A Sovereign Peer-to-Peer Digital Money System

Suraj Datir
surajdatir3@gmail.com
www.sahyadri.io

Abstract. A purely peer-to-peer version of electronic cash, built as a sovereign digital money system, would allow online payments to be sent directly from one party to another without going through a financial institution. Existing proof-of-work networks solve the double-spending problem but limit throughput to a single linear chain, discarding valid blocks when multiple miners discover them simultaneously. We propose a solution based on a directed acyclic graph structure, which accepts every valid block and orders them into a single deterministic history through a novel consensus protocol. The network is secured by a memory-hard proof-of-work algorithm that compresses the gap between specialized and general-purpose hardware. State is managed through an Account + Object model, enabling direct balance transfers without tracking individual unspent outputs, while a strictly capped supply with disinflationary emission ensures predictable monetary economics. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the finalized DAG ordering as proof of what happened while they were gone.

1. Introduction

Sahyadri Core introduces CSM — a sovereign digital money designed as a resilient, auditable, and highly accessible settlement layer. Sahyadri is a peer-to-peer monetary system built to enable trust-minimized value transfer at global scale while preserving simplicity, predictability, and broad participation. The protocol focuses on secure, low-friction digital money rather than general-purpose programmability. Sahyadri is implemented on SahyadriDAG, a directed acyclic graph (DAG) data structure that permits parallel block creation by independent miners. Unlike traditional single-chain blockchains that discard concurrent blocks as orphans, SahyadriDAG embraces them: every valid block contributes to the ledger, and the Sahyadri Consensus protocol produces a single deterministic total ordering of all transactions.

This eliminates the wasted-hash problem that plagues single-chain designs. The network is secured by SahyadriX — an application-layer Proof-of-Work combining Blake3 (super-speed hashing) with an 8-stage XOR memory loop requiring 16 MB of random-access memory per hash operation. This architecture makes SahyadriX genuinely memory-hard, compressing the performance gap between ASICs and general-purpose hardware (GPUs, CPUs) ensuring mining remains accessible to a broad population of participants.

Sahyadri operates on a hybrid Account + Object state model. Rather than tracking individual unspent outputs (UTXOs), the protocol maintains per-address balances and nonces. Each transaction atomically deducts from the sender and credits the receiver, with balance finality enforced at the block confirmation boundary. This model delivers account + object based ergonomics with the security of Proof-of-Work.

1.1 Key Properties

Property	Specification
Ticker Symbol	CSM (Cryptographic Sovereign Money)
Smallest Unit	Kana (1 CSM = 100,000,000 Kana, 8
Maximum Supply	21,000,000 CSM (hard cap, protocol-
Block Time	1 second (deterministic)
Throughput	10,000 TPS (max_block_mass: 30,000,000)
Initial Block Reward	0.08318123 CSM per block (8,318,123
Halving Interval	Every 4 years (126,230,400 blocks)
Block Reward Split	98% Miner / 2% Development Fund
TX Fee Split	90% Miner / 10% Development Fund
Minimum TX Fee	0.00001 CSM (1,000 Kana) — fixed flat
Consensus	Sahyadri Consensus
Mining Algorithm	SahyadriX (Blake3 + 16 MB memory
State Model	Account + Object (balance + nonce per
Address Format	CSM32 (Bech32-derived, prefix: csm1...)
Network Ports	gRPC: 26110 | P2P: 26111 | wRPC: 27110

2. Account + Object State Model

Sahyadri employs a hybrid Account + Object state model stored in RocksDB. Unlike pure UTXO systems, which track individual unspent outputs, the account model maintains a global state table where each address has an associated balance (in CSM) and a monotonically increasing nonce. This eliminates the "dust" problem inherent in high-frequency UTXO systems at 1-second block times.

