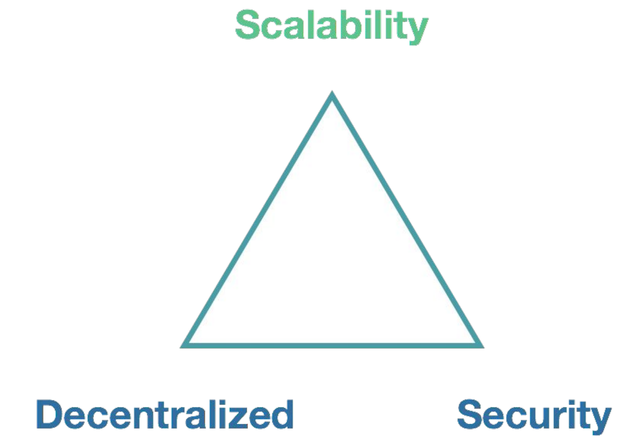
Sharding

A Secure and Decentralized **Scaling** Solution!

Sharding

A Secure and Decentralized **Scaling** Solution!

How secure are you talking about?



Sharding

A Secure and Decentralized **Scaling** Solution!

How secure are you talking about?

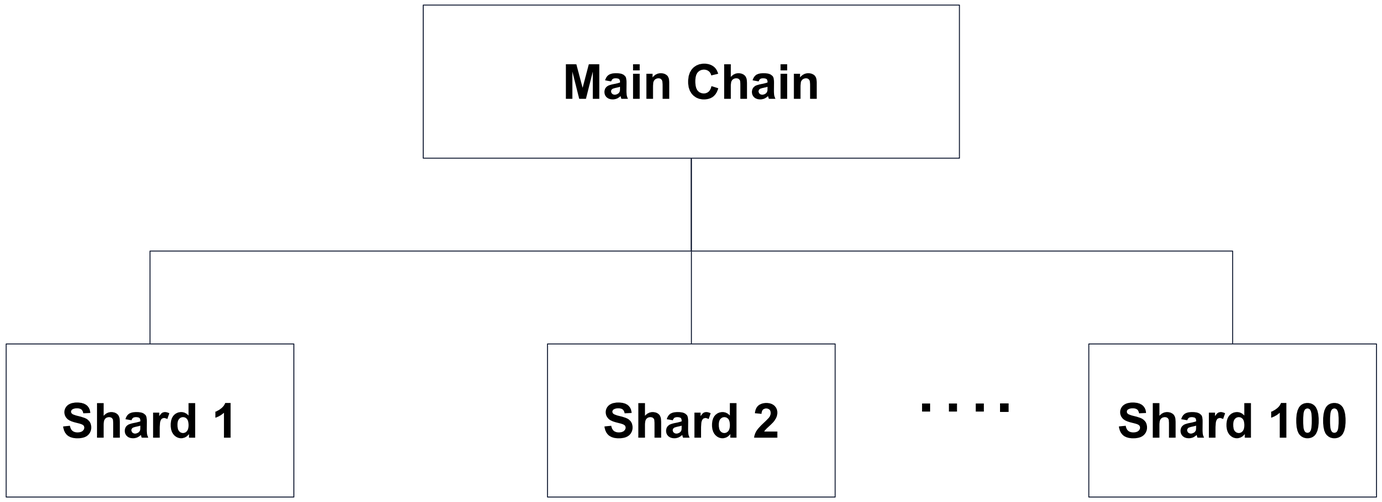
Spoiler! Day 3 - Security Models Mechanism Design

Main Chain

The main Ethereum blockchain

Shard Chain

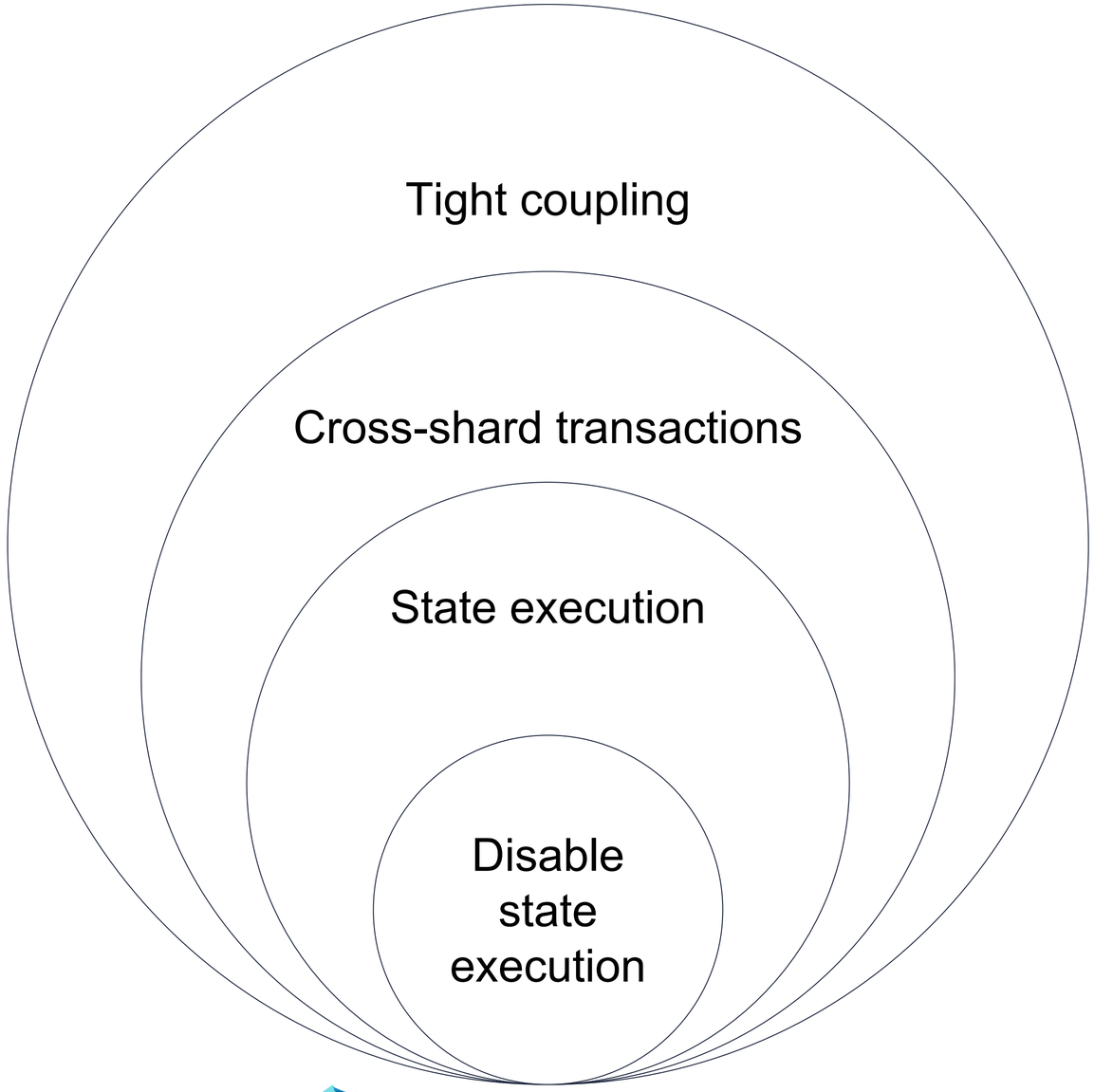
- **Create many new shard chains**
- **Each shard chain is a new galaxy**


Main Chain
Shard 1
Shard 2
. . . .
Shard 100

Scaling Goal

1. Scaling: The VISA level transaction rate
2. Usability:
    a. Cross-contracts transaction
    b. Cross-shards transaction
3. Tight coupling

Compatibilities


Tight coupling
Cross-shard transactions
State execution
Disable state execution

Sharding Phase 1

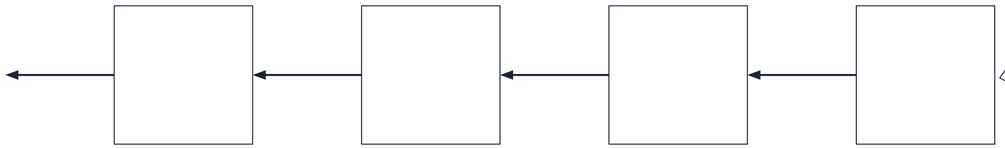
v2

Non-transactional Shard

https://ethresear.ch/t/1407

Collation

like block!



Collation

Collation Header

`shard_id:` `uint256`
the shard ID of the shard; the most significant byte is a “network ID” and the least significant byte goes from `0` to `SHARD_COUNT - 1`

`parent_hash:` `bytes32`
the hash of the parent collation

`chunk_root:` `bytes32`
the root of the chunks tree which identifies a collation body. Execution engines can authenticate blobs with Merkle paths to the chunks root

`period:` `uint256`
the period number in which this collation expects to be included

`proposer_address:` `address`
address of the collation proposer

`proposer_bid:` `uint256`
the reward from the proposer to the eligible collator for a winning proposal

`proposer_signature:` `bytes`
the proposer’s signature as part of a proposal

Collation Body

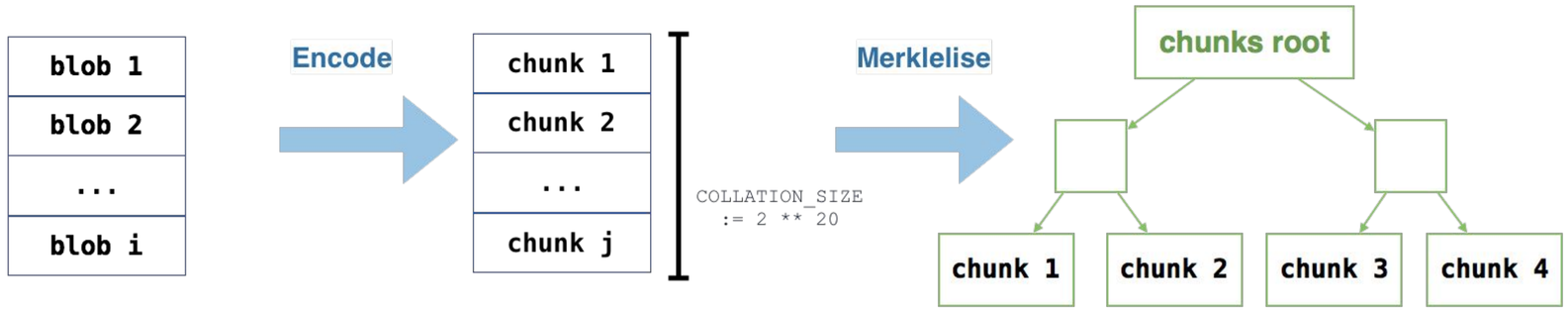
`collation_body:` `bytes`
32-byte chunks serialising a list of blobs

Blobs and Chunks

blob
A byte-aligned piece of data delimited and ordered within a collation.

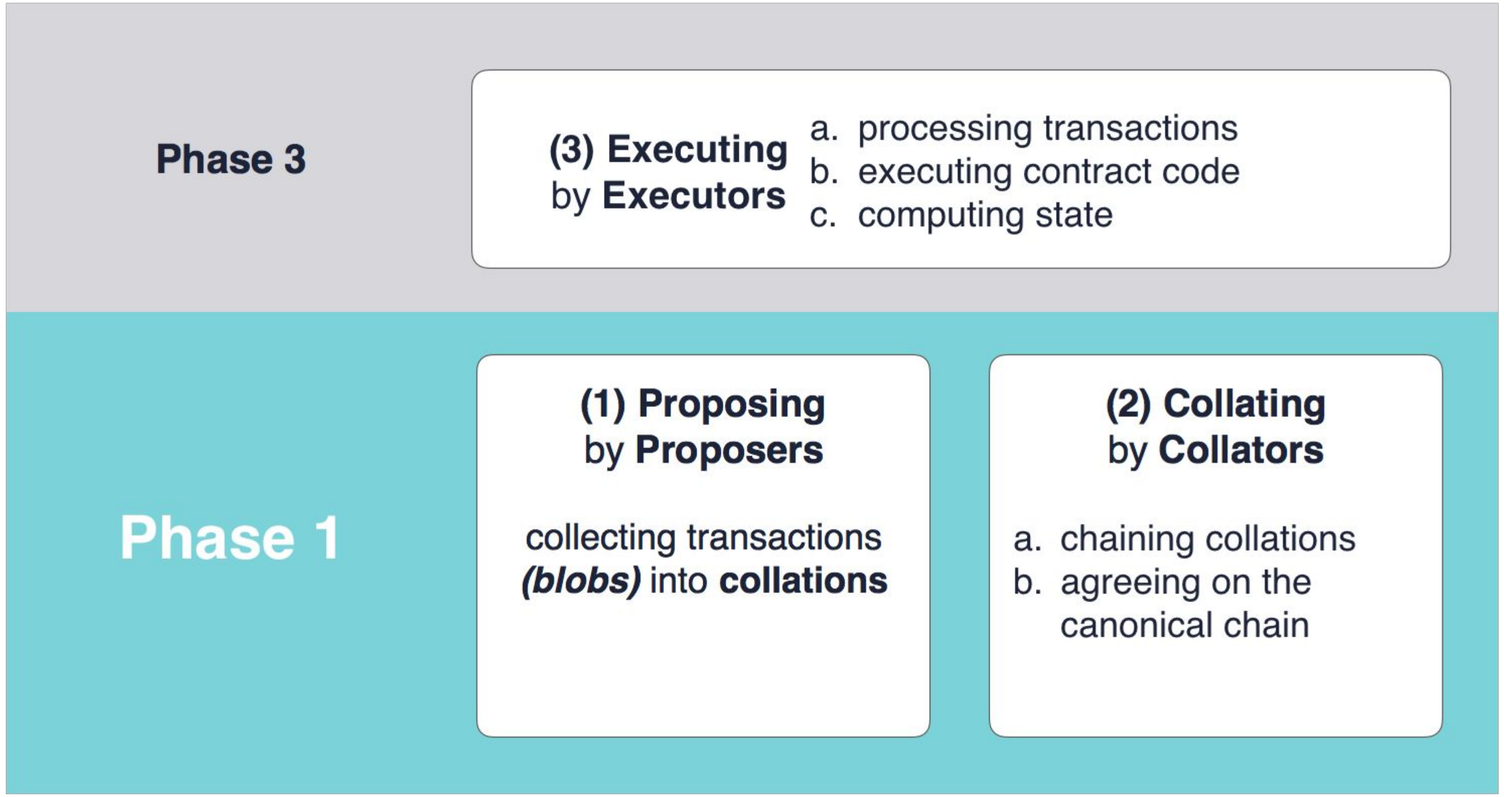
chunk
A 32-byte-aligned collation chunk of size 32 bytes, used to delimit and serialise blobs.

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



Spoiler! Day 1 Proposer / Collator Separation

Two Layers and Three Processes



Spoiler! Day 1 Proposer / Collator Separation

Proposer

1. Anyone could be a proposer
2. Maintains transaction pools
3. Collects the transactions to prepares the proposal

   (collatinon header)
4. Publishes/Reveals the collation body

Spoiler! Day 1 Proposer / Collator Separation

Collator

1. Is pseudo-randomly sampled as the eligible collator of “the specific shard and the specific period” from the collator pool of all shards
2. Collates the proposal to build the collation

