UKIYO NETWORK · COMBINED TECHNICAL WHITE PAPER

Ukiyo Network

A production-verified credit system for the industrial economy

Document type: Combined system white paper · Version 1.0

Execution environment: Arbitrum One · Settlement root: Ethereum L1

Components: SPOOL Protocol · Fulfillment Oracle · Agency for Smart Contracts

Classification: Public

Contents

1. Executive Summary
2. The Network Thesis
3. System Architecture: Three Layers, One System
4. Component Relationship Matrix
5. The Fulfillment Oracle — Evidence Layer
6. The Agency for Smart Contracts — Governance Layer
7. SPOOL Protocol — Credit and Settlement Layer
8. Cross-Layer Interfaces
9. End-to-End Flow: A Worked Transaction
10. Arbitrum L2 Architecture
11. Security Model: Separation of Powers
12. Failure Coupling and Degraded Modes
13. Testing and Verification Strategy

14. Deployment Topology and Observability

15. Governance

16. Risk Factors

1. Executive Summary

Ukiyo Network builds AI, oracle, and blockchain systems that make industrial production legible in real time, and turn that legibility into credit and settlement at industrial scale. The network is a single system composed of three layers, each answering a distinct question that industrial credit depends on.

The **Fulfillment Oracle** answers *what is true* about a production order. The **Agency for Smart Contracts** answers *what may be done* on the basis of that truth. **SPOOL Protocol** answers *what credit and settlement follows*. Together they form an unbroken chain from factory floor to on-chain credit, with attested evidence at the base, bounded authority in the middle, and capital movement at the top.

The network deploys on Arbitrum One for execution and anchors to Ethereum L1 for finality. Arbitrum's ~250ms block time makes continuous risk response viable, EIP-4844 blob economics make frequent oracle attestation affordable, and Stylus runs compute-heavy credit logic in Rust at a fraction of EVM gas costs. The result is a credit system that responds to production as it happens rather than at quarterly reporting intervals.

2. The Network Thesis

Industrial finance is constrained not by a shortage of capital but by a shortage of legible, continuous, verifiable information about production. Lenders cannot observe the factory floor; they observe documents about the factory floor, delivered weeks after the fact.

Where production can be observed in real time and attested cryptographically, credit can be priced and executed in real time as well. Each of the three layers removes one specific obstacle to that outcome.

> *AI, oracle, and blockchain systems that make production legible in real time — and turn that truth into markets, credit, and settlement at industrial scale.*

Obstacle	Layer that removes it	Mechanism
Evidence is slow, documentary, and single-sourced	Fulfillment Oracle	Continuous multi-source fusion, cryptographic attestation, fail-closed publication
Automation is either unbounded or too slow	Agency for Smart Contracts	Allowlisted actions, argument bounds, simulation gating, human escalation
Credit terms are periodic rather than continuous	SPOOL Protocol	Score-linked advances, continuous repricing, automated repayment waterfall

3. System Architecture: Three Layers, One System

The three layers are not independent products. They form a directed dependency chain in which each layer consumes the output of the one beneath it and produces an input for the one above it.



> **SPOOL Protocol — credit and settlement**
>
> A production-verified credit network where AI agents watch fulfillment oracles and execute allowlisted smart-contract actions: minting order NFTs, funding industry spools, and advancing stablecoins to suppliers.

Fulfillment Oracle — evidence

Oracle agents fuse ERP, computer vision, and logistics into a live Fulfillment Probability Score, cryptographically attested so agents never trust a PDF or a single opaque server.

Agency for Smart Contracts — governance

Policy-bounded DeFi agents that sit between live oracle feeds and on-chain markets, executing only allowlisted risk actions and escalating high-impact moves to human multi-sig.

3.1 Layer responsibilities

Layer	Holds	Produces	Never does
Fulfillment Oracle	Data connectors, scoring models, attestation keys	Attested Fulfillment Probability Scores	Move funds, decide policy, hold capital
Agency for Smart Contracts	Policy engine, allowlists, multi-sig authority	Approved or escalated call decisions	Produce evidence, hold capital
SPOOL Protocol	Order NFTs, industry spools, stablecoin balances	Credit positions, advances, settlements	Produce evidence, decide its own policy

3.2 Trust boundaries

Each boundary between layers is a point where a security property is either preserved or lost. No single layer's compromise is sufficient to move capital outside policy.