2.1 Account State

Each account entry in RocksDB contains:

- address: CSM32-encoded public key hash (csm1...)
- balance: Current spendable balance in CSM (8 decimal precision)
- nonce: Transaction sequence number (prevents replay attacks)

When a transaction is confirmed in a block, the state transition is:

```
sender.balance    -= (amount + fee)
receiver.balance  += amount
miner.balance     += fee * 0.90   // 90% of fee
company.balance   += fee * 0.10   // 10% of fee
sender.nonce      += 1
```

2.2 Object Model

The Object layer complements accounts by storing arbitrary state commitments in a Merkle trie rooted in each block header. Each block header commits to an object-based state root — a hash that cryptographically summarises the entire current state of the ledger. This enables:

- Efficient light-client verification via Merkle proofs
- Safe pruning: once a state root is committed, older block bodies may be discarded
- Fast sync: new nodes download a verified state snapshot rather than replaying all history

2.3 Why Not Pure UTXO?

At 1-second block times and 10,000 TPS, pure UTXO models generate millions of small outputs ("dust") that fragment user balances. The account model resolves this: each address maintains a single unified balance. Atomic state transitions at block confirmation boundaries ensure that partial updates are impossible — either the full transaction succeeds or the state remains unchanged.

3. Transactions

A Sahyadri transaction is a cryptographically signed state-transition instruction. It authorizes a transfer of value from a sender account to a receiver account, subject to nonce validation, balance sufficiency, and fee payment. Transactions are pending until included in a miner-confirmed block; only then do balance changes take effect.

3.1 Transaction Structure

```
Transaction {
  sender:    CSM32 address          // csm1q...
  receiver:  CSM32 address          // csm1q...
  amount:    u64 (in Kana)          // 1 CSM = 100,000,000 Kana
  fee:       u64 (in Kana)          // fixed: 1,000 Kana = 0.00001 CSM
  nonce:     u64                    // must equal sender.current_nonce
  tx_id:     SHA256(sender+receiver+amount+nonce+timestamp)
}
```

3.2 Full Transaction Lifecycle

The following diagram describes the complete lifecycle from user initiation to finalized state:

```
USER INITIATES TRANSACTION
  |
  +--> wallet_api: POST /api/send-csm
         |
         +--> [VALIDATION]
         |      |-- Sender account exists?
         |      |-- balance >= (amount + 0.00001 CSM fee)?
         |      |-- nonce == sender.current_nonce?
         |      |-- No pending TX would overdraft balance?
         |
         +--> [MEMPOOL INSERT]
         |      |-- pending_transactions table (status='pending')
         |      |-- transactions table       (status='pending')
         |      |-- sender.nonce += 1  (double-send prevention)
         |
         +--> Return: { txid, status:'pending', fee:0.00001 }
                              |
                              | (waiting for miner...)
                              |
MINER MINES NEXT BLOCK (1 second)
  |
  +--> indexer_grpc.py: flush_second()
         |
         +--> [BLOCK REWARD]
         |      |-- payload decoded -> actual_reward_kana
         |      |-- miner.balance += reward * 0.98
         |      |-- company.balance += reward * 0.02
         |
         +--> [PENDING TX CONFIRM] (up to 10,000 per second)
                |-- sender.balance -= (amount + fee)   [NOW deducted]
                |-- receiver.balance += amount
                |-- miner.balance += fee * 0.90
                |-- company.balance += fee * 0.10
                |-- tx status = 'confirmed', block_hash = current
                |-- If sender.balance < (amount+fee): REJECT TX
```

3.3 Mempool and Pending Queue

The mempool is implemented as a PostgreSQL table (pending_transactions) rather than an in-memory structure. This design provides:

- Persistence: pending TXs survive node restarts
- Atomic operations: PostgreSQL ON CONFLICT DO NOTHING prevents duplicate inserts
- Double-spend protection: pending locked amount is checked before accepting new TXs
- Capacity: 10,000 TX processed per 1-second block (LIMIT 10000 in confirm query)

Double-spend prevention logic in wallet_api.py:

```
# Check total locked in pending queue
SELECT COALESCE(SUM(amount + fee), 0)
FROM pending_transactions
WHERE from_address = %s AND status = 'pending'

# New TX accepted only if:
sender.balance >= pending_locked + (new_amount + fee)
```

