Iraquin: A new architecture for deterministic outcomes in high performance blockchain v0.1

Hari Hara Prasad

hari@iraquin.com

Abstract

Modern blockchains remain constrained by architectural choices that tightly bind transaction ordering, execution, and correctness to a single consensus critical virtual machine, forcing redundant execution, probabilistic finality, and recovery through rollback and reorganization. These systems scale by replication rather than verification, making them ill suited for decentralized compute, AI workloads, and applications where verified outcomes matter more than repeated execution.

Iraquin is a compute native Layer 1 blockchain designed to provide deterministic ordering and finality, high throughput horizontal scalability, and verifiable execution without a monolithic on-chain virtual machine, re-execution, rollback, or chain reorganization.

Rather than embedding execution directly into the consensus path, Iraquin abstracts execution away from consensus and treats verified outcomes as first class protocol objects. Consensus in iraquin operates at the level of execution intent, the protocol deterministically commits the transaction set and its ordering before execution occurs, ensuring that all nodes agree on what is to be executed independently of how execution proceeds. The protocol separates transaction intake, execution, and verification into distinct responsibilities, enabling parallel execution without state sharding and eliminating redundant work across the network. Invalid execution does not trigger rollback; instead, correctness is enforced through forward correction, where invalid effects are nullified and healed deterministically without reverting committed state. Iraquin natively supports decentralized compute and AI workloads through a consensus backed data availability layer and on-chain marketplace, enabling off chain execution in arbitrary environments while preserving verifiability and accountability. Governance further integrates verifiable AI systems to assist protocol evolution without relinquishing enforceable correctness. Together, these design choices position Iraquin as a blockchain optimized for outcomes, verification, and long lived correctness rather than virtual machines and speculative consensus.

1 Introduction

Blockchains have largely evolved by optimizing a single assumption: that execution itself must be consensus critical. In most existing systems, transaction ordering, execution, and correctness are inseparably bound to a monolithic on-chain virtual machine, requiring every validating node to repeatedly execute the same logic in lockstep. While this design simplifies reasoning, it introduces fundamental inefficiencies, redundant execution, probabilistic finality, and recovery through rollback and reorganization that become increasingly costly as throughput and complexity grow.

These limitations are not incidental; they are architectural. As long as execution remains the unit of consensus, scalability is achieved through replication rather than verification, and correctness becomes dependent on timing, fork choice, and speculative assumptions about network behavior. This model struggles to support decentralized compute, AI inference, and other workloads where-execution may be expensive, heterogeneous, or non deterministic, and where verifying outcomes is more practical than re-executing them everywhere.

Iraquin is built on the premise that these constraints are unnecessary. Rather than treating execution as the basis of agreement, Iraquin treats ordering and verified outcomes as first class protocol concerns. The system is designed to deterministically establish what should be executed and in what order, while allowing how execution occurs to be abstracted away from the consensus critical path. This separation enables parallelism without state sharding, eliminates redundant work, and removes the need for rollback or reorganization as a recovery mechanism.

At a high level, the protocol is organized into two complementary layers with distinct responsibilities. The Prime layer focuses on high throughput transaction intake and optimistic execution proposal, prioritizing speed and parallelism while remaining agnostic to final correctness. The Omega layer operates asynchronously, enforcing correctness, accountability, and settlement through verification, slashing, and correction rather than synchronous agreement. Together, these layers allow the network to progress optimistically while preserving deterministic state evolution.

A key design principle in Iraquin is the decoupling of transaction ordering from execution correctness. The protocol deterministically commits the transaction set of a slot before state is applied, establishing unique ordering without requiring immediate execution agreement. Execution proceeds optimistically and may overlap with ordering finalization, but state transitions are applied only once ordering is unambiguous. Invalid execution does not trigger rollback; instead, the protocol employs forward correction, where incorrect effects are nullified and healed deterministically without reverting committed state.

A defining feature of Iraquin is intent-first consensus. Rather than requiring agreement on execution results, the protocol first establishes a unique, deterministic ordering over the transaction set of each slot. Execution may proceed optimistically and in parallel, but state

transitions are applied only with respect to this committed intent. As a result, execution timing, execution environment, and network observation order do not influence consensus.

Iraquin does not embed a general purpose on-chain virtual machine. Instead, it natively supports off chain execution in arbitrary environments, allowing developers to run compute and AI workloads in languages and systems suited to their tasks. Execution results are committed to the chain through verifiable state diffs and proofs, while a consensus backed data availability layer ensures inputs and outputs remain publicly auditable. An on-chain marketplace coordinates decentralized compute, enabling jobs to be priced, executed, verified, and settled without requiring every node to perform the work.

Beyond execution, Iraquin extends verification principles to governance. Verifiable AI systems are integrated to propose protocol parameters and optimizations, while enforcement remains deterministic and challengeable by the network. This approach allows the protocol to benefit from adaptive intelligence without compromising transparency or control.

By separating ordering, execution, and verification into explicit protocol responsibilities, Iraquin abandons speculative consensus in favor of deterministic outcomes. The protocol explicitly favors safety over liveness: ambiguity is rejected rather than resolved through fork choice, and correctness is enforced through verification and correction instead of probabilistic agreement. This architecture positions Iraquin as a compute native Layer 1 designed for high throughput decentralized systems where correctness, auditability, and long term guarantees matter more than synchronous execution.

2 Design Goals and Non Goals

Iraquin is designed around a set of explicit goals that prioritize deterministic correctness, verified outcomes, and scalable execution over speculative consensus and synchronous execution. These goals reflect deliberate architectural choices rather than incremental optimizations of existing blockchain models. Equally important are the non goals: properties that Iraquin explicitly does not attempt to achieve, as pursuing them would conflict with the system's core guarantees.