Boundary	Crossing mechanism	Property enforced
Industrial → Oracle	ERP APIs, computer vision, logistics feeds	Source diversity; no single feed is authoritative
Oracle → Agency	EIP-712 attested score publication	Provenance and integrity; freshness window

Agency → SPOOL	`IPolicyEngine.evaluate` gate	Allowlist, argument bounds, simulation state diff
SPOOL → L2/L1	Arbitrum execution, Ethereum settlement root	Finality per action class; sequencer failure containment

4. Component Relationship Matrix

Every layer connects to every other layer in a defined direction. The matrix below is the authoritative statement of those connections.

From ↓ / To →	Fulfillment Oracle	Agency for Smart Contracts	SPOOL Protocol
Fulfillment Oracle	—	Supplies attested scores the Agency watches; state transitions (`Paused` , `Degraded`) are escalation inputs	Supplies attested scores SPOOL verifies before any order state change
Agency for Smart Contracts	Reads scores; treats Oracle state transitions as triggers	—	Gates every SPOOL-mutating call; escalates high-impact actions to multi-sig
SPOOL Protocol	Defines order identifiers (`supplierId` , `buyerId` , `deadline`) the Oracle scores against	Provides the state the Agency simulates against and the targets its allowlist names	—

4.1 Dependency properties

Relationship	Type	Failure propagation
Oracle → SPOOL	Hard dependency	Oracle Paused halts new advances via the freshness window

Oracle → Agency	Hard dependency	Oracle Paused triggers Agency escalation
Agency → SPOOL	Hard dependency for agent actions	Agency Paused stops new agent actions; settlement in flight completes
SPOOL → Agency	Simulation dependency	SPOOL Paused causes Agency advance proposals to revert and escalate
SPOOL → Oracle	Identifier dependency	Missing order identifiers leave scores unbound; credit cannot proceed
Agency → Oracle	Observation dependency	Missed state transition means the Agency continues proposing against a fail-closed Oracle

4.2 What each layer guarantees to the others

Layer	Guarantees to the network
Fulfillment Oracle	Scores are attested by authorized signers; stale scores are distinguishable from valid ones; publication fails closed when feeds go dark; L1 anchoring continues independently of L2 publication.
Agency for Smart Contracts	No agent call reaches SPOOL unless it passes the allowlist, argument bounds, and simulation; high-impact actions require multi-sig; keys and limits never enter model context.
SPOOL Protocol	Order state transitions are guarded by caller and state; advances are bounded by the attested score; the repayment waterfall cannot be blocked by a failing recipient; losses apply to junior capital first.

5. The Fulfillment Oracle — Evidence Layer

The Oracle converts physical production data into a form that on-chain contracts can verify and act upon. It does not custody funds and does not execute credit actions; its authority is limited to attesting a score.

5.1 Ingestion

Three data classes are ingested, each with different latency, schema, and reliability characteristics.

Source	Typical latency	Schema volatility	Handling
ERP APIs	Minutes to hours	High (vendor-specific)	Connector layer with per-vendor adapters
Computer vision	Seconds	Low (image + metadata)	Streaming inference pipeline
Logistics APIs	Minutes	Medium	Polling plus webhook ingestion



The normalizer rejects events with missing required fields rather than imputing them silently. Silent imputation produces a score that appears valid but is not grounded in observed production.

5.2 Scoring

The Fulfillment Probability Score is a continuous estimate of on-time, to-specification fulfillment, produced by a PyTorch model.

Feature family	Examples	Source
Order characteristics	Notional, lead time, complexity, deadline slack	ERP
Production velocity	Units completed per shift, cycle time trend	ERP, vision
Quality signals	Defect rate, rework events, visual anomaly score	Vision
Supplier history	On-time rate, past delays, capacity utilization	ERP

Logistics state	Carrier status, transit time deviation, port congestion	Logistics

The model outputs a calibrated probability, not a raw logit. Calibration is monitored per vertical, because a shift in Auto does not imply a shift in Pharma.