3.4 Nonce and Replay Attack Prevention

Every account carries a monotonically increasing nonce. The wallet_api.py validates that the submitted nonce equals the current on-chain nonce. Each accepted TX increments the nonce immediately (before block confirmation), preventing replay of the same transaction:

```
# wallet_api.py: nonce check
if nonce != current_nonce:
    return error('Invalid nonce. Expected: %d' % current_nonce)

# On mempool accept:
UPDATE accounts SET nonce = nonce + 1 WHERE address = %s
```

3.5 Fee Model

Sahyadri uses a fixed flat fee of 0.00001 CSM (1,000 Kana) per transaction, regardless of amount. This is the lowest transaction fee of any major blockchain:

Method	Fee Model	Fee (for $100 transfer)
Sahyadri	Fixed flat (Kana)	0.00001CSM (constant & negligible)
Payment Aplication	Percentage-based	$1.50 - $3.00
Bank Transfer (domestic)	Flat or percentage	$1.00 - $10.00
Bank Wire (international)	Flat + hidden fees	$15.00 - $50.00
Money Transfer Service	Flat + percentage	$5.00 - $30.00

Even at CSM = $10,000 USD, the fee is only $0.10 per transaction. The fee is defined once in code and applies universally:

```
# wallet_api.py + indexer_grpc.py
TX_FEE_KANA         = 1000       # 0.00001 CSM — fixed forever
TX_FEE_CSM          = 1000 / 1e8
TX_FEE_MINER_SPLIT  = 0.90       # 90% to miner
TX_FEE_COMPANY_SPLIT= 0.10       # 10% to development fund
```

4. SahyadriX — Proof of Work

SahyadriX is the application-layer Proof-of-Work function used by the Sahyadri network. It combines Blake3 (a cryptographic hash function offering SIMD-optimized speed) with an 8-stage XOR memory loop operating over a 16 MB random-access memory pad. This construction is genuinely memory-hard: the evaluating device must maintain a 16 MB working set throughout computation, making it resistant to ASIC implementations that rely on small, fast on-chip caches.

4.1 Algorithm Design

```
// SahyadriX: Blake3 + 16MB RandomX-style Memory Loop
// File: crypto/hashes/src/pow_hashers.rs

const MEM_SIZE: usize = 16 * 1024 * 1024;  // 16 MiB
const ROUNDS:   usize = 1024;              // random-access rounds
const WINDOW:   usize = 32;                // bytes per access

pub fn hash(data: &[u8]) -> Hash {
    // Step 1: Initial Blake3 hash
    let mut hasher = blake3::Hasher::new();
    hasher.update(data);
    let mut current_hash = *hasher.finalize().as_bytes();

    MEM_PAD.with(|pad| {
        let mut mem_pad = pad.borrow_mut();

        // Step A: Fill 16MB memory deterministically from seed hash
        let mut seed_hash = current_hash;
        let mut i = 0usize;
        while i < MEM_SIZE {
            let mut h = blake3::Hasher::new();
            h.update(&seed_hash);
            h.update(&(i as u64).to_le_bytes());
            let out = h.finalize();
                   mem_pad[i..i+WINDOW].copy_from_slice(&out.as_bytes()
[0..WINDOW]);
            seed_hash = *out.as_bytes();
            i += WINDOW;
        }

        // Step B: 1024 random-access mixing rounds (ASIC-killer)
        for _ in 0..ROUNDS {
            let idx1 = (u32::from_le_bytes(current_hash[0..4]
                         .try_into().unwrap()) as usize) % max_index;
            let idx2 = (u32::from_le_bytes(current_hash[4..8]
                         .try_into().unwrap()) as usize) % max_index;
            let mut h = blake3::Hasher::new();
            h.update(&current_hash);
```

```
            h.update(&mem_pad[idx1..idx1+WINDOW]);
            h.update(&mem_pad[idx2..idx2+WINDOW]);
            current_hash = *h.finalize().as_bytes();
        }
    });
    Hash::from_bytes(current_hash)
}
```

4.2 Device Balance

SahyadriX is calibrated so that the performance differential between dedicated mining ASICs and general-purpose hardware is compressed. The 16 MB memory requirement eliminates the ASIC advantage from cache size. CPU & GPU can all participate competitively in Sahyadri mining.

