Designing an End to End System for Based Preconfirmations

Anshu Jalan
Blockchain Engineer, Nethermind

Ahmad Bitar
Ethereum Core Developer, Nethermind

Speakers

Ahmad Bitar

Ethereum Core Developer, Nethermind

Anshu Jalan

Blockchain Engineer, Nethermind

No live coding.
All about Design Thinking.

Code at: github.com/NethermindEth/Taiko-Preconf-AVS

Foundational Knowledge

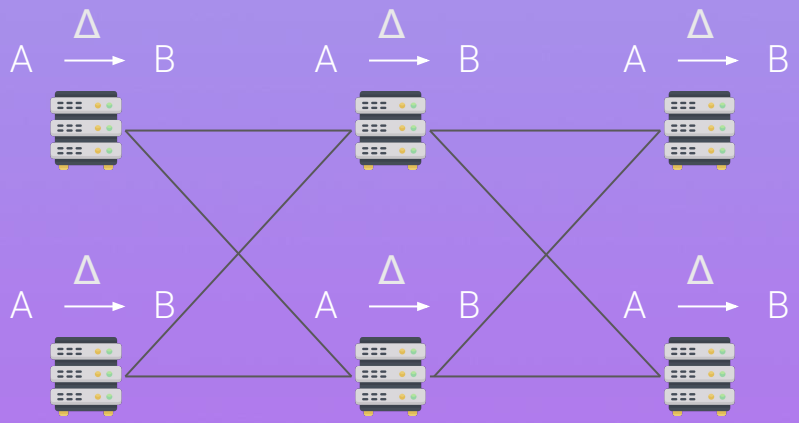
Rollups

L1 is Slow!

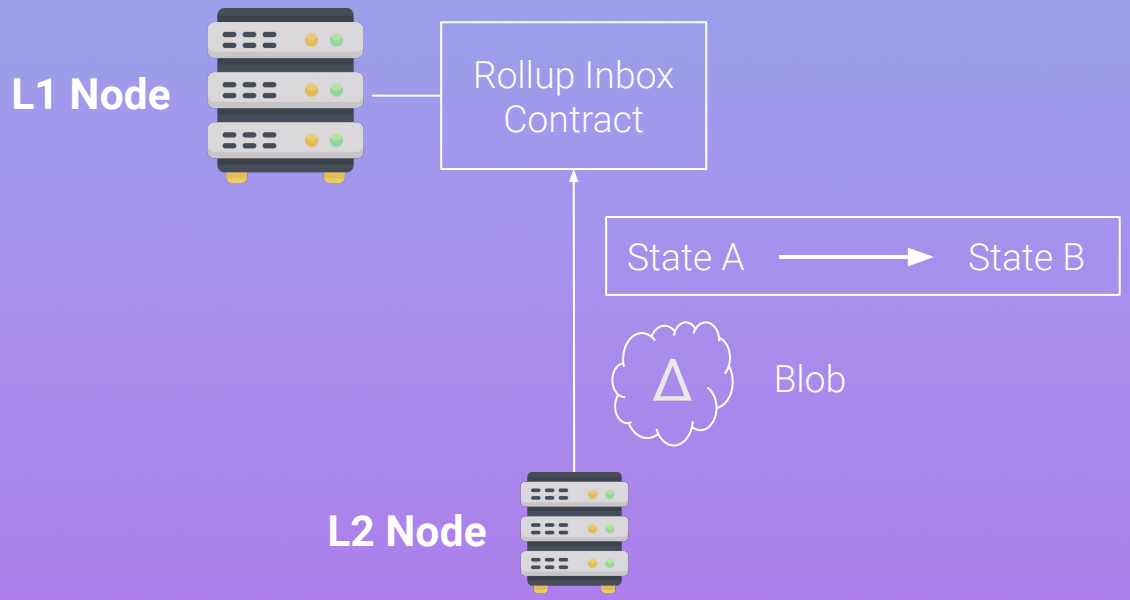
Txn 1
Txn 2
....

State A → State B

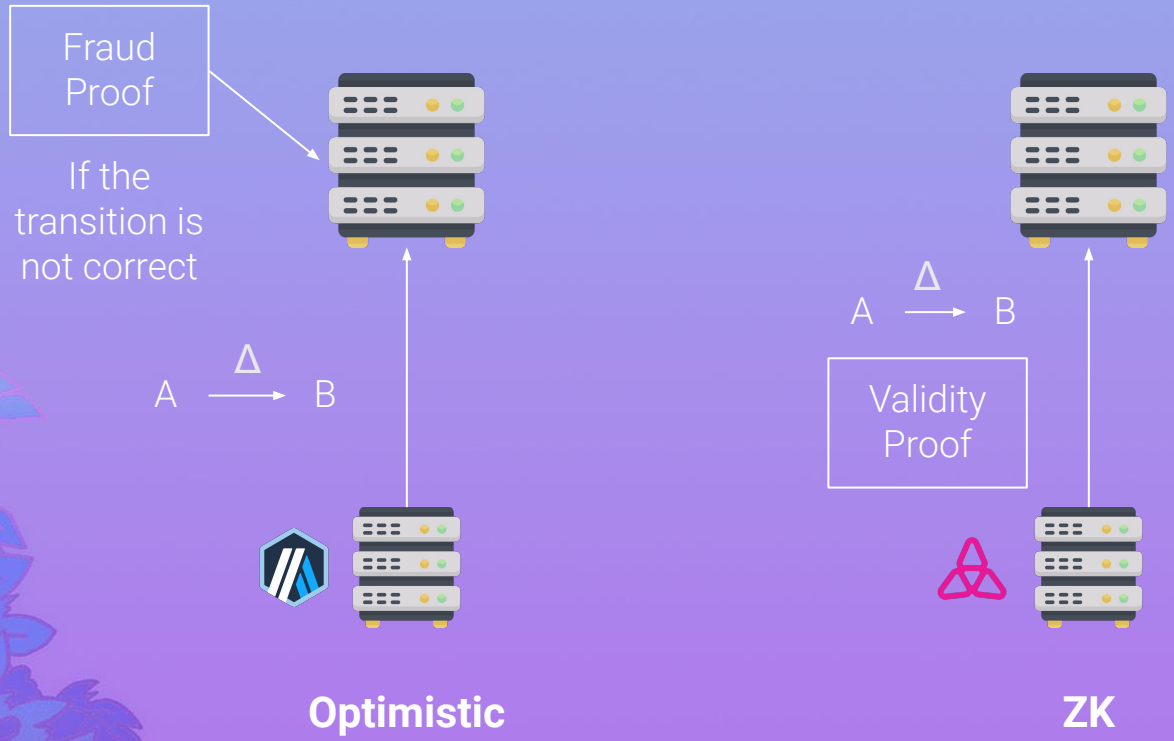
Δ



The Rollup Way

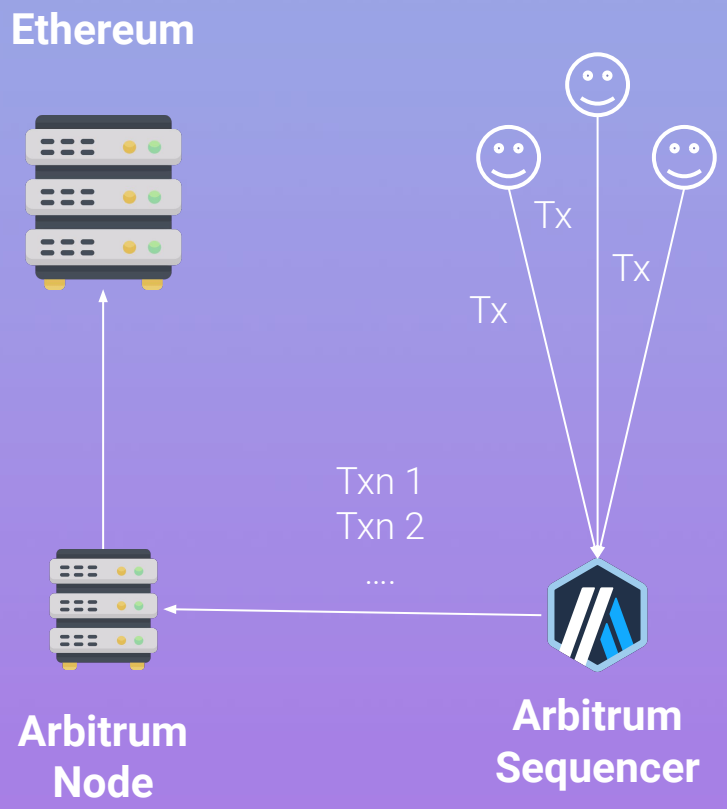


Is the transition correct?



Foundational Knowledge

Centralised Sequencing

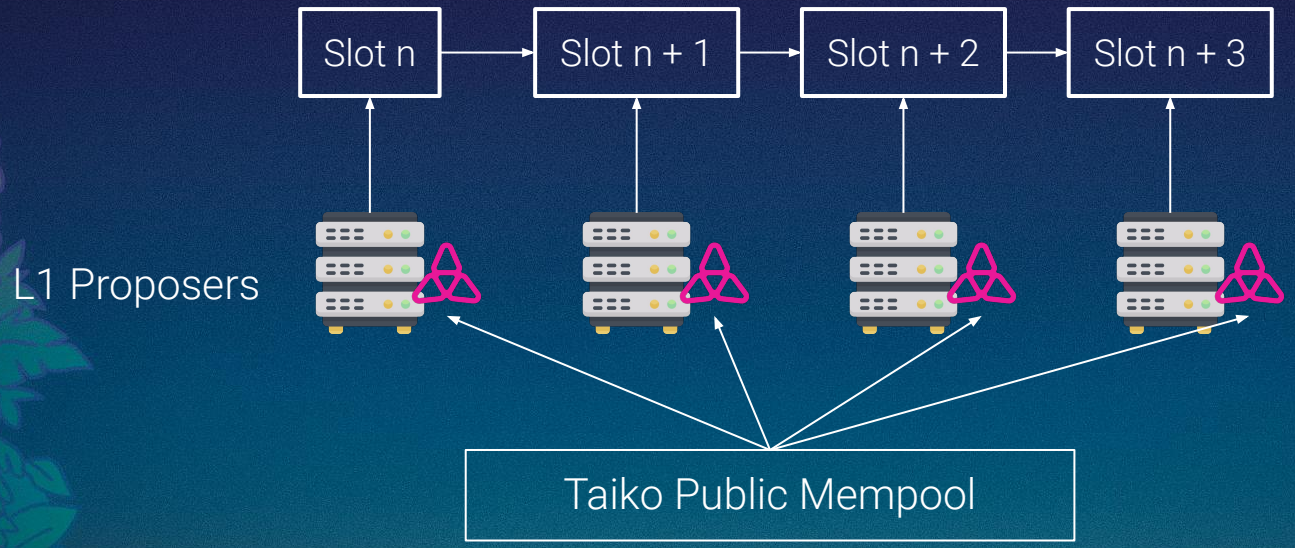


- A central party orders the transactions and pushes state transitions.
- Mempool is private.
- They “promise” to use a sequencing algorithm like FCFS.

Let’s get Based

Based Rollups and Preconfirmations

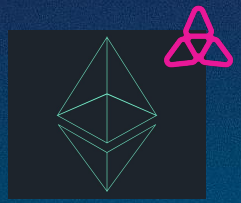
Based Sequencing



L1 Proposer orders L2 transactions and includes L2 blocks in their L1 block.