5.3 Attestation

Attestation turns a score into evidence. Consumers verify EIP-712 typed data signatures that bind the score to a domain (chain ID, verifying contract) and a structured payload.

```
bytes32 constant SCORE_TYPEHASH = keccak256(
    "Score(bytes32 orderId,uint16 scoreBps,uint64 timestamp,bytes32 dataRoot)"
);

struct Score {
    bytes32 orderId;
    uint16  scoreBps;
    uint64  timestamp;
    bytes32 dataRoot; // Merkle root of supporting evidence
}

function verifyScore(
    Score calldata s,
    bytes calldata signature,
    address authorizedOracle
) internal view returns (bool) {
    bytes32 digest = _hashTypedDataV4(
        keccak256(abi.encode(SCORE_TYPEHASH, s.orderId, s.scoreBps, s.timestamp, s.dataRoot))
    );
    return ECDSA.recover(digest, signature) == authorizedOracle;
}
```

Threshold signatures (k-of-n oracle nodes) are supported for scores that carry higher weight, preventing any single signer from publishing a score unilaterally.

5.4 Fail-closed publication

State	Trigger	Behavior
Normal	All feeds fresh; model healthy	Publish scores to Arbitrum

Degraded	One feed stale; model drift detected	Publish with uncertainty flag; consumers may widen haircuts
Paused	Critical feed absent or attestation failure	Stop publishing; emit pause event
Recovering	Feeds restored; model re-validated	Resume with conservative scores; monitor

Consumers distinguish "no score" from "score of zero." The registry exposes `lastPublishedAt` and `state` so a consumer can reject a score older than its freshness window. L1 root anchoring halts when the Oracle enters Paused, so consumers relying on the root see the same fail-closed signal.

6. The Agency for Smart Contracts — Governance Layer

The Agency holds authority and constraints. It never holds capital and never produces evidence. Its authority is bounded by enumeration: an action is permitted only if it appears on the allowlist, passes argument-level bounds, and survives simulation.

6.1 Policy engine

```
interface IPolicyEngine {
    struct Policy {
        address target;        // contract the agent may call
        bytes4  selector;      // function selector allowed on target
        uint256 maxValue;      // max native/stablecoin value per call
        uint64  validAfter;    // time window start
        uint64  validUntil;    // time window end
        bool    enabled;
    }

    function setPolicy(bytes32 policyId, Policy calldata p) external onlyGovernance;
    function evaluate(
        address agent,
        address target,
        bytes4 selector,
        uint256 value,
        bytes calldata data
```

```
    ) external view returns (bool allowed, string memory reason);
}
```

The engine stores policies, not intentions. The agent supplies the intended call; the engine decides whether that call is permitted. An empty allowlist authorizes nobody; an empty blocklist authorizes everybody.

6.2 Allowlist and argument bounds

```
mapping(address => mapping(bytes4 => bool)) public allowedSelector;

function evaluate(...) external view returns (bool, string memory) {
    if (!allowedSelector[target][selector]) {
        return (false, "SELECTOR_NOT_ALLOWED");
    }
    if (value > maxValue[target][selector]) {
        return (false, "VALUE_EXCEEDS_LIMIT");
    }
    return (true, "");
}
```

Selector filtering alone is insufficient when a single function can be called with dangerous arguments. Argument-level policy decodes calldata and enforces bounds:

```
function checkAdvanceArgs(bytes calldata data) internal pure returns (bool) {
    (, uint256 amount) = abi.decode(data[4:], (uint256, uint256));
    return amount <= MAX_ADVANCE;
}
```

6.3 Simulation gate

Before execution, the agent simulates the proposed call against a forked Arbitrum state. Simulation detects calldata errors and produces a state diff compared against expected bounds.

Action	Expected state diff	Rejection condition
Pause	Paused flag = true; no balance changes	Any balance change
Reprice	Spread parameter changes within bounds	Spread outside [min,max]

Notify	Event emitted; no storage change	Unexpected storage write
Advance	Outstanding increases by amount	Amount ≠ intended; recipient unexpected

6.4 Escalation

Criterion	Escalation trigger	Rationale
Value	Call value above threshold	Large transfers warrant human review
Irreversibility	Action cannot be undone	Errors are permanent
Scope	Action affects protocol-wide parameter	Systemic impact
Novelty	Selector not previously exercised	Unfamiliar code path

The multi-sig is a threshold scheme (m-of-n) with signers using hardware wallets or MPC. The policy engine enforces that escalated actions are only executable by the multi-sig address, so the agent cannot bypass escalation even if compromised.