Device Class	Memory Available	SahyadriX	Relative Efficiency
ASIC (custom)	Limited on-chip	Constrained by 16 MB req.	~2-3x GPU
GPU (consumer)	4+ GB VRAM	Full parallel	Baseline
CPU (modern)	System RAM	Single-thread	~0.3-0.5x GPU

4.3 Ignition Switch — Connecting PoW to Block Template

```
// PowHash: binds PoW to specific block contents
pub fn finalize_with_nonce(&self, nonce: u64) -> Hash {
    let mut data = Vec::with_capacity(48);
      data.extend_from_slice(self.pre_pow_hash.as_bytes());   // block
contents
    data.extend_from_slice(&self.timestamp.to_le_bytes());
    data.extend_from_slice(&nonce.to_le_bytes());
       SahyadriX::hash(&data)    //  16MB  computation  over  this  exact
template
}
```

5. Network Architecture

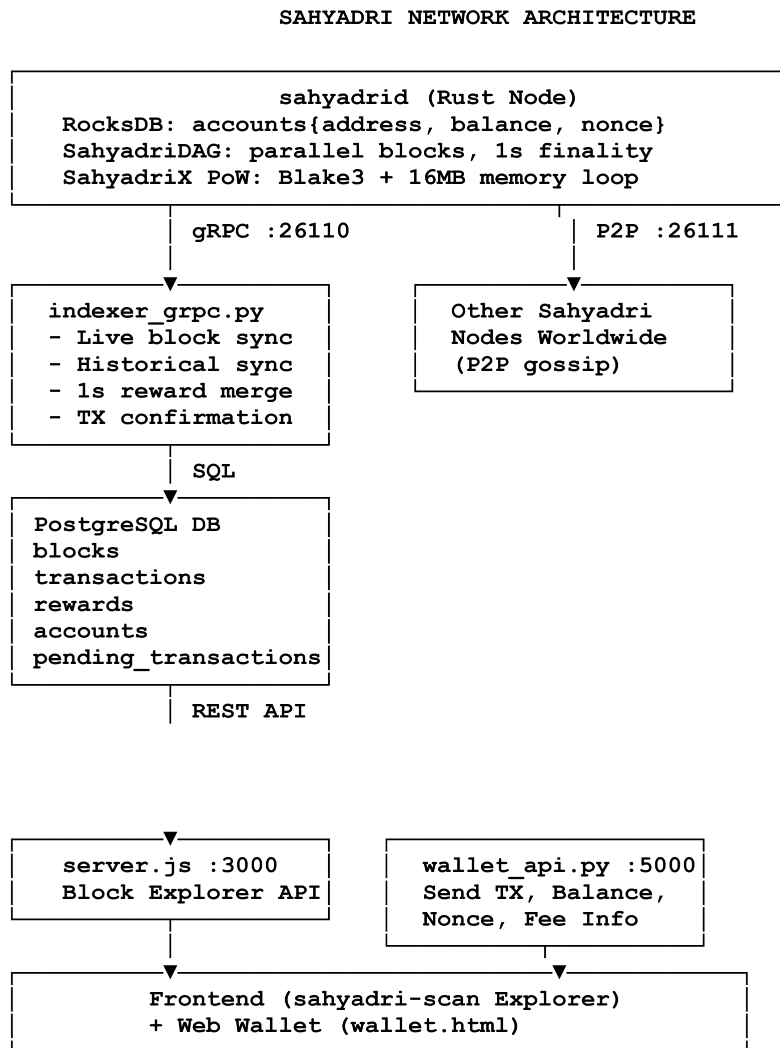
The Sahyadri network operates as a fully decentralized peer-to-peer system. Nodes communicate over three primary channels:

- gRPC (port 26110): Client-node communication; used by the indexer and wallet API
- P2P (port 26111): Block and transaction propagation between nodes worldwide
- wRPC / WebSocket (port 27110): Browser-based and light-client interfaces

5.1 Node Types

Node Type	Data Retained	Use Case	Storage
Full Archive Node	All blocks + full TX history + current state	Explorer, auditing	Growing (GB-TB)
Pruned Full Node	Current state + recent blocks (RocksDB)	Mining, validation	5-10 GB (stable)
Light Client	Block headers + Merkle proofs only	Wallet verification	Minimal

5.2 Data Flow Architecture

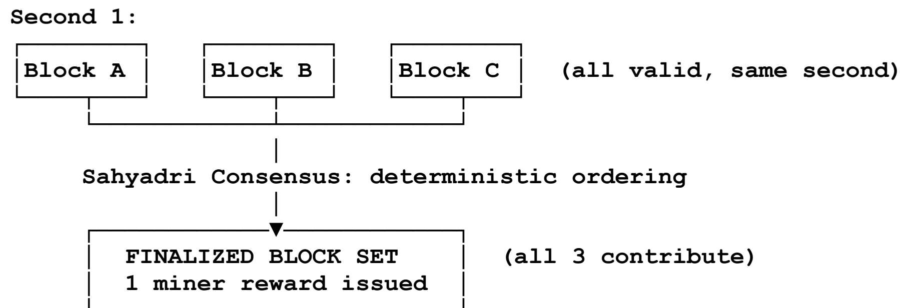


6. Consensus — SahyadriDAG

Sahyadri Consensus is a deterministic finality engine that operates on the SahyadriDAG data structure to produce a single, total, immutable ordering of all blocks and transactions. Unlike probabilistic longest-chain consensus, Sahyadri provides absolute finality: once a block is finalized, its position in the ordering is permanent.

6.1 Parallel Block Production

Because SahyadriDAG permits concurrent block proposals, multiple miners may simultaneously mine different blocks at the same height. All valid blocks contribute to the ledger — none are discarded as orphans. The Sahyadri Consensus algorithm traverses the DAG to compute a deterministic total ordering:



The indexer handles parallel blocks with a second-buffer mechanism: all blocks arriving within the same 1-second window are grouped, and a single combined reward is issued to the miner of the representative block (best_block_hash):

```
# indexer_grpc
second_buffer = defaultdict(list)  # timestamp_sec -> [blocks]

def process_block(block, conn):
    timestamp_sec = timestamp_ms // 1000
    second_buffer[timestamp_sec].append(block)
    if timestamp_sec > last_second:
        flush_second(last_second, second_buffer[last_second], conn)
```

6.2 Finality Guarantees

Sahyadri Consensus provides the following guarantees:

- Deterministic: Every honest node computing the same DAG arrives at identical ordering
- Irreversible: Finalized blocks cannot be reorganized or replaced
- No orphans: All valid blocks contribute to the DAG and ledger
- Sub-second propagation: New blocks reach the global network in under 1 second

6.3 10,000 TPS Configuration

Transaction throughput is governed by max_block_mass in the consensus parameters. Each account-model transaction has a mass of approximately 3,000 units:

```
// consensus/core/src/config/params.rs
// Set across all 4 network configs: mainnet, testnet, simnet, devnet
max_block_mass: 30_000_000,

// Capacity calculation:
// 30,000,000 mass / 3,000 mass-per-tx = 10,000 TX per block
// 10,000 TX per block * 1 block per second = 10,000 TPS
```

7. Block Rewards and Fee Economics

Sahyadri's incentive structure combines predictable block issuance with a flat transaction fee model. All monetary parameters are fixed at genesis and cannot be altered without a consensus-breaking hard fork.