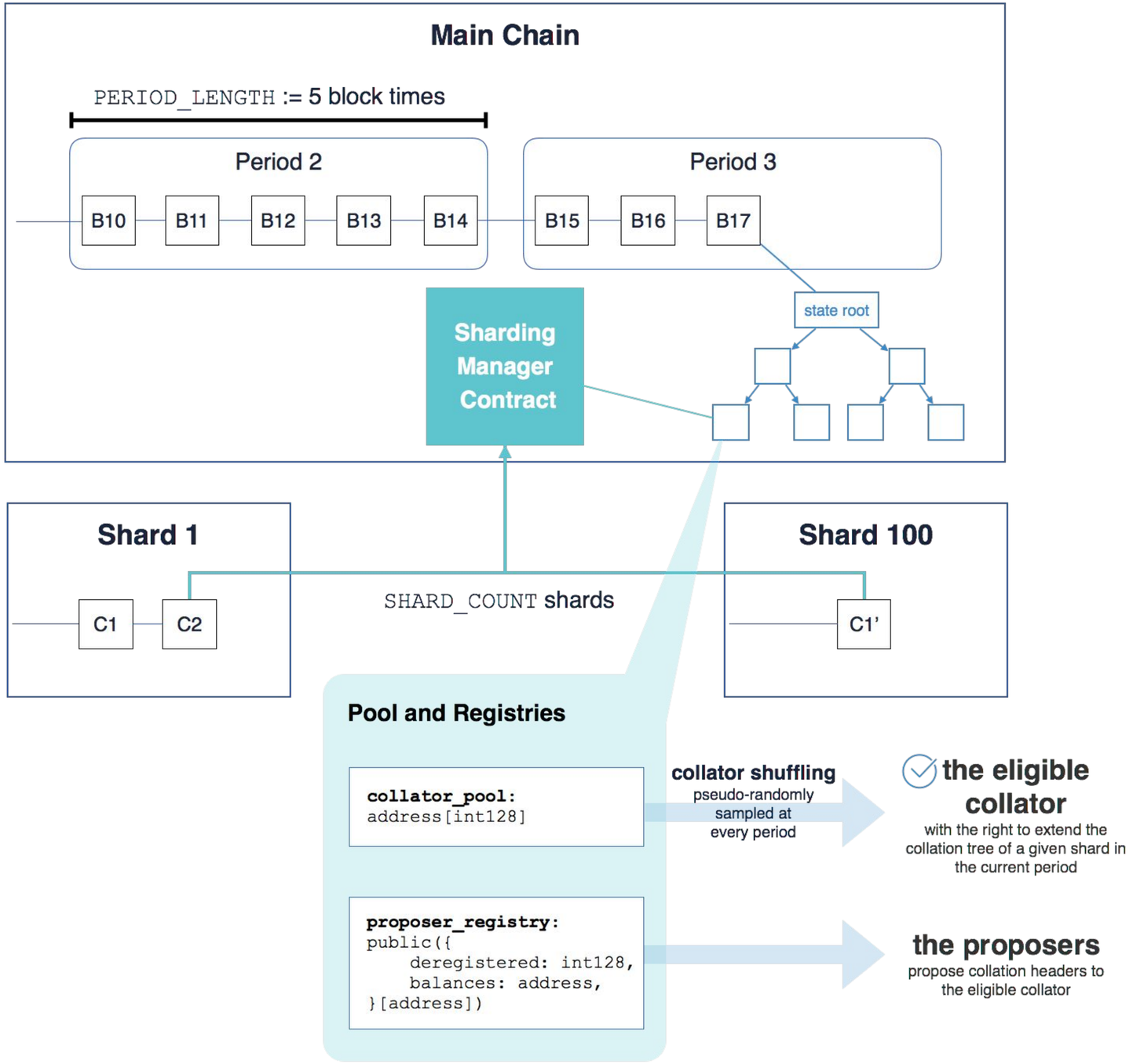
Spoiler! Day 1
Execution

Executor

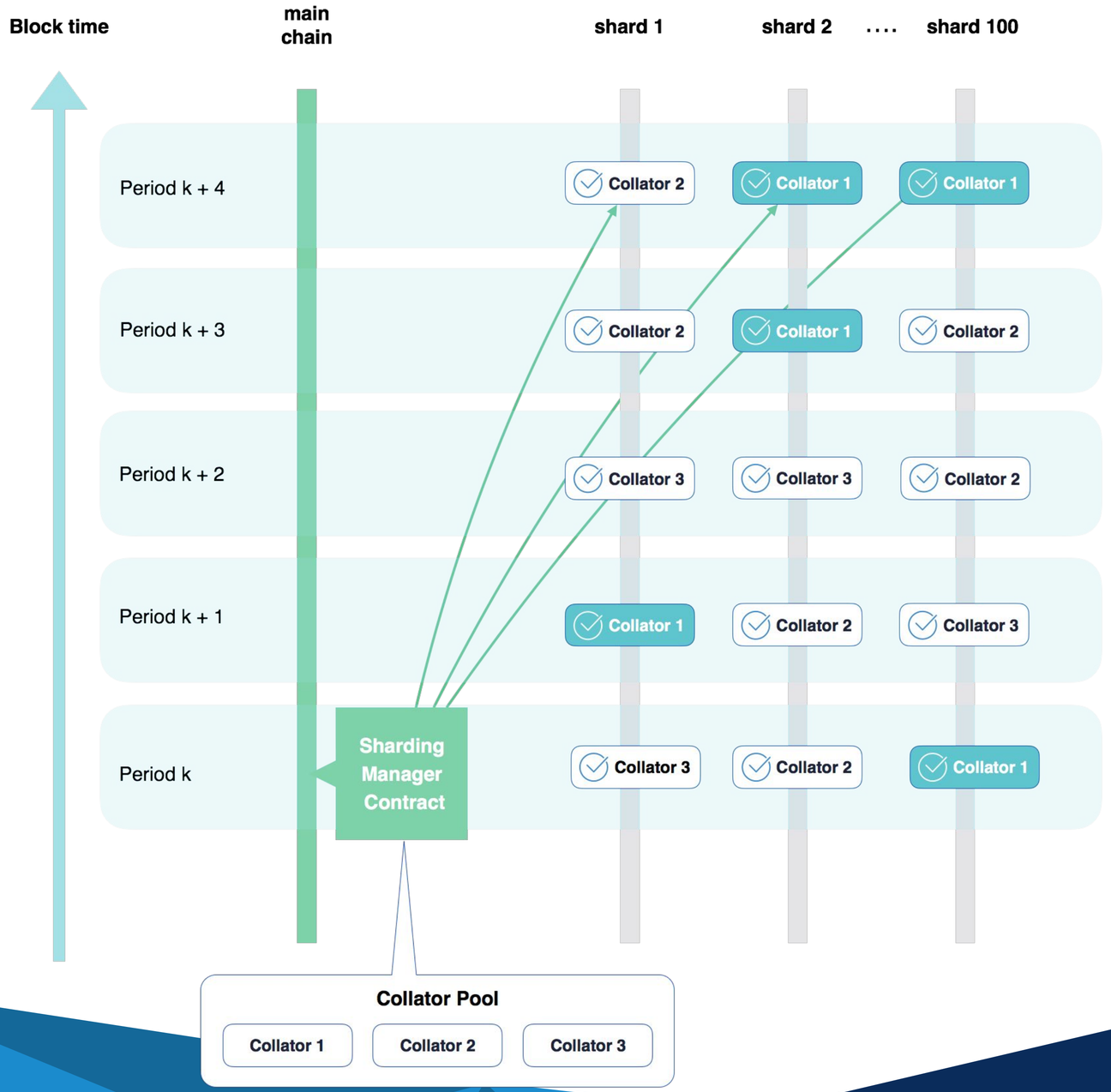
1. Executes the state transition function
2. Proposers are supposed to be the executors too to have to abilities to know the consuming gas of transactions and select transactions with high fee

Spoiler! Day 1
Sharding Manager Contract

Sharding Manager Contract (SMC)



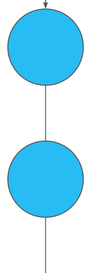
Lookahead


Block time
main chain
shard 1
shard 2
....
shard 100
Period k + 4
Period k + 3
Period k + 2
Period k + 1
Period k
Collator 2
Collator 1
Collator 1
Collator 2
Collator 1
Collator 2
Collator 3
Collator 3
Collator 2
Collator 1
Collator 2
Collator 3
Collator 3
Collator 2
Collator 1
Sharding Manager Contract
Collator Pool
Collator 1
Collator 2
Collator 3

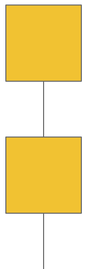
Karl Slides

Sharding Phase 1

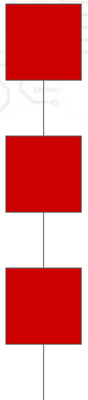
Root chain



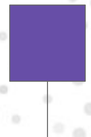
Shard 1



Shard 2

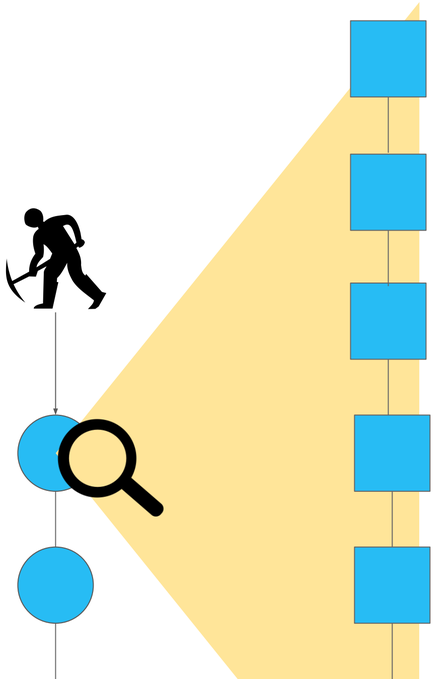


Shard 3



Sharding Phase 1

Root chain



> Each circle represents **1** period, which is **5** blocks

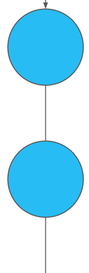
Shard 1

Shard 2

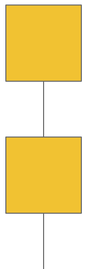
Shard 3

Sharding Phase 1

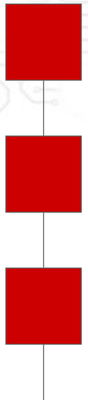
Root chain



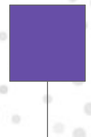
Shard 1

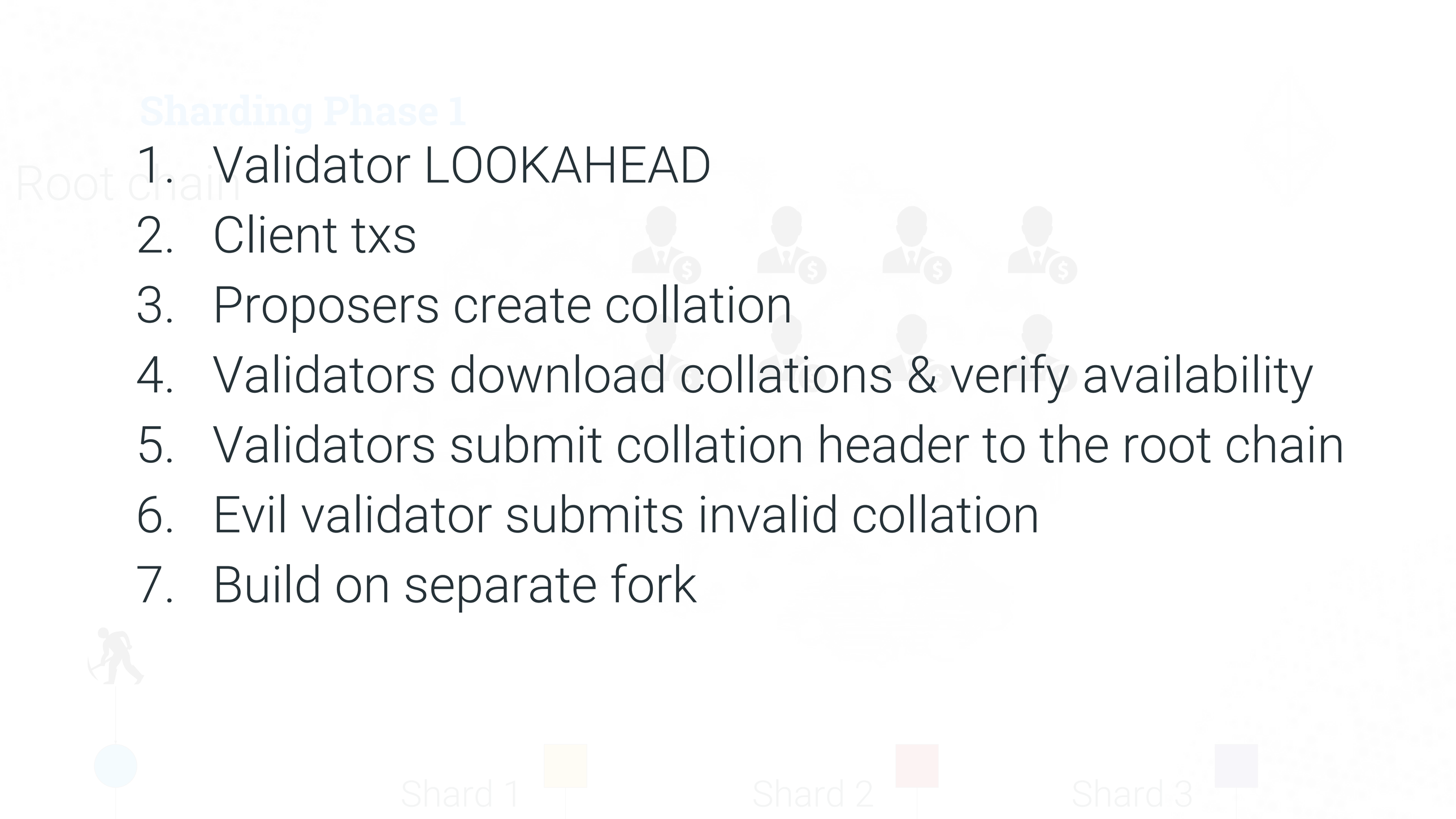


Shard 2



Shard 3





1. Validator LOOKAHEAD
2. Client txs
3. Proposers create collation
4. Validators download collations & verify availability
5. Validators submit collation header to the root chain
6. Evil validator submits invalid collation
7. Build on separate fork

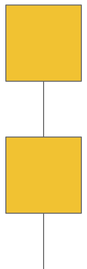
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1

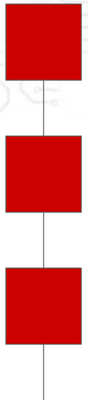
Root chain

LOOKAHEAD_PERIODS = 4

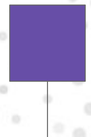
> ! LOOKAHEAD allows validators to check which shards they will be validating in advance



Shard 1



Shard 2

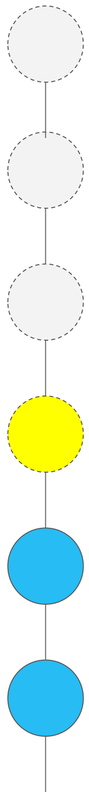


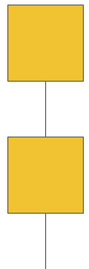
Shard 3

Sharding Phase 1

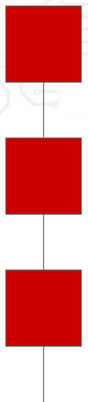
Root chain

LOOKAHEAD_PERIODS = 4

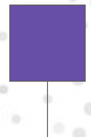




Shard 1



Shard 2

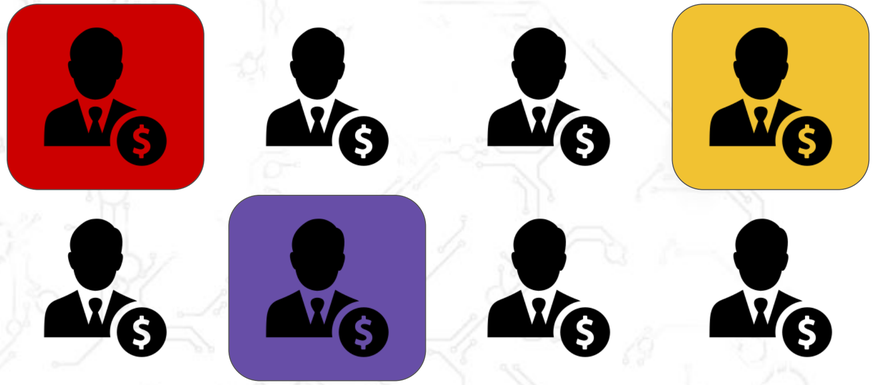


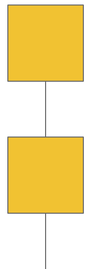
Shard 3

Sharding Phase 1

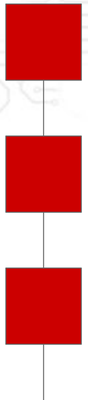
Root chain

LOOKAHEAD_PERIODS = 4

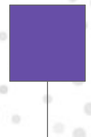




Shard 1



Shard 2

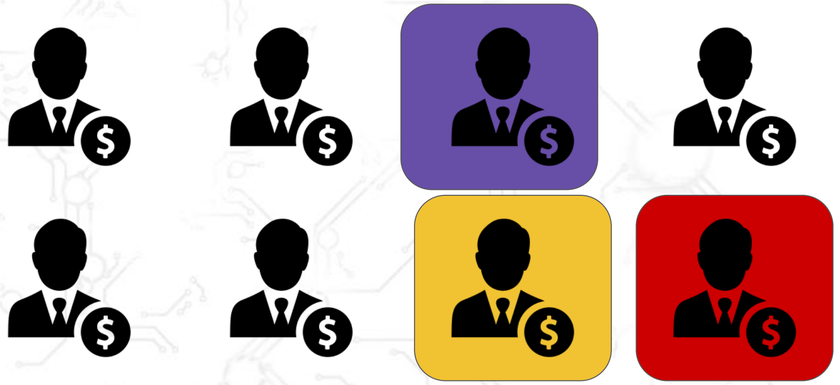


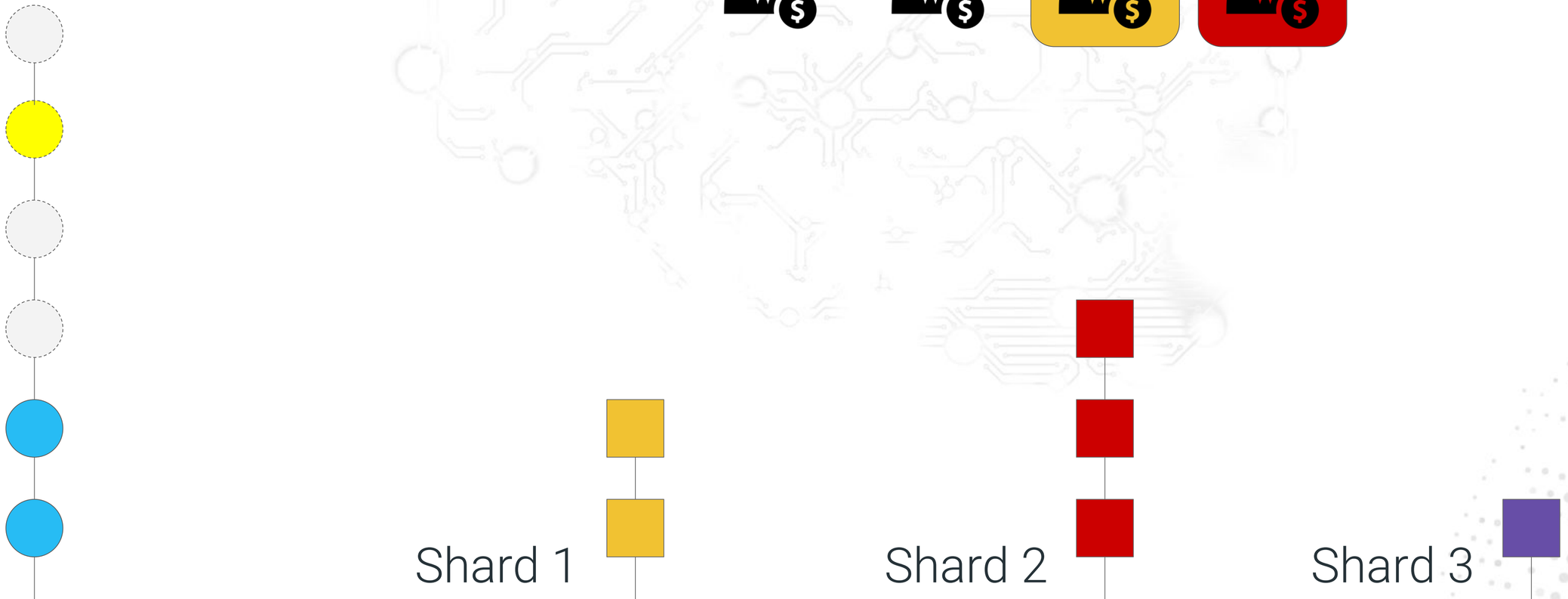
Shard 3

Sharding Phase 1

Root chain

LOOKAHEAD_PERIODS = 4

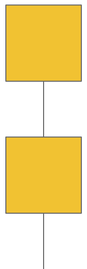




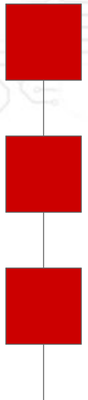
Sharding Phase 1

Root chain

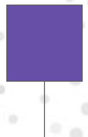
LOOKAHEAD_PERIODS = 4



Shard 1



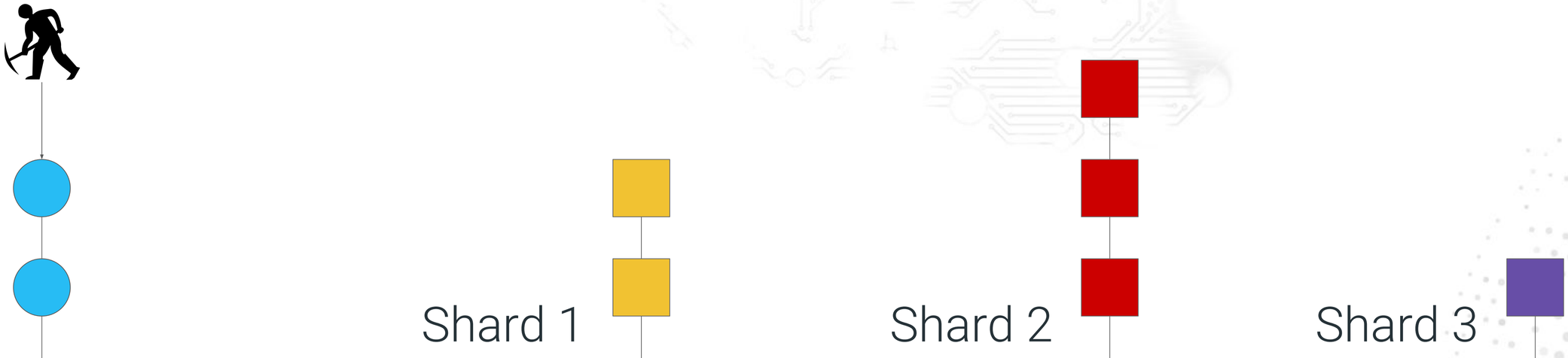
Shard 2



Shard 3

Sharding Phase 1

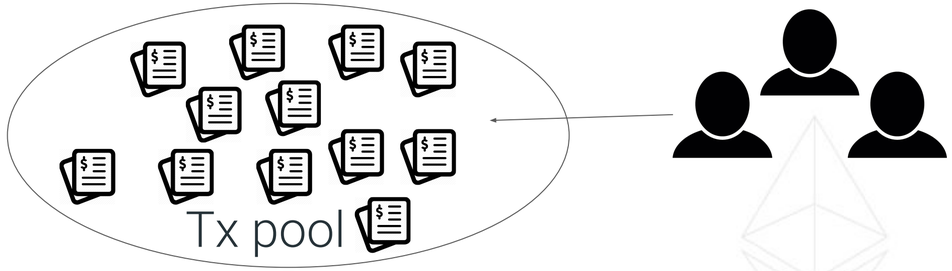
Root chain

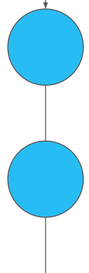


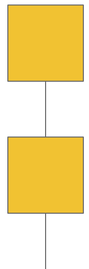
Sharding Phase 1

1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

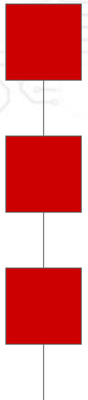
Sharding Phase 1



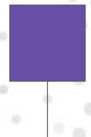




Shard 1



Shard 2

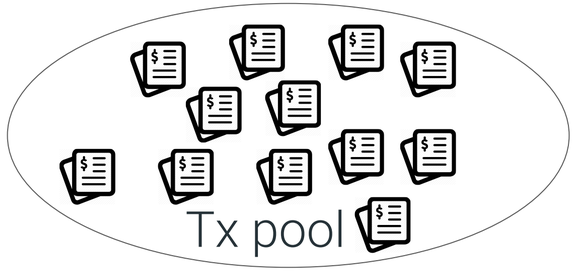


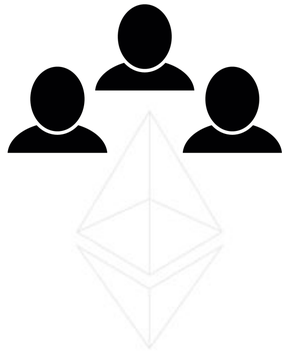
Shard 3

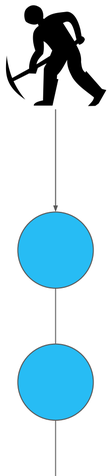
Sharding Phase 1

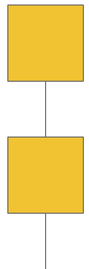
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1