L1 Security + L1 Liveness

Taiko Overview



Taiko geth

Mempool

L2 chain

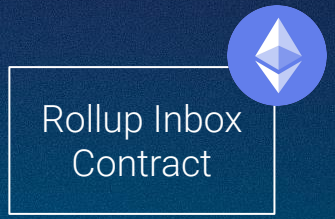
EVM

Taiko Prover

Taiko Proposer

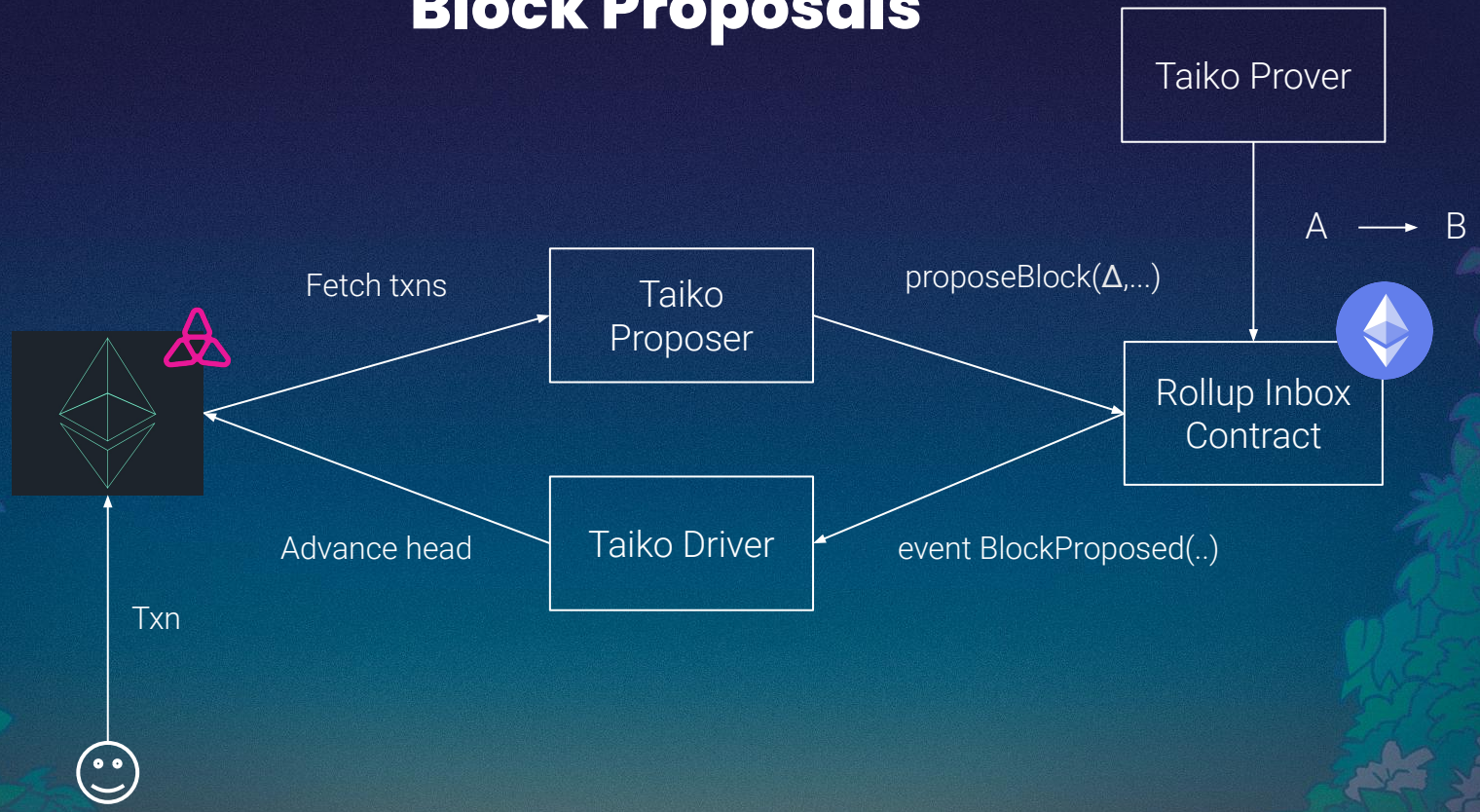
Taiko Driver

Taiko client

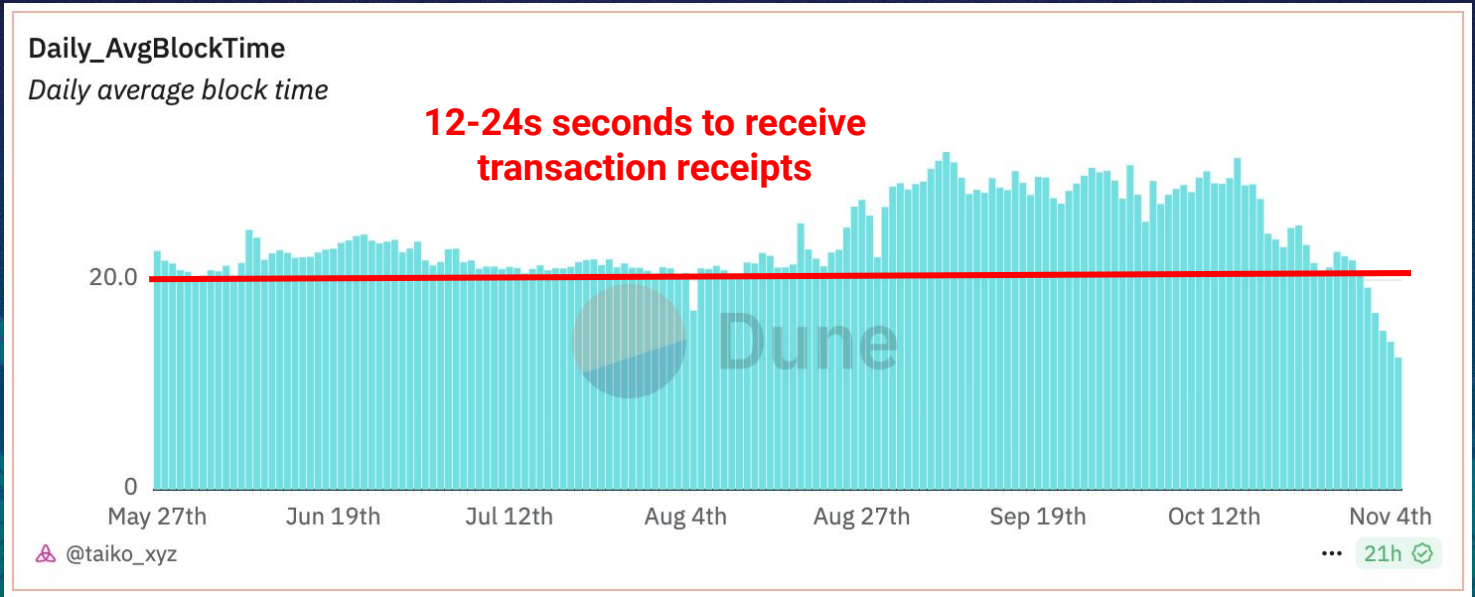


Taiko contracts

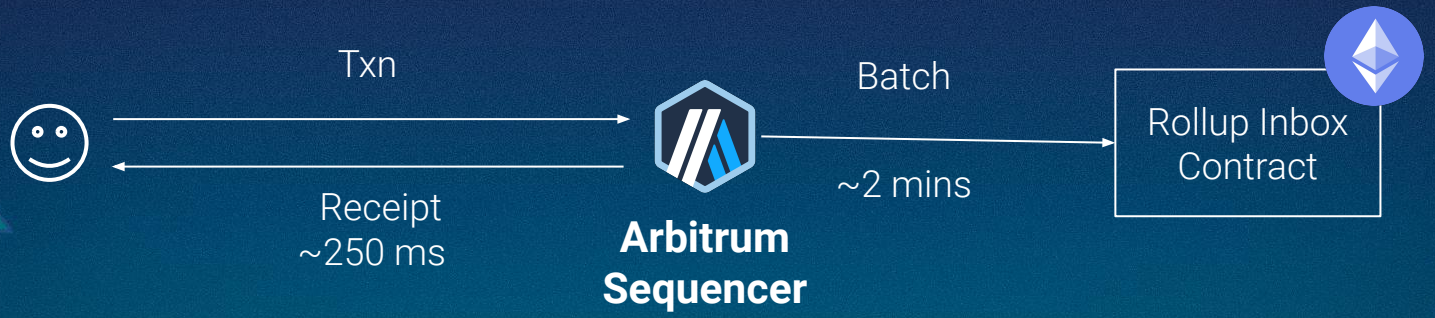
Block Proposals



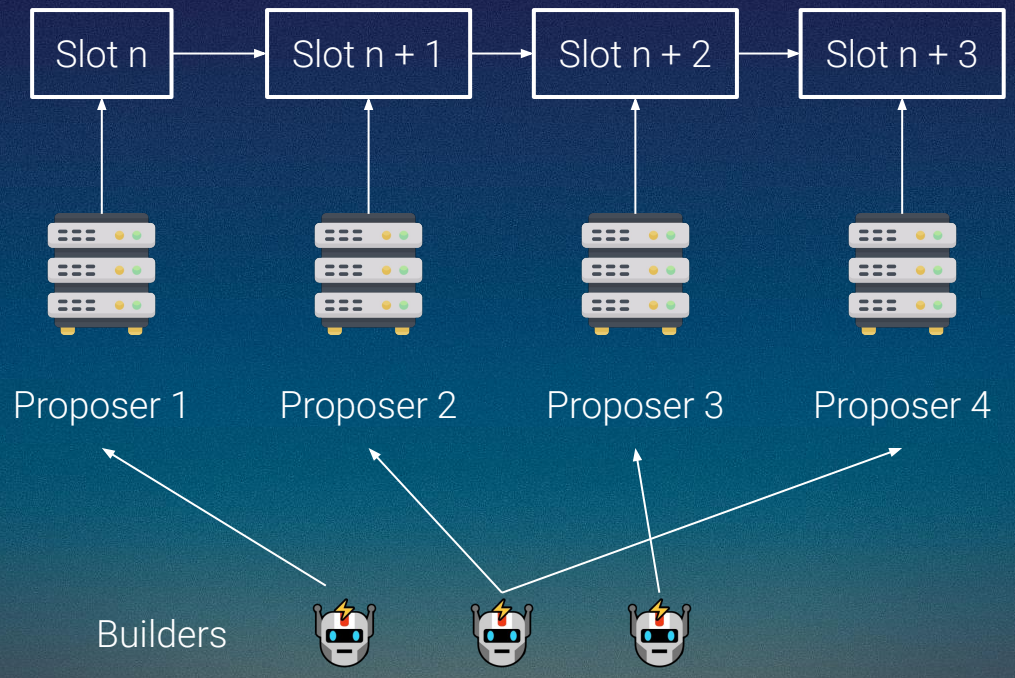
Problem? Transactions take too long



We need Preconfirmations



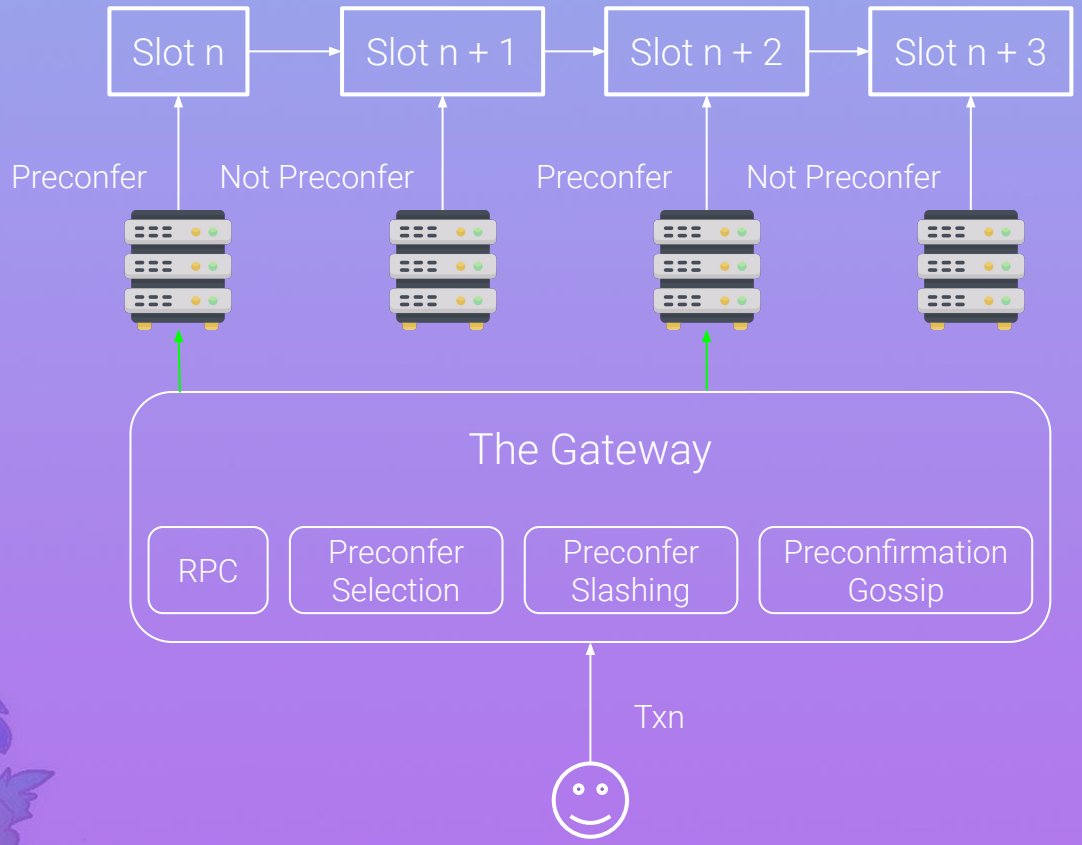
Based Preconfirmations are Tricky



Overview

Design Principles