2.1 Design Goals

Deterministic Ordering and Finality Iraquin aims to provide deterministic transaction ordering and finality without reliance on probabilistic confirmation or fork choice. Once ordering is finalized, state transitions proceed monotonically, and the committed state is never reverted or reorganized.

No Rollback, No Reorganization The protocol is designed such that invalid execution or adversarial behavior does not trigger rollback or chain reorganization. Instead, incorrect effects are handled through explicit verification and forward correction, preserving a single, irreversible state history.

Elimination of Redundant Execution Iraquin avoids network wide re-execution of transactions by abstracting execution away from the consensus critical path. Nodes verify outcomes rather

than repeatedly executing the same logic, reducing duplicated work and enabling support for compute intensive workloads.

Parallelism Without State Sharding The system enables parallel execution across disjoint transaction sets without partitioning global state into shards. Ordering and verification are handled independently of execution, allowing horizontal scalability without introducing cross shard complexity or fragmented state.

High Throughput with Bounded Latency Iraquin is optimized for high transaction throughput and predictable latency by overlapping transaction intake, execution, and verification. The protocol prioritizes fast ordering finality while deferring correctness enforcement to asynchronous verification.Native Operations over Arbitrary Execution

Iraquin defines state transitions through a native operations rule engine rather than a general purpose virtual machine. Operations are protocol defined, deterministic, and verifiable, enabling bounded state transitions, explicit correctness checks, and forward correction without global re-execution.

Execution Agnostic Compute Model The protocol does not embed a monolithic on-chain virtual machine. Instead, it supports execution in arbitrary off chain environments, enabling developers to use languages, runtimes, and systems appropriate to their workloads while preserving verifiability at the protocol level.

Native Support for Decentralized Compute and AI Iraquin treats decentralized compute and AI inference as first class use cases. A consensus backed data availability layer and on-chain marketplace coordinate job submission, execution, verification, and settlement without requiring global execution by all nodes.

Forward Correction and Accountability Correctness is enforced through verification, economic accountability, and forward correction rather than speculative execution and rollback. Invalid behavior results in slashing and deterministic correction of state effects without violating finality.

Verifiable and Assistive Governance Governance mechanisms integrate verifiable AI systems to assist in proposing protocol parameters and optimizations while preserving transparent, enforceable decision making by the network.

Long Lived Cryptographic Durability The protocol is designed with long term security in mind, accommodating evolving cryptographic assumptions and minimizing dependence on short lived primitives or trusted execution assumptions.

2.2 Non Goals

Synchronous Execution Consensus Iraquin does not attempt to make execution correctness a synchronous consensus requirement. The protocol intentionally separates ordering finality from execution verification, accepting delayed correctness enforcement in exchange for scalability and determinism.

Probabilistic Finality Models The system does not rely on probabilistic confirmation depths or fork based finality. Designs that depend on eventual convergence through reorganization are explicitly avoided.

General Purpose Smart Contract Virtual Machines The protocol does not aim to support arbitrary on-chain programs with unbounded control flow. Application logic that requires expressive computation is executed off chain and committed through verifiable outcomes.

Maximum Liveness Under All Conditions The protocol does not guarantee uninterrupted progress under all adversarial or network conditions. Safety and correctness are prioritized over liveness, and the system may stall rather than accept ambiguous or unverifiable state transitions.

Instant Correctness Guarantees Iraquin does not guarantee immediate correctness of execution at the moment of ordering finality. Correctness is enforced through asynchronous verification, correction, and economic penalties.

Minimal Protocol Complexity Simplicity for its own sake is not a primary goal. The protocol embraces additional complexity where necessary to achieve deterministic finality, verified outcomes, and scalable execution.

3 System Overview

Iraquin is designed as a verification first Layer 1 blockchain that separates transaction intake, execution, ordering, and correctness enforcement into explicit protocol responsibilities. The system is designed around the assumption that execution may be incorrect, adversarial, or heterogeneous, and that correctness should be enforced through verification and correction rather than speculative execution and fork choice. To achieve this, Iraquin organizes the protocol into two primary layers Prime and Omega supported by a native operations rule engine, a consensus backed data availability layer, and an integrated compute marketplace.

At a global level, Iraquin maintains a single logical state and a single canonical ordering of transactions. Parallelism is achieved through dynamic conflict analysis and disjoint execution sets, not through persistent state partitioning or independent shard consensus. State transitions are deterministic and monotonic: once applied, state is never reverted. Invalid execution is handled through forward correction rather than rollback or reorganization.

3.1 Global Architecture: Prime and Omega

The protocol is divided into two complementary layers with asymmetric responsibilities and trust assumptions.

The Prime layer is responsible for high throughput transaction intake, filtering, grouping, and optimistic execution proposals. Prime is designed for speed and scale. It operates under the explicit assumption that execution results may be invalid and that correctness will be enforced later. Prime nodes prioritize availability, parallelism, and throughput, producing execution artifacts that can be independently verified by other components of the system.

The Omega layer is responsible for verification, accountability, and settlement. Omega operates asynchronously with respect to Prime and is the sole authority for applying irreversible state

transitions. Omega validators verify artifacts produced by Prime, enforce protocol rules, apply economic penalties, and issue corrective state updates when necessary. Omega does not participate in execution and does not compete on throughput; its role is to ensure that only valid outcomes affect the canonical state.

This separation allows the network to progress optimistically without coupling execution speed to correctness enforcement, while still preserving deterministic state evolution.

Because consensus operates on intent rather than execution, ambiguity in execution does not introduce ambiguity in state. All valid state transitions and corrections are defined as deterministic functions of the committed transaction intent and protocol rules.

3.2 Roles and Responsibilities

Iraquin defines multiple protocol roles, each with a constrained and explicit responsibility.