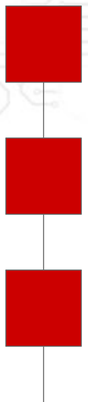




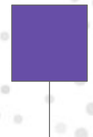




Shard 1

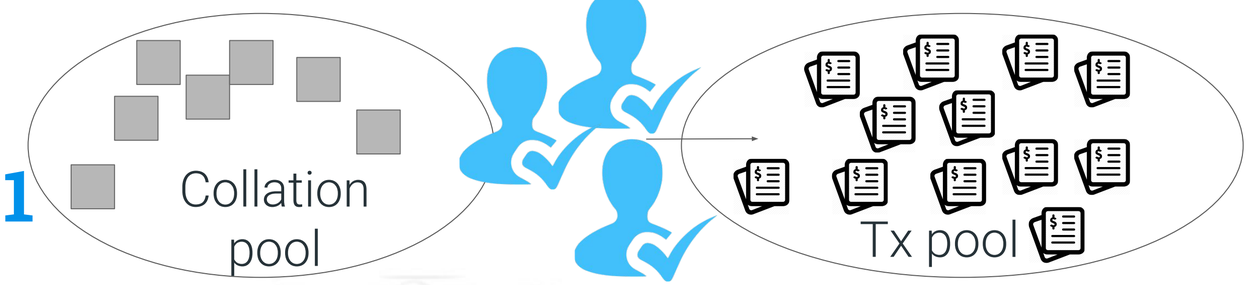


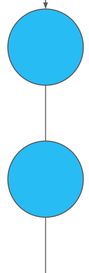
Shard 2

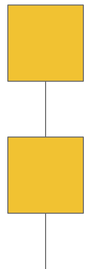


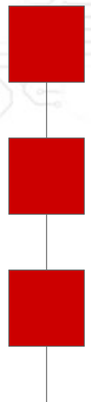
Shard 3

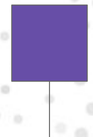
Sharding Phase 1









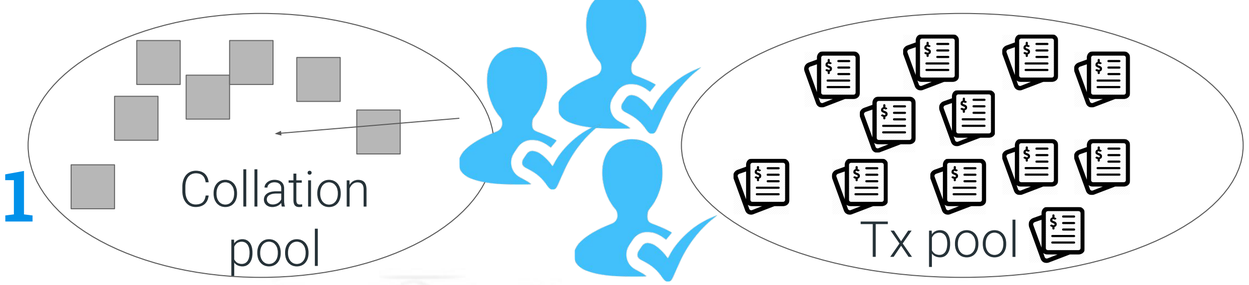


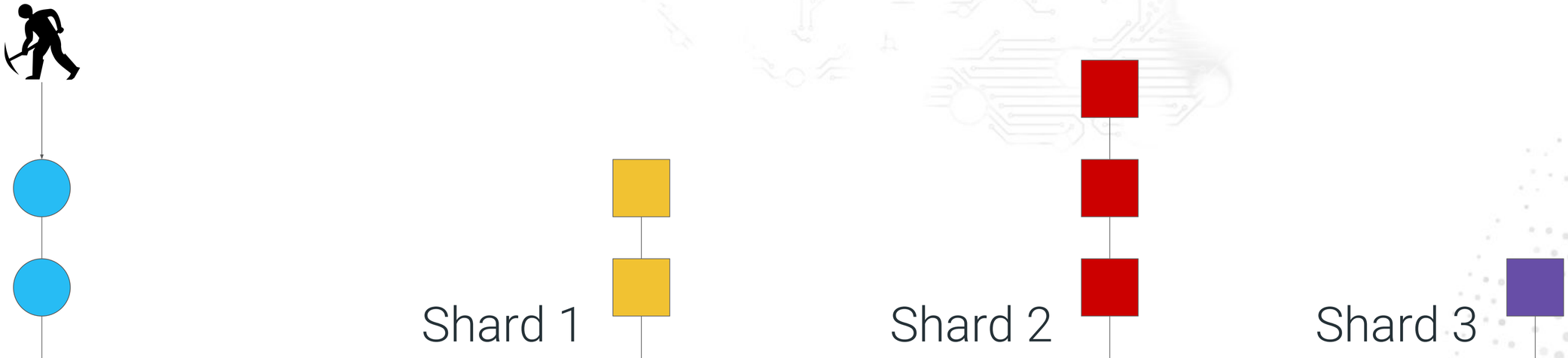
Shard 1

Shard 2

Shard 3

Sharding Phase 1

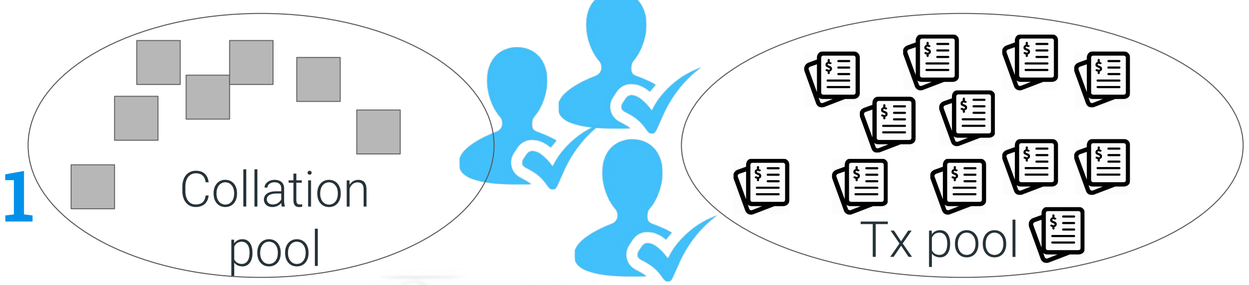


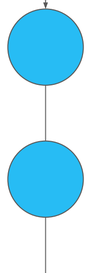


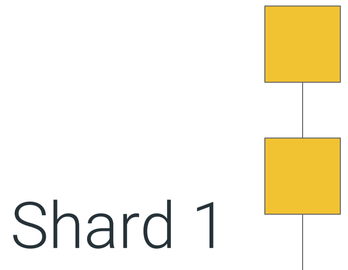
Sharding Phase 1

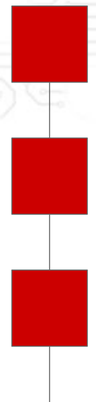
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1

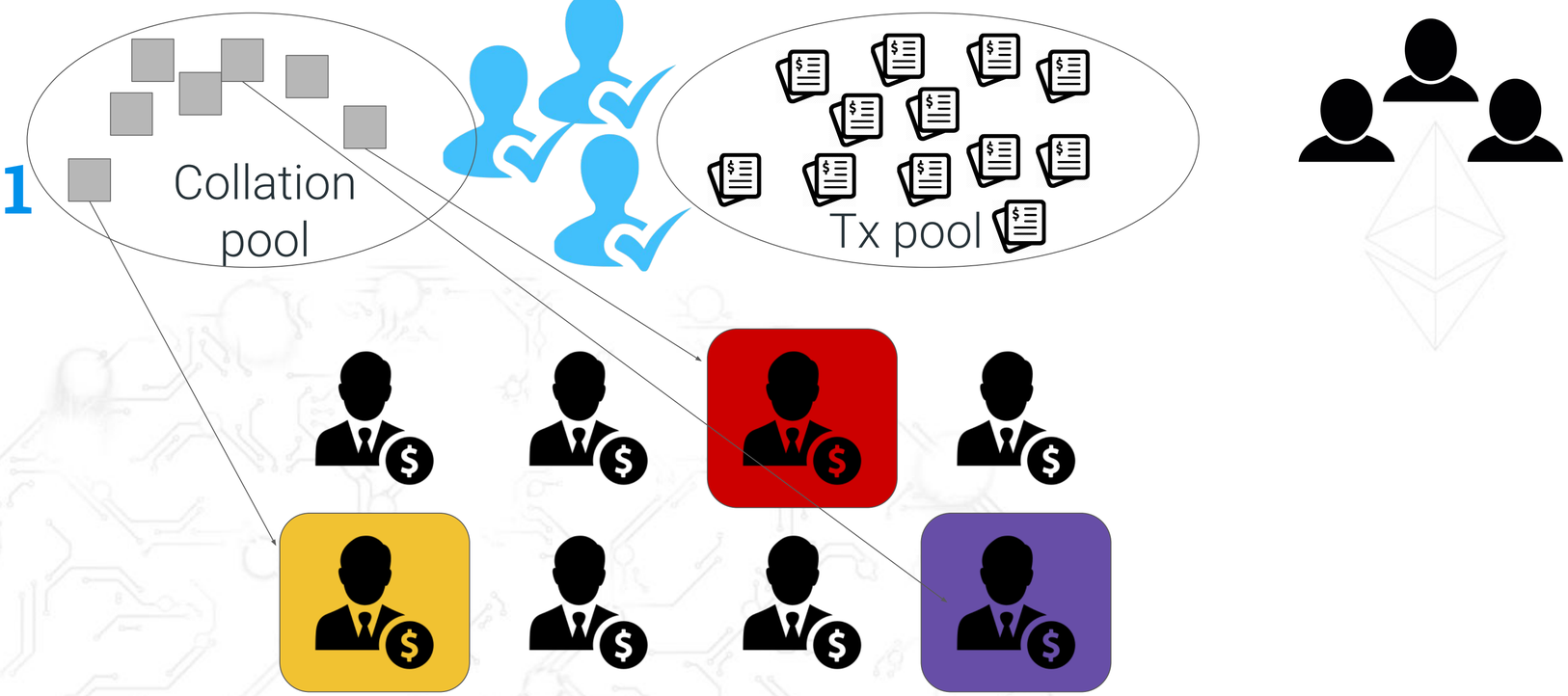


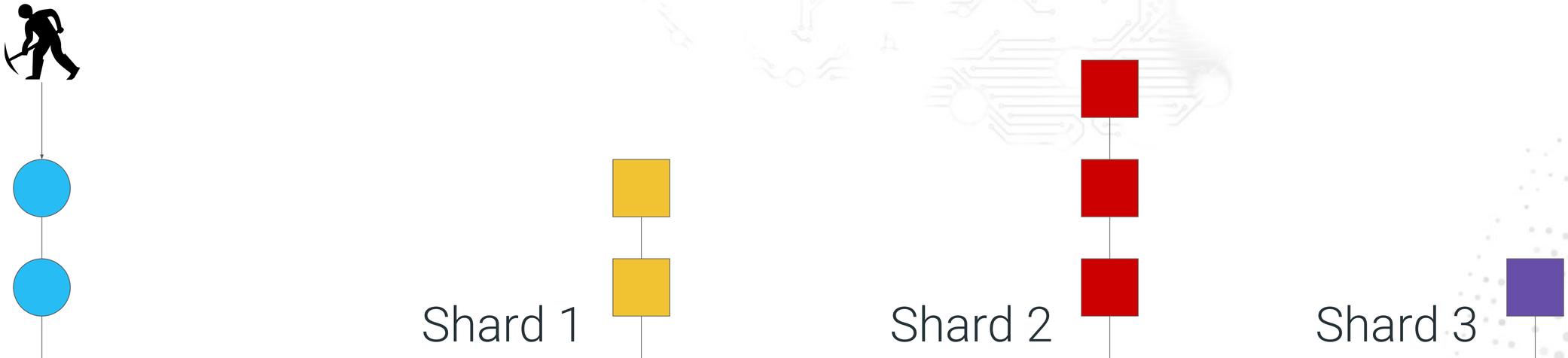






Sharding Phase 1

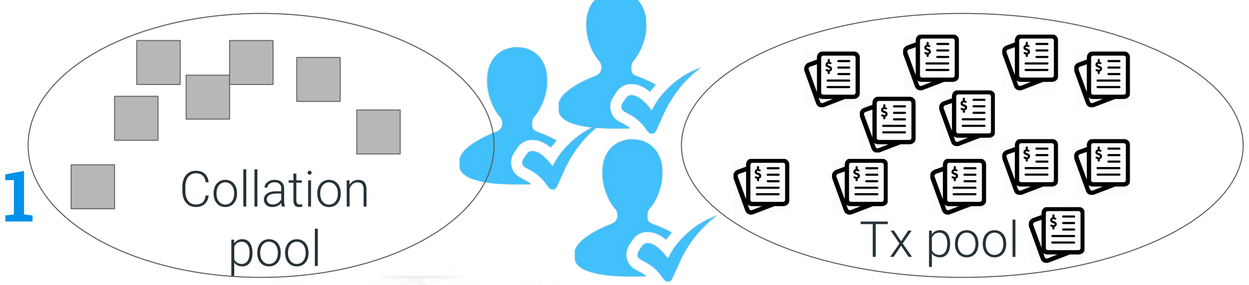




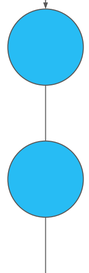
Sharding Phase 1

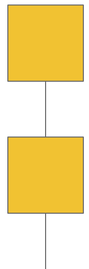
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1

