Unified Tri-VM Sovereign Ledger Architecture: EVM, Substrate FRAME, and Cosmos SDK Integration for the AROS Infrastructure and Agronomic Protocol

Heterogeneous Virtual Machine Convergence and Consensus Orchestration

Deploying a sovereign Layer-1 ledger capable of concurrently executing Ethereum Virtual Machine (EVM) bytecode, Substrate FRAME pallets, and Cosmos SDK / CosmWasm smart contracts requires resolving fundamental divergences in state representation, consensus finality, and runtime execution. Traditional distributed systems handle heterogeneous execution through external bridging or relay architectures, which introduce cross-chain latency, fragmented liquidity, and severe cryptographic attack surfaces. The AROS (Albert & Robot's Operating System) architecture eliminates these vulnerabilities by synthesizing all three execution models into a single, unified state machine that executes heterogeneously while preserving strict transactional atomicity.

The core execution environment can be structured around a Substrate-based WebAssembly meta-runtime hosting embedded virtual engines, or an extended Cosmos SDK core anchored by CometBFT. When deployed on a Substrate foundation, the node runtime incorporates Frontier and pallet-revm to expose full EVM compatibility alongside standard Ethereum JSON-RPC namespaces, while simultaneously embedding pallet-cosmwasm to execute deterministic WebAssembly smart contracts compiled from Rust. Conversely, when built upon a Cosmos SDK foundation, the architecture couples the native Cosmos EVM module (x/evm) with the CosmWasm engine (x/wasm), linking into the Substrate ecosystem through an embedded light-client pallet that natively verifies BEEFY (Bridge Efficiency Enabling Finality Yielder) and GRANDPA cryptographic commitments.

State management across these disparate engines converges within a hierarchical multi-store key-value database backed by RocksDB or ParityDB. The global ledger state root encapsulates three distinct cryptographic state representations:

- The EVM World State Trie, structured as a Modified Merkle Patricia Trie mapping twenty-byte Ethereum addresses to account nonces, token balances, storage roots, and Keccak-256 code hashes.
- The Cosmos IAVL+ and Sparse Merkle Tree (SMT) store, preserving deterministic, versioned state keys for system modules including bank, authentication, staking, and CosmWasm contract storage.
- The Substrate Blake2b-256 Storage Trie, structured as a base-16 Merkle radix tree holding FRAME pallet parameters, extrinsics, and system events.

To maintain transactional atomicity during inter-VM execution, state mutations are staged within

an in-memory transactional cache. If an EVM contract invokes a CosmWasm actor through an inter-runtime precompile, any execution failure, unhandled assertion, or out-of-gas condition encountered by the WebAssembly runtime rolls back the entire operational sequence across all state tries simultaneously, preventing partial execution states.

Deterministic single-slot finality is mandatory for autonomous agricultural procurement and edge sensor verification. AROS evaluates two primary consensus frameworks: CometBFT operating through the Application Blockchain Interface (ABCI++) and Substrate's hybrid consensus architecture combining BABE with GRANDPA. CometBFT provides deterministic, single-block finality within one to two seconds through classical Byzantine Fault Tolerant rounds of pre-votes and pre-commits, eliminating chain reorganizations. Its ABCI++ interface introduces lifecycle hooks—namely PrepareProposal, ProcessProposal, and ExtendVote—that allow the sovereign AROS node to validate, filter, and deterministically interleave multi-VM transactions directly within the consensus phase. Substrate's hybrid engine decouples block production from deterministic finality: Blind Assignment for Blockchain Extension (BABE) executes probabilistic slot allocations every six seconds, while GRANDPA finalizes block batches asynchronously. For low-latency edge deployment, CometBFT with ABCI++ serves as the primary consensus engine, ensuring that physical material escrows and sensory proofs commit without probabilistic settlement delays.

Consensus Metric / Architectural Dimension	CometBFT (ABCI++) Consensus Engine	Substrate Hybrid Consensus (BABE + GRANDPA)	AROS Sovereign Tri-VM Production Configuration
Finality Mechanism	Deterministic, single-block instant finality	Asynchronous finality gadget over block chains	CometBFT instant finality with ABCI++ transaction batching
Latency to Finality	1.0 - 2.5\text{ seconds} per block	6.0 - 12.0\text{ seconds} (1\text{ to }2\text{ rounds})	Sub-two-second deterministic settlement
Byzantine Fault Tolerance	Strict \frac{1}{[span_32](start_span)[span_32](end_span)3} validator voting power threshold	\frac{1}{3} for GRANDPA safety, \frac{1}{2} for BABE liveness	Strict \frac{1}{3} Byzantine safety bound
Mempool Execution Hooks	PrepareProposal, ProcessP[span_34](start_span)[span_34](end_span)roposal	Runtime inherents & Executive::app[span_35](start_span)[span_35](end_span)ly_extrinsic	ABCI++ atomic multi-VM proposal scheduling
Dynamic Validator Adjustments	Native staking module (x/staking[span_36](start_span)[span_36](end_span)) validator updates	Session keys and staking pallet (pallet-session)	On-chain governance-managed validator authority sets
Light Client Verification	Direct header and commit signature verification	BEEFY compact Merkle-tree signature commitments	Hybrid IBC client supporting BEEFY, GRANDPA, and CometBFT

Cross-VM Cryptographic Identity and Multi-Standard

Account Abstraction

A critical requirement of a tri-VM architecture is unifying divergent cryptographic key signatures, address schemes, and account state models into a cohesive identity standard. Users, edge compute nodes, autonomous field robots, and agricultural cooperatives must interact with EVM smart contracts, CosmWasm routines, and native Substrate pallets without maintaining disjoint keychains or fragmented token balances.