No “Gateway”


Slot n
Slot n + 1
Slot n + 2
Slot n + 3
Preconfer
Not Preconfer
Preconfer
Not Preconfer
The Gateway
RPC
Preconfer Selection
Preconfer Slashing
Preconfirmation Gossip
Txn

No New Transaction Types

```
{
    ...EIP-1559 fields,
    deadline,
    inclusion_preconf_fee_premium,
    inclusion_preconf_base_fee_per_gas,
}
```

```
{
    ...EIP-1559 fields,
    deadline,
    execution_preconf_fee_premium,
    execution_preconf_base_fee_per_gas,
}
```

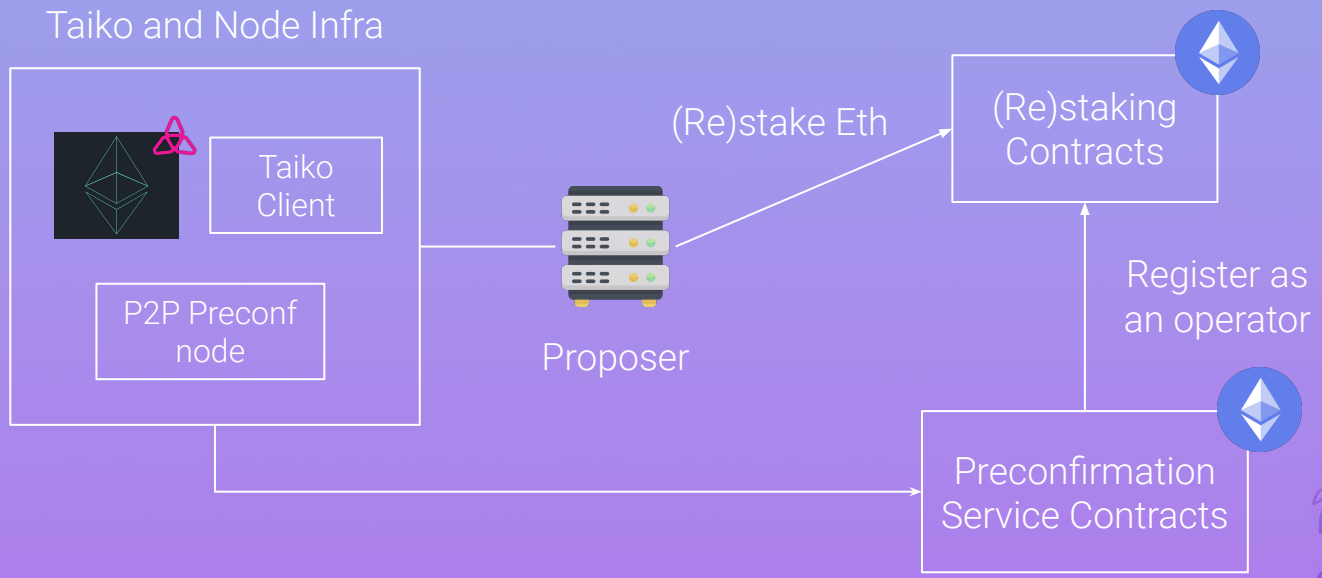
```
{
    ...EIP-1559 fields,
    priority_fee,
}
```