Ingress Nodes serve as the interface between users and the network. They accept user transactions and compute job submissions, perform basic validation, and forward them into the Prime layer. Ingress nodes do not execute transactions or participate in consensus.

Miners operate within the Prime layer and are responsible for transaction verification, filtering, and aggregation. Miners validate signatures, check basic transaction correctness, and produce intermediate transaction artifacts that can be consumed by colliders. Miners do not execute transactions or update state.

Colliders are responsible for execution proposals. They consume verified transaction artifacts from miners, identify disjoint transaction sets based on access patterns, and execute transactions optimistically. Colliders produce execution outputs, transaction set commitments, and candidate state diffs, but they do not finalize state. Colliders may produce invalid execution results; correctness is not enforced at this stage.

Omega Validators verify artifacts produced by Prime and colliders. They reconstruct transaction sets, validate commitments, re verify execution outcomes, and determine whether state transitions are valid. Omega validators enforce accountability through slashing and forward correction and are responsible for final reward settlement.

Compute Workers execute off chain compute and AI workloads submitted through the protocol. They operate in arbitrary environments and are required to provide verifiable outputs. Compute workers are economically accountable for correctness and timeliness.

Archival and DA Nodes maintain and serve data committed to the protocol’s data availability layer, ensuring that all referenced inputs and outputs remain publicly auditable for verification.

3.3 Native Operations and State Transitions

Iraquin does not embed a general purpose on-chain virtual machine. Instead, all state transitions are expressed through a native operations rule engine. Native operations are protocol defined, deterministic, and bounded. They define how verified outcomes such as balance changes, compute settlements, governance decisions, or correction actions modify the global state.

Execution logic that requires expressive computation is performed off chain. The chain's responsibility is not to execute arbitrary programs, but to verify that proposed outcomes satisfy the rules of the protocol and the semantics of the native operations. This design eliminates unbounded execution from the consensus critical path and avoids redundant re-execution across the network.

3.4 Intent Level Ordering and State Application

Transaction ordering in Iraquin is finalized independently of execution. The protocol deterministically commits the transaction set for a given slot before any state transition is applied. This establishes a single canonical ordering without relying on probabilistic confirmation or fork choice.

Execution may proceed optimistically and overlap with ordering finalization, but state transitions are applied only after ordering is unambiguous. If execution is later found to be invalid, the protocol applies forward correction through explicit state updates rather than reverting prior state. This ensures that state evolution remains monotonic and deterministic.

3.5 Transaction Lifecycle

A user transaction enters the system through an ingress node and is forwarded into the Prime layer. Miners verify and aggregate transactions, producing intermediate artifacts that represent validated transaction candidates. Colliders consume these artifacts, group transactions into disjoint execution sets, and execute them optimistically, producing execution outputs and state diffs.

The transaction set is committed for ordering independently of execution. Once ordering is finalized, Omega validators verify execution results against the committed transaction set. If valid, the corresponding state transitions are applied. If invalid, the protocol enforces penalties and issues corrective state updates without rolling back prior state.

At no point does the network rely on speculative fork resolution or reorganization to recover from invalid execution.

3.6 Compute and AI Job Lifecycle

Compute and AI jobs follow a similar lifecycle but are-executed entirely off chain. A job submission includes inputs, execution requirements, and verification criteria and is published through the data availability layer. Compute workers execute the job in arbitrary environments and submit verifiable outputs.

Prime ingests and orders compute job commitments, while Omega verifies outputs and applies settlement operations through native operations.Invalid or incomplete execution results in economic penalties and job reassignment. The protocol does not require global re-execution of compute tasks, allowing it to support compute intensive workloads efficiently.

3.7 Governance Lifecycle

Governance actions are proposed as protocol native operations. Verifiable AI systems may assist in proposing parameters or optimizations, but governance outcomes are enforced only through deterministic verification and native operations. Governance decisions follow the same ordering, verification, and settlement model as other state transitions, ensuring transparency and accountability.

4 Prime Protocol: Optimistic Block Production

The Prime protocol is responsible for transaction intake, cryptographic verification, execution proposal, and optimistic block production in Iraquin. Prime is designed to maximize throughput and parallelism while remaining explicitly non-final. It operates under the assumption that execution may be incorrect and adversarial, and that correctness will be enforced asynchronously by the Omega layer. As a result, Prime never applies irreversible state transitions and never resolves ambiguity through fork choice.

Prime produces execution proposals, not finalized blocks. These proposals may be invalid, conflicting, or incomplete, but they are always verifiable and economically accountable. This design allows Prime to progress under network asynchrony and adversarial behavior without introducing rollback or reorganization.

4.1 Transaction Ingress and Network Interface

Ingress nodes form the boundary between the external world and the Prime layer. Their sole responsibility is to accept transactions and compute job submissions from users and reliably stream them into the Prime pipeline.

Ingress nodes perform basic syntactic checks and signature validation sufficient to ensure that malformed transactions are not propagated. They do not execute transactions, participate in ordering, validate state, or influence consensus outcomes. Ingress nodes are intentionally stateless and authority free.

Once accepted, transactions are sharded and streamed to miner verification groups based on deterministic routing rules. Ingress nodes do not interpret transaction semantics; they simply route transactions to the appropriate verification paths.

4.2 Miner Verification and VDF Based Competition

Miners operate within the Prime layer and are responsible for cryptographic verification and transaction admissibility. Their function is to ensure that only well formed, correctly signed, and protocol compliant transactions enter the execution pipeline. Miners do not execute transactions and do not verify or update state.

Upon verification, miners emit verified transaction artifacts that represent eligible candidates for execution. These artifacts consist of transaction identifiers and txn information since miners do not care about the state of a txn.