Address generation across the three environments relies on distinct mathematical operations applied to public keys generated from a shared root secret:

For EVM execution, the ledger derives addresses using an ECDSA secp256k1 public key $K_{\text{pub}} \in \mathbb{R}^2$. The uncompressed 64-byte public key (excluding the prefix byte) is hashed via Keccak-256, and the least significant twenty bytes are retained to yield the standard H160 hexadecimal address format:

$$A_{\text{EVM}} = \text{Tr}^{20}\left(\text{Keccak-256}\left(K_{\text{pub}}^{\text{uncompressed}}\right)\right)$$

For the Cosmos SDK, the 33-byte compressed secp256k1 public key $K_{\text{pub}}^{\text{comp}}$ is hashed sequentially using SHA-256 followed by RIPEMD-160. The resulting twenty-byte digest is encoded into the Bech32 format using the system prefix aros:

$$H_{\text{Cosmos}} = \text{RIPEMD-160}\left(\text{SHA-256}\left(K_{\text{pub}}^{\text{comp}}\right)\right)$$[span_40](start_span)[span_40](end_span) $$A_{\text{Cosmos}} = \text{Bech32}\left(\text{"aros"}, H_{\text{Cosmos}}\right)$$

For Substrate FRAME pallets, the cryptographic primitive typically employs Schnorrkel/Ristretto255 (sr25519) or Edwards25519 (ed25519) keypairs. The raw 32-byte public key K_{pub}^{32} is prepended with a network-specific address type prefix, hashed with Blake2b-512 to generate a two-byte checksum, and encoded using Base58Check to create the SS58 address format:

$$\text{Checksum} = \text{Tr}^2\left(\text{Blake2b-512}\left(\text{"SS58PRE"} \,\Vert\, \text{Prefix} \,\Vert\, K_{\text{pub}}^{32}\right)\right)$$ $$A_{\text{Substrate}} = \text{Base58}\left(\text{Prefix} \,\Vert\, K_{\text{pub}}^{32} \,\Vert\, \text{Checksum}\right)$$

AROS unifies these disparate address spaces into a master identity layer conforming to W3C Decentralized Identifiers (DIDs) and Chain Agnostic Improvement Proposal (CAIP-10) specifications. Under CAIP-10, an account identifier is defined as the concatenation of a namespace, a reference network identifier, and the domain-specific account address. The sovereign ledger implements an on-chain identity translation pallet that maps these representations to a canonical DID:

`did:aros:caip10:eip155:1890:0x71C8F67B98c253457a4B7F49eBb16f6b0fE353a9`

When an EVM smart contract interacts with a CosmWasm actor or dispatches a Substrate runtime extrinsic, the system performs deterministic address padding and aliasing. The twenty-byte EVM address is mapped into a twenty-byte Bech32 address directly through byte equivalence or padded to thirty-two bytes with trailing zeroes when interfacing with FRAME's 32-byte Ac[span_94](start_span)[span_94](end_span)countId32 structure. This eliminates identity spoofing and preserves authorization context across virtual machine boundaries without requiring secondary key signatures.

Identity Standard	Cryptographic Curve & Algorithm	Address Length	Encoding Format	Architectural Utility in AROS
EVM Address	ECDSA over secp256k1	20 Bytes (160\text{ bits})	Hexadecimal (0x...)	Solidity procurement contracts, ERC-20/721 assets, web3 wallets
Cosmos Bech32	ECDSA secp256k1 / Ed25519	20 to 32 Bytes	Bech32 (aros1...)	CosmWasm actors, IBC cross-chain routing, Cosmos telemetry modules
Substrate SS58	Schnorrkel over Ristretto255 (sr25519)	32 Bytes (256\text{ bits})	Base58Check (5...)	Native FRAME runtime pallets, staking, on-chain governance
W3C Master DID	Multi-key cryptographic mapping	Variable string	URI specification	Sovereign legal identity, off-chain attestations, search indexing
CAIP-10 Identifier	Namespace + Chain ID + Account	Variable string	Cross-namespace URI	Universal multi-VM transaction routing and gas sponsorship

Synchronous Precompiles and Inter-Runtime Communication Architecture

Bridging execution between EVM contracts, CosmWasm actors, and Substrate FRAME pallets requires combining synchronous in-memory precompiled contracts with asynchronous cross-chain communication protocols. Rather than executing cross-VM operations through high-latency network relayers, internal inter-runtime operations execute synchronously via custom native precompiles embedded within the node runtime.

The ledger designates a reserved address space for these inter-VM execution interfaces, spanning 0\text{x}0000000000000000000000000000000000000800 through 0\text{x}0000000000000000000[span_117](start_span)[span_117](end_span)[span_118](start_span)[span_118](end_span)00000000000000008FF:

- Precompile 0\text{x}...0800 (CosmWasm Dispatcher): Receives the destination CosmWasm contract address, a serialized JSON execution message, and a gas allocation. It enters the WebAssembly execution context synchronously, executes the contract method, and returns the response bytes directly to the EVM execution stack.
- Precompile 0\text{x}...0801 (Substrate XCM Invoker): Accepts an encoded Cross-Consensus Messaging (XCM v4/v5) instruction payload, verifies caller authorization, and dispatches the instruction directly to the local Substrate executive engine.
- Precompile 0\text{x}...0802 (IBC Outbound Channel): Ingests ICS-20 (token transfer) or

ICS-27 (interchain account) call parameters, packaging them into an Inter-Blockchain Communication packet that is immediately written to the outbound IBC state queue.

