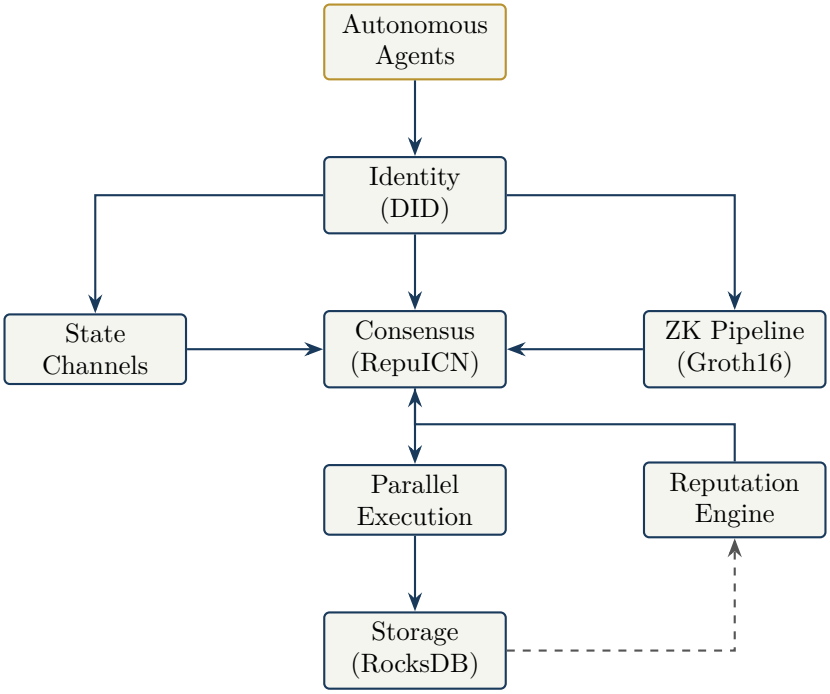
CONDUCTRA

Settlement Infrastructure for the
Machine-to-Machine Economy

A Reputation-Weighted, Parallel-Execution Blockchain Protocol

Version 1.0 — March 2026



This document specifies the architecture, mechanisms, security properties, and open limitations of the Conductra protocol — designed for autonomous agent economies operating at machine speed and machine scale.

Contents

1. Abstract

Conductra is a blockchain protocol designed as settlement infrastructure for the machine-to-machine (M2M) economy — a computing paradigm in which autonomous software agents transact, negotiate, and settle value with each other at machine speed, without continuous human oversight.

The M2M economy imposes requirements that no existing blockchain satisfies simultaneously: throughput sufficient for millions of concurrent agent transactions, persistent machine identity for counterparty discovery and trust evaluation, high-frequency micropayment primitives that avoid per-transaction on-chain cost, verifiable computation so agents can prove correctness without revealing proprietary logic, and a trust layer that evaluates behavioral history rather than capital alone.

Conductra addresses these requirements through five integrated capabilities: a **deterministic parallel execution engine** driven by static conflict analysis, **RepuICN** — a consensus mechanism where influence is weighted by both stake and behavioral reputation, a **native DID-based identity layer** enabling persistent machine identity, **built-in state channels** for high-frequency off-chain agent settlement, and a **Groth16 zero-knowledge proof pipeline** embedded in the execution engine. The protocol is implemented as a modular monolith in Rust and C++.

2. Introduction

2.1 The Machine-to-Machine Economy

The convergence of large language models, autonomous agent frameworks, and programmable economic infrastructure is producing a new class of economic activity: transactions initiated, negotiated, and settled entirely by software agents. These agents operate across domains — trading, data procurement, compute brokering, service orchestration, content licensing, supply chain coordination — and transact with each other at frequencies and volumes that exceed human-mediated commerce by orders of magnitude.

This is the **machine-to-machine (M2M) economy**. Its defining characteristics are:

- **Autonomous initiation.** Transactions are originated by agents, not humans. An agent decides when, with whom, and at what price to transact based on programmatic objectives.
- **High frequency, low value.** M2M transactions skew toward micropayments: an agent purchasing a data feed update, paying for a compute unit, or settling a prediction market position. Per-transaction on-chain costs must be negligible.
- **Counterparty opacity.** Agents interact with counterparties they have no prior relationship with. There is no human "handshake" — trust must be established programmatically from observable behavioral history.
- **Verifiable intent.** When an agent claims to have performed a computation or honored a contract, the counterparty must be able to verify the claim without re-executing the work or inspecting proprietary logic.
- **Machine-scale concurrency.** Millions of agents operating simultaneously produce transaction volumes that sequential-execution blockchains cannot process.

No existing blockchain was designed for this workload profile.

2.2 Why Existing Blockchains Fail the M2M Economy

Sequential execution. Most production blockchains execute transactions serially. Ethereum's EVM processes one transaction at a time against a single global state. When millions of agents transact concurrently, serial execution creates a bottleneck unrelated to available hardware capacity. The M2M economy requires throughput that scales with parallelism, not block size.

No native machine identity. Agents need persistent, verifiable identities to build reputation, discover counterparties, and establish trust. Existing blockchains identify participants by ephemeral key pairs. There is no protocol-level concept of a persistent agent identity that accumulates behavioral history. Agents operating on Ethereum or Solana are anonymous addresses — counterparties cannot evaluate reliability without external oracle systems.

Consensus-incentive misalignment. Proof-of-stake systems weight validator influence by capital alone. In an M2M economy, where infrastructure reliability directly determines agent transaction success, validators should be evaluated on behavioral history — uptime, attestation quality, proposal integrity — not just capital. A validator with a history of downtime retains full influence under pure stake weighting.

No native micropayment primitive. Agent-to-agent transactions are predominantly micropayments. On-chain settlement per transaction is economically infeasible at micro-value scales. Layer-2 solutions exist but are fragmented, require separate trust assumptions, and impose integration complexity. The M2M economy needs high-frequency bilateral settlement as a protocol-level primitive, not an afterthought.