Miners operate within verification shards and compete to advance verified transaction sets using a verifiable delay function (VDF) based selection mechanism. This competition introduces fairness and unpredictability into transaction advancement while remaining fully verifiable and free from synchronous coordination requirements.
Miner participation is economically accountable. Incorrect verification, equivocation, censorship, or protocol violations are detected and penalized through slashing and score degradation. Miner assignments are time bounded, and successful inclusion of candidates for execution increases future selection difficulty through adaptive delay adjustment, preventing persistent dominance of the verification pipeline by a fixed set of participants.Ingress Stream.

4.3 Collider Execution and State Validation

Colliders are responsible for state validation and execution proposals. They consume verified transaction artifacts produced by miners and determine which transactions can be executed together based on access patterns and conflict analysis.
Colliders operate over the global state and are aware of state semantics. Their role is not merely execution, but validation of whether proposed transactions can produce a consistent state transition.
To achieve high throughput, colliders identify disjoint transaction sets, groups of transactions whose read and write sets do not overlap and execute these sets in parallel. This parallelism is dynamic and recomputed per slot; no transaction or account is permanently bound to an execution group.
Execution performed by colliders is explicitly optimistic. Colliders may produce incorrect execution results, either due to bugs or adversarial behavior. Such errors do not compromise protocol safety and are detected later by Omega.
After execution, colliders assemble provisional blocks containing:

- ordered transaction sets
- execution outputs
- candidate state diffs

4.4 Parallelism Without State Sharding

Parallelism in Prime is achieved through conflict aware-execution over a single global state, not through persistent state partitioning. The protocol maintains a single canonical ordering and a single logical state.
Execution groups are formed dynamically based on observed access patterns and are discarded after use. No group has independent ordering authority, and no cross group coordination is required.

Because execution is decoupled from finality, Prime can aggressively parallelize execution without risking inconsistent state. Invalid execution results are handled later without rollback.

4.5 Prime Elections and Economic Accountability

Participation in Prime as a miner or collider requires explicit economic commitment. Elections determine active miner and collider sets for each epoch and are stake weighted, deterministic, and time bounded.Ingress nodes are responsible for routing and streaming transactions across miner shards while enforcing rate control, acting as a buffering and throttling layer that protects the miner pool from excess transaction floods and unbounded load injection.

Miners place stake at risk for incorrect verification, equivocation, or censorship. Colliders place higher stake at risk due to their influence on execution proposals and state validation.

A scoring system tracks long term participant behavior across epochs. Reliable participants are favored in future elections, while unreliable or adversarial participants are throttled or excluded. Scoring provides persistent memory without requiring immediate hard removal.

A burning mechanism enforces economic cost on participation and proposal activity, ensuring that throughput reflects real demand and that abuse is economically irrational.

Throttling limits the influence of low scoring participants in real time, reducing verification capacity for miners or execution bandwidth for colliders without halting the system.

Together, elections, burning, scoring, and throttling ensure that Prime remains optimistic but economically accountable.

4.6 Why Prime Never Reorganizes

Prime never triggers rollback or reorganization due to two architectural constraints.

First, Prime never applies irreversible state transitions. All execution outcomes are provisional and subject to later verification. Because the state is not finalized in Prime, there is nothing to revert.

Second, Prime does not resolve ambiguity through fork choice. If multiple execution proposals exist, Prime does not attempt to select a “best” block. Ambiguity is preserved and carried forward as verifiable artifacts to be resolved by Omega.

As a result, Prime can tolerate adversarial execution, conflicting proposals, and network asynchrony without compromising deterministic state evolution.

4.7 Relationship to Omega

Prime and Omega are deliberately asymmetric. Prime optimizes for throughput, availability, and parallelism. Omega optimizes for correctness, accountability, and settlement.

Prime does not wait for Omega to verify execution results before continuing operation. This overlap allows the protocol to pipeline execution and verification, achieving high throughput without sacrificing determinism.

5 Omega Protocol: Verification and Forward Correction

The Omega protocol is responsible for correctness, accountability, and irreversible state settlement in Iraquin. Unlike Prime, which optimizes for throughput and optimistic progress, Omega is correctness first by design. It operates asynchronously with respect to Prime and is the sole authority for applying final state transitions, settling rewards, and enforcing penalties.

Omega assumes that any artifact produced by Prime including execution results, block proposals, or state diffs may be incorrect or adversarial. Rather than preventing incorrect execution from occurring, Omega is designed to detect, prove, and deterministically correct incorrect outcomes without rolling back committed state. This separation allows the system to progress optimistically while preserving long term correctness and auditability.

5.1 Scope of Verification

Omega validators do not re-execute the entire Prime pipeline. Instead, they selectively re-check verifiable artifacts produced by Prime and colliders. Verification is structured to minimize redundant work while ensuring that any incorrect influence on state is detectable and provable.

Specifically, Omega validators re check:

- the integrity and completeness of ordered transaction sets
- the validity of cryptographic commitments and proofs
- adherence to protocol rules and invariants
- the consistency between declared execution outcomes and permitted state transitions

Omega validators are not required to reproduce the exact execution environment used by colliders. Verification is outcome oriented: validators determine whether a proposed state transition could have resulted from a valid execution under the protocol rules, not whether it was executed in a particular way.

This distinction is critical. Omega verifies correctness of effects, not correctness of execution process.

5.2 What Constitutes Incorrect Execution

In Iraquin, incorrect execution is defined narrowly and precisely. An execution is considered incorrect if it produces effects that violate protocol defined validity conditions, regardless of intent or cause.

Examples of incorrect execution include:

- state transitions that violate balance, ownership, or supply invariants
- execution outputs inconsistent with the committed transaction set
- invalid ordering assumptions or missing transactions
- malformed or unverifiable execution artifacts
- execution results that cannot be justified by any valid application of native operations

Importantly, incorrect execution does not imply malicious behavior. Errors may arise from software bugs, inconsistent environments, partial data, or adversarial inputs. Omega treats all incorrect outcomes uniformly: as invalid state effects to be corrected.