For communication with external sovereign networks, the ledger integrates native implementations of Cosmos IBC and Polkadot XCM. The Cosmos IBC stack functions via the x/ibc module or Substrate's pallet-ibc, utilizing ICS-08 WebAssembly light clients to verify block headers and Merkle state commitments from counterparty chains. Concurrently, the Substrate XCM engine evaluates incoming instructions—such as TransferAsset, Transact, and DepositAsset—enabling interoperability with external parachains and consensus hubs.

Communication Channel	Execution Latency	Trust and Security Model	State Mutation Mechanism	Payload Capacity and Overhead
EVM Precompile to CosmWasm	Synchronous (0\text{ blocks})	Shared process memory, atomic rollback on trap	Direct pointer invocation via Revm / Wasmtime hostcall	Memory-bounded (< 2\text{ MB} raw JSON buffer)
CosmWasm to Substrate Pallet	Synchronous (0\text{ blocks})	Direct runtime dispatch with origin privilege	Execution through pallet_cosmwasm::dis[span_139](start_span)[span_139](end_span)patch	Structured SCALE-encoded byte slices
IBC (ICS-20 / ICS-27)	Asynchronous (1\text{ to }2\text{ blocks})	Light client Merkle multi-store proof verification	Packet commitment, relayer transport, ack receipt	Bounded by maximum block gas limits
Substrate XCM v4/v5 Engine	Synchronous or Asynchronous	Native consensus origin checks (Origin::Signed)	Virtual instruction execution within XCVM engine	Bounded by Substrate Weight V2 execution units

Through this unified bus, a Solidity-based procurement escrow contract can query multi-spectral crop indices evaluated by a CosmWasm actor and settle physical delivery escrows through Substrate's multi-currency asset balances pallet within a single atomic transaction.

Cross-VM Zero-Knowledge Verification Circuits and Mathematical Mechanics

Preserving operational sovereignty across agricultural supply chains requires verifying land parcel boundaries, historical harvest volumes, and regulatory compliance without disclosing sensitive commercial coordinates or harvest data on public ledgers. AROS enforces this privacy through zero-knowledge SNARK (zk-SNARK) verification circuits, natively integrated across the EVM, CosmWasm, and Substrate runtime layers.

The agronomic verification protocol proves the following relational statement: A producer controls an agricultural parcel P located within County FIPS C that achieved historical crop yield Y \ge Y_{\text{threshold}}, while keeping the exact spatial coordinates of P, internal field geometry, and total yield Y entirely concealed within the private witness.

The system executes the Groth16 zero-knowledge proof system over the BN254 (alt_bn128) pairing-friendly elliptic curve, defined over the prime fields \mathbb{F}_p and \mathbb{F}_{[span_174](start_span)[span_174](end_span)[span_175](start_span)[span_175](end_span)[span_176](start_span)[span_176](end_span)p^2} with order r:

\mathbb{G}_1 \subset E(\mathbb{F}_p) : y^2 = x^3 + 3 \mathbb{G}_2 \subset E'(\mathbb{F}_{p^2}) : y^2 = x^3 + \frac{3}{9 + u}
The bilinear pairing e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T maps group elements to the target cyclotomic subgroup \mathbb{G}_T \subset \mathbb{F}_{p^{12}}. Given a verification key consisting of curve points [\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_2, and public input IC elements [\Psi_i]_1, a proof \pi = \left([A]_1, [B]_2, [C]_1\right) is cryptographically valid for a vector of public inputs \vec{x} = (x_1, \dots, x_l) \in \mathbb{F}_r^l with x_0 = 1 if and only if the pairing check evaluates to true:
e\left([A]_1, [B]_2\right) = e\left([\alpha]_1, [\beta]_2\right) \cdot e\left(\sum_{i=0}^l x_i \cdot [\Psi_i]_1, [\gamma]_2\right) \cdot e\left([C]_1, [\delta]_2\right)
In computational verification, this equation is reordered into a multi-pairing operation against the identity element of the target field to minimize field inversion costs:
e\left(-[A]_1, [B]_2\right)
\cdot[span_164](start_span)[span_164](end_span)[span_169](start_span)[span_169](end_span) e\left([\alpha]_1, [\beta]_2\right) \cdot e\left([K]_1, [\gamma]_2\right) \cdot e\left([C]_1, [\delta]_2\right) = 1_{\mathbb{G}_T}
where [K]_1 = \sum_{i=0}^l x_i \cdot [\Psi_i]_1 represents the linear combination of public inputs evaluated via multi-scalar multiplication (MSM).
Each virtual machine environment provides optimized host bindings to evaluate this pairing equation without the computational penalty of software emulation:

- In the EVM context, contracts invoke the standard precompiles for elliptic curve point addition (ecAdd at 0\text{x}06), scalar multiplication (ecMul at 0\text{x}07), and bilinear pairing verification (ecPairing at 0\text{x}08, specified in EIP-197), executing in approximately 113{,}000 gas for three pairing pairs.
- In the CosmWasm context, smart contracts execute the cryptographic check via native Rust bindings in cosmwasm_vm::crypto::bn254_pairing_check or compiled Arkworks libraries, transferring evaluation to the underlying host node rather than executing pairing loops inside WebAssembly.
- In the Substrate FRAME runtime, the verification logic routes directly to sp_io::crypto::alt_bn128_pairing_check, consuming Substrate Weight V2 execution units calibrated against bare-metal hardware benchmarks.