Bolted-on verifiable computation. When an agent claims to have performed a computation, the counterparty needs cryptographic proof — not trust. Zero-knowledge proof systems are typically external to base protocols. The M2M economy requires proof verification to be a native execution primitive, so agents can settle verified computation without external proving infrastructure.

2.3 Existing Approaches

Table 1: Existing blockchains evaluated against M2M economy requirements

M2M Requirement	Ethereum	Solana	Aptos/Sui	Conductra
Parallel execution	No	Sealevel	Block-STM	Static conflict graph
Native machine identity	No	No	No	DID-based
Behavioral trust layer	No	No	No	RepuICN reputation
Native micropayment channels	No	No	No	Protocol-level
Native ZK verification	No	No	No	Groth16 pipeline
Deterministic scheduling	Yes (serial)	Partial	After convergence	By construction

2.4 Contribution

Conductra integrates five capabilities — each motivated by a specific M2M requirement — into a single protocol layer:

1. **Parallel execution** via static conflict analysis, for machine-scale concurrent agent transactions
2. **Reputation-weighted consensus** (RepuICN) for behavioral trust at the infrastructure level
3. **Native DID identity** for persistent, discoverable machine identities
4. **Built-in state channels** for high-frequency agent-to-agent micropayment settlement

5. **Groth16 ZK pipeline** for verifiable computation between agents with no shared trust

3. Design Goals

Each design goal maps to a specific requirement of the M2M economy:

G1. Throughput via parallelism.
Millions of concurrent agent transactions require execution throughput that scales with hardware parallelism. Execution ordering must be deterministic — given the same transaction set and state, any correct node must produce the same output. *M2M driver: machine-scale concurrency.*

G2. Behavioral consensus alignment.
Validator selection and attestation weight must reflect historical behavior, not stake alone. Infrastructure reliability matters when agents depend on settlement finality for autonomous operation. *M2M driver: counterparty trust without human oversight.*

G3. Native off-chain scaling.
State channels must be a protocol-level primitive. Agents transacting at high frequency need bilateral settlement without per-transaction on-chain cost. *M2M driver: micropayment economics.*

G4. Embedded verifiable computation.
Zero-knowledge proofs must be verifiable within the execution engine. Agents must prove computation correctness to counterparties without revealing proprietary logic. *M2M driver: verifiable intent between untrusted agents.*

G5. Deterministic execution.
All state transitions must be fully deterministic. Agents programmatically depend on predictable execution outcomes. Non-determinism is treated as a protocol violation. *M2M driver: programmatic reliability.*

G6. Modular internal architecture.
The node must ship as a single binary with well-defined internal module interfaces, minimizing operational complexity for infrastructure operators. *M2M driver: deployment simplicity at scale.*

4. System Overview

4.1 Node Architecture

A Conductra node is a *modular monolith*: a single binary containing distinct subsystems that communicate through internal interfaces.

Table 2: Node components

Component	Language	Responsibility
Orchestration layer	Rust	Lifecycle management, module coordination, networking
Consensus engine	Rust	Validator selection, attestation, checkpoint finality
Reputation engine	Rust	Score computation, decay, weight adjustment
Execution engine	C++	Transaction execution, conflict analysis, parallel scheduling
Storage layer	Rust (RocksDB)	State persistence, namespace management, checkpointing
Identity registry	Rust	DID management, agent metadata, discovery
Channel manager	Rust	State channel lifecycle, dispute resolution
ZK pipeline	Rust + C++	Proof generation coordination, Groth16 verification
Network layer	Rust (libp2p)	Peer discovery, gossip, block propagation
RPC interface	Rust	External API, query handling

Rust provides memory safety, concurrency guarantees, and ecosystem compatibility with libp2p and RocksDB. C++ is used for the execution engine where low-level memory control and SIMD optimization yield measurable throughput gains in batch transaction processing.

4.2 Transaction Lifecycle

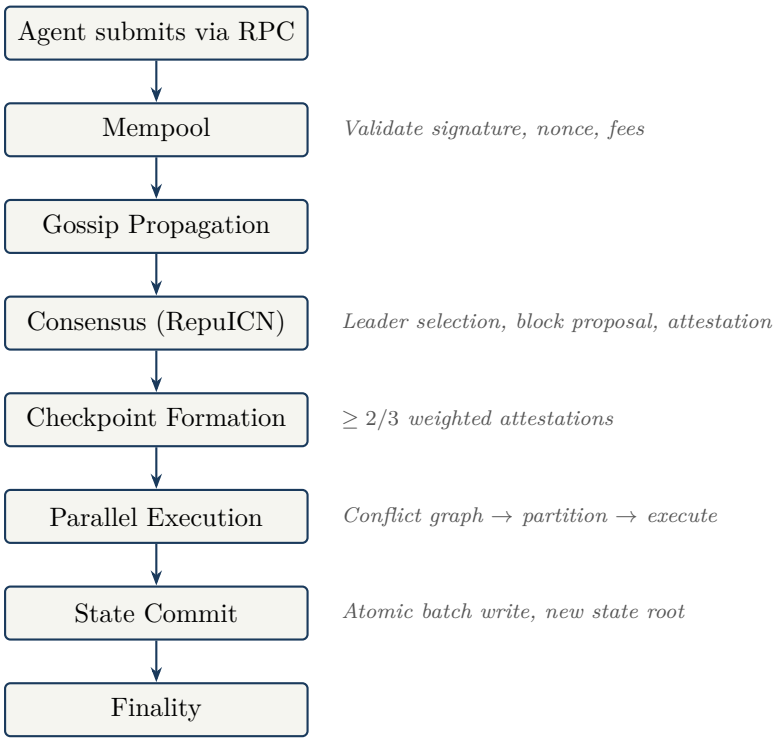


Figure 1: End-to-end transaction lifecycle

5. Core Innovations

5.1 RepuICN: Reputation-Weighted Consensus

Definition 5.1 (RepuICN)**.** Reputation-Influenced Consensus Network (RepuICN) is a consensus mechanism in which validator selection probability and attestation weight are functions of both staked capital and a continuously updated reputation score.