6.5 Key separation

1. **Model context separation:** the model receives observations and outputs proposals; it does not receive secrets.
2. **Signing separation:** the signer is a separate service (HSM, MPC, or hardware wallet) that accepts only policy-approved calldata.
3. **Policy separation:** policy parameters are stored on-chain and changed only through timelocked governance anchored to Ethereum L1.

7. SPOOL Protocol — Credit and Settlement Layer

SPOOL holds and moves capital. It never produces evidence and never decides its own policy. Every SPOOL-mutating call initiated by an agent passes through the Agency first.

7.1 Order NFT

```
interface IOrderNFT is IERC721 {
    struct OrderData {
        bytes32 supplierId;
        bytes32 buyerId;
        uint256 notional;
        uint64  createdAt;
        uint64  deadline;
        OrderState state;
        bytes32 latestAttestationHash;
        uint16  fulfillmentScoreBps; // 0–10000
    }
    enum OrderState { Created, Funded, InProduction, Shipped, Delivered, Settled, Defaulted }

    function mintOrder(OrderData calldata data) external returns (uint256 tokenId);
    function updateState(uint256 tokenId, OrderState newState) external;
    function attachAttestation(uint256 tokenId, bytes32 attestationHash) external;
    function orderData(uint256 tokenId) external view returns (OrderData memory);
}
```

Transition	Precondition	Authorized caller	On-chain effect
Created → Funded	Spool has sufficient liquidity; Agency approved	Spool contract	Advance recorded against order
Funded → InProduction	Supplier confirms start; score above threshold	Agent (allowlisted)	Milestone logged
InProduction → Shipped	Logistics event attested	Oracle callback	Score updated
Shipped → Delivered	Buyer receipt attested	Oracle callback	Repayment scheduled
Any → Defaulted	Score below floor or deadline exceeded	Agent or multi-sig	Spool loss allocation

7.2 Industry spools

```
interface ISpool {
    function deposit(uint256 assets, address receiver) external returns (uint256 shares);
```

```
    function withdraw(uint256 shares, address receiver, address owner) external returns (uint2
    function fundOrder(uint256 orderTokenId, uint256 amount) external onlyRole(FUNDER_ROLE);
    function totalAssets() external view returns (uint256);
    function totalShares() external view returns (uint256);
    function convertToShares(uint256 assets) external view returns (uint256);
    function convertToAssets(uint256 shares) external view returns (uint256);
}
```

Senior and junior tranches are held in separate share classes, with a subordination invariant enforced after every loss allocation:

```
juniorAssets >= seniorAssets * juniorRatioBps / 10000
```

When a default occurs, losses apply to junior capital first. A breach of the invariant triggers the spool into a `Paused` state.

7.3 Advances and repayment

```
advance = notional * advanceRatioBps / 10000 * scoreBps / 10000
```

`advanceRatioBps` is a per-vertical parameter and `scoreBps` is the latest attested Fulfillment Probability Score. The product form means a declining score linearly reduces available liquidity.

1. Buyer payment is received by escrow or a designated settlement address.
2. Principal is returned to the spool.
3. Accrued interest is distributed according to the tranche structure.
4. Any residual is released to the supplier.

The waterfall uses a pull-payment pattern so a failing recipient cannot block settlement.

7.4 Merkle-verified order metadata

Supporting documents—ERP snapshots, inspection reports, logistics records—are batched into Merkle trees. Only the Merkle root is stored on-chain; per-order leaves are published off-chain and verified on demand.

8. Cross-Layer Interfaces

The layers communicate through three narrow interfaces. Each is small enough to audit and to reason about in isolation.

8.1 Oracle → SPOOL: score delivery

```
interface IFulfillmentConsumer {
    function onScoreUpdate(
        bytes32 orderId,
        uint16  scoreBps,
        uint64  timestamp,
        bytes   calldata attestation
    ) external;
}
```

The callback verifies that `timestamp` is within a freshness window and that the attestation recovers to an authorized oracle signer before updating any order state. If verification fails, the callback reverts and the order remains in its prior state.