Runtime Verification Layer	Computational Cost Profile	Execution Duration	Underlying Cryptographic Library	Supported Pairing Curves
EVM ecPairing (0\text{x}08)	113{,}000\text{ gas} (3\text{ pairings})	\approx 1.2\text{ milliseconds}	Go-Ethereum / Revm C++ backend	BN254 (alt_bn128)
CosmWasm Host API	120{,}000\text{ compute units}	\approx 1.4\text{ milliseconds}	Rust ark-circom[span_191](start_span)[span_191](end_span) via Wasm hostcalls	BN254, BLS12-381
Substrate sp_io Primitive	45{,}000{,}000\text{ Weight Units}	\approx 0.9\text{ milliseconds}	Substrate primitives[span_196](start_span)[span_196](end_span)[span_19	BN254, BLS12-381

Runtime Verification Layer	Computational Cost Profile	Execution Duration	Underlying Cryptographic Library	Supported Pairing Curves
			7](start_span)[span_197](end_span) p_crypto (no_std)	
Software Wasm Fallback	> 2{,}500{,}000\text{ gas equivalent}	\approx 48.0\text{ milliseconds}	Pure WebAssembly interpreted runtime	Curve-agnostic

Universal Gas Metering, Protocol Fee Abstraction, and Economic Routing

Operational friction in multi-VM architectures frequently stems from gas token fragmentation. Requiring agricultural producers, machine operators, or automated supply agents to maintain separate reserves of ETH for EVM execution, ATOM for Cosmos SDK routines, and DOT for Substrate extrinsics creates an unworkable barrier to practical deployment. AROS eliminates this friction through a unified fee abstraction subsystem that allows transactions across any VM to be settled using agricultural credit tokens, stablecoins, or off-chain micropayments.

The ledger implements four interlocking fee abstraction layers:

Protocol-Level Node Abstraction (Celo CIP-64 Framework): At the transaction mempool and consensus level, the node execution engine accepts an optional feeCurrency field within the transaction envelope. During initial transaction validation, the node invokes debitGasFees on the designated ERC-20 or IBC asset contract to reserve the maximum gas cost, executes the state transition, and subsequently invokes creditGasFees to refund unused gas to the origin account while distributing the net fee to block validators. This occurs without deploying intermediate smart contract proxies or third-party relayers.

Substrate ChargeAssetTxPayment Pipeline: For transactions entering the FRAME runtime, the extrinsic pipeline utilizes a custom TransactionExtension. The ChargeAssetTxPayment extension intercepts extrinsics, accepts non-native asset identifiers registered in pallet-assets, queries pallet-asset-conversion to calculate the spot exchange rate against the native fee token, and provides post-dispatch weight refunds based on actual compute consumed.

Account Abstraction and Runtime Delegation (ERC-4337 and EIP-7702): Within the EVM layer, users submit UserOperations to an on-chain EntryPoint contract, which collaborates with a Paymaster contract to sponsor native gas fees. Gas reimbursements are collected via ERC-20 permit authorizations (EIP-2612) or signed balance transfers (EIP-3009). Under EIP-7702, externally owned accounts (EOAs) can temporarily delegate their execution context to paymaster-compatible contract code during transaction processing, bringing gas abstraction to standard accounts without requiring permanent account migration.

Machine-to-Machine Autonomous Micropayments (L402 Protocol): For high-frequency API endpoints and automated agent procurement pipelines, the system implements the L402 protocol (formerly LSAT). When an autonomous agent requests computational telemetry or triggers procurement actions, the edge proxy responds with an HTTP 402 Payment Required header containing a cryptographic Macaroon and a Lightning Network invoice. Upon settlement, the agent presents the payment preimage within the HTTP authorization header, allowing instantaneous, off-chain settlement without ledger bloat.

Fee Abstraction Paradigm	Supported Asset Types	Core Node & Infrastructure Dependency	Operational Execution Overhead	Runtime Compatibility
Node Protocol (CIP-64)	Approved ERC-20 and IBC tokens	Core transaction mempool & AnteHandler	Zero contract proxy overhead (0\text{ extra gas})	EVM and Cosmos SDK modules
Substrate ChargeAsset[span_213](start_span)[span_213](end_span)	Native pallet assets and foreign tokens	pallet-transaction-payment, pall[span_239](start_span)[span_239](end_span)[span_240](start_span)[span_240](end_span)et-asset-conversion	Extrinsic parameter decoding and weight charging	Substrate FRAME runtimes
ERC-4337 Paymaster	Arbitrary ERC-20 tokens with liquidity	Off-chain Bundlers, EntryPoint contract	35{,}000 - 65{,}000\text{ gas} validation penalty	EVM execution layer
EIP-7702 Runtime Delegation	ERC-20 and sponsored allowances	Consensus-layer Pectra delegation support	16\text{ gas} per calldata byte delegation cost	EVM Externally Owned Accounts
L402 Micro-Billing	Bitcoin Satoshis, wrapped Lightning assets	Edge Lightning Node (LND / Core Lightning)	Zero on-chain ledger footprint (100\%\text{ off-chain})	HTTP REST / RPC gateways

Remote Sensing Ingestion: NASA Earthdata HLS Pipeline and Deterministic Agronomic Modeling

The automated cataloging and procurement engine within AROS derives physical input requirements by ingesting high-resolution Earth observation data directly from NASA Earthdata. By analyzing multi-spectral reflectance data at county-level granularity, the platform detects crop emergence anomalies, soil moisture depletion, and vegetative stress, translating these physical phenomena into deterministic procurement recommendations.