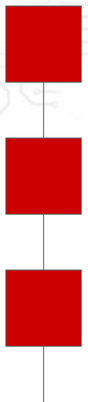


Root chain

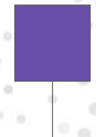




Shard 1

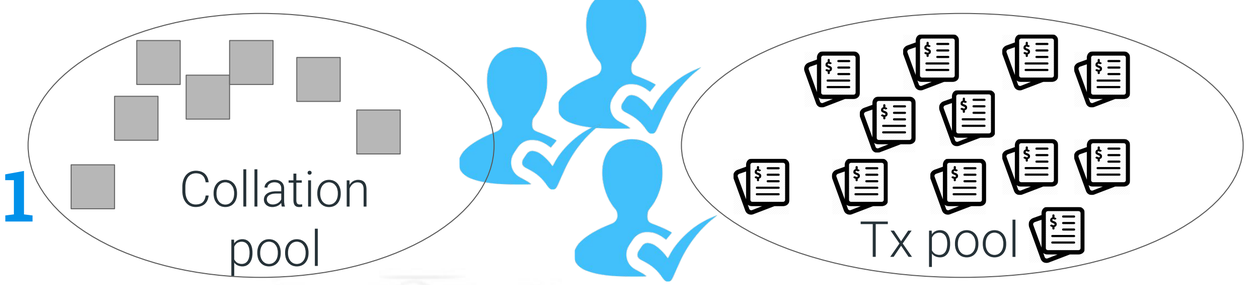


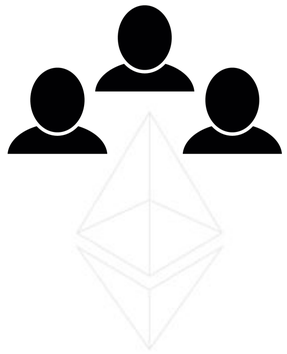
Shard 2



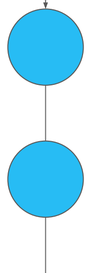
Shard 3

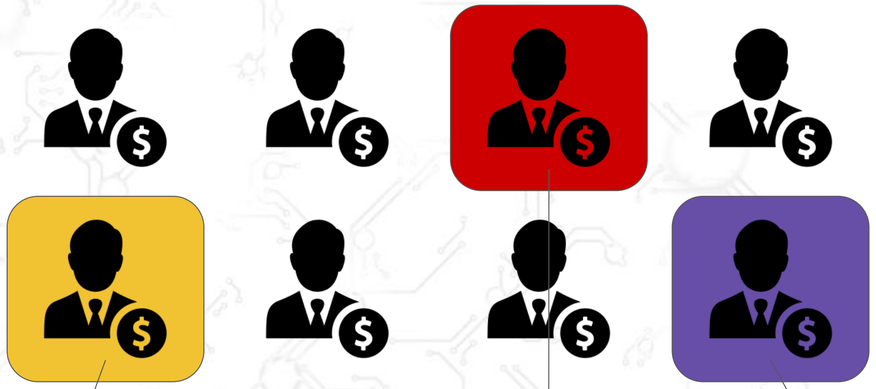
Sharding Phase 1





Root chain

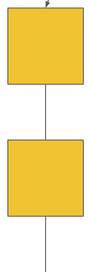


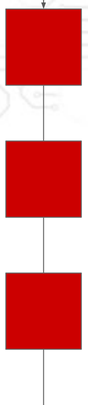


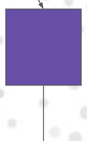
Verify

Verify

Verify





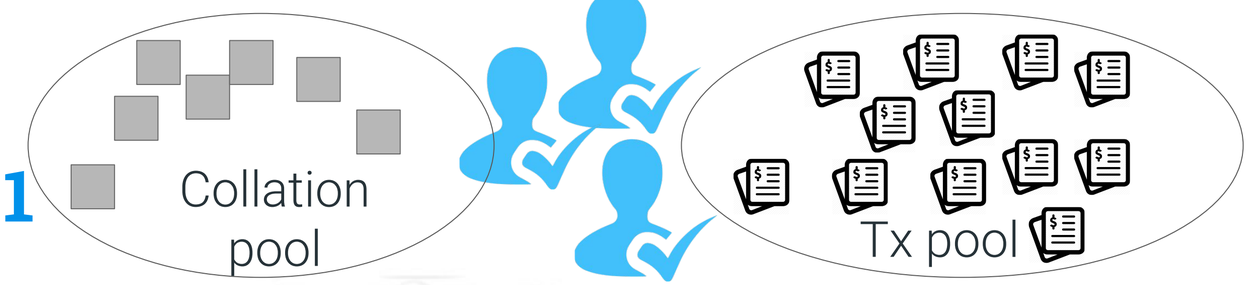


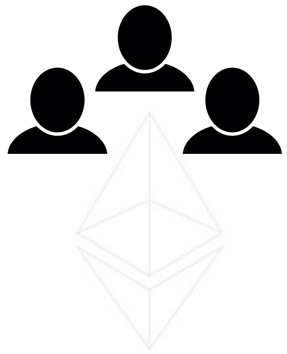
Shard 1

Shard 2

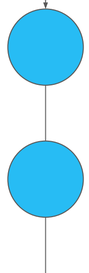
Shard 3

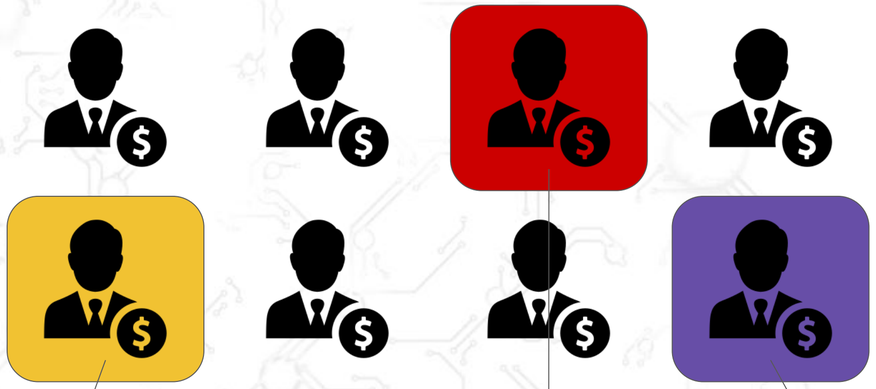
Sharding Phase 1





Root chain

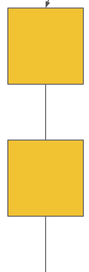


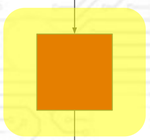


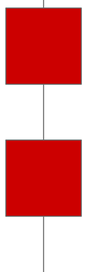
Verify

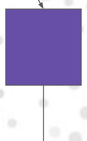
Verify

Verify







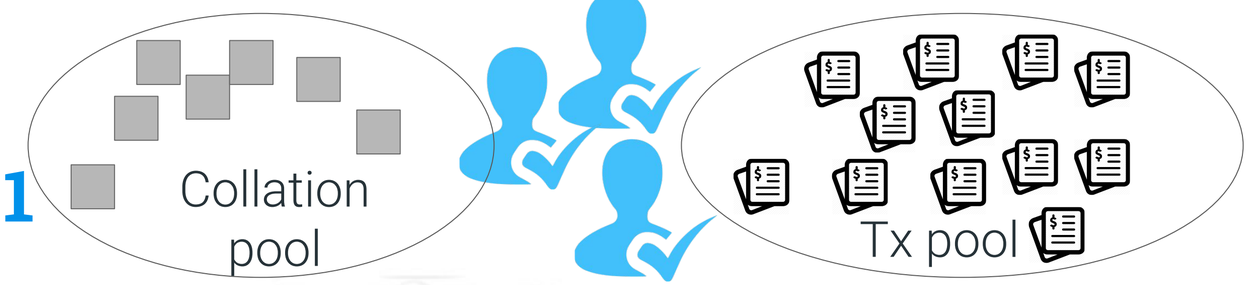


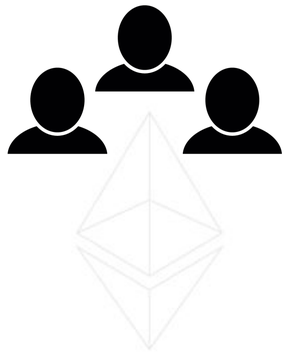
Shard 1

Shard 2

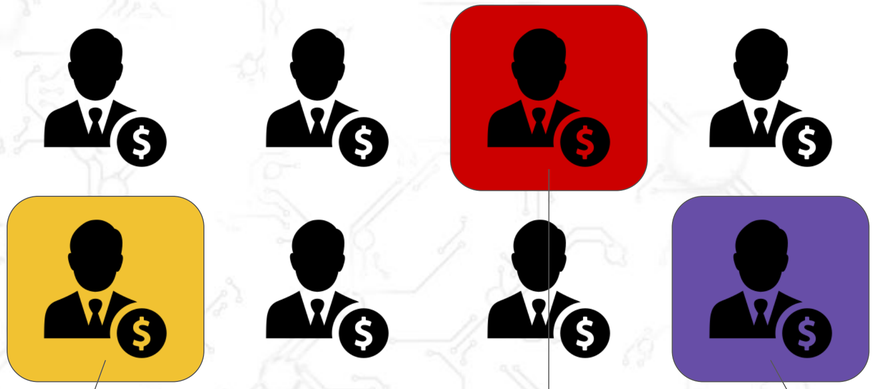
Shard 3

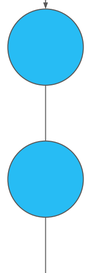
Sharding Phase 1

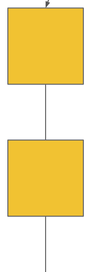


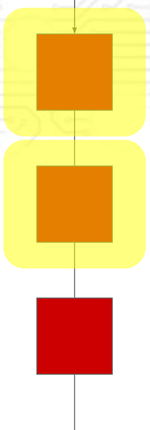


Root chain









Verify

Verify

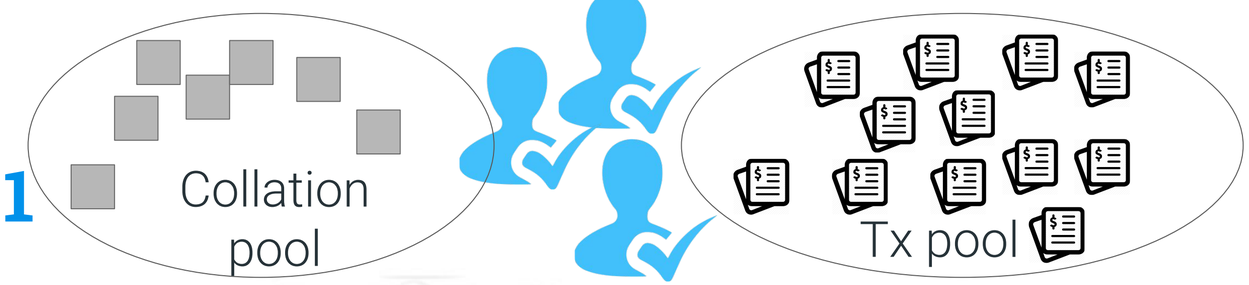
Verify

Shard 1

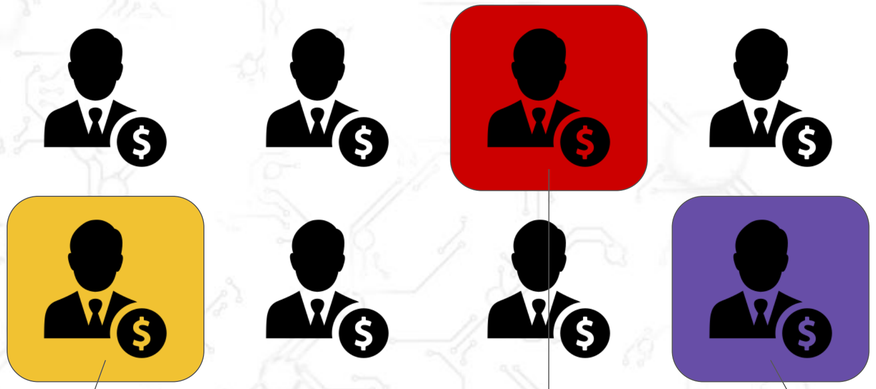
Shard 2

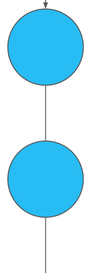
Shard 3

Sharding Phase 1



Root chain

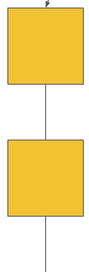


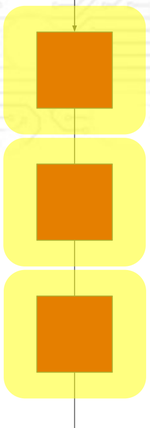


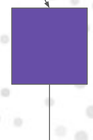
Verify

Verify

Verify





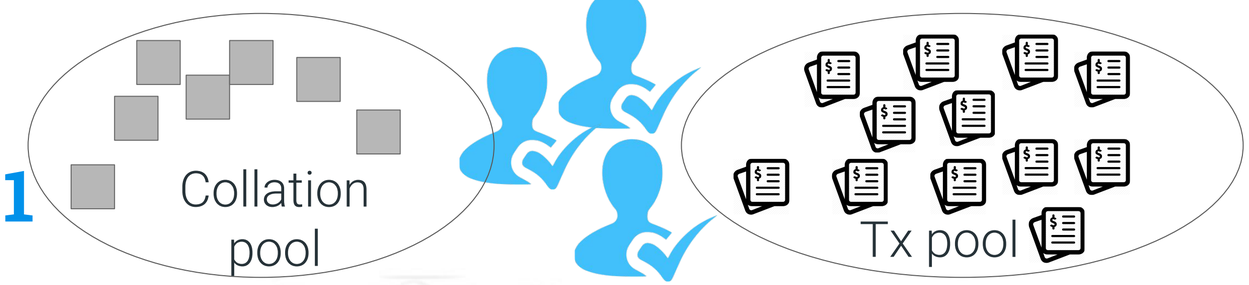


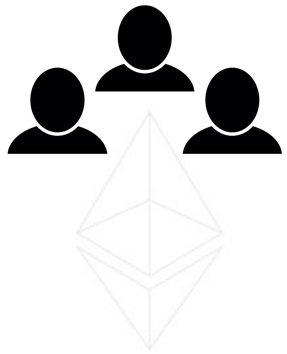
Shard 1

Shard 2

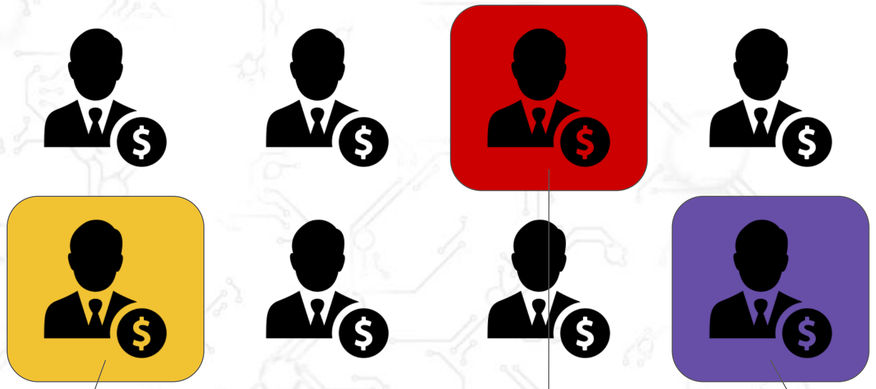
Shard 3

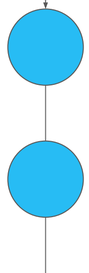
Sharding Phase 1





Root chain



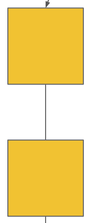


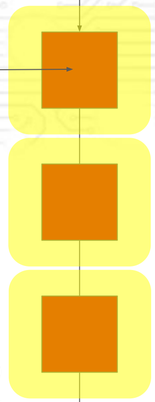
Verify

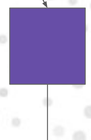
Verify

Verify

Choose head







Shard 1

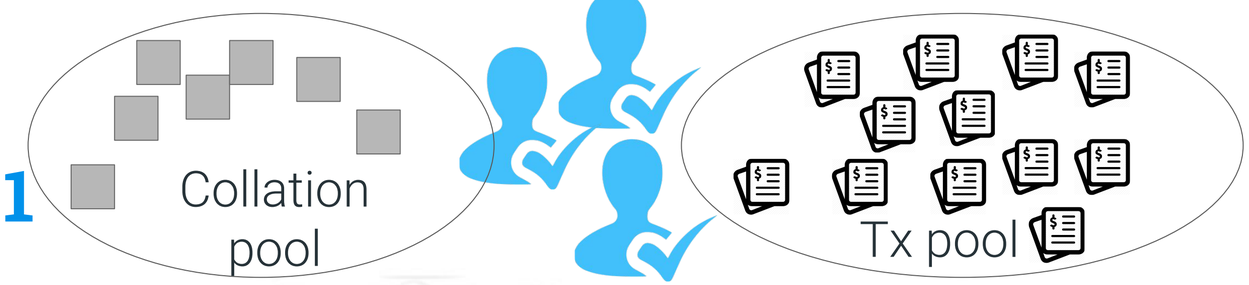
Shard 2

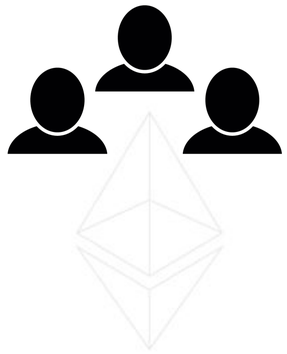
Shard 3

Sharding Phase 1

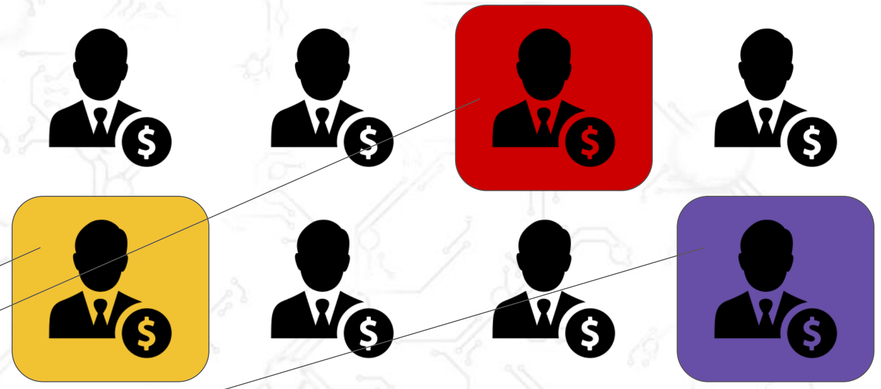
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

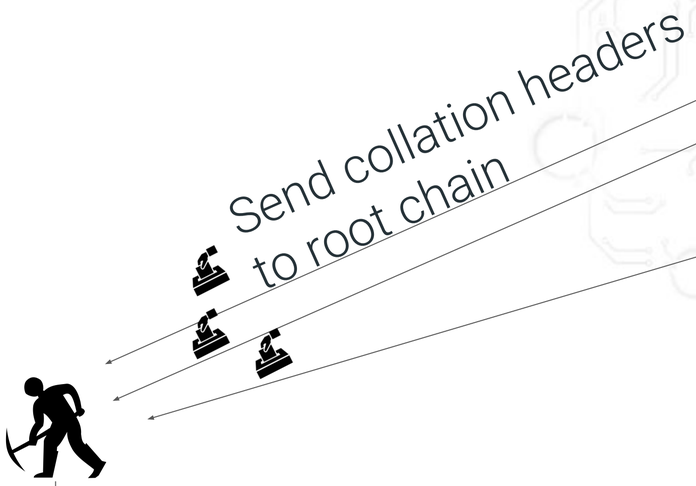
Sharding Phase 1



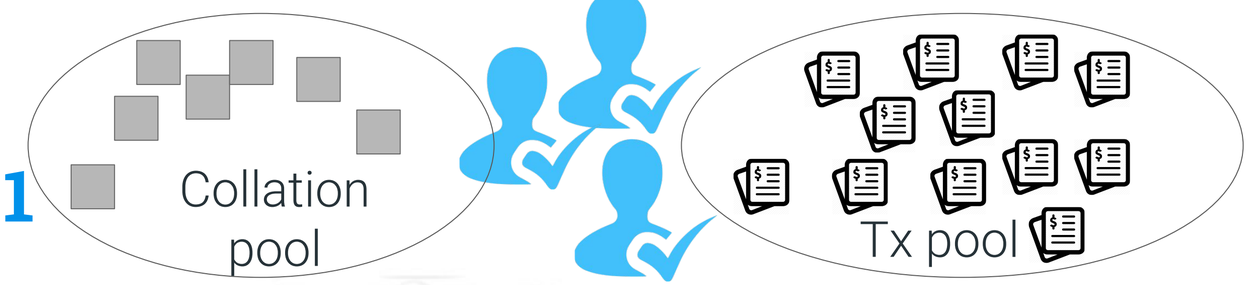


Root chain

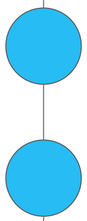


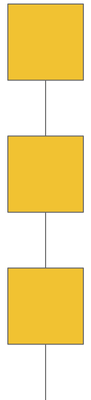


Sharding Phase 1



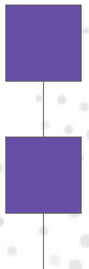
Root chain





Shard 1

Shard 2

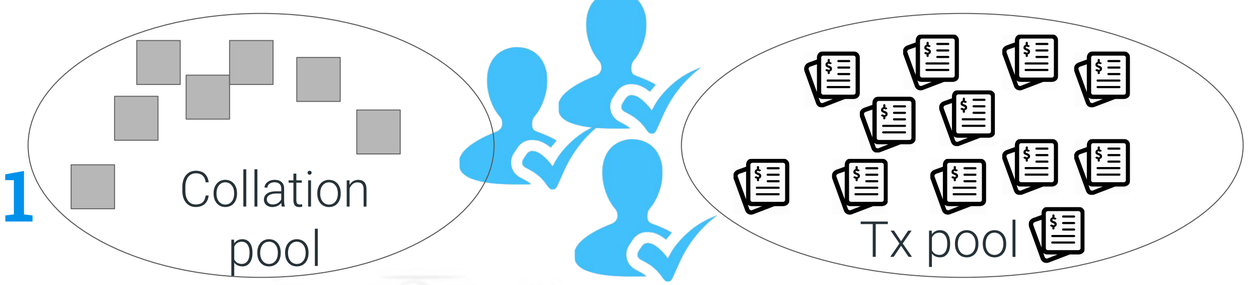


Shard 3

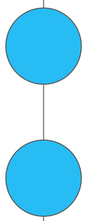
Sharding Phase 1

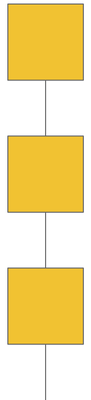
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1