8.2 Agency → SPOOL: gated execution

Every agent-initiated SPOOL call is routed through the policy engine. The engine validates the (target, selector, value) triple and decodes the arguments before forwarding.

```
function executeGuarded(
    address target,
    bytes calldata data,
    uint256 value
) external onlyAgent returns (bytes memory) {
    bytes4 selector = bytes4(data[:4]);
    (bool allowed, string memory reason) = policyEngine.evaluate(
        msg.sender, target, selector, value, data
    );
    require(allowed, reason);
    (bool ok, bytes memory ret) = target.call{value: value}(data);
    require(ok, "CALL_FAILED");
    return ret;
}
```

8.3 SPOOL → Agency: simulation state

The Agency's simulation stage forks Arbitrum state and executes the proposed SPOOL call against it. The resulting state diff is compared to the expected diff for that action class. SPOOL does not need to expose a dedicated simulation interface; the fork runs against the deployed contract bytecode.

```
function test_simulateAdvance_onArbitrum() public {
    vm.createSelectFork(vm.envString("ARBITRUM_RPC_URL"));
    uint256 orderId = _mintTestOrder();
    uint256 advance = spool.previewAdvance(orderId);
    vm.prank(agent);
    spool.fundOrder(orderId, advance);
    assertEq(spool.advances(orderId), advance);
}
```

9. End-to-End Flow: A Worked Transaction

The following traces a single advance from production event to settlement, naming the layer responsible at each step.

#	Layer	Action	Output
1	Oracle	ERP, vision, and logistics streams are fused; the scorer computes a Fulfillment Probability Score for order #1234.	Score: 8,750 bps
2	Oracle	The score is signed as EIP-712 typed data and published to the Arbitrum registry. A Merkle root is anchored to L1 hourly.	Attested score on Arbitrum
3	Agency	The agent observes the score change and proposes an advance of 50,000 USDC against order #1234.	Proposed calldata
4	Agency	`IPolicyEngine.evaluate` confirms the selector is allowlisted and the amount is within bounds.	Allowed
5	Agency	The agent forks Arbitrum state, executes the proposed call, and confirms the state diff matches the expected advance.	Simulation passed

6	Agency	If the amount exceeded the escalation threshold, the call would route to multi-sig. At 50,000 USDC it is within the agent's autonomous bound.	Execute
7	SPOOL	`fundOrder` verifies the order is in `Created` state, the spool has liquidity, and the attached score is fresh.	Order → `Funded`
8	SPOOL	Stablecoins are transferred to the supplier under a pull-payment pattern.	Supplier receives 50,000 USDC
9	Oracle	Subsequent production events update the score; a decline triggers an Agency reprice proposal.	Updated score
10	Agency	The reprice is proposed, evaluated, simulated, and executed within bounds.	Spread adjusted
11	SPOOL	Buyer payment settles; the waterfall returns principal, distributes interest, and releases residual.	Order → `Settled`

At no point does the Oracle move funds, does the Agency hold capital, or does SPOOL decide its own policy. Each step crosses exactly one boundary, and each boundary has a defined check.

10. Arbitrum L2 Architecture

The network deploys on Arbitrum One as its primary execution environment, with Ethereum L1 as the settlement root.

10.1 Cost model

Arbitrum fees have two components: L2 execution gas and L1 data posting. EIP-4844 blobs reduced L2 posting costs by 10–100× compared to calldata, with post-Dencun typical fees falling to fractions of a cent. Per-order state transitions and frequent oracle attestations are therefore economically viable.

Operation	Typical cost driver	Design
Mint order NFT	Storage writes	Pack state, score, and timestamps into one slot

State transition	Storage write + event	Event-based indexing; registry stores only latest
Oracle attestation	Calldata + signature verify	Batch multiple orders per attestation tx
Advance / repay	Token transfer + accounting	Pull-payment waterfall to avoid push failures

10.2 Finality model

Arbitrum exposes three confirmation levels. **Soft finality** (`latest`) is a sequencer promise, sub-second but reversible. **Parent-chain safe** (`safe`) means the batch is posted and the containing block has reached Ethereum's safe block—roughly 12–15 minutes, revertible only by a deep parent-chain reorg. **Parent-chain final** (`finalized`) means the batch has reached Ethereum's finalized block.

Action	Finality required	Rationale
Pause advance	Soft (sequencer, sub-second)	Risk-reducing; safe under any outcome
Reprice spread	Soft	Reversible; next tick can correct
Advance stablecoins	Parent-chain safe (~12–15 min)	Irreversible value transfer
Repayment settlement	Parent-chain final	Closes credit position

The native L1 withdrawal bridge carries a ~7-day optimistic challenge period. Small repayments settle entirely on Arbitrum; large repayments requiring L1 settlement either disclose the challenge period or route through a fast bridge with explicit counterparty risk disclosure.