7.1 Block Reward

Each finalized block carries an initial reward of 0.08318123 CSM (8,318,123 Kana). Rewards are split at the Rust node level before writing to RocksDB:

```
// consensus/src/processes/coinbase.rs
pub fn calc_block_subsidy(&self, daa_score: u64) -> u64 {
    let base_reward: u64 = 8_318_123;        // 0.08318123 CSM
    let halving_interval: u64 = 126_230_400; // 4 years in blocks
    let halvings = daa_score / halving_interval;
    if halvings >= 64 { return 0; }
    base_reward.checked_shr(halvings as u32).unwrap_or(0)
}

// Indexer split (indexer_grpc.py)
BLOCK_MINER_SPLIT   = 0.98  # 98% to miner
BLOCK_COMPANY_SPLIT = 0.02  # 2%  to development fund
```

7.2 Transaction Fee Split

```
# indexer_grpc.py — pending TX confirmation
TX_FEE_KANA          = 1000       # 0.00001 CSM (fixed forever)
TX_FEE_MINER_SPLIT   = 0.90       # 90% to block miner
TX_FEE_COMPANY_SPLIT = 0.10       # 10% to development fund

# Per confirmed TX:
miner.balance   += fee * 0.90
company.balance += fee * 0.10
```

7.3 Halving Schedule

Epoch (k)	Block Reward (CSM)	Years	Annual Emission (CSM)	Cumulative Supply (CSM)
0 (Genesis)	0.08318123	0-4	~437,459	~437,459
1	0.04159062	4-8	~218,729	~656,188
2	0.02079531	8-12	~109,365	~765,553
3	0.01039765	12-16	~54,682	~820,235
...	...	...	...	...
63	~0	>252	~0	~21,000,000

The total supply converges to 21,000,000 CSM asymptotically. The geometric series property ensures:

```
S(inf) = 2 x H x s_0  =  2 x 126,230,400 x 0.08318123  = 21,000,000 CSM
```

8. Blockchain Indexer and Explorer

Because Sahyadri nodes prune old block data (retaining only current account state in RocksDB), a separate indexer service captures and permanently archives the full transaction history in PostgreSQL. This two-tier architecture keeps nodes lightweight (5-10 GB) while the explorer maintains complete historical records.

8.1 Indexer Architecture (indexer_grpc.py)

The indexer connects to the node via gRPC on port 26110 and processes blocks in real time. It performs historical sync on startup (resuming from last processed blue_score) and then switches to live block listening:

```
INDEXER STARTUP SEQUENCE:

1. Connect to sahyadrid via gRPC :26110
2. Query PostgreSQL: SELECT MAX(blue_score) FROM blocks
3. Historical sync:
   loop:
     GET /getBlocksRequest(lowHash, includeTransactions=true)
     FOR each block WHERE blue_score > last_processed:
       process_block(block)
     IF no new blocks: BREAK
     low_hash = last_block.hash
4. Subscribe to live blocks:
   notifyBlockAddedRequest -> stream
   FOR each blockAddedNotification:
     GET full block with transactions
     process_block(block)
```

8.2 Database Schema

Table	Key Columns	Purpose
blocks	block_hash, blue_score,	Block registry
transactions	tx_id, from_address, to_address, amount, fee, status, block_hash	All user TXs
pending_transactions	tx_id, from_address, to_address, amount, fee, nonce, status	Mempool queue
rewards	tx_id, block_hash, miner,	Mining rewards
accounts	address, balance, nonce	Current state cache

8.3 Explorer API (server.js — port 3000)

Endpoint	Description
GET /api/blocks	Recent blocks with pagination
GET /api/transactions	All transactions with pagination
GET /api/block/:hash	Block detail + reward
GET /api/block-by-score/:score	Block by blue_score
GET /api/address/:address	Balance + TX history
GET /api/top-addresses	Rich list (top 100)
GET /api/transaction/:hash	TX detail
GET /api/stats	Network stats (supply, wallets, TXs)

8.4 Wallet API (wallet_api.py — port 5000)

Endpoint	Description
GET /api/balance/:address	Current balance from accounts table
GET /api/history/:address	User TX history (excludes mining
POST /api/send-csm	Submit pending transaction to mempool
GET /api/nonce/:address	Current nonce for sender address
GET /api/fee	Current network fee (TX_FEE_CSM, TX_FEE_KANA)

9. Disk Space and Pruning

Efficient disk management is essential to keeping Sahyadri nodes lightweight. The protocol architecture cleanly separates what the node keeps (current state) from what the explorer keeps (full history).