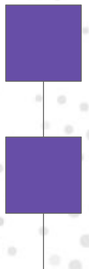
Root chain





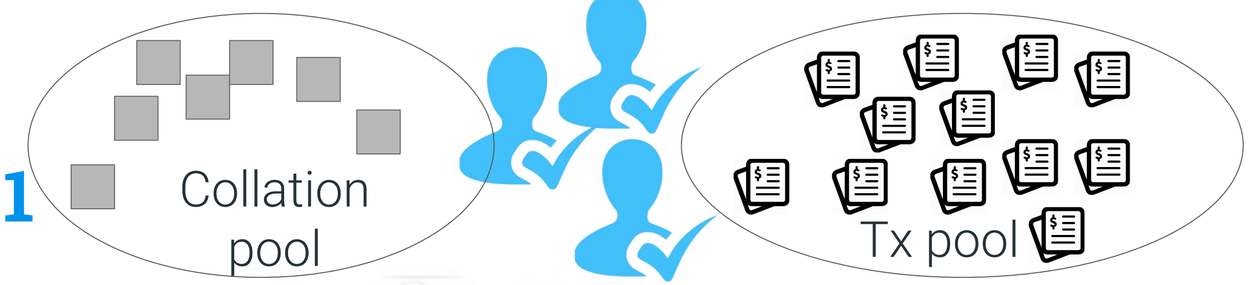
Shard 1

Shard 2

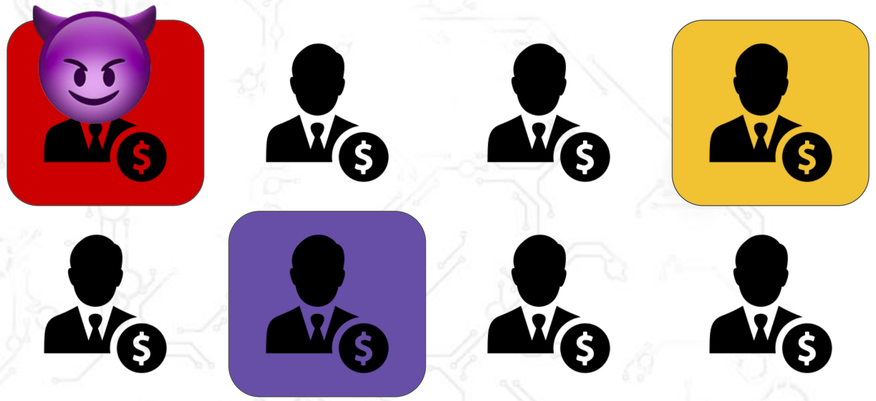


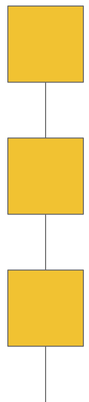
Shard 3

Sharding Phase 1



Root chain



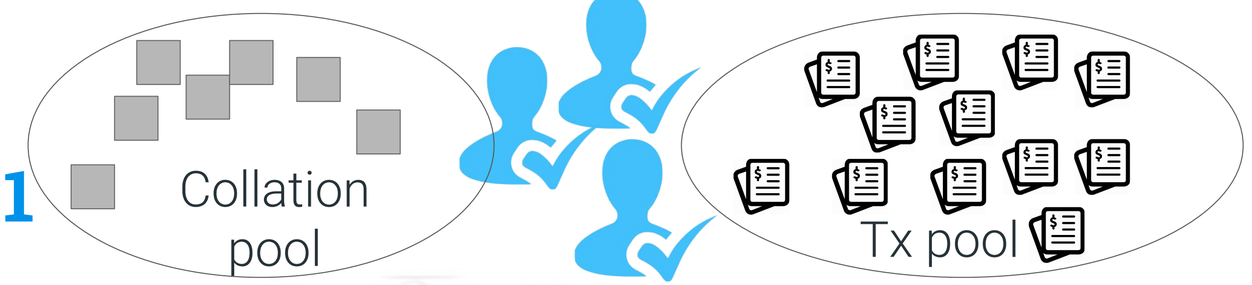


Shard 1

Shard 2

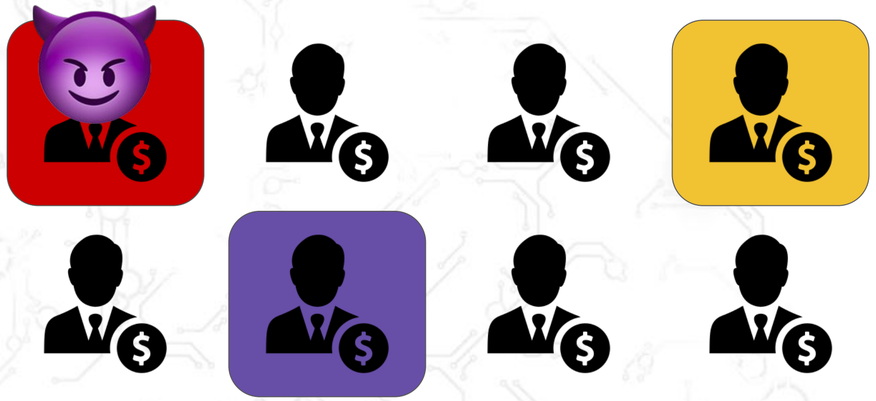
Shard 3

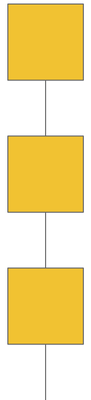
Sharding Phase 1



Root chain

1. Pick collation proposal
2. Verify availability
3. Pick head
4. Submit collation header



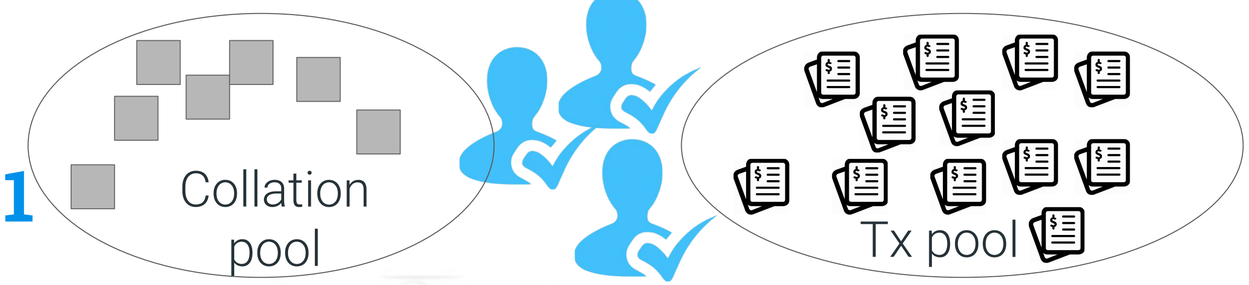


Shard 1

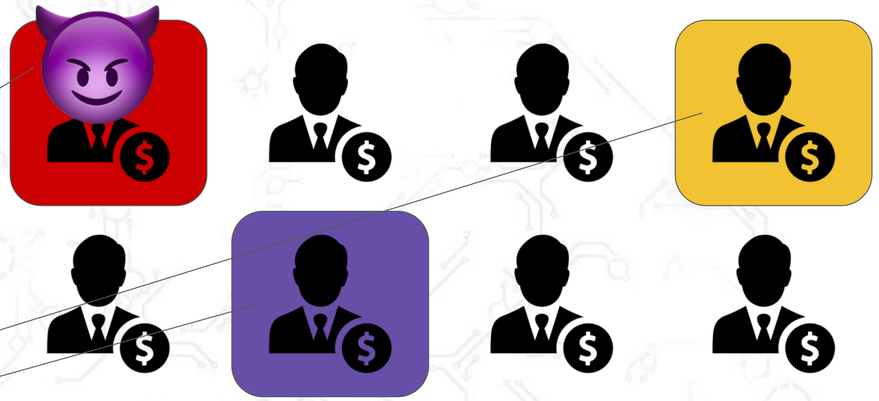
Shard 2

Shard 3

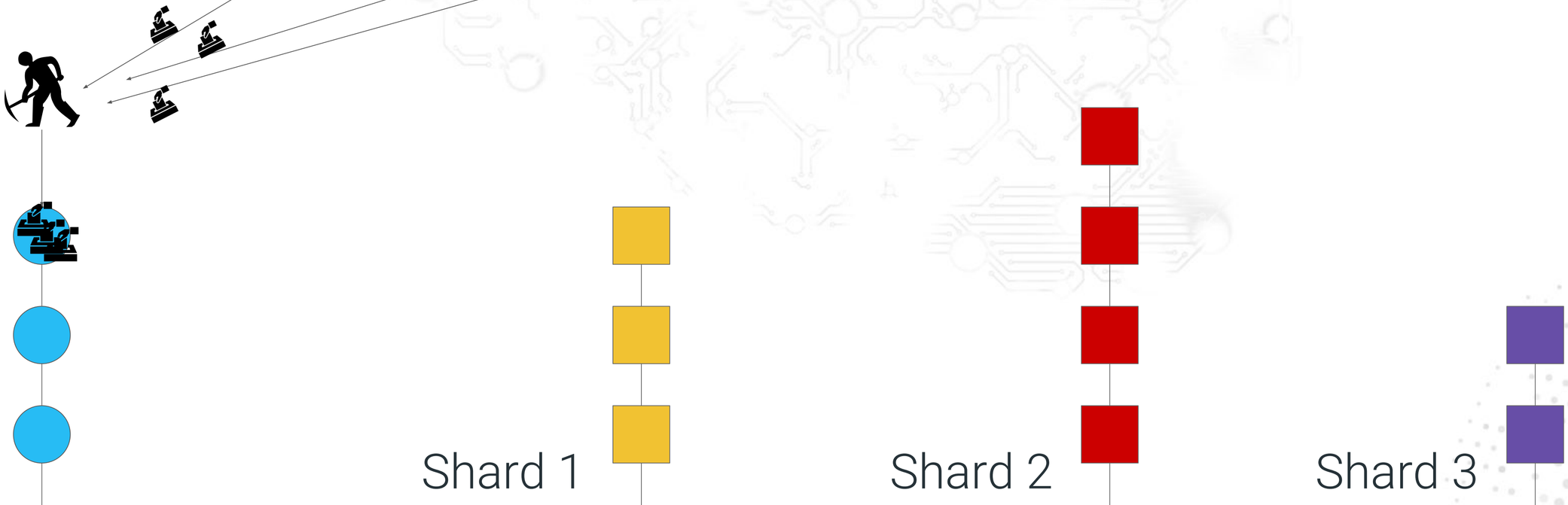
Sharding Phase 1



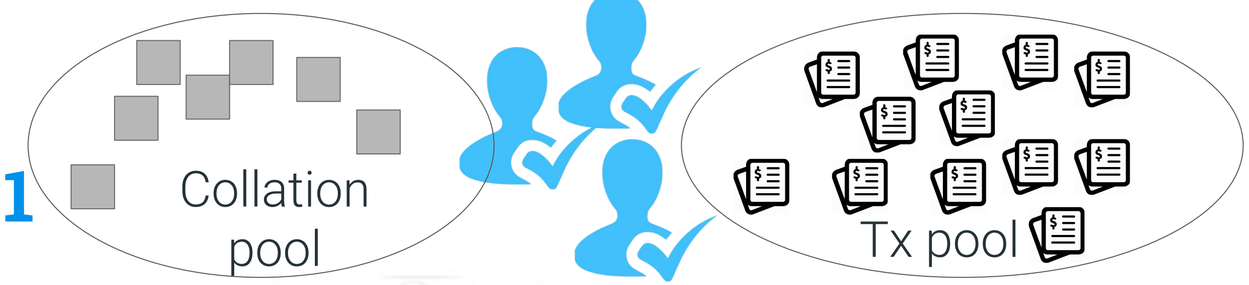
Root chain



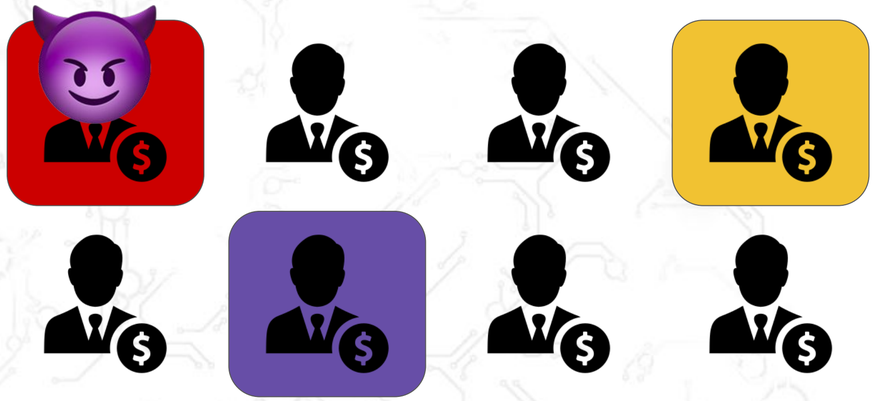
Send collation headers
to root chain

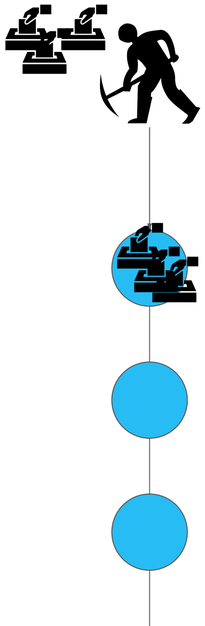


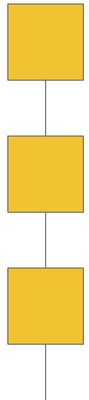
Sharding Phase 1



Root chain





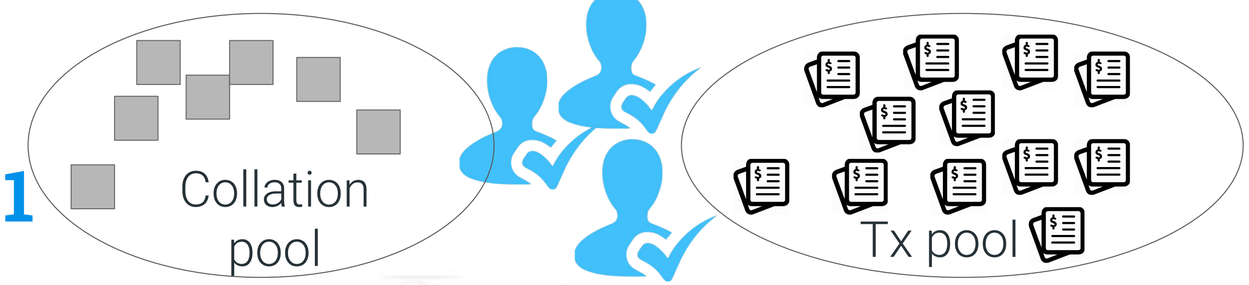


Shard 1

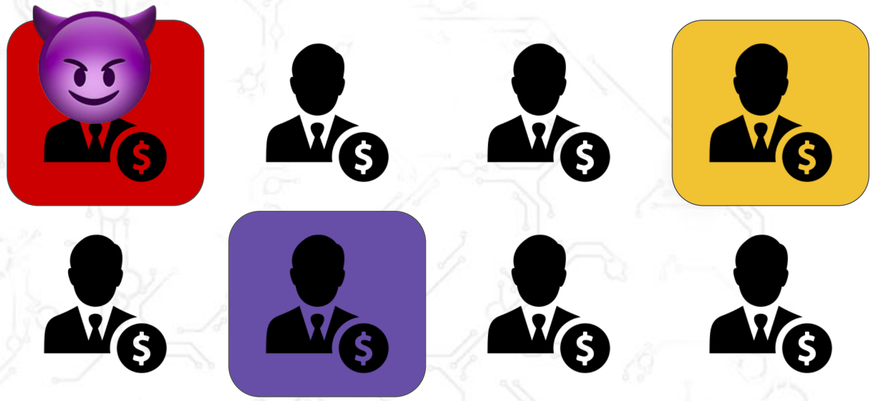
Shard 2

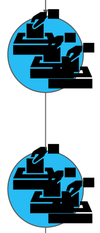
Shard 3

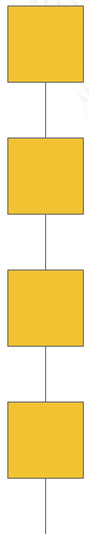
Sharding Phase 1



Root chain

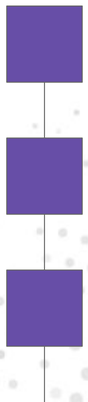






Shard 1

Shard 2

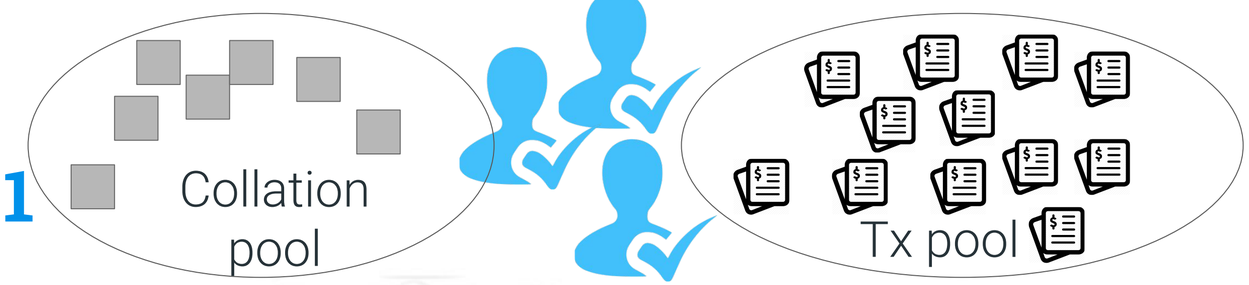


Shard 3

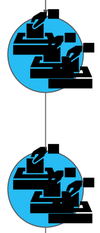
Sharding Phase 1

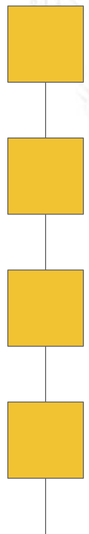
1. Validators use LOOKAHEAD to check which shards they will be validating in the near future
2. Client submits transactions to collation proposers
3. Collation proposers create collations which pay a fee to validators
4. Validators download potential collation proposals
5. Validators verify availability until some depth and pick head to build on
6. Validators submit collation header to the root chain
7. Evil validator submits invalid collation
8. Next validator notices and builds on separate fork

Sharding Phase 1



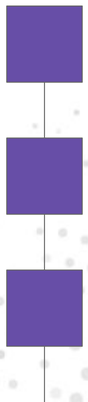
Root chain





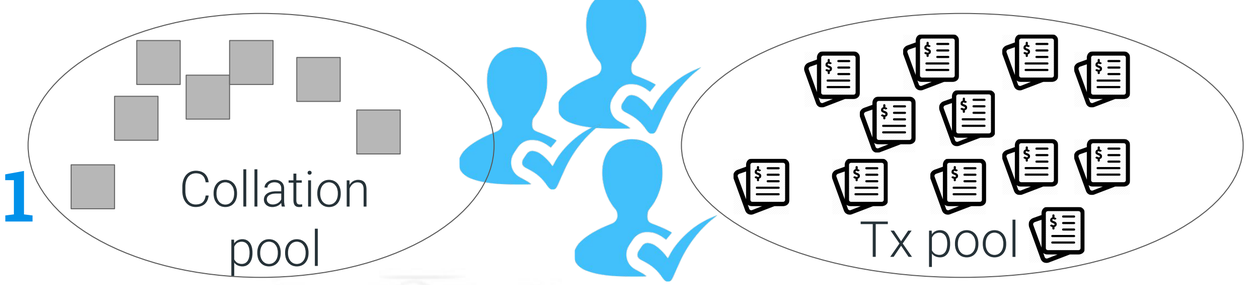
Shard 1

Shard 2

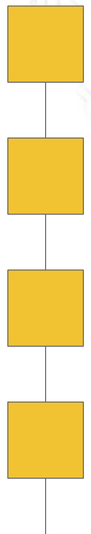


Shard 3

Sharding Phase 1

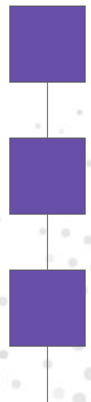


Root chain



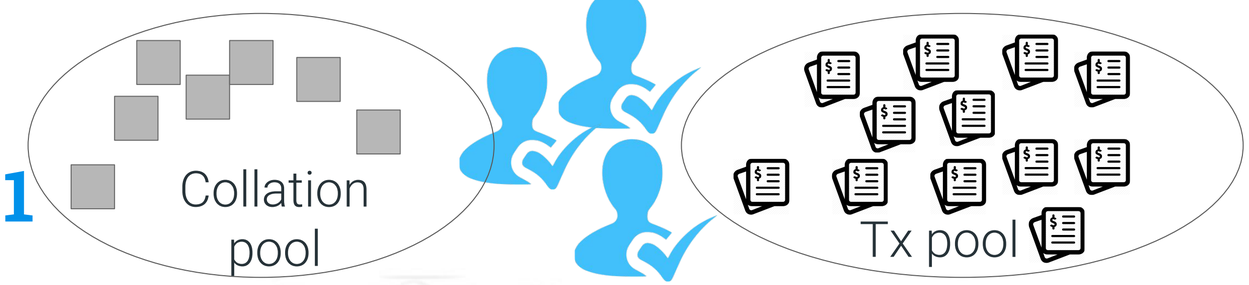
Shard 1

Shard 2

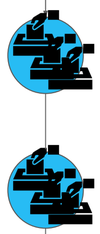


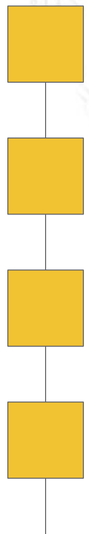
Shard 3

Sharding Phase 1



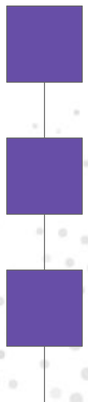
Root chain





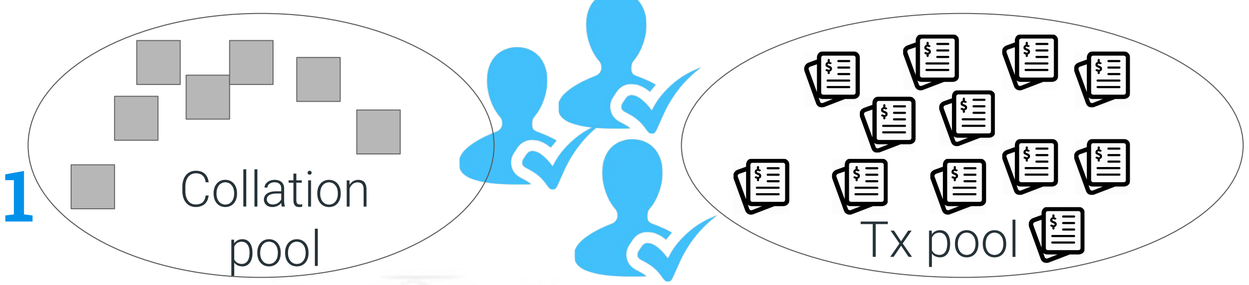
Shard 1

Shard 2



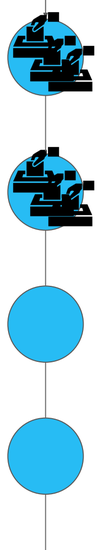
Shard 3

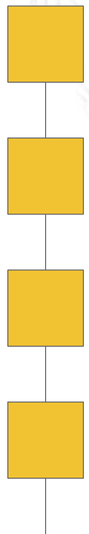
Sharding Phase 1

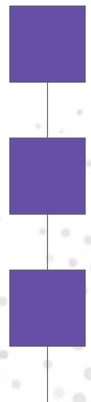


Root chain

During validation phase, the red validator notices unavailable collation, and builds on separate fork