10.3 Sequencer failure

Arbitrum's sequencer is currently operated by the Arbitrum Foundation with a decentralization roadmap. Sequencer unavailability is distinct from oracle feed loss and is handled separately.

- **Detection:** block height monitored; a stall beyond 30 seconds (roughly 120 missed blocks) triggers degraded mode.
- **Containment:** no new advances are proposed; outstanding positions remain open; submitted risk-reducing actions remain valid.

- **Recovery:** critical pause actions are force-included via the Arbitrum delayed inbox on L1.
- **Policy:** the policy engine records the maximum acceptable staleness of sequencer confirmation; beyond it, escalation triggers automatically.

10.4 Stylus

Arbitrum Stylus introduces a coequal WebAssembly virtual machine alongside the EVM, allowing contracts written in Rust to compile to WASM and run alongside Solidity contracts sharing the same state tree. Rust-compiled WASM contracts execute faster with significantly lower gas fees: 4–10× cost savings in AMM-style workloads and 100–500× cheaper memory for compute-intensive operations.

Component	Layer	Runtime	Why
Order NFT, spool accounting, access control	SPOOL	Solidity	Storage-heavy, integration-heavy
Advance calculation, score aggregation, invariant checking	SPOOL	Stylus (Rust)	Compute-heavy, memory-intensive
Policy evaluation	Agency	Solidity	Integration with call forwarding
Score verification	Oracle consumers	Solidity	EIP-712 recovery is EVM-native

Cross-contract calls use the shared state tree, so no bridge is needed between the two VMs.

10.5 Timeboost

Arbitrum Timeboost provides an optional express lane that grants a controller priority inclusion for a short window. In the default configuration, express-lane transactions are processed immediately while others face a ~200ms delay. The Agency uses Timeboost for risk-reducing actions that should not queue behind routine traffic.

Action	Timeboost use	Rationale
Pause	Recommended	Risk-reducing; latency matters

Reprice	Recommended	Time-sensitive; spread must track score
Notify	Optional	Event-only; latency less critical
Advance	Not used	Irreversible; waits for parent-chain safe finality

Empirical analysis of Timeboost (April–July 2025, 11.5 million express-lane transactions) found that express-lane control is highly centralized, with two entities winning more than 90% of auctions, and that auction competition declined over time. Timeboost is therefore an optimization for risk-reducing actions only, never a security dependency, and never used for value-moving actions.

10.6 Network allocation

Workload	Network	Reason
SPOOL credit contracts	Arbitrum One	Stylus for compute, Timeboost for agent latency
Consumer-facing UI, supplier portal	Base	Coinbase distribution, cheaper simple transfers
L1 settlement root, governance timelock	Ethereum	Canonical finality

Cross-chain state is reconciled explicitly: a supplier advancing on Arbitrum and receiving on Base implies a bridge hop that is accounted for in repayment timing and risk disclosure.

11. Security Model: Separation of Powers

The network's central security property is that no single component's compromise is sufficient to move capital outside policy.

Component	Holds	If compromised	Blast radius

Fulfillment Oracle	Attestation keys	False scores could be published	Bounded by SPOOL's freshness window and the Agency's simulation; no direct fund movement
Agency (agent)	Proposal capability	Malicious calldata proposed	Rejected by policy engine unless allowlisted and simulated; escalation prevents high-impact execution
Agency (policy)	Allowlists, bounds	Overly permissive policy	Governance timelock on L1 allows reversal; emergency pause available
SPOOL	Capital	Contract bug	Bounded by invariants; spool pause and junior-first loss allocation contain damage
Multi-sig	Escalated authority	Collusion	Requires m-of-n compromise; independent signers with hardware custody

> *Agents propose; contracts enforce; keys and limits stay off-model.*

The same principle applies at every layer. The Oracle proposes a score; SPOOL's freshness check enforces it. The agent proposes a call; the policy engine enforces it. The agent proposes a high-impact action; the multi-sig enforces it.

12. Failure Coupling and Degraded Modes

The layers are coupled by design, and the coupling is directional. Each component's state change propagates to the others in a defined way.