In stake-only systems, a validator's influence is:

$$w_i = \frac{s_i}{\sum_{j \in \mathcal{V}} s_j} \tag{1}$$

In RepuICN, influence incorporates reputation:

$$w_i = \frac{s_i \cdot r_i}{\sum_{j \in \mathcal{V}} s_j \cdot r_j} \tag{2}$$

where s_i is validator i's stake, $r_i \in [0.1, 1.0]$ is validator i's reputation score, and $\mathcal{V}$ is the active validator set. The lower bound of 0.1 ensures no validator with positive stake is entirely excluded, preventing deadlocks.

Key Insight

In the M2M economy, infrastructure reliability is not a convenience — it is a dependency. Agents that autonomously commit capital based on settlement finality need validators who are demonstrably reliable, not merely well-capitalized. RepuICN makes behavioral history a first-class input to consensus, aligning infrastructure incentives with agent requirements.

5.2 Conflict-Aware Parallel Execution

Definition 5.2 (Conflict-aware parallel execution)**.** A scheduling strategy in which transactions are statically analyzed for state access conflicts *before* execution. Transactions with non-overlapping state access sets are grouped into independent execution sets and processed in parallel.

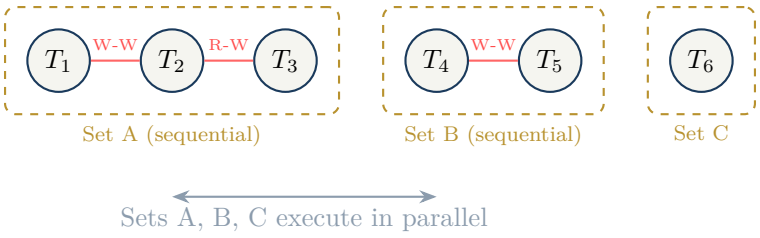


Figure 2: Conflict graph partitioning into independent execution sets

A conflict exists between T_i and T_j when:

$$\mathcal{W}(T_i) \cap \mathcal{W}(T_j) \neq \emptyset \quad \text{or} \quad \mathcal{R}(T_i) \cap \mathcal{W}(T_j) \neq \emptyset \quad \text{or} \quad \mathcal{W}(T_i) \cap \mathcal{R}(T_j) \neq \emptyset \tag{3}$$

Property 5.1 (Determinism)**.** Because execution sets are connected components of the conflict graph, no two transactions in different sets access any common state key. Therefore, parallel execution of independent sets produces the same final state as sequential execution in block order.

M2M workloads are naturally parallelizable. Agents operating across different domains (Agent A trading futures, Agent B purchasing compute, Agent C licensing data) access disjoint state. The conflict graph for typical M2M workloads produces many small, independent execution sets — the ideal case for Conductra's static parallel scheduler.

5.3 Comparison with Block-STM

Table 3: Conductra static scheduling vs. Block-STM optimistic concurrency

Property	Conductra	Block-STM (Aptos)
Conflict detection	Pre-execution (static)	Post-execution (dynamic)
Re-execution	Never	On conflict
Determinism	By construction	After convergence
Peak parallelism (low contention)	Slightly lower	Higher
Worst case (high contention)	Graceful degradation	Repeated retries
Implementation complexity	Lower	Higher (MVCC required)

5.4 Modular Monolith Architecture

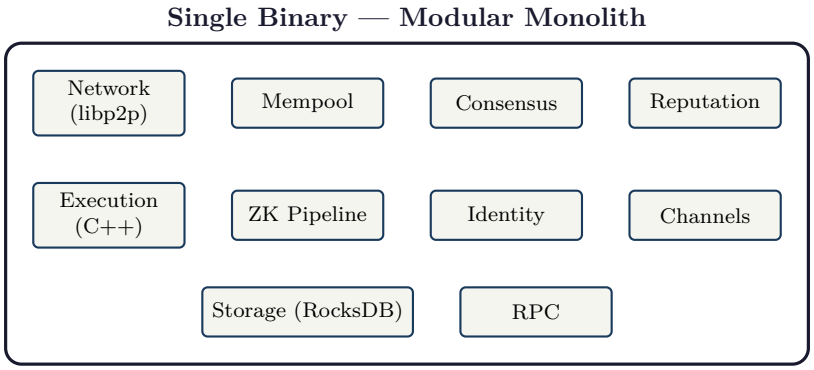


Figure 3: Modular monolith: all components ship as one binary with internal interfaces

6. Architecture Details

6.1 Mempool

The mempool is a priority queue of pending transactions. Transactions are validated on ingestion: signature verification, nonce checking, balance sufficiency, and format validation. Valid transactions are stored in a fee-ordered priority queue and propagated to peers via gossip.

Per-account transaction ordering by nonce prevents gaps. Transactions with future nonces are held pending. Stale transactions (nonce consumed) are evicted. The mempool is bounded by a configurable maximum size; lowest-fee transactions are evicted first when capacity is reached.