5.3 Forward Correction Model

Iraquin does not recover from incorrect execution through rollback or reorganization. Instead, Omega enforces correctness through forward correction.

When Omega validators detect an incorrect execution outcome, they construct a correction transaction. A correction transaction is a protocol native operation that explicitly nullifies the invalid effects and restores the state to what it would have been had the incorrect execution not occurred.

Correction transactions:

- are deterministic and verifiable
- reference the specific invalid effects being corrected
- do not revert prior committed state
- are applied as forward state transitions

This model ensures that state evolution remains monotonic. Once a state transition is committed, it is never undone; instead, its invalid effects are explicitly compensated by subsequent corrections.

Forward correction makes incorrect execution observable and auditable, rather than hidden by reversion. The chain records both the error and its resolution.

5.4 Determinism of Correction

Correction in Omega is deterministic by construction. Given the same ordered transaction set and the same invalid execution artifact, all honest Omega validators will derive the same correction outcome.

This determinism arises from three constraints:

1. Explicit invalidity criteriaInvalid execution is defined by protocol rules and native operation semantics, not by subjective judgment.
2. Canonical correction rulesFor each class of invalid effect, the protocol defines how correction must be applied.
3. Order independent applicationCorrection transactions reference specific invalid effects rather than relying on global state rewinds.

As a result, correction does not introduce ambiguity or secondary forks. Once a correction is agreed upon, it becomes part of the canonical state history and cannot itself be reversed.

5.5 Bounded Dispute Window

Omega enforces a bounded dispute window during which execution artifacts may be challenged and corrected. This window exists to accommodate asynchronous verification, delayed data availability, and off chain execution environments.

Within the dispute window:

- execution outcomes remain provisional
- challenges and correction proofs may be submitted
- rewards are not yet finalized

After the dispute window expires:

- the corresponding state transitions become irreversible
- rewards are settled
- further challenges are rejected

The dispute window is bounded and protocol defined. It is long enough to allow meaningful verification but short enough to ensure predictable finality. Crucially, the existence of a dispute window does not imply probabilistic finality or reorganization. State is never rolled back; only forward corrections are permitted within the window.

5.6 Economic Accountability and Slashing

Omega enforces economic accountability by penalizing participants whose actions result in incorrect execution or protocol violations.

- Colliders that produce invalid execution artifacts are slashed.
- Miners that contribute incorrect verification or equivocation are penalized.
- Rewards are finalized only after successful verification and dispute resolution.

Slashing and reward settlement are handled exclusively by Omega, ensuring that economic consequences are applied consistently and deterministically.

Importantly, Omega does not require immediate attribution of fault to correct state. Correction is applied first; attribution and penalties follow. This separation prevents disputes over blame from delaying correctness.

5.7 Why Omega Never Reorganizes

Omega never reorganizes state because it never relies on speculative execution or fork choice. Ordering is already finalized before state application, and incorrect execution is handled through explicit correction rather than reversion.

Even in the presence of adversarial behavior, delayed data, or conflicting execution artifacts, Omega resolves correctness by applying deterministic corrections forward in time. There is no concept of “undoing” state or selecting an alternative history.

As a result, once Omega commits a state transition, the chain’s history remains linear, monotonic, and auditable.

6 Forward Correction Semantics

Forward correction is the mechanism by which Iraquin enforces correctness without rollback or reorganization. Rather than reverting previously committed state, the protocol applies explicit corrective state transitions that nullify invalid effects while preserving the linearity and determinism of the state history. This section formalizes the intuition behind forward correction and explains why correction remains bounded, localized, and deterministic.

6.1 State Transitions and Diffs

Let the global state be represented as a mapping:

$$S:K \rightarrow V$$

where K denotes state keys (e.g., accounts, objects, resources) and V denotes their associated values.

A state transition is expressed not as a full state rewrite, but as a state diff:

$$\Delta = \{(k, v_{old}, v_{new}) \mid k \in K\}$$

A diff Δ Delta Δ captures the intended effects of executing an ordered transaction set. Applying a diff to a state produces a new state:

$$S' = S \oplus \Delta$$

where $\oplus$ denotes deterministic application of the diff.

Crucially, Prime produces candidate diffs, while Omega determines which diffs are valid and which require correction.

6.2 Validity of a Diff

A state diff Δ is considered valid if and only if:

1. It corresponds to an ordered transaction set already finalized by the protocol.
2. It satisfies all protocol defined invariants.
3. It can be justified by a valid application of native operations.

If any of these conditions fail, the diff is considered invalid, regardless of whether execution succeeded or failed in practice.

Incorrect execution, therefore, is defined as the production of an invalid diff.

6.3 Correction as a First Class State Transition

When Omega detects an invalid diff Δi, it does not revert state. Instead, it constructs a correction diff, Δic such that:

$$(S \oplus \Delta i) \oplus \Delta ic = S$$

with respect to the keys affected by Δi.

Intuitively, Δic explicitly compensates for the invalid effects introduced by Δi. The correction is itself a protocol native operation and is applied as a forward state transition.

This approach ensures that:

- state evolution remains monotonic
- history remains linear and auditable
- invalid effects are visible rather than erased

6.4 Dependency Graph of State Diffs

To reason about the scope of correction, Omega models state transitions using a dependency graph.

Each diff Δi is associated with:

- a write set $Wi \subseteq K$
- a read set $Ri \subseteq K$

A directed edge exists from Δi to Δj if:

$$Wi \cap Rj = \varnothing$$

This indicates that Δj depends on the state written by Δi.

The dependency graph is acyclic with respect to ordering, as transaction ordering is finalized prior to state application.