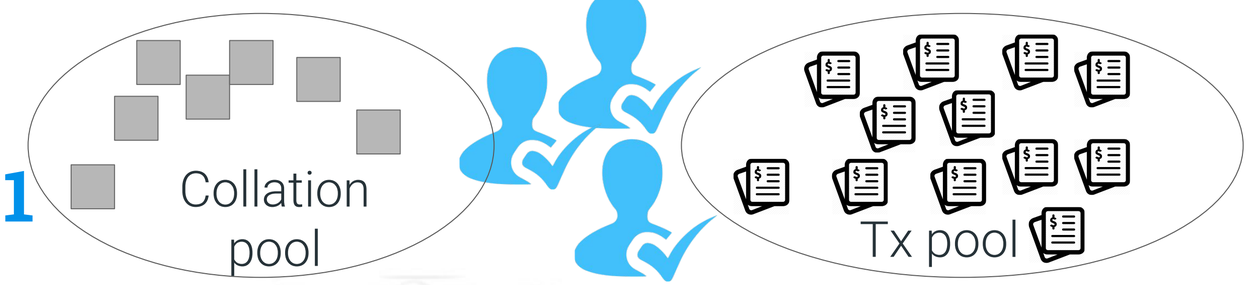


Shard 1

Shard 2

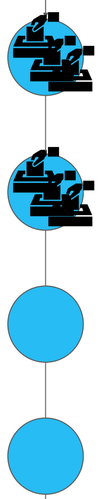
Shard 3

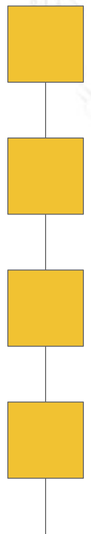
Sharding Phase 1

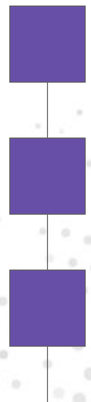


Root chain

During validation phase, the red validator notices unavailable collation, and builds on separate fork





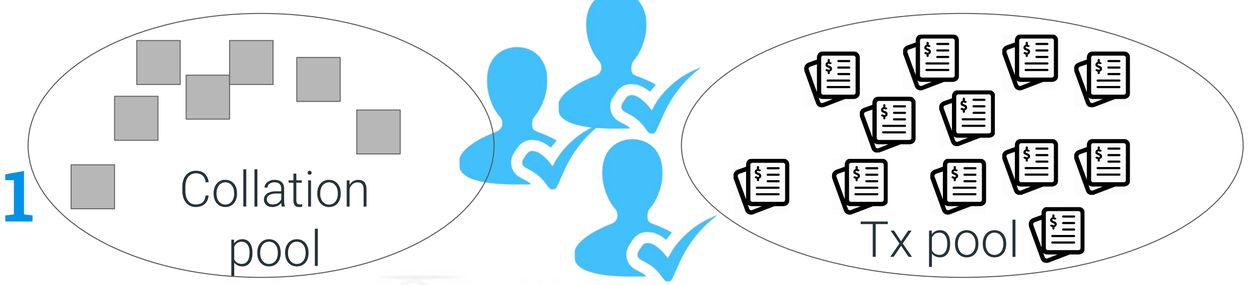


Shard 1

Shard 2

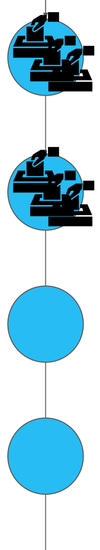
Shard 3

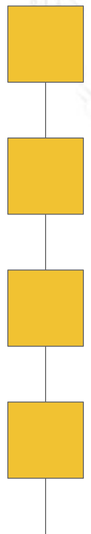
Sharding Phase 1

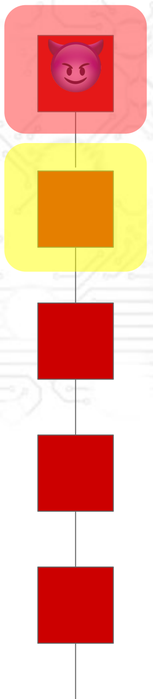


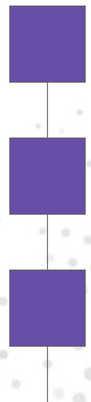
Root chain

During validation phase, the red validator notices unavailable collation, and builds on separate fork







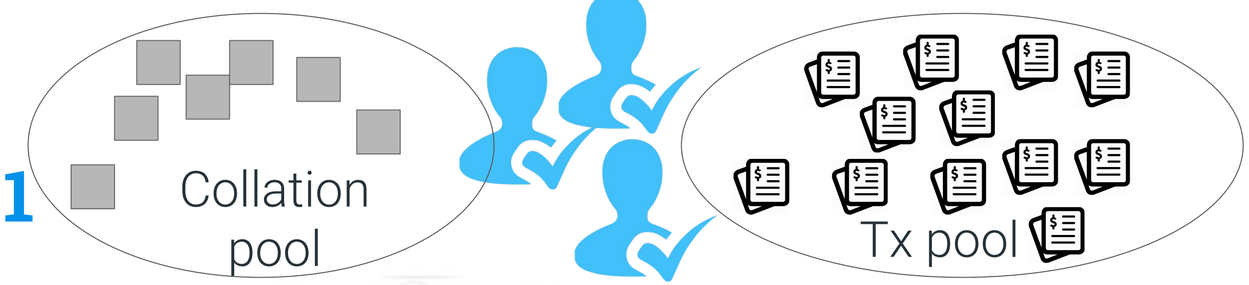


Shard 1

Shard 2

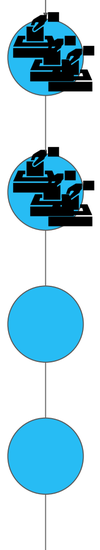
Shard 3

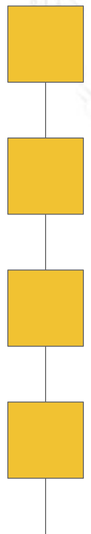
Sharding Phase 1

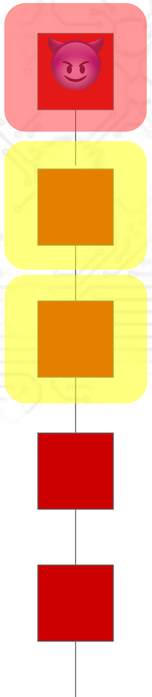


Root chain

During validation phase, the red validator notices unavailable collation, and builds on separate fork





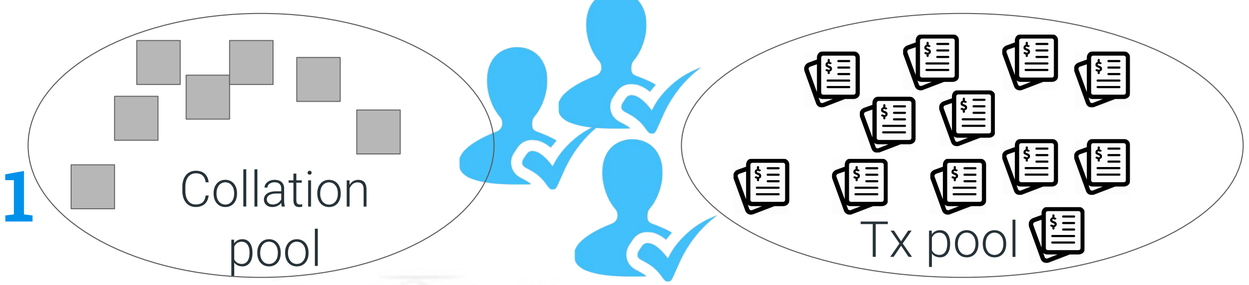


Shard 1

Shard 2

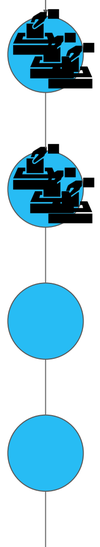
Shard 3

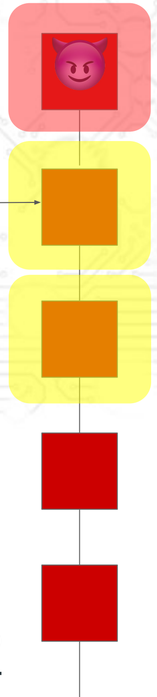
Sharding Phase 1



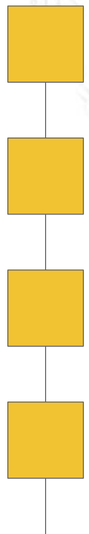
Root chain

During validation phase, the red validator notices unavailable collation, and builds on separate fork