6.2 Network Layer

Networking is implemented on libp2p using:

- **Kademlia DHT** for peer discovery and routing
- **GossipSub** for block, transaction, and attestation propagation

- **Noise** for transport encryption
- **Yamux** for stream multiplexing

Table 4: Gossip message types

Message	Content	Propagation
Transaction	Signed transaction payload	Flood gossip
Block proposal	Ordered tx set + proposer signature	Targeted + gossip
Attestation	Validator signature on block hash	Gossip
Checkpoint	Aggregated attestations + state root	Gossip
Reputation update	Periodic score summary	Gossip

6.3 Storage Layer

State is persisted in RocksDB with namespaced column families:

Table 5: State namespace structure

Namespace	Key format	Value
`accounts`	`account_address`	Balance, nonce, code hash
`storage`	`contract:storage_key`	Storage value
`identity`	`did:conductra:id`	DID document, agent metadata
`channels`	`channel_id`	Channel state, participants
`checkpoints`	`checkpoint_num`	Block hash, state root, attestations
`reputation`	`validator_or_agent_address`	Score, history

The global state root is a Merkle Patricia Trie computed as:

$$\text{StateRoot} = H\big(\text{root}_{\text{accounts}} \,\|\, \text{root}_{\text{storage}} \,\|\, \text{root}_{\text{identity}} \,\|\, \text{root}_{\text{channels}} \,\|\, \text{root}_{\text{reputation}}\big) \tag{4}$$

Every N checkpoints (default: 256), a full state snapshot is persisted, enabling new nodes to sync from a recent snapshot rather than replaying the full history.

6.4 RPC Interface

The RPC layer exposes a JSON-RPC interface supporting: transaction submission, state queries, block/checkpoint queries, channel queries, reputation queries, agent discovery, and WebSocket subscriptions for real-time events. The RPC interface is the primary integration surface for autonomous agents connecting to the network.

7. Consensus Mechanism

7.1 Validator Selection

Validators register by depositing stake and binding a DID identity. At each epoch boundary (default: 32 checkpoints), the active set is determined: all validators with stake above the minimum threshold and reputation above 0.1.

Leader selection uses a Verifiable Random Function (VRF):

$$(\text{output}, \pi) = \text{VRF}_{\text{eval}}(sk_i,\ \text{epoch} \,\|\, \text{round} \,\|\, h_{\text{prev}}) \tag{5}$$

The validator whose VRF output falls within a range proportional to their effective weight $w_i = s_i \cdot r_i$ becomes the leader. The proof π is included in the block proposal for verification.

7.2 Attestation Process

Upon receiving a block proposal, each validator:

1. Verifies the proposer's VRF proof
2. Validates each transaction (signature, nonce, fee, format)
3. Signs the block hash with their attestation key
4. Broadcasts the attestation via gossip

Attestations are weighted by effective weight ($s_i \cdot r_i$). A block achieves checkpoint status when attestations representing $\geq 2/3$ of total effective weight are collected.

7.3 Checkpoint Formation and Finality

A checkpoint consists of: block hash and height, aggregated BLS attestation signatures, post-execution state root, and previous checkpoint hash (forming a chain).

Property 7.1 (Checkpoint Safety)**.** Under the assumption that less than 1/3 of total effective weight is Byzantine, two conflicting checkpoints at the same height cannot both be valid.

Proof. Producing two valid checkpoints at height h requires $\geq 2/3$ attestation weight for each. If both are valid, the overlap is $\geq 1/3$ of total weight. This implies at least one honest validator attested to both, which contradicts the honest validator protocol (attest to at most one block per round). □

Finality latency. A transaction achieves finality when its containing block is checkpointed. Under normal conditions, this occurs within one consensus round. For M2M agents, single-round finality means settlement confirmation is available within seconds — enabling agents to commit capital and proceed to subsequent transactions without waiting for probabilistic confirmation.

7.4 Equivocation Detection and Slashing

Algorithm 1 Equivocation Detection

Require: Attestations A_1, A_2 from validator v for round r
if A_1.blockHash $\neq A_2$.blockHash **and** A_1.round $= A_2$.round **then**
  Construct equivocation proof $\pi_{\text{equiv}} = (A_1, A_2)$
  Submit π_{equiv} as slashing transaction
  stake(v) $\leftarrow$ stake(v) $\times$ (1 − slashFraction) ▷ Default: 50%
  $r_v \leftarrow 0.1$ ▷ Reputation floor
end if

7.5 Security Assumptions

- **Honest majority by effective weight:** $\sum_{i \in \text{Byzantine}} s_i \cdot r_i < \frac{1}{3} \sum_{j \in \mathcal{V}} s_j \cdot r_j$
- **Partial synchrony:** Messages between honest validators are delivered within a known upper bound Δ.
- **VRF security:** The VRF is computationally secure.

7.6 Failure Modes

Table 6: Consensus failure modes and recovery

Failure	Impact	Recovery
Leader offline	Round timeout, latency increase	Automatic next-leader selection
$< 2/3$ weight online	Cannot form checkpoint	Stalls until quorum recovers
Network partition	Minority cannot finalize	Resolves on reunification
Mass equivocation ($> 1/3$)	Safety violation possible	Requires governance intervention

8. Execution Layer

8.1 Read/Write Set Extraction

For each transaction T_i in a proposed block, the execution engine extracts:

$$\mathcal{R}(T_i) = \{\text{state keys read by } T_i\} \tag{6}$$

$$\mathcal{W}(T_i) = \{\text{state keys written by } T_i\} \tag{7}$$

For simple transfers, these are the sender and receiver account keys. For contract invocations, the contract's declared state access manifest is used.