The ingestion architecture interfaces with NASA's Harmonized Landsat Sentinel-2 (HLS) project via the Common Metadata Repository SpatioTemporal Asset Catalog (CMR-STAC) API. The HLS pipeline harmonizes radiometry across the Operational Land Imager (OLI) aboard Landsat 8 and Landsat 9 with the Multi-Spectral Instrument (MSI) aboard Sentinel-2A, Sentinel-2B, and Sentinel-2C. This virtual constellation delivers a combined global observation revisit frequency of 1.4 days at 30-meter spatial resolution, providing the temporal density required to guide precision agriculture.

The data processing pipeline ingests three primary HLS science products:

- **HLSL30**: Landsat-derived Nadir BRDF-Adjusted Surface Reflectance (NBAR).
- **HLSS30**: Sentinel-derived surface reflectance mapped to the identical 30-meter UTM Military Grid Reference System (MGRS) grid.
- **HLS-VI**: Daily 30-meter pre-computed Vegetation Indices suite.

Because these data products are stored as Cloud-Optimized GeoTIFFs (COGs), the self-hosted

edge processing cluster—powered by local enterprise accelerators such as an 8x NVIDIA Tesla V100 32GB cluster—executes spatial queries bounded by County Federal Information Processing Standard (FIPS) polygons using HTTP range requests. This extracts only the necessary pixel extents without downloading complete multi-gigabyte granules.
Once retrieved, surface reflectance values across narrow-band channels are evaluated across four primary agronomic indices:

1. Normalized Difference Vegetation Index (NDVI), measuring structural canopy density and photosynthetic greenness: \text{NDVI} = \frac{\rho_{\text{NIR}} - \rho_{\text{Red}}}{\rho_{\text{NIR}} + \rho_{\text{Red}}} = \frac{\text{B08} - \text{B04}}{\text{B08} + \text{B04}}
2. Normalized Difference Red Edge (NDRE), measuring chlorophyll concentration in mature, dense crop canopies where traditional NDVI saturates: \text{NDRE} = \frac{\rho_{\text{NIR}} - \rho_{\text{RedEdge}}}{\rho_{\text{NIR}} + \rho_{\text{RedEdge}}} = \fr[span_245](start_span)[span_245](end_span)[span_251](start_span)[span_251](end_span)ac{\text{B08} - \text{B05}}{\text{B08} + \text{B05}}
3. Enhanced Vegetation Index (EVI), which incorporates blue-band reflectance to minimize atmospheric aerosol distortion and background soil influences: \text{EVI} = 2.5 \cdot \frac{\rho_{\text{NIR}} - \rho_{\text{Red}}}{\rho_{\text{NIR}} + 6.0 \cdot \rho_{\text{R[span_246](start_span)[span_246](end_span)[span_252](start_span)[span_252](end_span)ed}} - 7.5 \cdot \rho_{\text{Blue}} + 1.0} = 2.5 \cdot \frac{\text{B08} - \text{B04}}{\text{B08} + 6.0 \cdot \text{B04} - 7.5 \cdot \text{B02} + 1.0}
4. Normalized Difference Water Index (NDWI), isolating plant liquid water content and root-zone waterlogging: \text{NDWI} = \frac{\rho_{\text{NIR}} - \rho_{\text{SWIR}}}{\rho_{\text{NIR}} + \rho_{\text{SWIR}}} = \frac{\te[span_247](start_span)[span_247](end_span)[span_253](start_span)[span_253](end_span)xt{B08} - \text{B11}}{\text{B08} + \text{B11}}

The system applies a deterministic agronomic decision engine to these calculated rasters: \Delta_{\text{NDRE}} = \overline{\text{NDRE}}_{\text{historical}} - \overline{\text{NDRE}}_{\text{current}}
If the mean NDRE across an agricultural zone demonstrates a historical negative deviation exceeding twelve percent (\Delta_{\text{NDRE}} > 0.12) while the moisture index confirms adequate hydration (\text{NDWI} \ge -0.10), the engine diagnoses acute nitrogen chlorosis rather than drought stress.
The decision engine immediately queries the decentralized material catalog registered on aros.albertlane.net/p[span_248](start_span)[span_248](end_span)[span_254](start_span)[span_254](end_span)rocurement to identify regional distribution depots within a 150-mile logistics corridor. It automatically stages a multi-signature purchase order for chelated foliar amendments and soil balancing agents, routing the procurement intent to the multi-VM escrow contract for cryptographic confirmation and settlement.

Search Sequestration, Metadata Anchoring, and Autonomous Machine Monetization

Publicly publishing architectural specifications, research data, and operational prompts exposes intellectual property to uncredited web scraping and corporate artificial intelligence ingestion. AROS addresses this vulnerability by deploying an active defense and monetization framework

across albertlane.net and aros.albertlane.net, transforming standard search engine indexing into verifiable legal priority and automated monetization channels.
Public endpoints are partitioned across six functional paths:

- /research: Serves comprehensive architectural whitepapers and technical prompts embedded with cryptographic metadata.
- /wallet: Unifies multi-VM address generation, zero-knowledge witness construction, and decentralized identity management.
- /procurement: Houses the interactive county-level geospatial dashboard rendering satellite vegetative layers and staging purchase escrows.
- /api-xmoney: Acts as the automated financial gateway handling L402 Lightning micropayments and ERC-20 paymaster fee routing.
- /api-meta: Provides an open directory of W3C DID document schemas, cryptographic verification keys, and smart contract ABIs.
- /google-sequestration: Maintains a real-time ledger tracking search crawler indexing events, scraping signatures, and cryptographic prior-art attestations.