Use existing priority fee for preconf tips

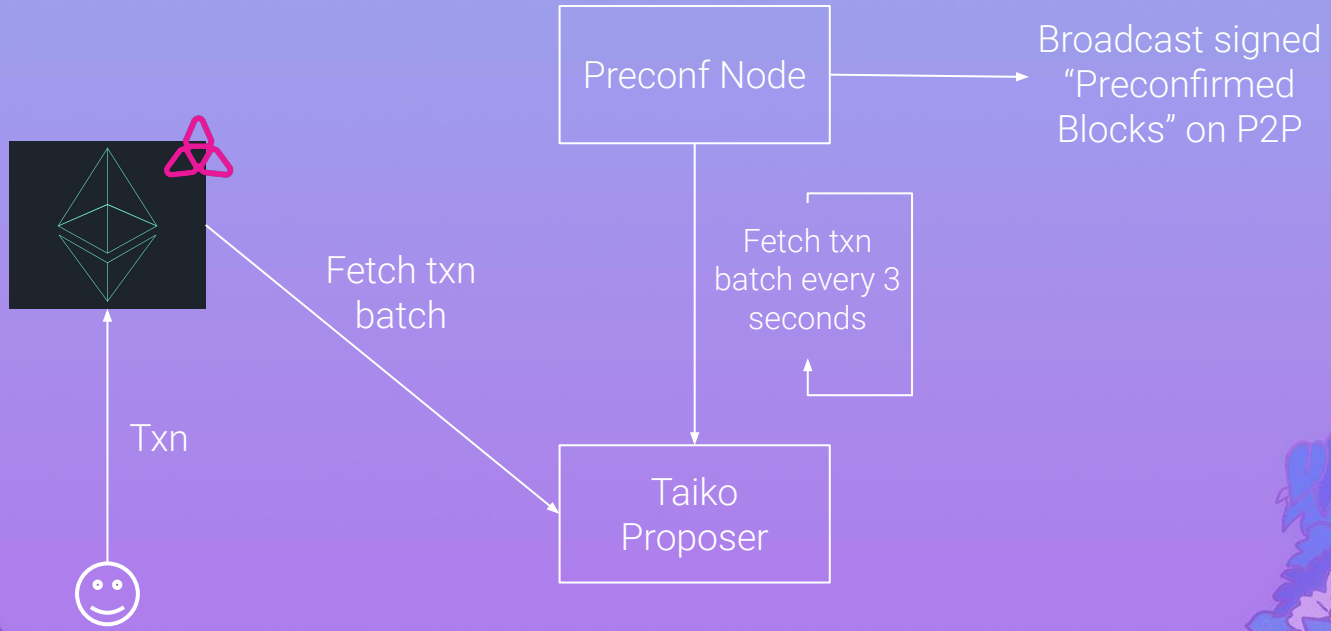
Overview

Taiko POC Design

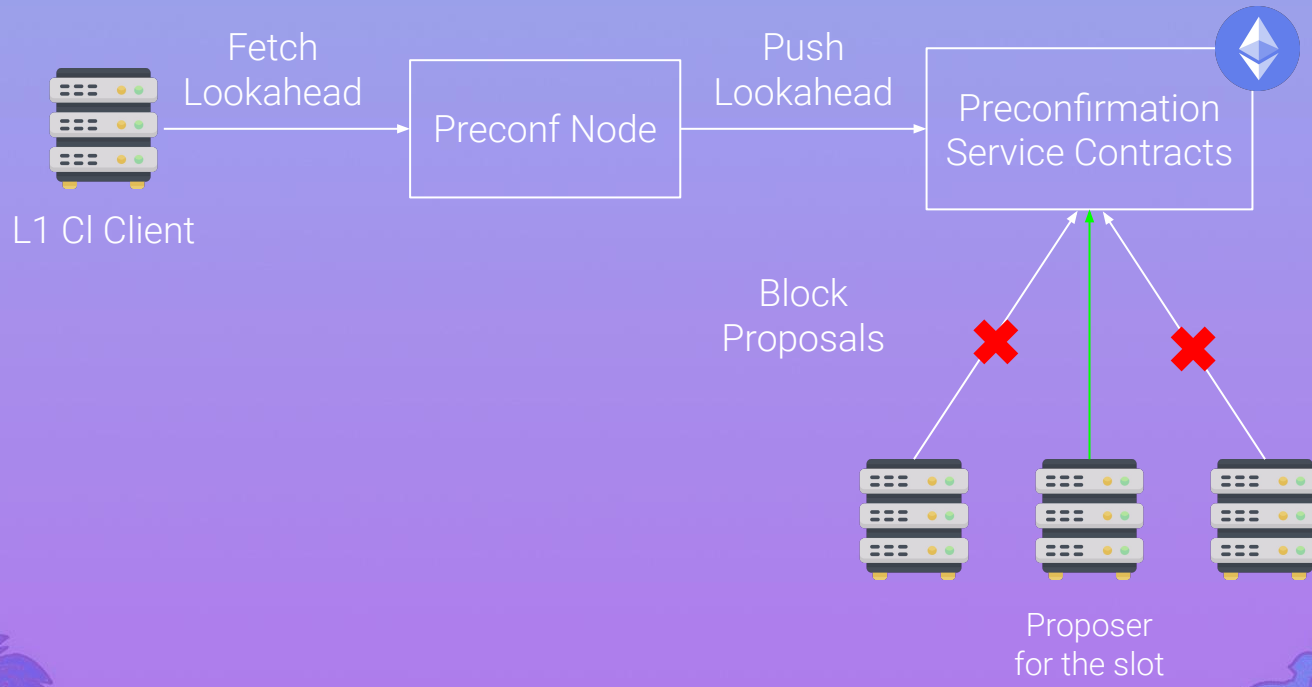
Preconfirmations as a Sidecar



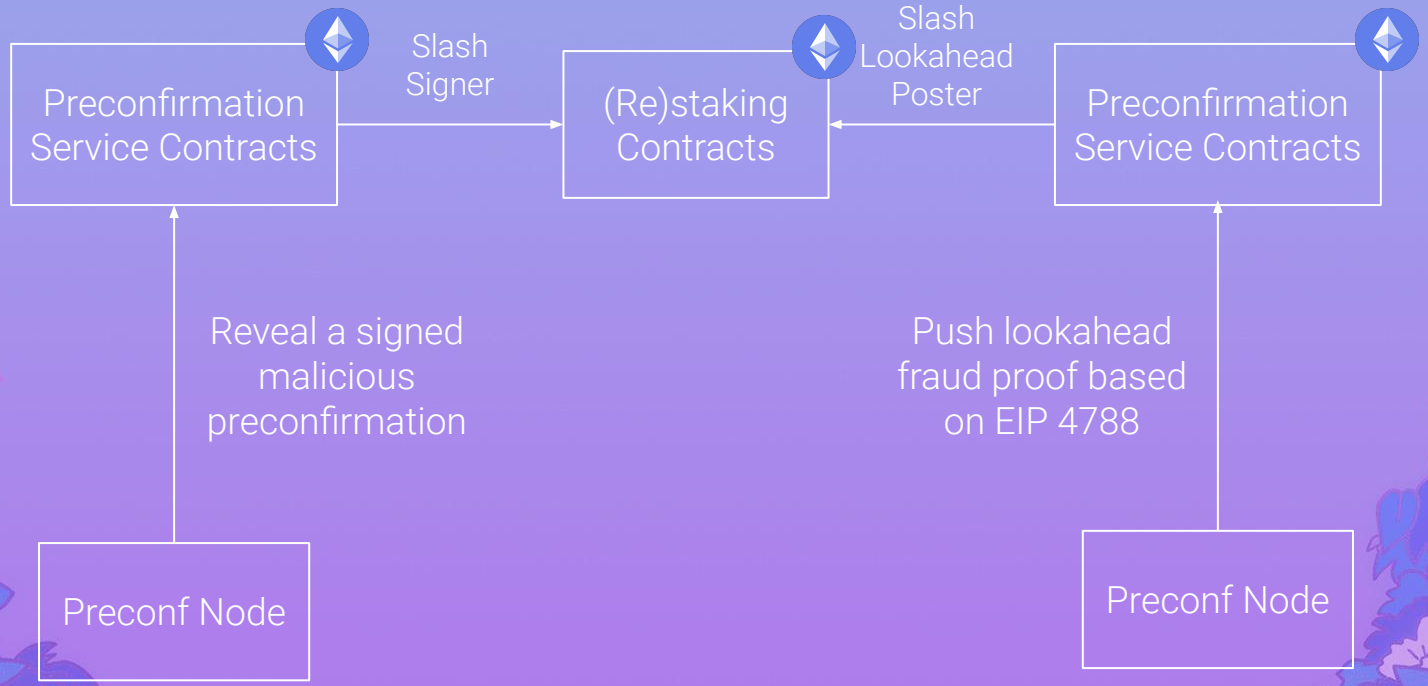
Preconfirmation Loop



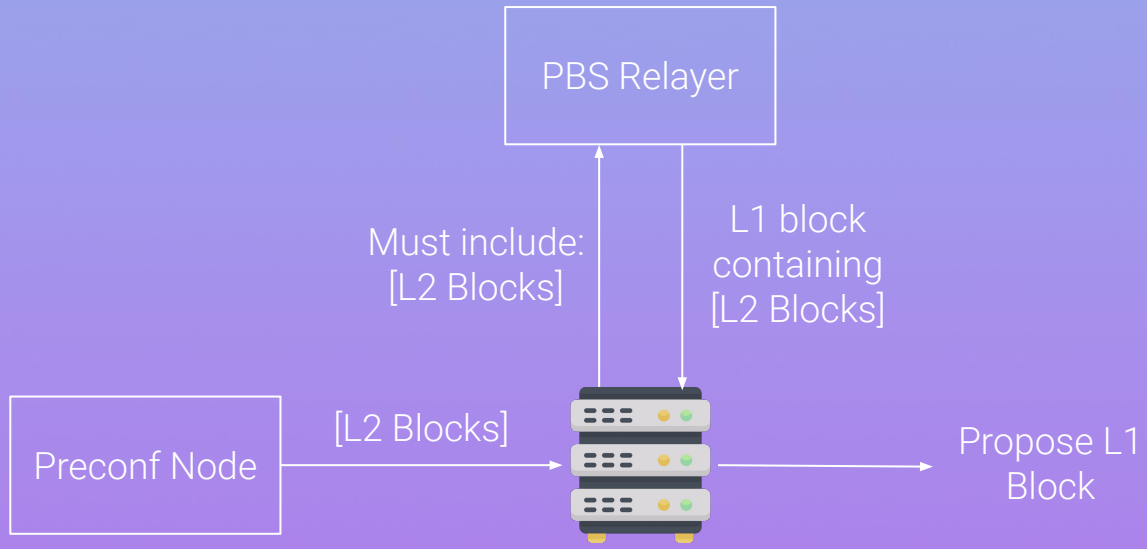
Permissionless Preconfer Selection



Permissionless Slashing



Forced Inclusions on L1

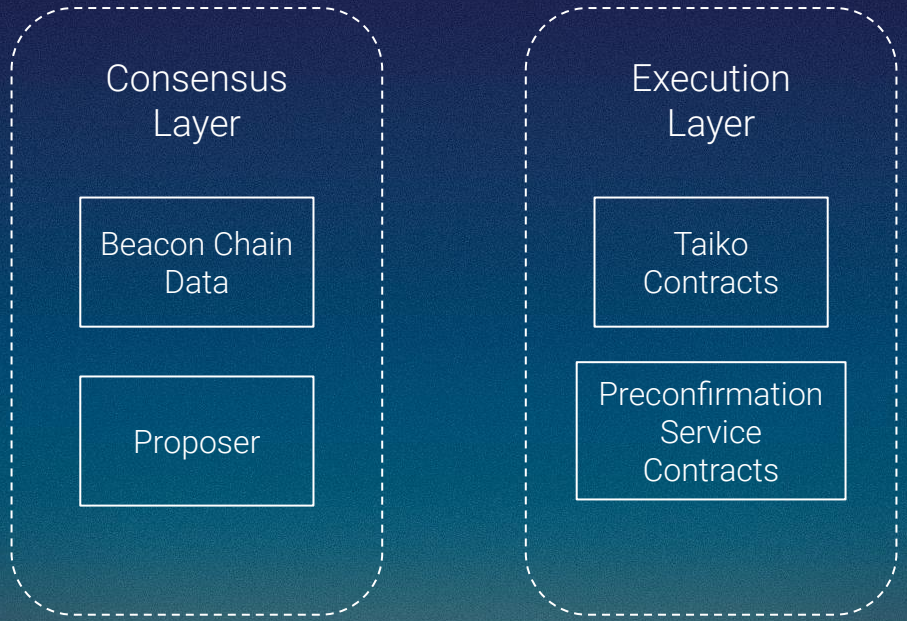


Mod 1

Preconfer Selection

Goal: Only allow current L1 proposer to propose L2 blocks

The BLS-ECDSA Problem



BLS Scheme **ECDSA Scheme**

BLS to ECDSA Mapping

Preconfirmation Registry

Preconfirmation Service Manager

Preconfirmation Task Manager

Prove ownership of a BLS public key

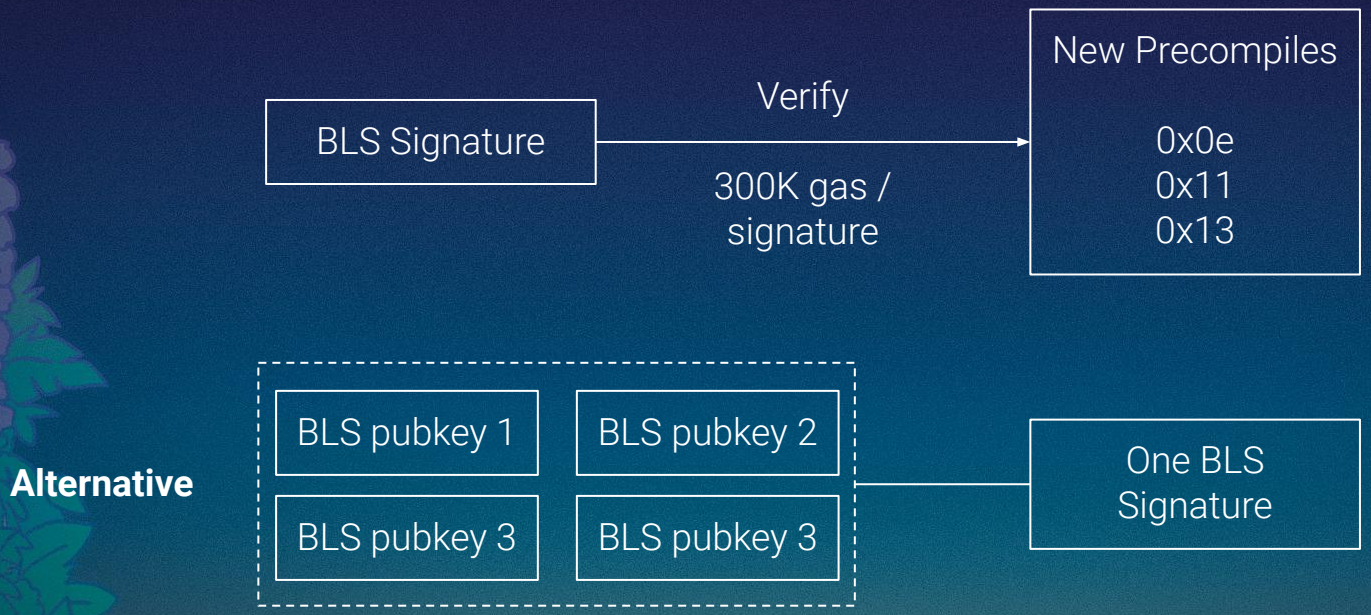
ECDSA Address

Sign using proposer's BLS key:

```
function _createMessage(ValidatorOp validatorOp, uint256 expiry, address preconfer)
    internal
    view
    returns (bytes memory)
{
    return abi.encodePacked(block.chainid, validatorOp, expiry, preconfer);
}
```