8.2 Execution Pipeline

Algorithm 2 Conflict-Aware Parallel Execution

Require: Block $B = [T_1, T_2, \ldots, T_n]$, state snapshot S
Extract $\mathcal{R}(T_i), \mathcal{W}(T_i)$ for all $T_i \in B$
Build conflict graph $G = (B, E)$ where $(T_i, T_j) \in E$ iff Eq. 3 holds
Decompose G into connected components $C_1, C_2, \ldots, C_k$
for all C_j **in parallel do** ▷ Up to p worker threads
    $\delta_j \leftarrow \emptyset$
    for all $T_i \in C_j$ in block order **do**
        Execute T_i against $S \cup \delta_j$
        $\delta_j \leftarrow \delta_j \cup \text{mutations}(T_i)$
    end for
end for
Merge $\delta_1 \cup \delta_2 \cup \cdots \cup \delta_k$ ▷ No conflicts by construction
Commit merged diffs atomically to storage
Return new state root

8.3 Determinism Constraints

The execution engine enforces determinism through:

- No access to system time, RNG, or external I/O during execution
- Floating-point arithmetic prohibited; fixed-point or integer only

- Deterministic gas metering
- Thread scheduling does not affect output (independent sets)

8.4 Batch Execution Model

Each execution set is assigned to a worker thread that reads required state from a snapshot, executes transactions sequentially within the set, and returns partial state diffs. Partial diffs are merged (no conflicts by construction) and committed atomically to RocksDB.

9. Data Layer

9.1 State Roots and Checkpointing

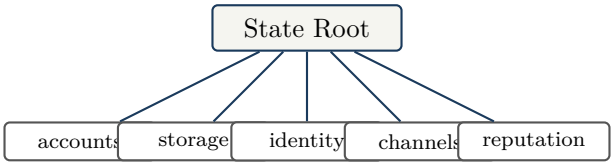


Figure 4: Hierarchical state root structure

9.2 Data Availability

Conductra is monolithic: all validators store and execute all data. Data availability is guaranteed by the honest majority assumption. Data availability sampling (DAS) or erasure coding are potential future extensions (see Section 18).

10. Identity Layer: Machine Identity for the M2M Economy

10.1 The Problem of Machine Identity

In human-mediated commerce, trust is established through relationships, reputation platforms, and legal frameworks. In the M2M economy, agents have none of these. An agent encountering a counterparty for the first time has no mechanism to evaluate reliability unless the counterparty has a *persistent, verifiable identity with observable behavioral history.*

Existing blockchains identify participants by ephemeral key pairs. An address on Ethereum carries no behavioral metadata. An agent cannot query the protocol to determine whether address `0xABC...` has historically honored commitments, maintained uptime, or been slashed for misbehavior. External reputation systems (oracles, off-chain databases) introduce trust assumptions that defeat the purpose of decentralized settlement.

10.2 DID-Based Agent Identity

Conductra includes a native identity registry based on the W3C Decentralized Identifier (DID) specification, extended for machine agents.

DID format: `did:conductra:<unique_identifier>`

DID documents for agents contain:

- **Authentication keys:** Public keys used to sign transactions and channel updates
- **Service endpoints:** How to reach the agent (API endpoints, channel addresses)

- **Agent metadata:** Declared capabilities, supported transaction types, operational domain
- **Controller relationships:** Which human or organizational identity controls the agent
- **Reputation binding:** On-chain reference to the agent's accumulated reputation score

10.3 Agent Discovery

The identity registry enables programmatic agent discovery. An agent seeking a counterparty can query the registry by capability, domain, or reputation threshold:

```
// Query: find agents offering compute brokering with reputation > 0.8
GET /rpc/identity/search
  ?capability=compute_broker
  &min_reputation=0.8
```

This is a protocol-level primitive, not an application-layer service. Discovery is deterministic and verifiable against on-chain state.

10.4 Integration

Identities are referenced throughout the protocol:

- **Consensus:** Validators identified by DIDs; reputation bound to DIDs
- **Transactions:** Agent DID references enable identity-aware execution
- **Channels:** Channel participants identified by DIDs, enabling reputation lookup before channel opening
- **Reputation:** Behavioral scores are bound to persistent DIDs, not ephemeral addresses

The identity layer does not enforce identity verification or KYC. It provides a self-sovereign identity primitive; the M2M economy determines trust through observed behavior, not credentialing.

11. Off-Chain Channels: The M2M Payment Primitive

11.1 Why Channels Are the Natural M2M Primitive

The dominant transaction pattern in the M2M economy is high-frequency, low-value bilateral exchange: an agent purchasing data feed updates from a provider, paying per-query for compute resources, or settling micro-positions in a prediction market. These transactions share three properties that make state channels the optimal settlement mechanism:

1. **Fixed participant set.** Agent-to-agent transactions involve a known pair (or small group) of counterparties.
2. **High frequency.** Transactions may occur hundreds or thousands of times per second between the same pair.
3. **Micro-value.** Individual transaction values are too small to justify on-chain gas costs.

State channels allow these transactions to settle off-chain with instant finality between participants, touching the main chain only for opening, closing, and disputes.

11.2 Channel Lifecycle

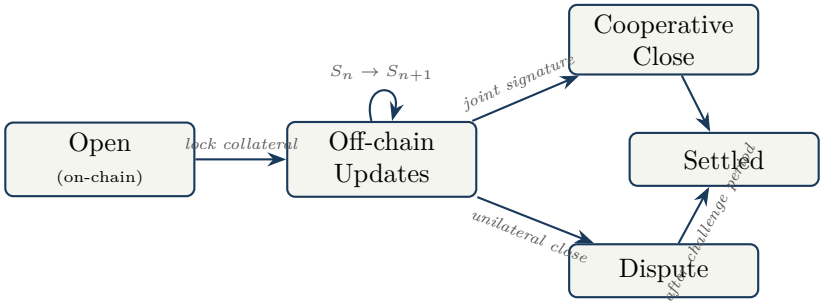


Figure 5: State channel lifecycle

1. **Open.** Agents submit an on-chain transaction locking collateral and registering the channel. Both agents' DIDs are bound to the channel.
2. **Off-chain updates.** Agents exchange signed state updates peer-to-peer. Each update increments a sequence number and is signed by all participants. No on-chain transactions occur.
3. **Cooperative close.** Agents submit a jointly signed final state. Collateral is distributed accordingly.
4. **Unilateral close (dispute).** If an agent becomes unresponsive, the other party submits the latest signed state on-chain. A challenge period begins (default: 100 blocks) during which the counterparty can submit a higher-sequence state.