Every document published across the platform embeds machine-readable JSON-LD schemas and cryptographic headers inside the HTML structure. When commercial search crawlers—such as Googlebot and Bingbot—index the site, their public web caches permanently record the cryptographic Decentralized Identifier, the content hash, and the copyright licensing terms:

```
<meta name="aros:identity"
content="did:aros:caip10:eip1[span_88](start_span)[span_88](end_span)5
5:1890:0x71C8F67B98c253457a4B7F49eBb16f6b0fE353a9" />
<meta name="aros:ledger-receipt"
content="tx:0x89f6d4e21a367c3b90a618e38f12c3f91845bb02e5b871c5a7114256
c7d1e784" />
<meta name="ar[span_89](start_span)[span_89](end_span)os:merkle-root"
content="urn:sha256:7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284a
ddd200126d9069" />
<meta name="aros:licensing" content="Albert Lane Proprietary Software
License; Unauthorized Scrapes Actionable." />

<script type="application/ld+json">
{
  "@context": [
    "https://schema.org",
    {
      "did": "https://www.w3.org/ns/did/v1",
      "aros": "https://aros.albertlane.net/api-meta/v1#"
    }
  ],
  "@type": "TechArticle",
  "name": "AROS Sovereign Multi-VM Procurement and Agronomic
Protocol",
  "headline": "Universal Layer-1 Virtual Machine Integration for
Decentralized Agricultural Infrastructure",
  "url": "https://albertlane.net/research",
  "author": {
    "@type": "Person",
```

```
    "name": "Albert Lane",
   [span_90](start_span)[span_90](end_span) "url":
"https://albertlane.net",
    "identifier":
"did:pkh:eip155:1890:0x71C8F67B98c253457a4B7F49eBb16f6b0fE353a9"
  },
  "license":
"htt[span_91](start_span)[span_91](end_span)ps://albertlane.net/licens
e",
  "offers": {
    "@type": "Offer",
    "priceCurrency": "USD",
    "price": "0.00",
    "url": "https://aros.albertlane.net/procurement"
  },
  "aros:protocolRegistry": {
    "walletGatew[span_92](start_span)[span_92](end_span)ay":
"https://albertlane.net/wallet",
    "monetizationGateway": "https://aros.albertlane.net/api-xmoney",
    "sequestrationAudit":
"https://aros.albertlane.net/google-sequestration",
    "contentCommitmentMerkleRoot":
"0x7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069"
  }
}
</script>
```

When autonomous commercial crawlers or AI data-scraping user agents attempt to harvest prompt specifications or multi-spectral agricultural models from /procurement or /research, the edge proxy enforces dynamic monetization via L402 challenges. The gateway returns an HTTP 402 Payment Required header specifying a payment challenge. The autonomous agent automatically satisfies the micro-invoice via the Lightning Network or routes an ERC-20 payment through /api-xmoney, returning the settlement preimage in its authorization headers to access the underlying technical payload. This mechanism ensures that commercial LLM scrapers cannot consume proprietary engineering assets without providing instantaneous financial compensation.