6.5 Bounded Scope of Correction

When a diff Δi is found to be invalid, correction applies only to:

1. The keys in Wi
2. Any downstream diffs whose correctness depends on those keys

Formally, correction propagates only along reachable paths in the dependency graph originating from Δi.

This ensures that correction is localized:

- Unrelated diffs with disjoint read/write sets are unaffected.
- State unrelated to the invalid execution remains intact.
- Correction does not cascade arbitrarily across the state.

Because dependency relationships are explicit and finite, the correction scope is bounded.

6.6 Why Correction Does Not Cascade Uncontrollably

Forward correction does not lead to infinite or uncontrolled cascades for three reasons.

First, ordering is finalized before state application. This guarantees that the dependency graph is fixed and does not change during correction.

Second, native operations are bounded and deterministic. Each correction diff has a well defined and finite write set.

Third, corrections reference specific invalid effects rather than re-evaluating entire-execution histories. Once a correction is applied, it resolves the invalid dependency and does not introduce new ambiguity.

As a result, correction converges after a finite number of steps.

6.7 Preservation of Unrelated State

A core property of forward correction is non interference.

For any state key k∉Wik , where Wi is the write set of an invalid diff Δi:

$$S'(k)=S(k)$$

That is, correction does not affect an unrelated state.

This property is critical for system composability. Independent applications and compute jobs are insulated from failures elsewhere in the system, even when execution is adversarial or incorrect.

6.8 Determinism of Correction

Given:

- the same ordered transaction set
- the same invalid diff Δi
- the same protocol rules

all honest Omega validators will derive the same correction diff Δic.

There is no discretion or fork choice involved in correction. Correction is a function of protocol rules and observed invalidity, not validator preference or timing.

This determinism ensures that forward correction preserves a single canonical state history.

7 Execution Model and Verifiable Computation

Iraquin adopts an execution model that explicitly decouples computation from consensus. Rather than embedding a general purpose virtual machine into the protocol, Iraquin treats execution as an external, verifiable process whose outcomes are committed to the chain through deterministic state transitions. This design choice is foundational: it removes unbounded computation from the consensus critical path and allows the protocol to support heterogeneous and compute intensive workloads without sacrificing determinism or correctness.

7.1 No Virtual Machine by Design

Iraquin does not provide a monolithic on-chain virtual machine. This is not a limitation, but a deliberate architectural decision.

Virtual machines tightly couple execution semantics, resource accounting, and correctness to consensus, forcing all validating nodes to re-execute arbitrary user defined logic. This model leads to redundant computation, bounded expressiveness through gas, and inherent scalability limits.

Instead, Iraquin defines state transitions through a native operations rule engine, where the chain verifies what changed, not how it was computed. Application logic executes off chain, and only verifiable outcomes are eligible to affect state. This eliminates the need for global re-execution while preserving protocol level correctness.

Developers do not deploy programs to the chain; they commit results to the chain, accompanied by verifiable evidence.

7.2 Pluggable Execution Environments

Execution in iraquin occurs in pluggable environments external to consensus. These environments may differ in language, runtime, hardware, or trust assumptions, but all share a common interface: they produce outcomes that can be verified against protocol rules.

Examples of execution environments include:

- trusted execution environments (TEEs)
- cryptographic proof systems
- deterministic sandboxes
- specialized accelerators

The protocol does not privilege or hard code any specific execution model. Instead, it defines verification boundaries that allow multiple execution environments to coexist and evolve independently. New environments can be introduced without protocol upgrades, provided they produce verifiable outputs compatible with native operations.

This abstraction ensures that Iraquin remains adaptable to advances in computation and verification without ossifying around a single execution paradigm.

7.3 Verifiable Outcomes and Proof Classes

Iraquin verifies execution outcomes through proofs, broadly defined as any verifiable evidence that an outcome satisfies protocol defined correctness conditions.

Rather than prescribing a single proof system, the protocol recognizes classes of proofs, such as:

- cryptographic validity proofs
- execution attestations
- consistency commitments
- fraud or challenge proofs

The role of the protocol is not to understand the internal structure of these proofs, but to enforce that:

- the proof corresponds to the claimed outcome
- the outcome conforms to native operation semantics
- The proof is publicly verifiable within the dispute window

By operating at the level of proof classes rather than specific constructions, Iraquin avoids coupling its execution model to any single cryptographic technique.

7.4 Generalization to AI and Compute Workloads

The execution model of Iraquin naturally generalizes to AI inference, data processing, and other compute heavy workloads.

AI systems are characterized by:

- high computational cost
- heterogeneous hardware
- non trivial execution environments
- outcomes that are more practical to verify than to reproduce

Embedding such workloads in an on-chain VM is impractical. Iraquin instead treats AI execution as an off chain process whose results predictions, inferences, aggregations are committed to the chain as verifiable outcomes.

The protocol does not require validators to re-run models or inspect internal parameters. Validators verify that:

- the claimed output corresponds to a valid execution commitment
- the proof meets the verification requirements
- the resulting state transition is permitted

This allows decentralized AI and compute systems to be coordinated and settled on-chain without imposing global execution costs.

7.5 on-chain Compute Marketplace

Iraquin includes a native on-chain marketplace for decentralized computation. The marketplace coordinates job submission, execution, verification, and settlement while remaining agnostic to the execution environment itself.

A compute job specifies:

- inputs and availability requirements
- expected outputs
- verification criteria
- economic terms

Execution is performed by external workers, who submit results and associated proofs. Prime ingests and orders these commitments, while Omega verifies correctness and settles outcomes through native operations.

Invalid or incomplete execution results are rejected and penalized through the same forward correction and slashing mechanisms used elsewhere in the protocol. Correct results are settled deterministically without requiring re-execution by the network.

7.6 Execution and Consensus Separation

At no point does execution participate directly in consensus. Ordering, finality, and correctness are enforced independently of how or where computation occurs.