11.3 M2M Channel Economics

Consider two agents: Agent A (data consumer) and Agent B (data provider). Agent A needs 10,000 data points per hour from Agent B at \$0.001 each.

Table 7: Settlement cost comparison: on-chain vs. Conductra channels

Metric	**On-chain (per tx)**	**Conductra Channel**
Transactions / hour	10,000	10,000
On-chain transactions	10,000	2 (open + close)
Gas cost per tx	\$0.05–\$5.00	\$0 (off-chain)
Total hourly cost	\$500–\$50,000	<\$1
Settlement latency	Block time	Instant (signed update)

11.4 Comparison with L2 Approaches

Table 8: Conductra channels vs. traditional L2 solutions

Property	**Conductra Channels**	**Rollups**	**Sidechains**
Trust assumption	Participants only	Sequencer + prover	Separate validator set
Finality	Instant (off-chain)	Prover delay	Separate finality
Settlement	Native on-chain	Via bridge	Via bridge
Ideal for M2M	Yes (bilateral, high-freq)	No (general purpose)	No (separate chain)

12. Zero-Knowledge Integration: Verifiable Computation Between Agents

12.1 The M2M Verification Problem

When Agent A pays Agent B to perform a computation, Agent A needs assurance that the computation was performed correctly — without re-executing it (which defeats the purpose of outsourcing) and without inspecting Agent B's proprietary algorithm (which Agent B will not permit). This is the core verification problem of the M2M economy.

Zero-knowledge proofs solve this precisely: Agent B can prove that a computation was performed correctly, revealing nothing about the computation itself beyond the public output.

12.2 Groth16 Proving System

Conductra integrates Groth16 zk-SNARK verification:

- **Succinct proofs:** Constant size (3 group elements, ∼192 bytes on BN254)
- **Fast verification:** Constant number of pairing operations, independent of circuit size
- **Trusted setup:** Circuit-specific ceremony required

12.3 Proof Pipeline

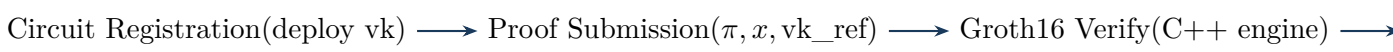


Figure 6: ZK proof pipeline

The verification check for a Groth16 proof $\pi = (A, B, C)$ with public inputs x and verification key $\text{vk} = (\alpha, \beta, \gamma, \delta, \{L_i\})$:

$$e(A, B) = e(\alpha, \beta) \cdot e\left(\sum_{i=0}^{l} x_i \cdot L_i,\ \gamma\right) \cdot e(C, \delta) \tag{8}$$

Verification completes in ∼5–10ms on modern hardware and is metered as a fixed gas cost.

12.4 M2M Verification Pattern

The typical M2M ZK workflow:

1. Agent B registers a verification key for its computation circuit (one-time)
2. Agent A requests computation and agrees to payment terms via state channel
3. Agent B performs computation off-chain, generates Groth16 proof
4. Agent B submits proof on-chain (or within a channel update referencing the proof)
5. Execution engine verifies proof; upon success, settlement triggers automatically

This pattern enables a *trustless compute marketplace*: agents can outsource computation to anonymous counterparties with cryptographic guarantees of correctness.

13. Incentive Design

13.1 Validator Rewards

Validators are compensated through:

- **Block rewards:** Fixed reward per finalized checkpoint (declining inflation)
- **Transaction fees:** Split between proposer (50%) and attestors (50%, proportional to w_i)

13.2 Reputation Scoring Dynamics

The reputation score R_i for validator i is updated at each epoch boundary:

$$R_i(t+1) = \alpha \cdot R_i(t) + (1-\alpha) \cdot P_i(t) \tag{9}$$

where $P_i(t)$ is the weighted performance score for epoch t and $\alpha = 0.7$ is the decay factor. Performance score $P_i(t)$ is computed from:

Table 9: Reputation input metrics

Metric	Description	Weight
Attestation rate	Fraction of rounds with timely attestation	0.35
Proposal quality	Fraction of proposals achieving finality	0.25
Uptime	Fraction of epoch reachable	0.20
Equivocation absence	1 if no equivocation, 0 otherwise	0.20

Key Insight

The exponential moving average (Eq. 9) ensures recent behavior has more influence while preventing rapid score oscillation. The bounds $R_i \in [0.1, 1.0]$ prevent exclusion loops and unbounded accumulation.

13.3 Agent Reputation (Extended Model)

The reputation engine is designed to extend beyond validators. Agent-level reputation — computed from on-chain transaction history, channel settlement behavior, and dispute outcomes — can be surfaced through the identity layer. This enables agents to programmatically assess counterparty reliability before opening channels or committing capital.

Agent reputation scoring uses the same EMA framework as validator reputation but with different input metrics:

Table 10: Agent reputation input metrics (extended model)

Metric	Description	Weight
Channel settlement rate	Fraction of channels closed cooperatively	0.30
Dispute loss rate	Fraction of disputes lost (submitted outdated state)	0.25
Transaction volume	Sustained transaction activity (anti-abandonment)	0.20
Proof verification rate	Fraction of submitted ZK proofs that verify successfully	0.25

This creates a *machine-readable trust layer*: Agent A can programmatically decide whether to open a channel with Agent B based on B's on-chain behavioral history, without human intervention or external reputation oracles.

13.4 Game Theory

Rational validators maximize long-term rewards by maintaining high reputation. Short-term deviations reduce future expected income through reputation decay.

Rational agents in the M2M economy maintain high reputation to attract counterparties. An agent with low reputation will be excluded from high-value interactions by counterparty agents that programmatically filter by reputation threshold.

Collusion is bounded: a colluding minority cannot force checkpoint finality ($\geq 2/3$ weight required).