Production Multi-VM Escrow Implementation

The smart contract deployed below exemplifies the operational intersection of multi-VM execution, pairing-based zero-knowledge proof verification, and synchronous inter-runtime communication. Operating within the EVM environment, the contract directly interfaces with the native BN254 pairing precompile at address 0\text{x}08 to verify producer parcel compliance, dispatches cross-VM execution calls to CosmWasm inventory actors via precompile 0\text{x}0800, and settles escrow releases through decentralized logistics oracles.

```
// SPDX-License-Identifier: Albert-Lane-Proprietary
// Copyright © Albert Lane. All Rights Reserved.
```

```
// Proprietary and Confidential — Albert Lane Proprietary Software
License.
// Unauthorized use, copying, modification, or distribution is
prohibited.
// Inquiries: https://albertlane.net / https://aros.albertlane.net

pragma solidity ^0.8.24;

interface ICosmWasmDispatcher {
    function executeCosmWasm(
        string calldata contractAddress,
        bytes calldata msgPayload,
        uint64 gasLimit
    ) external returns (bytes memory response);
}

contract AROSSovereignMultiVMEscrow {
    address public constant BN254_PAIRING_PRECOMPILE = address(0x08);
    address public constant COSMWASM_DISPATCHER =
address(0x0000000000000000000000000000000000000800);

    enum OrderStatus { None, Funded, InTransit, Delivered, Settled,
Cancelled }

    struct ProcurementOrder {
        bytes32 orderId;
        string countyFips;
        uint256 catalogSku;
        uint256 quantity;
        uint256 escrowAmount;
        address buyerAddress;
        address supplierAddress;
        address logisticsOracle;
        OrderStatus status;
        uint256 lockTimestamp;
        uint256 expiryTimestamp;
    }

    address public immutable protocolGovernor;
    mapping(bytes32 => ProcurementOrder) public orders;
    mapping(address => string) public accountDIDs;

    event DIDRegistered(address indexed account, string didURI);
    event EscrowInitiated(bytes32 indexed orderId, string countyFips,
address indexed buyer, uint256 amount);
    event CosmWasmNotified(bytes32 indexed orderId, bytes response);
    event EscrowCompleted(bytes32 indexed orderId, address supplier,
uint256 payout);
```

```
    modifier onlyGovernor() {
        require(msg.sender == protocolGovernor, "AROS: Unauthorized
governor");
        _;
    }

    constructor() {
        protocolGovernor = msg.sender;
    }

    function registerIdentity(string calldata didURI) external {
        require(bytes(didURI).length > 0, "AROS: DID payload empty");
        accountDIDs[msg.sender] = didURI;
        emit DIDRegistered(msg.sender, didURI);
    }

    function verifyAgronomicProof(
        uint256[2] calldata a,
        uint256[2][2] calldata b,
        uint256[2] calldata c,
        uint256[4] calldata input
    ) public view returns (bool) {
        uint256[24] memory pairingPayload;

        pairingPayload[0] = a[0];
        pairingPayload[1] = a[1];
        pairingPayload[2] = b[0][1];
        pairingPayload[3] = b[0][0];
        pairingPayload[4] = b[1][1];
        pairingPayload[5] = b[1][0];

        pairingPayload[6] = input[0];
        pairingPayload[7] = input[1];

        uint256[1] memory out;
        bool success;
        assembly {
            success := staticcall(
                sub(gas(), 2000),
                0x08,
                pairingPayload,
                0x300,
                out,
                0x20
            )
        }
        return success && out[0] == 1;
```

```
    }

    function initiateProcurement(
        bytes32 orderId,
        string calldata countyFips,
        uint256 catalogSku,
        uint256 quantity,
        address supplier,
        address oracle,
        uint256 duration,
        string calldata cosmWasmDepotActor,
        bytes calldata zkProof
    ) external payable {
        require(orders[orderId].status == OrderStatus.None, "AROS:
Order exists");
        require(msg.value > 0, "AROS: Empty escrow deposit");
        require(bytes(accountDIDs[msg.sender]).length > 0, "AROS:
Unregistered DID");

        if (zkProof.length > 0) {
            (
                uint256[2] memory a,
                uint256[2][2] memory b,
                uint256[2] memory c,
                uint256[4] memory pubInputs
            ) = abi.decode(zkProof, (uint256[2], uint256[2][2],
uint256[2], uint256[4]));
            require(verifyAgronomicProof(a, b, c, pubInputs), "AROS:
Invalid ZK proof");
        }

        orders[orderId] = ProcurementOrder({
            orderId: orderId,
            countyFips: countyFips,
            catalogSku: catalogSku,
            quantity: quantity,
            escrowAmount: msg.value,
            buyerAddress: msg.sender,
            supplierAddress: supplier,
            logisticsOracle: oracle,
            status: OrderStatus.Funded,
            lockTimestamp: block.timestamp,
            expiryTimestamp: block.timestamp + duration
        });

        emit EscrowInitiated(orderId, countyFips, msg.sender,
msg.value);
```

```
        if (bytes(cosmWasmDepotActor).length > 0) {
            bytes memory cwPayload = abi.encodeWithSignature(
                "reserve_inventory(string,uint256,uint256)",
                countyFips,
                catalogSku,
                quantity
            );
            bytes memory cwResponse =
ICosmWasmDispatcher(COSMWASM_DISPATCHER)
                .executeCosmWasm(cosmWasmDepotActor, cwPayload,
100000);
            emit CosmWasmNotified(orderId, cwResponse);
        }
    }

    function settleProcurement(bytes32 orderId) external {
        ProcurementOrder storage order = orders[orderId];
        require(msg.sender == order.logisticsOracle, "AROS: Caller not
assigned oracle");
        require(order.status == OrderStatus.Funded, "AROS: Order not
active");

        order.status = OrderStatus.Settled;
        uint256 payout = order.escrowAmount;

        (bool sent, ) = payable(order.supplierAddress).call{value:
payout}("");
        require(sent, "AROS: Settlement transfer failed");

        emit EscrowCompleted(orderId, order.supplierAddress, payout);
    }
}
```

Architectural Synthesis

The convergence of EVM execution, Substrate FRAME runtimes, and the Cosmos SDK establishes a comprehensive framework for decentralized infrastructure sovereignty. By executing EVM bytecode through Frontier and Revm engines while hosting CosmWasm within a Substrate WebAssembly meta-runtime, AROS delivers complete compatibility with existing Ethereum development tools while capturing the superior memory safety and execution guarantees of Rust-based smart contracts. Substrate's modular substrate layer enables forkless runtime upgrades, allowing protocol rules and cryptographic verification algorithms to evolve without chain splits.

Synchronous precompiled contract buses eliminate the multi-block latency and settlement risks inherent in cross-chain bridge architectures, enabling atomic execution across heterogeneous virtual machines within the same block. External interoperability remains secure through the

native integration of Cosmos IBC and Polkadot XCM, allowing the sovereign ledger to coordinate liquidity and messaging across the broader interchain landscape.
Universal fee abstraction ensures that physical producers, logistics operators, and autonomous AI agents interact with the ledger without managing volatile native gas assets. Stablecoins, localized asset balances, and off-chain Lightning micropayments serve as friction-free settlement currencies, lowering operational hurdles for non-technical users.
Furthermore, integrating NASA Earthdata HLS satellite telemetry links on-chain smart escrows directly to observable physical phenomena, establishing a closed-loop procurement system guided by automated remote sensing. Producers maintain privacy through zero-knowledge SNARK verification over the BN254 curve, proving land qualification and yield criteria without revealing confidential acreage or production volumes.
Anchored by cryptographically structured web metadata across albertlane.net and [span_201](start_span)[span_201](end_span)[span_204](start_span)[span_204](end_span)aros.albert[span_109](start_span)[span_109](end_span)[span_111](start_span)[span_111](end_span)lane.net, the platform establishes legal priority and prior art through regular search engine crawl indexing, while actively monetizing unauthorized automated access via L402 machine-to-machine micropayments. The resulting architecture provides an enterprise foundation for resilient, sovereign infrastructure and precision agricultural procurement.