This separation allows Iraquin to:

- support arbitrary computation without bloating consensus
- evolve execution environments independently of protocol upgrades
- preserve deterministic state evolution even under adversarial execution

Execution may fail, diverge, or be adversarial; consensus remains stable.

8 Economic Model and Incentives

Iraquin's economic model is designed to align participant incentives with protocol correctness, availability, and sustained contribution rather than short term execution advantage. Rewards are earned through coordinated participation, while penalties are applied for individual, provable faults. Economic settlement is delayed and conditional, ensuring that participants cannot profit from incorrect or adversarial behavior.

The protocol explicitly separates contribution from correctness enforcement: participants are rewarded for sustained, correct participation over time, and slashed only when individual responsibility for a violation is provable.

8.1 Economically Incentivized Roles

The protocol defines the following economically active roles:

- Ingress Nodes, which stream and rate limit transaction traffic into Prime.
- Miners, which perform cryptographic verification and advance transaction sets.
- Colliders, which validate state semantics and propose execution outcomes.
- Omega Validators, which verify correctness, apply correction, and finalize settlement.
- Data Availability (DA) Nodes, which store and serve protocol data until expiration.
- Compute Workers, which execute off chain compute and AI workloads.

Each role is rewarded for availability and correctness within its defined responsibility boundary.

8.2 Coordinated Rewards vs Individual Accountability

Iraquin distinguishes between collective contribution and individual fault.
Rewards for ingress nodes, miners, and colliders are earned through coordinated effort within the Prime pipeline. As long as the collective outcome is verified as correct, participants are eligible for rewards.
Slashing and penalties are applied individually and only when a participant's specific misbehavior is provable (e.g., equivocation, invalid verification, censorship, or malformed execution artifacts).
This separation prevents collective punishment while preserving strong accountability.

8.3 Reward Timing and Settlement

Rewards in iraquin are delayed and conditional.
Ingress Nodes earn rewards for reliably streaming transactions across miner shards, enforcing rate limits, and maintaining availability. Rewards accrue based on sustained participation and are finalized only after downstream verification confirms that forwarded traffic contributed to valid outcomes.
Miners earn rewards for correctly verifying transactions and advancing eligible transaction sets. Rewards are accrued per contribution but finalized only after Omega confirms that no invalid verification or equivocation occurred.
Colliders earn rewards for proposing execution outcomes that are ultimately verified as correct. Rewards are withheld until the dispute window expires without correction.
Omega Validators are rewarded per epoch, not per slot. Validators must remain active for a required fraction of the epoch to be eligible for rewards. This discourages opportunistic participation and incentivizes sustained availability.
DA Nodes earn rewards for storing and serving data committed to the protocol until a specified time to live (TTL). Rewards are conditional on availability proofs and successful data retrieval during the dispute window.
Compute Workers earn rewards only when submitted results are verified and settled. Incorrect, incomplete, or unverifiable results earn no compensation.
No rewards are finalized at the moment of execution or proposal.

8.4 Slashing and Penalties

Slashing is applied deterministically when a protocol violation is proven.
Participants may be penalized for:

- equivocation or inconsistent artifacts,
- incorrect cryptographic verification,
- invalid execution proposals,
- censorship or availability violations,
- failure to meet declared service obligations (e.g., DA storage until TTL).

Slashing affects only the responsible participant and does not retroactively penalize other contributors. Penalties may include stake forfeiture, burning, reward denial, and exclusion from future participation.

8.5 Epoch Based Validator Incentives

Omega validators are incentivized for sustained correctness enforcement, not short term activity.

- Rewards are calculated and distributed at the end of each epoch.
- Validators must meet a minimum activity threshold over the epoch to qualify.
- Validators that are intermittently active, unavailable, or selectively participating for profit are penalized through reward reduction or exclusion.

This design aligns validator incentives with long term protocol health rather than per slot opportunism.

8.6 Burning

In Iraquin, burning functions as a protocol level commitment mechanism, not as a transactional fee or congestion control primitive. Participants elected into the Prime protocol for an epoch miners, colliders, and ingress operators—are required to burn a predetermined amount of value as a condition of participation. This burn represents irreversible skin in the game and signals credible commitment to correct and sustained behavior for the duration of the epoch.

Burning is incurred upon election, not per action. Once burned, the value cannot be recovered regardless of subsequent behavior. This distinguishes burning from slashing, which is conditional and punitive. Burning establishes a non refundable cost of participation, while slashing enforces penalties for provable violations.

By requiring irreversible commitment upfront, the protocol ensures that only participants with sufficient economic conviction and long term alignment enter Prime roles. This reduces incentives for short lived, opportunistic, or griefing behavior that could otherwise exploit optimistic execution.

Burning does not substitute for accountability. Participants that misbehave after election are still subject to individual slashing, score degradation, and exclusion. Burning merely establishes a baseline cost of entry; correctness is enforced through verification, correction, and penalties.

This model ensures that participation in Prime is both permissionless and costly, allowing open access while preventing costless abuse.

8.7 Scoring, Throttling, and Long Term Incentives

A persistent scoring system tracks participant behavior across epochs.

- High scoring participants receive higher effective throughput and election preference.

- Low scoring participants are throttled, deprioritized, or excluded.
- Scoring decays gradually, allowing recovery from isolated faults while penalizing sustained misbehavior.

Throttling reduces the real time impact of unreliable participants without halting protocol progress.

8.8 Why Honest Behavior Dominates

Under Iraquin's incentive structure:

- Incorrect execution cannot yield profit because rewards are delayed until verification.
- Equivocation produces deterministic loss through slashing.
- Censorship reduces future earning capacity via scoring and throttling.
- Partial participation is discouraged by epoch level reward qualification.