9.1 Node Storage Model

The Sahyadri node (sahyadrid) uses RocksDB exclusively for current state:

```
Node (RocksDB) KEEPS:                  Node PRUNES (auto):
+----------------------------+          +----------------------------+
| Current account balances  |          | Raw block bodies (>24h)    |
| Current nonces             |          | Old TX history             |
| Recent blocks (consensus) |          | Old UTXO sets (not used)   |
| DAG tips (for new blocks) |          | Pruned state snapshots     |
+----------------------------+          +----------------------------+
Size: ~5-10 GB (stable)                 Size: grows then discarded
```

Because Sahyadri uses an Account model (not UTXO), account pruning does not apply. Zero-balance accounts may theoretically be removed, but this is not implemented — the cost is trivial (100 bytes per account).

9.2 Explorer Storage Model

The PostgreSQL explorer database grows continuously and is never pruned — it is the permanent historical archive:

```
Storage estimate per million transactions:
1 TX record    = ~500 bytes
1M TX          = ~500 MB
100M TX        = ~50 GB
1B TX (future) = ~500 GB

At current load (10-100 TX/sec):
1 day = 864,000 to 8,640,000 TX = 432 MB to 4.3 GB/day
```

9.3 Recommended Deployment

Component	Minimum Spec	Oracle Cloud Free Tier
Node (sahyadrid)	4 CPU, 8 GB RAM, 20 GB SSD	2 CPU, 24 GB RAM, 200 GB — Free Forever
Indexer (Python)	1 CPU, 2 GB RAM	Same instance as node
PostgreSQL	2 CPU, 4 GB RAM, 100 GB+	Same instance
server.js	1 CPU, 1 GB RAM	Same instance
wallet_api.py	1 CPU, 1 GB RAM	Same instance

10. Instant Payment Verification (IPV)

Instant Payment Verification allows lightweight clients to confirm transaction finality without downloading the full blockchain. Because Sahyadri uses deterministic consensus rather than probabilistic longest-chain, finality is absolute: a transaction confirmed in a block cannot be reversed.

A light client verifying a payment needs only:

- The finalized block header (contains state root and block hash)
- A compact Merkle inclusion proof linking the TX to the block
- Proof that the block has been finalized by Sahyadri Consensus

Sahyadri IPV is immediate: 1 block confirmation = absolute finality. This enables:

- Point-of-sale terminals: Accept payments as fast as block confirmation (1 second)
- Merchants: Same confidence as cash settlement
- Embedded payment processors: Minimal bandwidth and compute requirements

11. Privacy

Privacy in Sahyadri is achieved by separating value transfer from identity. All transactions are publicly verifiable on the explorer, but the protocol does not associate transfers with real-world identities. Ownership is defined exclusively by cryptographic control of private keys.

Each user generates a CSM32 address derived from their public key (secp256k1 ECDSA). Users are encouraged to generate fresh key pairs for each receiving address. The Account model naturally provides less transaction graph ambiguity than UTXO systems, but the protocol does not require identity disclosure.

Sahyadri does not include built-in mixing, confidential transactions, or zero-knowledge proofs at the base layer. Users requiring stronger privacy may layer additional tools on top of the base protocol. The L1 provides robust baseline privacy: no account registration, no identity verification, no link between real-world identity and on-chain activity.

12. Calculations and Economic Model

12.1 Emission Formula

```
s_k = s_0 x 2^(-k)

I_k = H x s_k   (total emission in epoch k)

S(n) = 2 x H x s_0 x (1 - 2^(-n))

S(inf) = 2 x H x s_0 = 21,000,000 CSM
```

Where s_0 = 0.08318123 CSM, H = 126,230,400 blocks (4 years at 1 BPS), k = halving epoch.