Trigger	SPOOL	Agency	Oracle
Oracle → Paused	Advances halt via freshness window; repayments continue	Escalation triggered; no new advances proposed	—
Oracle → Degraded	Advances reduce proportionally to the	Reprice proposed within bounds	—

	conservative score		
Agency → Paused	Settlement in flight completes; no new agent actions	—	Publication continues
SPOOL → Paused	—	Advance proposals revert and escalate	Publication continues
Sequencer stall	No new state changes	Degraded mode; force-include pause via L1 delayed inbox	L1 root anchoring continues independently
Multi-sig unavailable	Existing positions continue to settle	High-impact actions blocked; risk-reducing actions still execute within agent bounds	Publication continues

Each degraded mode is a controlled reduction in capability rather than a cascade. An Oracle pause stops new credit but does not prevent repayment. An Agency pause stops new agent actions but does not stop settlement. A SPOOL pause stops advances but does not blind the Oracle.

13. Testing and Verification Strategy

Layer	Scope	Tooling
Unit	Individual functions, state transitions	Foundry `forge test`
Integration	Spool + order NFT + oracle callback	Foundry with mock oracle
Fork	Real Arbitrum state, real stablecoins	`vm.createSelectFork(ARBITRUM_RPC_URL)`
Invariant	System-wide properties under fuzzing	Foundry invariant testing
Cross-layer	Oracle → Agency → SPOOL end-to-end	Fork suite with all three contracts deployed

Failure drill	Sequencer stall, Oracle pause, policy rejection	Simulated state + delayed-inbox path

The cross-layer suite verifies that a score change correctly propagates through the Agency's evaluation and produces the expected SPOOL state diff, and that a rejected call escalates rather than retries.

14. Deployment Topology and Observability

Component	Deployment	Anchor
SPOOL contracts	Arbitrum One	Ethereum settlement root
Agency policy engine + multi-sig	Arbitrum One	Ethereum governance timelock
Oracle registry	Arbitrum One	Ethereum Merkle root, hourly
Consumer surfaces	Base	Reads via signed off-chain payloads

Observability covers chain-specific and cross-layer signals: Arbitrum sequencer liveness, L1 finality lag, attestation freshness, policy rejection rate, escalation frequency, and spool utilization per tranche. Alerts fire when any of these crosses a threshold, and the agent runtime treats sequencer staleness as a fail-closed trigger independent of Oracle status.

15. Governance

Policy and contract changes are high-impact and are governed with a timelock anchored to Ethereum L1. The timelock gives users time to exit if a change is hostile, and gives defenders time to react to a compromised governance key.

- **Timelock delay:** sufficient for the protocol's risk profile (48–72 hours).
- **Policy versioning:** each policy set has a version hash; consumers verify which policy was active at execution time.

- **Emergency pause:** a separate, narrower mechanism on Arbitrum that can pause but not unpause or upgrade.
- **L1 anchor:** the timelock and policy version root live on Ethereum, so governance changes inherit L1 finality even though execution happens on Arbitrum.

16. Risk Factors

- **Evidence risk.** The network's credit decisions rest on the Fulfillment Probability Score. Model error or source corruption propagates through every downstream layer.
- **Governance risk.** Allowlists and escalation thresholds determine whether the Agency prevents unbounded automation or merely relocates it.
- **Execution risk.** Smart contracts governing credit and stablecoin movement are exposed to implementation defects and adversarial exploitation.
- **Industrial counterparty risk.** Suppliers and buyers in real supply chains may default, delay, or fail to meet specification.
- **Systemic coupling.** Because the three layers are interdependent, a failure at one layer is not isolated from the others; the failure matrix in Section 12 defines the propagation.
- **L2 risk.** Sequencer centralization, delayed hard finality, and the ~7-day L1 withdrawal challenge period apply to the Arbitrum deployment.
- **Regulatory risk.** Tokenized credit, stablecoin advances, and cross-border industrial finance are subject to evolving regulation across jurisdictions.

Disclaimer. This document is provided for informational purposes and does not constitute financial, legal, or investment advice. Digital assets and DeFi protocols carry significant risk, including total loss of capital. Stablecoin advances and industrial credit instruments may be regulated in some jurisdictions.

© 2025 Ukiyo Network. All rights reserved.

Combined system technical white paper, first edition.

Components: SPOOL Protocol · Fulfillment Oracle · Agency for Smart Contracts.