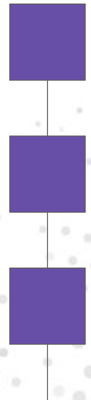


Choose head



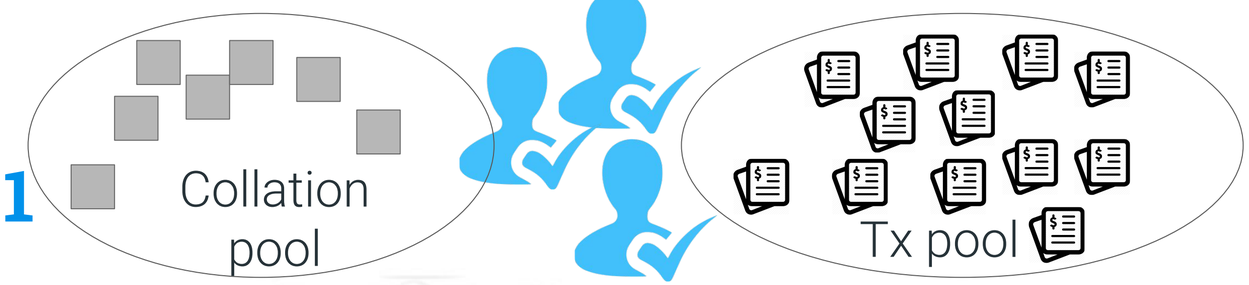
Shard 1

Shard 2

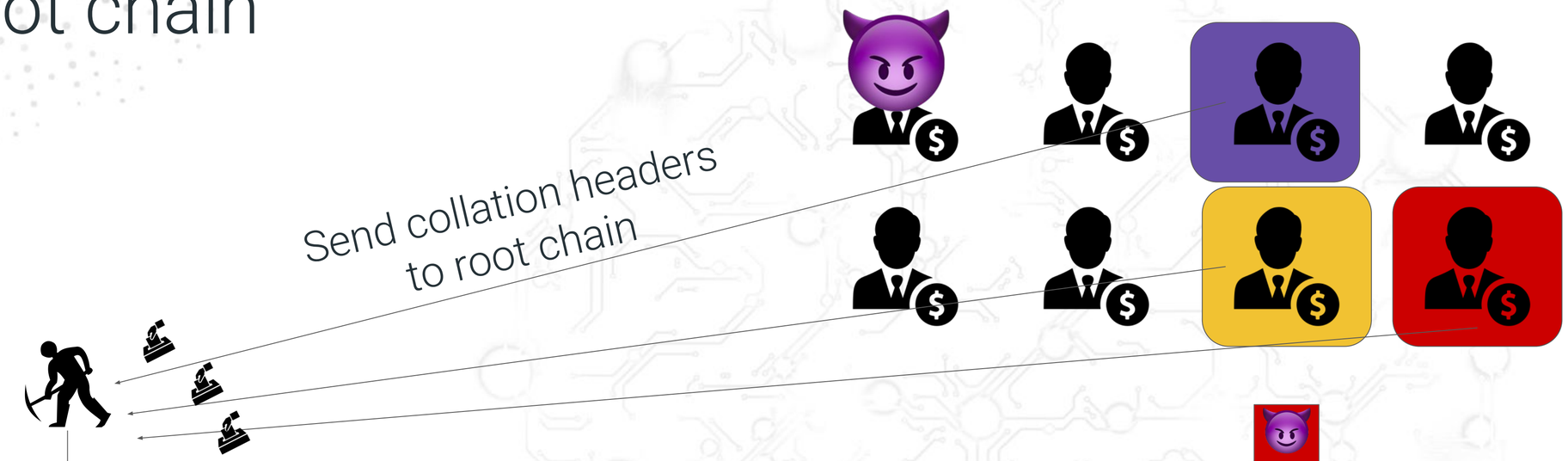


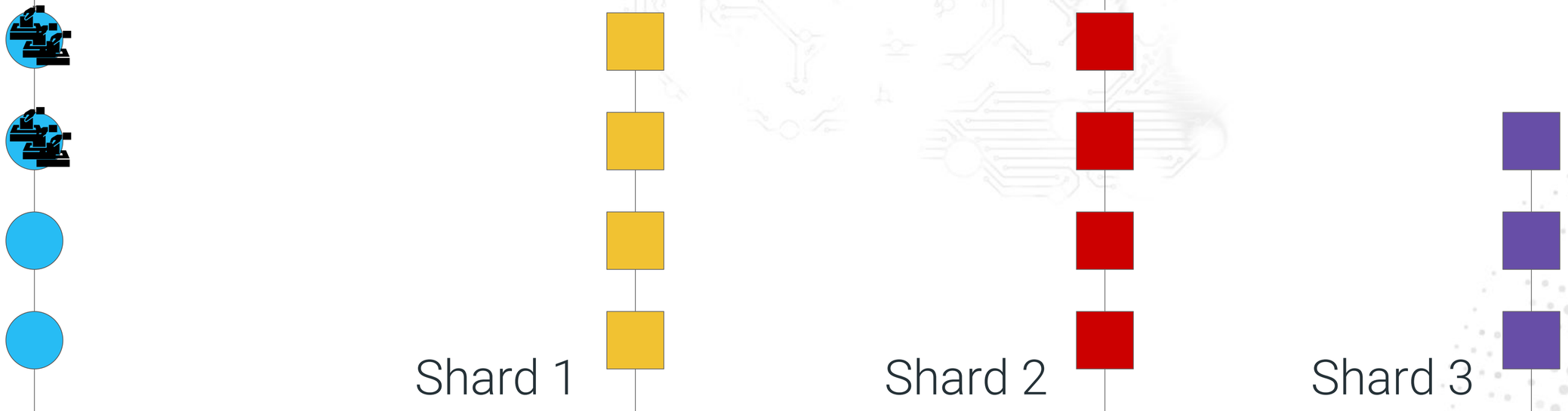
Shard 3

Sharding Phase 1

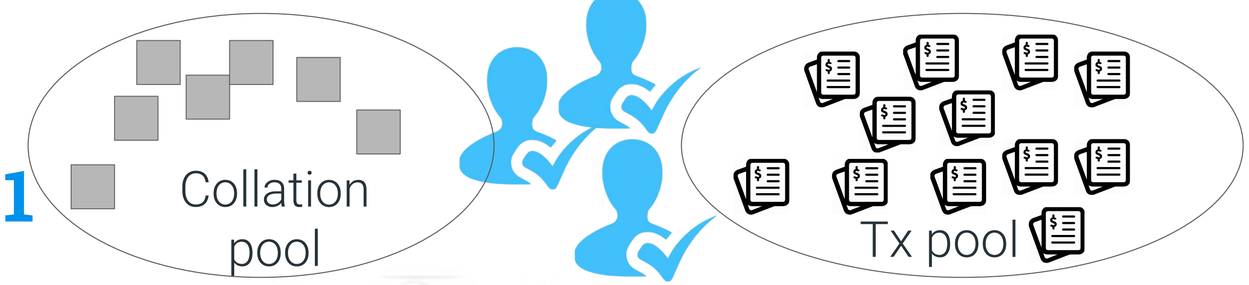


Root chain

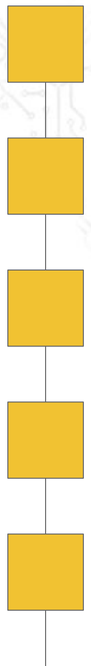




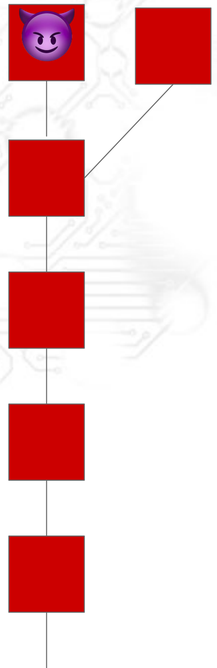
Sharding Phase 1



Root chain



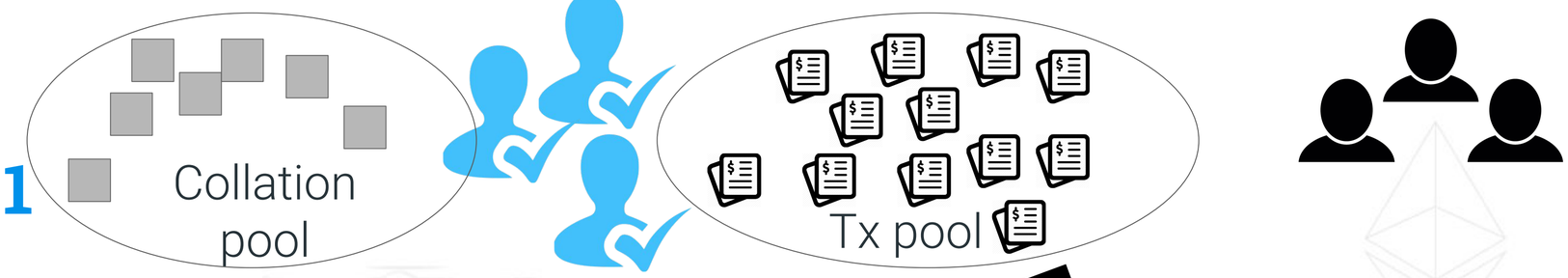
Shard 1



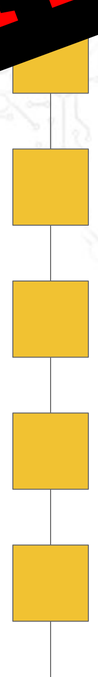
Shard 2

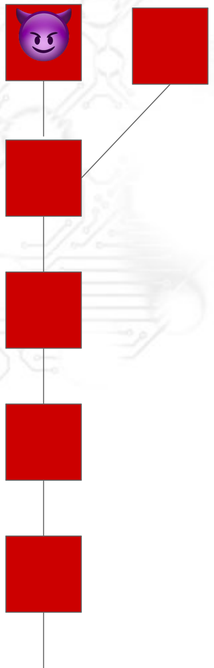
Shard 3

Sharding Phase 1



Root chain





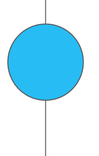
Shard 1

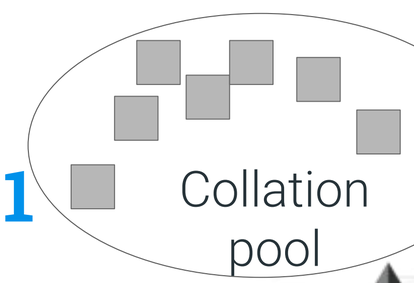
Shard 2

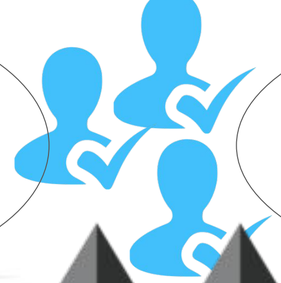
Shard 3

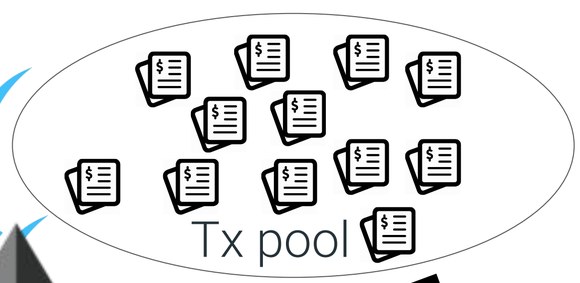
Sharding Phase 1

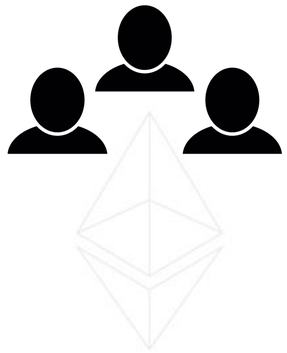
Root chain

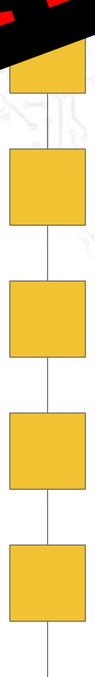












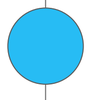
Shard 1

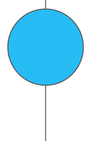
Shard 2

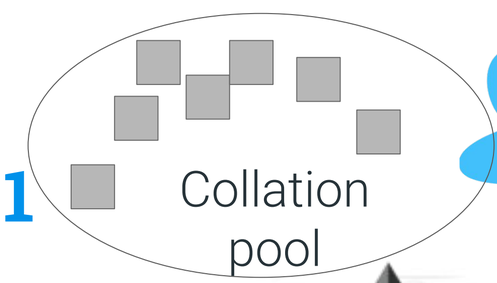
Shard 3

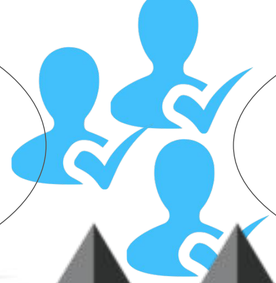
Sharding Phase 1

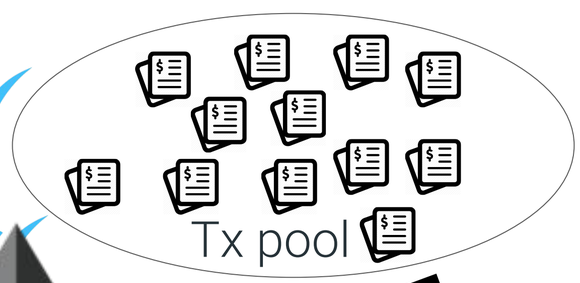
Root chain

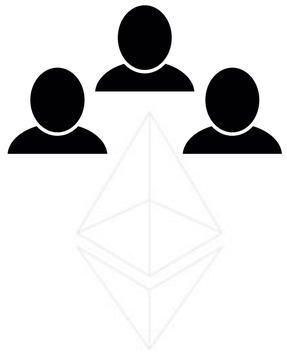












Go Sharding!

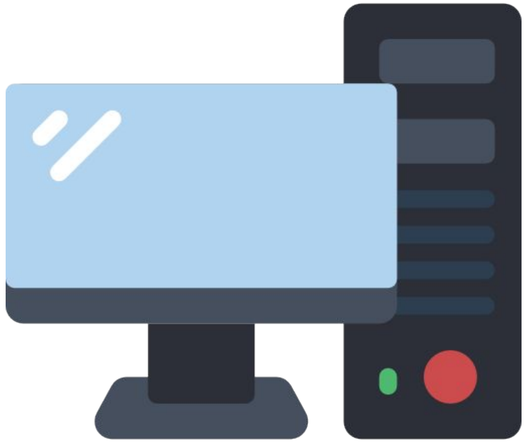
Shard 1

Shard 2

Shard 3

System Roles and Modes

Regular Executor



- Watches some specific shards
- Apply state transition
- Since the proposers would get the tx fee, it’s reasonable that the proposers also are executors.

Light Client

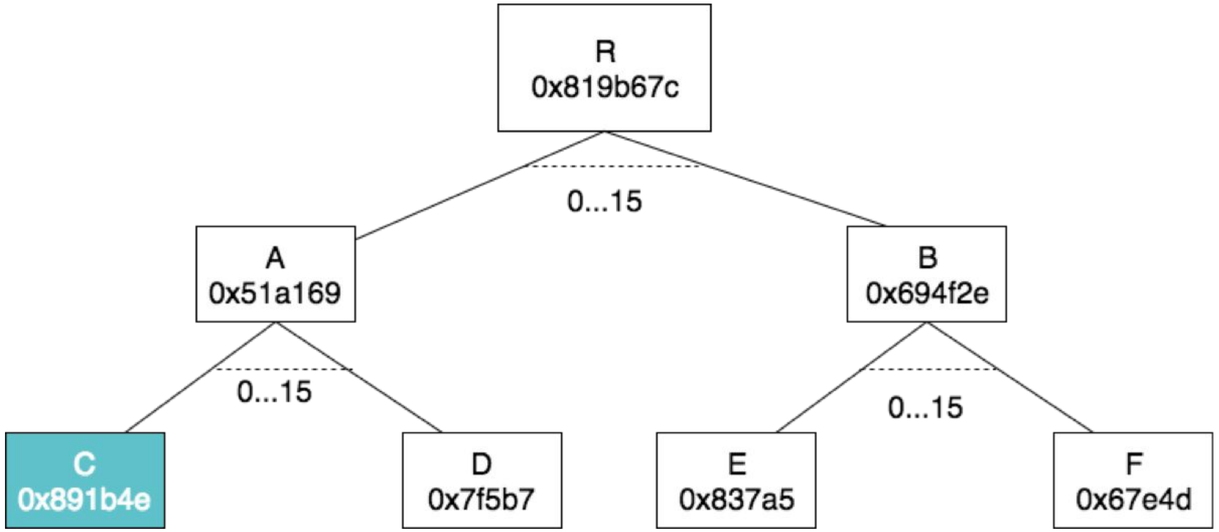
- Validates recent headers
- Watches some specific shards

Spoiler! Day 2
Stateless Client
Mechanism

Stateful or Stateless Modes

Spoiler! Day 2
Stateless Client
Mechanism

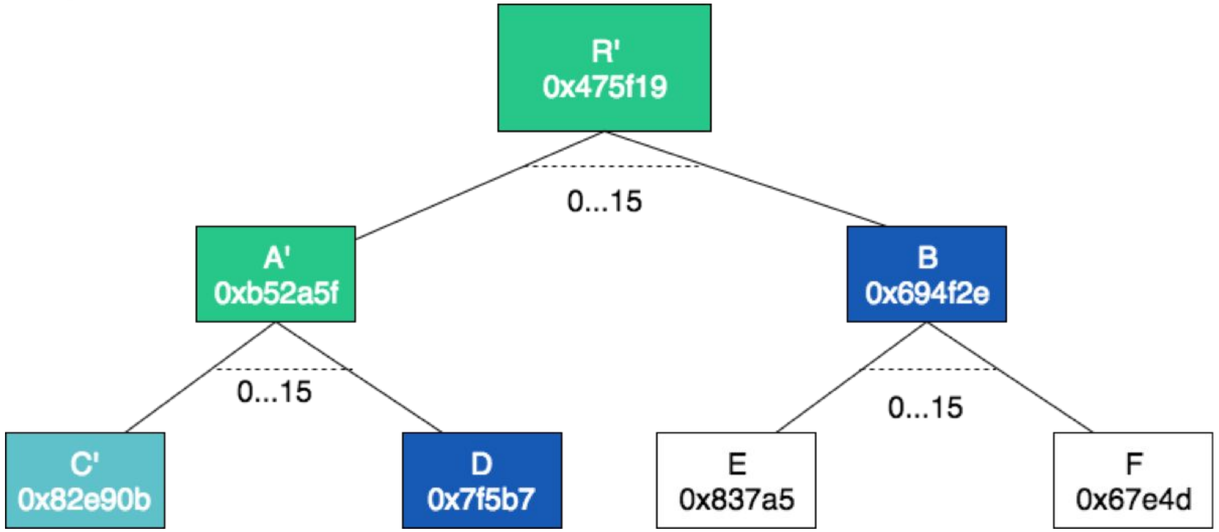
Pre-state



Spoiler! Day 2 Stateless Client Mechanism

Post-state

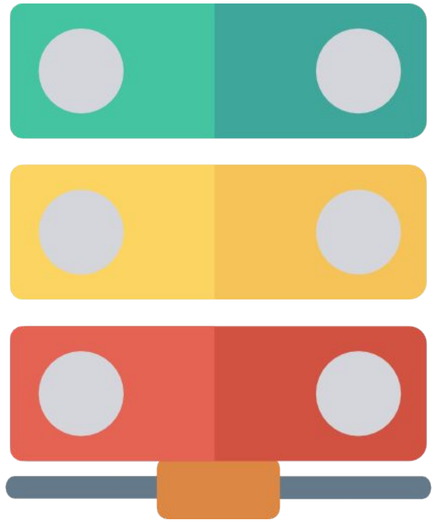
Witness: {R, A, C, D, B}



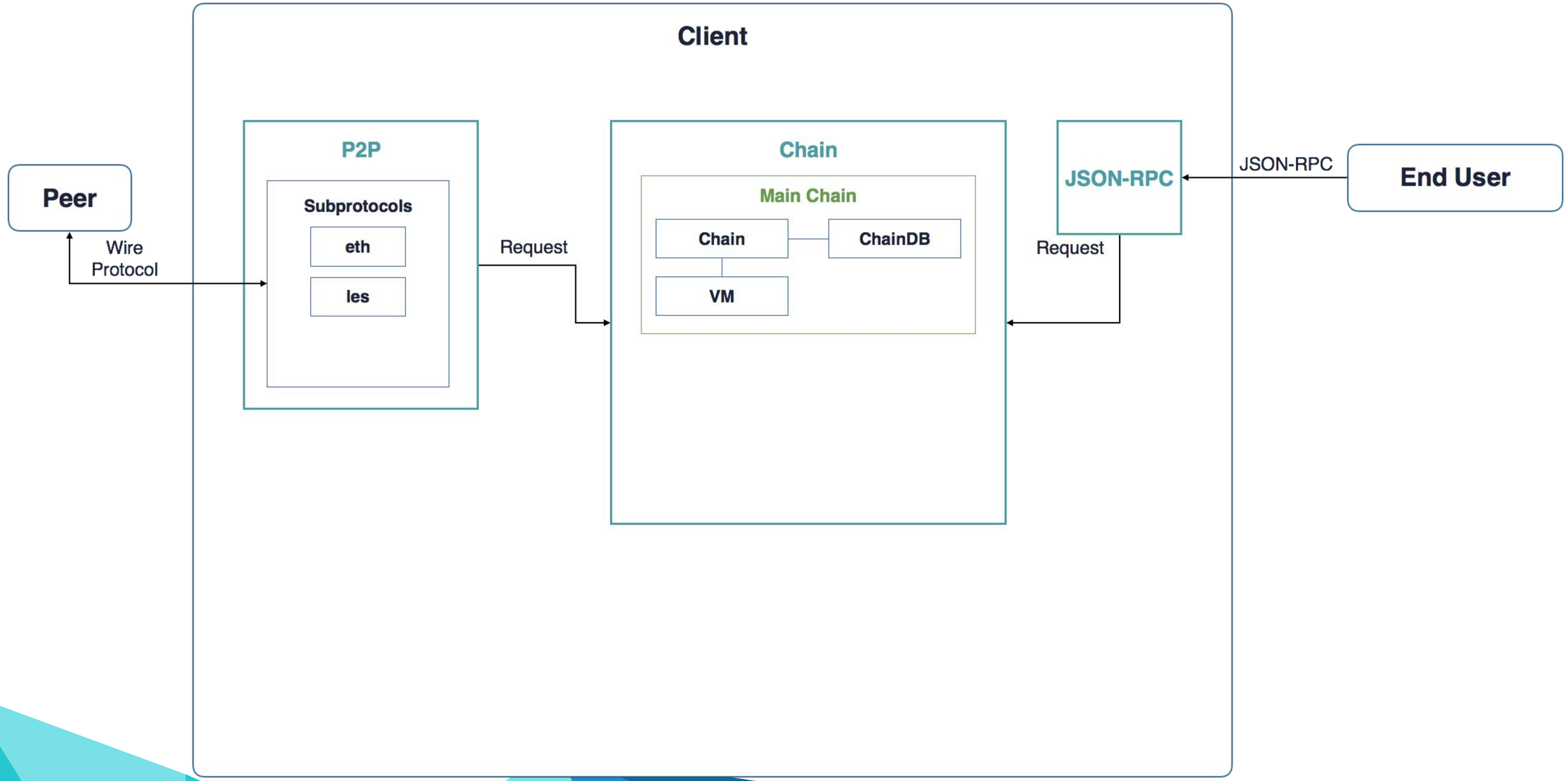
state_transition_function(state_root, collation, witness)