12.2 Block Revenue

```
R_k = s_k + f_avg   (total per-block revenue)

phi_k = f_avg / (s_k + f_avg)   (fee fraction of total revenue)

AnnualRevenue_k = r_yr x (s_k + f_avg)
```

Where r_yr = 31,536,000 blocks/year (at 1 BPS), f_avg = average fee revenue per block.

12.3 Fee Revenue at Scale

Daily TX Volume	Daily Fee Revenue (CSM)	Miner Share (90%)	Dev Fund (10%)
10,000 TX/day	0.1 CSM	0.09 CSM	0.01 CSM
1,000,000 TX/day	10 CSM	9 CSM	1 CSM
100,000,000 TX/day	1,000 CSM	900 CSM	100 CSM
864,000,000 TX/day (10k TPS)	8,640 CSM	7,776 CSM	864 CSM

12.4 Network Security Budget

As block subsidies halve over time, transaction fees progressively constitute a larger fraction of miner revenue ($phi_k \rightarrow 1$). At 10,000 TPS capacity, Sahyadri is positioned to generate substantial fee revenue even at low individual fees:

```
Full-capacity fee revenue = 10,000 TX/sec x 0.00001 CSM x 86,400 sec = 8,640 CSM/day
```

13. Security Model

The Sahyadri network derives security from four pillars: verifiable computation, deterministic consensus, economic incentives, and cryptographic primitives.

13.1 Formal Security Conditions

```
Transaction validity: Verify(tx) = TRUE iff

  sig_valid(tx.sig, tx.sender_pubkey) AND

  accounts[tx.sender].balance >= (tx.amount + tx.fee) AND

  accounts[tx.sender].nonce == tx.nonce



Block acceptance: Accept(B) iff

  VerifySahyadriX(B.header) AND

  ALL tx in B: Verify(tx) = TRUE AND

  ConsensusFinalized(B) = TRUE

Attack cost: P_attack = alpha (attacker hash fraction)

For alpha < 0.5: attack is economically irrational

After finalization: reorganization probability = 0 (deterministic)
```

13.2 Double-Spend Impossibility

Double-spend attacks are prevented at two layers:

- Mempool layer (wallet_api.py): Pending locked balance check prevents overdraft before block confirmation
- Confirmation layer (indexer_grpc.py): Sender balance checked again at confirmation; insufficient balance = TX rejected
- Nonce layer: Each TX carries a unique nonce; same nonce cannot be reused after increment
- Consensus layer: Finalized blocks are irreversible; deterministic ordering eliminates reorg risk

14. Governance

Sahyadri adopts a minimal governance model. All core monetary parameters are permanently fixed at genesis and are not subject to modification through proposals or voting:

- Total supply: 21,000,000 CSM — immutable
- Initial block reward: 0.08318123 CSM — immutable
- Halving interval: 126,230,400 blocks (~4 years) — immutable
- Minimum TX fee: 0.00001 CSM — immutable
- Block reward split: 98/2 — immutable
- TX fee split: 90/10 — immutable

Changing any monetary parameter requires a consensus-breaking hard fork, effectively creating a new chain. The development fund (2% block reward + 10% TX fee) is allocated to ongoing protocol development, infrastructure, and ecosystem support. No central governance body, no token-weighted voting, and no on-chain governance process exists. The protocol evolves through rough consensus and running code.

15. Conclusion

Sahyadri presents a comprehensive design for a sovereign peer-to-peer digital money system. By combining a directed acyclic graph data structure for parallel block production, a memory-hard Proof-of-Work algorithm for inclusive security, an Account + Object state model for simplified value transfer, and a fixed flat fee structure for predictable costs, the protocol achieves deterministic finality in seconds with high-throughput transaction processing. The strictly capped monetary supply, transparent halving schedule, and protocol-enforced immutable parameters provide a stable, auditable, and predictable foundation for global value transfer — digital money that works reliably, fairly, and forever.