Conversely, honest participants accumulate rewards, improve long term selection probability, and face bounded risk. Under rational assumptions, correctness aligned behavior strictly dominates adversarial strategies.

8.9 Separation from Monetary Policy

This section intentionally omits token supply, emission schedules, and valuation dynamics. The mechanisms described here are sufficient to reason about incentive alignment and protocol safety independently of monetary policy, which is specified separately.

9 Threat Model and Adversarial Scenarios

Iraquin is designed under the assumption that participants may behave arbitrarily, execution environments may be faulty or adversarial, and network communication may be asynchronous and unreliable. The protocol explicitly avoids relying on synchronous agreement, trusted execution, or universal re-execution for safety. Instead, it enforces correctness through verification, accountability, and deterministic correction.

This section outlines the adversarial capabilities assumed by the protocol, the guarantees Iraquin provides under those assumptions, and representative adversarial scenarios illustrating how the system responds.

9.1 Adversarial Model

The threat model distinguishes between execution faults, Byzantine participants, and network level adversaries.

Execution Adversary

Execution performed by colliders or external compute workers is assumed to be fully adversarial. Execution environments may:

- produce incorrect outputs
- omit transactions
- fabricate results
- diverge across nodes
- behave non deterministically

The protocol assumes no correctness guarantees from execution itself. Safety is derived solely from post execution verification.

Byzantine Protocol Participants

Miners, colliders, and validators may behave arbitrarily, including:

- Equivocation
- Censorship
- Producing malformed artifacts
- attempting to influence ordering or execution outcomes

The protocol assumes that a bounded fraction of Omega validators may be Byzantine, but that a quorum of honest validators exists to enforce verification and correction.

Network Adversary

The network is assumed to be asynchronous. Messages may be delayed, reordered, duplicated, or temporarily partitioned. No global clock or bounded message delay is assumed for safety. Liveness may degrade under prolonged partitions, but safety must be preserved.

9.2 Trust Assumptions

Iraquin makes the following explicit assumptions:

- Cryptographic primitives are secure.
- Data committed to the availability layer can be retrieved within the dispute window.
- A quorum of Omega validators eventually observes the same execution artifacts.
- Economic penalties (slashing, burning, reward denial) are enforceable once correctness is established.
- The protocol does not assume:
- honest execution,
- honest leaders,
- synchronous communication,
- immediate fault detection.

9.3 Safety Guarantees

Under the above assumptions, Iraquin guarantees:

- Deterministic ordering: Once ordering is finalized, it is never reverted.
- No rollback or reorganization: State evolution is monotonic.
- Correctness via correction: Invalid execution effects are nullified through forward correction.
- Non interference: Unrelated state is unaffected by incorrect execution elsewhere.
- Economic accountability: Adversarial behavior is penalized once detected.

Liveness is best effort and may degrade under extreme network or adversarial conditions, but safety is always preserved.

9.4 Representative Adversarial Scenarios

Scenario 1: Malicious Collider Produces Incorrect Execution
A collider produces an execution result that violates protocol invariants or native operation semantics.
Outcome:

- The invalid execution artifact is detected by Omega validators.
- A deterministic correction transaction is constructed.
- Invalid effects are nullified without rollback.
- The collider is penalized economically.

State remains consistent, and unrelated transactions are unaffected.

Scenario 2: Malicious Collider Attempts Equivocation
A collider produces conflicting execution artifacts for the same ordered transaction set.
Outcome:

- Conflicting artifacts are detectable through commitments and verification.
- Omega rejects inconsistent execution and applies correction if necessary.
- The collider is slashed and excluded from future participation.

No fork choice or reorganization occurs, as execution artifacts are provisional until verified.

Scenario 3: Miner Censorship or Verification Faults
A miner selectively filters transactions or verifies invalid transactions.
Outcome:

- Verification faults are detectable through downstream inconsistency.
- Miner scoring is degraded, and economic penalties apply.
- Other miners continue verification independently.

Because miners do not finalize ordering or state, censorship does not compromise safety.

Scenario 4: Network Partition During Execution
The network temporarily partitions, and different subsets observe different execution artifacts.
Outcome:

- Ordering remains deterministic and finalized independently of execution.
- Execution artifacts remain provisional.
- Upon network recovery, Omega observes all artifacts and applies verification and correction deterministically.

Temporary divergence in observed execution does not result in permanent state divergence.

Scenario 5: Invalid Compute or AI Result Submission
A compute worker submits an incorrect or fabricated result.
Outcome:

- The result fails verification within the dispute window.
- Settlement is rejected, and penalties are applied.
- The job may be reassigned without affecting unrelated state.

Validators do not re-execute the computation; they verify outcome correctness.

9.5 Bounded Dispute and Resolution

All execution artifacts are subject to a bounded dispute window during which they may be challenged. This window accommodates delayed data availability and asynchronous verification.
Within the dispute window:

- Execution effects are provisional.
- Challenges and correction proofs are admissible.

After the window expires:

- State transitions become irreversible.
- Rewards are finalized.
- Further challenges are rejected.

This mechanism provides predictable finality without probabilistic rollback.

9.6 Failure Modes and Explicit Non Goals

Iraquin explicitly does not guarantee:

- uninterrupted liveness under prolonged network partitions,
- instant correctness of execution outcomes,
- resistance to adversaries controlling a majority of verification power.

In such cases, the protocol prefers halting or delayed settlement over accepting ambiguous or unverifiable state transitions.

9.7 Comparison to Fork Based Systems

Traditional blockchains recover from adversarial behavior through fork choice and rollback. Iraquin rejects this approach. Instead of resolving ambiguity by selecting an alternative history, Iraquin treats ambiguity as invalidity and resolves it through verification and correction.

This design shifts complexity from speculative execution to explicit correctness enforcement, enabling deterministic safety at the cost of delayed settlement.