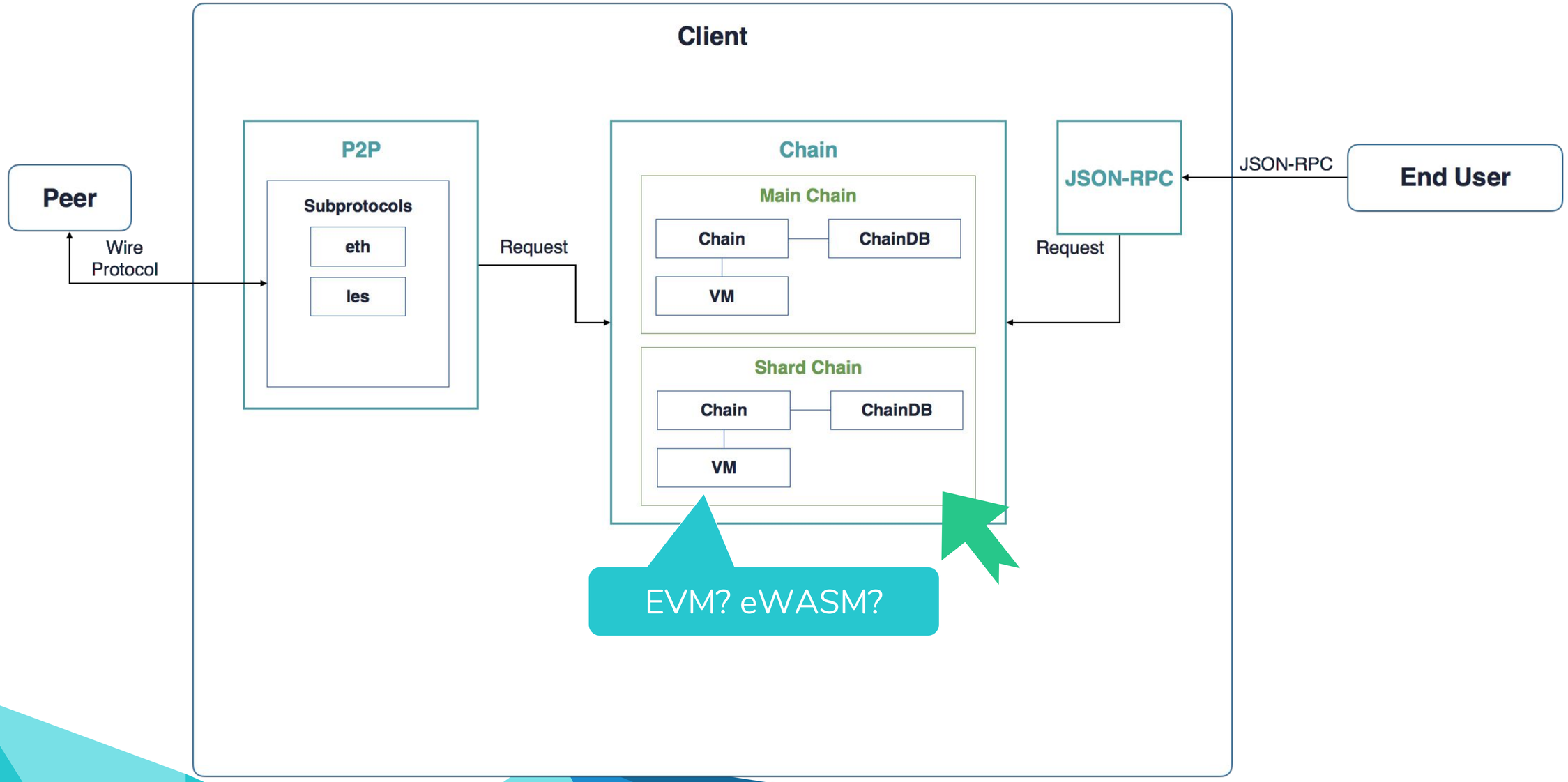
→ state_root', read_set, write_set

Archival Node

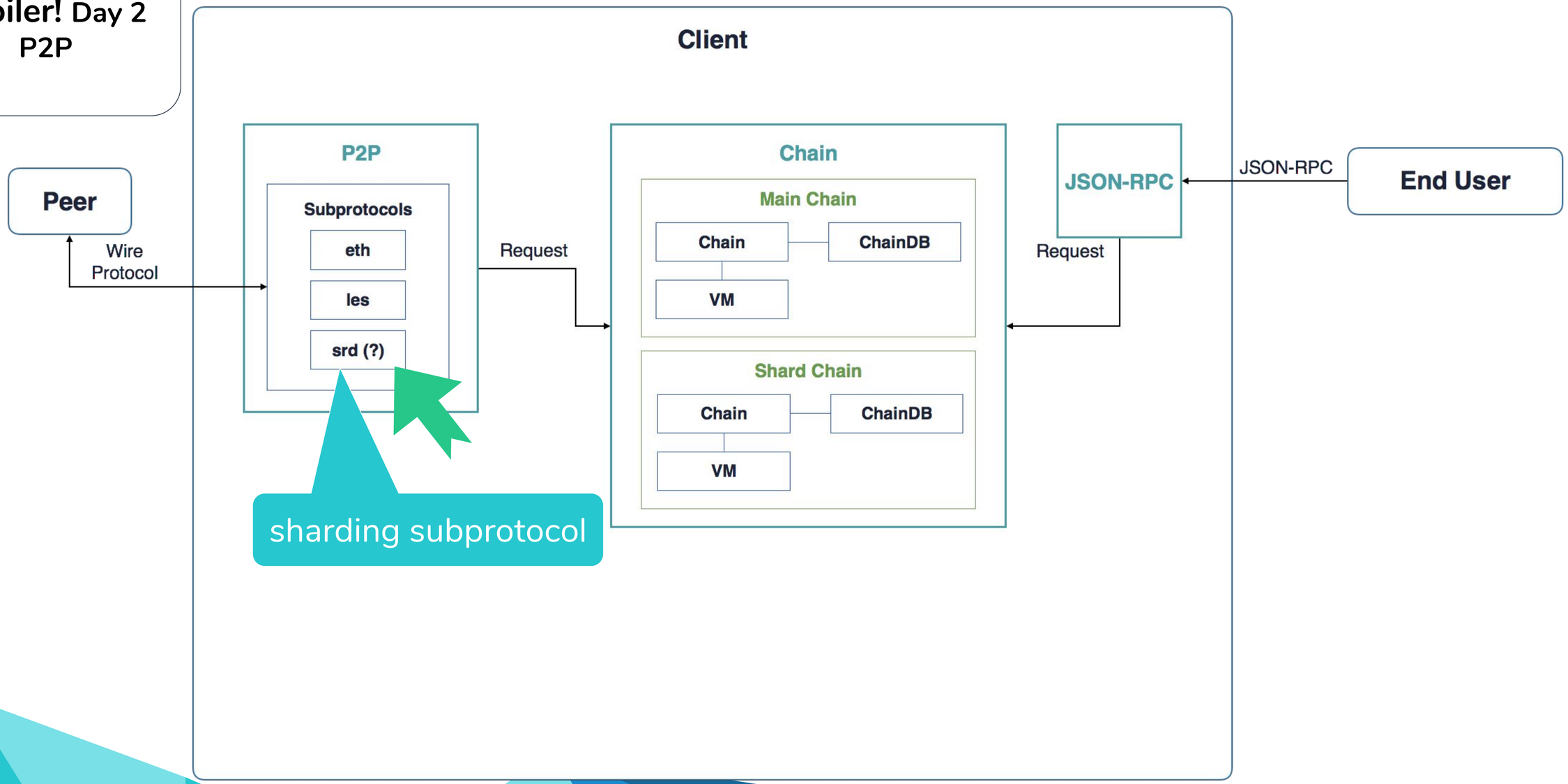


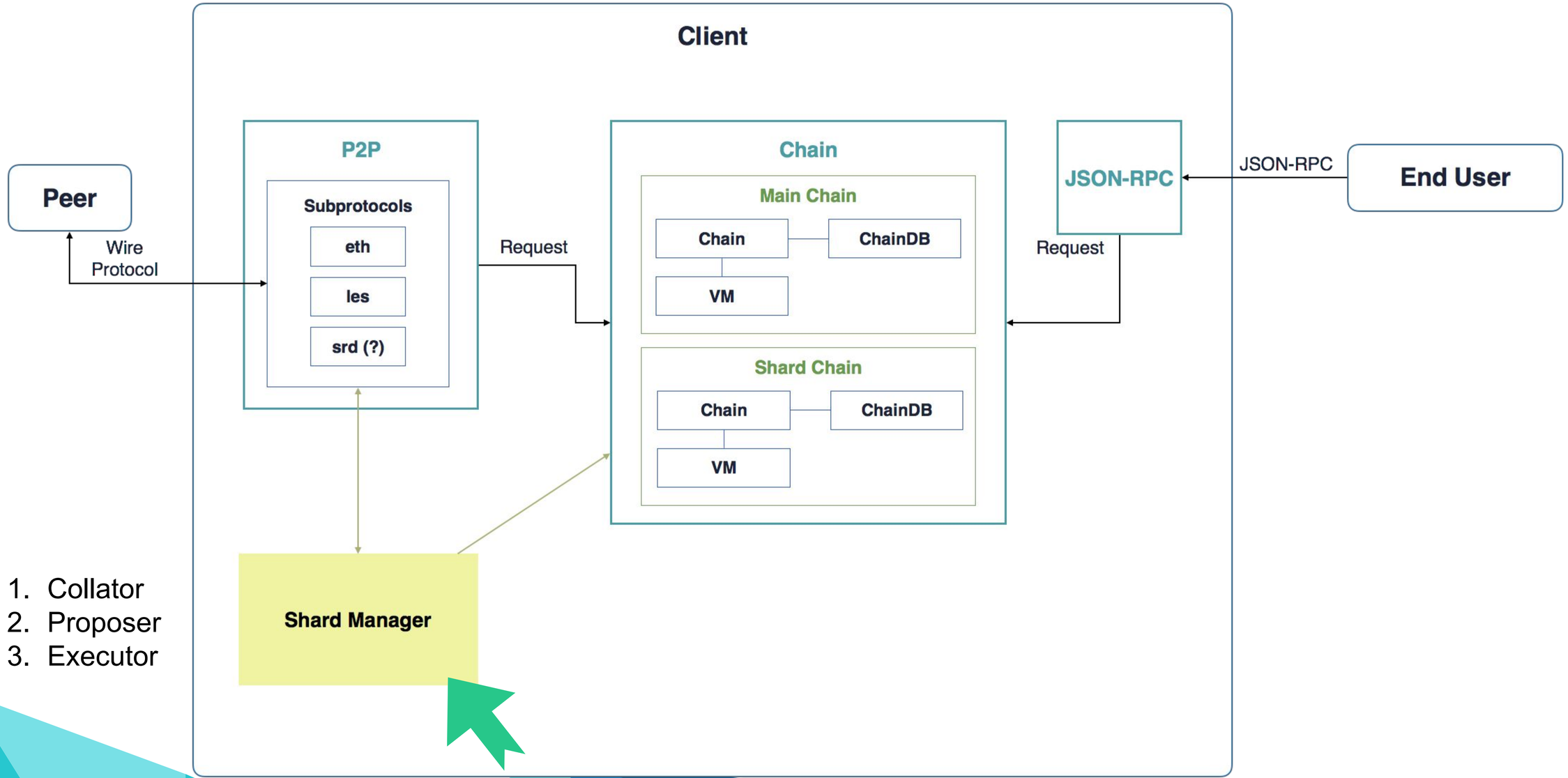
Shard Client Components


Client
P2P
Subprotocols
eth
les
Chain
Main Chain
Chain
ChainDB
VM
JSON-RPC
Peer
Wire
Protocol
Request
Request
JSON-RPC
End User

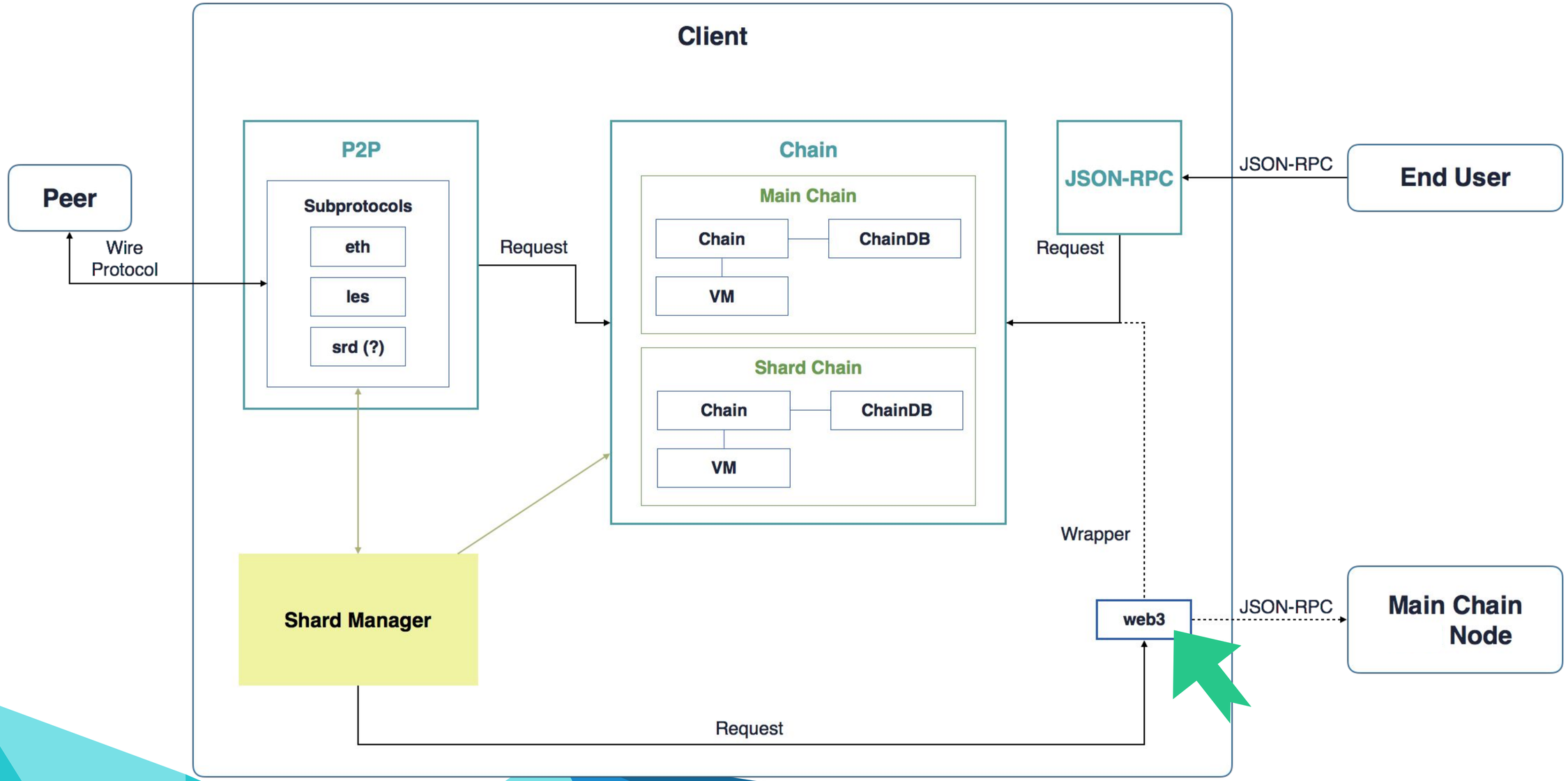

Client
P2P
Subprotocols
eth
les
Chain
Main Chain
Chain
ChainDB
VM
Shard Chain
Chain
ChainDB
VM
JSON-RPC
Peer
End User
Wire
Protocol
Request
Request
JSON-RPC
EVM? eWASM?

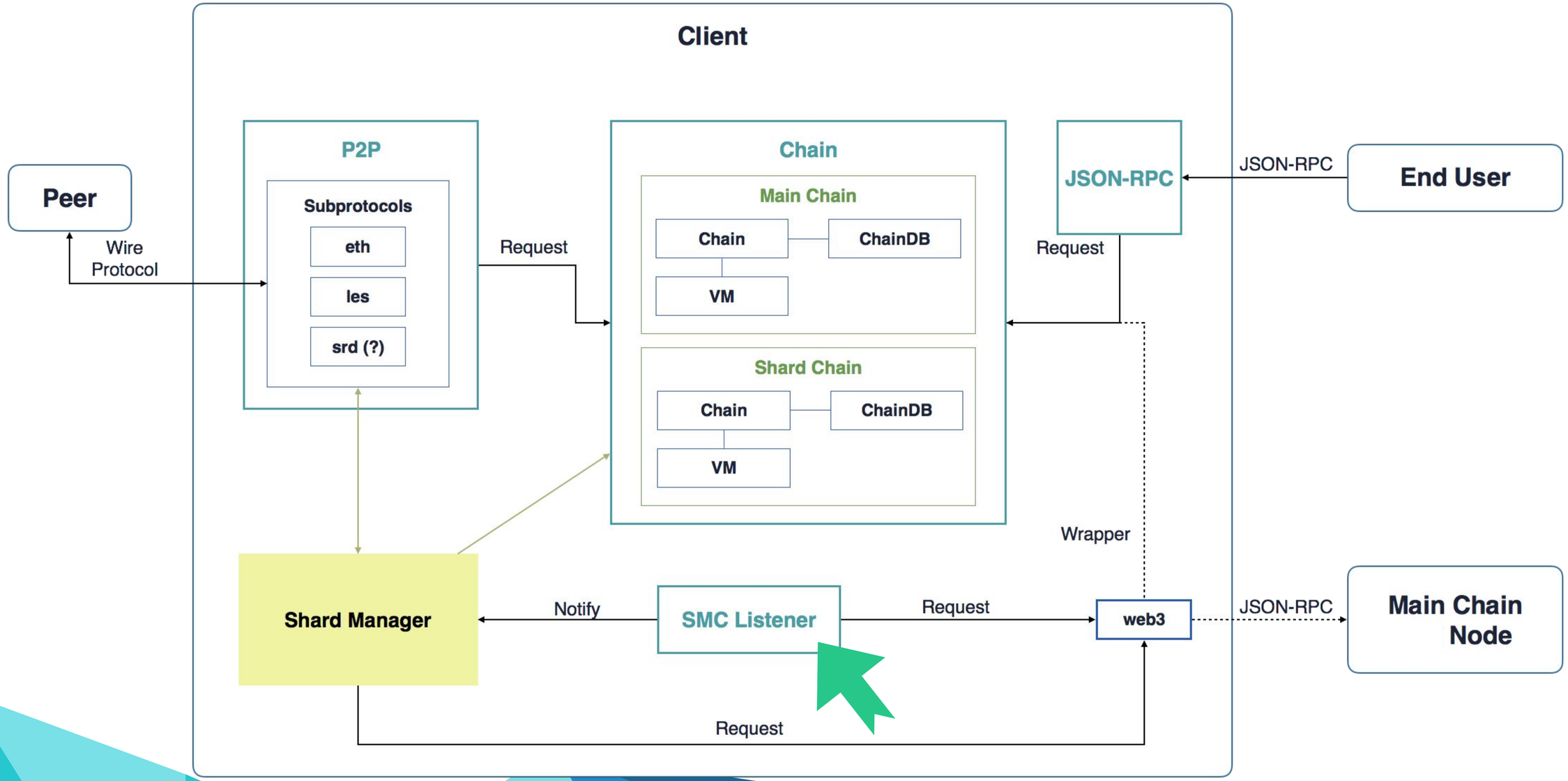
Spoiler! Day 2
P2P





1. Collator
2. Proposer
3. Executor


Client
P2P
Subprotocols
eth
les
srd (?)
Chain
Main Chain
Chain
ChainDB
VM
Shard Chain
Chain
ChainDB
VM
Peer
Wire
Protocol
Request
JSON-RPC
JSON-RPC
End User
Request
Wrapper
Shard Manager
web3
JSON-RPC
Main Chain
Node
Request


Client
Peer
Wire Protocol
P2P
Subprotocols
eth
les
srd (?)
Request
Chain
Main Chain
Chain
ChainDB
VM
Shard Chain
Chain
ChainDB
VM
JSON-RPC
JSON-RPC
End User
Request
Wrapper
Shard Manager
Notify
SMC Listener
Request
web3
JSON-RPC
Main Chain Node
Request

(New*) Roadmap

Phase 1

Basic sharding without EVM

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

1
2
3
4
5
6

Phase 2

EVM state transition function

- **Full nodes only**
- Asynchronous cross-contract transactions only
- Account abstraction
- eWASM
- Asynchronous zones
- Archive accumulators
- Storage rent

Spoiler! Day 3
Cross-contract communication

Phase 2

EVM state transition function

- Full nodes only
- **Asynchronous cross-contract transactions only**
- Account abstraction
- eWASM
- Asynchronous zones
- Archive accumulators
- Storage rent

1
2
3
4
5
6

Spoiler! Day 2 Account Abstraction and Gas Payment

Phase 2

EVM state transition function

- Full nodes only
- Asynchronous cross-contract transactions only
- **Account abstraction**
- eWASM
- Asynchronous zones
- Archive accumulators
- Storage rent

1
2
3
4
5
6

Spoiler! Day 2
eWASM

Phase 2

EVM state transition function

- Full nodes only
- Asynchronous cross-contract transactions only
- Account abstraction
- **eWASM**
- Asynchronous zones
- Archive accumulators
- Storage rent

1
2
3
4
5
6

Spoiler! Day 3
Cross-contract communication

Phase 2

EVM state transition function

- Full nodes only
- Asynchronous cross-contract transactions only
- Account abstraction
- eWASM
- **Asynchronous zones**
- Archive accumulators
- Storage rent

1
2
3
4
5
6

Spoiler! Day 2
Execution-minimisation and State-minimisation

Phase 2

EVM state transition function

- Full nodes only
- Asynchronous cross-contract transactions only
- Account abstraction
- eWASM
- Asynchronous zones
- **Archive accumulators**
- Storage rent

1
2
3
4
5
6

Phase 2

EVM state transition function

- Full nodes only
- Asynchronous cross-contract transactions only
- Account abstraction
- eWASM
- Asynchronous zones
- Archive accumulators
- **Storage rent**

1
2
3
4
5
6

Spoiler! Day 1
Execution

Phase 3

Light client state protocol

- **Executors**
- Stateless clients

Spoiler! Day 2
Stateless Client
Mechanism

Phase 3

Light client state protocol

- Executors
- **Stateless clients**

1
2
3
4
5
6

Spoiler! Day 3
Cross-contract communication

Phase 4

Cross-shard transactions

- **Zones**

Spoiler! Day 3
Scalable data availability checking

Phase 5

Tight coupling with main chain security

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

1
2
3
4
5
6

Spoiler! Day 3
Ethereum 2.0
End game

Phase 5

Tight coupling with main chain security

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

1
2
3
4
5
6

Spoiler! Day 3
Ethereum 2.0
End game

Phase 5

Tight coupling with main chain security

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

1
2
3
4
5
6

Spoiler! Day 3
Ethereum 2.0
End game

Phase 5

Tight coupling with main chain security

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

1
2
3
4
5
6

Spoiler! Day 3
Ethereum 2.0
End game

Phase 6

Super-quadratic sharding

- **Load balancing**

1
2
3
4
5
6

Research Topics

Research Topics

New Schemes

Research Topics

New Schemes

Vulnerability Analysis

Research Topics

New Schemes

Vulnerability Analysis

Optimization

Workshop Agenda

Day 1 Agenda

- **General Introduction** 10:30 - 12:00

— Lunch: 12:00 - 13:30 —

- **Sharding Manager Contract** 13:30 - 14:30
- **Proposer / Collator Separation** 14:30 - 16:00

— Break: 16:00 - 16:15 —

- **Execution** 16:15 - 17:45

- **Dinner** starts at 19:00

Day 2 Agenda

• **eWASM**	09:00 - 10:00
• **Execution-minimisation and State-minimisation**	10:00 - 11:00
• **Account Abstraction and Gas Payment**	11:00 - 12:00
— Lunch: 12:00 - 13:20 —	
• **Stateless Client Mechanism**	13:20 - 14:50
• **Access lists, Account Restriction and Parallelizability**	14:50 - 15:35
— Break: 16:00 - 16:15 —	
• **P2P Networking**	15:50 - 17:20

Day 3 Agenda

- **Cross-contract Communication** 09:00 - 10:30
- **Scalable Data Availability Checking** 10:30 - 12:00

— Lunch 12:00 - 13:30 —

- **Security Models Mechanism Design** 13:30 - 15:00
- **Ethereum 2.0 End-Game** 15:00 - 15:45

— Closing: 15:45 - 15:55 —

Enjoy it!






Thank you!

CREDITS

Special thanks to all people who made and share these awesome resources for free:

- Taiwan Emoji Project
- Presentation template designed by Slidesmash
- Photographs by unsplash.com and pexels.com
- Vector Icons by Matthew Skiles
- Icons made by DinosoftLabs from www.flaticon.com is licensed by CC 3.0 BY

Presentation Design

This presentation uses the following typographies and colors:

Free Fonts used:

https://www.fontsquirrel.com/fonts/nunito

Colors used