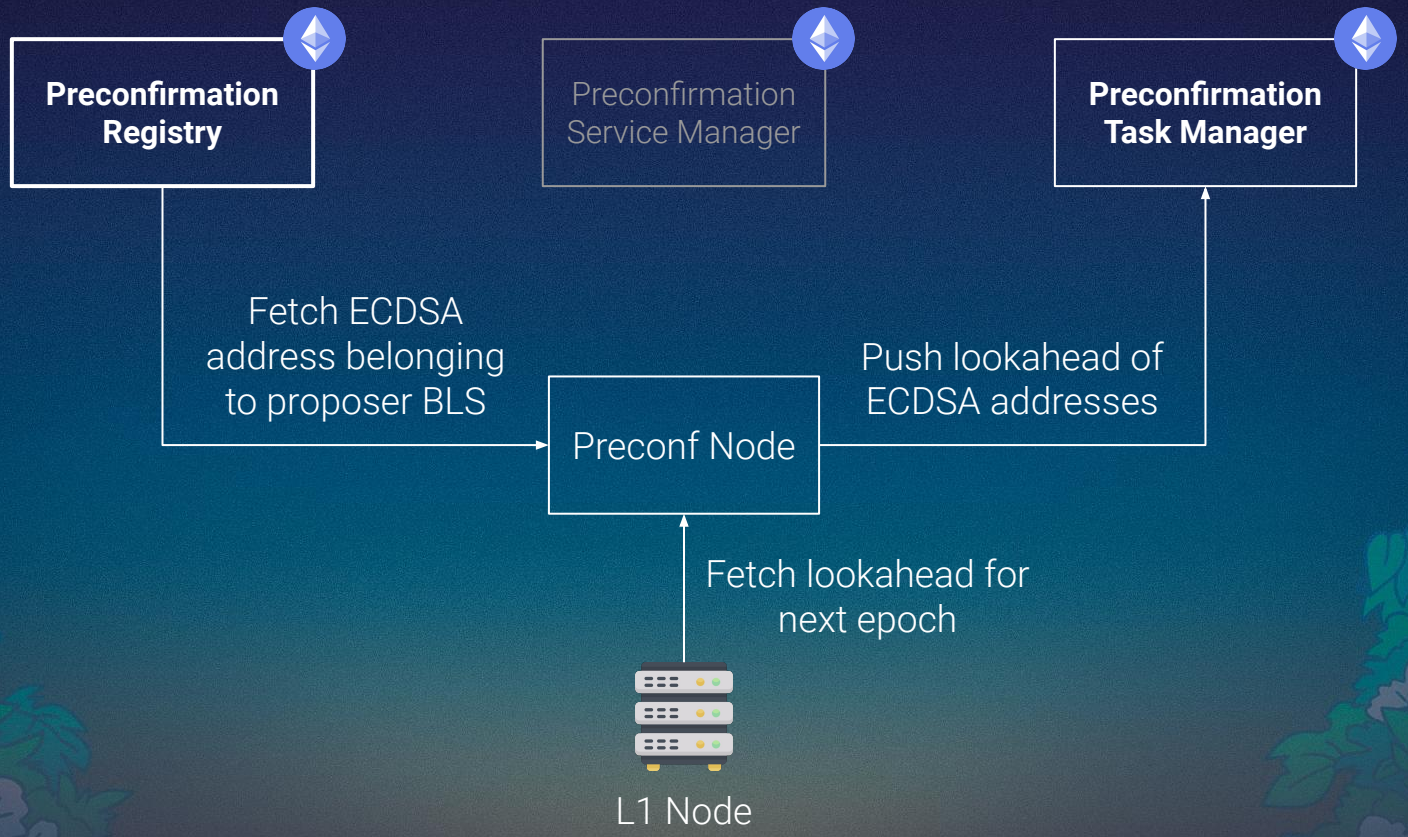
Code at: SmartContracts/src/avs/PreconfRegistry.sol

BLS Verification using EIP-2537



Code at: SmartContracts/src/avs/utils/BLSSignatureChecker.sol

Lookahead Construction

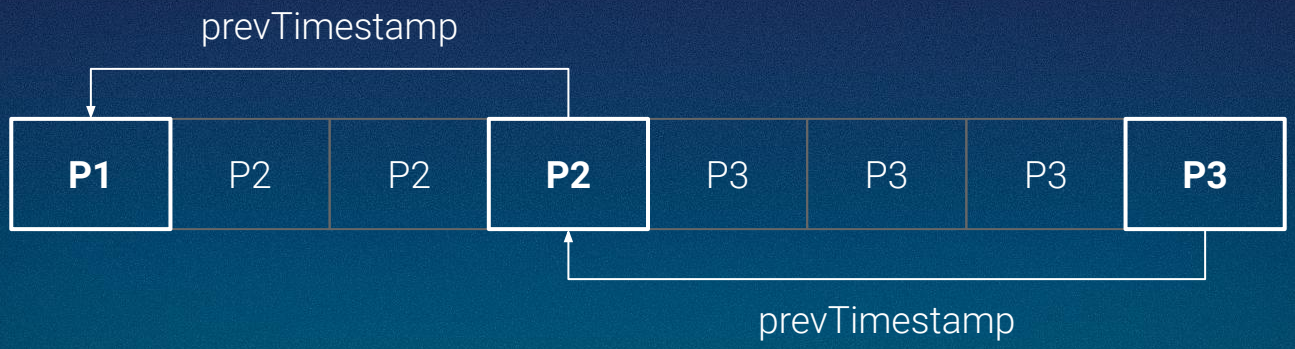


Lookahead Structure

```
struct LookaheadBufferEntry {
    // True when the preconfer is randomly selected
    bool isFallback;
    // Timestamp of the slot at which the provided preconfer is the L1 validator
    uint40 timestamp;
    // Timestamp of the last slot that had a valid preconfer
    uint40 prevTimestamp;
    // Address of the preconfer who is also the L1 validator
    // The preconfer will have rights to propose a block in the range (prevTimestamp, timestamp]
    address preconfer;
}
```

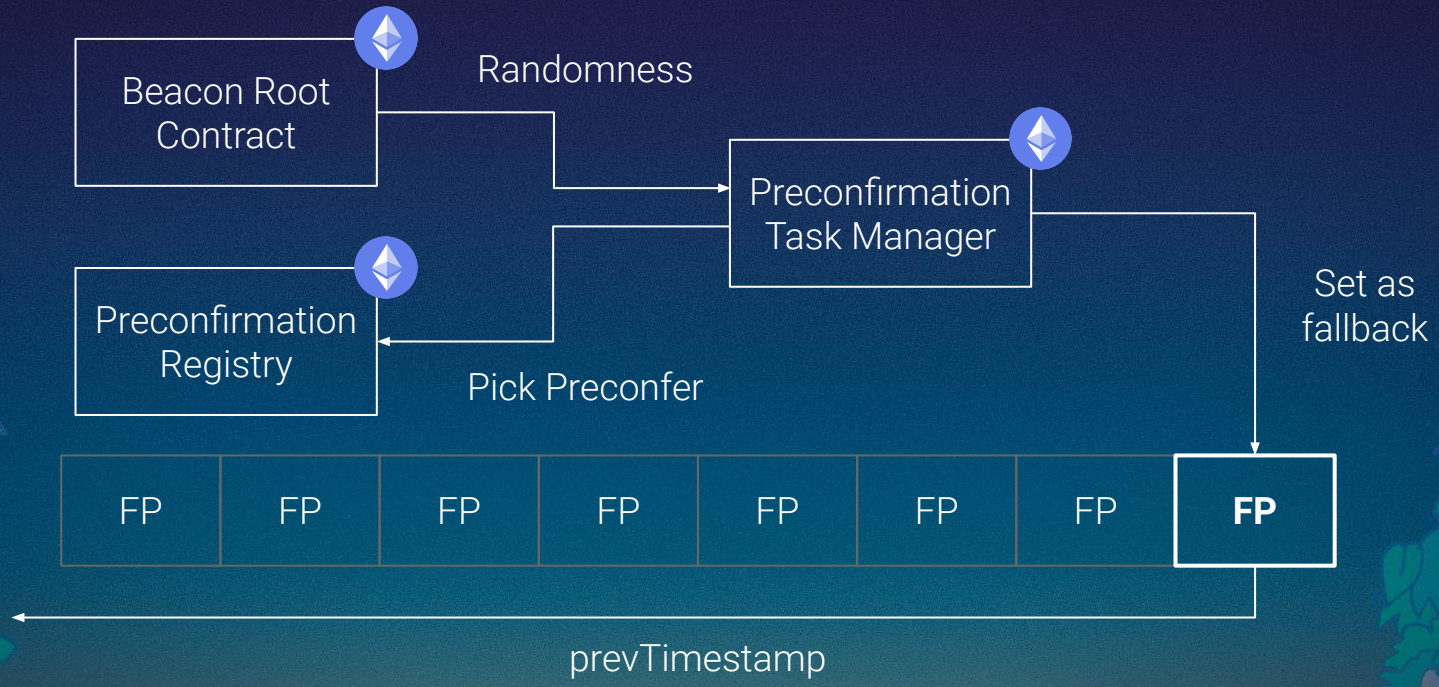
Code at: SmartContracts/src/avs/PreconfTaskManager.sol

LinkedList allows Advanced Proposals



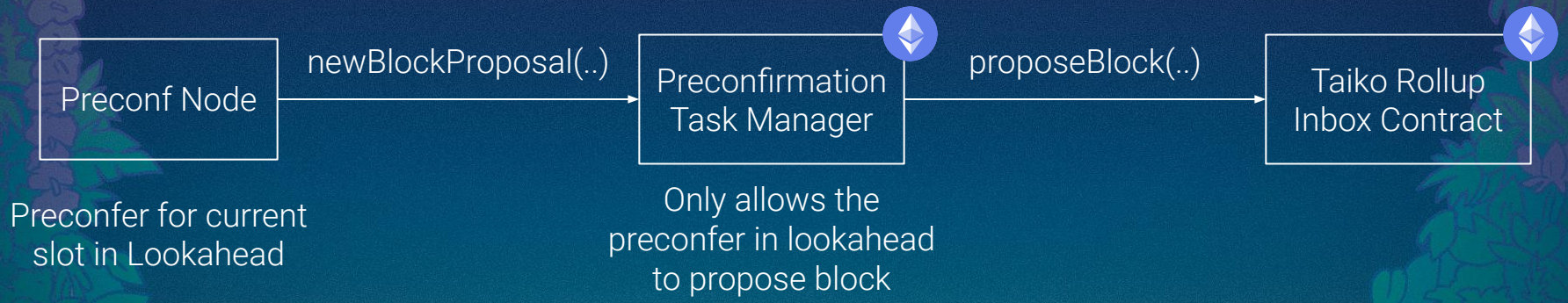
Code at: SmartContracts/src/avs/PreconfTaskManager.sol