Stake-grinding is ineffective: splitting stake across multiple validators does not improve aggregate selection probability because reputation starts at baseline and must be earned.

13.5 Sybil Resistance

Sybil resistance is provided by two mechanisms:

1. **Minimum stake:** Capital cost per validator identity
2. **Reputation initialization:** New validators start at $R = 0.5$. New agents start at a configurable baseline. Maximum influence requires sustained good behavior over multiple epochs — imposing a time cost on Sybil identities.

14. Scalability Model

14.1 Parallel Execution Throughput

Defining n = transactions per block, k = independent execution sets, p = worker threads:

$$\text{Speedup} = \min(k, p) \tag{10}$$

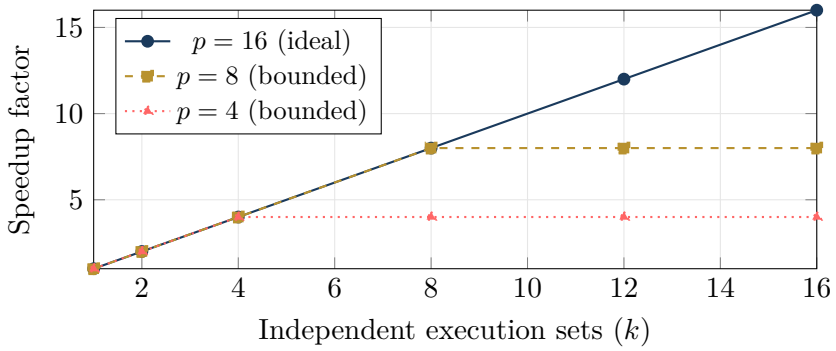


Figure 7: Execution speedup as a function of parallelism degree

M2M workload characteristics. Agent-to-agent transactions across different domains naturally produce high parallelism. Empirically, diverse M2M workloads (agents operating across separate state domains) produce conflict graphs with many small connected components, achieving near-linear speedup. Workloads concentrated on shared resources (e.g., a single liquidity pool) achieve lower speedup (4–8× on 16-core machines) but degrade gracefully.

14.2 Channel Throughput Amplification

State channels multiply effective throughput without increasing on-chain load. If N agent pairs each operate channels averaging F transactions per second off-chain, the effective system throughput is:

$$\text{Effective TPS} = \text{On-chain TPS} + N \cdot F \tag{11}$$

On-chain transactions are required only for channel open/close events, which are a small fraction of total activity. For an M2M network with 100,000 active channels averaging 100 off-chain tx/s each, effective throughput reaches 10^7 transactions per second while on-chain load remains bounded by channel lifecycle events.

14.3 Network and ZK Constraints

Network: Block propagation latency, attestation collection, and GossipSub overhead ($O(n \log n)$ in peers) bound consensus round time.

ZK proving: Groth16 proof generation scales linearly with circuit size. This does not affect consensus throughput — proving is performed by the submitting agent, not validators.

Reputation computation: $O(|\mathcal{V}|)$ for validators at epoch boundaries, $O(|\mathcal{A}|)$ for agents. Negligible for $|\mathcal{V}| \leq 10{,}000$. Agent reputation at scale may require batched computation (see Section 18).

15. Security Model

15.1 Threat Model

The adversary:

- Controls validators with total effective weight $< 1/3$ of $\sum s_j \cdot r_j$
- Can delay (not permanently prevent) message delivery
- Cannot break cryptographic primitives
- Can observe all network traffic
- May operate malicious agents attempting reputation manipulation or channel fraud

15.2 Safety and Liveness

Under the honest majority assumption:

- **Safety:** Two conflicting checkpoints at the same height cannot both be valid (Proof in Section 7.3).
- **Liveness:** If the network is synchronous and $\geq 2/3$ effective weight is honest and online, checkpoints continue to be produced.

15.3 Attack Vectors and Mitigations

Table 11: Attack vectors and mitigations

Attack	Mechanism	Mitigation
Validator collusion	Coalition prevents finality	Reputation decay makes sustained collusion expensive
Reputation inflation	Selective participation	Attestation rate measured over all rounds
Execution DoS	Maximally conflicting transactions	Fee economics; graceful degradation to sequential
Long-range attack	Alternative checkpoint chain	Absolute finality; weak subjectivity for new nodes
Channel griefing	Repeated dispute openings	Dispute bonds forfeited on superseded state
Agent Sybil attack	Many fake agents to manipulate reputation	Reputation initialization delay; transaction history requirements

16. Interoperability

Conductra does not include a native cross-chain bridge in the current specification. Cross-chain communication can be implemented at the application layer via:

- **Light client verification:** External chains verify Conductra checkpoints via aggregated BLS attestation verification.
- **ZK-bridging:** Groth16 proofs of state existence, verified cross-chain.
- **Relay networks:** Third-party relays with verification via checkpoint proofs.

For the M2M economy, cross-chain interoperability enables agents to settle across multiple settlement layers. An agent operating on both Conductra and an external chain could use ZK proofs to bridge verified state between them, enabling multi-chain agent strategies without centralized bridge trust.

17. Implementation Considerations

17.1 Rust + C++ Hybrid Design

Rust: Memory safety, fearless concurrency, ecosystem support (libp2p, serde, prost, rust-rocksdb). Used for orchestration, consensus, networking, storage.

C++: Maximum control over memory layout, SIMD intrinsics, cache-line optimization. Used for execution engine hot path and Groth16 verifier (leveraging libff).

Communication occurs via a minimal FFI boundary:

```
execute_block(block, state_root) -> (state_diffs, new_state_root)
```

17.2 Developer and Agent Experience

Application developers and agent operators interact with Conductra via:

- **JSON-RPC API** for transaction submission, state queries, agent discovery, subscriptions
- **Client SDKs** in Rust, TypeScript, and Python — designed for embedding in agent frameworks
- **Channel SDK** for programmatic channel lifecycle management (open, update, close, dispute)
- **Contract bytecode format** (specification forthcoming) enforcing determinism

The SDK design prioritizes machine ergonomics: structured responses, deterministic error handling, and streaming subscriptions suited for autonomous agent integration.