Works cited

1. Smart Contract Development | Blockchain - Webisoft, https://webisoft.com/blockchain/smart-contract 2. Blockchain Protocols to Consider: A Chain Analysis, https://nextbigthing.ag/blog/blockchain-protocols-to-consider-a-chain-analysis/ 3. Cosmos - Polkadot Wiki, https://wiki.polkadot.com/learn/learn-comparisons-cosmos/ 4. Custom Layer 1 Blockchain Development Services - SpaceDev, https://spacedev.io/blockchain-development-services/layer-1-blockchain-development 5. What Is Composable Finance? DeFi Interoperability Explained, https://www.gate.com/learn/articles/what-is-composable-finance/4996 6. Building a Blockchain in 2026: From-Scratch Engineering vs, https://dev.to/thevenice/building-a-blockchain-in-2026-from-scratch-engineering-vs-modern-sdks-34jn 7. Cross-VM Interoperability | QoreChain Documentation, https://docs.qorechain.io/developer-guide/cross-vm-interoperability/ 8. Polkadot Summit - XC Smart Contracts Notes - Ecosystem, https://forum.polkadot.network/t/polkadot-summit-xc-smart-contracts-notes/3414 9. Grants-Program/applications/CosmWasmVM-CoreProduct.md at, https://github.com/w3f/Grants-Program/blob/master/applications/CosmWasmVM-CoreProduct.md 10. ComposableFi/composable release-v7.10027.0 on GitHub, https://newreleases.io/project/github/ComposableFi/composable/release/release-v7.10027.0 11. Technical Architecture - Cosmos Docs, https://docs.cosmos.network/evm/latest/documentation/cosmos-sdk/protocol 12. Technology - Interchain Foundation, https://interchain.io/technology 13. ComposableFi/composable-ibc: A trustless, cross-chain bridging, https://github.com/ComposableFi/composable-ibc 14. How to make your Arbitrum dApp chain-agnostic with Universal, https://docs.arbitrum.io/third-party-docs/Particle/particle 15. Gas Abstraction for AI Agents | Support - Eco, https://eco.com/support/en/articles/14846272-gas-abstraction-for-ai-agents 16. paritytech/polkadot-cli - GitHub, https://github.com/paritytech/polkadot-cli 17. caip-standards-expert | Skills Marke... · LobeHub,

https://lobehub.com/es/skills/notabene-id-skills-caip-standards-expert 18. Web 3中的零知识证明高承实主编| PDF - Scribd, https://www.scribd.com/document/789129559/Web-3%E4%B8%AD%E7%9A%84%E9%9B%B6%E7%9F%A5%E8%AF%86%E8%AF%81%E6%98%8E-%E9%AB%98%E6%89%BF%E5%AE%9E-%E4%B8%BB%E7%BC%96 19. Introducing `TransactionExtension` - Tech Talk - Polkadot Forum, https://forum.polkadot.network/t/introducing-transactionextension/10827 20. @polkadot/rpc-core | Yarn, https://chore-update--yarnpkg.netlify.app/en/package/@polkadot/rpc-core 21. Smart Contract & Blockchain Audit Portfolio - Codezen, https://www.codezen.tech/portfolio/ 22. Expanding the Interchain — taking IBC beyond Cosmos appchains, https://medium.com/the-interchain-foundation/expanding-the-interchain-taking-ibc-beyond-cosmos-appchains-b0f9bebb4405 23. The First KSM<>DOT Bridge - Composable Finance - Medium, https://medium.com/composable-finance/the-first-ksm-dot-bridge-be038fdd40a0 24. Adding Light Clients Seamlessly with the Wasm Light Client - IBC, https://ibcprotocol.dev/blog/wasm-client 25. Picasso Kusama and Composable Polkadot parachains - GitHub, https://github.com/composablefi/composable 26. Groth16 - Alin Tomescu, https://alinush.github.io/groth16 27. Groth16 Explained | RareSkills, https://rareskills.io/post/groth16 28. Groth16 - IOTA Documentation, https://docs.iota.org/developer/cryptography/on-chain/groth16 29. Proof VM - The ZK-CosmWasm Book, https://zk.permissionless.money/using/vm/proof-vm.html 30. How we implemented the BN254 Ate pairing in lambdaworks, https://blog.lambdaclass.com/how-we-implemented-the-bn254-ate-pairing-in-lambdaworks/ 31. BN254 Point Compression for L2s - Math & Engineering, https://xn--2-umb.com/23/bn254-compression/ 32. Fee abstraction guide - Celo Docs, https://docs.celo.org/build/fee-abstraction/overview 33. Gas Abstraction Explained: Removing Transaction Fee Friction, https://chain.link/article/gas-abstraction-explained 34. kitchensink_runtime/ lib.rs - Parity, https://paritytech.github.io/substrate/master/src/kitchensink_runtime/lib.rs.html 35. Normalized Difference Vegetation Index (NDVI) - NASA Earthdata, https://www.earthdata.nasa.gov/topics/land-surface/normalized-difference-vegetation-index-ndvi 36. HLS Solutions | NASA Earthdata, https://www.earthdata.nasa.gov/data/projects/nsite/solutions/hls 37. Vegetation Index | NASA Earthdata, https://www.earthdata.nasa.gov/topics/biosphere/vegetation-index 38. Harmonized Landsat and Sentinel-2 - NASA Earthdata, https://www.earthdata.nasa.gov/data/projects/hls 39. Getting Started with Cloud-Native HLS Data in Python - LP DAAC, https://lpdaac.usgs.gov/documents/842/HLS_Tutorial.html 40. HLS Landsat Operational Land Imager Surface Reflectance and, https://www.earthdata.nasa.gov/data/catalog/lpcloud-hlsl30-2.0 41. HLS Operational Land Imager Vegetation Indices Daily Global 30 m, https://www.earthdata.nasa.gov/data/catalog/lpcloud-hlsl30-vi-2.0