Fallback Preconfer



Code at: SmartContracts/src/avs/PreconfTaskManager.sol

Block Proposal Routing



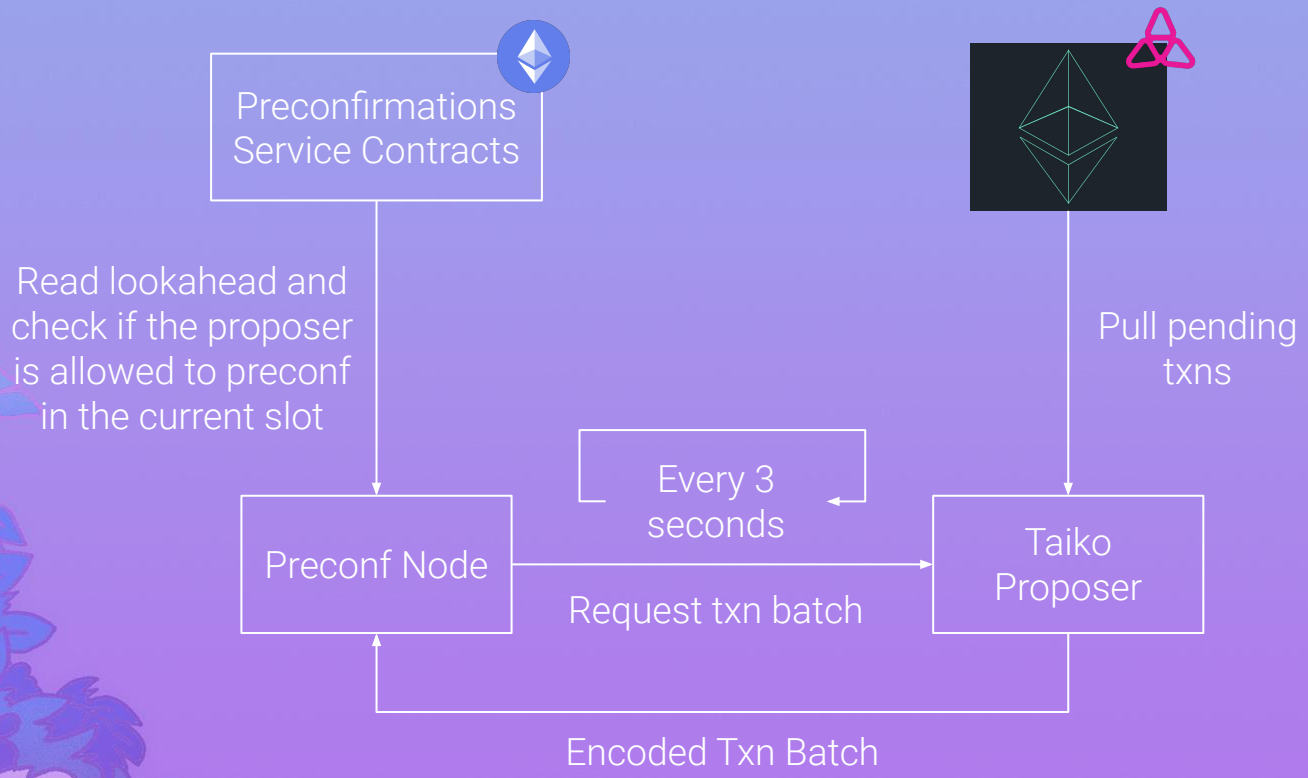
Code at: SmartContracts/src/avs/PreconfTaskManager.sol

Mod 2

Preconfirmation Loop

Previous Event Loop

New Event Loop



Signing the Preconfirmation Header

```
struct PreconfirmationHeader {
    // The block height for which the preconfirmation is provided
    uint256 blockId;
    // The chain id of the target chain on which the preconfirmed transactions are settled
    uint256 chainId;
    // The keccak hash of the RLP encoded transaction list
    bytes32 txListHash;
}
```

Code at: SmartContracts/src/avs/interfaces/IPreconfTaskManager.sol

Sending Preconfirmation Object on P2P

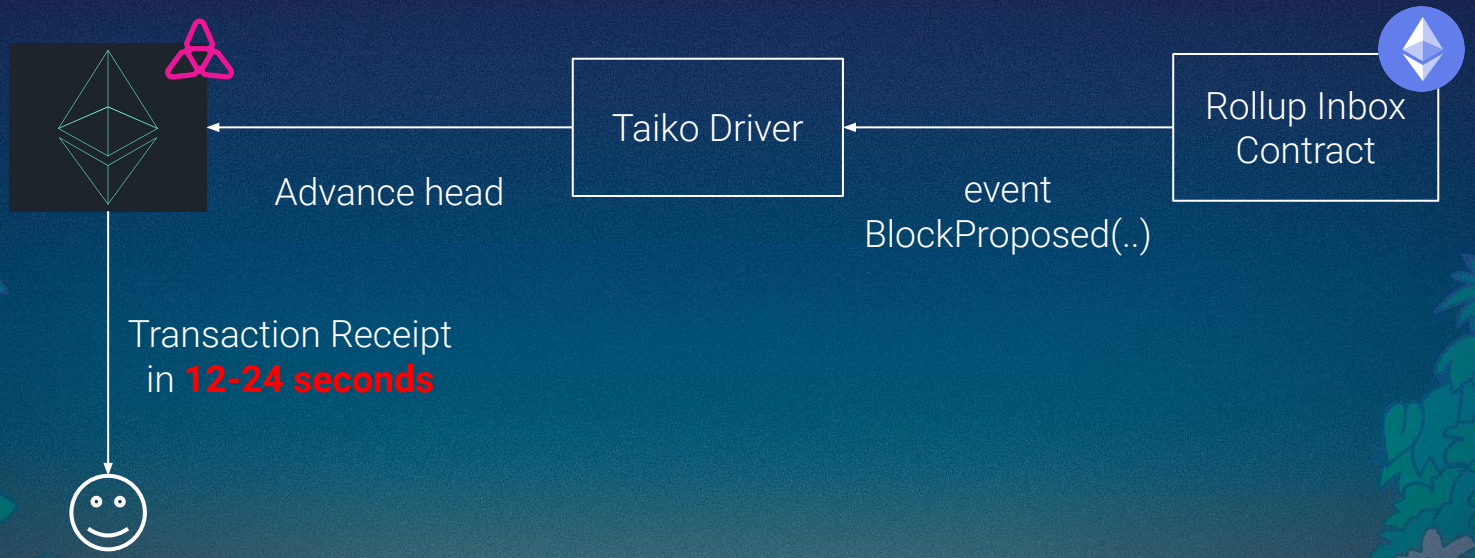
```
let preconf_message = PreconfirmationMessage::new(
    new_block_height,
    pending_tx_lists.tx_lists.clone(),
    &pending_tx_lists_bytes,
    proof.clone(),
);
self.send_preconfirmations_to_the_avs_p2p(preconf_message.clone());
```

Code at: Node/src/node/mod.rs

Mod 3

Advancing Taiko Geth Head

Previous Design



New Design - Preconfing Node


```
self.send_preconfirmations_to_the_avs_p2p(preconf_message.clone());
self.taiko
    .advance_head_to_new_l2_block(pending_tx_lists.tx_lists)
    .await?;
```

Code at: Node/src/taiko/mod.rs

New Design - Non Preconfing Node

Advance Head

Taiko Driver

Preconfirmed Txn List

Preconf Node

Preconfirmation Message

P2P

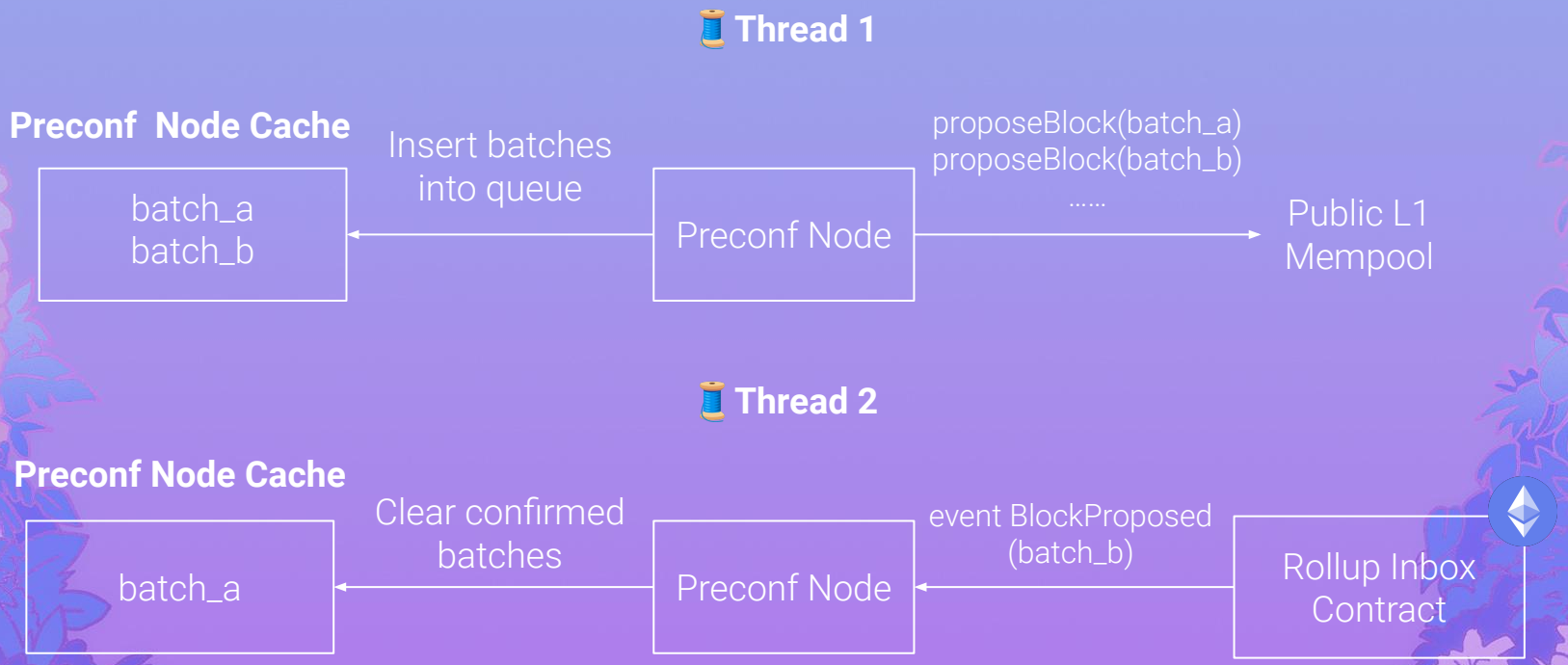
```
Some(p2p_message) = p2p_to_node_rx.recv() => {
    if !is_preconfer_now.load(Ordering::Acquire) {
        debug!("Received Message from p2p!");
        let msg: PreconfirmationMessage = p2p_message.into();
        l2_block_id.update(msg.block_height);
        Self::advance_l2_head(msg, &preconfirmed_blocks, ethereum_l1.clone(), taiko.clone()).await;
    } else {
        debug!("Node is Preconfer and received message from p2p: {:?}", p2p_message);
    }
}
```