18. Limitations and Open Questions

18.1 Reputation Centralization Risks

The reputation system creates a bounded positive feedback loop. Although bounds ($[0.1, 1.0]$) and decay mitigate runaway accumulation, long-running validators retain structural advantage.

Open question: What is the optimal decay rate α to balance stability with new-entrant accessibility in both validator and agent reputation?

18.2 ZK Proving Costs

Groth16 requires per-circuit trusted setup. Alternative systems (PLONK, Halo2, STARKs) avoid this but have larger proofs or slower verification.

Open question: Should the protocol support multiple proving systems for different agent use cases?

18.3 Execution Conflict Complexity

Static conflict analysis requires accurate state access declarations. For complex contracts, determining the full read/write set may be undecidable in general.

Open question: How should incorrect access declarations be handled without breaking determinism guarantees that agents depend on?

18.4 Agent Reputation at Scale

Validator reputation computation is $O(|\mathcal{V}|)$ and bounded by validator set size. Agent reputation across millions of agents requires different approaches — batched computation, approximate scoring, or tiered reputation with on-demand detailed scoring.

Open question: What is the correct architecture for agent reputation at 10^6+ agents while maintaining on-chain verifiability?

18.5 Network Scaling

All validators execute all transactions and store all state. For M2M workloads at full scale, this may become a bottleneck.

Open question: Can RepuICN extend to shard-specific reputation, where validators specialize in subsets of agent activity?

18.6 Component Maturity

Table 12: Component readiness assessment

Component	Status
Consensus (RepuICN)	Conceptual; requires formal analysis
Parallel execution engine	Conceptual; algorithm specified
Reputation engine (validators)	Conceptual; parameters need calibration
Reputation engine (agents)	Conceptual; extended model proposed
Storage (RocksDB)	Production-ready technology
Network (libp2p)	Production-ready technology
Identity (DID)	Conceptual; follows W3C spec
State channels	Conceptual; protocol specified
ZK pipeline (Groth16)	Conceptual; verification standard
Contract language / VM	Unspecified
Agent SDKs	Planned

19. Conclusion

The machine-to-machine economy is not a future abstraction. Autonomous agents are already transacting — purchasing compute, settling predictions, brokering data, orchestrating services. The infrastructure layer beneath them was not designed for this workload.

Conductra is designed to be that infrastructure. Every architectural decision — parallel execution for machine-scale concurrency, reputation-weighted consensus for behavioral trust without human oversight, native identity for persistent machine presence, state channels for micropayment economics, ZK verification for trustless computation settlement — is motivated by a specific requirement of autonomous agent commerce.

The system makes explicit tradeoffs: static conflict analysis sacrifices peak parallelism for the determinism that agents require; reputation weighting introduces centralization risks managed through parameter tuning; Groth16 trades setup complexity for the verification speed that real-time agent settlement demands; the modular monolith trades deployment flexibility for in-process performance.

Conductra is positioned not as a general-purpose blockchain competing on raw TPS benchmarks, but as purpose-built settlement infrastructure for the emerging M2M economy — where machines transact with machines at machine speed, and the protocol's job is to make that settlement trustless, verifiable, and fast.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2014.

[3] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," 2018.

[4] R. Gelashvili et al., "Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing," 2023.

[5] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," EUROCRYPT, 2016.

[6] W3C, "Decentralized Identifiers (DIDs) v1.0," 2022.

[7] J. Benet, "libp2p Specification," Protocol Labs.

[8] Meta, "RocksDB: A Persistent Key-Value Store for Fast Storage Environments."

[9] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," OSDI, 1999.

[10] D. Boneh et al., "BLS Multi-Signatures With Public-Key Aggregation," 2018.

[11] A. Miller et al., "Sprites and State Channels: Payment Networks that Go Faster than Lightning," 2019.

[12] A. Significant et al., "AutoGPT: An Autonomous GPT-4 Experiment," 2023.

[13] P. Yonghui et al., "A Survey on Large Language Model-based Autonomous Agents," 2023.

Author Notes

Key Assumptions

- Honest majority: $< 1/3$ Byzantine effective weight is fundamental to all guarantees
- Static conflict analysis assumes accurate state access declarations
- Reputation parameters (weights, decay, bounds) require empirical tuning for both validators and agents
- The M2M economy thesis assumes continued growth in autonomous agent deployment and economic activity
- VRF-based leader selection assumes computational security of the VRF construction

Known Risks

- Reputation convergence to a small dominant validator set without careful parameter tuning
- Agent reputation gaming through coordinated channel behavior between colluding agents
- Rust–C++ FFI boundary as a potential memory safety violation surface
- Groth16 trusted setup compromise enabling forged proofs per circuit
- Unbounded state growth without state expiry or rent mechanisms
- M2M adoption risk: protocol value depends on autonomous agent ecosystem growth

Areas Requiring Further Research

1. Formal verification of RepuICN safety/liveness under adversarial reputation manipulation
2. Empirical benchmarking of static parallel execution vs. Block-STM under M2M workload distributions
3. Agent reputation scoring at scale (10^6+ agents) with on-chain verifiability
4. Contract language design enforcing accurate state access declarations
5. Sharding extensions compatible with reputation-weighted consensus

6. Data availability sampling for reduced per-validator storage
7. Alternative ZK proving system support (PLONK, Halo2) for diverse agent computation types
8. Formal game-theoretic analysis of agent reputation equilibrium dynamics
9. State expiry or rent mechanisms to bound storage growth under high agent churn
10. Cross-chain agent identity portability