Code at: Node/src/node/mod.rs

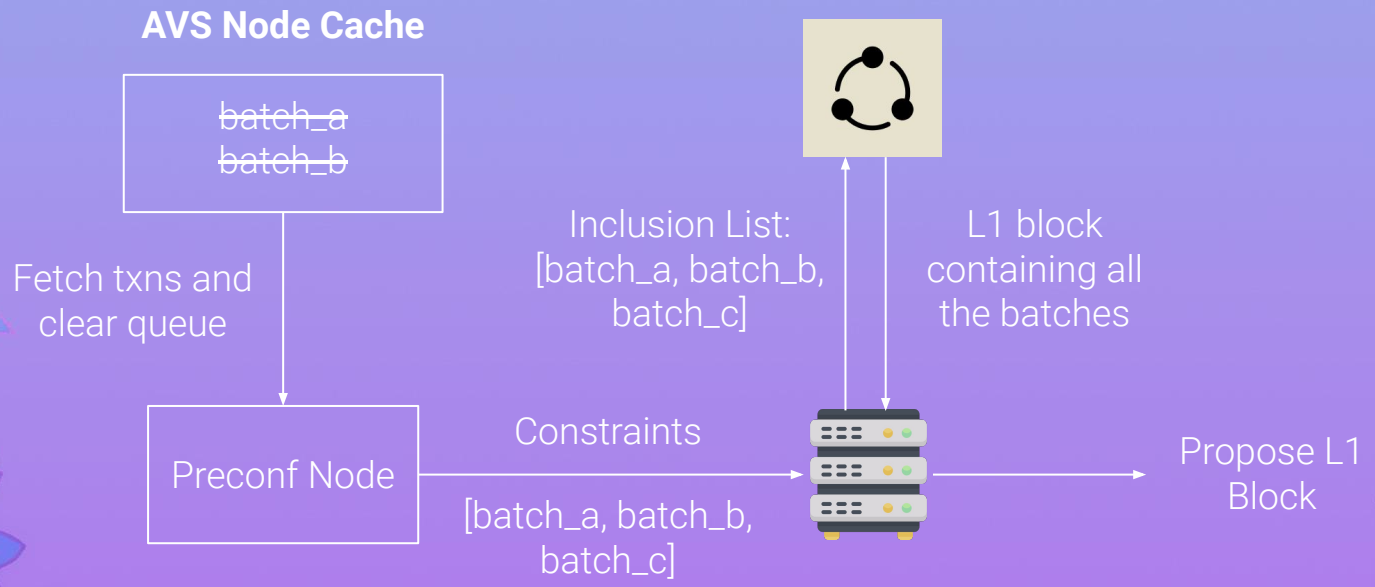
Mod 4

Forced Inclusions

Advanced Preconfing



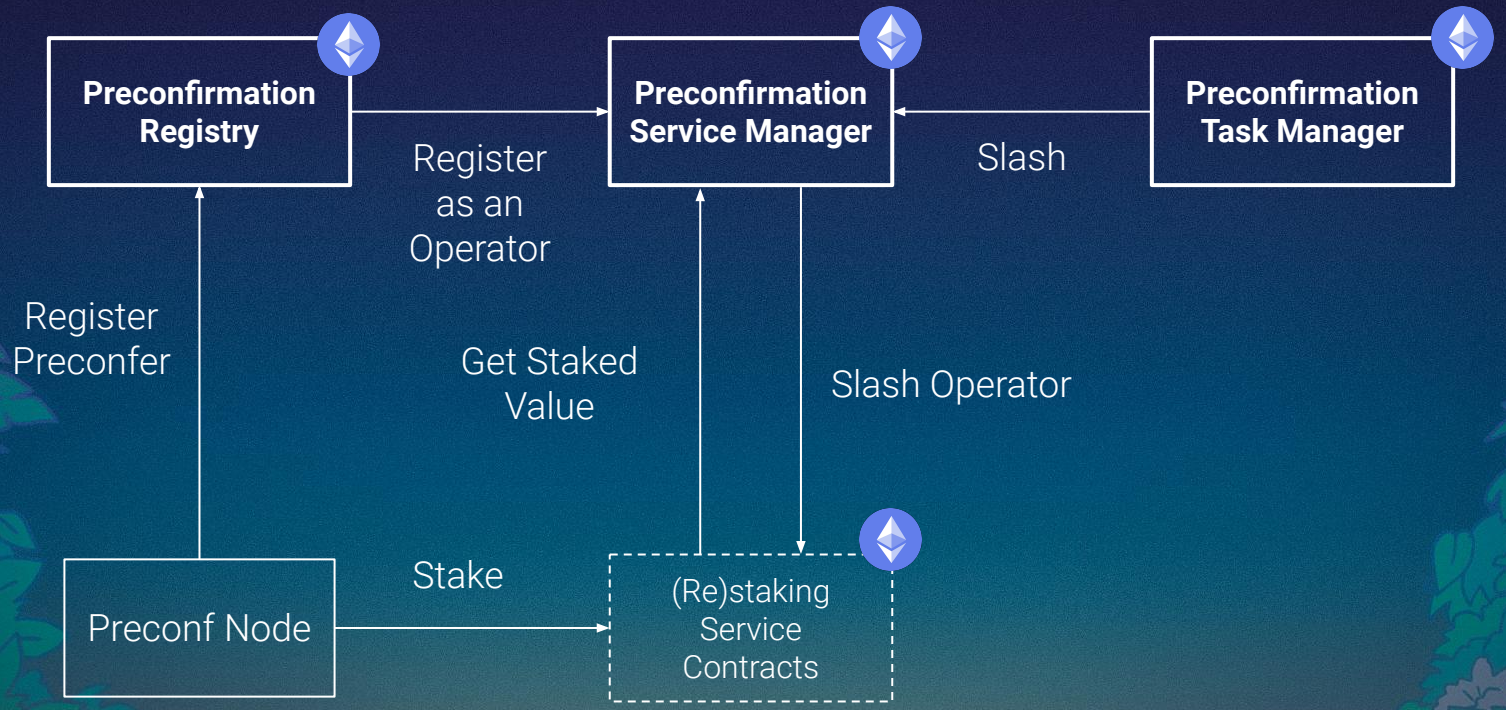
Preconfing in Proposal Slot



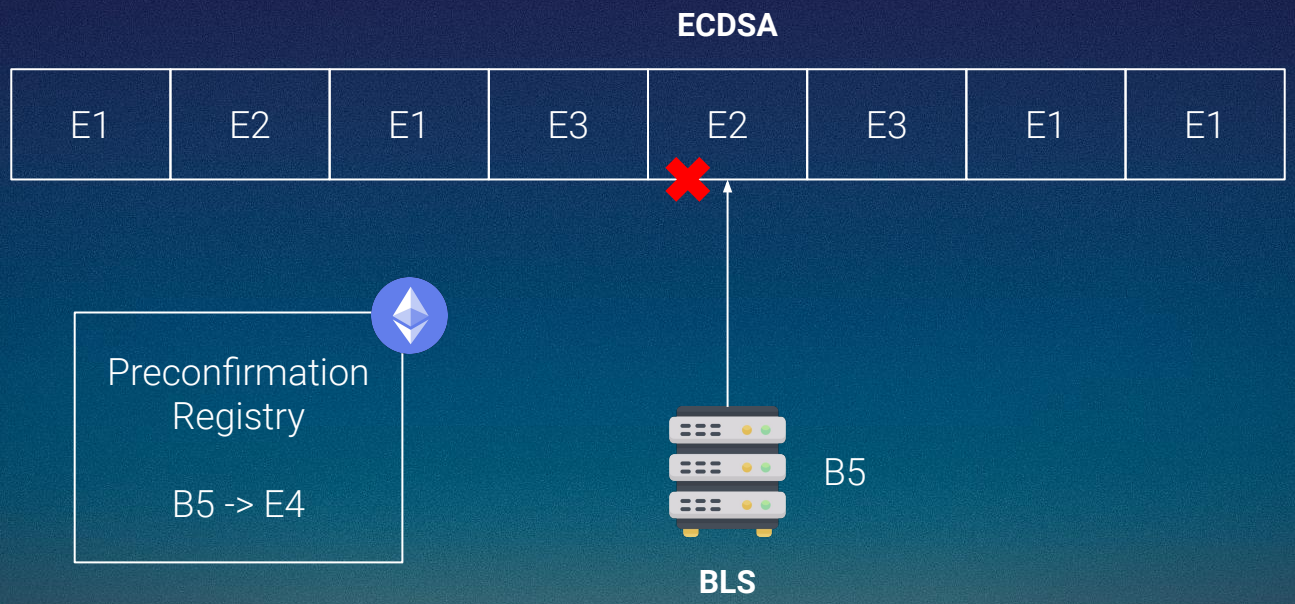
Mod 5

Staking and Slashing

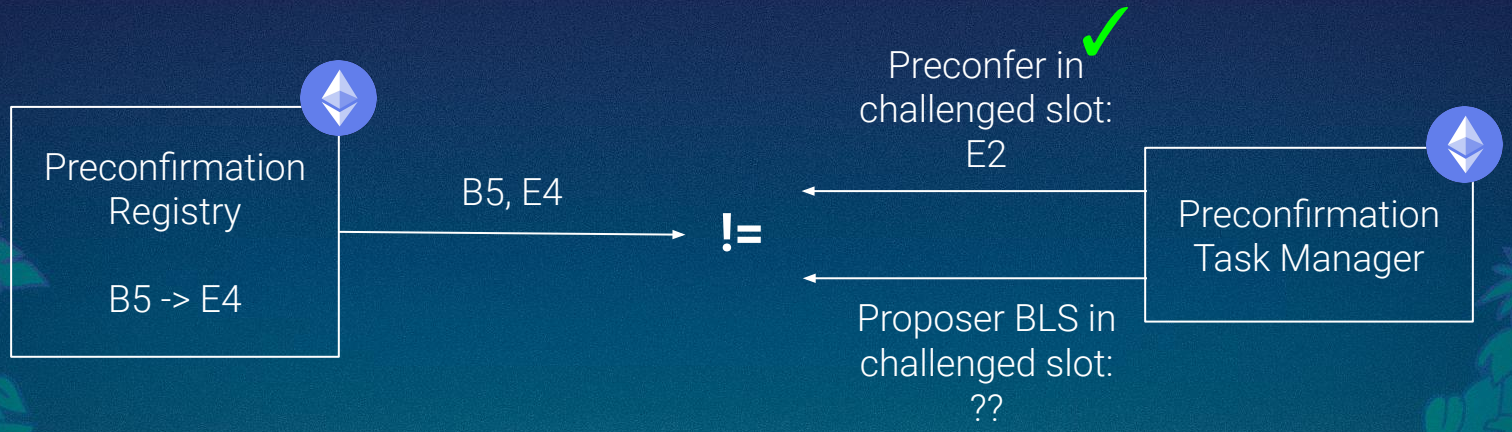
Flexible Interfacing



Slashing Incorrect Lookahead



Slashing Incorrect Lookahead



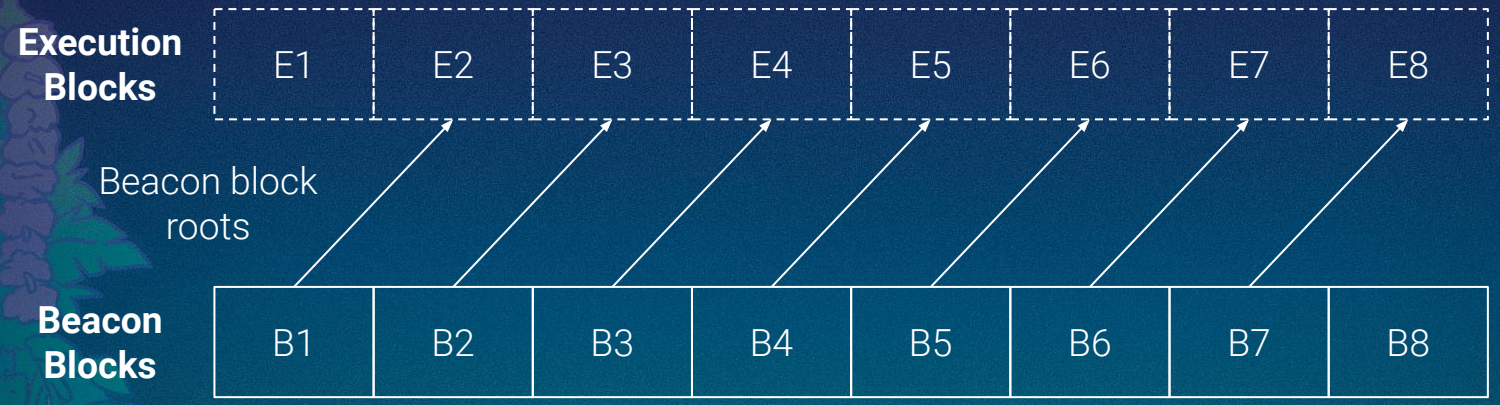
Beacon Block Root

```python
class BeaconState(Container):
    # Versioning
    genesis_time: uint64
    genesis_validators_root: Root
    slot: Slot
    fork: Fork
    # History
    latest_block_header: BeaconBlockHeader
    block_roots: Vector[Root, SLOTS_PER_HISTORICAL_ROOT]
    state_roots: Vector[Root, SLOTS_PER_HISTORICAL_ROOT]
    historical_roots: List[Root, HISTORICAL_ROOTS_LIMIT]  # Frozen in Capella,
        replaced by historical_summaries
    # Eth1
    eth1_data: Eth1Data
    eth1_data_votes: List[Eth1Data, EPOCHS_PER_ETH1_VOTING_PERIOD *
        SLOTS_PER_EPOCH]
    eth1_deposit_index: uint64
    # Registry
    validators: List[Validator, VALIDATOR_REGISTRY_LIMIT]
    #.....
    #.....
```

```python
class BeaconBlock(Container):
    slot: Slot
    proposer_index: ValidatorIndex
    parent_root: Root
    state_root: Root
    body: BeaconBlockBody
```

Merkle Root

EIP-4788



Available via a staticcall to 0x000F3df6D732807Ef1319fB7B8bB8522d0Beac02

Steps to Get BLS in Task Manager

1. Challenger sends the BLS key and index of current slot's proposer.

2. Challenger submits a merkle proof that confirms that BLS key and index are tied to the same proposer.

3. Challenger submits a merkle proof that confirms that the proposer index is present in the beacon block.

Slashing Bad Preconfirmations

Execution Preconf

Preconfirmed Block
{
blockId: 95
txListHash: **0x34526434....**
preconfer: 0x123456...
}

Proposed Block
{
blockId: 95
txListHash: **0x5565334....**
proposer: 0x123456...
}

Inclusion Preconf

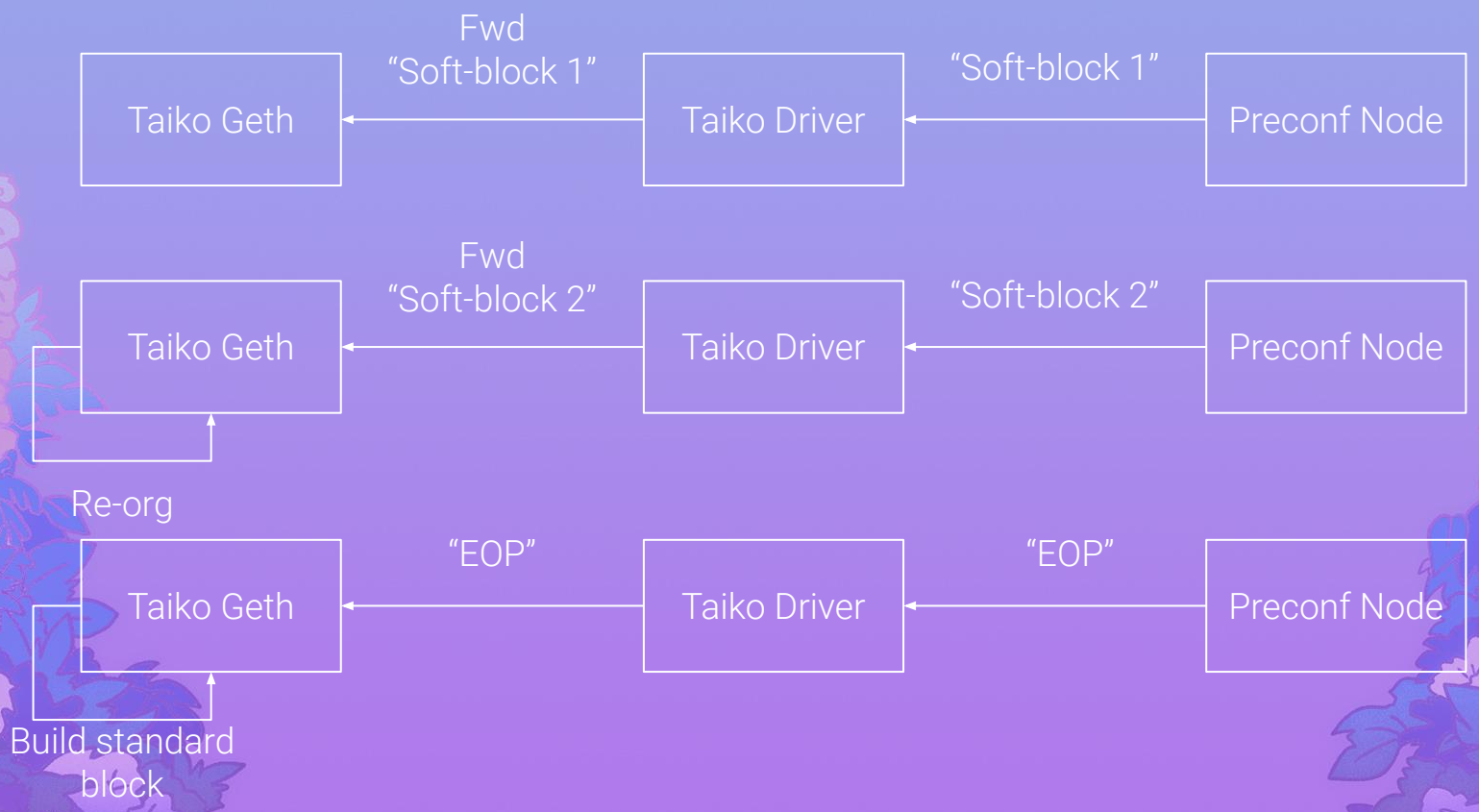
Preconfirmed Block
{
blockId: 95
txListHash: 0x34526434....
preconfer: **0x123456...**
}

Proposed Block
{
blockId: 95
txListHash: 0x34526434....
proposer: **0x746343...**
}

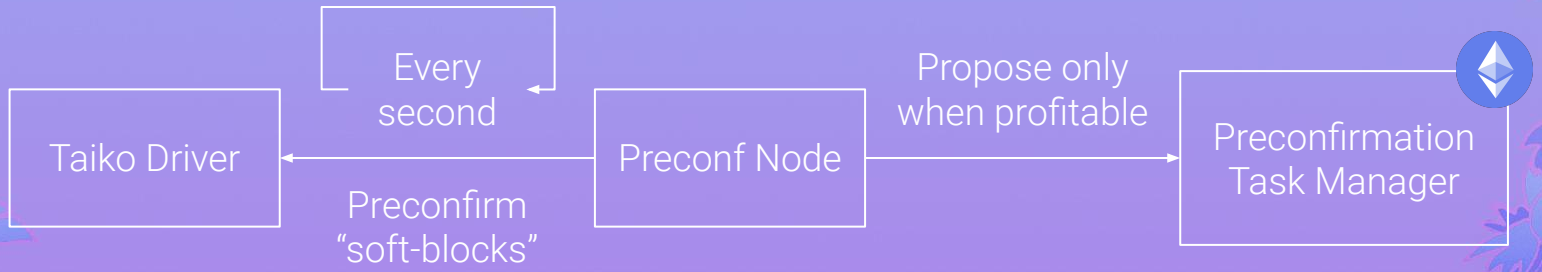
Conclusion

Future Work

“Soft Blocks”

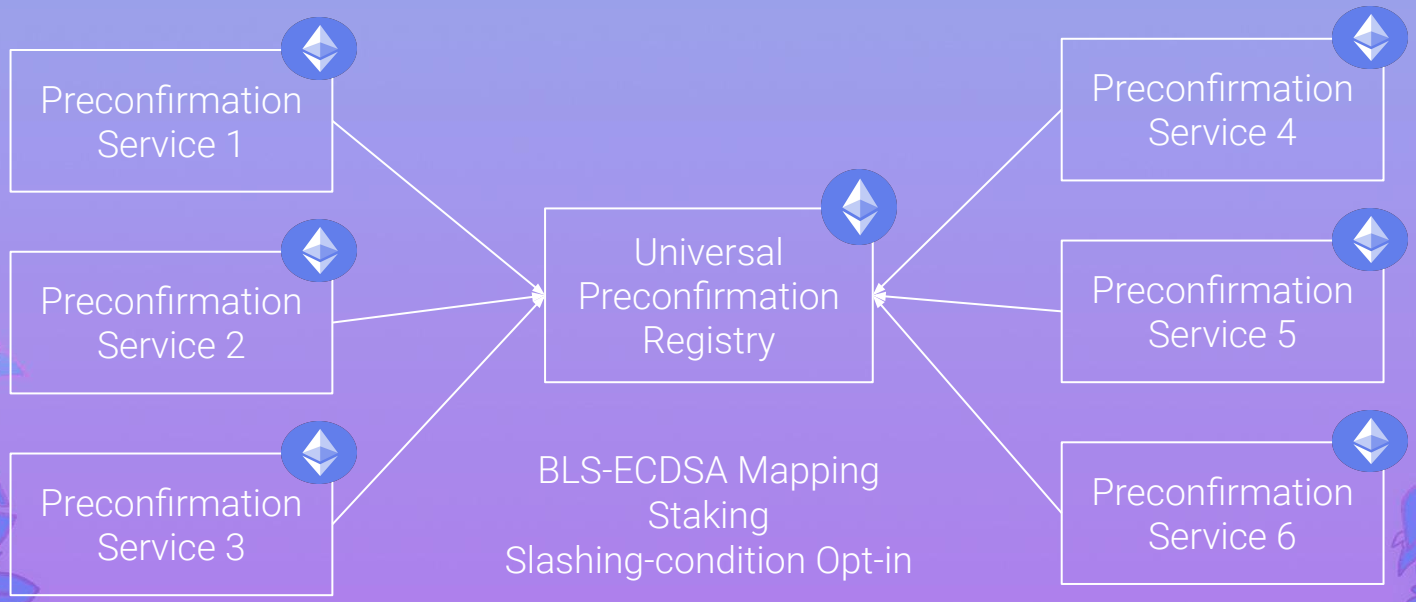

Fwd
“Soft-block 1”
Taiko Geth
Taiko Driver
“Soft-block 1”
Preconf Node
Fwd
“Soft-block 2”
Taiko Geth
Taiko Driver
“Soft-block 2”
Preconf Node
Re-org
“EOP”
Taiko Geth
Taiko Driver
“EOP”
Preconf Node
Build standard
block

1s Block Time



API at: https://taikoxyz.github.io/taiko-mono/#/default/post_softBlocks

Universal Preconfirmation Registry




Thank you!